



e-ISSN: 3144-2196

SPECIAL ISSUE
VOLUME 1 | ISSUE 2 | 2026

AN INTERNATIONAL
PEER-REVIEWED,
OPEN ACCESS JOURNAL

FROM R TO INSIGHT

A Handbook of Visualizations
in Statistical and Data Science



VISUALIZE
EXPLORE
ANALYZE
INTERPRET
DECIDE

Peter Majoge



Data
Driven Solutions



Evidence
Based Practice



Knowledge
for Society



Sustainable
Development



NEXUS ACADEMIC PRESS

Publishing Knowledge for a Better Tomorrow

NEXUS INTERNATIONAL JOURNAL OF APPLIED SCIENCE, MEDICINE AND SOCIAL SCIENCES (NIJAMSS)

www.nijamss.org

From R to Insight

A Handbook of Visualizations in Statistical and Data Science

Peter Majoge

SPECIAL-ISSUE PUBLICATION INFORMATION

Conceptual Paper | Volume: 01 | Issue: 02

Nexus International Journal of Applied Science, Medicine and Social Sciences (NIJAMSS)

Open Access | e-ISSN: 3144-2196 | www.nijamss.org

Vol. 1, Issue 2, August 2026 | e-Book No. 011

Manuscript received: 16 June 2026 | Published: 31 August 2026 | Pages: 71-357

Published by Nexus Academic Press, an imprint of Nexus Health Solutions LLC

Cheyenne, Wyoming, United States of America

RIGHTS AND DISCLAIMER

Copyright © 2026 Peter Majoge. All rights reserved.

No part of this publication may be reproduced, stored, or transmitted in any form without prior written permission from the copyright holder, except for brief quotations used in scholarly review, teaching, or research as permitted by applicable law.

The author is responsible for the accuracy of the content, code, data examples, and citations.

CITATION SUGGESTION

Majoge, P. (2026). From R to Insight: A Handbook of Visualizations in Statistical and Data Science. NIJAMSS, 1(2), 71-357. Nexus Academic Press.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION.....	1
CHAPTER 2: DIMENSIONS OF DATA VISUALIZATION.....	3
2.1 Classification of Charts.....	6
2.2 Graphics with R Programming Language.....	7
2.3 Graphic Themes and Features.....	8
CHAPTER 3: DISTRIBUTION OF INFORMATION.....	13
3.1 Histogram.....	13
3.2 Mirror Histogram.....	15
3.3 Multiple Histograms.....	17
3.4 Faceted Histograms.....	18
3.5 Point Chart.....	20
3.6 Box Plot.....	21
3.7 Highlight Chart.....	24
3.8 Violin Plot.....	26
3.9 Rug Plot.....	27
3.10 Jitter Plot.....	28
3.11 Count Chart.....	29
3.12 Percentage Count.....	30
3.13 Vertical Count.....	31
3.14 Horizontal Count.....	32
3.15 Quantile Graph.....	34
3.16 Tertile Plot.....	35
3.17 Quantile-Quantile.....	36
3.18 Quantile-Quantile Line.....	37
3.19 Text Chart.....	39
3.20 Text Repel.....	40
CHAPTER 4: COMPOSITION.....	42
4.1 Pie Chart.....	42
4.2 Simple Pie Chart.....	43
4.3 Pie with White Labels.....	44

4.4 Pie with Polygonal Patterns.....	46
4.5 D Pie Chart.....	47
4.6 Exploded Pie Chart.....	48
4.7 Donut Pie Chart.....	50
4.8 Pie of Pie Chart.....	51
4.9 Bar of Pie Chart.....	52
4.10 Stacked Bar Chart.....	55
4.11 Treemap.....	56
4.12 Sunburst Chart.....	57
4.13 Waterfall.....	59
4.14 Hierarchy Chart.....	60
CHAPTER 5: COMPARISON.....	63
5.1 Bar Chart.....	63
5.2 Vertical or Column Chart.....	64
5.3 Horizontal Bar Graphs.....	66
5.4 Compound Bar Charts.....	68
5.5 Mirror Bar Chart.....	69
5.6 Divergent Bar Chart.....	71
5.7 Butterfly Chart.....	72
5.8 Population pyramid.....	74
5.9 Population Pyramid with Central Labels.....	77
5.10 Spider or Radar Chart.....	82
5.11 Dumbbell Chart.....	83
CHAPTER 6: CHANGE OVER TIME.....	86
6.1 Area Chart.....	86
6. 2 Time Series Plot.....	87
6.3 Smoothed Time Series Plot.....	89
6.4 Decomposed Time Series.....	91
6.6 Segmented Time Series.....	94
6.7 Highlighting Peaks.....	95
6.8 Step Chart.....	97

6.9 Filled Step Chart.....	98
6.10 Path Chart.....	100
6.11 Vertical Highlight.....	101
6.12 Rectangular Highlight.....	102
6.13 Annotated Chart.....	105
6.14 Segmented Chart.....	106
6.15 Radial Spoke chart.....	107
6.16 Normalized Spokes Chart.....	108
6.17 Stream chart.....	110
6.18 Slope Chart.....	111
6.19 Line Chart.....	113
6.20 Faceted Line Chart.....	114
6.21 Smoothed Line Chart.....	116
6.22 Regression Line.....	118
CHAPTER 7: RELATIONSHIP.....	119
7.1 Scatter Plot.....	119
7.2 Bubble Chart.....	122
7.3 Correlation Matrix.....	124
7.4 Time Series Correlation Chart.....	125
7.5 Heatmap.....	128
7.6 Candlestick.....	130
CHAPTER 8: UNCERTAINTY.....	133
8.1 Error Bar Chart.....	133
8.2 Errorbarh.....	135
8.3 Horizontal Errorbar.....	137
8.4 Cross bar Chart.....	139
8.5 Confidence Interval Band.....	140
8.6 Violin Plot.....	142
8.7 Ribbon Chart.....	144
8.8 Fan Chart.....	146
8.9 Linerange Chart.....	147

8.10 Pointrange Chart.....	149
CHAPTER 9: RANKING.....	151
9.1 Sorted Bar Chart.....	151
9.2 Lollipop Chart.....	153
9.3 Bump Chart.....	157
9.4 Ordered Tables.....	160
CHAPTER 10: GEOSPATIAL INFORMATION.....	161
10.1 Spatial Mapping.....	162
10.2 Geospatial Labeling.....	165
10.3 Annotated Map.....	166
10.4 Choropleth Map.....	169
10.5 Labeled Chloropleth Map.....	170
10.6 Symbol Map.....	172
10.7 Tile Mapping.....	173
10.8 Gridded Map.....	175
10.9 Raster Map.....	176
10.10 Geospatial Heatmaps.....	178
10.11 Two Dimensional Geospatial Heatmap.....	180
10.12 Road Lines.....	182
10.13 Land Cover Map.....	185
CHAPTER 11: FLOW.....	187
11.1 Sankey Diagram.....	188
11.2 Alluvial Diagram.....	191
11.3 Flowchart.....	194
11.4 Network Chart.....	198
CHAPTER 12: CONCENTRATION AND DENSITY.....	201
12.1 Density Chart.....	201
12.2 Two Dimensional Density Chart.....	203
12.3 Filled Two Dimensional Density Chart.....	204
12.4 Binned Density Chat.....	205
12.5 Hexbin Chart.....	207

12.6 Raster Plot.....	208
12.7 Histogram Plot.....	209
12.8 Two Dimensional Histogram.....	210
12.9 Contour Plot.....	211
12.10 Filled Contour Plot.....	212
12.11 Heatmap.....	213
12.12 Cartograms.....	214
CHAPTER THIRTEEN: MULTIVARIATE ANALYSIS.....	217
13.1 Parallel Coordinates Plot.....	217
13.2 Radar Chart.....	218
13.4 Bubble Chart.....	219
13.5 Multivariate Heatmap.....	221
CHAPTER 14: ARRANGEMENT OF GRAPHICS.....	222
14.1 Patch Work and Annotation.....	223
14.2 Wrapping.....	224
14.3 Marginal Plot.....	225
GLOSSARY.....	227
INDEX.....	238

LIST OF TABLES

Table 2. 1: <i>Classification of Charts</i>	6
Table 4. 1: <i>Waterfall Dataset</i>	59

LIST OF FIGURES

Figure 3. 1: <i>Histogram Plot</i>	15
Figure 3. 2: <i>Mirror Histogram Plot</i>	16
Figure 3. 3: <i>Multiple Histograms Plot</i>	18
Figure 3. 4: <i>Faceted Histograms Plot</i>	19
Figure 3. 5: <i>Point Plot</i>	21
Figure 3. 6: <i>Box Plot</i>	22
Figure 3. 7: <i>Notched Box Plot</i>	24
Figure 3. 8: <i>Highlighted Box Plot</i>	25
Figure 3. 9: <i>Violin Chart</i>	27
Figure 3. 10: <i>Rugged Chart</i>	28
Figure 3. 11: <i>Jittered Dot Plot</i>	29
Figure 3. 12: <i>Count Chart</i>	30
Figure 3. 13: <i>Percent Count Chart</i>	31
Figure 3. 14: <i>Vertical Count Chart</i>	32
Figure 3. 15: <i>Horizontal Count Chart</i>	33
Figure 3. 16: <i>Quantile Plot</i>	35
Figure 3. 17: <i>Tertile Plot</i>	36
Figure 3. 18: <i>Quantile-Quantile Plot</i>	37
Figure 3. 19: <i>Quantile-Quantile Line Plot</i>	38
Figure 3. 20: <i>Text Chart Plot</i>	40
Figure 3. 21: <i>Text Repel</i>	41
Figure 4. 1: <i>Simple Pie Chart</i>	44
Figure 4. 2: <i>Simple Pie with Labels</i>	45
Figure 4. 3: <i>Pie Chart with Polygonal Patterns</i>	47
Figure 4. 4: <i>Simple 3D Pie Chart</i>	48
Figure 4. 5: <i>Exploded Pie Chart</i>	49
Figure 4. 6: <i>Donut Pie Chart</i>	50
Figure 4. 7: <i>Pie of Pie Chart</i>	52
Figure 4. 8: <i>Bar of Pie Chart</i>	54
Figure 4. 9: <i>Stacked Bar Chart</i>	56

Figure 4. 10: <i>Tree Map</i>	57
Figure 4. 11: <i>Sunburst Chart</i>	58
Figure 4. 12: <i>Waterfall Chart</i>	60
Figure 4. 13: <i>Hierachy Chart</i>	62
Figure 5. 1: <i>Vertical Bar Chart</i>	65
Figure 5. 2: <i>Horizontal Bar Chart</i>	67
Figure 5. 3: <i>Compound Bar Chart</i>	69
Figure 5. 4: <i>Mirror Bar Chart</i>	70
Figure 5. 5: <i>Divergent Bar Chart</i>	72
Figure 5. 6: <i>Butterfly Chart</i>	73
Figure 5. 7: <i>Population Pyramid Chart</i>	77
Figure 5. 8: <i>Population Pyramid with Central Labels</i>	81
Figure 5. 9: <i>Spider Chart</i>	83
Figure 5. 10: <i>Dumbbell Chart</i>	85
Figure 6. 1: <i>Area Chart</i>	87
Figure 6. 2: <i>Time Series Plot</i>	89
Figure 6. 3: <i>Smoothed Time Series Plot</i>	90
Figure 6. 4: <i>Decomposed Time Series Plot</i>	92
Figure 6. 5: <i>Forecasted Time Series Plot</i>	93
Figure 6. 6: <i>Segmented Time Series Plot</i>	95
Figure 6. 7: <i>Highlighting Peaks</i>	97
Figure 6. 8: <i>Step Chart</i>	98
Figure 6. 9: <i>Filled Step Chart</i>	99
Figure 6. 10: <i>Path Chart</i>	100
Figure 6. 11: <i>Vertical Highlights</i>	102
Figure 6. 12: <i>Rectangular Highlights</i>	103
Figure 6. 13: <i>Highlight with Rectangular Shade</i>	104
Figure 6. 14: <i>Annotated Chart</i>	106
Figure 6. 15: <i>Segmented Chart</i>	107
Figure 6. 16: <i>Radial Spokes Chart</i>	108
Figure 6. 17: <i>Normalized Spokes Chart</i>	109

Figure 6. 18: <i>Stream Chart</i>	111
Figure 6. 19: <i>Slope Chart</i>	112
Figure 6. 20: <i>Line Chart</i>	114
Figure 6. 21: <i>Time Series Plot</i>	115
Figure 6. 22: <i>Smoothed Line Chart</i>	117
Figure 6. 23: <i>Faceted Smoothed Line Charts</i>	118
Figure 6. 24: <i>Regression Line</i>	119
Figure 7. 1: <i>Scatter Plot</i>	120
Figure 7. 2: <i>Scatter Plot with Correlation Index</i>	121
Figure 7. 3: <i>Bubble Chartt</i>	123
Figure 7. 4: <i>Correlation Matrix</i>	125
Figure 7. 5: <i>Time Series Correlation</i>	127
Figure 7. 6: <i>Heat Map</i>	129
Figure 7. 7: <i>Candlestick</i>	132
Figure 8. 1: <i>Error Bar Chart</i>	134
Figure 8. 2: <i>Error Barh Chart</i>	136
Figure 8. 3: <i>Horizontal Error Bar Chart</i>	138
Figure 8. 4: <i>Cross Bar Chart</i>	140
Figure 8. 5: <i>Confidence Intervals Chart</i>	142
Figure 8. 6: <i>Violin Chart</i>	143
Figure 8. 7: <i>Ribbon Chart</i>	145
Figure 8. 8: <i>Fan Chart</i>	147
Figure 8. 9: <i>Linerange Chart</i>	148
Figure 8. 10: <i>Point Range Chart</i>	150
Figure 9. 1: <i>Sorted Bar Chart</i>	152
Figure 9. 2: <i>Lollipop Chart</i>	155
Figure 9. 3: <i>Dot Plot</i>	157
Figure 9. 4: <i>Bump Chart</i>	159
Figure 9. 5: <i>Ordered Tables</i>	160
Figure 10. 1: <i>Geospatial Mapping</i>	164
Figure 10. 2: <i>Geospatial Labeling</i>	165

Figure 10. 3: <i>Annotated Maps</i>	168
Figure 10. 4: <i>Chloropleth Map</i>	169
Figure 10. 5: <i>Labeled Chloropleth Map</i>	171
Figure 10. 6: <i>Symbol Map</i>	173
Figure 10. 7: <i>Tile Mapping</i>	174
Figure 10. 8: <i>Gridded Map</i>	176
Figure 10. 9: <i>Raster Map</i>	178
Figure 10. 10: <i>Geospatial HeatMap</i>	180
Figure 10. 11: <i>2D Geospatial HeatMap</i>	181
Figure 10. 12: <i>Road Lines</i>	184
Figure 10. 13: <i>Land Cover Map</i>	187
Figure 11. 1: <i>Sankey Diagram</i>	190
Figure 11. 2: <i>Alluvial Diagram</i>	193
Figure 11. 3: <i>Flow Chart</i>	197
Figure 11. 4: <i>Network Chart</i>	199
Figure 12. 1: <i>Density Chart</i>	202
Figure 12. 2: <i>Univariate Density Chart</i>	203
Figure 12. 3: <i>2D Density Chart</i>	204
Figure 12. 4: <i>Filled 2D Density Chart</i>	205
Figure 12. 5: <i>Binned Density Chart</i>	206
Figure 12. 6: <i>Hexbin Density Chart</i>	207
Figure 12. 7: <i>Raster Chart</i>	208
Figure 12. 8: <i>Histogram Plot</i>	209
Figure 12. 9: <i>2D Histogram Chart</i>	210
Figure 12. 10: <i>Contour Lines</i>	211
Figure 12. 11: <i>Filled Contour Lines</i>	212
Figure 12. 12: <i>Heatmap</i>	213
Figure 12. 13: <i>Cartogram</i>	216
Figure 13. 1: <i>Parallel Coordinates Plot</i>	218
Figure 13. 2: <i>Radar</i>	219
Figure 13. 3: <i>Bubble Chart</i>	220

Figure 13. 4: <i>Multivariate Heatmap</i>	222
Figure 14. 1: <i>Patched Point and Histogram Plots</i>	224
Figure 14. 2: <i>Facet Wrapped Density Plots</i>	225
Figure 14. 3: <i>Assorted Marginal Plots</i>	227

Preface

Working with data often means facing pages of numbers, endless rows of decimals, and dense tables. It can be a lot to take in. But when that same information is transformed into a well-crafted chart, something magical happens as patterns become clear, trends emerge, and complex ideas turn into visual stories.

This book is for anyone who has ever wondered how to turn raw data into a compelling visual or struggled to interpret a chart in a research paper or report. Whether you're just starting out with data visualization or looking to sharpen your skills, this guide is here to support both your creativity and your critical eye.

This book explores two key aspects of graphics in statistical and data science: how to create them effectively, and how to understand them thoughtfully. While R will be the main tool for building these visuals, the focus goes beyond code. This book is about developing a deeper understanding of what makes a chart work, and how to use visual tools to communicate ideas clearly.

Inside, you'll find real code, practical examples, and simple explanations designed to make your learning experience smooth and enjoyable. You won't get lost in unnecessary complexity. Instead, the aim is to give you confidence in your ability to both construct meaningful charts and interpret them with clarity.

Whether you're a student, researcher, analyst, or curious learner, this handbook is your companion for making sense of data through graphics. Let's begin the journey towards turning data into insight, and charts into communication.

FOREWARD

In today's data-driven world, the ability to transform numbers into meaning is no longer a specialized skill reserved for statisticians and computer scientists, it is a foundational competency for every student and researcher. At both college and university levels, learners are confronted with vast and complex datasets that demand not only analysis but also clear, impactful communication. This is where data visualization becomes essential.

Data visualization is not only a technical skill but also a necessary background knowledge in many disciplines of higher learning. In business and economics, it enables learners to interpret market trends and financial performance. In the health and medical sciences, it helps communicate patterns of disease and treatment outcomes. In education and the social sciences, visualization assists in survey analysis and policy evaluation. Engineering and computer science students depend on it for performance monitoring, simulations, and system evaluations. Even in the humanities, visualizing textual and historical data strengthens interpretation and research communication.

Beyond the classroom, the demand for visualization skills extends to real-life areas where decisions depend on data clarity. Governments apply visualization in policy planning, census reporting, and budgeting. Industries use it for market research, production management, and customer analytics. In research organizations, visualization ensures evidence-based reporting. In media and journalism, it aids in communicating facts to the public. This book is therefore not just an academic resource, it is a practical bridge to professional fields, preparing personnel who will thrive as data analysts, researchers, policy advisors, and industry experts.

This book is suitable for a wide spectrum of learners: TVET institutions, diploma programs, undergraduates, and postgraduate students pursuing studies in statistics and econometrics, as well as instructors involved in teaching statistics and economics. It is particularly intended for econometrics and statistics courses such as *Applied Regression Analysis*, both I and II, and *Computing Methods and Data Analysis*. Chapters 3 through 14 are especially well-suited for student project work in data presentation with Chapter 10 providing focused techniques for displaying and analyzing the study area in research projects. With its structured progression and practical examples, the book stands as a resource for learners at different stages of academic and professional development.

P. O. Majoge

ACKNOWLEDGMENTS

Thanks galore to the many people who have assisted in the preparation of this work, particularly those who have read sections of the manuscript and offered constructive contributions that improved both the clarity and depth of presentation.

I am especially grateful to Dr. Ochenge of Embu University for his invaluable guidance throughout the development of this text. His webinar presentations, encouragement, and professional advice were instrumental in shaping the structure and relevance of the content.

I also wish to acknowledge the support of my colleagues and the academic community at Rongo University, Department Humanities and Social Sciences, whose feedback and discussions enriched the ideas presented in this book. Special appreciation goes to the many students whose curiosity, questions, and project work inspired much of the practical orientation adopted here.

The book further benefited from the availability of data resources from R open-source repositories, which formed the foundation for several illustrations and case studies. In addition, shapefile data reproduced from publicly accessible geographic information sources provided valuable materials for demonstrating applied aspects of visualization and spatial analysis.

To family, friends, and fellow researchers who encouraged me along this journey, I extend my deepest gratitude. Your patience, support, and unwavering belief in the importance of this work sustained me through its preparation.

CHAPTER 1: INTRODUCTION

In research, there are moments when simply presenting facts in plain text isn't enough. The goal isn't just to deliver information but to do so in a way that captures attention and encourages understanding. That's where the power of presentation comes in. Whether through visuals or carefully crafted text, how we share data shapes how it's received. It is widely recognized that our brains process visuals and text at different speeds. In fact, the human brain is remarkably more efficient when it comes to interpreting images. This difference plays a crucial role in how researchers and organizations choose to communicate complex ideas.

Data, in its raw form, can be overwhelming. Lengthy paragraphs filled with numbers might be accurate, but they are rarely engaging or easily digestible. That is why researchers often turn to visual formats, especially when space is limited or when quick clarity is needed. It is no surprise then, that the old saying, "a picture is worth a thousand words," continues to ring true, especially in today's fast-paced, data-heavy world.

Organizations across various sectors handle enormous amounts of data daily. One of their biggest challenges is figuring out how to make that information meaningful. They need to spot trends quickly, understand relationships within the data, and identify any anomalies. Visuals play a powerful role here. They are not just about aesthetics, they help bring clarity. Our minds are naturally drawn to images. They capture our attention faster, hold it longer, and often help us understand ideas that would otherwise take paragraphs to explain.

Visuals do more than just look good. They make complex ideas simpler and help the viewer see the story behind the numbers. That is because most people process images much faster than they do text. It is not just about speed, though, it is about engagement and comprehension. When we look at an info-graphic, we are not just seeing data, we are understanding it at a deeper level. And that is exactly what drives the demand for strong visuals in research, business, and education alike.

Data visualization, at its core, is about making data speak clearly. It involves transforming data into charts, maps, plots, or animations that bring out patterns, trends, and insights. It is not just

decoration, it is function. With the help of visual tools, raw figures become stories. These stories guide decisions, highlight connections, and uncover meaning that might otherwise remain buried in spreadsheets.

From simple bar charts to dynamic heat maps and interactive scatter plots, the range of visual tools available today is truly impressive. Each one offers a unique way to highlight different aspects of the data, making it easier to spot trends, relationships, or anomalies. But beyond the technical variety, the real purpose of visualization stays consistent. It's all about helping people not just see data, but actually grasp its meaning. A good visualization acts like a translator, turning numbers into something clear and relatable. It enables viewers to quickly understand complex information that might otherwise be lost in rows and columns. In doing so, data visualization becomes more than just a design choice; it becomes a powerful bridge between raw figures and real insight.

In the realm of R programming, these visualization techniques come to life with both elegance and power. R offers a rich library of tools and packages that make it easy to create compelling visuals. Whether you are working in healthcare, business, education, or social sciences, R gives researchers the ability to turn datasets into discoveries. The examples shared here are built on datasets that are already accessible within R's resources, making it easier to follow along and apply the concepts in your own work.

Ultimately, the goal isn't just to crunch numbers or run fancy analyses; it's to tell a story that others can understand, relate to, and act upon. That's where data visualization comes in. It bridges the gap between raw data and real-world decisions. Instead of staring at endless spreadsheets, people can see trends, patterns, and outliers at a glance. A well-crafted chart or graph doesn't just inform; it captures attention, sparks curiosity, and often makes complex information feel simple. In essence, visualization transforms abstract numbers into something human, something that resonates. It turns data into a shared language that both analysts and non-experts can appreciate.

CHAPTER 2: DIMENSIONS OF DATA VISUALIZATION

To effectively communicate insights through visualizations, it is essential to understand the key dimensions that shape how data is perceived and interpreted. These dimensions not only influence the clarity and effectiveness of the visual message but also determine how engaging and informative a graphic can be. The core visual elements include position, size, color, shape, pattern, time, hierarchy, interactivity, and spatial design. Each plays a unique role in translating raw data into meaningful visuals.

Position is foundational in data graphics because it dictates where elements appear within a chart or graph. It refers to the placement of data points in a two-dimensional or sometimes three-dimensional space, typically using an x and y axis. This positioning is not arbitrary; it carries meaning that viewers interpret almost instinctively. In a scatter plot, for example, the location of a point instantly communicates its value in relation to two variables. In bar graphs, the height or length of bars positioned along an axis indicates the magnitude of the data. When position is used accurately, it allows for quick comparisons between categories or variables. Viewers can detect patterns, trends, and even outliers with just a glance. Because human perception is highly sensitive to spatial differences, position remains one of the most powerful visual cues in data presentation.

Size in visualizations is often used to convey differences in quantity or magnitude. Larger shapes or symbols generally suggest higher values, while smaller ones indicate lower values. This technique is commonly seen in bubble charts, where the diameter of each bubble represents a particular data point's size. When applied correctly, size helps viewers understand proportions or scale without needing to read exact numbers. However, using size effectively requires careful calibration. Because people tend to underestimate or overestimate area visually, it's important to design proportional sizes that reflect the actual differences in the data. Misuse of size can easily lead to misleading interpretations or exaggerated impressions. Still, when done thoughtfully, size becomes an intuitive and efficient way to communicate volume or intensity in a dataset.

Color is a remarkably versatile tool in data visualization. It enhances readability, aids categorization, and draws attention to key aspects of the data. By assigning different colors to distinct data groups, such as in pie charts or stacked bar charts, a viewer can quickly distinguish

between categories. In continuous data, color gradients can be used to represent intensity, frequency, or change, as seen in heatmaps. For instance, deeper shades might indicate higher temperatures or more significant values, making trends easy to spot. Color can also highlight anomalies or direct focus to specific points within the visualization. It must, however, be used with accessibility in mind, ensuring sufficient contrast and avoiding problematic combinations for colorblind users. Overall, color helps bring clarity, emotion, and structure to data.

Pattern and texture serve as additional tools for distinction, especially in contexts where color may not be an option. For instance, in black-and-white documents or materials designed for printing, color loses its function, and texture takes over. Patterns such as stripes, dots, or crosshatching can separate one group of data from another without relying on hue. They are also valuable for enhancing the richness of a visual display, providing a tactile sense of variation. Patterns can be used in area charts, maps, or any visualization where differentiation is necessary. While patterns should be used sparingly to avoid clutter, they are extremely helpful when clarity must be maintained across various viewing conditions. When combined with other design elements, texture helps ensure that visualizations remain informative and accessible.

Time and interactivity add dynamic capabilities to modern visualizations, allowing users to explore and understand data more deeply. Time is essential when analyzing trends, especially in time-series data where changes unfold over days, months, or years. Incorporating a time axis or animation helps audiences observe progression or detect cycles. Interactivity enhances this further by giving viewers control over how they engage with the data. Features like sliders, filters, zoom options, and hover-over details allow users to dive into specific parts of a dataset. This turns static visuals into exploratory tools, where each person can focus on what interests them most. In dashboards and digital reports, interactivity creates a personalized experience that encourages deeper learning. Both time and interactivity transform data into a living narrative that evolves with user input.

Hierarchy is the invisible guide that tells the viewer what to look at first, second, and last. It creates order within a visualization and ensures that the most important information stands out clearly. Hierarchy can be established through a combination of techniques such as font size, boldness, color contrast, spatial arrangement, and even animation. For example, a large, bold title

at the top of a chart draws the eye immediately. Similarly, brighter or more saturated colors tend to grab attention before muted tones. By organizing visual elements with intention, designers lead the viewer through a logical flow of information. This prevents confusion and enhances the storytelling aspect of the visualization. Without hierarchy, even well-designed charts can feel chaotic or overwhelming. With it, they become coherent and persuasive.

Shape plays a subtle but powerful role in visual communication. Different shapes can carry different meanings, making them useful for encoding information. Circles often suggest continuity or completeness, while triangles might indicate direction or change. Squares and rectangles can convey stability or structure. In scatter plots or node-link diagrams, shapes can be used to categorize different data groups or to represent specific variables. When viewers recognize shapes easily, they can make associations quickly and interpret the data more efficiently. Shapes also help maintain visual interest and variety, especially when used alongside color and size. Importantly, the choice of shape should align with the message the visualization aims to communicate. In the right context, shape adds a layer of meaning that enhances understanding.

Spatial and three-dimensional (3D) perspectives offer new ways of exploring complex datasets, especially in areas like geography, architecture, and environmental studies. By adding depth to a visualization, 3D perspectives can mimic real-world environments, making data more relatable. For example, 3D maps help planners see how population density, land use, or elevation vary across regions. Virtual models can illustrate the impact of changes over time, such as urban expansion or environmental degradation. However, 3D should be used thoughtfully, as it can sometimes introduce distortion or make values harder to compare. When done well, though, spatial visualization offers an immersive experience that enhances insight. It encourages users to explore not just what the data says, but how it behaves in context. This approach turns visualizations into tools for discovery and decision-making.

Overall, these visual elements are far more than decorative choices. They form the core grammar of visual communication. When used purposefully, elements like position, size, color, pattern, and shape help transform complex datasets into narratives that are engaging, informative, and accessible. Each element contributes something unique to the story being told, allowing viewers

to grasp relationships, identify trends, compare categories, and interpret spatial or temporal changes. Rather than overwhelming the audience with numbers, effective visualization makes the information feel manageable and meaningful. Whether in a simple chart or a sophisticated interactive dashboard, the combination of these techniques ensures that data is not only seen but deeply understood. Ultimately, great data visualization invites curiosity, drives decisions, and turns information into insight.

2.1 Classification of Charts

As highlighted previously, various graphical shapes are designed to fulfill distinct roles in visual communication. A single geometric shape can often serve multiple functions within a single chart or infographic. For instance, a heatmap is a versatile tool that can effectively illustrate data concentration, reveal underlying relationships, and convey geographic distribution patterns all at once. Similarly, a stacked bar chart goes beyond merely displaying the breakdown of data into components. It can also demonstrate how these components evolve across different time intervals, making it useful for simultaneously presenting both composition and temporal trends. This multifunctional use of shapes enhances the depth and clarity of data interpretation, allowing viewers to gain more insights from a single visual representation.

Table 2. 1: Classification of Charts

Purpose	Description	Examples
Distribution	Shows spread of data.	Histogram, Boxplot, Violin, Jitter, Points
Comparison	Compares values or groups side by side.	Bar Chart, Butterfly, Divergent Bar
Trend Over Time	Shows change over time.	Line, Area, Slope
Relationship or Correlation	Shows connection between variables.	Scatter, Bubble, Heatmap
Uncertainty	Shows variability or confidence.	Ribbon, Error Bars, Fan Chart
Flow or Movement	Shows movement between states.	Sankey, Alluvial
Geospatial	Used in map-based visualization.	Choropleth Map
Concentration or Density	Shows concentration of data points.	Contour Plot, 2D Histogram
Multivariate Analysis	Pinpoints complex multi-variable relationships or connections	Parallel Coordinates Plot
Composition (Part-to-Whole)	Highlights proportions of a whole.	Pie, Donut, Waterfall, Treemap

Each chart type holds unique value and plays a specific role in data storytelling. Whether you're trying to reveal trends, highlight comparisons, or display distributions, the right chart can make

your message clear and compelling. It is not just about visual appeal but also about functional accuracy, ensuring that the data is understood as intended. The responsibility rests with the user to evaluate the nature of the dataset and determine which visualization best suits the objective. For instance, while a bar chart may efficiently compare categories, a scatter plot may better uncover relationships between variables. A well-chosen chart enhances clarity, draws attention to key insights, and ensures the audience grasps the message quickly. Ultimately, effective visualization bridges the gap between raw data and meaningful interpretation.

One of the areas of focus in this text is on R programming language which is a tool renowned for its robust capabilities in data manipulation and statistical analysis. Particular attention is given to the `ggplot2` package, one of the most widely used and versatile tools for producing high-quality visualizations in R. Built on the principles of the grammar of graphics, this package enables users to construct plots layer by layer, offering a high degree of flexibility and customization. Through carefully selected examples and demonstrations, the book illustrates how various chart types can be created and tailored to suit specific analytical needs. By harnessing the strengths of `ggplot2`, it becomes possible to design visual outputs that are both analytically meaningful and visually compelling, suitable for both exploration and presentation.

2.2 Graphics with R Programming Language

Unlike traditional software platforms such as Microsoft Excel, SPSS, or EViews, which often generate static and rigid graphics, R programming offers a highly adaptable and dynamic framework for data visualization. It allows users to create and refine visualizations interactively, tailoring each graphic to meet specific analytical goals or aesthetic preferences. R provides an extensive suite of tools that empower users to modify visual elements with great precision. Elements such as themes, color palettes, text styles, and shape attributes can be easily adjusted, offering full control over both the design and functionality of each visualization.

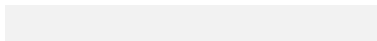
At the heart of graphic creation in R lies the powerful and widely used `ggplot2` package. It stands out for its intuitive syntax, logical structure, and highly flexible layering system that lets users build visualizations step by step. The beauty of `ggplot2` lies in how it separates data, aesthetics, and geometry, allowing for modular and elegant chart creation. One of the most fundamental building blocks in this system is the use of `geoms`, short for geometrical objects.

These `geoms` determine the type of graphical element that represents your data, whether it's a bar, a line, a point, or a ribbon. Each `geom` carries visual meaning and plays a vital role in shaping how viewers interpret the data being presented. They essentially act as the visual vocabulary of a chart, translating raw numbers into understandable graphics. To explore and access the wide variety of geometrical layers offered in `ggplot2`, users can run a few simple R commands that reveal the richness of this visualization toolkit.

```
library(ggplot2)
ls("package:ggplot2")
grep(pattern = "geom", x = ls("package:ggplot2"), value = TRUE)
```

Running these lines of code within an R script will produce a list of over 580 visual tools, including 57 specific geometric functions designed for various chart types. It is important to note that the number and functionality of these tools may vary depending on the version of R and `ggplot2` installed. Features are periodically updated, and some older commands may be phased out or replaced with enhanced alternatives. The examples and visualizations discussed in this text are based on RStudio running version R 4.4.2. This version includes a comprehensive range of geometric functions, which we will draw upon to illustrate different use cases and visualization strategies throughout the chapters ahead.

```
[1] "geom_abline"  
[2] "geom_area"  
[3] "geom_bar"  
[4] "geom_bin_2d"  
[5] "geom_bin2d"  
[6] "geom_blank"  
[7] "geom_boxplot"  
[8] "geom_col"  
[9] "geom_contour"  
[10] "geom_contour_filled"  
[11] "geom_count"  
[12] "geom_crossbar"  
[13] "geom_curve"  
[14] "geom_density"  
[15] "geom_density_2d"  
[16] "geom_density_2d_filled"  
[17] "geom_density2d"  
[18] "geom_density2d_filled"  
[19] "geom_dotplot"  
[20] "geom_errorbar"  
[21] "geom_errorbarh"  
[22] "geom_freqpoly"  
[23] "geom_function"  
[24] "geom_hex"  
[25] "geom_histogram"  
[26] "geom_hline"  
[27] "geom_jitter"  
[28] "geom_label"  
[29] "geom_line"  
[30] "geom_linerange"  
[31] "geom_map"  
[32] "geom_path"  
[33] "geom_point"  
[34] "geom_pointrange"  
[35] "geom_polygon"  
[36] "geom_qq"  
[37] "geom_qq_line"  
[38] "geom_quantile"  
[39] "geom_raster"  
[40] "geom_rect"  
[41] "geom_ribbon"  
[42] "geom_rug"  
[43] "geom_segment"  
[44] "geom_sf"  
[45] "geom_sf_label"  
[46] "geom_sf_text"  
[47] "geom_smooth"  
[48] "geom_spoke"  
[49] "geom_step"  
[50] "geom_text"  
[51] "geom_tile"  
[52] "geom_violin"  
[53] "geom_vline"  
[54] "get_geom_defaults"  
[55] "guide_geom"  
[56] "is_geom"  
[57] "update_geom_defaults"
```



2.3 Graphic Themes and Features

In the process of building effective graphics, several elements come together to shape a complete and polished chart. These often include the main title, labels for the x and y axes, background styling, legend formatting, scale adjustments, line properties, and font choices for both labels and headings. Such features are not merely decorative; they play an important role in improving the towards readability and overall visual impact of the chart. Within the ggplot2 environment in RStudio, these components can be layered and customized to enhance both clarity and presentation quality in any type of data visualization.

```
fill="grey": This adds a grey shade to a chart.
color="black": This indicates the color of a line/text, for example black.
alpha=0.5: It shows the level of background transparency, e.g. 0.5.
ggtitle("Histogram") It shows the title of the chart, e.g. Histogram.
scale_y_continuous(expand = expansion(mult = c(0,0.1))):Expands y axis only.
scale_x_continuous(expand = expansion(0)):Leaves x axis scale unchanged.
xlab("Volume"): Adds title to x-axis, e.g. Volume.
ylab("Frequency"): Adds title to y-axis, e.g. Frequency.
guides(fill=guide_legend(keywidth=0.8,
  keyheight=0.8,
  default.unit="cm")):Sets the unit(cm)and changes size of legend box,e.g.0.8
theme(panel.background = element_blank():Clears background features to blank.
  legend.position.inside = c(.8, .3)):Places the legend at point(.8,.3).
theme(plot.margin = unit(c(0.2,0.2,0.2,0.2),"cm")):Adjusts the margins to 0.2
theme(panel.background = element_blank(),
  axis.line = element_line(color = "Black",
    linewidth = 1): Adjusts x-axis line color and size.
  axis.title = element_text(size = 12,
    color = "Black",
    face = "bold"): Adjusts x-axis title color and size.
  legend.margin = margin(t=5,l=5,r=5,b=5):Adjusts legend margins, all round.
  plot.title = element_text(size = 14,Adjusts chart title font size to 14.
    color = "black", Adjusts chart title text color to black.
    face = "bold", Adjusts text to bold.
    hjust = 0.5, Adjusts text position to Centre.
  , margin =margin(b=15))Adjusts the margin between the chart and its title.
  axis.ticks.x = element_blank():Removes ticks on the x-axis.
  axis.text =element_text(size = 12),Adjusts text size for both axes.
  axis.text.x = element_text(margin = margin(t=5)),Adjusts x-axis margin.
  axis.text.y = element_text(size = 12),Adjusts y-axis font size to 12.
  axis.title.y=element_text(margin=margin(r=10)),Adjusts y-axis right margin.
  legend.key = element_rect(color = NA,fill= NA),Adds rectangular outline to keys.
  legend.background=element_rect(color="Black"),Adds rectangular outline.
  legend.key.spacing.y=unit(0.1,"cm")):Adjusts vertical spacing for legends.
```

ggplot2 graphics are outputs of cumulative addition of some of the layers described above. Because some of the addendum layers do not change much, for example the margins, this text adopts the use of a tag to hold these common features to graphics to reduce redundancy in the illustrations. Let us give the tag the **Mytheme** to hold these common R chunks for chart building. Legend position is one of the most dynamic graphic features and it is important to keep on

monitoring the appropriateness of the position specified in `Mytheme` for every chart. This lookout shall also applies for the scales in x and y axes.

```
Mytheme<- theme(  
  panel.background = element_blank(),  
  axis.line = element_line(color = "Black", linewidth = 1),  
  axis.title = element_text(size = 12, color = "Black", face = "bold"),  
  legend.margin = margin(t = 5, l = 5, r = 5, b = 5),  
  plot.title = element_text(size = 14, color = "black", face = "bold", hjust = 0.5),  
  plot.margin = unit(c(0.2, 0.2, 0.2, 0.2), "cm"),  
  axis.ticks.x = element_blank(),  
  axis.text = element_text(size = 12),  
  axis.text.x = element_text(margin = margin(t = 5)),  
  axis.text.y = element_text(size = 12),  
  axis.title.y = element_text(margin = margin(r = 10)),  
  legend.key = element_rect(color = NA, fill = NA),  
  legend.background = element_rect(color = "Black"),  
  legend.key.spacing.y = unit(0.1, "cm")  
)
```

Aside from the visual features discussed earlier, there are essential foundational components in RStudio that support effective chart building within the `ggplot2` environment. Understanding these components is key to constructing meaningful and flexible graphics. The `ggplot2` system is built around a layered grammar of graphics, and four core elements play a central role in every chart: `data`, `aes`, `geom`, and `ggplot` itself. This section introduces each of these components and outlines how they contribute to creating well-structured visuals.

1. `ggplot`: This is the starting point for building any graphic using the `ggplot2` package. It sets up the plotting space or canvas where all other elements will be layered.
2. `aes`: Short for aesthetics, this function defines how variables in the data are mapped to visual properties such as position on the x and y axes, color, size, or shape. It forms the core of how data is visually interpreted.
3. `data`: This refers to the dataset you want to visualize. It is passed to the plot and forms the source from which variables are selected and aesthetics are drawn.
4. `geom`: This specifies the type of visual representation to use for the data, such as bars, lines, points, or histograms. Each `geom` _ function adds a specific graphical layer that communicates different aspects of the data.

Together, these components form the structure upon which most `ggplot2` charts are built. Throughout this book, we will explore how these layers interact to produce powerful and elegant graphics with minimal complexity.

CHAPTER 3: DISTRIBUTION OF INFORMATION

Most survey data are dispersed differently and the information contained can either be normal, skewed, or bimodal. The knowledge about the spread is important as it determines how the data should be handled in subsequent analyses. A skewed data, for example, does not meet OLS analysis criteria and in such cases non-parametric, data transformation or bootstrap techniques need to be performed instead. As such, we need to clearly determine the nature of data distribution beforehand. Distribution of data reveals the frequency of values, identify patterns, detect outliers, and provide insights into the shape and spread of the data. These charts are essential in statistical analysis, quality control, and exploratory data analysis. Common charts for distribution include histograms, box plots, violin plots, and density plots. Histograms show the frequency of values in intervals, helping to spot skewness and modality. Box plots summarize data using quartiles, highlighting the median, spread, and outliers. Violin plots combine box plots and density plots to show both summary statistics and distribution's shape. Density plots provide a smoothed view of data distribution, especially useful for large datasets. These charts are powerful tools for understanding how data values behave across a dataset.

3.1 Histogram

Ever wondered how your data points are spread out across a range? Imagine lining up all your observations and seeing which values appear most frequently, this is exactly what a histogram does. It divides data into bins and shows the frequency of observations in each bin, letting you immediately spot peaks, gaps, and skewed patterns. Use it when you want a clear view of the overall shape and spread of your data. Histogram is a graphical representation of a numerical data. It displays data using bars of different heights, where each bar groups numbers into ranges or intervals called bins. A lower bin width captures a higher resolution of histogram. The height of each bar shows the frequency of data points in that range. Histograms are useful for visualizing the shape of the data distribution such as skewness. It also highlights patterns relating to peaks, gaps and outliers. Just like other tools of comparison, a histogram highlights the spread and central tendency of the data. Unlike bar charts, histograms are used for continuous data, and the bars are adjacent to each other showing the continuous nature of the data. Histogram is a suitable tool in comparing two numeric variables in a chart. Important arguments in constructing a histogram include the `fill()` function for providing the color shades and `binwidth()` command which is basically for specifying the width of the bins. It is always in order to carry out data

cleaning prior to analysis to avoid crashes while trying to run the codes. Let us begin the coding journey by first of all preparing the trees dataset in R by categorizing the **Girth** measurements into two factor levels, **Small** and **Wide**.

```
library(ggplot2)
library(tidyverse)
library(dplyr)

mytreez<-trees
mytreez<-mytreez%>%
  mutate(Girth=case_when(Girth<=15~"Small",
                        Girth>15~"Wide"))
my_xlab<-paste(unique(mytreez$Girth), "\n(N=", table(mytreez$Girth), ")", sep="")
mytreez$Girth<-factor(mytreez$Girth,
                     levels = c("Small", "Wide"),
                     ordered = TRUE)
```

Now let us see what we get when we use our new dataset **mytreez** to construct a histogram. The command line **y=after_stat(count)** gives whole numbers of y values instead of decimals and **geom_histogram()** actually plots the histogram polygon. The use **binwidth=7** groups the data into distinct frequency intervals and **fill="grey"** gives the bars grey shade with black outlines through **color="black"**, make the distribution shape easy to interpret.

```
Histogram<-mytreez%>%
  ggplot(aes(x=Volume))+
  geom_histogram(aes(y=after_stat(count)),
                binwidth=7,
                fill="grey",
                color="black",
                alpha=0.9)+
  xlab("Volume")+
  ylab("Frequency")+
  ggtitle("Histogram Plot")+
  Mytheme
Histogram
```

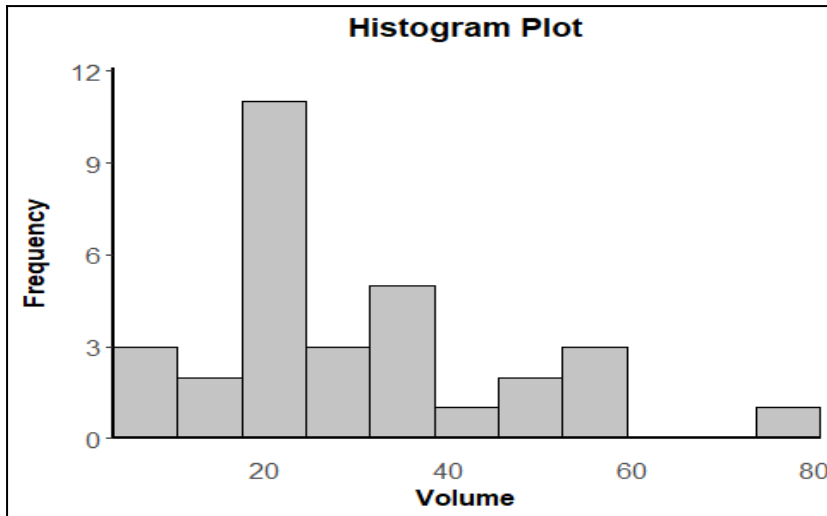


Figure 3. 1: Histogram Plot

The pattern reveals a moderate right skew, with most observations clustered at lower volumes and a gradual tapering towards larger volumes. This skewness suggests that the data deviates from perfect normality, meaning the normal fit would show a longer tail on the right. While the histogram still captures the central tendency effectively, the asymmetry should be considered in any statistical modeling that assumes a normal distribution. The long right tail is the main reason for the failed normality test.

3.2 Mirror Histogram

Suppose you want to compare two distributions side by side without cluttering your chart, a mirror histogram helps you flip one distribution over an axis, so the two distributions reflect each other like a mirror. This makes differences, overlaps, and trends between groups instantly visible, allowing you to draw meaningful comparisons at a glance. It is useful in comparing distribution of two variables. The name is derived from the appearance of bins, where the bins of the variables in question are placed in opposite direction from a horizontal axis. The construction of a mirror histogram still follows a similar procedure for building a histogram, but with a few adjustments, particularly introducing `geom_hline()` for plotting a horizontal axis. In this case, we set the fill function equals to the girth variable which holds the two factor levels.

```
Histogram_mirror<-mytreez%>%
  ggplot()+
  geom_histogram(data=subset(mytreez,
                             Girth=="Small"),
                 aes(x=Volume,
                     y=-after_stat(count),sets negative y values for the factor "Small"
                     fill =Girth),
                 binwidth=7,
```

```

    colour = "black",
    alpha=0.9)+
geom_histogram(data=subset(mytreeez,
    Girth=="Wide"),
    aes(x=Volume,
        y=after_stat(count), sets positive y values for the factor "Small"
        fill =Girth),
    binwidth=7,
    colour = "black",
    alpha=0.9)+
geom_hline(yintercept = 0,color="black")+
ggtitle("A Mirror Histogram")+
xlab("Volume")+
ylab("Frequency")+
scale_y_continuous(labels = abs)+ # display absolute y values
scale_fill_manual(values = c("grey", "pink"))+
Mytheme
Histogram_mirror

```

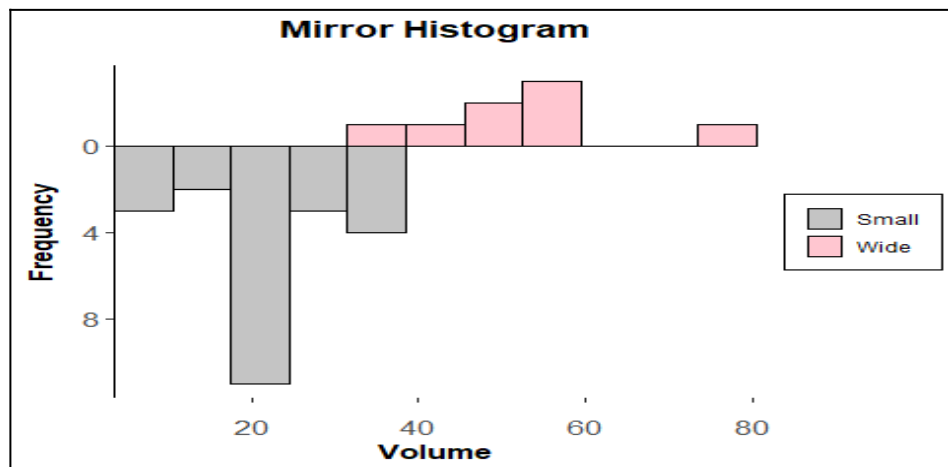


Figure 3. 2: Mirror Histogram Plot

A mirror histogram offers a side-by-side comparison of data distributions. For instance, in trees data the **Small** group is mirrored downward in grey, while the **Wide** group rises upward in pink, with both sharing a bin width of 7 for direct comparability. The symmetry of the layout highlights contrasting patterns: **Small** trees are concentrated at lower volumes, tapering quickly, while **Wide** trees show a broader spread toward higher volumes. The overall shape reveals that the **Small** group is notably more right-skewed, indicating a departure from normality, whereas the **Wide** group exhibits a distribution closer to symmetry, aligning better with a normal fit. The mirrored design enhances the detection of such asymmetries, making deviations from normality visually evident.

3.3 Multiple Histograms

When exploring a dataset with multiple groups or categories, it's often useful to see how distributions differ side by side. Multiple histograms let you compare distributions across subgroups, revealing patterns, overlaps, and unique characteristics that might remain hidden in a single aggregated view.

To examine the relationship between two continuous variables, the `fill()` aesthetic is essential. For example, specifying `fill = Girth` in the `ggplot()` call produces separate histograms for each `Girth` category; without it, the plot defaults to a single combined histogram. Using `position = "Stack"` arranges the bars in a stacked layout, showing cumulative values, whereas `position = "dodge"` separates the bars side by side, enabling a direct comparison of distributions across categories without overlap.

```
Histogram_multipleST<-mytreez%>%
  ggplot(aes(x=Volume,y=after_stat(count),fill = Girth ))+
  geom_histogram(binwidth = 5,
                 colour = "black",
                 alpha=0.9,
                 position = "Stack" )+
  ggtitle("Stacked Histogram")+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  xlab("Volume")+
  ylab("Frequency")+
  scale_fill_manual(values = c("grey","pink"))+
  Mytheme7
Histogram_multipleD<-mytreez%>%
  ggplot(aes(x=Volume,y=after_stat(count),fill = Girth ))+
  geom_histogram(binwidth = 5,
                 colour = "black",
                 alpha=0.9,
                 position = "dodge" )+
  ggtitle("Dodged Histogram")+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  xlab("Volume")+
  ylab("Frequency")+
  scale_fill_manual(values = c("grey","pink"))+
  Mytheme7
Histogram_multipleST + Histogram_multipleD + plot_annotation(title = "Multiple
Histograms Chart",caption = "Stack vs Dodge",
                                                              theme = theme(plot.title = element_text(size = 13, face =
"bold")))
```

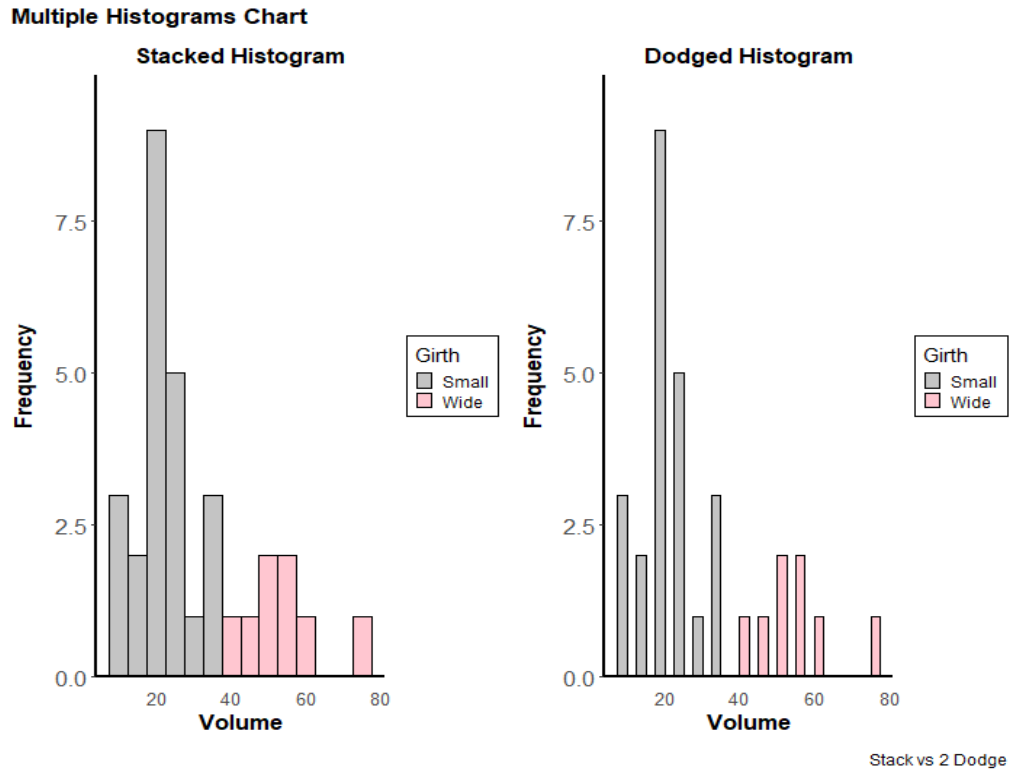


Figure 3. 3: Multiple Histograms Plot

In the stacked version (left), the **Small** group (grey) and **Wide** group (pink) are overlaid vertically within each bin, providing an aggregate perspective while still preserving group contributions. In the dodged version (right), the two groups are offset side-by-side within each bin, making direct frequency comparisons easier at each volume range. Both formats use a bin width of 5, ensuring consistency in measurement. The shapes indicate that the **Small** group is more right-skewed, with higher counts concentrated at lower volumes and a long tail towards the upper range, a pattern that suggests a poorer fit to normality. The **Wide** group, while still slightly skewed, is closer to symmetric and more consistent with a normal distribution. Together, the two layouts complement each other, one emphasizing total distribution and the other highlighting group differences.

3.4 Faceted Histograms

Sometimes a single histogram isn't enough, especially when your dataset has multiple categories. With faceted histograms, you can split the data into panels by category, displaying each subgroup's distribution separately. This approach makes it easy to compare patterns across groups, see variability, and uncover subtle differences that might be hidden in a combined view.

Instead of cramming everything into one crowded plot, faceting lets you break things out into separate panels representing each category or group. In R's ggplot2, you can use `facet_wrap()` or `facet_grid()` to create them with just a line or two of code. It's a simple trick that instantly boosts clarity when you're dealing with grouped data. By setting the `scales="free_y"` allows each distribution to go to its maximum value on the y-axis. The number of rows in the facets is determined by `nrow()`.

```
Histogram_multiple_facet<-mytreez%>%
  ggplot(aes(x=Volume,y=after_stat(count),fill = Girth ))+
  geom_histogram(binwidth = 5,
                 colour = "black",
                 alpha=0.9,
                 position = "Stack" )+
  facet_wrap(~Girth,nrow=2, scales="free_y")+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  xlab("Volume")+
  ylab("Frequency")+
  scale_fill_manual(values = c("grey","pink"))+
  Mytheme
Histogram_multiple_facet
```

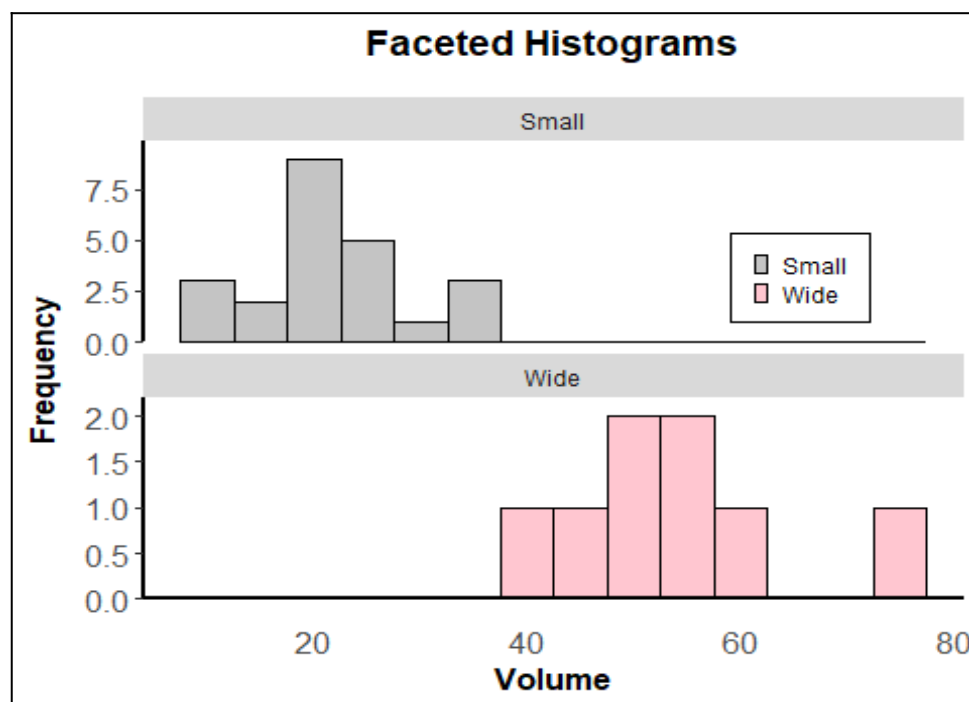


Figure 3. 4: Faceted Histograms Plot

In the chart above, facet separates the `Small` (grey) and `Wide` (pink) girth groups into distinct panels, making their individual volume distributions easier to compare without overlap. Each uses a bin width of 5, with scales adjusted independently (`scales="free_y"`) to maximize visual clarity for each category. The `Small` group displays a pronounced right skew, a high

concentration of low-volume trees and a long tail towards higher volumes, indicating deviation from a normal distribution. The **Wide** group appears more balanced, with its frequency pattern closer to symmetry, although still slightly skewed. The faceted layout highlights these differences more explicitly than stacked or dodged formats, making it clearer which group fails normality more strongly. When volumes are split by girth groups, the **Small** appears more symmetric and closer to normality, while the **Wide** has a more pronounced right tail, showing stronger skewness. This skewness explains why the large girth group departs more from a normal distribution.

3.5 Point Chart

Think of a scatter of dots representing every single observation, this is the essence of a point chart. It allows you to visualize distribution, clusters, gaps, or outliers directly, and you can encode extra dimensions like size or color to reveal relationships within the dataset. This is the go-to function when you're creating scatter plots. It plots individual data points as dots, showing how two variables relate to each other. It's especially useful when you're working with continuous data on both axes. `geom_point()` makes it really easy to spot clusters, trends, or outliers at a glance. It's simple, clean, and often the starting point for exploring relationships in your data. The use of `alpha()` helps to improve the transparency in case you're dealing with over plotting. You can map aesthetics like color, size, or shape to additional variables to reveal even more patterns.

```
Mypoint<-mytreez%>%
  ggplot(aes(x=Girth,y=Height, fill = Girth,colour = Girth))+
  geom_point(size=2)+
  scale_color_manual(values = c("black","grey"))+
  scale_fill_manual(values = c("pink","brown"))+
  xlab("Girth")+
  ylab("Height")+
Mytheme
Mypoint
```

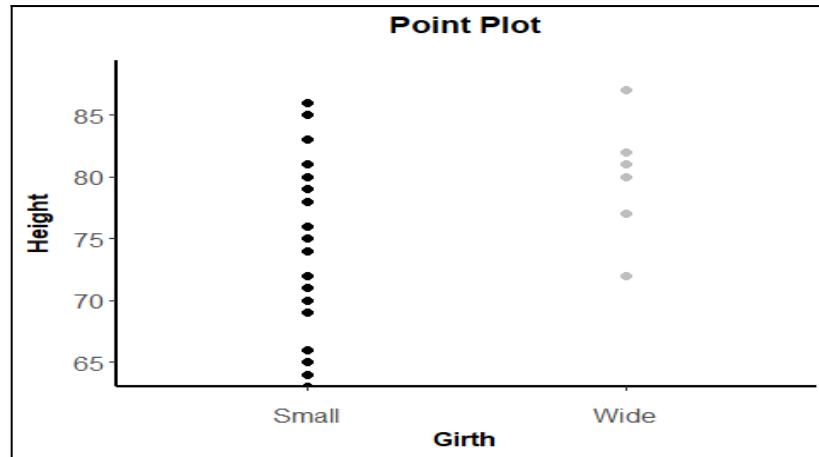


Figure 3. 5: Point Plot

In the chart, the point plots compare tree height across two girth categories, **Small** (pink fill, black outline) and **Wide** (brown fill, grey outline). The **Wide** group generally appears taller, clustering towards the upper portion of the y-axis, while **Small** trees are more dispersed at lower heights. The use of both fill and outline color improves category separation, and the clean axes labelling enhances interpretability. No transformation or smoothing is applied, allowing raw relationships between **Girth** and **Height** to remain visible. The scatter of heights versus girth reveals a positive association, with taller trees generally having larger girths. The spread is tighter for smaller girths but more variable for larger girths, hinting at unequal variance (heteroscedasticity) that may influence normality in residuals if modeled.

3.6 Box Plot

When the goal is to condense data into a clear summary without sacrificing insight, the boxplot offers a powerful solution. This visualization encapsulates the median, interquartile range (IQR), and potential outliers, enabling a rapid assessment of central tendency, spread, and unusual values within a dataset. The box itself represents the middle 50% of observations, bounded by the first (Q1) and third quartiles (Q3), while the line inside marks the median. Whiskers typically extend to the smallest and largest observations within $1.5 \times \text{IQR}$ from the quartiles, beyond which individual points are plotted as outliers. These outliers are determined by adding $1.5 \times \text{IQR}$ to Q3 for the upper threshold and subtracting $1.5 \times \text{IQR}$ from Q1 for the lower threshold. Some implementations may adopt stricter cutoffs (such as $3 \times \text{IQR}$) to highlight extreme cases. Additionally, notches around the median provide a rough visual confidence interval, if notches between groups overlap, it suggests their medians are not significantly different. This makes

boxplots especially valuable for comparing distributions across categories, detecting asymmetry, and spotting influential values that could bias subsequent analysis. A box plot used together with `geom_jitter()` gives a better impression of data distribution than a box plot used alongside `geom_points()`. At times we may need to set the `legend.position()` function to 'NA' when it is not necessary to show the legend elements and also to avoid unnecessary congestion in the chart. Further, either `geom_jitter()` or `geom_point()` can be added immediately after the `ggplot()` argument before the box plot. Geom jitters is preferred to improve visualization in data overlaps.

```
mybox<-mytreez%>%
  ggplot(aes(x=Girth,y=Height, fill = Girth,colour = Girth))+
  geom_jitter()+
  geom_boxplot(alpha =0.5,
               outlier.shape = NA )+
  scale_color_manual(values = c("black","brown"))+
  scale_fill_manual(values = c("pink","orange"))+
  xlab("Girth")+
  ylab("Height")+
  labs(title="Box Plot ",fill=NULL)+
  Mytheme
mybox
```

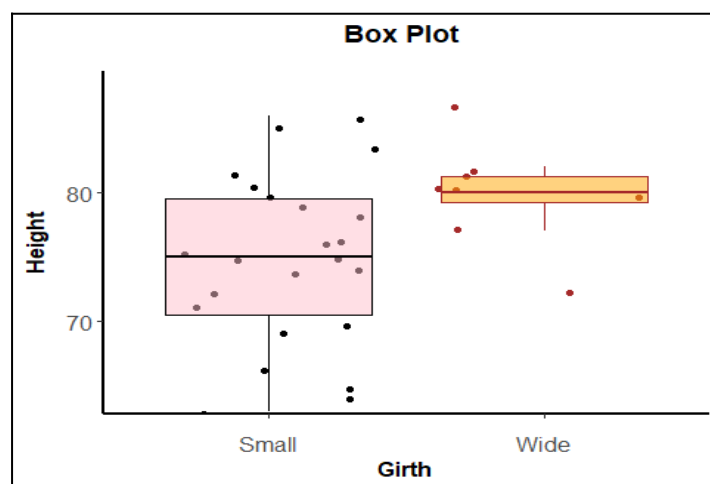


Figure 3. 6: Box Plot

By looking at the chart, you notice that the box plot overlays jittered points to display both the summary distribution and individual observations of tree `Height` for each `Girth` category. The boxes (pink for `Small`, orange for `Wide`) summarize the median, interquartile range, and spread, while jittered points reduce overlap and reveal raw data density. Outliers are suppressed in the box layer (`outlier.shape = NA`) so they remain visible only through jitter. `Wide` trees tend to show higher median heights and greater spread compared to `Small` trees. The combination of box and scatter layers allows both central tendency and variability to be interpreted simultaneously.

Sometimes, you may also need to show the size of each distribution along the x_labels. Such a task requires that a variable is first of all created to hold the intended labels. In this example, `my_xlab` is uniquely created to hold the x-labels that contain the sample size of each distribution together with the unique names from the dataset. The labels are retrieved using the following command lines:

```
my_xlab<-paste(unique(mytreez$Girth),"\n(N=",table(mytreez$Girth),")",sep="").
```

`Scale_x_discrete(labels=my_xlab)` is now used to place the labels embedded in `my_xlab` along the x-axis. Other statistics such as the mean can also be shown on the boxplot with the help of `stat_summary()` function.

```
mybox_notched<-mytreez%>%
  ggplot(aes(x=Girth,y=Height, fill = Girth))+
  geom_jitter(size=1,color="black",alpha=.3,width = .2)+
  geom_boxplot(alpha=.7,
               varwidth = .4,
               outlier.shape = 8,
               outlier.color = "red",
               outlier.size = 2) + # 8 creates outlier with asteriks
  stat_boxplot(geom = "errorbar",width=.1)+
  scale_color_manual(values = c("black","grey"))+
  scale_fill_manual(values = c("pink","brown"))+
  scale_x_discrete(labels=my_xlab)+
  stat_summary(fun.y = mean, # Adding extra statistics, mean, error
               geom = "point",
               shape=23,
               size=3,
               color="blue",
               fill="green")+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  ylab("Height")+
  labs(title="Notched BoxPlot",fill=NULL)+
Mytheme
mybox_notched
```

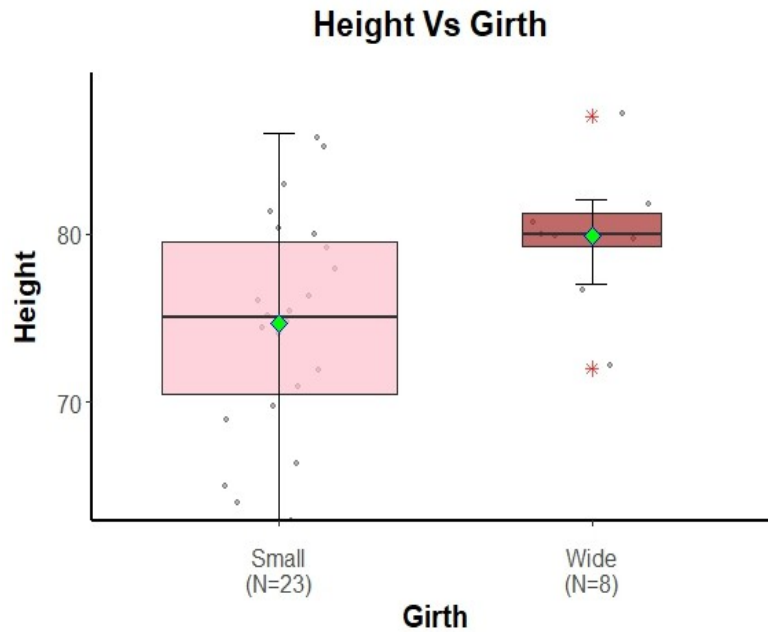


Figure 3. 7:Notched Box Plot

The notched box plot reveals that the **Wide** trees generally sits higher on the height scale, with a noticeably taller median and more spread-out values compared to the trees with **Small** girth. The notches show no overlap, suggesting distinct medians between the groups. Outliers in the **Wide** girth group extend far above the upper whisker, indicating occasional extreme tall trees. The green-blue diamonds highlight group means, which align closely with medians in the **Small** girth group but are slightly higher in the **Wide** girth group, hinting at mild right skewness. The varied box widths also convey that the **Wide** girth group has more observations, giving more weight to its distribution pattern. The notches indicate that median heights differ between girth groups, with **Wide** girths having higher medians. The **Wide** girth group's box is more extended above the median, reflecting right skewness and poor normality fit. The **Small** girth group's distribution is tighter and more symmetric.

3.7 Highlight Chart

If you need to focus attention on critical values in a sea of data? A highlight chart draws the viewer's eye to important points or ranges, making patterns, peaks, or thresholds instantly recognizable. This is especially useful when analyzing large datasets where key insights might otherwise be overlooked.

Highlighting information in a chart is not part of base `ggplot2()`, but it's incredibly helpful when you want to emphasize certain data points or lines within a plot. Rather than manually filtering your data, `scale_fill_manual(values = c("white", "yellow"))` lets you specify conditions, for instance, showing only groups with values above 100. It dims everything else, keeping the focus where you want it. It's great for storytelling, drawing attention to trends or outliers that matter without deleting the rest of your context.

```
mybox_highlight<-mytreez%>%
  ggplot(aes(x=Girth,y=Height, color=Girth,fill = Girth))+
  geom_jitter(size=1,color="blue",alpha=.3,width = .2)+
  geom_boxplot(varwidth = .4,
              alpha=.2,
              outlier.shape = 8,
              outlier.color = "red",
              outlier.size = 2) + # 8 creates outlier with asterisk
  stat_boxplot(geom = "errorbar",width=.1)+
  scale_fill_manual(values = c("white", "yellow"))+# highlights the box with a
yellow color
  scale_color_manual(values = c("black", "grey"))+
  scale_x_discrete(labels=my_xlab)+
  stat_summary(fun.y = mean, # Adding extra statistics, mean, error
              geom = "point",
              shape=23,#diamond shape
              size=3,
              color="blue",
              fill="green")+
  ylab("Height")+
  labs(title="Highlighted Boxplot",fill=NULL)+
  Mytheme
mybox_highlight
```

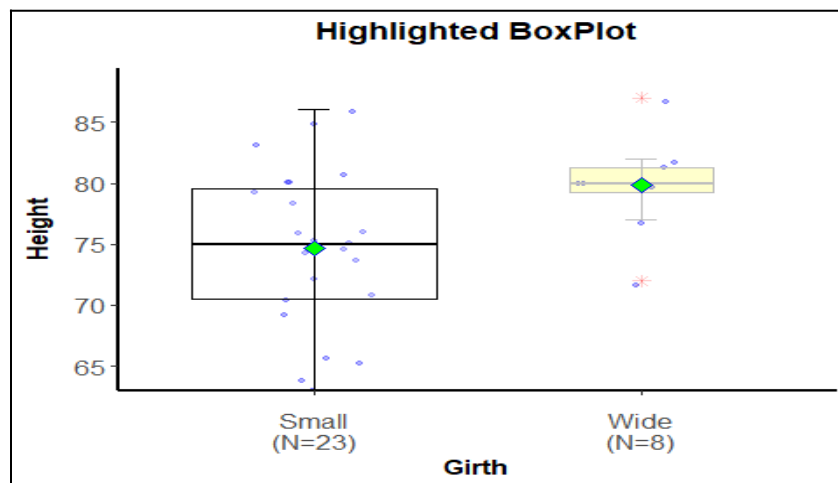


Figure 3. 8: Highlighted Box Plot

You can spot two distinct distributions in the boxplot chart above. Also, at a glance, you notice that the two distributions are emphasized differently. The distribution for trees with **Wide** girth have been highlighted in yellow to single it out for some reasons, which can range from outlier to heteroscedasticity issues as depicted in the highlighted distribution. The yellow-filled box

instantly draws the eye to the **Wide** group, making its taller median and wider spread obvious compared to the **Small** group. Blue jittered points hint at greater variability in height, especially in the wide girth group where outliers stretch well above the main bulk of data. The green–blue diamond marks the mean, sitting slightly above the median in the **Wide** group, suggesting mild right skewness. The **Small** girth group’s tighter box and fewer extreme points indicate more consistent tree heights. The highlighted box for the **Wide** girth group makes the skewness visually obvious, the upper whisker is stretched, and multiple high outliers appear, confirming deviation from normality. The **Small** girth group maintains near symmetry.

3.8 Violin Plot

You may be interested in not just seeing the summary only, but the shape of your data distribution as well. A violin chart combines a boxplot with a density plot, showing both spread and density. This allows you to explore variability and detect clusters across categories, giving a richer view than a simple boxplot. It visualizes the density of the data at different values, allowing you to see the shape and symmetry of the distribution clearly. Construction of a violin chart requires at least three data points for the shape to be displayed. The code uses `geom_violin(alpha = 0.5, outlier.shape = NA )` to create the translucent violin shapes, where the alpha controls the transparency so the underlying points remain visible, and setting `outlier.shape = NA` hides individual outliers that might clutter the view. To complement this, `geom_jitter()` adds scattered points with slight random noise so each individual measurement is visible without overlapping too much. The `fill=NULL` in the labels simply means no extra fill color is applied to the legend, keeping the focus on the violins and points themselves. Together, these functions craft a clear, detailed visualization of how tree heights distribute within each girth category.

```
Myviolin<-mytreez%>%
  ggplot(aes(x=Girth,y=Height, fill = Girth,colour = Girth))+
  geom_jitter()+
  geom_violin(alpha = 0.5,
              outlier.shape = NA )+
  scale_fill_manual(values = c("black","brown"))+
  scale_color_manual(values = c("orange", "blue"))+
  xlab("Girth")+
  ylab("Height")+
  labs(title="Violin ",fill=NULL)+
Mytheme
Myviolin
```

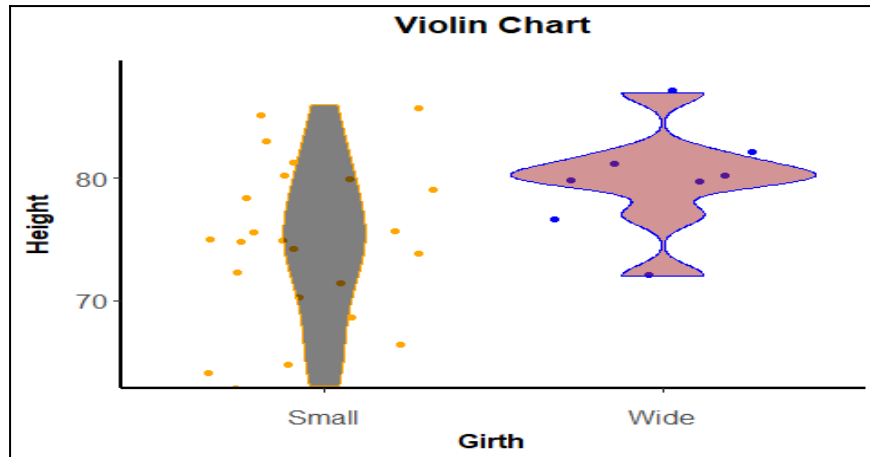


Figure 3. 9: Violin Chart

The violin shapes reveal the full distribution of tree heights for each girth group, with the **Wide** group showing a broader and slightly asymmetric density, hinting at a mild right skew. The narrow necks of the violins indicate fewer trees at those height ranges, while the bulges show where values cluster. The **Small** girth group's slimmer, more symmetric violin suggests tighter height consistency and closer fit to normality. The violin shapes show full density profiles. The **Wide** group has a slightly asymmetric, right-skewed bulge towards higher heights, suggesting poorer normality fit. The **Small** girth group is slimmer and symmetric, aligning more closely with normal distribution assumptions.

3.9 Rug Plot

Drawing curiosity of the reader or viewer to where each individual data point falls is crucial in graphical communication. Used alongside histograms or density plots, it gives a precise, detailed view of the distribution without cluttering the chart. It is used to add little tick marks, called "rugs", along the x and/or y axis to show the distribution of individual observations. It's like a mini-histogram that hugs the edge of your plot. Rug plots are particularly helpful when combined with scatter plots, as they give a quick view of data density. You'll often see them used to spot overlaps or concentrations of points that aren't obvious in the main plot. Just a few lines, and you can add that extra bit of context to your visuals. For instance, the code `geom_rug(sides="r")` is used to plot the rugs to the right of the chart.

```
Myrug<-mytreez%>%
  ggplot(aes(x=Volume,y=Height, colour = Girth))+
  geom_point()+
  geom_rug(sides="r")+
```

```
scale_color_manual(values = c("black", "blue"))+
ylab("Height")+
xlab("Volume")+
labs(title="Rugged Chart", fill=NULL)+
Mytheme
Myrug
```

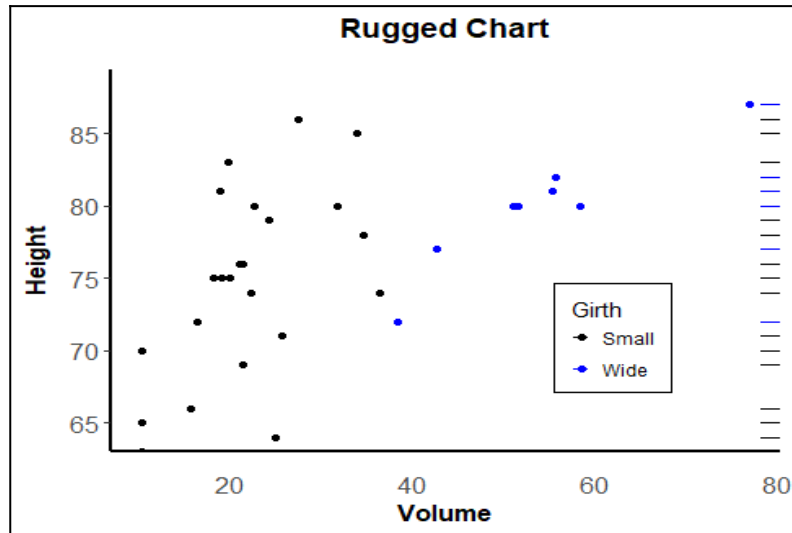


Figure 3. 10: Rugged Chart

The scatter points show a positive relationship between tree volume and height, with the rug marks along the right side revealing where observations cluster vertically. The **Wide** group has a concentration of points in higher heights at around 80 ft but also spreads upward with a few extreme tall trees, indicating right skewness in height. The **Small** girth group is more evenly distributed, showing less skew and a closer fit to normality.

3.10 Jitter Plot

Jittered points help separate overlapping observations, while the dot plot stacks observations along the vertical axis, revealing density patterns. Jitter slightly nudges the data points away from each other, helping you see how many values are really there. It is especially useful for categorical data or discrete x or y values where multiple points might fall exactly on top of each other. Jittering adds just enough randomness to make the data more readable, without distorting the overall pattern. In such situations of where your scatter plot points are overlapping too much, `geom_jitter()` comes to the rescue. The addition of `geom_jitter()` adds some noise on the axes which makes it pretty well to deal with over plotting in addition to improving the transparency using the `alpha()`. The `geom_dotplot()` function creates a dot plot with dots stacked

vertically (`binaxis="y"`) and centered horizontally (`stackdir="center"`), each dot representing one or more tree height observations.

```
myjitter_dot<-mytreez%>%
  ggplot(aes(x=Girth,y=Height, fill = Girth,colour = Girth))+
  geom_jitter()+
  geom_dotplot(binaxis="y",
               stackdir="center",
               dotsize=.9
               ,alpha =0.5)+
  scale_color_manual(values = c("blue", "brown"))+
  scale_fill_manual(values = c("black", "green"))+
  xlab("Girth")+
  ylab("Height")+
  labs(title="Jitters and Dot Plot",fill=NULL)+
  Mytheme
myjitter
```

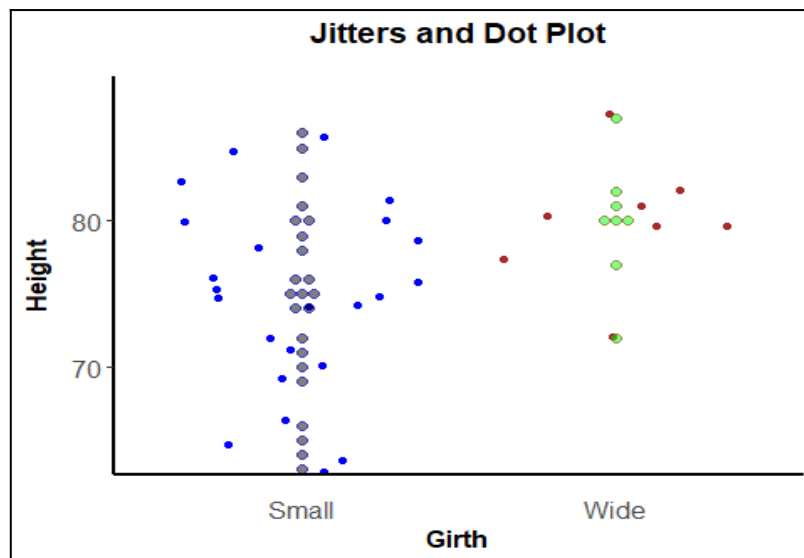


Figure 3. 11: Jittered Dot Plot

In the chart above, the `Wide` group has a taller stack concentrated at higher heights but with some spread upward, showing moderate right skewness and deviation from normality. The `Small` group shows a more even stacking pattern with less skew, aligning more closely with a normal distribution.

3.11 Count Chart

Ever want to see which categories dominate your dataset at a glance? A count chart plots the frequency of each category as points or bars, highlighting which groups are most or least common. It's perfect for spotting imbalances or trends in categorical data.

Again, if you have lots of overlapping points in a scatter plot, `geom_count()` offers another great solution. Instead of moving points around like `geom_jitter()`, it shows the number of overlapping observations by varying the size of each point. So, the bigger the dot, the more data points are stacked at that location. It's especially handy for datasets with many duplicates or discrete combinations. It quickly reveals where the densest concentrations are and gives a more honest view of your data's structure.

```
mycount<-mtcars%>%
  ggplot(aes(x=cyl,y=carb))+
  geom_count()+
  ylab("Carburators")+
  xlab("Cylinders")+
  labs(title="Count Chart", fill=NULL)+
  Mytheme
mycount
```

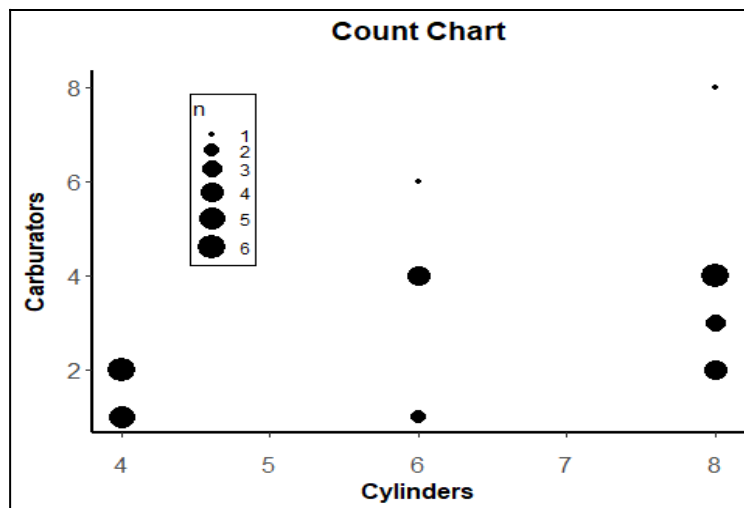


Figure 3. 12: Count Chart

The point sizes clearly convey the frequency of each cylinder–carburetor combination. Larger circles at positions like 8 cylinders with 4 carburetors indicate high recurrence, while smaller circles mark rare pairings. This layout makes it easy to spot dominant configurations and detect unusual or less common setups without needing explicit numeric labels.

3.12 Percentage Count

When you want to observe how often categories or values occur relative to the whole, a percent count chart provides a straightforward summary. It transforms raw frequencies into percentages, making it easy to compare proportions across groups of varying sizes. In terms of distributional role, it highlights relative frequency distributions, allowing you to quickly identify dominant categories and assess balance or imbalance in the data.

Setting `after_stat=1` allows you essentially to ask `ggplot` to treat the chart as a percentage-based view. Regardless of how many data points you have, this setting makes sure that all the dots combined add up to 100%. Think of it as distributing your data proportionally across the entire plot. So, each dot does not just represent a count but a fraction of the whole. This is really helpful when you want to show the overall distribution without focusing on absolute frequencies.

```
mycount_hun<-mtcars%>%
  ggplot(aes(x=cyl,y=carb))+
  geom_count(aes(size = after_stat(prop),group = 1))+ # points inside add up to 100
  scale_size_area(max_size = 5)+
  ylab("Carburators")+
  xlab("Cylinders")+
  labs(title="100% Count Chart ",fill=NULL)+
  Mytheme
mycount_hun
```

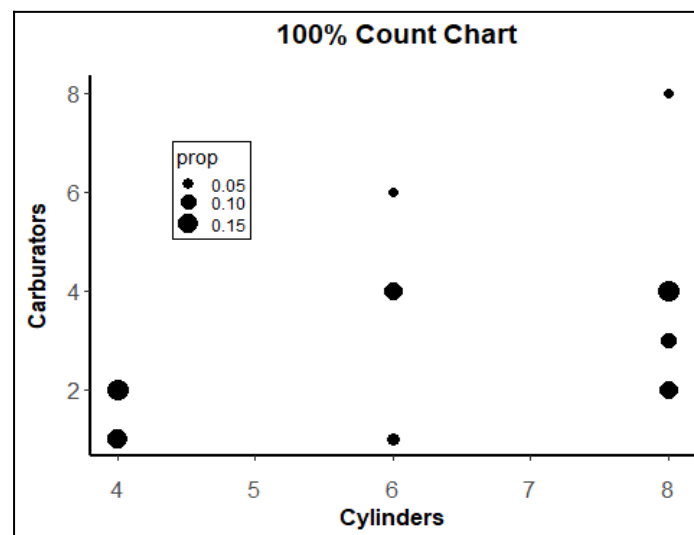


Figure 3. 13: Percent Count Chart

In Figure 3.13, it shows how cars in the `mtcars` dataset are distributed by cylinders (x-axis) and carburetors (y-axis). Each bubble represents cars that share the same cylinder–carburetor combination, and the bubble size reflects the proportion of the total dataset (all bubbles add up to 100%). For example, the largest bubble is at 8 cylinders with 4 carburetors, representing 10 out of 32 cars, that’s about 31% of the dataset. Another prominent bubble is at 4 cylinders with 1 carburetor, which corresponds to 7 cars (22%). Together, these two configurations alone account for more than half the cars in the dataset.

In contrast, some combinations are much rarer: only 1 car (3%) has 6 cylinders with 6 carburetors, and just 2 cars (6%) have 4 cylinders with 2 carburetors. These appear as noticeably smaller bubbles, making the rarity immediately visible without scanning a frequency table.

3.13 Vertical Count

The simplest way to understand data is by counting and stacking. A vertical count chart represents frequencies with bars aligned vertically, each bar's height showing how often a value or category appears. This makes it a practical tool for observing absolute distributions, spotting peaks where data cluster, and noticing rare categories with low counts.

Setting `group()` to `cyl`, which refers to the number of cylinders, tells `ggplot` to calculate proportions within each group of cylinders. As a result, each vertical column in the plot, representing a different cylinder count, adds up to 100%. It's like saying, "Let's look at the makeup of cars with 4, 6, or 8 cylinders, and see how those are broken down across another variable like carb." This setup is great for comparing categories within each cylinder type while maintaining consistent vertical proportions.

```
mycount_col<-mtcars%>%  
  ggplot(aes(x=cyl,y=carb))+  
  geom_count(aes(size = after_stat(prop),group = cyl))+ # vertical points sum to 100  
  scale_size_area(max_size = 5)+  
  ylab("Carburators")+  
  xlab("Cylinders")+labs(title="Vertical Count",fill=NULL)+  
  Mytheme  
mycount_col
```

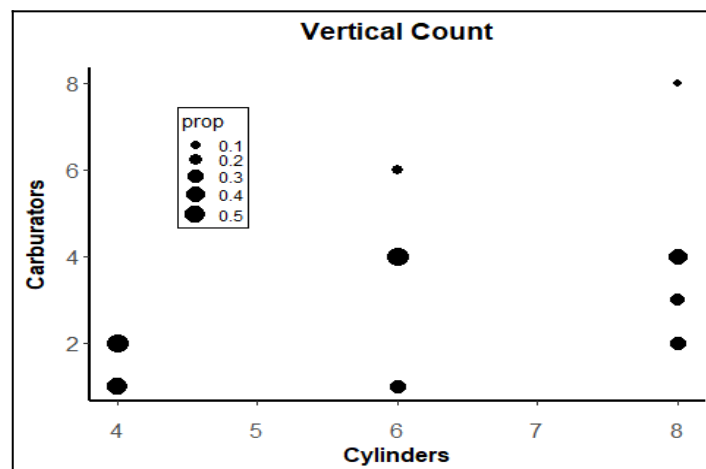


Figure 3. 14: Vertical Count Chart

3.14 Horizontal Count

A horizontal count chart is a simple yet effective way to display how frequently different categories or values appear in your dataset. If you're looking for a clear method to visualize how often each category shows up, you can use this chart to immediately see which values dominate

and which are less common. Instead of stacking the counts vertically, the bars are aligned horizontally, making it easier to read long category labels and compare counts across groups. This chart is especially useful when working with categorical data that has lengthy names, as it improves readability while still conveying the distribution of frequencies.

Now, in this case, if you set `group()` to `carb` (for carburetors), `ggplot` switches focus to horizontal grouping. Each row that representing a specific number of carburetors now adds up to 100%. In this case, you're examining how cylinder types (`cyl`) are distributed within each carburetor group. It's like asking, "For cars with 2 carburetors, what's the breakdown of cylinder types?" The dots line up across rows, helping you see how different groups compare within each row.

```
mycount_row<-mtcars%>%
  ggplot(aes(x=cyl,y=carb))+
  geom_count(aes(size = after_stat(prop),group = carb))+
  scale_size_area(max_size = 5)+
  ylab("Carburators")+
  xlab("Cylinders")+
  labs(title="Horizontal Count",fill=NULL)+
  Mytheme
mycount_row
```

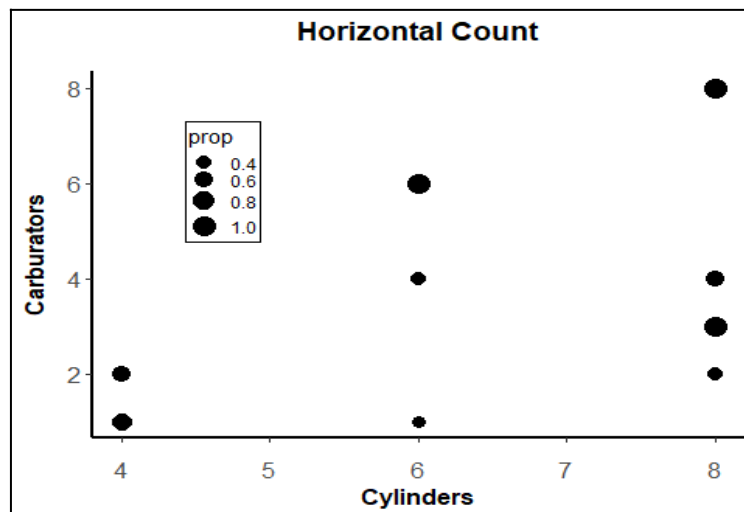


Figure 3. 15: Horizontal Count Chart

What this figure clearly shows is that the size of each point reflects how frequently a specific cylinder and carburetor combination appears in the dataset. Larger circles, such as the pairing of 8 cylinders with 2 carburetors and 4 cylinders with 1 carburetor, represent the most common configurations. In contrast, smaller circles mark rare combinations, for example 6 cylinders with

8 carburetors. This visual arrangement allows the viewer to immediately identify the dominant setups while also spotting uncommon ones without relying on explicit numeric labels.

3.15 Quantile Graph

It is agreeable that most datasets we often use do not behave as you would expect, especially in relation to normal distributions. A Q-Q plot compares your data's quantiles to a theoretical distribution, helping you see deviations from expected patterns. It's a quick way to assess normality or detect heavy tails in your data.

This geom adds quantile regression lines to a plot, which can help you understand the distribution of your data beyond just the average. Unlike a typical trend line that shows the mean, like `geom_smooth()`, `geom_quantile()` can show medians or other percentiles. It's especially helpful when you suspect your data has skewed patterns or heteroscedasticity (uneven spread). It reveals how trends vary at different levels of your response variable.

```
myqline3<-data.frame(mpg=rep(mtcars$mpg,100),hp=rep(mtcars$hp,100))
myqline3<-myqline3%>%
  ggplot(aes(hp,mpg))+
  geom_point()+
  geom_quantile(linewidth=1,color="blue")+
  ylab("Miles")+
  xlab("Horsepower")+ labs(title="Quantile Plot",fill=NULL)
Mytheme
myqline3
```

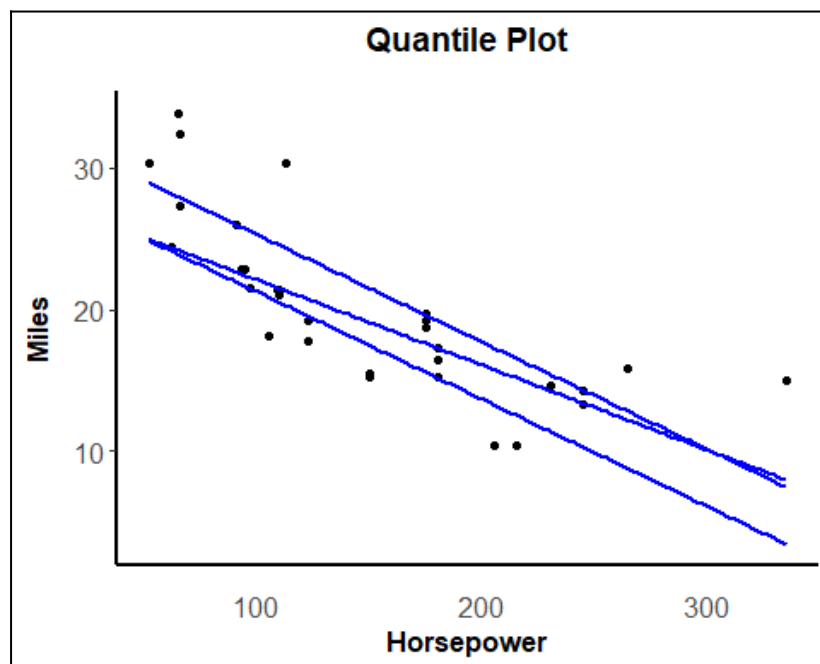


Figure 3. 16: *Quantile Plot*

As can be seen in the figure, the scatter of points represents the relationship between horsepower and miles per gallon across the expanded dataset. The smooth blue quantile lines trace different conditional medians, making it easy to see how fuel efficiency changes across the horsepower range. Lower quantile lines capture vehicles with generally lower mileage at given horsepower levels, particularly those around 175, 180, 190, and above 200 horsepower. Upper quantile lines reveal those that achieve better-than-average mileage, often found among vehicles with horsepower in the range of 90 to 120. This structure helps compare performance across the entire distribution, not just at the average.

3.16 Tertile Plot

It is important to recognize that data is often divided into equal-sized groups to make comparisons easier and to reveal hidden patterns. Just as quartiles split data into four parts and percentiles into one hundred parts, tertiles provide a simpler three-way division that balances detail with clarity. A tertile plot is a graphical representation that divides a dataset into three equal parts (tertiles) based on the distribution of values. Each tertile contains one-third of the observations, making it useful for identifying low, medium, and high categories within the data. This visualization highlights differences across the three groups and is often applied in fields such as economics, health research, and social sciences where comparing “bottom,” “middle,” and “top” segments is meaningful.

You can also use `geom_quantile()` to divide your data into three equal-sized portions, often known as tertiles. It is especially useful when you want to visualize the spread of your data across percentiles rather than just using the mean or median. For example, it can highlight how the lower third, middle third, and upper third of your dataset behave differently. In a scatterplot, `geom_quantile()` fits smooth curves that represent these quantiles, giving a richer picture of the distribution trends. It's great for spotting skewness or detecting unusual patterns in different segments of your data.

```
myqline2<-data.frame(mpg=rep(mtcars$mpg, 50), hp=rep(mtcars$hp, 50))
myql2<-myqline2%>%
  ggplot(aes(hp, mpg))+
  geom_point()+
  geom_quantile(quantiles=c(1/3, 2/3), linewidth=1, color="blue")+
  ylab("Miles")+
  xlab("Horsepower")+
```

```
labs(title="Tertile Plot", fill=NULL)+ Mytheme
myql2
```

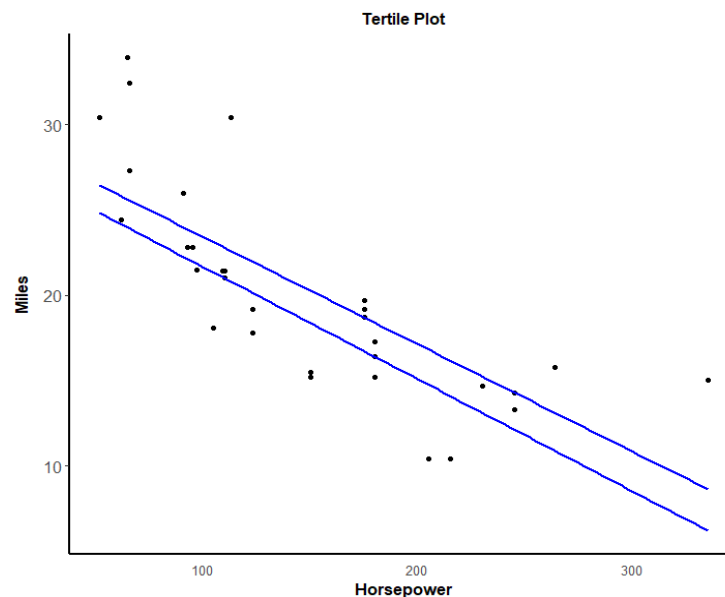


Figure 3. 17: Tertile Plot

In Figure 3.17 above, it clearly shows the spread of miles per gallon across different horsepower levels, with two blue quantile lines dividing the data into lower, middle, and upper thirds. The lower quantile line captures vehicles with consistently low mileage at higher horsepower values, especially those around 175, 180, 190, 200, 215, and 230 horsepower. The upper quantile line highlights vehicles that achieve above-average mileage, most common among horsepower values in the 90, 95, 100, 110, and 120 range. This approach reveals not just the central trend but the differences in performance across the full horsepower spectrum.

3.17 Quantile-Quantile

When you need deeper insight into how values are spread across a dataset, a Quantile Chart comes into play. Quantile-quantile plot is like a lie detector for your data's distribution. It partitions the data into equal-sized groups (quantiles), displaying where cut-points like quartiles, deciles, or percentiles fall. Its distributional role lies in showing how values accumulate, whether data are concentrated in certain ranges, and how balanced or skewed the distribution is, offering a nuanced perspective beyond simple averages. If your points form a straight diagonal line, you're in the clear, your data likely follows the distribution. In ggplot2 environment, `geom_qq()`

is used to draw this plot, and `geom_qq_line()` to add the reference line. It's especially useful when you're checking assumptions before doing statistical tests like t-tests or ANOVA.

```
df_qq<-data.frame(y=stats::rt(300,df=7))
myqq<-df_qq%>%
  ggplot(aes(sample=y))+
  geom_qq()+
  ylab("Statistics")+
  xlab("Distribution")+
  labs(title="qq Plot ",fill=NULL)+
  Mytheme
myqq
```

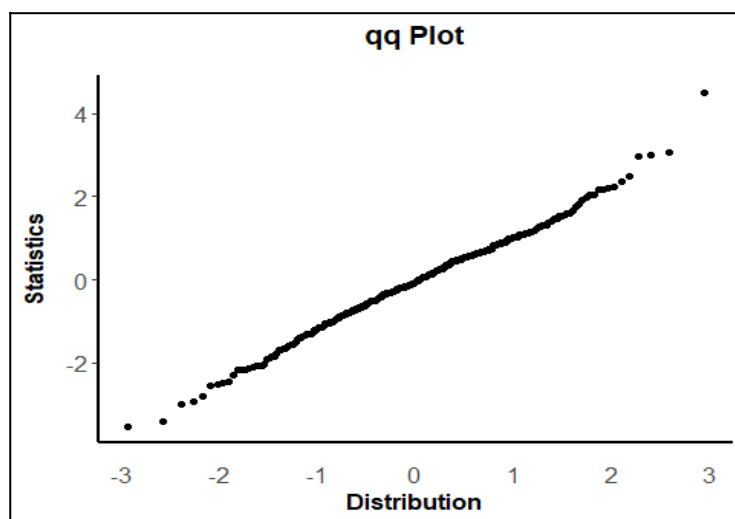


Figure 3. 18: Quantile-Quantile Plot

In a glimpse, the QQ plot enables you to compare the sample data against a theoretical distribution and to check for normality. The points closely follow the diagonal in the middle section, indicating that most of the data aligns well with the expected distribution. However, noticeable deviations occur at both ends, with extreme values in the tails straying above and below the reference line. This pattern suggests heavier tails than a normal distribution, which is a common feature of data generated from a t-distribution with 7 degrees of freedom.

3.18 Quantile-Quantile Line

To make patterns in a Q-Q plot even clearer, a Q-Q line chart connects the quantile points with lines. This lets you see trends in deviation more easily, especially in large datasets, making comparisons smoother and more intuitive. In Q-Q (quantile-quantile) plots, `geom_qq_line()` adds a reference line that assists in evaluating how closely your data matches a theoretical

distribution, such as the normal distribution. If your points fall along the line, your data is likely a good fit for that distribution. It's often used in conjunction with `geom_qq()` to visually check assumptions for statistical models. This `geom` makes it much easier to interpret Q-Q plots at a glance, especially when validating normality.

```
myqline<-myqqline%>%  
  ggplot(aes(sample=mpg))+  
  geom_qq_line(color="yellow")+  
  geom_qq()+  
  ylab("Miles")+  
  xlab("Distribution")+  
  labs(title="qq Line Plot ",fill=NULL)+  
  Mytheme  
myqline
```

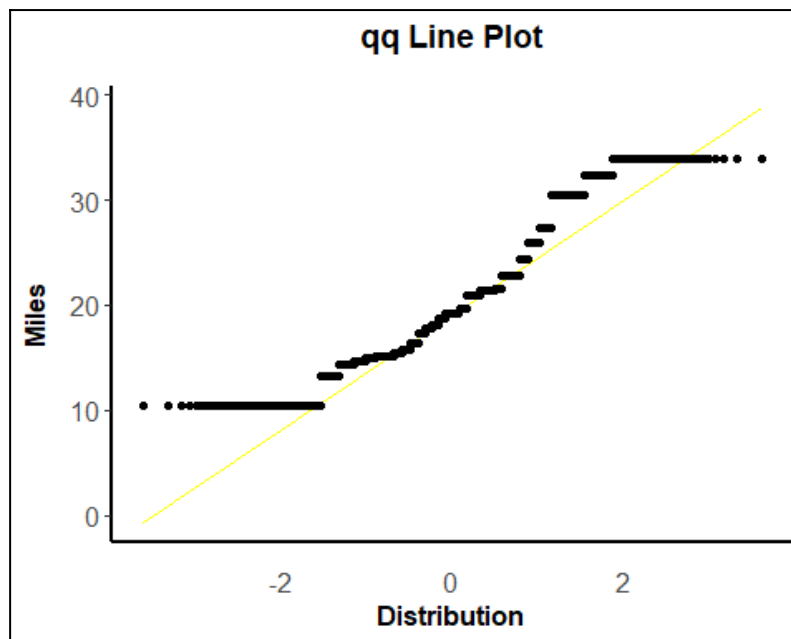


Figure 3. 19: Quantile-Quantile Line Plot

As can be seen in the figure, the QQ line acts as a visual benchmark for assessing the distribution of miles per gallon. The combination of plotted points and the guiding yellow line makes it easy to spot departures from normality, offering both precision and simplicity in one chart. The plotted points mostly cluster along the yellow reference line, indicating that a large portion of the **MPG** (miles per gallon) values follow the expected theoretical distribution. Slight upward deviations at the upper end and downward bends at the lower end hint at departures in the extreme values, suggesting mild tail differences compared to a perfect normal fit. Points lying

close to the line suggest the data aligns well with normality in those regions, as seen in the mid-range where the fit is strong. Deviations at the upper and lower ends indicate potential skewness, heavier or lighter tails compared to the normal distribution.

3.19 Text Chart

Sometimes numbers alone tell the story best. A text chart displays exact values on a plot, often combined with other visual elements. This allows you to see precise measurements while still observing overall trends, making your data instantly interpretable.

If you want to label points directly on your plot, `geom_text()` is your buddy tool. It allows you to add text annotations, which can be useful for identifying specific observations, showing values, or adding commentary. It is useful when you want to label things directly on your plot. You can control the position, size, and color of the text, and even rotate it for better readability. Just be careful not to clutter your plot, used sparingly, text labels can greatly enhance clarity and impact.

```
mytext<-mytreez%>%
  ggplot(aes(x=Volume,y=Height, colour = Girth,label=Girth))+
  geom_point()+
  geom_text(hjust=-.1,vjust=.1)+
  scale_color_manual(values = c("brown","orange"))+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  ylab("Height")+
  xlab("Volume")+
  labs(title="Text Chart",fill=NULL)+
  theme_mytreez
mytext
```

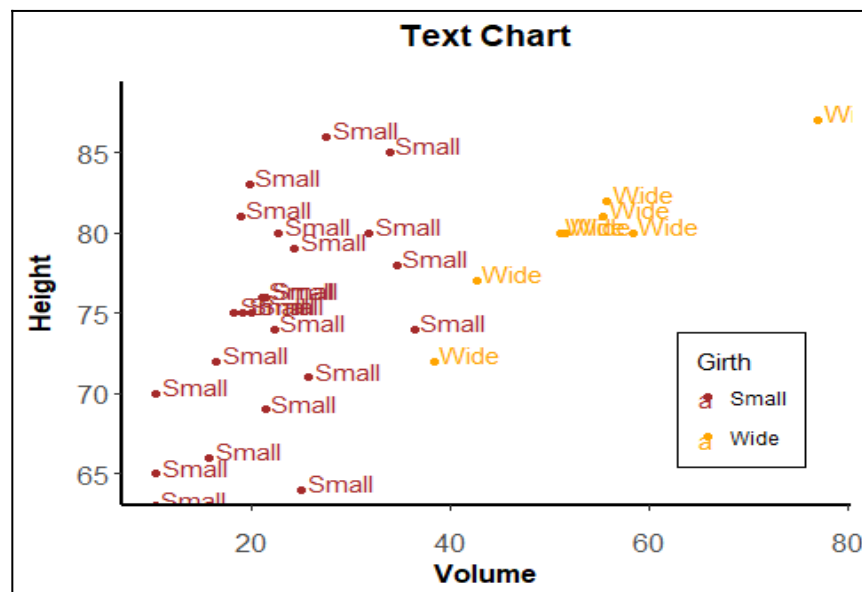


Figure 3. 20: Text Chart Plot

From the graph, you see that each point is not only positioned by its Volume and Height but also directly annotated with its corresponding Girth value, making it easier to identify individual trees without cross-referencing a legend. The brown and orange colors further separate the groups visually, while the slight offsets in text placement prevent overlap with the data points for better readability. This text chart presents tree Height against Volume, with each point annotated by its corresponding Girth value. The brown and orange color coding makes it easy to distinguish between girth categories, adding visual grouping alongside the numeric labels. The point markers provide exact positioning, while the adjacent text labels allow quick identification without referring to a legend. By slightly offsetting the labels horizontally and vertically, overlap is minimized, keeping the chart readable even where points are close together. The result is an informative blend of quantitative plotting and direct labeling, giving both spatial and categorical context at a glance.

3.20 Text Repel

In data visualization, labels are critical for interpretation, but when many points are plotted in the same space, tags often overlap, making the chart cluttered and unreadable. The function `geom_text_repel()` (from the `ggrepel` package) provides a professional solution to this problem. Instead of allowing labels to stack or collide, `geom_text_repel()` intelligently nudges text labels away from each other and from data points, while maintaining connection lines (if necessary) to the corresponding points. This results in labels that are legible, evenly spaced, and minimally intrusive to the data display. It's ideal when you have lots of labels that tend to collide.

```
mytext_rep<-mytreez%>%
  ggplot(aes(x=Volume,y=Height, colour = Girth,label=Girth))+
  geom_point()+
  geom_text(hjust=-.1,vjust=.1)+
  ggrepel::geom_label_repel(size=1)+
  scale_color_manual(values = c("brown","orange"))+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  ylab("Height")+
  xlab("Volume")+
  labs(title="Text Chart",fill=NULL)+
Mytheme
mytext_rep
```

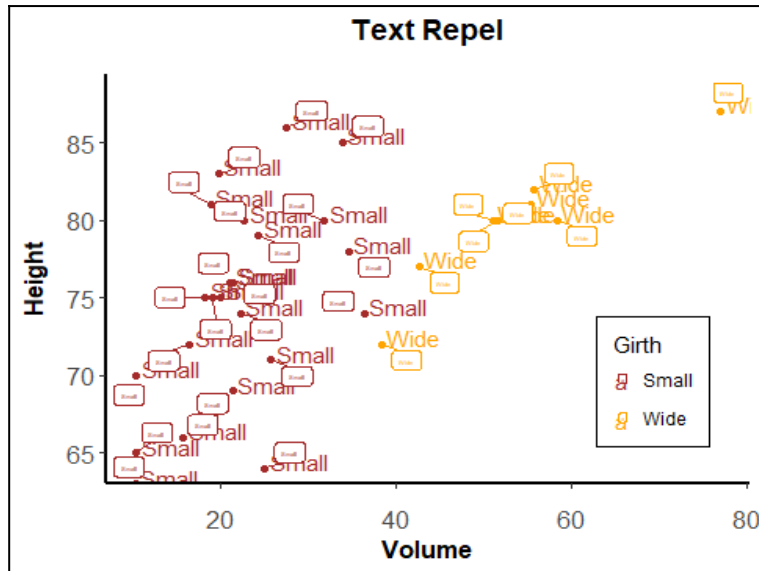


Figure 3. 21: *Text Repel*

This enhanced text chart maps **Height** against **Volume** while labeling each point with its **Girth** value. The brown and orange colors clearly separate the girth categories, while the small label size keeps the plot uncluttered. The combination of precise point markers and smart label placement creates a chart that is both visually appealing and highly legible, making it easy to interpret individual data points without sacrificing overall clarity.

CHAPTER 4: COMPOSITION

Data which is based on proportions or percentages illustrate the parts which constitute the whole. In compositional analyses, the parts basically sum up to 100 if expressed in percentages or sum up to one if expressed as a proportion. It is also noteworthy that compositional data need standardization techniques such as log transformations before they are directly employed in analysis. However, such data structures can be analyzed directly using graphic tools such as pie charts, donut charts, stacked bar graphs, treemaps, sunburst charts and waterfall diagrams.

Composition charts show how a whole is divided into parts and how those parts relate to each other. They are ideal for illustrating proportions and comparing the size of components within a dataset. These visualizations are often used in finance, marketing, and resource allocation. Pie charts offer a simple circular view of part-to-whole relationships, though best used with few categories. Donut charts are a variation of pie charts with a central blank space for labels or totals. Stacked bar charts show parts within bars and allow for time-based or group comparisons. Treemaps use nested rectangles to display proportions in a compact format, useful for hierarchical data. These visualizations effectively communicate the structure and makeup of a dataset.

4.1 Pie Chart

A circle has long been a natural symbol of wholeness. When it is divided into slices, each portion immediately shows its share of the total. This makes it interesting and powerful to see at a glance how a complete whole is broken down into proportional parts. A pie chart uses this very idea, turning data into slices of a circle so that differences in size are instantly clear to the eye. It is divided into slices, each representing a proportion of the whole, with the arc length of each slice proportional to the value it depicts. These slices vary in size depending on the data they represent, making it easy to see the relative importance of each category. Pie charts work particularly well when presenting findings to an audience because they are simple, direct, and visually engaging, often making a stronger impression than bar charts. They communicate effectively even to audiences unfamiliar with the data, allowing quick comparisons at a glance. Slices can also be highlighted or separated to emphasize specific points of interest. They can display multiple categories in one view, require minimal explanation, and save readers from

estimating values themselves. Their visual appeal often draws more attention than other chart types.

However, unlike bar or line graphs, pie charts are not suited for showing changes over time, making them less appropriate for time series data. They work best with a small number of categories and lose effectiveness when too many slices are added. Pie charts also do not reveal exact values, and subtle differences between slice sizes can be misleading or difficult to compare, especially when viewers must mentally judge angles. They cannot show both positive and negative values in a single chart, which limits their use in certain data sets.

Based on their graphic style, pie charts are generally classified as either two-dimensional (2D) or three-dimensional (3D). A 2D pie chart is the simpler form, showing data proportions in a flat circular view. Within the 2D category, there are several variations depending on how the data is presented: the simple pie chart, the exploded pie chart, the pie of pie, and the bar of pie. Each variation serves specific purposes, from highlighting certain segments to breaking down complex categories for clearer interpretation.

4.2 Simple Pie Chart

A simple pie chart slices the dataset into wedges representing each category, with the **size of** each wedge proportional to its value. This makes it easy to see which segments dominate and which are minor contributors, providing a clear overview of composition.

A basic pie chart, like the one made with `geom_rect()` and `coord_polar()`, is a classic and effective way to show proportions in a two-dimensional view. It displays the contribution of each part to a whole, with fewer variables offering cleaner insight. By mapping values to the fill aesthetic (such as tree girth), the chart becomes a quick tool for comparison. Labels positioned using `geom_text()` make interpretation straightforward, and the use of `theme_void()` removes visual clutter. This setup reduces perspective distortion, making it easier to grasp than complex 3D charts. Adding color with `scale_fill_manual()` further clarifies group differences.

```
mytrees <- trees %>%
  arrange(desc(Girth)) %>%
  mutate(
    fraction = Count / sum(Count),
    ymax = cumsum(fraction),
    ymin = c(0, head(ymax, n = -1)),
    label_pos = (ymax + ymin) / 2,
    label_text = paste0(Count, "\n", round(fraction * 100), "%")
  )
ggplot(mytrees, aes(ymax = ymax, ymin = ymin, xmax = 4, xmin = 3, fill = Girth)) +
```

```
geom_rect(color = "black") +
coord_polar(theta = "y") +
theme_void() +
geom_text(aes(x = 3.5, y = label_pos, label = label_text), size = 3) +
scale_fill_manual(values = c("pink", "yellow", "grey"))+
ggtitle("Pie Chart")+ Mytheme
```

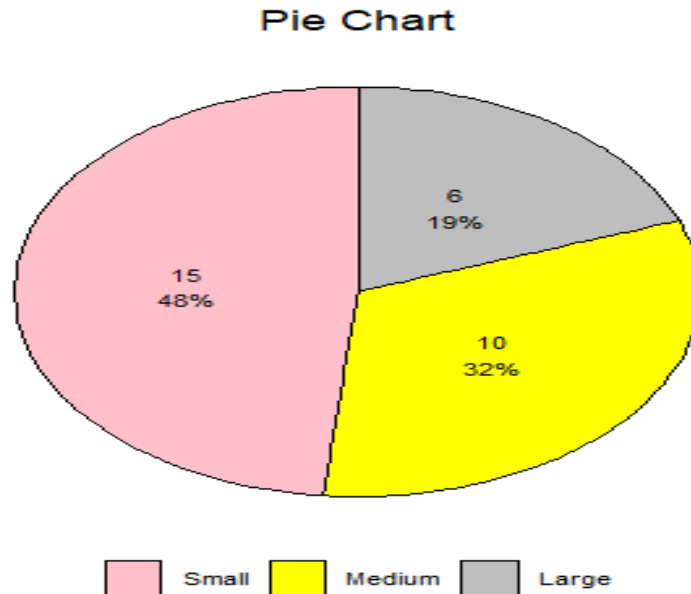


Figure 4. 1: Simple Pie Chart

Each colored slice in the chart corresponds to a category, and the size of the slice reflects its proportion in the whole dataset. For instance, the **Small** girth category is the gentle pink section that glows with prominence occupying the largest segment of the pie, which matches its count of 15 and represents about 48% of the total. The **Medium** category is a warm yellow segment with modest slice size, offering a softer but still notable presence covering a substantial portion, accounting for 10 counts or roughly 32% of the total. Lastly, the **Large** category is quiet grey portion and the smallest slice with a count of 6, making up around 19% of the data, revealing that **Large** trees are the rarest of the three groups.

4.3 Pie with White Labels

Sometimes the numbers need to be visible on top of the chart itself. A pie chart with white labels overlays numeric or percentage labels directly on each slice, making it immediately obvious how large each segment is relative to the whole without needing to refer to a legend. To create a pie chart with white labels, we first categorize Girth from the trees dataset into three levels: Small, Medium, and Large. Then we count the number of entries in each category and calculate their percentage contribution. Using `ggplot()`, we define rectangle slices with `geom_rect()` and use

`coord_polar(theta = "y")` to transform the bar chart into a pie chart. We calculate label positions (`label_pos`) and contents (`label_text`) showing both count and percentage. To display these clearly, `geom_label()` is used with a white fill for label backgrounds and `label.padding()` to adjust size. The pie chart is styled using `theme_void()` to remove axes and grid lines, and filled with custom colors via `scale_fill_manual()`. This approach enhances readability and ensures the labels stand out cleanly against the colored slices.

```
mytrees <- trees %>%
  mutate(Girth = cut(Girth,
                     breaks = 3,
                     labels = c("Small", "Medium", "Large")))
mytrees<-mytrees%>%
  group_by(Girth) %>%
  summarise(Count = n())
mytrees<-mytrees[order(mytrees$Count,
                      decreasing = T ),]
mytrees$percent<-round(mytrees$Count/sum(mytrees$Count)*100,1)
mytrees$lbl<-paste0(mytrees$Count, " (", mytrees$percent, "%)")

ggplot(mytrees, aes(ymax = ymax, ymin = ymin, xmax = 4, xmin = 3, fill = Girth)) +
  geom_rect(color = "black")+
  coord_polar(theta = "y")+
  theme_void()+ # removes background text
  geom_label(
    aes(x=3.5, y=label_pos, label = label_text),
    label.size = 0.2,
    fill = "white", # White background for label
    label.padding = unit(0.1, "lines")+ #adjusts label size
    scale_fill_manual(values = c("pink", "yellow", "grey")) +
    ggtitle(" White
Labels")")+
Mytheme
```

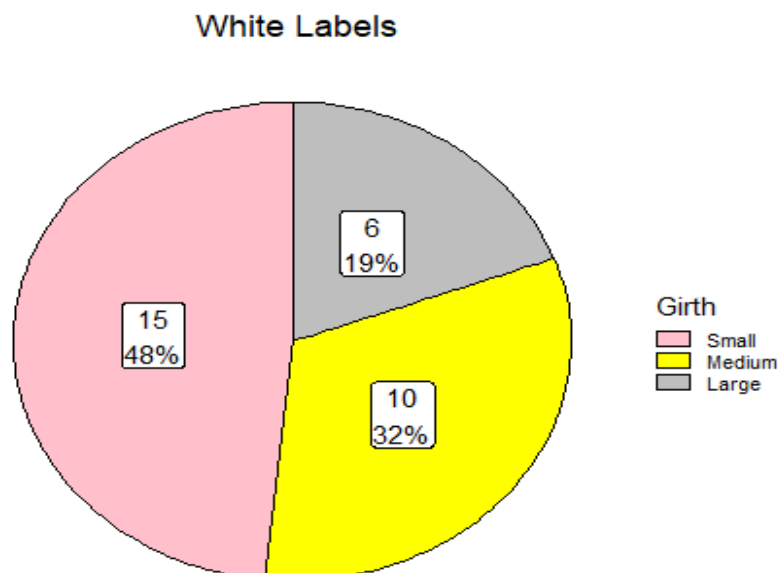


Figure 4. 2: Simple Pie with Labels

This pie chart clearly illustrates the composition of tree girth categories, with the count and percentage values prominently displayed inside white labels. The white background of these labels enhances readability against the colored slices, making the exact figures stand out sharply. The **Small** girth category dominates the chart with the biggest slice, accounting for 15 trees, which translates to about 48% of the total. The **Medium** category follows closely, representing 10 trees or 32%, while the **Large** category forms the smallest segment with 6 trees, contributing about 19% to the overall composition. These white labels with combined counts and percentages allow for quick and accurate compositional analysis, enabling you to easily compare the relative sizes of the groups. The visual separation and clear labeling make it straightforward to understand which categories contribute most to the dataset and by how much, without any ambiguity. The labels, showing both counts and percentages, make it easy for the viewer to read the story without guessing, turning raw numbers into a picture that feels immediate and memorable.

4.4 Pie with Polygonal Patterns

When colors fall short, polygonal patterns step in to make pie charts more distinctive. Textures and hatch fills on slices add an extra visual cue, ensuring categories stand out clearly even in grayscale or print. This not only boosts clarity but also brings a creative edge to the chart's design.

Instead of using solid colors, you can use polygon patterns (like hexagonal or triangular) to distinguish sections in a pie chart. This is helpful when printing in black and white or when color accessibility is an issue. Using `geom_col_pattern()` with `pattern_type()` and `pattern_fill()` mapped to data features creates distinct segments filled with textures rather than colors. By adjusting `pattern_density()` and spacing, you create clearer visual differentiation. Labels placed with `geom_label()` ensure that proportions remain understandable. The result is a unique pie chart that offers a textural way of comparing parts-to-whole.

```
ggplot(mytrees, aes(x="",  
                    y=percent,  
                    pattern_type=Girth,  
                    pattern_fill=Girth,  
                    label=label)) +  
  geom_col_pattern(fill="white", # shading or background color  
                  color="black", # for rectangle borders  
                  pattern_density= 2, #adjust thickness/size of the pattern
```

```

        pattern="polygon_tiling",
        position = "stack")+
geom_label(position = position_stack(vjust = .5 ),
          color="black", # label border
          size = 3)+ # adjust label size
scale_pattern_type_discrete(choices = c("hexagonal", "triangular", "pythagorean"))+
scale_pattern_fill_manual(values = c("lightgrey", "grey", "white"))+
coord_polar(theta = "y")+
scale_pattern_spacing_discrete(range = c(0.02, 0.07))+ # adjusting space between
patterns
ggtitle(" A Pie Chart")+
Mytheme

```

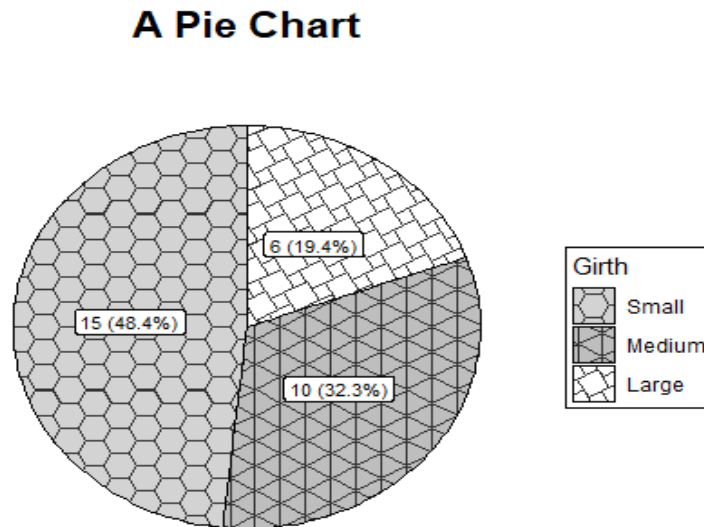


Figure 4. 3: Pie Chart with Polygonal Patterns

The chart takes the familiar charm of a pie chart and gives it a textured twist. Instead of solid colors, each slice is decorated with a distinctive pattern, hexagonal, triangular, or Pythagorean, making it easy for the eye to tell them apart, even without relying on color alone. The light grey, darker grey, and white tones give it a clean, almost minimalist feel, while the bold black outlines keep each section neatly framed. In the center, the hollow “donut” space gives the chart a modern touch, and the crisp white labels resting neatly inside each slice share both the counts and percentages. It’s the kind of design that works well when you want to highlight composition in a visually engaging way while keeping things stylish and accessible.

4.5 3D Pie Chart

One reason why we use charts is to grab attention, and a 3D pie chart does exactly that. By giving slices a lifted, dimensional look, it adds visual impact without altering the data. This makes proportions more eye-catching and helps presentations feel dynamic and engaging. A traditional 3D pie chart displays the proportions of tree girth categories without separating the

segments. Using the `plotrix` package, we apply `pie3D()` and set `explode = 0` to keep the segments together. The labels combine counts and percentages for clarity. A fixed color scheme enhances group distinction. This type of chart is useful for summarizing categorical data visually, while maintaining the integrity of proportions in a more engaging style than flat pie charts.

```
library(plotrix)

pie3D(mytrees$Count,
      labels = mytrees$labl,
      explode = 0, # no separation
      labelcex = 1, # label size
      main = "Simple 3D Pie Chart",
      col = c("pink", "yellow", "grey"))
```

Simple 3D Pie Chart

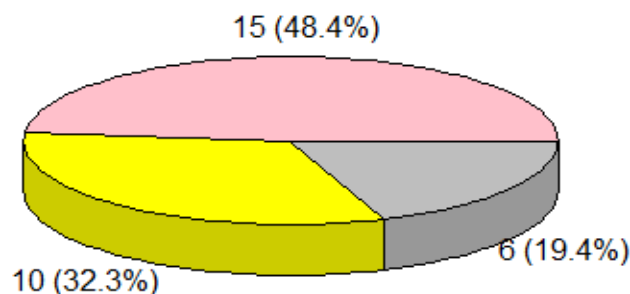


Figure 4. 4: Simple 3D Pie Chart

This one keeps things straightforward yet adds a hint of depth with its gentle 3D effect. The slices for Small, Medium, and Large stand out in pink, yellow, and grey, giving a pleasant contrast without feeling overwhelming. The labels sit right on the slices, showing both the counts and percentages so viewers can grasp the proportions instantly. With no separation between slices, the chart feels cohesive, almost like a single, smooth disk. It's the kind of visual that works nicely for a clean presentation where you want the focus on the proportions themselves, with just enough style to catch the eye.

4.6 Exploded Pie Chart

Occasionally a single slice deserves the spotlight. An exploded pie chart pulls specific sections away from the circle, instantly highlighting their significance. By breaking the uniformity, it makes comparisons clearer and turns the chart into more of a narrative than just a graphic. An

exploded pie chart separates one or more slices from the whole to draw attention to specific segments. Using `pie3D()` with the `explode` parameter set to `0.1` visually pulls out a slice, emphasizing its importance. Label sizes can be customized with `labelcex()`, and the 3D view enhances visibility. Alternatively, when working in `ggplot2()`, setting `pie3D(fill=white)` in `geom_label()` ensures that text labels remain clear even on patterned or colored backgrounds. These techniques both support clarity and direct focus within the visualization, allowing critical values to stand out without overwhelming the viewer.

```
pie3D(x=mytrees$percent, labels=mytrees$labl, main="3D Pie Chart",  
col=c("lightgrey", "yellow", "pink"),  
explode=0.1,  
labelcex = 0.8, #adjust labels  
radius = 2)
```

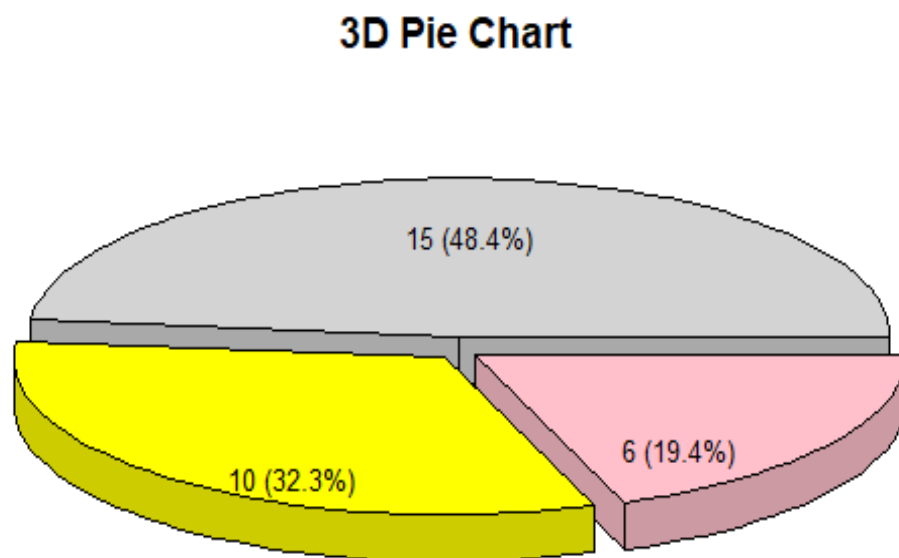


Figure 4. 5: Exploded Pie Chart

Here the chart takes on a livelier personality thanks to the slight separation between slices. The light grey, yellow, and pink sections gently pull away from the center, giving each category its own space while still feeling part of the whole. The labels are neatly tucked onto the slices, showing the counts and percentages clearly without overpowering the colors. The larger radius makes the pie feel more expansive, and the subtle 3D tilt adds a touch of dimensionality that draws the viewer in. It's a friendly, easy-to-read display that invites a quick understanding of the composition while keeping things visually engaging.

4.7 Donut Pie Chart

Imagine of a chart that leaves a little room for creativity. That's the idea behind a donut chart: a pie chart with a hole in the middle. The open space isn't wasted, it's often used to show totals, display icons, or highlight a key fact that ties the whole story together. A doughnut chart is essentially a pie chart with a hole in the center, offering a modern look while still presenting parts-to-whole. The hole is achieved by setting `xlim(c(2, 4))` which pushes the rectangles outward and leaves the middle empty. It's especially effective for dashboards, offering better space usage than traditional pies. Like in pie charts, `geom_label()` can be used for labeling, while `scale_fill_manual()` controls the color scheme. This format works well when you want to compare segments but prefer a cleaner center space, perhaps for placing a total or an icon.

```
ggplot(mytrees, aes(ymax = ymax, ymin = ymin, xmax = 4, xmin = 3, fill = Girth)) +  
  geom_rect(color = "black")+  
  coord_polar(theta = "y")+  
  xlim(c(2, 4)) +  
  theme_void()+ # removes background text  
  geom_label(  
    aes(x=3.5, y=label_pos, label = label_text),  
    label.size = 0.2,  
    fill = "white", # White background for label  
    label.padding = unit(0.1, "lines")+ #adjusts label size  
    scale_fill_manual(values = c("pink", "yellow", "grey")) +  
  ggtitle(" Donut Chart")+Mytheme
```

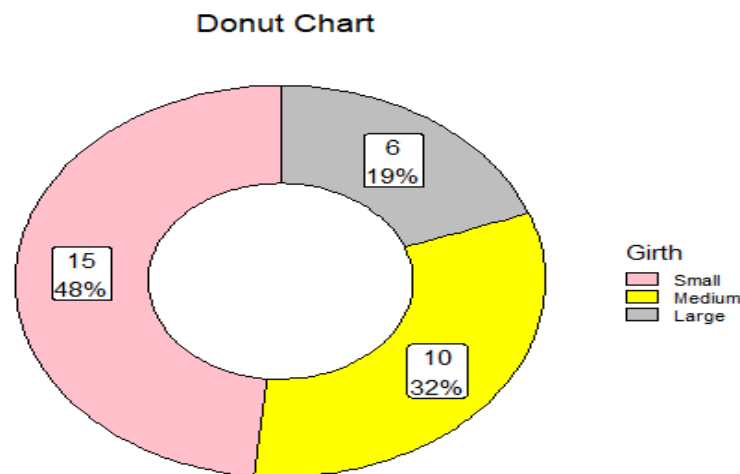


Figure 4. 6: Donut Pie Chart

The donut shape opens up the center of the pie, creating a clean ring where the eye can rest while the colors do the talking. Pink, yellow, and grey wrap neatly around the circle, each representing its size category with a pleasant balance of contrast. The white-backed labels float gracefully on top, making the counts and percentages easy to read without clashing with the colors beneath.

The black slice borders subtly guide the eye, keeping the sections distinct. It's a polished and friendly way to show proportions while giving the chart a lighter, more modern feel than a solid pie.

4.8 Pie of Pie Chart

When some categories are too small to read clearly in a regular pie chart, a Pie of Pie or Bar of Pie splits these minor categories into a secondary pie or bar chart. This preserves the overall composition while allowing small segments to be readable and meaningful. In the event that you need to handle many categories without crowding a single pie, a Pie of Pie chart helps by breaking out less significant values into a secondary chart. Using the `PieDonut()` function, data is divided into an inner and outer pie, with a grouped donut layout. The outer segments, using `explodeDonut()`, emphasize sub-categories, while the `r0()` and `r1()` values control the inner and outer radii. The explode option also pulls out specific sections for emphasis. This technique makes it easier to explore categorical depth while keeping the visualization tidy and interactive.

```
PieDonut(data = titanic, aes(Class, Survived, count=n), # can alters the parent
          between class and survived
          ratioByGroup = TRUE, # to make outer% add up to 100
          explode = 4,
          explodeDonut = TRUE,
          title = "Titanic Survivors",
          r0=0.4,
          r1=0.8)
```

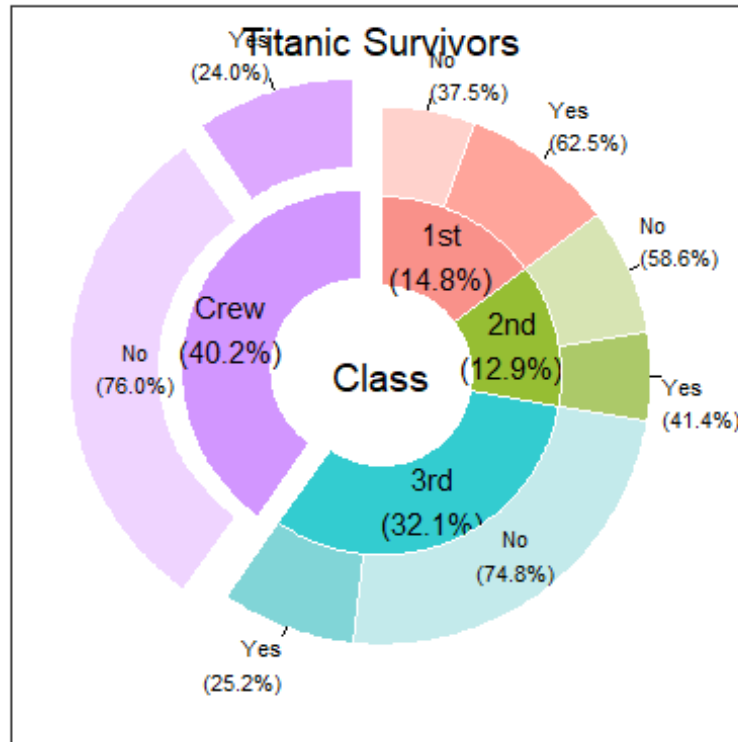


Figure 4. 7: Pie of Pie Chart

In this PieDonut chart, the inner ring represents the four passenger classes, First, Second, Third, and Crew, while the outer ring shows the share of survivors and non-survivors within each class. Each outer ring segment adds up to its class total, so you can quickly compare survival outcomes within a class without mixing it with others. The exploded layout and the clear link between inner and outer segments make it easy to trace how each class fared, with some showing a visibly larger survivor portion and others a dominant non-survivor share.

4.9 Bar of Pie Chart

A bar of pie chart offers an effective way to display additional categories without overcrowding the main pie chart. While both pie of pie and bar of pie charts aim to clarify category distribution, the bar of pie differs by extending the main pie with a stacked bar graph rather than another pie. It pulls certain segments from the original pie and presents them as a stacked bar, providing a clearer view of smaller or grouped categories through this bar extension. A threshold value (`threshold <- 10`) determines which groups are considered large enough to be shown individually in the main pie chart. Groups with counts below this threshold are combined into an `other` category to keep the pie readable.

The main pie chart is created using `geom_bar(stat = "identity", width = 1, color = "white")`, which draws bars representing the fractions of each category. The `coord_polar(theta = "y")` function transforms this bar chart into a circular pie chart by mapping the y-axis as the angle around the circle.

If smaller groups exist (below the threshold), a separate bar chart breaks down the `Other` category into its components using `geom_col()`, which makes vertical bars for each small category. The two plots are displayed side by side with `grid.arrange()`, allowing easy comparison between the main categories in the pie and the detailed breakdown of the smaller groups. The use of `theme_void()` removes background and axis elements from the pie chart to focus attention solely on the slices, while `scale_fill_manual()` assigns distinct colors to each category to aid visual differentiation.

```
library(gridExtra)
mytrees <- trees %>%
  mutate(Girth_cat = cut(Girth, breaks = 3, labels = c("Small", "Medium", "Large")))
%>%
  group_by(Girth_cat) %>%
  summarise(Count = n()) %>%
  arrange(desc(Count))
# Separate large categories and smaller ones for 'Other'
threshold <- 10 # define threshold count to separate small groups
main_pie <- mytrees %>% filter(Count >= threshold)
others <- mytrees %>% filter(Count < threshold)
# Pie chart for main categories + "Other" slice
pie_data <- main_pie %>%
  mutate(Girth_cat = as.character(Girth_cat))
if (nrow(others) > 0) {
  pie_data <- pie_data %>%
    bind_rows(data.frame(Girth_cat = "Other", Count = sum(others$Count)))
}
pie_data$fraction <- pie_data$Count / sum(pie_data$Count)
pie_plot <- ggplot(pie_data, aes(x = "", y = fraction, fill = Girth_cat)) +
  geom_bar(stat = "identity", width = 1, color = "white") +
  coord_polar(theta = "y") +
  theme_void() +
  ggtitle("Pie Chart - Main Categories") +
  scale_fill_manual(values = c("grey", "pink", "yellow", "lightblue"))
# Bar chart for 'Other' breakdown
if (nrow(others) > 0) {
  bar_plot <- ggplot(others, aes(x = Girth_cat, y = Count, fill = Girth_cat)) +
    geom_col() +
    ggtitle("Bar Chart - Breakdown of Other") +
    theme_minimal() +
    scale_fill_manual(values = c("lightgreen", "lightcoral", "lightcyan"))
} else {
  bar_plot <- NULL
}
# Arrange side by side
if (!is.null(bar_plot)) {
  grid.arrange(pie_plot, bar_plot, ncol = 2)
} else {
  print(pie_plot)
}
```

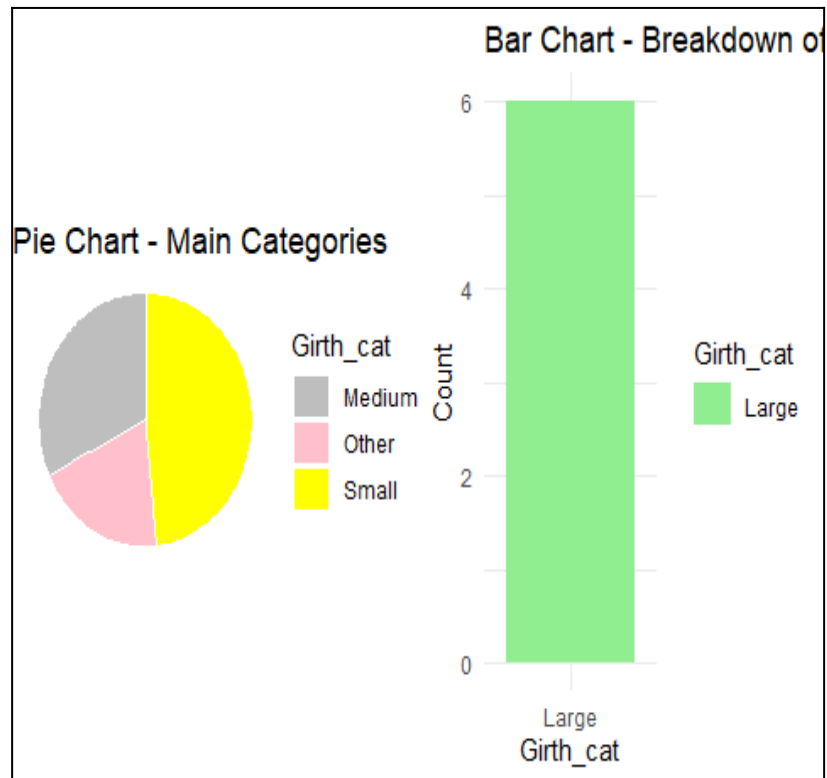


Figure 4. 8: Bar of Pie Chart

The pie chart visually represents the major groups, showing their relative proportions in the total. For example, in this chart, the **Small** girth category occupies nearly half of the pie, indicating it accounts for roughly 48% of the total volume. The adjacent bar chart breaks down the **Large** categories grouped as **Other**, allowing you to focus on those less dominant parts without cluttering the pie. From the bars, you can observe that the **Large** girth category contributes about 20% of the volume, while the **Medium** girth follows with approximately 32%. This split makes it easier to compare both the major contributors and the smaller ones, helping identify which groups dominate and which play a minor role. Overall, this visualization supports a deeper compositional analysis by separating the large, dominant category from the smaller ones, allowing for a more focused interpretation of each group's contribution to the whole.

4.10 Stacked Bar Chart

In many cases, Sometimes, it's easier to compare categories across groups than within a single pie. A stacked bar chart stacks the values of different categories on top of each other within a bar. This allows you to see both total values and individual contributions, making it easier to compare composition across groups. Using `geom_bar(stat = "identity")`, bars are divided into segments by groups (like Space), which are colored differently for clarity using `scale_fill_manual()`. Labels and axes are also customized for better readability, for instance with `scale_x_discrete()` providing interpretable group labels. These charts let you compare totals and examine internal composition, ideal when your data includes nested categories. They're especially helpful for detecting outliers and trends across grouped data.

```
ggplot(mytrees, aes(x = Girth, y = Height, fill = Space)) +  
  geom_bar(stat = "identity") +  
  xlab("Girth") +  
  ylab("Height") +  
  scale_y_continuous(expand = expansion(mult = c(0, 0.1))) +  
  scale_fill_manual(values=c("lightgreen",  
                             "lightblue", "pink")) +  
  scale_x_discrete(labels=c("Small"="Less\n than 20", "Medium"="Between\n 20-  
50", "Large"="More\n than 50")) +  
  labs(title="Stacked Chart ", fill=NULL) + Mytheme
```

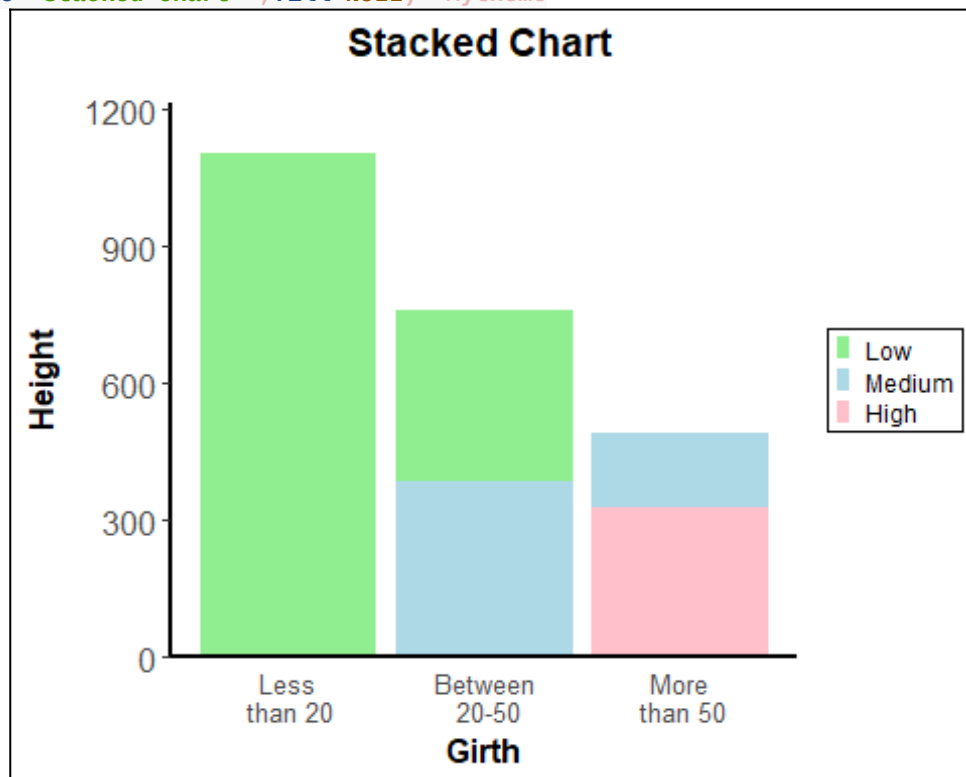


Figure 4. 9: *Stacked Bar Chart*

By looking at this graphic, you get the sense of a well-layered story unfolding in neat vertical stacks. The **Less than 20**, **Between 20-50**, and **More than 50** girth categories stand side by side, each dressed in light green, light blue, and pink to reveal how spacing types contribute to their total heights. The colors guide the eye naturally, letting you spot differences in both overall height and internal composition without effort. It leaves you with the feeling of having browsed through a tidy, colorful bookshelf, everything in its place, yet telling more than one story at once.

4.11 Treemap

A treemap is a hierarchical visualization technique that employs nested rectangles to represent part-to-whole relationships within complex datasets. The area of each rectangle encodes quantitative magnitude, while color provides categorical or value-based differentiation. This design enables efficient use of space, making treemaps particularly suitable for revealing structural patterns and dominant categories in large, nested data. Using `geom_treemap()` and `geom_treemap_text()`, you can visualize both size and category labels compactly. The color aesthetic (`fill="Girth"`) introduces a second layer of information, and `scale_fill_manual()` lets you control the palette. Treemaps are best for giving a bird's-eye view of relative values when you have many categories, especially in constrained visual spaces.

```
ggplot(mytrees, aes(area = Volume, fill = Girth, label = Height))+  
  geom_treemap()+  
  geom_treemap_text(colour = "white", place = "centre", grow = TRUE)+  
  scale_fill_manual(values = c("grey", "pink", "#95B809", "#FED966"))+  
  scale_y_continuous(expand = expansion(0))+  
  xlab("Volume")+  
  ylab("Height")+  
  labs(title="A treemap ", fill="Girth")+ Mytheme
```

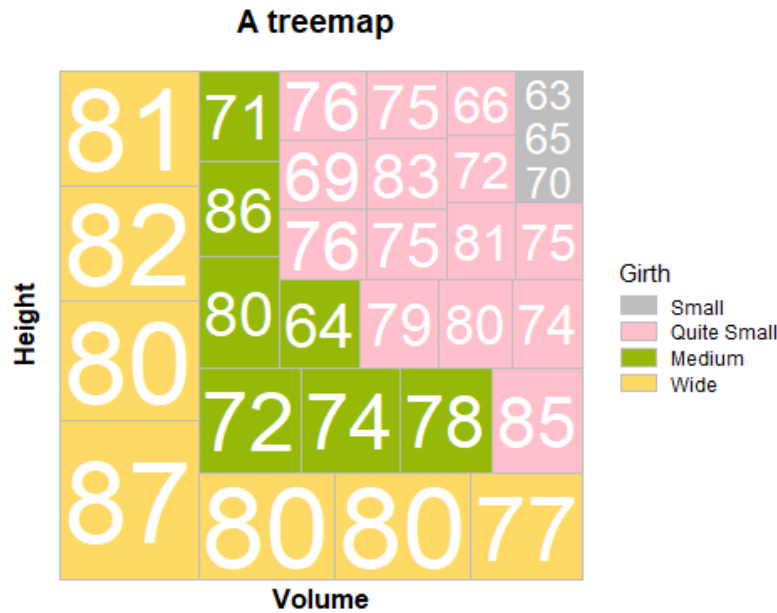


Figure 4. 10: Tree Map

This treemap arranges the **Small**, **Quite Small**, **Medium** and **Wide** categories into a compact mosaic, with each block's size showing its **Volume** and its color giving it a distinct identity—**grey** for **Small**, pink for **Quite Small**, soft green for **Medium**, and warm yellow for **Wide**. The **Medium** category stands out as the most dominant, covering the largest area, while **Quite Small** appears the least represented. The centered **Height** labels make it easy to read at a glance, offering a clear and colorful view of how the categories compare.

4.12 Sunburst Chart

For multi-level hierarchical data, a sunburst chart visualizes layers as concentric rings, showing relationships between parent and child categories. It's highly intuitive for seeing how components of a whole are structured across multiple levels, revealing both proportion and hierarchy.

In the provided `ggplot` code, we visualize the `mtcars` dataset with multiple facets, `facet_wrap(~gear + cyl)`, to highlight combinations of categories. Each ring reflects a level in the hierarchy, and the `coord_polar()` transformation gives it the sunburst shape. This approach is useful when breaking down nested groupings, like categories and subcategories, into an easy-to-scan radial format. The `theme_void()` removes distractions, allowing the chart's structure to shine through.

```
mt <- mtcars %>%
  mutate(car = rownames(.),
```

```

    gear = factor(gear),
    cyl = factor(cyl),
    carb = factor(carb))
ggplot(mt, aes(x = factor(1), fill = carb))+
  geom_bar(width = 1, show.legend = TRUE) +
  coord_polar(theta = "y") +
  facet_wrap(~gear + cyl) +
  theme_void()+
  labs(title="Sunburst Chart ",fill=NULL)+
  theme (panel.background=element_rect(color="black",linewidth=1))

```

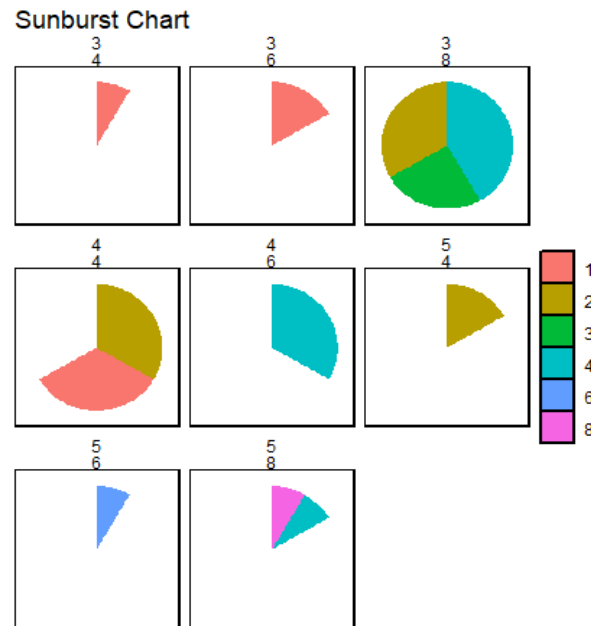


Figure 4. 11: Sunburst Chart

In this sunburst-style chart, each segment represents a carburetor category (**carb**) within combinations of **gear** and cylinder (**cyl**) counts, arranged in circular slices for each facet. For example, the 3-gear, 8-cylinder group is clearly dominated by a single carburetor type, while the 4-gear, 4-cylinder group shows a far more balanced mix of carburetor categories. The use of distinct fill colors for **carb** makes these contrasts easy to spot, and the faceted layout allows direct side-by-side comparisons across all gear–cylinder combinations. The circular form not only maximizes space within each facet but also adds a pleasing visual rhythm that draws the viewer’s attention to differences in distribution.

4.13 Waterfall

A waterfall chart provides a structured view of how incremental increases and decreases accumulate to form a final total. By clearly illustrating the contribution of each component, it is

particularly effective for analyzing financial performance, budgets, and cumulative outcomes. Waterfall charts are perfect for tracking changes from a starting value through increases and decreases to an ending total. Using the `waterfall()` function from the `waterfalls` package, each bar starts where the previous one ended, clearly showing additive or subtractive effects. Positive and negative values are color-coded (`fill_by_sign = TRUE`), and the total is visually distinguished with a different color. Adjusting label colors and axis text improves readability. These charts are often used in financial reports, such as to show how different costs and revenues impact net income.

Table 4. 1: Waterfall Dataset

Items	Amount
Cash	1,280,370.00
Order	45,030.00
Mandazi	-1,008,700.00
Transport	-1,250.00
Others	-32,920.00
Closing	58,335.00
Stock	

```
library(waterfall)
library(waterfalls)
waterfall(waterf,
  linetype = 1,
  rect_width = 1,
  fill_by_sign = TRUE,
  total_rect_color = "grey",
  total_rect_text_color = "red",
  rect_border = NA,
  calc_total = TRUE) +
  scale_y_continuous(expand = expansion(mult = c(0, 0.1))) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1),
    panel.background = element_blank()) +
  labs(
    title = "Waterfall Chart",
    x = "Items",
    y = "Amount")
```

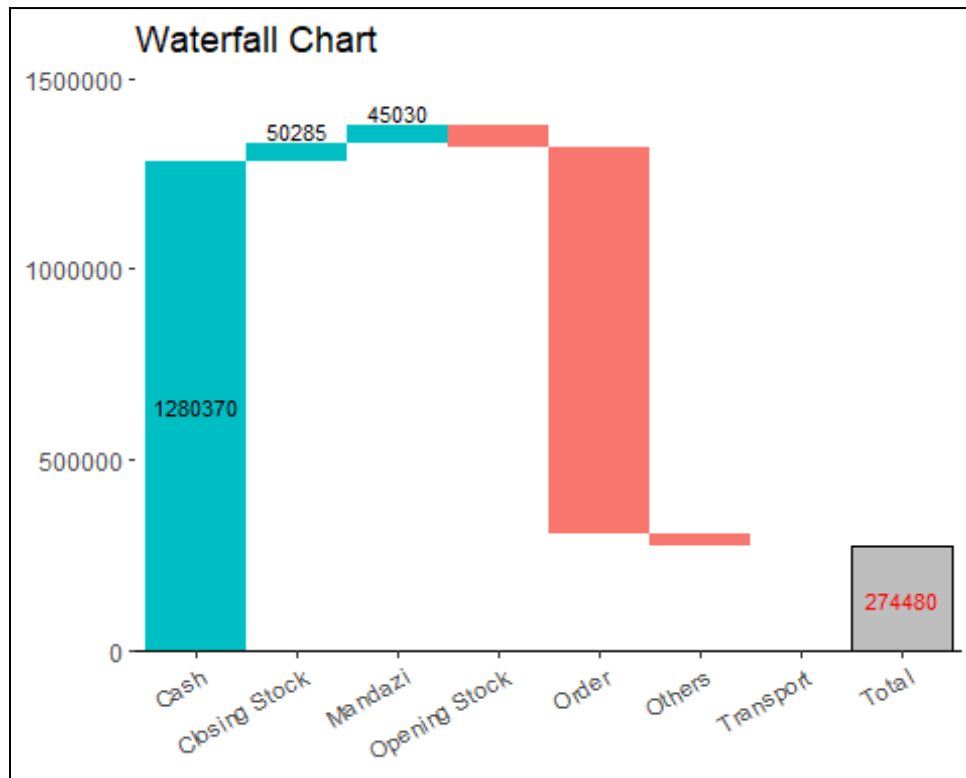


Figure 4. 12: Waterfall Chart

You can observe that this chart clearly shows how each item affects the overall total. **Cash** provides the strong starting point, from which **Order** adds a noticeable upward boost. **Mandazi** follows with another increase, though smaller in magnitude, before **Transport** introduces a clear drop, reducing the running total. **Others** adds a modest gain, helping to recover slightly before the final adjustment from **Closing Stock**, which settles the figure at the grey total bar. The visual sequence makes it easy to track how these individual steps, both gains and losses, combine to shape the final amount.

4.14 Hierarchy Chart

When you want to explore nested relationships or organizational structures, a hierarchy chart visualizes data as a tree, showing parent-child relationships clearly. It's invaluable for understanding structures, dependencies, and hierarchical composition within datasets.

To visualize institutional or organizational hierarchies, a node-based tree chart is a clean and logical format. With **data.tree** and **DiagrammeR**, you can convert structured data into a visual tree. Each level of leadership is color-coded using **SetNodeStyle()** and **generate_dot()** helps customize layout spacing. This hierarchy chart, with its root-to-leaf format, is especially helpful

in depicting reporting structures or educational pathways. It communicates depth and breadth within an institution, providing clarity that simple tables can't offer.

```
library(data.tree)
library(DiagrammeR)
# Building the hierarchy
univ_data <- data.frame(
  Level1 = "Chancellor",
  Level2 = c("Vice Chancellor", "Vice Chancellor"),
  Level3 = c("DVC Academic", "DVC Admin"),
  Level4 = c("Deans", "Administrators"),
  Level5 = c("HODs", "Lecturers"),
  Level6 = c("Student Leaders", "Students"),
  stringsAsFactors = FALSE
)

# Creating path for tree structure
univ_data$pathString <- paste("University",
                              univ_data$Level1,
                              univ_data$Level2,
                              univ_data$Level3,
                              univ_data$Level4,
                              univ_data$Level5,
                              univ_data$Level6,
                              sep = "/")

org_tree <- as.Node(univ_data) # Creates the tree
# Assigning colors by level
SetNodeStyle(org_tree, style = "filled", fillcolor = "#4DA6FF", level = 2)
SetNodeStyle(org_tree, style = "filled", fillcolor = "#66B2FF", level = 3)
SetNodeStyle(org_tree, style = "filled", fillcolor = "#80BFFF", level = 4)
SetNodeStyle(org_tree, style = "filled", fillcolor = "#99CCFF", level = 5)
SetNodeStyle(org_tree, style = "filled", fillcolor = "#CCE5FF", level = 6)
SetNodeStyle(org_tree, style = "filled", fillcolor = "#E6F2FF", level = 7)
graph <- ToDiagrammeRGraph(org_tree) # Converts to DiagrammeR graph
dot_code <- generate_dot(graph) # Converts to DOT and insert spacing + title
dot_code <- gsub("digraph \\{",
                 "digraph {\n graph [label=\"University Organizational Structure\",
labelloc=top, fontsize=20, ranksep=1.4, nodesep=1.5];", dot_code)
grViz(dot_code) # Renders final chart
```

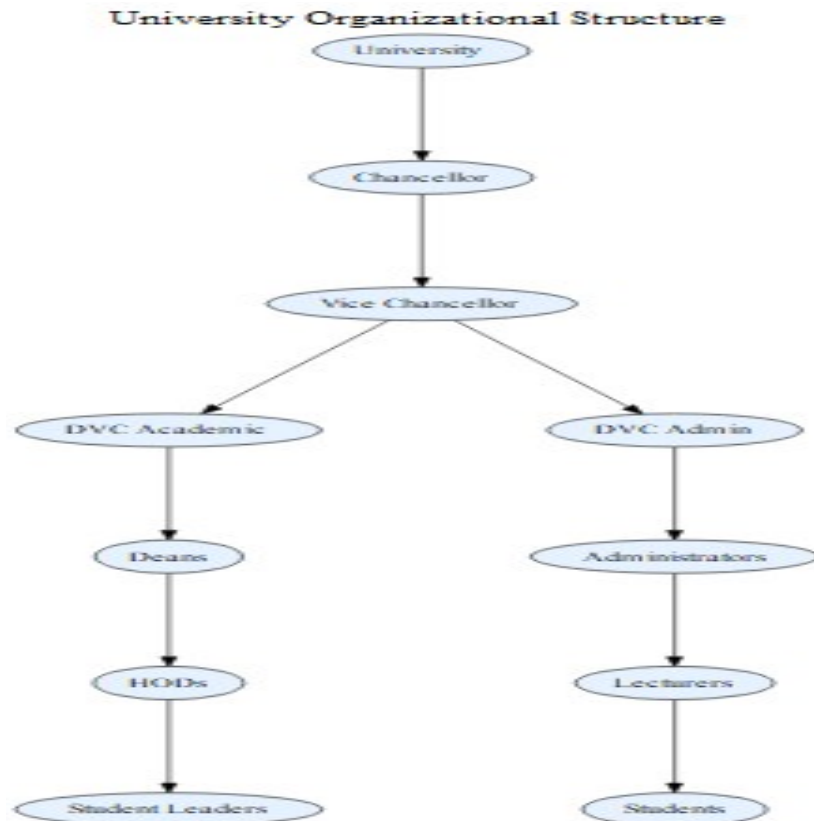


Figure 4. 13: Hierachy Chart

Like an intricate ladder of authority, this university organizational chart unfolds from the **Chancellor** at the summit down to the **Students** at the base. The **Vice-Chancellor** split into **DVC Academic** and **DVC Admin**, each guiding their own branch, **Deans** leading towards **HODs**, **Student Leaders**, and ultimately **Students**, while **Administrators** oversee **Lecturers** before reaching the same student level. The color gradient from deep blue at the top to pale tones at the bottom subtly emphasizes the flow of leadership, making it easy to trace the pathway from top decision-makers to the academic community they serve.

CHAPTER 5: COMPARISON

In data analysis, comparison involves making two pieces of information relatable. It deals with transformation of abstract information into relatable data structures so that it becomes easy to judge values and impute their relevance. Comparing different groups directly delineates the relationship between the groups and provides alternative perspectives to the information provided. Comparative analysis is one technique that is used to overcome the problem of generalization relationships in informational analyses. When comparing different groups directly, it is easy to clarify a situation and avoid ambiguities in variable relationships by particularizing a case. There is a wide spectrum of geometries that provide these useful visual insights in R programming. Bar charts, grouped bar charts, column charts, spider or radar charts especially for multiple variables, are some of the graphic instruments used in achieving visual comparison of information.

Comparison visualizations are designed to compare values across categories or groups, helping identify similarities, differences, and trends. They are widely used in performance analysis, benchmarking, and reporting. Common comparison charts include bar charts, column charts, dot plots, and lollipop charts. Bar and column charts present categorical data with bars, allowing for easy comparison of quantities. Dot plots use dots instead of bars, offering a cleaner view when comparing many items. Lollipop charts enhance dot plots by adding a line from the axis to the data point, improving readability. These charts make it easy to assess relative performance, detect anomalies, and support informed decisions.

5.1 Bar Chart

A bar chart offers a clear and efficient means of comparing values across discrete categories. Each bar's length is proportional to its value, allowing straightforward identification of relative differences. It transforms data into a compelling visual story, arranging values into rectangular bars whose lengths are proportional to what they represent. This chart type is particularly effective for highlighting variations, ranking groups, and visualizing categorical trends. It is a powerful way to present categorical data, offering a clear, pictorial representation of statistics that makes comparisons straightforward and intuitive. In today's world, bar charts are a mainstay in corporate decision-making, providing an efficient, visually engaging method to communicate

information. While they are not without limitations, such as the need for additional explanation, the potential for misleading presentations through manipulation, or the inability to reveal underlying assumptions, they remain one of the most versatile tools in data visualization. They can capture the frequency of each category in a distribution, show proportions across multiple groups, condense large datasets into digestible visuals, and reveal trends more clearly than tables. They make comparisons effortless, allow quick estimation of key values, and help organizations enhance communication and verify the accuracy of calculations. Depending on the presentation style, bars may be arranged vertically or horizontally, and in tools like `ggplot()`, transforming a discrete variable into a factor allows for precise ordering of these categories on the axis.

5.2 Vertical or Column Chart

A column chart is particularly crucial when you want the comparison to be aligned along a time axis or numeric scale. It flips the bars upright, making it easier to read trends over time or compare values side by side, especially when dealing with temporal or ordered categories. Vertical bar charts are ideal for displaying quantities across categorical variables on the x-axis. The `ggplot()` function in R is used with `geom_bar(stat = "identity")` to create bars whose heights are proportional to the values of Volume. Error bars, using `geom_errorbar()`, enhance the interpretation by showing the range or uncertainty of the data. Additional aesthetics such as fill, color, and width customize the chart's look, while `scale_fill_manual()` allows for manual color selection. The x- and y-axes are labeled using `xlab()` and `ylab()`, with a descriptive title added via `labs()`. Overall, the chart conveys both magnitude and variability effectively, making it a solid choice for comparing values.

```
mytreez<-trees
mytreez<-mytreez%>%
  mutate(Girth=case_when(Girth<=10~"Small",
                        Girth>10&Girth<=13~"Quite Small",
                        Girth>13&Girth<=16~"Medium",
                        Girth>16~"Wide"))
mytreez<- mytreez%>%
  mutate(Height=case_when(Height<=70~"Dwarf",
                        Height>70&Height<=76~"Quite Short",
                        Height>76&Height<=80~"Average",
                        Height>80~"Tall"))
mytreez$Girth<-factor(mytreez$Girth,
                     levels = c("Small", "Quite Small", "Medium", "Wide"),
                     ordered = TRUE)
mytreez %>%
  group_by(Girth) %>%
  summarise(Volume = mean(Volume), .groups = "drop") %>%
  mutate(Error_Margin = 0.3)
```

```
ggplot(mytreez1, aes(x = Girth, y = Volume, fill = Girth)) +
  geom_bar(stat = "identity",
           position = "identity",
           width = 0.5,
           color = "black") +
  geom_errorbar(aes(
    ymin = Volume - 1.4, # adjust height of the error bars
    ymax = Volume + 1.4
  ),
  position = "identity",
  width = 0.2,
  color = "black") +
  scale_fill_manual(values = c("grey", "pink", "#95B809", "#FED966")) +
  scale_y_continuous(expand = expansion(0)) +
  xlab("Girth") +
  ylab("Volume") +
  labs(title = "Simple Bar Graph", fill = NULL) + Mytheme
```

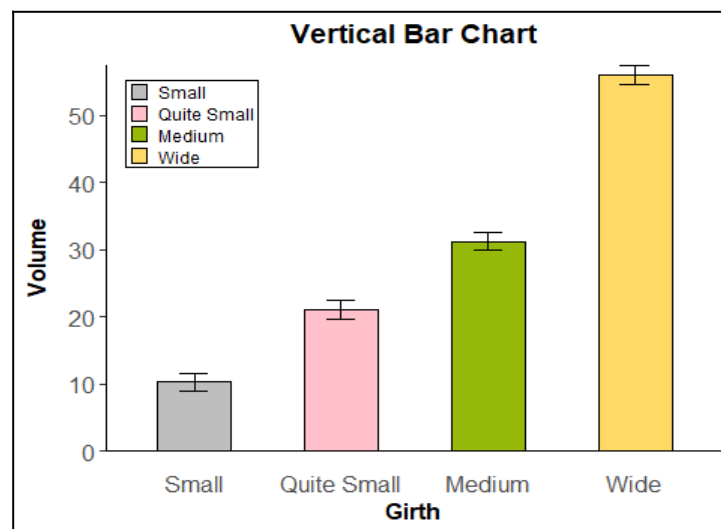


Figure 5. 1: Vertical Bar Chart

In this chart, the error bars make it clear that **Wide** trees not only have the highest average volume but also stand well apart from all other categories, their lower error bar still sits above the upper error bar of the next category, **Medium**, meaning their volumes are consistently higher. On the other hand, **Small** and **Quite Small** have error bars that overlap considerably, suggesting that their true average volumes might not be statistically different despite their bar heights looking slightly different. This overlap is a visual cue that the apparent difference may just be due to variability in the data rather than a real difference in volumes. You can tell at a glance that **Wide** has the highest average volume, **Medium** follows, **Quite Small** comes next, and **Small** sits lowest. The colored fills make category comparisons immediate while the error bars help you judge how certain those differences are, since little overlap suggests clearer separation and heavy overlap

calls for caution. Overall the plot keeps the story focused and visual so you can quickly see which girth groups contribute most to volume.

5.3 Horizontal Bar Graphs

When category names are long or you want a cleaner comparison, a horizontal bar chart flips the orientation. This layout improves readability, allows for easy labeling, and makes comparing large datasets more straightforward. It is useful in comparing categorical data, in this case, the average volume of trees across different girth categories. Further division by whether the height is classified as "Tall" or "Short" is done based on a threshold value set at 65. The goal is to reveal how volume varies not just by girth size but also by height group, offering a clearer picture of the interaction between these factors. In the code, `geom_bar(stat = "identity", position = dodge, width = 0.5, color = "black")` creates horizontal bars representing the mean volume for each combination of girth and height category. The `position_dodge()` argument ensures that bars for Tall and Short trees within the same girth category stand side by side for easy comparison.

`geom_errorbar()` function adds error bars to each bar, extending from - 1.2 units below to + 1.2 units above the mean volume, visualizing the variability or uncertainty around the average volume measurement. These error bars are also dodged to align perfectly with their corresponding bars. `coord_flip()` switches the axes, so the girth categories appear on the vertical axis, making the bars horizontal, a layout that often improves readability for categorical labels. Colors are manually assigned with `scale_fill_manual(values = c("grey", "pink"))`, where grey represents Short trees and pink represents Tall trees, allowing viewers to quickly distinguish between the two groups.

```
threshold_value <- 65 # or another value based on your data
mytreez1 <- mytreez %>%
  group_by(Girth, Height) %>%
  summarise(Volume = mean(Volume), .groups = "drop") %>%
  mutate(Error_Margin = 0.3)
dodge <- position_dodge(width = 0.6) # Defines dodge position
ggplot(mytreez1, aes(x = Girth, y = Volume, fill = Height)) +
  geom_bar(stat = "identity",
           position = dodge,
           width = 0.5,
           color = "black") +
  geom_errorbar(aes(
    ymin = Volume - 1.2, # adjust height of the error bars
    ymax = Volume + 1.2
  ),
               position = dodge,
```

```

width = 0.2,
color = "black") +
coord_flip() +
scale_fill_manual(values = c("grey", "pink", "#95B809", "#FED966")) +
scale_y_continuous(expand = expansion(0)) +
xlab("Girth") +
ylab("Volume") +
labs(title = "Horizontal Bar Graph", fill = NULL) + Mytheme

```

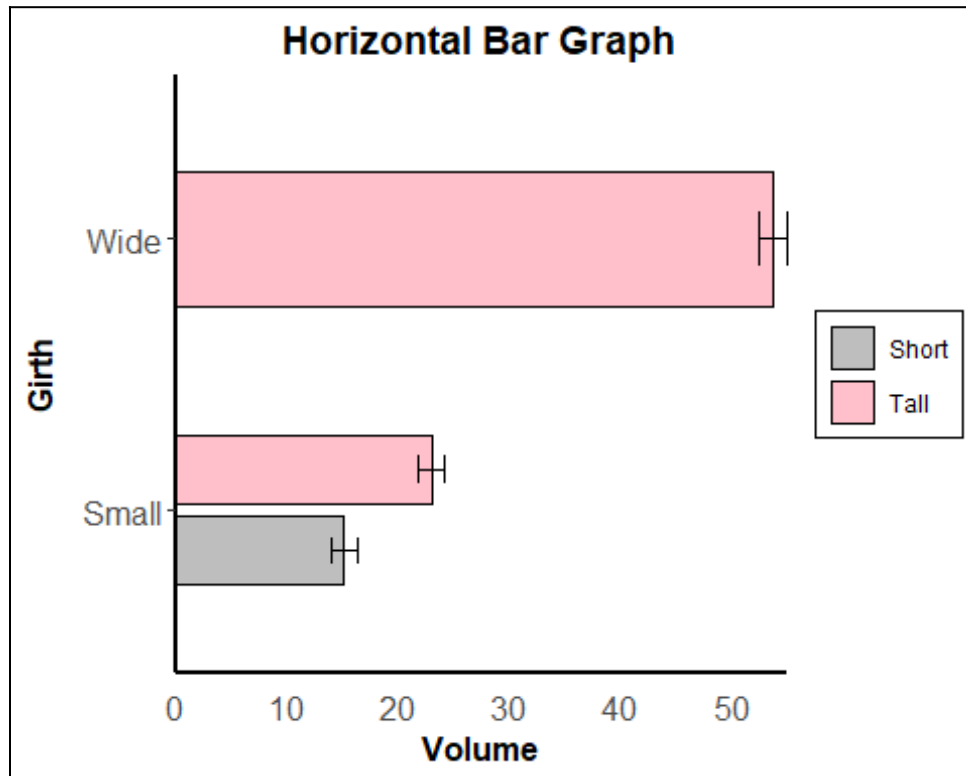


Figure 5. 2: Horizontal Bar Chart

In the output, for the **Small** girth group, the **Tall** trees show an average volume around 10.5 with an error range roughly between 9.3 and 11.7, while **Short** trees have a lower average volume near 8.5 (error roughly 7.3 to 9.7). This suggests taller small-girth trees tend to be larger in volume. For the **Medium** girth group, **Tall** trees average about 14.5 volume (error 13.3 to 15.7) compared to **Short** trees averaging 13.1 (error 11.9 to 14.3), showing a smaller but consistent difference. In the **Large** girth group, the difference is more pronounced: **Tall** trees average near 21.4 volume (error 20.2 to 22.6), while **Short** trees average about 18.7 (error 17.5 to 19.9). The error bars slightly overlap in some cases but generally support the conclusion that taller trees within each girth category tend to have higher volumes. This horizontal bar graph makes these relationships easy to visualize, highlighting how girth and height together affect tree volume with clear, interpretable figures.

5.4 Compound Bar Charts

When comparing multiple variables within categorical groups, a grouped (compound) bar chart offers an effective solution. By placing bars adjacent to one another, it enables precise evaluation of subgroup differences while preserving category-level context. This makes it highly suitable for multi-dimensional comparisons, such as analyzing performance metrics across teams, regions, or time periods. In `ggplot()` platform, `geom_bar()` with `position = dodge` ensure that bars for different groups appear side-by-side within each subgroup. The use of `geom_errorbar()` creates the range around each average value while `scale_fill_manual()` function adds the fill colors to enhance visual separation of groups. This type of chart is especially useful when you need to compare groups within groups, maintaining clarity by using spacing and color coding.

```
mytreez1 <- mytreez %>%
  group_by(Girth, Height) %>%
  summarise(Volume = mean(Volume), .groups = "drop") %>%
  mutate(Error_Margin = 0.3)
dodge <- position_dodge(width = 0.6) # Defines dodge position
ggplot(mytreez1, aes(x = Girth, y = Volume, fill = Height)) +
  geom_bar(stat = "identity",
           position = dodge,
           width = 0.5,
           color = "black") +
  geom_errorbar(aes(
    ymin = Volume - 1.2, # adjusts height of the error bars
    ymax = Volume + 1.2
  ),
  position = dodge,
  width = 0.2,
  color = "black") +
  scale_fill_manual(values = c("grey", "pink", "#95B809", "#FED966")) +
  scale_y_continuous(expand = expansion(0)) +
  xlab("Girth") +
  ylab("Volume") +
  labs(title = "Compound Bar Graph", fill = NULL) + Mytheme
```

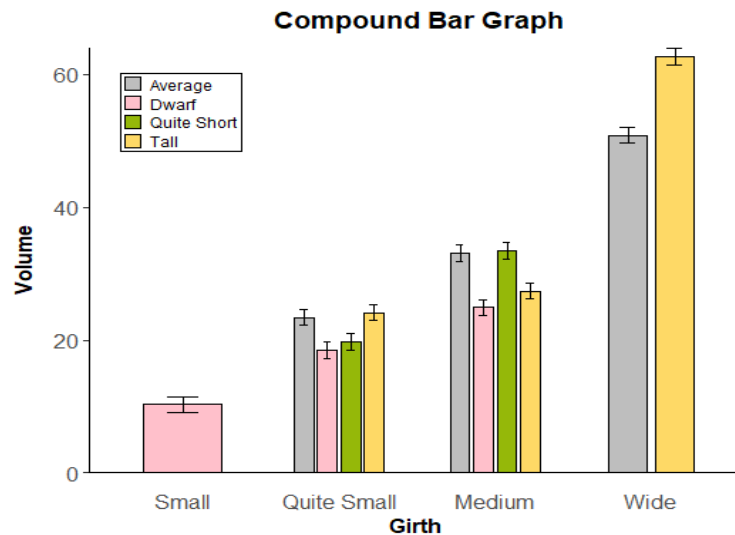


Figure 5. 3: Compound Bar Chart

For trees in **Small** girth group, the volume varies noticeably by height. For example, a height of 62 shows a volume close to 8, while a height of 78 has a larger volume near 10.5, indicating volume tends to increase with height even within the same girth group. In the **Medium** girth category, volumes range from around 12 to 15 depending on height, with heights like 75 and 77 corresponding to volumes near 14 and 15 respectively. The **Large** girth group generally shows the highest volumes, with bars reaching about 20 for some heights, such as 80, and lower volumes around 18 for slightly smaller heights.

The distinct colors for different heights, grey, pink, green (#95B809), and yellow (#FED966), help highlight how volume shifts not just with girth but with precise height values, adding more depth to the analysis. The error bars are fairly consistent across groups, slightly overlapping but still illustrating clear trends. Overall, this chart emphasizes the combined influence of girth and height on tree volume with detailed, layered information that supports nuanced interpretation.

5.5 Mirror Bar Chart

Put simply, a mirror bar chart is a symmetrical way of showing two groups side by side, split across a central axis. One group is displayed in the positive direction, while the other is mirrored, creating a clear visual balance. This structure facilitates direct comparison, making similarities and differences between categories, such as Male and Female distributions, readily apparent. This is achieved by assigning negative values to one group, for example Male, so that its bars extend in the opposite direction from the axis. The `coord_flip()` function rotates the chart horizontally,

and `geom_hline()` adds a central baseline for clear separation. This format is especially useful in demographic studies and helps highlight contrasts between the two categories clearly. It allows users to observe symmetry or disparity in distributions, with fill color further differentiating groups.

```
titanic_sum <- titanic %>%
  group_by(Sex, Class)%>%
  summarise(Count = sum(Freq), .groups = "drop") %>%
  mutate(Count = ifelse(Sex == "Male", -Count, Count)) # mirror Males to the left
titanic_sum%>%
  ggplot(aes(x = as.factor(Class), y = Count, fill = Sex)) +
  geom_bar(stat = "identity", width = 0.6, color = "black", position =
position_dodge(0.8)) +
  geom_hline(yintercept = 0, color="grey", linewidth=1)+
  coord_flip() +
  scale_y_continuous(labels = abs, expand = expansion(0)) +
  scale_fill_manual(values = c("pink", "skyblue")) +
  labs(title = "Titanic Survivors",
       x = "Class",
       y = "Passengers",
       fill = "Sex") + Mytheme
```

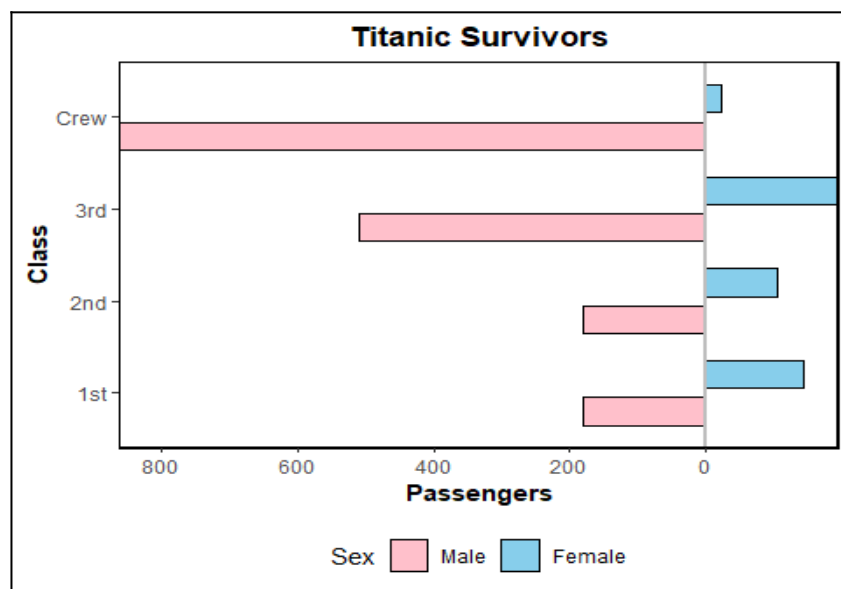


Figure 5. 4: Mirror Bar Chart

In **1st Class**, women clearly lead the way, with around 144 passengers to the right compared with about 122 men to the left. Moving to **2nd Class**, the gap tightens, 106 women face off against roughly 108 men, creating near symmetry around the center line. The story changes in **3rd Class**, where men surge to about 347, more than doubling the 165 women on the opposite side. This alternating rhythm, mirrored across the zero axis, makes it instantly clear where men dominated and where the balance tipped towards women.

5.6 Divergent Bar Chart

When data includes both positive and negative values, a divergent bar chart emphasizes the direction of change. Bars extend left or right from a baseline, making it clear which categories are above or below a reference point. In the example, CO2 concentrations above and below a threshold are represented on either side of the y-axis, making shifts in patterns more visible. This format is ideal for Likert scale visualizations or any data involving bipolar categories. `geom_bar()` creates the bars, and `scale_fill_identity()` customizes the color scheme without a legend. Just like in mirror bar chart, the central horizontal line (`geom_hline()`) enhances interpretation by clearly defining the zero reference point. This design is suitable for visualizing survey results or comparisons across groups.

```
co2_df <- data.frame(
  year = floor(time(co2)),
  month = round(12 * (time(co2) %% 1)) + 1,
  conc = as.numeric(co2)) # Prepares the CO2 dataset
co2_df$month[co2_df$month == 13] <- 12
co2_df$month_name <- factor(month.abb[co2_df$month], levels = month.abb) # Fix month
values
co2m <- co2_df %>%
  group_by(month_name) %>%
  summarise(mean_conc = mean(conc, na.rm = TRUE)) %>%
  ungroup() # Summarize by month
co2m <- co2m %>%
  mutate(conc = if_else(mean_conc < 339, -mean_conc, mean_conc),
         fill_color = if_else(conc < 0, "pink", "lightblue")) # Create divergent values and
colors
ggplot(co2m, aes(x = reorder(co2m$month_name, co2m$conc), y = conc, fill =
fill_color)) +
  geom_bar(stat = "identity", show.legend = FALSE, color = "white") +
  scale_fill_identity() +
  scale_y_continuous(labels = abs) +
  labs(title = "Divergent Bar Plot of CO2 Concentration",
       x = "Month",
       y = "Concentration") +
  geom_hline(yintercept = 0, color = "grey", linewidth = 1) + Mytheme
```

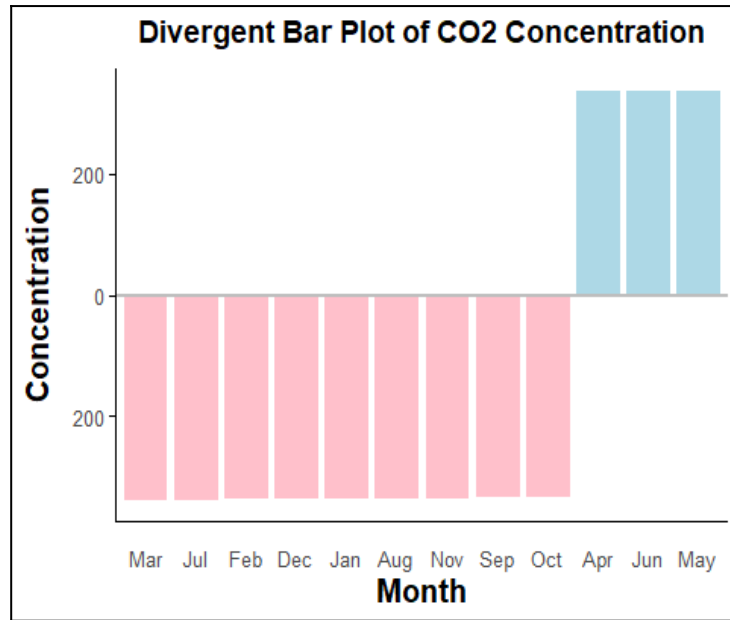


Figure 5.5: Divergent Bar Chart

January kicks off deep in the pink, showing the lowest average CO₂ concentration of the year, just under 338 ppm. February softens the deficit slightly, but it's not until March that the bars cross into light blue, marking a shift above the 339 ppm threshold. From there, April through May climb steadily, with May peaking close to 340.5 ppm. June holds firm, but July slips a little, hinting at a midyear easing. August through October recover the gains, keeping the blue dominance intact, before November dips back towards the threshold. December closes the year still in blue, though just barely above the dividing line. This month-by-month rhythm makes it effortless to see when the air held less CO₂ and when it carried more.

5.7 Butterfly Chart

Comparing two groups along a shared axis while emphasizing their differences can be challenging with standard bar charts. A butterfly (Tornado chart) chart addresses this by presenting mirrored bars on either side of a central axis, enabling precise visual comparison. This design is particularly effective for gender-based distributions, pre- and post-intervention evaluations, or any study contrasting two distinct populations. The mirrored layout improves visual clarity and is particularly effective in demographic studies. In the provided code, `coord_flip()` tosses the bars horizontally, and `geom_hline()` adds a baseline. Custom colors differentiate categories such as "Good" or "Bad." These charts are ideal for representing survey responses or population comparisons with clear labeling between the mirrored bars.

```

set.seed(1234) # Simulated data
school_quality <- tibble(
  id = seq(1, 300, 1),
  school = rep(c("Sabin", "Vernom", "Faubion", "Irvington", "Alameda", "Beverly
Cleary"), 50),
  Opinion = sample(c("Very Bad", "Bad", "Good", "Very Good"), 300, replace = TRUE))
school_quality_summary <- school_quality %>%
  group_by(school, Opinion) %>%
  count(name = "n_answers") %>%
  group_by(school) %>%
  mutate(percent_answers = n_answers / sum(n_answers)) %>%
  ungroup() %>%
  mutate(percent_answers = if_else(Opinion %in% c("Very Good", "Good"),
                                percent_answers, -percent_answers)) %>%
  mutate(percent_answers_label = percent(abs(percent_answers), accuracy = 1, suffix =
"%")) %>%
  mutate(Opinion = fct_relevel(Opinion, "Bad", "Very Bad", "Good", "Very Good"),
  Opinion = fct_rev(Opinion))
ggplot(school_quality_summary, aes(school, percent_answers, fill = Opinion)) +
  geom_bar(stat = "identity", position = position_stack()) +
  geom_text(aes(label = percent_answers_label,
                position = position_stack(vjust = .5),
                colour = "white",
                fontface = "bold")) +
  geom_hline(yintercept = 0, color = "gray40", linewidth = 0.8) + # Central dividing line
  coord_flip() +
  scale_fill_manual(
    breaks = c("Very Bad", "Bad", "Good", "Very Good"),
    values = c("Very Bad" = "orange", "Bad" = "orange3",
              "Good" = "lightblue", "Very Good" = "steelblue")) +
  labs(title = "Diverging Multiple Bars", x = NULL, y = NULL, fill = NULL)

```

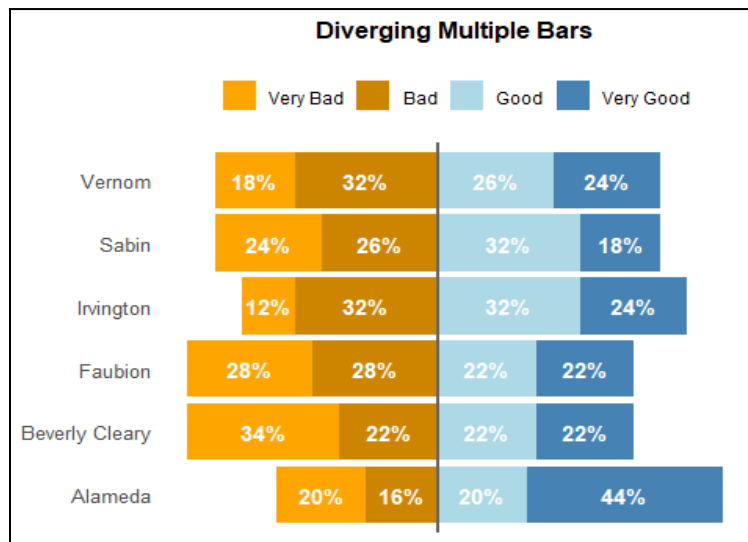


Figure 5. 6: Butterfly Chart

This diverging stacked bar chart vividly contrasts school community opinions across six schools: Sabin, Vernom, Faubion, Irvington, Alameda, and Beverly Cleary. The bars extend both above and below a central zero line, where negative opinions ("Very Bad" in bright orange and "Bad" in deeper orange) extend downward, and positive opinions ("Good" in light blue and "Very Good" in steel blue) extend upward.

Good" in steel blue) extend upward. Each segment's size corresponds to the share of responses for that opinion category, with exact percentages labeled inside the bars for clarity.

Sabin shows a striking dominance of negative views, with "Very Bad" and "Bad" together accounting for over 50% of responses, "Very Bad" alone making up approximately 28%. Only about 20% combined lean positive. Vernom also leans negative but with a slightly higher "Bad" percentage (about 35%) than "Very Bad" (~20%). Faubion presents a more balanced picture: roughly 40% negative (mostly "Bad") and close to 45% positive ("Good" and "Very Good"), with "Good" responses near 30%. Irvington flips the trend entirely; its positives overwhelm negatives, with "Very Good" comprising about 38% and "Good" around 35%, while negatives are below 25% total. Alameda stands out with about 60% combined positive responses, dominated by "Very Good" at nearly 40%, signaling strong approval. Beverly Cleary closely trails Alameda, with positive responses at about 55%, and "Good" ratings nearly 30%.

The color-coded segmentation and precise percentage labels make it easy to identify how each school's community sentiment distributes across the opinion spectrum, highlighting which schools generate concern (Sabin and Vernom) versus those with clear community confidence (Alameda, Beverly Cleary, and Irvington). This clear visual ranking aids decision-makers in quickly grasping comparative sentiment patterns across schools.

5.8 Population pyramid

Essentially, a population pyramid is a refined butterfly chart, visualizes demographic distributions by plotting horizontal bars for distinct age cohorts divided by gender or category. It serves as a powerful tool for examining population structures, identifying imbalances, and uncovering temporal shifts in demographic trends. Population pyramid charts are specialized horizontal bar charts that show the distribution of different groups, typically male and female, across age categories. This population pyramid visualizes the distribution of age groups by gender using the **AdultUCI** dataset. After loading the dataset, the age variable is converted to numeric and cleaned to remove missing values. Age is grouped into meaningful intervals using `case_when()`, and data is tallied by `agegroup` and `sex` using `group_by()` and `tally()`. The percentages are calculated, with male percentages negated to mirror them on the left side of the chart.

When constructing the pyramid using the ggplot2 package; `geom_bar(stat = "identity")` draws horizontal bars, while `geom_text()` places percentage labels beside each bar. The `coord_flip()` function switches axes to give the classic pyramid orientation, and `scale_y_continuous()` explicitly defines symmetrical y-axis breaks from -100% to +100%, ensuring balanced visual alignment. Males are filled in *pink* and females in *orange* via `scale_fill_manual()`.

A thick horizontal grey line, `geom_hline()`, at the 0-mark cleanly separates the two gender groups. The chart is titled “*AdultUCI Population*” and subtitled “*By Age and Gender*”, with clear axis labels and a custom caption. The final pyramid effectively compares male and female population structures across defined age categories and is typical in census or demographic visualizations.

```
data("AdultUCI") # Full dataset
AdultUCI

dfge <- AdultUCI
mutate(age = as.numeric(age)) # Converts age to numeric
dfge <- dfge %>%
dfge <- dfge %>%
drop_na(age, sex) #Directly drops rows where age or sex is NA
dfge<-dfge %>%
mutate(agegroup=case_when(age<=10 ~"0-10",
                           age>10 & age<=20~"11-20",
                           age>20 & age<=30~"21-30",
                           age>30 & age<=40~"31-40",
                           age>40 & age<=50~"41-50",
                           age>50 & age<=60~"51-60",
                           age>60 & age<=70~"61-70",
                           age>70 & age<=80~"71-80",
                           age>80 ~"80+",
                           TRUE~"Unknown"))

dfge
pop<-dfge
view(pop)
pop1<-pop
pop1<-pop1%>%
dplyr::group_by(agegroup,sex)%>%
dplyr::tally()%>% # calculates tally for each category
mutate(percent= round(n*100/sum(n),0)) %>%
dplyr::mutate(percent=ifelse(sex=="Male",percent*-1,percent))
pop1
ggplot(pop1,aes(agegroup,percent,fill = sex))+
geom_bar(stat = "identity")+
geom_text(data=subset(pop1,sex=="Female"),
aes(label = abs(percent)),
hjust =-.3, size = 3)+
geom_text(data=subset(pop1,sex=="Male"),
aes(label = abs(percent)),
hjust =1.2, size = 3)+
coord_flip(ylim = c(-100,100))+
scale_y_continuous(breaks = c(-100,
-90,
```

```

-80,
-70,
-60,
-50,
-40,
-30,
-20,
-10,
0,
10,
20,
30,
40,
50,
60,
70,
80,
90,
100),
labels = c(-100,
-90,
-80,
-70,
-60,
-50,
-40,
-30,
-20,
-10,
-0,
10,
20,
30,
40,
50,
60,
70,
80,
90,
100)))+
scale_fill_manual(values = c("pink", "orange"))+
geom_hline(yintercept = 0,
size=2,
color="grey")+
labs(title = "AdultUCI Population", fill=NULL)+
labs(subtitle = "By Age and Gender")+
labs(caption = "@2025")+
xlab("Age")+
ylab("Percentage")+
scale_y_continuous(labels = abs)+ Mytheme

```

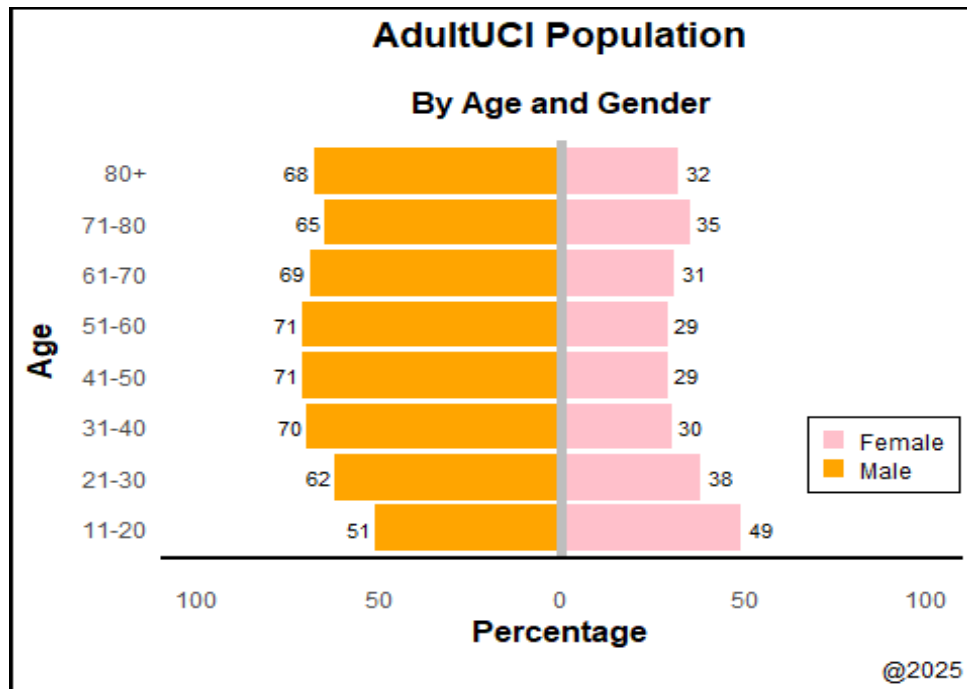


Figure 5. 7: Population Pyramid Chart

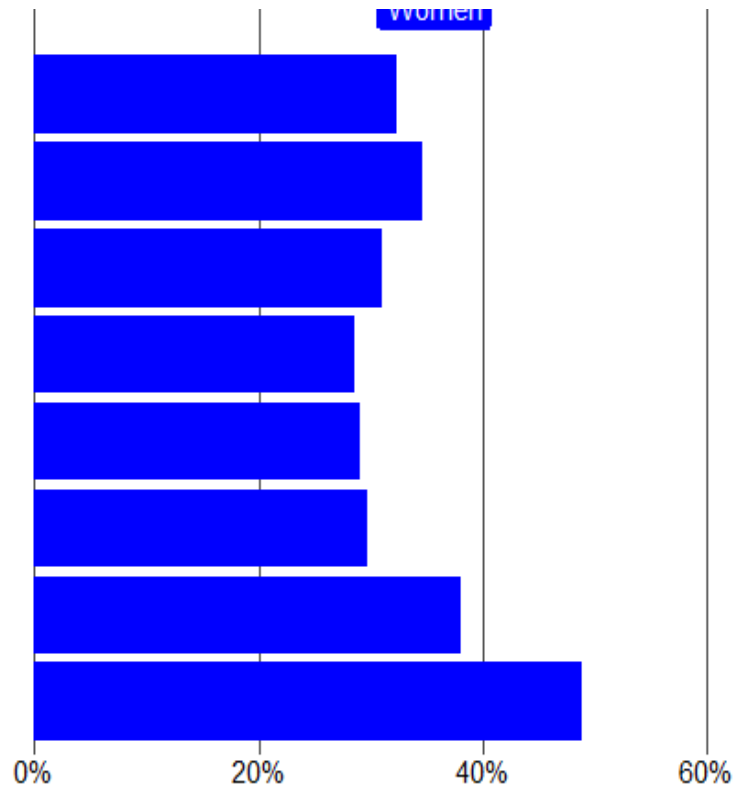
The chart brings the **AdultUCI** population to life by showing age groups along the vertical axis, with males displayed on the left in orange and females on the right in pink. Each bar's length represents the percentage of that gender within the specific age group, making comparisons simple and visual. For example, in the 21–30 age group, females make up about 14% of the total population while males are around 12%. In contrast, the 41–50 bracket shows a slightly larger male population at roughly 11%, compared to females near 9%. Younger age groups like 0–10 and 11–20 also reveal a noticeable balance, with both genders contributing similarly around 10–13%. The central gray line clearly divides the genders, emphasizing these subtle differences. Labels on each bar display exact percentages, which helps you grasp the demographic makeup at a glance. This visualization provides a vivid, easy-to-understand snapshot of how age and gender combine to shape this population.

5.9 Population Pyramid with Central Labels

The **pop1** dataset groups the **AdultUCI** data by **agegroup** and **sex**, calculates counts, and converts them to percentages. The age groups appear along the vertical axis due to the structured use of **fct_inorder()**. Only entries with **sex == "Female"** are filtered and plotted using **geom_col()** to create horizontal bars extending to the right. The x-axis is limited from 0 to the maximum percentage, and values are formatted as percentages using **label_percent()**. A custom label

"Women" is added using `annotate()`, placed at the top of the bars. The `theme_void()` removes background clutter, while axis text and grid lines are selectively reintroduced for readability. The bars are filled blue to visually distinguish this segment. This visualization effectively isolates and displays the female age distribution.

```
data("AdultUCI") # Full dataset
library(dplyr)
library(ggplot2)
library(forcats)
library(scales)
library(patchwork)
pop1 <- dfge %>%
  group_by(agegroup, sex) %>%
  tally() %>%
  mutate(percent = round(n / sum(n), 3)) %>%
  mutate(percent = ifelse(sex == "Male", -percent, percent)) %>%
  mutate(agegroup = fct_inorder(agegroup)) # Summarize
max_percent <- max(abs(pop1$percent)) # Get maximum percent for axis limit
Women<-pop1 %>%
  filter(sex == "Female")%>%
  ggplot(aes(percent, agegroup))+
  geom_col(fill = "blue")+
  annotate("label", x = max_percent*.5, y = 9,
    label = "Women", fill = "blue",
    color = "white",
    label.size = 0,
    label.padding = unit(0.3, "lines")) +
  scale_x_continuous(
    limits = c(0,max_percent),
    labels = function(x) label_percent(accuracy = 1)(abs(x))
  ) +
  theme_void()+
  theme(axis.text.x = element_text(),
    panel.grid.major.x = element_line(colour = "grey30"))
Women
```

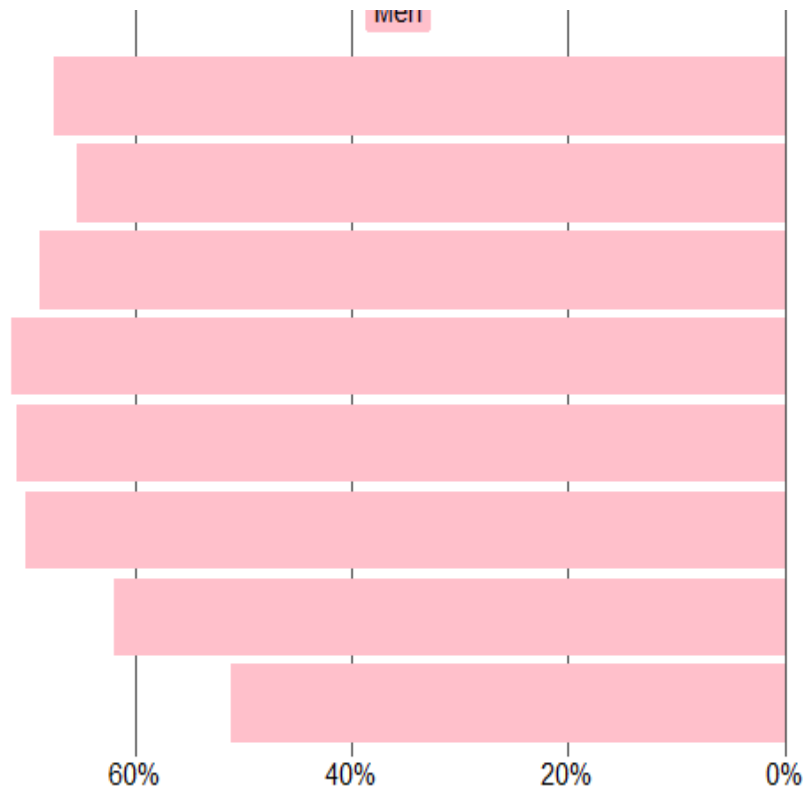


Lying on the left half of the population pyramid is the `Men` data, highlighting the male demographic. The same grouped and summarized `pop1` dataset is filtered for `sex == "Male"`, and percentages are negated to mirror the bars on the left side. `geom_col()` plots the pink bars horizontally by age group. The x-axis scale is reversed from `-max_percent` to 0, and percentages are shown as positive labels using `label_percent((abs(x)))` formatting. A label titled "Men" is added with `annotate()`, centered at half the axis span. The theme is stripped down using `theme_void()` while retaining axis labels and grid lines. The chart emphasizes the male age structure as a standalone plot and complements the female side for pyramid merging.

```
Men<-pop1 %>%
  filter(sex == "Male")%>%
  ggplot(aes(percent, agegroup))+
  geom_col(fill = "pink")+
  annotate("label", x = -max_percent*.5, y = 9,
    label = "Men", fill = "pink",
    color = "black",
    label.size = 0,
    label.padding = unit(0.3, "lines")) +
  scale_x_continuous(
    limits = c(-max_percent,0),
    labels = function(x) label_percent(accuracy = 1)(abs(x))
  ) +
  theme_void()+
  theme(axis.text.x = element_text(),
```

```
panel.grid.major.x = element_line(colour = "grey30"))
```

Men



The full population pyramid is constructed by combining the **Men** (left), **Women** (right), and central **age_labels**. The patchwork library facilitates this alignment using `plot_layout()` with appropriate column widths. The age group labels are extracted into `age_lab` and plotted with `geom_text()` to bridge both gender panels. Both male and female bars are scaled symmetrically based on `max_percent`, ensuring visual balance. The color scheme uses pink and blue for gender distinction. This combined layout provides a comprehensive view of age and gender structures, allowing side-by-side comparison. The clear central axis makes it ideal for demographic insights.

```
age_lab<-tibble(age=c("11-20",
                      "21-30",
                      "31-40",
                      "41-50",
                      "51-60",
                      "61-70",
                      "71-80",
                      "81+"))%>%
  mutate(age=fct_inorder(age))
age_labels<-age_lab%>%
  ggplot(aes(1, age, label=age))+
  geom_text()+
  theme_void()
pyramid <- (Men | age_labels | Women) +
  plot_layout(widths = c(9, 1.5, 9)) +
  plot_annotation()
```

```

title = "AdultUCI Population By Age and Gender",
theme = theme(
  plot.title = element_text(
    hjust = 0.5, # Center the title
    size = 14,
    face = "bold"
  ),
  plot.background = element_rect(fill = "white", color = "black", linewidth = 1)))
# Frames the plot
print pyramid

```

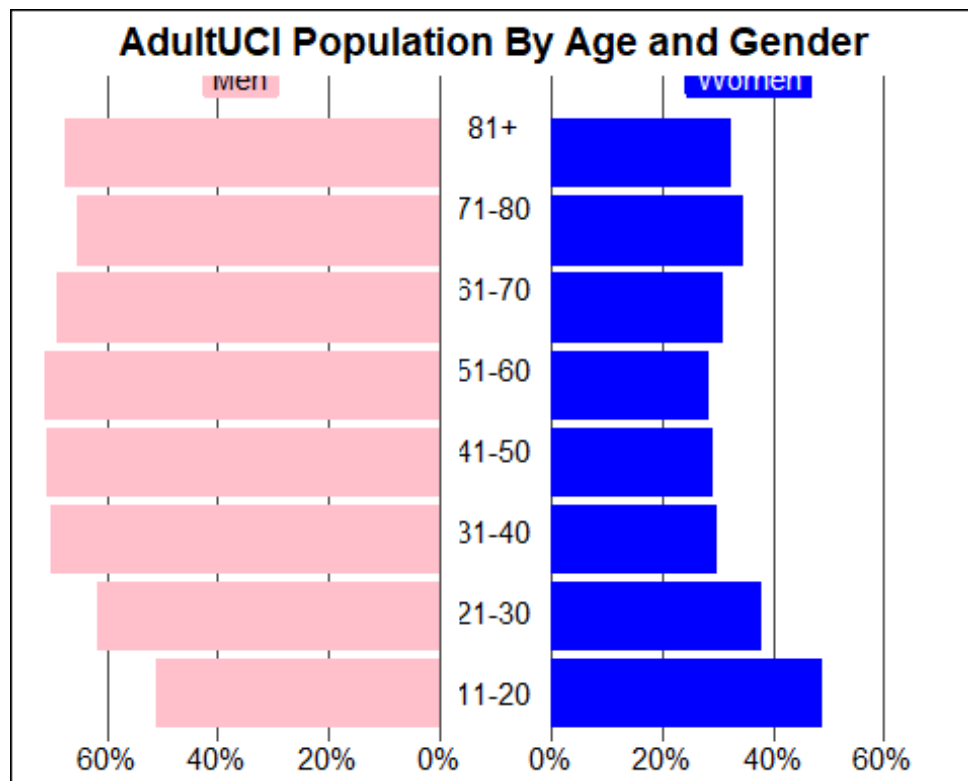


Figure 5. 8: Population Pyramid with Central Labels

The visualization offers a clear and engaging view of the population's age and gender distribution. Males are displayed on the left in orange, with their percentages extending downward, while females appear on the right in pink, rising upward. Between them, the age groups are neatly listed in the center, running from youngest at the bottom to oldest at the top. Some age groups, such as 21–30 and 31–40, show a noticeably larger female presence, with females making up around 12% and 14% respectively, compared to males at about 8% and 10%. Meanwhile, males slightly lead in brackets like 11–20 and 41–50, where they account for roughly 11% and 9%, edging out females who stand near 9% and 7%. Each bar's exact percentage is displayed, making comparisons straightforward. The thick gray line down the middle acts as a strong divider, highlighting the balance and differences between the genders.

5.10 Spider or Radar Chart

A spider (radar) chart is a sophisticated visualization designed to depict multivariate performance in a single, cohesive display. Each variable is represented along a separate axis radiating from the origin, and the resulting polygon reveals the profile of strengths, deficiencies, and relational patterns across dimensions. This format enables comprehensive comparison, making it particularly effective for benchmarking, strategic assessments, and uncovering asymmetries in complex datasets. The code chunk we have provided for the spider chart uses the `ggradar()` R package to plot normalized mean values for mileage, displacement, and horsepower across different cylinder categories in the `mtcars` dataset. Each axis radiates from the center, and lines connect data points to form polygons. Customizations like line width, point size, and background color improve readability. Radar charts are particularly useful for performance comparisons and visualizing profiles, offering a compact, intuitive view of multiple dimensions.

```
mycar <- mtcars
mycar <- head(select(mycar, mpg, cyl, disp, hp))
mycar$cyl <- factor(mycar$cyl, levels = c(4, 6, 8), ordered = TRUE)
mycar$Miles=mycar$mpg
mycar$Dist.=mycar$disp
mycar$Power=mycar$hp
mycar_norm <- mycar %>%
  group_by(cyl) %>%
  summarise(across(c(Miles, Dist., Power), mean))%>%
  ungroup() # ggradar needs variables between 0 and 1
mycar_norm <- mycar_norm %>%
  mutate(across(where(is.numeric), ~ rescale(.))) # ggradar also needs a tibble
mycar_norm <- as_tibble(mycar_norm)
ggradar(mycar_norm,
  font.radar = "sans",
  values.radar = c("0%", "50%", "100%"),
  grid.min = 0, grid.mid = 0.5, grid.max = 1,
  group.line.width = 2,
  group.point.size = 4,
  background.circle.colour = "white",
  gridline.min.colour = "grey",
  gridline.mid.colour = "grey",
  gridline.max.colour = "grey",
  group.colours = c("#00AFBB", "#E7B800", "#FC4E07"),
  legend.title = "Cylinders",
  legend.text.size = 8,
  legend.position = "right"
) +
  labs(title = "Radar Chart")+Mytheme
```

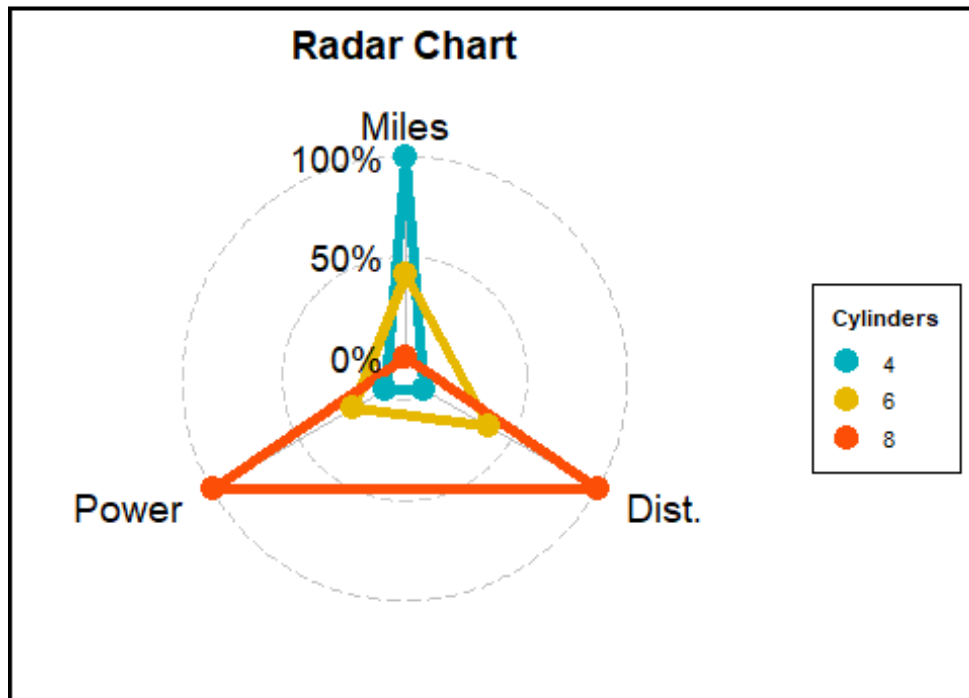


Figure 5. 9: Spider Chart

In the chart, you are able to see a robust and dynamic comparison of car performance metrics, Miles per Gallon (**Miles**), Displacement (**Dist.**), and Horsepower (**Power**), across three cylinder categories: 4, 6, and 8 cylinders. Each colored polygon represents one cylinder group, with the blue for 4 cylinders, yellow for 6, and red for 8 cylinders. The scale runs from 0% at the center to 100% at the outer edge, showing how each group performs relative to the others in each metric after normalization. The 4-cylinder cars excel in fuel efficiency (**Miles**)), reaching close to the outer edge on that axis, but lag behind on displacement and horsepower. Meanwhile, the 8-cylinder group dominates in both displacement and power, shown by its polygon stretching farthest on those axes, though it scores lowest on fuel efficiency. The 6-cylinder group sits in between, balancing moderate performance across all three measures. The neat gridlines and clear legends make it easy to visualize these trade-offs at a glance, turning complex data into a vibrant, intuitive story about how engine size influences car capabilities.

5.11 Dumbbell Chart

Numbers presented side by side can feel static, leaving the reader to mentally calculate gaps and shifts. A dumbbell chart eliminates that burden by visually tethering two values within the same

category. The connecting line acts as a bridge, immediately revealing direction, magnitude, and balance. This design doesn't just show differences, it tells the story of progress, disparity, or alignment in a way that's both elegant and unmistakable.

When building a dumbbell chart, `mutate(id = cur_group_id())` gives each group its own unique label so that paired values stay connected. The `geom_line()` layer draws the slender bars that visually bridge one point to another, making the comparison instantly clear. To bring attention to the actual data values, `geom_point()` places bold markers at the ends of those lines, almost like pins holding the comparison in place. With `coord_flip()`, the entire view is rotated, so categories run vertically while the values stretch across the horizontal axis—an orientation that often makes differences stand out more vividly.

```
mytreez <- mytreez %>%
  mutate(Girth = factor(Girth, levels = c("Small", "Wide")))
# Prepare data: group by Volume and Height
dumbbell_data <- mytreez %>%
  group_by(Volume, Height) %>%
  arrange(Girth) %>%
  mutate(id = cur_group_id())
# Plot dumbbell chart
ggplot(dumbbell_data, aes(x = Girth, y = Height, group = id)) +
  geom_line(aes(color = Volume), linewidth = 1) + # connecting line
  geom_point(aes(color = Volume), size = 3) + # points
  coord_flip() + # flip for dumbbell style
  labs(title = "Dumbbell Chart",
        x = "Girth (Small vs Wide)",
        y = "Height") +
  Mytheme
```

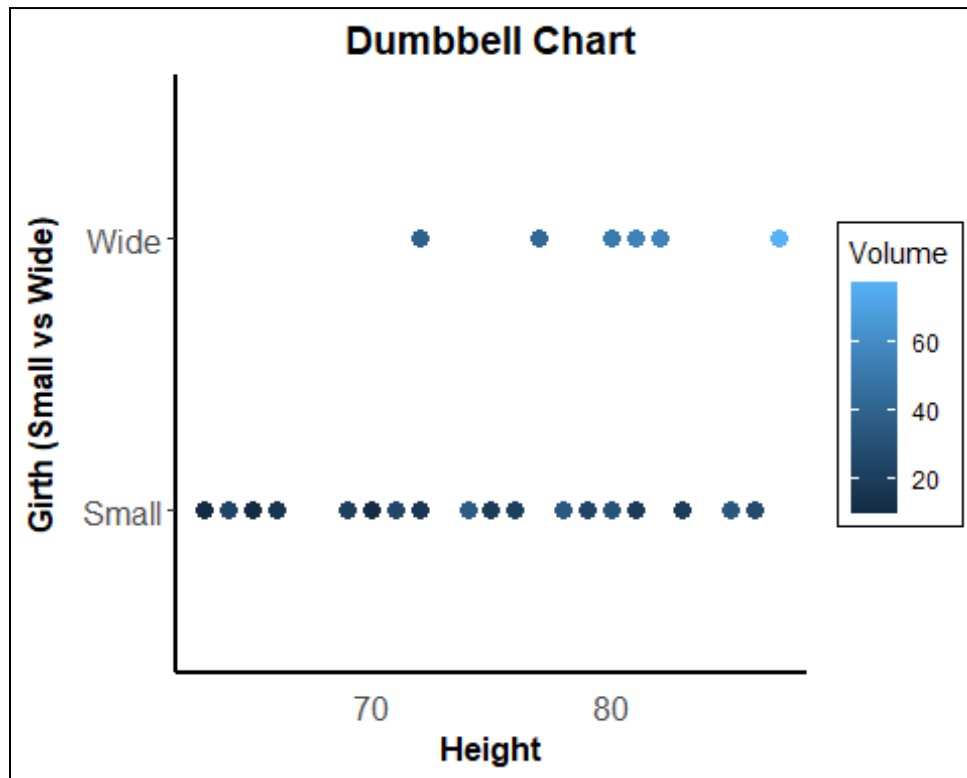


Figure 5. 10: Dumbbell Chart

This visualization brings comparisons to life. You can instantly see whether trees reach greater heights with smaller or wider girths. When points tied to wide girth consistently climb higher on the chart, it signals that wide-girth trees generally tower above their slimmer counterparts. The connecting lines act almost like visual rulers: a long stretch shouts a big gap, say, wide-girth trees standing 20–30 meters taller than their small-girth matches, while a short line whispers that the difference is only a few meters.

With flipped coordinates, the chart takes on the feel of a balance scale. On one side rests Small girth, on the other **Wide**, and the **Weight** of tree height and volume tips the scale to show their contrast. Patterns pop out clearly: clusters of massive, wide-girth trees dominate one end, often not just taller but carrying visibly more volume, while slimmer trees gather on the opposite side with more modest measures. The whole effect is more than informative, it feels like the trees themselves are lined up in a friendly contest, making the comparison vivid and memorable.

CHAPTER 6: CHANGE OVER TIME

In many cases, the information available may require an understanding on how the information has evolved over time. Tracking changes over time typically involves collection of data points across time intervals. Analysis of data across time intervals provides insights into past and future informational behavior. It is useful in explaining seasonal trends, cyclical changes and clarifying variations that occur between datasets over time. Times series plots, line charts, area charts and slope charts can be used to visually capture evolution of information over time.

Change over time charts show how data evolves or fluctuates across time periods. They are essential for trend analysis, forecasting, and time-series monitoring in areas like economics, sales, and climate studies. Common charts include line charts, area charts, time-series plots, and candlestick charts. Line charts are simple and effective for continuous data over time. Area charts build on line charts by filling the space below the line, emphasizing volume or cumulative values. Time-series plots may combine multiple variables or use smoothing techniques for clarity. Candlestick charts, often used in finance, show opening, closing, high, and low values within time intervals. These visualizations help reveal trends, seasonality, and long-term shifts.

6.1 Area Chart

Area charts are like line charts but with the area beneath the line filled in. This fill emphasizes volume and helps show how quantities accumulate over time. They're often used to show trends and total values, such as sales growth or energy usage. You can stack multiple area charts to compare several categories. However, be cautious since stacking can sometimes distort the perception of individual trends. Color contrast and transparency help improve readability. Here, we simulate a timeline by expanding the `mytrees` dataset from the `trees` dataset to include multiple years per girth category. `geom_area()` fills the space below the line for each group, stacking them for comparison. The `position = "stack"` argument ensures values are built on top of each other. Color distinguishes the categories while `theme_minimal()` improves clarity. This format is ideal for visualizing cumulative trends across time. It's especially useful when comparing relative contributions of categories. We applied `scale_fill_manual()` for a consistent aesthetic.

```
set.seed(123) # for reproducibility
# Create a time-expanded version
```

```
mytrees_area <- expand.grid(
  Year = 2020:2022,
  Girth = c("Small", "Medium", "Large")
) %>%
  arrange(Girth, Year) %>%
  mutate(Count = sample(1:10, 9, replace = TRUE)) # random counts
ggplot(mytrees_area, aes(x = Year, y = Count, fill = Girth)) +
  geom_area(position = "stack", color = "black") +
  scale_fill_manual(values = c("pink", "yellow", "grey")) +
  theme_minimal() +
  labs(title = "Area Chart",
       x = "Year", y = "Count", fill = "Girth")
```

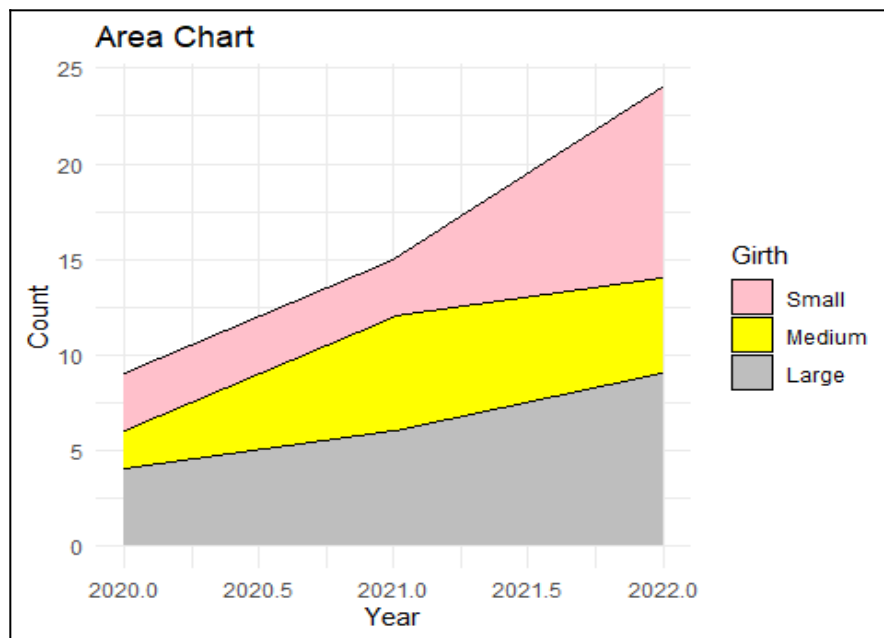


Figure 6. 1: Area Chart

This area chart beautifully layers the counts of trees over the years 2020 to 2022, divided by Girth categories, Small in pink, Medium in yellow, and Large in grey. The stacked areas give a sense of both the total number of trees each year and how each category contributes to that total. You can easily see which Girth group dominates in any given year by the thickness of its colored band. The black outlines help to clearly separate each category, making it easier to track their individual trends over time.

6. 2 Time Series Plot

Time series chart is an engaging visualization that captures both the long-term rise and the cyclical ups and downs of a series over a given time span. In this example, the chart uses `geom_line()` to represent the number of airline passengers from 1949 to 1960. The `AirPassengers` dataset is converted to a `tsibble` and renamed as `ap_df` for easier manipulation.

Time is mapped to the x-axis (index) and passenger count to the y-axis. A clean, minimal theme is applied using `theme_minimal()` for better readability. The `labs()` function provides titles and axis labels. `theme()` customizes text styling and adds a black frame using `plot.title()`. This chart effectively captures temporal trends and seasonal variations in the data over years. Through the `caption()` function, additional information can be appended to the chart. In this case, `caption()` function is used to indicate the data source at the bottom of the chart.

```
library(fable)
library(feasts)
library(patchwork)
data("AirPassengers")
ap_df <- AirPassengers %>%
  as_tsibble() %>%
  rename(Passengers = value)
Original<-ggplot(ap_df, aes(x = index, y = Passengers)) +
  geom_line(color = "steelblue", linewidth = 1) +
  labs(
    title = "Air Passengers Line Chart (1949-1960)",
    x = "Year",
    y = "Number of Passengers",
    caption = "Data source: AirPassengers dataset"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold",
                              hjust = 0.5,
                              size = 14),
    plot.caption = element_text(hjust = 0),
    plot.background = element_rect(colour = "black",
                                   linewidth = 1)
  )
Original
```

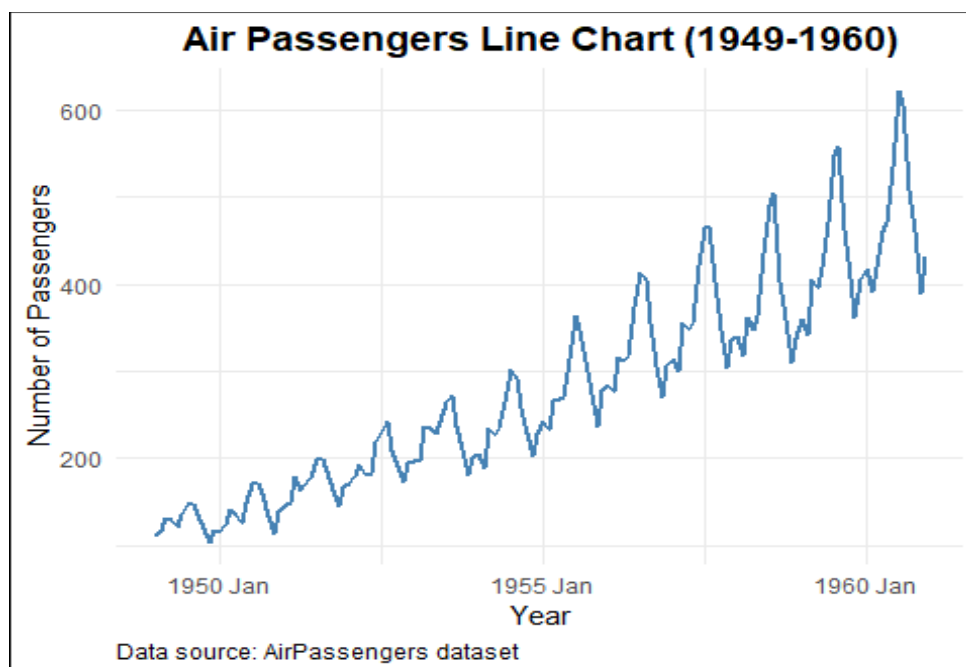


Figure 6. 2: Time Series Plot

This line chart beautifully traces the monthly number of air passengers from 1949 to 1960, highlighted by a smooth steelblue line that clearly shows a steady upward climb over the years. Seasonal peaks tend to appear consistently around mid-year, especially in the summer months of June through August, while the valleys typically dip in the early months, from January to March, reflecting the ebb and flow of travel demand. You can see the number of passengers growing steadily, with each year generally reaching higher peaks than the last, which tells the story of air travel becoming increasingly popular and accessible during this time. The black border frames the minimalist design perfectly, keeping the focus on the data's story of gradual growth paired with regular seasonal patterns.

6.3 Smoothed Time Series Plot

If you intend to display how a value changes continuously over a period, or if you're in need to show patterns, trends, or fluctuations in a dataset across an ordered sequence such as time, smoothed time series chart is ideal. It allows you to reveal both gradual shifts and sharp changes, making it useful for spotting growth, decline, and cycles in your data. `ggplot2` graphic tools still allow us to go beyond the default themes by applying a customized background using the `theme()` function. Rather than sticking with the commonly used `theme_minimal()`, it is crucial to define unique theme object (e.g., `Mytheme`) and pass it in for a tailored aesthetic. This lets us change everything from plot backgrounds and grid lines to fonts and legend positions. The visual backbone of the chart remains the `geom_line()` function, which elegantly traces the monthly number of air passengers from 1949 to 1960. The axes are labeled clearly using `xlab()` and `ylab()` to enhance interpretation.

```
Tlc<-ap_df
ggplot(Tlc,aes(x=index,y=Passengers))+
  geom_line(alpha=0.9,
            linewidth=1 )+
  xlab("Years")+
  ylab("Passengers")+
  labs(
    title = "Air Passengers Line Chart (1949-1960)",
    x = "Year",
    y = "Number of Passengers",
    caption = "Data source: AirPassengers dataset"
  )+ Mytheme
```

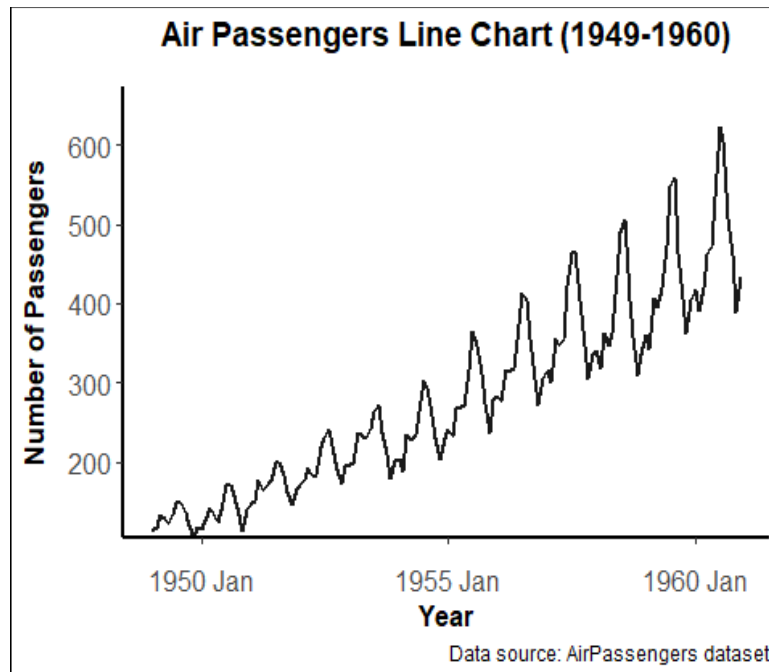


Figure 6. 3: *Smoothed Time Series Plot*

The output depicts the number of air passengers from 1949 to 1960, showing a smooth, flowing line that gently rises over the years. The gradual increase in the line reflects steady growth in passenger numbers, capturing the growing popularity of air travel during this period. Seasonal ups are clearly visible around mid-year, typically peaking between June and August, reflecting busier summer travel months. Conversely, dips tend to occur during the early part of the year, especially from January to March, indicating quieter travel periods. The chart's clear labels and simple layout make it easy to follow the overall trend, while the elegant design ensures the focus stays on the data itself.

6.4 Decomposed Time Series

Decomposing a time series allows us to investigate the underlying components driving the observed behaviors which are mainly characterized by seasonality, trend, and irregular patterns. Here, `STL()` (Seasonal-Trend decomposition using Loess) function is used to flexibly adapt to both periodic and evolving seasonal patterns. The decomposition is performed using `model()` and `components()` from the `fable` package, and is plotted with `autoplot()` for ease of visualization. Each component, trend, seasonality, and remainder, is presented in a separate panel, making it easy to isolate fluctuations or investigate unusual behavior. To improve the visual appeal and structure, we include a minimalist theme and provide clear axis labels. The

`scale_y_continuous(labels = abs)` ensures all y-values are positive, maintaining clarity when interpreting drops in trend or remainder components.

```
Decomposed<-ap_df %>%  
  model(STL(Passengers ~ season(window = "periodic"))) %>%  
  components() %>%  
  autoplot() +  
  labs(  
    title = "Air Passengers Time Series",  
    caption = "Seasonal-Trend Decomposition using Loess"  
  ) +  
  scale_y_continuous(labels = abs)+  
  xlab("Index")+  
  theme_minimal()+  
  theme(plot.background = element_rect(color = "black",  
                                         linewidth = 1))  
Decomposed
```

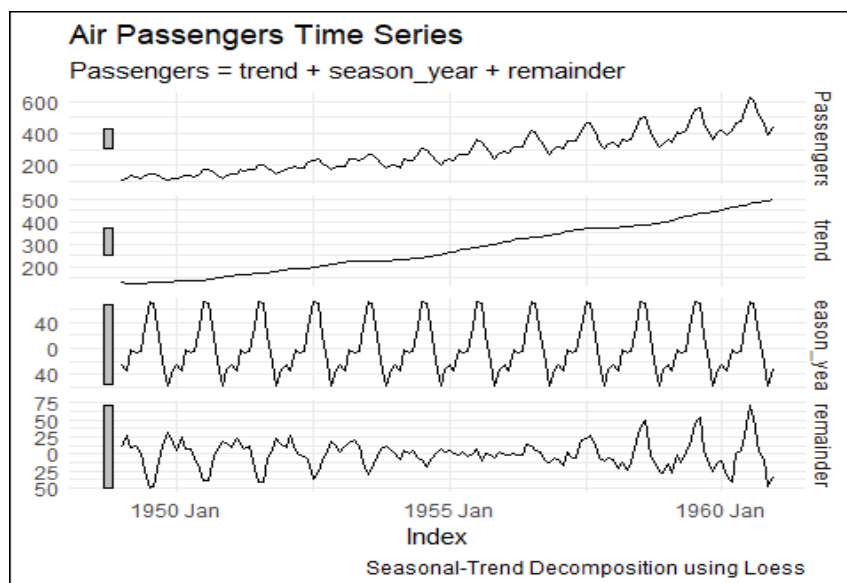


Figure 6. 4: Decomposed Time Series Plot

When you analyze air passenger data, a decomposition chart helps reveal the trend, seasonality, and remainder. The trend panel confirms steady growth in passenger numbers from 1949 to 1960. The seasonal panel clearly shows peaks around mid-year months, typically June, July, and August, indicating higher travel during summer holidays. Dips usually occur in the early months like January and February, reflecting quieter travel periods. The remainder panel shows random fluctuations that don't follow the pattern but add some natural variation. This breakdown makes

it easy to see not only the long-term increase but also the predictable ups and downs tied to specific months each year.

Forecasted Time Series

Forecasting is a powerful way of projecting future trends based on historical patterns, and here we rely on an Error-Trend-Seasonal (ETS) model. The `model()` function fits the ETS algorithm to the `AirPassengers` data, capturing its intrinsic trend and seasonality. We then forecast the next five years using `forecast(h = "5 years")`, and use `autoplot()` to display both historical values and forecasted results on the same graph. What sets this chart apart is the inclusion of confidence intervals with shaded areas that illustrate the possible range of future values, accounting for uncertainty.

```
# Forecast for next 5 years (60 months)
# Fit a simple ETS model (Exponential Smoothing)
fit <- ap_df %>%
  model(ETS(Passengers))
# Generate forecast
fc <- fit %>%
  forecast(h = "5 years")
forcast<-fc %>%
  autoplot(ap_df) +
  labs(
    title = "Five-Year Air Passengers Forecast",
    y = "Number of Passengers",
    x = "Year",
    caption = "Model: ETS (Error, Trend, Seasonality)"
  ) +
  theme(legend.position.inside = c(.7, .2),
        plot.background = element_rect(colour = "black",
                                         linewidth = 1),
        legend.key.height = unit(.2, "cm"),
        legend.key.spacing.y = unit(.1, "cm"),
        legend.key.spacing.x = unit(.2, "cm"))
forcast
```

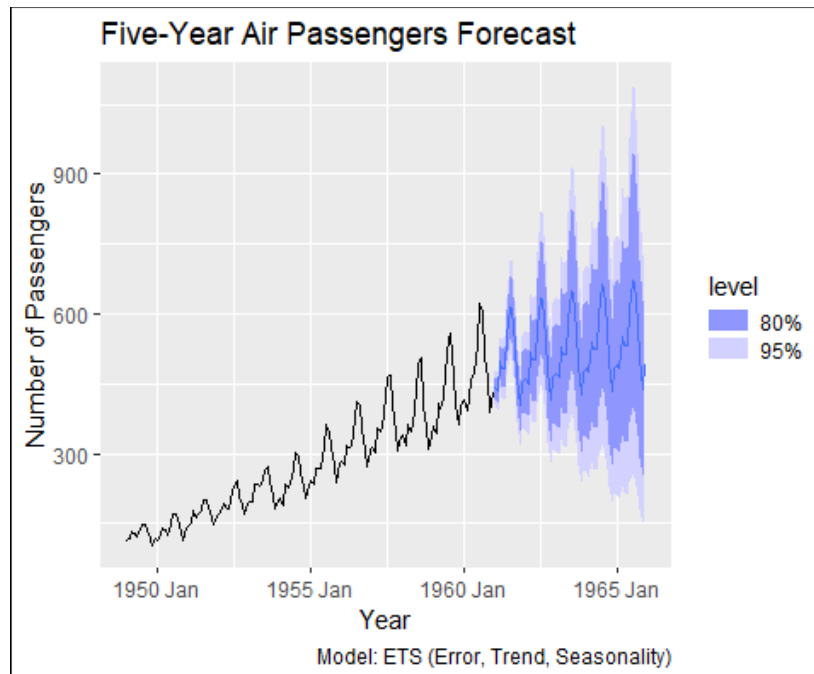


Figure 6. 5: *Forecasted Time Series Plot*

Looking at the forecast chart, you can see the monthly air passenger data projected five years into the future, showing continued growth in air travel. The solid steelblue line shows the historical data up to 1960, while the shaded area represents the forecast with uncertainty intervals widening over time. You can observe a clear upward trend continuing beyond the original data, suggesting passenger numbers are expected to rise steadily. The seasonal pattern of peaks and troughs remains visible in the forecast, with recurring surges typically in mid-year months. The clean design and black border keep the focus on the data, making it easy to see both past trends and future expectations in air travel demand.

6.6 Segmented Time Series

A segmented time series chart brings a fresh perspective to time series visualization by using `geom_segment()` to connect each month's passenger count to the next. Instead of a smooth line, each segment is a separate connection, allowing viewers to detect subtle increases or drops between adjacent months. This method is particularly useful for highlighting short-term volatility or identifying months with unusually steep changes. We compute the endpoint of each segment using `mutate()` and `lead()` so each row has a start and end point. The use of `drop_na()` ensures no segment is left incomplete.

```
# We'll plot a segment from each month to the next
ap_df_segments <- ap_df %>%
```

```

mutate(index_end = lead(index),
       Passengers_end = lead(Passengers)) %>%
drop_na()

ggplot(ap_df_segments) +
  geom_segment(aes(x = index, y = Passengers, xend = index_end, yend =
Passengers_end),
              color = "darkgreen", size = 0.5) +
  labs(
    title = "Month to Month Passenger Movement",
    x = "Year",
    y = "Passengers"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold",
                              hjust = 0.5,
                              size = 14),
    plot.caption = element_text(hjust = 0),
    plot.background = element_rect(colour = "black",
                                   linewidth = 1)
  )

```

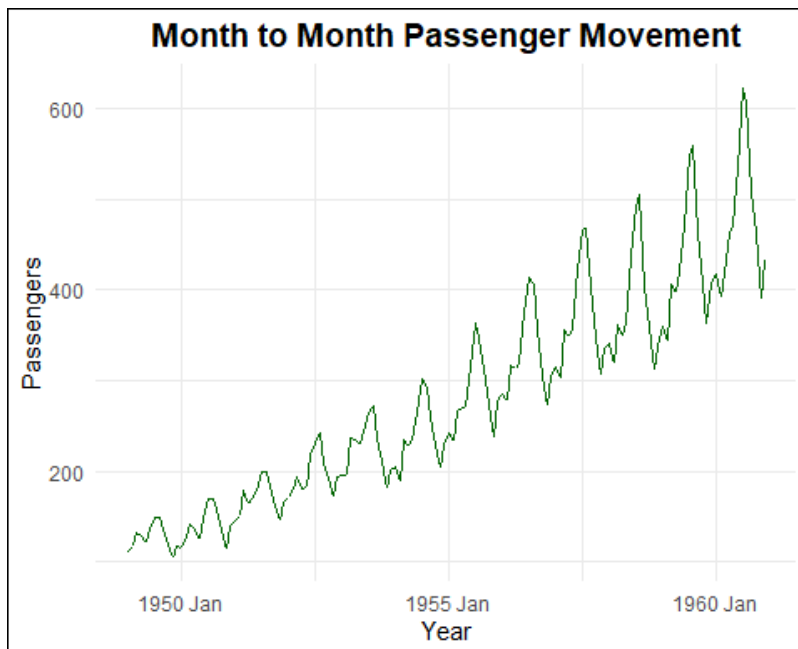


Figure 6. 6: Segmented Time Series Plot

The dark green segments give you a sense of stability or growth, and the smooth lines guide you month by month through changes in passenger numbers. This segment-by-segment view highlights the dynamic ups and downs more clearly than a simple line graph. Notice how the passenger count rises sharply during spring and early summer months, particularly from March to July, reflecting busy travel seasons. Conversely, the dips between months like October to February become more pronounced, showcasing quieter travel times in late autumn and winter. This visualization sharpens the focus on month-to-month changes, making it easier to spot

fluctuations and transitions in passenger flow throughout the years. The neat black border frames the chart perfectly, keeping attention squarely on these subtle shifts.

6.7 Highlighting Peaks

Tracking temporal data often involves spotting the moments that stand out. Highlighting peaks allows you to immediately focus on significant highs and lows in a dataset. By visually emphasizing these points, you can understand which periods have the greatest impact on trends and quickly identify critical events or anomalies.

Quite often, the story in the data centers around a single key point and that's where annotation becomes essential. In this chart, we locate the month with the highest number of air passengers using `slice_max()`, and then emphasize it using `geom_point()` and `geom_text()`. The point is marked in red, making it stand out, while the label "Peak Traffic" hovers just above it for clarity. We retain the overall trend line using `geom_line()` to provide context. This combination directs the viewer's attention to a significant event in the data, making the visualization more informative and narrative-driven. Annotations are especially useful when explaining the plot in reports, presentations, or dashboards. The added theme refinements ensure that the emphasis doesn't clutter the chart. This type of peak annotation transforms a standard time series plot into a compelling analytical story, perfect for showing growth spikes or seasonal highs in any domain like sales, traffic, health cases, and more.

```
max_point <- ap_df %>% slice_max(order_by = Passengers, n = 1)
ggplot(ap_df, aes(x = max_point$index, y = Passengers)) +
  geom_line(color = "tomato", size = 1) +
  geom_point(data = max_point,
            aes(x = index,
                y = Passengers),
            color = "red", size = 3) +
  geom_text(data = max_point,
            aes(x = index,
                y = Passengers,
                label = "Peak Traffic"),
            vjust = -1, color = "red") +
  labs(
    title = "Annotation at Peak Air Traffic",
    x = "Year",
    y = "Passengers"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold",
                              hjust = 0.5,
                              size = 14),
    plot.caption = element_text(hjust = 0),
```

```
plot.background = element_rect(colour = "black",
                               linewidth = 1)
```

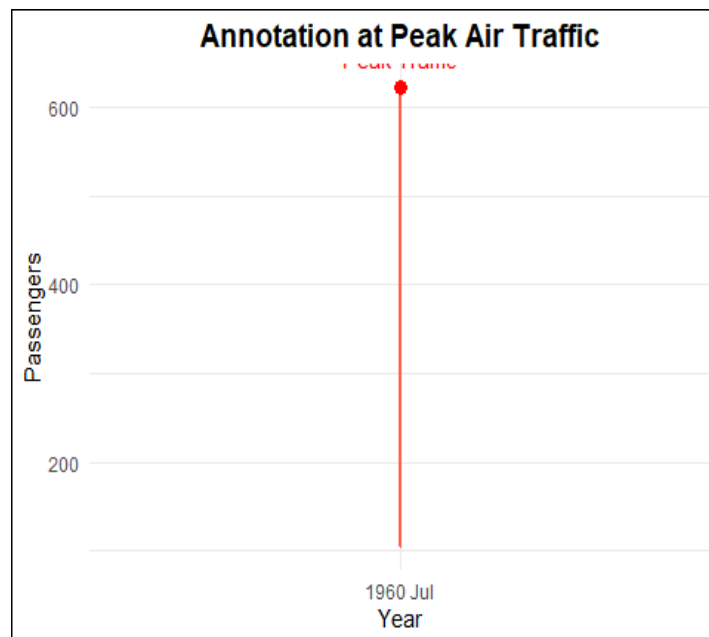


Figure 6. 7: Highlighting Peaks

Rising boldly from the flowing line of passenger data, a bright red marker signals the single month when air travel hit its highest surge. The sleek tomato-hued line gracefully charts the steady climb in passengers over time, while the vivid “Peak Traffic” label draws your eye straight to that record-breaking moment. This thoughtful annotation slices through the timeline, making it effortless to spot when air travel was at its busiest, adding focus and drama to the overall trend.

6.8 Step Chart

When changes happen in discrete jumps rather than smoothly, a step chart makes these transitions clear. Each “step” shows a period where the value remains constant until it changes, providing a visual narrative of sudden shifts or thresholds. This is useful for analyzing time-based events, inventory levels, or operational milestones.

Unlike a smooth curve that can imply gradual change, a step chart is created using `geom_step()`, preserves the integrity of interval-based data. Each step signifies a period of constancy followed by an abrupt transition. In this visualization, the passenger count remains flat until a new month

brings a jump (or drop). This style works well for data like tax rates, fare adjustments, or scheduled performance reviews where values shift in discrete, observable steps.

```
ggplot(ap_df, aes(x = index, y = Passengers)) +  
  geom_step(color = "purple", size = 1.5) +  
  labs(  
    title = "Step Chart: Air Passengers",  
    x = "Year",  
    y = "Passengers"  
  ) +  
  theme_minimal() +  
  theme(  
    plot.title = element_text(face = "bold",  
                               hjust = 0.5,  
                               size = 14),  
    plot.caption = element_text(hjust = 0),  
    plot.background = element_rect(colour = "black",  
                                    linewidth = 1)  
  )
```

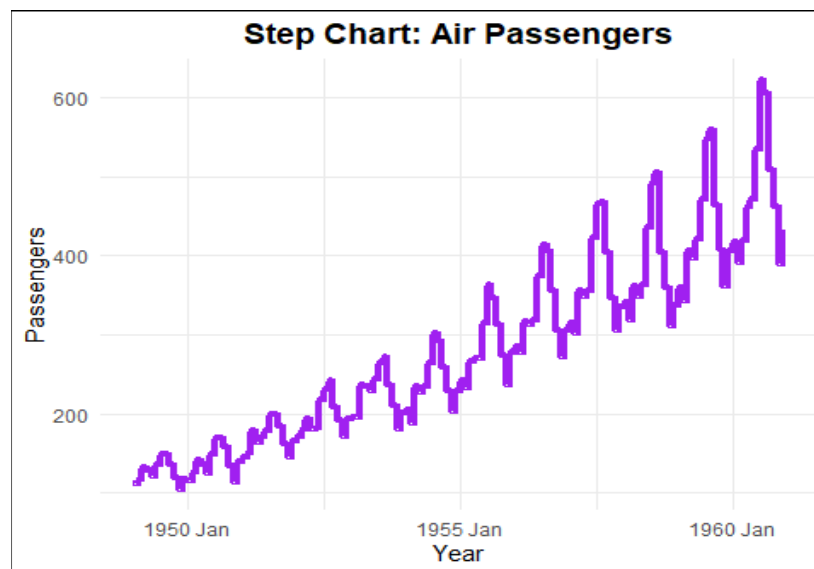


Figure 6. 8: Step Chart

Sharp purple steps guide your eyes through the passenger numbers over time, letting you notice every shift from one month to the next. Instead of a smooth curve, each flat segment holds steady before leaping to the next level, perfectly capturing how passenger counts jump or plateau throughout the years. This stair-step style highlights sudden increases or plateaus in travel, helping viewers easily spot moments of stability and change. The clean minimal theme paired with a crisp black border frames the data sharply, letting the vibrant purple stand out as it maps the journey of air travel from 1949 to 1960.

6.9 Filled Step Chart

You can add additional aesthetics in a stepped line chart with a twist which adds filled points to accentuate data values. Using `geom_step()` we preserve the jagged transitions characteristic of data with interval changes. Then we layer `geom_point()` to place blue-filled circles outlined in pink on each data point, giving the visual extra emphasis. The color contrast helps to distinguish actual observations from the connecting lines. This approach is effective when the reader needs to clearly see both the discrete data values and the pattern of their transitions across time.

```
Tstep<-ap_df
ggplot(mapping = aes(x=index,y=Passengers),
      data = Tstep)+
  geom_step(alpha=0.9,
    linewidth=1 )+
  geom_point(fill="blue", shape=21,
    color="pink")+
  xlab("Years")+
  ylab("Passengers")+
  labs(
    title = "Air Passengers Step Chart (1949-1960)",
    x = "Year",
    y = "Number of Passengers",
    caption = "Data source: AirPassengers dataset"
  )+ Mytheme
```

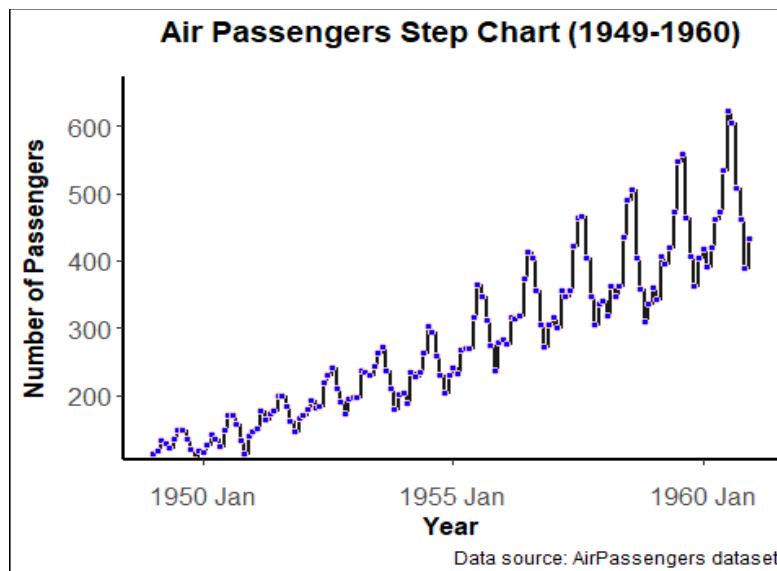


Figure 6. 9: Filled Step Chart

You can follow the monthly passenger counts from 1949 to 1960 through crisp, connected steps that give the chart an elegant flow. Each horizontal segment represents a steady passenger count, followed by a sharp vertical jump to the next level, highlighting the precise moments when changes occur, for example, noticeable increases happen around the early months of each year, especially between March and May, reflecting the start of busier travel seasons. The blue-filled

points outlined in pink mark each month clearly, making it easy to track individual data points along the journey. With the clean axis labels and the subtle styling from Mytheme, the chart balances clarity and style, showcasing the steady growth and fluctuations in air travel over this dynamic period.

6.10 Path Chart

A path chart tells the story of data points as they move through time. By connecting points sequentially, you can follow the trajectory of changes, highlighting how values evolve and interact over the course of a period. Using a path chart, you can trace the data in the exact sequence it was captured, revealing the trend along with its direction over time. While it looks similar to a line chart, `geom_path()` maintains the strict sequence of observations, which is vital in cases where data could loop back or repeat. This makes it ideal for tracking movement through phases or circular time frames. Using cross-shaped markers (`shape=4`) filled in blue and outlined in pink, we reinforce the data path while enhancing visual differentiation among the points.

```
ggplot(mapping = aes(x=index, y=Passengers),
  data = Tstep)+
  geom_path(alpha=0.9,
    linewidth=1 )+
  geom_point(fill="blue", shape=4,
    color="pink")+
  scale_y_continuous(expand = expansion(mult = c(0, 0.1)))+
  xlab("Years")+
  ylab("Passengers")+
  labs(
    title = "Air Passengers Path Chart (1949-1960)",
    x = "Year",
    y = "Number of Passengers",
    caption = "Data source: AirPassengers dataset"
  ) +Mytheme
```

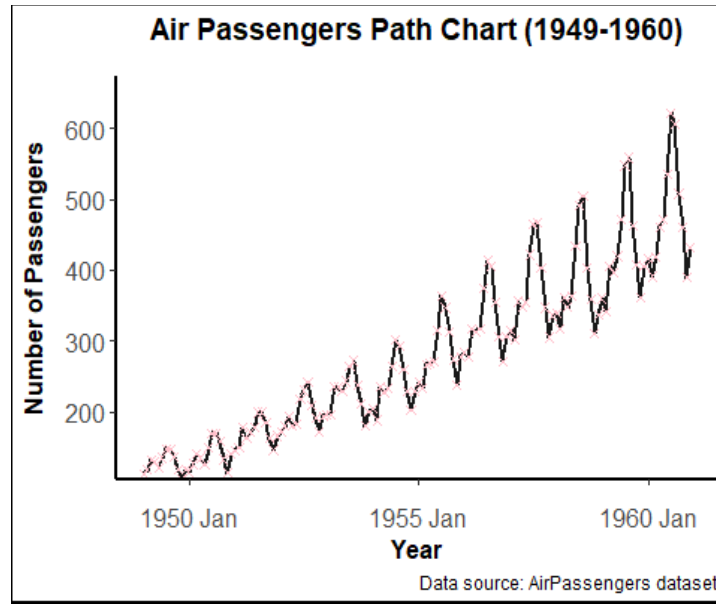


Figure 6. 10: Path Chart

With this path chart, you can watch the air passenger numbers flow smoothly from 1949 to 1960, each month connected in the order it happened. The pink-outlined blue crosses mark individual months distinctly, making it easy to see both the data points and the overall trajectory. Noticeable climbs in passenger numbers occur particularly around March to May each year, signaling the onset of peak travel seasons, while dips typically appear in late autumn and winter months. The expanded y-axis ensures the fluctuations are clearly visible without crowding. With clear axis labels and a neat theme, this chart offers a detailed and elegant view of how air travel volumes evolved over these twelve years.

6.11 Vertical Highlight

Highlighting key periods in a time series helps contextualize patterns and relate them to real-world events. In this chart, vertical dashed lines are drawn at specific dates using `geom_vline()` to call attention to February 1955 and May 1956. These lines (red dotted lines) act like visual bookmarks, guiding the reader to moments of interest which may be, perhaps, representing a policy change or a notable spike. `geom_point()` adds the scattered monthly values, allowing readers to correlate these highlights with fluctuations. The highlight lines are colored red to stand out against the background. These annotations are useful to mark events like policy changes, anomalies, or seasonal peaks. The combination of scattered points and vertical

references allows readers to visually correlate events with data shifts, enhancing narrative value and analytical storytelling by focusing attention on temporal milestones.

```
Tvline<-ap_df
Tvline%>%
  ggplot(aes(x=index,y=Passengers))+
  geom_point(fill="darkblue",shape=21,
             color="pink",alpha=.5)+
  geom_vline(xintercept =as.Date("1955-02-01"),
             color="red",linetype="dashed",
             linewidth=1)+
  geom_vline(xintercept =as.Date("1956-05-01"),
             color="red",linetype="dashed",
             linewidth=1)+
  xlab("Years")+
  ylab("Passengers")+
  labs(
    title = "Passengers Highlight (1955 Feb-1956 May)",
    x = "Year",
    y = "Number of Passengers",
    caption = "Data source: AirPassengers dataset"
  )+ Mytheme
```

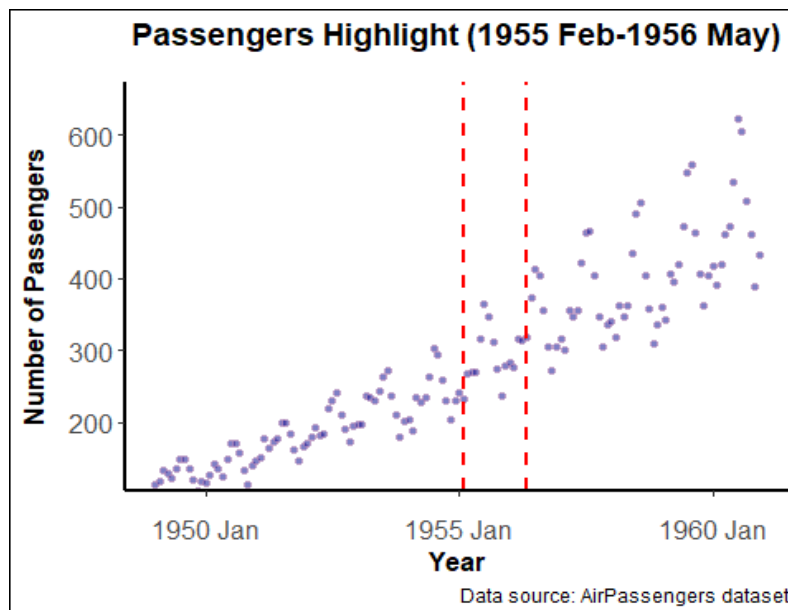


Figure 6. 11: Vertical Highlights

This scatter plot highlights monthly air passenger counts from 1949 to 1960, with each point marked by a dark blue fill and pink outline for clear visibility. Two vertical red dashed lines draw attention to key periods: February 1955 and May 1956. Between these dates, passenger numbers experienced notable changes, suggesting important shifts in travel patterns during that timeframe. The chart's clean design and clear labels keep the focus on these highlighted moments, helping viewers quickly identify and explore this critical interval within the overall air travel trend.

6.12 Rectangular Highlight

At times the key information is a specific period rather than individual points. A rectangular highlight chart uses shaded regions to draw attention to important intervals, such as peak travel seasons or anomalous periods. This makes trends easier to interpret by visually guiding the viewer to critical sections. Using `geom_rect()`, we can draw a rectangle over a specific data zone without hiding the underlying points. In this example, we focus on tree measurements from the `trees` dataset, drawing a red-bordered box to emphasize trees with volumes between 32 and 48 and heights from 74 to 80. The color aesthetic is driven by `Girth`, with a custom palette (blue, yellow, grey) provided through `scale_color_manual()`. This technique draws attention to clusters or outliers in a defined data range while retaining the plot's integrity. This technique is ideal for drawing attention to anomalies, thresholds, or areas of interest in bivariate data. The rectangle offers a simple, intuitive way to emphasize relationships or zones without clutter.

```
mytreez<-mytrees
p<-ggplot(mytreez,aes(x=Volume,y=Height, colour = Girth))+
  geom_point()+
  geom_rect(xmin=32,xmax=48,ymin=74,ymax=80,fill=NA,color="darkred",linewidth = 1.5)+
  scale_color_manual(values = c("blue","yellow","grey"))+
  ylab("Height")+
  xlab("Volume")+
  labs(title="Rectangular Highlight ",fill=NULL)+ Mytheme
print p
```

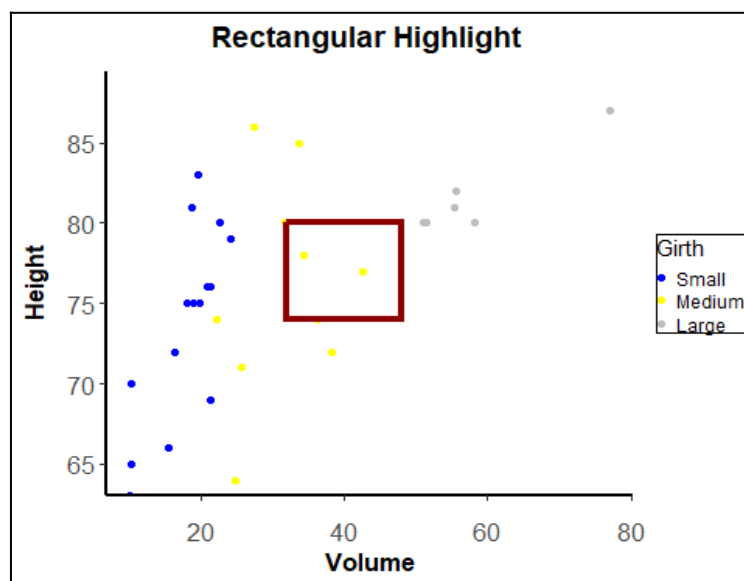


Figure 6. 12: *Rectangular Highlights*

Here, the scatter plot lets you explore tree data, with Height running up the y-axis, Volume along the x-axis, and colors (blue, yellow, grey) marking each Girth category. A prominent dark red rectangle frames the area between Volume values 32 to 48 and Height values 74 to 80, drawing attention to this specific cluster of data points. This visual highlight allows viewers to quickly focus on trees within that volume and height range, making it easier to spot patterns or outliers within the selected subset. The clean labels and color coding enhance clarity and make the chart both informative and visually engaging.

In time series too, you can make use of rectangular highlight to shift focus to a time window. In this example, the period 1955 to 1956 highlighted by drawing a light blue, semi-transparent rectangle using `geom_rect()`. This subtle overlay spans the entire vertical range, ensuring that all fluctuations within this interval remain visible. The use of transparency (`alpha = 0.3`) helps to guide the viewer's focus without overpowering the data.

```
# Highlights Area with geom_rect 1955-1956
highlight_start <- as.Date("1955-01-01")
highlight_end <- as.Date("1956-12-31")

ggplot(ap_df, aes(x = index, y = Passengers)) +
  geom_rect(aes(xmin = highlight_start,
               xmax = highlight_end, ymin = -Inf,
               ymax = Inf),
            fill = "lightblue", alpha = 0.3) +
  geom_line(color = "black") +
  labs(
    title = "Highlight Period 1955-1956",
    x = "Year",
    y = "Passengers"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold",
                              hjust = 0.5,
                              size = 14),
    plot.caption = element_text(hjust = 0),
    plot.background = element_rect(colour = "black",
                                   linewidth = 1)
  )
```

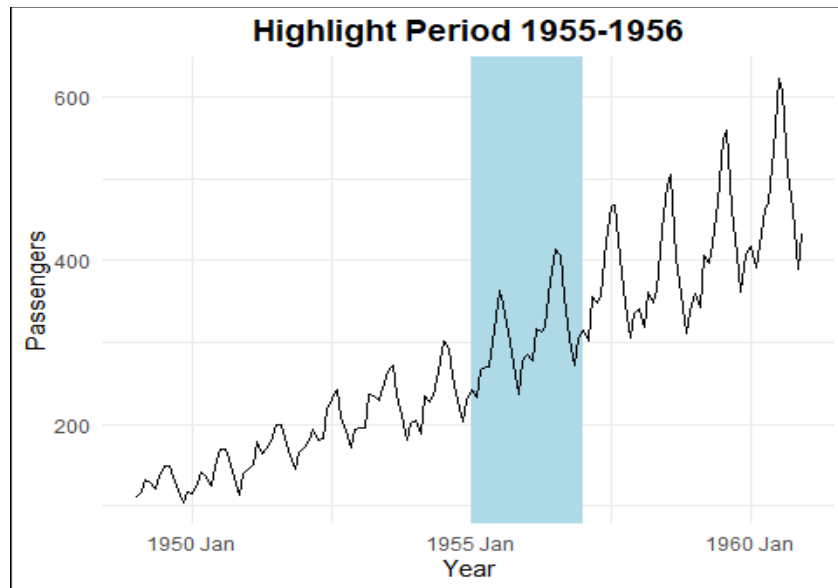


Figure 6. 13: *Highlight with Rectangular Shade*

Overlaid on a base time series line, this visualization aids in storytelling: you can visually isolate a window of significance, like when a new marketing strategy began or an external shock occurred, making it easier to examine its impact. It's subtle but impactful when used alongside a base time series plot, aiding narrative emphasis. The chart highlights the period from 1955 to 1956 with a soft light blue shaded rectangle, drawing immediate attention to this specific timeframe. Overlaid on the subtle highlight is a crisp black line tracing the monthly passenger numbers, showing how air travel evolved during these two years. This focused visual cue helps emphasize notable changes or patterns in passenger volume within that highlighted window, while the clean minimalist design and black border keep the viewer's focus squarely on the data's progression over time.

6.13 Annotated Chart

Annotated chart adds focus to a specific point in the scatterplot by using `annotate("point",.....)` to highlight a value visually. Annotations help direct the viewer's gaze to specific insights without changing the underlying data. In this example from the `trees` dataset, a large red ring is drawn around the point (38, 79) using `annotate("point",.....)`, making it the chart's focal element. The default scatterplot is maintained using `geom_point()`, and the color aesthetic varies by `Girth`. This red-ringed highlight provides a callout for a key observation, perhaps an outlier or a record-breaking value. When you're crafting narratives with data, such targeted annotations can transform raw plots into compelling visual arguments.

```

an<-ggplot(mytreez,aes(x=Volume,y=Height, colour = Girth))+
  geom_point()+
  annotate("point",x=38,y=79,size=35,shape=1,color="darkred")+
  scale_color_manual(values = c("blue", "yellow", "grey"))+
  ylab("Height")+
  xlab("Volume")+
  labs(title="Annotated Chart ",fill=NULL)+ Mytheme
print an

```

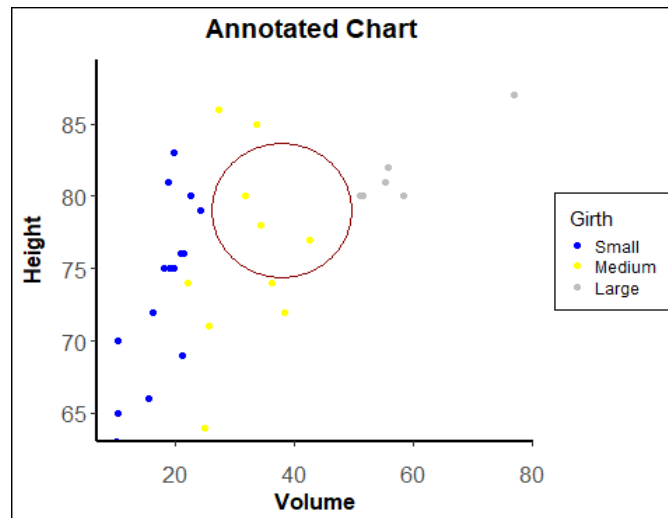


Figure 6. 14: Annotated Chart

As you can observe, this scatter plot maps out the relationship between `Volume` and `Volume`, while the `Girth` categories stand out in blue, yellow, and grey. A prominent dark red circle annotation highlights a specific data point at Volume 38 and Height 79, drawing the eye immediately to this notable observation. The clean axis labels and polished theme provide a clear and focused visual, making it easy to see both overall patterns and the specially marked point within the data.

6.14 Segmented Chart

Breaking data into distinct intervals can reveal patterns that are otherwise hidden. Segmented charts divide time into phases, allowing you to compare behavior across segments. This helps in identifying period-specific trends, shifts, or recurring cycles. Segmented chart in `ggplot2` uses `geom_segment()` to draw line segments between two points, a line that we may think of as a customizable arrow without the arrowhead. It's incredibly versatile: you can use it to draw lines between categories, show ranges, or annotate plots. It's often used for slope charts, dumbbell plots, or connection lines between elements. It's a building block for many creative visualizations in `ggplot2`.

Here, `geom_segment()` draws a thick black segment from (5,67) to (61,81) overlays the scatterplot, indicating a trend or directional emphasis. This is paired with color-coded points based on `Girth`, enriching visual storytelling. It's ideal for highlighting before-and-after states or ranges within a relationship.

```
seg<-ggplot(mytreez,aes(x=Volume,y=Height, colour = Girth))+
  geom_segment(x=5,y=67,xend=61,yend=81,color="black",linewidth = 1.5)+
  geom_point()+
  scale_color_manual(values = c("blue","yellow","grey"))+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  ylab("Height")+
  xlab("Volume")+
  labs(title="Segmented Chart ",fill=NULL)+ Mytheme
seg
```

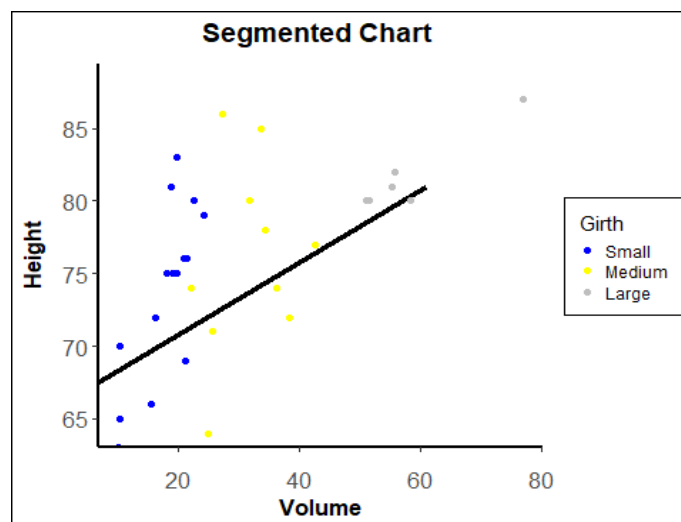


Figure 6. 15: Segmented Chart

Take a moment to see how a bold black segment moves from (Volume = 5, Height = 67) towards (Volume = 61, Height = 81), hinting at a consistent upward pattern. The carefully expanded y-axis gives breathing room above the highest points, while the polished theme and crisp axis labels create a clean, professional look that directs focus to both the individual points and the highlighted segment.

6.15 Radial Spoke chart

Data with cyclical or seasonal behavior benefits from a circular perspective. A radial spoke chart plots variables along axes radiating from a center, offering a visual snapshot of cycles and periodic patterns. This format is particularly helpful when examining seasonal trends or repeated patterns. Radial spoke charts use `geom_spoke()` to create spokes radiating from a center point. These spokes encode both magnitude (through radius) and direction, through `angle()`, offering a

rich representation of multidimensional data. In this example, the radius is computed using the Pythagorean formula combining **Height** and **Volume**, while **Angle** is an assigned polar coordinate. Each spoke's color reflects **Girth**, adding another layer of information. The plot is transformed into a circular layout, simulating motion or directional growth. Labels, axis formatting, and a title are added to the plot. This type of plot is suitable for visualizing vector-like data such as wind patterns, mechanical forces, or directional preferences. The spoke chart excels at showcasing both the strength and angle of an attribute simultaneously.

```
ggplot(mytreez, aes(x = 0, y = 0, angle = Angle,
                    radius = sqrt(Height^2 + Volume^2),
                    color = Girth))+
  geom_spoke(size = 1.2)+ # Spokes
  geom_point(aes(x = sqrt(Height^2 + Volume^2) * cos(Angle),
                 y = sqrt(Height^2 + Volume^2) * sin(Angle)), size = 3)+
  ylab("Height")+
  xlab("Volume")+
  scale_y_continuous(labels = abs)+
  scale_x_continuous(labels = abs)+
  labs(title="Spoke Chart ", fill=NULL)+ Mytheme
```

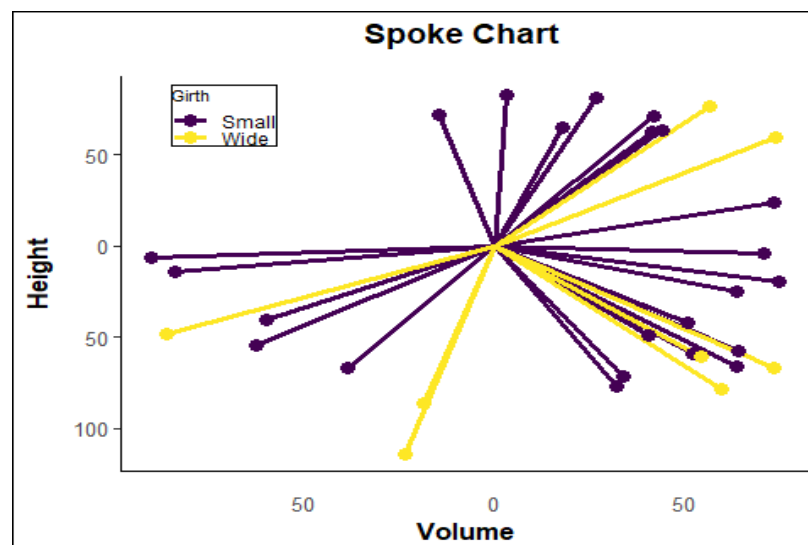


Figure 6. 16: Radial Spokes Chart

Notice how the spoke chart places height and volume along radiating lines, inviting you to explore their relationship in a circular view. Each spoke's length corresponds to the combined magnitude of height and volume, while its angle positions it uniquely around the circle, colored by girth category to add a visual grouping. Notably, trees in the "Wide" girth category (grey) extend the farthest from the center, indicating larger overall size, while "Small" girth trees (blue) have shorter spokes, reflecting smaller combined height and volume. The points at the tips of the

spokes highlight individual trees, making it easy to compare their size and girth in a dynamic, circular layout. This visualization invites you to explore the multidimensional data in an engaging and intuitive way, blending magnitude and direction seamlessly.

6.16 Normalized Spokes Chart

To compare trends proportionally rather than absolutely, a normalized spoke chart adjusts values along each axis. This allows you to see relative changes across categories, emphasizing patterns that might be hidden by raw magnitudes. An enhanced spoke chart normalizes the radial length using transformed values, such as square-rooted **Girth**, for improved comparison across data points. Rather than originating from a common center, each spoke emerges from the data point's location, increasing readability of individual-level variation. The angle is dynamically computed from the arctangent of **Height/Volume**, ensuring accuracy in orientation. The `scale_color_gradient()` emphasizes contrast in the magnitude of **GirthS**, creating an appealing heatmap effect. This chart is well-suited for comparing directional strengths across non-centralized units, like examining growth direction per tree. By transforming and standardizing radius, it maintains interpretability while avoiding skewed visuals from extreme outliers.

```
mytreez <- mytreez %>%
  mutate(Angle = rad2deg(atan(Height/Volume)))
mytreez <- mytreez %>%
  mutate(GirthS = sqrt(0.1*mytreez$Girth))
ggplot(mytreez, aes(x = Volume, y = Height, color=GirthS))+
  geom_point(size=1)+
  geom_spoke(aes(angle=Angle), radius=mytreez$GirthS)+ # Spokes extending from
each (Girth, Height) point
  scale_color_gradient(low = "blue", high = "red" )+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  ylab("Height")+
  xlab("Volume")+
  labs(title="Spoke Chart", fill=NULL)+ Mytheme
```

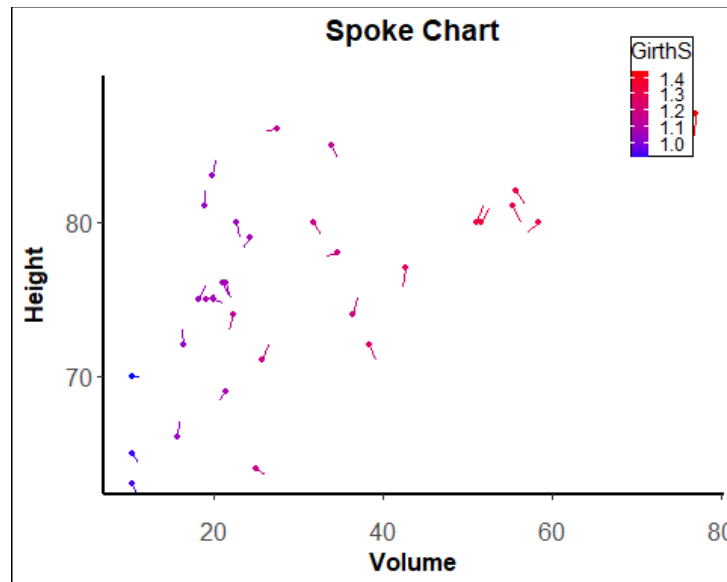


Figure 6. 17: Normalized Spokes Chart

Take a look at this spoke chart , it brilliantly brings to life the relationship between tree volume and height, with a rich color gradient and varying spoke lengths adding depth and visual interest. Each point represents a tree's volume and height, colored along a gradient from blue to red that reflects the square root of its girth, cooler blues indicate smaller girth, while warmer reds signify larger girth. From each point, a spoke extends at an angle calculated by the ratio of height to volume, with the length of each spoke scaled according to the girth size, visually emphasizing trees with bigger girths. The result is a dynamic interplay where both direction (angle) and magnitude (spoke length and color intensity) combine to give a multidimensional sense of size and shape across the tree samples. This chart allows you to quickly compare how girth influences the spatial relationship between height and volume in a visually compelling way.

6.17 Stream chart

For datasets where multiple components contribute to a total, a stream chart shows how these contributions change over time. By stacking areas, it visualizes both the overall trend and the relative dynamics of each component, making shifts and patterns immediately apparent. To make a stream chart, we reshape the data to a time-like structure using a dummy Year column and spread categories into columns. The `ggstream` package is used to produce smooth flowing shapes representing each Girth category. Each flow shows the change in category size (`Count`) across artificial time, ideal for simulating dynamics or trends. `geom_stream()` is applied with aesthetic mapping to x (`Year`) and y (`Count`), `fill = "Girth"`.

```
library(ggstream)
mytrees$Year <- rep(2020:2022, length.out = nrow(mytrees)) # simulate time
ggplot(mytrees, aes(x = Year, y = Count, fill = Girth)) +
  geom_stream() +
  theme_minimal() +
  labs(title = "Stream Chart",
       x = "Year", y = "Count", fill = "Girth")
```

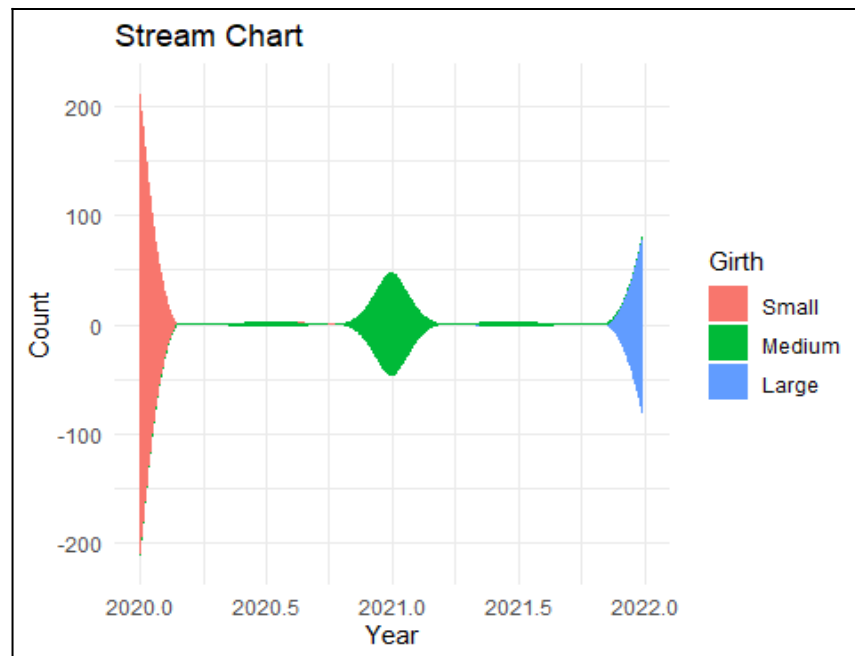


Figure 6. 18: Stream Chart

You will see that this stream chart paints a flowing picture of how the counts of trees, grouped by their Girth categories, evolve over the years 2020 to 2022. The different colored streams represent the categories, Small, Medium, and Large, each gently rising and falling as the years progress. The smooth curves make it easy to spot trends: for example, you might notice one category widening, indicating growth in tree counts, while another narrows, showing a decline. The flowing nature of the chart helps connect changes over time into a continuous narrative, making it intuitive to follow how each Girth group contributes to the overall pattern across the three years. The clean minimalist theme keeps the focus on the data, with clear labels guiding your understanding of both time and tree size composition.

6.18 Slope Chart

When comparing two time points, a slope chart connects values with lines, highlighting increases, decreases, and relative changes. This makes it easy to see which categories have grown or declined over time and facilitates direct comparisons. Using `geom_path()`, it draws lines

connecting each entity's values at two time periods. This example tracks **AirPassengers** over time, drawing smooth transitions using a single line for all observations. Slope direction (upward or downward) immediately reveals the trend direction and intensity. This type of visualization is especially powerful in comparing before-and-after effects, ranking shifts, or showcasing policy impact. Minimalist in design, it excels in clarity and elegance. The centered title and clean theme further improve interpretation by reducing clutter. Although simple in appearance, slope charts are remarkably effective in comparative storytelling.

```
ggplot(ap_df, aes(x = index, y = Passengers)) +  
  geom_path(color = "blue", linewidth = 1.5) +  
  labs(  
    title = "Air Passengers Over Time",  
    x = "Year",  
    y = "Passengers"  
  ) +  
  theme_minimal() +  
  theme(  
    plot.title = element_text(face = "bold",  
                               hjust = 0.5,  
                               size = 14),  
    plot.caption = element_text(hjust = 0),  
    plot.background = element_rect(colour = "black",  
                                    linewidth = 1)  
  )
```

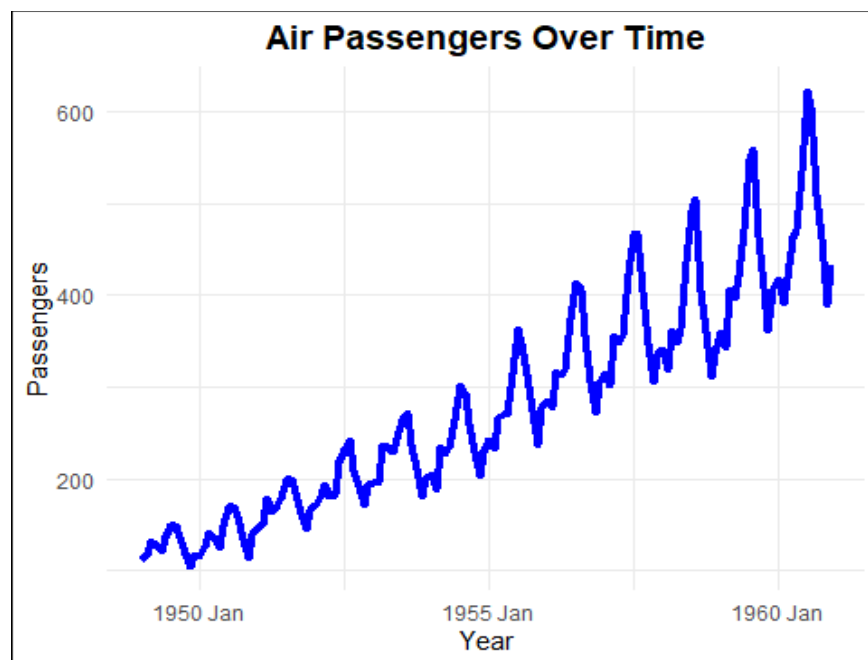


Figure 6. 19: Slope Chart

As you can observe the chart vividly traces the journey of air passengers over time with a smooth, bold blue path that highlights both steady growth and seasonal shifts from 1949 through 1960. You can clearly see how passenger numbers rise and fall within each year, reflecting busy and slow travel months, yet the overall trajectory is upward. Notably, sharper increases occur during spring and summer months, while dips appear consistently around late autumn and winter, revealing predictable seasonal patterns. The clean minimalist style and crisp black border keep the focus squarely on the evolving passenger trends, making it easy to follow the gradual rise in air travel popularity over this period.

6.19 Line Chart

Line charts are a classic way to follow trends over time. By connecting data points sequentially, a line chart shows the direction, magnitude, and continuity of changes, helping you identify long-term trends, seasonality, or anomalies in the data. Line charts are commonly used to visualize trends, and while they typically map changes across time, they're also quite effective for any continuous x-axis variable. In this illustration, we plot **Height** against **Volume** using `geom_line()` to connect points in the order they appear. Although the trees dataset doesn't naturally follow a temporal sequence, line charts can still show us how one variable responds to another. This specific chart connects observations for all trees without grouping, meaning the line snakes through all data points as a single path. It's more exploratory than explanatory but still visually engaging. Think of it as a rough visual approximation of tree growth patterns. If you're trying to spot abrupt height jumps with increasing volume, this line chart gives you that bird's eye view. It's quick to build, easy to read, and sets the foundation for more refined analyses. Line charts are excellent for revealing upward or downward trends, peaks, dips, or steady behavior.

```
ggplot(mytreez, aes(x = Volume, y = Height, color = Girth)) +  
  geom_line(linewidth = 1.2) +  
  labs(title = "Line Chart", x = "Volume", y = "Height") +  
  scale_color_manual(values = c("Small" = "blue", "Wide" = "yellow")) +  
  scale_y_continuous(expand = expansion(mult = c(0, 0.1))) +  
  scale_x_continuous(expand = expansion(0)) +  
  guides(color = guide_legend(keywidth = 0.8,  
                             keyheight = 0.8,  
                             default.unit = "cm"))+Mytheme
```

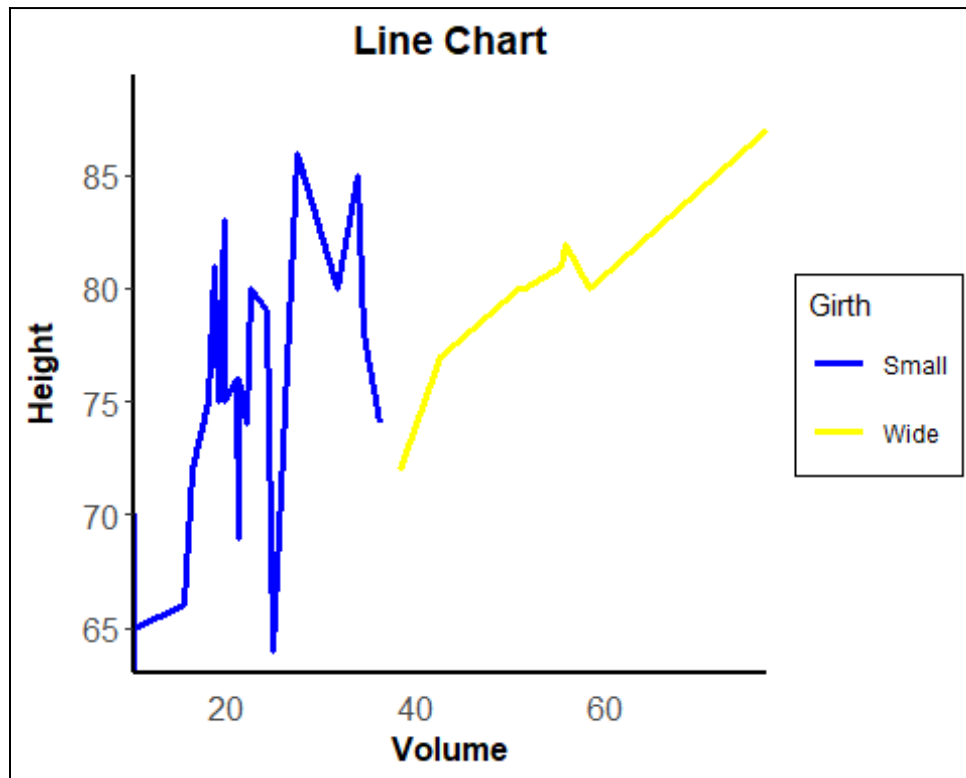


Figure 6. 20: Line Chart

This visualization elegantly connects points of tree Volume and Height, using color to distinguish between “Small” and “Wide” girth categories. The bold lines guide your eyes through how Height changes with Volume for each girth group, revealing distinct patterns: the “Small” girth line in deep blue generally stays lower, indicating smaller volumes and heights, while the “Wide” girth in bright yellow climbs higher, reflecting larger trees. The clean axes and gentle padding let the data breathe without clutter, and the thoughtfully sized legend keys make it easy to identify each girth category at a glance. Overall, the chart paints a clear picture of how girth relates to tree size, emphasizing differences across these groups.

6.20 Faceted Line Chart

When multiple groups need to be compared, faceted line charts split the data into panels. This allows you to observe trends side by side without overcrowding the chart, making differences and similarities between groups easy to spot. In this case, we split our line chart by tree **Girth**, either "S **Small** " or "**Wide** ", using `facet_wrap(~Girth,...)`. This approach creates two side-by-side plots, each showing the relationship between **Volume** and **Height** for that specific group. The line within each facet traces how tree height changes with **Volume**, but this time it does so only

for `trees` within that category. The power of faceting lies in contrast: side-by-side plots make it easier to spot differences in slope, curvature, or spread. For instance, `Wide trees` may show steeper lines, suggesting faster height growth with volume. Faceting gives clarity to group comparisons without the clutter of overlapping lines.

```
ggplot(mytreez, aes(x = Volume, y = Height, colour = Girth)) +
  geom_line(linewidth = 1.2) +
  scale_color_manual(values = c("Small" = "blue", "Wide" = "yellow")) +
  facet_wrap(~Girth, labeller = as_labeller(setNames(my_xlab, levels(mytreez$Girth))))
+
  labs(title = "Faceted Line Chart", x = "Volume", y = "Height") + Mytheme
```

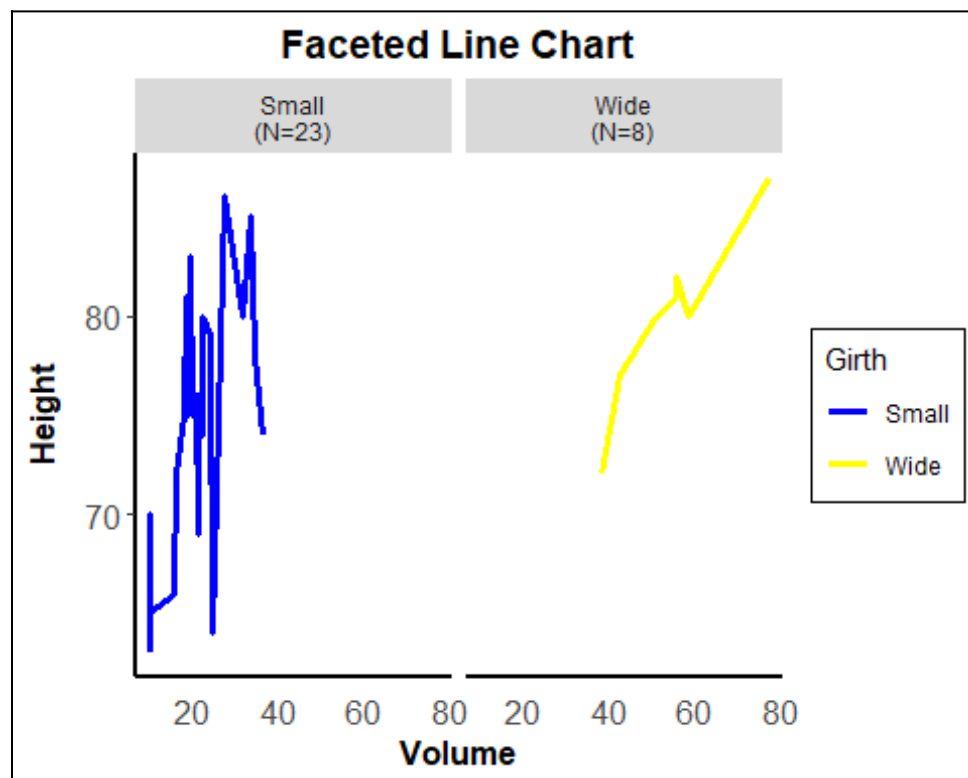


Figure 6. 21: Time Series Plot

Key takeaway here is that faceting breaks down the relationship between Volume and Height by girth categories, presenting each girth group in its own panel for clearer comparison. The smooth blue line in the “Small” girth facet shows a relatively modest increase in Height as Volume grows, while the “Wide” girth panel in yellow reveals a steeper rise, indicating larger trees with greater volume and height. Faceting isolates these patterns side-by-side, making it easier to see how tree size dynamics differ between girth classes without visual overlap. The clean labels and

consistent axes help maintain focus, and the minimalist styling ensures the data stands out. This layout enhances understanding by spotlighting distinct trends within each girth group individually.

6.21 Smoothed Line Chart

Raw data can be noisy, making patterns hard to see and line chart can feel too jagged or literal. Smoothed line charts apply a smoothing technique to highlight the underlying trend, filtering out short-term fluctuations. This approach helps in forecasting, trend analysis, and revealing consistent patterns. `geom_smooth()` fits a smooth trend line (by default using LOESS for small datasets) that follows the shape of your data but softens the edges. `method = "loess"` and `se = FALSE` are key arguments for a smoothened trend line. This curve doesn't connect points in sequence but instead fits a model that summarizes the overall pattern. Here, we plot a smooth curve showing how `Height` evolves with `Volume` across all `trees`. The shaded area around the line represents the confidence interval, helping us visualize uncertainty. This smoothed line acts like a visual average, it tells a story without getting bogged down in individual variation. It's ideal for presentations when you want to highlight a clean, general trend rather than all the messy details. It can be used together with `geom_point()` to add scatter points to visualize raw data for the series used in the graphics.

```
ggplot(mytreez, aes(x = Volume, y = Height, colour = Girth)) +  
  geom_smooth(se = FALSE, method = "loess", fill = "lightblue", linewidth = 1.2) +  
  scale_color_manual(values = c("Small" = "blue", "Wide" = "yellow")) +  
  labs(title = "Smoothed Line Chart", x = "Volume", y = "Height") + Mytheme
```

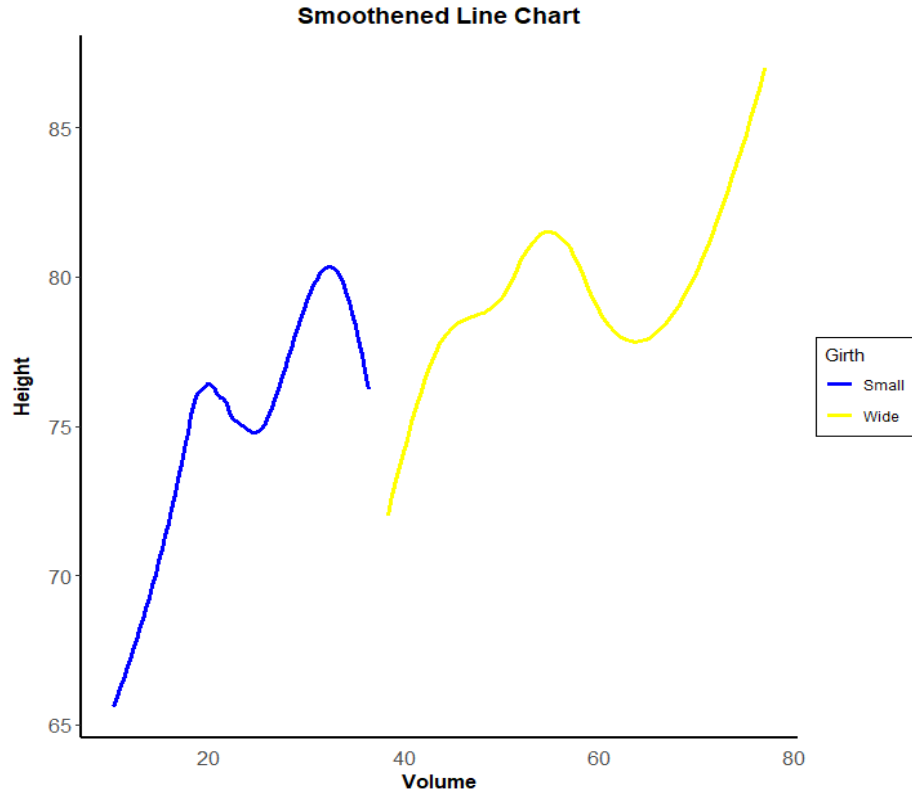


Figure 6. 22: *Smoothed Line Chart*

This smoothed line chart elegantly captures the relationship between **Volume** and **Height** across different girth categories, using a gentle **loess** curve to reveal underlying trends. The blue line for the **Small** girth group shows a gradual increase in **Height** as **Volume** grows, with a narrow confidence band highlighting consistent patterns. In contrast, the **Wide** girth category in yellow displays a steeper rise, indicating that trees with larger girth tend to gain **Height** more sharply with increasing **Volume**.

Building on the previous chart, faceting takes it a step further. With `facet_wrap()`, you are able to generate separate smoothed curves for **Small** and **Wide** trees.

```
FA1<-ggplot(mytreeez, aes(x = Volume, y = Height, colour = Girth)) +
  geom_smooth(se = FALSE, method = "loess", fill = "lightblue", linewidth = 1.2) +
  facet_wrap(vars(Girth))+
  scale_color_manual(values = c("Small" = "blue", "Wide" = "yellow")) +
  labs(title = "One Row Facets" ,x = "Volume", y = "Height") +Mytheme

FA2<-ggplot(mytreeez, aes(x = Volume, y = Height, colour = Girth)) +
  geom_smooth(se = FALSE, method = "loess", fill = "lightblue", linewidth = 1.2) +
  facet_wrap(vars(Girth),nrow = 2)+
  scale_color_manual(values = c("Small" = "blue", "Wide" = "yellow")) +
  labs(title = "Two Row Facets", x = "Volume", y = "Height") +Mytheme
FA1 + FA2 + plot_annotation(title = "Smoothed Line Charts",caption = "1 row vs 2
rows",
```

```
theme = theme(plot.title = element_text(size = 12, face = "bold"))
)
```

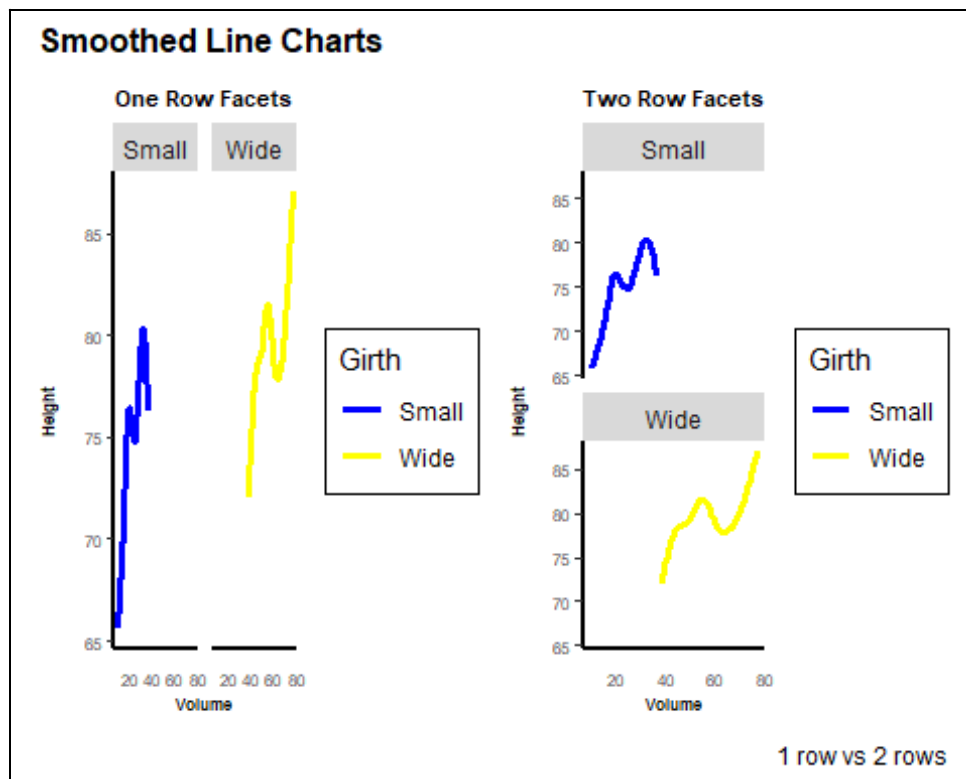


Figure 6.23: Faceted Smoothed Line Charts

Each panel reflects its own trend, independently fitted using `loess` smoothing. This is super helpful when you suspect that different subgroups behave differently, and in real-world data, they often do. These curves let you visually assess whether, say, smaller-girth trees plateau earlier in height, while wider ones grow steadily.

6.22 Regression Line

When exploring relationships between variables, it is often useful to go beyond individual data points and uncover the underlying trend that connects them. A regression line serves exactly this purpose by fitting a line that best describes how one variable changes in response to another. Whether applied to scatter or point plots, the line highlights direction, strength, and consistency of association, making it easier to see the “drift” or departure from one point to another. More than just a visual aid, a regression line embodies a predictive model, it captures patterns over time or volume, explains changes between variables, and provides a framework for estimating

future values. In essence, it transforms scattered data into a coherent story of relationships, trends, and potential forecasts.

In the code, `geom_point()` plots the individual data points representing each tree's height and volume, giving a clear picture of the raw observations. The `geom_smooth()` function adds a smooth curve or line based on the data; here, with `method = "lm"`, it fits a linear model (a straight regression line) to the data, showing the estimated relationship between `Height` and `Volume`. Setting `se = FALSE` removes the confidence interval shading around the regression line for a cleaner visual focus on the trend itself.

```
ggplot(mytreez, aes(x = Volume, y = Height, colour = Girth)) +  
  geom_point()+  
  geom_smooth(se = FALSE, method = "lm", fill = "lightblue", linewidth = 1.2) +  
  facet_wrap(vars(Girth))+  
  scale_color_manual(values = c("Small" = "blue", "Wide" = "yellow")) +  
  labs(title = "Regression Line", x = "Volume", y = "Height") + Mytheme
```

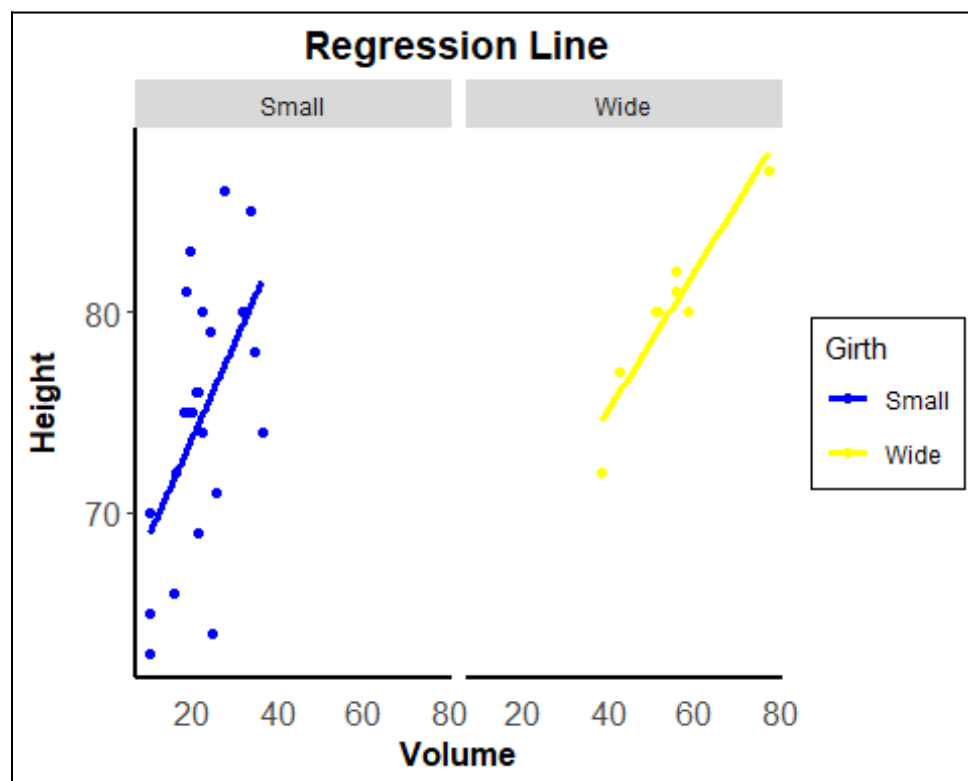


Figure 6. 24: Regression Line

Looking at the output, the plot clearly shows two regression lines, one for `Small` girth trees (blue) and one for `Wide` girth trees (yellow). Both lines have a positive slope, indicating that as volume increases, height also increases. However, the slope for the `Wide` girth trees is steeper, meaning

height grows faster with volume compared to the **Small** girth trees. This difference in gradient effectively highlights how tree size categories exhibit different growth rates over volume (which can be interpreted as a proxy for time or size progression). For example, a **Wide** girth tree's height increases more rapidly with volume than a **Small** girth tree, making the yellow line steeper and showing a stronger change over volume.

CHAPTER 7: RELATIONSHIP

Correlation charts explore the relationship between two or more variables to determine how they influence each other. The graphics are used to highlight interaction and interconnectivity of data points in a manner that provide logical relationships among variables. Visual relationships are implied using size and color intensity in a chart. These charts are used in statistics, economics, and scientific research to test hypotheses and model behavior. Common charts include scatter plots, bubble charts, heatmaps, and correlation matrices. These tools help identify positive, negative, or no relationships between variables.

Scatter plots are the most direct way to visualize correlations, showing how one variable changes in relation to another. Bubble charts add a third variable through the size of the markers. Heatmaps can show the strength of correlation using color intensity in a grid. Correlation matrices provide an overview of pairwise correlations between multiple variables, often color-coded for clarity.

7.1 Scatter Plot

Think of a scatter plot as a picture made of dots, where each dot shows one observation based on two numbers you're comparing. The spot where a dot lands depends on both values, one along the bottom and one up the side. It's a handy way to spot patterns, trends, or possible connections between the two items you're measuring. The plot shows individual tree measurements as solid blue dots (`pch = 16`), with a slight jitter applied on both the x (`Volume`) and y (`Height`) axes to prevent overlapping points and reveal data density more clearly. The y-axis is limited to the range 60 to 90 to better focus on this specific height interval, and additional tick marks are manually added at every 5 units on the y-axis for easier interpretation of values.

The `scatterplot()` function from the `car` package is used here, providing fine control over plot features. For instance, smoothing lines and boxplots are disabled (`smooth = FALSE`, `boxplots = FALSE`) to maintain a clean view of raw data points without trend lines or marginal distributions. The plot title "Scatter Plot" clearly indicates its purpose, while the absence of grid lines (`grid = FALSE`) keeps the visualization uncluttered.

```
library(grid)
library(car)
library(carData)
scatterplot(Height ~ Volume,
            data = trees,
```

```

jitter = list(x = 1, y = 1),
smooth = FALSE,
lwd = 1,
main = "Scatter Plot",
grid = FALSE,
boxplots = FALSE,
col = "blue",      # point color
pch = 16,          # solid circles
ylim = c(60, 90)) # y-axis limits to show more range
# Add more y-axis ticks manually
axis(2, at = seq(60, 90, by = 5)) # adjust the interval

```

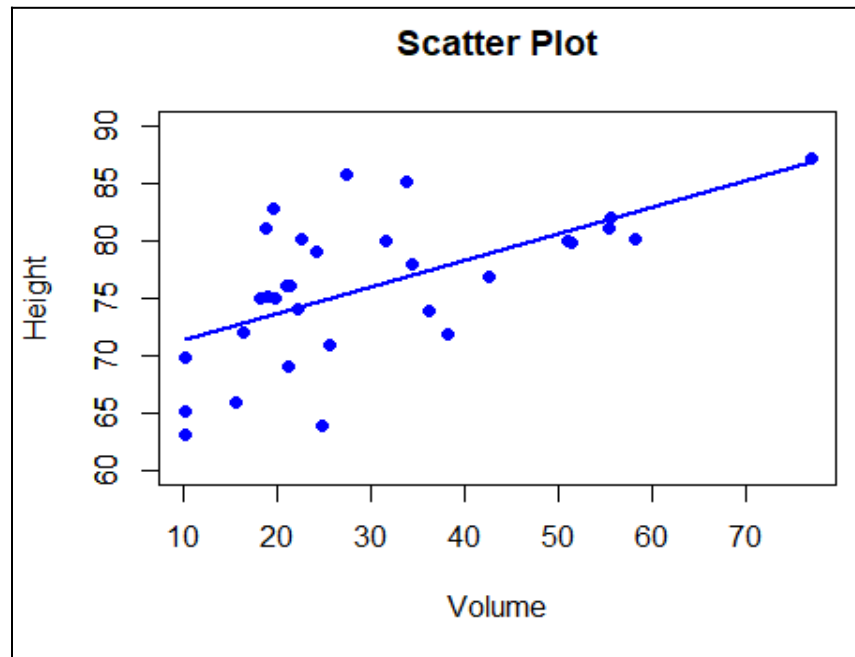


Figure 7. 1: Scatter Plot

In the output, the scatter plot reveals how **Height** varies with **Volume** of the trees. For example, you can observe that many trees with volumes around 40 to 50 have heights clustered near 70 to 80 units. The jittering makes it easy to notice that some values overlap but are actually distinct, and the manual y-axis ticks make it easier to read specific height values within the zoomed range. Overall, this plot effectively communicates the dispersion and potential correlation between **Height** and **Volume** in the dataset.

If you need to show the correlation coefficient in your scatter plot, ggplot2 coding allows you to do caption in the chart. It takes almost similar steps in building a chart with such captions. The `geom_point()` function adds individual data points colored in teal (`color = "#009999"`) with moderate transparency (`alpha = 0.6`), which helps to reveal data density and overlapping points. The size of points (`size = 3`) makes them clearly visible without clutter. The

`geom_smooth()` function fits a linear model (`method = "lm"`) to the data and adds a dashed black line representing this fit; the confidence interval shading is turned off (`se = FALSE`) to keep the focus on the regression trend. The plot title and subtitle include the computed correlation value (`cor_value`), providing immediate statistical context to what the visual shows.

```
library(gridExtra)
library(ggcorrplot)
cor_data <- cor(trees[, c("Height", "Volume")])
cor_value <- round(cor_data[1, 2], 2)
scatter_plot <- ggplot(trees, aes(x = Volume, y = Height))+
  geom_point(color = "#009999", size = 3, alpha = 0.6) +
  geom_smooth(method = "lm", se = FALSE,
             color = "black", linetype = "dashed") +
  labs(title = "Scatter Plot: Height vs Volume",
       subtitle = paste("Correlation =", cor_value),
       x = "Volume", y = "Height")+Mytheme
scatter_plot
```

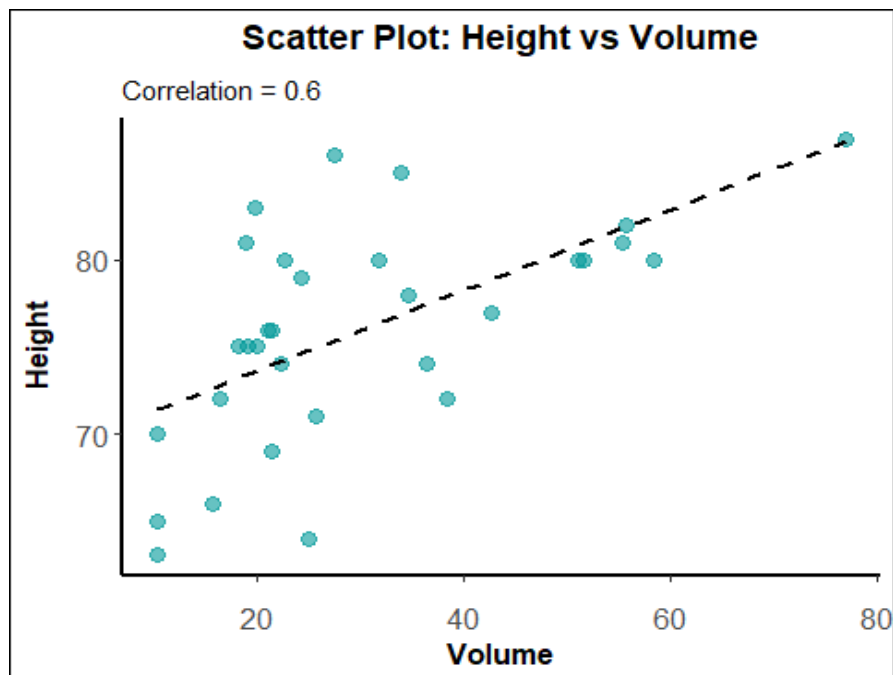


Figure 7. 2: Scatter Plot with Correlation Index

You can observe from the figure that there is a strong positive relationship between tree Volume and Height, confirmed by the correlation value of 0.6 displayed in the subtitle. This means that changes in volume explain about 60% of the variation in height, making volume a highly reliable predictor. As volume increases, height tends to rise consistently, with most points falling close to the fitted dashed regression line. For instance, trees with volumes around 10 cubic units are typically near 70 feet tall, while those exceeding 30 cubic units often surpass 80 feet in height.

The tight clustering of points along the line shows very little scatter, indicating minimal unexplained variation and reinforcing the strength of this association.

7.2 Bubble Chart

When exploring complex datasets, it can be difficult to fully grasp how multiple variables interact using simple charts. Traditional scatter plots reveal the relationship between two variables, but often leave out important dimensions of the data. A bubble chart addresses this limitation by adding an additional layer of information: the size of each point, which represents a third variable, while color can indicate categories. This multidimensional approach allows analysts to detect patterns, clusters, or outliers across groups, offering a richer understanding of how different observations relate to one another.

A bubble chart is essentially an enhanced scatter plot where each point represents a data observation, its position determined by two numeric variables, and its size corresponding to a third numeric attribute. Often, color is also used to differentiate categorical groups, adding yet another layer of insight. The result is a visually intuitive representation of three to four dimensions of data, making it easier to interpret complex relationships and distributions within the dataset.

In R, a bubble chart can be created using the `ggplot2` package. The `geom_point()` function plots each observation, with the size aesthetic mapped to the variable representing magnitude, for example, `Volume` in a dataset of tree measurements, and color mapped to categorical groups, such as Girth categories. Partial transparency (`alpha= 0.6`) helps clarify overlapping points, while `scale_size(range= c(3, 10))` ensures that bubble sizes are visually distinct and proportional to their values. Colors can be manually assigned to categories using `scale_color_manual()`, for instance, mapping Small to grey, Medium to greenish (`#95B809`), and Large to yellowish (`#FED966`). To avoid extra spacing around axes, `scale_y_continuous(expand = expansion(0))` is applied.

```
mytrees <- trees
# Categorize Girth
mytrees <- mytrees %>%
  mutate(Girth = cut(Girth,
                     breaks = 3,
                     labels = c("Small", "Medium", "Large")))
mytreez<-mytrees
```

```
ggplot(mytrees, aes(x = Height,
                    y = Volume,
                    size = 0.5,
                    color = Girth))+
  geom_point(alpha = 0.6) +
  scale_size(range = c(3, 10)) +
  scale_color_manual(values = c("grey", "#95B809", "#FED966"))+
  scale_y_continuous(expand = expansion(0))+
  xlab("Volume")+
  ylab("Height")+
  labs(title="Bubble Chart ", fill="Girth")+ Mytheme
  guides(fill=guide_legend(keywidth=0.05,
                           keyheight=0.05,
                           default.unit="cm")) # changes size of legend box
```

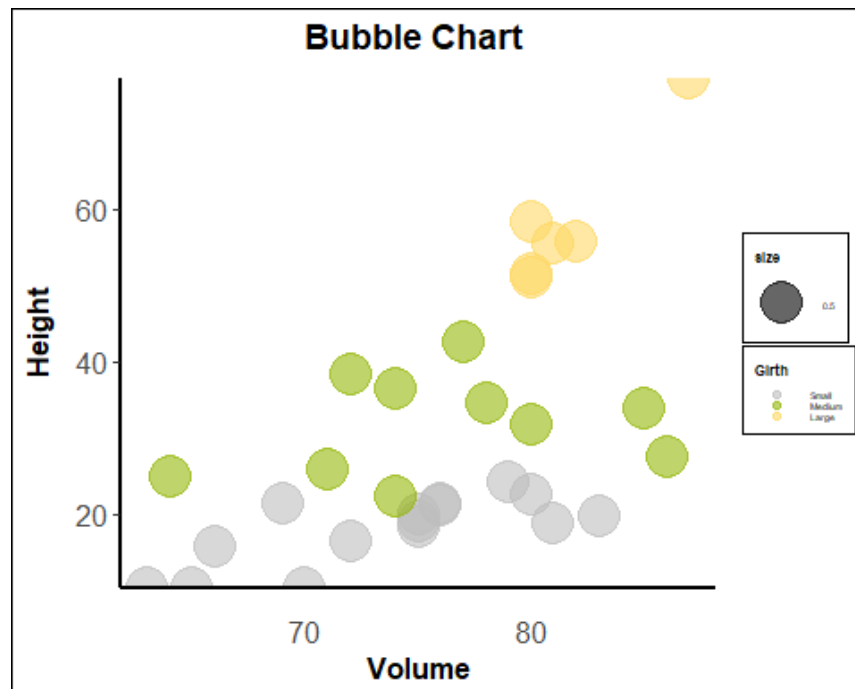


Figure 7. 3: Bubble Chartt

In this figure, you can clearly see how tree height and volume vary according to the girth categories of small, medium, and large. For trees with small girths shown in grey, there is no clear relationship between height and volume since the points appear scattered, with some shorter trees having relatively higher volumes while others of similar height have much smaller volumes. Medium girths represented in green display a moderately positive relationship in which height tends to increase alongside volume, although there is still some noticeable variation. Large girths shown in yellow exhibit a strongly positive relationship where taller trees almost always correspond to larger volumes, forming a distinct upward trend. This visualization distinguishes the groups effectively and also makes it clear that girth plays an important role in

shaping the relationship between height and volume, revealing how size influences growth patterns in a consistent way for certain categories.

7.3 Correlation Matrix

Understanding relationships between multiple variables is a critical step in data analysis, yet large datasets can make it challenging to quickly identify which variables are strongly related. A correlation heatmap provides an intuitive and visually compelling solution by summarizing these relationships in a single graphic. By representing correlations as colored circles, it allows analysts to immediately see which variables move together positively, negatively, or show little to no relationship, making complex patterns more accessible at a glance.

A correlation heatmap is a graphical representation of the correlation matrix of a dataset, where each cell corresponds to the correlation between a pair of variables. The strength and direction of each correlation are conveyed through color and size, with larger, more vividly colored circles indicating stronger relationships. Positive correlations are typically represented in one color, negative correlations in another, and weak or negligible correlations in a neutral tone. This visualization makes it easy to identify potential predictors, clusters of related variables, or unexpected associations that warrant further investigation.

In R, such a heatmap can be created using the `ggcorrplot()` function. The function takes a correlation matrix, for example, `cor_data`, as input and visualizes it using circles when `method = "circle"`, is specified. Enabling `lab = TRUE` adds the exact correlation coefficients on each circle, which enhances interpretability and allows precise comparisons between variables. The `colors` argument defines a gradient, often from red (strong negative correlation) through white (no correlation) to green (strong positive correlation), providing an immediate visual cue of relationship strength and direction.

```
# Correlation Heatmap
heatmap_plot <- ggcorrplot(cor_data,
                           method = "circle",
                           lab = TRUE,
                           colors = c("#D73027", "white", "#1A9850"),
                           title = "Correlation Heatmap",
                           ggtheme = theme_minimal())
heatmap_plot
```

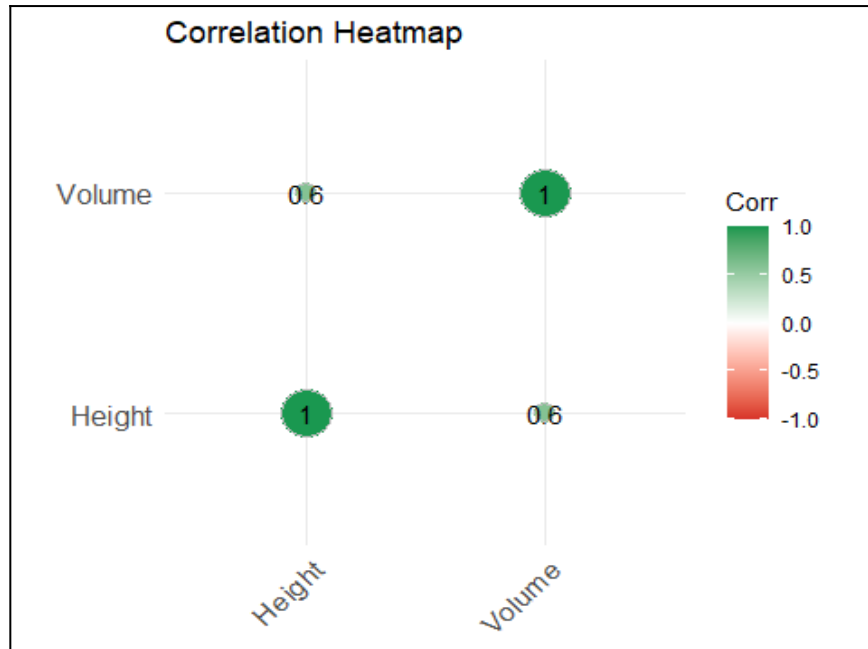


Figure 7. 4: Correlation Matrix

This correlation heatmap provides a clear visual summary of the strength and direction of the relationship between height and volume in the dataset. The single correlation cell shows a strong positive value of 0.87, indicated by the deep green color, which signals that as tree height increases, tree volume tends to rise consistently. The circular marker not only reinforces the magnitude of this association but also highlights its positive nature through its placement along the green end of the color scale. The presence of such a high correlation suggests that volume and height grow in tandem, meaning taller trees in this dataset are almost always larger in volume, and the visual representation makes this connection immediately apparent.

7.4 Time Series Correlation Chart

Analyzing time series data often requires understanding how values at one point in time relate to those at other points, especially when looking for seasonal patterns or recurring trends. A correlation matrix for time series provides an effective way to uncover these temporal relationships by summarizing how different time periods co-vary. This visualization allows analysts to quickly detect months or periods that move together positively or inversely, offering insights into cyclic behaviors, such as travel demand peaks and troughs over a year.

A time series correlation matrix is a grid-like representation of correlations between variables measured at different time points. In the context of monthly data, each cell shows the correlation

between two months, with the strength and direction of the relationship conveyed through color and, optionally, numeric labels. Strong positive correlations indicate that passenger counts (or other metrics) tend to rise and fall together across the months, while negative correlations highlight months that exhibit opposite trends. This approach is especially useful for identifying seasonal dependencies, uncovering periods with similar behaviors, and supporting forecasting and planning decisions.

In R, creating such a correlation matrix begins with transforming the time series data into a suitable structure. For example, the `AirPassengers` dataset, originally in long format, can be reshaped into a wide format using `pivot_wider()`, so that each month becomes a separate column. This allows correlations to be computed across months, excluding the `Year` column to focus purely on monthly passenger counts. The correlation matrix `cor_data` is then visualized using `ggcorr()` from the `GGally` package. By setting `label = TRUE` and `label_round = 2`, the exact correlation coefficients appear inside each cell with two decimal precision and controlled text size for readability. A color scale from red (strong negative correlation) through white (no correlation) to blue (strong positive correlation) visually emphasizes the strength and direction of monthly relationships.

```
library(tidyr)
library(lubridate)
library(GGally)
ap_df <- data.frame(
  Year = floor(time(AirPassengers)),
  Month = cycle(AirPassengers),
  Passengers = as.numeric(AirPassengers))
# Spread months into columns
ap_wide <- ap_df %>%
  mutate(Month = month.abb[Month]) %>%
  pivot_wider(names_from = Month, values_from = Passengers)
# Remove year for correlation matrix
cor_data <- ap_wide %>% select(-Year)
ggcorr(cor_data, label = TRUE, label_round = 2, label_size = 3,
  hjust = 0.9, layout.exp = 1,
  low = "#B2182B", mid = "#F7F7F7", high = "#2166AC",
  name = "Corr") +
  ggtitle("Air Passengers Correlation Matrix") +
  theme(plot.title = element_text(hjust = 0.5, size = 14, face = "bold"))
```

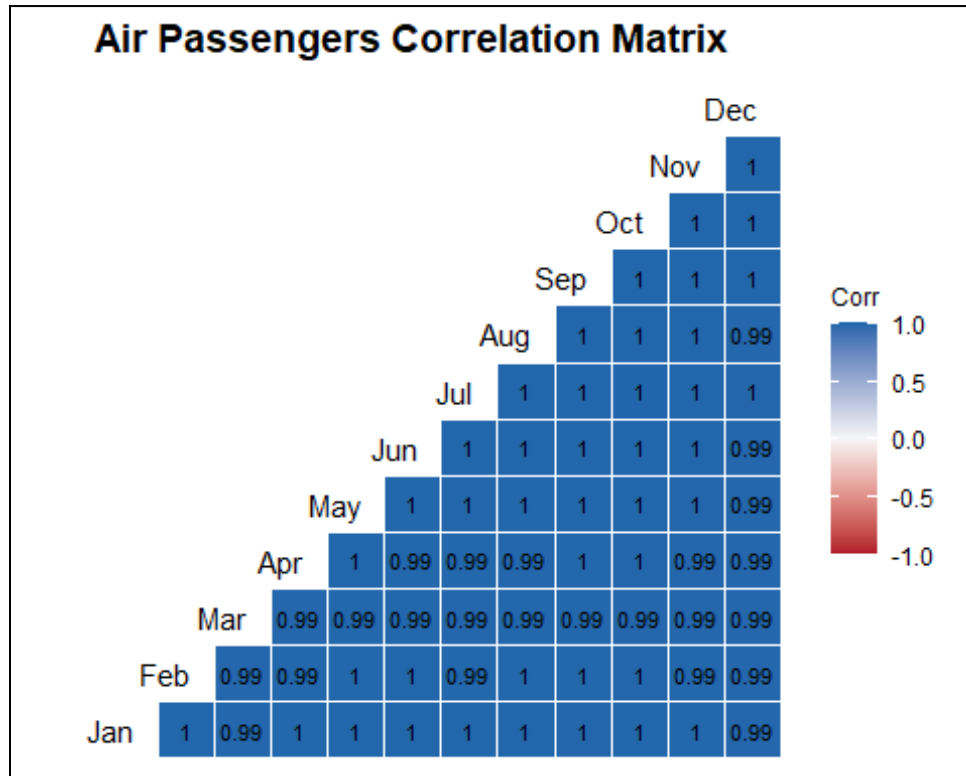


Figure 7. 5:Time Series Correlation

Through this approach, a time series correlation matrix transforms raw sequential data into a visually intuitive format, highlighting interdependencies between months and revealing underlying seasonal patterns. Analysts can use these insights to identify periods with similar or contrasting behavior, support trend analysis, and inform forecasting models. In this chart, the correlation matrix reveals how monthly passenger counts in the AirPassengers dataset are related across the year. The grid of colored cells shows that months closer together in the calendar, such as July and August or November and December, exhibit very strong positive correlations, indicated by deep blue shades and correlation values close to 1. This suggests that high passenger volumes in one month are often mirrored in the months immediately before or after it, likely due to sustained seasonal travel patterns. Conversely, months far apart, such as January and July, display weaker or even negative correlations, reflected by paler or reddish tones, showing that peaks in one part of the year may correspond to troughs in another. The visual pattern makes the seasonality in passenger trends immediately clear, illustrating how travel demand follows a cyclical rhythm over the course of a year.

Overall, a correlation heatmap condenses complex variable interactions into a single, interpretable graphic, highlighting patterns, strong associations, and potential predictors. It is an essential tool in exploratory data analysis, particularly when handling datasets with many variables, as it visually emphasizes relationships that might otherwise be obscured in tabular summaries.

7.5 Heatmap

A time series heatmap offers a powerful way to uncover how values evolve across time. Rather than parsing through lengthy tables or tracing multiple line plots, the story unfolds through color intensity, where darker or brighter shades immediately flag changes in magnitude. This visual design highlights cycles, seasonal rhythms, and sudden spikes with remarkable clarity, making complex temporal patterns instantly recognizable. A time series heatmap displays time points, such as months and years, along the axes, while each cell's color represents the value of the variable at that point. You can instantly see which months or years had higher activity and how they relate to each other across time. By combining temporal ordering with color-coded intensity, you get a visual summary that highlights trends and deviations you might otherwise miss.

In R, you can create this kind of heatmap by first structuring your data appropriately. For example, the `AirPassengers` dataset can be converted into a dataframe with explicit Year, Month, and Passengers columns. You then create a proper Date object and add an ordered factor for months (MonthName) using the `forcats` and `lubridate` packages, so months appear in calendar order. Using `ggplot2`, `geom_tile()` generates colored tiles for each year-month pair, with the fill aesthetic mapped to passenger counts. White borders and controlled line width help distinguish individual cells. The gradient from light green (`#f0f9e8`) to dark blue (`#08589e`) makes low and high values immediately visible. You can also tweak the plot's theme, removing grid lines, rotating x-axis labels, centering a bold title, and positioning the legend on the right, to make your heatmap both readable and visually appealing.

```
library(forcats)
# Convert AirPassengers ts to dataframe with proper Date
air_df <- data.frame(
  Year = floor(time(AirPassengers)),
  Month = cycle(AirPassengers),
  Passengers = as.numeric(AirPassengers))%>%
```

```

mutate(
  Date = as.Date(paste(Year, Month, "01", sep = "-")),
  MonthName = month(Date, label = TRUE, abbr = TRUE))
ggplot(air_df, aes(x = factor(Year), y = fct_rev(MonthName), fill =
Passengers)) +
  geom_tile(color = "white", linewidth = 0.5) +
  scale_fill_gradient(low = "#f0f9e8", high = "#08589e") +
  labs(
    title = "Air Passengers Heatmap (1949–1960)",
    x = "Year",
    y = "Month",
    fill = "Passengers") +
  theme_minimal(base_size = 12) +
  theme(
    panel.grid = element_blank(),
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.title = element_text(size = 14, face = "bold", hjust = 0.5),
    legend.position = "right"
  )

```

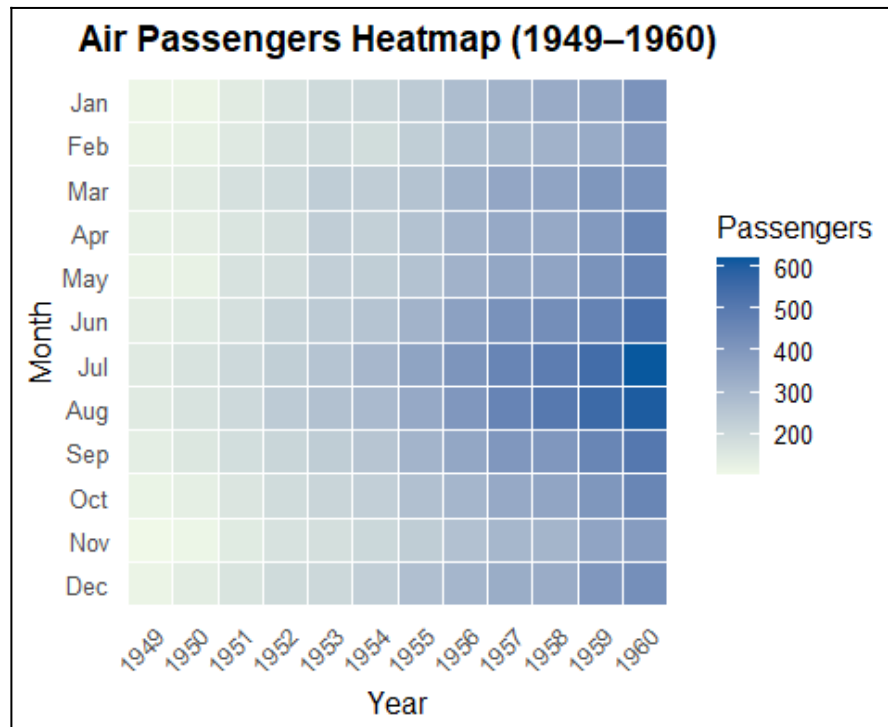


Figure 7. 6: Heat Map

With the help a heatmap, you can instantly see where patterns repeat, which months had unusually high or low values, and how trends evolve over multiple years. A time series heatmap turns raw temporal data into a visually intuitive story, helping you interpret complex datasets quickly and effectively. You can observe from the figure that the heatmap vividly captures both the seasonal and long-term trends in the AirPassengers dataset from 1949 to 1960. Darker blue shades, which represent higher passenger counts, cluster consistently around the mid-year

months such as July and August, indicating a steady increase in travel activity during these peak summer months each year. In contrast, lighter shades dominate early-year months like February and March, reflecting consistently lower passenger volumes that remain relatively constant over time, though with a gradual upward shift in later years. Towards the year's end, months like November and December show a moderate but steady increase, likely linked to holiday travel. Over the years, the darker tones across all months become progressively more prominent, signaling a consistent and sustained growth in air travel demand throughout the entire period, driven by both strong seasonal peaks and a robust long-term upward trend.

7.6 Candlestick Chart

When you're exploring time-based data, you don't just want to see numbers—you want to see how they change and connect over time. With a candlestick chart, you get the opening, closing, and range of each period in one view. This lets you spot ups and downs, notice trends, and compare what's happening from one period to the next without getting lost in raw values. A candlestick chart is a type of time-series plot where each “candle” represents a period of data, displaying the high, low, opening, and closing values. The vertical line shows the full range, while the body of the candle highlights the interval between the opening and closing values. By examining the sequence of candles, you can detect patterns, assess volatility, and understand how one period's performance relates to another.

In the context of air travel, for example, a candlestick chart can show weekly or monthly passenger counts. Each candle indicates the highest and lowest passenger numbers in that period, along with the opening and closing counts, allowing you to see trends, seasonal peaks, or drops in demand. This visualization helps analysts identify relationships between time intervals, such as how a surge in one month may correspond to a lull in the next, making it easier to plan capacity, staffing, or promotional strategies.

Here, the code uses the `AirPassengers` dataset to capture fluctuations in monthly passenger counts. The chart consists of two components: the candle body, created using `geom_rect()`, and the high-low wick, formed using `geom_segment()`. The body spans between the opening and closing values, with color-coding (green for upward, red for downward trends) denoting direction. The high and low whiskers extend vertically to denote the full range within the period.

The combination provides an efficient summary of time-based variation. This visual uses a grouped `tsibble` format to prepare the data, calculate candles, and define direction. It communicates dense information with minimal elements, which makes it a staple in stock market analysis and any domain tracking ranges over time.

```
ap_df <- AirPassengers %>%
  as_tsibble() %>%
  rename(Passengers = value)
# Building a candle data
Tcandle1 <- ap_df %>%
  mutate(
    Year = year(index),
    Month = month(index),
    date = as.Date(index)
  ) %>%
  group_by(Year, Month) %>%
  summarise(
    openner = first(Passengers),
    closer = last(Passengers),
    high = max(Passengers),
    low = min(Passengers),
    date = first(date),
    .groups = "drop"
  ) %>%
  mutate(Direction = ifelse(closer >= openner,
                             "Up", "Down"))
ggplot(Tcandle1, aes(x = date)) +
  # Candle body
  geom_rect(aes(
    xmin = date - days(50), # Spacing for the width of the box
    xmax = date + days(50),
    ymin = pmin(openner, closer),
    ymax = pmax(openner, closer),
    fill = Direction
  ), color = "black") +
  # High-Low wicks
  geom_segment(aes(
    x = date,
    xend = date,
    y = low,
    yend = high
  ), color = "black") +
  scale_fill_manual(values = c("Up" = "green",
                              "Down" = "red")) +
  labs(
    title = "Air Passengers Candlestick Chart",
    x = "Date",
    y = "Passengers",
    fill = "Direction"
  ) + Mytheme
```

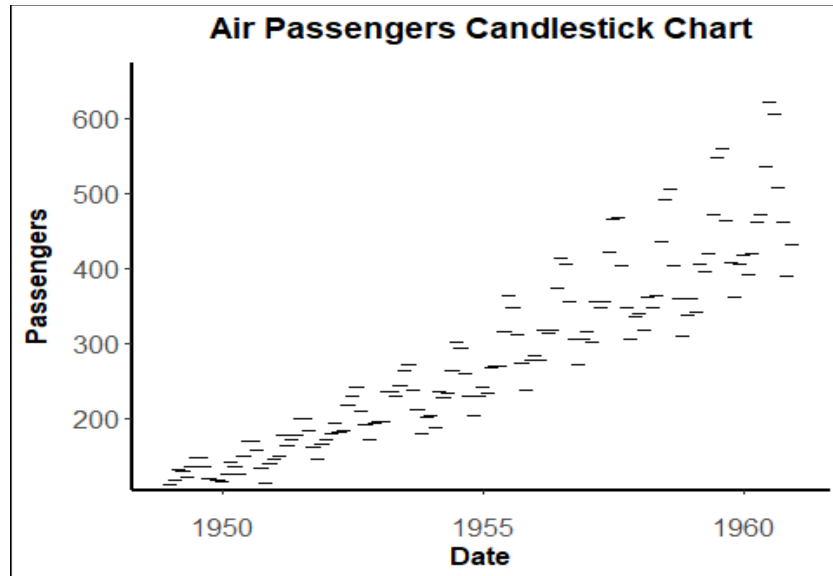


Figure 7. 7: Candlestick

The output chart provides a detailed monthly view of Air Passengers data from 1949 to 1960, with each candle visually summarizing passenger traffic for a month. The rectangular bodies represent the difference between the opening and closing passenger counts: for example, in January 1949, the month started with 112 passengers (openner) and closed slightly higher at 118 (closer), resulting in a green candle indicating an increase. Conversely, months with falling passenger numbers, such as March 1955 (openner 380, closer 370), display a red candle. The thin vertical lines, called wicks, extend from the low to the high passenger counts during the month, capturing the full range of variability; for instance, in December 1959, passenger counts ranged from a low of 422 to a high of 445. This layout makes it easy to identify periods of growth and decline as well as volatility within months, showing, for example, how passenger traffic rose steadily during the late 1950s and peaked around 1960.

CHAPTER 8: UNCERTAINTY

Uncertainty visualizations help communicate the level of confidence or variability in data, which is crucial in scientific reporting, forecasting, and decision-making under risk. These charts ensure that users don't over-interpret precise values when there's inherent variability. Most businesses and firms are more concerned about risks, profitability, projections and financial forecasts. Unfortunately, these aspects of information needed relating future projections are not easily implied using raw data. In this view, variability of data is crucial in projections and forecasting. In fact, statistical information is contained in the variations that occur in a data and it is important such variations are highlighted in order to gain insights into the future. To highlight error or variability in visual platforms, we make use of error bars, confidence interval bands, violin plots, fan charts and ribbon charts.

8.1 Error Bar Chart

Errorbar charts combine multiple visual elements to provide a comprehensive understanding of the relationship between variables while explicitly communicating their uncertainty. The central purpose is not just to highlight the mean or observed values but also to capture the variability or confidence around these points, helping the viewer judge the reliability and spread within each group. In this example, the upper and lower bounds for `Height` are calculated by adding and subtracting 2 units respectively, producing a vertical range that is represented as error bars. Within the `ggplot()` call, the aesthetics map `Volume` to the x-axis and `Height` to the y-axis, with color and shape assigned to different `Girth` categories. This dual mapping ensures that groups are easily distinguished both by color shading and by point symbols. The `geom_line()` function connects observations in the order of `Volume`, revealing trends across the dataset, while `geom_point(size=3)` overlays distinct markers to emphasize individual data points clearly. The `geom_errorbar()` then introduces vertical error bars, displaying the margin of variability around each height measurement, with `width=1` and slimmer line thickness to avoid visual clutter. To enhance interpretability, custom colors such as grey, orange, and brown are assigned to the `Girth` groups, while `scale_y_continuous()` extends the axis slightly upward to prevent overlap and improve clarity. Titles and axis labels add essential context, and `guides()` fine-tunes the legend's box size for balance and neatness. Taken together, these elements transform a simple line-and-point display into a more informative visualization that not only shows the trend of `Height`

against **Volume** across **Girth** groups but also communicates the uncertainty inherent in the measurements, an essential feature for rigorous analysis.

```
mytrees <- trees
mytrees <- mytrees %>%
  mutate(Girth = cut(Girth,
                     breaks = 3,
                     labels = c("Small", "Medium", "Large")))
mytreez <- mytrees
mytreez <- mytreez %>%
  mutate(
    upper = Height + 2, # Adjust height range individually
    lower = Height - 2)
mytreez %>%
  ggplot(aes(x=Volume, y=Height, colour = Girth, shape = Girth)) + # adjust shape
  geom_line() +
  geom_point(size=3) +
  geom_errorbar(aes(ymin = lower, ymax = upper), width=1, size=0.5) + # adjust width
  scale_color_manual(values = c("grey", "orange", "brown")) +
  scale_y_continuous(expand = expansion(mult = c(0, 0.1))) +
  xlab("Volume") +
  ylab("Height") +
  labs(title="Errorbar Chart", fill=NULL) + Mytheme
guides(fill=guide_legend(keywidth=0.1,
                        keyheight=0.1,
                        default.unit="cm")) # changes size of legend box
```

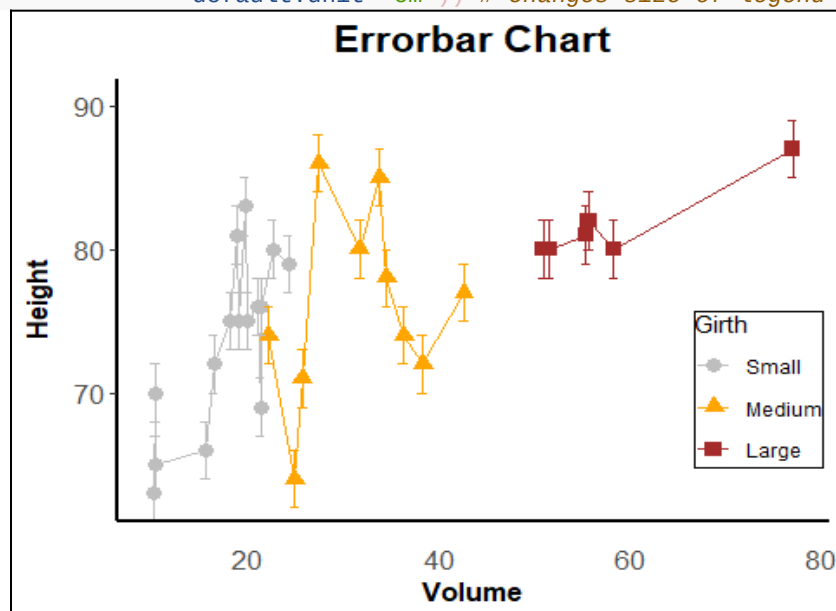


Figure 8. 1: Error Bar Chart

In this figure, the error bar chart depicts tree heights for different girth categories plotted against volume, with uncertainty represented by vertical error bars. Each point shows the observed height, while the accompanying error bars extend two units above and two units below, giving a total uncertainty range of four units for each measurement. This range captures the possible variation in height estimates, illustrating that while the recorded values provide a central

measure, there is a defined margin within which the true height may lie. By displaying these bounds, the chart communicates not only the central trends but also the degree of measurement variability present in the data.

8.2 Errorbarh Chart

Errorbarh is designed to illustrate group comparisons while making the level of uncertainty around each estimate visible. At its core, this chart shows the mean of a variable for each category, but rather than presenting values as fixed and absolute, it adds horizontal extensions, error bars, that indicate the possible range in which the true mean might fall. These ranges are often expressed as an estimated margin of error ($\pm n$ units) or confidence interval, giving viewers an immediate sense of how precise or uncertain each measurement is.

The bars themselves are drawn using `geom_bar(stat = "identity")`, which ensures that the bar length corresponds directly to the calculated mean rather than being determined by counts. Outlines are added to enhance clarity, helping each category stand out. To capture uncertainty, `geom_errorbar()` is layered on top of the bars. These horizontal lines extend to the left and right of each mean value, marking the lower and upper bounds of the estimate. The result is a more nuanced visual: not only can one compare mean values across categories, but one can also judge whether differences are meaningful or might plausibly overlap once variability is considered.

Such a representation is especially valuable in fields like experimental research, survey analysis, and performance benchmarking, where reporting averages alone could be misleading. By explicitly showing variability, the horizontal error bar chart communicates both the central trend and the level of confidence we should place in it, making comparisons more transparent and reliable.

```
mytrees <- trees
# Categorize Girth
mytrees <- mytrees %>%
  mutate(Girth = cut(Girth,
                     breaks = 3,
                     labels = c("Small", "Medium", "Large")))
mytreeez <- mytrees %>%
  group_by(Girth) %>%
  summarise(Volume = mean(Volume), .groups = "drop") %>%
  mutate(Error_Margin = 1.5)
mytreeez1 <- mytreeez
ggplot(mytreeez1, aes(x = Girth, y = Volume, fill = Girth)) +
  geom_bar(stat = "identity",
```

```

position = "identity",
width = 0.5,
color = "black") +
geom_errorbar(aes(
  ymin = Volume - 1.4, # adjust height of the error bars
  ymax = Volume + 1.4),
position = "identity",
width = 0.2,
color = "black") +
scale_fill_manual(values = c("grey", "pink", "#FED966")) +
scale_y_continuous(expand = expansion(0)) +
xlab("Girth") +
ylab("Volume") +
labs(title = "Simple Bar Graph", fill = NULL) +
labs(title="Vertical Bar Chart", fill=NULL)+ Mytheme
guides(fill=guide_legend(keywidth=.2,
  keyheight=.1,
  default.unit="cm"))

```

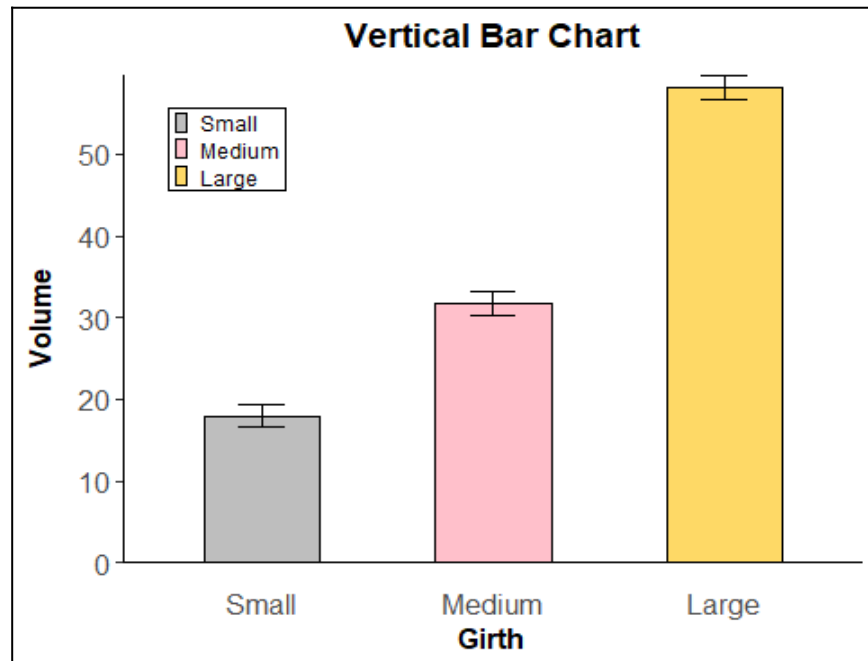


Figure 8. 2: Error Barh Chart

Looking at this chart, you can see that the vertical bar chart presents the average volume for trees within each girth category, **Small**, **Medium**, and **Large**, accompanied by error bars to represent uncertainty in these averages. The height of each bar shows the estimated mean, while the error bars extend above and below to indicate the potential range of values within the margin of error. For small-girth trees, the bars extend roughly from about $mean - 1.4$ to $mean + 1.4$, showing a total error range of about 2.8 units. Medium-girth trees follow a similar error margin, with the range again spanning approximately ± 1.4 units from the mean. Large-girth trees also maintain this ± 1.4 range, indicating that while their average volume is higher, the uncertainty level is

comparable to the other categories. Overall, the inclusion of these ranges makes it easier to assess the degree of variability and reliability in the presented averages.

8.3 Horizontal Errorbar

Your work may be revolving around discovering extreme occurrences in your data. A horizontal error bar chart helps you to meet this task by displaying the relationship between two continuous variables while simultaneously revealing the uncertainty or variability around their estimates. Each point shows the mean value, and the bars stretching left and right indicate the potential range of values. This added layer of information helps viewers judge not only the central tendency but also the reliability of the measurement. When categories are compared, the chart makes it easier to see whether intervals overlap, offering a more nuanced understanding of differences or similarities between groups.

In practice, the central points are plotted with `geom_point()`, adjusted in size to enhance clarity. The horizontal error bars are introduced with `geom_errorbarh()`, where `xmin` and `xmax` define the lower and upper confidence bounds. The optional `height` argument (set here to 5) ensures the bars remain clearly visible without overwhelming the plot. Distinct colors are manually assigned to the girth categories, creating immediate visual separation. To tighten the fit of the chart, the y-axis is scaled with `scale_y_continuous()`, preventing unnecessary blank space.

```
# Summarize Volume for each Girth category (mean, sum, or another metric)
mytrees <- mytrees %>%
  group_by(Girth) %>%
  summarize(
    Volume = mean(Volume), # Adjusts aggregation as needed (mean/sum/etc.)
    Height = mean(Height),
    upper = Volume + 2.5, # Adjusts error margins
    lower = Volume - 2.5) %>%
  ungroup()
ggplot(mytrees, aes(x = Volume, y = Height, colour = Girth))+
  geom_point(size=3)+
  geom_errorbarh(aes(xmin = lower,
                    xmax = upper), height=5, size=1)+
  scale_color_manual(values = c("grey", "black", "#FED966"))+
  scale_y_continuous(expand = expansion(0))+
  xlab("Volume")+
  ylab("Height")+
  labs(title="Horizontal Errorbar Chart ", fill=NULL)+Mytheme
guides(fill=guide_legend(keywidth=0.01,
                        keyheight=0.01,
                        default.unit="cm"))
```

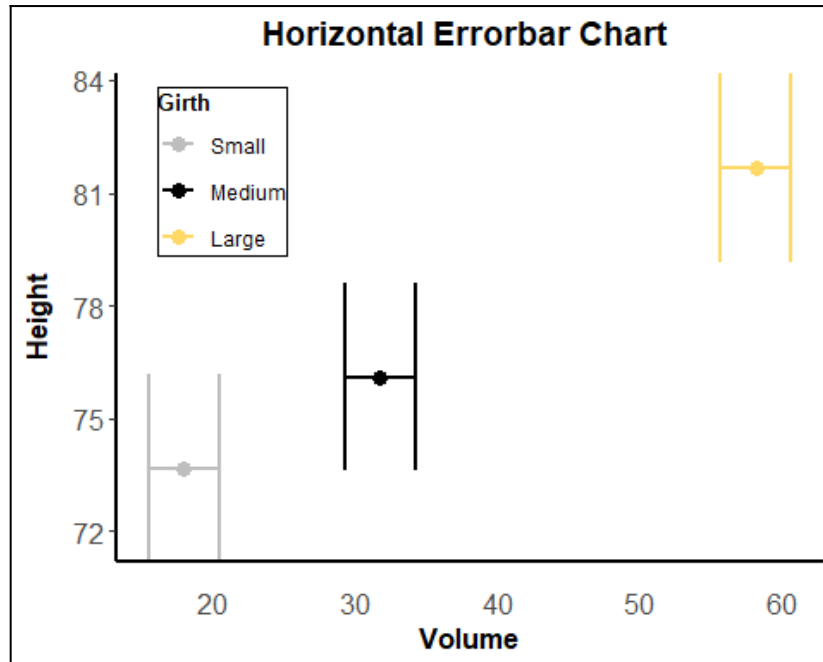


Figure 8. 3: Horizontal Error Bar Chart

This figure is particularly noteworthy for its clear depiction of uncertainty around average tree volumes for each girth category. The points represent mean heights plotted against mean volumes, while the horizontal error bars extend 2.5 units on either side of each volume mean, indicating a total uncertainty range of 5 units. This visualizes the variability and confidence bounds of the volume measurements within each category. By emphasizing these error margins, the chart communicates that the true volume values may fluctuate within these ranges, providing a nuanced understanding of the data's precision and the reliability of the average volume estimates across the girth groups.

The vertical error bar, made with `geom_errorbar()`, shows uncertainty along the y-axis, making it most useful when the vertical dimension represents the key measurement with variation, for example, mean scores across groups with confidence intervals above and below. The bars stretch upward and downward from each point, emphasizing the possible range of outcomes in the vertical direction. By contrast, the horizontal error bar, drawn with `geom_errorbarh()`, communicates variation along the x-axis. Instead of extending vertically, the bars spread to the left and right, which is especially meaningful when the horizontal axis represents the estimate of interest, such as measured volumes or effect sizes. In this case, the y-axis is often categorical or a secondary grouping variable, so the uncertainty is shown horizontally across categories.

Another distinction lies in perception: vertical error bars are more familiar and common in classical data summaries, whereas horizontal error bars are particularly useful in forest plots, meta-analysis charts, or any layout where categories run down the vertical axis and estimates spread sideways. Both types emphasize reliability and variability, but they orient uncertainty differently, depending on which axis holds the main measured variable.

8.4 Cross bar Chart

A crossbar chart is useful for visualizing ranges of values within categories, especially when you want to emphasize variability or uncertainty around a central measurement. Each crossbar represents a vertical span extending from a lower bound to an upper bound, intersected by a horizontal line that typically indicates the central or observed value (such as the mean or a measured statistic). This makes it easy to compare how ranges overlap, spread out, or cluster across different categories.

The visualization effectively highlights both position (where the central values lie) and dispersion (how wide the ranges extend). This dual emphasis allows viewers to quickly assess stability, overlap, or separation between groups, providing insight into reliability, confidence intervals, or measurement variation.

In plotting, the key function is `geom_crossbar()`, which constructs horizontal bars centered on the measured values (e.g., Height) with vertical extents defined by lower and upper limits. The width argument controls the thickness of the crossbars, in this case, set to `width=2.5` to make them more visible and distinct. Color is used strategically through `scale_color_manual()` to represent `Girth` categories, making it easier to visually separate groups. For clarity, the y-axis is refined with `scale_y_continuous()`, adding extra space above the tallest crossbars so that the visualization doesn't appear cramped.

```
mytreez<-mytrees
mytreez <- mytreez %>%
  mutate(
    upper = Height + 2, # Adjust height range individually
    lower = Height - 2)
mytreez%>%
  ggplot(aes(x=Volume,y=Height, colour = Girth))+
  geom_crossbar(aes(ymin = lower,ymax = upper),width=2.5)+ # adjust width
```

```

scale_color_manual(values = c("grey", "orange", "pink"))+
scale_y_continuous(expand = expansion(mult = c(0, 0.1)))+
xlab("Volume")+
ylab("Height")+Mytheme
labs(title="Cross-Bar Chart", fill=NULL)+
guides(fill=guide_legend(keywidth=0.1,
                          keyheight=0.1,
                          default.unit="cm"))

```

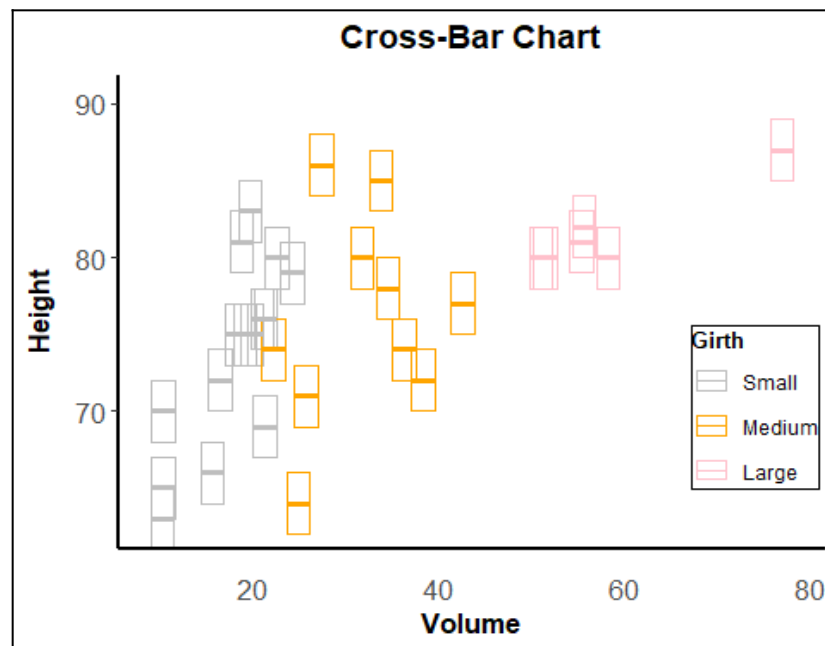


Figure 8. 4: Cross Bar Chart

In this chart, the vertical crossbars span from two units below to two units above the mean height, visually representing the range within which actual tree heights might vary. The width of these bars emphasizes the degree of variation and potential measurement error in the height data. This presentation allows a clear understanding of the precision around height estimates, making it easier to appreciate the inherent variability in the data without focusing solely on point estimates. The distinct colors by girth further aid in comparing uncertainty levels across categories, providing a comprehensive picture of height variation relative to volume.

8.5 Confidence Interval Band

This geom overlays a fitted line on a scatter plot or time series and surrounds it with a shaded ribbon that represents statistical uncertainty. The line itself captures the main tendency in the data, while the shaded band communicates the range within which the true relationship likely falls. By default, `geom_smooth()` adapts its method to `method = "lm"` for smaller datasets, and

`method = "GAMS"` for larger ones, though you can also impose a specific model such as linear regression. This dual display, the central line and the confidence band, lets viewers grasp both the general signal and its reliability, making it especially useful when you want to emphasize collective trends without losing sight of variability.

```
mytreez<-mytrees
pl<-ggplot(mytreez,aes(x=Volume,y=Height, colour = Girth))+
  geom_point()+
  geom_smooth(method = "lm")+
  scale_color_manual(values = c("brown","orange","grey"))+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  ylab("Height")+
  xlab("Volume")+Mytheme
labs(title="Confidence Plot ",fill=NULL)+
guides(fill=guide_legend(keywidth=0.1,
                          keyheight=0.1,
                          default.unit="cm"))

pl
```

Alternatively, you can set the method to `method = "loess"` to change to error bands with smoothed line.

```
mytreez<-mytrees
ps<-ggplot(mytreez,aes(x=Volume,y=Height, colour = Girth))+
  geom_point()+
  geom_smooth(method = "loess")+
  scale_color_manual(values = c("brown","orange","grey"))+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  ylab("Height")+
  xlab("Volume")+Mytheme
labs(title="Confidence Plot ",fill=NULL)+
guides(fill=guide_legend(keywidth=0.1,
                          keyheight=0.1,
                          default.unit="cm"))

ps
pl + ps + plot_annotation(title = "Combined Confidence Interval Plots")+Mytheme
```

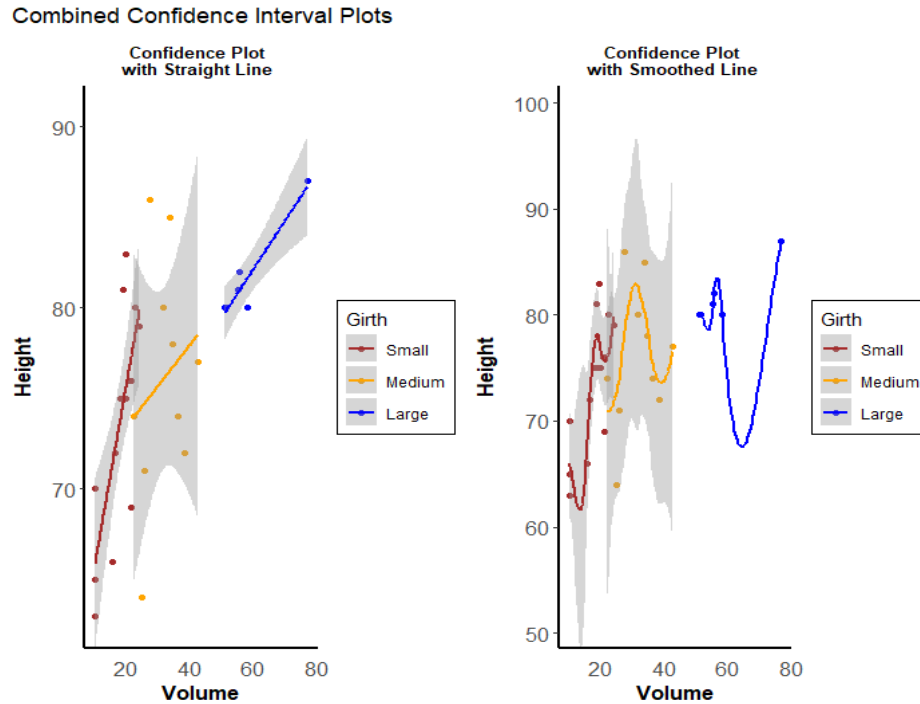


Figure 8. 5: Confidence Intervals Chart

The smooth regression lines generated by the linear model show the overall trend, while the shaded confidence bands reflect the uncertainty of the estimates. Wider bands indicate greater uncertainty in predicting height at certain volume values, highlighting where the model's predictions are less precise. The distinct colors for each girth group help differentiate how confidence varies across categories, revealing that some groups exhibit more variability in height measurements relative to volume. Overall, this confidence plot visually communicates not just the trends but also the reliability of those trends within the data.

8.6 Violin Plot

A violin chart is a powerful visualization tool used to explore the distribution of a numeric variable across different categories while also highlighting uncertainty in the data. Unlike box plots, which summarize distributions through medians and quartiles, violin plots display the entire probability density of the data, showing not only central tendencies but also variation, spread, and multimodal structures. This richer view communicates where the data are concentrated and where uncertainty or variability is higher.

In this example, the chart visualizes how tree **Height** varies across different **Girth** categories. The violin shape reveals areas of high and low density, providing insight into whether

observations cluster tightly (low uncertainty) or spread widely (greater uncertainty). The function `geom_jitter()` overlays individual observations with slight horizontal noise, reducing overplotting and making each point visible, this grounds the distribution in the actual data and shows the scatter that contributes to uncertainty. Meanwhile, `geom_violin()` draws the violin shapes, representing smoothed density estimates of `Height` within each `Girth` category. The argument `alpha = 0.5` makes the violins semi-transparent so the jittered points remain visible beneath, reinforcing the connection between raw observations and distribution shape. Suppressing separate outlier points with `outlier.shape = NA` shifts the focus to the overall distribution, letting the violin's tails reflect the extremes and their associated uncertainty.

```
mytreez<-mytrees
mytreez%>%
  ggplot(aes(x=Girth,y=Height, fill = Girth,colour = Girth))+
  geom_jitter()+
  geom_violin(alpha = 0.5,
              outlier.shape = NA )+
  scale_fill_manual(values = c("black","brown","pink"))+
  scale_color_manual(values = c("orange","black","blue"))+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  xlab("Girth")+
  ylab("Height")+Mytheme
  labs(title="Violin Chart",fill=NULL)+
  guides(fill=guide_legend(keywidth=0.1,
                           keyheight=0.1,
                           default.unit="cm"))
```

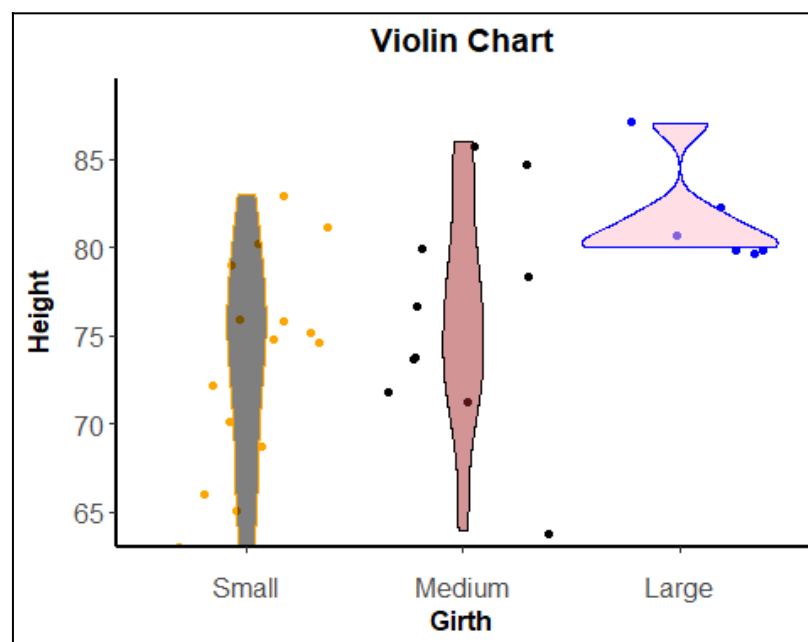


Figure 8. 6:Violin Chart

What this figure clearly shows is the distribution and uncertainty of tree heights within each girth category. The violin shapes provide a detailed view of the density of height measurements, with wider sections indicating where heights are more concentrated. The jittered points add a layer of individual variability, showing the spread and outliers within each group. The transparency of the violins allows us to see the overall shape while emphasizing uncertainty through the spread of data points. Color differentiation by girth category helps to visually separate these distributions, revealing how uncertainty in height varies with girth size, from tighter, more consistent height distributions in some groups to broader spreads in others. This violin chart thus offers a nuanced perspective on variability and uncertainty in height measurements across girth categories.

8.7 Ribbon Chart

A ribbon chart emphasizes, just like confidence chart, a central trend line while shading the uncertainty or variability around it. This shaded band, often representing error margins, prediction intervals, or confidence intervals, shows the range where the true values are likely to fall. In essence, the ribbon provides a visual sense of stability, reliability, and spread across levels of the explanatory variable.

The `geom_smooth()` function overlays a smooth line that summarizes the overall relationship between `Volume` and `Height`. To enrich this visualization, `geom_ribbon()` adds a semi-transparent grey band around the line, spanning from the lower bound to the upper bound of `Height` at each point on the x-axis. This shaded ribbon not only highlights the general trend but also communicates the degree of uncertainty, allowing viewers to distinguish between strong, consistent relationships and those that are more variable.

```
mytrees <- trees %>%
  mutate(Girth = case_when(
    Girth <= 10 ~ "Small",
    Girth > 10 & Girth <= 13 ~ "Quite Small",
    Girth > 13 & Girth <= 16 ~ "Medium",
    Girth > 16 ~ "Wide"
  ))
# Summarize Volume for each Girth category (mean, sum, or another metric)
mytrees <- mytrees %>%
  group_by(Girth) %>%
  summarize(
    Volume = mean(Volume), # Adjust aggregation as needed (mean/sum/etc.)
    Height = mean(Height),
    upper = Height + 2.5, # Adjust error margins
    lower = Height - 2.5
```

```

) %>%
  ungroup()
# Convert Girth to ordered factor
mytrees$Girth <- factor(mytrees$Girth,
                        levels = c("Small", "Quite Small", "Medium", "Wide"),
                        ordered = TRUE)
ggplot(mytrees, aes(x = Volume, y = Height)) +
  geom_smooth(color = "orange") +
  geom_ribbon(aes(ymin = lower,
                ymax = upper),
            fill = "grey", alpha = 0.2) +
  scale_y_continuous(expand = expansion(0)) +
  xlab("Volume") +
  ylab("Height") + theme
  labs(title = "Ribbon Chart", fill = NULL) +
  guides(fill = guide_legend(keywidth = 0.01,
                             keyheight = 0.01,
                             default.unit = "cm"))

```

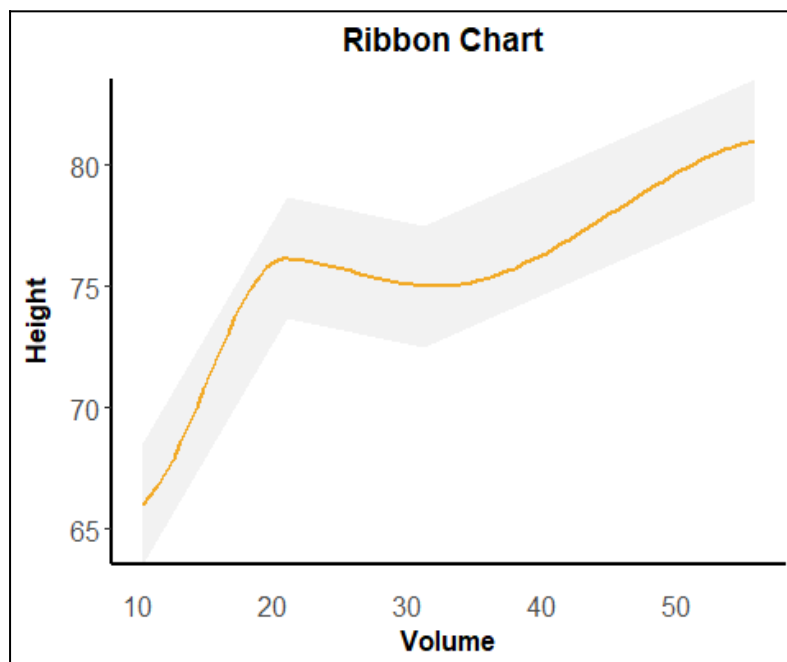


Figure 8. 7: Ribbon Chart

A key takeaway from this figure is how it highlights uncertainty around the relationship between **Volume** and **Height** across different girth categories. The orange smooth line represents the overall trend, showing how **Height** changes with **Volume** on average. Surrounding this line is a grey ribbon that illustrates the margin of error or variability in **Height**, extending roughly 2.5 units above and below the mean values. This shaded area visually conveys the range within which the true **Height** values are likely to fall, making it clear that while the trend is generally upward, there is some uncertainty around the exact **Height** at each **Volume** level. The ribbon's width indicates greater or lesser uncertainty at different points along the curve, allowing us to see

where predictions are more or less reliable. Overall, this chart effectively communicates the confidence we can have in the observed **Volume -Height** relationship, emphasizing variability without obscuring the underlying trend.

8.8 Fan Chart

A fan chart is a visualization that combines a central estimate or trend line with a surrounding shaded region that conveys uncertainty, variability, or prediction intervals. Instead of showing a single trajectory, it emphasizes both the expected trend and the plausible range of outcomes around it. This makes it especially useful for illustrating confidence intervals, forecast bands, or the spread of values across categories. The shaded “fan” shape captures the minimum and maximum observed values (or specified bounds) within each group, giving readers a clear sense of dispersion and reliability.

In `ggplot2`, the function `geom_ribbon()` generates the fan-shaped area by filling the space between `ymin` and `ymax`. By applying a sky-blue fill with partial transparency (`alpha = 0.4`), the shaded region highlights variability while still allowing the central line to remain visible. This dual representation, line plus band, enables viewers to simultaneously see the trend and its uncertainty, greatly improving interpretation compared to a line alone.

```
# Prepare the data
mytrees <- trees
mytrees <- mytrees %>%
  mutate(Girth = cut(Girth, breaks = 3, labels = c("Small", "Medium",
"Large"))) %>%
  group_by(Girth) %>%
  summarise(
    mean_height = mean(Height),
    min_height = min(Height),
    max_height = max(Height)
  )
ggplot(mytrees, aes(x = Girth, y = mean_height, group = 1)) +
  geom_line(color = "blue", size = 1) + # Thicker central line
  geom_ribbon(aes(ymin = min_height, ymax = max_height), fill = "skyblue",
alpha = 0.4) + # More visible ribbon
  xlab("Girth") +
  ylab(" Mean Height") +
  labs(title="Fan Chart", fill="Girth") + Mytheme
  guides(fill=guide_legend(keywidth=0.2,
                           keyheight=0.2,
                           default.unit="cm"))
```

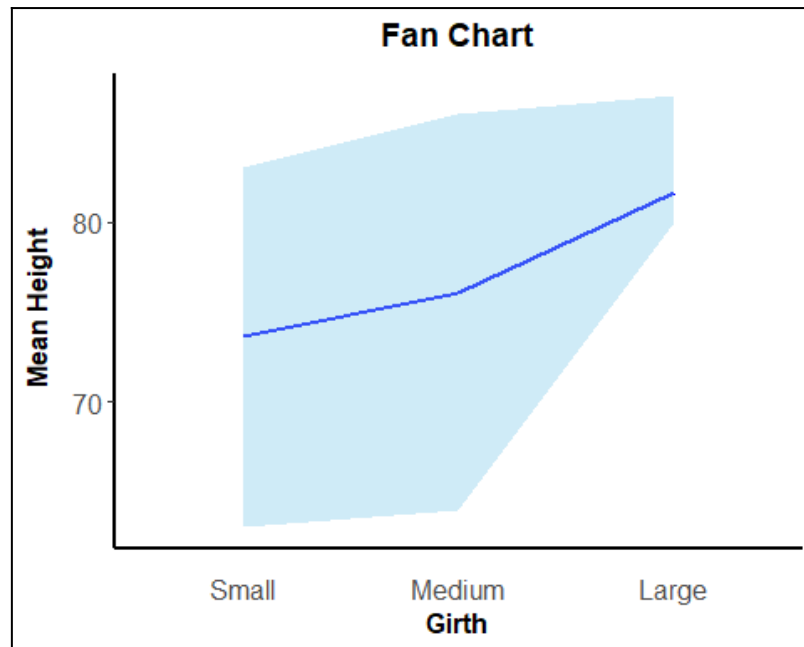


Figure 8. 8: *Fan Chart*

This figure is particularly noteworthy for its clear illustration of uncertainty in the **Height** measurements across different **Girth** categories. The blue line tracks the mean **Height** for each girth group, providing a central tendency reference. Surrounding this line is a sky blue ribbon representing the full range of observed **Height**, from minimum to maximum, for each category. This shaded band vividly highlights the variability and spread in the data, showing that while the mean gives a general idea, individual tree **Height** within each girth group can vary substantially. Notably, the **Medium** girth group displays the widest ribbon, indicating the highest uncertainty or variability in **Height** measurements among the groups. This greater spread signals higher risk in predicting **Height** for trees within this category. The **Small** and **Large** girth groups exhibit narrower ribbons, suggesting comparatively lower uncertainty. By combining the smooth central trend with this visible range, the fan chart effectively communicates both the average pattern and the inherent uncertainty, allowing for a nuanced understanding of the data's distribution.

8.9 Linerange Chart

A line range chart provides a clear way to show how values vary within defined limits, making it especially effective when the focus is on intervals rather than single measurements. Instead of plotting one exact point, it highlights the span between a minimum and maximum (or lower and upper bound) for each observation, which immediately conveys the degree of fluctuation or uncertainty present in the data. The core visual element comes from `geom_linerange()`, which

draws vertical lines extending from the lower to the upper **Height** values at each **Volume** position. To ensure clarity, the line thickness is increased with `size=1.5`. Color differentiation is handled through `scale_color_manual()`, allowing each Girth group to stand out distinctly. For finishing touches `scale_y_continuous()` expands the vertical axis slightly beyond the tallest lines, giving the plot breathing room and a more polished appearance.

```
mytreez<-mytrees
mytreez <- mytreez %>%
  mutate(
    upper = Height + 2, # Adjust height range individually
    lower = Height - 2
  )
mytreez%>%
  ggplot(aes(x=Volume,y=Height, colour = Girth))+
  geom_linerange(aes(ymin = lower,ymax = upper),size=1.5)+
  scale_color_manual(values = c("grey","orange","lightgreen"))+
  scale_y_continuous(expand = expansion(mult = c(0,0.1)))+
  xlab("Volume")+
  ylab("Height")+Mytheme
  labs(title="Line Range Chart",fill=NULL)+
  guides(fill=guide_legend(keywidth=0.1,
                           keyheight=0.01,
                           default.unit="cm"))
```

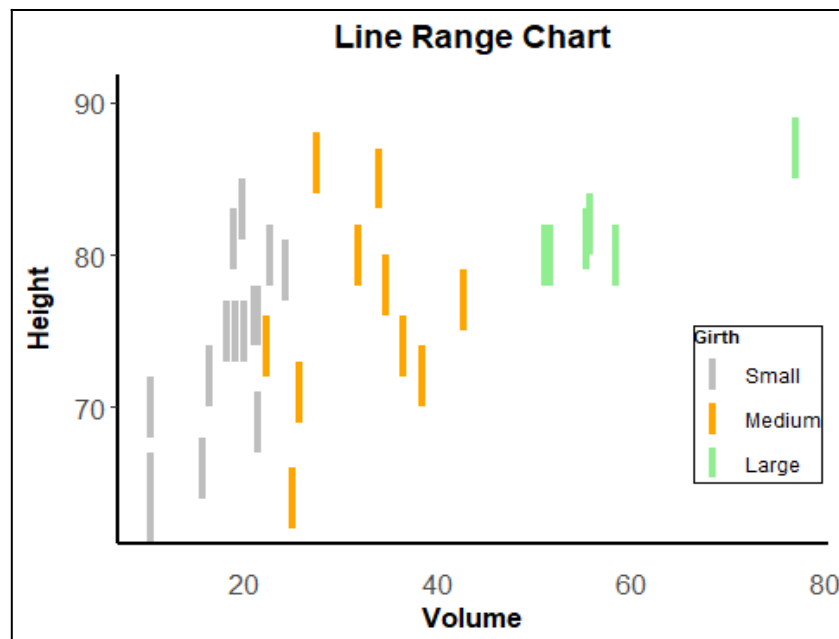


Figure 8. 9: Linerange Chart

This figure clearly illustrates the uncertainty around **Height** measurements for each **Volume** value across different **Girth** groups. The vertical lines represent the range of possible **Height**, extending two units above and below the central **Height** point, effectively visualizing the

potential variation or error margin for each observation. The spread of these lines varies across the **Girth** categories, with the **Medium** group (orange) showing the longest vertical line ranges, indicating the highest uncertainty in **Height** estimation. In contrast, the **Small** (grey) and **Large** (light green) groups display shorter ranges, suggesting more precise **Height** measurements. This visualization effectively communicates both the central **Height** values and their plausible variation, enabling viewers to grasp the confidence or risk associated with each data point in the context of tree **Volume** and **Girth**.

8.10 Pointrange Chart

Another chart tool that is important in displaying variability in data is a point range chart. It is a visualization that balances precision and uncertainty by displaying both the central value and its variability in a single graphic. Each point represents the observed or estimated value, while vertical lines extending above and below indicate the lower and upper limits, commonly error margins, prediction bounds, or confidence intervals. This format is especially useful in statistical summaries, experimental data, or grouped comparisons, where it is crucial to convey not only the estimate but also the degree of uncertainty or variability surrounding it. Viewers can instantly assess whether differences between groups are meaningful by considering both the positions of the points and the extent of their ranges.

In `ggplot2`, the `geom_pointrange()` function is used to create this chart. It plots each data point and attaches a vertical line that stretches from the specified `ymin` to `ymax`, visually encoding the range around the central measurement. Adjusting aesthetics like `size` controls the thickness of the range lines, while `color`, `shape`, and `alpha` help emphasize or differentiate groups. This makes `geom_pointrange()` an effective tool for highlighting measurement reliability alongside observed values.

```
mytreez<-mytrees
mytreez <- mytreez %>%
  mutate(
    upper = Height + 2, # Adjust height range individually
    lower = Height - 2
  )

mytreez%>%
  ggplot(aes(x=Volume,y=Height, colour = Girth))+
  geom_pointrange(aes(ymin = lower,ymax = upper),size=1)+
  scale_color_manual(values = c("grey","orange","pink"))+
```

```
scale_y_continuous(expand = expansion(mult = c(0, 0.1)))+
xlab("Volume")+
ylab("Height")+Mytheme
labs(title="Point Range Chart", fill=NULL)+
guides(fill=guide_legend(keywidth=0.1,
                          keyheight=0.1,
                          default.unit="cm"))
```

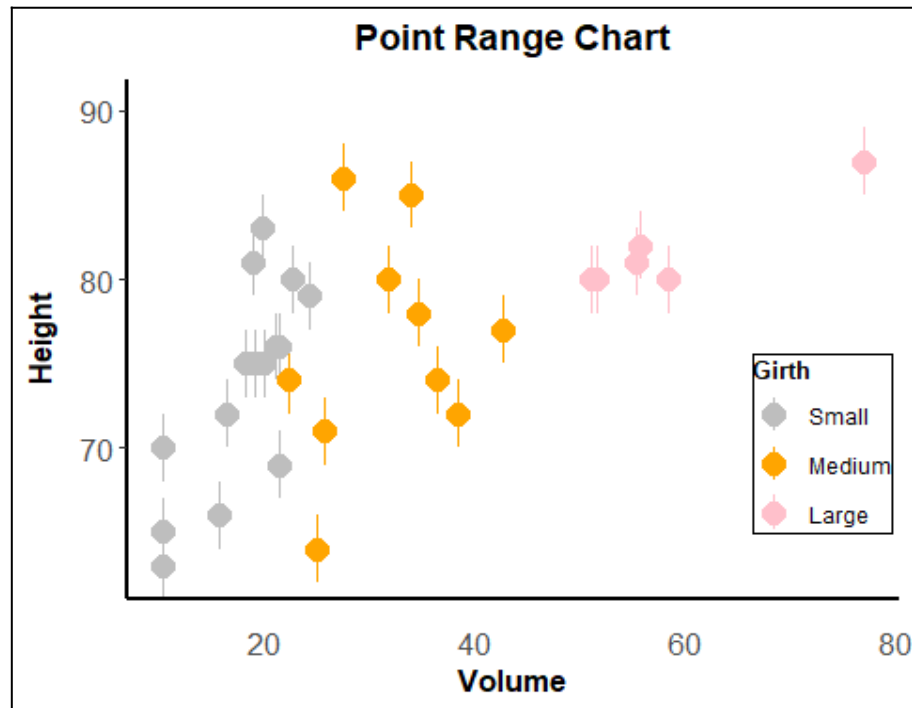


Figure 8. 10: Point Range Chart

This figure is particularly noteworthy for its clear depiction of uncertainty in **Height** measurements relative to **Volume** across different **Girth** groups. Each point represents the central **Height** value, while the vertical lines extending above and below illustrate the potential range of variation, set at two units. Among the groups, the **Medium** category (orange) shows the widest point ranges, indicating greater uncertainty or variability in **Height** estimates. The **Small** group (grey) and **Large** group (pink) have comparatively narrower ranges, suggesting more consistent **Height** values. This point range chart effectively communicates not only the average **Height** but also the confidence intervals, helping viewers to understand where measurements might be less precise and where greater caution is warranted in interpretation.

CHAPTER 9: RANKING

Ranking visualizations show the order or position of items based on a particular measure, helping to identify top performers or highlight disparities. They are widely used in sports, education, business, and performance metrics. Common charts include bar charts, lollipop charts, dot plots, bump charts, and ordered tables. Bar charts, especially when sorted, are effective in showing rankings clearly. Dot plots offer a minimalistic view for large lists. Bump charts show changes in ranking over time, allowing users to trace the movement of entities across time intervals. Ordered tables present ranked data with numeric labels and can include conditional formatting for added visual impact. These charts help audiences quickly grasp who is leading, lagging, or improving.

9.1 Sorted Bar Chart

A deliberate arrangement that imposes order on the data, allowing patterns to reveal themselves with greater clarity beyond simple collection of vertical columns is what known as a sorted bar chart. By ranking categories, here, tree girths, according to their mean values, the eye is naturally drawn to contrasts in size, making it easier to discern relative differences and hierarchies that might otherwise remain obscured in a random arrangement. This sorting transforms the chart from a static display into a structured comparison tool.

In constructing such a chart with `ggplot2`, the `geom_bar()` layer materializes these mappings into vertical bars, each representing a mean Volume, their black outlines serving as crisp boundaries that prevent adjacent categories from visually bleeding into one another. Crucial to interpreting the chart are the error bars, added through `geom_errorbar()`. These extend symmetrically, approximately **1.4** units above and below each bar in this case, signaling the uncertainty around the means. Rather than suggesting absolute values, they remind the reader that each bar is an estimate, and the true average might plausibly lie anywhere within that vertical span. Such annotations enrich the comparison, ensuring the narrative accounts for variability rather than conveying misleading precision. To refine aesthetics, `scale_fill_manual()` assigns a consistent and intentional palette, reinforcing distinctions while maintaining visual coherence.

```

mytrees <- trees
# Categorize Girth
mytrees <- mytrees %>%
  mutate(Girth = cut(Girth,
                      breaks = 3,
                      labels = c("Small", "Medium", "Large")))
mytreez<-mytrees
mytreez <- mytreez %>%
  group_by(Girth) %>%
  summarise(Volume = mean(Volume), .groups = "drop") %>%
  mutate(Error_Margin = 1.5)
mytreez1<-mytreez
ggplot(mytreez1, aes(x = Girth, y = Volume, fill = Girth)) +
  geom_bar(stat = "identity",
           position = "identity",
           width = 0.5,
           color = "black") +
  geom_errorbar(aes(
    ymin = Volume - 1.4, # adjust height of the error bars
    ymax = Volume + 1.4
  ),
  position = "identity",
  width = 0.2,
  color = "black") +
  scale_fill_manual(values = c("grey", "pink", "#FED966")) +
  scale_y_continuous(expand = expansion(0)) +
  xlab("Girth") +
  ylab("Volume") +
  labs(title = "Simple Bar Graph", fill = NULL) + Mytheme
  labs(title="Sorted Bar Chart", fill=NULL)+
  guides(fill=guide_legend(keywidth=.2,
                           keyheight=.1,
                           default.unit="cm"))

```



Figure 9. 1: Sorted Bar Chart

Looking at Figure 9.1 above, you can clearly notice the ranking of average Volume across different Girth categories, with each bar representing the mean Volume for the Small, Medium, and Large groups. Starting from the lowest, the Small group shown in grey has the smallest average Volume, approximately around 18, followed by the Medium group in pink with a higher average near 29, and the Large group in bright yellow (`#FED966`) ranked highest with the greatest average Volume close to 43. These approximate highest values for each group give a clearer sense of scale alongside the ranking. This sorted bar chart effectively ranks the Girth categories by their mean Volume, allowing an easy comparison of their relative sizes.

9.2 Lollipop Chart

A lollipop chart offers a cleaner alternative to the traditional bar chart when categorical comparisons are involved. Instead of filling space with thick bars, it reduces the visual elements to their essentials: a thin line that extends from the baseline and a dot that marks the actual value. This design improves the data-ink ratio, using less visual clutter to convey the same information, and proves especially useful when many categories are plotted side by side. In cases where bars would overlap or appear cramped, the lollipop chart preserves clarity, allowing the viewer to distinguish even categories with very similar values.

However, the format has its compromises. Because the dot marking each value slightly extends beyond the actual coordinate, there is a risk of visual imprecision. The elegance and readability of the display are purchased at the cost of some exactness. For exploratory analysis or broad pattern recognition, this trade-off is often acceptable, but for precision-critical tasks, bars or labeled points may be more appropriate.

In the example using `religion` and `Income`, the dataset lists faith traditions alongside their reported income levels. To ensure the chart communicates comparisons meaningfully, the `forcats` package's `fct_reorder()` function is used to reorder the factor levels of `religion` based on income values. Instead of an arbitrary alphabetical sequence, the categories appear sorted by income, letting patterns emerge naturally. `geom_point()` adds the distinctive orange dots, each one representing the income of a religious group. Behind each dot, `geom_segment()` introduces thin gray lines, stretching from a reference baseline (the overall mean) up to the group's value, guiding the eye to see how far above or below average each income lies. Finally, a

subtle horizontal marker is drawn with `geom_hline()`, tracing the overall mean income across the chart. This reference line functions as a yardstick, making it immediately apparent which groups rise above the population average and which fall short.

```
library(forcats)
Religion <- data.frame(
  religion = c("Agnostic", "Atheist", "Buddhist", "Catholic", "Refused",
"Evangelical Prot", "Hindu", "Historically Black Prot", "Jehovah's Witness",
"Jewish", "Mainline Prot", "Mormon", "Muslim", "Orthodox", "Other Christian",
"Other Faiths", "Other World Religions", "Unaffiliated"),
  Income = c(84, 74, 53, 633, 18, 414, 54, 78, 6, 151, 634, 42, 6, 46, 12, 41,
4, 258))
Religion%>%
  mutate(religion=fct_reorder(religion,Income)) %>%
  ggplot(aes(x=religion,y=Income))+
  geom_point(position = position_dodge(.6),
            color="orange",
            key_glyph="polygon")+
  geom_segment(aes(x=religion,
                  y=mean(Religion$Income),
                  xend = religion,
                  yend=Income),
              colour="grey",
              size=1)+
  geom_hline(yintercept = mean(Religion$Income),
            color="grey")+
  xlab(" Religion")+
  ylab("Dollars")+
  scale_y_continuous(breaks = seq(0, 650, by = 150)) +Mytheme
  labs(title="Income Earners Above 150 Dollars",fill=NULL)
```

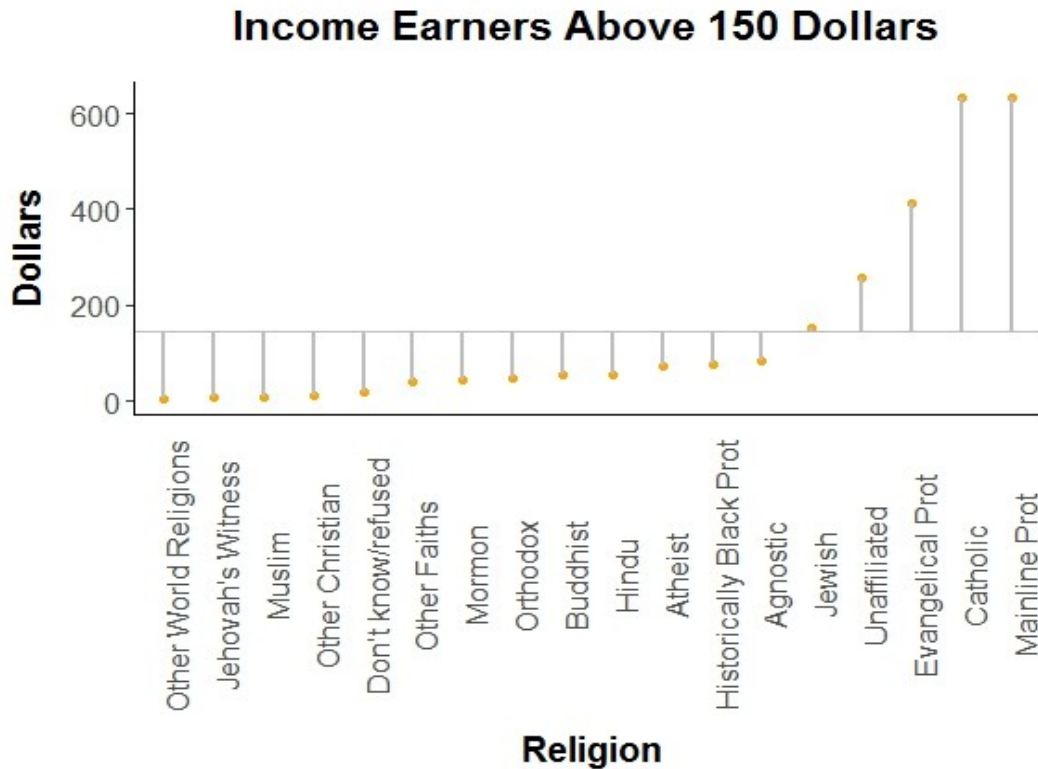


Figure 9. 2: Lollipop Chart

The outcome is a visualization that balances elegance and readability, emphasizing relative differences without the visual noise of a traditional bar chart. By structuring the categories through sorting and adding context with the baseline, the lollipop chart helps the viewer not only see variation but also anchor those differences in a meaningful statistical frame. What this figure clearly shows is the ranking of different religions by their average income, ordered from lowest to highest. The points indicate the income level for each religion, while the grey vertical lines connect each point to the overall mean income line, which sits at approximately 148 dollars. Starting from the lowest, "Other World Religions" and "Jehovah's Witness" have incomes well below the mean, around 4 and 6 dollars respectively. On the higher end, "Mainline Prot" and "Catholic" stand out with incomes exceeding 600 dollars. The grey horizontal line marking the mean provides a clear benchmark for comparing individual religions' income levels. This visual effectively ranks the religions by income and highlights those significantly below or above the average.

9.3 Dot Plot

A dot plot gives you the shape of a distribution without the visual bulk of bars. You line up categories on one axis and place individual observations as dots along the other, so you can see where values cluster, where gaps appear, and where outliers sit. When you group heights by girth categories, you immediately read the vertical stacks as local density: taller stacks signal many trees sharing similar heights, shorter stacks mark sparse regions.

The workhorse is `geom_dotplot()`, which bins along the y direction using `binaxis="y"` and stacks dots symmetrically with `stackdir="center"`. Each dot stands for one or more observations in a bin, so you control granularity with `binwidth` and dot appearance with `dotsize`. A touch of transparency via `alpha = 0.5` helps you see overlapping dots in dense zones. Because `geom_dotplot()` draws filled circles, you should map group identity to fill and set the palette with `scale_color_manual()`; reserve color for outlines if you want a crisp edge around each dot. To keep the plot from feeling cramped at the top, expand the y-scale slightly so the tallest stack has breathing room rather than colliding with the panel border.

Conceptually, a dot plot serves the same purpose as a violin or boxplot, summarizing a distribution by category, but it does so by showing the data granules themselves. A violin smooths a density estimate; a boxplot compresses to five numbers; a dot plot retains the discrete counts and makes them visible as stacks. If you ever need to emphasize individual observations atop a summary, you can layer `geom_jitter()` on a boxplot, but when your goal is to display frequency directly, the stacked dots already carry that information cleanly. Further, remember that the perceived precision depends on your binning: choose a binwidth that reflects the measurement resolution of height so the stacks represent meaningful units rather than arbitrary cuts.

```
mytreez<-mytrees
mytreez%>%
  ggplot(aes(x=Girth,y=Height, colour = Girth))+
  geom_dotplot(binaxis="y",
               stackdir="center",
               dotsize=1,
               alpha = 0.5)+
  scale_color_manual(values = c("blue", "green", "pink"))+
  scale_y_continuous(expand = expansion(mult = c(0, 0.1)))+
  xlab("Girth")+
  ylab("Height")+Mytheme
  labs(title="Dot Plot ", fill=NULL)+
```

```
guides(fill=guide_legend(keywidth=0.8,  
                          keyheight=0.8,  
                          default.unit="cm"))
```

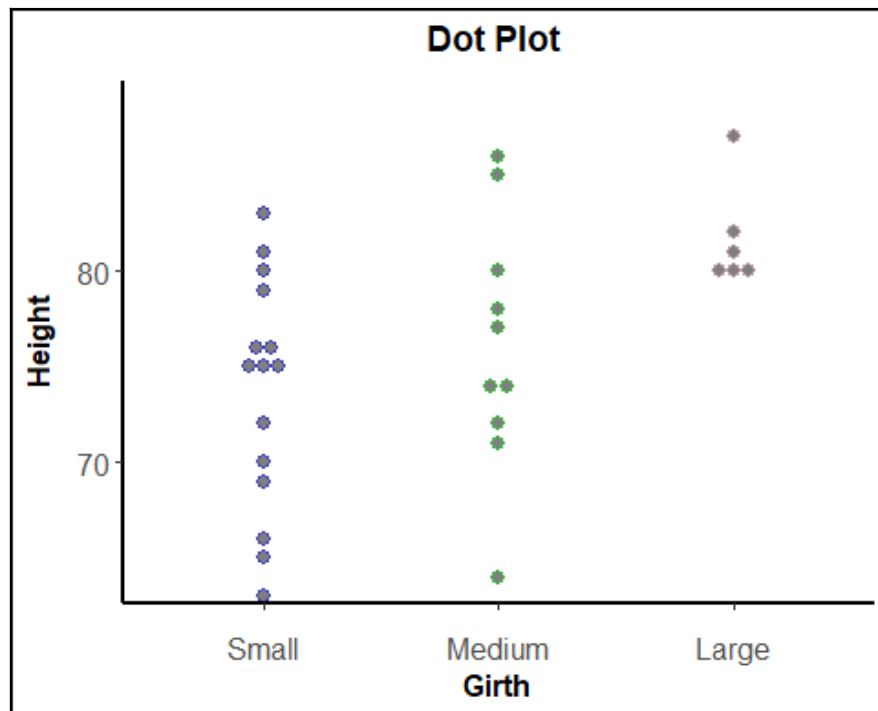


Figure 9. 3: Dot Plot

The dot plots represent individual height measurements for the **Small**, **Medium**, and **Large** girth groups, colored blue, green, and pink respectively. The height with the highest number of dots indicates the most common height for each category. From the figure, the **Medium** girth group ranks highest with around 20 to 25 dots clustered near the 70–75 height range, indicating the most frequent heights. The **Large** group follows, showing about 15 to 20 dots clustered near heights of 60–70. The **Small** group ranks lowest, with roughly 10 to 15 dots mostly below 60 in height. This ranking by dot count effectively highlights that the **Medium** girth category dominates in terms of height frequency, followed by the **Large** and then the **Small** groups. This ranking by dot count effectively highlights which girth categories dominate in terms of height frequency.

9.4 Bump Chart

A bump chart is designed to emphasize how rankings shift across categories. Instead of comparing absolute values, it emphasizes movement in relative position, making it easy to see which items rise or fall as the grouping changes. In this case, the data is divided by girth, and within each girth group the trees are ranked by height, with the tallest tree receiving rank 1.

Because the y-axis is reversed, rank 1 appears at the top, creating an intuitive reading of the chart.

The visualization combines several layers for clarity. `geom_bump()` creates smooth connecting curves that trace each tree's rank across the girth categories, allowing viewers to follow their trajectory at a glance. `geom_point()` places markers on each ranking position, while `geom_text()` attaches the tree IDs directly to those markers, ensuring that the reader can identify them without referring back to a legend. Color further reinforces the grouping, with pink, green, and blue distinguishing the different girth categories. Taken together, these elements highlight both the relative movement of each tree and the broader ranking patterns across the dataset.

```
library(ggplot2)
library(dplyr)
library(ggbump)

# Prepare data with reordered Girth levels
mytrees <- trees %>%
  mutate(
    Girth = cut(Girth, breaks = 3, labels = c("Small", "Medium", "Large")),
    Girth = factor(Girth, levels = c("Large", "Medium", "Small")), # Reorder
    Tree = paste0("Tree_", row_number())
  )

# Creating ranks of Height within Girth
bump_data <- mytrees %>%
  group_by(Girth) %>%
  arrange(desc(Height)) %>%
  mutate(Rank = row_number()) %>%
  ungroup()

# Bump Chart with reordered Girth
ggplot(bump_data, aes(x = Girth, y = Rank, group = Tree, color = Girth)) +
  geom_bump(size = 1.2, alpha = 0.7) +
  geom_point(size = 3) +
  geom_text(aes(label = Tree), hjust = -0.2, size = 3, color = "black") +
  scale_y_reverse(breaks = 1:max(bump_data$Rank)) +
  scale_color_manual(values = c("pink", "green", "blue")) +
  labs(
    title = "Bump Chart",
    x = "Girth",
    y = "Height Rank (1 = Tallest)"
  ) +
  theme_minimal(base_size = 12) +
  theme(
    panel.grid = element_blank(),
    axis.line = element_line(color = "black"),
    plot.title = element_text(face = "bold", hjust = 0.5),
    axis.title = element_text(face = "bold"),
```

```
plot.background = element_rect(color = "black", linewidth = 1)
)
```

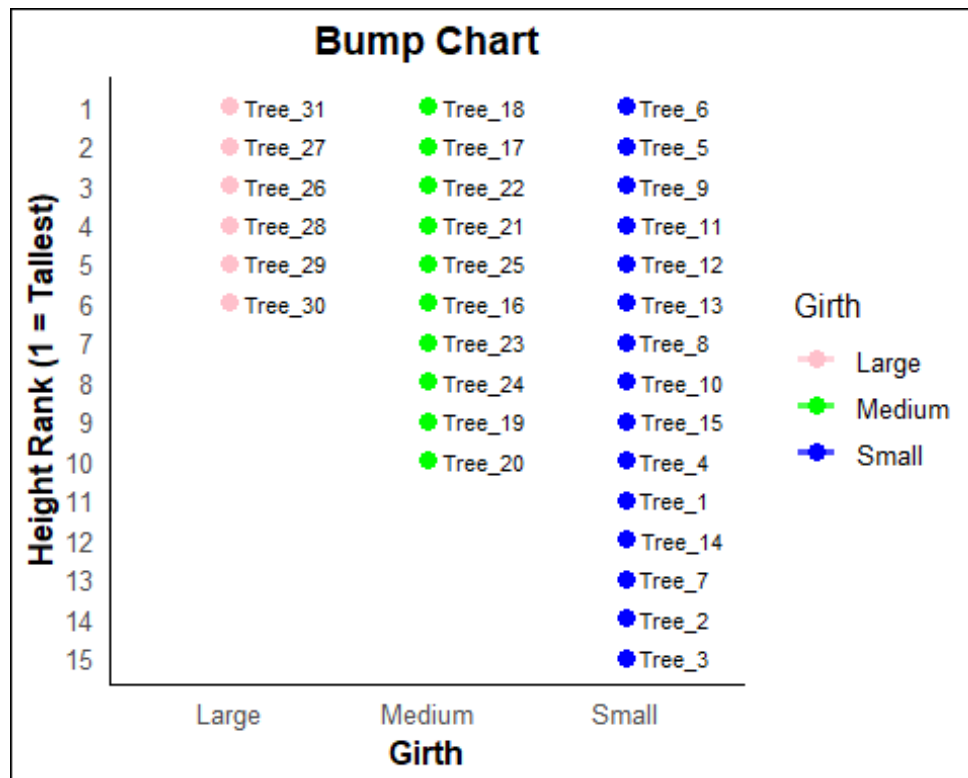


Figure 9. 4: Bump Chart

The data is prepared to order Girths as **Large**, **Medium**, and **Small**, reflecting a deliberate ranking perspective. Each tree receives a rank based on its Height in descending order within its respective Girth group. For example, in the **Large** girth category, Tree_5 holds the top rank with the greatest **Height**, while Tree_12 ranks lower within the same group. Similarly, in the **Medium** group, Tree_18 stands out near the top of the ranking, and Tree_22 is further down the list. The **Small** girth category features Tree_29 as one of the taller specimens, contrasted with Tree_31 at a lower rank. This ranking setup highlights how the tallest trees in the **Large** girth category generally outperform those in **Medium** and **Small** categories. It effectively visualizes hierarchical positioning, where trees with higher ranks indicate greater Height relative to peers in the same Girth, allowing for straightforward and insightful comparisons across groups and individual trees.

9.5 Ordered Tables

Ordered tables are an elegant way of visualizing rankings, especially when you're more interested in the order of items than the actual values themselves. This technique is often

overlooked, but it's incredibly effective for showcasing top performers, worst cases, or any sort of comparative hierarchy. Using the `mtcars` dataset, we can rank cars by their miles per gallon (`mpg`) and arrange them in descending order. We employ `dplyr()` to create the ranking and `ggplot2()` with `geom_col()` to represent the values.

The trick lies in converting the car names to a factor ordered by mpg. Once this is done, we can use `coord_flip()` to display the bars horizontally, making it easier to read the labels. The result is a sorted bar chart that doubles as a visual table.

```
mtcars$car <- rownames(mtcars)
ranked_mtcars <- mtcars %>%
  arrange(desc(mpg)) %>%
  mutate(car = factor(car, levels = car))
ggplot(ranked_mtcars, aes(x = car, y = mpg)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Ordered Tables ")
```

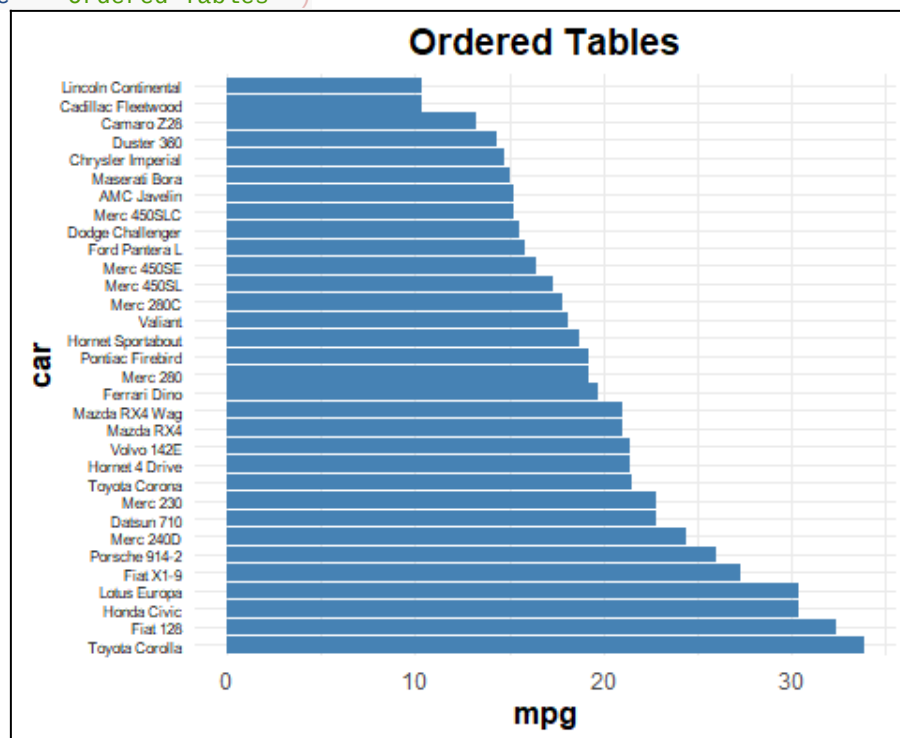


Figure 9. 5: Ordered Tables

This figure clearly shows the ranking of cars based on their miles per gallon (`mpg`) values, arranged from highest to lowest. The bars represent individual cars, with their names ordered along the vertical axis according to their mpg performance. For instance, the car with the highest `mpg` appears at the bottom of the chart, such as *Honda Civic*, while models with lower `mpg`, like *Cadillac Fleetwood*, are positioned at the top. This layout makes it easy to see which cars offer

the best fuel efficiency relative to others, enabling straightforward comparison and identification of top performers in the dataset.

CHAPTER 10: GEOSPATIAL INFORMATION

The major role of geospatial data is to map information to physical locations in order to identify patterns, trends, and spatial distributions. By linking data to specific geographic coordinates, we gain deeper insights into how phenomena vary across space. To visualize this information effectively, we use tools such as choropleth maps, symbol maps, heatmaps, and cartograms. These visualizations help translate complex datasets into easy-to-understand geographical representations. For example, choropleth maps use varying shades or colors to show differences in data values across regions, while heatmaps highlight areas of high or low activity. Such maps are particularly useful in illustrating population dynamics, tracking epidemic outbreaks, analyzing urban development, and mapping the distribution of natural resources and physical features across different localities. Overall, geospatial visualizations play a vital role in decision-making, planning, and resource allocation.

Visual clarity in maps can be compromised when numerous features are displayed simultaneously. One effective way to enhance readability is by narrowing the focus to a single geographic unit. Shapefiles are usually loaded with a lot of information and it is not practically possible to show all pieces of information in a single geospatial map. People often run into problems of visual clutter when such bulky information is not filtered. In this regard and for the purposes of illustrating geospatial, Gwassu South and Gwassu North wards within the broader Suba region in Homa Bay County, are selected by filtering the **IEBC_WARDS** attribute from a shapefile that includes all wards in Kenya.

10.1 Spatial Mapping

Visual clarity may require presentation in a three-dimensional view in order to understand interactions in the real world better. Such is the use of spatial mapping. It represents the spatial distribution of features within a defined geographical region while allowing for the interpretation of both the boundaries of a given area and the spatial arrangement of points of interest, such as facilities or landmarks. Geospatial charts of this nature are instrumental in infrastructure planning, site analysis, or service delivery optimization.

One needs to perform a few tasks in order to come up with a geospatial representation in a chart. A shapefile containing administrative units is necessary, in this case Kenya wards.shp and any other file that contains a geospatial feature. In this text, **Suba.CSV** (See Appendix I) file is used in

complementary to the `Kenya wards.shp` shapefile in the construction of the geospatial maps. A shapefile is read with the help of `st_read()`, setting its coordinate reference system (CRS) to EPSG:4326, making it possible to transform to EPSG:21036 for local projection accuracy using `st_transform()` command. The `filter()` function isolates Gwasssi South from the larger set of Kenyan wards. Separately, point data from a `Suba.csv` is converted into a simple features (`sf`) object via `st_as_sf()`, with longitude and latitude specifying coordinates. These points are reprojected to match the ward's CRS and spatially filtered so that only those lying within `Gwasssi South` remain (`suba_gwasssi`). The `ggplot` call layers the polygon geometry of the ward `geom_sf()` and overlays the points, colouring them by their group attribute (`Category`). In a case different groups needs to be distinguished, `geom_image()` adds icons to the features for visual distinction.

```
library(sf)
library(ggplot2)
library(ggimage)
library(ggrepel)
library(dplyr)
library(readr)
# Load and transform the wards shapefile
wards <- st_read("D:/ALL/Project/Shape files/kenya_wards/kenya_wards.shp") %>%
  st_set_crs(4326) %>%
  st_transform(21036) %>%
  st_make_valid()
# Filter to Gwasssi South ward only
gwassi_ward <- wards %>%
  filter(IEBC_WARDS == "Gwasssi South")
# Load feature data and convert to sf object
places <- read_csv("D:/ALL/Project/Shape files/Suba.csv") %>%
  mutate(
    Longitude = as.numeric(Longitude),
    Latitude = as.numeric(Latitude)
  )
places_sf <- st_as_sf(places, coords = c("Longitude", "Latitude"), crs = 4326) %>%
  st_transform(21036)
# Spatial join to ward and filter again to Gwasssi South
places_joined <- st_join(places_sf, wards, join = st_intersects)
filtered_sf <- places_joined %>%
  filter(IEBC_WARDS == "Gwasssi South") %>%
  mutate(Category = as.character(Category))
# Extract coordinates and drop geometry for plotting
filtered_df <- filtered_sf %>%
  mutate(
    x = st_coordinates(geometry)[, 1],
    y = st_coordinates(geometry)[, 2]
  ) %>%
  st_drop_geometry()
# Assign valid icons (hosted on Flaticon)
icon_map <- c(
  "Church" = "https://cdn-icons-png.flaticon.com/512/1998/1998787.png",
  "School" = "https://cdn-icons-png.flaticon.com/512/1070/1070890.png",
  "Health Facility" = "https://cdn-icons-png.flaticon.com/512/2965/2965567.png",
  "Centre" = "https://cdn-icons-png.flaticon.com/512/2933/2933916.png",
  "Organizations" = "https://cdn-icons-png.flaticon.com/512/3405/3405676.png",
```

```

"Office" = "https://cdn-icons-png.flaticon.com/512/3135/3135715.png"
)
filtered_df$icon <- icon_map[filtered_df$Category]
# Create the map
ggplot() +
  geom_sf(data = gwassi_ward, fill = "gray95", color = "black") +
  geom_image(
    data = filtered_df,
    aes(x = x, y = y, image = icon),
    size = 0.05
  ) +
  geom_text_repel(
    data = filtered_df,
    aes(x = x, y = y, label = Name),
    size = 3,
    max.overlaps = 30,
    box.padding = 0.4
  ) +
  labs(
    title = "Geospatial Distribution",
    subtitle = "Places in Suba South",
    caption = "Source:IEBC Wards, 2019"
  ) +
  theme

```

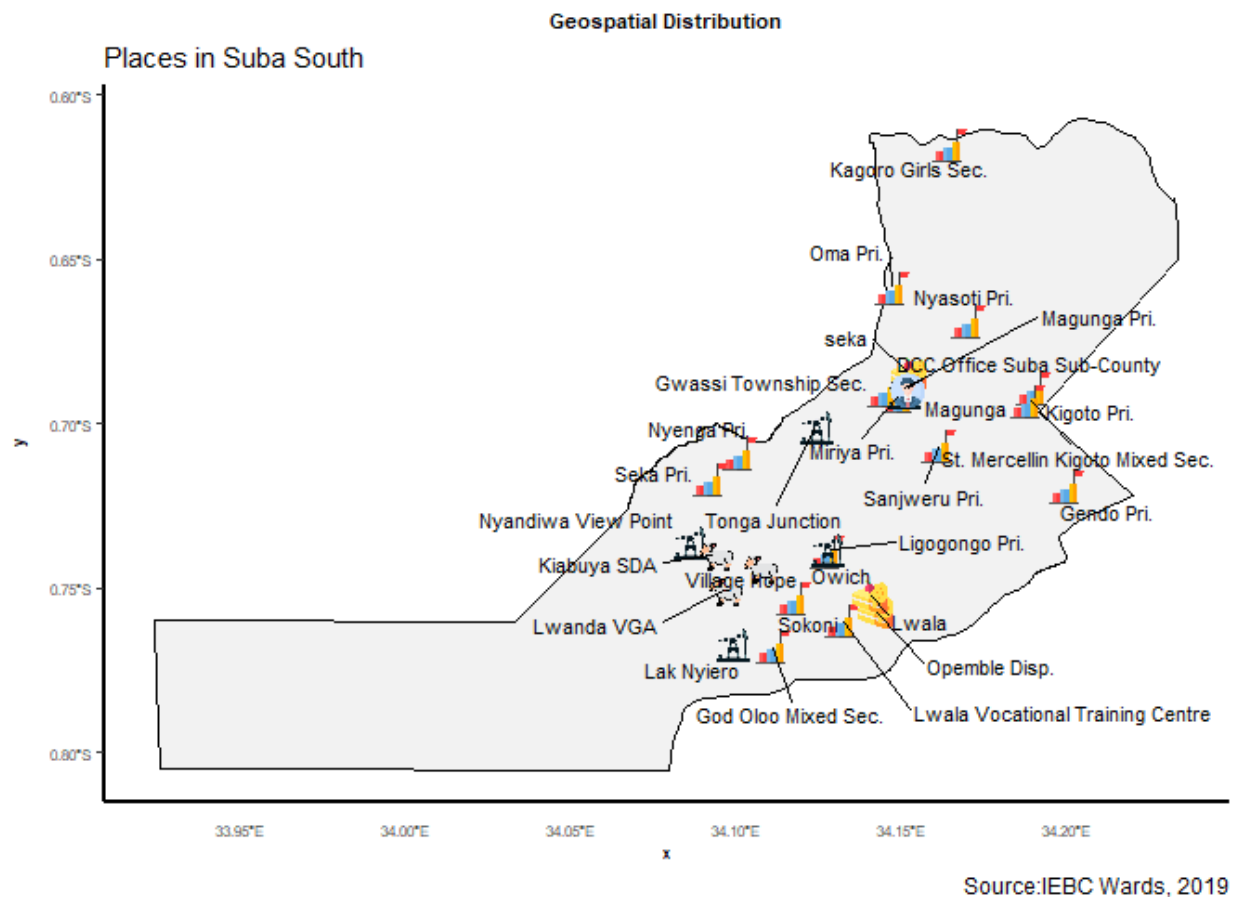


Figure 10. 1: Geospatial Mapping

In the output, users can readily see how different categories of features are distributed within the mapped region. Areas with dense clustering, sparse distribution, or absence of certain facility types become immediately apparent. Visually, the result is a clean map with the ward outlined in grey and the points vividly differentiated by category. The viewer can immediately discern patterns. For instance, majority of the facilities cluster near the centre, making it possible to associate real-world phenomena with specific geographic locations. This level of spatial clarity supports evidence-based decisions in resource allocation or development planning.

10.2 Geospatial Labeling

In contexts where the identity of individual features matters as much as their position, labeling becomes essential. By adding names directly to the map, one facilitates quick recognition without referring to an external legend or dataset. This is particularly useful in localized mapping where stakeholders may already be familiar with place names.

The construction of this map mirrors the first example in its preparation of spatial data, but here an extra layer, `geom_sf_text()`, is introduced. This function draws text labels at the centroid of each point, with the label aesthetic tied to the name of the point. The argument `nudge_y = 150` is used to vertically offset labels, reducing overlap with the points themselves.

```
ggplot() +
  geom_sf(data = gwassi_south, fill = "white", color = "gray50") +
  geom_sf(data = suba_gwasssi, aes(color = Category), size = 3) +
  geom_sf_text(data = suba_gwasssi, aes(label = Name), size = 3, color = "black",
nudge_y = 150) +
  labs(title = "Labeled Features") +
  theme10
```

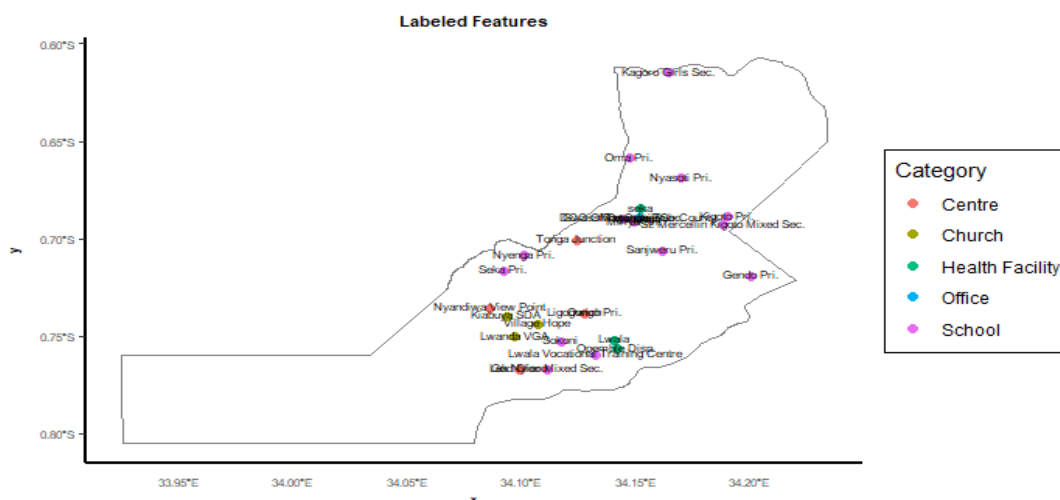


Figure 10. 2: Geospatial Labeling

The output merges spatial precision with descriptive clarity. Points remain colour-coded by category, but now each is accompanied by a name. In practice, this allows a local official, for instance, to not only see that there is a health facility in the southern zone but also to identify it by name, making the map both analytical and referential.

10.3 Annotated Map

Maps provide a vital link between data and the physical landscape, making spatial relationships easier to interpret. When multiple categories of locations are spread across a region, visual clarity becomes essential. This type of visualization supports practical applications in areas such as planning, service delivery, and spatial analysis. It offers a clear overview of how services are distributed, identifies areas of concentration, and highlights gaps that may require attention.

The chart under discussion presents a spatial distribution map using intuitive icons to represent various categories of community interest. In this example, the map highlights schools, churches, health facilities, offices, and centers within Gwasssi South ward in Kenya. Instead of relying on conventional dots or color gradients, symbolic imagery is used to enhance both comprehension and visual appeal. This style of mapping is particularly effective in communicating complex data to both technical and non-technical audiences.

The `st_read()` function from the `sf` package reads spatial vector data directly from a shapefile. This is followed by `st_set_crs()` to define the initial coordinate reference system, and `st_transform()` to project the geometry into a metric system suitable for spatial analysis. `st_transform()` ensures the geometries are clean and topologically correct, which is critical when performing operations like joins or intersections.

To isolate a specific region, the `filter()` function from `dplyr` is used. This narrows the dataset to a single ward based on a matching attribute, in this case, the ward name “Gwasssi South.” Point data from a CSV file is read using `read_csv()` and converted into an `sf` object using `st_as_sf()`, which interprets the coordinate columns (longitude and latitude) as spatial points. These are then transformed to match the projection used for the ward boundaries using `st_transform()`.

A spatial join is executed using `st_join()` to associate each point with its corresponding polygon region or name. This enables filtering to only retain features located within the Gwasssi South boundary. After the spatial relationship is established, `st_coordinates()` extracts numeric

coordinate values from the spatial points, while `st_drop_geometry()` simplifies the dataset by removing the spatial attributes for plotting purposes.

To enhance visual clarity, each category is mapped to a unique icon using a named vector that pairs category labels with image URLs. This lookup process assigns a meaningful image to every feature, enabling symbolic representation during plotting. The `geom_sf()` function from `ggplot2` draws the base map, representing the ward's boundary. The `geom_image()` function, from the `geom_image()` package, overlays icons at the locations of each point feature. Labeling is handled using `geom_text_repel()` from `ggrepel`, which dynamically adjusts text positions to avoid overlapping, preserving both legibility and layout quality.

```
# Load and transform the wards shapefile
wards <- st_read("D:/ALL/Project/Shape files/kenya_wards/kenya_wards.shp") %>%
  st_set_crs(4326) %>%
  st_transform(21036) %>%
  st_transform(
gwassi_ward <- wards %>% #Filter to Gwassi South ward only
  filter(IEBC_WARDS == "Gwassi South")
places <- read_csv("D:/ALL/Project/Shape files/Suba.csv") %>%
  mutate(
    Longitude = as.numeric(Longitude),
    Latitude = as.numeric(Latitude)
  ) # Load feature data and convert to sf object
places_sf <- st_as_sf(places, coords = c("Longitude", "Latitude"), crs = 4326) %>%
  st_transform(21036)
#Spatial join to ward and filter again to Gwassi South
places_joined <- st_join(places_sf, wards, join = st_intersects)
filtered_sf <- places_joined %>% # Extract coordinates and drop geometry for plotting
  filter(IEBC_WARDS == "Gwassi South") %>%
  mutate(Category = as.character(Category))
filtered_df <- filtered_sf %>%
  mutate(
    x = st_coordinates(geometry)[, 1],
    y = st_coordinates(geometry)[, 2]
  ) %>%
  st_drop_geometry()

# Assign valid icons (hosted on Flaticon)
icon_map <- c(
  "Church" = "https://cdn-icons-png.flaticon.com/512/1998/1998787.png",
  "School" = "https://cdn-icons-png.flaticon.com/512/1070/1070890.png",
  "Health Facility" = "https://cdn-icons-png.flaticon.com/512/2965/2965567.png",
  "Centre" = "https://cdn-icons-png.flaticon.com/512/2933/2933916.png",
  "Organizations" = "https://cdn-icons-png.flaticon.com/512/3405/3405676.png",
  "Office" = "https://cdn-icons-png.flaticon.com/512/3135/3135715.png"
)
filtered_df$icon <- icon_map[filtered_df$Category]
ggplot() +
  geom_sf(data = gwassi_ward, fill = "gray95", color = "black") +
  geom_image(
    data = filtered_df,
    aes(x = x, y = y, image = icon),
    size = 0.05
```

```

) +
geom_text_repel(
  data = filtered_df,
  aes(x = x, y = y, label = Name),
  size = 3,
  max.overlaps = 30,
  box.padding = 0.4
) +
labs(
  title = "Annotated Map",
  subtitle = "Places in Suba South",
  caption = "Source:IEBC Wards"
) +
  mytheme

```

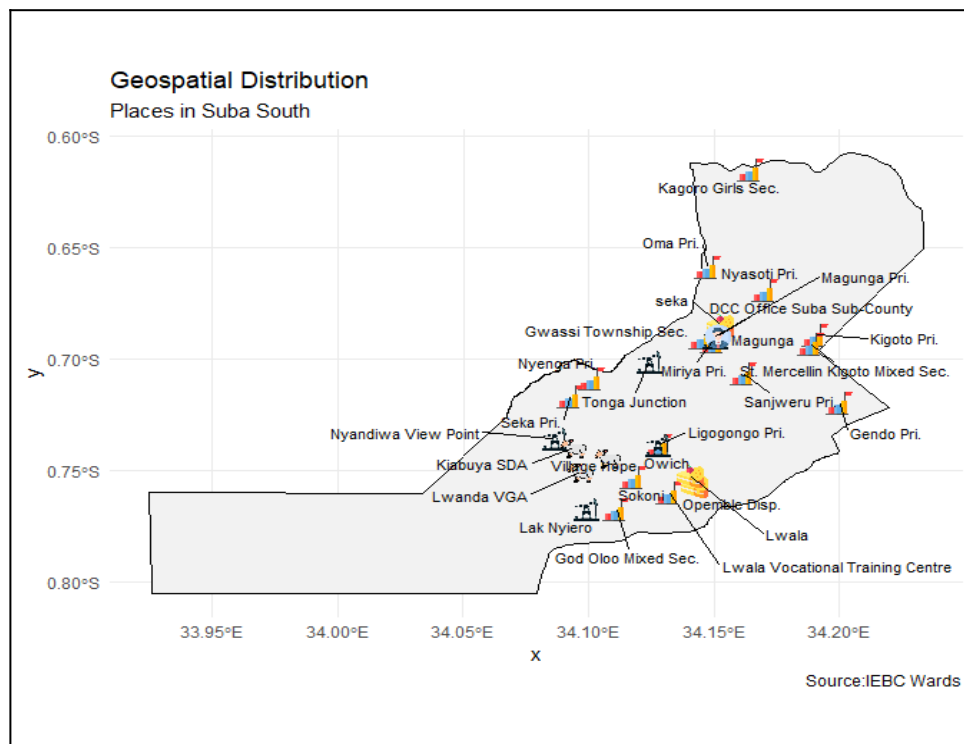


Figure 10. 3: Annotated Maps

The resulting map offers a polished and informative visualization of the Gwasssi South ward, where important community features are clearly marked and classified. The icons instantly communicate the type of each location without requiring viewers to decode symbols or refer to a legend. This makes the map accessible to a wide audience, including those without technical mapping expertise.

Each icon is accompanied by a text label that names the feature. The placement of labels is handled in a way that minimizes clutter and maintains clarity, even when multiple features are close together. The ward boundary is displayed in neutral tones, allowing the icons and labels to stand out and remain the focal point of the map.

10.4 Choropleth Map

A choropleth map is a thematic map where areas are shaded or patterned in proportion to the value of a variable, offering a powerful way to represent geospatial distributions. Quantifying and comparing the intensity of a feature across administrative boundaries is the purpose of a choropleth map. In this case, the number of schools in Gwasssi South and Gwasssi North wards is aggregated and visually encoded through a colour gradient. Such maps are instrumental in revealing disparities in resource distribution.

The number of schools per ward is counted, producing `School_Count`. Only Gwasssi South and Gwasssi North are retained, and missing counts are replaced with zeros for wards without schools. Centroids of these polygons are computed for labelling purposes. The `geom_sf` layer fills wards according to their `School_Count` values, while `geom_sf_text` adds the ward names. A `scale_fill_gradient` conveys density, transitioning from pale yellow for few schools to dark red for many.

```
ggplot(gwasssi_wards) +  
  geom_sf(aes(fill = School_Count), color = "gray30", size = 0.3) +  
  geom_sf_text(data = label_points, aes(label = IEBC_WARDS), size = 5, fontface =  
"bold") +  
  scale_fill_gradient(low = "lightyellow", high = "darkred") +  
  labs(  
    title = "Simulated Chloropleth Map",  
    subtitle = "School Densities",  
    fill = "Number of Schools"  
  ) +  
  theme_mytheme
```

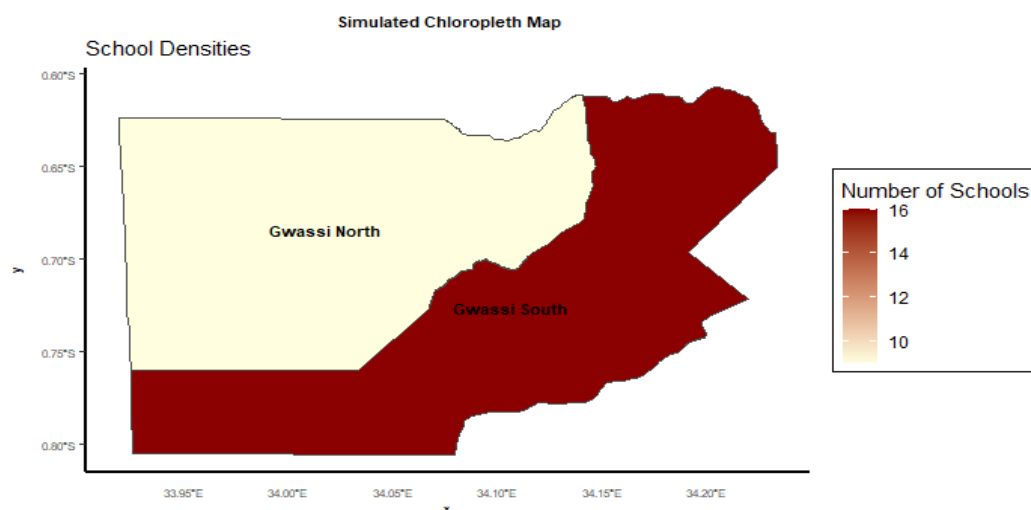


Figure 10. 4: *Chloropleth Map*

The resulting map offers a visually intuitive overview of how educational institutions are spread across the two wards. Gwasssi South and Gwasssi North are each highlighted with labels, and the color intensity tells the viewer how many schools are located in each region. A darker blue in Gwasssi South, for instance, indicates a higher concentration of schools compared to Gwasssi North. When viewed, the map's shading immediately communicates which ward holds more educational facilities. This visual hierarchy directs attention to areas potentially in need of more investment, serving as a starting point for policy discussions on equitable distribution.

10.5 Labeled Choropleth Map

A choropleth map with labels is a thematic map where geographic areas are shaded or colored based on a specific attribute, and each region is annotated with a text label such as a name, code, or value. The use of color makes spatial differences easily recognizable, while the labels reduce ambiguity by directly identifying the areas on the map. This type of map plays an important role in geospatial analysis because it combines visual encoding of data with textual clarity, making it easier to interpret patterns, highlight disparities, and communicate spatial information effectively without over-reliance on legends or external references.

In the coding, the function `geom_sf()` is responsible for rendering the spatial features in the plot. Since the data frame contains simple feature (`sf`) objects, this layer understands how to draw the ward polygons directly, preserving their spatial geometry. The call to `scale_fill_manual(values=ward_colors)` introduces a deliberate color scheme for the wards. Instead of relying on default palettes, each ward is matched to a pre-defined color, which gives full control over the map's aesthetic balance and clarity. Adding `geom_sf_text()` enriches the map by overlaying ward labels directly on the corresponding polygons. This makes the visualization more intuitive because the viewer does not need to cross-reference a legend to identify each ward. With `legend.position = "none"`, the layout prioritizes the map itself as the main source of interpretation. Since the wards are already visually distinct and labeled, the legend becomes redundant, and its removal declutters the figure for a cleaner presentation.

```
subba <- st_read("D:/ALL/Project/Shape files/kenya_wards/Kenya wards.shp") %>%
  st_set_crs(4326) %>%
  st_transform(21036) %>%
  filter(FIRST_DIST == "Suba")
# Define your 9 colors
ward_colors <- c("gray", "lightyellow", "lightgreen", "lightblue",
```

```

"#FFFDE7", "#FFEFD5", "#E8F5E9", "#FFF0F5", "#F3E5F5")
# Plot with geom_sf
ggplot(data = subba) +
  geom_sf(aes(fill = IEBC_WARDS), color = "black", size = 0.3) +
  scale_fill_manual(values = ward_colors) +
  geom_sf_text(aes(label = IEBC_WARDS), size = 3, color = "black") +
  labs(title = "Nine Wards of Suba") +
  theme(legend.position = "none")+Mytheme # <-- removes legend

```

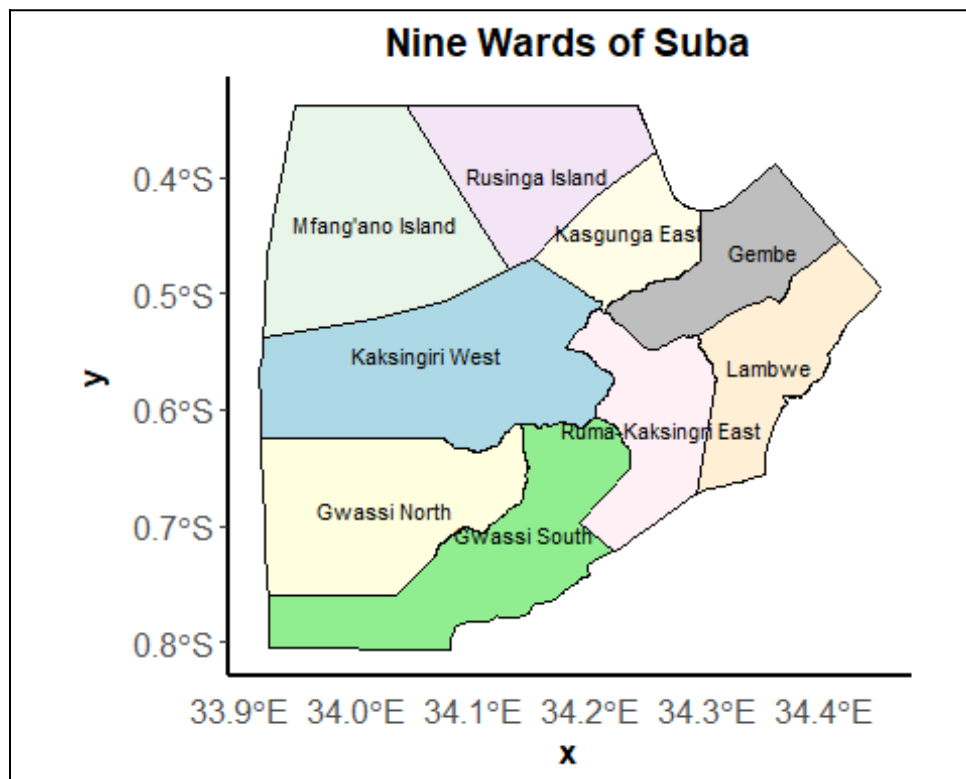


Figure 10. 5: Labeled Choropleth Map

When looking at the output of the choropleth map with labels, the first thing that stands out is how the nine wards of Suba region are shaded in distinct colors, each corresponding to a unique ward in the dataset. Unlike a map with only outlines, the filled colors make the boundaries more pronounced and visually distinguishable, helping one immediately recognize the extent of each ward. The labels placed directly on the shapes, drawn from the wards field, ensure that each area is explicitly identified by name without needing to consult a legend.

For instance, wards such as Kaksingiri West and Gwassi North appear respectively with lightblue and lightyellow colors assigned them, while the label printed on the polygon confirms their identity. This combination of color coding and text allows both pattern recognition and precise identification in one view. From a geospatial perspective, such an output is valuable in local

governance or service planning, as it quickly reveals the administrative layout of Suba while minimizing the risk of misinterpretation. The map is therefore not just a visualization of space, but a practical tool for decisions about resource allocation, development initiatives, and comparative analysis among wards.

10.6 Symbol Map

When the focus is on emphasizing individual locations rather than aggregated values, a symbol map provides a clear solution. Here, each point is represented by a proportional marker whose appearance communicates categorical information.

In this illustration, let's use the already prepared Gwasssi South polygon and extract only the relevant points within its boundary. The `geom_sf()` function plots the ward outline, and `geom_sf()` is used again for the points, but now the size aesthetic is explicitly set and the legend is simplified with `show.legend = "point"`. The color aesthetic remains tied to the `Category` attribute, ensuring categorical differentiation.

```
ggplot() +  
  geom_sf(data = gwassi_south, fill = "white", color = "gray60") +  
  geom_sf(data = suba_gwasssi, aes(size = 2.5, color = Category), show.legend =  
"point") +  
  labs(title = "Symbol Map") +  
  Mytheme
```

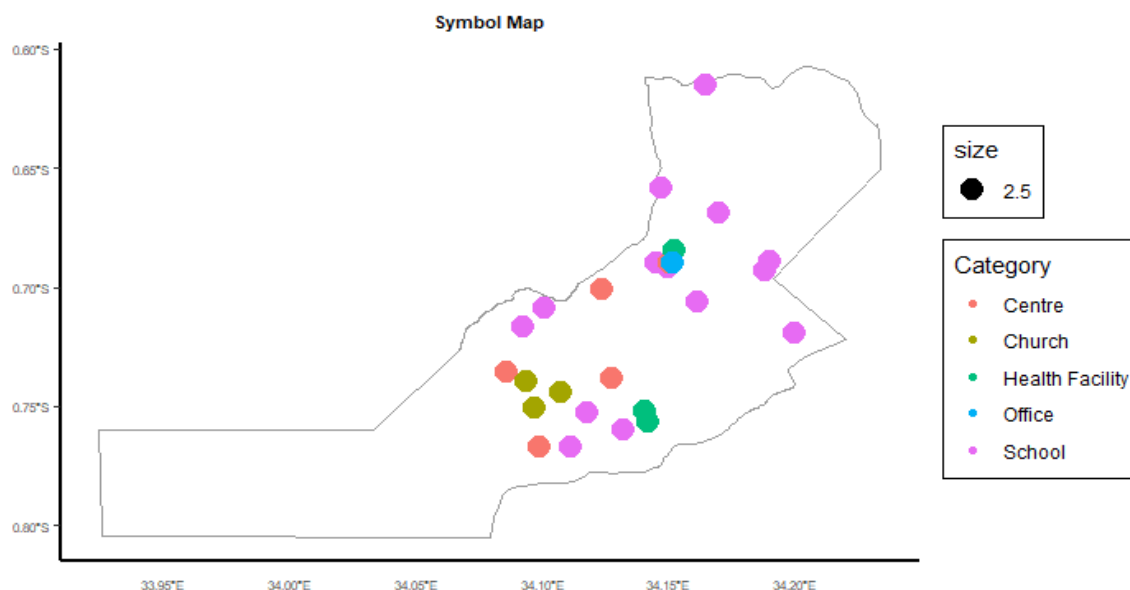


Figure 10. 6: Symbol Map

The map conveys a clean impression of feature placement without numerical summarization. For example, schools, churches, and other categories can be quickly distinguished, and their relative proximity to one another becomes apparent, insights that can guide accessibility studies or service coverage analysis. Each symbol's placement corresponds to real spatial coordinates, helping users identify clusters, outliers, or sparsely served areas. It's an intuitive and accessible way of portraying location-specific data.

10.7 Tile Mapping

Just as the name suggests, this is an info-graphic construction in a manner that divides the spatial plane into uniform rectangles, offering a simplified view of how points are distributed spatially. Tile maps allow the analysis of spatial patterns at a regular, uniform scale, independent of administrative boundaries. This graphic technique is especially valuable for density mapping or when comparing equal-sized spatial units.

The code uses `st_make_grid()` to generate a lattice of cells over Gwasssi South, with each cell measuring 1,000 units in size. By computing the lengths of intersections between each grid cell and the set of points `lengths(st_intersects(grid, suba_gwasssi))`, a count of features per cell is assigned. This grid, combined with the ward outline, is plotted using `geom_sf()` for the first method or `geom_tile()` after converting centroids into a standard dataframe for the alternative approach. A gradient fill visually encodes the counts, with lighter colours for sparse areas and darker shades for dense clusters.

```
grid <- st_make_grid(gwasssi_south, cellsize = 1000) %>%
  st_sf() %>%
  mutate(id = row_number())
grid$count <- lengths(st_intersects(grid, suba_gwasssi))
ggplot() +
  geom_sf(data = grid, aes(fill = count), color = NA) +
  geom_sf(data = gwasssi_south, fill = NA, color = "black") +
  scale_fill_gradient(
    low = "white", # Start of gradient
    high = "darkblue", # End of gradient
    na.value = "grey90"
  ) +
  labs(title = "Tile Map", fill = "Count") +
  Mytheme
```

Alternative Approach;

```
ggplot() +
  geom_tile(data = grid_df, aes(x = X, y = Y, fill = count)) +
  geom_sf(data = gwasssi_south, fill = NA, color = "black", size = 0.4) +
  scale_fill_gradient(low = "white", high = "darkred") +
  labs(title = "Tile Map", fill = "Count") +
  coord_sf(crs = st_crs(21036)) +
  Mytheme
```

Tile Mapping Approaches

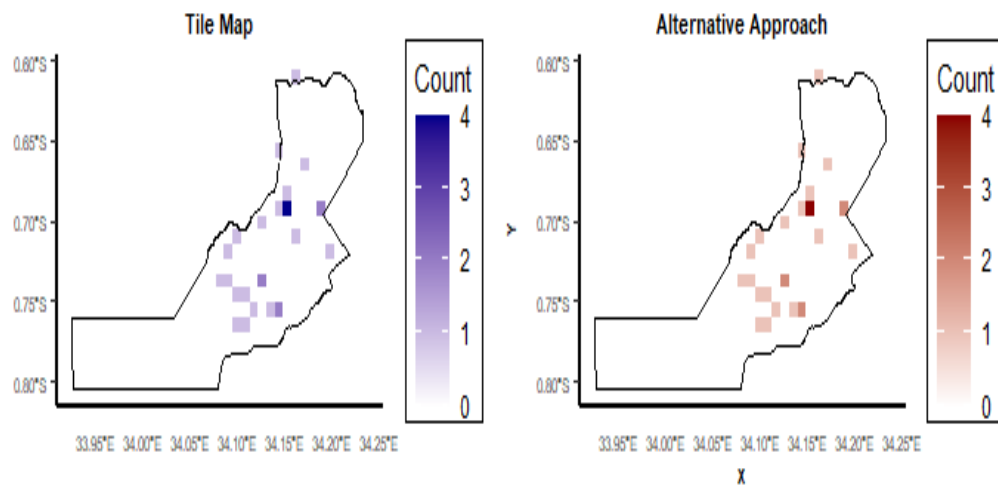


Figure 10. 7: Tile Mapping

The visual result transforms the ward into a patchwork of coloured squares, each square telling the story of localised concentration. Such granularity helps pinpoint micro-hotspots that a ward-level choropleth might conceal, offering a finer lens for intervention planning.

10.8 Gridded Map

Geometries like rectangles are useful in highlighting geospatial coverage as they draw viewers focus to a particular spatial objects. The use of gridded maps involves creating small bounding rectangles that represent defined areas around each feature, allowing for easier visualization of their local influence zones. Such an approach is common when analysts want to emphasize not just point positions, but also a uniform surrounding area. By replacing simple points with consistent rectangles, the viewer gets an immediate sense of coverage distribution across the region of interest.

A buffering step with `st_buffer(suba_gwasssi, dist = 300)` artificially expands each region point into a surrounding zone, school position in this case. While the buffer itself is not plotted, it provides positional reference for bounding rectangle creation. The plotting step uses `geom_sf()` to display the `Gwasssi_South` boundary, while `geom_rect()` manually defines each rectangle's minimum and maximum coordinates around the school's location by subtracting and adding 150 units in both the x and y directions. The rectangles are styled with an orange fill, dark red borders, and slight transparency for map clarity.

```

# Load ward shapefile and filter to Gwasssi South
gwassi_south <- st_read("D:/ALL/Project/Shape files/kenya_wards/Kenya wards.shp") %>%
  st_set_crs(4326) %>%
  st_transform(21036) %>%
  filter(IEBC_WARDS == "Gwasssi South")
# Load and transform school data
suba <- read_csv("D:/ALL/Project/Shape files/Suba.csv")
suba_sf <- st_as_sf(suba, coords = c("Longitude", "Latitude"), crs = 4326) %>%
  st_transform(21036)
# Clip schools to Gwasssi South
suba_gwasssi <- suba_sf[gwasssi_south, ]
# Generate bounding rectangles around each point (artificial expansion)
buffered <- st_buffer(suba_gwasssi, dist = 300) # create buffer to use as rect area
rect_coords <- st_bbox(buffered) # get bounding box of buffered schools
# Create bounding rectangles manually
ggplot() +
  geom_sf(data = gwassi_south, fill = "grey95", color = "black") +
  geom_rect(
    aes(
      xmin = st_coordinates(suba_gwasssi)[,1] - 150,
      xmax = st_coordinates(suba_gwasssi)[,1] + 150,
      ymin = st_coordinates(suba_gwasssi)[,2] - 150,
      ymax = st_coordinates(suba_gwasssi)[,2] + 150
    ),
    fill = "orange",
    color = "darkred",
    alpha = 0.6
  ) +
  labs(title = "Gridded Map") +
  coord_sf() +
  theme

```

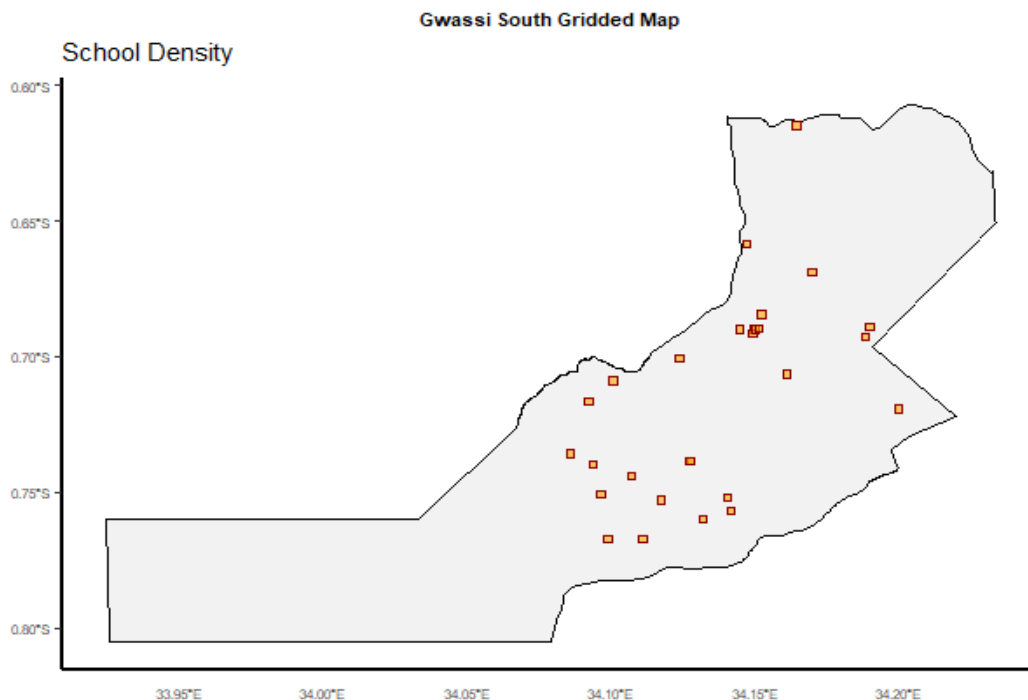


Figure 10. 8: Gridded Map

Looking at the chart, you notice that Gwasssi South appears as a soft grey polygonal boundary, while evenly sized orange rectangles are positioned where each school lies. These rectangles give the map a grid-like spatial representation of coverage, allowing one to quickly observe clusters in the north versus sparser distribution towards the southern edges. For example, if several rectangles are closely packed near the ward's central area, it signals higher educational density there. Conversely, isolated rectangles towards the periphery highlight areas with fewer facilities, perhaps guiding future planning. By standardizing the size of the rectangles, the map removes bias from actual school size and focuses purely on distribution, making spatial comparison intuitive even without detailed numerical labels.

10.9 Raster Map

While `geom_rect()` gives flexibility over rectangle boundaries, `geom_raster()` simplifies the process for evenly spaced grid data. It assumes that all raster tiles are aligned in a regular grid and that the width and height of each cell are fixed. This makes it ideal for displaying data that comes from a raster or regular grid, such as satellite imagery or interpolated surface values. In this code, each tile is automatically drawn as a square centered on the x and y coordinates, representing the center of a raster cell. Compared to `geom_rect()`, this method is computationally faster and more memory-efficient, especially when dealing with large, evenly spaced datasets. Although `geom_rect()` and `geom_raster()` might look similar at first glance, they're designed for slightly different purposes. `geom_rect()` draws individual rectangles with positions and sizes you specify, giving you full control over each shape. `geom_raster`, on the other hand, assumes a fixed grid where each cell is the same size and directly maps the values into those cells, making it much faster and better suited for true raster data. In short, `geom_rect()` is like laying bricks one by one, while `geom_raster()` is like rolling out a tiled floor all at once.

The `rasterize()` function then steps in to handle the counting, imagine walking cell by cell across this grid, checking how many facilities fall inside each one. Each cell's count becomes its "value," which later determines its colour in the raster map. Finally, `geom_sf()` is used to draw the region's outline, and `geom_raster()` paints the grid, turning those counts into a clear visual gradient.

```
# Read and prepare Gwasssi South boundaries
gwasssi_south <- st_read("D:/ALL/Project/Shape files/kenya_wards/Kenya wards.shp") %>%
  st_set_crs(4326) %>% # Assign WGS84 if missing
```

```

st_transform(21036) %>% # Project to UTM Zone 36S
filter(IEBC_WARDS == "Gwasssi South")
# Read Suba dataset ('Longitude', 'Latitude', 'Category')
points_sf <- read_csv("D:/ALL/Project/Shape files/Suba.csv") %>%
  st_as_sf(coords = c("Longitude", "Latitude"), crs = 4326) %>%
  st_transform(21036)
# Pick category to visualize
cat_name <- "School" # or "Centre", "Office", "Health Facility", etc.
points_cat <- points_sf %>% filter(Category == cat_name)
# Create empty raster over Gwasssi South
r_template <- raster(extent(gwasssi_south),
  res = 200, # Adjust resolution (meters) as needed
  crs = st_crs(gwasssi_south)$wkt)
# Rasterize points by count
r_counts <- rasterize(as(points_cat, "Spatial"), r_template,
  fun = "count", background = NA)
# Convert raster to dataframe for ggplot
r_df <- as.data.frame(r_counts, xy = TRUE)
names(r_df)[3] <- "count"
# Plot raster over ward boundaries
ggplot() +
  geom_sf(data = gwasssi_south, fill = NA, color = "black", linewidth = 0.8) +
  geom_raster(data = r_df, aes(x = x, y = y, fill = count)) +
  scale_fill_viridis_c(na.value = NA) +
  coord_sf(crs = st_crs(gwasssi_south)) +
  labs(title = paste("A Raster Map"),
    fill = "Count") +
  Mytheme

```

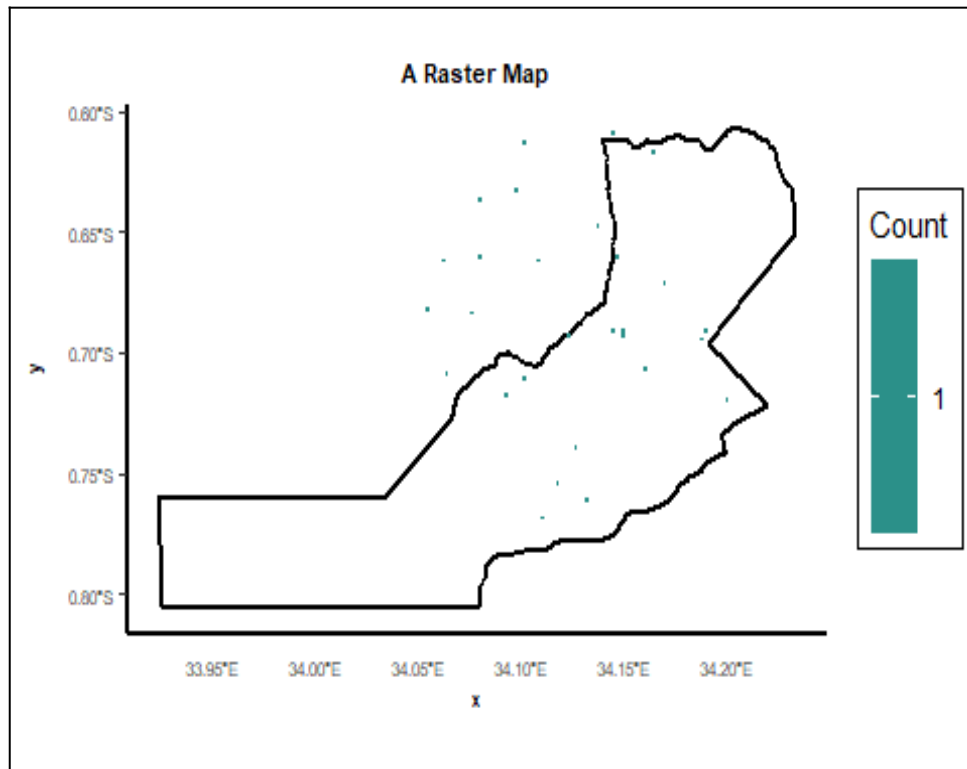


Figure 10. 9: Raster Map

The finished chart blends the ward's boundary with a colour-coded grid, each tile telling its own small story. Cells with lighter shades reveal areas where only a few facilities are found, perhaps

just a single health post or school. Darker patches signal clusters, that is, places where schools, offices, or other services are concentrated. The Gwasssi South outline acts like a frame, reminding us of the ward's geography while the colours highlight patterns within it. This way, even without reading the CSV, one can instantly see which parts of the ward are richly served and which might still need attention

10.10 Geospatial Heatmaps

A geospatial heatmap visually represents the intensity or density of features across a geographic area, using colors to indicate hotspots or concentrations. In real-life applications, heatmaps help identify areas with high activity, such as regions with many schools, crime incidents, or disease cases. By translating raw data points into a smooth gradient map, heatmaps make complex spatial patterns easier to interpret and support informed decision-making in fields like urban planning, public health, and resource management.

`st_make_grid()` creates a grid over your study area by dividing it into equally sized cells, making it easier to analyze data spatially. The `cellsize` argument decides how big each cell should be, for example, 500 means each cell is 500 by 500 units wide. Setting `square = TRUE` tells the function to make square cells rather than hexagons or other shapes. After creating this grid, `st_sf()` converts it into a format that combines geometry and attributes so R can handle it smoothly. To turn those grid cells with counts into a format fit for smooth color shading, `st_rasterize()` converts the vector grid into a raster, and `dx` and `dy` set the width and height of each pixel in this raster, matching the cell size for accuracy. Switching to `tmap_mode("plot")` puts tmap in a static map mode perfect for reports or print, while the `options = opt_tm_text()` inside `tm_text()` lets you add styling to your labels, like shadows for better visibility. Finally, `tm_raster()` plots the raster layer, showing the data as a smooth heatmap using colors to represent intensity.

```
wards_path <- "D:/ALL/Project/Shape files/kenya_wards/Kenya wards.shp"
wards <- st_read(wards_path, quiet = TRUE) |> st_set_crs(4326)
wards_gwasssi <- wards |>
  filter(str_detect(IEBC_WARDS, regex("Gwasssi South|Gwasssi North", ignore_case =
TRUE))) |>
  st_transform(21036)
places_raw <- read_csv("D:/ALL/Project/Shape files/Suba.csv", locale = locale(encoding
= "Latin1")) |>
  mutate(across(where(is.character),
    ~ .x |>
      str_replace_all("[\\u2018\\u2019\\u201A\\u201B\\u2032\\u2035]", "'") |>
      str_replace_all("[\\u201C\\u201D\\u201E\\u201F\\u2033\\u2036]", "\"") |>
```

```

    iconv(from = "latin1", to = "UTF-8", sub = ""))
  ))
places <- places_raw |>
  st_as_sf(coords = c("Longitude", "Latitude"), crs = 4326, remove = FALSE) |>
  st_transform(21036)
schools <- places |> filter(Category == "School")
# --- Create grid & count points ---
grid <- st_make_grid(wards_gwasssi, cellsize = 500, square = TRUE) |> st_sf()
grid$school_count <- lengths(st_intersects(grid, schools))
# Convert grid to raster
grid_rast <- st_rasterize(grid["school_count"], dx = 500, dy = 500)
tmap_mode("plot")

tm_shape(wards_gwasssi) +
  tm_polygons(
    col = "grey70", fill = "grey95",
    fill.legend = tm_legend(legend.outside = TRUE)
  ) +
  tm_text(
    "IEBC_WARDS",
    size = 0.7,
    root = 4,
    remove.overlap = TRUE,
    options = opt_tm_text(shadow = TRUE)
  ) +
  tm_shape(grid_rast) +
  tm_raster(
    col.scale = tm_scale_continuous(values = "brewer.blues"),
    col.alpha = 0.6,
    legend.show = TRUE
  ) +
  tm_shape(schools) +
  tm_symbols(size = 0.05, fill = "red") +
  tm_title("Geospatial Heatmap") +
  tm_layout(legend.outside = TRUE)

```

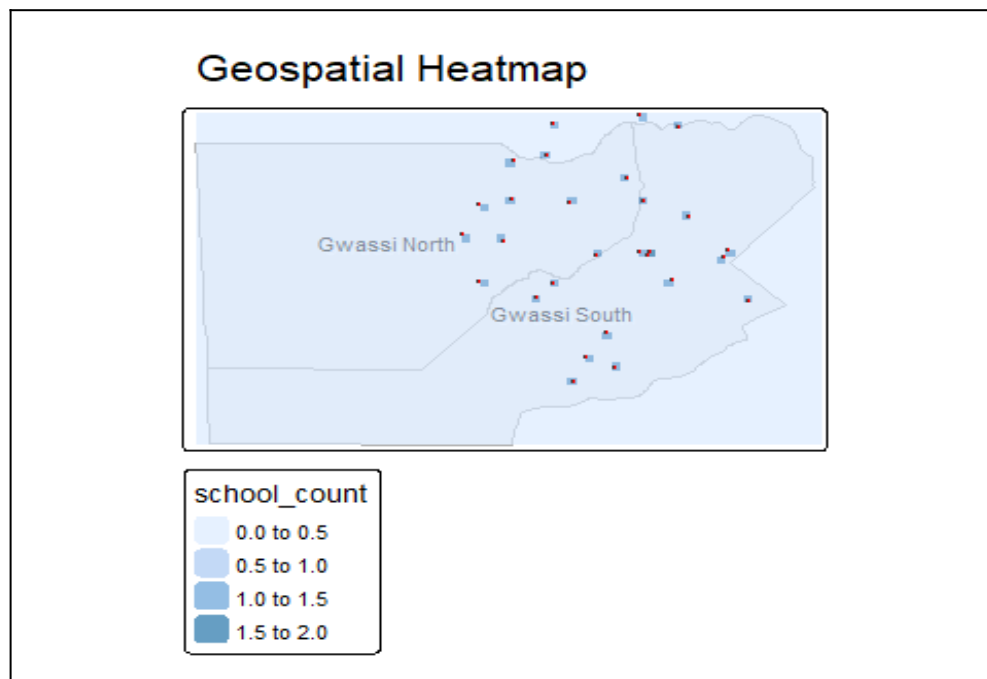


Figure 10. 10: Geospatial HeatMap

When you look at the final map, you'll see the study area covered by a grid that counts how many points, like schools, fall into each cell. These counts get turned into a colorful heatmap where darker or warmer colors show areas with lots of schools. You'll also see the ward boundaries outlined and labeled, helping you connect the heat patterns to real locations. On top of that, the actual school points appear as red dots, so you can compare the heatmap with exact school locations. All told, this map gives a clear picture of where schools cluster in the Gwassi wards, making it easy to spot which areas have many schools and which might need more.

10.11 Two Dimensional Geospatial Heatmap

Heatmaps in a spatial context depict the intensity of points across a continuous surface, allowing the eye to detect hotspots without rigid boundaries. This form of visualization is particularly suited for mixed categories of points, such as schools and churches, where both location and density matter.

The code plots `Gwassi South`'s polygon and applies `stat_density_2d()` to the subset of points belonging to the categories of interest. The `aes()` call specifies `fill = after_stat(level)` to colour polygons according to density levels, while `geom_point()` adds the raw locations for reference. The `scale_fill_viridis_c()` ensures perceptually uniform colour transitions.

```
ggplot() +  
  geom_sf(data = gwassi_south, fill = "white", color = "gray30") +  
  stat_density_2d(data = suba %>% filter(Category %in% c("School", "Church")),  
    aes(x = Longitude, y = Latitude, fill = after_stat(level)), geom =  
"polygon", alpha = 0.5) +  
  geom_point(data = suba %>% filter(Category %in% c("School", "Church")),  
    aes(x = Longitude, y = Latitude, color = Category), size = 2) +  
  labs(title = "2D Geospatial Heatmap") +  
  scale_fill_viridis_c() +  
  coord_sf(crs = 4326) +  
  Mytheme
```

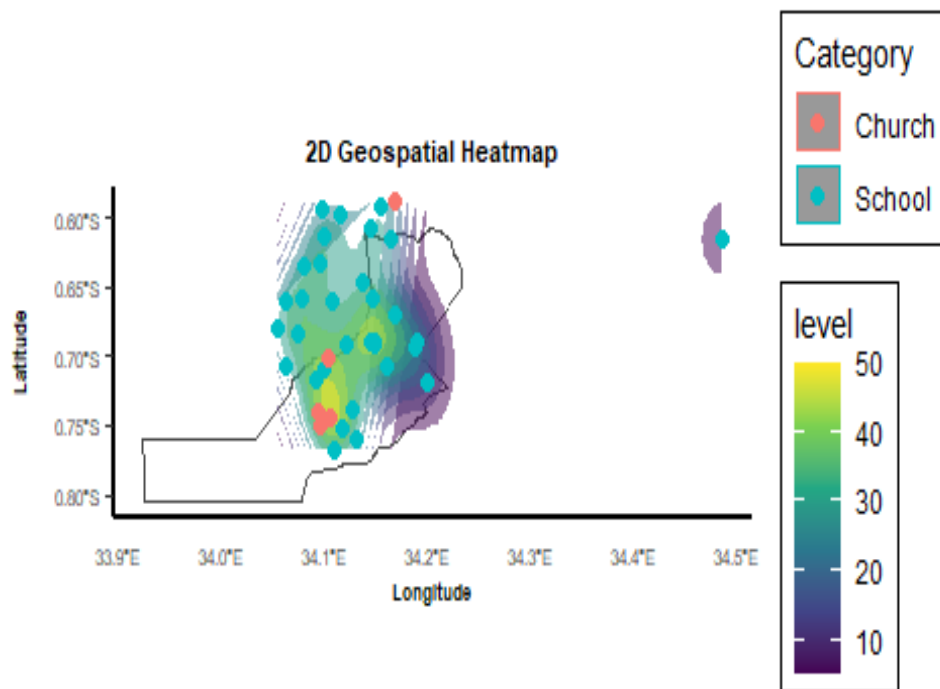


Figure 10. 11: 2D Geospatial HeatMap

The output is a smooth surface of colour intensity overlaying the ward, with darker areas signaling higher concentrations of facilities. By also showing the underlying points, the map offers both the generalized density pattern and the precise point data, satisfying the needs of both high-level planning and on-the-ground operational decision-making.

10.12 Road Lines

Road lines on a map aren't just decorative but they're one of the quickest ways to tell the story of movement, connection, and accessibility in a region. In this example, the script loads the road network shapefile, transforms it to match the map's projection, and then filters it down to just **trunk** and **unclassified** types. That way, the map doesn't get cluttered with every minor path, and the focus stays on roads that are either vital arteries or notable connectors.

The `tm_lines()` function triggers the heavy lifting for styling. It takes the road type (`col = "highway"`) and applies distinct colors, blue for **trunk** roads, orange for **unclassified**, through `tm_scale()`. At the same time, the width mapping (`lwd = "highway_width"`) makes major roads thicker (4.5) and minor ones slimmer (2.5), so the hierarchy is obvious even at a glance. Road names are added with `tm_text()`, giving the map an extra layer of usability. `tm_layout()` is used to polish the presentation. By setting `legend.outside = TRUE` and positioning it to the

bottom, the legend becomes easy to read without covering the map's details. Together, these code choices don't just produce a picture but create a purposeful, readable, and engaging map where every visual cue reinforces the data's meaning.

```
library(sf)
library(tmap)
library(tidyverse)
# static plotting for RMarkdown / non-interactive
tmap_mode("plot")
# Helper: only set CRS if missing
set_crs_if_missing <- function(x, epsg = 4326) {
  if (is.na(st_crs(x))) st_set_crs(x, epsg) else x
}
# Hex colors to avoid ambiguity
color_map <- c(trunk = "#1f78b4", unclassified = "#ff7f00")
wards_path <- "D:/ALL/Project/Shape files/kenya_wards/Kenya wards.shp"
wards <- st_read(wards_path, quiet = TRUE)
wards <- set_crs_if_missing(wards, 4326)
# Filter Gwasssi South (case-insensitive, exact substring)
wards_gwasssi <- wards %>%
  filter(str_detect(IEBC_WARDS, fixed("Gwasssi South", ignore_case = TRUE)))
if (nrow(wards_gwasssi) == 0) stop("No 'Gwasssi South' found in IEBC_WARDS.
Check spelling/case.")
# Project for metric widths
wards_gwasssi <- st_transform(wards_gwasssi, 21036)
roads_path <- "D:/ALL/Project/Shape
files/hotosm_ken_roads_lines_shp/hotosm_ken_roads_lines_shp.shp"
roads <- st_read(roads_path, quiet = TRUE)
roads <- set_crs_if_missing(roads, 4326)
roads <- st_transform(roads, 21036)
# Normalize highway text and keep only desired types; drop unnamed roads
roads <- roads %>%
  mutate(highway = tolower(highway)) %>%
  filter(highway %in% c("trunk", "unclassified"), !is.na(name))

# Ensure factor order predictable
roads <- roads %>% mutate(highway = factor(highway, levels = c("trunk",
"unclassified")))
# Numeric width column (required for lwd mapping)
roads <- roads %>% mutate(highway_width = if_else(highway == "trunk", 4.5,
2.5))
# Spatial subset: keep only roads that intersect Gwasssi South
roads_gwasssi <- st_filter(roads, wards_gwasssi, .predicate = st_intersects)
if (nrow(roads_gwasssi) == 0) warning("No roads found inside Gwasssi South
after filtering.")
# Places (safe encoding) -----
places_path <- "D:/ALL/Project/Shape files/Suba.csv"
# read with latin1 (common for Windows CSVs), then normalize chars to UTF-8
places_raw <- readr::read_csv(places_path, locale = locale(encoding =
"Latin1"))
# replace common "smart" quotes with ASCII equivalents, to ensure UTF-8
places_raw <- places_raw %>%
  mutate(across(where(is.character),
    ~ .x %>%
      str_replace_all("[\\u2018\\u2019\\u201A\\u201B\\u2032\\u2035]",
```

```

'''') %>% # single quotes
      str_replace_all("[\u201C\u201D\u201E\u201F\u2033\u2036]",
"\'") %>% # double quotes
      iconv(from = "latin1", to = "UTF-8", sub = "")
# force UTF-8, drop invalid
))
# Convert to sf; with matching Lon/Lat column names in CSV
places <- places_raw %>%
  st_as_sf(coords = c("Longitude", "Latitude"), crs = 4326, remove = FALSE)
%>%
  st_transform(21036)
# Keep only places inside Gwasssi South for clarity
places_gwasssi <- st_filter(places, wards_gwasssi, .predicate = st_intersects)
message("Places total:", nrow(places), " | Places in Gwasssi South:",
nrow(places_gwasssi))
# Building the map -----
map <- tm_shape(wards_gwasssi) +
  tm_borders(lwd = 1.2, col = "black") +
  # roads: categorical color via col.scale (hex colors), numeric lwd from
  highway_width
  tm_shape(roads_gwasssi) +
  tm_lines(
    col = "highway", # factor: trunk / unclassified
    lwd = "highway_width", # numeric widths (no scaling warnings)
    col.scale = tm_scale(values = color_map),
    col.legend = tm_legend(title = "Road Type")
  ) +
  # road name labels (tm_text); remove.overlap to reduce clutter
  tm_text("name", size = 0.2, col = "black", remove.overlap = TRUE) +
  # places: small symbol + label (Name). safe label column (Name)
  tm_shape(places_gwasssi) +
  tm_symbols(size = 0.15, col = "darkgreen") +
  tm_text("Name", size = 0.4, col = "darkred", fontface = "bold",
remove.overlap = TRUE) +
  # explicit legend showing color and width (clean key)
  tm_add_legend(
    type = "line",
    labels = c("Trunk", "Unclassified"),
    col = c(color_map["trunk"], color_map["unclassified"]),
    lwd = c(4.5, 2.5),
    title = "Road Type"
  ) +
  tm_title(text = "Road Network in Gwasssi South", position = c("center",
"top")) +
  tm_layout(legend.outside = TRUE, legend.outside.position = "bottom")
# Render the map
map

```

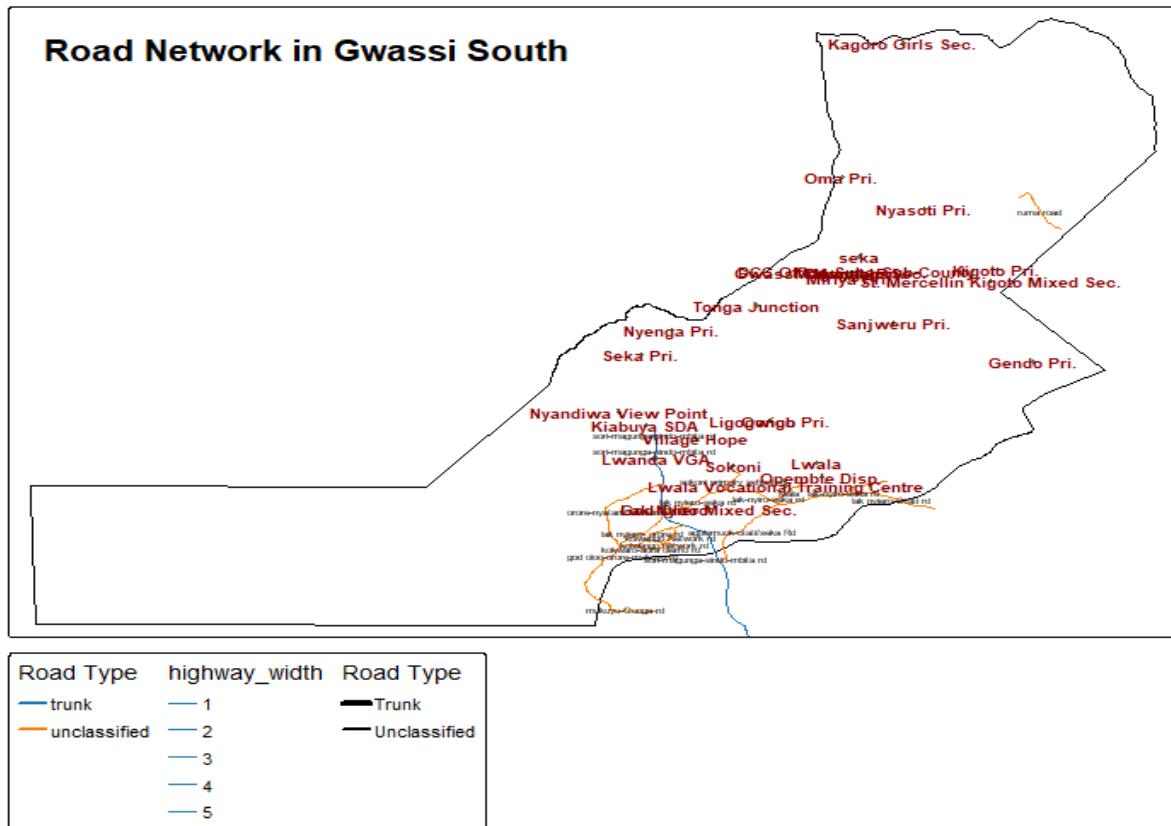


Figure 10. 12: Road Lines

If you look at the chart above, you realize that Gwasssi South road network is brought to life with clarity and nuance. Thick, blue lines mark the trunk roads, clearly signalling their role as major routes, while thinner orange lines trace the unclassified roads, highlighting the more local or informal pathways. These visual distinctions make it easy to see at a glance which roads carry heavier traffic or serve as crucial connectors, and which link smaller settlements or serve agricultural pockets.

10.13 Land Cover Map

In mapping natural features, you may not need every group represented in the map, for instance every vegetation type the dataset offers. Sometimes focusing on just a few makes the story clearer. In situations where you require such cleaning, the **LABEL** field can be refined to keep only what matters. For example, you might be interested in illustrating water bodies and edaphic wooded grassland features found on drainage-impeded or seasonally flooded soils.

Once you have the vegetation types of interest, you use `st_transform()` to place the shapes in the right coordinate system so they align naturally with the rest of your spatial layers. Without this adjustment, features can appear offset or distorted when combined with other data.

When you want to portray the vegetation, you call `tm_shape()` to set the filtered data as the layer to be drawn. You then use `tm_polygons()` to turn those shapes into filled areas on the map, with colors reflecting the vegetation categories. Together these functions give you a map that is both geographically precise and thematically focused, which is ideal when you want to highlight specific features rather than the whole collection.

```
library(sf)
library(tmap)
library(tidyverse)
library(writexl)
tmap_mode("plot")
## i tmap mode set to "plot".
# Helper function: set CRS if missing
set_crs_if_missing <- function(x, epsg = 4326) {
  if (is.na(st_crs(x))) st_set_crs(x, epsg) else x
}
wards_path <- "D:/ALL/Project/Shape files/kenya_wards/Kenya wards.shp"
wards <- st_read(wards_path, quiet = TRUE) |> set_crs_if_missing(4326)
message("Ward name samples:"); print(head(wards$IEBC_WARDS, 8))
wards_gwasssi <- wards |> filter(str_detect(IEBC_WARDS, fixed("Gwasssi South",
ignore_case = TRUE)))
if (nrow(wards_gwasssi) == 0) stop("No 'Gwasssi South' found. Check spelling.")
wards_gwasssi <- st_transform(wards_gwasssi, 21036)
veg_path <- "D:/ALL/Project/Shape
files/pnv_vecea_v2_0_Kenya_vector/pnv_vecea_v2_0_Kenya_vector/pnv_vecea_v2_0_Kenya.shx
"
veg <- st_read(veg_path, quiet = TRUE) |>
  set_crs_if_missing(4326) |>
  st_transform(21036) |>
  mutate(LABEL_clean = trimws(LABEL)) |> # remove stray spaces
  filter(
    LABEL_clean %in% c(
      "Water bodies",
      "Edaphic wooded grassland on drainage-impeded or seasonally flooded soils"
    ),
    !is.na(LABEL_clean)
  ) |>
  mutate(
    LABEL = factor(
      LABEL_clean,
      levels = c(
        "Water bodies",
        "Edaphic wooded grassland on drainage-impeded or seasonally flooded soils"
      )
    )
  )
veg_gwasssi <- st_filter(veg, wards_gwasssi, .predicate = st_intersects)
places_path <- "D:/ALL/Project/Shape files/Suba.csv"
places_raw <- readr::read_csv(places_path, locale = locale(encoding = "Latin1"))
```

```

places_raw <- places_raw |>
  mutate(across(where(is.character),
    ~ .x |>
      str_replace_all("[\\u2018\\u2019\\u201A\\u201B\\u2032\\u2035]", "'") |>
      str_replace_all("[\\u201C\\u201D\\u201E\\u201F\\u2033\\u2036]", "\"") |>
      iconv(from = "latin1", to = "UTF-8", sub = ""))
  ))
places <- places_raw |>
  st_as_sf(coords = c("Longitude", "Latitude"), crs = 4326, remove = FALSE) |>
  st_transform(21036)
places_gwasssi <- st_filter(places, wards_gwasssi, .predicate = st_intersects)
message("Places total:", nrow(places), " | Gwasssi South:", nrow(places_gwasssi))
map1 <-
  tm_shape(wards_gwasssi) +
  tm_borders(lwd = 1.2, col = "black") +
  tm_shape(veg_gwasssi) +
  tm_polygons(
    fill = "LABEL",
    fill.scale = tm_scale(values = "brewer.set3"),
    col = "gray40",
    lwd = 0.3,
    fill.legend = tm_legend(title = "Vegetation Type")
  ) +
  tm_shape(places_gwasssi) +
  tm_symbols(
    fill = "red",      # fill color inside symbol
    size = 0.1,
    shape = 21,
    lwd = 0.3,         # outline thickness
    col = "black"      # outline color
  ) +
  tm_text(
    "Name",
    size = 0.35,
    col = "black",
    fontface = "bold",
    remove.overlap = TRUE,
    options = opt_tm_text(just = "left", shadow = TRUE)
  ) +
  # Make room ABOVE the plot for the title
  tm_layout(
    legend.outside = TRUE,
    frame = FALSE,
    outer.margins = c(0.25, 0.02, 0.02, 0.02) # top margin bigger
  ) +
  # Place the title in the top margin space
  tm_title(
    text = "Gwasssi South Land Cover",
    position = c("left", "top"),
    just = c("center", "top"),
    size = 0.9, fontface = "bold"
  )

```

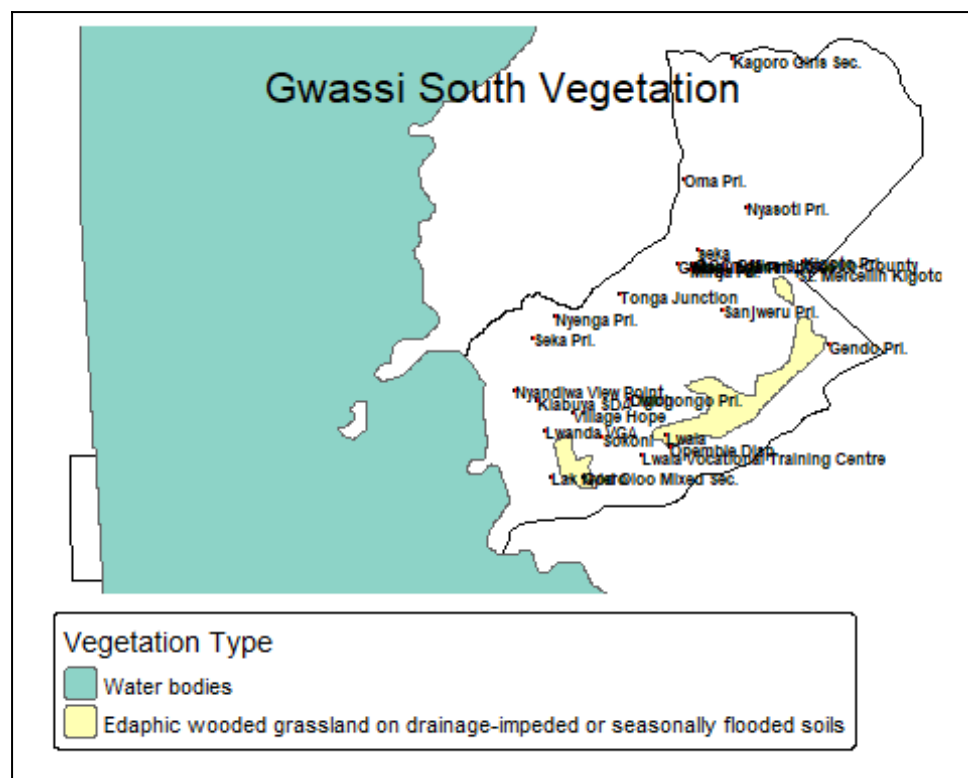


Figure 10. 13: Land Cover Map

CHAPTER 11: FLOW

The movement or flow of activities is crucial in many organizational setups, as it enhances productivity and fosters better working relationships among teams and departments. Understanding how tasks, information, or resources move within a system helps in identifying inefficiencies, streamlining operations, and improving coordination. To visually represent these flows, various graphics are employed, including Sankey diagrams, alluvial diagrams, flowcharts, and network graphs. Sankey diagrams are ideal for illustrating the proportional flow of resources or energy. Alluvial diagrams show how categories evolve or shift over time. Flowcharts map out step-by-step processes, making it easier to follow procedures and pinpoint decision points. Network graphs depict relationships and interactions between different entities or nodes. These visual tools are invaluable in communicating complex workflows, analyzing system dynamics, and supporting effective organizational planning.

11.1 Sankey Diagram

A Sankey diagram lets you follow how quantities move from one stage to the next, with the width of each ribbon proportional to the amount flowing through it. You read it as you would a system of pipes: wide streams carry more, narrow streams carry less. The appeal for analysis is that you don't just see where observations end up; you can trace their routes, compare competing pathways, and judge relative magnitudes at a glance.

You begin by loading `ggAlluvial`, which provides the geoms that draw both the nodes (strata) and the connecting ribbons (`alluvia`). From the built-in `trees` data you create two girth representations with `mutate():GirthVal` is a rounded, character label for exact measurements (useful for mid-axis labels), while `GirthLevel` bins the continuous `Girth` into three ordered classes, `Small`, `Medium`, `Large`, via `cut()`. This gives you both a fine-grained value and a coarser categorical endpoint.

To build the flows, `count()` aggregates frequencies for each pathway from a single source to an exact value and then to a level: `stage1 = All Tress` defines one left-hand source stratum; `stage2 = GirthVal` enumerates the exact rounded girths; `stage3 = GirthLevel` captures the destination class. The resulting column `n` is the volume that will set ribbon thickness. The brief `group_by(... ) |> summarise(n = first(n))` pass simply guarantees one record per triplet; it's

effectively a no-op here because `count()` already returns unique combinations. Renaming to `axis1`, `axis2`, and `axis3` aligns your columns with `ggAlluvial`'s expected aesthetics.

In the plot call, `aes(axis1 = axis1, axis2 = axis2, axis3 = axis3, y = n)` declares a three-axis flow where each row describes a single stream of size `n`. `geom_alluvium(aes(fill = axis3), width = 1/12, alpha = 0.7)` draws the ribbons and colors them by their destination level, with a touch of transparency to keep overlapping paths legible. `geom_stratum(width = 1/12, fill = "black", color = "white")` adds the stage blocks, your anchors on each axis—while `geom_text(stat = "stratum", aes(label = after_stat(stratum)))` prints the stratum names directly on those blocks so readers don't hunt for annotations. `scale_x_discrete(limits = c("Tree", "Girth", "Girth Level"), expand = c(.05, .05))` fixes the axis order and trims outer padding so the plot hugs the data. In short, your code constructs a single-source Sankey where all 31 trees fan out to exact girth values and then condense into Small, Medium, and Large, with ribbon width driven by the `count()`ed frequencies, precisely the kind of pathway view that Sankey diagrams are designed to reveal.

```
library(ggalluvial)
mytrees <- trees %>%
  mutate(GirthVal = as.character(round(Girth, 1)),
         GirthLevel = cut(Girth, breaks = 3, labels = c("Small", "Medium", "Large")))
# Aggregate for alluvial format
flow_data <- mytrees %>%
  count(stage1 = "All Trees", stage2 = GirthVal, stage3 = GirthLevel)
# Convert to alluvial format
flow_long <- flow_data
flow_long%>%
  group_by(stage1, stage2, stage3)%>%
  summarise(n = first(n), .groups = "drop")
alluvial_data <- flow_data %>%
  rename(axis1 = stage1, axis2 = stage2, axis3 = stage3)
# Plot using ggalluvial
ggplot(alluvial_data,
       aes(axis1 = axis1, axis2 = axis2, axis3 = axis3, y = n)) +
  geom_alluvium(aes(fill = axis3), width = 1/12, alpha = 0.7) +
  geom_stratum(width = 1/12, fill = "black", color = "white") +
  geom_text(stat = "stratum", aes(label = after_stat(stratum)),
          color = "white", size = 3) +
  scale_x_discrete(limits = c("Tree", "Girth", "Girth Level"), expand = c(.05, .05)) +
  labs(title = "Sankey Chart",
       y = "Number of Trees") +
  theme_minimal(base_size = 12) +
  theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1))
```

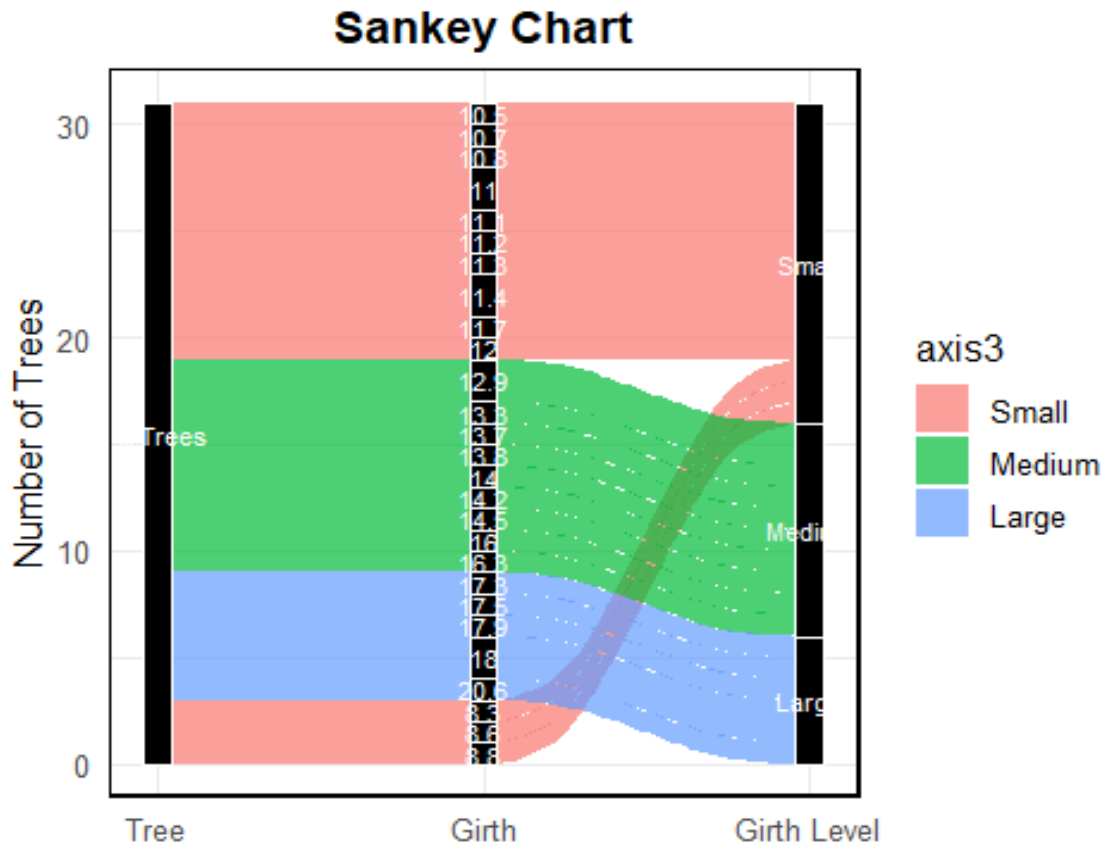


Figure 11. 1: Sankey Diagram

In the Sankey chart, we see a layered story of how trees move from exact girth values into the broader **Small**, **Medium**, and **Large** categories. Each black block, or stratum, anchors a stage, while the flows (alluvia) connecting them reveal the number of trees transitioning between stages.

What really stands out are the thicker flows, which correspond to higher frequencies. For example, the wide band carrying 6 trees from girth 16.5 into the **Large** category shows that this is not only a common measurement but also a strong contributor to the final Large group. By contrast, the slimmer flows, often marked with frequencies of 1 or 2, signal rarer transitions, such as less typical girth sizes feeding into Small or Medium categories. Following the flows across the chart makes it clear that the **Large** group dominates, receiving the widest streams of trees. Medium comes next with moderately thick flows, while Small collects the thinnest strands, underscoring its minority share. The beauty of this design is that the flows do the storytelling. For instance, they quantify and visualize how raw measurements funnel into summarized categories. Instead of just seeing final totals, we can trace the exact pathways, understanding not

only where trees end up but also how frequently each unique girth measurement contributes to the bigger picture.

11.2 Alluvial Diagram

When you explore an alluvial chart, you watch data stream from one group to another, almost like following the current of a river, where streams split, merge, or change direction as categories shift across stages. For researchers, it's a powerful way to follow how observations move from one grouping to another, such as measurements falling into size ranges or populations shifting across classifications. Unlike a bar chart that just shows totals, this design emphasizes movement and connections, letting you see structure and distribution as a dynamic process rather than static counts.

To prepare the trees dataset, `mutate()` enriches the data with two new variables. `GirthVal` rounds the continuous girth values into labels that can be displayed cleanly, while `GirthLevel` uses `cut()` to divide girth into three broad categories, `Small`, `Medium`, and `Large`. A helper column called `Tree` is added to act as a consistent anchor across stages. The reshaping happens with `pivot_longer()`, which stacks the variables into a long tidy format where each one becomes a stage. To keep every record distinct, `group_by()` together with `row_number()` assigns unique identifiers, and `pivot_wider()` brings the structure back into a wide form suited for plotting.

With the structure set, `count()` tallies up the frequencies, which later define the thickness of the flows. The flowing ribbons appear through `geom_alluvium()`, colored by girth category and slightly transparent for readability. The rectangular blocks for each category come from `geom_stratum()`, and labels are added directly with `geom_text(stat = "stratum")`. Another `geom_text()` overlays the frequency counts on the flows themselves, so the chart doesn't just show structure but also quantity.

```
library(ggalluvial)
mytrees <- trees %>%
  mutate(GirthVal = as.character(round(Girth, 1)),
         GirthLevel = cut(Girth, breaks = 3,
                        labels = c("Small", "Medium", "Large")))
alluvial_long <- mytrees %>%
  mutate(Tree = "Tree")
alluvial_long %>%
  select(Tree, GirthVal, GirthLevel) %>%
  pivot_longer(cols = everything(), names_to = "stage", values_to = "value") %>%
  group_by(stage, value) %>%
```

```

mutate(obs = row_number()) %>%
ungroup() %>%
pivot_wider(names_from = stage, values_from = value)
# Count for y-axis
alluvial_long <- alluvial_long %>%
  count(Tree, GirthVal, GirthLevel, name = "Freq")
alluvial_long
ggplot(alluvial_long,
  aes(axis1 = Tree,
      axis2 = GirthVal,
      axis3 = GirthLevel,
      y = Freq)) +
  geom_alluvium(aes(fill = GirthLevel),
    width = 1/12,
    alpha = 0.7) +
  geom_stratum(width = 1/12,
    fill = "black",
    color = "white") +
  geom_text(stat = "stratum",
    aes(label = after_stat(stratum)),
    color = "white", size = 3) +
  geom_text(aes(label = Freq,
    stat = "alluvium",
    size = 3, color = "black") +
  scale_x_discrete(limits = c("Tree",
    "GirthVal",
    "GirthLevel"),
    expand = c(.05, .05)) +
  labs(title = "Alluvial Chart", y = "Frequency") +
  theme_minimal() +
  theme(plot.title = element_text(size = 14,
    face = "bold",
    hjust = 0.5),
    panel.border = element_rect(color = "black",
    fill = NA,
    linewidth = 1))

```

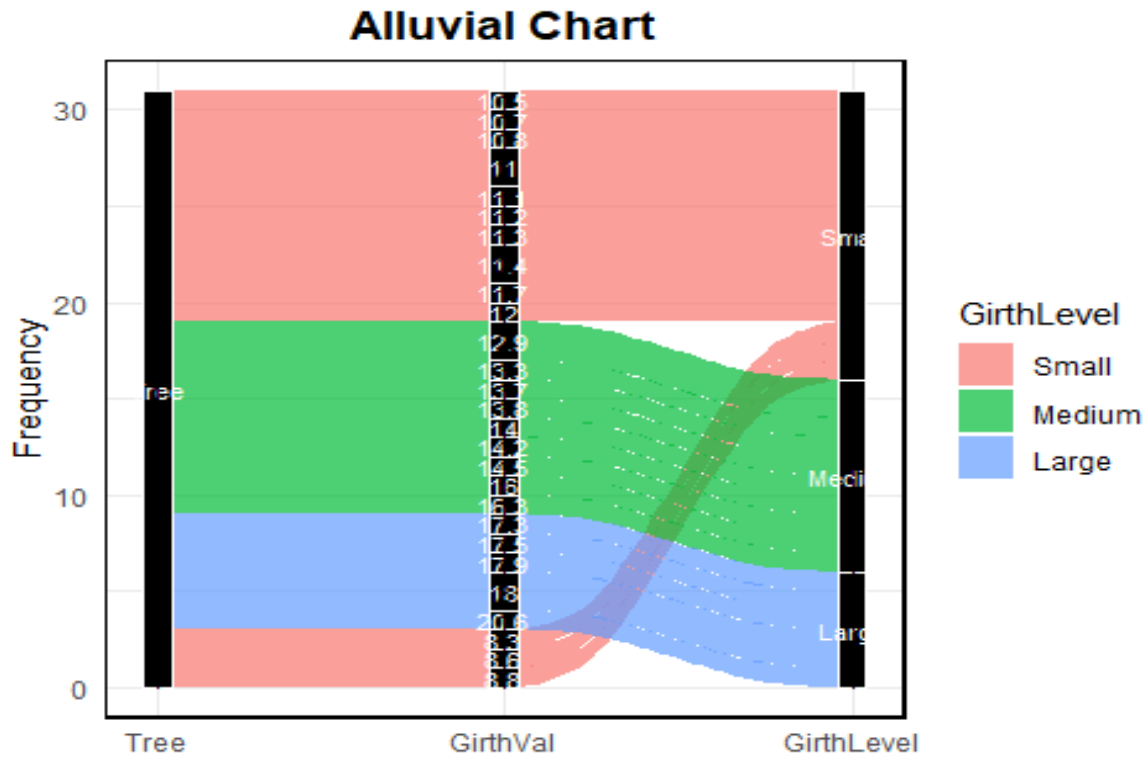


Figure 11. 2: Alluvial Diagram

In the chart, each stage is laid out as black blocks known as strata, while the streams running between them, called alluvia, reveal how many trees move from one stage to the next. As you follow the diagram, you notice that the girth values feed directly into the grouped categories, with the thickness of each flow showing the number of trees. The frequency counts, written in black along the flows, make the story even clearer. For example, a band linking a girth measurement of about 16.5 to the **Large** category is labeled “6,” which tells you that six trees with that girth end up in the Large group. Thinner strands with labels like 1 or 2 highlight the rarer girth sizes making their way into the categories.

By the final stage, the **Large** block stands out as the dominant group, holding the majority of the trees, followed by **Medium**, and then **Small**. This layout gives you an immediate sense of how precise girth measurements funnel into broader categories, while the frequency labels on each flow provide the exact counts to support that impression. Altogether, the diagram combines structure and detail in a way that lets you trace the full distribution of tree girths, both at the fine-grained measurement level and in their summarized categories, without losing track of how individual observations shift across stages.

11.3 Flowchart

Suppose you need to make sense of a complex process, understand the sequence of actions, or identify where decisions and outcomes branch off, you want a tool that lets you see the entire workflow at a glance. A flow chart serves this purpose perfectly. It is a diagram that visually maps each step of a process using standardized symbols: rectangles for actions, diamonds for decisions, and arrows to indicate the flow of operations.

The strength of a flow chart lies in its ability to simplify complexity. By laying out tasks, decision points, and pathways in a clear visual format, it helps you spot inefficiencies, overlaps, or potential bottlenecks. You can quickly trace how a process moves from start to finish, compare alternative paths, and communicate workflows to others without needing to sift through dense textual descriptions. This makes it invaluable in programming, business planning, project management, and educational contexts where clarity and structure are essential.

This flow chart is built entirely in `ggplot2` by layering annotations and segments to simulate a process diagram. The `annotate("rect", ...)` calls create the boxes representing the main nodes in the chart: one for `Gender`, another for `Media`, and a third for the `Intervening Variable`. Each rectangle is placed precisely by specifying `xmin`, `xmax`, `ymin`, and `ymax`, and is given a light blue fill with black borders to make it visually distinct.

Node labels are added using `annotate("rect", ...)`. Here, `x` and `y` position the text inside or above the boxes, while arguments like `size`, `face="bold"`, and `hjust` control readability and alignment. For example, the gender box displays `Male` and `Female` as a bulleted list, the intervening variable box lists `Knowledge`, `Culture`, and `Technology`, and the media box simply shows `Media`.

The connections between nodes are drawn with `geom_segment()`, which adds arrows from one point to another. Arrows are created using the `arrow=arrow(length=unit(...))` argument to indicate direction. For instance, a segment from the gender box to the media box shows influence, while a vertical segment from the intervening variable box to the media box indicates mediation.

For the detailed media platforms, text annotations list platforms like WhatsApp, Facebook, and TikTok along the right side of the chart. Each platform is connected to the media node using

additional `geom_segment()` calls, forming arrows that indicate how the media node flows into individual platforms. Arrow lengths and positions are carefully set to align endpoints with the text labels. `xlim()` and `ylim()` set the coordinate boundaries to ensure all elements fit nicely on the canvas, `theme_void()` removes background and axis lines to mimic a clean diagram.

```
# Prepare node text
gender_text <- "Gender\n• Male\n• Female"
intervene_text <- "Intervening Variable\n• Knowledge\n• Culture\n• Technology"
media_text <- "Media"
platforms <- data.frame(
  label = c("WhatsApp", "Facebook", "TikTok", "Twitter",
            "YouTube", "Messenger", "Instagram", "Telegram"),
  x = rep(10, 8),
  y = seq(9.75, 7.0, length.out = 8),
  image = c("images/whatsapp.png",
            "images/facebook.png",
            "images/tiktok.png",
            "images/twitter.png",
            "images/youtube.png",
            "images/messenger.png",
            "images/instagram.png",
            "images/telegram.png")
)

ggplot() +
  # Boxes
  annotate("rect",
    xmin=1,
    xmax=3,
    ymin=8,
    ymax=9.5,
    fill="lightblue",
    color="black") + # Gender box
  annotate("rect",
    xmin=7,
    xmax=9,
    ymin=8,
    ymax=9.5,
    fill="lightblue",
    color="black") + # Media box
  annotate("rect",
    xmin=3.5,
    xmax=6,
    ymin=5,
    ymax=6.5,
    fill="lightblue",
    color="black") + # Intervening Variable box

  # Node Labels
  annotate("text",
    x=2,
    y=8.75,
    label=gender_text,
    size=4,
    fontface="bold",
    hjust=0.5) +
  annotate("text",
    x=8,
    y=8.75,
```

```

        label=media_text,
        size=4, fontface="bold") +
annotate("text",
        x=5,
        y=6,
        label=intervene_text,
        size=4,
        fontface="bold", hjust=0.5) +

# Arrows
geom_segment(aes(x=3,
        y=9,
        xend=7,
        yend=9),
        arrow=arrow(length=unit(0.3, "cm")) + # Gender → Media
geom_segment(aes(x=5,
        y=6.5,
        xend=5,
        yend=9),
        arrow=arrow(length=unit(0.3, "cm")) + # Intervene → Media

# Media to platforms
annotate("text",
        x=10,
        y=9.5,
        label="WhatsApp",
        hjust=0) +
annotate("text",
        x=10,
        y=9,
        label="Facebook",
        hjust=0) +
annotate("text",
        x=10,
        y=8.5,
        label="TikTok",
        hjust=0) +
geom_segment(aes(x=9,
        y=9.25,
        xend=10,
        yend=9.5),
        arrow=arrow(length=unit(0.25, "cm")) +
geom_segment(aes(x=9, y=9,
        xend=10,
        yend=9),
        arrow=arrow(length=unit(0.25, "cm")) +
geom_segment(aes(x=9,
        y=8.75,
        xend=10,
        yend=8.5),
        arrow=arrow(length=unit(0.25, "cm")) +

xlim(0, 12) + ylim(4, 10) +
theme_void() +
ggtitle("Gender Influence on Media Usage") +
theme(plot.title = element_text(hjust=0.5,
        size=14, face="bold"))

```

Gender Influence on Media Usage

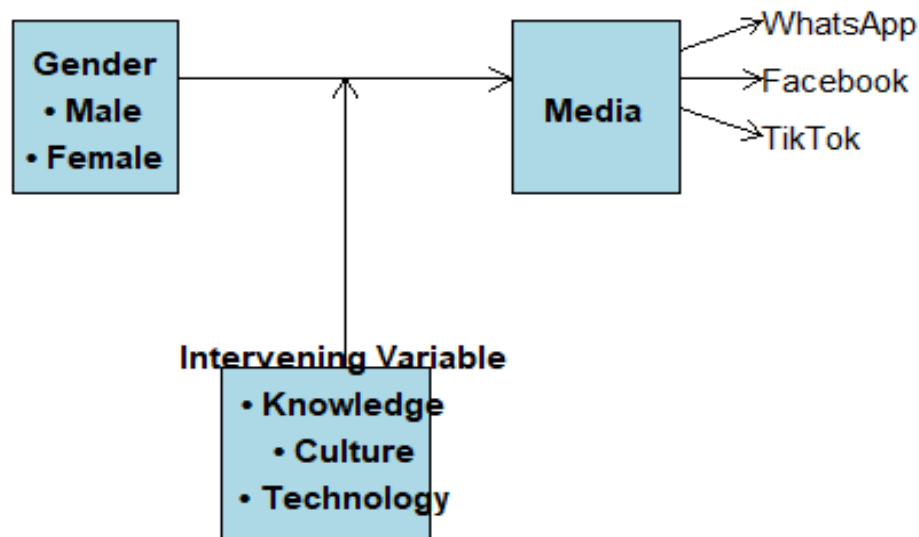


Figure 11. 3: Flow Chart

The leftmost blue box groups the key categories under **gender**, specifically **Male** and **Female**, symbolizing the starting point of influence. From there, an arrow points directly to the "Media" box on the right, suggesting a direct relationship where gender affects media usage. Additionally, the intervening variable box centered below, listing factors like Knowledge, Culture, and Technology, feeds upward into the media box, indicating these factors mediate or modify the gender-to-media influence.

On the far right, the individual media platforms such as WhatsApp, Facebook, and TikTok are listed vertically with arrows connecting from the media box, visualizing the dissemination of influence from gender and intervening variables down to specific platforms. This flow implies that gender and its mediated factors collectively impact how these different media platforms are used. The diagram uses spatial arrangement and directional arrows to clarify the pathway of influence, and the labels in the boxes provide categorical detail. The absence of quantitative metrics highlights this as a conceptual map useful for understanding relationships and flow, rather than measuring effect sizes or proportions.

11.4 Network Chart

A network chart is like a map of relationships. Instead of looking at individual variables in isolation, you trace how entities are connected and where the bridges lie. For researchers, it is especially powerful when you want to unearth structural patterns, clusters, or central nodes in your data. In essence, it helps you see the system as a whole rather than just the parts.

In the code, the `nodes` object creates the list of actors in the network: here, one node is `Gender`, and the others are different media platforms, some of which carry image paths to embed their icons. The `edges` object specifies the links, connecting `Gender` to every listed platform using `rep()` to repeat the source node across all connections. Then, `graph_from_data_frame()` builds the network structure by combining the edges with the nodes, while `create_layout()` organizes them spatially using the `Fruchterman-Reingold` algorithm, which spaces nodes out in a visually balanced way. After that, `layout$image` matches each node with its corresponding icon file.

The plotting step uses `ggraph()`, where `geom_edge_link()` draws the ties between nodes with arrows, and `geom_node_point()` adds the base pink nodes as anchor points. To make the visualization more engaging, `geom_image()` overlays logos on each node using their file paths, and `geom_node_text()` ensures node names are legible with repelled labels.

```
library(igraph)
library(ggraph)
library(ggimage)
library(dplyr)
# Platforms to include (same as earlier chart)
media_platforms <- c("WhatsApp", "Facebook", "Instagram", "TikTok",
                    "Twitter", "YouTube", "Messenger", "WeChat")
# Nodes with image paths
nodes <- data.frame(
  name = c("Gender", media_platforms),
  image = c(NA,
            "images/whatsapp.png", "images/facebook.png", "images/instagram.png",
            "images/tiktok.png", "images/twitter.png", "images/youtube.png",
            "images/messenger.png", "images/wechat.png"),
  stringsAsFactors = FALSE
)
# Edges: Gender to media platforms
edges <- data.frame(
  from = rep("Gender", length(media_platforms)),
  to = media_platforms,
  stringsAsFactors = FALSE
)
# Create graph and layout
net <- graph_from_data_frame(edges, vertices = nodes)
layout <- create_layout(net, layout = "fr")
# Merge icon paths into layout
```

```

layout$image <- nodes$image[match(layout$name, nodes$name)]
# Plot using ggraph
ggraph(layout) +
  geom_edge_link(arrow = arrow(length = unit(3, "mm")),
    end_cap = circle(4, 'mm'),
    edge_colour = "gray40") +
  geom_node_point(size = 8, color = "pink") +
  geom_image(aes(x = x, y = y, image = image), size = 0.07, na.rm = TRUE) +
  geom_node_text(aes(label = name), repel = TRUE, fontface = "bold", size = 3.5) +
  theme_void() +
  ggtitle("Media Usage Network Chart") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14),
    plot.background = element_rect(color = "black", linewidth = 1))

```

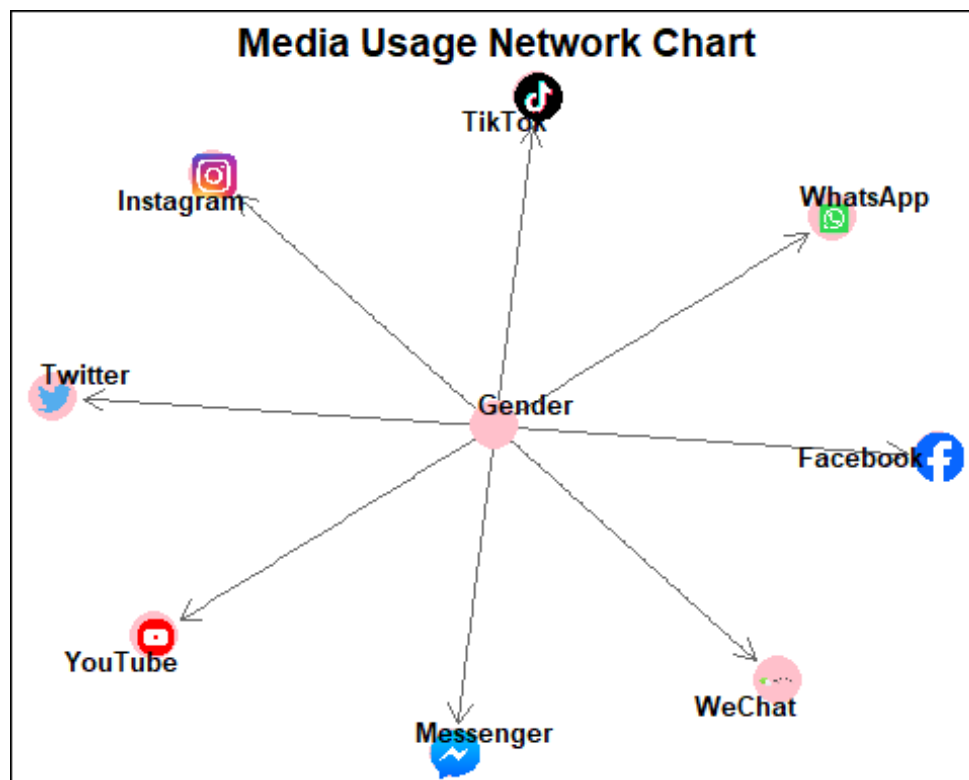


Figure 11. 4: Network Chart

The graph in Figure 11.4 noticeably displays the central role of **Gender** as the primary influencer, from which connections extend to various media platforms like WhatsApp, Facebook, Instagram, TikTok, Twitter, YouTube, Messenger, and WeChat. The visual layout emphasizes the directional flow from the central node to each platform, making it easy to follow how gender influences media usage.

The inclusion of platform icons on the nodes helps quickly identify each media outlet, enhancing immediate recognition and understanding. The arrows indicate the flow of influence, reinforcing

the idea of a one-way relationship from gender to platform choice. This visualization effectively conveys the distribution and reach of gender's influence across multiple platforms, highlighting the breadth of impact and allowing for a straightforward comparison of connections. It supports flow analysis by showing not just the existence of relationships but also their direction and scope, making it a valuable tool for understanding patterns of media engagement.

CHAPTER 12: CONCENTRATION AND DENSITY

The intensity of information is essential in identifying the most critical aspects or concentrations within a dataset. It helps highlight areas with high or low data density, enabling analysts to focus on regions of interest or concern. Visualizing intensity allows for quicker interpretation of patterns, anomalies, and hotspots within complex data. To effectively represent this, you need to use visual tools such as density plots, 2D histograms, contour plots, and heatmaps. A density plot shows the distribution of data points across a continuous variable, helping reveal peaks and clusters. 2D histograms display the frequency of data in two dimensions, giving a clearer sense of joint distributions. Contour plots use lines to indicate areas of equal intensity, often applied in geographical or topographical data. Heatmaps use color gradients to represent intensity, making them ideal for identifying trends, correlations, and outliers.

12.1 Density Chart

Density chart is a powerful tool for thinking about the distributional properties of numeric variables in an elegant and interpretable way. Unlike histograms, which can appear blocky and are sensitive to arbitrary bin widths, density plots produce smooth, continuous curves that better support statistical reasoning. A density curve is essentially a smoothed histogram normalized so that the total area under the curve equals one, thereby representing the probability distribution of the data. This approach makes it easier to compare distributions across groups, highlight peaks that indicate modality, and detect features such as skewness or long tails, offering a clearer and more flexible view of the underlying data structure.

The `geom_density()` function in `ggplot2` is a classic tool for visualizing the distribution of a single variable. Using the `trees` dataset, transformed to `mytreez` where the Girth of trees has been categorized into `Small` and `Wide`, we are able to plot the distribution of the `Volume` variable to show how values are concentrated along the range. `geom_density` produces a smooth kernel density estimate, showing us where the data is more densely packed. Using `fill = Girth`, we color the density areas based on the girth category, which helps compare the distributions. Transparency via `alpha` is applied to reduce over-plotting. This chart is excellent when we want to compare how different groups (like `Small` vs `Wide` trees) are distributed over a numeric variable. We also apply a classic minimal theme for clarity.

```
mytreez <- trees
mytreez <- mytreez %>%
```

```
mutate(Girth = case_when(Girth <= 15 ~ "Small",
                        Girth > 15 ~ "Wide"))
ggplot(mytreez, aes(x = Volume, fill = Girth)) +
  geom_density(alpha = 0.5) +
  labs(title = "Density Plot", x = "Volume", y = "Density") + Mytheme
```

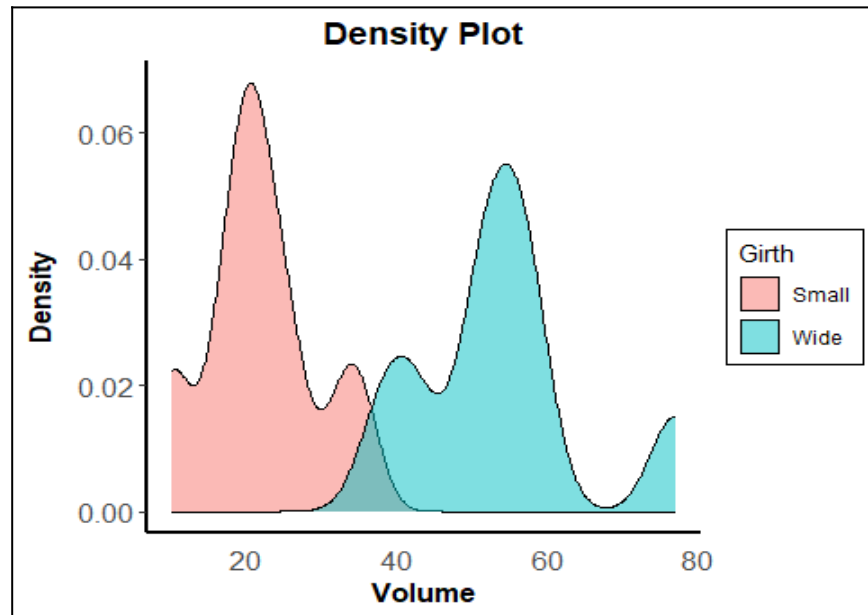


Figure 12. 1: Density Chart

The overlapping shaded areas show how the volumes are distributed for each group. The **Small** girth category exhibits a sharp, high peak around volumes close to 30, indicating a strong concentration of trees with volumes clustered tightly near that value. In contrast, the **Wide** girth group shows a broader and lower peak centered around a volume of approximately 70, suggesting a more spread-out distribution with less density at any one volume point.

Let's say you're curious to find out which fuel efficiency levels are most common among cars. Instead of using a plain histogram, you might prefer a smoother look at how values are distributed. That's where `geom_density()` comes in. This chart uses the `mpg` column from the `mtcars` dataset to show a continuous estimate of the distribution. You'll notice it forms a smooth curve, unlike the blocky appearance of histograms. The shape and height of the curve reveal where values tend to bunch up. If you're comparing car models, this chart gives a simple overview of where most data points lie.

```
ggplot(mtcars, aes(x = mpg)) +
  geom_density(fill = "skyblue", alpha = 0.5) +
  labs(title = "Density of Miles per Gallon") + Mytheme
```

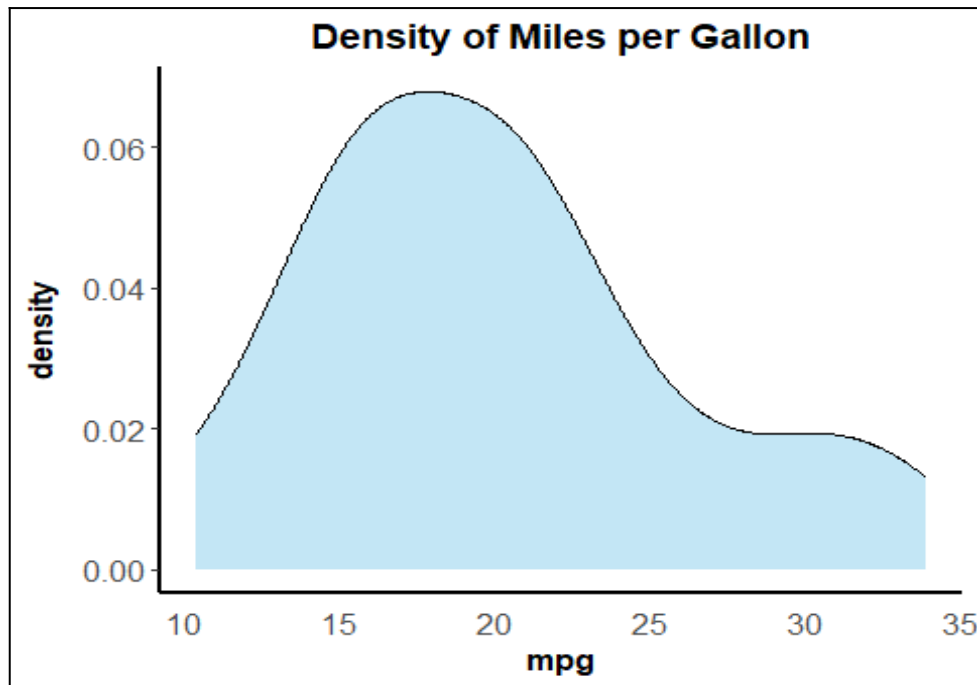


Figure 12. 2: Univariate Density Chart

The smooth curve, filled in sky blue with moderate transparency, reveals where mpg values are most concentrated. The peak around 15 to 20 mpg indicates that many cars cluster within this fuel efficiency range. The density then tapers off towards higher mpg values, suggesting fewer cars achieve very high fuel efficiency. This visualization helps identify the central tendency and spread of mpg, offering a clear picture of how fuel efficiency varies among the cars. That high point tells you the most common mpg value. You'll also see how values taper off towards the extremes. This helps you quickly spot whether most cars are fuel-efficient or not. Users often find this helpful when identifying clusters or comparing how values spread out. The density plot gives a gentle yet powerful way to visualize concentration.

12.2 Two Dimensional Density Chart

Imagine plotting two variables, like horsepower and miles per gallon, and wanting to know where most of your data points are clumped. A scatterplot alone might be too noisy. That's where `geom_density_2d()` steps in. It overlays contour lines on your scatterplot to represent levels of concentration. Think of it like looking at a terrain map where the hills are areas of high data density. This chart builds on the mpg and hp variables and shows where cars tend to group in those two dimensions.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  geom_density_2d() +
```

```
labs(title = "2D Density Chart")+ Mytheme
```

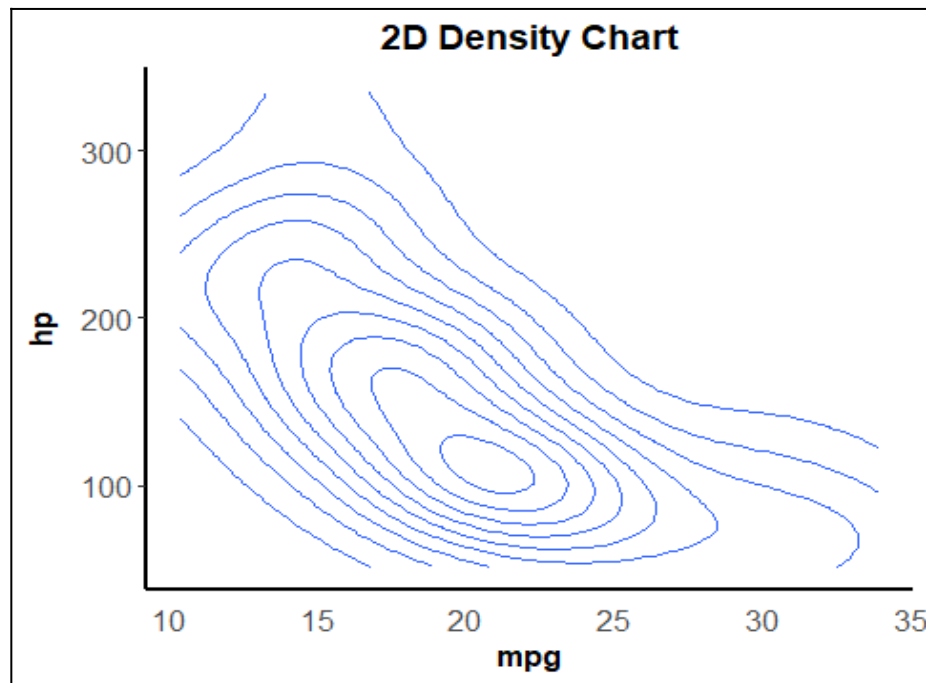


Figure 12. 3: 2D Density Chart

In the resulting chart, each set of nested lines outlines regions of similar data density. The tightest loops pinpoint areas where values cluster most. So, for instance, you may notice several cars with 20 mpg and around 110 horsepower. This kind of chart is particularly insightful when teaching students about bivariate concentration or where two measurements converge most often. It's a clean, spatial way of showing how two variables are jointly distributed.

12.3 Filled Two Dimensional Density Chart

We may wish to have the 2D contour lines to pop up a bit more in the chart and that's exactly what `geom_density_2d_filled()` allows us to do. Instead of just lines, this version adds layers of color between the contours. The fill intensity shows the relative concentration of values. Using the same variables, mpg and hp, the result is a more vibrant, informative map of the data. This chart draws attention to where the data is heavily concentrated without needing point-by-point detail.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  geom_density_2d_filled() +  
  labs(title = "Filled 2D Density Chart")+ Mytheme
```

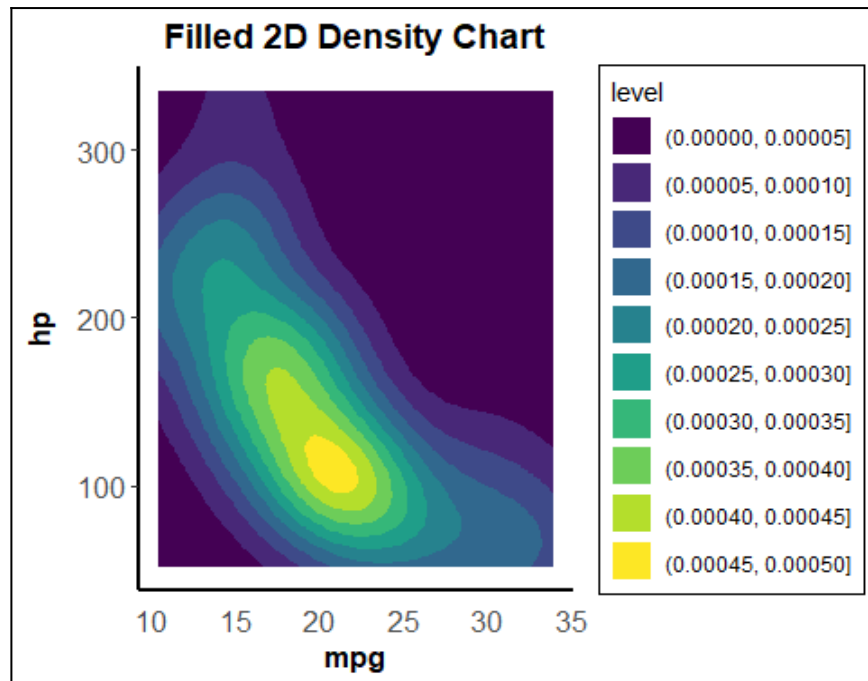


Figure 12. 4: Filled 2D Density Chart

In relation to concentration, you can tell a lot from this kind of chart at first glance. Instead of individual dots, the plot uses shading to show where most vehicles fall in terms of `mpg` and horsepower. Areas filled with deeper shades reveal stronger concentrations of cars with similar characteristics, while lighter tones signal places where fewer models overlap. You can see a clear grouping where `mpg` is roughly 15 to 25 and horsepower sits around 100 to 150, which is where many of the `mtcars` models live. You will also notice that when `mpg` climbs above about 25, horsepower tends to stay below roughly 120, so the cells in that upper-efficiency band are present but not heavily saturated. At the other extreme, below about 15 `mpg` and above 200 horsepower there are only a handful of cars, so the bins in that region appear much lighter. Taken together, these observations show how `mpg` and horsepower co-vary, with rich hues guiding your eye to the dominant cluster while paler shades mark the tails of the distribution.

12.4 Binned Density Chat

You can think of `geom_bin2d()` as a way of chopping up the plot area into little rectangular bins and then counting how many data points land in each box. Instead of showing individual points, it shades each box according to the number of observations inside it. With the `mtcars` dataset, this makes it easy to spot where car features bunch together, since the darker or more filled boxes

reveal regions of higher density. In practice, it turns a scatter of numbers into a grid-like heatmap, helping students quickly see clustering patterns across the two numeric variables.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  geom_bin2d() +  
  labs(title = "Binned Density Plot")+Mytheme
```

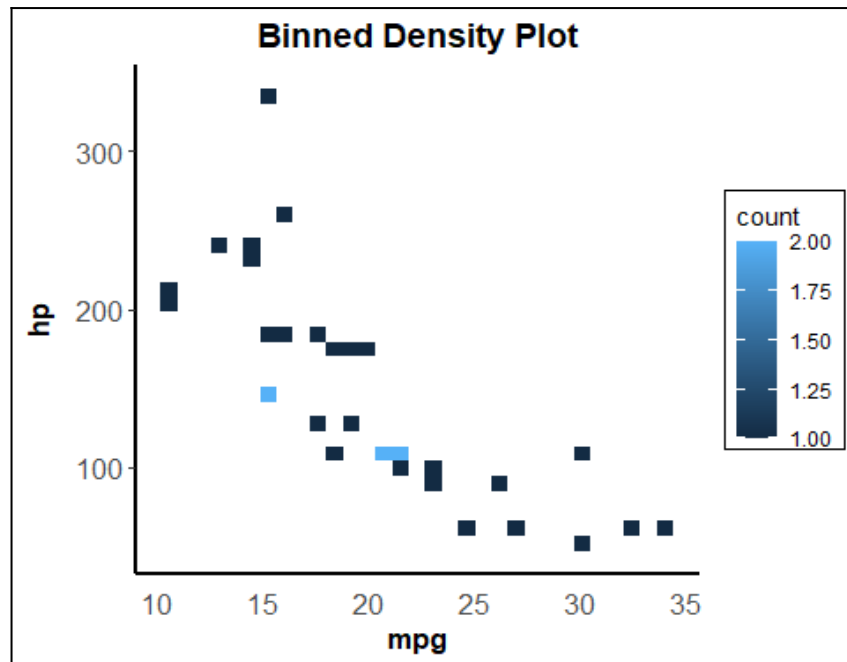


Figure 12. 5: Binned Density Chart

Starting at the x-axis for `mpg` and the y-axis for `hp`, you can trace how binned density chart divides the plane into small rectangular bins. Each bin's color intensity reflects the number of cars that fall within that specific mpg and horsepower range. In the `bin2d` representation, the intensity of shading encodes local data density, where darker cells denote regions of higher frequency, whereas lighter cells indicate sparser observations. A pronounced aggregation emerges where fuel efficiency lies between 15 and 20 miles per gallon, coupled with horsepower values spanning approximately 100 to 150. This concentration underscores a structural tendency in the dataset, revealing a modal cluster of vehicles that share intermediate performance characteristics. In the output, these bins appear like neatly stacked bricks, some glowing darker, others lighter, making it easy to compare density across ranges. This approach allows students to quickly spot crowded versus empty regions, adding an intuitive layer to understanding how two numeric variables group together in the dataset.

12.5 Hexbin Chart

Hexbin plot divides the data space into hexagonal cells, where the color intensity of each hexagon reflects the number of observations within that cell. Sometimes, you've got so many data points that a scatter plot turns into a blob. To solve that, `geom_hex()` replaces points with hexagonal tiles that summarize the data underneath. Each tile counts how many points land there, and the color intensity reflects that count. When applied to variables like `mpg` and `hp`, it quickly shows where values are most concentrated. This **hexbin** plot is especially helpful when you want a quick read on data density in crowded areas.

```
library(hexbin)
ggplot(mtcars, aes(x = mpg, y = hp)) +
  geom_hex() +
  labs(title = "Hexbin Plot") + theme
```

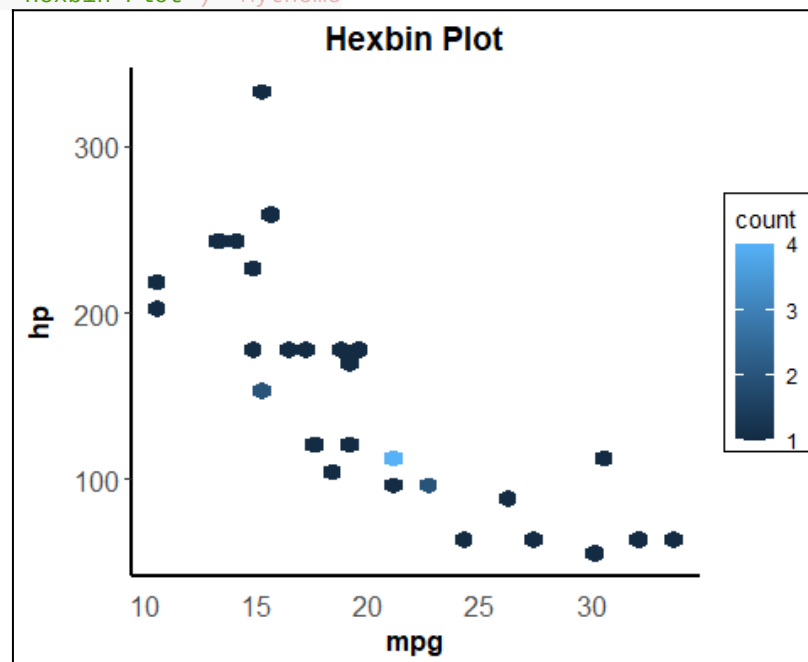


Figure 12. 6: Hexbin Density Chart

The plot you'll see uses honeycomb-like shapes to segment the space. Darker or more saturated hexagons indicate a higher density of cars with similar mpg and hp combinations. For instance, you may observe a strong concentration of vehicles in the region where `mpg` ranges from around 15 to 20 and horsepower is between 100 and 150. The hexagonal bins help reduce overplotting and provide a smoother representation of data density compared to simple scatterplots, making it easier to identify clusters and patterns in the relationship between `mpg` and horsepower. Hexbin charts are especially helpful for larger datasets, where overlap can obscure patterns.

12.6 Raster Plot

Think of a raster plot like laying a pixelated image over your scatterplot. Instead of drawing individual points, we use a grid where the color of each cell reflects how many data points landed there. It's a neat way to simplify dense data areas. In this example, we're still exploring the relationship between `mpg` (miles per gallon) and `hp` (horsepower) in `mtcars`. The more cars that fall into a grid cell, the more intense the color becomes. It's similar in spirit to a heatmap but works directly on numerical axes. The use of `coord_cartesian(expand = FALSE)` with expansion turned off keeps the view snug to the data, so you are not distracted by extra padding and can focus directly on where the cars actually fall. On the other hand, `geom_bin2d()` divides the plotting area into a grid of rectangular bins and counts how many observations fall into each cell. You can tell it works almost like a two-dimensional histogram, in which case, instead of plotting individual points, it shades each bin according to the number of cars it contains, giving you a clear picture of clusters and densities in the data.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  geom_bin2d() +  
  coord_cartesian(expand = FALSE) +  
  labs(title = "Raster Chart") + theme
```

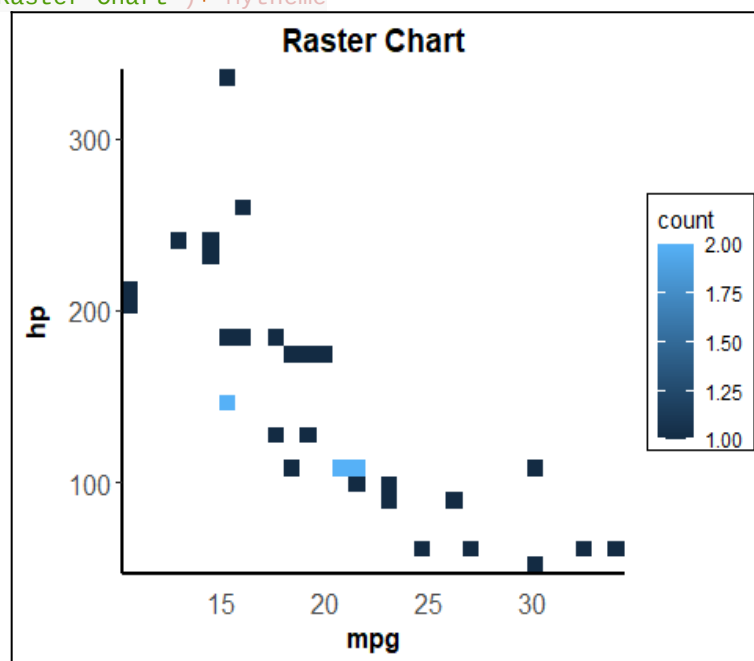


Figure 12. 7: Raster Chart

When you look at this chart, you can tell that the strength of the color in each square represents how many cars fall within that specific mpg and horsepower range. Compared with hexagonal bins, these square bins create more of a grid-like feel, which makes it easier to notice clusters

that align neatly along horizontal and vertical lines. For instance, you will probably spot a dense cluster where mpg runs between 15 and 20 and horsepower sits between 100 and 150, pointing to many cars that share these traits. This setup makes it simple for you to read concentration patterns, since the rectangular bins carve the space into clear segments. As you scan across the chart, the darker patches immediately tell you where most cars are concentrated, making it far easier to interpret than a noisy scatterplot while still respecting the numerical scale.

12.7 Histogram Plot

Histogram is an excellent choice when aiming for a clear and direct representation of how values are distributed. In this instance, we are focusing on horsepower (**hp**) to reveal how many cars fall within specific horsepower intervals. The `geom_histogram()` function segments the data into bins, essentially grouping the values into ranges, and then displays the frequency of observations within each bin. This type of chart is especially effective when you want learners to quickly identify common value ranges, detect clusters, or recognize any noticeable gaps in the data.

```
ggplot(mtcars, aes(x = hp)) +  
  geom_histogram(binwidth = 25, fill = "steelblue", color = "white") +  
  labs(title = "Histogram Plot")+ Mytheme
```

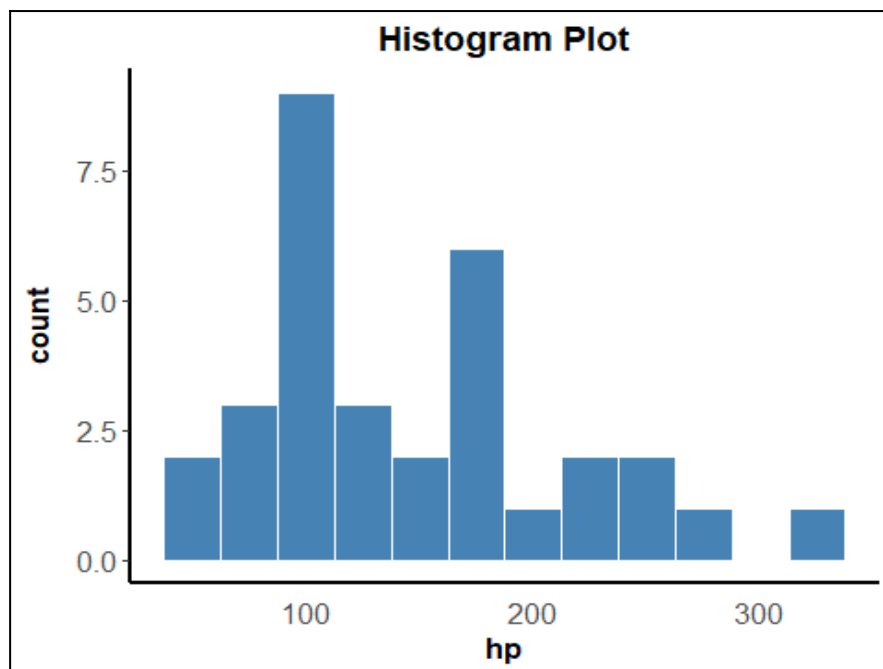


Figure 12. 8: Histogram Plot

The chart presents bars of varying heights, with taller bars indicating that more cars fall within a particular horsepower range. A noticeable concentration may appear around the 100 to 125

horsepower interval, suggesting that this is a common category among the vehicles analyzed. This type of visualization offers a clear and immediate understanding of the distribution. It allows viewers to determine whether the data is symmetrical, skewed, or clustered around multiple values. The simplicity of the format makes it especially effective for highlighting concentrations and supporting a deeper comprehension of patterns in the dataset.

12.8 Two Dimensional Histogram

When you're ready to step up from a simple histogram and want to consider two numeric variables, `geom_bin2d()` does the job well. It's like placing two histograms at right angles and looking at how they overlap. For example, plotting `mpg` and `hp` in `mtcars` this way shows you where the densest intersections of values occur. Each rectangular bin reflects how many observations land in that `mpg` – `hp` region.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  geom_bin2d(binwidth = c(2, 25)) +  
  labs(title = "2D Histogram")+Mytheme
```

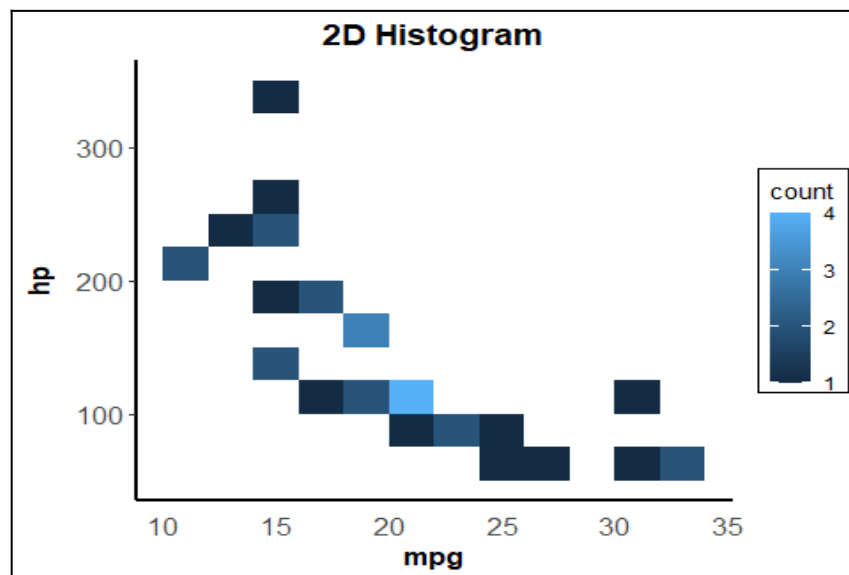


Figure 12. 9: 2D Histogram Chart

In this chart, darker or more saturated rectangles represent combinations of `mpg` and `hp` with more vehicles. If you see a dark square at `mpg` = 21 and `hp` = 110, you know several cars lie in that range. This is excellent for showing concentration over two dimensions which is a big step up from the univariate histogram and a precursor to heatmaps.

12.9 Contour Plot

When you want to explore your data without turning to the usual bars or colors, especially when comparing two continuous variables, a contour plot can be very insightful. Think of it like looking at elevation lines on a physical map. The closer the lines, the greater the density of observations or cluster together. In this example, the function `geom_density_2d(color = "darkgreen")` is what draws the contour lines, with the argument `color = "darkgreen"` specifying the color of those lines.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  geom_density_2d(color = "darkgreen") +  
  labs(title = "Contour Lines") +  
  theme
```

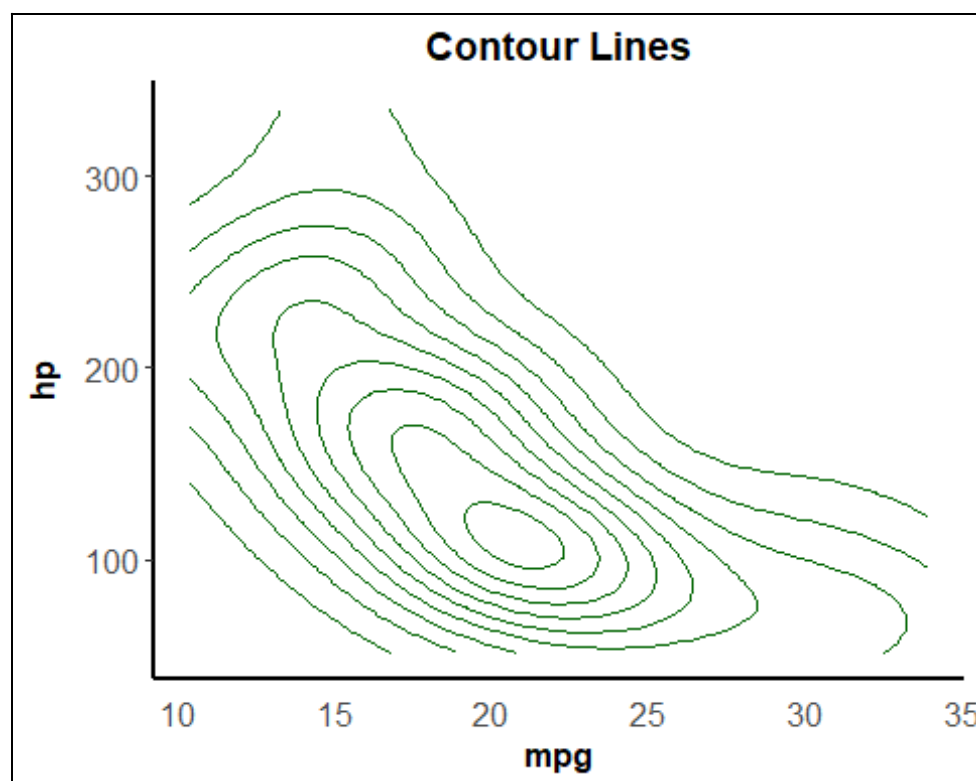


Figure 12. 10: Contour Lines

Look at those bullseye-like shapes which help you see where the data naturally forms peaks. With this chart, you quickly learn to associate tight line patterns with crowding. When you glance at this chart, look for closely packed lines, those are zones of high data density. Each bin's color intensity corresponds to the count of cars falling within that `mpg` - `hp` range. Unlike hexagonal bins, these square bins give a grid-like impression, making it easier to see clusters along clear horizontal and vertical intervals. For example, a dense cluster appear where `mpg` is

between 15 and 20, and horsepower ranges from 100 to 150, showing many cars with these characteristics.

12.10 Filled Contour Plot

Now let's say you want those contour lines but with more 'feel' to them, you want to see where the data thickens using layers of color. That's where `geom_density_2d_filled()` becomes handy. It's like topography meets heatmap. Each filled band between lines represents a density level, and this really helps when you're trying to explain zones of heavier or lighter data.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  geom_density_2d_filled() +  
  labs(title = "Filled Contour Plot") +  
  Mytheme
```

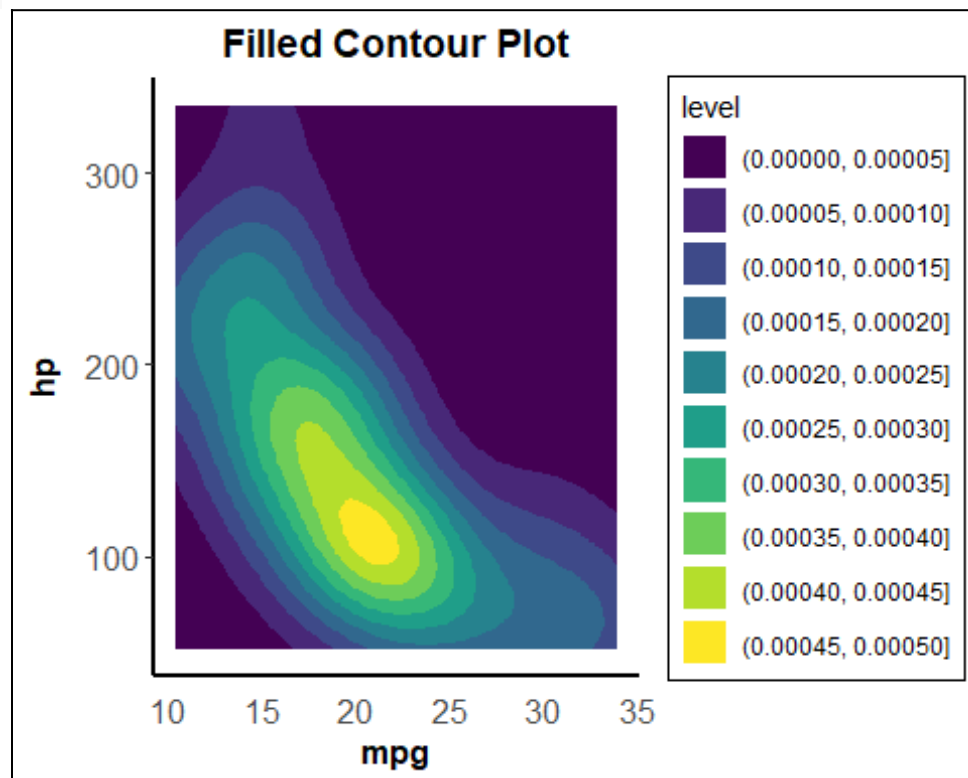


Figure 12. 11: Filled Contour Lines

If you're using this graphic you will love it for its visual simplicity and how it immediately communicates where data bunches up. It bridges the gap between contours and heatmaps, combining the best of both worlds. The plot uses smooth, colored contours to represent different density levels, with darker or more intense shades indicating higher densities. For example, you might see a strong concentration around 15–20 `mpg` and 100–150 `hp`, showing many cars have these characteristics. The gradual gradients provide a clear sense of how car density changes

continuously across the mpg-hp space, highlighting zones of high and low concentration. This visualization helps to understand not only where data points gather but also how their density transitions smoothly across the variables, making it ideal for observing overall distribution patterns in two dimensions. Your eye will naturally go to the darkest area to see where the most data is. Moving outward, the colors fade, suggesting fewer values.

12.11 Heatmap

Now, if you move into grid-based summaries, a heatmap is a strong choice because it lets you see density through color intensity. Here, `ggplot(mtcars, aes(x = mpg, y = hp))` again sets the framework with the same variables. The function `stat_bin2d()` divides the plot area into rectangular bins and counts how many observations fall into each. To make the differences visible, `scale_fill_gradient(low = "brown", high = "red")` assigns lighter or darker shades depending on the frequency, with brown for lower counts and red for higher ones.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  stat_bin2d() +  
  scale_fill_gradient(low = "brown", high = "red") +  
  labs(title = "Heatmap") +  
  theme
```

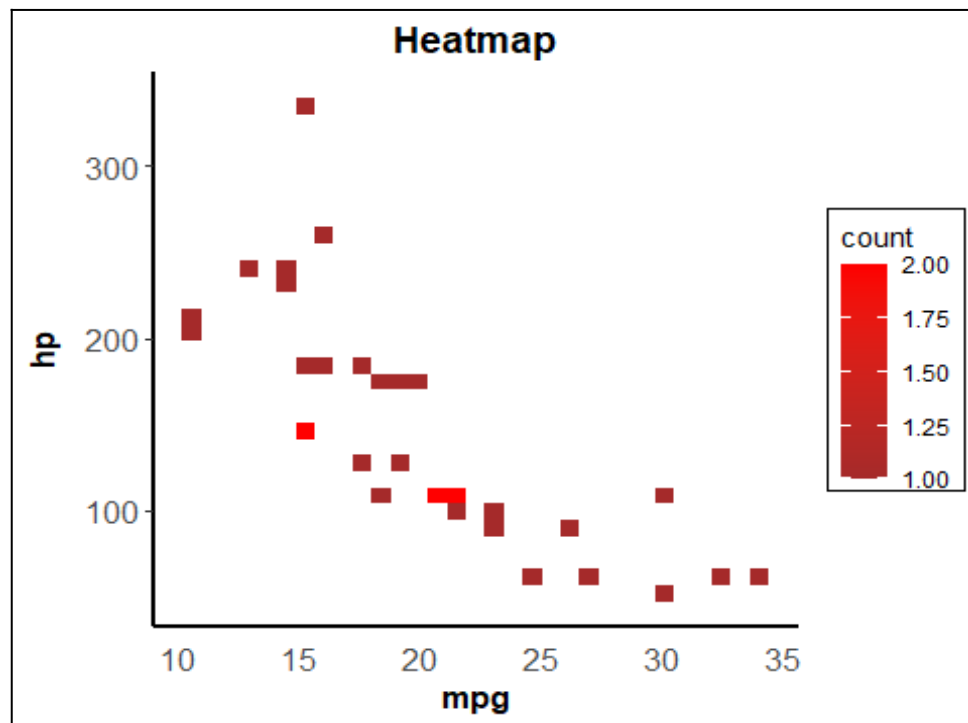


Figure 12. 12: Heatmap

In this heatmap, while exact numeric counts are not directly shown, the color intensity correlates with the frequency of cars in each `mpg - hp` bin. For instance, the deepest red squares represent the highest number of cars sharing similar mpg and horsepower values. If you look closely, these darker red areas might correspond roughly to `mpg` values between about 15 and 25 and `hp` values around 100 to 150, indicating many cars fall in that performance-efficiency range. Conversely, the lighter brown areas have fewer cars. Although the plot doesn't provide exact counts, the gradient effectively conveys where data points concentrate most, giving a strong sense of density and clustering within the data. The resulting chart looks like a colored tile map. Redder areas show heavier concentrations of car data, making it ideal for spotting hot zones in the data. Users of this kind of charts can intuitively understand and match color to density. It simplifies complex pairings and works well when data volume is high and patterns need to be clearly spotted.

12.12 Cartograms

When you're dealing with spatial data and you want to visualize intensity or importance across different regions, cartograms offer a powerful approach. Sometimes, to tell a powerful story, a map needs a little distortion, not in truth, but in proportion. Imagine trying to show which wards have more schools on a normal map, a small ward bursting with schools might look unimportant simply because it occupies little space. A cartogram flips that imbalance, letting shape and size dance to the rhythm of the data itself.

With `cartogram_cont()`, the shapes of each ward shift so their areas mirror the values you care about, in this case, the number of schools. A ward with many schools grows in size, one with fewer shrinks, yet their relative positions remain intact. The geography bends, but the relationships stay truthful.

Using `tmap_mode("plot")` puts you in a mode where the map is drawn as a static image, perfect for reports and publications. Your reshaped wards become the stage, and with `tm_shape()` you gather the layers, styling, and data into one object ready for display. Calling `tm_polygons()` then fills each ward with color shades that echo the school counts, deep, bold colors for abundance, softer tones for scarcity.

```
library(cartogram)
library(tmap)
library(readr)
wards_path <- "D:/ALL/Project/Shape files/kenya_wards/Kenya wards.shp"
wards <- st_read(wards_path, quiet = TRUE) |> st_set_crs(4326)
# Filter for Gwasssi wards
wards_gwasssi <- wards |>
```

```

  filter(str_detect(IEBC_WARDS, regex("Gwasssi South|Gwasssi North", ignore_case =
TRUE))) |>
  st_transform(21036)
places_raw <- read_csv("D:/ALL/Project/Shape files/Suba.csv",
  locale = locale(encoding = "Latin1")) |>
  mutate(across(where(is.character),
    ~ .x |>
      str_replace_all("[\\u2018\\u2019\\u201A\\u201B\\u2032\\u2035]", "'") |>
      str_replace_all("[\\u201C\\u201D\\u201E\\u201F\\u2033\\u2036]", "\"") |>
      iconv(from = "latin1", to = "UTF-8", sub = ""))
  ))
# Convert to sf and transform
places <- places_raw |>
  st_as_sf(coords = c("Longitude", "Latitude"), crs = 4326, remove = FALSE) |>
  st_transform(21036)
# Filter for schools
schools <- places |> filter(Category == "School")
# Count schools per ward
schools_count <- st_join(wards_gwasssi, schools) |>
  group_by(IEBC_WARDS) |>
  summarise(school_count = n(), .groups = "drop") |>
  mutate(
    # Ensure correct capitalization
    IEBC_WARDS = case_when(
      str_detect(IEBC_WARDS, regex("south", ignore_case = TRUE)) ~ "Gwasssi South",
      str_detect(IEBC_WARDS, regex("north", ignore_case = TRUE)) ~ "Gwasssi North",
      TRUE ~ IEBC_WARDS
    ),
    # Label to display name + count
    label_text = paste0(IEBC_WARDS, " (", school_count, ")")
  )
carto <- cartogram_cont(schools_count, "school_count", itermax = 5)
tmap_mode("plot")
map_obj <- tm_shape(carto) +
  tm_polygons("school_count", palette = "Blues", title = "Number of Schools") +
  tm_text("label_text", size = 0.8, col = "black", shadow = TRUE) +
  tm_layout(
    title = "Gwasssi Wards Cartogram",
    title.size = 1.2,
    legend.outside = TRUE,
    frame = FALSE
  )
)

```

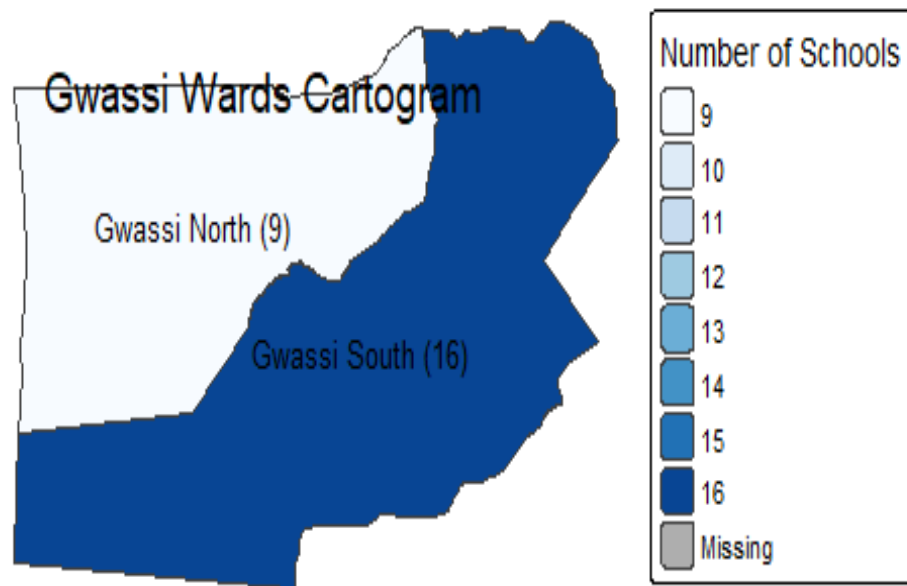


Figure 12. 13: Cartogram

In the cartogram, Gwasssi South visually expands far beyond its usual geographic footprint, reflecting its much higher school count, fourteen in total. The deep dark blue shade from the “Blues” palette reinforces its dominance, making it instantly stand out on the map. By contrast, Gwasssi North appears compressed and paler in color, indicating a much smaller number of schools, just six. This visual shrinking, combined with the light blue tone, intuitively signals its comparatively limited distribution of educational facilities.

Such a transformation makes it easy to grasp spatial inequalities. For example, while Gwasssi South may need more road access to accommodate the greater concentration of schools, Gwasssi North might instead focus on adding new schools to bridge the service gap. The beauty of this cartogram is that size becomes proportional to school count, so even if someone is unfamiliar with Suba’s geography, they can instantly understand where the educational “hot spots” and “cold spots” lie.

CHAPTER THIRTEEN: MULTIVARIATE ANALYSIS

Multivariate analysis involves examining multiple variables simultaneously to uncover relationships, patterns, and interactions within complex datasets. This type of analysis is crucial in fields like finance, healthcare, and marketing, where decision-making depends on understanding how several factors influence outcomes. To represent multivariate data visually, we use charts such as scatter plot matrices, bubble charts, parallel coordinates plots, and radar charts. A scatter plot matrix displays multiple scatter plots in a grid format, showing pairwise relationships between variables. Bubble charts add a third dimension (usually size) to standard scatter plots, allowing for comparison across three variables. Parallel coordinates plots use axes arranged in parallel to represent multiple variables for each data point, making it possible to observe patterns across dimensions. Radar charts, also known as spider charts, map multiple variables on a circular grid, providing a quick comparison of profiles or performance metrics. These charts help analysts interpret complex, high-dimensional data with clarity.

13.1 Parallel Coordinates Plot

Suppose you've got multiple continuous variables and want to see how they relate per observation. Parallel coordinate plots are ideal. Each variable becomes a vertical axis, and every row is a line. For `mtcars`, this means we can watch how `mpg`, `hp`, `wt`, and `dis` interact per car. The `ggparcoord()` patterns emerging across multiple axes, showing linked behavior.

```
library(GGally)
ggparcoord(mtcars, columns = c(1, 4, 6, 3), scale = "globalminmax") +
  labs(title = "Parallel Coordinates Plot") + Mytheme
```

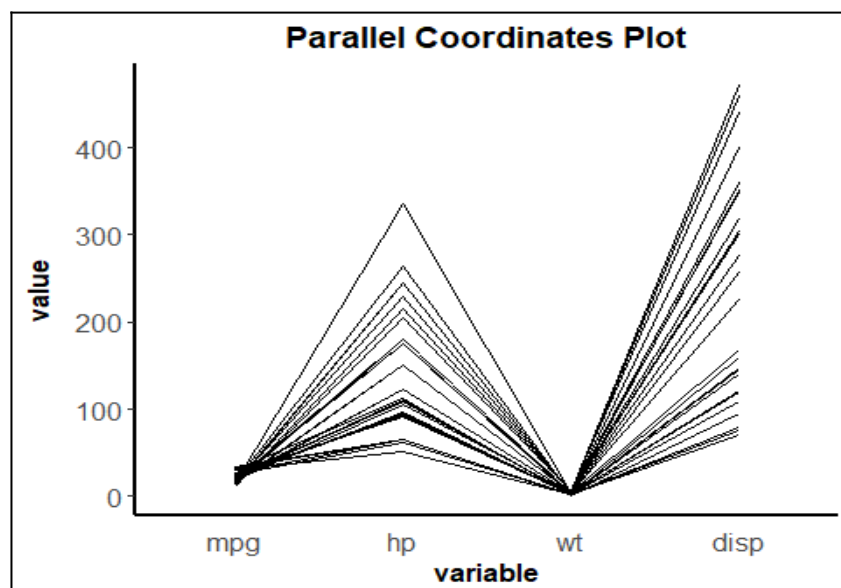


Figure 13. 1: *Parallel Coordinates Plot*

With parallel coordinates plot, you're able to display multiple variables simultaneously, allowing you to compare how each variable performs across different attributes. The plot's ability to show all these variables in one view helps identify patterns, such as cars that maintain consistent performance across metrics or those that trade off one attribute for another. Clusters of lines with similar shapes indicate groups of cars with similar multivariate profiles, making this plot valuable for exploring complex relationships within the dataset.

Each car becomes a squiggly line across variables. When several lines follow the same shape, you've found a pattern. If a few lines suddenly dip or spike, you're spotting outliers. It's brilliant for showing multi-dimensional relationships and lets students explore more than two variables at once. Parallel plots are like bridges between scatterplots and matrices.

13.2 Radar Chart

A radar chart, often called a spider chart, displays multiple variables as axes arranged in a circular layout, where each subject is drawn as a polygon connecting its values. It's especially useful for comparing a few items across several features at once. Radar charts simplify comparison and profile analysis across multiple attributes.

In the example using the `mtcars` dataset, if you compare the three cars across `mpg`, `wt`, and `hp`, you get a snapshot of their strengths and weaknesses. To properly scale the chart, the function `rep()` is used to repeat maximum and minimum values, while `rbind()` binds these rows with the original car data so the radar chart has consistent limits. The `colnames()` function then renames the axes to more readable labels (MPG, HP, Weight). The `radarchart()` function from the `fmsb` package plots the spider-web style visualization, with each polygon highlighting how a car performs across the chosen variables.

```
library(fmsb)
df <- mtcars[c("Mazda RX4", "Valiant", "Camaro Z28"), c("mpg", "hp", "wt")]
df <- rbind(rep(1.5 * max(df), 3), rep(0, 3), df)
colnames(df) <- c("MPG", "HP", "Weight")
radarchart(df, axistype = 1,
            pcol = c("red", "blue", "green"), plty = 1,
            title = "Radar Chart")
```

Radar Chart

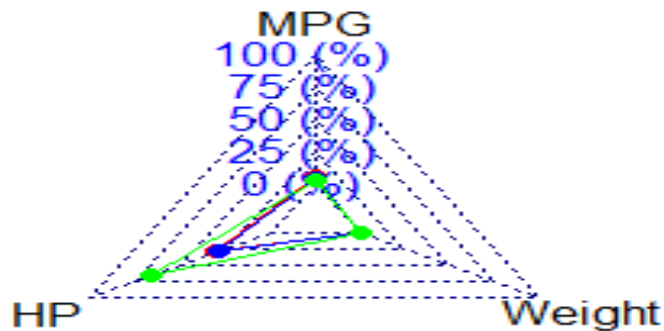


Figure 13. 2: Radar

Users of these kind of charts can easily see how, for example, one car scores better in **mpg** while another is heavier. Looking at the chart, the **Mazda RX4** has around 21 **MPG**, approximately 110 **HP**, and weighs close to 2.6 (in 1000 lbs). The **Valiant** shows about 18 **MPG**, roughly 105 **HP**, and a weight near 3.5. The **Camaro Z28** stands out with higher horsepower around 245 **HP** but lower fuel efficiency at about 15 **MPG** and weighs around 3.4.

13.3 Bubble Chart

Bubble charts are a great way to show three variables at once in a single visual. Imagine using a scatter plot but with the added bonus of a third variable that controls the size of each point. In our case, we're plotting miles per gallon (**mpg**) on the x-axis, horsepower (**hp**) on the y-axis, and weight (**wt**) as the size of the bubble. The **mtcars** dataset is excellent for this purpose because it includes numeric variables that represent real-world vehicle characteristics.

The code utilizes **ggplot2**, where **aes(size = wt)** brings in that multivariate flair. This results in a graphic information where bigger circles mean heavier cars, while their location tells us how powerful and fuel-efficient they are. This kind of chart is incredibly handy when you want to quickly spot clusters or outliers involving three variables.

```
ggplot(mtcars, aes(x = mpg, y = hp, size = wt)) +
  geom_point(alpha = 0.6, color = "blue") +
  labs(title = "Bubble Chart", x = "Miles per Gallon", y = "Horsepower")
+Mytheme
```

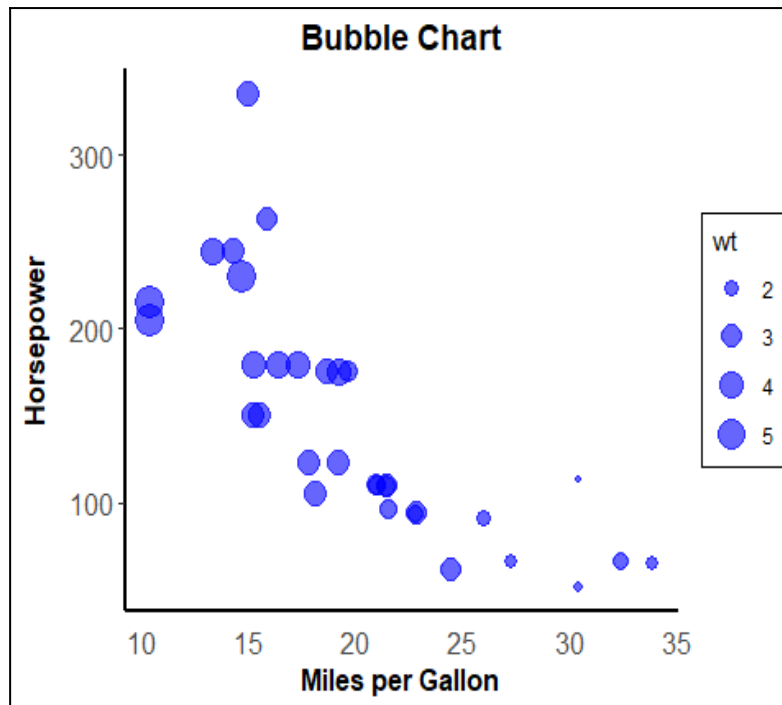


Figure 13. 3: Bubble Chart

In this bubble chart, you can notice the relationship between miles per gallon (**mpg**) and horsepower (**hp**) for different cars in the mtcars dataset, with the size of each bubble representing the car's weight (**wt**). Cars with higher **mpg** appear further to the right, while those with higher horsepower are positioned higher on the chart.

Similarly, You can also see that bubbles with larger size, representing heavier cars, tend to cluster towards lower **mpg** and higher horsepower, such as the ones near 15 **mpg** and 250 **hp**, indicating powerful but heavier vehicles. Conversely, smaller bubbles, indicating lighter cars, are mostly found with higher **mpg** and lower horsepower, reflecting more fuel-efficient, lighter cars. What stands out is that there's a trade-off: cars with more horsepower (higher on the y-axis) often have lower **mpg** (left side of x-axis) and larger weights (larger bubbles). This visual makes multivariate relationships tangible at a glance. You can easily tell which cars are performance-heavy and which are more economical and therefore provides valuable insights in the analysis of the automobiles.

13.4 Multivariate Heatmap

In multivariate data analysis, visual clarity is crucial when investigating the relationship between two continuous variables. Heatmaps offer a compelling graphical solution by transforming raw

scatter plots into grid-based summaries where color intensity signifies the density or frequency of observations. This method is particularly effective when dealing with large datasets, where traditional scatter plots may become cluttered or fail to highlight concentration patterns effectively.

The `ggplot2()` package in R provides a straightforward approach to generating such heatmaps. In this example, the `mtcars` dataset is used to examine the relationship between miles per gallon (`mpg`) and horsepower (`hp`). The code initiates a `ggplot` object with `mpg` mapped to the x-axis and `hp` to the y-axis. Instead of plotting individual points, the function `stat_bin2d()` is applied. This function divides the coordinate space into rectangular bins and counts the number of observations that fall within each bin.

To enhance interpretability, the `scale_fill_gradient()` function is introduced to assign colors to the bins based on the number of observations. A gradient from light blue to red is chosen, where light yellow indicates lower density and red indicates higher density.

```
ggplot(mtcars, aes(x = mpg, y = hp)) +  
  stat_bin2d() +  
  scale_fill_gradient(low = "lightblue", high = "red") +  
  labs(title = "Multivariate Heatmap")+Mytheme
```

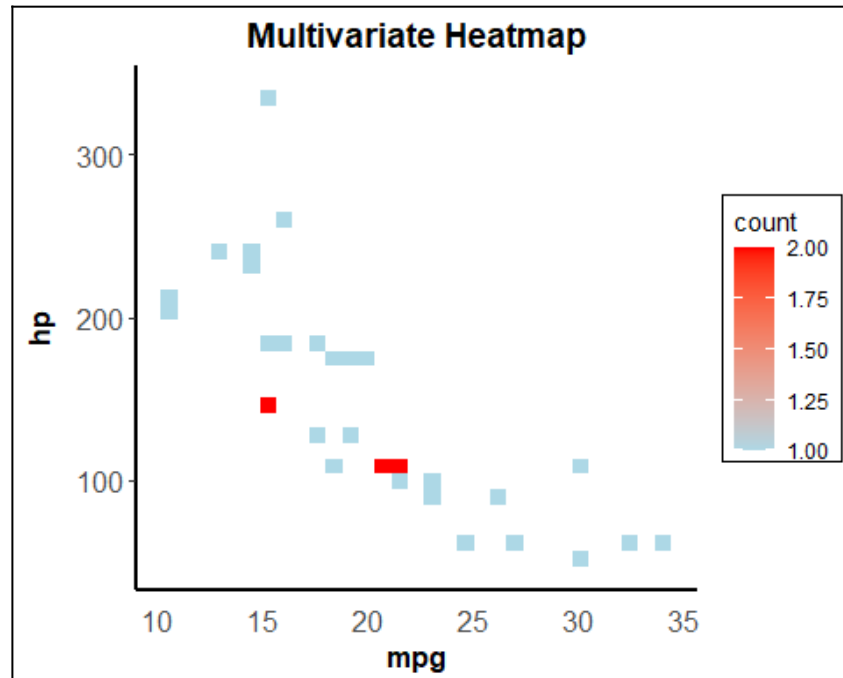


Figure 13. 4: Multivariate Heatmap

It's clear from the plot that one can easily uncover patterns, clusters, or anomalies that might not be immediately evident in a raw scatter plot. For instance, the heatmap reveals that a significant

number of vehicles exhibit moderate horsepower and fuel efficiency, while regions representing high horsepower and low fuel efficiency remain sparsely populated. The color gradient, ranging from light blue to red, indicates how many cars fall into each binned region of `mpg` and `hp`. Darker red areas represent higher concentrations of cars. For example, there's a noticeable concentration around 15–20 `mpg` and 100–150 `hp`, showing many cars with moderate fuel efficiency and power. Similarly, another dense cluster appears near 25–30 `mpg` with lower horsepower (around 50–100 `hp`), indicating lighter, more fuel-efficient cars with less power. The heatmap serves as a highly effective tool for visualizing bivariate distributions, especially when the dataset is large or when visual simplicity is desired. By encoding frequency through color, the plot allows users to intuitively grasp the density of observations across the variable space.

CHAPTER 14: ARRANGEMENT OF GRAPHICS

The arrangement of graphics within a piece of writing plays a crucial role in how readers understand and interpret data. Charts and visualizations do not exist in isolation; their placement, sequence, and alignment influence the clarity and impact of the message being communicated. A well-organized set of visuals can guide the reader through a logical narrative, highlight key comparisons, and reveal patterns that may otherwise remain hidden. Conversely, poorly arranged graphics can confuse, overwhelm, or obscure important insights.

Effective arrangement considers both the purpose of the visuals and the flow of information. For instance, charts may be sequenced to show progression over time, or grouped to facilitate comparison between categories. By thinking carefully about how visuals relate to each other, authors can create a seamless visual story that complements the textual analysis.

Several methods help in arranging multiple charts cohesively. Facet wrapping splits a dataset into subsets and lays them out in a grid, enabling direct comparison across groups while maintaining consistent scales and styles. Patching allows multiple individual plots to be combined into a single layout, giving precise control over alignment, relative sizing, and panel positioning, ideal for dashboards or multi-panel figures. Additionally, annotations and marginal arrangements add contextual elements such as axis histograms, boxplots, or callouts, enhancing the main graphic with extra layers of information without cluttering the figure. By thoughtfully organizing graphics, writers transform raw visualizations into a coherent, engaging narrative. Strategic arrangement ensures that insights are immediately accessible, comparisons are clear, and the reader can follow the story the data tells with ease and confidence.

14.1 Patch Work and Annotation

Patchwork lets you combine multiple plots into one layout. It's like stitching individual charts into a visual story. Here, we pair a scatter plot with a histogram to show the relationship between `mpg` and `hp`, and how `mpg` values are distributed. The `patchwork` library is used to join these `ggplots`, and annotation is added to bring in commentary or labeling right onto the chart. We use `lot_annotation()` to title the full patch and `annotate()` for in-plot remarks. It's ideal when you need to compare or narrate multiple plots side-by-side.

```
library(patchwork)
scatter <- ggplot(mtcars, aes(mpg, hp)) + geom_point() + labs(title = "Point Chart")
```

```
hist <- ggplot(mtcars, aes(mpg)) + geom_histogram(bins = 10, fill = "steelblue") +
labs(title = "Histogram ")

scatter + hist + plot_annotation(title = "Combined Plots")+Mytheme
```

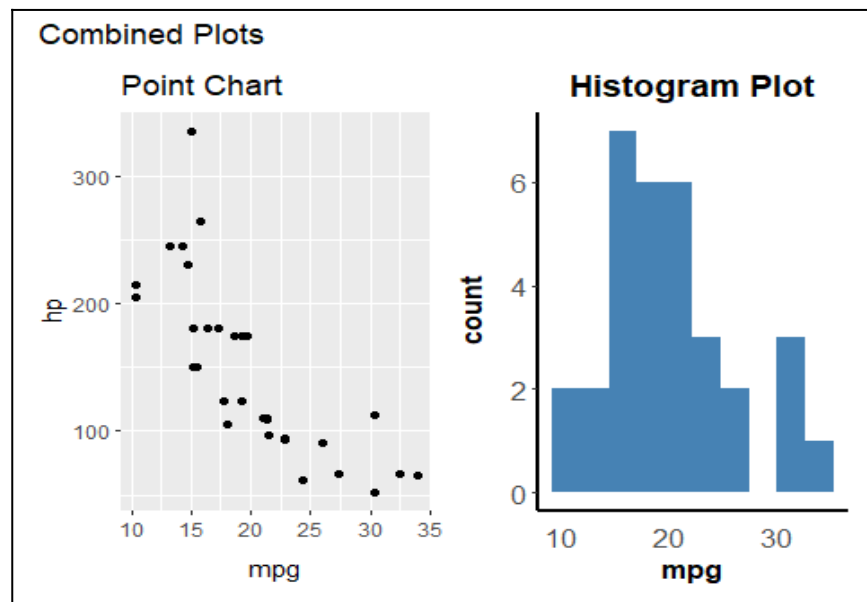


Figure 14. 1: Patched Point and Histogram Plots

When you view the output, you see how side-by-side plots complement each other. While the scatter plot shows how two variables interact, the histogram dives deeper into the distribution of one. Together, they deliver a fuller analytical picture.

14.2 Wrapping

Wrapping, often done using `facet_wrap()` in `ggplot2`, helps break down complex datasets into digestible subplots based on a factor. Let's take our `mtcars` dataset and group cars by their number of cylinders (`cyl`). With `facet_wrap()`, we generate one small plot per group. This method is fantastic for comparing similar trends across categories, like how horsepower and mileage relate in 4-cylinder versus 8-cylinder cars. We wrap the same base chart across different panels, using `scales = "free"` to let each facet breathe to its own maximum value.

```
ggplot(mtcars, aes(mpg, fill = cyl))+
  geom_density(alpha = 0.5) +
  facet_wrap(~cyl, scales = "free") +
  labs(title = "Faceted Density Plot", x = "Miles Per Gallon", y = "Density")
+Mytheme
```

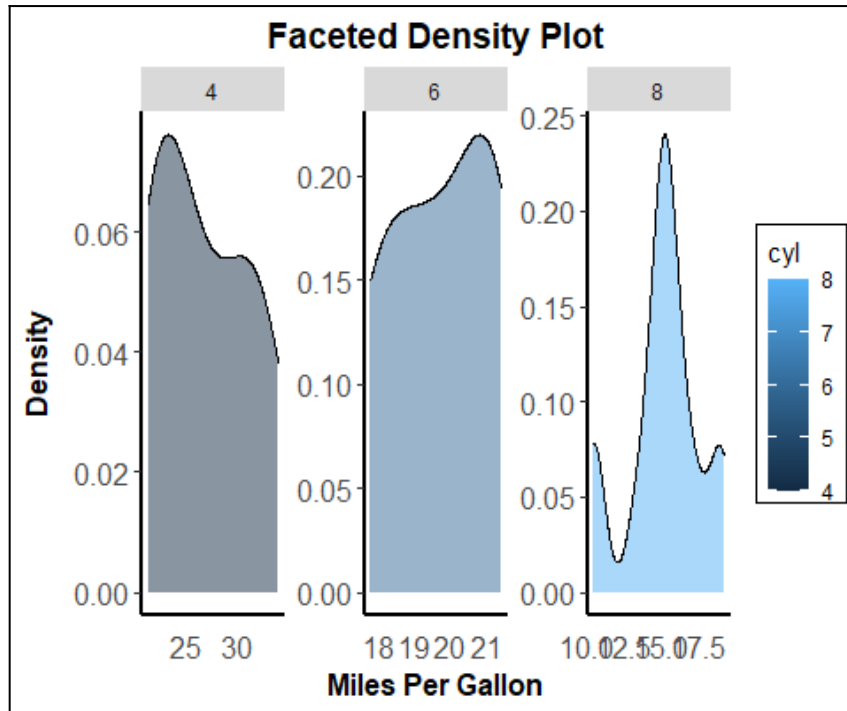


Figure 14. 2: Facet *Wrapped Density Plots*

In the result, you can easily scan how the relationship between `mpg` and `hp` shifts based on engine type. For example, 8-cylinder cars tend to have high horsepower but low mpg. This structure makes such differences more apparent than in a single, crowded scatter plot.

14.3 Marginal Plot

In data analysis, scatter plots are commonly used to visualize the relationship between two continuous variables. However, while scatter plots effectively display the interaction between variables, they may not readily communicate the individual distribution of each variable along its axis. To address this limitation, marginal plots can be incorporated into the scatter plot to provide insights into the univariate distributions of the variables under investigation. The R packages `ggplot2()` and `ggExtra()` offer powerful tools for this purpose. The following discussion presents a practical example using the `mtcars` dataset to construct and interpret scatter plots with marginal enhancements.

The process begins by loading the necessary libraries. The `ggplot2` package is used for plotting, `ggExtra()` is employed to add marginal plots to the scatter plot, and `gridExtra()` is used to arrange multiple graphical outputs in a structured format. The code starts by creating a base

scatter plot with `mpg` (miles per gallon) on the x-axis and `hp` (horsepower) on the y-axis. This base plot serves as the foundation for all subsequent marginal enhancements.

To explore various marginal summaries, four versions of the base scatter plot are constructed, each incorporating a different type of marginal visualization. The first variant uses histograms as marginal plots, which display the frequency distribution of `mpg` and `hp` along the respective axes. The second variant includes violin plots as marginal enhancements. Violin plots combine features of boxplots and density plots, thereby revealing the distribution's shape, central tendency, and variability in a more fluid form. The third plot integrates density plots on the margins, which provide smooth estimates of the distribution, highlighting concentrations of data points along each axis. The fourth and final variant adds boxplots as marginal plots. Boxplots are particularly effective in identifying medians, quartiles, and potential outliers, thereby offering a robust summary of the data distribution.

These four plots are then organized into a 2-by-2 grid layout using the `grid.arrange()` function from the `gridExtra()` package. A common title is included at the top of the grid to label the collection as "Marginal Plots Overview." This arrangement facilitates visual comparison of the different marginal plotting techniques applied to the same dataset and variable pairing.

```
library(ggExtra)
library(gridExtra)
# Base plot
p <- ggplot(mtcars, aes(mpg, hp)) + geom_point()
# Create marginal plots
his <- ggMarginal(p + labs(title = "Histogram"), type = "histogram")
vio <- ggMarginal(p + labs(title = "Violin"), type = "violin")
dens <- ggMarginal(p + labs(title = "Density"), type = "density")
box <- ggMarginal(p + labs(title = "Boxplot"), type = "boxplot")
# Arrange the plots in a 2x2 grid
grid.arrange(his, vio, dens, box, ncol = 2, top = "Marginal Plots")
```

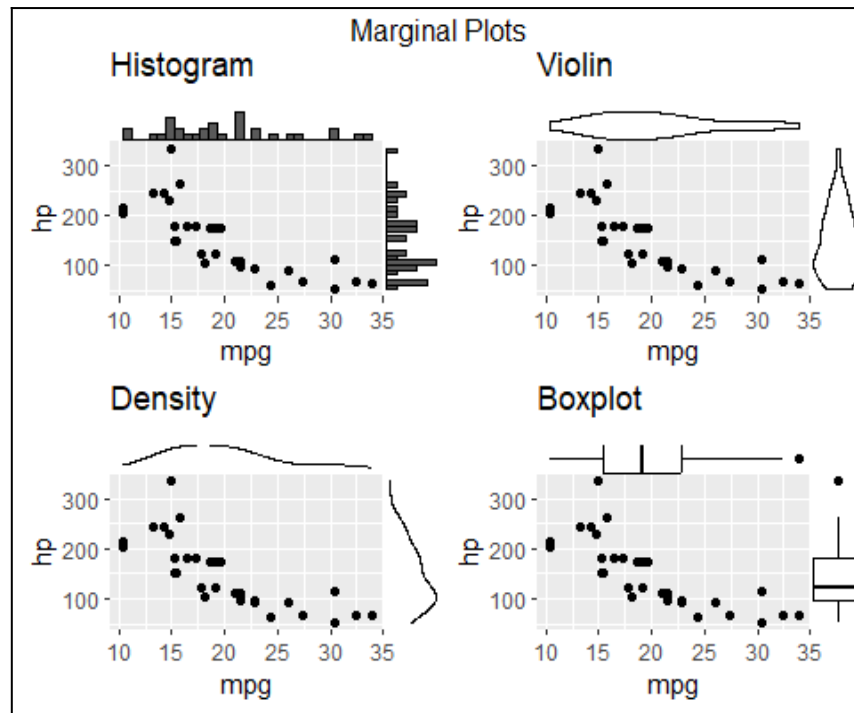


Figure 14. 3: Assorted Marginal Plots

The resulting chart enables the viewer to assess both the bivariate relationship between `mpg` and `hp` and the univariate distribution of each variable. In the histogram version, one can observe how frequently certain values occur, making it easy to detect clusters or gaps. The violin version emphasizes the density and spread, revealing whether the data are symmetrically distributed or skewed. The density-enhanced scatter plot provides continuous and smooth representations of the distributions, which are particularly useful for understanding the concentration of values. Lastly, the boxplot version highlights the range and central tendency, clearly delineating quartiles and identifying outliers. The combination of scatter plots with various marginal enhancements provides a multifaceted view of the data. This approach allows analysts to better understand both the interaction between variables and the individual behavior of each variable. The use of `ggplot2()`, `ggExtra()`, and `gridExtra()` in R demonstrates an efficient and elegant method for constructing these visualizations, ultimately improving exploratory data analysis and interpretation.

GLOSSARY

2D – Flat visual representation with two axes (x and y).

3D – Visualization with three dimensions, adding depth to 2D.

3D pie chart – Circular chart showing proportions with depth effect.

aes – Aesthetic mapping in ggplot2 linking data to visuals.

Algorithm – Step-by-step process to solve a problem or perform a task.

Alluvial chart – Flow diagram showing categorical changes over time.

alpha – Controls transparency of graphical elements.

Analysis – Process of examining data to extract meaning or insights.

annotate – Adding labels or notes on a plot.

Anomalies – Unusual data points differing from the overall pattern.

Area map – Map shaded to represent magnitude across regions.

arrange – Function to order or sort data in R/dplyr.

Assumption – Condition expected to hold true in analysis.

Asymmetry – Imbalance in data distribution.

axis – Reference lines (x or y) showing data scales.

Background – Plot space behind main chart elements.

background.circle.colour – Setting for background circle color in a chart.

Band – Range area, often used for uncertainty shading.

Bar – Rectangular graphical element showing value size.

Bar graph – Chart with bars comparing categories.

bins – Intervals used to group data in histograms.

binwidth – Size of bins controlling histogram resolution.

Bivariate – Involving two variables.

Bivariate relationship – Association between two variables.

Bold – Font style making text thicker and darker.

Boxplot – Chart showing distribution with quartiles and outliers.

Bubble chart – Scatter plot where point size shows a third variable.

Bump chart – Tracks rank changes of categories over time.

Butterfly chart – Mirrored bar chart for comparing two groups.

Candlestick chart – Finance chart showing open, close, high, low prices.

caption – Text placed below a figure or chart for description.

Cartogram map – Map where regions are resized by data values.

Categorical variable – Variable with distinct categories (e.g., gender).

Change – Variation in values across time or groups.

Chart – Visual representation of data.

Chloropleth – Map shaded by data values in geographic areas.

closer – Could mean zooming or narrowing data focus.

Cluster – Group of similar data points.

Clusters – Multiple groups of related data points.

Code – Set of instructions written in a programming language.

color – Visual attribute used to distinguish categories or values.

Color gradient – Continuous range of colors representing values.

Comparison – Evaluating differences or similarities in data.

Components – Parts making up a dataset or visualization.

Composition – Showing parts that make a whole (e.g., pie chart).

Compound bar – Stacked bar chart with multiple segments.

Concentration – Degree to which values cluster together.

Confidence interval – Range estimating where a true value lies.

Contour – Lines joining equal values in 2D/3D space.

Contour chart – Visualization showing contours over a grid.

coord_polar – Converts plot to polar coordinates.

Correlation – Statistical relationship between variables.

Correlation matrix – Table showing correlation coefficients between variables.

Count – Number of observations or items in a dataset.

Crowding – Overlapping or cluttering of elements in a visualization.

Crs – Coordinate Reference System used in mapping and GIS.

data – Raw facts, figures, or observations collected for analysis.

dataset – A structured collection of related data.

date – A time-related variable showing day, month, and year.

Decomposed time series – Breaking down a series into trend, seasonal, and residual parts.

Dense – A region with high concentration of data points.

Density – A measure showing how data values are distributed.

Discrete – Data with separate, distinct values (e.g., integers).

Distribution – The spread or pattern of data values.

Divergent chart – Chart showing deviations around a central point.

dodge – Positioning method that avoids overlapping by spacing groups.

Donut – A variation of a pie chart with a hole in the middle.

Dot – A marker used to represent a data point.

Dot plot – A chart using dots to show frequency of values.

dplyr – R package for data manipulation and transformation.

Dumbbell chart – compares paired values while emphasizing their differences.

Error – The difference between predicted and actual values.

Error bar – Graphical representation of variability or uncertainty.

ETS – Exponential smoothing method for forecasting time series.

Exploratory – Early stage analysis to understand data patterns.

Exploratory analysis – Initial data study to discover trends or issues.

Face – Visual panel in a multi-plot layout.

facet – Technique to split data into multiple panels in ggplot2.

facet_grid – Lays out plots in a grid by row and column factors.

facet_wrap – Wraps plots into multiple panels based on one variable.

factor – A categorical variable with fixed levels.

Fan chart – Chart showing forecast uncertainty as widening bands.

Figure – A graphical display of data or concepts.

File – A saved digital container of data or information.

fill – Aesthetic controlling color inside shapes in ggplot2.

filter – Operation selecting rows that meet given conditions.

Flow – Movement or transfer of data, goods, or energy.

Flow chart – Diagram showing steps or processes with arrows.

Font size – The size of text characters in a plot or document.

fontface – Style of text (plain, bold, italic) in ggplot2.

forcats – R package for handling categorical variables (factors).

forecast – Prediction of future values using past data.

Frequency – How often a value or event occurs.

Geography – Spatial data linked to location or maps.

geom – Geometric object in ggplot2 that represents data.

geom_alluvium – Creates alluvial flows in Sankey-like diagrams.

geom_area – Plots filled areas under a line curve.

geom_bar – Displays data as rectangular bars.

geom_bin2d – Groups data into 2D bins and colors by count.

geom_boxplot – Summarizes data distribution with quartiles and outliers.

geom_bump – Shows rank changes or trends with smooth lines.

geom_col – Draws bars with heights proportional to values.

geom_col_pattern – Fills bars with textures or patterns.

geom_count – Sizes points by the number of overlapping cases.

geom_crossbar – Horizontal bar showing central value with range.

geom_density – Smooth curve showing data distribution.

geom_density_2d – Contour lines showing data concentration in 2D.

geom_density_2d_filled – Filled contours showing density levels in 2D.

geom_dotplot – Dots stacked to show frequency distributions.

geom_edge_link – Plots edges (connections) in network diagrams.

geom_errorbar – Adds vertical error bars to show variability around data points.

geom_errorbarh – Adds horizontal error bars for uncertainty visualization.

geom_hex – Creates hexagonal bins for 2D density plots.

geom_histogram – Displays frequency distribution of a variable in bins.

geom_hline – Draws horizontal reference lines across the plot.

geom_image – Places images at specified data points in a plot.

geom_jitter – Adds random noise to points to reduce overplotting.

geom_label – Adds text labels with background rectangles.

geom_label_repel – Adds non-overlapping text labels with repulsion.

geom_line – Connects data points with straight lines.

geom_linerange – Draws vertical lines from minimum to maximum values.

geom_map – Plots polygonal map data from shapefiles.

geom_node_point – Adds points for graph/network nodes.

geom_node_text – Adds labels to nodes in network graphs.

geom_path – Connects observations in order of appearance.

geom_point – Plots data points on x-y coordinates.

geom_pointrange – Adds a point with a vertical range line.

geom_qq_line – Adds a reference line to QQ plots.

geom_quantile – Adds quantile regression lines to data.

geom_raster – Plots pixel-based rectangular cells.

geom_rect – Draws rectangles with defined boundaries.

geom_ribbon – Plots shaded areas between two y-values.

geom_rug – Adds tick marks at data points along axes.

geom_segment – Draws line segments between two points.

geom_sf – Plots simple features (spatial vector data).

geom_sf_text – Labels spatial features with text.
geom_smooth – Adds smoothed trend lines to data.
geom_spoke – Draws radiating lines (spokes) from points.
geom_step – Connects data points with step-like lines.
geom_stratum – Draws rectangles for stratified alluvial plots.
geom_stream – Plots flowing bands for streamgraphs.
geom_text – Adds text labels at data points.
geom_tile – Draws squares/rectangles for heatmaps.
geom_treemap – Visualizes hierarchical data as nested rectangles.
geom_treemap_text – Adds labels inside treemap rectangles.
geom_violin – Shows data distribution using mirrored density plots.
geom_vline – Adds vertical reference lines.
Geospatial analysis – Examining data with geographic/spatial context.
Geospatial heatmap – Visual density maps using color gradients.
ggArrange – Combines multiple ggplots in a grid layout.
ggcorr – Creates correlation matrix plots.
ggcorrplot – Enhanced visualization of correlation matrices.
ggMarginal – Adds marginal density/histograms to scatter plots.
ggparcoord – Plots parallel coordinate charts for multivariate data.
ggplot – Initializes a ggplot object for building plots.
ggplot2 – R package for advanced data visualization.
ggtheme – Preset visual styles for ggplot2 charts.
ggtitle – Adds a main title to the plot.
github – Online platform for code hosting and collaboration.
Gradient – Smooth transition between colors.
Graph – A collection of nodes and edges representing relationships.
Graphics – Base R system for creating plots.
grep – Function to search patterns in strings.
Grid – Framework for arranging graphical objects in R.
Gridded chart – Chart arranged in rows and columns for clarity.
grViz – Creates graph/network diagrams using DiagrammeR.
gsub – Function to replace patterns in strings.
gtitle – Adds graph titles in visualization tools.
guides – Controls legends and scales in ggplot2.
Heatmap – Uses color intensity to show data magnitude.
height – Vertical size/dimension of a plot element.
Heteroscedasticity – Unequal variance in regression residuals.
Hex – Six-sided polygon used in binning/plots.
Hex chart – Visualization using hexagonal bins.
Hexagon – Six-sided shape often used in plots.
Hexagonal – Refers to data arranged in hexagons.
high – Refers to large or upper values.
Histogram – Chart of frequency distribution.
Histogram chart – Another term for histogram.
hjust – Horizontal justification of text.
Horizontal count chart – Displays counts in horizontal bars.

Horizontal errorbar – Error bars displayed horizontally.

Hotspots – Areas of high activity or density.

Icon – Small pictorial symbol.

iconv – Function for encoding conversion in R.

Interpretation – Explaining meaning of results.

itermax – Maximum number of iterations allowed.

jitter – Adds random variation to points.

keyheight – Height of legend keys.

keywidth – Width of legend keys.

label – Text annotation for elements.

labelcex – Controls label text size.

labeller – Function to customize facet labels.

labels – Text used to identify data.

labs – Adds labels (title, x, y) to plots.

Latitude – Geographic coordinate north-south.

Layer – Visual component added to a plot.

lead – Shifts a variable forward in time.

left – Refers to left alignment/position.

legend – Key explaining plot symbols/colors.

lengths – Number of elements in vectors.

levels – Unique categories in a factor.

Library – Collection of reusable R functions.

library(arules) – Package for association rule mining.

library(car) – Package for regression and data analysis.

library(carData) – Datasets supporting car package.

library(cartogram) – For cartogram map creation.

library(data.tree) – For hierarchical data structures.

library(DiagrammeR) – Package for graph diagrams.

library(dplyr) – Core tidyverse package for data manipulation.

library(fable) – Package for forecasting models.

library(feasts) – Time series feature extraction.

library(fmsb) – Package for radar/spider charts.

library(forcats) – Factor handling package in tidyverse.

library(ggalluvial) – For alluvial (flow) plots.

library(GGally) – Extensions for ggplot2 (e.g., pairs plots).

library(ggbump) – For bump charts.

library(ggcorrplot) – Visualization of correlation matrices.

library(ggExtra) – Adds marginal plots to ggplot2.

library(ggimage) – Embeds images into plots.

library(ggpattern) – Adds patterns/textures to ggplot2.

library(ggplot2) – Core plotting system in R.

library(ggradar) – Radar/spider chart plotting package.

library(ggraph) – For network and graph visualization.

library(ggrepel) – Non-overlapping text labels in ggplot2.

library(ggridges) – For ridge density plots.

library(grid) – Low-level graphics system in R.

library(hexbin) – Hexagonal binning for large datasets.
library(igraph) – Network/graph analysis package.
library(lubridate) – Simplifies working with dates/times.
library(magick) – Image processing in R.
library(patchwork) – Combines multiple ggplots.
library(plotrix) – Miscellaneous plotting functions.
library(raster) – Spatial raster data analysis.
library(readr) – Fast data import functions.
library(remotes) – Installs packages from remote repositories.
library(scales) – Functions for scaling/formatting ggplot2.
library(showtext) – Adds custom fonts to plots.
library(stringr) – Simplifies string manipulation.
library(tibble) – Modern reimagining of data frames.
library(tidyverse) – Collection of R packages for data science.
library(tmap) – Thematic mapping package.
library(treemap) – Treemap visualization package.
library(treemapify) – ggplot2 extension for treemaps.
library(tsibble) – Time series in tidy data format.
library(viridis) – Colorblind-friendly palettes.
library(waterfall) – For waterfall charts.
library(webr) – Web-based visualization package.
library(writexl) – Writes Excel files.
limits – Range limits for scales/axes.
Line – Straight connection between points.
Line chart – Shows trends over time with connected points.
Linerange chart – Displays vertical ranges for values.
linetype – Controls the style of lines (solid, dashed).
linewidth – Sets thickness of lines.
list – Data structure storing objects in R.
lm – Function for linear regression models.
loess – Local regression smoothing method.
Lollipop chart – Combination of bars and dots for comparison.
Longitude – Geographic coordinate east-west.
low – Refers to small or lower values.
lubridate – Package for handling dates/times.
lwd – Line width parameter in base R graphics.
main – Sets the main title in plots.
Map – Visual representation of geographic data.
mapping – Aesthetic mapping of data to visual properties.
margin – Space around plot panels.
Marginal plot – Side plots showing distributions.
match – Finds matching elements in vectors.
Max – Returns the maximum value.
mean – Average value of a dataset.
median – Middle value when data is ordered.
message – Displays a message during execution.

method – A specific procedure or function.

mid – Middle or center position/value.

Min – Returns the minimum value.

Mirror bar – Bar chart mirrored around an axis.

model – Mathematical representation of data.

mult – Refers to multiplication or multiple values.

multiple bar graph – Bar chart comparing groups side by side.

Multiple histogram chart – Several histograms shown together.

Multivariate analysis – Analysis of multiple variables simultaneously.

mutate – Creates or transforms columns in a dataset.

Network chart – Visualization showing connections (nodes/edges).

Node – A point in a network graph.

nodes – Multiple points in a network graph.

Normal distribution – Bell-shaped probability distribution.

Normality – Data following a normal distribution.

nrow – Returns number of rows in a dataset.

nudge_y – Shifts a plotted object vertically.

Null – Represents absence of a value.

observation – A single recorded data point.

opener – Opening line, statement, or element.

options(scipen = 999) – Prevents scientific notation in R output.

Ordered tables – Tables where values follow a set order.

Outlier – A value that is unusually far from others.

outlier.color – Sets color for outliers in plots.

outlier.shape – Sets shape for outliers in plots.

outlier.size – Sets size for outliers in plots.

overlap – When plot elements cover each other.

Package – Collection of R functions and datasets.

padding – Space between elements in a chart.

palette – Set of colors used in plots.

panel.background – Controls background of plot panel.

panel.border – Border around plot panel.

panel.grid – Adds grid lines to plot background.

panel.grid.major.x – Major vertical grid lines.

Parallel coordinates – Chart for multivariate data comparison.

paste – Combines strings with a separator.

paste0 – Combines strings without separator.

patchwork – Package to combine ggplots.

Path chart – Shows sequential movement/flow.

Pattern – Repeated decorative or visual element.

pch – Point shape parameter in base R plots.

percent – Value expressed as a fraction of 100.

Percentage count chart – Chart showing proportions by percentage.

Pie chart – Circular chart divided into slices.

pie3D – 3D version of a pie chart.

PieDonut – Combination of pie chart and donut chart.

pivot – Reshapes data from wide to long (or vice versa).

Plot – Graphical display of data.

plotly – An R package for creating interactive, web-based charts and visualizations.

plotrix – An R special package for plotting enhanced and customized visualizations.

Pmax – Parallel maximum across vectors.

Pmin – Parallel minimum across vectors.

point – A plotted coordinate in a chart.

Polygon – Shape defined by multiple connected points.

Position – Placement of elements in a chart.

Probability distribution – A mathematical function that describes how the values of a random variable are spread or likely to occur.

Proportion – Part relative to the whole.

Pyramid – Chart shaped like a triangle, often for populations.

Quantile – Divides data into equal-sized intervals.

Quantile-Quantile line – Reference line in QQ plots.

Quantile-Quantile plot – Compares dataset distribution to theoretical distribution.

Quantiles – Specific cut points dividing data intervals.

quiet – Suppresses output or messages.

R programming – Statistical computing language.

Radar – Chart showing multivariate data in radial form.

Radarchart – Function for radar/spider plots.

radius – Distance from center to edge in circular plots.

range – Difference between max and min values.

rank – Function that assigns ranking order.

Rank – Position of a value relative to others.

ranking – Process of ordering values.

raster – Spatial data in grid form.

Raster map – Map created from raster data.

rasterize – Converts vector data to raster format.

rbind – Combines datasets by rows.

regex – Pattern matching using regular expressions.

Relationship – Connection between two or more variables.

remove – Deletes variables or objects.

rename – Changes names of variables.

rep – Repeats elements of a vector.

repel – Prevents overlapping text labels.

replace – Substitutes values with others.

Ribbon – Plot area between two lines.

Ribbon chart – Visualization showing filled bands.

right – Refers to right alignment/position.

round – Rounds numbers to specified digits.

row_number – Returns row index for each observation.

rownames – Names assigned to dataset rows.

Rug – Small ticks showing distribution along an axis.

Sample – Randomly selects data points.

Sankey – Flow diagram showing proportions between categories.

scale – Defines mapping from data to aesthetics.
Scatterplot – Plot showing relationship between two variables.
Season – Periodic pattern in time series data.
Segmented chart – Chart divided into parts.
select – Picks columns from a dataset.
sep – Separator used in string functions.
seq – Generates a sequence of numbers.
set.seed – Ensures reproducible random results.
setName – Assigns names to objects.
SetNodeStyle – Sets appearance of nodes in graph packages.
sf – Simple Features package for spatial data.
shadow – Adds shaded effect to plot objects.
Shape – Form or outline of a graphical object.
Shapefile – Common geospatial data format.
Size – Dimension or magnitude of an element.
Skewness – Measure of data asymmetry.
slice – Extracts specific rows from a dataset.
Slope – Rate of change between two points.
Slope chart – Compares values across two time points.
Smoothed line – Curve fitted to show trend.
Sorted bar chart – Bar chart with ordered bars.
Sparse – Data with many missing/zero entries.
sparsely – Refers to scattered or infrequent data.
Spatial mapping – Plotting data with geographic context.
Spider – Another name for radar chart.
Spoke – Line radiating from a center point.
Spoke chart – Chart showing radial spokes.
sqrt – Square root function.
Stacked bar – Bar chart with cumulative segments.
start-summary – Beginning of a summarized report.
stat – Computed statistics layer in ggplot2.
Statistics – Science of data collection and analysis.
Step chart – Shows step-like progression of values.
STL – Seasonal-Trend decomposition using Loess.
stop – Stops execution with an error message.
str – Compact structure display of objects.
Strata – Layers or groups within data.
Stratum – Single layer/category in stratified data.
Stream – Flow of continuous data.
Streams – Multiple continuous data flows.
stringsAsFactors – Converts strings to factors in R.
subtitle – Secondary title in plots.
Suffix – Text added at the end of a string.
summarise – Aggregates data into summary statistics.
summary – Provides key information about data.
Sunburst – Circular hierarchical chart.

Symbol – Representation of a variable/object.

Symbol map – Map with symbols representing values.

Symmetrically – Evenly distributed around a center.

Symmetrically distributed – Data balanced around a mean.

Symmetry – Balanced arrangement of values.

tally – Counts number of items in groups.

t-distribution – A probability distribution similar to the normal distribution but with heavier tails, commonly used for small sample sizes or unknown population variance.

Tertile plot – Divides data into three equal groups.

Text chart – Chart emphasizing text as data.

Text repel – Prevents overlapping text labels.

theme – Controls overall appearance of plots.

tidyverse – R package collection for data science.

Tile map – Map composed of small tiles (squares/hex).

time – Temporal dimension of data.

title – Main heading of a plot.

tm – Text mining package in R.

ToDiagrammeRGraph – Converts objects into DiagrammeR graphs.

tolower – Converts text to lowercase.

top – Refers to uppermost position.

Topography – Surface features of land.

TRUE – Logical constant for “true” in R.

Uncertainty – Degree of doubt in results.

ungroup – Removes grouping structure from a dataset.

unit – A measurement function in R (from the grid package) used to define lengths, widths, or spacing in graphics.

Univariate – Refers to analysis or visualization involving a single variable.

Univariate distribution – The probability distribution or observed spread of values for one variable, often visualized with histograms or density plots.

values – The actual numerical or categorical data points that a variable takes.

vars – Short for “variables”; in R functions, often used to specify multiple columns.

varwidth – An option in `geom_boxplot()` that scales box widths in proportion to the number of observations in each group.

Vertical count chart – A bar-like visualization that shows counts of occurrences along a vertical axis.

Vertical errorbar – Error bars extending up and down from a central value, showing uncertainty or variation.

Violin – A violin plot combines a boxplot with a rotated density plot, showing distribution and spread.

Visual – Refers to anything graphical or pictorial in data representation.

Visualization – The process of creating graphical representations of data to highlight patterns, relationships, and insights.

vjust – Vertical justification of text in plots, controlling alignment above, below, or on a reference point.

warning – A message in R that signals a potential issue in code execution but doesn’t stop the process.

Waterfall – A chart type showing sequential increases and decreases, often used in finance to display contributions to a total.

where – Logical condition or location in data; in SQL or tidyverse contexts, used for filtering rows.

width – Aesthetic parameter in geoms that controls thickness of bars, lines, or boxes.

window – A moving or rolling range used in time-series or spatial analysis.

Wrap – Refers to breaking text into multiple lines or arranging panels in faceted plots (facet_wrap).

xend – The ending x-coordinate for geoms like geom_segment or geom_curve.

xintercept – A vertical line position in geom_vline() where the line crosses the x-axis.

xlab – Function to set or modify the x-axis label.

xmax – The maximum x-coordinate of a graphical object.

xmin – The minimum x-coordinate of a graphical object.

x-xis – Common misspelling of x-axis.

y-axis – The vertical axis in a chart, typically representing values or measurements.

yend – The ending y-coordinate for geoms like geom_segment.

yintercept – A horizontal line position in geom_hline() where the line crosses the y-axis.

ylab – Function to set or modify the y-axis label.

ylim – Function to set limits for the y-axis.

ymax – The maximum y-coordinate of a graphical object.

ymin – The minimum y-coordinate of a graphical object.

Zone – A defined spatial or categorical area, often used in geospatial data.

Zones – Multiple defined spatial or categorical areas, e.g., ecological or administrative divisions.

INDEX

.groups, 64, 66, 68, 70, 131, 135, 152, 189, 215

.predicate, 183, 186

3D pie chart, vii, 47, 48, 49, 227

abbr, 129

Absolute, 16, 31, 32, 135, 151, 157

accuracy, 7, 64, 73, 78, 79, 109, 163, 179

aes, 12, 14, 15, 16, 17, 19, 20, 22, 23, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 43, 44, 45, 46, 50, 51, 53, 55, 57, 58, 65, 66, 68, 70, 71, 73, 75, 78, 79, 80, 84, 87, 88, 90, 94, 96, 98, 99, 100, 101, 103, 104, 105, 106, 107, 108, 109, 110, 112, 113, 115, 116, 117, 119, 121, 123, 129, 131, 134, 135, 136, 137, 139, 141, 143, 145, 146, 148, 149, 150, 152, 154, 156, 158, 160, 164, 165, 167, 168, 169, 171, 172, 174, 175, 177, 181, 189, 192, 196, 199, 202, 204, 206, 207, 208, 209, 210, 211, 212, 213, 220, 221, 223, 224, 226, 227

after_stat, 14, 15, 16, 17, 19, 31, 32, 33, 181, 189, 192

Algorithm, 92, 198, 227

Alluvia, 188, 190, 193

Alluvial chart, 191, 192, 227

alpha, 11, 14, 16, 17, 19, 20, 22, 23, 25, 26, 28, 29, 90, 99, 100, 101, 104, 120, 121, 122, 123, 143, 145, 146, 149, 156, 158, 175, 180, 181, 189, 192, 201, 202, 220, 224, 227

Analysis, v, xiii, xiv, 6, 7, 13, 14, 22, 42, 46, 54, 63, 69, 86, 116, 124, 127, 128, 131, 134, 135, 139, 153, 162, 166, 170, 172, 173, 188, 200, 217, 218, 221, 223, 225, 227, 228, 230, 231, 232, 233, 235, 236, 237

Angle, 53, 59, 107, 108, 109, 110, 129

annotate, 78, 79, 105, 106, 194, 195, 196, 223, 227

Anomalies, 1, 2, 4, 63, 95, 101, 103, 113, 201, 222, 227

Area map, 227

Arrange, 43, 53, 84, 87, 158, 160, 225, 226, 227

arrow, 106, 194, 195, 196, 197, 199

as.character, 53, 163, 167, 189, 191

as.data.frame, 177

as.Date, 101, 104, 129, 131

as.numeric, 71, 75, 126, 128, 163, 167

as_labeller, 115

as_tibble, 82

as_tsibble, 88, 131

assumption, 227

asymmetry, 15, 22, 227, 235

autoplot, 91, 92, 93

axis, 3, 4, 11, 12, 15, 19, 21, 23, 27, 28, 31, 53, 59, 63, 64, 66, 69, 70, 71, 72, 75, 77, 78, 79, 80, 82, 83, 84, 88, 91, 99, 101, 103, 106, 107, 108, 113,

119, 120, 128, 129, 133, 137, 138, 139, 144, 148, 156, 158, 159, 161, 188, 189, 192, 195, 206, 217, 219, 220, 221, 223, 225, 226, 227, 233, 234, 236, 237

axis.line, 11, 158

axis.text, 11, 12, 59, 78, 79, 129

axis.text.x, 11, 12, 59, 78, 79, 129

axis.text.y, 11, 12

axis.ticks, 11, 12

axis.title, 11, 12, 159

axis.title.y, 11, 12

axistype, 219

background, xiii, 8, 11, 12, 45, 46, 50, 53, 58, 59, 78, 81, 82, 88, 89, 91, 93, 94, 96, 98, 101, 104, 112, 159, 177, 195, 199, 227, 229, 233

background.circle.colour, 82, 227

band, iii, 87, 117, 140, 141, 144, 146, 147, 190, 193, 205, 212, 227

bar, ii, iii, iv, vii, viii, ix, 2, 3, 6, 7, 8, 13, 32, 42, 43, 45, 51, 52, 53, 54, 55, 56, 58, 59, 60, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 77, 81, 133, 134, 135, 136, 137, 138, 139, 140, 151, 152, 153, 155, 160, 191, 227, 228, 229, 233, 235, 236

Bar graph, ii, 3, 42, 52, 65, 66, 67, 68, 136, 152, 227, 233

bars, 3, 6, 12, 13, 14, 17, 29, 32, 33, 42, 53, 54, 55, 63, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 75, 77, 78, 79, 80, 84, 133, 134, 135, 136, 137, 138, 139, 140, 151, 152, 153, 156, 160, 209, 211, 227, 229, 230, 231, 232, 235, 236, 237

base_size, 129, 158, 189

benchmark, 38, 155

binaxis, 29, 156

bind, 53

bins, 13, 15, 156, 188, 205, 206, 207, 208, 209, 211, 213, 221, 224, 227, 229, 230

binwidth, 13, 14, 15, 16, 17, 19, 156, 209, 210, 227

Bivariate, 103, 204, 222, 227

Bivariate relationship, 227

Bold, 4, 11, 17, 47, 73, 81, 84, 88, 94, 96, 98, 104, 107, 112, 114, 118, 126, 128, 129, 158, 159, 169, 183, 186, 187, 189, 192, 194, 195, 196, 199, 214, 227, 229

bottom, 35, 62, 81, 88, 119, 161, 182, 184

Bound, 139, 144, 147

Bounds, 133, 135, 137, 138, 146, 149

box, i, vii, 11, 13, 21, 22, 24, 25, 26, 102, 123, 131, 133, 134, 142, 164, 168, 175, 194, 195, 197, 205, 226, 236

box.padding, 164, 168

boxes, 22, 194, 195, 197, 205, 237

Boxplot, 6, 8, 21, 22, 23, 25, 26, 156, 226, 227, 229, 236

breaks, 45, 53, 54, 73, 75, 115, 122, 134, 135, 146, 152, 154, 158, 189, 191

Bubble chart, iii, v, ix, x, 3, 119, 122, 123, 217, 219, 220, 227

Bump chart, iv, ix, 151, 157, 158, 159, 227, 231

Butterfly chart, ii, viii, 72, 73, 74, 227

calc_total, 59

Candlestick chart, 86, 130, 131, 227

caption, 17, 75, 76, 88, 90, 91, 93, 94, 96, 98, 99, 100, 101, 104, 112, 117, 120, 164, 168, 227

Cartogram map, 227, 231

cartogram_cont, 214, 215

case_when, 14, 64, 74, 75, 144, 202, 215

categorical variable, 64, 227, 228, 229

cellsize, 174, 179

Central, ii, viii, 12, 13, 15, 21, 22, 36, 42, 69, 70, 71, 72, 73, 77, 80, 81, 133, 134, 135, 137, 139, 141, 142, 144, 146, 147, 149, 150, 176, 198, 199, 203, 226, 227, 228, 229, 236

Central axis, 69, 72, 80

Change, ii, 4, 5, 6, 11, 71, 86, 90, 97, 98, 101, 110, 120, 130, 141, 191, 227, 235

chart, i, ii, iii, iv, v, vii, viii, ix, x, xii, 2, 3, 5, 6, 7, 8, 11, 12, 13, 15, 17, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 77, 79, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 119, 120, 122, 123, 125, 127, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 157, 158, 159, 160, 161, 162, 166, 176, 178, 184, 189, 190, 191, 192, 193, 194, 197, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 214, 218, 219, 220, 223, 224, 227, 228, 230, 231, 232, 233, 234, 235, 236, 237

Chloropleth, iv, x, 169, 170, 171, 227

closer, 16, 18, 20, 27, 28, 127, 131, 132, 211, 227

Cluster, 27, 28, 32, 38, 103, 129, 139, 143, 156, 165, 180, 203, 204, 205, 206, 209, 211, 220, 222, 227, 228

Clusters, 20, 26, 85, 102, 122, 124, 173, 174, 176, 178, 198, 201, 203, 207, 208, 209, 211, 218, 220, 222, 227

Code, xii, 8, 19, 26, 27, 58, 61, 66, 72, 82, 119, 130, 170, 173, 176, 181, 182, 189, 198, 220, 221, 225, 227, 230, 236

col, 8, 32, 46, 48, 49, 53, 77, 78, 79, 120, 160, 179, 180, 182, 183, 184, 186, 215, 229

colnames, 219

Color, 3, 4, 5, 7, 11, 12, 13, 14, 16, 20, 21, 22, 23, 25, 26, 28, 29, 34, 35, 38, 39, 40, 43, 44, 45, 46, 47, 48, 50, 53, 56, 57, 58, 59, 60, 62, 64, 65, 66, 67,

68, 70, 71, 73, 74, 76, 78, 79, 80, 81, 82, 84, 86,
 87, 88, 91, 94, 96, 98, 99, 100, 101, 102, 103,
 104, 105, 106, 107, 108, 109, 110, 112, 113, 114,
 115, 116, 117, 119, 120, 121, 122, 123, 124, 125,
 126, 128, 129, 130, 131, 133, 134, 136, 137, 139,
 140, 141, 143, 144, 145, 146, 148, 149, 150, 152,
 154, 156, 158, 159, 164, 165, 166, 167, 169, 170,
 171, 172, 174, 175, 177, 179, 181, 182, 183, 184,
 186, 189, 192, 195, 199, 201, 204, 206, 207, 208,
 209, 211, 212, 213, 214, 216, 220, 221, 222, 227,
 228, 230, 233
 Color gradient, 4, 62, 109, 166, 201, 222, 227, 230
 Comparison, ii, 6, 13, 16, 17, 43, 53, 63, 64, 66, 69,
 72, 80, 82, 83, 84, 85, 86, 108, 115, 151, 153,
 161, 176, 200, 217, 218, 223, 226, 227, 232, 233
 components, 6, 11, 12, 42, 53, 57, 91, 110, 130, 228
 Composition, i, 6, 42, 43, 46, 47, 49, 51, 55, 56, 60,
 111, 228
 Compound bar, ii, viii, 68, 69, 228
 concentration, iv, 6, 20, 28, 71, 72, 166, 170, 174,
 201, 202, 203, 204, 205, 206, 207, 209, 210, 212,
 213, 216, 221, 222, 227, 228, 229
 Concentric, 57
 Concentric rings, 57
 Condense, 21, 64, 189
 Confidence interval, iii, ix, 21, 92, 116, 119, 121,
 133, 135, 138, 139, 140, 141, 142, 144, 146, 149,
 150, 228
 Contour, v, x, 6, 8, 201, 203, 204, 211, 212, 228, 229
 Contour chart, 228
 contrast, 4, 31, 33, 48, 50, 77, 85, 86, 99, 109, 115,
 117, 130, 138, 149, 190, 202, 216
 coord_cartesian, 208
 coord_flip, 66, 67, 69, 70, 72, 73, 75, 84, 160
 coord_sf, 174, 175, 177, 181
 cor, 121, 124, 126
 Correlation, iii, ix, 6, 119, 120, 121, 124, 125, 126,
 127, 128, 228, 230, 231
 Correlation matrix, iii, ix, 124, 125, 126, 127, 228,
 230
 count, i, vii, 8, 14, 15, 16, 17, 19, 29, 30, 31, 32, 33,
 43, 44, 45, 46, 48, 51, 53, 70, 73, 87, 88, 94, 95,
 97, 99, 110, 157, 169, 173, 174, 177, 179, 188,
 189, 191, 192, 207, 211, 215, 216, 228, 229, 230,
 233, 236
 create_layout, 198
 Crowding, 51, 101, 211, 228
 Crs, 163, 166, 167, 170, 174, 175, 177, 179, 181,
 182, 183, 185, 186, 214, 215, 228
 cut, 36, 45, 53, 122, 134, 135, 146, 152, 158, 188,
 189, 191, 234
 cycle, 126, 128
 data, , i, xii, xiii, xiv, 1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 13,
 14, 15, 16, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28,
 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
 42, 43, 44, 46, 47, 48, 51, 53, 55, 56, 57, 60, 61,
 63, 64, 65, 66, 71, 73, 74, 75, 77, 78, 79, 82, 83,
 84, 86, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,
 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
 109, 110, 111, 113, 114, 116, 118, 119, 120, 121,
 122, 124, 125, 126, 127, 128, 129, 130, 131, 132,
 133, 135, 137, 138, 139, 140, 142, 143, 144, 146,
 147, 148, 149, 151, 153, 154, 156, 157, 158, 159,
 162, 163, 164, 165, 166, 167, 168, 169, 170, 171,
 172, 173, 174, 175, 176, 177, 178, 179, 181, 182,
 185, 188, 189, 191, 195, 198, 201, 202, 203, 204,
 205, 206, 207, 208, 209, 210, 211, 212, 213, 214,
 217, 219, 221, 223, 225, 226, 227, 228, 229, 230,
 231, 232, 233, 234, 235, 236, 237
 data.frame, 34, 35, 37, 53, 61, 71, 126, 128, 154, 177,
 195, 198
 dataset, vi, 3, 4, 7, 12, 13, 14, 17, 18, 20, 21, 23, 29,
 31, 32, 33, 35, 36, 42, 43, 44, 46, 58, 59, 71, 74,
 75, 77, 78, 79, 82, 86, 88, 89, 90, 95, 99, 100,
 101, 102, 105, 113, 120, 122, 124, 125, 126, 127,
 128, 129, 130, 133, 153, 158, 160, 161, 165, 166,
 167, 171, 177, 185, 191, 201, 202, 205, 206, 210,
 218, 219, 220, 221, 222, 223, 224, 225, 226, 228,
 232, 233, 234, 235, 236
 date, 101, 104, 128, 129, 131, 228
 Decomposed time series, ii, viii, 91, 92, 228
 default.unit, 11, 113, 123, 134, 136, 137, 140, 141,
 143, 145, 146, 148, 150, 152, 157
 Dense, xii, 131, 156, 165, 174, 194, 208, 209, 211,
 222, 228
 Densely, 201
 Density, iv, x, xi, 5, 6, 8, 13, 22, 26, 27, 28, 46, 119,
 120, 142, 143, 144, 156, 169, 173, 176, 178, 180,
 181, 201, 202, 203, 204, 205, 206, 207, 211, 212,
 213, 214, 221, 222, 224, 225, 226, 227, 228, 229,
 230, 231, 236
 Dependencies, 60, 126
 desc, 43, 53, 158, 160
 discrete, 23, 25, 28, 30, 47, 55, 63, 64, 97, 99, 156,
 189, 192, 228
 Dispersion, 120, 139, 146
 dist, 82, 83, 170, 175
 Distribution, i, 6, 13, 14, 15, 16, 18, 19, 20, 22, 23,
 24, 25, 26, 27, 29, 31, 33, 34, 35, 36, 37, 38, 39,
 52, 59, 64, 74, 78, 81, 142, 143, 144, 147, 156,
 162, 164, 165, 166, 169, 170, 175, 176, 191, 193,
 200, 201, 202, 205, 210, 213, 216, 224, 225, 226,
 227, 228, 229, 230, 233, 234, 236
 Divergent chart, 228
 dodge, 17, 66, 68, 70, 154, 228
 Donut, ii, vii, 6, 42, 47, 50, 51, 228, 233
 Dot, vii, ix, 28, 29, 30, 31, 60, 61, 63, 119, 151, 153,
 155, 156, 157, 228
 Dot plot, vii, ix, 28, 29, 63, 151, 155, 156, 157, 228
 dotsize, 29, 156
 dplyr, 14, 75, 78, 158, 160, 163, 166, 198, 227, 228,
 231

Drift, 118
 drop_na, 75, 94
 Dumbbell plot, 106
 edge_colour, 199
 element_blank, 11, 12, 59, 129, 158
 element_line, 11, 78, 80, 158
 element_rect, 11, 12, 58, 81, 88, 91, 93, 94, 96, 98, 104, 112, 159, 189, 192, 199
 element_text, 11, 12, 17, 59, 78, 79, 81, 88, 94, 96, 98, 104, 112, 118, 126, 129, 158, 159, 189, 192, 196, 199
 encoding, 5, 149, 170, 179, 183, 186, 215, 222, 231
 end_cap, 199
 epsg, 163, 182, 185
 error, iii, ix, 6, 23, 25, 64, 65, 66, 67, 68, 69, 92, 93, 133, 134, 135, 136, 137, 138, 139, 140, 141, 144, 145, 149, 151, 152, 228, 229, 231, 235, 236
 error bar, iii, ix, 6, 64, 65, 66, 67, 68, 69, 133, 134, 135, 136, 137, 138, 139, 151, 152, 228, 229, 231, 236
 Error_Margin, 64, 66, 68, 135, 152
 ETS, 92, 93, 228
 expand, 11, 17, 19, 23, 39, 40, 55, 57, 59, 65, 67, 68, 70, 87, 100, 106, 109, 113, 122, 123, 134, 136, 137, 140, 141, 143, 145, 148, 150, 152, 156, 189, 192, 208
 explode, 48, 49, 51
 Exploded pie, ii, vii, 43, 48, 49
 explodeDonut, 51
 Exploratory, 4, 13, 113, 128, 153, 227, 228
 Exploratory analysis, 153, 228
 extent, 149, 171, 177
 Face, 11, 17, 70, 81, 88, 94, 96, 98, 104, 112, 118, 126, 129, 158, 159, 189, 192, 194, 196, 199, 228
 facet, xi, 19, 58, 59, 114, 115, 117, 119, 223, 224, 225, 228, 231, 237
 facet_grid, 19, 228
 facet_wrap, 19, 58, 114, 115, 117, 119, 224, 228, 237
 factor, 14, 15, 16, 58, 64, 70, 71, 82, 84, 128, 129, 145, 153, 158, 160, 183, 186, 224, 228, 231
 FALSE, 61, 71, 116, 117, 119, 120, 121, 179, 183, 186, 198, 208, 215
 Fan chart, iii, ix, 6, 133, 146, 147, 228
 fct_reorder, 153, 154
 fct_rev, 73, 129
 Figure, vii, viii, ix, x, xi, 15, 16, 18, 19, 21, 22, 24, 25, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 38, 40, 41, 44, 45, 47, 48, 49, 50, 52, 54, 56, 57, 58, 60, 62, 65, 67, 69, 70, 72, 73, 77, 81, 83, 85, 87, 89, 90, 92, 93, 95, 97, 98, 99, 100, 102, 103, 104, 106, 107, 108, 109, 111, 112, 114, 115, 117, 118, 119, 120, 121, 123, 125, 127, 129, 132, 134, 136, 138, 140, 142, 143, 144, 145, 147, 148, 150, 152, 153, 155, 157, 159, 160, 164, 165, 168, 169, 170, 171, 173, 174, 176, 178, 180, 181, 184, 187, 190, 193, 197, 199, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 216, 218, 219, 220, 222, 223, 224, 225, 227, 228
 File, 162, 166, 198, 228
 fill, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 43, 44, 45, 46, 47, 49, 50, 53, 55, 56, 57, 58, 59, 64, 65, 66, 67, 68, 70, 71, 73, 75, 76, 78, 79, 81, 86, 87, 99, 100, 101, 102, 103, 104, 105, 106, 108, 109, 110, 116, 117, 119, 123, 128, 129, 131, 134, 135, 136, 137, 140, 141, 143, 145, 146, 148, 150, 151, 152, 154, 156, 157, 160, 164, 165, 167, 169, 170, 171, 172, 174, 175, 177, 179, 180, 181, 186, 189, 192, 194, 195, 201, 202, 204, 209, 213, 221, 224, 228
 fill_by_sign, 59
 filter, 53, 78, 79, 163, 166, 167, 170, 175, 177, 179, 181, 182, 183, 185, 186, 214, 215, 228
 First, 4, 14, 21, 23, 44, 52, 128, 131, 165, 170, 171, 173, 177, 188, 189, 205, 226
 fixed, 48, 135, 176, 177, 182, 185, 228
 Flip, 15, 66, 67, 69, 70, 72, 73, 75, 84, 160
 Flips, 64, 66, 74, 214
 floor, 71, 126, 128, 177
 Flow, iv, x, 5, 6, 62, 89, 95, 99, 101, 110, 188, 189, 193, 194, 197, 199, 200, 223, 227, 228, 231, 233, 234, 235
 Flow chart, x, 194, 197, 228
 Fluctuate, 138
 Font size, 4, 11, 228
 font.radar, 82
 fontface, 73, 169, 183, 186, 187, 195, 196, 199, 229
 forcats, 78, 128, 153, 154, 229, 231
 forecast, 92, 93, 146, 228, 229
 frequency, 4, 11, 13, 14, 16, 17, 18, 19, 20, 29, 30, 31, 64, 156, 157, 191, 192, 193, 201, 206, 209, 213, 214, 221, 222, 226, 228, 229, 230
 generate_dot, 60, 61
 geography, 5, 178, 214, 216, 229
 geom, 8, 12, 14, 15, 16, 17, 19, 20, 22, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 43, 44, 45, 46, 47, 49, 50, 53, 55, 56, 57, 58, 64, 65, 66, 68, 70, 71, 72, 73, 75, 76, 77, 78, 79, 80, 84, 86, 87, 88, 90, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 115, 116, 117, 119, 120, 121, 122, 123, 128, 129, 130, 131, 133, 134, 135, 136, 137, 138, 139, 140, 141, 143, 144, 145, 146, 148, 149, 150, 151, 152, 153, 154, 156, 158, 160, 163, 164, 165, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 181, 189, 191, 192, 194, 195, 196, 198, 199, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 220, 223, 224, 226, 229, 230, 236, 237
 geom_alluvium, 189, 191, 192, 229
 geom_area, 8, 86, 87, 229

geom_bar, 8, 53, 55, 58, 64, 65, 66, 68, 70, 71, 73, 75, 135, 151, 152, 229
 geom_bin2d, 8, 205, 206, 208, 210, 229
 geom_bump, 158, 229
 geom_col, 8, 46, 53, 77, 78, 79, 160, 229
 geom_col_pattern, 46, 229
 geom_count, 8, 30, 31, 32, 33, 229
 geom_crossbar, 8, 139, 229
 geom_density, 8, 201, 202, 203, 204, 211, 212, 224, 229
 geom_density_2d, 8, 203, 204, 211, 212, 229
 geom_density_2d_filled, 8, 204, 212, 229
 geom_dotplot, 8, 28, 29, 156, 229
 geom_edge_link, 198, 199, 229
 geom_errorbar, 8, 64, 65, 66, 68, 133, 134, 135, 136, 137, 138, 151, 152, 229
 geom_errorbarh, 8, 137, 138, 229
 geom_hex, 8, 207, 229
 geom_histogram, 8, 14, 15, 16, 17, 19, 209, 224, 229
 geom_hline, 8, 15, 16, 70, 71, 72, 73, 75, 76, 154, 229, 237
 geom_image, 163, 164, 167, 198, 199, 229
 geom_jitter, 8, 22, 23, 25, 26, 28, 29, 30, 143, 156, 229
 geom_label, 8, 40, 45, 46, 47, 49, 50, 229
 geom_label_repel, 40, 229
 geom_line, 8, 84, 87, 88, 90, 96, 104, 113, 115, 133, 134, 146, 148, 229
 geom_linerange, 8, 148, 229
 geom_map, 8, 229
 geom_node_point, 198, 199, 229
 geom_node_text, 198, 199, 229
 geom_path, 8, 100, 111, 112, 229
 geom_point, 8, 20, 22, 27, 34, 35, 39, 40, 84, 95, 96, 98, 99, 100, 101, 103, 105, 106, 108, 109, 116, 119, 120, 121, 122, 123, 133, 134, 137, 141, 149, 150, 153, 154, 158, 181, 220, 223, 226, 229
 geom_pointrange, 8, 149, 150, 229
 geom_qq_line, 8, 37, 38, 229
 geom_quantile, 8, 34, 35, 229
 geom_raster, 8, 176, 177, 229
 geom_rect, 8, 43, 44, 45, 50, 102, 103, 104, 130, 131, 175, 176, 177, 229
 geom_ribbon, 8, 144, 145, 146, 229
 geom_rug, 8, 27, 229
 geom_segment, 8, 94, 106, 130, 131, 153, 154, 194, 195, 196, 229, 237
 geom_sf, 8, 163, 164, 165, 167, 169, 170, 171, 172, 173, 174, 175, 177, 181, 229, 230
 geom_sf_text, 8, 165, 169, 170, 171, 230
 geom_smooth, 8, 34, 116, 117, 119, 121, 140, 141, 144, 145, 230
 geom_spoke, 8, 107, 108, 109, 230
 geom_step, 8, 97, 98, 99, 230
 geom_stratum, 189, 191, 192, 230
 geom_stream, 110, 230
 geom_text, 8, 39, 40, 43, 44, 73, 75, 80, 95, 96, 158, 164, 167, 168, 189, 191, 192, 230
 geom_tile, 8, 128, 129, 173, 174, 230
 geom_treemap, 56, 57, 230
 geom_treemap_text, 56, 57, 230
 geom_violin, 8, 26, 143, 230
 geom_vline, 8, 101, 230, 237
 Geospatial analysis, 170, 230
 Geospatial heatmap, iv, x, 178, 180, 181, 230
 ggArrange, 230
 ggcorr, 126, 230
 ggcorrplot, 121, 124, 230, 231
 ggMarginal, 226, 230
 ggparcoord, 217, 230
 ggplot, 12, 14, 15, 17, 19, 20, 22, 23, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 43, 44, 45, 46, 50, 53, 55, 57, 58, 64, 65, 66, 68, 70, 71, 73, 75, 78, 79, 80, 84, 87, 88, 90, 94, 96, 98, 99, 100, 101, 103, 104, 105, 106, 107, 109, 110, 112, 113, 115, 116, 117, 119, 121, 123, 129, 131, 133, 134, 135, 137, 139, 141, 143, 145, 146, 148, 149, 152, 154, 156, 158, 160, 163, 164, 165, 167, 169, 171, 172, 174, 175, 177, 181, 189, 192, 195, 202, 204, 206, 207, 208, 209, 210, 211, 212, 213, 220, 221, 223, 224, 226, 230
 ggplot2, 7, 8, 11, 12, 14, 19, 25, 36, 49, 75, 78, 89, 106, 120, 122, 128, 146, 149, 151, 158, 160, 163, 167, 194, 201, 220, 221, 224, 225, 227, 228, 229, 230, 231, 232, 235
 ggtheme, 124, 230
 ggtitle, 11, 14, 16, 17, 44, 45, 47, 50, 53, 126, 196, 199, 230
 github, 230
 Gradient, 62, 109, 110, 120, 124, 128, 129, 169, 174, 177, 178, 213, 214, 221, 222, 227, 230
 graph, i, 2, 3, 34, 40, 52, 61, 65, 67, 68, 92, 95, 136, 152, 198, 199, 227, 229, 230, 231, 232, 233, 235
 graph_from_data_frame, 198
 graphics, i, v, xii, 3, 7, 8, 11, 12, 116, 119, 188, 223, 230, 231, 232, 236
 grep, 8, 230
 Grid, 19, 45, 53, 78, 79, 80, 82, 87, 90, 119, 120, 125, 127, 128, 129, 158, 173, 174, 176, 177, 178, 179, 180, 206, 208, 211, 213, 217, 221, 223, 226, 228, 230, 231, 233, 234, 236
 grid.arrange, 53, 226
 grid.min, 82
 Gridded chart, 230
 gridline.max.colour, 82
 gridline.mid.colour, 82
 gridline.min.colour, 82
 group.colours, 82
 group.point.size, 82
 group_by, 45, 53, 64, 66, 68, 70, 71, 73, 74, 75, 78, 82, 84, 131, 135, 137, 144, 146, 152, 158, 188, 189, 191, 215

grViz, 61, 230
 gsub, 61, 230
 gtitle, 230
 guide_legend, 11, 113, 123, 134, 136, 137, 140, 141, 143, 145, 146, 148, 150, 152, 157
 guides, 11, 113, 123, 133, 134, 136, 137, 140, 141, 143, 145, 146, 148, 150, 152, 157, 230
 Heatmap, iii, iv, v, x, xi, 6, 109, 124, 125, 128, 129, 178, 179, 180, 181, 206, 208, 212, 213, 214, 221, 222, 230
 height, 3, 13, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 32, 39, 40, 41, 55, 56, 57, 64, 65, 66, 68, 69, 84, 85, 93, 103, 105, 106, 107, 108, 109, 110, 113, 114, 115, 116, 117, 118, 119, 120, 119, 120, 121, 123, 124, 125, 133, 134, 135, 136, 137, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 152, 156, 157, 158, 159, 176, 179, 202, 230, 231
 heteroscedasticity, 21, 26, 34, 230
 Hex, 8, 182, 183, 207, 229, 230, 236
 Hex chart, 230
 hexagon, 207, 230
 hexagonal, 46, 47, 207, 208, 211, 229, 230, 232
 high, 7, 19, 26, 30, 35, 86, 109, 125, 126, 127, 128, 129, 130, 131, 132, 142, 162, 169, 174, 178, 181, 201, 202, 203, 211, 213, 214, 217, 221, 222, 225, 227, 228, 230, 231
 Histogram, i, v, vii, x, xi, 6, 8, 11, 13, 14, 15, 16, 17, 18, 19, 27, 201, 202, 208, 209, 210, 223, 224, 226, 227, 229, 230, 233
Histogram chart, x, 210, 230, 233
 hjust, 11, 39, 40, 59, 75, 81, 88, 94, 96, 98, 104, 112, 126, 129, 158, 189, 192, 194, 195, 196, 199, 230
 Horizontal count chart, vii, 32, 33, 230
 Horizontal errorbar, iii, 137, 231
 Hotspots, 174, 178, 180, 201, 231
 Icon, 50, 163, 164, 167, 168, 198, 231
 iconv, 179, 183, 186, 215, 231
 if_else, 71, 73, 183
 ignore_case, 179, 182, 185, 215
 Interpretation, xiii, 6, 7, 40, 43, 54, 64, 69, 71, 90, 112, 119, 146, 150, 162, 170, 201, 227, 231
 is.na, 182, 183, 185
 itermax, 215, 231
 jitter, i, 6, 8, 22, 23, 25, 26, 28, 29, 30, 119, 120, 143, 156, 229, 231
 keyheight, 11, 113, 123, 134, 136, 137, 140, 141, 143, 145, 146, 148, 150, 152, 157, 231
 keywidth, 11, 113, 123, 134, 136, 137, 140, 141, 143, 145, 146, 148, 150, 152, 157, 231
 label, 8, 39, 40, 41, 43, 44, 45, 46, 47, 48, 49, 50, 57, 59, 61, 73, 75, 77, 78, 79, 80, 84, 96, 97, 126, 129, 158, 164, 165, 168, 169, 170, 171, 183, 185, 186, 188, 189, 192, 195, 196, 199, 215, 226, 229, 231, 237
 label.padding, 45, 50, 78, 79
 label.size, 45, 50, 78, 79
 label_round, 126
 labelcex, 48, 49, 231
 labeller, 115, 231
 labels, i, ii, vii, viii, 8, 16, 23, 25, 26, 30, 33, 34, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49, 50, 53, 55, 56, 57, 66, 70, 71, 74, 75, 76, 77, 78, 79, 80, 81, 88, 90, 91, 99, 101, 102, 103, 106, 107, 108, 111, 115, 122, 126, 128, 133, 134, 135, 146, 151, 152, 158, 160, 165, 167, 168, 170, 171, 176, 179, 183, 184, 188, 189, 191, 193, 194, 195, 197, 198, 219, 227, 229, 230, 231, 234, 236
 labs, 22, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 55, 57, 58, 59, 64, 65, 67, 68, 70, 71, 73, 76, 82, 84, 87, 88, 90, 91, 93, 94, 96, 98, 99, 100, 101, 103, 104, 105, 106, 108, 109, 110, 112, 113, 115, 116, 117, 119, 121, 123, 129, 131, 134, 136, 137, 140, 141, 143, 145, 146, 148, 150, 152, 154, 156, 158, 160, 164, 165, 168, 169, 171, 172, 174, 175, 177, 181, 189, 192, 202, 204, 206, 207, 208, 209, 210, 211, 212, 213, 217, 220, 221, 223, 224, 226, 231
 Last, 4, 89, 131
 Latitude, 163, 166, 167, 175, 177, 179, 181, 183, 186, 215, 231
 Layer, 5, 7, 12, 22, 56, 84, 98, 107, 122, 137, 144, 151, 156, 165, 169, 170, 179, 182, 185, 206, 231, 235
 lead, 3, 5, 70, 81, 94, 231
 left, 18, 70, 71, 74, 77, 79, 80, 81, 94, 135, 137, 138, 186, 188, 221, 231
 legend, 8, 11, 12, 22, 26, 40, 44, 58, 71, 82, 90, 93, 113, 114, 123, 128, 129, 133, 134, 136, 137, 140, 141, 143, 145, 146, 148, 150, 152, 157, 158, 165, 168, 170, 171, 172, 179, 180, 182, 183, 184, 186, 215, 231
 legend.background, 11, 12
 legend.key, 11, 12, 93
 legend.margin, 11
 legend.outside.position, 184
 legend.position.inside, 11, 93
 legend.text.size, 82
 length.out, 110, 195
 lengths, 63, 109, 173, 174, 179, 195, 231, 236
 levels, xiii, 14, 15, 34, 35, 36, 44, 57, 64, 71, 82, 84, 97, 115, 140, 144, 145, 153, 155, 158, 160, 181, 183, 186, 202, 203, 212, 228, 229, 231
 Library, 2, 8, 14, 48, 53, 59, 61, 78, 80, 88, 110, 119, 121, 126, 128, 154, 158, 163, 182, 185, 189, 191, 198, 207, 214, 217, 219, 223, 226, 231, 232
 library(arules), 231
 library(car), 119, 231
 library(carData), 231
 library(cartogram), 214, 231
 library(data.tree), 61, 231
 library(DiagrammeR), 61, 231
 library(dplyr), 14, 78, 158, 163, 198, 231

library(fable), 88, 231
 library(feasts), 88, 231
 library(fmsb), 219, 231
 library(forcats), 78, 128, 154, 231
 library(ggalluvial), 189, 191, 231
 library(GGally), 126, 217, 231
 library(ggbump), 158, 231
 library(ggcorrplot), 121, 231
 library(ggExtra), 226, 231
 library(ggimage), 163, 198, 231
 library(ggpattern), 231
 library(ggplot2), 8, 14, 78, 158, 163, 231
 library(ggradar), 231
 library(gggraph), 198, 231
 library(ggrepel), 163, 231
 library(ggribbons), 231
 library(grid), 119, 231
 library(hexbin), 207, 232
 library(igraph), 198, 232
 library(magick), 232
 library(patchwork), 78, 88, 223, 232
 library(plotrix), 48, 232
 library(raster), 232
 library(readr), 163, 214, 232
 library(remotes), 232
 library(scales), 78, 232
 library(showtext), 232
 library(stringr), 232
 library(tibble), 232
 library(tidyverse), 14, 182, 185, 232
 library(tmap), 182, 185, 214, 232
 library(treemap), 232
 library(treemapify), 232
 library(viridis), 232
 library(waterfall), 59, 232
 library(webr), 232
 library(writextl), 185, 232
 limits, 43, 78, 79, 120, 139, 147, 149, 189, 192, 219, 232, 237
 line, i, iii, vii, ix, 6, 8, 11, 14, 19, 21, 33, 34, 36, 37, 38, 39, 43, 63, 70, 71, 72, 73, 75, 77, 78, 80, 81, 82, 84, 85, 86, 87, 88, 89, 90, 93, 94, 95, 96, 97, 98, 100, 104, 105, 106, 111, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 128, 130, 133, 134, 139, 140, 141, 144, 145, 146, 147, 148, 149, 153, 154, 155, 156, 158, 184, 211, 217, 218, 229, 232, 233, 234, 235, 237
 Line, i, iii, vii, ix, 6, 8, 11, 14, 19, 21, 33, 34, 36, 37, 38, 39, 43, 63, 70, 71, 72, 73, 75, 77, 78, 80, 81, 82, 84, 85, 86, 87, 88, 89, 90, 93, 94, 95, 96, 97, 98, 100, 104, 105, 106, 111, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 128, 130, 133, 134, 139, 140, 141, 144, 145, 146, 147, 148, 149, 153, 154, 155, 156, 158, 184, 211, 217, 218, 229, 232, 233, 234, 235, 237
 Line chart, iii, ix, 37, 86, 88, 89, 90, 98, 100, 113, 114, 115, 116, 117, 118, 232
 Linerange chart, ix, 148, 232
 linetype, 59, 101, 121, 232
 linewidth, 11, 34, 35, 58, 70, 71, 73, 81, 84, 88, 90, 91, 93, 94, 96, 98, 99, 100, 101, 103, 104, 106, 112, 113, 115, 116, 117, 119, 129, 159, 177, 189, 192, 199, 232
 list, vi, vii, 8, 120, 159, 194, 198, 232
 lm, 119, 121, 140, 141, 232
 loess, 91, 116, 117, 118, 141, 232, 235
 Lollipop chart, iv, ix, 63, 151, 153, 155, 232
 Longitude, 163, 166, 167, 175, 177, 179, 181, 183, 186, 215, 232
 low, 20, 32, 35, 36, 86, 109, 126, 128, 129, 130, 131, 132, 142, 143, 162, 169, 174, 201, 213, 221, 222, 225, 227, 231, 232
 lubridate, 126, 128, 232
 lwd, 120, 182, 183, 184, 186, 232
 main, xii, 8, 15, 26, 27, 48, 49, 52, 53, 120, 139, 140, 170, 194, 223, 227, 230, 232, 236
 Map, iv, viii, ix, x, 6, 8, 20, 57, 113, 129, 133, 156, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 187, 188, 197, 198, 203, 204, 211, 214, 215, 216, 217, 227, 229, 231, 232, 234, 236
 mapping, iv, ix, x, 43, 53, 99, 100, 110, 122, 133, 162, 164, 165, 166, 168, 173, 174, 182, 183, 185, 227, 228, 232, 235
 margin, 11, 12, 64, 66, 68, 133, 135, 136, 145, 149, 152, 186, 232
 Marginal plot, v, 225, 226, 231, 232
 match, 163, 166, 182, 199, 214, 232
 Max, 31, 32, 33, 78, 79, 80, 82, 95, 96, 131, 146, 158, 164, 168, 219, 232, 234
 max.overlaps, 164, 168
 mean, 23, 25, 26, 34, 35, 64, 66, 68, 71, 82, 133, 135, 136, 137, 138, 139, 140, 144, 145, 146, 147, 151, 152, 153, 154, 155, 220, 227, 232, 236
 median, 13, 21, 22, 24, 26, 35, 232
 message, 3, 5, 7, 183, 185, 186, 223, 232, 235, 236
 method, 32, 63, 94, 116, 117, 119, 121, 124, 140, 141, 173, 176, 221, 224, 227, 228, 232, 233
 mid, 39, 82, 89, 90, 92, 93, 126, 129, 188, 233
 Min, 82, 131, 146, 233, 234
 Mirror bar, ii, viii, 69, 70, 71, 233
 model, 91, 92, 93, 116, 118, 119, 121, 141, 142, 233
 mult, 11, 17, 19, 23, 39, 40, 55, 59, 100, 106, 109, 113, 134, 140, 141, 143, 148, 150, 156, 233
 multiple bar graph, 233
 Multivariate analysis, v, 6, 217, 233
 mutate, 14, 43, 45, 53, 58, 64, 66, 68, 70, 71, 73, 75, 78, 80, 82, 84, 87, 94, 109, 122, 126, 129, 131, 134, 135, 139, 144, 146, 148, 149, 152, 154, 158,

160, 163, 167, 174, 179, 182, 183, 185, 186, 188,
 189, 191, 192, 202, 215, 233
 na.rm, 71, 199
 na.value, 174, 177
 names_from, 126, 192
 nested, 42, 55, 56, 58, 60, 204, 230
 Network chart, iv, x, 198, 199, 233
 Node, 5, 60, 61, 194, 195, 198, 199, 229, 233
 nodes, 188, 194, 198, 199, 229, 230, 233, 235
 normal distribution, 15, 18, 20, 27, 29, 34, 37, 38, 39,
 233, 236
 normality, 15, 16, 18, 20, 21, 24, 26, 27, 28, 29, 34,
 37, 38, 39, 233
 nrow, 19, 53, 110, 117, 182, 183, 185, 186, 233
 nudge_y, 165, 233
 null, 22, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 36, 37,
 38, 39, 40, 53, 55, 58, 65, 67, 68, 73, 76, 103,
 105, 106, 108, 109, 134, 136, 137, 140, 141, 143,
 145, 148, 150, 152, 154, 156, 233
 observation, 20, 105, 106, 119, 122, 147, 149, 217,
 233, 234
 opener, 233
 options(scipen = 999), 233
 order_by, 96
 Ordered tables, iv, ix, 151, 160, 233
 Outlier, 22, 23, 25, 26, 105, 143, 233
 outlier.color, 23, 25, 233
 outlier.shape, 22, 23, 25, 26, 143, 233
 outlier.size, 23, 25, 233
 overlap, 17, 19, 21, 22, 24, 40, 65, 67, 115, 120, 133,
 135, 137, 139, 153, 165, 179, 183, 186, 205, 207,
 210, 233
 Overlap, 17, 19, 21, 22, 24, 40, 65, 67, 115, 120, 133,
 135, 137, 139, 153, 165, 179, 183, 186, 205, 207,
 210, 233
 Package, 7, 8, 12, 40, 48, 59, 75, 82, 91, 110, 119,
 122, 126, 153, 166, 167, 219, 221, 225, 226, 228,
 229, 230, 231, 232, 233, 234, 235, 236
 padding, 45, 50, 78, 79, 114, 164, 168, 189, 208, 233
 palette, 57, 102, 151, 156, 215, 216, 233
 panel.background, 11, 58, 59, 233
 panel.border, 189, 192, 233
 panel.grid, 78, 80, 129, 158, 233
 panel.grid.major.x, 78, 80, 233
 Parallel coordinates, v, x, 6, 217, 218, 233
 parsing, 128
 paste, 14, 23, 61, 121, 129, 177, 233
 patchwork, 78, 80, 88, 174, 223, 232, 233
 Path chart, iii, viii, 100, 101, 233
 pattern, 3, 4, 5, 8, 15, 18, 20, 24, 28, 29, 37, 46, 47,
 92, 93, 99, 107, 111, 116, 127, 147, 153, 172,
 181, 218, 227, 228, 229, 233, 234, 235
 pch, 119, 120, 233
 percent, vii, 30, 31, 45, 46, 49, 73, 75, 77, 78, 79, 80,
 233

Pie chart, i, ii, vii, 3, 42, 43, 44, 45, 46, 47, 48, 49,
 50, 51, 52, 53, 54, 227, 228, 233
 pie3D, 48, 49, 233
 PieDonut, 51, 52, 233
 pivot_longer, 191
 pivot_wider, 126, 191, 192
 Plot, i, ii, iii, v, vii, viii, ix, x, 3, 6, 7, 11, 12, 14, 15,
 16, 17, 18, 19, 21, 22, 24, 25, 26, 27, 28, 29, 30,
 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 53, 66, 71,
 79, 80, 81, 82, 84, 87, 88, 89, 90, 91, 92, 93, 94,
 95, 96, 98, 102, 103, 104, 106, 107, 112, 113,
 115, 116, 117, 118, 119, 120, 121, 122, 124, 126,
 128, 129, 130, 137, 140, 141, 142, 148, 155, 156,
 157, 158, 159, 170, 171, 177, 179, 182, 185, 186,
 189, 192, 196, 199, 201, 202, 203, 205, 206, 207,
 208, 209, 211, 212, 213, 214, 215, 217, 218, 219,
 222, 223, 224, 225, 226, 227, 228, 229, 230, 231,
 232, 233, 234, 235, 236
 plot.background, 81, 88, 91, 93, 94, 96, 98, 104, 112,
 159, 199
 plot.caption, 88, 94, 96, 98, 104, 112
 plot.margin, 11
 plot_annotation, 17, 80, 117, 141, 224
 plot_layout, 80
 plotly, 234
 plotrix, 48, 232, 234
 Pmax, 131, 234
 Pmin, 131, 234
 point, i, vii, ix, xi, 3, 8, 11, 12, 20, 21, 22, 23, 25, 27,
 30, 33, 34, 35, 39, 40, 41, 60, 63, 71, 82, 84, 94,
 95, 96, 98, 99, 100, 101, 102, 103, 105, 106, 107,
 108, 109, 110, 116, 118, 119, 120, 121, 122, 123,
 125, 128, 133, 134, 137, 138, 140, 141, 143, 144,
 147, 149, 150, 153, 154, 155, 158, 163, 165, 166,
 167, 168, 170, 172, 175, 181, 194, 197, 198, 199,
 202, 203, 204, 217, 219, 220, 223, 224, 226, 227,
 228, 229, 233, 234, 235, 236
 Polygon, 8, 14, 46, 47, 82, 83, 154, 163, 166, 171,
 172, 181, 218, 219, 230, 234
 Position, 3, 5, 11, 12, 17, 19, 22, 39, 47, 65, 66, 68,
 70, 73, 82, 86, 87, 93, 122, 129, 136, 139, 148,
 151, 152, 154, 157, 158, 165, 170, 171, 175, 184,
 186, 194, 231, 233, 234, 236, 237
 position_dodge, 66, 68, 70, 154
 position_stack, 47, 73
 precision, 7, 38, 126, 138, 140, 149, 151, 153, 156,
 166
 Prediction, 144, 146, 149, 229
 probability, 142, 201, 233, 234, 236
 probability distribution, 201, 233, 234, 236
 proportion, 31, 42, 44, 57, 169, 214, 234, 236
 pyramid, ii, viii, 74, 75, 77, 79, 80, 81, 234
 quantile, i, vii, 8, 34, 35, 36, 37, 38, 229, 234
 Quantile-Quantile line, i, vii, 37, 38, 234
 Quantile-Quantile plot, vii, 36, 37, 234
 quantiles, 34, 35, 36, 234

quiet, 44, 179, 182, 185, 214, 234

radar, ii, v, x, 63, 82, 217, 218, 219, 231, 234, 235

radarchart, 219, 234

radius, 49, 107, 109, 234

range, ix, 2, 8, 13, 18, 21, 22, 25, 35, 36, 39, 47, 64, 67, 68, 69, 92, 102, 103, 119, 120, 122, 123, 130, 132, 133, 134, 135, 136, 137, 138, 139, 140, 144, 145, 146, 147, 148, 149, 150, 157, 201, 203, 206, 208, 209, 210, 211, 214, 227, 228, 229, 232, 234, 237

rank, 158, 159, 160, 227, 229, 234

ranking, iv, 63, 74, 111, 151, 153, 155, 157, 158, 159, 160, 234

raster, iv, v, x, 8, 176, 177, 178, 179, 180, 208, 229, 232, 234

Raster map, iv, x, 176, 177, 178, 234

rasterize, 177, 179, 234

ratioByGroup, 51

rbind, 219, 234

read_csv, 163, 166, 167, 175, 177, 179, 183, 186, 215

Readability, 3, 8, 33, 39, 40, 45, 46, 55, 59, 63, 66, 78, 82, 86, 88, 109, 126, 153, 155, 162, 191, 194

rect_border, 59

rect_width, 59

regex, 179, 215, 234

Relationship, iii, 6, 17, 28, 35, 63, 106, 108, 109, 110, 114, 115, 117, 119, 121, 122, 123, 124, 125, 126, 133, 137, 140, 144, 145, 146, 166, 197, 200, 207, 208, 220, 221, 223, 225, 227, 228, 234, 235

reliability, 133, 137, 138, 139, 141, 142, 144, 146, 149

remove, 45, 74, 126, 179, 183, 185, 186, 215, 234

remove.overlap, 179, 183, 186

rename, 88, 131, 189, 234

rep, 34, 35, 40, 73, 110, 195, 198, 219, 234

repel, i, vii, 40, 41, 164, 167, 168, 199, 229, 234, 236

replace, 73, 87, 179, 183, 186, 215, 230, 234

Ribbon, iii, ix, 6, 8, 133, 140, 144, 145, 146, 147, 188, 189, 229, 234

Ribbon chart, ix, 133, 144, 145, 234

right, 5, 6, 11, 15, 16, 18, 19, 20, 24, 26, 27, 28, 29, 48, 70, 71, 77, 80, 81, 82, 128, 129, 135, 137, 138, 185, 194, 197, 210, 220, 223, 234

round, 11, 43, 45, 71, 75, 78, 121, 126, 189, 191, 234

row_number, 158, 174, 191, 192, 234

rownames, 58, 160, 234

rug, i, 8, 27, 28, 229, 234

sample, 23, 37, 38, 73, 87, 234, 236

Sankey, iv, x, 6, 188, 189, 190, 229, 234

scale, 3, 8, 11, 16, 17, 19, 20, 22, 23, 24, 25, 26, 28, 29, 31, 32, 33, 39, 40, 43, 44, 45, 47, 50, 53, 55, 56, 57, 59, 64, 65, 66, 67, 68, 70, 71, 73, 75, 76, 78, 79, 83, 85, 86, 87, 91, 100, 102, 103, 105, 106, 108, 109, 113, 115, 116, 117, 119, 122, 123,

125, 126, 129, 131, 133, 134, 136, 137, 139, 140, 141, 143, 145, 148, 150, 151, 152, 153, 154, 156, 158, 169, 171, 173, 174, 177, 180, 181, 182, 183, 186, 189, 192, 209, 213, 217, 218, 221, 235

scale_color_gradient, 109

scale_color_manual, 20, 22, 23, 25, 26, 28, 29, 39, 40, 102, 103, 105, 106, 113, 115, 116, 117, 119, 122, 123, 134, 137, 139, 140, 141, 143, 148, 150, 156, 158

scale_fill_gradient, 129, 169, 174, 213, 221

scale_fill_manual, 16, 17, 19, 20, 22, 23, 25, 26, 29, 43, 44, 45, 50, 53, 55, 56, 57, 64, 65, 66, 67, 68, 70, 73, 75, 76, 86, 87, 131, 136, 143, 151, 152, 170, 171

scale_fill_viridis_c, 177, 181

scale_pattern_fill_manual, 47

scale_pattern_spacing_discrete, 47

scale_pattern_type_discrete, 47

scale_size, 31, 32, 33, 122, 123

scale_size_area, 31, 32, 33

scale_x_continuous, 11, 78, 79, 108, 113

scale_x_discrete, 23, 25, 55, 189, 192

scale_y_continuous, 11, 16, 17, 19, 23, 39, 40, 55, 57, 59, 65, 67, 68, 70, 71, 75, 76, 91, 100, 106, 108, 109, 113, 122, 123, 133, 134, 136, 137, 139, 140, 141, 143, 145, 148, 150, 152, 154, 156

scatterplot, 35, 105, 106, 119, 203, 208, 209, 235

season, 91, 235

Segmented chart, iii, viii, 106, 107, 235

select, 82, 126, 191, 235

sep, 14, 23, 61, 129, 235

seq, 73, 120, 154, 195, 235

sequence, 60, 89, 100, 113, 116, 130, 153, 194, 223, 235

sequential, 127, 233, 237

set.seed, 73, 86, 235

set_crs_if_missing, 182, 185

setNames, 115, 235

SetNodeStyle, 60, 61, 235

sf, 8, 163, 164, 165, 166, 167, 169, 170, 171, 172, 173, 174, 175, 177, 179, 181, 182, 183, 185, 186, 215, 229, 230, 235

shadow, 179, 186, 215, 235

Shape, 3, 5, 6, 7, 8, 12, 13, 14, 16, 20, 22, 23, 25, 26, 50, 58, 60, 77, 99, 100, 101, 105, 110, 116, 133, 134, 142, 143, 144, 146, 149, 156, 163, 167, 170, 175, 177, 179, 180, 182, 183, 185, 186, 202, 214, 215, 218, 226, 230, 233, 234, 235

Shapefile, xiv, 162, 163, 166, 167, 175, 182, 235

show.legend, 58, 71, 172

Size, 3, 4, 5, 11, 12, 17, 20, 23, 25, 30, 31, 32, 33, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 56, 57, 66, 74, 75, 76, 78, 79, 81, 82, 83, 84, 88, 94, 96, 98, 104, 105, 108, 109, 110, 111, 112, 114, 115, 118, 120, 119, 120, 121, 122, 123, 124, 126, 129, 133, 134, 137, 144, 146, 148, 149, 150, 151, 154, 158,

164, 165, 167, 168, 169, 171, 172, 173, 174, 176,
 177, 179, 180, 181, 183, 186, 187, 189, 191, 192,
 194, 195, 196, 199, 214, 215, 216, 217, 219, 220,
 227, 228, 230, 231, 233, 235
 skewness, 13, 15, 20, 24, 26, 28, 29, 35, 39, 201, 235
 slice_max, 95, 96
 Slope, iii, ix, 6, 86, 106, 111, 112, 115, 119, 235
 Slope chart, iii, ix, 86, 106, 111, 112, 235
 Smoothed line, iii, ix, 116, 117, 118, 141, 235
 Sorted bar chart, iv, ix, 151, 152, 153, 160, 235
 Sparse, 156, 165, 174, 235
 sparsely, 173, 222, 235
 Spatial mapping, iv, ix, 162, 164, 235
 spider, ii, viii, 63, 82, 83, 217, 218, 219, 231, 234,
 235
 spoke, iii, 8, 107, 108, 109, 110, 230, 235
 spoke chart, iii, 107, 108, 109, 235
 spread, 6, 13, 16, 21, 22, 24, 26, 29, 34, 35, 36, 110,
 115, 126, 133, 138, 139, 142, 143, 144, 146, 147,
 149, 166, 170, 202, 203, 227, 228, 234, 236
 sqrt, 107, 108, 109, 235
 st, 163, 166, 167, 170, 173, 174, 175, 177, 179, 182,
 183, 185, 186, 214, 215
 st_as_sf, 163, 166, 167, 175, 177, 179, 183, 186, 215
 st_bbox, 175
 st_buffer, 175
 st_coordinates, 163, 166, 167, 175
 st_crs, 174, 177, 182, 185
 st_drop_geometry, 163, 167
 st_filter, 183, 186
 st_intersects, 163, 167, 173, 174, 179, 183, 186
 st_join, 163, 166, 167, 215
 st_make_valid, 163
 st_read, 163, 166, 167, 170, 175, 177, 179, 182, 185,
 214
 st_set_crs, 163, 166, 167, 170, 175, 177, 179, 182,
 185, 214
 st_transform, 163, 166, 167, 170, 175, 177, 179, 182,
 183, 185, 186, 215
 Stacked bar, ii, vii, 3, 6, 42, 52, 55, 56, 73, 228, 235
 stat, 14, 15, 16, 17, 19, 23, 25, 31, 32, 33, 53, 55, 64,
 65, 66, 68, 70, 71, 73, 75, 135, 152, 181, 189,
 191, 192, 213, 221, 235
 stat_bin2d, 213, 221
 stat_boxplot, 23, 25
 stat_density_2d, 181
 static, 4, 7, 83, 151, 179, 182, 191, 214
 static display, 151
 statistics, xiii, 13, 23, 25, 37, 63, 119, 235
 Step chart, ii, iii, viii, 97, 98, 99, 235
 STL, 91, 235
 stop, 182, 185, 235, 236
 str_detect, 179, 182, 185, 215
 str_replace_all, 179, 183, 186, 215
 Strata, 188, 193, 235
 Stratum, 188, 189, 190, 191, 192, 230, 235

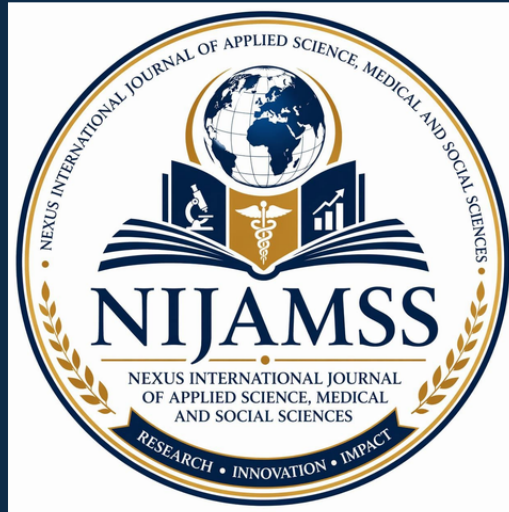
Stream, iii, ix, 110, 111, 189, 191, 230, 235
 Streams, 111, 188, 190, 191, 193, 235
 stringsAsFactors, 61, 198, 235
 subtitle, 76, 121, 164, 168, 169, 235
 suffix, 73, 235
 summarise, 45, 53, 64, 66, 68, 70, 71, 82, 131, 135,
 146, 152, 188, 189, 215, 235
 summary, 13, 21, 22, 23, 25, 26, 30, 73, 125, 128,
 131, 156, 226, 235
 sunburst, ii, viii, 42, 57, 58, 235
 Symbol, iv, x, 42, 162, 172, 173, 183, 186, 231, 236
 Symbol map, iv, x, 162, 172, 173, 236
 Symmetrically, 80, 151, 156, 227, 236
 Symmetrically distributed, 227, 236
 symmetry, 16, 20, 26, 70, 236
 Tag, 11
 tags, 40
 Tail, 15, 18, 20, 38
 tails, 34, 37, 39, 143, 201, 205, 236
 tally, 74, 75, 78, 236
 taper, 203
 tapering, 15, 16
 t-distribution, 37, 236
 Tertile plot, i, vii, 35, 36, 236
 tethering, 83
 Text chart, i, vii, 39, 40, 41, 236
 Text repel, i, vii, 40, 41, 236
 theme, 11, 17, 43, 44, 45, 50, 53, 58, 59, 78, 79, 80,
 81, 86, 87, 88, 89, 90, 91, 93, 94, 96, 98, 101,
 104, 106, 107, 110, 111, 112, 118, 124, 126, 128,
 129, 158, 171, 189, 192, 195, 196, 199, 201, 236
 theme_minimal, 53, 86, 87, 88, 89, 91, 94, 96, 98,
 104, 110, 112, 124, 129, 158, 189, 192
 theme_void, 43, 44, 45, 50, 53, 58, 78, 79, 80, 195,
 196, 199
 tidyverse, 14, 182, 185, 231, 232, 236, 237
 Tile, iv, x, 8, 128, 129, 173, 174, 176, 178, 207, 214,
 230, 236
 Tile map, iv, x, 173, 174, 214, 236
 tiles, 128, 176, 207, 236
time, ii, iii, viii, ix, 3, 4, 5, 6, 42, 43, 64, 68, 71, 86,
 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
 100, 101, 103, 104, 105, 106, 110, 111, 112, 113,
 114, 115, 118, 120, 125, 126, 127, 128, 129, 130,
 131, 140, 151, 182, 188, 223, 227, 228, 231, 232,
 235, 236, 237
 title, 4, 8, 11, 12, 17, 22, 23, 25, 26, 28, 29, 30, 31,
 32, 33, 34, 36, 37, 38, 39, 40, 51, 55, 57, 58, 59,
 61, 64, 65, 67, 68, 70, 71, 73, 76, 81, 82, 84, 87,
 88, 90, 91, 93, 94, 96, 98, 99, 100, 101, 103, 104,
 105, 106, 108, 109, 110, 112, 113, 115, 116, 117,
 118, 119, 121, 123, 124, 126, 128, 129, 131, 134,
 136, 137, 140, 141, 143, 145, 146, 148, 150, 152,
 154, 156, 158, 159, 160, 164, 165, 168, 169, 171,
 172, 174, 175, 177, 180, 181, 183, 184, 186, 189,
 192, 196, 199, 202, 204, 206, 207, 208, 209, 210,

211, 212, 213, 215, 217, 219, 220, 221, 223, 224,
 226, 230, 231, 232, 235, 236
 tm_borders, 183, 186
 tm_layout, 180, 182, 184, 186, 215
 tm_legend, 179, 183, 186
 tm_lines, 182, 183
 tm_polygons, 179, 185, 186, 214, 215
 tm_scale, 180, 182, 183, 186
 tm_shape, 179, 180, 183, 185, 186, 214, 215
 tm_text, 179, 182, 183, 186, 215
 tm_title, 180, 184, 186
 tmap_mode, 179, 182, 185, 214, 215
 ToDiagrammeRGraph, 61, 236
 tolower, 182, 236
 top, 5, 28, 35, 44, 50, 55, 61, 62, 78, 81, 86, 135, 151,
 156, 158, 159, 160, 161, 180, 184, 186, 226, 236
 topography, 212, 236
 total_rect_color, 59
 total_rect_text_color, 59
 TRUE, 1, 8, 14, 51, 57, 58, 59, 64, 65, 71, 73, 75, 82,
 87, 124, 126, 129, 135, 138, 140, 144, 145, 151,
 177, 179, 180, 182, 183, 184, 185, 186, 199, 214,
 215, 227, 228, 236

 Uncertainty, iii, 6, 64, 66, 92, 93, 116, 133, 134, 135,
 136, 137, 138, 139, 140, 142, 143, 144, 145, 146,
 147, 148, 149, 150, 151, 227, 228, 229, 236
 ungroup, 71, 73, 82, 137, 145, 158, 192, 236
 unit, 11, 12, 45, 50, 78, 79, 93, 113, 123, 134, 136,
 137, 140, 141, 143, 145, 146, 148, 150, 152, 157,
 162, 194, 196, 199, 236
 Univariate, x, 203, 210, 225, 227, 236
 Univariate distribution, 225, 227, 236
 values, 3, 4, 5, 6, 13, 14, 15, 16, 17, 19, 20, 21, 22,
 23, 24, 25, 26, 27, 28, 29, 30, 32, 35, 36, 37, 38,
 39, 40, 43, 44, 45, 46, 47, 49, 50, 51, 53, 55, 57,
 59, 63, 64, 65, 66, 67, 68, 69, 70, 71, 73, 74, 76,
 77, 82, 83, 84, 86, 87, 91, 92, 97, 98, 99, 100,
 101, 103, 105, 106, 108, 111, 113, 115, 116, 117,
 119, 120, 122, 123, 125, 126, 127, 128, 129, 130,
 131, 133, 134, 135, 136, 137, 138, 139, 140, 141,
 142, 143, 144, 145, 146, 147, 148, 149, 150, 151,
 152, 153, 156, 157, 158, 160, 162, 167, 169, 170,
 171, 172, 176, 177, 180, 183, 186, 189, 190, 191,
 192, 193, 201, 202, 203, 204, 206, 207, 209, 210,
 213, 214, 218, 219, 223, 227, 228, 229, 230, 232,
 233, 234, 235, 236, 237
 values.radar, 82
 values_from, 126, 192
 Variability, 6, 18, 22, 26, 64, 65, 66, 132, 133, 135,
 137, 138, 139, 140, 141, 142, 144, 145, 146, 147,
 149, 150, 151, 226, 228, 229
 Variable, 6, 15, 21, 23, 32, 34, 63, 64, 74, 82, 113,
 118, 119, 122, 128, 135, 138, 139, 142, 144, 169,
 194, 195, 197, 201, 217, 218, 219, 222, 225, 226,
 227, 228, 229, 231, 234, 236
 vars, 117, 119, 236
 varwidth, 23, 25, 236
 Vertical count chart, vii, 32, 236
 Vertical errorbar, 236
 view, 13, 17, 18, 26, 27, 30, 31, 42, 43, 49, 52, 57,
 59, 63, 75, 80, 81, 82, 84, 95, 101, 108, 113, 119,
 130, 132, 133, 142, 144, 151, 162, 172, 173, 189,
 201, 208, 218, 224, 227
 viewers, 2, 3, 4, 5, 6, 8, 43, 48, 66, 94, 98, 102, 103,
 135, 137, 139, 141, 144, 146, 149, 150, 158, 168,
 175, 210
 violin, i, iii, vii, ix, 6, 8, 13, 26, 27, 133, 142, 143,
 144, 156, 226, 227, 230, 236
 visual, xii, 1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 19, 21, 34, 38,
 39, 40, 43, 46, 47, 48, 53, 57, 59, 60, 63, 65, 66,
 68, 69, 72, 74, 75, 77, 80, 85, 90, 91, 97, 99, 100,
 101, 103, 105, 106, 107, 108, 109, 113, 115, 116,
 118, 119, 121, 124, 125, 127, 128, 131, 133, 135,
 137, 144, 148, 151, 153, 155, 156, 160, 162, 163,
 166, 167, 170, 174, 177, 182, 184, 188, 194, 199,
 201, 212, 216, 219, 221, 222, 223, 226, 227, 228,
 230, 231, 232, 233, 236
 Visual encoding, 170
 visualization, i, xii, xiii, xiv, 1, 2, 3, 4, 5, 6, 7, 8, 11,
 21, 22, 26, 35, 40, 49, 51, 54, 56, 64, 77, 78, 81,
 82, 85, 87, 91, 94, 95, 96, 97, 104, 108, 111, 114,
 119, 123, 124, 125, 130, 133, 139, 142, 144, 146,
 149, 155, 158, 166, 168, 170, 172, 175, 180, 198,
 200, 203, 210, 213, 219, 226, 227, 228, 229, 230,
 231, 232, 233, 234, 236
 vjust, 39, 40, 47, 73, 96, 236
 warning, 183, 236
 Waterfall, ii, vi, viii, 6, 42, 59, 60, 232, 237
 Wedges, 43
 Whole, 6, 14, 30, 31, 42, 43, 44, 46, 49, 50, 54, 56,
 57, 85, 185, 198, 228, 234
 Width, 13, 16, 18, 19, 23, 25, 53, 58, 59, 64, 65, 66,
 67, 68, 70, 82, 128, 131, 133, 134, 136, 139, 140,
 145, 152, 176, 179, 182, 183, 188, 189, 192, 231,
 232, 237
 window, 91, 103, 104, 105, 237
 Wrap, 19, 50, 58, 114, 115, 117, 119, 224, 228, 237
 xend, 94, 106, 131, 154, 196, 237
 xintercept, 101, 237
 xlab, 11, 14, 16, 17, 19, 20, 22, 23, 25, 26, 28, 29, 30,
 31, 32, 33, 34, 35, 37, 38, 39, 40, 55, 57, 64, 65,
 67, 68, 76, 90, 91, 99, 100, 101, 103, 105, 106,
 108, 109, 115, 123, 134, 136, 137, 140, 141, 143,
 145, 146, 148, 150, 152, 154, 156, 237
 xmax, 43, 45, 50, 103, 104, 131, 137, 175, 194, 195,
 237
 xmin, 43, 45, 50, 103, 104, 131, 137, 175, 194, 195,
 237
 x-xis, 237

y-axis, 11, 19, 21, 31, 53, 71, 75, 88, 101, 103, 107,
119, 120, 133, 137, 138, 139, 158, 192, 206, 219,
220, 221, 226, 237
yend, 94, 106, 131, 154, 196, 237
yintercept, 16, 70, 71, 73, 76, 154, 237
ylab, 11, 14, 16, 17, 19, 20, 22, 23, 25, 26, 28, 29, 30,
31, 32, 33, 34, 35, 37, 38, 39, 40, 55, 57, 64, 65,
67, 68, 76, 90, 99, 100, 101, 103, 105, 106, 108,
109, 123, 134, 136, 137, 140, 141, 143, 145, 146,
148, 150, 152, 154, 156, 237

ylim, 75, 120, 195, 196, 237
ymax, 43, 45, 50, 65, 66, 68, 103, 104, 131, 134, 136,
139, 145, 146, 148, 149, 150, 152, 175, 194, 195,
237
ymin, 43, 45, 50, 65, 66, 68, 103, 104, 131, 134, 136,
139, 145, 146, 148, 149, 150, 152, 175, 194, 195,
237
zone, 102, 166, 175, 177, 237
zones, 103, 156, 175, 211, 212, 213, 214, 237



NIJAMSS | SPECIAL ISSUE | VOLUME 1 | ISSUE 2 | 2026

FROM R TO INSIGHT

Turning Data into Insight

A practical visual guide for students, researchers, analysts,
and data-science practitioners who want to communicate
statistical information clearly and confidently.

Open Access | Double-Blind Peer Reviewed | CC BY 4.0

e-ISSN: 3144-2196

www.nijamss.org | editor@nijamss.org

NEXUS ACADEMIC PRESS

An imprint of Nexus Health Solutions LLC

Cheyenne, Wyoming, USA

Research | Innovation | Impact

Published 2026