R for Data Analytics

Siju Swamy

2025-06-23

Table of contents

Preface 6
Data vs. Information . . . 6
Overview of Modern Data Analytic Tools . . . 8
1. Programming Languages . . . 8
2. Data Storage Solutions . . . 9
3. Data Processing Frameworks . . . 11
4. Data Visualization Tools . . . 12
5. Specialized Platforms . . . 14
Integration with R . . . 15
Considerations for Choosing Tools . . . 15

1 Introduction to R and RStudio 16
1.1 Welcome! . . . 16
1.2 Learning Objectives . . . 16
1.3 Installing R . . . 16
1.4 Installing RStudio . . . 16
1.5 Exploring RStudio . . . 17

2 Basics of R programming for Data Analysis 18
2.1 Running Basic R Commands . . . 18
2.2 Installing and Loading Packages . . . 19
2.3 Downloading Sample Data . . . 19
2.4 Inspecting Data . . . 19
2.5 Data Input and Data Types . . . 20
2.5.1 Data Types in R . . . 20
2.5.2 Checking Data Types . . . 20
2.5.3 Converting Data Types . . . 21
2.6 Lists, Arrays, and Data Frames in R . . . 21
2.7 1. Lists . . . 21
2.7.1 Creating Lists . . . 22
2.7.2 Accessing List Elements . . . 22
2.7.3 Modifying Lists . . . 23
2.8 2. Arrays . . . 24
2.8.1 Creating Arrays . . . 24
2.8.2 Array Operations . . . 26

2.9 3. Data Frames 27
2.9.1 Accessing Data Frame Elements 27
2.9.2 Modifying Data Frames 28
2.9.3 Data Frame Operations 29

3 Data Cleaning and Transformation 31
3.1 Introduction 31
3.2 Learning Objectives 31
3.3 What is Dirty Data? 31
3.4 The `exam_scores` Dataset: Let's Get Specific 31
3.5 Identifying Data Quality Issues 32
3.6 Handling Missing Values 35
3.7 Handling Outliers 36
3.8 Correcting Inconsistent Data 37

4 Descriptive Statistics 38
4.1 Introduction 38
4.2 Learning Objectives 38
4.3 Measures of Central Tendency 38
4.4 Measures of Dispersion 39
4.5 Descriptive Statistics with `dplyr::summarize()` 40
4.5.1 Grouped Descriptive Statistics 41
4.6 Descriptive Statistics and Data Distribution 42
4.7 Practice 42

5 Exploratory Data Analysis 43
5.1 Visualizing a Single Variable 43
5.2 Introduction to Visualizing a Single Variable 43
5.2.1 Histograms 43
5.2.2 Histograms 45
5.2.3 Pie Charts 46
5.2.4 Bar Charts 47
5.2.5 Box Plots 48
5.2.6 Density Plots 49
5.3 Practice 50
5.4 Bivariate visualization 51
5.5 What is Bivariate EDA? 51
5.6 Scatter Plots 51
5.7 Box Plots 52
5.7.1 Exploring the Impact of Study Hours: Why Group? 53
5.8 Line Plots 55
5.8.1 Cobweb Plot (Radar Chart) 56
5.9 Multi variable plots in R 57

6 Data Exploration Versus Presentation 59
6.1 Introduction . . . 59
6.2 Learning Objectives . . . 59
6.3 What is Data Exploration? . . . 59
6.3.1 Characteristics of Data Exploration: . . . 59
6.4 What is Data Presentation? . . . 60
6.4.1 Characteristics of Data Presentation: . . . 60
6.5 Key Differences Between Data Exploration and Presentation . . . 60
6.6 Choosing Appropriate Visualization Methods . . . 61
6.6.1 Exploration Visualizations: . . . 61
6.6.2 Presentation Visualizations: . . . 61
6.6.3 Common rules of thumb for presenting datasets for people . . . 61
6.7 Creating Compelling Visuals for Data Presentation . . . 61
6.8 Dangers of Data Visualization . . . 65
6.8.1 common pitfalls . . . 65
6.9 Conclusion . . . 66

7 Statistical Methods for Evaluation 68
7.1 Introduction . . . 68
7.2 Learning Objectives . . . 68
7.3 What is a statistical test? . . . 68
7.3.1 Types of Data that work best with different tests . . . 68
7.3.2 Performing a Normal Statistical Test . . . 69
7.3.3 Performing a t- Test . . . 70
7.3.4 Performing a correlation Test . . . 73
7.3.5 Performing a chi-squared Test . . . 74

8 Perception of Engineering Students on Fruit Preference 76
8.1 Introduction . . . 76
8.2 Data cleaning and Wrangling . . . 76
8.2.1 Rename the attribute names . . . 77
8.3 Descriptive Analysis . . . 77
8.3.1 Conversion of data into long format . . . 79
8.4 Gender-wise preference of fruits . . . 83
8.5 Overall preference over fruits . . . 86
8.6 Finding descriptive statistics of continuous variables . . . 87
8.6.1 Rename the levels (For Numerical calculation) . . . 89
8.7 Inferential Analysis . . . 89
8.7.1 Testing of significance of difference in mean weight over gender . . . 89
8.7.2 Significance of difference in mean weight over Fruit interest . . . 90
8.8 Testing of significance difference in rating over Fruit category . . . 91
8.8.1 Percentage analysis of the rating of fruit category . . . 91
8.8.2 χ^2 test for confirmation of difference in rating over fruit category . . . 92

8.8.3 Fruit preference over Age group 92
8.9 Conclusion 93

Preface

Data analysis is a skill for everyone. In this module, we'll break down the barriers and make R programming accessible and empowering. You don't need to be a math whiz or a programming expert. We'll start with the fundamentals and guide you step-by-step as you learn to use R for data exploration, visualization, and analysis. More than just coding, you'll discover how to think critically about data, ask smart questions, and communicate your insights effectively. This module is your launchpad for a data-driven future!

Figure 1: Participate in a preference survey

Data vs. Information

Data:

- **Definition:** Data consists of raw, unorganized facts, figures, symbols, and details about people, objects, events, or situations. It can exist in various forms, such as numbers, text, images, audio, video, or signals. Data is often collected or generated through observation, measurement, or experimentation. Data, in itself, has no inherent meaning or context.
- **Characteristics:**
  - Raw and unorganized

    - Lacks context and meaning
    - Can be quantitative or qualitative
    - Collected through observation, measurement, or experimentation
    - Stored in various formats (e.g., tables, files, databases)

- **Examples:**
    - A list of temperature readings in Celsius for a city, such as: 25, 27, 29, 26, 24
    - A collection of customer names and addresses stored in a database.
    - A series of audio samples recorded by a microphone.
    - A set of images captured by a camera.

Information:

- **Definition:** Information is data that has been processed, organized, structured, and given context to make it meaningful and useful for decision-making. Information answers questions like "who," "what," "where," "when," and "how many."
- **Characteristics:**
    - Processed, organized, and structured
    - Provides context and meaning
    - Answers questions about the data
    - Useful for decision-making, analysis, and communication
    - Can be derived from data through analysis, interpretation, and summarization
- **Examples:**
    - The average temperature for the city based on the raw temperature readings.
    - A report showing the total sales by product category for a specific month.
    - A graph visualizing the trend of website traffic over time.
    - A summary of customer demographics based on data from a database.

Key Differences between Data and Information:

The easiest way to think of the difference is to consider that data needs to be worked upon to become information.

Feature	Data	Information
State	Raw, unorganized	Processed, organized, structured
Context	Lacks context and meaning	Provides context and meaning
Purpose	Storage, collection	Analysis, decision-making, communication
Format	Various (numbers, text, images)	Meaningful and understandable
Example	Temperature readings: 25, 27, 29	Average temperature: 26.2 °C

Feature	Data	Information
Dependency	It is fundamental and basic.	It is reliant on the processing and interpretation of data.

Analogy:

Think of data as the ingredients for a recipe. On their own, flour, eggs, and sugar are just raw materials. Information is the final baked cake that you understand, eat, and enjoy.

Conclusion

Data is the foundation upon which information is built. Data needs to be processed and transformed to become information, which is meaningful and useful for decision-making.

Overview of Modern Data Analytic Tools

Data analytics has become an integral part of decision-making across various industries. The field has evolved rapidly, resulting in a wide array of tools and technologies available to data professionals. Selecting the right tools for a project depends on factors like the size and complexity of the data, the analytical goals, the budget, and the team's skillset. This section provides an overview of some essential modern data analytic tools.

1. Programming Languages

Programming languages form the foundation for most data analytic tasks. They provide the flexibility to perform custom analyses, build complex models, and automate data processing.

R

- **Description:** R is a language and environment specifically designed for statistical computing and graphics. It excels in statistical analysis, data visualization, and building custom analytical solutions.
- **Key Features:**
  - Extensive collection of packages for statistical modeling, machine learning, and data manipulation.
  - Excellent visualization capabilities through packages like `ggplot2`, enabling the creation of high-quality static plots.
  - A vibrant and active community, providing ample resources, tutorials, and support.
  - Strong focus on statistical rigor and reproducible research.

- **Use Cases:** Statistical analysis, data visualization, building custom analytical models, reproducible research.
- **R Tidyverse:** A collection of R packages which helps create a tidy data set. The `tidyverse` package helps with:
  - Easier to read and interpret
  - Easier to maintain since all packages integrate better with each other.

Python

- **Description:** Python is a versatile, general-purpose programming language widely used for data science, machine learning, web development, and more. Its simplicity and extensive libraries make it a popular choice for data analysis.
- **Key Features:**
  - Rich ecosystem of libraries for data analysis (Pandas, NumPy), machine learning (Scikit-learn, TensorFlow, PyTorch), and visualization (Matplotlib, Seaborn).
  - Easy-to-learn syntax and a large, supportive community.
  - Strong integration with other systems and technologies, making it suitable for building end-to-end data pipelines.
  - Wide adoption in machine learning and artificial intelligence research.
- **Use Cases:** Data analysis, machine learning, building data pipelines, web development, automation.

SQL

- **Description:** SQL (Structured Query Language) is the standard language for managing and querying relational databases.
- **Key Features:**
  - Used to extract, manipulate, and analyze data stored in relational databases.
  - Essential for retrieving data for analysis and generating reports.
  - Used for database administration tasks.

2. Data Storage Solutions

Data storage solutions play a critical role in housing the data that fuels data analytics. These solutions range from traditional relational databases to modern NoSQL databases and cloud-based storage services.

Relational Databases (SQL Databases)

- **Description:** Relational databases, such as MySQL, PostgreSQL, Oracle, and SQL Server, organize data into structured tables with rows and columns. They use SQL to query and manage the data.
- **Key Features:**
  - Structured storage based on tables with well-defined relationships.
  - ACID compliance (Atomicity, Consistency, Isolation, Durability) ensures data integrity.
  - SQL language for querying, manipulating, and managing data.
  - Suitable for applications requiring structured data with clear schemas.
- **Use Cases:** Transaction processing, data warehousing, applications with structured data requirements.

NoSQL Databases

- **Description:** NoSQL databases are non-relational databases that offer flexibility and scalability for handling unstructured, semi-structured, and large-volume data.
- **Key Features:**
  - Support various data models, including document, key-value, graph, and column-family stores.
  - Designed to handle unstructured or semi-structured data, large volumes of data, and high-velocity data streams.
  - Scalability and high availability for demanding applications.
- **Use Cases:** Web applications, social media analytics, IoT data storage, applications with flexible data schemas.

Cloud Storage

- **Description:** Cloud storage solutions, such as Amazon S3, Azure Blob Storage, and Google Cloud Storage, provide scalable and cost-effective storage for large datasets.
- **Key Features:**
  - Scalable storage for large datasets in the cloud.
  - Cost-effective, pay-as-you-go pricing model.
  - Easy access and integration with other cloud services.
  - Suitable for storing data for big data analytics and machine learning.

- **Use Cases:** Data lakes, data warehousing, backup and archiving, serving content to web applications.

3. Data Processing Frameworks

Data processing frameworks are designed to handle the complexities of large-scale data processing, enabling data analysts to extract valuable insights from massive datasets.

Apache Hadoop

- **Description:** Apache Hadoop is an open-source framework for distributed storage and processing of large datasets.
- **Key Features:**
  - Distributed storage using the Hadoop Distributed File System (HDFS).
  - Distributed processing using the MapReduce programming model.
  - Scalability and fault tolerance for processing large datasets.
  - Suitable for batch processing of historical data.
- **Use Cases:** Batch processing of large datasets, data warehousing, log analysis.

Apache Spark

- **Description:** Apache Spark is a fast and general-purpose distributed processing engine.
- **Key Features:**
  - In-memory processing for faster analytics.
  - Support for various programming languages (Python, R, Scala, Java).
  - Integration with Hadoop and other data sources.
  - Libraries for machine learning (MLlib) and graph processing (GraphX).
- **Use Cases:** Real-time data processing, machine learning, graph analysis, ETL (Extract, Transform, Load) operations.

Cloud-Based Data Warehouses

- **Description:** Cloud-based data warehouses, such as Amazon Redshift, Google BigQuery, and Azure Synapse Analytics, provide scalable storage, data warehousing capabilities, and integration with other cloud services.
- **Key Features:**
  - Scalable storage and compute resources for data warehousing.
  - SQL-based querying and analysis.
  - Integration with cloud-based data sources and analytics services.
  - Pay-as-you-go pricing models for cost-effectiveness.
- **Use Cases:** Data warehousing, business intelligence, reporting, large-scale data analytics.

4. Data Visualization Tools

Data visualization tools enable data professionals to create visually appealing and informative representations of data, facilitating insights and effective communication.

ggplot2 (R)

- **Description:** `ggplot2` is a powerful and flexible visualization package for R, based on the Grammar of Graphics.
- **Key Features:**
  - A consistent and coherent system for creating a wide range of static plots.
  - Flexibility to customize every aspect of the plot, from colors and fonts to scales and labels.
  - Integration with other R packages, enabling the creation of dynamic and interactive visualizations.
- **Use Cases:** Creating static plots for reports, publications, and presentations.

Matplotlib and Seaborn (Python)

- **Description:** Matplotlib and Seaborn are popular visualization libraries in Python. Matplotlib provides basic plotting capabilities, while Seaborn builds on Matplotlib to offer more advanced and visually appealing plots.
- **Key Features:**
  - Matplotlib provides a wide range of plotting functions for creating static and interactive plots.
  - Seaborn offers pre-built styles and plot types for creating visually appealing visualizations.
  - Integration with Pandas and NumPy for easy data handling.
- **Use Cases:** Creating static and interactive plots for data exploration, presentation, and reporting.

Tableau

- **Description:** Tableau is a commercial data visualization and business intelligence tool known for its user-friendly interface, interactive dashboards, and ability to connect to various data sources.
- **Key Features:**
  - Drag-and-drop interface for creating visualizations.
  - Interactive dashboards for exploring data and uncovering insights.
  - Ability to connect to a wide range of data sources, including databases, spreadsheets, and cloud services.
- **Use Cases:** Business intelligence, data exploration, creating interactive dashboards, sharing insights.

Power BI

- **Description:** Microsoft Power BI is a business intelligence and data visualization tool offering similar capabilities to Tableau and integrating well with other Microsoft products.
- **Key Features:**
  - User-friendly interface for creating visualizations and dashboards.
  - Integration with Microsoft Excel, SQL Server, and other Microsoft services.

  - Ability to connect to various data sources, including cloud services and on-premises databases.

- **Use Cases:** Business intelligence, data exploration, creating interactive dashboards, sharing insights.

5. Specialized Platforms

In addition to the core tools and technologies, specialized platforms offer specific functionalities and capabilities for advanced data analytics.

Machine Learning Platforms

- **Description:** Cloud-based platforms like Amazon SageMaker, Google AI Platform, and Azure Machine Learning Studio provide tools for building, training, and deploying machine learning models.
- **Key Features:**
  - Managed environments for machine learning development.
  - Support for various machine learning frameworks and algorithms.
  - Scalable compute resources for training models.
  - Tools for model deployment and monitoring.
- **Use Cases:** Building and deploying machine learning models for predictive analytics, recommendation systems, image recognition, and natural language processing.

Data Science Notebooks

- **Description:** Interactive environments like Jupyter Notebook and R Markdown that allow you to combine code, text, and visualizations in a single document.
- **Key Features:**
  - Interactive coding and execution of code snippets.
  - Support for multiple programming languages (Python, R, Julia).
  - Ability to embed visualizations, equations, and narrative text.
  - Ideal for exploratory data analysis, reproducible research, and creating interactive reports.
- **Use Cases:** Exploratory data analysis, data cleaning and transformation, building analytical models, creating interactive reports.

Cloud-Based Analytics Services

- **Description:** Services like AWS Analytics, Google Cloud Analytics, and Azure Analytics provide a suite of tools for data ingestion, processing, analysis, and visualization in the cloud.
- **Key Features:**
  - Managed services for data ingestion, storage, processing, and analysis.
  - Integration with other cloud services.
  - Scalable compute and storage resources.
  - Pay-as-you-go pricing models.
- **Use Cases:** End-to-end data analytics pipelines, data warehousing, business intelligence, machine learning.

Integration with R

Many of the tools listed can be integrated with R, enhancing the analytical capabilities of the R environment:

- **RJDBC and RODBC:** Packages for connecting R to relational databases.
- **sparklyr:** An R interface to Apache Spark, enabling distributed data processing from R.
- **R API Clients:** Packages for accessing data from cloud services, such as AWS, Google Cloud, and Azure.

Considerations for Choosing Tools

- **Data Volume and Velocity:** Consider distributed processing frameworks like Spark for very large datasets.
- **Data Structure:** Choose relational databases for structured data and NoSQL databases for unstructured or semi-structured data.
- **Analytic Goals:** Select tools with libraries and functions tailored to your specific analytical needs.
- **Skill Set:** Consider your existing skills and the learning curve associated with each tool.
- **Cost:** Evaluate the costs of licenses, cloud services, and infrastructure.

This overview provides a comprehensive guide to modern data analytic tools, enabling data professionals to make informed decisions about selecting the right tools for their projects. Remember to consider the specific requirements of your analytical goals and the characteristics of your data.

1 Introduction to R and RStudio

1.1 Welcome!

This session will get you up and running with R and RStudio, the tools we'll be using throughout this module. You'll learn how to install the software, explore the RStudio interface, and run your first R commands. We'll also cover the crucial role of version control for all of your projects.

1.2 Learning Objectives

- Install R and RStudio.
- Understand the different panels in RStudio.
- Run basic R commands.
- Install and load R packages.
- Download sample data.
- Inspect a csv file using RStudio.
- Understand the basics of Version Control
- Understand the basics of a working branch in version control

1.3 Installing R

1. **Go to the CRAN website:** https://cran.r-project.org/
2. **Choose your operating system:** (Windows, macOS, or Linux)
3. **Download the installer:** Select the appropriate download link for your system.
4. **Run the installer:** Follow the on-screen instructions to install R.

1.4 Installing RStudio

RStudio is an integrated development environment (IDE) that makes working with R much easier and more productive.

1. **Go to the RStudio website:** https://www.rstudio.com/products/rstudio/download/
2. **Download RStudio Desktop:** Select the free RStudio Desktop version.
3. **Choose your operating system:** Select the appropriate installer for your system.
4. **Run the installer:** Follow the on-screen instructions to install RStudio.

1.5 Exploring RStudio

Open RStudio. You'll see four main panels:

1. **Source Editor (Top-Left):** This is where you write and save your R scripts.
2. **Console (Bottom-Left):** This is where you execute individual R commands and see the output.
3. **Environment/History (Top-Right):**
   - **Environment:** Shows a list of loaded data, variables, and functions.
   - **History:** Shows a history of commands you've entered.
4. **Files/Plots/Packages/Help (Bottom-Right):**
   - **Files:** Allows you to browse your computer's file system.
   - **Plots:** Displays any plots or graphs you create.
   - **Packages:** Allows you to install, update, and load R packages.
   - **Help:** Provides documentation for R functions and packages.

2 Basics of R programming for Data Analysis

Reference: Kabacoff (2022), Mayor (2015).

Good Text book

[R for Data Science]https://r4ds.hadley.nz/

2.1 Running Basic R Commands

1. **In the Console panel:** Type the following commands and press Enter after each one:

   ```
   1 + 1
   ```

   ```
   [1] 2
   ```

   ```
   x <- 10
   print(x)
   ```

   ```
   [1] 10
   ```

   You should see the output of the commands printed in the console.

2. **Create a new R Script:**

   - Click File -> New File -> R Script.
   - Type the same commands from above into the script.
   - Save the script as `lesson1.R`.

3. **Run the script:**

   - Click the "Source" button in the script editor (or press Ctrl+Shift+Enter).
   - The commands in the script will be executed in the console.

2.2 Installing and Loading Packages

R packages are collections of functions, data, and documentation that extend the capabilities of R. The `tidyverse` package is a collection of popular packages for data science.

1. **Install the `tidyverse` package:** In the Console, type the following command and press Enter:

   ```
   install.packages("tidyverse")
   ```

   R will download and install the `tidyverse` package and its dependencies. This may take a few minutes.

2. **Load the `tidyverse` package:** In the Console or in your script, type the following command and press Enter:

   ```
   library(tidyverse)
   ```

   This loads the `tidyverse` package into your R session, making its functions available for use.

2.3 Downloading Sample Data

We'll use a sample CSV file for demonstration.

- **Download the `exam_scores.csv` file** from the course materials to your data directory. You can also copy this link for downloading directly into R.: Sample CSV Data

2.4 Inspecting Data

Now, let's read the `exam_scores.csv` file into R and inspect it:

```
#Replace this link with your actual link to your data.

exam_scores <- read.csv("https://raw.githubusercontent.com/sijuswamyresearch/R-for-Data-Analy

#Display the first few rows.
head(exam_scores)
```

```
  student_id study_hours score grade
1          1          NA    65     C
2          2           5    88     B
3          3           1    52     F
4          4           3    76     c
5          5           4    82     B
6          6           2   100     C
```

2.5 Data Input and Data Types

2.5.1 Data Types in R

R supports several fundamental data types:

1. **Numeric:** Numbers (e.g., `1`, `3.14`, `-2.5`).
2. **Character:** Text strings (e.g., `"hello"`, `"Data Analysis"`).
3. **Factor:** Categorical variables (e.g., `"Low"`, `"Medium"`, `"High"`). Factors are important for statistical analysis.
4. **Logical:** Boolean values, `TRUE` or `FALSE`.
5. **Date:** Dates and times (e.g., `"2023-10-27"`).

2.5.2 Checking Data Types

The `class()` function tells you the data type of a variable:

```r
x <- 10
class(x)
```

```
[1] "numeric"
```

```r
y <- "hello"
class(y)
```

```
[1] "character"
```

2.5.3 Converting Data Types

You can convert between data types using the `as.*()` functions:

`as.numeric()`

`as.character()`

`as.factor()`

`as.logical()`

`as.Date()`

Example:

```r
x <- "123"
class(x)
```

```
[1] "character"
```

```r
x_numeric <- as.numeric(x)
class(x_numeric)
```

```
[1] "numeric"
```

Important Note:

Converting a character string that doesn't represent a number to numeric will result in NA.

2.6 Lists, Arrays, and Data Frames in R

2.7 1. Lists

Lists are versatile data structures that can hold elements of different types. A list can contain numbers, strings, vectors, arrays, or even other lists.

2.7.1 Creating Lists

You can create a list using the `list()` function.

2.7.1.1 Example 1: Creating a Simple List

```r
# Creating a simple list
my_list <- list(name = "John", age = 30, grades = c(85, 90, 78))
print(my_list)
```

```
$name
[1] "John"

$age
[1] 30

$grades
[1] 85 90 78
```

2.7.2 Accessing List Elements

You can access list elements using their names or indices.

2.7.2.1 Example 2: Accessing List Elements by Name

```r
# Accessing list elements by name
name <- my_list$name
age <- my_list$age
print(paste("Name:", name))
```

```
[1] "Name: John"
```

```r
print(paste("Age:", age))
```

```
[1] "Age: 30"
```

2.7.2.2 Example 3: Accessing List Elements by Index

```r
# Accessing list elements by index
name <- my_list[[1]]
age <- my_list[[2]]
print(paste("Name:", name))
```

```
[1] "Name: John"
```

```r
print(paste("Age:", age))
```

```
[1] "Age: 30"
```

2.7.3 Modifying Lists

You can modify lists by adding, updating, or deleting elements.

```r
# Adding elements to a list
my_list$city <- "New York"
print(my_list)
```

```
$name
[1] "John"

$age
[1] 30

$grades
[1] 85 90 78

$city
[1] "New York"
```

```r
# Updating list elements
my_list$age <- 31
print(my_list)
```

```
$name
[1] "John"

$age
[1] 31

$grades
[1] 85 90 78

$city
[1] "New York"
```

```
# Deleting list elements
my_list$grades <- NULL
print(my_list)
```

```
$name
[1] "John"

$age
[1] 31

$city
[1] "New York"
```

2.8 2. Arrays

Arrays are data structures that can hold elements of the same type in multiple dimensions.

2.8.1 Creating Arrays

You can create an array using the `array()` function.

```
#one dimensional array
a1=array(1:10)
print(a1)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```r
# Creating a 2D array (matrix)
my_matrix <- array(data = 1:9, dim = c(3, 3))
print(my_matrix)
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```r
# creating matrix using matrix function
m1=matrix(c(1,2,3,4,5,6,7,8,9),ncol=3,byrow=T)
m1
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```r
# Creating a 3D array
my_3d_array <- array(data = 1:27, dim = c(3, 3, 3))
print(my_3d_array)
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

, , 3

     [,1] [,2] [,3]
[1,]   19   22   25
```

```
[2,]   20   23   26
[3,]   21   24   27
```

```
# Accessing array elements
my_matrix
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
element <- my_matrix[2, 3]
print(paste("Element at (2, 3):", element))
```

```
[1] "Element at (2, 3): 8"
```

```
# Modifying array elements
my_matrix[1, 1] <- 10
print(my_matrix)
```

```
     [,1] [,2] [,3]
[1,]   10    4    7
[2,]    2    5    8
[3,]    3    6    9
```

2.8.2 Array Operations

You can perform various operations on arrays, such as transposing and performing arithmetic operations.

```
# Transposing a matrix
transposed_matrix <- t(my_matrix)
print(transposed_matrix)
```

```
     [,1] [,2] [,3]
[1,]   10    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```
# Creating another matrix
another_matrix <- array(data = 10:18, dim = c(3, 3))

# Matrix multiplication
multiplied_matrix <- my_matrix %*% another_matrix
print(multiplied_matrix)
```

```
     [,1] [,2] [,3]
[1,]  228  291  354
[2,]  171  216  261
[3,]  204  258  312
```

2.9 3. Data Frames

Data frames are table-like data structures that organize data into rows and columns. Each column can hold data of a different type.

You can create a data frame using the `data.frame()` function.

```
# Creating a data frame
my_data_frame <- data.frame(
  id = 1:3,
  name = c("Alice", "Bob", "Charlie"),
  age = c(25, 30, 28),
  score = c(85, 92, 78)
)
print(my_data_frame)
```

```
  id    name age score
1  1   Alice  25    85
2  2     Bob  30    92
3  3 Charlie  28    78
```

2.9.1 Accessing Data Frame Elements

You can access data frame elements using their column names or indices.

```r
# Accessing data frame columns by name
names <- my_data_frame$name
ages <- my_data_frame$age
print(names)
```

```
[1] "Alice"   "Bob"     "Charlie"
```

```r
print(ages)
```

```
[1] 25 30 28
```

2.9.2 Modifying Data Frames

You can modify data frames by adding, updating, or deleting columns and rows.

```r
# Adding a column to a data frame
my_data_frame$city <- c("New York", "Los Angeles", "Chicago")
print(my_data_frame)
```

```
  id    name age score        city
1  1   Alice  25    85    New York
2  2     Bob  30    92 Los Angeles
3  3 Charlie  28    78     Chicago
```

```r
# Updating data frame elements
my_data_frame$age[1] <- 26
print(my_data_frame)
```

```
  id    name age score        city
1  1   Alice  26    85    New York
2  2     Bob  30    92 Los Angeles
3  3 Charlie  28    78     Chicago
```

```r
# Deleting a column from a data frame
my_data_frame$city <- NULL
print(my_data_frame)
```

```
  id    name age score
1  1   Alice  26    85
2  2     Bob  30    92
3  3 Charlie  28    78
```

```r
# Adding rows to a data frame
new_row <- data.frame(id = 4, name = "David", age = 32, score = 90)
my_data_frame <- rbind(my_data_frame, new_row)
print(my_data_frame)
```

```
  id    name age score
1  1   Alice  26    85
2  2     Bob  30    92
3  3 Charlie  28    78
4  4   David  32    90
```

```r
#adding a column using cbind
new_col=data.frame(city=c("Kottayam","Ettumanoor","Elanji","Muvattupuzha"))
my_data_frame=cbind(my_data_frame,new_col)
my_data_frame
```

```
  id    name age score         city
1  1   Alice  26    85     Kottayam
2  2     Bob  30    92   Ettumanoor
3  3 Charlie  28    78       Elanji
4  4   David  32    90 Muvattupuzha
```

```r
# Deleting rows from a data frame
my_data_frame <- my_data_frame[-4, ]
print(my_data_frame)
```

```
  id    name age score       city
1  1   Alice  26    85   Kottayam
2  2     Bob  30    92 Ettumanoor
3  3 Charlie  28    78     Elanji
```

2.9.3 Data Frame Operations

You can perform various operations on data frames, such as subsetting, filtering, sorting, and merging.

```r
# Subsetting data frames
subset_df <- my_data_frame[, c("name", "score")]
print(subset_df)
```

```
     name score
1   Alice    85
2     Bob    92
3 Charlie    78
```

```r
# Filtering data frames
filtered_df <- my_data_frame[my_data_frame$age > 28, ]
print(filtered_df)
```

```
  id name age score       city
2  2  Bob  30    92 Ettumanoor
```

```r
# Sorting data frames
sorted_df <- my_data_frame[order(my_data_frame$age), ]
print(sorted_df)
```

```
  id    name age score       city
1  1   Alice  26    85   Kottayam
3  3 Charlie  28    78     Elanji
2  2     Bob  30    92 Ettumanoor
```

3 Data Cleaning and Transformation

3.1 Introduction

In this lesson, we'll tackle the often-messy reality of real-world data: dirty data. We'll learn how to identify common data quality issues like missing values, outliers, and inconsistencies, and then apply techniques in R to clean and transform the data into a usable format. This lesson will also focus on cloud deployments with version control.

3.2 Learning Objectives

- Identify common data quality issues.
- Detect missing values and apply appropriate handling methods (removal, imputation).
- Detect outliers and apply appropriate handling methods (trimming, capping, transformation).
- Correct inconsistent data entries (e.g., typos, inconsistent formatting).
- Commit and upload the project again to version control.

3.3 What is Dirty Data?

Dirty data refers to data that is inaccurate, incomplete, inconsistent, or otherwise unreliable. Common sources of dirty data include:

- Human error during data entry
- Data integration issues from multiple sources
- Software bugs
- Inconsistent data standards

3.4 The `exam_scores` Dataset: Let's Get Specific

We'll use our `exam_scores` dataset (or a modified version with intentional errors) to illustrate these cleaning techniques. We'll assume the dataset contains columns like:

- `student_id`: Unique identifier for each student.
- `study_hours`: Number of hours spent studying.
- `score`: Exam score (out of 100).
- `grade`: Letter grade (A, B, C, D, F).

3.5 Identifying Data Quality Issues

1. **Missing Values (NA):**

   - Let's check for missing values in the `score` column:

```r
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.2.3

Warning: package 'ggplot2' was built under R version 4.2.3

Warning: package 'tibble' was built under R version 4.2.3

Warning: package 'tidyr' was built under R version 4.2.3

Warning: package 'readr' was built under R version 4.2.3

Warning: package 'purrr' was built under R version 4.2.3

Warning: package 'dplyr' was built under R version 4.2.3

Warning: package 'stringr' was built under R version 4.2.3

Warning: package 'forcats' was built under R version 4.2.3

Warning: package 'lubridate' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becon
```

```
    exam_scores <- read.csv("https://raw.githubusercontent.com/sijuswamyresearch/R-for-Data-A

    sum(is.na(exam_scores))
```

```
[1] 1
```

cleaning the NAs

```
#Remove NA values in Exam Scores
exam_scores <- na.omit(exam_scores)
exam_scores
```

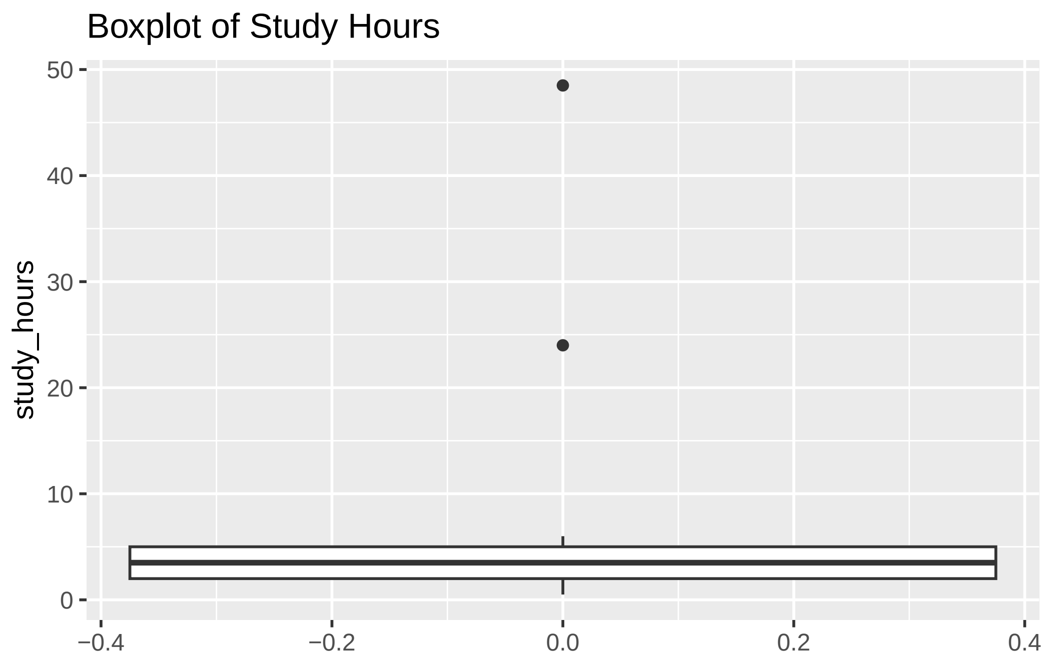
```
   student_id study_hours score grade
2           2         5.0    88     B
3           3         1.0    52     F
4           4         3.0    76     c
5           5         4.0    82     B
6           6         2.0   100     C
7           7         6.0    95     A
8           8         1.5    58     F
9           9         3.5    79     C
10         10         4.5    85     B
11         11         2.5    73     C
12         12         5.5    91     A
13         13         0.5    45     f
14         14         3.0    77     C
15         15         4.0    83     B
16         16        24.0    68     C
17         17         6.0    96     A
18         18         1.0    55     F
```

```
19          19          3.5     10      B
20          20          4.5     86      B
21          21          2.5     74      C
22          22          5.5     92      A
23          23          0.5     48      F
24          24          3.0     78      C
25          25          4.0     84      B
26          26          2.0     69      C
27          27          6.0     97      A
28          28          1.0     56      F
29          29          3.5     81      B
30          30         48.5     87      b
```

2. **Outliers:**

   - Let's identify potential outliers in the `study_hours` column using a boxplot:

```r
ggplot(exam_scores, aes(y = study_hours)) +
  geom_boxplot() +
  labs(title = "Boxplot of Study Hours")
```



   - We can then calculate the IQR and identify values outside the typical range:

```
Q1 <- quantile(exam_scores$study_hours, 0.25)
Q3 <- quantile(exam_scores$study_hours, 0.75)
IQR <- Q3 - Q1

lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR

outliers <- exam_scores %>%
  filter(study_hours < lower_bound | study_hours > upper_bound)

print(outliers)
```

```
  student_id study_hours score grade
1         16        24.0    68     C
2         30        48.5    87     b
```

3. **Inconsistent Data:**

   - Let's check for inconsistent grade entries (e.g., lowercase "a" instead of uppercase "A"):

```
unique(exam_scores$grade) #See if the dataset has what you expect
```

```
[1] "B" "F" "c" "C" "A" "f" "b"
```

3.6 Handling Missing Values

1. **Removal:** Let's remove rows where the exam score is NA.

```
exam_scores_no_na <- exam_scores %>%
  filter(!is.na(score))
```

2. **Imputation:** We can use a simple `ifelse` statement to impute with the mean.

   - Mean/Median Imputation: Replace with the mean or median of the column.

```
exam_scores <- exam_scores %>%
  mutate(score = ifelse(is.na(score), mean(score, na.rm = TRUE), score))
```

3.7 Handling Outliers

We can handle those extreme scores to help reduce the variability of the dataset.

1. **Capping (Winsorizing):**

```r
# Cap values above the 95th percentile for study hours.
upper_threshold <- quantile(exam_scores$study_hours, 0.95)
exam_scores <- exam_scores %>%
  mutate(study_hours = ifelse(study_hours > upper_threshold, upper_threshold, study_hour
print(exam_scores)
```

```
   student_id study_hours score grade
2           2         5.0    88     B
3           3         1.0    52     F
4           4         3.0    76     c
5           5         4.0    82     B
6           6         2.0   100     C
7           7         6.0    95     A
8           8         1.5    58     F
9           9         3.5    79     C
10         10         4.5    85     B
11         11         2.5    73     C
12         12         5.5    91     A
13         13         0.5    45     f
14         14         3.0    77     C
15         15         4.0    83     B
16         16        16.8    68     C
17         17         6.0    96     A
18         18         1.0    55     F
19         19         3.5    10     B
20         20         4.5    86     B
21         21         2.5    74     C
22         22         5.5    92     A
23         23         0.5    48     F
24         24         3.0    78     C
25         25         4.0    84     B
26         26         2.0    69     C
27         27         6.0    97     A
28         28         1.0    56     F
29         29         3.5    81     B
30         30        16.8    87     b
```

3.8 Correcting Inconsistent Data

Text Cleaning: Let's apply some cleaning techniques to ensure the dataset is formatted as best as possible.

```r
library(stringr)

exam_scores <- exam_scores %>%
  mutate(grade = str_to_upper(grade))

print(unique(exam_scores$grade)) #Check if things are good now!
```

```
[1] "B" "F" "C" "A"
```

4 Descriptive Statistics

4.1 Introduction

In this section, we'll delve into the world of descriptive statistics. Descriptive statistics provide a summary of your data, allowing you to understand its central tendency, variability, and shape. We'll learn how to calculate common descriptive statistics in R and interpret their meaning.

4.2 Learning Objectives

- Calculate measures of central tendency (mean, median, mode).
- Calculate measures of dispersion (standard deviation, variance, IQR, range).
- Use `dplyr::summarize()` to efficiently calculate descriptive statistics.
- Understand the relationship between descriptive statistics and data distribution.
- Calculate descriptive statistics for your cloud hosted R project.
- Commit changes to cloud hosted R project with descriptive statistics.

4.3 Measures of Central Tendency

Measures of central tendency describe the "center" of a dataset.

1. **Mean:** The average of all values.

   - Calculated as the sum of the values divided by the number of values.

   ```
   scores <- c(75, 80, 92, 68, 85)
   mean(scores)
   ```

   ```
   [1] 80
   ```

2. **Median:** The middle value when the data is sorted.

   - If there are an even number of values, the median is the average of the two middle values.

```r
median(scores)
```

```
[1] 80
```

3. **Mode:** The most frequent value in the dataset.

   - R doesn't have a built-in function to calculate the mode directly, so we'll create one:

```r
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

values <- c(1, 2, 2, 3, 4, 4, 4, 5)
getmode(values)
```

```
[1] 4
```

When to Use Which Measure:

- **Mean:** Sensitive to outliers (extreme values). Use when the data is relatively symmetrical and has no extreme outliers.
- **Median:** Robust to outliers. Use when the data has outliers or is skewed.
- **Mode:** Useful for categorical data or discrete data with repeating values.

4.4 Measures of Dispersion

Measures of dispersion describe the spread or variability of a dataset.

1. **Standard Deviation:** A measure of how spread out the data is around the mean.

   - A higher standard deviation indicates greater variability.

```r
sd(scores)
```

```
[1] 9.192388
```

2. **Variance:** The square of the standard deviation.

   - Also measures variability, but is less interpretable than standard deviation.

```r
var(scores)
```

```
[1] 84.5
```

3. **Interquartile Range (IQR):** The difference between the 75th percentile (Q3) and the 25th percentile (Q1).

    - Robust to outliers and provides a measure of the spread of the middle 50% of the data.

    ```r
    IQR(scores)
    ```

    ```
    [1] 10
    ```

4. **Range:** The difference between the maximum and minimum values.

    ```r
    range(scores)
    ```

    ```
    [1] 68 92
    ```

    ```r
    diff(range(scores)) #Calculate the range from the output
    ```

    ```
    [1] 24
    ```

4.5 Descriptive Statistics with `dplyr::summarize()`

The `dplyr` package provides the `summarize()` function, which makes it easy to calculate multiple descriptive statistics at once:

```r
library(dplyr)
```

```
Warning: package 'dplyr' was built under R version 4.2.3


Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
#Replace this file with your local file.
exam_scores <- read.csv("https://raw.githubusercontent.com/sijuswamyresearch/R-for-Data-Analy

exam_scores %>%
  summarize(
    mean_score = mean(score),
    median_score = median(score),
    sd_score = sd(score),
    iqr_score = IQR(score),
    min_score = min(score),
    max_score = max(score)
  )
```

```
  mean_score median_score sd_score iqr_score min_score max_score
1   74.33333         78.5 19.38598        21        10       100
```

4.5.1 Grouped Descriptive Statistics

You can calculate descriptive statistics for different groups within your data using `dplyr::group_by()` in combination with `summarize()`:

```
exam_scores %>%
  group_by(grade) %>%
  summarize(
    mean_score = mean(score),
    median_score = median(score),
    sd_score = sd(score)
  )
```

```
# A tibble: 7 x 4
  grade mean_score median_score sd_score
  <chr>      <dbl>        <dbl>    <dbl>
1 A           94.2         95       2.59
2 B           74.9         83.5    26.3
3 C           75.9         74      10.2
4 F           53.8         55       3.90
5 b           87           87      NA
6 c           76           76      NA
7 f           45           45      NA
```

4.6 Descriptive Statistics and Data Distribution

Descriptive statistics provide insights into the distribution of your data:

- **Symmetrical Distribution:**
  - Mean   Median   Mode
  - Standard deviation is relatively small.
- **Right-Skewed Distribution:**
  - Mean > Median > Mode
  - Long tail on the right side.
- **Left-Skewed Distribution:**
  - Mean < Median < Mode
  - Long tail on the left side.

4.7 Practice

1. Load the `exam_scores.csv` dataset (or your own dataset).
2. Calculate the mean, median, standard deviation, IQR, and range for a numerical column.
3. Create a custom function to calculate the mode.
4. Calculate descriptive statistics for different groups within the data (e.g., by gender, by region).
5. Describe the shape of the data based on the descriptive statistics you calculated.

5 Exploratory Data Analysis

5.1 Visualizing a Single Variable

In this section, we will delve into the techniques for visualizing a single variable, which is a fundamental aspect of exploratory data analysis (EDA). Understanding the distribution and characteristics of individual variables is crucial before exploring relationships between multiple variables.

5.2 Introduction to Visualizing a Single Variable

Visualizing a single variable helps us understand its distribution, central tendency, and spread. This provides valuable insights into the data's characteristics and potential anomalies. In this section, we will explore different types of plots suitable for single-variable visualization using `ggplot2`.

5.2.1 Histograms

Histograms are used to visualize the distribution of a numerical variable by dividing the data into bins and counting the number of observations in each bin.

5.2.1.1 Example 1: Creating a Histogram

```r
library(tidyverse)
```

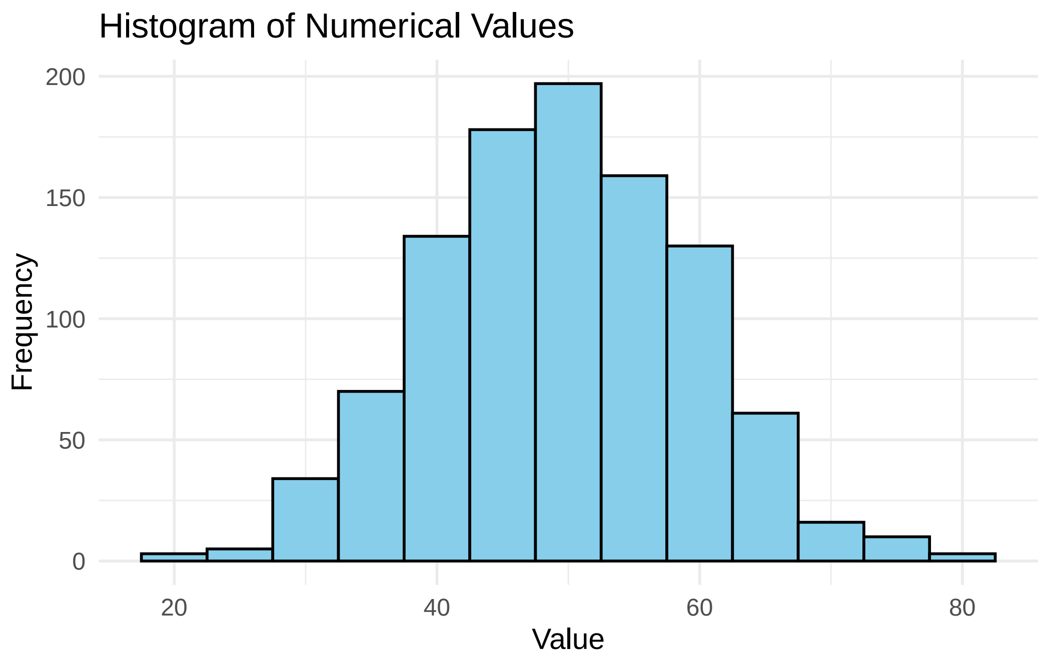
```
Warning: package 'tidyverse' was built under R version 4.2.3
```

```
Warning: package 'ggplot2' was built under R version 4.2.3
```

```
Warning: package 'tibble' was built under R version 4.2.3
```

```
Warning: package 'tidyr' was built under R version 4.2.3

Warning: package 'readr' was built under R version 4.2.3

Warning: package 'purrr' was built under R version 4.2.3

Warning: package 'dplyr' was built under R version 4.2.3

Warning: package 'stringr' was built under R version 4.2.3

Warning: package 'forcats' was built under R version 4.2.3

Warning: package 'lubridate' was built under R version 4.2.3

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becon
```

```r
# Sample data (replace with your data)
data <- data.frame(
  values = rnorm(1000, mean = 50, sd = 10)
)

# Creating a histogram
ggplot(data, aes(x = values)) +
  geom_histogram(binwidth = 5, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Numerical Values",
       x = "Value",
       y = "Frequency") +
  theme_minimal()
```

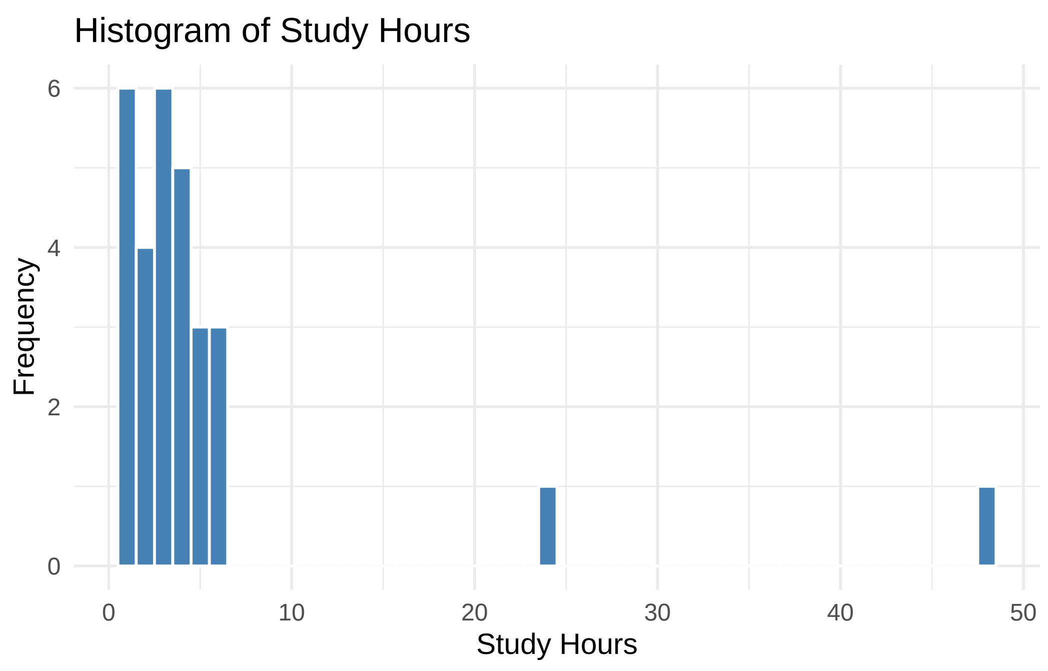


5.2.2 Histograms

Histograms are an extremely common way to visualize the distribution of a single numerical variable. It groups the data into bins, and displays the frequency of the data in each bin as a vertical bar. This type of chart is extremely common, and is one of the easiest way to visualize your data.

```r
exam_scores <- read.csv("https://raw.githubusercontent.com/sijuswamyresearch/R-for-Data-Analy
ggplot(exam_scores, aes(x = study_hours)) +
  geom_histogram(binwidth = 1, fill = "steelblue", color = "white") +
  labs(title = "Histogram of Study Hours", x = "Study Hours", y = "Frequency") +
  theme_minimal()
```

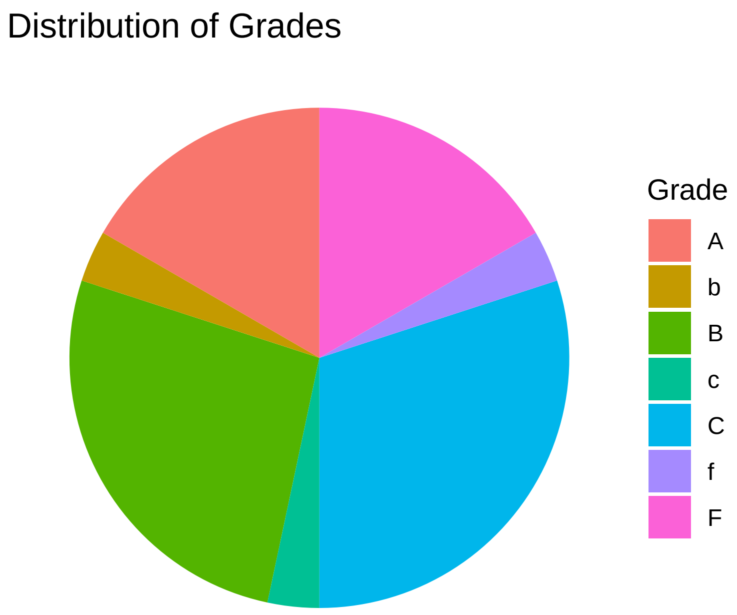
```
Warning: Removed 1 rows containing non-finite values (`stat_bin()`).
```



5.2.3 Pie Charts

Pie charts are commonly used to visualize the proportions or percentages of different categories within a categorical variable. While simple, they can be less effective than other chart types when dealing with many categories or subtle differences in proportions.

```
#Create a data frame to display
exam_scores <- read.csv("https://raw.githubusercontent.com/sijuswamyresearch/R-for-Data-Analy
grade_counts <- exam_scores %>%
  group_by(grade) %>%
  summarize(count = n())

ggplot(grade_counts, aes(x = "", y = count, fill = grade)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  labs(title = "Distribution of Grades", fill = "Grade") +
  theme_void()
```



Explanation:

- `stat = "identity"`: Tells `geom_bar` to use the provided `count` values directly, instead of counting them.
- `coord_polar("y", start = 0)`: Transforms the bar chart into a pie chart.
- `theme_void()`: Removes unnecessary chart elements for a cleaner look.

Interpreting Pie Charts:

- **Proportions:** The size of each slice represents the proportion of that category within the whole.
- **Dominant Categories:** Easily identify the largest and smallest categories.

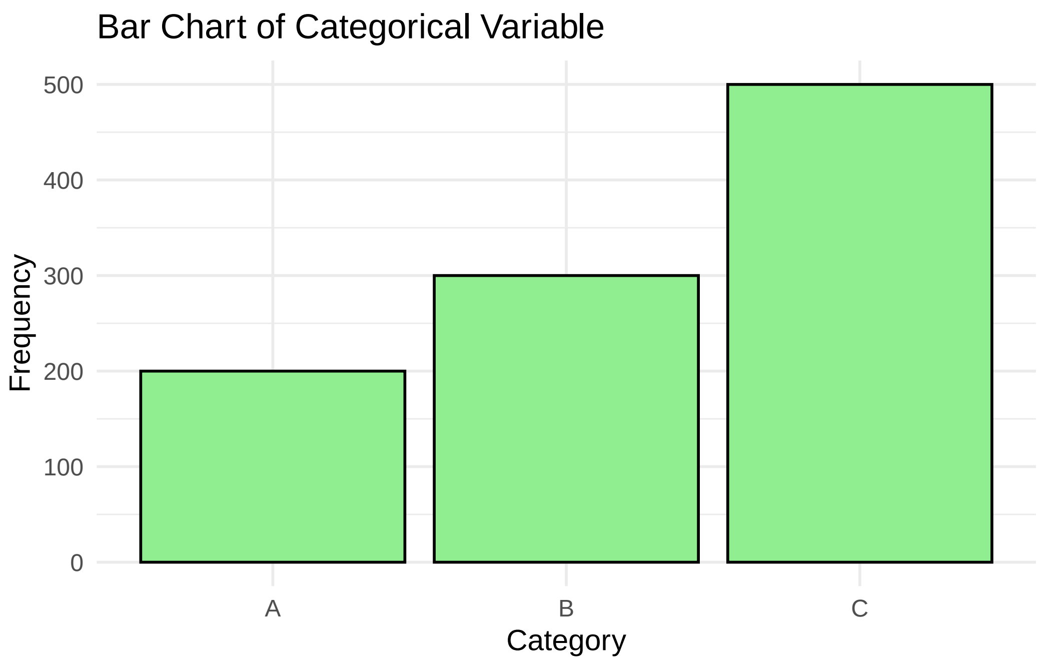
ℹ Note

Pie charts can be difficult to interpret accurately, especially with many slices of similar size. Consider using bar charts or other visualizations for better clarity.

5.2.4 Bar Charts

Bar charts are used to visualize the distribution of a categorical variable by displaying the frequency or proportion of each category.

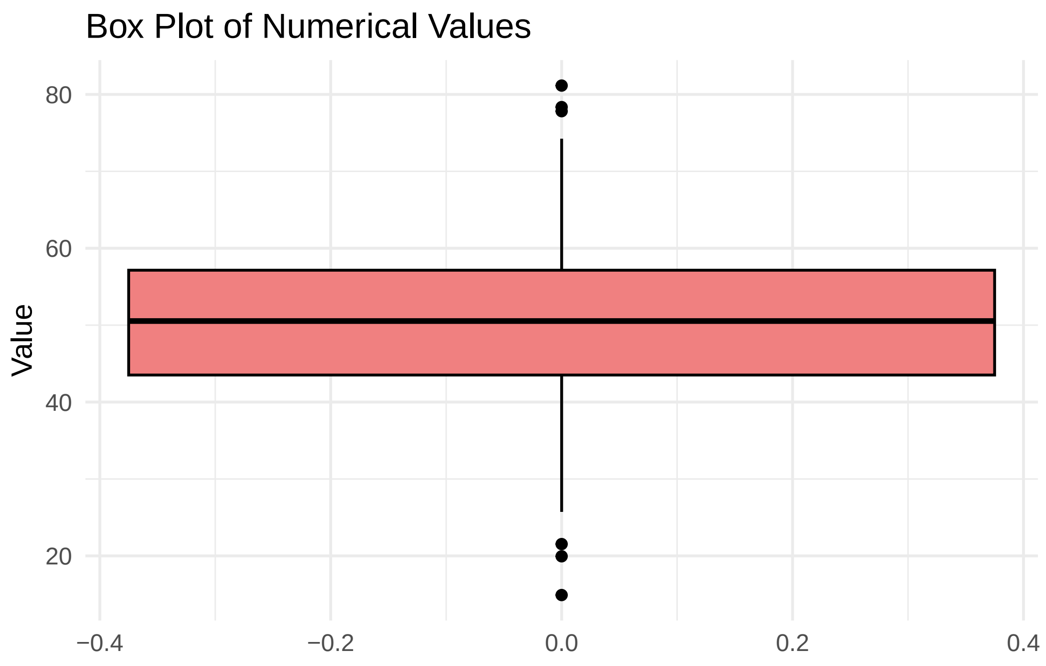
```
# Sample data
data <- data.frame(
  categories = factor(rep(c("A", "B", "C"), times = c(200, 300, 500)))
)

# Creating a bar chart
ggplot(data, aes(x = categories)) +
  geom_bar(fill = "lightgreen", color = "black") +
  labs(title = "Bar Chart of Categorical Variable",
       x = "Category",
       y = "Frequency") +
  theme_minimal()
```



5.2.5 Box Plots

Box plots provide a concise summary of the distribution of a numerical variable, showing the median, quartiles, and potential outliers.

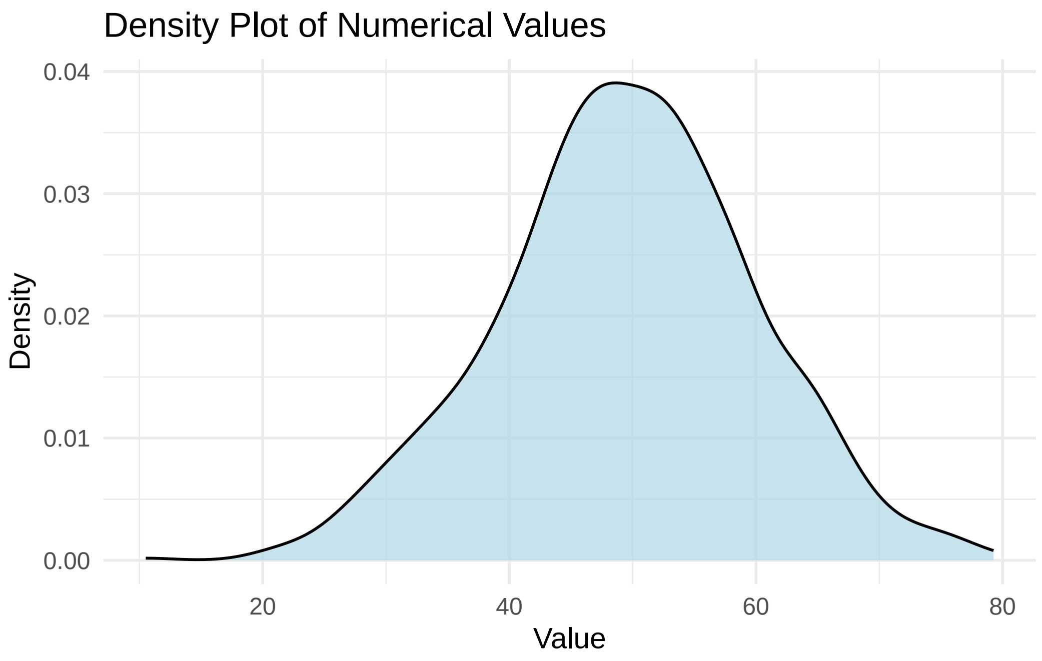
```
# Sample data
data <- data.frame(
  values = rnorm(1000, mean = 50, sd = 10)
)

# Creating a box plot
ggplot(data, aes(y = values)) +
  geom_boxplot(fill = "lightcoral", color = "black") +
  labs(title = "Box Plot of Numerical Values",
       y = "Value") +
  theme_minimal()
```



5.2.6 Density Plots

Density plots provide a smoothed representation of the distribution of a numerical variable.

```
# Sample data
data <- data.frame(
  values = rnorm(1000, mean = 50, sd = 10)
)

# Creating a density plot
ggplot(data, aes(x = values)) +
  geom_density(fill = "lightblue", alpha = 0.7) +
  labs(title = "Density Plot of Numerical Values",
       x = "Value",
       y = "Density") +
  theme_minimal()
```



Choosing the Right Visualization

- **Numerical Variable:** Histograms, box plots, and density plots are suitable for visualizing numerical variables.
- **Categorical Variable:** Bar charts are used to visualize categorical variables.

5.3 Practice

1. Load a dataset containing both numerical and categorical variables.
2. Create histograms, bar charts, box plots, and density plots to visualize single variables.
3. Interpret the characteristics and distribution of each variable based on the visualizations.
4. Identify potential outliers or anomalies.

5.4 Bivariate visualization

In this section, we'll extend our exploration of data by examining relationships between two variables simultaneously. This is called bivariate exploratory data analysis (EDA). We'll focus on using `ggplot2` to create visualizations that reveal patterns, correlations, and dependencies between variables.

5.5 What is Bivariate EDA?

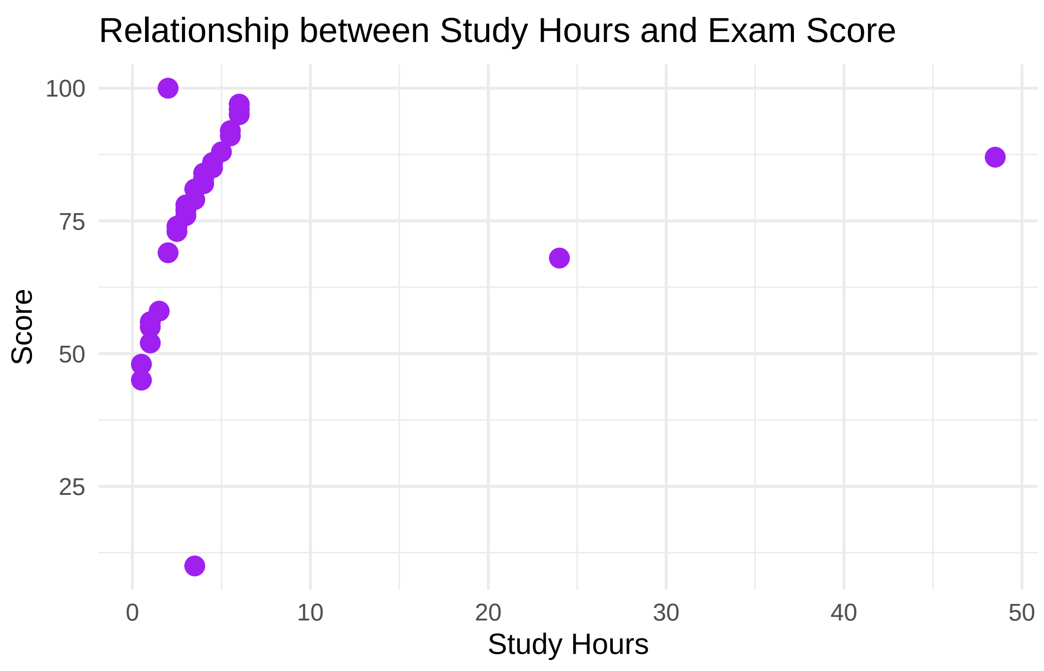
Bivariate EDA helps you answer questions like:

- Is there a relationship between these two variables?
- How strong is the relationship?
- Is the relationship positive or negative?
- Does the relationship vary across different groups?

5.6 Scatter Plots

Scatter plots visualize the relationship between two numerical variables. Each point on the plot represents a single observation, with its position determined by its values for the two variables.

```r
library(tidyverse)

#Replace with your local file, or paste dataset info here
exam_scores <- read.csv("https://raw.githubusercontent.com/sijuswamyresearch/R-for-Data-Analy

ggplot(exam_scores, aes(x = study_hours, y = score)) +
  geom_point(color = "purple", size = 3) +
  labs(title = "Relationship between Study Hours and Exam Score", x = "Study Hours", y = "Sco
  theme_minimal()
```



Explanation:

- `ggplot(exam_scores, aes(x = study_hours, y = score))`: Creates a `ggplot` object, specifying the dataset (`exam_scores`) and the two variables to visualize (`study_hours` and `score`).
- `geom_point()`: Adds a scatter plot layer to the plot.
- `color = "purple", size = 3`: Sets the color and size of the points.
- `labs(...)`: Adds a title and axis labels to the plot.
- `theme_minimal()`: Applies a minimalist theme for a cleaner look.

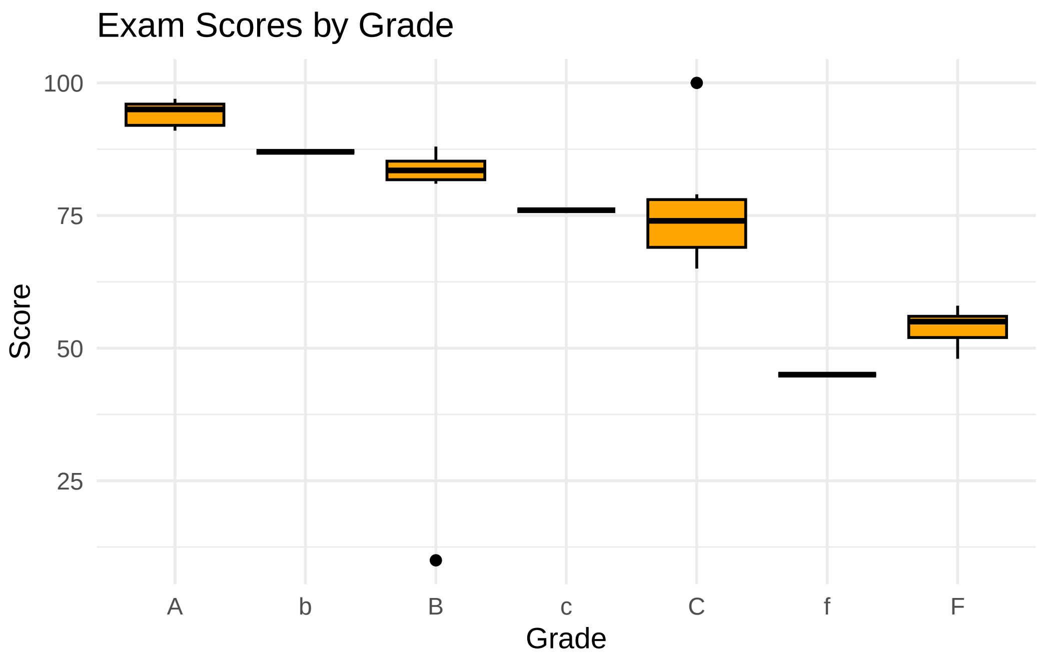
Interpreting Scatter Plots:

- **Direction:** Is the relationship positive (as x increases, y increases), negative (as x increases, y decreases), or non-existent?
- **Strength:** How closely do the points cluster around a line or curve? A tight cluster indicates a strong relationship.
- **Form:** Is the relationship linear, non-linear (curved), or clustered?
- **Outliers:** Are there any points that deviate significantly from the overall pattern?

5.7 Box Plots

Side-by-side box plots compare the distribution of a numerical variable across different categories of a categorical variable.

```r
ggplot(exam_scores, aes(x = grade, y = score)) +
  geom_boxplot(fill = "orange", color = "black") +
  labs(title = "Exam Scores by Grade", x = "Grade", y = "Score") +
  theme_minimal()
```



Explanation:

- `aes(x = grade, y = score)`: Maps the `grade` column to the x-axis (categorical variable) and the `score` column to the y-axis (numerical variable).

Interpreting Side-by-Side Box Plots:

- **Median Differences:** Are the medians significantly different across categories?
- **Spread Differences:** Do the categories have different levels of variability?
- **Outliers:** Are there more outliers in some categories than others?
- **Overlap:** How much do the distributions of the categories overlap?

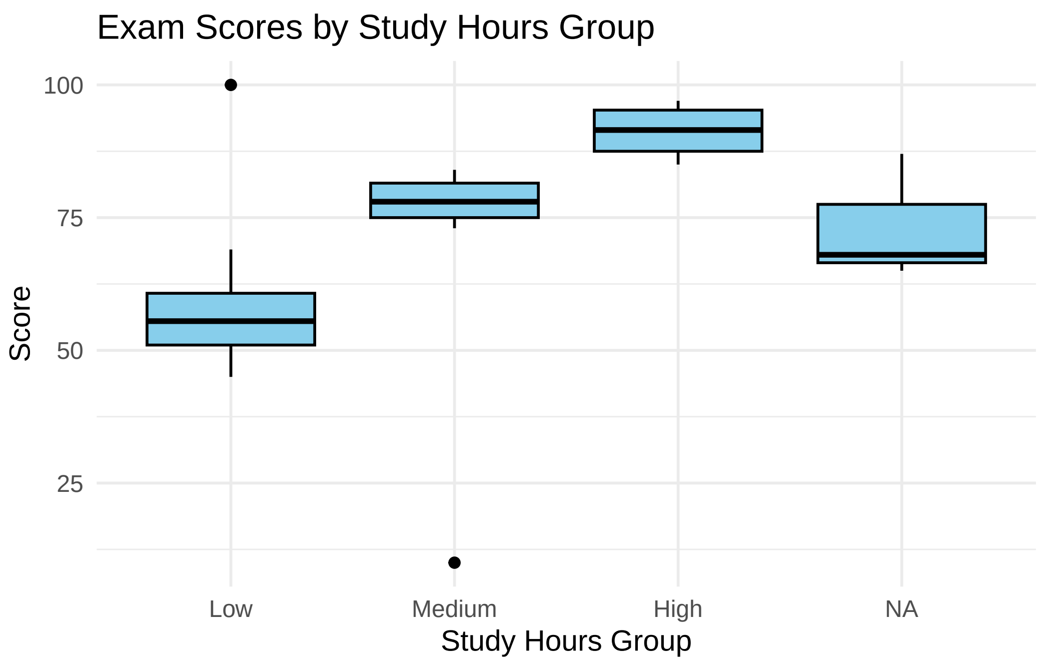
5.7.1 Exploring the Impact of Study Hours: Why Group?

We've seen how `grade` (a letter grade) relates to `score` using a side-by-side boxplot. Now, let's consider the impact of *study hours* on exam performance. We suspect that students who study more tend to get higher scores. However, `study_hours` is a numerical variable, and it might not make sense to treat each specific hour value as a separate category. We want to see if students in general can improve a test score given different hours of study.

Instead, we can *group* study hours into meaningful ranges like “Low,” “Medium,” and “High.” This will allow us to compare the distribution of exam scores for students with different levels of study commitment. This also gives the benefit of avoiding a test set that is more confusing, or hard to see the individual data points. This also gives an extra layer of generality, since we may want to test this dataset against other student data.

Here’s how we can create those groups in R and then visualize the relationship with a boxplot.

```r
exam_scores <- exam_scores %>%
  mutate(
    study_hours_group = cut(study_hours,
                            breaks = c(0, 2, 4, 6),
                            labels = c("Low", "Medium", "High"),
                            include.lowest = TRUE)
  )

ggplot(exam_scores, aes(x = study_hours_group, y = score)) +
  geom_boxplot(fill = "skyblue", color = "black") +
  labs(title = "Exam Scores by Study Hours Group", x = "Study Hours Group", y = "Score") +
  theme_minimal()
```



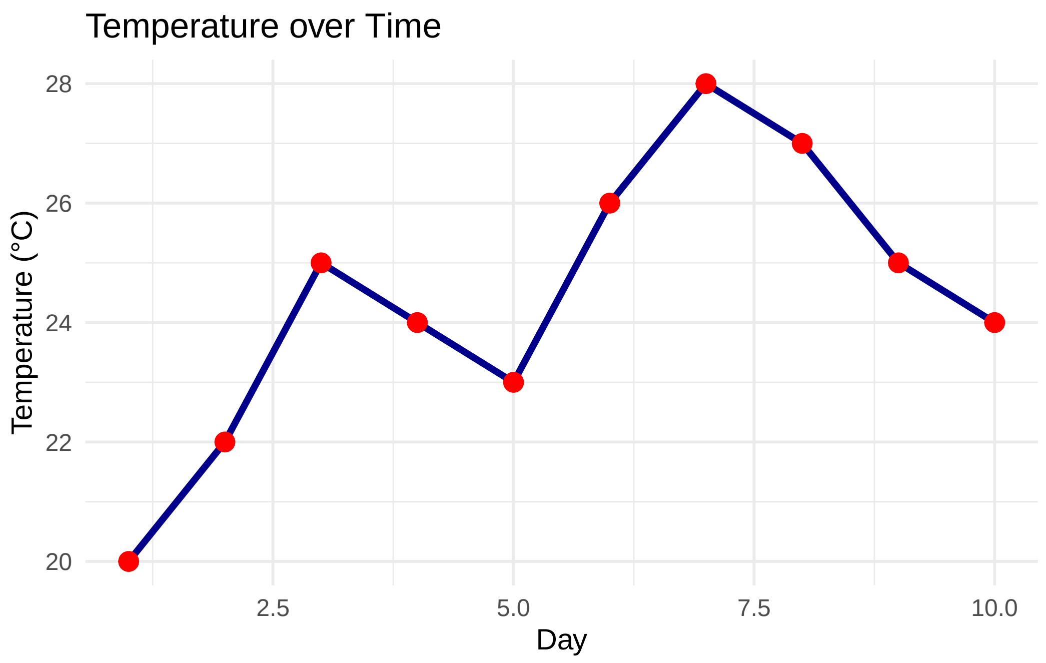
5.8 Line Plots

Line plots visualize trends in a numerical variable over time or across an ordered category.

Note:

Our exam_scores dataset doesn't have a time component, so we'll create a simple example.

```r
#If there isn't a time component, we'll create a simple example.
day <- 1:10
temperature <- c(20, 22, 25, 24, 23, 26, 28, 27, 25, 24)

weather_data <- data.frame(day, temperature)

ggplot(weather_data, aes(x = day, y = temperature)) +
  geom_line(color = "darkblue", linewidth = 1.2) +
  geom_point(color = "red", size = 3) +
  labs(title = "Temperature over Time", x = "Day", y = "Temperature (°C)") +
  theme_minimal()
```



Explanation:

- `geom_line()`: Adds a line connecting the data points.
- `geom_point()`: Adds points at each data point for clarity.

  Interpreting Line Plots:

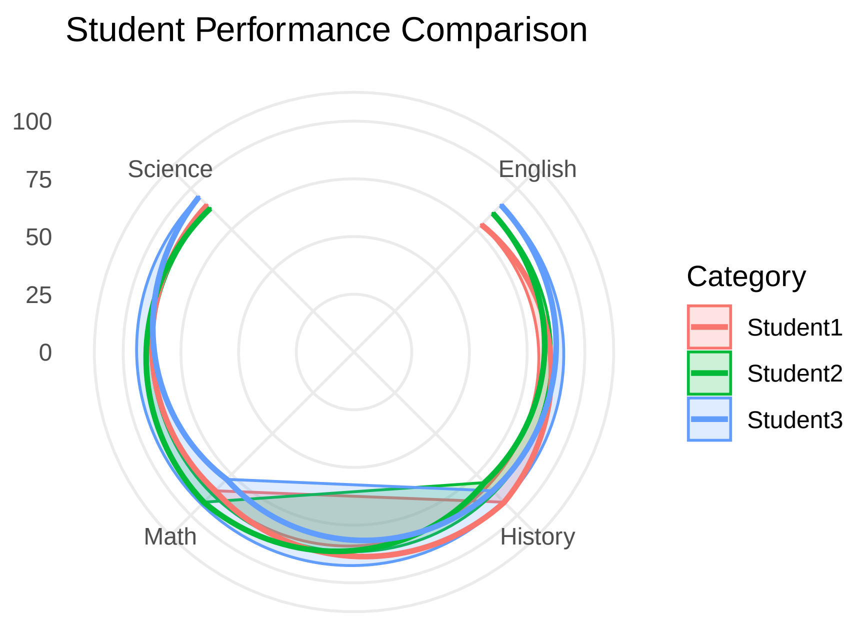
- **Trends:** Is the variable increasing, decreasing, or stable over time?
- **Seasonality:** Are there any repeating patterns?
- **Cyclicality:** Are there any longer-term cycles?
- **Outliers:** Are there any sudden spikes or dips?

5.8.1 Cobweb Plot (Radar Chart)

Cobweb plots, also known as radar charts, are useful for comparing multiple quantitative variables for several different items. The values for each variable are plotted along spokes radiating from a center point. This type of chart works best when comparing items across a limited number of variables (usually less than 10) and when the relative values are more important than the absolute values.

```r
library(tidyverse)

# Create a sample dataset
data <- data.frame(
  Category = c("Student1", "Student2", "Student3"),
  Math = c(85, 92, 78),
  Science = c(90, 88, 95),
  English = c(78, 85, 90),
  History = c(92, 80, 85)
)

# Reshape the data for plotting
data_long <- data %>%
  pivot_longer(
    cols = -Category,  # Exclude the 'Category' column
    names_to = "Variable",
    values_to = "Value"
  )

# Create the radar chart
ggplot(data_long, aes(x = Variable, y = Value, group = Category, color = Category)) +
  geom_polygon(alpha = 0.2, aes(fill = Category)) +  # Fill the area
  geom_line(size = 1) +  # Add lines connecting the data points
  coord_polar() +  # Convert to polar coordinates
  ylim(0, 100) +  # Set the y-axis limits (adjust based on your data)
  labs(title = "Student Performance Comparison",
       x = NULL, y = NULL) +
  theme_minimal()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

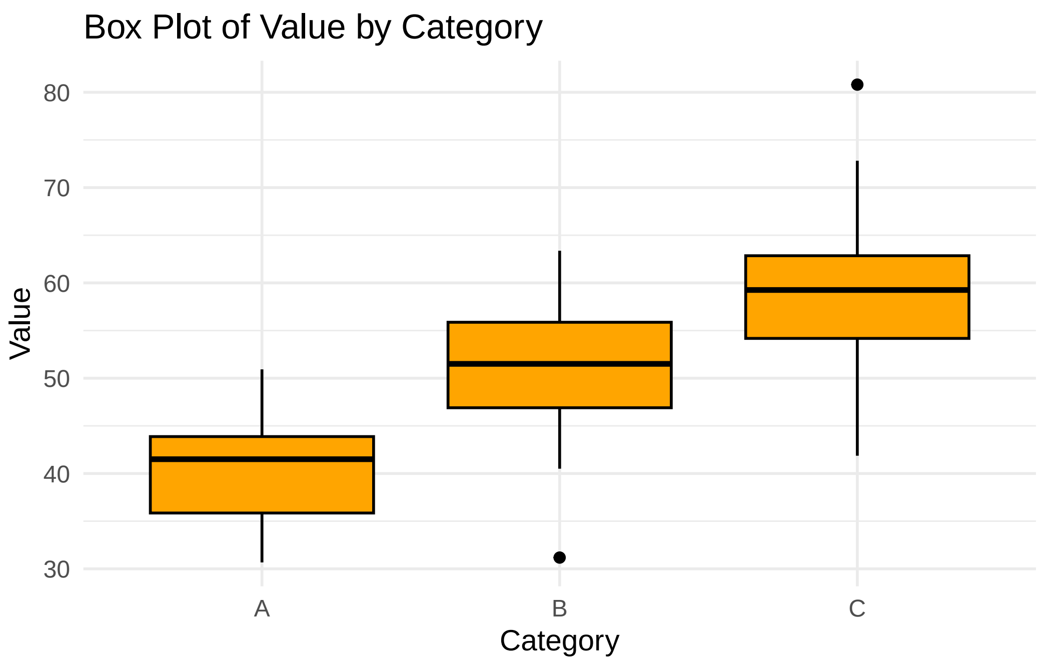


5.9 Multi variable plots in R

In this section, we will explore how to examine multiple variables simultaneously. This is a fundamental step in understanding relationships, dependencies, and patterns within a dataset. We'll cover visualizations and methods that help uncover these insights.

```
# Sample data
data <- data.frame(
  category = factor(rep(c("A", "B", "C"), times = c(30, 35, 35))),
  value = c(rnorm(30, mean = 40, sd = 5),
            rnorm(35, mean = 50, sd = 7),
            rnorm(35, mean = 60, sd = 8))
```

```r
)

# Creating a box plot
ggplot(data, aes(x = category, y = value)) +
  geom_boxplot(fill = "orange", color = "black") +
  labs(title = "Box Plot of Value by Category",
       x = "Category",
       y = "Value") +
  theme_minimal()
```



6 Data Exploration Versus Presentation

6.1 Introduction

In this section, we'll explore the critical distinction between data exploration and data presentation. Both are essential components of the data analysis process, but they have different goals, methods, and considerations. Knowing the difference between them will help you tailor your analysis and communication to the appropriate context and audience.

6.2 Learning Objectives

- Understand the purpose and characteristics of data exploration.
- Understand the purpose and characteristics of data presentation.
- Identify the key differences between data exploration and presentation.
- Choose appropriate visualization methods for exploration vs. presentation.
- Create compelling visuals for data presentation.
- Understand data visualization bias.

6.3 What is Data Exploration?

Data exploration is an iterative process of discovering patterns, relationships, and anomalies in a dataset. It's a detective-like activity, guided by curiosity and a desire to understand the underlying data.

6.3.1 Characteristics of Data Exploration:

- **Goal:** Discover patterns, generate hypotheses, and understand data characteristics.
- **Audience:** Primarily for the analyst/data scientist themselves (or a small team).
- **Methods:**
  - Data cleaning and transformation.
  - Calculation of summary statistics.
  - Visualization (histograms, scatter plots, box plots, etc.).

  - Hypothesis generation.

- **Emphasis:** Flexibility, experimentation, and uncovering insights.
- **Level of Polish:** Often informal, with less emphasis on aesthetics or perfection.

6.4 What is Data Presentation?

Data presentation focuses on communicating findings and insights to a specific audience. It's about telling a compelling story with the data in a clear, concise, and visually appealing manner.

6.4.1 Characteristics of Data Presentation:

- **Goal:** Communicate specific findings, insights, and conclusions to an audience.
- **Audience:** Stakeholders, decision-makers, clients, or the general public.
- **Methods:**
  - Carefully selected visualizations (charts, graphs, tables).
  - Clear and concise narrative.
  - Context and background information.
  - Key takeaways and recommendations.
- **Emphasis:** Clarity, accuracy, visual appeal, and persuasiveness.
- **Level of Polish:** High, with attention to detail in design, labeling, and formatting.

6.5 Key Differences Between Data Exploration and Presentation

Feature	Data Exploration	Data Presentation
Goal	Discovery, understanding	Communication, persuasion
Audience	Analyst, small team	Stakeholders, decision-makers, wider audience
Focus	Flexibility, experimentation	Clarity, accuracy, visual appeal
Visualizations	Quick, diverse, exploratory	Polished, focused, impactful
Narrative	Minimal, internal notes	Clear, compelling, tailored to audience
Level of Detail	High (all aspects of the data)	Selective (highlights key insights)
"Aha!" moments	New insights	Confirmation of insights
Polished Level	Quick, functional	Professional, polished

6.6 Choosing Appropriate Visualization Methods

The choice of visualization method depends on whether you are in the exploration or presentation phase.

6.6.1 Exploration Visualizations:

- **Histograms:** Quickly understand the distribution of a single variable.
- **Scatter Plots:** Explore relationships between two numerical variables.
- **Box Plots:** Compare the distribution of a variable across different groups.
- **Correlation Matrices:** Identify correlations between multiple numerical variables.
- **Density Plots:** See the shape of a distribution.
- **Parallel Coordinate Plots:** To visualize several different dimensions.
- **3-D visualizations:** Useful for getting a bird's-eye view of your data.

6.6.2 Presentation Visualizations:

- **Bar Charts:** Compare the magnitudes of different categories (ensure proper baseline and clear labels).
- **Line Charts:** Show trends over time or across ordered categories (ensure clear labels and scales).
- **Pie Charts:** (Use sparingly) Show proportions or percentages of different categories (limit the number of slices and ensure clear labels).
- **Maps:** Display geographical data and patterns (use appropriate projections and color schemes).
- **Scatter Plots (Annotated):** Highlight specific points or trends in a relationship between two variables.

6.6.3 Common rules of thumb for presenting datasets for people

- Always Label clearly
- Make sure axes are consistent
- Do not cut off axes
- Do not hide information

6.7 Creating Compelling Visuals for Data Presentation

For data presentation, it's crucial to create visuals that are clear, accurate, and engaging. Here are some tips:

- **Choose the Right Chart Type:** Select the chart type that best conveys your message and suits your data.
- **Simplify the Visual:** Remove unnecessary clutter, such as gridlines, extra labels, or overly complex designs.
- **Use Color Effectively:** Use color to highlight important elements or create visual contrast. Be mindful of colorblindness.
- **Tell a Story:** Use the visualization to tell a clear and compelling story. Add a title, caption, and annotations to guide the viewer.

6.7.0.1 Example: Improved Bar Chart for Presentation

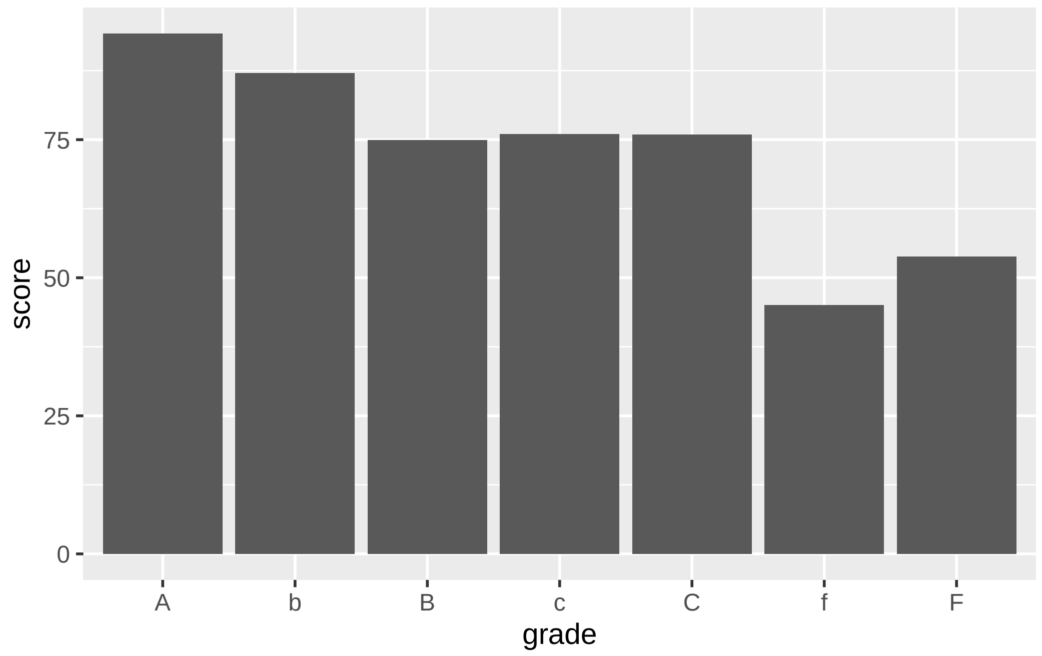
Let's say you want to present the average exam score for each grade. Here's how you might create a more polished version for presentation:

Initial Exploration Chart:

```
library(ggplot2)
```

```
Warning: package 'ggplot2' was built under R version 4.2.3
```

```
#Basic graph, for exploration
 exam_scores <- read.csv("https://raw.githubusercontent.com/sijuswamyresearch/R-for-Data-Anal
ggplot(exam_scores, aes(x = grade, y = score)) +
  geom_bar(stat = "summary", fun = "mean")
```

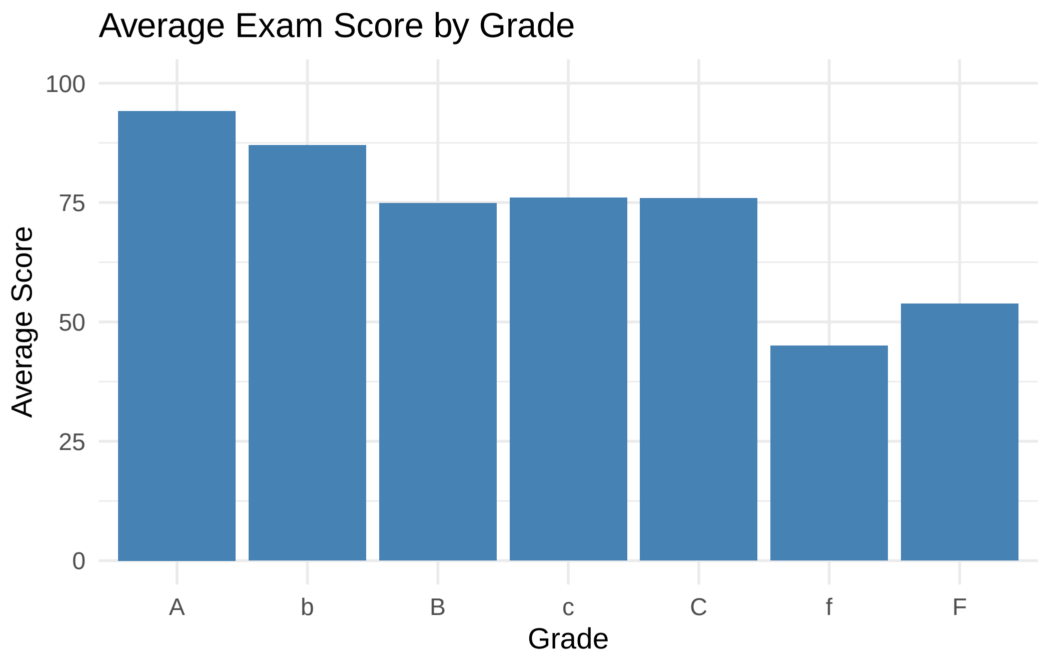


```r
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.2.3

Warning: package 'tibble' was built under R version 4.2.3

Warning: package 'tidyr' was built under R version 4.2.3

Warning: package 'readr' was built under R version 4.2.3

Warning: package 'purrr' was built under R version 4.2.3

Warning: package 'dplyr' was built under R version 4.2.3

Warning: package 'stringr' was built under R version 4.2.3

Warning: package 'forcats' was built under R version 4.2.3

Warning: package 'lubridate' was built under R version 4.2.3
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v lubridate 1.9.2     v tibble    3.2.1
v purrr     1.0.2     v tidyr     1.3.0
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
#Assuming the grade is the main column with the average score.
ggplot(exam_scores, aes(x = grade, y = score)) +
  geom_bar(stat = "summary", fun = "mean", fill = "steelblue") +
  labs(title = "Average Exam Score by Grade",
       x = "Grade",
       y = "Average Score") +
  theme_minimal() +
  ylim(0, 100) # Force score values between 0 and 100.
```



6.8 Dangers of Data Visualization

Data visualization can be very powerful for communicating the information with your stakeholders. But it can also give the audience an incorrect understanding of your data

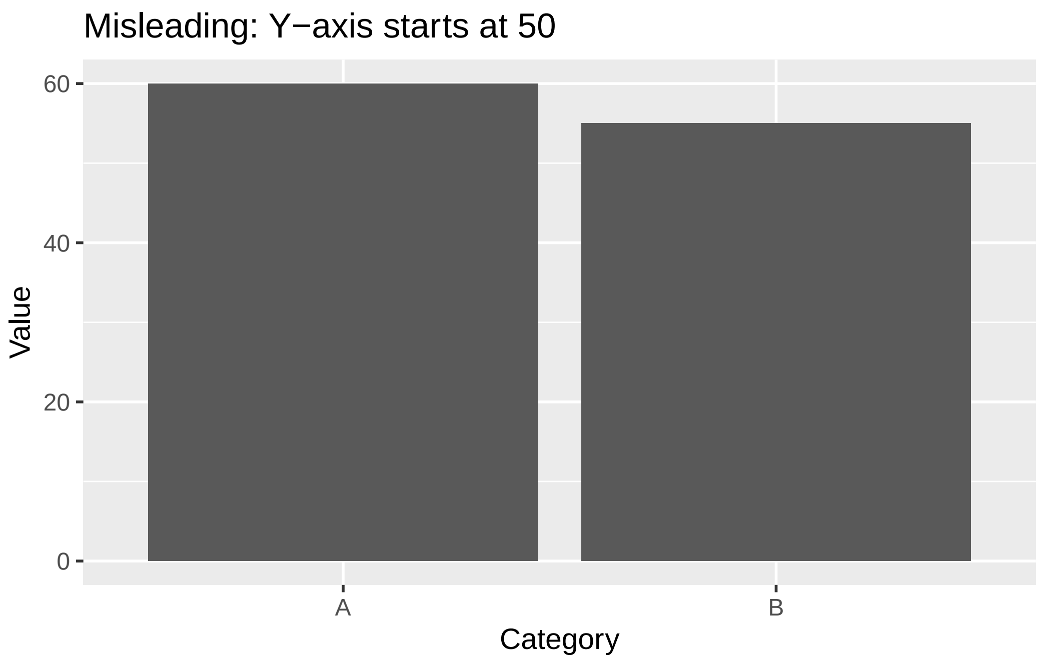
6.8.1 common pitfalls

- Incorrect data scale
- Hidden and misrepresented data
- Skewed charts

1. Data Scale

When communicating results make sure that the Y scale correctly represents the values for an accurate picture. Here is an example of an incorrect visualization of the data.

```r
library(tidyverse)
set.seed(123)
data <- data.frame(Category = c("A", "B"),
                   Value = c(60, 55))

ggplot(data, aes(x = Category, y = Value)) +
  geom_bar(stat = "identity") +
  ylim(0, 60) +
  labs(title = "Misleading: Y-axis starts at 50",
       x = "Category", y = "Value")
```



2. Hidden or misinterpreted data

One common visualziation mistake is to have incorrect scales and to remove data. If you had one set of data, and only communicated a small part of the data set the audience can get a very wrong idea about the quality and the overall trend.

This also applies to things such as selecting the data to use, or cleaning data that may be relevant.

3. Skewing Charts

Make sure you're choosing the best methods to display the data. Charts like a pie chart can often be more confusing than helpful.

6.9 Conclusion

Data exploration and data presentation are two distinct but complementary phases of the data analysis process. Data exploration is about discovering patterns and generating hypotheses. Data presentation is about communicating findings and insights to a specific audience. By understanding the key differences between these phases, you can tailor your analysis and communication to the appropriate context and audience. Choose the right visualization and format for the given situation.

Practice

- Take a dataset you've used previously.
- Create exploratory visualizations to understand the data.
- Select a key insight you want to communicate.
- Design a polished visualization to present that insight to a specific audience (e.g., a non-technical stakeholder).

7 Statistical Methods for Evaluation

7.1 Introduction

In this section, we will cover statistical methods used for evaluating data and validating hypotheses. Knowing the different types of statistical tests to perform will be critical for your own dataset.

7.2 Learning Objectives

- Explain the purpose of performing statistical tests.
- Identify what kind of data types to use for statistical tests.
- Create a new working git branch with statistical tests

7.3 What is a statistical test?

Statistical methods are used to interpret the data. It can include data cleaning, transformation and finding the right models or methods to test.

Some statistical tests are meant to understand the distributions of data. Other types of statistical tests are meant to compare one distribution against another, or to see if there is a relationships between the dataset.

7.3.1 Types of Data that work best with different tests

Different data formats have unique distributions, which are important to understand before performing any statistical tests. Tests like T-Tests work best with normal-style data, which has to be manually created. Also, there are tests that can convert data into a numerical result to run another test.

For this exercise we will discuss the common types of tests:

- **Normal test** Normal test involves checking a statistical assumption. One of the most common checks is the normality assumption.

- **T-test** Allows for 2 means to be compared. The most common type involves the use of numerical values to perform a comparison.
- **Pearson Test** Determines how strong a relationship is between 2 distributions
- **Chi-squared Test:** Use for evaluating the relationship between 2 categorical features.

7.3.2 Performing a Normal Statistical Test

In general, to create a normal set of distributions, you can use a built in function, for example `rnorm` to quickly create a normal distribution. Here's how to create the data and to run a "Normal test" to validate the data:

1. Create Dataset

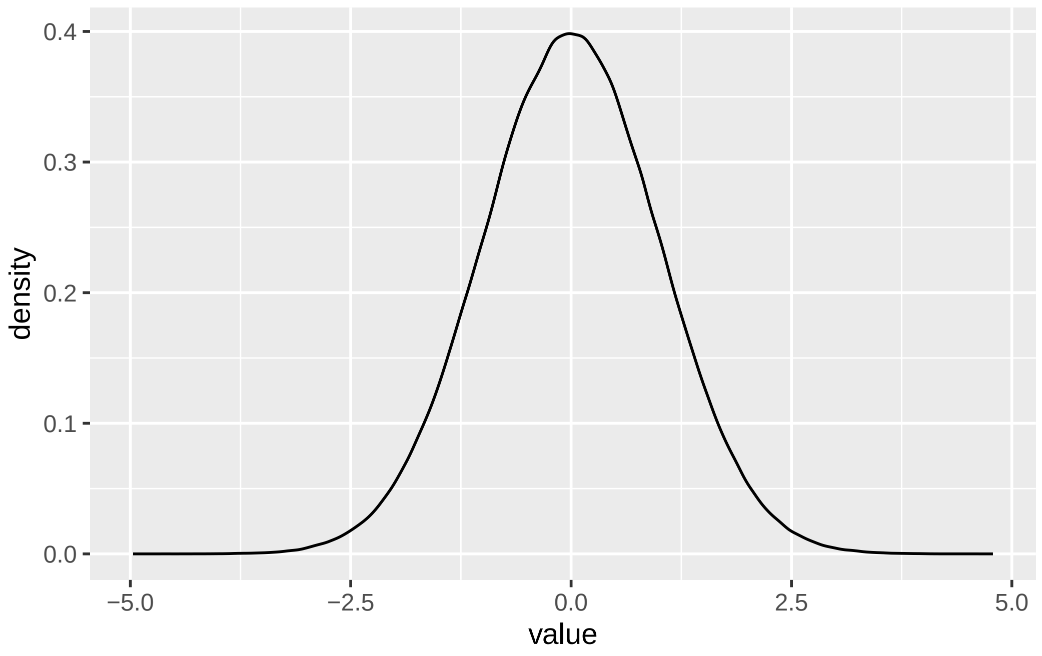
```
# create a normally distributed population set
population_norm <- data.frame(value = rnorm(n = 1000000, mean = 0, sd = 1))
summary(population_norm)
```

```
     value
 Min.   :-4.969771
 1st Qu.:-0.676367
 Median :-0.001311
 Mean   :-0.001212
 3rd Qu.: 0.673681
 Max.   : 4.785604
```

```
library(ggplot2)
```

```
Warning: package 'ggplot2' was built under R version 4.2.3
```

```
#Visualize distribution, from previous steps.
ggplot(population_norm, aes(value)) +
  geom_density()
```



1. Testing of normality of the data

```
shapiro.test(population_norm$value[1:5000]) #Run the test.
```

```

    Shapiro-Wilk normality test

data:  population_norm$value[1:5000]
W = 0.99959, p-value = 0.3834
```

If the p-value is less than the significance level (alpha) you reject the null hypothesis and it means there is a statistical significance. In this case, reject null. The shapiro test is testing if the data "normal". But in this case if the test passes it has to be rejected, since it will be testing if data is "non-normal", thus the need to invert the value.

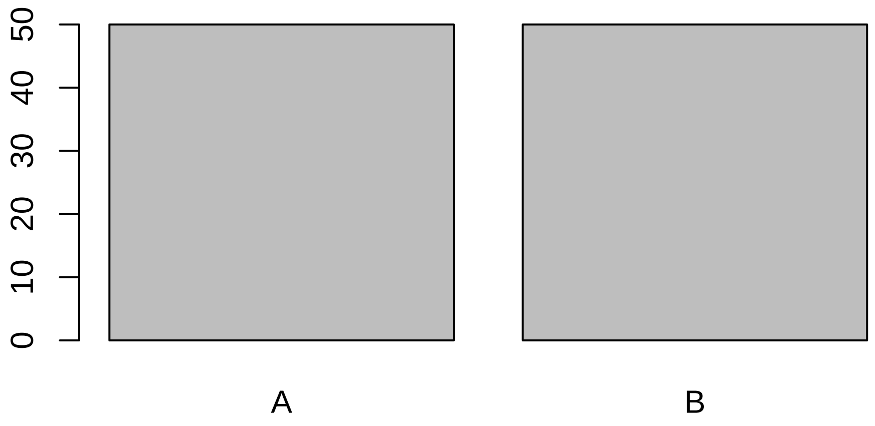
7.3.3 Performing a t- Test

A t-test can be very useful in checking if 2 samples have differences. Here's how you would run a t-Test.

```r
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.2.3

Warning: package 'tibble' was built under R version 4.2.3

Warning: package 'tidyr' was built under R version 4.2.3

Warning: package 'readr' was built under R version 4.2.3

Warning: package 'purrr' was built under R version 4.2.3

Warning: package 'dplyr' was built under R version 4.2.3

Warning: package 'stringr' was built under R version 4.2.3

Warning: package 'forcats' was built under R version 4.2.3

Warning: package 'lubridate' was built under R version 4.2.3

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v lubridate 1.9.2     v tibble    3.2.1
v purrr     1.0.2     v tidyr     1.3.0
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```r
# Create two different populations
population1 <- data.frame(value = rnorm(n = 50, mean = 0, sd = 1), group = "A")
population2 <- data.frame(value = rnorm(n = 50, mean = .5, sd = 1), group = "B")

# Combine populations
population_combined <- rbind(population1, population2)
tble=table(population_combined$group)
barplot(tble)
```



```r
library(dplyr)

# mean of samples
# Calculate mean of samples over group
mean_by_group <- population_combined %>%
  group_by(group) %>%
  summarize(mean_value = mean(value))

print(mean_by_group)
```

```
# A tibble: 2 x 2
  group mean_value
  <chr>      <dbl>
1 A         0.0451
2 B         0.484
```

Perform t Test. Note, that for this particular function, you will need at least 2 variable.

```r
t.test(data = population_combined, value ~ group)
```

```
	Welch Two Sample t-test

data:  value by group
t = -2.2338, df = 97.381, p-value = 0.02779
alternative hypothesis: true difference in means between group A and group B is not equal to
95 percent confidence interval:
 -0.82897846 -0.04896017
sample estimates:
mean in group A mean in group B 
     0.04509902      0.48406833 
```

From here the t-test is the main value to look for, which validates whether the averages are different. The important detail here is the p-value. As mentioned earlier, if the P-Value is less than the alpha, which is typically 0.05, it means that we should reject that distribution.

7.3.4 Performing a correlation Test

Correlation test will help find any relationships within the data. The most common type of test is the Pearson Test.

```r
#Create data with relationships
library(tidyverse)
set.seed(123)
x = rnorm(100)
data <- data.frame(
  x,
  y = 2*x + rnorm(100)
)
#data
cor(data)
```

```
          x         y
x 1.0000000 0.8786993
y 0.8786993 1.0000000
```

Correlation between the two variables can be tested using the pearson corelation test.

```r
cor.test(data$x, data$y, method = "pearson")
```

```
	Pearson's product-moment correlation

data:  data$x and data$y
t = 18.222, df = 98, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8246011 0.9168722
sample estimates:
      cor 
0.8786993 
```

Looking at the information, the details for Pearson tests show a "correlation" value.

7.3.5 Performing a chi-squared Test

The Chi-squared test is used to determine if there is a statistically significant association between two categorical variables. It's used when you have data organized in a contingency table (a cross-tabulation of two categorical variables).

Hypotheses:

- **Null Hypothesis (H0):** There is no association between the two categorical variables.
- **Alternative Hypothesis (H1):** There is an association between the two categorical variables.

Here's how to perform a Chi-squared test in R:

1. **Create a Contingency Table:** Create a cross-tabulation of the two categorical variables.

```r
library(tidyverse)

# Create a sample dataset
data <- data.frame(
  gender = factor(rep(c("Male", "Female"), times = c(40, 70))),
  smoker = factor(rep(c("Yes", "No"), times = c(60, 50)))
)

# Create a contingency table
contingency_table <- table(data$gender, data$smoker)
print(contingency_table)
```

```
        No Yes
  Female 50  20
  Male    0  40
```

Performing chi-squared test.

```
# Perform the Chi-squared test
chi_squared_test <- chisq.test(contingency_table)
print(chi_squared_test)
```

```

	Pearson's Chi-squared test with Yates' continuity correction

data:  contingency_table
X-squared = 49.54, df = 1, p-value = 1.944e-12
```

Interpreting the Results:

The output of `chisq.test()` will provide the following information:

- **Chi-squared Statistic:** A measure of the difference between the observed and expected frequencies.
- **Degrees of Freedom:** Reflects the number of independent pieces of information used in the test.
- **P-value:** The probability of obtaining the observed results (or more extreme results) if the null hypothesis is true.
  - A small P-value indicates that the null hypothesis is unlikely to be true.
- **Decision:** If the p-value is less than or equal to the significance level (alpha), you reject the null hypothesis.
  - If `p-value <= alpha`, reject the null hypothesis.
  - If `p-value > alpha`, fail to reject the null hypothesis.

8 Perception of Engineering Students on Fruit Preference

Abstract: This article is a short analysis of the data collected from the course participants of the `Fundamentals of Data Analytics using R Programming`. A baseline descriptive analysis is conducted and the results are tested using hypothesis testing for generalizations.

Key words: Baseline analysis, categorical data, Likert scaled items, correlation testing, regression models, box plot, bar plot, percentage analysis, χ^2-test, ANOVA.

```r
# loading the .csv file containing data

df=read.csv("https://raw.githubusercontent.com/sijuswamyresearch/R-for-Data-Analytics/refs/he
df=df[,-1]
size=nrow(df)
#df
```

8.1 Introduction

This article is prepared during the hands-on sessions of the first phase of the Faculty Development Programme- `Data Analytics Using R`. A questionnaire is prepared and administered through *Google form*. Total number of samples collected in the study is 316.

8.2 Data cleaning and Wrangling

As the first stage of the data pre-processing, the row data is cleaned and wrangled to make it ready for statistical analysis. Main processes involved are:

1. Removing unnecessary columns
2. Rename the attributes for make then clear and precise
3. Mapping of data types for statistical analysis
4. Re-frame the structure of the data if required

8.2.1 Rename the attribute names

Since the column titles obtained through the *Google forms* are the questions given in the questionnaire, it will be not suitable to represent an attribute. There are two ways to correct it- manually correct in the downloaded excel file or rename the column names programatically.

In this article, the later approach is demonstrated.

```
colnames(df) <- c("Gender","Age","Weight","Height","Orange","Grapes","Banana","Apple","Mango
```

8.3 Descriptive Analysis

A cleaned data is examined with basic statistical tools to understand the distribution of various attributes and its relationship between control variables. At this initial stage, fundamental tools like frequency tables, cross tabulations and percentage analysis followed by proper visualizations will be used. All the socio-demographic variables and control variables will be analysed statistically.

Gender

The gender-wise distribution of the responses is shown in the following Table.

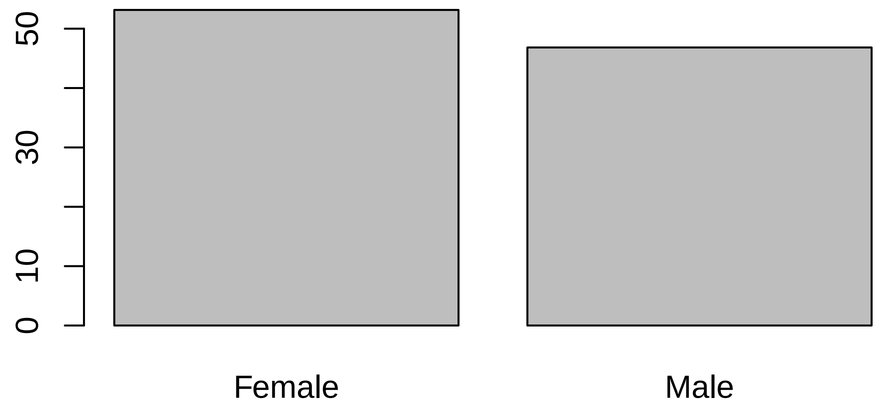
```
Gtab=round(prop.table(table(df$Gender))*100,2)
Gtab
```

```
Female   Male 
 53.16  46.84 
```

The percentage analysis shows that majority of the respondents are from Female category.

A barplot showing this distribution is shown in the following Figure.

```
barplot(Gtab)
```

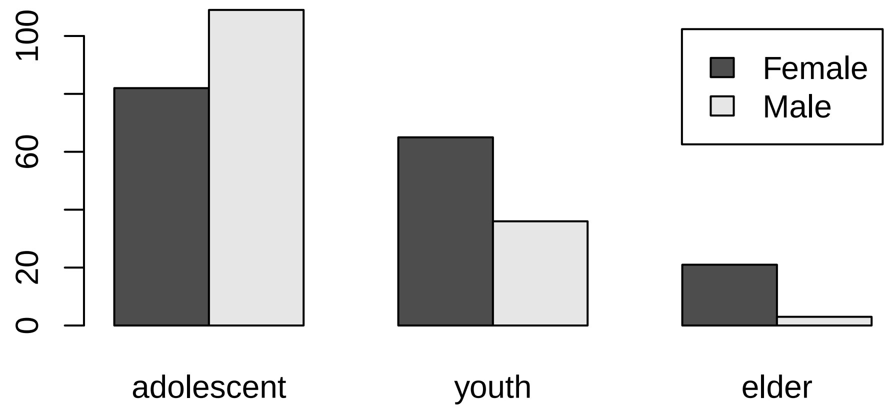


Age

Completed age of the respondents are collected through the form. Interest of respondents may be varied over age range rather than individual ages. So the continuous variable `age` shall be converted to a categorical variable for reasonable use of this attribute. Respondents with age upto 20 is consider as 'adolescent', between 20 and 30 as 'youth' and above 30 is considered as the 'elder'. A new variable `Age_group` is created as follows.

```
df$Age_group <- cut(df$Age,
                     breaks=c(-Inf, 20, 30, Inf),
                     labels=c("adolescent","youth","elder"))
#df
```

```
df$Gender=as.factor(df$Gender)
barplot(table(df$Gender,df$Age_group),beside = TRUE,legend.text = levels(df$Gender))
```



A contingency table of the gender-wise distribution over age-group is shown in the Table below:

```
knitr::kable(table(df$Age_group,df$Gender))
```

	Female	Male
adolescent	82	109
youth	65	36
elder	21	3

8.3.1 Conversion of data into long format

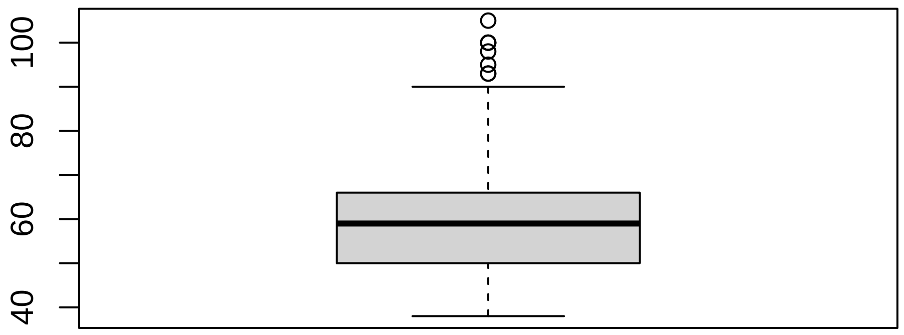
The basic objective of this work is to compare the respondent preference of fruit based on their interest shown in the survey. So responses regarding preference of various fruits under the study should br bring together before statistical investigations. To achieve this goal, the *wide formatted** data is transformed into the long format with a columns for fruits and the corresponding preference level.

```
library(reshape2)
library(plyr)
data_long <- melt(df,
        # ID variables - all the variables to keep but not split apart on
    id.vars=c("Gender","Age","Weight","Age_group","Height"),
        # The source columns
    measure.vars=c("Orange", "Grapes", "Banana","Apple","Mango","Cherry" ),
        # Name of the destination column that will identify the original
        # column that the measurement came from
    variable.name="Fruit",
    value.name="Rating"
)
data_long <- arrange(data_long,data_long$Fruit, data_long$Rating)# to get a sorted view
#data_long
```

Weight

The distribution of respondent's weights is shown in the `Boxplot` shown below:

```
boxplot(df$Weight,notch = F)
```



The five point summary of the attribite 'weight' is shown in the Table below:

```
summary(df$Weight)
```

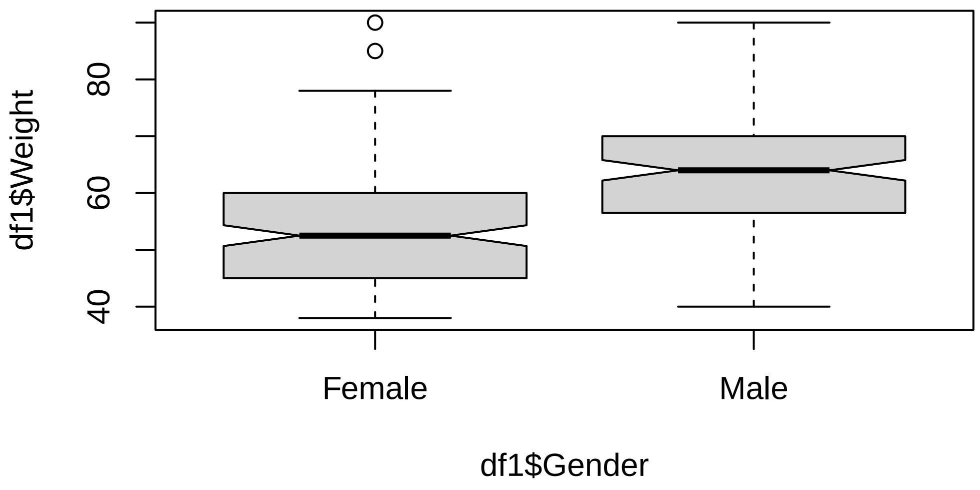
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  38.00   50.00   59.00   59.59   66.00  105.00 
```

It is noted that the box plot shows the presence of outliers in the weights. These samples can be removed as follows:

```
outliers <- boxplot(df$Weight, plot=FALSE)$out
df1<-df;
df1<- df1[-which(df1$Weight %in% outliers),]
#dim(df1)
```

A gender-wise comparison of weights is shown in the following figure. It is clear from the plot that, the majority of the female category is above the average body weight category!

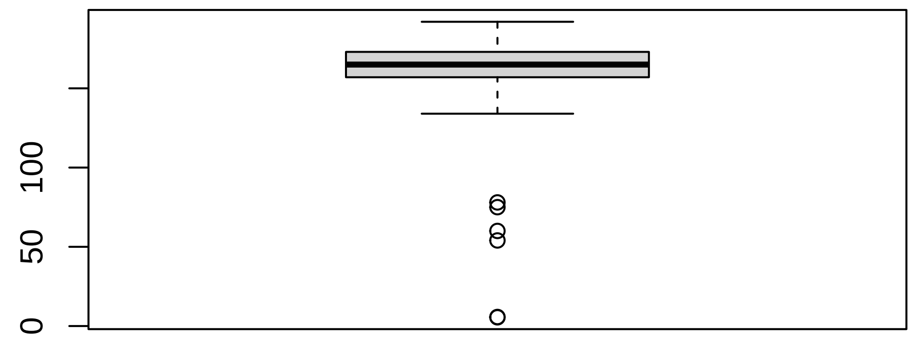
```
boxplot(df1$Weight~df1$Gender,notch = T)
```



Height

The distribution of respondent's height is shown in the `Boxplot` shown below:

```
boxplot(df$Height,notch = F)
```



The five point summary of the attribute 'Height' is shown in the Table below:

```r
summary(df$Height)
```

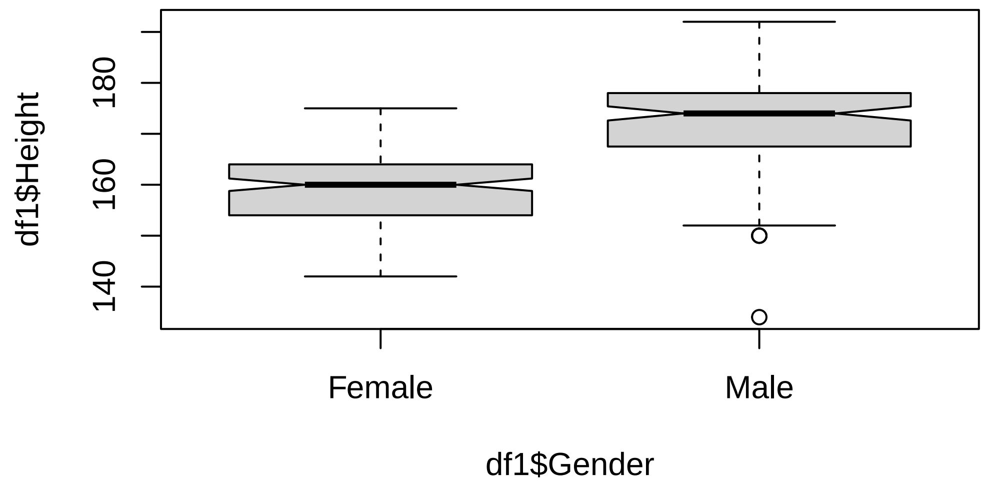
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    5.5   157.0   165.0   162.9   173.0   192.0 
```

It is noted that the box plot shows the presence of outliers in the weights. These samples can be removed as follows:

```r
outliers <- boxplot(df$Height, plot=FALSE)$out
df1<-df;
df1<- df1[-which(df1$Height %in% outliers),]
#dim(df1)
```

A gender-wise comparison of heights is shown in the following figure. It is clear from the plot that, the majority of the female category is below the average height category!

```r
boxplot(df1$Height~df1$Gender,notch = T)
```



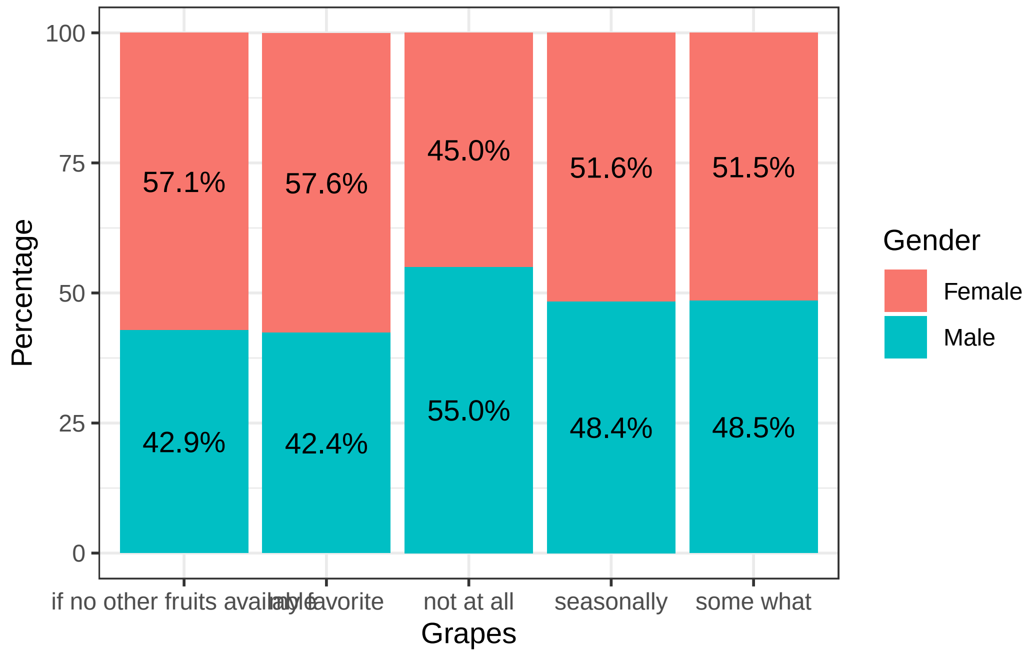
8.4 Gender-wise preference of fruits

A percentage analysis of gender-wise preference of various fruits is given in theis section.

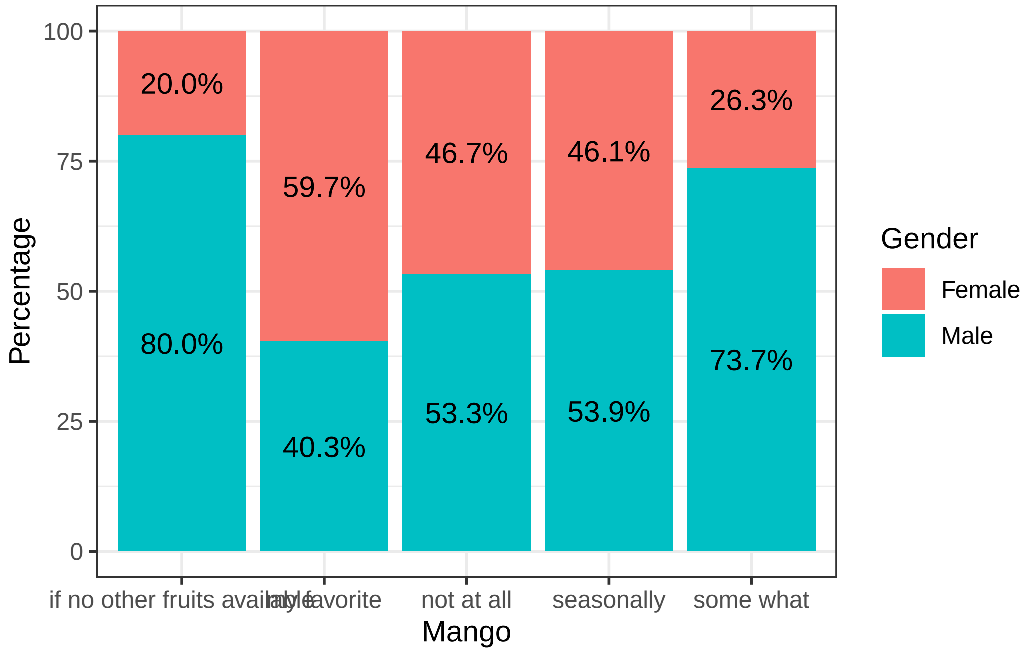
```r
library(dplyr)
```

```

Attaching package: 'dplyr'

The following objects are masked from 'package:plyr':

    arrange, count, desc, failwith, id, mutate, rename, summarise,
    summarize

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

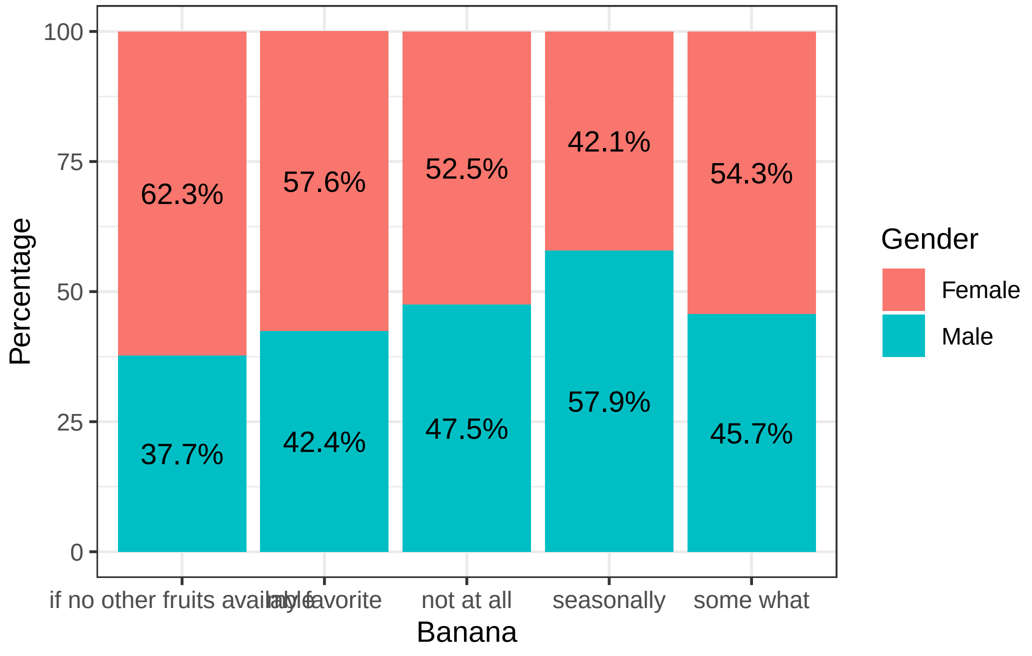
```
library(ggplot2)

# Ensure 'Grapes' is treated as a factor
df$Grapes <- as.factor(df$Grapes)
df$Gender <- as.factor(df$Gender)

# Check if the column 'Grapes' exists
if ("Grapes" %in% colnames(df)) {
  df %>%
    dplyr::count(Grapes, Gender) %>%
    group_by(Grapes) %>%
    mutate(pct = prop.table(n) * 100) %>%
    ggplot(aes(Grapes, pct, fill = Gender)) +
    geom_bar(stat = "identity") +
    ylab("Number of respondents") +
    geom_text(aes(label = paste0(sprintf("%1.1f", pct), "%")),
              position = position_stack(vjust = 0.5)) +
    labs(x = "Grapes", y = "Percentage", fill = "Gender") +
    theme_bw()
} else {
  print("Column 'Grapes' not found in the data frame.")
}
```



```
#gender-wise
library(dplyr)
library(ggplot2)

df %>%
  dplyr::count(Mango,Gender) %>%
  group_by(Mango) %>%
  mutate(pct= prop.table(n) * 100) %>%
  ggplot() + aes(Mango, pct, fill=Gender) +
  geom_bar(stat="identity") +
  ylab("Number of respondents") +
  geom_text(aes(label=paste0(sprintf("%1.1f", pct),"%")),
            position=position_stack(vjust=0.5)) +labs(x ="Mango", y = "Percentage",fill="Genc
  theme_bw()
```



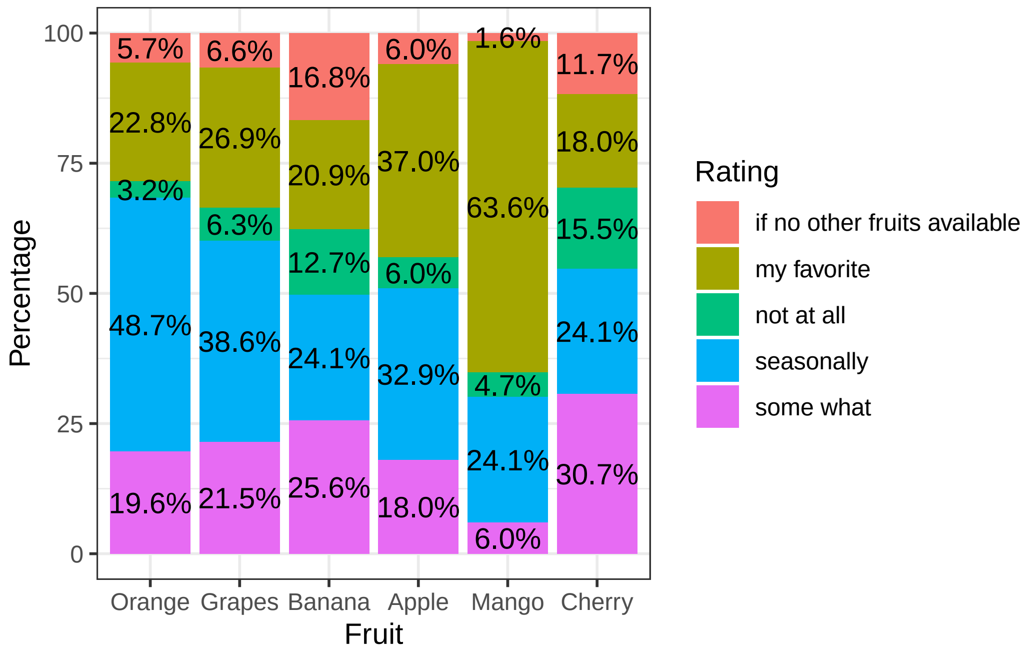
```
#gender-wise
library(dplyr)
library(ggplot2)

df %>%
  dplyr::count(Banana,Gender) %>%
  group_by(Banana) %>%
  mutate(pct= prop.table(n) * 100) %>%
  ggplot() + aes(Banana, pct, fill=Gender) +
  geom_bar(stat="identity") +
  ylab("Number of respondents") +
  geom_text(aes(label=paste0(sprintf("%1.1f", pct),"%")),
            position=position_stack(vjust=0.5)) +labs(x ="Banana", y = "Percentage",fill="Ge
  theme_bw()
```



8.5 Overall preference over fruits

A percentage analysis of overall preference over various fruits is shown in the following figure.

```
#gender-wise
library(dplyr)
library(ggplot2)

data_long%>%
  dplyr::count(Fruit,Rating) %>%
  group_by(Fruit) %>%
  mutate(pct= prop.table(n) * 100) %>%
  ggplot() + aes(Fruit, pct, fill=Rating) +
  geom_bar(stat="identity") +
  ylab("Number of respondents") +
  geom_text(aes(label=paste0(sprintf("%1.1f", pct),"%")),
            position=position_stack(vjust=0.5)) +labs(x ="Fruit", y = "Percentage",fill="Rat
  theme_bw()
```



8.6 Finding descriptive statistics of continuous variables

Weight over Gender

```
library(dplyr)
Fa=group_by(df, Gender) %>%
  dplyr::summarise(
    count = n(),
    mean =round( mean(Weight, na.rm = TRUE),2),
    sd = round(sd(Weight, na.rm = TRUE),2)
  )
knitr::kable(Fa)
```

Gender	count	mean	sd
Female	168	53.89	9.96
Male	148	66.06	12.78

Height over Gender

```r
library(dplyr)
Fa=group_by(df, Gender) %>%
  dplyr::summarise(
    count = n(),
    mean =round( mean(Height, na.rm = TRUE),2),
    sd = round(sd(Height, na.rm = TRUE),2)
  )
knitr::kable(Fa)
```

Gender	count	mean	sd
Female	168	158.57	10.40
Male	148	167.85	25.34

Height over Age Group

```r
library(dplyr)
Fa=group_by(df, Age_group) %>%
  dplyr::summarise(
    count = n(),
    mean =round( mean(Height, na.rm = TRUE),2),
    sd = round(sd(Height, na.rm = TRUE),2)
  )
knitr::kable(Fa)
```

Age_group	count	mean	sd
adolescent	191	163.33	20.97
youth	101	162.55	18.38
elder	24	161.21	8.73

```r
library(dplyr)
Fa=group_by(df, Gender) %>%
  dplyr::summarise(
    count = n(),
    mean =round( mean(Age, na.rm = TRUE),2),
    sd = round(sd(Age, na.rm = TRUE),2)
  )
knitr::kable(Fa)
```

Gender	count	mean	sd
Female	168	22.66	5.84
Male	148	20.30	2.13

8.6.1 Rename the levels (For Numerical calculation)

```r
df$Orange <- factor(df$Orange,
                    levels = c("not at all","if no other fruits available","some what","seasona
                    labels = c("1","2","3","4","5"))
#head(df)
```

```r
mean(as.integer(df$Orange))
```

```
[1] 3.822785
```

8.7 Inferential Analysis

The inferential analysis is the generalization part of statistics. This phase focuses on possibilities of generalization of observations in the descriptive analysis. This is achieved through the `hypothesis testing` aspects of inferential statistics.

8.7.1 Testing of significance of difference in mean weight over gender

Significance of difference in mean of continuous variable over two categories can be tested using the `t-test`. The null hypothesis of the t-test is;

H_0 : there is no significance difference in the mean

The alternative hypothesis is:

$$H_1 : \text{there is significance difference in the mean}$$

If the `p-value` of the test result is less than 0.05, the null hypothesis is rejected at 5% level of significance. Otherwise the null hypothesis can't be rejected.

As an example, let us investigate whether there is significant difference in the mean weight over gender.

```
t.test(Weight~Gender,alternative="less",data=df)
```

```

	Welch Two Sample t-test

data:  Weight by Gender
t = -9.3516, df = 276.65, p-value < 2.2e-16
alternative hypothesis: true difference in means between group Female and group Male is less
95 percent confidence interval:
      -Inf -10.02187
sample estimates:
mean in group Female   mean in group Male 
            53.88988             66.05946 
```

```
#plot(df$Weight~df$Gender)
```

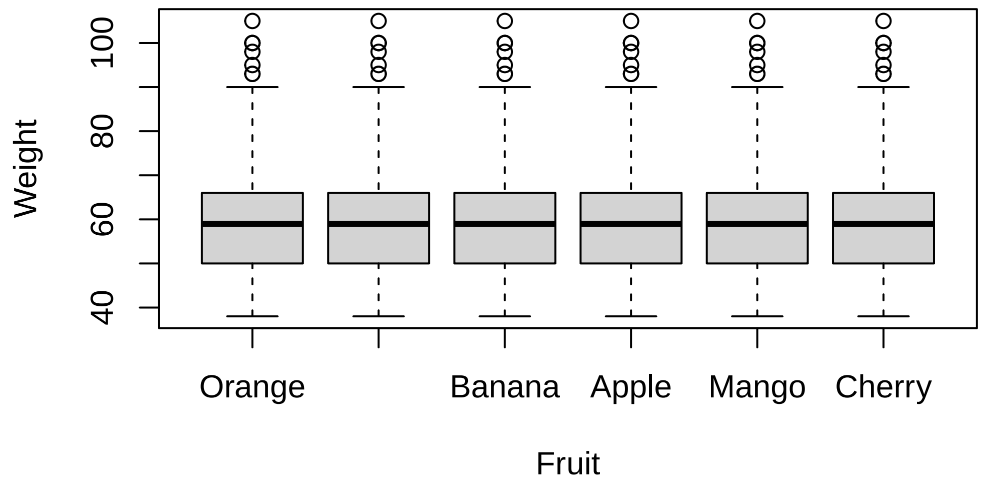
Since the p-value is less than 0.05, the null hypothesis is rejected. So it is statistically reasonable to conclude that the mean weight of female respondents is less than male respondents.

8.7.2 Significance of difference in mean weight over Fruit interest

```
# Compute the analysis of variance
res.aov <- aov(Weight ~ Fruit, data = data_long)
# Summary of the analysis
summary(res.aov)
```

```
              Df Sum Sq Mean Sq F value Pr(>F)
Fruit          5      0     0.0       0      1
Residuals   1890 313287   165.8               
```

```r
plot(Weight~Fruit,data=data_long)
```



8.8 Testing of significance difference in rating over Fruit category

As a first step a percentage analysis is conducted on the fruit preference data as shown below:

8.8.1 Percentage analysis of the rating of fruit category

```r
prop.table(table(data_long$Fruit, data_long$Rating))*100
```

```
         if no other fruits available my favorite not at all seasonally
  Orange                    0.9493671   3.7974684  0.5274262  8.1223629
  Grapes                    1.1075949   4.4831224  1.0548523  6.4345992
  Banana                    2.7953586   3.4810127  2.1097046  4.0084388
  Apple                     1.0021097   6.1708861  1.0021097  5.4852321
  Mango                     0.2637131  10.6012658  0.7911392  4.0084388
  Cherry                    1.9514768   3.0063291  2.5843882  4.0084388
```

```
        some what
Orange  3.2700422
Grapes  3.5864979
Banana  4.2721519
Apple   3.0063291
Mango   1.0021097
Cherry  5.1160338
```

8.8.2 χ^2 test for confirmation of difference in rating over fruit category

```r
# chi square
chisq.test(table(data_long$Fruit,data_long$Rating))
```

```

    Pearson's Chi-squared test

data:  table(data_long$Fruit, data_long$Rating)
X-squared = 352.9, df = 20, p-value < 2.2e-16
```

Since the p-value is less than 0.05, the null hypothesis that there is no significant difference in rating over fruit category is rejected. So it is statistically reasonable to conclude that the respondent's preference over fruits is statistically significant.

8.8.3 Fruit preference over Age group

Significance of difference in fruit rating over age group is tested using the chi-squared test.

```r
# chi square
chisq.test(table(data_long$Fruit,data_long$Age_group))
```

```

    Pearson's Chi-squared test

data:  table(data_long$Fruit, data_long$Age_group)
X-squared = 0, df = 10, p-value = 1
```

Since the p-value is greater than 0.05, the null hypothesis that there is no significant difference in the fruit rating over age group.

8.9 Conclusion

Based on the statistical analysis the following findings are elicited.

1. There is significant difference in the mean weight of the respondents
2. There is significant difference in the mean weight over fruit preference.
3. There is significant difference in the fruit ratings over fruit type.
4. There is no significant difference in the fruit preference over age group.

Kabacoff, Robert. 2022. *R in Action: Data Analysis and Graphics with r and Tidyverse.* Simon; Schuster.

Mayor, Eric. 2015. *Learning Predictive Analytics with r.* Packt Publishing Ltd.