**Program 1 : To implement Data Acquisition R Script**

ages <- c(21, 25, 30, 28, 35) #Entering the Data Manually names <- c("John", "Anna", "Mike", "Sara", "Paul")

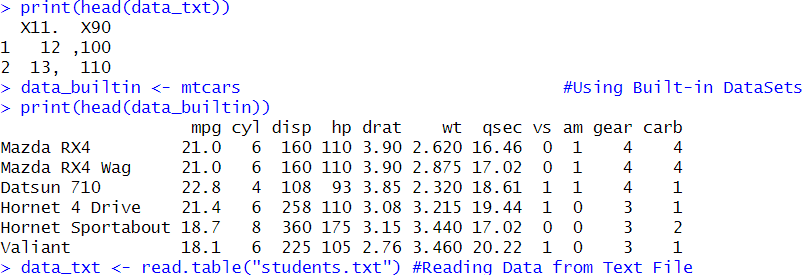
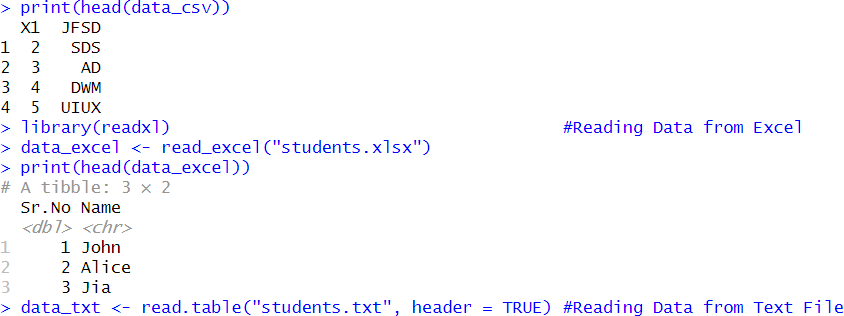
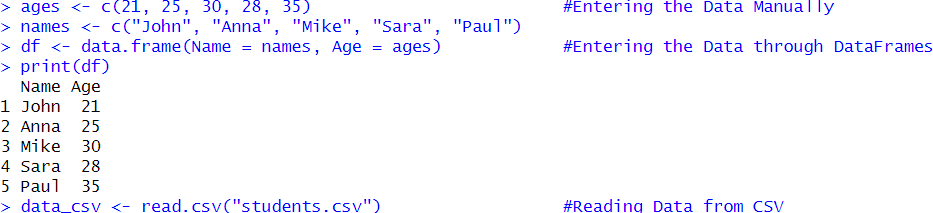
df <- data.frame(Name = names, Age = ages) #Entering the Data through DataFrames print(df)

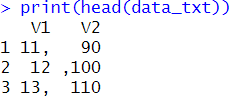
data\_csv <- read.csv("students.csv") #Reading Data from CSV print(head(data\_csv))

library(readxl) #Reading Data from Excel data\_excel <- read\_excel("students.xlsx") print(head(data\_excel))

data\_txt <- read.table("students.txt", header = TRUE) #Reading Data from Text File print(head(data\_txt))

data\_builtin <- mtcars #Using Built-in DataSets print(head(data\_builtin))

**Output**



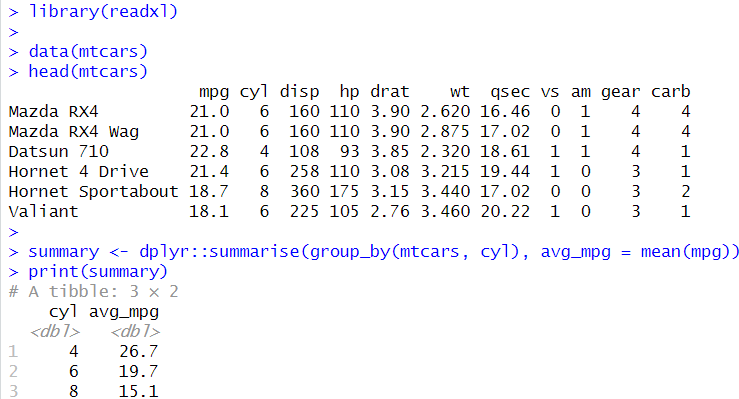
**Program 2 : To Install and Load the Packages R Script**

install.packages("ggplot2") # For data visualization install.packages("dplyr") # For data manipulation install.packages("readxl") # For reading Excel files library(ggplot2)

library(dplyr) library(readxl) data(mtcars) head(mtcars)

summary <- dplyr::summarise(group\_by(mtcars, cyl), avg\_mpg = mean(mpg)) print(summary)

**Output**



**Program 3 : To implement a Data-Types and Check the type of Variables R Script**

x <- 10.5

class(x)

is.numeric(x) y <- 25L

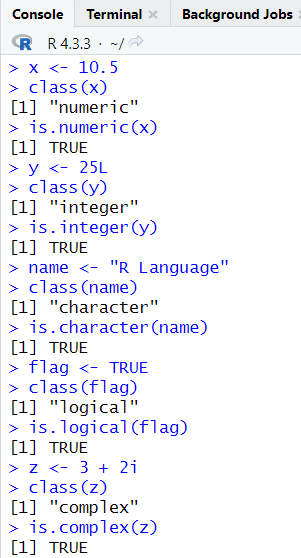
class(y) is.integer(y)

name <- "R Language" class(name) is.character(name) flag <- TRUE class(flag) is.logical(flag)

z <- 3 + 2i

class(z) is.complex(z)

**Output**



**Program 4 : Implementation of Vectors R Script**

flowers <- c("lily","daisy","tulip","rose")

flowers #Print a vector

flowers[2] #Access the item

flowers[c(1,3)] #Access the items

flowers[c(-1)] #Access all items except for the first item

length(flowers) #Length of the vector

sort(flowers) #Sort items Alphabetically flowers[1] <- "dadlieons" #Change the item of index 1 numbers <-c(1,5,3) #Print a numeric vector numbers

sort(numbers) #Sort numbers

numberic <-1:10 #Print numbers from 1-10

numeric

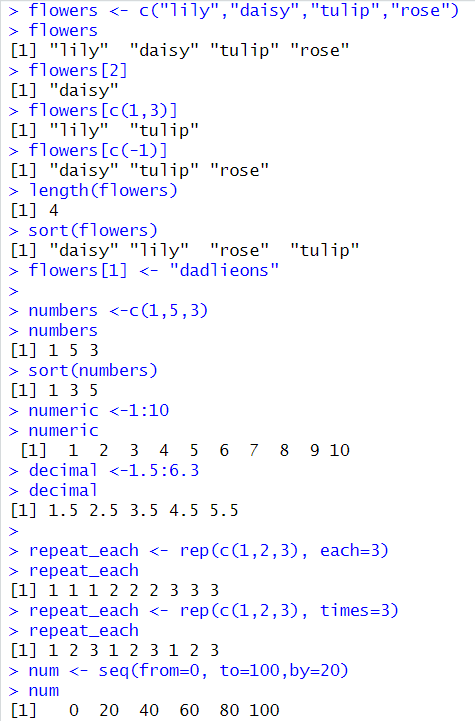
decimal <-1.5:6:3 #Print decimals decimal

repeat\_each <- rep(c(1,2,3), each=3) #Print numbers 3 times repeat\_each

repeat\_each <- rep(c(1,2,3), times=3) #Print all the numbers 3 times repeat\_each

num <- seq(from=0, to=100,by=20) #Print numbers from 20-100 num

**Output**



**Program 5 : Implementation of Matrix R Script**

thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2) thismatrix #Print Matrix

thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2) thismatrix

thismatrix[1, 2] #Access Matrix Items thismatrix[2,]

thismatrix[,2]

thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "pear", "melon", "fig"), nrow = 3, ncol = 3)

thismatrix[c(1,2),] #Access more than one row

thismatrix[, c(1,2)] #Access more than one column

newmatrix <- cbind(thismatrix, c("strawberry", "blueberry", "raspberry")) #Combines column newmatrix

newmatrix <- rbind(thismatrix, c("strawberry", "blueberry", "raspberry")) #Combines rows newmatrix

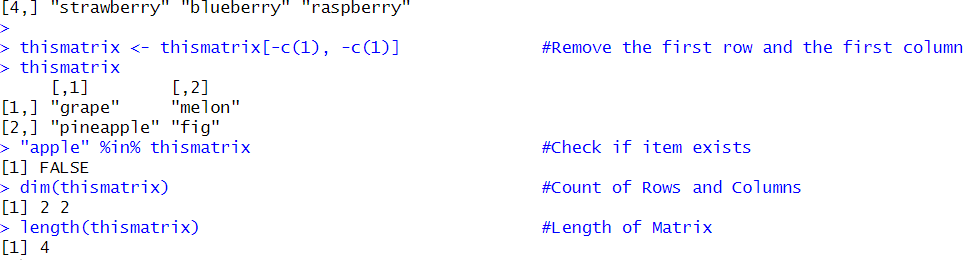
thismatrix <- thismatrix[-c(1), -c(1)] #Remove the first row and the first column thismatrix

"apple" %in% thismatrix #Check if item exists

dim(thismatrix) #Count of Rows and Columns

length(thismatrix) #Length of Matrix

**Output**



**Program 6 : Implementation of List R Script**

fruits <- list("apple", "banana", "cherry","jackfruit","pineapple")

fruits #Print list

fruits[1] #Access list

fruits[1] <- "blackcurrant" #Change item value

length(fruits) #Length of the list "apple" %in% fruits #Check if items exists

append(fruits, "graphes") #Add new item in a list

append(fruits, "graphes", after = 2) #Add new item after=index number new <- fruits[-1] #Remove item

fruits[2:5] #Access items Range of Indexes

for (x in fruits) { #Access through Loop

print(x)

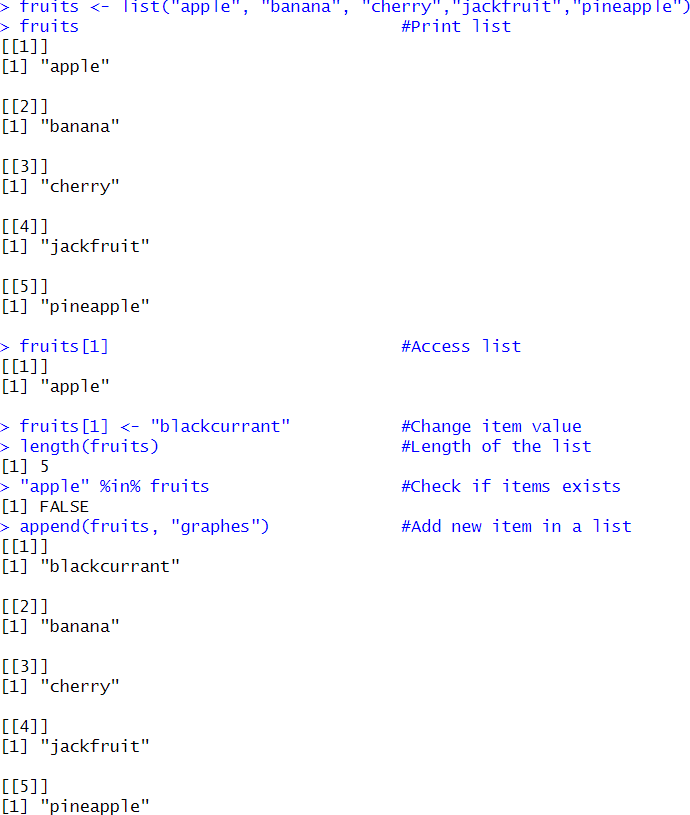
}

list1 <- list("a", "b", "c") #Join Two List list2 <- list(1,2,3)

list3 <- c(list1,list2)

List3

**Output:**





**Program 7 : Implementation of Factor R Script**

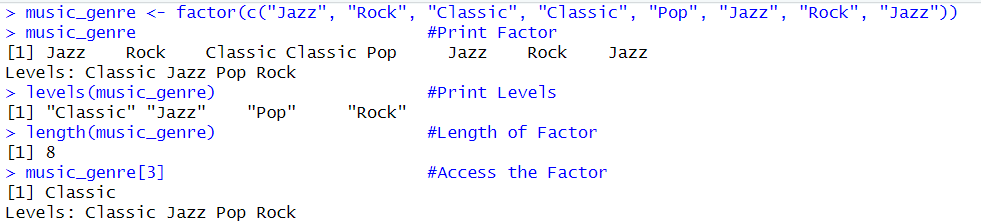
music\_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz")) music\_genre #Print Factor

levels(music\_genre) #Print Levels

length(music\_genre) #Length of Factor

music\_genre[3] #Access the Factor

**Output**



**Program 8 : Implementation of Data Frame**

**R Script**

Data\_Frame <- data.frame (

Training = c("Strength", "Stamina", "Other"), Pulse = c(100, 150, 120),

Duration = c(60, 30, 45))

Data\_Frame #Print DataFrame

summary(Data\_Frame) #DataFrame

Summary Data\_Frame[1] #Access DataFrame through Column Data\_Frame[["Training"]]

Data\_Frame$Training

New\_row\_DF <- rbind(Data\_Frame, c("Strength", 110, 110)) #Add new row New\_row\_DF

New\_col\_DF <- cbind(Data\_Frame, Steps = c(1000, 6000, 2000)) #Add new column

New\_col\_DF

Data\_Frame\_New <- Data\_Frame[-c(1), -c(1)] #Remove first column and row Data\_Frame\_New

dim(Data\_Frame) #Amount of rows and columns

ncol(Data\_Frame) #Count of column

nrow(Data\_Frame) #Count of row

length(Data\_Frame) #Length of DataFrame

Data\_Frame1 <- data.frame (

Training = c("Strength", "Stamina", "Other"), Pulse = c(100, 150, 120),

Duration = c(60, 30, 45)) Data\_Frame2 <- data.frame (

Training = c("Stamina", "Stamina", "Strength"), Pulse = c(140, 150, 160),

Duration = c(30, 30, 20))

New\_Data\_Frame <- rbind(Data\_Frame1, Data\_Frame2) #Combines rows New\_Data\_Frame

Data\_Frame3 <- data.frame (

Training = c("Strength", "Stamina", "Other"), Pulse = c(100, 150, 120),

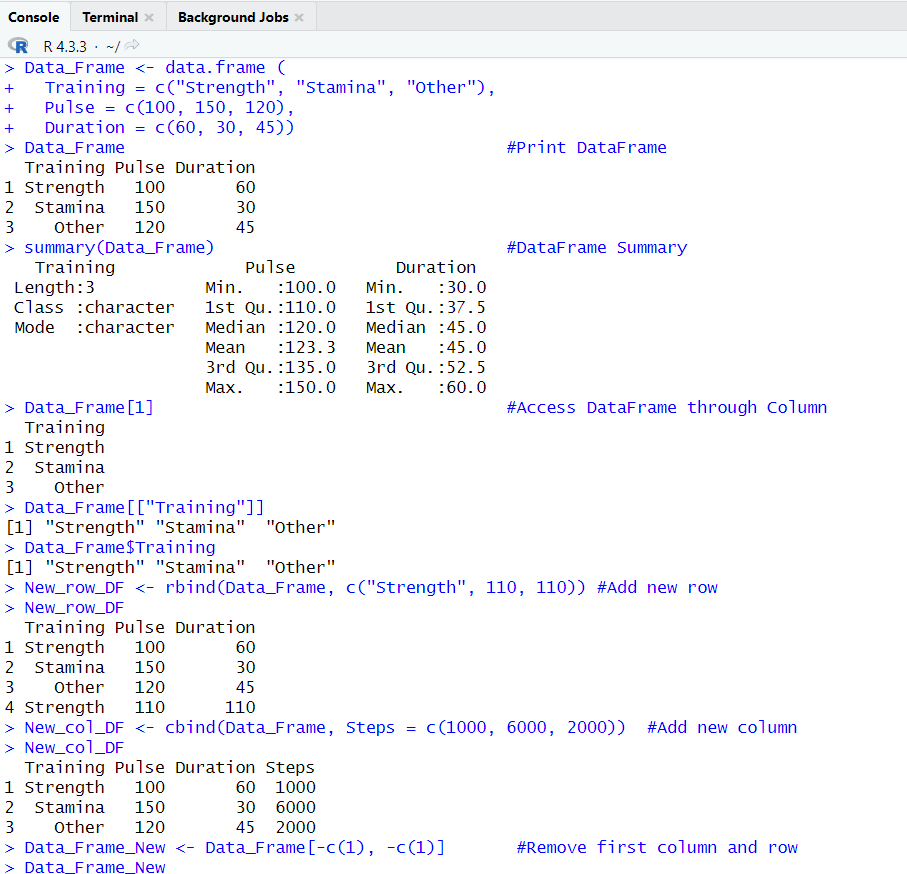
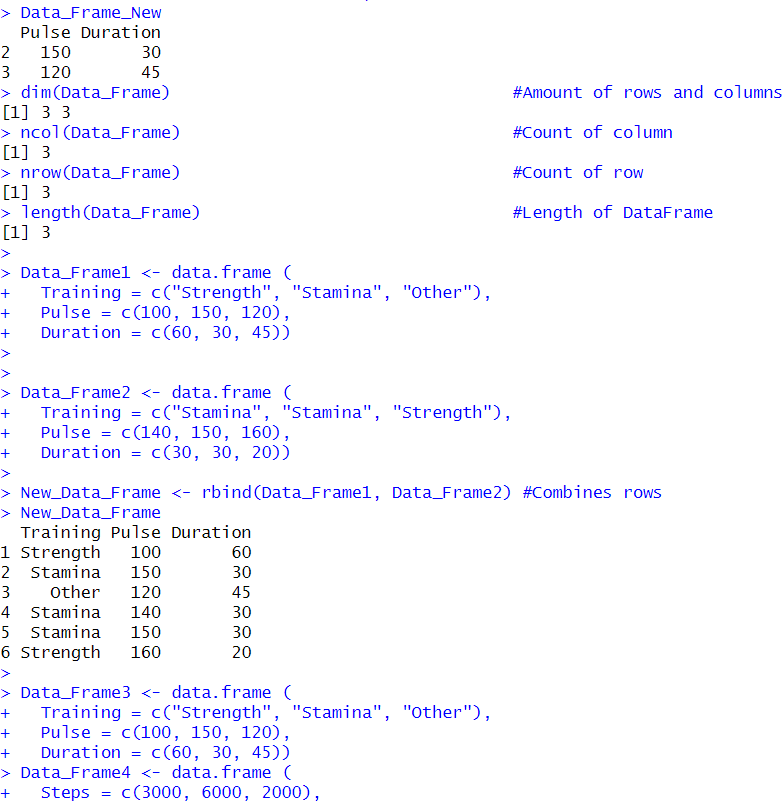
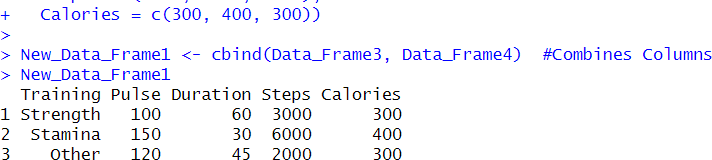
Duration = c(60, 30, 45)) Data\_Frame4 <- data.frame ( Steps = c(3000, 6000, 2000),

Calories = c(300, 400, 300))

New\_Data\_Frame1 <- cbind(Data\_Frame3, Data\_Frame4)

New\_Data\_Frame1

**Output**



**Program 9 : Implementation of Table**

**R Script**

tab <- c(1,2,3,4,5,6,7,8,9,0)

table(tab)

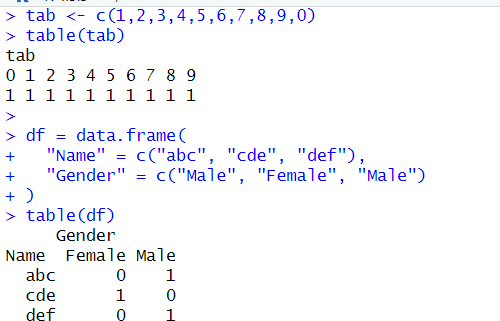
df = data.frame(

"Name" = c("abc", "cde", "def"),

"Gender" = c("Male", "Female", "Male")

)

table(df)

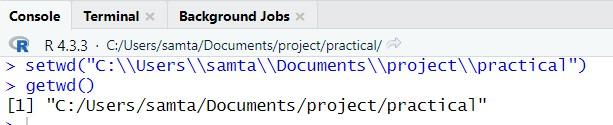
**Output**

**Program 10 : Implementation of setwd() and getwd()**

**R Script**

setwd("C:\\Users\\samta\\Documents\\project\\practical") getwd()

**Output**



**Program 11 : Using data()**

**R Script** data() **Output**

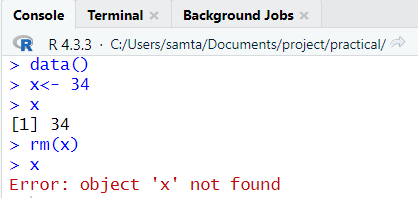
**Program 13 : Using rm()**

**R Script**

x<- 34

x rm(x) x

**Output**



**Program 14 : Attaching and Detaching Data R Script**

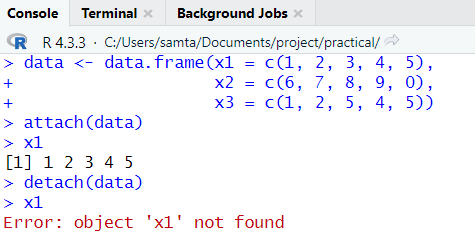
data <- data.frame(x1 = c(1, 2, 3, 4, 5),

x2 = c(6, 7, 8, 9, 0),

x3 = c(1, 2, 5, 4, 5))

attach(data) x1 detach(data) x1

**Output**



**Program 15 : Add rows and columns using rbind() and cbind() R Script**

Data\_Frame <- data.frame (

Training = c("Strength", "Stamina", "Other"), Pulse = c(100, 150, 120),

Duration = c(60, 30, 45)

)

Data\_Frame

New\_row\_DF <- rbind(Data\_Frame, c("Strength", 110, 110)) # Add a new row New\_row\_DF

New\_col\_DF <- cbind(Data\_Frame, Steps = c(1000, 6000, 2000)) # Add a new column

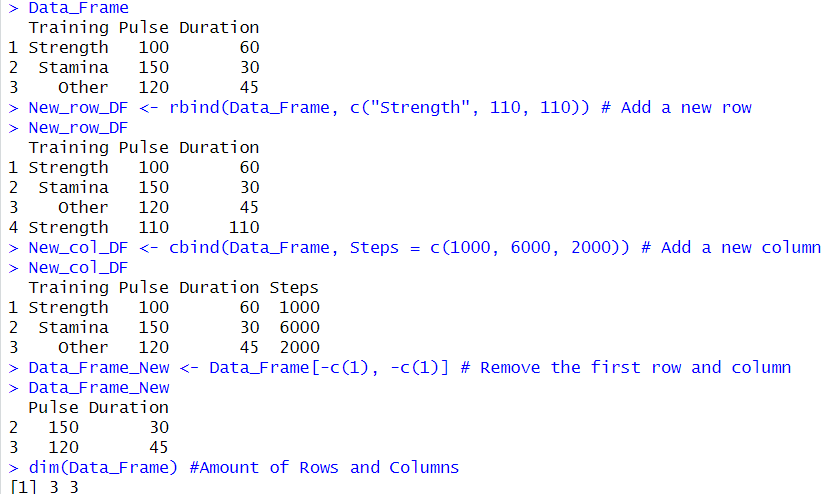
New\_col\_DF

Data\_Frame\_New <- Data\_Frame[-c(1), -c(1)] # Remove the first row and column

Data\_Frame\_New

dim(Data\_Frame) #Amount of Rows and Columns

**Output**



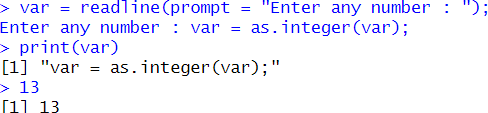
**Program 16 – Reading data from Console.**

**R Script**

var = readline(prompt = "Enter any number : "); var = as.integer(var);

print(var)

**Output**



**Program 17 - Loading data from different data sources.(CSV).**

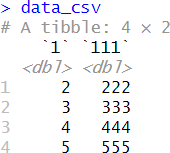
**R Script**

install.packages("readr") library(readr)

data\_csv <- read\_csv("students.csv") data\_csv

**Output**

****

****

**Program 18 - Loading data from different data sources.(Excel).**

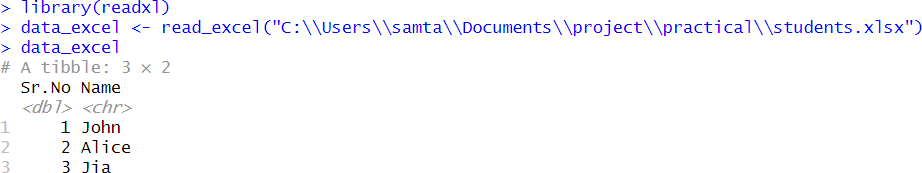
**R Script**

install.packages("readxl") library(readxl) data\_excel <-

read\_excel("C:\\Users\\samta\\Documents\\project\\practical\\students. xlsx")

data\_excel

**Output**



**Code:**

# Load required package

library(ggplot2)

# 1. Histogram

ggplot(data.frame(values = rnorm(1000)), aes(x = values)) +

geom\_histogram(binwidth = 0.2, fill = "blue", color = "black", alpha = 0.7) +

labs(title = "Histogram", x = "Values", y = "Frequency")

# 2. Boxplot

ggplot(data.frame(values = rnorm(100)), aes(y = values)) +

geom\_boxplot(fill = "orange", color = "black", outlier.colour = "red") +

labs(title = "Boxplot", y = "Values")

# 3. Bar Chart

data <- data.frame(

categories = c("A", "B", "C", "D"),

values = c(10, 15, 20, 25)

)

ggplot(data, aes(x = categories, y = values, fill = categories)) +

geom\_bar(stat = "identity") +

labs(title = "Bar Chart", x = "Categories", y = "Values") +

theme\_minimal()

# 4. Line Graph

data <- data.frame(

time = 1:10,

value = c(2, 3, 5, 7, 6, 8, 9, 12, 13, 15)

)

ggplot(data, aes(x = time, y = value)) +

geom\_line(color = "green", size = 1) +

geom\_point(color = "red", size = 2) +

labs(title = "Line Graph", x = "Time", y = "Value") +

theme\_minimal()

# 5. Scatterplot

data <- data.frame(

x = rnorm(100),

y = rnorm(100)

)

ggplot(data, aes(x = x, y = y)) +

geom\_point(color = "blue", size = 3) +

labs(title = "Scatterplot", x = "X-axis", y = "Y-axis") +

theme\_minimal()

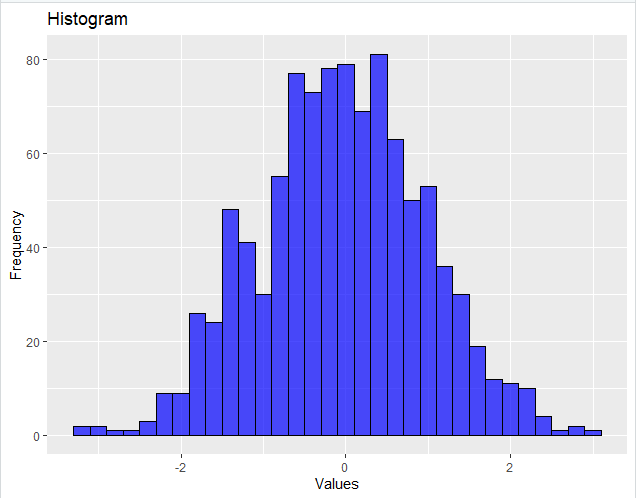
# 6. Pie Chart (Base R)

pie(c(40, 30, 20, 10), labels = c("Category A", "Category B", "Category C", "Category D"),

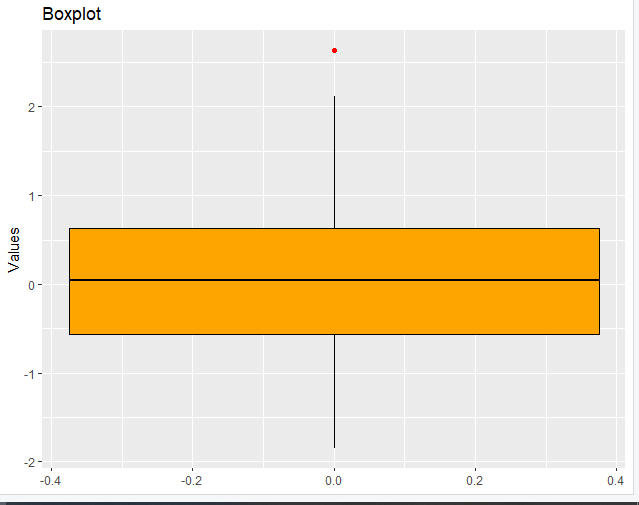
col = c("blue", "green", "red", "purple"), main = "Pie Chart")

**Output:**

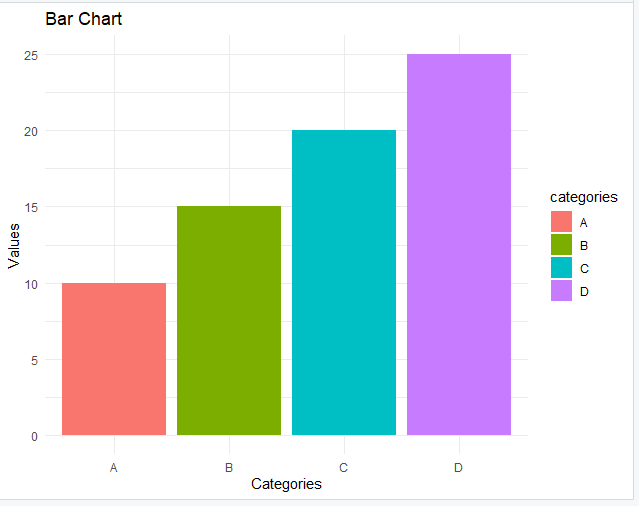
**1. Histogram**

****

**2. Boxplot**



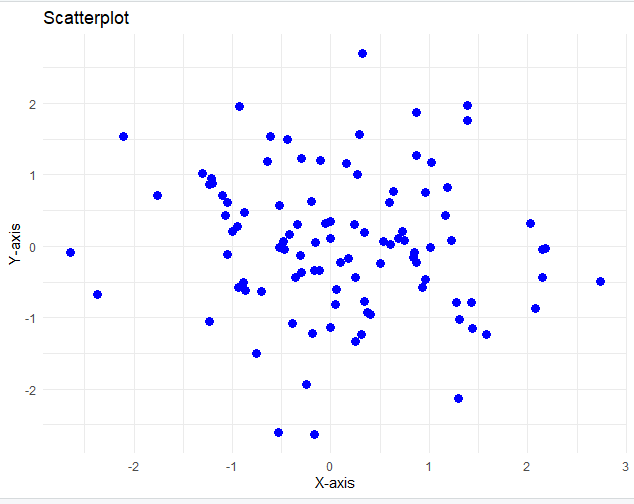
**3. Bar Chart**



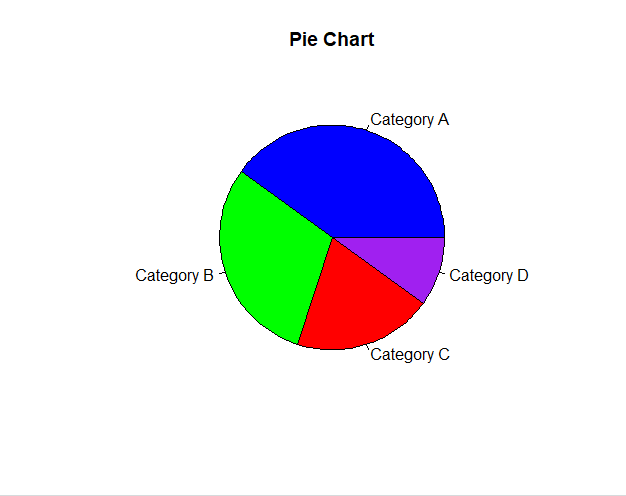
**4. Line Graph**



**5. Scatterplot**



**6. Pie Chart**



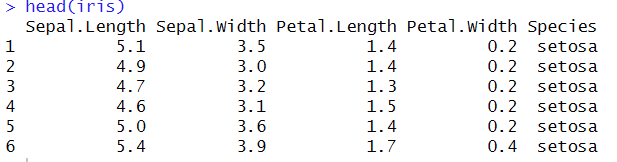
**Conclusion:**

Each plot serves a distinct purpose: Histograms and Boxplots analyze data distribution, Bar Charts and Pie Charts compare categories and proportions, and Line Graphs and Scatterplots reveal trends and relationships. Together, they demonstrate the essential methods for quickly summarizing and understanding any dataset.

**Code :**

head(iris)

**Output :**



**Dealing with Categorical Data**

Categorical data represents distinct groups or categories. Proper handling involves converting categorical variables into a format suitable for analysis, such as converting them to factors or using one-hot encoding.

**Code :**

# Load necessary library

library(ggplot2)

# Load the built-in iris dataset

data(iris)

# View the first few rows

head(iris)

# Check the structure to identify categorical variables

str(iris)

# The 'Species' column is categorical

# Summary of categorical data

table(iris$Species)

# Bar plot showing count of each Species

ggplot(iris, aes(x = Species, fill = Species)) +

geom\_bar() +

labs(title = "Count of Iris Species",

x = "Species",

y = "Count") +

theme\_minimal()

# Box plot to compare Sepal.Length across Species

ggplot(iris, aes(x = Species, y = Sepal.Length, fill = Species)) +

geom\_boxplot() +

labs(title = "Sepal Length Distribution by Species",

x = "Species",

y = "Sepal Length") +

theme\_light()

# Violin plot for a smoother distribution view

ggplot(iris, aes(x = Species, y = Petal.Width, fill = Species)) +

geom\_violin(trim = FALSE) +

labs(title = "Petal Width Distribution by Species",

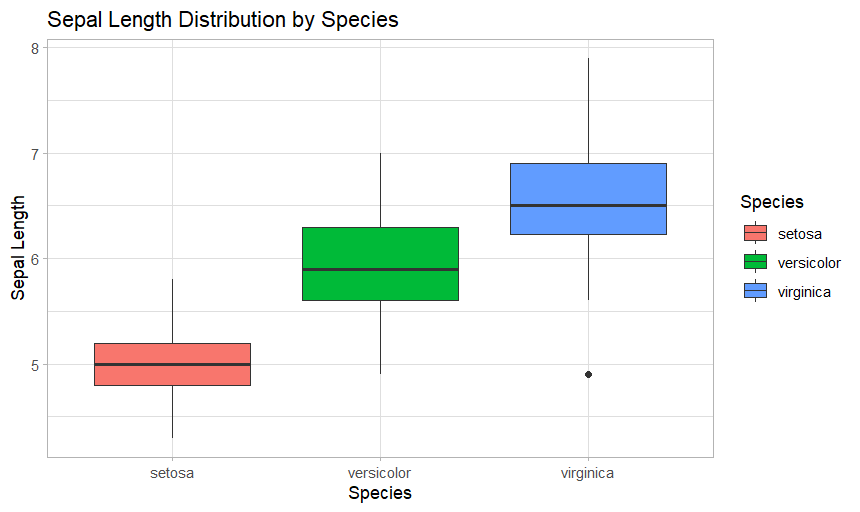
x = "Species",

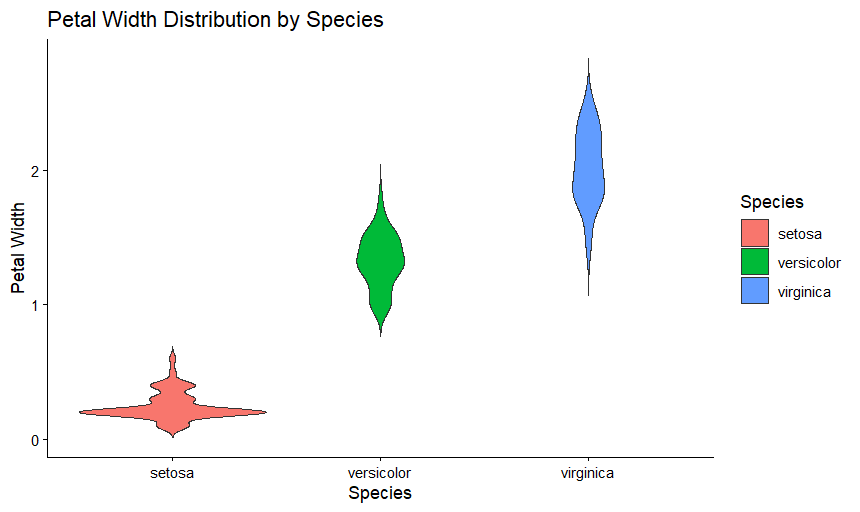
y = "Petal Width") +

theme\_classic()

**Output :**







| **table(iris$Species)** → summarizes the count of each categorical level.  **geom\_bar()** → visualizes frequency of categories.  **geom\_boxplot() and geom\_violin()** → compare numeric features across categories.  The Species variable is categorical and used to group the data visually. |
| --- |

**Data Reduction using Subsetting**

Data reduction through subsetting involves selecting specific rows or columns from a dataset to simplify analysis, focus on relevant data, or reduce the size of the dataset.

**Code :**

# Load the built-in iris dataset

data(iris)

# View the first few rows

head(iris)

# Check the structure of the dataset

str(iris)

# The dataset has 150 rows and 5 columns

# Let's perform data reduction by subsetting specific rows and columns

# 1. Subset: Selecting specific columns (only Sepal.Length and Species)

iris\_subset\_cols <- iris[, c("Sepal.Length", "Species")]

head(iris\_subset\_cols)

# 2. Subset: Selecting specific rows (first 30 observations)

iris\_subset\_rows <- iris[1:30, ]

head(iris\_subset\_rows)

# 3. Subset: Filtering data for a specific Species (e.g., "setosa")

iris\_setosa <- subset(iris, Species == "setosa")

head(iris\_setosa)

# 4. Subset: Filtering based on condition (e.g., Sepal.Length > 6)

iris\_large\_sepal <- subset(iris, Sepal.Length > 6)

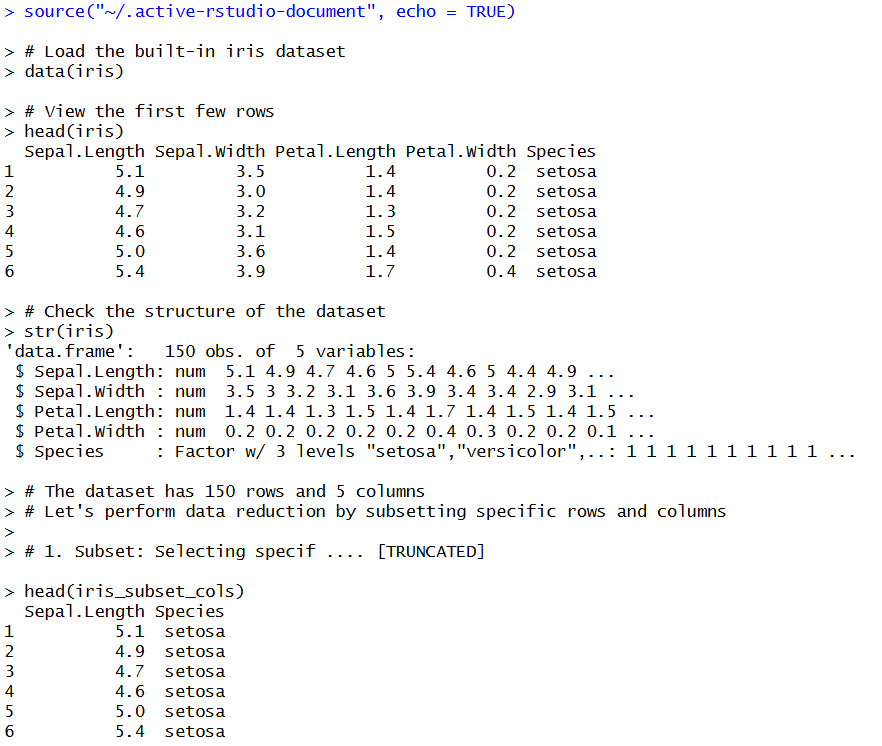
head(iris\_large\_sepal)

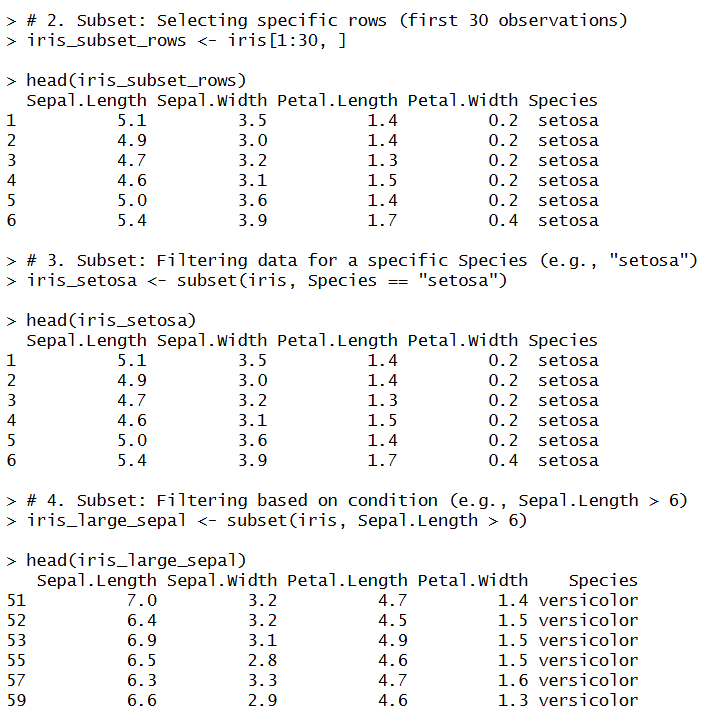
# 5. Combine conditions: Species = "versicolor" and Petal.Width > 1.2

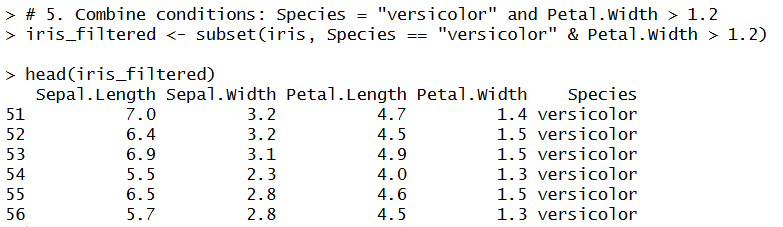
iris\_filtered <- subset(iris, Species == "versicolor" & Petal.Width > 1.2)

head(iris\_filtered)

**Output :**

****

****

****

**Code :**

# Load libraries

library(ggplot2)

library(tidyr)

# Reshape iris into long format

iris\_long <- iris %>%

pivot\_longer(cols = c(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width),

names\_to = "Feature",

values\_to = "Value")

# Plot histograms in a grid

ggplot(iris\_long, aes(x = Value, fill = Species)) +

geom\_histogram(color = "black", alpha = 0.7, bins = 20) +

facet\_wrap(~Feature, scales = "free\_x") +

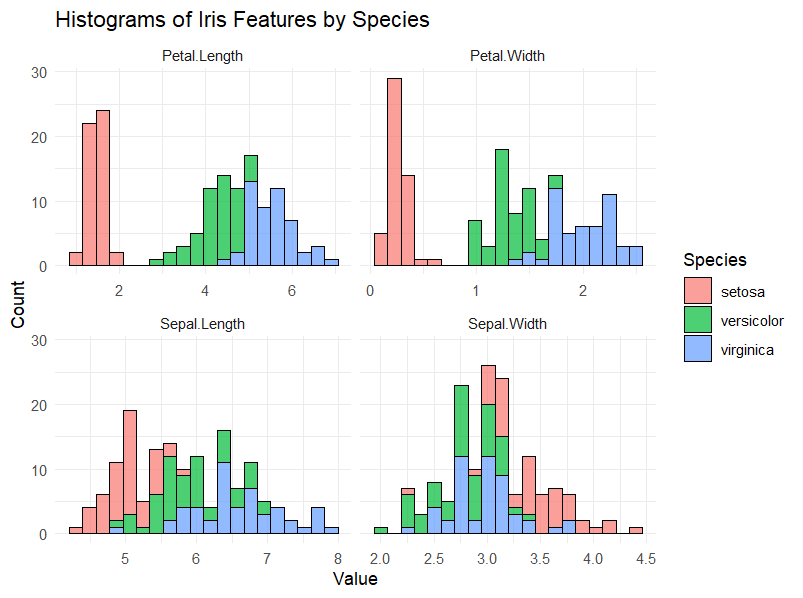
labs(title = "Histograms of Iris Features by Species",

x = "Value",

y = "Count") +

theme\_minimal()

**Output :**



**Scatterplots**

**Purpose:** Scatterplots visualize the relationship between two continuous variables, with each point representing an observation.

**Code :**

ggplot(iris, aes(x = Petal.Length, y = Petal.Width, color = Species)) +

geom\_point(size = 3, alpha = 0.7) +

geom\_smooth(method = "lm", se = FALSE) +

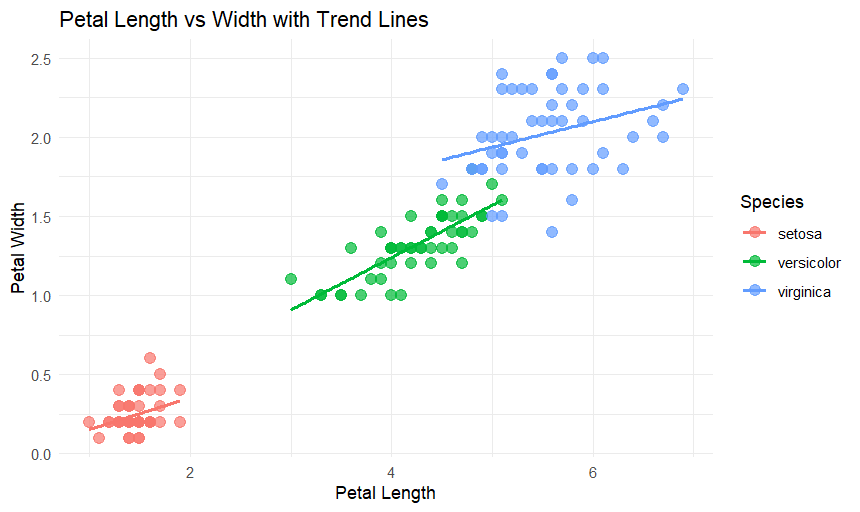
labs(title = "Petal Length vs Width with Trend Lines",

x = "Petal Length",

y = "Petal Width") +

theme\_minimal()

**Output :**

****

**Conclusion:**

The implementation of data visualization using ggplot2 in R provides an effective way to analyze and interpret data through graphical representation. It simplifies the process of identifying trends, patterns, and relationships within datasets using clear and customizable plots. Data reduction through subsetting in the **Iris dataset** allows efficient handling and focused analysis of relevant features or specific species, improving computational efficiency and interpretability.

**Creating a sample Data Frame:**

set.seed(123)

data <- data.frame(

A = rnorm(100, mean = 50, sd = 10),

B = rnorm(100, mean = 30, sd = 5),

C = rnorm(100, mean = 70, sd = 8),

D = rnorm(100, mean = 45, sd = 12)

)

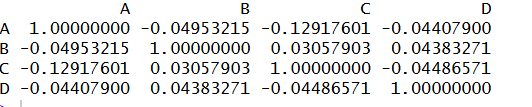
print(head(data))

**Calculating the Correlation Matrix:**

cor\_matrix <- cor(data)

print(cor\_matrix)

**Output:**

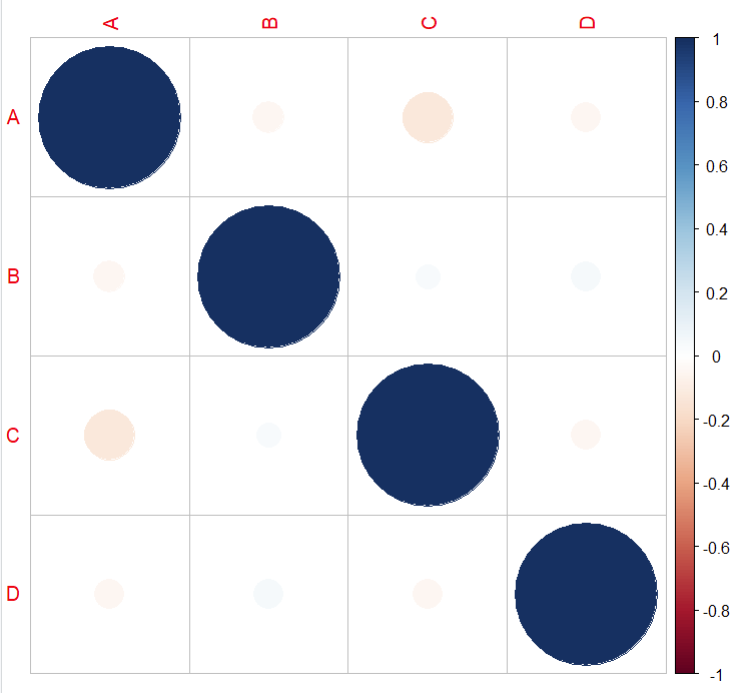
****

**Correlation Plot using Corrplot package:**

library(corrplot)

corrplot(cor\_matrix, method = "circle")

**Output:**

****

You can adjust the method and appearance by tweaking parameters such as:

method: Change the visualization method (e.g., "circle", "square", "ellipse").

type: Whether to plot the full matrix or just the upper triangle ("lower", "upper").

col: Adjust the color palette.

addCoef.col: Add correlation coefficients to the plot.

**Code:**

corrplot(

cor\_matrix,

method = "circle",

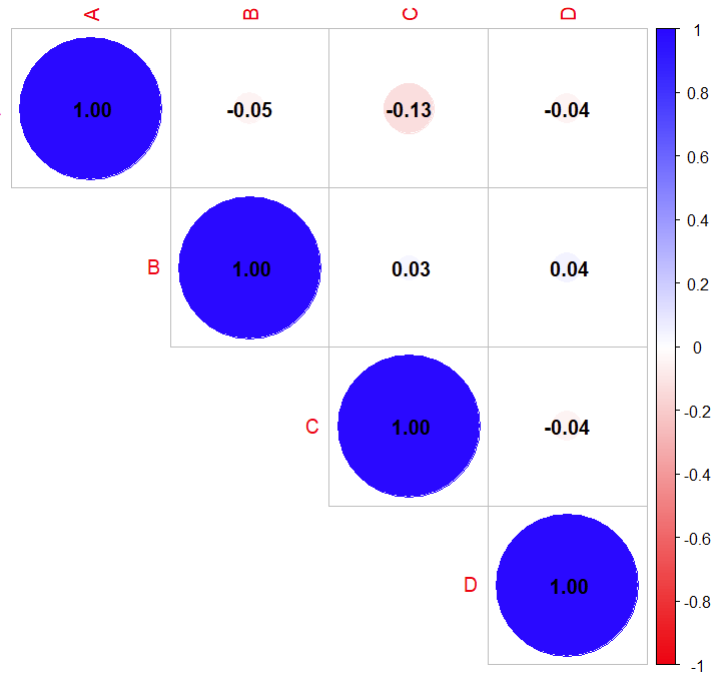
type = "upper",

col = colorRampPalette(c("red", "white", "blue"))(200),

addCoef.col = "black"

)

**Output:**

****

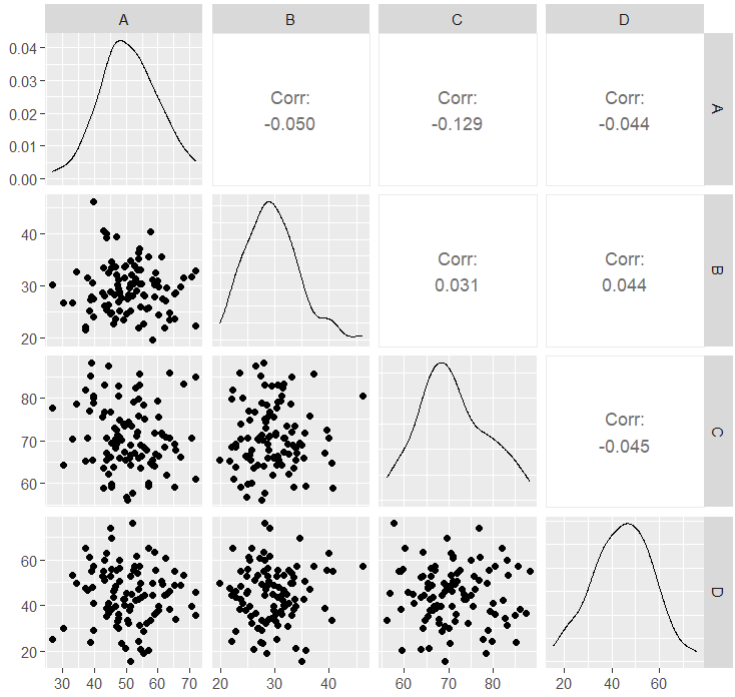
**Correlation Plot using GGally package:**

**Code:**

library(GGally)

ggpairs(data)

**Output:**

****

**Correlation Plot using ggplot2 and geom\_tile package:**

**Code:**

library(ggplot2)

library(reshape2)

cor\_matrix <- cor(data)

cor\_melt <- melt(cor\_matrix)

ggplot(cor\_melt, aes(Var1, Var2, fill = value)) +

geom\_tile() +

scale\_fill\_gradient2(low = "red", high = "blue", mid = "white", midpoint = 0) +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1)) +

ggtitle("Correlation Heatmap")

In this plot:

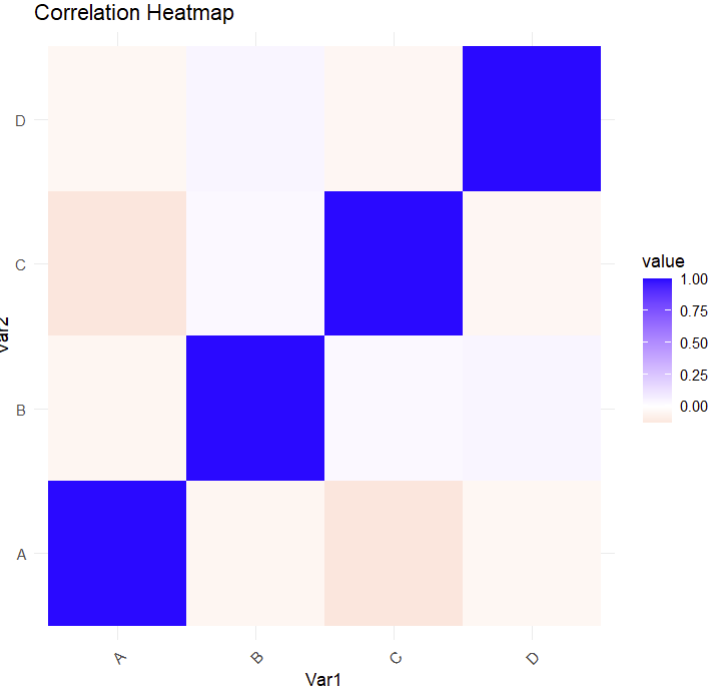
●melt() is used to reshape the correlation matrix into a format that ggplot2 can use.

●geom\_tile() creates the heatmap with color representing the correlation values.

●scale\_fill\_gradient2() defines a color scale for the correlation values (you can

customize the colors).

**Output:**

****

**Summary Statistics**

set.seed(123)

data <- data.frame(

A = rnorm(100, mean = 50, sd = 10),

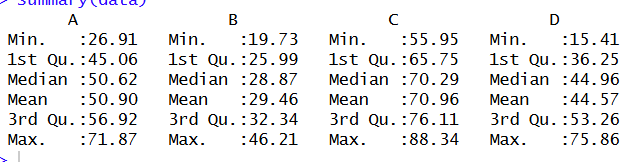
B = rnorm(100, mean = 30, sd = 5),

C = rnorm(100, mean = 70, sd = 8),

D = rnorm(100, mean = 45, sd = 12)

)

summary(data)



**Histogram for Each Variable**

data\_long <- pivot\_longer(data, cols = everything(),

names\_to = "Variable", values\_to = "Value")

ggplot(data\_long, aes(x = Value)) +

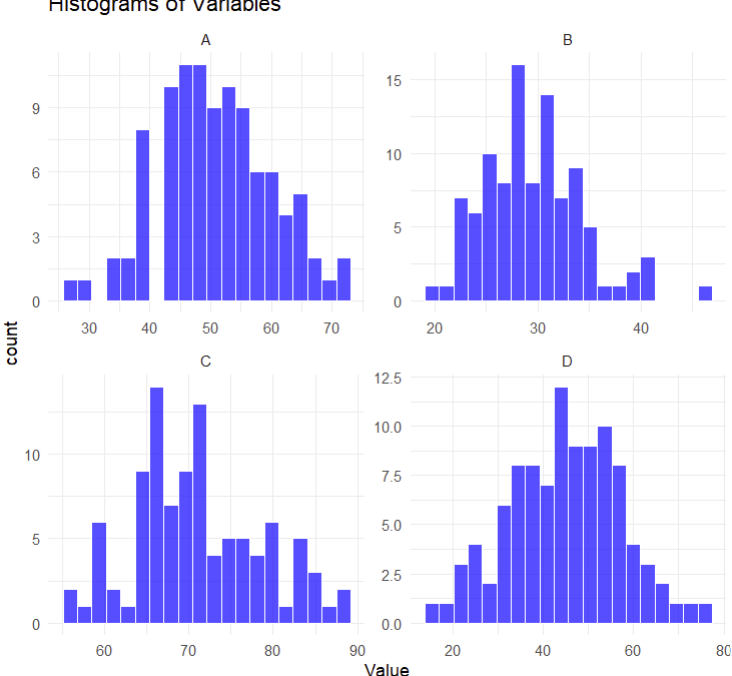
geom\_histogram(bins = 20, fill = "blue", color = "white", alpha = 0.7) +

facet\_wrap(~Variable, scales = "free") +

theme\_minimal() +

ggtitle("Histograms of Variables")

**Output:**

****

**Box Plots**

**Code:**

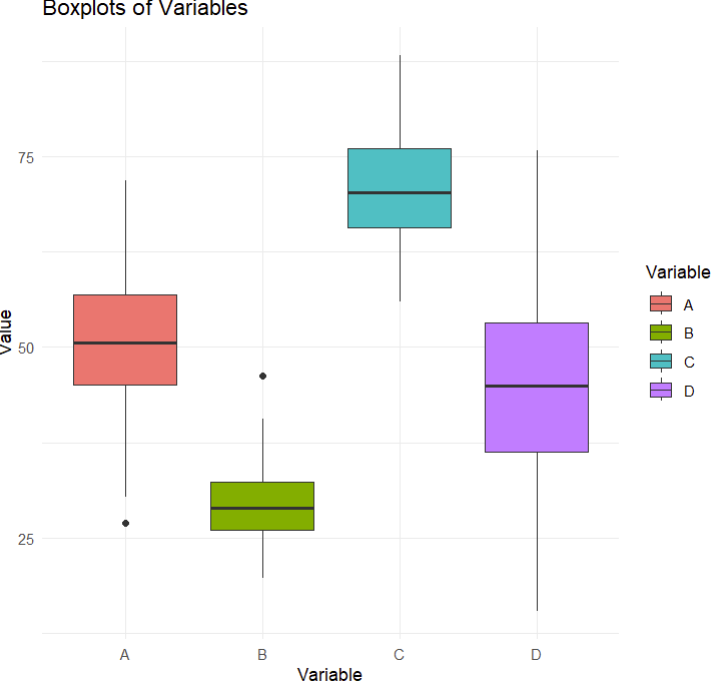
ggplot(data\_long, aes(x = Variable, y = Value, fill = Variable)) +

geom\_boxplot() +

theme\_minimal() +

ggtitle("Boxplots of Variables")

**Output:**

****

## **Conclusion:**

In this practical, we implemented correlation plots and performed Exploratory Data Analysis (EDA) using R.  
We visualized relationships using *corrplot*, *GGally*, and *ggplot2*. Correlation heatmaps and pairwise plots revealed dependencies between variables, and EDA plots like histograms and boxplots highlighted data distribution and variability.These methods help in understanding data patterns and preparing it for modeling.

**Aim :**  To implement Normal and Binomial distributions, Univariate and Bivariate Analysis.

**Title :** Implementation of Normal and Binomial distributions, Univariate and Bivariate Analysis

**Description :**

### **Normal Distribution -** A continuous, symmetric, bell-shaped distribution. Defined by **mean (μ)** and **standard deviation (σ)**.

### **Use :** many natural measurements (height, weight, test scores) approximate Normal due to the central limit theorem.

### **Binomial Distribution -** A discrete distribution showing the number of successes in *n* independent trials Defined by **number of trials (n)** and **probability of success (p)**.

**Use :** Modeling counts of successes (e.g., number of defectives, hits in fixed number of trials).

### **Univariate Analysis -** Analysis of a **single variable** — focuses on distribution, central tendency, and dispersion.

**Use :** Used to **summarize and understand one variable** — e.g., analyzing customer age distribution, product prices, or exam scores to identify patterns, averages, and outliers.

### **Bivariate Analysis -** Analysis of the **relationship between two variables** — correlation, covariance, scatter plots, etc.

### **Use :** Used to study relationships between two variables — e.g., examining how advertising spend affects sales, how height relates to weight, or how income varies by education level.

**R Script :**

# Load required libraries

library(ggplot2)

# Normal Distribution

# Generate random data following Normal Distribution

set.seed(123)

normal\_data <- rnorm(1000, mean = 50, sd = 10)

# Summary statistics

summary(normal\_data)

# Histogram with density curve

ggplot(data.frame(x = normal\_data), aes(x)) +

geom\_histogram(aes(y = ..density..), bins = 30, fill = "skyblue", color = "black") +

geom\_density(color = "red", size = 1) +

labs(title = "Normal Distribution", x = "Values", y = "Density")

# Binomial Distribution

# Generate Binomial data (n=10 trials, p=0.5 success prob)

binomial\_data <- rbinom(1000, size = 10, prob = 0.5)

# Summary statistics

summary(binomial\_data)

# Bar plot

ggplot(data.frame(x = binomial\_data), aes(x = factor(x))) +

geom\_bar(fill = "lightgreen", color = "black") +

labs(title = "Binomial Distribution", x = "Number of Successes", y = "Frequency")

# Univariate Analysis

# Example dataset

data <- data.frame(

Age = c(21, 25, 30, 22, 23, 31, 28, 35, 40, 27)

)

# Descriptive statistics

summary(data$Age)

sd(data$Age)

var(data$Age)

# Histogram

ggplot(data, aes(x = Age)) +

geom\_histogram(fill = "orange", color = "black", bins = 5) +

labs(title = "Univariate Analysis - Age Distribution", x = "Age", y = "Frequency")

# Bivariate Analysis

# Example dataset

biv\_data <- data.frame(

Height = c(150, 160, 165, 170, 175, 180, 185, 190),

Weight = c(50, 55, 60, 65, 70, 75, 80, 85)

)

# Correlation and covariance

cor(biv\_data$Height, biv\_data$Weight)

cov(biv\_data$Height, biv\_data$Weight)

# Scatter plot

ggplot(biv\_data, aes(x = Height, y = Weight)) +

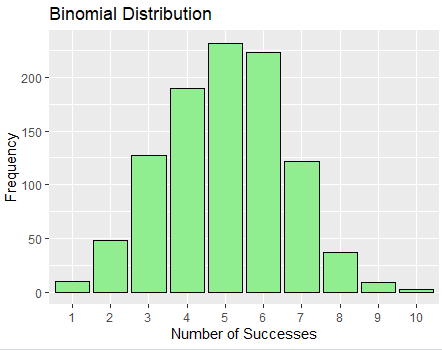
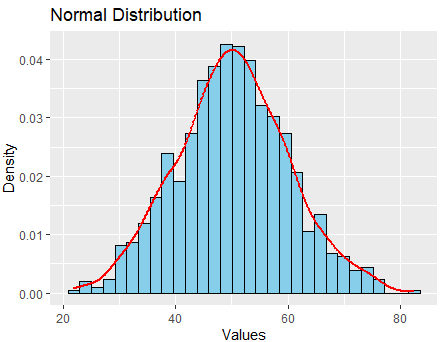
geom\_point(color = "blue", size = 3) +

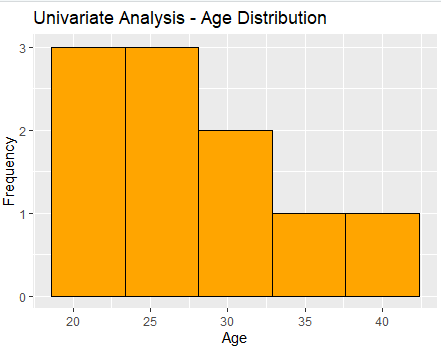
geom\_smooth(method = "lm", se = FALSE, color = "red") +

labs(title = "Bivariate Analysis - Height vs Weight",

x = "Height (cm)", y = "Weight (kg)")

**Output :**





### **Conclusion**

Normal and Binomial distributions help in understanding the patterns and probabilities of continuous and discrete data, respectively. Univariate analysis provides insights into the characteristics of a single variable, while bivariate analysis explores relationships between two variables. Together, these methods form the foundation of statistical analysis, enabling better data interpretation and informed decision-making.

**Aim** : To implement and analyse Apriori Algorithm using Market Basket Analysis.

**Title** : Implementation and analysis of Apriori Algorithm using Market Basket Analysis

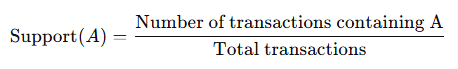
**Description** :

The **Apriori Algorithm** is a **classic data mining technique** used for discovering **frequent itemsets** and generating **association rules** from transactional data — for example, market basket data from a supermarket. In simple words : “If a customer buys *bread* and *milk*, how likely are they to also buy *butter*?”

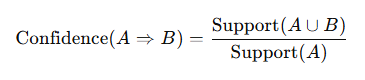
## **How Apriori Works — Step by Step**

### **Step 1: Define Key Metrics**

1. **Support** — frequency of an itemset in the dataset

  
 Example: If “Milk” appears in 3 out of 4 transactions → Support = 3/4 = 0.75

1. **Confidence** — strength of an implication rule

  
 Example: If 2 out of 3 “Milk” transactions also contain “Butter” → Confidence = 2/3 ≈ 0.67

1. **Lift** — how much more likely B is bought when A is bought compared to random chance



Lift > 1 → positive correlation

Lift = 1 → independent

Lift < 1 → negative correlation

### **Step 2: Find Frequent Itemsets**

Apriori starts by finding **individual items** that meet a **minimum support threshold**.

1. Generate **1-itemsets** (e.g., {Milk}, {Bread}, {Butter})  
    Keep only those with enough support.
2. Combine frequent 1-itemsets to form **2-itemsets** (e.g., {Milk, Bread}).  
    Keep those meeting the minimum support.
3. Continue generating **k-itemsets** until no more frequent itemsets can be found.  
   **Apriori Principle**:

If an itemset is **infrequent**, then all of its **supersets** must also be infrequent.  
 This principle helps reduce the search space (improves efficiency).

### **Step 3: Generate Association Rules**

Once frequent itemsets are found, the algorithm generates rules of the form:

A⇒B

(where A and B are disjoint itemsets)

Rules are evaluated based on:

* **Confidence** (how reliable the rule is)
* **Lift** (how strong the association is)

**R Script :**

install.packages("arules") # Install Packages

install.packages("arulesViz")

library(arules) # Load libraries

library(arulesViz)

data("Groceries")

summary(Groceries)

inspect(Groceries[1:5])

# Apply Apriori Algorithm

rules <- apriori(Groceries, parameter = list(support = 0.005, confidence = 0.25, minlen = 2))

summary(rules) #Summary

inspect(rules[1:10])

rules\_sorted <- sort(rules, by = "lift", decreasing = TRUE) #Sort by lift

inspect(rules\_sorted[1:10]) #View top 10 rules with highest lift

plot(rules) #Basic scatter plot

plot(rules, method = "grouped") #Group Matrix Plot

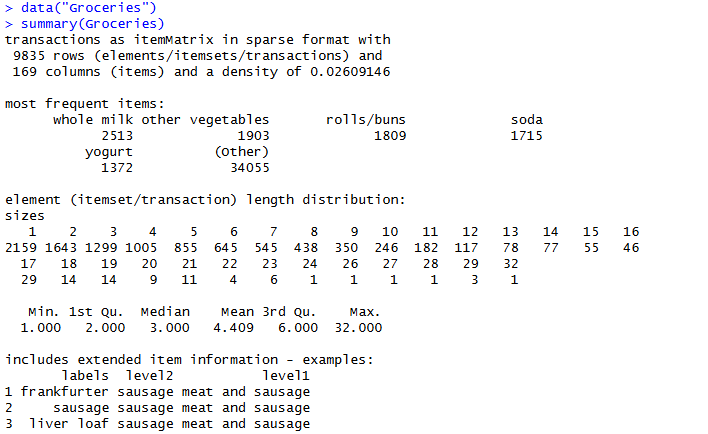
plot(rules, method = "graph", interactive = TRUE) #Graph-based visualization

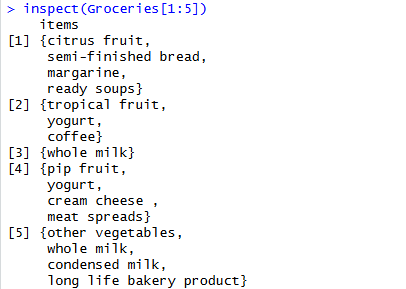
plot(rules, method = "paracoord", control = list(reorder = TRUE)) #Parallel coordinates visualization

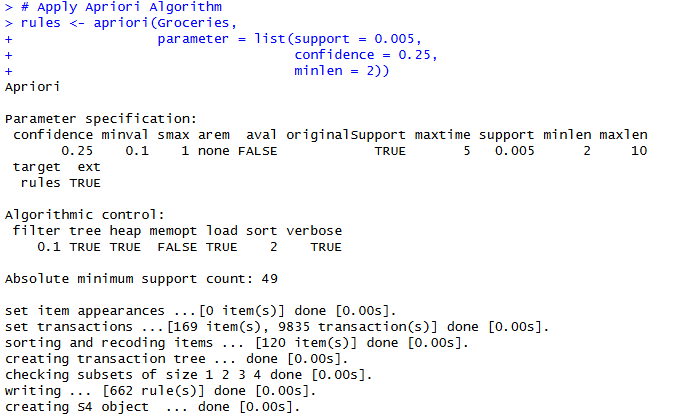
strong\_rules <- subset(rules, lift > 3 & confidence > 0.4) #Filter Strong Rules

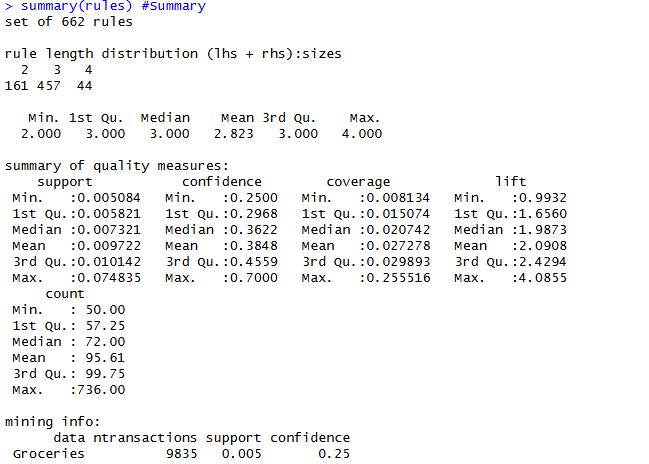
inspect(strong\_rules)

**Output :**

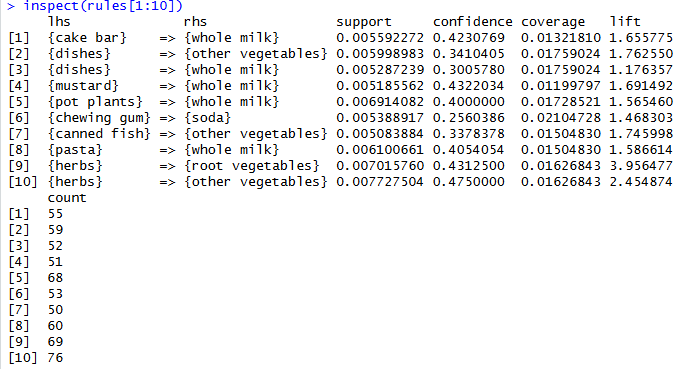


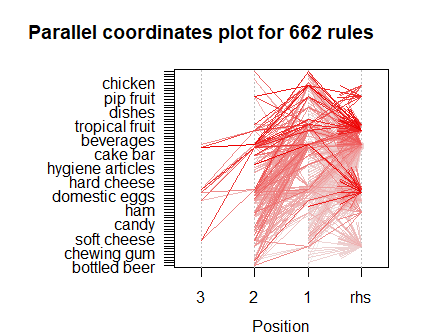




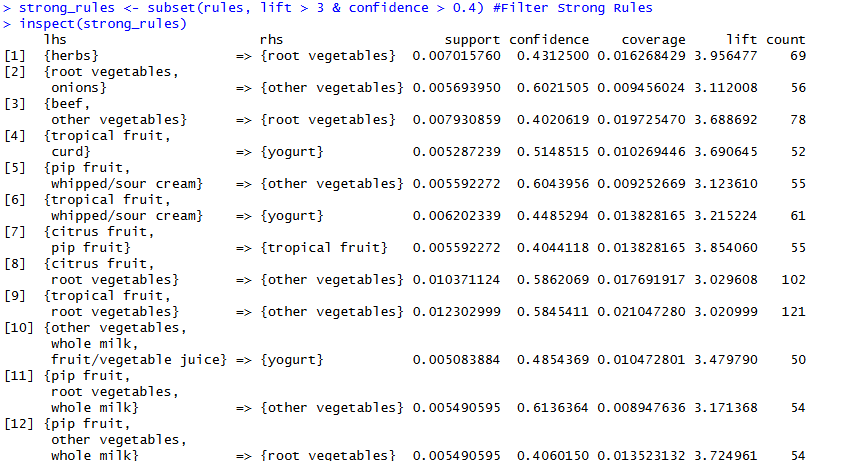












**Conclusion :** The Apriori algorithm efficiently discovers hidden patterns and associations in transactional data. It helps identify relationships between items, making it valuable for market basket analysis, recommendations, and business decision-making.

**Aim** : To implement and analyse Linear regression through graphical methods.

**Title :** Implementation and analysis of Linear regression through graphical methods.

**Description :**

Linear regression is one of the most fundamental statistical techniques used to model and analyze the relationship between a **dependent variable (Y)** and one or more **independent variables (X)**.  
 It assumes a **linear relationship** between the variables, meaning that changes in the predictor variable(s) cause proportional changes in the response variable.

In this experiment, we use **R programming** to implement linear regression, visualize the fitted model, and analyze its performance using **graphical methods** such as scatter plots, regression lines, and diagnostic plots.

### **1. Scatter Plot**

A **scatter plot** is used to visualize the relationship between two variables — the independent variable (X) and the dependent variable (Y).  
 In linear regression, it helps determine whether the relationship between X and Y appears **linear**. Adding a regression line to the scatter plot shows how well the model fits the data.

### **2. Diagnostic Plot**

**Diagnostic plots** are used to check whether the assumptions of linear regression are satisfied.  
 The four default diagnostic plots in R help analyze:

* Linearity of data
* Normality of residuals
* Homoscedasticity (constant variance)
* Detection of outliers and influential points

They help assess the model’s reliability and fit quality.

### **3. ggplot2 Regression Plot**

The **ggplot2 regression plot** provides a visually enhanced version of the scatter plot with a regression line and a **confidence interval** around it.  
 It helps in visualizing how well the regression model fits the data, while also showing the range of uncertainty in predictions.

**4. Residuals vs Fitted Plot**

This plot shows **residuals (errors)** on the y-axis and **fitted values** on the x-axis.  
 It is used to check:

* Linearity (residuals should be randomly scattered)
* Homoscedasticity (constant spread of residuals)  
   If any pattern appears, it indicates model misspecification or non-linearity.

### **5. Normal Q-Q Plot**

A **Normal Q-Q (Quantile-Quantile) plot** checks whether the residuals are **normally distributed**, which is an assumption of linear regression.  
 If the residuals follow the diagonal line closely, they are approximately normal. Large deviations indicate non-normality or outliers.

**R Script**

if(!require(ggplot2)) install.packages("ggplot2", dependencies = TRUE)

library(ggplot2)

set.seed(123) # Generate synthetic dataset

x <- 1:50

y <- 2.5 \* x + rnorm(50, mean = 0, sd = 10)

data <- data.frame(x, y)

model <- lm(y ~ x, data = data) # Fit Linear Regression Model

cat("===== LINEAR REGRESSION SUMMARY =====\n")

print(summary(model)) # Display summary

plot(data$x, data$y,

main = "Scatter Plot with Regression Line",

xlab = "X variable", ylab = "Y variable",

pch = 19, col = "blue") # Scatter Plot with Regression Line (Base R)

abline(model, col = "red", lwd = 2)

par(mfrow = c(2, 2)) # Diagnostic Plots (Base R)

plot(model)

ggplot(data, aes(x = x, y = y)) +

geom\_point(color = "darkblue") +

geom\_smooth(method = "lm", color = "red", fill = "pink") +

labs(title = "Linear Regression using ggplot2",

x = "X variable", y = "Y variable") # ggplot2 Regression Plot with Confidence Interval

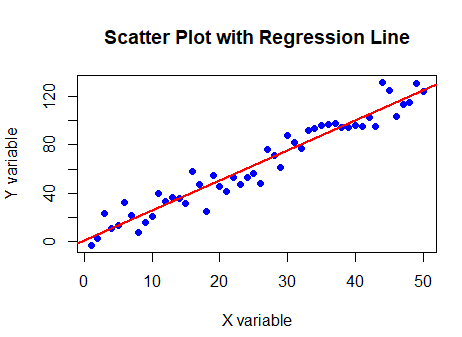
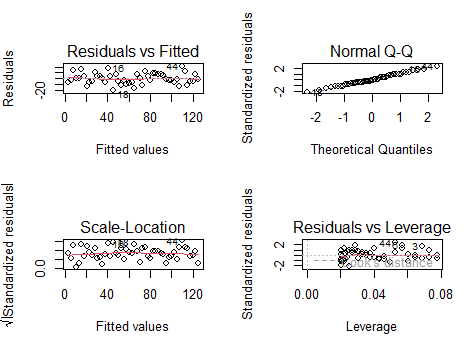
ggplot(model, aes(.fitted, .resid)) + geom\_point(color = "blue") + geom\_hline(yintercept = 0, color = "red") + labs(title = "Residuals vs Fitted Values", x = "Fitted Values", y = "Residuals") ggplot(model, aes(sample = .resid)) +

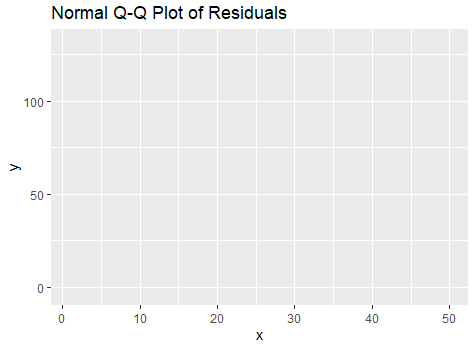
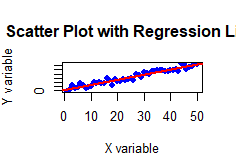
stat\_qq() +

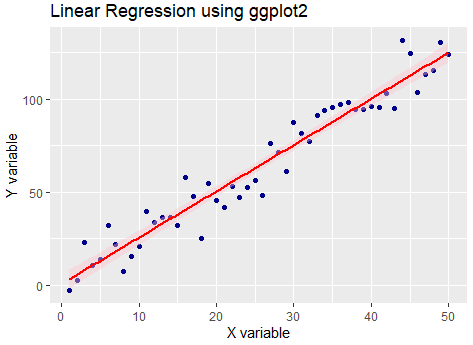
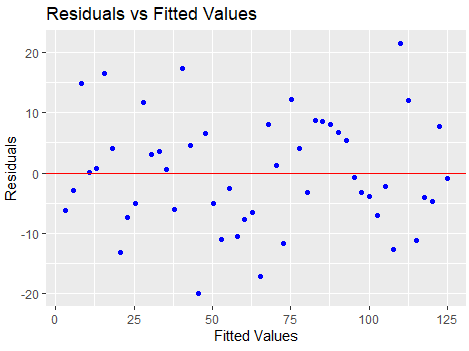
stat\_qq\_line(color = "red") +

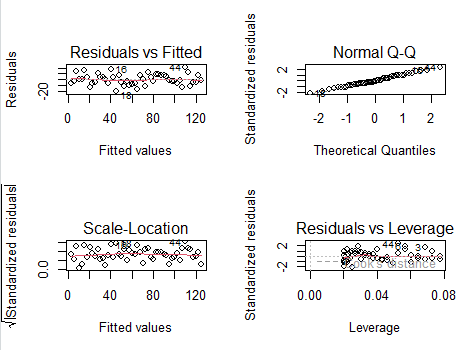
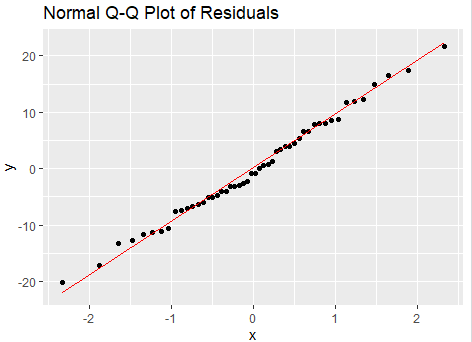
labs(title = "Normal Q-Q Plot of Residuals") # Normal Q-Q Plot for Residuals

**Output**

** **

** **

** **

****

**Conclusion :** Graphical analysis in R helps verify the fit and assumptions of a linear regression model. Scatter and regression plots show the relationship, while diagnostic and Q-Q plots

**Aim :** To Implement and analyse Classification algorithms: Naive Bayesian, K-Nearest Neighbour, ID3, C4.5

**Title :** Implementation and analysis of Classification algorithms: Naive Bayesian, K-Nearest Neighbour, ID3, C4.5

**Description :**

Classification is a fundamental task in machine learning and data mining that involves assigning data instances to predefined classes based on input features. Various algorithms exist to perform classification, each with its own underlying principles, advantages, and limitations.

This study focuses on four popular classification algorithms — **Naive Bayes**, **K-Nearest Neighbour (KNN)**, **ID3**, and **C4.5** — and their implementation and comparative analysis using the **R programming language**. The goal is to evaluate their performance on a benchmark dataset and understand their suitability for different problem types.

### 

### **Naive Bayes Classifier**

The **Naive Bayes** classifier is a probabilistic model based on **Bayes’ Theorem**, which describes the probability of a class given certain features. The term *naive* refers to the assumption that all input features are **conditionally independent** of each other given the class label.

### 

### **K-Nearest Neighbour (KNN)**

The **K-Nearest Neighbour** algorithm is an **instance-based**, non-parametric classifier. It classifies a new sample by analyzing the classes of its *k* nearest neighbors in the feature space, using a distance metric such as **Euclidean distance**.

### 

### **ID3 Algorithm (Iterative Dichotomiser 3)**

**ID3** is a **decision tree** algorithm developed by *Ross Quinlan*. It uses a **top-down greedy search** to build a tree by selecting the attribute that yields the **highest information gain** at each node.

### 

### **C4.5 Algorithm**

**C4.5** is an improved version of ID3, also developed by Quinlan. It overcomes the limitations of ID3 by:

Handling **both categorical and continuous** attributes, Managing **missing values,** Using **gain ratio** instead of information gain to avoid bias, Incorporating **tree pruning** to reduce overfitting

**R Script**

install.packages("e1071") # For Naive Bayes

install.packages("class") # For KNN

install.packages("rpart") # For Decision Trees

install.packages("rpart.plot") # For visualizing Decision Trees

# Load libraries

library(e1071)

library(class)

library(rpart)

library(rpart.plot)

# Load the iris dataset

data(iris)

# View the structure of the dataset

str(iris)

# Split the data into training and testing sets

set.seed(123) # For reproducibility

index <- sample(1:nrow(iris), 0.7 \* nrow(iris)) # 70% training, 30% testing

train\_data <- iris[index, ]

test\_data <- iris[-index, ]

# Train the Naive Bayes classifier

nb\_model <- naiveBayes(Species ~ ., data = train\_data)

# Predict on test data

nb\_predictions <- predict(nb\_model, test\_data)

# Evaluate accuracy

nb\_accuracy <- mean(nb\_predictions == test\_data$Species)

print(paste("Naive Bayes Accuracy:", round(nb\_accuracy \* 100, 2), "%"))

# Define features and labels for training and testing

train\_features <- train\_data[, -5]

test\_features <- test\_data[, -5]

train\_labels <- train\_data$Species

test\_labels <- test\_data$Species

# Apply KNN with k = 5

knn\_predictions <- knn(train\_features, test\_features, train\_labels, k = 5)

# Evaluate accuracy

knn\_accuracy <- mean(knn\_predictions == test\_labels)

print(paste("KNN Accuracy:", round(knn\_accuracy \* 100, 2), "%"))

# Train a decision tree using rpart (C4.5)

dt\_model <- rpart(Species ~ ., data = train\_data, method = "class")

# Visualize the decision tree

rpart.plot(dt\_model, main = "Decision Tree (C4.5)")

# Predict on test data

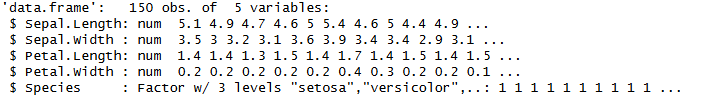
dt\_predictions <- predict(dt\_model, test\_data, type = "class")

# Evaluate accuracy

dt\_accuracy <- mean(dt\_predictions == test\_data$Species)

print(paste("Decision Tree Accuracy:", round(dt\_accuracy \* 100, 2), "%"))

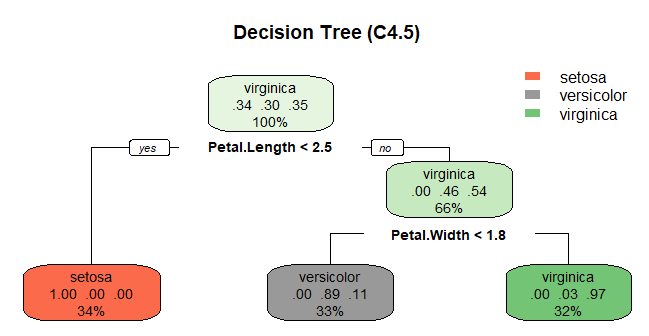
**Output**



****

****

****

****

**Conclusion :** The comparative analysis of Naive Bayes, K-Nearest Neighbour, ID3, and C4.5 algorithms in R shows that all perform effectively on the Iris dataset, but their efficiency varies with data characteristics. Naive Bayes is fast and simple, KNN is accurate but sensitive to noise, ID3 is easy to interpret yet prone to overfitting, and C4.5 provides the best balance of accuracy and robustness. Overall, C4.5 is the most reliable and versatile classifier among the four.

**Aim -** To implement and analyse the clustering algorithms : K-Means and hierarchical clustering.

**Title -** Implementation and analysis of the following clustering algorithms : K-Means and hierarchical clustering.

**Description -** Clustering is an **unsupervised machine learning technique** used to group data points into meaningful clusters based on similarity. Unlike supervised learning, clustering does not use labeled data. Its applications include data segmentation, pattern recognition, market basket analysis, image compression, and anomaly detection.

Two widely used clustering algorithms are:

1. **K-Means Clustering**
2. **Hierarchical Clustering**

# 

# **K–Means Clustering**

K-means is a centroid-based algorithm that partitions data into **K clusters**, where each cluster is represented by the mean (centroid) of data points within it.

**Working Steps**

1. Select the number of clusters **K**.
2. Initialize **K** centroids randomly.
3. Assign each data point to the nearest centroid (forming clusters).
4. Recalculate the centroids by computing the mean of each cluster.
5. Repeat steps 3–4 until centroids stabilize (i.e., no change in assignments).

# **Hierarchical Clustering**

Hierarchical clustering builds a multilevel hierarchy of clusters either by merging smaller clusters (**agglomerative**, bottom-up) or splitting larger ones (**divisive**, top-down). The result is a **dendrogram**, a tree-like diagram representing cluster merges.

## 

## **Agglomerative Method**

## **Steps:**

1. Treat each data point as a separate cluster.
2. Compute distance between all clusters.
3. Merge the two closest clusters.
4. Recalculate distances using a **linkage method** (single, complete, average, Ward).
5. Repeat until all points merge into one root cluster.

**R Script**

1. **Install necessary packages**

# Install necessary packages

install.packages("e1071") # For Naive Bayes

install.packages("class") # For KNN

install.packages("rpart") # For Decision Trees

install.packages("rpart.plot") # For visualizing Decision Trees

# Load libraries

library(e1071)

library(class)

library(rpart)

library(rpart.plot)

1. **Load the Dataset**

# Load the iris dataset

data(iris)

# View the structure of the dataset

str(iris)

# Split the data into training and testing sets

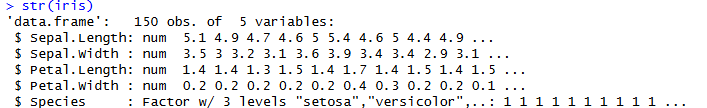
set.seed(123) # For reproducibility

index <- sample(1:nrow(iris), 0.7 \* nrow(iris)) # 70% training, 30% testing

train\_data <- iris[index, ]

test\_data <- iris[-index, ]

**Output**



1. **Naive Bayes classifier**

# Train the Naive Bayes classifier

nb\_model <- naiveBayes(Species ~ ., data = train\_data)

# Predict on test data

nb\_predictions <- predict(nb\_model, test\_data)

# Evaluate accuracy

nb\_accuracy <- mean(nb\_predictions == test\_data$Species)

print(paste("Naive Bayes Accuracy:", round(nb\_accuracy \* 100, 2), "%"))

# Define features and labels for training and testing

train\_features <- train\_data[, -5]

test\_features <- test\_data[, -5]

train\_labels <- train\_data$Species

test\_labels <- test\_data$Species

**Output**



1. **KNN Algorithm**

# Apply KNN with k = 5

knn\_predictions <- knn(train\_features, test\_features, train\_labels, k = 5)

# Evaluate accuracy

knn\_accuracy <- mean(knn\_predictions == test\_labels)

print(paste("KNN Accuracy:", round(knn\_accuracy \* 100, 2), "%"))

**Output**



1. **Decision Tree**

# Train a decision tree using rpart (C4.5)

dt\_model <- rpart(Species ~ ., data = train\_data, method = "class")

# Visualize the decision tree

rpart.plot(dt\_model, main = "Decision Tree (C4.5)")

# Predict on test data

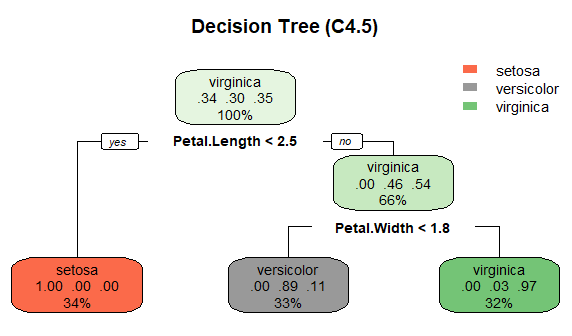
dt\_predictions <- predict(dt\_model, test\_data, type = "class")

# Evaluate accuracy

dt\_accuracy <- mean(dt\_predictions == test\_data$Species)

print(paste("Decision Tree Accuracy:", round(dt\_accuracy \* 100, 2), "%"))

**Output**



**Conclusion -** K-Means and Hierarchical Clustering were successfully implemented and compared. K-Means is fast and suitable for large datasets, while Hierarchical Clustering provides better visualization through a dendrogram but is slower. Both methods effectively grouped the data into meaningful clusters.

Aim - To implement various data visualization techniques using Tableau.

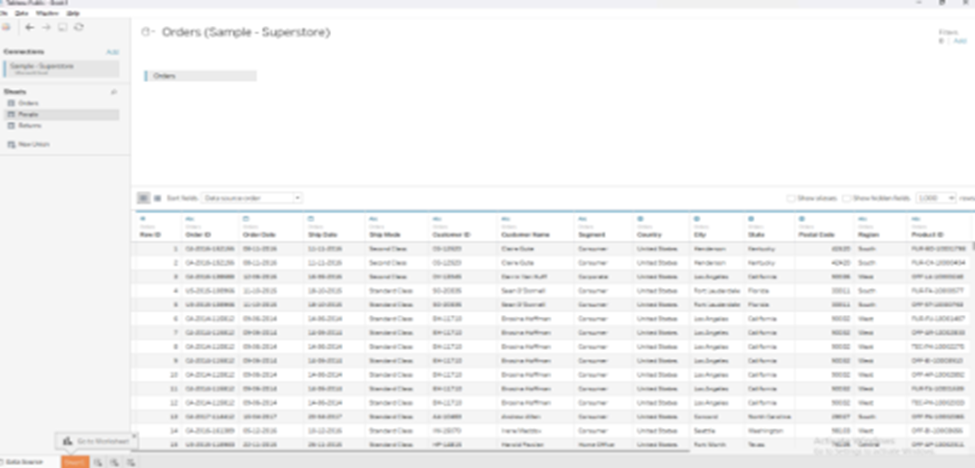
Title - Implement various data visualization techniques using Tableau.

**Description: Data Visualization Techniques**

**Step 1:** Download the DataSet or create your own.



**Step 2:** Select MSExcel to import data Imported Dataset appears as follows:



**1. Bar Chart**

Bar charts are best for comparing categories or discrete data.

1. Open Tableau and connect to your data source.

2. Drag a dimension (e.g., Product Category) to the Columns shelf.

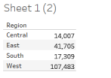
3. Drag a measure (e.g., Sales) to the Rows shelf.

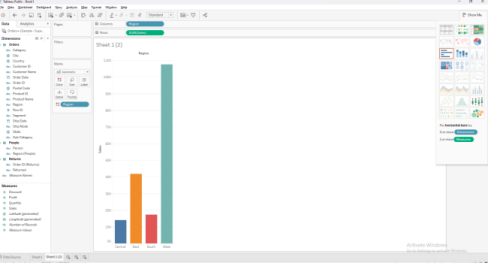
4. Tableau will automatically generate a bar chart. You can adjust the chart type using the Show Me panel if needed.

5. Customize your chart:

○ Use Color to differentiate categories.

○ Add labels by dragging the measure to the Label mark on the Marks card. ○ Adjust the axis and filters to focus on specific data.





**2. Line Chart**

Line charts are ideal for showing trends over time or continuous data.

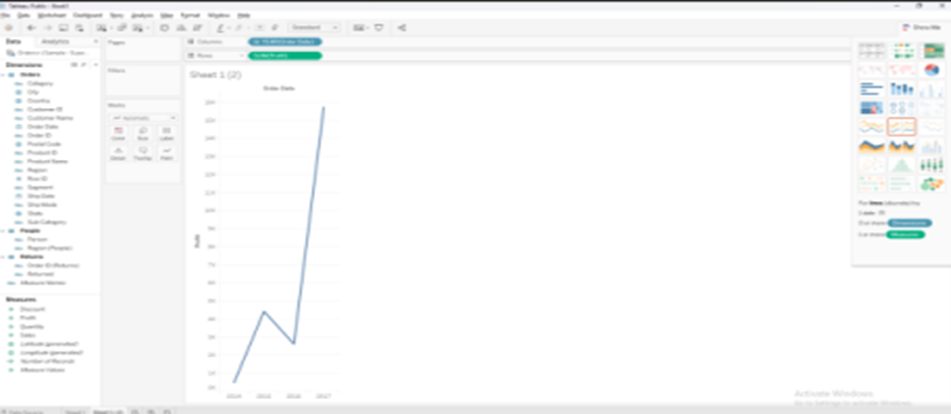
1. Open Tableau and connect to your data source.

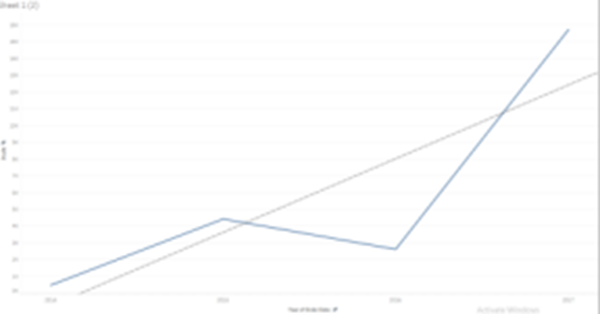
2. Drag a date field (e.g., Order Date) to the Columns shelf.

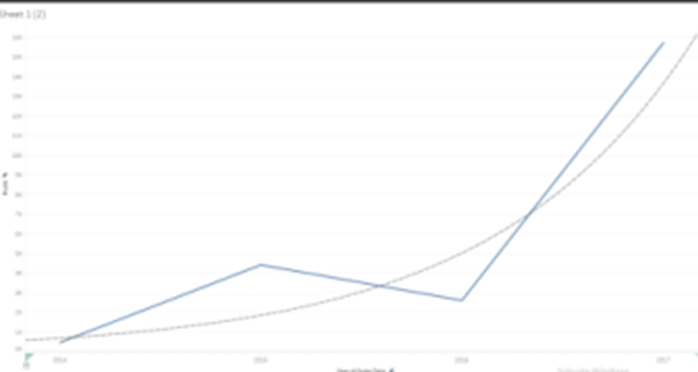
3. Drag a measure (e.g., Revenue) to the Rows shelf.

4. Tableau will create a line chart by default. If it doesn't, choose the Line chart from the Show Me panel.

5. Customize your chart: Drag a dimension (e.g., Region) to the Color mark to differentiate the lines by segment. Add trend lines by going to Analytics > Trend Line. Adjust the axis for better visualization.







**3. Pie Chart**

**Pie charts are used to show proportions of a whole.**

1. Open Tableau and connect to your data source.

2. Drag a dimension (e.g., Region) to the Columns shelf.

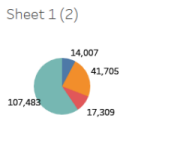
3. Drag a measure (e.g., Sales) to the Rows shelf.

4. Change the chart type to Pie by selecting Pie from the Show Me panel.

5. Customize the pie chart:

○ Drag the dimension to the Color or Label mark to differentiate segments. ○ Adjust labels to show percentage or value.

○ Resize the pie chart for better visibility.



**4. Scatter Plot**

**Scatter plots are used to visualize the relationship between two measures.**

1. Open Tableau and connect to your data source.

2. Drag one measure (e.g., Profit) to the Columns shelf.

3. Drag another measure (e.g., Sales) to the Rows shelf.

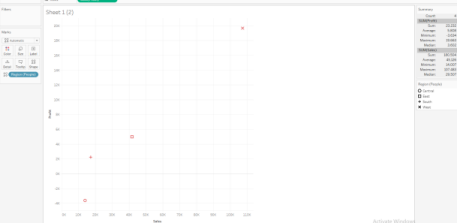
4. Tableau will create a scatter plot.

5. Customize your scatter plot:

○ Drag a dimension (e.g., Region) to the Color or Detail mark to color code the data points.

○ Add trend lines by going to Analytics > Trend Line.

○ Adjust the axis to scale the data appropriately.



**5. Histogram**

**Histograms help visualize the distribution of a single measure.**

1. Open Tableau and connect to your data source.

2. Drag a measure (e.g., Sales) to the Columns shelf.

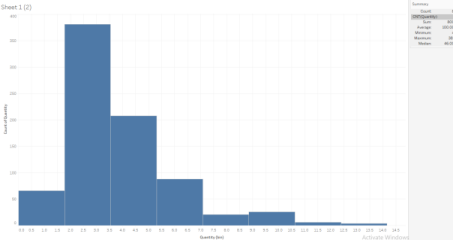
3. Right-click on the measure in the Columns shelf and select Histogram.

4. Tableau will create a histogram, displaying bins of data.

5. Customize your histogram:

○ Adjust the bin size by right-clicking on the bin field and selecting Edit.

○ Use the Color or Label marks for additional detail.



**6. Heat Map**

**Heat maps are useful for identifying patterns and comparing values across dimensions.**

1. Open Tableau and connect to your data source.

2. Drag a dimension (e.g., Region) to the Columns shelf.

3. Drag another dimension (e.g., Product Category) to the Rows shelf.

4. Drag a measure (e.g., Sales) to the Color mark.

5. Adjust the color scale by clicking on the Color mark and selecting Edit Colors.



**7. Bubble Chart**

**Bubble charts are similar to scatter plots but with an additional dimension represented by the size of the bubbles.**

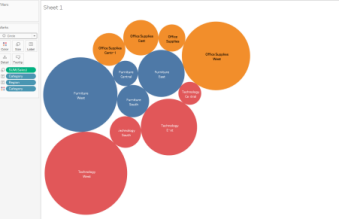
1. Open Tableau and connect to your data source.

2. Drag one measure (e.g., Profit) to the Columns shelf.

3. Drag another measure (e.g., Sales) to the Rows shelf.

4. Drag a dimension (e.g., Region) to the Color mark.

5. Drag a measure (e.g., Quantity) to the Size mark to represent bubble size. 6. Customize colors and size ranges for better visualization.



**8. Box Plot**

**A box plot, also known as a box-and-whisker plot, is used to show the distribution of a dataset and identify outliers. It displays the minimum, first quartile (Q1), median, third quartile (Q3), and maximum values.**

1. Connect to Data: Open Tableau and connect to your dataset.

2. Add Dimension to Columns: Drag a dimension (e.g., Region) to the Columns shelf.

3. Add Measure to Rows: Drag a measure (e.g., Sales) to the Rows shelf.

4. Enable Box Plot:

○ Go to the Analytics Pane and drag Box Plot to the view, or

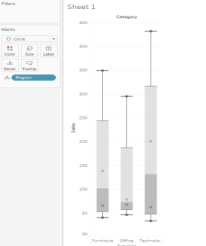
○ Change the Marks Type to "Box Plot" in the Marks card.

5. Customize Visualization:

○ Adjust whisker settings via the box plot settings.

○ Add dimensions (e.g., Category) to Color for segmentation.

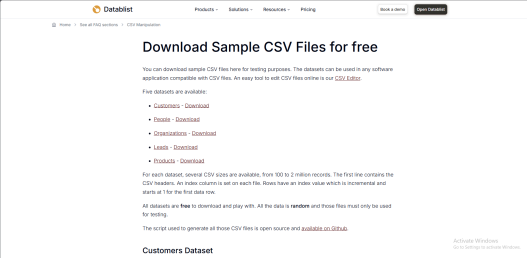
6. Finalize: Enhance tooltips, labels, and formatting for clarity.



**Conclusion:** Box plots are powerful tools for visualizing data distribution, identifying variability, and spotting outliers. Using Tableau's intuitive interface, creating and customizing box plots helps uncover key insights and makes data-driven decision-making more effective.

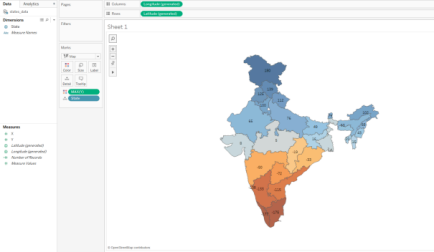
**Aim -**  Implementation to identify patterns and trends of geographic data using maps and spatial visualizations.

**Step 1: Download the DataSet or create your own.**

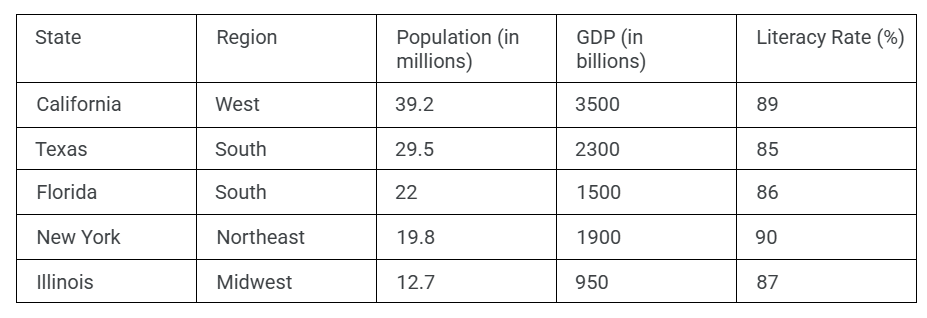
****

**Step 2: Right click on “State, X, Y” -> Select Geographic Role -> State/Province**

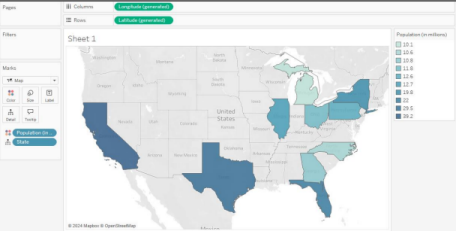
**Step 3: Select a 0 or more dimensions and 1 or more measures**



DataSet



Pennsylvania Northeast 12.6 800 89 Ohio Midwest 11.8 750 88 Georgia South 10.8 700 84 North Carolina South 10.6 650 85 Michigan Midwest 10.1 600 87



**Conclusion:** By creating and analyzing datasets for states, we can uncover patterns and trends across various metrics such as population, GDP, and literacy rate. Using tools like Tableau, these insights help in understanding regional disparities, resource allocation, and strategic planning for economic and social development.

**Step 1: Connect to Data**

**1. Open Tableau:**

○ Open Tableau Desktop.

**2. Connect to Data:**

○ Click “Connect” (left side of the screen).

○ Choose the data source. For this example, choose Excel if your data is in an

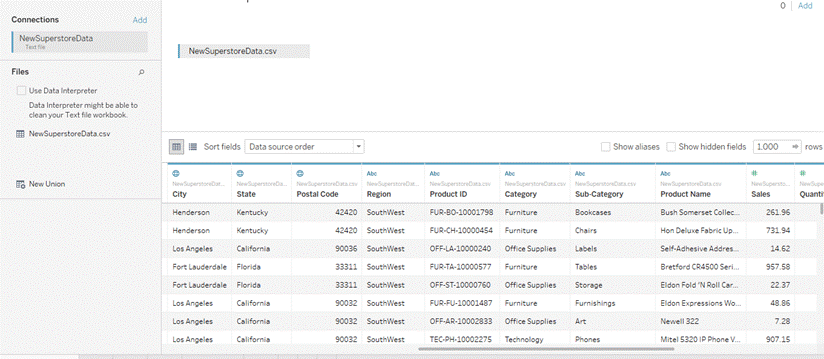
Excel file.

○ Locate your file and click Open.

**3. Preview the Data:**

○ Tableau will show a preview of your data. Click Sheet 1 at the bottom to start

working with your data.



**Step 2: Prepare Data**

### **1. Check the Data**

○ In the Data Pane, ensure that each field has the correct data type:

● Order Date & Ship Date → **Date**

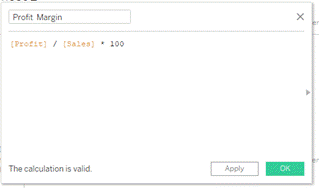
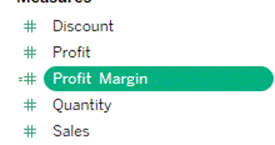
● Sales, Profit, Discount, Quantity → **Number**

● Category, Region, Customer Name → **Text**

### **2. Create Calculated Field (Optional)**

#### **A. Profit Margin**

○ Right-click the Data Pane → Create Calculated Field.  
 ○ Name: **Profit Margin**  ○ Formula:



**Step 3: Build Your First Visualization**

**A. Total Sales by Region**

### **1. Create a New Worksheet**

○ Click **New Worksheet**.

### **2. Drag Fields**

○ Drag **Region** to the **Rows** shelf.  
 ○ Drag **Sales** to the **Columns** shelf.

### **3. Change to Bar Chart**

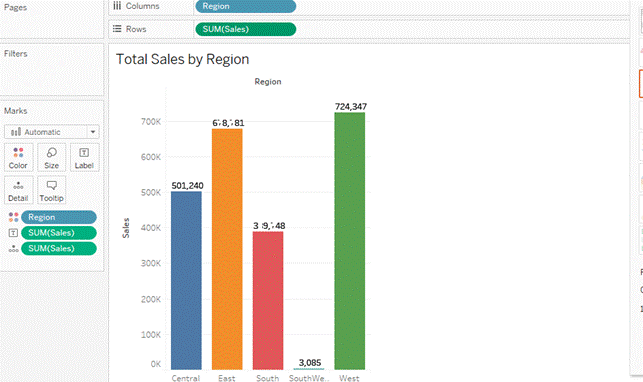
○ Tableau will automatically generate a bar chart.  
 ○ If not, open **Show Me** → select **Bar Chart**.

### **4. Format the Chart**

○ Right-click the Sales axis → **Format** → set **Currency** format.

### **5. Add Labels**

○ Drag **Sales** to the **Label** shelf on the Marks Card.



## **B. Sales Trend by Month**

### **1. Create a New Worksheet**

○ Click **New Worksheet**.

### **2. Drag Fields**

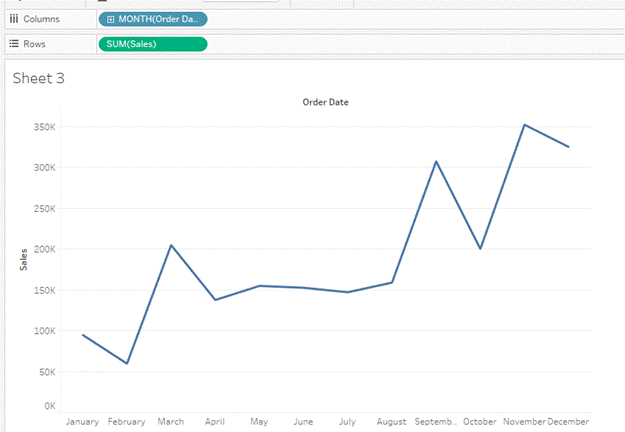
○ Drag **Order Date** to **Columns** (Tableau automatically treats it as a date).  
 ○ Click the date pill → choose **Month**.  
 ○ Drag **Sales** to **Rows**.

### **3. Change to Line Chart**

○ If Tableau doesn’t auto-select a line chart, open **Show Me → Line Chart**.

### **4. Format Axis**

○ Right-click Order Date axis → set it to **Month** if needed.



## **C. Profit by Product Category**

### **1. Create a New Worksheet**

○ Click on **New Worksheet**.

### **2. Drag Fields**

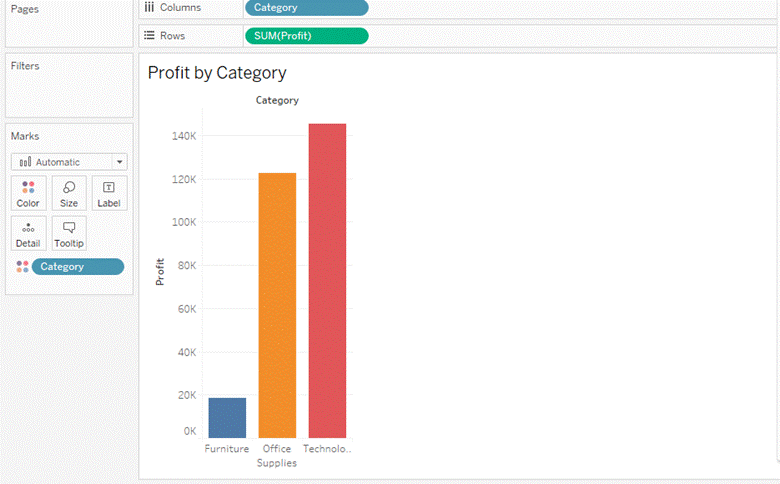
○ Drag **Category** to the **Rows** shelf.  
 ○ Drag **Profit** to the **Columns** shelf.

### **3. Change to Bar Chart**

○ From Show Me, select **Bar Chart**.

### **4. Format**

○ Right-click Profit axis → Format → Currency.



# **Step 4: Build a Dashboard**

### **1. Create a New Dashboard**

○ Click **New Dashboard** at the bottom.

### **2. Add Worksheets**

○ Drag your three worksheets into the dashboard:

● Total Sales by Region

● Sales Trend by Month

● Profit by Category

### **3. Add Interactivity (Optional)**

#### **Add Filters:**

#### **○ Drag Region or Category to the Filters shelf.**

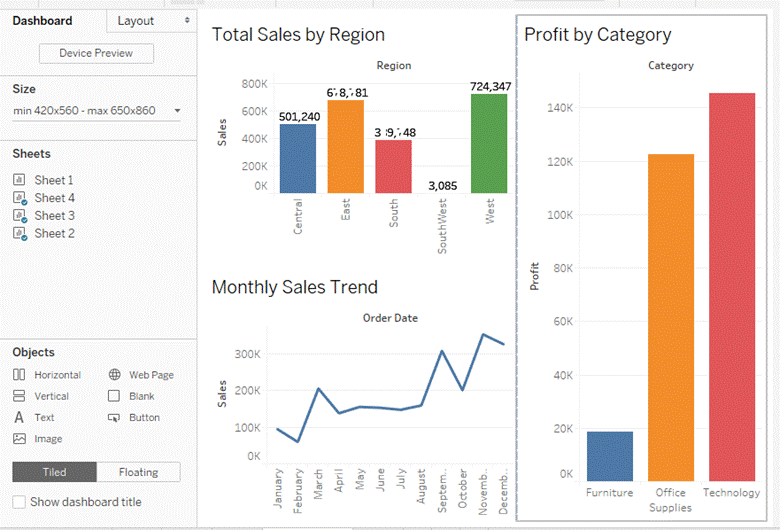
#### **○ Apply filters to the dashboard.**

#### **Add Filter Actions:**

○ Go to **Dashboard → Actions → Add Action → Filter**.  
 ○ Example: clicking a Region bar filters the monthly trend and category profit chart.

### **4. Format Dashboard**

○ Adjust chart size to fit neatly.  
 ○ Add titles or text boxes such as:  
 **“Sales & Profit Analysis Dashboard”**.



**Conclusion**

In this practical, we implemented and analyzed reports using Tableau. We connected to a dataset containing customer, product, and sales information. We prepared the data by verifying data types and creating calculated fields. We built multiple visualizations, including **Sales by Region**, **Monthly Sales Trend**, and **Profit by Category**.These were combined into an interactive dashboard that allows filtering and dynamic exploration of the dataset.Through this process, we learned how Tableau helps analyze business performance visually, making it easier to identify key patterns and support decision-making.

**Aim: To implement various advanced functions in Tableau.**

**Advanced Functions in Tableau:**

**1. Using Table Calculations**

Table Calculations in Tableau allow you to perform calculations on the data that has

already been aggregated. These are used for running totals, percent of total, ranking, and

more.

1. Create a Basic Visualization: Start by dragging a dimension (e.g., "Region") to the

Rows shelf and a measure (e.g., "Sales") to the Columns shelf.

2. Apply a Table Calculation:

○ Right-click on the measure on the Columns shelf (e.g., "Sales").

○ Choose Quick Table Calculation.

○ Select a calculation like Running Total, Percent of Total, or Rank.

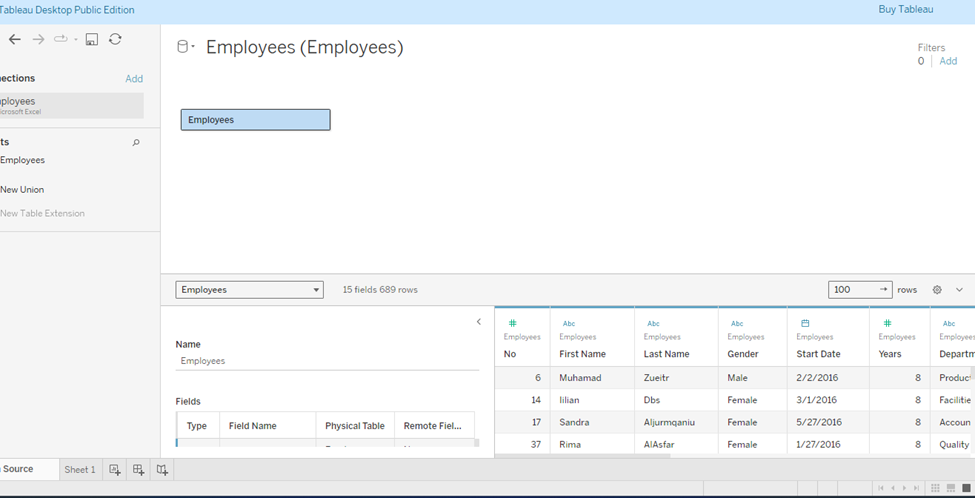
3. Edit Table Calculation:

○ Right-click on the measure again and select Edit Table Calculation to

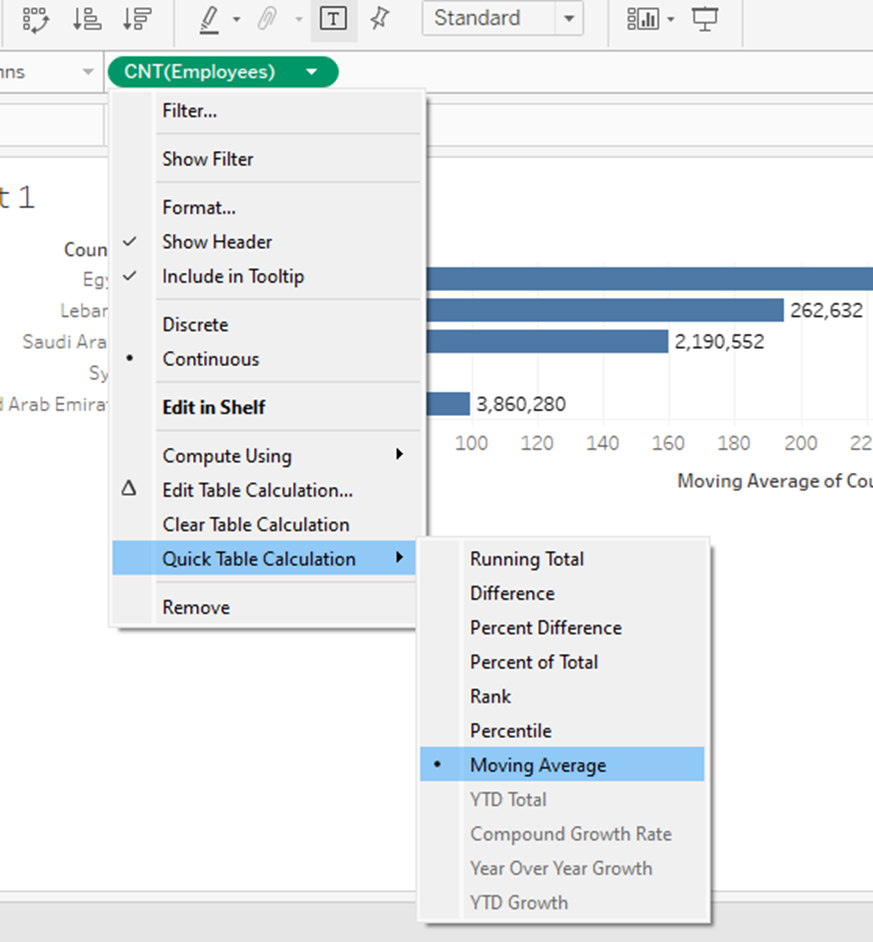
customize the calculation further.

○ Choose whether the calculation should be applied to the entire table, by pane,

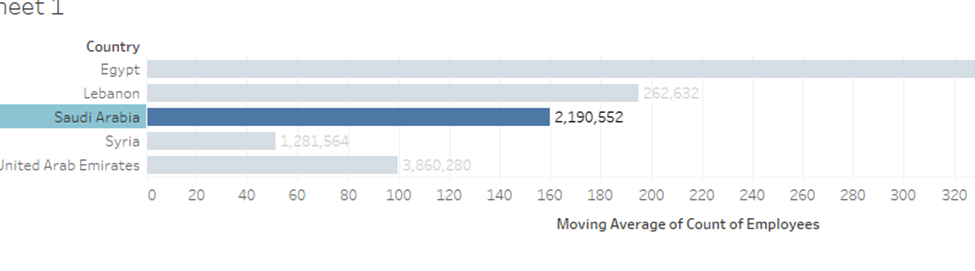
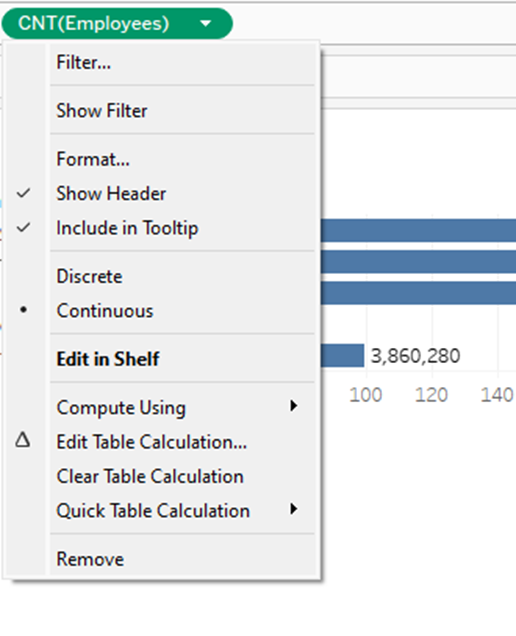
or by cell.



Select the Quick Table Calculation -> Moving Average



Select Edit Table Calculation to perform more operations:



**2. Using Level of Detail (LOD) Expressions**

Level of Detail (LOD) expressions allow you to control the granularity of your calculation.

Tableau supports three types of LOD expressions: Fixed, Include, and Exclude.

Steps for FIXED LOD Expression:

1. Create a Basic Visualization: Drag a dimension (e.g., "Category") to the Rows shelf

and a measure (e.g., "Sales") to the Columns shelf.

2. Create a New Calculated Field:

○ Go to Analysis → Create Calculated Field.

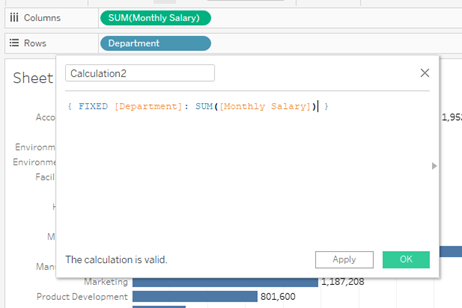
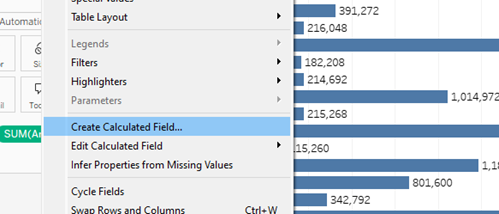
○ Enter the LOD expression, such as: { FIXED [Department]: SUM([Monthly Salary]) }

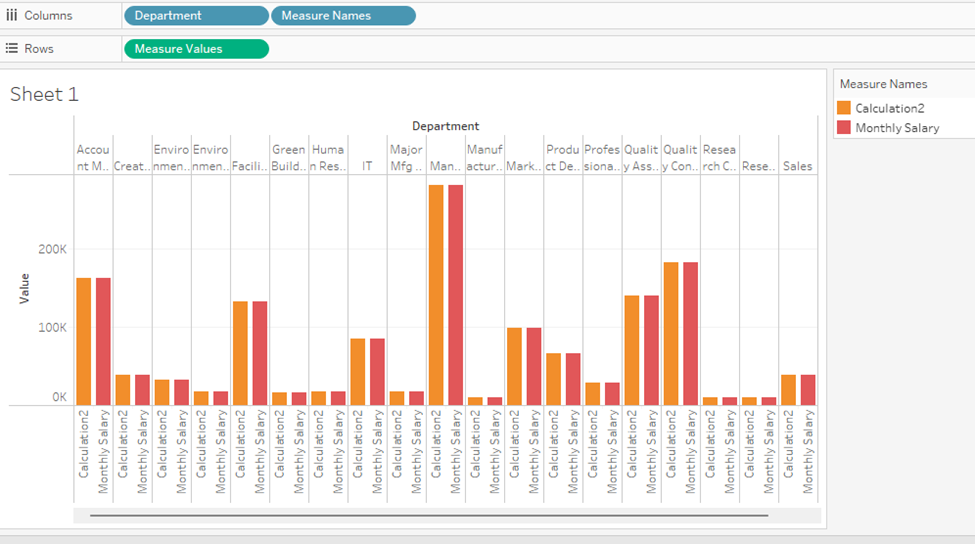
○ This will calculate the sum of sales at the region level regardless of any other

filters.

3. Use the Calculated Field:

Drag the calculated field onto the view (e.g., onto the Columns shelf or Tooltip) to see the results.





Steps for INCLUDE LOD Expression:

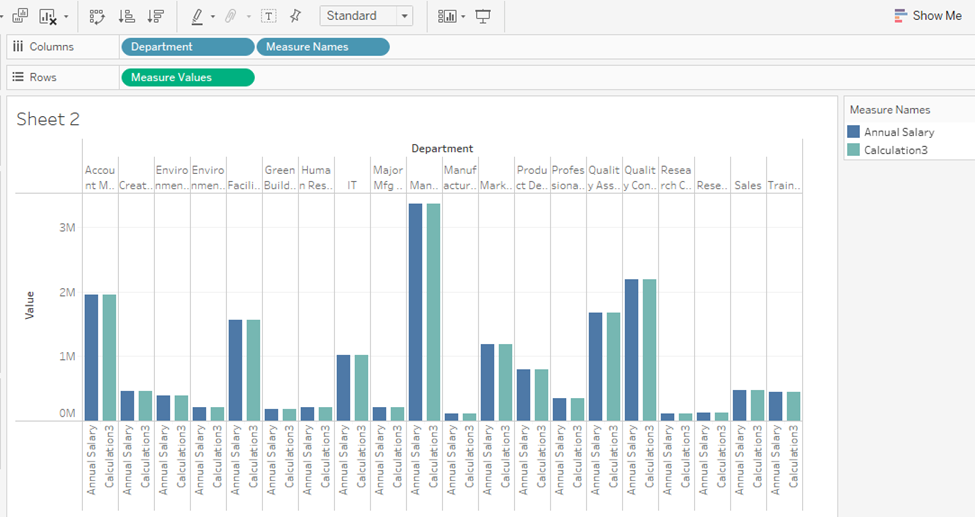
1. Create a New Calculated Field:

In the calculated field editor, enter the LOD expression:{ INCLUDE [Product]:

SUM([Sales]) }

○ This calculation includes the "Product" level of detail in the aggregation of

sales.



**3. Using Window Functions**

Window functions in Tableau perform calculations across a specified range of data in a

window, such as average or sum across a set of rows.

Steps for WINDOW\_SUM:

1. Create a Basic Visualization: Drag a dimension (e.g., "Region") to the Rows shelf and

a measure (e.g., "Sales") to the Columns shelf.

2. Add a Window Function:

○ Add a Table Calculation

○ Right-click on the measure pill (Sales) on the Columns shelf.

○ Hover over Quick Table Calculation.

○ Choose Running Total (or any similar aggregation option available).

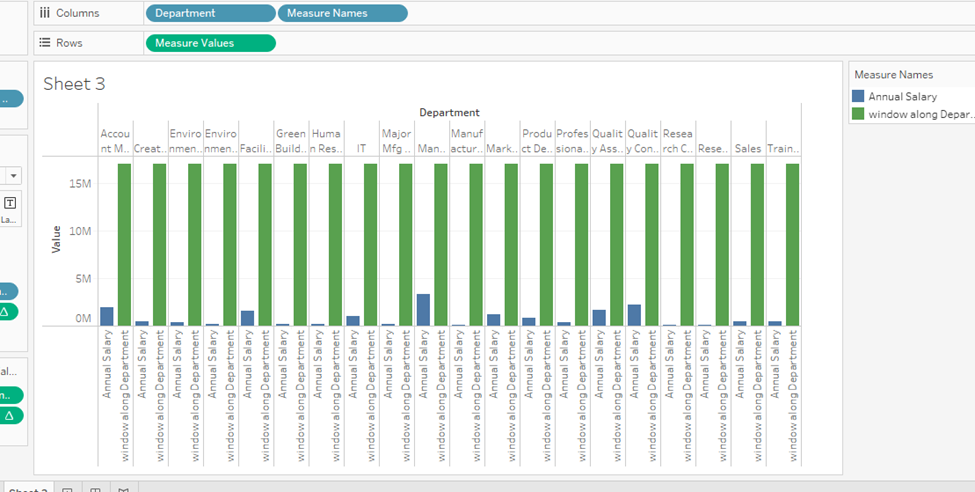
3. Edit Window Calculation:

○ Go to Analysis → Create Calculated Field.

○ Enter a formula like: WINDOW\_SUM(SUM([Sales]))

○ Drag the newly created calculated field to the Columns shelf or into the

visualization.



**4. Using Date Functions**

Tableau provides advanced date functions to manipulate date values, such as DATEPART,

DATETRUNC, and DATEADD.

Steps for DATEPART:

1. Create a Basic Visualization: Drag a measure (e.g., "Sales") to the Columns shelf.

2. Create a Calculated Field:

○ Go to Analysis → Create Calculated Field.

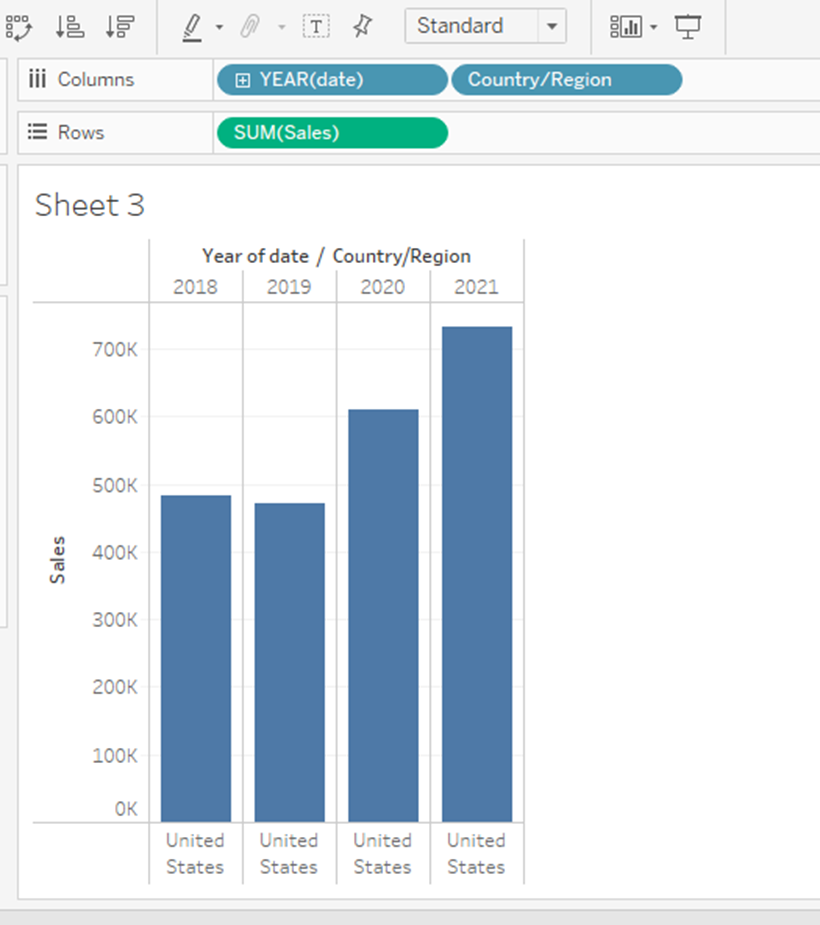
Enter the formula to extract a specific part of a date, such as:

DATEPART('month', [Order Date])

○ This will extract the month from the "Order Date" field.

3. Use the Calculated Field: Drag the calculated field to the Rows shelf or Tooltip to

visualize the results.



**5. Using IF Statements**

Conditional logic functions like IF and CASE allow you to create dynamic calculations based

on conditions.

Step-by-Step for IF Statements:

1. Create a New Calculated Field:

○ Go to Analysis → Create Calculated Field.

Enter the IF statement logic. For example:

IF [Sales] > 1000 THEN "High"

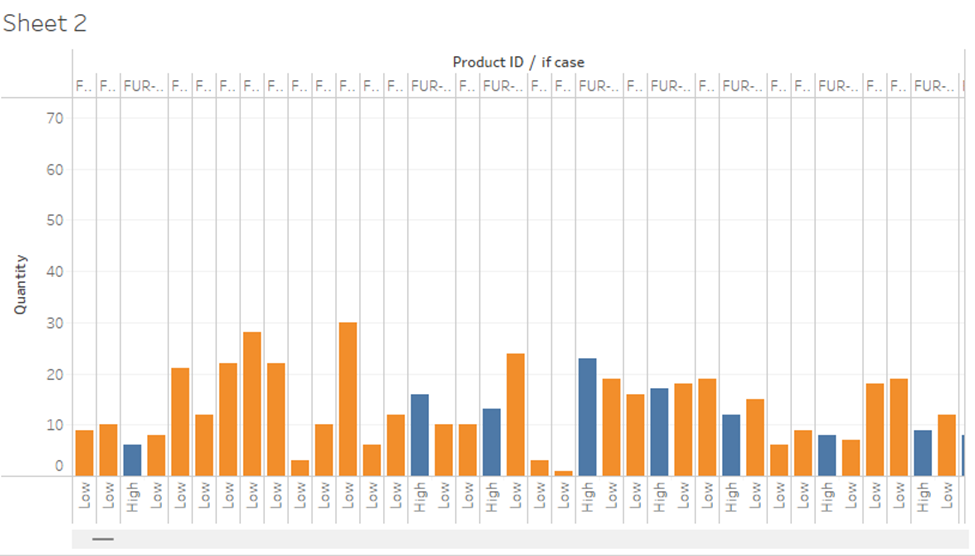
ELSE "Low"

END

○ This categorizes sales into "High" and "Low" based on the value.

2. Use the Calculated Field: Drag the calculated field to the Rows or Columns shelf to

see the segmentation.



**Conclusion:**

Tableau provides various advanced functions for data manipulation, analysis, and

visualization. By using Table Calculations, LOD Expressions, Window Functions, Date

Functions and Conditional Statements users can create sophisticated dashboards that

deliver deeper insights and more advanced analytics.