1. Check if a number is divisible by both 2 and 3 and Calculate Sum, Average, Minimum, and Maximum of First 10 Natural Numbers

# Check if a number is divisible by both 2 and 3

number <- as.numeric(readline(prompt = "Enter a number to check: "))

if (number %% 2 == 0 && number %% 3 == 0) {

cat(number, "is divisible by both 2 and 3.")

} else {

cat(number, "is not divisible by both 2 and 3.")

}

numbers <- 1:10

sum\_result <- 0

min\_result <- Inf

max\_result <- -Inf

for (i in numbers) {

sum\_result <- sum\_result + i

if (i < min\_result) {

min\_result <- i

}

if (i > max\_result) {

max\_result <- i

}

}

# Calculate average

average\_result <- sum\_result / 10

# Print results

cat("Sum:", sum\_result, "\n")

cat("Average:", average\_result, "\n")

cat("Minimum:", min\_result, "\n")

cat("Maximum:", max\_result, "\n")

2. Combine Multiple Logical Conditions and Calculate Sum, Average, Minimum, and Maximum of the First 10 Natural Numbers using a ForLoop

# Function to filter elements from a list based on multiple logical conditions

filter\_elements <- function(input\_list) {

filtered\_list <- input\_list[input\_list %% 2 == 0 & input\_list %% 3 == 0]

return(filtered\_list)

}

# Take input from the user for the list of numbers

user\_input <- as.numeric(strsplit(readline(prompt = "Enter a list of numbers separated by spaces: "), " ")[[1]])

cat(user\_input)

# Filter elements from the list using logical conditions

filtered\_numbers <- filter\_elements(user\_input)

# Print the filtered elements

cat("Filtered Elements:", filtered\_numbers, "\n")

# Create a for loop to calculate the sum, average, minimum, and maximum of the provided list

sum\_result <- 0

min\_result <- Inf

max\_result <- -Inf

for (i in user\_input) {

sum\_result <- sum\_result + i

if (i < min\_result) {

min\_result <- i

}

if (i > max\_result) {

max\_result <- i

}

}

# Calculate average

average\_result <- sum\_result / length(user\_input)

# Print the results

cat("Sum:", sum\_result, "\n")

cat("Average:", average\_result, "\n")

cat("Minimum:", min\_result, "\n")

cat("Maximum:", max\_result, "\n")

3. Create a Dataframe, Filter Rows, and Compute Squared Values using For Loop

# Create a dataframe

data <- data.frame(

Name = c("Srushti", "Madhuja", "Madhvi", "Manasi", "Priya"),

Age = c(20, 19, 18, 19, 17),

Score = c(85, 92, 78, 89, 95)

)

# Print the original dataframe

cat("Original Dataframe:\n")

print(data)

# Filter rows based on a specific condition (e.g., age greater than 19)

filtered\_data <- data[data$Age > 19, ]

# Print the filtered dataframe

cat("\nFiltered Dataframe (Age > 19):\n")

print(filtered\_data)

# Use a for loop to create a new vector with squared values

numeric\_vector <- 1:5

squared\_vector <- numeric\_vector # Initialize a new vector

for (i in numeric\_vector) {

squared\_vector <- numeric\_vector^2

}

# Print the original numeric vector

cat("\nOriginal Numeric Vector:\n")

print(numeric\_vector)

# Print the new vector with squared values

cat("\nSquared Vector:\n")

print(squared\_vector)

4. multidimensional array and demonstrates how to perform array slicing to extract a specific subarray. Also, use a for loop to iterate over a numeric vector and create a new vector Addition of two vectors.

# Create a multidimensional array

# In this example, a 3D array is created

array\_dim <- c(3, 4, 1) # Dimensions: 3x4x2

multidimensional\_array <- array(1:24, dim = array\_dim)

# Print the original array

cat("Original Multidimensional Array:\n")

print(multidimensional\_array)

# Perform array slicing to extract a specific subarray

subarray <- multidimensional\_array[2, , 1] # Extracts the second row of the first layer

cat("\nExtracted Subarray:\n")

print(subarray)

# Use a for loop to iterate over a numeric vector and create a new vector representing the addition of two vectors

vector1 <- c(1, 2, 3, 4, 5)

vector2 <- c(6, 7, 8, 9, 10)

result\_vector <- numeric(length(vector1))

# Iterate over the vectors and perform addition

for (i in (vector1)) {

result\_vector[i] <- vector1[i] + vector2[i]

}

# Print the result vector

cat("\nResult Vector (Addition of two vectors):\n")

print(result\_vector)

5. creates a list (or array) of integers and performs the following operations: append, insert, delete, display min, max, sum and average and find any specific element.

# Function to display the list

display\_list <- function(my\_list) {

cat("List:", my\_list, "\n")

}

# Function to append an element to the list

append\_element <- function(my\_list, element) {

my\_list <- c(my\_list, element)

return(my\_list)

}

# Function to insert an element at a specific position in the list

insert\_element <- function(my\_list, position, element) {

if (position > length(my\_list) + 1) {

cat("Position is out of range\n")

return(my\_list)

}

my\_list <- c(my\_list[1:(position - 1)], element, my\_list[position:length(my\_list)])

return(my\_list)

}

# Function to delete an element from the list

delete\_element <- function(my\_list, element) {

my\_list <- my\_list[my\_list != element]

return(my\_list)

}

# Function to find the minimum, maximum, sum, and average of the list

calculate\_stats <- function(my\_list) {

min\_val <- min(my\_list)

max\_val <- max(my\_list)

sum\_val <- sum(my\_list)

avg\_val <- mean(my\_list)

return(list(min = min\_val, max = max\_val, sum = sum\_val, avg = avg\_val))

}

# Function to find a specific element in the list

find\_element <- function(my\_list, element) {

index <- which(my\_list == element)

if (length(index) > 0) {

return(paste("Element", element, "found at index:", index))

} else {

return(paste("Element", element, "not found in the list"))

}

}

# Sample list

my\_list <- c(3, 7, 1, 5, 8)

# Display the initial list

display\_list(my\_list)

# Append an element

my\_list <- append\_element(my\_list, 10)

display\_list(my\_list)

# Insert an element at position 3

my\_list <- insert\_element(my\_list, 3, 4)

display\_list(my\_list)

# Delete the element 5

my\_list <- delete\_element(my\_list, 5)

display\_list(my\_list)

# Calculate statistics

stats <- calculate\_stats(my\_list)

cat("Min:", stats$min, "\n")

cat("Max:", stats$max, "\n")

cat("Sum:", stats$sum, "\n")

cat("Average:", stats$avg, "\n")

# Find an element (e.g., 7)

result <- find\_element(my\_list, 7)

cat(result, "\n")

#from the user

# Function to display the list

display\_list <- function(my\_list) {

cat("List:", my\_list, "\n")

}

# Function to append an element to the list

append\_element <- function(my\_list, element) {

my\_list <- c(my\_list, element)

return(my\_list)

}

# Function to insert an element at a specific position in the list

insert\_element <- function(my\_list, position, element) {

if (position > length(my\_list) + 1) {

cat("Position is out of range\n")

return(my\_list)

}

my\_list <- c(my\_list[1:(position - 1)], element, my\_list[position:length(my\_list)])

return(my\_list)

}

# Function to delete an element from the list

delete\_element <- function(my\_list, element) {

my\_list <- my\_list[my\_list != element]

return(my\_list)

}

# Function to find the minimum, maximum, sum, and average of the list

calculate\_stats <- function(my\_list) {

min\_val <- min(my\_list)

max\_val <- max(my\_list)

sum\_val <- sum(my\_list)

avg\_val <- mean(my\_list)

return(list(min = min\_val, max = max\_val, sum = sum\_val, avg = avg\_val))

}

# Function to find a specific element in the list

find\_element <- function(my\_list, element) {

index <- which(my\_list == element)

if (length(index) > 0) {

return(paste("Element", element, "found at index:", index))

} else {

return(paste("Element", element, "not found in the list"))

}

}

# Take input from the user for the list of numbers

user\_input <- as.numeric(strsplit(readline(prompt = "Enter a list of numbers separated by spaces: "), " ")[[1]])

cat(user\_input)

# Display the initial list

display\_list(user\_input)

# Append an element

append\_value <- as.numeric(readline(prompt = "Enter an element to append: "))

user\_input <- append\_element(user\_input, append\_value)

display\_list(user\_input)

# Insert an element at a specific position

insert\_value <- as.numeric(readline(prompt = "Enter an element to insert: "))

insert\_position <- as.numeric(readline(prompt = "Enter the position to insert: "))

user\_input <- insert\_element(user\_input, insert\_position, insert\_value)

display\_list(user\_input)

# Delete an element

delete\_value <- as.numeric(readline(prompt = "Enter an element to delete: "))

user\_input <- delete\_element(user\_input, delete\_value)

display\_list(user\_input)

# Calculate statistics

stats <- calculate\_stats(user\_input)

cat("Min:", stats$min, "\n")

cat("Max:", stats$max, "\n")

cat("Sum:", stats$sum, "\n")

cat("Average:", stats$avg, "\n")

# Find an element

find\_value <- as.numeric(readline(prompt = "Enter an element to find: "))

result <- find\_element(user\_input, find\_value)

cat(result, "\n")

6. user-input number and uses an if/else statement to determine whether it is positive, negative, or zero. Take some numbers from users and store in x and y vectors to plot any kind of graph using it.

# Take user input for a number

user\_number <- as.numeric(readline(prompt = "Enter a number: "))

# Determine the sign using if/else statement

if (user\_number > 0) {

cat("The entered number is: Positive\n")

} else if (user\_number < 0) {

cat("The entered number is: Negative\n")

} else {

cat("The entered number is: Zero\n")

}

# Take user input for vectors x and y

x <- as.numeric(strsplit(readline(prompt = "Enter values for vector x separated by spaces: "), " ")[[1]])

y <- as.numeric(strsplit(readline(prompt = "Enter values for vector y separated by spaces: "), " ")[[1]])

x

y

# Check if the length of x and y is the same

if (length(x) != length(y)) {

cat("Error: Vectors x and y must have the same length.\n")

} else {

# Plot a line plot

plot(x, y, type = "l", main = "Line Plot", xlab = "Vector x", ylab = "Vector y", col = "red")

# Plot a scatter plot

plot(x, y, main = "Scatter Plot", xlab = "Vector x", ylab = "Vector y", col = "blue")

}

7. into a dataframe and display the first few rows, summary, information of dataset. Display the data from csv in line, scatter, histogram and dot plot.

# Read CSV file into a dataframe

data <- read.csv("D:/ASA/csv/bikes.csv")

# Display the first few rows of the dataframe

cat("First few rows of the dataframe:\n")

head(data)

# Display summary of the dataframe

cat("\nSummary of the dataframe:\n")

summary(data)

# Display information about the dataframe

cat("\nInformation about the dataframe:\n")

str(data)

# Plot data in line

cat("\nLine Plot:\n")

plot(data$bike.id, type = "l", main = "Line Plot", xlab = "X-axis Label", ylab = "Y-axis Label")

# Scatter Plot

cat("\nScatter Plot:\n")

plot(data$bike.id, data$price, main = "Scatter Plot", xlab = "X-axis Label", ylab = "Y-axis Label", pch= 7)

# Histogram

cat("\nHistogram:\n")

hist(data$bike.id, main = "Histogram", xlab = "Values", col = "green", border = "black")

# Dot Plot

cat("\nDot Plot:\n")

stripchart(data$price, method = "stack", main = "Dot Plot", xlab = "X-axis Label",col = "blue", pch = 2)

8. Implement a nested if/else statement to classify a given number as odd or even and, within each category, as positive, negative, or zero. Display the data from csv in line, scatter, histogram and dot plot.

# Take user input for a number

user\_number <- as.numeric(readline(prompt = "Enter a number: "))

# Nested if/else statement to classify the number

if (user\_number %% 2 == 0) {

# Even number

cat("The number is even.\n")

if (user\_number > 0) {

cat("The number is positive.\n")

} else if (user\_number < 0) {

cat("The number is negative.\n")

} else {

cat("The number is zero.\n")

}

} else {

# Odd number

cat("The number is odd.\n")

if (user\_number > 0) {

cat("The number is positive.\n")

} else if (user\_number < 0) {

cat("The number is negative.\n")

} else {

cat("The number is zero.\n")

}

}

# Read CSV file into a dataframe

data <- read.csv("D:/ASA/csv/bikes.csv")

# Display the first few rows of the dataframe

cat("First few rows of the dataframe:\n")

head(data)

# Display summary of the dataframe

cat("\nSummary of the dataframe:\n")

summary(data)

# Display information about the dataframe

cat("\nInformation about the dataframe:\n")

str(data)

# Plot data in line

cat("\nLine Plot:\n")

plot(data$bike.id, type = "l", main = "Line Plot", xlab = "X-axis Label", ylab = "Y-axis Label")

# Scatter Plot

cat("\nScatter Plot:\n")

plot(data$bike.id, data$price, main = "Scatter Plot", xlab = "X-axis Label", ylab = "Y-axis Label", pch= 7)

# Histogram

cat("\nHistogram:\n")

hist(data$bike.id, main = "Histogram", xlab = "Values", col = "green", border = "black")

# Dot Plot

cat("\nDot Plot:\n")

stripchart(data$price, method = "stack", main = "Dot Plot", xlab = "X-axis Label",col = "blue", pch = 2)

9. data from an Excel file ("data.xlsx") and print 5-point summary of the dataframe along with box-whisker plot and dot plot.

# Load required libraries

library(readxl)

# Read data from Excel file into a dataframe

data <- read\_excel("C:/Users/Srushti Ladkat/Downloads/xlsx.xlsx")

# Print 5-point summary

cat("5-Point Summary:\n")

summary(data)

# Box-Whisker Plot

boxplot(data$Age, main = "Box-Whisker Plot", xlab = "Variables", ylab = "Values")

# Dot Plot

stripchart(data$Id, method = "stack", main = "Dot Plot", xlab = "Variables", pch = 19)

10. calculates the average of a specific column, and uses an if/else statement to determine whether the average is above or below a certain threshold. Display the data from csv in line, scatter, histogram and dot plot.

# Read data from CSV file into a dataframe

data <- read.csv("D:/ASA/csv/bikes.csv")

data$bike.id

# Specify the column for which you want to calculate the average

target\_column <- "bike.id"

# Calculate the average of the specified column

average\_value <- mean(data[[target\_column]])

# Set the threshold for comparison

threshold <- 50 # Replace with your desired threshold value

# Use an if/else statement to determine whether the average is above or below the threshold

if (average\_value > threshold) {

cat("The average is above the threshold.\n")

} else {

cat("The average is below or equal to the threshold.\n")

}

average\_value

# Visualize the data: Line Plot

plot(data[[target\_column]], type = "l", main = "Line Plot", xlab = "Index", ylab = "Values")

# Visualize the data: Scatter Plot

plot(data[[target\_column]], main = "Scatter Plot", xlab = "Index", ylab = "Values")

# Visualize the data: Histogram

hist(data[[target\_column]], main = "Histogram", xlab = "Values", col = "blue", border = "black")

# Visualize the data: Dot Plot

stripchart(data[[target\_column]], method = "stack", main = "Dot Plot", xlab = "Index", pch = 19)

11. standardized (Z-) scores for several variables Using the preexisting Drinks.csv data file and Display the data from drinks.csv using line graph, scatter plot, histogram and dot plot.

# Read data from the "Drinks.csv" file into a dataframe

drinks\_data <- read.csv("D:/ASA/csv/Drinks.csv")

# Display the first few rows of the dataframe

cat("First few rows of the dataframe:\n")

head(drinks\_data)

# Standardize (Z-) scores for several variables

standardized\_data <- scale(drinks\_data[, c("beer\_servings", "spirit\_servings", "wine\_servings")])

# Display standardized data

cat("Standardized data:\n")

head(standardized\_data)

# Visualize the data: Line Graph

matplot(standardized\_data, type = "l", main = "Line Graph", xlab = "Index", ylab = "Z-Scores")

# Visualize the data: Scatter Plot

plot(standardized\_data, main = "Scatter Plot", xlab = "Index", ylab = "Z-Scores")

# Visualize the data: Histogram

hist(standardized\_data, main = "Histogram", xlab = "Z-Scores", col = "blue", border = "black")

# Visualize the data: Dot Plot

stripchart(standardized\_data, method = "stack", main = "Dot Plot", xlab = "Index", pch = 19)

12. Frequencies to explore the distributions of several variables Using the preexisting Census.csv data file and display the data from csv file in line graph, scatter plot, histogram and dot plot.

# Read data from the "Census.csv" file into a dataframe

census\_data <- read.csv("Census.csv")

# Display the first few rows of the dataframe

cat("First few rows of the dataframe:\n")

head(census\_data)

# Run Frequencies to explore the distributions

frequencies\_result <- summary(census\_data)

# Display frequencies result

cat("Frequencies Result:\n")

print(frequencies\_result)

# Visualize the data: Line Graph (replace variables with actual column names)

matplot(census\_data[, c("Var1", "Var2", "Var3")], type = "l", main = "Line Graph", xlab = "Index", ylab = "Values")

# Visualize the data: Scatter Plot (replace variables with actual column names)

plot(census\_data$Var1, census\_data$Var2, main = "Scatter Plot", xlab = "Var1", ylab = "Var2")

# Visualize the data: Histogram (replace variables with actual column names)

hist(census\_data$Var3, main = "Histogram", xlab = "Var3", col = "green", border = "black")

# Visualize the data: Dot Plot (replace variables with actual column names)

stripchart(census\_data$Var1, method = "stack", main = "Dot Plot", xlab = "Var1", pch = 19)

13. create two way cross tabulations to explore the relationship between several variables and to use the Chart Builder to visualize the relationship Using the preexisting Census.csv data file.

library(gmodels)

library(ggplot2)

# Read data from the "bikes.csv" file into a dataframe

census\_data <- read.csv("D:/ASA/csv/bikes.csv")

# Explore the relationship between variables with Cross-Tabulations

cross\_tab\_result <- CrossTable(census\_data$bike.id, census\_data$price, expected = TRUE, chisq = TRUE, prop.chisq = TRUE)

# Display the Cross-Tabulation result

print(cross\_tab\_result)

# Use the Chart Builder to visualize the relationship (replace variables with actual column names)

chart\_builder\_plot <- ggplot(census\_data, aes(x = as.factor(bike.id), fill = as.factor(price))) +

geom\_bar(position = "fill", stat = "count") +

labs(title = "Chart Builder Visualization", x = "Bike ID", y = "Count") +

theme\_minimal()

# Display the Chart Builder plot

print(chart\_builder\_plot)

14. visualize the relationship between two scale variables creating scatter plots and to quantify this relationship with the correlation coefficient using census.csv data file.

# Read Census.csv file into a dataframe

census\_data <- read.csv("D:/ASA/csv/bikes.csv")

# Visualize the relationship between two scale variables using scatter plots

# Assuming 'variable1' and 'variable2' are columns of interest

plot(census\_data$bike.id, census\_data$price)

# Quantify the relationship with the correlation coefficient

correlation <- cor(census\_data$bike.id, census\_data$price) print(correlation)

15. Independent-Samples T Test, to interpret the output and visualize the results with an error bar chart. Using the preexisting Census.csv data file.

# Assuming you have loaded the required libraries

library(ggplot2)

library(ggpubr)

# Read data from the "Census.csv" file into a dataframe

census\_data <- read.csv("D:/ASA/csv/Bangalore.csv")

# Perform Independent-Samples T Test

t\_test\_result <- t.test(Price(continuous) ~ Resale(binary), data = census\_data)

# Interpret the output

cat("Independent-Samples T Test Result:\n")

print(t\_test\_result)

# Visualize the results with an error bar chart

error\_bar\_chart <- ggplot(census\_data, aes(x = Resale, y = Price, fill = Resale)) +

geom\_bar(stat = "summary", fun = "mean", position = "dodge") +

geom\_errorbar(stat = "summary", fun.data = "mean\_se", position = "dodge", width = 0.2) +

labs(title = "Error Bar Chart", x = "Resale", y = "Price") +

theme\_minimal()

# Display the error bar chart

print(error\_bar\_chart)

16. One-Way ANOVA with post hoc tests to explore the relationship between several variables Using the preexisting data file Census.csv and represent it using any two suitable graphs.

# Assuming 'Census.csv' contains necessary columns for ANOVA

census\_data <- read.csv("D:/ASA/csv/ass7 Iris.csv")

# Perform One-Way ANOVA

library(car)

anova\_result <- aov(Sno(continuous) ~ Species(categorical), data = census\_data)

summary(anova\_result)

# Post hoc tests (e.g., Tukey's HSD)

posthoc <- TukeyHSD(anova\_result)

print(posthoc)

# Visual representation using suitable graphs

# For instance, boxplot and bar plot

# Boxplot

boxplot(Sno ~ Species, data = census\_data, main = "Boxplot - One-Way ANOVA", xlab = "Species", ylab = "Sno")

# Bar plot

barplot(table(census\_data$Species), main = "Bar Plot - Species Count", xlab = "Species", ylab = "Count", col = "skyblue")

17. dataframe and filters rows based on a specific condition using subsetting. use a for loop to iterate over two numeric vector and create a new vector containing the sum values of each element, and represent it using any two suitable graphs.

# Create a dataframe and add a condition

categorical\_variable <- c("A", "B", "A", "B", "A")

numeric\_vector1 <- c(1, 2, 3, 4, 5)

numeric\_vector2 <- c(6, 7, 8, 9, 10)

# Combine into a dataframe

census\_data <- data.frame(

categorical\_variable,

numeric\_vector1,

numeric\_vector2

)

# Filter rows based on a specific condition

filtered\_data <- subset(census\_data, categorical\_variable == "A")

# Initialize an empty vector to store sum values

sum\_vector <- numeric(length(numeric\_vector1))

# Iterate over the vectors and calculate the sum

for (i in seq\_along(numeric\_vector1)) {

sum\_vector[i] <- numeric\_vector1[i] + numeric\_vector2[i]

}

sum\_vector

# Line plot

plot(sum\_vector, type = "l", xlab = "Index", ylab = "Sum", main = "Line Plot")

# Histogram

hist(sum\_vector, xlab = "Sum", main = "Histogram", col = "skyblue")

18. a multidimensional array and demonstrates how to perform array slicing to extract a specific subarray. Also, use line or scatter plot to represent the matrix values visually, and represent it using any two suitable graphs.

# Create a multidimensional array

multidim\_array <- array(1:9, dim = c(3, 3, 1))

multidim\_array

# Perform array slicing to extract a specific subarray

subarray <- multidim\_array[,3,]

subarray

# Visual representation of matrix values using line or scatter plot

plot(subarray, type = "l", xlab = "Index", ylab = "Value", main = "Line Plot")

plot(subarray, xlab = "Column", ylab = "Value", main = "Scatter Plot")

19. Make stacked dotplots of the same variable from csv file provided to you based on the values of one of your categorical variables. For example, if your quantitative variable is GPAs of students, your categorical variable could be gender. Comment on the similarities and differences between the distributions for the different values of your categorical variable.

# Load necessary libraries

library(ggplot2)

# Read data from CSV file

data <- read.csv("D:/ASA/csv/Bangalore.csv")

# Create stacked dotplots

ggplot(data, aes(x = quantitative\_variable, y =Price, color quantitative\_variable)) +

geom\_dotplot(binaxis = 'y', stackdir = 'center', dotsize = 0.7) +

labs(title = "Stacked Dotplots", x = "Category", y = "Variable") +

theme\_minimal()

20. Calculate summary measures (mean, standard deviation, first quartile, third quartile and interquartile range) for the variable you graphed above. Do this for the entire data set, as well as for the different groups formed by the categorical variable that you used to divide the data set.

# Load necessary libraries

library(dplyr)

# Read data from CSV file

data <- read.csv("path/to/your/data.csv")

# Summary measures for the entire dataset

summary\_entire <- summary(data$Variable)

# Summary measures for each group

summary\_by\_group <- data %>%

group\_by(Category) %>%

summarise(

Mean = mean(Variable),

SD = sd(Variable),

Q1 = quantile(Variable, 0.25),

Q3 = quantile(Variable, 0.75),

IQR = IQR(Variable)

)

# Display the results

cat("Summary Measures for the Entire Dataset:\n")

print(summary\_entire)

cat("\nSummary Measures by Group:\n")

print(summary\_by\_group)

21. Create a histogram and a dotplot of the data. Comment on any symmetry or skewness and on the presence of clusters and any potential outliers. And draw a box-and-whisker plot for the entire data set.

census\_data <- read.csv("D:/ASA/csv/ass9 color anova.csv")

# Create histogram and dotplot of the data

hist(census\_data$variable, xlab = "variable", main = "Histogram")

dotchart(census\_data$variable, labels = rownames(census\_data), cex = 0.7)

# Create box-and-whisker plot for the entire dataset

boxplot(census\_data$variable, xlab = "variable", main = "Boxplot")

22. Prepare an appropriate type of frequency distribution table for one of the quantitative variables and then compute relative frequencies and cumulative relative frequencies using census.csv dataset.

# Load necessary library

library(dplyr)

# Read the dataset

census\_data <- read.csv("D:/ASA/csv/Bangalore.csv")

# Choose a quantitative variable (replace 'variable' with the actual variable name)

variable <- census\_data$variable

# Create a frequency table

freq\_table <- table(variable)

# Compute relative frequencies

rel\_freq <- prop.table(freq\_table)

# Compute cumulative relative frequencies

cum\_rel\_freq <- cumsum(rel\_freq)

# Display the results

data.frame(Frequency = as.vector(freq\_table),

Relative\_Frequency = rel\_freq, Cumulative\_Relative\_Frequency = cum\_rel\_freq)

23. Prepare an appropriate type of frequency distribution table for one of the quantitative variables and then compute relative frequencies and cumulative relative frequencies using Drinks.csv data file.

# Load necessary libraries

library(dplyr)

# Read data from CSV file

drinks\_data <- read.csv("D:/ASA/csv/Drinks.csv")

# Choose a quantitative variable (replace "variable\_name" with the actual variable name)

variable\_name <- "beer\_servings"

# Create a frequency distribution table

freq\_table <- table(drinks\_data[[variable\_name]])

# Convert to data frame for better manipulation

freq\_df <- as.data.frame(freq\_table)

# Rename columns for clarity

colnames(freq\_df) <- c("Value", "Frequency")

# Compute relative frequencies

freq\_df$Relative\_Frequency <- freq\_df$Frequency / sum(freq\_df$Frequency)

# Compute cumulative relative frequencies

freq\_df$Cumulative\_Relative\_Frequency <- cumsum(freq\_df$Relative\_Frequency)

# Print the frequency distribution table

print(freq\_df)

24. to visualize the relationship between two scale variables creating scatter plots and to quantify this relationship with the correlation coefficient using Drinks.csv data file.

drinks\_data <- read.csv("D:/ASA/csv/Drinks.csv")

# Choose two scale variables (replace 'Variable1' and 'Variable2' with actual variable names)

variable1 <- drinks\_data$beer\_servings

variable2 <- drinks\_data$spirit\_servings

# Create a scatter plot

plot(variable1, variable2, main = "Scatter Plot", xlab = "Variable1", ylab = "Variable2")

# Calculate correlation coefficient

correlation\_coefficient <- cor(variable1, variable2)

# Print the correlation coefficient

cat("Correlation Coefficient:", correlation\_coefficient, "\n")

25. Calculate summary measures (mean, standard deviation, first quartile, third quartile and interquartile range) for the variable you graphed above. Do this for the entire data set, as well as for the different groups formed by the categorical variable that you used to divide the data set using Census.csv data file.

# Load necessary library

library(dplyr)

# Read the census.csv dataset

census\_data <- read.csv("census.csv")

# Choose a quantitative variable (replace 'VariableName' with the actual variable name)

variable <- census\_data$VariableName

# Calculate summary measures for the entire dataset

summary\_measures\_all <- summary(variable)

# Calculate summary measures for different groups formed by a categorical variable

summary\_measures\_by\_group <- census\_data %>%

group\_by(CategoricalVariable) %>%

summarise(

Mean = mean(VariableName),

SD = sd(VariableName),

Q1 = quantile(VariableName, 0.25),

Q3 = quantile(VariableName, 0.75),

IQR = IQR(VariableName)

)

# Display the results

print("Summary Measures for the Entire Dataset:")

print(summary\_measures\_all)

print("Summary Measures by Group:")

print(summary\_measures\_by\_group)

26. Use the appropriate R package to read data from an Excel file ("data.xlsx") and print 5-point summary of the dataframe along with box-whisker plot and dot plot using drinks.csv data file.

library(readxl)

library(ggplot2)

# Read data from Excel file

excel\_data <- read\_excel("path/to/data.xlsx")

# Read drinks data from CSV file

drinks\_data <- read.csv("path/to/drinks.csv")

# Print 5-point summary

cat("5-Point Summary of DataFrame:\n")

summary(excel\_data)

# Box-whisker plot

boxplot(excel\_data, main = "Box-Whisker Plot")

# Dot plot using ggplot2

dot\_plot <- ggplot(drinks\_data, aes(x = your\_variable)) +

geom\_dotplot(binaxis = "y", stackdir = "center", dotsize = 0.7) +

labs(title = "Dot Plot") +

theme\_minimal()

# Display dot plot

print(dot\_plot)

27. to create standardized (Z-) scores for several variables Using the preexisting Drinks.csv data file and Display the data from census.csv using line graph, scatter plot, histogram and dot plot.

# Read data from Drinks.csv

drinks\_data <- read.csv("Drinks.csv")

# Calculating Z-scores for several variables

z\_scores <- scale(drinks\_data[, c("variable1", "variable2", "variable3")])

# Displaying data using different plots

plot(drinks\_data$variable1, type = "l", col = "blue", xlab = "X", ylab = "Y")

plot(drinks\_data$variable1, drinks\_data$variable2, xlab = "Var1", ylab = "Var2")

hist(drinks\_data$variable3, xlab = "Var3")

dotchart(drinks\_data$variable1, labels = rownames(drinks\_data), cex = 0.7)

28. run Frequencies to explore the distributions of several variables Using the preexisting provided .csv data file and display the data from csv file in line graph, scatter plot, histogram and dot plot

# Assuming 'csv\_data' is the preexisting CSV data read into a dataframe

# Plot line graph

plot(csv\_data$variable1, type = "l", col = "blue", xlab = "X", ylab = "Y")

# Scatter plot

plot(csv\_data$variable1, csv\_data$variable2, xlab = "Var1", ylab = "Var2")

# Histogram

hist(csv\_data$variable3, xlab = "Var3")

# Dot plot

dotchart(csv\_data$variable1, labels = rownames(csv\_data), cex = 0.7)

29. to obtain summary statistics for scale variables Using the preexisting Drinks.csv data file and visualize it using any two types of graphs.

drinks\_data <- read.csv("D:/ASA/csv/Drinks.csv")

# Summary statistics for scale variables

summary\_stats <- summary(drinks\_data[, c("variable1", "variable2")])

# Visualize summary statistics using boxplots

boxplot(drinks\_data$variable1, drinks\_data$variable2, names = c("Var1", "Var2"), col = c("blue", "red"))

30. to visualize the relationship between two scale variables from provided data file creating scatter plots and to quantify this relationship with the correlation coefficient.

drinks\_data <- read.csv("D:/ASA/csv/Drinks.csv")

# Scatter plot for two scale variables

plot(census\_data$variable1, census\_data$variable2, xlab = "Var1", ylab = "Var2")

# Calculate correlation coefficient

correlation\_coeff <- cor(census\_data$variable1, census\_data$variable2)

print(paste("Correlation Coefficient:", correlation\_coeff))