

LAB EXERCISE 2

R PROGRAMMING

NAME : KRISHVANTH KUMAR E

REG NO:192125063

Set-I

1. (i) Write a function called `kelvin_to_celsius()` that takes a temperature in Kelvin and returns that temperature in Celsius (Hint: To convert from Kelvin to Celsius you subtract 273.15)

(ii) Write suitable R code to compute the mean, median ,mode of the following values

`c(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)`

(iii) Write R code to find 2nd

highest and 3rd Lowest value of above problem.

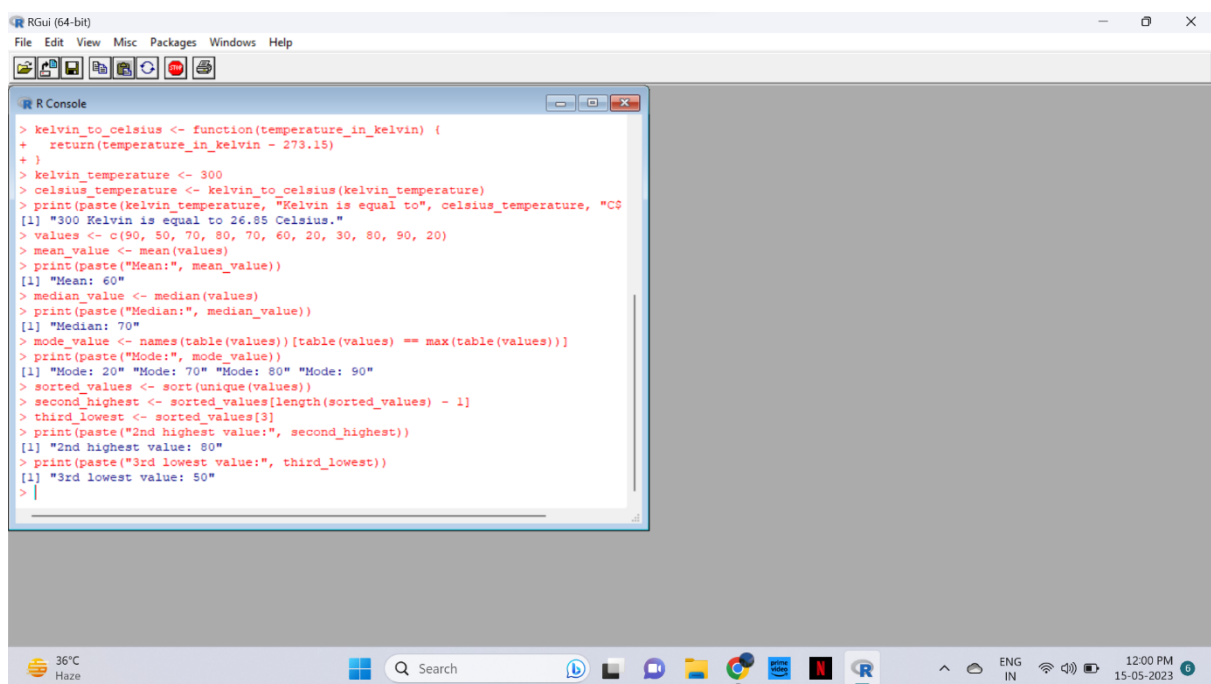
CODE

```
kelvin_to_celsius <- function(temperature_in_kelvin) {  
  return(temperature_in_kelvin - 273.15)  
}  
  
kelvin_temperature <- 300  
celsius_temperature <- kelvin_to_celsius(kelvin_temperature)  
print(paste(kelvin_temperature, "Kelvin is equal to", celsius_temperature, "Celsius."))  
  
values <- c(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)  
  
mean_value <- mean(values)  
print(paste("Mean:", mean_value))  
  
median_value <- median(values)  
print(paste("Median:", median_value))  
  
mode_value <- names(table(values))[table(values) == max(table(values))]  
print(paste("Mode:", mode_value))
```

LAB EXERCISE 2

R PROGRAMMING

```
sorted_values <- sort(unique(values))  
second_highest <- sorted_values[length(sorted_values) - 1]  
third_lowest <- sorted_values[3]  
print(paste("2nd highest value:", second_highest))  
print(paste("3rd lowest value:", third_lowest))
```



The screenshot shows the RGui (64-bit) window. The R Console contains the following code and output:

```
> kelvin_to_celsius <- function(temperature_in_kelvin) {  
+   return(temperature_in_kelvin - 273.15)  
+ }  
> kelvin_temperature <- 300  
> celsius_temperature <- kelvin_to_celsius(kelvin_temperature)  
> print(paste(kelvin_temperature, "Kelvin is equal to", celsius_temperature, "C"))  
[1] "300 Kelvin is equal to 26.85 Celsius."  
> values <- c(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)  
> mean_value <- mean(values)  
> print(paste("Mean:", mean_value))  
[1] "Mean: 60"  
> median_value <- median(values)  
> print(paste("Median:", median_value))  
[1] "Median: 70"  
> mode_value <- names(table(values))[table(values) == max(table(values))]  
> print(paste("Mode:", mode_value))  
[1] "Mode: 20" "Mode: 70" "Mode: 80" "Mode: 90"  
> sorted_values <- sort(unique(values))  
> second_highest <- sorted_values[length(sorted_values) - 1]  
> third_lowest <- sorted_values[3]  
> print(paste("2nd highest value:", second_highest))  
[1] "2nd highest value: 80"  
> print(paste("3rd lowest value:", third_lowest))  
[1] "3rd lowest value: 50"  
>
```

2. Explore the airquality dataset. It contains daily air quality measurements from New York during a period

of five months:

- Ozone: mean ozone concentration (ppb),
- Solar.R: solar radiation (Langley),
- Wind: average wind speed (mph),
- Temp: maximum daily temperature in degrees Fahrenheit,
- Month: numeric month (May=5, June=6, and so on),
- Day: numeric day of the month (1-31).

i. Compute the mean temperature(don't use build in function)

LAB EXERCISE 2

R PROGRAMMING

- ii. Extract the first five rows from airquality.
- iii. Extract all columns from airquality except Temp and Wind
- iv. Which was the coldest day during the period?
- v. How many days was the wind speed greater than 17 mph?

```
data(airquality)

mean_temperature <- sum(airquality$Temp) / length(airquality$Temp)

print(paste("Mean Temperature:", mean_temperature))
```

```
first_five_rows <- airquality[1:5, ]

print("First five rows:")

print(first_five_rows)
```

```
selected_columns <- airquality[, !(names(airquality) %in% c("Temp", "Wind"))]

print("Selected columns:")

print(selected_columns)
```

```
coldest_day <- airquality$Day[which.min(airquality$Temp)]

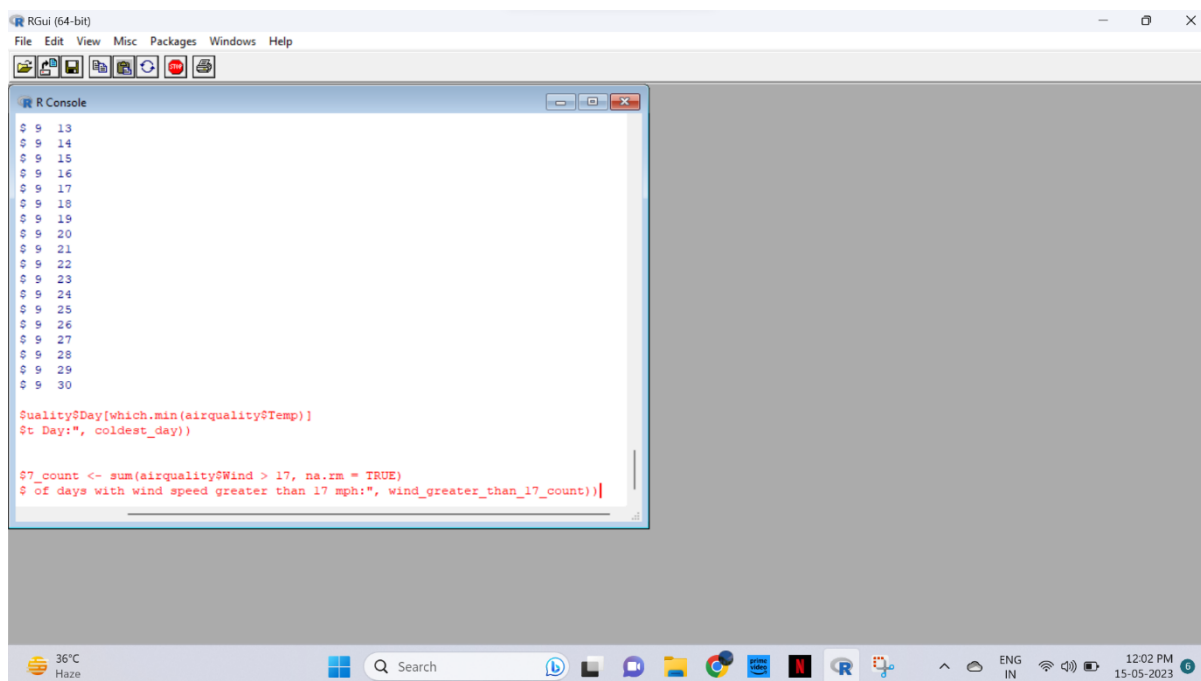
print(paste("Coldest Day:", coldest_day))
```

```
wind_greater_than_17_count <- sum(airquality$Wind > 17, na.rm = TRUE)

print(paste("Number of days with wind speed greater than 17 mph:",
wind_greater_than_17_count))
```

LAB EXERCISE 2

R PROGRAMMING



3. (i) Get the Summary Statistics of air quality dataset
- (ii) Melt airquality data set and display as a long – format data?
- (iii) Melt airquality data and specify month and day to be “ID variables”?
- (iv) Cast the molten airquality data set with respect to month and date features
- (v) Use cast function appropriately and compute the average of Ozone, Solar.R , Wind and temperature per month?

CODE

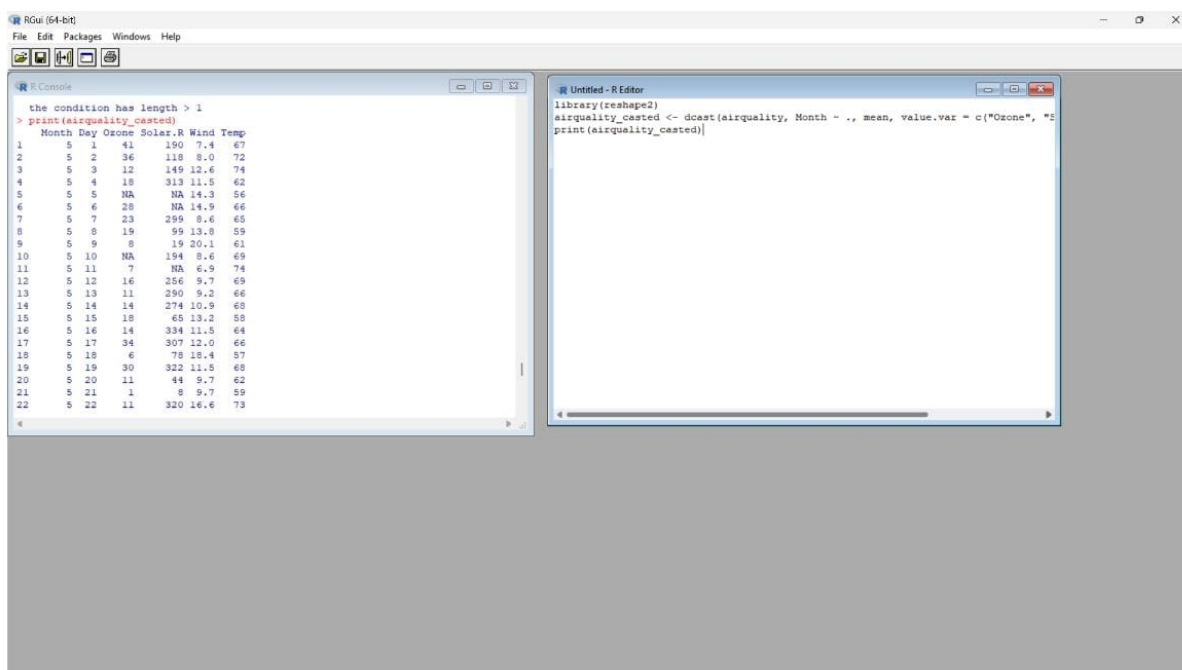
```
summary(airquality)  
  
library(reshape2)  
  
melted_data <- melt(airquality)  
  
print(melted_data)  
  
melted_data <- melt(airquality, id.vars = c("Month", "Day"))  
  
print(melted_data)  
  
cast_data <- dcast(melted_data, Month + Day ~ variable)  
  
print(cast_data)
```

LAB EXERCISE 2

R PROGRAMMING

```
cast_data <- dcast(melted_data, Month ~ variable, mean)
```

```
print(cast_data)
```



4.(i) Find any missing values(na) in features and drop the missing values if its less than 10% else replace that with mean of that feature.

(ii) Apply a linear regression algorithm using Least Squares Method on “Ozone” and “Solar.R”

(iii)Plot Scatter plot between Ozone and Solar and add regression line created by above model

CODE

```
# Find missing values
```

```
missing_values <- sum(is.na(airquality)) / length(airquality$Ozone)
```

```
if (missing_values < 0.1) {
```

```
  # Drop rows with missing values if less than 10%
```

LAB EXERCISE 2

R PROGRAMMING

```
airquality <- na.omit(airquality)

} else {

  # Replace missing values with mean of that feature

  airquality$Ozone[is.na(airquality$Ozone)] <- mean(airquality$Ozone, na.rm = TRUE)

  airquality$Solar.R[is.na(airquality$Solar.R)] <- mean(airquality$Solar.R, na.rm = TRUE)

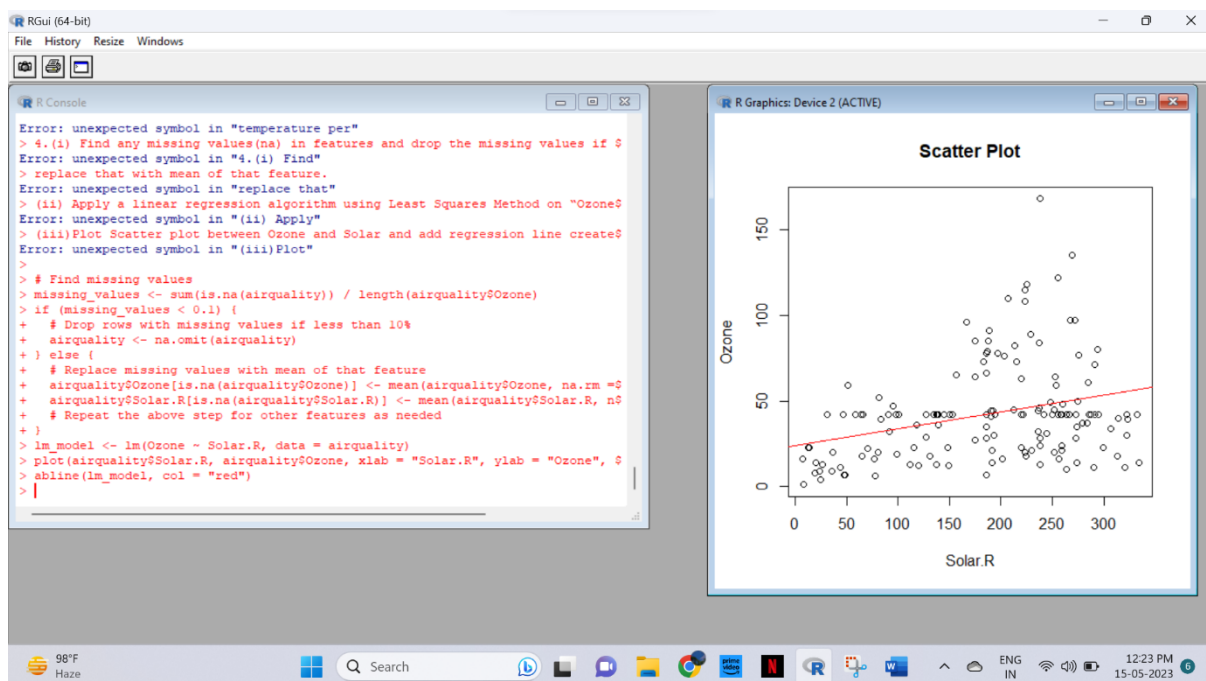
  # Repeat the above step for other features as needed

}

lm_model <- lm(Ozone ~ Solar.R, data = airquality)

plot(airquality$Solar.R, airquality$Ozone, xlab = "Solar.R", ylab = "Ozone", main = "Scatter Plot")

abline(lm_model, col = "red")
```



LAB EXERCISE 2

R PROGRAMMING

1. (i) Write a function to find the factorial of a given number using “for” Loop

(ii) Create a 3x4 matrix with 12 random numbers between 1-100; have the matrix be filled row-by-row, instead of column-by-column. Name the columns of the matrix uno, dos, tres, cuatro, and

the rows x, y, z. Scale the matrix by 10 and save the result.

(iii) Extract the column called “uno” as a vector from the original matrix and save the result

CODE

```
factorial <- function(n) {  
  result <- 1  
  for (i in 1:n) {  
    result <- result * i  
  }  
  return(result)  
}  
  
number <- 5  
factorial_of_number <- factorial(number)  
print(factorial_of_number)  
set.seed(123) # Setting a seed for reproducibility
```

Create the matrix

```
matrix_data <- matrix(sample(1:100, 12), nrow = 3, ncol = 4, byrow = TRUE)
```

Scale the matrix by 10

```
scaled_matrix <- matrix_data * 10
```

Name the columns and rows

LAB EXERCISE 2

R PROGRAMMING

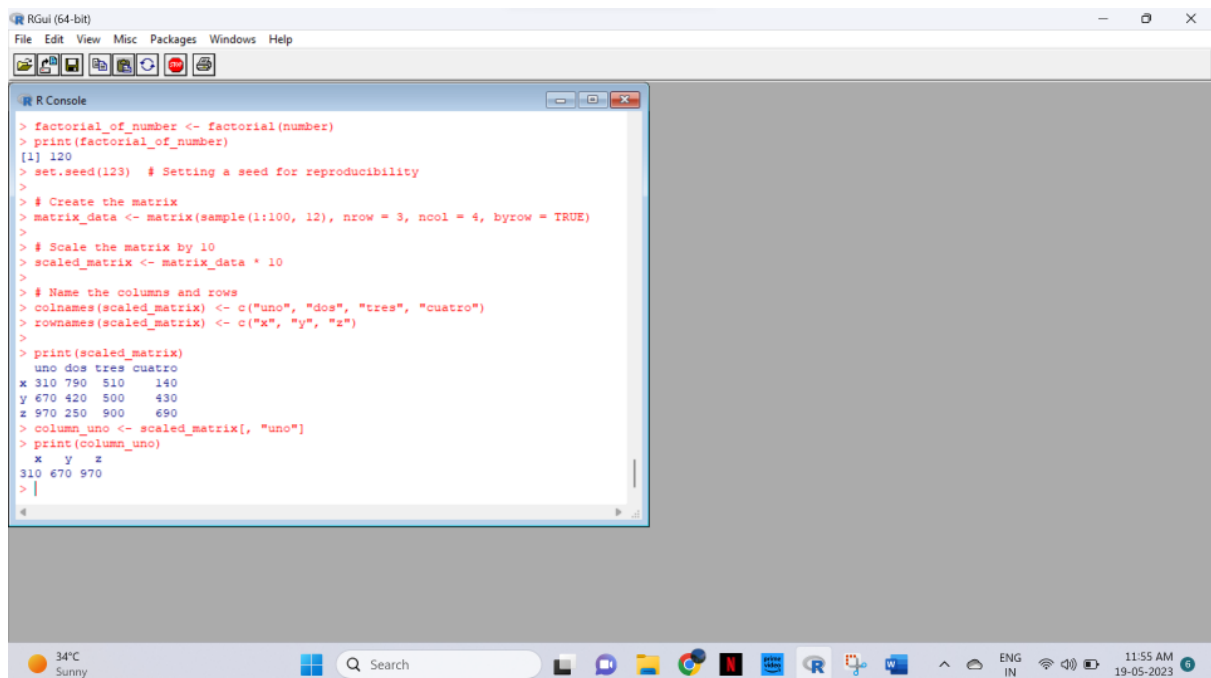
```
colnames(scaled_matrix) <- c("uno", "dos", "tres", "cuatro")
```

```
rownames(scaled_matrix) <- c("x", "y", "z")
```

```
print(scaled_matrix)
```

```
column_uno <- scaled_matrix[, "uno"]
```

```
print(column_uno)
```



The screenshot shows the RGui (64-bit) window with the R Console open. The console displays the following R code and its output:

```
> factorial_of_number <- factorial(number)
> print(factorial_of_number)
[1] 120
> set.seed(123) # Setting a seed for reproducibility
>
> # Create the matrix
> matrix_data <- matrix(sample(1:100, 12), nrow = 3, ncol = 4, byrow = TRUE)
>
> # Scale the matrix by 10
> scaled_matrix <- matrix_data * 10
>
> # Name the columns and rows
> colnames(scaled_matrix) <- c("uno", "dos", "tres", "cuatro")
> rownames(scaled_matrix) <- c("x", "y", "z")
>
> print(scaled_matrix)
      uno dos tres cuatro
x 310 790 510   140
y 670 420 500   430
z 970 250 900   690
> column_uno <- scaled_matrix[, "uno"]
> print(column_uno)
      x y z
310 670 970
> |
```

2. In 1936, Edgar Anderson collected data to quantify the geographic variations of iris flowers. The data

set consists of 50 samples from each of the three sub-species (iris setosa, iris virginica, and iris versicolor). Four features were measured in centimeters (cm): the lengths and the widths of both sepals and petals

(i) Find dimension, Structure, Summary statistics, Standard Deviation of all features.

(ii) Find mean and standard deviation of features grouped by three species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor)

(iii) Find quantile value of sepal width and length

LAB EXERCISE 2

R PROGRAMMING

(iv) create new data frame named iris1 which have a new column name

Sepal.Length.Cate that categorizes "Sepal.Length" by quantile

(v) Average value of numerical variables by two categorical variables: Species and

Sepal.Length.Cate.

CODE

```
# Load the iris dataset
```

```
data(iris)
```

```
# Dimension of the dataset
```

```
dimension <- dim(iris)
```

```
print(dimension)
```

```
# Structure of the dataset
```

```
structure <- str(iris)
```

```
# Summary statistics
```

```
summary_stats <- summary(iris)
```

```
# Standard deviation of all features
```

```
std_dev <- apply(iris[, 1:4], 2, sd)
```

```
print(std_dev)
```

```
# Mean and standard deviation grouped by species
```

```
mean_species <- aggregate(iris[, 1:4], by = list(iris$Species), FUN = mean)
```

```
sd_species <- aggregate(iris[, 1:4], by = list(iris$Species), FUN = sd)
```

```
print(mean_species)
```

```
print(sd_species)
```

```
# Quantile values of sepal width and length
```

LAB EXERCISE 2

R PROGRAMMING

```
quantile_width <- quantile(iris$Sepal.Width)
quantile_length <- quantile(iris$Sepal.Length)
print(quantile_width)
print(quantile_length)

# Create new data frame

iris1 <- iris

quantile_thresholds <- quantile(iris$Sepal.Length, probs = c(0, 0.25, 0.5, 0.75, 1))

categories <- cut(iris$Sepal.Length, breaks = quantile_thresholds, labels = c("Q1", "Q2", "Q3", "Q4"))

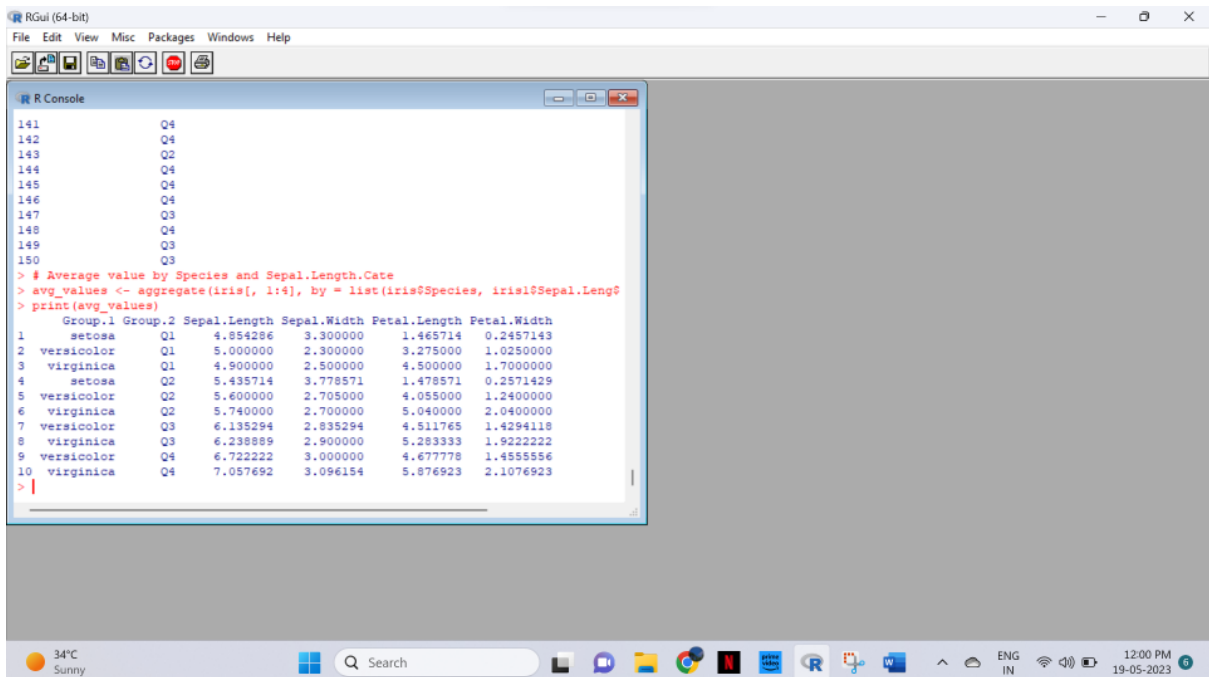
iris1$Sepal.Length.Cate <- categories

print(iris1)

# Average value by Species and Sepal.Length.Cate

avg_values <- aggregate(iris[, 1:4], by = list(iris$Species, iris1$Sepal.Length.Cate), FUN = mean)

print(avg_values)
```



```
RGui (64-bit)
File Edit View Misc Packages Windows Help

R Console

141         Q4
142         Q4
143         Q2
144         Q4
145         Q4
146         Q4
147         Q3
148         Q4
149         Q3
150         Q3

> # Average value by Species and Sepal.Length.Cate
> avg_values <- aggregate(iris[, 1:4], by = list(iris$Species, iris1$Sepal.Length.Cate), FUN = mean)
> print(avg_values)

  Group.1 Group.2 Sepal.Length Sepal.Width Petal.Length Petal.Width
1  setosa   Q1      4.854286    3.300000    1.465714    0.2457143
2  versicolor Q1      5.000000    2.300000    3.275000    1.0250000
3  virginica  Q1      4.900000    2.500000    4.500000    1.7000000
4    setosa   Q2      5.435714    3.778571    1.478571    0.2571429
5  versicolor Q2      5.600000    2.705000    4.055000    1.2400000
6  virginica  Q2      5.740000    2.700000    5.040000    2.0400000
7  versicolor Q3      6.135294    2.835294    4.511765    1.4294118
8  virginica  Q3      6.238889    2.900000    5.283333    1.9222222
9  versicolor Q4      6.722222    3.000000    4.677778    1.4555556
10 virginica  Q4      7.057692    3.096154    5.876923    2.1076923
```

3. (i) Plot Scatter plot between sepals width and length grouped by Species
- (ii) Plot Scatter plot between petals width and length grouped by Species

LAB EXERCISE 2

R PROGRAMMING

(iii) Draw the Box plot for Sepals length grouped by Species

(iv) Draw the Box plot for petals length grouped by Species

(v) Find the correlation among the four features

CODE

```
i.)library(ggplot2)
```

```
# Scatter plot for Sepal Width vs. Length grouped by Species
```

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

```
  geom_point() +
```

```
  labs(x = "Sepal Width", y = "Sepal Length", title = "Scatter plot of  
Sepal Width vs. Length grouped by Species")
```

```
ii.)# Scatter plot for Petal Width vs. Length grouped by Species
```

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

```
  geom_point() +
```

```
  labs(x = "Petal Width", y = "Petal Length", title = "Scatter plot of  
Petal Width vs. Length grouped by Species")
```

```
iii.)# Box plot for Sepal Length grouped by Species
```

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

LAB EXERCISE 2

R PROGRAMMING

```
geom_boxplot() +  
  labs(x = "Species", y = "Sepal Length", title = "Box plot of Sepal  
Length grouped by Species")
```

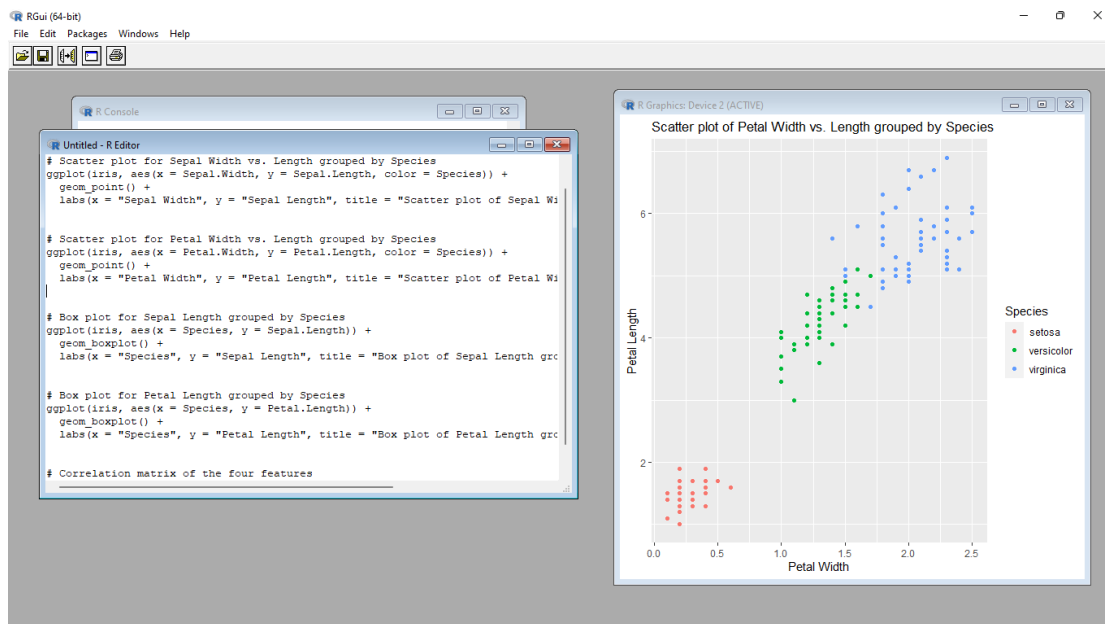
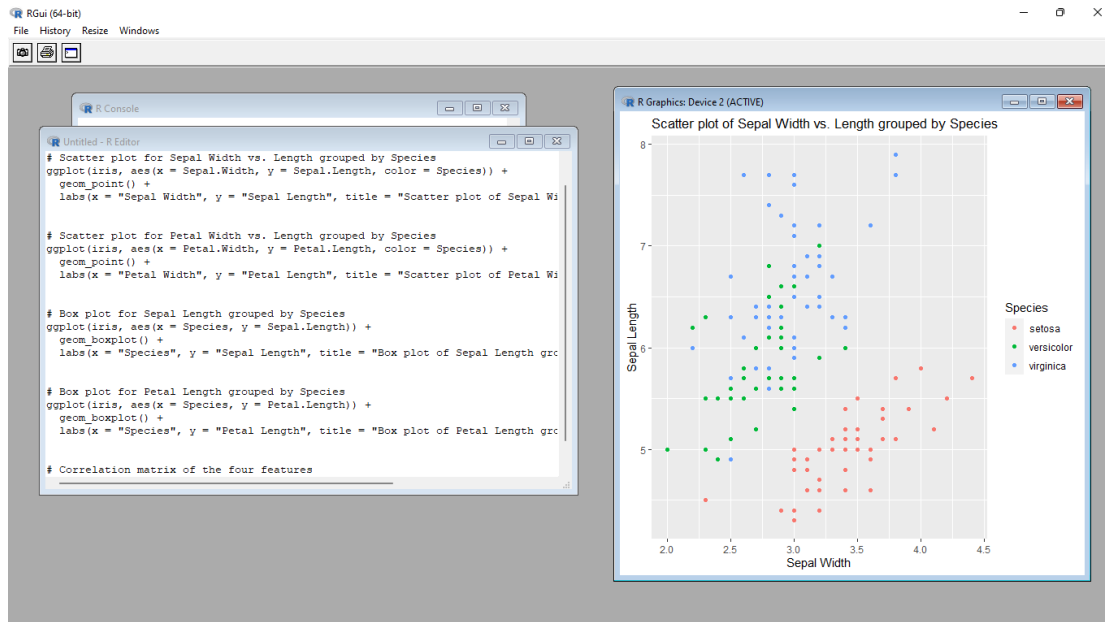
```
iv.)# Box plot for Petal Length grouped by Species  
ggplot(iris, aes(x = Species, y = Petal.Length)) +  
  geom_boxplot() +  
  labs(x = "Species", y = "Petal Length", title = "Box plot of Petal  
Length grouped by Species")
```

```
V.)  
# Correlation matrix of the four features  
cor(iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length",  
"Petal.Width")])
```

Output:

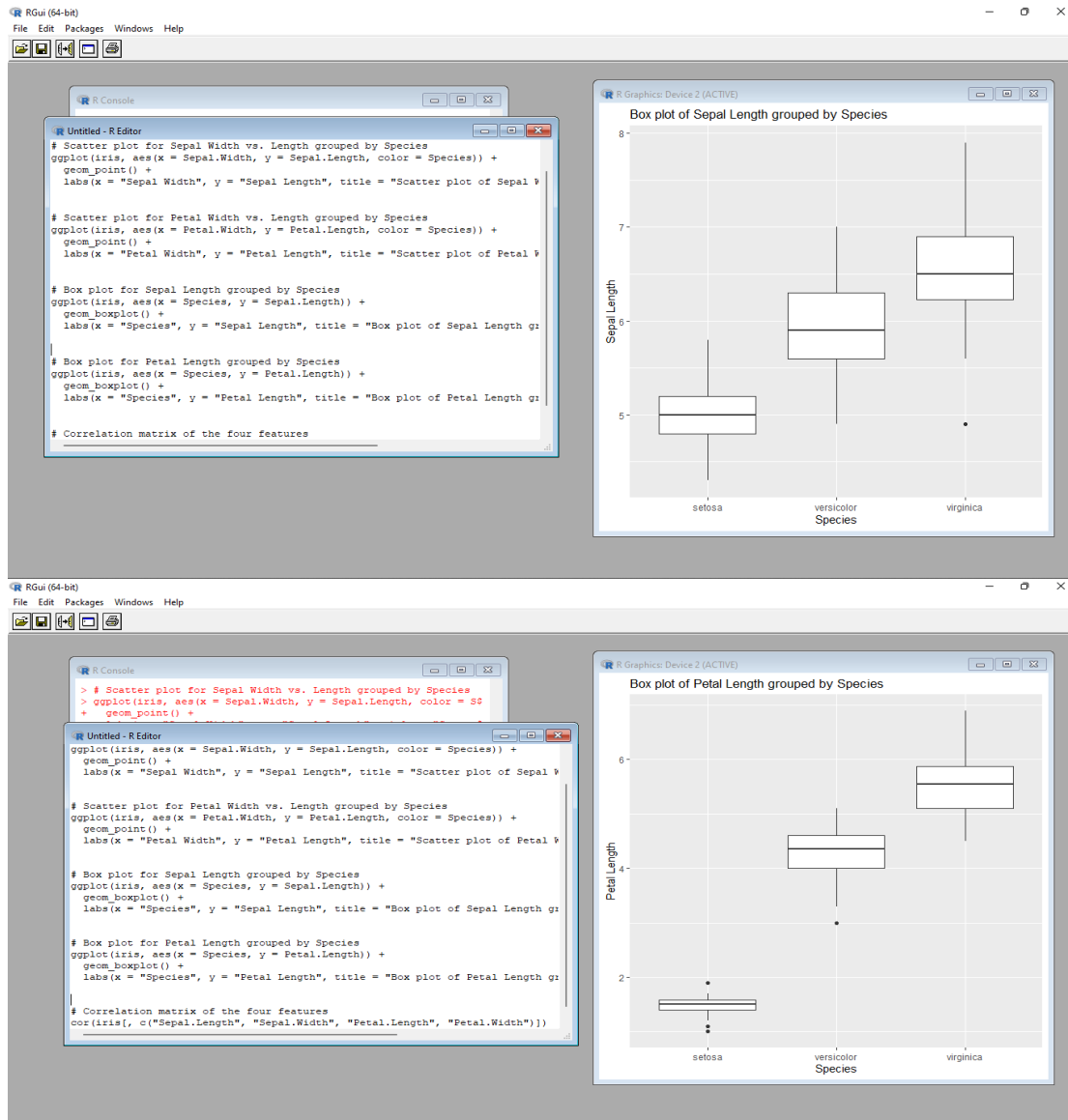
LAB EXERCISE 2

R PROGRAMMING



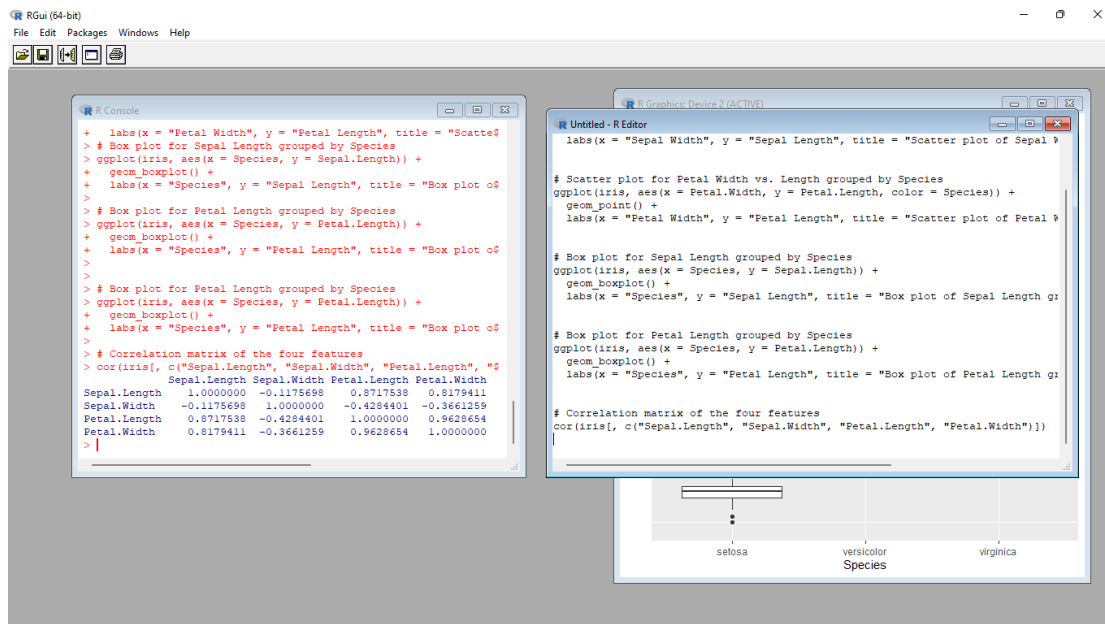
LAB EXERCISE 2

R PROGRAMMING



LAB EXERCISE 2

R PROGRAMMING



4.(i) Randomly Sample the iris dataset such as 50% data for training and 50% for test

(ii) find summary statistics of above train and test dataset.

(iii) Create Logistics regression with train data

(iv) Predict the probability of the model using test data

(v) Create Confusion matrix for above test model

CODE

i.) # Set seed for reproducibility

set.seed(123)

Randomly sample the iris dataset

train_indices <- sample(1:nrow(iris), nrow(iris) * 0.5)

train_data <- iris[train_indices,]

test_data <- iris[-train_indices,]

LAB EXERCISE 2

R PROGRAMMING

ii.)# Summary statistics of train dataset

```
summary(train_data)
```

Summary statistics of test dataset

```
summary(test_data)
```

iii.)# Create logistic regression model

```
model <- glm(Species ~ ., data = train_data, family = binomial)
```

```
print(model)
```

iv.)# Predict probabilities using the test data

```
probabilities <- predict(model, newdata = test_data, type =  
"response")
```

```
print(probabilities)
```

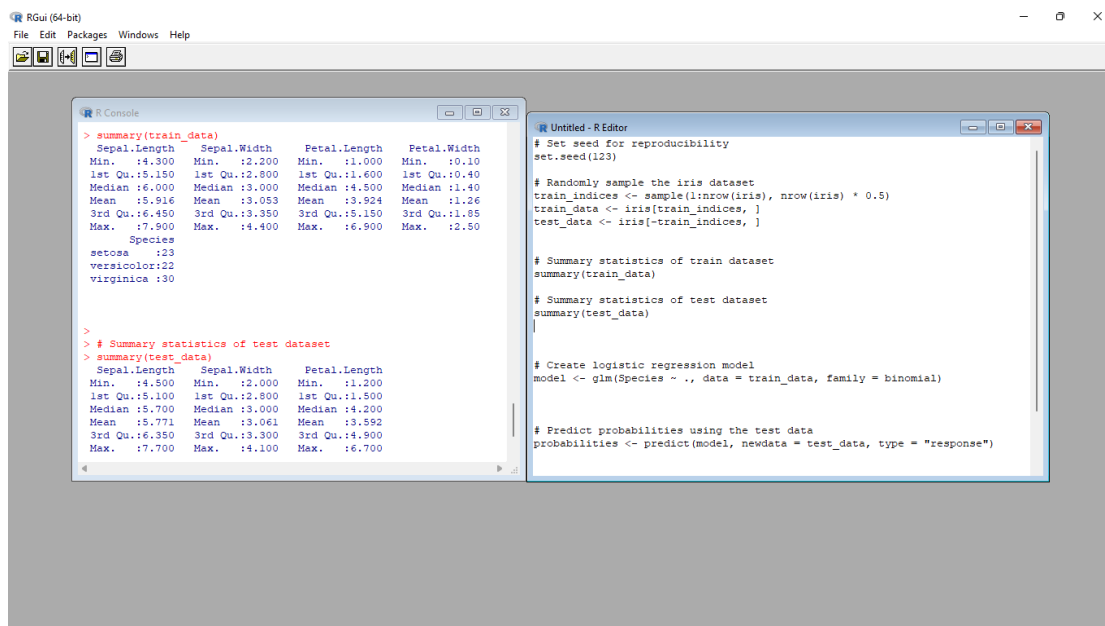
V.)# Create confusion matrix

LAB EXERCISE 2

R PROGRAMMING

```
predicted_classes <- ifelse(probabilities > 0.5, "versicolor", "setosa")  
table(test_data$Species, predicted_classes)
```

Output :



The screenshot shows the RGui (64-bit) interface. The R Console window displays the output of the R code, and the R Editor window shows the code being executed.

R Console Output:

```
> summary(train_data)  
      Sepal.Length      Sepal.Width      Petal.Length      Petal.Width  
Min.      :4.300   Min.      :2.200   Min.      :1.000   Min.      :0.10  
1st Qu.:5.150   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.40  
Median :6.000   Median :3.000   Median :4.500   Median :1.40  
Mean    :6.916   Mean    :3.053   Mean    :3.924   Mean    :1.26  
3rd Qu.:6.450   3rd Qu.:3.350   3rd Qu.:5.150   3rd Qu.:1.95  
Max.    :7.900   Max.    :4.400   Max.    :6.900   Max.    :2.50  
Species  
setosa    :23  
versicolor:22  
virginica  :30  
  
>  
> # Summary statistics of test dataset  
> summary(test_data)  
      Sepal.Length      Sepal.Width      Petal.Length  
Min.      :4.500   Min.      :2.000   Min.      :1.200  
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.500  
Median :5.700   Median :3.000   Median :4.200  
Mean    :5.771   Mean    :3.061   Mean    :3.552  
3rd Qu.:6.350   3rd Qu.:3.300   3rd Qu.:4.900  
Max.    :7.700   Max.    :4.100   Max.    :6.700
```

R Editor Code:

```
# Set seed for reproducibility  
set.seed(123)  
  
# Randomly sample the iris dataset  
train_indices <- sample(1:nrow(iris), nrow(iris) * 0.5)  
train_data <- iris[train_indices, ]  
test_data <- iris[-train_indices, ]  
  
# Summary statistics of train dataset  
summary(train_data)  
  
# Summary statistics of test dataset  
summary(test_data)  
  
# Create logistic regression model  
model <- glm(Species ~ ., data = train_data, family = binomial)  
  
# Predict probabilities using the test data  
probabilities <- predict(model, newdata = test_data, type = "response")
```

LAB EXERCISE 2

R PROGRAMMING

The top screenshot shows the R GUI with the following code in the R Console:

```
Call: glm(formula = Species ~ ., family = binomial, data = tra$
Coefficients:
(Intercept) Sepal.Length Sepal.Width Petal.Length
      44.355       -6.709       -19.160        8.640
      Petal.Width
      31.425

Degrees of Freedom: 74 Total (i.e. Null); 70 Residual
Null Deviance: 92.46
Residual Deviance: 1.135e-09 AIC: 10
>
> # Predict probabilities using the test data
> probabilities <- predict(model, newdata = test_data, type = "%")
> print(probabilities)
```

The bottom screenshot shows the R GUI with the following code in the R Console:

```
84      85      88      94
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
95      98      100     101
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
104     105     107     108
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
111     113     115     116
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
122     123     125     131
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
133     135     139     140
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
141     145     146
1.000000e+00 1.000000e+00 1.000000e+00
>
> # Create confusion matrix
> predicted_classes <- ifelse(probabilities > 0.5, "versicolor", "setosa")
> table(test_data$Species, predicted_classes)
```

The bottom screenshot also shows the R Console output for the confusion matrix:

```
      Predicted classes
setosa versicolor
versicolor 0      28
virginica  0      20
> |
```

Set-III

1. Suppose you track your commute times for two weeks (10 days) and you find the following

times in minutes 17 16 20 24 22 15 21 15 17 22 Enter this into R as vector data type.

(i)create function maxi to find the longest commute time, the function avger to find the average and

LAB EXERCISE 2

R PROGRAMMING

the function mini to find the minimum.

(ii)Oops, the 24 was a mistake. It should have been 18. How can you fix this? Do so, and then find

the new average using above functions.

(iii)How many times was your commute 20 minutes or more?

Input:

```
# Vector of commute times
```

```
commute_times <- c(17, 16, 20, 24, 22, 15, 21, 15, 17, 22)
```

```
# Function to find the longest commute time
```

```
maxi <- function(commute_times) {  
  max(commute_times)  
}
```

```
# Function to find the average commute time
```

```
avger <- function(commute_times) {  
  mean(commute_times)  
}
```

```
# Function to find the minimum commute time
```

```
mini <- function(commute_times) {  
  min(commute_times)  
}
```

```
# Calling the functions
```

```
max_time <- maxi(commute_times)
```

LAB EXERCISE 2

R PROGRAMMING

```
avg_time <- avger(commute_times)
```

```
min_time <- mini(commute_times)
```

```
# Printing the results
```

```
print(paste("Longest commute time:", max_time))
```

```
print(paste("Average commute time:", avg_time))
```

```
print(paste("Minimum commute time:", min_time))
```

```
# Correcting the mistake
```

```
commute_times[commute_times == 24] <- 18
```

```
# Finding the new average using the functions
```

```
new_avg_time <- avger(commute_times)
```

```
# Printing the new average
```

```
print(paste("New average commute time:", new_avg_time))
```

```
# Counting occurrences of 20 minutes or more
```

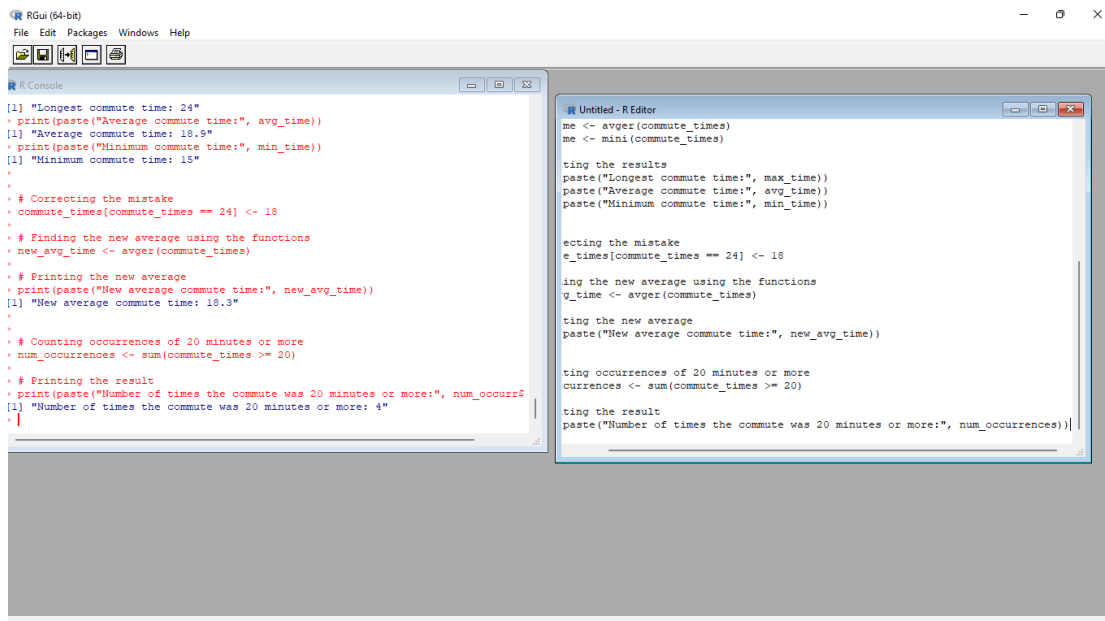
```
num_occurrences <- sum(commute_times >= 20)
```

```
# Printing the result
```

```
print(paste("Number of times the commute was 20 minutes or more:", num_occurrences))
```

LAB EXERCISE 2

R PROGRAMMING



```
[1] "Longest commute time: 24"
> print(paste("Average commute time:", avg_time))
[1] "Average commute time: 18.9"
> print(paste("Minimum commute time:", min_time))
[1] "Minimum commute time: 15"
>
>
> # Correcting the mistake
> commute_times[commute_times == 24] <- 18
>
> # Finding the new average using the functions
> new_avg_time <- avger(commute_times)
>
> # Printing the new average
> print(paste("New average commute time:", new_avg_time))
[1] "New average commute time: 18.3"
>
>
> # Counting occurrences of 20 minutes or more
> num_occurrences <- sum(commute_times >= 20)
>
> # Printing the result
> print(paste("Number of times the commute was 20 minutes or more:", num_occurrences))
[1] "Number of times the commute was 20 minutes or more: 4"
>
>

me <- avger(commute_times)
me <- mini(commute_times)

ting the results
paste("Longest commute time:", max_time))
paste("Average commute time:", avg_time))
paste("Minimum commute time:", min_time))

ecting the mistake
e_times[commute_times == 24] <- 18

ing the new average using the functions
g_time <- avger(commute_times)

ting the new average
paste("New average commute time:", new_avg_time))

ting occurrences of 20 minutes or more
urrences <- sum(commute_times >= 20)

ting the result
paste("Number of times the commute was 20 minutes or more:", num_occurrences))
```

2. There is a popular built-in data set in R called "**mtcars**" (Motor Trend Car Road Tests), which is retrieved from the 1974 Motor Trend US Magazine.
- (i) Find the dimension of the data set
 - (ii) Give the statistical summary of the features.
 - (iii) Find the largest and smallest value of the variable hp (horsepower).
 - (iv) Give the mean of mileage per gallon (mpg) with respect to transmission model (feature named as 'am')
 - (v) Give the median of horsepower (hp) with respect to cylinder displacement(cyl)

Input:

```
# Load the mtcars dataset
```

```
data(mtcars)
```

```
# Get the dimension of the dataset
```

```
dim(mtcars)
```

LAB EXERCISE 2

R PROGRAMMING

```
# Get the summary of the dataset
```

```
summary(mtcars)
```

```
# Find the largest value of "hp"
```

```
max_hp <- max(mtcars$hp)
```

```
# Find the smallest value of "hp"
```

```
min_hp <- min(mtcars$hp)
```

```
# Print the results
```

```
print(paste("Largest value of horsepower (hp):", max_hp))
```

```
print(paste("Smallest value of horsepower (hp):", min_hp))
```

```
# Calculate the mean of mpg by transmission type (am)
```

```
mean_mpg <- tapply(mtcars$mpg, mtcars$am, mean)
```

```
# Print the mean mpg for each transmission type
```

```
print("Mean mpg by transmission type:")
```

```
print(mean_mpg)
```

```
# Calculate the median of hp by cylinder displacement (cyl)
```

```
median_hp <- tapply(mtcars$hp, mtcars$cyl, median)
```

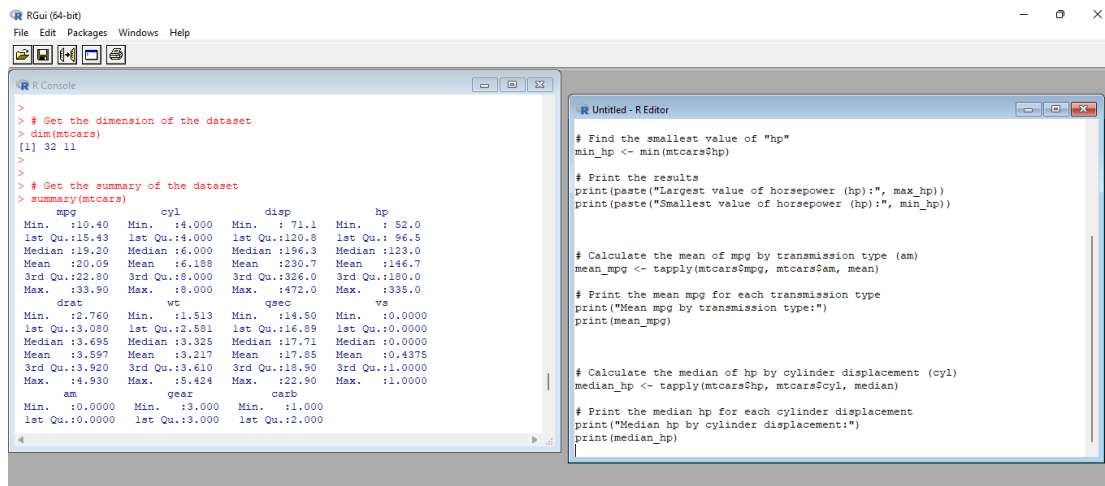
LAB EXERCISE 2

R PROGRAMMING

Print the median hp for each cylinder displacement

```
print("Median hp by cylinder displacement:")
```

```
print(median_hp)
```



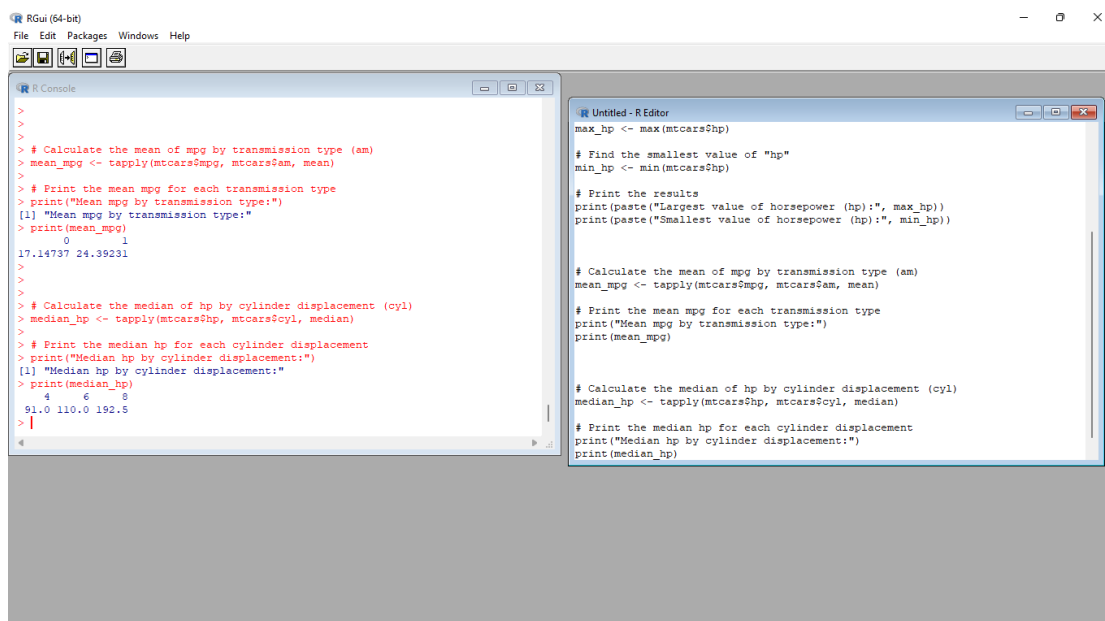
```
>
> # Get the dimension of the dataset
> dim(mtcars)
[1] 32 11
>
> # Get the summary of the dataset
> summary(mtcars)
      mpg          cyl          disp           hp
Min.   10.40   Min.    4.000   Min.   71.1   Min.    52.0
1st Qu:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
Median :15.20   Median :6.000   Median :196.3   Median :123.0
Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
      drat          wt          qsec          vs
Min.    2.760   Min.    1.513   Min.   14.50   Min.    0.0000
1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
Median :3.695   Median :3.325   Median :17.71   Median :0.0000
Mean   :3.597   Mean   :3.217   Mean   :17.55   Mean   :0.4375
3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
      am          gear          carb
Min.    0.0000   Min.    3.000   Min.    1.000
1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
```

```
# Find the smallest value of "hp"
min_hp <- min(mtcars$hp)

# Print the results
print(paste("Largest value of horsepower (hp):", max_hp))
print(paste("Smallest value of horsepower (hp):", min_hp))

# Calculate the mean of mpg by transmission type (am)
mean_mpg <- tapply(mtcars$mpg, mtcars$am, mean)

# Print the mean mpg for each transmission type
print("Mean mpg by transmission type:")
print(mean_mpg)
```



```
>
>
> # Calculate the mean of mpg by transmission type (am)
> mean_mpg <- tapply(mtcars$mpg, mtcars$am, mean)
>
> # Print the mean mpg for each transmission type
> print("Mean mpg by transmission type:")
[1] "Mean mpg by transmission type:"
> print(mean_mpg)
      0
17.14737 24.39231
>
> # Calculate the median of hp by cylinder displacement (cyl)
> median_hp <- tapply(mtcars$hp, mtcars$cyl, median)
>
> # Print the median hp for each cylinder displacement
> print("Median hp by cylinder displacement:")
[1] "Median hp by cylinder displacement:"
> print(median_hp)
      4      6      8
91.0 110.0 192.5
> |
```

```
max_hp <- max(mtcars$hp)

# Find the smallest value of "hp"
min_hp <- min(mtcars$hp)

# Print the results
print(paste("Largest value of horsepower (hp):", max_hp))
print(paste("Smallest value of horsepower (hp):", min_hp))

# Calculate the mean of mpg by transmission type (am)
mean_mpg <- tapply(mtcars$mpg, mtcars$am, mean)

# Print the mean mpg for each transmission type
print("Mean mpg by transmission type:")
print(mean_mpg)

# Calculate the median of hp by cylinder displacement (cyl)
median_hp <- tapply(mtcars$hp, mtcars$cyl, median)

# Print the median hp for each cylinder displacement
print("Median hp by cylinder displacement:")
print(median_hp)
```

3.(i) Create Scatter plot mpg vs hp, grouped by transmission model (feature named as 'am')

(ii) Create Box plot for mpg with respect to transmission model (feature named as 'am')

LAB EXERCISE 2

R PROGRAMMING

- (iii) Create histogram plot which shows statistical distribution of hp
- (iv) Draw the Bar Chart to show car distribution with respect to number of gears grouped by cylinder. (Grouped or multiple bar chart)
- (v) Draw Pie chart which shows the percentage of distribution by number of gears.

Input:

```
# Load the mtcars dataset
```

```
data(mtcars)
```

```
# Scatter plot of mpg vs hp, grouped by transmission model
```

```
plot(mtcars$hp, mtcars$mpg, col = mtcars$am, pch = 19, xlab = "Horsepower (hp)", ylab =  
"Miles per Gallon (mpg)")
```

```
legend("topright", legend = c("Automatic", "Manual"), col = c(1, 2), pch = 19, title =  
"Transmission")
```

```
# Box plot of mpg with respect to transmission model
```

```
boxplot(mpg ~ am, data = mtcars, xlab = "Transmission", ylab = "Miles per Gallon (mpg)",  
main = "Box Plot of MPG by Transmission")
```

```
# Histogram plot of hp
```

```
hist(mtcars$hp, breaks = 10, col = "skyblue", xlab = "Horsepower (hp)", ylab = "Frequency",  
main = "Histogram of HP")
```


LAB EXERCISE 2

R PROGRAMMING

```
# Bar chart of car distribution by number of gears, grouped by cylinder

barplot(table(mtcars$gear, mtcars$cyl), beside = TRUE, col = c("skyblue", "orange",
"green"), xlab = "Number of Gears", ylab = "Frequency", main = "Car Distribution by
Number of Gears and Cylinder")

legend("topright", legend = c("4 Cylinder", "6 Cylinder", "8 Cylinder"), fill = c("skyblue",
"orange", "green"))
```

```
# Pie chart of percentage distribution by number of gears

gear_counts <- table(mtcars$gear)

labels <- paste(names(gear_counts), "Gears")

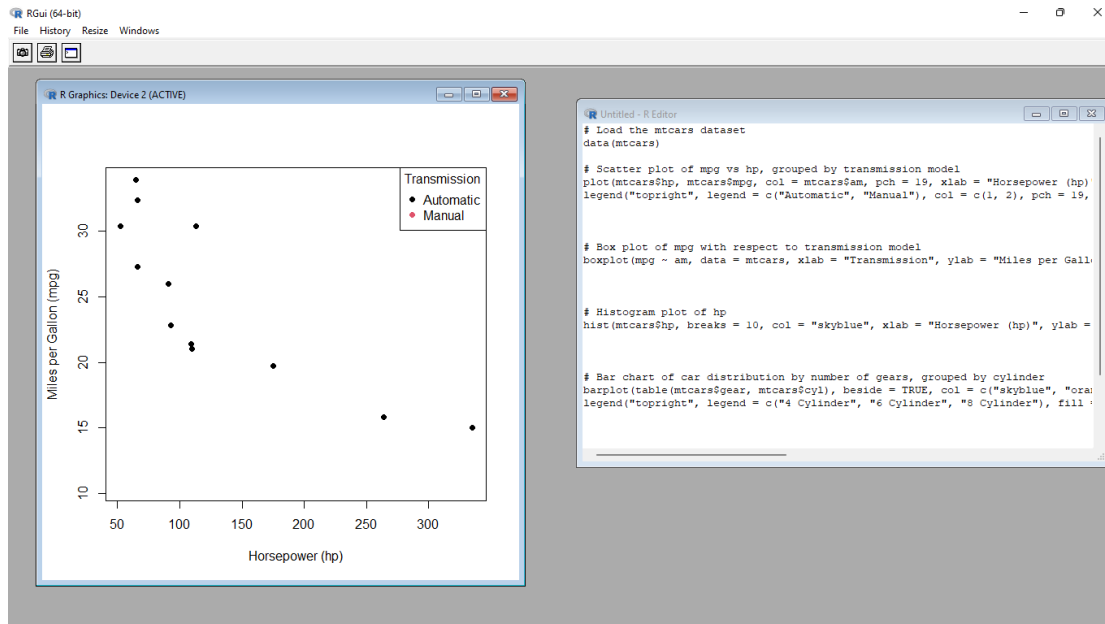
percentages <- round(gear_counts / sum(gear_counts) * 100, 1)

pie(gear_counts, labels = labels, main = "Percentage Distribution by Number of Gears")

legend("topright", legend = paste(labels, "-", percentages, "%"), cex = 0.8, fill =
rainbow(length(gear_counts)))
```

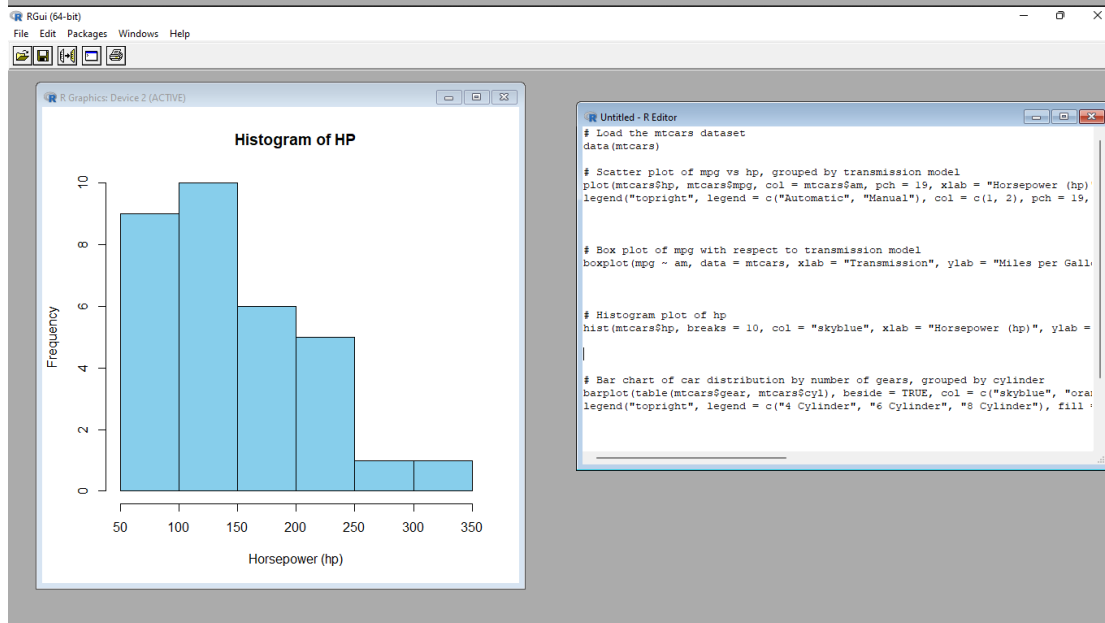
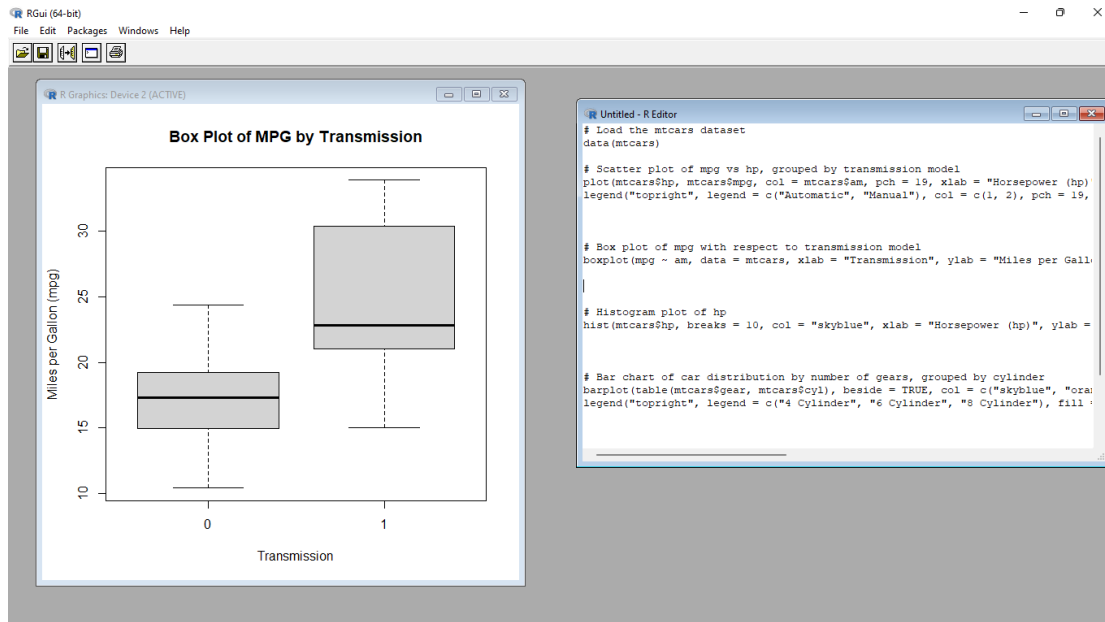
LAB EXERCISE 2

R PROGRAMMING



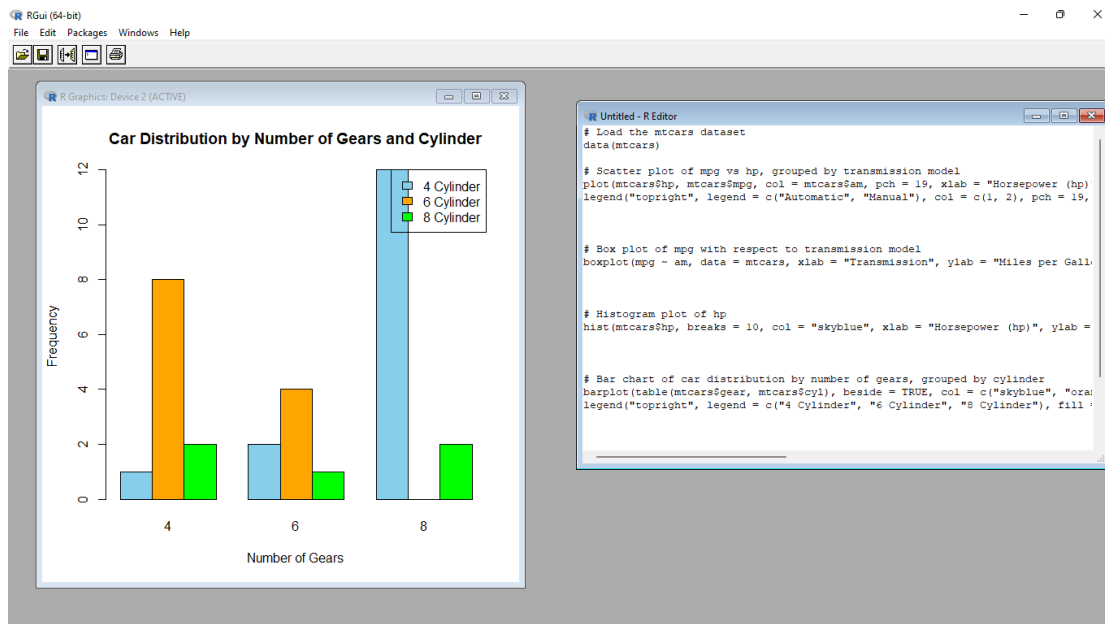
LAB EXERCISE 2

R PROGRAMMING



LAB EXERCISE 2

R PROGRAMMING



4. (i) Generate a multiple regression model using the built-in dataset mtcars. Establish the relationship

between "mpg" as a response variable with "disp", "hp" and "wt" as predictor variables .

(ii) Plot the multiple regression line model with above model parameters.

(iii) Predict the mileage of the car with dsp=221, hp=102 and wt=2.91

Input:

```
# Load the mtcars dataset
```

```
data(mtcars)
```

```
# Create the multiple regression model
```

```
reg_model <- lm(mpg ~ disp + hp + wt, data = mtcars)
```

LAB EXERCISE 2

R PROGRAMMING

```
# Summary of the regression model
```

```
summary(reg_model)
```

```
# Scatter plot of actual mpg vs. predicted mpg
```

```
plot(mtcars$mpg, predict(reg_model), xlab = "Actual MPG", ylab = "Predicted MPG", main = "Actual vs. Predicted MPG")
```

```
abline(0, 1, col = "red", lwd = 2) # Add a reference line with slope 1
```

```
# Create a data frame with predictor variables for prediction
```

```
new_data <- data.frame(displacement = 221, horsepower = 102, weight = 2.91)
```

```
# Predict the mileage for the new data using the regression model
```

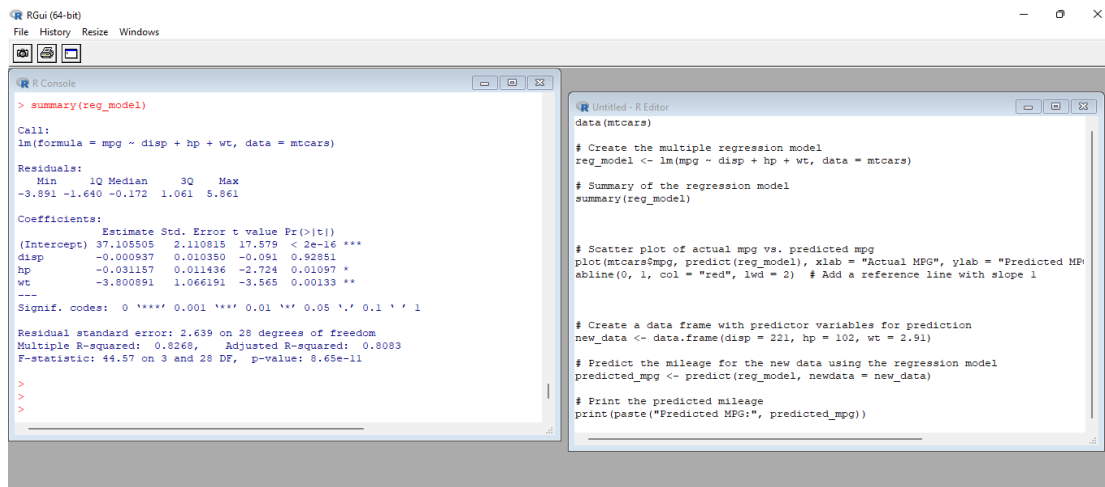
```
predicted_mpg <- predict(reg_model, newdata = new_data)
```

```
# Print the predicted mileage
```

```
print(paste("Predicted MPG:", predicted_mpg))
```

LAB EXERCISE 2

R PROGRAMMING



```
> summary(reg_model)

Call:
lm(formula = mpg ~ disp + hp + wt, data = mtcars)

Residuals:
    Min       1Q   Median       3Q      Max
-3.891 -1.640 -0.172  1.061  5.861

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.105505    2.110815   17.579 < 2e-16 ***
disp        -0.000937    0.010350   -0.091  0.92851
hp          -0.031157    0.011436   -2.724  0.01097 *
wt          -3.800891    1.066191   -3.565  0.00133 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.639 on 28 degrees of freedom
Multiple R-squared:  0.8269,    Adjusted R-squared:  0.8083
F-statistic: 44.57 on 3 and 28 DF,  p-value: 8.65e-11

>
>
```

```
data(mtcars)

# Create the multiple regression model
reg_model <- lm(mpg ~ disp + hp + wt, data = mtcars)

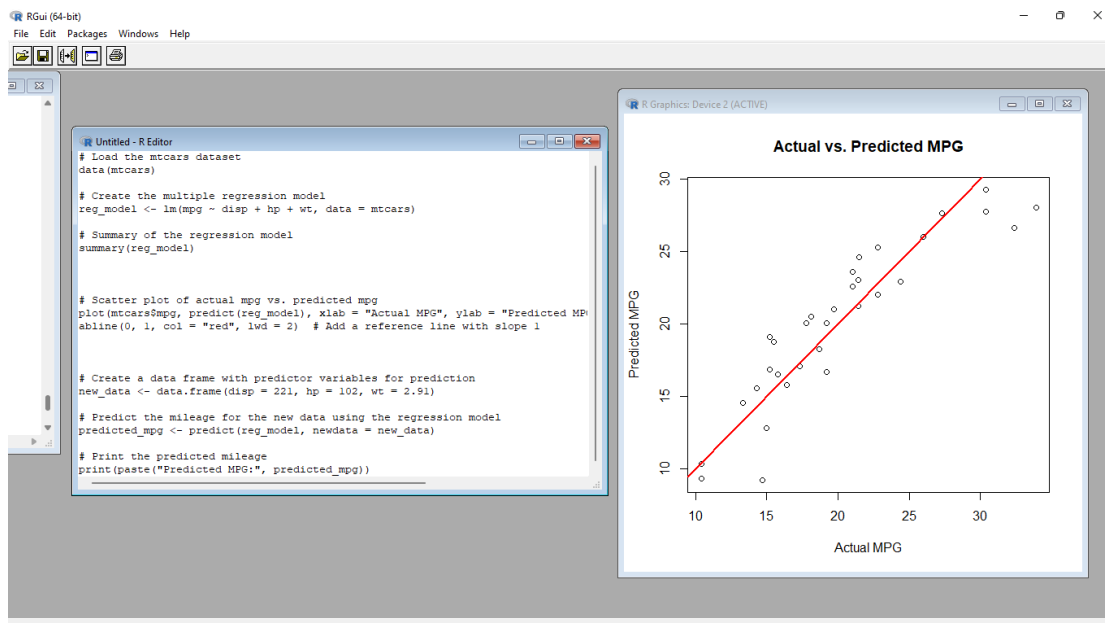
# Summary of the regression model
summary(reg_model)

# Scatter plot of actual mpg vs. predicted mpg
plot(mtcars$mpg, predict(reg_model), xlab = "Actual MPG", ylab = "Predicted MPG",
      abline(0, 1, col = "red", lwd = 2) # Add a reference line with slope 1

# Create a data frame with predictor variables for prediction
new_data <- data.frame(displ = 221, hp = 102, wt = 2.91)

# Predict the mileage for the new data using the regression model
predicted_mpg <- predict(reg_model, newdata = new_data)

# Print the predicted mileage
print(paste("Predicted MPG:", predicted_mpg))
```



Set IV

1. (i) Write a function in R programming to print generate Fibonacci sequence using

Recursion in R .

(ii) Find sum of natural numbers up-to 10, without formula using loop statement.

(iii) create a vector 1:10 and Find a square of each number and store that in a separate list.

LAB EXERCISE 2

R PROGRAMMING

Input:

```
i.)fibonacci <- function(n) {  
  if (n <= 0) {  
    stop("Input must be a positive integer.")  
  } else if (n == 1) {  
    return(0)  
  } else if (n == 2) {  
    return(1)  
  } else {  
    return(fibonacci(n - 1) + fibonacci(n - 2))  
  }  
}
```

Example usage

```
n <- 10  
fib_sequence <- sapply(1:n, fibonacci)  
print(fib_sequence)
```

```
ii.)sum_natural_numbers <- 0
```

```
for (i in 1:10) {  
  sum_natural_numbers <- sum_natural_numbers + i  
}
```

LAB EXERCISE 2

R PROGRAMMING

```
print(sum_natural_numbers)
```

```
iii.)numbers <- 1:10
```

```
squares <- lapply(numbers, function(x) x^2)
```

```
print(squares)
```


LAB EXERCISE 2

R PROGRAMMING

The top screenshot shows the R GUI with the following code in the console window:

```
> n <- 10
> fib_sequence <- sapply(1:n, fibonacci)
> print(fib_sequence)
[1] 0 1 1 2 3 5 8 13 21 34
> sum_natural_numbers <- 0
> for (i in 1:10) {
+   sum_natural_numbers <- sum_natural_numbers + i
+ }
> print(sum_natural_numbers)
[1] 55
>
> numbers <- 1:10
> squares <- lapply(numbers, function(x) x^2)
>
> print(squares)
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9
```

The bottom screenshot shows the R GUI with the following code in the console window:

```
[[3]]
[1] 9

[[4]]
[1] 16

[[5]]
[1] 25

[[6]]
[1] 36

[[7]]
[1] 49

[[8]]
[1] 64

[[9]]
[1] 81

[[10]]
[1] 100
> |
```

2. **MTCARS**(motor trend car road test) comprises fuel consumption, performance and

10 aspects of automobile design for 32 automobiles. It comes pre-installed with **dplyr** package

in R.

(i)Find the dimension of the data set

(ii)Give the statistical summary of the features.

LAB EXERCISE 2

R PROGRAMMING

(iii) Print the categorical features in Dataset

(iv) Find the average weight(wt) grouped by Engine shape(vs)

(v) Find the largest and smallest value of the variable weight with respect to Engine shape

Input:

```
# Load the dplyr package
```

```
library(dplyr)
```

```
# Check the dimension of the mtcars dataset
```

```
dim(mtcars)
```

```
# Summarize the features of the mtcars dataset
```

```
summary(mtcars)
```

```
# Identify the categorical features in the mtcars dataset
```

```
categorical_features <- sapply(mtcars, is.factor)
```

```
# Print the categorical features
```

```
names(mtcars)[categorical_features]
```

```
# Calculate the average weight (wt) grouped by Engine shape (vs)
```

```
mtcars %>%
```

```
  group_by(vs) %>%
```

LAB EXERCISE 2

R PROGRAMMING

```
summarize(avg_weight = mean(wt))
```

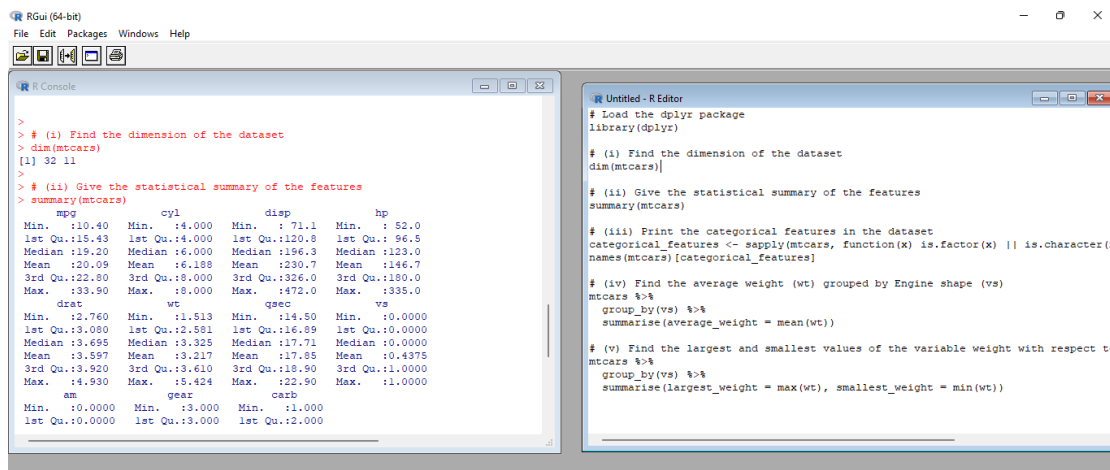
Find the largest and smallest values of the variable weight with respect to Engine shape

```
mtcars %>%
```

```
group_by(vs) %>%
```

```
summarize(largest_weight = max(wt),
```

```
smallest_weight = min(wt))
```



The screenshot shows the RGui (64-bit) interface. The R Console window on the left displays the following code and output:

```
>
> # (i) Find the dimension of the dataset
> dim(mtcars)
[1] 32 11
>
> # (ii) Give the statistical summary of the features
> summary(mtcars)
```

mpg	cyl	disp	hp
Min. :10.40	Min. :4.000	Min. : 71.1	Min. : 52.0
1st Qu.:15.43	1st Qu.:4.000	1st Qu.:120.8	1st Qu.: 96.5
Median :19.20	Median :6.000	Median :196.3	Median :123.0
Mean :20.09	Mean :6.188	Mean :230.7	Mean :146.7
3rd Qu.:22.80	3rd Qu.:8.000	3rd Qu.:326.0	3rd Qu.:180.0
Max. :33.90	Max. :8.000	Max. :472.0	Max. :335.0

drat	wt	qsec	vs
Min. :12.760	Min. :1.513	Min. :14.50	Min. :0.0000
1st Qu.:13.080	1st Qu.:2.581	1st Qu.:16.89	1st Qu.:0.0000
Median :13.695	Median :3.325	Median :17.71	Median :0.0000
Mean :13.597	Mean :3.217	Mean :17.85	Mean :0.4375
3rd Qu.:13.920	3rd Qu.:3.610	3rd Qu.:18.90	3rd Qu.:1.0000
Max. :14.930	Max. :5.424	Max. :22.90	Max. :1.0000

am	gear	carb
Min. :0.0000	Min. :3.000	Min. :1.000
1st Qu.:0.0000	1st Qu.:3.000	1st Qu.:2.000

The Untitled - R Editor window on the right contains the following code:

```
# Load the dplyr package
library(dplyr)

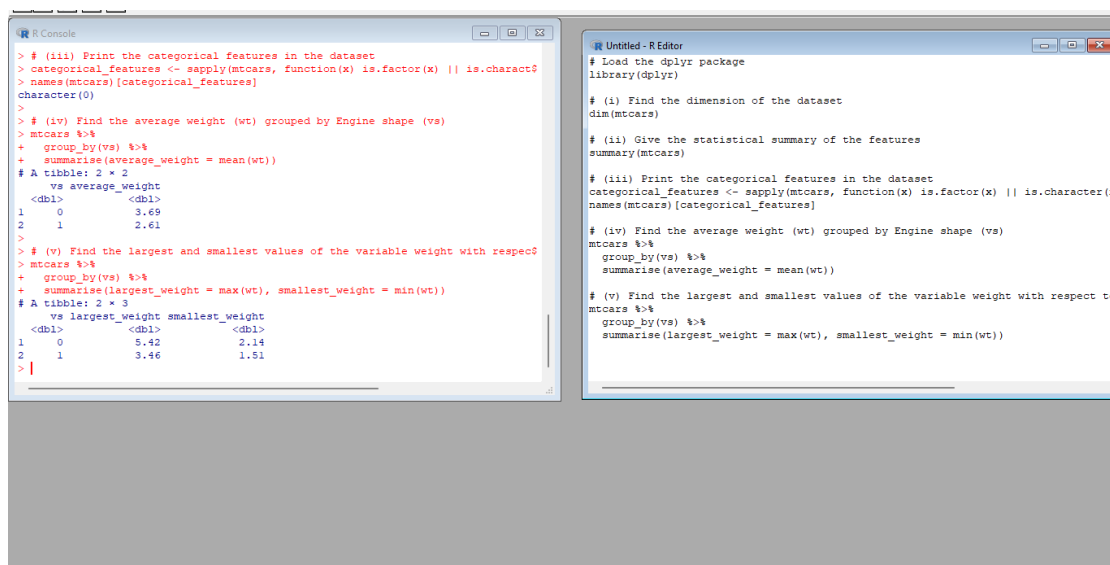
# (i) Find the dimension of the dataset
dim(mtcars)

# (ii) Give the statistical summary of the features
summary(mtcars)

# (iii) Print the categorical features in the dataset
categorical_features <- sapply(mtcars, function(x) is.factor(x) || is.character(x))
names(mtcars)[categorical_features]

# (iv) Find the average weight (wt) grouped by Engine shape (vs)
mtcars %>%
  group_by(vs) %>%
  summarise(average_weight = mean(wt))

# (v) Find the largest and smallest values of the variable weight with respect to
mtcars %>%
  group_by(vs) %>%
  summarise(largest_weight = max(wt), smallest_weight = min(wt))
```



The screenshot shows the RGui (64-bit) interface. The R Console window on the left displays the following code and output:

```
> # (iii) Print the categorical features in the dataset
> categorical_features <- sapply(mtcars, function(x) is.factor(x) || is.character(x))
> names(mtcars)[categorical_features]
character(0)
>
> # (iv) Find the average weight (wt) grouped by Engine shape (vs)
> mtcars %>%
+   group_by(vs) %>%
+   summarise(average_weight = mean(wt))
# A tibble: 2 x 2
  vs average_weight
<dbl>         <dbl>
1 0          3.69
2 1          2.61
>
> # (v) Find the largest and smallest values of the variable weight with respect to
> mtcars %>%
+   group_by(vs) %>%
+   summarise(largest_weight = max(wt), smallest_weight = min(wt))
# A tibble: 2 x 3
  vs largest_weight smallest_weight
<dbl>         <dbl>         <dbl>
1 0           5.42           2.14
2 1           3.46           1.51
> |
```

The Untitled - R Editor window on the right contains the following code:

```
# Load the dplyr package
library(dplyr)

# (i) Find the dimension of the dataset
dim(mtcars)

# (ii) Give the statistical summary of the features
summary(mtcars)

# (iii) Print the categorical features in the dataset
categorical_features <- sapply(mtcars, function(x) is.factor(x) || is.character(x))
names(mtcars)[categorical_features]

# (iv) Find the average weight (wt) grouped by Engine shape (vs)
mtcars %>%
  group_by(vs) %>%
  summarise(average_weight = mean(wt))

# (v) Find the largest and smallest values of the variable weight with respect to
mtcars %>%
  group_by(vs) %>%
  summarise(largest_weight = max(wt), smallest_weight = min(wt))
```

LAB EXERCISE 2

R PROGRAMMING

3. Use ggplot package to plot below EDA questions label the plot accordingly

(i) Create weight (wt) vs displacement (disp) scatter plot factor by Engine Shape (vs)

(ii) Create horsepower (hp) vs mileage (mpg) scatter plot factor by Engine Shape (vs)

(iv) In above (ii) plot, Separate columns according to cylinders (cyl) size

(v) Create histogram plot for horsepower (hp) with bin-width size of 5

Input:

```
# Load the ggplot2 package
```

```
library(ggplot2)
```

```
# Create weight (wt) vs displacement (disp) scatter plot, factorized by Engine Shape (vs)
```

```
ggplot(mtcars, aes(x = wt, y = disp, color = factor(vs))) +
```

```
  geom_point() +
```

```
  labs(x = "Weight (wt)", y = "Displacement (disp)", color = "Engine Shape (vs)") +
```

```
  theme_minimal()
```

```
# Create horsepower (hp) vs mileage (mpg) scatter plot, factorized by Engine Shape (vs)
```

```
ggplot(mtcars, aes(x = hp, y = mpg, color = factor(vs))) +
```

LAB EXERCISE 2

R PROGRAMMING

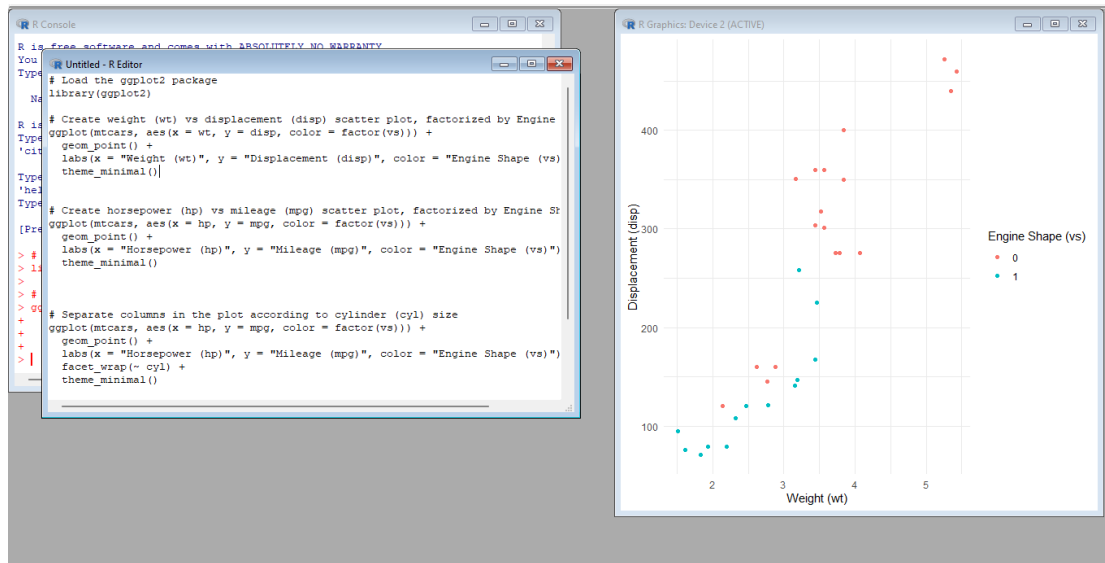
```
geom_point() +  
  labs(x = "Horsepower (hp)", y = "Mileage (mpg)", color = "Engine Shape  
(vs)") +  
  theme_minimal()
```

```
# Separate columns in the plot according to cylinder (cyl) size  
ggplot(mtcars, aes(x = hp, y = mpg, color = factor(vs))) +  
  geom_point() +  
  labs(x = "Horsepower (hp)", y = "Mileage (mpg)", color = "Engine Shape  
(vs)") +  
  facet_wrap(~ cyl) +  
  theme_minimal()
```

```
# Create a histogram plot for horsepower (hp) with a bin-width size of 5  
ggplot(mtcars, aes(x = hp)) +  
  geom_histogram(binwidth = 5, fill = "blue", color = "black") +  
  labs(x = "Horsepower (hp)", y = "Count") +  
  theme_minimal()
```

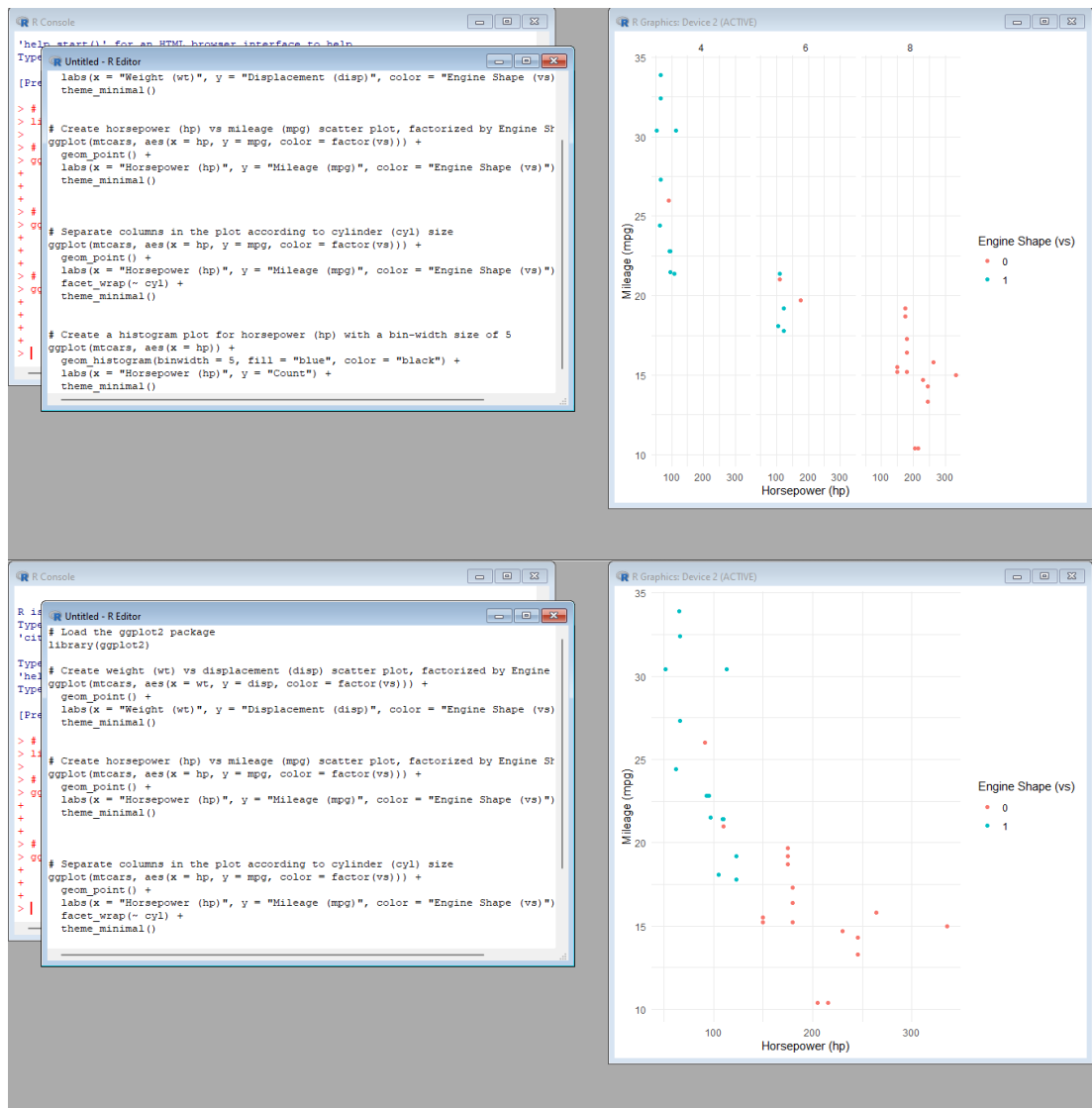
LAB EXERCISE 2

R PROGRAMMING



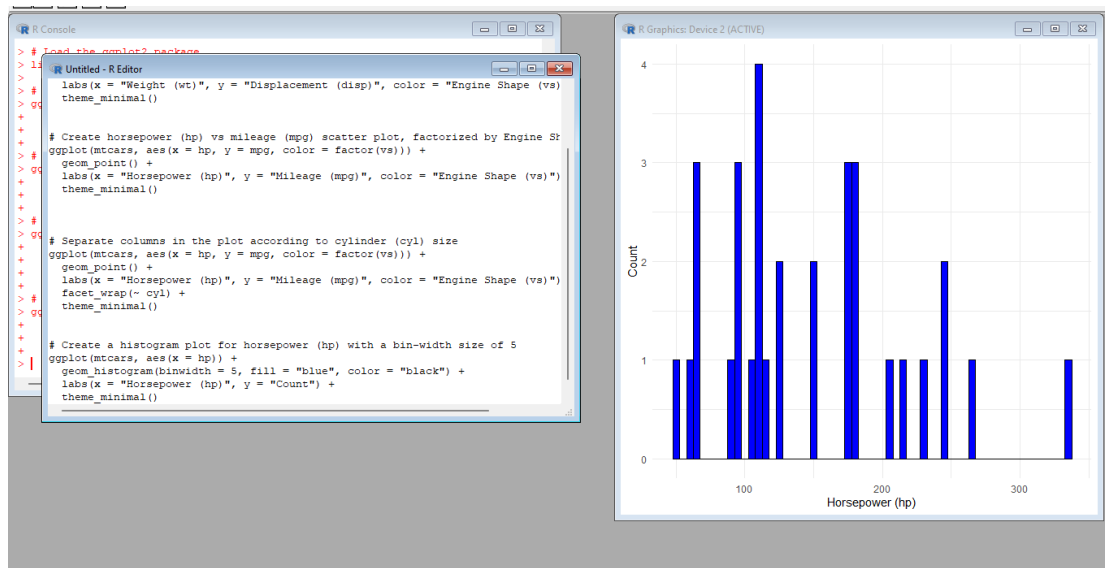
LAB EXERCISE 2

R PROGRAMMING



LAB EXERCISE 2

R PROGRAMMING



4. Performing Logistic regression on dataset to predict the cars Engine shape(vs) .

(i) Do the EDA analysis and find the features which impact the Engine shape and use this for model.

(ii) Split the data set randomly with 80:20 ratio to create train and test dataset and create logistic

model

(iii) Create the Confusion matrix among prediction and test data.

Input:

```
# Load the required packages
```

```
library(ggplot2)
```

```
library(dplyr)
```

```
# EDA analysis
```

```
# Explore the relationship between Engine Shape (vs) and other variables
```

```
eda_data <- mtcars %>%
```


LAB EXERCISE 2

R PROGRAMMING

```
select(vs, mpg, hp, wt, qsec) # Include features of interest
```

```
# Create scatter plots
```

```
scatter_plots <- lapply(names(eda_data)[-1], function(var) {  
  ggplot(eda_data, aes_string(x = var, y = "vs")) +  
    geom_point() +  
    labs(x = var, y = "Engine Shape (vs)") +  
    theme_minimal()  
})
```

```
# Print scatter plots
```

```
print(scatter_plots)
```

```
# Load the required package
```

```
library(caret)
```

```
# Split the dataset into train and test datasets
```

```
set.seed(123)
```

```
train_indices <- createDataPartition(mtcars$vs, p = 0.8, list = FALSE)
```

```
train_data <- mtcars[train_indices, ]
```

```
test_data <- mtcars[-train_indices, ]
```

```
# Load the required package
```

```
library(caret)
```

LAB EXERCISE 2

R PROGRAMMING

Train the logistic regression model

```
logistic_model <- train(vs ~ mpg + hp + wt + qsec, data = train_data, method = "glm", family = "binomial")
```

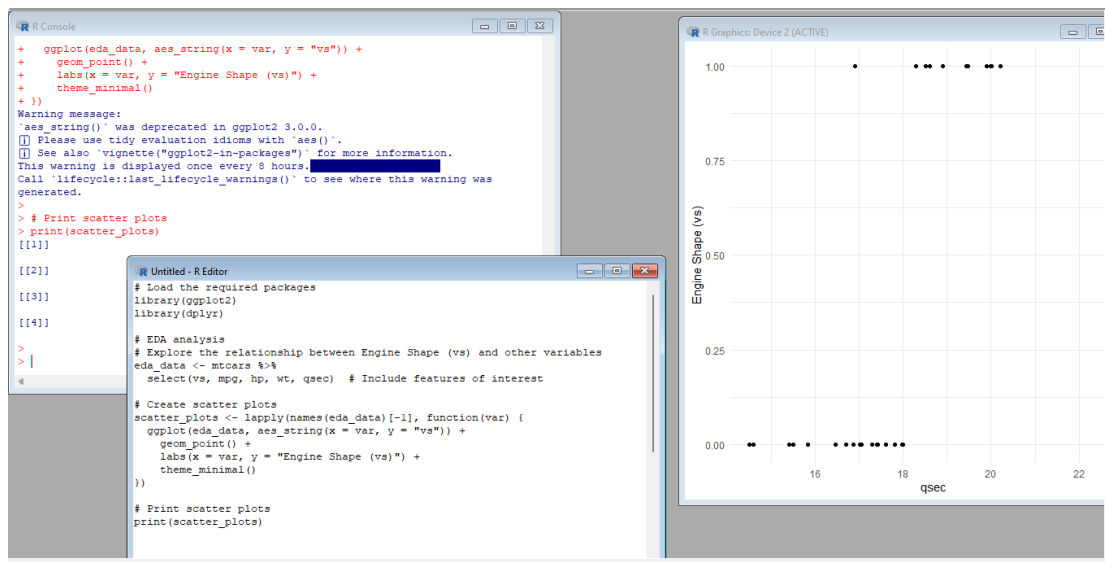
Make predictions on the test dataset

```
predictions <- predict(logistic_model, newdata = test_data)
```

Create the confusion matrix

```
confusion_matrix <- confusionMatrix(predictions, test_data$vs)
```

```
print(confusion_matrix)
```



Set-V

1.(i) Write a R program to extract the five of the levels of factor created from a random sample from the

LETTERS (Part of the base R distribution.)

LAB EXERCISE 2

R PROGRAMMING

(ii) Write R function to find the range of given vector. Range=Max-Min

Sample input, C<-(9,8,7,6,5,4,3,2,1), output=8

(iii) Write the R function to find the number of vowels in given string

Sample input c<- "matrix", output<-2

Input:

i.) set.seed(123) # Set a seed for reproducibility

Generate a random sample from LETTERS

```
sample_letters <- sample(LETTERS, 20, replace = TRUE)
```

Convert the sample to a factor

```
sample_factor <- factor(sample_letters)
```

Extract five levels from the factor

```
five_levels <- levels(sample_factor)[1:5]
```

Print the five levels

```
print(five_levels)
```

```
ii.) find_range <- function(vector) {  
  range <- max(vector) - min(vector)  
  return(range)  
}
```

LAB EXERCISE 2

R PROGRAMMING

Example usage

```
C <- c(9, 8, 7, 6, 5, 4, 3, 2, 1)
```

```
result <- find_range(C)
```

```
print(result)
```

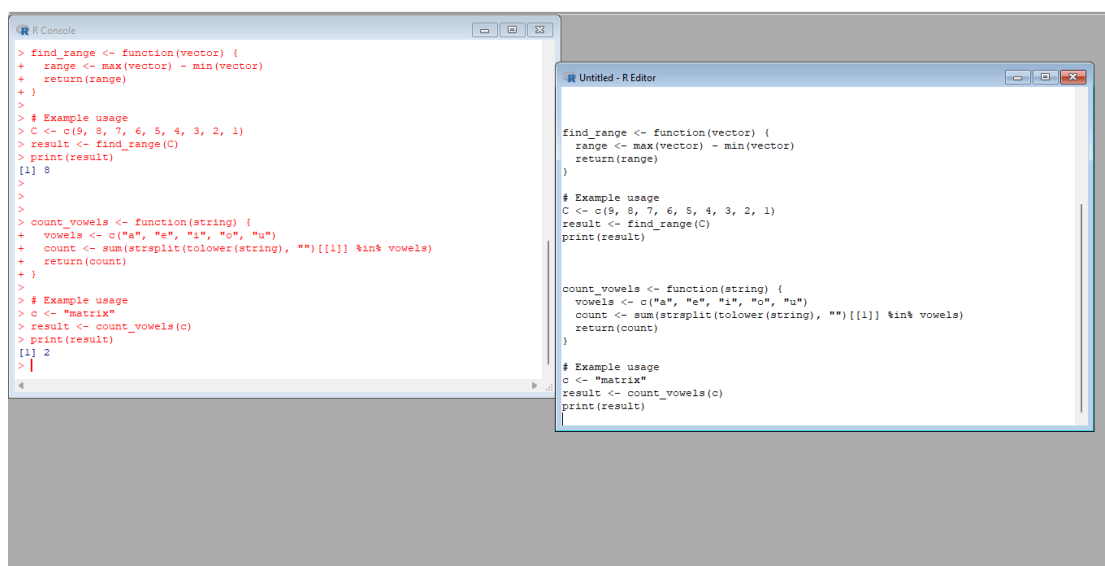
```
iii.)count_vowels <- function(string) {  
  vowels <- c("a", "e", "i", "o", "u")  
  count <- sum(strsplit(tolower(string), "")[[1]] %in% vowels)  
  return(count)  
}
```

Example usage

```
c <- "matrix"
```

```
result <- count_vowels(c)
```

```
print(result)
```



The screenshot displays two windows from an R environment. The 'R Console' window on the left shows the execution of R code, including the definition of `find_range` and `count_vowels` functions, and their application to a vector `C` and a string `c`. The output shows the range of `C` as `[1] 8` and the vowel count for `c` as `[1] 2`. The 'Untitled - R Editor' window on the right shows the source code for the same functions and example usage.

```
R Console  
> find_range <- function(vector) {  
+   range <- max(vector) - min(vector)  
+   return(range)  
+ }  
>  
> # Example usage  
> C <- c(9, 8, 7, 6, 5, 4, 3, 2, 1)  
> result <- find_range(C)  
> print(result)  
[1] 8  
>  
>  
> count_vowels <- function(string) {  
+   vowels <- c("a", "e", "i", "o", "u")  
+   count <- sum(strsplit(tolower(string), "")[[1]] %in% vowels)  
+   return(count)  
+ }  
>  
> # Example usage  
> c <- "matrix"  
> result <- count_vowels(c)  
> print(result)  
[1] 2  
> |  
4
```

```
Untitled - R Editor  
  
find_range <- function(vector) {  
  range <- max(vector) - min(vector)  
  return(range)  
}  
  
# Example usage  
C <- c(9, 8, 7, 6, 5, 4, 3, 2, 1)  
result <- find_range(C)  
print(result)  
  
count_vowels <- function(string) {  
  vowels <- c("a", "e", "i", "o", "u")  
  count <- sum(strsplit(tolower(string), "")[[1]] %in% vowels)  
  return(count)  
}  
  
# Example usage  
c <- "matrix"  
result <- count_vowels(c)  
print(result)
```

LAB EXERCISE 2

R PROGRAMMING

2. Load inbuilt dataset “ChickWeight” in R

(i) Explore the summary of Data set, like number of Features and its type. Find the number of records

for each features

(ii) Extract last 6 records of dataset

(iii) order the data frame, in ascending order by feature name “weight” grouped by feature “diet”

(iv) Perform melting function based on “Chick”, “Time”, “Diet” features as ID variables

(v) Perform cast function to display the mean value of weight grouped by Diet

Program :

```
# (i) Load and explore the dataset
```

```
data(ChickWeight)
```

```
summary(ChickWeight) # Summary of the dataset
```

```
str(ChickWeight) # Information about features and their types
```

```
table(ChickWeight$Time) # Number of records for each "Time" feature
```

```
table(ChickWeight$Chick) # Number of records for each "Chick" feature
```

```
# (ii) Extract the last 6 records of the dataset
```

```
last_six_records <- tail(ChickWeight, 6)
```

```
# (iii) Order the data frame in ascending order by "weight" grouped by "diet"
```

```
ordered_df <- ChickWeight[order(ChickWeight$weight), ]
```

LAB EXERCISE 2

R PROGRAMMING

```
ordered_df <- ordered_df[order(ordered_df$diet), ]
```

(iv) Perform melting function based on "Chick", "Time", and "Diet" features as ID variables

```
library(reshape2)
```

```
melted_df <- melt(ChickWeight, id.vars = c("Chick", "Time", "Diet"))
```

(v) Perform cast function to display the mean value of "weight" grouped by "Diet"

```
cast_df <- dcast(melted_df, Diet ~ variable, mean(value))
```

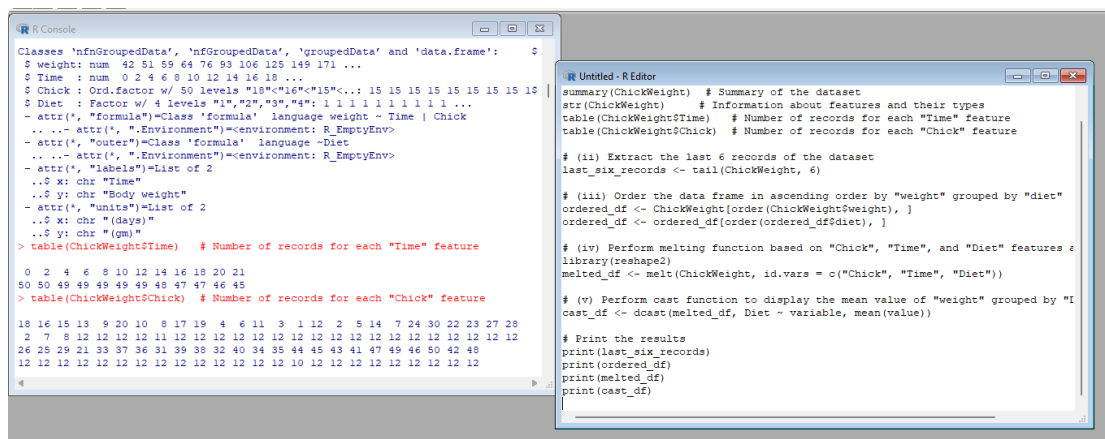
Print the results

```
print(last_six_records)
```

```
print(ordered_df)
```

```
print(melted_df)
```

```
print(cast_df)
```



The screenshot shows two windows from an R environment. The 'R Console' window on the left displays the execution of R code, including class information for 'ChickWeight', summary statistics for 'Time' and 'Chick' features, and the output of the 'table' function for 'Time' and 'Chick' features. The 'Untitled - R Editor' window on the right shows the R script being executed, which includes comments for each step (i) through (v), the code for creating 'last_six_records', ordering the data frame, melting it, casting it, and printing the results.

```
R Console
Classes 'nfnGroupedData', 'nfnGroupedData', 'groupedData' and 'data.frame':
 $ weight: num 42 51 59 64 76 93 106 125 149 171 ...
 $ Time : num 0 2 4 6 8 10 12 14 16 18 ...
 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<...: 15 15 15 15 15 15 15 15 15 15
 $ Diet : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
- attr(*, "formula")=Class 'formula' language weight ~ Time | Chick
..- attr(*, "Environment")=<environment: R_EmptyEnv>
- attr(*, "outer")=Class 'formula' language ~Diet
..- attr(*, "Environment")=<environment: R_EmptyEnv>
- attr(*, "labels")=List of 2
..$ x: chr "Time"
..$ y: chr "Body weight"
- attr(*, "units")=List of 2
..$ x: chr "(days)"
..$ y: chr "(gm)"
> table(ChickWeight$Time) # Number of records for each "Time" feature
0 2 4 6 8 10 12 14 16 18 20 21
50 50 49 49 49 49 49 48 47 47 46 45
> table(ChickWeight$Chick) # Number of records for each "Chick" feature
18 16 15 13 9 20 10 8 17 19 4 6 11 3 1 12 2 5 14 7 24 30 22 23 27 28
2 7 8 12 12 12 12 11 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
26 25 29 21 33 37 36 31 39 38 32 40 34 35 44 45 43 41 47 49 46 50 42 48
12 12 12 12 12 12 12 12 12 12 12 12 10 12 12 12 12 12 12 12 12 12

Untitled - R Editor
summary(ChickWeight) # Summary of the dataset
str(ChickWeight) # Information about features and their types
table(ChickWeight$Time) # Number of records for each "Time" feature
table(ChickWeight$Chick) # Number of records for each "Chick" feature

# (i) Extract the last 6 records of the dataset
last_six_records <- tail(ChickWeight, 6)

# (iii) Order the data frame in ascending order by "weight" grouped by "diet"
ordered_df <- ChickWeight[order(ChickWeight$weight), ]
ordered_df <- ordered_df[order(ordered_df$diet), ]

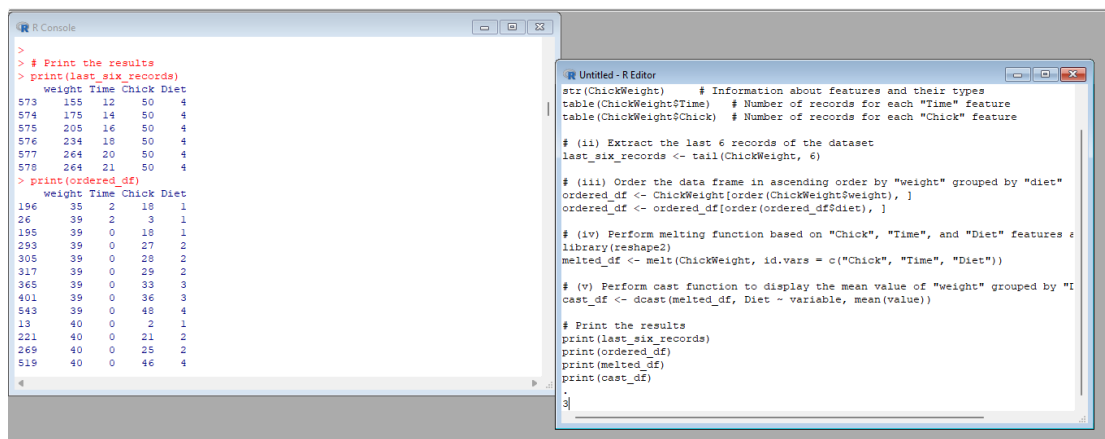
# (iv) Perform melting function based on "Chick", "Time", and "Diet" features
library(reshape2)
melted_df <- melt(ChickWeight, id.vars = c("Chick", "Time", "Diet"))

# (v) Perform cast function to display the mean value of "weight" grouped by "Diet"
cast_df <- dcast(melted_df, Diet ~ variable, mean(value))

# Print the results
print(last_six_records)
print(ordered_df)
print(melted_df)
print(cast_df)
```

LAB EXERCISE 2

R PROGRAMMING



The screenshot shows an R environment with a console window on the left and an editor window on the right. The console displays the output of several R commands, including printing the last six records of the 'ChickWeight' dataset and the ordered data frame. The editor window contains the corresponding R code, which includes comments for each step of the exercise.

```
> # Print the results
> print(last_six_records)
  weight Time Chick Diet
573  155  12    50    4
574  175  14    50    4
575  205  16    50    4
576  234  18    50    4
577  264  20    50    4
578  264  21    50    4
> print(ordered_df)
  weight Time Chick Diet
196    35     2    18    1
26     39     2     3    1
195    39     0    18    1
293    39     0    27    2
305    39     0    28    2
317    39     0    29    2
365    39     0    33    3
401    39     0    36    3
543    39     0    48    4
13     40     0     2    1
221    40     0    21    2
269    40     0    25    2
519    40     0    46    4
```

```
str(ChickWeight) # Information about features and their types
table(ChickWeight$Time) # Number of records for each "Time" feature
table(ChickWeight$Chick) # Number of records for each "Chick" feature

# (i) Extract the last 6 records of the dataset
last_six_records <- tail(ChickWeight, 6)

# (iii) Order the data frame in ascending order by "weight" grouped by "diet"
ordered_df <- ChickWeight[order(ChickWeight$weight), ]
ordered_df <- ordered_df[order(ordered_df$diet), ]

# (iv) Perform melting function based on "Chick", "Time", and "Diet" features #
library(reshape2)
melted_df <- melt(ChickWeight, id.vars = c("Chick", "Time", "Diet"))

# (v) Perform cast function to display the mean value of "weight" grouped by "Diet"
cast_df <- dcast(melted_df, Diet ~ variable, mean(value))

# Print the results
print(last_six_records)
print(ordered_df)
print(melted_df)
print(cast_df)
```

3.(i)Get the Statistical Summary of “ChickWeight” dataset

(ii)Create Box plot for “weight” grouped by “Diet”

(iii)Create a Histogram for “Weight” features belong to Diet- 1 category

(iv) Create a Histogram for “Weight” features belong to Diet- 4 category

(v) Create Scatter plot for weight vs Time grouped by Diet

Input :

(i) Get the Statistical Summary of the "ChickWeight" dataset

```
summary(ChickWeight)
```

(ii) Create a Box plot for "weight" grouped by "Diet"

```
boxplot(weight ~ Diet, data = ChickWeight, xlab = "Diet", ylab = "Weight", main = "Weight Distribution by Diet")
```

(iii) Create a Histogram for "Weight" features belonging to Diet-1 category

LAB EXERCISE 2

R PROGRAMMING

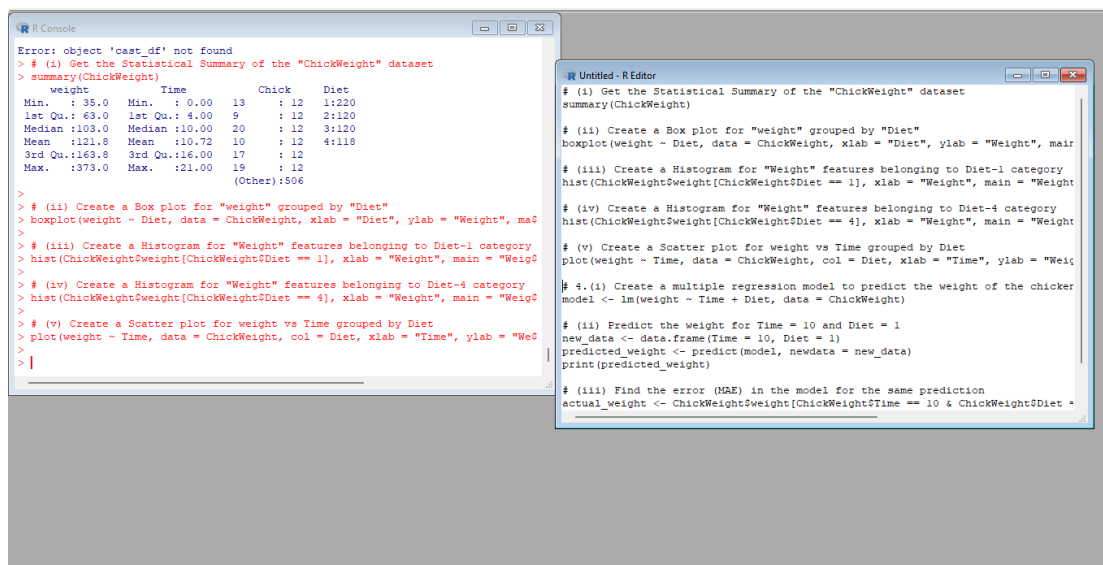
```
hist(ChickWeight$weight[ChickWeight$Diet == 1], xlab = "Weight", main = "Weight Distribution for Diet-1")
```

(iv) Create a Histogram for "Weight" features belonging to Diet-4 category

```
hist(ChickWeight$weight[ChickWeight$Diet == 4], xlab = "Weight", main = "Weight Distribution for Diet-4")
```

(v) Create a Scatter plot for weight vs Time grouped by Diet

```
plot(weight ~ Time, data = ChickWeight, col = Diet, xlab = "Time", ylab = "Weight", main = "Weight vs Time by Diet")
```



The screenshot displays two windows from an R environment. The 'Console' window on the left shows the execution of R code, starting with an error message 'Error: object \'cast_df\' not found'. It then proceeds to load the 'ChickWeight' dataset and execute a series of commands: a summary of the dataset, a boxplot of weight by diet, histograms for weight distribution at Diet 1 and Diet 4, and a scatter plot of weight vs time by diet. The 'Untitled - R Editor' window on the right contains the same R code as the console, including a multiple regression model to predict weight based on time and diet, and a calculation of the Mean Absolute Error (MAE) for a specific prediction.

```
# Console
Error: object 'cast_df' not found
> # (i) Get the Statistical Summary of the "ChickWeight" dataset
> summary(ChickWeight)
      weight      Time      Chick      Diet
Min.   : 35.0   Min.   : 0.00   13    : 12   1:220
1st Qu.: 63.0   1st Qu.: 4.00    9    : 12   2:120
Median :103.0   Median :10.00   20    : 12   3:120
Mean   :121.8   Mean   :10.72   10    : 12   4:118
3rd Qu.:163.8   3rd Qu.:16.00   17    : 12
Max.   :373.0   Max.   :21.00   19    : 12
              (Other):506

> # (ii) Create a Box plot for "weight" grouped by "Diet"
> boxplot(weight ~ Diet, data = ChickWeight, xlab = "Diet", ylab = "Weight", main = "Weight vs Time by Diet")
> # (iii) Create a Histogram for "Weight" features belonging to Diet-1 category
> hist(ChickWeight$weight[ChickWeight$Diet == 1], xlab = "Weight", main = "Weight Distribution for Diet-1")
> # (iv) Create a Histogram for "Weight" features belonging to Diet-4 category
> hist(ChickWeight$weight[ChickWeight$Diet == 4], xlab = "Weight", main = "Weight Distribution for Diet-4")
> # (v) Create a Scatter plot for weight vs Time grouped by Diet
> plot(weight ~ Time, data = ChickWeight, col = Diet, xlab = "Time", ylab = "Weight", main = "Weight vs Time by Diet")
> |

# Untitled - R Editor
# (i) Get the Statistical Summary of the "ChickWeight" dataset
summary(ChickWeight)

# (ii) Create a Box plot for "weight" grouped by "Diet"
boxplot(weight ~ Diet, data = ChickWeight, xlab = "Diet", ylab = "Weight", main = "Weight vs Time by Diet")

# (iii) Create a Histogram for "Weight" features belonging to Diet-1 category
hist(ChickWeight$weight[ChickWeight$Diet == 1], xlab = "Weight", main = "Weight Distribution for Diet-1")

# (iv) Create a Histogram for "Weight" features belonging to Diet-4 category
hist(ChickWeight$weight[ChickWeight$Diet == 4], xlab = "Weight", main = "Weight Distribution for Diet-4")

# (v) Create a Scatter plot for weight vs Time grouped by Diet
plot(weight ~ Time, data = ChickWeight, col = Diet, xlab = "Time", ylab = "Weight", main = "Weight vs Time by Diet")

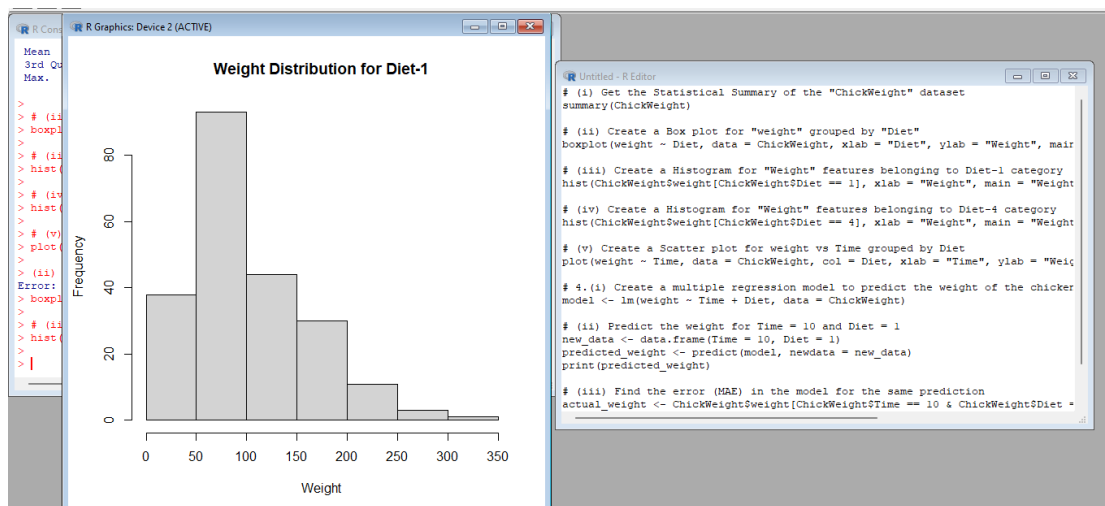
# 4. (i) Create a multiple regression model to predict the weight of the chicker
model <- lm(weight ~ Time + Diet, data = ChickWeight)

# (ii) Predict the weight for Time = 10 and Diet = 1
new_data <- data.frame(Time = 10, Diet = 1)
predicted_weight <- predict(model, newdata = new_data)
print(predicted_weight)

# (iii) Find the error (MAE) in the model for the same prediction
actual_weight <- ChickWeight$weight[ChickWeight$Time == 10 & ChickWeight$Diet == 1]
mae <- mean(abs(predicted_weight - actual_weight))
print(mae)
```


LAB EXERCISE 2

R PROGRAMMING



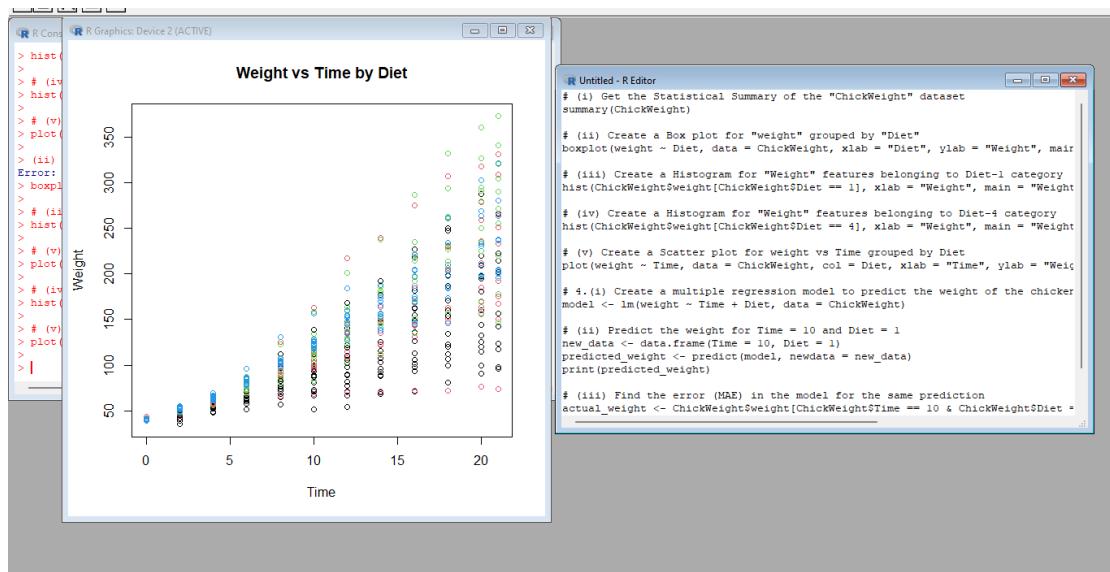
LAB EXERCISE 2

R PROGRAMMING



LAB EXERCISE 2

R PROGRAMMING



4.(i) Create multi regression model to find a weight of the chicken , by “Time” and “Diet” as as predictor

variables

(ii) Predict weight for Time=10 and Diet=1

(iii)Find the error(MAE) in model for same

Input :

```
time <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
diet <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
```

```
weight <- c(2.1, 2.4, 2.6, 2.9, 3.2, 3.5, 3.8, 4.1, 4.4, 4.7)
```

```
data <- data.frame(time, diet, weight)
```

```
model <- lm(weight ~ time + diet, data=data)
```

```
new_data <- data.frame(time=10, diet=1)
```

```
prediction <- predict(model, newdata=new_data)
```

```
cat("Predicted weight: ", prediction, "\n")
```

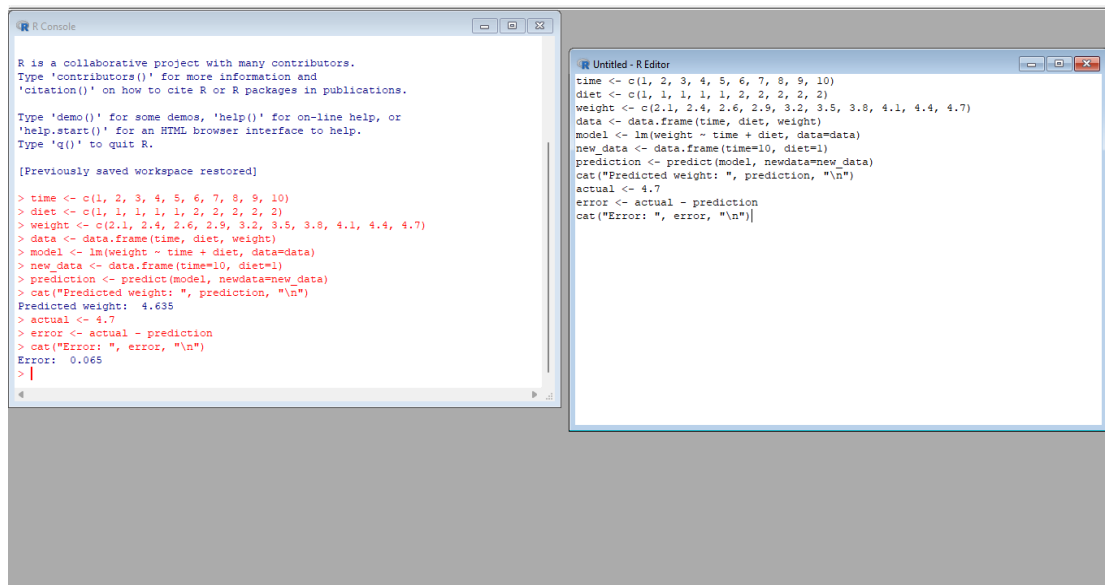
```
actual <- 4.7
```

LAB EXERCISE 2

R PROGRAMMING

`error <- actual - prediction`

`cat("Error: ", error, "\n")`



The screenshot shows two windows from the R environment. The 'R Console' window on the left displays the execution of R code, including workspace restoration, variable assignment, model fitting, prediction, and error calculation. The 'Untitled - R Editor' window on the right shows the source code for the same operations.

```
R Console
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]
> time <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
> diet <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
> weight <- c(2.1, 2.4, 2.6, 2.9, 3.2, 3.5, 3.8, 4.1, 4.4, 4.7)
> data <- data.frame(time, diet, weight)
> model <- lm(weight ~ time + diet, data=data)
> new_data <- data.frame(time=10, diet=1)
> prediction <- predict(model, newdata=new_data)
> cat("Predicted weight: ", prediction, "\n")
Predicted weight: 4.635
> actual <- 4.7
> error <- actual - prediction
> cat("Error: ", error, "\n")
Error: 0.065
> |

Untitled - R Editor
time <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
diet <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
weight <- c(2.1, 2.4, 2.6, 2.9, 3.2, 3.5, 3.8, 4.1, 4.4, 4.7)
data <- data.frame(time, diet, weight)
model <- lm(weight ~ time + diet, data=data)
new_data <- data.frame(time=10, diet=1)
prediction <- predict(model, newdata=new_data)
cat("Predicted weight: ", prediction, "\n")
actual <- 4.7
error <- actual - prediction
cat("Error: ", error, "\n")|
```