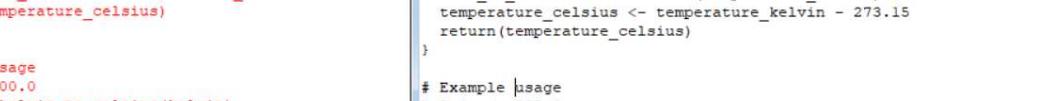


Set-I

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)

## SOURCE CODE:

```
kelvin_to_celsius <- function(temperature_kelvin) {  
  temperature_celsius <- temperature_kelvin - 273.15  
  return(temperature_celsius)  
}  
  
# Example usage  
  
kelvin <- 300.0  
  
celsius <- kelvin_to_celsius(kelvin)  
  
print(c(celsius))
```



```
> kelvin_to_celsius <- function(temperature_kelvin) {  
+   temperature_celsius <- temperature_kelvin - 273.15  
+   return(temperature_celsius)  
+ }  
>  
> # Example usage  
> kelvin <- 300.0  
> celsius <- kelvin_to_celsius(kelvin)  
> print(c(celsius))  
[1] 26.85  
>
```

E:\3-3\R\lab\New folder\day4 p3\lab practice\L2-11.R - R Editor

```
kelvin_to_celsius <- function(temperature_kelvin) {  
  temperature_celsius <- temperature_kelvin - 273.15  
  return(temperature_celsius)  
}  
  
# Example usage  
kelvin <- 300.0  
celsius <- kelvin_to_celsius(kelvin)  
print(c(celsius))
```

- (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)

#### SOURCE CODE:

```
# Define the data  
  
values <- c(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)  
  
# Compute the mean  
  
mean_value <- mean(values)
```

```

# Compute the median
median_value <- median(values)

# Compute the mode
mode_value <- as.numeric(names(table(values))[table(values) == max(table(values))])

# Print the results
print(paste("Mean:", mean_value))
print(paste("Median:", median_value))
print(paste("Mode:", mode_value))

```

The screenshot shows an RStudio interface with two panes. The left pane contains the R code provided above. The right pane shows the R console output:

```

# Define the data
values <- c(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)

# Compute the mean
mean_value <- mean(values)

# Compute the median
median_value <- median(values)

# Compute the mode
mode_value <- as.numeric(names(table(values))[table(values) == max(table(values))])

# Print the results
print(paste("Mean:", mean_value))
[1] "Mean: 60"
print(paste("Median:", median_value))
[1] "Median: 70"
print(paste("Mode:", mode_value))
[1] "Mode: 20" "Mode: 70" "Mode: 80" "Mode: 90"

```

(iii) Write R code to find 2<sup>nd</sup> highest and 3<sup>rd</sup> Lowest value of above problem.

### SOURCE CODE:

```

# Define the data
values <- c(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)

# Sort the values in ascending order
sorted_values <- sort(values)

# Find the 2nd highest value

```

```

second_highest <- sorted_values[length(sorted_values) - 1]

# Find the 3rd lowest value

third_lowest <- sorted_values[3]

# Print the results

print(paste("2nd Highest:", second_highest))

print(paste("3rd Lowest:", third_lowest))

```

The screenshot shows an RStudio session window titled "E:\3-3\R\lab\New folder\day4 p3\lab practice\L2-1iii.R - R Editor". The code in the editor is identical to the one above. The console area shows the execution of the script, including the assignment of `sorted\_values` to a sorted vector of 20 values, and the output of the `print` statements:

```

> # Sort the values in ascending order
> sorted_values <- sort(values)
>
> # Find the 2nd highest value
> second_highest <- sorted_values[length(sorted_values) - 1]
>
> # Find the 3rd lowest value
> third_lowest <- sorted_values[3]
>
> # Print the results
> print(paste("2nd Highest:", second_highest))
[1] "2nd Highest: 90"
> print(paste("3rd Lowest:", third_lowest))
[1] "3rd Lowest: 30"
> |

```

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)
    - 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?

#### SOURCE CODE:

```
# Load the airquality dataset
```

```
data(airquality)

# i. Compute the mean temperature (without using built-in function)
mean_temperature <- sum(airquality$Temp) / length(airquality$Temp)
print(paste("Mean Temperature:", mean_temperature))

# ii. Extract the first five rows from airquality
first_five_rows <- airquality[1:5, ]
print("First five rows of airquality:")
print(first_five_rows)

# iii. Extract all columns from airquality except Temp and Wind
selected_columns <- airquality[, !(names(airquality) %in% c("Temp", "Wind"))]
print("Selected columns from airquality:")
print(selected_columns)

# iv. Determine the coldest day during the period
coldest_day <- airquality$Day[which.min(airquality$Temp)]
print(paste("Coldest Day:", coldest_day))

# v. Count the number of days with wind speed greater than 17 mph
wind_gt_17_count <- sum(airquality$Wind > 17, na.rm = TRUE)
print(paste("Number of Days with Wind Speed > 17 mph:", wind_gt_17_count))
```

The image shows two R console windows. The left window is titled 'R Console' and contains R code for tasks i-v. The right window is titled 'E:\3-3\Lab\New folder\day4 p3\lab practice\L2-2.R - R Editor' and contains the same R code with some parts highlighted in blue.

```

> # i. Compute the mean temperature (without using built-in function)
> mean_temperature <- sum(airquality$Temp) / length(airquality$Temp)
> print(paste("Mean Temperature:", mean_temperature))
[1] "Mean Temperature: 77.8823529411765"
>
> # ii. Extract the first five rows from airquality
> first_five_rows <- airquality[1:5, ]
> print("First five rows of airquality:")
[1] "First five rows of airquality:"
> print(first_five_rows)
   Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67      5    1
2    36     118  8.0   72      5    2
3    12     149 12.6   74      5    3
4    18     313 11.5   62      5    4
5    NA     NA 14.3   56      5    5
>
> # iii. Extract all columns from airquality except Temp and Wind
> selected_columns <- airquality[, !(names(airquality) %in% c("Temp", "Wind"))]
> print("Selected columns from airquality:")
[1] "Selected columns from airquality:"
> print(head(selected_columns))
   Ozone Solar.R Month Day
1    41     190      5    1
2    36     118      5    2
3    12     149      5    3
4    18     313      5    4
5    NA     NA       5    5
6    28     NA       5    6
>
> # iv. Determine the coldest day during the period
> coldest_day <- airquality$Day[which.min(airquality$Temp)]
> print(paste("Coldest Day:", coldest_day))
[1] "Coldest Day: 5"
>
> # v. Count the number of days with wind speed greater than 17 mph
> wind_gt_17_count <- sum(airquality$Wind > 17, na.rm = TRUE)
> print(paste("Number of Days with Wind Speed > 17 mph:", wind_gt_17_count))
[1] "Number of Days with Wind Speed > 17 mph: 3"
>

```

```

# Load the airquality dataset
data(airquality)

# i. Compute the mean temperature (without using built-in function)
mean_temperature <- sum(airquality$Temp) / length(airquality$Temp)
print(paste("Mean Temperature:", mean_temperature))

# ii. Extract the first five rows from airquality
first_five_rows <- airquality[1:5, ]
print("First five rows of airquality:")
print(first_five_rows)

# iii. Extract all columns from airquality except Temp and Wind
selected_columns <- airquality[, !(names(airquality) %in% c("Temp", "Wind"))]
print("Selected columns from airquality:")
print(head(selected_columns))

# iv. Determine the coldest day during the period
coldest_day <- airquality$Day[which.min(airquality$Temp)]
print(paste("Coldest Day:", coldest_day))

# v. Count the number of days with wind speed greater than 17 mph
wind_gt_17_count <- sum(airquality$Wind > 17, na.rm = TRUE)
print(paste("Number of Days with Wind Speed > 17 mph:", wind_gt_17_count))

```

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?

#### SOURCE CODE:

```

# Load the airquality dataset
data(airquality)

# (i) Get the Summary Statistics of airquality dataset
summary_stats <- summary(airquality)
print("Summary Statistics of airquality dataset:")
print(summary_stats)

# (ii) Melt airquality dataset and display as long-format data
melted_data <- melt(airquality)
print("Melted airquality dataset (long-format):")
print(head(melted_data))

# (iii) Melt airquality data and specify month and day as ID variables

```

```

melted_data_id <- melt(airquality, id.vars = c("Month", "Day"))
print("Melted airquality dataset with Month and Day as ID variables:")
print(head(melted_data_id))

```

```

# (iv) Cast the molten airquality dataset with respect to month and day features
casted_data <- dcast(melted_data_id, Month + Day ~ variable)
print("Casted airquality dataset:")
print(head(casted_data))

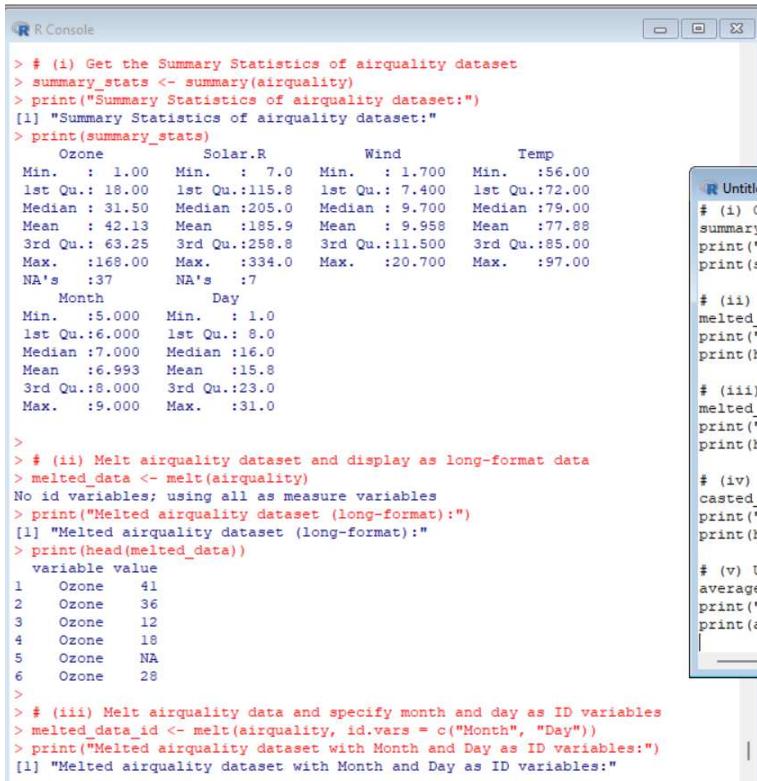
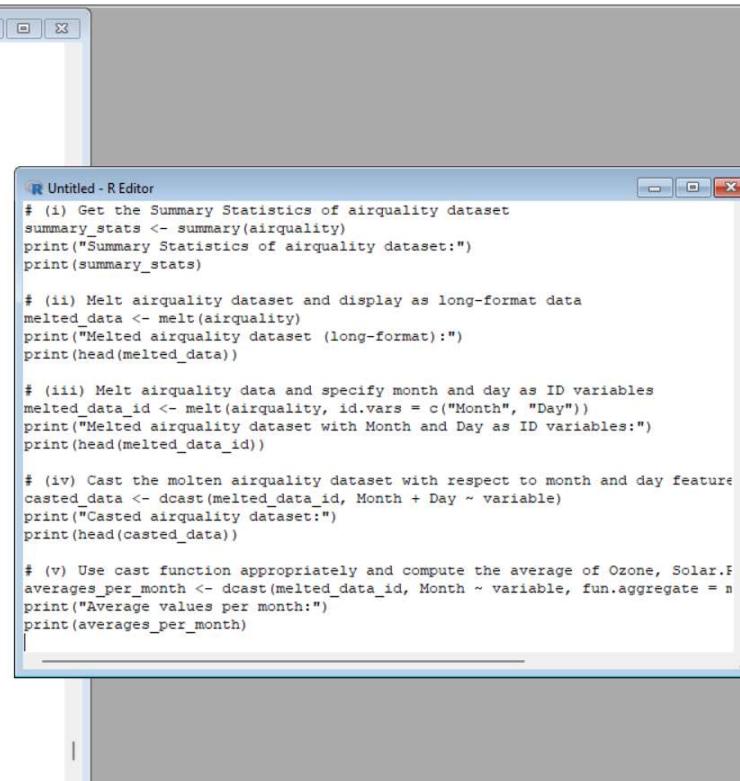
```

# (v) Use cast function appropriately and compute the average of Ozone, Solar.R, Wind, and Temperature per month

```

averages_per_month <- dcast(melted_data_id, Month ~ variable, fun.aggregate = mean)
print("Average values per month:")
print(averages_per_month)

```

```

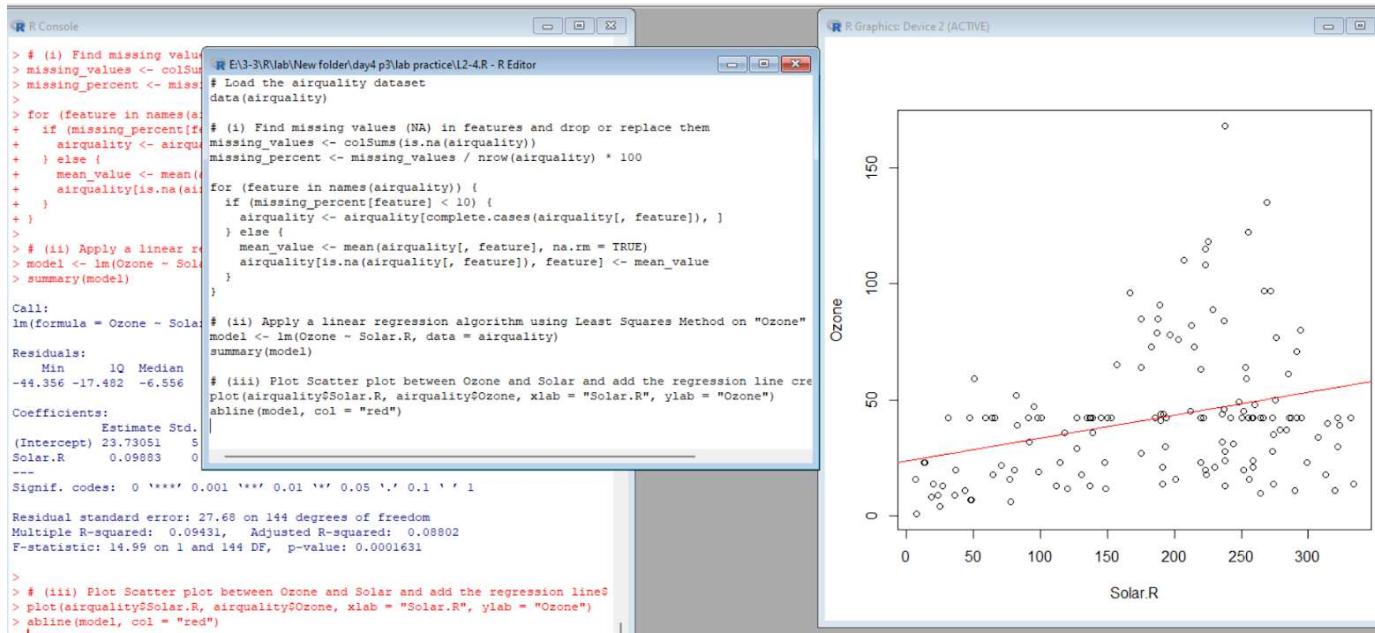
R Console
> # (i) Get the Summary Statistics of airquality dataset
> summary_stats <- summary(airquality)
> print("Summary Statistics of airquality dataset:")
[1] "Summary Statistics of airquality dataset:"
> print(summary_stats)
   Ozone      Solar.R      Wind      Temp
Min.   : 1.00  Min.   : 7.0  Min.   : 1.700  Min.   :56.00
1st Qu.: 18.00 1st Qu.:115.8 1st Qu.: 7.400 1st Qu.:72.00
Median : 31.50 Median :205.0 Median : 9.700 Median :79.00
Mean   : 42.13 Mean   :185.9 Mean   : 9.958 Mean   :77.88
3rd Qu.: 63.25 3rd Qu.:258.8 3rd Qu.:11.500 3rd Qu.:85.00
Max.   :168.00 Max.   :334.0 Max.   :20.700 Max.   :97.00
NA's    :37    NA's    :7

Month          Day
Min.   :5.000  Min.   : 1.0
1st Qu.:6.000 1st Qu.: 8.0
Median :7.000  Median :16.0
Mean   :6.993  Mean   :15.8
3rd Qu.:8.000 3rd Qu.:23.0
Max.   :9.000  Max.   :31.0

>
> # (ii) Melt airquality dataset and display as long-format data
> melted_data <- melt(airquality)
No id variables; using all as measure variables
> print("Melted airquality dataset (long-format):")
[1] "Melted airquality dataset (long-format):"
> print(head(melted_data))
  variable value
1   Ozone   41
2   Ozone   36
3   Ozone   12
4   Ozone   18
5   Ozone   NA
6   Ozone   28
>
> # (iii) Melt airquality data and specify month and day as ID variables
> melted_data_id <- melt(airquality, id.vars = c("Month", "Day"))
> print("Melted airquality dataset with Month and Day as ID variables:")
[1] "Melted airquality dataset with Month and Day as ID variables:"
```

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 Set-II



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

### SOURCE CODE:

```

factorial <- function(n) {
  result <- 1
  for (i in 1:n) {
    result <- result * i
  }
  return(result)
}

# Example usage
num <- 5
factorial_num <- factorial(num)
print(paste("Factorial of", num, "is", factorial_num))

```

```

> abline(model, col = "red")
> factorial <- function(n) {
+   result <- 1
+   for (i in 1:n) {
+     result <- result * i
+   }
+   return(result)
+ }
>
> # Example usage
> num <- 5
> factorial_num <- factorial(num)
> print(paste("Factorial of", num, "is", factorial_num))
[1] "Factorial of 5 is 120"
> |

```

(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.

#### SOURCE CODE:

```

# Create a matrix with 12 random numbers between 1-100, filled row-by-row
set.seed(123) # For reproducibility
matrix_data <- matrix(sample(1:100, 12, replace = TRUE), nrow = 3, ncol = 4, byrow = TRUE)
# Name the columns and rows
colnames(matrix_data) <- c("uno", "dos", "tres", "cuatro")
rownames(matrix_data) <- c("x", "y", "z")
# Scale the matrix by 10
scaled_matrix <- matrix_data * 10
# Print the scaled matrix
print("Scaled Matrix:")
print(scaled_matrix)

```

```

"-----"
colnames(matrix_data) <- c("uno", "dos",
rownames(matrix_data) <- c("x", "y", "z")

# Scale the matrix by 10
scaled_matrix <- matrix_data * 10

# Print the scaled matrix
print("Scaled Matrix:")
1] "Scaled Matrix:"
print(scaled_matrix)
uno dos tres cuatro
310 790 510 140
670 420 500 430
140 250 900 910
|

```

```

E:\3\3\R\lab\New folder\day4 p3\lab practice\2matrixscaling.R - R Editor
# Create a matrix with 12 random numbers between 1-100, filled row-by-row
set.seed(123) # For reproducibility
matrix_data <- matrix(sample(1:100, 12, replace = TRUE), nrow = 3, ncol = 4, byrow = TRUE)

# Name the columns and rows
colnames(matrix_data) <- c("uno", "dos", "tres", "cuatro")
rownames(matrix_data) <- c("x", "y", "z")

# Scale the matrix by 10
scaled_matrix <- matrix_data * 10

# Print the scaled matrix
print("Scaled Matrix:")
print(scaled_matrix)

```

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

#### SOURCE CODE:

```
# Extract the column called "uno" from the original matrix
```

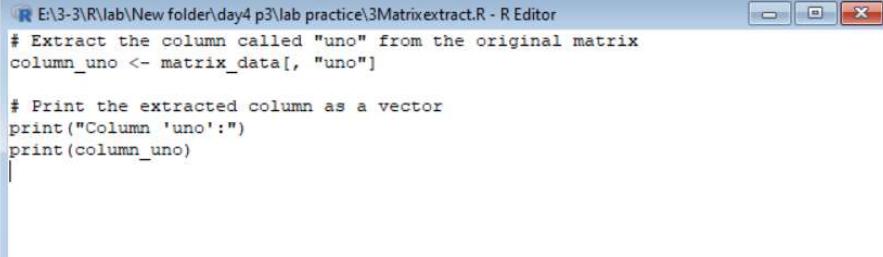
```
column_uno <- matrix_data[, "uno"]
```

```
# Print the extracted column as a vector
```

```
print("Column 'uno':")
```

```
print(column_uno)
```

```
> # Extract the column called "uno" from the original matrix
> column_uno <- matrix_data[, "uno"]
>
> # Print the extracted column as a vector
> print("Column 'uno':")
[1] "Column 'uno':"
> print(column_uno)
x  y  z
31 67 14
> |
```



The screenshot shows the RStudio interface with the code in the top pane and its output in the bottom pane. The output shows the command `print(column\_uno)` followed by the resulting vector `x y z` with values 31, 67, and 14 respectively.

```
R E:\3-3\R\lab\New folder\day4 p3\lab practice\3Matrixextract.R - R Editor
# Extract the column called "uno" from the original matrix
column_uno <- matrix_data[, "uno"]

# Print the extracted column as a vector
print("Column 'uno':")
print(column_uno)
```

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

(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.

## SOURCE CODE:

```
# Load the iris dataset
```

```
data(iris)
```

```
# (i) Find dimension, structure, summary statistics, and standard deviation of all features
```

```
print("Dimension:")
```

```
print(dim(iris))
```

```
print("Structure:")
```

```
print(str(iris))

print("Summary Statistics:")

print(summary(iris))

print("Standard Deviation:")

print(sapply(iris, sd))

# (ii) Find mean and standard deviation of features grouped by three species of Iris flowers

grouped_mean <- aggregate(. ~ Species, data = iris, FUN = mean)

grouped_sd <- aggregate(. ~ Species, data = iris, FUN = sd)

print("Mean values grouped by species:")

print(grouped_mean)

print("Standard deviation grouped by species:")

print(grouped_sd)

# (iii) Find quantile value of sepal width and length

sepal_width_quantile <- quantile(iris$Sepal.Width)

sepal_length_quantile <- quantile(iris$Sepal.Length)

print("Quantile values of sepal width:")

print(sepal_width_quantile)

print("Quantile values of sepal length:")

print(sepal_length_quantile)

# (iv) Create a new data frame named iris1 with a new column named Sepal.Length.Cate that
# categorizes "Sepal.Length" by quantile

iris1 <- iris
```

```

iris1$Sepal.Length.Cate <- cut(iris1$Sepal.Length, breaks = sepal_length_quantile)

print("New data frame iris1:")

print(head(iris1))

```

# (v) Average value of numerical variables by two categorical variables: Species and Sepal.Length.Cate

```
averages <- aggregate(. ~ Species + Sepal.Length.Cate, data = iris1, FUN = mean)
```

```
print("Average values by Species and Sepal.Length.Cate:")
```

```
print(averages)
```

The screenshot shows the R Console window with two panes. The left pane contains R code and its output. The right pane shows the results of the code execution.

```

R Console
2 versicolor      5.936      2.770      4.260      1.326
3 virginica       6.588      2.974      5.552      2.026
> print("Standard deviation grouped by species:")
[1] "Standard deviation grouped by species:"
> print(grouped_sd)
   Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1  setosa    0.3524897  0.3790644  0.1736640  0.1053856
2 versicolor   0.5161711  0.3137983  0.4699110  0.1977527
3 virginica    0.6358796  0.3224966  0.5518947  0.2746501
>
> # (iii) Find quantile value of sepal width and length
> sepal_width_quantile <- quantile(iris$Sepal.Width)
> sepal_length_quantile <- quantile(iris$Sepal.Length)
> print("Quantile values of sepal width:")
[1] "Quantile values of sepal width:"
> print(sepal_width_quantile)
  0% 25% 50% 75% 100%
2.0 2.8 3.0 3.3 4.4
> print("Quantile values of sepal length:")
[1] "Quantile values of sepal length:"
> print(sepal_length_quantile)
  0% 25% 50% 75% 100%
4.3 5.1 5.8 6.4 7.9
>
> # (iv) Create a new data frame named iris1 with a new column named Sepal.Length.Cate
> iris1 <- iris
> iris1$Sepal.Length.Cate <- cut(iris1$Sepal.Length, breaks = sepal_length_quantile)
> print("New data frame iris1:")
[1] "New data frame iris1:"
> print(head(iris1))
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Length.Cate
1          5.1        3.5         1.4        0.2   setosa     (4.3,5.1]
2          4.9        3.0         1.4        0.2   setosa     (4.3,5.1]
3          4.7        3.2         1.3        0.2   setosa     (4.3,5.1]
4          4.6        3.1         1.5        0.2   setosa     (4.3,5.1]
5          5.0        3.6         1.4        0.2   setosa     (4.3,5.1]
6          5.4        3.9         1.7        0.4   setosa     (5.1,5.8]
>
> # (v) Average value of numerical variables by two categorical variables: Species
> averages <- aggregate(. ~ Species + Sepal.Length.Cate, data = iris1, FUN = mean)

```

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

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

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

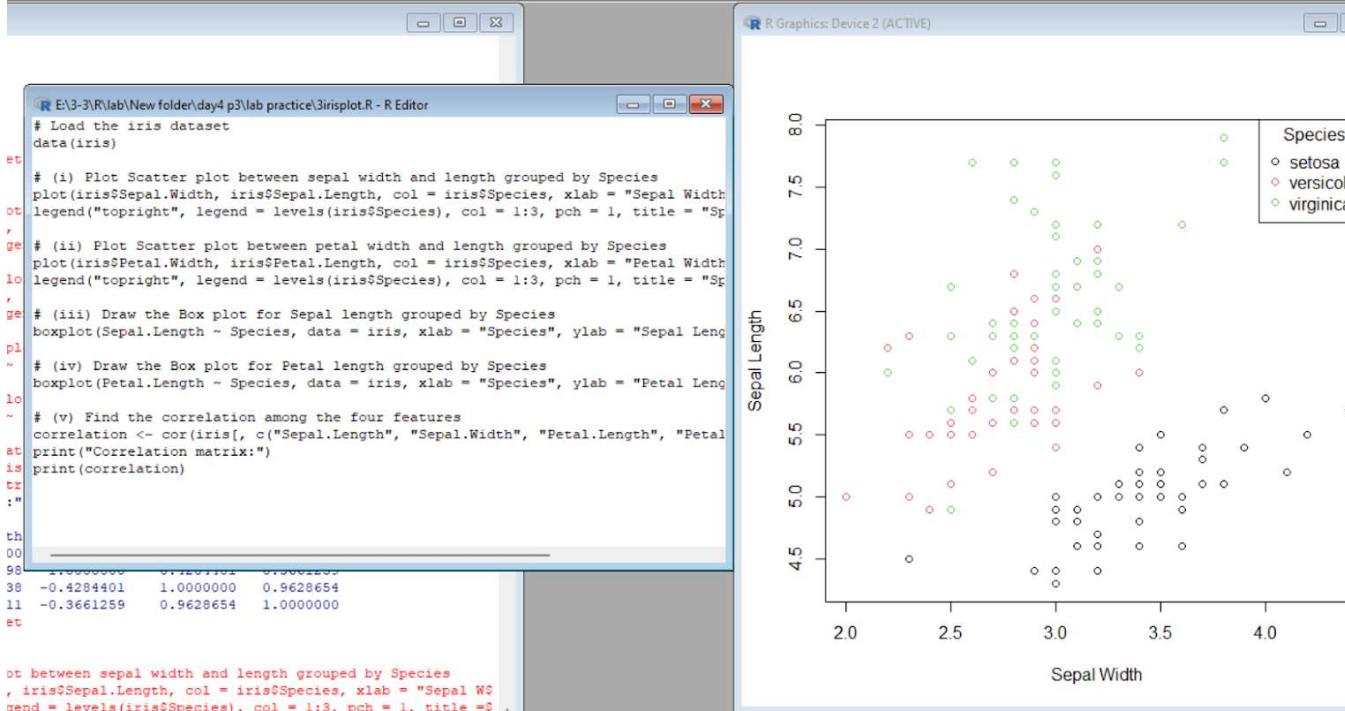
(v) Find the correlation among the four features

### SOURCE CODE:

```
# Load the iris dataset
```

```
data(iris)
```

```
# (i) Plot Scatter plot between sepal width and length grouped by Species  
plot(iris$Sepal.Width, iris$Sepal.Length, col = iris$Species, xlab = "Sepal Width", ylab = "Sepal Length")  
legend("topright", legend = levels(iris$Species), col = 1:3, pch = 1, title = "Species")
```



```
# (ii) Plot Scatter plot between petal width and length grouped by Species  
plot(iris$Petal.Width, iris$Petal.Length, col = iris$Species, xlab = "Petal Width", ylab = "Petal Length")  
legend("topright", legend = levels(iris$Species), col = 1:3, pch = 1, title = "Species")
```

```

et
ot E:\3-3\R\lab\New folder\day4 p3\lab practice\3irisplot.R - R Editor
# Load the iris dataset
ge
data(iris)

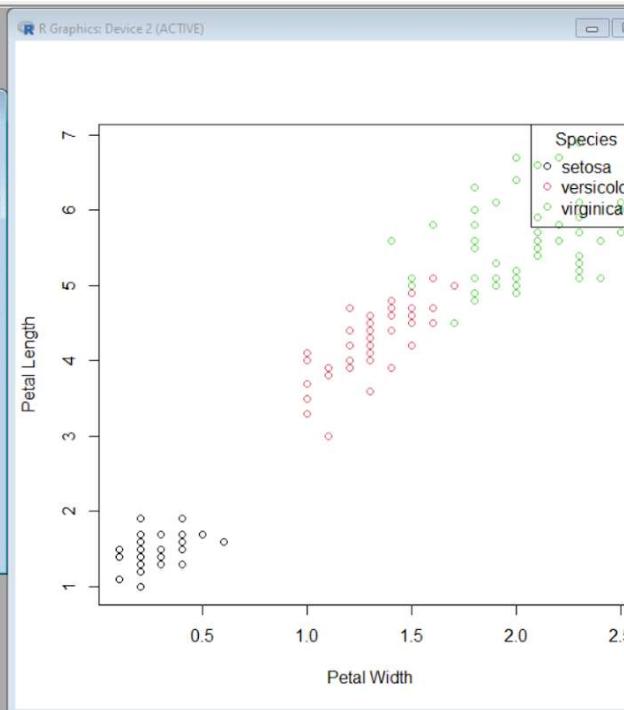
lo # (ii) Plot Scatter plot between petal width and length grouped by Species
, plot(iris$Petal.Width, iris$Petal.Length, col = iris$Species, xlab = "Petal Width"
ge legend("topright", legend = levels(iris$Species), col = 1:3, pch = 1, title = "Sp
pl # (iii) Draw the Box plot for Sepal length grouped by Species
~ boxplot(Sepal.Length ~ Species, data = iris, xlab = "Species", ylab = "Sepal Length"
lo # (iv) Draw the Box plot for Petal length grouped by Species
~ boxplot(Petal.Length ~ Species, data = iris, xlab = "Species", ylab = "Petal Length"
at # (v) Find the correlation among the four features
is correlation <- cor(iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal
tr print("Correlation matrix:")
:"

th
00
98
38
11
et

ot between Sepal Width and Length grouped by species
, iris$Sepal.Length, col = iris$Species, xlab = "Sepal W
gend = levels(iris$Species), col = 1:3, pch = 1, title = $t
et

lot between petal width and length grouped by Species
, iris$Petal.Length, col = iris$Species, xlab = "Petal W
gend = levels(iris$Species), col = 1:3, pch = 1, title = $t
et

```



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

```
boxplot(Sepal.Length ~ Species, data = iris, xlab = "Species", ylab = "Sepal Length")
```

```

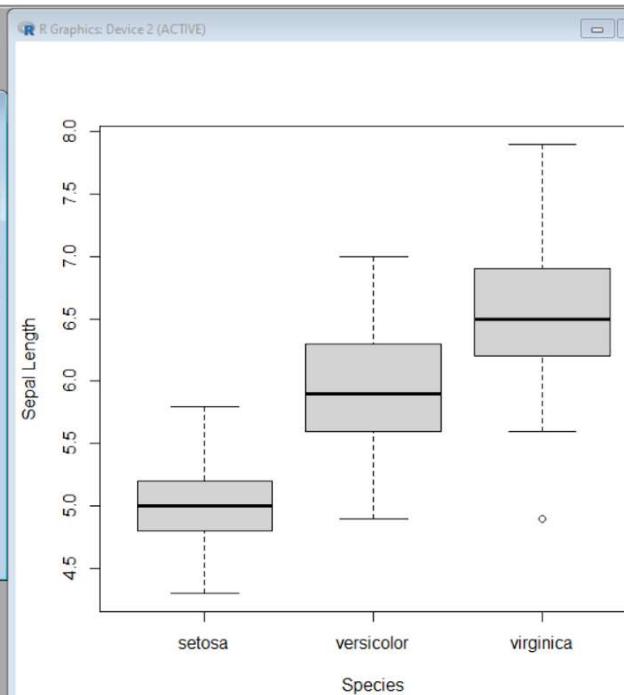
gend = levels(iris$Species), col = 1:3, pch = 1, title = $t
ot E:\3-3\R\lab\New folder\day4 p3\lab practice\3irisplot.R - R Editor
# Load the iris dataset
ge
data(iris)

lo # (iii) Draw the Box plot for Sepal length grouped by Species
boxplot(Sepal.Length ~ Species, data = iris, xlab = "Species", ylab = "Sepal Length"
o
# (iv) Draw the Box plot for Petal length grouped by Species
boxplot(Petal.Length ~ Species, data = iris, xlab = "Species", ylab = "Petal Length"
t
# (v) Find the correlation among the four features
correlation <- cor(iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal
tr print("Correlation matrix:")
print(correlation)
h
00
98
38
11
et

ot between petal width and length grouped by Species
iris$Petal.Length, col = iris$Species, xlab = "Petal W
gend = levels(iris$Species), col = 1:3, pch = 1, title = $t
et

lot for Sepal length grouped by Species
Species data = iris vlab = "Species" vlab = "Sepal L

```



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

```
boxplot(Petal.Length ~ Species, data = iris, xlab = "Species", ylab = "Petal Length")
```

The screenshot shows the RStudio interface. On the left, the R Editor window displays the following R code:

```
boxplot(Petal.Length ~ Species, data = iris, xlab = "Species", ylab = "Petal Length")
```

On the right, the R Graphics window titled "R Graphics: Device 2 (ACTIVE)" shows a boxplot of Petal Length versus Species. The y-axis is labeled "Petal Length" and ranges from 1 to 7. The x-axis is labeled "Species" and has three categories: "setosa", "versicolor", and "virginica". The boxplot shows that the "virginica" species has the highest median petal length (around 5.5), followed by "versicolor" (around 4.3), and "setosa" (around 1.5). There are several outliers for "versicolor" and one outlier for "setosa".

# (v) Find the correlation among the four features

```
correlation <- cor(iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")])
print("Correlation matrix:")
print(correlation)
```

The screenshot shows the RStudio interface. The R Editor window displays the following R code:

```
# (v) Find the correlation among the four features
correlation <- cor(iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")])
print("Correlation matrix:")
print(correlation)
```

Below the code, the console window shows the resulting correlation matrix:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.0000000	-0.1175698	0.8717538	0.8179411
Sepal.Width	-0.1175698	1.0000000	-0.4284401	-0.3661259
Petal.Length	0.8717538	-0.4284401	1.0000000	0.9628654
Petal.Width	0.8179411	-0.3661259	0.9628654	1.0000000

The screenshot shows the RStudio interface. The R Editor window displays the following R code:

```
# (v) Find the correlation among the four features
correlation <- cor(iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")])
print("Correlation matrix:")
print(correlation)
```

Below the code, the console window shows the resulting correlation matrix:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.0000000	-0.1175698	0.8717538	0.8179411
Sepal.Width	-0.1175698	1.0000000	-0.4284401	-0.3661259
Petal.Length	0.8717538	-0.4284401	1.0000000	0.9628654
Petal.Width	0.8179411	-0.3661259	0.9628654	1.0000000

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

## SOURCE CODE:

```
# Load the iris dataset
```

```
data(iris)

# Set seed for reproducibility
set.seed(123)

# (i) Randomly sample the iris dataset for training and test data (50% each)
train_indices <- sample(1:nrow(iris), nrow(iris) * 0.5)
train_data <- iris[train_indices, ]
test_data <- iris[-train_indices, ]

# (ii) Summary statistics of the train and test datasets
print("Summary Statistics of Train Data:")
print(summary(train_data))
print("Summary Statistics of Test Data:")
print(summary(test_data))

# (iii) Create Logistic Regression model with train data
model <- glm(Species ~ ., data = train_data, family = binomial)
model

# (iv) Predict the probability of the model using test data
predictions <- predict(model, newdata = test_data, type = "response")
predictions

# (v) Create a Confusion Matrix for the test model
library(caret)
confusion_matrix <- confusionMatrix(predictions, test_data$Species)
print("Confusion Matrix:")
print(confusion_matrix)
```

The screenshot shows two windows of an R environment. The left window is the R Console, and the right window is the R Editor.

**R Console Output:**

```

> predictions
      1       2       3       5       8      10
2.220446e-16 1.010925e-12 2.220446e-16 2.220446e-16 2.220446e-16
      11      12      15      17      18      19
2.220446e-16 2.220446e-16 2.220446e-16 2.220446e-16 2.220446e-16
      20      24      28      29      30      33
2.220446e-16 1.398135e-10 2.220446e-16 2.220446e-16 4.717021e-13 2.220446e-16
      36      37      40      42      44      45
2.220446e-16 2.220446e-16 2.220446e-16 9.651460e-05 5.78361e-11 2.220446e-16
      46      48      49      51      54      55
4.579871e-11 1.639041e-13 2.220446e-16 1.000000e+00 1.000000e+00 1.000000e+00
      56      57      58      59      61      62
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
      64      65      66      67      68      70
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
      71      73      75      77      80      83
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
      84      85      88      94      95      98
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
      100     101     104     105     107     108
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
      111     113     115     116     122     123
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
      125     131     133     135     139     140
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
      141     145     146
1.000000e+00 1.000000e+00 1.000000e+00
>
> # (v) Create a Confusion Matrix for the test model
> library(caret)
> confusion_matrix <- confusionMatrix(predictions, test_data$Species)
Error: 'data' and 'reference' should be factors with the same levels.
> print("Confusion Matrix:")
[1] "Confusion Matrix:"
> print(confusion_matrix)

predicted_classes setosa versicolor virginica
  setosa       10        0        0
  versicolor     0       15        5

```

**R Editor Content:**

```

# (i) Randomly sample the iris dataset for training and test data (50% each)
train_indices <- sample(1:nrow(iris), nrow(iris) * 0.5)
train_data <- iris[train_indices, ]
test_data <- iris[-train_indices, ]

# (ii) Summary statistics of the train and test datasets
print("Summary Statistics of Train Data:")
print(summary(train_data))
print("Summary Statistics of Test Data:")
print(summary(test_data))

# (iii) Create Logistic Regression model with train data
model <- glm(Species ~ ., data = train_data, family = binomial)
model

# (iv) Predict the probability of the model using test data
predictions <- predict(model, newdata = test_data, type = "response")
predictions

# (v) Create a Confusion Matrix for the test model
library(caret)
confusion_matrix <- confusionMatrix(predictions, test_data$Species)
print("Confusion Matrix:")
print(confusion_matrix)

```

### Set-III

- 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.
  - create function maxi to find the longest commute time, the function avger to find the average and the function mini to find the minimum.
  - 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.
  - How many times was your commute 20 minutes or more?

#### SOURCE CODE:

```
# (i) Create functions to find the longest commute time, average, and minimum
commute_times <- c(17, 16, 20, 24, 22, 15, 21, 15, 17, 22)
```

```
maxi <- function(data) {
```

```

max(data)
}

avger <- function(data) {
  mean(data)
}

mini <- function(data) {
  min(data)
}

longest_time <- maxi(commute_times)
average_time <- avger(commute_times)
minimum_time <- mini(commute_times)

print(paste("Longest commute time:", longest_time, "minutes"))
print(paste("Average commute time:", average_time, "minutes"))
print(paste("Minimum commute time:", minimum_time, "minutes"))

# (ii) Fix the mistake (replace 24 with 18) and find the new average using the above functions
commute_times_fixed <- commute_times
commute_times_fixed[which(commute_times_fixed == 24)] <- 18

new_average_time <- avger(commute_times_fixed)

print(paste("New average commute time (after fixing the mistake):", new_average_time,
"minutes"))

# (iii) Count how many times the commute was 20 minutes or more
count_20_or_more <- sum(commute_times_fixed >= 20)

print(paste("Number of times the commute was 20 minutes or more:", count_20_or_more))

```

```

mini <- function(data) {
  min(data)
}

longest_time <- maxi(commute_times)
average_time <- avger(commute_times)
minimum_time <- mini(commute_times)

print(paste("Longest commute time:", longest_time, "minutes"))
1] "Longest commute time: 24 minutes"
print(paste("Average commute time:", average_time, "minutes"))
1] "Average commute time: 18.9 minutes"
print(paste("Minimum commute time:", minimum_time, "minutes"))
1] "Minimum commute time: 15 minutes"

# (ii) Fix the mistake (replace 24 with 18) and find the new average using this
commute_times_fixed <- commute_times
commute_times_fixed[which(commute_times_fixed == 24)] <- 18

new_average_time <- avger(commute_times_fixed)

print(paste("New average commute time (after fixing the mistake):", new_average_time))
1] "New average commute time (after fixing the mistake): 18.3 minutes"

# (iii) Count how many times the commute was 20 minutes or more
count_20_or_more <- sum(commute_times_fixed >= 20)

print(paste("Number of times the commute was 20 minutes or more:", count_20_or_more))
1] "Number of times the commute was 20 minutes or more: 4"

```

```

# E:\3-B\RLab\New folder\day4 p3\lab practice\set3-1.R - R Editor
# (i) Create functions to find the longest commute time, average, and minimum
commute_times <- c(17, 16, 20, 24, 22, 15, 21, 15, 17, 22)

maxi <- function(data) {
  max(data)
}

avger <- function(data) {
  mean(data)
}

mini <- function(data) {
  min(data)
}

longest_time <- maxi(commute_times)
average_time <- avger(commute_times)
minimum_time <- mini(commute_times)

print(paste("Longest commute time:", longest_time, "minutes"))
print(paste("Average commute time:", average_time, "minutes"))
print(paste("Minimum commute time:", minimum_time, "minutes"))

# (ii) Fix the mistake (replace 24 with 18) and find the new average using this
commute_times_fixed <- commute_times

```

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)

### SOURCE CODE:

```
# Load the mtcars dataset
```

```
data(mtcars)
```

```
# (i) Find the dimension of the dataset
```

```
dimension <- dim(mtcars)
```

```
print(paste("Dimension of the dataset:", paste(dimension, collapse = "x")))
```

```
# (ii) Statistical summary of the features
```

```
summary_stats <- summary(mtcars)

print("Statistical Summary of the Features:")

print(summary_stats)
```

# (iii) Find the largest and smallest value of the variable hp (horsepower)

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

smallest_hp <- min(mtcars$hp)

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

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

# (iv) Mean of mileage per gallon (mpg) with respect to transmission model (am)

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

print("Mean of mileage per gallon (mpg) with respect to transmission model (am):")

print(mean_mpg_by_am)
```

# (v) Median of horsepower (hp) with respect to cylinder displacement (cyl)

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

print("Median of horsepower (hp) with respect to cylinder displacement (cyl):")

print(median_hp_by_cyl)
```

The image shows two windows from an R environment. The left window is the R Console, displaying the results of various R commands. The right window is the R Editor, showing the source code for these commands.

```

R Console:
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   :17.85  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
  Median :0.0000  Median :4.000  Median :2.000
  Mean   :0.4062  Mean   :3.688  Mean   :2.812
  3rd Qu.:1.0000  3rd Qu.:4.000  3rd Qu.:4.000
  Max.   :1.0000  Max.   :5.000  Max.   :8.000
> # (iii) Find the largest and smallest value of the variable hp (horsepower)
> largest_hp <- max(mtcars$hp)
> smallest_hp <- min(mtcars$hp)
> print(paste("Largest value of horsepower (hp):", largest_hp))
[1] "Largest value of horsepower (hp): 335"
> print(paste("Smallest value of horsepower (hp):", smallest_hp))
[1] "Smallest value of horsepower (hp): 52"
>
> # (iv) Mean of mileage per gallon (mpg) with respect to transmission model (am)
> mean_mpg_by_am <- tapply(mtcars$mpg, mtcars$am, mean)
> print("Mean of mileage per gallon (mpg) with respect to transmission model (am):")
[1] "Mean of mileage per gallon (mpg) with respect to transmission model (am):"
> print(mean_mpg_by_am)
0          1
17.14737 24.39231
>
> # (v) Median of horsepower (hp) with respect to cylinder displacement (cyl)
> median_hp_by_cyl <- tapply(mtcars$hp, mtcars$cyl, median)
> print("Median of horsepower (hp) with respect to cylinder displacement (cyl):")
[1] "Median of horsepower (hp) with respect to cylinder displacement (cyl):"
> print(median_hp_by_cyl)
 4      6      8
51.0 110.0 192.5
> |
```

```

R Editor:
# (i) Find the dimension of the dataset
dimension <- dim(mtcars)
print(paste("Dimension of the dataset:", paste(dimension, collapse = "x")))

# (ii) Statistical summary of the features
summary_stats <- summary(mtcars)
print("Statistical Summary of the Features:")
print(summary_stats)

# (iii) Find the largest and smallest value of the variable hp (horsepower)
largest_hp <- max(mtcars$hp)
smallest_hp <- min(mtcars$hp)
print(paste("Largest value of horsepower (hp):", largest_hp))
print(paste("Smallest value of horsepower (hp):", smallest_hp))

# (iv) Mean of mileage per gallon (mpg) with respect to transmission model (am)
mean_mpg_by_am <- tapply(mtcars$mpg, mtcars$am, mean)
print("Mean of mileage per gallon (mpg) with respect to transmission model (am):")
print(mean_mpg_by_am)

# (v) Median of horsepower (hp) with respect to cylinder displacement (cyl)
median_hp_by_cyl <- tapply(mtcars$hp, mtcars$cyl, median)
print("Median of horsepower (hp) with respect to cylinder displacement (cyl):")
print(median_hp_by_cyl)
```

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’)

(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.

### SOURCE CODE:

# Load the mtcars dataset

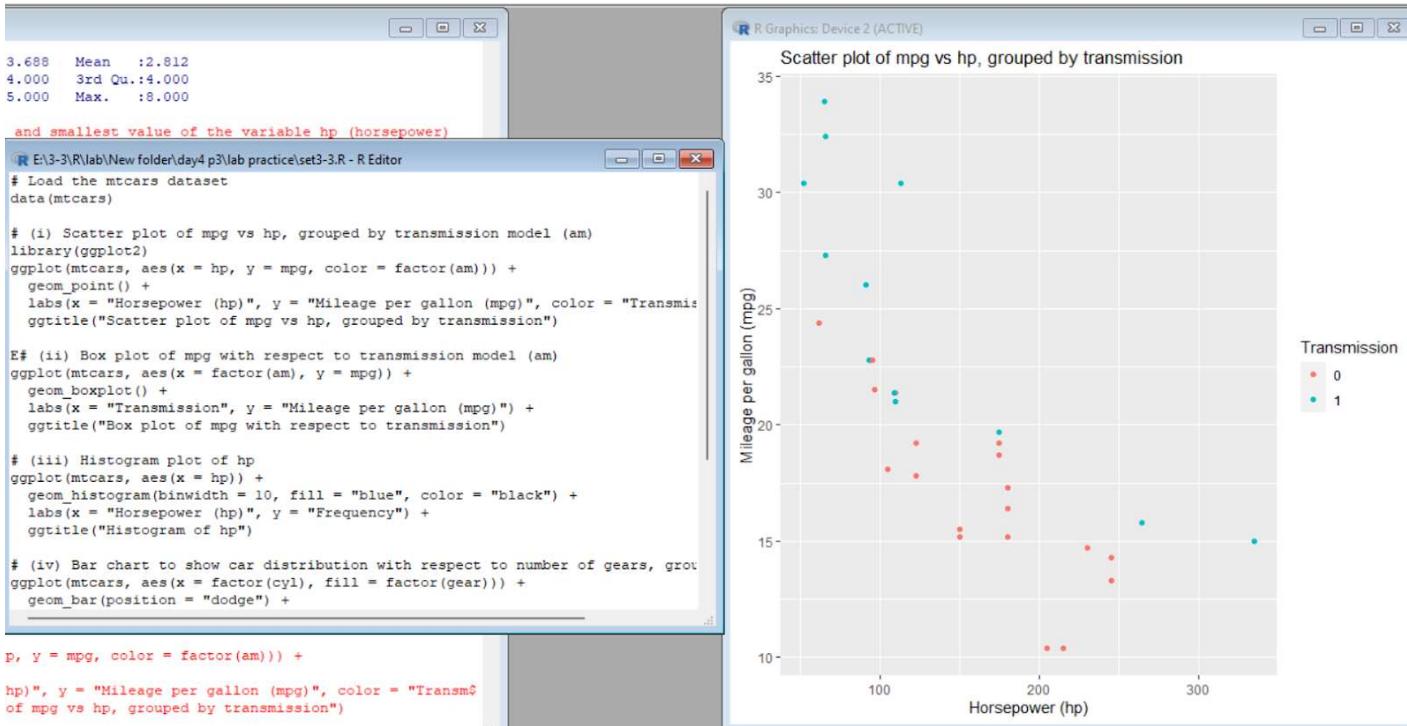
data(mtcars)

# (i) Scatter plot of mpg vs hp, grouped by transmission model (am)

library(ggplot2)

```

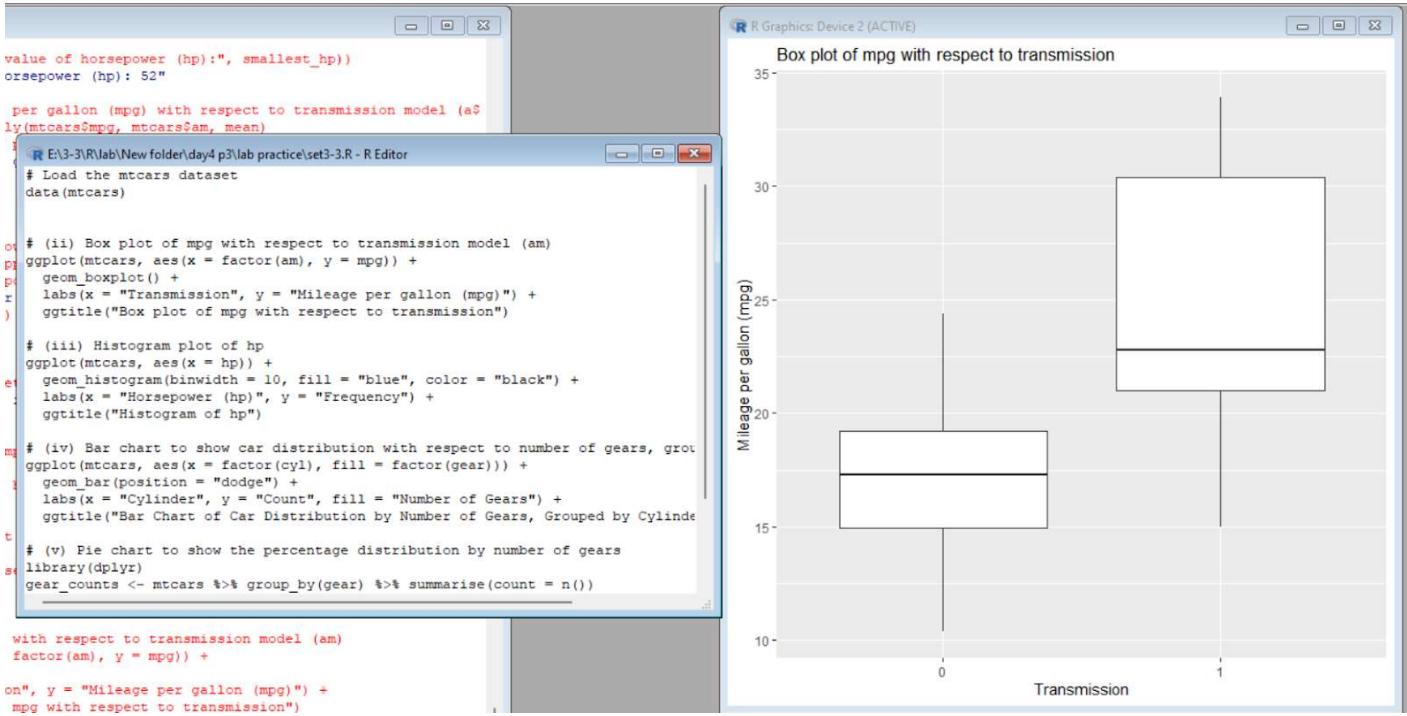
ggplot(mtcars, aes(x = hp, y = mpg, color = factor(am))) +
  geom_point() +
  labs(x = "Horsepower (hp)", y = "Mileage per gallon (mpg)", color = "Transmission") +
  ggtitle("Scatter plot of mpg vs hp, grouped by transmission")
```



```

# (ii) Box plot of mpg with respect to transmission model (am)
ggplot(mtcars, aes(x = factor(am), y = mpg)) +
  geom_boxplot() +
  labs(x = "Transmission", y = "Mileage per gallon (mpg)") +
  ggtitle("Box plot of mpg with respect to transmission")

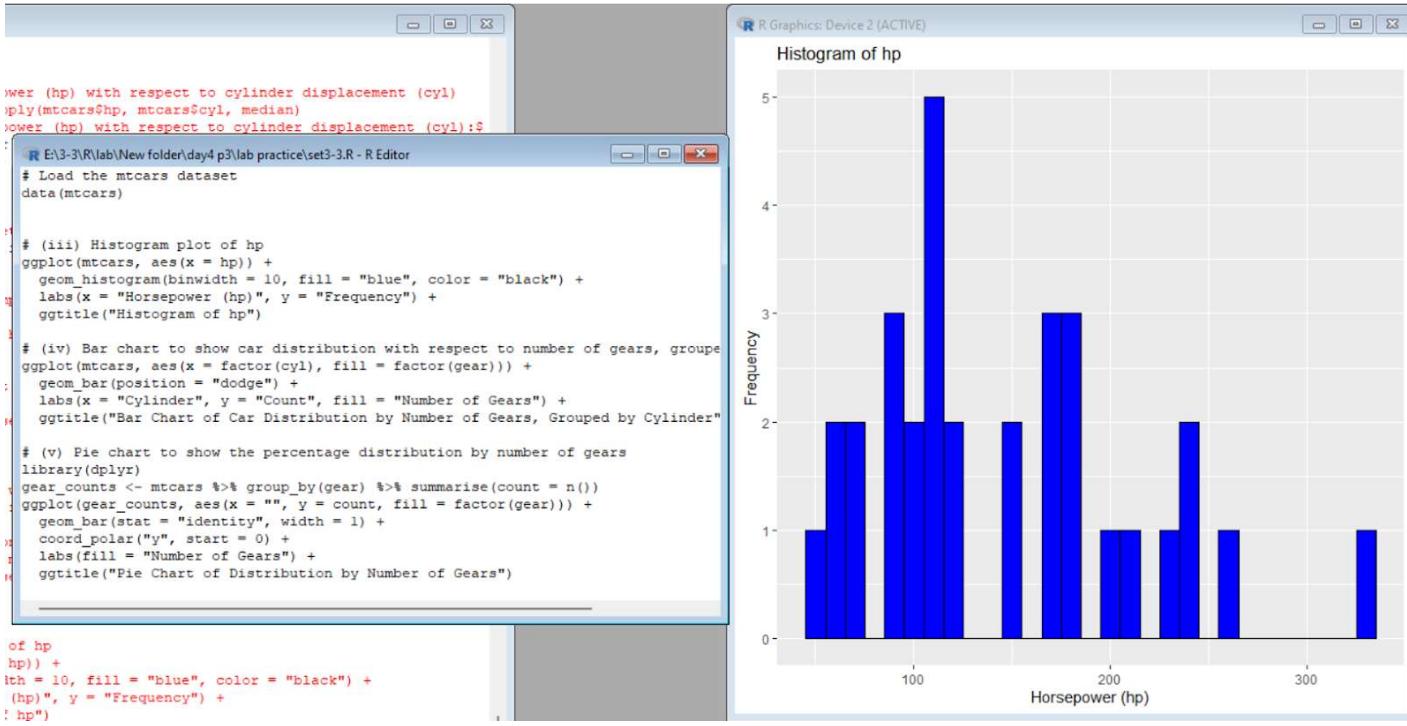
```



```

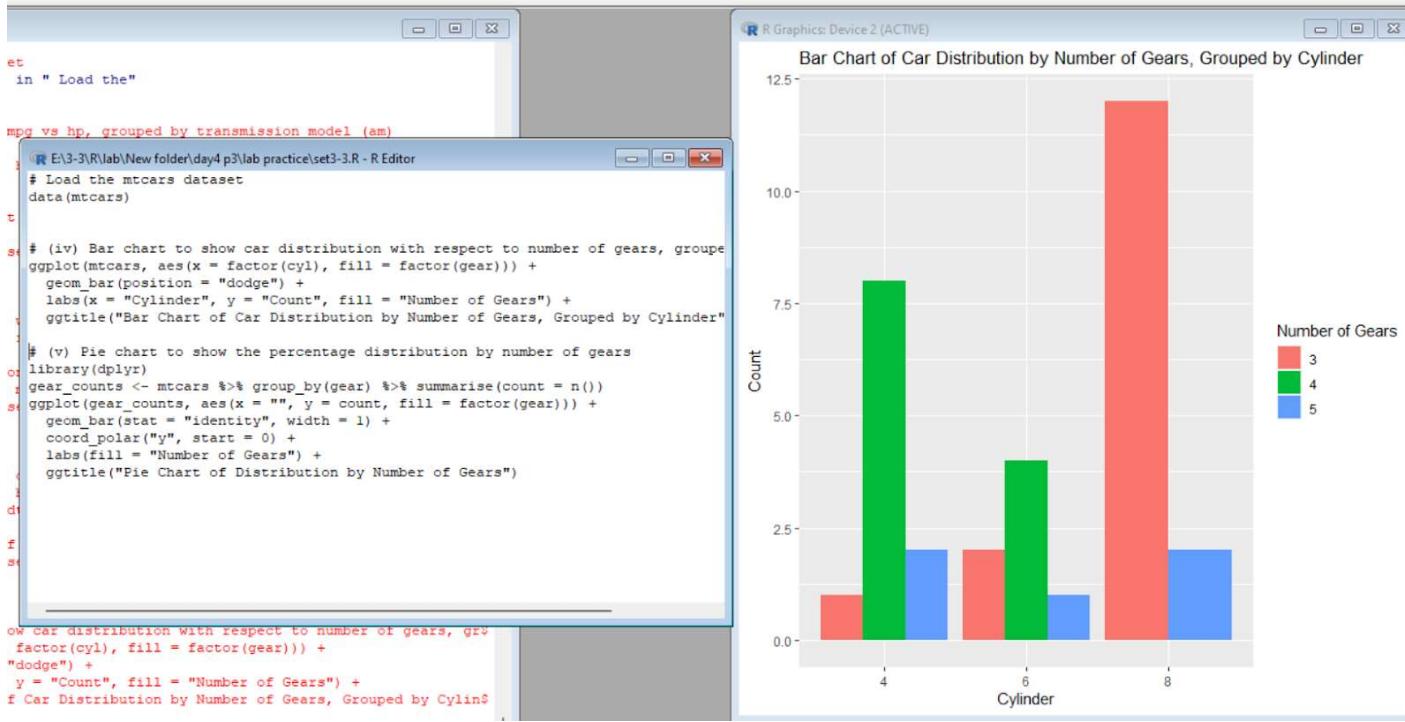
# (iii) Histogram plot of hp
ggplot(mtcars, aes(x = hp)) +
  geom_histogram(binwidth = 10, fill = "blue", color = "black") +
  labs(x = "Horsepower (hp)", y = "Frequency") +
  ggtitle("Histogram of hp")

```



```

# (iv) Bar chart to show car distribution with respect to number of gears, grouped by cylinder
ggplot(mtcars, aes(x = factor(cyl), fill = factor(gear))) +
  geom_bar(position = "dodge") +
  labs(x = "Cylinder", y = "Count", fill = "Number of Gears") +
  ggtitle("Bar Chart of Car Distribution by Number of Gears, Grouped by Cylinder")
  
```

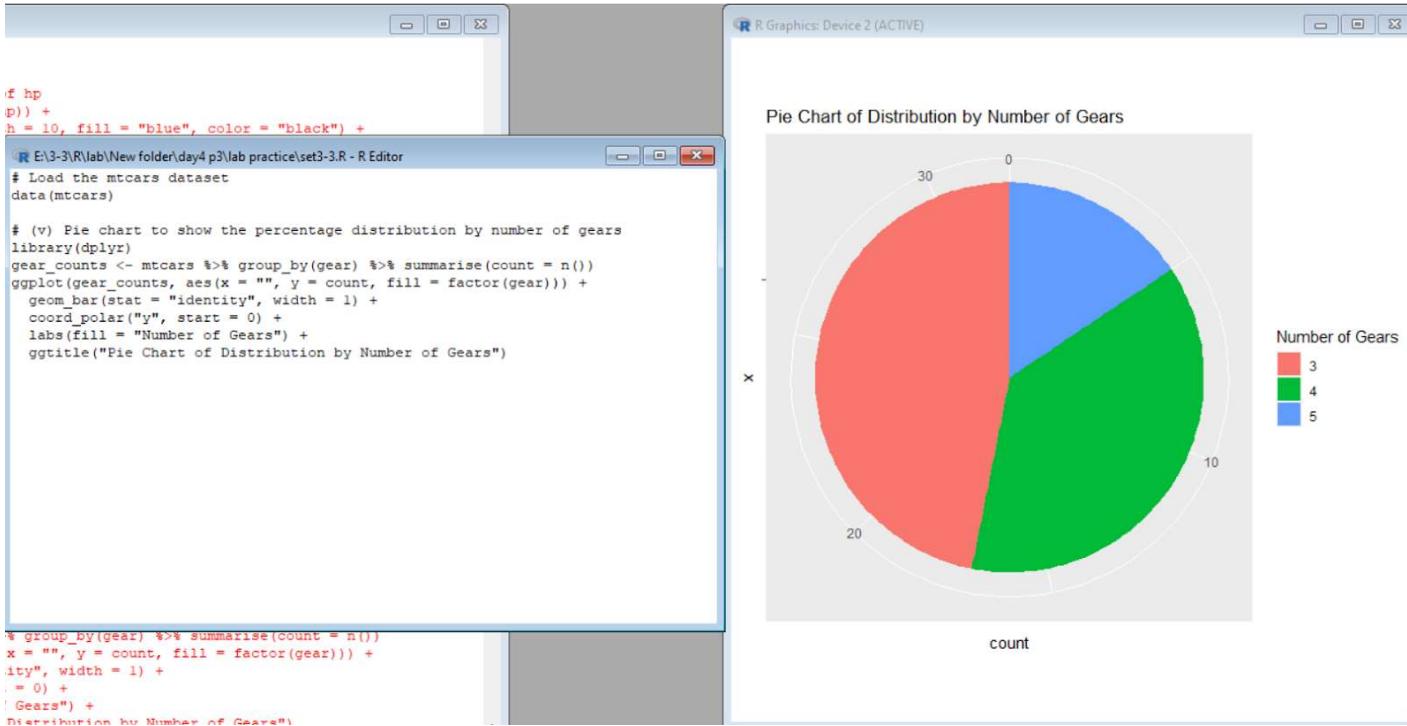


# (v) Pie chart to show the percentage distribution by number of gears

library(dplyr)

```

gear_counts <- mtcars %>% group_by(gear) %>% summarise(count = n())
ggplot(gear_counts, aes(x = "", y = count, fill = factor(gear))) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  labs(fill = "Number of Gears") +
  ggtitle("Pie Chart of Distribution by Number of Gears")
  
```



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 disp=221, hp=102 and wt=2.91

### SOURCE CODE:

```
# Load the mtcars dataset
```

```
data(mtcars)
```

```
# (i) Generate a multiple regression model
```

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

```
# (ii) Plot the multiple regression line model
```

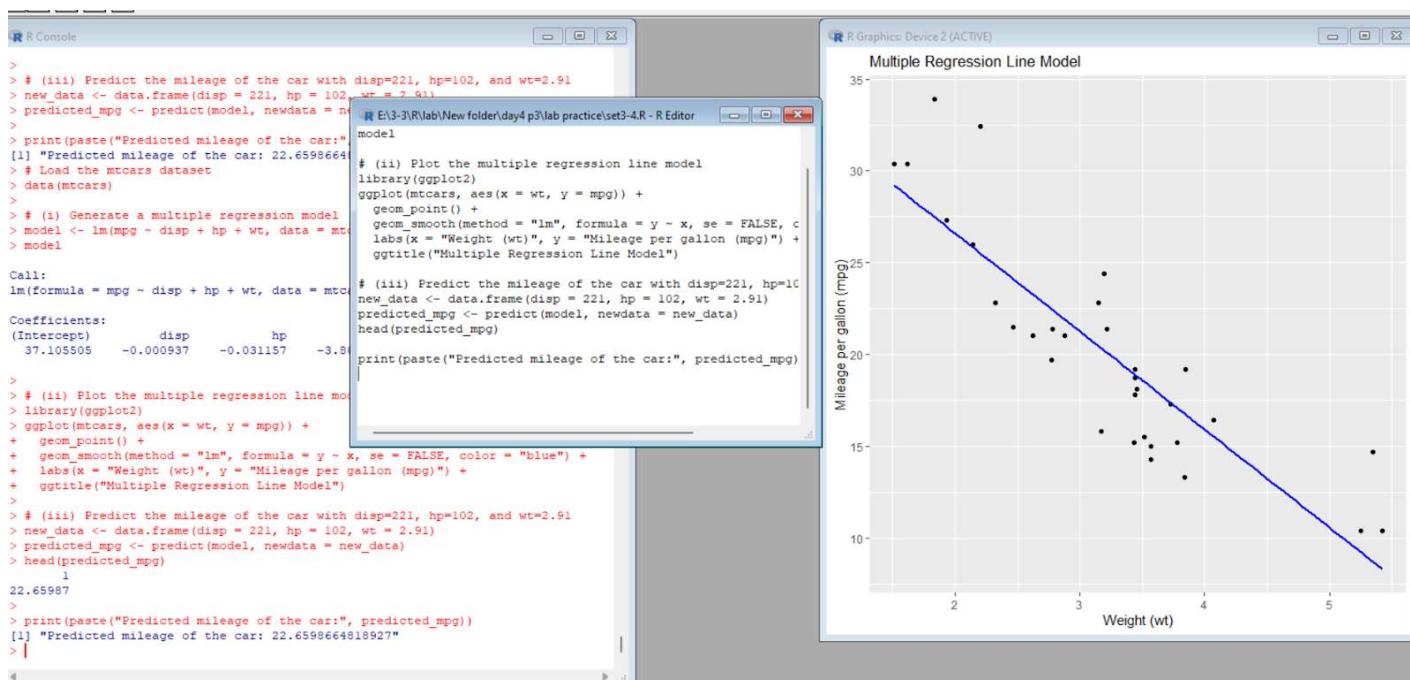
```
library(ggplot2)
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ x, se = FALSE, color = "blue") +
  labs(x = "Weight (wt)", y = "Mileage per gallon (mpg)") +
  ggtitle("Multiple Regression Line Model")
```

```

# (iii) Predict the mileage of the car with disp=221, hp=102, and wt=2.91
new_data <- data.frame(disp = 221, hp = 102, wt = 2.91)
predicted_mpg <- predict(model, newdata = new_data)

print(paste("Predicted mileage of the car:", predicted_mpg))

```



## Set IV

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

### **SOURCE CODE:**

```

fibonacci <- function(n) {
  if (n <= 1) {
    return(n)
  } else {
    return(fibonacci(n-1) + fibonacci(n-2))
  }
}

```

```

}

}

# Testing the function
for (i in 1:10) {
  result <- fibonacci(i)
  print(result)
}
+ }
>
> # Testing the function
> for (i in 1:10) {
+   result <- fibonacci(i)
+   print(result)
+ }
[1] 1
[1] 1
[1] 2
[1] 3
[1] 5
[1] 8
[1] 13
[1] 21
[1] 34
[1] 55
> |

```

```

R E:\3-3\R\lab\New folder\day4 p3\lab practice\set4-1i.R - R Editor
fibonacci <- function(n) {
  if (n <= 1) {
    return(n)
  } else {
    return(fibonacci(n-1) + fibonacci(n-2))
  }
}

# Testing the function
for (i in 1:10) {
  result <- fibonacci(i)
  print(result)
}

```

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

**SOURCE CODE:**

```

sum_natural_numbers <- 0
for (i in 1:10) {
  sum_natural_numbers <- sum_natural_numbers + i
}
print(sum_natural_numbers)

```

```

] 21
] 34
] 55
sum_natural_numbers <- 0
for (i in 1:10) {
  sum_natural_numbers <- sum_natural_numbers + i
}
print(sum_natural_numbers)
] 55
| 

```

```

R E:\3-3\R\lab\New folder\day4 p3\lab practice\set4-1ii.R - R Editor
sum_natural_numbers <- 0
for (i in 1:10) {
  sum_natural_numbers <- sum_natural_numbers + i
}
print(sum_natural_numbers)

```

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

**SOURCE CODE:**

```

vector <- 1:10
squares <- list()
for (num in vector) {
  square <- num^2
}

```

```

squares <- c(squares, square)
}
print(squares)
> vector <- 1:10
> squares <- list()
> for (num in vector) {
+   square <- num^2
+   squares <- c(squares, square)
+ }
> print(squares)
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9

[[4]]
[1] 16

[[5]]
[1] 25

[[6]]
[1] 36

[[7]]
[1] 49

[[8]]
[1] 64

[[9]]
[1] 81

[[10]]
[1] 100
> |

```

R E:\3-B\R\lab\New folder\day4 p3\lab practice\SET4-1iii.R - R Editor

```

vector <- 1:10
squares <- list()
for (num in vector) {
  square <- num^2
  squares <- c(squares, square)
}
print(squares)

```

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.
  - (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

**SOURCE CODE:**

```
# Load the dplyr package
```

```
library(dplyr)
```

```
# (i) Find the dimension of the dataset
```

```
dim(mtcars)
```

```
# (ii) Statistical summary of the features
```

```
summary(mtcars)
```

```
# (iii) Print the categorical features in the dataset
```

```
categorical_features <- c("cyl", "vs", "am", "gear", "carb")
```

```
cat_features <- mtcars %>% select(all_of(categorical_features))
```

```
print(cat_features)
```

```
# (iv) Average weight (wt) grouped by Engine shape (vs)
```

```
avg_weight_by_vs <- mtcars %>% group_by(vs) %>% summarize(avg_weight = mean(wt))
```

```
print(avg_weight_by_vs)
```

```
# (v) Largest and smallest value of weight with respect to Engine shape (vs)
```

```
largest_smallest_weight_by_vs <- mtcars %>% group_by(vs) %>% summarize(largest_weight = max(wt), smallest_weight = min(wt))
```

```
print(largest_smallest_weight_by_vs)
```

The screenshot shows two windows side-by-side. The left window is the R Console, displaying the mtcars dataset and some R code. The right window is the R Editor, showing the same R code.

```

R Console:
Merc 450SE     8   0   0     3     3
Merc 450SL     8   0   0     3     3
Merc 450SLC    8   0   0     3     3
Cadillac Fleetwood 8   0   0     3     4
Lincoln Continental 8   0   0     3     4
Chrysler Imperial 8   0   0     3     4
Fiat 128       4   1   1     4     1
Honda Civic    4   1   1     4     2
Toyota Corolla 4   1   1     4     1
Toyota Corona   4   1   0     3     1
Dodge Challenger 8   0   0     3     2
AMC Javelin    8   0   0     3     2
Camaro Z28     8   0   0     3     4
Pontiac Firebird 8   0   0     3     2
Fiat X1-9       4   1   1     4     1
Porsche 914-2   4   0   1     5     2
Lotus Europa    4   1   1     5     2
Ford Pantera L 8   0   1     5     4
Ferrari Dino    6   0   1     5     6
Maserati Bora   8   0   1     5     8
Volvo 142E      4   1   1     4     2
>
> # (iv) Average weight (wt) grouped by Engine shape (vs)
> avg_weight_by_vs <- mtcars %>% group_by(vs) %>%
> print(avg_weight_by_vs)
# A tibble: 2 × 2
  vs     avg_weight
  <dbl>      <dbl>
1 0         3.69
2 1         2.61
>
> # (v) Largest and smallest value of weight with respect to Engine shape (vs)
> largest_smallest_weight_by_vs <- mtcars %>% group_by(vs) %>% summarize(largest_weight = max(wt),
> print(largest_smallest_weight_by_vs)
# A tibble: 2 × 3
  vs     largest_weight smallest_weight
  <dbl>      <dbl>        <dbl>
1 0          5.42        2.14
2 1          3.46        1.51
> |

```

```

R Editor:
# Load the dplyr package
library(dplyr)

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

# (ii) Statistical summary of the features
summary(mtcars)

# (iii) Print the categorical features in the dataset
categorical_features <- c("cyl", "vs", "am", "gear", "carb")
cat_features <- mtcars %>% select(all_of(categorical_features))
print(cat_features)

# (iv) Average weight (wt) grouped by Engine shape (vs)
avg_weight_by_vs <- mtcars %>% group_by(vs) %>% summarize(avg_weight = mean(wt))
print(avg_weight_by_vs)

# (v) Largest and smallest value of weight with respect to Engine shape (vs)
largest_smallest_weight_by_vs <- mtcars %>% group_by(vs) %>% summarize(largest_weight = max(wt),
print(largest_smallest_weight_by_vs)

```

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

### SOURCE CODE:

```

library(ggplot2)

ggplot(mtcars, aes(x = disp, y = wt, color = factor(vs))) +
  geom_point() +
  labs(x = "Displacement (disp)", y = "Weight (wt)", color = "Engine Shape (vs)") +
  ggtitle("Scatter plot of Weight vs Displacement, factor by Engine Shape")

```



```
ggplot(mtcars, aes(x = mpg, y = hp, color = factor(vs))) +  
  geom_point() +  
  labs(x = "Mileage (mpg)", y = "Horsepower (hp)", color = "Engine Shape (vs)") +  
  ggtitle("Scatter plot of Horsepower vs Mileage, factor by Engine Shape")
```

```
0 3 2
E:\3-3\Rxlab\New folder\day4 p3\lab practice\set4-3.R - R Editor
library(ggplot2)

ggplot(mtcars, aes(x = mpg, y = hp, color = factor(vs))) +
  geom_point() +
  labs(x = "Mileage (mpg)", y = "Horsepower (hp)", color = "Engine Shape (vs)") +
  ggtitle("Scatter plot of Horsepower vs Mileage, factor by Engine Shape")

ggplot(mtcars, aes(x = mpg, y = hp, color = factor(vs))) +
  geom_point() +
  facet_wrap(~ factor(cyl), ncol = 1) +
  labs(x = "Mileage (mpg)", y = "Horsepower (hp)", color = "Engine Shape (vs)") +
  ggtitle("Scatter plot of Horsepower vs Mileage, factor by Engine Shape (Separate Facets)")

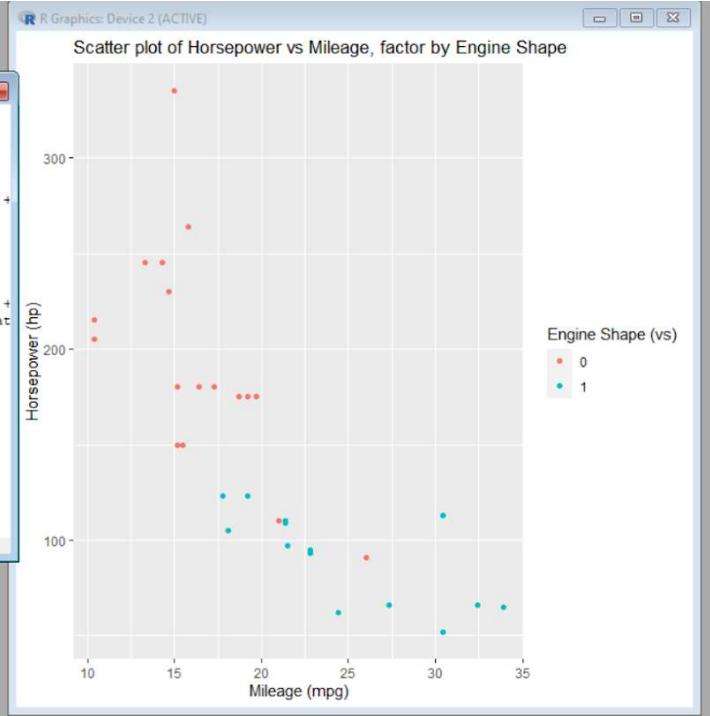
ggplot(mtcars, aes(x = hp)) +
  geom_histogram(binwidth = 5, fill = "blue", color = "black") +
  labs(x = "Horsepower (hp)", y = "Frequency") +
  ggtitle("Histogram of Horsepower")



---


(disp)", y = "Weight (wt)", color = "Engine Shape (vs")
f Weight vs Displacement, factor by Engine Shape)

g, y = hp, color = factor(vs)) +
", y = "Horsepower (hp)", color = "Engine Shape (vs)"
f Horsepower vs Mileage, factor by Engine Shape)
```



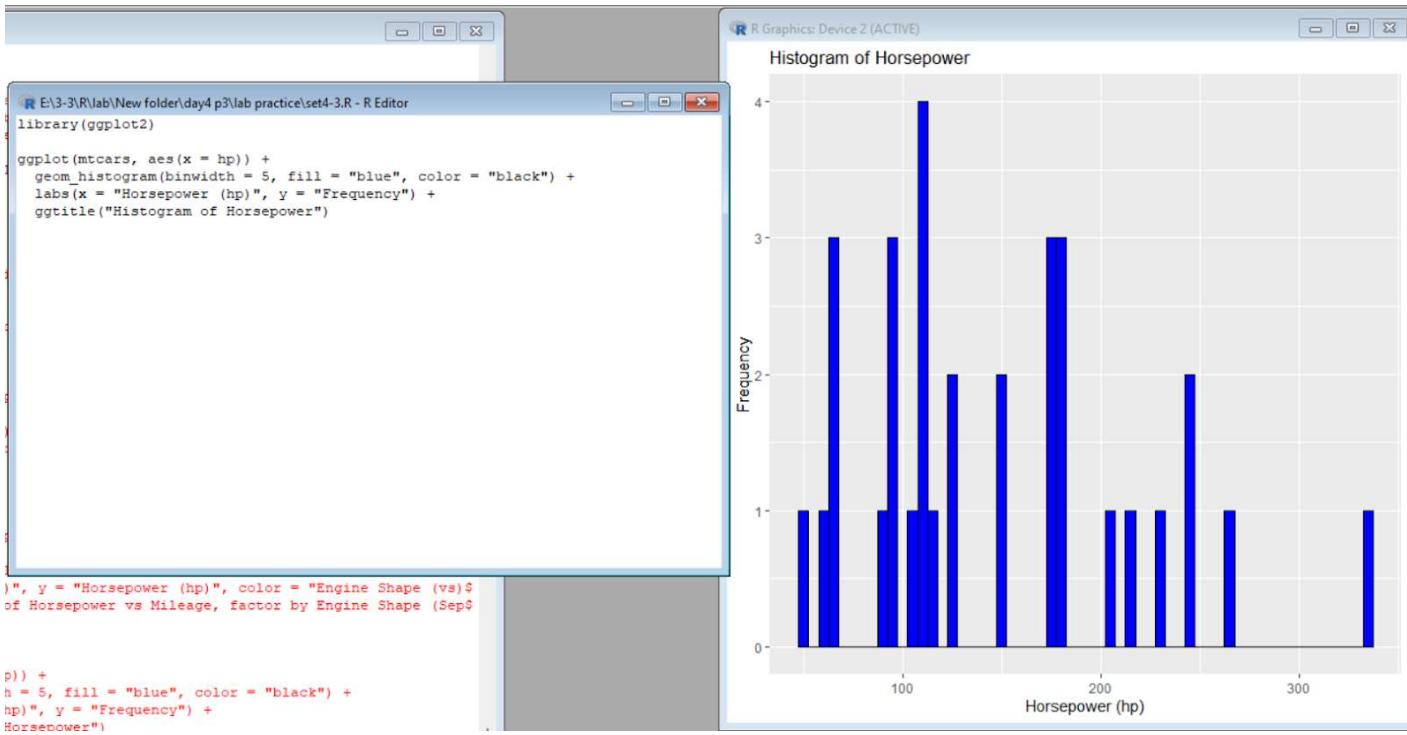
```
ggplot(mtcars, aes(x = mpg, y = hp, color = factor(vs))) +  
  geom_point() +  
  facet_wrap(~ factor(cyl), ncol = 1) +  
  labs(x = "Mileage (mpg)", y = "Horsepower (hp)", color = "Engine Shape (vs)") +  
  ggtitle("Scatter plot of Horsepower vs Mileage, factor by Engine Shape (Separate Columns by  
Cylinder)")
```



```

ggplot(mtcars, aes(x = hp)) +
  geom_histogram(binwidth = 5, fill = "blue", color = "black") +
  labs(x = "Horsepower (hp)", y = "Frequency") +
  ggtitle("Histogram of Horsepower")

```



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.

#### **SOURCE CODE:**

```
library(caret)
```

```
# Set the seed for reproducibility
```

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

```
# Split the dataset into training and test sets
```

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

```
train_data <- mtcars[train_index, ]  
test_data <- mtcars[-train_index, ]  
head(train_data)  
head(test_data)  
  
# Fit the logistic regression model  
logistic_model <- glm(vs ~ feature1 + feature2 + ..., data = train_data, family = binomial)  
logistic_model  
  
# Make predictions on the test dataset  
predictions <- predict(logistic_model, newdata = test_data, type = "response")  
predictions  
  
# Convert predicted probabilities to binary classes (0 or 1)  
predicted_classes <- ifelse(predictions > 0.5, 1, 0)  
predicted_classes  
  
# Create confusion matrix  
confusion_matrix <- table(predicted_classes, test_data$vs)  
print(confusion_matrix)
```

The image shows two windows from an R environment. The left window is the R Console, displaying a series of numerical values and R code. The right window is the R Editor, showing a script with comments and code related to logistic regression on the mtcars dataset.

```

R Console:
4.579871e-11 1.639041e-13 2.220446e-16 1.000000e+00 1.000000e+00 1.000000e+00
 56      57      58      59      61      62
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
 64      65      66      67      68      70
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
 71      73      75      77      80      83
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
 84      85      88      94      95      98
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
100     101     104     105     107     108
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
111     113     115     116     122     123
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
125     131     133     135     139     140
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
141     145     146
1.000000e+00 1.000000e+00 1.000000e+00
>
> # Convert predicted probabilities to binary classes (0 or 1)
> predicted_classes <- ifelse(predictions > 0.5, 1, 0)
> predicted_classes
 1  2  3  5  8 10 11 12 15 17 18 19 20 24 28 29 30 33 36 37
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
40 42 44 45 46 48 49 51 54 55 56 57 58 59 61 62 64 65 66 67
 0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
68 70 71 73 75 77 80 83 84 85 88 94 95 98 100 101 104 105 107 108
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
111 113 115 116 122 123 125 131 133 135 139 140 141 145 146
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
>
> # Create confusion matrix
> confusion_matrix <- table(predicted_classes, test_data$vs)
Error in table(predicted_classes, test_data$vs) :
  all arguments must have the same length
> print(confusion_matrix)

predicted_classes setosa versicolor virginica
  setosa    10      0      0
  versicolor   0     15      0
  virginica    0      0     15

R Editor:
set.seed(123)

# Split the dataset into training and test sets
train_index <- createDataPartition(mtcars$vs, p = 0.8, list = FALSE)
train_data <- mtcars[train_index, ]
test_data <- mtcars[-train_index, ]
head(train_data)
head(test_data)

# Fit the logistic regression model
logistic_model <- glm(vs ~ feature1 + feature2 + ..., data = train_data, family =
logistic_model

# Make predictions on the test dataset
predictions <- predict(logistic_model, newdata = test_data, type = "response")
predictions

# Convert predicted probabilities to binary classes (0 or 1)
predicted_classes <- ifelse(predictions > 0.5, 1, 0)
predicted_classes

# Create confusion matrix
confusion_matrix <- table(predicted_classes, 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.)

(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

## SOURCE CODE:

# i) Create a random sample from LETTERS

```
sample_letters <- sample(LETTERS, 100, replace = TRUE)
```

```
# Convert the sample into a factor
```

```
factor_sample <- factor(sample_letters)
```

```
# Extract five levels from the factor
```

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

```
# Print the extracted levels
```

```
print(five_levels)
```

```
versicolor      0      15      5
> # i)Create a random sample from LETTERS
> sample_letters <- sample(LETTERS, 100, replace = TRUE)
>
> # Convert the sample into a factor
> factor_sample <- factor(sample_letters)
>
> # Extract five levels from the factor
> five_levels <- levels(factor_sample)[1:5]
>
> # Print the extracted levels
> print(five_levels)
[1] "A" "B" "C" "D" "E"
> |
```

```
E:\3-3\R\lab\New folder\day4 p3\lab practice\set5-1.R - R Editor
# i)Create a random sample from LETTERS
sample_letters <- sample(LETTERS, 100, replace = TRUE)

# Convert the sample into a factor
factor_sample <- factor(sample_letters)

# Extract five levels from the factor
five_levels <- levels(factor_sample)[1:5]

# Print the extracted levels
print(five_levels)|

#ii
```

```
#ii
```

```
find_range <- function(vec) {
```

```
    max_val <- max(vec)
```

```
    min_val <- min(vec)
```

```
    range_val <- max_val - min_val
```

```
    return(range_val)
```

```
}
```

```
# Example usage
```

```

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

range_output <- find_range(C)

print(range_output)

```

```

+   range_val ~ max_val - min_val
+   return(range_val)
+ }
>
> # Example usage
> C <- c(9, 8, 7, 6, 5, 4, 3, 2, 1)
> range_output <- find_range(C)
> print(range_output)
[1] 8
>
> |

```

The screenshot shows an R editor window with the following code:

```

E:\3-3\lab\New folder\day4 p3\lab practice\set5-1.R - R Editor

print(five_levels)

#ii
find_range <- function(vec) {
  max_val <- max(vec)
  min_val <- min(vec)
  range_val <- max_val - min_val
  return(range_val)
}

```

#iii)

```

count_vowels <- function(str) {

  vowels <- c("a", "e", "i", "o", "u")

  str <- tolower(str)

  count <- sum(strsplit(str, "")[[1]] %in% vowels)

  return(count)
}

# Example usage

input_string <- "matrix"

vowel_count <- count_vowels(input_string)

print(vowel_count)

```

```

print(five_levels)
] "A" "B" "C" "D" "E"
#ii
find_range <- function(vec) {
  max_val <- max(vec)
  min_val <- min(vec)
  range_val <- max_val - min_val
  return(range_val)
}

# Example usage
C <- c(9, 8, 7, 6, 5, 4, 3, 2, 1)
range_output <- find_range(C)
print(range_output)
] 8

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

# Example usage
input_string <- "matrix"
vowel_count <- count_vowels(input_string)
print(vowel_count)
] 2

```

R E:\3-3\R\lab\New folder\day4 p3\lab practice\set5-1.R - R Editor

```

find_range <- function(vec) {
  max_val <- max(vec)
  min_val <- min(vec)
  range_val <- max_val - min_val
  return(range_val)
}

# Example usage
C <- c(9, 8, 7, 6, 5, 4, 3, 2, 1)
range_output <- find_range(C)
print(range_output)

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

# Example usage
input_string <- "matrix"
vowel_count <- count_vowels(input_string)
print(vowel_count)

```

## 2.Load inbuild dataset “ChickWeight” in R

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

### SOURCE CODE:

```

# Load the ChickWeight dataset

data(ChickWeight)

```

```
# (i) Explore the summary of the dataset
```

```
summary(ChickWeight)
```

```
str(ChickWeight)
```

```
table(ChickWeight$weight)
```

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

```
last_6_records <- tail(ChickWeight, 6)
```

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

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

```
ordered_data_grouped <- split(ordered_data, ordered_data$diet)
```

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

```
library(reshape2)
```

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

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

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

The image shows two windows from an R environment. The left window is the R Console, displaying R code and its output. The right window is the R Editor, showing a script with comments explaining the steps.

```

R Console
575 205 16 50 4
576 234 18 50 4
577 264 20 50 4
578 264 21 50 4
>
> # (iii) Order the data frame in ascending order by the "weight" feature, group$ ordered_data <- ChickWeight[order(ChickWeight$weight), ]
> ordered_data_grouped <- split(ordered_data, ordered_data$diet)
Error in split.default(x = seq_len(nrow(x)), f = f, drop = drop, ...):
  group length is 0 but data length > 0
> head(ordered_data)
Grouped Data: weight ~ Time | Chick
  weight Time Chick Diet
1 196 35 2 18 1
2 26 39 2 3 1
3 195 39 0 18 1
4 293 39 0 27 2
5 305 39 0 28 2
6 317 39 0 29 2
> # (iv) Perform melting function based on "Chick", "Time", "Diet"
> library(reshape2)
> melted_data <- melt(ChickWeight, id.vars = c("Chick", "Time"))
> head(melted_data)
  Chick Time Diet variable value
1 1 0 1 weight 42
2 1 2 1 weight 51
3 1 4 1 weight 59
4 1 6 1 weight 64
5 1 8 1 weight 76
6 1 10 1 weight 93
>
> # (v) Perform cast function to display the mean value of weight grouped by diet
> cast_data <- dcast(melted_data, Diet ~ variable, mean)
> head(cast_data)
  Diet weight
1 1 102.6455
2 2 122.6167
3 3 142.9500
4 4 135.2627
> |

```

```

R Editor
# Load the ChickWeight dataset
data(ChickWeight)

# (i) Explore the summary of the dataset
summary(ChickWeight)
str(ChickWeight)
table(ChickWeight$weight)

# (ii) Extract last 6 records of the dataset
last_6_records <- tail(ChickWeight, 6)
last_6_records

# (iii) Order the data frame in ascending order by the "weight" feature, grouped
ordered_data <- ChickWeight[order(ChickWeight$weight), ]
ordered_data_grouped <- split(ordered_data, ordered_data$diet)
head(ordered_data)
# (iv) Perform melting function based on "Chick", "Time", "Diet" features as ID v
library(reshape2)
melted_data <- melt(ChickWeight, id.vars = c("Chick", "Time", "Diet"))
head(melted_data)

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

```

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

### SOURCE CODE:

```
# Load the ChickWeight dataset
```

```
data(ChickWeight)
```

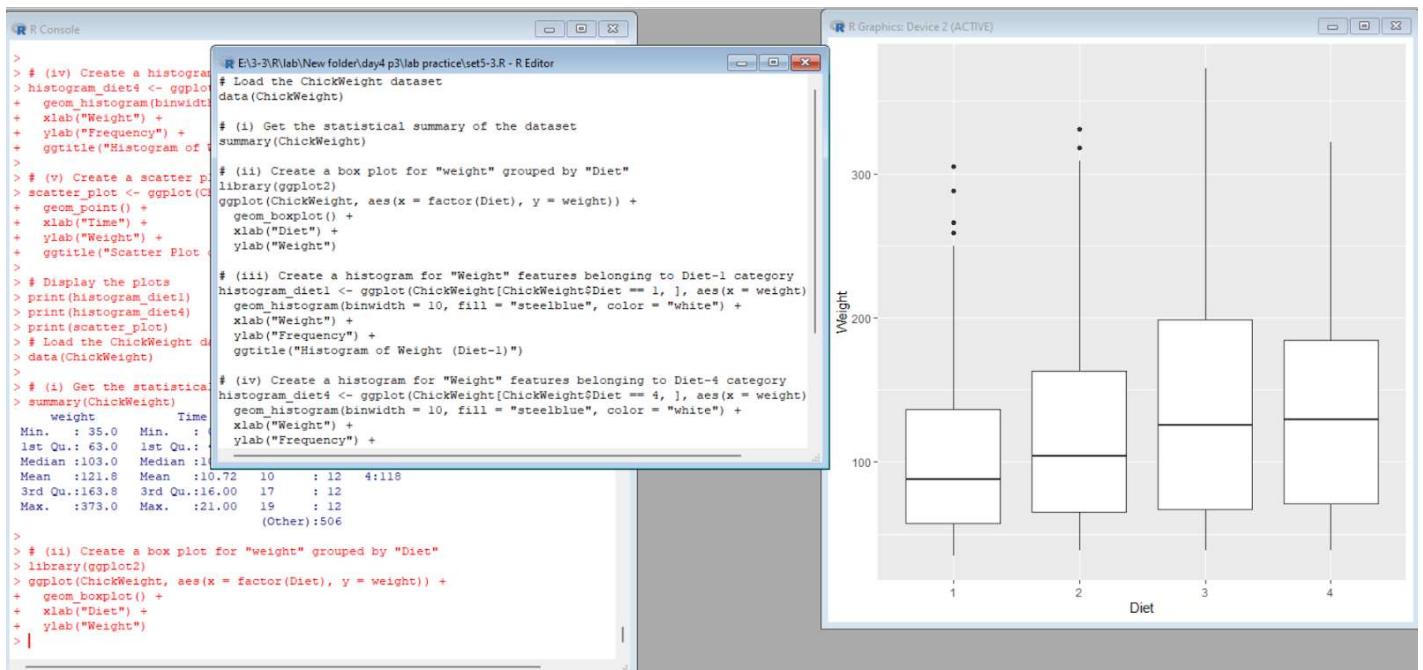
```
# (i) Get the statistical summary of the dataset
```

```
summary(ChickWeight)
```

```
# (ii) Create a box plot for "weight" grouped by "Diet"
```

```
library(ggplot2)

ggplot(ChickWeight, aes(x = factor(Diet), y = weight)) +
  geom_boxplot() +
  xlab("Diet") +
  ylab("Weight")
```



```
# (iii) Create a histogram for "Weight" features belonging to Diet-1 category
```

```
histogram_diet1 <- ggplot(ChickWeight[ChickWeight$Diet == 1, ], aes(x = weight)) +
  geom_histogram(binwidth = 10, fill = "steelblue", color = "white") +
  xlab("Weight") +
  ylab("Frequency") +
  ggtitle("Histogram of Weight (Diet-1)")
```

```

E:\3-3\R\lab\New folder\day4 p3\lab practice\set5-3.R - R Editor

# Load the ChickWeight dataset
data(ChickWeight)

# (iii) Create a histogram for "Weight" features belonging to Diet-1 category
histogram_diet1 <- ggplot(ChickWeight[ChickWeight$Diet == 1, ], aes(x = weight))
geom_histogram(binwidth = 10, fill = "steelblue", color = "white") +
xlab("Weight") +
ylab("Frequency") +
ggtitle("Histogram of Weight (Diet-1)")

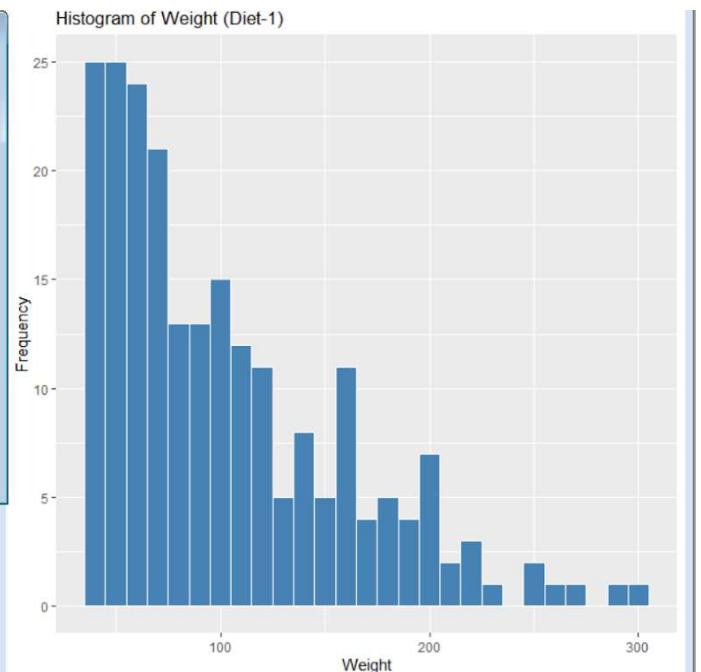
histogram_diet1

# (iv) Create a histogram for "Weight" features belonging to Diet-4 category
histogram_diet4 <- ggplot(ChickWeight[ChickWeight$Diet == 4, ], aes(x = weight))
geom_histogram(binwidth = 10, fill = "steelblue", color = "white") +
xlab("Weight") +
ylab("Frequency") +
ggtitle("Histogram of Weight (Diet-4)")

# (v) Create a scatter plot for weight vs Time grouped by Diet
scatter_plot <- ggplot(ChickWeight, aes(x = Time, y = weight, color = factor(Diet)))
geom_point() +
xlab("Time") +
ylab("Weight") +
ggtitle("Scatter Plot of Weight vs Time (Grouped by Diet)")

dataset

```

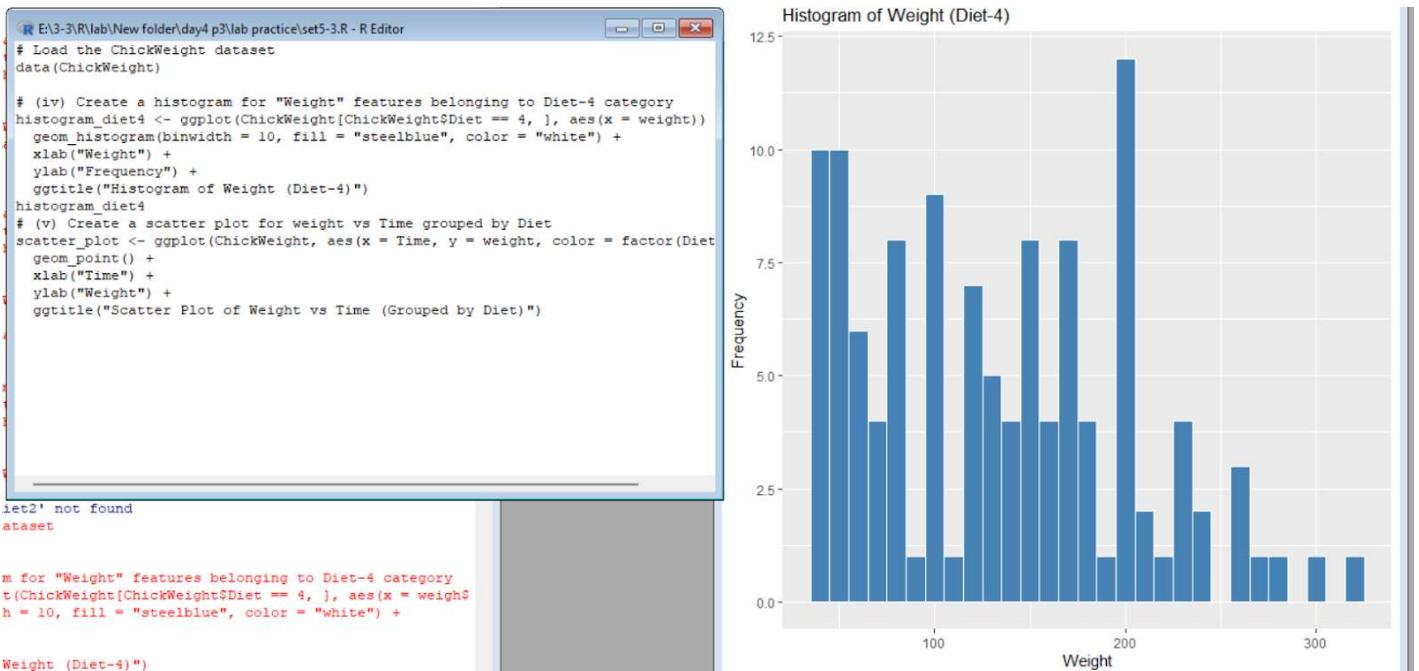


# (iv) Create a histogram for "Weight" features belonging to Diet-4 category

```

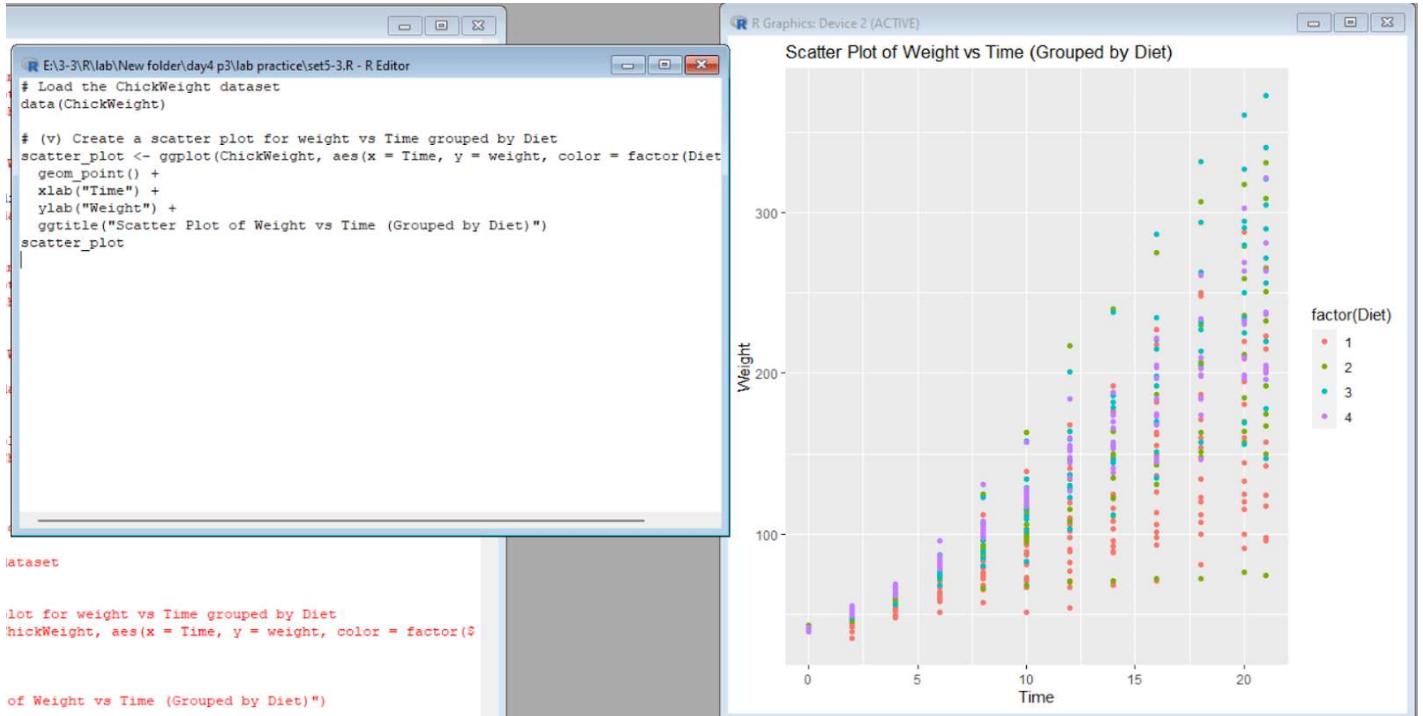
histogram_diet4 <- ggplot(ChickWeight[ChickWeight$Diet == 4, ], aes(x = weight)) +
geom_histogram(binwidth = 10, fill = "steelblue", color = "white") +
xlab("Weight") +
ylab("Frequency") +
ggtitle("Histogram of Weight (Diet-4)")

```



```
# (v) Create a scatter plot for weight vs Time grouped by Diet
```

```
scatter_plot <- ggplot(ChickWeight, aes(x = Time, y = weight, color = factor(Diet))) +
  geom_point() +
  xlab("Time") +
  ylab("Weight") +
  ggtitle("Scatter Plot of Weight vs Time (Grouped by Diet)")
```



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

#### SOURCE CODE:

```
# Load the ChickWeight dataset
data(ChickWeight)
```

```
# (i) Create a multiple regression model to predict weight using "Time" and "Diet" as predictor variables
```

```
model <- lm(weight ~ Time + Diet, data = ChickWeight)
```

```
model
```

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

```
new_data <- data.frame(Time = 10, Diet = factor(1, levels = levels(ChickWeight$Diet)))
```

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

```
head(predicted_weight)
```

```
# (iii) Calculate the mean absolute error (MAE) in the model
```

```
actual_weight <- ChickWeight$weight[ChickWeight$Time == 10 & ChickWeight$Diet == 1]
```

```
mae <- mean(abs(actual_weight - predicted_weight))
```

```
mae
```

```

# Print the predicted weight and MAE
print(paste("Predicted weight:", predicted_weight))
print(paste("MAE:", mae))

> # (i) Create a multiple regression model to predict weight using "Time" and "Diet"
> model <- lm(weight ~ Time + Diet, data = ChickWeight)
> model

Call:
lm(formula = weight ~ Time + Diet, data = ChickWeight)

Coefficients:
(Intercept)      Time       Diet2       Diet3       Diet4
   10.92        8.75      16.17      36.50      30.23

> # (ii) Predict weight for Time = 10 and Diet = 1
> new_data <- data.frame(Time = 10, Diet = factor(1, levels = levels(ChickWeight)))
> predicted_weight <- predict(model, newdata = new_data)
> head(predicted_weight)
  1
98.42931

> # (iii) Calculate the mean absolute error (MAE) in the model
> actual_weight <- ChickWeight$weight[ChickWeight$Time == 10 & ChickWeight$Diet == 1]
> mae <- mean(abs(actual_weight - predicted_weight))
> mae
[1] 18.21824

> # Print the predicted weight and MAE
> print(paste("Predicted weight:", predicted_weight))
[1] "Predicted weight: 98.4293085241932"
> print(paste("MAE:", mae))
[1] "MAE: 18.218239085314"
>

```

The screenshot shows an R script running in an IDE. The code is identical to the one above, but the output is displayed in a separate window titled 'E:\3-3\lab\New folder\day4 p3\lab practice\set5-4.R - R Editor'. The output pane contains the results of the R commands, including the coefficient table, the predicted weight for Time=10 and Diet=1, the calculation of the mean absolute error (MAE), and the final printed outputs.

```

# Load the ChickWeight dataset
data(ChickWeight)

# (i) Create a multiple regression model to predict weight using "Time" and "Diet"
model <- lm(weight ~ Time + Diet, data = ChickWeight)
model

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

# (iii) Calculate the mean absolute error (MAE) in the model
actual_weight <- ChickWeight$weight[ChickWeight$Time == 10 & ChickWeight$Diet == 1]
mae <- mean(abs(actual_weight - predicted_weight))
mae

# Print the predicted weight and MAE
print(paste("Predicted weight:", predicted_weight))
print(paste("MAE:", mae))

```