FUNDAMENTALS OF DATA SCIENCE LAB

CODE_CSDS2001P

LAB REPORT FILE

Name: Somya Vats

Roll Number: R2142230349

SAP ID: 500119012

Batch: 1_Data Science

Semester- 4

Submitted to Prof. Neeraj Chugh dated 26 April'2025

Acknowledgment

I would like to express my sincere gratitude to my course faculty, Prof. Neeraj Chugh for his valuable guidance, support, and encouragement throughout the completion of these laboratory exercises. His clear explanations, insightful feedback, and deep understanding of the subject greatly enhanced my learning experience.

His dedication and approach to teaching have not only helped me understand R programming and its applications in data science but have also inspired me to explore the field further.

Thank you for your continued support, Sir.

Abstract

This lab manual is a compilation of the work completed over ten labs focused on learning data science concepts using the R programming language. Each lab was designed to introduce or reinforce a specific concept, starting from the basics of R, such as conditional statements and data visualization, to more advanced topics like regression, clustering, and decision tree classification.

Through working on real-world datasets such as the Iris dataset, Pima Indians Diabetes dataset, and the Titanic dataset, I was able to apply statistical and machine learning techniques practically. Labs on hypothesis testing and correlation helped build a strong foundation in understanding data, while tasks like association rule mining and feature engineering showed how insights can be drawn and model accuracy can be improved.

Overall, these labs helped strengthen both my theoretical understanding and practical coding skills in R. This hands-on experience has not only improved my knowledge of data science tools but has also increased my interest in exploring this field further.

Contents

Introduction to the Manual

This lab manual is a collection of practical assignments completed as part of the coursework in R programming and data science. The labs are designed to cover a range of topics that are fundamental to understanding how data is handled, analyzed, and used to make informed decisions. Each lab focuses on a specific concept—ranging from basic programming and data visualization to machine learning techniques like regression, clustering, and classification.

By working with real-world datasets and implementing various analytical methods, this manual reflects a step-by-step learning journey, combining theoretical knowledge with hands-on application. It aims to serve both as a record of completed work and a helpful resource for revisiting key concepts in the future.

LAB-1

Title: Introduction to R

Dated: 08.01.2025

Objective:

Write a code to find out the greatest number of three using a conditional statement, inbuilt function, and user-defined function at R.

Introduction:

This lab session focuses on getting familiar with R programming by working with conditional statements and functions. We learn how to find the greatest number among three given numbers using different approaches in R programming. We explore three methods: using conditional statements, utilizing inbuilt functions, and creating user-defined functions. This lab will provide a comprehensive understanding of control structures and functions in R.

Lab Work:

1. Compare three numbers using if-else conditional statements

```
a <- 25

b <- 42

c <- 36

if (a > b & a > c) {

print(paste("Greatest number is:", a))

} else if (b > a & b > c) {
```

```
print(paste("Greatest number is:", b))
} else {
print(paste("Greatest number is:", c))
}
```

```
> #using condiitonal statements:
> a <- 12
> b<- 13
> c -<15
Error: unexpected '<' in "c -<"
> c <- 15
> if (a > b && a > c) {
+     print(paste(a, "is the largest"))
+ } else if (b > c) {
+     print(paste(b, "is the largest"))
+ } else {
+     print(paste(c, "is the largest"))
+ }
[1] "15 is the largest"
>
```

2. Compare three numbers using a custom function.

```
compare_numbers <- function(l, m, n) {
 if (l > m && l > n) {
   return(paste(l, "is the largest"))
 } else if (m > n) {
   return(paste(m, "is the largest"))
 } else {
   return(paste(n, "is the largest"))
 }
}
```

```
>
> compare_numbers(1,2,3)
[1] "3 is the largest"
> compare_numbers(324,325,200)
[1] "325 is the largest"
> compare_numbers(56,5,1)
[1] "56 is the largest"
> #done using a user defined function
>
```

3. Compare three numbers using R's built-in max() function

```
a <- 25

b <- 42

c <- 36

greatest <- max(a, b, c)

print(paste("Greatest number is:", greatest))
```

```
> print("QUESTION 2")
[1] "QUESTION 2"
> #using the built in fucntion:
>
> n <- p(2,5,8)
Error in p(2, 5, 8) : could not find function "p"
> n<- (2,5,8)
Error: unexpected ',' in "n<- (2,"
> #function "c" is used to combine/concantate
> n<- c(2, 5, 8)
> largestn <- max(n)
> paste(largestn, "is the largest")
[1] "8 is the largest"
>
> m <- c(100, 200, 450)
> largestm <- max(m)
> paste(largestm, "is the largest number")
[1] "450 is the largest number"
>
```

Conclusion

In this lab, we explored multiple ways to find the greatest number among three values in R. Using conditional statements, we could manually compare values, while the max() function provided a concise solution. Also, In R, print(), paste(), and cat() serve different purposes. print() outputs objects to the console, typically adding quotation marks for character strings. paste() concatenates strings together with a specified separator, returning the result as a single string. cat() outputs strings to the console without quotation marks and allows for multiple arguments, making it suitable for formatted text output.

LAB-2

Title: Working with visualization of Data in R

Dated: 15.01.2025

Objective:

Conduct basic data exploration by calculating summary statistics, creating histograms, and generating scatterplots.

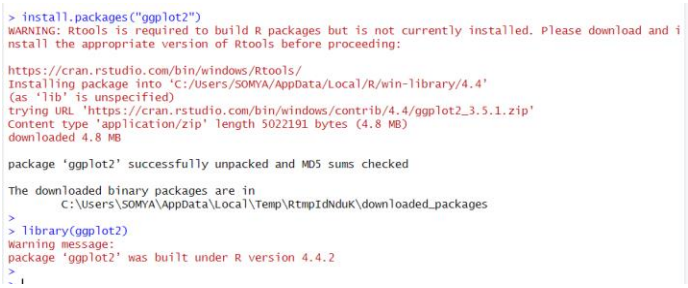
Introduction:

This report outlines the process of conducting basic data exploration using the ggplot2 package in R. The exploration includes calculating summary statistics, creating histograms, and generating scatterplots. Additionally, we delve into exploratory analysis using the airquality dataset.

Lab Work:

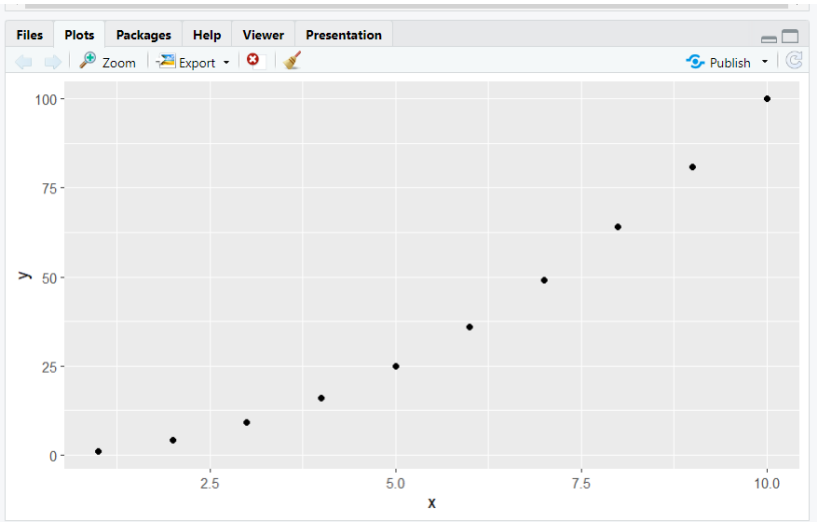
To begin, we ensured that the ggplot2 package was installed and up-to-date. Initially, the installed version was outdated, so we updated it with the following command

```
install.packages("ggplot2")

library(ggplot2)
```



1. Basic Plot Creation

```
data <- data.frame(x = 1:10, y = (1:10)^2)
ggplot(data, aes(x = x, y = y)) + geom_point()
```



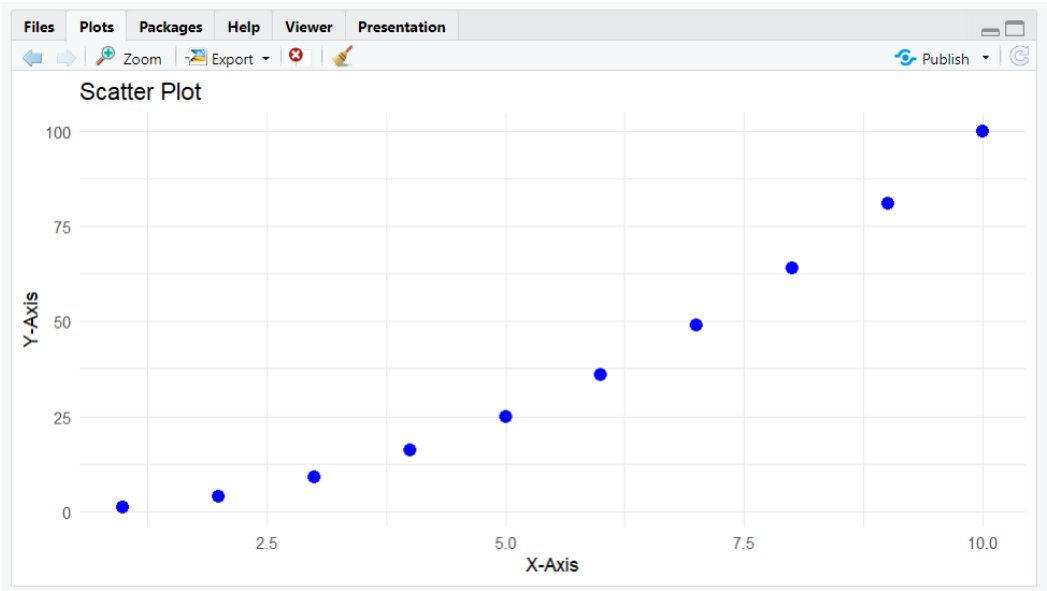
2. Common Components of ggplot2

- ggplot(): Initializes the plot and specifies data and aesthetics.
- geom_*(): Specifies the type of plot, e.g., points, lines, bars.
- scale_*(): Customizes axes, colors, and other visual elements.
- facet_*(): Creates subplots based on a factor variable.
- theme_*(): Adjusts the overall appearance of the plot.

3. Common Plots

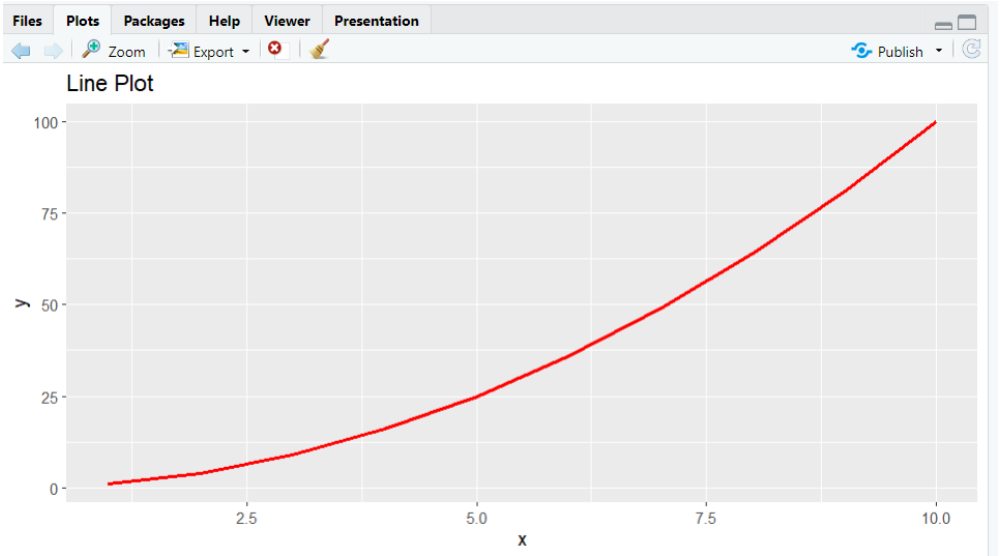
- Scatter Plot with Customizations:

```
ggplot(data, aes(x = x, y = y)) +
geom_point(color = "blue", size = 3) +
labs(title = "Scatter Plot", x = "X-Axis", y = "Y-Axis") +
theme_minimal()
```



- **Line** Plot:

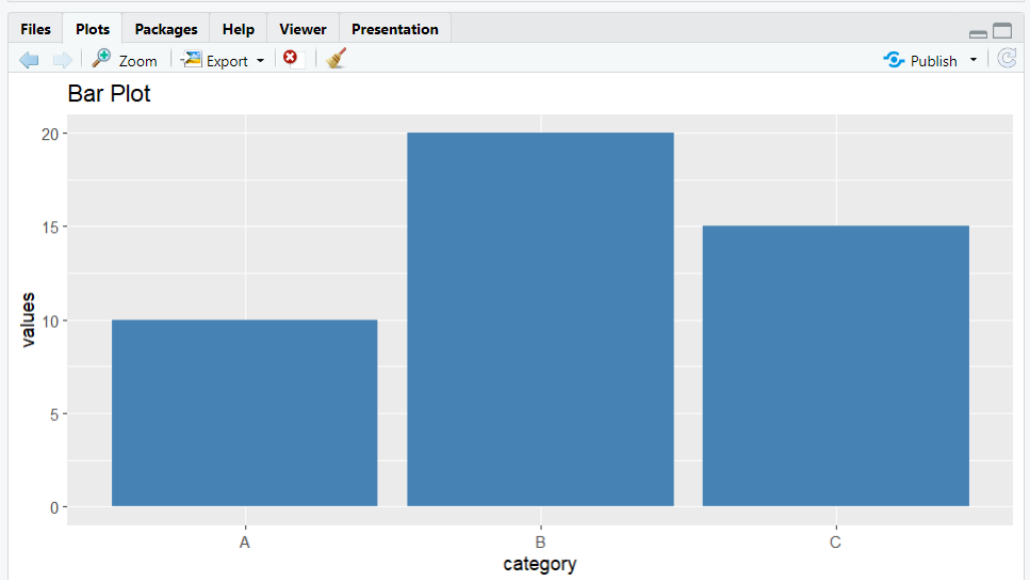
```
ggplot(data, aes(x = x, y = y)) +
geom_line(color = "red", size = 1) +
labs(title = "Line Plot")
```



- Bar Plot

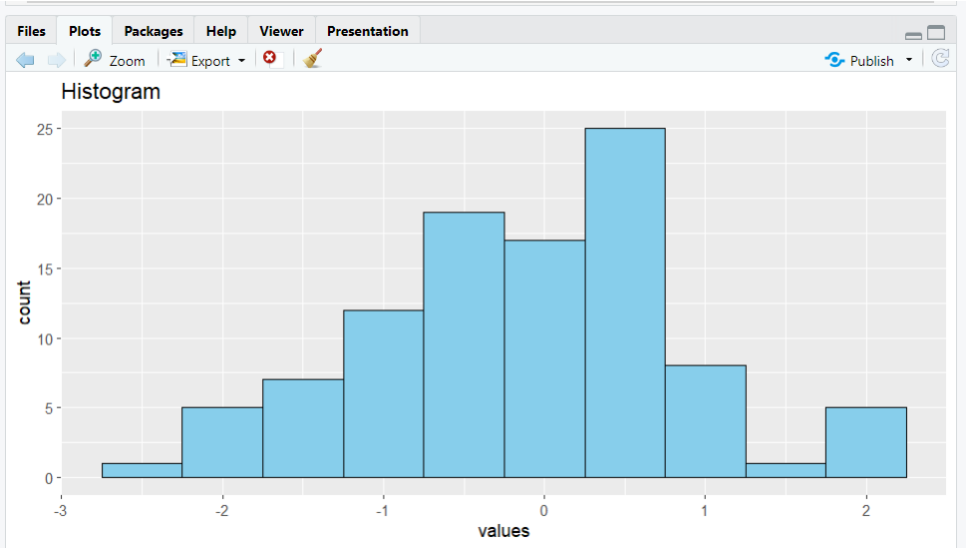
```
data_bar <- data.frame(category = c("A", "B", "C"), values = c(10, 20, 15))
ggplot(data_bar, aes(x = category, y = values)) +
```

```
geom_bar(stat = "identity", fill = "steelblue") +
labs(title = "Bar Plot")
```



- Histogram

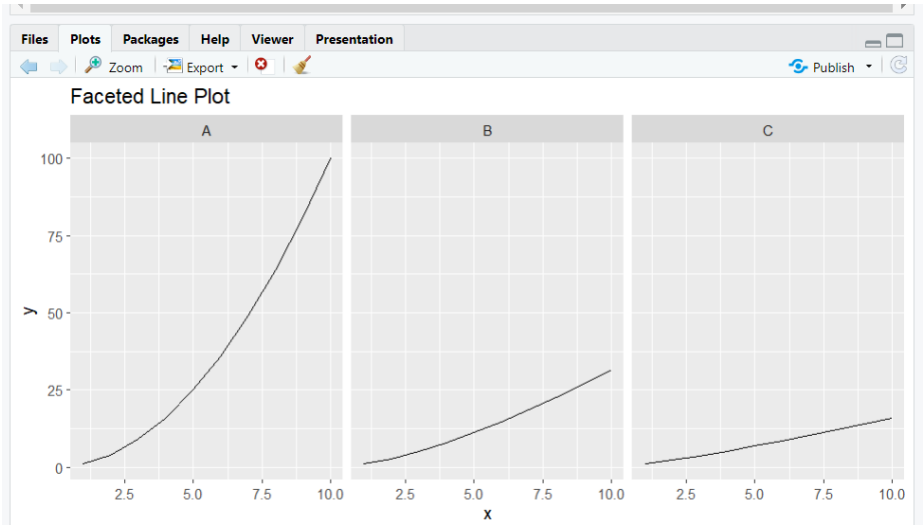
```
data_hist <- data.frame(values = rnorm(100))
ggplot(data_hist, aes(x = values)) +
geom_histogram(binwidth = 0.5, fill = "skyblue", color = "black") +
labs(title = "Histogram")
```



4. Advanced Features

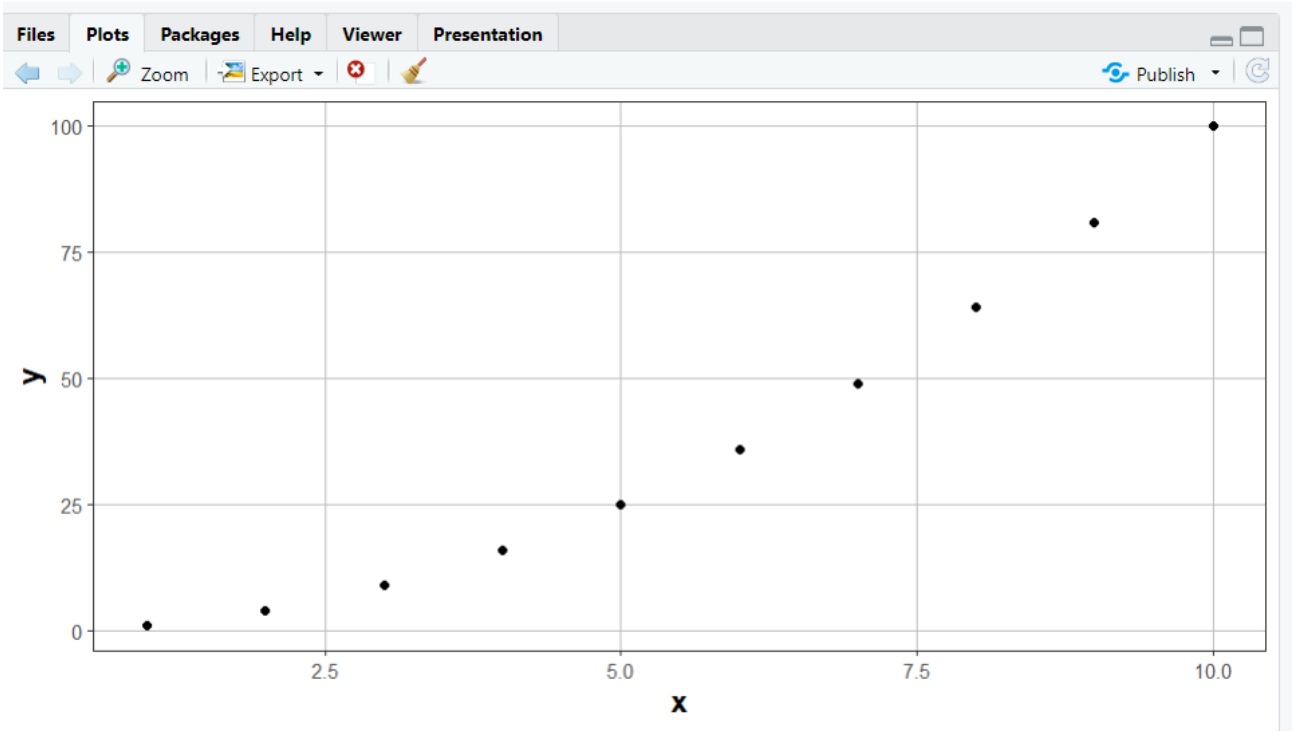
- **Faceted Line Plot**:

```
data_facet <- data.frame(
  x = rep(1:10, 3),
  y = c((1:10)^2, (1:10)^1.5, (1:10)^1.2),
  group = rep(c("A", "B", "C"), each = 10)
)
ggplot(data_facet, aes(x = x, y = y)) +
geom_line() +
facet_wrap(~ group) +
labs(title = "Faceted Line Plot")
```



1. **Custom Themes**:

```
ggplot(data, aes(x = x, y = y)) +
geom_point() +
theme_bw() +
theme(
  panel.grid.major = element_line(color = "gray"),
  panel.grid.minor = element_blank(),
  axis.title = element_text(size = 14, face = "bold")
)
```



Additional analysis for AIR QUALITY DATA

For the exploratory work of this lab, the dataset chosen is a built-in dataset in R: Air Quality that includes features like ozone, temperature, month etc.

1. Loaded the ggplot2 library and the airquality dataset.
2. Displayed the first few rows, summary statistics, and structure of the dataset.

```
print("Data visualization in R using ggplot2")
library(ggplot2)
dataset <- airquality
head(dataset)
summary(dataset)
str(dataset)
class(dataset)
```

```
> source("C:/Users/SOMYA/Desktop/sem4/FODS lab/lab2.R")
[1] "Data visualization in R using ggplot2"
>
> dataset <- airquality
> head(dataset)
  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
6    28      NA 14.9   66     5   6
>
> summary(dataset)
     Ozone           Solar.R           Wind             Temp           Month
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   Min.   :5.000
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   1st Qu.:6.000
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   Median :7.000
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   Mean   :6.993
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   3rd Qu.:8.000
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   Max.   :9.000
 NA's   :37       NA's   :7
      Day
 Min.   : 1.0
 1st Qu.: 8.0
 Median :16.0
 Mean   :15.8
 3rd Qu.:23.0
 Max.   :31.0
```

```
>
> str(dataset)
'data.frame':   153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
>
>
> class(dataset)
[1] "data.frame"
>
```

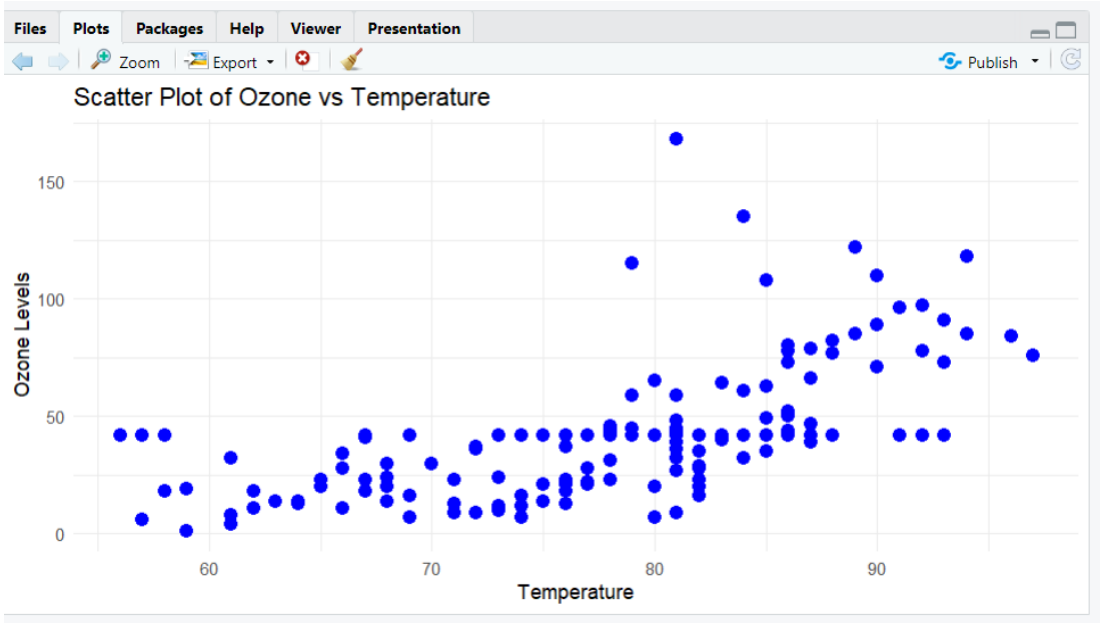
3. Checked for missing values in the Ozone column and replaced them with the column's mean.

```
sum(is.na(dataaset))

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

```
>
> #handle missing values:
> sum(is.na(dataset))
[1] 44
>
> dataset$Ozone[is.na(dataset$Ozone)] <- mean(dataset$Ozone, na.rm = TRUE)
> sum(is.na(dataaset))
Error: object 'dataaset' not found
> sum(is.na(dataset))
[1] 7
> dataset$Ozone[is.na(dataset$Ozone)] <- mean(dataset$Ozone, na.rm = TRUE)
> sum(is.na(dataset))
[1] 7
>
```

4. Created a scatter plot to visualize the relationship between Temperature and Ozone.

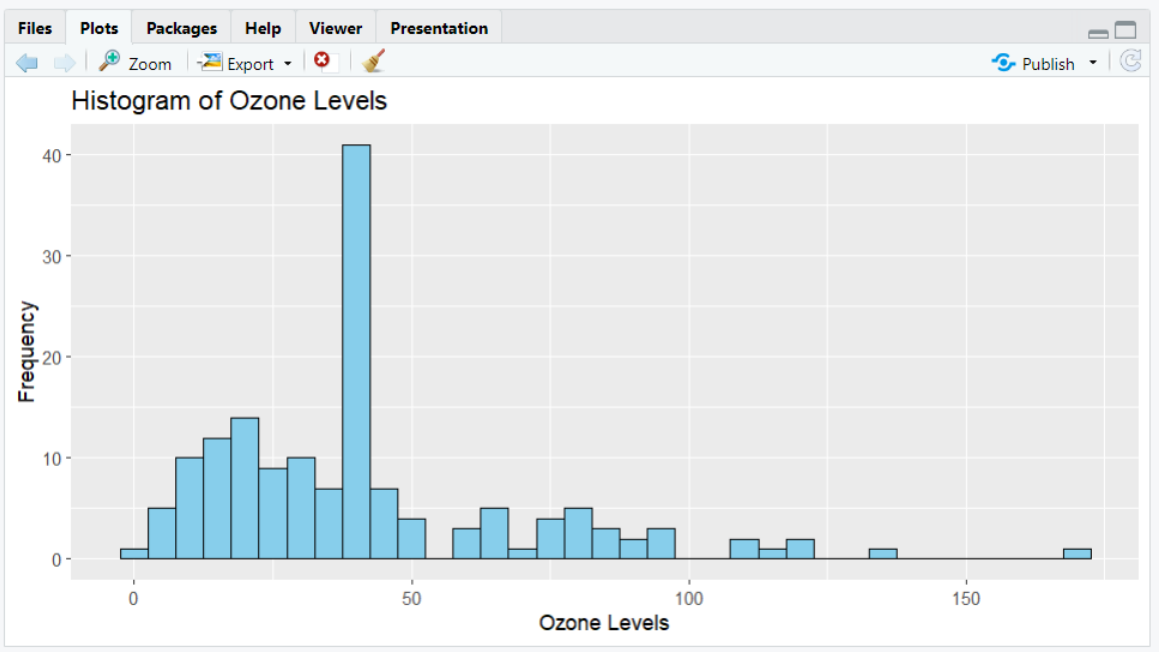
```
ggplot(dataset, aes(x = Temp, y = Ozone)) +
  geom_point(color = "blue", size = 3) +
  labs(title = "Scatter Plot of Ozone vs Temperature", x = "Temperature", y = "Ozone Levels") +
  theme_minimal()
```



5. Created a histogram to visualize the distribution of Ozone levels.

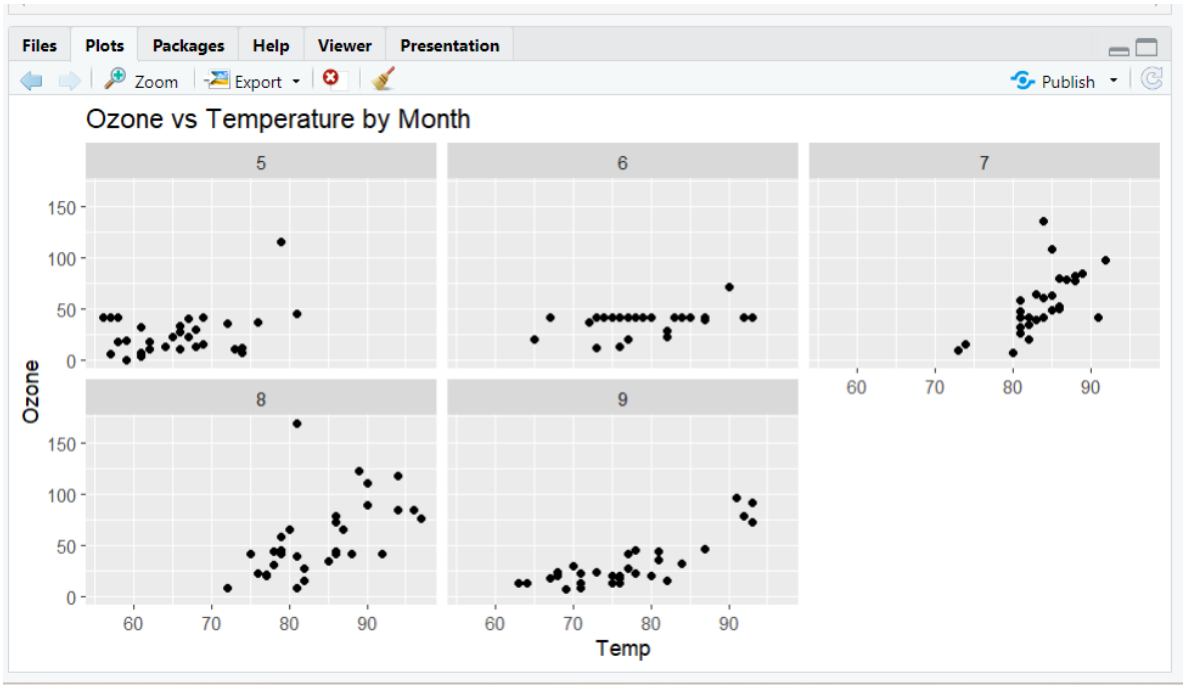
```
ggplot(dataset, aes(x = Ozone)) +
  geom_histogram(binwidth = 5, fill = "skyblue", color = "black") +
```

```
labs(title = "Histogram of Ozone Levels", x = "Ozone Levels", y = "Frequency")
```



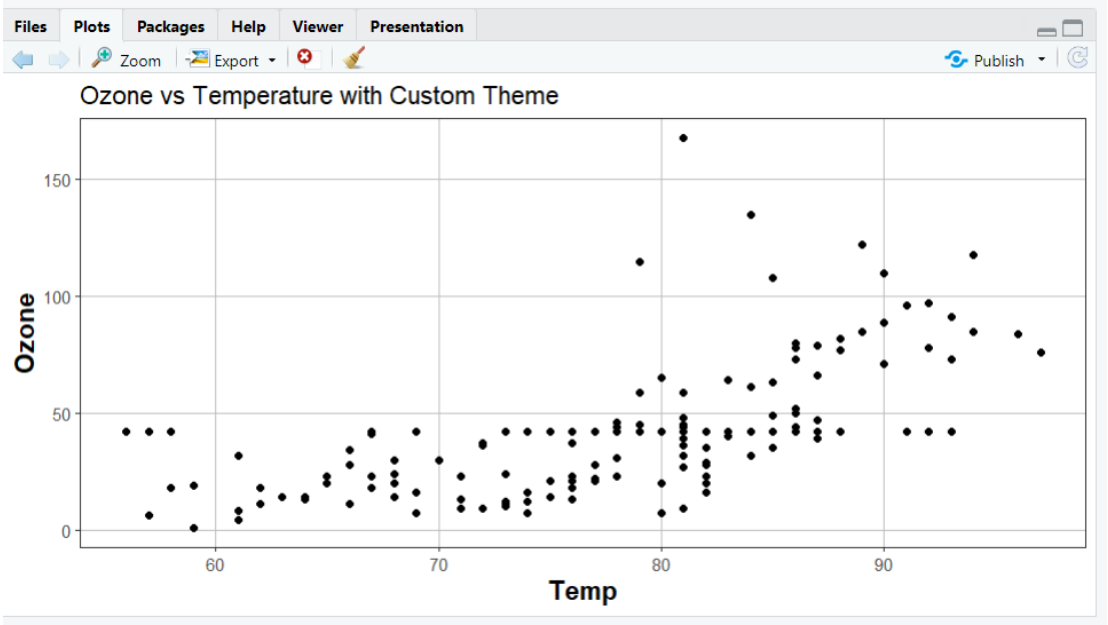
6. Applied faceting to the scatter plot to visualize Ozone vs. Temperature for each month.

```
ggplot(dataset, aes(x = Temp, y = Ozone)) +
 geom_point() +
 facet_wrap(~ Month) +
 labs(title = "Ozone vs Temperature by Month")
```



7. Customized the scatter plot with themes and specific axis limits.

```
ggplot(dataset, aes(x = Temp, y = Ozone)) +
  geom_point() +
  theme_bw() +
  theme(
    panel.grid.major = element_line(color = "gray"),
    panel.grid.minor = element_blank(),
    axis.title = element_text(size = 14, face = "bold")
  ) +
  labs(title = "Ozone vs Temperature with Custom Theme")
```



Conclusion

In this lab, we explored the fundamentals of data visualization in R using the ggplot2 package. Through a series of plots—including scatter plots, line charts, bar plots, and histograms—we gained hands-on experience in presenting data graphically for clearer interpretation. By customizing plot aesthetics, applying faceting, and using themes, we learned how to enhance the visual appeal and readability of data graphics.

The use of the built-in airquality dataset allowed for practical, real-world exploratory data analysis. Key steps included handling missing data, summarizing statistics, and visualizing relationships between variables such as Temperature and Ozone levels. Faceting helped break down trends across different months, while thematic customizations improved plot presentation.

This lab reinforced the importance of data visualization in revealing patterns, trends, and outliers that may not be immediately apparent through raw data. Overall, it served as a solid foundation for building effective, insightful visualizations in R.

LAB-3

Title: Cleaning of Data in R

Dated: 22.01.2025

Objective:

The objective is to:

1. Identify and analyze missing values in the dataset.
2. Handle missing data using techniques such as deletion, mean imputation, KNN imputation, and multiple imputation.
3. Detect outliers using statistical methods like the interquartile range (IQR) and visualize them with boxplots.
4. Apply techniques for handling outliers, including removal and transformation methods.

Introduction:

we explored methods to handle missing data and detect outliers in the airquality dataset using R. Handling missing values is crucial for accurate data analysis, as missing or incorrect data can lead to biased results. Additionally, detecting and managing outliers ensures the reliability and validity of statistical models.

Lab Work:

- Load the Data

```
data <- airquality

head(data)

summary(data)

str(data)

class(data)
```

```
[Workspace loaded from C:/Users/SOMYA/Desktop/sem4/FODS lab/.RData]

> data <- airquality
>
> head(data)
  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
6    28      NA 14.9   66     5   6
> summary(data)
     Ozone           Solar.R           Wind             Temp           Month            Day
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   Min.   :5.000   Min.   : 1.0
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   1st Qu.:6.000   1st Qu.: 8.0
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   Median :7.000   Median :16.0
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   Mean   :6.993   Mean   :15.8
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   3rd Qu.:8.000   3rd Qu.:23.0
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   Max.   :9.000   Max.   :31.0
 NA's   :37       NA's   :7
> str(data)
'data.frame':   153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
> class(data)
[1] "data.frame"
>
```

- **Identify Missing Data and remove them**

```
total_missing <- sum(is.na(airquality))
cat("Total missing values in the dataset:", total_missing, "\n")

summary(airquality)

missing_per_column <- colSums(is.na(airquality))
print(missing_per_column)
cleaned_airquality <- na.omit(airquality)
cat("Cleaned dataset (na.omit):\n")
print(head(cleaned_airquality))
```

```
> # Checking for missing values in the dataset
> total_missing <- sum(is.na(airquality))
> cat("Total missing values in the dataset:", total_missing, "\n")
Total missing values in the dataset: 44
> summary(airquality)
     Ozone           Solar.R          Wind             Temp           Month            Day
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   Min.   :5.000   Min.   : 1.0
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   1st Qu.:6.000   1st Qu.: 8.0
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   Median :7.000   Median :16.0
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   Mean   :6.993   Mean   :15.8
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   3rd Qu.:8.000   3rd Qu.:23.0
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   Max.   :9.000   Max.   :31.0
 NA's   :37       NA's   :7
> # Counting missing values by column
> missing_per_column <- colSums(is.na(airquality))
> print(missing_per_column)
  Ozone Solar.R    Wind    Temp   Month     Day
     37       7       0       0       0       0
>
```

```
> # Remove rows with any missing values
> cleaned_airquality <- na.omit(airquality)
> cat("Cleaned dataset (na.omit):\n")
Cleaned dataset (na.omit):
> print(head(cleaned_airquality))
  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
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
>
```

- **Imputation Methods**

1. **Mean Imputation:**

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

cat("Cleaned dataset (na.omit):\n")

print(head(cleaned_airquality))
```

```
> # Mean imputation for the 'Ozone' column (example)
> mean_ozone <- mean(airquality$Ozone, na.rm = TRUE)
> airquality$Ozone[is.na(airquality$Ozone)] <- mean_ozone
> cat("\nDataset after mean imputation:\n")

Dataset after mean imputation:
> print(head(airquality))
     Ozone Solar.R Wind Temp Month Day
1 41.00000     190  7.4   67     5   1
2 36.00000     118  8.0   72     5   2
3 12.00000     149 12.6   74     5   3
4 18.00000     313 11.5   62     5   4
5 42.12931      NA 14.3   56     5   5
6 28.00000      NA 14.9   66     5   6
>
```

2. **kNN Imputation:**

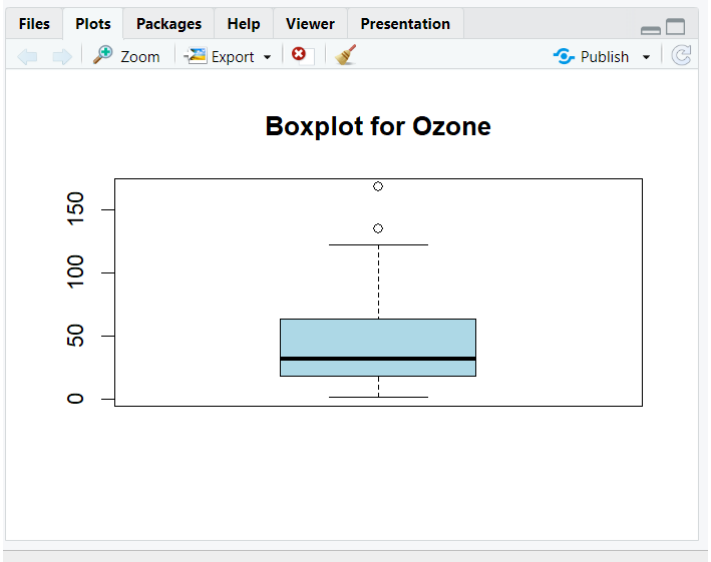
```
library(VIM)

airquality_imputed <- kNN(airquality, k = 5)

cat("\nDataset after KNN imputation:\n")
```

```
print(head(airquality_imputed))
```

```
The following object is masked from 'package:datasets':

    sleep

Warning message:
package 'VIM' was built under R version 4.4.2
> airquality_imputed <- kNN(airquality, k = 5)
> cat("\nDataset after KNN imputation:\n")

Dataset after KNN imputation:
> print(head(airquality_imputed))
  Ozone Solar.R Wind Temp Month Day Ozone_imp Solar.R_imp Wind_imp Temp_imp Month_imp Day_imp
1    41     190  7.4   67     5   1     FALSE       FALSE    FALSE    FALSE     FALSE   FALSE
2    36     118  8.0   72     5   2     FALSE       FALSE    FALSE    FALSE     FALSE   FALSE
3    12     149 12.6   74     5   3     FALSE       FALSE    FALSE    FALSE     FALSE   FALSE
4    18     313 11.5   62     5   4     FALSE       FALSE    FALSE    FALSE     FALSE   FALSE
5    18      99 14.3   56     5   5      TRUE        TRUE    FALSE    FALSE     FALSE   FALSE
6    28     242 14.9   66     5   6     FALSE        TRUE    FALSE    FALSE     FALSE   FALSE
>
```

- **Detecting Outliers**

```
boxplot(airquality$Ozone, main = "Boxplot for Ozone", col = "lightblue")
```




```
iqr_ozone <- IQR(airquality$Ozone, na.rm = TRUE)

lower_bound <- quantile(airquality$Ozone, 0.25, na.rm = TRUE) - 1.5 * iqr_ozone

upper_bound <- quantile(airquality$Ozone, 0.75, na.rm = TRUE) + 1.5 * iqr_ozone

outliers <- airquality$Ozone[airquality$Ozone < lower_bound | airquality$Ozone > upper_bound]

cat("\nOutliers detected in Ozone column:", outliers, "\n")
```

```
> boxplot(airquality$Ozone, main = "Boxplot for Ozone", col = "lightblue")
> # Identifying outliers using IQR method
> iqr_ozone <- IQR(airquality$Ozone, na.rm = TRUE)
> lower_bound <- quantile(airquality$Ozone, 0.25, na.rm = TRUE) - 1.5 * iqr_ozone
> upper_bound <- quantile(airquality$Ozone, 0.75, na.rm = TRUE) + 1.5 * iqr_ozone
> outliers <- airquality$Ozone[airquality$Ozone < lower_bound | airquality$Ozone > upper_bound]
> cat("\nOutliers detected in Ozone column:", outliers, "\n")

Outliers detected in Ozone column: NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA N
A NA NA 135 NA NA NA NA NA NA NA NA 168 NA NA
>
```

- **Treating Outliers**

airquality_no_outliers <- airquality[airquality$Ozone >= lower_bound & airquality$Ozone <= upper_bound,]

cat("\nDataset after removing outliers:\n")

print(head(airquality))

```
> # Removing outliers (example for Ozone)
> airquality_no_outliers <- airquality[airquality$Ozone >= lower_bound & airquality$Ozone <= upper_bound,
]
> cat("\nDataset after removing outliers:\n")

Dataset after removing outliers:
> print(head(airquality))
  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
6    28      NA 14.9   66     5   6
>
```

airquality$Ozone[airquality$Ozone < lower_bound] <- lower_bound

airquality$Ozone[airquality$Ozone > upper_bound] <- upper_bound

cat("\nDataset after capping outliers:\n")

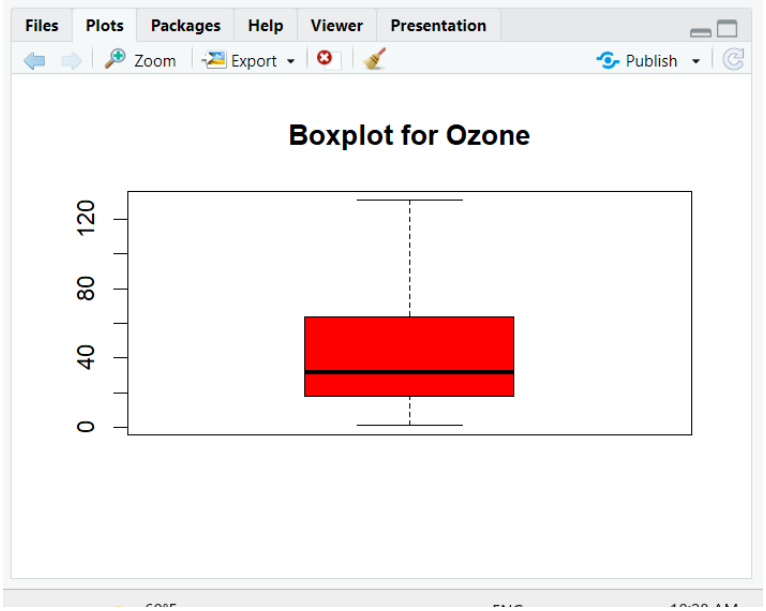
print(head(airquality))

print(head(airquality_no_outliers))

```
Dataset after capping outliers:
> print(head(airquality))
  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
6    28      NA 14.9   66     5   6
> print(head(airquality_no_outliers))
   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
NA    NA      NA   NA   NA    NA  NA
6     28      NA 14.9   66     5   6
> boxplot(airquality$Ozone, main = "Boxplot for Ozone", col = "red")
>
```

- Boxplot (after removing outliers)

```
boxplot(airquality$Ozone, main = "Boxplot for Ozone", col = "red")
```



Conclusion

In this lab session, we gained hands-on experience in cleaning data using R. We learned to identify and handle missing values through various methods such as deletion, mean imputation, and kNN imputation. We also explored how to detect and treat outliers using statistical techniques like the interquartile range (IQR) and visualized them through boxplots. These preprocessing steps are essential to ensure the accuracy, reliability, and quality of data before performing any analysis or building models. Mastering data cleaning techniques is a fundamental skill for any data scientist.

LAB-4

Title: Correlation Analysis in R

Dated: 28.01.2025

Objective:

Exploring Correlation in the Pima Indians Diabetes Dataset using R

Introduction:

In this analysis, we will explore the correlation between different features in the Pima Indians Diabetes dataset using R. We will compute correlation and covariance matrices, visualize relationships with scatter plots, and create heatmaps.

LAB WORK:

Setup: necessary libraries were imported and data set was loaded

```
install.packages("mlbench")
install.packages("ggplot2")
install.packages("corrplot")
install.packages("reshape2")
library(mlbench)
library(ggplot2)
library(corrplot)
library(reshape2)
```

- Load the data set and check its summary/description.

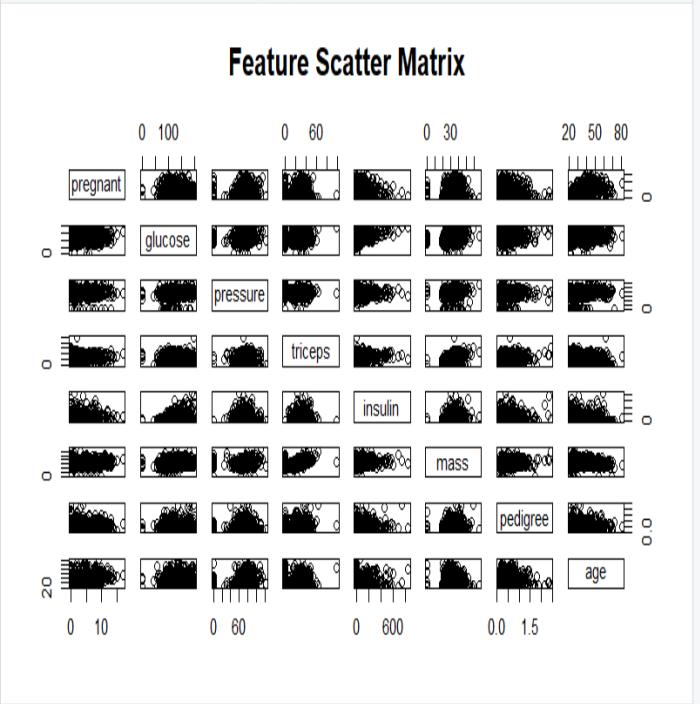
```
data(PimaIndiansDiabetes)
df <- PimaIndiansDiabetes
head(data)
summary(data)
str(data)
class(data)
```

```
> head(data)
  pregnant glucose pressure triceps insulin mass pedigree age diabetes
1        6     148       72      35       0 33.6    0.627  50      pos
2        1      85       66      29       0 26.6    0.351  31      neg
3        8     183       64       0       0 23.3    0.672  32      pos
4        1      89       66      23      94 28.1    0.167  21      neg
5        0     137       40      35     168 43.1    2.288  33      pos
6        5     116       74       0       0 25.6    0.201  30      neg
> summary(data)
    pregnant         glucose         pressure         triceps         insulin           mass
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00   Min.   :  0.0   Min.   : 0.00
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00   1st Qu.:  0.0   1st Qu.:27.30
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00   Median : 30.5   Median :32.00
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54   Mean   : 79.8   Mean   :31.99
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00   3rd Qu.:127.2   3rd Qu.:36.60
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00   Max.   :846.0   Max.   :67.10
    pedigree           age         diabetes
 Min.   :0.0780   Min.   :21.00   neg:500
 1st Qu.:0.2437   1st Qu.:24.00   pos:268
 Median :0.3725   Median :29.00
 Mean   :0.4719   Mean   :33.24
 3rd Qu.:0.6262   3rd Qu.:41.00
 Max.   :2.4200   Max.   :81.00
```

```
> str(data)
'data.frame':   768 obs. of  9 variables:
 $ pregnant: num  6 1 8 1 0 5 3 10 2 8 ...
 $ glucose : num  148 85 183 89 137 116 78 115 197 125 ...
 $ pressure: num  72 66 64 66 40 74 50 0 70 96 ...
 $ triceps : num  35 29 0 23 35 0 32 0 45 0 ...
 $ insulin : num  0 0 0 94 168 0 88 0 543 0 ...
 $ mass    : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
 $ pedigree: num  0.627 0.351 0.672 0.167 2.288 ...
 $ age     : num  50 31 32 21 33 30 26 29 53 54 ...
 $ diabetes: Factor w/ 2 levels "neg","pos": 2 1 2 1 2 1 2 1 2 2 ...
> class(data)
[1] "data.frame"
>
```

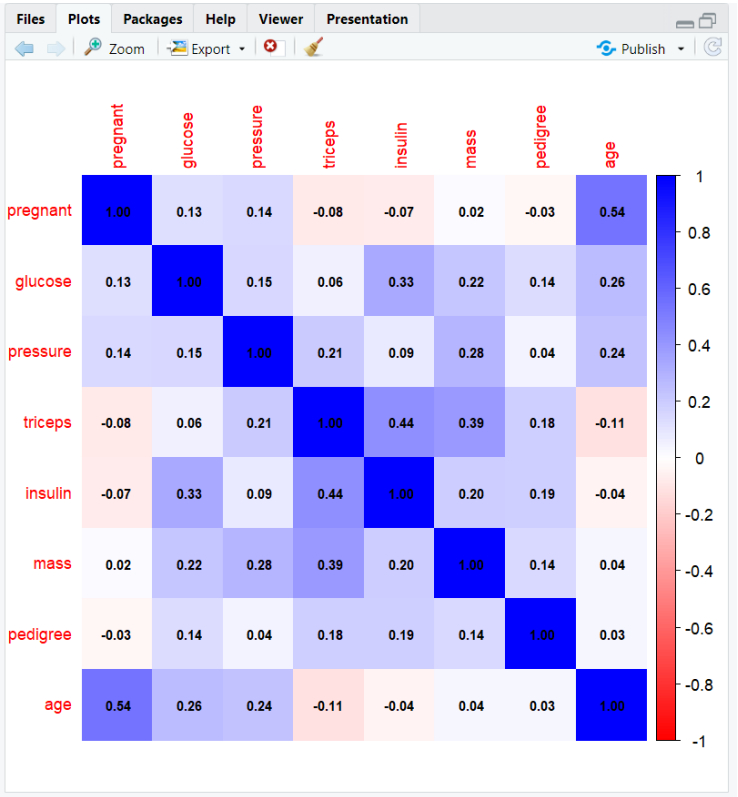
- Create a scatter matrix to visualize feature relationships.

```
pairs(data[,1:8], main="Feature Scatter Matrix")
```



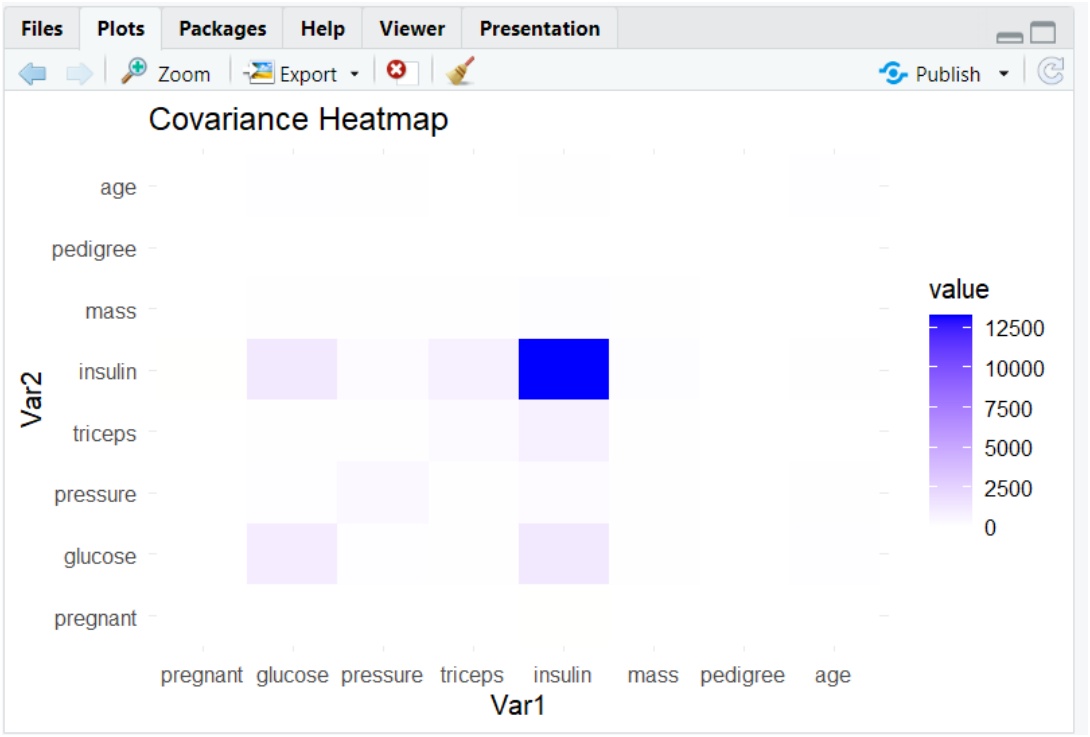
- **Compute and visualize the correlation matrix.**

```
cor_matrix <- cor(data[,1:8])
library(corrplot)
corrplot(cor_matrix, method="color",
    col=colorRampPalette(c("red", "white", "blue"))(200),
    addCoef.col="black", tl.cex=0.8, number.cex=0.7)
```



- **Create a covariance heatmap**

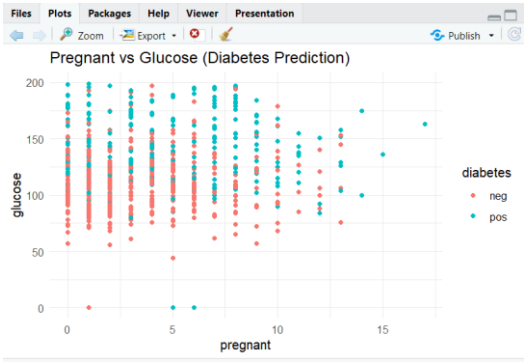
```
cor_matrix <- cor(data[, 1:8])  # Excluding the 'diabetes' column (categorical)
corrplot(cor_matrix, method="color",
         col=colorRampPalette(c("red", "white", "blue"))(200),
         addCoef.col="black", tl.cex=0.8, number.cex=0.7)
```

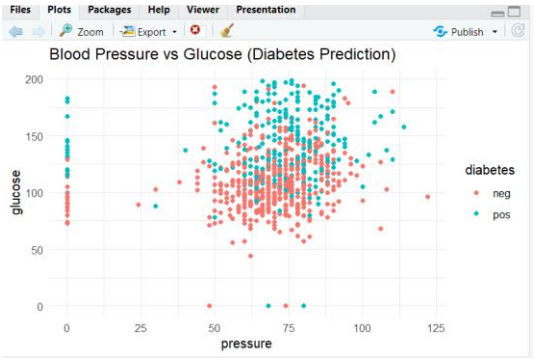


- **Create scatter plots for Glucose vs each feature**

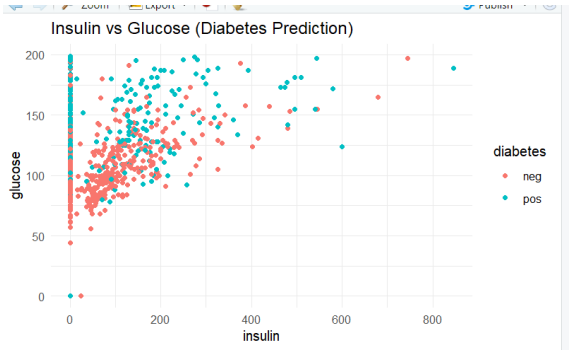
```
# Glucose vs Pregnant
ggplot(data, aes(x=pregnant, y=glucose, color=diabetes)) +
  geom_point() +
  ggtitle("Pregnant vs Glucose (Diabetes Prediction)") +
  theme_minimal()

# Glucose vs Blood Pressure
ggplot(data, aes(x=pressure, y=glucose, color=diabetes)) +
  geom_point() +
  ggtitle("Blood Pressure vs Glucose (Diabetes Prediction)") +
  theme_minimal()
```

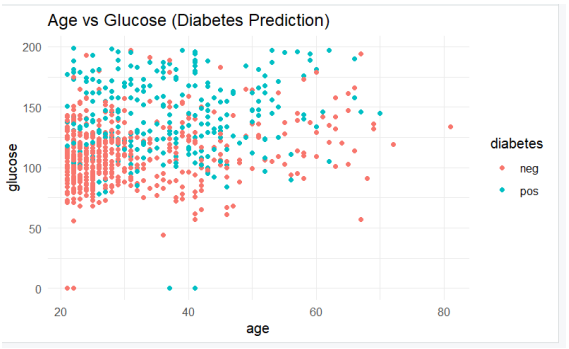
```
# Glucose vs Triceps Skinfold
ggplot(data, aes(x=triceps, y=glucose, color=diabetes)) +
  geom_point() +
  ggtitle("Triceps Skinfold vs Glucose (Diabetes Prediction)") +
  theme_minimal()
```

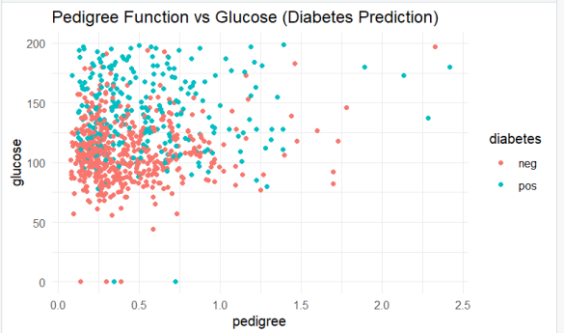





```
# Glucose vs Insulin
ggplot(data, aes(x=insulin, y=glucose, color=diabetes)) +
  geom_point() +
  ggtitle("Insulin vs Glucose (Diabetes Prediction)") +
  theme_minimal()
# Glucose vs Pedigree Function
ggplot(data, aes(x=pedigree, y=glucose, color=diabetes)) +
  geom_point() +
  ggtitle("Pedigree Function vs Glucose (Diabetes Prediction)") +
  theme_minimal()
# Glucose vs Age
ggplot(data, aes(x=age, y=glucose, color=diabetes)) +
  geom_point() +
  ggtitle("Age vs Glucose (Diabetes Prediction)") +
  theme_minimal()
```







Conclusion

We analyzed the correlation between features in the Pima Indians Diabetes dataset using R. After loading and inspecting the data, we computed correlation and covariance matrices and visualized relationships using scatter plots and heatmaps. Key findings showed that Glucose and BMI have a strong correlation with diabetes, while some features may require preprocessing due to missing values. This analysis helps in identifying important predictors for diabetes detection.

LAB-5

Title: Linear Regression

Dated: 05.02.2025

Objective:

- Implement Simple Linear Regression on **iris dataset**
- Train a model and evaluate its performance
- Apply regression on a **custom dataset** and compare results
- Visualize relationships and error metrics

Introduction:

In this lab, we implemented **Simple Linear Regression** to analyze relationships between variables and make predictions. We explored how a dependent variable change based on an independent variable using R.

Lab Work:

Step 1: Load Dataset

Load iris dataset.

Display the first few rows to check and then print out the summary statistics.

```
dataset <- iris
summary(dataset)
head(dataset)
```

```
+   ggtitle("Age vs Glucose (Diabetes Prediction)") +
+   theme_minimal()
> dataset <- iris
>
> summary(dataset)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
> head(dataset)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
>
```

Step 2: Train-Test Split (80-20)

- Divide the dataset into **training (80%)** and **testing (20%)**
- Ensure reproducibility using set.seed()

```
set.seed(123)  # Setting seed for reproducibility

train_indices <- sample(1:nrow(dataset), size = 0.8 * nrow(dataset))  # 80% for training

train_data <- dataset[train_indices, ]

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

```
> summary(model)

Call:
lm(formula = Sepal.Length ~ Petal.Length, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-1.23656 -0.30673 -0.03334  0.26958  1.02598

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.27856    0.08921   47.96   <2e-16 ***
Petal.Length  0.41289    0.02120   19.47   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4148 on 118 degrees of freedom
Multiple R-squared:  0.7627,    Adjusted R-squared:  0.7607
F-statistic: 379.2 on 1 and 118 DF,  p-value: < 2.2e-16

>
```

Step 4: Make Predictions on Test Data

- Use the trained model to predict Sepal Length on test data

- Store actual vs predicted values in a dataframe

```
# Fit the linear model on the training data
model <- lm(Sepal.Length ~ Petal.Length, data = train_data)
summary(model)


predictions <- predict(model, newdata = test_data)


# Compare predicted vs. actual values for the test data
results <- data.frame(Actual = test_data$Sepal.Length, Predicted = predictions)
print(results)
```

```
> results <- data.frame(Actual = test_data$Sepal.Length, Predicted = predictions)
> print(results)
    Actual Predicted
1      5.1  4.856602
2      4.9  4.856602
3      4.7  4.815313
11     5.4  4.897891
18     5.1  4.856602
19     5.7  4.980469
28     5.2  4.897891
33     5.2  4.897891
36     5.0  4.774024
48     4.6  4.856602
55     6.5  6.177850
56     5.7  6.136561
57     6.3  6.219139
58     4.9  5.641093
59     6.6  6.177850
61     5.0  5.723671
62     5.9  6.012694
65     5.6  5.764960
66     6.7  6.095272
70     5.6  5.888827
77     6.8  6.260428
83     5.8  5.888827
84     6.0  6.384295
98     6.2  6.053983
100    5.7  5.971405
105    6.5  6.673318
113    6.8  6.549451
125    6.7  6.632029
131    7.4  6.797184
141    6.7  6.590740
>
```

Step 5: Evaluate Model Performance

- Compute **Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE)** to measure accuracy

```
# Calculate error metrics
```

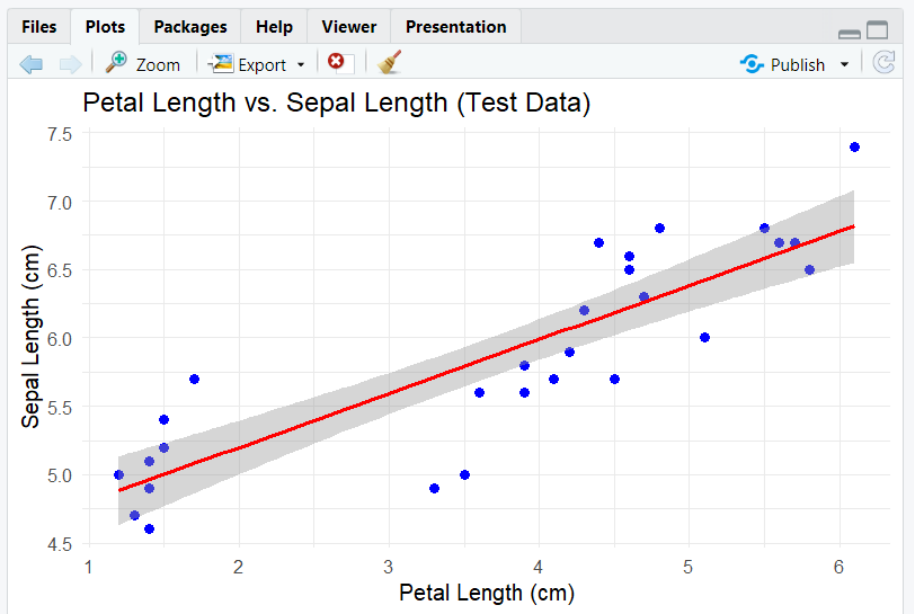
```
mae <- mean(abs(results$Actual - results$Predicted))  # Mean Absolute Error
mse <- mean((results$Actual - results$Predicted)^2)   # Mean Squared Error
rmse <- sqrt(mse)  # Root Mean Squared Error

# Print error metrics
cat("Mean Absolute Error (MAE):", mae, "\n")
cat("Mean Squared Error (MSE):", mse, "\n")
cat("Root Mean Squared Error (RMSE):", rmse, "\n")
```

```
141    6.7  6.590740
> # Calculate error metrics
> mae <- mean(abs(results$Actual - results$Predicted))  # Mean Absolute Error
> mse <- mean((results$Actual - results$Predicted)^2)   # Mean Squared Error
> rmse <- sqrt(mse)  # Root Mean Squared Error
>
> # Print error metrics
> cat("Mean Absolute Error (MAE):", mae, "\n")
Mean Absolute Error (MAE): 0.3161889
> cat("Mean Squared Error (MSE):", mse, "\n")
Mean Squared Error (MSE): 0.1419489
> cat("Root Mean Squared Error (RMSE):", rmse, "\n")
Root Mean Squared Error (RMSE): 0.376761
>
```

Step 6: Visualizing Regression Line

- Scatter plot **(Petal Length vs Sepal Length)**
- Add regression line using ggplot2

```
# Plotting the regression line using the test data
ggplot(test_data, aes(x = Petal.Length, y = Sepal.Length)) +
 geom_point(color = "blue", size = 2) +  # Scatter plot
 geom_smooth(method = "lm", formula = y ~ x, color = "red") +  # Regression line
 labs(title = "Petal Length vs. Sepal Length (Test Data)",
     x = "Petal Length (cm)",
     y = "Sepal Length (cm)") +
 theme_minimal()
```



Step 7: Making Predictions for New Data

- Predict Sepal Length for new Petal Length values (e.g., **1.5, 3.0, 4.5 cm**)

```
new_petal_length <- c(1.5, 3.0, 4.5)

predicted_sepal_length <- predict(model, newdata = data.frame(Petal.Length = new_petal_length))

print(predicted_sepal_length)
```

```
> predicted_sepal_length <- predict(model, newdata = da
h))
> print(predicted_sepal_length)
       1        2        3
4.897891 5.517226 6.136561
>
```

Additional Work:

Applying Regression on Custom Dataset

Step 8: Create Custom Dataset

- Define a dataset representing **Sunlight Hours vs Solar Yield**

```
data <- data.frame(

 Sunlight_Hours = c(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13),
```

```
  Solar_Yield = c(1.8, 2.6, 3.5, 4.4, 5.3, 6.1, 7.0, 8.2, 9.1, 10.0, 11.2, 12.5)
)

print(data)

summary(data)
```

```
> print(data)
   Sunlight_Hours Solar_Yield
1               2         1.8
2               3         2.6
3               4         3.5
4               5         4.4
5               6         5.3
6               7         6.1
7               8         7.0
8               9         8.2
9              10         9.1
10             11        10.0
11             12        11.2
12             13        12.5
> summary(data)
 Sunlight_Hours   Solar_Yield
 Min.   : 2.00   Min.   : 1.800
 1st Qu.: 4.75   1st Qu.: 4.175
 Median : 7.50   Median : 6.550
 Mean   : 7.50   Mean   : 6.808
 3rd Qu.:10.25   3rd Qu.: 9.325
 Max.   :13.00   Max.   :12.500
>
```

Step 9: Compute Regression Manually

- Calculate **Slope (a) and Intercept (b)** using the least squares method

```
# Extract variables

Sunlight_Hours <- data$Sunlight_Hours

Solar_Yield <- data$Solar_Yield


# Calculate Mean

mean_x <- mean(Sunlight_Hours)

mean_y <- mean(Solar_Yield)


# Calculate Slope (a) and Intercept (b)

a <- sum((Sunlight_Hours - mean_x) * (Solar_Yield - mean_y)) /
sum((Sunlight_Hours - mean_x)^2)

b <- mean_y - (a * mean_x)
```

```
# Display slope and intercept
cat("Slope (a):", a, "\n")
cat("Intercept (b):", b, "\n")
```

```
> # Extract variables
> Sunlight_Hours <- data$Sunlight_Hours
> Solar_Yield <- data$Solar_Yield
>
> # Calculate Mean
> mean_x <- mean(Sunlight_Hours)
> mean_y <- mean(Solar_Yield)
>
> # Calculate Slope (a) and Intercept (b)
> a <- sum((Sunlight_Hours - mean_x) * (Solar_Yield - mean_y)) / sum((Sunlight_Hours - mean_x)^
2)
> b <- mean_y - (a * mean_x)
>
> # Display slope and intercept
> cat("Slope (a):", a, "\n")
Slope (a): 0.956993
> cat("Intercept (b):", b, "\n")
Intercept (b): -0.3691142
>
```

- Predict values using the regression equation

```
# Predict values using the regression equation
predicted_y <- a * Sunlight_Hours + b
```

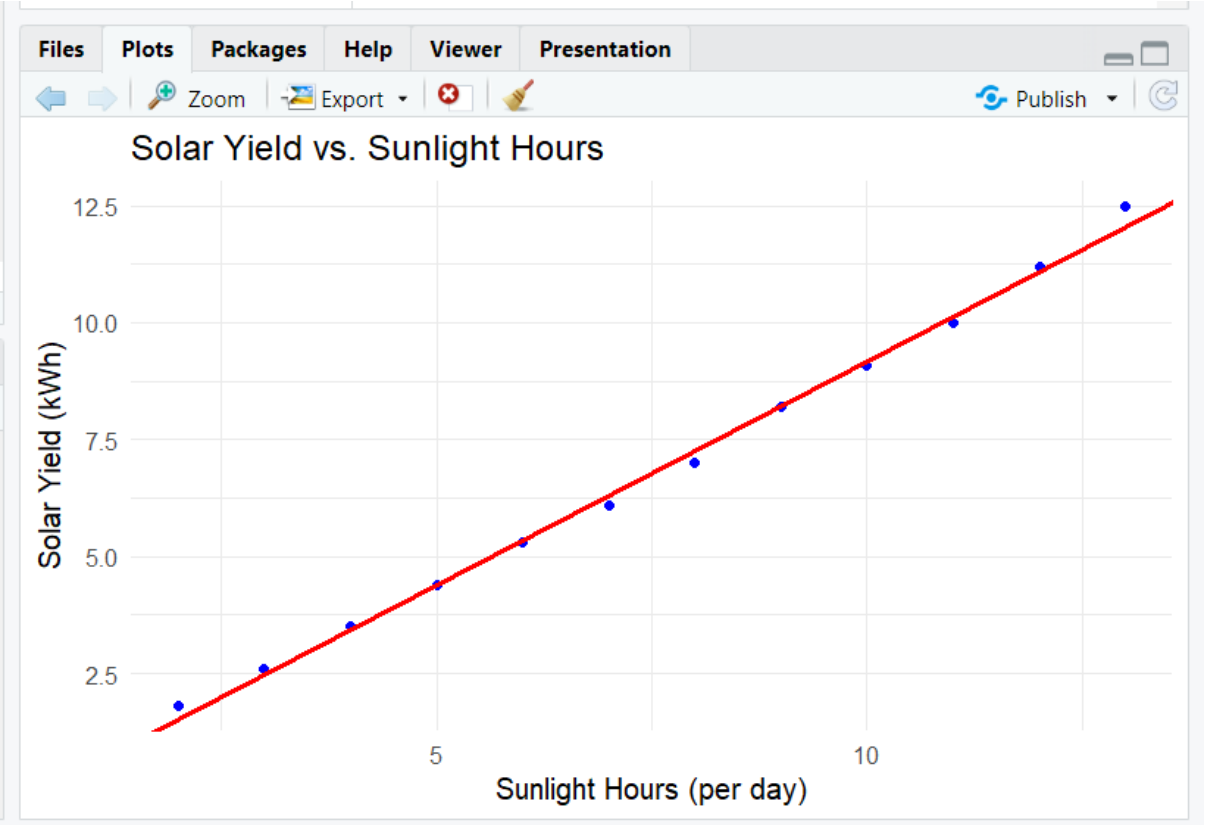
```
Intercept (b): -0.3691142
> # Predict values using the regression equation
> predicted_y <- a * Sunlight_Hours + b
>
```

Step 10: Visualizing Regression Line on Custom Data

- Scatter plot with regression line for **Sunlight Hours vs Solar Yield**

```
# Plot with ggplot
ggplot(data, aes(x = Sunlight_Hours, y = Solar_Yield)) +
  geom_point(color = "blue", linewidth = 2.3) +  # Scatter plot
  geom_abline(intercept = b, slope = a, color = "red", linewidth = 1) +  # Regression line
  labs(title = "Solar Yield vs. Sunlight Hours",
```

```
    x = "Sunlight Hours (per day)",
    y = "Solar Yield (kWh)") +
  theme_minimal()
```



Step 11: Predicting Solar Yield for New Data

- Use the formula to predict Solar Yield for **14, 15, and 16 hours** of sunlight

```
# New Sunlight Hours (for predictions)
new_sunlight_hours <- data.frame(Sunlight_Hours = c(14, 15, 16))
predicted_new_values <- a * new_sunlight_hours$Sunlight_Hours + b
cat("Predicted Solar Yield for", new_sunlight_hours$Sunlight_Hours, "hours of sunlight:", predicted_new_values, "\n")
```

```
  Ignoring unknown parameters:  linewidth
> # New Sunlight Hours (for predictions)
> new_sunlight_hours <- data.frame(Sunlight_Hours = c(14, 15, 16))
> predicted_new_values <- a * new_sunlight_hours$Sunlight_Hours + b
> cat("Predicted Solar Yield for", new_sunlight_hours$Sunlight_Hours, "hours of su
icted_new_values, "\n")
Predicted Solar Yield for 14 15 16 hours of sunlight: 13.02879 13.98578 14.94277
>
```

Step 12: Compare Manual Calculations with lm() Predictions

- Train a model using lm() and compare results with manual predictions

```
# Verify using lm()
model <- lm(Solar_Yield ~ Sunlight_Hours, data = data)
summary(model)
```

```
Call:
lm(formula = Solar_Yield ~ Sunlight_Hours, data = data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.28683 -0.11506 -0.02984  0.08843  0.42821

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    -0.36911    0.14559  -2.535   0.0296 *
Sunlight_Hours  0.95699    0.01763  54.270 1.09e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2109 on 10 degrees of freedom
Multiple R-squared:  0.9966,    Adjusted R-squared:  0.9963
F-statistic:  2945 on 1 and 10 DF,  p-value: 1.093e-13

> |
```

```
# Model Predictions
lm_predictions <- predict(model, newdata = new_sunlight_hours)

comparison <- data.frame(
  Sunlight_Hours = new_sunlight_hours$Sunlight_Hours,
  Manual_Prediction = predicted_new_values,
  LM_Prediction = lm_predictions
)
print(comparison)
```

```
> # Model Predictions
> lm_predictions <- predict(model, newdata = new_sunlight_hours)
>
>
> # Compare manual vs lm() predictions
> comparison <- data.frame(
+   Sunlight_Hours = new_sunlight_hours$Sunlight_Hours,
+   Manual_Prediction = predicted_new_values,
+   LM_Prediction = lm_predictions
+ )
>
>
>
> print(comparison)
  Sunlight_Hours Manual_Prediction LM_Prediction
1             14          13.02879      13.02879
2             15          13.98578      13.98578
3             16          14.94277      14.94277
>
```

Step 13: Calculate Error Metrics

- Compute **MAE, MSE, and RMSE** to evaluate model accuracy

```
# Error Metrics
mae <- mean(abs(Solar_Yield - predicted_y))  # Mean Absolute Error
mse <- mean((Solar_Yield - predicted_y)^2)  # Mean Squared Error
rmse <- sqrt(mse)  # Root Mean Squared Error


cat("\nError Metrics:\n")
cat("MAE:", mae, "\n")
cat("MSE:", mse, "\n")
cat("RMSE:", rmse, "\n")
```

```
> cat("\nError Metrics:\n")

Error Metrics:
> cat("MAE:", mae, "\n")
MAE: 0.1513015
> cat("MSE:", mse, "\n")
MSE: 0.03705614
> cat("RMSE:", rmse, "\n")
RMSE: 0.1924997
>
```

Conclusion

In this lab, we applied Simple Linear Regression to analyze relationships between variables in both the Iris dataset and a custom dataset (Sunlight Hours vs Solar Yield). By training the model on a training dataset and testing it on unseen data, we evaluated its performance using error metrics like MAE, MSE, and RMSE. We manually calculated the regression coefficients and compared them with the results from the lm() function in R, which yielded similar outcomes. Visualizing the regression line helped us better understand the data trends. Overall, this lab strengthened our understanding of linear regression, model evaluation, and the importance of error metrics in assessing prediction accuracy.

LAB-6

Title: Hypothesis testing

Dated: 12.02.2025

Objective:

The objective of this lab is to conduct hypothesis tests, including:

- One-sample t-test to check if the mean Ozone level significantly differs from 50.
- Two-sample t-test to compare Ozone levels between May and June.
- Z-score calculation to determine the standardized test statistic.

Introduction:

Hypothesis testing is a fundamental statistical method used to determine whether there is enough evidence to reject a given assumption (null hypothesis) about a population parameter. In this lab, we will perform hypothesis tests on the AirQuality dataset in R**.**

Lab Work:

Setup.

To perform the hypothesis tests, we will:

1. Load and explore the AirQuality dataset.
2. Handle missing values if necessary.
3. Conduct one-sample and two-sample t-tests.
4. Compute the Z-score for Ozone levels.

1. Load and Explore Dataset

```
data("airquality")
```

```
head(airquality)

summary(airquality)
```

Handling missing values:

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

```
>
> # View first few rows and summary statistics
> head(airquality)
  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
6    28      NA 14.9   66     5   6
> summary(airquality)
     Ozone           Solar.R           Wind             Temp           Month
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   Min.   :5.000
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   1st Qu.:6.000
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   Median :7.000
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   Mean   :6.993
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   3rd Qu.:8.000
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   Max.   :9.000
 NA's   :37       NA's   :7
      Day
 Min.   : 1.0
 1st Qu.: 8.0
 Median :16.0
 Mean   :15.8
 3rd Qu.:23.0
 Max.   :31.0

> |
```

2. Hypothesis Testing: One-Sample t-Test

We check if the mean Ozone level is significantly different from 50.

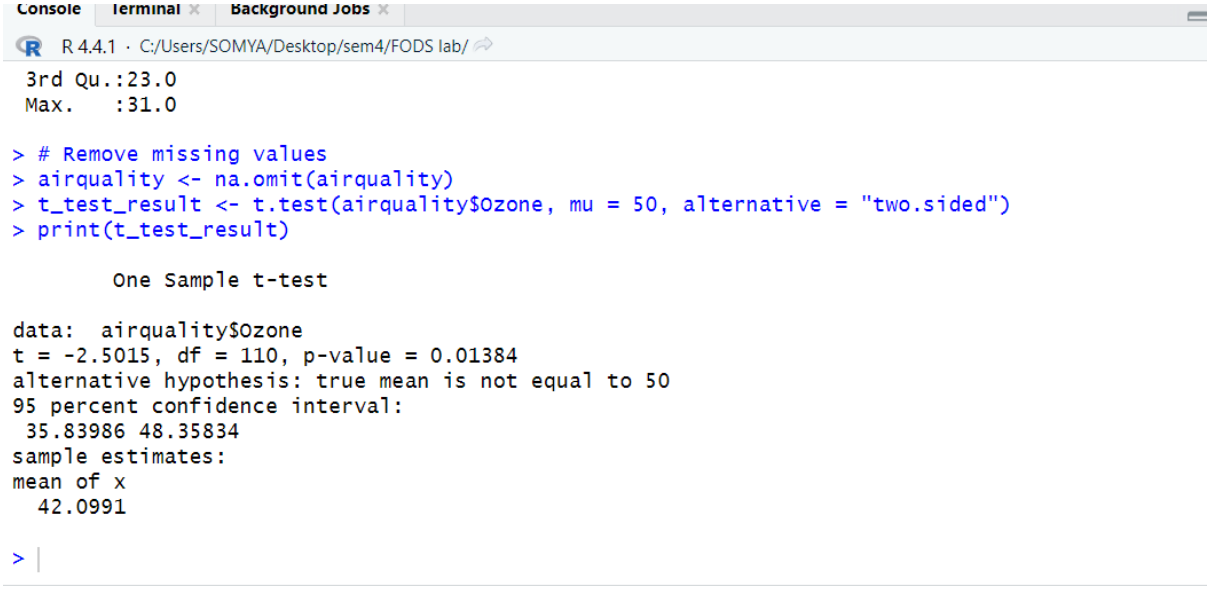
- **Null Hypothesis (H_0):** $\mu = 50$ (Ozone levels are 50 on average)
- **Alternative Hypothesis (H_1):** $\mu \neq 50$ (Ozone levels differ from 50)

Perform t-Test

```
t_test_result <- t.test(airquality$Ozone, mu = 50, alternative = "two.sided")

print(t_test_result)
```

Interpretation:

- If p-value < 0.05, reject H_0 → Ozone levels are significantly different from 50.
- Otherwise, fail to reject H_0.



3. Hypothesis Testing: Two-Sample t-Test

We compare Ozone levels between **May and June**.

- **Null Hypothesis (H_0):** $\mu_1 = \mu_2$ (Ozone levels in May and June are the same)
- **Alternative Hypothesis (H_1):** $\mu_1 \neq \mu_2$ (Ozone levels in May and June differ)

Perform t-Test

```
may_ozone <- airquality$Ozone[airquality$Month == 5]
june_ozone <- airquality$Ozone[airquality$Month == 6]

t_test_months <- t.test(may_ozone, june_ozone, var.equal = TRUE)
print(t_test_months)
```

Interpretation:

- If p-value < 0.05, reject H_0 → Significant difference in Ozone levels.
- Otherwise, fail to reject H_0.

```
> # Perform two-sample t-test
> t_test_months <- t.test(may_ozone, june_ozone, var.equal = TRUE)
> print(t_test_months)

        Two Sample t-test

data:  may_ozone and june_ozone
t = -0.62499, df = 31, p-value = 0.5365
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -22.67815  12.03927
sample estimates:
mean of x mean of y
 24.12500  29.44444

> |
```

4. Z-Score Calculation (Standardized Test Statistic)

Z-score helps understand how far a sample mean is from the population mean in terms of standard deviation.

Formula:

$$Z = X - \mu\sigma Z \ = \backslash frac\{X \ - \backslash mu\}\{\backslash sigma\}Z = \sigma X - \mu$$

Where:

- XXX = Sample Mean
- μ\mu = Hypothesized Mean
- σ\sigma = Standard Deviation

Calculate Z-Score for Ozone Levels

```
sample_mean <- mean(airquality$Ozone)

population_mean <- 50

std_dev <- sd(airquality$Ozone)

n <- length(airquality$Ozone)


z_score <- (sample_mean - population_mean) / (std_dev / sqrt(n))

z_score
```

Interpretation:

- If $|Z| > 1.96$ (for 95% confidence level), reject H_0.
- Otherwise, fail to reject H_0.

```
sample estimates:
mean of x mean of y
 24.12500  29.44444

> # Z-score Formula: Z = (x̄ - μ) / (σ / sqrt(n))
> population_mean <- 50  # Assumed population mean for hypothesis testing
>
> z_score <- (sample_mean - population_mean) / (std_dev / sqrt(n))
> cat("Z-Score:", z_score, "\n")
Z-Score: -2.50154
>
> # Interpretation: If |Z| > 1.96 (for 95% confidence level), reject H₀.
>
```

Conclusion

In this lab, we conducted various hypothesis tests to analyze the Ozone levels in the AirQuality dataset. First, we performed a one-sample t-test to determine if the mean Ozone level significantly differed from 50, and based on the p-value, we decided whether to reject or fail to reject the null hypothesis. Next, we used a two-sample t-test to compare Ozone levels between May and June to check if there was a significant difference between the two months. Additionally, we calculated the Z-score for the Ozone levels to determine how far the sample mean was from the hypothesized mean in terms of standard deviations. The results of these tests provided insights into whether the Ozone levels were significantly different from the expected values or between the two months. This lab reinforced the practical application of hypothesis testing, helping us understand how to evaluate data and make informed conclusions about population parameters.

LAB- 7

Title: Impact of Feature Engineering on Model Accuracy: A Study Using Crop Recommendation Dataset

Dated: 26.03.2025

Objective:

The objective of this lab is to explore the impact of feature engineering on model performance, specifically by examining how dropping certain features influences the accuracy of a machine learning model. Using the Crop Recommendation Dataset, the task involves analyzing the effects of removing specific features on the performance of the Random Forest Regressor and Linear Regression models.

Introduction:

Feature engineering is a crucial step in the machine learning pipeline, where raw data is transformed into a set of relevant features that improve model performance. One common approach to feature engineering is dropping certain features to see how their absence affects the model's predictions. In this lab, we examine how removing different features—such as 'Phosphorus', 'Potassium', and 'pH Value'—affects the performance of the model, particularly focusing on metrics like R2 Score, Mean Absolute Error (MAE), and Mean Squared Error (MSE).

Lab Work:

Data set used: Crop Recommendation Data, picked up from Kaggle.com.

Setup: Worked in google collab environment. First few steps included the loading of the dataset and basic study about the summary of the data.

```
recommendation_data.shape
```

```
(2200, 8)
```

```
recommendation_data.head()
```

	Nitrogen	Phosphorus	Potassium	Temperature	Humidity	pH_Value	Rainfall	Crop
0	90	42	43	20.879744	82.002744	6.502985	202.935536	Rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	Rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	Rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	Rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	Rice

```
recommendation_data.describe()
```

	Nitrogen	Phosphorus	Potassium	Temperature	Humidity	pH_Value	Rainfall
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	50.551818	53.362727	48.149091	25.616244	71.481779	6.469480	103.463655
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938	54.958389
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752	20.211267
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693	64.551686
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045	94.867624
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643	124.267508
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091	298.560117

Step 1: Label Encoding of Categorical Variables

- The Crop Recommendation Dataset includes categorical values for the crop types. We performed label encoding to convert these categorical labels into numerical values, allowing the model to interpret them effectively.
- Before encoding, the unique crop values were categorical (e.g., 'Rice', 'Maize'), and after encoding, they were represented by integer values (e.g., 20, 11).

```
print("\n Unique Crop Values Before Encoding:\n", recommendation_data["Crop"].unique())
```

```
Unique Crop Values Before Encoding:
['Rice' 'Maize' 'ChickPea' 'KidneyBeans' 'PigeonPeas' 'MothBeans'
'MungBean' 'Blackgram' 'Lentil' 'Pomegranate' 'Banana' 'Mango' 'Grapes'
'Watermelon' 'Muskmelon' 'Apple' 'Orange' 'Papaya' 'Coconut' 'Cotton'
'Jute' 'Coffee']
```

```
[ ] # Apply Label Encoding
label_encoder = LabelEncoder()
recommendation_data ["Crop"] = label_encoder.fit_transform(recommendation_data ["Crop"])

print("\n Unique Crop Values After Encoding:\n", recommendation_data ["Crop"].unique())
```

```
Unique Crop Values After Encoding:
[20 11  3  9 18 13 14  2 10 19  1 12  7 21 15  0 16 17  4  6  8  5]
```

Step 2: Model Training with All Features

- The first model was trained using all available features in the dataset. We used the **RandomForestRegressor** to fit the model on the training data and evaluated its performance using the test set. Key metrics like R2 Score, MAE, and MSE were computed to assess the model's accuracy.
-

```
[ ] # Model 1: All Features
model1 = RandomForestRegressor()
model1.fit(X_train, y_train)
y_pred1 = model1.predict(X_test)
```

```
[ ] # Model 2: Drop Two Features (Drop 'Phosphorus' & 'Potassium')
X_train2, X_test2 = X_train.drop(columns=["Phosphorus", "Potassium"]), X_test.drop(columns=["Phosphorus", "Potassium"])
model2 = RandomForestRegressor()
model2.fit(X_train2, y_train)
y_pred2 = model2.predict(X_test2)
```

```
[ ] # Model 3: Drop One More Feature (Drop 'pH_Value' additionally)
X_train3, X_test3 = X_train2.drop(columns=["pH_Value"]), X_test2.drop(columns=["pH_Value"])
model3 = RandomForestRegressor()
model3.fit(X_train3, y_train)
y_pred3 = model3.predict(X_test3)
```

Step 3: Dropping Features and Model Comparison

- **Model 2**: We dropped the 'Phosphorus' and 'Potassium' features and trained a new model. The model's performance was compared with the first model to observe how the removal of these features affected accuracy.
- **Model 3**: Further, we dropped the 'pH_Value' feature from the dataset and retrained the model. The performance was again assessed to see how removing another feature impacted the model's predictive ability.

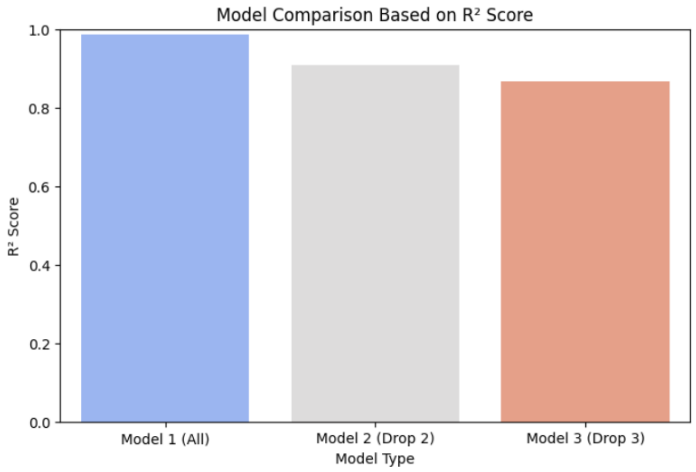
```
[ ]
    # Print table
    # Compute metrics
    print("\n Model Performance Metrics:")
    print(tabulate(metrics_data, headers=["Model", "R2 Score", "MAE", "MSE"], tablefmt="pretty"))
```

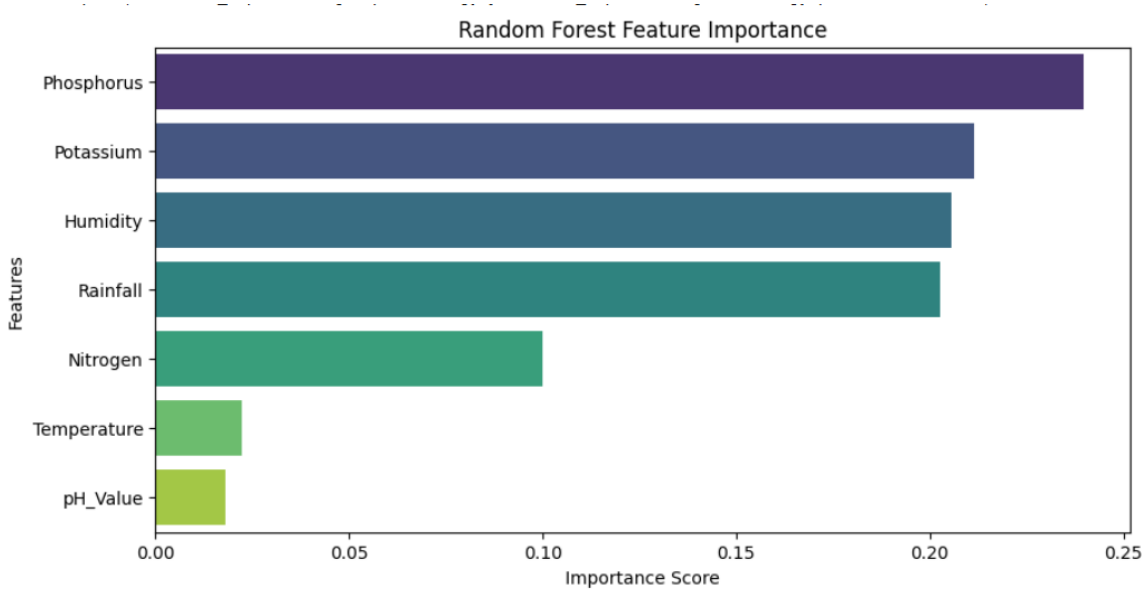
```
 Model Performance Metrics:
+---------------------------+----------+--------+--------+
|           Model           | R2 Score |  MAE   |  MSE   |
+---------------------------+----------+--------+--------+
|  Model 1 (All Features)   |  0.9875  | 0.2013 | 0.4938 |
| Model 2 (Drop 2 Features) |  0.9088  | 0.6125 | 3.6097 |
| Model 3 (Drop 3 Features) |  0.8666  | 0.8378 | 5.2775 |
+---------------------------+----------+--------+--------+
```

Step 4: Model Evaluation and Metrics Calculation

- For each model, we calculated the R2 Score, Mean Absolute Error (MAE), and Mean Squared Error (MSE). These metrics helped evaluate the models' ability to predict crop recommendations and their generalization on unseen data.

```
<ipython-input-43-e28ba382caec>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=

  sns.barplot(x=model_names, y=r2_scores, palette="coolwarm")
```





Conclusion

In this lab, we explored the impact of feature engineering on model accuracy by dropping specific features from the dataset. Initially, the model trained with all features performed the best with the highest R2 Score and the lowest error metrics. As we dropped more features (Phosphorus, Potassium, and pH_Value), the model's accuracy gradually decreased, as shown by the declining R2 Score and increasing MAE and MSE. This demonstrates the importance of feature selection in machine learning and highlights how removing critical features can negatively impact model performance.

LAB- 8

Title: Clustering on Iris Dataset using K-Means Algorithm

Dated: 26.03.2025

Objective:

The objective of this lab is to explore the impact of feature engineering on model performance, specifically by examining how dropping certain features influences the accuracy of a machine learning model. Using the Crop Recommendation Dataset, the task involves analyzing the effects of removing specific features on the performance of the Random Forest Regressor and Linear Regression models.

Introduction:

Feature engineering is a crucial step in the machine learning pipeline, where raw data is transformed into a set of relevant features that improve model performance. One common approach to feature engineering is dropping certain features to see how their absence affects the model's predictions. In this lab, we examine how removing different features—such as 'Phosphorus', 'Potassium', and 'pH Value'—affects the performance of the model, particularly focusing on metrics like R2 Score, Mean Absolute Error (MAE), and Mean Squared Error (MSE).

Lab Work:

Step 1: Loading the Dataset and Summary

```
data(iris)
summary(data)
```

```
Console   Terminal   Background Jobs
R 4.4.1 · C:/Users/SOMYA/Desktop/sem4/FODS lab/

[Workspace loaded from C:/Users/SOMYA/Desktop/sem4/FODS lab/.RData]

> data(iris)
> summary(data)
    pregnant         glucose         pressure         triceps         insulin
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00   Min.   :  0.0
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00   1st Qu.:  0.0
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00   Median : 30.5
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54   Mean   : 79.8
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00   3rd Qu.:127.2
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00   Max.   :846.0
      mass           pedigree           age         diabetes
 Min.   : 0.00   Min.   :0.0780   Min.   :21.00   neg:500
 1st Qu.:27.30   1st Qu.:0.2437   1st Qu.:24.00   pos:268
 Median :32.00   Median :0.3725   Median :29.00
 Mean   :31.99   Mean   :0.4719   Mean   :33.24
 3rd Qu.:36.60   3rd Qu.:0.6262   3rd Qu.:41.00
 Max.   :67.10   Max.   :2.4200   Max.   :81.00
>
```

Step 2: Data Preparation

```
iris_data <- iris[, -5]
```

Step 3: Applying K-Means Clustering

```
set.seed(123)

k_result <- kmeans(iris_data, centers = 3, nstart = 20)
```

Step 4: Assigning Cluster Labels to Original Dataset

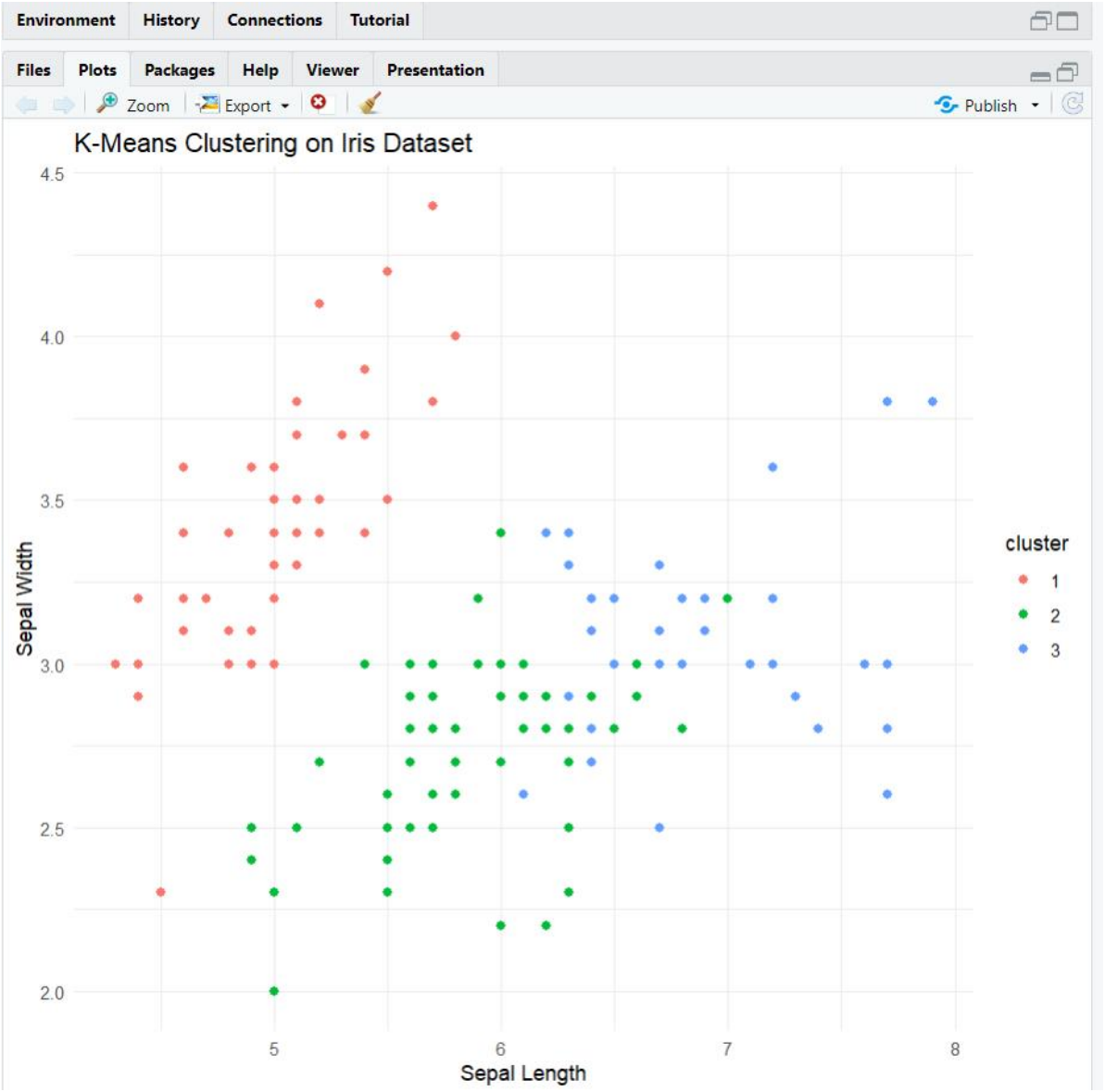
```
iris$cluster <- as.factor(k_result$cluster)
```

```
> iris_data <- iris[, -5]
> set.seed(123)
> k_result <- kmeans(iris_data, centers = 3, nstart = 20)
> iris$cluster <- as.factor(k_result$cluster)
> library(ggplot2)
Warning message:
package 'ggplot2' was built under R version 4.4.2
> # Plotting using Sepal features
> ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = cluster)) +
+   geom_point(size = 2) +
+   labs(title = "K-Means Clustering on Iris Dataset", x = "Sepal Length", y = "Sepal Width")
+
+   theme_minimal()
>
```

Step 5: Visualization

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = cluster)) +
 geom_point(size = 2) +
```

```
        labs(title = "K-Means Clustering on Iris Dataset", x = "Sepal Length", y = "Sepal
Width") +
        theme_minimal()
```



Conclusion:

Through this lab, we successfully applied the K-Means clustering algorithm on the Iris dataset and visualized the results. Despite not using the actual species labels, the algorithm effectively grouped similar flowers based on their features. This lab demonstrated how unsupervised learning can uncover meaningful patterns in data and reinforced our understanding of clustering techniques in R.

LAB- 9

Title: Association Rule Mining on Transaction Data using Support and Confidence Metrics

Dated: 02.04.2025

Objective:

To understand the concept of **association rules** and key metrics: support and confidence.

Introduction:

Association Rule Mining is a popular data mining technique used to uncover interesting relationships, patterns, or correlations between items in large datasets. One of the most well-known applications of this method is **market basket analysis**, which helps understand which items are frequently purchased together.

Two primary metrics used to evaluate the strength of an association rule are:

Support: How often a particular item or itemset appears in the dataset.

Confidence: Given that an item A is bought, how often item B is also bought.

In this lab, we implemented a simple market basket analysis on a custom transaction dataset using R, and manually computed support and confidence values to derive meaningful rules.

Lab Work:

Step 1: Creating the Transaction Dataset

```
transactions <- list(
 c("Shirt", "Trouser", "Tie"),
 c("Shirt", "Trouser"),
 c("Trouser", "Socks"),
 c("Shirt", "Tie"),
```

```
  c("Shirt", "Trouser", "Socks"),
  c("Shirt", "Tie", "Trouser"),
  c("Socks", "Trouser")
 )
```

Step 2: Defining the Support Function

```
calculate_support <- function(itemset, transactions) {
  count <- sum(sapply(transactions, function(tr) all(itemset %in% tr)))
  return(count / length(transactions))
}
```

Step 3: Defining the Confidence Function

```
calculate_confidence <- function(A, B, transactions) {
  support_AB <- calculate_support(c(A, B), transactions)
  support_A <- calculate_support(A, transactions)
  return(support_AB / support_A)
}
```

Step 4: Setting Thresholds

```
support_threshold <- 0.3       # 30%
confidence_threshold <- 0.5    # 50%
```

Step 5: Rule Generation and Evaluation

```
all_items <- unique(unlist(transactions))
rules <- list()
 for (from_item in all_items) {
  for (to_item in all_items) {
    if (from_item != to_item) {
```

```
  # Calculate Metrics
  support_from <- calculate_support(from_item, transactions)
  support_to <- calculate_support(to_item, transactions)
  support_both <- calculate_support(c(from_item, to_item), transactions)
  confidence <- calculate_confidence(from_item, to_item, transactions)

  # Check thresholds
  if (support_both > support_threshold && confidence > confidence_threshold) {

    # Store rule
    rules <- append(rules, list(list(
      from_item = from_item,
      to_item = to_item,
      support_from = round(support_from, 2),
      support_to = round(support_to, 2),
      support_both = round(support_both, 2),
      confidence = round(confidence, 2)
    )))

    #  Print Rule Details
    cat("=====================================\n")
    cat("Rule: ", from_item, "→", to_item, "\n")
    cat("Support of", from_item, ":", round(support_from, 2), "\n")
    cat("Support of", to_item, ":", round(support_to, 2), "\n")
    cat("Support of (", from_item, ",", to_item, "):", round(support_both, 2), "\n")
    cat("Confidence (", from_item, "→", to_item, "):", round(confidence, 2), "\n")
    cat("=====================================\n\n")
  }
}
```

```
            }

          }
```

```
====================================
Rule:  Shirt → Trouser
Support of Shirt : 0.71
Support of Trouser : 0.86
Support of ( Shirt , Trouser ): 0.57
Confidence ( Shirt → Trouser ): 0.8
====================================

====================================
Rule:  Shirt → Tie
Support of Shirt : 0.71
Support of Tie : 0.43
Support of ( Shirt , Tie ): 0.43
Confidence ( Shirt → Tie ): 0.6
====================================


====================================
Rule:  Trouser → Shirt
Support of Trouser : 0.86
Support of Shirt : 0.71
Support of ( Trouser , Shirt ): 0.57
Confidence ( Trouser → Shirt ): 0.67
====================================

====================================
Rule:  Tie → Shirt
Support of Tie : 0.43
Support of Shirt : 0.71
Support of ( Tie , Shirt ): 0.43
Confidence ( Tie → Shirt ): 1
====================================
------------------------------------

====================================
Rule:  Socks → Trouser
Support of Socks : 0.43
Support of Trouser : 0.86
Support of ( Socks , Trouser ): 0.43
Confidence ( Socks → Trouser ): 1
====================================

>
```

Conclusion:

This lab successfully demonstrated how to implement **association rule mining** from scratch using basic R programming. By calculating support and confidence manually, we understood how rules are evaluated and selected based on defined thresholds. This foundational approach forms the basis for advanced techniques used in real-world recommendation engines and market analysis.

LAB- 10

Title: Classification using Decision Trees

Dated: 09.04.2025

Objective: To implement classification using decision trees in R.

- To visualize the decision tree using rpart.plot.
- To evaluate prediction accuracy using a confusion matrix.
- To understand the decision-making process behind node selection (e.g., why a certain feature becomes the root).

Introduction:

Decision Trees are powerful and intuitive tools for classification and regression tasks. They work by splitting the data into subsets based on the value of input features, forming a tree-like structure of decisions. At each node, the algorithm chooses the feature that best separates the classes using metrics like **Gini index** or **Information Gain**.

Lab Work:

Step 1: Load Libraries and Dataset

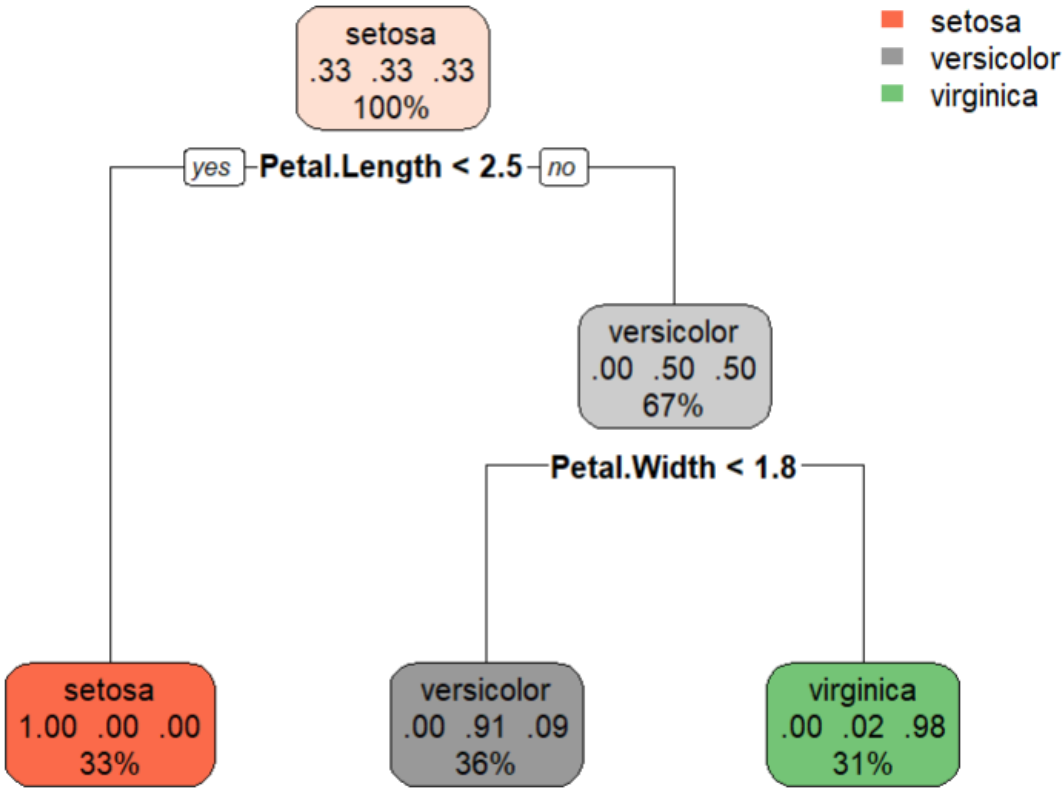
```
library(rpart)
library(rpart.plot)
data(iris)
head(iris)
```

Step 2: Building the Decision Tree Model

```
tree_model <- rpart(Species ~ ., data = iris, method = "class")
summary(tree_model)
```

Step 3: Visualizing the Tree

```
rpart.plot(tree_model)
```



📝 **Note:**
In the Iris dataset, **Petal.Length** was chosen as the **root node** because it provides the **highest information gain**. It splits the data most effectively, especially distinguishing the **Setosa** class clearly (e.g., if Petal.Length < 2.45, it's almost always Setosa).

Conclusion:

This lab gave hands-on experience with decision trees using R's rpart package. We explored classification using two real-world datasets, understood how decision nodes are selected, and visualized the learning process. The exercise also highlighted how powerful tree-based models can be in identifying patterns and making predictions with high interpretability.

Final Conclusion

Completing this series of labs has been a valuable learning experience. Each lab introduced a new concept or tool in R programming and data science, helping to build both theoretical understanding and practical skills. From data cleaning and visualization to machine learning techniques like clustering and decision trees, this manual reflects the progression of knowledge gained throughout the course.

Working with real-world datasets and writing code to solve problems has made me more confident in my ability to analyze data and apply machine learning techniques. I now have a much stronger foundation in R and data analysis, which I look forward to building upon in future projects and studies.

FUNDAMENTALS OF DATA SCIENCE LAB

CODE_CSDS2001P

LAB MANUAL REPORT FILE

THE END

Name: Somya Vats

Roll Number: R2142230349

SAP ID: 500119012

Batch: 1_Data Science

Semester- 4

Submitted to Prof. Neeraj Chugh dated 26 April'2025