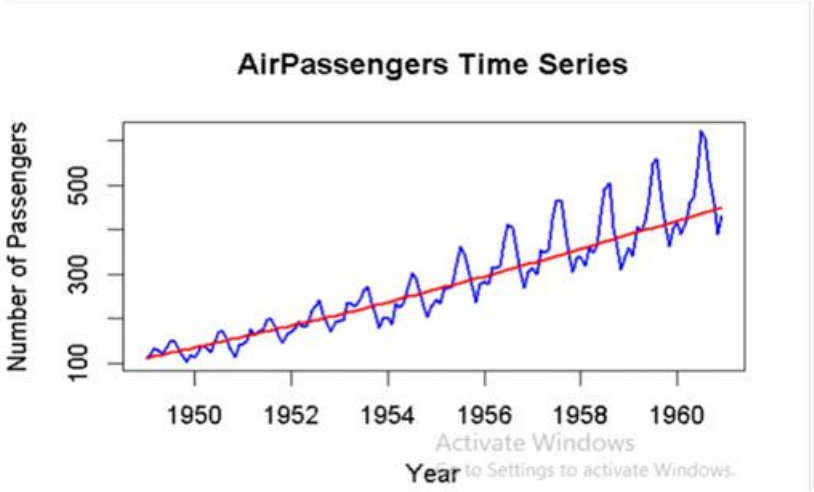
12 a)

Aim: a) Plot the Time series data using R visualizations.

Code:

```
data(AirPassengers)
ts_data <- AirPassengers
plot(ts_data,
    main = "AirPassengers Time Series",
    ylab = "Number of Passengers",
    xlab = "Year",
    col = "blue",
    lwd = 2)
lines(lowess(time(ts_data), ts_data), col = "red", lwd = 2)
```

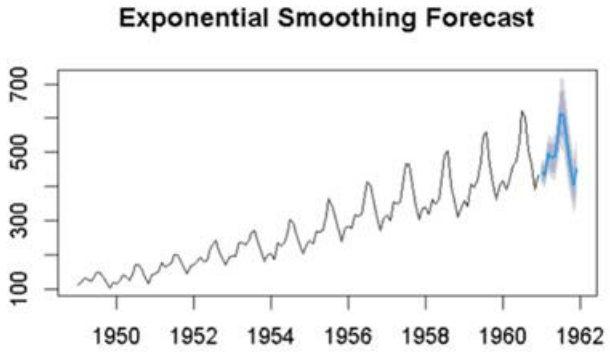
Output:

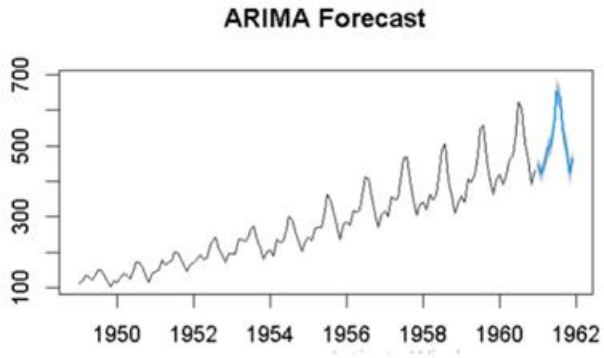


12 b)

```
library(forecast)
data(AirPassengers)
ts_data <- AirPassengers
# Exponential Smoothing
fit_ets <- ets(ts_data)
forecast_ets <- forecast(fit_ets, h = 12)
plot(forecast_ets, main = "Exponential Smoothing Forecast")
# ARIMA Model
fit_arima <- auto.arima(ts_data)
forecast_arima <- forecast(fit_arima, h = 12)
plot(forecast_arima, main = "ARIMA Forecast")
```

Output:



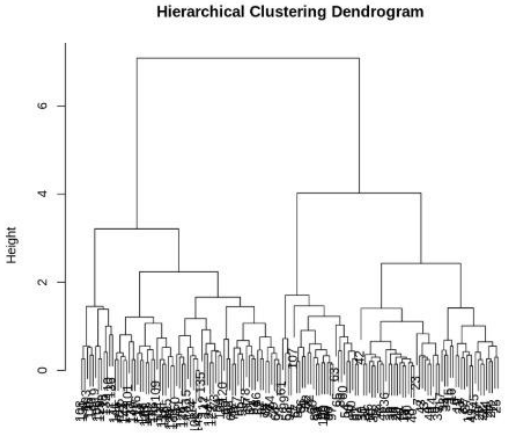


11 a)

```
data(iris)
iris_data <- iris[, -5]

set.seed(123)
km_model <- kmeans(iris_data, centers = 3, nstart = 20)

print(km_model)
table(km_model$cluster)

dist_matrix <- dist(iris_data)

hc_model <- hclust(dist_matrix, method = "complete")

plot(hc_model, main = "Hierarchical Clustering Dendrogram")

clusters <- cutree(hc_model, k = 3)

table(clusters)
```

```
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"

 1  2  3
50 62 38
clusters
 1  2  3
50 72 28
```



11 b)

```r
# ================================
# 3. VISUALIZATION FOR K-MEANS
# ================================

library(ggplot2)

# Add K-means cluster labels to iris dataset
iris$KMeans_Cluster <- as.factor(km_model$cluster)

# Scatter plot using Sepal Length vs Sepal Width
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = KMeans_Cluster)) +
  geom_point(size = 2) +
  labs(
    title = "K-Means Clustering of Iris Data",
    x = "Sepal Length",
    y = "Sepal Width"
  ) +
  theme_minimal()


# ================================
# 4. VISUALIZATION FOR HIERARCHICAL CLUSTERING
# ================================

# Add hierarchical cluster labels to iris dataset
iris$HC_Cluster <- as.factor(clusters)

# Scatter plot for hierarchical clustering
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = HC_Cluster)) +
  geom_point(size = 2) +
  labs(
    title = "Hierarchical Clustering of Iris Data",
    x = "Sepal Length",
    y = "Sepal Width"
  ) +
  theme_minimal()
```

Load the Iris dataset

----

data(iris)

iris_data <- iris[, -5] # Remove Species column for unsupervised clustering

```
# ==============================
# 1. K-MEANS CLUSTERING
# ==============================

set.seed(123)   # For reproducible results

# Run K-means with 3 clusters
km_model <- kmeans(iris_data, centers = 3, nstart = 20)

# Print details of the k-means model
print(km_model)

# Show how many points are in each cluster
table(km_model$cluster)



# ==============================
# 2. HIERARCHICAL CLUSTERING
# ==============================

# Compute distance matrix
dist_matrix <- dist(iris_data)

# Perform hierarchical clustering using complete linkage
hc_model <- hclust(dist_matrix, method = "complete")
```

```
# Plot the dendrogram
plot(hc_model, main = "Hierarchical Clustering Dendrogram")

# Cut tree into 3 clusters
clusters <- cutree(hc_model, k = 3)

# Count cluster sizes
table(clusters)



# ==============================
# 3. VISUALIZATION FOR K-MEANS
# ==============================

library(ggplot2)

# Add K-means cluster labels to iris dataset
iris$KMeans_Cluster <- as.factor(km_model$cluster)

# Scatter plot using Sepal Length vs Sepal Width
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = KMeans_Cluster)) +
  geom_point(size = 2) +
  labs(
```

```
    title = "K-Means Clustering of Iris Data",
    x = "Sepal Length",
    y = "Sepal Width"
  ) +
  theme_minimal()



# ==============================
# 4. VISUALIZATION FOR HIERARCHICAL CLUSTERING
# ==============================

# Add hierarchical cluster labels to iris dataset
iris$HC_Cluster <- as.factor(clusters)

# Scatter plot for hierarchical clustering
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = HC_Cluster)) +
  geom_point(size = 2) +
  labs(
    title = "Hierarchical Clustering of Iris Data",
    x = "Sepal Length",
    y = "Sepal Width"
  ) +
  theme_minimal()
```

week 10 a and b

```
# ============================================
#  Load Required Libraries
# ============================================
# Install packages ONCE if not installed:
# install.packages(c("caret", "e1071", "rpart", "randomForest", "nnet"))

library(caret)         # For train-test split & confusionMatrix
library(e1071)         # For Naive Bayes & SVM
library(randomForest)   # For Random Forest
library(rpart)         # For Decision Tree
library(nnet)          # For Multinomial Logistic Regression

# ============================================
#  Load Dataset
# ============================================
data(iris)         # Load the iris dataset
set.seed(123)      # Ensures reproducible results

# ============================================
#  Train-Test Split (70% Train, 30% Test)
# ============================================
trainIndex <- createDataPartition(iris$Species, p = 0.7, list = FALSE)
```

```
train <- iris[trainIndex, ]     # Training data
test  <- iris[-trainIndex, ]    # Testing data

# ============================================
#  1. Decision Tree Model
# ============================================
dt_model <- rpart(Species ~ ., data = train, method = "class")
dt_pred  <- predict(dt_model, test, type = "class")

# ============================================
#  2. Random Forest Model
# ============================================
rf_model <- randomForest(Species ~ ., data = train)
rf_pred  <- predict(rf_model, test)

# ============================================
#  3. Support Vector Machine (SVM)
# ============================================
svm_model <- svm(Species ~ ., data = train)
svm_pred  <- predict(svm_model, test)

# ============================================
#  4. Naive Bayes Model
# ============================================
nb_model <- naiveBayes(Species ~ ., data = train)
```

```
nb_pred  <- predict(nb_model, test)

# ============================================
#  5. Multinomial Logistic Regression
# ============================================
log_model <- multinom(Species ~ ., data = train)
log_pred  <- predict(log_model, test)

# ============================================
#  Evaluation Function (Accuracy, Precision, Recall, F1)
# ============================================
evaluate <- function(actual, pred, name) {
  cat("=== ", name, " ===\n")
  cm <- confusionMatrix(pred, actual)          # Confusion Matrix
  print(cm$overall['Accuracy'])                # Accuracy
  print(cm$byClass[, c("Precision", "Recall", "F1")])  # Metrics
  cat("\n")
}

# ============================================
#  Evaluate All Models
# ============================================
evaluate(test$Species, dt_pred,  "Decision Tree")
evaluate(test$Species, rf_pred,  "Random Forest")
evaluate(test$Species, svm_pred, "SVM")
```

evaluate(test$Species, nb_pred, "Naive Bayes")

evaluate(test$Species, log_pred, "Logistic Regression")

---

week 8 a)

```
require(foreign)
require(MASS)
url <- "https://stats.idre.ucla.edu/stat/data/binary.csv"
admission_data <- read.csv(url)
admission_data$rank <- as.factor(admission_data$rank)
head(admission_data)
set.seed(123)
sample_index <- sample(1:nrow(admission_data), 0.7 * nrow(admission_data))
train <- admission_data[sample_index, ]
test  <- admission_data[-sample_index, ]
log_model <- glm(admit ~ gre + gpa + rank, data = train, family = binomial)
summary(log_model)

p_value <- with(log_model, pchisq(null.deviance - deviance,
                                  df.null - df.residual,
                                  lower.tail = FALSE))
cat("\nModel Fit (Deviance Test) p-value:", p_value, "\n")

predicted_prob <- predict(log_model, newdata = test, type = "response")
predicted_class <- ifelse(predicted_prob > 0.5, 1, 0)
cat("\nConfusion Matrix:\n")
print(table(Predicted = predicted_class, Actual = test$admit))

accuracy <- mean(predicted_class == test$admit)
cat("\nModel Accuracy:", round(accuracy * 100, 2), "%\n")
```

week 8 b)

Aim: b) Apply regression Model to predict the data on any dataset.

Code:

```
require(MASS)
data(mtcars)
head(mtcars)

set.seed(123)
sample_index <- sample(1:nrow(mtcars), 0.7 * nrow(mtcars))
train <- mtcars[sample_index, ]
test  <- mtcars[-sample_index, ]

#Robust Regression Model
robust_model <- rlm(mpg ~ wt + hp + cyl, data = train)

summary(robust_model)

predictions <- predict(robust_model, newdata = test)

#Compare actual vs predicted
results <- data.frame(Actual = test$mpg, Predicted = predictions)
print(results)

mse <- mean((results$Actual - results$Predicted)^2)
cat("\nMean Squared Error (MSE):", round(mse, 3), "\n")
```

Week 9 a)

Aim: a) Develop Multiple Regression model for any dataset

Code:

```
data(mtcars)  # Load built-in dataset

# Build a multiple regression model
multi_model <- lm(mpg ~ wt + hp + disp, data = mtcars)

# Display summary of model
summary(multi_model)
```

Aim: b) Apply Multiple Regression Model to predict the data on any dataset.

Code:

```
new_car <- data.frame(wt = 3.0, hp = 120, disp = 200)

# Predict using the fitted model
predicted_mpg <- predict(multi_model, new_car)
predicted_mpg
```

Output:

```
> # Predict using the fitted model
> predicted_mpg <- predict(multi_model, new_car)
> predicted_mpg
       1
21.77665
>
```

week 7

a)

Aim: Correlation and Covariance:
a) Develop a program to find the correlation matrix on iris data.

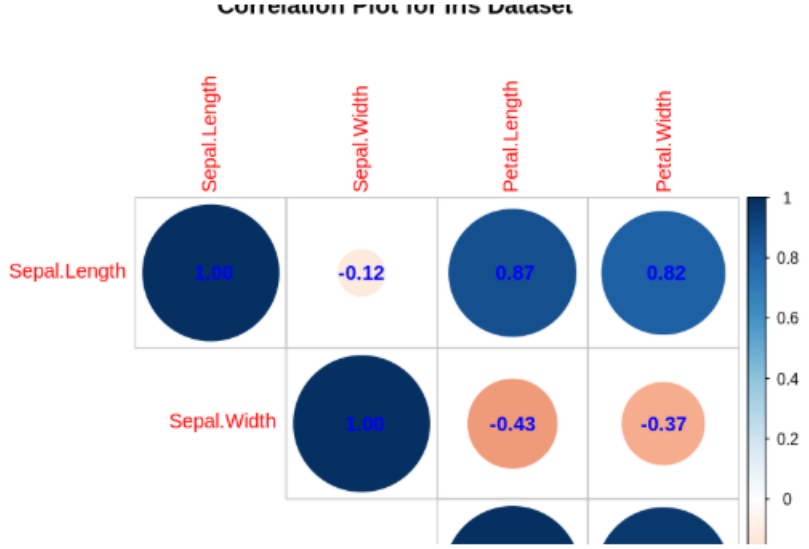
Code:

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

# Select only the numeric columns for correlation analysis
iris_numeric <- iris[, 1:4]

cor_matrix <- cor(iris_numeric)
print(cor_matrix)
```

b)

```
install.packages("corrplot")
library(corrplot)
cor_matrix <- cor(iris[, 1:4])
corrplot(cor_matrix, method="circle", type="upper",
addCoef.col="blue", title="Correlation Plot for Iris Dataset")
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)
```



c)

Aim: c) Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.

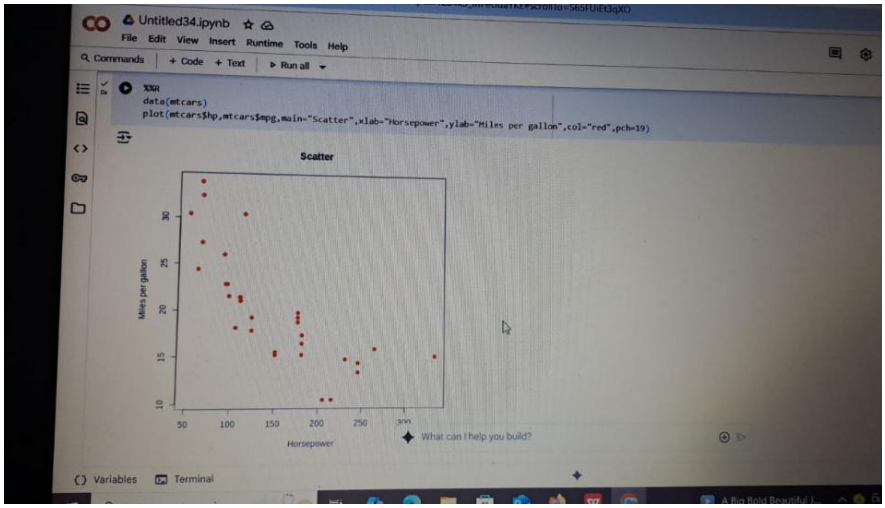
Code:

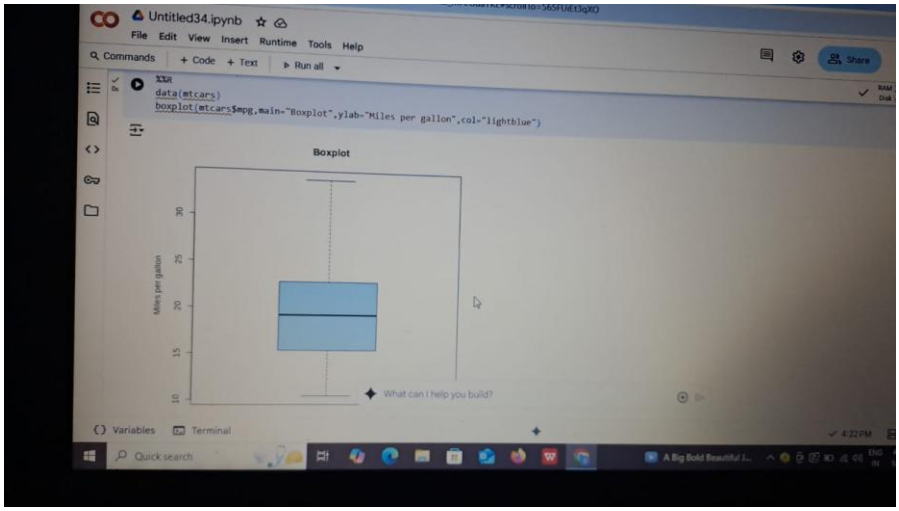
```
# Load the iris dataset
data(iris)

# Perform ANOVA to test if Sepal.Length differs by Species
anova_result <- aov(Sepal.Length ~ Species, data=iris)

# Display the summary of the ANOVA
summary(anova_result)
```

Output:

```
data(iris)
> # Perform ANOVA to test if Sepal.Length differs by Species
> anova_result <- aov(Sepal.Length ~ Species, data=iris)
> # Display the summary of the ANOVA
> summary(anova_result)
```

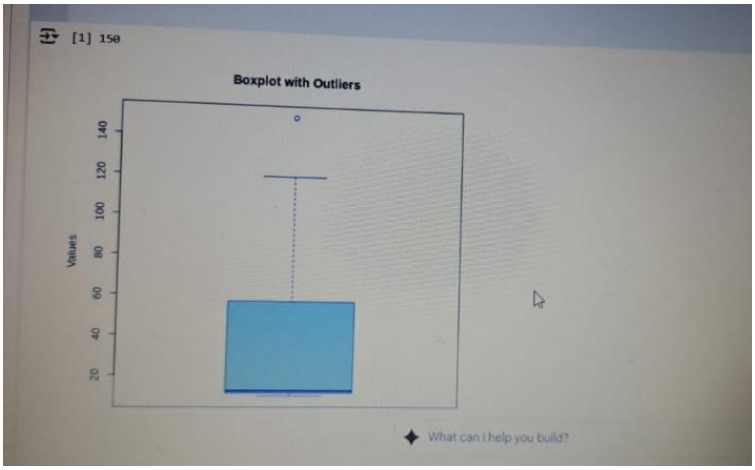
week 5)





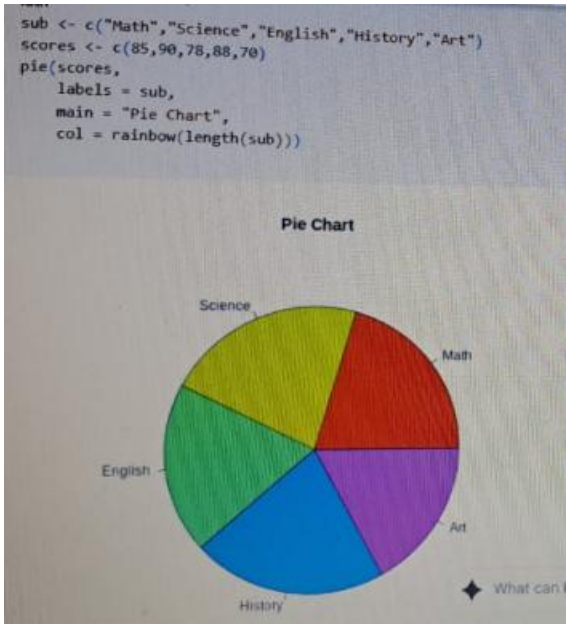
```r
# Sample data with clear outliers
data <- c(10, 12, 11, 13, 12, 14, 11, 13, 12, 100, 120, 150)

# Boxplot
boxplot(data,
        main = "Boxplot with Outliers",
        ylab = "Values",
        col = "skyblue",
        border = "darkblue")

# Mark outliers
outliers <- boxplot.stats(data)$out
print(outliers)
```

```
[1] 150
```



What can I help you build?

```r
marks<-c(60,75,80,90,55,70,85,95,50,65)
hist(marks,main="Histogram",xlab="marks",col="lightgreen",border="black")
```



week 2

descriptive statistics

```r
# Load dataset
data(iris)

# 1. Show first 6 rows
head(iris)

# 2. Show last 6 rows
tail(iris)

# 3. Structure of data (data types + preview)
str(iris)

# 4. Summary statistics (min, max, mean, median, quartiles)
summary(iris)

# 5. Mean of numeric columns
colMeans(iris[, 1:4])

# 6. Standard deviation of numeric columns
apply(iris[, 1:4], 2, sd)

# 7. Variance of numeric columns
apply(iris[, 1:4], 2, var)

# 8. Correlation matrix (numeric columns)
cor(iris[, 1:4])
```

Subset

```r
data(mtcars)

# Subset of cars with mpg > 20
subset1 <- subset(mtcars, mpg > 20)

# Subset selecting only mpg and hp columns
subset2 <- subset(mtcars, select = c(mpg, hp))

# Cars with mpg > 20 AND 4 cylinders, selecting few columns
subset3 <- subset(mtcars, mpg > 20 & cyl == 4,
                  select = c(mpg, cyl, hp))


print("Cars with mpg > 20:")
print(subset1)

print("Subset with only mpg and hp columns:")
print(head(subset2))

print("Cars with mpg > 20 and 4 cylinders:")
print(subset3)
```

week 1-2

a)

(a) Using with and without R objects on console

Code

r Copy code

```r
# Without using R objects
result_without_objects <- (10 + 5) * 2
print(result_without_objects)

# Using R objects (variables)
x <- 10
y <- 5
result_with_objects <- (x + y) * 2
print(result_with_objects)
```

b)

Aim: Calculator Application:
b)Using mathematical functions on console.

Program:

```
num1<-as.numeric(readline("Enter first number: "))
num2<-as.numeric(readline("Enter Second number: "))
abs(num1)
sqrt(num2)
round(num1,3)
ceiling(num1)
floor(num1)
sin(num2)
cos(num2)
tan(num2)
log(num2)
exp(num2)
factorial(num2)
choose(5,2)
sign(num1)
```

c)

```
op <- readline("Enter operator (+, -, *, /): ")
num1 <- as.numeric(readline("Enter first number: "))
num2 <- as.numeric(readline("Enter second number: "))
# Perform operation
if (op == "+") {
  result <- num1 + num2
} else if (op == "-") {
  result <- num1 - num2
} else if (op == "*") {
  result <- num1 * num2
} else if (op == "/") {
  result <- num1 / num2
} else {
  result <- NA
  cat("Invalid Operator\n")
}
# Create R object
calculator_data <- list(
  number1 = num1,
  number2 = num2,
  operator = op,
  result = result
)
# File path (IMPORTANT: use double \\ in Windows)
save_path <- "C:\\Users\\admin\\Desktop\\data.RData"
# Save to disk
save(calculator_data, file = save_path)
```

d)

ggplot2

✅ 2 ggplot2 – Bar Chart

✅ Simple Code

```r
library(ggplot2)

df <- data.frame(
  Subject = c("Math", "Science", "English"),
  Score = c(85, 90, 78)
)

ggplot(df, aes(x = Subject, y = Score)) +
  geom_bar(stat = "identity")
```

🟦 Simple Explanation

- We create a small table with subjects and scores.
- `ggplot` draws the chart.
- `geom_bar(stat="identity")` means
  use the numbers as they are to draw bar height.

ggraph

✅ 3 ggraph – Simple Graph Plot

✅ Simple Code

```r
library(ggraph)
library(igraph)

g <- make_ring(4)

ggraph(g) +
  geom_edge_link() +
  geom_node_point()
```

🟦 Simple Explanation

- `make_ring(4)` creates **4 nodes connected in a circle**.
- `geom_node_point()` draws the points (nodes).
- `geom_edge_link()` draws the lines (edges).
- Final output: a simple ring-shaped network.

glue

✅ 4️⃣ glue – Insert variables into text

✅ Simple Code

```r
library(glue)

name <- "Alice"
score <- 92

glue("Hello {name}, your score is {score}")
```

🟦 Simple Explanation

- Glue helps you **insert variable values inside text**.
- `{name}` → changes to "Alice".
- `{score}` → becomes "92".
- Makes printing messages very easy.

shiny

1️⃣ Shiny – Hello World

```r
library(shiny)

ui <- fluidPage(
  textOutput("greeting")
)

server <- function(input, output) {
  output$greeting <- renderText({
    "Hello, World!"
  })
}

shinyApp(ui, server)
```

Explanation

- `ui` → user interface, just displays text.

tidyr

✅ Simple Code

```r
library(tidyr)

df <- data.frame(
  Name = c("Alice", "Bob"),
  Math = c(90, 80),
  Science = c(85, 95)
)

pivot_longer(df, cols = c(Math, Science),
             names_to = "Subject",
             values_to = "Marks")
```

🟦 Simple Explanation

- `df` is a normal table: one row per student.
- Math and Science are **separate columns** → wide format.
- `pivot_longer` converts it into **long format**, meaning:
  - Each **student + subject + marks** becomes o row.
- Very useful for plotting and analysis.

Week 4


11:25
89
########## 3. XML FILES
############
# Install package if needed
# install.packages("XML")
library(XML)
# Read XML
xml_file <- xmlParse("input.xml")
# Convert XML to data frame
xml_data <- xmlToDataFrame(xml_file)
print(xml_data)
# Write XML
saveXML(xml_file, file = "output.xml")
############### 4. TEXT FILES (TXT)
###############
# Read TXT (table format)
txt_data <- read.table("input.txt", header
= TRUE)
print(txt_data)
# Read TXT (line by line)
txt_lines <- readLines("input.txt")
print(txt_lines)
# Write TXT (table)
write.table(txt_data, "output.txt",
row.names = FALSE)
# Write simple TXT
writeLines(c("Hello", "This is a text
file"), "simple_output.txt")
Ask ChatGPT


11:25
89
# Read CSV
csv_data <- read.csv("input.csv")
print(csv_data)
# Write CSV
write.csv(csv_data, "output.csv", row.names
= FALSE)
############### 2. EXCEL (XLSX) FILES
###############
# Install packages if needed
# install.packages("readxl")
# install.packages("writexl")
library(readxl)
library(writexl)
# Read XLSX
excel_data <- read_excel("input.xlsx")
print(excel_data)
# Write XLSX
write_xlsx(excel_data, "output.xlsx")
############### 3. XML FILES
###############
# Install package if needed
# install.packages("XML")
library(XML)
# Read XML
Ask ChatGPT