1. Write a R program for different types of datastructures in R.

Vector

```
# Creating a character vector
character_vector <- c ("apple", "banana", "cherry")
character_vector
```

output: [1] "apple" "banana" "cherry"

Matrix

```
# Creating a numeric matrix
numeric_matrix <- matrix (1:6, nrow = 2, ncol = 3)
numeric_matrix
```

```
output:      [,1] [,2] [,3]
        [1,]    1    3    5
        [2,]    2    4    6
```

Lists

```
# Creating a list
my_list <- list (name = c("John","Daniel","Jack"), age = c (30,53,40), hobbies =c ("reading",
"golf","Gaming"))

my_list
```

```
output: $name:[1] "John"   "Daniel" "Jack"
        $age:[1] 30 53 40
        $hobbies:[1] "reading" "golf"    "Gaming"
```

DataFrame

```
# Creating a data frame
data_frame <- data.frame (Name = c ("Alice", "Bennett", "Charlie"),Age = c (25, 30, 22),
Gender = c ("Female", "Male", "Male"))

data_frame
```

```
output:   Name    Age  Gender
        1 Alice    25  Female
        2 Bennett  30  Male
        3 Charlie  22  Male
```

Factors

```
# Creating a factor
gender <- c ("Male","Female","Male","Female","Male")
factor_gender <- factor (gender, levels = c ("Male","Female"))

factor_gender
```

output:
```
[1] Male Female Male Female Male
Levels: Male Female
```

Array

```
#Creating an Array
arr <- array (1:24, dim = c (4,3,2))
arr
```

output:
```
, , 1
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2
     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

2.Write a R program that include variables, constants, data types.

```
# Define variables

radius <-5
radius
```

output:[1] 5

```
name <- "Alice"
name
```

output:[1] "Alice"

```
age <- 30L
age
```


output:[1] 30

```
is_student <- TRUE
is_student
```

output: [1] TRUE

```
# Constants
PI <- 3.14159265359
paste ("Constant Value:",PI)
```

output:[1] "Constant Value: 3.14159265359"

```
GREETING <- "Hello, World!"
paste ("Constant Value:", GREETNG)
```

output:[1] "Constant Value: Hello, World!"

```
# Data types
```

print(class(radius))	output: [1] "numeric"
print(class(name))	output: [1] "character"
print(class(age))	output: [1] "integer"
print(class(is_student))	output: [1] "logical"

3.Write a R program that include different operators, control structures, default values for arguments, returning complex objects

```
# Arithmetic operators
a<-11
b<-4

sum_result <-  a + b
sum_result

output:[1] 15

diff_result <-  a - b
diff_result

output:[1] 7

product_result <-  a * b
product_result

output:[1] 44

division_result <- a / b
division_result

output:[1] 2.75

modulus_result<-a%%b
modulus_result

output:[1] 3

# Control structure (if-else)

if (a > b) {
        print ("a is greater than b")
} else if (a < b) {
        print <- "a is less than b"
} else {
        print <- "a is equal to b"
}
output:[1] "a is greater than b"
```

 # Default values for arguments

```
my_function <- function (country = "INDIA") {
  paste("I am from", country)
}
my_function("USA")
my_function () # will get the default value, which is INDIA
```

output: [1] "I am from USA"

[1] "I am from INDIA"

 # Returning complex objects

```
res<-function () {
        v<-c (1,2,5,3,8)
        m<-matrix (1:8, ncol=4)
        v1<-mean(v)
        m1<-min(m)
        L<-list (vec=v1, mat=m1)
        return(L)
}
res ()
```

output: $vec

[1] 3.8

$mat

[1] 1

5.Write a R program for calculating cumulative sums, and products minima maxima and calculus

```
# Sample vector of numbers
numbers <- c (1, 2, 3, 4, 5)

# Calculate cumulative sum
cumulative_sum <- cumsum(numbers)
cat ("Cumulative Sum:", cumulative_sum, "\n")
```

output: Cumulative Sum: 1 3 6 10 15

```
# Calculate cumulative product

cumulative_product <- cumprod(numbers)
cat("Cumulative Product:", cumulative_product, "\n")
```

output: Cumulative Product: 1 2 6 24 120

```
# Calculate minimum and maximum

min_value <- min(numbers)
max_value <- max(numbers)
cat ("Minimum:", min_value,"\n")
cat ("Maximum:", max_value,"\n")
```

output: Minimum:1   Maximum:5

```
library (Deriv) # Basic calculus operations

# Define a function, e.g., f(x) = x^2

f <- function(x) x^2

# Calculate the derivative of the function

derivative <- Deriv(f)
cat ("Derivative of f(x) = x^2:", derivative (2), "\n")  # Evaluate the derivativeat x = 2
```

output: Derivative of f(x) = x^2: 4

```
# Integrate the function from 1 to 5

integral <- integrate (f, lower = 1, upper = 5)
cat ("Integral of f(x) = x^2 from 1 to 5:", integral$value, "\n")
```

output: Integral of f(x) = x^2 from 1 to 5: 41.33333

6.Write a R program for finding stationary distribution of markanov chains

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

# Define the transition matrix for your Markov chain

transition_matrix <- matrix (c (0.8, 0.2,0.4, 0.6), nrow = 2, byrow =TRUE)

# Define the states

states <- c ("State A", "State B")

# Create a Markov chain object

my_markov_chain <- new ("markovchain", states = states, transitionMatrix =transition_matrix)

# Find the stationary distribution

stationary_dist <- steadyStates(my_markov_chain)

# Print the stationary distribution

cat ("Stationary Distribution:")
print(stationary_dist)
```

output: Stationary Distribution:

```
      State A      State B
[1,] 0.6666667    0.3333333
```

4.Write a R program for quick sort implementation, binary search tree Quick Sort

```
# Quick Sort

quick_sort <- function(arr) {
        if (length(arr) <= 1) {
                return (arr)
        }
        pivot <- arr[length(arr) %/% 2]
        left <- arr [arr < pivot]
        middle <- arr [arr == pivot]
        right <- arr [arr > pivot]
        return(c(quick_sort(left), middle, quick_sort(right)))
}

vect = c (2,5,3,6,8,4,1,3,10)
print ("Unsorted Vector")
print(vect)

output:  [1]  2  5  3  6  8  4  1  3 10

sorted_vector <- quick_sort(vect)
print("sorted vector")
print(sorted_vector)

output:  [1]  1  2  3  3  4  5  6  8 10
```

```
# Define the structure for a Binary Search Tree node

bst_node <- function(key) {
                                        return (list (key = key, left = NULL,right = NULL))
}

# Function to insert a key into the BST

insert <- function (root, key) {
                    if (is.null(root)) {
                                                  return(bst_node(key))
                    }
                    if (key < root$key) {
                                                  root$left <- insert(root$left, key)
                    } else if (key > root$key) {
                                                  root$right <- insert(root$right, key)
                    }
        return(root)
}

# Function to perform an in-order traversal of the BST

in_order_traversal <- function(root) {
                                                  if (!is.null(root)) {
                                                                              in_order_traversal(root$left)
                                                                              cat(root$key, " ")
                                                                              in_order_traversal(root$right)
                                                                       }
}

# Example usage:

bst <- NULL
keys <- c (5, 3, 8, 1, 9, 2)

for (key in keys) {
                            bst <- insert (bst, key)
}

cat ("In-order traversal of BST:", "\n")
in_order_traversal(bst)

output:1 2 3 5 8 9
```

7.Write a R program that include linear algebra operations on vectors & matrices

```
# Create two square matrices

matrix_A <- matrix (1:4, nrow = 2)
matrix_B <- matrix (5:8, nrow = 2)

# Matrix determinant (for square matrices)

determinant_A <- det(matrix_A)
cat ("Determinant of Matrix A:", determinant_A, "\n")
```

output: Determinant of Matrix A: -2

```
# Matrix inverse (for square matrices)

inverse_A <- solve(matrix_A)
cat("Inverse of Matrix A:\n")
print(inverse_A)
```

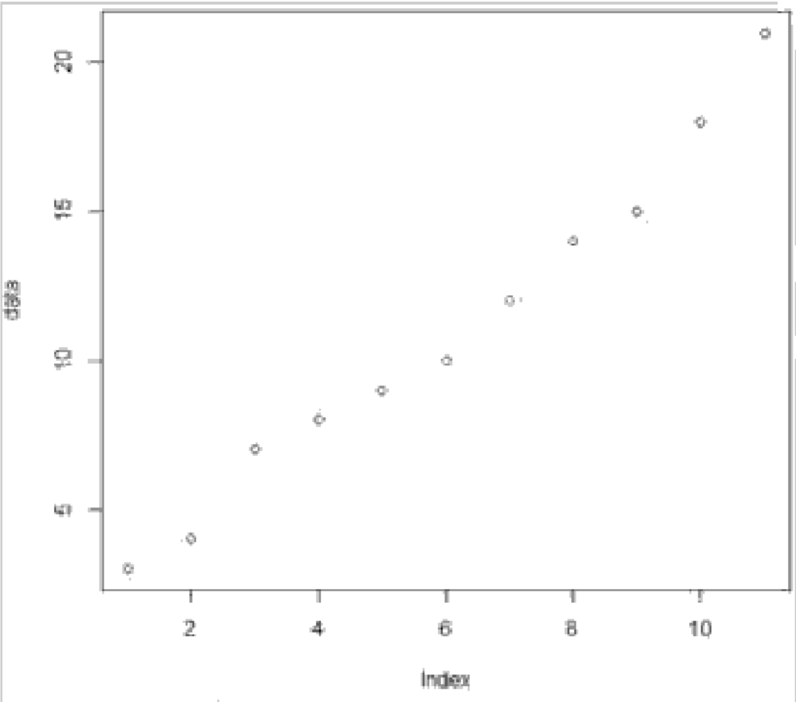
output: Inverse of Matrix A:

```
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

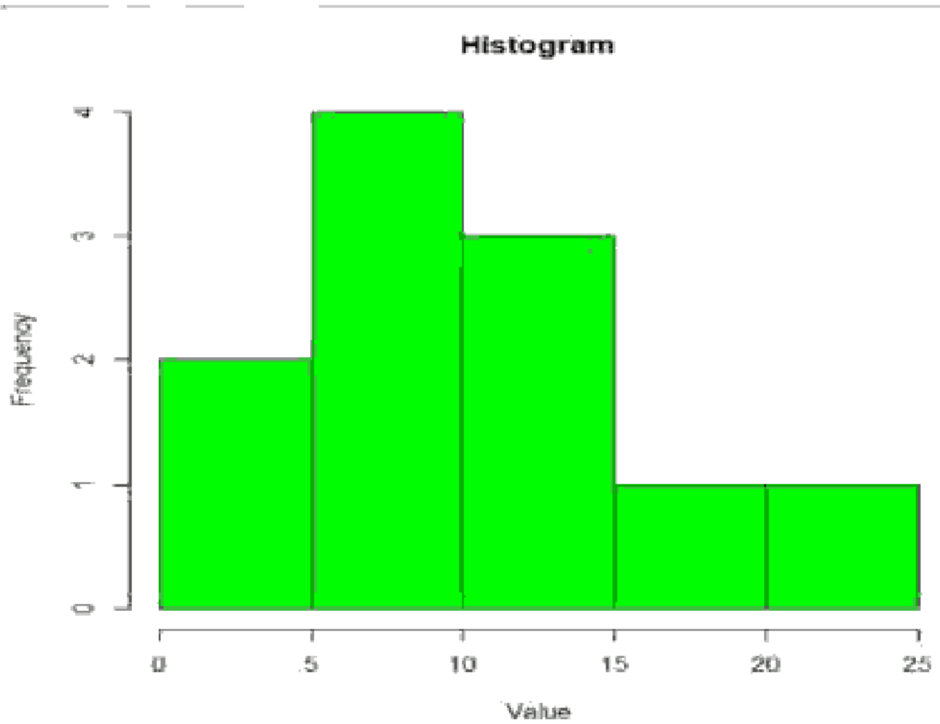
```r
# Simple Linear Algebra Operations in R

# Create two vectors
vector_a <- c(2, 4, 6)
vector_b <- c(1, 3, 5)

# Vector addition
vector_sum <- vector_a + vector_b
cat("Vector Sum:\n")
print(vector_sum)

# Create two matrices
matrix_A <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
matrix_B <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2)

# Matrix multiplication
matrix_product <- matrix_A %*% matrix_B
cat("Matrix Product:\n")
print(matrix_product)

# Solve a linear equation Ax = b
A <- matrix(c(2, 1, 1, 3), nrow = 2)
b <- c(8, 7)

# Solve for x
x <- solve(A, b)
cat("Solution of the linear equation Ax = b:\n")
print(x)
```

8.Write a R program for any visual representation of an object with creating graphs using graphic functions: Plot (), Hist (), Linechart (), Pie (), Boxplot (), Scatterplots()

```
# Create a sample data set
data <- c(3, 4, 7, 8, 9, 10, 12, 14, 15, 18, 21)
# Plot
plot(data)
```

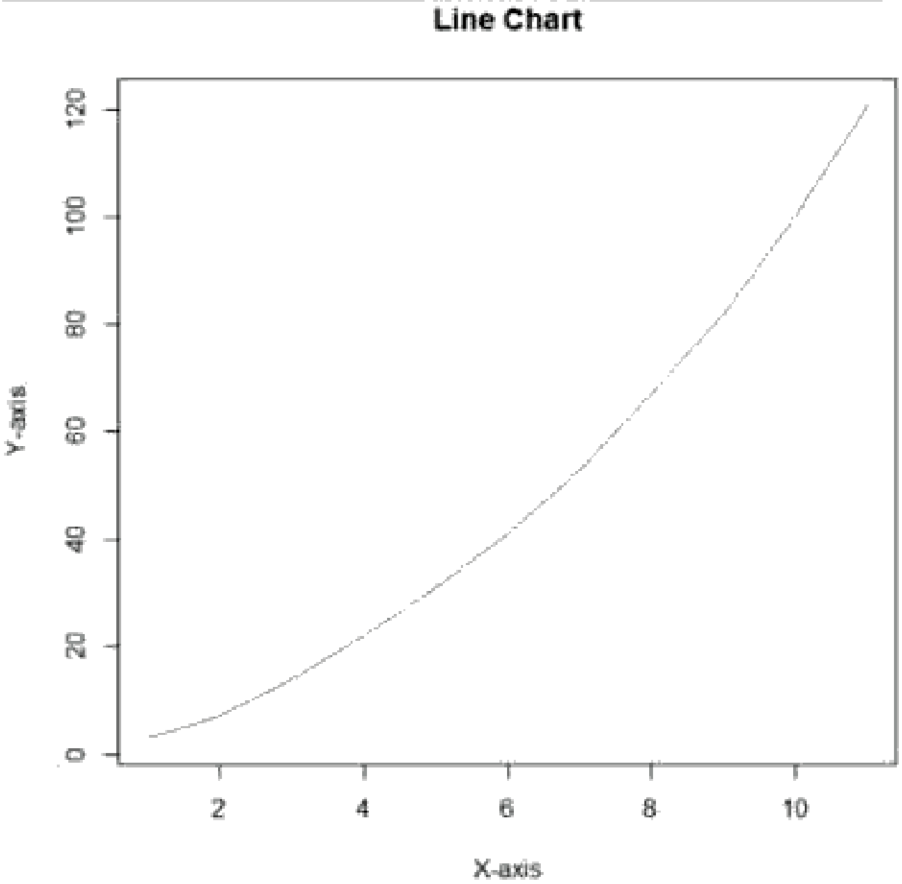



```
# Create a histogram
hist(data, breaks = 5, main = "Histogram", xlab = "Value", ylab = "Frequency", col = "green")
```



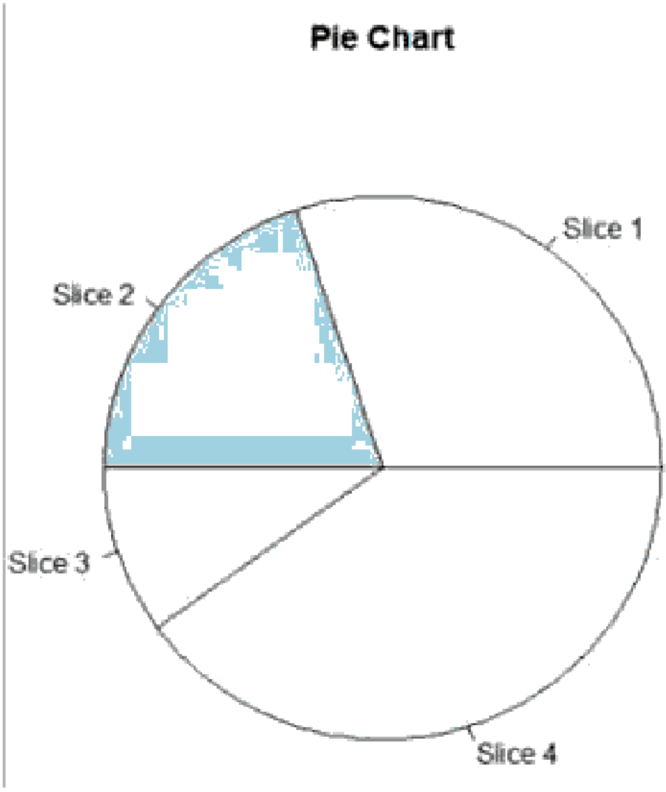
Create a line chart

```
x <- 1:length(data)
line_data <- cumsum(data)
plot(x, line_data, type = "l", col = "red", main = "Line Chart", xlab = "X-axis", ylab = "Y-axis")
```



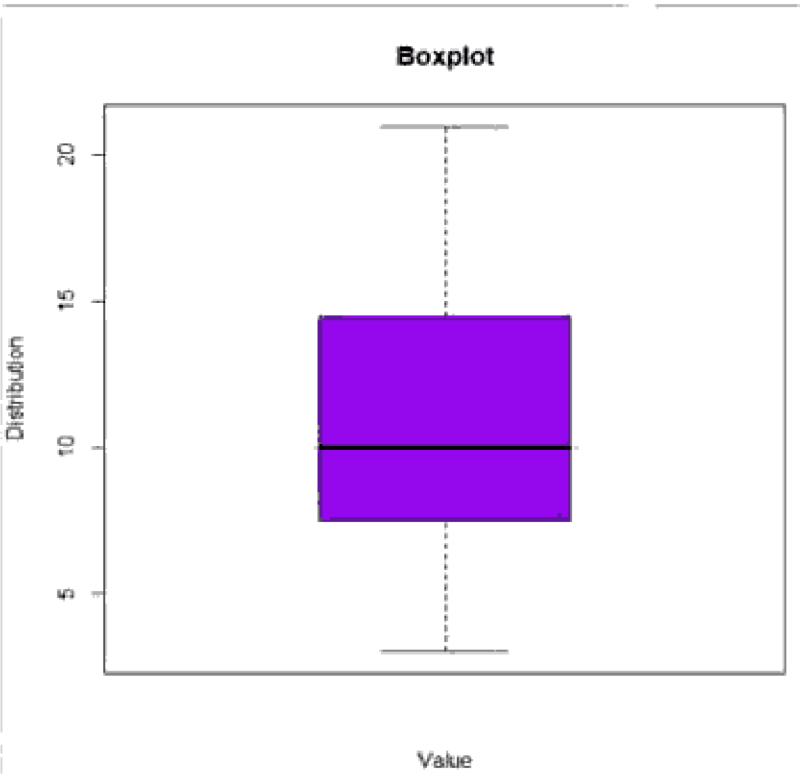
Create a pie chart

```
slices <- c (30, 20, 10, 40)
lbls <- c ("Slice 1", "Slice 2", "Slice 3", "Slice 4")
pie (slices, labels = lbls, main = "Pie Chart")
```



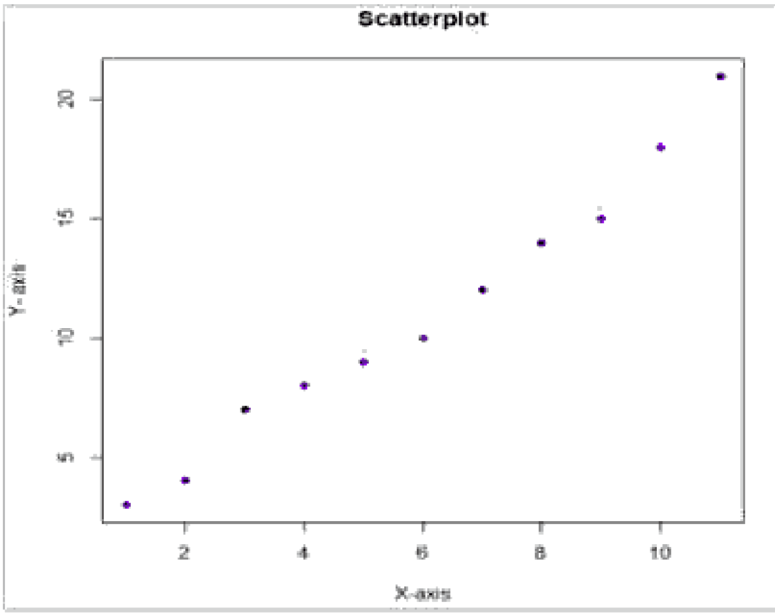
 # Create a boxplot

boxplot (data, main = "Boxplot", xlab = "Value", ylab = "Distribution", col = "purple")



 # Create a scatterplot

x2 <- c(2, 4, 6, 8, 10)
y2 <- c(5, 7, 3, 9, 2)
plot(x2, y2, type = "p", pch = 20, col = "orange", main = "Scatterplot", xlab = "X-axis", ylab ="Y-axis")



9.Write a R program for with any dataset containing data frame objects, indexing and sub setting data frames, and employ manipulating and analyzing data

```
# Create a sample data frame

data_frame <- data.frame(Name = c ("Alice", "Bennett", "Charlie", "David", "Emma"),
                         Age = c (25, 30, 22, 28, 35),
                         Gender = c ("Female", "Male", "Male", "Male", "Female"),
                         Score = c (85, 92, 78, 88, 95))

# Indexing and Subsetting

cat("\nSubset of Data Frame (Age > 25):\n")
subset_data <- data_frame[data_frame$Age > 25, ]
print(subset_data)
```

output:
```
  Name    Age Gender  Score
2 Bennett 30  Male    92
4 David   28  Male    88
5 Emma    35  Female  95
```

```
# Calculate Summary Statistics

summary_stats <- summary(data_frame$Score)
summary_stats
```

output:
```
Min.  1st Qu.  Median  Mean  3rd Qu.  Max.
78.0  85.0     88.0    87.6  92.0     95.0
```

```
# Add a new column

data_frame$Grade <- ifelse(data_frame$Score >= 90, "A", ifelse(data_frame$Score >= 80,"B", "C"))
print(data_frame)
```

output:
```
  Name     Age  Gender  Score  Grade
1 Alice    25   Female  85     B
2 Bennete  30   Male    92     A
3 Charlie  22   Male    78     C
4 David    28   Male    88     B
5 Emma     35   Female  95     A
```

 # Grouping and Aggregation

```
gender_avg_score <- aggregate (data_frame$Score, by = list(data_frame$Gender), FUN =mean)
colnames(gender_avg_score) <- c("Gender", "Avg_Score")
print(gender_avg_score)
```

output:

```
  Gender Avg_Score
1 Female    90
2 Male      86
```

10.Write a program to create an any application of Linear Regression in multivariate context for predictive purpose.

 # Load the mtcars dataset

data(mtcars)

 # Explore the dataset

head(mtcars)

output:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

 # Fit a multivariate linear regression model

 # We'll predict 'mpg' (miles per gallon) based on 'hp' (horsepower) and 'wt' (weight)

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

model

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

Coefficients:
(Intercept)           hp           wt
   37.22727     -0.03177     -3.87783
```

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


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

Coefficients:
(Intercept)           hp           wt
   37.22727     -0.03177     -3.87783
```

 # Print the model summary
summary(model)

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

Residuals:
   Min     1Q Median     3Q    Max
-3.941 -1.600 -0.182  1.050  5.854

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
hp          -0.03177    0.00903  -3.519  0.00145 **
wt          -3.87783    0.63273  -6.129 1.12e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.593 on 29 degrees of freedom
Multiple R-squared:  0.8268,    Adjusted R-squared:  0.8148
F-statistic: 69.21 on 2 and 29 DF,  p-value: 9.109e-12
```

 # Make predictions using the model

```
new_data <- data.frame(hp = c(150, 200), wt = c(3.5, 4.0))
predictions <- predict(model, newdata = new_data)
cat("Predicted MPG for new data:\n")
print(predictions)
```

output: Predicted MPG for new data

1	2
18.88892	15.36136