1)
```
#input username and age
name=readline(prompt="Enter the name :")
age=readline(prompt="Enter the age :")
print(paste("Iam",name,"and am",age,"years of old"))
print(R.version.string)
```

2)
```
#object in memory
name = "Python";
n1 =  10;
n2 =  0.5
nums = c(10, 20, 30, 40, 50, 60)
print(ls())
print("Details of the objects in memory:")
print(ls.str())
```

3)
```
20-50, mean(20-60), sonum(51-91)
sequence<-20:50
mean_20_to_60<-mean(20:60)
sum_51_to_91<-sum(51:91)
cat("Sequence of numbers from 20 to 50 :",sequence,"\n")
cat("Mean of numbers from 20 to 60 :",mean_20_to_60,"\n")
cat("Sum of numbers from 51 to 91 :",sum_51_to_91,"\n")
```

4)
```
#10 random int(-50,50)
random_vector<-sample(-50:50,10,replace = TRUE)
cat("Random vector:",random_vector,"\n")
```

5)
```
#first 10 fibonacci series
fibonacci<-function(n){
  if(n<=0){
    return(NULL)
  }else if(n==1){
    return(0)
  }else if(n==2){
    return(1)
  }else{
    fib<-numeric(n)
    fib[1]<-0
    fib[2]<-1
    for (i in 3:n) {
      fib[i]<-fib[i-1]+fib[i-2]
    }
    return(fib)
  }
}
first_10_fib<-fibonacci(10)
cat("First 10 Fibonacci numbers:",first_10_fib,"\n")
```

6)
```
#all primes
get_primes_up_to_n <- function(n) {
  if (n <= 1) {
    return(NULL)
  }
  sieve <- rep(TRUE, n + 1)
  sieve[1] <- FALSE
  for (i in 2:sqrt(n)) {
    if (sieve[i]) {
      sieve[i * i:n] <- FALSE
    }
  }
  primes <- which(sieve)

  return(primes)
}
given_number <- 50
prime_numbers <- get_primes_up_to_n(given_number)
print(prime_numbers)
```

7)

```
#fizz,buzz
for (num in 1:100) {
  if (num %% 3 == 0 && num %% 5 == 0) {
    cat("FizzBuzz", "\n")
  } else if (num %% 3 == 0) {
    cat("Fizz", "\n")
  } else if (num %% 5 == 0) {
    cat("Buzz", "\n")
  } else {
    cat(num,"\n")
  }
}
```

8)

```
#f10 up case,l10 lower case
first_10_lower <- letters[1:10]
last_10_upper <- toupper(letters[17:26])
letters_between_22_24 <- toupper(letters[22:24])
cat("First 10 English letters in lower case:", first_10_lower, "\n")
cat("Last 10 English letters in upper case:", last_10_upper, "\n")
cat("Letters between 22nd and 24th in upper case:",
letters_between_22_24,"\n")
```

9)

```
#factors
find_factors <- function(number) {
  factors <- c()
  for (i in 1:number) {
    if (number %% i == 0) {
      factors <- c(factors, i)
    }
  }
  return(factors)
}

# Replace 'your_number' with the number for which you want to find the
factors
your_number <- 24
factors_of_your_number <- find_factors(your_number)
print(paste("Factors of", your_number, "are:", factors_of_your_number))
```

```
10)
#max and min

11)
#unique elements and numbers of a given string,vector
# For unique elements of a string
get_unique_elements <- function(input_string) {
  unique_chars <- unique(strsplit(input_string, '')[[1]])
  return(unique_chars)
}

# For unique numbers of a vector
get_unique_numbers <- function(input_vector) {
  unique_numbers <- unique(input_vector)
  return(unique_numbers)
}

# Example usage
input_string <- "hello world"
unique_elements <- get_unique_elements(input_string)
cat("Unique elements of the string:", unique_elements, "\n")

input_vector <- c(12, 45, 6, 23, 9, 15, 30, 7, 42, 12, 15)
unique_numbers <- get_unique_numbers(input_vector)
cat("Unique numbers of the vector:", unique_numbers, "\n")

12)
#Write a R program to create three vectors a,b,c with 3 integers. Combine
the three vectors to
#become a 3Ã—3 matrix where each column represents a vector. Print the
content of the matrix.
# Create three vectors
a <- c(1, 2, 3)
b <- c(4, 5, 6)
c <- c(7, 8, 9)

# Combine vectors into a matrix
combined_matrix <- matrix(c(a, b, c), nrow = 3, byrow = TRUE)

# Print the matrix
print(combined_matrix)

13)
#a list of random numbers in normal distribution and count occurrences of
each value.
# Set the seed for reproducibility (optional)
set.seed(123)

# Generate a list of random numbers following a normal distribution
num_samples <- 100
mean_value <- 0
sd_value <- 1
random_numbers <- rnorm(num_samples, mean = mean_value, sd = sd_value)

# Count occurrences of each value
occurrences <- table(random_numbers)

# Print the occurrences
```

```
print(occurrences)

14)
#read the .csv file and display the content
# Replace 'your_file.csv' with the actual path to your CSV file
file_path <- "your_file.csv"

# Read the CSV file
data <- read.csv(file_path)

# Display the content of the CSV file
print(data)

15)
#three vectors numeric data, character data and logical data. Display the
content of the vectors and their type
# Create a numeric vector
numeric_vector <- c(1.5, 2.7, 3.0, 4.2, 5.8)

# Create a character vector
character_vector <- c("apple", "banana", "orange", "grape", "pear")

# Create a logical vector
logical_vector <- c(TRUE, FALSE, TRUE, FALSE, TRUE)

# Display content and type of the vectors
cat("Numeric vector:", numeric_vector, "\n")
cat("Type:", typeof(numeric_vector), "\n\n")

cat("Character vector:", character_vector, "\n")
cat("Type:", typeof(character_vector), "\n\n")

cat("Logical vector:", logical_vector, "\n")
cat("Type:", typeof(logical_vector), "\n")

16)
#o create a 5 x 4 matrix , 3 x 3 matrix with labels and fill the matrix
by rows and 2 Ã— 2 matrix with labels and fill the matrix by columns
# Create a 5 x 4 matrix with labels and fill by rows
matrix_5x4 <- matrix(1:20, nrow = 5, ncol = 4, byrow = TRUE)
rownames(matrix_5x4) <- c("Row 1", "Row 2", "Row 3", "Row 4", "Row 5")
colnames(matrix_5x4) <- c("Col 1", "Col 2", "Col 3", "Col 4")

# Create a 3 x 3 matrix with labels and fill by rows
matrix_3x3 <- matrix(21:29, nrow = 3, ncol = 3, byrow = TRUE)
rownames(matrix_3x3) <- c("R1", "R2", "R3")
colnames(matrix_3x3) <- c("C1", "C2", "C3")

# Create a 2 x 2 matrix with labels and fill by columns
matrix_2x2 <- matrix(c(30, 32, 31, 33), nrow = 2, ncol = 2, byrow =
FALSE)
rownames(matrix_2x2) <- c("Row 1", "Row 2")
colnames(matrix_2x2) <- c("Col A", "Col B")

# Display the matrices
cat("5 x 4 Matrix:\n")
print(matrix_5x4)
cat("\n")
cat("3 x 3 Matrix:\n")
print(matrix_3x3)
cat("\n")

cat("2 x 2 Matrix:\n")
print(matrix_2x2)
```

17)
#o create an array, passing in a vector of values and a vector of dimensions. Also provide names for each dimension

```
# Create a vector of values
values <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)

# Create a vector of dimensions
dimensions <- c(3, 2, 2)  # 3 rows, 2 columns, 2 depth levels

# Provide names for each dimension
dim_names <- list(c("Row1", "Row2", "Row3"), c("Col1", "Col2"),
c("Depth1", "Depth2"))

# Create the array
my_array <- array(values, dim = dimensions, dimnames = dim_names)

# Print the array
print(my_array)
```

18)
#to create an array with three columns, three rows, and two "tables", taking two vectors as input to the array. Print the array.

```
# Create two input vectors
vector1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vector2 <- c(10, 11, 12, 13, 14, 15, 16, 17, 18)

# Combine the vectors into a single matrix
combined_matrix <- rbind(vector1, vector2)

# Reshape the combined matrix into a 3x3x2 array
my_array <- array(combined_matrix, dim = c(3, 3, 2))

# Print the array
print(my_array)
```

19)
#m to create a list of elements using vectors, matrices and a functions. Print the content of the list.

```
# Create a vector
vector_element <- c(1, 2, 3, 4, 5)

# Create a matrix
matrix_element <- matrix(1:9, nrow = 3, ncol = 3)

# Create a function
custom_function <- function(x) {
  return(x^2)
}

# Create the list with the elements
```

```
my_list <- list(
  vector_element,
  matrix_element,
  custom_function
)

# Assign names to each element in the list
names(my_list) <- c("vector", "matrix", "function")

# Print the contents of the list
print(my_list)

20)
#R program to draw an empty plot and an empty plot specify the axes
limits of the graphic
plot(NULL, xlim = c(0, 10), ylim = c(0, 20), xlab = "X Axis", ylab = "Y
Axis", main = "Empty Plot with Axes Limits")

21)
# array of two 3x3 matrices each with 3 rows and 3 columns from two given
two vectors. Print the second row of the second matrix of the array and
the element in the 3rd row and 3rd column of the 1st matrix
# Create the two vectors
vector1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vector2 <- c(10, 11, 12, 13, 14, 15, 16, 17, 18)

# Reshape the vectors into 3x3 matrices
matrix1 <- matrix(vector1, nrow = 3)
matrix2 <- matrix(vector2, nrow = 3)

# Create an array of the two matrices
matrix_array <- array(c(matrix1, matrix2), dim = c(3, 3, 2))

# Print the second row of the second matrix
print("Second row of the second matrix:")
print(matrix_array[2, , 2])

# Print the element in the 3rd row and 3rd column of the 1st matrix
print("Element in the 3rd row and 3rd column of the 1st matrix:")
print(matrix_array[3, 3, 1])

22)
#Write a R program to combine three arrays so that the first row of the
first array is followed by the first row of the second array and then
first row of the third array
# Create three example arrays (replace these with your own arrays)
array1 <- matrix(1:9, nrow = 3)
array2 <- matrix(10:18, nrow = 3)
array3 <- matrix(19:27, nrow = 3)

# Combine the arrays row-wise
combined_array <- rbind(array1[1,], array2[1,], array3[1,])

# Print the combined array
print(combined_array)

23)
```

```
# Write a R program to create an array using four given columns, three
given rows, and two given tables and display the content of the array.
# Given columns, rows, and tables
columns <- 4
rows <- 3
tables <- 2

# Create example data for two tables (replace these with your own data)
table1_data <- matrix(1:(columns * rows), nrow = rows)
table2_data <- matrix((columns * rows + 1):(columns * rows * 2), nrow =
rows)

# Create an array using the given columns, rows, and tables
array <- array(c(table1_data, table2_data), dim = c(rows, columns,
tables))

# Display the content of the array
print(array)

24)
#R program to create a two-dimensional 5x3 array of sequence of even
integers greater than 50
# Initialize parameters
rows <- 5
columns <- 3
start_value <- 52  # First even integer greater than 50

# Create the 2-dimensional array
even_array <- matrix(seq(start_value, by = 2, length.out = rows *
columns), nrow = rows)

# Print the array
print(even_array)

25)
#a. Write a R program to extract 3rd and 5th rows with 1st and 3rd
columns from a given data frame
#b. Write a R program to add a new column named country in a given data
frameCountry<-
c("USA","USA","USA","USA","UK","USA","USA","India","USA","USA")
#c. Write a R program to add new row(s) to an existing data
framenew_exam_data = data.frame(name = c('Robert', 'Sophia'),score =
c(10.5, 9), attempts = c(1, 3),qualify = c('yes', 'no'))
#d. Write a R program to sort a given data frame by name and score
# Original data frame
exam_data <- data.frame(
  name = c('Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael',
'Matthew', 'Laura', 'Kevin', 'Jonas'),
  score = c(12.5, 9, 16.5, 12, 9, 20, 14.5, 13.5, 8, 19),
  attempts = c(1, 3, 2, 3, 2, 3, 1, 1, 2, 1),
  qualify = c('yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no',
'yes')
)

# b. Add a new column named "country"
Country <- c("USA", "USA", "USA", "USA", "UK", "USA", "USA", "India",
"USA", "USA")
exam_data$country <- Country
```

```
print("Data Frame with New 'country' Column:")
print(exam_data)

# c. Add new row(s) to the existing data frame
new_exam_data <- data.frame(
  name = c('Robert', 'Sophia'),
  score = c(10.5, 9),
  attempts = c(1, 3),
  qualify = c('yes', 'no'),
  country = c('Canada', 'France')  # Adding corresponding country values
)
exam_data <- rbind(exam_data, new_exam_data)
print("Data Frame with Added Row:")
print(exam_data)

# d. Sort the data frame by name and score
sorted_data <- exam_data[order(exam_data$name, exam_data$score), ]
print("Sorted Data Frame by Name and Score:")
print(sorted_data)

# Display the contents of the CSV file
read_data <- read.csv(csv_file_path)
print("Contents of the CSV file:")
print(read_data)
```

26)

```
#R program to call the (built-in) dataset airquality. Check whether it is
a data frame or not? Order the entire data frame by the first and second
column. remove the variables 'Solar.R' and
'Wind' and display the data frame.
# Load the built-in 'airquality' dataset
data(airquality)

# Check if the loaded data is a data frame
is_dataframe <- is.data.frame(airquality)
if (is_dataframe) {
  print("The loaded dataset 'airquality' is a data frame.")
} else {
  print("The loaded dataset 'airquality' is not a data frame.")
}

# Order the data frame by the first and second columns
ordered_airquality <- airquality[order(airquality$Month, airquality$Day),
]

# Remove 'Solar.R' and 'Wind' variables
trimmed_airquality <- ordered_airquality[, !(colnames(ordered_airquality)
%in% c('Solar.R', 'Wind'))]

# Display the trimmed data frame
print("Trimmed Data Frame:")
print(trimmed_airquality)
```

27)

```
#Write a R program to create a factor corresponding to height of women
data set , which inbuild in R, contains height and weights for a sample
of women.
# Load the built-in 'women' dataset
```

```
data(women)

# Create a factor corresponding to the height column
height_factor <- as.factor(women$height)

# Print the factor
print(height_factor)
```

28)

```
#Write a R program to create a factor corresponding to height of women
data set , which inbuild in R, contains height and weights for a sample
of women.
# Load the built-in 'women' dataset
data(women)

# Create a factor corresponding to the height column
height_factor <- factor(women$height)

# Print the factor
print(height_factor)
```

29)

```
#
# Load the built-in 'iris' dataset
data(iris)

# (i) Find dimension, structure, summary statistics, and standard
deviation of all features
print("Dimension:")
print(dim(iris))
cat("\n")

print("Structure:")
str(iris)
cat("\n")

print("Summary Statistics:")
summary(iris)
cat("\n")

print("Standard Deviation of All Features:")
sapply(iris[, 1:4], sd)
cat("\n")

# (ii) Find mean and standard deviation of features grouped by species
species_summary <- aggregate(iris[, 1:4], list(Species = iris$Species),
FUN = function(x) c(Mean = mean(x), SD = sd(x)))

print("Mean and Standard Deviation Grouped by Species:")
print(species_summary)
cat("\n")

# (iii) Find quantile value of sepal width and length
sepal_quantiles <- quantile(iris$Sepal.Width, probs = c(0.25, 0.5, 0.75))
print("Quantile Values of Sepal Width:")
print(sepal_quantiles)
cat("\n")
```

```
# (iv) Create new data frame 'iris1' with a new column
'Sepal.Length.Cate'
iris$Sepal.Length.Cate <- cut(iris$Sepal.Length, breaks =
quantile(iris$Sepal.Length))

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

print("Average Values by Species and Sepal.Length.Cate:")
print(avg_values)
cat("\n")

# (vi) Average mean value of numerical variables by Species and
Sepal.Length.Cate
avg_mean_values <- aggregate(avg_values[, 3:6], list(Species =
avg_values$Species), FUN = mean)

print("Average Mean Values by Species and Sepal.Length.Cate:")
print(avg_mean_values)
cat("\n")

# (vii) Create a Pivot Table based on Species and Sepal.Length.Cate
pivot_table <- reshape(avg_values, idvar = "Species", timevar =
"Sepal.Length.Cate", direction = "wide")

print("Pivot Table based on Species and Sepal.Length.Cate:")
print(pivot_table)

30)
#Randomly Sample the iris dataset such as 80% data for training and 20%
for test and create Logistics regression with train data, use species as
target and petals width andlength as feature variables , Predict the
probability of the model using test data, Create Confusion
#matix for above test model

# Load the built-in 'iris' dataset
data(iris)

# Set the random seed for reproducibility
set.seed(123)

# Sample the dataset (80% training, 20% test)
split <- sample(1:nrow(iris), size = 0.8 * nrow(iris))
train_data <- iris[split, ]
test_data <- iris[-split, ]

# Create a logistic regression model using train data
model <- glm(Species ~ Petal.Length + Petal.Width, data = train_data,
family = "binomial")

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

# Convert predicted probabilities to predicted classes
predicted_classes <- ifelse(test_probs > 0.5, "versicolor", "setosa")
```

```
# Create confusion matrix
conf_matrix <- table(predicted_classes, test_data$Species)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

31)

```
#(i)Write suitable R code to compute the mean, median ,mode of the
following valuesc(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)
#(ii) Write R code to find 2nd highest and 3rd
 Lowest value of above problem
# Given data
values <- c(90, 50, 70, 80, 70, 60, 20, 30, 80, 90, 20)

# (i) Compute mean, median, and mode
mean_value <- mean(values)
median_value <- median(values)

# Calculate mode (if it exists)
mode_value <- as.numeric(names(sort(table(values), decreasing =
TRUE))[1])

# Print results
print(paste("Mean:", mean_value))
print(paste("Median:", median_value))
if (!is.na(mode_value)) {
  print(paste("Mode:", mode_value))
} else {
  print("No unique mode found.")
}

# (ii) Find 2nd highest and 3rd lowest values
sorted_values <- sort(values)
second_highest <- sorted_values[length(sorted_values) - 1]
third_lowest <- sorted_values[3]

print(paste("2nd Highest Value:", second_highest))
print(paste("3rd Lowest Value:", third_lowest))
```

32)

```
#solar radiation
# Load the built-in 'airquality' dataset
data(airquality)

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

# ii. Extract the first five rows from airquality
first_five_rows <- airquality[1:5, ]
print("First Five Rows:")
print(first_five_rows)
cat("\n")
```

```
# iii. Extract all columns from airquality except Temp and Wind
columns_to_extract <- setdiff(names(airquality), c("Temp", "Wind"))
extracted_data <- airquality[, columns_to_extract]
print("Extracted Data with Columns Except Temp and Wind:")
print(extracted_data)
cat("\n")

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

33)
#

34)
#missing values
# Load the airquality dataset
data(airquality)

# (i) Find and handle missing values
missing_values <- colSums(is.na(airquality))
missing_percent <- (missing_values / nrow(airquality)) * 100

# Drop missing values if less than 10%, else replace with mean
for (col in colnames(airquality)) {
  if (missing_percent[col] < 10) {
    airquality[is.na(airquality[, col]), col] <- mean(airquality[, col],
na.rm = TRUE)
  } else {
    airquality <- airquality[complete.cases(airquality), ]
  }
}

# (ii) Apply linear regression on "Ozone" and "Solar.R"
model <- lm(Ozone ~ Solar.R, data = airquality)
summary(model)

# (iii) Plot Scatter plot between Ozone and Solar with regression line
plot(airquality$Solar.R, airquality$Ozone, main = "Scatter Plot of Ozone
vs. Solar.R",
     xlab = "Solar Radiation (Langley)", ylab = "Ozone (ppb)")
abline(model, col = "red")

35)
#

40)
#sales
# Create the data frame
data <- data.frame(
  Month = 1:12,
  Spends = c(1000, 4000, 5000, 4500, 3000, 4000, 9000, 11000, 15000,
12000, 7000, 3000),
  Sales = c(9914, 40487, 54324, 50044, 34719, 42551, 94871, 118914,
158484, 131348, 78504, 36284)
```

```
)

# b. Create a regression model
model <- lm(Sales ~ Spends, data = data)

# c. Predict the Sales if Spend=13500
new_data <- data.frame(Spends = 13500)
predicted_sales <- predict(model, newdata = new_data)

cat("Predicted Sales for Spend = 13500:", predicted_sales, "\n")
```

a)

```
# Load the airquality dataset

data(airquality)

# Get the summary statistics of the dataset

summary(airquality)
```

b)

```
# Load the reshape2 package

library(reshape2)

# Melt the airquality dataset and display it as a long-format data

airquality_melt <- melt(airquality, id.vars = c("Month", "Day"))

# Display the first few rows of the melted data

head(airquality_melt)
```

c)

```
# Melt the airquality dataset and specify month and day to be "ID

variables"

airquality_melt <- melt(airquality, id.vars = c("Month", "Day"),

measure.vars = c("Ozone", "Solar.R", "Wind", "Temp"))

# Display the first few rows of the melted data

head(airquality_melt)
```

d)

```
# Cast the molten airquality dataset with respect to month and

date features

airquality_cast <- dcast(airquality_melt, Month + Day ~ variable)

# Display the first few rows of the cast data
```

```
head(airquality_cast)
e)
# Use the cast function appropriately and compute the average of
Ozone, Solar.R, Wind, and temperature per month
airquality_mean <- dcast(airquality_melt, Month ~ variable,
fun.aggregate = mean)
# Display the average of each variable per month
airquality_mean
```

35)

```
# Load required libraries
library(dplyr)
library(reshape2)

# Load the dataset
data("ChickWeight")

# Define mode function
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

# (i) Order the data frame in ascending order by "weight" grouped
by "diet"
# Extract the last 6 records from ordered data frame
ordered_df <- ChickWeight %>%
  arrange(diet, weight) %>%
  group_by(diet) %>%
  slice_tail(n = 6)

# (ii) Perform melting function based on "Chick", "Time", "Diet"
features as ID variables
melted_df <- melt(ChickWeight, id.vars = c("Chick", "Time",
"Diet"))

# Perform cast function to display the mean value of weight
grouped by Diet
mean_df <- dcast(melted_df, Chick + Time + Diet ~ variable, mean)

# Perform cast function to display the mode of weight grouped by
Diet
mode_df <- dcast(melted_df, Chick + Time + Diet ~ variable,
getmode)
```

36)

```
# Load required library
library(ggplot2)

# a. Create Box plot for “weight” grouped by “Diet”
ggplot(ChickWeight, aes(x = factor(Diet), y = weight)) +
  geom_boxplot() +
  labs(x = "Diet", y = "Weight") +
  theme_minimal()

# b. Create a Histogram for “weight” features belong to Diet- 1
category
ggplot(subset(ChickWeight, Diet == 1), aes(x = weight)) +
  geom_histogram(binwidth = 10, fill = "blue", color = "black") +
  labs(x = "Weight", y = "Count") +
  theme_minimal()

# c. Create Scatter plot for ‟ weight” vs “Time” grouped by Diet
ggplot(ChickWeight, aes(x = Time, y = weight, color =
factor(Diet))) +
  geom_point() +
  labs(x = "Time", y = "Weight", color = "Diet") +
  theme_minimal()
```

37)

```
# Load required library
library(stats)

# a. Create multi regression model to find a weight of the
chicken, by “Time” and “Diet” as predictor variables
model <- lm(weight ~ Time + Diet, data = ChickWeight)

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

# c. Find the error in model for same
residuals <- resid(model)
```

```
# Load required library
library(ggplot2)

# Load the dataset
data("Titanic")

# Convert the dataset to a data frame
Titanic_df <- as.data.frame(Titanic)

# a. Draw a Bar chart to show details of “Survived” on the Titanic
based on passenger Class
ggplot(Titanic_df, aes(x = Class, fill = Survived)) +
  geom_bar(position = "dodge") +
  labs(x = "Class", y = "Count", fill = "Survived") +
  theme_minimal()

# b. Modify the above plot based on gender of people who survived
ggplot(Titanic_df, aes(x = Class, fill = interaction(Survived,
Sex))) +
  geom_bar(position = "dodge") +
  labs(x = "Class", y = "Count", fill = "Survived/Sex") +
  theme_minimal()

# c. Draw histogram plot to show distribution of feature “Age”
# Note: The Titanic dataset in R does not have an "Age" column.
# If you have a different dataset with an "Age" column, you can
use the following code:
# ggplot(your_data, aes(x = Age)) +
#   geom_histogram(binwidth = 5, fill = "blue", color = "black") +
#   labs(x = "Age", y = "Count") +
#   theme_minimal()
```

39)

```
# Load required library
library(stats)

# Load the dataset
data("USArrests")

# (i) a. Explore the summary of Data set
print(str(USArrests))
print(summary(USArrests))

# b. Print the state which saw the largest total number of rape
max_rape_state <- rownames(USArrests)[which.max(USArrests$Rape)]
print(max_rape_state)

# c. Print the states with the max & min crime rates for murder
max_murder_state <-
rownames(USArrests)[which.max(USArrests$Murder)]
min_murder_state <-
rownames(USArrests)[which.min(USArrests$Murder)]
print(c(max_murder_state, min_murder_state))

# (ii) a. Find the correlation among the features
correlation_matrix <- cor(USArrests)
print(correlation_matrix)

# b. Print the states which have assault arrests more than median
of the country
median_assault <- median(USArrests$Assault)
high_assault_states <- rownames(USArrests)[USArrests$Assault >
median_assault]
print(high_assault_states)
```

```
# c. Print the states are in the bottom 25% of murder
murder_quantile <- quantile(USArrests$Murder, 0.25)
low_murder_states <- rownames(USArrests)[USArrests$Murder <
murder_quantile]
print(low_murder_states)
```

10)

```
# Define the vector
vec <- c(1, 2, 3, 4, 5)

# Find the maximum value
max_val <- max(vec)

# Find the minimum value
min_val <- min(vec)

# Print the results
print(paste("Maximum Value: ", max_val))
print(paste("Minimum Value: ", min_val)
```