Slip 1

```
# Q1.Write a R program to add, multiply and divide two vectors of integertype.
(Vector
# length should be minimum 4)

# Define two integer vectors
vector1 <- c(1, 2, 3, 4)
vector2 <- c(5, 6, 7, 8)

# Addition
addition_result <- vector1 + vector2
print("Addition Result:")
print(addition_result)

# Multiplication
multiplication_result <- vector1 * vector2
print("Multiplication Result:")
print(multiplication_result)

# Division
division_result <- vector1 / vector2
print("Division Result:")
print(division_result)

# Q2.Consider the student data set. It can be downloaded from:
# https://drive.google.com/open?id=1oakZCv7g3mlmCSdv9J8kdSaqO 5_6dIOw .
# Write a programme in python to apply simple linear regression and find out mean
# absolute error, mean squared error and root mean squared error

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load the dataset
data = pd.read_csv("student_data.csv")

# Extract the features (X) and target (y)
X = data[['X']]  # Feature
y = data['Y']    # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate mean absolute error (MAE)
mae = mean_absolute_error(y_test, y_pred)

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculate root mean squared error (RMSE)
rmse = np.sqrt(mse)

# Print the results
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)

-----------------------------------------------------------------------------------
--------
slip 2

# Q1. Write an R program to calculate the multiplication table using afunction

# Function to generate a multiplication table
multiplication_table <- function(number, limit) {
  for (i in 1:limit) {
    result <- number * i
    cat(paste(number, "x", i, "=", result), "\n")
  }
}

# Input the number for which you want the multiplication table
number <- as.integer(readline("Enter a number for the multiplication table: "))

# Input the limit for the table
limit <- as.integer(readline("Enter the limit for the multiplication table: "))

# Call the function to generate and print the multiplication table
multiplication_table(number, limit)

# Q2. Write a python program to implement k-means algorithms on asynthetic
# dataset

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate a synthetic dataset with make_blobs
```

```
n_samples = 300
n_features = 2
n_clusters = 3

X, y = make_blobs(n_samples=n_samples, n_features=n_features, centers=n_clusters,
random_state=42)

# Create a K-Means clustering model
kmeans = KMeans(n_clusters=n_clusters)

# Fit the model to the data
kmeans.fit(X)

# Get cluster centers and labels
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_

# Plot the data points and cluster centers
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1,], marker='x', s=200,
linewidths=3, color='red')
plt.title("K-Means Clustering")
plt.show()

-----------------------------------------------------------------------------------
---------
slip 3

# Q1. Write a R program to reverse a number and also calculate the sum ofdigits of
that
# number.(i don't know how to run this r program)


# Function to reverse a number
reverse_number <- function(number) {
  reversed <- 0
  while (number > 0) {
    digit <- number %% 10
    reversed <- reversed * 10 + digit
    number <- number %/% 10
  }
  return(reversed)
}

# Function to calculate the sum of digits of a number
sum_of_digits <- function(number) {
  sum_digits <- 0
  while (number > 0) {
    digit <- number %% 10
    sum_digits <- sum_digits + digit
    number <- number %/% 10
  }
  return(sum_digits)
}

# Input the number
number <- as.integer(readline("Enter a number: "))

# Reverse the number
reversed <- reverse_number(number)

# Calculate the sum of digits
sum_digits <- sum_of_digits(number)

# Print the results
cat("Reversed Number:", reversed, "\n")
cat("Sum of Digits:", sum_digits, "\n")

# Q2. Consider the following observations/data. And apply simple linear regression
and find
# out estimated coefficients b0 and b1.( use numpypackage)
# x=[0,1,2,3,4,5,6,7,8,9,11,13]
# y = ([1, 3, 2, 5, 7, 8, 8, 9, 10, 12,16, 18]

import numpy as np

# Given data
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12, 16, 18])

# Calculate the means
x_mean = np.mean(x)
y_mean = np.mean(y)

# Calculate b1 (slope)
numerator = np.sum((x - x_mean) * (y - y_mean))
denominator = np.sum((x - x_mean) ** 2)
b1 = numerator / denominator

# Calculate b0 (intercept)
b0 = y_mean - b1 * x_mean

# Print the coefficients
print("b0 (intercept):", b0)
print("b1 (slope):", b1)
-----------------------------------------------------------------------------------
--------

slip 4
```

```
# Q1. Write a R program to calculate the sum of two matrices of given size

# Define the size of the matrices
rows <- 3
cols <- 3

# Create two example matrices
matrix1 <- matrix(1:9, nrow = rows, ncol = cols)
matrix2 <- matrix(9:1, nrow = rows, ncol = cols)

# Calculate the sum of the matrices
sum_matrix <- matrix1 + matrix2

# Print the original matrices and their sum
cat("Matrix 1:\n")
print(matrix1)

cat("Matrix 2:\n")
print(matrix2)

cat("Sum of the matrices:\n")
print(sum_matrix)

# Q2. Consider following dataset
#
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sun
ny','Rainy','Sunn
# y','Overcast','Overcast','Rainy']
#
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mi
ld','Hot','Mild']
#
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No
'].
# Use Naïve Bayes algorithm to predict [0: Overcast, 2: Mild]tuple belongs to which
class
# whether to play the sports or not.

from collections import Counter
from functools import reduce

# Given dataset
weather = ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast',
'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy']
temp = ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool',
'Mild', 'Mild', 'Mild', 'Hot', 'Mild']
play = ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes',
'Yes', 'Yes', 'No']

# Tuple to predict
```

```
new_weather = 'Overcast'
new_temp = 'Mild'

# Calculate prior probabilities
prior_yes = Counter(play)['Yes'] / len(play)
prior_no = Counter(play)['No'] / len(play)

# Calculate conditional probabilities for "weather"
def conditional_prob_weather(class_value):
    return Counter([weather[i] for i, val in enumerate(play) if val ==
class_value])[new_weather] / Counter(play)[class_value]

conditional_prob_weather_yes = conditional_prob_weather('Yes')
conditional_prob_weather_no = conditional_prob_weather('No')

# Calculate conditional probabilities for "temp"
def conditional_prob_temp(class_value):
    return Counter([temp[i] for i, val in enumerate(play) if val ==
class_value])[new_temp] / Counter(play)[class_value]

conditional_prob_temp_yes = conditional_prob_temp('Yes')
conditional_prob_temp_no = conditional_prob_temp('No')

# Calculate the posterior probabilities
posterior_yes = prior_yes * conditional_prob_weather_yes *
conditional_prob_temp_yes
posterior_no = prior_no * conditional_prob_weather_no * conditional_prob_temp_no

# Make the prediction
prediction = 'Yes' if posterior_yes > posterior_no else 'No'

# Print the results
print('Prior Probability (Yes):', prior_yes)
print('Prior Probability (No):', prior_no)
print('Posterior Probability (Yes):', posterior_yes)
print('Posterior Probability (No):', posterior_no)
print('Prediction:', prediction)
-------------------------------------------------------------------------------------
---------
slip 5

# Q1. Write a R program to concatenate two given factors.

# Define two factors
factor1 <- factor(c("A", "B", "C", "D"))
factor2 <- factor(c("E", "F", "G", "H"))

# Concatenate the two factors
concatenated_factor <- c(factor1, factor2)
```

```
# Print the concatenated factor
print(concatenated_factor)

# Q2. Write a Python program build Decision Tree Classifier using Scikit- learn
package for
# diabetes data set (download database from
https://www.kaggle.com/uciml/pima indians-diabetes-database)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
diabetes_data = pd.read_csv("diabetes.csv")  # Replace with the actual path to the
dataset

# Split the data into features (X) and the target variable (y)
X = diabetes_data.drop("Outcome", axis=1)
y = diabetes_data["Outcome"]

# Split the data into training and testing sets (e.g., 70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Create a Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("\nClassification Report:\n", report)

-----------------------------------------------------------------------------------
----

slip 6
# Q1. Write a R programto create a data frame using two given vectors and
displaythe duplicate
# elements.
# Create two vectors
vector1 <- c(1, 2, 3, 4, 5, 6, 2, 8, 9, 10)
vector2 <- c("A", "B", "C", "D", "E", "F", "A", "H", "I", "J")

# Create a data frame
my_data_frame <- data.frame(Vector1 = vector1, Vector2 = vector2)

# Display the duplicate elements
duplicates <- my_data_frame[duplicated(my_data_frame) | duplicated(my_data_frame,
fromLast = TRUE), ]
print(duplicates)

# Q2. Write a python program to implement hierarchical Agglomerative
clusteringalgorithm.
# (Download Customer.csv dataset from github.com).


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Load the customer data from the CSV file
data = pd.read_csv("customer.csv")

# Select the features for clustering (e.g., "Age" and "Income")
X = data[["Age", "Income"]]

# Perform hierarchical agglomerative clustering
n_clusters = 2  # Number of clusters to create
linkage_type = 'ward'  # Linkage method

model = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage_type)
labels = model.fit_predict(X)

# Plot the dendrogram (optional)
linkage_matrix = linkage(X, method=linkage_type)
dendrogram(linkage_matrix, orientation="top")
plt.title("Dendrogram")
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()

# Add cluster labels to the original dataset
data["Cluster"] = labels

# Print the resulting clusters
for cluster in range(n_clusters):
    print(f"Cluster {cluster}:")
    cluster_data = data[data["Cluster"] == cluster]
    print(cluster_data)

# Visualize the clusters
for cluster in range(n_clusters):
    cluster_data = data[data["Cluster"] == cluster]
    plt.scatter(cluster_data["Age"], cluster_data["Income"], label=f"Cluster
{cluster}")

plt.xlabel("Age")
plt.ylabel("Income")
plt.legend()
plt.title("Hierarchical Agglomerative Clustering")
plt.show()

-----------------------------------------------------------------------------------
---------
slip 7

# Q1. Write a R program to create a sequence of numbers from 20 to 50 and findthe
mean of
# numbers from 20 to 60 and sum of numbers from 51 to 91.

# Create a sequence of numbers from 20 to 50
sequence_20_to_50 <- seq(20, 50)

# Calculate the mean of numbers from 20 to 60
mean_20_to_60 <- mean(sequence_20_to_50[sequence_20_to_50 <= 60])

# Create a sequence of numbers from 51 to 91
sequence_51_to_91 <- seq(51, 91)

# Calculate the sum of numbers from 51 to 91
sum_51_to_91 <- sum(sequence_51_to_91)

# Print the results
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")

# Q2. Consider the following observations/data. And apply simple linear regression
and find out
# estimated coefficients b1 and b1 Also analyse theperformance of the model
# (Use sklearn package)
# x = np.array([1,2,3,4,5,6,7,8])
# y = np.array([7,14,15,18,19,21,26,23])

import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Given data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([7, 14, 15, 18, 19, 21, 26, 23])

# Reshape the data (required by scikit-learn)
x = x.reshape(-1, 1)
y = y.reshape(-1, 1)

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the data
model.fit(x, y)

# Get the estimated coefficients
b0 = model.intercept_[0]
b1 = model.coef_[0][0]

# Make predictions using the model
y_pred = model.predict(x)

# Calculate the performance metrics
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# Print the results
print("Estimated Coefficients:")
print(f"Intercept (b0): {b0}")
print(f"Slope (b1): {b1}")

print("\nPerformance Metrics:")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")

-----------------------------------------------------------------------------------
-------

slip 8

# Q1. Write a R program to get the first 10 Fibonacci numbers.

# Function to generate the first 10 Fibonacci numbers
get_first_10_fibonacci <- function() {
  n <- 10  # Number of Fibonacci numbers to generate
  fibonacci <- numeric(n)

  if (n >= 1) {
    fibonacci[1] <- 0
  }
  if (n >= 2) {
    fibonacci[2] <- 1
  }

  for (i in 3:n) {
    fibonacci[i] <- fibonacci[i - 1] + fibonacci[i - 2]
  }

  return(fibonacci)
}

# Get and print the first 10 Fibonacci numbers
fibonacci_numbers <- get_first_10_fibonacci()
cat("First 10 Fibonacci numbers:", paste(fibonacci_numbers, collapse = ", "))

# Q2. Write a python program to implement k-means algorithm to build prediction
model (Use
# Credit Card Dataset CC GENERAL.csv Download from kaggle.com)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load the credit card dataset
data = pd.read_csv("CCGENERAL.csv")

# Preprocess the data (e.g., handle missing values and scaling)
# You may need to customize this part based on your specific dataset and analysis
goals.

# Select the features you want to use for clustering (e.g., "PURCHASES" and
"CASH_ADVANCE")
X = data[["PURCHASES", "CASH_ADVANCE"]]

# Choose the number of clusters (k) - you can experiment with different values
k = 3

# Create and fit the KMeans model
model = KMeans(n_clusters=k, random_state=42)
model.fit(X)

# Add cluster labels to the original dataset
data["Cluster"] = model.labels_

# Visualize the clusters
for cluster in range(k):
    cluster_data = data[data["Cluster"] == cluster]
    plt.scatter(cluster_data["PURCHASES"], cluster_data["CASH_ADVANCE"],
label=f"Cluster {cluster}")

plt.xlabel("PURCHASES")
plt.ylabel("CASH_ADVANCE")
plt.legend()
plt.title("K-Means Clustering")
plt.show()
----------------------------------------------------------------------------------
------

slip 9

# Q1. Write an R program to create a Data frames which contain details of 5
employees and display
# summary of the data

# Create a data frame with details of 5 employees
employee_data <- data.frame(
  EmployeeID = c(1, 2, 3, 4, 5),
  Name = c("saurabh", "yogesh", "arbaj", "pranav", "hrushali"),
  Age = c(30, 28, 32, 25, 34),
  Department = c("HR", "Engineering", "Finance", "Marketing", "Sales"),
  Salary = c(50000, 60000, 55000, 48000, 65000)
)

# Display the summary of the data frame
summary(employee_data)

# Q2. Write a Python program to build an SVM model to Cancer dataset. The dataset
is
# available in the scikit-learn library. Check the accuracyof model with precision
and
# recall.

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Load the Breast Cancer dataset
data = datasets.load_breast_cancer()
X = data.data
y = data.target

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create an SVM model
svm_model = SVC(kernel='linear')

# Train the model on the training data
svm_model.fit(X_train, y_train)
```

```
# Make predictions on the testing data
y_pred = svm_model.predict(X_test)

# Calculate accuracy, precision, and recall
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
------------------------------------------------------------------------------------
------

slip 10

# Q1. Write a R program to find the maximum and the minimum value of a givenvector
[10
# Marks]

# Create a vector
my_vector <- c(12, 34, 5, 23, 9, 2, 45, 8, 31)

# Find the maximum value
max_value <- max(my_vector)

# Find the minimum value
min_value <- min(my_vector)

# Print the maximum and minimum values
cat("Maximum value:", max_value, "\n")
cat("Minimum value:", min_value, "\n")

# Q2. Write a Python Programme to read the dataset (“Iris.csv”). dataset download
from
# (https://archive.ics.uci.edu/ml/datasets/iris) and apply Apriori algorithm.

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Read the Iris dataset
data = pd.read_csv("iris.csv")

# Split the data into features (X) and target labels (y)
X = data.drop("species", axis=1)
y = data["species"]
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Create a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
-----------------------------------------------------------------------------------
--------
slip 11

# Q1. Write a R program to find all elements of a given list that are not inanother
given list.
# = list("x", "y", "z")
# = list("X", "Y", "Z", "x", "y", "z")

# Define the two lists
list1 <- list("x", "y", "z")
list2 <- list("X", "Y", "Z", "x", "y", "z")

# Find elements in list1 that are not in list2
elements_not_in_list2 <- list1[!(list1 %in% list2)]

# Print the result
print(elements_not_in_list2)

# output= list()


# or program

# Define the two lists
list1 <- list("x", "y", "z")
list2 <- list("X", "Y", "Z", "x", "y", "z")

# Convert the lists to vectors (if not already)
vector1 <- unlist(list1)
vector2 <- unlist(list2)

# Find elements in list1 that are not in list2
```

```
elements_not_in_list2 <- vector1[!(vector1 %in% vector2)]

# Print the result
print(elements_not_in_list2)

# output= character(0)

# Q2. Write a python program to implement hierarchical clustering
algorithm.(Download
# Wholesale customers data dataset from github.com).

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering

# Load the Wholesale Customers dataset (or replace with your dataset)
# Download the dataset and replace 'wholesale_customers_data.csv' with the file
path
data = pd.read_csv('wholesale_customers_data.csv')

# Select the features for clustering (e.g., 'Fresh', 'Milk', 'Grocery', etc.)
# Customize this based on your dataset and analysis goals
selected_features = ['Fresh', 'Milk', 'Grocery']

# Prepare the data
X = data[selected_features]

# Standardize the data to have zero mean and unit variance
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Perform hierarchical clustering
linkage_matrix = linkage(X_std, method='ward')

# Plot the dendrogram
plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix, labels=data.index, leaf_rotation=90, leaf_font_size=12)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Wholesale Customers")
plt.ylabel("Distance")
plt.show()

# Fit Agglomerative Clustering with a specific number of clusters (e.g., 3)
n_clusters = 3
model = AgglomerativeClustering(n_clusters=n_clusters)
data['Cluster'] = model.fit_predict(X_std)
```

```
# Print the clusters
for cluster_id in range(n_clusters):
    cluster_data = data[data['Cluster'] == cluster_id]
    print(f"Cluster {cluster_id}:\n{cluster_data[selected_features]}\n")

# You can explore the cluster assignments and analyze the results as needed.
-----------------------------------------------------------------------------------
-----
slip 12
# Q1. Write a R program to create a Dataframes which contain details of 5employees
and
# display the details.
# Employee contain (empno,empname,gender,age,designation)

# Create a DataFrame for employee details
employee_data <- data.frame(
  empno = c(1, 2, 3, 4, 5),
  empname = c("saurabh", "pranav", "yogesh", "hrushali", "arbaj"),
  gender = c("Male", "Male", "Male", "Female", "Male"),
  age = c(20, 20, 20, 20, 20),
  designation = c("Manager", "Engineer", "Analyst", "Manager", "Designer")
)

# Display the employee details
print(employee_data)

# Q2. Write a python program to implement multiple Linear Regression modelfor a car
dataset.
# Dataset can be downloaded from:
# https://www.w3schools.com/python/python_ml_multiple_regression.asp

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset from data.csv
df = pd.read_csv('data.csv')

# Split the data into features (X) and the target (y)
X = df[['Volume', 'Weight']]
y = df['CO2']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and fit a linear regression model
model = LinearRegression()
```

```
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Mean Squared Error (MSE):", mse)
print("R-squared (R2):", r2)

----------------------------------------------------------------------------------
------
slip 13

# Q1. Draw a pie chart using R programming for the following datadistribution:
# Digits on
# Dice
# 1 2 3 4 5 6
# Frequency of
# getting each
# number
# 7 2 6 3 4 8


# Dice roll data
numbers <- c(1, 2, 3, 4, 5, 6)
frequency <- c(7, 2, 6, 3, 4, 8)

# Create a pie chart
pie(frequency, labels = numbers, main = "Dice Roll Frequencies", col =
rainbow(length(numbers)))

# Add a legend
legend("topright", numbers, fill = rainbow(length(numbers))

# Add a title
title("Dice Roll Frequencies")

# Display the pie chart

# Q2. Write a Python program to read “StudentsPerformance.csv” file.
Solvefollowing:
# - To display the shape of dataset.
# - To display the top rows of the dataset with their columns.Note: Download
# dataset from following link :
# (https://www.kaggle.com/spscientist/students-performance-inexams?
# select=StudentsPerformance.csv)

import pandas as pd

# Load the dataset from the CSV file
df = pd.read_csv("StudentsPerformance.csv")

# Display the shape of the dataset (number of rows and columns)
print("Shape of the dataset:", df.shape)

# Display the top rows of the dataset with their columns
print("Top rows of the dataset:")
print(df.head())
-----------------------------------------------------------------------------------
----

slip 14
# Q1. Write a script in R to create a list of employees (name) and perform
thefollowing:
# a. Display names of employees in the list.
# b. Add an employee at the end of the list
# c. Remove the third element of the list.

# Create a list of employees' names
employees <- list("saurabh", "pranav", "hrushali", "yogesh")

# a. Display names of employees in the list
print(employees)

# b. Add an employee at the end of the list
new_employee <- "arbaj"
employees <- c(employees, new_employee)
print(employees)

# c. Remove the third element of the list
removed_employee <- employees[3]
employees <- employees[-3]

print(removed_employee)
print(employees)

# Q2. Write a Python Programme to apply Apriori algorithm on Groceries dataset.
Dataset
# can be downloaded from
# (https://github.com/amankharwal/Websitedata/blob/master/Groceries
# _dataset.csv).
# Also display support and confidence for each rule.

# pip install apyori
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from apyori import apriori

store_data = pd.read_csv('Groceries_dataset.csv', header=None)

records = []
for i in range(0, 300):
    records.append([str(store_data.values[i, j]) for j in range(0, 3)])

association_rules = apriori(records, min_support=0.0045, min_confidence=0.2,
min_lift=3, min_length=2)
association_results = list(association_rules)

print(len(association_results))
print(association_results[0])

for item in association_results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])
    print("Support: " + str(item[1]))
    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("=========================================")

-----------------------------------------------------------------------------------
--------
# Q1.Write a R program to add, multiply and divide two vectors of integer
type.(vector length
# should be minimum 4)

# Create two integer vectors of the same length (minimum 4)
vector1 <- c(5, 10, 15, 20)
vector2 <- c(2, 4, 5, 8)

# Perform vector addition
addition_result <- vector1 + vector2
cat("Vector Addition Result: ", addition_result, "\n")

# Perform vector multiplication
multiplication_result <- vector1 * vector2
cat("Vector Multiplication Result: ", multiplication_result, "\n")

# Perform vector division
division_result <- vector1 / vector2
cat("Vector Division Result: ", division_result, "\n")

# Q2. Write a Python program build Decision Tree Classifier forshows.csvfrom pandas
and
```

```
# predict class label for show starring a 40 years old American comedian, with 10
# years of experience, and a comedy ranking of 7? Create a csv file as shown in
# https://www.w3schools.com/python/python_ml_decision_tree.asp

import pandas as pd
from sklearn.tree import DecisionTreeClassifier

# Load the dataset from "shows.csv"
data = pd.read_csv("shows.csv")

# Preprocess the data: Map 'Nationality' to numeric values
data['Nationality'] = data['Nationality'].apply(lambda x: 1 if x == 'American' else
0)

# Define features and target variable
X = data[['Age', 'Nationality', 'Experience', 'Rank']]
y = data['Go']

# Create a Decision Tree Classifier
clf = DecisionTreeClassifier()

# Fit the classifier to the data
clf = clf.fit(X, y)

# Create an input data for prediction
input_data = pd.DataFrame({'Age': [40], 'Nationality': [1], 'Experience': [10],
'Rank': [7]})

# Predict the class label
predicted_label = clf.predict(input_data)

# Create a DataFrame to store the input and predicted label
result_df = pd.DataFrame({'Age': input_data['Age'],
                          'Nationality': ['American' if
input_data['Nationality'].values[0] == 1 else 'Other'],
                          'Experience': input_data['Experience'],
                          'Rank': input_data['Rank'],
                          'Predicted Label': predicted_label})

# Save the result to a CSV file
result_df.to_csv('predicted_show_label.csv', index=False)

print("Predicted Label:", predicted_label)
print("Result saved to predicted_show_label.csv")

------------------------------------------------------------------------------------
-----
slip 16

# Q1. Write a R program to create a simple bar plot of given data
# Year Export Import
# 2001 26 35
# 2002 32 40
# 2003 35 50

# Create a data frame for the given data
data <- data.frame(
  Year = c(2001, 2002, 2003),
  Export = c(26, 32, 35),
  Import = c(35, 40, 50)
)

# Create a bar plot
barplot(height = t(data[, c("Export", "Import")]),
        beside = TRUE,
        names.arg = data$Year,
        col = c("blue", "red"),
        legend.text = c("Export", "Import"),
        args.legend = list(title = "Type"))

# Add axis labels and a title
x <- c("2001", "2002", "2003")
xlabel <- "Year"
ylabel <- "Value"
title <- "Export and Import by Year"

axis(1, at = 1:3, labels = x, pos = 0)
axis(2, las = 1, at = seq(0, 60, by = 10))
title(main = title, xlab = xlabel, ylab = ylabel)

# Q2. Write a Python program build Decision Tree Classifier using
Scikit-learnpackage for
# diabetes data set (download database from
https://www.kaggle.com/uciml/pima-indians diabetes-database)

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset from the CSV file
data = pd.read_csv("diabetes.csv")

# Define features (X) and the target variable (y)
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Create a Decision Tree Classifier
clf = DecisionTreeClassifier()

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the Decision Tree Classifier:", accuracy)

--------------------------------------------------------------------------------------
------
slip 17
# Q1. Write a R program to get the first 20 Fibonacci numbers

# Initialize the first two Fibonacci numbers
fibonacci <- c(0, 1)

# Calculate and print the first 20 Fibonacci numbers
for (i in 3:20) {
  next_fib <- fibonacci[i-1] + fibonacci[i-2]
  fibonacci <- c(fibonacci, next_fib)
}

# Print the first 20 Fibonacci numbers
cat("The first 20 Fibonacci numbers are:\n")
cat(fibonacci, sep = " ")

# Q2. Write a python programme to implement multiple linear regression modelfor
stock market
# data frame as follows:
# Stock_Market = {'Year':
# [2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2016,2
# 016,20,16,2016,2016,2016,2016,2016,2016,2016,2016,2016],
# 'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1],
# 'Interest_Rate':
[2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.75,1.75,1.75,1.75,1.75,1
# .75,1.75,1.75,1.75,1.75,1.75],
# 'Unemployment_Rate':
# [5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,5.8,6.1,6.2,6.1,6.1,6.1,5
# .9,6.2,6.2,6.1],
# 'Stock_Index_Price':
[1464,1394,1357,1293,1256,1254,1234,1195,1159,1167,1130,1075,1047,
# 965,943,958,971,949,884,866,876,822,704,719] }
# And draw a graph of stock market price verses interest rate.
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Define the stock market data
Stock_Market = {
    'Year': [2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017,
2017, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016],
    'Month': [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 12, 11, 10, 9, 8, 7, 6, 5, 4,
3, 2, 1],
    'Interest_Rate': [2.75, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.25, 2.25, 2.25, 2, 2,
2, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75],
    'Unemployment_Rate': [5.3, 5.3, 5.3, 5.3, 5.4, 5.6, 5.5, 5.5, 5.5, 5.6, 5.7,
5.9, 6, 5.9, 5.8, 6.1, 6.2, 6.1, 6.1, 6.1, 5.9, 6.2, 6.2, 6.1],
    'Stock_Index_Price': [1464, 1394, 1357, 1293, 1256, 1254, 1234, 1195, 1159,
1167, 1130, 1075, 1047, 965, 943, 958, 971, 949, 884, 866, 876, 822, 704, 719]
}

# Create a DataFrame from the data
df = pd.DataFrame(Stock_Market)

# Extract the features and target variable
X = df[['Interest_Rate']]
y = df['Stock_Index_Price']

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict stock market prices
predicted_prices = model.predict(X)

# Plot the actual and predicted prices against interest rate
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual Prices')
plt.plot(X, predicted_prices, color='red', label='Predicted Prices')
plt.xlabel('Interest Rate')
plt.ylabel('Stock Index Price')
plt.legend()
plt.title('Stock Market Price vs. Interest Rate')
plt.show()
------------------------------------------------------------------------------------
--------slip 18
# Q1. Write a R program to find the maximum and the minimum value of a givenvector

# Create a vector
my_vector <- c(34, 56, 12, 98, 23, 45, 67)

# Find the maximum and minimum values
```

```
max_value <- max(my_vector)
min_value <- min(my_vector)

# Print the maximum and minimum values
cat("Maximum Value:", max_value, "\n")
cat("Minimum Value:", min_value, "\n")

# Q2. Consider the following observations/data. And apply simple linear regression
and find out
# estimated coefficients b1 and b1 Also analyse theperformance of the model
# (Use sklearn package)
# x = np.array([1,2,3,4,5,6,7,8])
# y = np.array([7,14,15,18,19,21,26,23])

import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Input data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([7, 14, 15, 18, 19, 21, 26, 23])

# Reshape x to a 2D array
x = x.reshape(-1, 1)

# Create a linear regression model
regression_model = LinearRegression()

# Fit the model to the data
regression_model.fit(x, y)

# Get the estimated coefficients
b0 = regression_model.intercept_
b1 = regression_model.coef_[0]

# Make predictions
y_pred = regression_model.predict(x)

# Calculate Mean Squared Error
mse = mean_squared_error(y, y_pred)

# Calculate R-squared
r_squared = r2_score(y, y_pred)

print(f"Estimated Coefficient b0 (Intercept): {b0:.4f}")
print(f"Estimated Coefficient b1 (Slope): {b1:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R2): {r_squared:.4f}")

-----------------------------------------------------------------------------------
```

```
--------
slip 19
# Q1. Write aR program to create a Dataframes which contain details of 5
Studentsand display the
# details.
# Students contain (Rollno,Studname,Address,Marks)

# Create a data frame with student details
students_data <- data.frame(
  Rollno = c(1, 2, 3),
  Studname = c("saurabh", "yogesh", "arbaj"),
  Address = c("wagholi", "pune", "hadapsar"),
  Marks = c(78, 92, 88)
)

# Display the details of the students
print(students_data)

# Q2. Write a python program to implement multiple Linear Regression modelfor a car
dataset.
# Dataset can be downloaded from:
# https://www.w3schools.com/python/python_ml_multiple_regression.asp

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the car dataset from the CSV file
data = pd.read_csv("car_data.csv")

# Define the features (independent variables) and the target variable (dependent
variable)
X = data[['Weight', 'Volume']]
y = data['CO2']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a multiple linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate Mean Squared Error
```

```
mse = mean_squared_error(y_test, y_pred)

# Calculate R-squared
r_squared = r2_score(y_test, y_pred)

# Display the model coefficients (b0, b1, b2)
b0 = model.intercept_
b1, b2 = model.coef_

print("Model Coefficients:")
print(f"Intercept (b0): {b0:.4f}")
print(f"Weight (b1): {b1:.4f}")
print(f"Volume (b2): {b2:.4f}")

print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R2): {r_squared:.4f}")
-----------------------------------------------------------------------------------
-------
slip 20
# Q1. Write a R program to create a data frame from four given vectors

# Define four vectors
vector1 <- c(1, 2, 3, 4, 5)
vector2 <- c("A", "B", "C", "D", "E")
vector3 <- c(10.5, 20.5, 30.5, 40.5, 50.5)
vector4 <- c(TRUE, FALSE, TRUE, FALSE, TRUE)

# Create a data frame from the vectors
df <- data.frame(
  int_num = vector1,
  char = vector2,
  float_num = vector3,
  boolean = vector4
)

# Display the data frame
print(df)

# Q2. Write a python program to implement hierarchical Agglomerativeclustering
algorithm.
# (Download Customer.csv dataset from github.com).

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

# Load the Customer.csv dataset
data = pd.read_csv("Customer.csv")
```

```
# Extract the features (attributes) for clustering
X = data[['Annual Income (k$)', 'Spending Score (1-100)']]

# Perform hierarchical clustering
linkage_matrix = linkage(X, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix)
plt.title('Hierarchical Agglomerative Clustering Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()

# Fit the Agglomerative Clustering model
n_clusters = 3  # You can adjust the number of clusters as needed
model = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')
model.fit(X)

# Add cluster labels to the dataset
data['Cluster'] = model.labels_

# Display the dataset with cluster labels
print(data)

-----------------------------------------------------------------------------------
-------
```