**Slip 1**

**Q1.Write a R program to add, multiply and divide two vectors of integer type. (Vector**

**length should be minimum 4) [10 Marks]**

vector1 <- c(4, 8, 12, 16)

vector2 <- c(2, 4, 6, 8)

add\_vectors <- function(v1, v2) {

return(v1 + v2)

}

multiply\_vectors <- function(v1, v2) {

return(v1 \* v2)

}

divide\_vectors <- function(v1, v2) {

return(ifelse(v2 != 0, v1 / v2, NA))

}

sum\_result <- add\_vectors(vector1, vector2)

product\_result <- multiply\_vectors(vector1, vector2)

division\_result <- divide\_vectors(vector1, vector2)

cat("Vector 1: ", vector1, "\n")

cat("Vector 2: ", vector2, "\n")

cat("Sum: ", sum\_result, "\n")

cat("Product: ", product\_result, "\n")

cat("Division: ", division\_result, "\n")

**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. [20 Marks]**

pip install pandas numpy scikit-learn // terminal command

jupyter notebook

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

url = 'https://drive.google.com/uc?id=1oakZCv7g3mlmCSdv9J8kdSaqO5\_6dIOw'

data = pd.read\_csv(url)

print("Dataset preview:")

print(data.head())

X = data[['Hours']] # Predictor variable

y = data['Scores'] # Response variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

print(f"Mean Absolute Error: {mae}")

print(f"Mean Squared Error: {mse}")

print(f"Root Mean Squared Error: {rmse}")

**Slip 2**

**Q1. Write an R program to calculate the multiplication table using a function.**

**[10 Marks]**

generate\_multiplication\_table <- function(number, max\_multiplier) {

for (i in 1:max\_multiplier) {

result <- number \* i

cat(number, "x", i, "=", result, "\n")

}

}

number <- as.integer(readline(prompt = "Enter a number to generate its multiplication table: "))

max\_multiplier <- as.integer(readline(prompt = "Enter the maximum multiplier: "))

cat("Multiplication Table for", number, ":\n")

generate\_multiplication\_table(number, max\_multiplier)

**Q2. Write a python program to implement k-means algorithms on a synthetic**

**dataset. [20 Marks]**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

n\_samples = 300

n\_clusters = 4

random\_state = 42

X, y\_true = make\_blobs(n\_samples=n\_samples, centers=n\_clusters, cluster\_std=0.60, random\_state=random\_state)

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=random\_state)

y\_kmeans = kmeans.fit\_predict(X)

plt.figure(figsize=(10, 6))

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X')

plt.title('K-Means Clustering on Synthetic Dataset')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.grid()

plt.show()

**Slip 3**

**Q1. Write a R program to reverse a number and also calculate the sum of digits of that**

**number. [10 Marks]**

reverse\_and\_sum <- function(number) {

original\_number <- number

reversed\_number <- 0

sum\_of\_digits <- 0

while (number > 0) {

digit <- number %% 10

reversed\_number <- reversed\_number \* 10 + digit

sum\_of\_digits <- sum\_of\_digits + digit

number <- number %/% 10

}

return(list(reversed = reversed\_number, sum = sum\_of\_digits))

}

number <- as.integer(readline(prompt = "Enter a number: "))

result <- reverse\_and\_sum(number)

cat("Reversed Number:", result$reversed, "\n")

cat("Sum of Digits:", result$sum, "\n")

**Q2. Consider the following observations/data. And apply simple linear regression and find**

**out estimated coefficients b0 and b1.( use numpy package)**

**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] [20 Marks]**

import numpy as np

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])

n = len(x)

x\_mean = np.mean(x)

y\_mean = np.mean(y)

b1\_numerator = np.sum((x - x\_mean) \* (y - y\_mean))

b1\_denominator = np.sum((x - x\_mean) \*\* 2)

b1 = b1\_numerator / b1\_denominator

b0 = y\_mean - b1 \* x\_mean

print(f"Estimated coefficients:\n b0 (intercept): {b0}\n b1 (slope): {b1}")

**Slip 4**

**Q1. Write a R program to calculate the sum of two matrices of given size. [10 Marks]**

input\_matrix <- function(rows, cols, name) {

cat(paste("Enter the elements of matrix", name, "row-wise:\n"))

matrix\_data <- numeric(rows \* cols) # Create a numeric vector to hold the matrix elements

for (i in 1:(rows \* cols)) {

matrix\_data[i] <- as.numeric(readline(prompt = paste("Element", i, ": ")))

}

return(matrix(matrix\_data, nrow = rows, ncol = cols)) # Convert vector to matrix

}

rows <- as.integer(readline(prompt = "Enter the number of rows: "))

cols <- as.integer(readline(prompt = "Enter the number of columns: "))

matrix1 <- input\_matrix(rows, cols, "A")

matrix2 <- input\_matrix(rows, cols, "B")

matrix\_sum <- matrix1 + matrix2

cat("Matrix A:\n")

print(matrix1)

cat("Matrix B:\n")

print(matrix2)

cat("Sum of Matrix A and B:\n")

print(matrix\_sum)

**Q2. Consider following dataset**

**weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny','Rainy','Sunn**

**y','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'].**

**Use Naïve Bayes algorithm to predict [0: Overcast, 2: Mild] tuple belongs to which class**

**whether to play the sports or not.**

**[20 Marks]**

import numpy as np

import pandas as pd

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']

data = pd.DataFrame({

'Weather': weather,

'Temperature': temp,

'Play': play

})

prior\_play = data['Play'].value\_counts(normalize=True)

print("Prior Probabilities:\n", prior\_play)

likelihood = {}

for feature in ['Weather', 'Temperature']:

likelihood[feature] = {}

for class\_value in prior\_play.index:

likelihood[feature][class\_value] = data[data['Play'] == class\_value][feature].value\_counts(normalize=True)

input\_tuple = ('Overcast', 'Mild')

posterior = {}

for class\_value in prior\_play.index:

posterior[class\_value] = prior\_play[class\_value] # Start with prior probability

for i, feature in enumerate(['Weather', 'Temperature']):

posterior[class\_value] \*= likelihood[feature][class\_value].get(input\_tuple[i], 0)

print("\nPosterior Probabilities:\n", posterior)

predicted\_class = max(posterior, key=posterior.get)

print("\nPredicted Class for the input tuple (Weather: 'Overcast', Temperature: 'Mild'):", predicted\_class)

**Slip 5**

**Q1. Write a R program to concatenate two given factors. [10 Marks]**

concatenate\_factors <- function(factor1, factor2) {

combined\_factor <- c(factor1, factor2)

result <- factor(combined\_factor)

return(result)

}

factor1 <- factor(c("Apple", "Banana", "Cherry"))

factor2 <- factor(c("Date", "Elderberry", "Fig"))

result\_factor <- concatenate\_factors(factor1, factor2)

print(result\_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/pimaindians-diabetes-database)**

**[20 Marks]**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import matplotlib.pyplot as plt

from sklearn import tree

data = pd.read\_csv('diabetes.csv')

print(data.head())

print(data.info())

print(data.describe())

X = data.drop('Outcome', axis=1) # Features

y = data['Outcome'] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = DecisionTreeClassifier(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

print('Confusion Matrix:\n', conf\_matrix)

print('Classification Report:\n', class\_report)

plt.figure(figsize=(12, 8))

tree.plot\_tree(model, feature\_names=X.columns, class\_names=['No Diabetes', 'Diabetes'], filled=True)

plt.title("Decision Tree for Diabetes Classification")

plt.show()

**Slip 6**

**Q1. Write a R program to create a data frame using two given vectors and display the duplicate**

**elements. [10 Marks]**

find\_duplicates <- function(vec1, vec2) {

df <- data.frame(Vector1 = vec1, Vector2 = vec2)

cat("Data Frame:\n")

print(df)

duplicates\_vec1 <- vec1[duplicated(vec1) | duplicated(vec1, fromLast = TRUE)]

duplicates\_vec2 <- vec2[duplicated(vec2) | duplicated(vec2, fromLast = TRUE)]

cat("\nDuplicate elements in Vector 1:\n")

print(unique(duplicates\_vec1))

cat("\nDuplicate elements in Vector 2:\n")

print(unique(duplicates\_vec2))

}

vector1 <- c(1, 2, 3, 2, 4, 5, 3, 6)

vector2 <- c("A", "B", "C", "A", "D", "B", "E")

find\_duplicates(vector1, vector2)

**Q2. Write a python program to implement hierarchical Agglomerative clustering algorithm.**

**(Download Customer.csv dataset from github.com).**

**[20 Marks]**

pip install pandas matplotlib seaborn scipy scikit-learn // install terminal

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

from sklearn.preprocessing import StandardScaler

url = 'https://raw.githubusercontent.com/yourusername/yourrepo/main/Customer.csv'

data = pd.read\_csv(url)

data.drop(['CustomerID'], axis=1, inplace=True, errors='ignore')

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

linked = linkage(data\_scaled, method='ward')

plt.figure(figsize=(10, 7))

dendrogram(linked, orientation='top', labels=data.index, distance\_sort='descending', show\_leaf\_counts=True)

plt.title('Dendrogram for Hierarchical Clustering')

plt.xlabel('Customers')

plt.ylabel('Euclidean distances')

plt.show()

num\_clusters = 3

clusters = fcluster(linked, num\_clusters, criterion='maxclust')

data['Cluster'] = clusters

plt.figure(figsize=(10, 6))

sns.scatterplot(x=data[data.columns[0]], y=data[data.columns[1]], hue=data['Cluster'], palette='deep')

plt.title('Hierarchical Agglomerative Clustering Results')

plt.xlabel(data.columns[0])

plt.ylabel(data.columns[1])

plt.legend(title='Cluster')

plt.show()

**Slip 7**

**Q1. Write a R program to create a sequence of numbers from 20 to 50 and find the mean of**

**numbers from 20 to 60 and sum of numbers from 51 to 91.**

**[10 Marks]**

sequence\_20\_to\_50 <- 20:50

print("Sequence from 20 to 50:")

print(sequence\_20\_to\_50)

mean\_20\_to\_60 <- mean(20:60)

cat("Mean of numbers from 20 to 60:", mean\_20\_to\_60, "\n")

sum\_51\_to\_91 <- sum(51:91)

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 the performance 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]) [20 Marks]**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

x = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(-1, 1) # Reshape for sklearn

y = np.array([7, 14, 15, 18, 19, 21, 26, 23])

model = LinearRegression()

model.fit(x, y)

b0 = model.intercept\_ # y-intercept

b1 = model.coef\_[0] # slope

print(f"Estimated coefficients:\nb0 (intercept) = {b0}\nb1 (slope) = {b1}")

y\_pred = model.predict(x)

mae = mean\_absolute\_error(y, y\_pred)

mse = mean\_squared\_error(y, y\_pred)

r2 = r2\_score(y, y\_pred)

print(f"\nModel Performance:\nMean Absolute Error (MAE) = {mae}\nMean Squared Error (MSE) = {mse}\nR² Score = {r2}")

plt.scatter(x, y, color='blue', label='Actual data')

plt.plot(x, y\_pred, color='red', label='Fitted line')

plt.xlabel('X values')

plt.ylabel('Y values')

plt.title('Simple Linear Regression')

plt.legend()

plt.show()

**Slip 8**

**Q1. Write a R program to get the first 10 Fibonacci numbers. [10 Marks]**

fibonacci <- function(n) {

fib\_sequence <- numeric(n)

fib\_sequence[1] <- 0 # First Fibonacci number

fib\_sequence[2] <- 1 # Second Fibonacci number

for (i in 3:n) {

fib\_sequence[i] <- fib\_sequence[i - 1] + fib\_sequence[i - 2] # Calculate next Fibonacci number

}

return(fib\_sequence)

}

first\_10\_fib <- fibonacci(10)

cat("The first 10 Fibonacci numbers are:\n")

print(first\_10\_fib)

**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) [20 Marks]**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

url = 'https://raw.githubusercontent.com/your-repo/CC-GENERAL.csv' # Update with actual path if needed

data = pd.read\_csv(url)

data.drop(columns=['CUST\_ID'], inplace=True)

data.fillna(0, inplace=True) # Replace missing values with 0

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

inertia = []

silhouette\_scores = []

K = range(2, 11)

for k in K:

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(data\_scaled)

inertia.append(kmeans.inertia\_)

silhouette\_scores.append(silhouette\_score(data\_scaled, kmeans.labels\_))

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(K, inertia, marker='o')

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia')

plt.subplot(1, 2, 2)

plt.plot(K, silhouette\_scores, marker='o')

plt.title('Silhouette Scores for k')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Silhouette Score')

plt.show()

optimal\_k = 5

kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)

data['Cluster'] = kmeans.fit\_predict(data\_scaled)

cluster\_centers = scaler.inverse\_transform(kmeans.cluster\_centers\_)

print("Cluster Centers (original scale):")

print(cluster\_centers)

print("\nNumber of points in each cluster:")

print(data['Cluster'].value\_counts())

plt.scatter(data\_scaled[:, 0], data\_scaled[:, 1], c=data['Cluster'], cmap='viridis')

plt.title('Clusters Visualization')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

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. [10 Marks]**

employee\_data <- data.frame(

EmployeeID = 1:5,

Name = c("John Doe", "Jane Smith", "Alice Johnson", "Bob Brown", "Charlie Davis"),

Age = c(28, 34, 29, 40, 37),

Department = c("HR", "Finance", "IT", "Marketing", "Sales"),

Salary = c(50000, 60000, 55000, 62000, 58000)

)

print("Employee Data:")

print(employee\_data)

print("Summary of Employee Data:")

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 accuracy of model with precision and**

**recall. [20 Marks]**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, classification\_report

data = load\_breast\_cancer()

X = data.data # Features

y = data.target # Target labels

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = SVC(kernel='linear') # You can choose different kernels like 'rbf', 'poly', etc.

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

**Slip 10**

**Q1. Write a R program to find the maximum and the minimum value of a given vector [10**

**Marks]**

numbers <- c(10, 25, 5, 40, 15, 30, 2)

max\_value <- max(numbers)

min\_value <- min(numbers)

cat("The maximum value in the vector is:", max\_value, "\n")

cat("The minimum value in the vector is:", 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. [20 Marks]**

pip install pandas mlxtend // terminal command

import pandas as pd

from mlxtend.frequent\_patterns import apriori, association\_rules

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

columns = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'species']

iris\_data = pd.read\_csv(url, header=None, names=columns)

print("Iris Dataset:")

print(iris\_data.head())

iris\_data['species'] = iris\_data['species'].astype('category')

one\_hot = pd.get\_dummies(iris\_data, columns=['species'])

frequent\_itemsets = apriori(one\_hot, min\_support=0.1, use\_colnames=True)

print("\nFrequent Itemsets:")

print(frequent\_itemsets)

rules = association\_rules(frequent\_itemsets, metric="support", min\_threshold=0.1)

print("\nAssociation Rules:")

print(rules)

**Slip 11**

**Q1. Write a R program to find all elements of a given list that are not in another given list.**

**= list("x", "y", "z")**

**= list("X", "Y", "Z", "x", "y", "z") [10 Marks]**

list1 <- list("x", "y", "z")

list2 <- list("X", "Y", "Z", "x", "y", "z")

vector1 <- unlist(list1)

vector2 <- unlist(list2)

not\_in\_list2 <- vector1[!(vector1 %in% vector2)]

cat("Elements in list1 that are not in list2:", not\_in\_list2, "\n")

**Q2. Write a python program to implement hierarchical clustering algorithm. (Download**

**Wholesale customers data dataset from github.com).**

pip install pandas scipy matplotlib seaborn // terminal command

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.cluster.hierarchy import dendrogram, linkage

from sklearn.preprocessing import StandardScaler

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/Wholesale%20customers.csv"

data = pd.read\_csv(url)

print("Dataset Overview:")

print(data.head())

features = data.drop(['Channel', 'Region'], axis=1)

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(features)

linked = linkage(scaled\_features, method='ward')

plt.figure(figsize=(10, 7))

dendrogram(linked, orientation='top', labels=data.index, distance\_sort='descending', show\_leaf\_counts=True)

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('Customer Index')

plt.ylabel('Distance')

plt.show()

**Slip 12**

**Q1. Write a R program to create a Dataframes which contain details of 5 employees and**

**display the details.**

**Employee contain (empno,empname,gender,age,designation)**

**[10 Marks]**

employees <- data.frame(

empno = c(101, 102, 103, 104, 105),

empname = c("Alice", "Bob", "Charlie", "David", "Eva"),

gender = c("Female", "Male", "Male", "Male", "Female"),

age = c(25, 30, 28, 35, 27),

designation = c("Manager", "Developer", "Designer", "Analyst", "Tester"),

stringsAsFactors = FALSE # Avoid converting strings to factors

)

print("Employee Details:")

print(employees)

**Q2. Write a python program to implement multiple Linear Regression model for a car dataset.**

**Dataset can be downloaded from:**

**https://www.w3schools.com/python/python\_ml\_multiple\_regression.asp**

**[20 Marks]**

import pandas as pd

import numpy as np

import statsmodels.api as sm

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

url = "https://www.w3schools.com/python/pandas/data/carprices.csv"

data = pd.read\_csv(url)

print("Dataset Head:")

print(data.head())

X = data[['Age', 'Mileage']]

y = data['Price']

X = sm.add\_constant(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = sm.OLS(y\_train, X\_train).fit()

print("\nModel Summary:")

print(model.summary())

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("\nPerformance Metrics:")

print(f"Mean Squared Error: {mse:.2f}")

print(f"R-squared: {r2:.2f}")

**Slip 13**

**Q1. Draw a pie chart using R programming for the following data distribution:**

**Digits on**

**Dice**

**1 2 3 4 5 6**

**Frequency of**

**getting each**

**number**

**7 2 6 3 4 8**

**[10 Marks]**

digits\_on\_dice <- c(1, 2, 3, 4, 5, 6)

frequency <- c(7, 2, 6, 3, 4, 8)

labels <- paste("Digit:", digits\_on\_dice, "\nFrequency:", frequency)

pie(frequency,

labels = labels,

main = "Frequency of Getting Each Number on Dice",

col = rainbow(length(digits\_on\_dice)))

legend("topright",

legend = labels,

fill = rainbow(length(digits\_on\_dice)),

cex = 0.8)

**Q2. Write a Python program to read “StudentsPerformance.csv” file. Solve following:**

**- 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) [20 Marks]**

import pandas as pd

url = "https://raw.githubusercontent.com/your\_username/your\_repo/main/StudentsPerformance.csv"

data = pd.read\_csv(url)

print("Shape of the dataset:", data.shape)

print("\nTop rows of the dataset:")

print(data.head())

**Slip 14**

**Q1. Write a script in R to create a list of employees (name) and perform the following:**

**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. [10 Marks]**

employees <- list("Alice", "Bob", "Charlie", "David", "Eve")

cat("Employees in the list:\n")

print(employees)

new\_employee <- "Frank"

employees <- append(employees, new\_employee)

cat("\nEmployees after adding a new employee:\n")

print(employees)

employees <- employees[-3] # Removing the third element

cat("\nEmployees after removing the third element:\n")

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.**

**[20 Marks]**

import pandas as pd

from mlxtend.frequent\_patterns import apriori, association\_rules

url = 'https://github.com/amankharwal/Websitedata/raw/master/Groceries\_dataset.csv'

data = pd.read\_csv(url)

print("Data Sample:")

print(data.head())

transactions = data.groupby(['Member\_number', 'Date'])['Item\_description'].apply(list).tolist()

from mlxtend.preprocessing import TransactionEncoder

encoder = TransactionEncoder()

onehot = encoder.fit(transactions).transform(transactions)

onehot\_df = pd.DataFrame(onehot, columns=encoder.columns\_)

frequent\_itemsets = apriori(onehot\_df, min\_support=0.01, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.1)

print("\nAssociation Rules:")

print(rules[['antecedents', 'consequents', 'support', 'confidence']])

**Slip 15**

**Q1.Write a R program to add, multiply and divide two vectors of integer type. (vector length**

**should be minimum 4) [10 Marks]**

vector1 <- c(10, 20, 30, 40)

vector2 <- c(2, 4, 5, 10)

addition <- vector1 + vector2

multiplication <- vector1 \* vector2

division <- vector1 / vector2

cat("Vector 1:", vector1, "\n")

cat("Vector 2:", vector2, "\n")

cat("Addition Result:", addition, "\n")

cat("Multiplication Result:", multiplication, "\n")

cat("Division Result:", division, "\n")

**Q2. Write a Python program build Decision Tree Classifier for shows.csv from 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**

**[20 Marks]**

Create CSV file :-

Age,Experience,ComedyRanking,ClassLabel

25,2,5,Not Popular

30,5,6,Popular

35,8,8,Popular

40,10,7,Popular

45,12,5,Not Popular

50,15,4,Not Popular

Code :-

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

data = pd.read\_csv('shows.csv')

print("Dataset:\n", data)

X = data[['Age', 'Experience', 'ComedyRanking']]

y = data['ClassLabel']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

new\_show = [[40, 10, 7]] # 40 years old, 10 years of experience, comedy ranking of 7

predicted\_class = clf.predict(new\_show)

print(f"The predicted class label for the show is: {predicted\_class[0]}")

**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**

**[10 Marks]**

data <- data.frame(

Year = c(2001, 2002, 2003),

Export = c(26, 32, 35),

Import = c(35, 40, 50)

)

library(ggplot2)

data\_long <- reshape2::melt(data, id.vars = "Year")

ggplot(data\_long, aes(x = Year, y = value, fill = variable)) +

geom\_bar(stat = "identity", position = "dodge") +

labs(title = "Export and Import Data (2001-2003)",

x = "Year",

y = "Value",

fill = "Category") +

theme\_minimal()

**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-indiansdiabetes-database)**

**[20 Marks]**

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, confusion\_matrix

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

column\_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',

'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=column\_names)

print("First few rows of the dataset:")

print(data.head())

X = data.drop('Outcome', axis=1) # Features

y = data['Outcome'] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = DecisionTreeClassifier(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print("\nAccuracy of the model: {:.2f}%".format(accuracy \* 100))

print("\nConfusion Matrix:")

print(conf\_matrix)

print("\nClassification Report:")

print(class\_report)

**Slip 17**

**Q1. Write a R program to get the first 20 Fibonacci numbers.**

**[10 Marks]**

fibonacci <- function(n) {

fib\_sequence <- numeric(n) # Create a numeric vector of length n

fib\_sequence[1] <- 0 # First Fibonacci number

fib\_sequence[2] <- 1 # Second Fibonacci number

for (i in 3:n) {

fib\_sequence[i] <- fib\_sequence[i - 1] + fib\_sequence[i - 2] # Calculate next Fibonacci number

}

return(fib\_sequence)

}

first\_20\_fib <- fibonacci(20)

print("The first 20 Fibonacci numbers are:")

print(first\_20\_fib)

**Q2. Write a python programme to implement multiple linear regression model for 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.**

**[20 Marks]**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

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, 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]

}

df = pd.DataFrame(Stock\_Market)

X = df[['Interest\_Rate', 'Unemployment\_Rate']]

y = df['Stock\_Index\_Price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

print("Coefficients:")

print(f"Intercept (b0): {model.intercept\_}")

print(f"Interest Rate (b1): {model.coef\_[0]}")

print(f"Unemployment Rate (b2): {model.coef\_[1]}")

plt.figure(figsize=(10, 6))

plt.scatter(df['Interest\_Rate'], df['Stock\_Index\_Price'], color='blue', label='Data Points')

plt.title('Stock Index Price vs Interest Rate')

plt.xlabel('Interest Rate (%)')

plt.ylabel('Stock Index Price')

plt.grid()

plt.legend()

plt.show()

**Slip 18**

**Q1. Write a R program to find the maximum and the minimum value of a given vector [10**

**Marks]**

numbers <- c(12, 5, 8, 19, 3, 27, 15)

max\_value <- max(numbers)

min\_value <- min(numbers)

cat("The maximum value is:", max\_value, "\n")

cat("The minimum value is:", min\_value, "\n")

**Q2. Consider the following observations/data. And apply simple linear regression and find out**

**estimated coefficients b1 and b1 Also analyse the performance 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]) [20 Marks]**

import numpy as np

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

x = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(-1, 1) # Reshape for sklearn

y = np.array([7, 14, 15, 18, 19, 21, 26, 23])

model = LinearRegression()

model.fit(x, y)

b0 = model.intercept\_

b1 = model.coef\_[0]

y\_pred = model.predict(x)

mse = mean\_squared\_error(y, y\_pred)

r2 = r2\_score(y, y\_pred)

print(f"Estimated coefficients:")

print(f"b0 (Intercept): {b0}")

print(f"b1 (Slope): {b1}")

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

**Slip 19**

**Q1. Write a R program to create a Dataframes which contain details of 5 Students and display the**

**details.**

**Students contain (Rollno,Studname,Address,Marks) [10 Marks]**

Rollno <- c(1, 2, 3, 4, 5)

Studname <- c("Alice", "Bob", "Charlie", "David", "Eve")

Address <- c("123 Maple St", "456 Oak St", "789 Pine St", "135 Birch St", "246 Cedar St")

Marks <- c(85, 90, 78, 88, 92)

students\_df <- data.frame(Rollno, Studname, Address, Marks)

print(students\_df)

**Q2. Write a python program to implement multiple Linear Regression model for a car dataset.**

**Dataset can be downloaded from:**

**https://www.w3schools.com/python/python\_ml\_multiple\_regression.asp**

**[20 Marks]**

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

url = "https://www.w3schools.com/python/pandas/data/cardata.csv"

data = pd.read\_csv(url)

print("Dataset head:")

print(data.head())

X = data[['Mileage', 'Age', 'Horsepower']]

y = data['Price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("\nModel Coefficients:")

print(f"Intercept: {model.intercept\_}")

print(f"Coefficients: {model.coef\_}")

print("\nModel Performance:")

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

new\_car = pd.DataFrame({'Mileage': [20000], 'Age': [3], 'Horsepower': [130]})

predicted\_price = model.predict(new\_car)

print(f"\nPredicted price for the new car: {predicted\_price[0]}")

**Slip 20**

**Q1. Write a R program to create a data frame from four given vectors.**

**[10 Marks]**

names <- c("Alice", "Bob", "Charlie", "Diana")

ages <- c(25, 30, 22, 28)

scores <- c(85.5, 90.0, 78.5, 88.0)

cities <- c("New York", "Los Angeles", "Chicago", "Houston")

students\_df <- data.frame(Name = names, Age = ages, Score = scores, City = cities)

print(students\_df)

**Q2. Write a python program to implement hierarchical Agglomerative clustering algorithm.**

**(Download Customer.csv dataset from github.com).**

**[20 Marks]**

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

url = "https://raw.githubusercontent.com/amankharwal/WebsiteData/master/Customer.csv"

data = pd.read\_csv(url)

print("First few rows of the dataset:")

print(data.head())

features = data.select\_dtypes(include=[np.number])

linked = linkage(features, method='ward')

plt.figure(figsize=(10, 7))

dendrogram(linked,

orientation='top',

distance\_sort='descending',

show\_leaf\_counts=True)

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('Samples')

plt.ylabel('Distance')

plt.show()

n\_clusters = 4 # Example: change as required

agglo = AgglomerativeClustering(n\_clusters=n\_clusters)

clusters = agglo.fit\_predict(features)

data['Cluster'] = clusters

print("\nData with cluster labels:")

print(data.head())