4.

import pandas as pd

# Define the file paths

csv\_file\_path = 'C:\\Users\\laxmi\\OneDrive\\Documents\\mlrevision\\prog1.csv'

excel\_file\_path ='https://d.docs.live.net//3c2f72303c0c453e//Documents//mlrevision//program4.xlsx'

# Load the CSV file

data\_csv = pd.read\_csv(csv\_file\_path)

print("CSV File Data:")

print(data\_csv)

# Load the Excel file

data\_excel = pd.read\_excel(excel\_file\_path)

print("\nExcel File Data:")

print(data\_excel)

# Basic Data Exploration

print("\nData Descriptions:")

print("CSV Data Description:")

print(data\_csv.describe())

print("\nExcel Data Description:")

print(data\_excel.describe())

# Displaying data types

print("\nData Types in CSV File:")

print(data\_csv.dtypes)

print("\nData Types in Excel File:")

print(data\_excel.dtypes)

5.

import pandas as pd

import matplotlib.pyplot as plt

# Load the data

data = pd.read\_csv('C:\\Users\\laxmi\\OneDrive\\Documents\\mlrevision\\program5.csv')

# Scatter plot of Study Hours vs. Exam Scores

plt.figure(figsize=(12, 7))

plt.subplot(1, 2, 1) # 1 row, 2 columns, 1st subplot

plt.scatter(data['Study Hours'], data['Exam Scores'], color='dodgerblue', edgecolor='k')

plt.title('Study Hours vs Exam Scores')

plt.xlabel('Study Hours')

plt.ylabel('Exam Scores')

plt.grid(True)

# Bar chart of Average Exam Score by Study Hour Range

bins = [0, 2, 4, 6, 8, 10, 12]

labels = ['0-2', '2-4', '4-6', '6-8', '8-10', '10-12']

data['Study Hour Range'] = pd.cut(data['Study Hours'], bins=bins, labels=labels)

grouped\_data = data.groupby('Study Hour Range')['Exam Scores'].mean()

plt.subplot(1, 2, 2)

grouped\_data.plot(kind='bar', color='salmon')

plt.title('Average Exam Score by Study Hour Range')

plt.xlabel('Study Hour Range')

plt.ylabel('Average Exam Score')

plt.xticks(rotation=0)

plt.tight\_layout() # Adjust subplots to fit in figure area

plt.show()

6.

import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Create dummy data

data = {

'Age': [25, 30, None, 28, 35],

'Gender': ['Female', 'Male', 'Female', 'Female', 'Male'],

'Income': [50000, 60000, 45000, None, 70000]

}

df = pd.DataFrame(data)

# Handling missing data

imputer = SimpleImputer(strategy='mean')

df[['Age', 'Income']] = imputer.fit\_transform(df[['Age', 'Income']])

# Print data after handling missing values

print("Data after handling missing values:")

print(df)

# Categorical encoding

encoder = OneHotEncoder()

encoded\_data = encoder.fit\_transform(df[['Gender']]).toarray()

# Create DataFrame of categorical encoded data

encoded\_df = pd.DataFrame(encoded\_data, columns=encoder.get\_feature\_names\_out(['Gender']))

# Merge encoded data with original dataframe

df = pd.concat([df.drop('Gender', axis=1), encoded\_df], axis=1)

# Print data after categorical encoding

print("\nData after categorical encoding:")

print(df)

# Feature scaling

scale = StandardScaler()

scaled\_data = scale.fit\_transform(df[['Age', 'Income']])

# Print data after feature scaling

scaled\_df = pd.DataFrame(scaled\_data, columns=['Scaled Age', 'Scaled Income'])

print("\nData after feature scaling:")

print(scaled\_df)

7.

import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

# Program 7: Write a program to implement a k-Nearest Neighbours (k-NN) classifier

# using scikit-learn and train the classifier on the dataset and evaluate its performance.

import numpy as np

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Dummy student data: exam score 1, exam score 2, pass/fail (features)

X = np.array([[100, 75], [95, 90], [60, 90], [45, 30], [80, 45], [85, 95], [70, 60], [90, 55], [60, 45], [80, 70]])

y = np.array([1, 1, 0, 0, 1, 1, 0, 1, 0, 1]) # Binary classes for demonstration

# 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 the k-NN classifier with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Train the classifier on the training data

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

# Evaluate the classifier's performance

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

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

print("Accuracy on the test set: {:.2f}".format(accuracy))

# Take user input for exam scores

exam\_score1 = float(input("Enter Exam Score 1: "))

exam\_score2 = float(input("Enter Exam Score 2: "))

# Prepare the user input for prediction

user\_input = np.array([[exam\_score1, exam\_score2]])

# Use the trained k-NN classifier to predict the outcome

predicted\_outcome = knn.predict(user\_input)

if predicted\_outcome[0] == 1:

print("Based on the exam scores provided, the student is predicted to pass.")

else:

print("Based on the exam scores provided, the student is predicted to fail.")

8.

import numpy as np

from sklearn.linear\_model import LinearRegression

X= np.array([[1000, 2], [1500, 3], [1200, 2], [1800, 4], [900, 2], [2000, 3]])

Y= np.array([300000, 400000, 350000, 500000, 280000, 450000])

model=LinearRegression()

model.fit(X, Y)

size = float(input("Enter the size of the house in sqft: "))

bedrooms= int(input("Enter the number of bedrooms: "))

new\_data = np.array([[size, bedrooms]])

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

print("Predicted price for a house with size {} sqft and {} bedrooms: Rs.{:.2f}".format(size, bedrooms, predicted\_price[0]))

9.

import numpy as np

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.tree import export\_text

import matplotlib.pyplot as plt

# Custom dummy data for fruit classification

# Features: [Weight, Texture] -> Target: [Fruit Type]

X = np.array([[150, 0], [170, 1], [120, 0], [140, 1], [200, 1], [130, 0]])

y = np.array(['Apple', 'Orange', 'Apple', 'Orange', 'Melon', 'Apple'])

# Initialize the Decision Tree Classifier

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X, y)

# Visualize the Decision Tree splits

tree\_rules = export\_text(clf, feature\_names=['Weight', 'Texture'])

print("Decision Tree Classifier Rules:\n", tree\_rules)

# Plot the Decision Tree

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

plot\_tree(clf, filled=True, feature\_names=['Weight', 'Texture'], class\_names=np.unique(y))

plt.show()

10.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

X=np.array([[30,50000],[35,60000],[40,80000],[25,30000],[45,100000],[20,20000],[50,120000],[55,150000],[60,140000],[28,40000]])

KMeans=KMeans(n\_clusters=3,random\_state=0)

KMeans.fit(X)

labels=KMeans.labels\_

centers=KMeans.cluster\_centers\_

plt.figure(figsize=(8,6))

plt.scatter(X[:,0],X[:,1],c=labels,cmap='viridis',s=50,alpha=0.8)

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

plt.xlabel('Age')

plt.ylabel('Income')

plt.title('K-Means Clustering of customers')

plt.legend()

plt.show()