**4th program : Write a program to Load and explore the dataset of .CSV and excel files using pandas.**

import pandas as pd

import pandas as pd

csv\_file\_path = r'C:\Users\91702\Documents\sample\_data.csv'

excel\_file\_path = r'C:\\ml\_projects\\sample\_data.xlsx'

data\_csv = pd.read\_csv( r'C:\Users\91702\Documents\sample\_data.csv')

print("CSV File Data:")

print(data\_csv)

data\_excel = pd.read\_excel(r'C:\\ml\_projects\\sample\_data.xlsx')

print("\nExcel File Data:")

print(data\_excel)

print("\nData Descriptions:")

print("CSV Data Description:")

print(data\_csv.describe())

print("\nExcel Data Description:")

print(data\_excel.describe())

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

print(data\_csv.dtypes)

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

print(data\_excel.dtypes)

| **ID** | **Name** | **Age** | **Score** |
| --- | --- | --- | --- |
| 1 | Alice | 23 | 85 |
| 2 | Bob | 25 | 78 |
| 3 | Charlie | 22 | 90 |
| 4 | David | 24 | 76 |
| 5 | Emma | 23 | 88 |

**5th program: Write a program to Visualize the dataset to gain insights using Matplotlib or Seaborn by plotting scatter plots, and bar charts.**

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read\_csv(r'C:\Users\91702\Documents\study\_data.csv')

plt.figure(figsize=(14, 7))

plt.subplot(1, 2, 1)

plt.scatter(data['Studyhours'], data['Examscores'],

color='dodgerblue', edgecolor='k', alpha=0.7)

plt.title('Studyhours vs. Examscores')

plt.xlabel('Studyhours')

plt.ylabel('Examscores')

plt.grid(True)

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

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

data['StudyhourRange'] = pd.cut(data['Studyhours'], bins=bins, labels=labels, right=False)

grouped\_data = data.groupby('StudyhourRange')['Examscores'].mean()

plt.subplot(1, 2, 2)

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

plt.title('Average Examscore by Studyhour Range')

plt.xlabel('Studyhour Range')

plt.ylabel('Average Examscore')

plt.xticks(rotation=0)

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

| **Studyhours** | **Examscores** |
| --- | --- |
| **1.5** | **45** |
| **3.0** | **51** |
| **4.5** | **60** |
| **6.0** | **68** |
| **7.5** | **75** |
| **9.0** | **85** |
| **10.5** | **88** |
| **2.0** | **50** |
| **5.5** | **65** |
| **8.0** | **80** |

**6th Program: Write a program to Handle missing data, encode categorical variables, and perform feature scaling.**

import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder, StandardScaler data = {

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

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

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

}

df = pd.DataFrame(data)

imputer = SimpleImputer(strategy='mean')

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

print(" Data after handling missing values:")

print(df)

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False)

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

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

print("\nData after categorical encoding:")

print(encoded\_df)

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

print("\n Full data after encoding:")

print(df\_encoded)

scaler = StandardScaler()

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

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

print("\n Data after feature scaling:")

print(scaled\_df)

**7th Program: 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

X = np.array([

[80, 75], [95, 90], [60, 50], [45, 30], [30, 40],

[85, 95], [70, 60], [50, 55], [40, 45], [60, 70]

])

y = np.array([1, 1, 0, 0, 0, 1, 1, 0, 0, 1])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=3)

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

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

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

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

try:

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

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

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

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

if predicted\_outcome[0] == 1:

print("The student is predicted to pass.")

else:

print("The student is predicted to fail.")

except ValueError:

print("Invalid input. Please enter numeric values.")

**8th Program: Write a program to implement a linear regression model for regression tasks and Train the model on a dataset with continuous target variables.**

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)

try:

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

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

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

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

print("Predicted price for a house with size {} sqft and {} rooms is Rs. {:.2f}".format(

size, rooms, predicted\_price[0]))

except ValueError:

print("Invalid input. Please enter numeric values.")

**9th Program : Write a program to implement a decision tree classifier using scikit-learn and visualize the decision tree and understand its splits.**

import numpy as np

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.tree import export\_text

import matplotlib.pyplot as plt

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

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

clf= DecisionTreeClassifier(random\_state=42)

clf.fit(X, y)

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

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

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

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

plt.show()

**10th Program: Write a program to Implement K-Means clustering and Visualize clusters.**

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()