

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**
“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT
on**

Machine Learning (23CS6PCMAL)

Submitted by

Adikar Charvi Sree Teja(1BM22CS012)

in partial fulfillment for the award of the degree of

**BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025**

**B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019**
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by **Adikar Charvi Sree Teja(1BM22CS012)**, who is bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Dr.Seema Patil Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE
---	--

Index

Sl. No.	Date	Experiment Title	Page No.
1	4-3-2025	Write a python program to import and export data using Pandas library functions	1-5
2	11-3-2025	Demonstrate various data pre-processing techniques for a given dataset	6-9
3	18-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	10-14
4	1-4-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	15-18
5	8-4-2025	Build Logistic Regression Model for a given dataset	19-23
6	15-4-2025	Build KNN Classification model for a given dataset.	24-30
7	15-4-2025	Build Support vector machine model for a given dataset	31-35
8	22-4-2025	Implement Random forest ensemble method on a given dataset.	36-39
9	22-4-2025	Implement Boosting ensemble method on a given dataset.	40-41
10	29-4-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	42-45
11	29-4-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	46-49

Github Link:

<https://github.com/charviadikar/1BM22CS012-ML-LAB.git>

Program 1

Write a python program to import and export data using Pandas library functions

Code:

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

```
Sample data:
      Name  Age        City
0     Alice   25  New York
1      Bob   30  Los Angeles
2  Charlie   35     Chicago
3   David   40    Houston
```

```
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
print("Sample data:")
print(df.head())
```

```
Sample data:
      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm) \
0                5.1            3.5            1.4            0.2
1                4.9            3.0            1.4            0.2
2                4.7            3.2            1.3            0.2
3                4.6            3.1            1.5            0.2
4                5.0            3.6            1.4            0.2

      target
0         0
1         0
2         0
3         0
4         0
```

```
from google.colab import files
uploaded = files.upload()
file_path = 'data.csv' # Ensure the file exists in the same directory
df = pd.read_csv(file_path)
```

```

print("Sample data:")
print(df.head())
print("\n")
Saving data.csv to data.csv
Sample data:
   ID      Name  Age        City
0   1      Alice  25  New York
1   2       Bob  30  Los Angeles
2   3  Charlie  35    Chicago
3   4    David  40   Houston
4   5      Eva  28   Phoenix

```

```

import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]
data = yf.download(tickers, start="2022-10-01", end="2023-10-01", group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()

```

```

First 5 rows of the dataset:
Ticker      RELIANCE.NS \
Price        Open       High      Low     Close    Volume
Date
2022-10-03  1092.199963 1103.822945 1079.184026 1082.152588 11852723
2022-10-04  1095.077201 1104.302526 1091.583396 1102.110352 8948850
2022-10-06  1109.326261 1118.916888 1104.371007 1106.174927 13352162
2022-10-07  1102.772687 1116.131217 1102.772687 1110.856323 7714340
2022-10-10  1098.365412 1104.119885 1090.601536 1098.730835 6329527

Ticker      INFY.NS \
Price        Open       High      Low     Close    Volume
Date
2022-10-03  1337.743240 1337.743240 1313.110574 1320.453003 4943169
2022-10-04  1345.038201 1356.928245 1339.638009 1354.228149 6631341
2022-10-06  1369.007786 1383.029504 1368.155094 1378.624023 6180672
2022-10-07  1370.286676 1381.181893 1364.412779 1374.881592 3994466
2022-10-10  1351.338576 1387.956005 1351.338576 1385.729614 5274677

Ticker      TCS.NS \
Price        Open       High      Low     Close    Volume
Date
2022-10-03  2799.044354 2823.062819 2779.418333 2789.651855 1763331
2022-10-04  2831.707297 2895.304988 2825.212065 2888.983076 2145875
2022-10-06  2907.454262 2919.603702 2890.117902 2898.996338 1790816
2022-10-07  2894.744206 2901.847047 2858.015696 2864.370605 1939879
2022-10-10  2813.062629 2922.407588 2808.389768 2914.510498 3064063

Shape of the dataset:
(247, 15)

Column names:
MultiIndex([['RELIANCE.NS', 'Open'],
            ('RELIANCE.NS', 'High'),
            ('RELIANCE.NS', 'Low'),
            ('RELIANCE.NS', 'Close'),
            ('RELIANCE.NS', 'Volume'),
            ('INFY.NS', 'Open'),
            ('INFY.NS', 'High'),
            ('INFY.NS', 'Low'),
            ('INFY.NS', 'Close'),
            ('INFY.NS', 'Volume'),
            ('TCS.NS', 'Open'),
            ('TCS.NS', 'High'),
            ('TCS.NS', 'Low'),
            ('TCS.NS', 'Close'),
            ('TCS.NS', 'Volume')],
           names=['Ticker', 'Price'])

Summary statistics for Reliance Industries:
Price      Open       High      Low     Close    Volume
count   247.000000 247.000000 247.000000 247.000000 2.470000e+02
mean   1151.456593 1160.153640 1141.068147 1150.426972 1.316652e+07
std    66.114624 67.077801 65.976400 66.914372 6.754099e+06
min    1011.592400 1013.875900 995.607838 1005.312744 3.370033e+06
25%    1102.624229 1107.157058 1088.489391 1101.094238 8.717141e+06
50%    1151.342807 1158.969749 1142.665512 1151.160156 1.158959e+07
75%    1200.335361 1207.724151 1189.020599 1199.134460 1.530302e+07
max    1292.463484 1304.337734 1277.392357 1297.875366 5.708188e+07

```

```

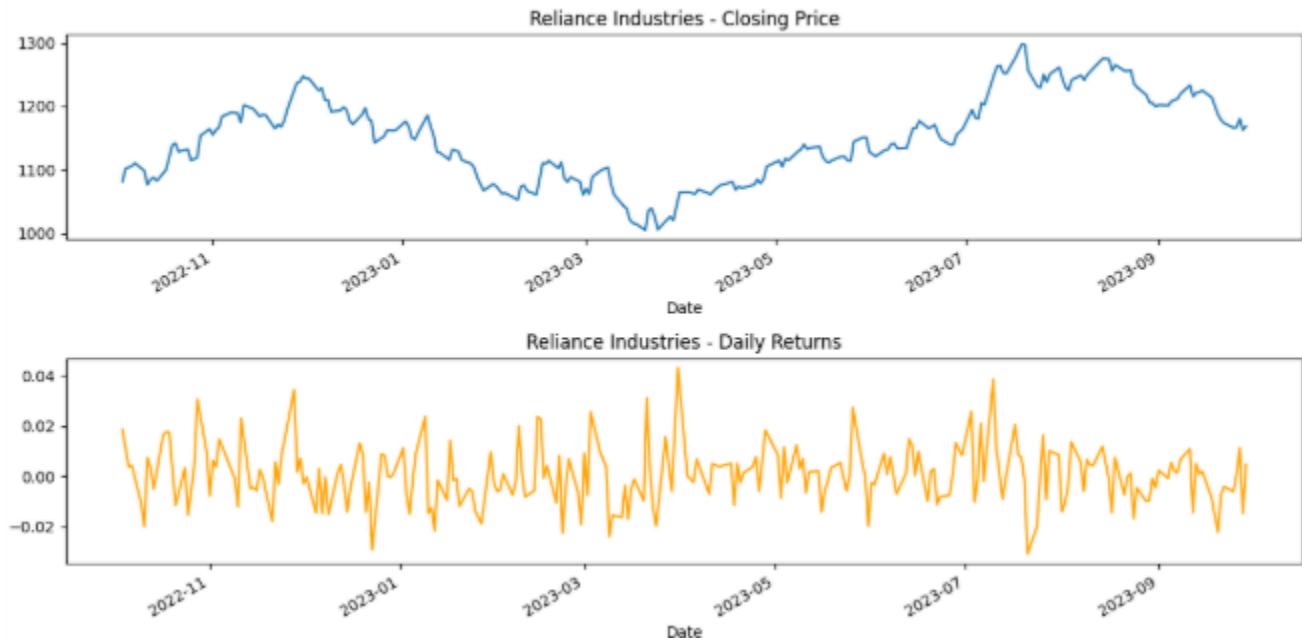
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="Reliance Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

```

```

reliance_data.to_csv('reliance_stock_data.csv')
print("\nReliance stock data saved to 'reliance_stock_data.csv'.")

```



```

import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
plt.figure(figsize=(14, 10))
for i, ticker in enumerate(tickers):
    bank_data = data[ticker]
    bank_data['Daily Return'] = bank_data['Close'].pct_change()
    plt.subplot(3, 2, 2*i+1)
    bank_data['Close'].plot(title=f'{ticker} - Closing Price', color='blue')
    plt.ylabel('Closing Price')
    plt.subplot(3, 2, 2*i+2)
    bank_data['Daily Return'].plot(title=f'{ticker} - Daily Returns', color='orange')
    plt.ylabel('Daily Return')
plt.tight_layout()
plt.show()
for ticker in tickers:
    bank_data = data[ticker]
    bank_data.to_csv(f'{ticker}_stock_data.csv')
    print(f"\n{ticker} stock data saved to '{ticker}_stock_data.csv'.")

```

First 5 rows of the dataset:

Ticker	KOTAKBANK.NS				
Price	Open	High	Low	Close	Volume
Date					
2024-01-01	1986.989954	1916.899886	1891.827338	1987.059814	1425982
2024-01-02	1985.911188	1985.911188	1858.063525	1883.088179	5128796
2024-01-03	1861.059234	1867.952665	1845.627158	1863.857178	3781515
2024-01-04	1860.451068	1869.451068	1858.513105	1861.559692	2865786
2024-01-05	1863.457575	1867.852782	1839.383985	1845.577148	7799341

Ticker	HDFCBANK.NS				
Price	Open	High	Low	Close	Volume
Date					
2024-01-01	1683.817508	1686.125187	1669.286199	1675.223999	7119843
2024-01-02	1675.014685	1679.868799	1665.958651	1676.218571	14621846
2024-01-03	1679.071488	1681.735059	1646.466666	1650.363525	14194881
2024-01-04	1655.394918	1672.116528	1648.193283	1668.071777	13367028
2024-01-05	1664.421596	1681.932477	1645.628188	1659.538288	15944735

Ticker	ICICIBANK.NS				
Price	Open	High	Low	Close	Volume
Date					
2024-01-01	983.086778	996.273246	982.541485	998.869812	7683792
2024-01-02	988.490253	989.134738	971.883221	973.886158	16263825
2024-01-03	976.295294	979.567116	966.777197	975.658818	16826752
2024-01-04	977.088757	988.787205	973.519178	978.724385	22789148
2024-01-05	979.567884	989.779158	975.482928	985.218445	14875409

Shape of the dataset:
(244, 15)

Column names:

```
MultiIndex([('KOTAKBANK.NS', 'Open'),
 ('KOTAKBANK.NS', 'High'),
 ('KOTAKBANK.NS', 'Low'),
 ('KOTAKBANK.NS', 'Close'),
 ('KOTAKBANK.NS', 'Volume'),
 ('HDFCBANK.NS', 'Open'),
 ('HDFCBANK.NS', 'High'),
 ('HDFCBANK.NS', 'Low'),
 ('HDFCBANK.NS', 'Close'),
 ('HDFCBANK.NS', 'Volume'),
 ('ICICIBANK.NS', 'Open'),
 ('ICICIBANK.NS', 'High'),
 ('ICICIBANK.NS', 'Low'),
 ('ICICIBANK.NS', 'Close'),
 ('ICICIBANK.NS', 'Volume')], names=['Ticker', 'Price'])
```

The figure consists of four subplots arranged in a 2x2 grid. The top row shows the 'HDFCBANK.NS - Closing Price' and 'HDFCBANK.NS - Daily Returns'. The bottom row shows the 'KOTAKBANK.NS - Closing Price' and 'KOTAKBANK.NS - Daily Returns'. Each plot has 'Date' on the x-axis and either 'Closing Price' or 'Daily Returns' on the y-axis. The closing price plots show a general upward trend over the period, while the daily return plots show high volatility with many sharp fluctuations.

HDFCBANK.NS stock data saved to 'HDFCBANK.NS_stock_data.csv'.

ICICIBANK.NS stock data saved to 'ICICIBANK.NS_stock_data.csv'.

KOTAKBANK.NS stock data saved to 'KOTAKBANK.NS_stock_data.csv'.

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:

min_max = MinMaxScaler()
 $\text{df} = \text{min_max}.fit_\text{transform}(\text{df}[\text{numerical columns}])$, columns = df[numerical columns]

standard_scler = StandardScaler()
 $\text{df} = \text{standard}_\text{scler}.fit_\text{transform}(\text{df}[\text{numerical columns}])$, columns = df[numerical columns]

Q1: Which columns in the dataset had missing values? How to handle them?
 Ans: Missing values are present in numerical columns if present, which are replaced by the mean of the respective column.

Q2: Which categorical column did you identify? How did you handle them?
 Ans: No categorical column, hence no encoding.

Q3: What are the differences between scaling & standardization? When would you use one over the other?
 Ans: Min max Scaling transforms data to a fixed range [0,1] using $y' = \frac{x - \text{min}}{\text{max} - \text{min}}$

used when :- dataset doesn't follow a normal distribution
 - features have different ranges & need to be scaled

Standardization transforms data to have zero mean & unit variance $y' = \frac{x - \mu}{\sigma}$

used when:- dataset follows a Gaussian distribution
 - many ML algos assume normality.

iii) Adult-income.csv
 $\text{import pandas as pd}$
 $\text{import numpy as np}$
 $\text{from sklearn import input_impl_ingress}$
 $\text{from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder}$
 $\text{file_path = "adult.csv"}$

df = pd.read_csv(file_path)
 $\text{df.replace('?', np.nan).info()}$
 num_imputer = SimpleImputer(strategy = "mean")
 $\text{df['age'].replace(np.nan, num_imputer.fit_transform(df[['age']]))}$, columns = df[['age']]
 cat_imputer = SimpleImputer(strategy = "most_frequent")
 $\text{label_encoder} = \text{LabelEncoder()}$
 $\text{for col in df.select_dtypes(include=['object'])}: \text{columns}:$
 $\text{df[col].replace(np.nan, label_encoder.fit_transform(df[col]))}$, columns = df[[col]]
 $\text{Q1} = \text{df.quantile(0.25)}$
 $\text{Q3} = \text{df.quantile(0.75)}$
 $\text{IQR} = \text{Q3} - \text{Q1}$
 $\text{df} = (\text{df} < (\text{Q1} - 1.5 * \text{IQR}) | \text{df} > (\text{Q3} + 1.5 * \text{IQR})) \text{ay}(\text{and} _z)$
 $\text{num_max} = \text{MinMaxScaler}()$
 $\text{df} = \text{num_max}.fit_\text{transform}(\text{df}), \text{columns} = \text{df.columns}$
 $\text{std_scler} = \text{StandardScaler}()$
 $\text{df} = \text{std_scler}.fit_\text{transform}(\text{df}), \text{columns} = \text{df.columns}$

Q4: Which columns in the dataset had missing values? How to handle them?
 Ans: Missing values are represented by "?" which we replace with NaN

for numerical columns - replace with mean
 for categorical column - replace with mode

Q5: Which categorical column did you identify? How did you encode them?
 Ans: Work class, education, marital status, occupation, relationship, native_x
 encoding method: Label Encoding to convert categorical to numerical format.

Code:

```
import pandas as pd
file_path = 'housing.csv'
df = pd.read_csv(file_path)
print("\nDataset Information:")
print(df.info())
print("\nStatistical Information of Numerical Columns:")
print(df.describe())
print("\nUnique Labels Count for 'Ocean Proximity' column:")
print(df['ocean_proximity'].value_counts())
print("\nAttributes with Missing Values:")
missing_values = df.isnull().sum()
columns_with_missing_values = missing_values[missing_values > 0]
print(columns_with_missing_values)
```

```

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   longitude        20640 non-null   float64
 1   latitude         20640 non-null   float64
 2   housing_median_age 20640 non-null   float64
 3   total_rooms      20640 non-null   float64
 4   total_bedrooms   20433 non-null   float64
 5   population       20640 non-null   float64
 6   households       20640 non-null   float64
 7   median_income    20640 non-null   float64
 8   median_house_value 20640 non-null   float64
 9   ocean_proximity  20640 non-null   object  
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
None

Statistical Information of Numerical Columns:
          longitude      latitude  housing_median_age  total_rooms \
count  20640.000000  20640.000000     20640.000000  20640.000000
mean   -119.569704   35.631861     28.639486   2635.763081
std     2.003532    2.135952    12.585558   2181.615252
min    -124.350000   32.540000    1.000000    2.000000
25%   -121.800000   33.930000    18.000000   1447.750000
50%   -118.490000   34.260000    29.000000   2127.000000
75%   -118.010000   37.710000    37.000000   3148.000000
max    -114.310000   41.950000    52.000000   39328.000000

          total_bedrooms  population  households  median_income \
count  20433.000000  20640.000000  20640.000000  20640.000000
mean   537.870553  1425.476744  499.539688   3.870571
std    421.385070  1132.462122  382.329753   1.898922
min    1.000000    3.000000    1.000000   0.499900
25%   296.000000   787.000000   280.000000   2.563400
50%   435.000000  1166.000000  489.000000   3.534800
75%   647.000000  1725.000000  685.000000   4.743250
max    6445.000000  35682.000000  6882.000000   15.000100

          median_house_value
count  20640.000000
mean   266855.816099
std    115395.615874
min    14999.000000
25%   119600.000000
50%   179700.000000
75%   264725.000000
max    580001.000000

Unique Labels Count for 'Ocean Proximity' column:
ocean_proximity
<1H OCEAN      9136
INLAND          6551
NEAR OCEAN      2658
NEAR BAY         2290
ISLAND           5
Name: count, dtype: int64

Attributes with Missing Values:
total_bedrooms    287
dtype: int64

import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler
file_path = "diabetes.csv"
df = pd.read_csv(file_path)
df_numeric = df.select_dtypes(include=['number']).copy() # Select only numeric columns
imputer = SimpleImputer(strategy="mean")
df_numeric.iloc[:, :] = imputer.fit_transform(df_numeric)
df[df_numeric.columns] = df_numeric
Q1 = df_numeric.quantile(0.25) # Only compute quartiles on numeric data
Q3 = df_numeric.quantile(0.75) # Only compute quartiles on numeric data
IQR = Q3 - Q1
df = df[~((df_numeric < (Q1 - 1.5 * IQR)) | (df_numeric > (Q3 + 1.5 * IQR))).any(axis=1)]
min_max_scaler = MinMaxScaler()
df_minmax = pd.DataFrame(min_max_scaler.fit_transform(df_numeric),
columns=df_numeric.columns) # Only transform the numeric columns
standard_scaler = StandardScaler()
df_standard = pd.DataFrame(standard_scaler.fit_transform(df_numeric), columns=df_numeric.columns)
# Only transform the numeric columns
print("\nProcessed Diabetes Dataset (Min-Max Scaled):")
print(df_minmax.head())

```

```

print("\nProcessed Diabetes Dataset (Standard Scaled):")
print(df_standard.head())



---


Processed Diabetes Dataset (Min-Max Scaled):
   ID No_Pation AGE Urea Cr HbA1c Chol \
0  0.627034  0.000237  0.508475  0.109375  0.050378  0.264901  0.407767
1  0.918648  0.000452  0.191695  0.184167  0.070529  0.264901  0.359223
2  0.524406  0.000634  0.508475  0.109375  0.050378  0.264901  0.407767
3  0.849812  0.001160  0.508475  0.109375  0.050378  0.264901  0.407767
4  0.629537  0.000452  0.220339  0.171875  0.050378  0.264901  0.475728

      TG HDL LDL VLDL BMI
0  0.844444  0.226804  0.114583  0.011461  0.173913
1  0.081481  0.092784  0.187500  0.014327  0.139130
2  0.844444  0.226804  0.114583  0.011461  0.173913
3  0.844444  0.226804  0.114583  0.011461  0.173913
4  0.051852  0.061856  0.177083  0.008596  0.069565

Processed Diabetes Dataset (Standard Scaled):
   ID No_Pation AGE Urea Cr HbA1c Chol \
0  0.672148 -0.074747 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
1  1.641852 -0.069940 -3.130017 -0.212954 -0.115804 -1.334983 -0.893730
2  0.338868 -0.065869 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
3  1.412950 -0.054126 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
4  0.680463 -0.069939 -2.334096  0.673299 -0.382672 -1.334983  0.028576

      TG HDL LDL VLDL BMI
0 -1.035084  1.810756 -1.085457 -0.369958 -1.124622
1 -0.678063 -0.158692 -0.457398 -0.342649 -1.326239
2 -1.035084  1.810756 -1.085457 -0.369958 -1.124622
3 -1.035084  1.810756 -1.085457 -0.369958 -1.124622
4 -0.963680 -0.613180 -0.547121 -0.397267 -1.729472

```

```

import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder
file_path = "adult.csv" # Update the path if needed
df = pd.read_csv(file_path)
df.replace("?", np.nan, inplace=True)
num_imputer = SimpleImputer(strategy="mean")
df[df.select_dtypes(include=['number']).columns] =
num_imputer.fit_transform(df.select_dtypes(include=['number']))
cat_imputer = SimpleImputer(strategy="most_frequent")
df[df.select_dtypes(include=['object']).columns] =
cat_imputer.fit_transform(df.select_dtypes(include=['object']))
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

```

```

df = df[((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
min_max_scaler = MinMaxScaler()
df_minmax = pd.DataFrame(min_max_scaler.fit_transform(df), columns=df.columns)
standard_scaler = StandardScaler()
df_standard = pd.DataFrame(standard_scaler.fit_transform(df), columns=df.columns)
print("\nProcessed Adult Income Dataset (Min-Max Scaled):")
print(df_minmax.head())
print("\nProcessed Adult Income Dataset (Standard Scaled):")
print(df_standard.head())

```

Processed Adult Income Dataset (Min-Max Scaled):

	age	workclass	fnlwgt	education	educational-num	marital-status
0	0.344262	0.0	0.188277	0.555556	0.363636	0.333333
1	0.114754	0.0	0.881156	1.000000	0.454545	0.666667
2	0.147541	0.0	0.169156	0.555556	0.363636	0.666667
3	0.672131	0.0	0.708251	0.555556	0.363636	0.333333
4	0.131148	0.0	0.475807	0.333333	0.727273	0.333333

	occupation	relationship	race	gender	capital-gain	capital-loss
0	0.307692	0.0	0.0	1.0	0.0	0.0
1	0.538462	0.8	0.0	0.0	0.0	0.0
2	0.000000	0.2	0.0	0.0	0.0	0.0
3	0.692308	0.0	0.0	1.0	0.0	0.0
4	0.692308	0.0	0.0	1.0	0.0	0.0

	hours-per-week	native-country	income
0	0.894737	0.0	0.0
1	0.368421	0.0	0.0
2	0.315789	0.0	0.0
3	0.105263	0.0	0.0
4	0.368421	0.0	0.0

Processed Adult Income Dataset (Standard Scaled):

	age	workclass	fnlwgt	education	educational-num	marital-status
0	0.220179	0.0	-0.022983	-0.151256	-0.654083	-0.398228
1	-0.955630	0.0	2.234629	1.457372	-0.073261	0.828047
2	-0.787657	0.0	-1.112882	-0.151256	-0.654083	0.828047
3	1.899906	0.0	1.421707	-0.151256	-0.654083	-0.398228
4	-0.871644	0.0	0.328856	-0.955571	1.669207	-0.398228

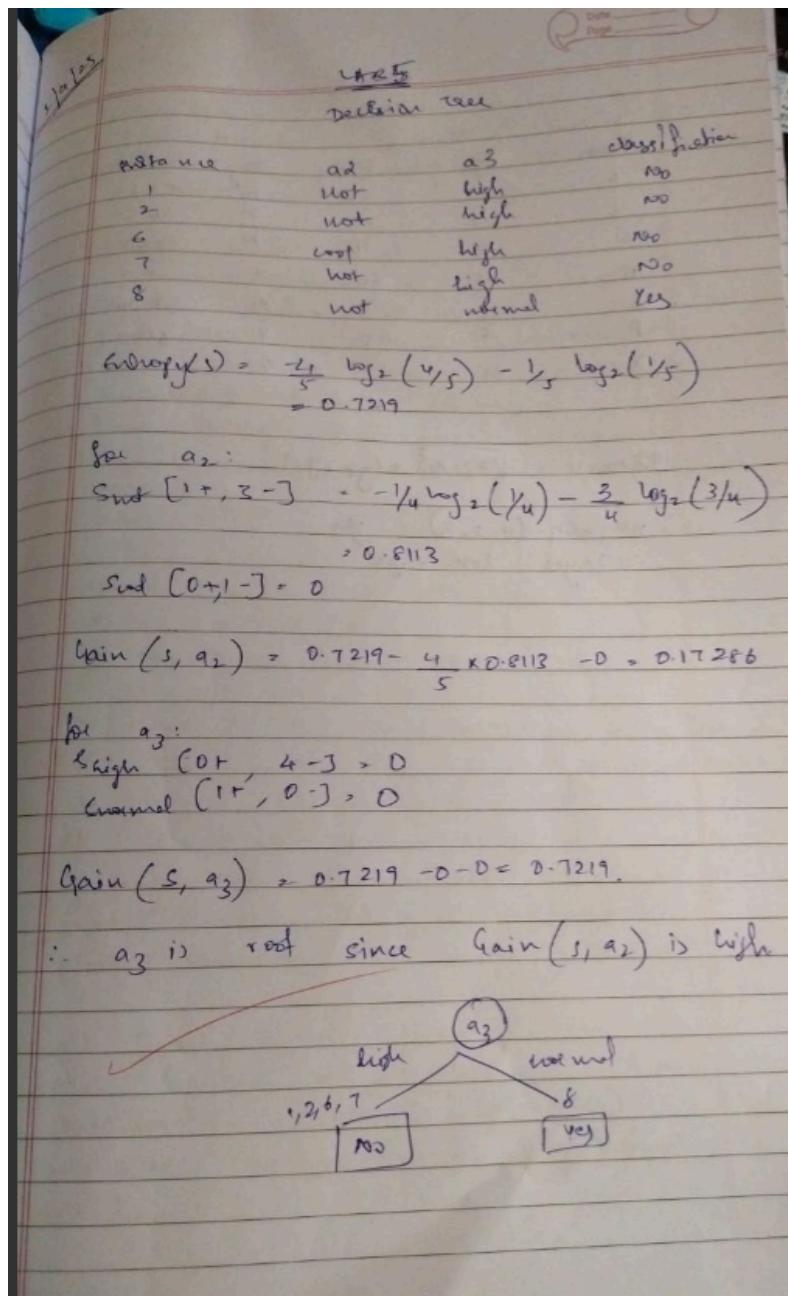
	occupation	relationship	race	gender	capital-gain	capital-loss
0	-0.428679	-1.044582	0.0	0.770972	0.0	0.0
1	0.305840	1.629927	0.0	-1.297064	0.0	0.0
2	-1.389371	-0.375955	0.0	-1.297064	0.0	0.0
3	0.790186	-1.044582	0.0	0.770972	0.0	0.0
4	0.790186	-1.044582	0.0	0.770972	0.0	0.0

	hours-per-week	native-country	income
0	2.312838	0.0	0.0
1	-0.329781	0.0	0.0
2	-0.594043	0.0	0.0
3	-1.651090	0.0	0.0
4	-0.329781	0.0	0.0

Program 3

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Screenshot:



Code:

```

Iris.csv
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv('iris.csv')
le = LabelEncoder()
df['species'] = le.fit_transform(df['species'])
X = df.drop('species', axis=1)
y = df['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=le.classes_))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
            yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns)
plt.show()

```

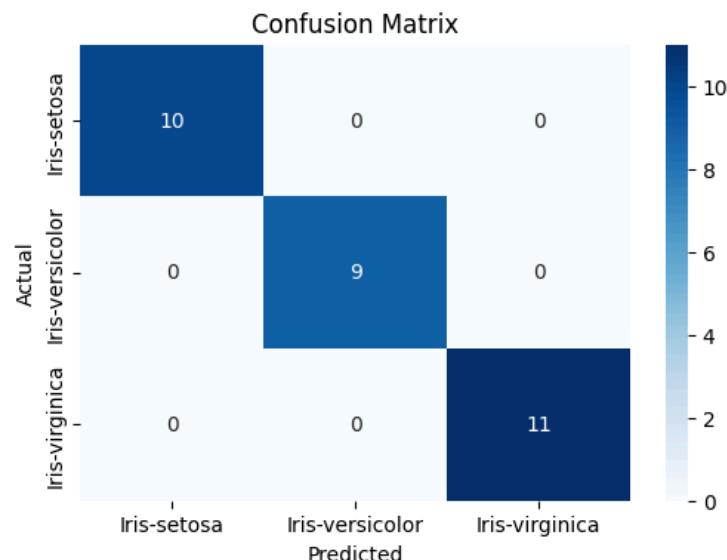
```

Accuracy: 1.00
      precision    recall  f1-score   support

Iris-setosa      1.00      1.00      1.00      10
Iris-versicolor  1.00      1.00      1.00       9
Iris-virginica   1.00      1.00      1.00      11

   accuracy          1.00
macro avg       1.00      1.00      1.00      30
weighted avg    1.00      1.00      1.00      30

```



Drug.csv

```

import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
df = pd.read_csv('drug.csv')
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
X = df.drop('Drug', axis=1)
y = df['Drug']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

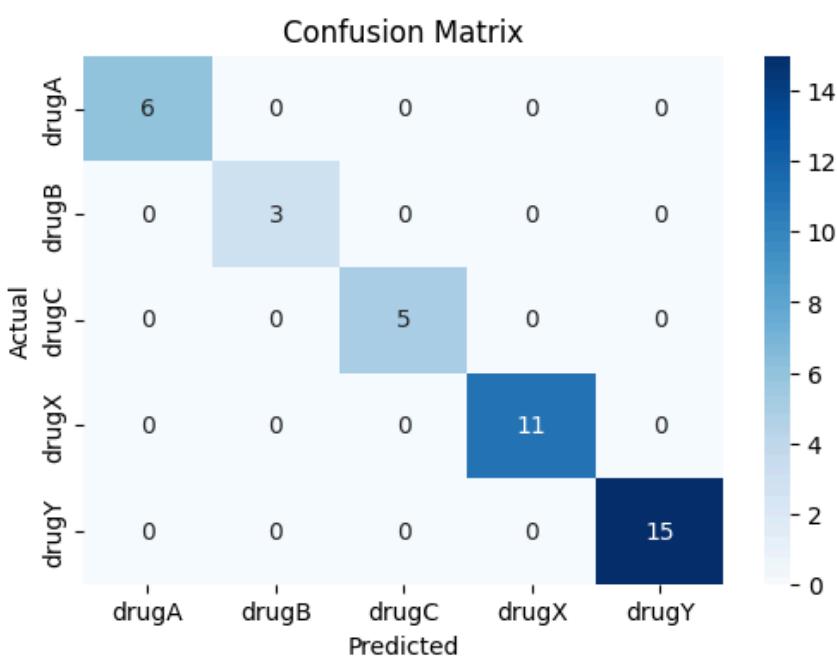
```

```

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
            yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns)
plt.show()

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6
1	1.00	1.00	1.00	3
2	1.00	1.00	1.00	5
3	1.00	1.00	1.00	11
4	1.00	1.00	1.00	15
accuracy			1.00	40
macro avg	1.00	1.00	1.00	40
weighted avg	1.00	1.00	1.00	40



```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
import math
data = pd.read_csv('petrol_consumption.csv')
X = data.drop('Petrol_Consumption', axis=1)
y = data['Petrol_Consumption']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
regressor = DecisionTreeRegressor(random_state=42, max_depth=3) # Limiting depth for readability
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

print("Regression Tree Evaluation Metrics:")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
plt.figure(figsize=(20,6))
plot_tree(
    regressor,
    feature_names=X.columns,
    filled=True,
    rounded=True,
    fontsize=10,
)
plt.title("Regression Tree for Petrol Consumption Prediction", fontsize=14)
plt.show()

```

```

Regression Tree Evaluation Metrics:
Mean Absolute Error (MAE): 80.63
Mean Squared Error (MSE): 14718.40
Root Mean Squared Error (RMSE): 121.32

```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset.

Screenshot:

1/2/25
AB 4

Linear Regression
Solve using matrix approach

week y
 1 2
 2 4
 3 5
 4 9

1. $x^T x = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$

2. Compute $(x^T x)^{-1} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$

3. $((x^T x)^{-1} x^T) = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$
 $= \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix}$

4. $(y^T x)^T x^T y = \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix} \begin{pmatrix} 2 \\ 4 \\ 5 \\ 9 \end{pmatrix}$

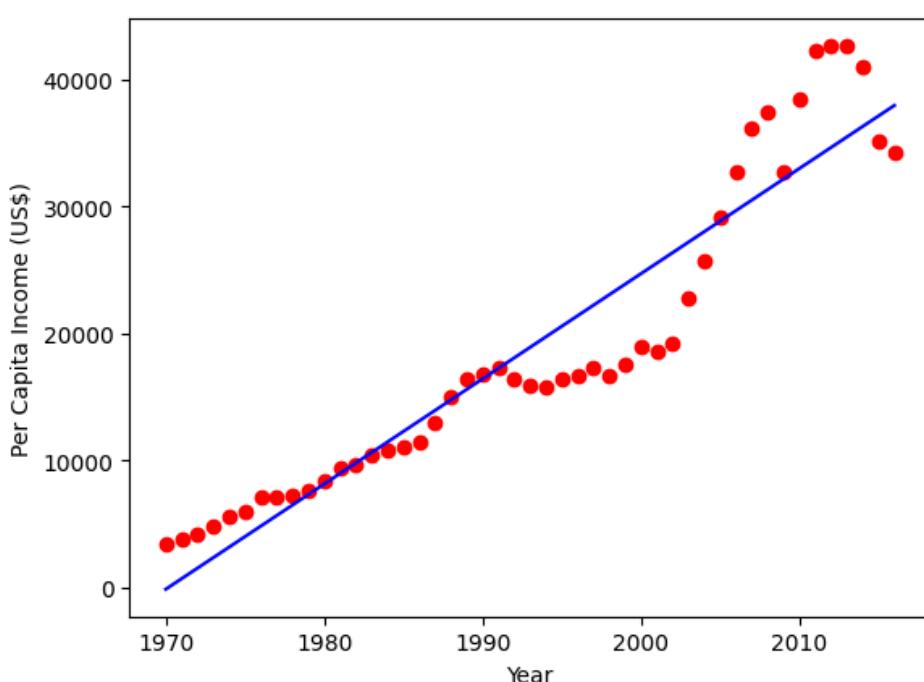
$\beta = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$ Pass intercept slope

$\beta_0 = -0.5$ $\beta_1 = 2.2$

$y_5 = \beta_0 + \beta_1 x_5 = -0.5 + (2.2) 5 = 10.5$

Linear Regression:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
df = pd.read_csv('canada_per_capita_income.csv')
reg = linear_model.LinearRegression()
reg.fit(df[['year']], df['per capita income (US$)'])
print(f"Coefficient: {reg.coef_}")
print(f"Intercept: {reg.intercept_}")
predicted_income = reg.predict([[2020]])
print(f"Predicted per capita income for 2020: ${predicted_income[0]:,.2f}")
plt.scatter(df['year'], df['per capita income (US$)', color='red']
plt.plot(df['year'], reg.predict(df[['year']]), color='blue')
plt.xlabel('Year')
plt.ylabel('Per Capita Income (US$)')
plt.show()
print(results.head())
Coefficient: [828.46507522]
Intercept: -1632210.7578554575
Predicted per capita income for 2020: $41,288.69
```



Multiple Regression:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
df_hiring = pd.read_csv('hiring.csv')
```

```

print("Original Data:")
print(df_hiring.head())
df_hiring['experience'] = df_hiring['experience'].replace({
    'five': 5,
    'four': 4,
    'three': 3
})
df_hiring['experience'] = pd.to_numeric(df_hiring['experience'], errors='coerce')
df_hiring['test_score(out of 10)'] = pd.to_numeric(df_hiring['test_score(out of 10)'], errors='coerce')
df_hiring['interview_score(out of 10)'] = pd.to_numeric(df_hiring['interview_score(out of 10)'],
errors='coerce')
df_hiring['salary($)'] = pd.to_numeric(df_hiring['salary($)'), errors='coerce')
df_hiring['experience'] = df_hiring['experience'].fillna(df_hiring['experience'].median())
df_hiring['test_score(out of 10)'] = df_hiring['test_score(out of 10)'].fillna(df_hiring['test_score(out of 10)').median())
df_hiring['interview_score(out of 10)'] = df_hiring['interview_score(out of 10)'].fillna(df_hiring['interview_score(out of 10)').median())
df_hiring['salary($)'] = df_hiring['salary($)'].fillna(df_hiring['salary($)'].median())
X_hiring = df_hiring[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y_hiring = df_hiring['salary($)']
model_hiring = LinearRegression()
model_hiring.fit(X_hiring, y_hiring)
print(f"\nCoefficients: {model_hiring.coef_}")
print(f"Intercept: {model_hiring.intercept_}")
predictions_hiring = model_hiring.predict([[2, 9, 6], [12, 10, 10]])
print(f"\nPredicted salary for candidate 1 (2 years experience, 9 test score, 6 interview score): ${predictions_hiring[0]:,.2f}")
print(f"Predicted salary for candidate 2 (12 years experience, 10 test score, 10 interview score): ${predictions_hiring[1]:,.2f}")

Original Data:
   experience  test_score(out of 10)  interview_score(out of 10)  salary($)
0        NaN             8.0                  9            50000
1        NaN             8.0                  6            45000
2       five             6.0                  7            60000
3       two             10.0                 10            65000
4      seven             9.0                  6            70000

Coefficients: [-793.62416107   -5.03355705   139.26174497]
Intercept: 65117.44966442955

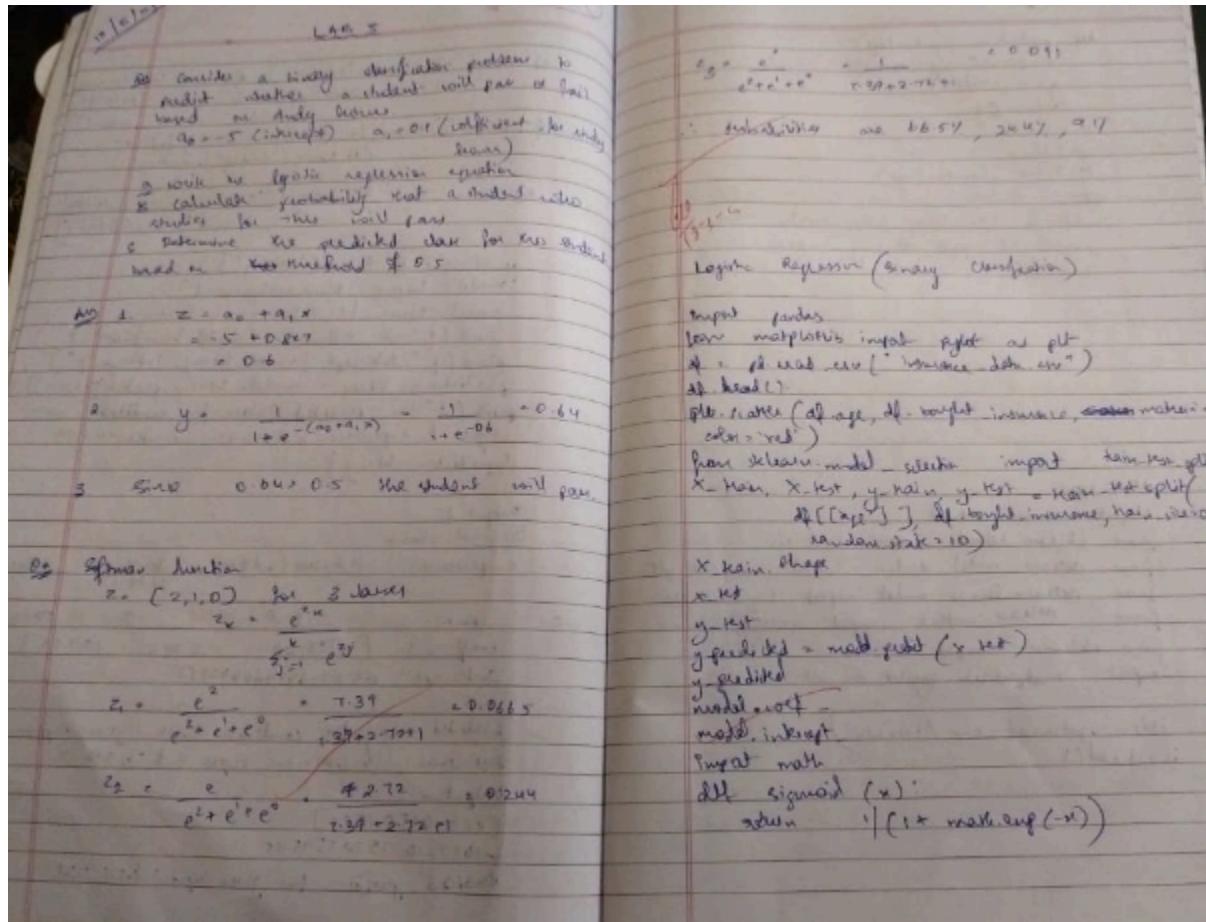
Predicted salary for candidate 1 (2 years experience, 9 test score, 6 interview score): $64,320.47
Predicted salary for candidate 2 (12 years experience, 10 test score, 10 interview score): $56,936.24

```

Program 5

Build Logistic Regression Model for a given dataset.

Screenshot:



Binary Logical Regression

```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
hr_data = pd.read_csv("HR_comma_sep.csv")
print(hr_data.head())
print("\nMissing values:\n", hr_data.isnull().sum())
print("\nData Types:\n", hr_data.dtypes)
print("\nUnique values in categorical columns:\n",
hr_data.select_dtypes(include=['object']).nunique())
    
```

```

plt.figure(figsize=(8,5))
sns.countplot(x="salary", hue="left", data=hr_data)
plt.title("Impact of Salary on Employee Retention")
plt.xlabel("Salary Level")
plt.ylabel("Number of Employees")
plt.legend(["Stayed", "Left"])
plt.show()

```

```

satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                  0.38           0.53            2                 157
1                  0.80           0.86            5                 262
2                  0.11           0.88            7                 272
3                  0.72           0.87            5                 223
4                  0.37           0.52            2                 159

time_spend_company  Work_accident  left  promotion_last_5years Department  \
0                  3              0     1                      0      sales
1                  6              0     1                      0      sales
2                  4              0     1                      0      sales
3                  5              0     1                      0      sales
4                  3              0     1                      0      sales

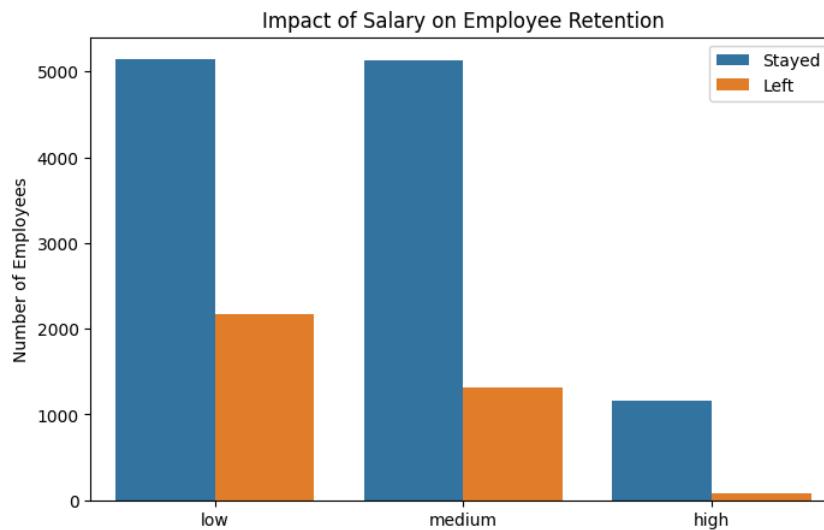
salary
0   low
1  medium
2  medium
3   low
4   low

Missing values:
satisfaction_level      0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                      0
promotion_last_5years    0
Department                0
salary                     0
dtype: int64

Data Types:
satisfaction_level      float64
last_evaluation          float64
number_project           int64
average_montly_hours     int64
time_spend_company       int64
Work_accident            int64
left                      int64
promotion_last_5years    int64
Department                object
salary                     object
dtype: object

```

```
Unique values in categorical columns:
Department      10
salary           3
dtype: int64
```



Multi Linear classification

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
zoo_data = pd.read_csv("zoo-data.csv")
class_types = pd.read_csv("zoo-class-type.csv")
print("Zoo Data:\n", zoo_data.head())
print("\nClass Type Data:\n", class_types.head())
if "class_type" not in zoo_data.columns:
    print("\nError: 'class_type' column is missing in zoo-data.csv. Available columns: ", zoo_data.columns)
else:
    X = zoo_data.drop(columns=["animal_name", "class_type"], errors="ignore")
    y = zoo_data["class_type"]
    print("\nUnique class types:", y.unique())
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    model = LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=200)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\nModel Accuracy: {accuracy:.4f}")
```

```

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=y.unique(), yticklabels=y.unique())
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.title("Confusion Matrix - Zoo Dataset")
plt.show()
print("\nClassification Report:\n", classification_report(y_test, y_pred))

Zoo Data:
   animal_name  hair  feathers  eggs  milk  airborne  aquatic  predator \
0      aardvark     1        0     0     1        0         0        1
1      antelope     1        0     0     1        0         0        0
2        bass       0        0     1     0        0         1        1
3       bear       1        0     0     1        0         0        1
4       boar       1        0     0     1        0         0        1

   toothed  backbone  breathes  venomous  fins  legs  tail  domestic  catsize \
0        1         1        1        0        0       4       0        0        1
1        1         1        1        0        0       4       1        0        1
2        1         1        0        0        1       0       1        0        0
3        1         1        1        0        0       4       0        0        1
4        1         1        1        0        0       4       1        0        1

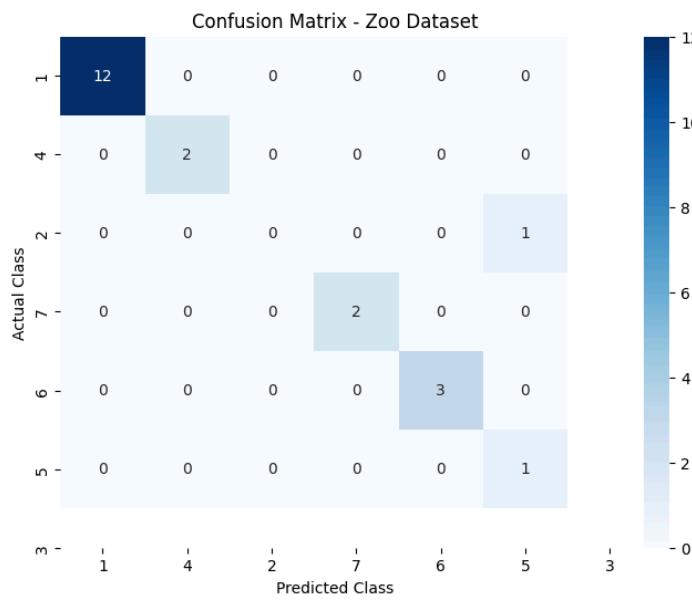
   class_type
0            1
1            1
2            4
3            1
4            1

Class Type Data:
   Class_Number  Number_Of_Animal_Species_In_Class Class_Type \
0            1                           41      Mammal
1            2                           20       Bird
2            3                           5    Reptile
3            4                          13      Fish
4            5                           4 Amphibian

   Animal_Names
0  aardvark, antelope, bear, boar, buffalo, calf, ...
1  chicken, crow, dove, duck, flamingo, gull, haw...
2  pitviper, seasnake, slowworm, tortoise, tuatara
3  bass, carp, catfish, chub, dogfish, haddock, h...
4  frog, frog, newt, toad

Unique class types: [1 4 2 7 6 5 3]
Model Accuracy: 0.9524

```



Classification Report:

	precision	recall	f1-score	support
1	1.00	1.00	1.00	12
2	1.00	1.00	1.00	2
3	0.00	0.00	0.00	1
4	1.00	1.00	1.00	2
5	1.00	1.00	1.00	3
6	0.50	1.00	0.67	1
7	0.50	1.00	0.67	1
accuracy		0.95		21
macro avg	0.75	0.83	0.78	21
weighted avg	0.93	0.95	0.94	21

Program 6

Build KNN Classification model for a given dataset.

Screenshot:

LAB 6						
KNN						
Person	Age	Salary	Target	Distance	Rank	
A	18	40	N	52.8		
B	23	55	N	46.5		
C	24	70	N	31.9		
D	41	60	Y	40.4		
E	43	70	Y	31.04		
F	38	40	Y	60.07		
X	35	100	?			

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

majority (1, 2, 3) : yes
∴ target don't = yes

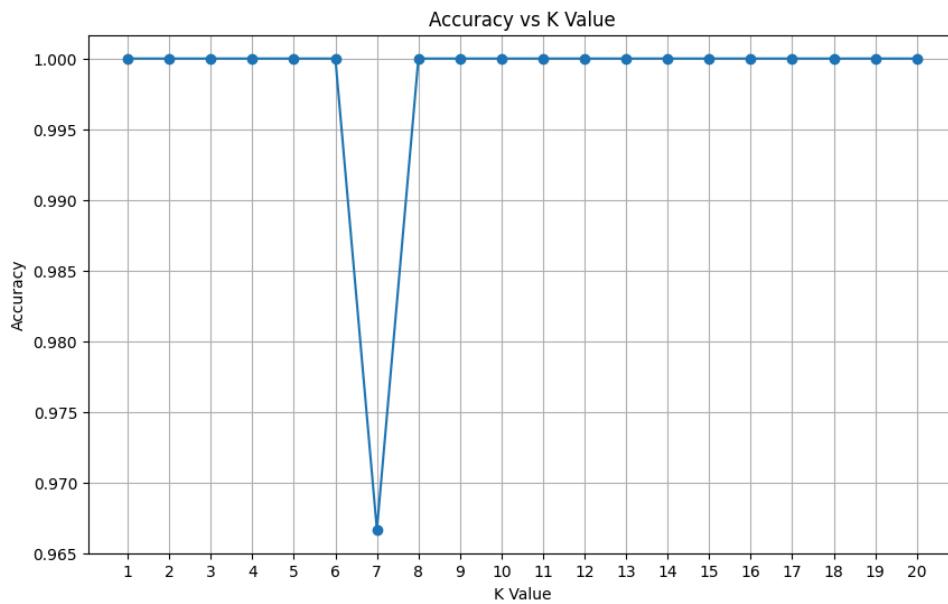
Iris.csv

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (accuracy_score, confusion_matrix, classification_report,
ConfusionMatrixDisplay)
from sklearn.preprocessing import LabelEncoder
data = pd.read_csv('iris.csv')
```

```

X = data.drop('species', axis=1)
y = data['species']
le = LabelEncoder()
y = le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
accuracy = []
for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy.append(accuracy_score(y_test, y_pred))
plt.figure(figsize=(10, 6))
plt.plot(range(1, 21), accuracy, marker='o')
plt.title('Accuracy vs K Value')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.xticks(range(1, 21))
plt.grid()
plt.show()
optimal_k = np.argmax(accuracy) + 1 # +1 because range starts at 1
print(f"\nOptimal K value: {optimal_k}")
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy:.4f}")
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=le.classes_)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=le.classes_))

```

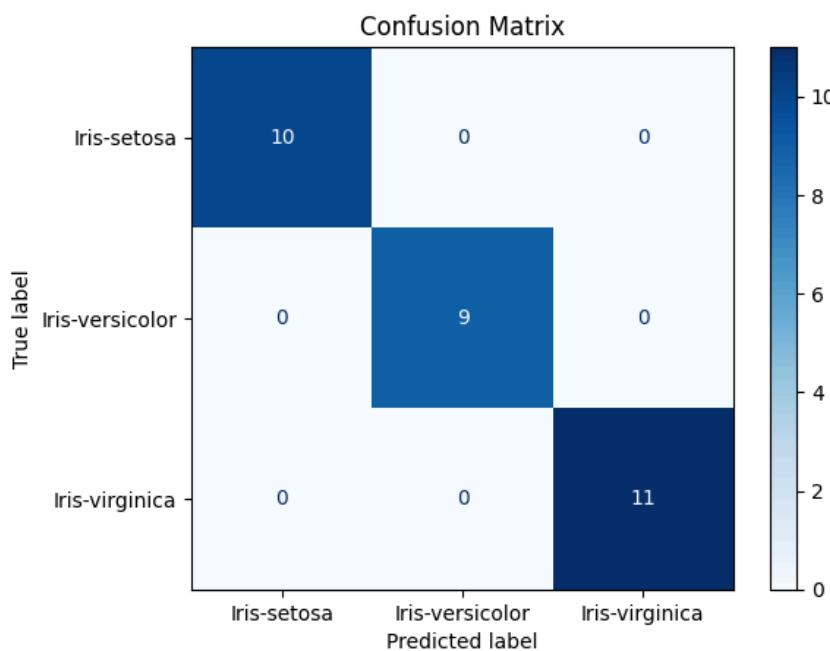


Optimal K value: 1

Accuracy: 1.0000

Confusion Matrix:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

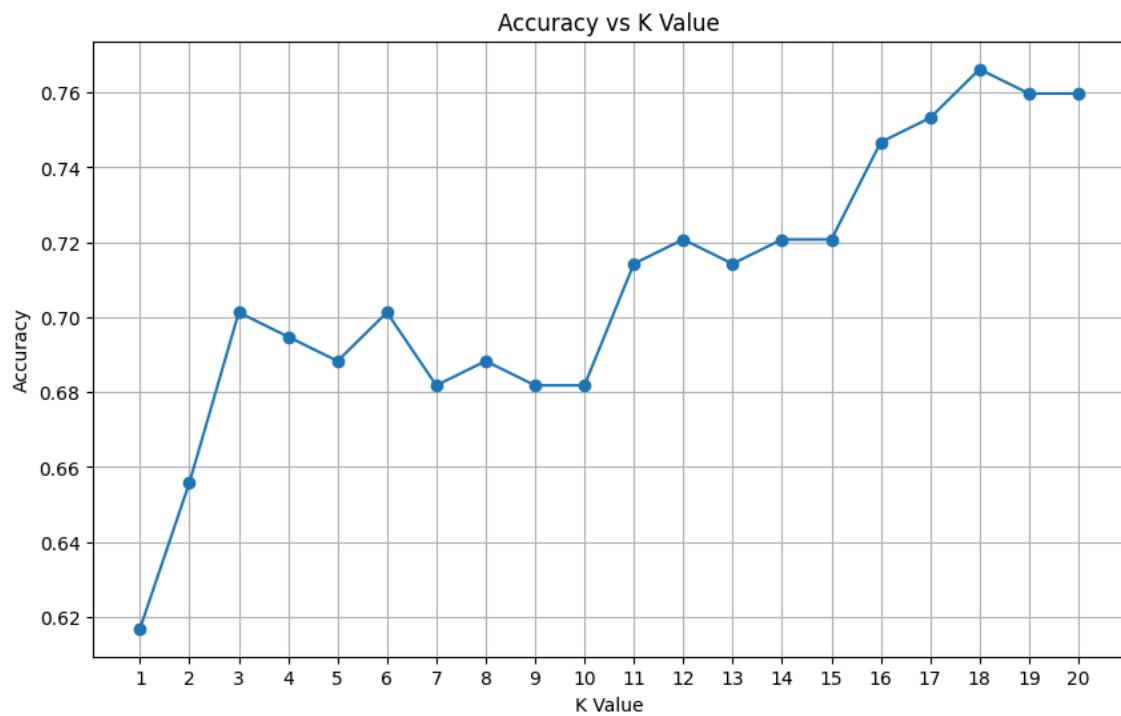


Classification Report:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	10
Iris-versicolor	1.00	1.00	1.00	9
Iris-virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Diabetes.csv

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
data = pd.read_csv('diabetes.csv')
X = data.drop('Outcome', axis=1)
y = data['Outcome']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
k_values = range(1, 21)
accuracy_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy_scores, marker='o')
plt.title('Accuracy vs K Value')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid()
plt.show()
optimal_k = k_values[np.argmax(accuracy_scores)]
print(f"Optimal K value: {optimal_k}")
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy:.4f}")
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No Diabetes', 'Diabetes'])
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
print("\nModel Evaluation:")
print(f"- True Positives: {cm[1,1]}")
print(f"- True Negatives: {cm[0,0]}")
print(f"- False Positives: {cm[0,1]}")
print(f"- False Negatives: {cm[1,0]}")
```

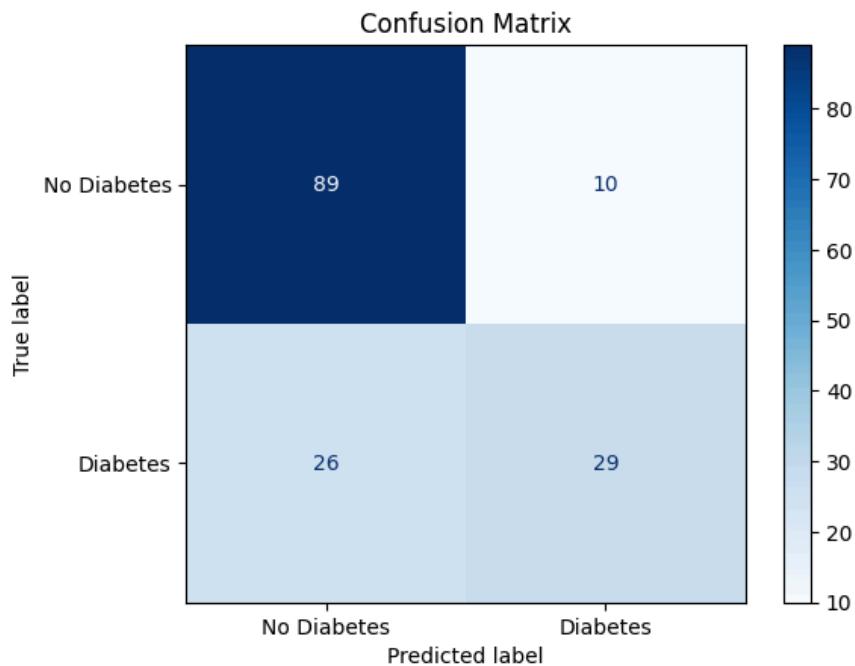


Optimal K value: 18

Accuracy: 0.7662

Confusion Matrix:

```
[[89 10]
 [26 29]]
```



Model Evaluation:
 - True Positives: 29
 - True Negatives: 89
 - False Positives: 10
 - False Negatives: 26

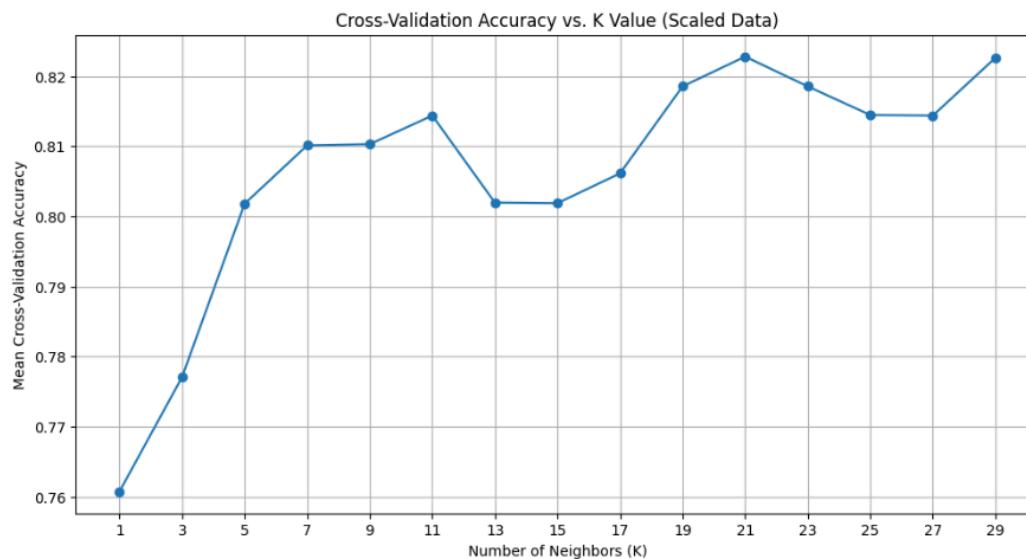
Heart.csv

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_score
try:
    df = pd.read_csv("heart.csv")
except FileNotFoundError:
    print("Error: 'heart.csv' not found. Please make sure the file is in the correct directory.")
    exit()
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
k_values = list(range(1, 31, 2)) # Try odd k values from 1 to 30
cv_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_scaled, y_train, cv=5, scoring='accuracy')
    cv_scores.append(scores.mean())
plt.figure(figsize=(12, 6))
plt.plot(k_values, cv_scores, marker='o')
plt.title('Cross-Validation Accuracy vs. K Value (Scaled Data)')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Mean Cross-Validation Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()
best_k_index = cv_scores.index(max(cv_scores))
best_k = k_values[best_k_index]
print(f"\nThe optimal K value found through cross-validation is: {best_k}")
knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
knn_classifier.fit(X_train_scaled, y_train)
y_pred = knn_classifier.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"\nK-Nearest Neighbors Classifier with K = {best_k} (Scaled Data)")
print("Accuracy Score on Test Data:", accuracy)
plt.figure(figsize=(8, 6))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Heart Disease', 'Heart Disease'], yticklabels=['No Heart Disease', 'Heart Disease'])
```

```

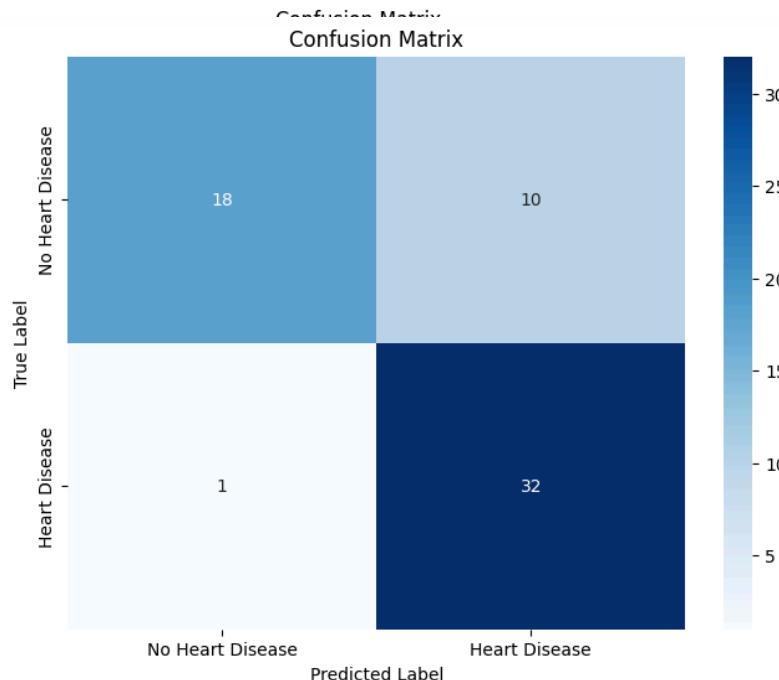
Disease'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
print("\nClassification Report on Test Data:\n", report)

```



The optimal K value found through cross-validation is: 21

K-Nearest Neighbors Classifier with K = 21 (Scaled Data)
Accuracy Score on Test Data: 0.819672131147541



```

Classification Report on Test Data:
      precision    recall  f1-score   support

          0       0.95     0.64      0.77      28
          1       0.76     0.97      0.85      33

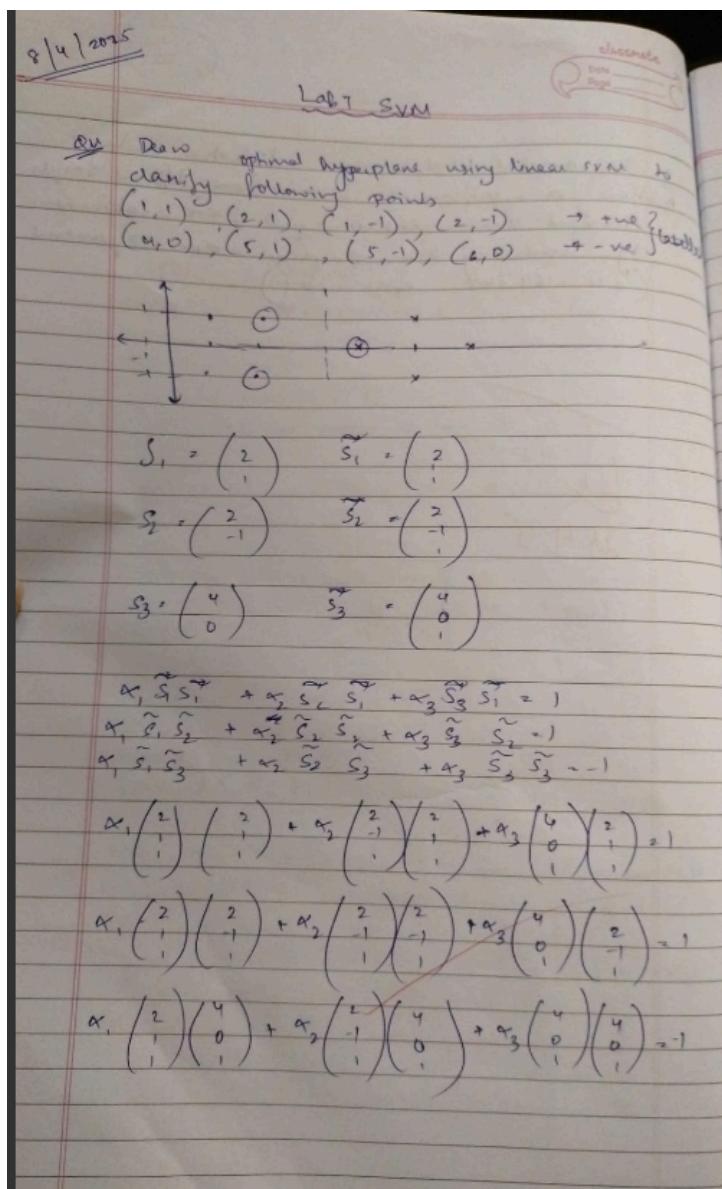
   accuracy                           0.82      61
  macro avg       0.85     0.81      0.81      61
weighted avg       0.85     0.82      0.81      61

```

Program 7

Build Support vector machine model for a given dataset

Screenshot:



classmate
Date _____
Page _____

$$\begin{aligned}
 & 6\alpha_1 + 4\alpha_2 + 9\alpha_3 = 1 \\
 & 4\alpha_1 + 6\alpha_2 + 9\alpha_3 = 1 \\
 & 9\alpha_1 + 9\alpha_2 + 17\alpha_3 = -1
 \end{aligned}$$

$$\begin{aligned}
 \alpha_1 &= \frac{13}{4} \\
 \alpha_2 &= \frac{13}{4} \\
 \alpha_3 &= \frac{-1}{2}
 \end{aligned}$$

$$\begin{aligned}
 w_0 \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3 \\
 = \frac{13}{4} \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + \frac{13}{4} \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} \\
 = \begin{pmatrix} -1 \\ 0 \\ 3 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ -3 \end{pmatrix}
 \end{aligned}$$

$$w = \begin{pmatrix} -1 \\ 0 \\ 3 \end{pmatrix}, \quad b = -3 \Rightarrow b + 3 = 0$$

\rightarrow line \parallel to y -axis

intercept at x axis?

Iris.csv

```

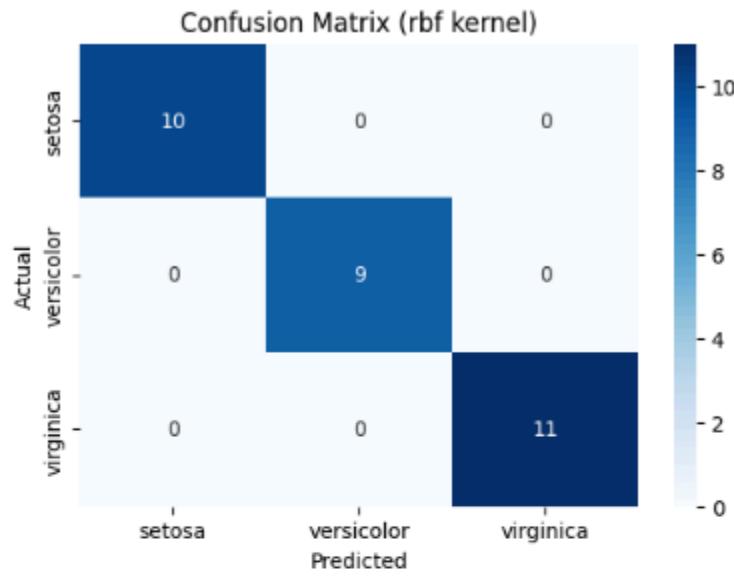
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
iris_df = pd.read_csv('iris.csv')
X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
def evaluate_svm(kernel_type):
    svm_clf = SVC(kernel=kernel_type, random_state=42)
    svm_clf.fit(X_train, y_train)
    y_pred = svm_clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=svm_clf.classes_,
                yticklabels=svm_clf.classes_)
    plt.title(f'Confusion Matrix ({kernel_type} kernel)')
    plt.ylabel('Actual')

```

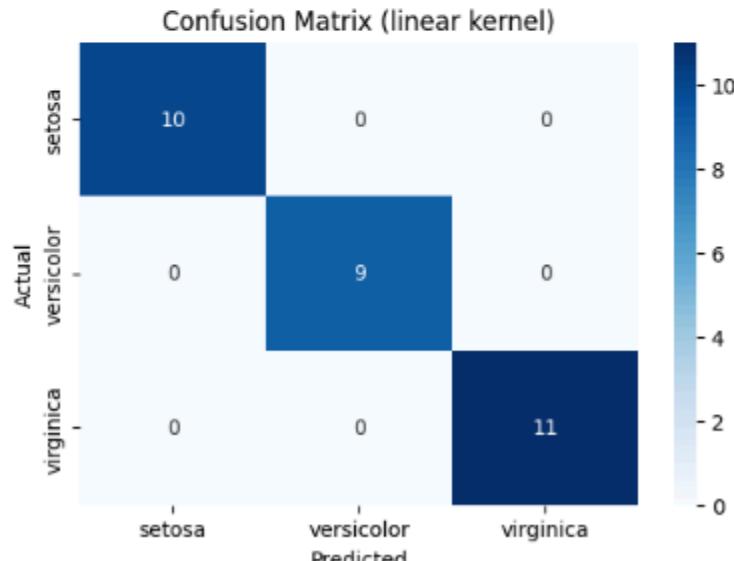
```

plt.xlabel('Predicted')
plt.show()
print(f'Accuracy with {kernel_type} kernel: {accuracy:.4f}')
print("\n")
evaluate_svm('rbf')
evaluate_svm('linear')

```



Accuracy with rbf kernel: 1.0000



Accuracy with linear kernel: 1.0000

Letter Recognition

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score,

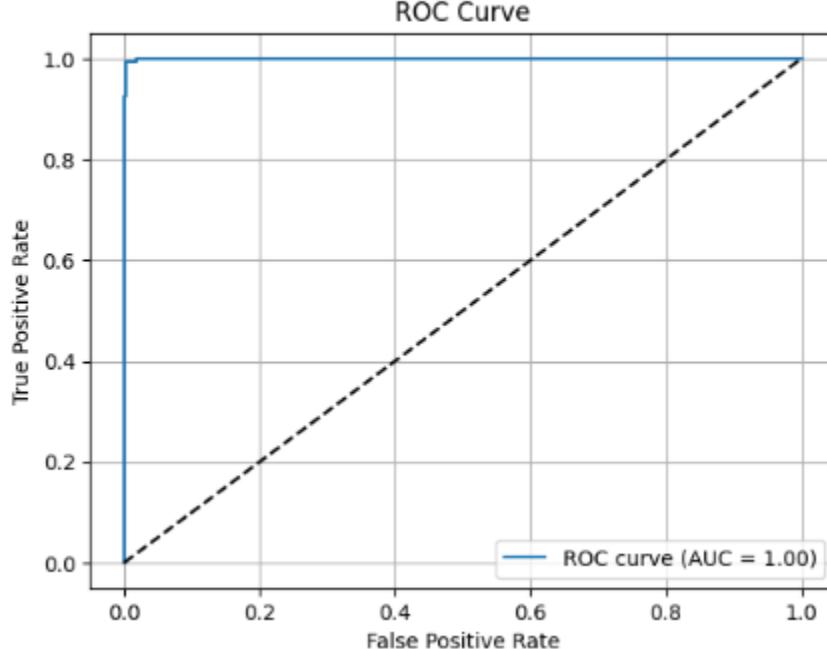
```

```

roc_curve
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt
letters = pd.read_csv("letter-recognition.csv")
X = letters.iloc[:, 1:]
y = letters.iloc[:, 0] # assuming first column is label
y_binary = (y == 'A').astype(int) # modify based on actual dataset
X_train, X_test, y_train_bin, y_test_bin = train_test_split(X, y_binary,
test_size=0.2, random_state=42)
svm_model = SVC(kernel='rbf', probability=True)
svm_model.fit(X_train, y_train_bin)
y_pred = svm_model.predict(X_test)
y_prob = svm_model.predict_proba(X_test)[:, 1]
print("Letter Recognition Accuracy:", accuracy_score(y_test_bin, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test_bin, y_pred))
fpr, tpr, _ = roc_curve(y_test_bin, y_prob)
auc_score = roc_auc_score(y_test_bin, y_prob)
plt.figure()
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {auc_score:.2f})")
plt.plot([0, 1], [0, 1], 'k-')
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()

Letter Recognition Accuracy: 0.996
Confusion Matrix:
[[3850  1]
 [ 15 134]]

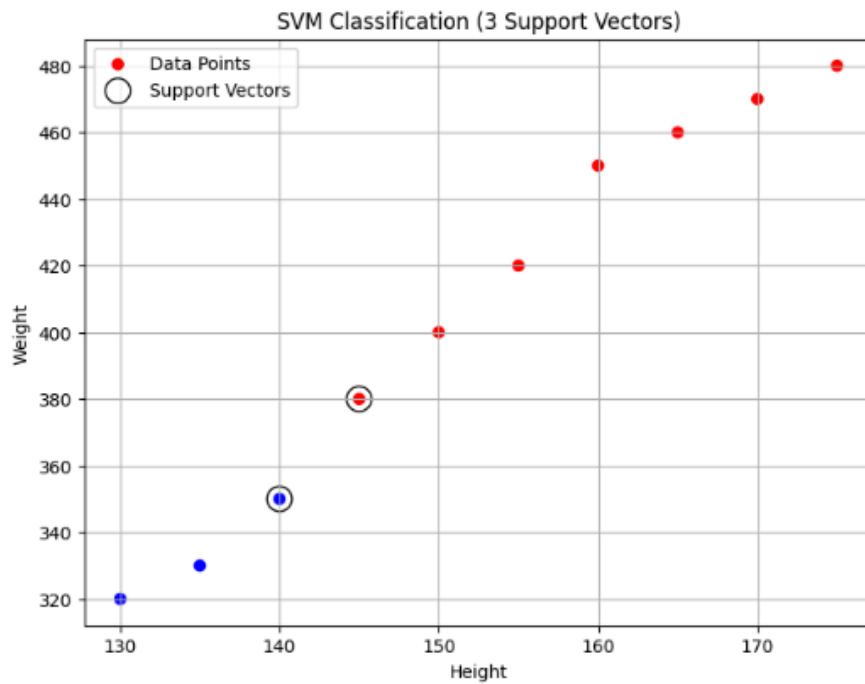
```



Horse Mule dataset

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv("horse_mule_data.csv")
# Encode 'Horse'=0, 'Mule'=1
df['Label'] = LabelEncoder().fit_transform(df['Label'])
X = df[['Height', 'Weight']]
y = df['Label']
model = SVC(kernel='linear', C=1000) # High C -> fewer support vectors
model.fit(X, y)
support_vectors = model.support_vectors_
accuracy = model.score(X, y)
print("Accuracy:", accuracy)
print("Support Vectors:\n", support_vectors)
print("Number of Support Vectors:", len(support_vectors))
colors = ['red' if label == 0 else 'blue' for label in y]
plt.figure(figsize=(8,6))
plt.scatter(X['Height'], X['Weight'], c=colors, label='Data Points')
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
           s=200, facecolors='none', edgecolors='black', label='Support Vectors')
plt.xlabel("Height")
plt.ylabel("Weight")
plt.title("SVM Classification (3 Support Vectors)")
plt.legend()
plt.grid(True)
plt.show()
```

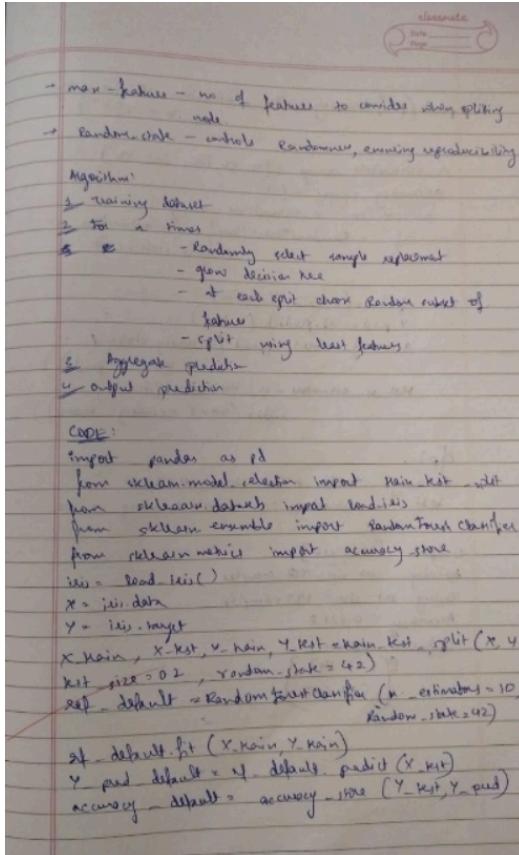
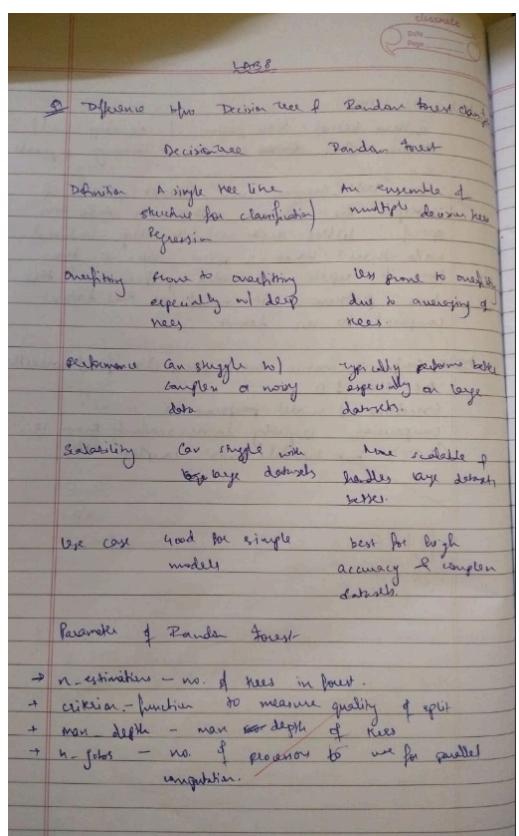
Accuracy: 1.0
Support Vectors:
[[145, 380.]
[148, 350.]]
Number of Support Vectors: 2



Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot:



Iris.csv

```

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
iris = pd.read_csv("iris.csv")
X = iris.iloc[:, :-1]
y = iris.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy with default n_estimators (10): {accuracy:.4f}')
best_accuracy = 0
best_n_estimators = 0
for n_estimators in range(10, 201, 10):
    rf_classifier = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_n_estimators = n_estimators
print(f'Best Accuracy: {best_accuracy:.4f} with n_estimators: {best_n_estimators}')

```

```

accuracy = accuracy_score(y_test, y_pred)
if accuracy > best_accuracy:
    best_accuracy = accuracy
    best_n_estimators = n_estimators
print(f"Best accuracy: {best_accuracy:.4f} achieved with n_estimators = {best_n_estimators}")
Accuracy with default n_estimators (10): 1.0000
Best accuracy: 1.0000 achieved with n_estimators = 10

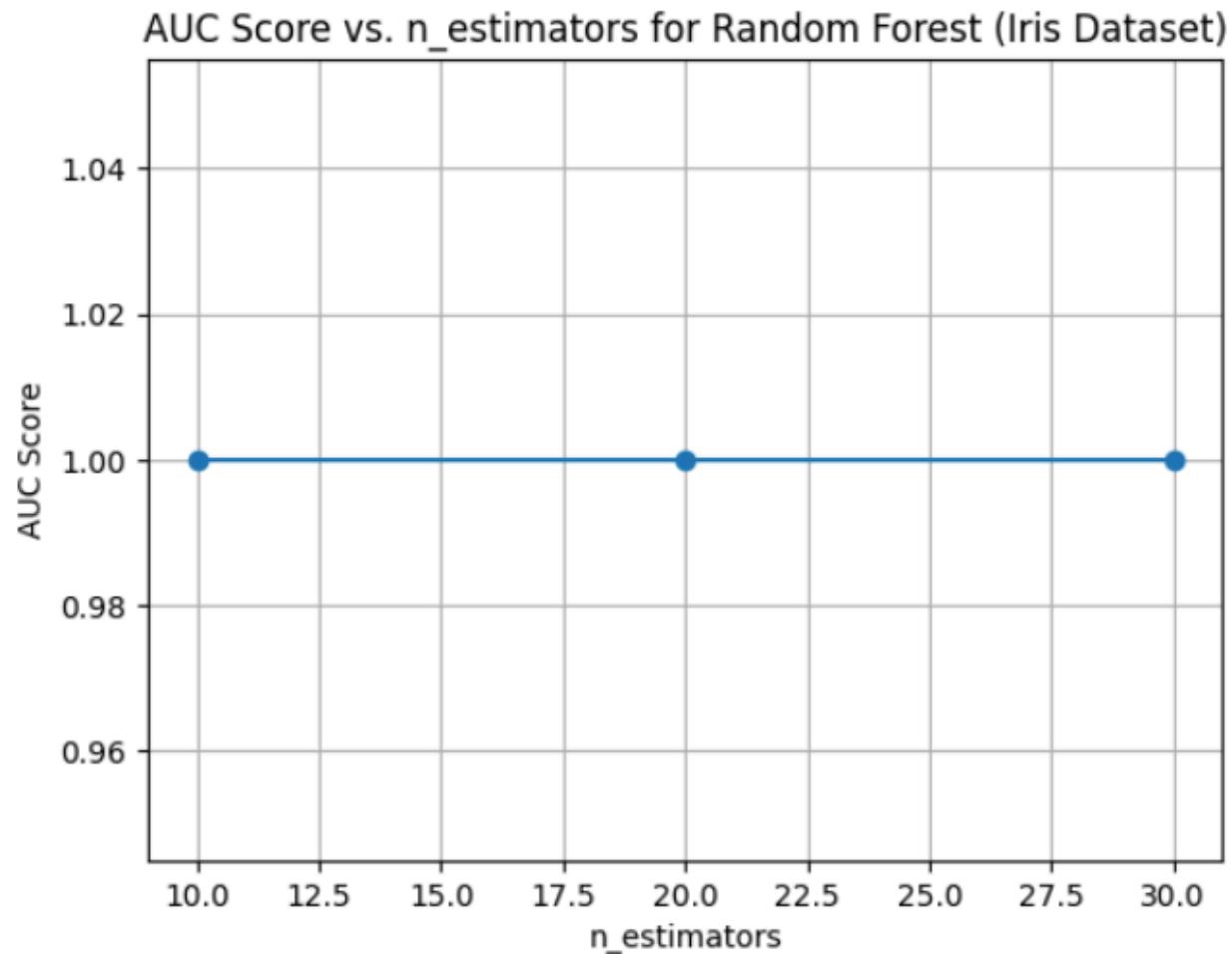
```

```

import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
iris = load_iris()
X, y = iris.data, iris.target
y = label_binarize(y, classes=[0, 1, 2])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
n_estimators_values = [10, 20, 30]
auc_scores = []
for n_estimators in n_estimators_values:
    rf_classifier = OneVsRestClassifier(RandomForestClassifier(n_estimators=n_estimators,
                                                               random_state=42))
    rf_classifier.fit(X_train, y_train)
    y_pred_proba = rf_classifier.predict_proba(X_test)
    auc_scores.append(roc_auc_score(y_test, y_pred_proba, average='weighted', multi_class='ovr'))
    print(f"AUC Score for n_estimators = {n_estimators}: {auc_scores[-1]}")
plt.plot(n_estimators_values, auc_scores, marker='o')
plt.xlabel('n_estimators')
plt.ylabel('AUC Score')
plt.title('AUC Score vs. n_estimators for Random Forest (Iris Dataset)')
plt.grid(True)
plt.show()

```

```
AUC Score for n_estimators = 10: 1.0
AUC Score for n_estimators = 20: 1.0
AUC Score for n_estimators = 30: 1.0
```



Train.csv

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv("train.csv")
df = df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
label_encoders = {}
for col in ['Sex', 'Embarked']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
X = df.drop(columns=['Survived'])
y = df['Survived']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = RandomForestClassifier(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)
print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
```

Accuracy: 0.8212

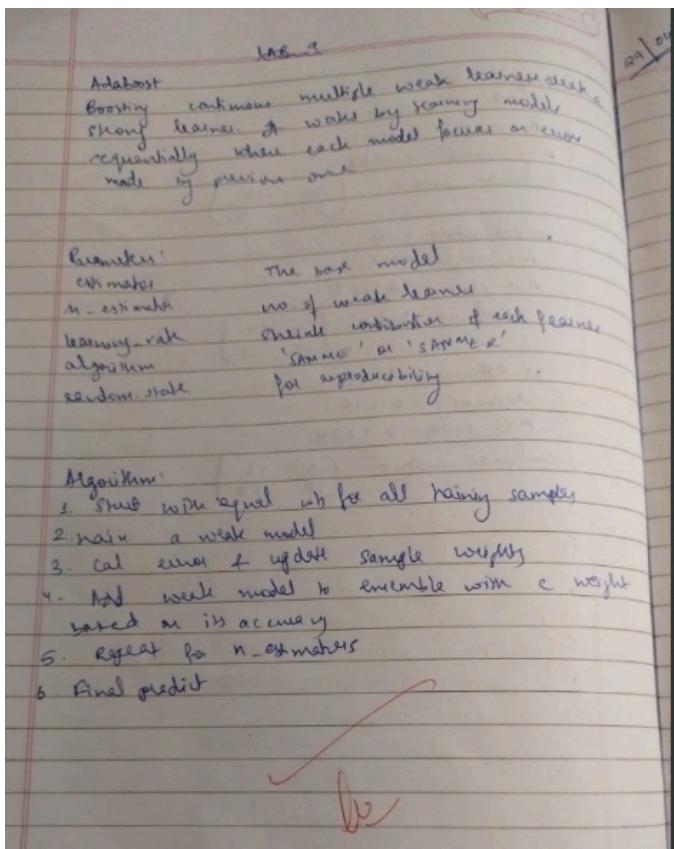
Confusion Matrix:

92	13
19	55

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot:



Income.csv

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
data = pd.read_csv('income.csv')
X = data.drop('income_level', axis=1)
y = data['income_level']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
ada_boost = AdaBoostClassifier(n_estimators=50, random_state=42)
ada_boost.fit(X_train, y_train)
```

```

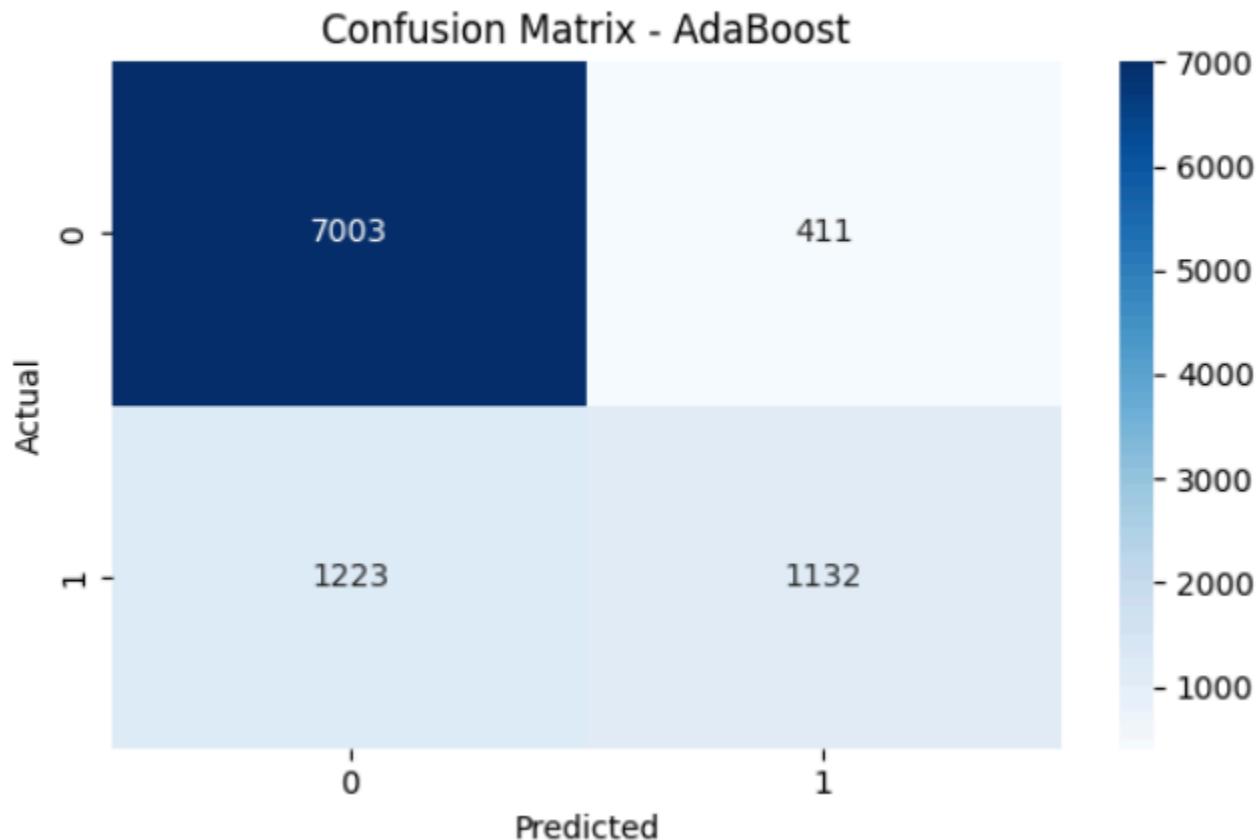
y_pred = ada_boost.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
conf_matrix = confusion_matrix(y_test, y_pred)
print(f"Confusion Matrix:\n{conf_matrix}")
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=ada_boost.classes_, yticklabels=ada_boost.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - AdaBoost')
plt.tight_layout()
plt.show()

```

Accuracy: 0.8327362063670796

Confusion Matrix:

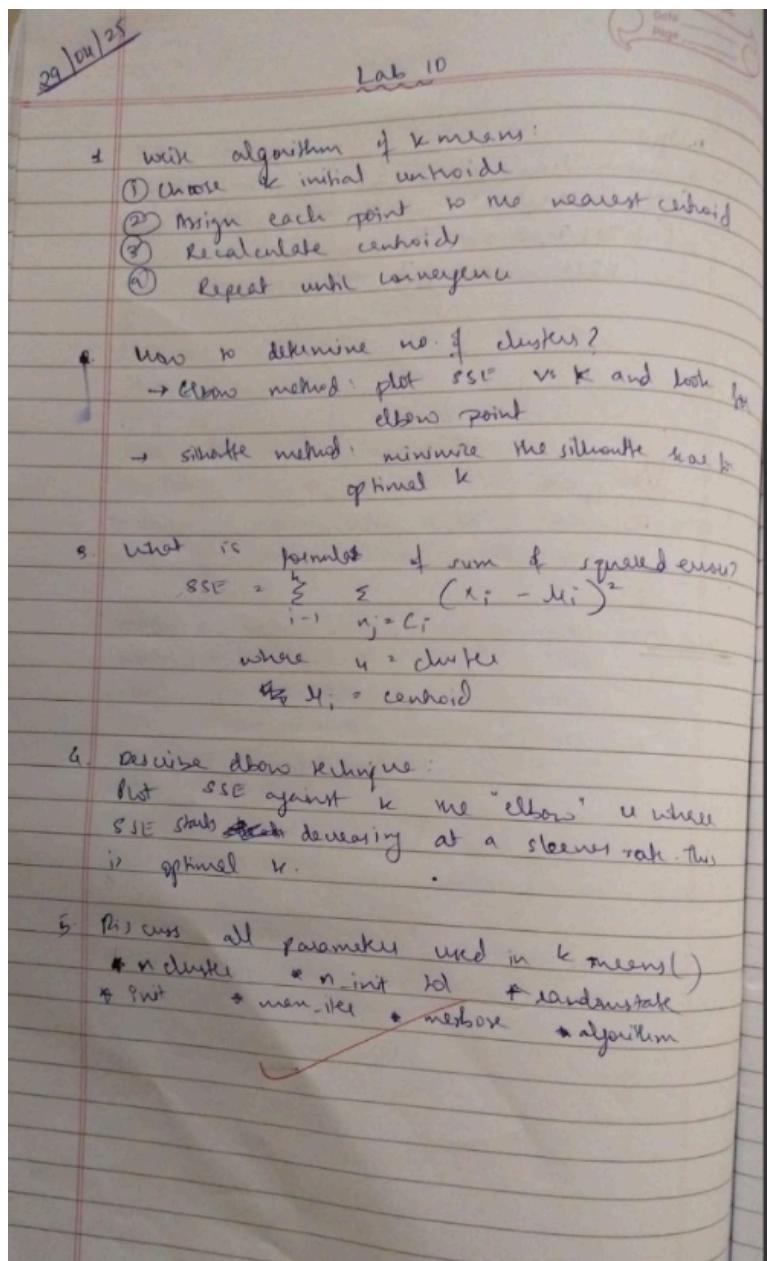
[[7003 411]
[1223 1132]]



Program 10

Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Screenshot:



Income.csv

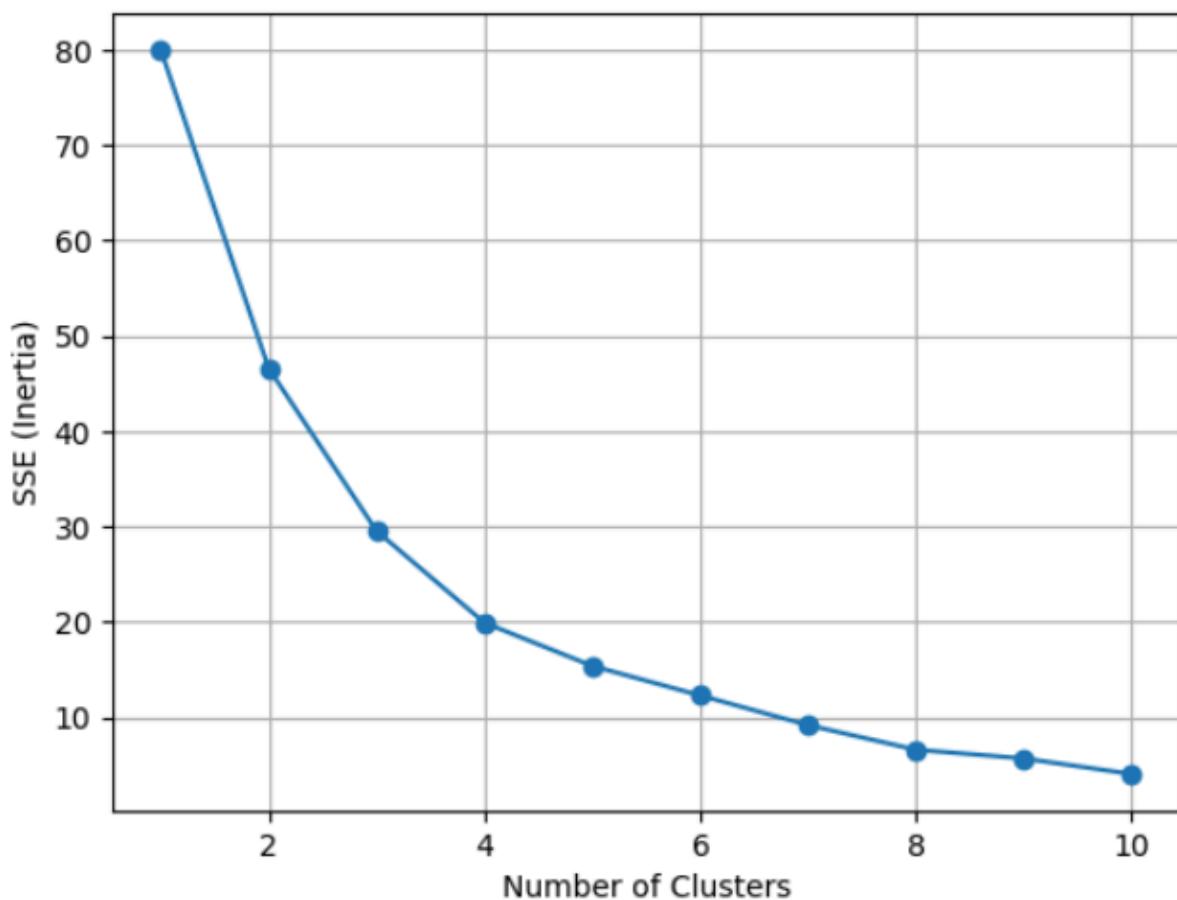
```
import pandas as pd  
import numpy as np
```

```

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
import matplotlib.pyplot as plt
import random
np.random.seed(42)
names = [f"Person_{i}" for i in range(1, 51)]
ages = np.random.randint(20, 60, 50)
incomes = np.random.randint(20000, 100000, 50)
df = pd.DataFrame({
    'Name': names,
    'Age': ages,
    'Income': incomes
})
df.to_csv('income.csv', index=False)
print("income.csv' created successfully.")
from google.colab import files
files.download('income.csv')
data = pd.read_csv('income.csv')
X = data[['Age', 'Income']]
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)
    sse.append(kmeans.inertia_)
plt.plot(k_range, sse, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('SSE (Inertia)')
plt.title('Elbow Method: SSE vs Number of Clusters')
plt.grid(True)
plt.show()
k = 3 # Change this based on the elbow plot
model = KMeans(n_clusters=k, random_state=42)
model.fit(X_train_scaled)
train_preds = model.predict(X_train_scaled)
test_preds = model.predict(X_test_scaled)
true_labels_train = [random.randint(0, k-1) for _ in range(len(train_preds))]
accuracy = adjusted_rand_score(true_labels_train, train_preds)
print("Adjusted Rand Index (proxy accuracy):", round(accuracy, 2))

```

Elbow Method: SSE vs Number of Clusters

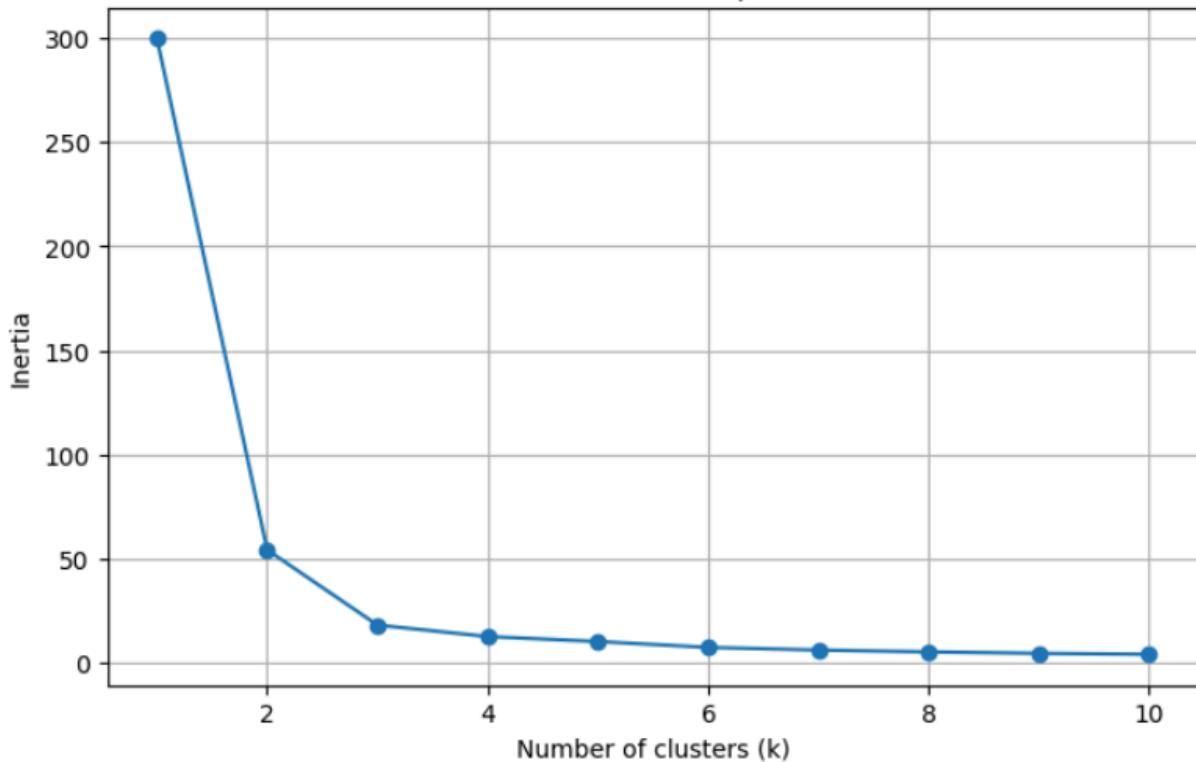


Iris.csv

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
iris = load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)
X = data[["petal length (cm)", "petal width (cm)"]]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
inertia = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of clusters (k)")
```

```
plt.ylabel("Inertia")
plt.grid(True)
plt.show()
```

Elbow Method for Optimal k



Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Screenshot:

Handwritten notes for PCA implementation:

- 1. Calculate mean
- 2. Calculate covariance matrix
- 3. Eigen values of covariance matrix
- 4. Computation of PC eigenvalues - unit eigenvectors
- 5. Computation of first principal components
- 6. Geometrical meaning of first principal components

Given features:

	x_{11}	x_{12}	x_{13}	x_{14}
x_1	8	8.5	13	7
x_2	11	4	5	14

$\bar{x}_1 = 8$ $\bar{x}_2 = 8.5$

$\text{cov}(x_1, x_2) = \frac{1}{N-1} \sum_{n=1}^N (x_{1n} - \bar{x}_1)(x_{2n} - \bar{x}_2)$

$= \frac{1}{3} ((8-8)(11-8.5) + (8-8)(4-8.5) + (13-8)(5-8.5))$
 $= 14$

$\text{cov}(x_1, x_1) = \frac{1}{N-1} \sum_{k=1}^N (x_{1k} - \bar{x}_1)(x_{1k} - \bar{x}_1)$

$= \frac{1}{3} ((8-8)^2 + (8-8)^2 + (13-8)^2)$
 $= 11$

$\text{cov}(x_2, x_2) = -11 - \text{cov}(x_1, x_2)$
 $= 23$

$\text{cov mat}(\delta) = \begin{bmatrix} \text{cov}(x_1, x_1) & \text{cov}(x_1, x_2) \\ \text{cov}(x_2, x_1) & \text{cov}(x_2, x_2) \end{bmatrix}$

• $\begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$

3. Eigenvalues of covariance matrix
 $O = \text{det}(S - dd)$

$\begin{bmatrix} 14-d & -11 \\ -11 & 23-d \end{bmatrix}$

$\Rightarrow d^2 - 31d + 201$
 $d = \frac{1}{2}(31 \pm \sqrt{525})$

$\Rightarrow 30.3849, 6.6151$
 $\Rightarrow d_1, d_2$

4. Computation of eigenvectors
 $U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$

$\begin{bmatrix} v \\ d \end{bmatrix} = (S - dd) x$

$\vdots \begin{bmatrix} 14-d_1 & -11 \\ -11 & 23-d_1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$

$\Rightarrow \begin{bmatrix} (14-d_1)u_1 - 11u_2 \\ -11u_1 + (23-d_1)u_2 \end{bmatrix}$
 $\Rightarrow 0(14-d_1)u_2 - 11u_2 = 0$
 $\Rightarrow -11u_1 + (23-d_1)u_2 = 0$

$\frac{u_1}{11} = \frac{u_2}{(14-d_1)} = 1$

$u_1 = 11u_2$
 $u_1 = \begin{bmatrix} 11 \\ 14-d_1 \end{bmatrix}$

To find eigenvector we compute length of x ,
 $||u_1|| = \sqrt{11^2 + (14-d_1)^2}$
 $= 19.7388$

∴ unit eigenvectors corresponding to d_1 ,
 $e_1 = \begin{bmatrix} 11/\|u_1\| \\ (14-d_1)/\|u_1\| \end{bmatrix}$
 $\approx \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$

$e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

5. Computation of 1st principal component

$\begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix}$

$e_1^T \cdot \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix} = [0.5574 \quad -0.8303] \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix}$
 $= 0.5574(x_{1k} - \bar{x}_1) - 0.8303(x_{2k} - \bar{x}_2)$

```

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
digits = load_digits()
X = digits.data
y = digits.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y, test_size=0.2, random_state=42
)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
score = model.score(X_test, y_test)
print("Model Score (accuracy using .score()):", round(score, 4))

```

```

print(" Accuracy using PCA with 2 components:", round(accuracy, 4))

Model Score (accuracy using .score()): 0.5389
Accuracy using PCA with 2 components: 0.5389

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from scipy.stats import zscore
df = pd.read_csv("/content/heart.csv") # Adjust path if needed
z_scores = np.abs(zscore(df.select_dtypes(include=[np.number])))
df = df[(z_scores < 3).all(axis=1)]
df_encoded = df.copy()
for col in df_encoded.select_dtypes(include=["object"]).columns:
    if df_encoded[col].nunique() <= 2:
        le = LabelEncoder()
        df_encoded[col] = le.fit_transform(df_encoded[col])
    else:
        df_encoded = pd.get_dummies(df_encoded, columns=[col], drop_first=True)
X = df_encoded.drop("target", axis=1) # Replace 'target' if it's named differently
y = df_encoded["target"]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC()
}
print("Model Accuracies (without PCA):")
for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    print(f'{name}: {acc:.4f}')
pca = PCA(n_components=0.95) # Retain 95% variance
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, _, _ = train_test_split(X_pca, y, test_size=0.2, random_state=42)
print("\nModel Accuracies (with PCA):")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    preds = model.predict(X_test_pca)

```

```
acc = accuracy_score(y_test, preds)
print(f'{name}: {acc:.4f}')
```

Model Accuracies (without PCA):

Logistic Regression: 0.8103

Random Forest: 0.7759

SVM: 0.7931

Model Accuracies (with PCA):

Logistic Regression: 0.8276

Random Forest: 0.8103

SVM: 0.7586