VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Anjan C (1BM22CS044)

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
Feb-2025 to June-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 **Anjan C (1BM22CS044),** 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.

Saritha A N Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

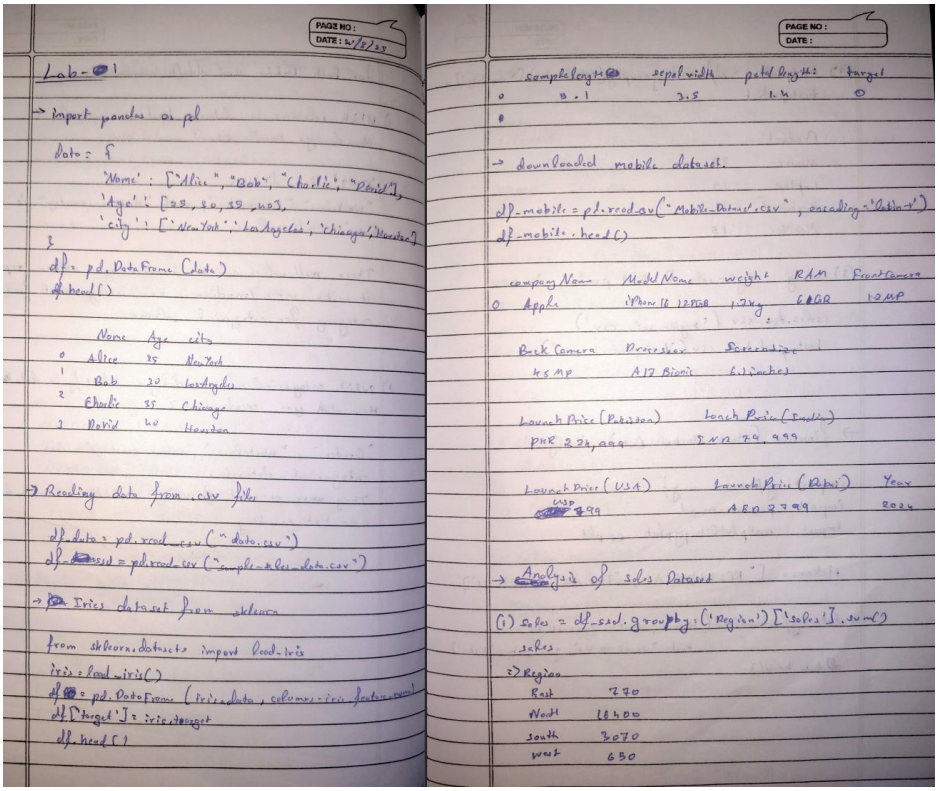
Index

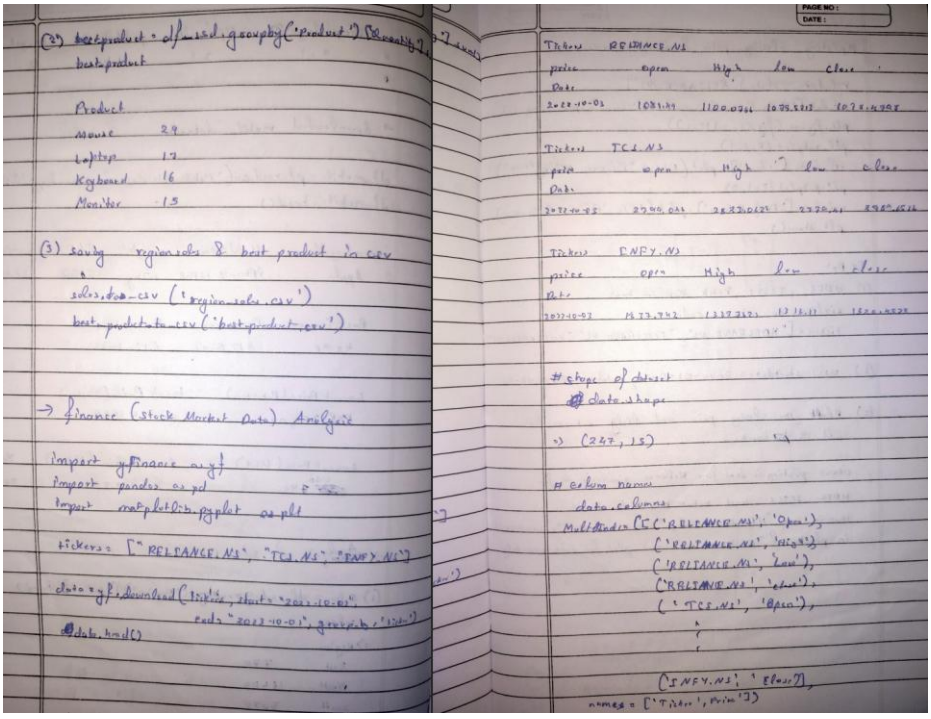
Github Link: https://github.com/AnjanC12/1BM22CS044_Anjan_C_ML_Lab

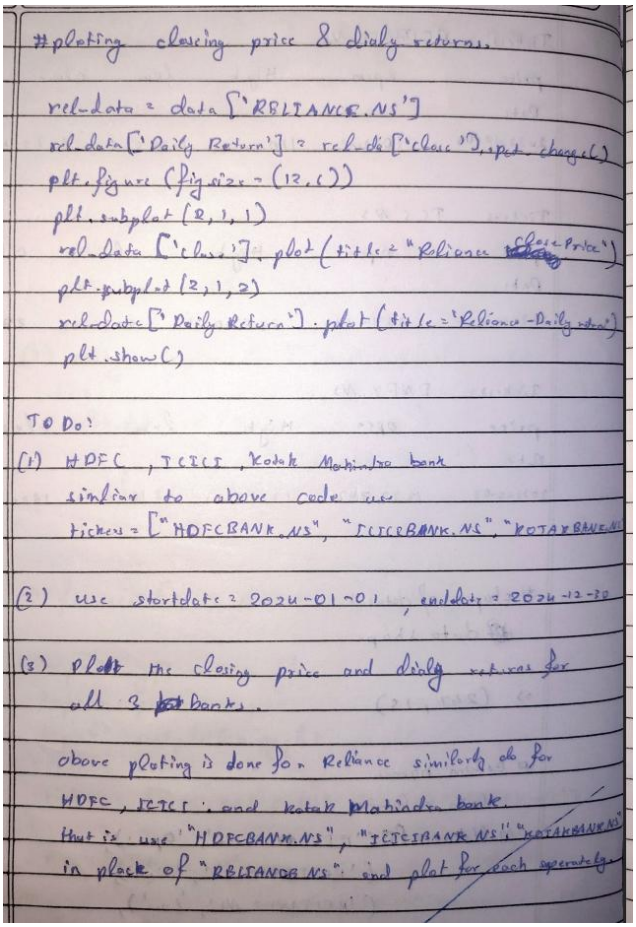
Program 1

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

Screenshots







Code:

```
import pandas as pd

# Create a DataFrame directly from a dictionary

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
```

```
# Load data from a CSV file (replace 'data.csv' with your file path)
file_path = '/content/industry.csv'
# Ensure the file exists in the same directory
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")
```

```
Sample data:
                          Industry
0             Accounting/Finance
1   Advertising/Public Relations
2            Aerospace/Aviation
3  Arts/Entertainment/Publishing
4                      Automotive
```

```
import pandas as pd
# Reading data from a CSV file
data = {
'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Evangline'],
'USN': ['1BM22CS025', '1BM22CS030', '1BM22CS035', '1BM22CS040', '1BM22CS045'],
'Marks': [25, 30, 35, 40, 45]
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

```
Sample data:
          Name          USN  Marks
0      Alice  1BM22CS025     25
1        Bob  1BM22CS030     30
2    Charlie  1BM22CS035     35
3      David  1BM22CS040     40
4  Evangline  1BM22CS045     45
```

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

```
Sample data:
        age       sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6  target
0 -0.002592  0.019907 -0.017646   151.0
1 -0.039493 -0.068332 -0.092204    75.0
2 -0.002592  0.002861 -0.025930   141.0
3  0.034309  0.022688 -0.009362   206.0
4 -0.002592 -0.031988 -0.046641   135.0
```

```
# Load data from a CSV file (replace 'data.csv' with your file path)
file_path = '/content/sample_data/california_housing_train.csv' # Ensure the file exists in the same directory
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")
```

```
Sample data:
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -114.31     34.19                15.0       5612.0          1283.0
1    -114.47     34.40                19.0       7650.0          1901.0
2    -114.56     33.69                17.0        720.0           174.0
3    -114.57     33.64                14.0       1501.0           337.0
4    -114.57     33.57                20.0       1454.0           326.0

   population  households  median_income  median_house_value
0      1015.0       472.0         1.4936             66900.0
1      1129.0       463.0         1.8200             80100.0
2       333.0       117.0         1.6509             85700.0
3       515.0       226.0         3.1917             73400.0
4       624.0       262.0         1.9250             65500.0
```

```
# Load data from a CSV file (replace 'data.csv' with your file path)
# downloading and loading
file_path = '/content/Dataset of Diabetes .csv' # Ensure the file exists in the same directory
df = pd.read_csv(file_path)
print("Sample data:")
```

```
print(df.head())
print("\n")
```

```
Sample data:
    ID  No_Pation Gender  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0  502      17975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1  735      34221      M   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2  420      47975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
3  680      87656      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
4  504      34223      M   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4

    BMI CLASS
0  24.0     N
1  23.0     N
2  24.0     N
3  24.0     N
4  21.0     N
```

```
import pandas as pd
# Reading data from a CSV file
df =pd.read_csv('/content/sample_data/california_housing_test.csv')
# Displaying the first few rowsof the DataFrame
print(df.head())
# Writing the DataFrame to a CSV file
df.to_csv('output.csv',index=False)
print("Data saved tooutput.csv")
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.05     37.37                27.0       3885.0           661.0
1    -118.30     34.26                43.0       1510.0           310.0
2    -117.81     33.78                27.0       3589.0           507.0
3    -118.36     33.82                28.0         67.0            15.0
4    -119.67     36.33                19.0       1241.0           244.0

   population  households  median_income  median_house_value
0      1537.0       606.0         6.6085            344700.0
1       809.0       277.0         3.5990            176500.0
2      1484.0       495.0         5.7934            270500.0
3        49.0        11.0         6.1359            330000.0
4       850.0       237.0         2.9375             81700.0
Data saved tooutput.csv
```

```
# Reading sales data from a CSV file
california_df =pd.read_csv('/content/sample_data/california_housing_test.csv')
# Displaying the first fewrows of the dataset
print("First few rows of the california_housing_test data:")
print(california_df.head())
```

```
First few rows of the california_housing_test data:
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.05     37.37                27.0       3885.0           661.0
1    -118.30     34.26                43.0       1510.0           310.0
2    -117.81     33.78                27.0       3589.0           507.0
3    -118.36     33.82                28.0         67.0            15.0
4    -119.67     36.33                19.0       1241.0           244.0

   population  households  median_income  median_house_value
0      1537.0       606.0         6.6085            344700.0
1       809.0       277.0         3.5990            176500.0
2      1484.0       495.0         5.7934            270500.0
3        49.0        11.0         6.1359            330000.0
4       850.0       237.0         2.9375             81700.0
```

```
# Grouping by Region and calculating total sales
```

```
california =california_df.groupby('total_rooms')['total_bedrooms'].sum()
print("\nTotal housing by region:")
print(california)
```

```
Total housing by region:
total_rooms
6.0           2.0
16.0          4.0
18.0          3.0
19.0         19.0
21.0          7.0
             ...
21988.0    4055.0
23915.0    4135.0
24121.0    4522.0
27870.0    5027.0
30450.0    5033.0
Name: total_bedrooms, Length: 2215, dtype: float64
```

```
# Grouping by Product and calculating total quantity sold
best_selling_homes =
california_df.groupby('housing_median_age')['households'].sum().sort_values(ascending=False)
print("\nBest-selling products by quantity:")
print(best_selling_homes)
```

```
Best-selling products by quantity:
housing_median_age
52.0    64943.0
17.0    58184.0
16.0    49321.0
19.0    47612.0
35.0    45376.0
25.0    44133.0
34.0    42328.0
26.0    42320.0
18.0    42040.0
24.0    41335.0
36.0    40843.0
15.0    40482.0
32.0    39534.0
29.0    38879.0
33.0    38627.0
27.0    38492.0
20.0    37554.0
5.0     37454.0
21.0    37112.0
4.0     35466.0
30.0    35027.0
22.0    34291.0
14.0    33256.0
37.0    31574.0
28.0    30872.0
12.0    28560.0
23.0    28165.0
11.0    25067.0
```

```
# Saving the sales by region data to a CSV file
california.to_csv('california.csv')
# Saving the best-selling products data to a CSV file
best_selling_homes.to_csv('best_selling_homes.csv')
print("\nAnalysis results saved to CSV files.")
```

```
Analysis results saved to CSV files.
```

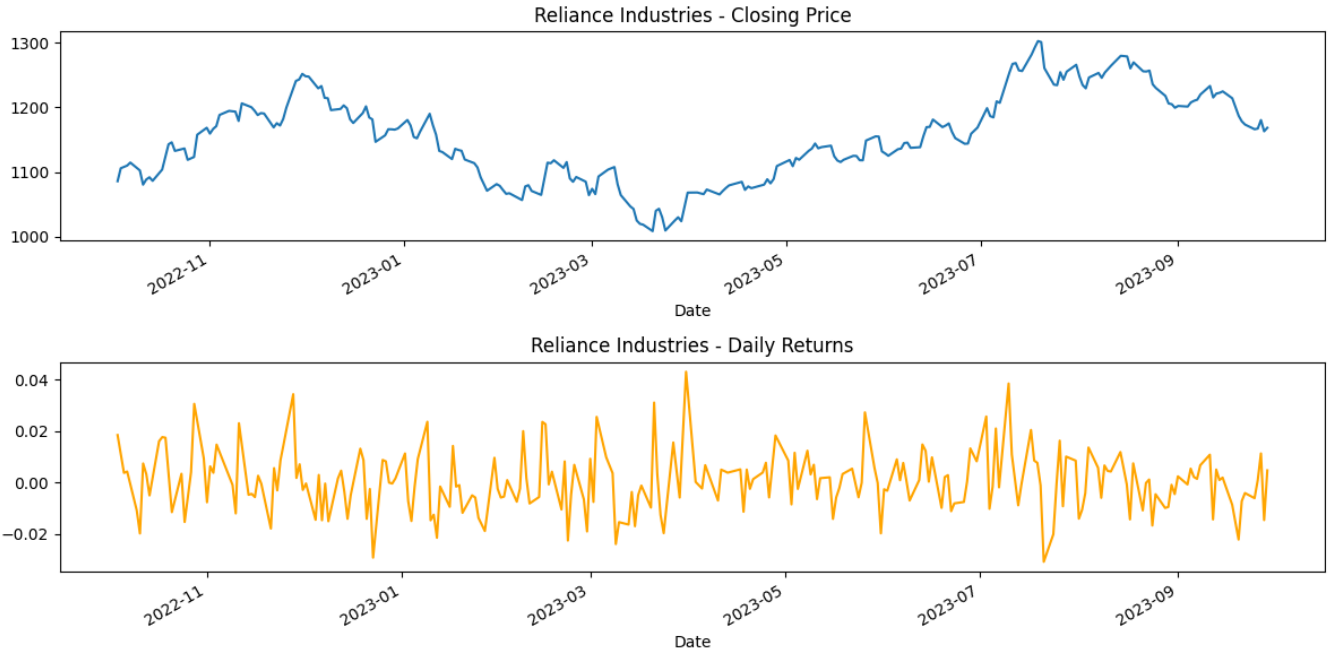
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
# Step 2: Downloading Stock Market Data
# Define the ticker symbols for Indian companies
# Example: Reliance Industries (RELIANCE.NS), TCS (TCS.NS), Infosys (INFY.NS)
tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]
# Fetch historical data for the last 1 year
data = yf.download(tickers, start="2022-10-01", end="2023-10-01",
group_by='ticker')
# Display the first 5 rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  3 of 3 completedFirst 5 rows of the dataset:
Ticker      RELIANCE.NS                                                \
Price             Open         High          Low        Close    Volume
Date
2022-10-03  1096.071886  1107.736072  1083.009806  1085.988892  11852723
2022-10-04  1098.959251  1108.217280  1095.453061  1106.017334   8948850
2022-10-06  1113.258819  1122.883445  1108.285998  1110.096313  13352162
2022-10-07  1106.681897  1120.087782  1106.681897  1114.794189   7714340
2022-10-10  1102.259136  1108.034009  1094.467737  1102.625854   6329527

Ticker           TCS.NS                                                \
Price              Open         High          Low        Close   Volume
Date
2022-10-03  2894.197635  2919.032606  2873.904430  2884.485840  1763331
2022-10-04  2927.970939  2993.730628  2921.254903  2987.111084  2145875
2022-10-06  3006.293304  3018.855764  2988.367592  2997.547852  1790816
2022-10-07  2993.150777  3000.495078  2955.173685  2961.744629  1939879
2022-10-10  2908.692292  3021.754418  2903.860578  3013.588867  3064063

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  1370.286797  1381.182015  1364.412900  1374.881714  3994466
2022-10-10  1351.338576  1387.956005  1351.338576  1385.729614  5274677
```

```
# Step 3: Basic Data Exploration
# Check the shape of the dataset
print("\nShape of the dataset:")
print(data.shape)
# Check column names
print("\nColumn names:")
print(data.columns)
# Summary statistics for a specific stock (e.g., Reliance)
reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
# Calculate daily returns
# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
reliance_data = data['RELIANCE.NS'].copy()
# Now, apply the calculation
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
```

```
Shape of the dataset:
(247, 15)

Column names:
MultiIndex([('RELIANCE.NS',   'Open'),
            ('RELIANCE.NS',   'High'),
            ('RELIANCE.NS',    'Low'),
            ('RELIANCE.NS',  'Close'),
            ('RELIANCE.NS', 'Volume'),
            (     'TCS.NS',   'Open'),
            (     'TCS.NS',   'High'),
            (     'TCS.NS',    'Low'),
            (     'TCS.NS',  'Close'),
            (     'TCS.NS', 'Volume'),
            (    'INFY.NS',   'Open'),
            (    'INFY.NS',   'High'),
            (    'INFY.NS',    'Low'),
            (    'INFY.NS',  'Close'),
            (    'INFY.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   1155.033899  1163.758985  1144.612976  1154.002433  1.316652e+07
std      65.890843    66.876907    65.755901    66.726021  6.754099e+06
min    1015.178443  1017.470038   999.137216  1008.876526  3.370033e+06
25%    1106.532938  1111.081861  1092.347974  1104.997559  8.717141e+06
50%    1155.424265  1163.078198  1146.716157  1155.240967  1.158959e+07
75%    1202.667031  1209.102783  1193.235594  1201.447937  1.530302e+07
max    1297.045129  1308.961472  1281.920577  1302.476196  5.708188e+07
```

```
# Plot the closing price and daily returns
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()
```




```
# Step 4: Saving the Processed Data to a New CSV File
# Save the Reliance data to a CSV file
reliance_data.to_csv('reliance_stock_data.csv')
print("\nReliance stock data saved to 'reliance_stock_data.csv'.")
```

```
Reliance stock data saved to 'reliance_stock_data.csv'.
```

```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
group_by='ticker')
# Display the first 5 rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())
```

```
[*********************100%***********************]  3 of 3 completed
First 5 rows of the dataset:
Ticker     ICICIBANK.NS                                          \
Price              Open        High         Low       Close    Volume
Date
2024-01-01   983.086778  996.273246  982.541485  990.869812   7683792
2024-01-02   988.490253  989.134730  971.883221  973.866150  16263825
2024-01-03   976.295294  979.567116  966.777197  975.650818  16826752
2024-01-04   977.980767  980.707295  973.519176  978.724365  22789140
2024-01-05   979.567084  989.779158  975.402920  985.218445  14875499

Ticker      HDFCBANK.NS                                           \
Price              Open         High          Low        Close    Volume
Date
2024-01-01  1683.017598  1686.125187  1669.206199  1675.223999   7119843
2024-01-02  1675.914685  1679.860799  1665.950651  1676.210571  14621046
2024-01-03  1679.071480  1681.735059  1646.466666  1650.363525  14194881
2024-01-04  1655.394910  1672.116520  1648.193203  1668.071777  13367028
2024-01-05  1664.421596  1681.932477  1645.628180  1659.538208  15944735

Ticker     KOTAKBANK.NS
Price              Open         High          Low        Close   Volume
Date
2024-01-01  1906.909954  1916.899006  1891.027338  1907.059814  1425902
2024-01-02  1905.911108  1905.911108  1858.063525  1863.008179  5120796
2024-01-03  1861.959234  1867.952665  1845.627158  1863.857178  3781515
2024-01-04  1869.451068  1869.451068  1858.513105  1861.559692  2865766
2024-01-05  1863.457575  1867.852782  1839.383985  1845.577148  7799341
```

```
HDFC = data['HDFCBANK.NS']
print("\nSummary statistics for HDFC:")
print(HDFC.describe())
# Calculate daily returns
# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
HDFC = data['HDFCBANK.NS'].copy()
# Now, apply the calculation
HDFC['Daily Return'] = HDFC['Close'].pct_change()
```

```
Summary statistics for HDFC:
Price          Open          High           Low         Close        Volume
count   244.000000   244.000000   244.000000   244.000000  2.440000e+02
mean   1601.375295  1615.443664  1588.221245  1601.898968  2.119658e+07
std     134.648125   134.183203   132.796819   133.748372  2.133860e+07
min    1357.463183  1372.754374  1345.180951  1365.404785  8.798460e+05
25%    1475.316358  1494.072805  1460.259509  1474.564087  1.274850e+07
50%    1627.724976  1638.350037  1616.000000  1625.950012  1.686810e+07
75%    1696.474976  1711.425018  1679.250000  1697.062531  2.295014e+07
max    1877.699951  1880.000000  1858.550049  1871.750000  2.226710e+08
```

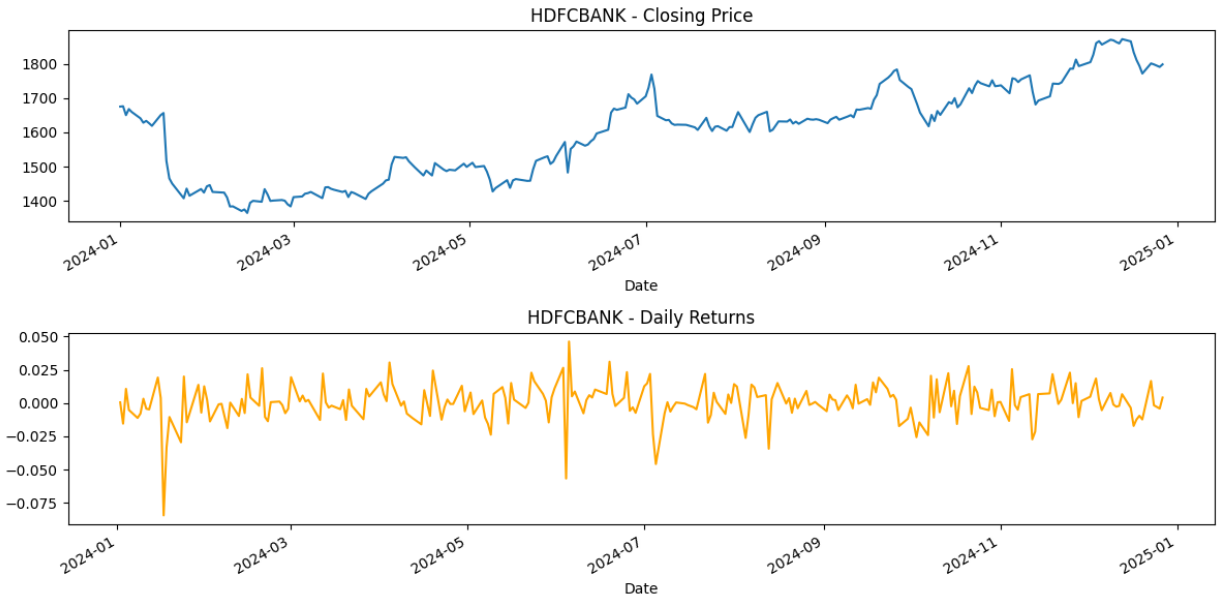
```
ICICI = data['ICICIBANK.NS']
print("\nSummary statistics for ICICI:")
print(ICICI.describe())
# Calculate daily returns
# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
ICICI = data['ICICIBANK.NS'].copy()
# Now, apply the calculation
ICICI['Daily Return'] = ICICI['Close'].pct_change()
```

```
Summary statistics for ICICI:
Price         Open         High          Low        Close        Volume
count   244.000000   244.000000   244.000000   244.000000  2.440000e+02
mean   1161.723560  1173.687900  1151.318979  1162.751791  1.539172e+07
std     104.905646   105.668229   105.083015   105.520481  9.503609e+06
min     965.637027   979.567116   961.869473   971.387512  1.007022e+06
25%    1073.818215  1085.368782  1067.386038  1075.107086  1.014533e+07
50%    1169.443635  1178.450012  1157.361521  1165.470703  1.291768e+07
75%    1248.512512  1261.399994  1236.649963  1250.812531  1.755770e+07
max    1344.900024  1362.349976  1340.050049  1346.099976  7.325777e+07
```

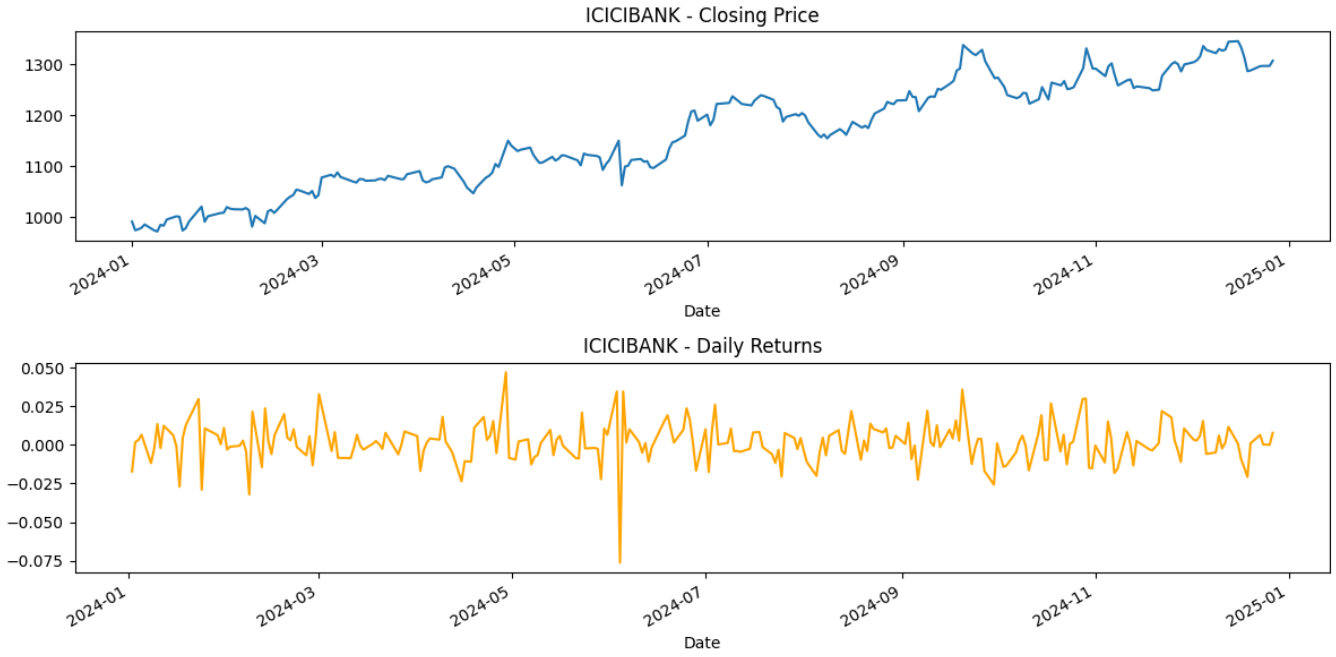
```
KOTAKBANK = data['KOTAKBANK.NS']
print("\nSummary statistics for KOTAKBANK:")
print(KOTAKBANK.describe())
# Calculate daily returns
# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
KOTAKBANK = data['KOTAKBANK.NS'].copy()
# Now, apply the calculation
KOTAKBANK['Daily Return'] = KOTAKBANK['Close'].pct_change()
```

```
Summary statistics for KOTAKBANK:
Price         Open         High          Low        Close        Volume
count   244.000000   244.000000   244.000000   244.000000  2.440000e+02
mean   1771.245907  1787.548029  1754.395105  1770.792347  5.736598e+06
std      62.189675    61.978802    62.765980    62.594747  5.388927e+06
min    1581.266899  1586.161558  1542.159736  1545.006592  1.824890e+05
25%    1733.974927  1754.131905  1719.028421  1736.297058  3.300380e+06
50%    1769.500000  1789.450012  1758.099976  1773.681030  4.307680e+06
75%    1809.925018  1826.998164  1789.912506  1808.155670  6.159475e+06
max    1935.000000  1942.000000  1909.599976  1934.699951  6.617908e+07
```

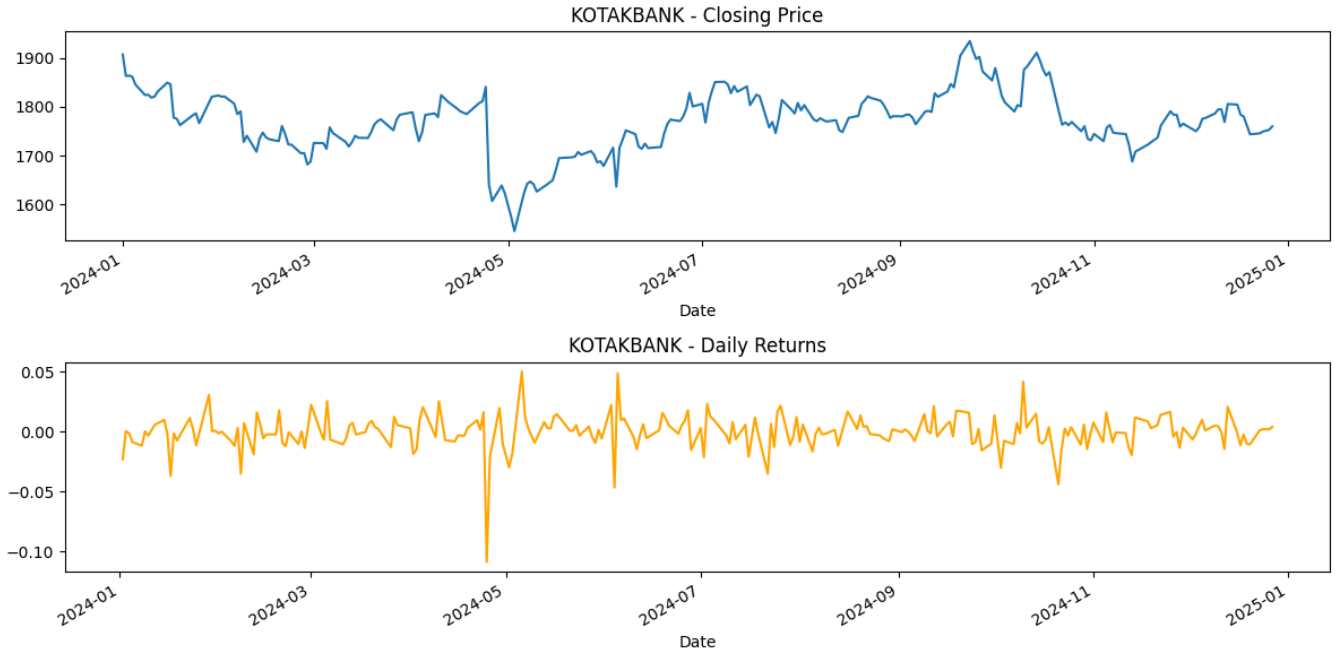
```
# Plot the closing price and daily returns
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
HDFC['Close'].plot(title="HDFCBANK - Closing Price")
plt.subplot(2, 1, 2)
HDFC['Daily Return'].plot(title="HDFCBANK - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```




```
# Plot the closing price and daily returns
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
ICICI['Close'].plot(title="ICICIBANK - Closing Price")
plt.subplot(2, 1, 2)
ICICI['Daily Return'].plot(title="ICICIBANK - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```



```
# Plot the closing price and daily returns
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
KOTAKBANK['Close'].plot(title="KOTAKBANK - Closing Price")
plt.subplot(2, 1, 2)
KOTAKBANK['Daily Return'].plot(title="KOTAKBANK - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```



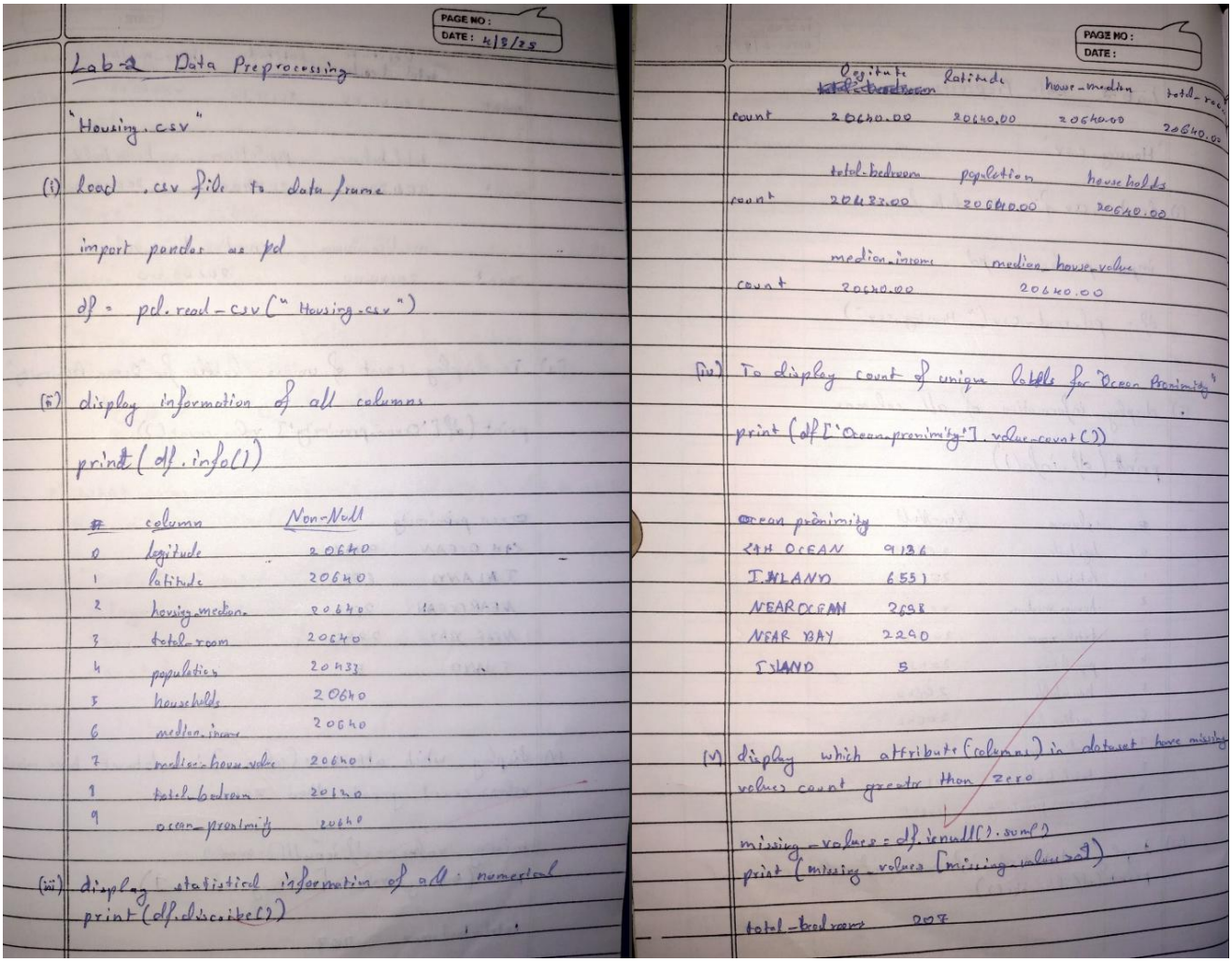

```
# Step 4: Saving the Processed Data to a New CSV File
# Save the Reliance data to a CSV file
HDFC.to_csv('HDFC.csv')
ICICI.to_csv('ICICI.csv')
KOTAKBANK.to_csv('KOTAKBANK.csv')
print("\nSAVED")
```

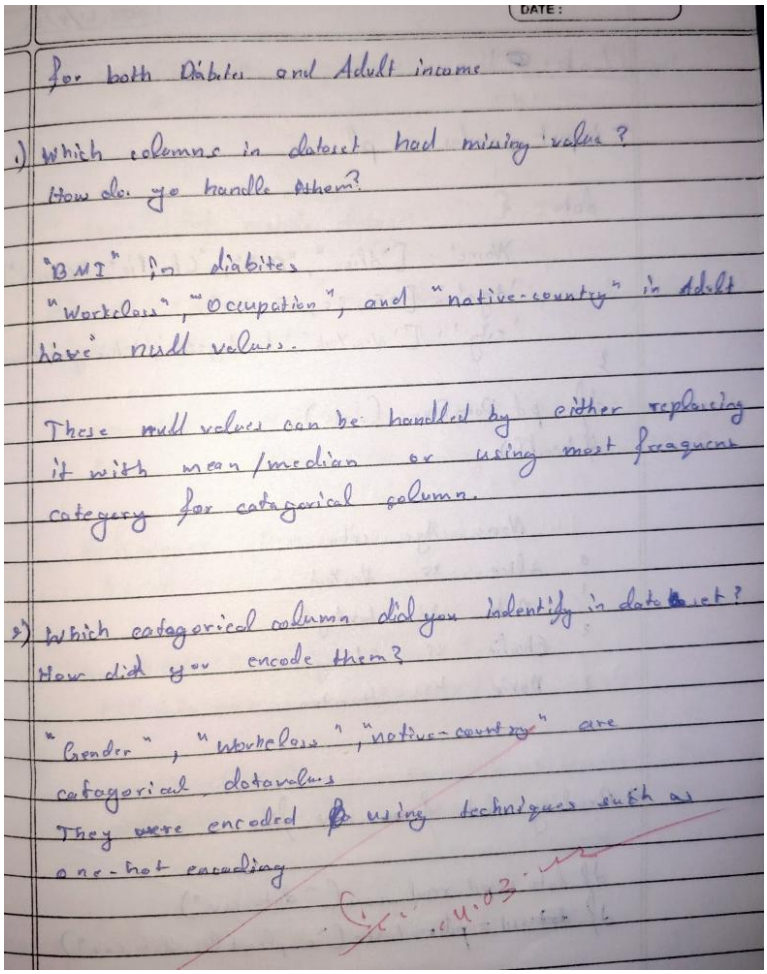
```
SAVED
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset.

Screenshots





Code:

```
import pandas as pd

file_path = '/content/housing.csv'

df = pd.read_csv(file_path)

print("Sample data:")

print(df.head())

print("\n")
```

```
Sample data:
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

#To display information of all columns

print(df.info)

```
<bound method DataFrame.info of        longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0        -122.23     37.88                41.0        880.0           129.0
1        -122.22     37.86                21.0       7099.0          1106.0
2        -122.24     37.85                52.0       1467.0           190.0
3        -122.25     37.85                52.0       1274.0           235.0
4        -122.25     37.85                52.0       1627.0           280.0
...          ...       ...                 ...          ...             ...
20635    -121.09     39.48                25.0       1665.0           374.0
20636    -121.21     39.49                18.0        697.0           150.0
20637    -121.22     39.43                17.0       2254.0           485.0
20638    -121.32     39.43                18.0       1860.0           409.0
20639    -121.24     39.37                16.0       2785.0           616.0

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
...               ...
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[20640 rows x 10 columns]>
```

#To display statistical information of all numerical

print(df.describe())

```
          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  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000
```

#To display the count of unique labels for “Ocean Proximity” column

print(df['ocean_proximity'].value_counts())

```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

#To display which attributes (columns) in a dataset have missing values count greater than zero

print(df.isnull().sum())

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
def createdata():
  data = {
     'Age': np.random.randint(18, 70, size=20),
     'Salary': np.random.randint(30000, 120000, size=20),
     'Purchased': np.random.choice([0, 1], size=20),
     'Gender': np.random.choice(['Male', 'Female'], size=20),
     'City': np.random.choice(['New York', 'San Francisco', 'Los Angeles'], size=20)
  }

  df = pd.DataFrame(data)
  return df
```

Vdf = createdata()

df.head(10)

	Age	Salary	Purchased	Gender	City
0	67	95582	0	Female	Los Angeles
1	58	75108	1	Female	San Francisco
2	64	94631	1	Male	New York
3	42	71454	0	Female	New York
4	44	108391	1	Male	San Francisco
5	20	106194	1	Male	New York
6	37	82085	0	Male	New York
7	27	110483	1	Female	New York
8	42	57678	1	Female	San Francisco
9	63	59239	0	Female	San Francisco

```
df.shape
```

```
(20, 5)
```

Introduce some missing values for demonstration

df.loc[5, 'Age'] = np.nan

df.loc[10, 'Salary'] = np.nan

df.head(10)

	Age	Salary	Purchased	Gender	City
0	67.0	95582.0	0	Female	Los Angeles
1	58.0	75108.0	1	Female	San Francisco
2	64.0	94631.0	1	Male	New York
3	42.0	71454.0	0	Female	New York
4	44.0	108391.0	1	Male	San Francisco
5	NaN	106194.0	1	Male	New York
6	37.0	82085.0	0	Male	New York
7	27.0	110483.0	1	Female	New York
8	42.0	57678.0	1	Female	San Francisco
9	63.0	59239.0	0	Female	San Francisco

Basic information about the dataset

print(df.info())

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Age        19 non-null     float64
 1   Salary     19 non-null     float64
 2   Purchased  20 non-null     int64  
 3   Gender     20 non-null     object 
 4   City       20 non-null     object 
dtypes: float64(2), int64(1), object(2)
memory usage: 932.0+ bytes
None
```

Summary statistics

print(df.describe())

```
                Age         Salary  Purchased
count  19.000000      19.000000  20.000000
mean   45.947368   78821.315789   0.550000
std    15.356771   24850.883175   0.510418
min    19.000000   37052.000000   0.000000
25%    33.500000   58458.500000   0.000000
50%    42.000000   77139.000000   1.000000
75%    60.500000  101866.000000   1.000000
max    68.000000  112223.000000   1.000000
```

#Code to Find Missing Values

Check for missing values in each column

missing_values = df.isnull().sum()

Display columns with missing values

print(missing_values[missing_values > 0])

```
Age       1
Salary    1
dtype: int64
```

#Set the values to some value (zero, the mean, the median, etc.).

Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary

imputer1 = SimpleImputer(strategy="median")

imputer2 = SimpleImputer(strategy="mean")

df_copy=df

Step 2: Fit the imputer on the "Age" and "Salary"column

Note: SimpleImputer expects a 2D array, so we reshape the column

imputer1.fit(df_copy[["Age"]])

imputer2.fit(df_copy[["Salary"]])

Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column

```
df_copy["Age"] = imputer1.transform(df[["Age"]])
df_copy["Salary"] = imputer2.transform(df[["Salary"]])
# Verify that there are no missing values left
print(df_copy["Age"].isnull().sum())
print(df_copy["Salary"].isnull().sum())
```

```
0
0
```

```
#Handling Categorical Attributes
#Using Ordinal Encoding for gender COlumn and One-Hot Encoding for City Column
# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
# Fit and transform the data
df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])
# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder()
# Fit and transform the "City" column
encoded_data = onehot_encoder.fit_transform(df[["City"]])
# Convert the sparse matrix to a dense array
encoded_array = encoded_data.toarray()
# Convert to DataFrame for better visualization
encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["City"]))
df_encoded = pd.concat([df_copy, encoded_df], axis=1)
df_encoded.drop("Gender", axis=1, inplace=True)
df_encoded.drop("City", axis=1, inplace=True)
print(df_encoded. head())
```

```
    Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  City_New York  \
0  67.0   95582.0          0             1.0               1.0            0.0
1  58.0   75108.0          1             1.0               0.0            0.0
2  64.0   94631.0          1             0.0               0.0            1.0
3  42.0   71454.0          0             1.0               0.0            1.0
4  44.0  108391.0          1             0.0               0.0            0.0

   City_San Francisco
0                 0.0
1                 1.0
2                 0.0
3                 0.0
4                 1.0
```

```
#Data Transformation
```

Min-Max Scaler/Normalization (range 0-1)

#Pros: Keeps all data between 0 and 1; ideal for distance-based models.

#Cons: Can distort data distribution, especially with extreme outliers.

```
normalizer = MinMaxScaler()
df_encoded[['Salary']] = normalizer.fit_transform(df_encoded[['Salary']])
df_encoded.head()
```

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	67.0	0.778625	0	1.0	1.0	0.0	0.0
1	58.0	0.506259	1	1.0	0.0	0.0	1.0
2	64.0	0.765974	1	0.0	0.0	1.0	0.0
3	42.0	0.457650	0	1.0	0.0	1.0	0.0
4	44.0	0.949023	1	0.0	0.0	0.0	1.0

Standardization (mean=0, variance=1)

#Pros: Works well for normally distributed data; suitable for many models.

#Cons: Sensitive to outliers.

```
scaler = StandardScaler()
df_encoded[['Age']] = scaler.fit_transform(df_encoded[['Age']])
df_encoded.head()
```

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	1.456069	0.778625	0	1.0	1.0	0.0	0.0
1	0.839381	0.506259	1	1.0	0.0	0.0	1.0
2	1.250506	0.765974	1	0.0	0.0	1.0	0.0
3	-0.256953	0.457650	0	1.0	0.0	1.0	0.0
4	-0.119912	0.949023	1	0.0	0.0	0.0	1.0

#Removing Outliers

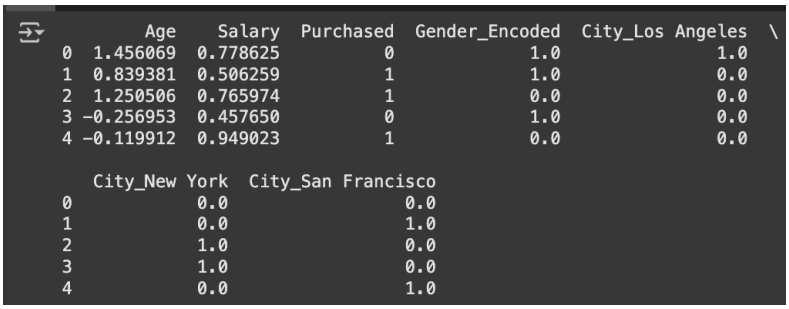
Outlier Detection and Treatment using IQR

#Pros: Simple and effective for mild outliers.

#Cons: May overly reduce variation if there are many extreme outliers.

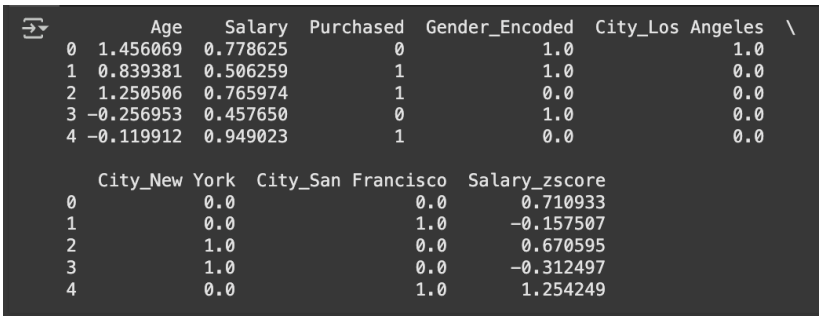
```
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded
Q1 = df_encoded_copy1['Salary'].quantile(0.25)
Q3 = df_encoded_copy1['Salary'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
```

```
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['Salary'] = np.where(df_encoded_copy1['Salary'] > upper_bound, upper_bound,
                    np.where(df_encoded_copy1['Salary'] < lower_bound, lower_bound,
df_encoded_copy1['Salary']))
print(df_encoded_copy1.head())
```



```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0  1.456069  0.778625          0             1.0               1.0
1  0.839381  0.506259          1             1.0               0.0
2  1.250506  0.765974          1             0.0               0.0
3 -0.256953  0.457650          0             1.0               0.0
4 -0.119912  0.949023          1             0.0               0.0

   City_New York  City_San Francisco
0            0.0                 0.0
1            0.0                 1.0
2            1.0                 0.0
3            1.0                 0.0
4            0.0                 1.0
```

```
#Removing Outliers
# Z-score method
#Pros: Good for normally distributed data.
#Cons: Not suitable for non-normal data; may miss outliers in skewed distributions.
df_encoded_copy2['Salary_zscore'] = stats.zscore(df_encoded_copy2['Salary'])
df_encoded_copy2['Salary'] = np.where(df_encoded_copy2['Salary_zscore'].abs() > 3, np.nan,
df_encoded_copy2['Salary'])  # Replace outliers with NaN
print(df_encoded_copy2.head())
```



```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0  1.456069  0.778625          0             1.0               1.0
1  0.839381  0.506259          1             1.0               0.0
2  1.250506  0.765974          1             0.0               0.0
3 -0.256953  0.457650          0             1.0               0.0
4 -0.119912  0.949023          1             0.0               0.0

   City_New York  City_San Francisco  Salary_zscore
0            0.0                 0.0       0.710933
1            0.0                 1.0      -0.157507
2            1.0                 0.0       0.670595
3            1.0                 0.0      -0.312497
4            0.0                 1.0       1.254249
```

```
#Removing Outliers
# Median replacement for outliers
#Pros: Keeps distribution shape intact, useful when capping isn't feasible.
#Cons: May distort data if outliers represent real phenomena.
df_encoded_copy3['Salary_zscore'] = stats.zscore(df_encoded_copy3['Salary'])
median_salary = df_encoded_copy3['Salary'].median()
df_encoded_copy3['Salary'] = np.where(df_encoded_copy3['Salary_zscore'].abs() > 3,
```

median_salary, df_encoded_copy3['Salary'])

print(df_encoded_copy3.head())

```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0  1.456069  0.778625          0             1.0               1.0
1  0.839381  0.506259          1             1.0               0.0
2  1.250506  0.765974          1             0.0               0.0
3 -0.256953  0.457650          0             1.0               0.0
4 -0.119912  0.949023          1             0.0               0.0

   City_New York  City_San Francisco  Salary_zscore
0            0.0                 0.0       0.710933
1            0.0                 1.0      -0.157507
2            1.0                 0.0       0.670595
3            1.0                 0.0      -0.312497
4            0.0                 1.0       1.254249
```

➔ **Diabetes**

import pandas as pd

file_path = '/content/Dataset of Diabetes .csv'

df = pd.read_csv(file_path)

print("Sample data:")

print(df.head())

print("\n")

```
Sample data:
    ID  No_Pation Gender  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0  502      17975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1  735      34221      M   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2  420      47975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
3  680      87656      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
4  504      34223      M   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4

    BMI CLASS
0  24.0     N
1  23.0     N
2  24.0     N
3  24.0     N
4  21.0     N
```

#1. Which columns in the dataset had missing values? How did you handle them ?

print(df.isnull().sum())

```
ID           0
No_Pation    0
Gender       0
AGE          0
Urea         0
Cr           0
HbA1c        0
Chol         0
TG           0
HDL          0
LDL          0
VLDL         0
BMI          0
CLASS        0
dtype: int64
```

#2. Which categorical columns did you identify in the dataset? How did you encode them ?

print(df.info)

```
<bound method DataFrame.info of       ID  No_Pation Gender  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0    502      17975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1    735      34221      M   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2    420      47975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
3    680      87656      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
4    504      34223      M   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4
..   ...        ...    ...  ...   ...  ..    ...   ...  ...  ...  ...   ...
995  200     454317      M   71  11.0  97    7.0   7.5  1.7  1.2  1.8   0.6
996  671     876534      M   31   3.0  60   12.3   4.1  2.2  0.7  2.4  15.4
997  669      87654      M   30   7.1  81    6.7   4.1  1.1  1.2  2.4   8.1
998   99      24004      M   38   5.8  59    6.7   5.3  2.0  1.6  2.9  14.0
999  248      24054      M   54   5.0  67    6.9   3.8  1.7  1.1  3.0   0.7

      BMI CLASS
0    24.0     N
1    23.0     N
2    24.0     N
3    24.0     N
4    21.0     N
..    ...   ...
995  30.0     Y
996  37.2     Y
997  27.4     Y
998  40.5     Y
999  33.0     Y

[1000 rows x 14 columns]>
```

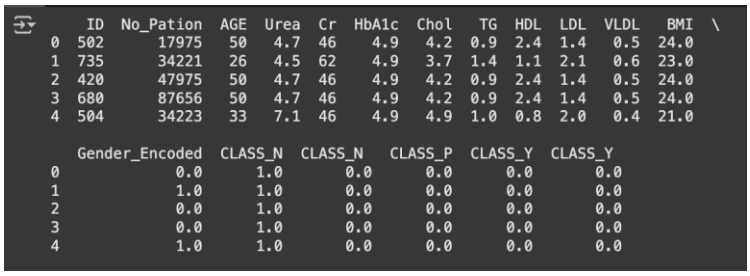
```
# Clean the Gender column: Convert all values to uppercase
df["Gender"] = df["Gender"].str.upper()
# Initialize OrdinalEncoder for Gender
ordinal_encoder = OrdinalEncoder(categories=[["F", "M"]], handle_unknown="use_encoded_value",
unknown_value=-1)
# Fit and transform the Gender column
df["Gender_Encoded"] = ordinal_encoder.fit_transform(df[["Gender"]])
# Initialize OneHotEncoder for CLASS
onehot_encoder = OneHotEncoder()
# Fit and transform the CLASS column
encoded_data = onehot_encoder.fit_transform(df[["CLASS"]])
# Convert the sparse matrix to a dense array
encoded_array = encoded_data.toarray()
# Convert to DataFrame for better visualization
encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df, encoded_df], axis=1)
```

Drop the original Gender and CLASS columns

df_encoded.drop("Gender", axis=1, inplace=True)

df_encoded.drop("CLASS", axis=1, inplace=True)

print(df_encoded.head())



	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI
0	502	17975	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0
1	735	34221	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0
2	420	47975	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0
3	680	87656	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0
4	504	34223	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0

	Gender_Encoded	CLASS_N	CLASS_N	CLASS_P	CLASS_Y	CLASS_Y
0	0.0	1.0	0.0	0.0	0.0	0.0
1	1.0	1.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0	0.0	0.0
4	1.0	1.0	0.0	0.0	0.0	0.0

➔ **ADULT INCOME DATA**

import pandas as pd

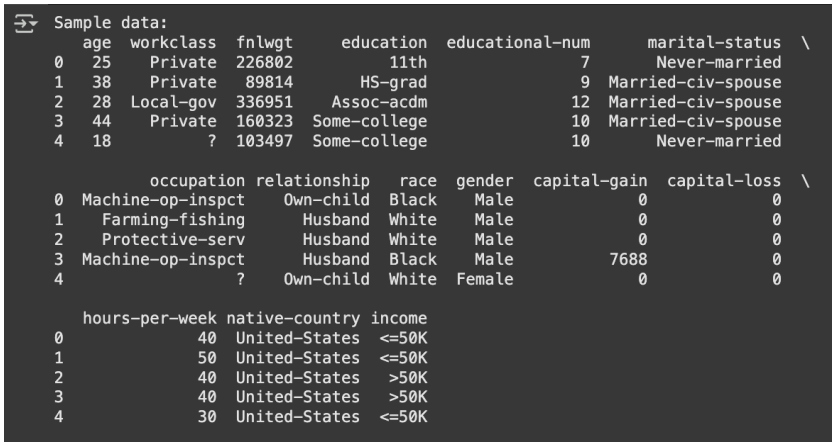
file_path = '/content/adult.csv'

df = pd.read_csv(file_path)

print("Sample data:")

print(df.head())

print("\n")



Sample data:

	age	workclass	fnlwgt	education	educational-num	marital-status
0	25	Private	226802	11th	7	Never-married
1	38	Private	89814	HS-grad	9	Married-civ-spouse
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse
3	44	Private	160323	Some-college	10	Married-civ-spouse
4	18	?	103497	Some-college	10	Never-married

	occupation	relationship	race	gender	capital-gain	capital-loss
0	Machine-op-inspct	Own-child	Black	Male	0	0
1	Farming-fishing	Husband	White	Male	0	0
2	Protective-serv	Husband	White	Male	0	0
3	Machine-op-inspct	Husband	Black	Male	7688	0
4	?	Own-child	White	Female	0	0

	hours-per-week	native-country	income
0	40	United-States	<=50K
1	50	United-States	<=50K
2	40	United-States	>50K
3	40	United-States	>50K
4	30	United-States	<=50K

print(df.isnull().sum())

```
age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

print(df.info)

```
<bound method DataFrame.info of       age     workclass  fnlwgt     education  educational-num  \
0      25       Private  226802          11th               7
1      38       Private   89814       HS-grad               9
2      28     Local-gov  336951    Assoc-acdm              12
3      44       Private  160323  Some-college              10
4      18             ?  103497  Some-college              10
...   ...           ...     ...           ...             ...
48837  27       Private  257302    Assoc-acdm              12
48838  40       Private  154374       HS-grad               9
48839  58       Private  151910       HS-grad               9
48840  22       Private  201490       HS-grad               9
48841  52  Self-emp-inc  287927       HS-grad               9

          marital-status         occupation relationship   race  gender  \
0          Never-married  Machine-op-inspct    Own-child  Black    Male
1     Married-civ-spouse    Farming-fishing      Husband  White    Male
2     Married-civ-spouse    Protective-serv      Husband  White    Male
3     Married-civ-spouse  Machine-op-inspct      Husband  Black    Male
4          Never-married                  ?    Own-child  White  Female
...                  ...                ...          ...    ...     ...
48837  Married-civ-spouse       Tech-support         Wife  White  Female
48838  Married-civ-spouse  Machine-op-inspct      Husband  White    Male
48839             Widowed       Adm-clerical    Unmarried  White  Female
48840       Never-married       Adm-clerical    Own-child  White    Male
48841  Married-civ-spouse    Exec-managerial         Wife  White  Female

       capital-gain  capital-loss  hours-per-week native-country income
0                 0             0              40  United-States  <=50K
1                 0             0              50  United-States  <=50K
2                 0             0              40  United-States   >50K
3              7688             0              40  United-States   >50K
4                 0             0              30  United-States  <=50K
...             ...           ...             ...            ...    ...
48837             0             0              38  United-States  <=50K
48838             0             0              40  United-States   >50K
48839             0             0              40  United-States  <=50K
48840             0             0              20  United-States  <=50K
48841         15024             0              40  United-States   >50K

[48842 rows x 15 columns]>
```

Encode binary categorical columns (e.g., gender) using OrdinalEncoder

binary_columns = ['gender']

Initialize OrdinalEncoder for binary columns

ordinal_encoder = OrdinalEncoder(categories=[["Female", "Male"]],

handle_unknown="use_encoded_value", unknown_value=-1)

df[binary_columns] = ordinal_encoder.fit_transform(df[binary_columns])

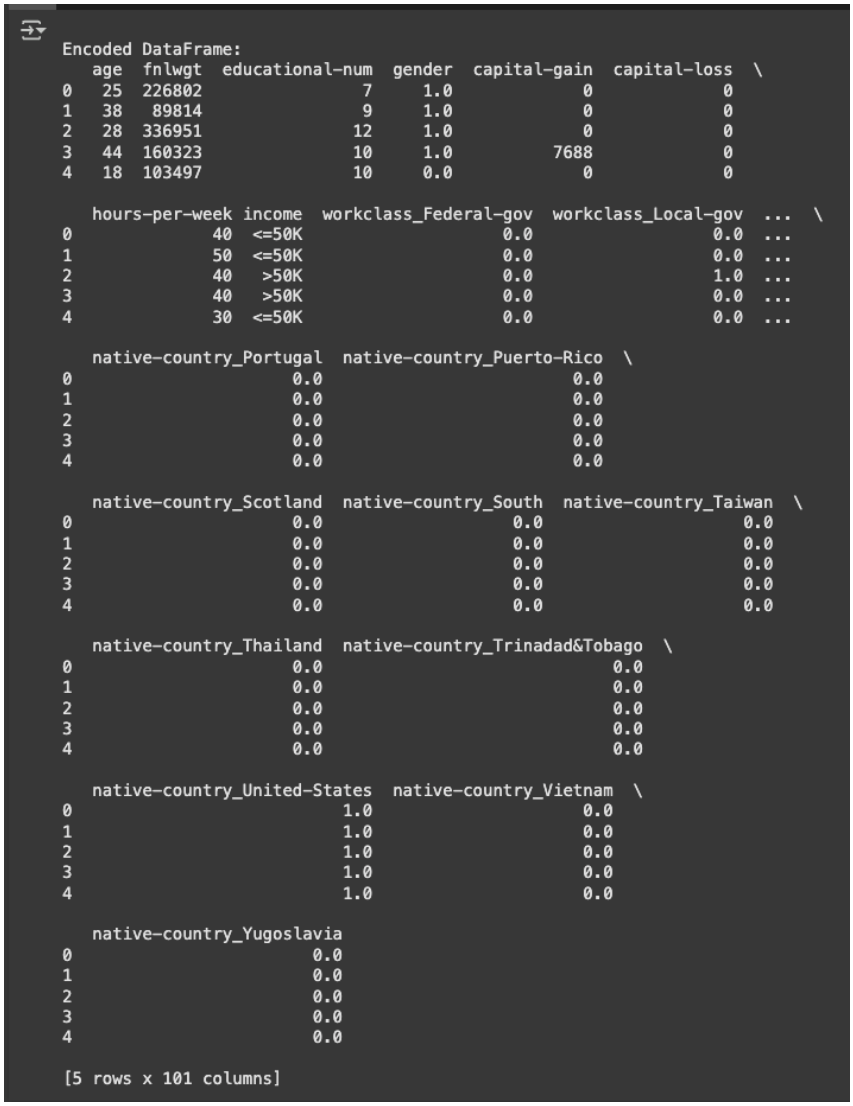
Encode multi-category columns using OneHotEncoder

multi_category_columns = ['workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race', 'native-country']

onehot_encoder = OneHotEncoder(sparse_output=False, drop='first') # Drop first column to avoid

multicollinearity

```
encoded_data = onehot_encoder.fit_transform(df[multi_category_columns])
# Convert encoded data to DataFrame
encoded_df = pd.DataFrame(encoded_data,
columns=onehot_encoder.get_feature_names_out(multi_category_columns))
# Concatenate encoded data with the original DataFrame
df_encoded = pd.concat([df.drop(multi_category_columns, axis=1), encoded_df], axis=1)
# Display the encoded DataFrame
print("\nEncoded DataFrame:")
print(df_encoded.head())
```

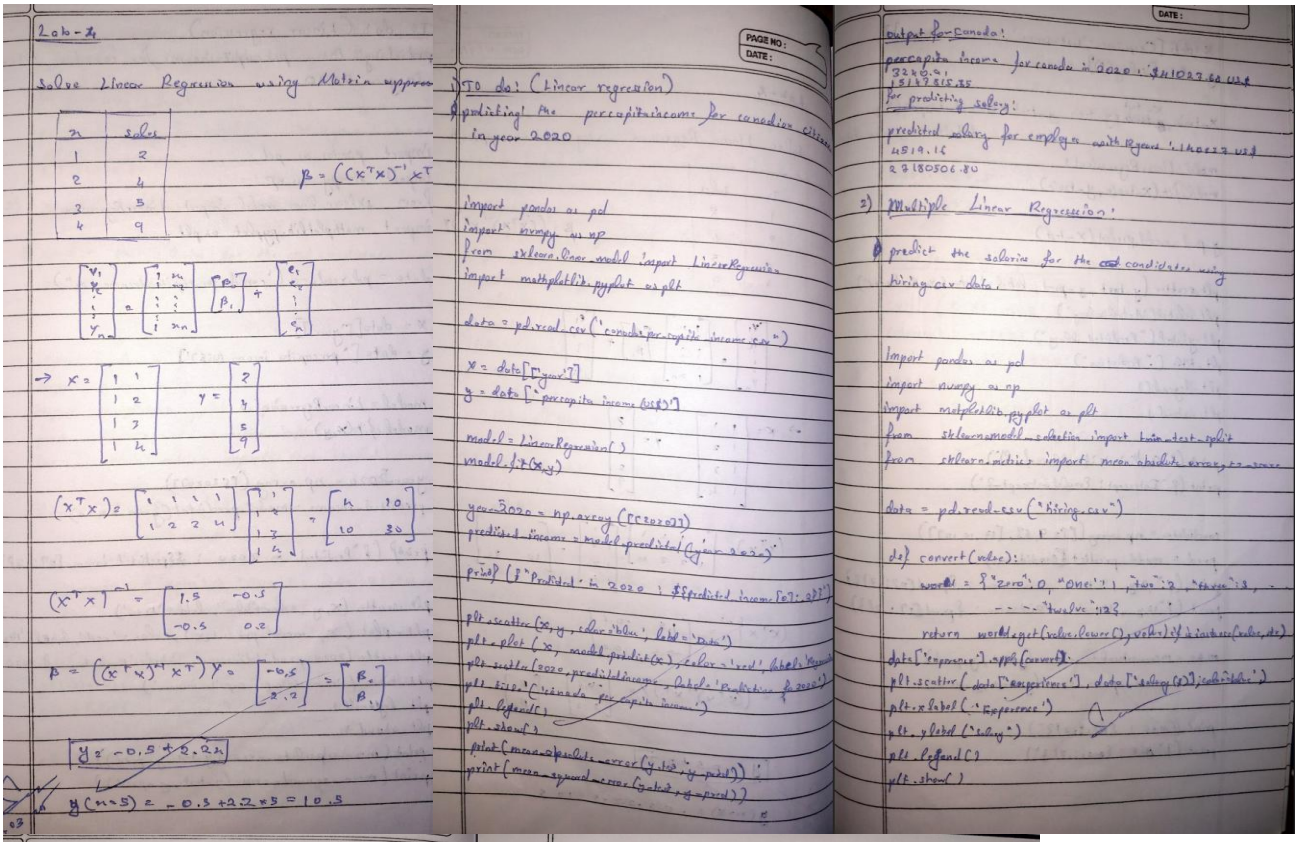


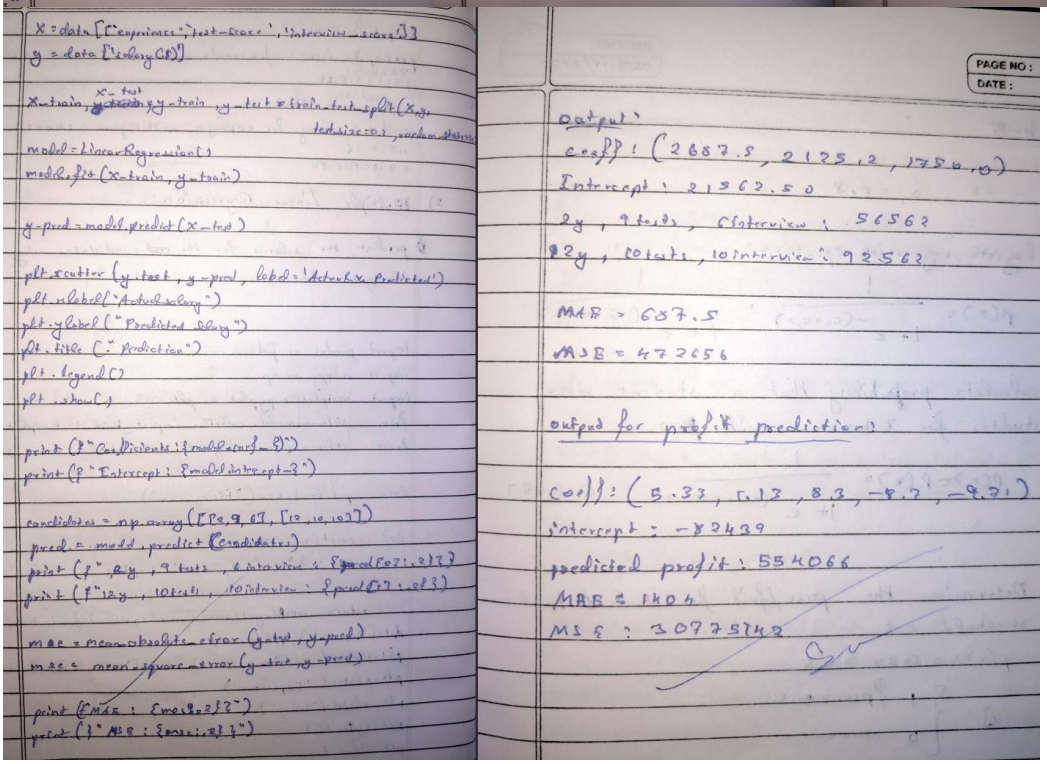
```
Encoded DataFrame:
   age  fnlwgt  educational-num  gender  capital-gain  capital-loss  \
0   25  226802                7     1.0             0             0
1   38   89814                9     1.0             0             0
2   28  336951               12     1.0             0             0
3   44  160323               10     1.0          7688             0
4   18  103497               10     0.0             0             0

   hours-per-week income  workclass_Federal-gov  workclass_Local-gov  ...  \
0              40  <=50K                    0.0                  0.0  ...
1              50  <=50K                    0.0                  0.0  ...
2              40   >50K                    0.0                  1.0  ...
3              40   >50K                    0.0                  0.0  ...
4              30  <=50K                    0.0                  0.0  ...

   native-country_Portugal  native-country_Puerto-Rico  \
0                      0.0                         0.0
1                      0.0                         0.0
2                      0.0                         0.0
3                      0.0                         0.0
4                      0.0                         0.0

   native-country_Scotland  native-country_South  native-country_Taiwan  \
0                      0.0                   0.0                    0.0
1                      0.0                   0.0                    0.0
2                      0.0                   0.0                    0.0
3                      0.0                   0.0                    0.0
4                      0.0                   0.0                    0.0

   native-country_Thailand  native-country_Trinadad&Tobago  \
0                      0.0                             0.0
1                      0.0                             0.0
2                      0.0                             0.0
3                      0.0                             0.0
4                      0.0                             0.0

   native-country_United-States  native-country_Vietnam  \
0                           1.0                     0.0
1                           1.0                     0.0
2                           1.0                     0.0
3                           1.0                     0.0
4                           1.0                     0.0

   native-country_Yugoslavia
0                         0.0
1                         0.0
2                         0.0
3                         0.0
4                         0.0

[5 rows x 101 columns]
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots





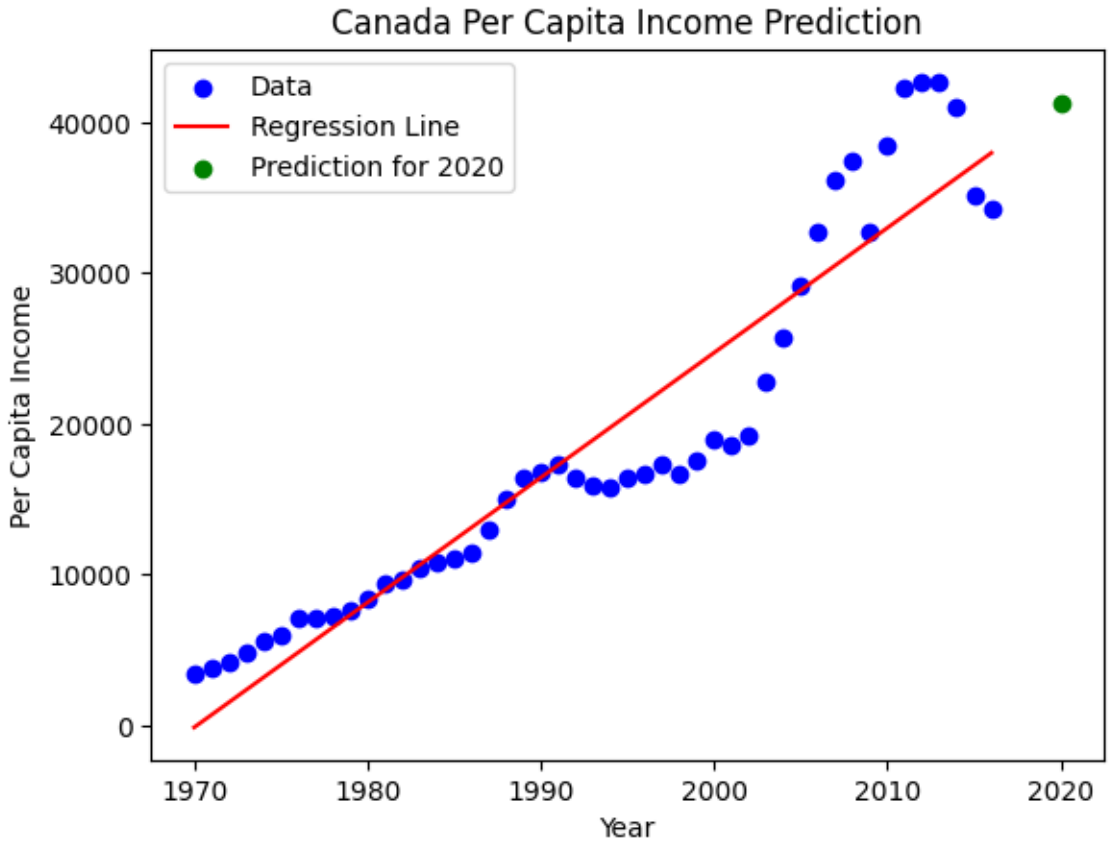
Code:

Linear Regression:

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Step 1: Load the dataset
data = pd.read_csv('/content/sample_data/canada_per_capita_income.csv')

# Step 2: Data preprocessing
# Checking for any missing values or cleaning if necessary
print(data.isnull().sum())  # To check for any missing values

# Assuming the data has columns "Year" and "PerCapitaIncome"
# If there are any missing values, you can fill them or drop rows accordingly
data.dropna(inplace=True)

# Step 3: Train a regression model
X = data[['year']]
y = data['per capita income (US$)']

# Initialize and fit the model
model = LinearRegression()
model.fit(X, y)

# Step 4: Predict the per capita income for the year 2020
year_2020 = np.array([[2020]])
predicted_income = model.predict(year_2020)

print(f"Predicted per capita income for Canada in 2020: ${predicted_income[0]:.2f}")

# Step 5: (Optional) Visualizing the data and prediction
```

```
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X, model.predict(X), color='red', label='Regression Line')
plt.scatter(2020, predicted_income, color='green', label='Prediction for 2020')
plt.xlabel('Year')
plt.ylabel('Per Capita Income')
plt.title('Canada Per Capita Income Prediction')
plt.legend()
plt.show()
```

```
year                    0
per capita income (US$)    0
dtype: int64
```



 # Step 1: Load the dataset for Salary prediction

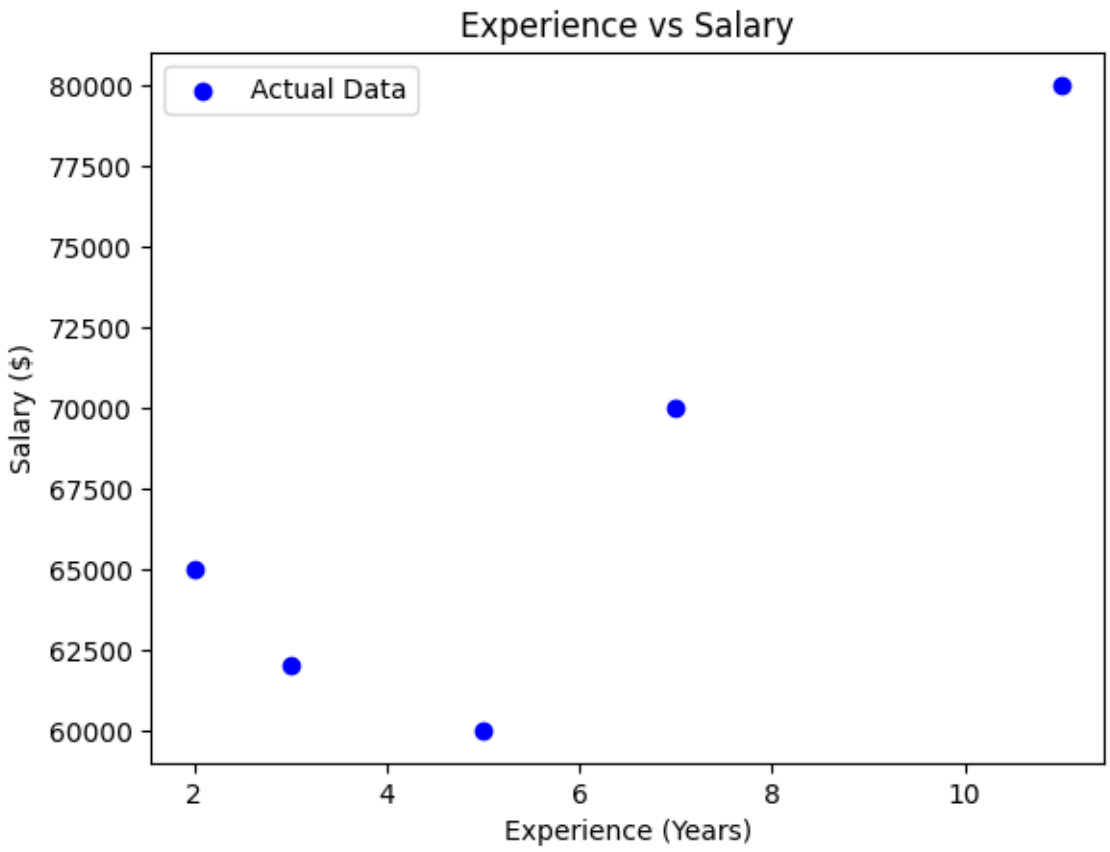
```
salary_data = pd.read_csv('/content/sample_data/salary.csv')

# Step 2: Data preprocessing
print(salary_data.isnull().sum())

# Assuming the data has columns "YearsExperience" and "Salary"
# salary_data.dropna(inplace=True)
salary_data.fillna(salary_data['YearsExperience'].mean(), inplace=True)



# Step 3: Train a regression model
X_salary = salary_data[['YearsExperience']]
y_salary = salary_data['Salary']

# Initialize and fit the model
salary_model = LinearRegression()
salary_model.fit(X_salary, y_salary)

# Step 4: Predict the salary for an employee with 12 years of experience
years_of_experience = np.array([[12]])
predicted_salary = salary_model.predict(years_of_experience)

print(f"Predicted salary for an employee with 12 years of experience: ${predicted_salary[0]:.2f}")

# Step 5: Visualizing the data and prediction
plt.scatter(X_salary, y_salary, color='blue', label='Data')
plt.plot(X_salary, salary_model.predict(X_salary), color='red', label='Regression Line')
plt.scatter(12, predicted_salary, color='green', label='Prediction for 12 years of experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Employee Salary Prediction based on Experience')
plt.legend()
```

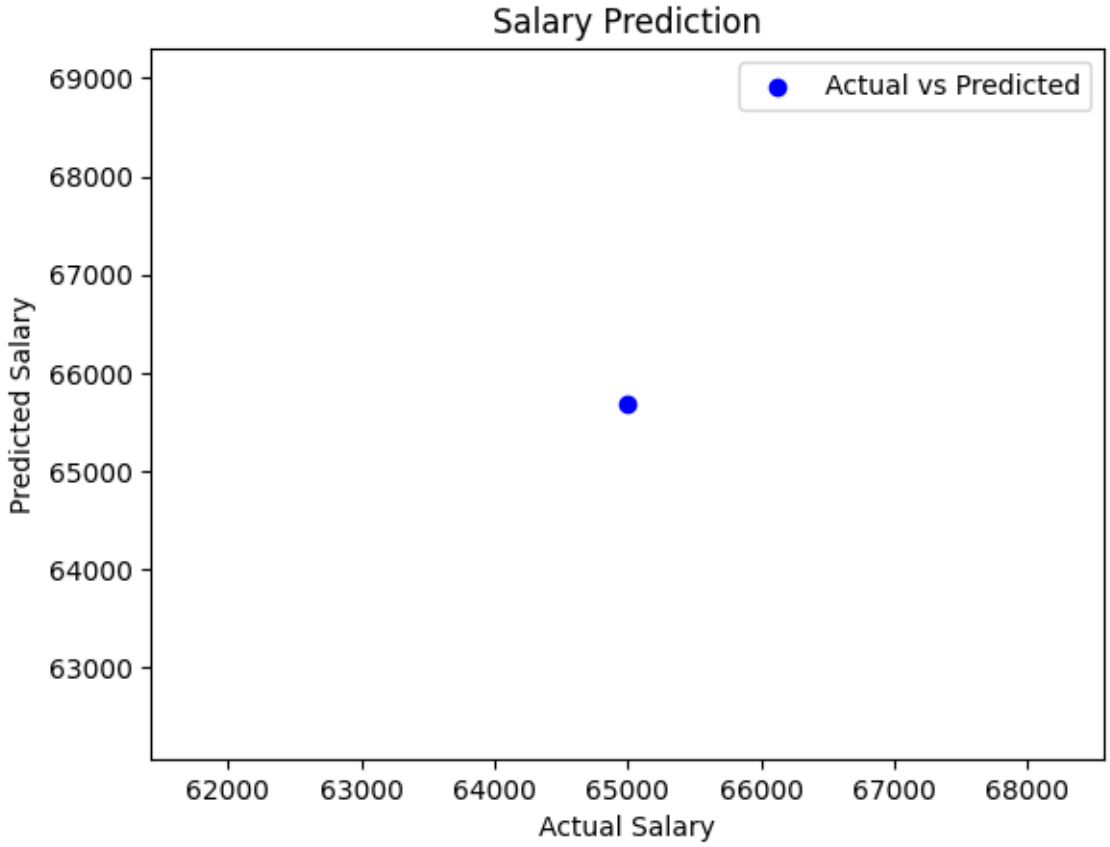
```
plt.show()
```

```
YearsExperience    2
Salary             0
dtype: int64
Predicted salary for an employee with 12 years of experience: $139980.89
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv("/content/sample_data/hiring.csv")

# Function to convert experience from words to numbers
def convert_experience(value):
    word_to_num = {"zero": 0, "one": 1, "two": 2, "three": 3, "four": 4, "five": 5, "six": 6,
                   "seven": 7, "eight": 8, "nine": 9, "ten": 10, "eleven": 11, "twelve": 12}
    return word_to_num.get(value.lower(), value) if isinstance(value, str) else value

# Apply conversion
data['experience'] = data['experience'].apply(convert_experience)

# Handle missing values by removing rows with NaN
data = data.dropna()

# Convert all columns to numeric
data = data.astype(float)
```

```
# Distribution plot visualization
plt.scatter(data['experience'], data['salary($)'], color='blue', label='Actual Data')
plt.xlabel("Experience (Years)")
plt.ylabel("Salary ($)")
plt.title("Experience vs Salary")
plt.legend()
plt.show()

# Relationship between variables
X = data[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y = data['salary($)']

# Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict the results
y_pred = model.predict(X_test)

# Visualize prediction
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel("Actual Salary")
plt.ylabel("Predicted Salary")
plt.title("Salary Prediction")
plt.legend()
plt.show()

# Check values of coefficients and intercept
print(f"Coefficients: {model.coef_}")
```

```
print(f"Intercept: {model.intercept_}")

# Predict salary for given candidates
candidates = np.array([[2, 9, 6], [12, 10, 10]])
salary_predictions = model.predict(candidates)
print(f"Predicted salary for 2 yrs experience, 9 test score, 6 interview score:
{salary_predictions[0]:.2f} US$")
print(f"Predicted salary for 12 yrs experience, 10 test score, 10 interview score:
{salary_predictions[1]:.2f} US$")

# Calculate errors
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")
```


Experience vs Salary
Actual Data
80000
77500
75000
72500
70000
67500
65000
62500
60000
Salary ($)
2
4
6
8
10
Experience (Years)


Salary Prediction
Actual vs Predicted
69000
68000
67000
66000
65000
64000
63000
Predicted Salary
62000
63000
64000
65000
66000
67000
68000
Actual Salary

Coefficients: [2687.5 2125. 1750.]

Intercept: 21562.500000000022

Predicted salary for 2 yrs experience, 9 test score, 6 interview score: 56562.50 US$

Predicted salary for 12 yrs experience, 10 test score, 10 interview score: 92562.50 US$

Mean Absolute Error (MAE): 687.50

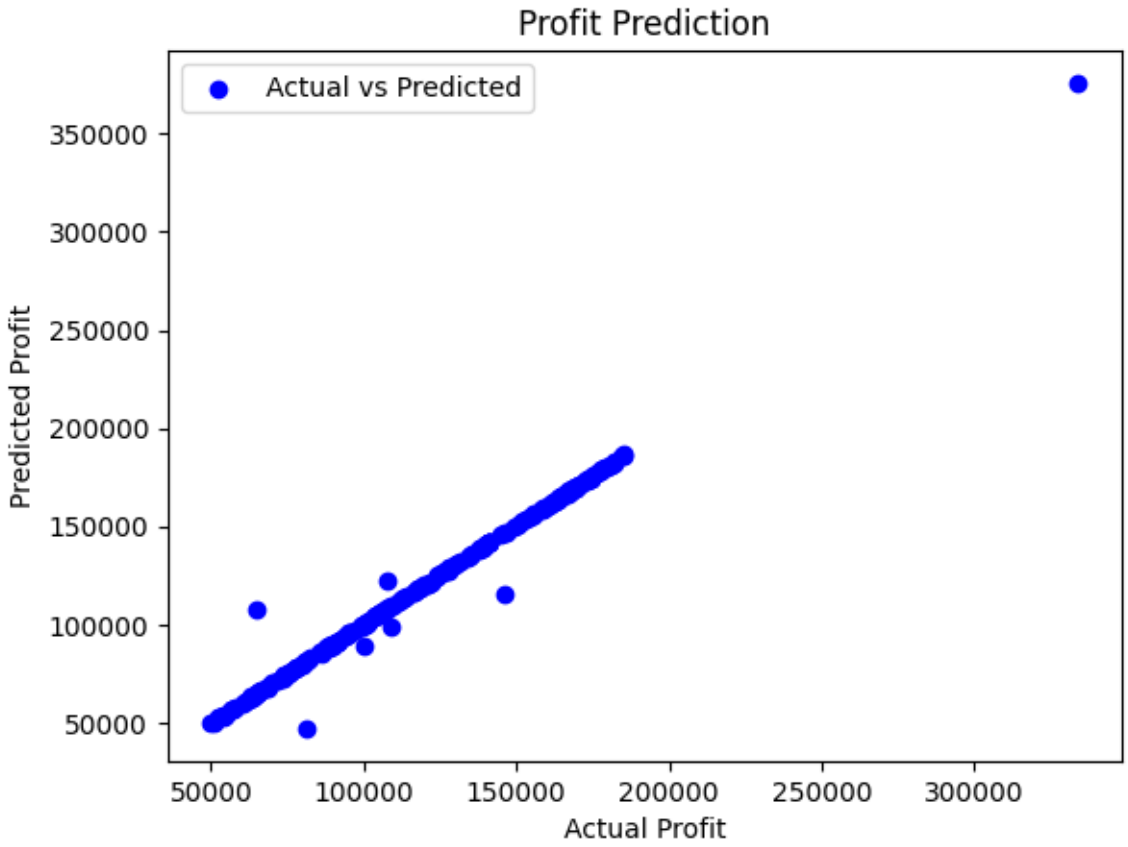
Mean Squared Error (MSE): 472656.25

R-squared (R2) Score: nan

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv("/content/sample_data/1000_Companies.csv")

# Handle missing values by removing rows with NaN
data = data.dropna()

# Encode categorical variable (State) using OneHotEncoder
encoder = OneHotEncoder(drop='first', sparse_output=False)
state_encoded = encoder.fit_transform(data[['State']])
state_encoded_df = pd.DataFrame(state_encoded,
columns=encoder.get_feature_names_out(['State']))

# Concatenate encoded state data with original dataset
data = pd.concat([data.drop(['State'], axis=1), state_encoded_df], axis=1)

# Define independent (X) and dependent (y) variables
X = data[['R&D Spend', 'Administration', 'Marketing Spend'] + list(state_encoded_df.columns)]
```

```
y = data['Profit']

# Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict the results
y_pred = model.predict(X_test)

# Visualize prediction
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel("Actual Profit")
plt.ylabel("Predicted Profit")
plt.title("Profit Prediction")
plt.legend()
plt.show()

# Check values of coefficients and intercept
print(f"Coefficients: {model.coef_}")
print(f"Intercept: {model.intercept_}")

# Predict profit for given candidate dynamically
state_names = encoder.get_feature_names_out(['State'])
florida_encoded = (state_names == "State_Florida").astype(int)
candidate_features = np.array([91694.48, 515841.3, 11931.24] + list(florida_encoded)).reshape(1, -
1)
profit_prediction = model.predict(candidate_features)
print(f"Predicted profit for given candidate: {profit_prediction[0]:.2f} US$")
```

```
# Calculate errors
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")
```



Coefficients: [5.33045605e-01 1.13893831e+00 8.30755037e-02 -8.74491486e+02
 -9.71337988e+01]

Intercept: -82439.15560711118

Predicted profit for given candidate: 554066.30 US$

Mean Absolute Error (MAE): 1404.44

Mean Squared Error (MSE): 30775142.86

R-squared (R2) Score: 0.98

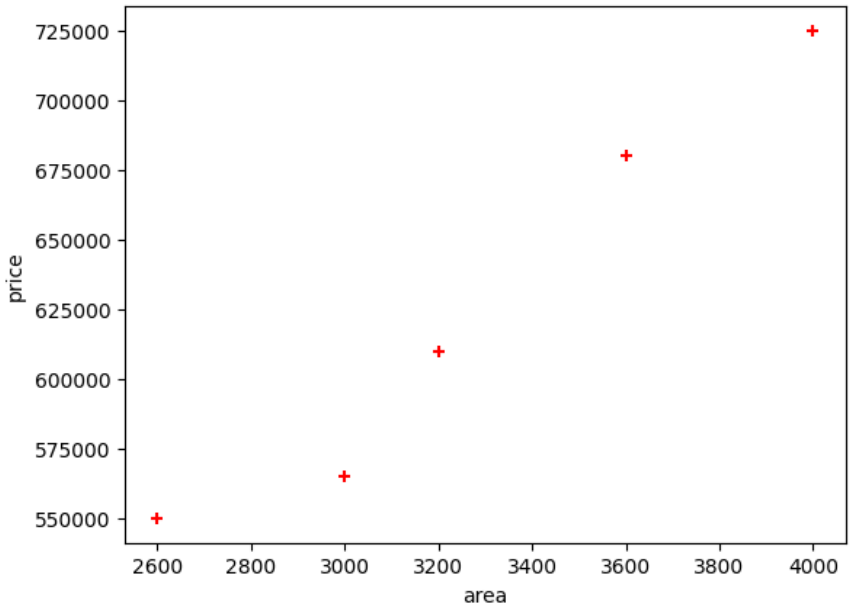
```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('/content/sample_data/housing_area_price.csv')

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')

new_df = df.drop('price',axis='columns')
new_df

price = df.price
price

# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df,price)

"""(1) Predict price of a home with area = 3300 sqr ft"""

reg.predict([[3300]])

reg.coef_

reg.intercept_

"""Y = m * X + b (m is coefficient and b is intercept)"""
```

"""(1) Predict price of a home with area = 5000 sqr ft"""

reg.predict([[5000]])

Out[59]:

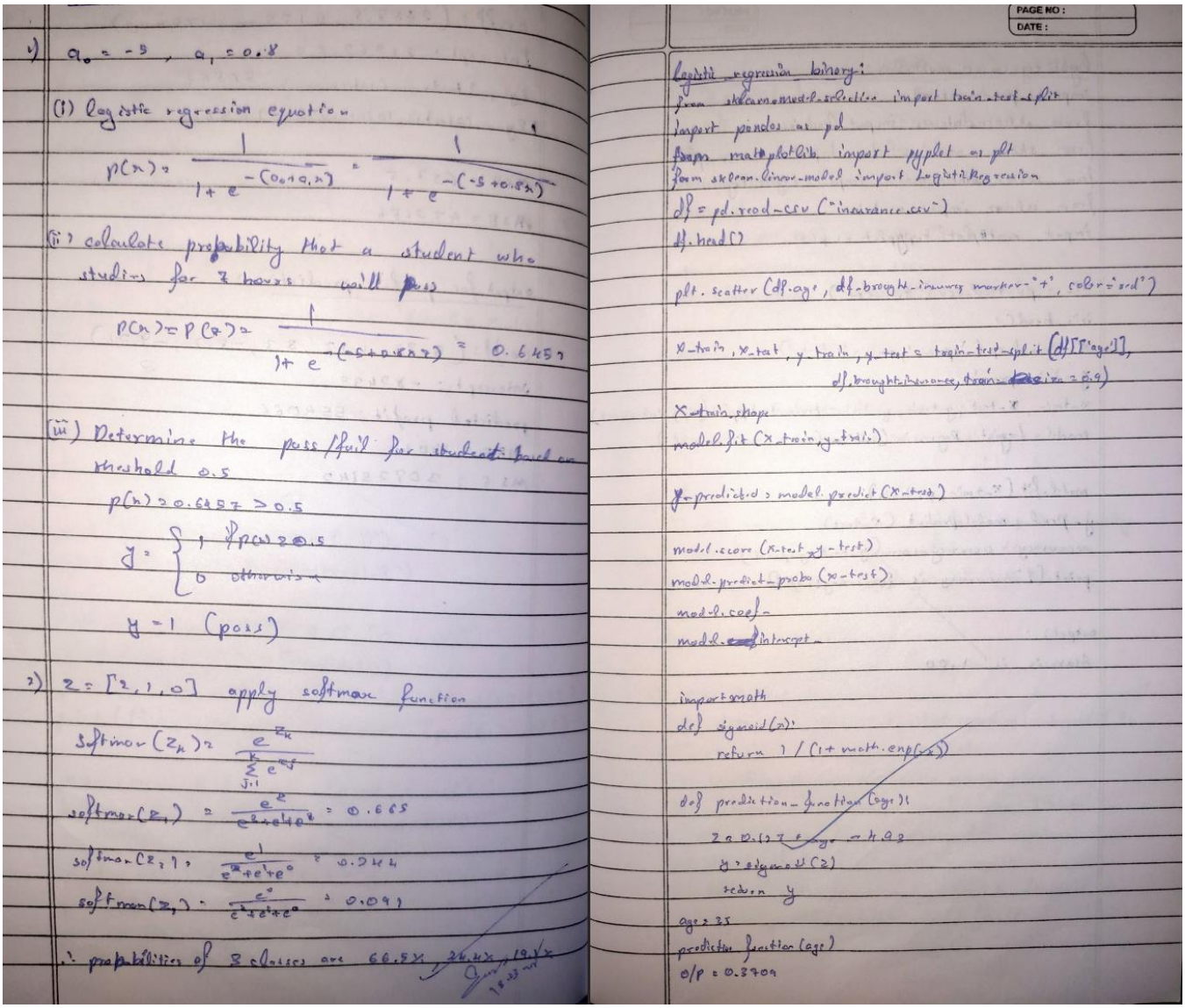
array([859554.79452055])

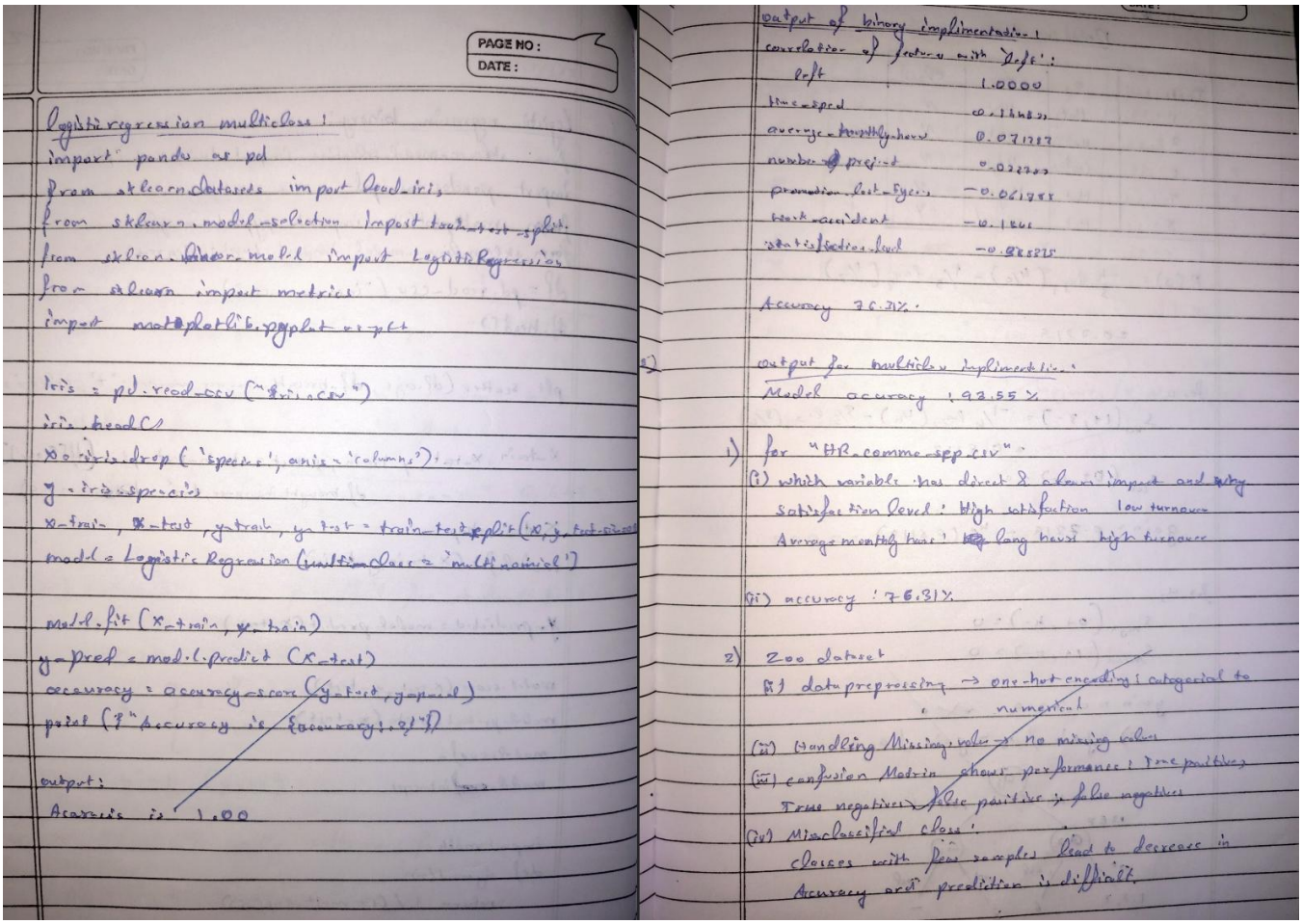


Program 4

Build Logistic Regression Model for a given dataset

Screenshots





Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

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

# Display the first few rows
print(data.head())
```

```
# Check correlation of numerical variables with 'left'
correlation = data.corr(numeric_only=True)['left'].sort_values(ascending=False)

# Plot correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(data.corr(numeric_only=True), annot=True, cmap='Blues', fmt='.2f')
plt.title('Correlation of Features with Employee Retention (Left)')
plt.show()

# Print correlation results
print("Correlation of features with 'left':\n", correlation)

# Plot bar chart for salary and retention
plt.figure(figsize=(8, 5))
sns.countplot(x='salary', hue='left', data=data, palette='pastel')
plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary Level')
plt.ylabel('Number of Employees')
plt.legend(title='Left', labels=['Stayed', 'Left'])
plt.show()

# Plot bar chart for department and retention
plt.figure(figsize=(12, 6))
sns.countplot(x='Department', hue='left', data=data, palette='pastel')
plt.title('Correlation between Department and Employee Retention')
plt.xlabel('Department')
plt.ylabel('Number of Employees')
plt.xticks(rotation=45)
plt.legend(title='Left', labels=['Stayed', 'Left'])
plt.show()

# Prepare the data for modeling
X = data[['satisfaction_level', 'time_spend_company', 'average_montly_hours', 'salary']]
y = data['left']
```

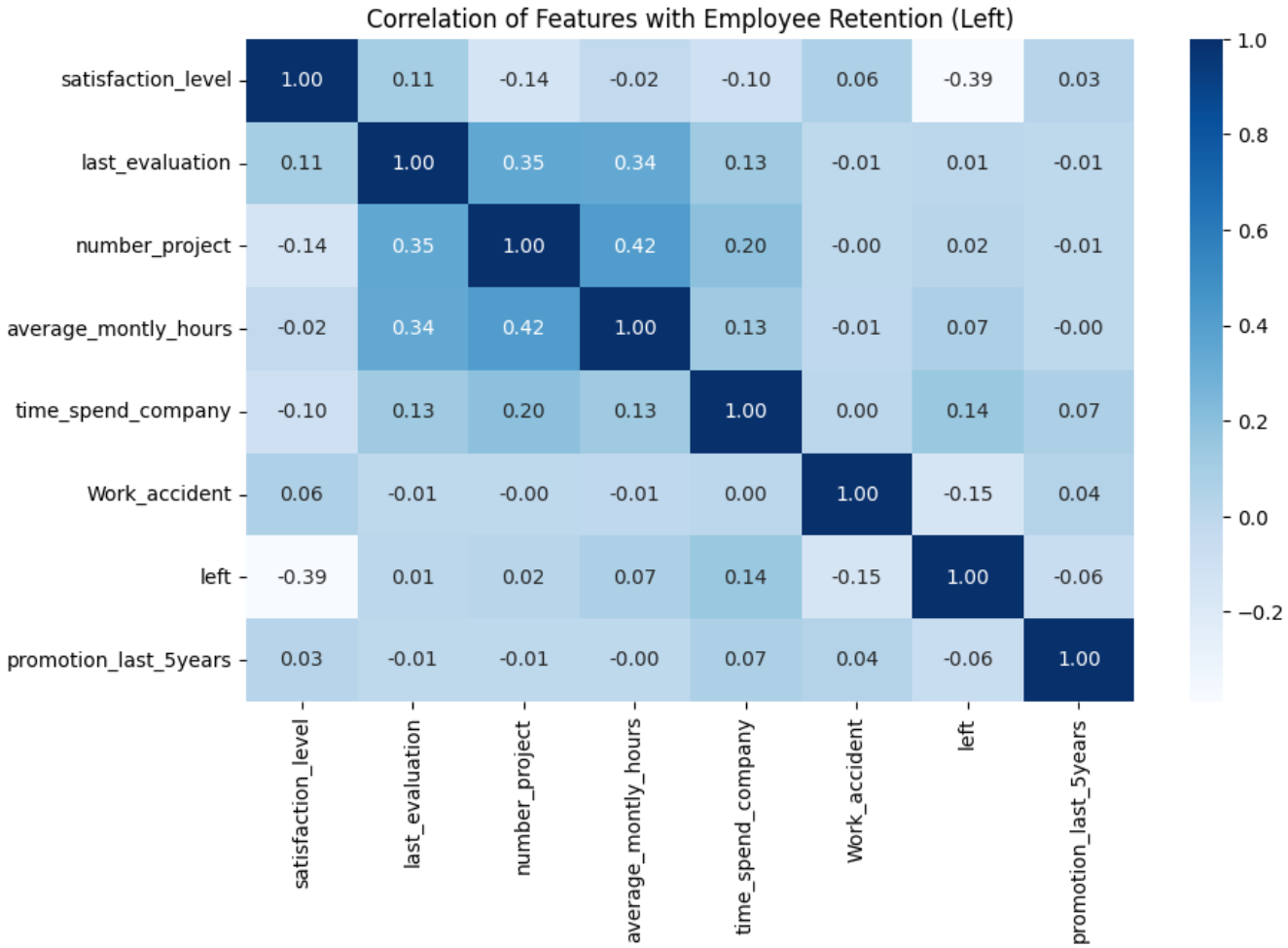
```
# One-hot encode the salary column
X = pd.get_dummies(X, columns=['salary'], drop_first=True)

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

# Build and train the logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict and calculate accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

	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



Correlation of features with 'left':

left 1.000000

time_spend_company 0.144822

average_montly_hours 0.071287

number_project 0.023787

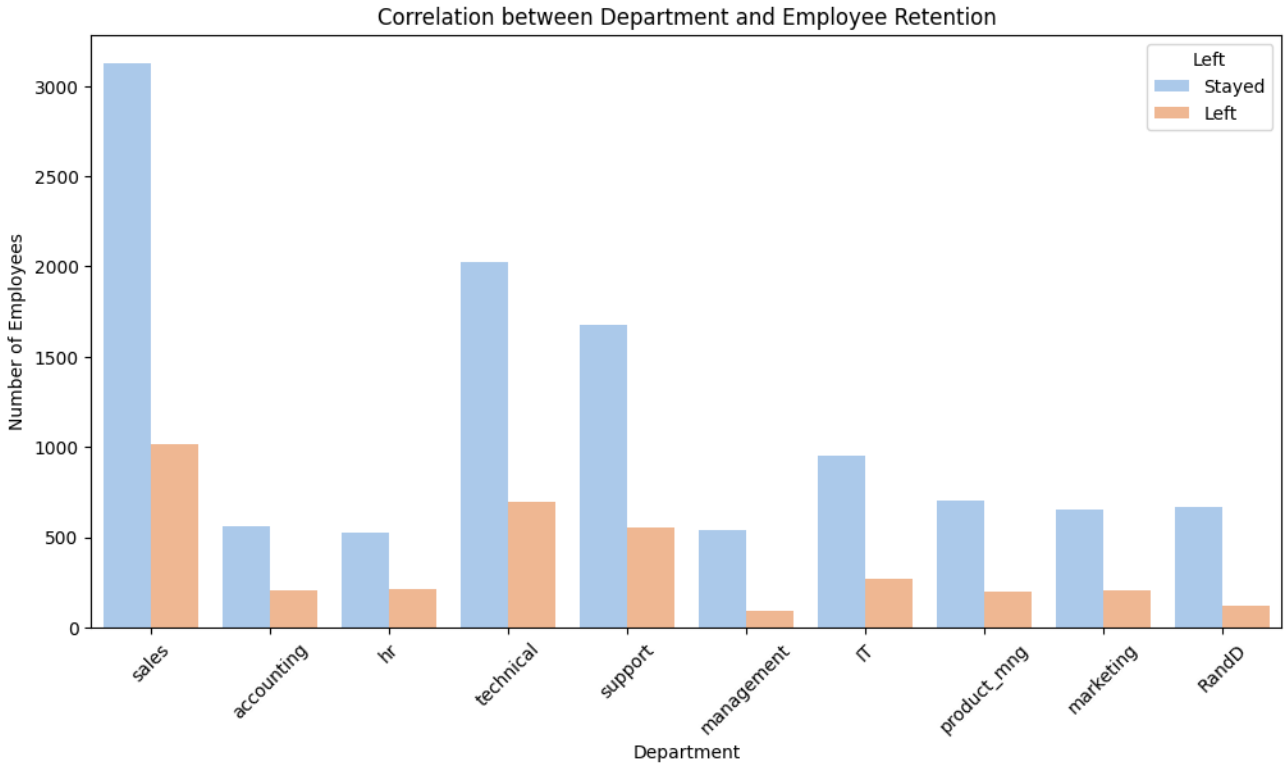
last_evaluation 0.006567

promotion_last_5years -0.061788

Work_accident -0.154622

satisfaction_level -0.388375

Name: left, dtype: float64



Model Accuracy: 76.31%

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the datasets
zoo_data_path = "zoo-data.csv"
class_type_path = "zoo-class-type.csv"

# Load datasets
zoo_data = pd.read_csv(zoo_data_path)
class_type = pd.read_csv(class_type_path)

# Display the first few rows of each dataset
print("Zoo Data Sample:\n", zoo_data.head())
print("\nClass Type Mapping Sample:\n", class_type.head())

# Drop 'animal_name' as it's not required for model training
zoo_data.drop('animal_name', axis=1, inplace=True)

# Prepare feature matrix (X) and target vector (y)
X = zoo_data.drop('class_type', axis=1)
y = zoo_data['class_type']

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

# Build and train the logistic regression model
model = LogisticRegression(max_iter=2000, multi_class='multinomial', solver='lbfgs')
model.fit(X_train, y_train)

# Predict and calculate accuracy
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy: {accuracy * 100:.2f}%")

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_type['Class_Type'], yticklabels=class_type['Class_Type'])
plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')
plt.title('Confusion Matrix for Zoo Dataset')
plt.show()
```

Zoo Data Sample:

	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

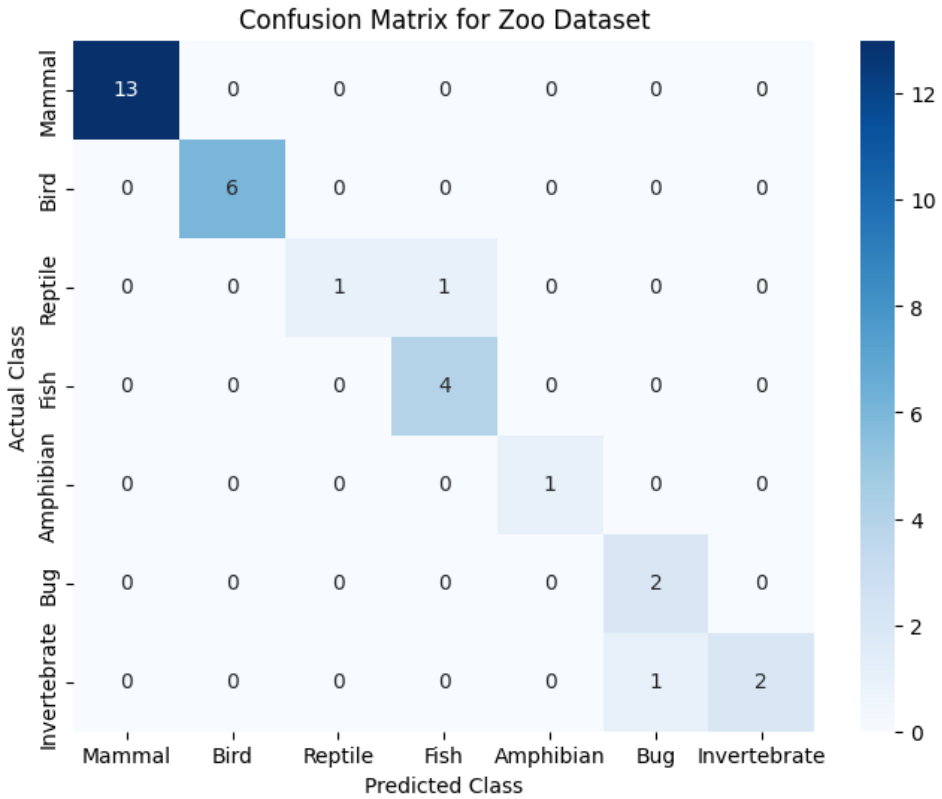
	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	class_type
0	1	1	1	0	0	4	0	0	1	1
1	1	1	1	0	0	4	1	0	1	1
2	1	1	0	0	1	0	1	0	0	4
3	1	1	1	0	0	4	0	0	1	1
4	1	1	1	0	0	4	1	0	1	1

Class Type Mapping Sample:

	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	

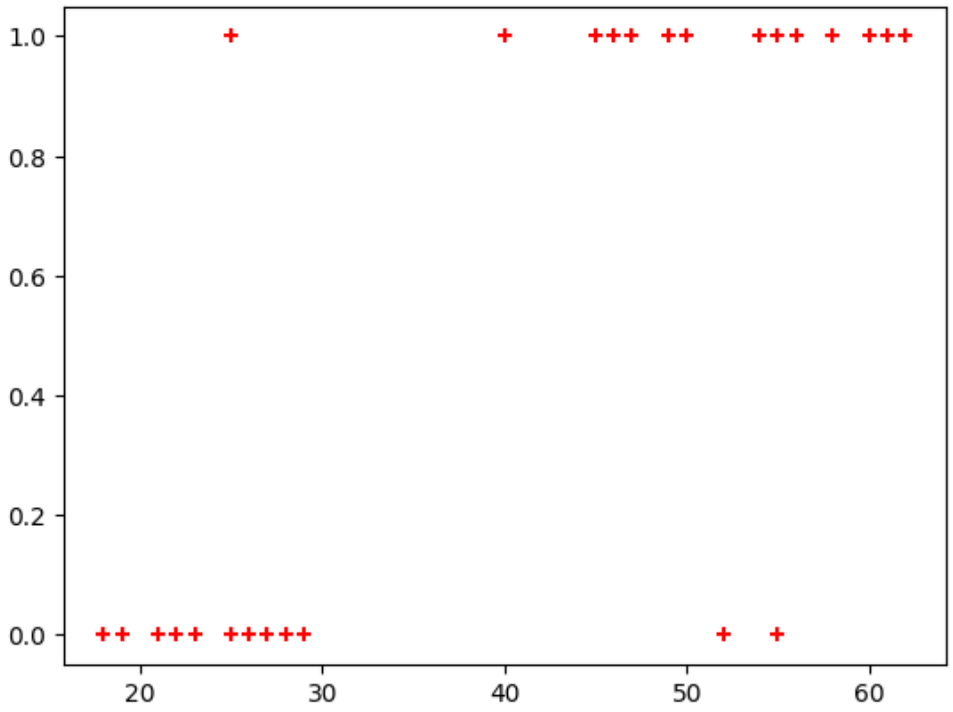
	Animal_Names
0	aardvark, antelope, bear, boar, buffalo, calf,...
1	chicken, crow, dove, duck, flamingo, gull, haw...
2	pitviper, seasnake, slowworm, tortoise, tuatara
3	bass, carp, catfish, chub, dogfish, haddock, h...
4	frog, frog, newt, toad

Model Accuracy: 93.55%



```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv("insurance_data.csv")
df.head()

plt.scatter(df.age,df.bought_insurance,marker='+',color='red')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']],df.bought_insurance,train_size=0.9,random_state=10)
X_train.shape

X_test

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

model.fit(X_train, y_train)

X_test

y_test

y_predicted = model.predict(X_test)
y_predicted

model.score(X_test,y_test)

model.predict_proba(X_test)

y_predicted = model.predict([[60]])
y_predicted

#model.coef_ indicates value of m in y=m*x + b equation
model.coef_
```

```
#model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

def prediction_function(age):
   z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
   y = sigmoid(z)
   return y

age = 35
prediction_function(age)
Out[ ]:
0.3709834769552775
```

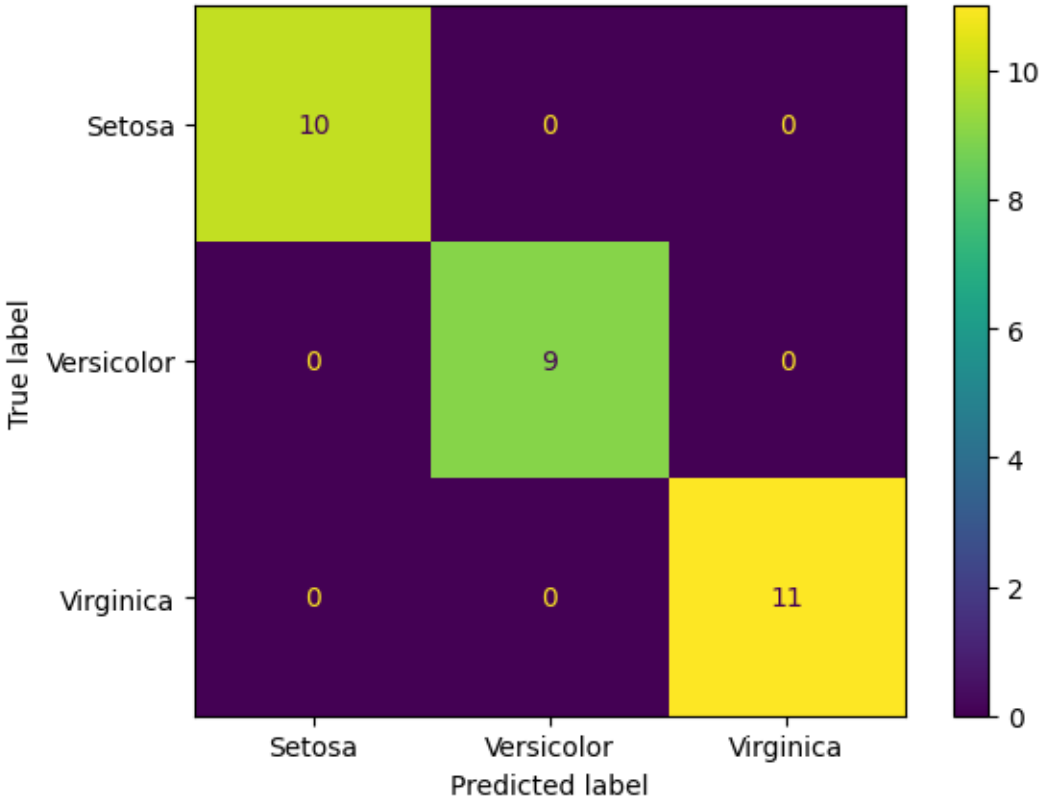


```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt

# Load the Iris dataset
iris =  pd.read_csv("iris.csv")
iris.head()

X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, petal width)
y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

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

# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver
model = LogisticRegression(multi_class='multinomial')

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

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

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)

# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
```

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ["Setosa", "Versicolor", "Virginica"])

cm_display.plot()
plt.show()

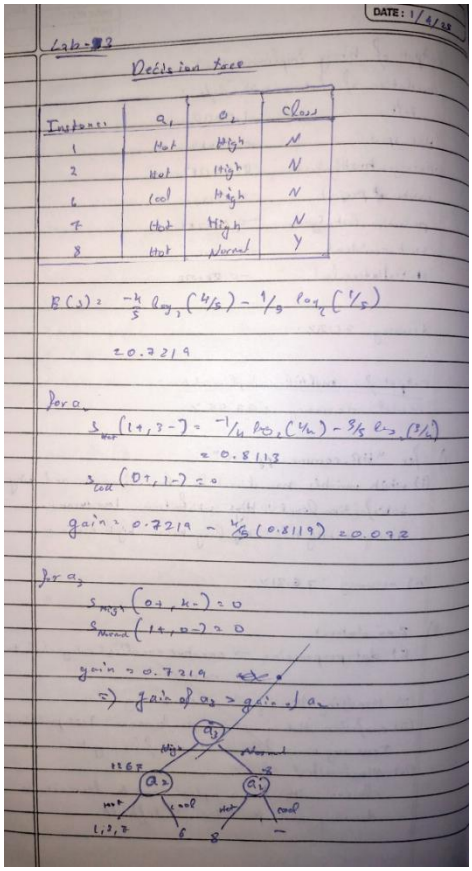
Accuracy of the Multinomial Logistic Regression model on the test set: 1.00

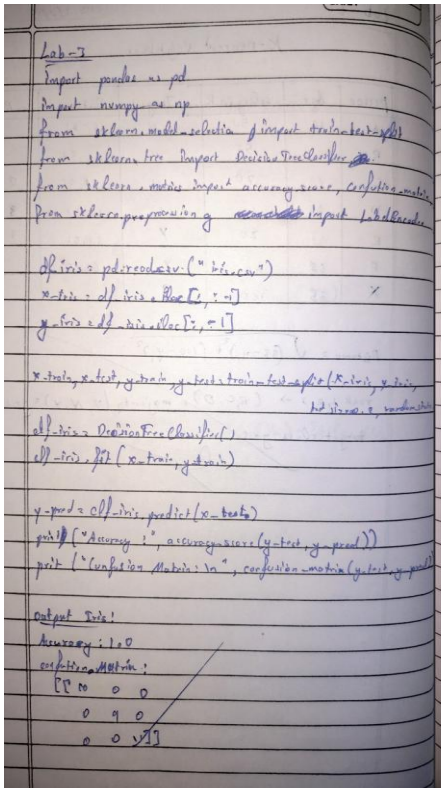


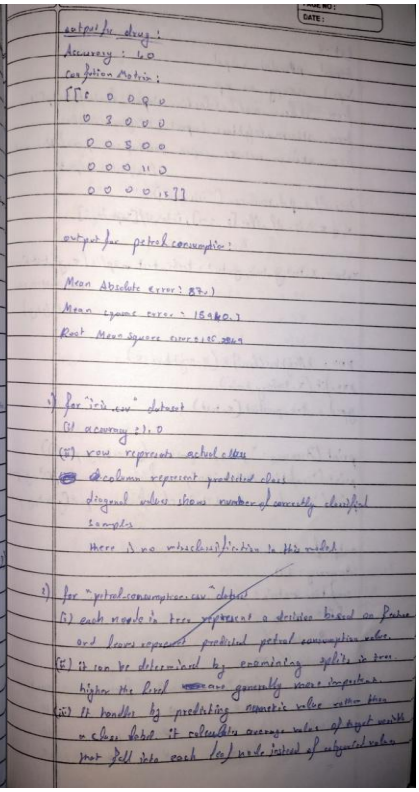
Program 5

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

Screenshots







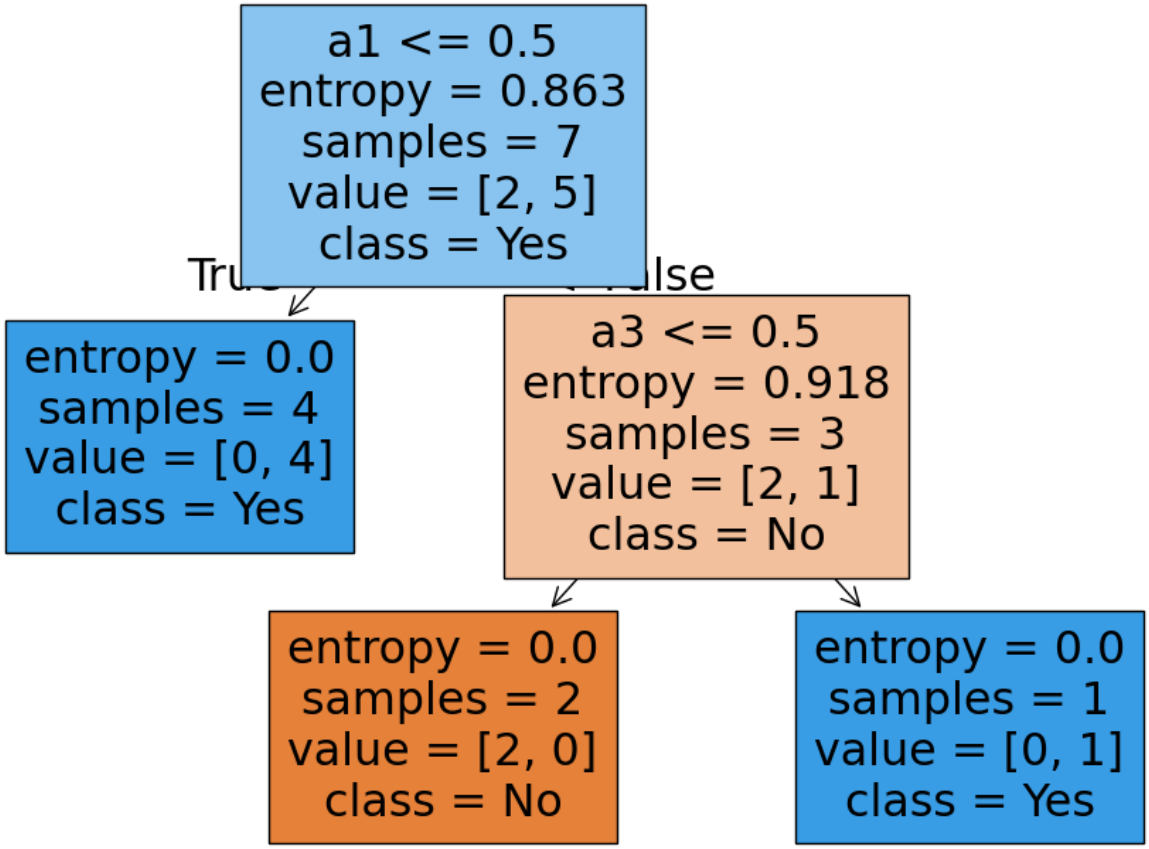
Code:

```
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

# Create the dataset
data = {
    'a1': [True, True, False, False, False, True, True, True, False, False],
```

```
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']
}

data

# Convert to DataFrame
df = pd.DataFrame(data)

# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Split the dataset into features and target
X = df.drop('Classification', axis=1)
y = df['Classification']

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

# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')

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

# Make predictions
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, target_names=['No', 'Yes']))

# Optionally, visualize the decision tree
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, class_names=['No', 'Yes'])
plt.show()
```

```
Accuracy: 1.00
              precision    recall  f1-score   support

          No       1.00      1.00      1.00         2
         Yes       1.00      1.00      1.00         1

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```



In [9]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import accuracy_score, confusion_matrix, mean_absolute_error,
mean_squared_error
from sklearn.preprocessing import LabelEncoder

# Task 1: DecisionTree Classifier for IRIS dataset
df_iris = pd.read_csv("iris (2).csv")
X_iris = df_iris.iloc[:, :-1]  # Features
y_iris = df_iris.iloc[:, -1]   # Target
```

```
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

clf_iris = DecisionTreeClassifier()
clf_iris.fit(X_train, y_train)

y_pred = clf_iris.predict(X_test)
print("IRIS Dataset:")
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
df_drug = pd.read_csv("drug.csv")
X_drug = df_drug.iloc[:, :-1]  # Features
y_drug = df_drug.iloc[:, -1]   # Target

for col in X_drug.select_dtypes(include=['object']).columns:
    X_drug[col] = le.fit_transform(X_drug[col])

X_train, X_test, y_train, y_test = train_test_split(X_drug, y_drug, test_size=0.2, random_state=42)
clf_drug = DecisionTreeClassifier()
clf_drug.fit(X_train, y_train)

y_pred = clf_drug.predict(X_test)
print("\nDrug Dataset:")
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
IRIS Dataset:
Accuracy Score: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Drug Dataset:

Accuracy Score: 1.0

Confusion Matrix:

[[6 0 0 0 0]

[0 3 0 0 0]

[0 0 5 0 0]

[0 0 0 11 0]

[0 0 0 0 15]]

In [15]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import accuracy_score, confusion_matrix, mean_absolute_error,
mean_squared_error
df_petrol = pd.read_csv("petrol_consumption.csv")
X_petrol = df_petrol.iloc[:, :-1]  # Features
y_petrol = df_petrol.iloc[:, -1]   # Target

X_train, X_test, y_train, y_test = train_test_split(X_petrol, y_petrol, test_size=0.2,
random_state=42)
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

Mean Absolute Error: 98.4

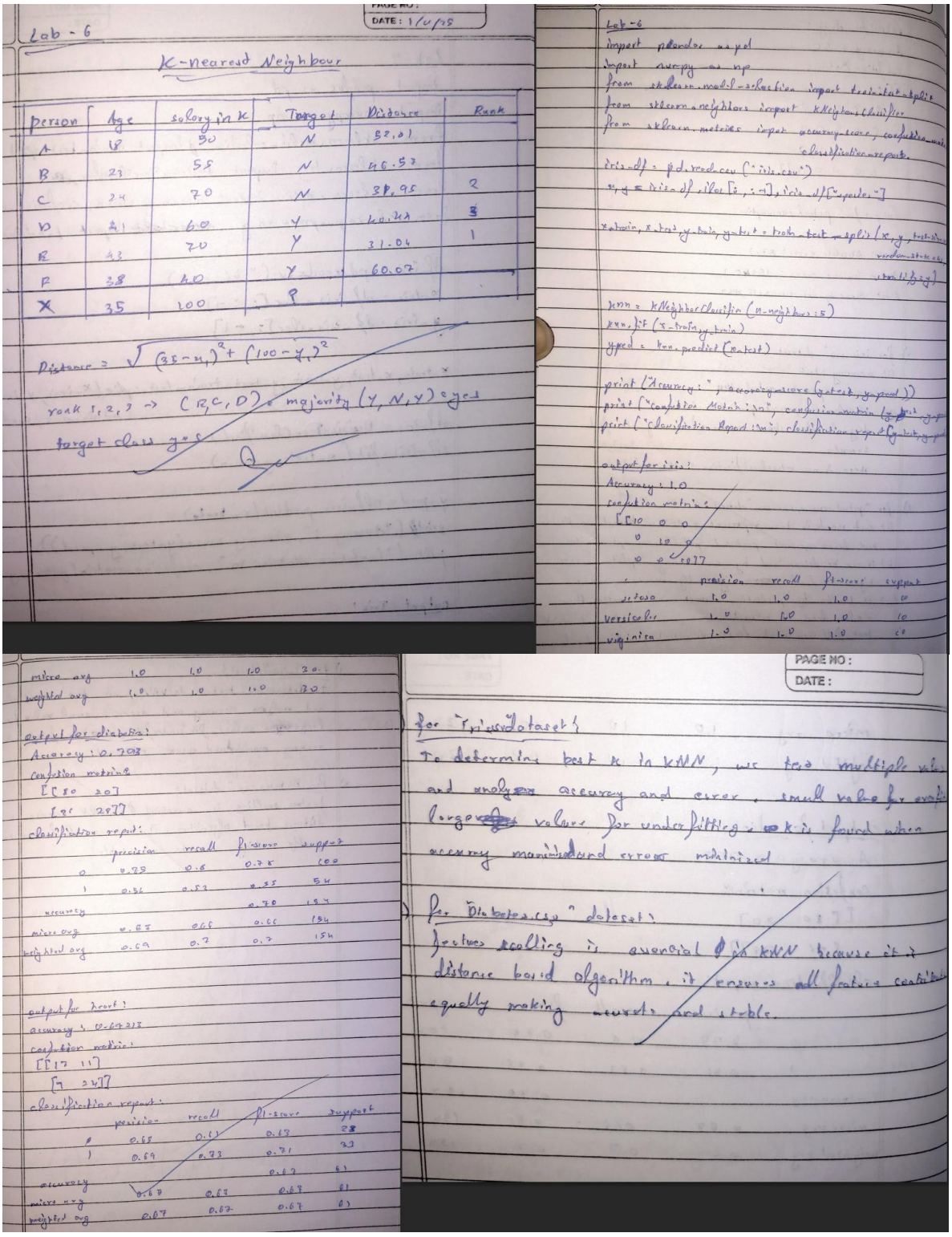
Mean Squared Error: 18007.8

Root Mean Squared Error: 134.19314438524793

Program 6

Build KNN Classification model for a given dataset

Screenshots



Code:

```
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

# Load dataset
iris_df = pd.read_csv("iris (1).csv")
X, y = iris_df.iloc[:, :-1], iris_df["species"]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Train KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Display results
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy Score: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10

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

In [2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load dataset
diabetes_df = pd.read_csv("diabetes.csv")
X, y = diabetes_df.iloc[:, :-1], diabetes_df["Outcome"]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Apply feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Display results
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy Score: 0.7012987012987013
Confusion Matrix:
 [[80 20]
 [26 28]]
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.80      0.78       100
           1       0.58      0.52      0.55        54

    accuracy                           0.70       154
```

```
   macro avg       0.67      0.66      0.66       154
weighted avg       0.69      0.70      0.70       154
```

In [3]:

```
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

# Load dataset
heart_df = pd.read_csv("heart.csv")
X, y = heart_df.iloc[:, :-1], heart_df["target"]

# Find the best k value
best_k, best_score = 0, 0
for k in range(1, 21, 2):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    score = accuracy_score(y_test, knn.predict(X_test))
    if score > best_score:
        best_k, best_score = k, score

print(f"Best k for Heart dataset: {best_k} with accuracy {best_score:.4f}")

# Train KNN with best k
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Display results
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Best k for Heart dataset: 17 with accuracy 0.6721
Accuracy Score: 0.6721311475409836
```

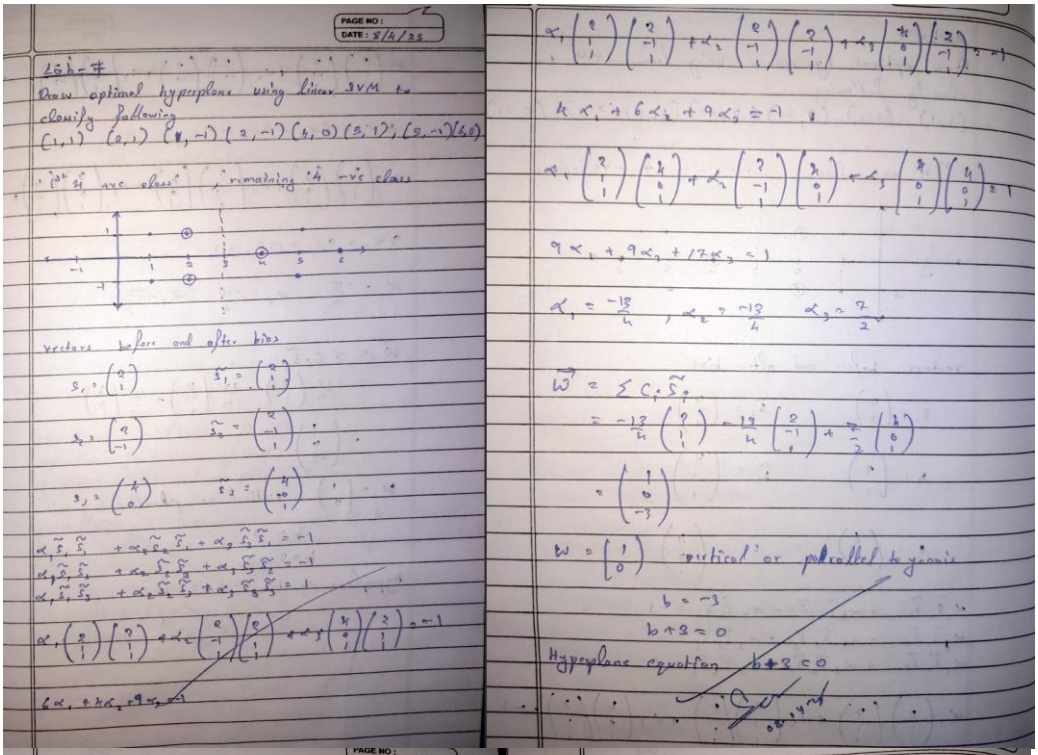
Confusion Matrix:
[[17 11]
[9 24]]
Classification Report:

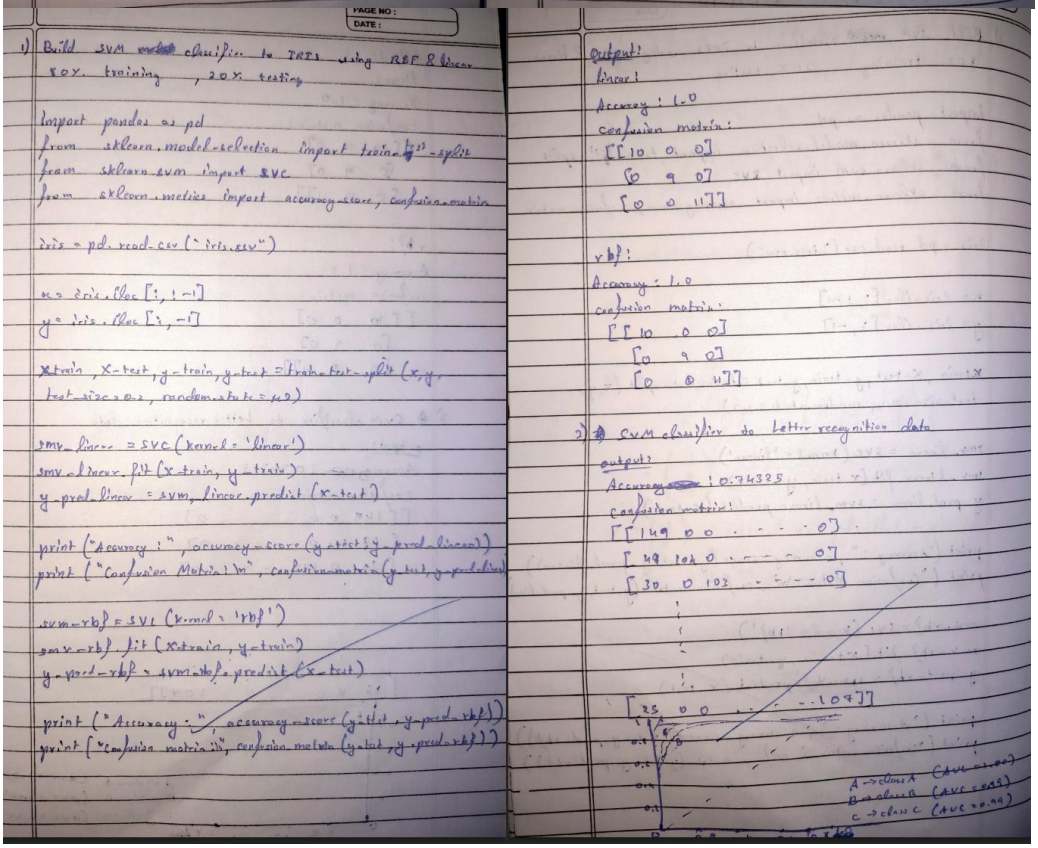
	precision	recall	f1-score	support
0	0.65	0.61	0.63	28
1	0.69	0.73	0.71	33
accuracy			0.67	61
macro avg	0.67	0.67	0.67	61
weighted avg	0.67	0.67	0.67	61

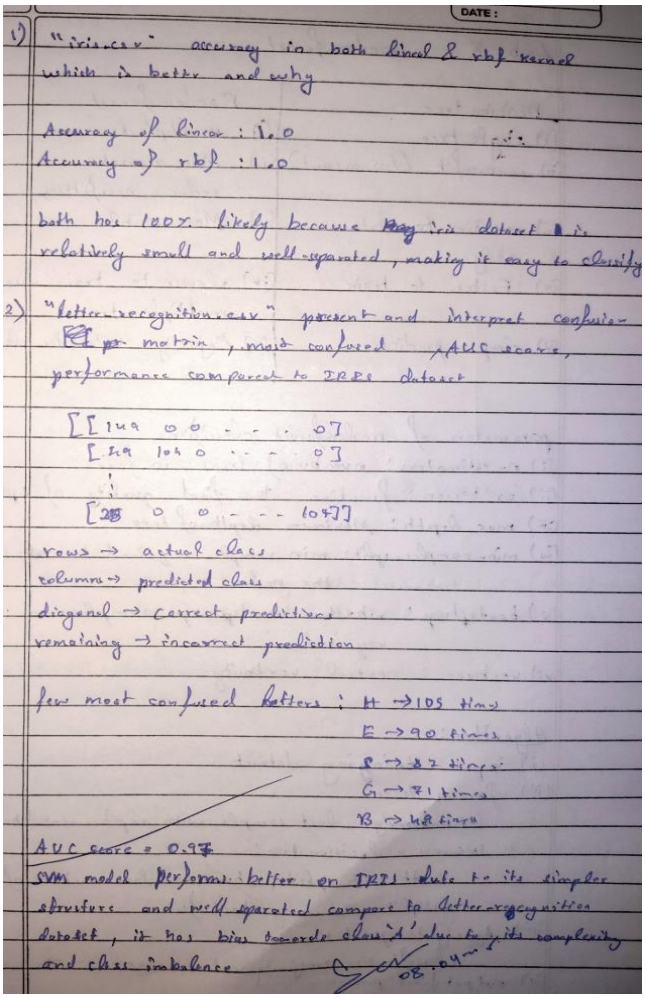
Program 7

Build Support vector machine model for a given dataset

Screenshots







Code:

```
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

# Load dataset
iris = pd.read_csv("iris (1) (1).csv")

# Features and labels
X = iris.iloc[:, :-1]  # assuming last column is target
y = iris.iloc[:, -1]

# Train-test split (80-20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Linear Kernel SVM
```

```
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)

print("Linear Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, y_pred_linear))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_linear))

# RBF Kernel SVM
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)

print("\nRBF Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, y_pred_rbf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rbf))
```

```
Linear Kernel SVM:
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

RBF Kernel SVM:
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
In [4]:
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier

# Load dataset
```

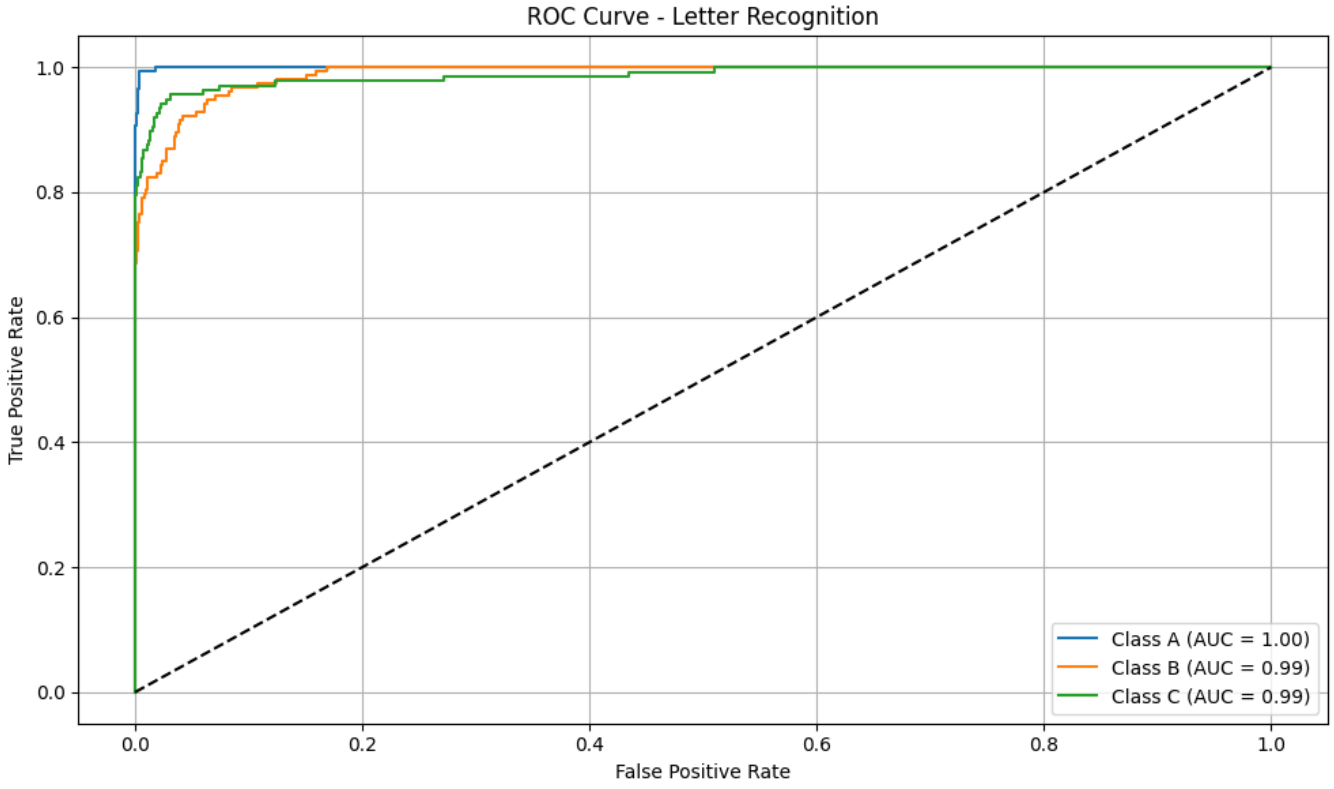
```
letters = pd.read_csv("letter-recognition.csv")  # assuming this contains letter recognition data

# Features and labels
X = letters.iloc[:, 1:]  # assuming first column is label
y = letters.iloc[:, 0]

# Binarize the output for ROC
lb = LabelBinarizer()
y_bin = lb.fit_transform(y)

# Train-test split
X_train, X_test, y_train_bin, y_test_bin = train_test_split(X, y_bin, test_size=0.2, random_state=42)
y_train = lb.inverse_transform(y_train_bin)
y_test = lb.inverse_transform(y_test_bin)

# One-vs-Rest SVM with RBF kernel for multiclass ROC
svm_model = OneVsRestClassifier(SVC(kernel='rbf', probability=True))
svm_model.fit(X_train, y_train_bin)
y_score = svm_model.decision_function(X_test)

# Accuracy and confusion matrix
y_pred = lb.inverse_transform(svm_model.predict(X_test))
print("\nLetter Recognition SVM Classifier:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ROC Curve and AUC
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(len(lb.classes_)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for first 3 classes only (to avoid clutter)
plt.figure(figsize=(10, 6))
for i in range(3):
    plt.plot(fpr[i], tpr[i], label=f'Class {lb.classes_[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # diagonal
plt.title('ROC Curve - Letter Recognition')
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Letter Recognition SVM Classifier:
Accuracy: 0.74325
Confusion Matrix:

```
[[149   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0]
 [ 49 104   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0]
 [ 30   0 103   0   0   0   1   0   0   0   2   0   0   0   1   0   0   0
   0   0   0   0   0   0   0   0]
 [ 43   0   0 113   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0]
 [ 90   0   0   0  51   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0]
 [ 42   0   0   0   0  97   0   0   0   0   0   0   0   0   0   0   0   0
   0   1   0   0   0   0   0   0]
 [ 71   0   0   0   0   0  87   0   0   0   0   0   0   0   0   0   0   2
   0   0   0   0   0   0   0   0]
 [105   0   0   2   0   0   0  33   0   0   0   0   0   0   0   0   0   4
   0   0   0   0   0   0   0   0]
 [ 21   0   0   0   0   0   0   0 122   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   3   0   0]
 [ 25   0   0   0   0   1   0   0   1 121   0   0   0   0   1   0   0   0
   0   0   0   0   0   0   0   0]
 [ 45   0   0   0   0   0   0   0   0   0  80   0   0   0   0   0   0   5
   0   0   0   0   0   0   0   0]
 [ 19   1   0   0   0   0   1   0   0   0   0 133   0   0   0   0   1   0
   0   0   0   0   0   0   0   0]
 [  8   0   0   0   0   0   0   0   0   0   0   0 160   0   0   0   0   0
   0   0   0   0   0   0   0   0]
 [ 19   0   0   1   0   0   0   0   0   0   0   0   0 130   1   0   0   0
   0   0   0   0   0   0   0   0]
 [ 42   0   0   1   0   0   0   0   0   0   0   0   0   0 102   0   0   0
   0   0   0   0   0   0   0   0]
 [ 26   0   0   0   0   7   0   0   0   0   0   0   0   0   1 137   0   0
   0   0   0   0   0   0   2   0]
```

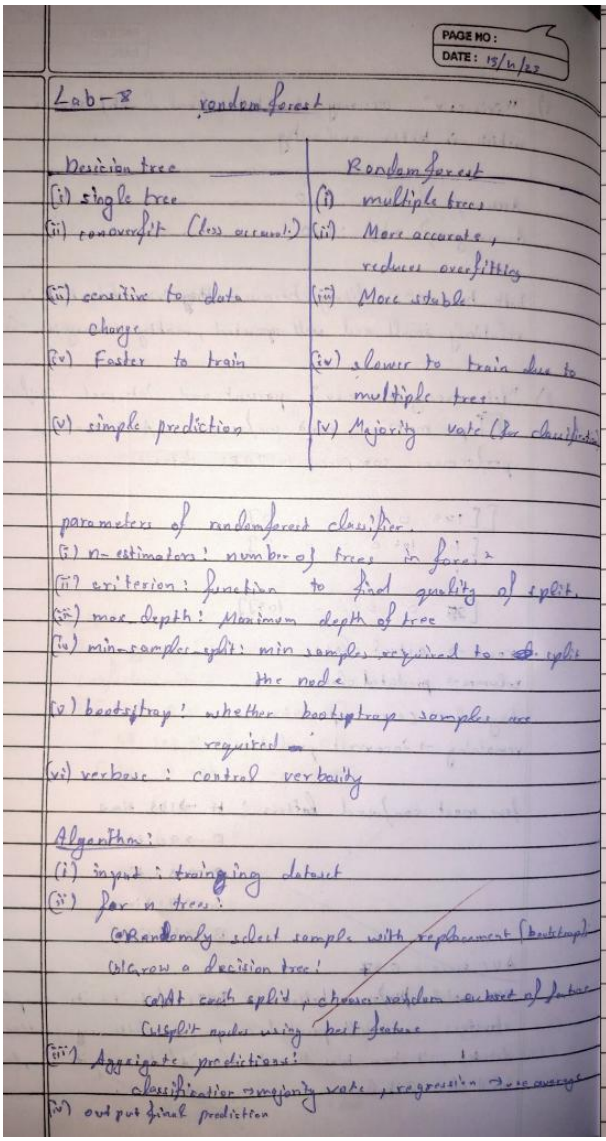
```
[ 35   1   0   1   0   0   0   0   0   0   0   0   0   0   0   0 129   0
   0   0   0   0   0   0   0   0]
[ 48   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0 111
   0   0   0   0   0   0   0   0]
[ 87   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2
  82   0   0   0   0   0   0   0]
[ 26   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0 137   0   0   0   0   0   0]
[ 31   0   2   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
   0   0 149   0   0   0   0   0]
[ 24   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0 134   0   0   0   0]
[  9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0 139   0   0   0]
[ 33   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0 120   1   0]
[ 19   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   5   0   0 143   0]
[ 25   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 107]]
```



Program 8

Implement Random forest ensemble method on a given dataset.

Screenshots

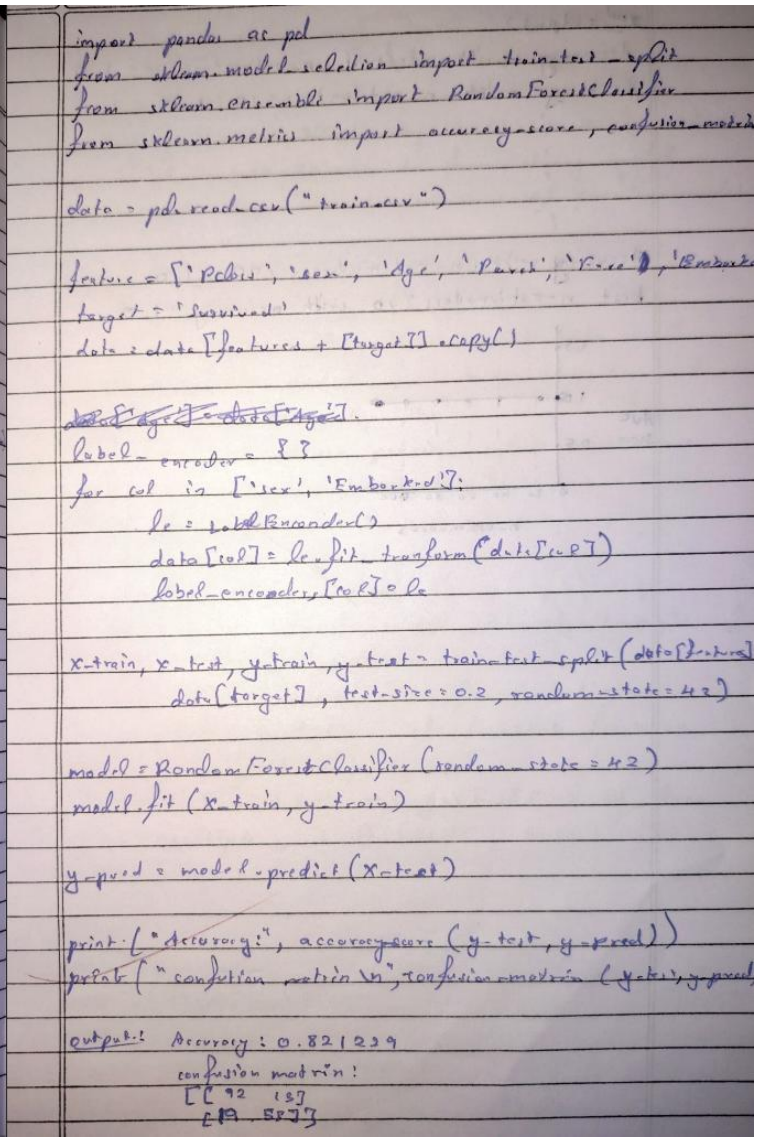


Lab-8 random forest

Decision tree	Random forest
(i) single tree	(i) multiple trees
(ii) overfit (less accurate)	(ii) More accurate, reduces overfitting
(iii) sensitive to data changes	(iii) More stable
(iv) Faster to train	(iv) slower to train due to multiple trees
(v) simple prediction	(v) Majority vote (for classification)

parameters of randomforest classifier.
(i) n_estimators: number of trees in forest
(ii) criterion: function to find quality of split.
(iii) max_depth: Maximum depth of tree
(iv) min_samples_split: min samples required to split the node
(v) bootstrap: whether bootstrap samples are required
(vi) verbose: control verbosity

Algorithm:
(i) input: training dataset
(ii) for n trees:
(a) Randomly select sample with replacement (bootstrap)
(b) Grow a decision tree:
at each split, choose random subset of features
Split node using best feature
(iii) Aggregate predictions:
classification: majority vote, regression: mean average
(iv) output final prediction



```
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
data = pd.read_csv("train.csv")
features = ['Pclass', 'sex', 'Age', 'Parch', 'Fare', 'Embarked']
target = 'Survived'
data = data[features + [target]].copy()
label_encoders = {}
for col in ['sex', 'Embarked']:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
X_train, X_test, y_train, y_test = train_test_split(data[features],
    data[target], test_size=0.2, random_state=42)
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("confusion matrix\n", confusion_matrix(y_test, y_pred
```

output: Accuracy: 0.821229
confusion matrix:
[[92 13]
[19 55]]

Code:

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

```
# Load the dataset
file_path = "train.xlsx"
data = pd.read_excel(file_path)

# Select features and target
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
target = 'Survived'
data = data[features + [target]].copy()  # avoid SettingWithCopyWarning

# Handle missing values (future-safe syntax)
data['Age'] = data['Age'].fillna(data['Age'].median())
data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode()[0])

# Encode categorical features
label_encoders = {}
for col in ['Sex', 'Embarked']:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

# Split the dataset
X = data[features]
y = data[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Random Forest model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Output results
print("Accuracy Score:", accuracy)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy Score: 0.8212290502793296
Confusion Matrix:
[[92 13]
 [19 55]]
In [9]:
```

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

# Load the dataset
file_path = "iris.xlsx"  # Ensure this file is in your working directory
iris_data = pd.read_excel(file_path)

# Prepare features and target
X = iris_data.drop('species', axis=1)
y = iris_data['species']
le = LabelEncoder()
y_encoded = le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
```

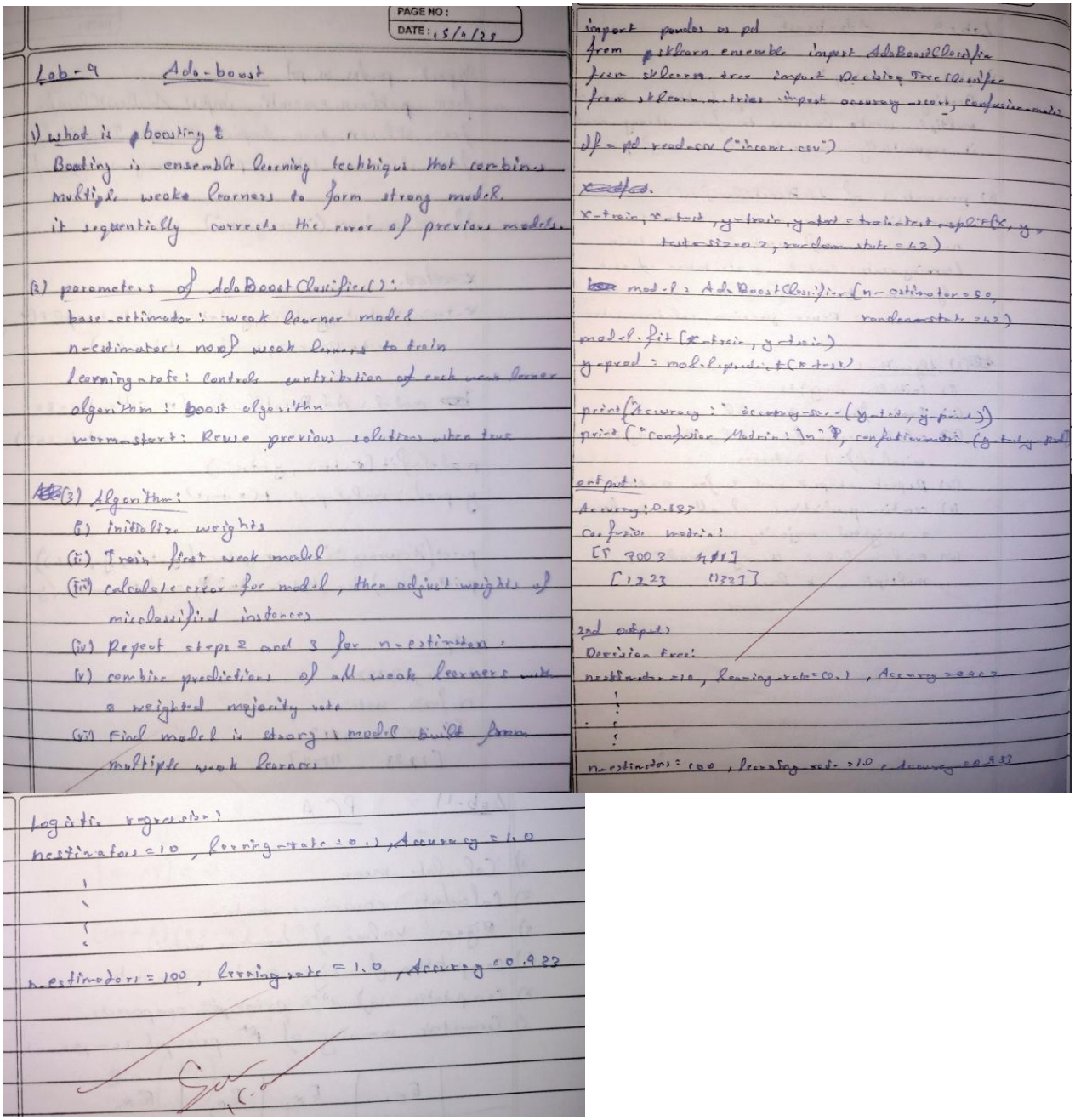
```
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred_default)
print("Accuracy with default n_estimators=10:", default_accuracy)
best_score = 0
best_n = 10
for n in range(10, 110, 10):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"Accuracy with n_estimators={n}: {acc}")
    if acc > best_score:
        best_score = acc
        best_n = n

print(f"\nBest n_estimators: {best_n} with accuracy: {best_score}")
```

```
Accuracy with default n_estimators=10: 1.0
Accuracy with n_estimators=10: 1.0
Accuracy with n_estimators=20: 1.0
Accuracy with n_estimators=30: 1.0
Accuracy with n_estimators=40: 1.0
Accuracy with n_estimators=50: 1.0
Accuracy with n_estimators=60: 1.0
Accuracy with n_estimators=70: 1.0
Accuracy with n_estimators=80: 1.0
Accuracy with n_estimators=90: 1.0
Accuracy with n_estimators=100: 1.0

Best n_estimators: 10 with accuracy: 1.0
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshots



Code:

```
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset
df = pd.read_csv("income.csv")

# Define features and target
X = df.drop('income_level', axis=1)
y = df['income_level']

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

# Initialize AdaBoost with a Decision Tree base estimator
base_estimator = DecisionTreeClassifier(max_depth=1)
model = AdaBoostClassifier(n_estimators=50, random_state=42)

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

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display results
```

```
print(f"Accuracy: {accuracy:.3f}")
print("Confusion Matrix:")
print(conf_matrix)

Accuracy: 0.833
Confusion Matrix:
[[7003  411]
 [1223 1132]]
In [5]:

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
iris_df = pd.read_csv('iris.csv')

# Encode the target variable (species)
le = LabelEncoder()
iris_df['species'] = le.fit_transform(iris_df['species'])

# Features and target
X = iris_df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = iris_df['species']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Function to train and evaluate AdaBoost
def evaluate_adaboost(estimator, n_estimators, learning_rate, estimator_name):
    model = AdaBoostClassifier(
        estimator=estimator,
        n_estimators=n_estimators,
        learning_rate=learning_rate,
        random_state=42
    )
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{estimator_name} with n_estimators={n_estimators}, learning_rate={learning_rate}:
Accuracy = {accuracy:.3f}")
    return accuracy

# Experiment 1: Vary n_estimators and learning_rate with Decision Tree
print("AdaBoost with Decision Tree:")
dt_base = DecisionTreeClassifier(max_depth=1)
n_estimators_list = [10, 50, 100]
learning_rates = [0.1, 0.5, 1.0]

for n in n_estimators_list:
    for lr in learning_rates:
        evaluate_adaboost(dt_base, n, lr, "Decision Tree")

# Experiment 2: Use Logistic Regression as base classifier
print("\nAdaBoost with Logistic Regression:")
logreg_base = LogisticRegression(max_iter=1000)
for n in n_estimators_list:
    for lr in learning_rates:
        evaluate_adaboost(logreg_base, n, lr, "Logistic Regression")
```

AdaBoost with Decision Tree:

Decision Tree with n_estimators=10, learning_rate=0.1: Accuracy = 0.967

Decision Tree with n_estimators=10, learning_rate=0.5: Accuracy = 1.000

Decision Tree with n_estimators=10, learning_rate=1.0: Accuracy = 1.000

Decision Tree with n_estimators=50, learning_rate=0.1: Accuracy = 1.000

Decision Tree with n_estimators=50, learning_rate=0.5: Accuracy = 0.967

Decision Tree with n_estimators=50, learning_rate=1.0: Accuracy = 0.933

Decision Tree with n_estimators=100, learning_rate=0.1: Accuracy = 1.000

Decision Tree with n_estimators=100, learning_rate=0.5: Accuracy = 1.000

Decision Tree with n_estimators=100, learning_rate=1.0: Accuracy = 0.933

AdaBoost with Logistic Regression:

Logistic Regression with n_estimators=10, learning_rate=0.1: Accuracy = 1.000

Logistic Regression with n_estimators=10, learning_rate=0.5: Accuracy = 0.967

Logistic Regression with n_estimators=10, learning_rate=1.0: Accuracy = 0.933

Logistic Regression with n_estimators=50, learning_rate=0.1: Accuracy = 1.000

Logistic Regression with n_estimators=50, learning_rate=0.5: Accuracy = 1.000

Logistic Regression with n_estimators=50, learning_rate=1.0: Accuracy = 0.933

Logistic Regression with n_estimators=100, learning_rate=0.1: Accuracy = 1.000

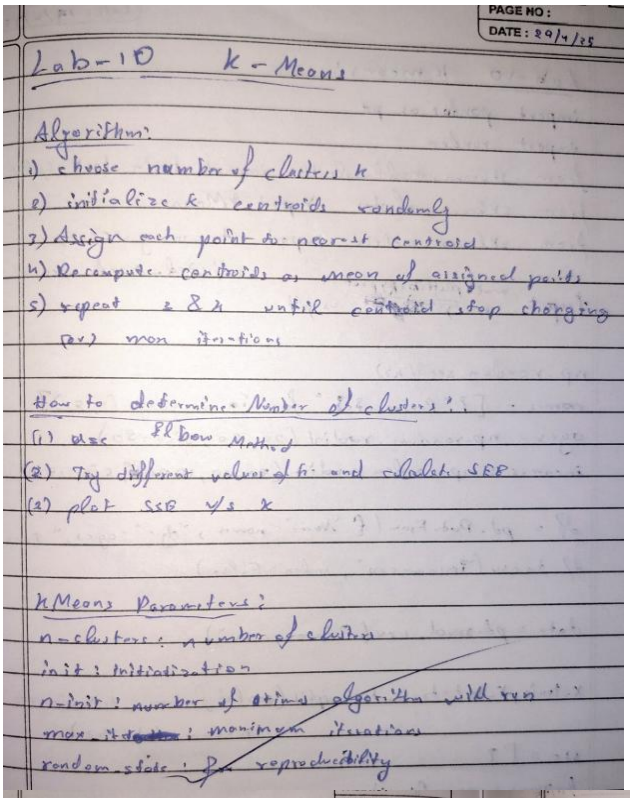
Logistic Regression with n_estimators=100, learning_rate=0.5: Accuracy = 1.000

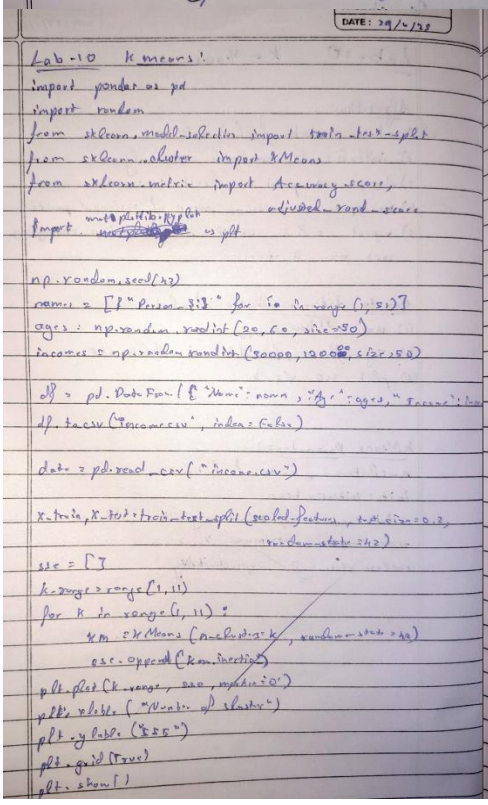
Logistic Regression with n_estimators=100, learning_rate=1.0: Accuracy = 0.933

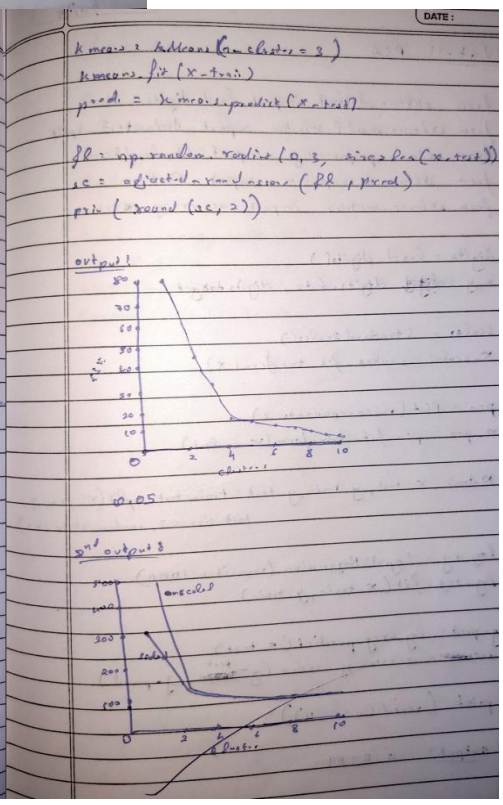
Program 10

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

Screenshots



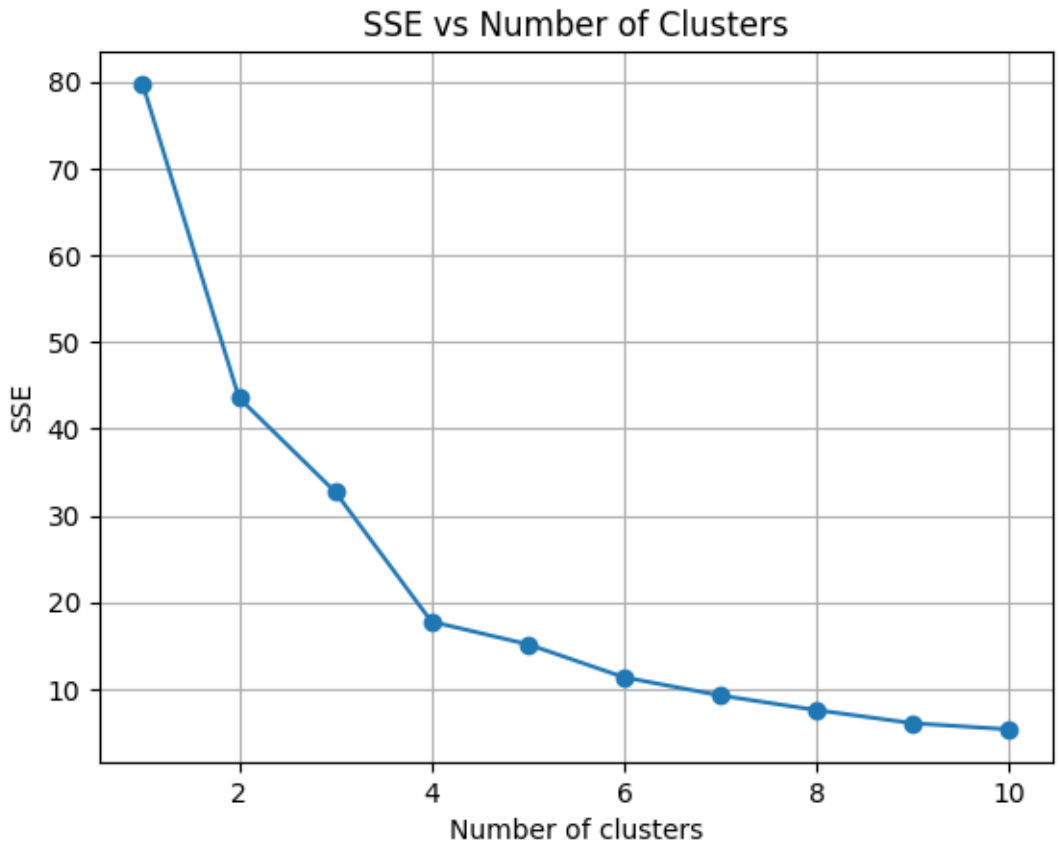




Code:

```
import pandas as pd
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, adjusted_rand_score
import matplotlib.pyplot as plt

# Step 1: Generate synthetic data
np.random.seed(42)
names = [f"Person_{i}" for i in range(1, 51)]
ages = np.random.randint(20, 60, size=50)
incomes = np.random.randint(30000, 120000, size=50)

df = pd.DataFrame({
    "Name": names,
    "Age": ages,
    "Income": incomes
})

# Save to CSV
df.to_csv("income.csv", index=False)

# Step 2: Load the data
data = pd.read_csv("income.csv")

# Step 3: Preprocess (Drop 'Name', scale 'Age' and 'Income')
features = data[["Age", "Income"]]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

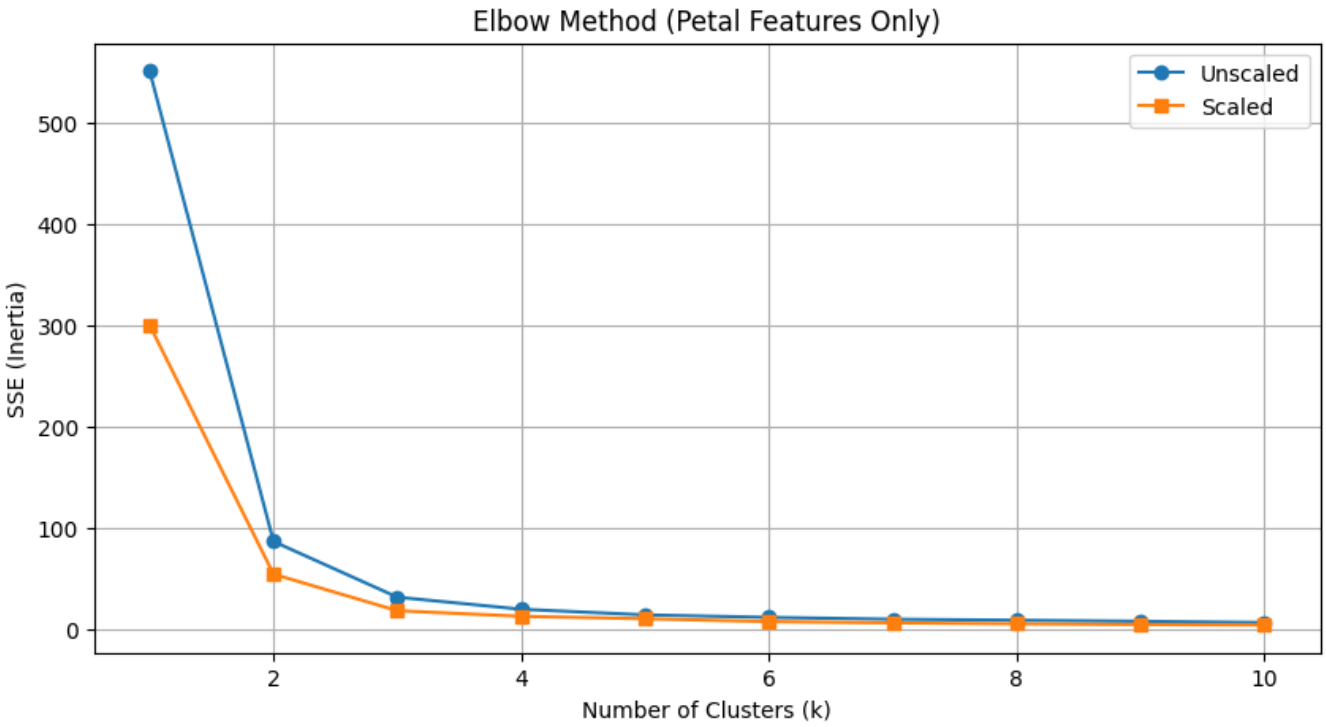
```
# Step 4: Train-test split
X_train, X_test = train_test_split(scaled_features, test_size=0.2, random_state=42)

# Step 5: Elbow Method for optimal clusters
sse = []
k_range = range(1, 11)
for k in k_range:
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X_train)
    sse.append(km.inertia_)

plt.plot(k_range, sse, marker='o')
plt.title("SSE vs Number of Clusters")
plt.xlabel("Number of clusters")
plt.ylabel("SSE")
plt.grid(True)
plt.show()

# Step 6: Choose number of clusters (e.g., 3), train and predict
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_train)
predictions = kmeans.predict(X_test)

# Note: In unsupervised learning, accuracy isn't well-defined because we don't have true labels.
# We'll just show how to use predict and compute adjusted_rand_score on synthetic labels.
# Fake labels for demonstration (not realistic, only for illustrating scoring usage)
fake_labels = np.random.randint(0, 3, size=len(X_test))

# Use adjusted rand index for clustering evaluation
score = adjusted_rand_score(fake_labels, predictions)
print("Adjusted Rand Index Score (for demonstration):", round(score, 2))
```



Adjusted Rand Index Score (for demonstration): 0.05

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# Step 1: Load Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
```

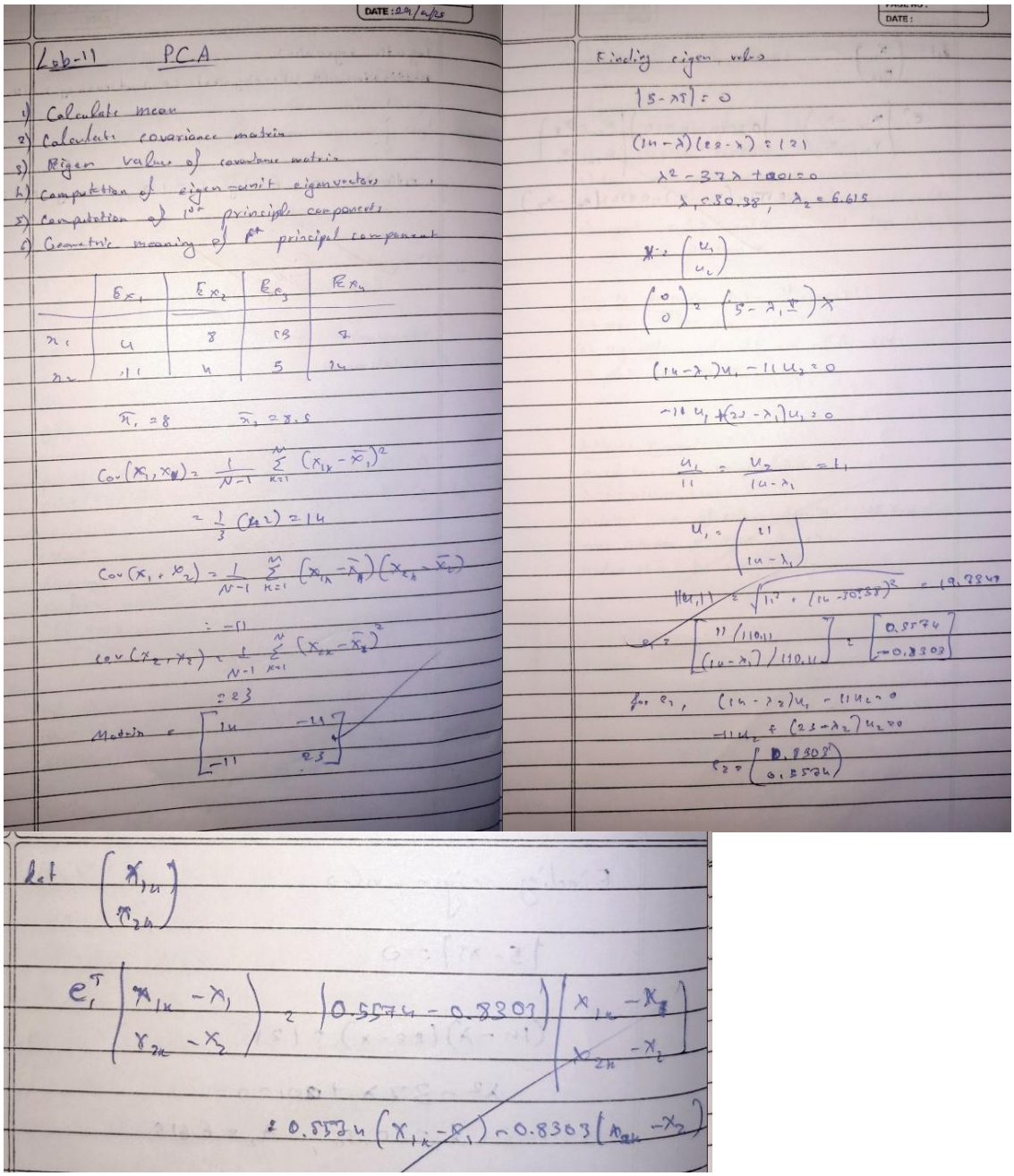
```
# Keep only petal length and petal width
X = df[['petal length (cm)', 'petal width (cm)']].values

# Step 2: Check impact of scaling
# Try without scaling
sse_unscaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    sse_unscaled.append(kmeans.inertia_)

# Now scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

sse_scaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    sse_scaled.append(kmeans.inertia_)

# Step 3: Plot Elbow Comparison (Scaled vs Unscaled)
plt.figure(figsize=(10, 5))

plt.plot(range(1, 11), sse_unscaled, marker='o', label='Unscaled')
plt.plot(range(1, 11), sse_scaled, marker='s', label='Scaled')
plt.title('Elbow Method (Petal Features Only)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('SSE (Inertia)')
plt.legend()
plt.grid(True)
plt.show()
```

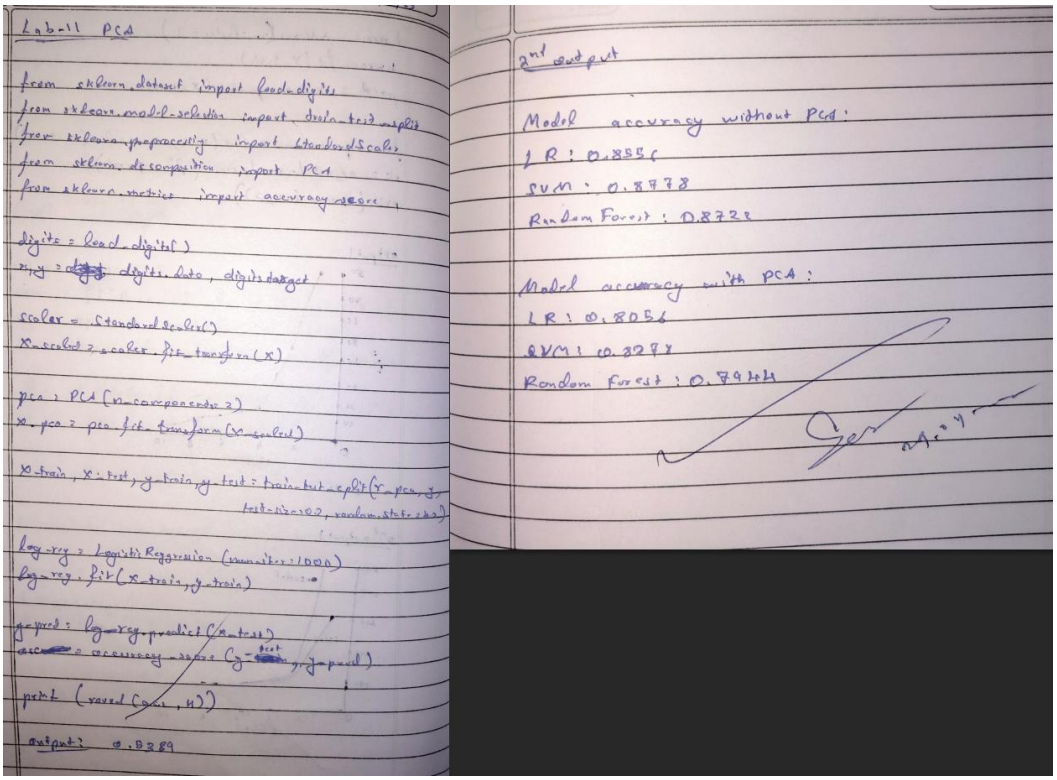

Elbow Method (Petal Features Only)
Unscaled
Scaled
SSE (Inertia)
0
100
200
300
400
500
2
4
6
8
10
Number of Clusters (k)

Program 11

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

Screenshots





Code:

```
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

# Step 1: Load the dataset
digits = load_digits()
X, y = digits.data, digits.target

# Step 2: Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA with n_components=2
```

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Step 5: Train Logistic Regression on PCA-reduced data
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Step 6: Predict and evaluate
y_pred = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy using PCA (2 components) and Logistic Regression:", round(accuracy, 4))
```

```
Accuracy using PCA (2 components) and Logistic Regression: 0.5389
```

In [5]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from scipy.stats import zscore

# Step 1: Load the dataset
```

```
df = pd.read_csv("heart.csv")

# Step 2: Remove outliers using Z-score
numeric_cols = df.select_dtypes(include=[np.number])
z_scores = np.abs(zscore(numeric_cols))
df = df[(z_scores < 3).all(axis=1)]

# Step 3: Encode categorical features (if any)
df_encoded = df.copy()
for col in df_encoded.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col])

# Step 4: Scaling
X = df_encoded.drop('HeartDisease', axis=1)
y = df_encoded['HeartDisease']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 5: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Step 5: Classification with different models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "SVM": SVC(),
    "Random Forest": RandomForestClassifier()
}

print("Model accuracies without PCA:")
for name, model in models.items():
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name}: {acc:.4f}")

# Step 6: PCA with 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=42)

print("\nModel accuracies with PCA (2 components):")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name}: {acc:.4f}")
```

```
Model accuracies without PCA:
Logistic Regression: 0.8556
SVM: 0.8778
Random Forest: 0.8722

Model accuracies with PCA (2 components):
Logistic Regression: 0.8056
SVM: 0.8278
Random Forest: 0.7944
```