VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Devanshi Slathia(1BM22CS083)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

Sep-2024 to Jan-2025

B.M.S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **Devanshi Slathia(1BM22CS083),** 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.

Rashmi H Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

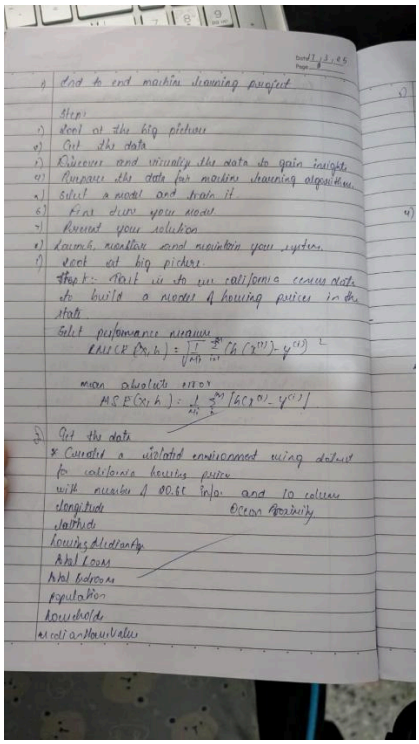
Index

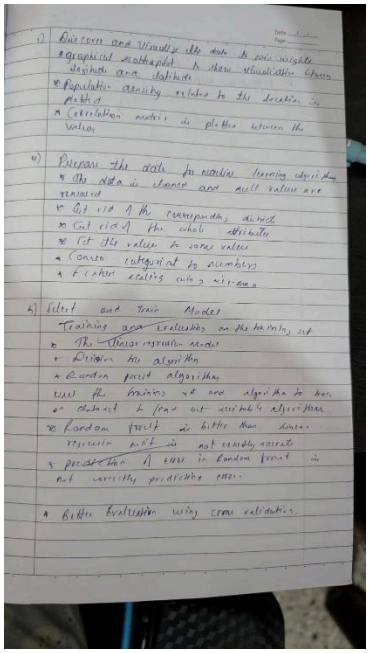
https://github.com/Devanshislathia/Machine-Learning-Lab/tree/main

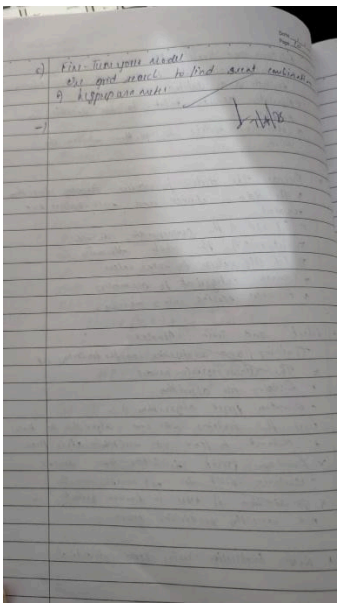
Program 1

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

Screenshots







Code:

```
import pandas as pd
data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, 30, 35, 40],

'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']

}

df = pd.DataFrame(data)

print("Sample data:")

print(df.head())
```

```
Sample data:
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
```

```
2  Charlie   35      Chicago
3    David   40      Houston
```
In [ ]:

```
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
```
In [ ]:

```
# 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
```

In [ ]:

```
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
```

In [ ]:

```
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
```

In [ ]:

```
# 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
```

In [ ]:

```
# 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
```

In [ ]:

```
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
In [ ]:

# 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
In [ ]:

# 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
In [ ]:

# 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
31.0    25032.0
```

13.0   24657.0
38.0   23639.0
39.0   22211.0
43.0   22042.0
6.0   20872.0
44.0   19610.0
42.0   19163.0
41.0   19140.0
45.0   17695.0
10.0   16622.0
46.0   16571.0
9.0   15913.0
40.0   14746.0
8.0   14511.0
48.0   12280.0
3.0   12250.0
7.0   12015.0
47.0   9384.0
49.0   6696.0
2.0   6085.0
50.0   5701.0
51.0   4037.0
1.0   17.0
Name: households, dtype: float64
In [ ]:

```
# 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.
In [ ]:

```
import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt
```

In [ ]:

```
# 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 completed
First 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
In [ ]:
```

```
# 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'),
```

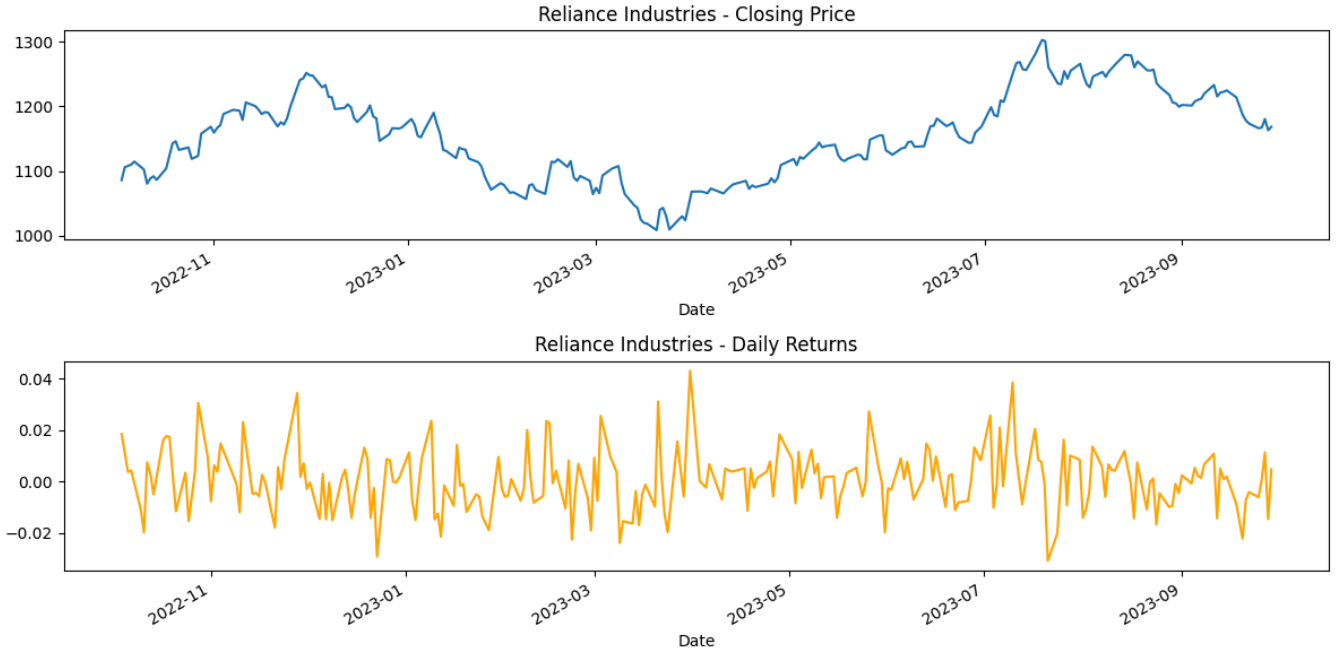
```
        (    'TCS.NS',  'Close'),
        (    'TCS.NS', 'Volume'),
        (   'INFY.NS',   'Open'),
        (   'INFY.NS',   'High'),
        (   'INFY.NS',    'Low'),
        (   'INFY.NS',  'Close'),
        (   'INFY.NS', 'Volume')],
       names=['Ticker', 'Price'])
```

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

In [ ]:

```
# 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()
```



In [ ]:

 # 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'.

In [ ]:

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

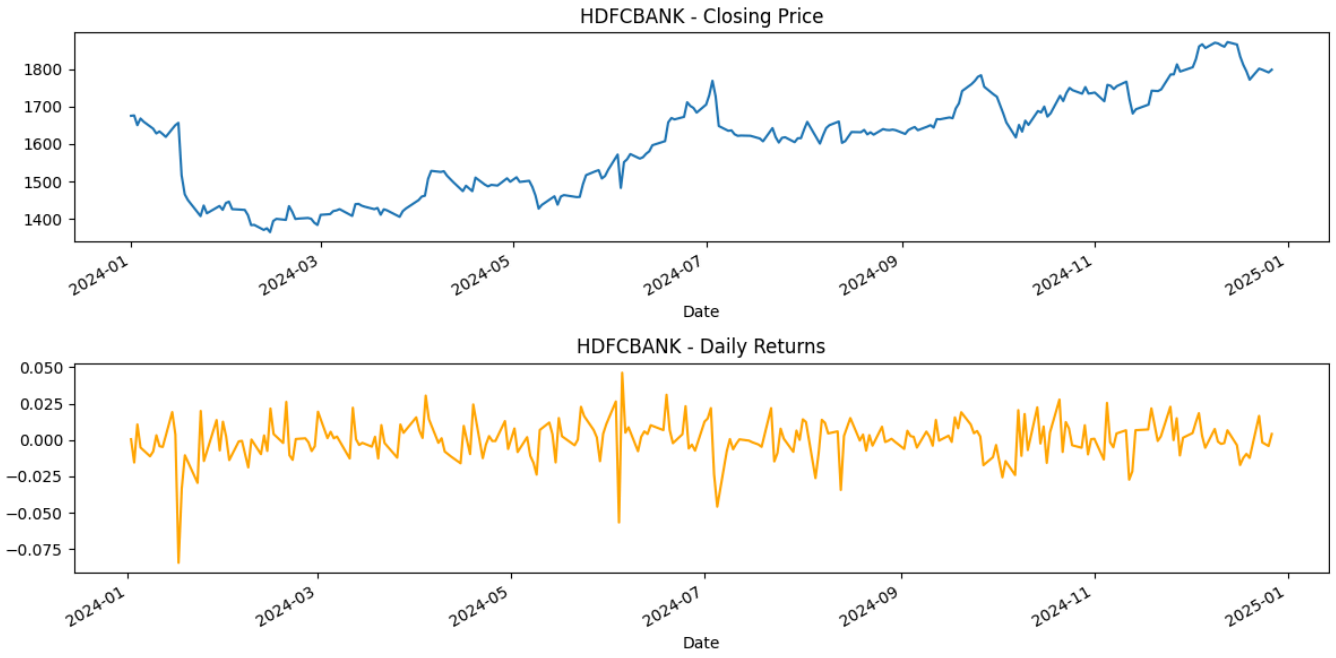
```
2024-01-04   977.980767  980.707295  973.519176  978.724365  22789140
2024-01-05   979.567084  989.779158  975.402920  985.218445  14875499

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
In [ ]:

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
In [ ]:

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
In [ ]:

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

In [ ]:

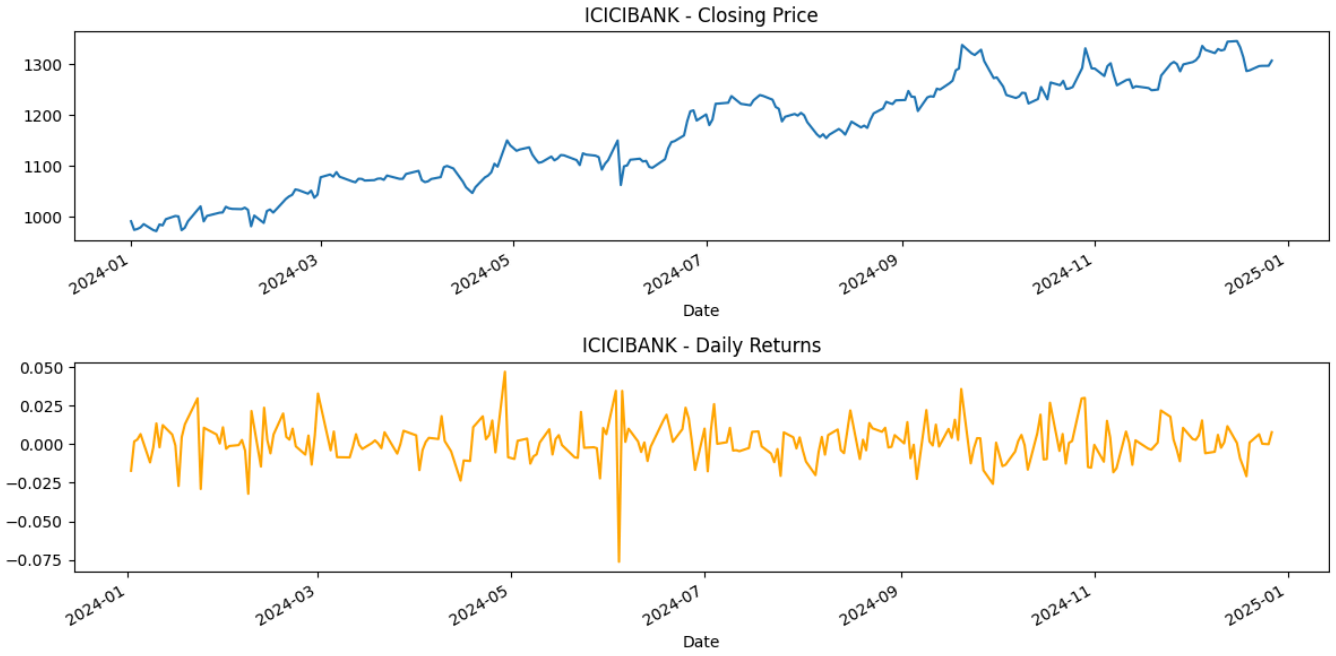
```
# Plot the closing price and daily returns

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()
```



In [ ]:

```
# Plot the closing price and daily returns

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

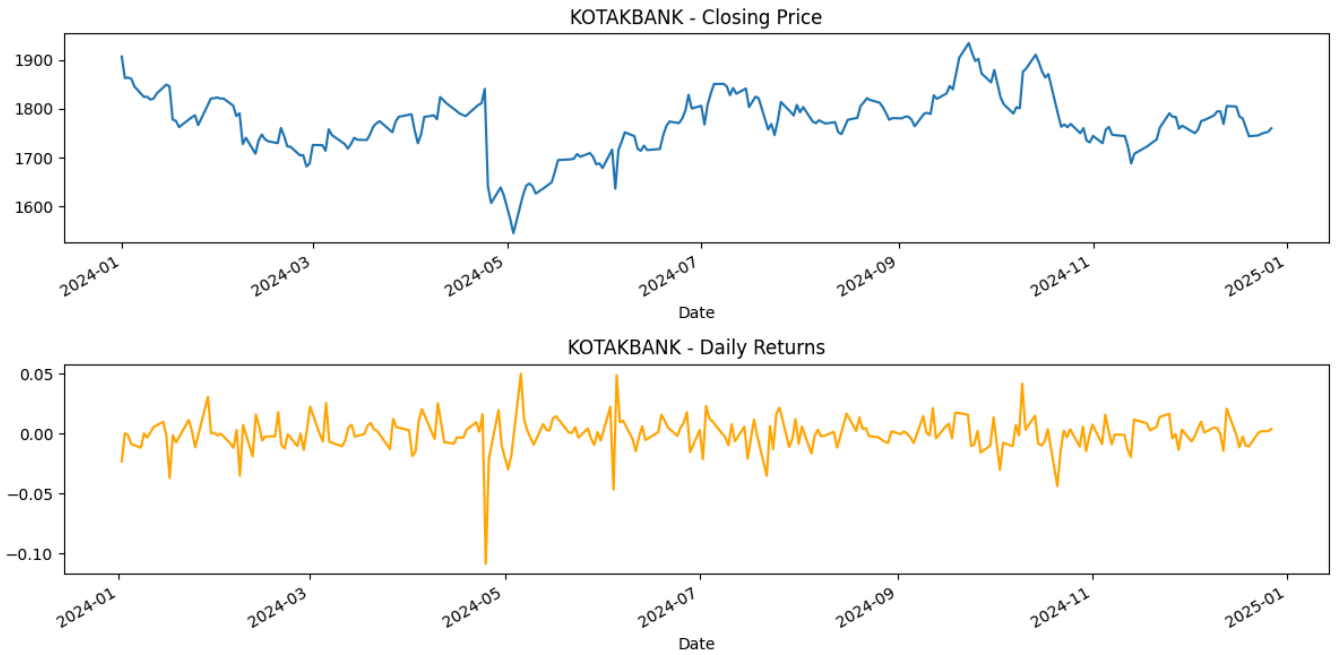
```
plt.subplot(2, 1, 1)

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()
```



In [ ]:

```
# 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()



In [ ]:

 # 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')

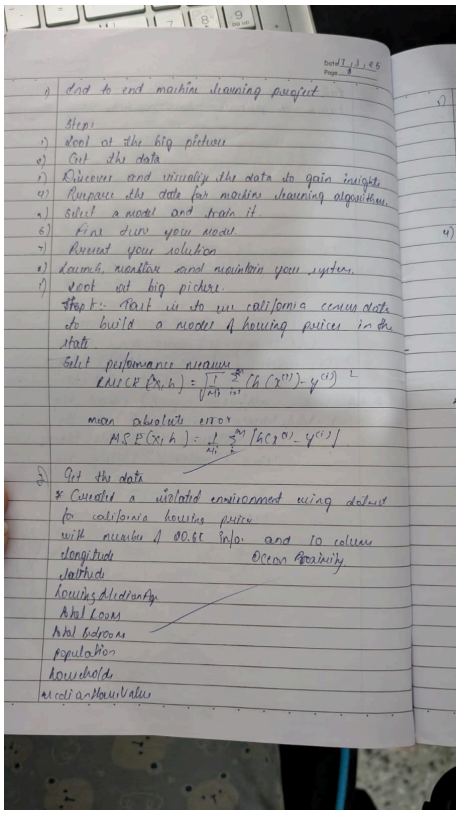
print("\nSAVED")

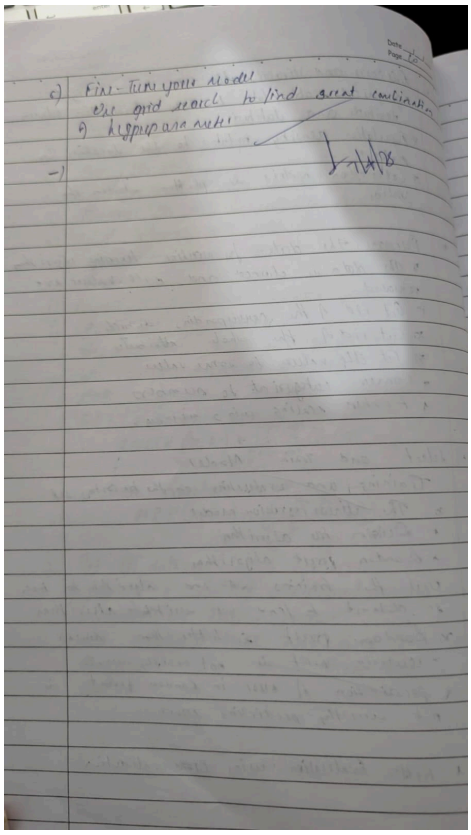
SAVED

Program 2

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
```

```
#To display information of all columns
print(df.info)
```

```
<bound method DataFrame.info of        longitude  latitude  housing_median_age
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
```

#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
```

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

#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())
#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	City_San Francisco
0	67.0	95582.0	0	1.0	1.0	0.0	0.0
1	58.0	75108.0	1	1.0	0.0	0.0	1.0
2	64.0	94631.0	1	0.0	0.0	1.0	0.0
3	42.0	71454.0	0	1.0	0.0	1.0	0.0
4	44.0	108391.0	1	0.0	0.0	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.

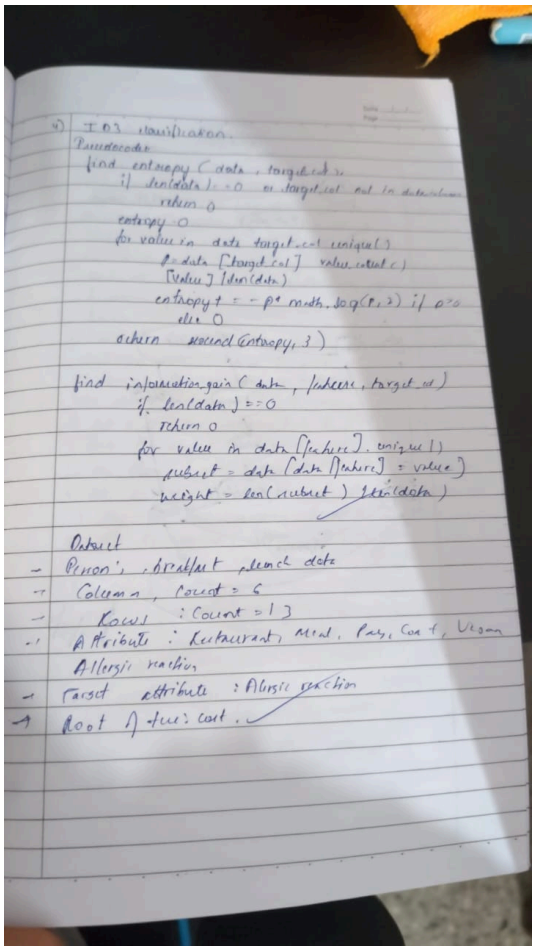
```
df_encoded_copy3['Salary_zscore'] = stats.zscore(df_encoded_copy3['Salary'])
median_salary = df_encoded_copy3['Salary'].median()
df_encoded_copy3['Salary'] = np.where(df_encoded_copy3['Salary_zscore'].abs() > 3, median_salary, df_encoded_copy3['Salary'])
print(df_encoded_copy3.head())
```

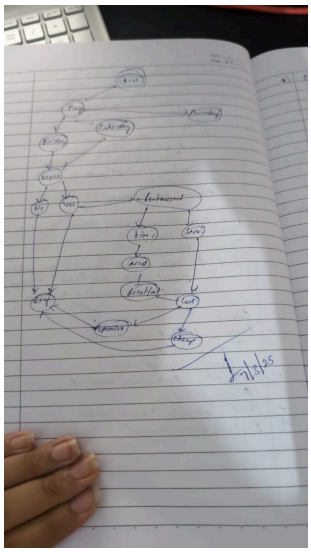
```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0  1.456069  0.778625          0             1.0               1.0
1  0.839381  0.506259          1             1.0               0.0
2  1.250506  0.765974          1             0.0               0.0
3 -0.256953  0.457650    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

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  # for manipulating the csv data
import numpy as np    # for mathematical calculation
```

```
# Load dataset
train_data_m = pd.read_csv("/content/PlayTennis.csv")

# Function to calculate total entropy of the dataset
def calc_total_entropy(train_data, label, class_list):
    total_row = train_data.shape[0]
    total_entr = 0
    for c in class_list:
        total_class_count = train_data[train_data[label] == c].shape[0]
        if total_class_count != 0:
            probability = total_class_count / total_row
            total_entr -= probability * np.log2(probability)
    return total_entr

# Function to calculate entropy for a subset of the data
def calc_entropy(feature_value_data, label, class_list):
    class_count = feature_value_data.shape[0]
    entropy = 0
    for c in class_list:
        label_class_count = feature_value_data[feature_value_data[label] == c].shape[0]
        if label_class_count != 0:
            probability_class = label_class_count / class_count
            entropy -= probability_class * np.log2(probability_class)
    return entropy

# Function to calculate information gain for a feature
def calc_info_gain(feature_name, train_data, label, class_list):
    feature_value_list = train_data[feature_name].unique()
    total_row = train_data.shape[0]
    feature_info = 0.0
    for feature_value in feature_value_list:
        feature_value_data = train_data[train_data[feature_name] == feature_value]
        feature_value_count = feature_value_data.shape[0]
```

```
        feature_value_entropy = calc_entropy(feature_value_data, label, class_list)
        feature_info += (feature_value_count / total_row) * feature_value_entropy
    return calc_total_entropy(train_data, label, class_list) - feature_info

# Find the most informative feature
def find_most_informative_feature(train_data, label, class_list):
    feature_list = train_data.columns.drop(label)
    max_info_gain = -1
    max_info_feature = None
    for feature in feature_list:
        feature_info_gain = calc_info_gain(feature, train_data, label, class_list)
        if max_info_gain < feature_info_gain:
            max_info_gain = feature_info_gain
            max_info_feature = feature
    return max_info_feature

# Generate subtree for a feature
def generate_sub_tree(feature_name, train_data, label, class_list):
    feature_value_count_dict = train_data[feature_name].value_counts(sort=False)
    tree = {}
    rows_to_remove = []

    for feature_value, count in feature_value_count_dict.items():
        feature_value_data = train_data[train_data[feature_name] == feature_value]
        assigned_to_node = False
        for c in class_list:
            class_count = feature_value_data[feature_value_data[label] == c].shape[0]
            if class_count == count:
                tree[feature_value] = c
                rows_to_remove.append(feature_value_data.index)
                assigned_to_node = True
                break
        if not assigned_to_node:
```

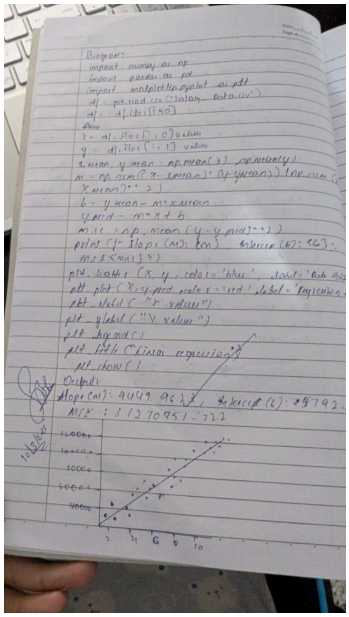
```
            tree[feature_value] = "?"
    train_data = train_data.drop(index=np.concatenate(rows_to_remove)) if
rows_to_remove else train_data
    return tree, train_data

# Recursive tree-building function
def make_tree(root, prev_feature_value, train_data, label, class_list):
    if train_data.shape[0] != 0:
        max_info_feature = find_most_informative_feature(train_data, label, class_list)
        if max_info_feature is None:
            return
        tree, updated_train_data = generate_sub_tree(max_info_feature, train_data, label,
class_list)
        next_root = None
        if prev_feature_value is not None:
            root[prev_feature_value] = {max_info_feature: tree}
            next_root = root[prev_feature_value][max_info_feature]
        else:
            root[max_info_feature] = tree
            next_root = root[max_info_feature]
        for node, branch in list(next_root.items()):
            if branch == "?":
                feature_value_data =
updated_train_data[updated_train_data[max_info_feature] == node]
                make_tree(next_root, node, feature_value_data, label, class_list)

# ID3 entry point
def id3(train_data_m, label):
    train_data = train_data_m.copy()
    tree = {}
    class_list = train_data[label].unique()
    make_tree(tree, None, train_data, label, class_list)
    return tree
```

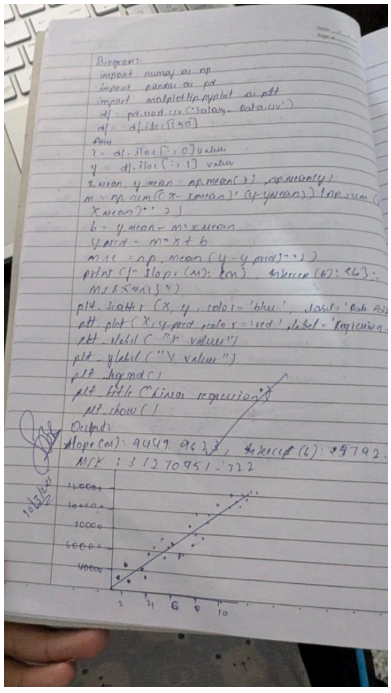
```
# Prediction function for a single instance
def predict(tree, instance):
    if not isinstance(tree, dict):
        return tree
    root_node = next(iter(tree))
    feature_value = instance[root_node]
    if feature_value in tree[root_node]:
        return predict(tree[root_node][feature_value], instance)
    else:
        return None

# Evaluate the decision tree
def evaluate(tree, test_data_m, label):
    correct_predict = 0
    wrong_predict = 0
    for index, row in test_data_m.iterrows():
        result = predict(tree, row)
        actual = row[label]
        if result == actual:
            correct_predict += 1
        else:
            wrong_predict += 1
    total = correct_predict + wrong_predict
    accuracy = correct_predict / total if total != 0 else 0
    return accuracy

# Build and evaluate tree
tree = id3(train_data_m, 'Play Tennis')
test_data_m = pd.read_csv("/content/PlayTennis.csv")
accuracy = evaluate(tree, test_data_m, 'Play Tennis')
print("Accuracy:", accuracy)
print(tree)
```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots





Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing

# Load the California Housing dataset
california_housing = fetch_california_housing()

# Assign the data (features) and target (house prices)
X = pd.DataFrame(california_housing.data,
columns=california_housing.feature_names)
y = pd.Series(california_housing.target)

# Select features for Linear Regression
X = X[['MedInc', 'AveRooms']]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print the actual vs predicted values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print("Linear Regression Results:")
print(results.head())
```

```
Linear Regression Results:
        Actual  Predicted
20046  0.47700   1.162302
3024   0.45800   1.499135
15663  5.00001   1.955731
20484  2.18600   2.852755
9814   2.78000   2.001677
```

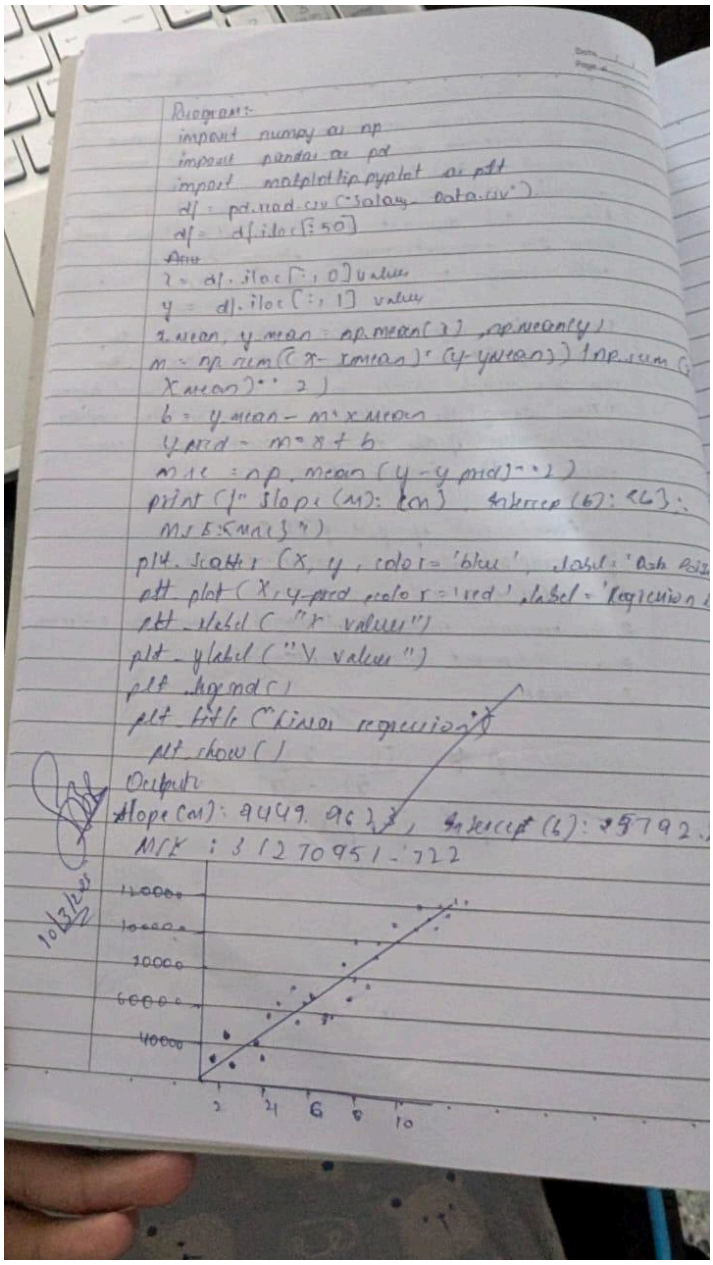
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing
california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data,
columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
X = X[['MedInc', 'AveRooms']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
# Print the actual vs predicted values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

```
        Actual  Predicted
20046  0.47700   1.162302
3024   0.45800   1.499135
15663  5.00001   1.955731
20484  2.18600   2.852755
9814   2.78000   2.001677
```

Program 6

Build Logistic Regression Model for a given dataset

Screenshots



Code

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris

# Load the Iris dataset
```

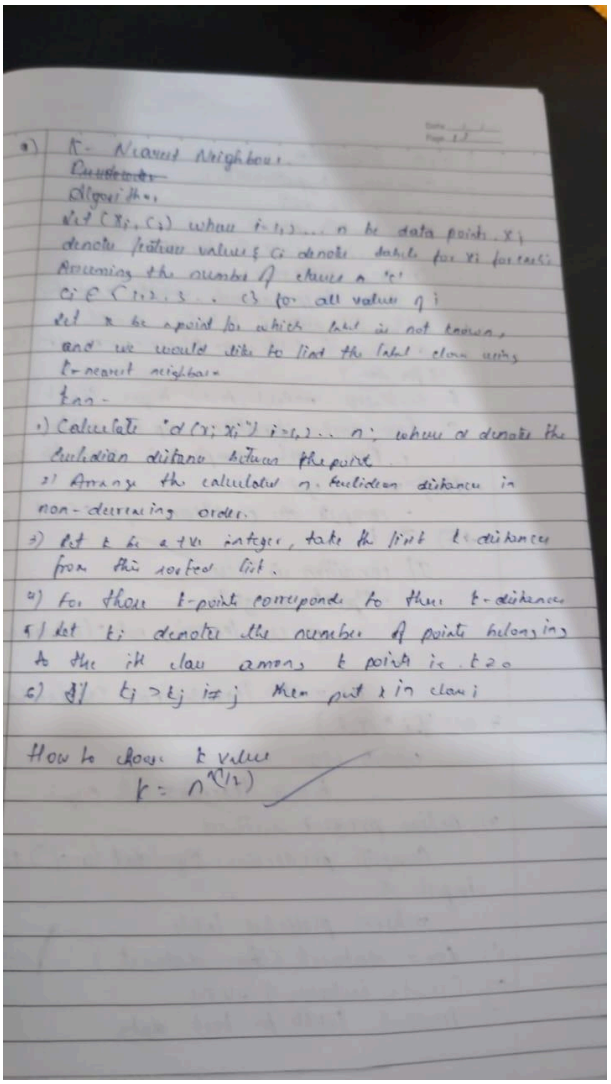
```
iris = load_iris()

# Assign the data (features) and target (species)
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# For simplicity, we will classify only two classes (0 and 1)
X = X[y.isin([0, 1])]  # Select only classes 0 and 1
y = y[y.isin([0, 1])]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print the actual vs predicted values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print("Logistic Regression Results:")
print(results.head())
```

```
Logistic Regression Results:
    Actual  Predicted
83       1          1
53       1          1
70       1          1
45       0          0
44       0          0
```

Program 6

Build KNN Classification model for a given dataset

Screenshots



Code

```
import math
from collections import Counter
import pandas as pd
from sklearn.model_selection import train_test_split

def euclidean_distance(point1, point2):
    return math.sqrt(sum((x - y) ** 2 for x, y in zip(point1, point2)))

class KNN:
    def __init__(self, k=3):
```

```
        self.k = k
        self.X_train = []
        self.y_train = []

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        return [self._predict(x) for x in X]

    def _predict(self, x):
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]
        k_indices = sorted(range(len(distances)), key=lambda i: distances[i])[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

    def score(self, X, y):
        predictions = self.predict(X)
        return sum(pred == true for pred, true in zip(predictions, y)) / len(y)


if __name__ == "__main__":

    from sklearn.datasets import load_iris
    iris = load_iris()
    df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
    df['target'] = iris.target

    X = df.iloc[:, :-1].values.tolist()
    y = df['target'].tolist()
```

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

model = KNN(k=3)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
print("Predictions:", predictions)
print("Accuracy:", model.score(X_test, y_test))
```

```
Predictions: [1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2, 0, 2, 2, 2, 2, 2, 0, 0]
Accuracy: 1.0
```

Program 7

Build Support vector machine model for a given dataset

Screenshots

Code

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
```

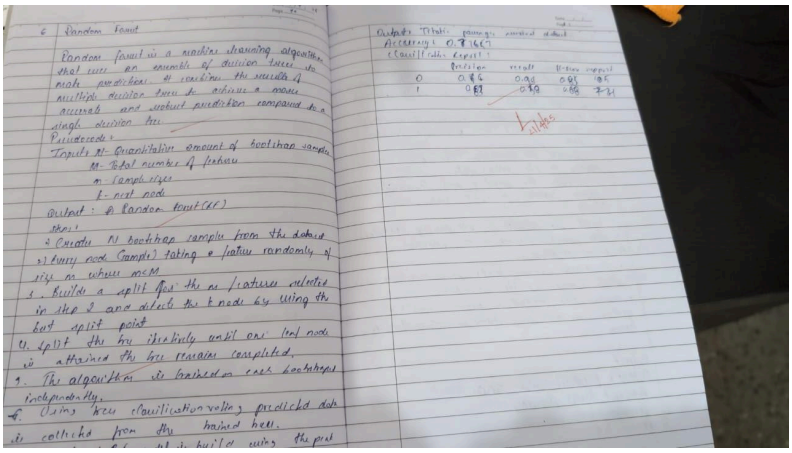
```
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) + self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y_[idx]))
                    self.b -= self.lr * y_[idx]

    def predict(self, X):
        approx = np.dot(X, self.w) + self.b
        return np.sign(approx)

if __name__ == "__main__":
    from sklearn.datasets import load_iris

    iris = load_iris()
    df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
    df['target'] = iris.target
    df = df[df['target'] != 2]

    X = df.iloc[:, :2].values
    y = df['target'].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    model = SVM()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)

    acc = np.mean(predictions == np.where(y_test == 0, -1, 1))
    print("Predictions:", predictions)
    print("Accuracy:", acc)
```

```
Predictions: [ 1.  1.  1. -1. -1. -1. -1.  1. -1. -1. -1. -1.  1. -1.  1. -1.  1.  1.
 -1. -1.]
Accuracy: 1.0
```

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshots



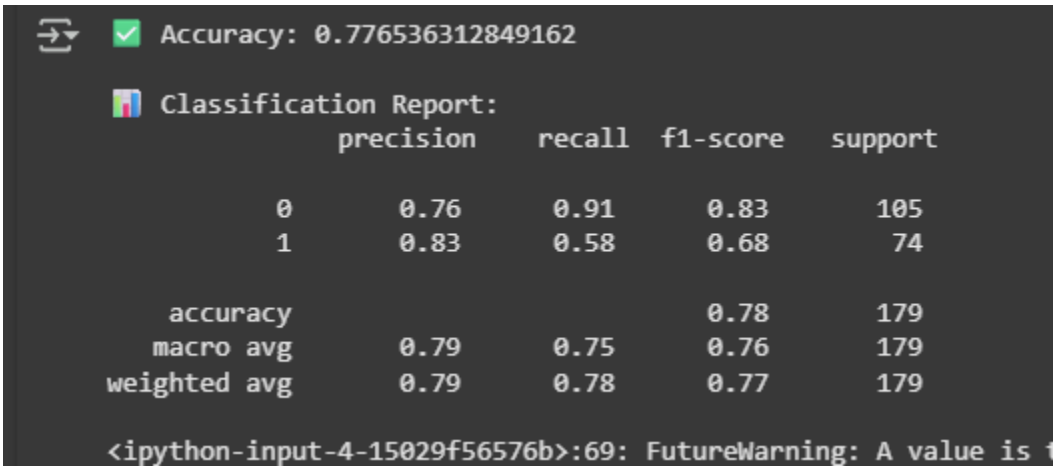
Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
from collections import Counter
import random

# ------------------------
# Random Forest From Scratch
# ------------------------
class CustomRandomForest:
    def __init__(self, n_estimators=10, max_features='sqrt', max_depth=None):
        self.n_estimators = n_estimators
        self.max_features = max_features
        self.max_depth = max_depth
        self.trees = []
```

```
def _bootstrap_sample(self, X, y):
    indices = np.random.choice(len(X), size=len(X), replace=True)
    return X.iloc[indices], y.iloc[indices]

def _get_max_features(self, n_features):
    if self.max_features == 'sqrt':
        return int(np.sqrt(n_features))
    elif self.max_features == 'log2':
        return int(np.log2(n_features))
    elif isinstance(self.max_features, int):
        return self.max_features
    else:
        return n_features

def fit(self, X, y):
    self.trees = []
    for _ in range(self.n_estimators):
        X_sample, y_sample = self._bootstrap_sample(X, y)
        max_feats = self._get_max_features(X.shape[1])
        features = random.sample(list(X.columns), max_feats)

        tree = DecisionTreeClassifier(max_depth=self.max_depth)
        tree.fit(X_sample[features], y_sample)
        self.trees.append((tree, features))

def predict(self, X):
    tree_preds = []
    for tree, features in self.trees:
        preds = tree.predict(X[features])
        tree_preds.append(preds)

    tree_preds = np.array(tree_preds).T
```

```
        final_preds = [Counter(row).most_common(1)[0][0] for row in tree_preds]
        return np.array(final_preds)

# ------------------------
# Load and Preprocess Titanic Dataset
# ------------------------
df = pd.read_csv('train.csv')

# Feature engineering
df['Title'] = df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
df['Title'] = df['Title'].replace(['Lady', 'Countess','Capt','Col','Don',
                                   'Dr','Major','Rev','Sir','Jonkheer','Dona'], 'Rare')
df['Title'] = df['Title'].replace('Mlle', 'Miss')
df['Title'] = df['Title'].replace('Ms', 'Miss')
df['Title'] = df['Title'].replace('Mme', 'Mrs')

# Fill missing values
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
df['Fare'].fillna(df['Fare'].median(), inplace=True)

# Drop unused columns
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

# Encode categoricals
for col in ['Sex', 'Embarked', 'Title']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

# Features and target
X = df.drop('Survived', axis=1)
y = df['Survived']
```

```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# -------------------------
# Train and Evaluate Custom Random Forest
# -------------------------
model = CustomRandomForest(n_estimators=20, max_features='sqrt', max_depth=7)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("✅ Accuracy:", accuracy_score(y_test, y_pred))
print("\n📊 Classification Report:\n", classification_report(y_test, y_pred))
```

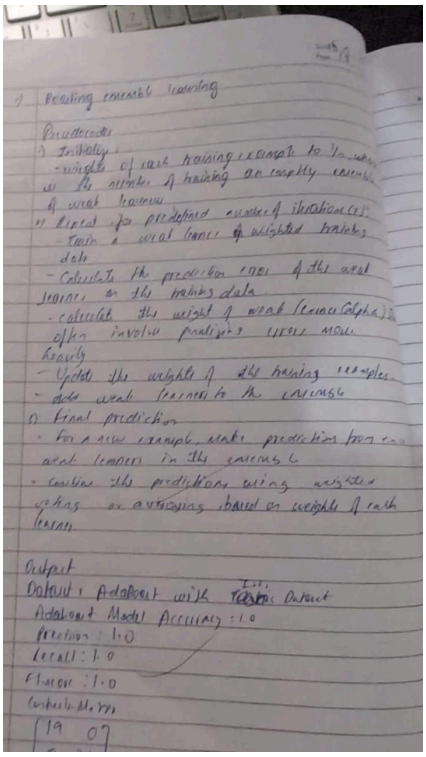


```
✅ Accuracy: 0.776536312849162

📊 Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.91      0.83       105
           1       0.83      0.58      0.68        74

    accuracy                           0.78       179
   macro avg       0.79      0.75      0.76       179
weighted avg       0.79      0.78      0.77       179

<ipython-input-4-15029f56576b>:69: FutureWarning: A value is t
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshots



Code

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

# Function to calculate weighted accuracy
def calculate_weighted_error(y_true, y_pred, weights):
    return np.sum(weights * (y_true != y_pred)) / np.sum(weights)

# Function to update weights
def update_weights(weights, alpha, y_true, y_pred):
    return weights * np.exp(alpha * (y_true != y_pred).astype(float))

# AdaBoost implementation
def adaboost(X, y, n_estimators):
    n_samples, n_features = X.shape
    weights = np.ones(n_samples) / n_samples
    estimators = []
```

```
    alphas = []

    for _ in range(n_estimators):
        # Train a weak learner (Decision Stump)
        best_feature, best_threshold, best_polarity, best_error = None, None, None,
float('inf')

        for feature in range(n_features):
            thresholds = np.unique(X[:, feature])

            for threshold in thresholds:
                for polarity in [1, -1]:
                    y_pred = np.ones(n_samples)
                    y_pred[polarity * X[:, feature] < polarity * threshold] = -1

                    error = calculate_weighted_error(y, y_pred, weights)

                    if error < best_error:
                        best_feature = feature
                        best_threshold = threshold
                        best_polarity = polarity
                        best_error = error

        # Calculate alpha (model weight)
        alpha = 0.5 * np.log((1 - best_error) / (best_error + 1e-10))

        # Update weights
        y_pred = np.ones(n_samples)
        y_pred[best_polarity * X[:, best_feature] < best_polarity * best_threshold] = -1
        weights = update_weights(weights, alpha, y, y_pred)

        estimators.append((best_feature, best_threshold, best_polarity))
        alphas.append(alpha)
```

```
    return estimators, alphas

# Prediction function
def predict(X, estimators, alphas):
    n_samples = X.shape[0]
    final_prediction = np.zeros(n_samples)

    for (feature, threshold, polarity), alpha in zip(estimators, alphas):
        prediction = np.ones(n_samples)
        prediction[polarity * X[:, feature] < polarity * threshold] = -1
        final_prediction += alpha * prediction

    return np.sign(final_prediction)

# Load dataset
iris = pd.read_csv('/Iris.csv')

# Prepare features and target
X = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values
y = iris['Species']

# Convert target to numerical labels
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
y = np.where(y == 0, -1, 1)  # Convert labels to -1 and 1

# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train AdaBoost
```

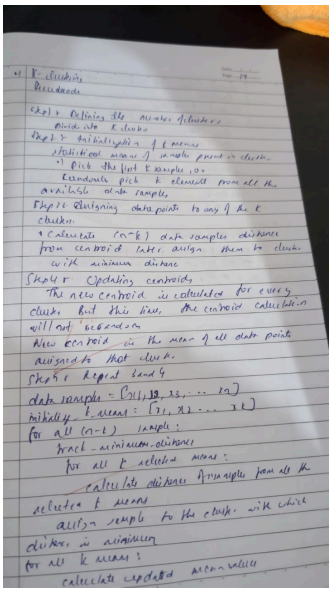
```
n_estimators = 50
estimators, alphas = adaboost(X_train, y_train, n_estimators)

# Make predictions
y_pred = predict(X_test, estimators, alphas)

# Evaluate
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
conf_matrix = confusion_matrix(y_test, y_pred)

print("AdaBoost Model Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

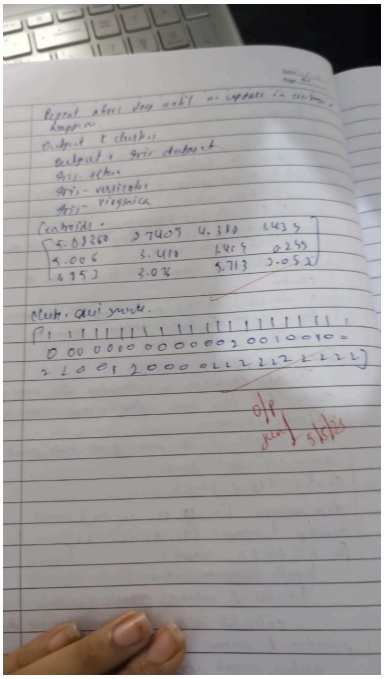
```
AdaBoost Model Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
Confusion Matrix:
 [[19  0]
 [ 0 26]]
Classification Report:
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        26

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Program 10

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

Screenshots





Code

```
import numpy as np
import pandas as pd
from sklearn.metrics import silhouette_score

# K-Means Clustering Implementation
def kmeans(X, n_clusters, max_iters=300, tol=1e-4):
    n_samples, n_features = X.shape
```

```
    # Randomly initialize cluster centers
    rng = np.random.default_rng(seed=42)
    centroids = X[rng.choice(n_samples, n_clusters, replace=False)]

    for _ in range(max_iters):
        # Assign samples to nearest centroid
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
        cluster_assignments = np.argmin(distances, axis=1)

        # Calculate new centroids
        new_centroids = np.array([X[cluster_assignments == k].mean(axis=0) for k in
range(n_clusters)])

        # Check for convergence
        if np.linalg.norm(new_centroids - centroids) < tol:
            break

        centroids = new_centroids

    return centroids, cluster_assignments

# Load dataset
iris = pd.read_csv('/Iris.csv')

# Prepare features
X = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values

# Number of clusters
n_clusters = 3

# Apply K-Means
centroids, cluster_assignments = kmeans(X, n_clusters)
```

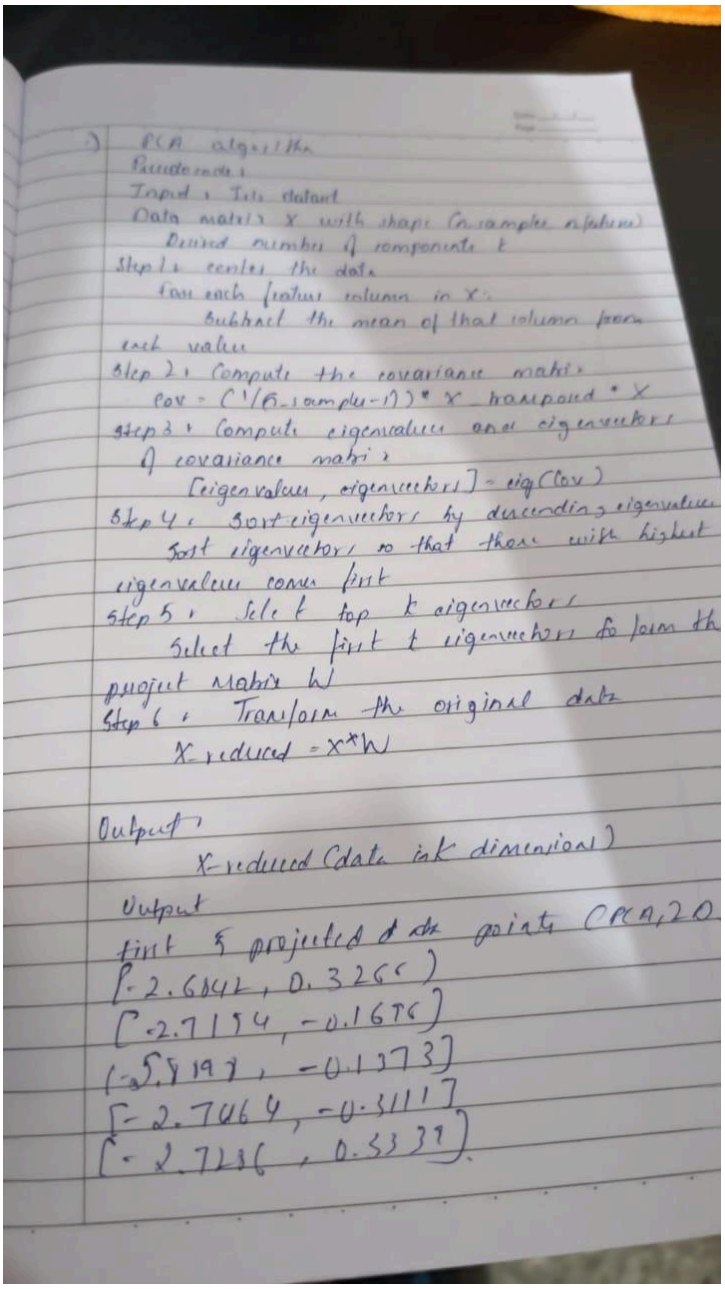
```
# Evaluate clustering using silhouette score
silhouette_avg = silhouette_score(X, cluster_assignments)

# Print results
print("Centroids:")
print(centroids)
print("\nCluster Assignments:")
print(cluster_assignments)
print("\nSilhouette Score:", silhouette_avg)

# Add cluster assignments to the original dataset
iris['Cluster'] = cluster_assignments
print("\nDataset with Clusters:")
print(iris.head())
```

```
Centroids:
[[5.88360656 2.74098361 4.38852459 1.43442623]
 [5.006      3.418      1.464      0.244     ]
 [6.85384615 3.07692308 5.71538462 2.05384615]]

Cluster Assignments:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 0 2 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 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]

Silhouette Score: 0.5509643746707443

Dataset with Clusters:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

   Cluster
0        1
1        1
2        1
3        1
4        1
```

Program 11

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

Screenshots



Code

```
# PCA implementation from scratch using only built-in Python functions

# Step 1: Load the dataset
import csv

file_path = "Iris.csv"  # Change path if needed

with open(file_path, "r") as file:
```

```
    reader = csv.reader(file)
    data = list(reader)

header = data[0]
rows = data[1:]

# Step 2: Extract only numerical features (columns 1 to 4)
features = []
for row in rows:
    features.append([float(row[1]), float(row[2]), float(row[3]), float(row[4])])

# Step 3: Mean centering
def mean_center(data):
    n = len(data)
    d = len(data[0])
    mean = [0.0] * d
    for i in range(n):
        for j in range(d):
            mean[j] += data[i][j]
    for j in range(d):
        mean[j] /= n
    centered = []
    for i in range(n):
        centered.append([data[i][j] - mean[j] for j in range(d)])
    return centered, mean

centered_data, mean_vector = mean_center(features)

# Step 4: Compute covariance matrix
def compute_covariance_matrix(data):
    n = len(data)
    d = len(data[0])
    cov_matrix = [[0.0 for _ in range(d)] for _ in range(d)]
```

```
    for i in range(d):
        for j in range(d):
            for k in range(n):
                cov_matrix[i][j] += data[k][i] * data[k][j]
            cov_matrix[i][j] /= (n - 1)
    return cov_matrix

cov_matrix = compute_covariance_matrix(centered_data)

# Step 5: Eigenvalue and eigenvector using power iteration
def dot(v1, v2):
    return sum(x * y for x, y in zip(v1, v2))

def mat_vec_mult(mat, vec):
    return [sum(mat[i][j] * vec[j] for j in range(len(vec))) for i in range(len(mat))]

def norm(vec):
    return sum(x * x for x in vec) ** 0.5

def normalize(vec):
    n = norm(vec)
    return [x / n for x in vec]

def power_iteration(mat, num_iter=1000):
    b_k = [1.0 for _ in range(len(mat))]
    for _ in range(num_iter):
        b_k1 = mat_vec_mult(mat, b_k)
        b_k = normalize(b_k1)
    eigenvalue = dot(b_k, mat_vec_mult(mat, b_k)) / dot(b_k, b_k)
    return eigenvalue, b_k

def deflate_matrix(mat, eigenvalue, eigenvector):
    d = len(mat)
```

```
    for i in range(d):
        for j in range(d):
            mat[i][j] -= eigenvalue * eigenvector[i] * eigenvector[j]
    return mat

# Step 6: Get top 2 eigenvectors (principal components)
eigvals = []
eigvecs = []
cov_copy = [row[:] for row in cov_matrix]

for _ in range(2):
    val, vec = power_iteration(cov_copy)
    eigvals.append(val)
    eigvecs.append(vec)
    cov_copy = deflate_matrix(cov_copy, val, vec)

# Step 7: Project data
def project_data(data, components):
    projected = []
    for row in data:
        proj = [dot(row, comp) for comp in components]
        projected.append(proj)
    return projected

projected_data = project_data(centered_data, eigvecs)

# Step 8: Print first 5 projected 2D data points
print("First 5 projected data points (PCA 2D):")
for point in projected_data[:5]:
    print([round(x, 4) for x in point])
```

```
First 5 projected data points (PCA 2D):
[-2.6842, 0.3266]
[-2.7154, -0.1696]
[-2.8898, -0.1373]
[-2.7464, -0.3111]
[-2.7286, 0.3339]
```