# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT**
**on**

# Machine Learning (23CS6PCMAL)

***Submitted by***

**Bhuvana M(1BM22CS071)**

***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 **Bhuvana M(1BM22CS071),** who is a 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.

| | |
|---|---|
| **Lab Faculty Incharge**<br><br>Name:**Megha J**<br>Assistant Professor<br>Department of CSE, BMSCE | **Dr. Kavitha Sooda**<br>Professor & HOD<br>Department of CSE, BMSCE |

# Index



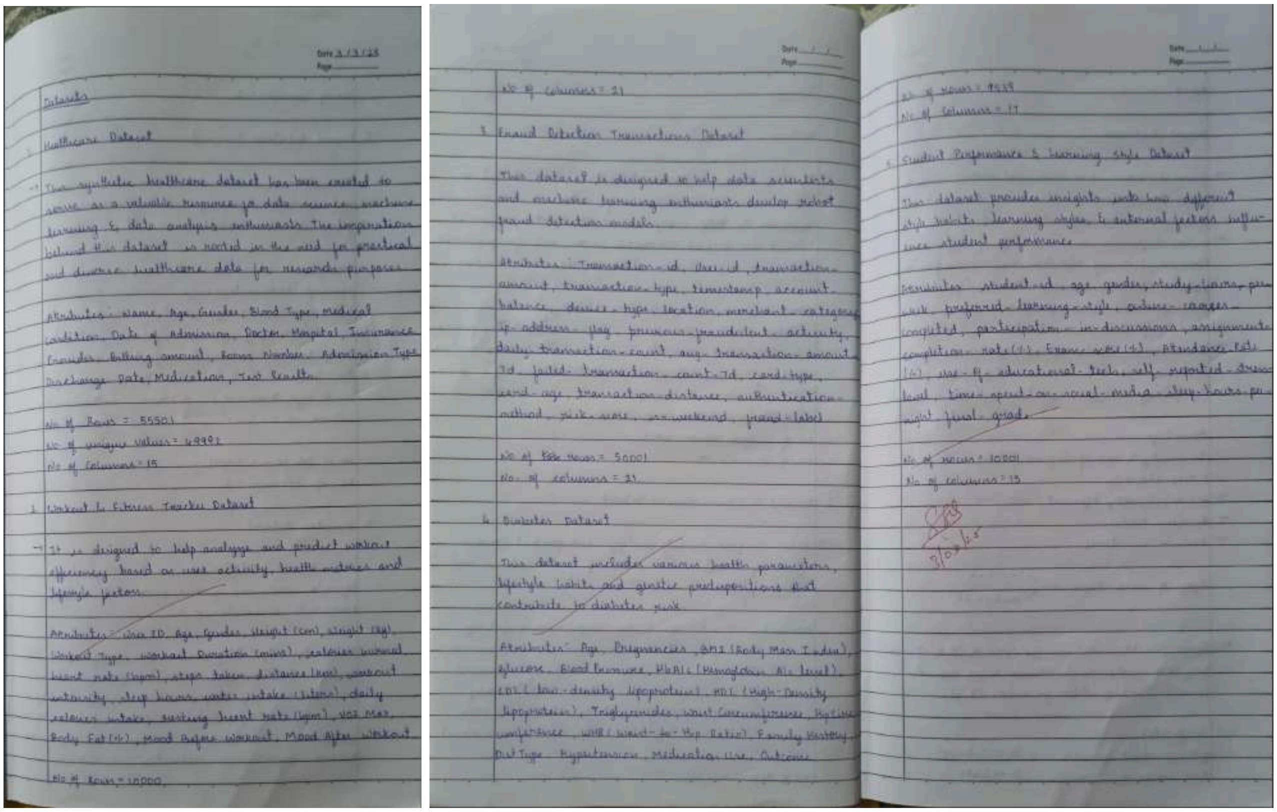
**Github Link:**

https://github.com/Bhu06/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)

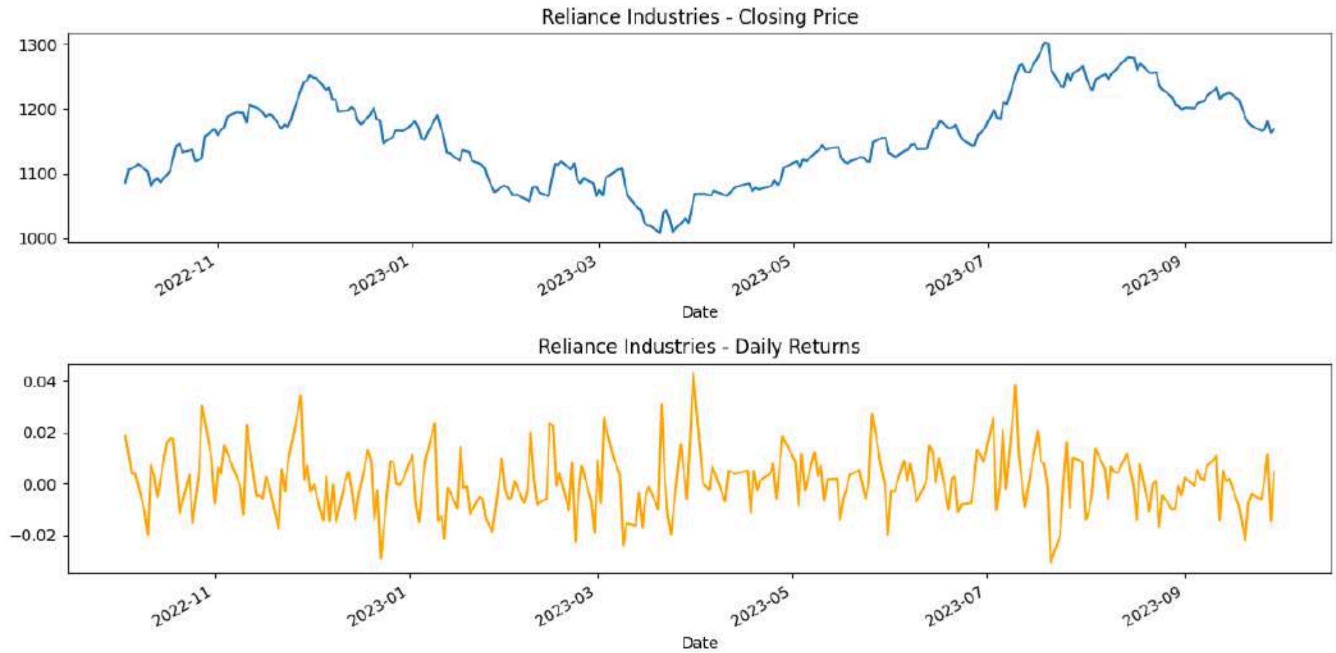
reliance_data['Close'].plot(title="Reliance Industries - Closing Price")

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)

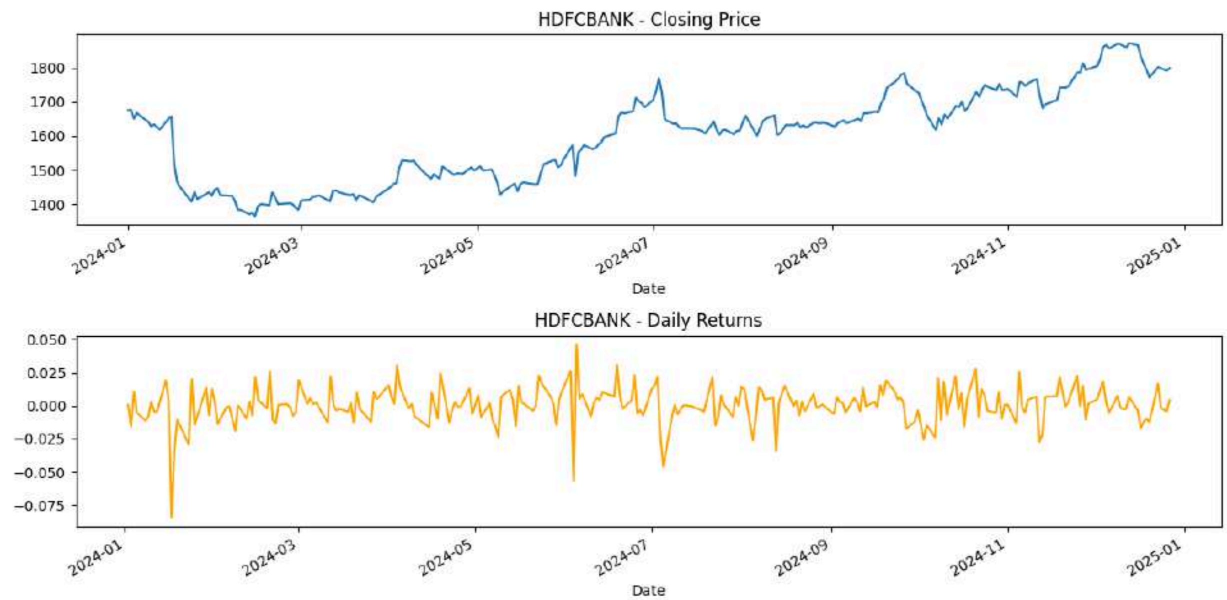
HDFC['Close'].plot(title="HDFCBANK - Closing Price")

plt.subplot(2, 1, 2)

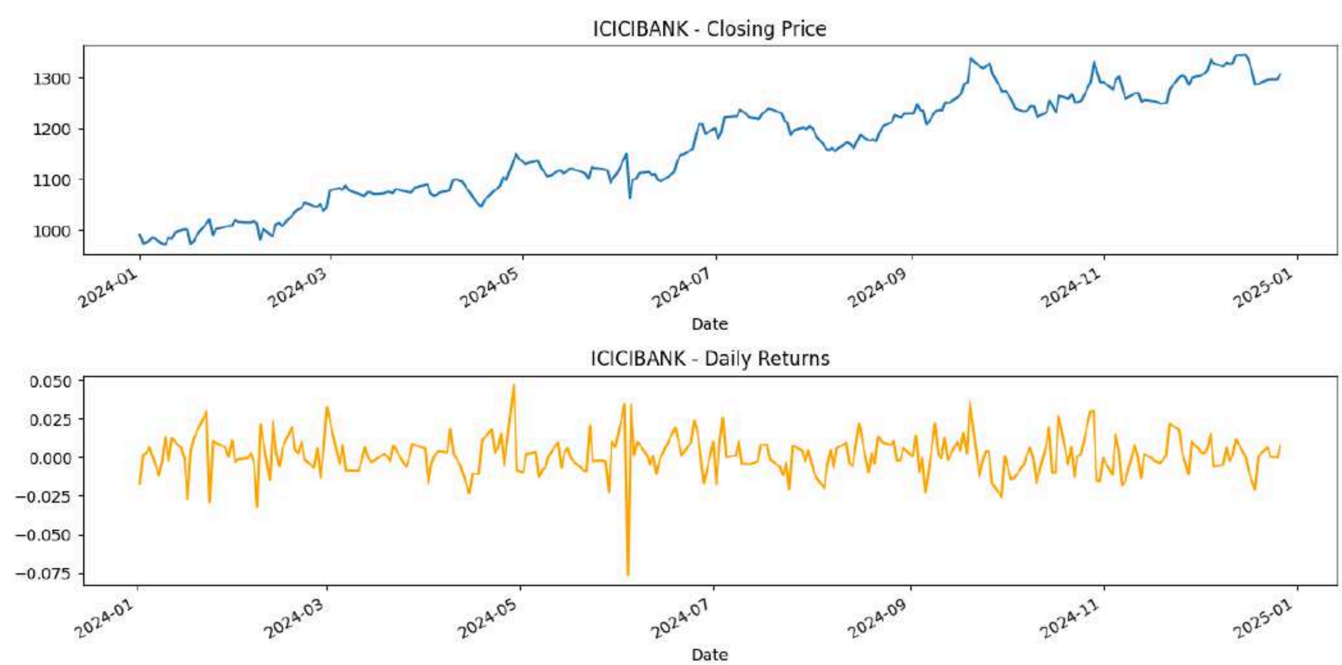
HDFC['Daily Return'].plot(title="HDFCBANK - Daily Returns", color='orange')

plt.tight_layout()

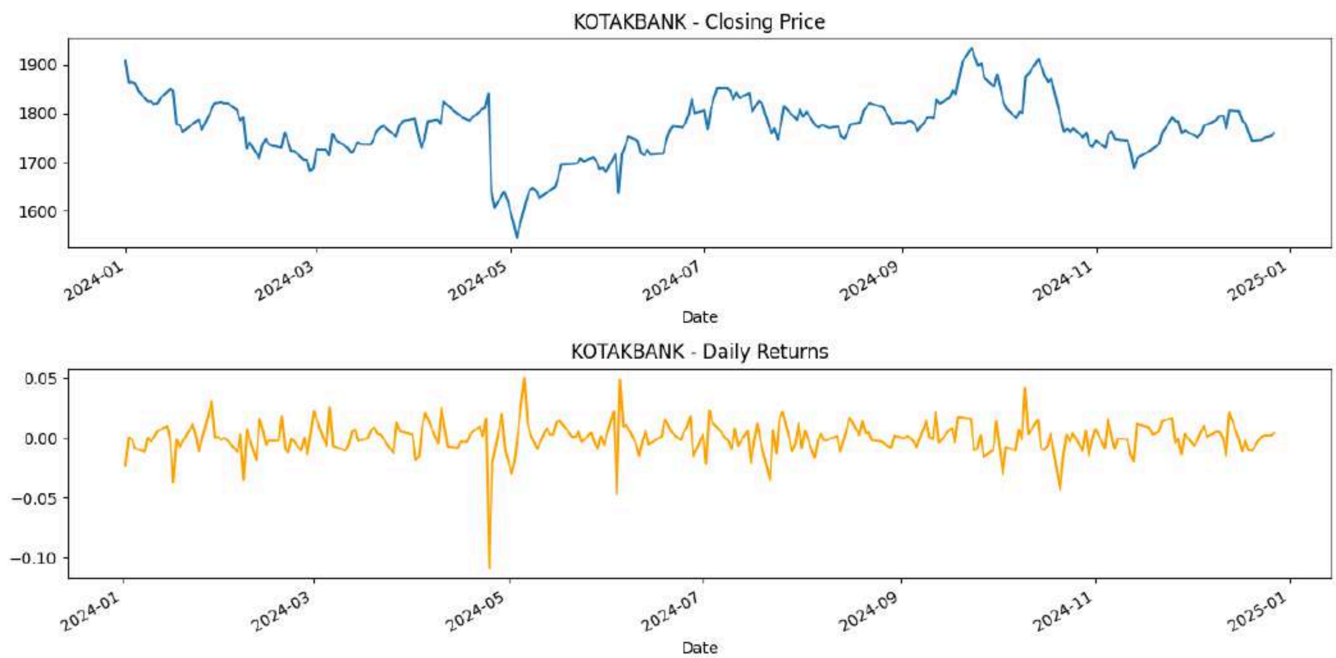
plt.show()




```
# 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()
```



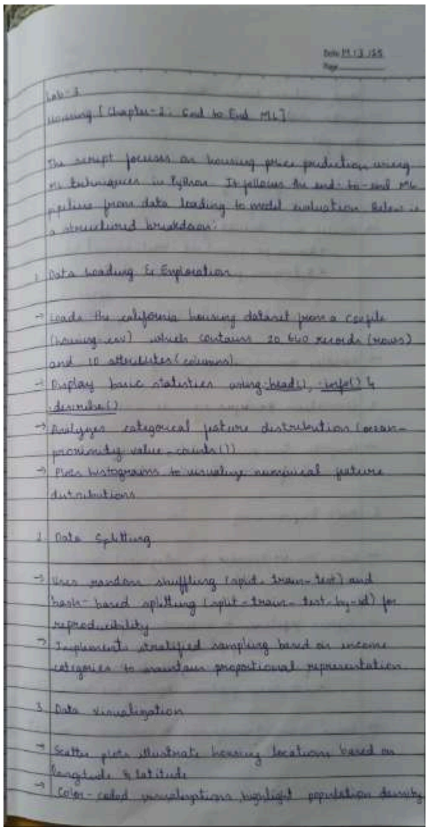

```
# Step 4: Saving the Processed Data to a New CSV File
# Save the Reliance data to a CSV file
HDFC.to_csv('HDFC.csv')
ICICI.to_csv('ICICI.csv')
KOTAKBANK.to_csv('KOTAKBANK.csv')
print("\nSAVED")
```

```
SAVED
```

# Program 2

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

**Screenshots**



Date 11 / 3 / 25

Lab-2

Housing [Chapter-2: End to End ML]

The script focuses on housing price prediction using ML techniques in Python. It follows the end-to-end ML pipeline from data loading to model evaluation. Below is a structured breakdown:

1. Data Loading & Exploration

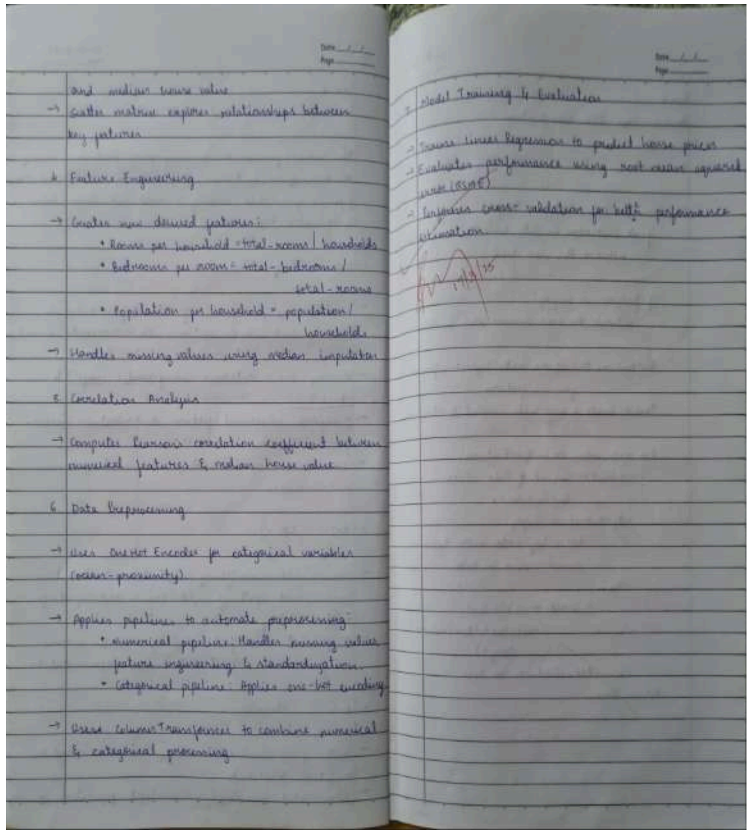
→ Loads the California housing dataset from a csv file (housing.csv) which contains 20 640 records (rows) and 10 attributes (columns).
→ Display basic statistics using head(), info() & describe()
→ Analyzes categorical feature distribution (ocean-proximity value_counts())
→ Plots histograms to visualize numerical feature distributions

2. Data Splitting

→ Uses random shuffling (split_train_test) and hash-based splitting (split_train_test_by_id) for reproducibility
→ Implements stratified sampling based on income categories to maintain proportional representation

3. Data Visualization

→ Scatter plots illustrate housing locations based on longitude & latitude
→ Color-coded visualizations highlight population density



and median house value
→ Scatter matrix explores relationships between key features

4. Feature Engineering

→ Creates new derived features:
• Rooms per household = total_rooms / households
• Bedrooms per room = total_bedrooms / total_rooms
• Population per household = population / households
→ Handles missing values using median imputation

5. Correlation Analysis

→ Computes Pearson correlation coefficient between numerical features & median house value

6. Data Preprocessing

→ Uses OneHotEncoder for categorical variables (ocean-proximity)
→ Applies pipelines to automate preprocessing
• numerical pipeline: Handles missing values, feature engineering & standardization
• Categorical pipeline: Applies one-hot encoding
→ Uses ColumnTransformer to combine numerical & categorical processing

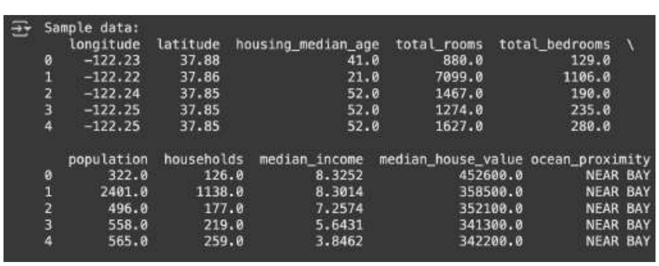
7. Model Training & Evaluation

→ Trains Linear Regression to predict house prices
→ Evaluates performance using root mean squared error (RMSE)
→ Employs cross-validation for better performance estimation

11/3/25

**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:")

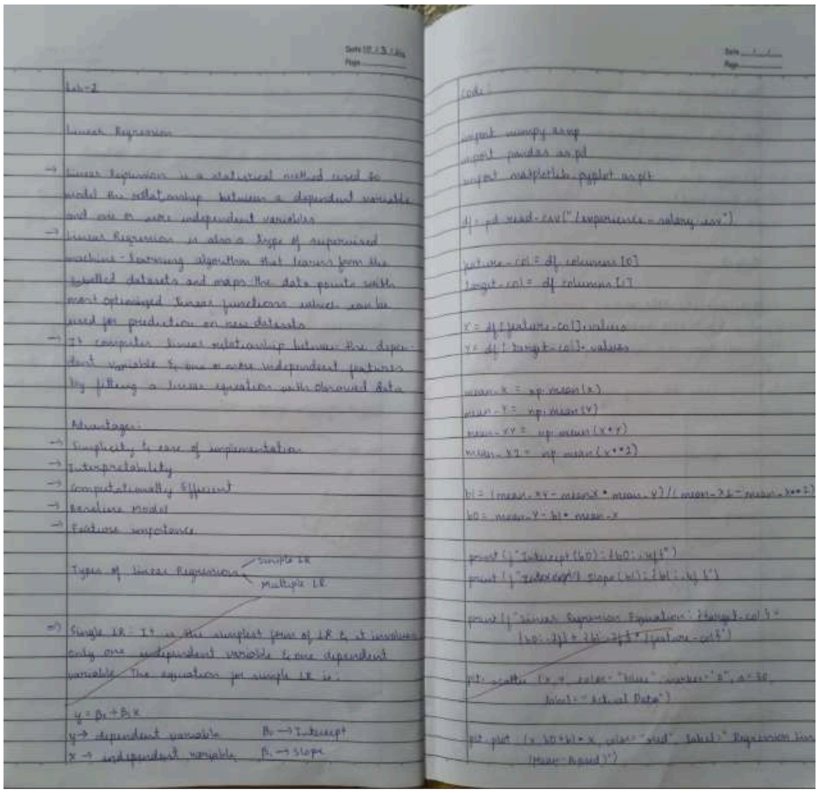
print(df_encoded.head())

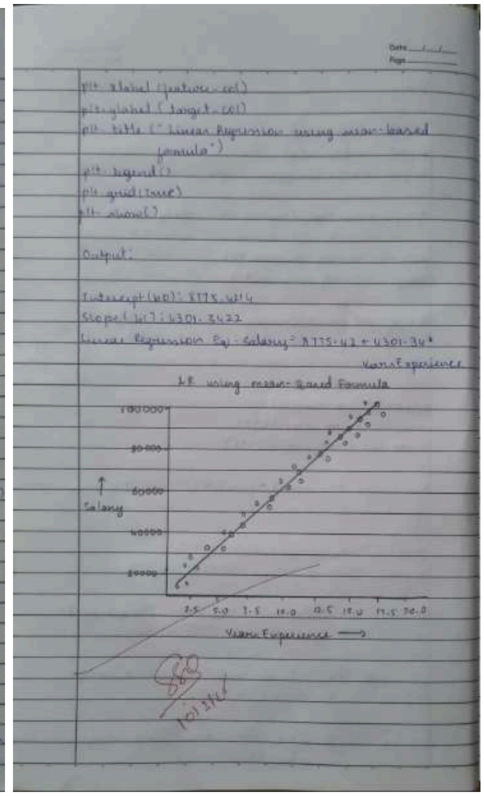
```
Encoded DataFrame:
   age  fnlwgt  educational-num  gender  capital-gain  capital-loss  \
0   25  226802                7     1.0             0             0
1   38   89814                9     1.0             0             0
2   28  336951               12     1.0             0             0
3   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]
```

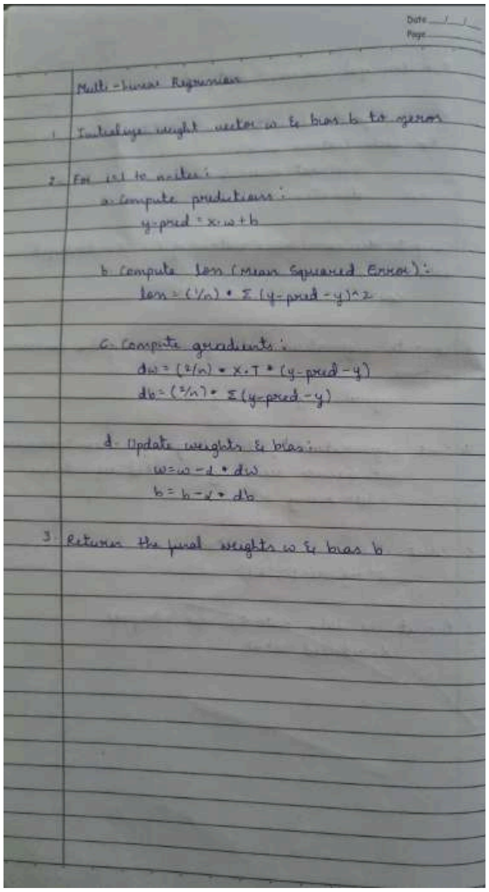
## **Program 3**

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

### Screenshots





Date / /
Page

Multi-Linear Regression

1. Initialize weight vector w & bias b to zeros

2. For i=1 to n_iter:

a. Compute predictions:

y_pred = X.w + b

b. Compute loss (Mean Squared Error):

loss = (1/n) * Σ (y_pred - y)^2

c. Compute gradients:

dw = (2/n) * X.T * (y_pred - y)

db = (2/n) * Σ (y_pred - y)

d. Update weights & bias:

w = w - α * dw

b = b - α * db

3. Return the final weights w & bias b

**Code:**

**Linear Regression:**

```
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
```

**Multiple Regression:**

```
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)
```

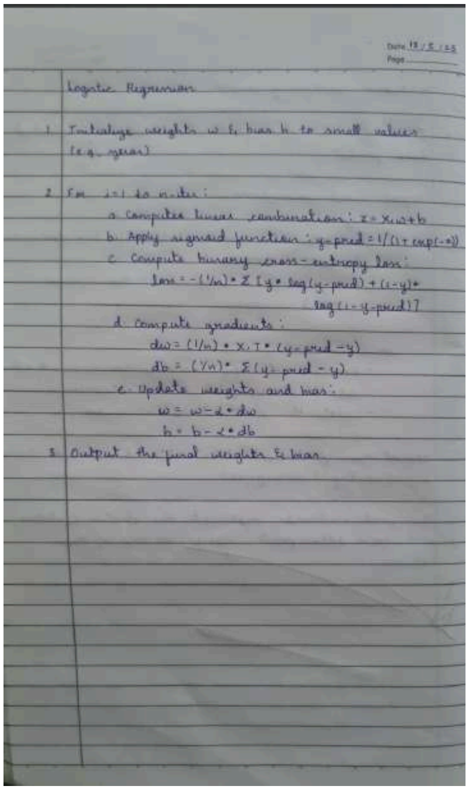
```
y_pred = model.predict(X_test)
# Print the actual vs predicted values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

```
        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 4

Build Logistic Regression Model for a given dataset

**Screenshots**


Date 13/5/25

Logistic Regression

1. Initialize weights w & bias b to small values (e.g. zeros)

2. For i=1 to n_iter:
   a. Compute linear combination: z = X.w+b
   b. Apply sigmoid function: y_pred = 1/(1+exp(-z))
   c. Compute binary cross-entropy loss:
      loss = -(1/n) * Σ [y * log(y_pred) + (1-y) * log(1-y_pred)]
   d. Compute gradients:
      dw = (1/n) * X.T * (y_pred - y)
      db = (1/n) * Σ(y_pred - y)
   e. Update weights and bias:
      w = w - α * dw
      b = b - α * db

3. Output the final weights & bias

**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
```

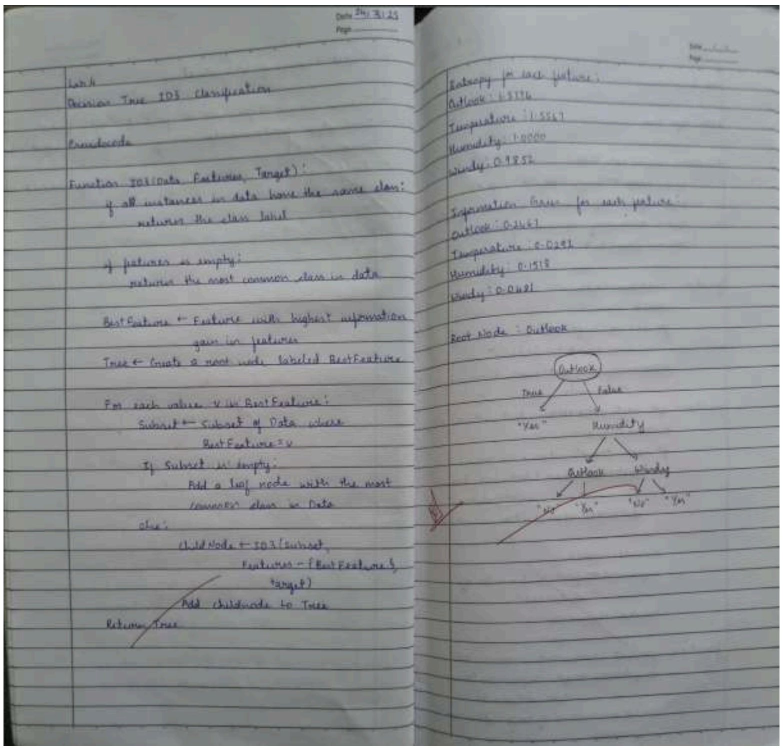
```
y = y[y.isin([0, 1])]
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create and train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# 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 5**

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

**Screenshots**



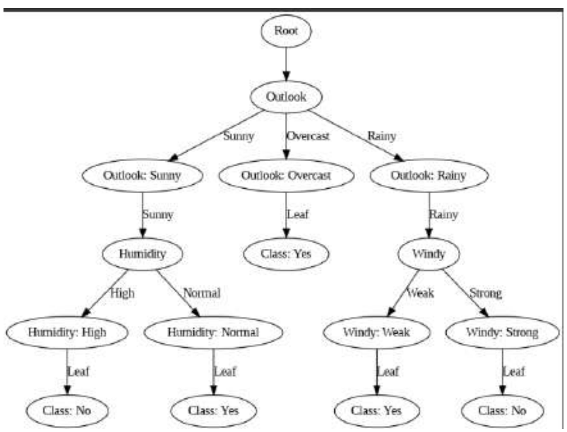

**Code:**

```
import pandas as pd
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny',
'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool','Mild', 'Cool','Mild', 'Mild', 'Mild',
'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'High'],
    'Windy': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
'Strong', 'Weak', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}
```

```
df = pd.DataFrame(data)
import math
def entropy(data):
    total = len(data)
    counts = data.value_counts()
    entropy_value = 0
    for count in counts:
        probability = count / total
        entropy_value -= probability * math.log2(probability)
    return entropy_value
def information_gain(data, feature, target):
    total_entropy = entropy(data[target])
    feature_values = data[feature].unique()
    weighted_entropy = 0
    for value in feature_values:
        subset = data[data[feature] == value]
        weighted_entropy += (len(subset) / len(data)) * entropy(subset[target])
    return total_entropy - weighted_entrop
def best_split(data, target):
    features = data.drop(columns=[target]).columns
    best_feature = None
    best_gain = -1
    for feature in features:
        gain = information_gain(data, feature, target)
        if gain > best_gain:
            best_gain = gain
            best_feature = feature
    return best_feature
def best_split(data, target):
    features = data.drop(columns=[target]).columns
    best_feature = None
    best_gain = -1
```

```
    for feature in features:
        gain = information_gain(data, feature, target)
        if gain > best_gain:
            best_gain = gain
            best_feature = feature
    return best_feature
# Build the decision tree using the dataset
tree = build_tree(df, target='PlayTennis')
print(tree)
```

```
{'Outlook': {'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}, 'Overcast': 'Yes', 'Rainy': {'Windy': {'Weak': 'Yes', 'Strong': 'No'}}}}
```




➔ **California Housing Prices (Splitting)**

```
import pandas as pd
housing = pd.read_csv("housing.csv")
housing.head()
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | median_income | median_house_value | ocean_proximity |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322.0 | 126.0 | 8.3252 | 452600.0 | NEAR BAY |
| 1 | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401.0 | 1138.0 | 8.3014 | 358500.0 | NEAR BAY |
| 2 | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496.0 | 177.0 | 7.2574 | 352100.0 | NEAR BAY |
| 3 | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558.0 | 219.0 | 5.6431 | 341300.0 | NEAR BAY |
| 4 | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565.0 | 259.0 | 3.8462 | 342200.0 | NEAR BAY |

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
housing["ocean_proximity"].value_counts()
```

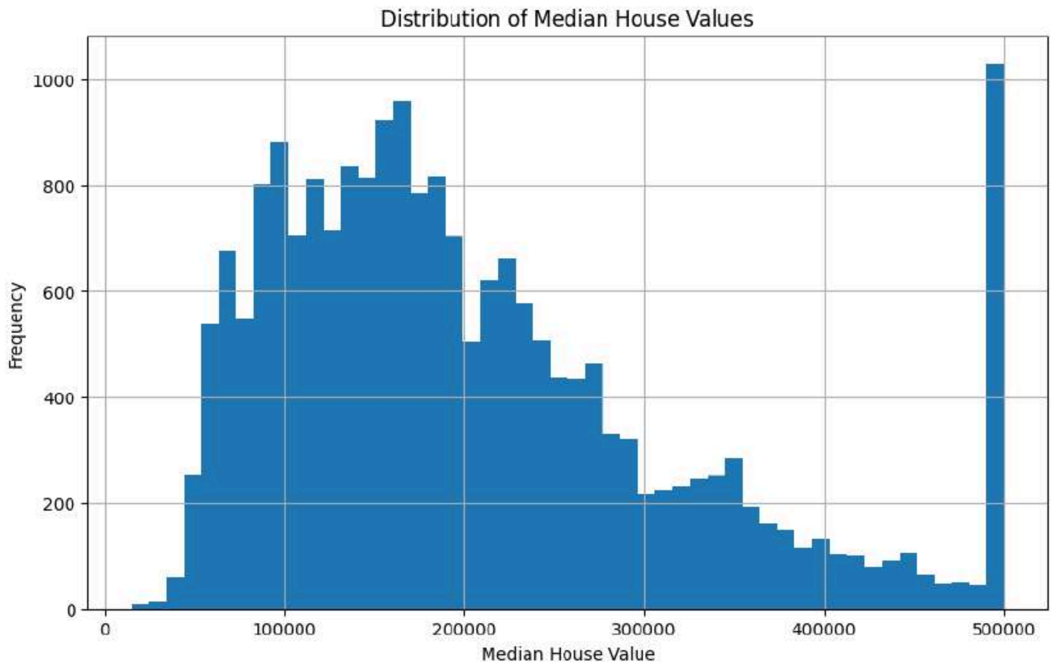
| ocean_proximity | count |
|---|---|
| <1H OCEAN | 9136 |
| INLAND | 6551 |
| NEAR OCEAN | 2658 |
| NEAR BAY | 2290 |
| ISLAND | 5 |

dtype: int64

```
housing.describe()
```

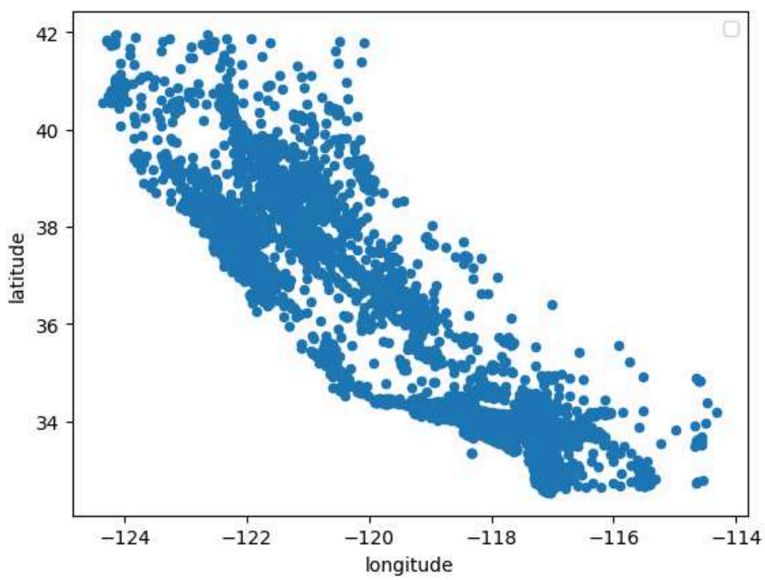
| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | median_income | median_house_value |
|---|---|---|---|---|---|---|---|---|---|
| count | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20433.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 |
| mean | -119.569704 | 35.631861 | 28.639486 | 2635.763081 | 537.870553 | 1425.476744 | 499.539680 | 3.870671 | 206855.816909 |
| std | 2.003532 | 2.135952 | 12.585558 | 2181.615252 | 421.385070 | 1132.462122 | 382.329753 | 1.899822 | 115395.615874 |
| min | -124.350000 | 32.540000 | 1.000000 | 2.000000 | 1.000000 | 3.000000 | 1.000000 | 0.499900 | 14999.000000 |
| 25% | -121.800000 | 33.930000 | 18.000000 | 1447.750000 | 296.000000 | 787.000000 | 280.000000 | 2.563400 | 119600.000000 |
| 50% | -118.490000 | 34.260000 | 29.000000 | 2127.000000 | 435.000000 | 1166.000000 | 409.000000 | 3.534800 | 179700.000000 |
| 75% | -118.010000 | 37.710000 | 37.000000 | 3148.000000 | 647.000000 | 1725.000000 | 605.000000 | 4.743250 | 264725.000000 |
| max | -114.310000 | 41.950000 | 52.000000 | 39320.000000 | 6445.000000 | 35682.000000 | 6082.000000 | 15.000100 | 500001.000000 |

```
import matplotlib.pyplot as plt
# Plot a histogram of median house values
housing['median_house_value'].hist(bins=50, figsize=(10, 6))
plt.xlabel("Median House Value")
plt.ylabel("Frequency")
plt.title("Distribution of Median House Values")
plt.show()
```

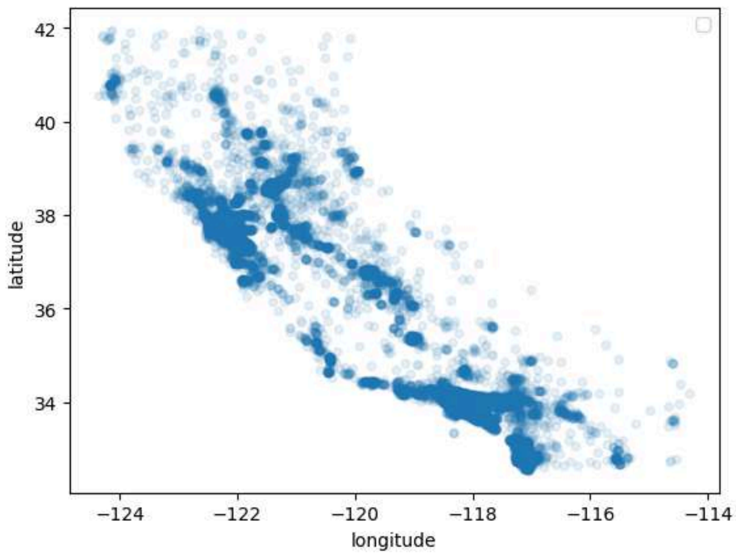



```
import pandas as pd
from sklearn.model_selection import train_test_split
housing = pd.read_csv("housing.csv")
# Random sampling
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
# Separate into features (X) and target (y)
X = housing.drop("median_house_value", axis=1)  # Features (all columns except the target)
y = housing["median_house_value"]  # Target variable
# Split into train and test sets with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
import pandas as pd
from sklearn.model_selection import train_test_split
# Separate into features (X) and target (y)
X = housing.drop("median_house_value", axis=1)  # Features (all columns except the target)
y = housing["median_house_value"]  # Target variable
# Create categories for the target variable
housing["income_cat"] = pd.cut(housing["median_house_value"],
                      bins=[0, 100000, 200000, 300000, 400000, np.inf],
```

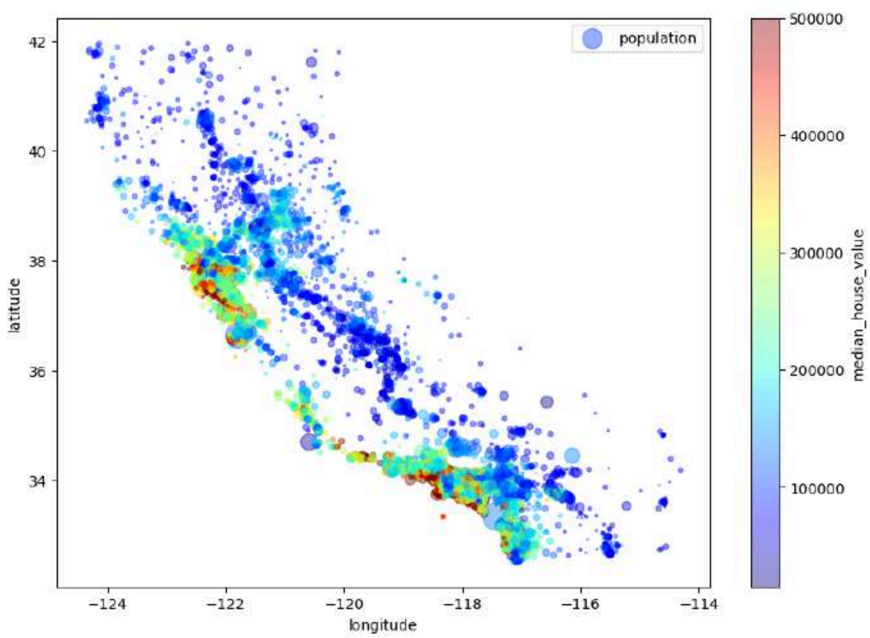
```
                    labels=[1, 2, 3, 4, 5])
# Split into train and test sets with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=housing["income_cat"])
import matplotlib.pyplot as plt
# Add the target variable back to the training set for visualization
train_set = X_train.copy()
train_set["median_house_value"] = y_train
# Plot the training set
train_set.plot(kind="scatter", x="longitude", y="latitude")
plt.legend()
```




```
import matplotlib.pyplot as plt
# Add the target variable back to the training set for visualization
train_set = X_train.copy()
train_set["median_house_value"] = y_train
# Plot the training set
train_set.plot(kind="scatter", x="longitude", y="latitude",alpha=0.1)
plt.legend()
```



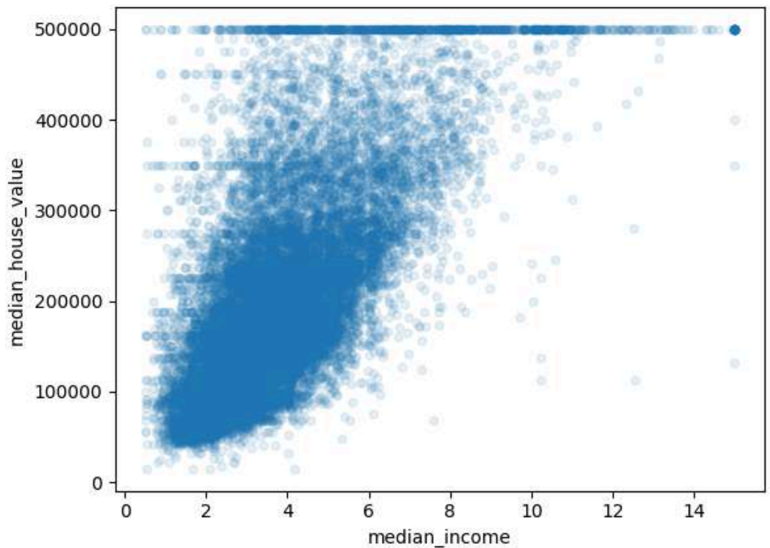

train_set.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4, s=train_set["population"]/100, label="population", figsize=(10,7), c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True)




```
# Select only numerical columns (excluding categorical columns like 'ocean_proximity')
numerical_columns = housing.select_dtypes(include=['float64', 'int64'])
# Calculate the correlation matrix
correlation_matrix = numerical_columns.corr()
# Display the correlation of 'median_house_value' with other numerical columns
print(correlation_matrix["median_house_value"].sort_values(ascending=False))
```

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

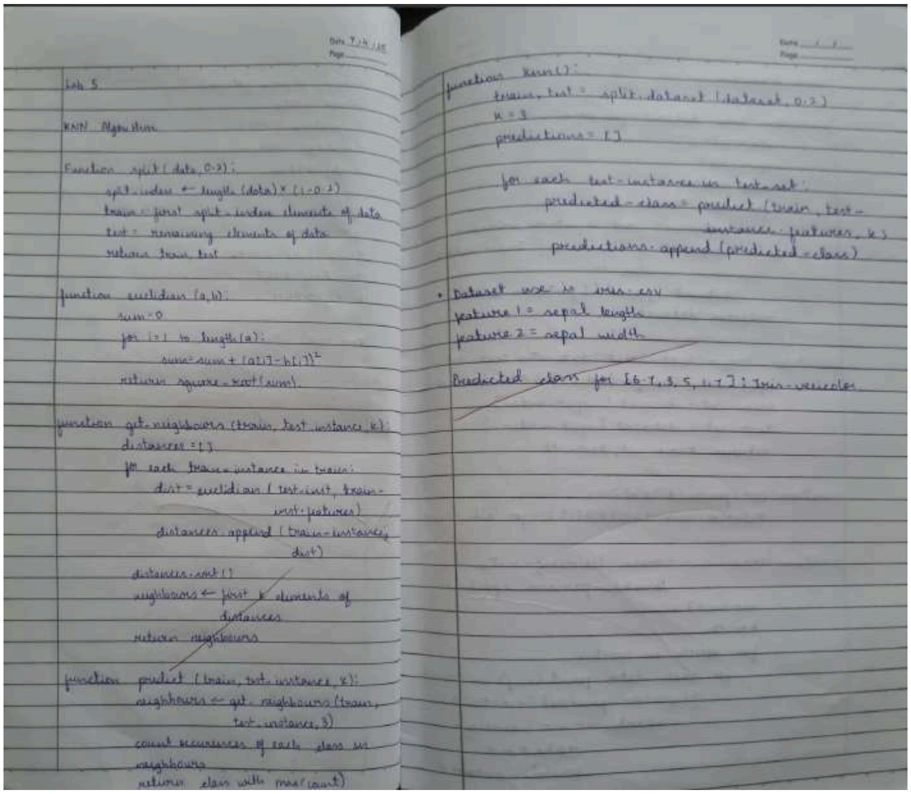
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)



## Program 6

Build KNN Classification model for a given dataset

**Screenshots**

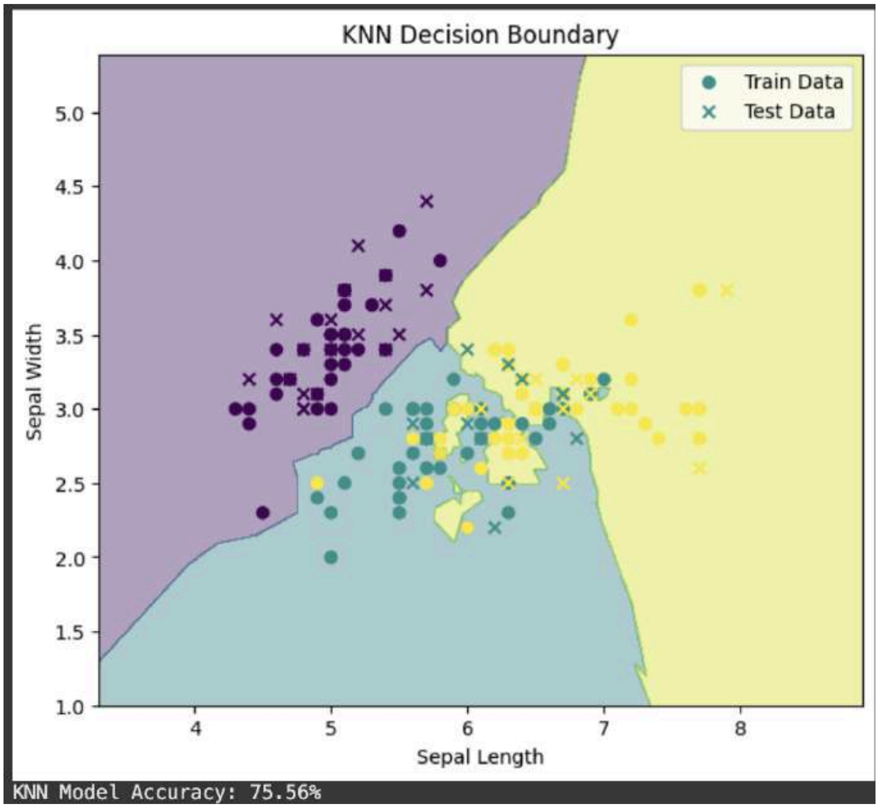


**Code:**

*Using Iris Dataset and visualizing:*

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # Only use the first two features (sepal length and sepal width)
y = iris.target  # Target labels
# Split the dataset into training and testing sets
```

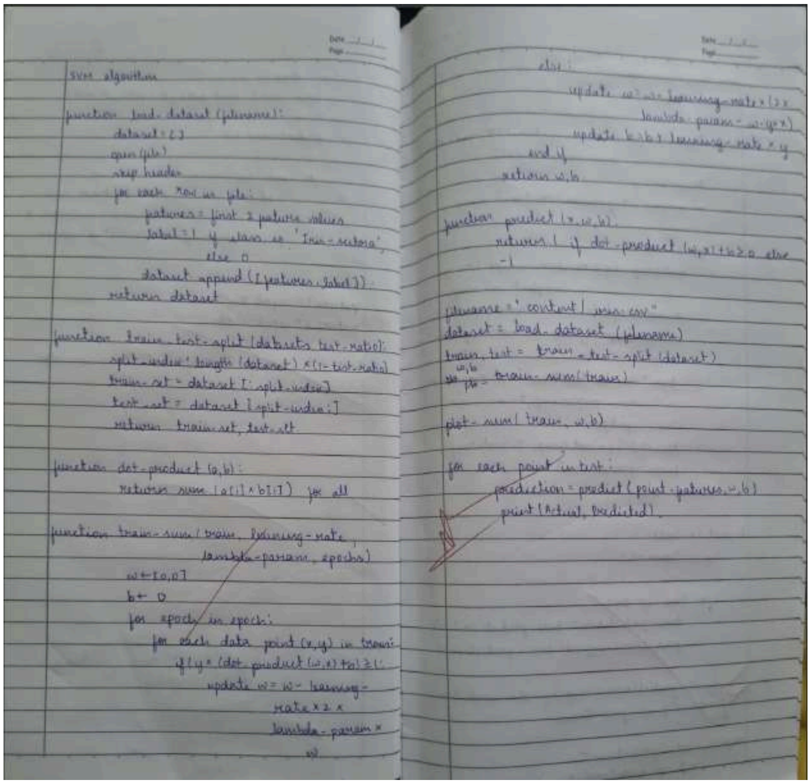
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Create the KNN classifier (k=3 for this example)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
# Create a mesh grid for plotting decision boundaries
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))
# Plotting the decision boundaries for KNN
plt.figure(figsize=(7, 6))
Z_knn = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z_knn = Z_knn.reshape(xx.shape)
plt.contourf(xx, yy, Z_knn, alpha=0.4)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, marker='o', label="Train Data")
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, marker='x', label="Test Data")
plt.title("KNN Decision Boundary")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend()
plt.show()
# Print accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Model Accuracy: {accuracy_knn * 100:.2f}%")
```


KNN Decision Boundary
Train Data
Test Data
Sepal Width
Sepal Length
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8
KNN Model Accuracy: 75.56%

## Program 7

Build Support vector machine model for a given dataset

**Screenshots**

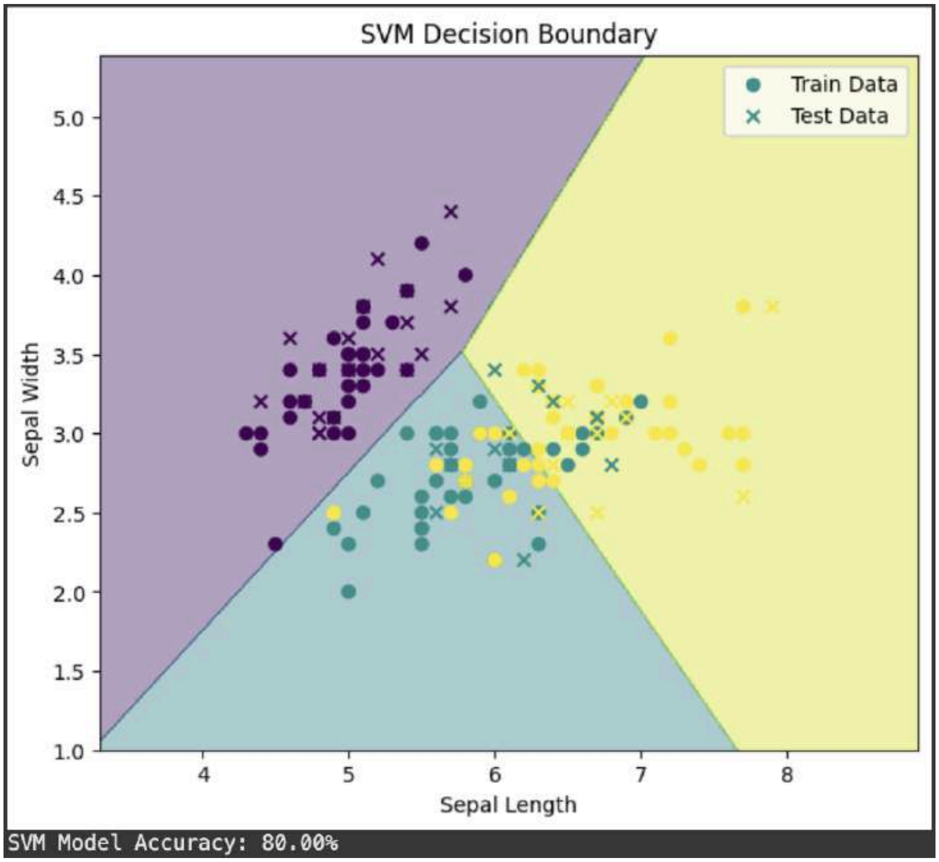


**Code:**

*Using Iris Dataset and visualizing:*

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # Only use the first two features (sepal length and sepal width)
y = iris.target  # Target labels
# Split the dataset 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)
```

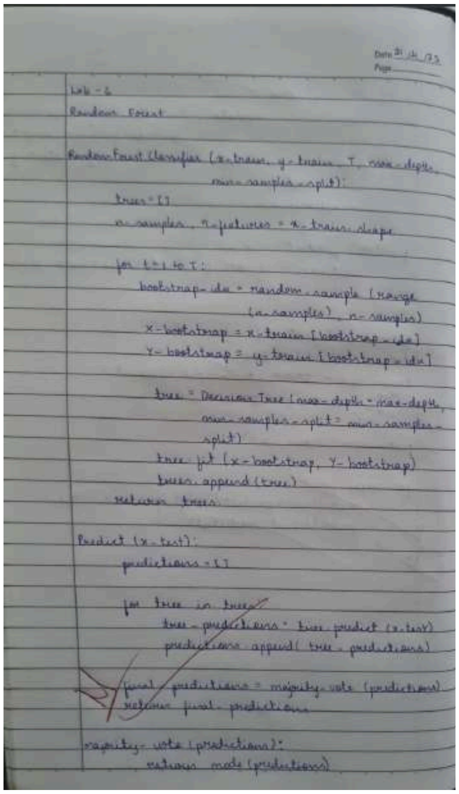
```
# Create the SVM classifier (using a linear kernel for this example)
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
# Create a mesh grid for plotting decision boundaries
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))
# Plotting the decision boundaries for SVM
plt.figure(figsize=(7, 6))
Z_svm = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z_svm = Z_svm.reshape(xx.shape)
plt.contourf(xx, yy, Z_svm, alpha=0.4)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, marker='o', label="Train Data")
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, marker='x', label="Test Data")
plt.title("SVM Decision Boundary")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend()
plt.show()
# Print accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Model Accuracy: {accuracy_svm * 100:.2f}%")
```


SVM Decision Boundary
Train Data
Test Data
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Sepal Width
4
5
6
7
8
Sepal Length
SVM Model Accuracy: 80.00%

## Program 8

Implement Random forest ensemble method on a given dataset.

**Screenshots**



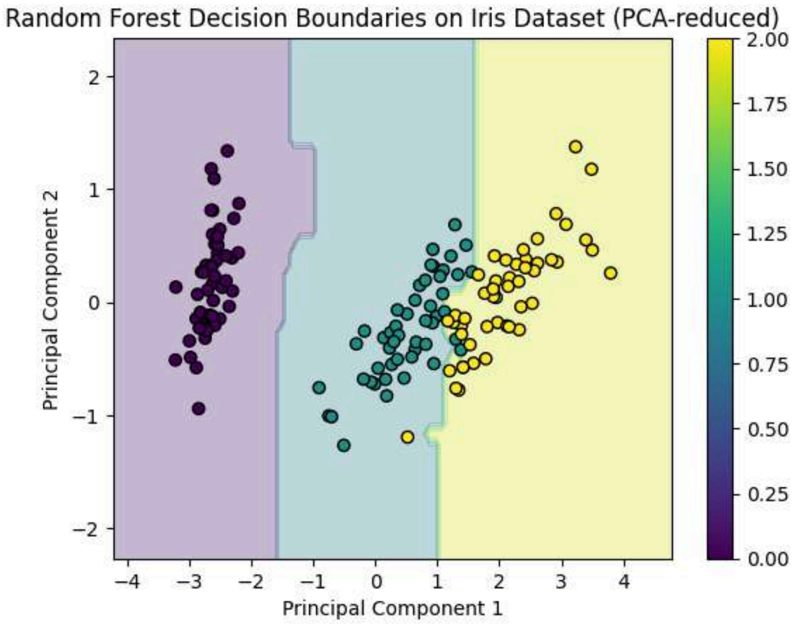
```
Lab - 6
Random Forest

RandomForestClassifier (x_train, y_train, T, max_depth,
                        min_samples_split):
    trees = []
    n_samples, n_features = x_train.shape

    for t=1 to T:
        bootstrap_idx = random.sample (range
                        (n_samples), n_samples)
        x_bootstrap = x_train [bootstrap_idx]
        Y_bootstrap = y_train [bootstrap_idx]

        tree = DecisionTree (max_depth = max_depth,
               min_samples_split = min_samples_
               split)
        tree.fit (x_bootstrap, Y_bootstrap)
        trees.append (tree)
    return trees

Predict (x_test):
    predictions = []

    for tree in trees:
        tree_predictions = tree.predict (x_test)
        predictions.append( tree_predictions)

    final_predictions = majority_vote (predictions)
    return final_predictions

majority_vote (predictions):
    return mode (predictions)
```

**Code:**

*Using Iris Dataset and visualizing:*

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Apply PCA for 2D visualization (reduce to 2 components)
pca = PCA(n_components=2)
```

```
X_pca = pca.fit_transform(X)
# Initialize Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
# Train the model
clf.fit(X_pca, y)
# Create a mesh grid to plot decision boundaries
x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
# Predict on mesh grid
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# Plot decision boundaries
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, edgecolors='k', cmap='viridis')
plt.title('Random Forest Decision Boundaries on Iris Dataset (PCA-reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar()
plt.show()
```


Random Forest Decision Boundaries on Iris Dataset (PCA-reduced)
Principal Component 2
Principal Component 1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

## Program 9

Implement Boosting ensemble method on a given dataset.

**Screenshots**

Date 5/5/25
Page

Lab-7

Boosting Ensemble [Adaboost]

1. Initialize weights for each training sample.
   - Set each sample's weight to 1/N (where N is the no. of samples)

2. For each iteration (t = 1 to T):
   a. Train a weak classifier (ex: a decision stump) using the weighted samples.

   b. Calculate the classifier's error:
   - Error = (sum of weights of misclassified samples) / (sum of all sample weights)

   c. Calculate the classifier's importance (alpha):
   - Alpha = 0.5 • log((1 - Error) / Error)
   - If Error is high, alpha will be low, meaning less importance for this classifier.

   d. Update the sample weights:
   - increase the weight of misclassified sample
   - decrease the weight of correctly classified samples
   - Normalize the weight to make sure they can sum to 1

3. After all iterations, combine the weak classifiers:
   - each classifier contributes based on its importance (alpha)

   - Final predictions = sign (sum of all classifiers)

Date __/__/__
Page

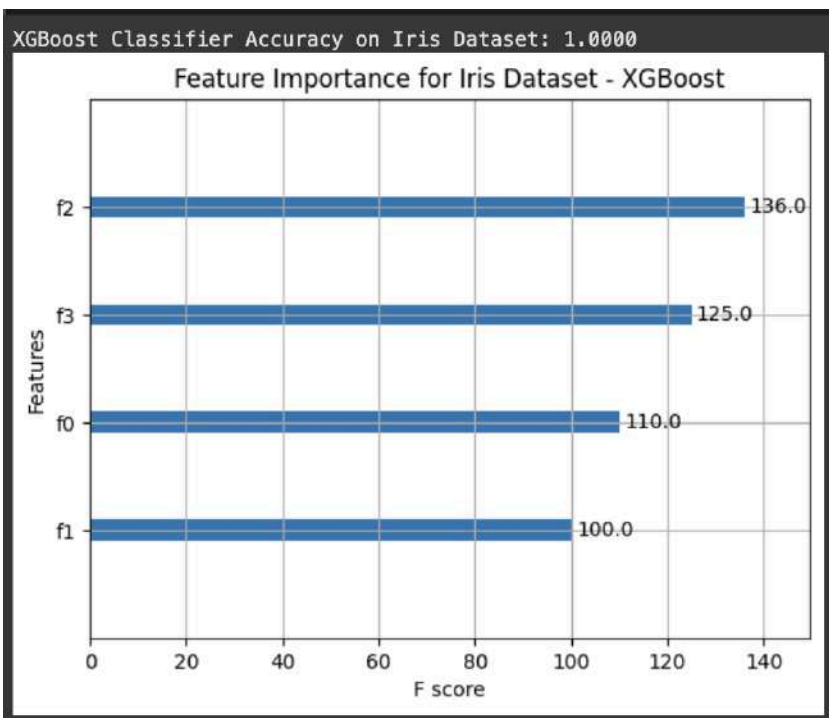
4. Use the combined classifier to make predictions on new data.

**Code:**

*Using Iris Dataset and visualizing:*

```
# XGBoost on Iris Dataset with Feature Importance Visualization
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
```

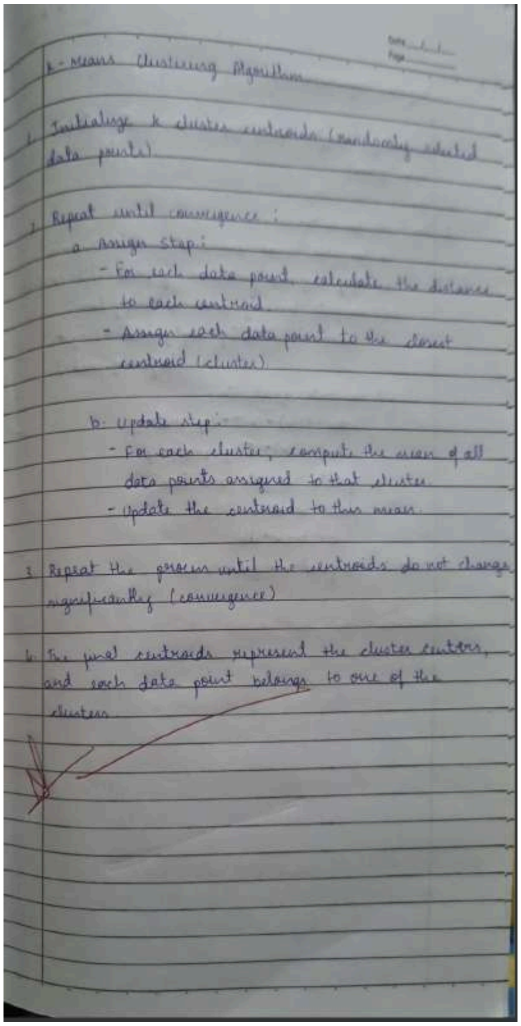
```
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Initialize XGBoost Classifier
xgb_clf = xgb.XGBClassifier(n_estimators=100, random_state=42)
# Train the model
xgb_clf.fit(X_train, y_train)
# Make predictions
y_pred = xgb_clf.predict(X_test)
# Calculate accuracy
accuracy = (y_pred == y_test).mean()
print(f'XGBoost Classifier Accuracy on Iris Dataset: {accuracy:.4f}')
# Feature importance visualization
xgb.plot_importance(xgb_clf, importance_type='weight', max_num_features=10)
plt.title('Feature Importance for Iris Dataset - XGBoost')
plt.show()
```



## Program 10

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

**Screenshots**



k-Means Clustering Algorithm

1. Initialize k cluster centroids (randomly selected data points)

2. Repeat until convergence:
   a. Assign step:
      - For each data point, calculate the distance to each centroid.
      - Assign each data point to the closest centroid (cluster).

   b. Update step:
      - For each cluster, compute the mean of all data points assigned to that cluster.
      - Update the centroid to this mean.

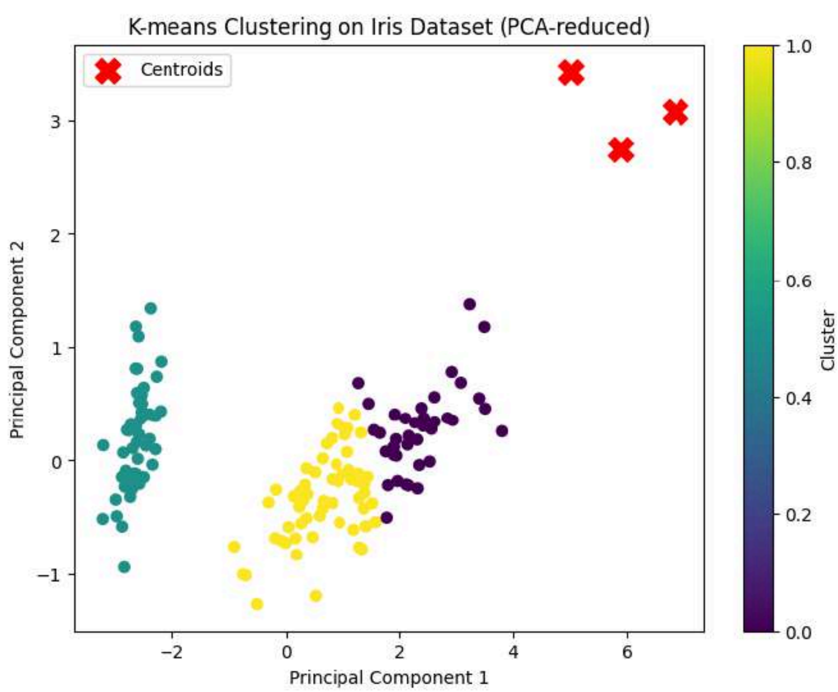
3. Repeat the process until the centroids do not change significantly (convergence)

4. The final centroids represent the cluster centers, and each data point belongs to one of the clusters.

**Code:**

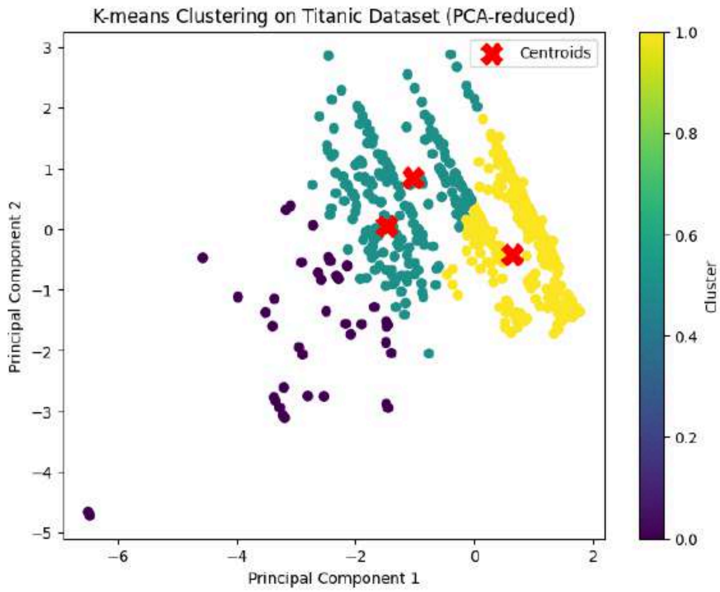
```
# K-means on Iris Dataset with Visualization

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

# Load Iris dataset

iris = load_iris()

X = iris.data

y = iris.target
```

```
# Apply PCA for 2D visualization (reduce to 2 components)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
# Perform K-means clustering (3 clusters for 3 species in Iris)
kmeans = KMeans(n_clusters=3, random_state=42)
y_kmeans = kmeans.fit_predict(X)
# Visualize the clustering result in 2D (PCA-reduced space)
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X',
label='Centroids')
plt.title('K-means Clustering on Iris Dataset (PCA-reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.legend()
plt.show()
```




```
# K-means on Kaggle Titanic Dataset with Visualization
import pandas as pd
```

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
# Load Titanic dataset (You can replace the URL with your own Kaggle dataset link)
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)
# Preprocessing: Selecting relevant features and dropping missing values
data = data.dropna(subset=['Pclass', 'Age', 'Fare'])
X = data[['Pclass', 'Age', 'Fare']]
# Normalize the data (optional, but helps with clustering)
X = (X - X.mean()) / X.std()
# Apply K-means clustering (let's assume we use 3 clusters for this example)
kmeans = KMeans(n_clusters=3, random_state=42)
y_kmeans = kmeans.fit_predict(X)
# Apply PCA for 2D visualization (reduce to 2 components)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
# Plot the clusters in 2D
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X',
label='Centroids')
plt.title('K-means Clustering on Titanic Dataset (PCA-reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.legend()
plt.show()
```


K-means Clustering on Titanic Dataset (PCA-reduced)
Centroids
Principal Component 1
Principal Component 2
Cluster

## **Program 11**

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

**Screenshots**

Date __/__/__
Page ____

PCA (Principal Component Analysis)

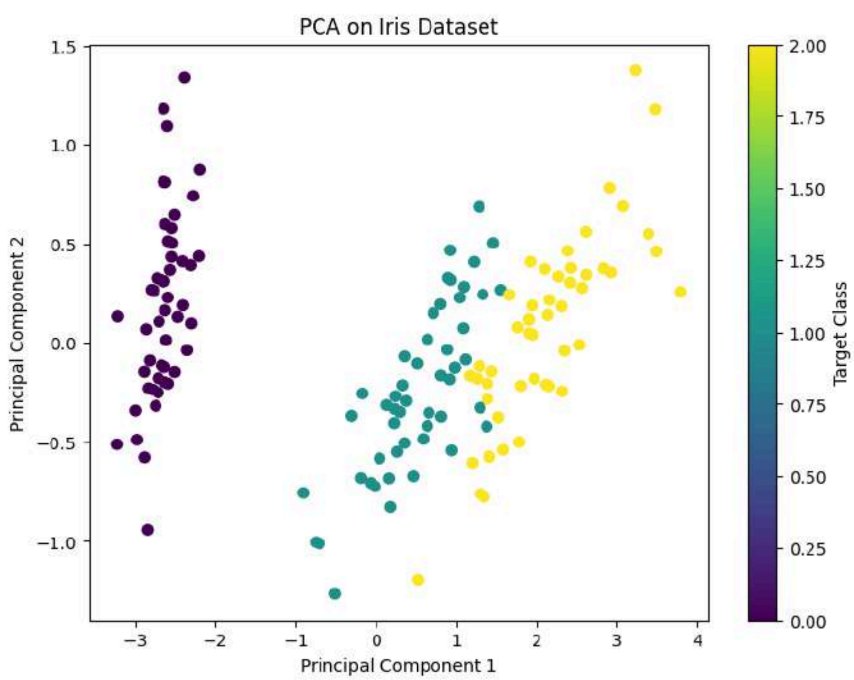
1. Standardize the dataset (mean = 0, std = 1)
   For each feature in X:
   subtract mean & divide by standard deviation
2. Compute the covariance matrix of the standardized data
   cov = (X.T * X) / (n_samples - 1)
3. Compute eigenvalues and eigenvectors of the covariance matrix
   eig_vals, eig_vecs = eig(cov)
4. Sort eigenvectors by decreasing eigenvalues (variance)
   sort eig_vecs according to sorted eig_vals in decreasing order
5. Select the top k eigenvectors
   W = eig_vecs[:, :k]
6. Project the data onto the new subspace
   X_reduced = X * W

**Code:**

*Using Iris Dataset and visualizing:*

```
# PCA on Iris Dataset with Visualization
import matplotlib.pyplot as plt
```

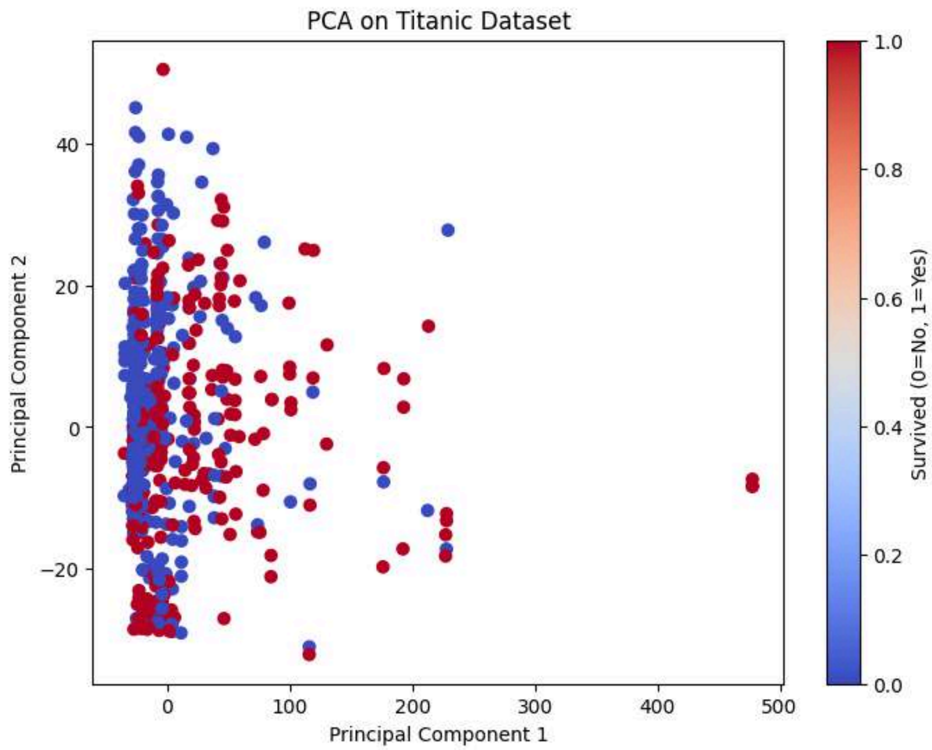
```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Apply PCA to reduce data to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
# Plot the PCA-reduced data
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.title('PCA on Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Target Class')
plt.show()
```




*Using Kaggle Titanic Dataset and visualizing:*

```
# PCA on Kaggle Titanic Dataset with Visualization
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
# Load Titanic dataset (Replace with your dataset URL or local file path)
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)
# Preprocess the dataset: Handle missing values and encode categorical features
data = data.dropna(subset=['Pclass', 'Age', 'Fare'])
data['Sex'] = LabelEncoder().fit_transform(data['Sex'])
# Select relevant features
X = data[['Pclass', 'Age', 'Fare']]
y = data['Survived']
# Apply PCA to reduce data to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
# Plot the PCA-reduced data
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm')
plt.title('PCA on Titanic Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Survived (0=No, 1=Yes)')
plt.show()
```


PCA on Titanic Dataset
Principal Component 1
Principal Component 2
Survived (0=No, 1=Yes)
0
100
200
300
400
500
−20
0
20
40
0.0
0.2
0.4
0.6
0.8
1.0