

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT
on**

Machine Learning (23CS6PCMAL)

Submitted by

VISHWAS H KUMAR (1BM22CS338)

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 **Vishwas H Kumar (1BM22CS338)**, 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.

Prof. Sarala D V Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE
---	--

Index

Sl. No.	Date	Experiment Title	Page No.
1	21-2-2025	Write a python program to import and export data using Pandas library functions	1
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	14
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	28
4	17-3-2025	Build Logistic Regression Model for a given dataset	32
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	35
6	7-4-2025	Build KNN Classification model for a given dataset	43
7	21-4-2025	Build Support vector machine model for a given dataset	47
8	5-5-2025	Implement Random forest ensemble method on a given dataset	51
9	5-5-2025	Implement Boosting ensemble method on a given dataset	54
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file	57
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	61

GitHub Link: <https://github.com/vishwaas-hkumarCS338/ML-LAB>

Program 1

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

Screenshots

4/3/25 LAB1 - LOADING DATASETS

```

tickets = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yt.download(tickers, start = "2024-01-01", end = "2024-12-30",
                   group_by = 'ticker')
import yfinance as yt
import pandas as pd
import matplotlib.pyplot as plt
nse_data = data['HDFCBANK.NS']
nse_data['Daily Returns'] = nse_data['close'].pct_change()
print("In Daily returns for HDFC Industries :")
print(nse_data['Daily Returns'].head())
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
nse_data['close'].plot(title = "HDFC Industries - Closing Price")
plt.subplot(2,1,2)
nse_data['Daily Returns'].plot(title = "HDFC Industries - Daily Returns")
plt.tight_layout()
plt.show()

```

→ Daily Returns for HDFC Industries

Date	Daily Returns
2024-01-01	NAN
2024-02-01	0.000589
2024-03-01	-0.015420
2024-01-04	0.010730
2024-01-05	-0.005116

import pandas as pd

- dt = pd.DataFrame({'USN': [1, 2, 3, 4, 5], 'Name': ['A', 'B', 'C', 'D', 'E'], 'Marks': [10, 20, 30, 40, 50]})
dt = pd.DataFrame(dt)
print(dt)
- from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
dt = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
dt['target'] = diabetes.target
dt.head()
- file-path = "sample-sales-data.csv"
dt = pd.read_csv(file-path)
print(sample_data)
- dt.to_csv('output.csv', index=False)
print("Data saved to output.csv")

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_Patients 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 rows of the DataFrame

print(df.head())

# Writing the DataFrame to a CSV file

df.to_csv('output.csv', index=False)

print("Data saved to output.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 to output.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  
21.0    25022.0
```

```
▶ # 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())

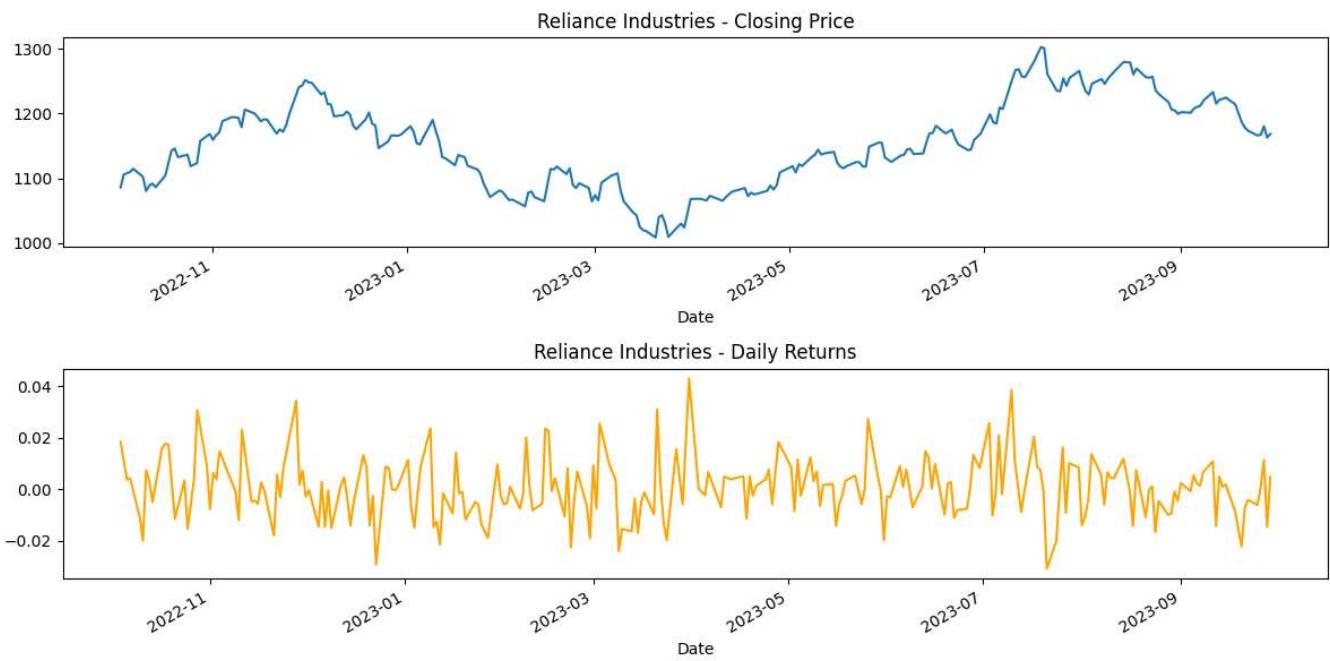
# Calculate daily returns
# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
reliance_data = data['RELIANCE.NS'].copy()
# Now, apply the calculation
```

```
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					
Date	Open	High	Low	Close	Volume	\
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()
```

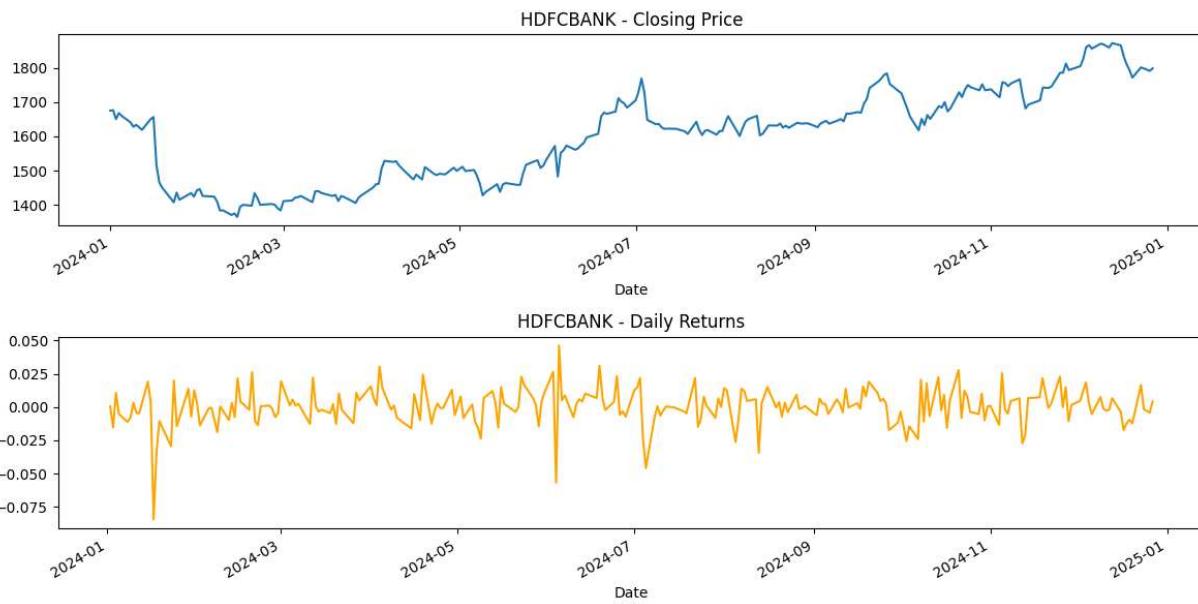
Summary statistics for KOTAKBANK:						
Price	Open	High	Low	Close	Volume	
count	244.000000	244.000000	244.000000	244.000000	2.440000e+02	
mean	1771.245907	1787.548029	1754.395105	1770.792347	5.736598e+06	
std	62.189675	61.978802	62.765980	62.594747	5.388927e+06	
min	1581.266899	1586.161558	1542.159736	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()
```

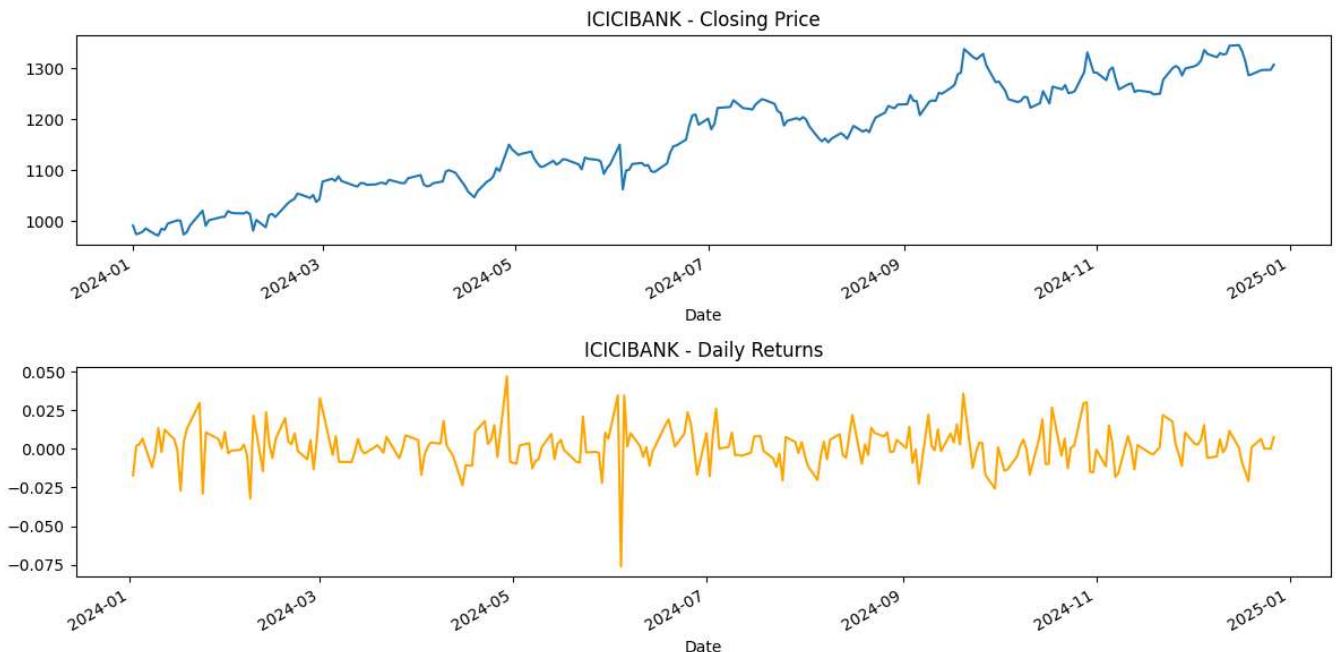


```
# Plot the closing price and daily returns
plt.figure(figsize=(12, 6))

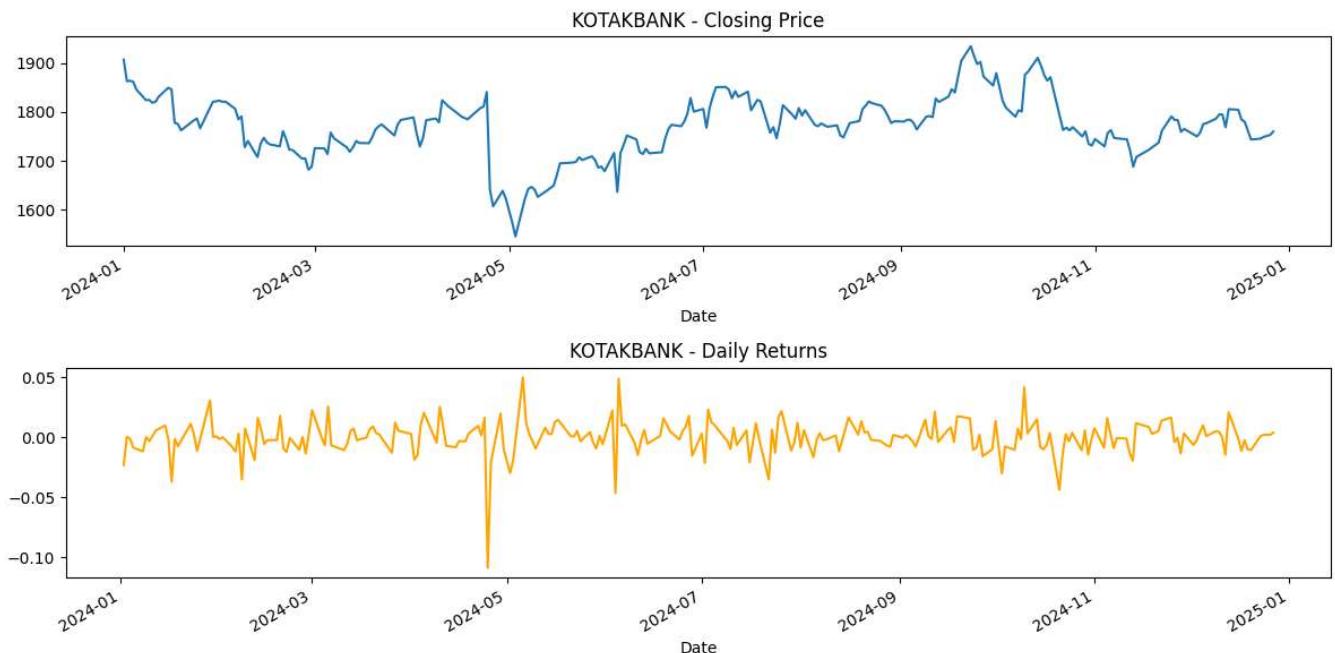
plt.subplot(2, 1, 1)
ICICI['Close'].plot(title="ICICIBANK - Closing Price")

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

The image shows handwritten notes on a lined notebook page. At the top left, it says "4/3/25 LAB-2". In the top right corner, there is a logo for "URBAN EDGE". The main heading is "DATA PREPROCESSING". Below this, the notes are organized into numbered steps:

- (i) housing.csv
 - (i) Load .csv file into DataFrame
 - dt = pd.read_csv("housing.csv")
- (ii) Display information about all columns
- print(dt.info())
- (iii) Display statistical information of all numerical columns
- print(dt.describe())
- (iv) Display count of all unique labels for 'Ocean Proximity' column
- print(dt['ocean-proximity'].value_counts())
- (v) Display which attributes (columns) have missing values count greater than zero.
- print(dt.isnull().sum() [dt.isnull().sum() > 0])

Below these steps, the word "Output" is written, followed by "Information of all columns" under a red line. A table is shown:

#	column	Non-null count	Type
0	longitude	20640 non-null	float64

Below the table, it says "Statistical information about all numerical columns:" and lists the following values:

count	longitude	latitude	housing.median_age	total_rooms
20640.000	20640.000	20640.000	20640.000	20640.000

URBAN
EDGE

	total_bedrooms	population	households	median_income	med_house_val					
0	20433.000	20640.000	20640.000	20640.000	20640.000					
Count of unique labels for 'ocean_Proximity' column										
OCEAN-PROXIMITY										
NEAR OCEAN	9136									
INLAND	6551									
Columns with missing values count greater than 0 :										
total_bedrooms	207									
→ Analysis 1) Diabetes & Adult Income Datasets										
(q1)	Which columns in the dataset had missing values? How did you handle them?									
* Diabetes dataset -	missing values : numerical columns : glucose levels, blood pressure categorical columns : gender, ethnic group									
Handling missing values :										
• numerical columns : mean imputation - missing values filled with mean of column										
• categorical columns : most frequent imputation - missing values filled with most frequent value of that column.										
* Adult dataset -	missing values : numerical : hours worked, age categorical : marital status, education									
Handling missing values - same as above.										
(q2)	Which categorical columns did you identify in the dataset? How did you encode them?									
* Diabetes dataset	categorical columns like gender, race/ethnicity & pregnancy status Encoding - Label encoding - we convert each category into a column into a unique integer (0: male - 1: female - 1)									
* Adult income dataset	categorical columns like workclass, education, maritalstatus, occupation. Encoding - Label encoding - unique integer assigned to each category 0: private in occupation → 0 self-employed-not-inc → 1									

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 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
   0      -122.23    37.88        41.0     880.0     129.0
   1      -122.22    37.86        21.0     7999.0    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    -1201.09   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     2524.0     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())
```

```
count    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" 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	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	0.0
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

#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
  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	\
	Gender_Encoded	CLASS_N	CLASS_N	CLASS_P	CLASS_Y	CLASS_Y							
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	

→ 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	\	occupation	relationship	race	gender	capital-gain	capital-loss	\
0	25	Private	226802	11th		7	Never-married					0	0	
1	38	Private	89814	HS-grad		9	Married-civ-spouse					0	0	
2	28	Local-gov	336951	Assoc-acdm		12	Married-civ-spouse					0	0	
3	44	Private	160323	Some-college		10	Married-civ-spouse					0	0	
4	18	?	103497	Some-college		10	Never-married					0	0	
0	Machine-op-inspct	Own-child	Black	Male	0							0	0	
1	Farming-fishing	Husband	White	Male	0							0	0	
2	Protective-serv	Husband	White	Male	0							0	0	
3	Machine-op-inspct	Husband	Black	Male	7688							0	0	
4	?	Own-child	White	Female	0							0	0	
0		hours-per-week	native-country	income										
1		40	United-States	<=50K										
2		50	United-States	<=50K										
3		40	United-States	>50K										
4		40	United-States	>50K										
		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
   ... 25      Private  226802      age  workclass  fnlwgt  education  educational-num \
0      25      Private  226802      11th      7      9
1      38      Private  89814      HS-grad     9
2      28  Local-gov  336951  Assoc-acdm    12
3      44      Private  160323  Some-college  10
4      18      ?  183497  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-inspt  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-inspt  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-inspt  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

The left page of the notebook contains handwritten notes on linear regression. It includes a table of data points, the normal equation for finding coefficients, and the derivation of the coefficient matrix $(X^T X)^{-1} X^T Y$. The right page shows a Python script for performing linear regression on the same dataset using the scikit-learn library.

Handwritten Notes (Left Page):

- 1/3/25 LAB-4
- Linear Regression & Multiple Linear Regression
- Find linear regression of data of week & product sales
- xi (week) yi (sales in thousands)
- 1 2
- 2 4
- 3 5
- 4 9
- $\beta = [(X^T X)^{-1} X^T] Y$
- $$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$
- $X^T = [1 2 3 4]$
- $Y^T = [2 4 5 9]$
- $X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$
- $(X^T X)^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$
- $(X^T X)^{-1} X^T = \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.5 & -0.1 & 0.1 & 0.3 \end{bmatrix}$
- $\downarrow \times Y$
- $\begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$
- $y = \beta_0 + \beta_1 x$
- $y = -0.5 + 2.2 \cdot x$
- for $x = 5$
- $y = -0.5 + 2.2 \cdot (5) \approx 10.5$

Python Code (Right Page):

```

→ Linear Regression
① canada_per_capita_income.csv

import pandas as pd
import numpy as np
import sklearn as linear_model
from scipy.stats import mstats import winsorize

canada_income_dt = pd.read_csv("canada_per_capita_income.csv")
print(canada_income_dt.head())

X_canada = canada_income_dt[['year']]
Y_canada = canada_income_dt['per capita income (us$)']

reg_canada = linear_model.LinearRegression()
reg_canada.fit(X_canada, Y_canada)

income_2020 = reg_canada.predict(pd.DataFrame([[2020]], columns=['year']))

print("Coefficient:", reg_canada.coef_)
print("Intercept:", reg_canada.intercept_)

print("Predicted per capita income for 2020:", income_2020[0])

year per capita income (us$)
0 1970 3399.299037
1 1971 3768.297935
2 1972 4251.175484
3 1973 4804.463248
4 1974 5576.54583

```

coefficient : 828.46501522
intercept : -1632210.757854575
predicted per capita income for 2020 : \$91288.69

output for (2) salary.csv		O/P
Years	Experience	Salary
0	1.1	39843
1	1.8	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
Coefficient :	9318.10928245	Predicted salary for (2yrs, 9test, 6interview) : \$53205.77
Intercept :	28079.143922303017	Predicted salary for (12yrs, 10test, 10interview) : \$92002.18
(ii) 1000-companies.csv		
		Predicted profit : [51017.34614637]

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

<p>18/3/25 LAB 3 - Logistic Regression</p> <p>(1) Problem 1 - Binary Classification (Sigmoid Fn)</p> $a_0 = -5, a_1 = 0.8$ $\text{LR Eqn} \Rightarrow z = a_0 + a_1 x$ $= -5 + 0.8x$ <p>$p(x)$ for $x = 7$</p> $y = p(x) = \frac{1}{1 + e^{-(a_0 + a_1 x)}} = \frac{1}{1 + e^{-(5 + 0.8(7))}} = 0.64.$ <p>$0.64 > 0.5 \rightarrow 1 \rightarrow \text{Pass}$</p> <p>∴ Student who studies for 7 hours will Pass. The predicted class is <u>Pass</u>.</p> <p>(2) Problem 2 - MultiClass Regression (Softmax Fn)</p> <p>$z = [2, 1, 0]$. Find probability values of the 3 classes</p> <p>$\text{Softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^k e^{z_j}}$</p> $\text{Softmax}(z_1) = \frac{e^2}{e^2 + e^1 + e^0} = 0.665 \quad (66.5\%)$ $\text{Softmax}(z_2) = \frac{e^1}{e^2 + e^1 + e^0} = 0.244 \quad (24.4\%)$ $\text{Softmax}(z_3) = \frac{e^0}{e^2 + e^1 + e^0} = 0.091 \quad (9.1\%)$	<p>(1) Binary Classification.py</p> <pre>import pandas as pd from matplotlib import pyplot as plt from sklearn.model_selection import train_test_split from sklearn.linear_model import LogisticRegression import math</pre> $dt = pd.read_csv("insurance-data.csv")$ $print("Null Values:", dt.isnull().sum())$ $plt.scatter(dt.age, dt.bought_insurance, marker='+', color='red')$ $plt.xlabel("Age")$ $plt.ylabel("Bought Insurance")$ $plt.title("Age vs Insurance Purchases")$ <p>$x_train, x_test, y_train, y_test = train_test_split(dt[["age"]], dt.bought_insurance, train_size=0.7, random_state=10)$</p> $model = LogisticRegression()$ $model.fit(x_train, y_train)$ $y_predicted = model.predict(x_test)$ $print("Model Accuracy: ", model.score(x_test, y_test))$ $print("Prediction Probabilities: ", model.predict_proba(x_test))$ $age_input = 60$ $y_predicted = model.predict(pd.DataFrame([age_input]), columns=['age'])$ $print("Prediction for age={} input={} : {}-predicted({})".format(age_input, y_predicted[0]))$ $print("Coefficient (m): ", model.coef_)$ $print("Intercept (b): ", model.intercept_)$ <p>def sigmoid(x):</p> $return 1 / (1 + math.exp(-x))$ <p>def prediction_function(age):</p> $z = model.coef_[0][0] * age + model.intercept_[0]$ $y = sigmoid(z)$ $return y$
---	---

<pre> age = 35 print("Sigmoid prediction for age : ", prediction_function(age)) if p < 0.5: print("Person will not buy insurance") else: print("Person will buy insurance") (2) Multiclass.py import pandas as pd ... # same as before dt = pd.read_csv('iris.csv') print("Null values in dataset : ", dt.isnull().sum()) dt.head() X = df.drop('species', axis='columns') y = df['species'] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) model = LogisticRegression(solver='lbfgs', max_iter=200) model.fit(X_train, y_train) y_pred = model.predict(X_test) accuracy = accuracy_score(y_test, y_pred) print("Accuracy : ", accuracy) cont_matrix = confusion_matrix(y_test, y_pred) cm_display = ConfusionMatrixDisplay(confusion_matrix=cont_matrix, display_labels=["setosa", "versicolor", "virginica"]) cm_display.plot() plt.show() dp Accuracy of the Multinomial Logistic Regression model on test set : 1.00 </pre>	<p style="text-align: right;">URBAN EDGE</p> <p>(3) HR.comma_sep.csv</p> <p>(i) salary & work-satisfaction are the two attributes that have a strong correlation with employee retention. As per the dataset, those employees with low salary & low work-satisfaction are the ones that have less retention.</p> <p>(ii) The accuracy of the logical regression model is 78.17%.</p> <p>(4) zoo-data.csv</p> <p>(i) The most frequently misclassified label is mammal according to the confusion matrix</p> <p>(ii) Accuracy of the model : 95%</p> <p>(iii) The confusion matrix tells us that the model is mostly accurate.</p>
--	---

Code:

```

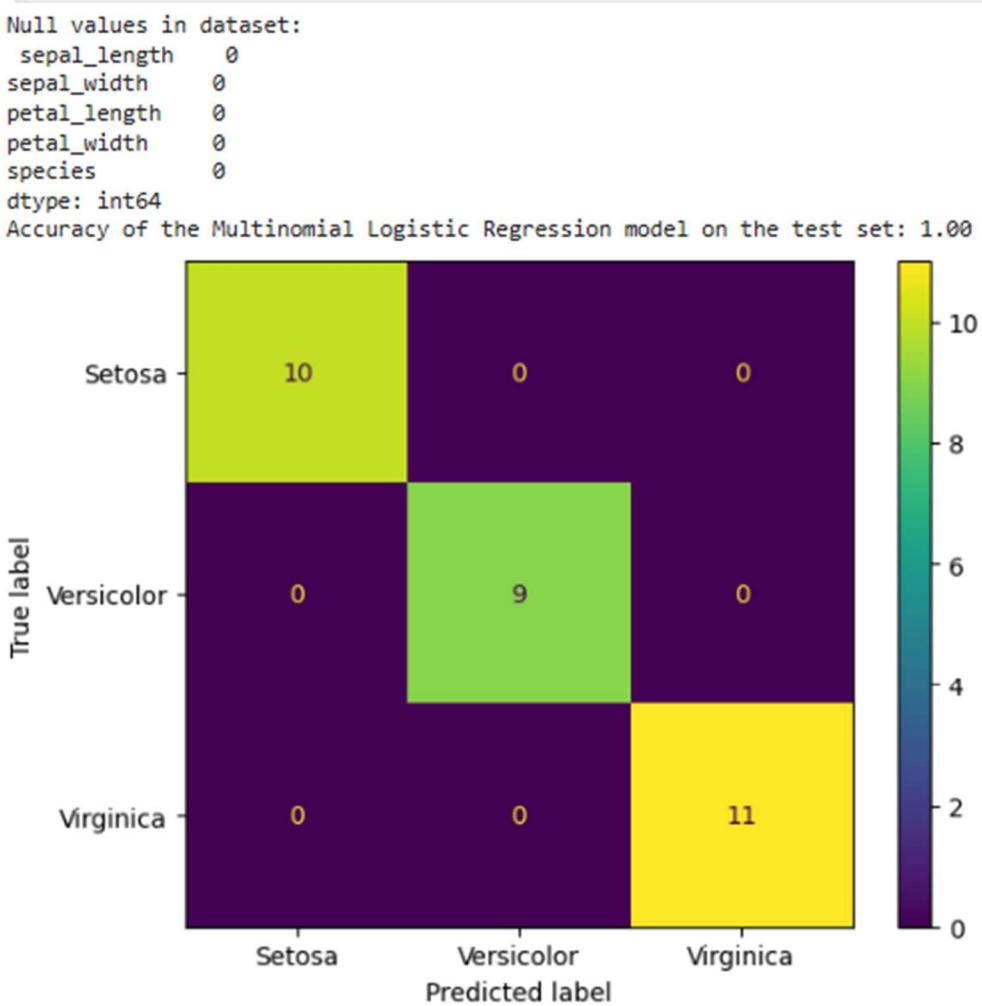
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
df = pd.read_csv("iris.csv")
print("Null values in dataset:\n", df.isnull().sum())
df.head()
X = df.drop('species', axis='columns') # Features (sepal length, sepal width, petal length, petal width)
y = df.species # Target labels (Setosa, Versicolor, Virginica)

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(solver='lbfgs', max_iter=200)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}')
conf_matrix = confusion_matrix(y_test, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=["Setosa",
"Versicolor", "Virginica"])
cm_display.plot()
plt.show()

```



Program 5

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

Screenshots

LAB 5 - DECISION TREES

Instance a₁ a₂ a₃ Classification

1	T	H	H	N
2	T	H	H	N
3	F	H	H	Y
4	F	C	N	Y
5	F	C	N	Y
6	T	C	H	N
7	T	K	H	N
8	T	H	N	Y
9	F	C	N	Y
10	F	C	H	Y

Attribute : a₁
values (a₁) = True, False
 $S = [6+, 4-] \text{ Entropy}(S) = -\frac{6}{10} \log_2 \frac{6}{10} - 4/10 \log_2 \frac{4}{10} = 0.9709$
 $S_{True} = [1+, 4-] \text{ Entropy}(S_{True}) = 1/5 \log_2 1/5 - 4/5 \log_2 4/5 = 0.7219$
 $S_{False} = [5+, 0-] \text{ Entropy}(S_{False}) = 0$
 $\text{gain}(S, a_1) = \text{Entropy}(S) - \sum S_i v_i / |S| \cdot \text{Entropy}(S_i)$
 $= 0.9709 - 5/10 \text{Entropy}(S_{True}) - 5/10 \text{Entropy}(S_{False})$
 $= 0.6049$

Attribute : a₂
values (a₂) = Hot, Cold
 $S = [6+, 4-] \Rightarrow \text{Entropy}(S) = 6/10 \log_2 6/10 - 4/10 \log_2 4/10 = 0.9709$
 $S_{Hot} = [2+, 3-] \Rightarrow \text{Entropy}(S_{Hot}) = 2/5 \log_2 2/5 - 3/5 \log_2 3/5 = 0.9709$
 $S_{Cold} = [4+, 1-] \Rightarrow \text{Entropy}(S_{Cold}) = 4/5 \log_2 1/5 - 1/5 \log_2 1/5 = 0.7219$
 $\text{gain}(S, a_2) = \text{Entropy}(S) - 5/10 \text{Entropy}(S_{Hot}) - 5/10 \text{Entropy}(S_{Cold})$
 $= 0.4199$

Attribute : a₃
values (a₃) = High, Normal
 $S = [6+, 4-] \Rightarrow \text{Entropy}(S) = 6/10 \log_2 6/10 - 4/10 \log_2 4/10 = 0.9709$
 $S_{High} = [1+, 4-] \Rightarrow \text{Entropy}(S_{High}) = 1/5 \log_2 1/5 - 4/5 \log_2 4/5 = 0.7219$
 $S_{Normal} = [5+, 0-] \Rightarrow \text{Entropy}(S_{Normal}) = 0$
 $\text{gain}(S, a_3) = \text{Entropy}(S) - \sum S_i v_i / |S| \cdot \text{Entropy}(S_i)$
 $= 0.9709 - 5/10 \text{Entropy}(S_{High}) - 5/10 \text{Entropy}(S_{Normal})$
 $= 0.7219$

Decision Tree Diagrams:

- Root Node (a₁):
 - True (T) leads to Node 1: 1, 2, 6, 7, 8
 - False (F) leads to Node 2: 3, 4, 5, 9, 10
- Node 1 (a₂):
 - True (T) leads to Node 3: 1, 2, 6, 7, 8
 - False (F) leads to Node 4: 3, 4, 5, 9, 10
- Node 2 (a₃):
 - True (T) leads to Node 5: High
 - False (F) leads to Node 6: Normal

Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

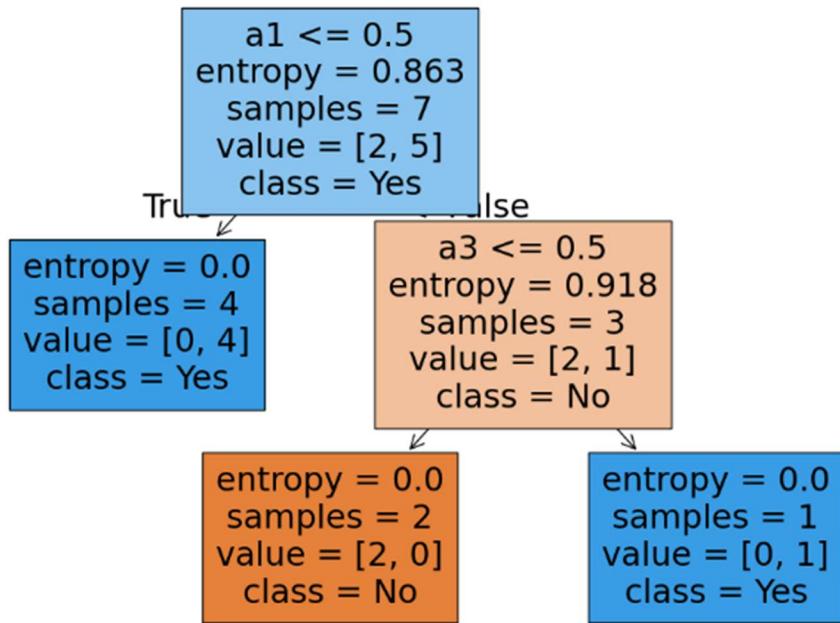
```

from sklearn.metrics import accuracy_score, classification_report
data = {
    'a1': [True, True, False, False, True, True, True, False, False],
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']
}
data
df = pd.DataFrame(data)
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

X = df.drop('Classification', axis=1)
y = df['Classification']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])
plt.show()

```

	precision	recall	f1-score	support
No	1.00	1.00	1.00	2
Yes	1.00	1.00	1.00	1
accuracy			1.00	3
macro avg	1.00	1.00	1.00	3
weighted avg	1.00	1.00	1.00	3



→ 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
```

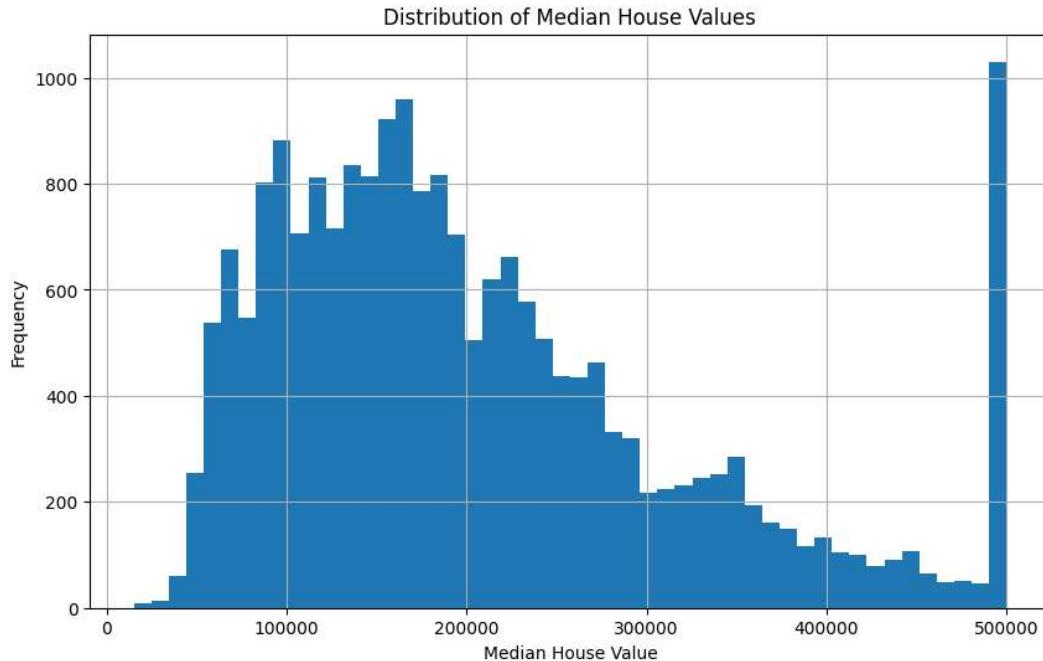
```
▶ housing["ocean_proximity"].value_counts()  
  
▶ count  
  
ocean_proximity  
  <1H OCEAN      9136  
  INLAND          6551  
  NEAR OCEAN      2658  
  NEAR BAY         2290  
  ISLAND           5  
  
dtype: int64
```

```
▶ housing.describe()  
  
▶ count    longitude     latitude  housing_median_age  total_rooms  total_bedrooms  population  households  median_income  median_house_value  
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()

```



```

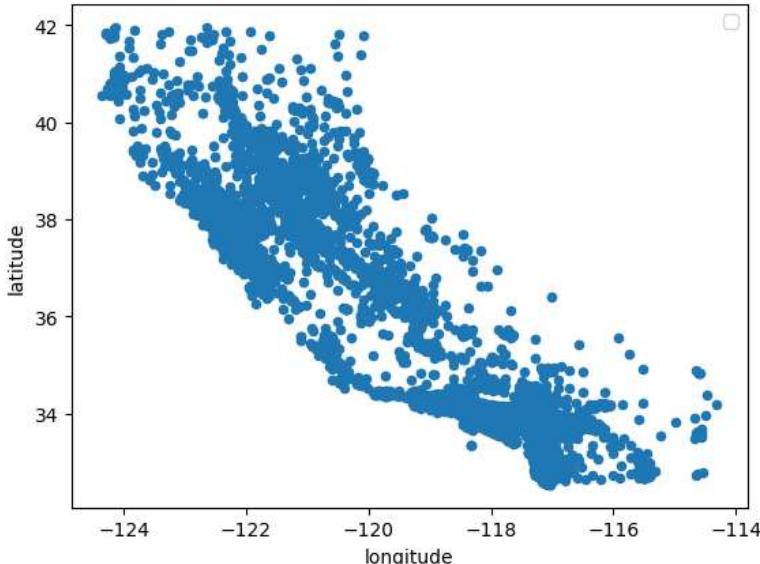
import pandas as pd
from sklearn.model_selection import train_test_split
housing = pd.read_csv("housing.csv")
# Random sampling
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
# 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()

```

```

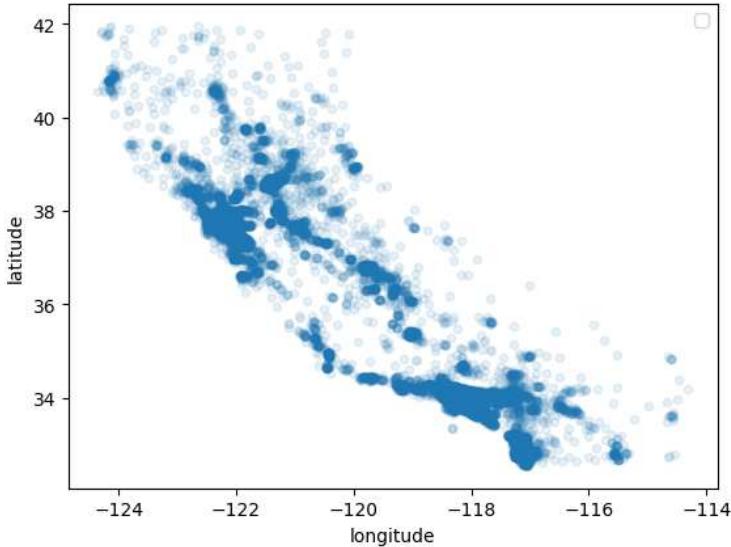
train_set["median_house_value"] = y_train

# Plot the training set

train_set.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)

plt.legend()

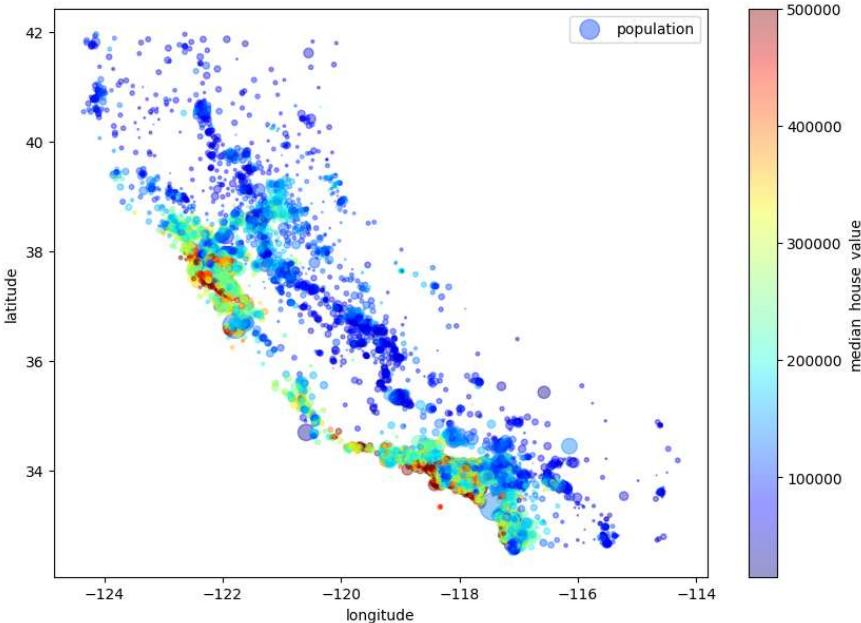
```



```

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)

```



```

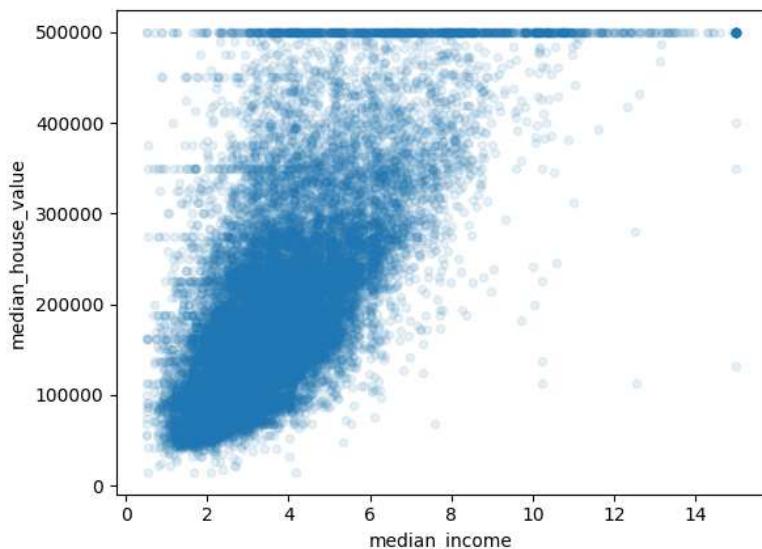
# Select only numerical columns (excluding categorical columns like 'ocean_proximity')
numerical_columns = housing.select_dtypes(include=['float64', 'int64'])

```

```
# 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

1/4/25 LAB-6 : KNN

Consider the following dataset, for k=3 and test data (x, 35, 100) as (Person, Age, Salary) solve using KNN classification & predict the target

Person	Age	Salary	Target	dist/dlist	Rank
A	18	50	N	52.81	5
B	23	55	N	46.57	9
C	24	70	N	31.95	2
D	41	60	Y	40.44	3
E	43	70	Y	31.09	1
F	38	40	Y	60.07	6
X	35	100	?		

k=3

Rank1 = Y

Rank 2 = N

Rank 3 = Y

The target predicted is Y

For Lab-5

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

dt = pd.read_csv('petrol_consumption.csv')
X = dt.drop(['Petrol_Consumption'], axis=1)
y = dt['Petrol_Consumption']

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

regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5

print("Mean Absolute Error: ", mae)
print("Mean Squared Error: ", mse)
print("Root Mean Squared Error: ", rmse)

```

(a) The accuracy score for the Iris dataset is 1.0. The confusion matrix says that there is no misclassification. All classes are correctly classified/labelled.

U R B A N E D G E																																									
Ques 1) We need to visualize the tree, examine the nodes and identify the important features. In decision tree classifier, discrete class labels are predicted for each instance. In regression tree, we assign a continuous value for each instance. It aims to make a split that minimizes the variance of target variable.																																									
<u>For Lab-6</u>																																									
<pre>import pandas as pd from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import classification_report, confusion_matrix, accuracy_score</pre>																																									
<pre>iris = load_iris() X = iris.data y = iris.target</pre>																																									
<pre>X-train, X-test, y-train, y-test = train-test-split(X, y, testsize=0.2, random_state=42)</pre>																																									
<pre>knn = KNeighborsClassifier(n_neighbors=3) knn.fit(X-train, y-train) y-pred = knn.predict(X-test)</pre>																																									
<pre>accuracy = accuracy_score(y-test, y-pred) conf-matrix = confusion_matrix(y-test, y-pred) class-report = classification_report(y-test, y-pred)</pre>																																									
<pre>print("Accuracy:", accuracy) print("Confusion matrix:", conf-matrix) print("Classification Report:", class-report)</pre>																																									
<pre>2/8 Accuracy: 1.0 Confusion Matrix : [[10 0 0 0 9 0 0 0 1]]</pre>																																									
<p>U R B A N E D G E</p> <p>Classification Report :</p> <table border="1"> <thead> <tr> <th></th> <th>precision</th> <th>recall</th> <th>f-score</th> <th>support</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1.00</td> <td>1.00</td> <td>1.00</td> <td>10</td> </tr> <tr> <td>1</td> <td>1.00</td> <td>1.00</td> <td>1.00</td> <td>9</td> </tr> <tr> <td>2</td> <td>1.00</td> <td>1.00</td> <td>1.00</td> <td>11</td> </tr> <tr> <td>accuracy</td> <td>1.00</td> <td>1.00</td> <td>1.00</td> <td>30</td> </tr> <tr> <td>macro avg</td> <td>1.00</td> <td>1.00</td> <td>1.00</td> <td>30</td> </tr> <tr> <td>weighted avg</td> <td>1.00</td> <td>1.00</td> <td>1.00</td> <td>30</td> </tr> </tbody> </table>								precision	recall	f-score	support	0	1.00	1.00	1.00	10	1	1.00	1.00	1.00	9	2	1.00	1.00	1.00	11	accuracy	1.00	1.00	1.00	30	macro avg	1.00	1.00	1.00	30	weighted avg	1.00	1.00	1.00	30
	precision	recall	f-score	support																																					
0	1.00	1.00	1.00	10																																					
1	1.00	1.00	1.00	9																																					
2	1.00	1.00	1.00	11																																					
accuracy	1.00	1.00	1.00	30																																					
macro avg	1.00	1.00	1.00	30																																					
weighted avg	1.00	1.00	1.00	30																																					
<p>(B1) For choosing k-value - we loop through values from 1 to 20 & choose the one that has maximum accuracy.</p>																																									
<p>(B2) To normalize or standardize the range of values for different features/variables to ensure no single feature dominates the learning process due to its scale, which can improve model performance & scalability.</p>																																									

Code:

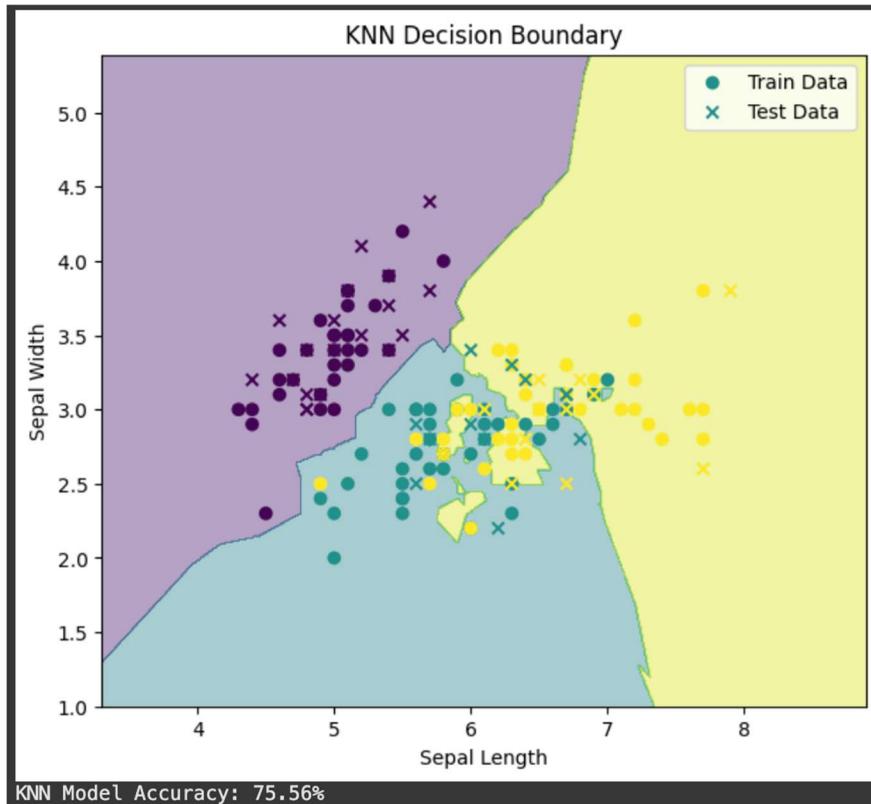
Using Iris Dataset and visualizing:

```
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}%')

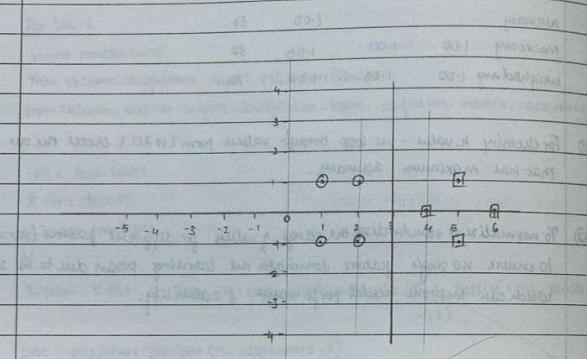
```



Program 7

Build Support vector machine model for a given dataset

Screenshots

<p>8/4/25 LAB-7: SVM</p> <p>draw an optimal hyperplane using linear SVM to classify the following points</p> <p>$\{(1,1), (2,1), (1,-1), (2,-1)\}$ - +ve labelled $\{(4,0), (5,1), (5,-1), (6,0)\}$ - -ve labelled</p>  <p>$s_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, s_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, s_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$</p> <p>$\tilde{s}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \tilde{s}_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \tilde{s}_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$</p> <p>$\alpha_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3 = +1$ $\alpha_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3 = +1$ $\alpha_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3 = -1$</p> <p>$\alpha_1(6) + \alpha_2(4) + \alpha_3(9) = +1 \quad \alpha_1 = 13/4$ $\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1 \quad \alpha_2 = 13/4$ $\alpha_1(9) + \alpha_2(4) + \alpha_3(17) = -1 \quad \alpha_3 = -7/2$</p> <p>$w = \alpha_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3$ $= 13/4 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 13/4 \begin{bmatrix} 2 \\ -1 \end{bmatrix} + -7/2 \begin{bmatrix} 4 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} = -\begin{bmatrix} 1 \\ 0 \end{bmatrix}$</p>	<p>URBAN EDGE</p> <p>$2 = \begin{bmatrix} +1 \\ 0 \end{bmatrix}, b = 3, b+3 = 0$</p> <p>intercept on x-axis = 3 $2 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow$ line parallel to y-axis</p> <p>(ss) Iris dataset.</p> <pre> import pandas as pd import sklearn.svm import SVC from sklearn.model_selection import train_test_split import sklearn as sns url = "iris/iris.data" columns = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'species'] iris = pd.read_csv(url, header=None) iris['species'] = iris['species'].map({0: 'Iris-setosa', 1: 'Iris-versicolor', 2: 'Iris-virginica'}) X = iris.drop('species', axis=1) Y = iris['species'] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) sum_linear = SVC(kernel='linear', random_state=42) sum_linear.fit(X_train, y_train) y_pred_linear = sum_linear.predict(X-test) accuracy_linear = accuracy_score(y-test, y_pred_linear) conf_matrix_linear = confusion_matrix(y-test, y_pred_linear) sum_rbt = SVC(kernel='rbf', random_state=42) sum_rbt.fit(X_train, y_train) y_pred_rbt = sum_rbt.predict(X-test) accuracy_rbt = accuracy_score(y-test, y_pred_rbt) conf_matrix_rbt = confusion_matrix(y-test, y_pred_rbt) print("Accuracy of SVM with Linear kernel:", accuracy_linear) print("Accuracy of SVM with RBF kernel:", accuracy_rbt) </pre>
--	--

```

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
sns.heatmap(cov_matrix_linear, annot=True, fmt='d', cmap='Blues',
            xticklabels=iris['species'].unique(),
            yticklabels=iris['species'].unique(),
            ax=axes[0])
axes[0].set_title('Confusion Matrix - linear kernel')
plt.show()

```

O/P

Accuracy of SVM with Linear Kernel : 1.00

Accuracy of SVM with RBF Kernel : 1.00

- (Q1) SVM w/ Linear Kernel & SVM with RBF Kernel both gave accuracy of 1.00.
 For Iris dataset, both kernels give the same perfect accuracy, because
 the dataset is linearly separable - decision boundaries b/w the classes
 are simple.

Linear kernel is more efficient - fewer parameters to tune.

- (Q2) Overall Accuracy - 85.45%
 Common confusions - letters such as C, G, Q, O or I, J, L, T show more
 confusion among each other. (visually similar - easier to misclassify)
 Avg AUC Score - 0.987 - excellent performance across all 26 letter classes

Iris dataset has only 3 well separated classes - fewer features & samples.
 Letter recognition is more complex

Code:

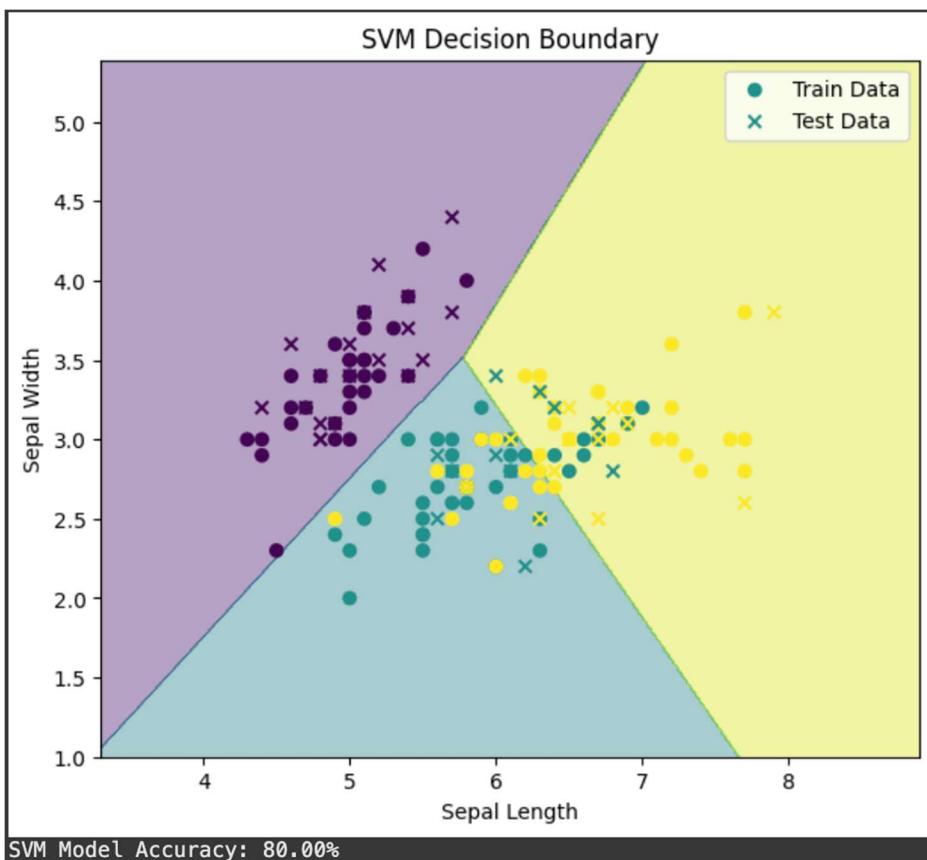
Using Iris Dataset and visualizing:

```
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}%")

```



Program 8

Implement Random forest ensemble method on a given dataset.

Screenshots

15/4/25 LAB 8 - Random Forest

(Q1) Differences b/w Decision Trees & Random Forest

Decision Tree Classifier	Random Forest Classifier
* A single decision tree that splits the data on feature values to make predictions	* An ensemble of multiple decision trees (1000s). Each tree built using random subset of data & features.
* Prone to overfitting	* Less prone to overfitting.
* high variance, low bias	* Low variance, comparable bias
* Single class prediction (majority vote)	* Majority vote from multiple trees
* May perform poorly on unseen data	* More robust, better generalization.
* Train faster due to single tree	* Slower due to multiple trees.
* Can be affected by class imbalance	* Better handling through bootstrapping
* Uses all features in each split	* Randomly selects features for each split.

(Q2)

- * n_estimators (int, default=100) - defines the number of trees in the forest.
More trees - improves performance, less compute time.
- * max_depth (int, default=None) - specifies max depth of each tree. Limiting depth controls overfitting by preventing complex trees.
- * min_samples_split (int or float, default=2) - min no. of samples required to split an internal node.
- * min_samples_leaf (int or float, default=1) - min no. of leaf nodes samples at each node. Smoothens model.
- * max_features ('auto', 'sqrt', 'log2', float, default='auto') - the no. of features to consider while looking for best split.
- * bootstrap (bool, default=True) - determines whether or not to use bootstrap sampling (with replacement) while building trees.
- * oob_score (bool, default=False) - whether to use out-of-bag samples to estimate generalization error.

<p>(83) Input: Training the dataset of features & labels Select: Number of decision trees to build (n-estimators)</p> <p>For each tree:</p> <ul style="list-style-type: none"> draw a bootstrap sample (random sample of replacement) from dataset. Grow a decision tree from one sample: <ul style="list-style-type: none"> At each split, consider only random subset of features Split nodes based on the feature among the subset. Continue until stopping criteria (e.g.: max_depth) <p>Aggregate predictions:</p> <ul style="list-style-type: none"> For classification: use majority voting across all trees. For regression (if applicable): use avg. of outputs. <p>Output: Final prediction based on ensemble voting.</p> <pre> code from sklearn.datasets import load_iris from sklearn.ensemble import RandomForestClassifier from sklearn.model_selection import train_test_split import matplotlib.pyplot as plt iris = load_iris() X, y = iris.data, iris.target y_binarized = label_binarize(y, classes=[0, 1, 2]) X_train, X_test, y_train, y_test = train_test_split(X, y_binarized, test_size=0.2, random_state=42) n_estimators_list = [10, 20, 30] auc_scores = [] rf_model_default = RandomForestClassifier(n_estimators=10, random_state=42) rf_model_default.fit(X_train, y_train) </pre>	<p>accuracy_default = rf_model_default.score(X_test, y_test)</p> <p>print("Accuracy with default n-estimators = 10 : ", accuracy_default * 100)</p> <p>best_accuracy = 0</p> <p>best_n_estimators = 10</p> <p>for n_estimators in [10, 50, 100, 200, 500]: rf_model = RandomForestClassifier(n_estimators=n_estimators, random_state=42) rf_model.fit(X_train, y_train) y_pred = rf_model.predict(X_test) accuracy = accuracy_score(y_test, y_pred) if accuracy > best_accuracy: best_accuracy = accuracy best_n_estimators = n_estimators print("Best Accuracy with n-estimators = " + str(best_n_estimators) + " : " + str(best_accuracy * 100))</p> <p>Output:</p> <p>accuracy with default n-estimators = 10 : 100.00%</p> <p>Best accuracy achieved with n-estimators = 10 : 100.00%.</p> <p>(81) Accuracy remains same for n = 10, 20, 30</p> <p>Accuracy ~ 1.00</p> <p>(82) Acc Score</p> <table border="1"> <caption>Data points estimated from the graph</caption> <thead> <tr> <th>n-estimators</th> <th>Acc Score</th> </tr> </thead> <tbody> <tr><td>10</td><td>1.00</td></tr> <tr><td>15</td><td>1.00</td></tr> <tr><td>20</td><td>1.00</td></tr> <tr><td>25</td><td>1.00</td></tr> <tr><td>30</td><td>1.00</td></tr> </tbody> </table>	n-estimators	Acc Score	10	1.00	15	1.00	20	1.00	25	1.00	30	1.00
n-estimators	Acc Score												
10	1.00												
15	1.00												
20	1.00												
25	1.00												
30	1.00												

Code:

```

from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

iris = load_iris()
X, y = iris.data, iris.target
y_binarized = label_binarize(y, classes=[0, 1, 2])
X_train, X_test, y_train, y_test = train_test_split(X, y_binarized, test_size=0.2, random_state=42)
n_estimators_list = [10, 20, 30]

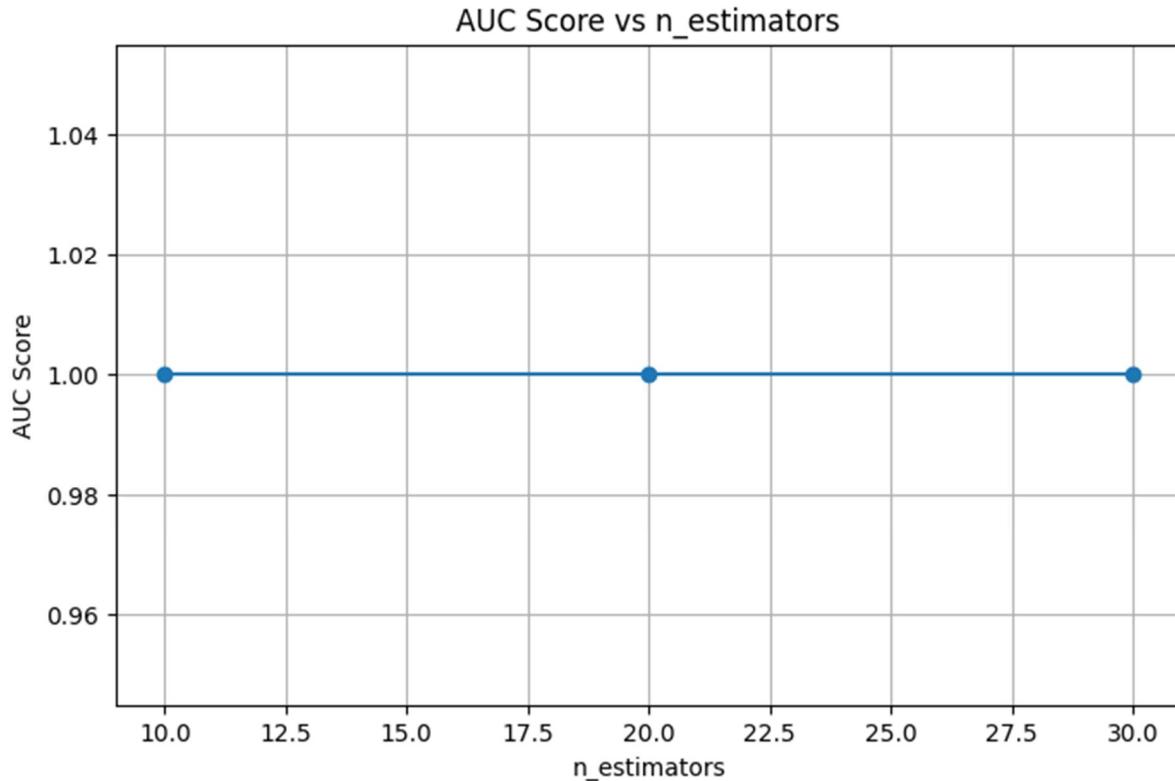
```

```

auc_scores = []
for n in n_estimators_list:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_prob = rf.predict_proba(X_test)
    y_prob_combined = [prob[:, 1] for prob in y_prob]
    y_prob_stack = list(map(list, zip(*y_prob_combined)))
    auc = roc_auc_score(y_test, y_prob_stack, average='macro', multi_class='ovr')
    auc_scores.append(auc)

plt.figure(figsize=(8, 5))
plt.plot(n_estimators_list, auc_scores, marker='o', linestyle='-')
plt.title("AUC Score vs n_estimators")
plt.xlabel("n_estimators")
plt.ylabel("AUC Score")
plt.grid(True)
plt.show()

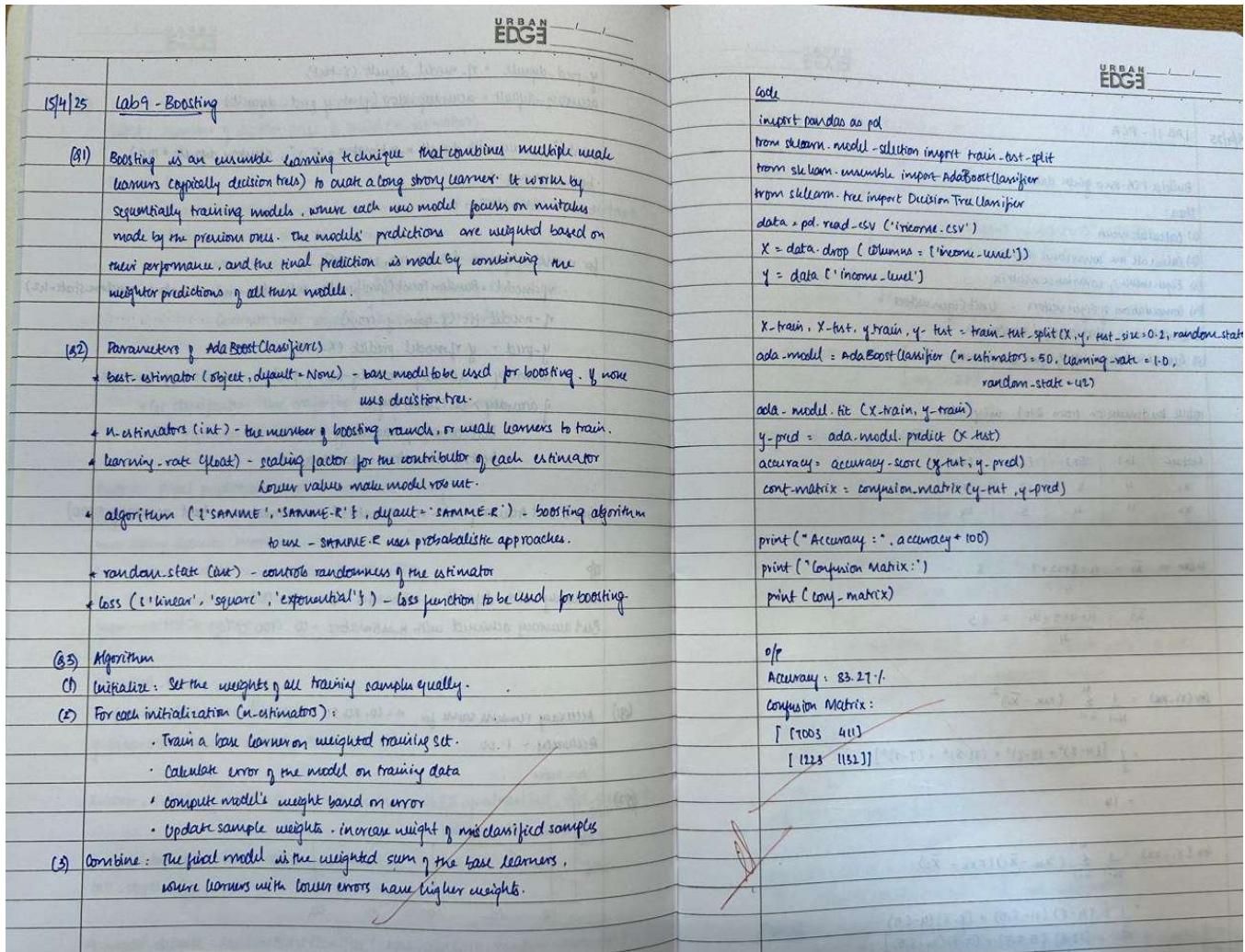
```



Program 9

Implement Boosting ensemble method on a given dataset.

Screenshots



Code:

```

import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
iris = load_iris()

```

```

X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
n_estimators_values = [50, 100, 150]
learning_rates = [0.1, 0.5, 1.0]
print("Results with DecisionTreeClassifier as base estimator:")
for n in n_estimators_values:
    for lr in learning_rates:
        model = AdaBoostClassifier(
            estimator=DecisionTreeClassifier(max_depth=1),
            n_estimators=n,
            learning_rate=lr,
            random_state=42
        )
        model.fit(X_train, y_train)
        preds = model.predict(X_test)
        acc = accuracy_score(y_test, preds)
        print(f'n_estimators={n}, learning_rate={lr} -> Accuracy: {acc * 100:.2f}%')
print("\nResults with LogisticRegression as base estimator:")
for n in n_estimators_values:
    for lr in learning_rates:
        model = AdaBoostClassifier(
            estimator=LogisticRegression(max_iter=1000, random_state=42),
            n_estimators=n,
            learning_rate=lr,
            random_state=42
        )
        model.fit(X_train, y_train)
        preds = model.predict(X_test)
        acc = accuracy_score(y_test, preds)
        print(f'n_estimators={n}, learning_rate={lr} -> Accuracy: {acc * 100:.2f}%')

```

```
Results with DecisionTreeClassifier as base estimator:  
n_estimators=50, learning_rate=0.1 -> Accuracy: 100.00%  
n_estimators=50, learning_rate=0.5 -> Accuracy: 96.67%  
n_estimators=50, learning_rate=1.0 -> Accuracy: 93.33%  
n_estimators=100, learning_rate=0.1 -> Accuracy: 100.00%  
n_estimators=100, learning_rate=0.5 -> Accuracy: 100.00%  
n_estimators=100, learning_rate=1.0 -> Accuracy: 93.33%  
n_estimators=150, learning_rate=0.1 -> Accuracy: 100.00%  
n_estimators=150, learning_rate=0.5 -> Accuracy: 96.67%  
n_estimators=150, learning_rate=1.0 -> Accuracy: 93.33%
```

```
Results with LogisticRegression as base estimator:  
n_estimators=50, learning_rate=0.1 -> Accuracy: 100.00%  
n_estimators=50, learning_rate=0.5 -> Accuracy: 100.00%  
n_estimators=50, learning_rate=1.0 -> Accuracy: 93.33%  
n_estimators=100, learning_rate=0.1 -> Accuracy: 100.00%  
n_estimators=100, learning_rate=0.5 -> Accuracy: 100.00%  
n_estimators=100, learning_rate=1.0 -> Accuracy: 93.33%  
n_estimators=150, learning_rate=0.1 -> Accuracy: 100.00%  
n_estimators=150, learning_rate=0.5 -> Accuracy: 100.00%  
n_estimators=150, learning_rate=1.0 -> Accuracy: 93.33%
```

Program 10

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

Screenshots

LAB 10 - KMEANS	
29/4/25	URBAN EDGE
1. Algorithm	(1) Initialize k -cluster centroids randomly. (2) Assign each datapoint to the nearest centroid. (3) Recompute centroids as the mean of assigned points. (4) Repeat steps 2-3 until centroids converge or max iterations reached.
2. * Use Elbow method or Gap Statistic	* Elbow is most common - it analyzes curve of SEE vs k .
3. Formula for sum of squared Errors (SSE)	$SSE = \sum_{i=1}^k \sum_{x \in C_i} \ x - \mu_i\ ^2$ μ_i is cluster i μ_i is its centroid.
4. We run K-mean for a range of k (4 to 10)	
5. compute SSE for each	Plot k on x-axis, SSE on y-axis At first SSE drops quickly as k increases. At a certain pt the rate of decrease slows - this elbow pt indicates optimal k value. This pt suggests good balance b/w compactness & no. of clusters.
6. Parameters	* n_clusters - number of clusters (k) * init - method to initialize centroids ('k-means++', 'random') * n_init - times algo is run with diff centroid seeds. * max_iter - max iterations per run * tol - convergence tolerance * algorithm - 'auto', 'full', 'elkan' * random_state - seed for reproducibility.

```

python code
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

iris = load_iris()
iris_dt = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_dt = iris_dt[['petal length (cm)', 'petal width (cm)']]

```

```

scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_dt)

```

```
SSE = []
```

```
k_range = range(1,11)
```

```
for k in k_range(1,11):
```

```
km = KMeans(n_clusters=k, random_state=42)
```

```
km.fit(iris_scaled)
```

```
SSE.append(km.inertia_)
```

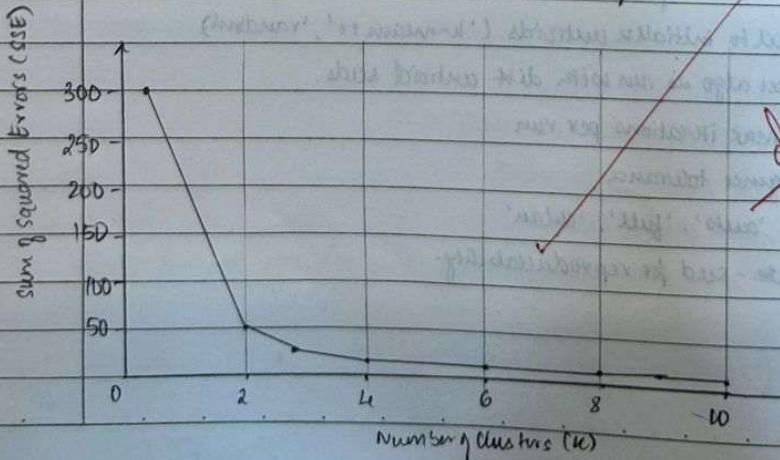
```
plt.figure(figsize=(5,8))
```

```
plt.plot(k_range, SSE, marker='o')
```

```
plt.show()
```

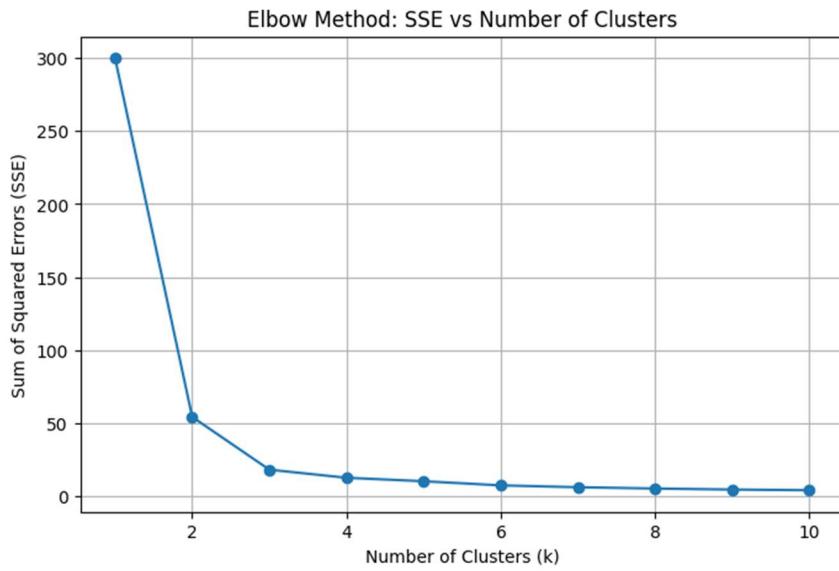
O/P

Elbow Method : SSE vs No. of Clusters



Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df = iris_df[['petal length (cm)', 'petal width (cm)']] # drop other features
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_df)
sse = []
k_range = range(1, 11)
for k in k_range:
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(iris_scaled)
    sse.append(km.inertia_)
plt.figure(figsize=(8, 5))
plt.plot(k_range, sse, marker='o')
plt.title("Elbow Method: SSE vs Number of Clusters")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Sum of Squared Errors (SSE)")
plt.grid(True)
plt.show()
```



Program 11

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

Screenshots

29/4/25 LAB 11 - PCA

Build a PCA on a given dataset

Steps:

- (1) Calculate mean
- (2) Calculate the covariance matrix Σ
- (3) Eigen values & covariance matrix
- (4) Computation of Eigen vectors - Unit Eigen vectors
- (5) Computation of first principal component
- (6) Geometrical meaning of first components

reduce dimension from 2 to 1 using PCA.

Feature	Ex1	Ex2	Ex3	Ex4
x_1	4	8	13	7
x_2	11	4	5	14

Mean $\Rightarrow \bar{x}_1 = \frac{4+8+13+7}{4} = 8$

$\bar{x}_2 = \frac{11+4+5+14}{4} = 8.5$

$\text{Cov}(x_1, x_1) = \frac{1}{N-1} \sum_{k=1}^N (x_{1k} - \bar{x}_1)^2$

$= \frac{1}{3} [(4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2] = 14$

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

$= \frac{1}{3} [(4-8)(11-8.5) + (8-8)(4-8.5) + (13-8)(5-8.5) + (7-8)(14-8.5)] = -11$

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

$\text{Cov}(x_2, x_2) = \frac{1}{N-1} \sum_{k=1}^N (x_{2k} - \bar{x}_2)^2$

$= \frac{1}{3} [(11-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2] = 23$

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

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

$= \frac{1}{3} [(11-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2] = 23$

Cov matrix $= C = \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$

Eigen values & Eigen vectors.

$C - \lambda I = \begin{bmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{bmatrix} = 0$

$(14-\lambda)(23-\lambda) - (121) = 0$

$\lambda^2 - 37\lambda + 201 = 0$

$\lambda = 37 \pm \sqrt{565} \Rightarrow \lambda_1 = 30.3849, \lambda_2 = 6.6151$

Select $\lambda_1 - \max \lambda$

$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$

$\begin{bmatrix} 14 - \lambda_1 & -11 \\ -11 & 23 - \lambda_1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = 0$

① $(14 - \lambda_1)u_1 + -11u_2 = 0$

② $-11u_1 + (23 - \lambda_1)u_2 = 0$

From ① $\Rightarrow \frac{u_1}{11} = \frac{u_2}{14 - \lambda_1} = t \Rightarrow u_1 = 11t, u_2 = (14 - \lambda_1)t \Rightarrow U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 11t \\ (14 - \lambda_1)t \end{bmatrix}$

Handwritten Calculations	Python code
$\ U_1\ _U = \sqrt{11^2 + (14 - 11)^2} \\ = \sqrt{11^2 + (14 - 10.305)^2} \\ = 19.7348$	
<p>unit eigenvector corresponding to λ_1</p> $e_1 = \begin{bmatrix} 11/\ U_1\ _U \\ (14 - 11)/\ U_1\ _U \end{bmatrix} = \begin{bmatrix} 11/(19.7348) \\ (14 - 11)/(19.7348) \end{bmatrix}$ $= \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$	
<p>(14 - 11) Similarly we obtain e_2 from eqn ②</p> $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$	<pre>from sklearn.datasets import load_digits from sklearn.model_selection import train_test_split from sklearn.linear_model import LogisticRegression from sklearn.decomposition import PCA</pre>
<p>Let $\begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix}$ be the k-th sample in above dataset.</p> <p>The first PCA component:</p> $e_1^T \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix} = [0.5574 \quad -0.8303] \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix}$ $= (0.5574)(x_{1k} - \bar{x}_1) - (0.8303)(x_{2k} - \bar{x}_2)$	<pre>digit = load_digits() X, y = digit.data, digit.target scaler = StandardScaler() X_scaled = scaler.fit_transform(X) x_train, x_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42) lr = LogisticRegression(max_iter=1000) lr.fit(x_train, y_train) preds_lr = lr.predict(x_test) print("Accuracy without PCA:", accuracy_score(y_test, preds_lr))</pre>
	<pre>pca = PCA(n_components=2) X_pca = pca.fit_transform(X_scaled) x_train_pca, x_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2, random_state=42) lr_pca = LogisticRegression(max_iter=1000) lr_pca.fit(x_train_pca, y_train_pca) preds_pca = lr_pca.predict(x_test_pca) print("Accuracy with PCA:", accuracy_score(y_test_pca, preds_pca))</pre>

Code:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
digits = load_digits()
X, y = digits.data, digits.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
preds = lr.predict(X_test)
print(f'Accuracy without PCA: {accuracy_score(y_test, preds):.4f}')
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2,
random_state=42)
lr_pca = LogisticRegression(max_iter=1000)
lr_pca.fit(X_train_pca, y_train_pca)
preds_pca = lr_pca.predict(X_test_pca)
print(f'Accuracy with PCA (2 components): {accuracy_score(y_test_pca, preds_pca):.4f}')
```

```
Accuracy without PCA: 0.9722
Accuracy with PCA (2 components): 0.5389
```