VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Nishanth K S (1BM22CS183)

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 **Nishanth K S (1BM22CS183),** 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.

Lab Faculty Incharge	
Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

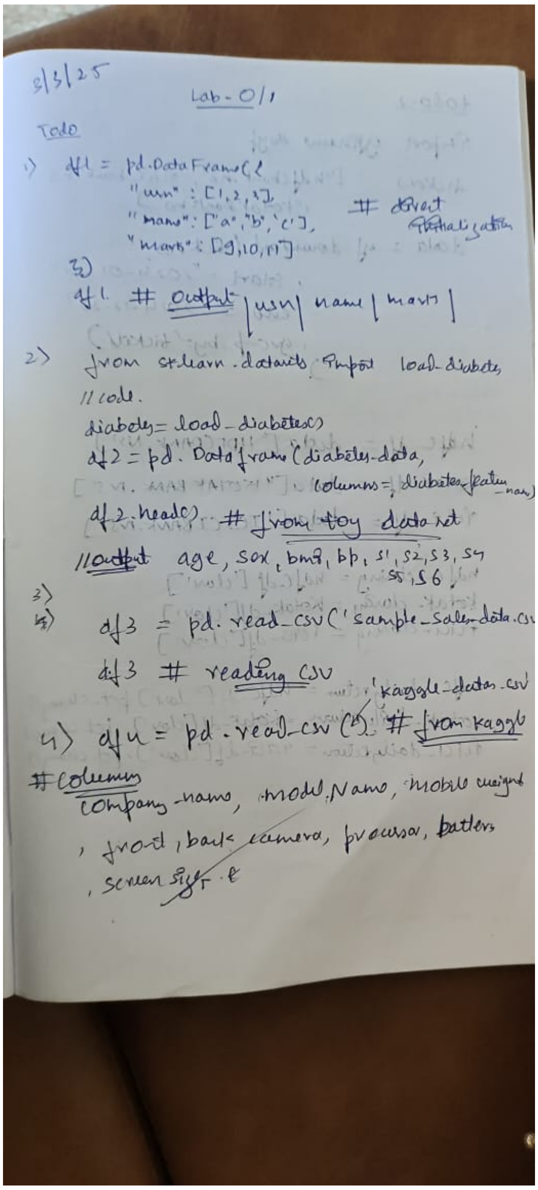
Github Link:

https://github.com/nishanthhks/ML_LAB

Program 1

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

Screenshot



Open in Colab

```
import pandas as pd

data = {
    'Name' : ['alice', 'bob', 'charlie'], 'Age' : [25, 30, 35],
    'City' : ['New York', 'San Francisco', 'Los Angeles']
}

df=pd.DataFrame(data) df
df.to_csv('sample.csv')
```

Start coding or generate with AI.

```
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
```

TO - DO LIST

```
# 1
data2 = {
    "USN" : [19, 20, 21],
    "Name" : ["Adi", 'Charlie', "Bobby"], "Marks" : [29,
    40, 50]
}

df2=pd.DataFrame(data2) df2
```

	USN	Name	Marks
0	19	Adi	29
1	20	Charlie	40
2	21	Bobby	50

2

from sklearn.datasets import load_diabetes diabetes =

load_diabetes()
df3=pd.DataFrame(diabetes.data, columns=diabetes.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
```

# 3
df4=pd.read_csv("/content/sample.csv") df4.head()

	Unnamed: 0	Name	Age	City
0	0	alice	25	New York
1	1	bob	30	San Francisco
2	2	charlie	35	Los Angeles

4

df5=pd.read_csv("/content/Dataset of Diabetes .csv") df5.head()

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

STOCK MARKET DATA ANALYSIS

```
import yfinance as yf import
pandas as pd
import matplotlib.pyplot  as plt

tickers = ["RELIANCE.NS", "TCS.NS", "INFY,NS"]

data = yf.download(tickers, start="2022-01-01", end="2023-01-01", group_by='ticker') print("First 5 rows of data")
print(data.head())

# 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
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
```

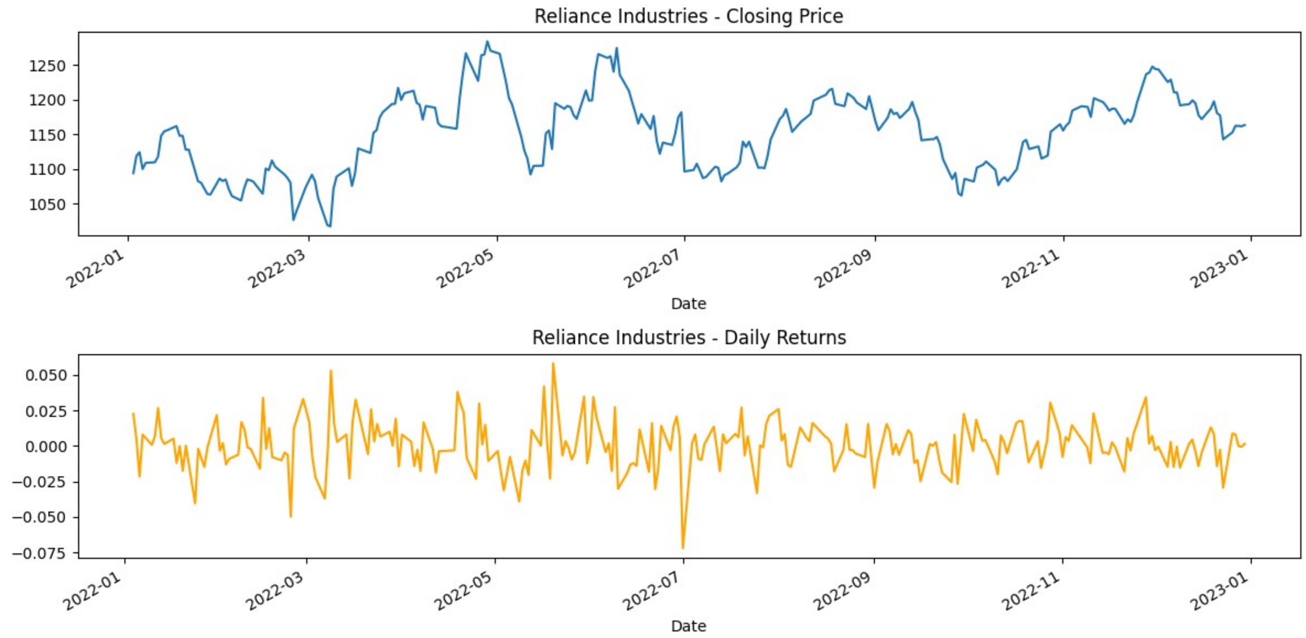
```
[**********************100%***********************]  3 of 3 completed ERROR:yfinance:
1 Failed download:
ERROR:yfinance:['INFY,NS']: YFTzMissingError('possibly delisted; no timezone found') First 5 rows of data
Ticker          INFY,NS                                              RELIANCE.NS                 \
Price                  Open  High  Low   Close  Adj   Close  Volume         Open          High
Date
2022-01-03              NaN   NaN  NaN   NaN          NaN     NaN    1076.584961   1096.136465
2022-01-04              NaN   NaN  NaN   NaN          NaN     NaN    1099.755428   1120.285608
2022-01-05              NaN   NaN  NaN   NaN          NaN     NaN    1120.740874   1127.569096
2022-01-06              NaN   NaN  NaN   NaN          NaN     NaN    1115.824417   1117.099057
2022-01-07              NaN   NaN  NaN   NaN          NaN     NaN    1106.606442   1118.942732
```

```
Ticker                                                              TCS.NS              \
Price                Low          Close       Volume          Open            High
Date
2022-01-03     1075.924884   1094.270020    5421611    3465.941320     3539.881401
2022-01-04     1094.338268   1118.965454   10847728    3540.898636     3594.551234
2022-01-05     1107.516816   1124.200439   11643813    3572.230567     3576.851822
2022-01-06     1096.614271   1100.028442   14447422    3523.245916     3544.503696
2022-01-07     1097.775231   1108.905273   13112115    3530.638909     3572.137690

Ticker
Price Date           Low          Close       Volume
2022-01-03
               3461.320065   3528.559326    2346158
2022-01-04     3522.968028   3590.484619    2488606
2022-01-05     3523.614868   3568.487305    1733031
2022-01-06     3486.275864   3519.040527    1810293
2022-01-07     3508.826495   3561.601318    2460591
```

```
Shape of the dataset:
(248, 16)

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

Summary statistics for Reliance Industries:
Price          Open          High          Low          Close          Volume
```

```
# 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='ora plt.tight_layout()
plt.show()
```

C C




```
# 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'.
```

Using the code given in the above slides, do the exercise of the “Stock Market Data Analysis”, considering the follwoing

1. HDFC Bank Ltd. , ICICI Bank Ltd , Kotak Mahindra Bank Ltd. tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
2. Start date: 2024-01-01, End date: 2024-12-30
3. Plot the closing price and daily returns for all the three banks mentioned.

```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')
```

print("First 5 rows of data") print(data.head())

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)

```
[*********************100%***********************]  3 of 3 completedFirst 5 rows of da
Ticker          ICICIBANK.NS                                                        \
Price 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          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

Ticker Price    HDFCBANK.N
                S                     High           Low         Close      Volume
                        Open
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

Shape of the dataset:
(244, 15)

Column names:
    MultiIndex([('ICICIBANK.NS',    'Open'),
                ('ICICIBANK.NS',    'High'),
                ('ICICIBANK.NS',     'Low'),
                ('ICICIBANK.NS',   'Close'),
                ('ICICIBANK.NS',  'Volume'),
                ('KOTAKBANK.NS',    'Open'),
                ('KOTAKBANK.NS',    'High'),
                ('KOTAKBANK.NS',     'Low'),
                ('KOTAKBANK.NS',   'Close'),
                ('KOTAKBANK.NS',  'Volume'),
                ( 'HDFCBANK.NS',    'Open'),
                ( 'HDFCBANK.NS',    'High'),
                ( 'HDFCBANK.NS',     'Low'),
                ( 'HDFCBANK.NS',   'Close'),
```

```
( 'HDFCBANK.NS', 'Volume')],
names=['Ticker',  'Price'])
```

HDFC BANK

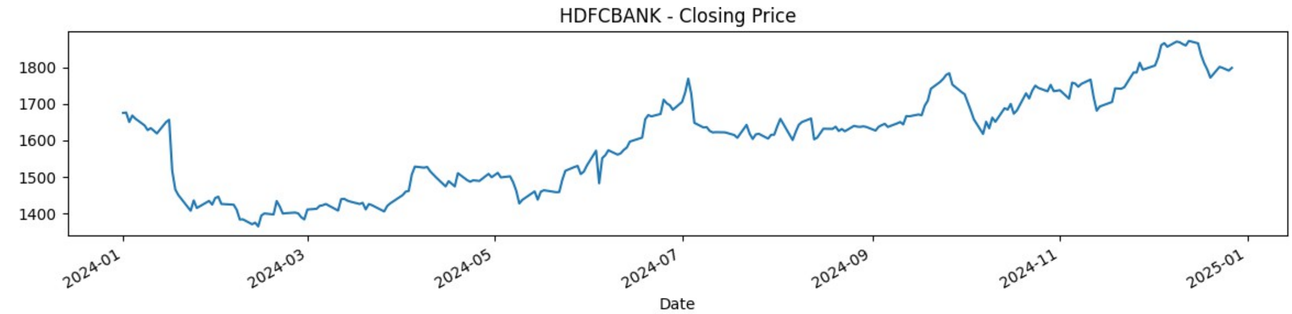
```
# Summary statistics for a specific stock (e.g., Reliance) HDFCBANK =
data['HDFCBANK.NS']
print("\nSummary statistics for HDFCBANK Industries:") print(reliance_data.describe())

# Calculate daily returns
HDFCBANK['Daily Return'] = HDFCBANK['Close'].pct_change()


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

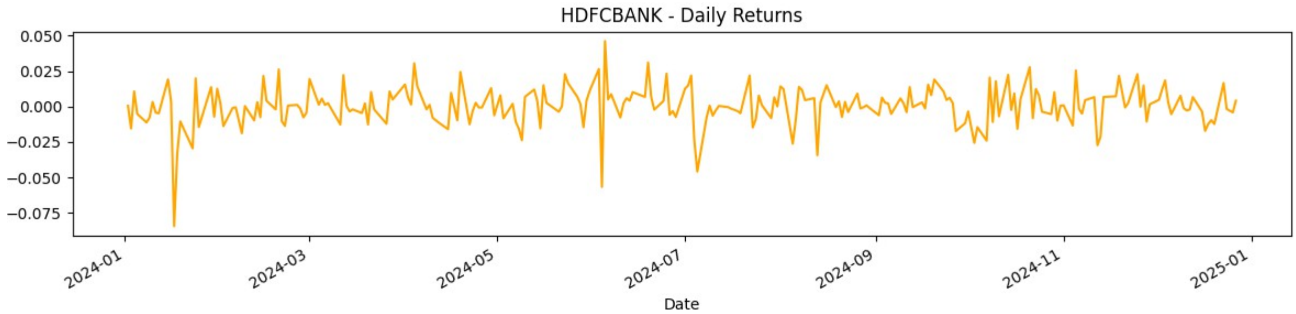
Summary statistics for HDFCBANK Industries:

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

<ipython-input-54-52b3f0e5df81>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use HDFCBANK['Daily Return'] = HDFCBANK['Close'].pct_change()





ICICI BANK

# Summary statistics for a specific stock (e.g., Reliance) ICICIBANK = data['ICICIBANK.NS']
print("\nSummary statistics for ICICIBANK Industries:") print(reliance_data.describe())

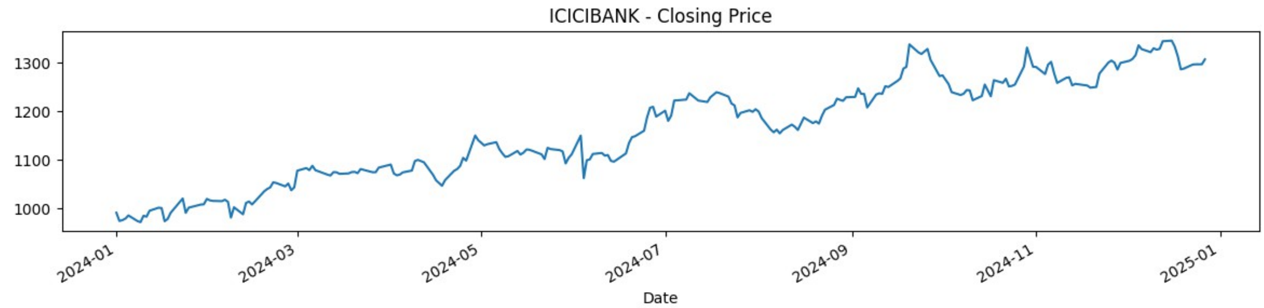
```
# Calculate daily returns
ICICIBANK['Daily Return'] = ICICIBANK['Close'].pct_change()

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

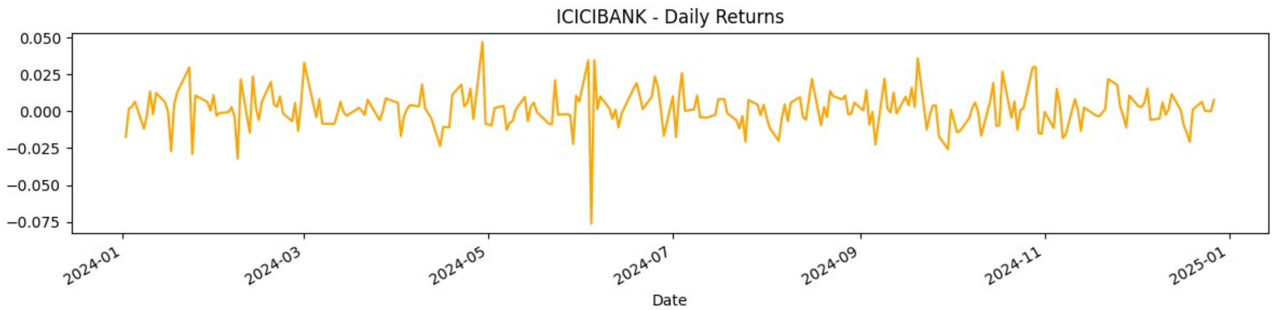
Summary statistics for ICICIBANK Industries:

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

<ipython-input-51-8f0b2e87ee7d>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use ICICIBANK['Daily Return'] = ICICIBANK['Close'].pct_change()





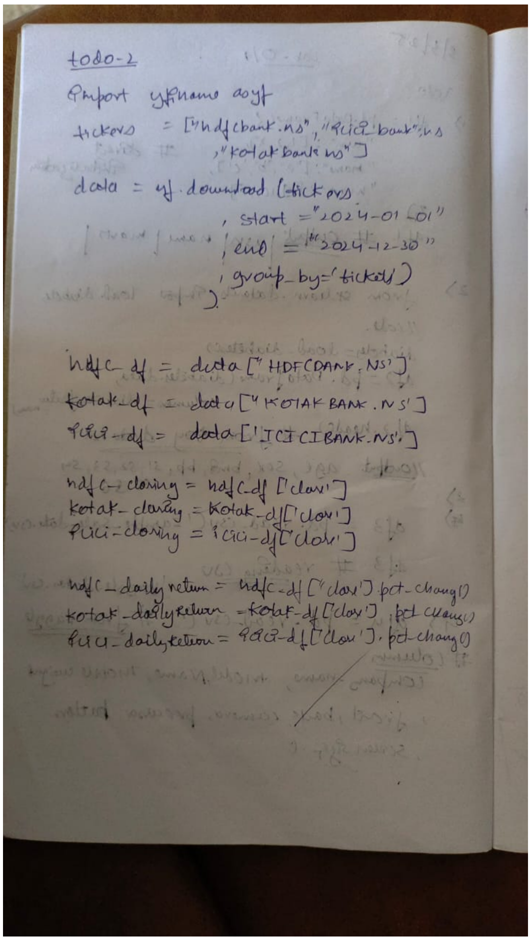
KOTAK BANK

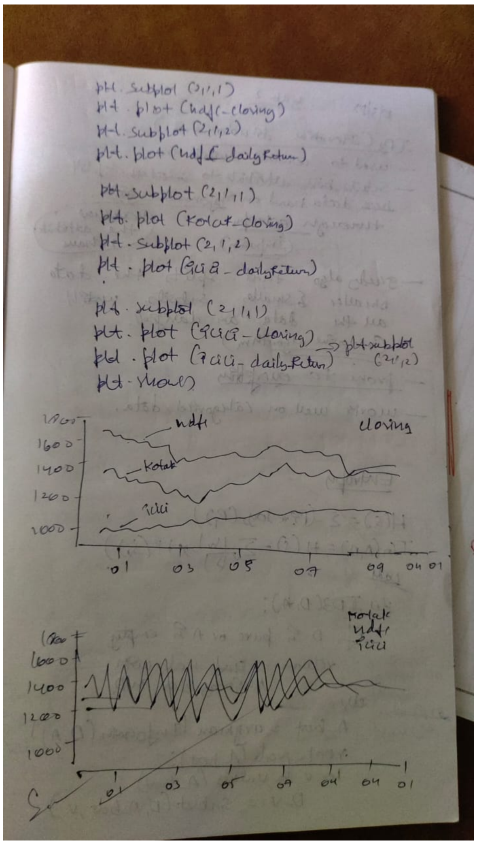
```
# Summary statistics for a specific stock (e.g., Reliance) KOTAKBANK =
data['KOTAKBANK.NS']
print("\nSummary statistics for KOTAKBANK Industries:") print(reliance_data.describe())

# Calculate daily returns
KOTAKBANK['Daily Return'] = KOTAKBANK['Close'].pct_change()


# 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()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot





Open in Colab

```
import pandas as pd

# i. To load .csv file into the data frame # df =
pd.read_csv("housing.csv")

# ii. To display information of all columns print("Information of all columns:")
print(df.info())

# iii. To display statistical information of all numerical columns print("\nStatistical information
of all numerical columns:")
print(df.describe())

# iv. To display the count of unique labels for the "Ocean Proximity" column print("\nCount of unique labels for 'Ocean
Proximity' column:")
print(df['ocean_proximity'].value_counts())

# v. To display which attributes (columns) in a dataset have missing values count greater th print("\nColumns with missing values
count greater than zero:")
missing_values = df.isnull().sum()
missing_columns=missing_values[missing_values>0]
print(missing_columns)
```

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

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

```
Count of unique labels for 'Ocean Proximity' column:
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
```

Write Python code to implement the following data preprocessing techniques for Diabetes and Adult income data sets

Data Preprocessing techniques:

1. Data Cleaning: Handling Missing Values, Handling categorical data, Handling Outliers
2. Data Transformations: Min-max Scaler/Normalization , Standard

Scaler Download the following dataset files and upload in your Google

Colab folder

I. Diabetes datasets

https://data.mendeley.com/datasets/wj9rwkp9c2/1

II. Adult income dataset

https://www.kaggle.com/datasets/wenruliu/adult-income-dataset

DIABETES DATASET

```
import pandas as pd import
numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler from sklearn.impute
import SimpleImputer
from sklearn.preprocessing import LabelEncoder import
matplotlib.pyplot as plt
```

```
# Load Diabetes Dataset
diabetes_df=pd.read_csv("/content/Dataset of Diabetes .csv") print("Diabetes Dataset Loaded")
print(diabetes_df.head())

# -------------------- Data Preprocessing for Diabetes Dataset -------------------- # 1. Handling Missing Values in Diabetes
Dataset
# Impute missing values for numerical columns with mean and categorical with most frequent v numerical_cols =
diabetes_df.select_dtypes(include=[np.number]).columns
categorical_cols = diabetes_df.select_dtypes(include=['object']).columns

# Impute missing values for numerical columns (mean strategy) num_imputer =
SimpleImputer(strategy='mean')
diabetes_df[numerical_cols] = num_imputer.fit_transform(diabetes_df[numerical_cols])

# Impute missing values for categorical columns (most frequent strategy) cat_imputer =
SimpleImputer(strategy='most_frequent')
diabetes_df[categorical_cols] = cat_imputer.fit_transform(diabetes_df[categorical_cols])

# 2. Handling Categorical Data (Label Encoding for categorical columns like 'Gender') label_encoder = LabelEncoder()
for col in categorical_cols:
    diabetes_df[col] = label_encoder.fit_transform(diabetes_df[col])

# 3. Handling Outliers in Diabetes Dataset
# Removing outliers based on Z-score (values beyond 3 standard deviations) from scipy import stats
z_scores = np.abs(stats.zscore(diabetes_df[numerical_cols])) outliers = (z_scores >
3).all(axis=1)
diabetes_df_cleaned = diabetes_df[~outliers]

# 4. Data Transformation (Min-Max Scaling and Standardization) # Min-Max Scaling
(Normalization)
min_max_scaler = MinMaxScaler()
diabetes_df_scaled = pd.DataFrame(min_max_scaler.fit_transform(diabetes_df_cleaned), columns

# Standard Scaling (Z-score Normalization) standard_scaler
= StandardScaler()
diabetes_df_standardized = pd.DataFrame(standard_scaler.fit_transform(diabetes_df_cleaned),

# Display results
print("\nProcessed Diabetes Data (after preprocessing):") print(diabetes_df_standardized.head())
```

```
Diabetes Dataset Loaded
     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

Processed Diabetes Data (after preprocessing):
          ID     No_Pation     Gender       AGE       Urea        Cr      HbA1c   \
0   0.672140   -0.074747  -1.139688  -0.401144  -0.144781  -0.382672  -1.334983
1   1.641852   -0.069940   0.870343  -3.130017  -0.212954  -0.115804  -1.334983
2   0.330868   -0.065869  -1.139688  -0.401144  -0.144781  -0.382672  -1.334983
3   1.412950   -0.054126  -1.139688  -0.401144  -0.144781  -0.382672  -1.334983
4   0.680463   -0.069939   0.870343  -2.334096   0.673299  -0.382672  -1.334983

        Chol         TG        HDL        LDL       VLDL        BMI      CLASS
0 -0.509436  -1.035084   1.810756  -1.085457  -0.369958  -1.124622  -2.864124
1 -0.893730  -0.678063  -0.158692  -0.457398  -0.342649  -1.326239  -2.864124
2 -0.509436  -1.035084   1.810756  -1.085457  -0.369958  -1.124622  -2.864124
3 -0.509436  -1.035084   1.810756  -1.085457  -0.369958  -1.124622  -2.864124
4  0.028576  -0.963680  -0.613180  -0.547121  -0.397267  -1.729472  -2.864124
```

ADULT INCOME DATASET

```
import pandas as pd import
numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler from sklearn.impute
import SimpleImputer
from sklearn.preprocessing import LabelEncoder import
matplotlib.pyplot as plt
from scipy import stats

# Load Adult Income Dataset
adult_df = pd.read_csv("adult.csv") print("Adult
Income Dataset Loaded") print(adult_df.head())

# -------------------- Data Preprocessing for Adult Income Dataset -------------------- # 1. Handling Missing Values in Adult
Income Dataset
# Impute missing values for numerical columns with mean and categorical with most frequent numerical_cols_adult =
adult_df.select_dtypes(include=[np.number]).columns
categorical_cols_adult = adult_df.select_dtypes(include=['object']).columns

# Impute missing values for numerical columns (mean strategy) num_imputer_adult =
SimpleImputer(strategy='mean')
adult_df[numerical_cols_adult] = num_imputer_adult.fit_transform(adult_df[numerical_cols_a

# Impute missing values for categorical columns (most frequent strategy) cat_imputer_adult =
SimpleImputer(strategy='most_frequent')
adult_df[categorical_cols_adult] = cat_imputer_adult.fit_transform(adult_df[categorical_co

# 2. Handling Categorical Data (Label Encoding for categorical columns) label_encoder =
LabelEncoder()
for col in categorical_cols_adult:
    adult_df[col] = label_encoder.fit_transform(adult_df[col])

# 3. Handling Outliers in Adult Income Dataset
# Removing outliers based on Z-score (values beyond 3 standard deviations)
```

```
z_scores_adult = np.abs(stats.zscore(adult_df[numerical_cols_adult])) outliers_adult =
(z_scores_adult > 3).all(axis=1)
adult_df_cleaned = adult_df[~outliers_adult]

# -------------------- Data Transformation (Normalization and Scaling) ------------------- # 1. Min-Max Scaling (Normalization)
min_max_scaler_adult = MinMaxScaler()
adult_df_scaled = pd.DataFrame(min_max_scaler_adult.fit_transform(adult_df_cleaned), colum

# 2. Standard Scaling (Z-Score Normalization) standard_scaler_adult =
StandardScaler()
adult_df_standardized = pd.DataFrame(standard_scaler_adult.fit_transform(adult_df_cleaned)

# Display results
print("\nProcessed Adult Income Data (after preprocessing):") print(adult_df_standardized.head())
```

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

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

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

Processed Adult Income Data (after preprocessing):
        age  workclass    fnlwgt  education  educational-num  marital-status  \
0 -0.995129   0.088484  0.351675  -2.397350        -1.197259        0.916138
1 -0.046942   0.088484 -0.945524   0.183660        -0.419335       -0.410397
2 -0.776316  -1.277432  1.394723  -0.848744         0.747550       -0.410397
3  0.390683   0.088484 -0.277844   1.216063        -0.030373       -0.410397
4 -1.505691  -2.643348 -0.815954   1.216063        -0.030373        0.916138

   occupation  relationship      race    gender  capital-gain  capital-loss  \
0    0.099824      0.971649 -1.971746   0.70422     -0.144804     -0.217127
1   -0.372938     -0.900852  0.392384   0.70422     -0.144804     -0.217127
2    1.045346     -0.900852  0.392384   0.70422     -0.144804     -0.217127
3    0.099824     -0.900852 -1.971746   0.70422      0.886874     -0.217127
4   -1.554840      0.971649  0.392384  -1.42001     -0.144804     -0.217127

   hours-per-week  native-country    income
0       -0.034087        0.289462 -0.560845
1        0.772930        0.289462 -0.560845
2       -0.034087        0.289462  1.783024
```

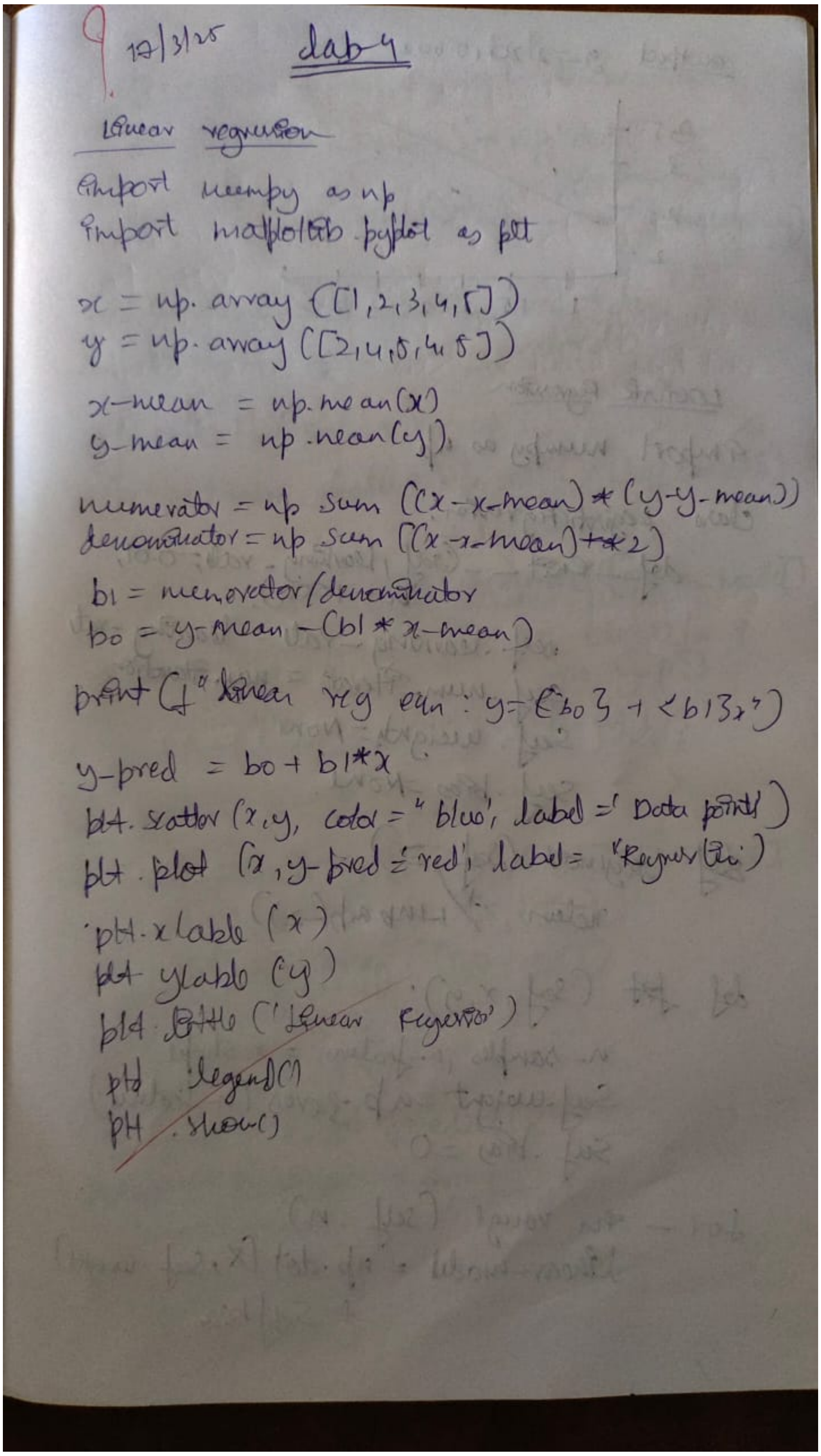
3	-0.034087	0.289462	1.783024
4	-0.841104	0.289462	-0.560845

Start coding or generate

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



9 12/3/25 Lab-4

Linear regression

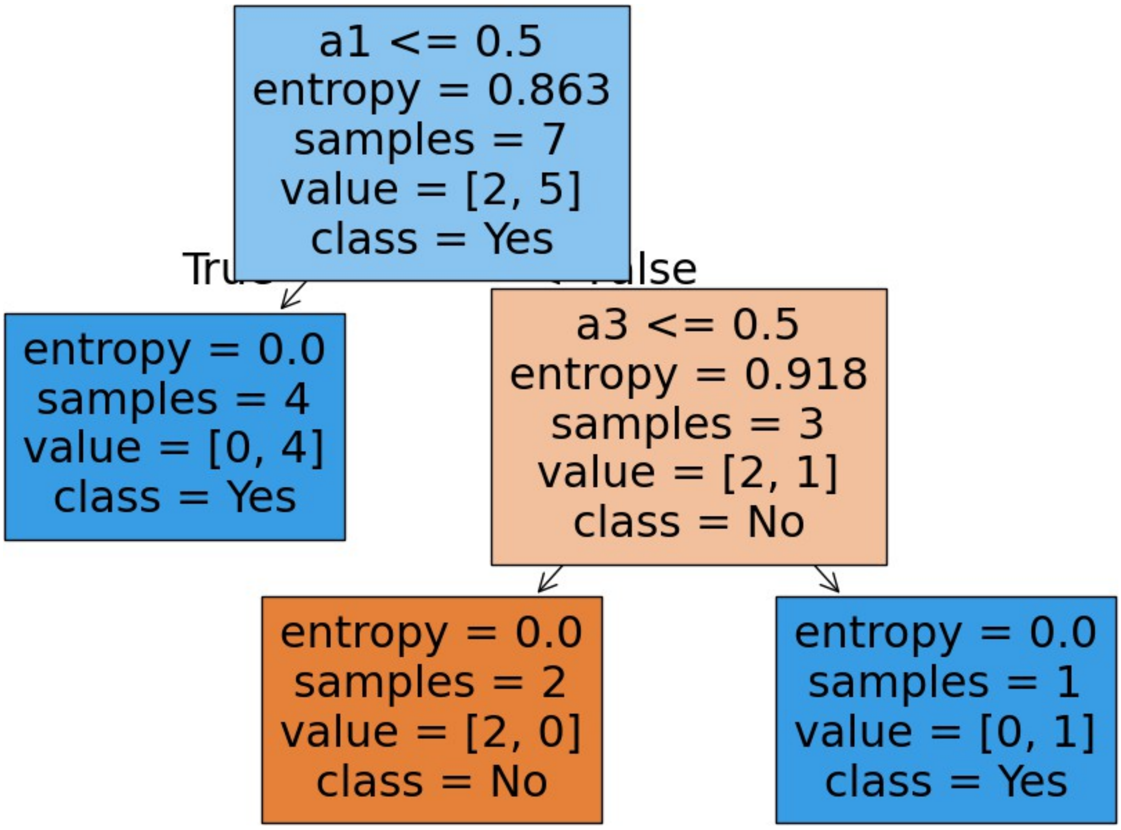
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([[1,2,3,4,5]])
y = np.array([[2,4,5,4,5]])

x-mean = np.mean(x)
y-mean = np.mean(y)

numerator = np sum ((x-x-mean) * (y-y-mean))
denominator = np sum ((x-x-mean)**2)
b1 = numerator/denominator
b0 = y-mean - (b1 * x-mean)

print(f"linear reg eqn: y={b0} + {b1}x")

y-pred = b0 + b1*x
plt.scatter(x,y, color="blue", label='Data points')
plt.plot (x,y-pred='red', label='Regression line')
plt.xlabel(x)
plt ylabel(y)
plt.title('Linear Regression')
plt legend()
plt .show()
```

```
# -*- coding: utf-8 -*- """Decision_Tree.ipynb

Automatically generated by Colab.

Original file is located at
        https://colab.research.google.com/drive/1RXDK8CR1doVCMHgkaXpJsNLAvzOIaXdd
"""

import pandas as pd
from sklearn.preprocessing import LabelEncoder from  sklearn.tree
import  DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Create the dataset data = {
        'a1': [True, True, False, False, False, True, True, True, False, False],
        'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],
        'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', '
        'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']
}

data

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

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

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

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

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

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

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred) print(f'Accuracy:
{accuracy:.2f}')
print(classification report(y test, y pred, target names=['No', 'Yes']))
```

```
# Optionally, visualize the decision tree from sklearn.tree
import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes']) plt.show()
```

Accuracy: 1.00

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




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

# Load the dataset
iris_data = pd.read_csv('iris (1).csv')

# Prepare the features and target variable
```

```
X=iris_data.drop('species',axis=1) y =
iris_data['species']

# 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 Decision Tree classifier dt_classifier =
DecisionTreeClassifier()
dt_classifier.fit(X_train,  y_train)

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

# Calculate accuracy and confusion matrix accuracy =
accuracy_score(y_test, y_pred)
conf_matrix  =  confusion_matrix(y_test,  y_pred)

# Display results
print("Accuracy Score:", accuracy)
print("Confusion  Matrix:\n",  conf_matrix)
```

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

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

# Load the dataset
drug_data = pd.read_csv('drug.csv')

# Prepare the features and target variable
X=drug_data.drop('Drug',axis=1) y =
drug_data['Drug']

# Convert categorical variables to dummy variables
X = pd.get_dummies(X, drop_first=True)

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

# Create and train the Decision Tree classifier dt_classifier =
DecisionTreeClassifier()
dt_classifier.fit(X_train,  y_train)

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

# Calculate accuracy and confusion matrix accuracy =
accuracy_score(y_test, y_pred)
conf_matrix  =  confusion_matrix(y_test,  y_pred)
```

```
# Display results
print("Accuracy Score:", accuracy)
print("Confusion  Matrix:\n",  conf_matrix)
```

```
Accuracy Score: 1.0
Confusion Matrix:
 [[ 6   0   0   0   0]
 [ 0   3   0   0   0]
 [ 0   0   5   0   0]
 [ 0   0   0  11   0]
 [ 0   0   0   0  15]]
```

```
import pandas as pd
from sklearn.model_selection import train_test_split from sklearn.tree
import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error import numpy as np

# Load the dataset
petrol_data = pd.read_csv('petrol_consumption.csv')

# Prepare the features and target variable
X = petrol_data.drop('Petrol_Consumption', axis=1) y =
petrol_data['Petrol_Consumption']

# 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 Regression Tree regressor =
DecisionTreeRegressor()
regressor.fit(X_train,   y_train)

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

# Calculate errors
mae = mean_absolute_error(y_test, y_pred) mse =
mean_squared_error(y_test, y_pred) rmse = np.sqrt(mse)

# Display results
print("Mean Absolute Error:", mae) print("Mean Squared
Error:", mse)
print("Root Mean Squared Error:", rmse)
```

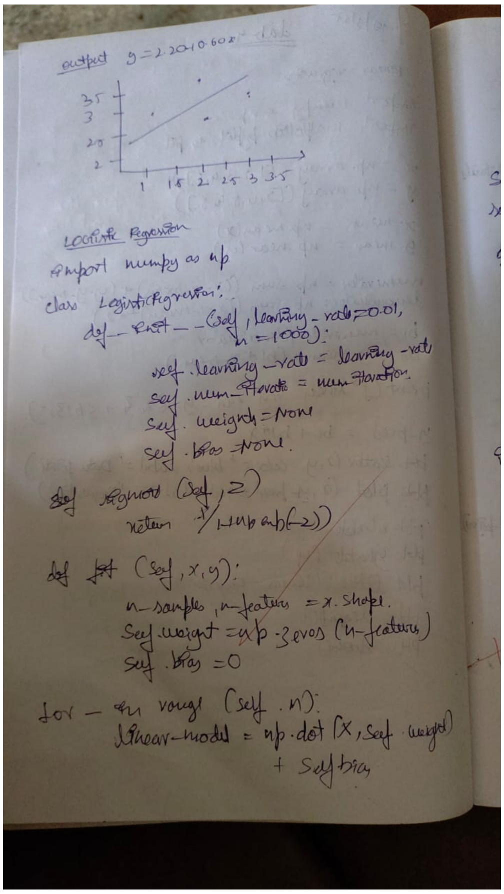
```
Mean Absolute Error: 91.7
Mean Squared Error: 16657.9
Root Mean Squared Error: 129.0654872535644
```

Start coding or generate with AI.

Program 4

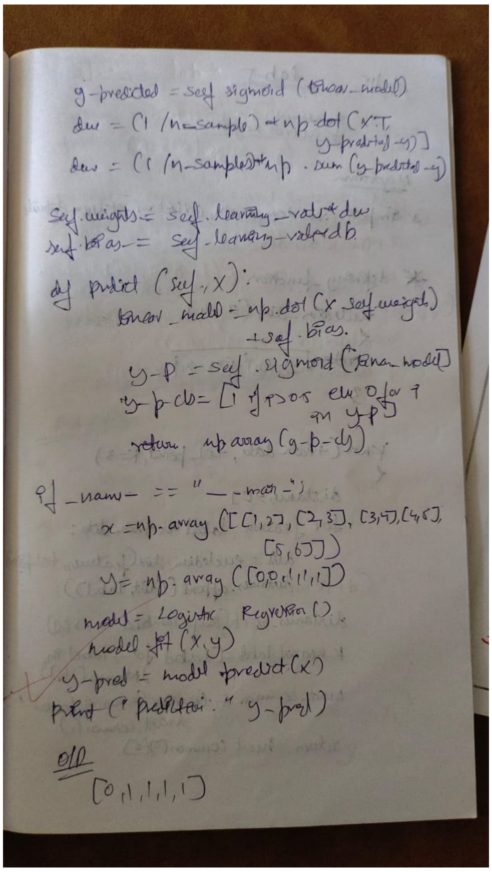
Build Logistic Regression Model for a given dataset

Screenshot



output y = 2.2010.60x

```
Logistic Regression
import numpy as np
class LogisticRegression:
    def __init__(self, learning_rate=0.01, n=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iteration
        self.weights = None
        self.bias = None
    def sigmoid(self, z)
        return 1/(1+np.exp(-z))
    def fit(self, x, y):
        n_samples, n_features = x.shape
        self.weight = np.zeros(n_features)
        self.bias = 0
    for _ in range(self.n):
        linear_model = np.dot(x, self.weight)
                       + self.bias
```



```
y_predicted = self.sigmoid(linear_model)
dw = (1/n_sample) * np.dot(x.T, (y_predicted - y))
dw = (1/n_samples) * np.sum(y_predicted - y)
self.weights -= self.learning_rate * dw
self.bias -= self.learning_rate * db
def predict(self, x):
    linear_model = np.dot(x, self.weight)
                   + self.bias
    y_p = self.sigmoid(linear_model)
    y_p_cls = [1 if p>0.5 else 0 for p in y_p]
    return np.array(y_p_cls)
if __name__ == "__main__":
    x = np.array([[1,2],[2,3],[3,4],[4,5],
                  [5,6]])
    y = np.array([0,0,1,1,1])
    model = LogisticRegression()
    model.fit(x,y)
    y_pred = model.predict(x)
    print("prediction:", y_pred)
O/P
[0,1,1,1,1]
```

Open in Colab

LINEAR REGERESSION

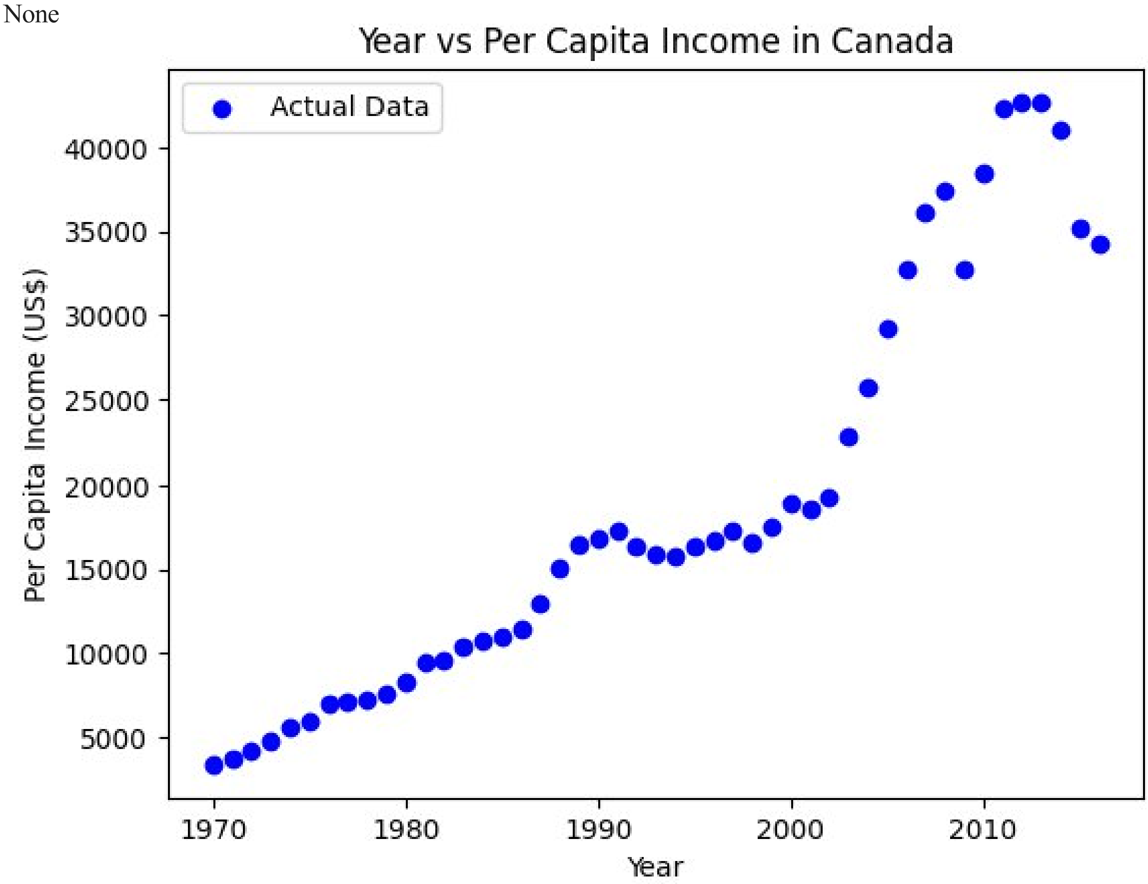
▸ linear_regression_housing_area_price.py

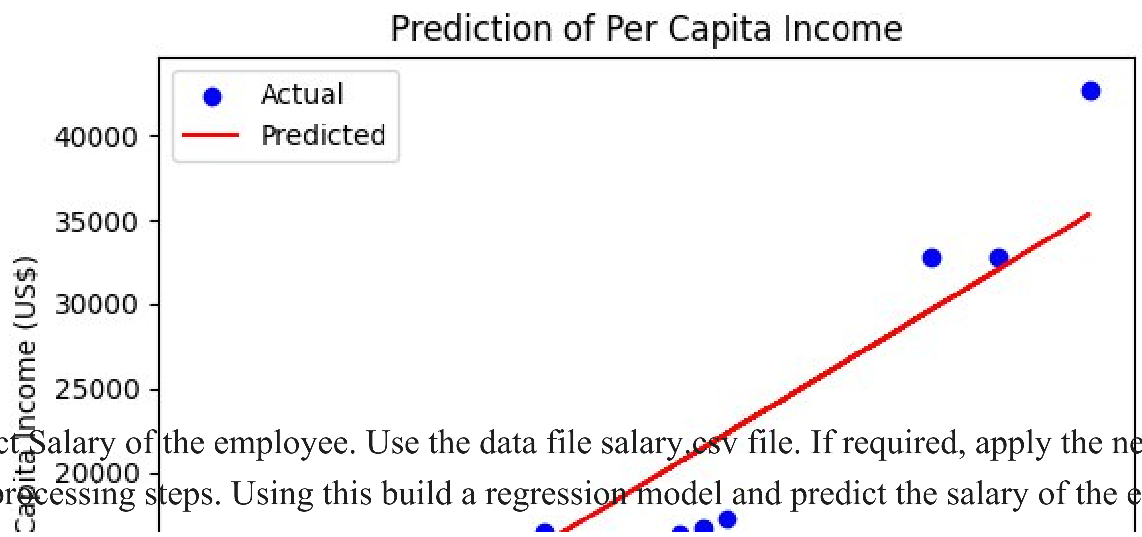
Predict canada's per capita income in year 2020. Use the data file canada_per_capita_income.csv file. If required, apply the necessary data processing steps. Using this build a regression model and predict the per capita income for canadian citizens in year 2020

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

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

# Analyze data distribution print(data.describe())
print(data.info())

# Distribution plot visualization
plt.scatter(data['year'], data['per capita income (US$)'], color='blue', label='Actual Data' plt.xlabel("Year")
plt.ylabel("Per Capita Income (US$)")
plt.title("Year vs Per Capita Income in Canada") plt.legend()
plt.show()

# Relationship between variables
X = data[['year']]
y = data['per capita income (US$)']

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

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

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

# Visualize prediction
plt.scatter(X_test,  y_test,  color='blue',  label='Actual')
```

```
plt.plot(X_test, y_pred, color='red', label='Predicted') plt.xlabel("Year")
plt.ylabel("Per Capita Income (US$)")
plt.title("Prediction of Per Capita Income") plt.legend()
plt.show()

# Check values of coefficient and intercept print(f"Coefficient: {model.coef_[0]}")
print(f"Intercept:   {model.intercept_}")

# Predict per capita income for 2020 y_2020 =
model.predict([[2020]])
print(f"Predicted per capita income in 2020: {y_2020[0]:.2f} US$")

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

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

```
              year   per  capita    income (US$)
count     47.000000                    47.000000
mean    1993.000000                 18920.137063
std       13.711309                 12034.679438
min     1970.000000                  3399.299037
25%     1981.500000                  9526.914515
50%     1993.000000                 16426.725480
75%     2004.500000                 27458.601420
max     2016.000000                 42676.468370
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47 entries, 0 to 46
Data columns (total 2 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   year                      47  non-null    int64
 1   per capita  income (US$)  47  non-null    float64
dtypes: float64(1), int64(1) memory usage:
884.0 bytes
None
```





Predict Salary of the employee. Use the data file salary.csv file. If required, apply the necessary data processing steps. Using this build a regression model and predict the salary of the employee with 12 years of experience.

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

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

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

# Analyze data distribution print(data.describe())
print(data.info())

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

# Relationship between variables
X = data[['YearsExperience']] y =
data['Salary']

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

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

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

# Visualize prediction
plt.scatter(X_test, y_test, color='blue', label='Actual') plt.plot(X_test, y_pred, color='red',
label='Predicted') plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Prediction of Salary") plt.legend()
plt.show()

# Check values of coefficient and intercept print(f"Coefficient:  {model.coef_[0]}")
print(f"Intercept:   {model.intercept_}")

# Predict salary for an employee with 12 years of experience
```

```
salary_12_years = model.predict([[12]])
print(f"Predicted salary for 12 years of experience: {salary_12_years[0]:.2f} US$")

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

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

```
       YearsExperience         Salary
count        28.000000      28.000000
mean          5.192857   75071.785714
std           2.821600   27128.441103
min           1.100000   37731.000000
25%           3.150000   56430.000000
50%           4.500000   65237.000000
75%           7.300000   99030.250000
max          10.500000  122391.000000
<class 'pandas.core.frame.DataFrame'> Index: 28
entries, 0 to 29
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   YearsExperience   28 non-null     float64
 1   Salary            28 non-null     int64
dtypes: float64(1), int64(1) memory usage:
672.0 bytes
None
```

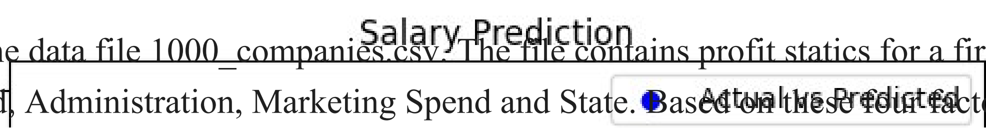
Considering the data file hiring.csv. The file contains hiring statics for a firm such as experience of candidate, his written test score and personal interview score. Based on these 3 factors, HR will decide the salary. Given this data, you need to build a Multiple Linear Regression model for HR department that can help them decide salaries for future candidates. Using this predict salaries for following candidates, 2 yr experience, 9 test score, 6 interview score 12 yr experience, 10 test score ,10 interview score

```
Coefficient:  9569.796810835274
Intercept:  25499.79528372343
Predicted salary for 12 years of experience: 140337.54 US$Mean Absolute Error
(MAE): 4519.16
Mean Squared Error (MSE): 27180506.80
```

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

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

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

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

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

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

# Analyze data distribution print(data.describe())
print(data.info())

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

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

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

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

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

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

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

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

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

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

```
       experience  test_score(out of 10)  interview_score(out of 10)  \
count    5.000000               5.000000                    5.000000
mean     5.600000               7.800000                    8.200000
std      3.577709               1.643168                    1.788854
min      2.000000               6.000000                    6.000000
25%      3.000000               7.000000                    7.000000
50%      5.000000               7.000000                    8.000000
75%      7.000000               9.000000                   10.000000
max     11.000000              10.000000                   10.000000

          salary($) 5.000000
count
mean   67400.000000
std     7987.490219
min    60000.000000
25%    62000.000000
50%    65000.000000
75%    70000.000000
max    80000.000000
<class 'pandas.core.frame.DataFrame'>
Index: 5 entries, 2 to 7
Data columns (total 4 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   experience                    5 non-null      float64
 1   test_score(out of 10)         5 non-null      float64
 2   interview_score(out of 10)    5 non-null      float64
 3   salary($)                     5 non-null      float64
dtypes: float64(4)
memory usage: 200.0 bytes None
```



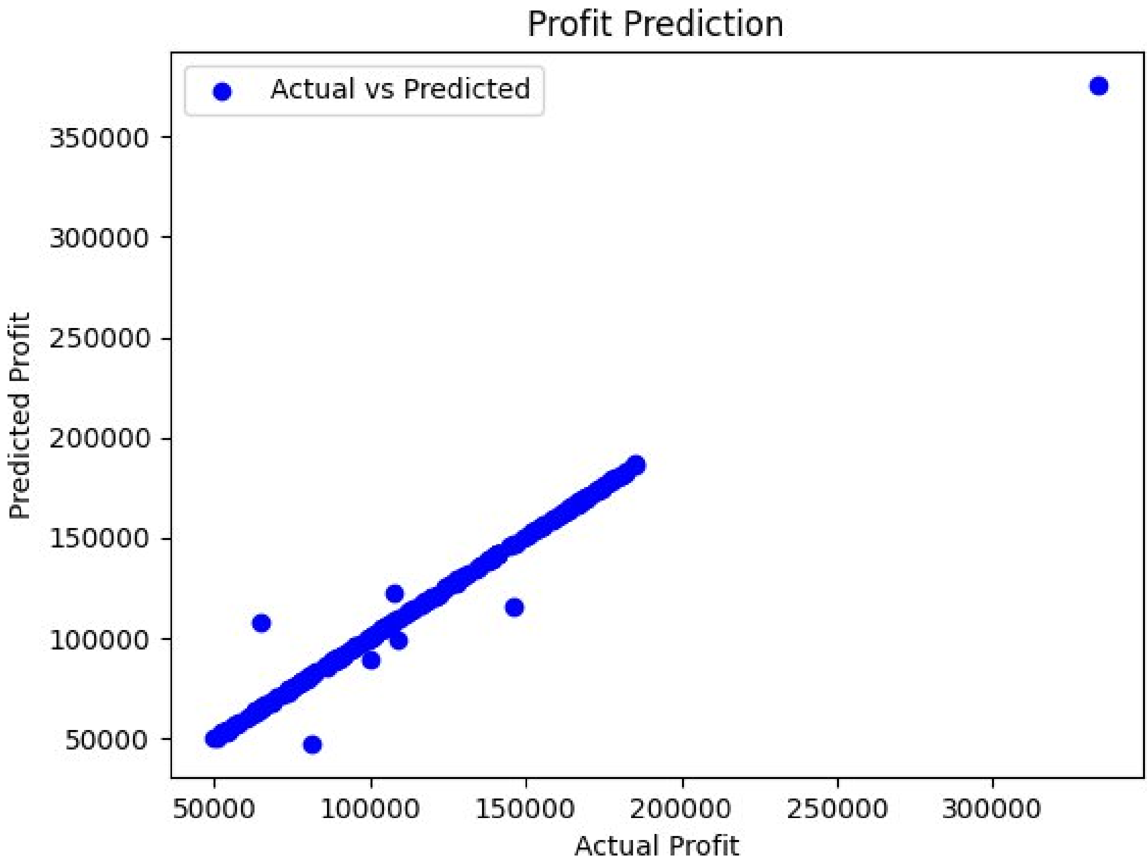
Considering the data file 1000_companies.csv. The file contains profit statics for a firm such as R&D Spend, Administration, Marketing Spend and State. Based on these four factors build a

Multiple Linear Regression model to predict the profit. Using this predict profit for following, 91694.48 68000 R&D Spend, 515841.3 Administration, 11931.24 Marketing Spend, Florida State

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

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

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

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

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

# Analyze data distribution print(data.describe())
print(data.info())

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

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

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

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

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

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

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

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

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

```
                R&D Spend   Administration   Marketing Spend          Profit  \
count    1000.000000     1000.000000       1000.000000     1000.000000
mean    81668.927200   122963.897612     226205.058419   119546.164656
std     46537.567891    12613.927535      91578.393542    42888.633848
min         0.000000    51283.140000          0.000000    14681.400000
25%     43084.500000   116640.684850     150969.584600    85943.198543
50%     79936.000000   122421.612150     224517.887350   117641.466300
75%    124565.500000   129139.118000     308189.808525   155577.107425
max    165349.200000   321652.140000     471784.100000   476485.430000

        State_Florida   State_New York
count     1000.000000      1000.000000
mean         0.322000         0.334000
std          0.467477         0.471876
min          0.000000         0.000000
25%          0.000000         0.000000
50%          0.000000         0.000000
75%          1.000000         1.000000
max          1.000000         1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        1000 non-null   float64
 1   Administration   1000 non-null   float64
 2   Marketing Spend  1000 non-null   float64
 3   Profit           1000 non-null   float64
 4   State_Florida    1000 non-null   float64
 5   State_New York   1000 non-null   float64
dtypes: float64(6)
memory usage: 47.0 KB None
```




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

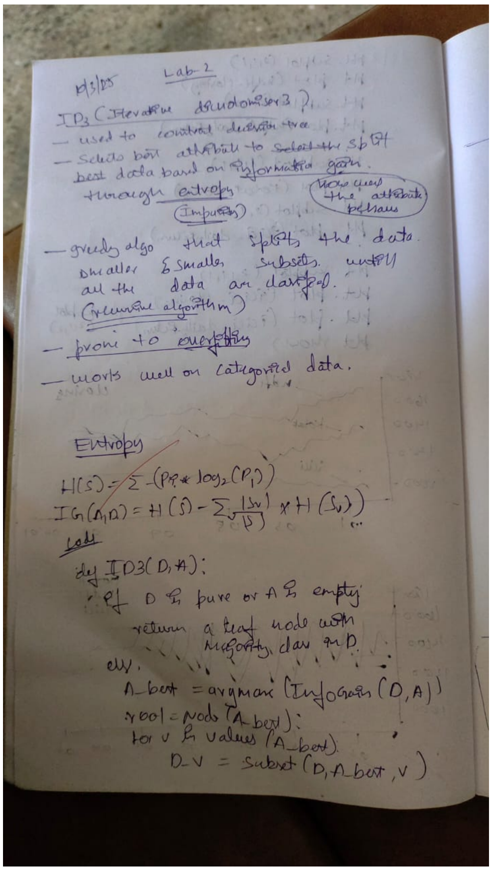
Intercept: -82439.15560711118

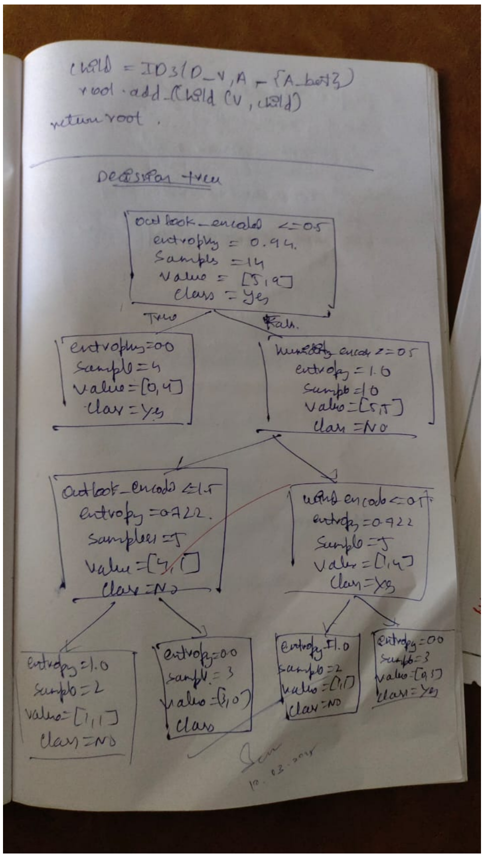
```
# -*- coding: utf-8 -*-
"""Linear-Regression-Housing_Area_Price.ipynb Automatically generated

by Colab.

Original file is located at
    https://colab.research.google.com/drive/1CAlZml-P6V2V1RIrodgMfF8L3Ux4V9FT
```

Program 5

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

Screenshot





Open in Colab

BINARY CLASSIFICATION

1. HR_COMMA-SEP

```
# Step 1: Import libraries import pandas as
pd
import numpy as np
import matplotlib.pyplot as plt import seaborn as
sns
from sklearn.model_selection import train_test_split from
sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Step 2: Load the dataset
df = pd.read_csv("HR_comma_sep.csv")

# Step 3: Basic exploratory data analysis (EDA) print(df.info())  # Check basic info
print(df.describe())  # Summary statistics print(df.head())  #
Preview first few rows

# Step 4: Visualize the impact of salary on retention plt.figure(figsize=(8, 6))
salary_retention = df.groupby(['salary', 'left']).size().unstack()
salary_retention.plot(kind='bar', stacked=True, color=['#1f77b4', '#ff7f0e'], figsize=(8, 6) plt.title('Impact of Salary on Employee
Retention')
plt.xlabel('Salary')
plt.ylabel('Number of Employees') plt.xticks(rotation=0)
plt.legend(title='Retention Status', labels=['Stayed', 'Left']) plt.tight_layout()
plt.show()

# Step 5: Visualize the correlation between department and retention plt.figure(figsize=(10, 6))
department_retention = df.groupby(['Department', 'left']).size().unstack()
department_retention.plot(kind='bar', stacked=True, color=['#1f77b4', '#ff7f0e'], figsize=(1 plt.title('Impact of Department on
Employee Retention')
plt.xlabel('Department')
plt.ylabel('Number of Employees') plt.xticks(rotation=45)
plt.legend(title='Retention Status', labels=['Stayed', 'Left']) plt.tight_layout()
plt.show()

# Step 6: Data Preprocessing
# Convert categorical columns into numerical variables
df['salary'] = df['salary'].map({'low': 0, 'medium': 1, 'high': 2})
df['Department'] = df['Department'].map({'sales': 0, 'technical': 1, 'support': 2, 'IT': 3,
```

```
# Features (independent variables) and target (dependent variable)
X = df[['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours', ' y = df['left']

# Step 7: Split 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)

# Step 8: Build logistic regression model model =
LogisticRegression(max_iter=1000)       model.fit(X_train,
y_train)

# Step 9: Make predictions on the test set y_pred =
model.predict(X_test)

# Step 10: Measure the accuracy of the model accuracy =
accuracy_score(y_test, y_pred)
print(f'Accuracy of the Logistic Regression model: {accuracy * 100:.2f}%')

# Step 11: Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Stayed', 'Left'], yticklabe plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout() plt.show()
```

```
<class 'pandas.core.frame.DataFrame'> RangeIndex:
14999 entries, 0 to 14998 Data columns (total 10
columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   Department             14999 non-null  object
 9   salary                 14999 non-null  object dtypes:
float64(2), int64(6), object(2)
memory usage: 1.1+ MB
None
       satisfaction_level  last_evaluation  number_project  \
count        14999.000000     14999.000000    14999.000000
mean             0.612834         0.716102        3.803054
std              0.248631         0.171169        1.232592
min              0.090000         0.360000        2.000000
25%              0.440000         0.560000        3.000000
50%              0.640000         0.720000        4.000000
75%              0.820000         0.870000        5.000000
max              1.000000         1.000000        7.000000

       average_montly_hours  time_spend_company  Work_accident          left  \
count          14999.000000        14999.000000   14999.000000  14999.000000
mean             201.050337            3.498233       0.144610      0.238083
std               49.943099            1.460136       0.351719      0.425924
min               96.000000            2.000000       0.000000      0.000000
25%              156.000000            3.000000       0.000000      0.000000
50%              200.000000            3.000000       0.000000      0.000000
75%              245.000000            4.000000       0.000000      0.000000
max              310.000000           10.000000       1.000000      1.000000

       promotion_last_5years
count           14999.000000
mean                0.021268
std                 0.144281
min                 0.000000
25%                 0.000000
50%                 0.000000
75%                 0.000000
max                 1.000000
   satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
2                0.11             0.88               7                   272
3                0.72             0.87               5                   223
4                0.37             0.52               2                   159

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

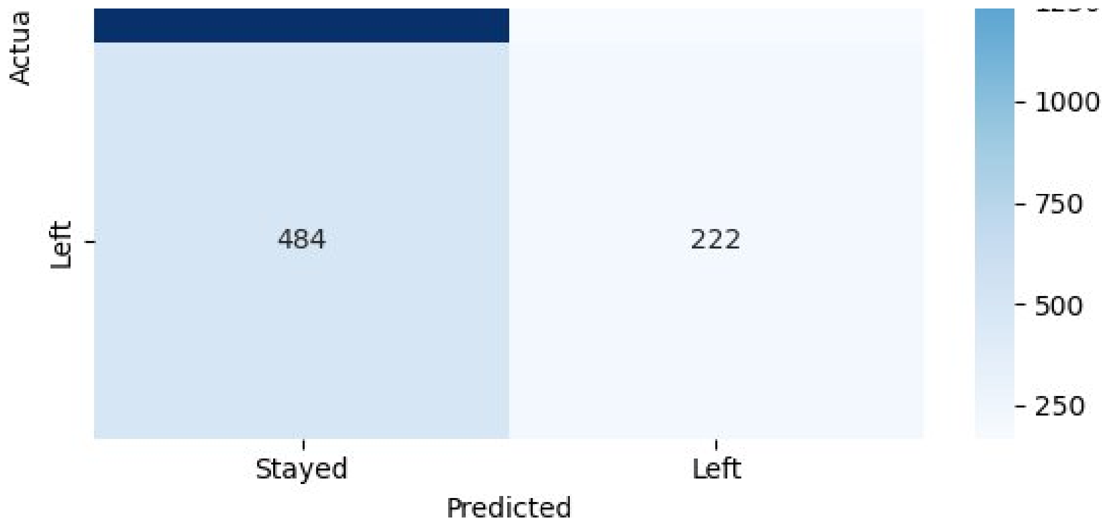
salary
0 low
MULTIPLE CLASSIFICATION
1 medium
2 medium
3 low
4 low

1. ZOO-DATA.CSV

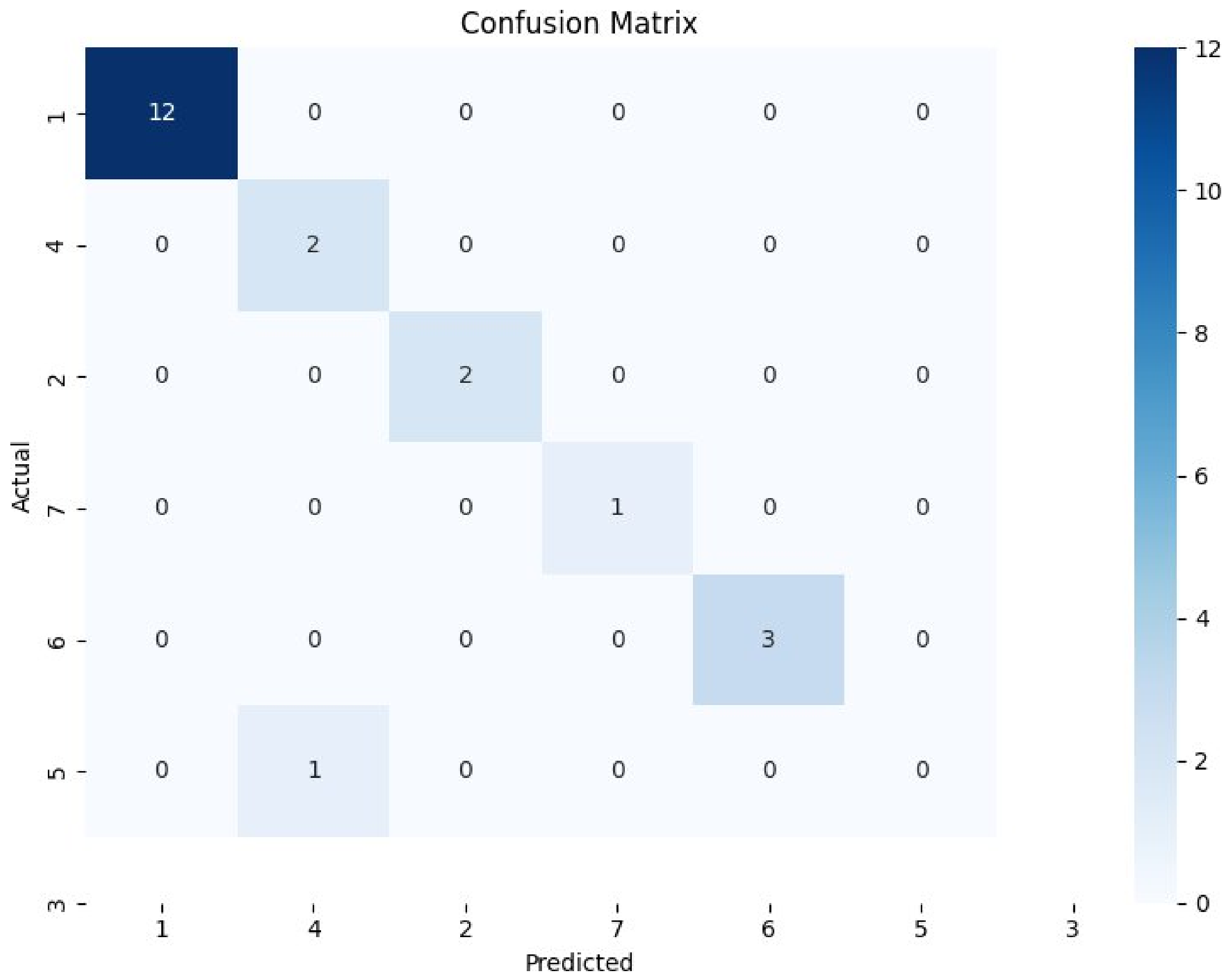
<Figure size 800x600 with 0 Axes>

```
# Step 1: Import necessary libraries import pandas as
pd
import numpy as np
import matplotlib.pyplot as plt import seaborn as
sns
from sklearn.model_selection import train_test_split from
sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Step 2: Load the datasets
zoo_data = pd.read_csv("zoo-data.csv")

# Step 3: Data Exploration and Preprocessing # Check for

missing values
print(zoo_data.isnull().sum())

# We can drop the 'animal_name' column as it is a non-numeric feature and won't help in pr zoo_data =
zoo_data.drop(columns=['animal_name'])

# Convert 'class_type' to numerical categories (assuming it's categorical) # If class_type is already numeric,
you can skip this step
class_type_mapping = {label: idx for idx, label in enumerate(zoo_data['class_type'].unique zoo_data['class_type'] =
zoo_data['class_type'].map(class_type_mapping)

# Check the class_type mapping print(class_type_mapping)

# Step 4: Split data into features (X) and target (y)
X = zoo_data.drop(columns=['class_type']) y =
zoo_data['class_type']

# Step 5: Split 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)

# Step 6: Build Logistic Regression Model model =
LogisticRegression(max_iter=1000)    model.fit(X_train,
y_train)

# Step 7: Make predictions on the test set y_pred =
model.predict(X_test)

# Step 8: Measure the accuracy of the model accuracy =
accuracy_score(y_test, y_pred)
print(f'Accuracy of the Logistic Regression model: {accuracy * 100:.2f}%')

# Step 9: Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

```
# Plot the confusion matrix using seaborn heatmap plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_type_mapping.keys(), plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout() plt.show()
```



```
animal_name    0
hair           0
feathers       0
eggs           0
milk           0
airborne       0
aquatic        0
predator       0
toothed        0
backbone       0
breathes       0
venomous       0
fins           0
legs           0
tail           0
domestic       0
catsize        0
class_type     0
dtype: int64
{np.int64(1): 0, np.int64(4): 1, np.int64(2): 2, np.int64(7): 3, np.int64(6): 4, np. Accuracy of the Logistic Regression mod
95.24%
```



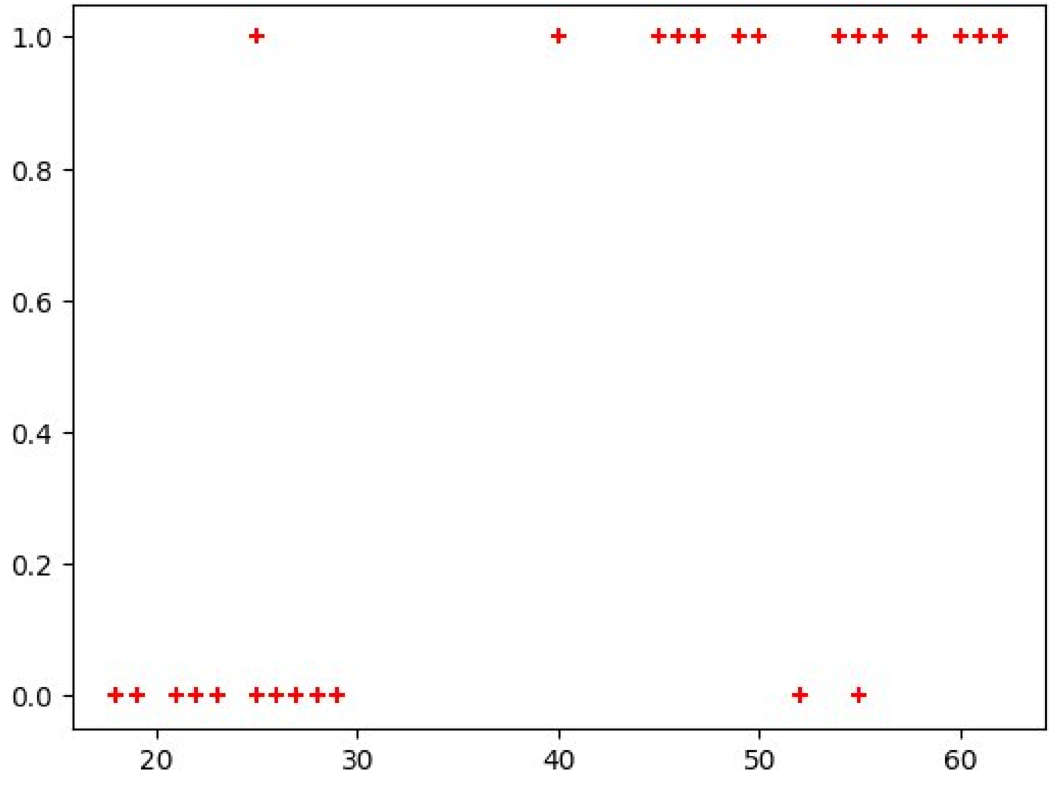
SAMPLE DATASETSETS

1. INSURANCE DATASET FOR BINARY CLASSIFICATION

```
# -*- coding: utf-8 -*-
"""LogisticRegression_Binary.ipynb Automatically

generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1M8PXdcmPsrQtqyVXpET3sgghAMr_MCg5
"""

# Commented out IPython magic to ensure Python compatibility. import pandas as pd
from matplotlib import pyplot as plt # %matplotlib
inline
#"%matplotlib inline" will make your plot outputs appear and be stored within the notebook

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

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

sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']],df.bought_insurance,train_
X_train.shape X_test

from sklearn.linear_model import LogisticRegression model =
LogisticRegression()

model.fit(X_train,  y_train)

X_test y_test

y_predicted = model.predict(X_test) y_predicted

model.score(X_test,y_test) model.predict_proba(X_test)

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

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

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

#Lets defined sigmoid function now and do the math with hand
```

```
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

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

age = 35
prediction_function(age)

"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: warnings.warn(
'0.37 is less than 0.5 which means person with 35 will not buy the insurance'
```



2. IRIS DATASET FOR MULTIPLECLASS CLASSIFICATION

```
# -*- coding: utf-8 -*-
"""LogisticRegression_Multiclass.ipynb Automatically

generated by Colab.

Original file is located at
        https://colab.research.google.com/drive/1anBybVXILenh0a_R4aM_ZemLrEqYWnJl
"""

#Import necessary libraries import pandas
as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

from sklearn.linear_model import LogisticRegression from sklearn.metrics
import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt

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

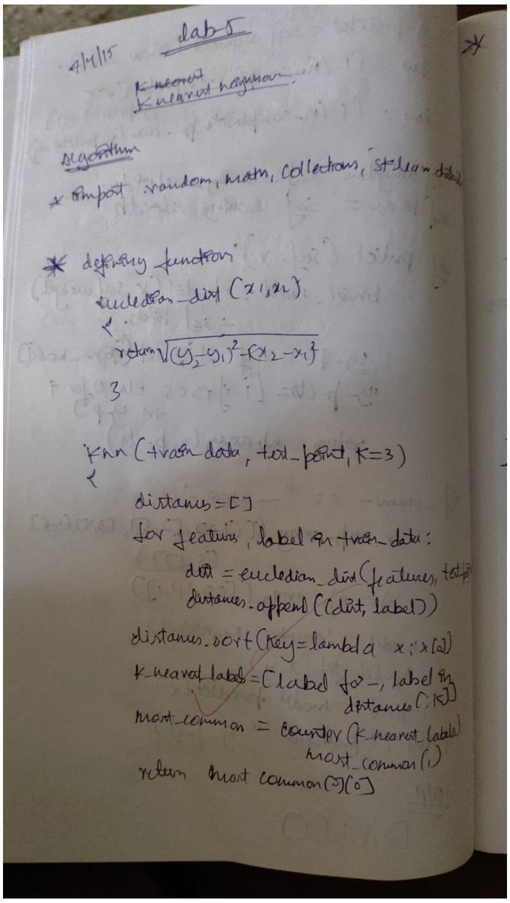
X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

Split the dataset into 80% training and 20% testing

Program 6

Build KNN Classification model for a given dataset

Screenshot



4/4/15

lab-5

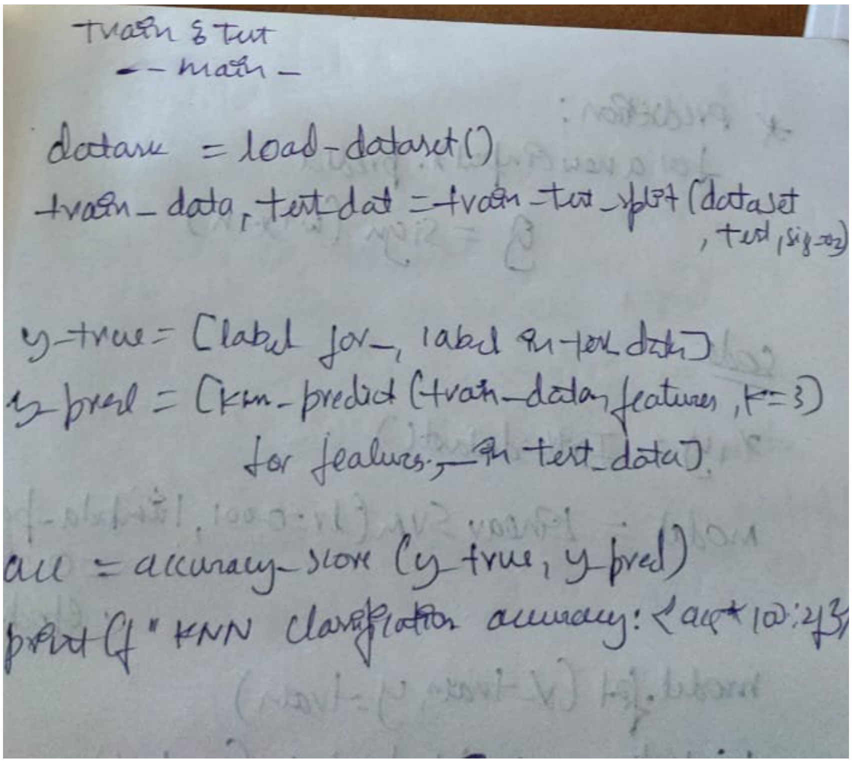
K nearest neighbour

algorithm

* import random, math, collections, sklearn datasets

* defining function
euclidean_dist (x1, x2)
{
return $\sqrt{(y_2-y_1)^2-(x_2-x_1)^2}$
}

knn (train data, test_point, k=3)
{
distances = []
for features, label in train_data:
dist = euclidean_dist(features, test_point)
distances.append((dist, label))
distances.sort(key=lambda x: x[0])
k_nearest_labels = [label for _, label in distances[:k]]
most_common = counter(k_nearest_labels).most_common(1)
return most_common[0][0]



Train & test
-- main --

dataset = load-dataset()
train_data, test_data = train_test_split(dataset, test_size=0.2)

y_true = [label for _, label in test_data]
y_pred = [knn_predict(train_data, features, k=3)
for features, _ in test_data]

acc = accuracy_score(y_true, y_pred)
print(f"KNN classification accuracy: {acc*100:.2f}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

           0       0.75      0.80      0.78       100
           1       0.58      0.52      0.55        54

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

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

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

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

print(f"Best k for Heart dataset: {best_k} with accuracy {best_score:.4f}") # Train KNN with best k
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
```

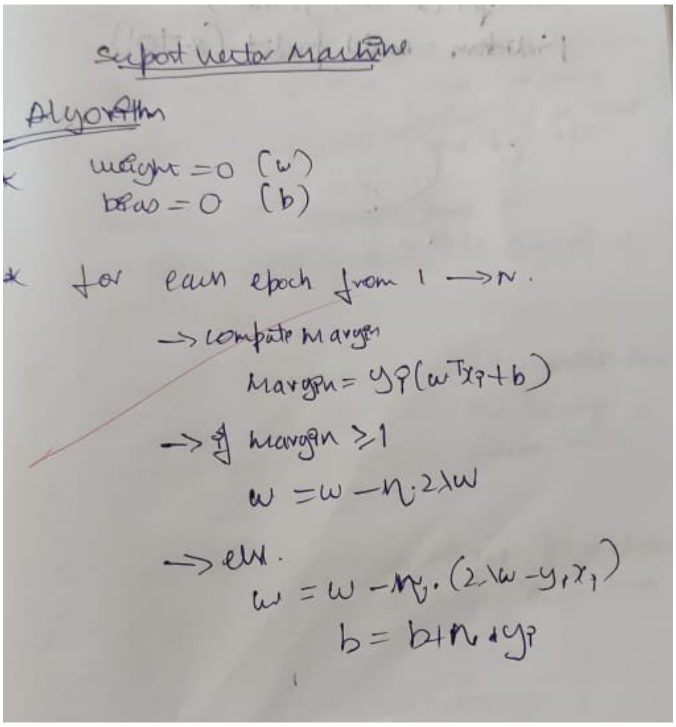
```
knn = KNeighborsClassifier(n_neighbors=best_k) knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

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

```
Best k for Heart dataset: 17 with accuracy 0.6721
--- Heart Dataset ---
Accuracy Score: 0.6721311475409836 Confusion
Matrix:
 [[17 11]
 [ 9 24]]
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.61      0.63        28
           1       0.69      0.73      0.71        33

    accuracy                           0.67        61
   macro avg       0.67      0.67      0.67        61
weighted avg       0.67      0.67      0.67        61
```

Program 7

Build Support vector machine model for a given dataset

Screenshot

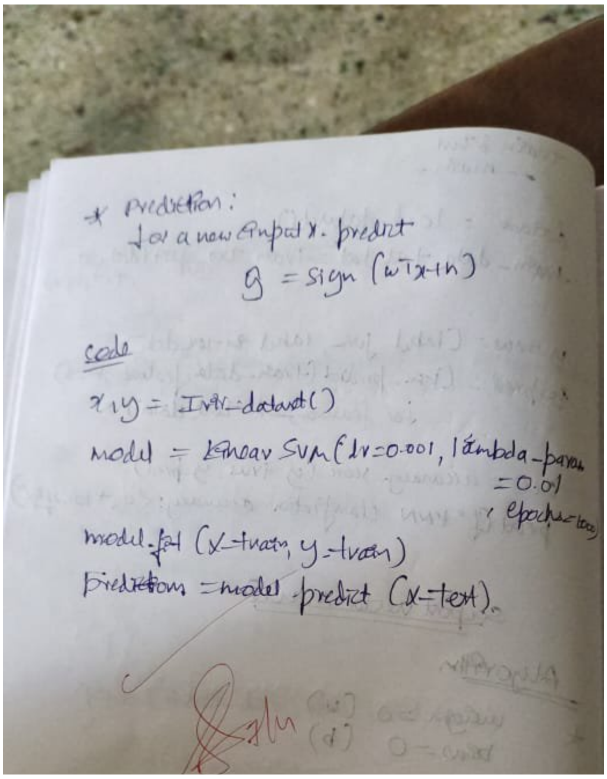


Suport Vector Machine

Algorithm

* weight = 0 (w)
bias = 0 (b)

* for each epoch from 1 → n.
→ compute Margin
Margin = $y_i(w^T x_i + b)$
→ if margin ≥ 1
$w = w - \eta \cdot 2\lambda w$
→ else.
$w = w - \eta \cdot (2\lambda w - y_i x_i)$
$b = b + \eta \cdot y_i$



* Prediction:
for a new input x. predict
$\hat{y} = \text{sign}(w^T x + b)$

code

x, y = Iris_dataset()
model = LinearSVM(lr=0.001, lambda_param=0.01, epochs=1000)
model.fit(x_train, y_train)
predictions = model.predict(x_test).

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

# Load dataset
iris_df = pd.read_csv("/content/sample_data/iris.csv")

# Split features and target
X = iris_df.drop(columns=["species"]) y =
iris_df["species"]

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

# SVM with RBF kernel
svm_rbf = SVC(kernel='rbf', random_state=42) svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)

# SVM with Linear kernel
svm_linear = SVC(kernel='linear', random_state=42) svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)

# Display results
print("RBF  Kernel  Accuracy:",  accuracy_score(y_test,  y_pred_rbf))
print("RBF  Confusion  Matrix:\n",  confusion_matrix(y_test,  y_pred_rbf))

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

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

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

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

# Load dataset
letter_df = pd.read_csv("/content/sample_data/letter-recognition.csv")
```

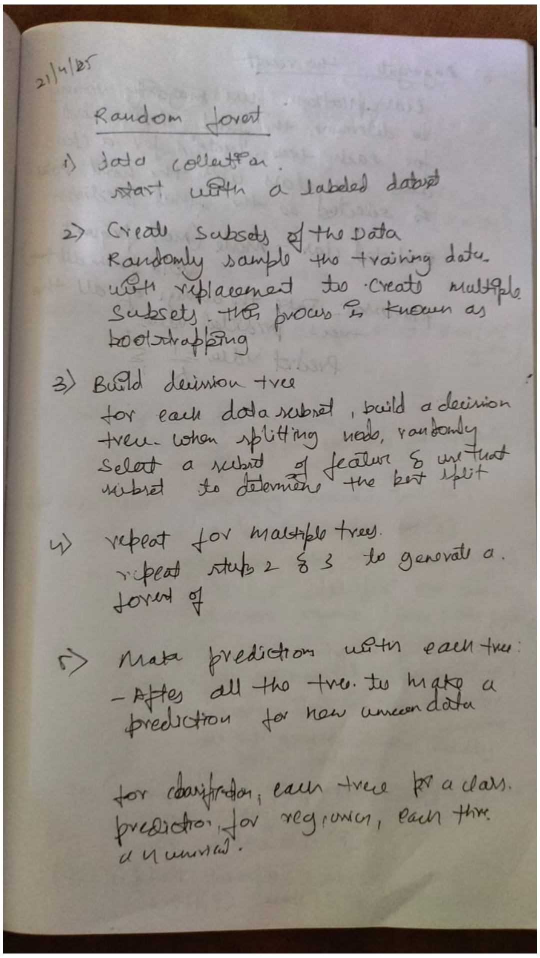
```
# Split features and target
X = letter_df.drop(columns=["letter"]) y =
letter_df["letter"]

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

# SVM Classifier
svm_clf = SVC(kernel='rbf', probability=True, random_state=42) svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)

# Accuracy and Confusion Matrix
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ROC Curve and AUC Score (One-vs-Rest Approach) lb =
LabelBinarizer()
y_test_binarized = lb.fit_transform(y_test) y_score =
svm_clf.predict_proba(X_test)

# Compute ROC curve and AUC for the first class
fpr, tpr, _ = roc_curve(y_test_binarized[:, 0], y_score[:, 0])
auc_score = roc_auc_score(y_test_binarized, y_score, multi_class='ovr')

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray') plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for Letter Recognition") plt.legend()
plt.show()
```

```
Accuracy: 0.9305
Confusion Matrix:
[[144   0   1   0   0   0   1   0   0   0   1   0   1   0   0   0   0   0
    0   0   1   0   0   0   0   0]
 [  0 143   0   5   0   1   0   0   0   0   0   0   0   0   0   0   0   4
    0   0   0   0   0   0   0   0]
 [  0   0 123   0   2   0   3   1   0   0   1   0   0   0   4   0   0   2
    0   0   1   0   0   0   0   0]
 [  0   1   0 153   0   0   0   2   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [  0   3   1   0 130   0   5   0   0   0   0   0   0   0   0   0   1   0
    0   0   0   0   0   0   0   1]
 [  0   2   0   0   1 134   0   0   1   0   0   0   0   0   0   0   0   0
    1   1   0   0   0   0   0   0]
 [  1   0   1   4   0   0 149   0   0   0   2   0   0   0   0   0   0   2
    0   0   0   1   0   0   0   0]
 [  0   4   0   8   0   0   0 106   0   0   5   0   0   1   2   1   1  13
    0   0   2   0   0   0   1   0]
 [  0   0   0   1   0   2   0   0 134   7   0   0   0   0   0   0   0   0
    0   0   0   0   0   2   0   0]
 [  0   0   0   1   1   1   0   0   3 139   0   0   0   0   1   0   0   0
    3   0   0   0   0   0   0   0]
 [  0   0   0   2   0   0   0   1   0   0 112   0   0   0   0   0   0  12
    0   0   0   0   0   3   0   0]
 [  0   0   1   0   4   0   3   0   0   0   0 142   0   0   1   0   1   1
    2   0   0   0   0   0   0   0]
 [  0   2   0   0   0   0   0   0   0   0   0   0 164   0   0   0   0   2
    0   0   0   0   0   0   0   0]
 [  1   0   0   3   0   0   0   0   0   0   0   0   0 139   5   0   0   2
    0   0   0   0   1   0   0   0]
 [  0   0   0   4   0   0   0   0   0   0   0   0   1   0 134   0   2   0
    0   0   1   0   3   0   0   0]
 [  0   1   0   0   0  14   8   0   0   0   0   0   0   0   0 148   0   0
    0   0   0   0   0   0   2   0]
 [  0   3   0   1   3   0   0   0   0   0   0   0   0   0   0   0 159   0
    0   0   0   0   0   0   0   0]
 [  0   5   0   2   0   0   0   0   0   0   2   1   0   2   0   0   0 148
    0   0   0   0   0   0   0   0]
 [  0   2   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0
  167   0   0   0   0   0   0   1]
 [  0   1   0   0   1   0   0   0   0   0   2   0   0   0   0   0   0   1
    0 153   1   0   0   2   1   1]
 [  0   0   0   0   0   0   0   0   0   0   2   0   4   0   2   0   0   0
    0   0 170   2   3   0   0   0]
 [  1   5   0   0   0   1   0   0   0   0   0   0   1   0   0   0   0   1
    0   0   0 147   2   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   3   0 145   0   0   0]
 [  0   1   0   2   0   0   0   0   0   0   1   0   0   0   0   0   0   0
    0   0   0   0   0 150   0   0]
 [  1   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   1   0   1   0   0 164   0]
 [  1   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   1
    4   0   0   0   0   0   0 125]]
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot



21/4/25

Random forest

1) data collection.
start with a labeled dataset

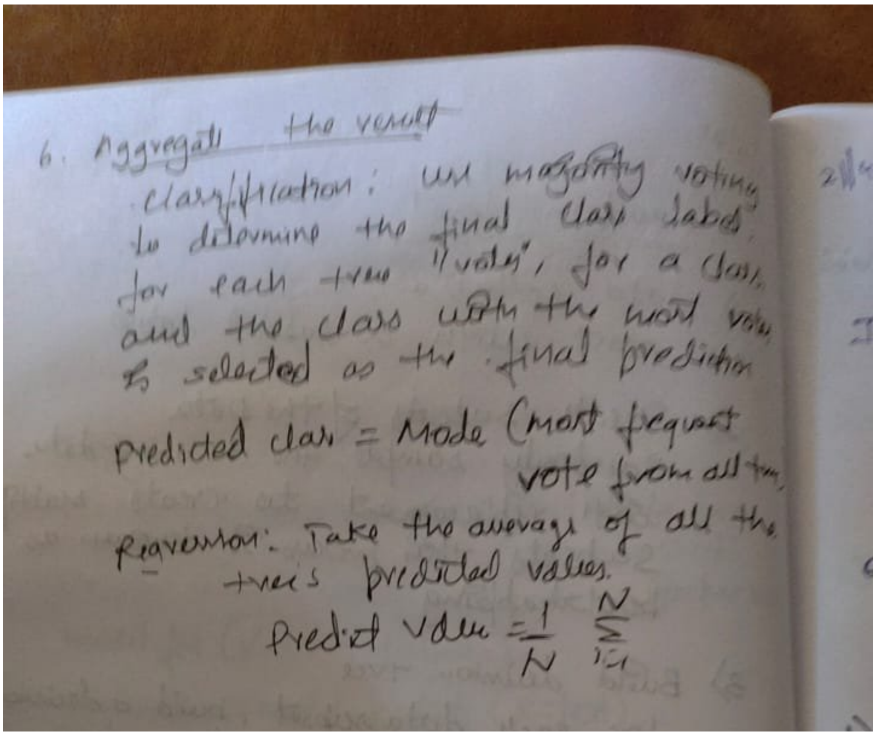
2) Create subsets of the Data
Randomly sample the training data
with replacement to create multiple
subsets. this process is known as
bootstrapping

3) Build decision tree
for each data subset, build a decision
tree. when splitting node, randomly
select a subset of feature & use that
subset to determine the best split

4) repeat for multiple trees.
repeat steps 2 & 3 to generate a
forest of

5) Make prediction with each tree:
- After all the tree. to make a
prediction for new unseen data

for classification, each tree [illegible] a class.
prediction, for regression, each tree
a numerical.



6. Aggregate the result
classification: use majority voting
to determine the final class label.
for each tree "votes", for a class,
and the class with the most votes
is selected as the final prediction

predicted class = Mode (most frequent
vote from all trees)

Regression: Take the average of all the
tree's predicted values.

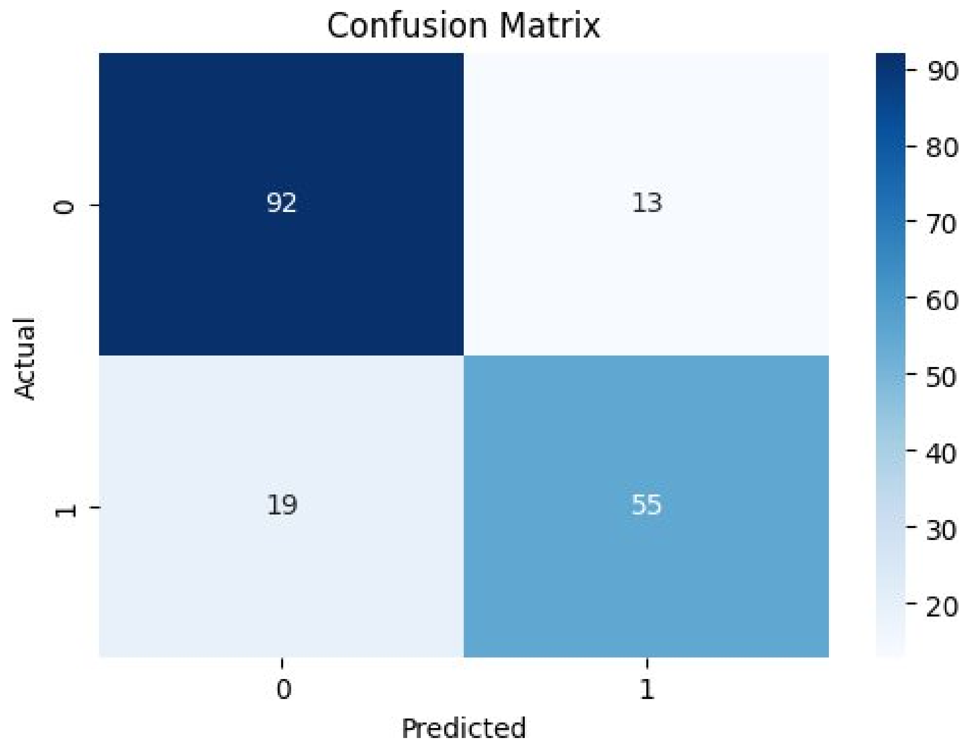
Predict value $= \frac{1}{N} \sum_{i=1}^{N}$

Open in Colab

```
import pandas as pd import
numpy as np
from sklearn.model_selection import train_test_split from sklearn.ensemble
import RandomForestClassifier from sklearn.preprocessing import
LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('train.csv')

# Data preprocessing
# Handle missing values
data['Age'] = data['Age'].fillna(data['Age'].median())
data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode()[0])

# Encode categorical variables le =
LabelEncoder()
data['Sex'] = le.fit_transform(data['Sex'])
data['Embarked'] = le.fit_transform(data['Embarked'])

# Select features and target
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X = data[features]
y = data['Survived']

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

# Build and train Random Forest model
rf=RandomForestClassifier(random_state=42) rf.fit(X_train, y_train)

# Make predictions
y_pred = rf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred) print(f"Accuracy
Score: {accuracy:.4f}")

# Confusion matrix
cm=confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Visualize confusion matrix plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues') plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

plt.show()

Accuracy Score: 0.8212 Confusion Matrix:
[[92 13]
[19 55]]




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

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

# Prepare features and target
X = iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] y = iris['species']

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

# Random Forest with default n_estimators=10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42) rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
accuracy_default = accuracy_score(y_test, y_pred_default)
print(f"Accuracy with n_estimators=10: {accuracy_default:.4f}")

# Fine-tune n_estimators
estimators = [10, 50, 100, 200, 500] scores = []
```

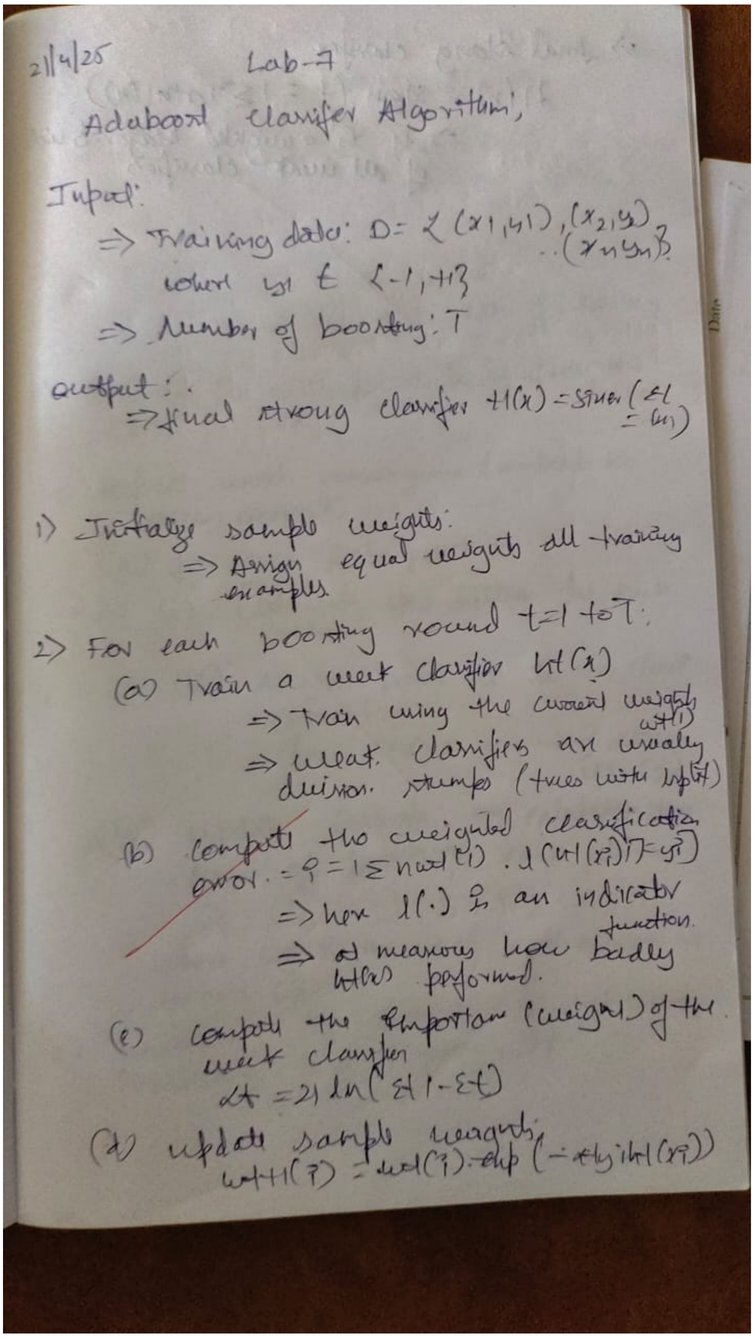
```
for n in estimators:
    rf = RandomForestClassifier(n_estimators=n, random_state=42) rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
    print(f"Accuracy with n_estimators={n}: {score:.4f}")

# Find the best n_estimators
best_n = estimators[scores.index(max(scores))] best_score = max(scores)
print(f"\nBest n_estimators: {best_n}")
print(f"Best Accuracy Score: {best_score:.4f}")
```

```
Accuracy with n_estimators=10: 1.0000 Accuracy
with n_estimators=10: 1.0000 Accuracy with
n_estimators=50: 1.0000 Accuracy with
n_estimators=100: 1.0000 Accuracy with
n_estimators=200: 1.0000 Accuracy with
n_estimators=500: 1.0000

Best n_estimators: 10
Best Accuracy Score: 1.0000
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot



21/4/25 Lab-7

Adaboost classifier Algorithm;

Input:

=> Training data: D = {(x1, y1), (x2, y2) .. (xn yn)}
where yi ∈ {-1, +1}

=> Number of boosting: T

output:

=> final strong classifier H(x) = sign(Σ ...)

1) Initialize sample weights:
=> Assign equal weights all training examples.

2) For each boosting round t=1 to T:

(a) Train a weak classifier ht(x)
=> Train using the current weights wt(i)
=> weak classifiers are usually decision stumps (trees with depth 1)

(b) compute the weighted classification error: ε = 1 Σ wt(i) . I (ht(xi) ≠ yi)
=> here I(.) is an indicator function.
=> ε measures how badly ht(x) performed.

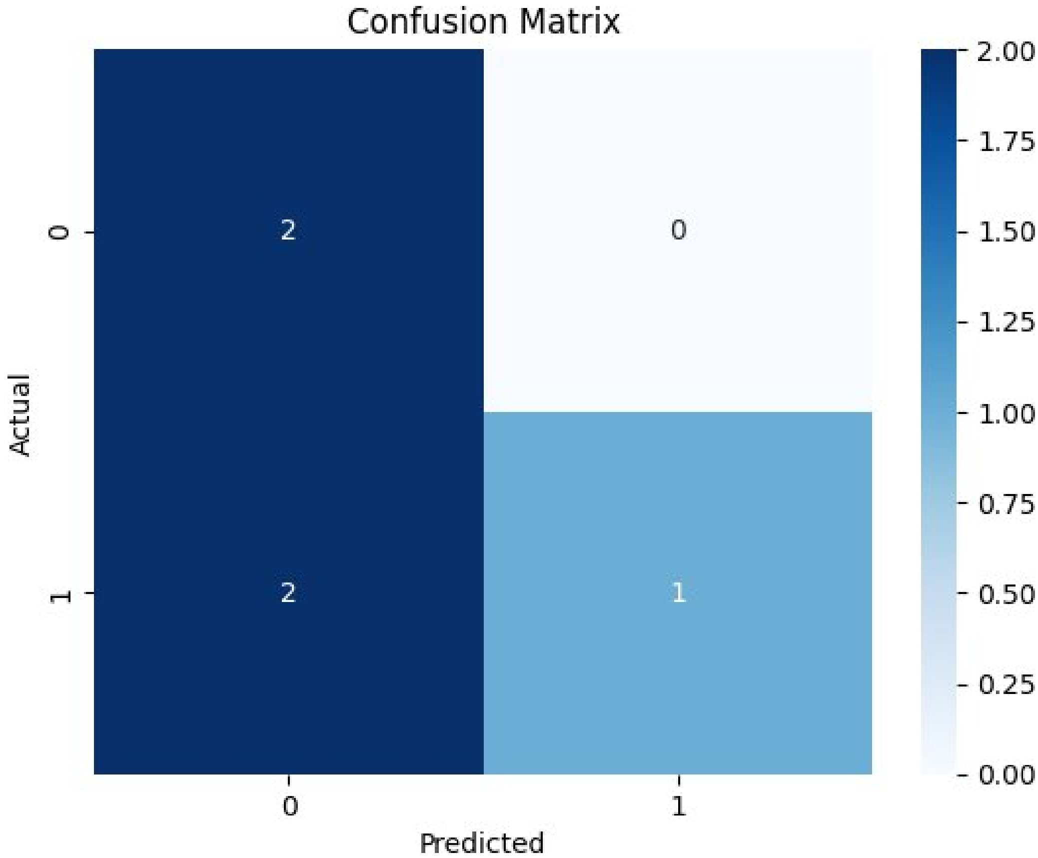
(c) compute the importance (weight) of the weak classifier
αt = 1/2 ln(εt / 1 - εt)

(d) update sample weights;
wt+1(i) = wt(i) . exp(-αt yi ht(xi))

Open in Colab

```
import pandas as pd import
numpy as np
from sklearn.model_selection import train_test_split from
sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
income_df = pd.read_csv('income.csv')

# Preprocess: Convert income to binary classes (above/below median) median_income =
income_df['Income($)'].median()
income_df['Income_Class'] = np.where(income_df['Income($)'] >= median_income, 1, 0)

# Features and target
X = income_df[['Age']]  # Using Age as feature y =
income_df['Income_Class']

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

# Initialize and train AdaBoost model
estimator = DecisionTreeClassifier(max_depth=1)
ada_model = AdaBoostClassifier(estimator=estimator, n_estimators=50, random_state=42)  ada_model.fit(X_train, y_train)

# Predict on test data
y_pred = ada_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred) print(f"Accuracy
Score: {accuracy:.2f}")

# Confusion matrix
cm=confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Visualize confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues') plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix') plt.show()
```

```
Accuracy Score: 0.60
Confusion Matrix:
[[2 0]
 [2 1]]
```




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

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

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

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

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

# Function to train and evaluate AdaBoost
def evaluate_adaboost(estimator, n_estimators, learning_rate, estimator_name): model = AdaBoostClassifier(
        estimator=estimator,
```

```
            n_estimators=n_estimators,
            learning_rate=learning_rate, random_state=42
        )
        model.fit(X_train,  y_train)
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        print(f"{estimator_name} with n_estimators={n_estimators}, learning_rate={learning_rat  return accuracy

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

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

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

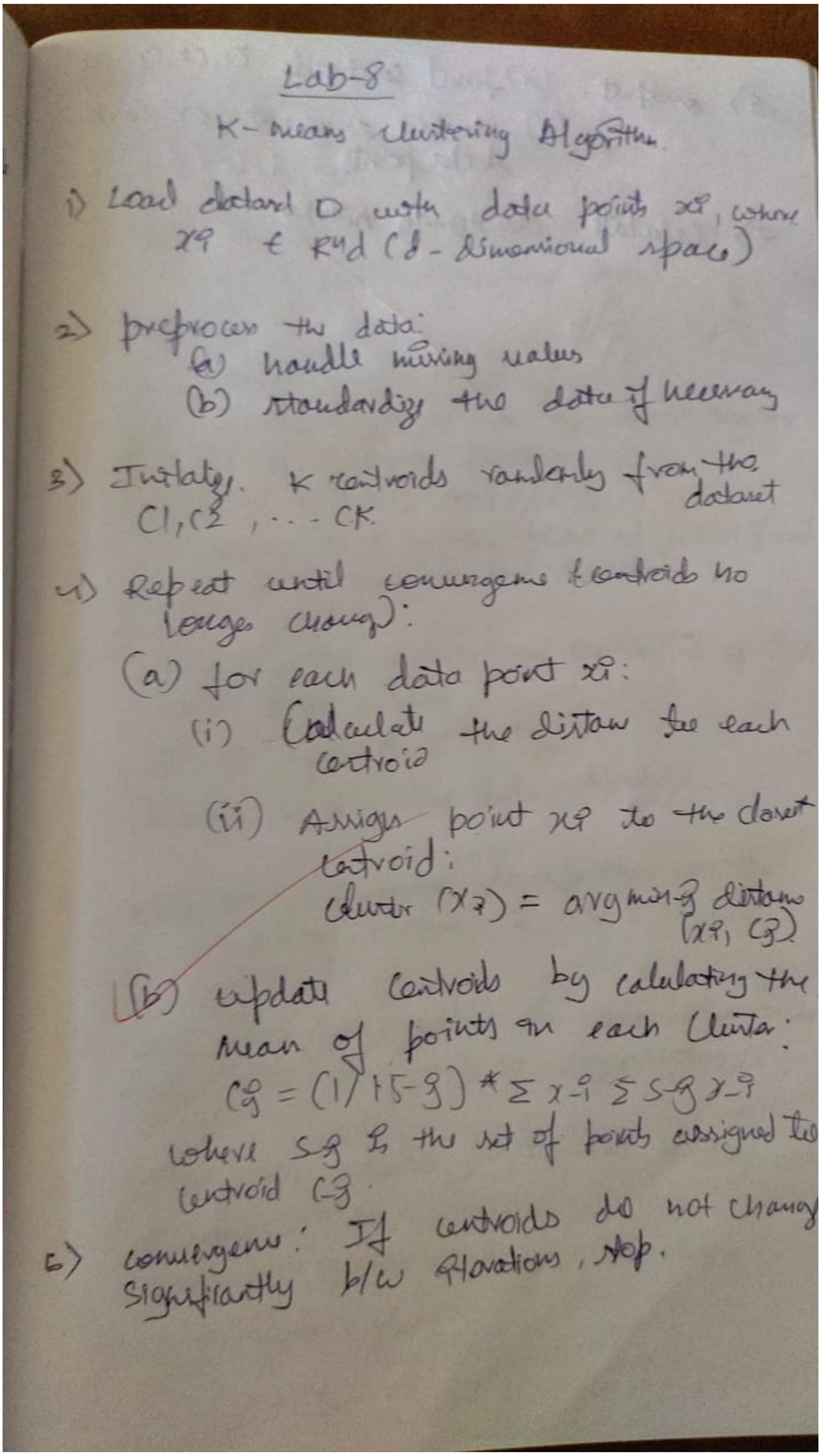
```
AdaBoost with Decision Tree:
Decision Tree with n_estimators=10, learning_rate=0.1: Accuracy = 0.967
Decision Tree with n_estimators=10, learning_rate=0.5: Accuracy = 1.000
Decision Tree with n_estimators=10, learning_rate=1.0: Accuracy = 1.000
Decision Tree with n_estimators=50, learning_rate=0.1: Accuracy = 1.000
Decision Tree with n_estimators=50, learning_rate=0.5: Accuracy = 0.967
Decision Tree with n_estimators=50, learning_rate=1.0: Accuracy = 0.933
Decision Tree with n_estimators=100, learning_rate=0.1: Accuracy = 1.000
Decision Tree with n_estimators=100, learning_rate=0.5: Accuracy = 1.000
Decision Tree with n_estimators=100, learning_rate=1.0: Accuracy = 0.933

AdaBoost with Logistic Regression:
Logistic Regression with n_estimators=10, learning_rate=0.1: Accuracy = 1.000 Logistic Regression with n_estimators=10, learning_rate=0.5: Accuracy = 0.967 Logistic Regression with n_estimators=10, learning_rate=1.0: Accuracy = 0.933 Logistic Regression with n_estimators=50, learning_rate=0.1: Accuracy = 1.000 Logistic Regression with n_estimators=50, learning_rate=0.5: Accuracy = 1.000 Logistic Regression with n_estimators=50, learning_rate=1.0: Accuracy = 0.933 Logistic Regression with n_estimators=100, learning_rate=0.1: Accuracy = 1.000 Logistic Regression with n_estimators=100, learning_rate=0.5: Accuracy = 1.000 Logistic Regression with n_estimators=100, learning_rate=1.0: Accuracy = 0.933
```

Start coding or generate with AI.

Program 10

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

Screenshot



Lab-8

K-Means Clustering Algorithm.

1) Load dataset D with data points xi, where xi ∈ R^d (d-Dimensional space)

2) preprocess the data:
   (a) handle missing values
   (b) standardize the data if necessary

3) Initialize. K centroids randomly from the dataset
   C1, C2, ... CK.

4) Repeat until convergence (centroids no longer change):
   (a) for each data point xi:
      (i) Calculate the distance to each centroid
      (ii) Assign point xi to the closest centroid:
      cluster (xi) = argmin-j distance (xi, Cj)
   (b) update centroids by calculating the mean of points in each Cluster:
   Cj = (1/|S-j|) * Σ x-i Σ S-j x-i
   where S-j is the set of points assigned to centroid Cj.

5) convergence: If centroids do not change significantly b/w iterations, stop.

Open in Colab

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits from
sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression from sklearn.metrics
import accuracy_score

# Step 1: Load Digits dataset digits =
load_digits()
X = digits.data   # Features y =
digits.target  # Labels

# Step 2: Split the data into training and testing sets (80% - 20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Scale the data scaler =
StandardScaler()
X_train_scaled = scaler.fit_transform(X_train) X_test_scaled =
scaler.transform(X_test)

# Step 4: Apply PCA with n_components = 2 pca =
PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled) X_test_pca =
pca.transform(X_test_scaled)

# Step 5: Train Logistic Regression on the PCA-transformed data logreg =
LogisticRegression(max_iter=10000)
logreg.fit(X_train_pca,  y_train)

# Step 6: Make predictions on the test set y_pred =
logreg.predict(X_test_pca)

# Step 7: Evaluate accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy using PCA with 2 components: {accuracy:.4f}')
```

```
Accuracy using PCA with 2 components: 0.5167
```

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

```
from sklearn.decomposition import PCA from
sklearn.pipeline import Pipeline

# Step 1: Load the dataset
# Assuming the dataset is stored in 'heart.csv' df =
pd.read_csv('heart.csv')

# Step 2: Remove outliers using Z-score
numeric_cols = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak'] z_scores =
stats.zscore(df[numeric_cols])
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
df_clean = df[filtered_entries].reset_index(drop=True)

# Step 3: Encode categorical variables and apply scaling # Identify categorical
and numerical columns
categorical_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope'] numeric_cols = ['Age', 'RestingBP',
'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak']

# Create preprocessing steps
preprocessor = ColumnTransformer( transformers=[
        ('cat', OneHotEncoder(drop='first', sparse_output=False), categorical_cols), ('num', StandardScaler(),
        numeric_cols)
    ])

# Apply preprocessing
X = df_clean.drop('HeartDisease', axis=1) y =
df_clean['HeartDisease']
X_processed = preprocessor.fit_transform(X)

# Get feature names after one-hot encoding
cat_encoded_cols = preprocessor.named_transformers_['cat'].get_feature_names_out(categoric  feature_names =
list(cat_encoded_cols) + numeric_cols

# Convert processed features back to DataFrame for clarity
X_processed_df = pd.DataFrame(X_processed, columns=feature_names)

# Step 4: Build classification models and evaluate accuracy # Split data into training
and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_processed_df, y, test_size=0.2, rand

# Define models models =
{
    'SVM': SVC(kernel='rbf', random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42), 'Random  Forest':
    RandomForestClassifier(n_estimators=100, random_state=42)
}

# Train and evaluate each model accuracies = {}
for name, model in models.items(): model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred) accuracies[name] = accuracy
```

```
        print(f"{name} Accuracy: {accuracy:.4f}")

# Find the best model
best_model_name=max(accuracies, key=accuracies.get)
print(f"\nBest Model: {best_model_name} with Accuracy: {accuracies[best_model_name]:.4f}")

# Step 5: Apply PCA and evaluate impact on model accuracy # Determine number
of components to retain 95% variance
pca = PCA()
pca.fit(X_processed_df)
cumulative_variance = np.cumsum(pca.explained_variance_ratio_) n_components =
np.argmax(cumulative_variance >= 0.95) + 1
print(f"\nNumber of PCA components to retain 95% variance: {n_components}")

# Create pipeline with PCA and the best model best_model =
models[best_model_name]
pipeline = Pipeline([
        ('pca', PCA(n_components=n_components)), ('classifier', best_model)
])

# Train and evaluate model with PCA pipeline.fit(X_train,
y_train)
y_pred_pca = pipeline.predict(X_test)
accuracy_pca=accuracy_score(y_test, y_pred_pca)
print(f"{best_model_name} Accuracy with PCA: {accuracy_pca:.4f}")
print(f"Accuracy Change with钽Impact: {accuracy_pca - accuracies[best_model_name]:.4f}")

# Save the processed dataset (optional)
X_processed_df['HeartDisease'] = y
X_processed_df.to_csv('processed_heart.csv',  index=False)
```

```
SVM Accuracy: 0.8889
Logistic Regression Accuracy: 0.8889 Random
Forest Accuracy: 0.8889

Best Model: SVM with Accuracy: 0.8889

Number of PCA components to retain 95% variance: 10 SVM Accuracy
with PCA: 0.8944
Accuracy Change with钽Impact: 0.0056
```

Start coding or generate with AI.

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot

21/4/25 Lab-9

Principle Component Analysis (PCA)

1) load dataset D with data point xi, where xi ∈ R^d (d-dimensional space)

2) Preprocess the data:
   (a) handle missing values
   (b) standardize the data by subtracting the mean of each features
   $x_j' = (x_j - m_j) / \sigma_j$
   where m_j is the mean of feature j and σ_j is the standard deviation.

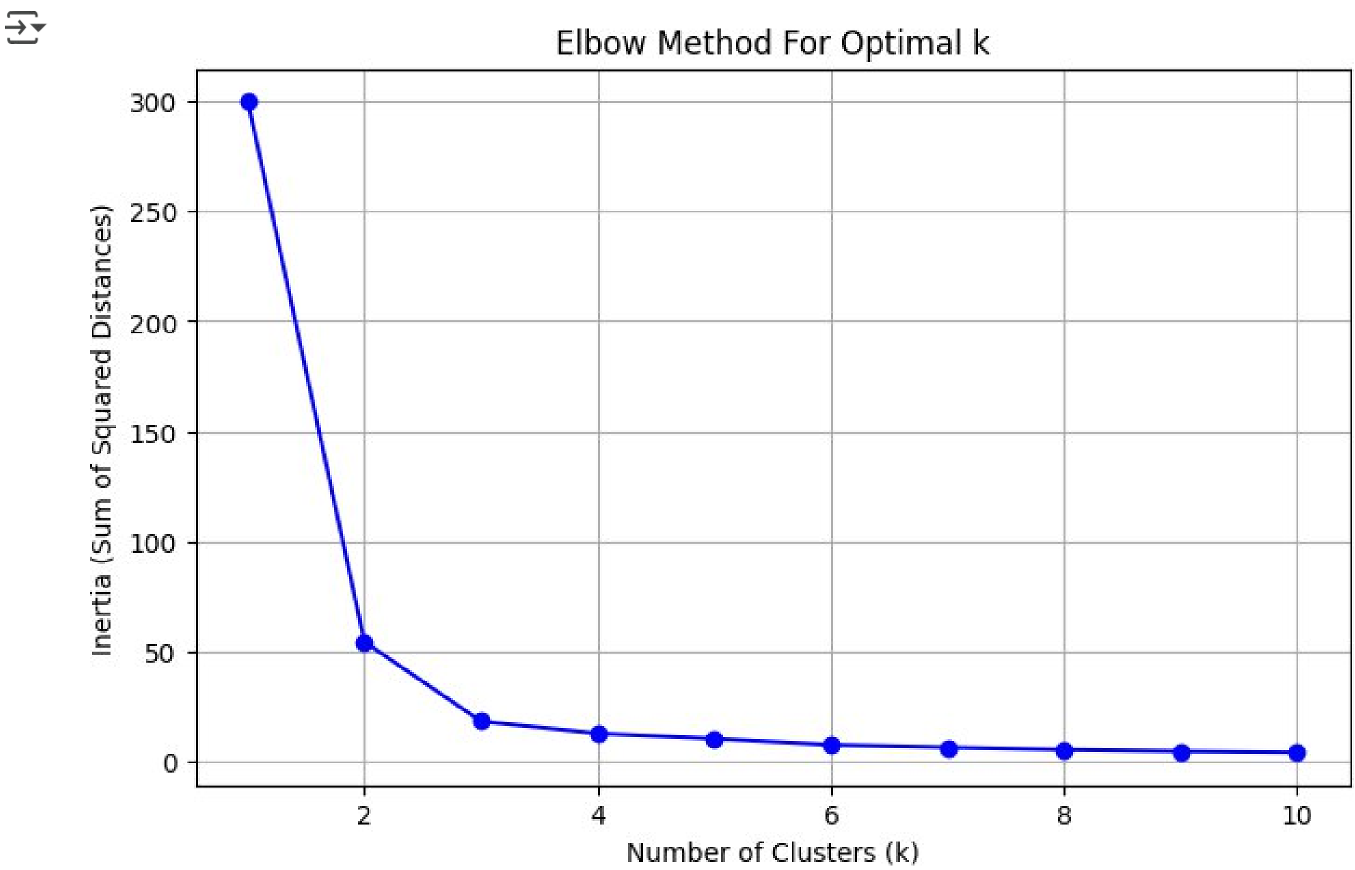
3) compute the covariance matrix Σ of the standardized data:
   $\Sigma = (1/N) * X^T * X$
   where X is the data Matrix

4) perform eigen value decomposition on the covariance matrix $\Sigma * V = \lambda * V$

5) Sort the eigenvalues in descending order and select the top K eigenvectors.

6) project the data onto the K eigenvalues
   $X_{new} = X * V$
   where V is the matrix of the top K eigenvector

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler df =

pd.read_csv("/iris.csv")

X = df[['petal_length', 'petal_width']]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

plt.figure(figsize=(8,  5))
plt.plot(k_range,  inertia,  'bo-')
plt.xlabel('Number of Clusters (k)')
```




```
plt.ylabel('Inertia (Sum of Squared Distances)') plt.title('Elbow Method For Optimal
k'
```