VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

AGAM TIWARI (1BM22CS023)

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 **Agam Tiwari (1BM22CS023),** 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.

Seema Patil Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link:

https://github.com/Agam1611/ML-Lab

Program 1

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

Screenshot:

Date: Page:

4/3/25 LAB 0 & LAB 1

TODO-1

```
import pandas as pd
(i) df = pd.read_csv('housing.csv')
(ii) df.head()
(iii) df.describe()
(iv) df['ocean proximity'].count()
(v) m = df.isnull().sum() , m0 = m[m>0]
```

Output

(ii)

	longitude	latitude	housing median age	total rooms	~~total bedrooms~~
0	-122.23	37.88	41.0	880.0	129.0
1	-122.22	37.86	21.0	7099.0	1106.0

	population	households	median income	median house	ocean proximity
0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	2401.0	1138.0	8.3014	358500.0	NEAR BAY

(iii)

	longitude	latitude	housing median age	~~total rooms~~	~~total bedrooms~~
min	-124.35	32.54	1.00	200	1.00

	population	households	median_income	median house	~~ocean proxy~~
min	3.00	1.00	0.499	14999.00	

Bafna Gold

Code:

```
import pandas as pd
data = {
  'USN':[22,23,24,30,11],
  'Name':['Aditya Singh','Agam Tiwari','Agneya','Akshat','Adarsh'],
  'Marks':[90,97,99,91,92]
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

```
Sample data:
   USN          Name  Marks
0   22  Aditya Singh     90
1   23   Agam Tiwari     97
2   24        Agneya     99
3   30        Akshat     91
4   11        Adarsh     92
```

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

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

         s4        s5        s6
0 -0.002592  0.019907 -0.017646
1 -0.039493 -0.068332 -0.092204
2 -0.002592  0.002861 -0.025930
3  0.034309  0.022688 -0.009362
4 -0.002592 -0.031988 -0.046641
```

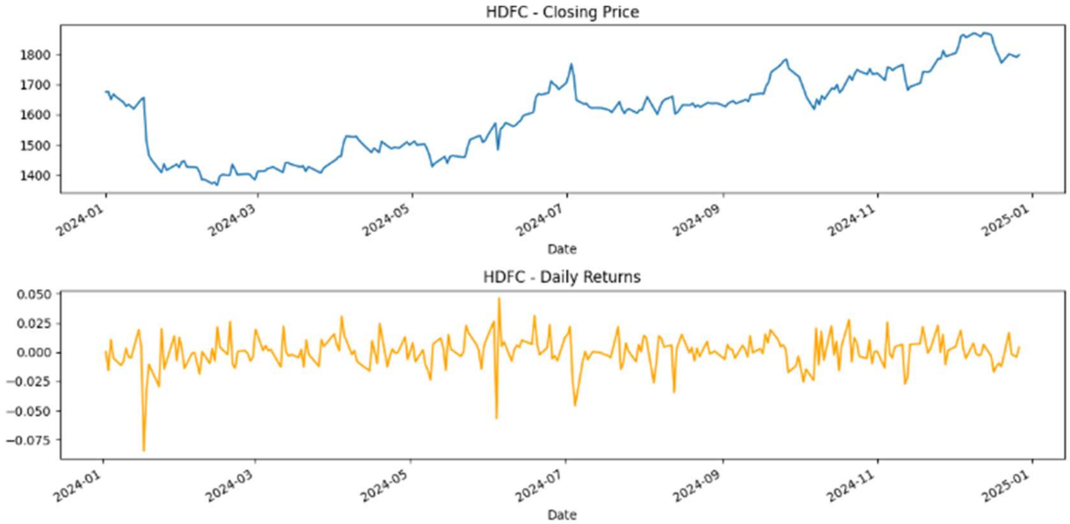
```
file_path = 'data.csv' # Ensure the file exists in the same directory
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")
```

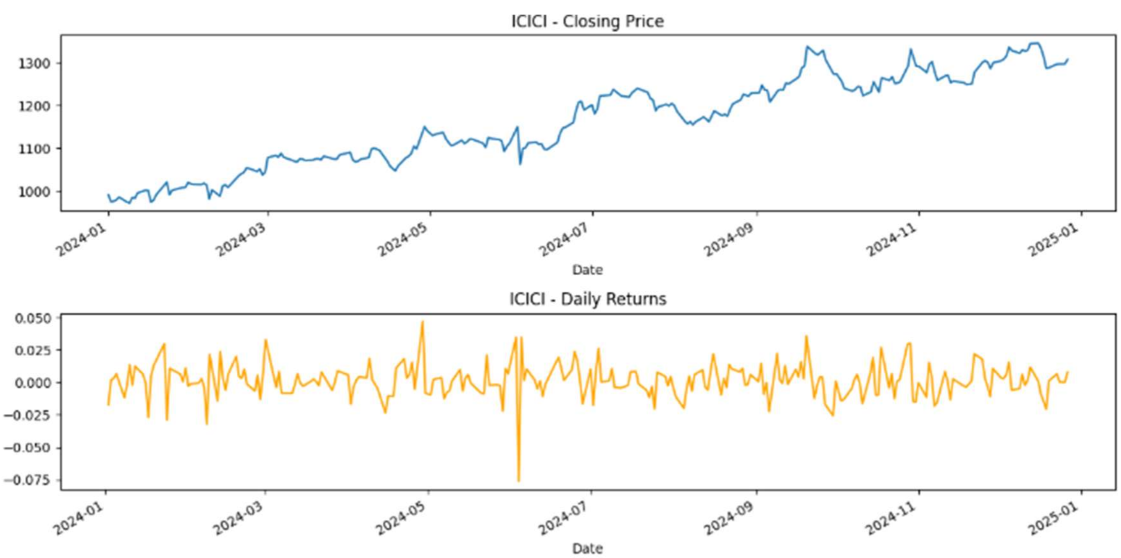
```
Sample data:
   ID     Name  Age         City
0   1    Alice   25     New York
1   2      Bob   30  Los Angeles
2   3  Charlie   35      Chicago
3   4    David   40      Houston
4   5      Eva   28      Phoenix
```

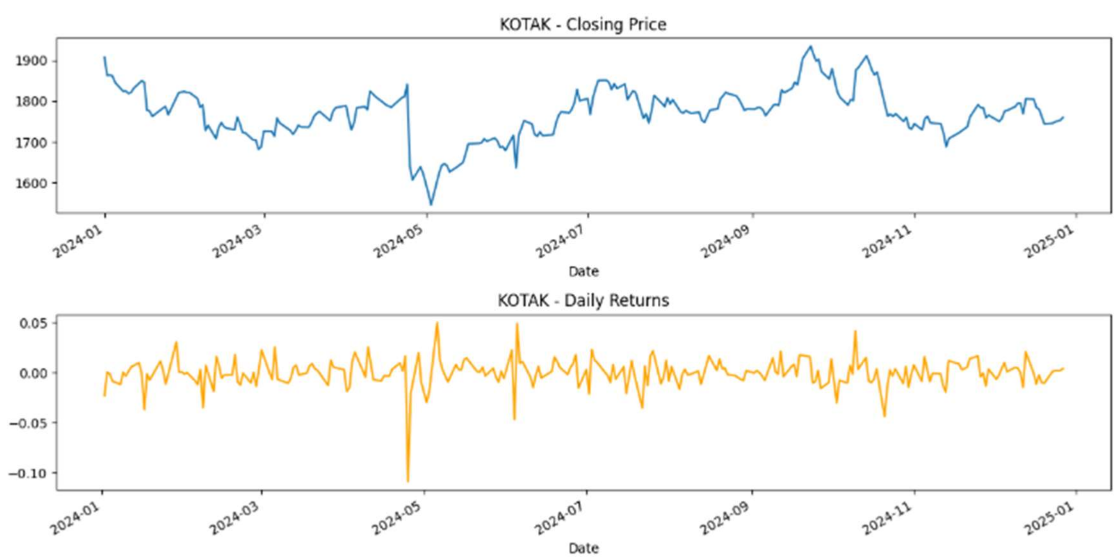
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",group_by='ticker')
print("\n\n\nFirst 5 rows of the dataset:")
print(data.head())

print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
hdfcData= data["HDFCBANK.NS"]
hdfcData['Daily Return'] = hdfcData['Close'].pct_change()

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfcData['Close'].plot(title="HDFC - Closing Price")
plt.subplot(2, 1, 2)
hdfcData['Daily Return'].plot(title="HDFC - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

#2
iciciData= data["ICICIBANK.NS"]
iciciData['Daily Return'] = iciciData['Close'].pct_change()

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
iciciData['Close'].plot(title="ICICI - Closing Price")
plt.subplot(2, 1, 2)
iciciData['Daily Return'].plot(title="ICICI - Daily Returns", color='orange')
plt.tight_layout()
plt.show()


#3
```

```
kotakData= data["KOTAKBANK.NS"]
kotakData['Daily Return'] = kotakData['Close'].pct_change()

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
kotakData['Close'].plot(title="KOTAK - Closing Price")
plt.subplot(2, 1, 2)
kotakData['Daily Return'].plot(title="KOTAK - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

```
First 5 rows of the dataset:
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      HDFCBANK.NS                                                   \
Price              Open         High          Low        Close    Volume
Date
2024-01-01  1683.017598  1686.125187  1669.206199  1675.223999   7119843
2024-01-02  1675.914685  1679.860799  1665.950651  1676.210571  14621046
2024-01-03  1679.071480  1681.735059  1646.466666  1650.363525  14194881
2024-01-04  1655.394910  1672.116520  1648.193203  1668.071777  13367028
2024-01-05  1664.421596  1681.932477  1645.628180  1659.538208  15944735

Ticker     ICICIBANK.NS
Price              Open        High         Low       Close    Volume
Date
2024-01-01   983.086778  996.273246  982.541485  990.869812   7683792
2024-01-02   988.490253  989.134730  971.883221  973.866150  16263825
2024-01-03   976.295294  979.567116  966.777197  975.650818  16826752
2024-01-04   977.980767  980.707295  973.519176  978.724365  22789140
2024-01-05   979.567084  989.779158  975.402920  985.218445  14875499

Shape of the dataset:
(244, 15)

Column names:
MultiIndex([('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'),
            ('ICICIBANK.NS',   'Open'),
            ('ICICIBANK.NS',   'High'),
            ('ICICIBANK.NS',    'Low'),
            ('ICICIBANK.NS',  'Close'),
            ('ICICIBANK.NS', 'Volume')],
           names=['Ticker', 'Price'])
```


HDFC - Closing Price
1800
1700
1600
1500
1400
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01
Date
HDFC - Daily Returns
0.050
0.025
0.000
−0.025
−0.050
−0.075
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01
Date

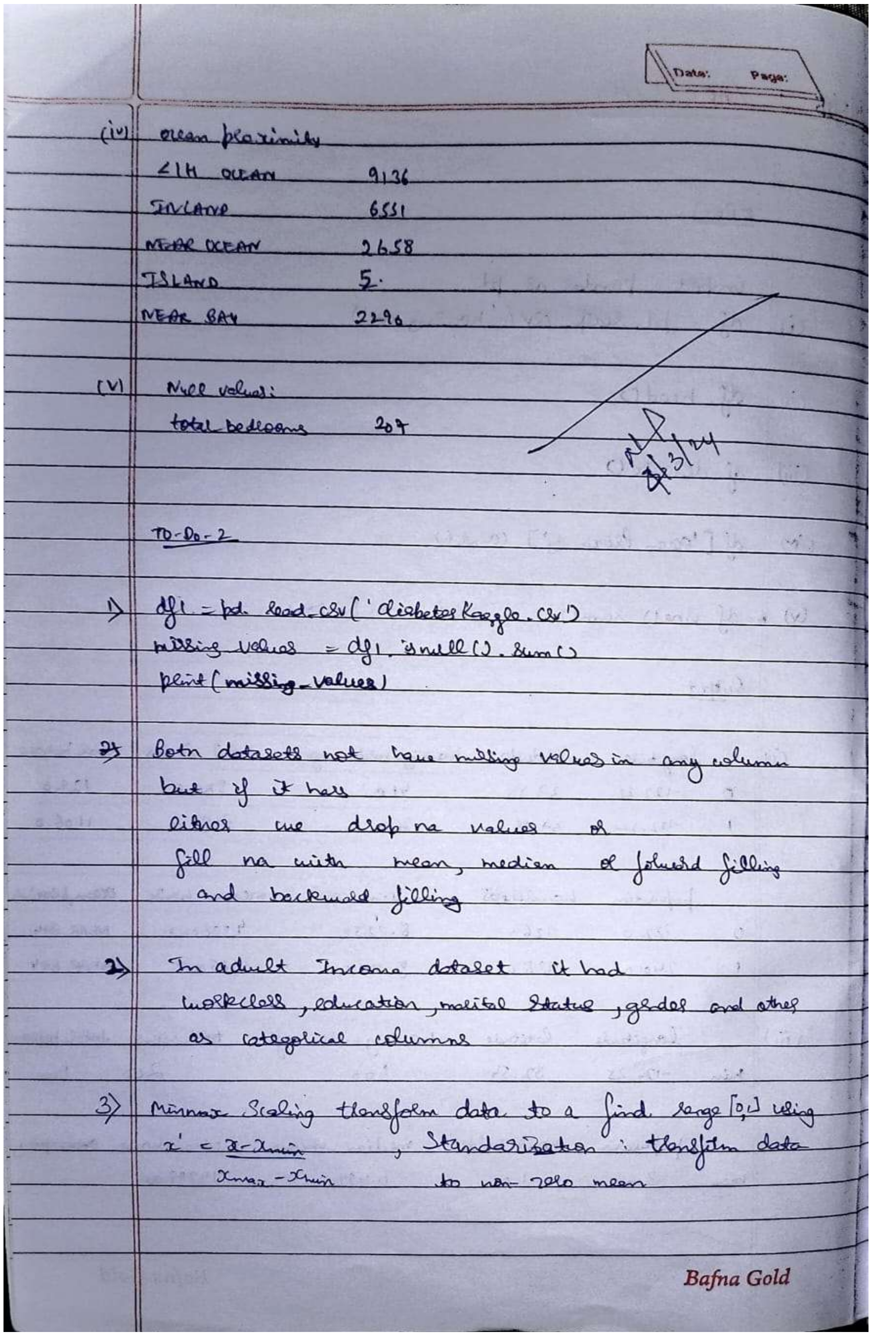

ICICI - Closing Price
1300
1200
1100
1000
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01
Date
ICICI - Daily Returns
0.050
0.025
0.000
−0.025
−0.050
−0.075
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01
Date


KOTAK - Closing Price
1900
1800
1700
1600
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01
Date
KOTAK - Daily Returns
0.05
0.00
−0.05
−0.10
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01
Date

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



Date: Page:

(iv) ocean proximity

<1H OCEAN	9136
INLAND	6551
NEAR OCEAN	2658
ISLAND	5.
NEAR BAY	2290

(v) Null values:

total_bedrooms 207

TO-DO-2

1) df1 = pd.read_csv('diabetes Kaggle.csv')
missing_values = df1.isnull().sum()
print(missing_values)

2) Both datasets not have missing values in any column but if it has
either we drop na values or
fill na with mean, median or forward filling and backward filling

2) In adult Income dataset it had workclass, education, marital status, gender and other as categorical columns

3) Minmax Scaling transform data to a fixed range [0,1] using $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$, Standarization : transform data to non-zero mean

Bafna Gold

Code:

```
import pandas as pd
df = pd.read_csv('housing.csv')
print(df.head())
print(df.describe())
print("\ncount of unique labels for “Ocean Proximity” column")
print(df['ocean_proximity'].count())

print("Null Values : ")
miss = df.isna().sum()
miss0 = miss[miss>0]
```

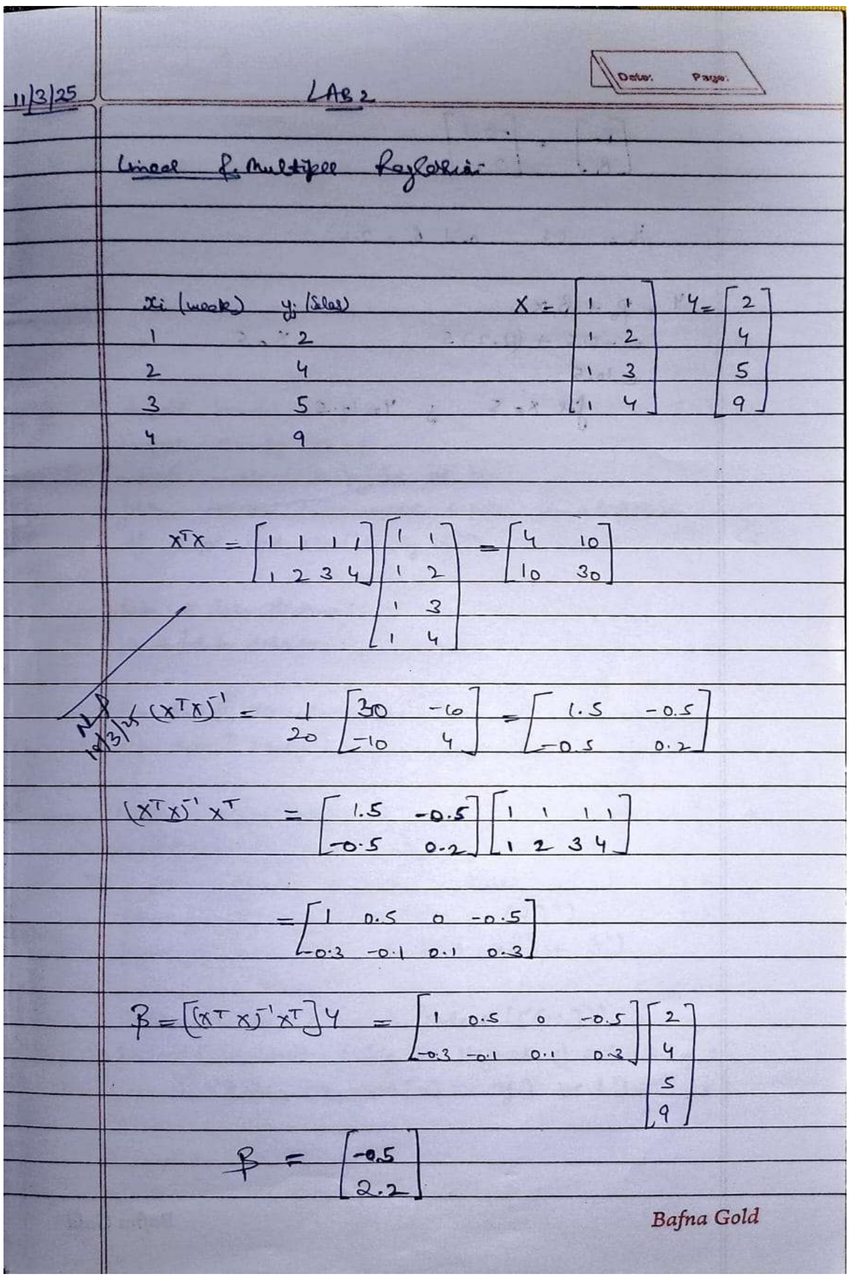
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25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

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
          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
20640
Null Values :
total_bedrooms    207
dtype: int64
```

```
#adult income dataset
df2 = pd.read_csv('adult.csv')
missing_values = df2.isnull().sum()
print(missing_values)
print(df2.head())
```

```
   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
```

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:



Date: Page:

11/3/25 LAB 2

Linear & Multiple Regression

x_i (week)	y_i (Sales)
1	2
2	4
3	5
4	9

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \quad Y = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$

$$X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

18/3/25

$$(X^T X)^{-1} = \frac{1}{20} \begin{bmatrix} 30 & -10 \\ -10 & 4 \end{bmatrix} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

$$(X^T X)^{-1} X^T = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix}$$

$$\beta = \left[(X^T X)^{-1} X^T\right] Y = \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$

$$\beta = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

Bafna Gold

$$\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

$\beta_0 = -0.5$ and $\beta_1 = 2.2$

$$y = \beta_0 + \beta_1 x$$
$$= -0.5 + (2.2)5 \qquad x = 5$$
$$= 10.5$$

for $x = 5$; $y = 10.5$

```
=> import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
df = pd.read_csv('salary.csv')

data = data.dropna()
print(data.describe())

x = data[['YearsExperience']]
y = data['Salary']

model = LinearRegression()
model.fit(x_train, y_train)
* y_pred = model.predict(x_test)
print(f"coefficient : {model.coef_[0]}")
print(f" Intercept : {model.intercept_}")

salary_12_year = model.predict([[12]])
print(f"Predicted salary for 12 years of experience :
    {salary_12_years[0]:.2f} US$")
```

Bafna Gold

Date: Page:

Output:

Coefficient : 9569.79
Intercept : 25499
Predicted Salary for 12 years of experience : 140337.5 US$

MAE : 4519.16
MSE : 27180506.8
RMSE : 5213.49

2) Considering the data file

```
from sklearn.preprocessing import OneHotEncoder

data = pd.read_csv("1000_Companies.csv")
data = data.dropna()

encoder = OneHotEncoder(drop='first', sparse_output=False)
state_encoded = encoder.fit_transform(data[['State']])

data = pd.concat([data.drop['State'], axis=1),
        state_encoded_df], axis=1)
```


Date: Page:

```
X = data[['R&D Spend', 'Administration', 'Marketing Spend']
    + list(State_encoded_df.columns)]
y = data['Profit']

model = LinearRegression()
model.fit(X_train, y_train)
* y_pred = model.predict

State_names = encoder.get_feature_names_out(['State'])
florida_encoded = (State_names == "State_Florida")
candidate_features = np.array([91694.48, 515841.3, 11931.24]
 + list(florida_encoded)).reshape(1, -1)

profit_prediction = model.predict(candidate_features)

print(f"Predicted profit for given candidate : {profit_
prediction[0]:.2f} US$")
```

Output

Coefficients : [5.3304e-01 1.13893e+00 8.3075e-02
 -8.74491e+02 -9.7133e+01]

Intercept : -82439.1536

Predicted Profit for given candidate : 554066.30 US$
MAE : 1404.44
MSE : 30775142.86
RMSE : 5547.53

Code:

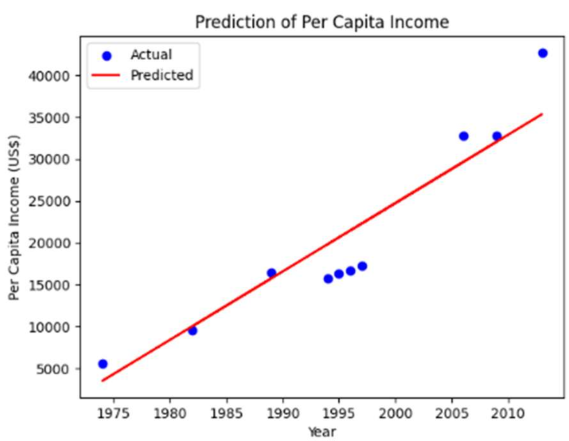
Linear Regression:

```
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
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)
```

rmse = np.sqrt(mse)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.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
```




```
Coefficient: 815.1425130089498
Intercept: -1605560.1987964255
Predicted per capita income in 2020: 41027.68 US$
Mean Absolute Error (MAE): 3240.91
Mean Squared Error (MSE): 15147815.55
Root Mean Squared Error (RMSE): 3892.02
```

Multiple Regression:

```
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_scor
# 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": 6,
                   "seven": 7, "eight": 8, "nine": 9, "ten": 10, "eleven": 11, "twelve": 12}
    return word_to_num.get(value.lower(), value) if isinstance(value, str) else value
# Apply conversion
data['experience'] = data['experience'].apply(convert_experience)
# Handle missing values by removing rows with NaN
data = data.dropna()
# Convert all columns to numeric
data = data.astype(float)
# 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_}")
```

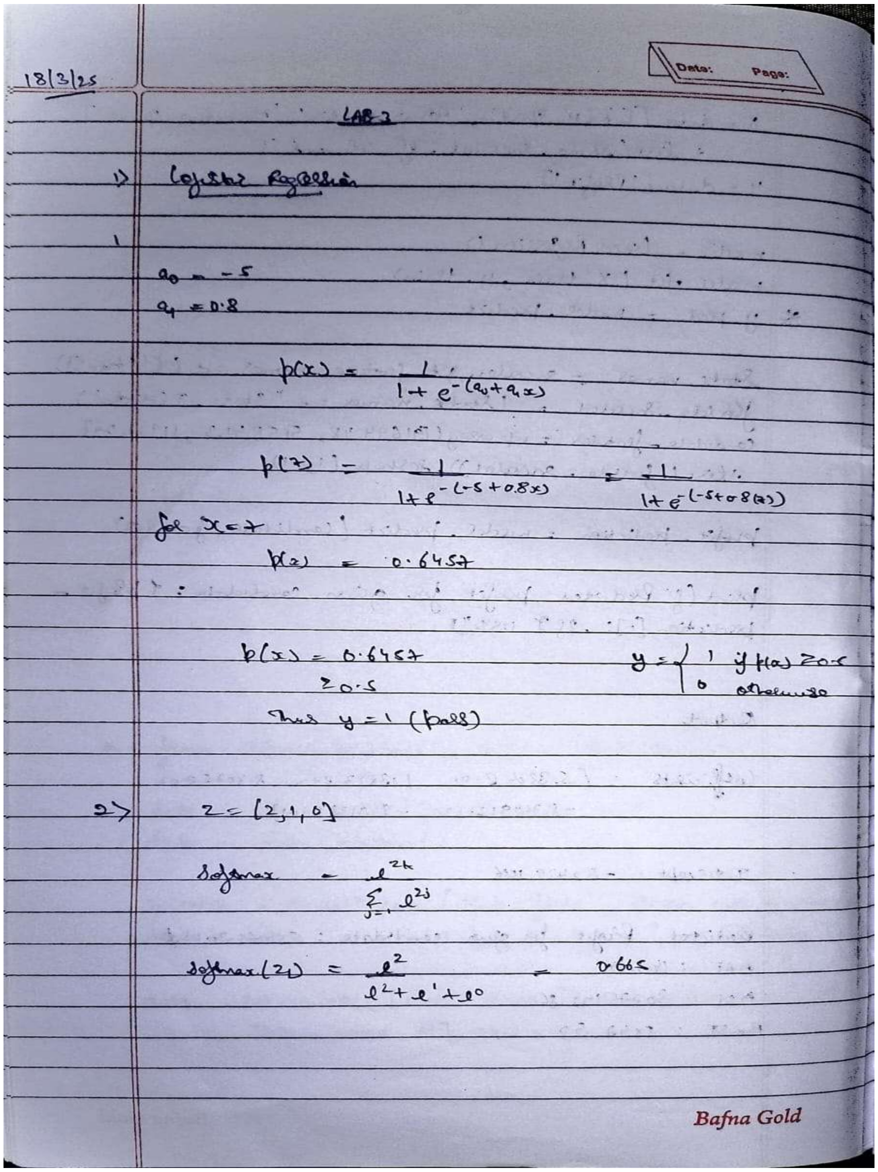
```
# Predict salary for given candidates
candidates = np.array([[2, 9, 6], [12, 10, 10]])
salary_predictions = model.predict(candidates)
print(f"Predicted salary for 2 yrs experience, 9 test score, 6 interview score: {salary_predictions[0]:.2f} US$")
print(f"Predicted salary for 12 yrs experience, 10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

# Calculate errors
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

```
Coefficients: [2687.5 2125.  1750. ]
Intercept: 21562.500000000022
Predicted salary for 2 yrs experience, 9 test score, 6 interview score: 56562.50 US$
Predicted salary for 12 yrs experience, 10 test score, 10 interview score: 92562.50 US$
Mean Absolute Error (MAE): 687.50
Mean Squared Error (MSE): 472656.25
Root Mean Squared Error (RMSE): 687.50
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:



18/3/25

Date: Page:

LAB 3

1> Logistic Regression

1

$a_0 = -5$

$a_1 = 0.8$

$$p(x) = \frac{1}{1+e^{-(a_0+a_1x)}}$$

$$p(7) = \frac{1}{1+e^{-(-5+0.8x)}} = \frac{1}{1+e^{-(-5+0.8(7))}}$$

for $x=7$

$$p(x) = 0.6457$$

$$p(x) = 0.6457 \geq 0.5$$

Thus $y = 1$ (pass)

$$y = \begin{cases} 1 & \text{if } p(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

2> $z = [2, 1, 0]$

$$\text{softmax} = \frac{e^{z_k}}{\sum_{j=1} e^{z_j}}$$

$$\text{softmax}(z_1) = \frac{e^2}{e^2+e^1+e^0} = 0.665$$

Bafna Gold

Date: Page:

```
accuracy = accuracy_score (y_test, y_pred)
print (f"Accuracy of the Multinomial Logistic
Regression model on the test set: {accuracy:.2f}")

confusion_matrix = metrics.confusion_matrix (y_test,
y_pred)
cm_display = metrics.ConfusionMatrixDisplay (
confusion_matrix = confusion_matrix, display_labels =
["Setosa", "Versicolor", "Virginica"])
cm_display.plot()
plt.show ()
```

<u>Output</u>

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

True label	10	0	0
	0	9	0
	0	0	11
	Setosa	Versicolor	Virginica

Predicted label

Date: Page:

softmax $(z_1) = \frac{e^1}{e^2+e^1+e^0} = 0.244$

softmax $(z_0) = \frac{e^0}{e^2+e^1+e^0} = 0.092$

Ans ⇒ 66.6%, 24.4%, 9.1%

13/3/25

Code :

1) Logistic Regression (~~Binary~~ multiclass Classification)

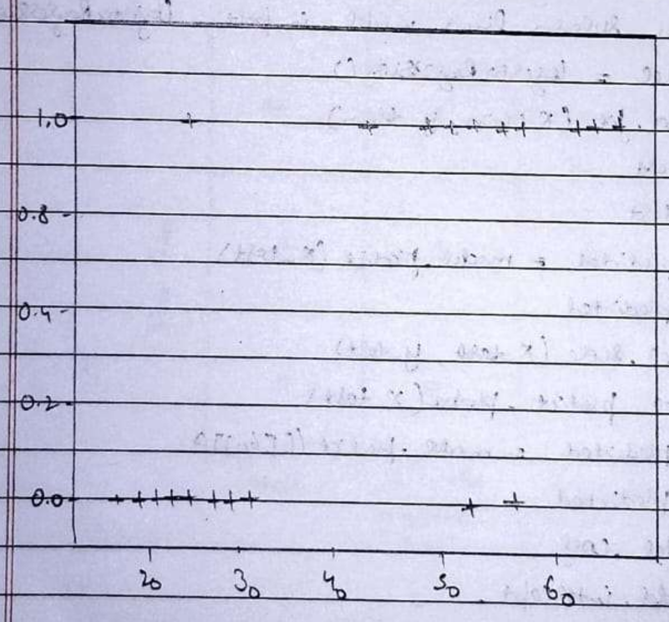
```
⇒ import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

iris = pd.read_csv("iris.csv")
iris.head()
X = iris.drop('species', axis='columns')
y = iris.species
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.2, random_state = 42
model = LogisticRegression(multi_class = 'multimodal')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Bafna Gold

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

def prediction_function (age):
    z = 0.127*age - 4.973
    y = sigmoid (z)
    return y

age = 35
prediction_function (age)
```

Output:

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



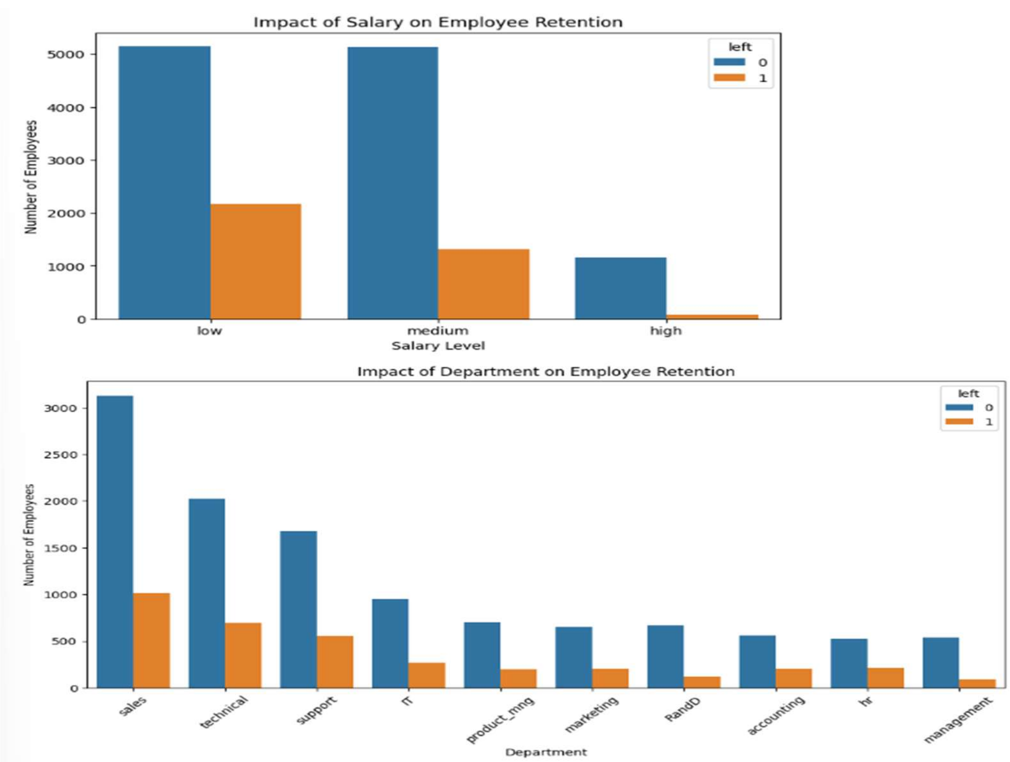
Code:
Binary Logical Regression

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report
# Load the dataset
file_path = 'HR_comma_sep.csv'
df = pd.read_csv(file_path)
# Display basic info
print(df.info())
print(df.head())
# Bar chart for Salary vs Retention
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='salary', hue='left')
plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary Level')
plt.ylabel('Number of Employees')
plt.show()
# Bar chart for Department vs Retention
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='Department', hue='left', order=df['Department'].value_counts().index)
plt.title('Impact of Department on Employee Retention')
plt.xlabel('Department')
plt.ylabel('Number of Employees')
plt.xticks(rotation=45)
plt.show()
# Define features and target variable
X = df[['satisfaction_level', 'number_project', 'average_montly_hours', 'time_spend_company', 'salary',
'Department']]
y = df['left']
# Preprocessing: Scale numeric features and one-hot encode categorical features
preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), ['satisfaction_level', 'number_project', 'average_montly_hours',
'time_spend_company']),
    ('cat', OneHotEncoder(), ['salary', 'Department'])
])

# Create pipeline with logistic regression
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

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

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

# Measure accuracy and display classification report
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy:.2%}')
print('Classification Report:')
print(report)

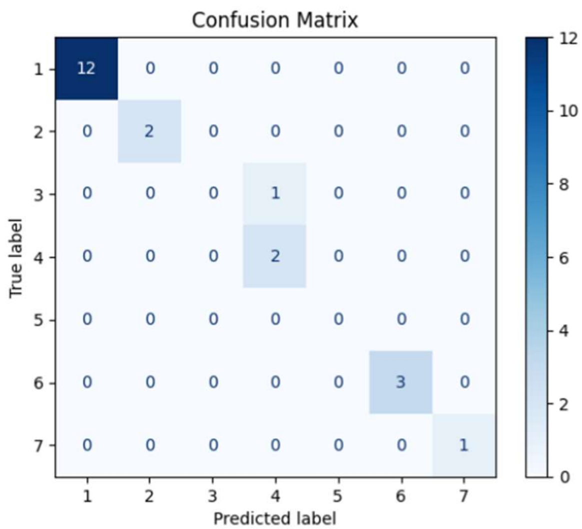



```
Accuracy: 76.23%
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.92      0.86      2294
           1       0.49      0.25      0.33       706

    accuracy                           0.76      3000
   macro avg       0.65      0.59      0.59      3000
weighted avg       0.73      0.76      0.73      3000
```

Multi Logical classification

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

# Load the datasets
zoo_data = pd.read_csv('zoo-data.csv')
zoo_class_type = pd.read_csv('zoo-class-type.csv')

# Display basic info
print(zoo_data.info())
print(zoo_data.head())
print(zoo_class_type.info())
print(zoo_class_type.head())

# Drop the animal_name column since it's not useful for prediction
zoo_data_cleaned = zoo_data.drop('animal_name', axis=1)

# Define features and target variable
X = zoo_data_cleaned.drop('class_type', axis=1)
y = zoo_data_cleaned['class_type']

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

# Train logistic regression model using softmax for multi-class classification
model = LogisticRegression(max_iter=1000, multi_class='multinomial', solver='lbfgs')
model.fit(X_train, y_train)

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

# Measure accuracy and generate classification report
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2%}')
print('Classification Report:')
print(report)

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=sorted(y.unique()))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=sorted(y.unique()))
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

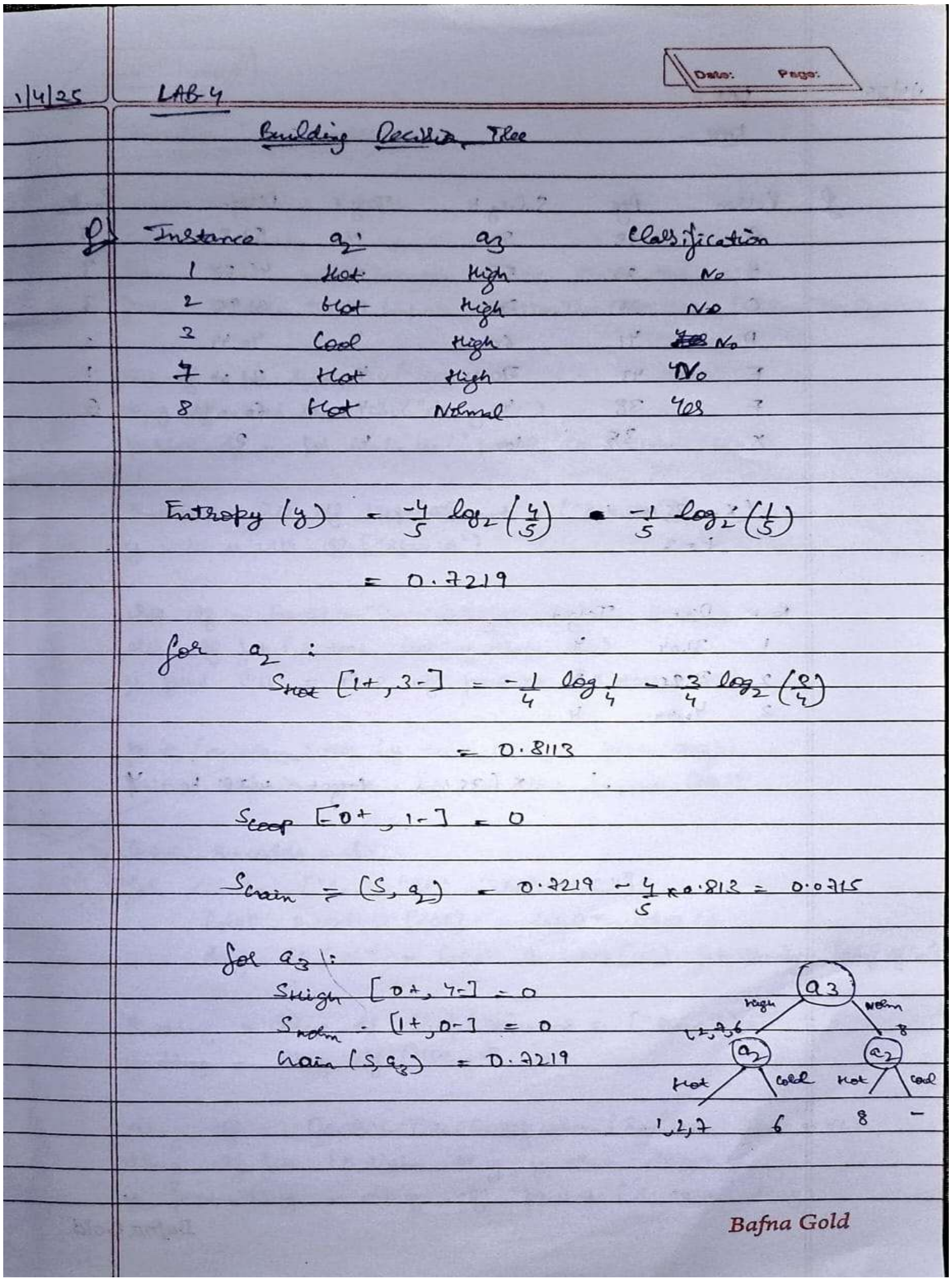
```
Accuracy: 95.24%
Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00         2
           3       0.00      0.00      0.00         1
           4       0.67      1.00      0.80         2
           6       1.00      1.00      1.00         3
           7       1.00      1.00      1.00         1

    accuracy                           0.95        21
   macro avg       0.78      0.83      0.80        21
weighted avg       0.92      0.95      0.93        21
```



Program 5

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

Screenshot:



1/4/25 LAB-4

Building Decision Tree

Q)

Instance	a_2	a_3	Classification
1	Hot	High	No
2	Hot	High	No
3	Cool	High	~~Yes~~ No
7	Hot	High	No
8	Hot	Normal	Yes

$$Entropy(y) = -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right)$$

$$= 0.7219$$

for a_2:

$$S_{Hot}[1+,3-] = -\frac{1}{4}\log\frac{1}{4} - \frac{3}{4}\log_2\left(\frac{3}{4}\right)$$

$$= 0.8113$$

$$S_{cool}[0+,1-] = 0$$

$$S_{Gain} = (S, a_2) = 0.7219 - \frac{4}{5} \times 0.813 = 0.0715$$

for a_3:

$$S_{High}[0+,4-] = 0$$

$$S_{norm}[1+,0-] = 0$$

$$Gain(S, a_3) = 0.7219$$

Date: Page:

Decision Tree Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

iris_df = pd.read_csv("iris.csv")
drug_df = pd.read_csv("drug.csv")
petrol_df = pd.read_csv("petrol_consumption.csv")

x_iris = iris_df.drop(columns = ["species"])
y_iris = iris_df["species"]

iris_clf = DecisionTreeClassifier(random_state = 42)
iris_clf.fit(x_train_iris, y_train_iris)
y_pred_iris = iris_clf.predict(x_test_iris)

print(accuracy_score(y_test_iris, y_pred_iris))
print(confusion_matrix(y_test_iris, y_pred_iris))

label_encoders = {}
for col in ["Sex", "BP", "cholesterol"]:
    label_encoders[col] = LabelEncoder()
    drug_df[col] = label_encoders[col].fit_transform(drug_df[col])

x_drug = drug_df.drop(columns = ["Drug"])
y_drug = drug_df["Drug"]

drug_clf = DecisionTreeClassification(random_state = 42)
drug_clf.fit(x_train_drug, y_train_drug)
y_pred_drug = drug_clf.predict(x_test_drug)
```

Bafna Gold

Date: Page:

Output:

Iris Accuracy : 1.0

The Confusion Matrix

[[10 0 0

0 9 0

0 0 11]]

Drug Confusion Matrix :

[[6 0 0 0 0

0 3 0 0 0

0 0 5 0 0

0 0 11 0

0 0 0 0 15]]

Bafna Gold

Code:

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

# Load datasets
iris_df = pd.read_csv("iris.csv")
drug_df = pd.read_csv("drug.csv")
petrol_df = pd.read_csv("petrol_consumption.csv")

# Prepare IRIS dataset
X_iris = iris_df.drop(columns=["species"])
y_iris = iris_df["species"]
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

# Train Decision Tree Classifier for IRIS
iris_clf = DecisionTreeClassifier(random_state=42)
iris_clf.fit(X_train_iris, y_train_iris)
y_pred_iris = iris_clf.predict(X_test_iris)

# IRIS Metrics
print("Iris Accuracy:", accuracy_score(y_test_iris, y_pred_iris))
print("Iris Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_iris))

# Prepare Drug dataset (encode categorical variables)
label_encoders = {}
for col in ["Sex", "BP", "Cholesterol"]:
    label_encoders[col] = LabelEncoder()
    drug_df[col] = label_encoders[col].fit_transform(drug_df[col])

X_drug = drug_df.drop(columns=["Drug"])
y_drug = drug_df["Drug"]
X_train_drug, X_test_drug, y_train_drug, y_test_drug = train_test_split(X_drug, y_drug, test_size=0.2,
random_state=42)

# Train Decision Tree Classifier for Drug dataset
drug_clf = DecisionTreeClassifier(random_state=42)
drug_clf.fit(X_train_drug, y_train_drug)
y_pred_drug = drug_clf.predict(X_test_drug)

# Drug Metrics
print("Drug Accuracy:", accuracy_score(y_test_drug, y_pred_drug))
print("Drug Confusion Matrix:\n", confusion_matrix(y_test_drug, y_pred_drug))

# Prepare Petrol Consumption dataset
X_petrol = petrol_df.drop(columns=["Petrol_Consumption"])
y_petrol = petrol_df["Petrol_Consumption"]
X_train_petrol, X_test_petrol, y_train_petrol, y_test_petrol = train_test_split(X_petrol, y_petrol, test_size=0.2,
```

```
random_state=42)

# Train Decision Tree Regressor for Petrol dataset
petrol_reg = DecisionTreeRegressor(random_state=42)
petrol_reg.fit(X_train_petrol, y_train_petrol)
y_pred_petrol = petrol_reg.predict(X_test_petrol)

# Petrol Regression Metrics
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test_petrol, y_pred_petrol))
print("Mean Squared Error (MSE):", mean_squared_error(y_test_petrol, y_pred_petrol))
print("Root Mean Squared Error (RMSE):", np.sqrt(mean_squared_error(y_test_petrol, y_pred_petrol)))
```

```
Iris Accuracy: 1.0
Iris Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Drug Accuracy: 1.0
Drug 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]]
Mean Absolute Error (MAE): 94.3
Mean Squared Error (MSE): 17347.7
Root Mean Squared Error (RMSE): 131.7106677532234
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot :

1/4/25 LAB 5

KNN :-

Person	Age	Salary K	Target	Distance	Rank
A	18	50	N	52.8	5
B	23	55	N	46.57	4
C	24	70	N	31.95	3
D	41	60	Y	40.44	2
E	43	70	Y	31.04	1
F	38	40	Y	60.07	6
X	35	100	Y	?	

(X = 35, 100) =

k = 3

Rank	Dist	Target
1	31.04	Y
2	31.95	N
3	40.44	Y

∴ X (35, 100) ~~target~~ will be Y

Bafna Gold

Date: Page:

KNN Code:

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

diabetes = pd.read_csv('diabetes.csv')

x_diabetes = diabetes.iloc[:, :-1].values
y_diabetes = diabetes.iloc[:, -1].values
scaler = StandardScaler()
X_train_diabetes, X_test_diabetes, y_train_diabetes,
y_test_diabetes = train_test_split (X_scaled_diabetes,
y_diabetes, test_size = 0.2, random_state = 42)

knn_diabetes = KNeighborsClassifier (n_neighbors = 5)
knn_diabetes.fit (x_train_diabetes, y_train_diabetes)

y_pred_diabetes = knn_diabetes.predict (x_test_diabetes)
print ("Diabetes Dataset Accuracy:", accuracy_score (y_test_diabetes
print ("Diabetes Confusion Matrix:\n",
confusion_matrix (y_test_diabetes, y_pred_diabetes))
```

Date: Page:

Output :

Diabetes Dataset Accuracy : 0.688

Diabetes Confusion matrix :

[[79 20

28 27]]

~~Optional~~

Diabetes Classification Report :

	precision	recall	f1-score	support
0	0.74	0.80	0.77	99
1	0.57	0.49	0.53	55
accuracy			0.69	154
macro avg	0.66	0.64	0.65	154
weighted avg	0.68	0.69	0.68	154

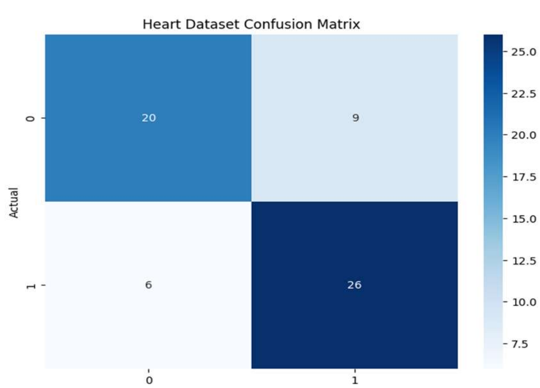
1/4/25

Code:

```
import pandas as pd
import numpy as np
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
import matplotlib.pyplot as plt
import seaborn as sns
iris = pd.read_csv('iris.csv')
X_iris = iris.iloc[:, :-1].values
y_iris = iris.iloc[:, -1].values
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)
knn_iris = KNeighborsClassifier(n_neighbors=3)
knn_iris.fit(X_train_iris, y_train_iris)
y_pred_iris = knn_iris.predict(X_test_iris)
print("IRIS Dataset Accuracy:", accuracy_score(y_test_iris, y_pred_iris))
print("IRIS Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_iris))
print("IRIS Classification Report:\n", classification_report(y_test_iris, y_pred_iris))
diabetes = pd.read_csv('diabetes.csv')
X_diabetes = diabetes.iloc[:, :-1].values
y_diabetes = diabetes.iloc[:, -1].values
scaler = StandardScaler()
X_scaled_diabetes = scaler.fit_transform(X_diabetes)
X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes = train_test_split(X_scaled_diabetes,
y_diabetes, test_size=0.2, random_state=42)
knn_diabetes = KNeighborsClassifier(n_neighbors=5)
knn_diabetes.fit(X_train_diabetes, y_train_diabetes)
y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)
print("Diabetes Dataset Accuracy:", accuracy_score(y_test_diabetes, y_pred_diabetes))
print("Diabetes Confusion Matrix:\n", confusion_matrix(y_test_diabetes, y_pred_diabetes))
print("Diabetes Classification Report:\n", classification_report(y_test_diabetes, y_pred_diabetes))
X_heart = heart.iloc[:, :-1].values
y_heart = heart.iloc[:, -1].values
X_train_heart, X_test_heart, y_train_heart, y_test_heart = train_test_split(X_heart, y_heart, test_size=0.2,
random_state=42)
accuracy_scores = []
for k in range(1, 21):
   knn_heart = KNeighborsClassifier(n_neighbors=k)
   knn_heart.fit(X_train_heart, y_train_heart)
   y_pred_heart = knn_heart.predict(X_test_heart)
   accuracy_scores.append(accuracy_score(y_test_heart, y_pred_heart))
optimal_k = accuracy_scores.index(max(accuracy_scores)) + 1
print("Optimal K for Heart Dataset:", optimal_k)
knn_heart = KNeighborsClassifier(n_neighbors=optimal_k)
knn_heart.fit(X_train_heart, y_train_heart)
y_pred_heart = knn_heart.predict(X_test_heart)
print("Heart Dataset Accuracy:", accuracy_score(y_test_heart, y_pred_heart))
print("Heart Confusion Matrix:\n", confusion_matrix(y_test_heart, y_pred_heart))
print("Heart Classification Report:\n", classification_report(y_test_heart, y_pred_heart))
plt.figure(figsize=(8, 6))
```

```
sns.heatmap(confusion_matrix(y_test_heart, y_pred_heart), annot=True, fmt="d", cmap="Blues")
plt.title("Heart Dataset Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
IRIS Dataset Accuracy: 1.0
IRIS Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
IRIS Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

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

Diabetes Dataset Accuracy: 0.6883116883116883
Diabetes Confusion Matrix:
 [[79 20]
 [28 27]]
Diabetes Classification Report:
               precision    recall  f1-score   support

           0       0.74      0.80      0.77        99
           1       0.57      0.49      0.53        55

    accuracy                           0.69       154
   macro avg       0.66      0.64      0.65       154
weighted avg       0.68      0.69      0.68       154

Optimal K for Heart Dataset: 11
Heart Dataset Accuracy: 0.7540983606557377
Heart Confusion Matrix:
 [[20  9]
 [ 6 26]]
Heart Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.69      0.73        29
           1       0.74      0.81      0.78        32

    accuracy                           0.75        61
   macro avg       0.76      0.75      0.75        61
weighted avg       0.76      0.75      0.75        61
```



Program7

Build Support vector machine model for a given dataset

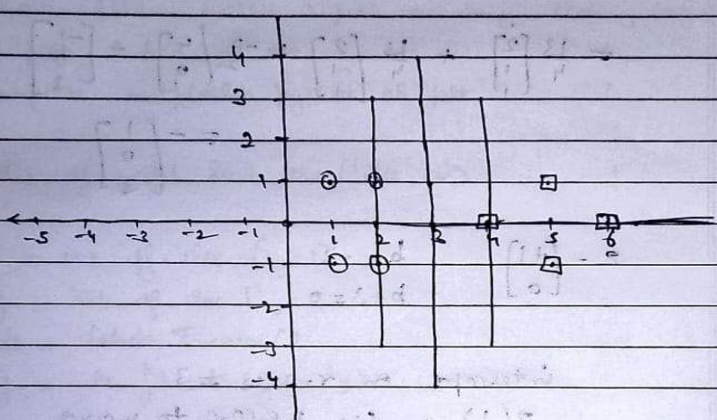
Screenshot:

8/4/25 SVM

Q Draw an optimal hyperplane using linear SVM to classify the following points

$\{(1,1), (2,1), (1,-1), (2,-1)\}$ +ve labelled

$\{(4,0), (5,1), (5,-1), (6,0)\}$ -ve labelled



$s_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ $s_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$ $s_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

$\tilde{s}_1 = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$ $\tilde{s}_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$ $\tilde{s}_3 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

$$\alpha_1 \tilde{s}_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 \tilde{s}_1 + \alpha_3 \tilde{s}_3 \tilde{s}_1 = +1$$
$$\alpha_1 \tilde{s}_1 \tilde{s}_2 + \alpha_2 \tilde{s}_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3 \tilde{s}_2 = +1$$
$$\alpha_1 \tilde{s}_1 \tilde{s}_3 + \alpha_2 \tilde{s}_2 \tilde{s}_3 + \alpha_3 \tilde{s}_3 \tilde{s}_3 = -1$$

Bafna Gold

$\alpha_1(6) + \alpha_2(4) + \alpha_3(9) = +1$
$\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1$
$\alpha_1(9) + \alpha_2(9) + \alpha_3(17) = -1$

$w = \alpha_1 \tilde{S}_1 + \alpha_2 \tilde{S}_2 + \alpha_3 \tilde{S}_3$

$= \frac{13}{4}\begin{bmatrix}2\\1\\1\end{bmatrix} + \frac{13}{4}\begin{bmatrix}2\\-1\\1\end{bmatrix} + \frac{-7}{2}\begin{bmatrix}4\\0\\1\end{bmatrix} = \begin{bmatrix}-1\\0\\3\end{bmatrix}$

$= -\begin{bmatrix}1\\0\\-3\end{bmatrix}$

$z = \begin{bmatrix}+1\\0\end{bmatrix}$ $\quad b = -3$

$b+3=0$

intercept on x-axis $= 3$

$z\begin{pmatrix}1\\0\end{pmatrix} \rightarrow$ line parallel to y-axis

Date: Page:

SVM Codes

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

iris_df = pd.read_csv('iris.csv')

X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]
le = LabelEncoder()
y = le.fit_transform(y)

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

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

for kernel in ['linear', 'rbf']:
    clf = SVC(kernel = kernel)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
```

Bafna Gold

Date: Page:

```
print(f"\nSVM with {kernel} Kernel")
print(f"Accuracy : {acc:.4f}")
print("Confusion Matrix:")
print(cm)

sns.heatmap(cm, annot=True, fmt='d',
    cmap='Blues')
plt.title(f'Confusion Matrix ({kernel} Kernel)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Output

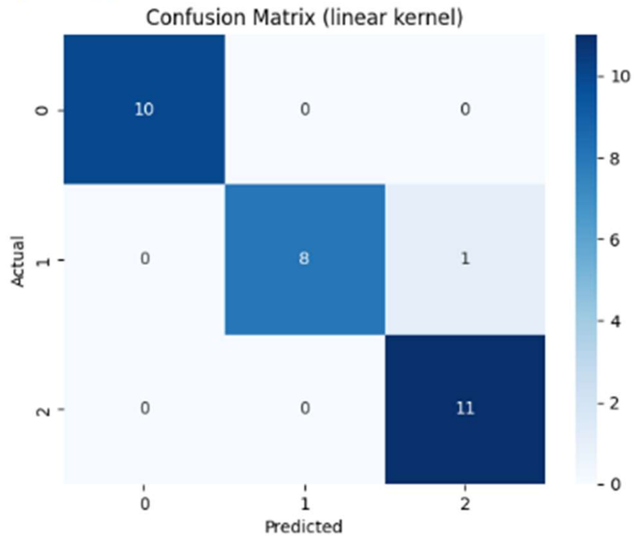
```
SVM with linear Kernel
Accuracy : 0.9667
Confusion Matrix:
[[10 0 0]
 [0 8 1]
 [0 0 11]]

SVM with rbf Kernel
Accuracy : 1.0000
Confusion Matrix:
[[10 0 0]
 [0 9 0]
 [0 0 11]]
```

8/4/2

Code:
Iris.csv

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

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

# Encode labels if necessary
X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]
le = LabelEncoder()
y = le.fit_transform(y)

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

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train and evaluate both linear and RBF kernels
for kernel in ['linear', 'rbf']:
    clf = SVC(kernel=kernel)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    print(f"\nSVM with {kernel} kernel")
    print(f"Accuracy: {acc:.4f}")
    print("Confusion Matrix:")
    print(cm)

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

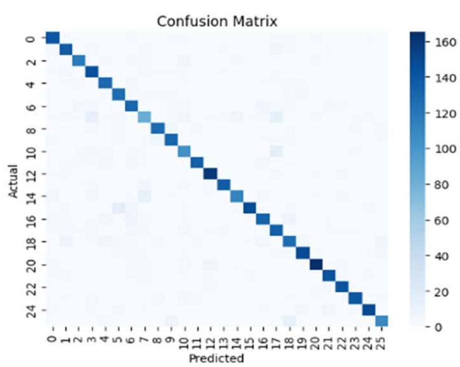
```
SVM with linear kernel
Accuracy: 0.9667
Confusion Matrix:
[[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]
```

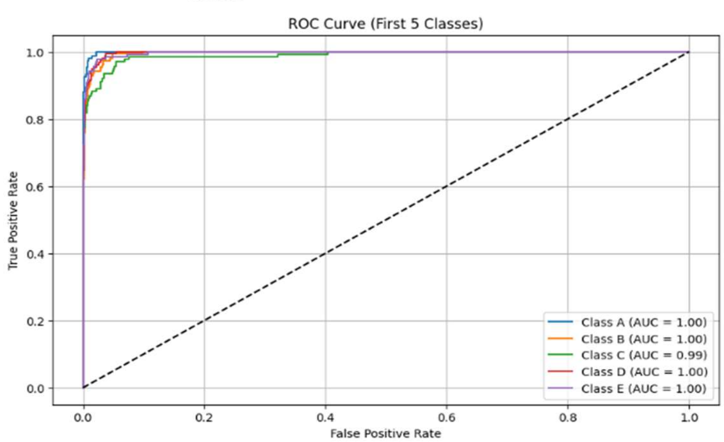


Letter Recognition

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
# Load the Letter Recognition dataset
letter_df = pd.read_csv('letter-recognition.csv')
X = letter_df.iloc[:, 1:]
y = letter_df.iloc[:, 0]
# Encode labels
le = LabelEncoder()
y_enc = le.fit_transform(y)
# Binarize for ROC
y_bin = label_binarize(y_enc, classes=range(len(le.classes_)))
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_enc, test_size=0.2, random_state=42)
# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Train SVM
clf = SVC(kernel='linear', probability=True)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_score = clf.predict_proba(X_test)
# Accuracy & confusion matrix
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print(f"\nLetter Recognition - SVM (RBF Kernel)")
print(f"Accuracy: {acc:.4f}")
print("Confusion Matrix:")
print(cm)
```

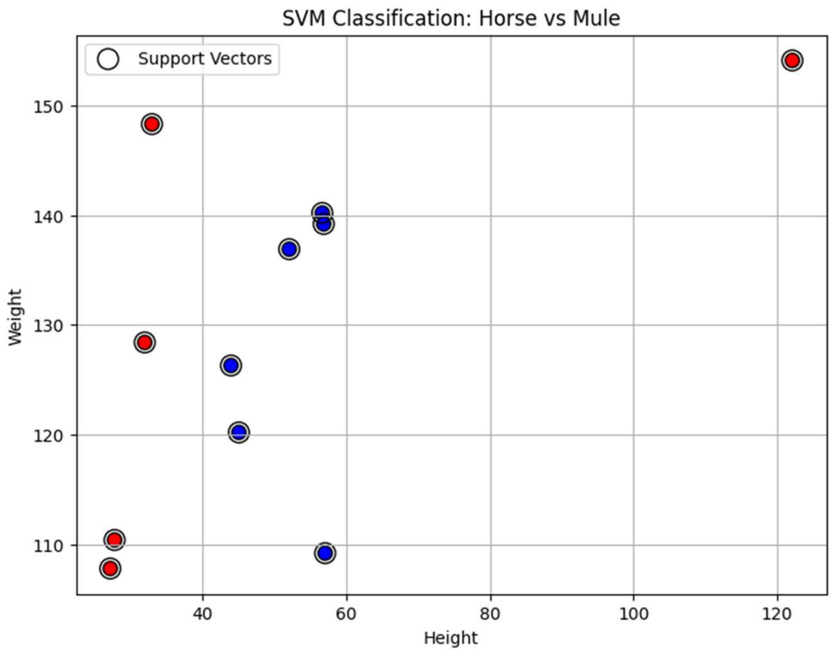
```
# Plot confusion matrix
sns.heatmap(cm, cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
# ROC & AUC for first few classes
y_test_bin = label_binarize(y_test, classes=range(len(le.classes_)))
fpr = dict()
tpr = dict()
roc_auc = dict()
n_classes = y_test_bin.shape[1]
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
# Plot ROC curves for first 5 classes
plt.figure(figsize=(10, 6))
for i in range(5):  # limit to first 5 for clarity
    plt.plot(fpr[i], tpr[i], label=f'Class {le.classes_[i]} (AUC = {roc_auc[i]:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve (First 5 Classes)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.show()
```





Horse Mule dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.metrics import accuracy_score

# Step 1: Create dataset
data = {
    "Height": [44.0, 52.1, 57.1, 33.0, 27.8, 27.2, 32.0, 45.1, 56.7, 56.9, 122.1],
    "Weight": [126.3, 136.9, 109.2, 148.3, 110.4, 107.8, 128.4, 120.2, 140.2, 139.2, 154.1],
    "Label":  ["Horse", "Horse", "Horse", "Mule", "Mule", "Mule", "Mule", "Horse", "Horse", "Horse",
"Mule"]
}
df = pd.DataFrame(data)

# Step 2: Save CSV
df.to_csv("horses_mules_dataset.csv", index=False)

# Step 3: Load dataset
df = pd.read_csv("horses_mules_dataset.csv")
# Step 4: Preprocess data
X = df[["Height", "Weight"]].values
y = df["Label"].map({"Horse": 0, "Mule": 1}).values  # Convert to numeric
# Step 5: Train SVM model
model = svm.SVC(kernel='linear')
model.fit(X, y)
# Step 6: Plot data and support vectors
plt.figure(figsize=(8,6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', s=70, edgecolors='k')
plt.xlabel("Height")
plt.ylabel("Weight")
plt.title("SVM Classification: Horse vs Mule")
# Plot support vectors
plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
            s=150, facecolors='none', edgecolors='k', label='Support Vectors')
plt.legend()
plt.grid(True)
plt.show()
# Step 7: Accuracy
y_pred = model.predict(X)
acc = accuracy_score(y, y_pred)
print("Accuracy:", acc)
# Step 8: Support Vectors
print("Support Vectors:\n", model.support_vectors_)
```




```
Accuracy: 0.9090909090909091
Support Vectors:
 [[ 44.  126.3]
 [ 52.1 136.9]
 [ 57.1 109.2]
 [ 45.1 120.2]
 [ 56.7 140.2]
 [ 56.9 139.2]
 [ 33.  148.3]
 [ 27.8 110.4]
 [ 27.2 107.8]
 [ 32.  128.4]
 [122.1 154.1]]
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:

Date: Page:

15/4/25

Lab-8 RF

Q Difference b/w decision tree and Random Forest

Ans

Feature	Decision Tree	Random Forest
model Type	Single tree based model	Ensemble of multiple decision trees
Accuracy	Lower, prone to overfitting	Higher, better generalization
Overfitting	High risk, especially on noise data	Reduced due to averaging over many trees
Training Time	Fast (Simple to build one tree)	Slower (due to training multiple trees)
Interpretability	Easy to interpret and visualize	Harder to interpret (many trees)
Robustness	Sensitive to data changes	More robust due to ensemble voting

Q Parameters of RandomForestClassifier ()

Ans 1) n_estimators → Number of trees in the forest

2) criterion → Function to measure split quality : "gini" or "entropy"

3) max_depth → Maximum depth of each tree

4) min_samples_split → minimum samples required to split a node

Bafna Gold

Q Algorithm of Random Forest

1) Input : Dataset D with N Samples
Number of trees T
Number of features F of select per split

2) For each tree (repeat T times)
1) Draw a bootstrap sample from dataset
2) Build a Decision Tree
• At each Node
• Randomly select F features from total set
• Find the best feature

15/4/25 • Continue until stopping criteria is met

3) Aggregate Predictions
For classification : majority vote from all trees
For Regression : avg of all tree outputs
4) Output

Bafna Gold

Code:
Iris.csv

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
iris_df = pd.read_csv('iris.csv')
# Features and target
X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train model with default n_estimators = 10
model_default = RandomForestClassifier(n_estimators=10, random_state=42)
model_default.fit(X_train, y_train)
y_pred_default = model_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default score with n_estimators=10: {default_score:.4f}")
# Fine-tune n_estimators
best_score = 0
best_n = 10
for n in range(5, 105, 5):
    model = RandomForestClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    print(f"n_estimators={n}, Accuracy={score:.4f}")
    if score > best_score:
        best_score = score
        best_n = n

print(f"\nBest score: {best_score:.4f} with n_estimators={best_n}")
```

```
Default score with n_estimators=10: 1.0000
n_estimators=5, Accuracy=0.9667
n_estimators=10, Accuracy=1.0000
n_estimators=15, Accuracy=1.0000
n_estimators=20, Accuracy=1.0000
n_estimators=25, Accuracy=1.0000
n_estimators=30, Accuracy=1.0000
n_estimators=35, Accuracy=1.0000
n_estimators=40, Accuracy=1.0000
n_estimators=45, Accuracy=1.0000
n_estimators=50, Accuracy=1.0000
n_estimators=55, Accuracy=1.0000
n_estimators=60, Accuracy=1.0000
n_estimators=65, Accuracy=1.0000
n_estimators=70, Accuracy=1.0000
n_estimators=75, Accuracy=1.0000
n_estimators=80, Accuracy=1.0000
n_estimators=85, Accuracy=1.0000
n_estimators=90, Accuracy=1.0000
n_estimators=95, Accuracy=1.0000
n_estimators=100, Accuracy=1.0000

Best score: 1.0000 with n_estimators=10
```

Train.csv

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

# Load dataset
df = pd.read_csv('train.csv')
# Separate features and target
X_raw = df.iloc[:, :-1]
y_raw = df.iloc[:, -1]
# Encode categorical features using one-hot encoding
X_encoded = pd.get_dummies(X_raw)
# Encode target if it's categorical
if y_raw.dtype == 'object':
    y_encoded = LabelEncoder().fit_transform(y_raw)
else:
    y_encoded = y_raw
# Handle missing values using imputation
imputer = SimpleImputer(strategy='most_frequent')
X_imputed = pd.DataFrame(imputer.fit_transform(X_encoded), columns=X_encoded.columns)
# Split dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_encoded, test_size=0.2, random_state=42)
# Train Random Forest classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
# Predict and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

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

```
Accuracy Score: 0.6927374301675978
Confusion Matrix:
 [[  3   0  40]
 [  0   2  15]
 [  0   0 119]]
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:

15/4/25

Lab-8 RF

Q Difference b/w decision tree and Random Forest

Ans

Feature	Decision Tree	Random Forest
Model Type	Single tree based model	Ensemble of multiple decision trees
Accuracy	Lower, prone to overfitting	Higher, better generalization
Overfitting	High risk, especially on noise data	Reduced due to averaging over many trees
Training Time	Fast (Simple to build one tree)	Slower (due to training multiple trees)
Interpretability	Easy to interpret and visualize	Harder to interpret (many trees)
Robustness	Sensitive to data changes	More robust due to ensemble voting

Q Parameters of RandomForestClassifier()

Ans 1) n_estimators → Number of trees in the forest

2) criterion → Function to measure split quality : "gini" or "entropy"

3) max_depth → Maximum depth of each tree

4) min_samples_split → minimum samples required to split a node

Bafna Gold

Q. Algorithm of Random Forest

1) Input : Dataset D with N Samples
Number of trees T
Number of features F of select per split

2) For each tree (repeat T time)
  1) Draw a bootstrap Sample from dataset
  2) Build a Decision Tree
    • At each Node
      • Randomly Select F features from total Set
      • Find the best feature
    • Continue until stopping criteria is met

3) Aggregate Predictions
For classification : majority vote from all trees
For Regression : avg of all tree outputs

4) Output

Bafna Gold

Code:

Income.csv

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, confusion_matrix

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

# Separate features and target
X_raw = df.iloc[:, :-1]
y_raw = df.iloc[:, -1]

# One-hot encode categorical features
X_encoded = pd.get_dummies(X_raw)

# Encode target if it's categorical
if y_raw.dtype == 'object':
    y_encoded = LabelEncoder().fit_transform(y_raw)
else:
    y_encoded = y_raw

# Handle missing values using most frequent strategy
imputer = SimpleImputer(strategy='most_frequent')
X_imputed = pd.DataFrame(imputer.fit_transform(X_encoded), columns=X_encoded.columns)
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_encoded, test_size=0.2, random_state=42)
model = AdaBoostClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy Score:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy Score: 0.8327362063670796
Confusion Matrix:
 [[7003  411]
 [1223 1132]]
```

Program 10

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

Screenshot:

Date: Page:

29/4/25

K-Means Code

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

iris = load_iris()
X = iris.data[:, 2:4]

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

inertia = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_cluster = K, random_state = 42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia)

plt.figure(figsize = (8, 5))
plt.plot(K_range, inertia, marker = 'o')
plt.xlabel('Number of cluster (K)')
plt.ylabel('Inertia (within-cluster sum of squares)')
plt.grid(True)
plt.show()
```

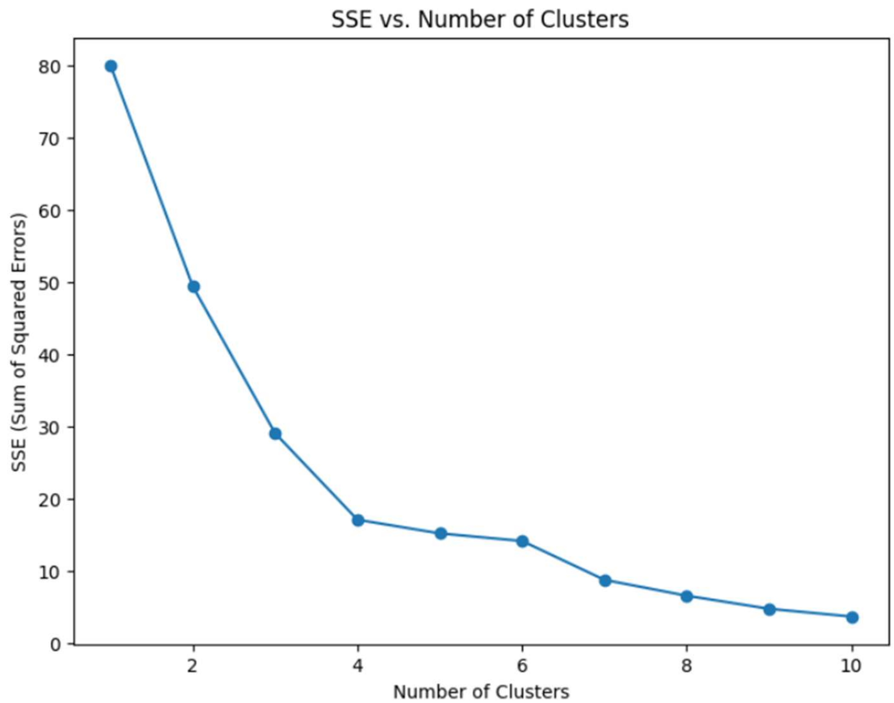
19/5/25

Bafna Gold

Code:

Income.csv

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
import matplotlib.pyplot as plt
import random
np.random.seed(42)
names = [f"Person_{i}" for i in range(1, 51)]
ages = np.random.randint(20, 60, 50)
incomes = np.random.randint(20000, 100000, 50)
df = pd.DataFrame({
    'Name': names,
    'Age': ages,
    'Income': incomes
})
df.to_csv('income.csv', index=False)
print("'income.csv' created successfully.")
from google.colab import files
files.download('income.csv')
data = pd.read_csv('income.csv')
X = data[['Age', 'Income']]
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)
    sse.append(kmeans.inertia_)
plt.plot(k_range, sse, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('SSE (Inertia)')
plt.title('Elbow Method: SSE vs Number of Clusters')
plt.grid(True)
plt.show()
k = 3  # Change this based on the elbow plot
model = KMeans(n_clusters=k, random_state=42)
model.fit(X_train_scaled)
train_preds = model.predict(X_train_scaled)
test_preds = model.predict(X_test_scaled)
true_labels_train = [random.randint(0, k-1) for _ in range(len(train_preds))]
accuracy = adjusted_rand_score(true_labels_train, train_preds)
print("Adjusted Rand Index (proxy accuracy):", round(accuracy, 2))
```

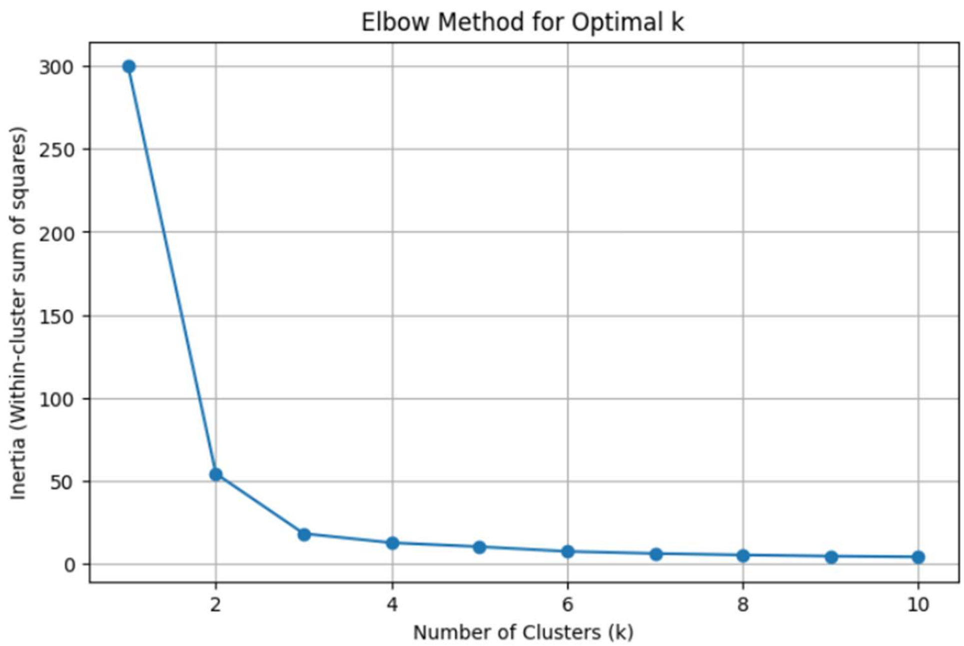



```
Silhouette Score (as a measure of clustering quality): 0.5103143530502396
```

Iris.csv

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data[:, 2:4]  # Only petal length and width

# Step 2: Preprocessing - Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Elbow method to find optimal k
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_)

# Step 4: Plot the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
```

```
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Within-cluster sum of squares)')
plt.title('Elbow Method for Optimal k')
plt.grid(True)
plt.show()
```



<u>**Program 11**</u>

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

Screenshot:

Date: Page:

29/4/25

PCA

x_1	4	8	13	7
x_2	11	4	5	14

$\bar{x}_1 = 8 \qquad \bar{x}_2 = 8.5$

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

$$= \frac{1}{3} \left((4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2\right)$$

$$\Rightarrow 14$$

29/4/25

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

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

$$= -11$$

$$Cov(x_2, x_2) = \frac{1}{3} \left[(11-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2\right]$$

$$= 23$$

$$S = \begin{bmatrix} Cov(x_1, x_1) & Cov(x_1, x_2) \\ Cov(x_2, x_1) & Cov(x_2, x_2) \end{bmatrix} = \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$$

Bafna Gold

Date: Page:

$0 = \det(S - \lambda I)$

$$\begin{vmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{vmatrix} = 0$$

$(14-\lambda)(23-\lambda) - (-11)(-11) = 0$

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

$\lambda = \frac{1}{2}(37 \pm \sqrt{565})$

$\lambda = 30.38, 6.6$

(λ_1, λ_2)

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

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda_1 I)x$$

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

$(14-\lambda_1)u_1 - 11u_2 = 0$

$-11u_1 + (23-\lambda_1)u_2 = 0$

$\frac{u_1}{11} = \frac{u_2}{14-\lambda_1} = t$

$u_1 = 11t \quad u_2 = (14-\lambda_1)t_1$

$$U_1 = \begin{bmatrix} 11 \\ 14-\lambda_1 \end{bmatrix}$$

Bafna Gold

$$e_1 = \begin{bmatrix} 11/\|u_1\| \\ 14-\lambda_1/\|u_1\| \end{bmatrix} = \begin{matrix} 11/19.73 \\ (14-30.38)/19.73 \end{matrix}$$

$$e_1 = \begin{bmatrix} 0.55 \\ -0.83 \end{bmatrix}$$

$$e_2 = \begin{bmatrix} 0.83 \\ 0.55 \end{bmatrix}$$

$$e_1^T \begin{bmatrix} X_{1k} - \bar{X}_1 \\ X_{2k} - \bar{X}_2 \end{bmatrix} = [0.55 \ -0.83] \begin{bmatrix} X_{1k} - \bar{X}_1 \\ X_{2k} - \bar{X}_2 \end{bmatrix}$$

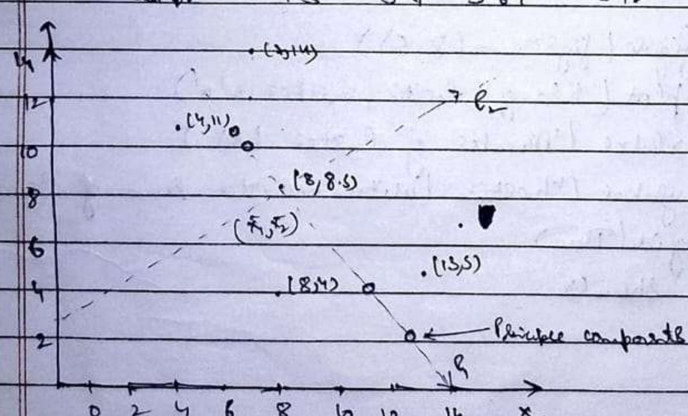
$$= 0.557(X_{1k} - \bar{X}_1) - 0.83(X_{2k} - \bar{X}_2)$$

$$= 0.55(4-8) - 0.83(11-8.5)$$

$$\begin{bmatrix} X_{11} \\ X_{21} \end{bmatrix} = \begin{bmatrix} 4 \\ 11 \end{bmatrix} \qquad = -4.3052$$

Final

X_1	4	8	13	7
X_2	11	4	5	14
PC Comp.	-4.3	3.7	5.69	-5.12



Code:

```
# Import required libraries
import numpy as np
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 the digits dataset
digits = load_digits()
X = digits.data
y = digits.target
# Step 2: Split the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 3: Perform scaling (standardization)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Step 4: Build PCA model 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 a Logistic Regression model
logreg = LogisticRegression(max_iter=10000)
logreg.fit(X_train_pca, y_train)
# Step 6: Predict on the test set
y_pred = logreg.predict(X_test_pca)
# Step 7: Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy score using PCA with 2 components: {accuracy}")
```

```
Accuracy score using PCA with 2 components: 0.5166666666666667
```

Heart.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from scipy.stats import zscore
# Step 1: Load dataset
df = pd.read_csv("heart.csv")  # Update path if needed
# Step 2: Remove outliers using Z-score
z_scores = np.abs(zscore(df.select_dtypes(include=[np.number])))
df_no_outliers = df[(z_scores < 3).all(axis=1)]
# Step 3: Convert text to numbers (if needed)
df_encoded = pd.get_dummies(df_no_outliers, drop_first=True)
# Step 4: Apply scaling
scaler = StandardScaler()
target_col = 'HeartDisease'  # Update if different
X = df_encoded.drop(target_col, axis=1)
y = df_encoded[target_col]
X_scaled = scaler.fit_transform(X)
# Step 5: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Step 5a: Train multiple models and compare
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier()
}
print("=== Accuracy without PCA ===")
```

```
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name}: {acc:.4f}")
# Step 6: PCA for dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2,
random_state=42)
print("\n=== Accuracy with PCA (2 components) ===")
for name, model in models.items():
    model.fit(X_train_pca, y_train_pca)
    y_pred = model.predict(X_test_pca)
    acc = accuracy_score(y_test_pca, y_pred)
    print(f"{name}: {acc:.4f}")
```

```
=== Accuracy without PCA ===
Logistic Regression: 0.8889
SVM: 0.8889
Random Forest: 0.8667

=== Accuracy with PCA (2 components) ===
Logistic Regression: 0.8722
SVM: 0.8611
Random Forest: 0.8333
```