VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Abhishek Gouda (1BM22CS006)

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 **Abhishek Gouda (1BM22CS006),** 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.

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

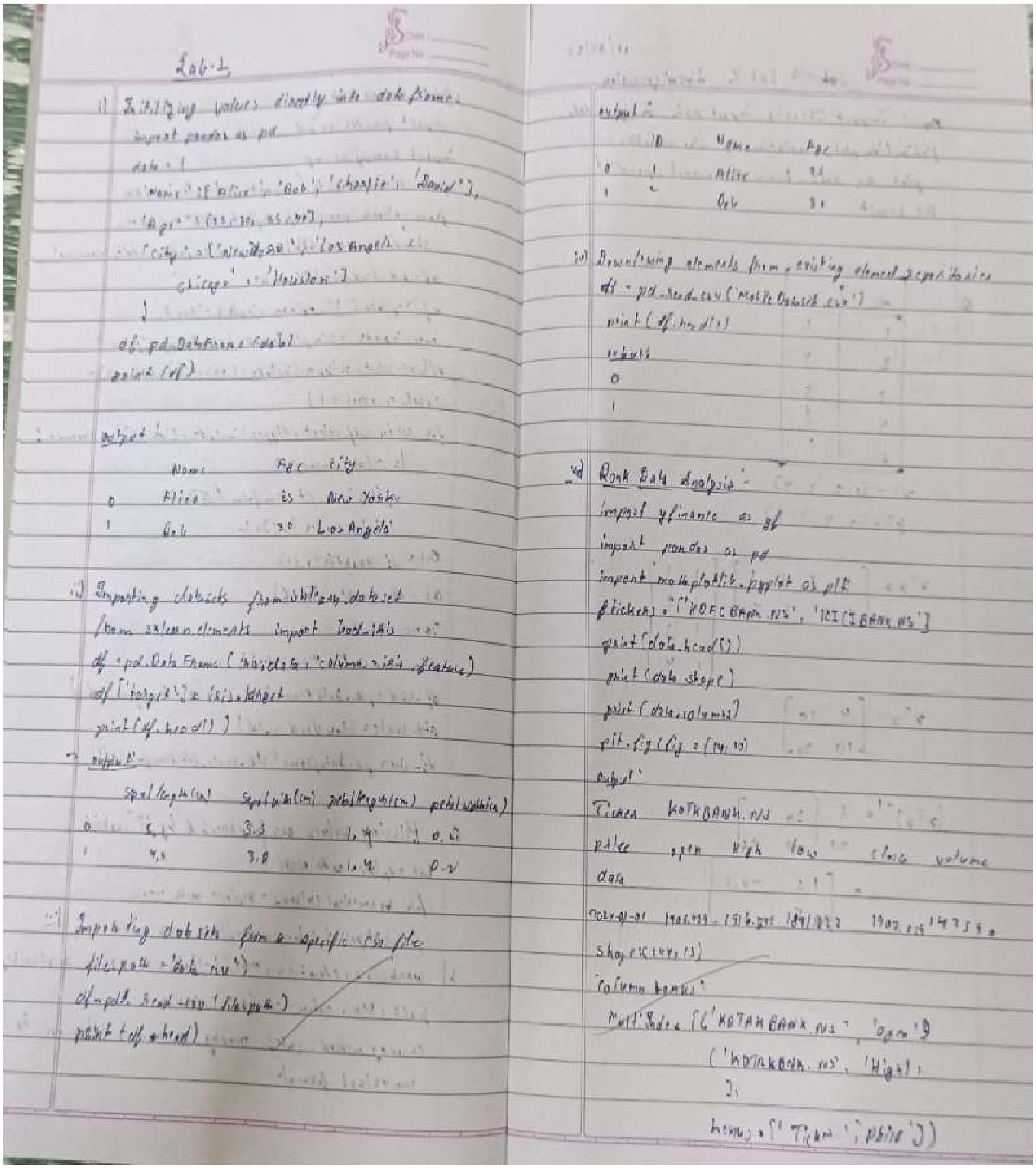
Index

Github Link: https://github.com/Abhii2404/6thSem-ML-Lab

Program 1

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

Screenshot



Code:

```
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("First 5 rows of the dataset:")

print(data.head())

print("\nShape of the dataset:")

print(data.shape)

print("\nColumn names:")

print(data.columns)

hdfc_data = data['HDFCBANK.NS']

print("\nSummary statistics for HDFC Bank:")

print(hdfc_data.describe())

hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

icici_data = data['ICICIBANK.NS']

print("\nSummary statistics for ICICI Bank:")

print(icici_data.describe())

icici_data['Daily Return'] = icici_data['Close'].pct_change()

kotak_data = data['KOTAKBANK.NS']

print("\nSummary statistics for Kotak Mahindra Bank:")

print(kotak_data.describe())
```

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

plt.figure(figsize=(14, 10))

plt.subplot(3, 2, 1)

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

plt.subplot(3, 2, 2)

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

plt.subplot(3, 2, 3)

icici_data['Close'].plot(title="ICICI Bank - Closing Price")

plt.subplot(3, 2, 4)

icici_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='orange')

plt.subplot(3, 2, 5)

kotak_data['Close'].plot(title="Kotak Mahindra Bank - Closing Price")

plt.subplot(3, 2, 6)

kotak_data['Daily Return'].plot(title="Kotak Mahindra Bank - Daily Returns", color='orange')

plt.tight_layout()

plt.show()


hdfc_data.to_csv('hdfc_bank_data.csv')

icici_data.to_csv('icici_bank_data.csv')

kotak_data.to_csv('kotak_bank_data.csv')


print("\nHDFC Bank data saved to 'hdfc_bank_data.csv'.")

print("ICICI Bank data saved to 'icici_bank_data.csv'.")

print("Kotak Bank data saved to 'kotak_bank_data.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot

Lab 2 - Data Processing 4/2/25

Demonstrate various data pre-processing techniques for a given data set

Housing.csv

```
import pandas as pd
i) file_path = 'housing.csv'
   df = pd.read_csv(file_path)

ii) print("Information about all columns:")
    print(df.info())
```

output:

```
. Information about all columns
  Range Index : 20640 entries, 0 to 20639
  Data columns (total 10 columns) :
     column                non-null count     Dtype
  0  longitude             20640 non null     float64
  1  latitude                "     "            "
  2  housing_median_age      "     "            "
  3  total_rooms             "     "            "
  4  total_bedrooms          "     "            "
  5  population              "     "            "
  6  households              "     "            "
  7  median_income           "     "            "
  8  house_value             "     "            "
  9  ocean_proximity         "     "          object
  dtypes : float64
```

iii) print ("statistical Information of numerical columns :")
print (df.describe ())
output:
statistical Information of numerical columns :

	longitude	latitude	housing_age	total_rooms	bedrooms
count	20640	20640	20640	20640	20640
mean	-119.5	35.631	28.639	2635.76	26433.1
std	2	2.135	12.58	2181.6	537.3

iv) Print ("count of unique labels for 'Ocean proximity':")
print (df['ocean_proximity'].value_counts())
output:
<1H Ocean 9136
INLAND 6551
NEAR OCEAN 2658
NEAR BAY 2290
ISLAND 5

v) print ("columns with missing values :")
print (df.isnull().sum()[df.isnull().sum() >0])
output:
columns with missing values :
total_bedrooms 207

Diabetes and Adult Income data sets

```
import pandas as pd
import numpy as np
diabetes_df = pd.read_csv('diabetes.csv')
adult_df = pd.read_csv('adult.csv')
```

1) Missing values

```
print(diabetes_df.isnull().sum())
print(adult_df.isnull().sum())
```

output

Missing values in diabetes dataset :

ID	0
No_Pation	0

Missing values in Adult income dataset :

age	0
workclass	0

```
from sklearn.processing import MinMaxScaler, standard
import matplotlib.pyplot as plt
file_path = 'diabetes.csv'
df = pd.readcsv(file_path)
df_num = df.select_dtypes(include=['number']).copy()
input = simpleImputer(strategy='mean')
df_num.iloc[:, :] = input.fit_transform(df_numeric)
df[df_num.columns] = df.numeric
Q1 = df_num.quantile(0.25)
Q3 = df_num.quantile(0.75)
IQR = Q3 - Q1
min_max_scaler = minMaxScaler(df)
```

```
standard_scaler = StandardScaler()
df_standard = pd.DataFrame(standard_scaler.fit_transform(df))
print("\n Dataset (MinMax Scaled):\n")
print(df_minmax.head())
print("\n Dataset (Standard Scaled):\n")
print(df_standard.head())
```

4) Which categorical columns did you identify? How did you encode them?
→ The categorical columns did you identifying encoding

5) What is the diff b/w minmax scaling and standardization? When would you use one over the other?
→ Min Max scaling transform data to find range [0,1]

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

used when : - dataset does not follow a normal dist
- features have diff ranges and need to be bound

Standardization transforms data to have zero mean and unit variance.

$$x' = \frac{x - \mu}{\sigma}$$

used when - dataset follows a Gaussian dist
- many ML algo assume normality

Adult_income.csv

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler,
  StandardScaler, LabelEncoder file_path = "adult_income.csv"
df = pd.read_csv(file_path)
df.replace("?", np.nan, inplace=True)
num_imputer = SimpleImputer(strategy="mean")
df[df.select_dtypes(include=["int64"]).columns] =
label_encoders = {}
for col in df.select_dtypes(include=["object"]).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
min_max = MinMaxScaler()
df_std = pd.DataFrame(std_scale.fit_transform(df))
std_scale = StandardScaler()
df_std = pd.DataFrame(std_scale.fit_transform(df))
```

1) Missing values are represented by "?" which we replace with NaN
for numerical columns - replace with mean
for categorical columns - replace with mode

2) workclass, education, marital-status, occupation, relationship, race, sex, native-country, income
Encoding method label encoding to convert categorical to numerical format.

Code:

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

diabetes_data = pd.read_csv('/content/Dataset of Diabetes .csv')
adult_income_data = pd.read_csv('/content/adult.csv')

print("Diabetes Dataset:")
print(diabetes_data.head())

print("\nAdult Income Dataset:")
print(adult_income_data.head())

diabetes_numerical_cols = diabetes_data.select_dtypes(include=[np.number]).columns
diabetes_categorical_cols = diabetes_data.select_dtypes(include=[object]).columns

diabetes_imputer_num = SimpleImputer(strategy='median')
diabetes_data[diabetes_numerical_cols] =
diabetes_imputer_num.fit_transform(diabetes_data[diabetes_numerical_cols])

diabetes_imputer_cat = SimpleImputer(strategy='most_frequent')
diabetes_data[diabetes_categorical_cols] =
diabetes_imputer_cat.fit_transform(diabetes_data[diabetes_categorical_cols])

adult_income_numerical_cols = adult_income_data.select_dtypes(include=[np.number]).columns
adult_income_categorical_cols = adult_income_data.select_dtypes(include=[object]).columns

adult_income_imputer_num = SimpleImputer(strategy='median')
adult_income_data[adult_income_numerical_cols] =
adult_income_imputer_num.fit_transform(adult_income_data[adult_income_numerical_cols])

adult_income_imputer_cat = SimpleImputer(strategy='most_frequent')
adult_income_data[adult_income_categorical_cols] =
adult_income_imputer_cat.fit_transform(adult_income_data[adult_income_categorical_cols])

categorical_columns_adult = adult_income_data.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()

for col in categorical_columns_adult:
    adult_income_data[col] = label_encoder.fit_transform(adult_income_data[col])
```

```
def detect_and_remove_outliers(df):
    numerical_df = df.select_dtypes(include=[np.number])
    Q1 = numerical_df.quantile(0.25)
    Q3 = numerical_df.quantile(0.75)
    IQR = Q3 - Q1
    return df[~((numerical_df < (Q1 - 1.5 * IQR)) | (numerical_df > (Q3 + 1.5 * IQR))).any(axis=1)]

diabetes_data_cleaned = detect_and_remove_outliers(diabetes_data)
adult_income_data_cleaned = detect_and_remove_outliers(adult_income_data)

min_max_scaler = MinMaxScaler()

diabetes_numerical_cols = diabetes_data_cleaned.select_dtypes(include=[np.number]).columns
diabetes_data_normalized = diabetes_data_cleaned.copy()

diabetes_data_normalized[diabetes_numerical_cols] =
min_max_scaler.fit_transform(diabetes_data_cleaned[diabetes_numerical_cols])

adult_income_numerical_cols =
adult_income_data_cleaned.select_dtypes(include=[np.number]).columns
adult_income_data_normalized = adult_income_data_cleaned.copy()

adult_income_data_normalized[adult_income_numerical_cols] =
min_max_scaler.fit_transform(adult_income_data_cleaned[adult_income_numerical_cols])

standard_scaler = StandardScaler()

diabetes_data_standardized = diabetes_data_cleaned.copy()
diabetes_data_standardized[diabetes_numerical_cols] =
standard_scaler.fit_transform(diabetes_data_cleaned[diabetes_numerical_cols])

adult_income_data_standardized = adult_income_data_cleaned.copy()
adult_income_data_standardized[adult_income_numerical_cols] =
standard_scaler.fit_transform(adult_income_data_cleaned[adult_income_numerical_cols])
```

Program 3

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

Screenshot:

Lab-5.

Date: 1/4/25

Page No.:

Instance	A_2	A_3	Classification
1	Hot	High	No
2	Hot	High	No
6	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes

$$\text{Entropy} = -\frac{4}{5}\log\left(\frac{4}{5}\right) - \frac{1}{5}\log\left(\frac{1}{5}\right)$$

$$= 0.7219$$

For a_2,

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

$$= 0.8113$$

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

$\text{Gain}(S, a_2) = 0.7219 - \frac{4}{5} \times 0.8113 - 0 = 0.07286$

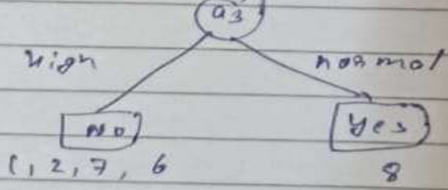
For a_3,

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

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

$\text{Gain}(S, a_3) = 0.7219$

∴ a_3 has highest gain value, it is taken as root



```
dtc = DecisionTreeClassifier()
dtc.fit (X_train , y_train)
y_pred = dtc.predict (x_test)

X_petrol = petrol.iloc[:,:-1]
y_petrol = petrol.iloc[:,-1]
X_train, X_test, y_train, y_test =
train_test_split (x_petrol, y_petrol, test_size=0.2,
                  random_state=42)
dtr = DecisionTreeRegressor()
dtc.fit (x_train, y_train)
y_pred = dtr.predict (x_test)

print (mean_absolute_error (y_test, y_pred))
```

Output)

Decision tree classification for Iris

Accuracy : 1.0

Confusion Matrix :

```
[[10  0  0]
 [0   0  0]
 [0   0  11]]
```

classification report

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1	9
virginica	1.00	1.00	1	11
accuracy	1	1		30
macro avg	1	1	1	30
weighted avg	1	1	1	30

Decision tree for drug:
Accuracy : 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]]

Classification Report:

	precision	recall	f1 score	support
Drug A	1	1	1	6
B	1	1	1	3
C	1	1	1	5
X	1	1	1	11
Y	1	1	1	15
accuracy			1	40
macro avg	1	1	1	40
weighted avg	1	1	1	40

Decision Tree regressor for Petrol consumption
Mean Absolute error : 92.9
Mean square Error : 17857.3
Root Mean square Squared Error : 133.6312089

Code:

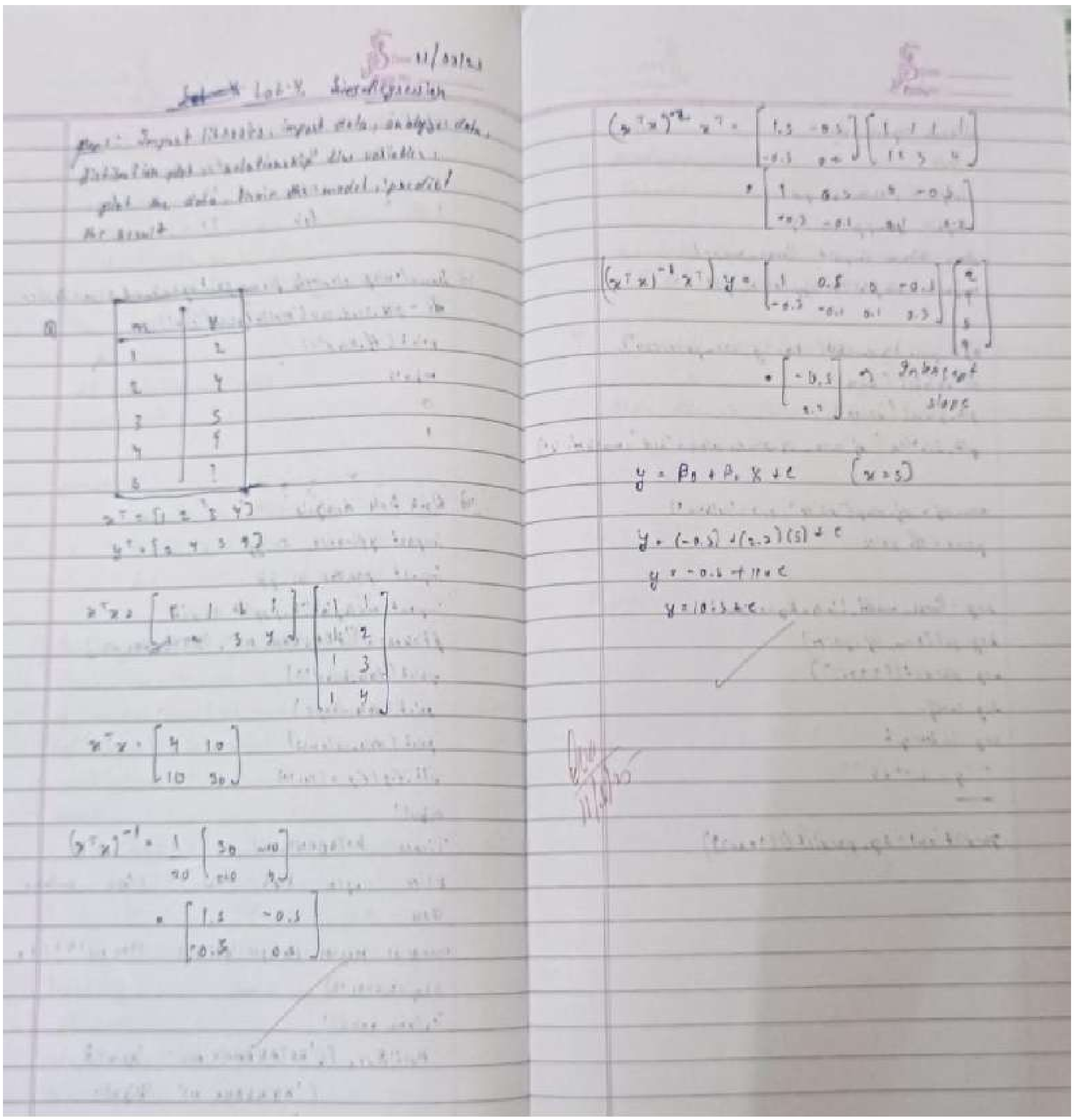
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
mean_absolute_error,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

iris = pd.read_csv("/content/iris (4).csv")
drug = pd.read_csv("/content/drug.csv")
petrol = pd.read_csv("/content/petrol_consumption.csv")

X_iris = iris.iloc[:, :-1]
y_iris = iris.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print("Decision Tree Classification for IRIS Dataset:")
print("Accuracy:", 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))

X_drug = drug.iloc[:, :-1]
y_drug = drug.iloc[:, -1]

le = LabelEncoder()

for col in X_drug.select_dtypes(include=['object']).columns:
    X_drug[col] = le.fit_transform(X_drug[col])

X_train, X_test, y_train, y_test = train_test_split(X_drug, y_drug, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print("\nDecision Tree Classification for Drug Dataset:")
print("Accuracy:", 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))

X_petrol = petrol.iloc[:, :-1]
```

```
y_petrol = petrol.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_petrol, y_petrol, test_size=0.2, random_state=42)

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

print("\nDecision Tree Regression for Petrol Consumption:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



[illegible]

Linear Regression

Code:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv("housing_area_price.csv")
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area, df.price, color='red', marker='+')

new_df = df.drop('price', axis='columns')
price = df.price

reg = linear_model.LinearRegression()
reg.fit(new_df, price)
reg.predict([[3300]])
reg.coef_
reg.intercept_
"y = m*x+b"

predict_val = reg.predict([[3000]])
```

Output:

1) predicted per capita income for Canada in 2020 : 41,037.6724,896

predicted salary for 12 years of experience
=> 140337.54153839303

2) Mean Absolute Error (canada) - 3240.9539747

Mean Absolute Error (Salary) : 4519.14062351313

Multiple

```
import pandas as pd
import numpy as np
from sklearn import linear_model
df = pd.read_csv('homeprices_Multiple.csv')
df
df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
df

reg = multiple_model.LinearRegression()
reg.fit(df.drop('price', axis='columns'),
        df.price)

reg.coef_
reg.intercept_

reg.predict([[3000, 3, 40]])
```

Output.

1) predicted salary for candidate 1 (2 yr experience, 9 test score, 6 interview score)

   53356.50000001

2) Predicted salary for candidate 2 (12 yr experience, 10 test score, 10 interview score): 92582.40999

3) Mean Absolute Error for salary prediction: 667
   Predicted profit for the given inputs
   (Florida State) 155400.6928352
   Mean absolute for profit prediction:
   1278.4674492333337

Code:

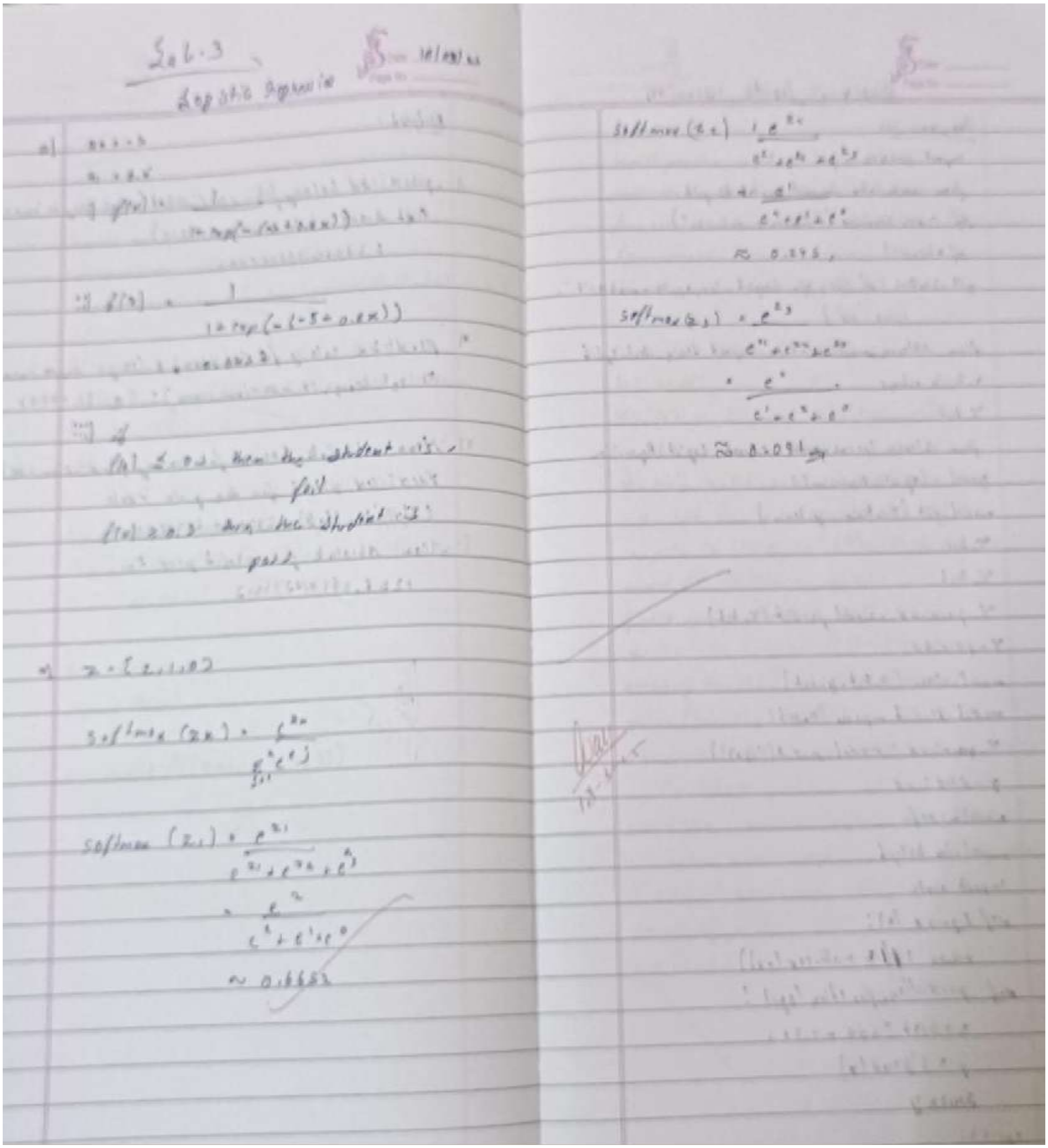
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

hiring_data = pd.read_csv('hiring.csv')
print(hiring_data.head())
hiring_data = hiring_data.dropna()

experience_mapping = {
    'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'seven': 7, 'eight': 8,
    'nine': 9, 'ten': 10, 'eleven': 11, 'twelve': 12, 'thirteen': 13, 'fourteen': 14,
}

hiring_data['experience'] = hiring_data['experience'].replace(experience_mapping)
hiring_data['experience'] = pd.to_numeric(hiring_data['experience'], errors='coerce')

if hiring_data['experience'].isnull().any():
    print("Warning: There are still non-numeric values in the 'experience' column.")
    hiring_data = hiring_data.dropna(subset=['experience'])

X_hiring = hiring_data[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y_hiring = hiring_data['salary($)']

X_train_hiring, X_test_hiring, y_train_hiring, y_test_hiring = train_test_split(X_hiring, y_hiring,
test_size=0.2, random_state=42)

regressor_hiring = LinearRegression()
regressor_hiring.fit(X_train_hiring, y_train_hiring)

candidate_1 = np.array([[2, 9, 6]])
candidate_2 = np.array([[12, 10, 10]])

salary_1 = regressor_hiring.predict(candidate_1)
salary_2 = regressor_hiring.predict(candidate_2)

print(f"Predicted salary for candidate 1 (2 yr experience, 9 test score, 6 interview score):
{salary_1[0]}")
print(f"Predicted salary for candidate 2 (12 yr experience, 10 test score, 10 interview score):
{salary_2[0]}")
```

```
companies_data = pd.read_csv('/content/1000_Companies.csv')
print(companies_data.head())
companies_data = companies_data.dropna()

label_encoder = LabelEncoder()
companies_data['State'] = label_encoder.fit_transform(companies_data['State'])

X_companies = companies_data[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]
y_companies = companies_data['Profit']

X_train_companies, X_test_companies, y_train_companies, y_test_companies =
train_test_split(X_companies, y_companies, test_size=0.2, random_state=42)

regressor_companies = LinearRegression()
regressor_companies.fit(X_train_companies, y_train_companies)

input_data = np.array([[91694.48, 515841.3, 11931.24, label_encoder.transform(['Florida'])[0]]])
predicted_profit = regressor_companies.predict(input_data)

print(f"Predicted profit for the given inputs (Florida State): {predicted_profit[0]}")

y_pred_hiring = regressor_hiring.predict(X_test_hiring)
mae_hiring = mean_absolute_error(y_test_hiring, y_pred_hiring)
print(f"Mean Absolute Error for Salary Prediction: {mae_hiring}")

y_pred_companies = regressor_companies.predict(X_test_companies)
mae_companies = mean_absolute_error(y_test_companies, y_pred_companies)
print(f"Mean Absolute Error for Profit Prediction: {mae_companies}")
```

Program 5

Build Logistic Regression Model for a given dataset

Screenshot



Code:

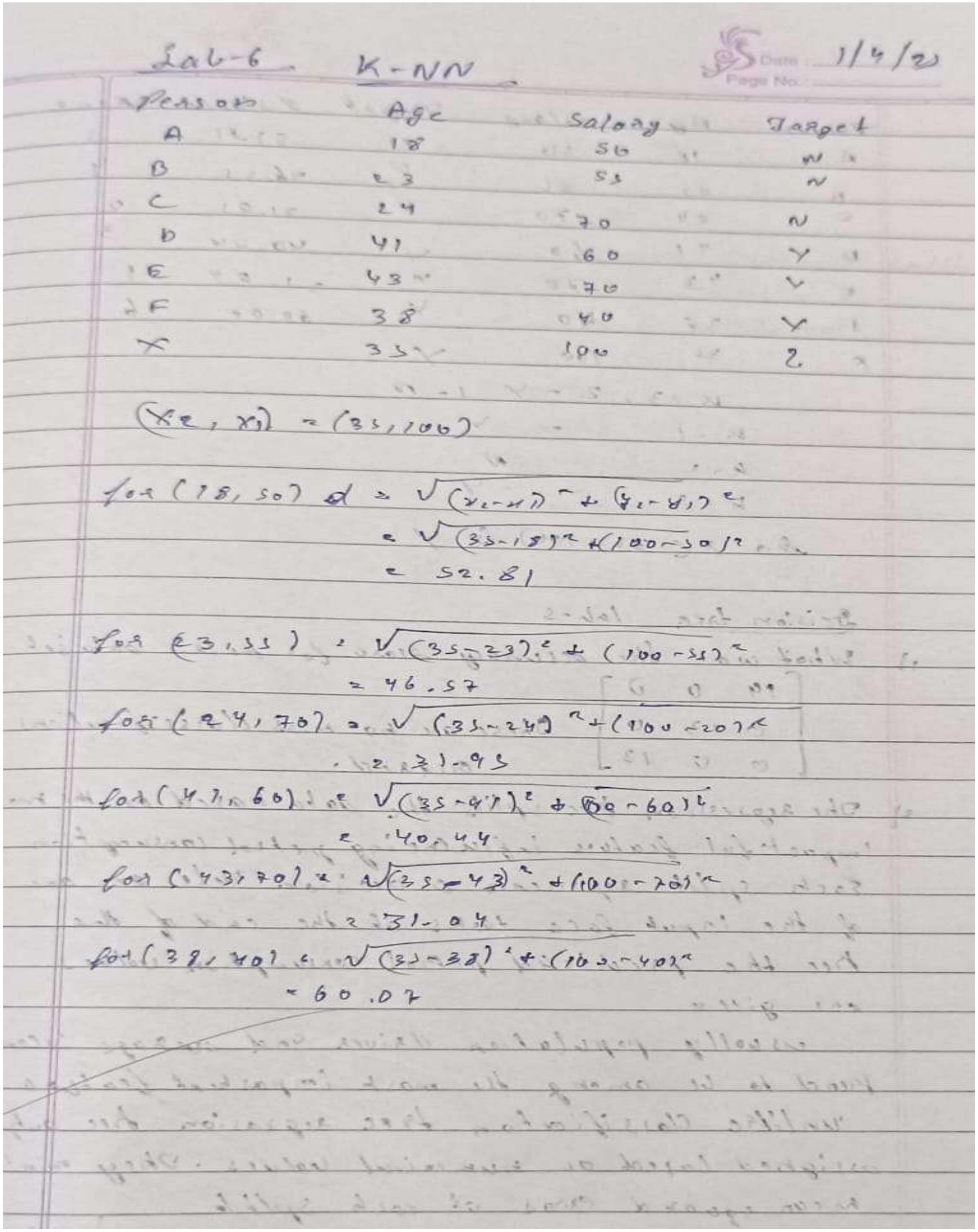
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

file_path = 'HR_comma_sep.csv'
data = pd.read_csv(file_path)

print(data.info())

print(data.head())

print(data.describe())

plt.figure(figsize=(8, 5))
sns.countplot(x='salary', hue='left', data=data)
plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary')
plt.ylabel('Count')
plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])
plt.show()

plt.figure(figsize=(10, 6))
sns.countplot(x='Department', hue='left', data=data)
plt.title('Impact of Department on Employee Retention')
plt.xlabel('Department')
plt.ylabel('Count')
plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])
plt.xticks(rotation=45)
plt.show()

data_encoded = pd.get_dummies(data, columns=['salary', 'Department'], drop_first=True)

print(data_encoded.info())

X = data_encoded.drop('left', axis=1)
y = data_encoded['left']

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
logreg = LogisticRegression(max_iter=1000)

logreg.fit(X_train_scaled, y_train)

y_pred = logreg.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Logistic Regression Model: {accuracy * 100:.2f}%")

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['Stayed', 'Left'],
yticklabels=['Stayed', 'Left'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot



Lab-6 K-NN Date 1/4/2?

Person	Age	Salary	Target
A	18	50	N
B	23	55	N
C	24	70	N
D	41	60	Y
E	43	70	Y
F	38	40	Y
X	35	100	?

$(x_2, x_1) = (35, 100)$

for $(18, 50)$ $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

$= \sqrt{(35-18)^2 + (100-50)^2}$

$= 52.81$

for $(23, 55)$ $= \sqrt{(35-23)^2 + (100-55)^2}$

$= 46.57$

for $(24, 70)$ $= \sqrt{(35-24)^2 + (100-70)^2}$

$= 31.95$

for $(41, 60)$ $= \sqrt{(35-41)^2 + (100-60)^2}$

$= 40.44$

for $(43, 70)$ $= \sqrt{(35-43)^2 + (100-70)^2}$

$= 31.04$

for $(38, 40)$ $= \sqrt{(35-38)^2 + (100-40)^2}$

$= 60.07$

Page No.:

Person	Age	salary	Danger	distance	Rank
A	18	50	N	52.81	5
B	23	55	N	46.51	4
C	24	70	N	31.93	2
D	41	60	Y	40.44	3
E	43	70	Y	21.04	1
F	38	40	Y	60.07	6
X	35	100	Y		

K=3, 2 - Y 1 - N

K=1 - Y

K=2 - N

K=3 - Y

So X(35, 100) is Y.

Lab-6 KNN

1) import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns

heart_df = pd.read_csv("heart.csv")
print(heart_df.head())
X = heart_df.drop(columns=['target'])
y = heart_df['target']
X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
k_values = range(1, 21)
best_k = 1
best_score = 0
for k in k_values:
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_scaled, y_train)
y_pred = knn.predict(X_test_scaled)
if score > best_score:
best_score = score
best_k = k
accuracy = accuracy_score(y_test, y_pred_test)
print(f"accuracy on test data: {accuracy*100:.2f}")

Output:
Accuracy on test data : 91.80%

Output
• Accuracy score of Iris test data 100.00%
• Accuracy on diabetes test data 69.48%

1) By plotting accuracy and error rate vs various k values select the k with highest accuracy and lowest error rate. This value is considered optimal for dataset

2) Feature scalling ensures that all features contribute equally to the model especially for distance based algorithms like KNN. It prevents features with large values from dominating the learning process
How to perform it
Standard scaling : Transform feature to have mean = 0 and standard deviation = 1
min-max scalling . scales features to a fixed range usually [0,1]

These scaling techniques help improve model performance and convergence speed

Code:

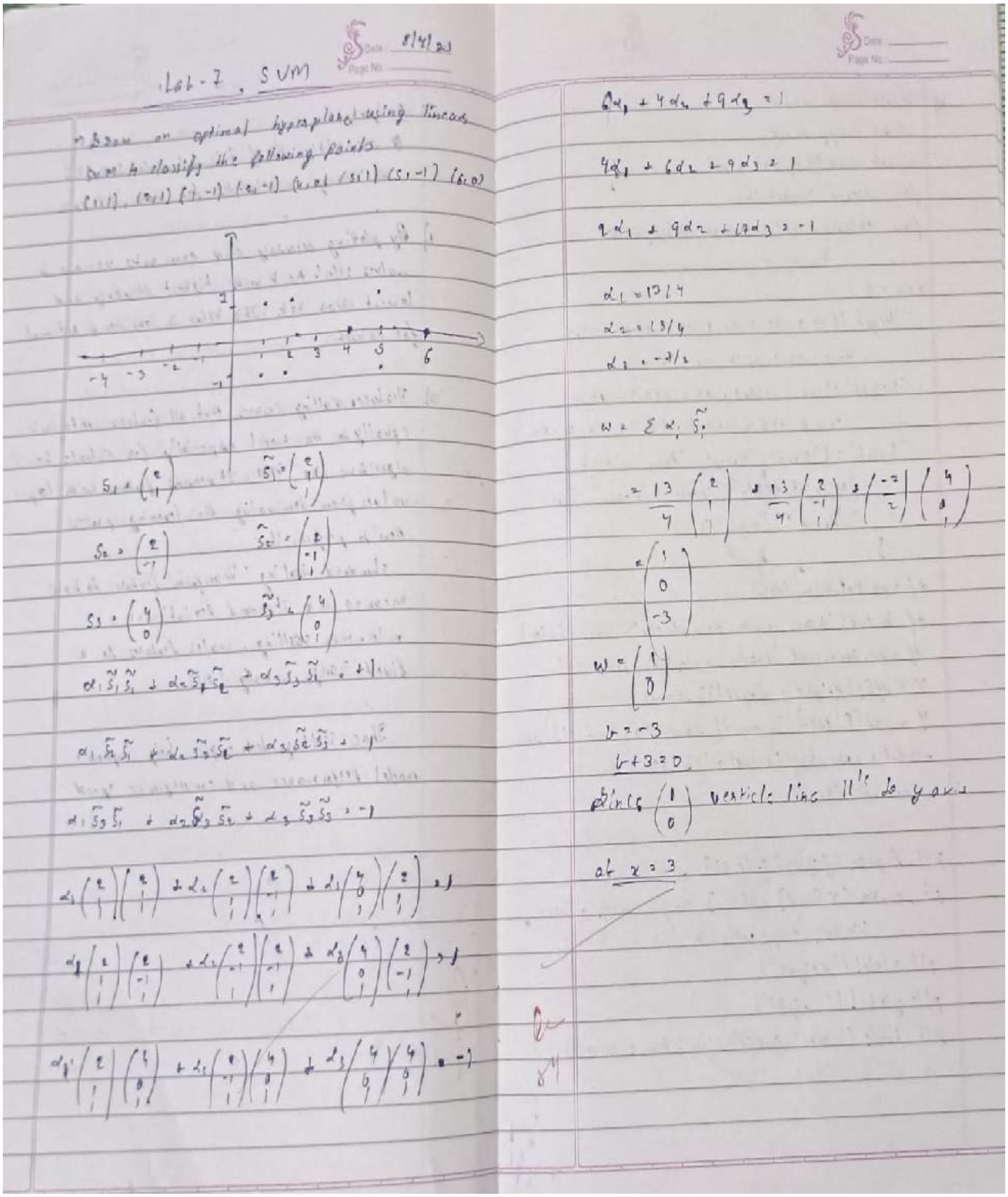
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot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

iris_df = pd.read_csv('/content/iris (3).csv')

print(iris_df.head())

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)

scaler = StandardScaler()
X_train_iris = scaler.fit_transform(X_train_iris)
X_test_iris = scaler.transform(X_test_iris)

knn_iris = KNeighborsClassifier(n_neighbors=3)

knn_iris.fit(X_train_iris, y_train_iris)

y_pred_iris = knn_iris.predict(X_test_iris)

accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
print(f"Accuracy on Iris test data: {accuracy_iris * 100:.2f}%")

cm_iris = confusion_matrix(y_test_iris, y_pred_iris)
sns.heatmap(cm_iris, annot=True, fmt="d", cmap="Blues", xticklabels=knn_iris.classes_,
yticklabels=knn_iris.classes_)
plt.title("Confusion Matrix for Iris Dataset")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report for Iris Dataset:")
print(classification_report(y_test_iris, y_pred_iris))

diabetes_df = pd.read_csv('diabetes.csv')
print(diabetes_df.head())
```

```
X_diabetes = diabetes_df.drop(columns=['Outcome'])
y_diabetes = diabetes_df['Outcome']

X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes = train_test_split(X_diabetes,
y_diabetes, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_diabetes = scaler.fit_transform(X_train_diabetes)
X_test_diabetes = scaler.transform(X_test_diabetes)

knn_diabetes = KNeighborsClassifier(n_neighbors=5)

knn_diabetes.fit(X_train_diabetes, y_train_diabetes)

y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)

accuracy_diabetes = accuracy_score(y_test_diabetes, y_pred_diabetes)
print(f"Accuracy on Diabetes test data: {accuracy_diabetes * 100:.2f}%")

cm_diabetes = confusion_matrix(y_test_diabetes, y_pred_diabetes)
sns.heatmap(cm_diabetes, annot=True, fmt="d", cmap="Blues", xticklabels=knn_diabetes.classes_,
yticklabels=knn_diabetes.classes_)
plt.title("Confusion Matrix for Diabetes Dataset")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report for Diabetes Dataset:")
print(classification_report(y_test_diabetes, y_pred_diabetes))
```

Program 7

Build Support vector machine model for a given dataset

Screenshot



Date: 8/4/23

Lab-7 . SVM

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

(1,1), (2,1) (1,-1) (2,-1) (4,0) (5,1) (5,-1) (6,0)

$S_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ $\tilde{S}_1 = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$

$S_2 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$ $\tilde{S}_2 = \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$

$S_3 = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$ $\tilde{S}_3 = \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}$

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

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

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

$\alpha_1 \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} = 1$

$\alpha_1 \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} = 1$

$\alpha_1 \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} = -1$

$6\alpha_1 + 4\alpha_2 + 9\alpha_3 = 1$

$4\alpha_1 + 6\alpha_2 + 9\alpha_3 = 1$

$9\alpha_1 + 9\alpha_2 + 17\alpha_3 = -1$

$\alpha_1 = 13/4$

$\alpha_2 = 13/4$

$\alpha_3 = -7/2$

$w = \sum \alpha_i \tilde{S}_i$

$= \frac{13}{4}\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + \frac{13}{4}\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} + \left(\frac{-7}{2}\right)\begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}$

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

$w = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$b = -3$

$b + 3 = 0$

Since $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ vertical line ll' to y axis

at $x = 3$

Code:

```
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, roc_auc_score, roc_curve
from sklearn.preprocessing import LabelEncoder, label_binarize
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

df = pd.read_csv("/content/letter-recognition.csv")

top_classes = df['letter'].value_counts().head(5).index.tolist()
df = df[df['letter'].isin(top_classes)]

X = df.iloc[:, 1:]
y = df.iloc[:, 0]

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

y_bin = label_binarize(y_encoded, classes=np.unique(y_encoded))
n_classes = y_bin.shape[1]

X_train, X_test, y_train, y_test_bin = train_test_split(X, y_bin, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear', probability=True)
svm_model.fit(X_train, y_train.argmax(axis=1))
y_score = svm_model.predict_proba(X_test)

y_pred = svm_model.predict(X_test)
y_true = y_test_bin.argmax(axis=1)

print("Accuracy:", accuracy_score(y_true, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))

plt.figure()
for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    auc = roc_auc_score(y_test_bin[:, i], y_score[:, i])
    plt.plot(fpr, tpr, label=f"{label_encoder.inverse_transform([i])[0]} AUC={auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve (Top 5 Classes)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.tight_layout()
```

```python
plt.show()

macro_auc = roc_auc_score(y_test_bin, y_score, average="macro")
print("Macro AUC Score:", macro_auc)
```

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot

Lab-8

Date: 15/04/25
Page No.:

Q1 Get the difference in decision tree and random forest classifier.

→

Decision tree	Random forest
- A decision tree is a single tree structure where decisions are made by splitting the dataset at each node	• It is an ensemble learning method method that builds multiple decision trees and combines their results
• These are highly prone to overfitting, especially with complex datasets	These reduce overfitting by averaging multiple decision trees, which generally improves performance
- A model has high variance and low bias.	- A models tend to have low variance and moderate bias as they combine the predictions from many trees

Q2 Discuss all the parameters of random forest classifier

→ n_estimators: The no of trees in the forest

• criterion: The functions to measure the quality of a split

- max_depth: The max depth of the tree

- min_samples_split: The min no of sample required to split an internal node

- min_samples_leaf: The min no of samples required to be featured in leaf node

- max_features: The no of features to consider when looking for the best split

- bootstrap: whether to use bootstrapping when creating trees

- oob_score: whether to use out-of-bag samples to estimate the generalization accuracy.

- n_jobs: The no of jobs to run in parallel for both fit() and predict()

- random_state: Controls the randomness of the estimator

Page No.

8) Algorithm of Random forest

→ Step 1: The data set is divided into input features (X) and the target labels (Y)

Step 2: For each tree, create a bootstrap sample from the dataset

Step 3: For each tree, randomly select a subset of features, It is determined by max-features parameter

Step 4: Build a decision tree on the bootstrapped dataset using the selected subset of features

Step 5: Once all trees are built, make predictions by aggregating the predictions of all the trees

Step 6: During training, the data points that were not model's generalization performance

Step 7: The final model is evaluated using metrics like accuracy, confusion matrix and AUC score, depending on the task at hand.

Code:

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

df = pd.read_csv('/content/train.csv')

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

for column in X.columns:
    if X[column].dtype == 'object':
      le = preprocessing.LabelEncoder()
      X[column] = le.fit_transform(X[column])

if y.dtype == 'object':
  le = preprocessing.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)

rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot

Lab-9 : AdaBoost

Q) What is Boosting?

→ Boosting is an ensemble learning technique that combines multiple weak learners to form a strong learner. It works by training models sequentially each trying to correct the errors made by the previous ones.

Q) Discuss all the parameters of AdaBoost classifier?

→ - estimator: The base estimator to be boosted.

- n_estimators: The max no. of estimators to be used
- learning rate: shrinks the contribution of each classifier
- algorithm: The boosting algorithm. options are 'SAMME' and 'SAMME.R'. typically performs better as it uses probabilities
- random_state: controls the random number generator for reproducibility.
- base_estimator: previously used to specify the weak learner.

Q) Write an algorithm of AdaBoost.

→ Step 1: Initialize the weights for all training instances equally.

Step 2: For each iteration t from 1 to T:

Step 3: Train a weak learner ht(x) on the training data with weighted samples.

Step 4: Compute the error rate: error_t = sum(w_i * 1(y_i != ht(x_i)))

Step 5: Compute model's weight of the training samples

w_i = w_i * exp(-α_t * y_i * ht(x_i))

Step 7: Normalize the weights so they sum to 1

Step 8: H(x) = sign (Σ α_t * ht(x))

Code:

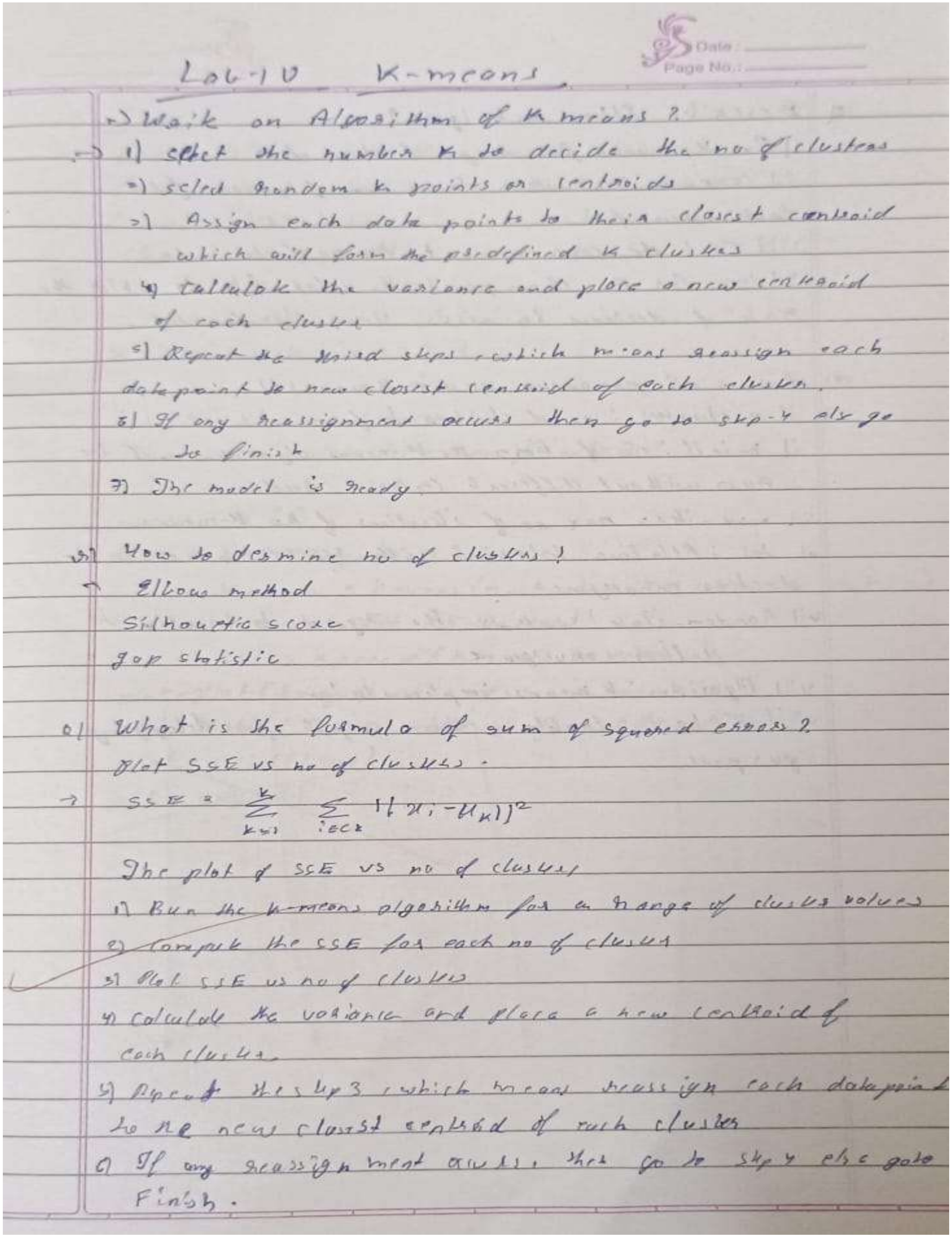
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X = iris.data
y = iris.target

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

results = []

n_estimators_list = [10, 50, 100]
learning_rates = [0.01, 0.1, 1]

for n in n_estimators_list:
    for lr in learning_rates:
        tree_base = DecisionTreeClassifier(max_depth=1)
        model = AdaBoostClassifier(estimator=tree_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'DecisionTree',
            'n_estimators': n,
            'learning_rate': lr,
            'Accuracy': acc
        })

for n in n_estimators_list:
    for lr in learning_rates:
        log_reg_base = LogisticRegression(max_iter=1000)
        model = AdaBoostClassifier(estimator=log_reg_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'LogisticRegression',
```

```
            'n_estimators': n,
            'learning_rate': lr,
            'Accuracy': acc
        })

results_df = pd.DataFrame(results)
print(results_df)

import seaborn as sns
plt.figure(figsize=(12, 6))
sns.barplot(x='n_estimators', y='Accuracy', hue='Base', data=results_df, ci=None)
plt.title('AdaBoost Accuracy with Different Estimators and n_estimators')
plt.show()
```

Program 10

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

Screenshot



Lab-10 K-means

Date :
Page No.:

-) Work on Algorithm of K means ?

-> 1) select the number K to decide the no of clusters

2) select random K points or centroids

3) Assign each data points to their closest centroid which will form the predefined K clusters

4) calculate the variance and place a new centroid of each cluster

5) Repeat the third steps, which means reassign each datapoint to new closest centroid of each cluster.

6) If any reassignment occurs then go to step-4 else go to finish

7) The model is ready.

Q) How to determine no of clusters ?

-> Elbow method

Silhouette score

gap statistic

Q) What is the formula of sum of squared errors ? plot SSE vs no of clusters.

-> $SSE = \sum_{k=1}^{k} \sum_{i \in C_k} ||x_i - \mu_k||^2$

The plot of SSE vs no of clusters

1) Run the k-means algorithm for a range of cluster values

2) Compute the SSE for each no of clusters

3) Plot SSE vs no of clusters

4) Calculate the variance and place a new centroid of each cluster.

5) Repeat the step 3, which means reassign each datapoint to the new closest centroid of each cluster

6) If any reassignment occurs, then go to step 4 else goto Finish.

Q Describe Elbow technique
i] Run K-means for range of values of k
ii] compute the within cluster sum of squares for each value of k
iii] Plot the wcss against the no of cluster (k)
iv] look for the elbow point - the value of k at which the rate of decrease in wcss slows significantly

Q) Discuss all parameters used in kmeans()
i] n-clusters : no of clusters to form
ii] n-init : no of times the k-means algorithm will be run without different centroid seeds
iii] max-iter : max no of iterations of the k-means.
v] tol : Relative tolerance with regards to inertia to declare convergence
vi] random-state : controls the regards to inertia to declare convergence
vii] Algorithm : k means implementation
viii] verbose : Enable verbose output for debugging purpose

Q. Cluster eight points with (x,y) representing locations into 3 clusters $A_1(2,10)$, $A_2(2,5)$, $A_3(8,4)$, $A_4(5,8)$, $A_5(7,5)$, $A_6(6,4)$, $A_7(1,2)$, $A_8(4,9)$

Initial clusters: $A_1(2,10)$, $A_4(5,8)$, $A_7(1,2)$

→ Calculation of distance b/w points $A_1(2,10)$ and $C_1(2,10)$

$P(A_1, C_1) = |x_2 - x_1| + |y_2 - y_1| = |2-2| + |10-10| = 0$

Calculating dis b/w $A_1(2,10)$ and $C_2(5,8)$:

$P(A_1, C_2) = |5-2| + |8-10| = 3+2 = 5$

Calculating distance b/w $A_1(2,10)$ and $C_3(1,2)$:

$P(A_1, C_3) = |2-1| + |10-2| = 9$

Illy we calculate the distance of other points from each of the centers of 3 clusters.

for cluster-01

- we have only one point $A_1(2,10)$ in cluster-01

→ cluster center remains the same

Iteration	Given pts	Distance from C_1	Distance from C_2	Distance from C_3	Point belongs to
	$A_1(2,10)$	0	5	9	C_1
	$A_2(2,5)$	5	6	4	C_3
	$A_3(8,4)$	12	7	9	C_2
	$A_4(5,8)$	5	0	10	C_2
	$A_5(7,5)$	10	5	9	C_2
	$A_6(6,4)$	10	5	7	C_2
	$A_7(1,2)$	9	10	0	C_3
	$A_8(4,9)$	3	2	10	C_2

For cluster-02

center of cluster-02 = [illegible]

$(8+5+7+6+4)/5$, $(4+8+5+4+9)/5$ = (6, 6)

For cluster-03 = $((2+1)/2, (5+2)/2)$

= (1.5, 3.5). This is completion of Iteration-01

Iteration-02

Given pts	Dist from C_1	Dist from C_2	Dist from C_3	Point belongs to cluster
$A_1(2,10)$	0	8	7	C_1
$A_2(2,5)$	5	5	2	C_3
$A_3(8,4)$	12	4	7	C_2
$A_4(5,8)$	5	3	8	C_2
$A_5(7,5)$	10	2	7	C_2
$A_6(6,4)$	10	2	5	C_2
$A_7(1,2)$	9	9	2	C_3
$A_8(4,9)$	3	5	8	C_1

For C_1: center = $((2+4)/2, (10+9)/2)$ = (3, 9.5)

For C_2: Center = $((8+5+7+6)/4, (4+8+5+4)/4)$ = (6.5, 5.25)

For C_3: center = $((2+1)/2, (5+2)/2)$ = (1.5, 3.5)

Date ______
Page No. ______

Iteration-03

Given Pts	Distance from C_1	Dist from C_2	Dist from C_3	Point belongs to cluster
$A_1(2,10)$	1.5	9.25	7	C_1
$A_2(2,5)$	5.5	4.75	2	C_3
$A_3(8,4)$	10.5	2.75	7	C_2
$A_4(5,8)$	3.5	4.75	8	C_1
$A_5(7,5)$	8.5	0.75	7	C_2
$A_6(6,4)$	8.5	1.75	5	C_2
$A_7(1,2)$	9.5	8.75	2	C_3
$A_8(4,9)$	1.5	6.25	8	C_1

For C_1 ((2+5+4)/3 = 3.66 , (10+8+9)/3 = 9

For C_2 (8+7+6)/3 = 7 , (4+5+4)/3 = 4.33

For C_3 (2+1)/2 = 1.5 , 3.5

Iteration

Given points	Dist from C_1	Dist from C_2	Dist from C_3	Point
$A_1(2,10)$	1.66	10.67	8	C_1
$A_2(2,5)$	5.66	5.67	2	C_3
$A_3(8,4)$	7.34	1.33	7	C_2
$A_4(5,8)$	2.34	5.67	8	C_1
$A_5(7,5)$	7.34	3.67	7	C_2
$A_6(6,4)$	7.34	1.33	6	C_2
$A_7(1,2)$	9.66	8.33	2	C_3
$A_8(4,9)$	0.34	7.66	8	C_1

→ Centre of clusters are (3.66, 9) (7, 4.33) (1.5, 3.5)

14/5/2_

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data = {
    'Name': [f'Person_{i+1}' for i in range(50)],
    'Age': np.random.randint(18, 70, size=50),
    'Income': np.random.randint(20000, 120000, size=50)
}

df = pd.DataFrame(data)

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

df = pd.read_csv('income.csv')

X = df[['Age', 'Income']]

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

X_train, X_test = train_test_split(X_scaled, test_size=0.2, random_state=42)

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

plt.plot(k_range, sse, marker='o')
plt.title('SSE vs Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Errors (SSE)')
plt.show()

optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train)
y_pred = kmeans.predict(X_test)

print(f'Predicted Clusters for Test Data: {y_pred}')
```

Program 11

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

Screenshot

29/4/23

Sol-1 PCA

1) calculate mean
2) calculation of covariance matrix
3) calculation of Eigen value of the covariance matrix
4) computation of the eigen vectors unit eigen vectors
5) computation of first principal component
6) geometrical meaning of first principle components

Given the data in table, reduce the dimension from 2 to 1 using the principle component analysis algorithm

Feature	example 1	2	3	4
X_1	4	8	13	7
X_2	11	4	5	14

step 1: calculate mean

$$\bar{X}_1 = \frac{4+8+13+7}{4} = 8$$

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

step 2: calculate the covariance matrix

$$cov(X_1, X_1) = \frac{1}{N-1}\sum_{k=1}^{N} (x_{1k} - \bar{x}_1)^2$$

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

$$= 14$$

$$cov(X_1, X_2) = \frac{1}{N-1}\sum_{k=1}^{N} (x_{1k} - \bar{x}_1)(x_{2k} - \bar{x}_2)$$

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

$$cov(X_2, X_1) = cov(X_1, X_2)$$

$$= -11$$

$$cov(X_2, X_2) = \frac{1}{N-1}\sum_{k=1}^{N} (x_{2k} - \bar{x}_2)^2$$

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

$$= 23$$

2) The covariance matrix is

$$S = \begin{bmatrix} cov(X_1, X_1) & cov(X_1, X_2) \\ cov(X_2, X_1) & cov(X_2, X_2) \end{bmatrix}$$

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

step 3:

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

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

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

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

solving the characteristic equation we get

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

$$= 30.3849,\ 6.6151$$

$$= \lambda_1, \lambda_2 \text{ (say)}$$

Step 4: computation of the eigen vectors

$$\lambda = \lambda_1$$

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

$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \cdot (C - \lambda I)$

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

$= \begin{bmatrix} (14-\lambda_1) u_1 - 11 u_2 \\ -11 u_1 + (23-\lambda_1) u_2 \end{bmatrix}$

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

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

$\frac{u_1}{11} = \frac{u_2}{(14-\lambda) u_2} = t$

$u_1 = 11t \qquad u_2 = (14-\lambda) t$

Taking t is only real number

Taking t = 1

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

$\|u_1\| = \sqrt{11^2 + (14-\lambda_1)^2}$

$= \sqrt{11^2 + (14 - 30.3849)^2}$

$= 19.7384$

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

$e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$

2) by computing similar steps we get

$e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

Date : ____________
Page No. : ____________

Step 5: Computation of first principal component

$$\begin{bmatrix} x_{1,12} \\ x_{2,12} \end{bmatrix}$$

$$e_1^T \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - x_2 \end{bmatrix} = (0.5574 \; -0.83) \begin{pmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{pmatrix}$$

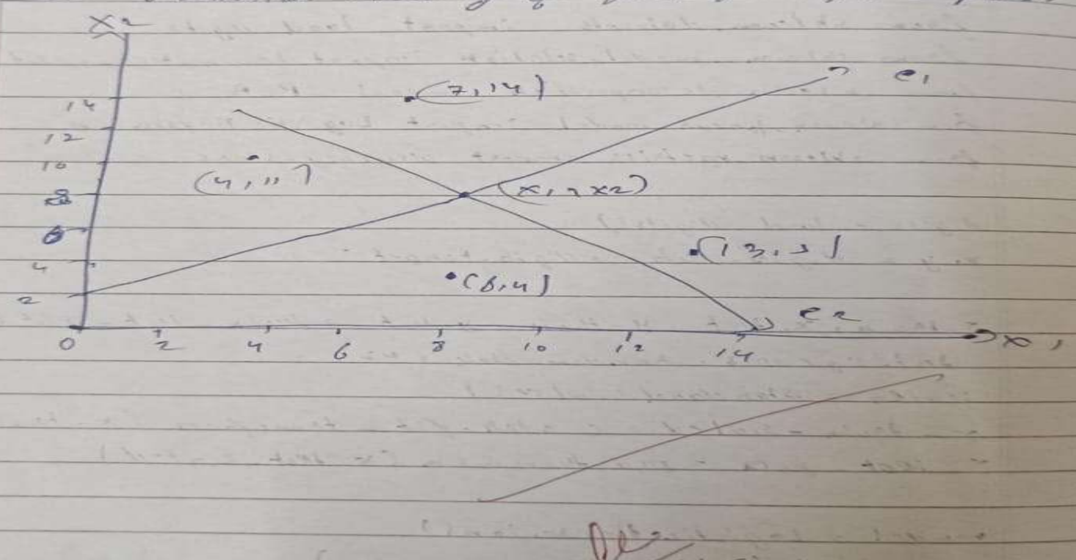
$$= 0.5574\,(x_{1k} - \bar{x}_1) - 0.8303\,(x_{2k} - \bar{x}_2)$$

Date :
Page No. :

Q) Define PCA ?

→ It is a dimensionality reduction and ML method used to a simplify a large data set into a smaller set while still maintaining significant patterns and trends

Step 6: Geometrical meaning of first principal component



Code:

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

df = pd.read_csv('heart (2).csv')

z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
df_no_outliers = df[(z_scores < 3).all(axis=1)]

df_cleaned = df_no_outliers.copy()
for col in df_cleaned.select_dtypes(include='object').columns:
    df_cleaned[col] = LabelEncoder().fit_transform(df_cleaned[col])

X = df_cleaned.drop('HeartDisease', axis=1)
y = df_cleaned['HeartDisease']

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

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

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC()
}

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}")

pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42,
stratify=y)
```

```
print("\nAccuracy with PCA:")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name}: {acc:.4f}")
```