VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Manya Vaid(1BM22CS150)

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 **Manya Vaid(1BM22CS150),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

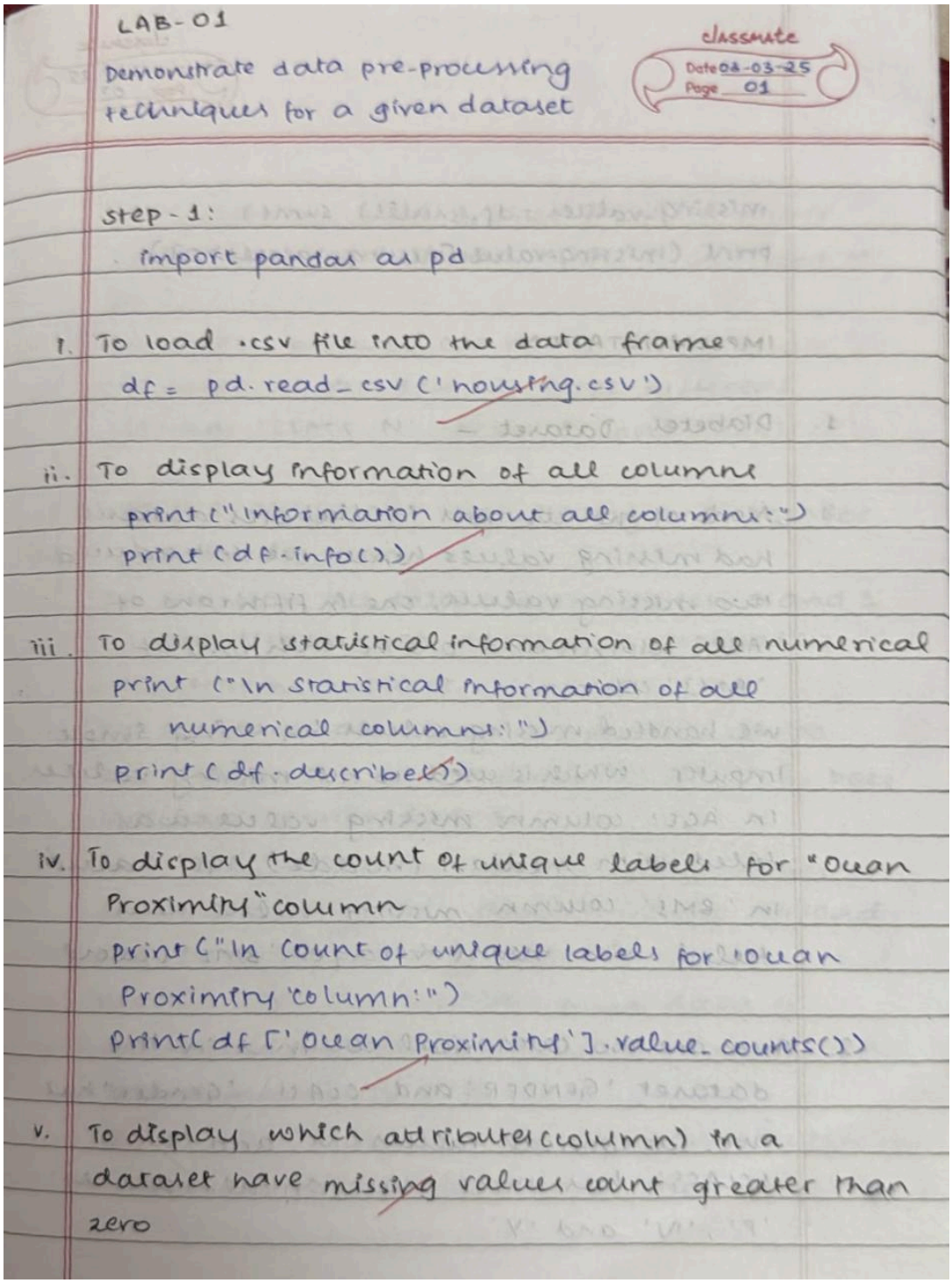
Index

Github Link: https://github.com/Manyav20/Machine_Learning_1BM22CS150

Program 1

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

Screenshot



LAB-01

Demonstrate data pre-processing techniques for a given dataset

Date 08-03-25
Page 01

step-1:

import pandas as pd

i. To load .csv file into the data frame

```
df = pd.read_csv('housing.csv')
```

ii. To display information of all columns

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

iii. To display statistical information of all numerical

```
print("\n Statistical information of all
    numerical columns:")
print(df.describe())
```

iv. To display the count of unique labels for "Ocean Proximity" column

```
print("\n Count of unique labels for 'Ocean
    Proximity' column:")
print(df['Ocean Proximity'].value_counts())
```

v. To display which attribute(column) in a dataset have missing values count greater than zero

Code:

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

```
def createdata():
  data = {
      'Age': np.random.randint(18, 70, size=20),
      'Salary': np.random.randint(30000, 120000, size=20),
      'Purchased': np.random.choice([0, 1], size=20),
      'Gender': np.random.choice(['Male', 'Female'], size=20),
      'City': np.random.choice(['New York', 'San Francisco', 'Los Angeles'], size=20)
  }

  df = pd.DataFrame(data)
  return df

df = createdata()
df.head(10)
```

	Age	Salary	Purchased	Gender	City
0	42	99908	1	Female	New York
1	27	89661	1	Female	Los Angeles
2	29	95665	0	Female	New York
3	53	59711	1	Female	Los Angeles
4	31	98323	0	Male	New York
5	52	78059	1	Male	San Francisco
6	65	77807	1	Male	Los Angeles
7	69	104990	0	Male	San Francisco
8	55	81753	0	Male	New York
9	56	62566	0	Female	San Francisco

```
df.shape
```

```
(20, 5)
```

```
# Introduce some missing values for demonstration
df.loc[5, 'Age'] = np.nan
df.loc[10, 'Salary'] = np.nan
df.head(10)
```

Show hidden output

```
# Basic information about the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Age        19 non-null     float64
 1   Salary     19 non-null     float64
 2   Purchased  20 non-null     int64
```

```
 3   Gender     20 non-null     object
 4   City       20 non-null     object
dtypes: float64(2), int64(1), object(2)
memory usage: 932.0+ bytes
None
```

```
# Summary statistics
print(df.describe())
```

Show hidden output

```
#Code to Find Missing Values
# Check for missing values in each column
missing_values = df.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
Age       1
Salary    1
dtype: int64
```

```
#Set the values to some value (zero, the mean, the median, etc.).
# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column
imputer1.fit(df_copy[["Age"]])
imputer2.fit(df_copy[["Salary"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column
df_copy["Age"] = imputer1.transform(df[["Age"]])
df_copy["Salary"] = imputer2.transform(df[["Salary"]])

# Verify that there are no missing values left
print(df_copy["Age"].isnull().sum())
print(df_copy["Salary"].isnull().sum())
```

```
0
0
```

```
#Handling Categorical Attributes
#Using Ordinal Encoding for gender COlumn and One-Hot Encoding for City Column

# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
# Fit and transform the data
df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])

# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column
encoded_data = onehot_encoder.fit_transform(df[["City"]])

# Convert the sparse matrix to a dense array
encoded_array = encoded_data.toarray()

# Convert to DataFrame for better visualization
encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["City"]))
df_encoded = pd.concat([df_copy, encoded_df], axis=1)

df_encoded.drop("Gender", axis=1, inplace=True)
```

```
df_encoded.drop("City", axis=1, inplace=True)

print(df_encoded. head())
```

```
    Age   Salary  Purchased  Gender_Encoded  City_Los Angeles  City_New York  \
0  42.0  99908.0          1             1.0               0.0            1.0
1  27.0  89661.0          1             1.0               1.0            0.0
2  29.0  95665.0          0             1.0               0.0            1.0
3  53.0  59711.0          1             1.0               1.0            0.0
4  31.0  98323.0          0             0.0               0.0            1.0

   City_San Francisco
0                 0.0
1                 0.0
2                 0.0
3                 0.0
4                 0.0
```

```
#Data Transformation
# Min-Max Scaler/Normalization (range 0-1)
#Pros: Keeps all data between 0 and 1; ideal for distance-based models.
#Cons: Can distort data distribution, especially with extreme outliers.
normalizer = MinMaxScaler()
df_encoded[['Salary']] = normalizer.fit_transform(df_encoded[['Salary']])
df_encoded.head()
```

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	42.0	0.921414	1	1.0	0.0	1.0	0.0
1	27.0	0.762958	1	1.0	1.0	0.0	0.0
2	29.0	0.855802	0	1.0	0.0	1.0	0.0
3	53.0	0.299824	1	1.0	1.0	0.0	0.0
4	31.0	0.896904	0	0.0	0.0	1.0	0.0

```
# Standardization (mean=0, variance=1)
#Pros: Works well for normally distributed data; suitable for many models.
#Cons: Sensitive to outliers.
scaler = StandardScaler()
df_encoded[['Age']] = scaler.fit_transform(df_encoded[['Age']])
df_encoded.head()
```

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	-0.204903	0.921414	1	1.0	0.0	1.0	0.0
1	-1.196369	0.762958	1	1.0	1.0	0.0	0.0
2	-1.064174	0.855802	0	1.0	0.0	1.0	0.0
3	0.522172	0.299824	1	1.0	1.0	0.0	0.0
4	-0.931978	0.896904	0	0.0	0.0	1.0	0.0

```
#Removing Outliers
# Outlier Detection and Treatment using IQR
#Pros: Simple and effective for mild outliers.
#Cons: May overly reduce variation if there are many extreme outliers.
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['Salary'].quantile(0.25)
Q3 = df_encoded_copy1['Salary'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
```

```
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['Salary'] = np.where(df_encoded_copy1['Salary'] > upper_bound, upper_bound,
                       np.where(df_encoded_copy1['Salary'] < lower_bound, lower_bound, df_encoded_copy1['Salary

print(df_encoded_copy1.head())
```

```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0 -0.204903  0.921414          1             1.0               0.0
1 -1.196369  0.762958          1             1.0               1.0
2 -1.064174  0.855802          0             1.0               0.0
3  0.522172  0.299824          1             1.0               1.0
4 -0.931978  0.896904          0             0.0               0.0

   City_New York  City_San Francisco
0            1.0                 0.0
1            0.0                 0.0
2            1.0                 0.0
3            0.0                 0.0
4            1.0                 0.0
```

```
#Removing Outliers
# Z-score method
#Pros: Good for normally distributed data.
#Cons: Not suitable for non-normal data; may miss outliers in skewed distributions.

df_encoded_copy2['Salary_zscore'] = stats.zscore(df_encoded_copy2['Salary'])
df_encoded_copy2['Salary'] = np.where(df_encoded_copy2['Salary_zscore'].abs() > 3, np.nan, df_encoded_copy2['Sal
print(df_encoded_copy2.head())
```

```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0 -0.204903  0.921414          1             1.0               0.0
1 -1.196369  0.762958          1             1.0               1.0
2 -1.064174  0.855802          0             1.0               0.0
3  0.522172  0.299824          1             1.0               1.0
4 -0.931978  0.896904          0             0.0               0.0

   City_New York  City_San Francisco  Salary_zscore
0            1.0                 0.0       1.213641
1            0.0                 0.0       0.575541
2            1.0                 0.0       0.949421
3            0.0                 0.0      -1.289500
4            1.0                 0.0       1.114940
```

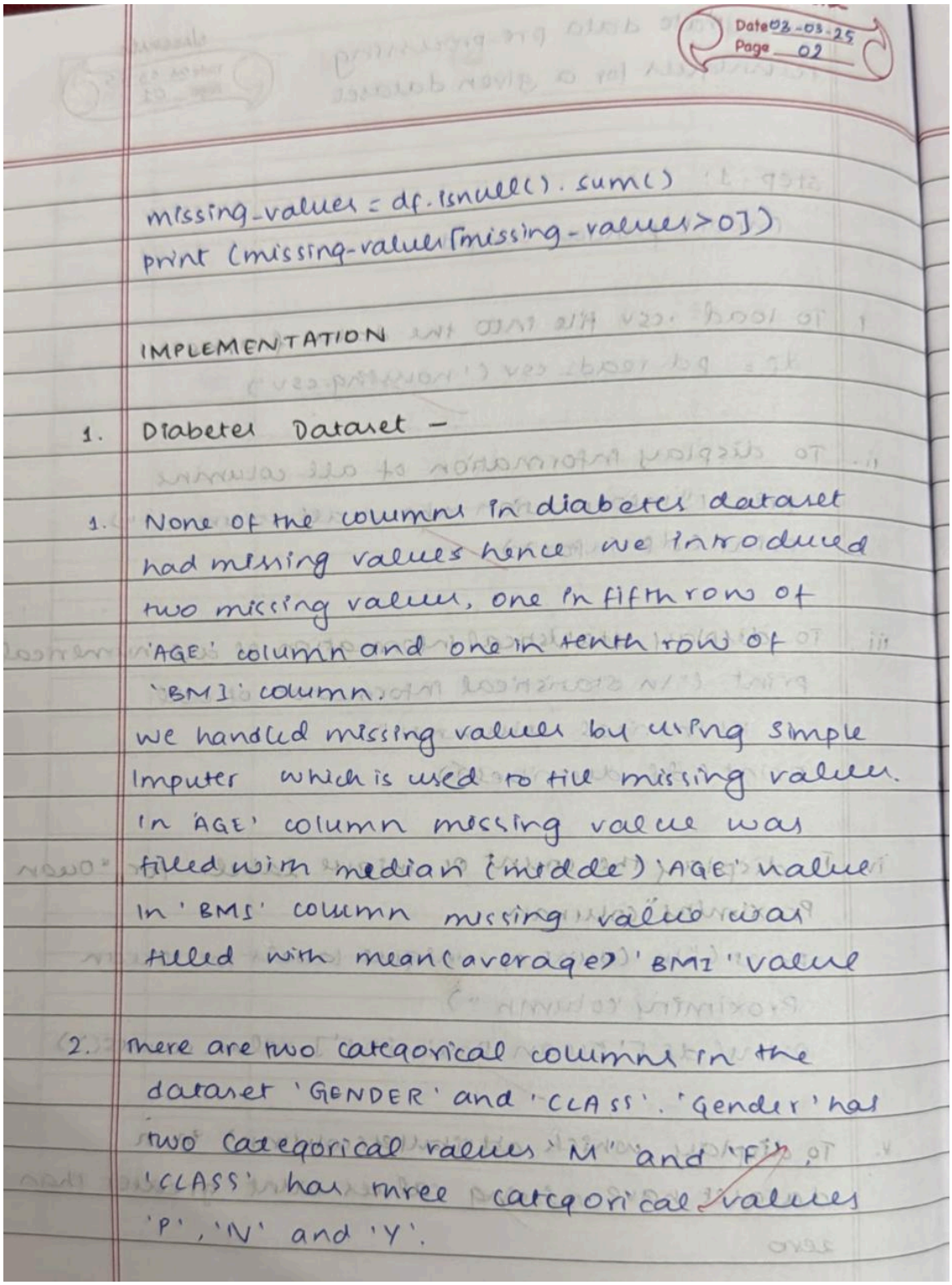
```
#Removing Outliers
# Median replacement for outliers
#Pros: Keeps distribution shape intact, useful when capping isn't feasible.
#Cons: May distort data if outliers represent real phenomena.
df_encoded_copy3['Salary_zscore'] = stats.zscore(df_encoded_copy3['Salary'])
median_salary = df_encoded_copy3['Salary'].median()
df_encoded_copy3['Salary'] = np.where(df_encoded_copy3['Salary_zscore'].abs() > 3, median_salary, df_encoded_cop
print(df_encoded_copy3.head())
```

```
        Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  \
0 -0.204903  0.921414          1             1.0               0.0
1 -1.196369  0.762958          1             1.0               1.0
2 -1.064174  0.855802          0             1.0               0.0
3  0.522172  0.299824          1             1.0               1.0
4 -0.931978  0.896904          0             0.0               0.0

   City_New York  City_San Francisco  Salary_zscore
0            1.0                 0.0       1.213641
1            0.0                 0.0       0.575541
2            1.0                 0.0       0.949421
3            0.0                 0.0      -1.289500
4            1.0                 0.0       1.114940
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



Date 03-03-25
Page 02

```
missing_values = df.isnull().sum()
print(missing_values[missing_values>0])
```

IMPLEMENTATION

1. Diabetes Dataset –

1. None of the columns in diabetes dataset had missing values hence we introduced two missing values, one in fifth row of 'AGE' column and one in tenth row of 'BMI' column.
we handled missing values by using simple Imputer which is used to fill missing values. In 'AGE' column missing value was filled with median (middle) 'AGE' value In 'BMI' column missing value was filled with mean(average) 'BMI' value

2. There are two categorical columns in the dataset 'GENDER' and 'CLASS'. 'Gender' has two categorical values 'M' and 'F', 'CLASS' has three categorical values 'P', 'N' and 'Y'.

For 'Gender' we used ordinal encoding where output is a single column with 'F' as 0 and 'M' as 1.
For 'CLASS' we used one hot encoding where the single column becomes three different columns 'CLASS_N', 'CLASS_P' and 'CLASS_Y'.

3. We use 'TG' for Min-Max scaling and 'AGE' for standardization.
In Min-Max all data is kept between 0 and 1. It can distort data distribution, especially with extreme outliers.
In standardization data is transformed to have mean 0 and standard deviation 1. Does not have specified range
We use min-max scaling when either distance matters or data is evenly spread and fixed bounds are present.
We use standardization when data is normally distributed, ideal for regression-based models or PCA.
We would choose standardization if outliers are a concern Min-Max if they aren't.

Date 03-07-25
Page 04

1. Adult Income Dataset
None of the columns in Adult Income dataset had null values. We introduced null values in 5th row of 'educational-num' and 7th row of 'age'.
We handled missing values by SimpleImputer for both columns. We replaced 'educational-num' null value with median and 'age' null value with mean of the given columns respectively.

2. We identified multiple categorical columns such as 'gender', 'race', 'marital-status', 'relationship' etc. We used Ordinal Encoding in 'gender' and One Hot encoding in 'race'.

3. We used 'hours-per-week' for Min-Max scaling where all data is in range of 0 to 1 and 'age' for standardization where all data is transformed to have mean 0 and standard deviation 1. We use Min-Max for evenly spread data and standardization

Date 03-03-25
Page 05

for normally distributed data. Min-Max is chosen when outliers are not important and standardization when they are.

Code:
Adult income dataset

```
print(df.describe())
```

```
                age        fnlwgt  educational-num  capital-gain  \
count  48842.000000  4.884200e+04     48842.000000  48842.000000
mean      38.643585  1.896641e+05        10.078089   1079.067626
std       13.710510  1.056040e+05         2.570973   7452.019058
min       17.000000  1.228500e+04         1.000000      0.000000
25%       28.000000  1.175505e+05         9.000000      0.000000
50%       37.000000  1.781445e+05        10.000000      0.000000
75%       48.000000  2.376420e+05        12.000000      0.000000
max       90.000000  1.490400e+06        16.000000  99999.000000

       capital-loss  hours-per-week
count  48842.000000    48842.000000
mean      87.502314       40.422382
std      403.004552       12.391444
min        0.000000        1.000000
25%        0.000000       40.000000
50%        0.000000       40.000000
75%        0.000000       45.000000
max     4356.000000       99.000000
```

```
#Code to Find Missing Values
# Check for missing values in each column
missing_values = df.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
Series([], dtype: int64)
```

```
import numpy as np

# Introduce missing values at specific locations
df.loc[5, 'educational-num'] = np.nan  # Set missing value for 'AGE' at row index 5
df.loc[7, 'age'] = np.nan  # Set missing value for 'BMI' at row index 10

# Display the first 10 rows to check the changes
print(df.head(10))
```

```
    age         workclass  fnlwgt     education  educational-num  \
0  25.0           Private  226802          11th              7.0
1  38.0           Private   89814       HS-grad              9.0
2  28.0         Local-gov  336951    Assoc-acdm             12.0
3  44.0           Private  160323  Some-college             10.0
4  18.0                 ?  103497  Some-college             10.0
5  34.0           Private  198693          10th              NaN
6  29.0                 ?  227026       HS-grad              9.0
7   NaN  Self-emp-not-inc  104626   Prof-school             15.0
8  24.0           Private  369667  Some-college             10.0
9  55.0           Private  104996       7th-8th              4.0

       marital-status         occupation   relationship   race  gender  \
0       Never-married  Machine-op-inspct      Own-child  Black    Male
1  Married-civ-spouse    Farming-fishing        Husband  White    Male
2  Married-civ-spouse    Protective-serv        Husband  White    Male
3  Married-civ-spouse  Machine-op-inspct        Husband  Black    Male
4       Never-married                  ?      Own-child  White  Female
5       Never-married      Other-service  Not-in-family  White    Male
6       Never-married                  ?      Unmarried  Black    Male
7  Married-civ-spouse     Prof-specialty        Husband  White    Male
8       Never-married      Other-service      Unmarried  White  Female
9  Married-civ-spouse       Craft-repair        Husband  White    Male

   capital-gain  capital-loss  hours-per-week native-country income
0             0             0              40  United-States  <=50K
1             0             0              50  United-States  <=50K
2             0             0              40  United-States   >50K
```

```
3          7688             0             40  United-States   >50K
4             0             0             30  United-States  <=50K
5             0             0             30  United-States  <=50K
6             0             0             40  United-States  <=50K
7          3103             0             32  United-States   >50K
8             0             0             40  United-States  <=50K
9             0             0             10  United-States  <=50K
```

```
#Code to Find Missing Values
# Check for missing values in each column
missing_values = df.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
age               1
educational-num   1
dtype: int64
```

```
#Set the values to some value (zero, the mean, the median, etc.).
# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column
imputer1.fit(df_copy[["educational-num"]])
imputer2.fit(df_copy[["age"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column
df_copy["educational-num"] = imputer1.transform(df[["educational-num"]])
df_copy["age"] = imputer2.transform(df[["age"]])

# Verify that there are no missing values left
print(df_copy["educational-num"].isnull().sum())
print(df_copy["age"].isnull().sum())
```

```
0
0
```

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

# Normalize the Gender column to be consistent (uppercase in this case)
df['gender'] = df['gender'].str.upper()  # Convert to uppercase

# Initialize OrdinalEncoder for 'Gender' column
ordinal_encoder = OrdinalEncoder(categories=[["FEMALE", "MALE"]])  # Encoding 'F' as 0, 'M' as 1

# Fit and transform the data in the 'Gender' column
df["gender_encoded"] = ordinal_encoder.fit_transform(df[["gender"]])

# Initialize OneHotEncoder for the 'City' column (if the column exists)
# You should replace "City" with the actual column name in your dataset
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column (replace 'City' with the actual name of the column)
if 'race' in df.columns:
    encoded_data = onehot_encoder.fit_transform(df[["race"]])

    # Convert the sparse matrix to a dense array
    encoded_array = encoded_data.toarray()

    # Convert to DataFrame for better visualization
    encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["race"]))
```

```
    # Concatenate the one-hot encoded columns with the original DataFrame
    df_encoded = pd.concat([df, encoded_df], axis=1)

    # Drop the original 'City' column as it is now encoded
    df_encoded.drop("race", axis=1, inplace=True)

# If there is no 'City' column, proceed with just encoding 'Gender'
else:
    df_encoded = df.copy()

# Drop the original 'Gender' column
df_encoded.drop("gender", axis=1, inplace=True)

# Display the first few rows of the encoded dataframe
print(df_encoded.head())
df_encoded
```

```
    age  workclass  fnlwgt     education  educational-num      marital-status  \
0  25.0    Private  226802          11th              7.0       Never-married
1  38.0    Private   89814       HS-grad              9.0  Married-civ-spouse
2  28.0  Local-gov  336951    Assoc-acdm             12.0  Married-civ-spouse
3  44.0    Private  160323  Some-college             10.0  Married-civ-spouse
4  18.0          ?  103497  Some-college             10.0       Never-married

          occupation relationship  capital-gain  capital-loss  hours-per-week  \
0  Machine-op-inspct    Own-child             0             0              40
1    Farming-fishing      Husband             0             0              50
2    Protective-serv      Husband             0             0              40
3  Machine-op-inspct      Husband          7688             0              40
4                  ?    Own-child             0             0              30

  native-country income  gender_encoded  race_Amer-Indian-Eskimo  \
0  United-States  <=50K             1.0                      0.0
1  United-States  <=50K             1.0                      0.0
2  United-States   >50K             1.0                      0.0
3  United-States   >50K             1.0                      0.0
4  United-States  <=50K             0.0                      0.0

   race_Asian-Pac-Islander  race_Black  race_Other  race_White
0                      0.0         1.0         0.0         0.0
1                      0.0         0.0         0.0         1.0
2                      0.0         0.0         0.0         1.0
3                      0.0         1.0         0.0         0.0
4                      0.0         0.0         0.0         1.0
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	capital-gain	cap
0	25.0	Private	226802	11th	7.0	Never-married	Machine-op-inspct	Own-child	0	
1	38.0	Private	89814	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	0	
2	28.0	Local-gov	336951	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	0	
3	44.0	Private	160323	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	7688	
4	18.0	?	103497	Some-college	10.0	Never-married	?	Own-child	0	
...	...	...	...	...	...	...	...	...	...	
48837	27.0	Private	257302	Assoc-acdm	12.0	Married-civ-spouse	Tech-support	Wife	0	
48838	40.0	Private	154374	HS-grad	9.0	Married-civ-spouse	Machine-op-inspct	Husband	0	
48839	58.0	Private	151910	HS-grad	9.0	Widowed	Adm-clerical	Unmarried	0	
48840	22.0	Private	201490	HS-grad	9.0	Never-married	Adm-clerical	Own-child	0	
48841	52.0	Self-emp-inc	287927	HS-grad	9.0	Married-civ-spouse	Exec-managerial	Wife	15024	

48842 rows × 19 columns

```
#Pros: Keeps all data between 0 and 1; ideal for distance-based models.
#Cons: Can distort data distribution, especially with extreme outliers.
normalizer = MinMaxScaler()
df_encoded[['hours-per-week']] = normalizer.fit_transform(df_encoded[['hours-per-week']])
df_encoded.head()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	capital-gain	capital-loss	
0	25.0	Private	226802	11th	7.0	Never-married	Machine-op-inspct	Own-child	0	0	0
1	38.0	Private	89814	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	0	0	0
2	28.0	Local-gov	336951	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	0	0	0
3	44.0	Private	160323	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	7688	0	0
4	18.0	?	103497	Some-college	10.0	Never-married	?	Own-child	0	0	0

```
# Standardization (mean=0, variance=1)
#Pros: Works well for normally distributed data; suitable for many models.
#Cons: Sensitive to outliers.
scaler = StandardScaler()
df_encoded[['age']] = scaler.fit_transform(df_encoded[['age']])
df_encoded.head()
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	capital-gain	capita lo
0	-0.995125	Private	226802	11th	7.0	Never-married	Machine-op-inspct	Own-child	0	
1	-0.046907	Private	89814	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	0	
2	-0.776305	Local-gov	336951	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	0	
3	0.390732	Private	160323	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	7688	
4	-1.505704	?	103497	Some-college	10.0	Never-married	?	Own-child	0	

```
#Removing Outliers
# Outlier Detection and Treatment using IQR
#Pros: Simple and effective for mild outliers.
#Cons: May overly reduce variation if there are many extreme outliers.
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['fnlwgt'].quantile(0.25)
Q3 = df_encoded_copy1['fnlwgt'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
```

Diabetes Dataset

```
#Code to Find Missing Values
# Check for missing values in each column
missing_values = df.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])
```

```
AGE    1
BMI    1
dtype: int64
```

```
#Set the values to some value (zero, the mean, the median, etc.).
# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column
imputer1.fit(df_copy[["AGE"]])
imputer2.fit(df_copy[["BMI"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column
df_copy["AGE"] = imputer1.transform(df[["AGE"]])
df_copy["BMI"] = imputer2.transform(df[["BMI"]])

# Verify that there are no missing values left
print(df_copy["AGE"].isnull().sum())
print(df_copy["BMI"].isnull().sum())
```

```
0
0
```

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

# Normalize the Gender column to be consistent (uppercase in this case)
df['Gender'] = df['Gender'].str.upper()  # Convert to uppercase

# Initialize OrdinalEncoder for 'Gender' column
ordinal_encoder = OrdinalEncoder(categories=[["F", "M"]])  # Encoding 'F' as 0, 'M' as 1

# Fit and transform the data in the 'Gender' column
```

```
df["Gender_Encoded"] = ordinal_encoder.fit_transform(df[["Gender"]])

# Initialize OneHotEncoder for the 'City' column (if the column exists)
# You should replace "City" with the actual column name in your dataset
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column (replace 'City' with the actual name of the column)
if 'CLASS' in df.columns:
    encoded_data = onehot_encoder.fit_transform(df[["CLASS"]])

    # Convert the sparse matrix to a dense array
    encoded_array = encoded_data.toarray()

    # Convert to DataFrame for better visualization
    encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["CLASS"]))

    # Concatenate the one-hot encoded columns with the original DataFrame
    df_encoded = pd.concat([df, encoded_df], axis=1)

    # Drop the original 'City' column as it is now encoded
    df_encoded.drop("CLASS", axis=1, inplace=True)

# If there is no 'City' column, proceed with just encoding 'Gender'
else:
    df_encoded = df.copy()

# Drop the original 'Gender' column
df_encoded.drop("Gender", axis=1, inplace=True)

# Display the first few rows of the encoded dataframe
print(df_encoded.head())
df_encoded
```

```
     ID  No_Pation   AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL   BMI  \
0  502      17975  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
1  735      34221  26.0   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6  23.0
2  420      47975  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
3  680      87656  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
4  504      34223  33.0   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4  21.0

   Gender_Encoded  CLASS_N  CLASS_N   CLASS_P  CLASS_Y  CLASS_Y
0             0.0      1.0       0.0      0.0      0.0       0.0
1             1.0      1.0       0.0      0.0      0.0       0.0
2             0.0      1.0       0.0      0.0      0.0       0.0
3             0.0      1.0       0.0      0.0      0.0       0.0
4             1.0      1.0       0.0      0.0      0.0       0.0
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Gender_Encoded	CLASS_N	CLASS_N	(
0	502	17975	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	
1	735	34221	26.0	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	1.0	1.0	0.0	
2	420	47975	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	
3	680	87656	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0	
4	504	34223	33.0	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	1.0	1.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
995	200	454317	71.0	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	1.0	0.0	0.0	
996	671	876534	31.0	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	1.0	0.0	0.0	
997	669	87654	30.0	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	1.0	0.0	0.0	
998	99	24004	38.0	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	1.0	0.0	0.0	
999	248	24054	54.0	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	1.0	0.0	0.0	

1000 rows × 18 columns

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

# Ensure the Gender column is in the dataframe
if 'Gender' in df.columns:
    # Normalize the Gender column to be consistent (uppercase in this case)
    df['Gender'] = df['Gender'].str.upper()  # Convert to uppercase

    # Initialize OrdinalEncoder for 'Gender' column
    ordinal_encoder = OrdinalEncoder(categories=[["F", "M"]])  # Encoding 'F' as 0, 'M' as 1

    # Fit and transform the data in the 'Gender' column
    df['Gender_Encoded'] = ordinal_encoder.fit_transform(df[['Gender']])

    # Drop the original 'Gender' column
    df.drop('Gender', axis=1, inplace=True)

# One-hot encoding for 'CLASS' column (if it exists)
if 'CLASS' in df.columns:
    onehot_encoder = OneHotEncoder()

    # Fit and transform the "CLASS" column
    encoded_data = onehot_encoder.fit_transform(df[['CLASS']])

    # Convert the sparse matrix to a dense array
    encoded_array = encoded_data.toarray()

    # Convert to DataFrame for better visualization
    encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(['CLASS']))

    # Concatenate the one-hot encoded columns with the original DataFrame
    df = pd.concat([df, encoded_df], axis=1)

    # Drop the original 'CLASS' column as it is now encoded
    df.drop('CLASS', axis=1, inplace=True)

# Display the first few rows of the encoded dataframe
print(df.head())
```

```
     ID  No_Pation   AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL   BMI  \
0  502      17975  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
1  735      34221  26.0   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6  23.0
2  420      47975  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
3  680      87656  50.0   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
4  504      34223  33.0   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4  21.0

   Gender_Encoded  CLASS_N  CLASS_N   CLASS_P  CLASS_Y  CLASS_Y
0             0.0      1.0       0.0      0.0      0.0      0.0
1             1.0      1.0       0.0      0.0      0.0      0.0
2             0.0      1.0       0.0      0.0      0.0      0.0
3             0.0      1.0       0.0      0.0      0.0      0.0
4             1.0      1.0       0.0      0.0      0.0      0.0
```

```
df
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Gender_Encoded	CLASS_N	CLASS_N
0	502	17975	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0
1	735	34221	26.0	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	1.0	1.0	0.0
2	420	47975	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0
3	680	87656	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0
4	504	34223	33.0	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	1.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	200	454317	71.0	11.0	97	7.0	7.5	1.7	1.2	1.8	0.6	30.0	1.0	0.0	0.0
996	671	876534	31.0	3.0	60	12.3	4.1	2.2	0.7	2.4	15.4	37.2	1.0	0.0	0.0
997	669	87654	30.0	7.1	81	6.7	4.1	1.1	1.2	2.4	8.1	27.4	1.0	0.0	0.0
998	99	24004	38.0	5.8	59	6.7	5.3	2.0	1.6	2.9	14.0	40.5	1.0	0.0	0.0
999	248	24054	54.0	5.0	67	6.9	3.8	1.7	1.1	3.0	0.7	33.0	1.0	0.0	0.0

1000 rows × 18 columns

```
#Data Transformation
# Min-Max Scaler/Normalization (range 0-1)
#Pros: Keeps all data between 0 and 1; ideal for distance-based models.
#Cons: Can distort data distribution, especially with extreme outliers.
normalizer = MinMaxScaler()
df_encoded[['TG']] = normalizer.fit_transform(df_encoded[['TG']])
df_encoded.head()
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Age	Salary	Gender_Encoded	CL
0	502	17975	50.0	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	50.0	24.0	0.0	
1	735	34221	26.0	4.5	62	4.9	3.7	0.081481	1.1	2.1	0.6	23.0	26.0	23.0	1.0	
2	420	47975	50.0	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	50.0	24.0	0.0	
3	680	87656	50.0	4.7	46	4.9	4.2	0.044444	2.4	1.4	0.5	24.0	50.0	24.0	0.0	
4	504	34223	33.0	7.1	46	4.9	4.9	0.051852	0.8	2.0	0.4	21.0	33.0	21.0	1.0	

```
# Standardization (mean=0, variance=1)
#Pros: Works well for normally distributed data; suitable for many models.
#Cons: Sensitive to outliers.
scaler = StandardScaler()
df_encoded[['Age']] = scaler.fit_transform(df_encoded[['Age']])
df_encoded.head()
```

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Gender_Encoded	CLASS_N	CLASS_N
0	502	17975	-0.402465	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0
1	735	34221	-3.132585	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	1.0	1.0	0.0
2	420	47975	-0.402465	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0
3	680	87656	-0.402465	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	0.0	1.0	0.0
4	504	34223	-2.336300	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	1.0	1.0	0.0

```
#Removing Outliers
# Outlier Detection and Treatment using IQR
#Pros: Simple and effective for mild outliers.
#Cons: May overly reduce variation if there are many extreme outliers.
```

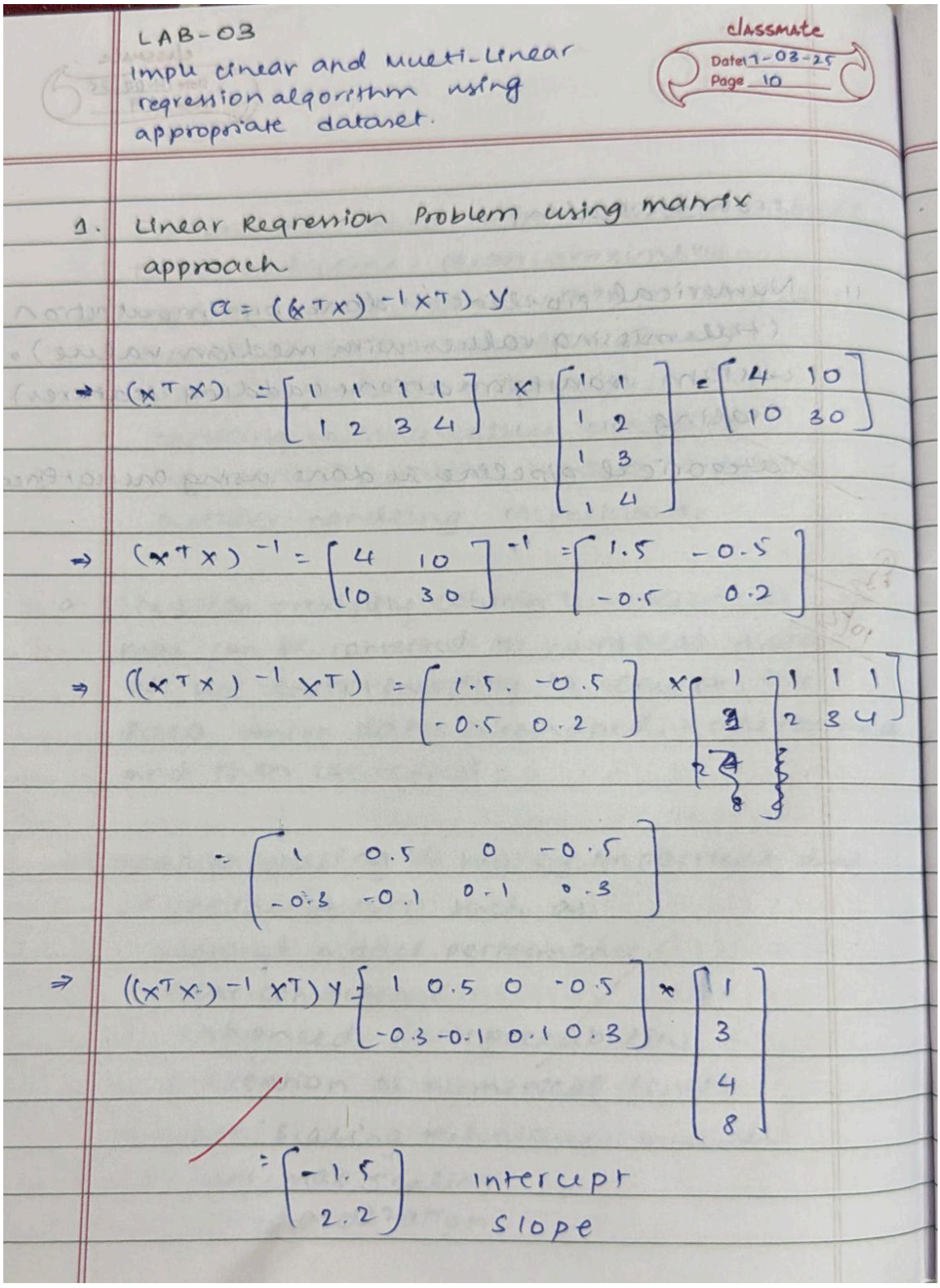
```
#Removing Outliers
# Median replacement for outliers
#Pros: Keeps distribution shape intact, useful when capping isn't feasible.
#Cons: May distort data if outliers represent real phenomena.
df_encoded_copy3['Chol_zscore'] = stats.zscore(df_encoded_copy3['Chol'])
median_salary = df_encoded_copy3['Chol'].median()
df_encoded_copy3['Chol'] = np.where(df_encoded_copy3['Chol_zscore'].abs() > 3, median_salary, df_encoded_copy3['
print(df_encoded_copy3.head())
```

```
   ID  No_Pation       AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL   BMI  \
0  502      17975 -0.402465   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
1  735      34221 -3.132585   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6  23.0
2  420      47975 -0.402465   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
3  680      87656 -0.402465   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
4  504      34223 -2.336300   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4  21.0

   Gender_Encoded  CLASS_N  CLASS_N  CLASS_P  CLASS_Y  CLASS_Y  Chol_zscore
0             0.0      1.0      0.0      0.0      0.0      0.0    -0.532005
1             1.0      1.0      0.0      0.0      0.0      0.0    -0.945425
2             0.0      1.0      0.0      0.0      0.0      0.0    -0.532005
3             0.0      1.0      0.0      0.0      0.0      0.0    -0.532005
4             1.0      1.0      0.0      0.0      0.0      0.0     0.046783
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



2. Linear Regression Problem

$$Y = B_0 + B_1 X$$

$B_0 \rightarrow$ intercept $B_1 \rightarrow$ slope

X	Y	X^2	XY
8	10	64	80
10	13	100	130
12	16	144	192
Sum = 30	Sum = 39	Sum = 308	Sum = 402
$\bar{x} = 10$	$\bar{y} = 13$	$\bar{x^2} = 102.67$	$\bar{xy} = 134$

$$B_1 = \frac{(134) - 10 \times 39}{102.67 - (10)^2} = 1.5$$

$$B_0 = 13 - (1.5) \times 10 = -2$$

For $X = 20$

$$Y = -2 + (1.5)(20)$$

$$= -2 + 30 = 28$$

∴ Price of 20 inch pizza is $28

3. We did not use any data preprocessing steps in canada_per_capita_income and salary.csv as null values can only be handled and not encoded or scaled as linear regression models have only one feature. in multi linear regression since features might have different scales to avoid mistakes we use scaling and encoding

4. Yes we visualized the regression line. Plot indicates steady increase of income over time

5. Predicted salary is $ 864 24.67

6. Yes we used OneHotEncoding for categorical values. since dataset values have similar magnitudes scaling was not used.

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.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
```

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving housing_area_price.csv to housing_area_price.csv
```

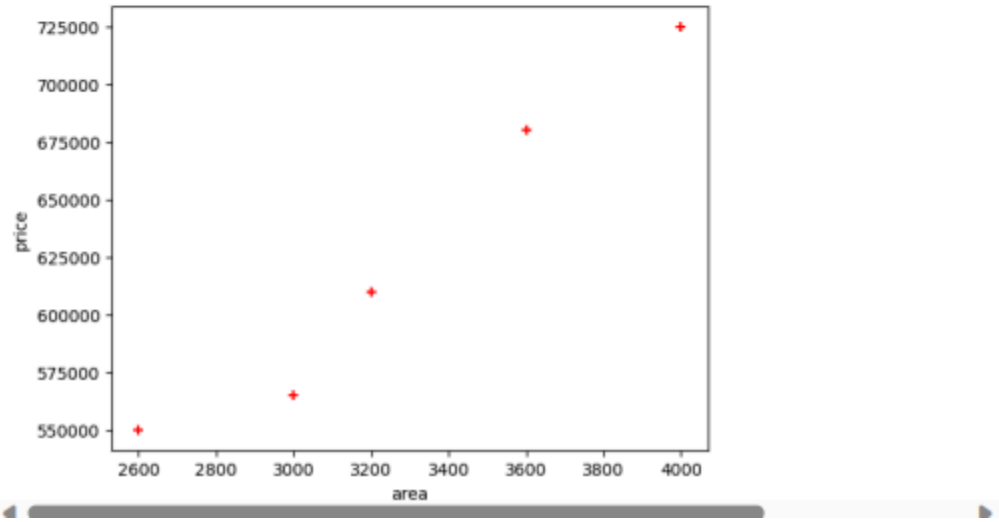
```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
```

```
import pandas as pd
df = pd.read_csv('housing_area_price.csv')
df.head()
```

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

```
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')
```

```
<matplotlib.collections.PathCollection at 0x7b1745104b50>
```



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

	area
0	2600
1	3000
2	3200
3	3600
4	4000

```
price = df.price
price
```

	price
0	550000
1	565000
2	610000
3	680000
4	725000

```
reg = linear_model.LinearRegression()
reg.fit(new_df,price)
```

```
LinearRegression()
```

```
reg.predict([[3300]])
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
  warnings.warn(
array([628715.75342466])
```

```
reg.coef_

reg.intercept_
```

```
180616.43835616432
```

```
3300*135.78767123 + 180616.43835616432
```

```
628715.7534151643
```

```
reg.predict([[5000]])
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
  warnings.warn(
array([859554.79452055])
```

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.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
```

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving homeprices_Multiple_LR.csv to homeprices_Multiple_LR.csv
```

```
import pandas as pd
import numpy as np
from sklearn import linear_model
```

```
import pandas as pd
df = pd.read_csv('homeprices_Multiple_LR.csv')
df.head()
```

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000

```
df.bedrooms.median()
```

```
4.0
```

```
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
df
```

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	4.0	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000
5	4100	6.0	8	810000

```
reg = linear_model.LinearRegression()

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

```
LinearRegression()
```

```
reg.coef_
```

```
array([  112.06244194, 23388.88007794, -3231.71790863])
```

```
reg.intercept_
```

```
221323.00186540396
```

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

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
  warnings.warn(
array([498408.25158031])
```

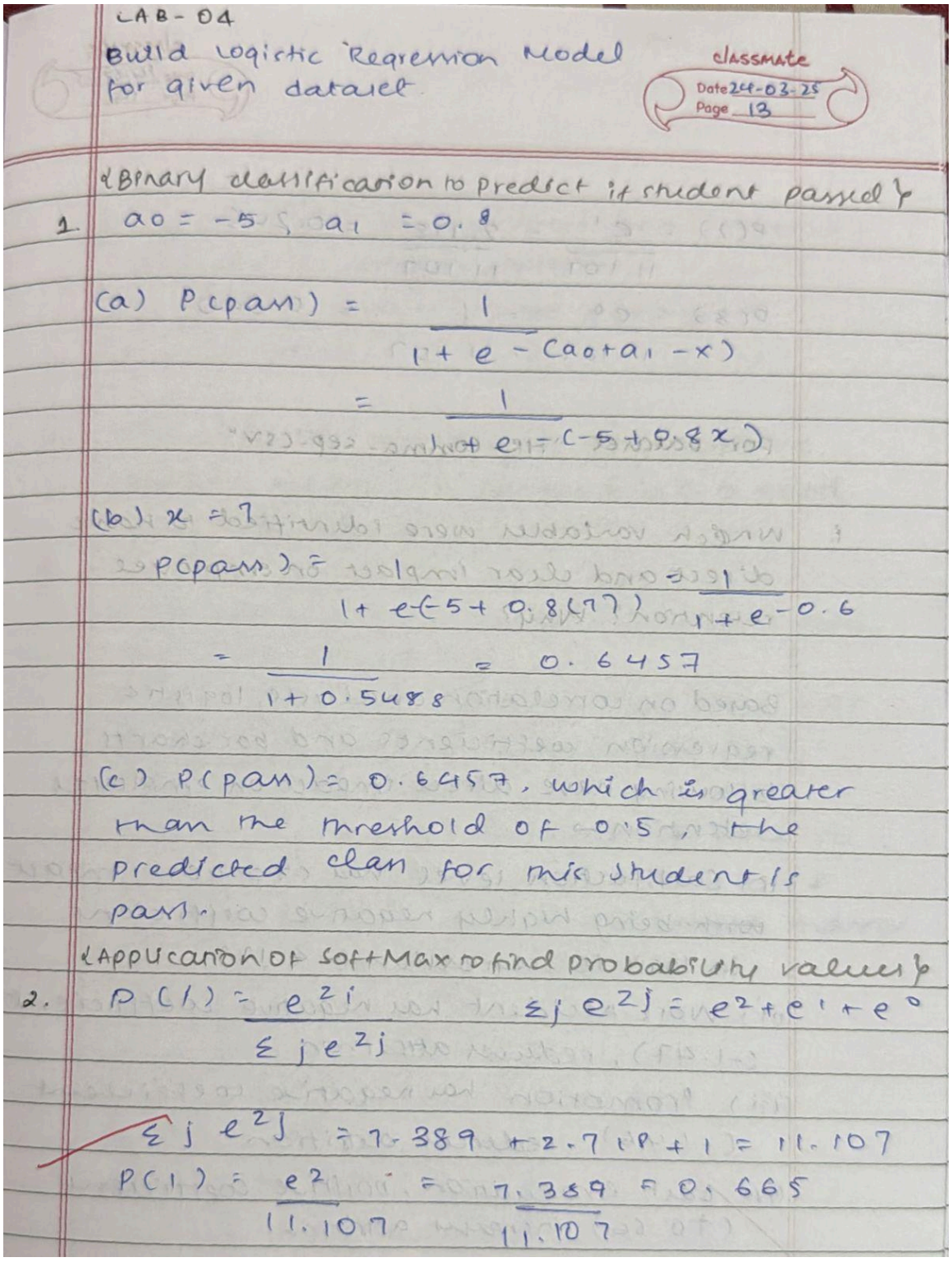
```
112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384
```

```
498408.25157402386
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot



Date 24-02-25
Page 14

$P(2) = \frac{e^1}{11.107} = \frac{2.718}{11.107} = 0.245$

$P(3) = \frac{e^0}{11.107} = \frac{1}{11.107} = 0.090$

For Dataset "HR-comma-sep.csv"

1. Which variables were identified to have direct and clear impact on employee retention? Why?

Based on correlation analysis, logistic regression coefficients and bar charts following have direct impact on employee retention →

(i) Satisfaction level has strongest impact ~~with~~ being highly negative coefficient in logistic regression (-4.17)

(ii) Work Accident has negative coefficient (-1.47), reduces attrition

(iii) Promotion has negative coefficient (-1.46), reduces attrition

(iv) Last evaluation, positive coefficient (+0.66) increases attrition

(v) Time spent : positive coefficient (+0.23) steadily increased attrition

(vi) Salary level : strong impact

(vii) Department

ii. What was the accuracy of your logistic regression model? Do you think it is a good accuracy? Why or why not?

Accuracy of logistic regression model was 0.756667 ≈ 75%

It is a decent accuracy but may not be the best metric to evaluate the model's performance.

Accuracy may be misleading due to class imbalance issue, precision & recall being low and confusion matrix showing high false negatives

Code:

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

```
import pandas as pd
import numpy as np
from sklearn import linear_model
```

```
from google.colab import files
uploaded = files.upload()
```

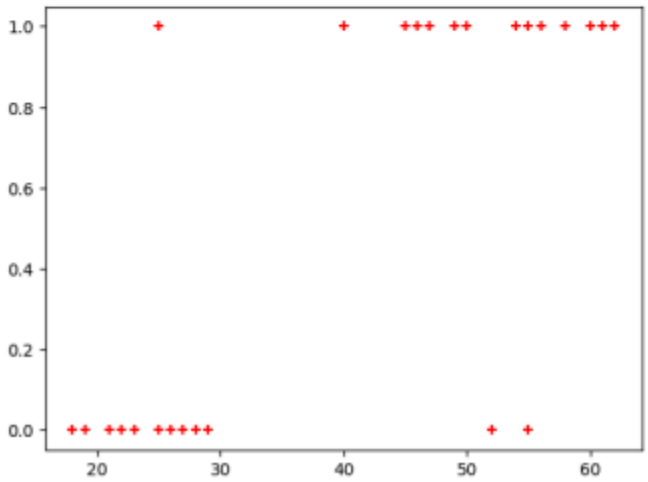
```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving insurance_data.csv to insurance_data.csv
```

```
import pandas as pd
df = pd.read_csv('insurance_data.csv')
df.head()
```

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

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

```
<matplotlib.collections.PathCollection at 0x7e5edc965a10>
```




```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']],df.bought_insurance,train_size=0.9,random_state=
```

```
X_train.shape
```

```
(24, 1)
```

```
X_test
```

	age
7	60
5	56
18	19

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

```
model.fit(X_train, y_train)
```

```
▾ LogisticRegression ① ②
LogisticRegression()
```

```
X_test
```

	age
7	60
5	56
18	19

```
y_test
```

	bought_insurance
7	1
5	1
18	0

dtype: int64

```
y_predicted = model.predict(X_test)
y_predicted
```

```
array([1, 1, 0])
```

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

```
1.0
```

```
model.predict_proba(X_test)
```

```
array([[0.06470655, 0.93529345],
       [0.10327333, 0.89672667],
       [0.92775258, 0.07224742]])
```

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

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
  warnings.warn(
array([1])
```

```
model.coef_
```

```
array([[0.1274065]])
```

```
model.intercept_
```

```
array([-4.97339194])
```

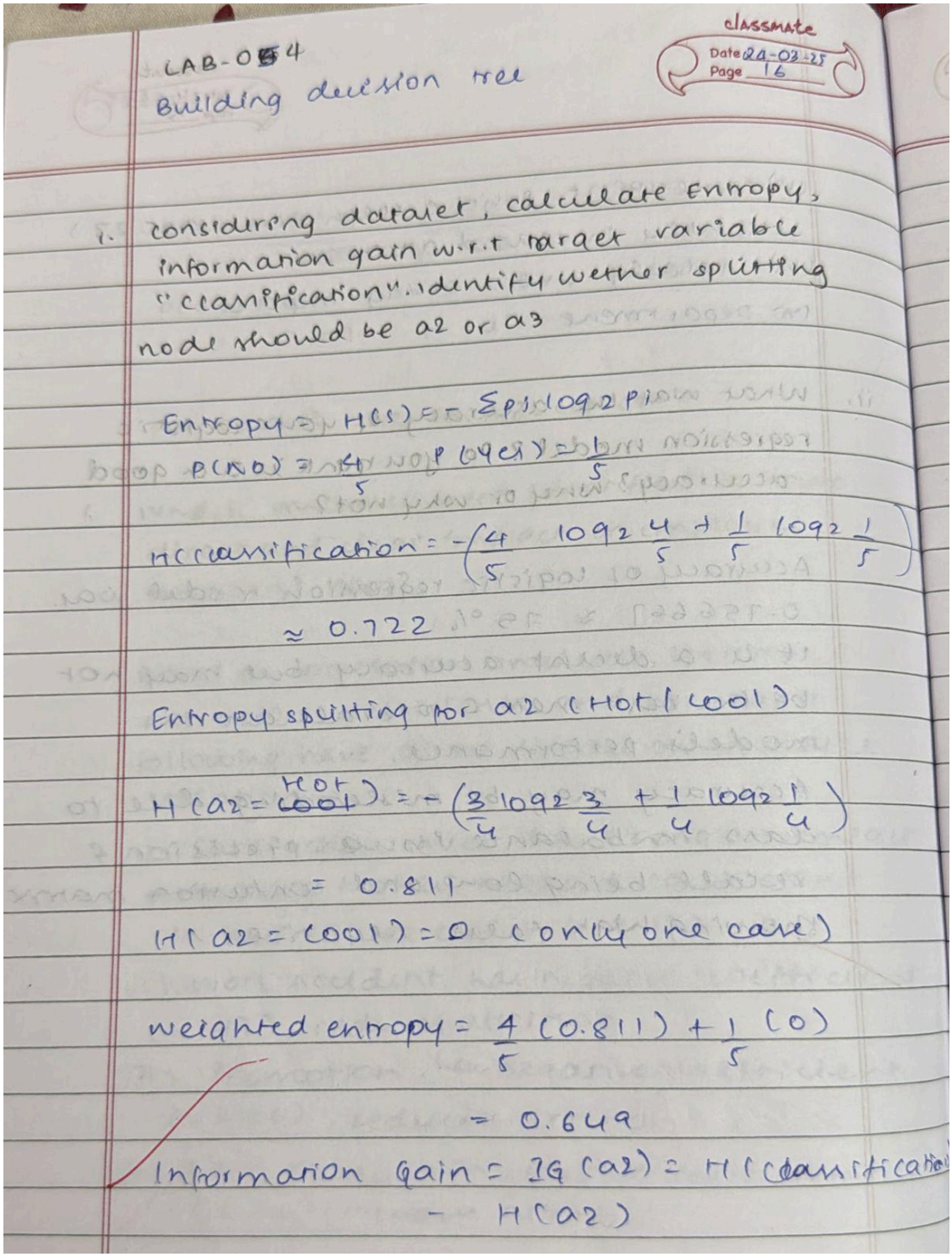
```
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))


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

age = 35
prediction_function(age)
```

```
0.3709834769552775
```

Program 5

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

Screenshot



= 0.722 - 0.649 = 0.073

Entropy splitting for a3 (High/Normal)
H(a3 = High) = 0 (only one case)
H(a3 = Normal) = 0 (only one case)
Weighted entropy,

$$H(a3) = \frac{4}{5}(0) + \frac{1}{5}(0) = 0$$

Information Gain, IG(a3) = H(classification) - H(a3)

= 0.722 - 0 = 0.722

Since IG(a3) > IG(a2) we choose a3 as splitting attribute

24/2

For 'iris.csv' dataset

1. What was accuracy score?
Accuracy score was 1.0

2. What does confusion matrix tell about model's performance? were there misclassifications? If so which classes are most confused?

Code:

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


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


data
```

```
{'a1': [True, True, False, False, False, True, True, True, False, False],
 'a2': ['Hot',
  'Hot',
  'Hot',
  'Cool',
  'Cool',
  'Cool',
  'Hot',
  'Hot',
  'Cool',
  'Cool'],
 'a3': ['High',
  'High',
  'High',
  'Normal',
  'Normal',
  'High',
  'High',
  'Normal',
  'Normal',
  'High'],
 'Classification': ['No',
  'No',
  'Yes',
  'Yes',
  'Yes',
  'No',
  'No',
  'Yes',
  'Yes',
  'Yes']}
```

```
df = pd.DataFrame(data)


label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le


X = df.drop('Classification', axis=1)
y = df['Classification']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


clf = DecisionTreeClassifier(criterion='entropy')


clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

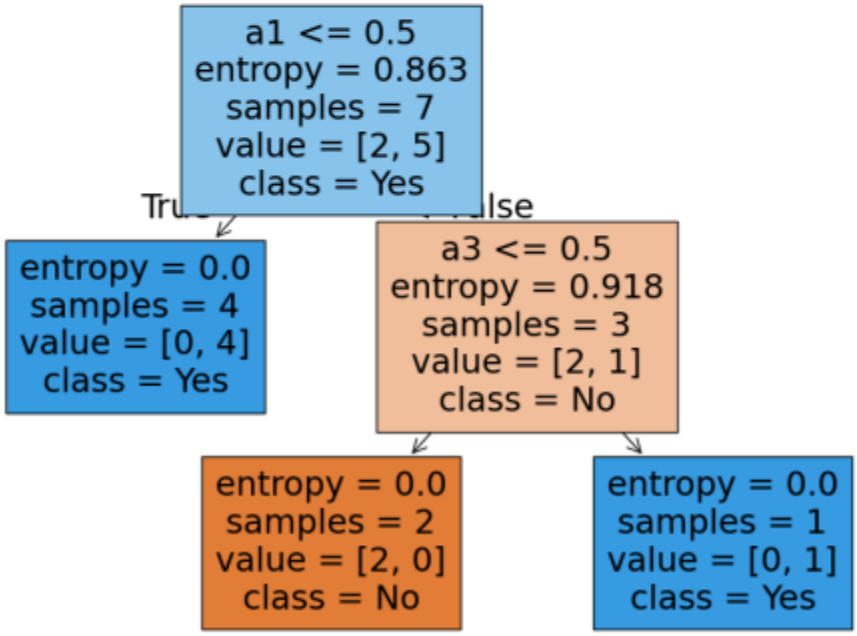
```
y_pred = clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))
```

```
Accuracy: 1.00
              precision    recall  f1-score   support

          No       1.00      1.00      1.00         2
         Yes       1.00      1.00      1.00         1

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

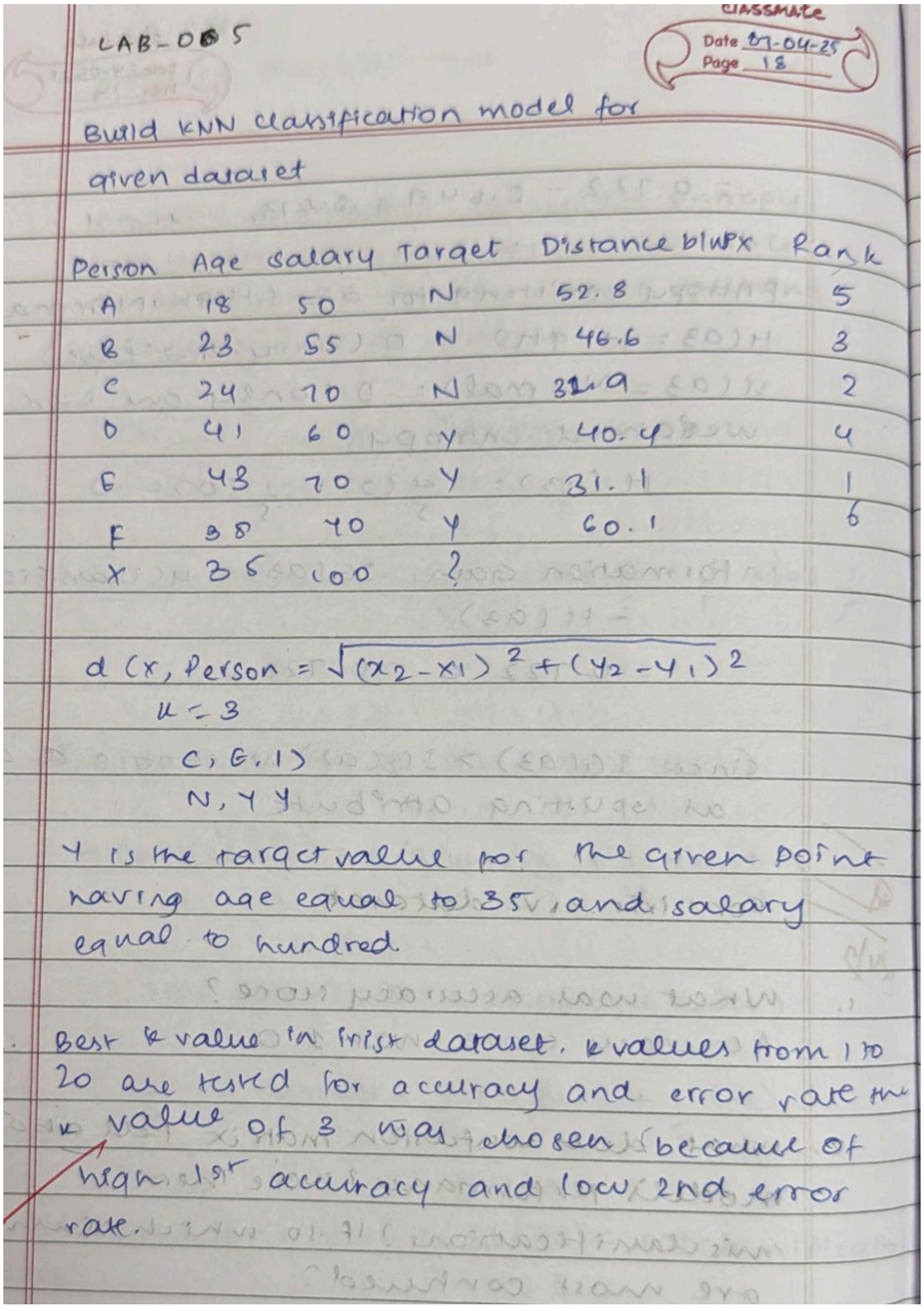
```
plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])
plt.show()
```



Program 6

Build KNN Classification model for a given dataset.

Screenshot



LAB-05

Date 07-04-25
Page 18

Build KNN classification model for given dataset

Person	Age	salary	Target	Distance b/w X	Rank
A	18	50	N	52.8	5
B	23	55	N	46.6	3
C	24	70	N	32.9	2
D	41	60	Y	40.4	4
E	43	70	Y	31.1	1
F	38	40	Y	60.1	6
X	35	100	?		

$d(x, Person) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

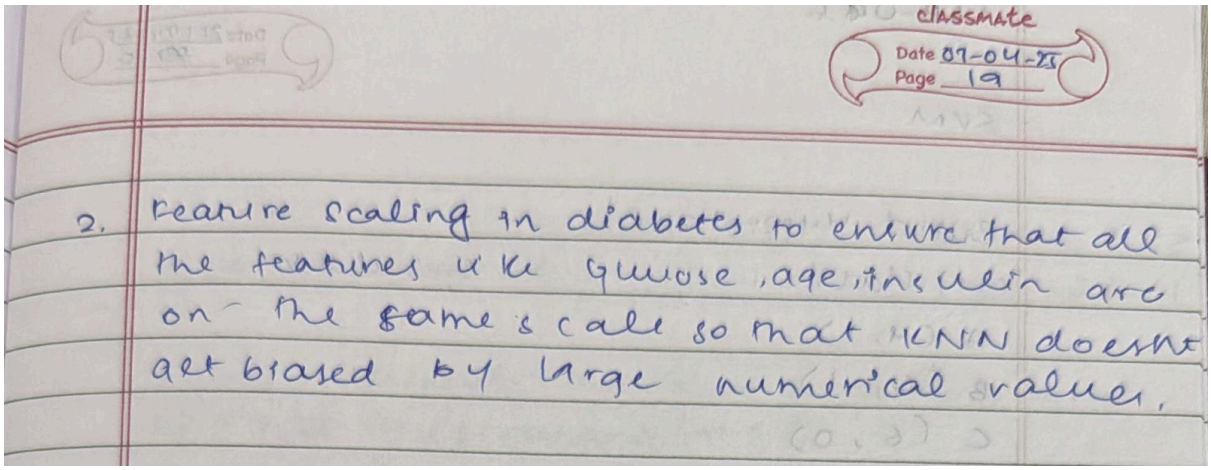
$k = 3$

C, E, 1)

N, Y Y

Y is the target value for the given point having age equal to 35, and salary equal to hundred.

Best k value in first dataset. k values from 1 to 20 are tested for accuracy and error rate the k value of 3 was chosen because of high 1st accuracy and low 2nd error rate.



Code:

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


from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving iris.csv to iris.csv
```

```
df = pd.read_csv('iris.csv')
df
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

```
df['species'] = df['species'].astype('category').cat.codes


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


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


param_grid = {'n_neighbors': list(range(1, 21))}
knn = KNeighborsClassifier()
grid = GridSearchCV(knn, param_grid, cv=5)
grid.fit(X_train, y_train)
```

```
GridSearchCV
best_estimator_:
KNeighborsClassifier
KNeighborsClassifier
```

```
best_k = grid.best_params_['n_neighbors']
print(f"Best k value: {best_k}")
```

```
Best k value: 3
```

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

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred = knn.predict(X_test)
```

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

```
Accuracy Score: 1.00
```

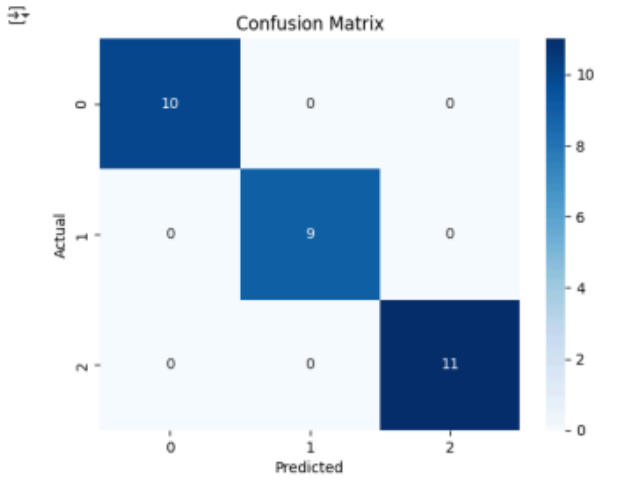
```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

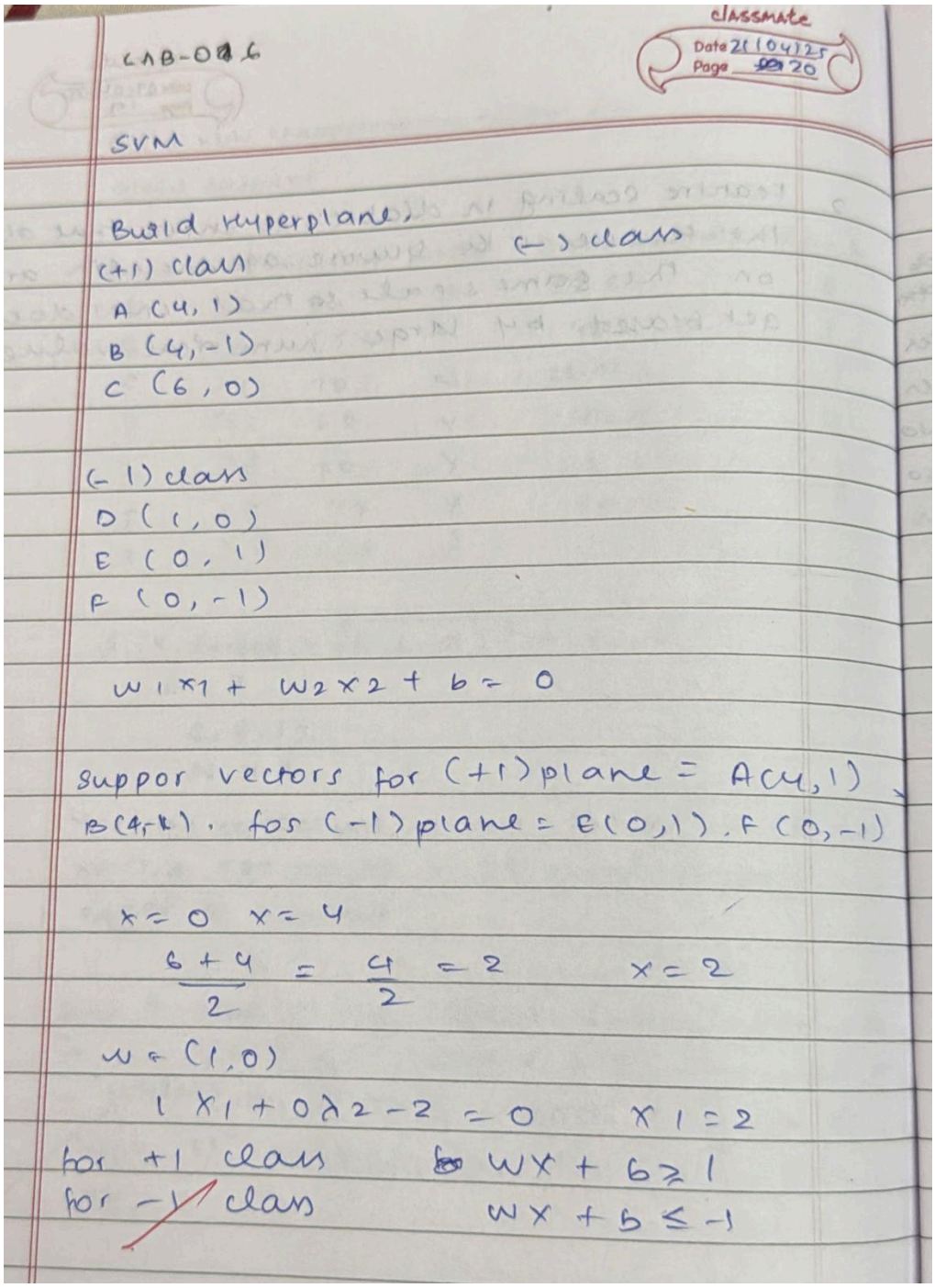
```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Program 7

Build Support vector machine model for a given dataset

Screenshot



A(4,1) = wx+b = 1(4)+0(1)-2 = 2 ≥ 1

E(0,1) = wx+b = 1(0)+0(1)-2 = -2 ≤ -1

Margin boundaries x=1 and x=3 and optimal hyperplane x=2.

1. In Iris dataset the accuracy for both of them is 100% ∴ both performed equally well. as a result one cannot be chosen over the other as they don't give 100% accuracy.

2. In letter recognition.csv : Yes I and P, S and Z, V and W, Y and R, K and R are some of the examples where in the model is getting confused. Area under the were = 0.99 > 0.95 ∴ the model is excellent at seperating the classes across thresholds. In Iris it is 100% accuracy. & for letter recognition, the accuracy is 93% or 0.9305, it performed better on Iris dataset than on other datasets.

Code:

```
import pandas as pd
from sklearn.datasets import load_digits
digits = load_digits()
digits.target
dir(digits)
digits.target_names
df = pd.DataFrame(digits.data,digits.target)
df.head()
df['target'] = digits.target
df.head(20)
```

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	target
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4
5	0.0	0.0	12.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	9.0	16.0	16.0	10.0	0.0	0.0	5
6	0.0	0.0	0.0	12.0	13.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	9.0	15.0	11.0	3.0	0.0	6
7	0.0	0.0	7.0	8.0	13.0	16.0	15.0	1.0	0.0	0.0	...	0.0	0.0	0.0	13.0	5.0	0.0	0.0	0.0	0.0	7
8	0.0	0.0	9.0	14.0	8.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	11.0	16.0	15.0	11.0	1.0	0.0	8
9	0.0	0.0	11.0	12.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	9.0	12.0	13.0	3.0	0.0	0.0	9
0	0.0	0.0	1.0	9.0	15.0	11.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	10.0	13.0	3.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	14.0	13.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	13.0	16.0	1.0	0.0	1
2	0.0	0.0	5.0	12.0	1.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	3.0	11.0	8.0	13.0	12.0	4.0	2
3	0.0	2.0	9.0	15.0	14.0	9.0	3.0	0.0	0.0	4.0	...	0.0	0.0	2.0	12.0	12.0	13.0	11.0	0.0	0.0	3
4	0.0	0.0	0.0	8.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	10.0	15.0	4.0	0.0	0.0	4
5	0.0	5.0	12.0	13.0	16.0	16.0	2.0	0.0	0.0	11.0	...	0.0	0.0	4.0	15.0	16.0	2.0	0.0	0.0	0.0	5
6	0.0	0.0	0.0	8.0	15.0	1.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	7.0	15.0	16.0	11.0	0.0	6
7	0.0	0.0	1.0	8.0	15.0	10.0	0.0	0.0	0.0	3.0	...	0.0	0.0	0.0	0.0	11.0	9.0	0.0	0.0	0.0	7
8	0.0	0.0	10.0	7.0	13.0	9.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	11.0	14.0	5.0	0.0	0.0	0.0	8
9	0.0	0.0	6.0	14.0	4.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	7.0	16.0	16.0	13.0	11.0	1.0	9

20 rows × 65 columns

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.drop('target',axis='columns'), df.target, test_size=0.3)
```

```
from sklearn.svm import SVC
rbf_model = SVC(kernel='rbf')
len(X_train)
len(X_test)
rbf_model.fit(X_train, y_train)
rbf_model.score(X_test,y_test)
```

```
0.9907407407407407
```

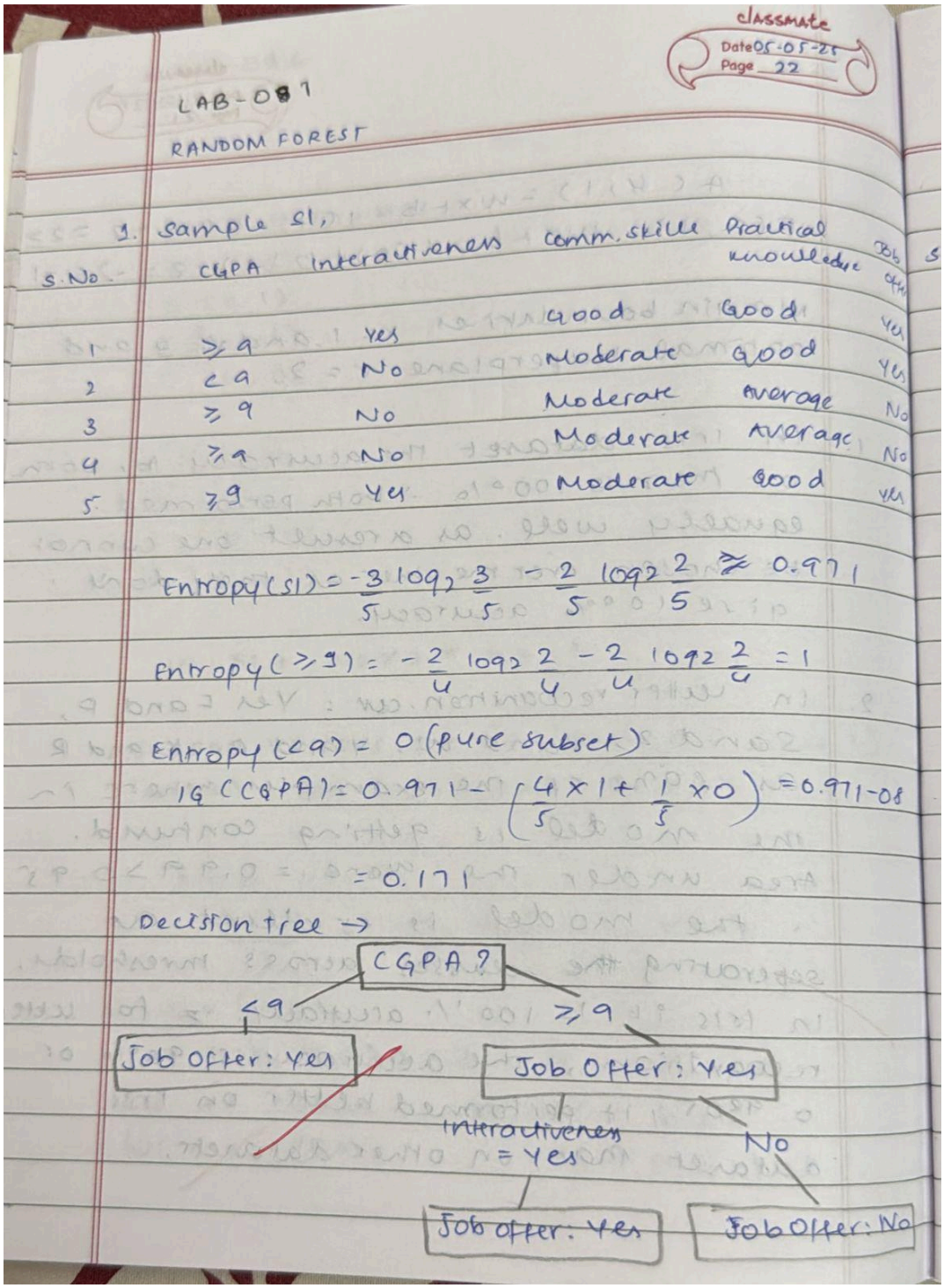
```
linear_model = SVC(kernel='linear')
linear_model.fit(X_train,y_train)
linear_model.score(X_test,y_test)
```

```
0.9833333333333333
```

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot



Date 05-05-25
Page 23

2. sample S2,

S.No.	CGPA	Interactiveness	comm. skills	Practical knowledge	Job offer
1	< 9	No	Moderate	Good	Yes
3	≥ 9	No	Moderate	Avg	No
8	≥ 9	No	Moderate	Avg	No
5	≥ 9	Yes	Moderate	good	Yes
5	≥ 9	Yes	Moderate	Good	Yes

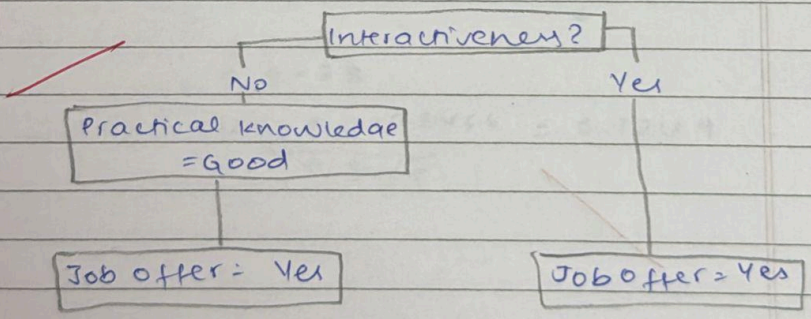
$$\text{Entropy}(S) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} \approx 0.971$$

$$\text{Entropy(Interactiveness)} = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}$$

$$\approx 0.918.$$

$$IG(\text{Interactiveness}) = 0.971 - \left(\frac{2}{5}\times 0 + \frac{3}{5}\times 0.918\right)$$

$$\approx 0.971 - 0.551 = 0.42$$



Date 05-05-25
Page 24

1. What is the best accuracy score for dataset & confusion matrix of classifier observed & how many trees used?

Best accuracy = 1.000 with 1 tree

Confusion Matrix = $\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$

Code:

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


from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving iris (1).csv to iris (1).csv
```

```
df = pd.read_csv("iris (1).csv", encoding='latin1')

print("Columns:", df.columns)

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

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

```
Columns: Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

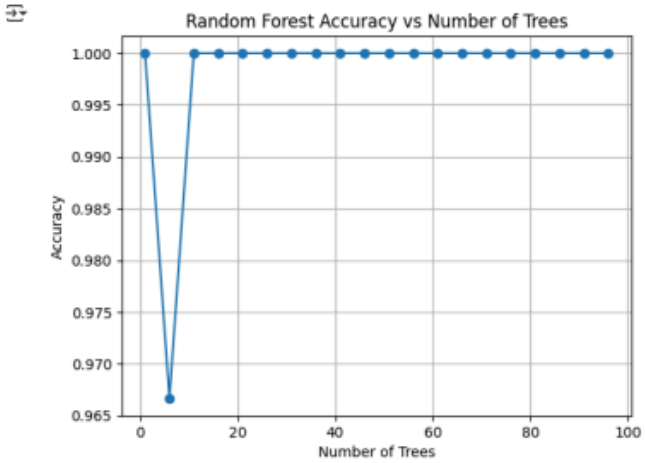
```
# 1. Train default Random Forest (n_estimators = 10)
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)

print(f"Default RF Accuracy (10 trees): {default_score:.4f}")
```

```
Default RF Accuracy (10 trees): 1.0000
```

```
# 2. Tune number of trees
scores = []
tree_range = range(1, 101, 5)

for n in tree_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    scores.append(acc)

best_score = max(scores)
best_n = tree_range[scores.index(best_score)]

print(f"Best RF Accuracy: {best_score:.4f} with {best_n} trees")
```

```
Best RF Accuracy: 1.0000 with 1 trees
```

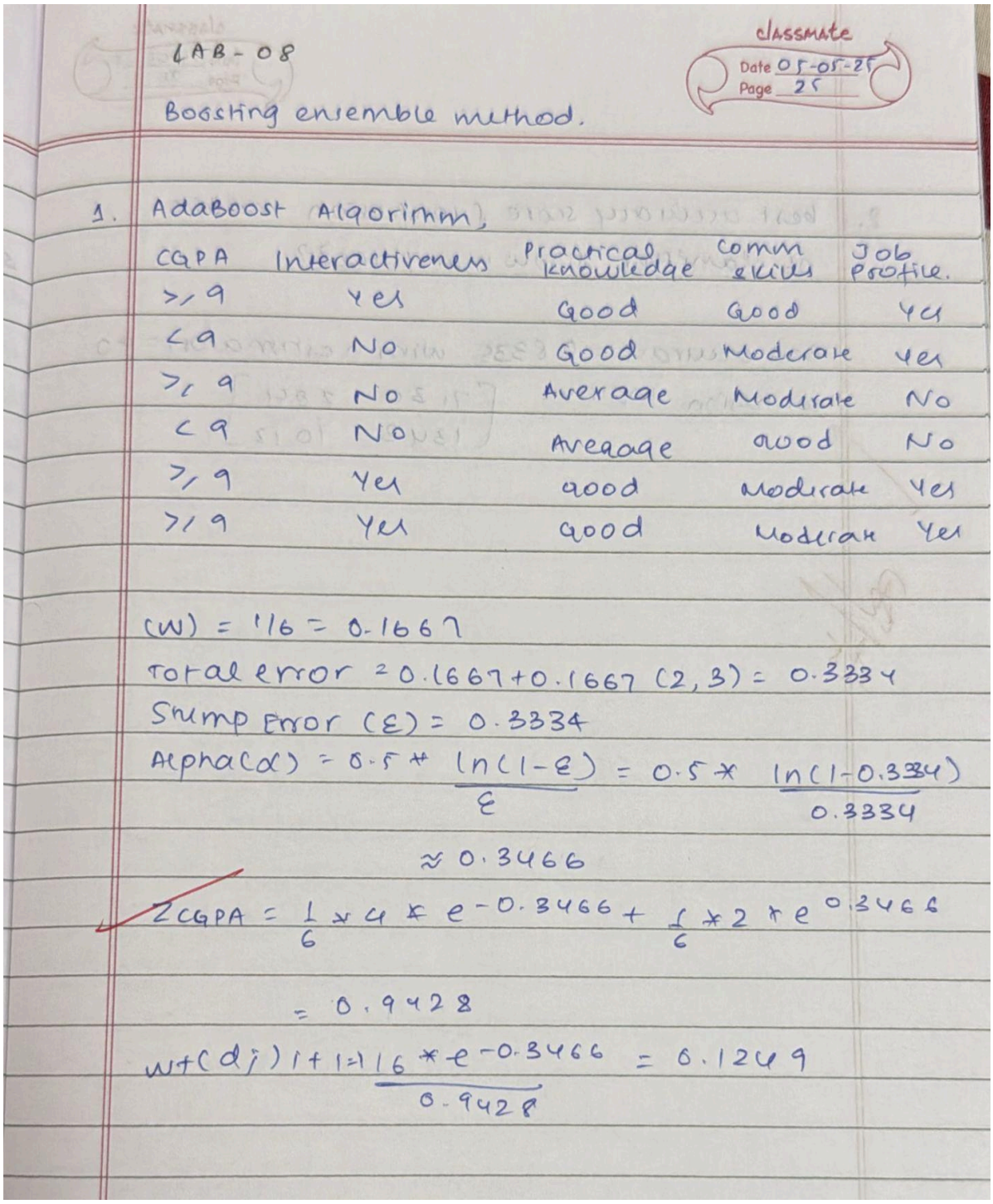
```
# Plot results
plt.plot(tree_range, scores, marker='o')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot



LAB - 08

Boosting ensemble method.

Date 05-05-25
Page 25

1. AdaBoost Algorithm,

CGPA	Interactiveness	Practical knowledge	Comm skills	Job profile.
>/ 9	Yes	Good	Good	Yes
< 9	No	Good	Moderate	Yes
>/ 9	No	Average	Moderate	No
< 9	No	Average	good	No
>/ 9	Yes	good	Moderate	Yes
>/ 9	Yes	good	Moderate	Yes

(w) = 1/6 = 0.1667

Total error = 0.1667 + 0.1667 (2, 3) = 0.3334

Sum Error (ε) = 0.3334

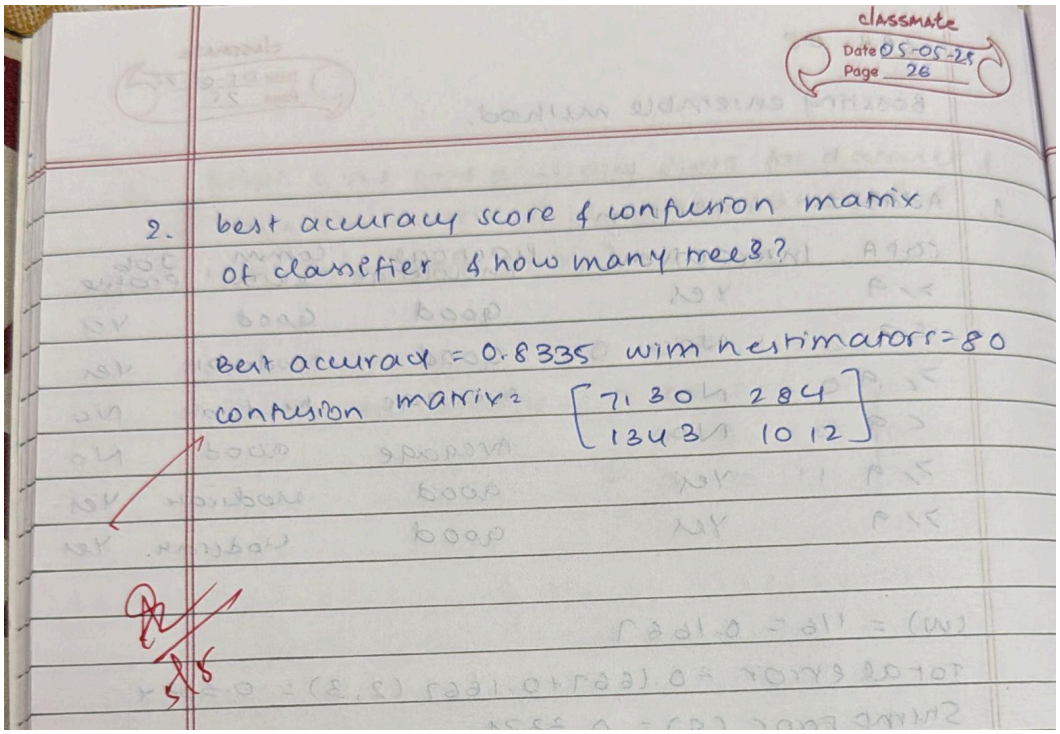
Alpha(α) = 0.5 * $\frac{\ln(1-\varepsilon)}{\varepsilon}$ = 0.5 * $\frac{\ln(1-0.3334)}{0.3334}$

≈ 0.3466

ΣCGPA = $\frac{1}{6} * 4 * e^{-0.3466} + \frac{1}{6} * 2 * e^{0.3466}$

= 0.9428

wt(dj)1+1 = $\frac{1/6 * e^{-0.3466}}{0.9428}$ = 0.1249


classmate
Date 05-05-25
Page 26
2. best accuracy score & confusion matrix of classifier & how many trees?
Best accuracy = 0.8335 with n_estimators=80
confusion matrix: [[7130 284] [1343 1012]]

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
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
from sklearn.datasets import load_iris, load_diabetes
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving income.csv to income.csv
```

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

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	income_level
0	39	77516	13	2174	0	40	0
1	50	83311	13	0	0	13	0
2	38	215646	9	0	0	40	0
3	53	234721	7	0	0	40	0
4	28	338409	13	0	0	40	0
...	...	...	...	...	...	...	...
48837	39	215419	13	0	0	36	0
48838	64	321403	9	0	0	40	0
48839	38	374983	13	0	0	50	0
48840	44	83891	13	5455	0	40	0
48841	35	182148	13	0	0	60	1

48842 rows × 7 columns

```
X = df.drop("income_level", axis=1)
y = df["income_level"]
```

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

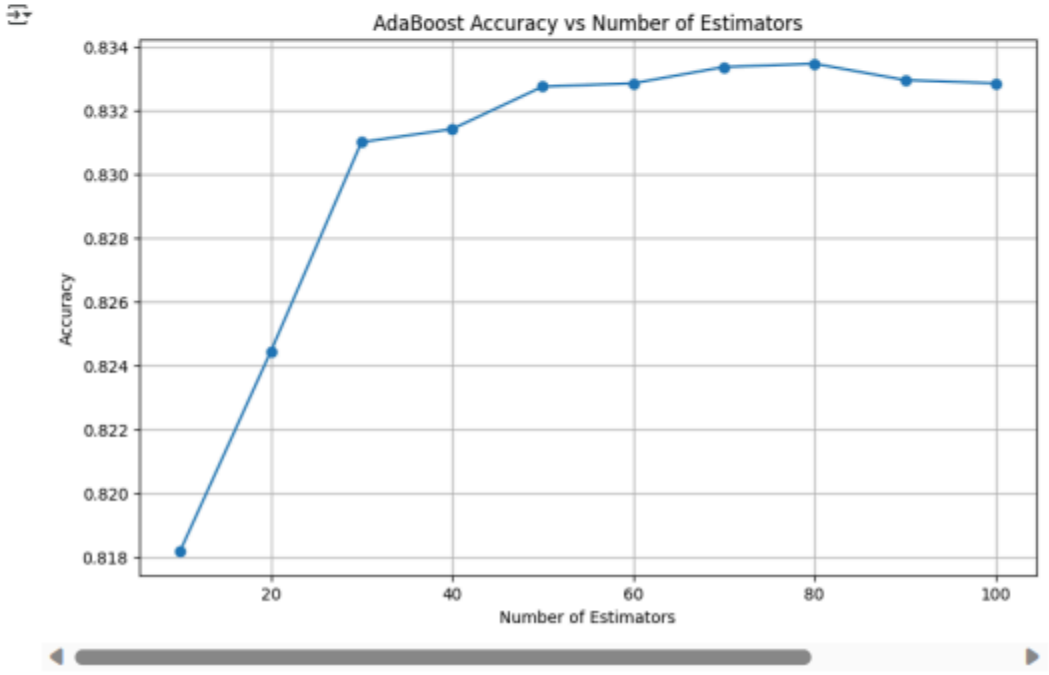
```
model_10 = AdaBoostClassifier(n_estimators=10, random_state=42)
model_10.fit(X_train, y_train)
y_pred_10 = model_10.predict(X_test)
score_10 = accuracy_score(y_test, y_pred_10)
print(f"Accuracy with 10 estimators: {score_10:.4f}")
```

```
Accuracy with 10 estimators: 0.8182
```

```
estimator_range = range(10, 101, 10)
scores = []

for n in estimator_range:
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    scores.append(acc)
    print(f"n_estimators={n}, Accuracy={acc:.4f}")
```

```
n_estimators=10, Accuracy=0.8182
n_estimators=20, Accuracy=0.8244
n_estimators=30, Accuracy=0.8310
n_estimators=40, Accuracy=0.8314
n_estimators=50, Accuracy=0.8327
n_estimators=60, Accuracy=0.8328
n_estimators=70, Accuracy=0.8334
n_estimators=80, Accuracy=0.8335
n_estimators=90, Accuracy=0.8329
n_estimators=100, Accuracy=0.8328
```

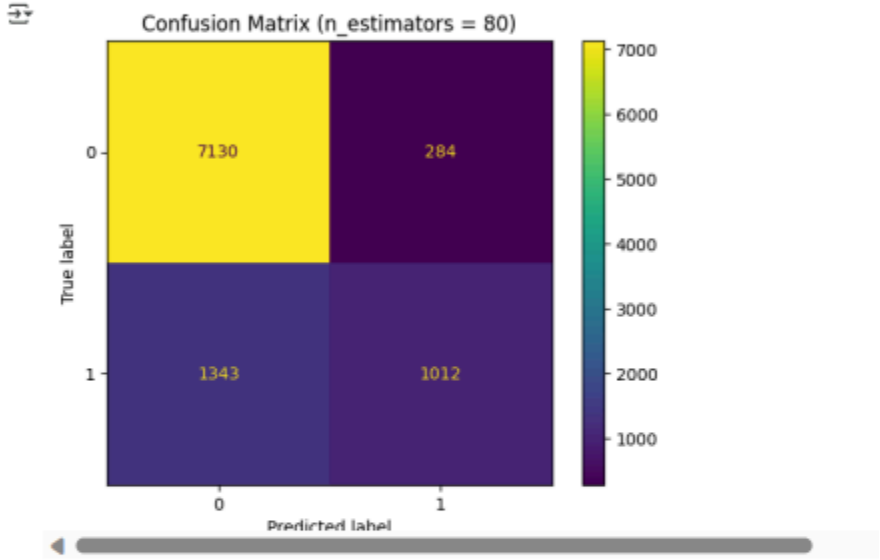
```
plt.figure(figsize=(10,6))
plt.plot(estimator_range, scores, marker='o')
plt.title("AdaBoost Accuracy vs Number of Estimators")
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```



```
best_n = estimator_range[scores.index(max(scores))]
best_score = max(scores)
print(f"\nBest Accuracy: {best_score:.4f} with n_estimators={best_n}")
```

```
Best Accuracy: 0.8335 with n_estimators=80
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
best_model = AdaBoostClassifier(n_estimators=80, random_state=42)
best_model.fit(X_train, y_train)
y_best_pred = best_model.predict(X_test)
cm = confusion_matrix(y_test, y_best_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix (n_estimators = 80)")
plt.show()
```




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

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

df_encoded = df.copy()
for col in df.columns:
    if df[col].dtype == 'object':
        df_encoded[col] = LabelEncoder().fit_transform(df[col])

X = df_encoded.drop("income_level", axis=1)
y = df_encoded["income_level"]

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

# Base AdaBoost with 10 estimators
model = AdaBoostClassifier(n_estimators=10, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluation
print("Accuracy (10 trees):", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

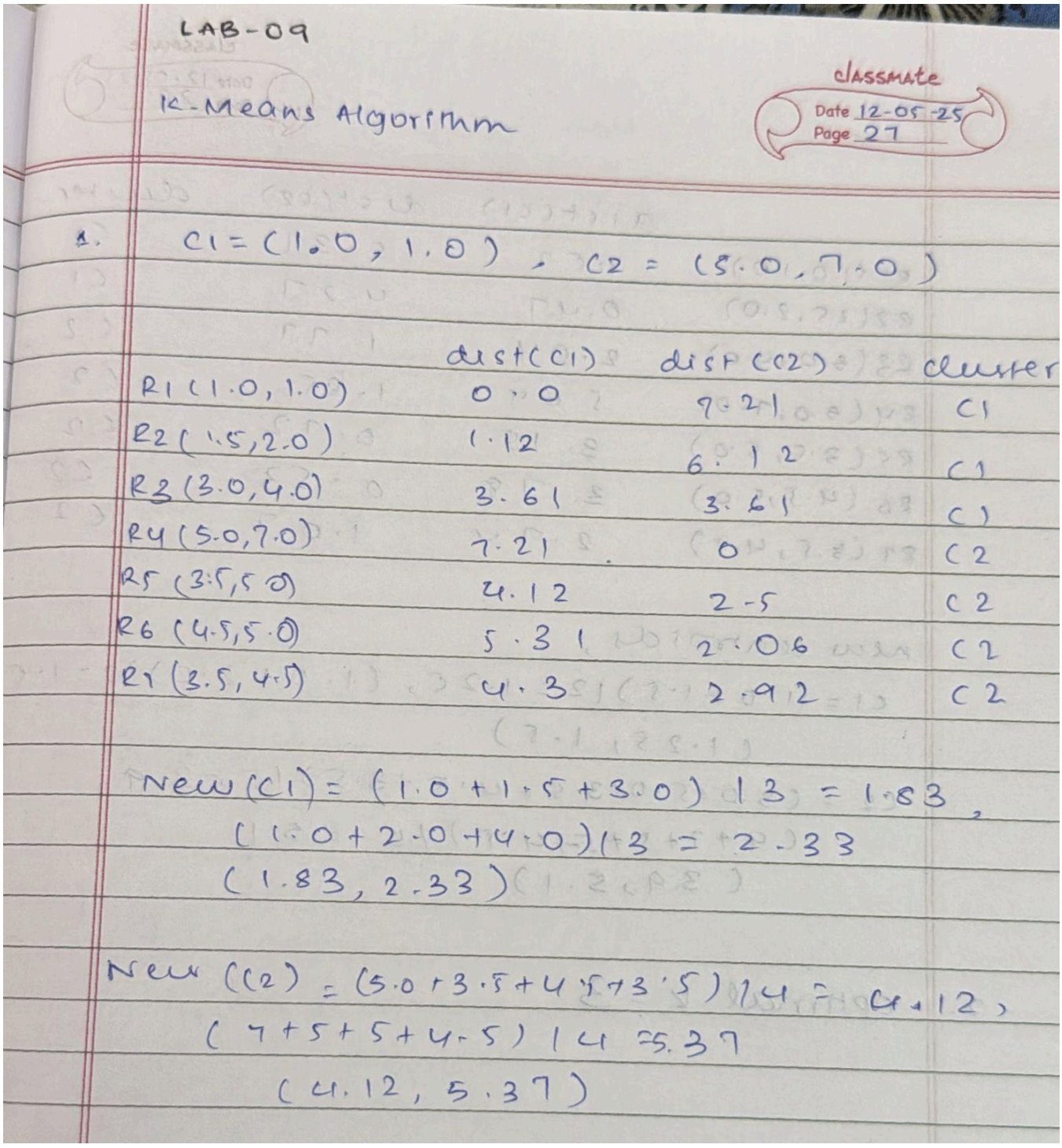
```
# Fine tuning
best_score = 0
best_tree = 0
for n in range(10, 101, 10):
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    score = accuracy_score(y_test, model.predict(X_test))
    if score > best_score:
        best_score = score
        best_tree = n

print(f"\nBest Score: {best_score} using {best_tree} trees")
```

```
Accuracy (10 trees): 0.8182004299314157
Confusion Matrix:
 [[6782  632]
 [1144 1211]]

Best Score: 0.8334527587265841 using 80 trees
```

Program 10

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

Screenshot



CLASSMATE
Date 12-05-25
Page 28

	dist(c1)	dist(c2)	cluster
R1 (1.0, 1.0)	1.57	5.37	C1
R2 (1.5, 2.0)	0.47	4.27	C1
R3 (3.0, 4.0)	2.04	1.77	C2
R4 (5.0, 7.0)	5.64	1.85	C2
R5 (3.5, 5.0)	3.15	0.72	C2
R6 (4.5, 5.0)	3.78	0.53	C2
R7 (3.5, 4.5)	2.74	1.07	C2

new centroids,

c1 = (1.0 + 1.5)/2 = 1.25, (1.0 + 2.0)/2 = 1.5

(1.25, 1.5)

c2 = (3.0 + 5.0 + 3.5 + 4.5 + 3.5)/5 = 3.9,

(4 + 7 + 5 + 5 + 4.5)/5 = 5.1

(3.9, 5.1)

2. optimal k = 3

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
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
from sklearn.datasets import load_iris, load_diabetes
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt


from google.colab import files
uploaded = files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving iris (1).csv to iris (1).csv
```

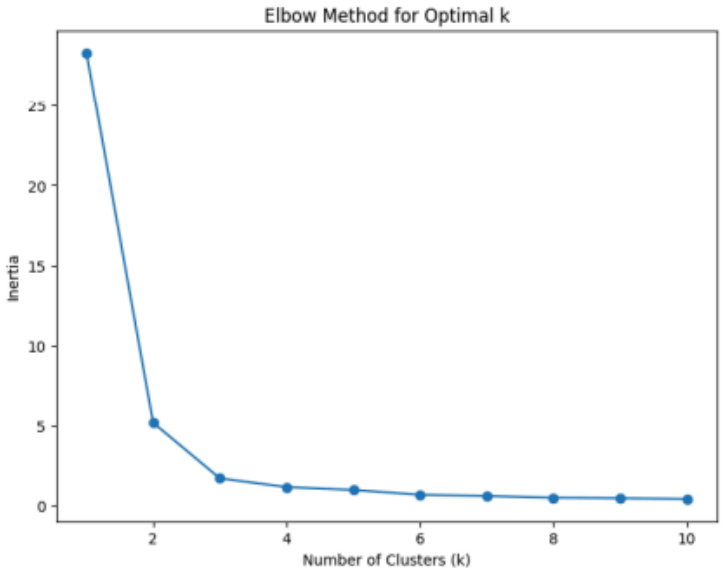
```
df = pd.read_csv('iris (1).csv')
df
```

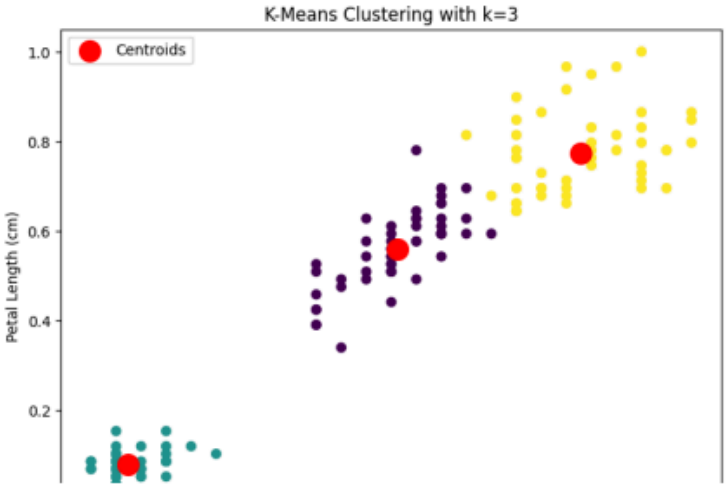
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

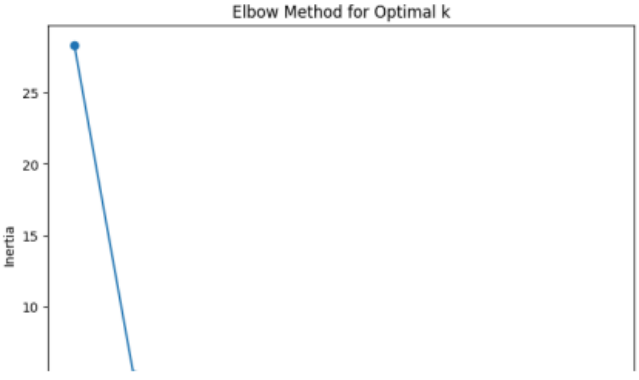
```
df['target'] = iris.target

# Select petal width and length features
df = df[['petal width (cm)', 'petal length (cm)', 'target']]

# Scale the features using MinMaxScaler
scaler = MinMaxScaler()
df[['petal width (cm)', 'petal length (cm)']] = scaler.fit_transform(df[['petal width (cm)', 'petal length (cm

# Calculate inertia (sum of squared distances to the nearest cluster center) for different k values
inertia = []
for k in range(1, 11):
  kmeans = KMeans(n_clusters=k, random_state=0)
  kmeans.fit(df[['petal width (cm)', 'petal length (cm)']])
  inertia.append(kmeans.inertia_)

# Plot the elbow plot to find the optimal number of clusters
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.show()

# Based on the elbow plot, choose the optimal k value (e.g., k=3 in this case)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=0)
df['cluster'] = kmeans.fit_predict(df[['petal width (cm)', 'petal length (cm)']])

# Visualize the clusters
plt.figure(figsize=(8, 6))
plt.scatter(df['petal width (cm)'], df['petal length (cm)'], c=df['cluster'])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', label='Centroids')
plt.xlabel('Petal Width (cm)')
plt.ylabel('Petal Length (cm)')
plt.title(f'K-Means Clustering with k={optimal_k}')
plt.legend()
plt.show()
```






```
inertia = []
for k in range(1, 11):
  kmeans = KMeans(n_clusters=k, random_state=0)
  kmeans.fit(df[['petal width (cm)', 'petal length (cm)']])
  inertia.append(kmeans.inertia_)

# Plot the elbow plot to find the optimal number of clusters
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
```

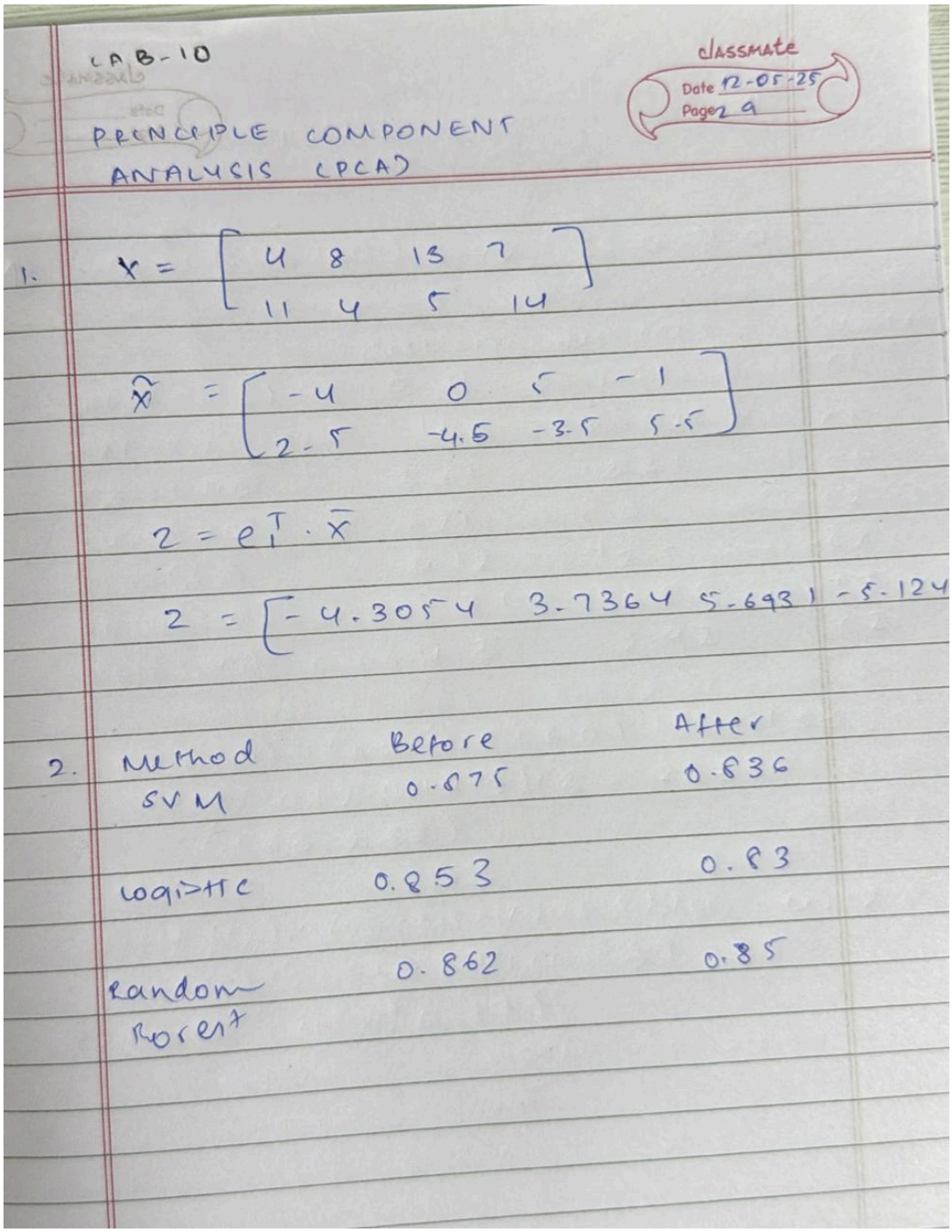
```
plt.show()

#optimal_k = 3
```



Program 11

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

Screenshot



Code:

```
import seaborn as sns
import matplotlib.pyplot as plt


from google.colab import files
uploaded = files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser
session. Please rerun this cell to enable.
Saving heart.csv to heart.csv
```

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

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak
0	40	M	ATA	140	289	0	Normal	172	N	0.0
1	49	F	NAP	160	180	0	Normal	156	N	1.0
2	37	M	ATA	130	283	0	ST	98	N	0.0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5
4	54	M	NAP	150	195	0	Normal	122	N	0.0
...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.2
914	68	M	ASY	144	193	1	Normal	141	N	3.4
915	57	M	ASY	130	131	0	Normal	115	Y	1.2
916	57	F	ATA	130	236	0	LVH	174	N	0.0
917	38	M	NAP	138	175	0	Normal	173	N	0.0

918 rows × 12 columns

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

```
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.datasets import load_iris, load_diabetes
from sklearn.model_selection import train_test_split, GridSearchCV

text_cols = df.select_dtypes(include=['object']).columns

# Label Encoding
label_encoder = LabelEncoder()
df_label_encoded = df.copy()
for col in text_cols:
    df_label_encoded[col] = label_encoder.fit_transform(df_label_encoded[col])

# One-Hot Encoding
df_onehot_encoded = pd.get_dummies(df, columns=text_cols)

scaler = StandardScaler()
numerical_cols = df.select_dtypes(include=['number']).columns
df_scaled = df.copy()
df_scaled[numerical_cols] = scaler.fit_transform(df[numerical_cols])

print(df_label_encoded.head())
print(df_onehot_encoded.head())
print(df_scaled.head())
```

```
   Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS  RestingECG  \
0   40    1              1        140          289          0           1
1   49    0              2        160          180          0           1
2   37    1              1        130          283          0           2
3   48    0              0        138          214          0           1
4   54    1              2        150          195          0           1

   MaxHR  ExerciseAngina  Oldpeak  ST_Slope  HeartDisease
0    172               0      0.0         2             0
1    156               0      1.0         1             1
2     98               0      0.0         2             0
3    108               1      1.5         1             1
4    122               0      0.0         2             0
   Age  RestingBP  Cholesterol  FastingBS  MaxHR  Oldpeak  HeartDisease  \
0   40        140          289          0    172      0.0             0
1   49        160          180          0    156      1.0             1
2   37        130          283          0     98      0.0             0
3   48        138          214          0    108      1.5             1
4   54        150          195          0    122      0.0             0

   Sex_F  Sex_M  ChestPainType_ASY  ...  ChestPainType_NAP  ChestPainType_TA  \
0  False   True              False  ...              False             False
1   True  False              False  ...               True             False
2  False   True              False  ...              False             False
3   True  False               True  ...              False             False
4  False   True              False  ...               True             False

   RestingECG_LVH  RestingECG_Normal  RestingECG_ST  ExerciseAngina_N  \
0           False               True          False              True
1           False               True          False              True
2           False              False           True              True
3           False               True          False             False
4           False               True          False              True

   ExerciseAngina_Y  ST_Slope_Down  ST_Slope_Flat  ST_Slope_Up
0             False          False          False         True
1             False          False           True        False
2             False          False          False         True
3              True          False           True        False
4             False          False          False         True

[5 rows x 21 columns]
       Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  \
```

```
      0 -1.433140   M           ATA   0.410909     0.825070  -0.551341     Normal
      1 -0.478484   F           NAP   1.491752    -0.171961  -0.551341     Normal
      2 -1.751359   M           ATA  -0.129513     0.770188  -0.551341         ST
      3 -0.584556   F           ASY   0.302825     0.139040  -0.551341     Normal
      4  0.051881   M           NAP   0.951331    -0.034755  -0.551341     Normal

            MaxHR ExerciseAngina   Oldpeak ST_Slope  HeartDisease
      0  1.382928              N -0.832432       Up     -1.113115
      1  0.754157              N  0.105664     Flat      0.898380
      2 -1.525138              N -0.832432       Up     -1.113115
      3 -1.132156              Y  0.574711     Flat      0.898380
      4 -0.581981              N -0.832432       Up     -1.113115
```

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

# Load the dataset
df = pd.read_csv("heart.csv")

# Step 1: Encode categorical columns
categorical_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
df_encoded = df.copy()

le = LabelEncoder()
for col in categorical_cols:
    df_encoded[col] = le.fit_transform(df_encoded[col])

# Step 2: Separate features and target
X = df_encoded.drop("HeartDisease", axis=1)
y = df_encoded["HeartDisease"]

# Step 3: Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: 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 5: Train models and evaluate accuracy
models = {
    "SVM": SVC(),
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "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: Apply PCA
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)

print("\n=== Accuracy with PCA (5 components) ===")
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}")
```

```
=== Accuracy without PCA ===
SVM: 0.8641
Logistic Regression: 0.8478
Random Forest: 0.8913

=== Accuracy with PCA (5 components) ===
SVM: 0.8533
Logistic Regression: 0.8424
Random Forest: 0.8315
```