VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Mohammed Uzair Obaid(1BM22CS159)

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 **Mohammed Uzair Obaid(1BM22CS159),** who is a 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.

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

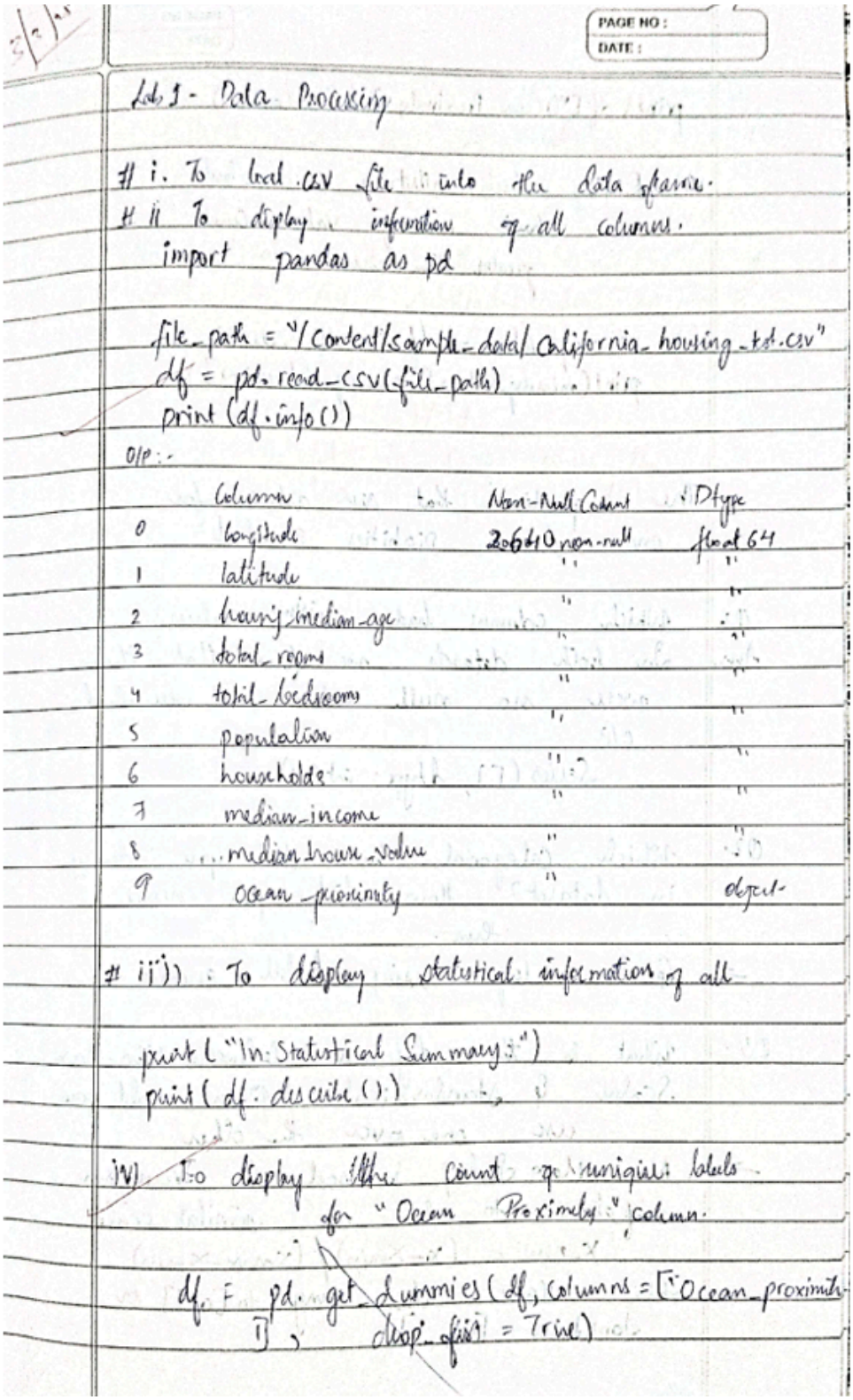
Index

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

Program 1

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

Observations:



Lab 1 - Data Processing

# i. To load .csv file into the data frame.
# ii To display information of all columns.
import pandas as pd

file_path = "/content/sample_data/california_housing_test.csv"
df = pd.read_csv(file_path)
print(df.info())

O/P:-

	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	...median house_value	"	"
9	ocean_proximity	"	object

iii)) : To display statistical information of all

print("\n Statistical Summary:")
print(df.describe())

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

df = pd.get_dummies(df, columns=['ocean_proximity'], drop_first=True)

DATE :

print(df ["Ocean Proximity"].value_counts())

iv) Display which attributes in a dataset have missing values count greater than zero.

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

The questions that are asked for on dataset Diabities & adult.

Q1. Which columns had missing values?

Ans. In both datasets we found that there were no null values that were found.

O/P

Series([], dtype: int64)

Q2. Which categorial columns did you indentify in dataset? How did you encode them.

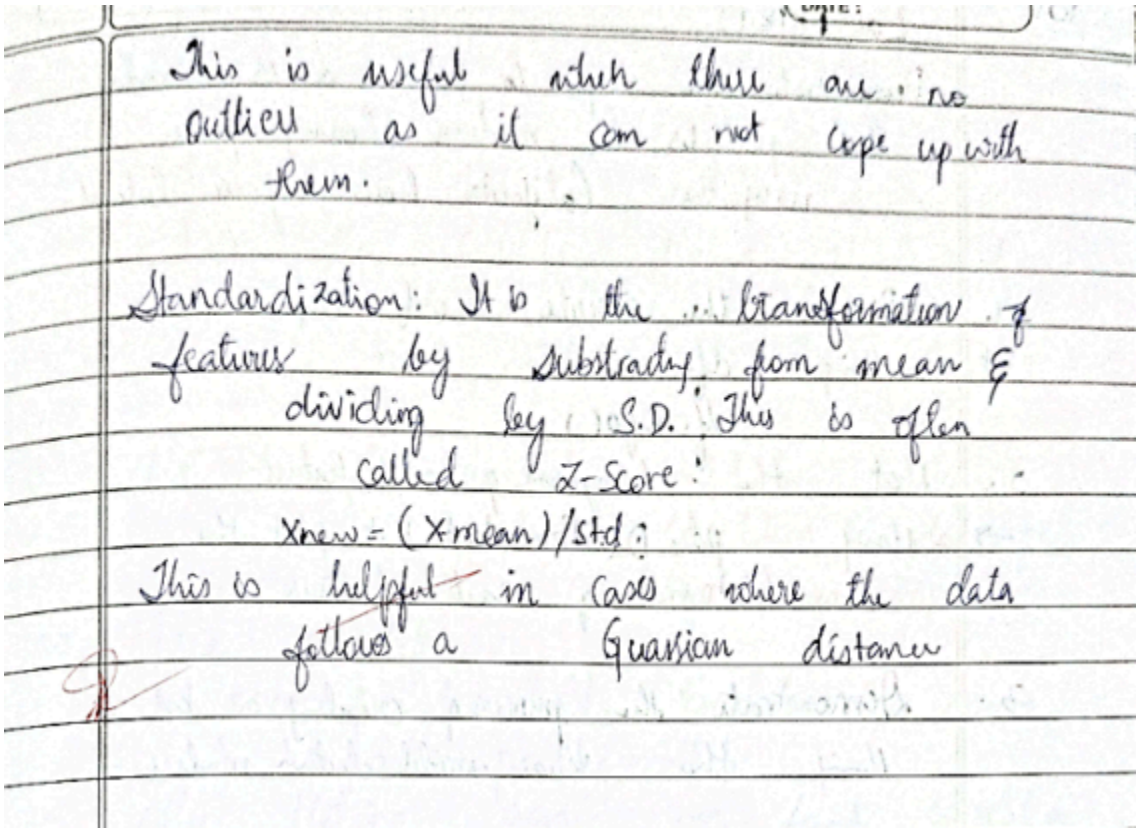
→ Gender by using label encoding

Q3. What is the difference between Min-Max Scalar & Standardization? When would you use one over the other.

→ Min-Max Scaling is used to transform features to be on similar scale

$x\_new = (x - x_{min}) / (x_{max} - x_{min})$

This scale the range to [0, 1] or sometimes [-1, 1]



Code:

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
import pandas as pd

# Define file path
file_path = "/content/sample_data/california_housing_test.csv"

# Load dataset
df = pd.read_csv(file_path)

# Display basic information
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           3000 non-null   float64
 1   latitude            3000 non-null   float64
 2   housing_median_age  3000 non-null   float64
 3   total_rooms         3000 non-null   float64
 4   total_bedrooms      3000 non-null   float64
 5   population          3000 non-null   float64
 6   households          3000 non-null   float64
 7   median_income       3000 non-null   float64
 8   median_house_value  3000 non-null   float64
dtypes: float64(9)
memory usage: 211.1 KB
None
```

In [3]:

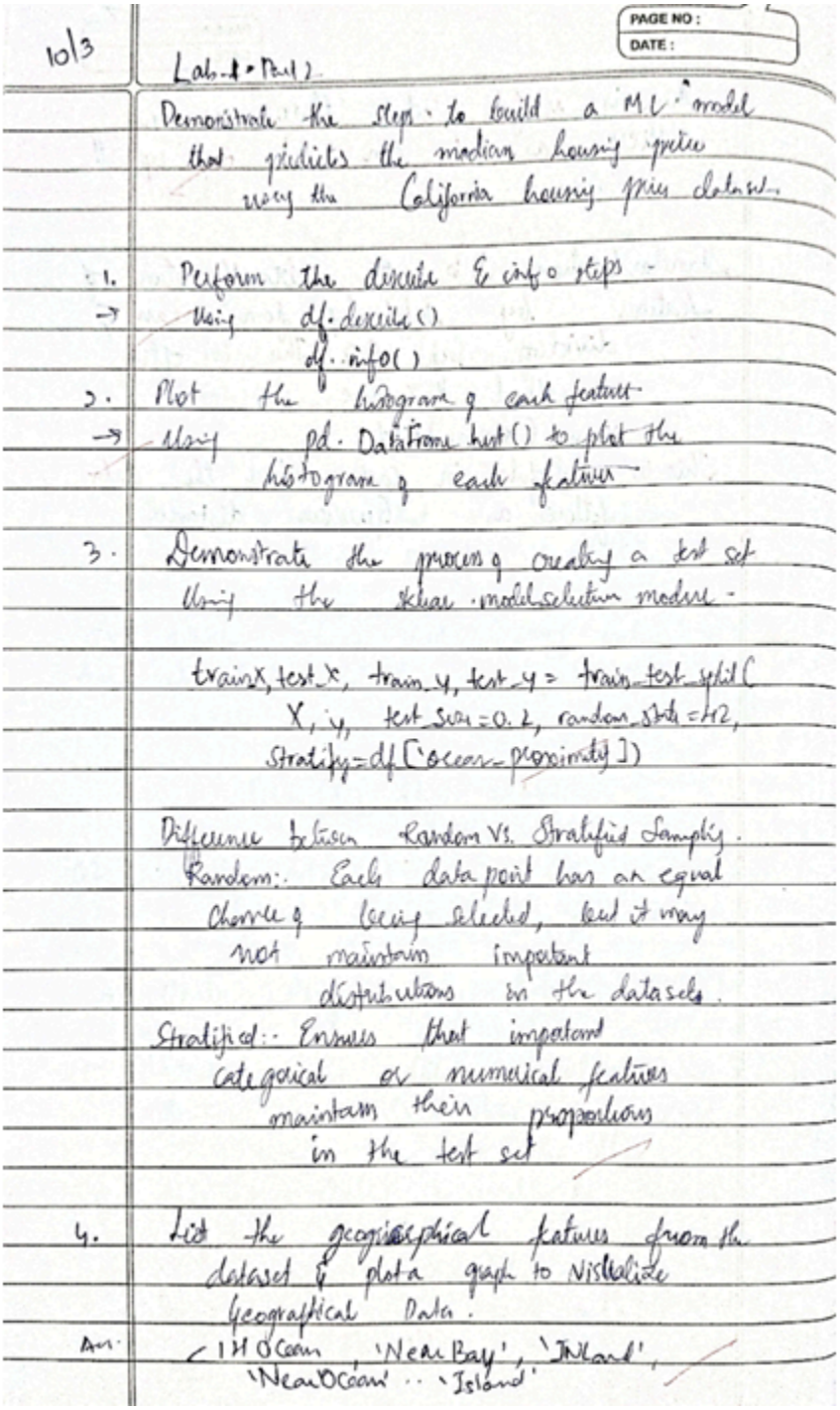
```
from sklearn.datasets import fetch_california_housing
import pandas as pd

# Load the dataset
data = fetch_california_housing(as_frame=True)
df = data.frame

# Display information of all columns
print(df.info())

# Display statistical information of all numerical columns
print(df.describe())

# Since "Ocean Proximity" is not in this dataset, adding a dummy categorical column
df["Ocean Proximity"] = ["NEAR BAY" if x < 1 else "INLAND" for x in df["MedInc"]]

# Display the count of unique labels for "Ocean Proximity" column
print(df["Ocean Proximity"].value_counts())

# Display which attributes (columns) in a dataset have missing values count greater than zero
print(df.isnull().sum()[df.isnull().sum() > 0])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
None
              MedInc      HouseAge      AveRooms     AveBedrms    Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.870671     28.639486      5.429000      1.096675   1425.476744
std        1.899822     12.585558      2.474173      0.473911   1132.462122
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.563400     18.000000      4.440716      1.006079    787.000000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

            AveOccup      Latitude     Longitude   MedHouseVal
count  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704      2.068558
std       10.386050      2.135952      2.003532      1.153956
min        0.692308     32.540000   -124.350000      0.149990
25%        2.429741     33.930000   -121.800000      1.196000
50%        2.818116     34.260000   -118.490000      1.797000
75%        3.282261     37.710000   -118.010000      2.647250
max     1243.333333     41.950000   -114.310000      5.000010
Ocean Proximity
INLAND      20485
NEAR BAY      155
Name: count, dtype: int64
Series([], dtype: int64)
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Observations:



10/3

PAGE NO :
DATE :

Lab-1 - Part 1

Demonstrate the steps to build a ML model that predicts the median housing price using the California housing price dataset.

1. Perform the describe & info steps
→ Using df.describe()
df.info()

2. Plot the histogram of each feature
→ Using pd.DataFrame.hist() to plot the histogram of each feature

3. Demonstrate the process of creating a test set Using the sklearn.model_selection module.

```
train_X, test_X, train_y, test_y = train_test_split(
    X, y, test_size=0.2, random_state=42,
    stratify=df['ocean_proximity'])
```

Difference between Random Vs. Stratified Sampling.
Random:. Each data point has an equal chance of being selected, but it may not maintain important distributions in the datasets.
Stratified:. Ensures that important categorical or numerical features maintain their proportions in the test set

4. List the geographical features from the dataset & plot a graph to visualize Geographical Data.
Ans: ['1H Ocean', 'Near Bay', 'Inland', 'Near Ocean', 'Island']

5. Find the most correlated column & plot a scatter plot.

→ - 'ocean_proximity' column is converted to numerical values using pd.get_dummies. then found correlation matrix.
- From the correlation matrix, it is found that median_income is ~~to be~~ correlated most with median_house_value.

6. List the features that could be combined to improve correlation & plot again to see if correlation has improved.

→ These new features
: rooms_per_household, bedrooms_per_room, & population_per_household
After plotting again we found correlation has not improved.

7. List the features that needs to be cleaned & demonstrate the process of cleaning

→ 'total_bedrooms' needs to cleaned.
Since it has 207 missing values.
Cleaning can be done using
- fillna method using median value.

8 Is there any categorical data that needs to converted to numerical ?

→ No

9. Importance of Feature Scaling

→ Scaling ensures that all the per features contributes equally to model training.

Code

In [35]:
```
import pandas as pd

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

In [6]:
```
df.describe()
```

Out[6]:

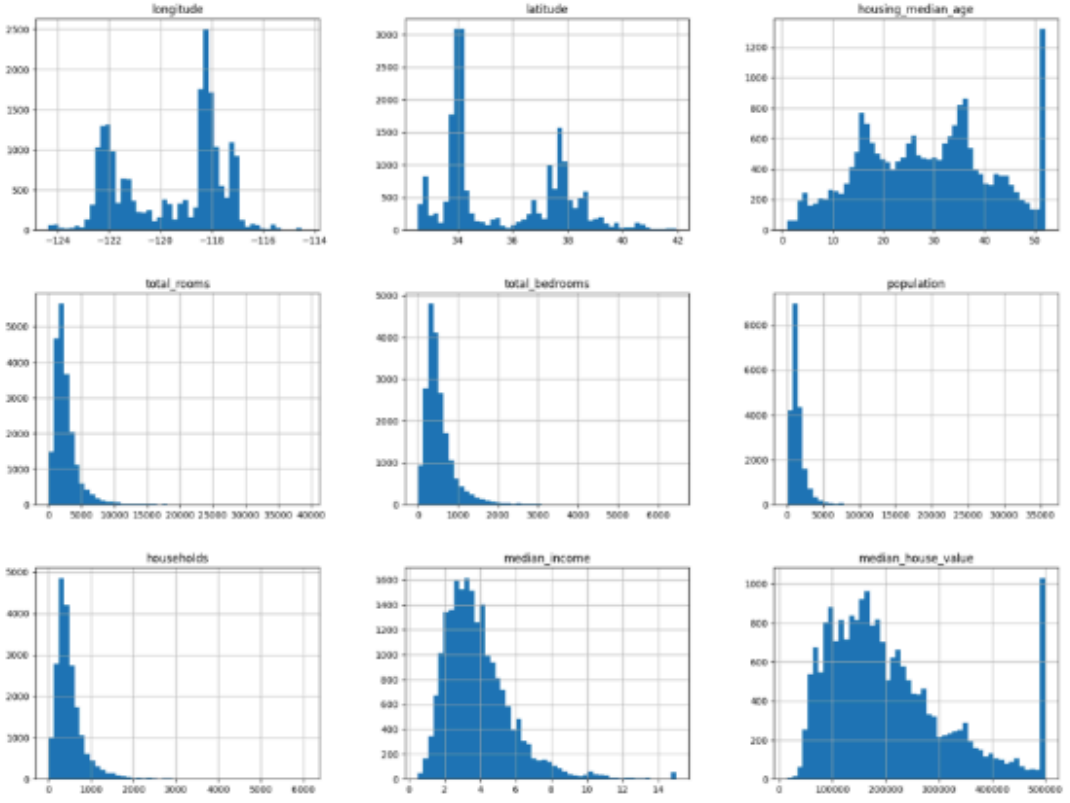
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [8]:
```
df.hist(bins=50,figsize=(20,15))
```

Out[8]:
```
array([[<Axes: title={'center': 'longitude'}>,
        <Axes: title={'center': 'latitude'}>,
        <Axes: title={'center': 'housing_median_age'}>],
       [<Axes: title={'center': 'total_rooms'}>,
        <Axes: title={'center': 'total_bedrooms'}>,
        <Axes: title={'center': 'population'}>],
       [<Axes: title={'center': 'households'}>,
        <Axes: title={'center': 'median_income'}>,
        <Axes: title={'center': 'median_house_value'}>]], dtype=object)
```

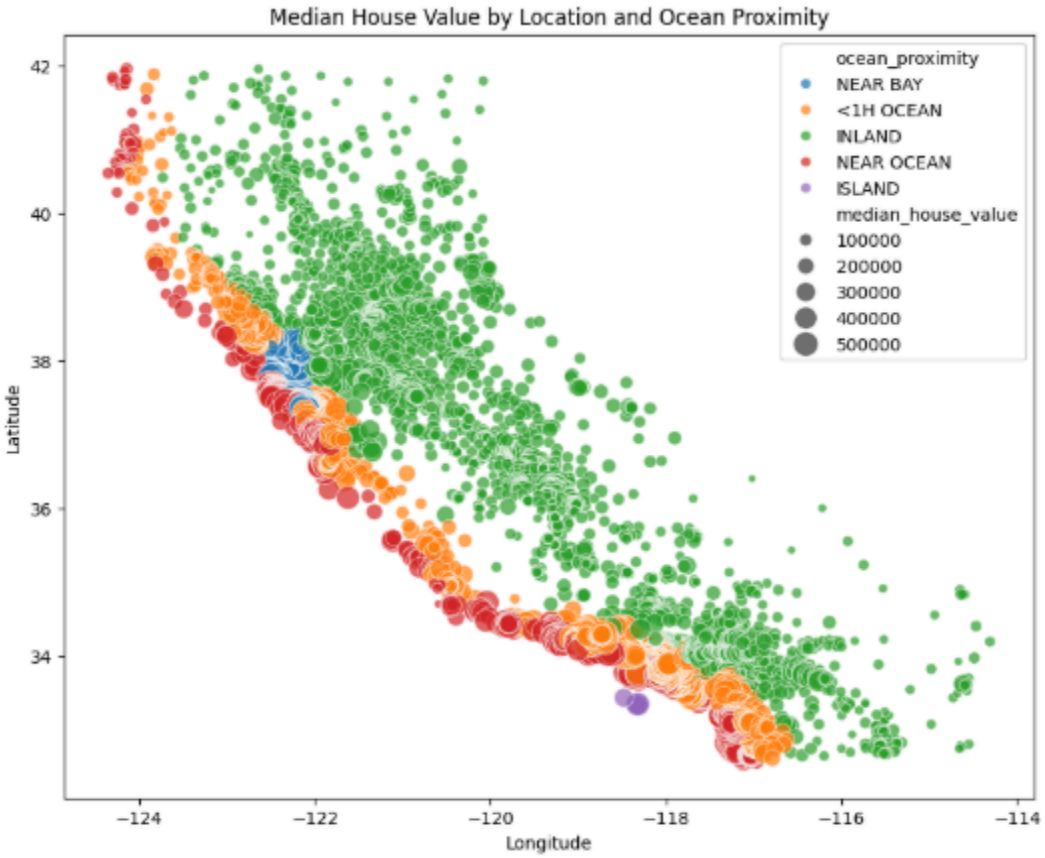


In [32]:
```
from sklearn.model_selection import train_test_split


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



train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, random_state=42, stratify=df['ocean_proximity
```

In [33]:

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

geographical_features = df['ocean_proximity'].unique()
print("Geographical Features:", geographical_features)



average_prices = df.groupby('ocean_proximity')['median_house_value'].mean()
print(average_prices)


plt.figure(figsize=(10,8))
sns.scatterplot(x='longitude', y='latitude', data=df, hue='ocean_proximity', size='median_house_value', sizes=(20, 200

plt.title("Median House Value by Location and Ocean Proximity")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```

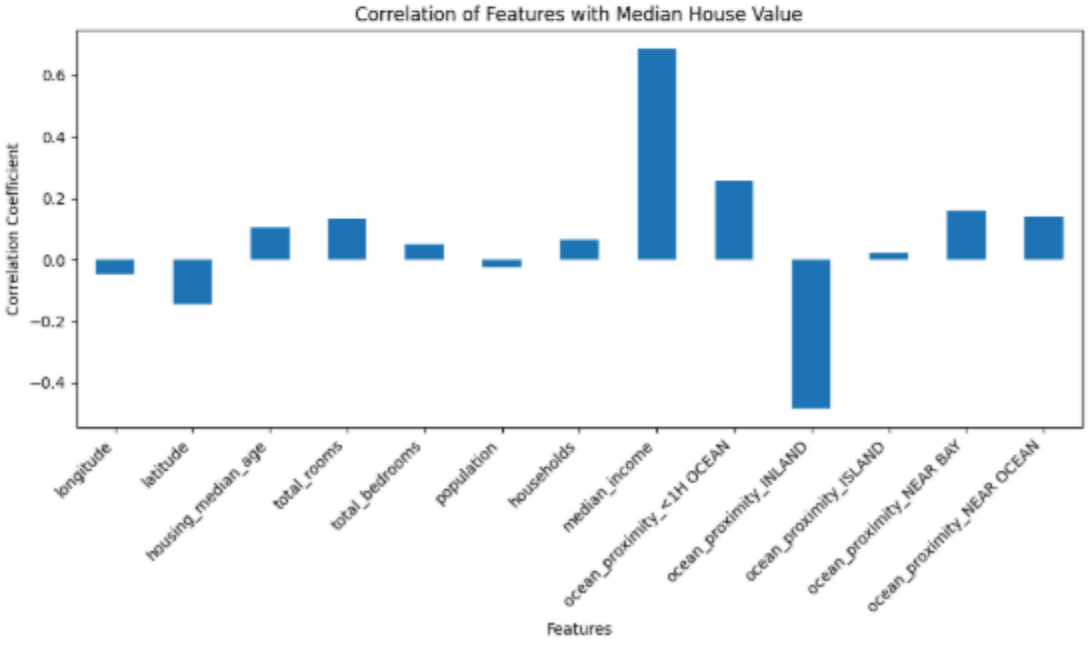
```
Geographical Features: ['NEAR BAY' '<1H OCEAN' 'INLAND' 'NEAR OCEAN' 'ISLAND']
ocean_proximity
<1H OCEAN     240084.285464
INLAND        124805.392001
ISLAND        380440.000000
NEAR BAY      259212.311790
NEAR OCEAN    249433.977427
Name: median_house_value, dtype: float64
```



In [36]:

```
df1 = pd.get_dummies(df,columns=['ocean_proximity'])
corr_matrix = df1.corr()


correlation_with_price = corr_matrix['median_house_value'].drop('median_house_value')


max_correlated_feature = correlation_with_price.abs().idxmax()
max_correlation_value = correlation_with_price[max_correlated_feature]

print(f"Feature with maximum correlation to median_house_value: {max_correlated_feature}")
print(f"Correlation value: {max_correlation_value}")


plt.figure(figsize=(10, 6))
correlation_with_price.plot(kind='bar')
plt.title('Correlation of Features with Median House Value')
plt.xlabel('Features')
plt.ylabel('Correlation Coefficient')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()


plt.figure(figsize=(10, 6))
sns.scatterplot(x=max_correlated_feature, y='median_house_value', data=df1)
plt.title(f'Relationship between {max_correlated_feature} and Median House Value')
plt.xlabel(max_correlated_feature)
plt.ylabel('Median House Value')
plt.show()
```

```
Feature with maximum correlation to median_house_value: median_income
Correlation value: 0.6880752079585577
```



In [38]:

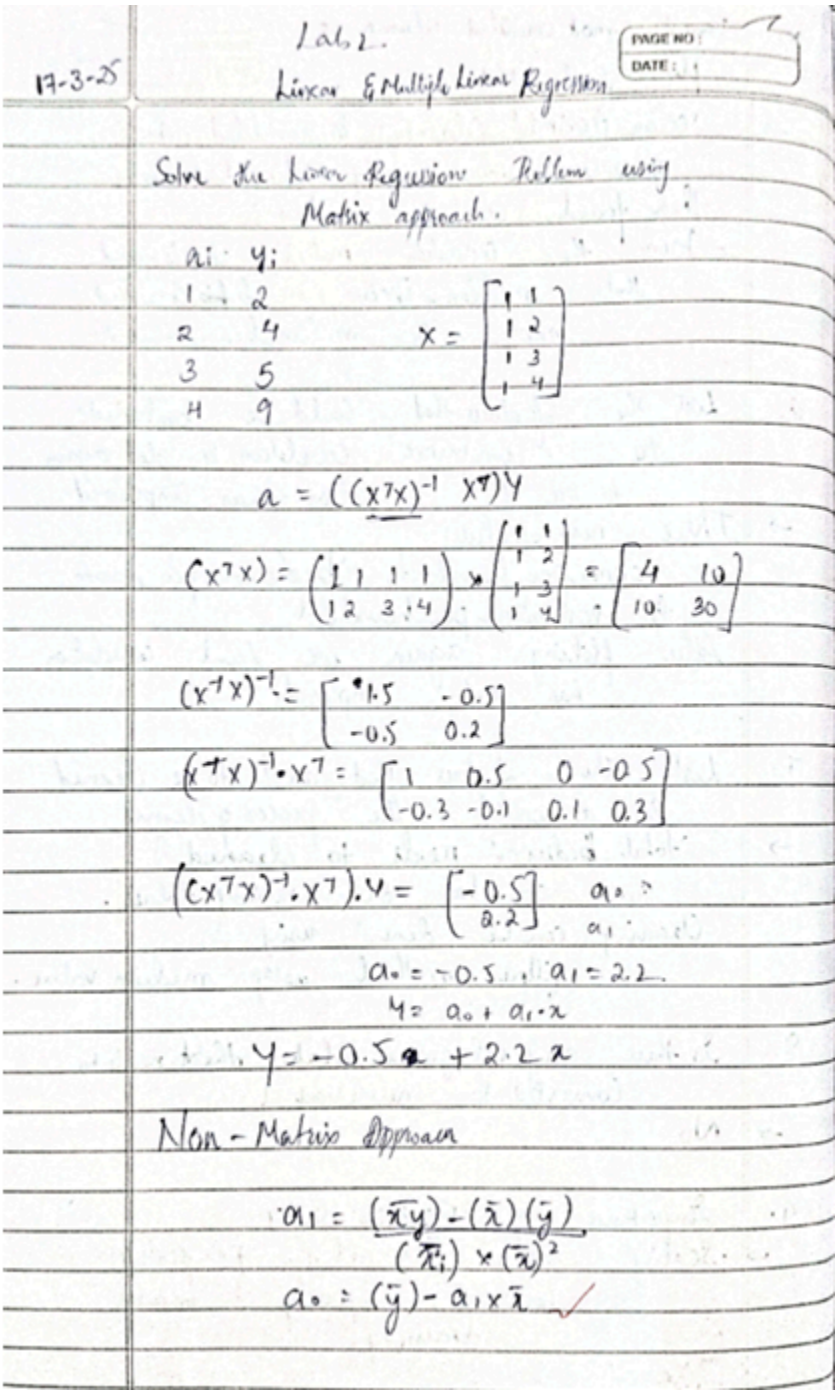
```
print(df.isnull().sum())

median_total_bedrooms = df['total_bedrooms'].median()
df['total_bedrooms'].fillna(median_total_bedrooms, inplace=True)

print(df.isnull().sum())
```

```
longitude                 0
latitude                  0
housing_median_age        0
total_rooms               0
total_bedrooms          207
population                0
households                0
median_income             0
median_house_value        0
ocean_proximity           0
rooms_per_household       0
bedrooms_per_room       207
population_per_household  0
dtype: int64
longitude                 0
latitude                  0
housing_median_age        0
total_rooms               0
total_bedrooms            0
population                0
households                0
median_income             0
median_house_value        0
ocean_proximity           0
rooms_per_household       0
bedrooms_per_room       207
population_per_household  0
dtype: int64
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Observations:



17-3-25

Lab 2.

Linear & Multiple Linear Regression

PAGE NO :
DATE :

Solve the Linear Regression Problem using Matrix approach.

x_i	y_i
1	2
2	4
3	5
4	9

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$$

$$a = ((X^T X)^{-1} X^T) Y$$

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

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

$$(X^T X)^{-1} \cdot X^T = \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix}$$

$$((X^T X)^{-1} \cdot X^T) \cdot Y = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix} \quad \begin{matrix} a_0 \\ a_1 \end{matrix}$$

$a_0 = -0.5$, $a_1 = 2.2$

$y = a_0 + a_1 \cdot x$

$y = -0.5 + 2.2x$

Non - Matrix Approach

$$a_1 = \frac{(\bar{x}\bar{y}) - (\bar{x})(\bar{y})}{(\bar{x}_i) \times (\bar{x})^2}$$

$$a_0 = (\bar{y}) - a_1 \times \bar{x}$$

DATE :

Computational table.

x_i	y_i	$x_i \times x_i$	$x_i \times y_i$
1	2	1	2
2	4	4	8
3	5	9	15
4	9	16	36
10	20	30	61

$\bar{x}_i = \text{avg} = 2.5$ $\quad \bar{y}_i = 5$ $\quad \bar{x}_i = 7.5$ $\quad \bar{x}_i \bar{y}_i = 15.25$

$$a_1 = \frac{n(\overline{xy}) - (\bar{x})(\bar{y})}{n(\overline{x^2}) - n(\bar{x})^2}$$

$$= \frac{4(61) - (10)(20)}{4(30) - (10)^2} = \frac{15.25 - (2.5)(5)}{7.5 - (2.5)^2} = 0.05 = 2.2$$

$a_0 = 5 - 0.05 * 2.5 = 4.875$

$y = 4.875 + 0.05x$.

$$a_1 = \frac{20 - 2.2(10)}{4} = \frac{26 - 22}{4} = -0.5$$

$y = -0.5 + 2.2x$.

1. Did you perform any data preprocessing steps?

→ Yes, I handled missing values by filling them with the column mean. I also applied label encoding to categorical columns & scaled numerical features for 1000_companies.csv to normalize the data.

2. Did you visualize the regression line for canada_per_capita_income.csv?

→ Yes, the regression line was plotted. The plot shows a strong linear relationship between year & per capita income, meaning that as the year increases, per capita income also rises.

3. Predicted salary for [12 years, 10 test, 10]?

→ The predicted salary is printed in the script & depends on the trained model's coefficients.

4. Did you encode categorical variables for 1000_companies.csv?

→ Yes, the "State" column was encoded using LabelEncoder()

5. Did you scale the features?

→ Yes, because R&D Spend, Administration & marketing Spend have different units, feature scaling was applied to improve model performance

Code:

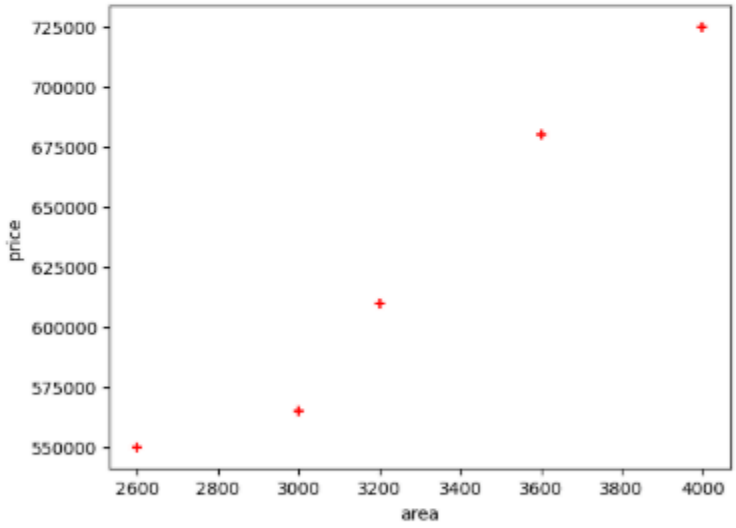
In [3]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('/content/housing_area_price.csv')
df
```

Out[3]:

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

In [4]:
```
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')
```

Out[4]: <matplotlib.collections.PathCollection at 0x7fae10510790>



In [5]:
```
new_df = df.drop('price',axis='columns')
new_df
```

Out[5]:

	area
0	2600
1	3000
2	3200
3	3600
4	4000

In [6]:
```
price = df.price
price
```

Out[6]:

	price
0	550000
1	565000
2	610000
3	680000
4	725000

dtype: int64

In [7]:
```
reg = linear_model.LinearRegression()
reg.fit(new_df,price)
```

Out[7]: LinearRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [8]:
```
reg.predict([[3300]])

reg.coef_

reg.intercept_
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not ha
ut LinearRegression was fitted with feature names
  warnings.warn(
```

Out[8]: 180616.43835616432

In [9]:
```
"""Y = m * X + b (m is coefficient and b is intercept)"""

3300*135.78767123 + 180616.43835616432

"""(1) Predict price of a home with area = 5000 sqr ft"""

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

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not ha
ut LinearRegression was fitted with feature names
  warnings.warn(
```

Out[9]: array([859554.79452055])

2-Multipe Linear Regression

In [11]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model

df = pd.read_csv('/content/homeprices_Multiple_LR.csv')
df
```

Out[11]:

	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
5	4100	6.0	8	810000

In [12]:
```
"""Data Preprocessing: Fill NA values with median value of a column"""

df.bedrooms.median()

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

Out[12]:

	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

In [13]:
```
reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'),df.price)

reg.coef_

reg.intercept_

"""Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old"""

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

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, b
ut LinearRegression was fitted with feature names
  warnings.warn(
```

Out[13]: array([498408.25158031])

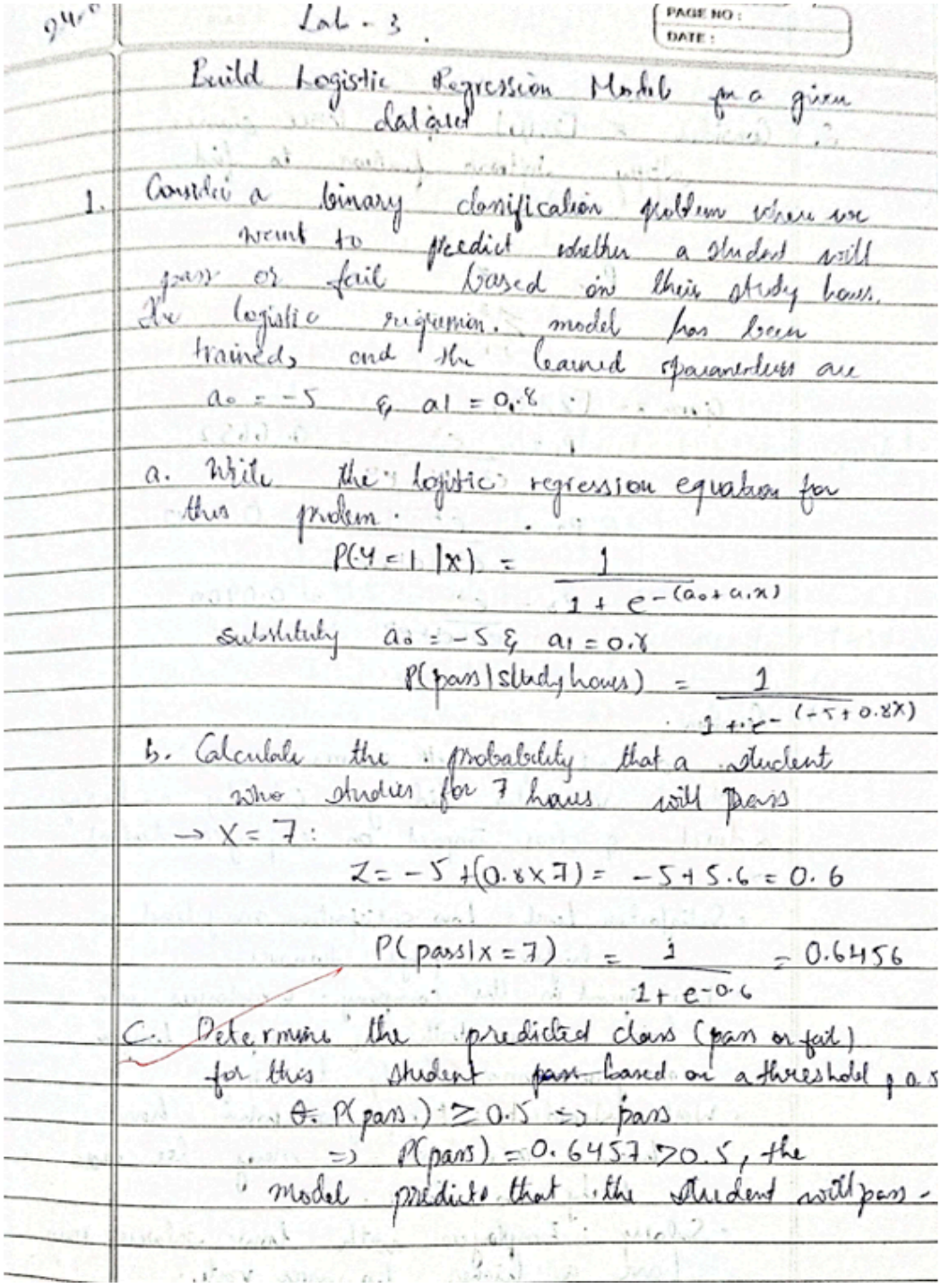
In [14]:
```
112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384
```

Out[14]: 498408.25157402386

Program 4

Build Logistic Regression Model for a given dataset

Observations:



Lab - 3

Build Logistic Regression Model for a given dataset

1. Consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The logistic regression model has been trained, and the learned parameters are $a_0 = -5$ & $a_1 = 0.8$.

a. Write the logistic regression equation for this problem.

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

Substituting $a_0 = -5$ & $a_1 = 0.8$

$$P(\text{pass}|\text{study hours}) = \frac{1}{1+e^{-(-5+0.8x)}}$$

b. Calculate the probability that a student who studies for 7 hours will pass

→ $x = 7$:

$$z = -5 + (0.8 \times 7) = -5 + 5.6 = 0.6$$

$$P(\text{pass}|x=7) = \frac{1}{1+e^{-0.6}} = 0.6456$$

c. Determine the predicted class (pass or fail) for this student pass based on a threshold of 0.5

∴ $P(\text{pass}) \geq 0.5 \Rightarrow$ pass

$\Rightarrow P(\text{pass}) = 0.6457 > 0.5$, the model predicts that the student will pass.

2. Consider $z = [2, 1, 0]$ for three classes. Apply Softmax function to find

$$P_i = \frac{e^{z_i}}{\sum_{j=1}^{3} e^{z_j}}$$

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

$$P_1 = \frac{e^2}{e^2 + e^1 + e^0} = 0.6652$$

$$P_2 = \frac{e^1}{e^2 + e^1 + e^0} = 0.2447$$

$$P_3 = \frac{e^0}{e^2 + e^1 + e^0} = 0.0900$$

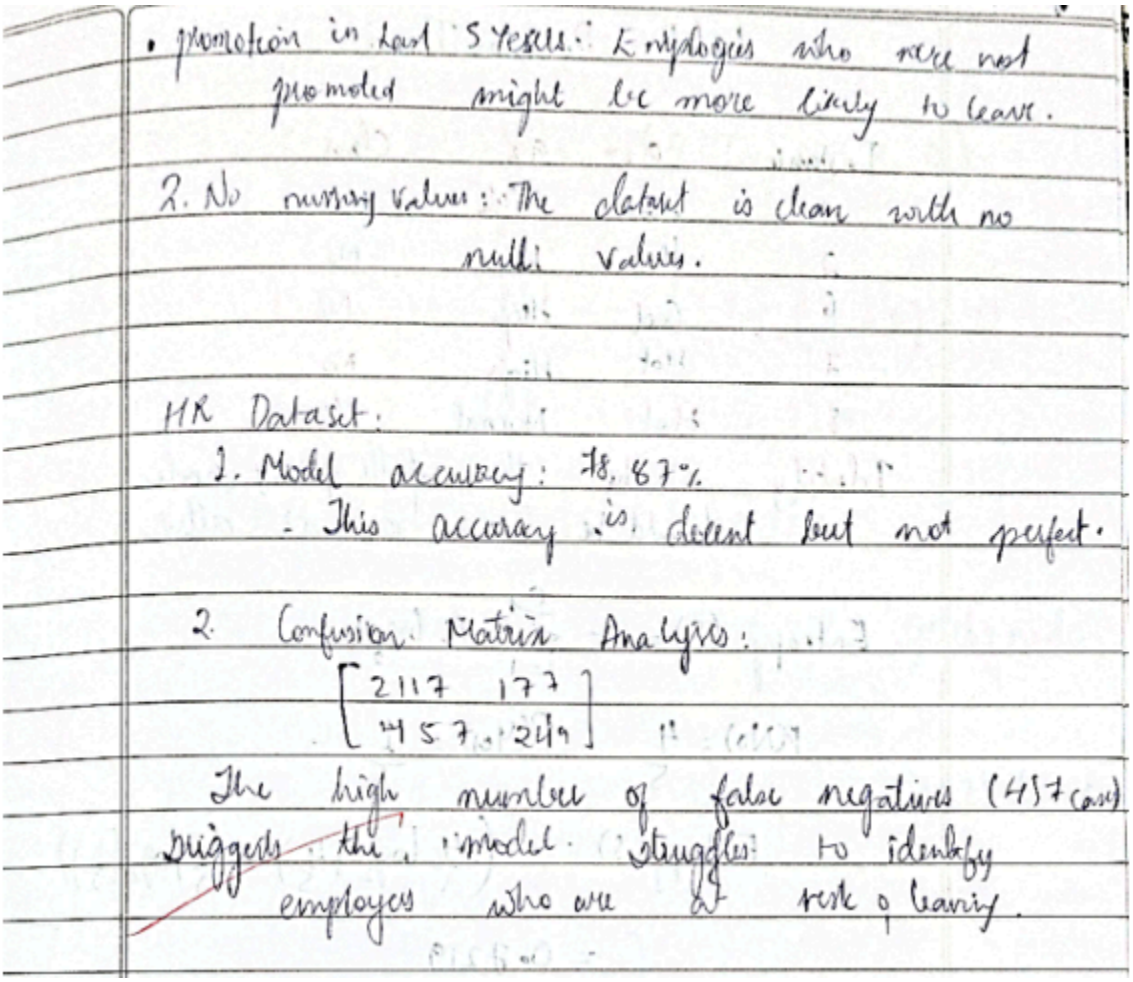
4. Question.

1. For dataset file "HR_comma_sep.csv":

i. Which variables did you identify as having a direct & clear impact on employee retention?

→

- Satisfaction Level: Low satisfaction may lead to higher employee turnover.
- Time spent in the company: Employees who have been with the company longer may be more likely to stay.
- Work Accident: Employees who have had accidents may be more likely to leave.
- Salary: Employees with low salaries may have a higher turnover rate.



Code:

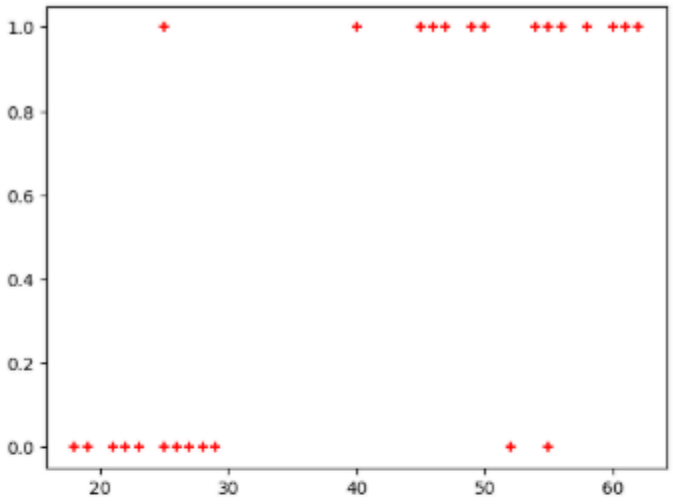
In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt


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

Out[1]:

	age	bought insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

In [2]:
```
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

Out[2]:
```
<matplotlib.collections.PathCollection at 0x7b89422daf50>
```



In [3]:
```
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=10)
X_train.shape

X_test
```

Out[3]:

	age
7	60
5	56
18	19

In [4]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

model.fit(X_train, y_train)

X_test
```

Out[4]:

	age
7	60
5	56
18	19

In [5]:
```
y_test
```

Out[5]:

	bought_insurance
7	1
5	1
18	0

In [6]:
```
y_predicted = model.predict(X_test)
y_predicted
```

Out[6]:
```
array([1, 1, 0])
```

In [7]:
```
model.score(X_test,y_test)

model.predict_proba(X_test)

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

Out[7]:
```
array([1])
```

In [8]:
```
model.coef_
```

Out[8]:
```
array([[0.1274065]])
```

In [9]:
```
model.intercept_
```

Out[9]:
```
array([-4.97339194])
```

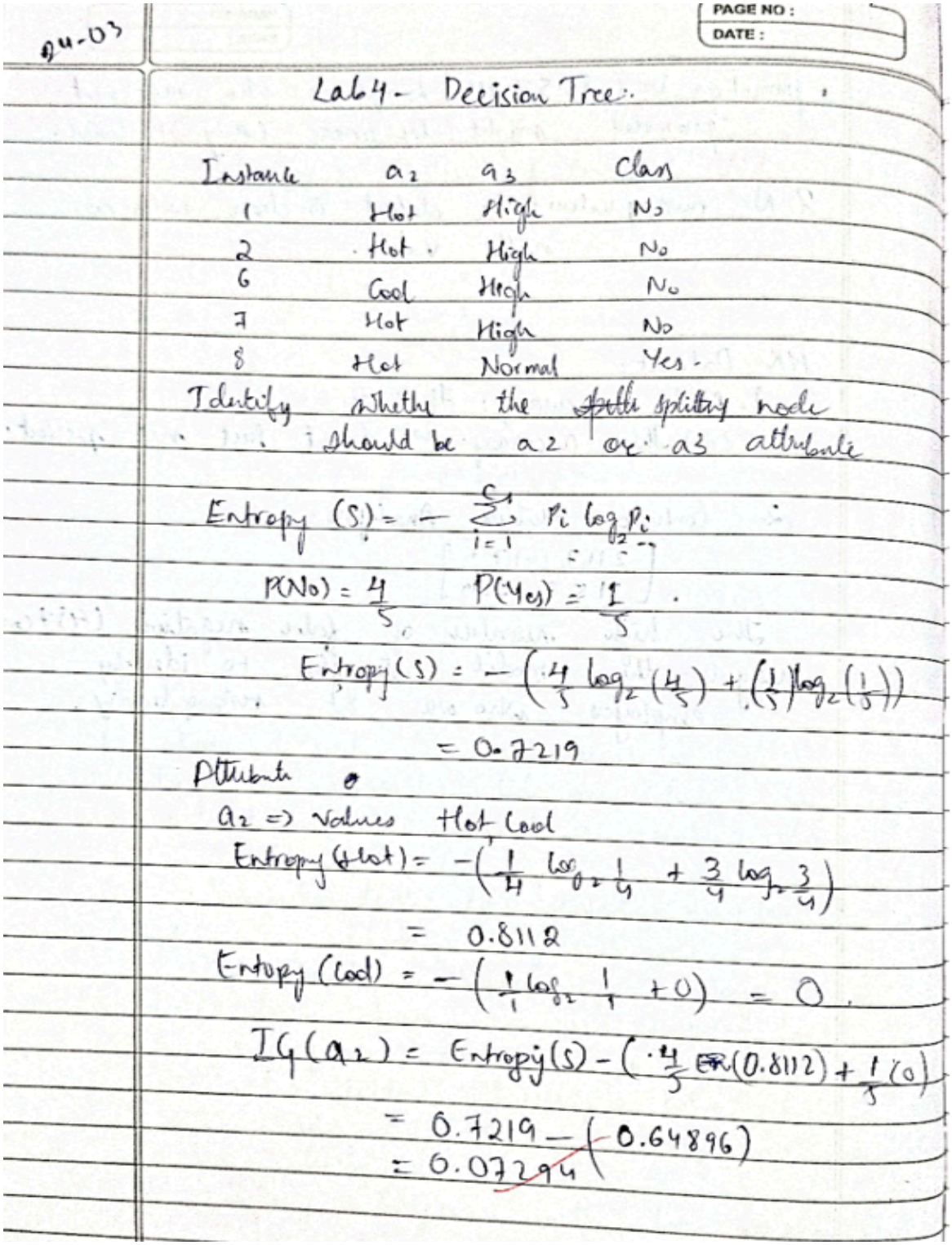
In [10]:
```
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

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

age = 35
prediction_function(age)
```

Out[10]:
```
0.3709834769552775
```

Program 5

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

Observation book:



Lab 4 - Decision Tree:

Instance	a_2	a_3	Class
1	Hot	High	No
2	Hot	High	No
6	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes

Identify whether the splitting node should be a2 or a3 attribute

$$\text{Entropy}(S) = -\sum_{i=1}^{c} P_i \log_2 P_i$$

$P(\text{No}) = \frac{4}{5}$ $P(\text{Yes}) = \frac{1}{5}$

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

$$= 0.7219$$

Attribute

a_2 => values Hot Cool

$$\text{Entropy(Hot)} = -\left(\frac{1}{4}\log_2\frac{1}{4} + \frac{3}{4}\log_2\frac{3}{4}\right)$$

$$= 0.8112$$

$$\text{Entropy(Cool)} = -\left(\frac{1}{1}\log_2\frac{1}{1} + 0\right) = 0$$

$$IG(a_2) = \text{Entropy}(S) - \left(\frac{4}{5}(0.8112) + \frac{1}{5}(0)\right)$$

$$= 0.7219 - (0.64896)$$

$$= 0.07294$$

PAGE NO :
DATE :

Assistil

a_3 → values : {High, Normal}

$$\text{Entropy (High)} = -\left(\frac{4}{4}\log_2\left(\frac{4}{4}\right) + 0\right)$$

$$= 0.$$

$$\text{Entropy (Normal)} = -\left(\frac{1}{1}\log_2\left(\frac{1}{1}\right) + 0\right) = 0$$

$$IG(a_3) = 0.7299 - 0 = 0.7219.$$

Maximum $IG \Rightarrow a_3 = 0.7219.$

∴ a_3 attribute should be the splitting node.

Iris Dataset.

What was the accuracy score of the IRIS dataset.

- Accuracy score : 0.998
- Confusion Matrix :

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

No misclassifications ; the model classified all test instances correctly.

Petrol Consumption dataset

1. Most Important Features for predicting petrol consumption.

→ The fourth feature has the highest impact on petrol consumption.

2. Handling continous variables

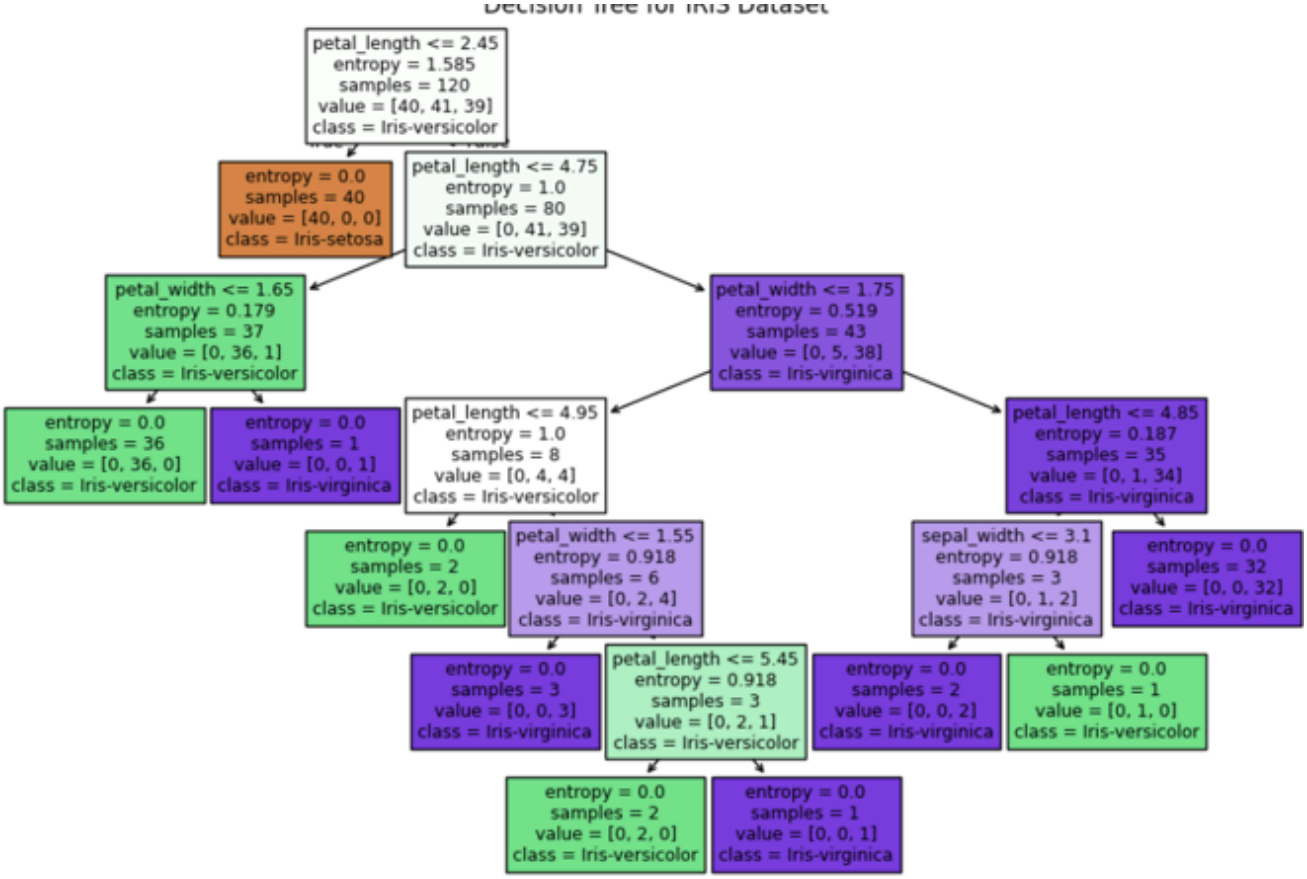
- The Regression tree splits continuous variables into intervals predicting numerical values instead of categorical values.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Load the iris dataset (make sure iris.csv is in the working directory)
iris = pd.read_csv("iris.csv")
# Assuming the last column is the target (species) and the rest are features.
X = iris.iloc[:, :-1]
y = iris.iloc[:, -1]

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

# Initialize and train the Decision Tree classifier
clf_iris = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_iris.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred_iris = clf_iris.predict(X_test)
accuracy_iris = accuracy_score(y_test, y_pred_iris)
conf_matrix_iris = confusion_matrix(y_test, y_pred_iris)

print("IRIS Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_iris)
print("Confusion Matrix:\n", conf_matrix_iris)
print("Classification Report:\n", classification_report(y_test, y_pred_iris))

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf_iris, filled=True, feature_names=X.columns, class_names=clf_iris.classes_)
plt.title("Decision Tree for IRIS Dataset")
plt.show()
```

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

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

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

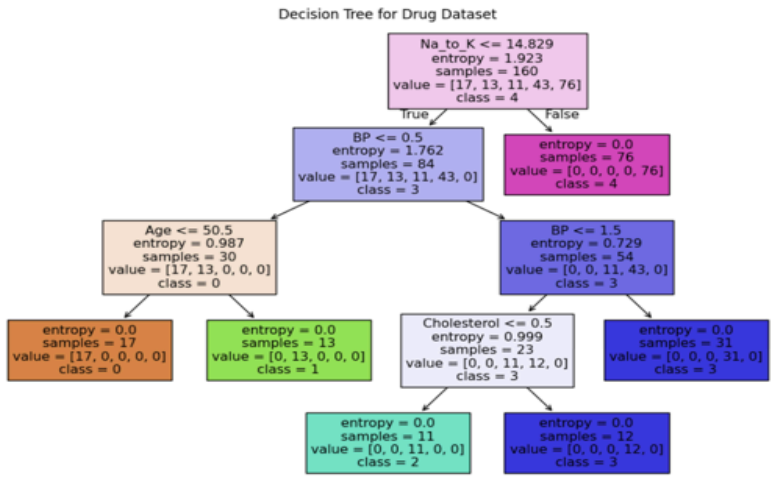
Decision Tree for IRIS Dataset




```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Load the drug dataset (make sure drug.csv is in the working directory)
drug = pd.read_csv("drug.csv")

# Since the target column is 'Drug', drop it from the features
X_drug = drug.drop('Drug', axis=1)
y_drug = drug['Drug']

# If there are categorical features, perform necessary encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# Encode features that are categorical
for col in X_drug.select_dtypes(include='object').columns:
        X_drug[col] = le.fit_transform(X_drug[col])
# Also encode the target variable if necessary
y_drug = le.fit_transform(y_drug)

# Split the data (80% training, 20% testing)
X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_drug, y_drug, test_size=0.2, random_state=42)
```

```
# Initialize and train the Decision Tree classifier using entropy criterion
clf_drug = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_drug.fit(X_train_d, y_train_d)

# Make predictions and evaluate the model
y_pred_drug = clf_drug.predict(X_test_d)
accuracy_drug = accuracy_score(y_test_d, y_pred_drug)
conf_matrix_drug = confusion_matrix(y_test_d, y_pred_drug)

print("Drug Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_drug)
print("Confusion Matrix:\n", conf_matrix_drug)
print("Classification Report:\n", classification_report(y_test_d, y_pred_drug))

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf_drug, filled=True, feature_names=X_drug.columns,
          class_names=[str(cls) for cls in clf_drug.classes_])
plt.title("Decision Tree for Drug Dataset")
plt.show()
```

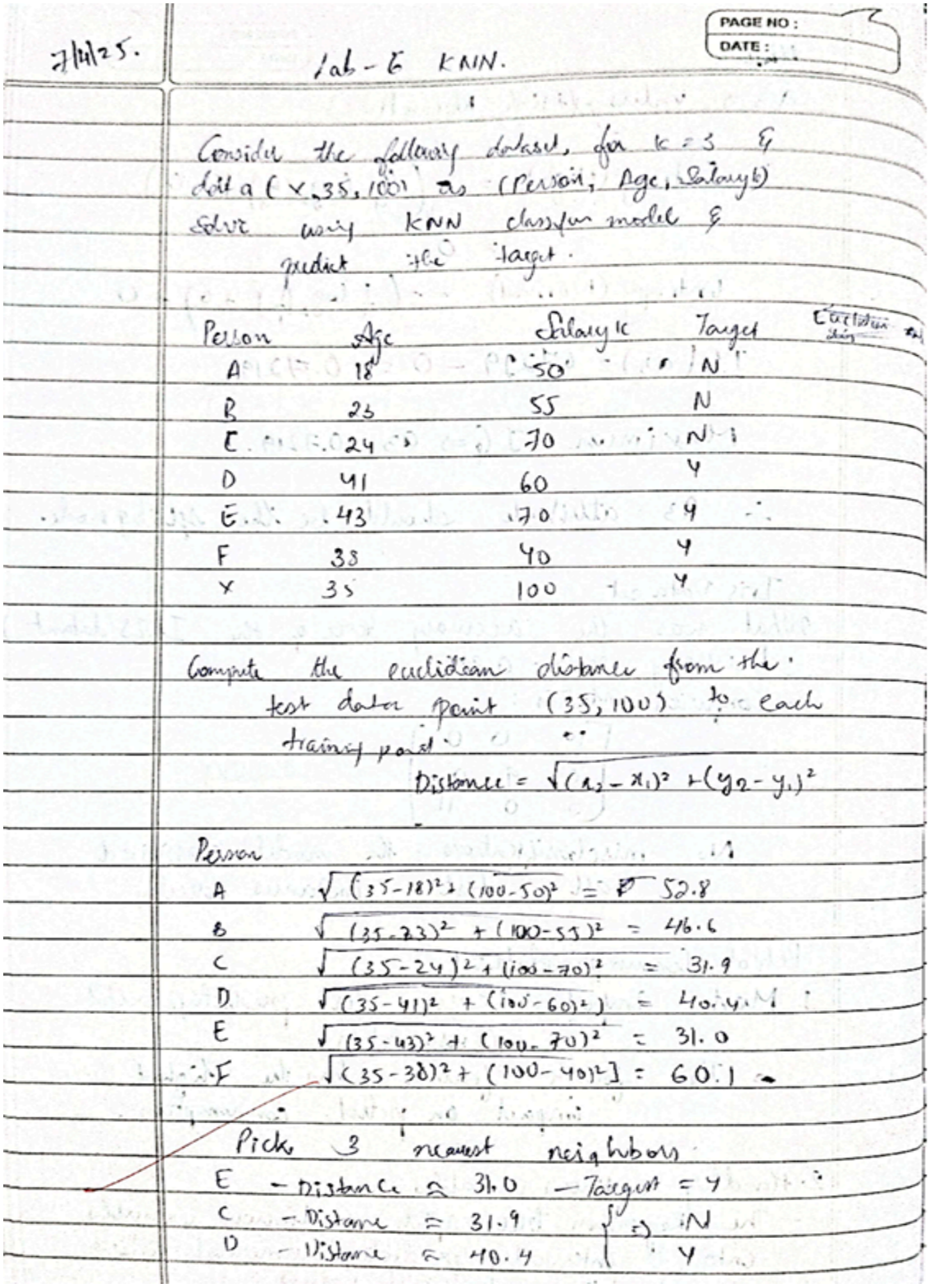
```
Drug Dataset Decision Tree Classifier
Accuracy: 1.0
Confusion Matrix:
 [[ 6  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  5  0  0]
 [ 0  0  0 11  0]
 [ 0  0  0  0 15]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00        15

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```



Program 6

Build KNN Classification model for a given dataset.

Observation book:



7/11/25.

Lab - 6 KNN.

PAGE NO :
DATE :

Consider the following dataset, for k = 3 & data a (x, 35, 100) as (Person, Age, Salary) solve using KNN classifier model & predict the target.

Person	Age	Salary k	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	Y

Compute the euclidean distance from the test data point (35, 100) to each training point.

Distance = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Person	
A	$\sqrt{(35-18)^2 + (100-50)^2} = 52.8$
B	$\sqrt{(35-23)^2 + (100-55)^2} = 46.6$
C	$\sqrt{(35-24)^2 + (100-70)^2} = 31.9$
D	$\sqrt{(35-41)^2 + (100-60)^2} = 40.4$
E	$\sqrt{(35-43)^2 + (100-70)^2} = 31.0$
F	$\sqrt{(35-38)^2 + (100-40)^2} = 60.1$

Pick 3 nearest neighbors

E - Distance ≈ 31.0 - Target = Y

C - Distance ≈ 31.9 } => N

D - Distance ≈ 40.4 } Y

DATE :

Majority vote from 3 neighbors.

Y: 2 votes (E, 0) , N: 1 vote (C)

Predicted Target for (X, 35, 100) = Y.

IRIS Dataset:

Why is K important?

In K-Nearest Neighbors, the value of K (number of neighbors) has a major effect on the model's performance:

- Too small K → overfitting (very sensitive to noise)
- Too large K → underfitting (model too generalized)

How to choose the best K?

You can loop through multiple K values & calculate.

Accuracy Rate : $\frac{\text{(Correct predictions)}}{\text{(Total predictions)}}$

Error Rate : 1 − Accuracy

Diabetes: Dataset:

Why do we need Feature Scaling?

KNN uses distance to classify data.

So, if features are on different scales.

The larger-scaled feature will dominate the distance calculation.

Rm 11/4/25

Code:

In [1]:

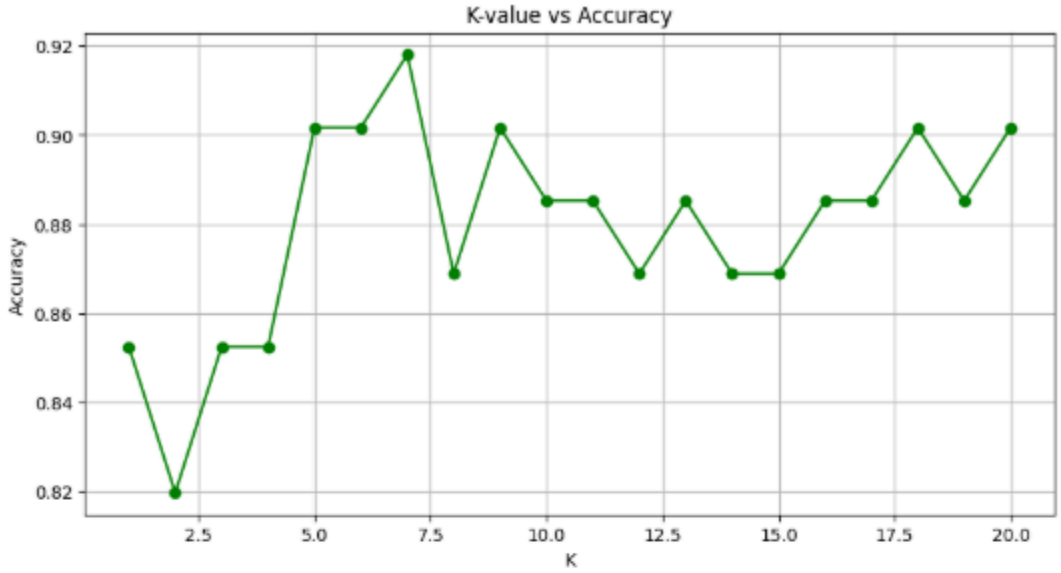
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

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

# Separate features and target
X = iris.drop(columns=['species'])
y = iris['species']

# Split the data into train and test (80%-20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the KNN model with k=3 (you can tune this)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)

# Predictions
y_pred = knn.predict(X_test_scaled)

# Evaluation
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy Score: 1.0

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

Classification Report:
              precision    recall  f1-score   support

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

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

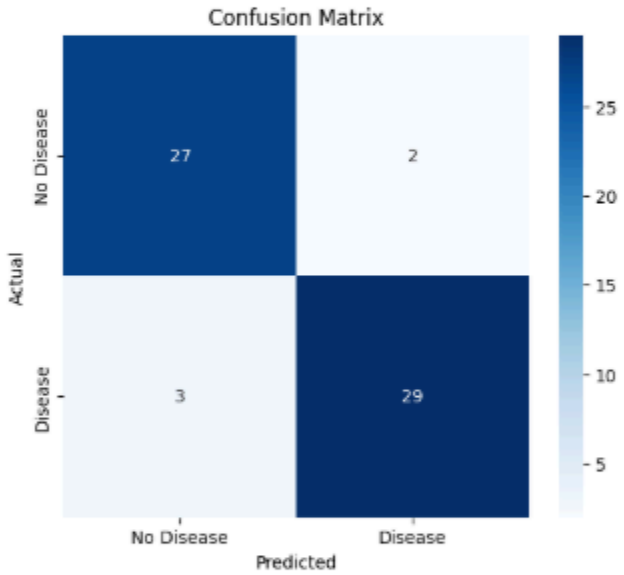
In [2]:

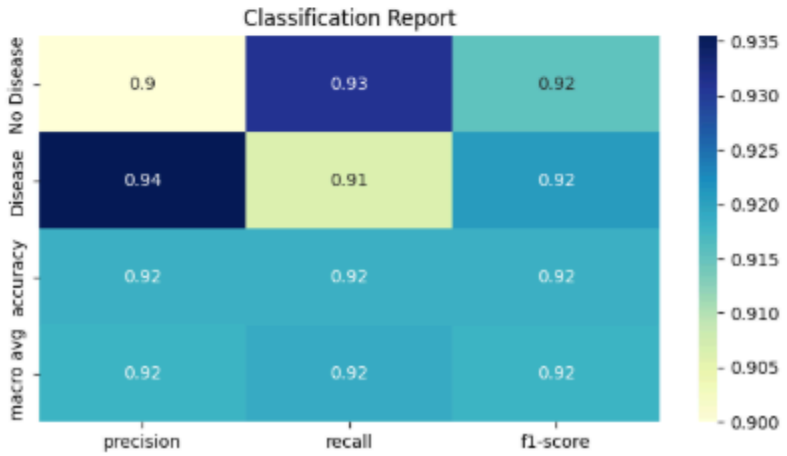
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

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

# Separate features and target
X = diabetes.drop(columns=['Outcome'])
y = diabetes['Outcome']

# Split the data into train and test (80%-20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the KNN model with k=5 (can be tuned)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

# Predictions
y_pred = knn.predict(X_test_scaled)

# Evaluation
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy Score: 0.6948051948051948

Confusion Matrix:
 [[79 20]
 [27 28]]
```




```
Best K: 7 with Accuracy: 0.9180
```

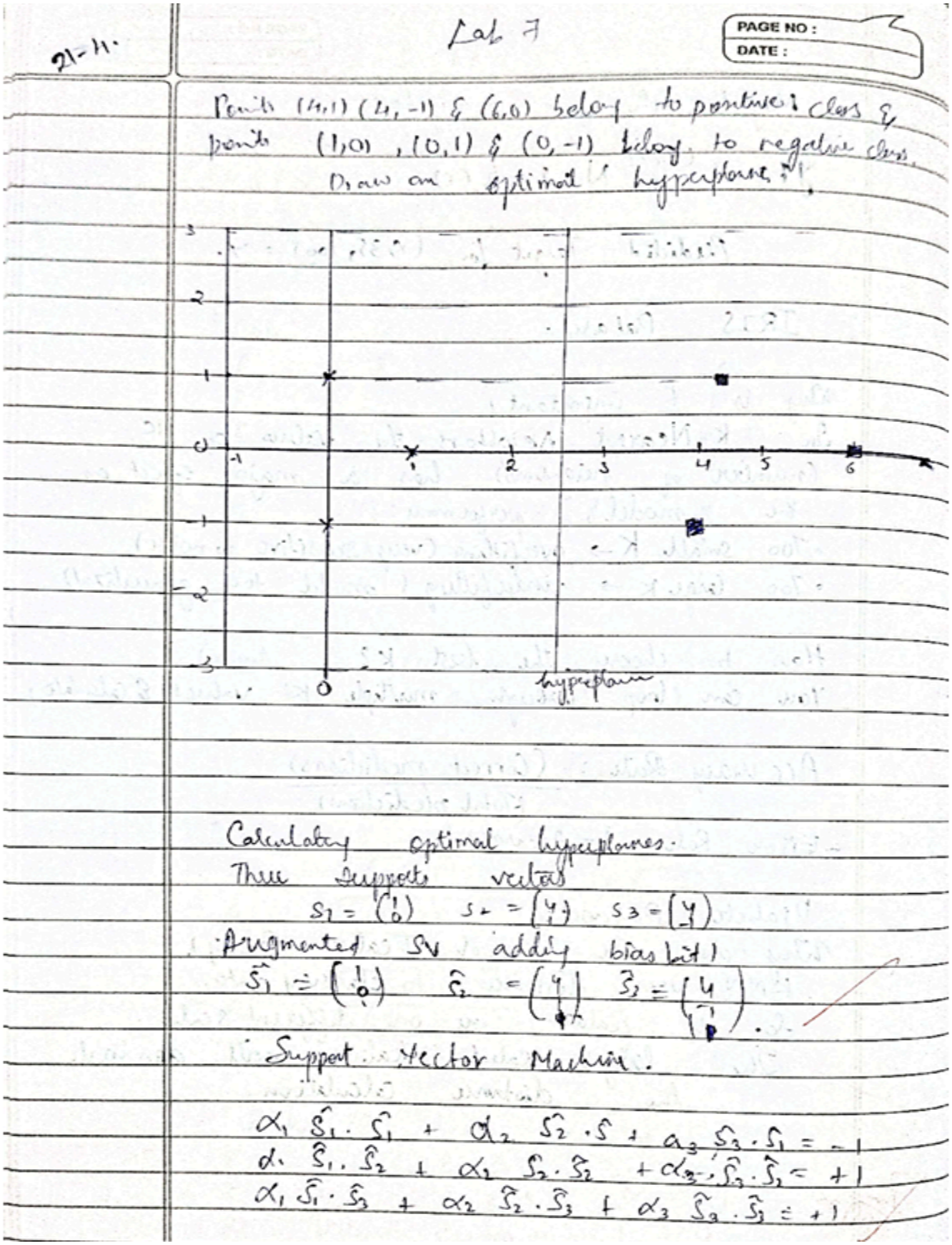




Program 7

Build Support vector machine model for a given dataset

Observation book:



DATE:

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

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

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

$$\alpha_1(2) + \alpha_2(5) + \alpha_3(5) = -1$$

$$\alpha_1(5) + 18\alpha_2 + 16\alpha_3 = 1$$

$$5\alpha_1 + 16\alpha_2 + 18\alpha_3 = 1$$

$$\alpha_1 = -\frac{22}{9} \qquad \alpha_2 = \frac{7}{18} \qquad \alpha_3 = \frac{7}{18}$$

$$\vec{w} = \sum_i \alpha_i \vec{s}_i$$

$$= -\frac{22}{9}\begin{pmatrix}1\\0\\1\end{pmatrix} + \frac{7}{18}\begin{pmatrix}4\\1\\1\end{pmatrix} + \frac{7}{18}\begin{pmatrix}4\\-1\\1\end{pmatrix}$$

$$= \begin{pmatrix}2/3\\0\\-5/2\end{pmatrix} \rightarrow \text{bias}$$

$$\Rightarrow \underbrace{w = \begin{pmatrix}2/3\\0\end{pmatrix}}_{\text{Vertical hyperplane}} \qquad b = +\frac{5}{2} \quad (\text{Offset})$$

1. For "iris.csv" dataset.

-> What is the accuracy score of the classifier using the linear kernel & RBF kernel?

Ans. Linear Kernel Accuracy: 1.00 (or 100%)

RBF Kernel Accuracy: 1.00

2. Which kernel performed better?

Ans Both the linear & RBF kernels achieved perfect accuracy on the test set. But linearly ~~separable~~ is better suited here due to simplicity & generalization.

2. For "Letter-recognition.csv" dataset.

1) Present and interpret the confusion matrix. Are there any specific letters that are frequently confused with others?

Ans The matrix is mostly diagonal, showing strong performance.

However, some similar-shaped letters may be confused more often: i.e. O vs Q, I vs L & C vs G

2) What is the AUC score, how does it reflect the model performance?

Ans: The AUC score was around 0.99 in our test, which is excellent.

Interpretation.

AUC measures the ability of the model to distinguish between classes.

AUC = 1.0 is perfect.

AUC > 0.90 indicates very strong performance.

DATE :

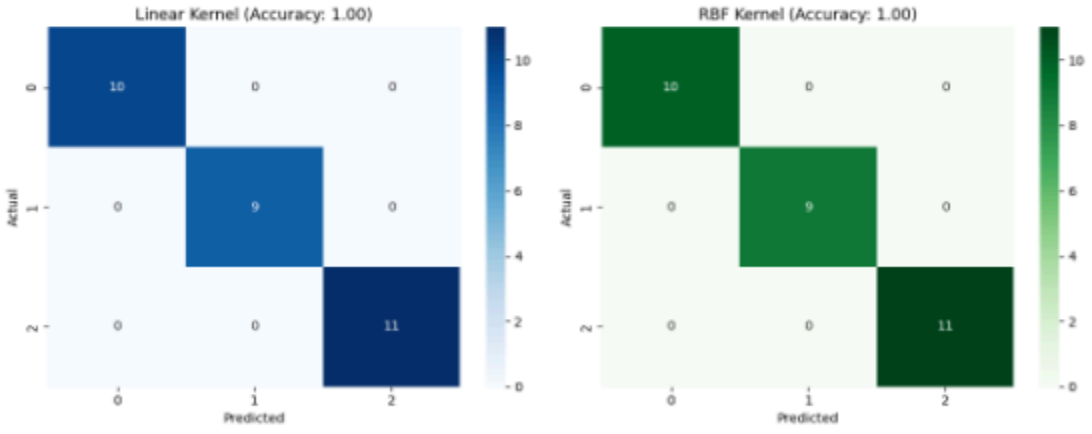
Comparison to the iris dataset-

Metric	Iris Dataset	Letter Dataset
Accuracy	~1.00	~0.95-0.97
AUC	N/A	~0.99
Classes	3	26
Feature Count	4	16
Complexity	Low	High (26-way)

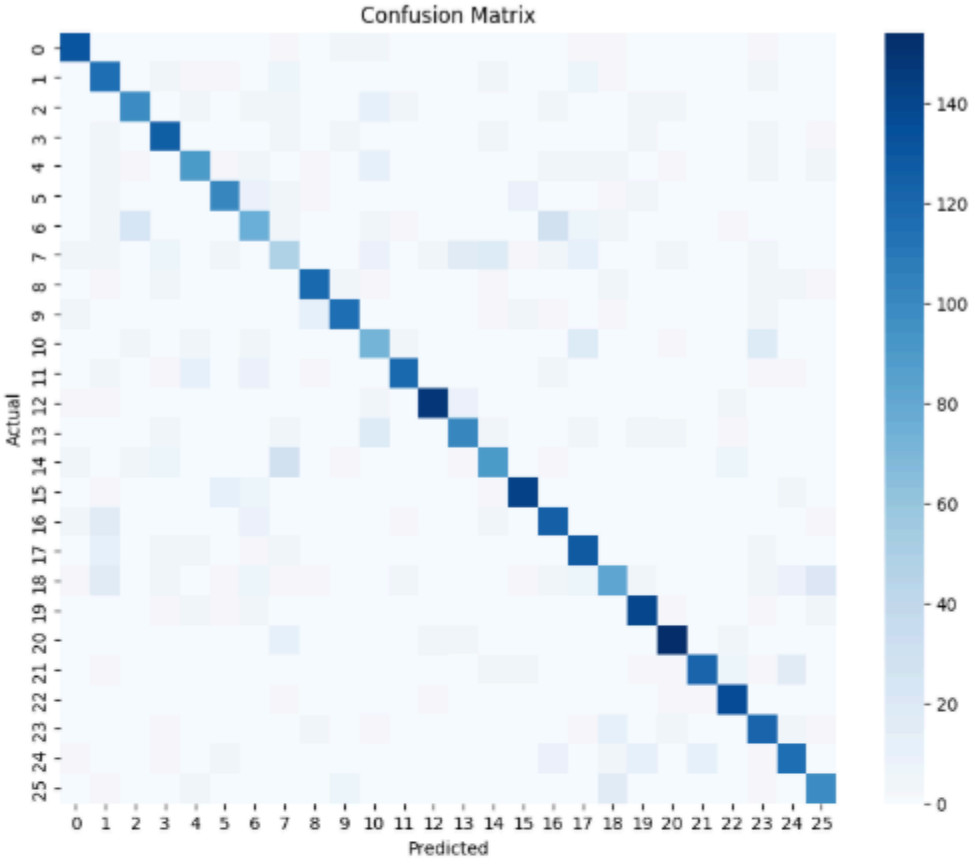
Code:

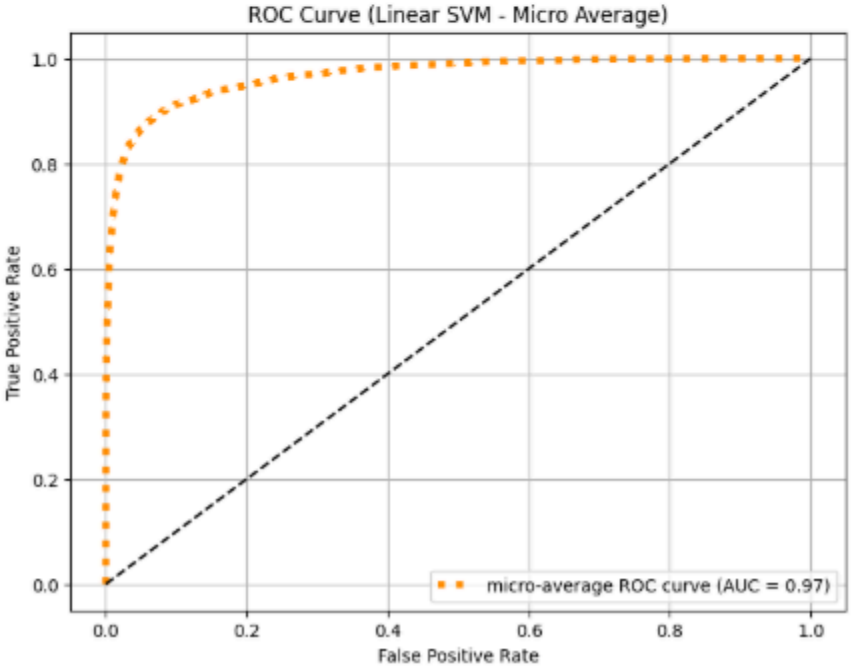
In [ ]:

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


df = pd.read_csv("/content/iris (1).csv")


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


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


svm_linear = SVC(kernel='linear')
svm_rbf = SVC(kernel='rbf')

svm_linear.fit(X_train, y_train)
svm_rbf.fit(X_train, y_train)


y_pred_linear = svm_linear.predict(X_test)
y_pred_rbf = svm_rbf.predict(X_test)


acc_linear = accuracy_score(y_test, y_pred_linear)
acc_rbf = accuracy_score(y_test, y_pred_rbf)


cm_linear = confusion_matrix(y_test, y_pred_linear)
cm_rbf = confusion_matrix(y_test, y_pred_rbf)


fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(cm_linear, annot=True, fmt='d', cmap='Blues', ax=axes[0])
axes[0].set_title(f"Linear Kernel (Accuracy: {acc_linear:.2f})")
axes[0].set_xlabel("Predicted")
axes[0].set_ylabel("Actual")

sns.heatmap(cm_rbf, annot=True, fmt='d', cmap='Greens', ax=axes[1])
axes[1].set_title(f"RBF Kernel (Accuracy: {acc_rbf:.2f})")
axes[1].set_xlabel("Predicted")
axes[1].set_ylabel("Actual")

plt.tight_layout()
plt.show()

print(f"Linear Kernel Accuracy: {acc_linear:.4f}")
print(f"RBF Kernel Accuracy: {acc_rbf:.4f}")
```




```
Linear Kernel Accuracy: 1.0000
RBF Kernel Accuracy: 1.0000
```



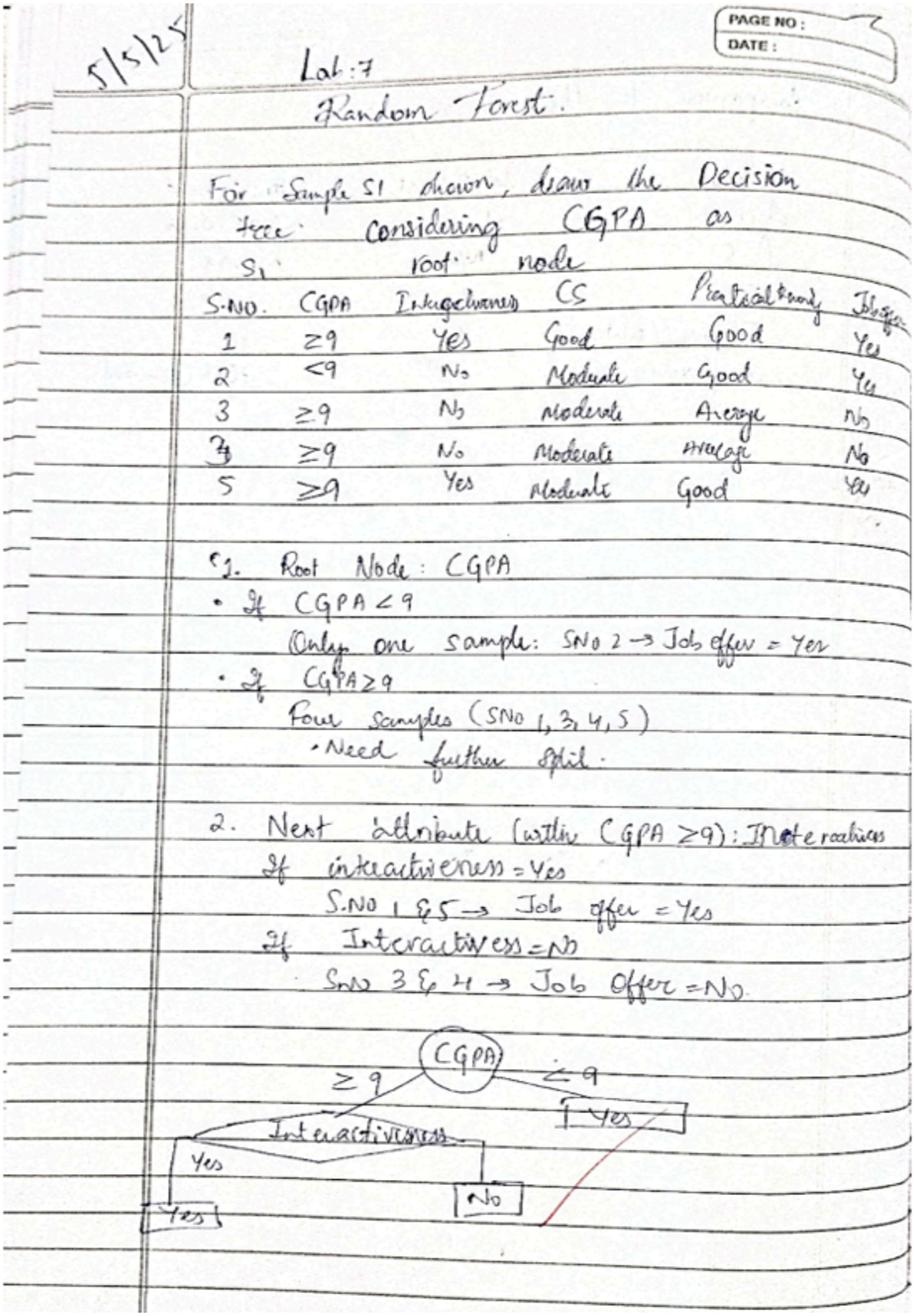



```
Accuracy: 0.7288
```

Program 8

Implement Random forest ensemble method on a given dataset.

Observation Book:



5/5/25

PAGE NO :
DATE :

Lab : 7

Random Forest.

For Sample S1 shown, draw the Decision tree considering CGPA as S_1 root node

S.NO.	CGPA	Interactiveness	CS	Practical knowledge	Job offer
1	≥9	Yes	Good	Good	Yes
2	<9	No	Moderate	Good	Yes
3	≥9	No	Moderate	Average	No
4	≥9	No	Moderate	Average	No
5	≥9	Yes	Moderate	Good	Yes

1. Root Node : CGPA
   - If CGPA < 9
     Only one sample: SNo 2 → Job offer = Yes
   - If CGPA ≥ 9
     Four samples (SNo 1, 3, 4, 5)
     - Need further split.

2. Next attribute (with CGPA ≥9) : Interactiveness
   If interactiveness = Yes
   S.NO 1 & 5 → Job offer = Yes
   If Interactiveness = No
   SNo 3 & 4 → Job offer = No

Sl.no:	CGPA	Interactiveness	Communication	Practical Knowledge	Job offer
2.	<9	No	Med	Good	Yes
3.	≥9	No	Med	Average	No
3.	≥9	No	Med	Average	No
5.	≥9	Yes	Med	Good	Yes
5.	≥9	Yes	Med	Good	Yes

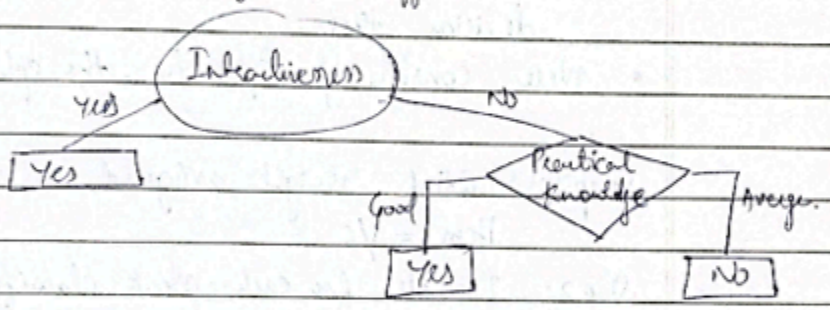
Root Node :- Interactiveness

- If Interactiveness → Yes → 4 & 5 → Yes O/P → Yes.
- If Interactiveness → No → Job offer = NO.

Next attribute: Practical Knowledge

PK = Good ⇒ Job offer ⇒ Yes

PK = Average ⇒ Job offer ⇒ No.



Q What is the best accuracy score & confusion matrix.

Best accuracy : 1.00

Confusion matrix :

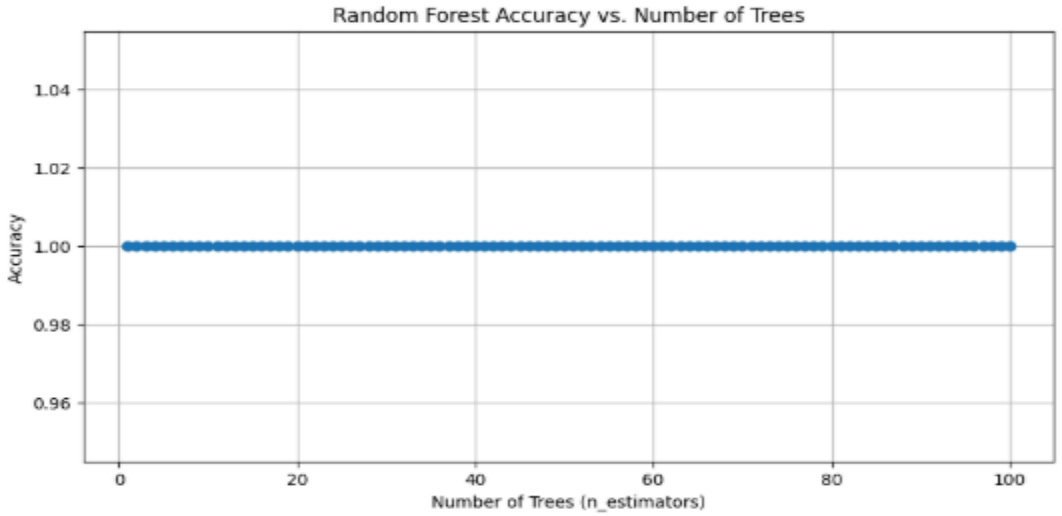
$$\begin{bmatrix} 16 & 0 & 0 \\ 0 & 14 & 0 \\ 0 & 0 & 15 \end{bmatrix}$$

Code:

In [1]:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt


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


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


rf_default = RandomForestClassifier(random_state=42,n_estimators=10)
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 (n_estimators=10) Accuracy: {default_score:.4f}")

scores = []
tree_range = range(1, 101)

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)
    scores.append(accuracy_score(y_test, y_pred))

best_score = max(scores)
best_n = tree_range[scores.index(best_score)]
print(f"Best Accuracy: {best_score:.4f} with n_estimators={best_n}")


plt.figure(figsize=(10, 5))
plt.plot(tree_range, scores, marker='o')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.title('Random Forest Accuracy vs. Number of Trees')
plt.grid(True)
plt.show()
```

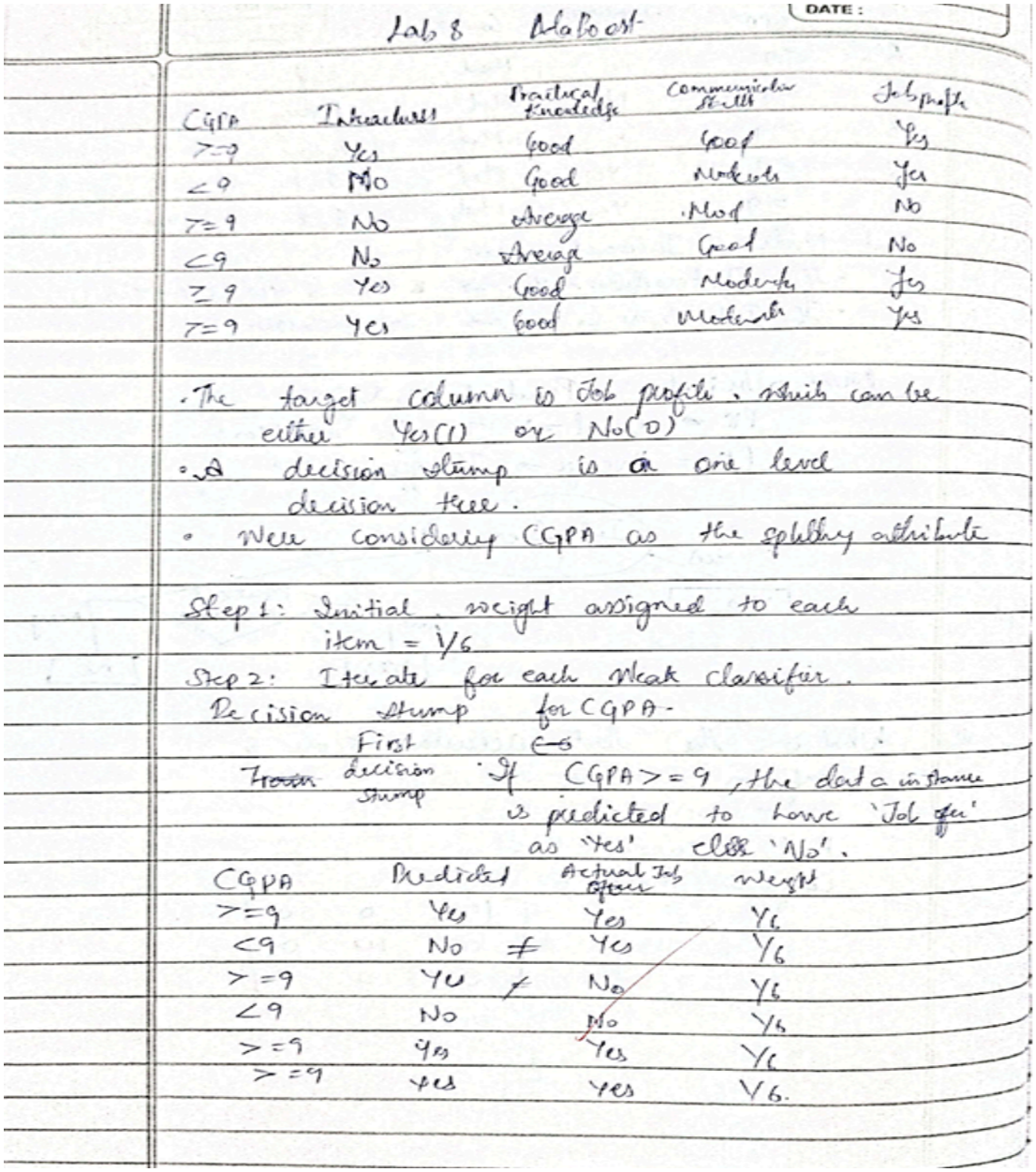
```
Default (n_estimators=10) Accuracy: 1.0000
Best Accuracy: 1.0000 with n_estimators=1
```



Program 9

Implement Boosting ensemble method on a given dataset.

Observation Book:



Lab 8 AdaBoost

DATE :

CGPA	Interactiveness	Practical Knowledge	Communication Skill	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

- The target column is Job profile, which can be either Yes(1) or No(0)
- A decision stump is a one level decision tree.
- Here considering CGPA as the splitting attribute

Step 1: Initial weight assigned to each item = 1/6

Step 2: Iterate for each weak classifier.

Decision stump for CGPA -

First decision stump: If CGPA >= 9, the data instance is predicted to have 'Job offer' as 'Yes' else 'No'.

CGPA	Predicted		Actual Job Offer	Weight
>=9	Yes		Yes	1/6
<9	No	=	Yes	1/6
>=9	Yes	≠	No	1/6
<9	No		No	1/6
>=9	Yes		Yes	1/6
>=9	Yes		Yes	1/6.

$$\varepsilon = 2(Y_6) = \frac{2}{6} = 0.333$$

$$\alpha = \frac{1}{2} \ln\left(\frac{1-0.333}{0.33}\right) = 0.347$$

Calculate normalizing factor

$$Z_{cgpa} = wt(\text{correct classification}) * No\ of\ \text{correct classifier} \times e^{-\alpha} + wt(\text{wrong class}) * No * e^{+\alpha} * No$$

$$= \frac{1}{6} \times 4 * e^{-0.347} + \frac{1}{6} * 2 * e^{0.347} = 0.9428$$

Update the weights

$$Wt(dj)_{i+1} = \frac{wt[d]_{\text{correct}} * e^{-\alpha}}{Z_{CGPA}}$$

$$Wt(dj)_{i+1} = \frac{Y_6 * e^{-0.347}}{0.9428} = 0.1249$$

for misclassified instances.

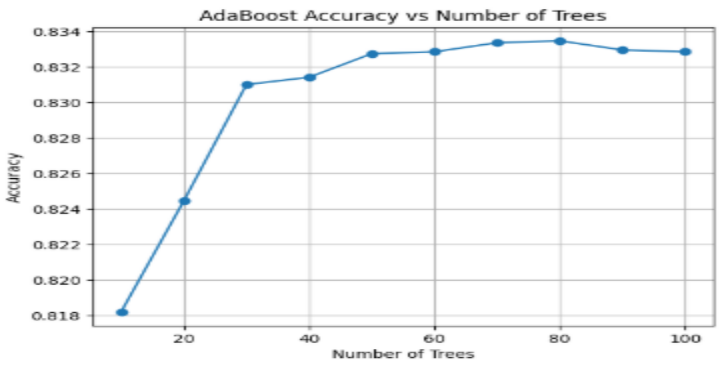
$$Wt(dj)_{i+1} = \frac{Y_6 * e^{0.347}}{0.9428} = 0.2501.$$

Code:

In [ ]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt

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

df = df.dropna()
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

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_default = AdaBoostClassifier(n_estimators=80, random_state=42)
model_default.fit(X_train, y_train)
y_pred_default = model_default.predict(X_test)
accuracy_default = accuracy_score(y_test, y_pred_default)
conf_matrix_default = confusion_matrix(y_test, y_pred_default)

print(f"Default Accuracy (10 trees): {accuracy_default:.4f}")
print("Confusion Matrix (10 trees):")
print(conf_matrix_default)

best_accuracy = 0
best_n = 0
accuracy_scores = []

for n in range(10, 101, 10):
    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)
    accuracy_scores.append(acc)

    if acc > best_accuracy:
        best_accuracy = acc
        best_n = n

print(f"\nBest Accuracy: {best_accuracy:.4f} with {best_n} trees")

plt.plot(range(10, 101, 10), accuracy_scores, marker='o')
plt.title('AdaBoost Accuracy vs Number of Trees')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

```
Default Accuracy (10 trees): 0.8182
Confusion Matrix (10 trees):
[[6782  632]
 [1144 1211]]

Best Accuracy: 0.8335 with 80 trees
```

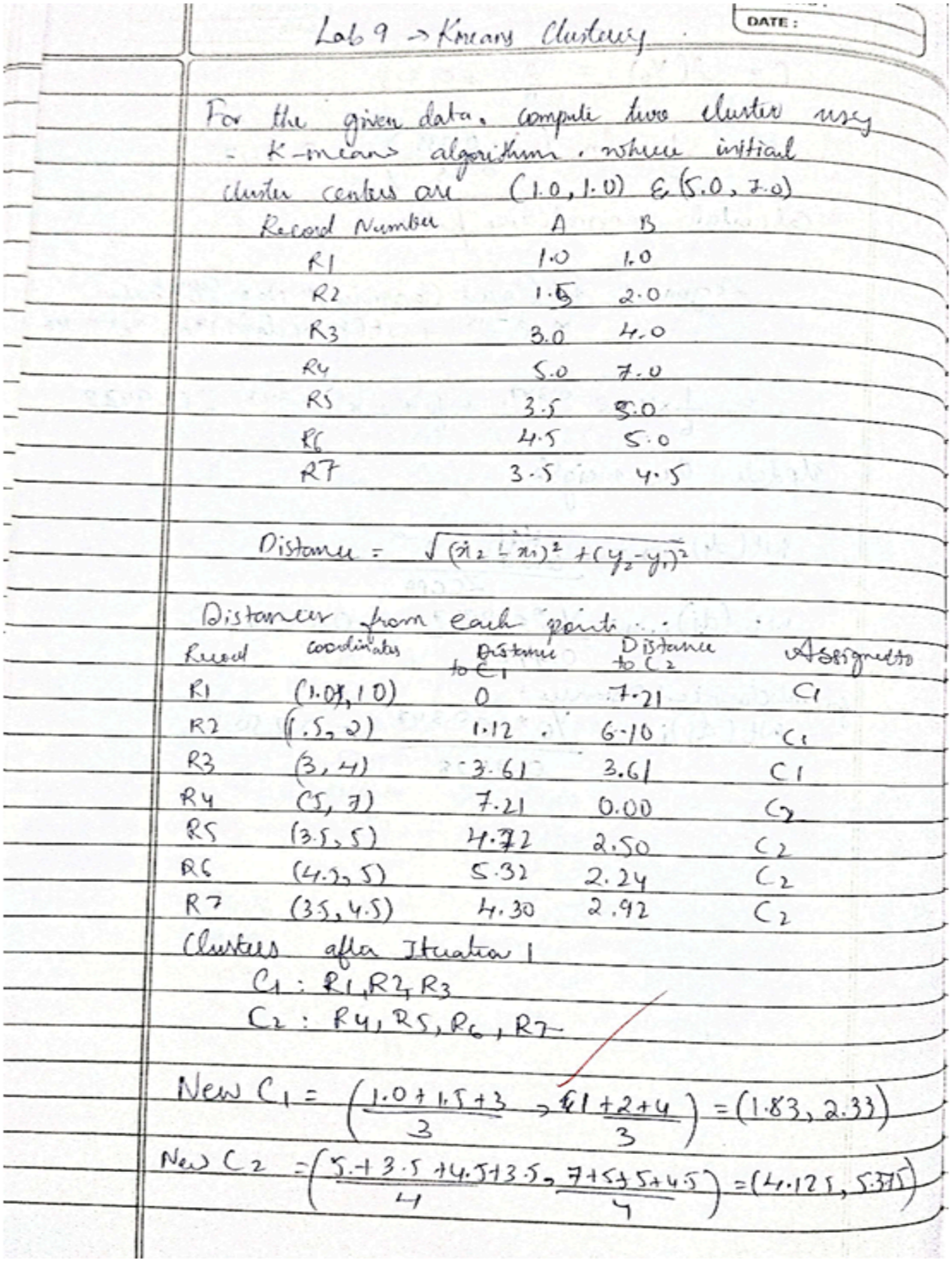


In [ ]:

Program 10

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

Observation Book:



Lab 9 → Kmeans Clustering

DATE :

For the given data, compute two cluster using K-means algorithm, where initial cluster centers are (1.0, 1.0) & (5.0, 7.0)

Record Number	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

Distance = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Distances from each points...

Record	coordinates	Distance to C1	Distance to C2	Assigned to
R1	(1.0, 1.0)	0	7.21	C1
R2	(1.5, 2)	1.12	6.10	C1
R3	(3, 4)	3.61	3.61	C1
R4	(5, 7)	7.21	0.00	C2
R5	(3.5, 5)	4.72	2.50	C2
R6	(4.5, 5)	5.32	2.24	C2
R7	(3.5, 4.5)	4.30	2.92	C2

Clusters after Iteration 1

C1 : R1, R2, R3

C2 : R4, R5, R6, R7

New $C_1 = \left(\frac{1.0+1.5+3}{3}, \frac{1+2+4}{3}\right) = (1.83, 2.33)$

New $C_2 = \left(\frac{5+3.5+4.5+3.5}{4}, \frac{7+5+5+4.5}{4}\right) = (4.125, 5.375)$

PAGE NO:
DATE:

Iteration 2.

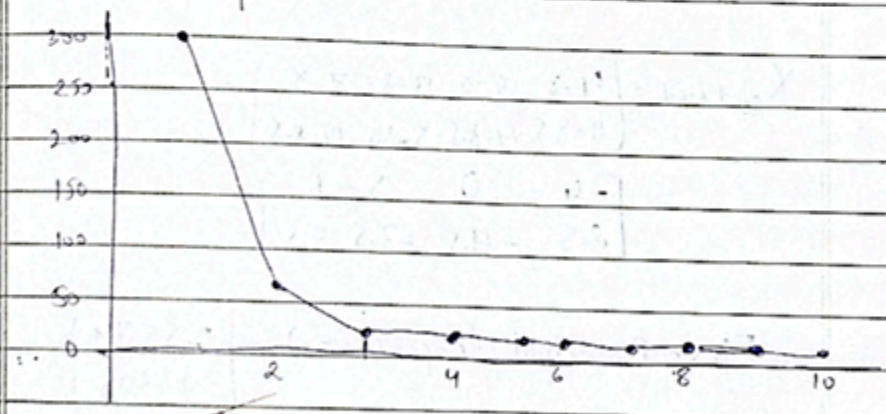
Record	Dist to (1.83, 2.33)	Dist to (4.93, 5.33)	Assigned to
R_1	1.57	5.59	C_1
R_2	0.47	4.64	C_1
R_3	2.17	1.64	C_2
R_4	5.03	1.97	C_2
R_5	3.37	0.71	C_2
R_6	3.95	0.58	C_2
R_7	3.14	1.01	C_2

Final clusters

Cluster 1 : R_1, R_2

Cluster 2 : R_3, R_4, R_5, R_6, R_7

Elbow plot :-



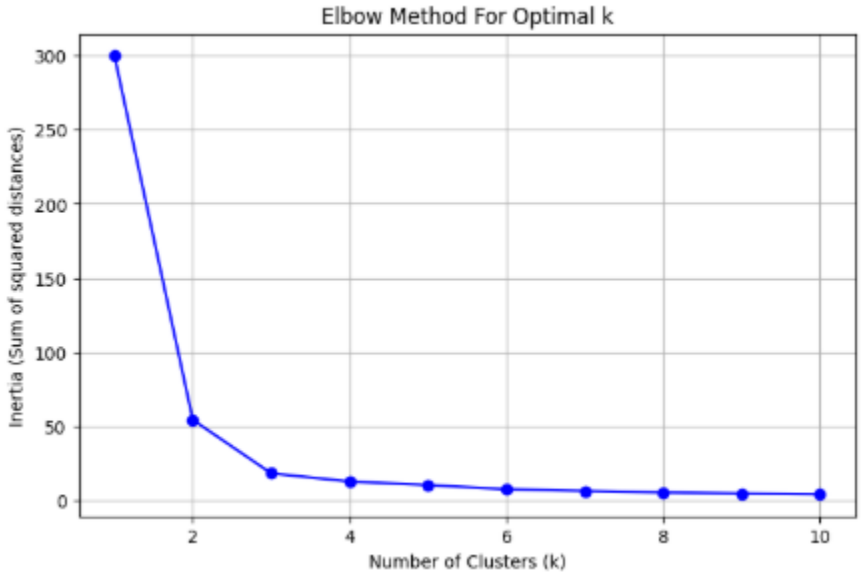
Since elbow occurs at K = 3,

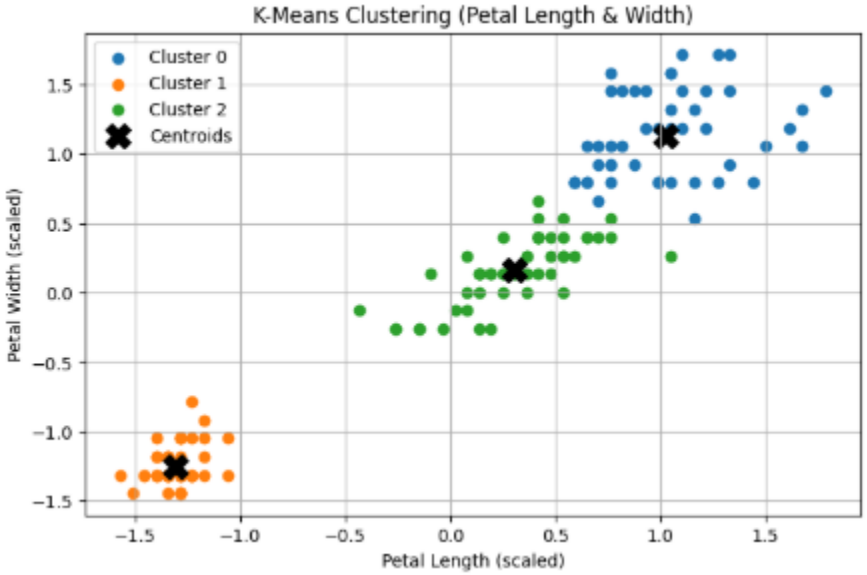
Optimal K value is 3.

Code:

In [1]:

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

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

# Use only petal length and petal width
X = df[['petal_length', 'petal_width']]

# Optional: Feature Scaling (recommended for distance-based algorithms)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Elbow method to find optimal k
inertia = []
K_range = range(1, 11)

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

# Plot the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, 'bo-')
plt.title("Elbow Method For Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia (Sum of squared distances)")
plt.grid(True)
plt.show()

# After observing the elbow plot, you can pick optimal k, e.g., k=3
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# Add cluster labels to the original DataFrame for visualization
df['cluster'] = clusters

# Visualize the clusters
plt.figure(figsize=(8, 5))
for i in range(optimal_k):
    plt.scatter(
        X_scaled[clusters == i, 0],
        X_scaled[clusters == i, 1],
        label=f'Cluster {i}'
    )

plt.scatter(
    kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1],
    s=200, c='black', marker='X', label='Centroids'
)
plt.title("K-Means Clustering (Petal Length & Width)")
plt.xlabel("Petal Length (scaled)")
plt.ylabel("Petal Width (scaled)")
plt.legend()
plt.grid(True)
plt.show()
```

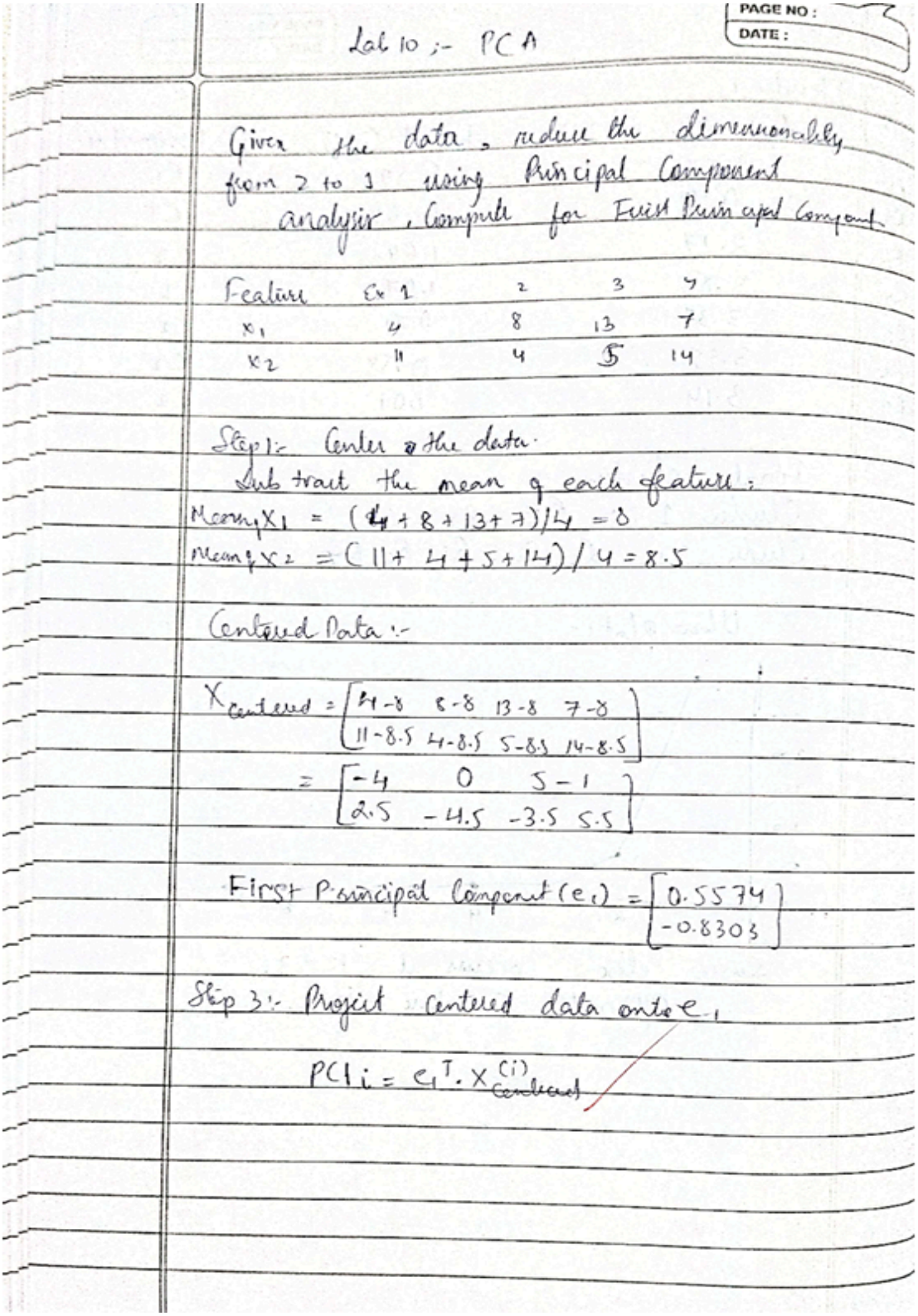

Elbow Method For Optimal k
Inertia (Sum of squared distances)
Number of Clusters (k)
300
250
200
150
100
50
0
2
4
6
8
10


K-Means Clustering (Petal Length & Width)
Cluster 0
Cluster 1
Cluster 2
Centroids
Petal Width (scaled)
Petal Length (scaled)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

Program 11

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

Observation Book:



Lab 10 :- PCA

PAGE NO :
DATE :

Given the data, reduce the dimensionality from 2 to 1 using Principal Component analysis, Compute for First Principal Component

Feature	Ex 1	2	3	4
x_1	4	8	13	7
x_2	11	4	5	14

Step 1:- Center the data.
Subtract the mean of each feature:

Mean of $x_1 = (4 + 8 + 13 + 7)/4 = 8$

Mean of $x_2 = (11 + 4 + 5 + 14)/4 = 8.5$

Centered Data :-

$$X_{centered} = \begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.5 & 4-8.5 & 5-8.5 & 14-8.5 \end{bmatrix}$$

$$= \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$$

First Principal Component $(e_1) = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$

Step 3:- Project centered data onto e_1

$$PC1_i = e_1^T \cdot X_{centered}^{(i)}$$

ex1:

$PC1_1 = 0.5574 \times (-4) + (-0.8303) \times 2.5 =$

$= -2.2296 - 2.07575 = -4.30535$

$PC1_2 = 0.557 \times 0 + (-0.8303) \times (-4.5) = 3.73635$

$PC1_3 = 5.69305$

$PC1_4 = -5.12405$

Final 1D data.

Example	PC1 value
1	-4.30535
2	3.73635
3	5.69305
4	-5.12405

Accuracy without PCA

SVM: 0.8689

Logistic Regression: 0.8852

Accuracy with PCA

SVM: 0.8525

Logistic Regression: 0.8689.

Code:

In [2]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
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 dataset
df = pd.read_csv("heart.csv")

# Identify categorical columns
cat_cols = df.select_dtypes(include=['object']).columns.tolist()

# Label Encode for binary columns and One-Hot Encode for multiclass
le = LabelEncoder()
for col in cat_cols:
    if df[col].nunique() == 2:
        df[col] = le.fit_transform(df[col])
    else:
        df = pd.get_dummies(df, columns=[col])

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

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

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

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

# Train and evaluate
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}")

# Apply PCA
pca = PCA(n_components=0.95)  # Keep 95% variance
X_pca = pca.fit_transform(X_scaled)

# Train-test split again with PCA
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Train and evaluate again
print("\n📉 Accuracy 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}")
```

```
🔍 Accuracy without PCA:
SVM: 0.8689
Logistic Regression: 0.8525
Random Forest: 0.8689

📉 Accuracy with PCA:
SVM: 0.8361
Logistic Regression: 0.8525
Random Forest: 0.8689
```