VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Neelvani Varsha Vittal (1BM23CS412)

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 **Neelvani Varsha Vittal (1BM23CS412),** 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.

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

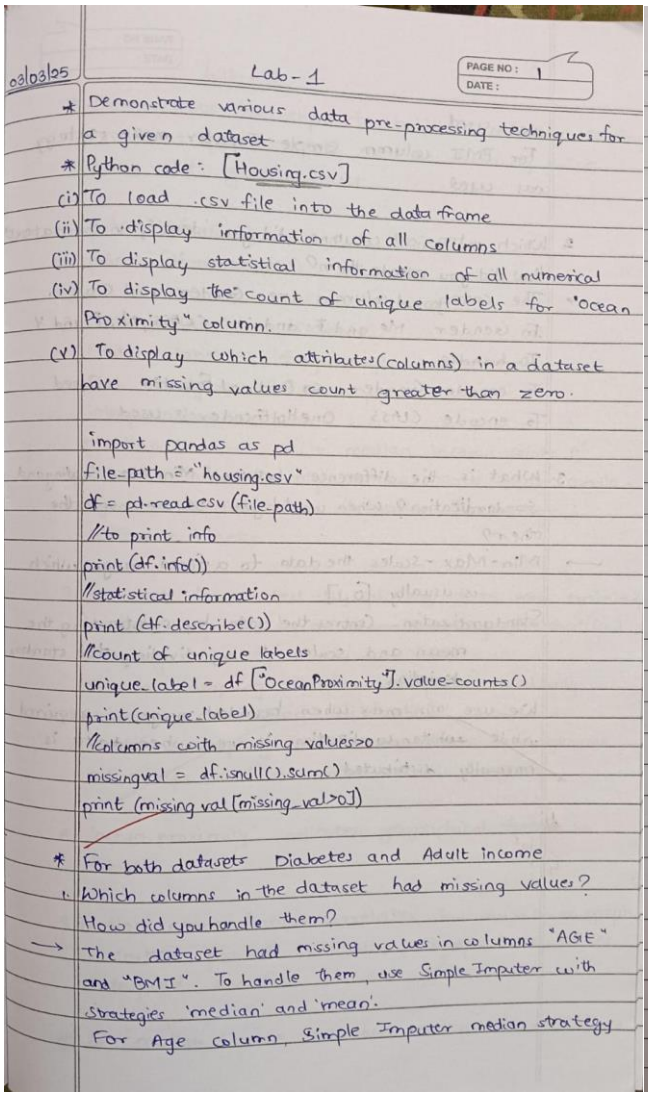
Index

Github Link: https://github.com/NeelvaniVarsha/MLLab.git

Program 1

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

Screenshot:



03/03/25

Lab - 1

PAGE NO : 1
DATE :

* Demonstrate various data pre-processing techniques for a given dataset
* Python code : [Housing.csv]

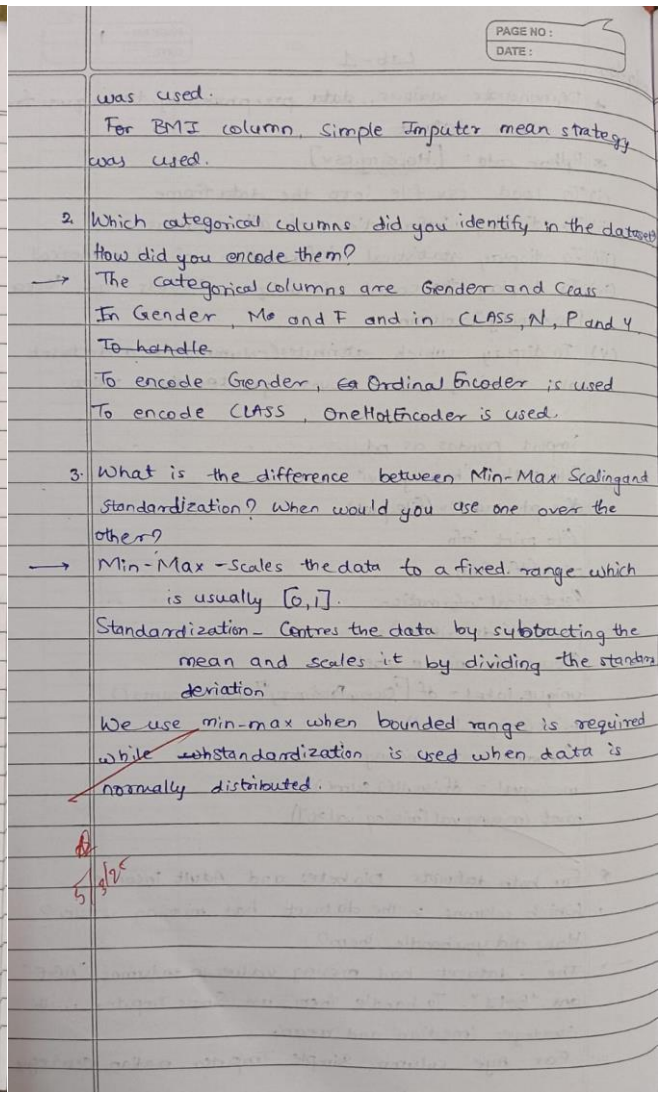
(i) To load .csv file into the data frame
(ii) To display information of all columns
(iii) To display statistical information of all numerical
(iv) To display the count of unique labels for 'Ocean Proximity" column.
(v) To display which attributes(columns) in a dataset have missing values count greater than zero.

```
import pandas as pd
file_path = "housing.csv"
df = pd.read_csv(file_path)
//to print info
print(df.info())
//statistical information
print(df.describe())
//count of unique labels
unique_label = df["OceanProximity"].value_counts()
print(unique_label)
//columns with missing values>0
missingval = df.isnull().sum()
print(missingval[missing_val>0])
```

* For both datasets Diabetes and Adult income

1. Which columns in the dataset had missing values? How did you handle them?

→ The dataset had missing values in columns "AGE" and "BMI". To handle them, use SimpleImputer with strategies 'median' and 'mean'.
For Age column, SimpleImputer median strategy was used.
For BMI column, SimpleImputer mean strategy was used.



PAGE NO :
DATE :

2. Which categorical columns did you identify in the dataset? How did you encode them?

→ The categorical columns are Gender and Class
In Gender, M and F and in CLASS, N, P and Y
To handle
To encode Gender, an OrdinalEncoder is used
To encode CLASS, OneHotEncoder is used.

3. What is the difference between Min-Max Scaling and Standardization? When would you use one over the other?

→ Min-Max - Scales the data to a fixed range which is usually [0,1].
Standardization - Centres the data by subtracting the mean and scales it by dividing the standard deviation
We use min-max when bounded range is required while ~~un~~standardization is used when data is normally distributed.

Code with output:

In [1]:
```
import pandas as pd
```

In [2]:
```
try:
    data = pd.read_csv('sample_data.csv')
    print("Data imported from CSV:")
    print(data)
except FileNotFoundError:
    print("sample_data.csv not found. Creating sample data instead.")
    data = pd.DataFrame({
        'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'Los Angeles', 'Chicago']
    })
    print(data)
```

```
sample_data.csv not found. Creating sample data instead.
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
```

In [3]:
```
data.to_csv('exported_data.csv', index=False)
print("\nData exported to 'exported_data.csv'.")
```

```
Data exported to 'exported_data.csv'.
```

In [5]:
```
pd.read_csv('exported_data.csv')
```

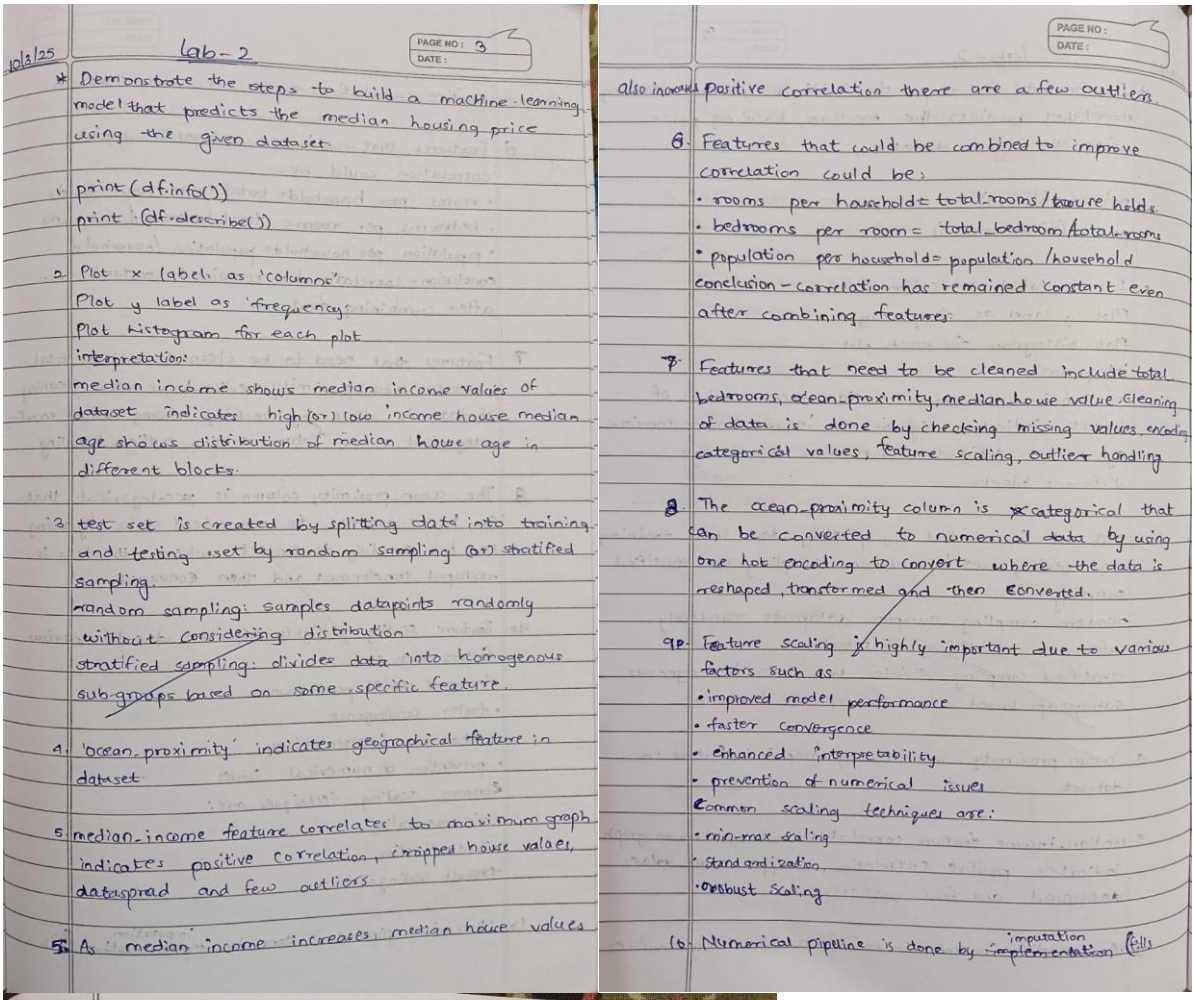
Out[5]:

	Name	Age	City
0	Alice	25	New York
1	Bob	30	Los Angeles
2	Charlie	35	Chicago

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



10/3/25 Lab - 2

PAGE NO: 3
DATE:

* Demonstrate the steps to build a machine learning model that predicts the median housing price using the given dataset

1. print(df.info())
print(df.describe())

2. Plot x label as 'columns'
Plot y label as 'frequency'
Plot histogram for each plot
interpretation:
median income shows median income values of dataset indicates high (or) low income house median age shows distribution of median house age in different blocks.

3. test set is created by splitting data into training and testing set by random sampling (or) stratified sampling.
random sampling: samples datapoints randomly without considering distribution
stratified sampling: divides data into homogenous sub-groups based on some specific feature.

4. 'ocean_proximity' indicates geographical feature in dataset

5. median_income feature correlates to maximum graph indicates positive correlation, cropped house values, data spread and few outliers

6. As median income increases, median house values also increases positive correlation there are a few outliers

7. Features that could be combined to improve correlation could be:
- rooms per household = total_rooms / houre holds
- bedrooms per room = total_bedroom / total_rooms
- population per household = population / household
conclusion - correlation has remained constant even after combining features.

8. Features that need to be cleaned include total_bedrooms, ocean_proximity, median_howe value. Cleaning of data is done by checking missing values, encoding categorical values, feature scaling, outlier handling

9. The ocean_proximity column is categorical that can be converted to numerical by using one hot encoding to convert where the data is reshaped, transformed and then converted.

10. Feature scaling is highly important due to various factors such as
- improved model performance
- faster convergence
- enhanced interpretability
- prevention of numerical issues
Common scaling techniques are:
- min-max scaling
- standardization
- Robust scaling

11. Numerical pipeline is done by imputation (fills missing value with median value), custom transformations (adding features), scaling
Categorical pipelining is done by one hot encoding

10/3/25

Code with output:

In [2]:
```
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
```

In [4]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving diabetes.csv to diabetes.csv

In [5]:
```
import pandas as pd
df = pd.read_csv('diabetes.csv')
df.head(10)
```

Out[5]:

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N
5	634	34224	F	45	2.3	24	4.0	2.9	1.0	1.0	1.5	0.4	21.0	N
6	721	34225	F	50	2.0	50	4.0	3.6	1.3	0.9	2.1	0.6	24.0	N
7	421	34227	M	48	4.7	47	4.0	2.9	0.8	0.9	1.6	0.4	24.0	N
8	670	34229	M	43	2.6	67	4.0	3.8	0.9	2.4	3.7	1.0	21.0	N
9	759	34230	F	32	3.6	28	4.0	3.8	2.0	2.4	3.8	1.0	24.0	N

In [6]:
```
df.shape
```

Out[6]: (1000, 14)

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ID         1000 non-null   int64
 1   No_Pation  1000 non-null   int64
 2   Gender     1000 non-null   object
 3   AGE        1000 non-null   int64
 4   Urea       1000 non-null   float64
 5   Cr         1000 non-null   int64
 6   HbA1c      1000 non-null   float64
 7   Chol       1000 non-null   float64
 8   TG         1000 non-null   float64
 9   HDL        1000 non-null   float64
 10  LDL        1000 non-null   float64
 11  VLDL       1000 non-null   float64
 12  BMI        1000 non-null   float64
 13  CLASS      1000 non-null   object
dtypes: float64(8), int64(4), object(2)
memory usage: 109.5+ KB
```

In [8]:
```
df.describe()
```

Out[8]:

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	L
count	1000.000000	1.000000e+03	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.0000
mean	340.500000	2.705514e+05	53.528000	5.124743	68.943000	8.281160	4.862820	2.349610	1.204750	2.6097
std	240.397673	3.380758e+06	8.799241	2.935165	59.984747	2.534003	1.301738	1.401176	0.660414	1.1151
min	1.000000	1.230000e+02	20.000000	0.500000	6.000000	0.900000	0.000000	0.300000	0.200000	0.3000
25%	125.750000	2.406375e+04	51.000000	3.700000	48.000000	6.500000	4.000000	1.500000	0.900000	1.8000
50%	300.500000	3.439550e+04	55.000000	4.600000	60.000000	8.000000	4.800000	2.000000	1.100000	2.5000
75%	550.250000	4.538425e+04	59.000000	5.700000	73.000000	10.200000	5.600000	2.900000	1.300000	3.3000
max	800.000000	7.543566e+07	79.000000	38.900000	800.000000	16.000000	10.300000	13.800000	9.900000	9.9000

In [9]:
```
df.loc[5, 'AGE'] = np.nan
df.loc[10, 'BMI'] = np.nan
df.head(11)
```

Out[9]:

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26.0	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33.0	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N
5	634	34224	F	NaN	2.3	24	4.0	2.9	1.0	1.0	1.5	0.4	21.0	N
6	721	34225	F	50.0	2.0	50	4.0	3.6	1.3	0.9	2.1	0.6	24.0	N
7	421	34227	M	48.0	4.7	47	4.0	2.9	0.8	0.9	1.6	0.4	24.0	N
8	670	34229	M	43.0	2.6	67	4.0	3.8	0.9	2.4	3.7	1.0	21.0	N
9	759	34230	F	32.0	3.6	28	4.0	3.8	2.0	2.4	3.8	1.0	24.0	N
10	636	34231	F	31.0	4.4	55	4.2	3.6	0.7	1.7	1.6	0.3	NaN	N

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ID         1000 non-null   int64
 1   No_Pation  1000 non-null   int64
 2   Gender     1000 non-null   object
 3   AGE        999 non-null    float64
 4   Urea       1000 non-null   float64
 5   Cr         1000 non-null   int64
 6   HbA1c      1000 non-null   float64
 7   Chol       1000 non-null   float64
 8   TG         1000 non-null   float64
 9   HDL        1000 non-null   float64
 10  LDL        1000 non-null   float64
 11  VLDL       1000 non-null   float64
 12  BMI        999 non-null    float64
 13  CLASS      1000 non-null   object
dtypes: float64(9), int64(3), object(2)
memory usage: 109.5+ KB
```

In [11]:
```
df.describe()
```

Out[11]:

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LD
count	1000.000000	1.000000e+03	999.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	340.500000	2.705514e+05	53.536537	5.124743	68.943000	8.281160	4.862820	2.349610	1.204750	2.60979
std	240.397673	3.380758e+06	8.799504	2.935165	59.984747	2.534003	1.301738	1.401176	0.660414	1.11510
min	1.000000	1.230000e+02	20.000000	0.500000	6.000000	0.900000	0.000000	0.300000	0.200000	0.30000
25%	125.750000	2.406375e+04	51.000000	3.700000	48.000000	6.500000	4.000000	1.500000	0.900000	1.80000
50%	300.500000	3.439550e+04	55.000000	4.600000	60.000000	8.000000	4.800000	2.000000	1.100000	2.50000
75%	550.250000	4.538425e+04	59.000000	5.700000	73.000000	10.200000	5.600000	2.900000	1.300000	3.30000
max	800.000000	7.543566e+07	79.000000	38.900000	800.000000	16.000000	10.300000	13.800000	9.900000	9.90000

In [12]:
```
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])
```

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

In [13]:
```
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

imputer1.fit(df_copy[["AGE"]])
imputer2.fit(df_copy[["BMI"]])

df_copy["AGE"] = imputer1.transform(df[["AGE"]])
df_copy["BMI"] = imputer2.transform(df[["BMI"]])

print(df_copy["AGE"].isnull().sum())
print(df_copy["BMI"].isnull().sum())
```

```
0
0
```

In [14]:

```
ordinal_encoder = OrdinalEncoder(categories=[["M", "F"]])
df_copy['Gender'] = df_copy['Gender'].str.upper()

df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])

onehot_encoder = OneHotEncoder(sparse_output=False)

encoded_data = onehot_encoder.fit_transform(df_copy[["CLASS"]])

encoded_df = pd.DataFrame(encoded_data, columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df_copy, encoded_df], axis=1)

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

print(df_encoded. 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             1.0      1.0       0.0      0.0      0.0      0.0
1             0.0      1.0       0.0      0.0      0.0      0.0
2             1.0      1.0       0.0      0.0      0.0      0.0
3             1.0      1.0       0.0      0.0      0.0      0.0
4             0.0      1.0       0.0      0.0      0.0      0.0
```

In [15]:

```
normalizer = MinMaxScaler()
df_encoded[['BMI']] = normalizer.fit_transform(df_encoded[['BMI']])
df_encoded.head()
```

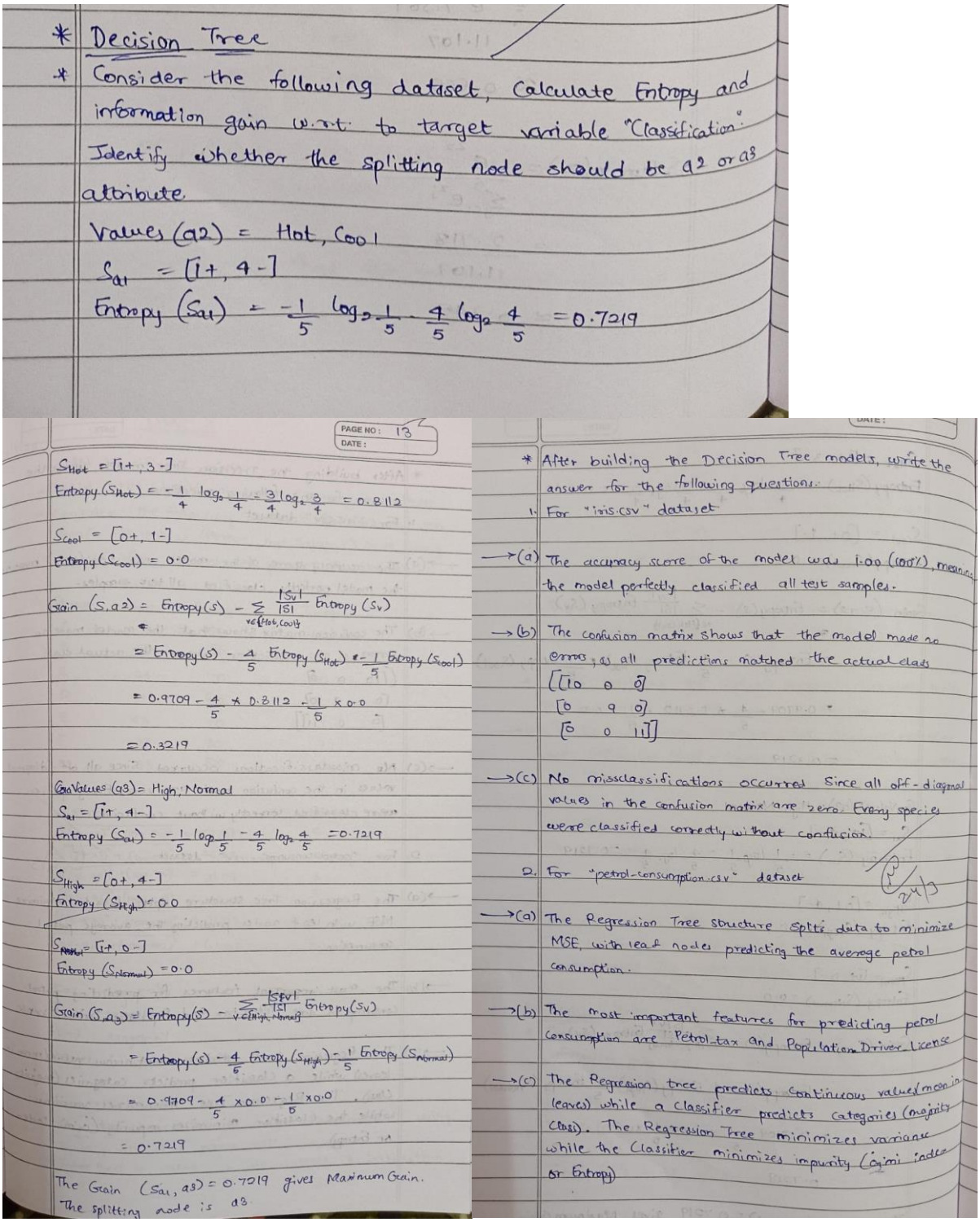
Out[15]:

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	Gender_Encoded	CLASS_N	CLASS_N	CLASS_P	CLA
0	502	17975	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	0.173913	1.0	1.0	0.0	0.0	
1	735	34221	26.0	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	0.139130	0.0	1.0	0.0	0.0	
2	420	47975	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	0.173913	1.0	1.0	0.0	0.0	
3	680	87656	50.0	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	0.173913	1.0	1.0	0.0	0.0	
4	504	34223	33.0	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	0.069565	0.0	1.0	0.0	0.0	

Program 3

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

Screenshot:



* Decision Tree
* Consider the following dataset, Calculate Entropy and information gain w.r.t to target variable "Classification". Identify whether the splitting node should be a2 or a3 attribute.

Values (a2) = Hot, Cool

$S_{all} = [1+, 4-]$

$Entropy(S_{all}) = -\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5} = 0.7219$

PAGE NO: 13
DATE:

$S_{Hot} = [1+, 3-]$

$Entropy(S_{Hot}) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.8112$

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

$Entropy(S_{Cool}) = 0.0$

$Gain(S, a2) = Entropy(S) - \sum_{v \in \{Hot, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$

$= Entropy(S) - \frac{4}{5} Entropy(S_{Hot}) - \frac{1}{5} Entropy(S_{Cool})$

$= 0.9709 - \frac{4}{5} \times 0.8112 - \frac{1}{5} \times 0.0$

$= 0.3219$

a3 Values (a3) = High, Normal

$S_{all} = [1+, 4-]$

$Entropy(S_{all}) = -\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5} = 0.7219$

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

$Entropy(S_{High}) = 0.0$

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

$Entropy(S_{Normal}) = 0.0$

$Gain(S, a3) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$

$= Entropy(S) - \frac{4}{5} Entropy(S_{High}) - \frac{1}{5} Entropy(S_{Normal})$

$= 0.9709 - \frac{4}{5} \times 0.0 - \frac{1}{5} \times 0.0$

$= 0.7219$

The Grain $(S_{all}, a3) = 0.7219$ gives Maximum Grain.
The splitting node is a3.

* After building the Decision Tree models, write the answer for the following questions:

1. For "iris.csv" dataset

→(a) The accuracy score of the model was 1.00 (100%), meaning the model perfectly classified all test samples.

→(b) The confusion matrix shows that the model made no errors, as all predictions matched the actual class.

[[10 0 0]
[0 9 0]
[0 0 11]]

→(c) No misclassifications occurred. Since all off-diagonal values in the confusion matrix are zero. Every species were classified correctly without confusion.

2. For "petrol-consumption.csv" dataset

→(a) The Regression Tree structure splits data to minimize MSE, with leaf nodes predicting the average petrol consumption.

→(b) The most important features for predicting petrol consumption are Petrol_tax and Population_Driver_License.

→(c) The Regression tree predicts continuous values (mean in leaves) while a classifier predicts categories (majority class). The Regression Tree minimizes variance while the Classifier minimizes impurity (Gini index or Entropy).

Code with output:

In [55]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
```

In [2]:
```
from google.colab import files
uploaded = files.upload()
```

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

In [3]:
```
df = pd.read_csv("iris (1).csv")
df.head()
```

Out[3]:

	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

In [4]:
```
df.columns
```

Out[4]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

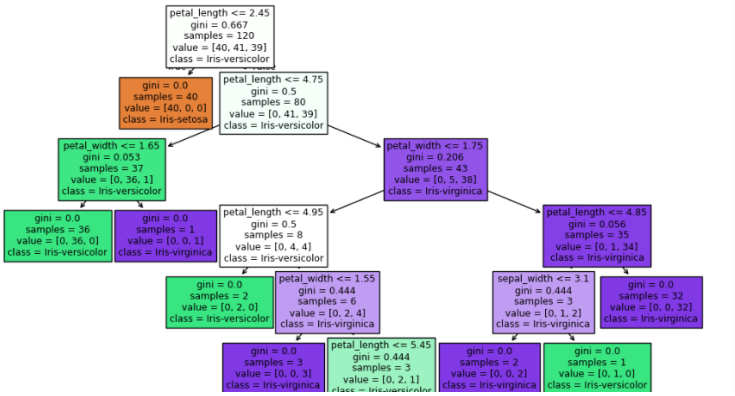
In [5]:
```
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]  # Independent variables
y = df['species']
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [9]:
```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy Score: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy Score: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [19]:
```
plt.figure(figsize=(12, 8))
tree.plot_tree(
    clf,
    feature_names=X.columns,
    class_names=clf.classes_.astype(str),
    filled=True
)
plt.show()
```



In [31]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving drug.csv to drug.csv

In [32]:
```
df = pd.read_csv("drug.csv")
df.head()
```

Out[32]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

In [33]:
```
df.columns
```

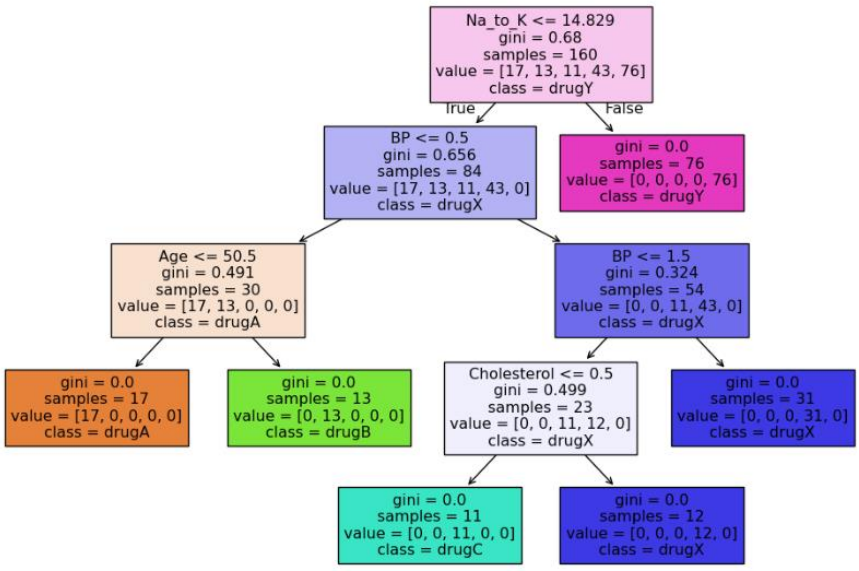
Out[33]:
```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='object')
```

In [38]:
```
from sklearn.preprocessing import OrdinalEncoder

# Make a copy of the data
X = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].copy()

# Columns to encode
categorical_cols = ['Sex', 'BP', 'Cholesterol']

# Apply encoding
encoder = OrdinalEncoder()
X[categorical_cols] = encoder.fit_transform(X[categorical_cols])

y = df['Drug']
```

In [39]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
Accuracy Score: 1.00
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]]
```

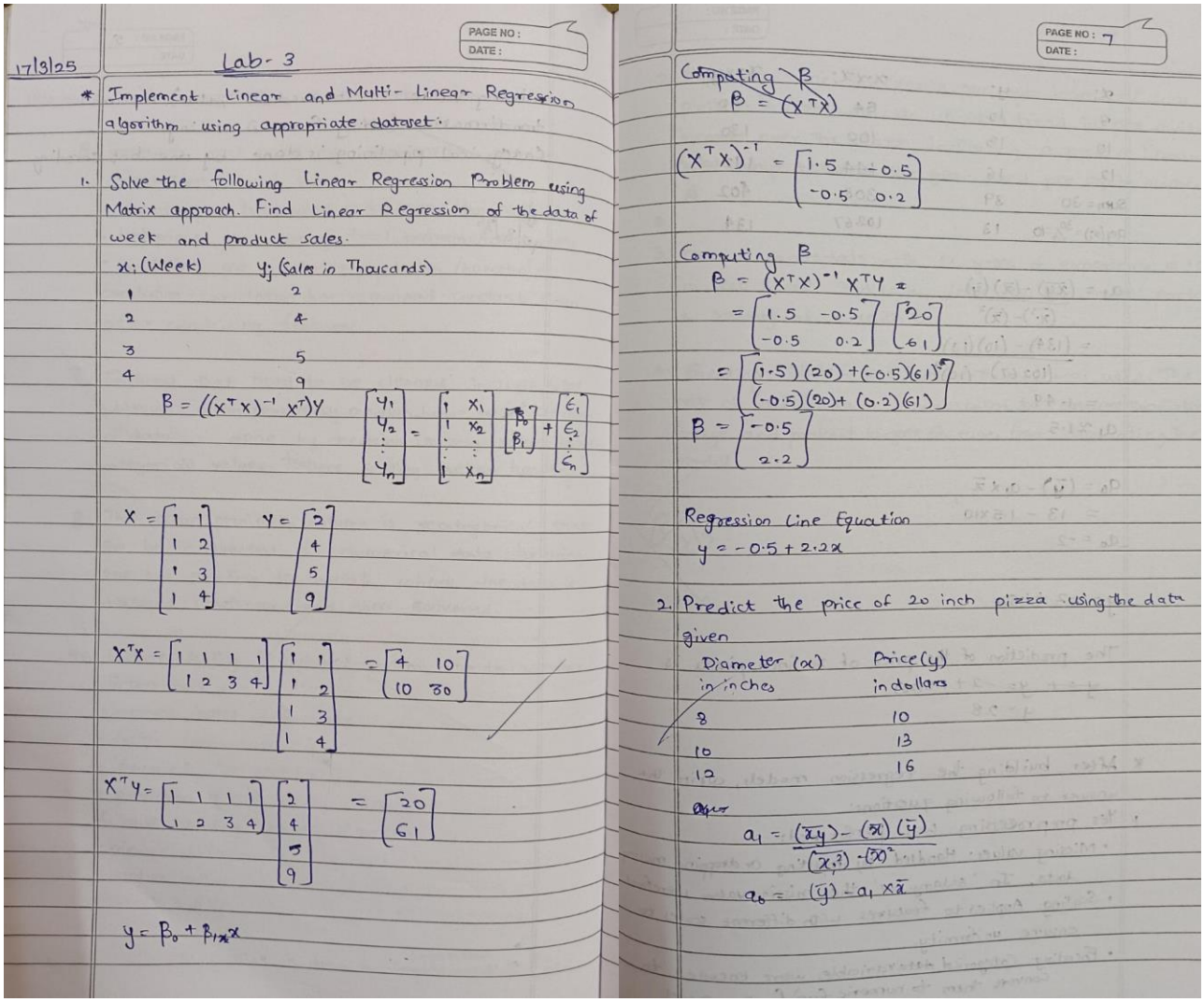
In [43]:
```
plt.figure(figsize=(12, 8))
tree.plot_tree(
    clf,
    feature_names=X.columns,
    class_names=clf.classes_.astype(str),
    filled=True
)
plt.show()
```



Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:



17/3/25 Lab- 3

* Implement Linear and Multi- Linear Regression algorithm using appropriate dataset.

1. Solve the following Linear Regression Problem using Matrix approach. Find Linear Regression of the data of week and product sales.

x_i (Week)	y_i (Sales in Thousands)
1	2
2	4
3	5
4	9

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

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

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

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

$$X^T Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix} = \begin{bmatrix} 20 \\ 61 \end{bmatrix}$$

$$y = \beta_0 + \beta_1 x$$

PAGE NO: 7
DATE:

Computing β

$\beta = (X^T X)$

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

Computing β

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

$$= \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix} \begin{bmatrix} 20 \\ 61 \end{bmatrix}$$

$$= \begin{bmatrix} (1.5)(20) + (-0.5)(61) \\ (-0.5)(20) + (0.2)(61) \end{bmatrix}$$

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

Regression Line Equation

$y = -0.5 + 2.2x$

2. Predict the price of 20 inch pizza using the data given

Diameter (x) in inches	Price (y) in dollars
8	10
10	13
12	16

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

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

PAGE NO :
DATE :

x_i	y_i	$x_i \times x_i$	$x_i \times y_i$
8	10	64	80
10	13	100	130
12	16	144	192
Sum = 30	39	308	402
avg $(x_i) = \frac{30}{3} = 10$	13	102.67	134

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

$$= \frac{(134) - (10)(13)}{(102.67) - (10)^2}$$

$$= 1.49$$

$a_1 \approx 1.5$

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

$$= 13 - 1.5 \times 10$$

$a_0 = -2$

$y = -2 + 1.5x$

The prediction of the price of 20 inch pizza is

$y = -2 + 1.5(20)$

$y = 28$

* After building the regression models, write the answer to following questions:

1. Yes, preprocessing was performed:
   - Missing values: Handled by imputing or dropping missing data. In 'salary.csv' the missing values were found
   - Scaling: Applied to features with different scales to ensure uniformity.
   - Encoding: Categorical data variables were encoded to convert them to numeric form for the model.

PAGE NO : 9
DATE :

2. Regression line for canada_per_capita_income.csv was plotted. It showed an upward trend in per capita income over the years, indicating a positive linear relationship between the year and per capita income.

3. For a candidate with 12 years of experience, a 10 in test score and 10 in interview the model predicted a salary of $69629.25

4. Encoding: State - one hot encoding was used. The rest of the features were scaled to ensure consistent range and prevent larger features from dominating the model.

Code with output:

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

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

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

In [ ]:
```
df = pd.read_csv('housing_area_price.csv')
df
```

Out[ ]:

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

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

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7de8cb10bdd0>
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:

	0
area	0
price	0

dtype: int64

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

Out[ ]:

	area
0	2600
1	3000
2	3200
3	3600
4	4000

In [ ]:
```
price = df['price']
price
```

Out[ ]:

	price
0	550000
1	565000
2	610000
3	680000
4	725000

dtype: int64

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

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
reg.predict([[2030]])
```

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

Out[ ]: array([456265.4109589])

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving canada_per_capita_income.csv to canada_per_capita_income (1).csv

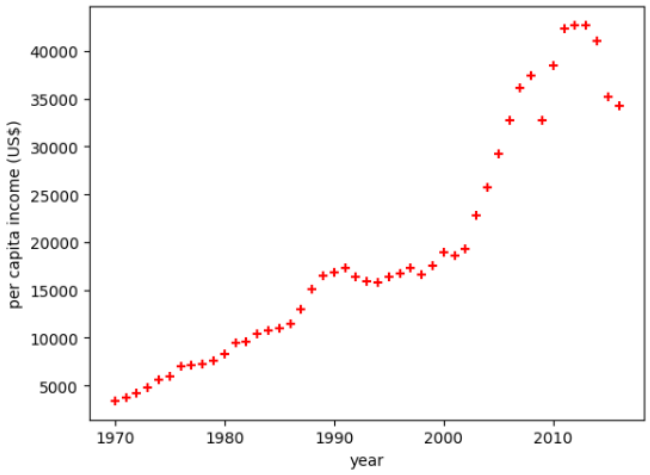
In [ ]:
```
df = pd.read_csv('canada_per_capita_income.csv')
df
```

Out[ ]:

	year	per capita income (US$)
0	1970	3399.299037
1	1971	3768.297935
2	1972	4251.175484
3	1973	4804.463248
4	1974	5576.514583
5	1975	5998.144346
6	1976	7062.131392
7	1977	7100.126170
8	1978	7247.967035
9	1979	7602.912681
10	1980	8355.968120
11	1981	9434.390652
12	1982	9619.438377
13	1983	10416.536590
14	1984	10790.328720
15	1985	11018.955850
16	1986	11482.891530

In [ ]:
```
plt.xlabel('year')
plt.ylabel('per capita income (US$)')
plt.scatter(df['year'],df['per capita income (US$)'],color='red',marker='+')
```

Out[ ]: <matplotlib.collections.PathCollection at 0x7de8ca285b10>



In [ ]:
```
df.isnull().sum()
```

Out[ ]:

	0
year	0
per capita income (US$)	0

dtype: int64

30	2000
31	2001
32	2002
33	2003
34	2004
35	2005
36	2006
37	2007
38	2008
39	2009
40	2010
41	2011
42	2012
43	2013
44	2014
45	2015
46	2016

In [ ]:
```
X = df.drop('per capita income (US$)', axis=1)
y = df['per capita income (US$)']
```

In [ ]:
```
reg = linear_model.LinearRegression()
reg.fit(X,y)
```

Out[ ]:
```
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.

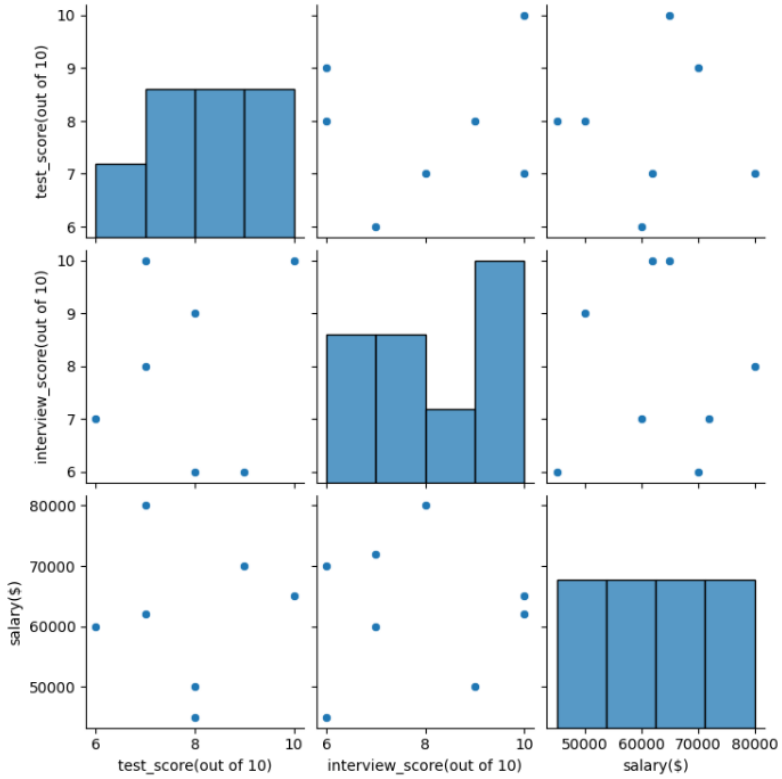
In [ ]:
```
reg.predict([[2030]])
```

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

Out[ ]:
```
array([49573.34484664])
```

In [ ]:
```
import seaborn as sns
sns.pairplot(df)
```

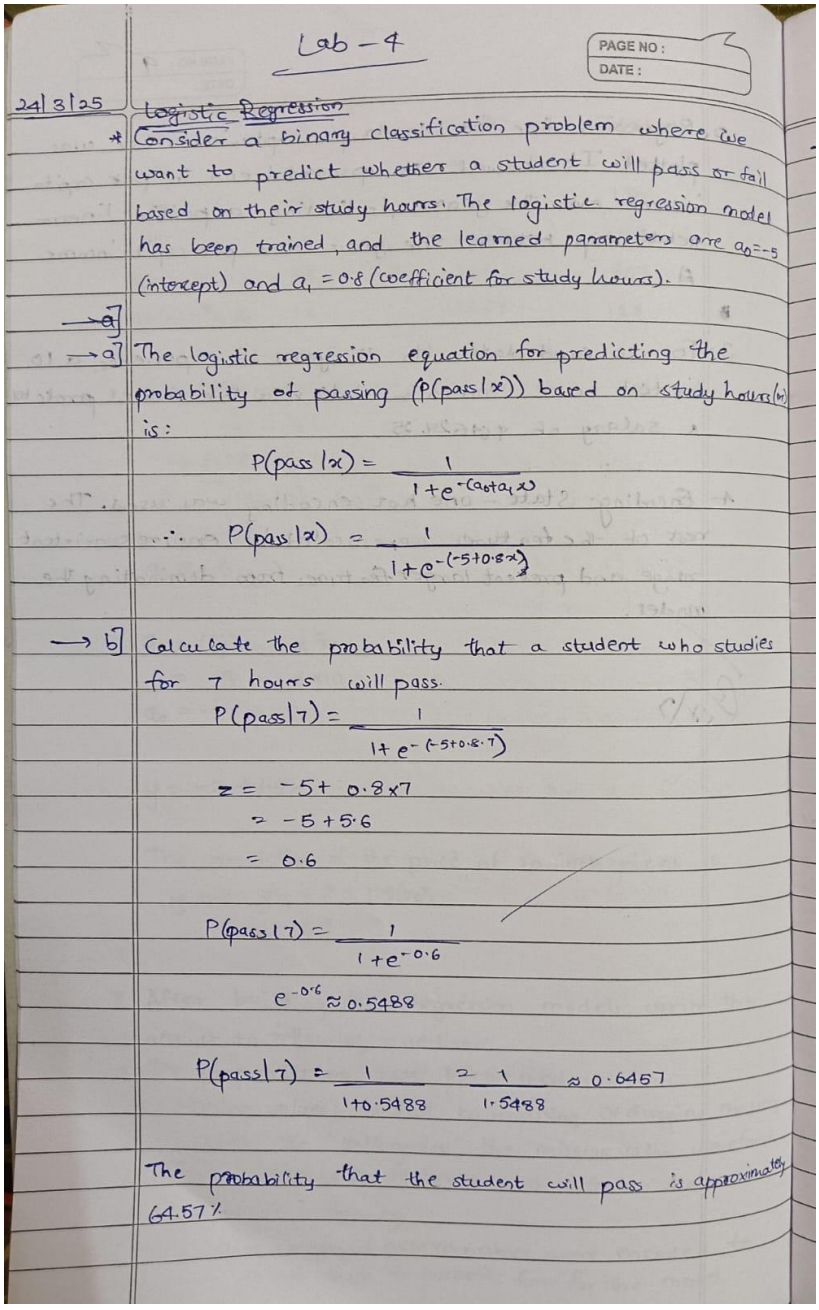
Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7de8ca449150>
```



Program 5

Build Logistic Regression Model for a given dataset

Screenshot:



Lab – 4

PAGE NO :
DATE :

24/3/25

~~Logistic Regression~~

* 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 $a_0 = -5$ (intercept) and $a_1 = 0.8$ (coefficient for study hours).

→ a]

→ a] The logistic regression equation for predicting the probability of passing ($P(pass|x)$) based on study hours (x) is:

$$P(pass|x) = \frac{1}{1+e^{-(a_0+a_1 x)}}$$

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

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

$$P(pass|7) = \frac{1}{1+e^{-(-5+0.8\cdot 7)}}$$

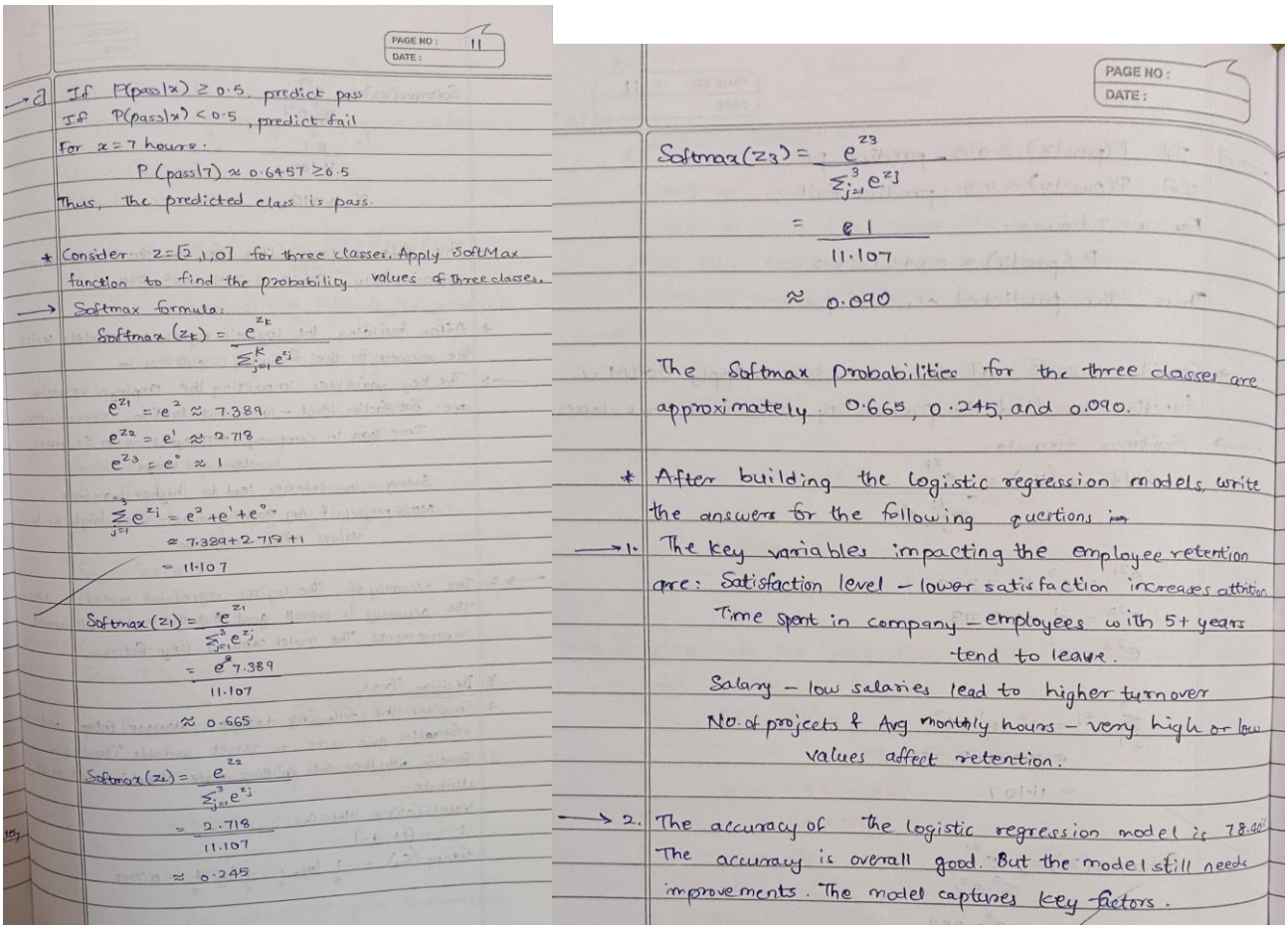
$$z = -5 + 0.8 \times 7$$
$$= -5 + 5.6$$
$$= 0.6$$

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

$$e^{-0.6} \approx 0.5488$$

$$P(pass|7) = \frac{1}{1+0.5488} = \frac{1}{1.5488} \approx 0.6457$$

The probability that the student will pass is approximately 64.57%



Code with output:

In [1]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving HR_comma_sep.csv to HR_comma_sep.csv

In [4]:
```
df = pd.read_csv('HR_comma_sep.csv')
df
```

Out[4]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_l
0	0.38	0.53	2	157	3	0	1	
1	0.80	0.86	5	262	6	0	1	
2	0.11	0.88	7	272	4	0	1	
3	0.72	0.87	5	223	5	0	1	
4	0.37	0.52	2	159	3	0	1	
...	...	...	...	...	...	...	...	
14994	0.40	0.57	2	151	3	0	1	
14995	0.37	0.48	2	160	3	0	1	
14996	0.37	0.53	2	143	3	0	1	
14997	0.11	0.96	6	280	4	0	1	
14998	0.37	0.52	2	158	3	0	1	

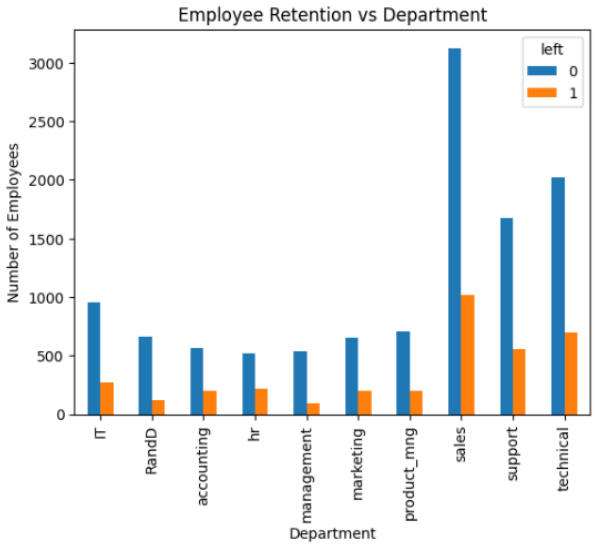
14999 rows × 10 columns

In [5]:
```
print(df.isnull().sum())
print(df.groupby('left').mean(numeric_only=True))
print(df.groupby('salary').mean(numeric_only=True))
```

```
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                     0
promotion_last_5years    0
Department               0
salary                   0
dtype: int64
      satisfaction_level  last_evaluation  number_project  \
left
0               0.666810         0.715473        3.786664
1               0.440098         0.718113        3.855503

      average_montly_hours  time_spend_company  Work_accident  \
left
0               199.060203            3.380032       0.175009
1               207.419210            3.876505       0.047326

      promotion_last_5years
left
0                  0.026251
1                  0.005321
        satisfaction_level  last_evaluation  number_project  \
salary
high              0.637470         0.704325        3.767179
low               0.600753         0.717017        3.799891
medium            0.621817         0.717322        3.813528

        average_montly_hours  time_spend_company  Work_accident      left  \
salary
high              199.867421            3.692805       0.155214  0.066289
low               200.996583            3.438218       0.142154  0.296884
medium            201.338349            3.529010       0.145361  0.204313

        promotion_last_5years
salary
high                 0.058205
low                  0.009021
medium               0.028079
```

In [6]:
```
import matplotlib.pyplot as plt
pd.crosstab(df.salary,df.left).plot(kind='bar')
plt.title('Employee Retention vs Salary')
plt.xlabel('Salary')
plt.ylabel('Number of Employees')
plt.show()
```

In [7]:
```
pd.crosstab(df.Department,df.left).plot(kind='bar')
plt.title('Employee Retention vs Department')
plt.xlabel('Department')
plt.ylabel('Number of Employees')
plt.show()
```



In [8]:
```
salary_dummies = pd.get_dummies(df.salary, prefix="salary")
dept_dummies = pd.get_dummies(df.Department, prefix="dept")
```

In [9]:
```
df_with_dummies = pd.concat([df, salary_dummies, dept_dummies], axis=1)
df_with_dummies = df_with_dummies.drop(['salary', 'Department'], axis=1)
```

In [10]:
```
X_features = ['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours', 'time_spend_company', 'Wo
X = df_with_dummies[X_features]
y = df_with_dummies.left
```

In [11]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

In [12]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[12]:
```
LogisticRegression()
```
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 [13]:
```
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test)
```

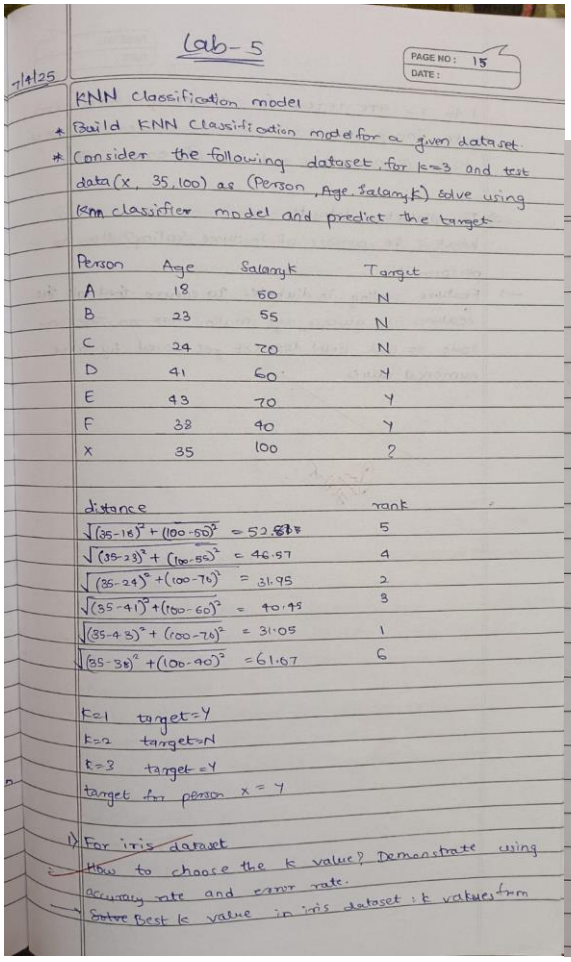
In [14]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the model:", accuracy)
```

```
Accuracy of the model: 0.7924444444444444
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot:



Lab-5

7/4/25

PAGE NO : 15
DATE :

KNN Classification model

* Build KNN Classification model for a given dataset
* Consider the following dataset, for k=3 and test data(x, 35, 100) as (Person, Age, Salary, k) solve using knn classifier model and predict the target

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

distance	rank
$\sqrt{(35-18)^2+(100-50)^2}$ = 52.88	5
$\sqrt{(35-23)^2+(100-55)^2}$ = 46.57	4
$\sqrt{(35-24)^2+(100-70)^2}$ = 31.95	2
$\sqrt{(35-41)^2+(100-60)^2}$ = 40.45	3
$\sqrt{(35-43)^2+(100-70)^2}$ = 31.05	1
$\sqrt{(35-38)^2+(100-40)^2}$ = 61.07	6

k=1 target=Y
k=2 target=N
k=3 target=Y
target for person x = Y

1) For iris dataset
How to choose the k value? Demonstrate using accuracy rate and error rate.
→ Solve Best k value in iris dataset : k values from 1 to 20 are tested for accuracy and error rate. The k value of 3 was choosen because of high ① accuracy and low ② error rate.

PAGE NO :
DATE :

2) For diabetes dataset
What is the purpose of feature scaling? How to perform it?
→ Feature scaling in diabetes to ensure that all the features like glucose, age, insulin, are on the same scale so that KNN does not get biased by large numerical values

7/4/25

Code with output:

In [ ]:
```
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
```

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

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) (2).csv
```

In [ ]:
```
df = pd.read_csv('iris (1).csv')
df
```

Out[ ]:

	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

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

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

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

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

Out[ ]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19, 20]})
```

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

```
Best k value: 3
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
```

Out[ ]:
```
KNeighborsClassifier(n_neighbors=3)
```

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 [ ]:
```
y_pred = knn.predict(X_test)
```

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

```
Accuracy Score: 1.00
```

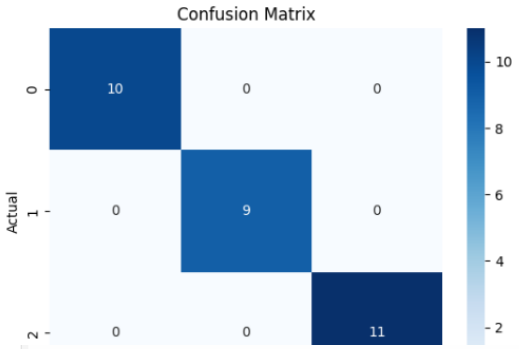
In [ ]:
```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

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

In [ ]:
```
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
```

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




```
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
```

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving diabetes.csv to diabetes.csv
```

In [ ]:
```
df = pd.read_csv('diabetes.csv')
```

In [ ]:
```
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

In [ ]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

Out[ ]:
```
KNeighborsClassifier()
```

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 [ ]:
```
y_pred = knn.predict(X_test)
```

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

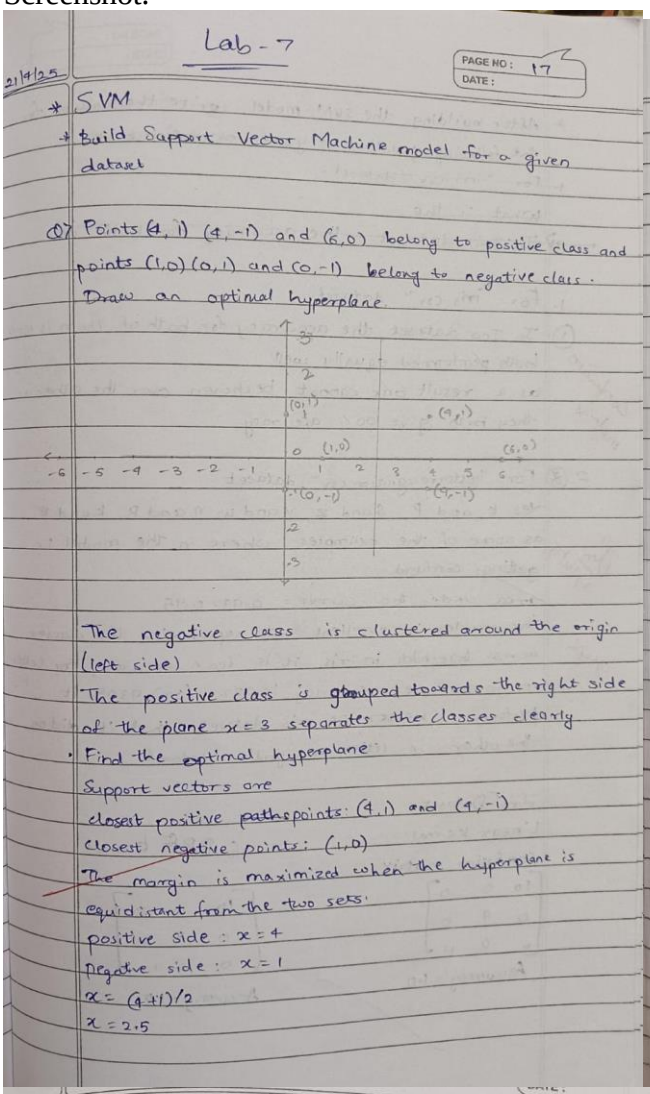
In [ ]:
```
print("Accuracy Score:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
```

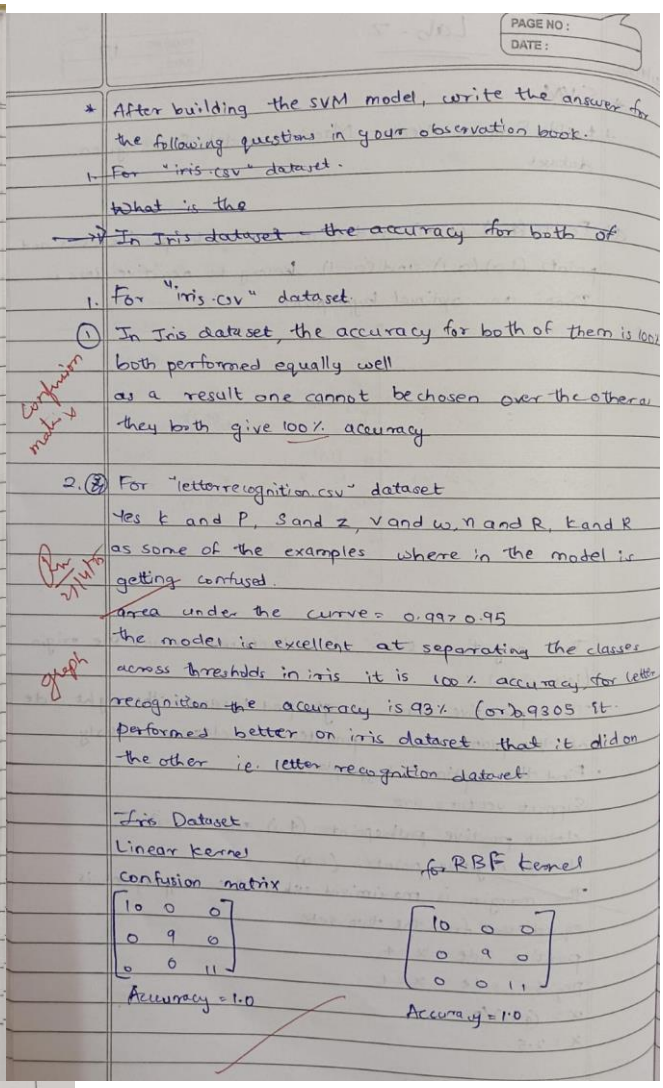
```
Accuracy Score: 0.6883116883116883
Confusion Matrix:
 [[79 20]
 [28 27]]
```

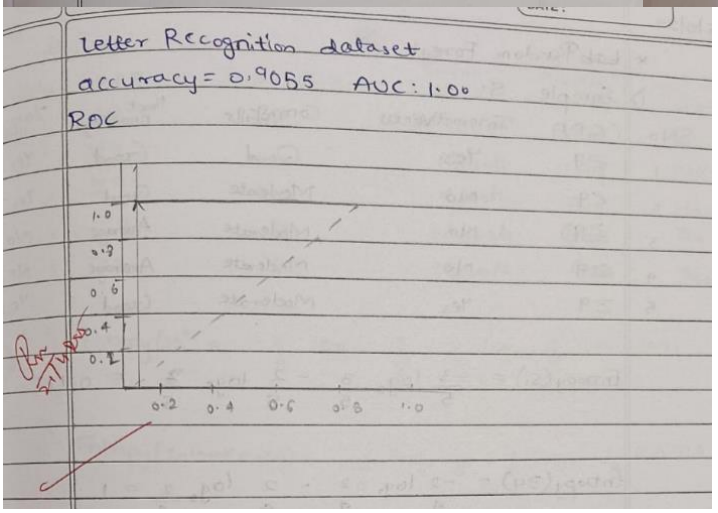
Program 7

Build Support vector machine model for a given dataset

Screenshot:







Code with output:

1. SVM Classifier for IRIS Dataset (RBF and Linear Kernels)

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

In [ ]:
```
from google.colab import files
uploaded=files.upload()
```

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 (1).csv
```

In [ ]:
```
df=pd.read_csv('iris.csv')
df
```

Out[ ]:

	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

In [ ]:
```
X_iris = df.iloc[:, :-1]
y_iris = df.iloc[:, -1]
```

In [ ]:
```
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)
```

In [ ]:
```
model_linear = SVC(kernel='linear')
model_linear.fit(X_train_iris, y_train_iris)
y_pred_linear = model_linear.predict(X_test_iris)
```

In [ ]:
```
print("Linear Kernel:")
print("Accuracy:", accuracy_score(y_test_iris, y_pred_linear))
print("Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_linear))
```

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

In [ ]:
```
model_rbf = SVC(kernel='rbf')
model_rbf.fit(X_train_iris, y_train_iris)
y_pred_rbf = model_rbf.predict(X_test_iris)
```

In [ ]:
```
print("\nRBF Kernel:")
print("Accuracy:", accuracy_score(y_test_iris, y_pred_rbf))
print("Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_rbf))
```

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

2. SVM Classifier for Letter Recognition Dataset (ROC + AUC)

In [ ]:
```
from sklearn.preprocessing import LabelEncoder, label_binarize
from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import StandardScaler
import numpy as np
```

In [ ]:
```
from google.colab import files
uploaded=files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

In [ ]:
```
drug=pd.read_csv('letter-recognition.csv')
drug
```

Out[ ]:

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19995	D	2	2	3	3	2	7	7	7	6	6	6	4	2	8	3	7
19996	C	7	10	8	8	4	4	8	6	9	12	9	13	2	9	3	7
19997	T	6	9	6	7	5	6	11	3	7	11	9	5	2	12	2	4
19998	S	2	3	4	2	1	8	7	2	6	10	6	8	1	9	5	8
19999	A	4	9	6	6	2	9	5	3	1	8	1	8	2	7	2	8

20000 rows × 17 columns

In [ ]:
```
X_letter = drug.iloc[:, 1:]
y_letter = drug.iloc[:, 0]
```

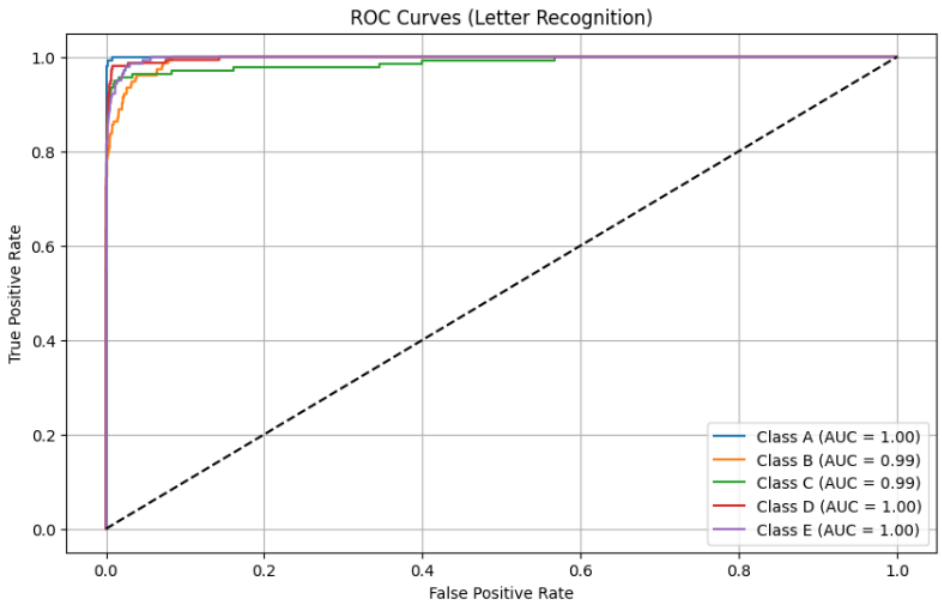
In [ ]:
```
label_encoder = LabelEncoder()
y_letter_encoded = label_encoder.fit_transform(y_letter)
```

In [ ]:
```
y_letter_bin = label_binarize(y_letter_encoded, classes=np.unique(y_letter_encoded))
n_classes = y_letter_bin.shape[1]
```

In [ ]:
```
X_train_let, X_test_let, y_train_let, y_test_let = train_test_split(X_letter, y_letter_bin, test_size=0.2, random_state=42
```

In [ ]:
```
scaler = StandardScaler()
X_train_let = scaler.fit_transform(X_train_let)
X_test_let = scaler.transform(X_test_let)
```

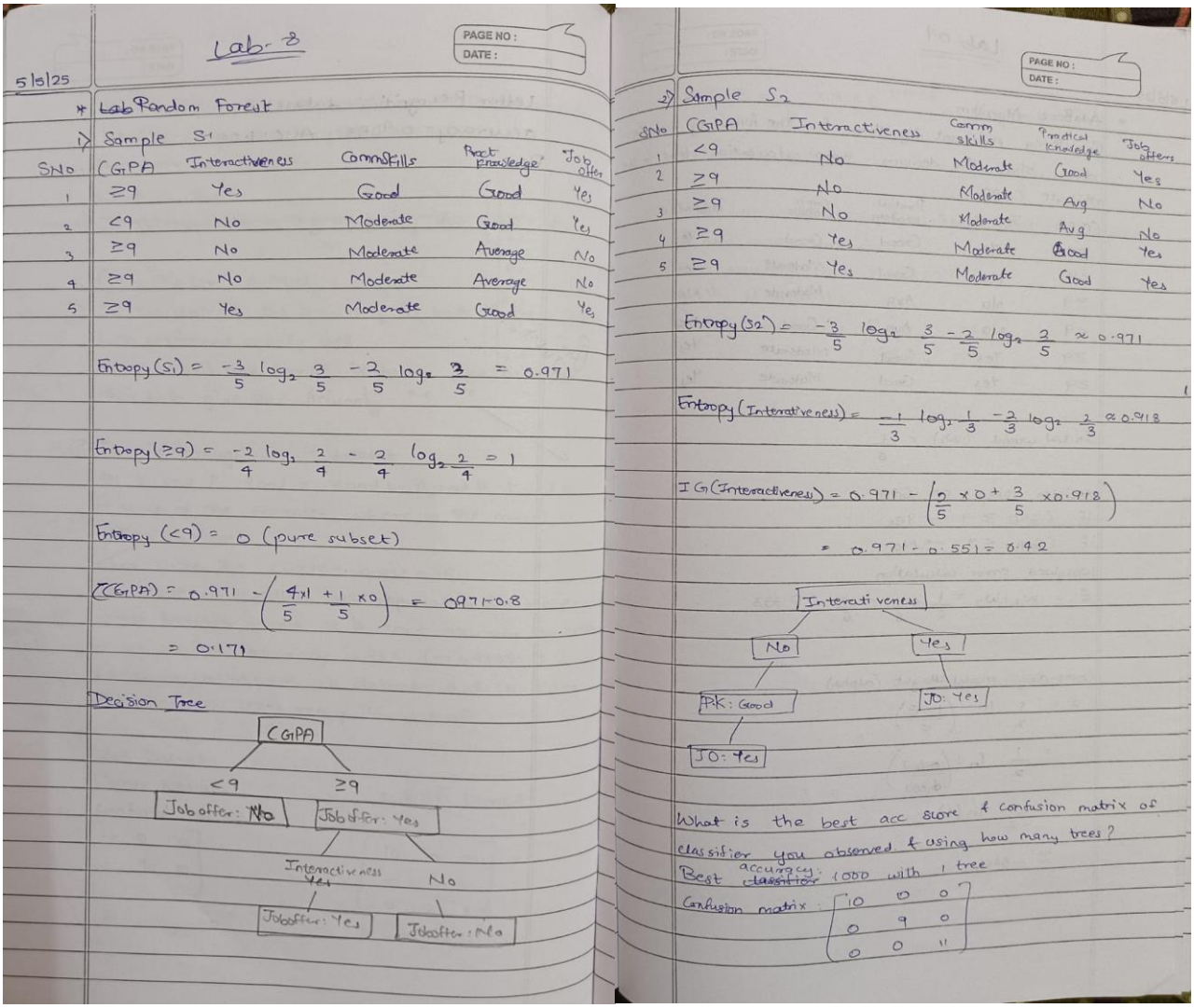
In [ ]:
```
plt.figure(figsize=(10, 6))
for i in range(min(5, n_classes)):  # Plotting only first 5 classes for clarity
    plt.plot(fpr[i], tpr[i], label=f'Class {label_encoder.inverse_transform([i])[0]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curves (Letter Recognition)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot:



Code with output:

In [1]:
```
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
```

In [2]:
```
from google.colab import files
uploaded = files.upload()
```

```
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
```

In [3]:
```
df = pd.read_csv('iris (1).csv')
df
```

Out[3]:

	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

In [4]:
```
X = df.drop(columns=['species'])
y = df['species']
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [6]:
```
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
```

In [7]:
```
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default RF accuracy (n_estimators=10): {default_score:.4f}")
```

```
Default RF accuracy (n_estimators=10): 1.0000
```

In [8]:
```
scores = []
n_range = range(1, 101)
```

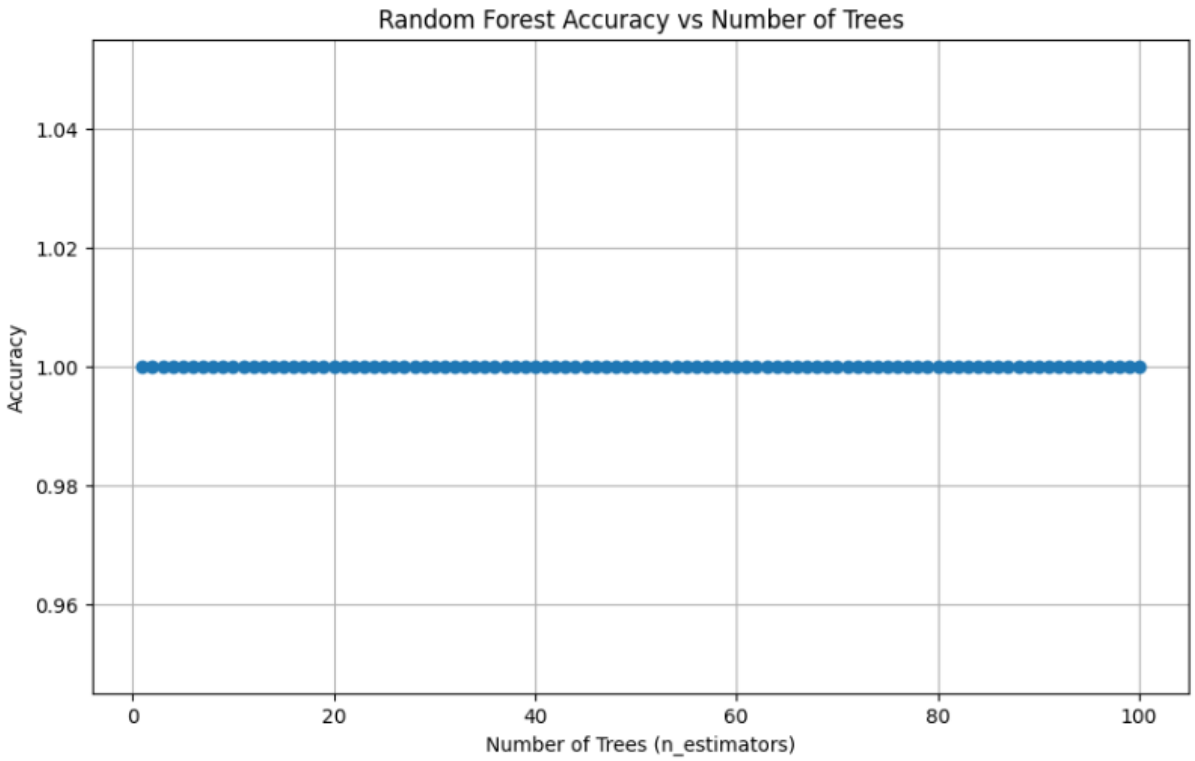
In [9]:
```
for n in n_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
```

In [10]:
```
best_score = max(scores)
best_n = n_range[scores.index(best_score)]
print(f"Best RF accuracy: {best_score:.4f} with n_estimators={best_n}")
```

```
Best RF accuracy: 1.0000 with n_estimators=1
```

In [11]:

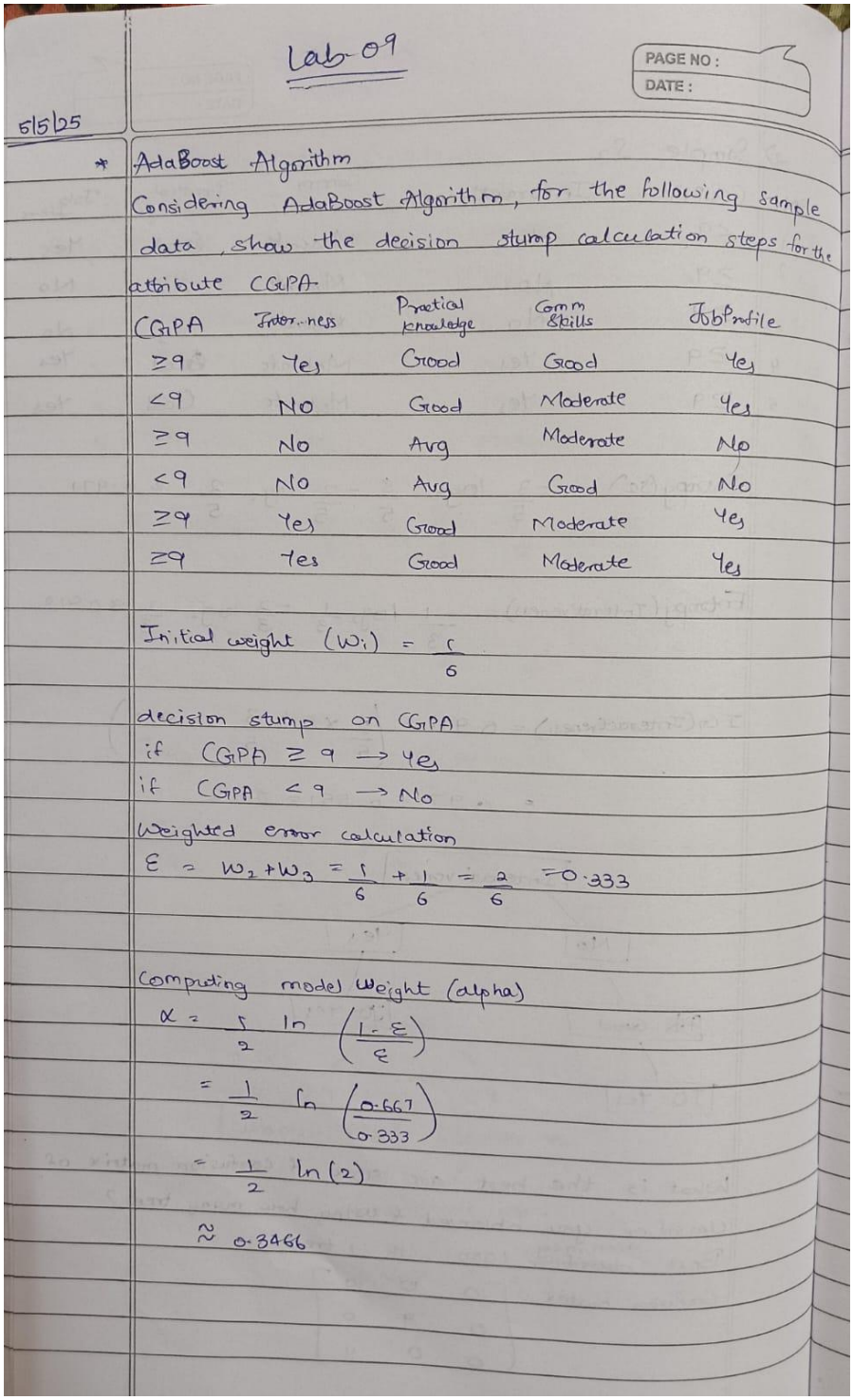
```
plt.figure(figsize=(10, 6))
plt.plot(n_range, scores, marker='o')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot:



Lab 09

5/5/25

PAGE NO :
DATE :

* AdaBoost Algorithm

Considering AdaBoost Algorithm, for the following sample data, show the decision stump calculation steps for the attribute CGPA

CGPA	Inter..ness	Practical knowledge	Comm Skills	JobProfile
≥9	Yes	Good	Good	Yes
<9	No	Good	Moderate	Yes
≥9	No	Avg	Moderate	No
<9	No	Avg	Good	No
≥9	Yes	Good	Moderate	Yes
≥9	Yes	Good	Moderate	Yes

Initial weight $(w_i) = \frac{1}{6}$

decision stump on CGPA

if CGPA ≥ 9 → Yes

if CGPA < 9 → No

Weighted error calculation

$\varepsilon = w_2 + w_3 = \frac{1}{6} + \frac{1}{6} = \frac{2}{6} = 0.333$

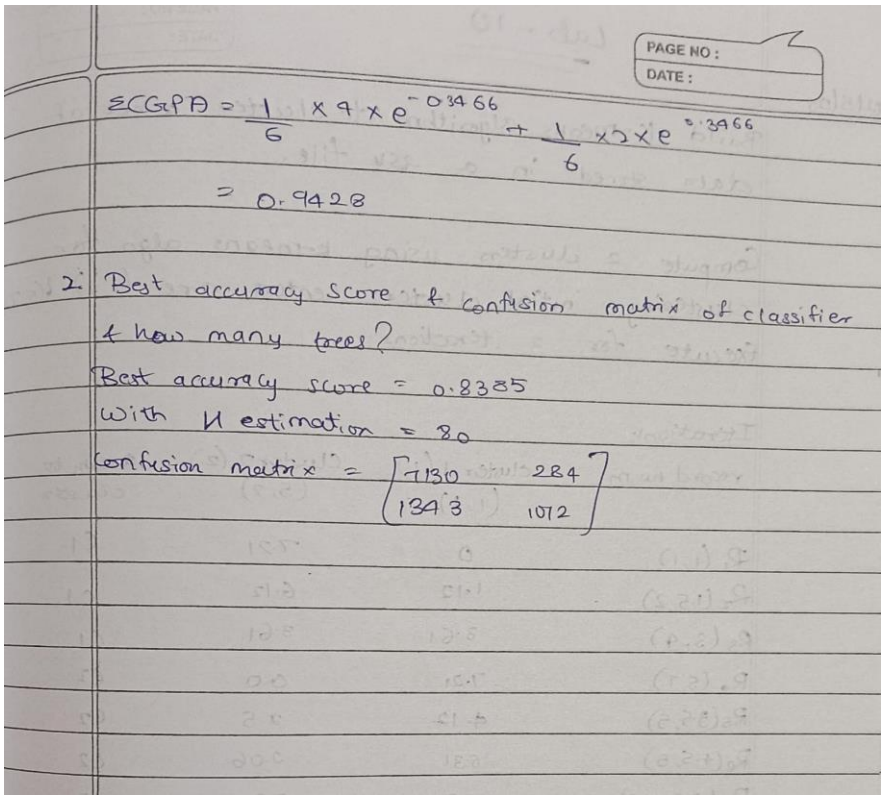
Computing model weight (alpha)

$\alpha = \frac{1}{2} \ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$

$= \frac{1}{2} \ln\left(\frac{0.667}{0.333}\right)$

$= \frac{1}{2} \ln(2)$

≈ 0.3466



Code with output:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
from google.colab import files
uploaded = files.upload()
```

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
```

In [3]:
```
df = pd.read_csv('income.csv')
df
```

Out[3]:

	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

In [4]:
```
df.dropna(inplace=True)
```

In [5]:
```
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

In [6]:
```
X = df.drop(columns=['income_level'], errors='ignore', axis=1)
y = df['income_level']
```

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [8]:
```
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}")
```

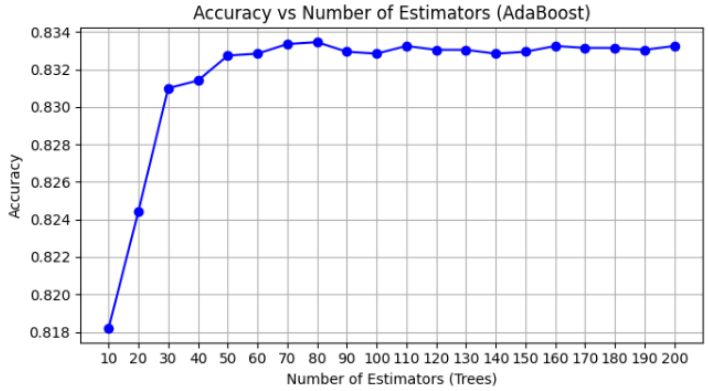
```
Accuracy with 10 estimators: 0.8182
```

In [9]:
```
best_score = 0
best_n = 0
estimators_range = list(range(10, 201, 10))
scores = []
```

In [10]:
```
for n in estimators_range:
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
    print(f"n_estimators={n}, Accuracy={score:.4f}")
    if score > best_score:
        best_score = score
        best_n = n

print(f"\nBest Accuracy: {best_score:.4f} using {best_n} estimators")
```

```
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
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
n_estimators=110, Accuracy=0.8332
n_estimators=120, Accuracy=0.8330
n_estimators=130, Accuracy=0.8330
n_estimators=140, Accuracy=0.8328
n_estimators=150, Accuracy=0.8329
n_estimators=160, Accuracy=0.8332
n_estimators=170, Accuracy=0.8331
n_estimators=180, Accuracy=0.8331
n_estimators=190, Accuracy=0.8330
n_estimators=200, Accuracy=0.8332

Best Accuracy: 0.8335 using 80 estimators
```

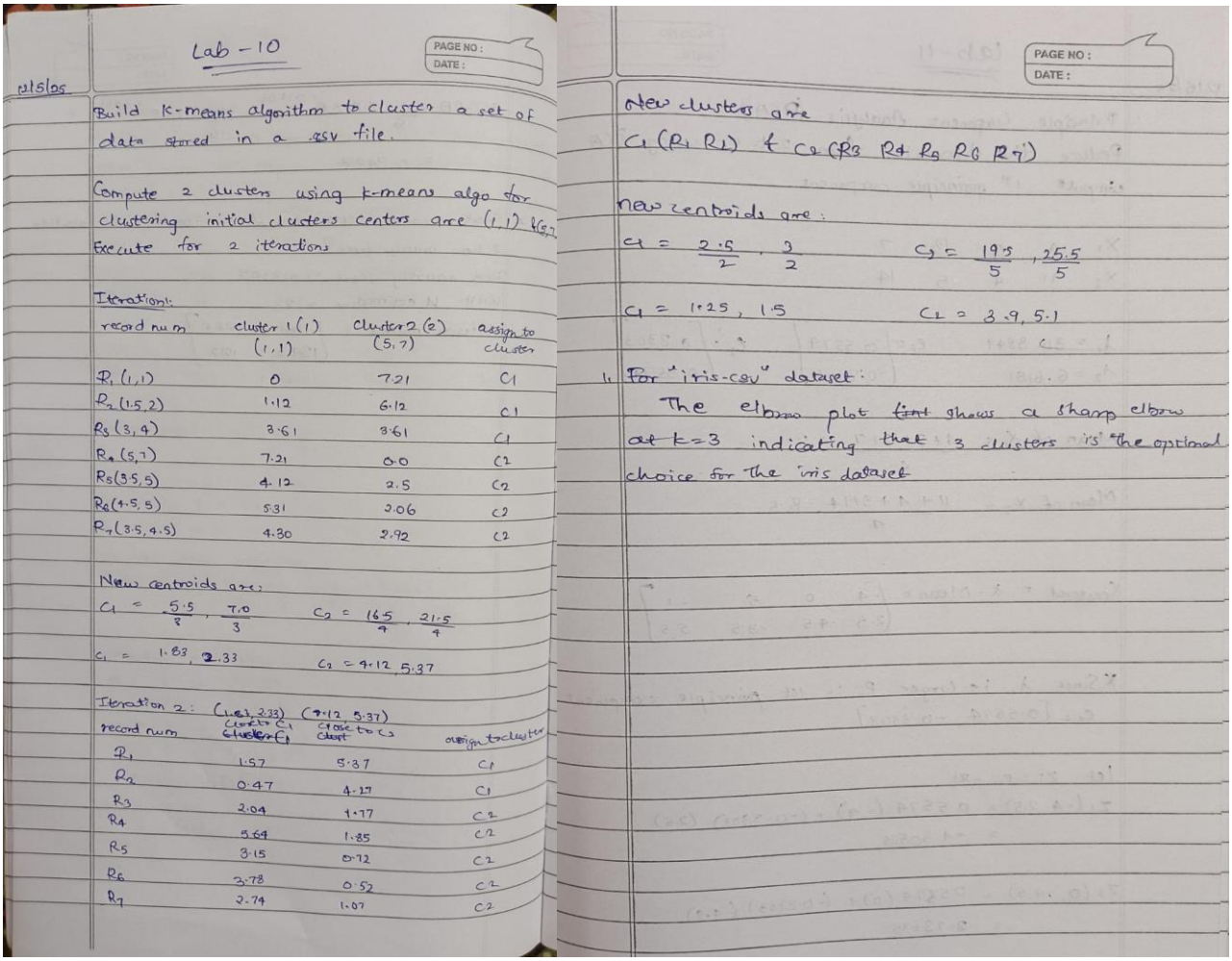
In [11]:
```
plt.figure(figsize=(7, 4))
plt.plot(estimators_range, scores, marker='o', linestyle='-', color='blue')
plt.title("Accuracy vs Number of Estimators (AdaBoost)")
plt.xlabel("Number of Estimators (Trees)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.xticks(estimators_range)
plt.tight_layout()
plt.show()
```



Program 10

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

Screenshot:



Lab - 10

PAGE NO :
DATE :

11/5/25

Build k-means algorithm to cluster a set of data stored in a csv file.

Compute 2 clusters using k-means algo for clustering initial clusters centers are (1,1) & (5,7) Execute for 2 iterations

Iteration1:

record num	cluster 1 (1) (1,1)	cluster 2 (2) (5,7)	assign to cluster
P_1 (1,1)	0	7.21	C1
P_2 (1.5,2)	1.12	6.12	C1
P_3 (3,4)	3.61	3.61	C1
P_4 (5,7)	7.21	0.0	C2
P_5 (3.5,5)	4.12	2.5	C2
P_6 (4.5,5)	5.31	2.06	C2
P_7 (3.5,4.5)	4.30	2.92	C2

New centroids are:

$C_1 = \frac{5.5}{3}, \frac{7.0}{3}$ $C_2 = \frac{16.5}{4}, \frac{21.5}{4}$

$C_1 = 1.83, 2.33$ $C_2 = 4.12, 5.37$

Iteration 2: (1.83, 2.33) (4.12, 5.37)

record num	cluster C1	cluster C2	assign to cluster
P_1	1.57	5.37	C1
P_2	0.47	4.27	C1
P_3	2.04	1.77	C2
P_4	5.64	1.85	C2
P_5	3.15	0.72	C2
P_6	3.78	0.52	C2
P_7	2.74	1.07	C2

PAGE NO :
DATE :

New clusters are

C_1 (P_1 P_2) & C_2 (P_3 P_4 P_5 P_6 P_7)

New centroids are:

$C_1 = \frac{2.5}{2}, \frac{2}{2}$ $C_2 = \frac{19.5}{5}, \frac{25.5}{5}$

$C_1 = 1.25, 1.5$ $C_2 = 3.9, 5.1$

4. For "iris-csv" dataset:

The elbow plot ~~first~~ shows a sharp elbow at k=3 indicating that 3 clusters is the optimal choice for the iris dataset

Code with output:

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from scipy.stats import mode
import matplotlib.pyplot as plt
```

In [6]:
```
np.random.seed(42)
names = [f"Person_{i}" for i in range(50)]
ages = np.random.randint(20, 60, 50)
income = np.random.randint(30000, 120000, 50)
```

In [7]:
```
df = pd.DataFrame({'Name': names, 'Age': ages, 'Income': income})
df.to_csv("income.csv", index=False)
```

In [8]:
```
data = pd.read_csv("income.csv")
```

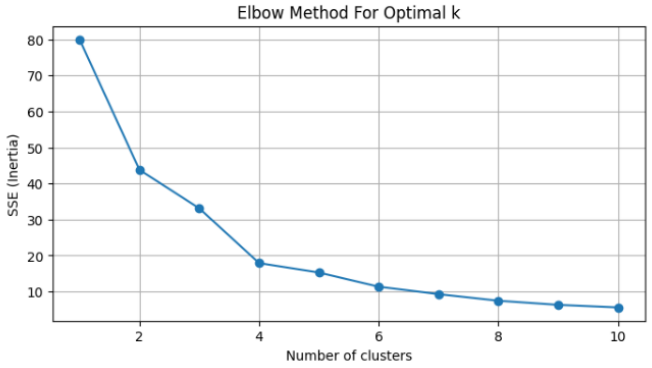
In [9]:
```
X = data[['Age', 'Income']]
```

In [10]:
```
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)
```

In [11]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [12]:
```
sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)
    sse.append(kmeans.inertia_)
```

In [13]:
```
plt.figure(figsize=(8, 4))
plt.plot(k_range, sse, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('SSE (Inertia)')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.show()
```



In [14]:
```
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train_scaled)
```

Out[14]:
```
KMeans(n_clusters=3, random_state=42)
```
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 [15]:
```
predictions = kmeans.predict(X_test_scaled)
```

In [16]:
```
full_kmeans = KMeans(n_clusters=optimal_k, random_state=42)
true_clusters = full_kmeans.fit_predict(scaler.fit_transform(X))
```

In [17]:

```
def map_clusters(true_labels, pred_labels):
    labels = np.zeros_like(pred_labels)
    for i in range(optimal_k):
        mask = (pred_labels == i)
        if np.sum(mask) == 0:
            continue
        labels[mask] = mode(true_labels[mask])[0]
    return labels
```

In [19]:

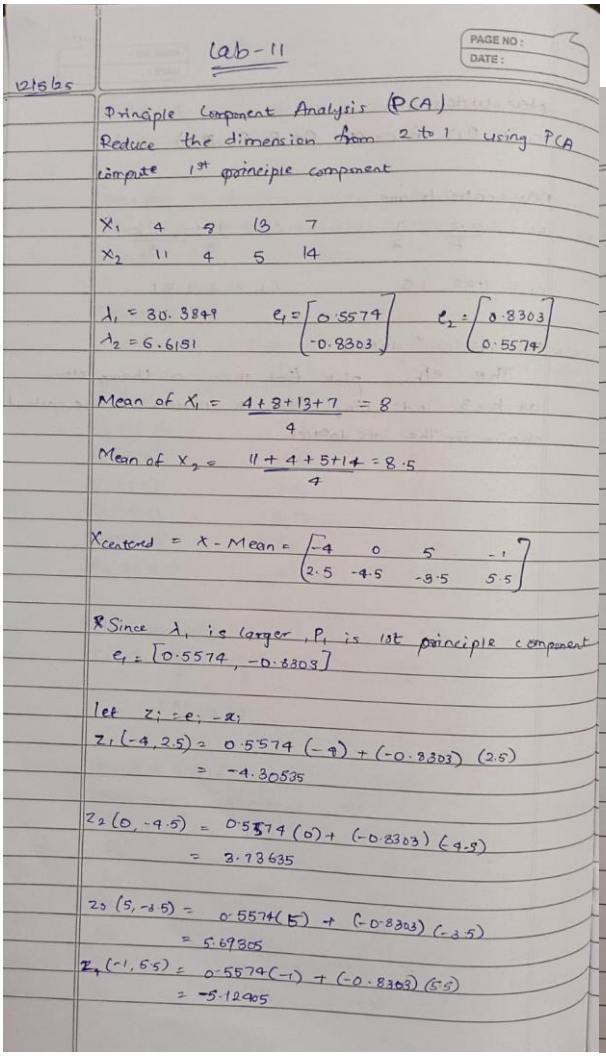
```
mapped_preds = map_clusters(true_clusters[X_test.index], predictions)
accuracy = accuracy_score(true_clusters[X_test.index], mapped_preds)
print(f"Clustering Accuracy: {accuracy:.2f}")
```

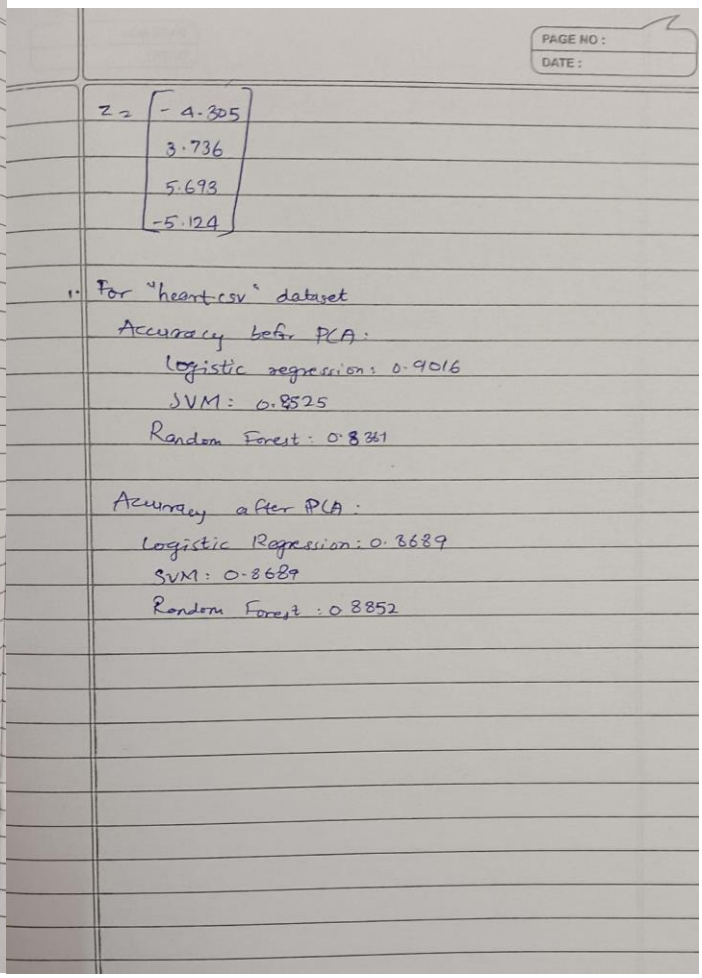
```
Clustering Accuracy: 0.70
```

Program 11

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

Screenshot:





Code with output:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [2]:
```
digits = load_digits()
X = digits.data
y = digits.target
```

In [3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [4]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [5]:
```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

In [6]:
```
logreg = LogisticRegression(max_iter=10000)
logreg.fit(X_train_pca, y_train)
```

Out[6]:
```
LogisticRegression(max_iter=10000)
```
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 [7]:
```
y_pred = logreg.predict(X_test_pca)
```

In [8]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy using PCA with 2 components: {accuracy:.4f}')
```

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

In [9]:
```
import pandas as pd
import numpy as np
from scipy import stats
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
```

In [10]:
```
from google.colab import files
uploaded = files.upload()
```

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
```

In [11]:
```
df = pd.read_csv('heart.csv')
df
```

Out[11]:

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

918 rows × 12 columns

In [12]:
```
numeric_cols = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
z_scores = stats.zscore(df[numeric_cols])
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
df_clean = df[filtered_entries].reset_index(drop=True)
```

In [13]:
```
categorical_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
numeric_cols = ['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak']
```

In [14]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(drop='first', sparse_output=False), categorical_cols),
        ('num', StandardScaler(), numeric_cols)
    ])
```

In [15]:
```
X = df_clean.drop('HeartDisease', axis=1)
y = df_clean['HeartDisease']
X_processed = preprocessor.fit_transform(X)
```

In [16]:
```
cat_encoded_cols = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_cols)
feature_names = list(cat_encoded_cols) + numeric_cols
```

In [17]:
```
X_processed_df = pd.DataFrame(X_processed, columns=feature_names)
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X_processed_df, y, test_size=0.2, random_state=42)
```

In [19]:
```
models = {
    'SVM': SVC(kernel='rbf', random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42)
}
```

In [20]:
```
accuracies = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies[name] = accuracy
    print(f"{name} Accuracy: {accuracy:.4f}")
```

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

In [21]:
```
best_model_name = max(accuracies, key=accuracies.get)
print(f"\nBest Model: {best_model_name} with Accuracy: {accuracies[best_model_name]:.4f}")
```

```
Best Model: SVM with Accuracy: 0.8889
```

In [22]:
```
pca = PCA()
pca.fit(X_processed_df)
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
n_components = np.argmax(cumulative_variance >= 0.95) + 1
print(f"\nNumber of PCA components to retain 95% variance: {n_components}")
```

```
Number of PCA components to retain 95% variance: 10
```

In [23]:
```
best_model = models[best_model_name]
pipeline = Pipeline([
    ('pca', PCA(n_components=n_components)),
    ('classifier', best_model)
])
```

In [24]:
```
pipeline.fit(X_train, y_train)
y_pred_pca = pipeline.predict(X_test)
accuracy_pca = accuracy_score(y_test, y_pred_pca)
print(f"{best_model_name} Accuracy with PCA: {accuracy_pca:.4f}")
print(f"Accuracy Change with钽Impact: {accuracy_pca - accuracies[best_model_name]:.4f}")
```

```
SVM Accuracy with PCA: 0.8944
Accuracy Change with钽Impact: 0.0056
```

In [25]:
```
X_processed_df['HeartDisease'] = y
X_processed_df.to_csv('processed_heart.csv', index=False)
```

In [27]:
```
df = pd.read_csv('processed_heart.csv')
df
```

Out[27]:

	Sex_M	ChestPainType_ATA	ChestPainType_NAP	ChestPainType_TA	RestingECG_Normal	RestingECG_ST	ExerciseAngina_Y	ST_Slope
0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	
1	0.0	0.0	1.0	0.0	1.0	0.0	0.0	
2	1.0	1.0	0.0	0.0	0.0	1.0	0.0	
3	0.0	0.0	0.0	0.0	1.0	0.0	1.0	
4	1.0	0.0	1.0	0.0	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	
894	1.0	0.0	0.0	1.0	1.0	0.0	0.0	