VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Sparsha Srinath Kadaba (1BM22CS287)

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
Feb-2025 to June-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)" has been carried out by **Sparsha Srinath Kadaba (1BM22CS287),** 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.

Spoorthi D M Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

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

Program 1

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

Code:

```
import pandas as pd
# Method-1: Initializing values directly into DataFrame
data_method1 = {'USN': ['1JS17CS001', '1JS17CS002', '1JS17CS003',
'1JS17CS004', '1JS17CS005'],
'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
'Marks': [90, 85, 92, 78, 88]}
df_method1 = pd.DataFrame(data_method1)
print("Method-1:")
print(df_method1)
print("-" * 20)
# Method-2: Importing datasets from sklearn.datasets
from sklearn.datasets import load_diabetes
diabetes_data = load_diabetes()
df_method2 = pd.DataFrame(data=diabetes_data.data,
columns=diabetes_data.feature_names)
df_method2['target'] = diabetes_data.target
print("Method-2:")
print(df_method2.head())
print("-" * 20)
# Method-3: Importing datasets from a specific .csv file
try:
```

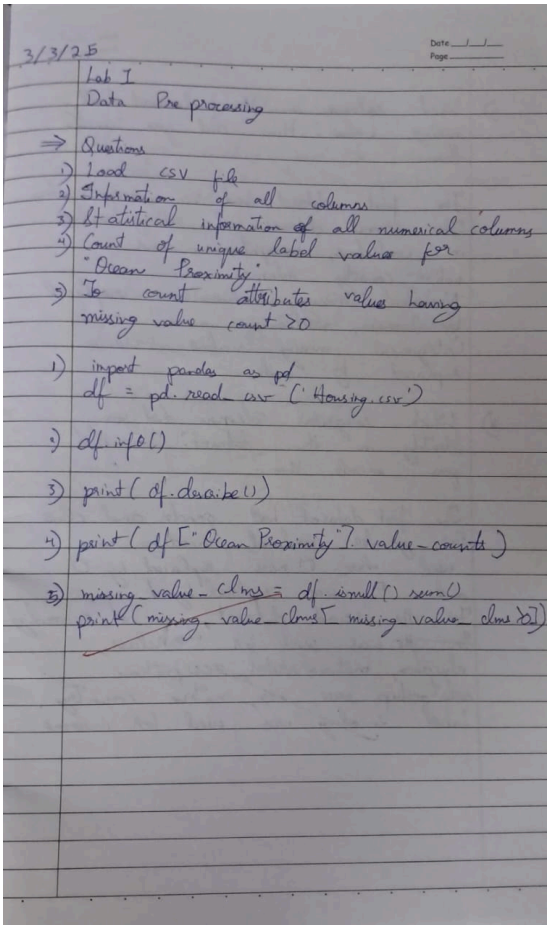
```
df_method3 = pd.read_csv('sample_sales_data.csv')
print("Method-3:")
print(df_method3.head())
print("-" * 20)
except FileNotFoundError:
print("sample_sales_data.csv not found. Please upload the file.")
print("-" * 20)
import yfinance as yf
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
start_date = "2024-01-01"
end_date = "2024-12-30"
data = yf.download(tickers, start=start_date, end=end_date)
closing_prices = data['Close']
daily_returns = closing_prices.pct_change().dropna()
plt.figure(figsize=(12, 6))
closing_prices.plot()
plt.title('Closing Prices (2024)')
plt.xlabel('Date')
plt.ylabel('Price (INR)')
plt.grid(True)
plt.show()
plt.figure(figsize=(12, 6))
daily_returns.plot()
```

```
plt.title('Daily Returns (2024)')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.grid(True)
plt.show()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset.

Screenshot:



3/3/25

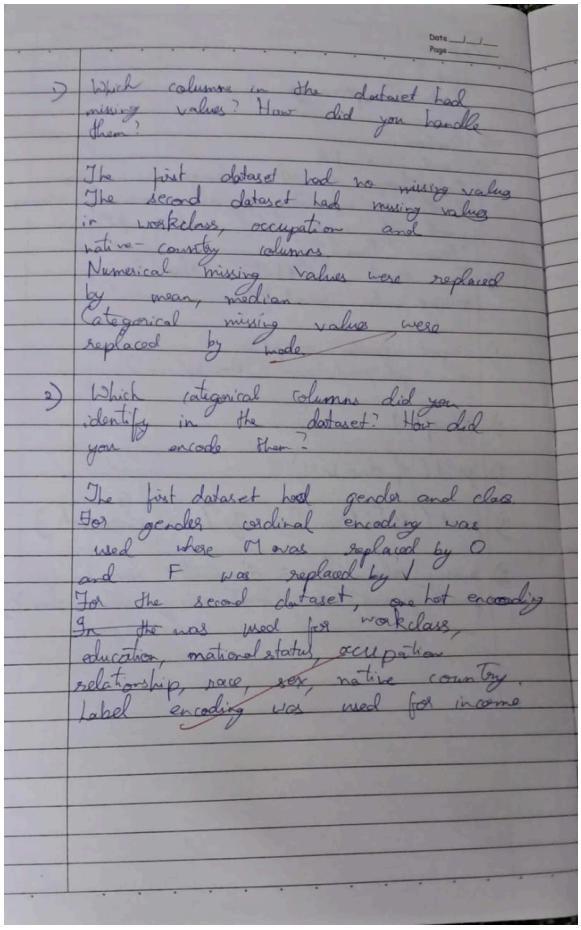
Lab 1

Data Pre processing

⇒ Questions

1) Load csv file
2) Information of all columns
3) Statistical information of all numerical columns
4) Count of unique label value for "Ocean Proximity"
5) To count attributes value having missing value count >0

1) import pandas as pd
   df = pd.read_csv('Housing.csv')
2) df.info()
3) print(df.describe())
4) print(df["Ocean Proximity"].value_counts())
5) missing_value_clms = df.isnull().sum()
   print(missing_value_clms[missing_value_clms>0])

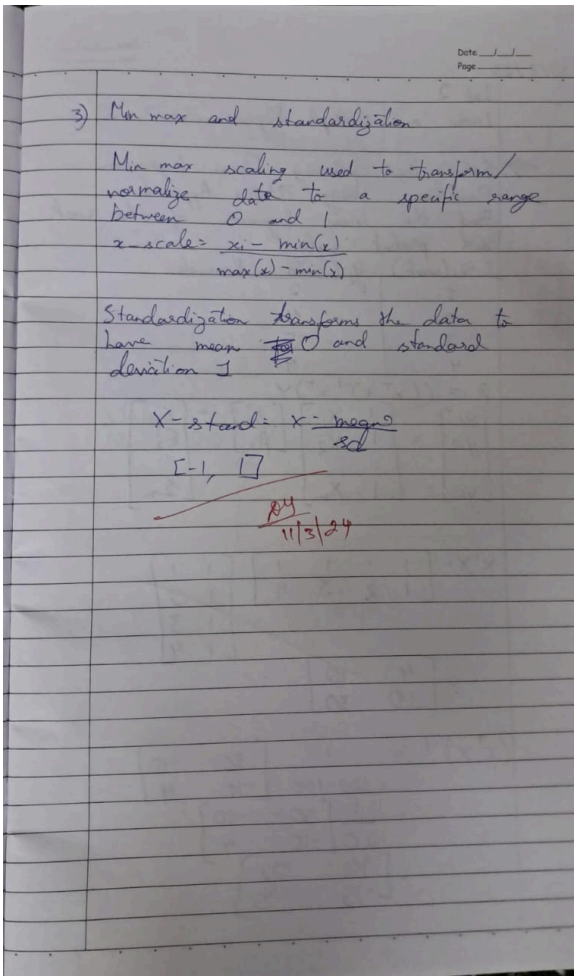


1) Which columns in the dataset had missing values? How did you handle them?

The first dataset had no missing value
The second dataset had missing value in workclass, occupation and native-country columns
Numerical missing values were replaced by mean, median.
Categorical missing value were replaced by mode

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

The first dataset had gender and class
For gender ordinal encoding was used where M was replaced by 0 and F was replaced by 1
For the second dataset, one hot encoding in the was used for workclass, education, matrimonial status, occupation relationship, race, sex, native country.
Label encoding was used for income



Code:

Lab 1 - 3/3/25

Data Preprocessing

In [9]:

```
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 adult.csv to adult.csv
```

In [3]:

```
import pandas as pd
data = pd.read_csv('Dataset of Diabetes .csv')
```

```
data.head()
print(data.isnull().sum())
data.head()

new_row = {col: float('nan') for col in data.columns}
data = pd.concat([data, pd.DataFrame([new_row])], ignore_index=True)
print(data.isnull().sum())
```

```
ID           0
No_Pation    0
Gender       0
AGE          0
Urea         0
Cr           0
HbA1c        0
Chol         0
TG           0
HDL          0
LDL          0
VLDL         0
BMI          0
CLASS        0
dtype: int64
ID           1
No_Pation    1
Gender       1
AGE          1
Urea         1
Cr           1
HbA1c        1
Chol         1
TG           1
HDL          1
LDL          1
VLDL         1
BMI          1
CLASS        1
dtype: int64
```

In [4]:

```
from sklearn.impute import SimpleImputer
imputer_median = SimpleImputer(strategy="median")
imputer_mean = SimpleImputer(strategy="mean")


data_copy = data.copy()

numerical_cols = data.select_dtypes(include=['number']).columns

for col in numerical_cols:

    if data_copy[col].isnull().any():
        print(f"Imputing missing values in column '{col}'")

        if col == "AGE":
```

```
            data_copy[col] = imputer_median.fit_transform(data_copy[[col]])
        else:
            data_copy[col] = imputer_mean.fit_transform(data_copy[[col]])

    print(f"Missing values in column '{col}': {data_copy[col].isnull().sum()}")

print(data_copy.head())
```

```
Imputing missing values in column 'ID'
Missing values in column 'ID': 0
Imputing missing values in column 'No_Pation'
Missing values in column 'No_Pation': 0
Imputing missing values in column 'AGE'
Missing values in column 'AGE': 0
Imputing missing values in column 'Urea'
Missing values in column 'Urea': 0
Imputing missing values in column 'Cr'
Missing values in column 'Cr': 0
Imputing missing values in column 'HbA1c'
Missing values in column 'HbA1c': 0
Imputing missing values in column 'Chol'
Missing values in column 'Chol': 0
Imputing missing values in column 'TG'
Missing values in column 'TG': 0
Imputing missing values in column 'HDL'
Missing values in column 'HDL': 0
Imputing missing values in column 'LDL'
Missing values in column 'LDL': 0
Imputing missing values in column 'VLDL'
Missing values in column 'VLDL': 0
Imputing missing values in column 'BMI'
Missing values in column 'BMI': 0
      ID  No_Pation Gender   AGE  Urea    Cr  HbA1c  Chol   TG  HDL  LDL  \
0  502.0    17975.0      F  50.0   4.7  46.0    4.9   4.2  0.9  2.4  1.4
1  735.0    34221.0      M  26.0   4.5  62.0    4.9   3.7  1.4  1.1  2.1
2  420.0    47975.0      F  50.0   4.7  46.0    4.9   4.2  0.9  2.4  1.4
3  680.0    87656.0      F  50.0   4.7  46.0    4.9   4.2  0.9  2.4  1.4
4  504.0    34223.0      M  33.0   7.1  46.0    4.9   4.9  1.0  0.8  2.0

   VLDL   BMI CLASS
0   0.5  24.0     N
1   0.6  23.0     N
2   0.5  24.0     N
3   0.5  24.0     N
4   0.4  21.0     N
```

In [5]:

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

df=pd.read_csv('Dataset of Diabetes .csv')
```

```
df_copy = df.copy()

df_copy["CLASS"] = df_copy["CLASS"].astype(str).str.strip()
df_copy["CLASS"] = df_copy["CLASS"].replace("", "Unknown")
df_copy["CLASS"].fillna("Unknown", inplace=True)

print("Unique values in CLASS column:", df_copy["CLASS"].unique())

onehot_encoder = OneHotEncoder(sparse_output=False, drop=None)
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("CLASS", axis=1, inplace=True)

df_encoded["Gender"] = df_encoded["Gender"].str.upper()
ordinal_encoder = OrdinalEncoder(categories=[["M", "F"]])
df_encoded["Gender_Encoded"] = ordinal_encoder.fit_transform(df_encoded[["Gender"]])
df_encoded.drop("Gender", axis=1, inplace=True)

output_file = "transformed_dataset.csv"
df_encoded.to_csv(output_file, index=False)

print(df_encoded.head())
print(f"Fixed dataset saved: {output_file}")

data=pd.read_csv('transformed_dataset.csv')
data.head(-50)
```

```
Unique values in CLASS column: ['N' 'P' 'Y']
    ID  No_Pation  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL   BMI  \
0  502      17975   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
1  735      34221   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6  23.0
2  420      47975   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
3  680      87656   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
4  504      34223   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4  21.0

   CLASS_N  CLASS_P  CLASS_Y  Gender_Encoded
0      1.0      0.0      0.0             1.0
1      1.0      0.0      0.0             0.0
2      1.0      0.0      0.0             1.0
3      1.0      0.0      0.0             1.0
4      1.0      0.0      0.0             0.0
Fixed dataset saved: transformed_dataset.csv
```

```
<ipython-input-5-700584fbba0d>:11: FutureWarning: A value is trying to be set on a
copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to
perform the operation inplace on the original object.


  df_copy["CLASS"].fillna("Unknown", inplace=True)
```

Out[5]:

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS_N	CLASS_P	CLASS_Y	Gender_Encoded
0	502	17975	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	1.0	0.0	0.0	1.0
1	735	34221	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	1.0	0.0	0.0	0.0
2	420	47975	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	1.0	0.0	0.0	1.0
3	680	87656	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	1.0	0.0	0.0	1.0
4	504	34223	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
945	120	565474	51	4.4	67	6.8	6.7	3.7	0.9	2.9	1.7	32.0	0.0	0.0	1.0	0.0
944	12	65756	62	4.8	52	11.8	3.7	0.	0.8	2.6	0.3	33.0	0.0	0.0	1.0	1.0

6	1						8								
947	122	345	60	3.3	59	7.6	3.5	1.0	1.3	1.5	0.7	30.0	0.0	0.0	1.0	0.0
948	123	5676	60	3.4	27	14.7	3.5	1.9	1.3	1.6	1.4	35.0	0.0	0.0	1.0	1.0
949	124	75634	61	4.3	70	9.1	6.6	2.9	1.1	4.3	1.3	29.0	0.0	0.0	1.0	0.0

950 rows × 16 columns

In [6]:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

df_encoded = pd.read_csv("transformed_dataset.csv")
numerical_cols = df_encoded.select_dtypes(include=['number']).columns
normalizer = MinMaxScaler()
df_encoded[numerical_cols] = normalizer.fit_transform(df_encoded[numerical_cols])
output_file = "normalized.csv"
df_encoded.to_csv(output_file, index=False)
print(df_encoded.head())
print(f"Normalized dataset saved: {output_file}")
```

```
         ID  No_Pation       AGE      Urea        Cr     HbA1c      Chol  \
0  0.627034   0.000237  0.508475  0.109375  0.050378  0.264901  0.407767
1  0.918648   0.000452  0.101695  0.104167  0.070529  0.264901  0.359223
2  0.524406   0.000634  0.508475  0.109375  0.050378  0.264901  0.407767
3  0.849812   0.001160  0.508475  0.109375  0.050378  0.264901  0.407767
4  0.629537   0.000452  0.220339  0.171875  0.050378  0.264901  0.475728

         TG       HDL       LDL      VLDL       BMI  CLASS_N  CLASS_P  \
0  0.044444  0.226804  0.114583  0.011461  0.173913      1.0      0.0
1  0.081481  0.092784  0.187500  0.014327  0.139130      1.0      0.0
2  0.044444  0.226804  0.114583  0.011461  0.173913      1.0      0.0
3  0.044444  0.226804  0.114583  0.011461  0.173913      1.0      0.0
4  0.051852  0.061856  0.177083  0.008596  0.069565      1.0      0.0

   CLASS_Y  Gender_Encoded
0      0.0             1.0
1      0.0             0.0
```

```
2      0.0             1.0
3      0.0             1.0
4      0.0             0.0
Normalized dataset saved: normalized.csv
```

In [7]:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

df_encoded = pd.read_csv("transformed_dataset.csv")
numerical_cols = df_encoded.select_dtypes(include=['number']).columns

scaler = StandardScaler()

df_encoded[numerical_cols] = scaler.fit_transform(df_encoded[numerical_cols])

output_file = "normalized2.csv"
df_encoded.to_csv(output_file, index=False)

print(df_encoded.head())

print(f"Standardized dataset saved: {output_file}")
```

```
        ID  No_Pation       AGE      Urea        Cr     HbA1c      Chol  \
0  0.672140  -0.074747 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
1  1.641852  -0.069940 -3.130017 -0.212954 -0.115804 -1.334983 -0.893730
2  0.330868  -0.065869 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
3  1.412950  -0.054126 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
4  0.680463  -0.069939 -2.334096  0.673299 -0.382672 -1.334983  0.028576

         TG       HDL       LDL      VLDL       BMI   CLASS_N   CLASS_P  \
0 -1.035084  1.810756 -1.085457 -0.369958 -1.124622  2.951057 -0.236572
1 -0.678063 -0.158692 -0.457398 -0.342649 -1.326239  2.951057 -0.236572
2 -1.035084  1.810756 -1.085457 -0.369958 -1.124622  2.951057 -0.236572
3 -1.035084  1.810756 -1.085457 -0.369958 -1.124622  2.951057 -0.236572
4 -0.963680 -0.613180 -0.547121 -0.397267 -1.729472  2.951057 -0.236572

    CLASS_Y  Gender_Encoded
0 -2.325996        1.139671
1 -2.325996       -0.877446
2 -2.325996        1.139671
3 -2.325996        1.139671
4 -2.325996       -0.877446
Standardized dataset saved: normalized2.csv
```

In [11]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, MinMaxScaler,
StandardScaler

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

```
for col in df.columns:
    if df[col].dtype == "object":
        df[col].fillna(df[col].mode()[0], inplace=True)
    else:
        df[col].fillna(df[col].median(), inplace=True)

categorical_cols = df.select_dtypes(include=["object"]).columns.tolist()

onehot_encoder = OneHotEncoder(sparse_output=False, drop="first")
encoded_data = onehot_encoder.fit_transform(df[categorical_cols])
encoded_df                          =                          pd.DataFrame(encoded_data,
columns=onehot_encoder.get_feature_names_out(categorical_cols))

df_encoded = pd.concat([df, encoded_df], axis=1)
df_encoded.drop(columns=categorical_cols, inplace=True)

def remove_outliers(df, threshold=1.5):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
        return  df[~((df  <  (Q1  -  threshold  *  IQR))  |  (df  >  (Q3  +  threshold  *
IQR))).any(axis=1)]

df_encoded = remove_outliers(df_encoded)

numerical_cols = df_encoded.select_dtypes(include=["number"]).columns

minmax_scaler = MinMaxScaler()
df_encoded[numerical_cols] = minmax_scaler.fit_transform(df_encoded[numerical_cols])

standard_scaler = StandardScaler()
df_standardized = df_encoded.copy()
df_standardized[numerical_cols]                                                    =
standard_scaler.fit_transform(df_encoded[numerical_cols])

output_file_minmax = "/content/adult2.csv"
df_encoded.to_csv(output_file_minmax, index=False)

output_file_standard = "/content/adult3.csv"
df_standardized.to_csv(output_file_standard, index=False)

print("Min-Max Scaled Data:")
print(df_encoded.head())

print("\nStandard Scaled Data:")
print(df_standardized.head())

print(f"\n Normalized dataset saved: {output_file_minmax}")
print(f" Standardized dataset saved: {output_file_standard}")
```

```
<ipython-input-11-a2ad29aa08bb>:11: FutureWarning: A value is trying to be set on a
copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-11-a2ad29aa08bb>:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(df[col].mode()[0], inplace=True)
<ipython-input-11-a2ad29aa08bb>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_encoded[numerical_cols] = minmax_scaler.fit_transform(df_encoded[numerical_cols])
```

```
Min-Max Scaled Data:
           age    fnlwgt  educational-num  capital-gain  capital-loss  \
13    0.714286  0.714004              1.0           0.0           0.0
35    0.839286  0.433967              1.0           0.0           0.0
132   0.035714  0.802886              1.0           0.0           0.0
266   0.767857  0.444192              1.0           0.0           0.0
1417  0.285714  0.896515              1.0           0.0           0.0

      hours-per-week  workclass_Federal-gov  workclass_Local-gov  \
13          0.000000                    0.0                  0.0
35          0.294118                    0.0                  0.0
132         0.294118                    0.0                  0.0
266         0.294118                    0.0                  0.0
1417        0.294118                    0.0                  0.0

      workclass_Never-worked  workclass_Private  ...  \
13                       0.0                0.0  ...
35                       0.0                0.0  ...
132                      0.0                0.0  ...
266                      0.0                0.0  ...
1417                     0.0                0.0  ...

      native-country_Puerto-Rico  native-country_Scotland  \
13                           0.0                      0.0
35                           0.0                      0.0
132                          0.0                      0.0
266                          0.0                      0.0
1417                         0.0                      0.0

      native-country_South  native-country_Taiwan  native-country_Thailand  \
```

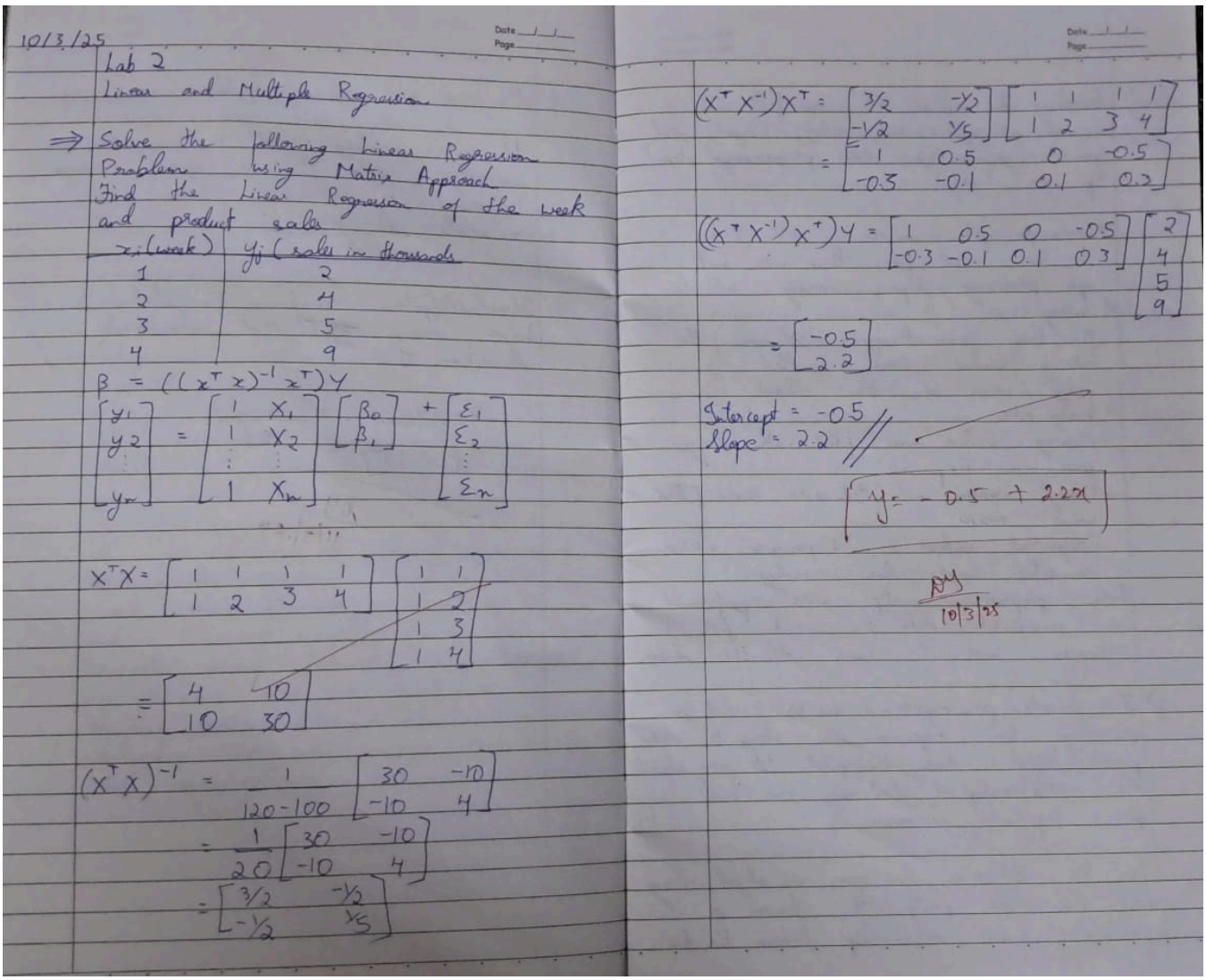
```
13                    0.0                    0.0                    0.0
35                    0.0                    0.0                    0.0
132                   0.0                    0.0                    0.0
266                   0.0                    0.0                    0.0
1417                  0.0                    0.0                    0.0

      native-country_Trinadad&Tobago  native-country_United-States  \
13                                0.0                           0.0
35                                0.0                           0.0
132                               0.0                           0.0
266                               0.0                           0.0
1417                              0.0                           0.0

      native-country_Vietnam  native-country_Yugoslavia  income_>50K
13                       0.0                        0.0          0.0
35                       0.0                        0.0          0.0
132                      0.0                        0.0          0.0
266                      0.0                        0.0          0.0
1417                     0.0                        0.0          0.0

[5 rows x 101 columns]

Standard Scaled Data:
           age    fnlwgt  educational-num  capital-gain  capital-loss  \
13    0.648319  1.387656         0.352089           0.0           0.0
35    1.085429  0.186624         0.352089           0.0           0.0
132  -1.724565  1.768858         0.352089           0.0           0.0
266   0.835652  0.230479         0.352089           0.0           0.0
1417 -0.850345  2.170415         0.352089           0.0           0.0

      hours-per-week  workclass_Federal-gov  workclass_Local-gov  \
13         -1.941908                    0.0                  0.0
35         -0.087277                    0.0                  0.0
132        -0.087277                    0.0                  0.0
266        -0.087277                    0.0                  0.0
1417       -0.087277                    0.0                  0.0

      workclass_Never-worked  workclass_Private  ...  \
13                       0.0                0.0  ...
35                       0.0                0.0  ...
132                      0.0                0.0  ...
266                      0.0                0.0  ...
1417                     0.0                0.0  ...

      native-country_Puerto-Rico  native-country_Scotland  \
13                           0.0                      0.0
35                           0.0                      0.0
132                          0.0                      0.0
266                          0.0                      0.0
1417                         0.0                      0.0

      native-country_South  native-country_Taiwan  native-country_Thailand  \
13                     0.0                    0.0                      0.0
35                     0.0                    0.0                      0.0
132                    0.0                    0.0                      0.0
266                    0.0                    0.0                      0.0
1417                   0.0                    0.0                      0.0
```

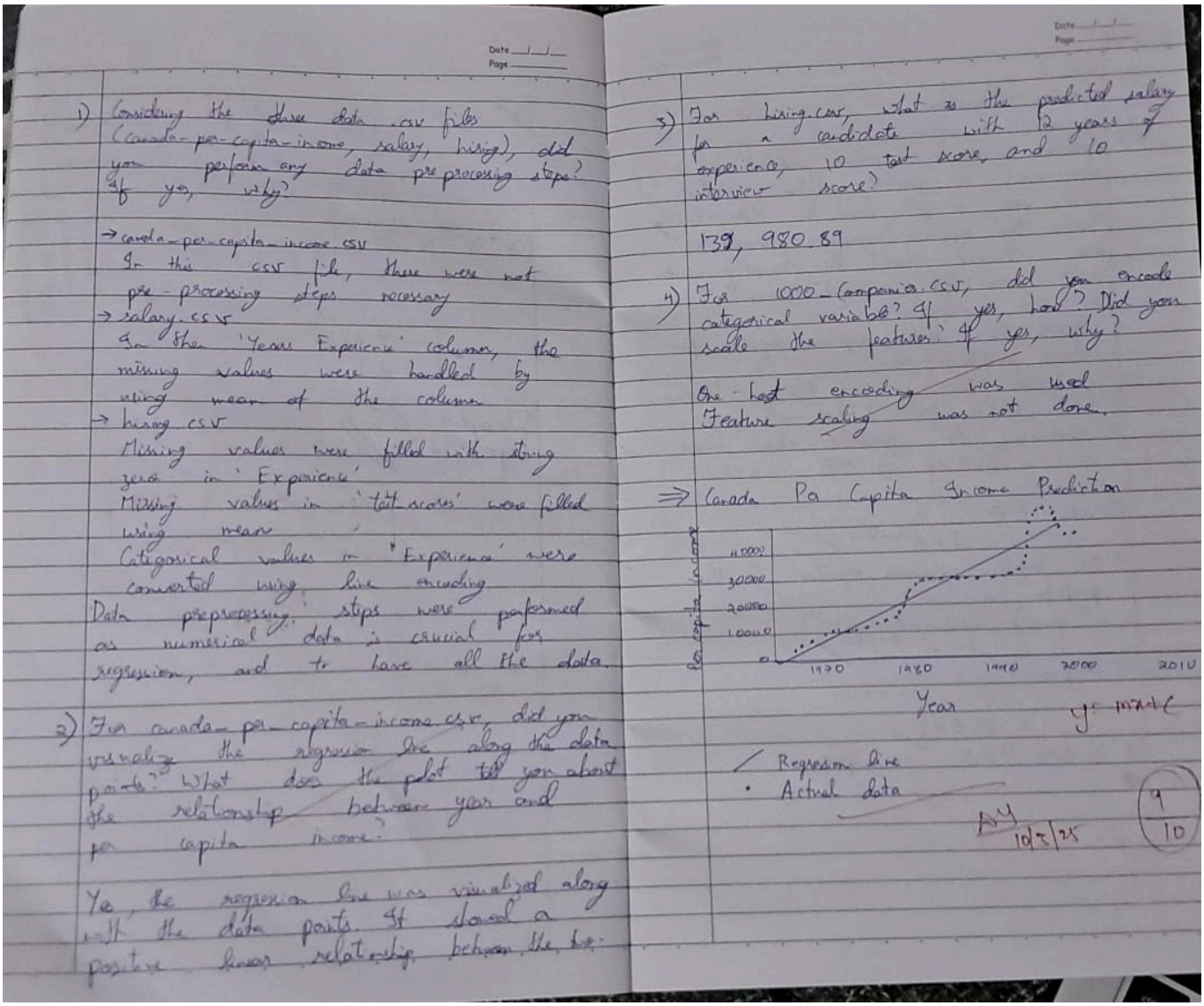
```
      native-country_Trinadad&Tobago  native-country_United-States  \
13                                0.0                           0.0
35                                0.0                           0.0
132                               0.0                           0.0
266                               0.0                           0.0
1417                              0.0                           0.0

      native-country_Vietnam  native-country_Yugoslavia  income_>50K
13                       0.0                        0.0          0.0
35                       0.0                        0.0          0.0
132                      0.0                        0.0          0.0
266                      0.0                        0.0          0.0
1417                     0.0                        0.0          0.0

[5 rows x 101 columns]

 Normalized dataset saved: /content/adult2.csv
 Standardized dataset saved: /content/adult3.csv
```

Program 3

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

Screenshot:





Code:

Lab 2 - 10/3/25

Linear and Multiple Regression

In [15]:

```
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 1000_Companies.csv to 1000_Companies.csv
```

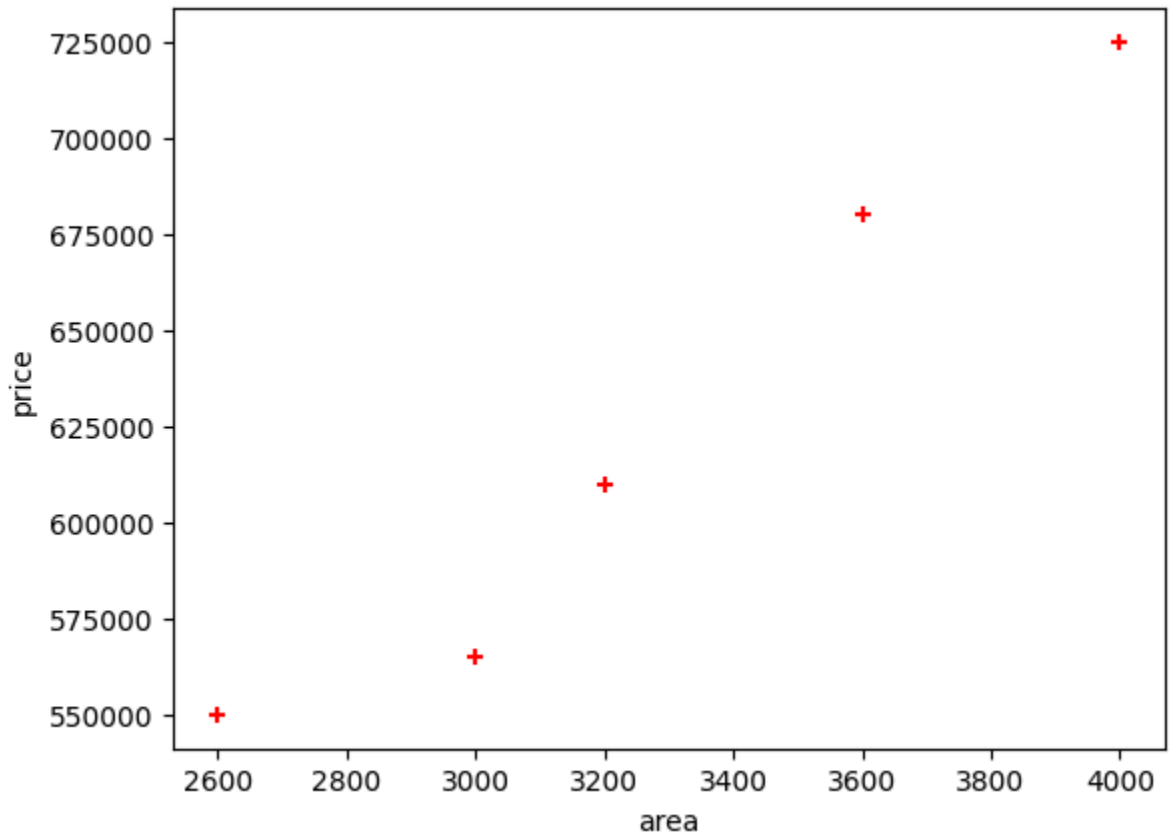
In [2]:

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

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

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

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

price = df.price
price

# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df,price)

"""(1) Predict price of a home with area = 3300 sqr ft"""

reg.predict([[3300]])

reg.coef_

reg.intercept_

"""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 have valid feature names, but LinearRegression was fitted
with feature names
  warnings.warn(
/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[2]:

```
array([859554.79452055])
```



In [5]:

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

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

"""Data Preprocessing: Fill NA values with median value of a column"""

df.bedrooms.median()

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

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]])

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

```
/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[5]:

```
498408.25157402386
```

To Do: Implementation – Linear Regression

Write Python code to implement the following

a) Predict canada's per capita income in year 2020. Use the data file canada_per_capita_income.csv file. If required, apply the necessary data processing steps. Using this build a regression model and predict the per capita income for canadian citizens in year 2020

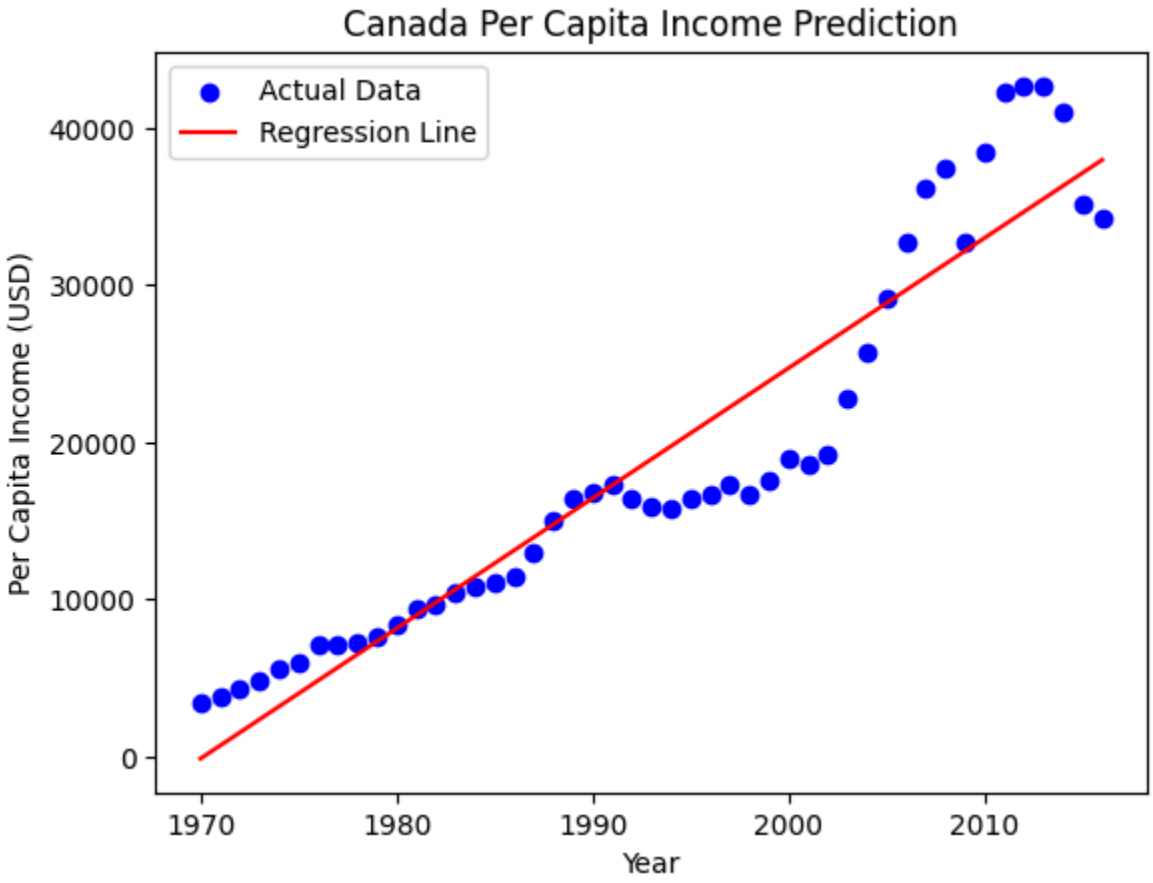
b) Predict Salary of the employee. Use the data file salary.csv file. If required, apply the necessary data processing steps. Using this build a regression model and predict the salary of the employee with 12 years of experience.

In [7]:

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

# Load and prepare the data (same as before)
df = pd.read_csv('canada_per_capita_income.csv')
df.columns = ['year', 'per_capita_income']
df['year'] = df['year'].astype(int)
df['per_capita_income'] = df['per_capita_income'].astype(int)

# Build and train the model (same as before)
model = linear_model.LinearRegression()
X = df[['year']]
y = df['per_capita_income']
model.fit(X, y)

# Predict for the year 2020 (same as before)
year_2020 = [[2020]]
predicted_income = model.predict(year_2020)

# Create a scatter plot of the data
plt.scatter(df['year'], df['per_capita_income'], color='blue', label='Actual Data')

# Plot the regression line
plt.plot(df['year'],  model.predict(df[['year']]),  color='red',  label='Regression
Line')
```

```
# Add labels and title
plt.xlabel('Year')
plt.ylabel('Per Capita Income (USD)')
plt.title('Canada Per Capita Income Prediction')

# Add a legend
plt.legend()

# Show the plot
plt.show()

# Print the prediction for 2020
print("Predicted per capita income for Canada in 2020:", predicted_income[0])
```

```
/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(
```




```
Predicted per capita income for Canada in 2020: 41288.294172063004
```

In [9]:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

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

print("Missing values in the dataset:")
print(df.isnull().sum())

df['YearsExperience'] = df['YearsExperience'].fillna(df['YearsExperience'].mean())

print("\nMissing values after filling:")
print(df.isnull().sum())

X = df[['YearsExperience']]
y = df['Salary']

reg = LinearRegression()
reg.fit(X, y)

predicted_salary_12_years = reg.predict([[12]])
print(f"\nPredicted salary for an employee with 12 years of experience:
${predicted_salary_12_years[0]:,.2f}")
```

```
Missing values in the dataset:
YearsExperience    2
Salary             0
dtype: int64

Missing values after filling:
YearsExperience    0
Salary             0
dtype: int64

Predicted salary for an employee with 12 years of experience: $139,980.89

/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(
```

To Do: Implementation – Multiple Linear Regression

Write Python code to implement the following

Considering the data file hiring.csv. The file contains hiring statics for a firm such as experience of candidate, his written test score and personal interview score. Based on these 3 factors, HR will decide the salary. Given this data, you need to build a Multiple Linear Regression model for HR department that can help them decide salaries for future candidates. Using this predict salaries for

following candidates,

- **2 yr experience, 9 test score, 6 interview score**
- **12 yr experience, 10 test score, 10 interview score**

Considering the data file 1000_companies.csv. The file contains profit statics for a firm such as R&D Spend, Administration, Marketing Spend and State. Based on these four factors build a Multiple Linear Regression model to predict the profit. Using this predict profit for following,

91694.48 R&D Spend, 515841.3 Administration, 11931.24 Marketing Spend, Florida State

Note: If required, apply the necessary data processing steps to data files.

In [13]:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

df = pd.read_csv('hiring.csv')
print("Missing values in the dataset:")
print(df.isnull().sum())

df['experience'] = df['experience'].fillna('zero')
df['test_score(out of 10)'] = df['test_score(out of 10)'].fillna(df['test_score(out
of 10)'].mean())
```

```
Missing values in the dataset:
experience                  2
test_score(out of 10)       1
interview_score(out of 10)  0
salary($)                   0
dtype: int64
```

In [14]:

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

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

df['experience'] = df['experience'].replace(
    {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6,
     'seven': 7, 'eight': 8, 'nine': 9, 'ten': 10, 'eleven': 11}
)
df['experience'] = df['experience'].fillna(df['experience'].median())
df['test_score(out of 10)'] = df['test_score(out of 10)'].fillna(df['test_score(out
of 10)'].median())
```

```
print("\nMissing values after filling:")
print(df.isnull().sum())

X = df[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y = df['salary($)']

reg = LinearRegression()
reg.fit(X, y)

candidate_1 = [[2, 9, 6]]
candidate_2 = [[12, 10, 10]]

predicted_salary_1 = reg.predict(candidate_1)
predicted_salary_2 = reg.predict(candidate_2)

print(f"\nPredicted salary for candidate 1 (2 years experience, 9 test score, 6 interview score): ${predicted_salary_1[0]:,.2f}")
print(f"Predicted salary for candidate 2 (12 years experience, 10 test score, 10 interview score): ${predicted_salary_2[0]:,.2f}")
```

```
Missing values after filling:
experience                    0
test_score(out of 10)         0
interview_score(out of 10)    0
salary($)                     0
dtype: int64

Predicted salary for candidate 1 (2 years experience, 9 test score, 6 interview score): $47,056.91
Predicted salary for candidate 2 (12 years experience, 10 test score, 10 interview score): $88,227.64


<ipython-input-14-cf7ec233a4f2>:9: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df['experience'] = df['experience'].replace(
/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(
/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(
```

In [16]:

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

```
df = pd.read_csv('1000_Companies.csv')

df
```

Out[16]:

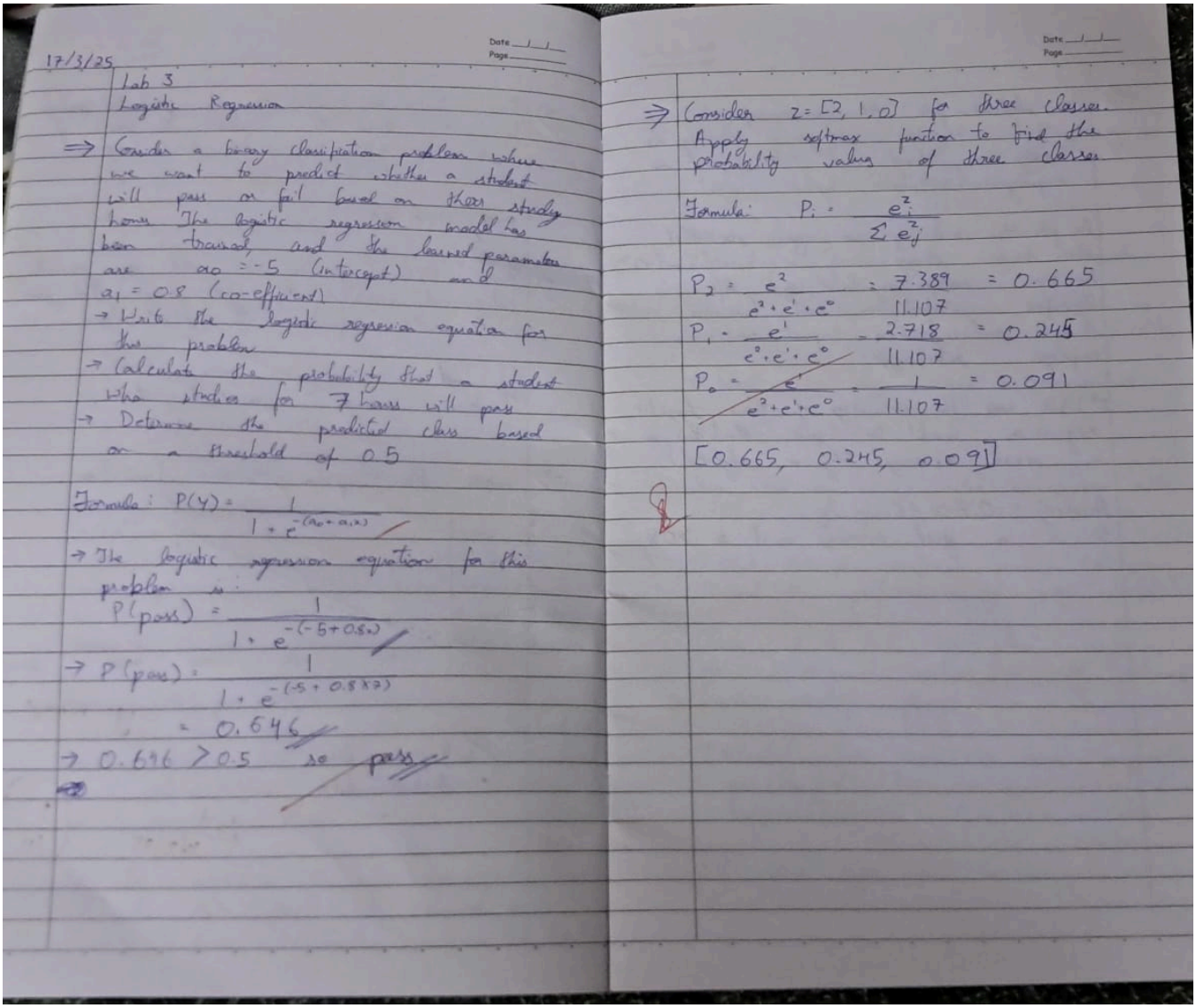
	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.800	471784.1000	New York	192261.83000
1	162597.70	151377.590	443898.5300	California	191792.06000
2	153441.51	101145.550	407934.5400	Florida	191050.39000
3	144372.41	118671.850	383199.6200	New York	182901.99000
4	142107.34	91391.770	366168.4200	Florida	166187.94000
...	...	...	...	...	...
995	54135.00	118451.999	173232.6695	California	95279.96251
996	134970.00	130390.080	329204.0228	California	164336.60550
997	100275.47	241926.310	227142.8200	California	413956.48000

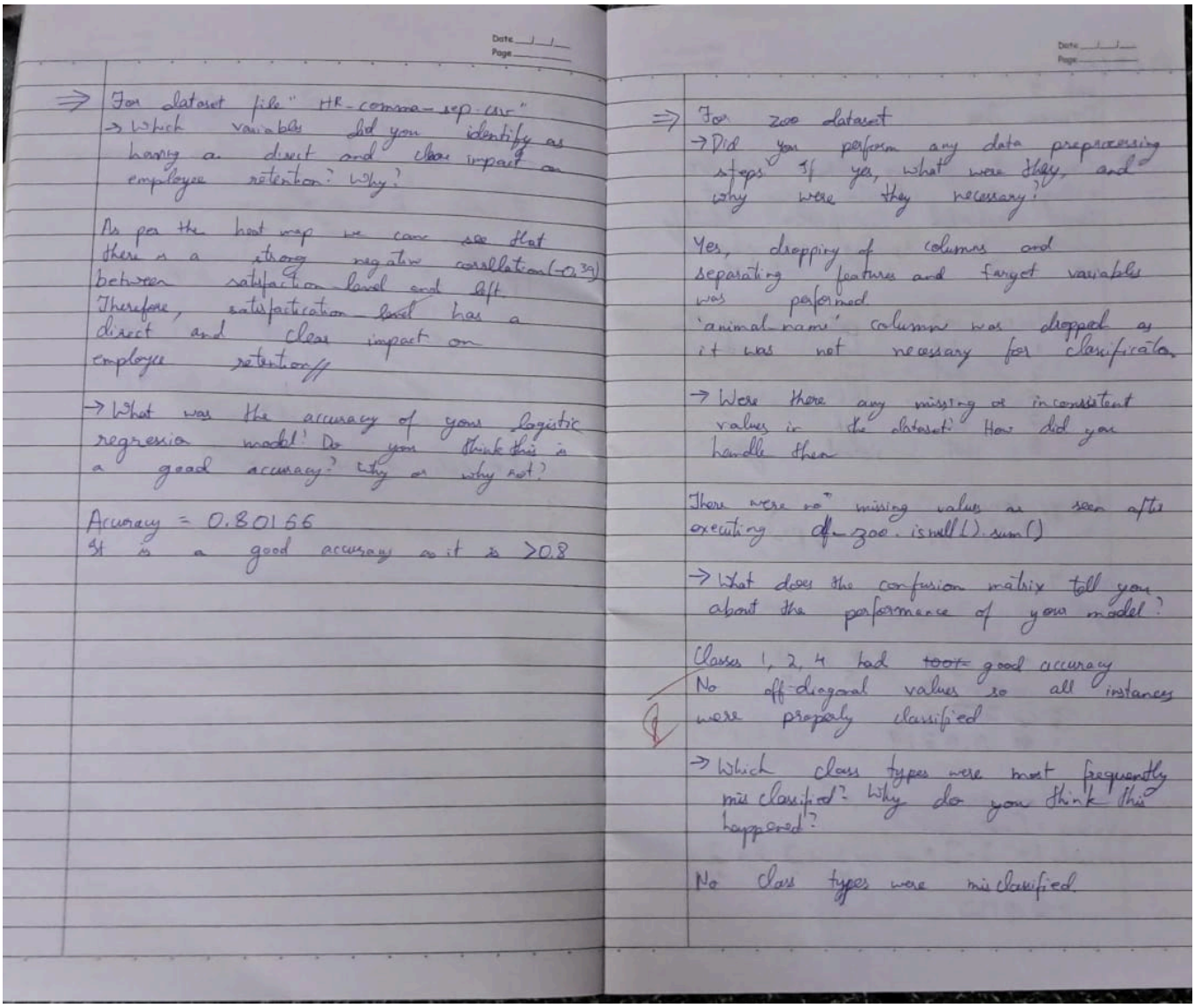
998	128456.23	321652.140	281692.3200	California	333962.19000
999	161181.72	270939.860	295442.1700	New York	476485.43000

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:





⇒ For dataset file " HR-comma-sep.csv"
→ Which variables did you identify as having a direct and clear impact on employee retention? Why?

As per the heat map we can see that there is a strong negative correlation(-0.39) between satisfaction level and left. Therefore, satisfaction level has a direct and clear impact on employee retention.

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

Accuracy = 0.80166
It is a good accuracy as it is >0.8

⇒ For zoo dataset
→ Did you perform any data preprocessing steps? If yes, what were they, and why were they necessary?

Yes, dropping of columns and separating features and target variables was performed.
'animal_name' column was dropped as it was not necessary for classification.

→ Were there any missing or inconsistent values in the dataset? How did you handle them?

There were no missing values as seen after executing df_zoo.isnull().sum()

→ What does the confusion matrix tell you about the performance of your model?

Classes 1, 2, 4 had good accuracy
No off-diagonal values so all instances were properly classified

→ Which class types were most frequently misclassified? Why do you think this happened?

No class types were misclassified.

Code:

Lab-3

Build Logistic Regression Model for a given dataset

In [8]:

```
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.

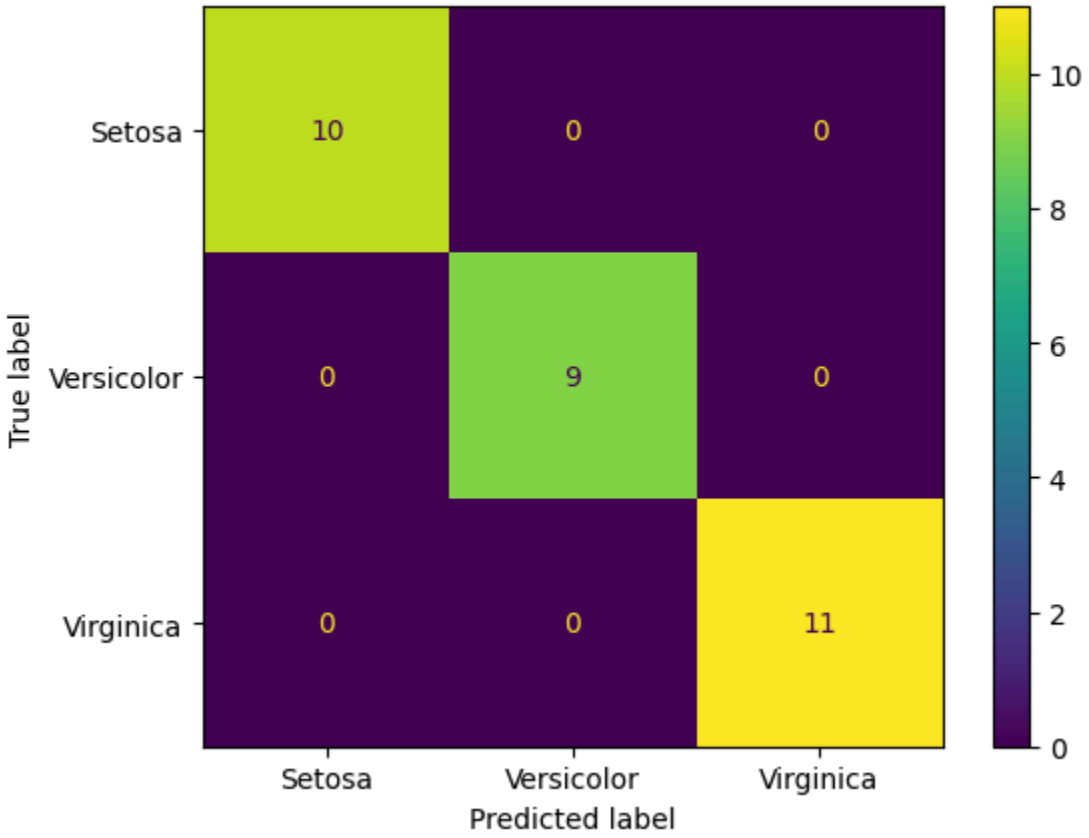
```
Saving zoo-class-type.csv to zoo-class-type.csv
```

In [3]:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt

# Load the Iris dataset
iris =  pd.read_csv("iris (3).csv")
iris.head()

X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, petal width)
y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

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

# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver
model = LogisticRegression(multi_class='multinomial')

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)

# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ["Setosa", "Versicolor", "Virginica"])

cm_display.plot()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247:
FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in
1.7. From then on, it will always use 'multinomial'. Leave it to its default value
to avoid this warning.
  warnings.warn(
```

```
Accuracy of the Multinomial Logistic Regression model on the test set: 1.00
```



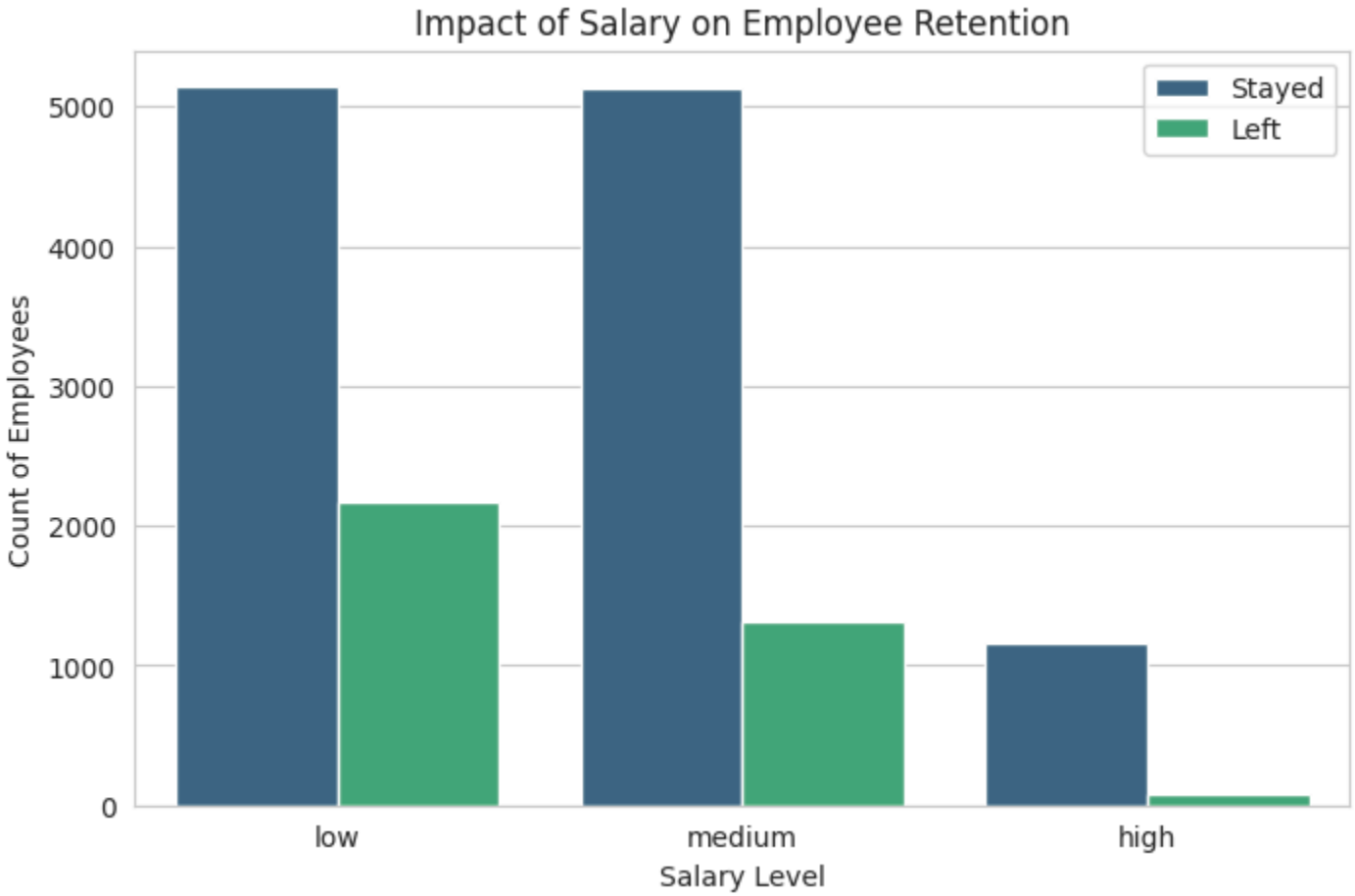
In [4]:

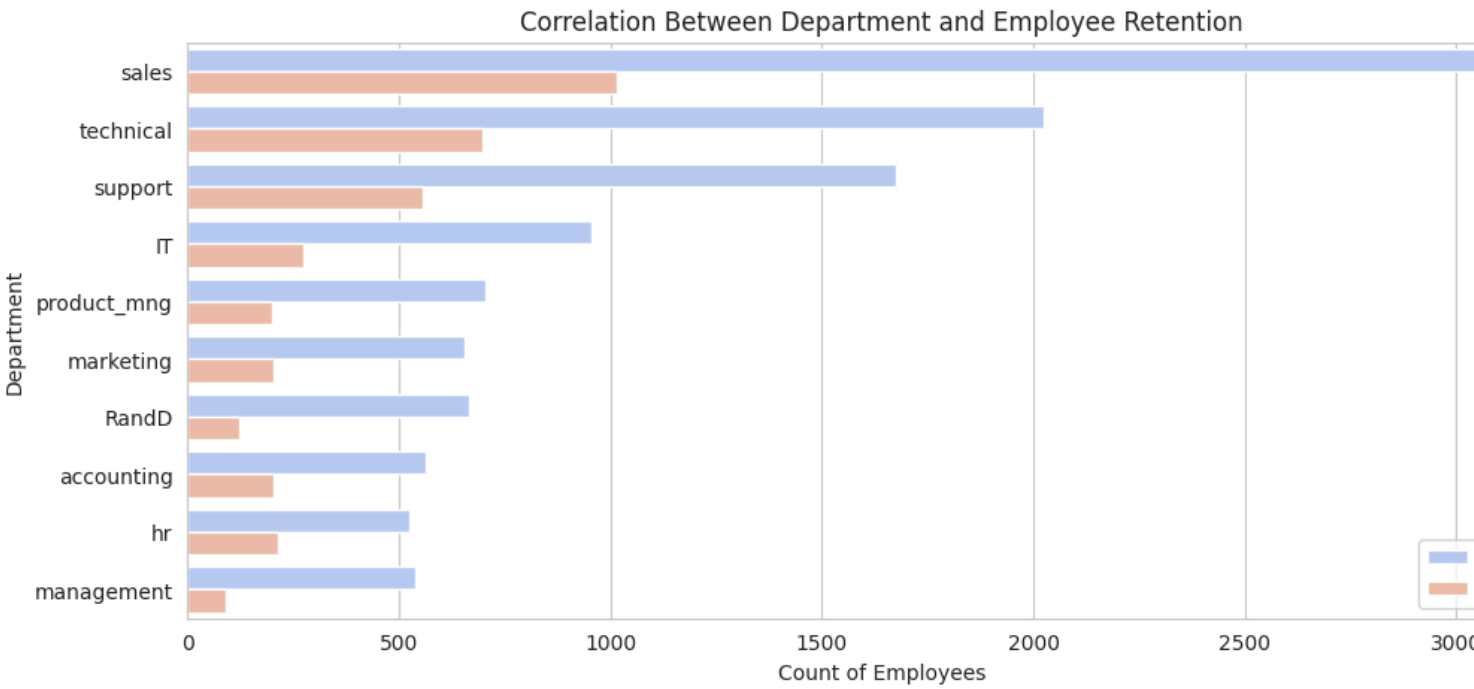
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

df = pd.read_csv("HR_comma_sep.csv")
print(df.head())

missing_values = df.isnull().sum()
# Display columns with missing values
print(missing_values[missing_values > 0])

# Set seaborn style
sns.set_style("whitegrid")

# Plot bar chart for salary vs retention
plt.figure(figsize=(8, 5))
sns.countplot(x="salary", hue="left", data=df, palette="viridis")
plt.xlabel("Salary Level")
plt.ylabel("Count of Employees")
```

```
plt.title("Impact of Salary on Employee Retention")
plt.legend(["Stayed", "Left"])
plt.show()

# Plot bar chart for department vs retention
plt.figure(figsize=(12, 5))
sns.countplot(y="Department",      hue="left",      data=df,      palette="coolwarm",
order=df["Department"].value_counts().index)
plt.xlabel("Count of Employees")
plt.ylabel("Department")
plt.title("Correlation Between Department and Employee Retention")
plt.legend(["Stayed", "Left"])
plt.show()

# Encode categorical variables
label_encoders = {}
for col in ["salary", "Department"]:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Select relevant features
features     =     ["satisfaction_level",     "last_evaluation",     "number_project",
"average_montly_hours",
                    "time_spend_company",  "Work_accident",  "promotion_last_5years",
"salary", "Department"]
X = df[features]
y = df["left"]

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

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

# Train logistic regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Predict on test set
y_pred = log_reg.predict(X_test)

# Measure accuracy
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(classification_rep)
```

```
  satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
```

```
2                0.11             0.88                7                   272
3                0.72             0.87                5                   223
4                0.37             0.52                2                   159

   time_spend_company  Work_accident  left  promotion_last_5years Department  \
0                   3              0     1                      0      sales
1                   6              0     1                      0      sales
2                   4              0     1                      0      sales
3                   5              0     1                      0      sales
4                   3              0     1                      0      sales

   salary
0     low
1  medium
2  medium
3     low
4     low
Series([], dtype: int64)
```






```
Model Accuracy: 0.7577
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.92      0.85      2294
           1       0.47      0.23      0.31       706

    accuracy                           0.76      3000
   macro avg       0.63      0.57      0.58      3000
weighted avg       0.72      0.76      0.72      3000
```

To Do: Implementation – Logistic Regression (Binary Classification)

Write Python code to implement the following. Consider dataset file as "HR_comma_sep.csv"

Do some exploratory data analysis to figure out which variables have direct and clear impact on employee retention (i.e. whether they leave the company or continue to work)

Plot bar charts showing impact of employee salaries on retention

Plot bar charts showing corelation between department and employee retention

Build logistic regression model using variables that were narrowed down in step 1

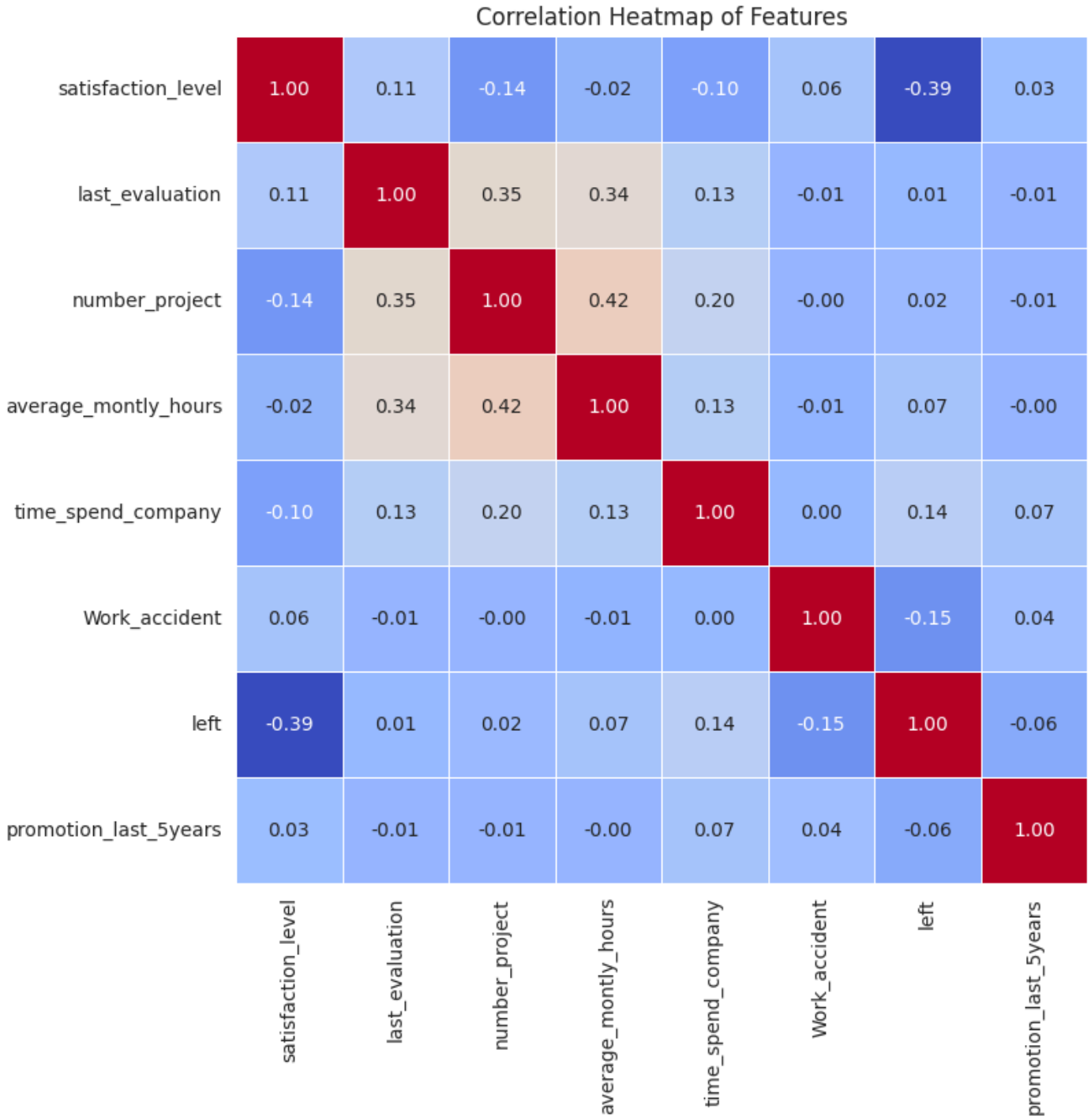
Measure the accuracy of the model

In [6]:

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

# Load the dataset
df = pd.read_csv("HR_comma_sep.csv")
print(df.head())

# Correlation Heatmap (Before any transformation)
numerical_features = df.select_dtypes(include=['number']).columns

# Compute the correlation matrix for numerical features
correlation_matrix = df[numerical_features].corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix,     annot=True,     cmap='coolwarm',     fmt='.2f',
linewidths=0.5)
plt.title('Correlation Heatmap of Features')
plt.show()

# Create a bar chart showing the impact of employee salaries on retention
plt.figure(figsize=(8, 6))
pd.crosstab(df.salary, df.left).plot(kind='bar')
plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary')
plt.ylabel('Number of Employees')
plt.show()

# Create a bar chart showing the correlation between department and employee
retention
plt.figure(figsize=(10, 6))
pd.crosstab(df.Department, df.left).plot(kind='bar')
plt.title('Impact of Department on Employee Retention')
plt.xlabel('Department')
plt.ylabel('Number of Employees')
plt.show()

# Convert categorical features into numerical using one-hot encoding
df = pd.get_dummies(df, columns=['salary', 'Department'], drop_first=True)

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

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

# Initialize and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

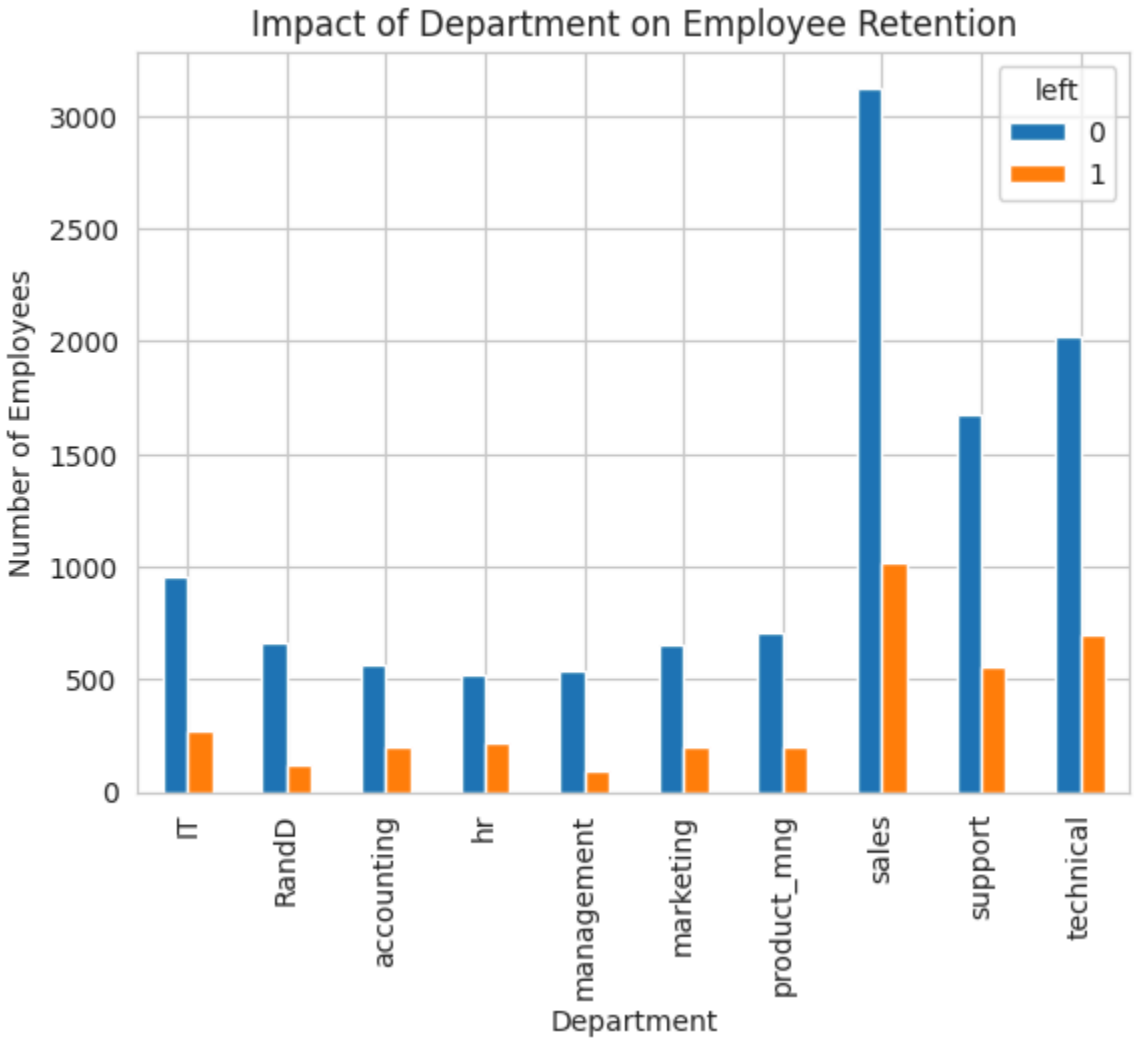
```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy of the model:", accuracy, "\n")
```

```
  satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
2                0.11             0.88               7                   272
3                0.72             0.87               5                   223
4                0.37             0.52               2                   159

   time_spend_company  Work_accident  left  promotion_last_5years Department  \
0                   3              0     1                      0      sales
1                   6              0     1                      0      sales
2                   4              0     1                      0      sales
3                   5              0     1                      0      sales
4                   3              0     1                      0      sales

   salary
0     low
1  medium
2  medium
3     low
4     low
```




```
<Figure size 800x600 with 0 Axes>
```


```
<Figure size 1000x600 with 0 Axes>
```




```
Accuracy of the model: 0.7986666666666666


/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(
```

To Do: Implementation – Logistic Regression (Multiclass Classification)

Write Python code to implement the following. Consider dataset file "zoo-data.csv" to predi . Details of class type is provided in "zoo-class_type .csv"

If require apply necessary data preprocessing.

Build logistic regression model to predict “class_type”

Measure the accuracy of the model

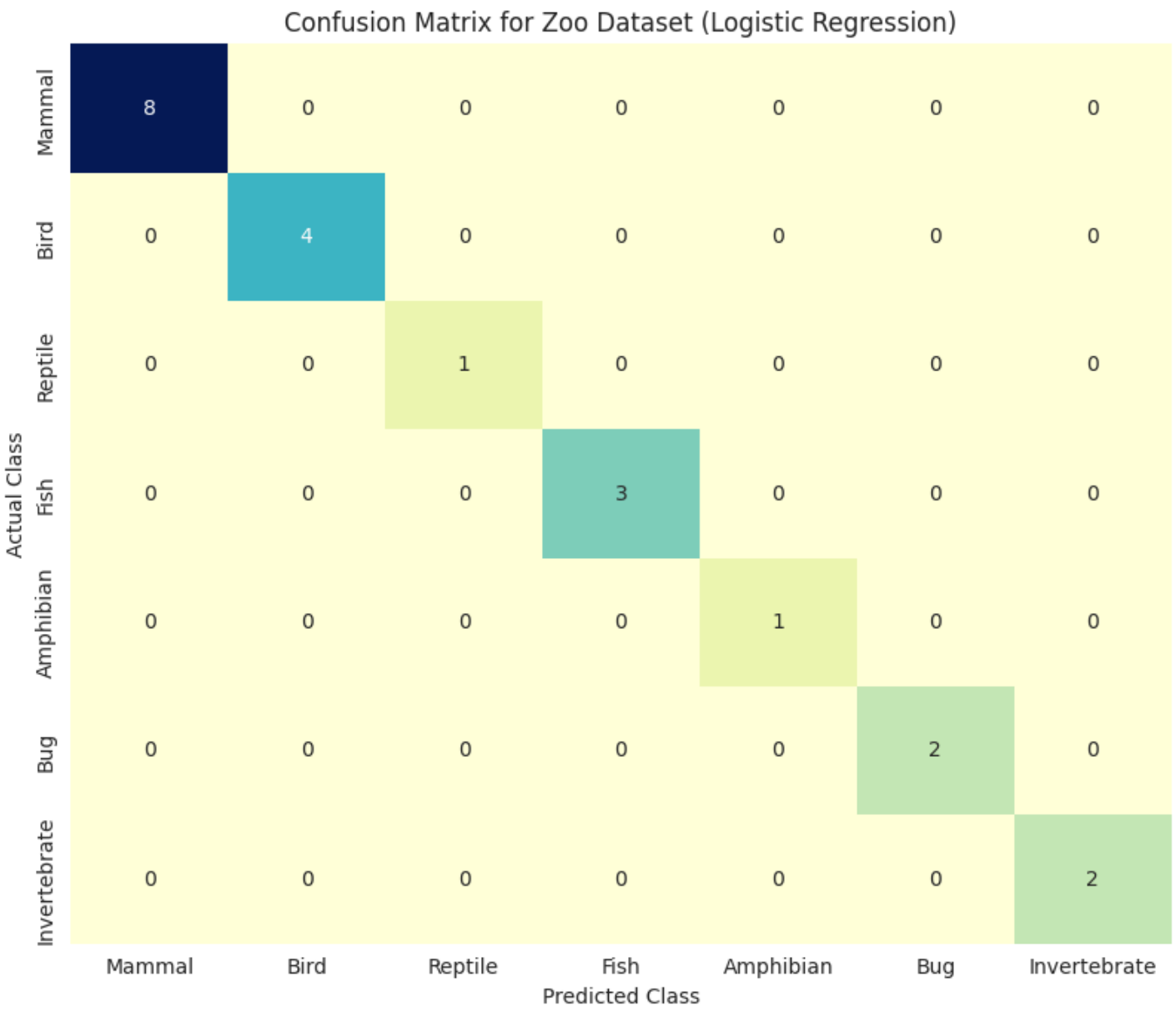
Plot the confusion matrix

In [9]:

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

# Load datasets
df_zoo = pd.read_csv("zoo-data.csv")
df_class = pd.read_csv("zoo-class-type.csv")

# Drop 'animal_name' as it's not needed for prediction
df_zoo = df_zoo.drop(columns=["animal_name"])

# Prepare features and target
X = df_zoo.drop(columns=["class_type"])
y = df_zoo["class_type"]

# Map class numbers to class names for later use in confusion matrix
class_map = df_class.set_index("Class_Number")["Class_Type"].to_dict()
class_labels = [class_map[i] for i in sorted(class_map.keys())]

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

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

# Train logistic regression
model = LogisticRegression(multi_class="multinomial", max_iter=1000)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred,
```

```
target_names=class_labels))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix with class names
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix,          annot=True,          fmt="d",          cmap="YlGnBu",
xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.title("Confusion Matrix for Zoo Dataset (Logistic Regression)")
plt.tight_layout()
plt.show()
```

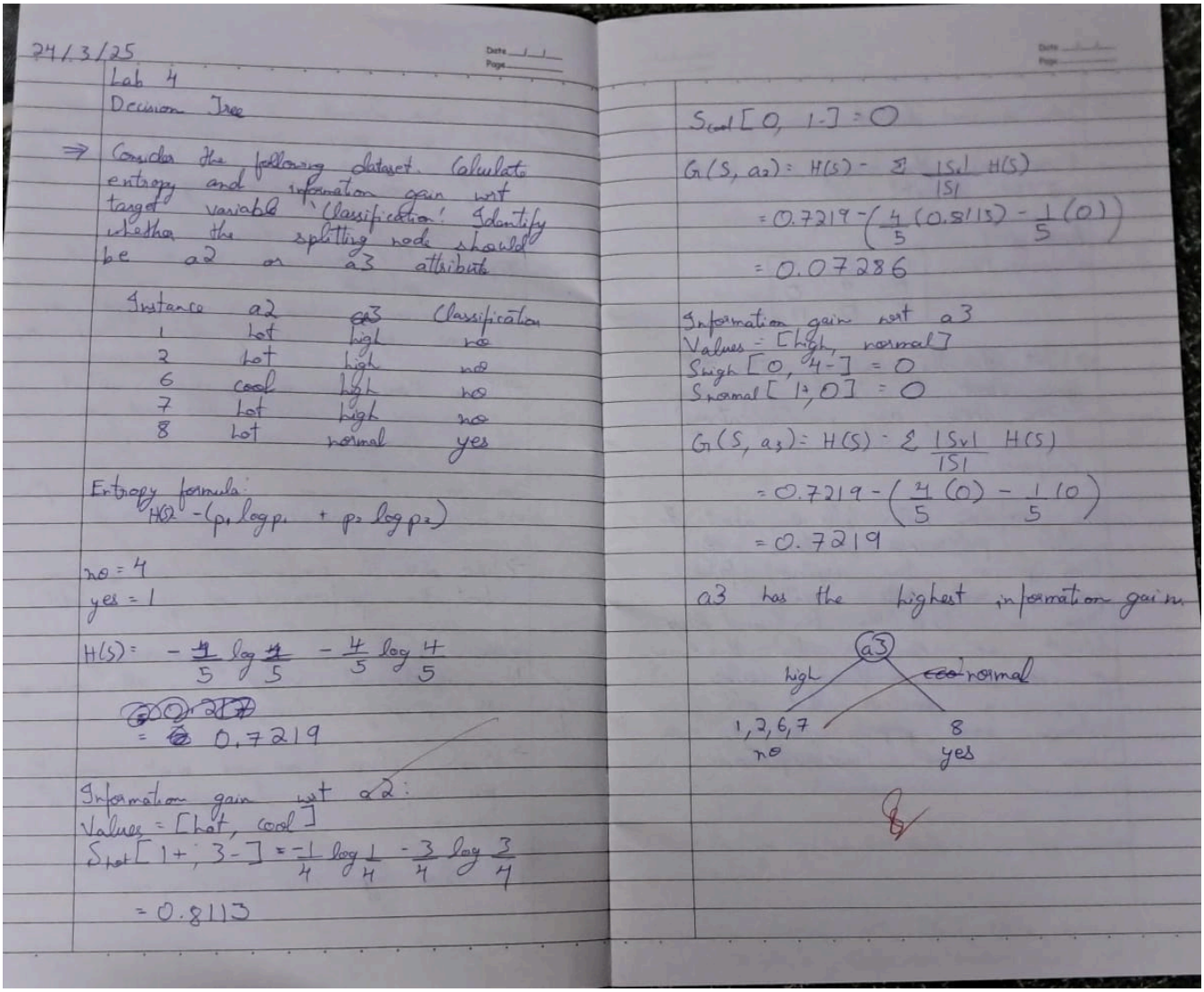
```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247:
FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in
1.7. From then on, it will always use 'multinomial'. Leave it to its default value
to avoid this warning.
  warnings.warn(

Model Accuracy: 1.0000

Classification Report:
               precision    recall  f1-score   support

      Mammal       1.00      1.00      1.00         8
        Bird       1.00      1.00      1.00         4
     Reptile       1.00      1.00      1.00         1
        Fish       1.00      1.00      1.00         3
   Amphibian       1.00      1.00      1.00         1
         Bug       1.00      1.00      1.00         2
Invertebrate       1.00      1.00      1.00         2

    accuracy                           1.00        21
   macro avg       1.00      1.00      1.00        21
weighted avg       1.00      1.00      1.00        21
```

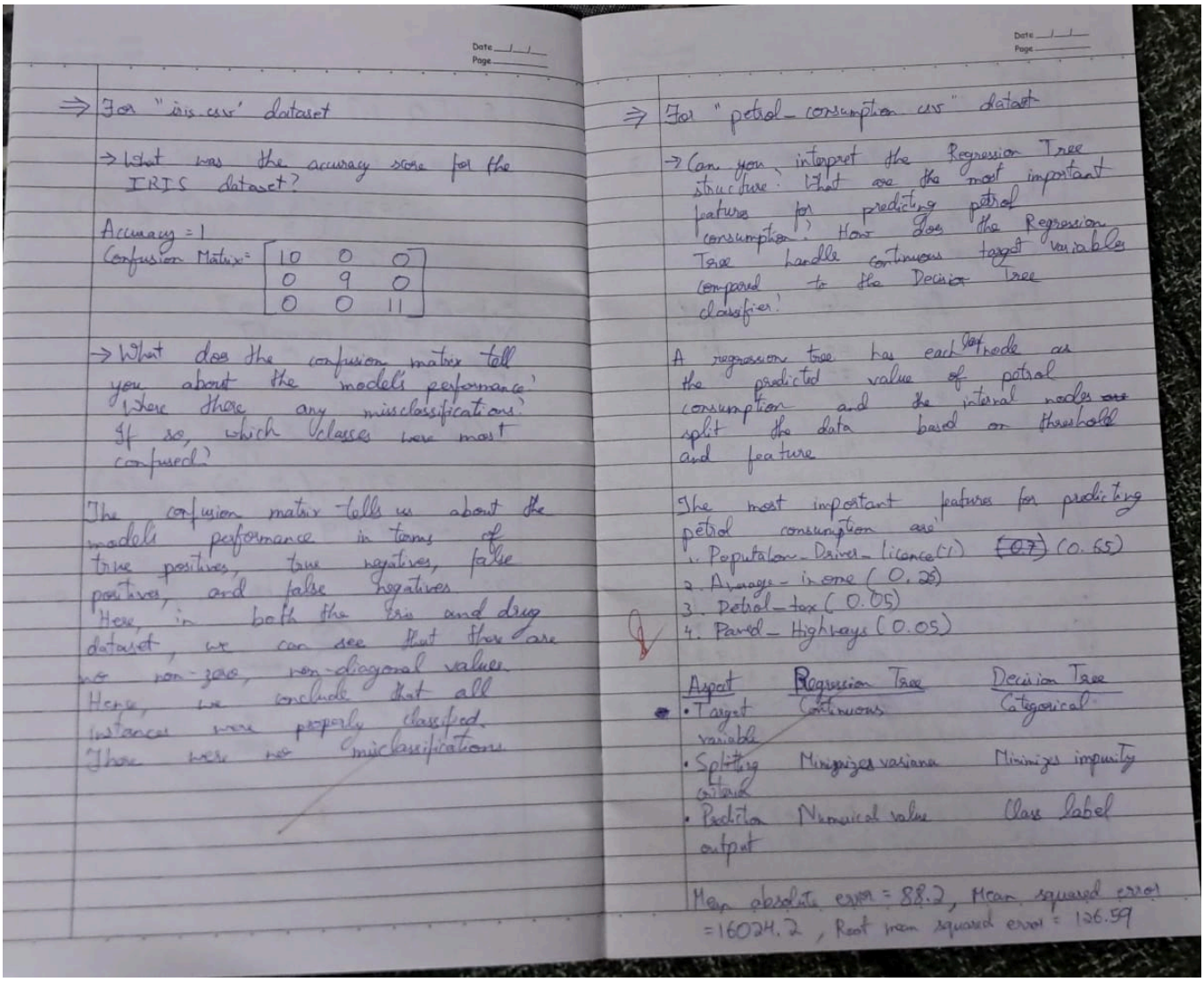

Confusion Matrix for Zoo Dataset (Logistic Regression)
Actual Class
Predicted Class
Mammal
Bird
Reptile
Fish
Amphibian
Bug
Invertebrate
8 0 0 0 0 0 0
0 4 0 0 0 0 0
0 0 1 0 0 0 0
0 0 0 3 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 2 0
0 0 0 0 0 0 2

Program 5

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

Screenshot:





Code:

Lab 4 - 24/03/25

Decision Tree

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.

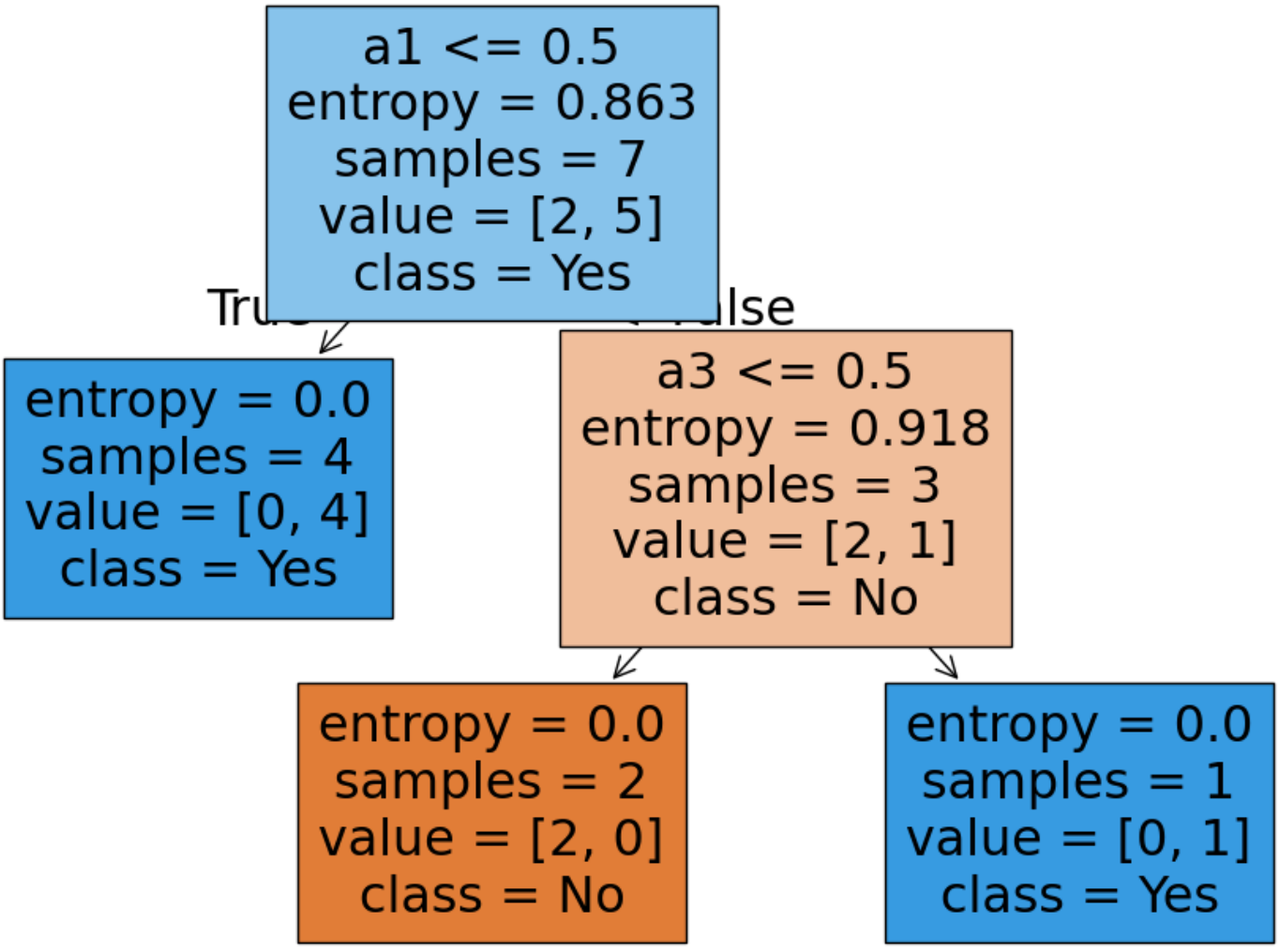
```
Saving petrol_consumption.csv to petrol_consumption.csv
```

In [1]:

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

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

data

# Convert to DataFrame
df = pd.DataFrame(data)

# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

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

# Split the data into training and testing sets
X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,   test_size=0.3,
random_state=42)

# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

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

# Optionally, visualize the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

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

plt.show()
```

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

          No       1.00      1.00      1.00         2
         Yes       1.00      1.00      1.00         1

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



To Do: Implementation – Decision Tree(for

Classification)

Write Python code to implement the following. Consider dataset files as “iris.csv” and “drug.csv”

1. **Build a DecisionTree classifier to classify IRIS flower dataset Use 80% of data for training and 20% for testing. Display accuracy score and confusion matrix of the trained model on test data.**
2. **Build a DecisionTree classifier to classify Drugdataset Use 80% of data for training and 20% for testing. Display accuracy score and confusion matrix of the trained model on test data.**

In [6]:

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

# Load the IRIS dataset
iris = pd.read_csv("iris.csv")
X_iris = iris.iloc[:, :-1]  # Features
y_iris = iris.iloc[:, -1]   # Target

# Split the dataset
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris,
y_iris, test_size=0.2, random_state=42)

# Train the Decision Tree classifier
clf_iris = DecisionTreeClassifier()
clf_iris.fit(X_train_iris, y_train_iris)

# Make predictions
y_pred_iris = clf_iris.predict(X_test_iris)

# Evaluate the model
accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
conf_matrix_iris = confusion_matrix(y_test_iris, y_pred_iris)
print("IRIS Dataset - Decision Tree Classifier")
print(f"Accuracy: {accuracy_iris:.2f}")
print("Confusion Matrix:")
print(conf_matrix_iris)

# Load the Drug dataset
drug = pd.read_csv("drug.csv")
X_drug = drug.iloc[:, :-1]  # Features
y_drug = drug.iloc[:, -1]   # Target

# Convert categorical columns into numeric using Label Encoding
label_encoders = {}   # Store label encoders for later inverse transformation if
needed
categorical_columns = ["Sex", "BP", "Cholesterol"]

for col in categorical_columns:
    le = LabelEncoder()
```

```
    X_drug[col] = le.fit_transform(X_drug[col])
    label_encoders[col] = le  # Store the encoder

# Split the dataset
X_train_drug, X_test_drug, y_train_drug, y_test_drug = train_test_split(X_drug,
y_drug, test_size=0.2, random_state=42)

# Train the Decision Tree classifier
clf_drug = DecisionTreeClassifier()
clf_drug.fit(X_train_drug, y_train_drug)

# Make predictions
y_pred_drug = clf_drug.predict(X_test_drug)

# Evaluate the model
accuracy_drug = accuracy_score(y_test_drug, y_pred_drug)
conf_matrix_drug = confusion_matrix(y_test_drug, y_pred_drug)

print("\nDrug Dataset - Decision Tree Classifier")
print(f"Accuracy: {accuracy_drug:.2f}")
print("Confusion Matrix:")
print(conf_matrix_drug)
```

```
IRIS Dataset - Decision Tree Classifier
Accuracy: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Drug Dataset - Decision Tree Classifier
Accuracy: 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]]
```

To Do: Implementation – Decision Tree(For Classification)

Write Python code to implement the following. Consider dataset file "petrol_consumption.csv"

Build a RegressionTree to predict petrol consumption

Use 80% of data for training and 20% for testing.

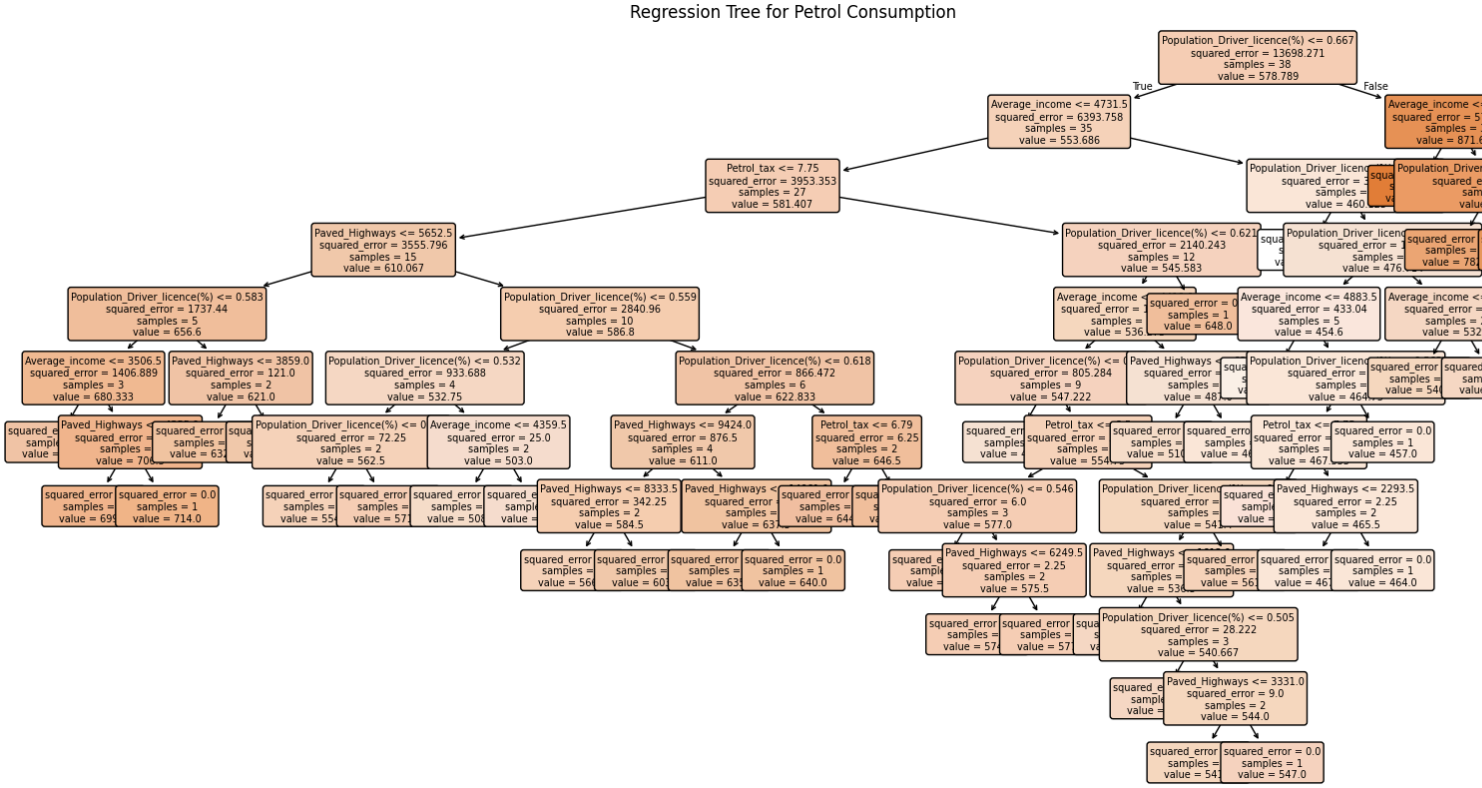
Display "Mean Absolute Error, Mean Squared Error, Root Mean Squared Error" for test data.

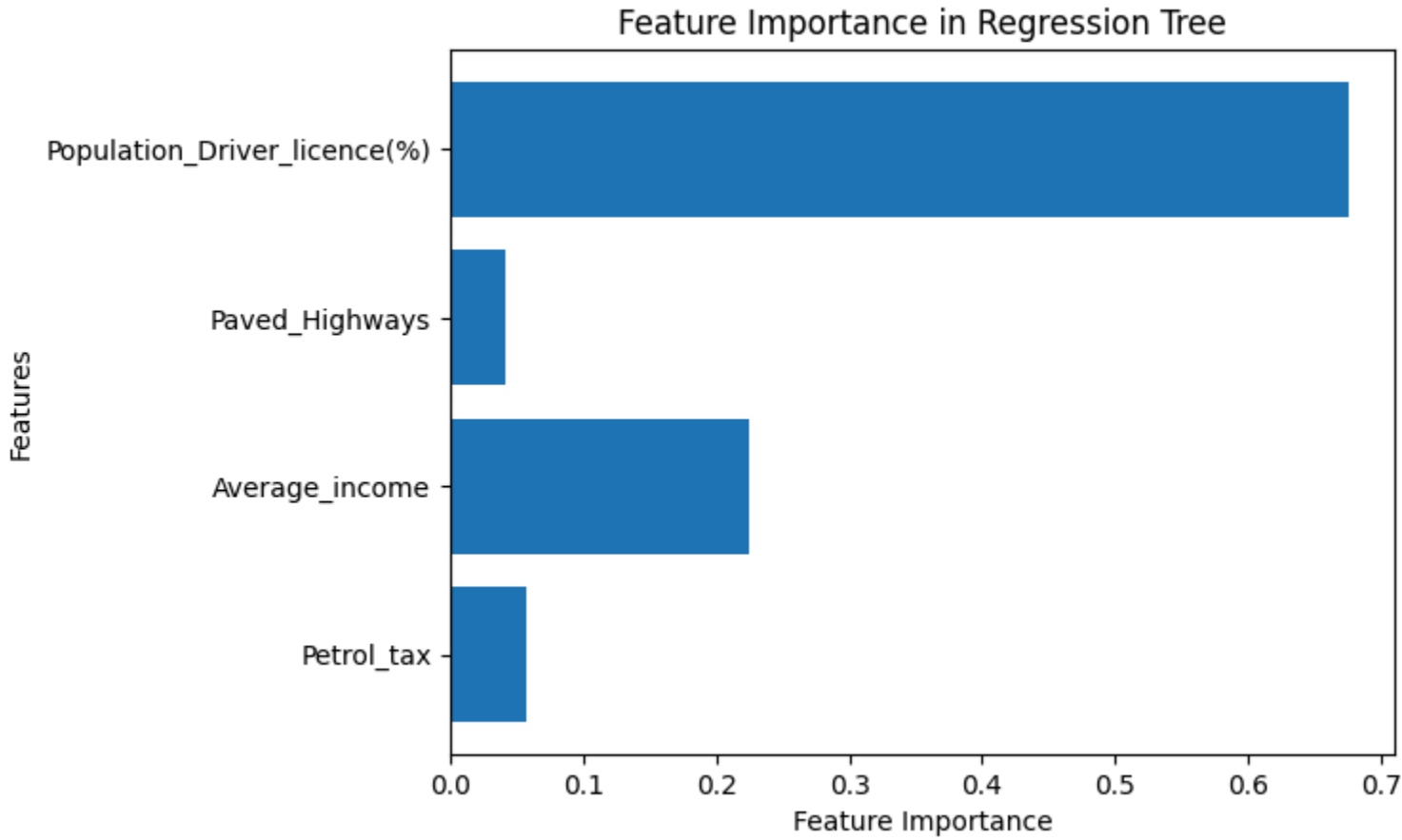
In [7]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Load the Petrol Consumption dataset
petrol = pd.read_csv("petrol_consumption.csv")

# Separate features and target
X_petrol = petrol.iloc[:, :-1]  # Features
y_petrol = petrol.iloc[:, -1]   # Target

# Split the dataset
X_train_petrol,      X_test_petrol,      y_train_petrol,      y_test_petrol      =
train_test_split(X_petrol, y_petrol, test_size=0.2, random_state=42)

# Train the Regression Tree model
regressor = DecisionTreeRegressor()
regressor.fit(X_train_petrol, y_train_petrol)

# Make predictions
y_pred_petrol = regressor.predict(X_test_petrol)

# Evaluate the model
mae = mean_absolute_error(y_test_petrol, y_pred_petrol)
mse = mean_squared_error(y_test_petrol, y_pred_petrol)
rmse = np.sqrt(mse)

print("Regression Tree - Petrol Consumption Prediction")
print(f"Mean Absolute Error: {mae:.2f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}\n")

plt.figure(figsize=(20, 10))
plot_tree(regressor,   filled=True,   feature_names=X_petrol.columns,   fontsize=7,
rounded=True)
plt.title("Regression Tree for Petrol Consumption")
plt.show()
print("\n")

# Get feature importance
importance = regressor.feature_importances_

# Plot feature importance
plt.barh(range(len(importance)), importance, tick_label=X_petrol.columns)
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importance in Regression Tree")
plt.show()
```

```
Regression Tree - Petrol Consumption Prediction
```

Mean Absolute Error: 97.80
Mean Squared Error: 18369.60
Root Mean Squared Error: 135.53

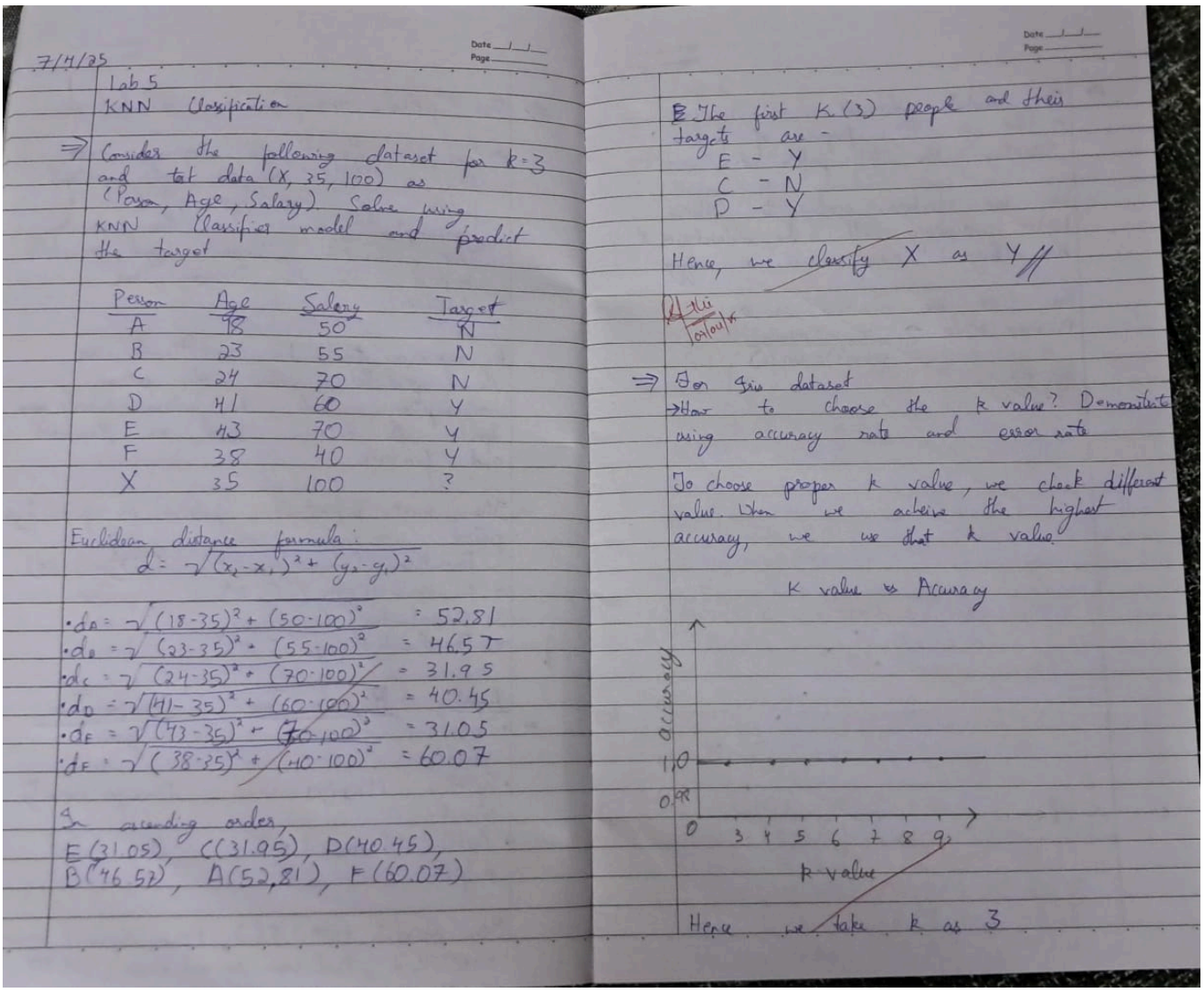



Feature Importance in Regression Tree
Population_Driver_licence(%)
Paved_Highways
Average_income
Petrol_tax
Features
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Feature Importance

Program 6

Build KNN Classification model for a given dataset.

Screenshot:



7/4/25

Lab 5

KNN Classification

⇒ Consider the following dataset for k=3 and test data (X, 35, 100) as (Person, Age, Salary). Solve using KNN Classifier model and predict the target

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

Euclidean distance formula:

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

- $d_A = \sqrt{(18-35)^2 + (50-100)^2} = 52.81$
- $d_B = \sqrt{(23-35)^2 + (55-100)^2} = 46.57$
- $d_C = \sqrt{(24-35)^2 + (70-100)^2} = 31.95$
- $d_D = \sqrt{(41-35)^2 + (60-100)^2} = 40.45$
- $d_E = \sqrt{(43-35)^2 + (70-100)^2} = 31.05$
- $d_F = \sqrt{(38-35)^2 + (40-100)^2} = 60.07$

In ascending order,

E(31.05), C(31.95), D(40.45), B(46.57), A(52.81), F(60.07)

∴ The first K (3) people and their targets are -

E - Y

C - N

D - Y

Hence, we classify X as Y //

⇒ Based on given dataset

→ How to choose the k value? Demonstrate using accuracy rate and error rate

To choose proper k value, we check different value. When we achieve the highest accuracy, we use that k value.

k value vs Accuracy

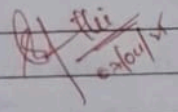
Hence we take k as 3

Date __/__/__
Page _____

⇒ For diabetes dataset
→ What is the purpose of feature scaling? How to perform it?

We use feature scaling to achieve higher accuracy with this dataset as it has a large range and is varying.

Here, we use standard scaler.
Minimax scaler → $\frac{x - \min(x)}{\max(x) - \min(x)}$



Code:

Lab 5

Build KNN Classification model for a given dataset

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 heart.csv to heart.csv
```

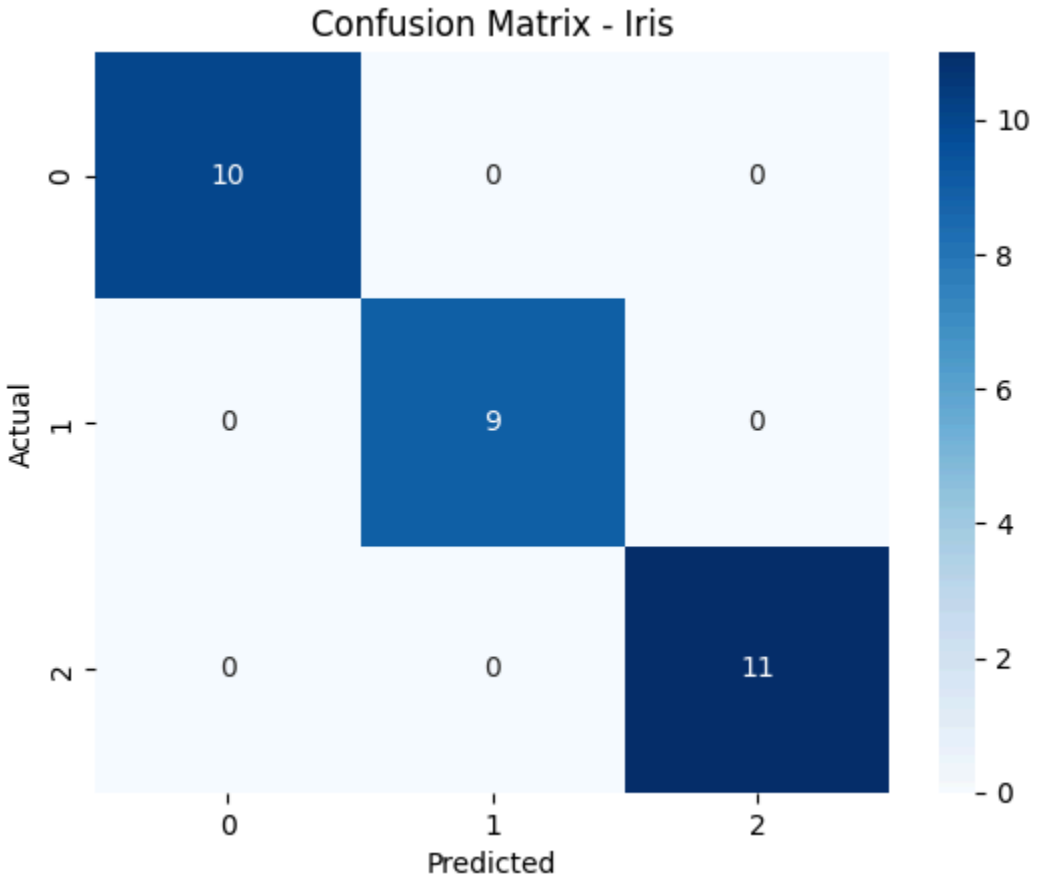
Build a KNN classifier to classify IRIS flower dataset. Choose appropriate k value and predict the score ,display confusion matrix and classification report. Use 80% of data for training and 20% for testing. Display accuracy score and confusion matrix of the trained model on test data.

In [ ]:

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

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

# Encode categorical target labels
le = LabelEncoder()
iris_df['species'] = le.fit_transform(iris_df['species'])

# Split features and labels
X = iris_df.drop('species', axis=1)
y = iris_df['species']

# Split 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)

# Initialize and train KNN
knn_iris = KNeighborsClassifier(n_neighbors=3)
knn_iris.fit(X_train, y_train)
```

```python
# Predict
y_pred = knn_iris.predict(X_test)

# Evaluation
print("Accuracy Score:", accuracy_score(y_test, y_pred),"\n")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred),"\n")
print("Classification Report:\n", classification_report(y_test, y_pred),"\n")

# Plot confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt="d")
plt.title("Confusion Matrix - Iris")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Accuracy Score: 1.0

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

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
```



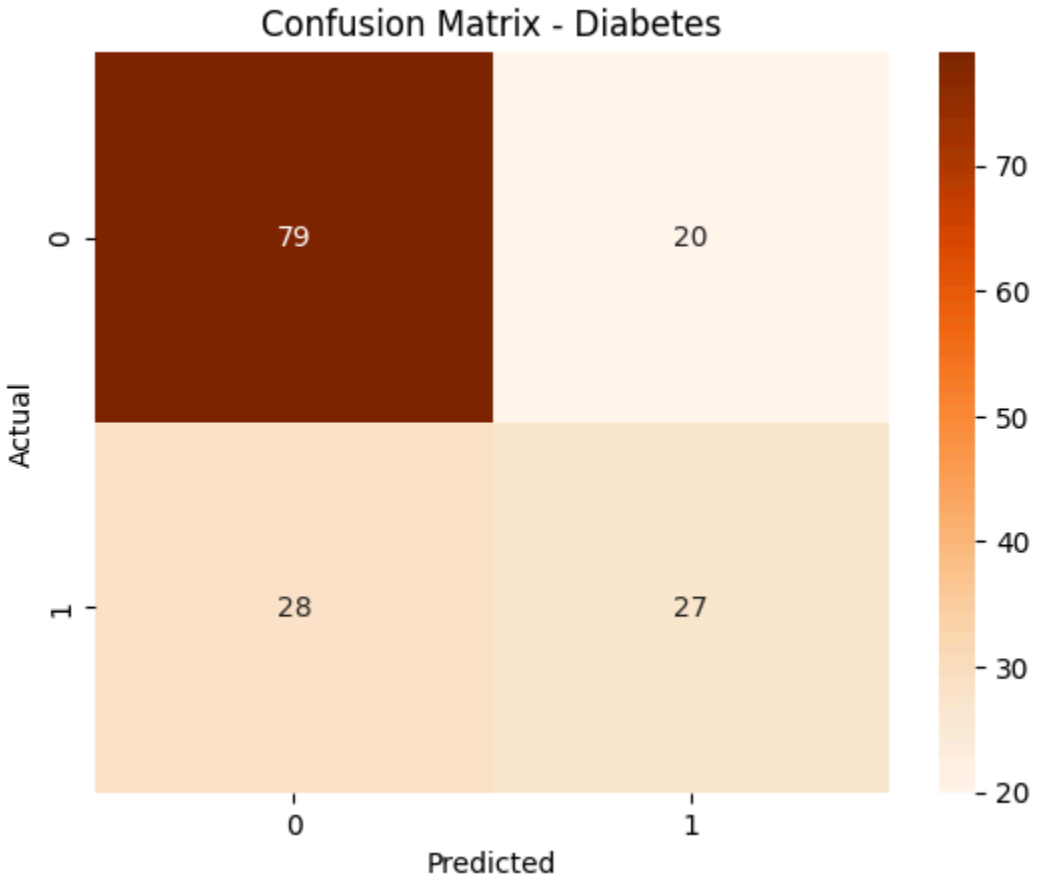
Build a KNN classifier to classify diabetes dataset. Choose k value and perform feature scaling. Use 80% of data for training and 20% for testing. Display accuracy score and confusion matrix of the trained model on test data.

In [ ]:

```
from sklearn.preprocessing import StandardScaler

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

# Features and labels
X = diabetes_df.drop('Outcome', axis=1)
y = diabetes_df['Outcome']

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Initialize and train KNN
knn_diabetes = KNeighborsClassifier(n_neighbors=5)
```

```
knn_diabetes.fit(X_train, y_train)

# Predict
y_pred = knn_diabetes.predict(X_test)

# Evaluation
print("Accuracy Score:", accuracy_score(y_test, y_pred),"\n")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred),"\n")
print("Classification Report:\n", classification_report(y_test, y_pred),"\n")

# Plot confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Oranges", fmt="d")
plt.title("Confusion Matrix - Diabetes")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Accuracy Score: 0.6883116883116883

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

Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.80      0.77        99
           1       0.57      0.49      0.53        55

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



Using heart.csv dataset and do following-

Classify the target using KNN classifier. You can use different values for k neighbors and need to figure out a value of K that gives you a maximum score.

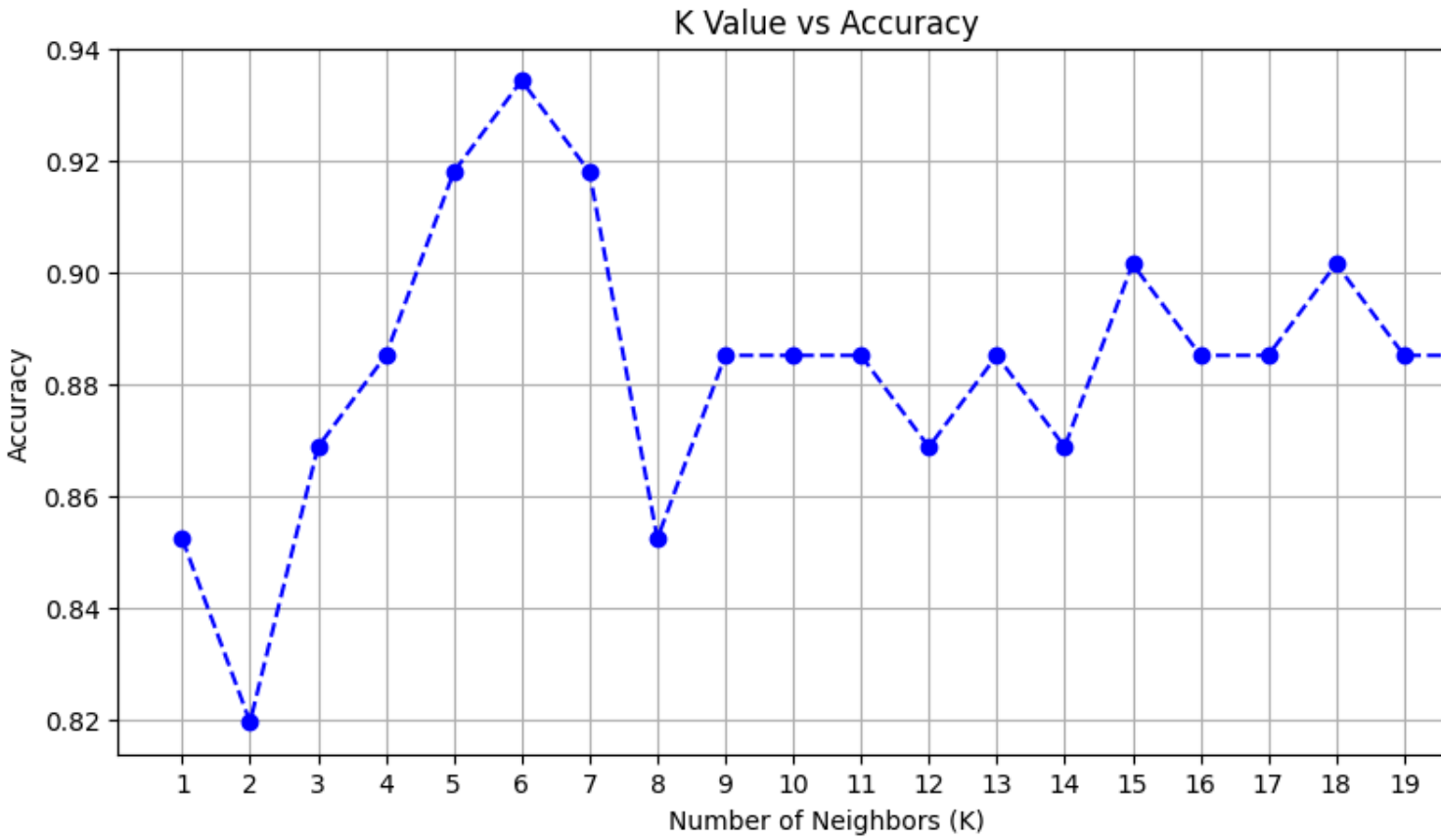
Plot confusion matrix

Plot classification report

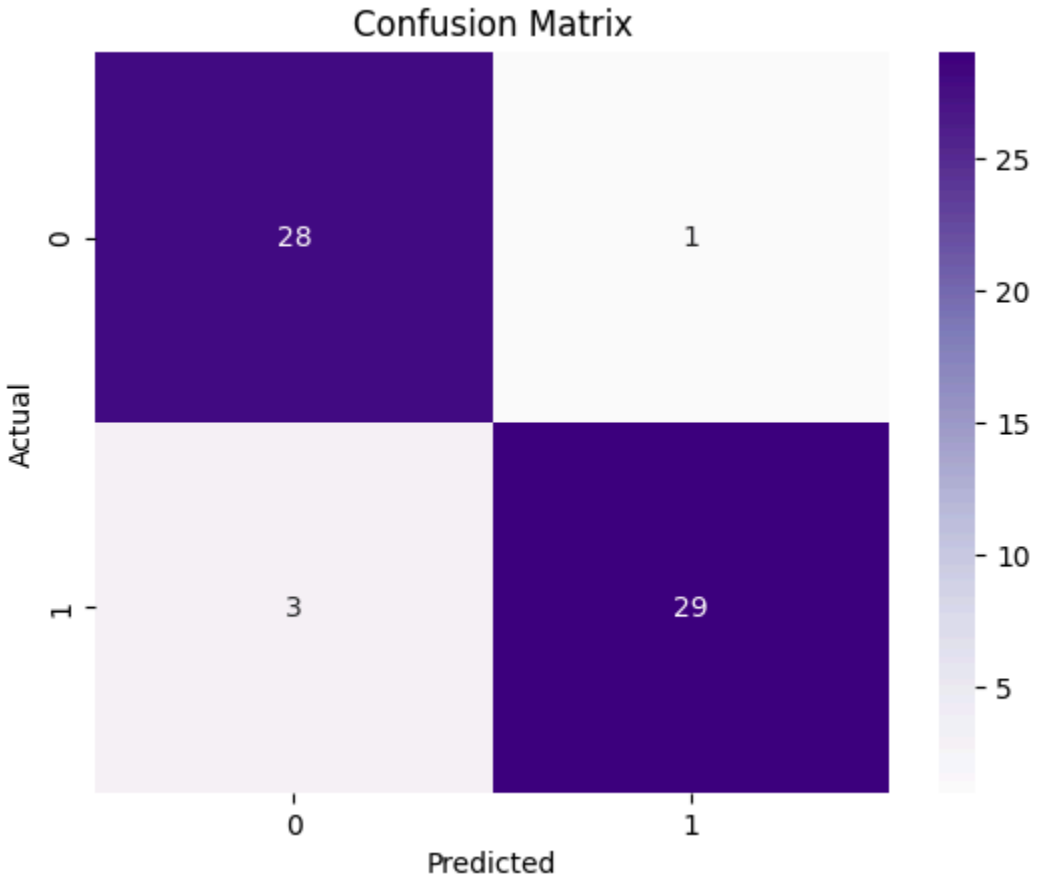
In [ ]:

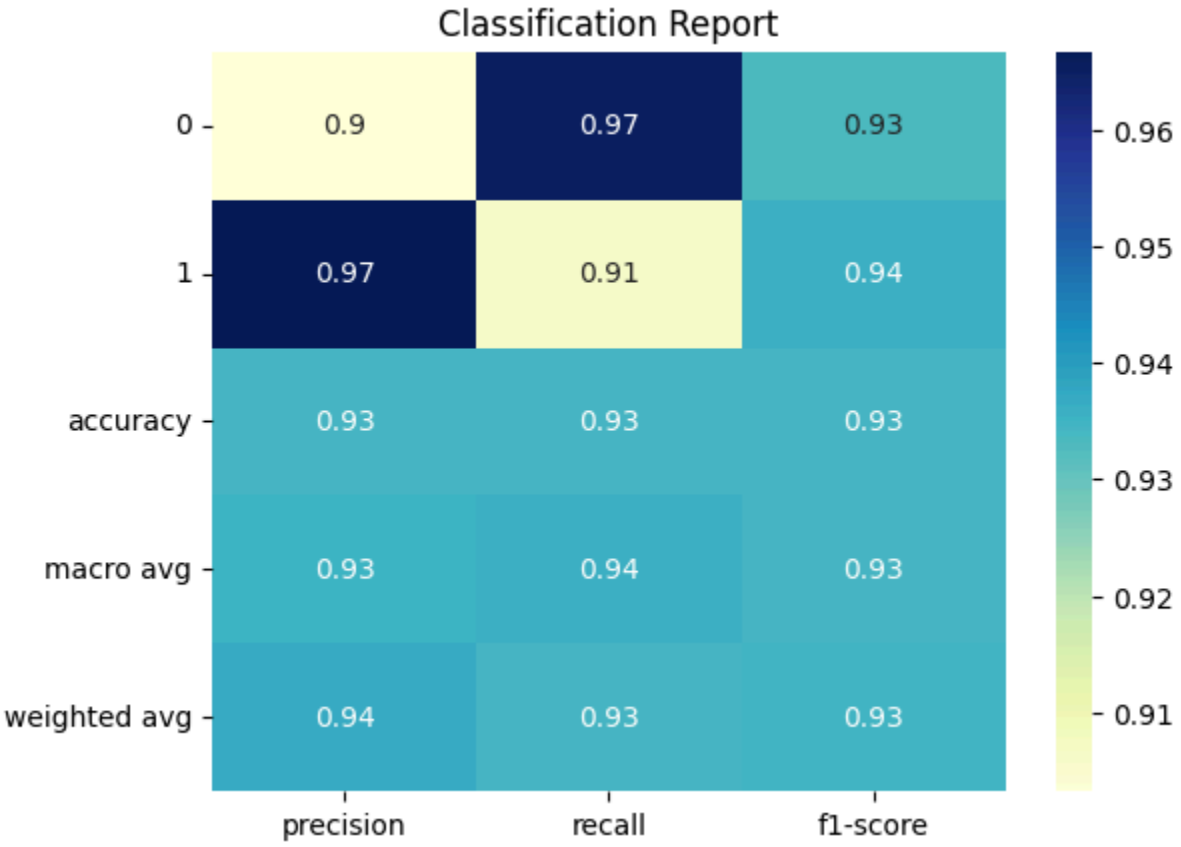
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

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

```
# Features and Target
X = df.drop("target", axis=1)
y = df["target"]

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

# Train-test split (80-20)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Find the best k value
accuracy_scores = []
k_values = range(1, 21)

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred_k)
    accuracy_scores.append(acc)

# Plot accuracy vs k
plt.figure(figsize=(10, 5))
plt.plot(k_values, accuracy_scores, marker='o', linestyle='--', color='b')
plt.title("K Value vs Accuracy")
plt.xlabel("Number of Neighbors (K)")
plt.ylabel("Accuracy")
plt.xticks(k_values)
plt.grid(True)
plt.show()

# Best k value
best_k = k_values[np.argmax(accuracy_scores)]
print(f"\nBest K Value: {best_k} with Accuracy: {max(accuracy_scores):.4f}")

# Final model using best k
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(X_train, y_train)
y_pred = best_knn.predict(X_test)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap="Purples", fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
print("\n")

# Classification Report
report = classification_report(y_test, y_pred, output_dict=True)
sns.heatmap(pd.DataFrame(report).iloc[:-1, :].T, annot=True, cmap="YlGnBu")
plt.title("Classification Report")
plt.show()
```




```
Best K Value: 6 with Accuracy: 0.9344
```

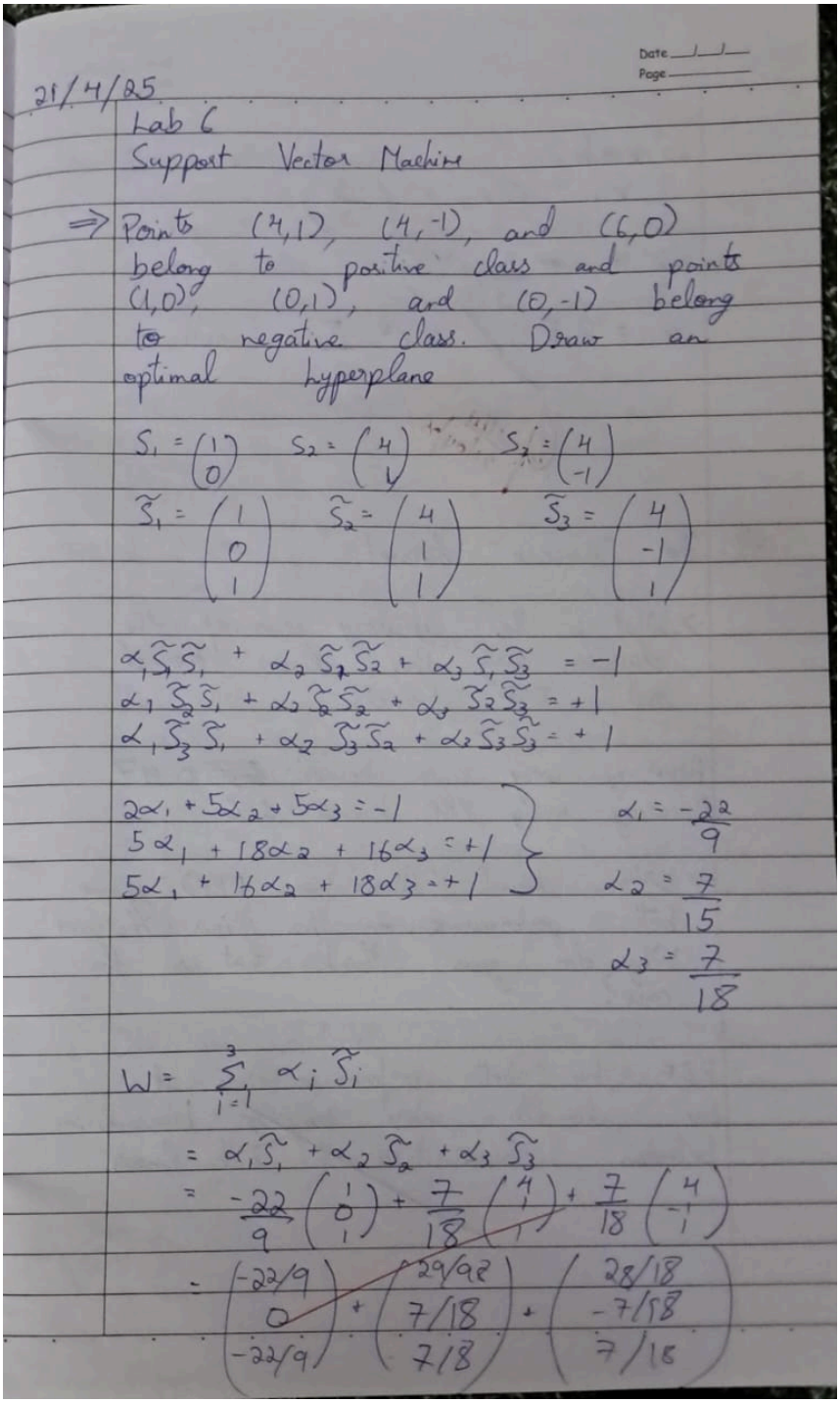

Confusion Matrix
28
1
3
29
0
1
Actual
0
1
Predicted
25
20
15
10
5

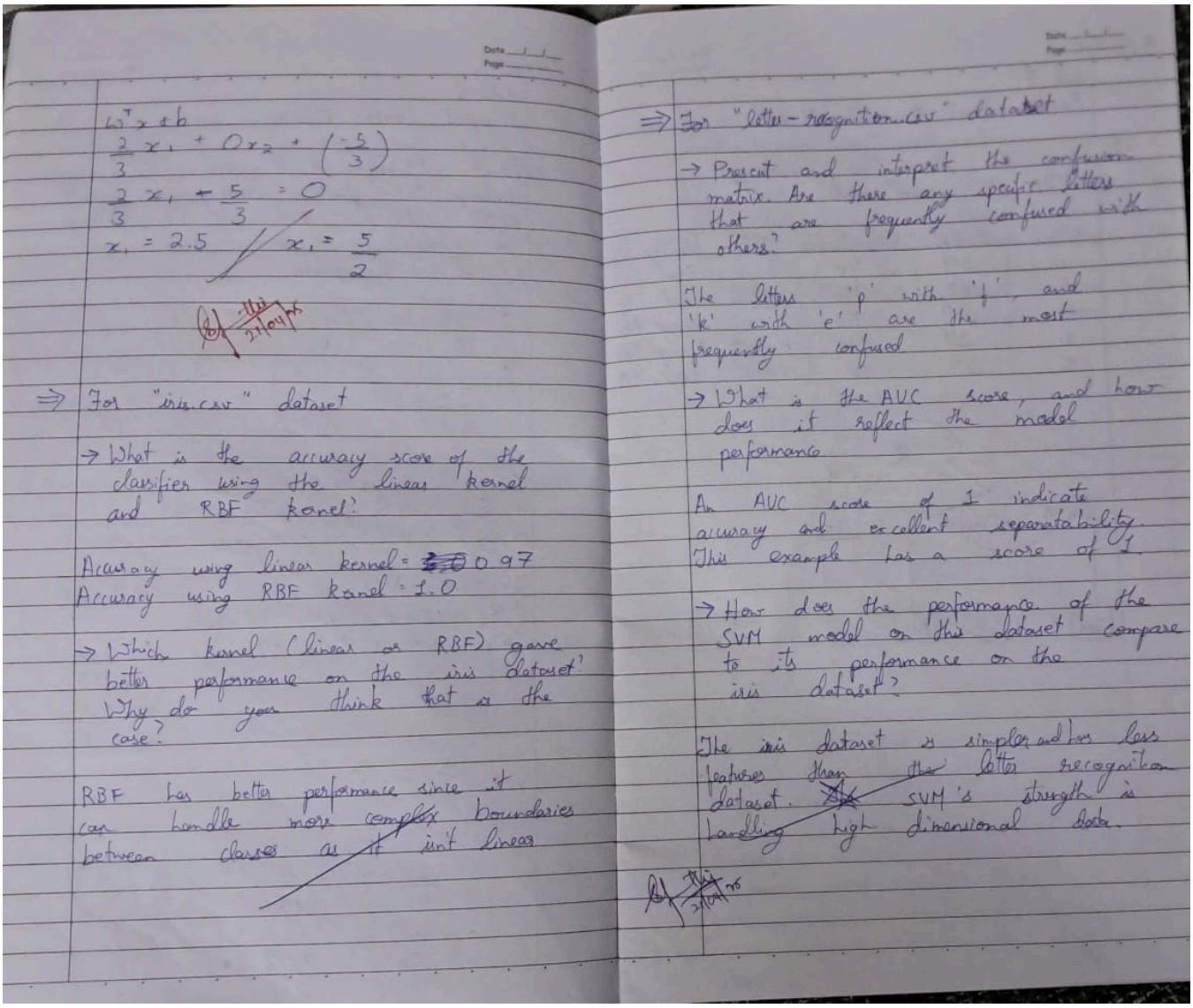

Classification Report
0
1
accuracy
macro avg
weighted avg
0.9
0.97
0.93
0.97
0.91
0.94
0.93
0.93
0.93
0.93
0.94
0.93
0.94
0.93
0.93
precision
recall
f1-score
0.96
0.95
0.94
0.93
0.92
0.91

Program 7

Build Support vector machine model for a given dataset.

Screenshot:





Code:

Lab-6

Build Support Vector Machine model for a given dataset

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 letter-recognition.csv to letter-recognition.csv
```

Build a SVM classifier to classify IRIS flower dataset using the kernels RBF and linear.

Use 80% of data for training and 20% for testing.

Display accuracy score and confusion matrix of the trained model on test data.

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

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

# Encode labels
le = LabelEncoder()
iris['species'] = le.fit_transform(iris['species'])

# Split data
X = iris.drop('species', axis=1)
y = iris['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Linear Kernel SVM
svc_linear = SVC(kernel='linear')
svc_linear.fit(X_train, y_train)
y_pred_linear = svc_linear.predict(X_test)

print("Linear Kernel:")
print("Accuracy:", accuracy_score(y_test, y_pred_linear))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_linear))

# RBF Kernel SVM
svc_rbf = SVC(kernel='rbf')
svc_rbf.fit(X_train, y_train)
y_pred_rbf = svc_rbf.predict(X_test)

print("\nRBF Kernel:")
print("Accuracy:", accuracy_score(y_test, y_pred_rbf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rbf))
```

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

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

Build a SVM classifier to classify Letter-recognition data set

Use 80% of data for training and 20% for testing.

Display accuracy score and confusion matrix of the trained model on test data. Plot the ROC curve and display AUC score.

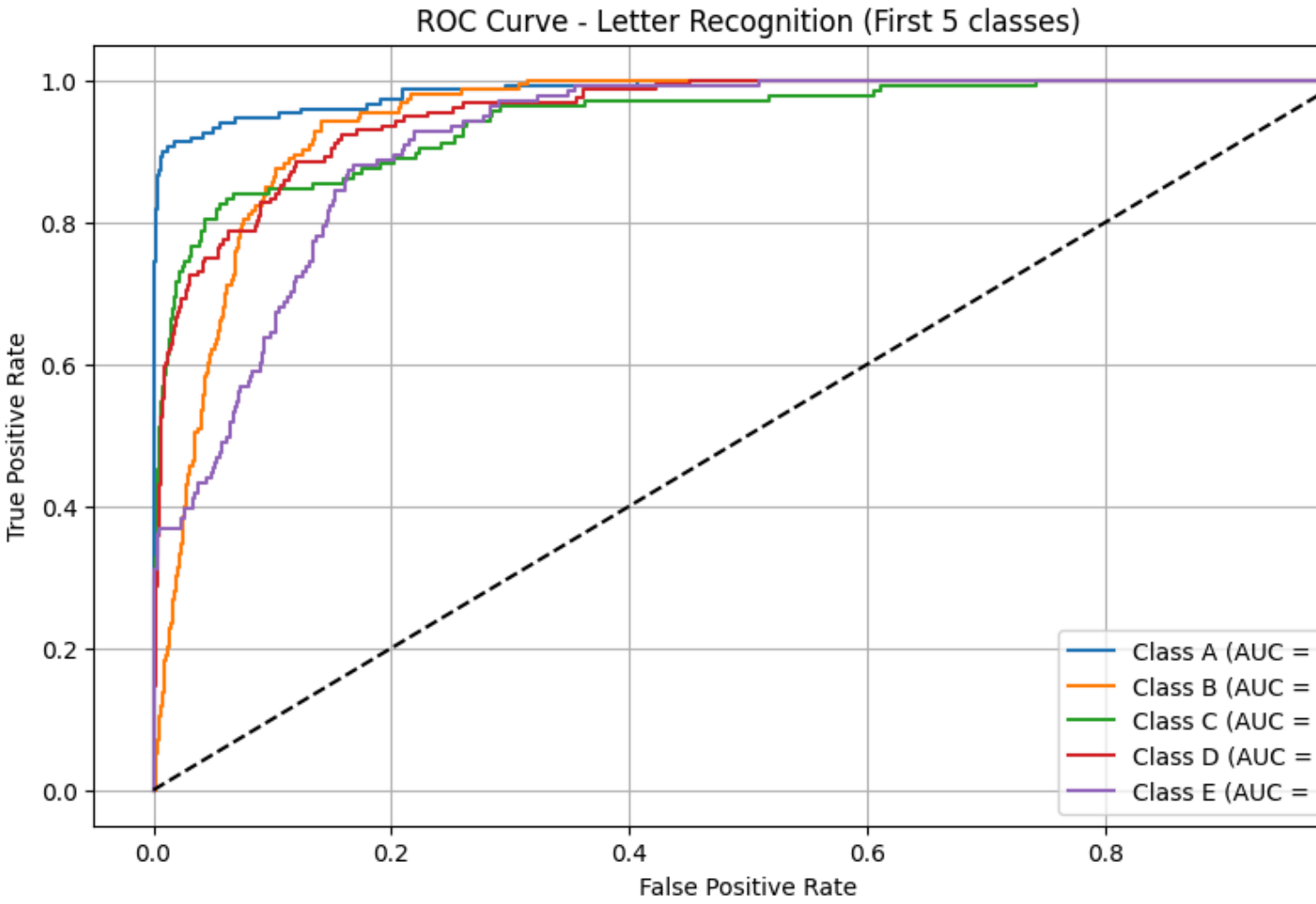
In [ ]:

```
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier

# Load Letter-recognition dataset
letter = pd.read_csv("letter-recognition.csv")

# Assume first column is the label, rest are features
X = letter.iloc[:, 1:]
y = letter.iloc[:, 0]

# Binarize labels for ROC
lb = LabelBinarizer()
y_bin = lb.fit_transform(y)

# Train-Test Split
X_train, X_test, y_train_bin, y_test_bin = train_test_split(X, y_bin, test_size=0.2,
random_state=42)

# One-vs-Rest SVM with linear kernel
classifier = OneVsRestClassifier(SVC(kernel='linear', probability=True))
classifier.fit(X_train, y_train_bin)
y_score = classifier.decision_function(X_test)

# Accuracy and Confusion Matrix (for multiclass)
y_pred_class = classifier.predict(X_test)
y_pred_labels = lb.inverse_transform(y_pred_class)
y_true_labels = lb.inverse_transform(y_test_bin)

print("Accuracy:", accuracy_score(y_true_labels, y_pred_labels))
print("Confusion Matrix:\n", confusion_matrix(y_true_labels, y_pred_labels))

# Compute ROC and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
```

```
for i in range(len(lb.classes_)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plotting ROC curve for a few classes (for clarity)
plt.figure(figsize=(10, 6))
for i in range(min(5, len(lb.classes_))):  # Plot first 5 classes
            plt.plot(fpr[i],   tpr[i],   label=f'Class   {lb.classes_[i]}   (AUC   =
{roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Letter Recognition (First 5 classes)')
plt.legend()
plt.grid()
plt.show()
```

```
Accuracy: 0.36825
Confusion Matrix:
 [[149   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [149   0   0   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [ 68   0  69   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [ 88   0   0  68   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [124   0  17   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [106   0   0   1   0  31   0   0   0   0   0   0   0   0   0   1   0   0
    0   1   0   0   0   0   0   0]
 [157   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2
    0   0   0   0   0   0   0   0]
 [140   0   0   2   0   0   0   0   0   0   0   0   0   0   0   1   0   0
    0   0   1   0   0   0   0   0]
 [ 48   0   0   0   0   0   0   0  95   0   0   0   0   0   0   1   0   0
    0   0   0   0   0   0   2   0]
 [ 35   0   0   0   0   1   0   0   7 103   0   0   0   0   0   3   0   0
    0   0   0   0   0   0   0   0]
 [122   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   8
    0   0   0   0   0   0   0   0]
 [ 39   0   0   5   0   0   0   0   1   0   0 110   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [ 29   0   0   5   0   0   0   0   0   0   0   0 131   0   0   0   0   0
    0   1   0   0   2   0   0   0]
 [143   0   0   2   0   0   0   0   0   0   0   0   1   0   0   1   0   0
    0   4   0   0   0   0   0   0]
 [144   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   1   0   0   0]
 [ 49   0   0   1   0   7   0   0   0   0   0   0   0   0   0 116   0   0
    0   0   0   0   0   0   0   0]
 [164   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   2]
 [112   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0  47
    0   0   0   0   0   0   0   0]
 [154   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   2
```

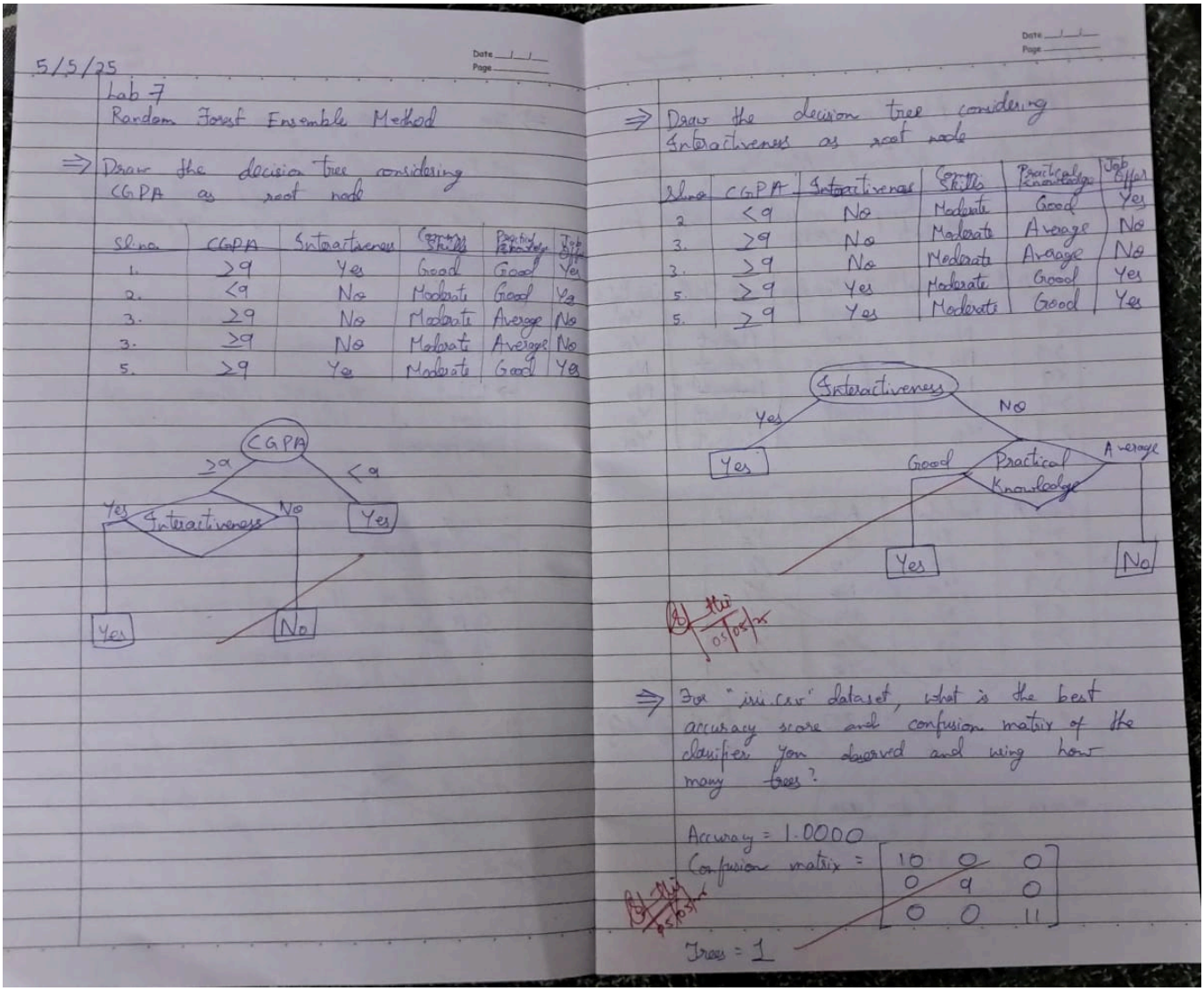
```
   0   1   0   0   0   0   7   6]
[ 40   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   1   0   0
   0 120   0   0   0   0   1   0]
[103   0   1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   1
   0   0  77   0   0   0   0   0]
[ 81   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
   0   0   0  73   1   0   2   0]
[ 12   0   0   1   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0
   0   0   0   3 128   0   0   0]
[152   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   1   1]
[ 50   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   9   0  33   0   0  76   0]
[ 45   0   0   1   0   1   0   0   0   3   0   0   0   0   0   0   0   0   0
   0   1   0   0   0   0   1  80]]
```



Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot:



Code:

Lab-7

Implement Random forest ensemble method on a given dataset.

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.csv
```

To Do: Implementation – Random Forest (for Classification)

Write Python code to implement the following. Consider dataset files as "iris.csv"

Build a Random Forest (RF) classifier to classify IRIS flower dataset

Measure prediction score using default n_estimators (10).

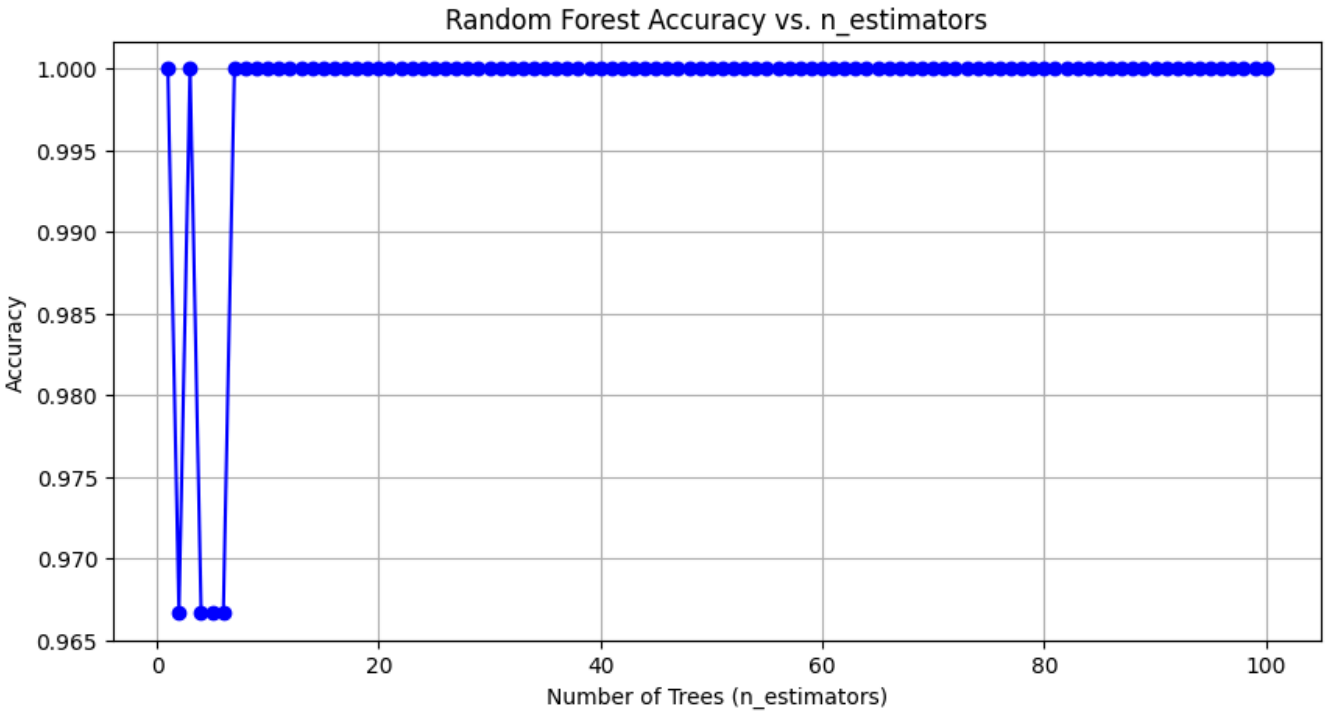
Now fine tune your model by changing number of trees in your classifer and identify what best score you can get using how many trees

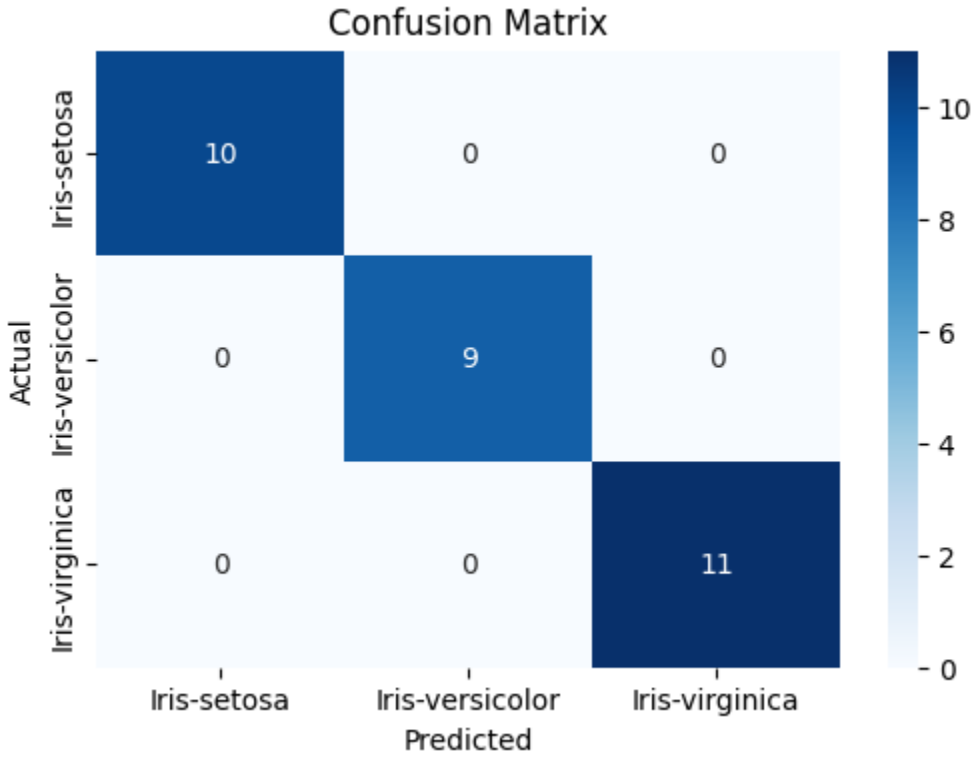
In [4]:

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

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

# Separate features and target
X = data.drop("species", axis=1)
y = data["species"]

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

# Default Random Forest (n_estimators=10)
rf_default = RandomForestClassifier(random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default RF Accuracy (n_estimators=10): {default_score:.4f}")

# Fine-tuning: Test different n_estimators values
scores = {}
for n in range(1, 101):  # from 1 to 100 trees
```

```
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    scores[n] = acc

# Find the best n_estimators value
best_n = max(scores, key=scores.get)
best_score = scores[best_n]
print(f"Best Accuracy: {best_score:.4f} with n_estimators={best_n}")

# Train and predict with best model
rf_best = RandomForestClassifier(n_estimators=best_n, random_state=42)
rf_best.fit(X_train, y_train)
y_pred_best = rf_best.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_best)
print("Confusion Matrix:\n", cm)

# Plotting accuracy vs n_estimators
plt.figure(figsize=(10, 5))
plt.plot(list(scores.keys()), list(scores.values()), marker='o', color='blue')
plt.title("Random Forest Accuracy vs. n_estimators")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()

# Plotting confusion matrix as heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=rf_best.classes_, yticklabels=rf_best.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Default RF Accuracy (n_estimators=10): 1.0000
Best Accuracy: 1.0000 with n_estimators=1
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

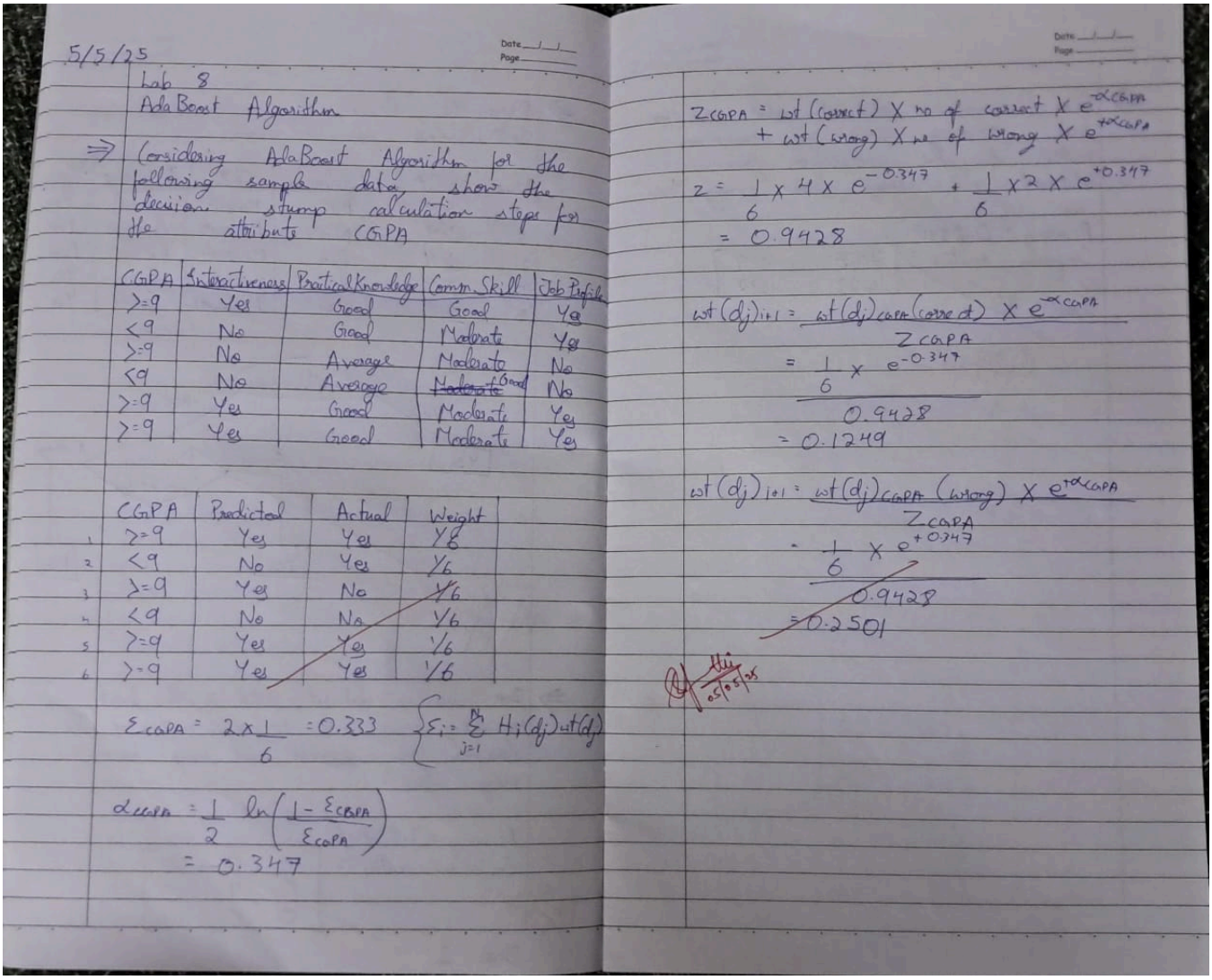

Random Forest Accuracy vs. n_estimators
Accuracy
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0
20
40
60
80
100
Number of Trees (n_estimators)


Confusion Matrix
Actual
Iris-setosa
Iris-versicolor
Iris-virginica
10
0
0
0
9
0
0
0
11
10
8
6
4
2
0
Iris-setosa
Iris-versicolor
Iris-virginica
Predicted

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot:



5/5/25

Lab 8

Ada Boost Algorithm

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

CGPA	Interactiveness	Practical Knowledge	Comm. Skill	Job Profile
>=9	Yes	Good	Good	Yes
<9	No	Good	Moderate	Yes
>=9	No	Average	Moderate	No
<9	No	Average	~~Moderate~~ Good	No
>=9	Yes	Good	Moderate	Yes
>=9	Yes	Good	Moderate	Yes

	CGPA	Predicted	Actual	Weight
1	>=9	Yes	Yes	1/6
2	<9	No	Yes	1/6
3	>=9	Yes	No	1/6
4	<9	No	No	1/6
5	>=9	Yes	Yes	1/6
6	>=9	Yes	Yes	1/6

$\varepsilon_{CGPA} = 2 \times \frac{1}{6} = 0.333$ $\left\{\varepsilon_i = \sum_{j=1}^{n} H_i(d_j) \cdot wt(d_j)\right.$

$\alpha_{CGPA} = \frac{1}{2} \ln\left(\frac{1-\varepsilon_{CGPA}}{\varepsilon_{CGPA}}\right)$

$= 0.347$

$Z_{CGPA} = wt(correct) \times no\ of\ correct \times e^{-\alpha_{CGPA}} + wt(wrong) \times no\ of\ wrong \times e^{+\alpha_{CGPA}}$

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

$= 0.9428$

$wt(d_j)_{i+1} = \frac{wt(d_j)_{CGPA}(correct) \times e^{-\alpha_{CGPA}}}{Z_{CGPA}}$

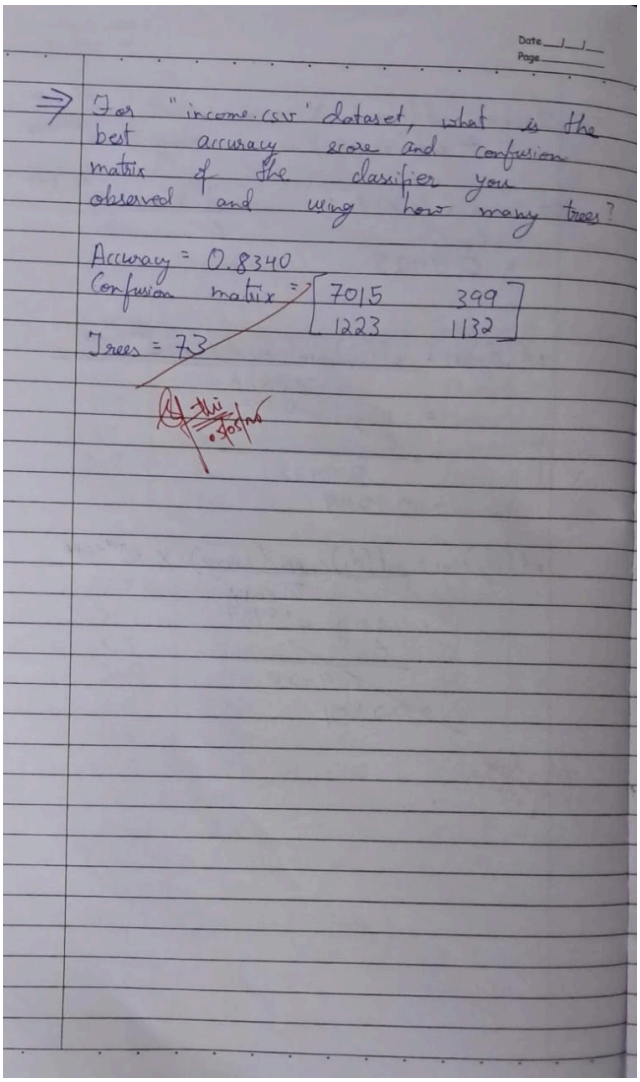
$= \frac{\frac{1}{6} \times e^{-0.347}}{0.9428}$

$= 0.1249$

$wt(d_j)_{i+1} = \frac{wt(d_j)_{CGPA}(wrong) \times e^{+\alpha_{CGPA}}}{Z_{CGPA}}$

$= \frac{\frac{1}{6} \times e^{+0.347}}{0.9428}$

$= 0.2501$



Code:

Lab-8

Implement Boosting ensemble method on a given dataset.

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 income.csv to income.csv
```

To Do: Implementation – AdaBoost

Write Python code to implement the following. Consider dataset files as "income.csv"

Build a AdaBoost classifier to classify Income dataset

Measure prediction score using n_estimators (10).

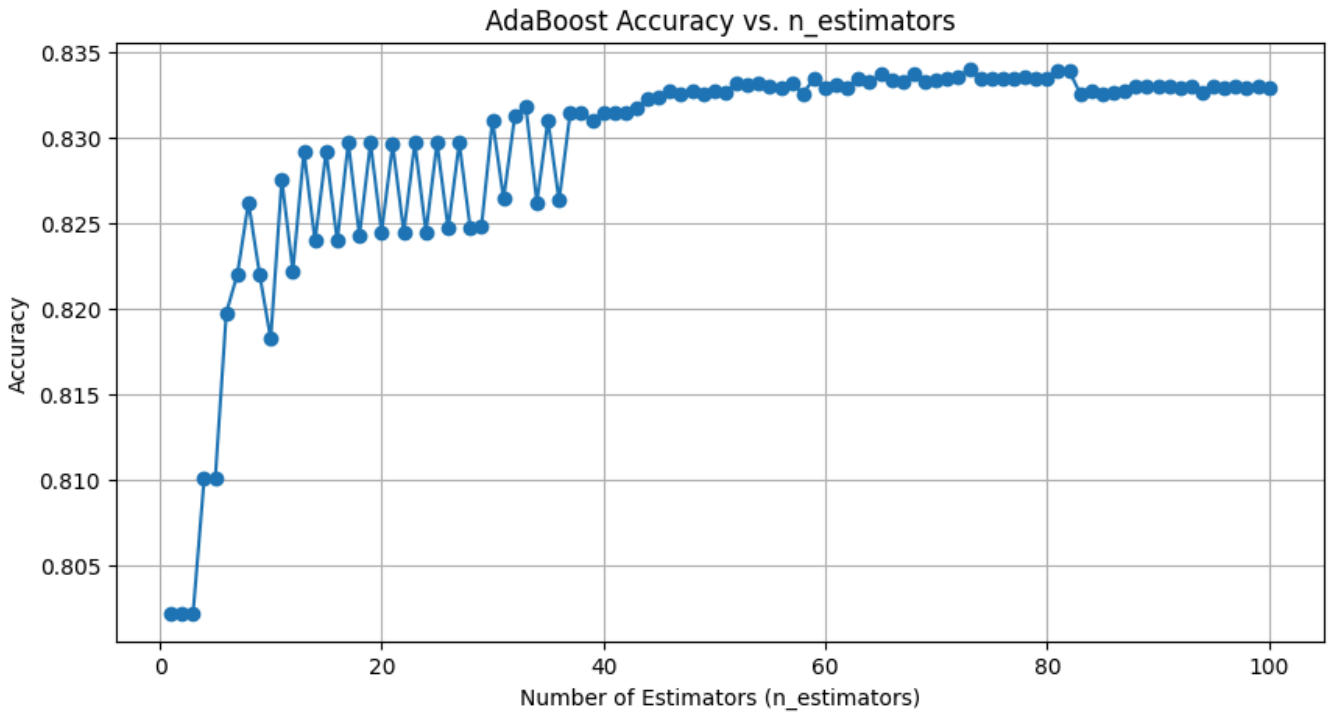
Now fine tune your model by changing number of trees in your classifier and identify what best score you can get using how many trees

In [ ]:

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

# Load the dataset
data = pd.read_csv("income.csv")

# Split into features and target
X = data.drop("income_level", axis=1)
y = data["income_level"]

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

# Default AdaBoost with n_estimators=10
ada_default = AdaBoostClassifier(n_estimators=10, random_state=42)
ada_default.fit(X_train, y_train)
y_pred_default = ada_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default AdaBoost Accuracy (n_estimators=10): {default_score:.4f}")

# Fine-tuning: Try different numbers of estimators
scores = {}
for n in range(1, 101):
    ada = AdaBoostClassifier(n_estimators=n, random_state=42)
    ada.fit(X_train, y_train)
    y_pred = ada.predict(X_test)
    scores[n] = accuracy_score(y_test, y_pred)

# Find best n_estimators
best_n = max(scores, key=scores.get)
best_score = scores[best_n]
print(f"Best Accuracy: {best_score:.4f} with n_estimators={best_n}")

# Plot
plt.figure(figsize=(10, 5))
```

```
plt.plot(list(scores.keys()), list(scores.values()), marker='o')
plt.title("AdaBoost Accuracy vs. n_estimators")
plt.xlabel("Number of Estimators (n_estimators)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

```
Default AdaBoost Accuracy (n_estimators=10): 0.8182
Best Accuracy: 0.8340 with n_estimators=73
```

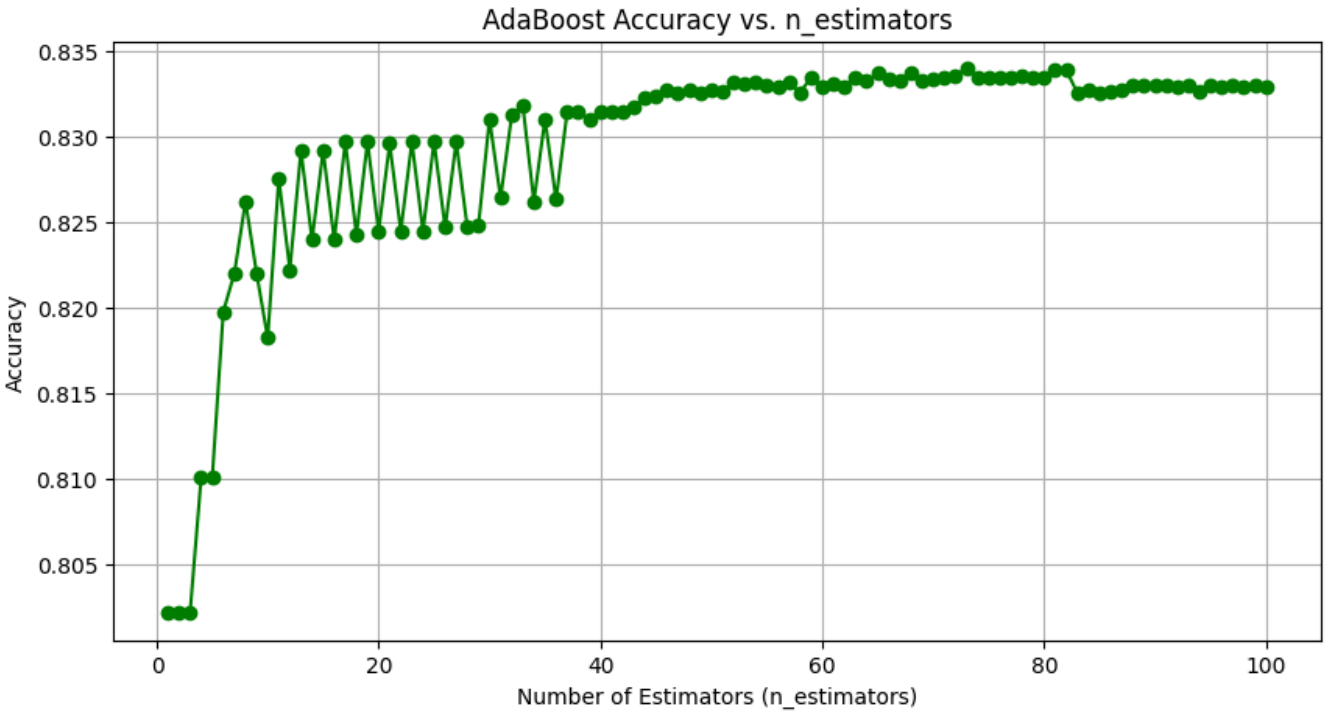


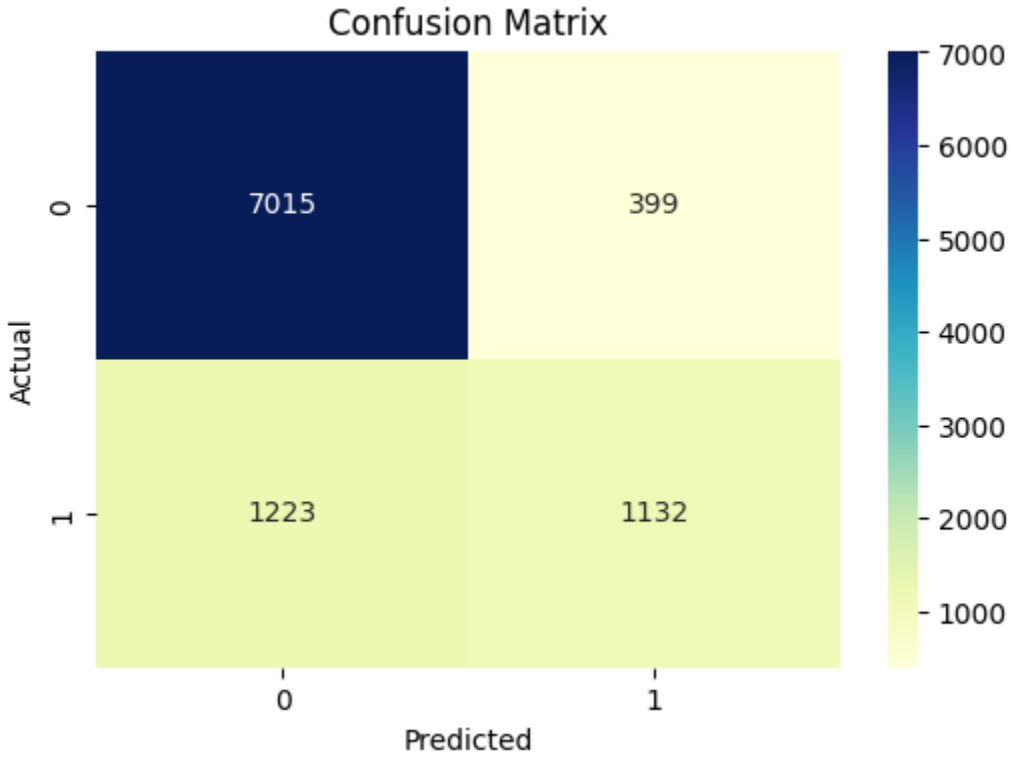
In [4]:

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

# Load the dataset
data = pd.read_csv("income.csv")

# Split into features and target
X = data.drop("income_level", axis=1)
y = data["income_level"]

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

```
# Default AdaBoost with n_estimators=10
ada_default = AdaBoostClassifier(n_estimators=10, random_state=42)
ada_default.fit(X_train, y_train)
y_pred_default = ada_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default AdaBoost Accuracy (n_estimators=10): {default_score:.4f}")

# Fine-tuning: Try different numbers of estimators
scores = {}
for n in range(1, 101):
    ada = AdaBoostClassifier(n_estimators=n, random_state=42)
    ada.fit(X_train, y_train)
    y_pred = ada.predict(X_test)
    scores[n] = accuracy_score(y_test, y_pred)

# Find best n_estimators
best_n = max(scores, key=scores.get)
best_score = scores[best_n]
print(f"Best Accuracy: {best_score:.4f} with n_estimators={best_n}")

# Train best model and generate confusion matrix
ada_best = AdaBoostClassifier(n_estimators=best_n, random_state=42)
ada_best.fit(X_train, y_train)
y_pred_best = ada_best.predict(X_test)

cm = confusion_matrix(y_test, y_pred_best)
print("Confusion Matrix:\n", cm)

# Plotting accuracy vs n_estimators
plt.figure(figsize=(10, 5))
plt.plot(list(scores.keys()), list(scores.values()), marker='o', color='green')
plt.title("AdaBoost Accuracy vs. n_estimators")
plt.xlabel("Number of Estimators (n_estimators)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()

# Confusion matrix heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="YlGnBu",
            xticklabels=ada_best.classes_, yticklabels=ada_best.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Default AdaBoost Accuracy (n_estimators=10): 0.8182
Best Accuracy: 0.8340 with n_estimators=73
Confusion Matrix:
 [[7015  399]
 [1223 1132]]
```

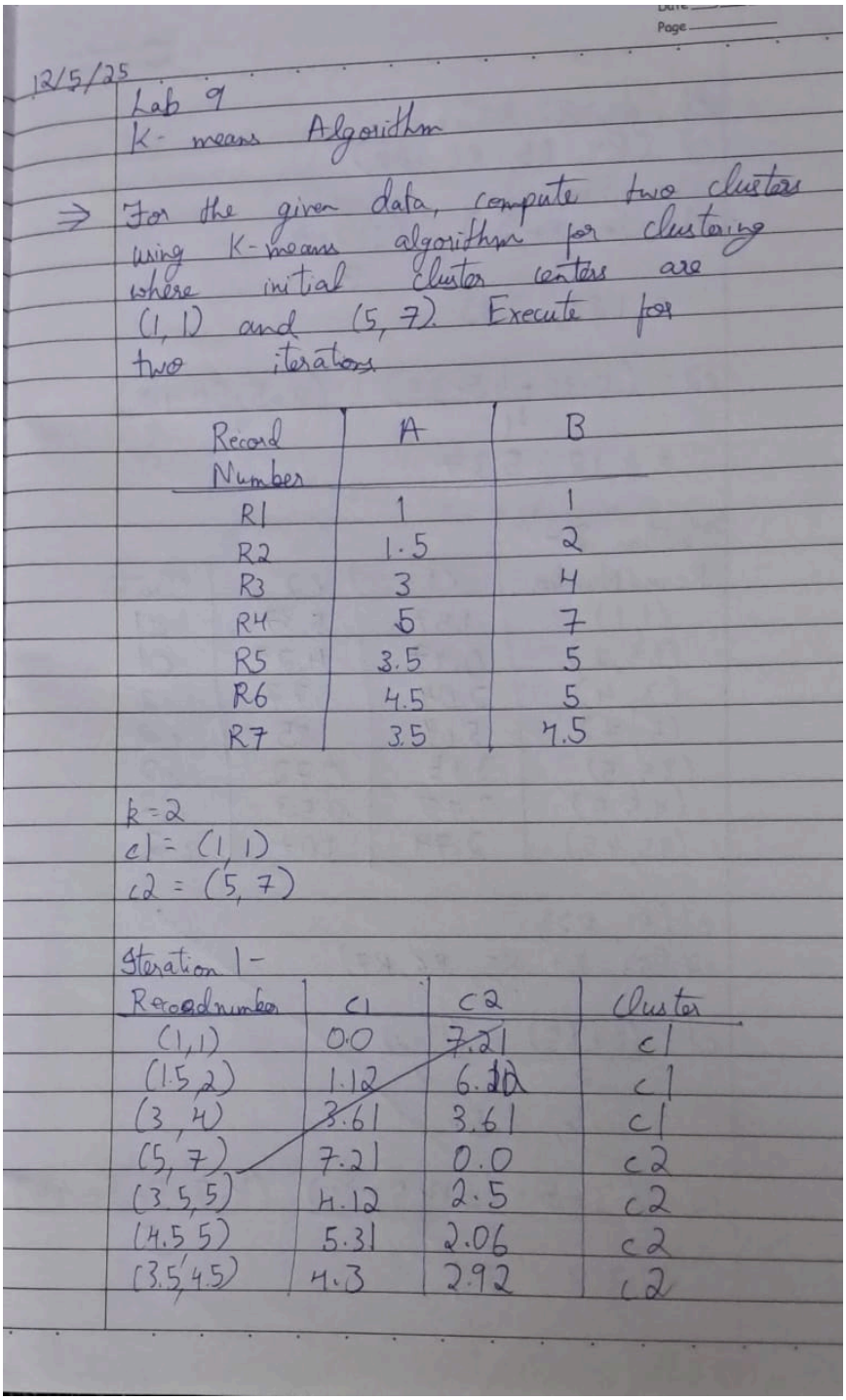

AdaBoost Accuracy vs. n_estimators
Accuracy
0.835
0.830
0.825
0.820
0.815
0.810
0.805
0
20
40
60
80
100
Number of Estimators (n_estimators)


Confusion Matrix
7015
399
1223
1132
Actual
0
1
Predicted
0
1
7000
6000
5000
4000
3000
2000
1000

Program 10

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

Screenshot:.



12/5/25

Lab 9

K- means Algorithm

⇒ For the given data, compute two clusters using K-means algorithm for clustering where initial cluster centers are (1, 1) and (5, 7). Execute for two iterations

Record Number	A	B
R1	1	1
R2	1.5	2
R3	3	4
R4	5	7
R5	3.5	5
R6	4.5	5
R7	3.5	4.5

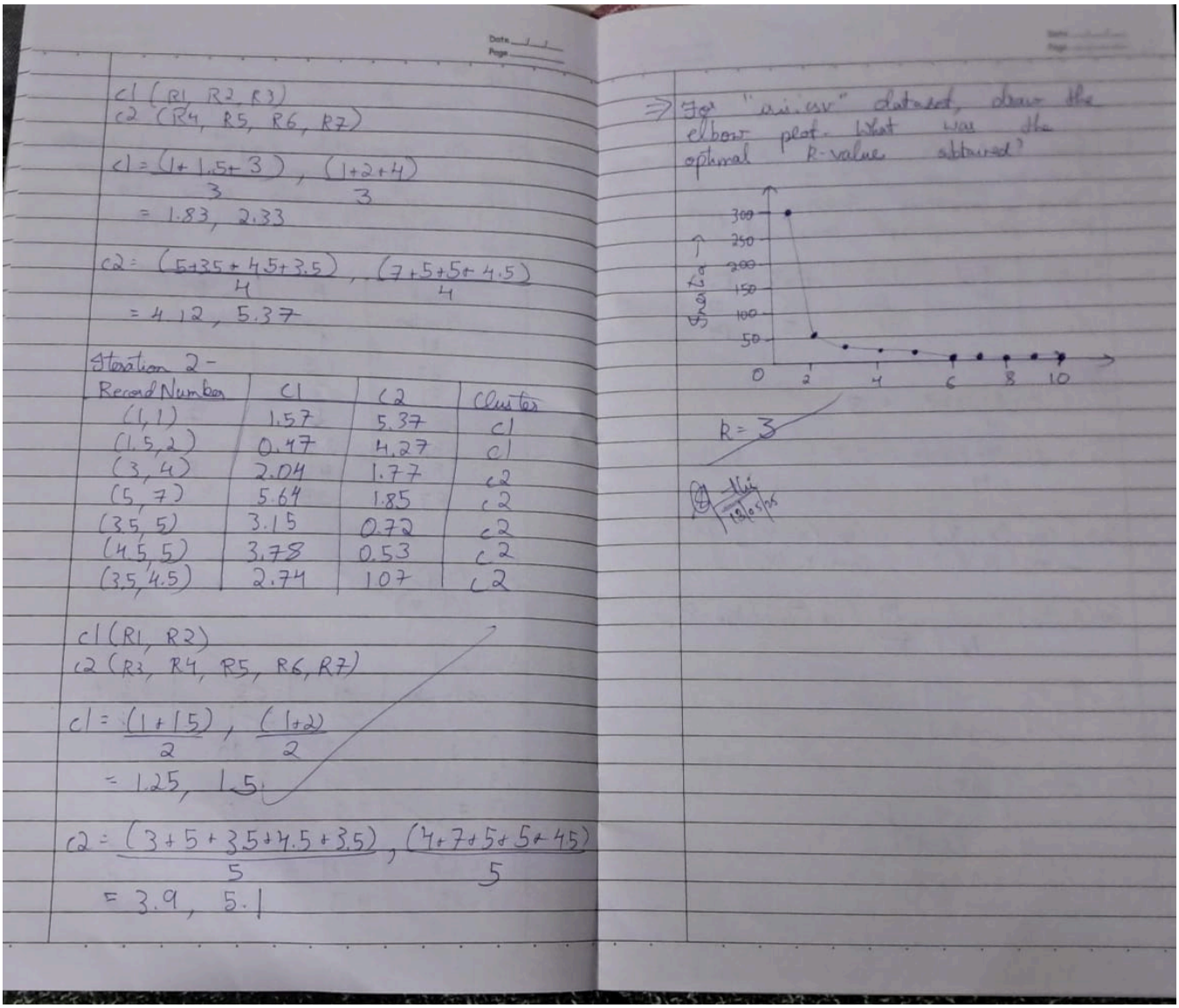
k = 2

c1 = (1, 1)

c2 = (5, 7)

Iteration 1 -

Record number	c1	c2	Cluster
(1,1)	0.0	7.21	c1
(1.5,2)	1.12	6.10	c1
(3, 4)	3.61	3.61	c1
(5, 7)	7.21	0.0	c2
(3.5, 5)	4.12	2.5	c2
(4.5 5)	5.31	2.06	c2
(3.5, 4.5)	4.3	2.92	c2



Code:

Lab-9

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

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.csv
```

To Do: Implementation – K-Means Clustering

Write Python code to implement the following. Consider dataset files as “iris.csv”

Build a K-Means Clustering algorithm to cluster IRIS flower dataset

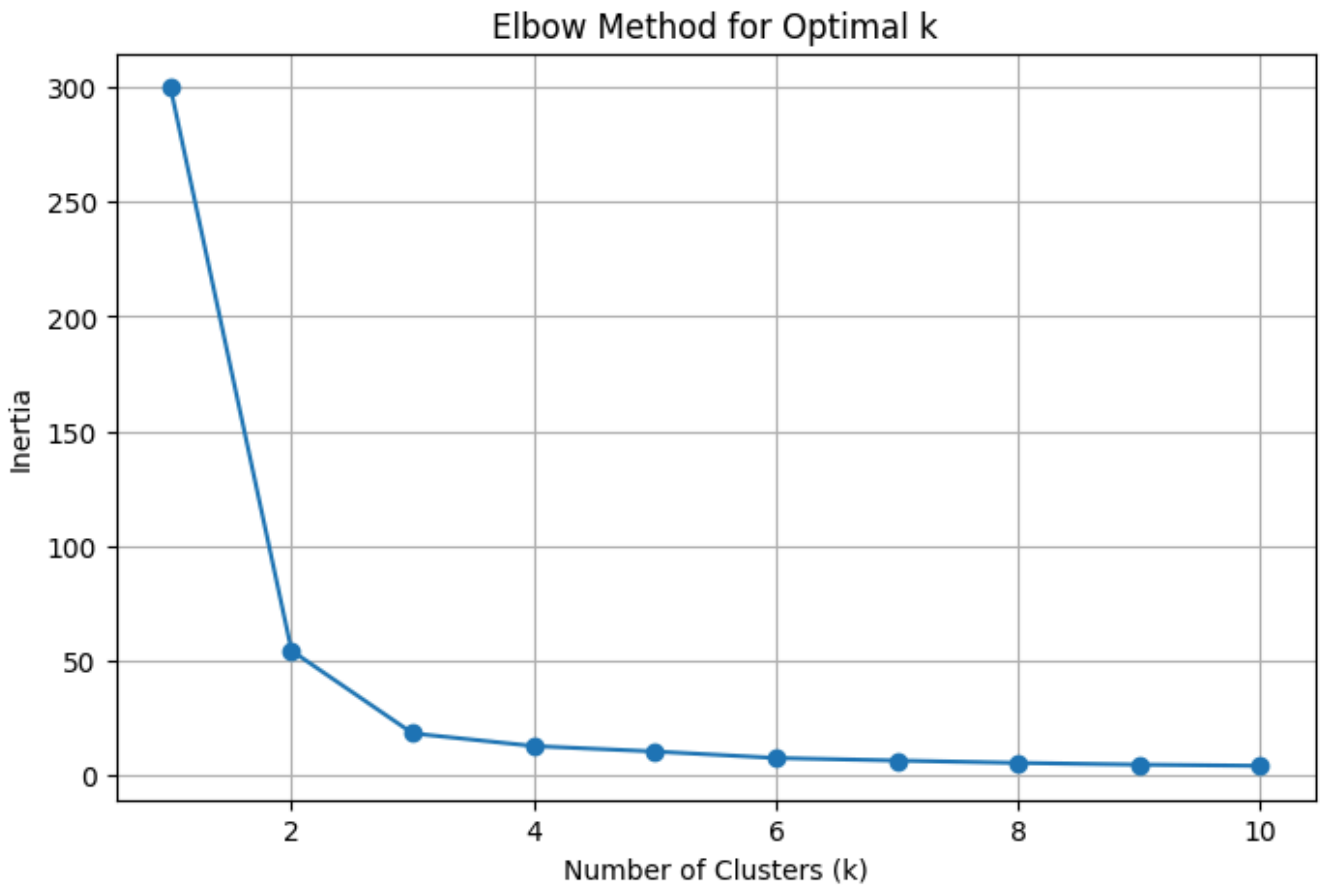
Use iris flower dataset to form clusters of flowers using petal width and length features. Drop other two features for simplicity.

Figure out if any preprocessing such as scaling would help here

Draw elbow plot and from that figure out optimal value of k

In [ ]:

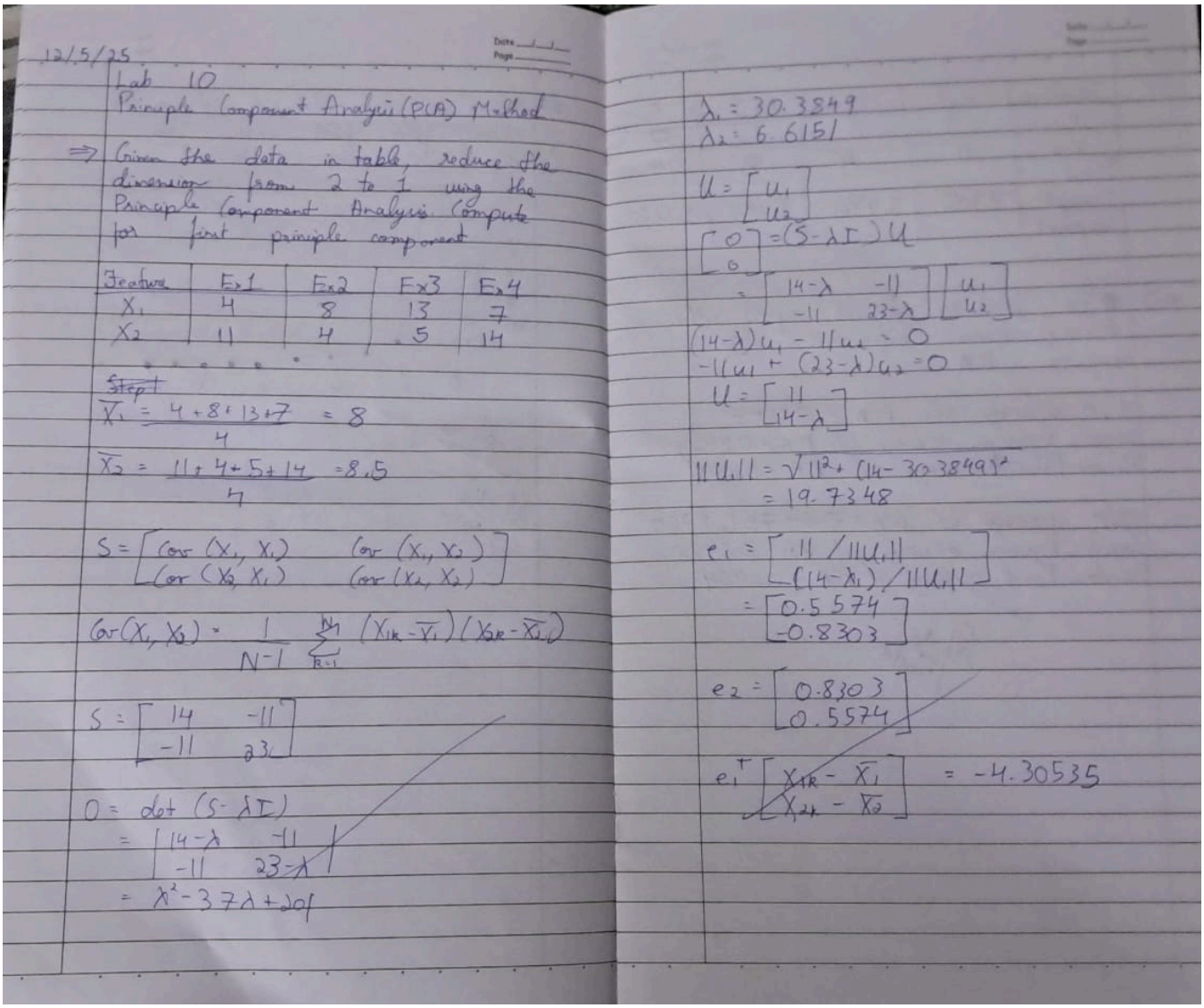
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

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

# Select only petal length and width
X = data[["petal_length", "petal_width"]]

# Check if scaling helps (KMeans is sensitive to scale)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Elbow method: Try k from 1 to 10 and compute inertia
inertias = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

# Plot elbow graph
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertias, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.show()

#k=3 from the graph, k where it bends like an elbow
```

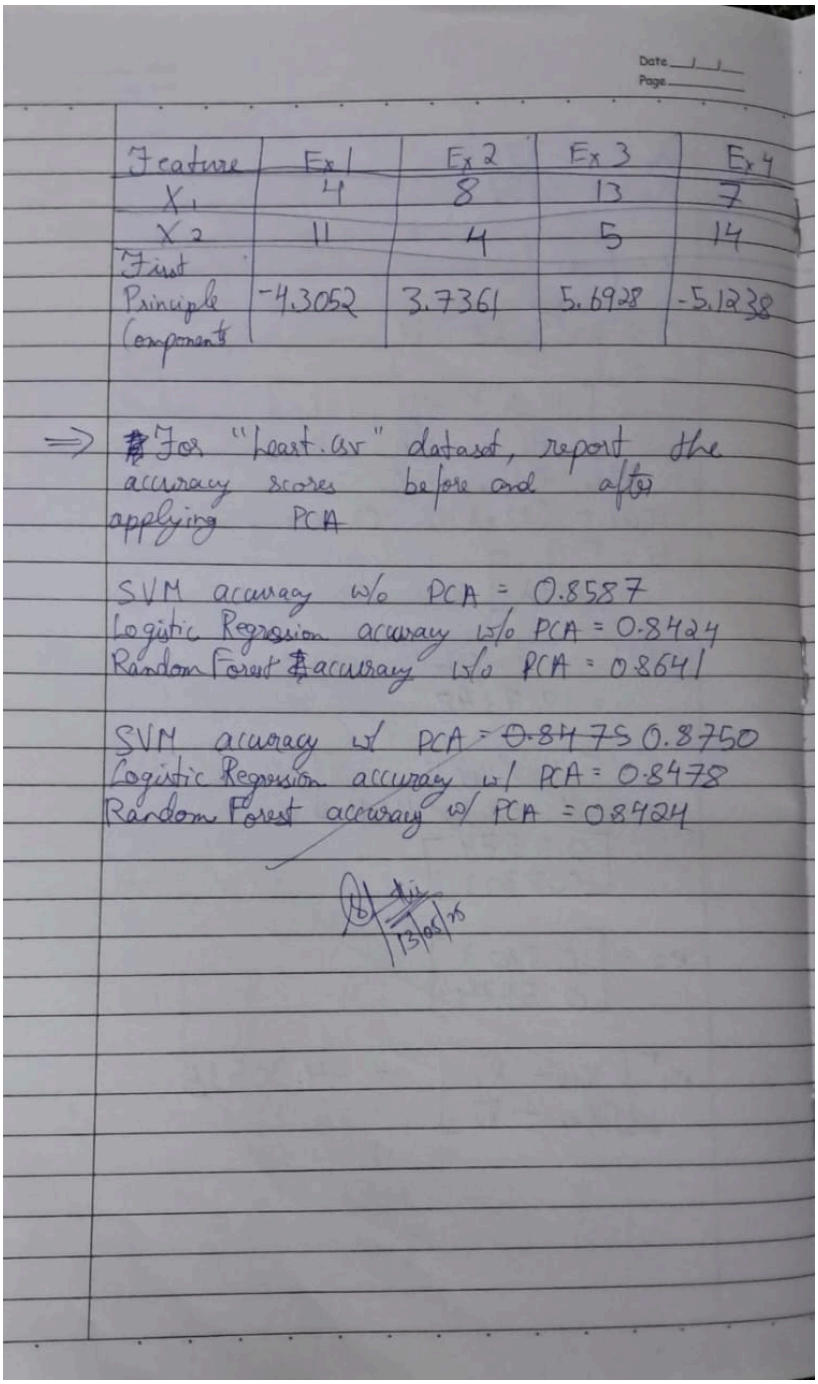

Elbow Method for Optimal k
Inertia
300
250
200
150
100
50
0
2
4
6
8
10
Number of Clusters (k)

Program 11

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

Screenshot:





Feature	Ex 1	Ex 2	Ex 3	Ex 4
X_1	4	8	13	7
X_2	11	4	5	14
First Principle Component	-4.3052	3.7361	5.6928	-5.1238

⇒ For "heart.csv" dataset, report the accuracy scores before and after applying PCA

SVM accuracy w/o PCA = 0.8587
Logistic Regression accuracy w/o PCA = 0.8424
Random Forest accuracy w/o PCA = 0.8641

SVM accuracy w/ PCA = ~~0.8475~~ 0.8750
Logistic Regression accuracy w/ PCA = 0.8478
Random Forest accuracy w/ PCA = 0.8424

Code:

Lab-10

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

In [1]:

```
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
```

To Do: Implementation – PCA

Write Python code to implement the following. Consider dataset files as "heart.csv"

Convert text columns to numbers using label encoding and one hot encoding

Apply scaling

Build a classification model using various methods (SVM, logistic regression, random forest) and check which model gives you the best accuracy

Now use PCA to reduce dimensions, retrain your model and see what impact it has on your model in terms of accuracy. Keep in mind that many times doing PCA reduces the accuracy but computation is much lighter and that's the trade off you need to consider while building models in real life.

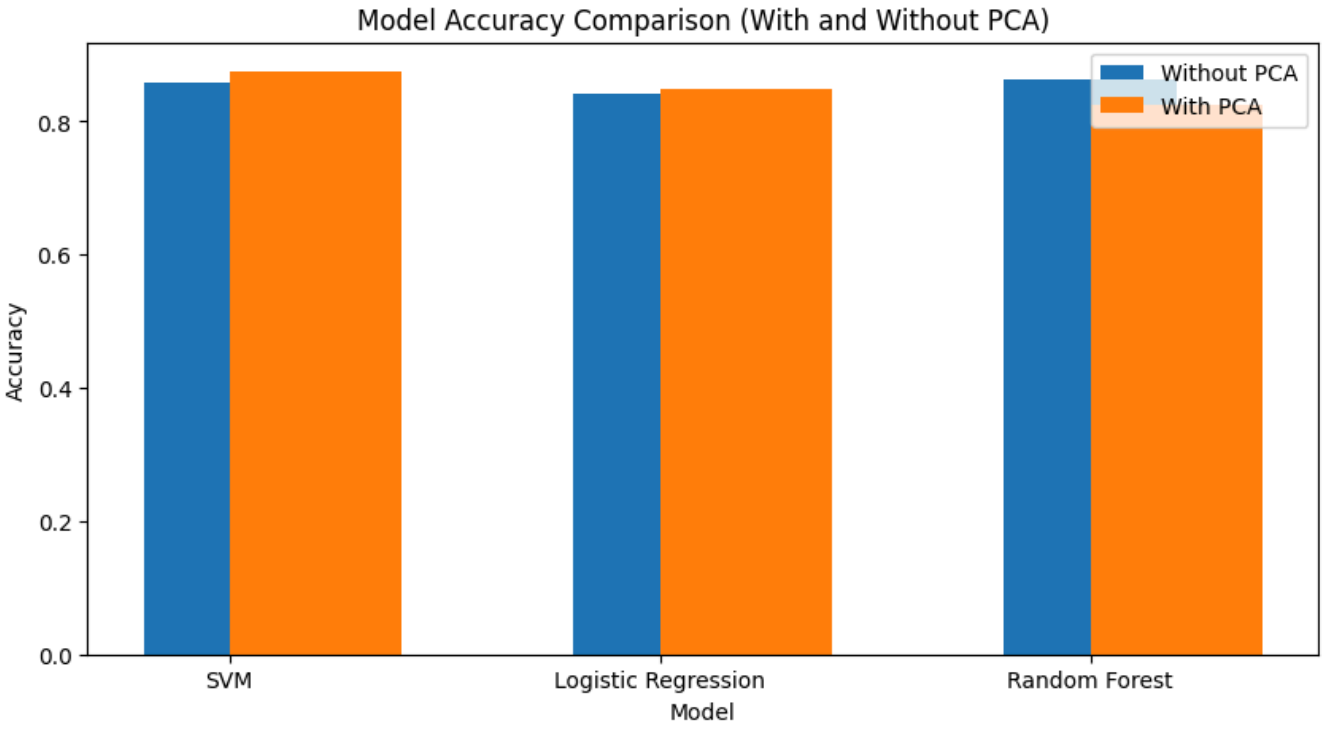
In [4]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
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
import matplotlib.pyplot as plt

# Load the dataset (replace with your own file path)
data = pd.read_csv("heart.csv")

# Display first few rows to understand the dataset structure
print(data.head())

# Encode categorical columns using Label Encoding
label_encoder = LabelEncoder()

# Label Encoding for 'Sex', 'RestingECG', 'ExerciseAngina', and 'ST_Slope'
data['Sex'] = label_encoder.fit_transform(data['Sex'])
data['RestingECG'] = label_encoder.fit_transform(data['RestingECG'])
data['ExerciseAngina'] = label_encoder.fit_transform(data['ExerciseAngina'])
```

```
data['ST_Slope'] = label_encoder.fit_transform(data['ST_Slope'])

# One Hot Encoding for 'ChestPainType' (if necessary, based on dataset)
data = pd.get_dummies(data, columns=['ChestPainType'], drop_first=True)

# Split data into features and target
X = data.drop("HeartDisease", axis=1)  # Features
y = data["HeartDisease"]  # Target

# Train-test split (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

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

# Build and evaluate the models: SVM, Logistic Regression, and Random Forest
models = {
    "SVM": SVC(),
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier()
}

# Train and evaluate models without PCA
for model_name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model_name} Accuracy without PCA: {accuracy:.4f}")

# Apply PCA for dimensionality reduction
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Train and evaluate models with PCA
for model_name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model_name} Accuracy with PCA: {accuracy:.4f}")

# Plotting the accuracy comparison (without PCA vs with PCA)
accuracies_without_pca = []
accuracies_with_pca = []

for model_name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracies_without_pca.append(accuracy_score(y_test, y_pred))

    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    accuracies_with_pca.append(accuracy_score(y_test, y_pred))
```

```
# Bar plot comparison
labels = list(models.keys())
x = range(len(models))

plt.figure(figsize=(10, 5))
plt.bar(x, accuracies_without_pca, width=0.4, label='Without PCA', align='center')
plt.bar(x, accuracies_with_pca, width=0.4, label='With PCA', align='edge')
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison (With and Without PCA)")
plt.xticks(x, labels)
plt.legend()
plt.show()
```

```
  Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
SVM Accuracy without PCA: 0.8587
Logistic Regression Accuracy without PCA: 0.8424
Random Forest Accuracy without PCA: 0.8641
SVM Accuracy with PCA: 0.8750
Logistic Regression Accuracy with PCA: 0.8478
Random Forest Accuracy with PCA: 0.8424
```


Model Accuracy Comparison (With and Without PCA)
Without PCA
With PCA
Accuracy
0.0
0.2
0.4
0.6
0.8
SVM
Logistic Regression
Random Forest
Model