

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum- 590014, Karnataka.



LAB REPORT on **Machine Learning (23CS6PCMAL)**

Submitted by

Sanjana Shetty (1BM22CS238)

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
February 2025 – July 25

B.M.S. College of Engineering

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by **Sanjana Shetty (1BM22CS238)**, who is bonafide student of **B.M.S. College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Laboratory report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Saritha A. N
Assistant Professor
Department of CSE, BMSCE

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

INDEX

Sl. No.	Date	Experiment Title	Page No.
1	03.03.25	Write a python program to import and export data using pandas library functions.	1
2	10.03.25	Demonstrate various data pre-processing techniques for a given dataset.	6
3	24.03.25	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	12
4	17.03.25	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset.	17
5	24.03.25	Build Logistic Regression Model for a given dataset.	23
6	07.04.25	Build KNN Classification model for a given dataset.	28
7	21.04.25	Build Support vector machine model for a given dataset.	32
8	05.05.25	Implement Random forest ensemble method on a given dataset.	37
9	05.05.25	Implement Boosting ensemble method on a given dataset.	40
10	05.05.25	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	47
11	05.05.25	Implement Dimensionality reduction using Principle Component Analysis (PCA) method.	52

Github Link: <https://github.com/sanjanashettybmsce/6thSem-ML-Lab>

LABORATORY PROGRAM – 1

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

OBSERVATION BOOK

5/3/25 Lab 0		Method 3									
<pre>from google.colab import drive drive.mount('/content/drive')</pre>		<pre>bmi bp s1 s2 s3 s4 s5 s6 target 0 0.061696 0.021872 -0.019 -0.039821 -0.043491 -0.00259 -0.019307 -0.017616 151.0</pre>									
<u>Method 1:</u> Initializing values directly into DataFrame		<u>Method 3</u>									
<pre>import pandas as pd data = { 'USN': ['IBS17CS001', 'IBS17CS002', 'IBS17CS003'], 'Name': ['Alice', 'Bob', 'Charlie'], 'Marks': [85, 78, 92] } df = pd.DataFrame(data)</pre>		<pre>import pandas as pd file_path = '/content/drive/My Drive/Lab-0/samples-sales.csv' df = pd.read_csv(file_path) df.head()</pre>									
<u>Output:</u>		<u>Output:</u>									
<pre>0 IBS17CS001 Alice 85.0 1 IBS17CS002 Bob 78.0 2 IBS17CS003 Charlie 92.0</pre>		<pre>Product Quantity Price Sales Region 0 Laptop 5 1000 5000 North 1 Mouse 15 20 300 West</pre>									
<u>Method 2:</u>		<u>Method 4</u>									
<pre>import pandas as pd from sklearn.datasets import load_diabetes diabetes_data = load_diabetes() df = pd.DataFrame(diabetes_data.data, columns=diabetes_data.feature_names) df['target'] = diabetes_data.target df.head()</pre>		<pre># STOCK MARKET DATA ANALYSIS import yfinance as yf import matplotlib.pyplot as plt tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"] data = yf.download(tickers, start="2024-01-01", end="2024-12-31", group_by='ticker') data.head()</pre>									
<u>Output:</u>		<u>Summary statistics for a specific stock (3 banks - HDFC, ICICI, KOTAK):</u>									
<pre>age sex bmi bp st</pre>		<pre>hdfc = data['HDFCBANK.NS'] print(hdfc.describe())</pre>									

similarly for ICICI, KOTAK

icici = data['ICICI BANK.NS']

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

Plotting closing price and daily returns:

plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)

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

plt.subplot(2, 1, 2)

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

plt.tight_layout()

plt.show()

Similarly, for Kotak and ICICI

CODE WITH OUTPUT

Diabetes Dataset

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

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N

```
df.shape
```

```
(1000, 14)
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --    
 0   ID          1000 non-null   int64  
 1   No_Pation   1000 non-null   int64  
 2   Gender      1000 non-null   object 
 3   AGE         1000 non-null   int64  
 4   Urea        1000 non-null   float64 
 5   Cr          1000 non-null   int64  
 6   HbA1c       1000 non-null   float64 
 7   Chol        1000 non-null   float64 
 8   TG          1000 non-null   float64 
 9   HDL         1000 non-null   float64 
 10  LDL         1000 non-null   float64 
 11  VLDL        1000 non-null   float64 
 12  BMI         1000 non-null   float64 
 13  CLASS        1000 non-null   object 
dtypes: float64(8), int64(4), object(2)
memory usage: 109.5+ KB
None
```

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

	ID	No_Pation	AGE	Urea	Cr	\
count	1000.000000	1.000000e+03	1000.000000	1000.000000	1000.000000	
mean	340.500000	2.705514e+05	53.528000	5.124743	68.943000	
std	240.397673	3.380758e+06	8.799241	2.935165	59.984747	
min	1.000000	1.230000e+02	20.000000	0.500000	6.000000	
25%	125.750000	2.406375e+04	51.000000	3.700000	48.000000	
50%	300.500000	3.439550e+04	55.000000	4.600000	60.000000	
75%	550.250000	4.538425e+04	59.000000	5.700000	73.000000	
max	800.000000	7.543566e+07	79.000000	38.900000	800.000000	

	HbA1c	Chol	TG	HDL	LDL	\
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	
mean	8.281160	4.862820	2.349610	1.204750	2.609790	
std	2.534003	1.301738	1.401176	0.660414	1.115102	
min	0.900000	0.000000	0.300000	0.200000	0.300000	
25%	6.500000	4.000000	1.500000	0.900000	1.800000	
50%	8.000000	4.800000	2.000000	1.100000	2.500000	
75%	10.200000	5.600000	2.900000	1.300000	3.300000	
max	16.000000	10.300000	13.800000	9.900000	9.900000	

	VLDL	BMI
count	1000.000000	1000.000000
mean	1.854700	29.578020
std	3.663599	4.962388
min	0.100000	19.000000
25%	0.700000	26.000000
50%	0.900000	30.000000
75%	1.500000	33.000000
max	35.000000	47.750000

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

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

```
categorical_cols = df.select_dtypes(include=['object']).columns  
print("Categorical columns identified:", categorical_cols)  
if len(categorical_cols) > 0:  
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)  
    print("\nDataFrame after one-hot encoding:")  
    print(df.head())  
else:  
    print("\nNo categorical columns found in the dataset.")
```

```
Categorical columns identified: Index(['Gender', 'CLASS'], dtype='object')
```

```
DataFrame after one-hot encoding:
```

	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	

	Gender_M	Gender_f	CLASS_N	CLASS_P	CLASS_Y	CLASS_Z
0	False	False	False	False	False	False
1	True	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	True	False	False	False	False	False

```

from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd

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

scaler = MinMaxScaler()
df_minmax = df.copy() # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df[numerical_cols])

scaler = StandardScaler()
df_standard = df.copy()
df_standard[numerical_cols] = scaler.fit_transform(df[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())

```

DataFrame after Min-Max Scaling:

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

TG	HDL	LDL	VLDL	BMI	Gender_M	Gender_f
0	0.044444	0.226804	0.114583	0.011461	0.173913	False
1	0.081481	0.092784	0.187500	0.014327	0.139130	True
2	0.044444	0.226804	0.114583	0.011461	0.173913	False
3	0.044444	0.226804	0.114583	0.011461	0.173913	False
4	0.051852	0.061856	0.177083	0.008596	0.069565	True

CLASS_N	CLASS_P	CLASS_Y	CLASS_Y
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False

DataFrame after Standardization:

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

TG	HDL	LDL	VLDL	BMI	Gender_M	Gender_f
0	-1.035084	1.810756	-1.085457	-0.369958	-1.124622	False
1	-0.678063	-0.158692	-0.457398	-0.342649	-1.326239	True
2	-1.035084	1.810756	-1.085457	-0.369958	-1.124622	False
3	-1.035084	1.810756	-1.085457	-0.369958	-1.124622	False
4	-0.963680	-0.613180	-0.547121	-0.397267	-1.729472	True

CLASS_N	CLASS_P	CLASS_Y	CLASS_Y
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False

LABORATORY PROGRAM – 2

Demonstrate various data pre-processing techniques for a given dataset

OBSERVATION BOOK

5/03/25

Lab 1

```
import pandas as pd
# Load .csv file
df = pd.read_csv("housing.csv")
# display column information
print(df.info())
# display statistical information of all numerical columns
print(df.describe())
# display count of unique labels for "Ocean Proximity"
if "ocean_proximity" in df.columns:
    print(df["ocean_proximity"].value_counts())
else:
    print("not found")
# display columns with count of missing values > 0
missing_values = df.isnull().sum()
columns_with_missing = missing_values[missing_values > 0]
print(columns_with_missing)
Sum  
06.13.2021
```

To-DO: Diabetes dataset

```
① df = pd.read_csv("diabetes.csv")
② Missing values:
    val = df.isnull().sum()
    df: series [], dtype: int64
③ Handling missing values:
    num_col = df.select_dtypes(include=['float64', 'int64']).columns
    df[num_col] = df[num_col].fillna(0)
    imputer = SimpleImputer(strategy='mean')
    df[num_col] = imputer.fit_transform(df[num_col])
④ Handling Categorical Attributes
    df1 = df.copy()
    df1['Gender'] = df1['Gender'].str.upper()
    ordinal_encoder = OrdinalEncoder(categories=[['M', 'F']])
    df1['Gender-encoded'] = ordinal_encoder.fit_transform(df1[['Gender']])
    onehot_encoder = OneHotEncoder()
    encoded_data = onehot_encoder.fit_transform(df1[['CLASS']])
    encoded_array = encoded_data.toarray()
    encoded_df = pd.DataFrame(encoded_array,
    columns=onehot_encoder.get_feature_names_out(['CLASS']))
```

`df_encoded = pd.concat([df1, encoded_df], axis=1)`
`df_encoded = df_encoded.drop(['Gender', 'Class'], axis=1, inplace=True)`

⑤ Data Normalization

norm = Min Max Scale()

df-encoded [('Urea')] = normalised fit-transform
(df-encoded [('Urea')])

scals = StandardScaler()

`df['AGE'].fit_transform(df['AGE'])`

⑥ Removing Outliers

$$df_1 = df_encoded$$

$R1 = df1['Urea'].quantile(0.25)$

`Q3 = df1['urea'].quantile(0.75)`

$$IQR = Q3 - Q1$$

$$\text{lower} = Q1 - 1.5 * IQR$$

$$\text{upper} = Q1 + 1.5 * IQR$$

$\text{def } [\text{'area'}] = \text{np. where } (\text{df } [\text{'area'}]) > \text{upper}$,

upper, np.where (df['area'] < lower,
lower, df['area'])

~~df 2 = df- encoded~~

df2['urea'] = df2['urea'] - 2 * score - (df2['urea'].mean())

$\text{df } 2 \left[\cdot \text{Vrea} \right] = \text{np} \cdot \text{where} \left(\text{df } 2 \left[\cdot \text{Vrea} - \text{zscore} \right] \right)$

abs() > 3, np.nan,
df2['area'])

$\text{df3} = \text{df_encoded}$

```
df3['urea - zscore'] = stats.zscore(df3['urea'])
```

median-wrea = df3['wrea'] · median()

`df3['urea'] = np.where(df3['urea-zscore'] > abs(1))`

>3, median - urea, df3

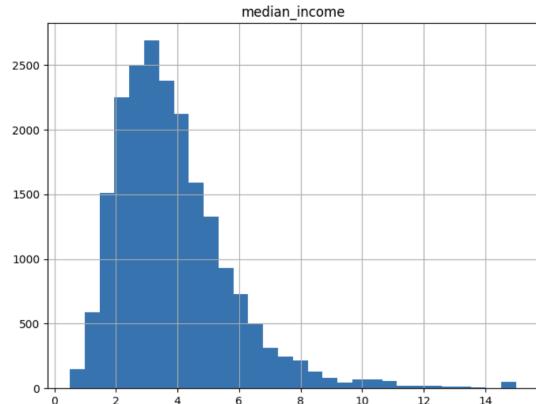
['urea'])

CODE WITH OUTPUT

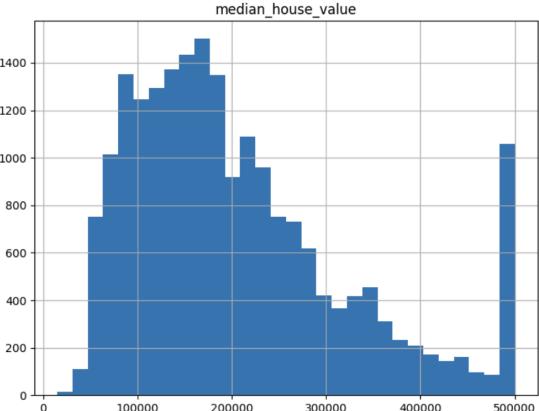
```
# Load the dataset into a pandas DataFrame  
df = pd.read_csv('housing.csv')  
  
# Display descriptive statistics  
df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000

```
import matplotlib.pyplot as plt  
df.hist(column="median_income", bins=30, figsize=(8,6))  
plt.show()
```



```
import matplotlib.pyplot as plt  
df.hist(column="median_house_value", bins=30, figsize=(8,6))  
plt.show()
```

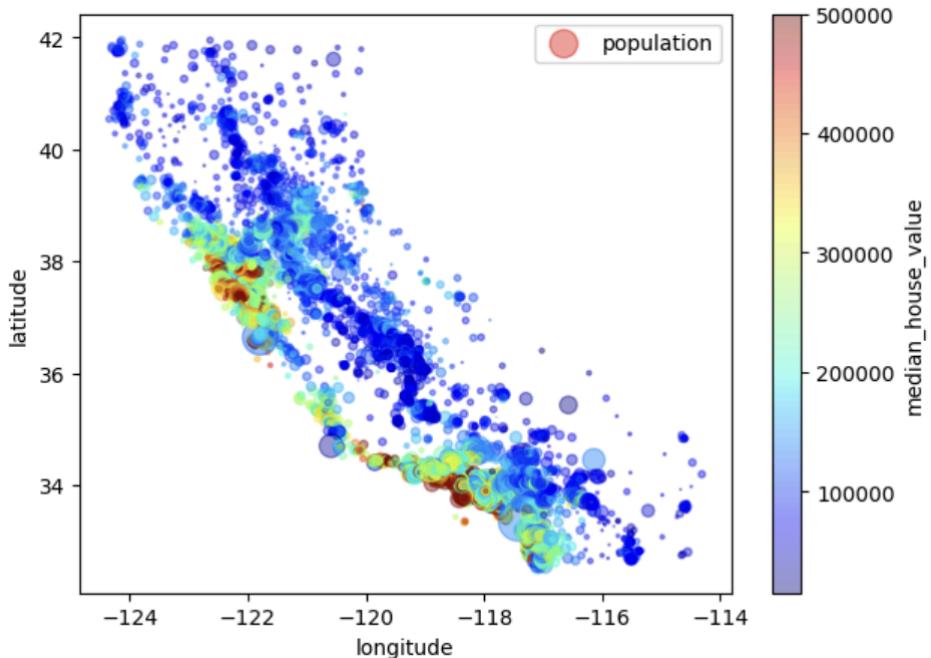


```
import pandas as pd  
import numpy as np  
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit  
  
# Load the dataset  
housing = pd.read_csv('housing.csv')  
  
# For this demonstration, consider only 'median_income' and 'median_house_value'  
housing_selected = housing[['median_income', 'median_house_value']].copy()  
  
# Random split: This splits the data randomly without preserving any specific distribution.  
train_set_random, test_set_random = train_test_split(housing_selected, test_size=0.2, random_state=42)  
  
# For stratified sampling, first create an income category.  
housing_selected['income_cat'] = pd.cut(housing_selected['median_income'],  
                                         bins=[0., 1.5, 3.0, 4.5, 6., np.inf],  
                                         labels=[1, 2, 3, 4, 5])  
  
# Use StratifiedShuffleSplit to ensure the income distribution is preserved in both sets.  
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)  
for train_index, test_index in split.split(housing_selected, housing_selected['income_cat']):  
    strat_train_set = housing_selected.loc[train_index]  
    strat_test_set = housing_selected.loc[test_index]  
  
# Remove the temporary income category attribute.
```

```
for dataset in (strat_train_set, strat_test_set):
    dataset.drop("income_cat", axis=1, inplace=True)
```

```
import matplotlib.pyplot as plt
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(7,5),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7e55a2076b10>
```

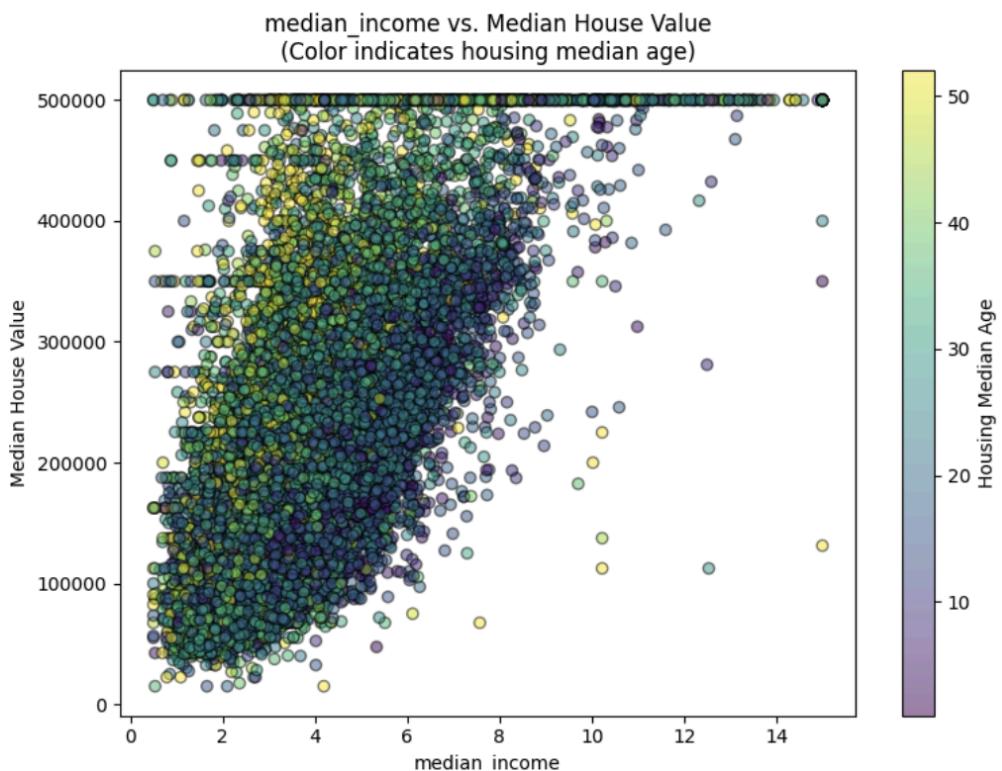


```

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,6))
# Differentiate by using 'housing_median_age' for the color
scatter = plt.scatter(housing_numeric[max_feature],
                      housing_numeric["median_house_value"],
                      alpha=0.5,
                      c=housing_numeric["housing_median_age"],
                      cmap='viridis',
                      edgecolor='k')
plt.xlabel(max_feature)
plt.ylabel("Median House Value")
plt.title(f"{max_feature} vs. Median House Value\n(Color indicates housing median age)")
# Add a colorbar to explain the color mapping
cbar = plt.colorbar(scatter)
cbar.set_label("Housing Median Age")
plt.tight_layout()
plt.show()

```



```

from sklearn.preprocessing import OneHotEncoder

# Extract the categorical attribute
housing_cat = housing[["ocean_proximity"]]

# Perform one-hot encoding
encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat).toarray()

# Create a DataFrame for the encoded features
housing_cat_1hot_df = pd.DataFrame(housing_cat_1hot,
                                   columns=encoder.get_feature_names_out(["ocean_proximity"]))
housing_cat_1hot_df.head()
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

# Custom transformer to add engineered attributes
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):

```

```

def __init__(self, add_bedrooms_per_room=True):
    self.add_bedrooms_per_room = add_bedrooms_per_room
def fit(self, X, y=None):
    return self
def transform(self, X):
    # Assumes X is a NumPy array with the following columns:
    # total_rooms (index 3), total_bedrooms (index 2), population (index 4), households (index 5)
    rooms_per_household = X[:, 3] / X[:, 5]
    population_per_household = X[:, 4] / X[:, 5]
    if self.add_bedrooms_per_room:
        bedrooms_per_room = X[:, 2] / X[:, 3]
        return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
    else:
        return np.c_[X, rooms_per_household, population_per_household]

# Identify numerical and categorical columns
num_attribs = housing.drop("ocean_proximity", axis=1).columns # All numeric columns
cat_attribs = ["ocean_proximity"]

# Build numerical pipeline: impute missing values, add new attributes, then scale
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

```

Build the full pipeline combining numerical and categorical processing

```

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])

```

Process the dataset using the pipeline

```

housing_prepared = full_pipeline.fit_transform(housing)
print("Shape of processed data:", housing_prepared.shape)

```

LABORATORY PROGRAM – 3

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

OBSERVATION BOOK

12/03/25

Lab 2

```

import pandas as pd
from collections import Counter
def entropy(data):
    labels = data['label'].tolist()
    counts = Counter(labels)
    probabilities = [count / len(labels) for count in counts.values()]
    entropy_value = -sum(p * math.log2(p) for p in probabilities if p > 0)
    return entropy_value

def gain(data, feature):
    initial_entropy = entropy(data)
    feature_values = data[feature].unique()
    weighted_entropy = 0
    for value in feature_values:
        subset = data[data[feature] == value]
        weighted_entropy += (len(subset) / len(data)) * entropy(subset)
    return initial_entropy - weighted_entropy

def id3(data, features, target_attribute):
    if len(data['label'].unique()) == 1:
        return data['label'].iloc[0]

    if len(features) == 0:
        return data['label'].value_counts().index[0]

```

```

best_feature = max(features, key=lambda feature:
                    gain(data, feature))

tree = {best_feature: {}}
features = [f for f in features if f != best_feature]
for value in data[best_feature].unique():
    subset = data[data[best_feature] == value].drop(
        columns=[best_feature])
    if len(subset) == 0:
        tree[best_feature][value] = data['label'].value_counts().index[0]
    else:
        tree[best_feature][value] = id3(subset,
                                         features, target_attribute)

return tree

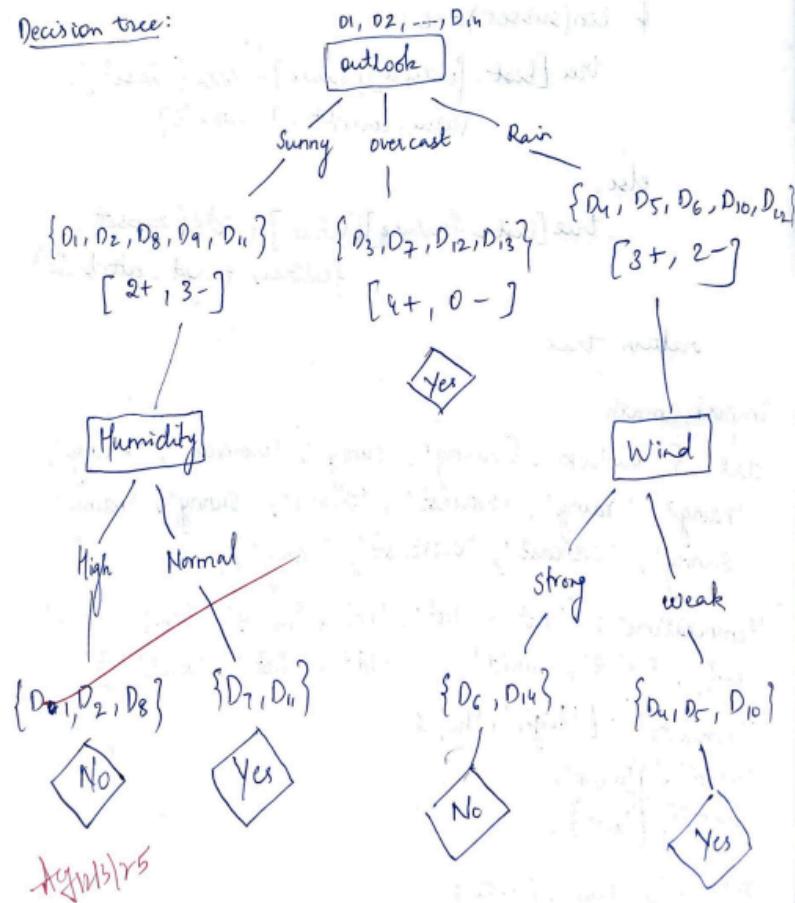
import math
data = {'outlook': ['sunny', 'sunny', 'overcast', 'rainy',
                    'rainy', 'rainy', 'overcast', 'sunny', 'sunny', 'rainy',
                    'sunny', 'overcast', 'overcast', 'rainy'],
        'temperature': ['hot', 'hot', 'hot', 'mild', 'cool', 'cool',
                        'cool', 'mild', 'mild', 'mild', 'hot', 'mild'],
        'humidity': ['high', 'high'],
        'wind': ['weak'],
        'label': ['no']}
df = pd.DataFrame(data)
features = ['outlook', 'temperature', 'humidity', 'wind']
target_attribute = 'label'

```

decision-tree = id3(df, features, target-attribute)
print(decision-tree)

Output: { 'outlook': { 'sunny': { 'humidity': { 'high': 'no', 'normal': 'yes' }, 'overcast': 'yes' }, 'rainy': { 'wind': { 'weak': 'yes', 'strong': 'no' } } }

Decision tree:



CODE WITH OUTPUT

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

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

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

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

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

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

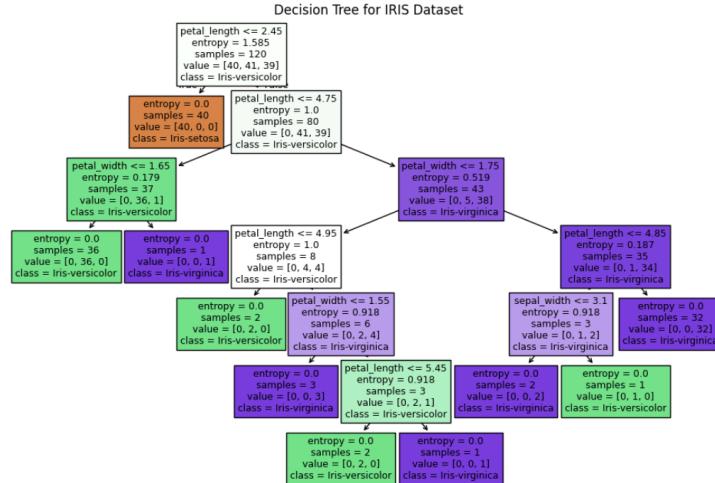
# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf_iris, filled=True, feature_names=X.columns, class_names=clf_iris.classes_)
plt.title("Decision Tree for IRIS Dataset")
plt.show()
```

```

IRIS Dataset Decision Tree Classifier
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
             precision    recall   f1-score   support
Iris-setosa      1.00     1.00     1.00      10
Iris-versicolor  1.00     1.00     1.00       9
Iris-virginica   1.00     1.00     1.00      11

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

```



```

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

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

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

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

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

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

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

print("Drug Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_drug)

```

```

print("Confusion Matrix:\n", conf_matrix_drug)
print("Classification Report:\n", classification_report(y_test_d, y_pred_drug))

```

```

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

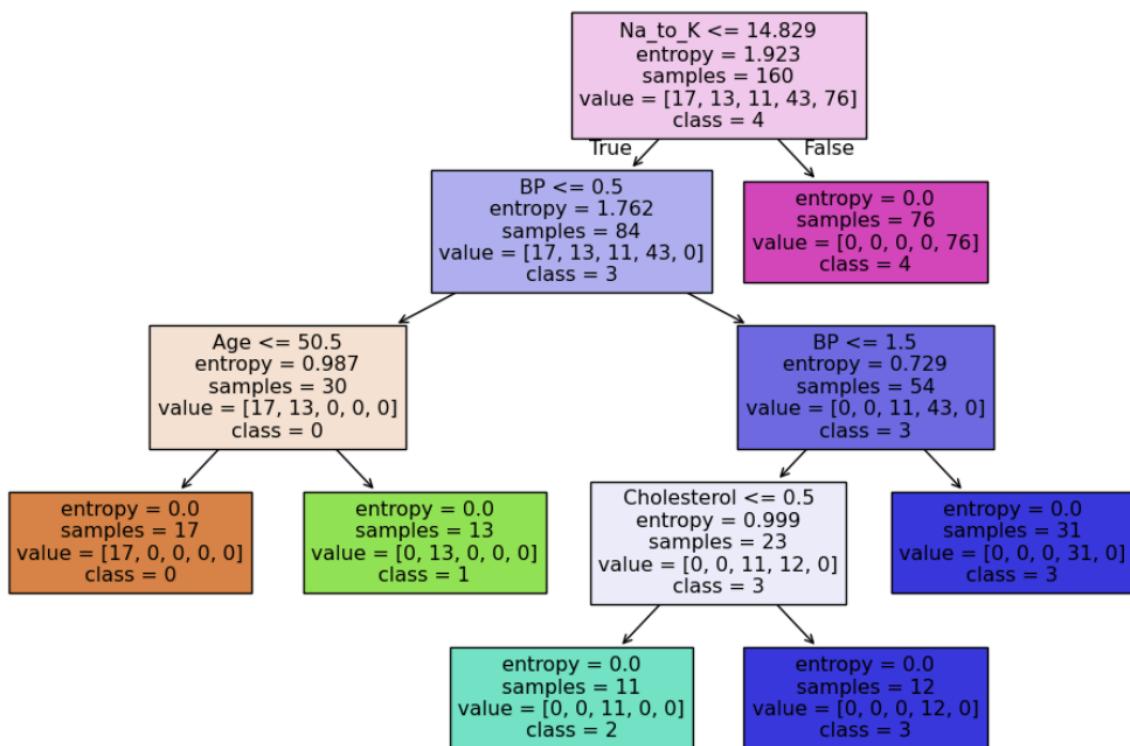
```

```

Drug Dataset Decision Tree Classifier
Accuracy: 1.0
Confusion Matrix:
[[ 6  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  5  0  0]
 [ 0  0  0 11  0]
 [ 0  0  0 15]]
Classification Report:
      precision    recall   f1-score   support
          0       1.00    1.00    1.00      6
          1       1.00    1.00    1.00      3
          2       1.00    1.00    1.00      5
          3       1.00    1.00    1.00     11
          4       1.00    1.00    1.00     15
          accuracy           1.00      40
          macro avg       1.00    1.00    1.00      40
          weighted avg    1.00    1.00    1.00      40

```

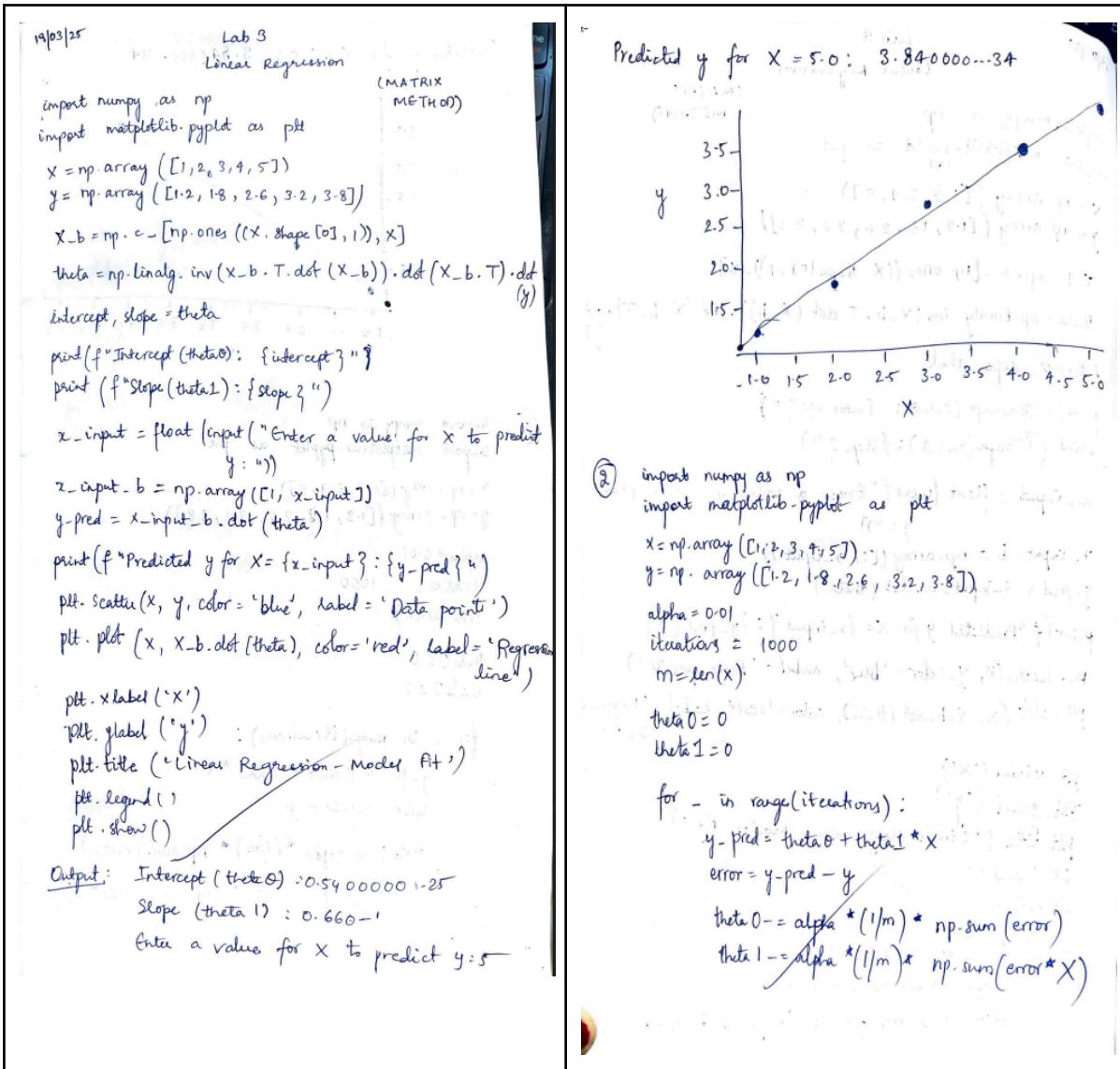
Decision Tree for Drug Dataset



LABORATORY PROGRAM – 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

OBSERVATION BOOK



```

print(f"Intercept (theta 0) : {theta0}")
print(f"Slope (theta 1) : {theta1}")

x_input = float(input("Enter a value for x to predict y:"))
y_pred = theta0 + theta1 * x_input

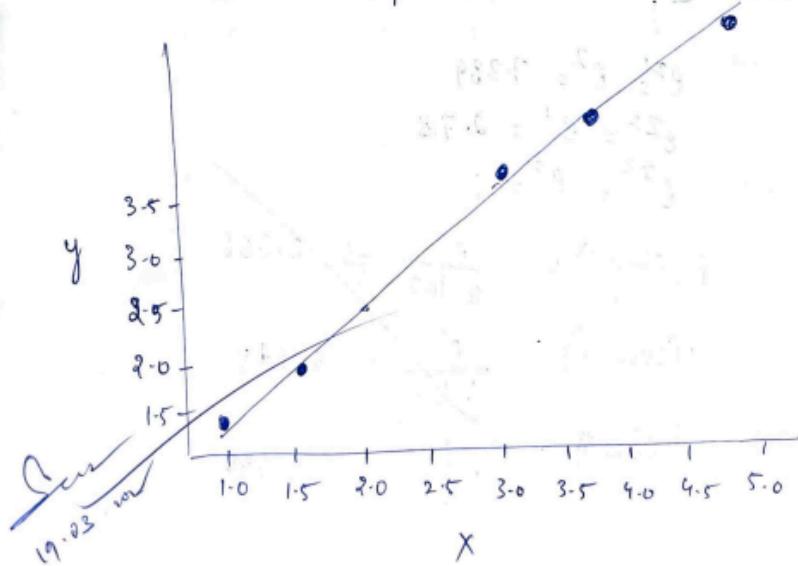
print(f"Predicted y for x = {x_input} : {y_pred}")

plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, theta0 + theta1 * x, color='red',
          label='Regression line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.show()

```

Output:

Intercept (theta 0) = 0.47890448451315564
 Slope (theta 1) = 0.6769224781201147
 Enter a value for X to predict y: 9
 Predicted y for X = 9.0 = 6.57120678759411



CODE WITH OUTPUT

```
import pandas as pd
from sklearn.linear_model import LinearRegression
# Load the data
income_data = pd.read_csv("canada_per_capita_income.csv")
# Assumed data columns: 'Year' and 'PerCapitaIncome'
print("Canada Income Data Head:")
print(income_data.head())
# Prepare feature and target
X_income = income_data[["year"]] # Predictor variable: Year
y_income = income_data["per capita income (US$)"]
# Build and train the linear regression model
model_income = LinearRegression()
model_income.fit(X_income, y_income)

# Predict per capita income for the year 2020
predicted_income = model_income.predict([[2020]])

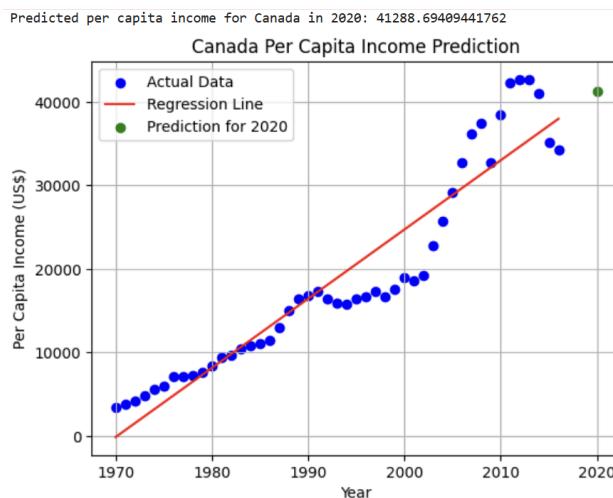
print("\nPredicted per capita income for Canada in 2020:", predicted_income[0])

# Plot the data points and the regression line
plt.scatter(X_income, y_income, color='blue', label='Actual Data')
plt.plot(X_income, model_income.predict(X_income), color='red', label='Regression Line')

# Plot the prediction for 2020
plt.scatter(2020, predicted_income[0], color='green', label='Prediction for 2020')

# Customize the plot
plt.xlabel('Year')
plt.ylabel('Per Capita Income (US$)')
plt.title('Canada Per Capita Income Prediction')
plt.legend()
plt.grid(True)

# Display the plot
plt.show()
```



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

# Load the salary data
salary_data = pd.read_csv("salary.csv")
print(salary_data.head())
```

```

# Prepare feature and target
X_salary = salary_data[["YearsExperience"]] # Predictor variable: Years of Experience
y_salary = salary_data["Salary"]

# Build and train the linear regression model
model_salary = LinearRegression()
model_salary.fit(X_salary, y_salary)

import matplotlib.pyplot as plt
# Plot the data points and the regression line
plt.scatter(X_salary, y_salary, color='blue', label='Actual Data')
plt.plot(X_salary, model_salary.predict(X_salary), color='red', label='Regression Line')

# Plot the prediction for 12 years of experience
plt.scatter(12, predicted_salary[0], color='green', label='Prediction for 12 years')

# Customize the plot
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Salary Prediction based on Experience')
plt.legend()
plt.grid(True)

# Display the plot
plt.show()

```

Predicted salary for an employee with 12 years of experience: 139980.88923969213



```

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

# Read the CSV file (ensure the file is uploaded in your Colab environment)
df = pd.read_csv("hiring.csv")

# Rename columns for convenience
df.columns = ['experience', 'test_score', 'interview_score', 'salary']

```

```

print("Original Data:")
print(df)
# Function to convert experience values to numeric
def convert_experience(x):
    try:
        return float(x)
    except:
        x_lower = str(x).strip().lower()
        return num_map.get(x_lower, np.nan)

# Convert the 'experience' column using the mapping
df['experience'] = df['experience'].apply(convert_experience)

# Convert 'test_score', 'interview_score', and 'salary' to numeric (coerce errors to NaN)
df['test_score'] = pd.to_numeric(df['test_score'], errors='coerce')
df['interview_score'] = pd.to_numeric(df['interview_score'], errors='coerce')
df['salary'] = pd.to_numeric(df['salary'], errors='coerce')

print("\nData After Conversion:")
print(df)

# Fill missing values in numeric columns using the column mean
df['experience'].fillna(df['experience'].mean(), inplace=True)
df['test_score'].fillna(df['test_score'].mean(), inplace=True)
df['interview_score'].fillna(df['interview_score'].mean(), inplace=True)

print("\nData After Filling Missing Values:")
print(df)

# Prepare the feature matrix X and target vector y
X = df[['experience', 'test_score', 'interview_score']]
y = df['salary']

# Build and train the Multiple Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Predict salaries for the given candidate profiles
# Candidate 1: 2 years of experience, 9 test score, 6 interview score
candidate1 = np.array([[2, 9, 6]])
predicted_salary1 = model.predict(candidate1)

# Candidate 2: 12 years of experience, 10 test score, 10 interview score
candidate2 = np.array([[12, 10, 10]])
predicted_salary2 = model.predict(candidate2)

print("\nPredicted Salary for Candidate (2 yrs, 9 test, 6 interview): $", round(predicted_salary1[0], 2))
print("Predicted Salary for Candidate (12 yrs, 10 test, 10 interview): $", round(predicted_salary2[0], 2))

import matplotlib.pyplot as plt

# Create the plot
plt.figure(figsize=(10, 6)) # Adjust figure size for better visualization
plt.scatter(df['experience'], y, color='blue', label='Actual Salary') # Plot actual salary against years of experience

# Plot the regression line (this is an approximation since it's a multi-variable regression)
# You can visualize a single feature against the predicted salary
plt.plot(df['experience'], model.predict(X), color='red', label='Regression Line')

# Highlight predictions
plt.scatter(candidate1[0, 0], predicted_salary1, color='green', label='Candidate 1 Prediction')
plt.scatter(candidate2[0, 0], predicted_salary2, color='purple', label='Candidate 2 Prediction')

# Add labels and title
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Salary Prediction based on Experience, Test Score, Interview Score")

```

```
# Add a legend  
plt.legend()  
plt.grid(True)  
plt.show()
```



LABORATORY PROGRAM – 5

Build Logistic Regression Model for a given dataset

OBSERVATION BOOK

<p><u>24/25.</u></p> <p><u>Lab 4 (Logistic Regression)</u></p> <p>① $a_0 = -5$, $a_1 = 0.8$ (intcept) (slope)</p> <p>a. $Z = \frac{1}{1 + e^{-(5+0.8x)}}$ $\Rightarrow P(\text{Pass})$</p> <p>b. $Z = \frac{1}{1 + e^{-(5+0.8x)}} \Rightarrow P(\text{Pass}) = 0.645$</p> <p>c. $P(\text{Pass}) > 0.5 \rightarrow \text{Hence, "Pass"}$</p> <p>② $P(\text{class } i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$ for $Z_i = [2, 1, 0]$</p> <p>$\sum_j e^{z_j} = e^2 + e^1 + e^0 \approx 11.107$</p> <p>$e^{z_1} = e^2 = 7.389$</p> <p>$e^{z_2} = e^1 = 2.718$</p> <p>$e^{z_3} = e^0 = 1$</p> <p>$P(\text{class 1}) = \frac{e^2}{11.107} \approx 0.666$</p> <p>$P(\text{class 2}) = \frac{e^1}{11.107} \approx 0.245$</p> <p>$P(\text{class 3}) = \frac{1}{11.107} \approx 0.089$</p>	<p><u>Logistic Regression</u></p> <p>i. HR - comma - sep. csv most</p> <p>① 'Satisfaction - level' as it had 'direct impact of 'left' output variable'</p> <p>② 77.07% accuracy</p> <p>2. Zoo dataset</p> <p>① Yes. check for missing (null) values in the dataset & dropping rows with missing values.</p> <p>② Accuracy of the model is 95%.</p> <p>③ 4th class is the only misclassified class due to probable inconsistency in data & mislead - endings in relationships of the data.</p> <p><u>KNN</u></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th></th> <th>Person</th> <th>Age</th> <th>Salary(k)</th> <th>Target</th> <th>Distance</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>18</td> <td>50</td> <td>N</td> <td>52.8</td> <td>5</td> </tr> <tr> <td>B</td> <td>28</td> <td>55</td> <td>N</td> <td>46.6</td> <td>4</td> </tr> <tr> <td>C</td> <td>24</td> <td>90</td> <td>N</td> <td>31.9</td> <td>2</td> </tr> <tr> <td>D</td> <td>41</td> <td>60</td> <td>Y</td> <td>40.5</td> <td>3</td> </tr> <tr> <td>E</td> <td>43</td> <td>70</td> <td>Y</td> <td>31.1</td> <td>1</td> </tr> <tr> <td>F</td> <td>38</td> <td>90</td> <td>Y</td> <td>60.1</td> <td>6</td> </tr> <tr> <td>X</td> <td>35</td> <td>100</td> <td>?</td> <td>-</td> <td>-</td> </tr> </tbody> </table> <p>$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$</p>		Person	Age	Salary(k)	Target	Distance	A	18	50	N	52.8	5	B	28	55	N	46.6	4	C	24	90	N	31.9	2	D	41	60	Y	40.5	3	E	43	70	Y	31.1	1	F	38	90	Y	60.1	6	X	35	100	?	-	-
	Person	Age	Salary(k)	Target	Distance																																												
A	18	50	N	52.8	5																																												
B	28	55	N	46.6	4																																												
C	24	90	N	31.9	2																																												
D	41	60	Y	40.5	3																																												
E	43	70	Y	31.1	1																																												
F	38	90	Y	60.1	6																																												
X	35	100	?	-	-																																												

① Iris

For k=5 \Rightarrow 96.67% accuracy
3.33% error rate

$K=4 \Rightarrow$ 100% accuracy

0% error rate

$K=3 \Rightarrow$ 93.33% accuracy
6.67% error rate

Intuitively, higher k value gives better accuracy & lower error rate.

② Diabetes

It is important as it is distance based.

Otherwise cannot be scaled properly.

$$\text{Standard scalar: } Z = \frac{x - \mu}{\sigma}$$

μ = mean

σ = std

x = feature value

CODE WITH OUTPUT

```
import pandas as pd
from matplotlib import pyplot as plt
# %matplotlib inline
#"%matplotlib inline" will make your plot outputs appear and be stored within the notebook.

df = pd.read_csv("insurance_data.csv")
df.head()

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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']], df.bought_insurance, train_size=0.9, random_state=10)
X_train.shape

X_test

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

model.fit(X_train, y_train)

X_test

y_test

y_predicted = model.predict(X_test)
y_predicted

model.score(X_test, y_test)

model.predict_proba(X_test)

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

#model.coef_ indicates value of m in y=m*x + b equation
model.coef_

#model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

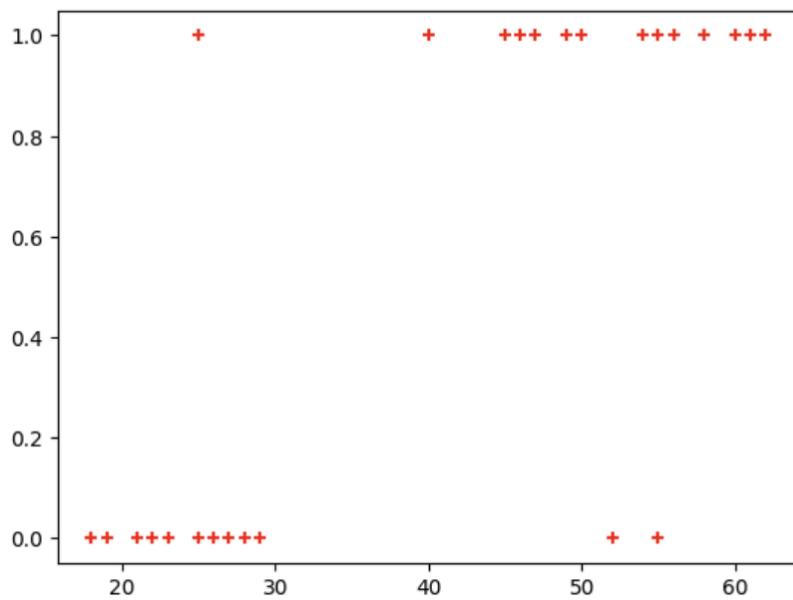
def prediction_function(age):
    z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
    y = sigmoid(z)
    return y

age = 35
prediction_function(age)

"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""


```

'0.37 is less than 0.5 which means person with 35 will not buy the insurance'



```
# Import necessary libraries
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.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)

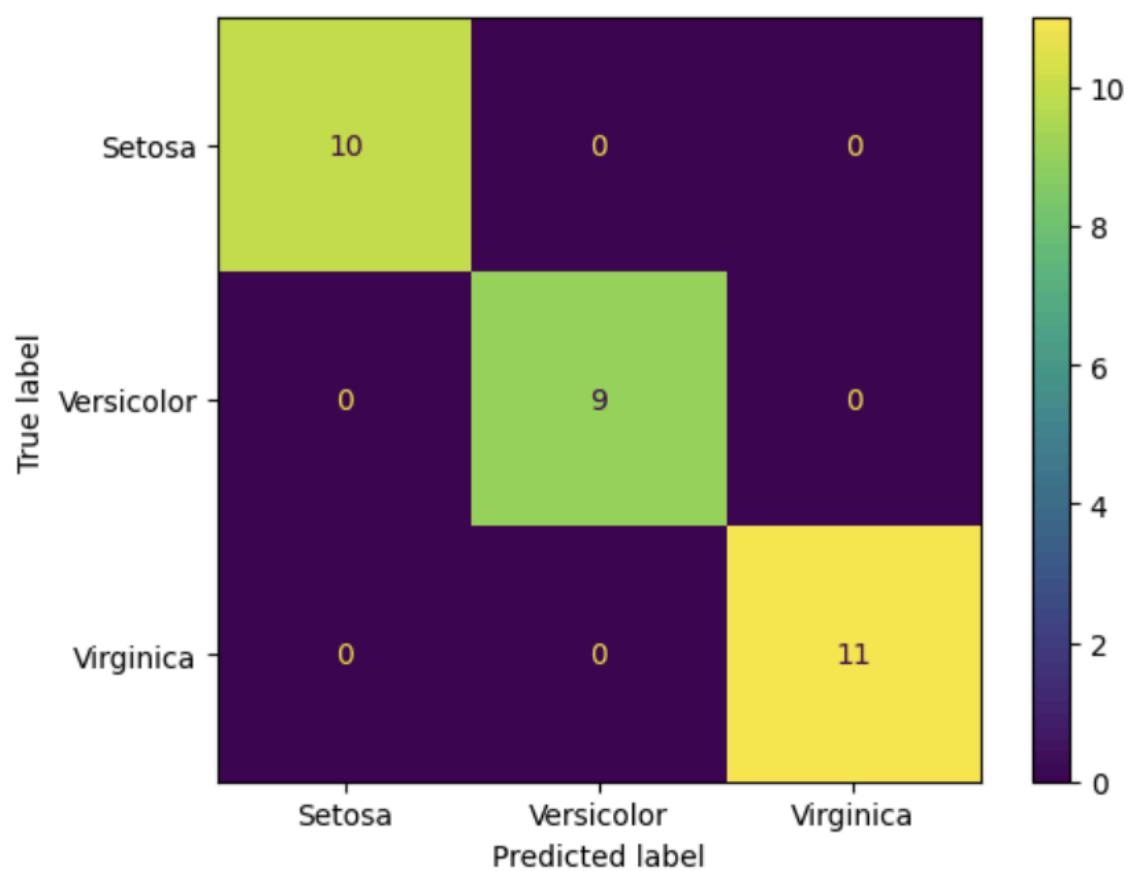
# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")

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()
```

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



LABORATORY PROGRAM – 6

Build KNN Classification model for a given dataset.

CODE WITH OUTPUT

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# For model building and evaluation
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

# ----- Part 1: IRIS Dataset -----
# Load the iris dataset (ensure iris.csv is in the same directory or provide correct path)
iris_df = pd.read_csv("iris.csv")

# Separate features and target
X_iris = iris_df.drop("species", axis=1)
y_iris = iris_df["species"]

# Split the data (80% training, 20% testing)
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
)

# Choose a value for k; here K=3 is used as an example.
knn_iris = KNeighborsClassifier(n_neighbors=3)

# Train the model on training data
knn_iris.fit(X_train_iris, y_train_iris)

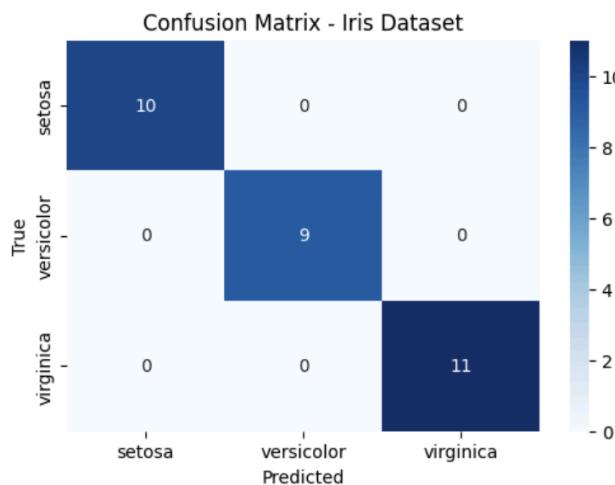
# Predict on test data
y_pred_iris = knn_iris.predict(X_test_iris)

# Calculate accuracy score
acc_iris = accuracy_score(y_test_iris, y_pred_iris)
print("IRIS Dataset Accuracy Score:", acc_iris)

# Compute confusion matrix and classification report
cm_iris = confusion_matrix(y_test_iris, y_pred_iris)
print("\nIRIS Dataset Confusion Matrix:\n", cm_iris)

cr_iris = classification_report(y_test_iris, y_pred_iris)
print("\nIRIS Dataset Classification Report:\n", cr_iris)
```

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



```

# ----- Part 2: Diabetes Dataset ----- #
# Load the diabetes dataset (ensure diabetes.csv is in the same directory or provide correct path)
diabetes_df = pd.read_csv("diabetes.csv")

# Separate features and target (Outcome column is assumed to be the target)
X_diabetes = diabetes_df.drop("Outcome", axis=1)
y_diabetes = diabetes_df["Outcome"]

# Perform feature scaling on the features
scaler = StandardScaler()
X_scaled_diabetes = scaler.fit_transform(X_diabetes)

# Split the scaled data (80% training, 20% testing)
X_train_diab, X_test_diab, y_train_diab, y_test_diab = train_test_split(
    X_scaled_diabetes, y_diabetes, test_size=0.2, random_state=42
)

# Choose a value for k; here K=5 is used as an example.
knn_diabetes = KNeighborsClassifier(n_neighbors=5)

# Train the model on training data
knn_diabetes.fit(X_train_diab, y_train_diab)

# Predict on test data
y_pred_diab = knn_diabetes.predict(X_test_diab)

# Calculate accuracy score
acc_diab = accuracy_score(y_test_diab, y_pred_diab)
print("Diabetes Dataset Accuracy Score:", acc_diab)

# Compute confusion matrix and classification report
cm_diab = confusion_matrix(y_test_diab, y_pred_diab)
print("\nDiabetes Dataset Confusion Matrix:\n", cm_diab)

```

```

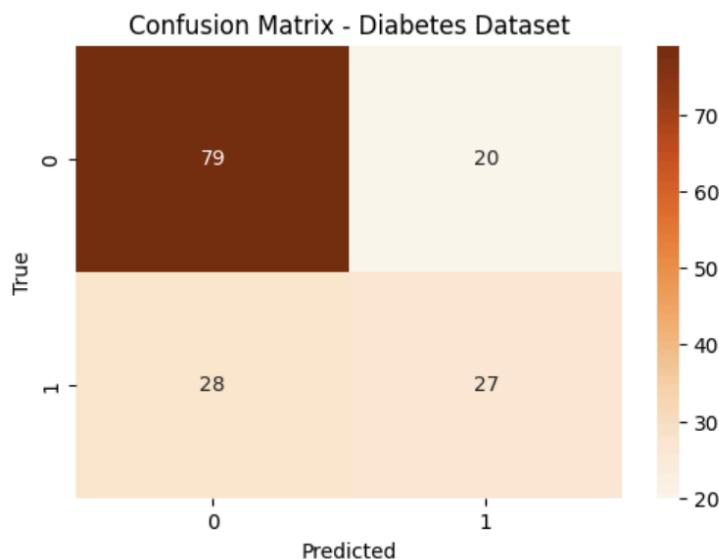
cr_diab = classification_report(y_test_diab, y_pred_diab)
print("\nDiabetes Dataset Classification Report:\n", cr_diab)

Diabetes Dataset 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

```



```

# ----- Load the Dataset -----
# Load heart.csv (make sure the file is in your working directory)
heart_df = pd.read_csv("heart.csv")

# Display the first few rows to check the data
heart_df.head()

# ----- Data Preparation -----
# Separate features and target
X_heart = heart_df.drop("target", axis=1)
y_heart = heart_df["target"]

# Perform feature scaling (important for distance-based algorithms like KNN)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_heart)

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_heart, test_size=0.2, random_state=42)
# ----- Finding the Best k -----
# We will try a range of k values (neighbors) and select the one with maximum accuracy.
k_range = range(1, 21)
accuracy_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred)

```

```

accuracy_scores.append(acc)
print(f"k = {k} --> Accuracy: {acc:.4f}")

    k = 1 --> Accuracy: 0.8525
    k = 2 --> Accuracy: 0.8197
    k = 3 --> Accuracy: 0.8689
    k = 4 --> Accuracy: 0.8852
    k = 5 --> Accuracy: 0.9180
    k = 6 --> Accuracy: 0.9344
    k = 7 --> Accuracy: 0.9180
    k = 8 --> Accuracy: 0.8525
    k = 9 --> Accuracy: 0.8852
    k = 10 --> Accuracy: 0.8852
    k = 11 --> Accuracy: 0.8852
    k = 12 --> Accuracy: 0.8689
    k = 13 --> Accuracy: 0.8852
    k = 14 --> Accuracy: 0.8689
    k = 15 --> Accuracy: 0.9016
    k = 16 --> Accuracy: 0.8852
    k = 17 --> Accuracy: 0.8852
    k = 18 --> Accuracy: 0.9016
    k = 19 --> Accuracy: 0.8852
    k = 20 --> Accuracy: 0.8852

|: # Determine the best k value
best_k = k_range[np.argmax(accuracy_scores)]
print("\nBest k value:", best_k)

Best k value: 6

```

```

# ----- Train Final Model with Best k -----
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(X_train, y_train)
y_pred_best = best_knn.predict(X_test)

```

```

# Compute final accuracy, confusion matrix and classification report
final_accuracy = accuracy_score(y_test, y_pred_best)
cm = confusion_matrix(y_test, y_pred_best)
cr_text = classification_report(y_test, y_pred_best)
print("\nFinal Accuracy Score:", final_accuracy)
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", cr_text)

```

Final Accuracy Score: 0.9344262295081968

Confusion Matrix:

[[28 1]	
[3 29]]	

Classification Report:				
	precision	recall	f1-score	support
0	0.90	0.97	0.93	29
1	0.97	0.91	0.94	32
accuracy			0.93	61
macro avg	0.93	0.94	0.93	61
weighted avg	0.94	0.93	0.93	61

LABORATORY PROGRAM – 7

Build Support vector machine model for a given dataset

OBSERVATION BOOK

Date _____
Page _____

LABORATORY PROGRAM - 7

BUILT SUPPORT VECTOR MACHINE FOR A GIVEN DATASET

- Points $(1, 1)$, $(4, -1)$ and $(6, 0)$ belong to positive class and points $(1, 0)$, $(0, 1)$ & $(0, -1)$ belong to negative class. Draw an optimal hyperplane.
- Point-to-line distance: $Ax + by + c = 0$ and point (x_0, y_0)
 $|Ax_0 + By_0 + c| / \sqrt{A^2 + B^2}$
- Deviation for these points:
 - Positives: $(1, 1)$, $(4, -1)$, $(6, 0)$
 - Negatives: $(1, 0)$, $(0, 1)$, $(0, -1)$
- Shape of separating line: $x = b$, because the points differ only in their x values.
- Nearest positive support points are $(1, 1)$, & $(4, -1)$. Thus: $x = b$ is $|4 - b|$.
- Nearest negative support point is $(1, 0)$. Thus: $x = b$ is $|1 - b|$.
- Maximizing the minimal distance:
 $b - 1 = |4 - b| \Rightarrow 2b = 5 \Rightarrow b = 2.5$
 Hence margin on each is $4 - 2.5 = 1.5$
- Final Hyperplane and Margin Line:
- Decision boundary: $x = 2.5$; $w = [1, 0]$, $b = 2.5$
 $\therefore x + b = 0$
- Margin - defining lines: $x = 1.0$ and $x = 4.0$
 These pass through the support vectors $(1, 0)$ and $(4, -1)$ respectively.

Observation:

1. For "iris.csv" dataset:
 - Accuracy Score: 100%
 Linear Kernel: 100%
 RBF Kernel: 100% (all)
 - Radial Basis Function Kernel gave better performance.
 RBF Kernel is capable of capturing non-linear relationships in data by mapping input features into a higher-dimensional space.
2. For "Letter-Recognition.csv" dataset:
 - Confusion Matrix Interpretation: The SVM model achieved an accuracy of 95% on the test data. The confusion matrix shows how each letter was predicted versus its actual class.
 - AUC Score: The micro-average AUC score is 1.00, which means the model is very effective at ranking predictions - it assigns higher scores to correct classes with high confidence.

Performance Comparison: SVM on DRIS vs Letter-Recognition Dataset:

Dataset		DRIS Dataset	Letter Recognition
• Classes	3	3	26 (A-Z)
• Accuracy	100%	95%	95%
• AUC	Not computed	Not computed	1.0 (Micro Avg)

CODE WITH OUTPUT

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

# Data points
X = np.array([[4, 1], [4, -1], [6, 0], [1, 0], [0, 1], [0, -1]])
y = np.array([1, 1, 1, -1, -1, -1])

# Fit linear SVM with a very large C to approximate hard-margin
clf = SVC(kernel='linear', C=1e6)
clf.fit(X, y)

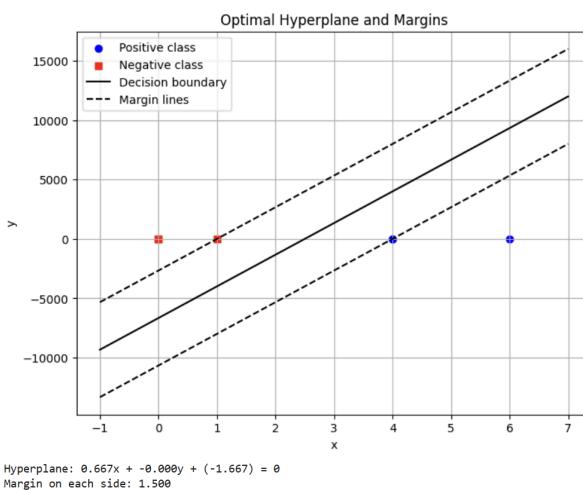
# Extract model parameters
w = clf.coef_[0]
b = clf.intercept_[0]

# Compute decision boundary and margins
xx = np.linspace(-1, 7, 500)
yy = -(w[0] * xx + b) / w[1]

# Margin offset: distance = 1/||w||
margin = 1 / np.linalg.norm(w)
yy_down = yy - np.sqrt(1 + (w[0] / w[1])**2) * margin
yy_up = yy + np.sqrt(1 + (w[0] / w[1])**2) * margin

# Plotting
plt.figure(figsize=(8, 6))
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='blue', marker='o', label='Positive class')
plt.scatter(X[y == -1, 0], X[y == -1, 1], c='red', marker='s', label='Negative class')
plt.plot(xx, yy, 'k-', label='Decision boundary')
plt.plot(xx, yy_down, 'k--', label='Margin lines')
plt.plot(xx, yy_up, 'k--')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Optimal Hyperplane and Margins')
plt.grid(True)
plt.show()

# Print hyperplane equation
print(f"Hyperplane: {w[0]:.3f}x + {w[1]:.3f}y + ({b:.3f}) = 0")
print(f"Margin on each side: {margin:.3f}")
```



```
import pandas as pd
```

```
# Load both datasets
```

```

iris_df = pd.read_csv("/content/iris.csv")
# 1. IRIS DATASET - SVM with RBF and Linear Kernels
X_iris = iris_df.drop("species", axis=1)
y_iris = iris_df["species"]

# Encode labels
le_iris = LabelEncoder()
y_iris_encoded = le_iris.fit_transform(y_iris)

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

# Train models
svm_rbf = SVC(kernel='rbf')
svm_linear = SVC(kernel='linear')

svm_rbf.fit(X_train_iris, y_train_iris)
svm_linear.fit(X_train_iris, y_train_iris)

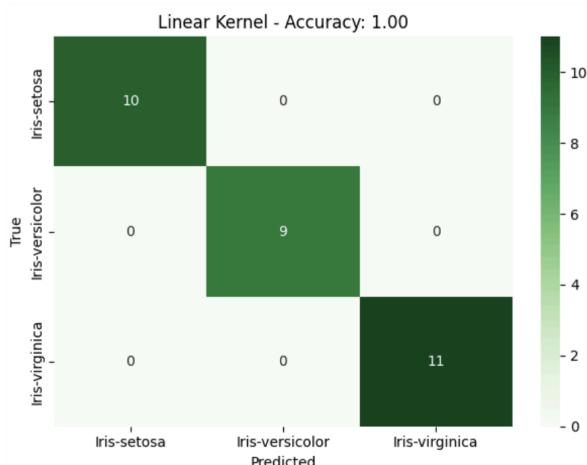
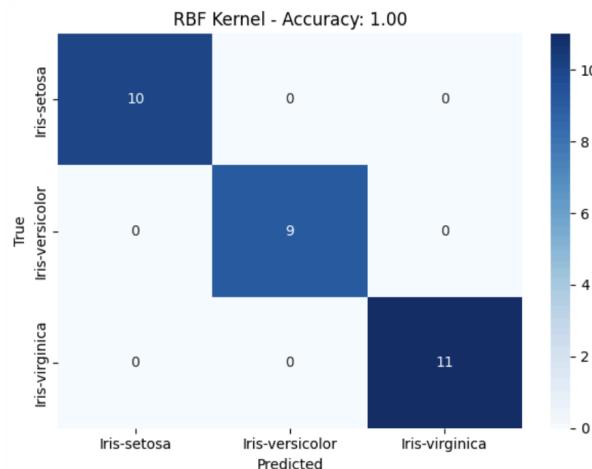
# Predictions
y_pred_rbf = svm_rbf.predict(X_test_iris)
y_pred_linear = svm_linear.predict(X_test_iris)

```

```

# Accuracy and Confusion Matrix
acc_rbf = accuracy_score(y_test_iris, y_pred_rbf)
acc_linear = accuracy_score(y_test_iris, y_pred_linear)
cm_rbf = confusion_matrix(y_test_iris, y_pred_rbf)
cm_linear = confusion_matrix(y_test_iris, y_pred_linear)

```



```

# Load dataset
letter_df = pd.read_csv("/content/letter-recognition.csv") # Update path if needed

```

```

letter_df['letter'] = LabelEncoder().fit_transform(letter_df['letter'])

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

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

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

# Train SVM
svm = SVC(kernel='rbf', probability=True)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
y_prob = svm.predict_proba(X_test)

# Accuracy and Confusion Matrix
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ROC and AUC (one-vs-rest)
y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

fpr = dict()
tpr = dict()
roc_auc = dict()

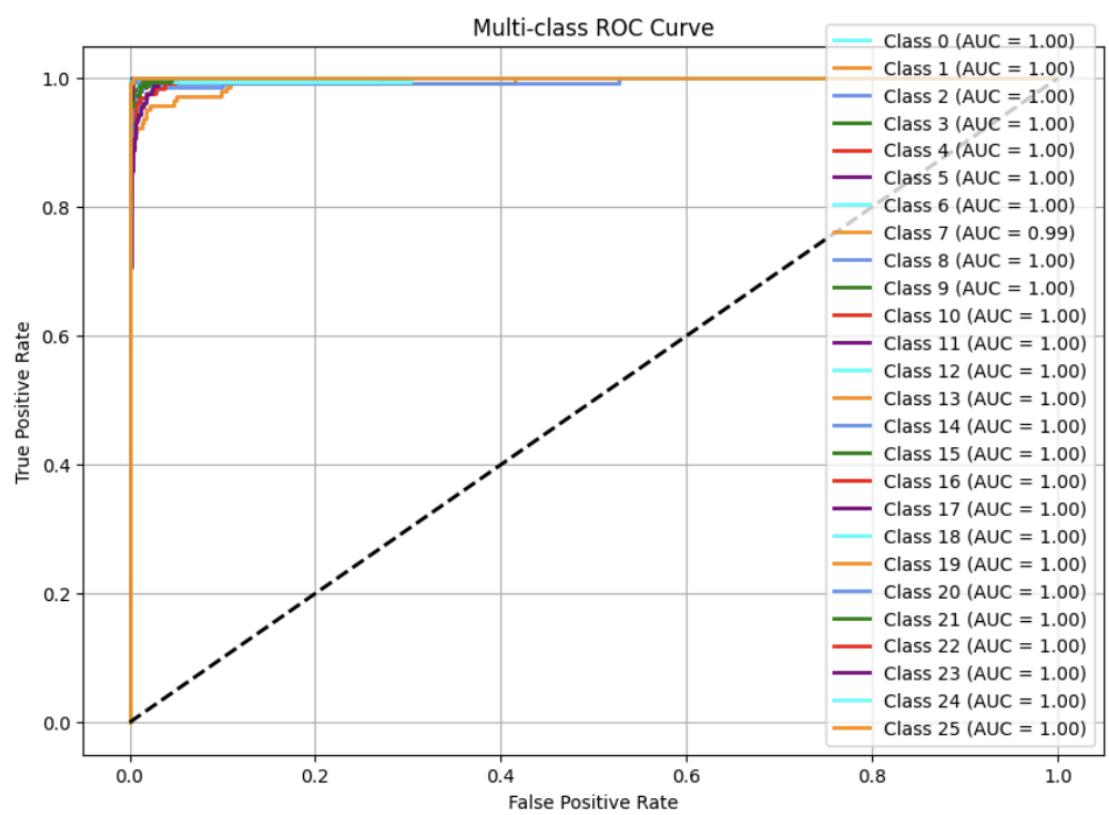
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC Curve
plt.figure(figsize=(10, 7))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'red', 'purple'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f'Class {i} (AUC = {roc_auc[i]:0.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multi-class ROC Curve")
plt.legend(loc="lower right")
plt.grid()
plt.show()

```



LABORATORY PROGRAM – 8

Implement Random forest ensemble method on a given dataset.

OBSERVATION BOOK

Random Forest:

(16-04-2024)

Algo: S1: Build a Random Forest

a) If the no. of examples, in the training set is N , take a sample of n -examples as random but with replacement from original data. This sample will be training set for generating the tree.

S2: If there are M input variables, ' m ' variables are selected at random, out of M , then best split on these ' m ' - used to split the node. The value at ' m ' is held constant during the generation of various trees in the forest.

S3: Each tree is grown to the largest extent possible.

For new data point find the prediction of each DTs & assign new data point to the category that wins the majority votes.

i.e. whenever a new data pt. is given give that DT to that particular DT.

Done

CODE WITH OUTPUT

```
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

# Load the dataset
df = pd.read_csv("iris.csv") # Adjust filename if needed

# Prepare data
X = df.drop(columns=["species"]) # Assuming 'species' is the target column
y = df["species"]

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

# Default Random Forest with 10 trees
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
acc_default = accuracy_score(y_test, y_pred_default)
conf_matrix_default = confusion_matrix(y_test, y_pred_default)

print(f"Default RF (10 trees) Accuracy: {acc_default}")
print("Confusion Matrix:\n", conf_matrix_default)

# Try different numbers of trees to find the best
best_acc = 0
best_n = 10
acc_list = []

for n in range(1, 101):
    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)
    acc_list.append((n, acc))
    if acc > best_acc:
        best_acc = acc
        best_n = n
        best_conf_matrix = confusion_matrix(y_test, y_pred)

print(f"\nBest Accuracy: {best_acc} using {best_n} trees")
print("Best Confusion Matrix:\n", best_conf_matrix)
# Plot accuracy vs number of trees
x_vals, y_vals = zip(*acc_list)
plt.plot(x_vals, y_vals, marker='o')
plt.title("Accuracy vs Number of Trees")
plt.xlabel("Number of Trees")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

```
Default RF (10 trees) Accuracy: 1.0
```

```
Confusion Matrix:
```

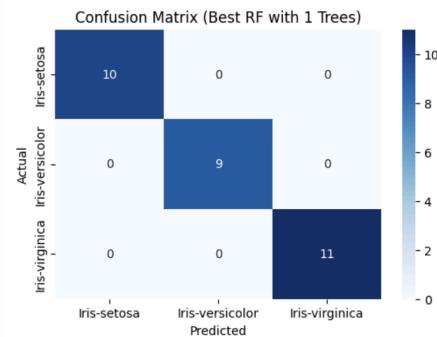
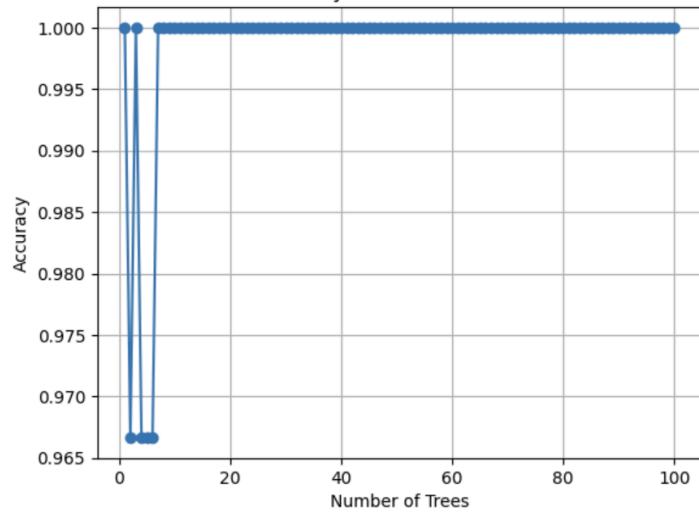
```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
Best Accuracy: 1.0 using 1 trees
```

```
Best Confusion Matrix:
```

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

Accuracy vs Number of Trees



LABORATORY PROGRAM – 9

Implement Boosting ensemble method on a given dataset.

OBSERVATION BOOK

<p>7/05/25</p> <p>Lab 8</p> <p>AdaBoost</p> <p>① Initial weight assigned to each item = $\frac{1}{6}$</p> <p>Hence first Decision stump classifier</p> <p><u>CGPA</u>: based on CGPA : $\begin{cases} \geq 9 & \rightarrow \text{Yes} \\ < 9 & \rightarrow \text{No} \end{cases}$</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>CGPA</th> <th>Predicted Job offer</th> <th>Actual Job offer</th> <th>Weight</th> </tr> </thead> <tbody> <tr> <td>≥ 9</td> <td>Yes</td> <td>Yes</td> <td>$\frac{1}{6}$</td> </tr> <tr> <td>$\{ < 9$</td> <td>No</td> <td>Yes</td> <td>$\frac{1}{6}$</td> </tr> <tr> <td>≥ 9</td> <td>Yes</td> <td>No</td> <td>$\frac{1}{6}$</td> </tr> <tr> <td>< 9</td> <td>No</td> <td>No</td> <td>$\frac{1}{6}$</td> </tr> <tr> <td>≥ 9</td> <td>Yes</td> <td>Yes</td> <td>$\frac{1}{6}$</td> </tr> <tr> <td>≥ 9</td> <td>Yes</td> <td>Yes</td> <td>$\frac{1}{6}$</td> </tr> </tbody> </table> <p>$\epsilon_{\text{CGPA}} = 2 \times \frac{1}{6} = 0.333$</p> <p>Weight of each weak classifier:</p> $\alpha_{\text{CGPA}} = \frac{1}{2} \frac{\ln(1 - \epsilon_{\text{CGPA}})}{\epsilon_{\text{CGPA}}} = \frac{1}{2} \frac{\ln(1 - 0.333)}{0.333}$ <p>$\alpha_{\text{CGPA}} = 0.347$</p> <p>Normalizing factor:</p> $Z_{\text{CGPA}} = \frac{1}{6} \times 4 \times e^{-0.347} + \frac{1}{6} \times 2 \times e^{0.347}$ <p>$Z_{\text{CGPA}} = 0.9428$</p>	CGPA	Predicted Job offer	Actual Job offer	Weight	≥ 9	Yes	Yes	$\frac{1}{6}$	$\{ < 9$	No	Yes	$\frac{1}{6}$	≥ 9	Yes	No	$\frac{1}{6}$	< 9	No	No	$\frac{1}{6}$	≥ 9	Yes	Yes	$\frac{1}{6}$	≥ 9	Yes	Yes	$\frac{1}{6}$	<p>update weights of all data instances:</p> $wt(d_i)_{t+1} = \frac{\frac{1}{6} \times e^{-0.347}}{0.9428} = 0.1249$ $wt(d_i)_{t+1} = \frac{\frac{1}{6} \times e^{0.347}}{0.9428} = 0.2501$ <p>$E_{\text{Interact}} = 1 \times 0.2501 = 0.2501$</p> <p>Interact: Yes \rightarrow Yes No \rightarrow No</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Interact</th> <th>Predicted Job offer</th> <th>Actual Job offer</th> <th>Weight</th> </tr> </thead> <tbody> <tr> <td>Yes</td> <td>Yes</td> <td>Yes</td> <td>0.1249</td> </tr> <tr> <td>\rightarrow No</td> <td>No</td> <td>Yes</td> <td>0.2501</td> </tr> <tr> <td>No</td> <td>No</td> <td>No</td> <td>0.2501</td> </tr> <tr> <td>No</td> <td>No</td> <td>No</td> <td>0.1249</td> </tr> <tr> <td>Yes</td> <td>Yes</td> <td>Yes</td> <td>0.1249</td> </tr> <tr> <td>Yes</td> <td>Yes</td> <td>Yes</td> <td>0.1249</td> </tr> </tbody> </table> <p>$E_{\text{Interact}} = 1 \times 0.2501 = 0.2501$</p> <p>$\alpha_{\text{Interact}} = \frac{1}{2} \frac{\ln(1 - E_{\text{Interact}})}{E_{\text{Interact}}}$</p> <p>$\alpha_{\text{Interact}} = \frac{1}{2} \frac{\ln(1 - 0.2501)}{0.2501}$</p> <p>$\alpha_{\text{Interact}} = 0.5490$</p>	Interact	Predicted Job offer	Actual Job offer	Weight	Yes	Yes	Yes	0.1249	\rightarrow No	No	Yes	0.2501	No	No	No	0.2501	No	No	No	0.1249	Yes	Yes	Yes	0.1249	Yes	Yes	Yes	0.1249
CGPA	Predicted Job offer	Actual Job offer	Weight																																																						
≥ 9	Yes	Yes	$\frac{1}{6}$																																																						
$\{ < 9$	No	Yes	$\frac{1}{6}$																																																						
≥ 9	Yes	No	$\frac{1}{6}$																																																						
< 9	No	No	$\frac{1}{6}$																																																						
≥ 9	Yes	Yes	$\frac{1}{6}$																																																						
≥ 9	Yes	Yes	$\frac{1}{6}$																																																						
Interact	Predicted Job offer	Actual Job offer	Weight																																																						
Yes	Yes	Yes	0.1249																																																						
\rightarrow No	No	Yes	0.2501																																																						
No	No	No	0.2501																																																						
No	No	No	0.1249																																																						
Yes	Yes	Yes	0.1249																																																						
Yes	Yes	Yes	0.1249																																																						

$$Z_{\text{Interact}} = 0.1249 * 4 * e^{-0.549} + 0.2501 * 1 * e^{-0.549} + 0.2501 * 1 * e^{0.549}$$

$$Z_{\text{Interact}} = 0.866$$

$$\text{wt}(d_i)_{i+1} = \frac{0.1249 * e^{-0.549}}{0.866} = 0.0832$$

$$\text{wt}(d_i)_{i+1} = \frac{0.2501 * e^{-0.549}}{0.866} = 0.1667$$

$$\text{wt}(d_i)_{i+1} = \frac{0.2501 * e^{0.549}}{0.866} = 0.5001$$

Practical Knowledge: Good → Yes
Average → No

Prk	Predicted	Actual	Weight
G	Yes	Yes	0.0832
AG	Yes	Yes	0.5001
A	No	No	0.1667
A	No	No	0.0832
G	Yes	Yes	0.0832
G	Yes	Yes	0.0832

No instances are misclassified

Communication Skill Good → Yes
Moderate → No

Comm Skill	Predicted	Actual	Weight
G	Yes	Yes	0.0832
M	No	Yes	0.5001
M	No	No	0.1667
G	Yes	No	0.0832
M	No	Yes	0.0832
M	No	Yes	0.0832

$$E_{CS} = 1 * 0.5001 + 3 * 0.0832 = 0.7497$$

$$\delta_{CS} = \frac{1}{2} \frac{\ln(1 - 0.7497)}{0.7497} = -0.5485$$

$$Z_{CS} = 0.0832 * 1 * e^{(-0.5485)} + 0.1667 * 1 * e^{(-0.5485)} + 0.5001 * 1 * e^{(-0.5485)} + 0.0832 * 3 * e^{(-0.5485)} = 0.866$$

$$\text{wt}(d_i)_{i+1} = \frac{0.0832 * e^{(-0.5485)}}{0.866} = 0.1667$$

$$\text{wt}(d_i)_{i+1} = \frac{0.1667 * e^{(-0.5485)}}{0.866} = 0.3337$$

$$\text{wt}(d_i)_{i+1} = \frac{0.5001 * e^{(-0.5485)}}{0.866} = 0.3337$$

$$\text{wt}(d_i)_{i+1} = \frac{0.0832 * e^{(-0.5485)}}{0.866} = 0.0555$$

$$H_F(d_j) = \alpha_{CGPA} * 1 + 0.5490 * 1 - 0.5485 = 0.34$$

$$H_F(d_j) = 0.347 * 0 + 0.5490 * 0 - 0.5485 * 0 = 0.$$

$\alpha_{CGPA} = 0.347$	$\alpha_{intuad} = 0.5490$	$\alpha_{cs} = -0.5485$	Weighted Avg	Final Pred.
Yes	Yes	Yes	0.3475	Yes
No	No	No	0.0	No
Yes	No	No	0.347	Yes
No	No	Yes	-0.5485	No
Yes	Yes	No	0.896	Yes
Yes	Yes	No	0.896	Yes

② Best accuracy = 0.83 ~ 83%

No. of trees = 160

$$\text{Confusion matrix} = \begin{bmatrix} 10658 & 45 \\ 2023 & 1521 \end{bmatrix}$$

CODE WITH OUTPUT

```

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay

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

# Display basic info
print("First five rows:")

```

```

print(data.head())
print(f"\nDataset shape: {data.shape}")

# Define features and target
target_column = 'income_level'
y = data[target_column]
X = data.drop(columns=[target_column])

# Identify categorical vs numerical columns
categorical_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
print(f"\nNumerical columns: {numerical_cols}")
print(f"Categorical columns: {categorical_cols}")

# Preprocessor: scale numericals, one-hot encode categoricals
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ]
)

# Initial AdaBoost model with 10 estimators
pipeline = Pipeline([
    ('preprocess', preprocessor),
    ('clf', AdaBoostClassifier(n_estimators=10, random_state=42))
])

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

# Train and evaluate initial model
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
initial_acc = accuracy_score(y_test, y_pred)
print(f"Initial test accuracy (n_estimators=10): {initial_acc:.4f}")

# Hyperparameter tuning: find best n_estimators
tree_counts = list(range(10, 201, 10)) # 10,20,...,200
cv_scores = []
for n in tree_counts:
    model = Pipeline([
        ('preprocess', preprocessor),
        ('clf', AdaBoostClassifier(n_estimators=n, random_state=42))
    ])
    scores = cross_val_score(
        model, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1
    )
    mean_score = scores.mean()
    cv_scores.append(mean_score)
    print(f"n_estimators={n}: CV mean accuracy={mean_score:.4f}")

# Plot CV accuracy vs. number of estimators
plt.figure()
plt.plot(tree_counts, cv_scores, marker='o')
plt.title('AdaBoost CV Accuracy vs. n_estimators')
plt.xlabel('Number of Estimators')
plt.ylabel('CV Mean Accuracy')
plt.grid(True)
plt.tight_layout()
plt.show()

# Determine optimal number of trees
best_score = max(cv_scores)
best_n = tree_counts[cv_scores.index(best_score)]

```

```

print(f"\nBest CV accuracy={best_score:.4f} with n_estimators={best_n}")

# Retrain and evaluate best model
best_model = Pipeline([
    ('preprocess', preprocess),
    ('clf', AdaBoostClassifier(n_estimators=best_n, random_state=42))
])
best_model.fit(X_train, y_train)
y_best = best_model.predict(X_test)
best_test_acc = accuracy_score(y_test, y_best)
print(f"Test accuracy with best n_estimators ({best_n}): {best_test_acc:.4f}")

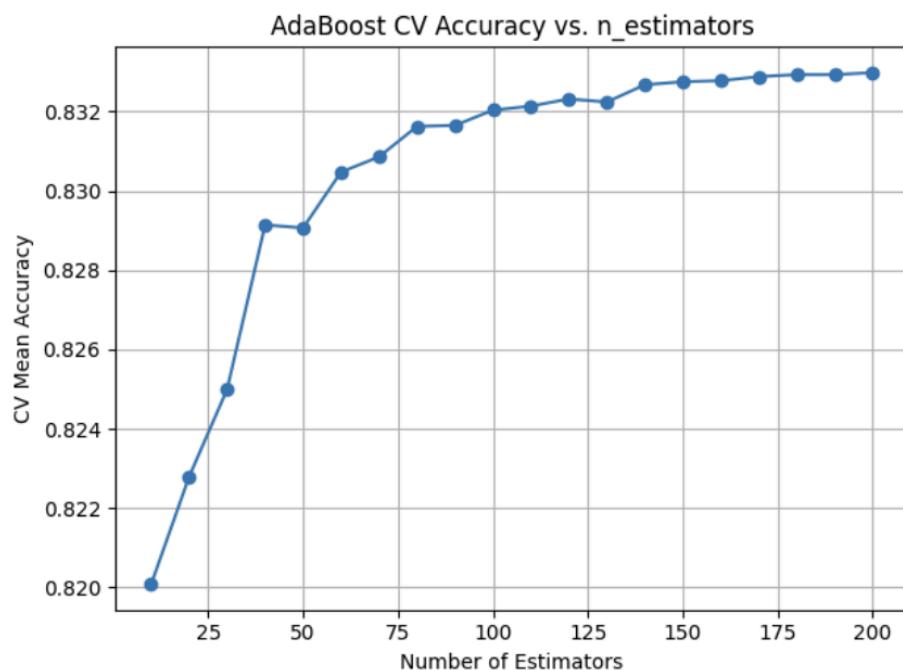
# Plot comparison of initial vs. best test accuracy
plt.figure()
plt.bar(['n=10', f'n={best_n}'], [initial_acc, best_test_acc])
plt.title('Test Accuracy: Initial vs. Optimized')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.tight_layout()
plt.show()

# Plot confusion matrix for best model
cm = confusion_matrix(y_test, y_best)
labels = best_model.named_steps['clf'].classes_
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
plt.figure()
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Best AdaBoost Model')
plt.tight_layout()
plt.show()

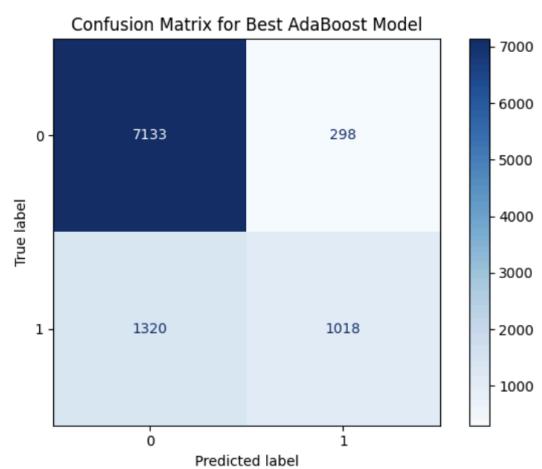
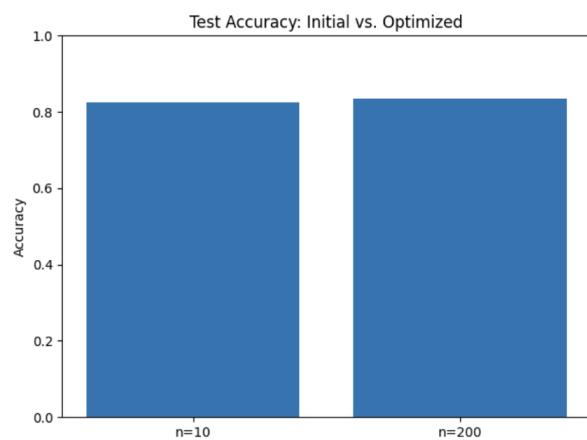
```

Dataset shape: (48842, 7)

Numerical columns: ['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']
Categorical columns: []
Initial test accuracy (n_estimators=10): 0.8257
n_estimators=10: CV mean accuracy=0.8201
n_estimators=20: CV mean accuracy=0.8228
n_estimators=30: CV mean accuracy=0.8250
n_estimators=40: CV mean accuracy=0.8291
n_estimators=50: CV mean accuracy=0.8291
n_estimators=60: CV mean accuracy=0.8305
n_estimators=70: CV mean accuracy=0.8309
n_estimators=80: CV mean accuracy=0.8316
n_estimators=90: CV mean accuracy=0.8316
n_estimators=100: CV mean accuracy=0.8320
n_estimators=110: CV mean accuracy=0.8321
n_estimators=120: CV mean accuracy=0.8323
n_estimators=130: CV mean accuracy=0.8322
n_estimators=140: CV mean accuracy=0.8327
n_estimators=150: CV mean accuracy=0.8327
n_estimators=160: CV mean accuracy=0.8328
n_estimators=170: CV mean accuracy=0.8329
n_estimators=180: CV mean accuracy=0.8329
n_estimators=190: CV mean accuracy=0.8329
n_estimators=200: CV mean accuracy=0.8330



Best CV accuracy=0.8330 with n_estimators=200
Test accuracy with best n_estimators (200): 0.8344



LABORATORY PROGRAM – 10

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

OBSERVATION BOOK

Lab 9 K Means			
<u>Iteration 1</u>			
Record Number	Close to C1 (1.0, 1.0)	Close to C2 (5.0, 7.0)	Assign to Cluster
R1(1.0, 1.0)	dist(R1, C1)=0.0	(R1, C2)=7.21	Cluster 1
R2(1.5, 2.0)	dist(R2, C1)=1.12	(R2, C2)=6.12	Cluster 1
R3(3.0, 4.0)	(R3, C1)=3.61	(R3, C2)=3.61	Cluster 1
R4(4.5, 5.0)	(R4, C1)=7.21	(R4, C2)=0.0	Cluster 2
R5(3.5, 5.0)	(R5, C1)=9.21	(R5, C2)=2.5	Cluster 2
R6(4.5, 5.0)	(R6, C1)=5.31	(R6, C2)=2.06	Cluster 2
R7(3.5, 4.5)	(R7, C1)=4.3	(R7, C2)=2.92	Cluster 2
Cluster 1 { R1, R2, R3 } Cluster 2 { R4, R5, R6, R7 }			
New centroids are:			
$C_1 = (1.0 + 1.5 + 3.0)/3, (1.0 + 2.0 + 4.0)/3$			
$= 5.5/3, 7.0/3$			
$= 1.83, 2.33$			
$C_2 = (5.0 + 3.5 + 4.5 + 3.5)/4, (7.0 + 5.0 + 4.0 + 5.0)/4$			
$= 16.5/4, 21.5/4$			
$= 4.12, 5.37$			
<u>Iteration 2:</u>			
Record No.	Close to C1 (1.83, 2.33)	Close to C2 (4.12, 5.37)	Assign to Cluster
R1(1.0, 1.0)	(R1, C1)=1.57	(R1, C2)=5.37	C1
R2(1.5, 2.0)	(R2, C1)=0.47	(R2, C2)=4.27	C1
R3(3.0, 4.0)	(R3, C1)=2.04	(R3, C2)=1.77	C2
R4(4.5, 5.0)	(R4, C1)=5.64	(R4, C2)=1.85	C2
R5(3.5, 5.0)	(R5, C1)=3.15	(R5, C2)=0.72	C2
R6(4.5, 5.0)	(R6, C1)=3.78	(R6, C2)=0.53	C2
R7(3.5, 4.5)	(R7, C1)=2.74	(R7, C2)=1.07	C2
Cluster 1 (R1, R2) & Cluster 2 (R3, R4, R5, R6, R7)			
New centroids are:			
$C_1 = (1.0 + 1.5 + 3.0)/2, (1.0 + 2.0 + 4.0)/2$			
$= 2.5/2, 3.0/2$			
$= 1.25, 1.5$			
$C_2 = (3.0 + 5.0 + 3.5 + 4.5 + 3.5)/5, (9.0 + 7.0 + 5.0 + 4.5 + 4.5)/5$			
$= 19.5/5, 27.5/5$			
$= 3.9, 5.5$			

CODE WITH OUTPUT

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def load_data(csv_path='iris.csv'):
    """
    Try loading from csv_path; if not found, load via sklearn.
    Expects columns: sepal_length, sepal_width, petal_length, petal_width, species.
    Returns DataFrame with a 'species' column.
    """
    try:
        df = pd.read_csv(csv_path)
        # Fixed typo here: use c.strip().replace, not ace()
        df.columns = [c.strip().replace(' ', '_') for c in df.columns]
    except FileNotFoundError:
        iris = load_iris()
        df = pd.DataFrame(
            data=np.c_[iris['data'], iris['target']],
            columns=iris['feature_names'] + ['target']
        )
        df.columns = [c.strip().replace(' (cm)', "").replace(' ', '_')
                     for c in df.columns]
        df['species'] = df['target'].map(lambda x: iris['target_names'][int(x)])
    return df

def preprocess(df):
    """
    Select only petal_length & petal_width, then standard-scale.
    Returns scaled numpy array.
    """
    X = df[['petal_length', 'petal_width']].values
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, scaler

def plot_elbow(X_scaled, max_k=10):
    """
    Compute KMeans inertia for k=1..max_k and plot the elbow curve.
    Returns list of inertias.
    """
    inertias = []
    ks = range(1, max_k + 1)
    for k in ks:
        km = KMeans(n_clusters=k, random_state=42)
        km.fit(X_scaled)
        inertias.append(km.inertia_)
    plt.figure(figsize=(6, 4))
    plt.plot(ks, inertias, 'o-', linewidth=2)
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal k')
    plt.xticks(ks)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()
    return inertias

def run_kmeans(X_scaled, k):
    """
    Fit KMeans with k clusters, return labels and fitted model.
    """
    pass
```

```

km = KMeans(n_clusters=k, random_state=42)
labels = km.fit_predict(X_scaled)
return km, labels

def plot_confusion(df, labels, k):
    """
    Builds and displays a confusion matrix comparing true species vs. cluster.
    """
    species_names = df['species'].unique()
    species_to_num = {name: idx for idx, name in enumerate(species_names)}
    true_nums = df['species'].map(species_to_num)

    cm = confusion_matrix(true_nums, labels)
    disp = ConfusionMatrixDisplay(
        confusion_matrix=cm,
        display_labels=[f"Cluster {i}" for i in range(k)])
    fig, ax = plt.subplots(figsize=(6, 6))
    disp.plot(ax=ax, cmap='Blues', colorbar=True)
    ax.set_xlabel('Predicted Cluster')
    ax.set_ylabel('True Species')
    plt.title('K-Means Clustering Confusion Matrix')
    plt.tight_layout()
    plt.show()

    cm_df = pd.DataFrame(
        cm,
        index=[f"True: {name}" for name in species_names],
        columns=[f"Cluster {i}" for i in range(k)])
    )
    print("\nConfusion Matrix (counts):")
    print(cm_df)

def main():
    # 1) Load data
    df = load_data('iris.csv')
    if 'species' not in df.columns:
        print("Error: 'species' column not found.")
        return

    # 2) Preprocess
    X_scaled, scaler = preprocess(df)

    # 3) Elbow plot to decide k
    print("Generating elbow plot to find optimal k...")
    inertias = plot_elbow(X_scaled, max_k=10)

    # 4) From the elbow you'll typically see a bend at k=3
    optimal_k = 3
    print(f"Choosing k = {optimal_k} (you can adjust this based on the plot).")

    # 5) Run K-Means and assign clusters
    km_model, labels = run_kmeans(X_scaled, optimal_k)
    df['cluster'] = labels

    # 6) Visualize clusters in feature space
    plt.figure(figsize=(6, 4))
    plt.scatter(
        X_scaled[:, 0], X_scaled[:, 1],
        c=labels, cmap='viridis', edgecolor='k', s=50
    )
    centroids = km_model.cluster_centers_
    plt.scatter(
        centroids[:, 0], centroids[:, 1],
        marker='X', c='red', s=200, label='Centroids'
    )
    plt.xlabel('Scaled Petal Length')

```

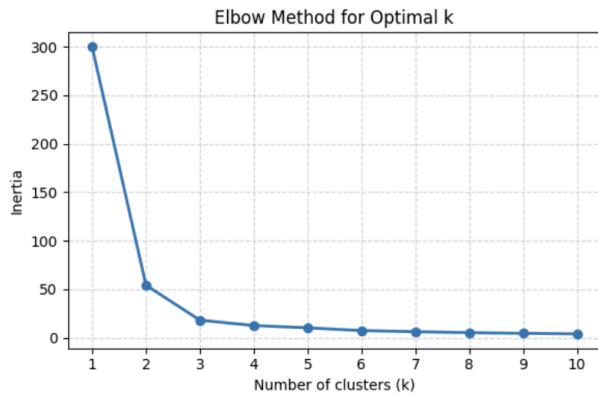
```

plt.ylabel("Scaled Petal Width")
plt.title(f'K-Means Clusters (k={optimal_k})')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

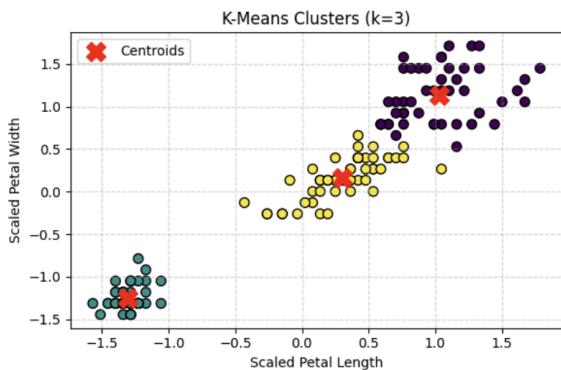
# 7) Confusion matrix vs. true species
plot_confusion(df, labels, optimal_k)

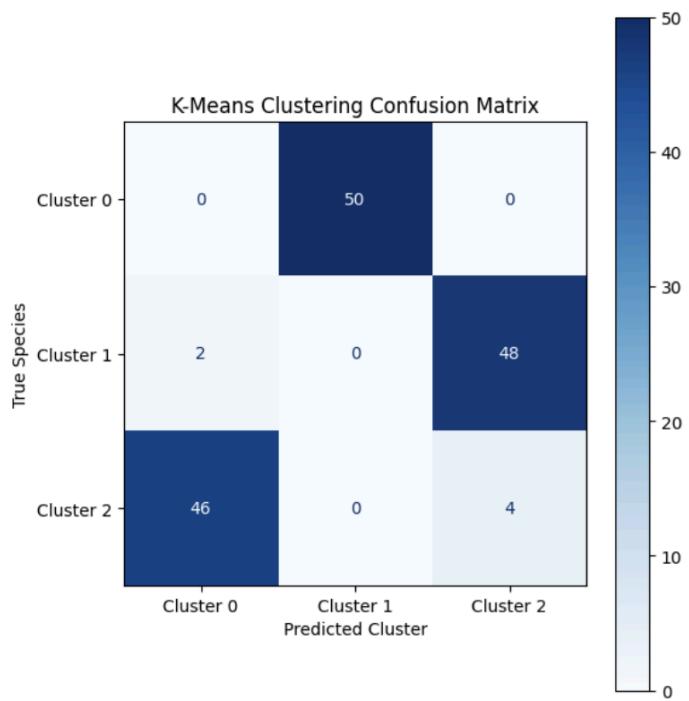
if __name__ == "__main__":
    main()

```



Choosing k = 3 (you can adjust this based on the plot).

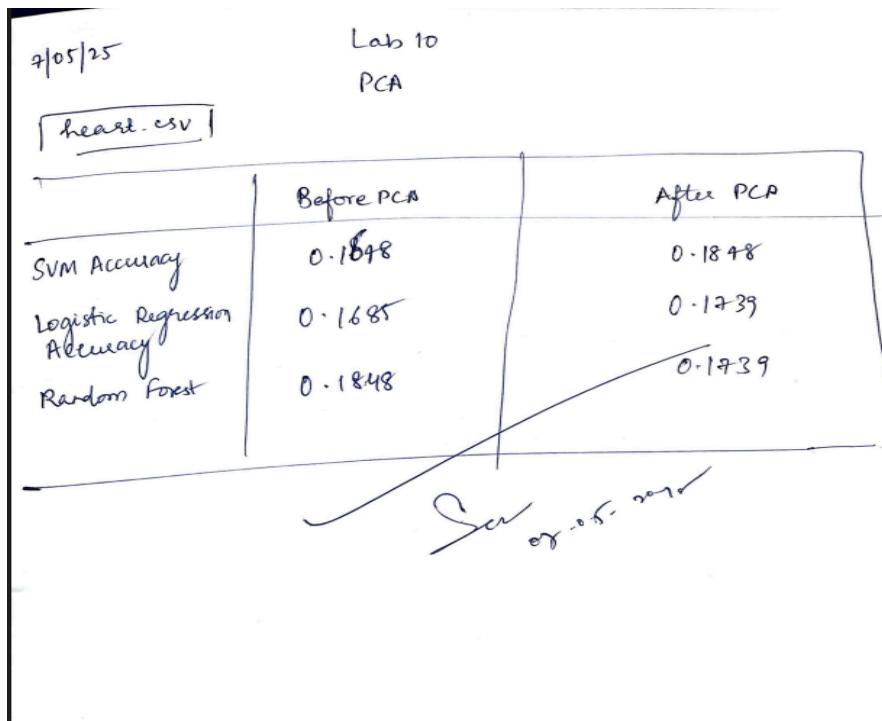




LABORATORY PROGRAM – 11

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

OBSERVATION BOOK



CODE WITH OUTPUT

```
import pandas as pd

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

# Step 3: Split Features and Target
X = df.drop("target", axis=1)
y = df["target"]

# Step 4: Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

categorical_features = ["cp", "thal", "slope"]
numerical_features = [col for col in X.columns if col not in categorical_features]

preprocessor = ColumnTransformer(transformers=[
    ("num", StandardScaler(), numerical_features),
    ("cat", OneHotEncoder(), categorical_features)
])

# Step 5: Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

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

# Step 7: Train and Evaluate Models (Before PCA)
print("Accuracy Before PCA:")
results = {}
for name, model in models.items():
    pipeline = Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("classifier", model)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"{name}: {acc:.4f}")

from sklearn.decomposition import PCA

print("\nAccuracy After PCA (n_components=5):")
pca_results = {}

for name, model in models.items():
    pipeline_pca = Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("pca", PCA(n_components=5)),
        ("classifier", model)
    ])
    pipeline_pca.fit(X_train, y_train)
    y_pred_pca = pipeline_pca.predict(X_test)
    acc_pca = accuracy_score(y_test, y_pred_pca)
    pca_results[name] = acc_pca
    print(f"{name}: {acc_pca:.4f}")

```

```

→  Accuracy Before PCA:
  Logistic Regression: 0.9016
  SVM: 0.8525
  Random Forest: 0.8361

  Accuracy After PCA (n_components=5):
  Logistic Regression: 0.8689
  SVM: 0.8689
  Random Forest: 0.8852

```