

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



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

Submitted by

Shuvam Rajbanshi (1BM22CS275)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by Shuvam Rajbanshi (**1BM22CS275**), 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.

Lab Faculty Incharge Name:
Spoorthi D M
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.	5
3	24.03.25	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	10
4	17.03.25	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset.	14
5	24.03.25	Build Logistic Regression Model for a given dataset.	19
6	07.04.25	Build KNN Classification model for a given dataset.	23
7	21.04.25	Build Support vector machine model for a given dataset.	27
8	05.05.25	Implement Random forest ensemble method on a given dataset.	31
9	05.05.25	Implement Boosting ensemble method on a given dataset.	33
10	05.05.25	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	36
11	05.05.25	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	42

Github Link: <https://github.com/Shuvam00/ML>

LABORATORY PROGRAM – 1

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

OBSERVATION BOOK

- ```
import pandas as pd
df = pd.DataFrame({ "USN": [22, 21, 23, 24, 25],
 "Name": ["Sam", "Ram", "Dam", "Rakul", "rakshi"],
 "Marks": [45, 78, 24, 90, 60]
 })
df
```
- Method 2: Import datasets from sklearn.datasets.datasets
- ```
from sklearn.datasets import load_diabetes
dia = load_diabetes()
df = pd.DataFrame(dia.data, columns = dia.feature_names)
df
```
- Method 3: Import dataset from a specific .csv file
- ```
filepath = "/content/sample-sales-data.csv"
pd.read_csv(filepath)
df.head()
```
- Method 4: Downloading datasets from existing dataset repositories like Kaggle, UCI, MSLab, KEEL, etc
- ```
df = pd.read_csv("/content/Dataset of Diabetes.csv")
```

Using the code given in the above slides, do the exercise of the stock Market Data Analysis! considering the following

1. HDFC Bank Ltd ICICI Bank Ltd, Kotak Mahindra Bank Ltd
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

2. start_date : 2024-01-01, End_Date = 2024-12-30

3. Plot the closing price and daily returns for all three banks mentioned.

import finance as yf

import pandas as pd

import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

start = "2024-01-01"

end = "2024-12-30"

data = yf.download(tickers, start=start, end=end)

print(data.head())

closing_prices = data['Close']

closing_prices = closing_prices.fillna()

daily_returns = closing_prices.pct_change(1)

moving_avg = closing_prices.rolling(window=50).mean()

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	42	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26	4.5	62	4.9	37	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50	4.7	46	4.9	42	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50	4.7	46	4.9	42	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33	7.1	46	4.9	49	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_Patien AGE   Urea    Cr  \
count 1000.000000 1.000000e+03 1000.000000 1000.000000 1000.000000
mean 348.500000 2.785514e+05 53.528000 5.124743 68.943000
std 240.357273 3.380758e+06 8.799241 2.935165 59.984747
min 1.000000 1.210000e+02 28.000000 8.500000 6.000000
25% 125.750000 1.466375e+04 51.000000 3.700000 48.000000
50% 300.500000 3.439550e+04 55.000000 4.680000 68.000000
75% 550.250000 4.538425e+04 59.000000 5.700000 73.000000
max 800.000000 7.343500e+07 79.000000 36.500000 820.000000

      HbA1c    Chol     TG    HDL    LDL  \
count 1000.000000 1000.000000 1000.000000 1000.000000 1000.000000
mean 5.281160 4.052320 2.549610 1.284750 2.029790
std 1.554403 1.301738 1.401176 0.668414 1.115162
min 0.900000 0.300000 0.300000 0.200000 0.300000
25% 6.500000 4.000000 2.500000 1.200000 2.000000
50% 8.000000 4.000000 2.000000 1.100000 2.000000
75% 10.200000 5.500000 2.900000 1.300000 3.300000
max 16.000000 18.300000 13.800000 9.900000 9.900000

      VLDL     BMI
count 1000.000000 1000.000000
mean 1.854798 29.578029
std 3.663199 4.962388
min 0.100000 19.300000
25% 6.700000 26.800000
50% 9.900000 30.200000
75% 15.500000 33.800000
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_Patien	AGE	Urea	Cr	HbA1c	Chol	TG	HDl	LDL	VLDL	BMI
0	502	17975	59	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	59	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0
3	689	87656	59	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_Y
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	Rn_Patien	Age	Urea	Cr	HbA1c	Chol
0	0.627034	0.000237	0.508475	0.109375	0.050378	0.264901
1	0.918048	0.000452	0.181095	0.184167	0.079529	0.204901
2	0.524406	0.000634	0.508475	0.109375	0.050378	0.204901
3	0.849812	0.001168	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	0.814464	0.228893	0.114583	0.011461	0.173913	False
1	0.881481	0.092764	0.187500	0.014327	0.130190	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	Rn_Patien	Age	Urea	Cr	HbA1c	Chol
0	0.572140	-0.074747	-0.401144	-0.144781	-0.382672	-1.334985
1	1.541852	-0.009348	-1.130017	-0.212954	-0.115054	-1.334983
2	0.530068	-0.005809	-0.401144	-0.144781	-0.382672	-1.334983
3	1.412958	-0.854126	-0.401144	-0.144781	-0.382672	-1.334983
4	0.630463	-0.069939	-2.234095	0.673299	-0.382672	-1.334983

TG	HDL	LDL	VLDL	BMI	Gender_M	Gender_F
0	-1.035084	1.810756	-1.885457	-0.309558	-1.124622	False
1	-0.678063	0.158893	-0.457348	-0.542649	-1.126259	True
2	-1.035084	1.810756	-1.885457	-0.309558	-1.124622	False
3	-1.035084	1.810756	-1.885457	-0.309558	-1.124622	False
4	-0.963680	0.613180	-0.547121	-0.397167	-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

LAB 1:	
	i) To load .csv into the data frame ii) To display info of all columns iii) To display statistical information of all numerical iv) To display the count of unique values for "ocean proximity" v) To display which all in database have missing values count greater than zero.
	<pre>import pandas as pd df = pd.read_csv("housing.csv")</pre>
	i) print(df.info) ii) print(df.describe()) iii) df['Ocean proximity'].count iv) missing_values = df.isnull().sum() print(missing_values[missing_values > 0])
	<pre>import pandas as pd import numpy as np</pre> <pre>diabetes_df = pd.read_csv("path/Diabetes.csv") adult_df = pd.read_csv("path/adult.csv")</pre>
	<pre>def preprocess_data(df): missing_values = df.isnull().sum() print(missing_values[missing_values > 0]) numerical_cols = df.select(include).cols categorical_cols = df.select(include).cols</pre>

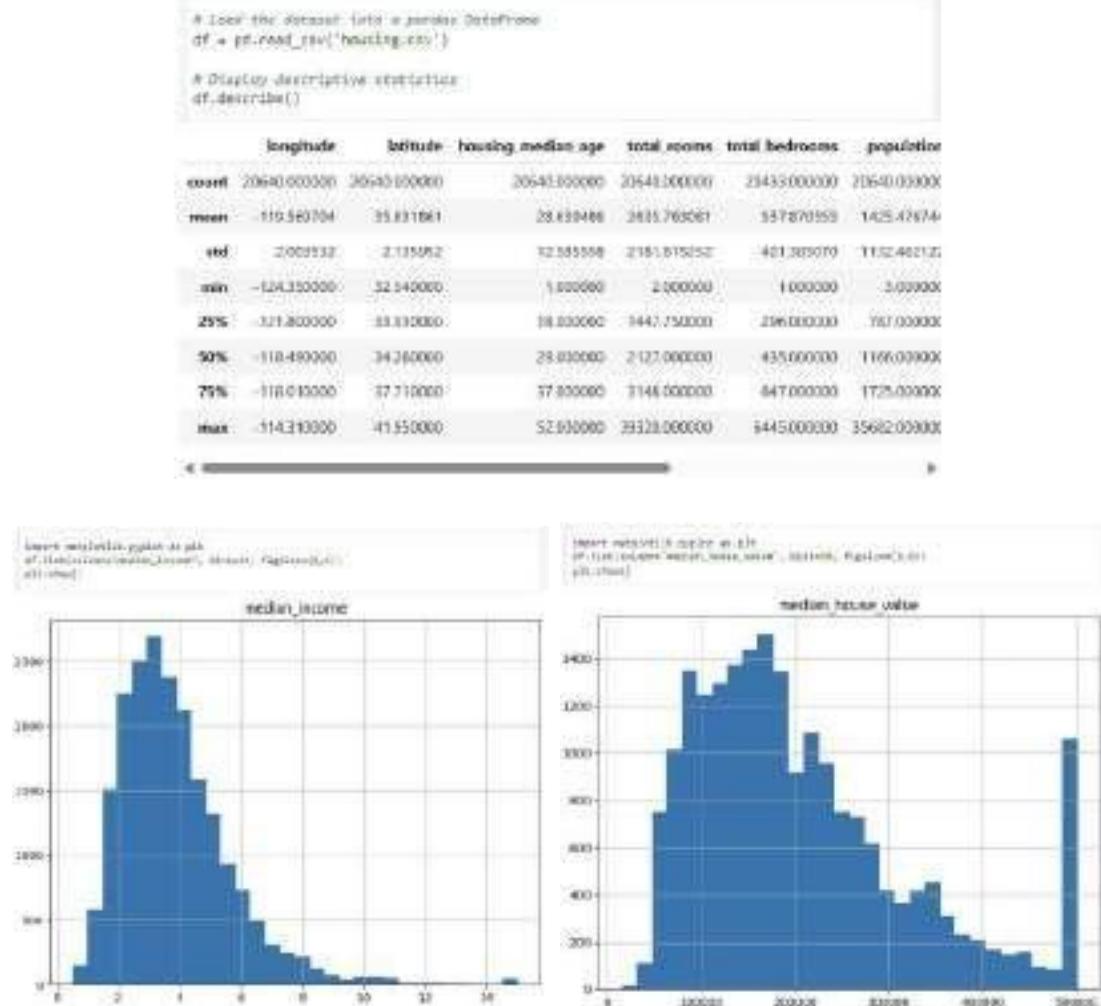
inputer = SimpleImputer (strategy = 'most_frequent')
df['numerical_cols'] = inputer.fit_transform(df[['numerical_cols']])

encoder = OneHotEncoder (drop = 'first', sparse_output = False)

encoded_categorical_df = pd.DataFrame
(encoded_categorical_data, columns = encoder.get_feature_names_out(categorical_cols)).



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 26543.000000 20541.000000 20541.000000 23435.000000 20540.000000
mean 119.560704 35.831861 28.839486 2835.765261 53787.0559 1425.426744
std 2.002532 2.135052 12.585558 2151.815252 401.385079 11.52.402123
min -124.350000 32.540000 1.000000 2.000000 1.000000 3.000000
25% -118.480000 34.260000 28.000000 2127.000000 296.000000 767.000000
50% -118.480000 34.260000 28.000000 2127.000000 435.000000 1166.000000
75% -118.010000 37.710000 37.800000 3148.000000 647.000000 1725.000000
max -514.310000 41.550000 52.930000 3932.000000 5445.000000 35682.000000

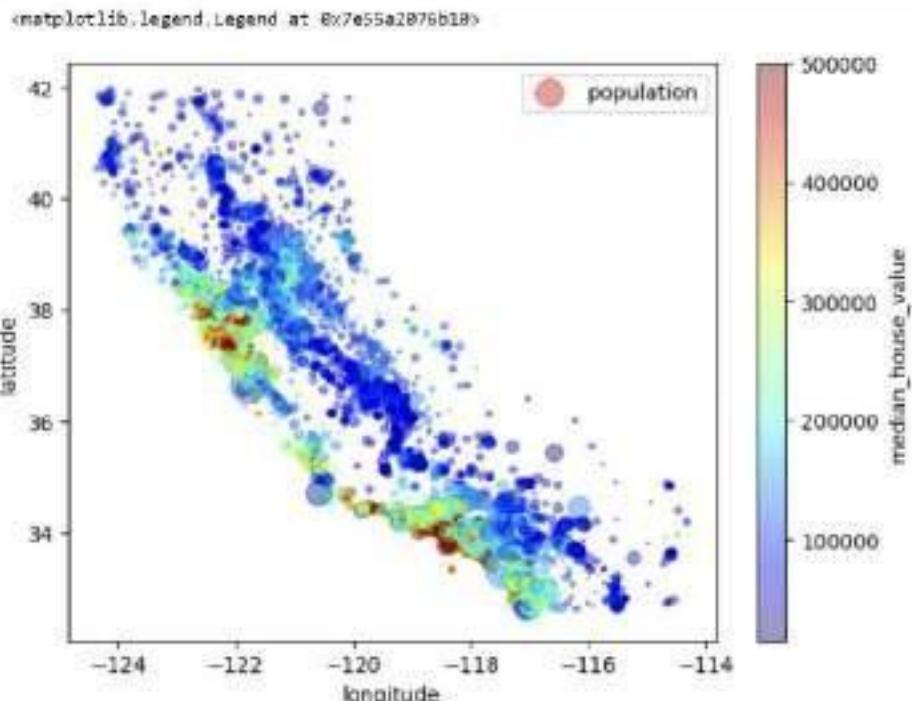
import numpy as np
df['income_cat'] = np.ceil(np.log1p(df['median_income'])) - 1

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title('median_income')
plt.hist(df['median_income'], bins=100)
plt.subplot(1, 2, 2)
plt.title('median_house_value')
plt.hist(df['median_house_value'], bins=100)
```

```
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()
```

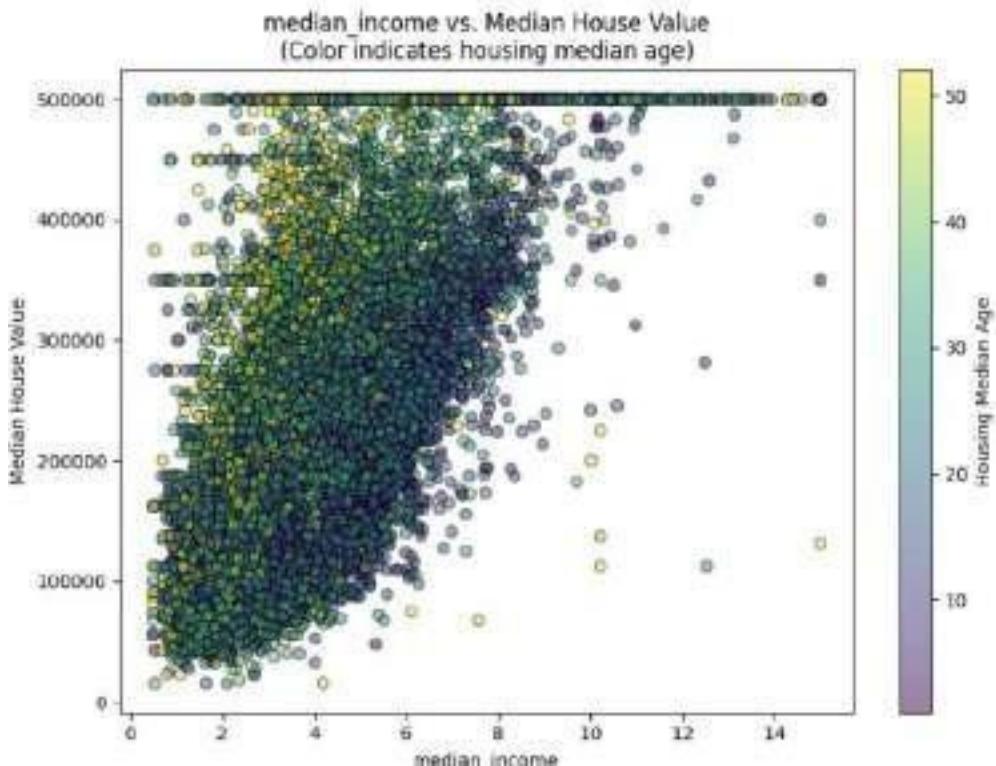


```

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,6))
# differentiation by using "housing_median_age" for the color
scatter = plt.scatter(housing_numeric[msk_feature],
                      housing_numeric["median_house_value"],
                      alpha=0.5,
                      c=housing_numeric["housing_median_age"],
                      cmap='viridis',
                      edgecolor='k')
plt.xlabel(msk_feature)
plt.ylabel("Median House Value")
plt.title(f'{msk_feature} vs. Median house value\n(color indicates housing median age)")
# add a colorbar to explain the color mapping
cbar = plt.colorbar(scatter)
cbar.ax.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

Q. Create a decision tree for a weather data using the ID3 algorithm (Iterative Dichotomiser 3)

```
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]
```

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

```
tree = {best_feature: {}}
```

```
for value, subset in data.groupby(best_feature)
```

```
subset = subset.drop(columns=[best_feature])
```

```
tree[best_feature][value] = id3(subset, [f for f in features  
if f != best_feature])
```

```
data = {
```

```
'outlook': ['sunny', 'sunny', 'overcast', 'rainy', 'rainy']
```

```
'temperature': ['hot', 'hot', 'hot', 'mild', 'cold']
```

```
'humidity': ['high', 'high', 'high', 'high', 'normal']
```

```
'wind': ['weak', 'strong', 'weak', 'weak', 'weak']
```

```
'label': ['no', 'no', 'yes', 'yes', 'yes']}
```

```
}
```

```
df = pd.DataFrame(data)
```

```
features = ['outlook', 'temperature', 'humidity', 'wind']
```

```
tree = id3(df, features)
```

```
print(tree)
```

```
• Output: {
```

```
    'outlook': {
```

```
        'sunny': {
```

```
            'humidity': {
```

```
                'high': 'no',
```

```
                'normal': 'yes'
```

```
}
```

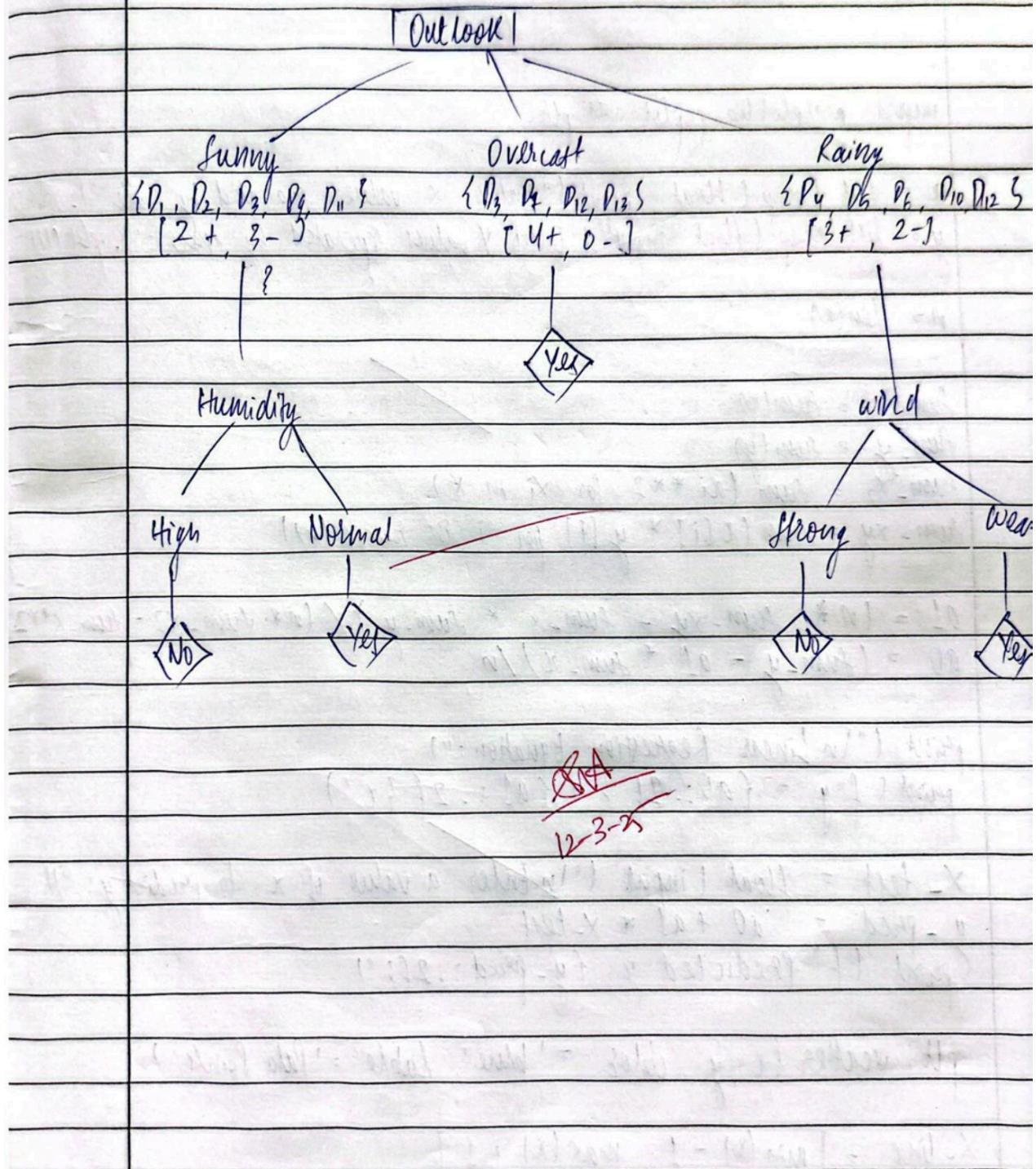
```
{
```

```
    'overcast': 'yes',
```

```
    'normal': {
```

Decision Tree

$D_1, D_2, D_3 \dots D_{14}$



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]

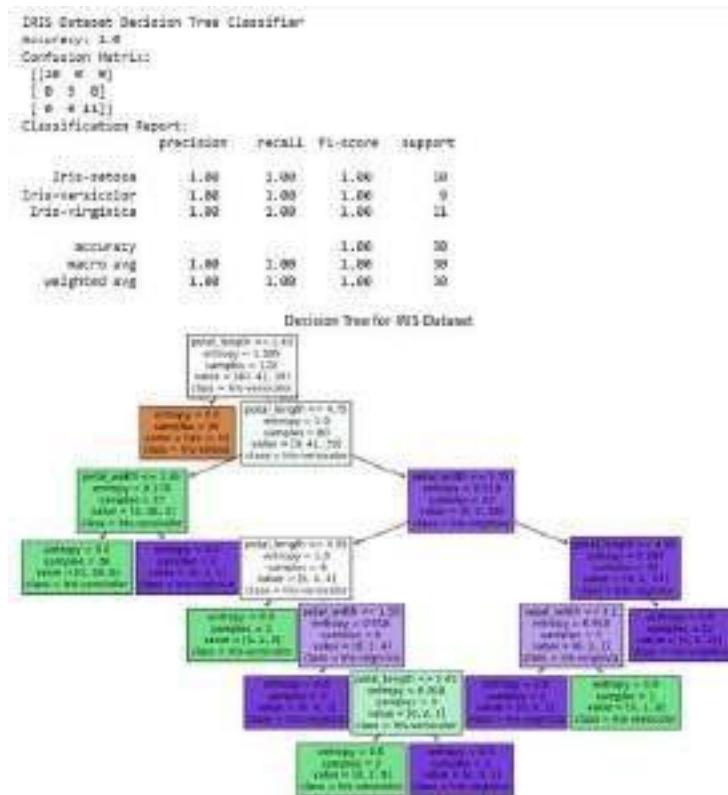
# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 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()
```



```

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

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

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

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

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

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

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

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

```

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

```

```

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

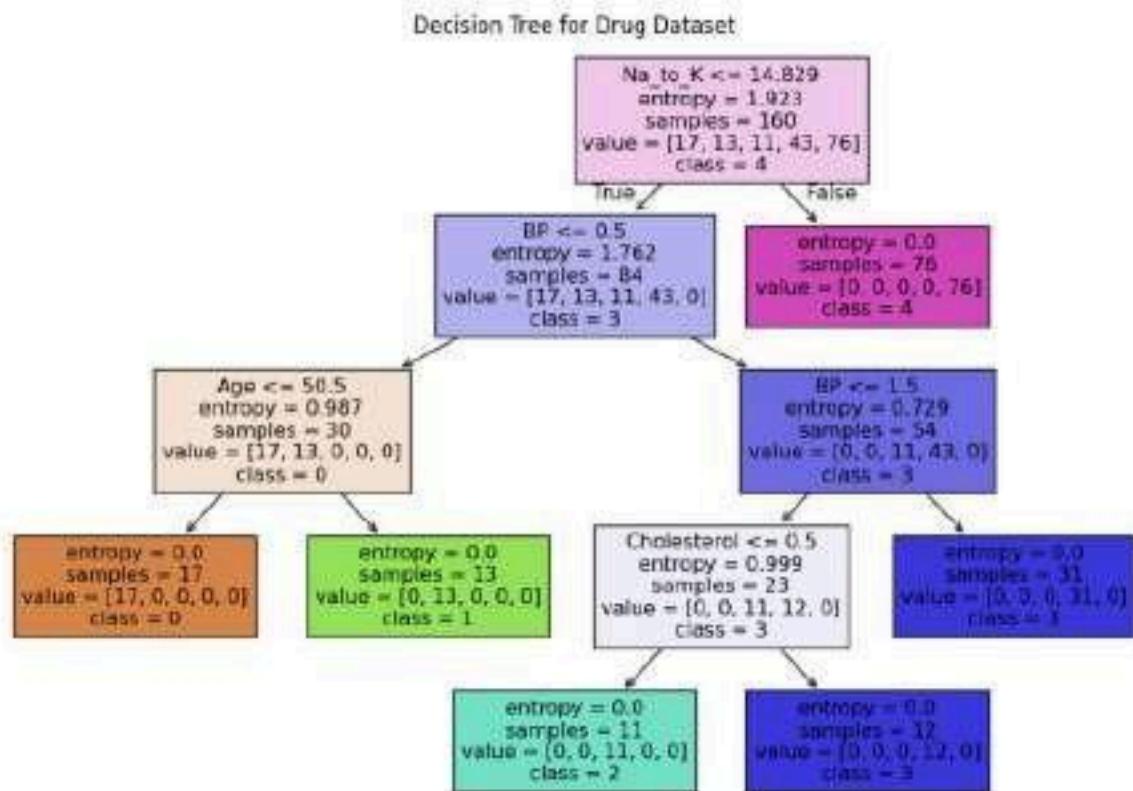
```

```

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

   accuracy                           1.00      48
  macro avg       1.00      1.00      1.00      48
weighted avg     1.00      1.00      1.00      48

```



LABORATORY PROGRAM - 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

OBSERVATION BOOK

LAB - 3

Linear Regression:

```
import matplotlib.pyplot as plt
x = list(map(float, input("Enter x values separated by space:").split()))
y = list(map(float, input("Enter y values separated by space:").split()))
n = len(x)

sum_x = sum(x)
sum_y = sum(y)
sum_x2 = sum([xi ** 2 for xi in x])
sum_xy = sum([x[i] * y[i] for i in range(n)])
a1 = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x * sum_x)
a0 = (sum_y - a1 * sum_x) / n

print("The Linear Regression Equation is:")
print(f"y = {a0} + {a1}x")
```

~~x-test = float(input("Enter a value of x to predict y:"))~~

~~y-pred = a0 + a1 * x-test~~

~~print(f"Predicted y: {y-pred:.2f}")~~

```
plt.scatter(x, y, color='blue', label='Data Points')
x_line = [min(x) - 1, max(x) + 1]
```

```

plt.plot(x_line, y_line, color = 'red', label = 'Regression Line')
plt.scatter(x_true, y_pred, color = 'green', label = f'Prediction: {x_true}, {y_pred : .2f} ')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('Linear Regression Visualization')
plt.legend()
plt.grid(True)
plt.show()

```

Output :

Enter X values separated by space : 1 2 3 4 5

Enter Y values separated by space : 12 18 22 28 35

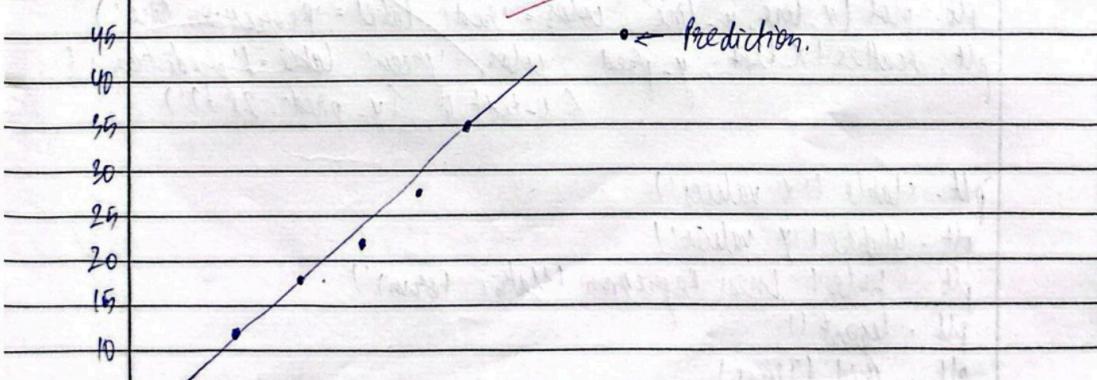
Linear Regression Equation :

$$y = 6.20 + 7.60x$$

Enter a value of x to predict y : 7

Predicted y : 49.40

LINEAR REGRESSION ~~VISUALIZATION~~



Output.

Enter X values separated by space : 1 2 3 4

Enter Y values separated by space : 1 3 4 7

Linear Regression Equation (Matrix Form):

$$y = -1.90 + 2.20x$$

Enter a value of x to predict y : 7

Predicted y : 13.90

LINEAR REGRESSION (MATRIX FORM)

14

- Data Points

12

- Regression Line

10

- Prediction (7.0, 13.90)

8

6

4

2

0

-2

0 1 2

$$(3(x^4) + \text{y}) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(1^4) + 1) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(2^4) + 3) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(3^4) + 4) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(4^4) + 7) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(7^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

$$(3(13.90^4) + 13.90) - 1.96 \cdot \sqrt{7} \rightarrow f = y$$

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]])

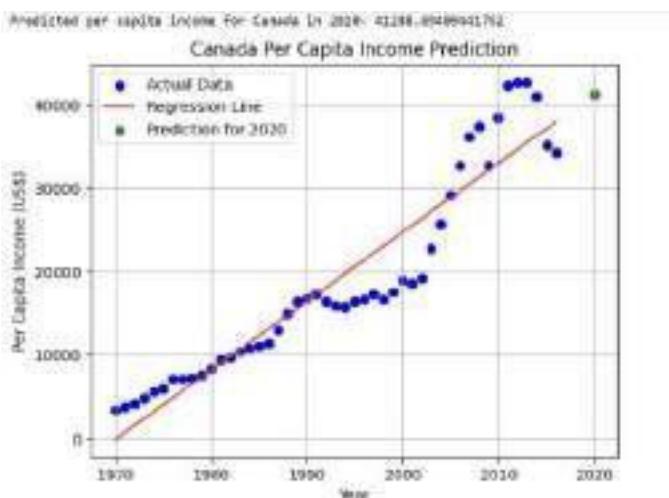
print("\nPredicted per capita income for Canada in 2020:", predicted_income[0])

# 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(income_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: 139988.88923969213



```

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')

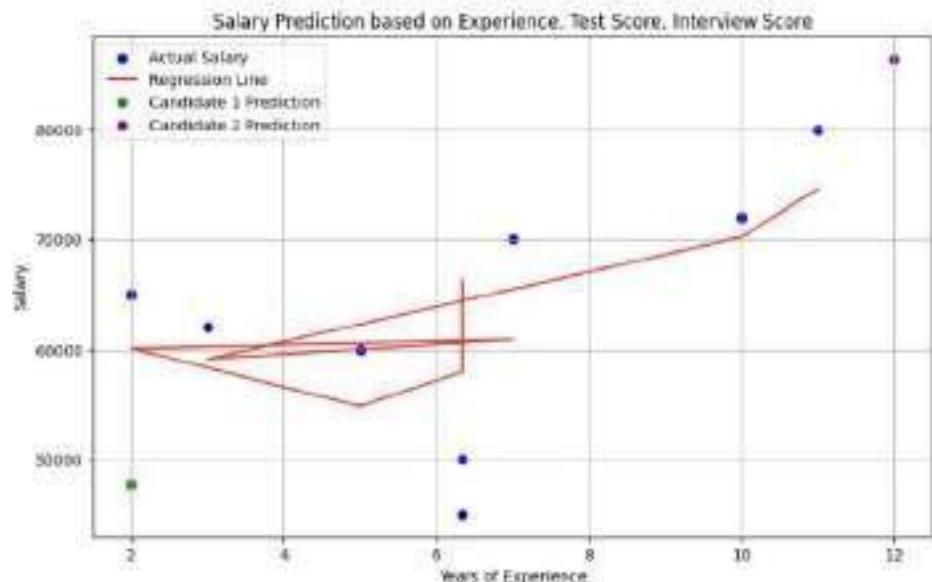
# Highlight predictions
plt.scatter(candidate1[0, 0], predicted_salary1, color='green', label='Candidate 1 Prediction')
plt.scatter(candidate2[0, 0], predicted_salary2, color='purple', label='Candidate 2 Prediction')

# 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

EDGE 12

LAB - 4

LOGISTIC REGRESSION:

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

- a. Write the logistic regression equation for this problem.
→ The logistic regression model follows:

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

$$z = a_0 + a_1 x$$

- b. Probability for 7 hours

→ for $x = 7$

import numpy as np

$a_0, a_1 = -5, 0.8$

$x = 7$

$\text{prob} = 1 / (1 + \text{np.exp}(-(a_0 + a_1 * x)))$

print (prob)

- c. predicted class (threshold = 0.5)

$\text{predicted-class} = \text{"pass" if prob} > 0.5$

else "fail"

print (predicted-class)

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
t
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_

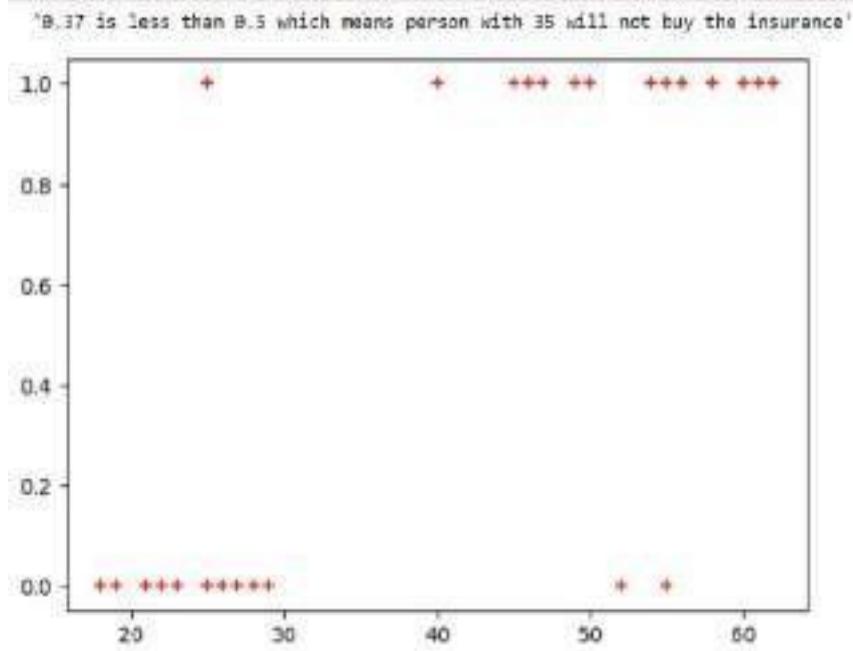
#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

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"""


```



```
# 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}")

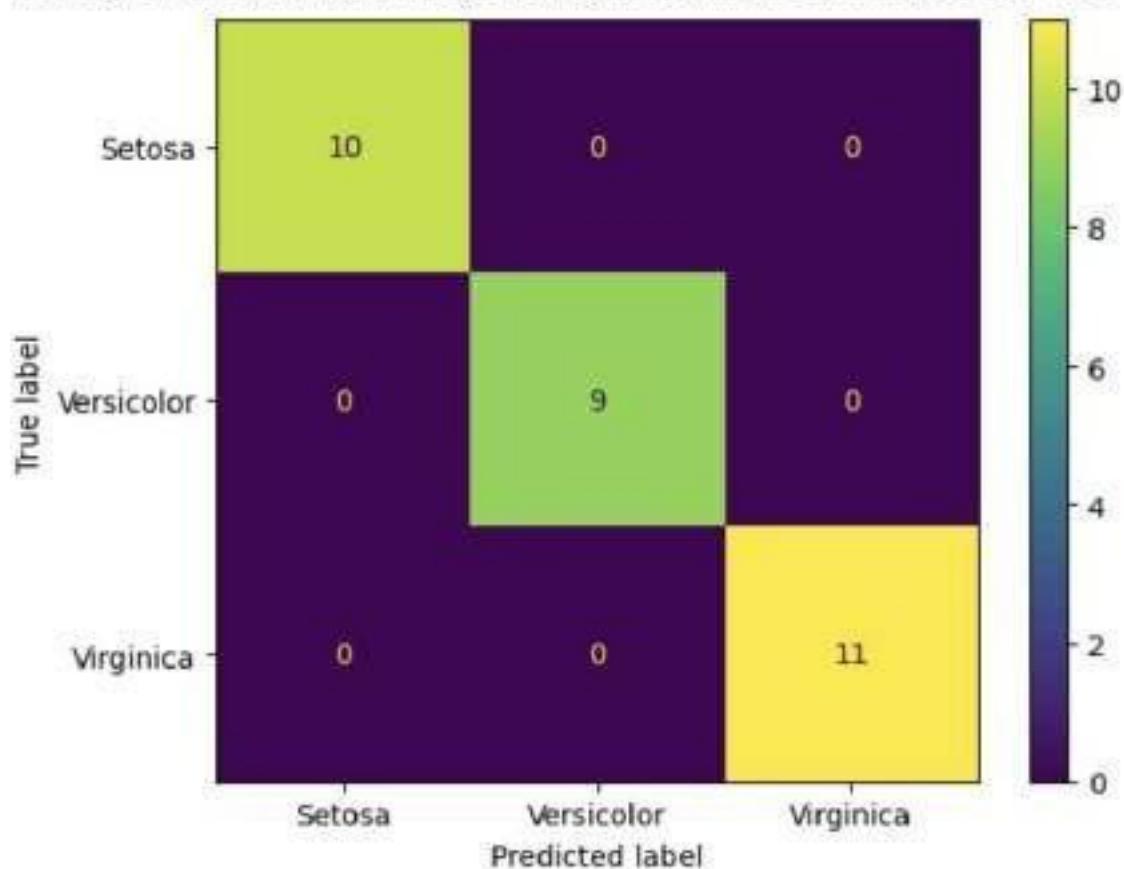
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

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.

OBSERVATION BOOK

KNN :	
	for iris dataset
1.	How to choose the K value? demonstrate Accuracy rate and error rate.
→	<ul style="list-style-type: none">• K (small) ex 1, 3, 5 → overfitting• K (large) K>15 → underfitting• optimal K is chosen by plotting Accuracy rate vs K Error rate vs K
*	from sklearn.metrics import accuracy_score acc = [accuracy_score(y_test, kNeighborsClassifier(n_neighbors=k).fit(X_train, y_train).predict(X_test)) for k in range(1, 21)]
2.	for diabetes data-set what is purpose of feature scaling?
→	<ul style="list-style-type: none">• Bring all feature to similar range• Prevent feature with large values from domination.• Speed up model convergence
BBM 21	
Scanned with CamScanner	

2. For Zoo dataset.

i) Data processing steps:

- - Dropped the animal-name column (not needed for classification)
- No missing values were found
- Standardized features using StandardScaler!!

ii) Handling missing or inconsistent values.

- No missing values or inconsistent values were found

iii) Confusion Matrix interpretation

- The model achieved 100% accuracy, correctly classifying all instances

iv) Misclassified Class Types

- No misclassification were observed.



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)

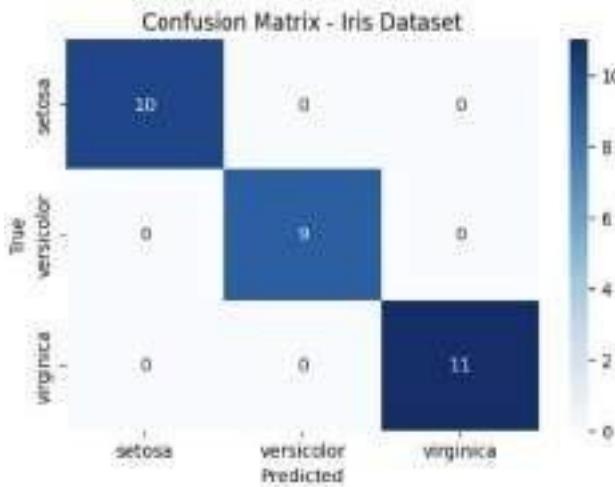
# Calculate accuracy score
acc_iris = accuracy_score(y_test_iris, y_pred_iris)
print("IRIS Dataset Accuracy Score:", acc_iris)

# 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)

IRIS Dataset Classification Report:
      precision    recall   f1-score   support
  setosa       1.00     1.00     1.00      50
  versicolor   0.98     0.98     0.98      50
  virginica    0.98     0.98     0.98      50

  accuracy         0.98      0.98      0.98      150
  macro avg       0.98     0.98     0.98      150
  weighted avg    0.98     0.98     0.98      150
```



```

----- 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)

```

```

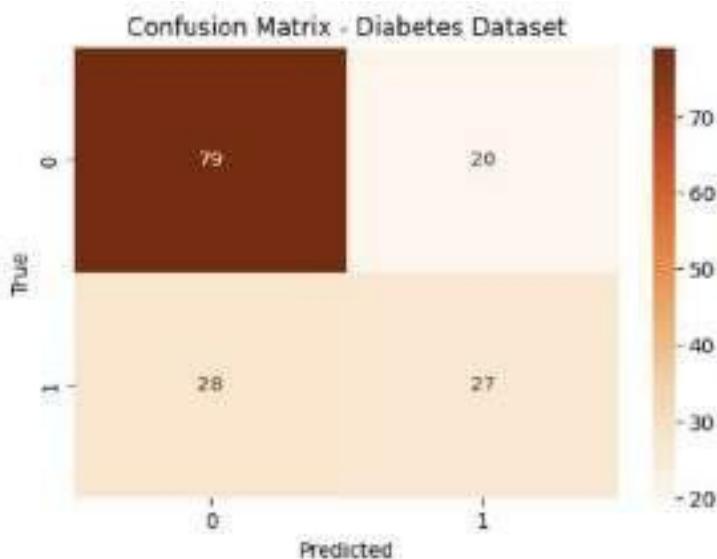
cr_diab = classification_report(y_test_diab, y_pred_diab)
print("\nDiabetes Dataset Classification Report:\n", cr_diab)

Diabetes Dataset Classification Report:
      precision    recall  f1-score   support

          0       0.74      0.88      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: 0.8689
K = 12 --> Accuracy: 0.8689
K = 13 --> Accuracy: 0.8852
K = 14 --> Accuracy: 0.8689
K = 15 --> Accuracy: 0.8616
K = 16 --> Accuracy: 0.8852
K = 17 --> Accuracy: 0.8852
K = 18 --> Accuracy: 0.9016
K = 19 --> Accuracy: 0.8892
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: 0.9544262296081948

Confusion Matrix:
[[28  1]
 [ 3 29]]

Classification Report:
precision    recall    f1-score   support
          0       0.98      0.97      0.97      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

1. Bootstrap Sampling:

- Generate N random samples from the training data with replacement.

2. Build Decision Trees:

- For each sample train a decision tree using a random subset of features for each split. (CART) Algorithm.
- The key difference in Random forest is that when choosing best split at each node it only considers a random subset of available features.

3. Tree Growth:

- Continue splitting the node of each decision tree until:
 - A specified tree depth is reached
 - Or, the number of instances in a node falls below a threshold.

4. Prediction.

- Classification, each tree votes for a class label; majority votes becomes the prediction.

- Regression: The average of the individual tree predictions is the final output.



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)

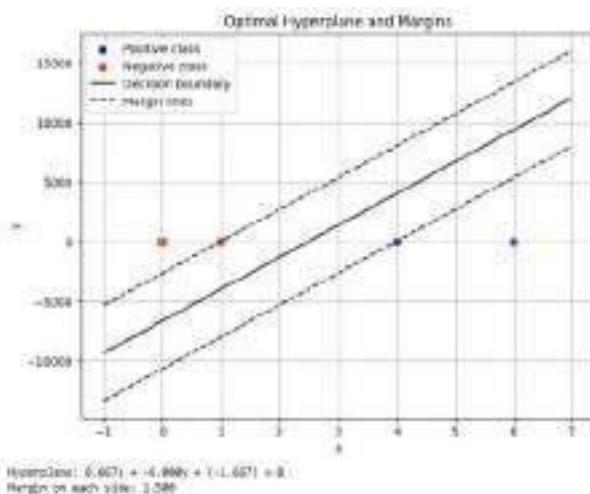
# Extract model parameters
w = clf.coef_[0]
b = clf.intercept_[0]

# 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)

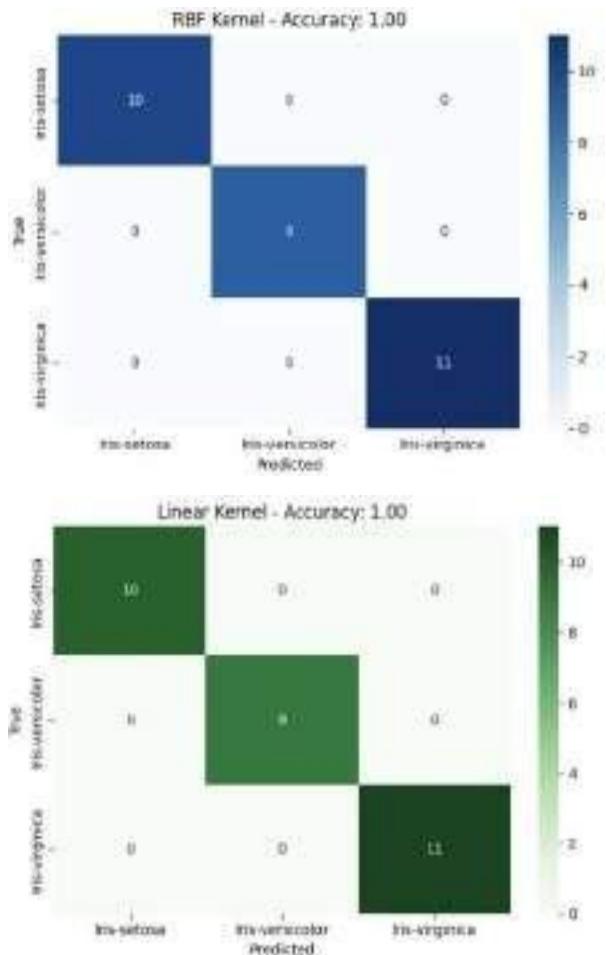
# Train models
svm_rbf = SVC(kernel='rbf')
svm_linear =
SVC(kernel='linear')

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()

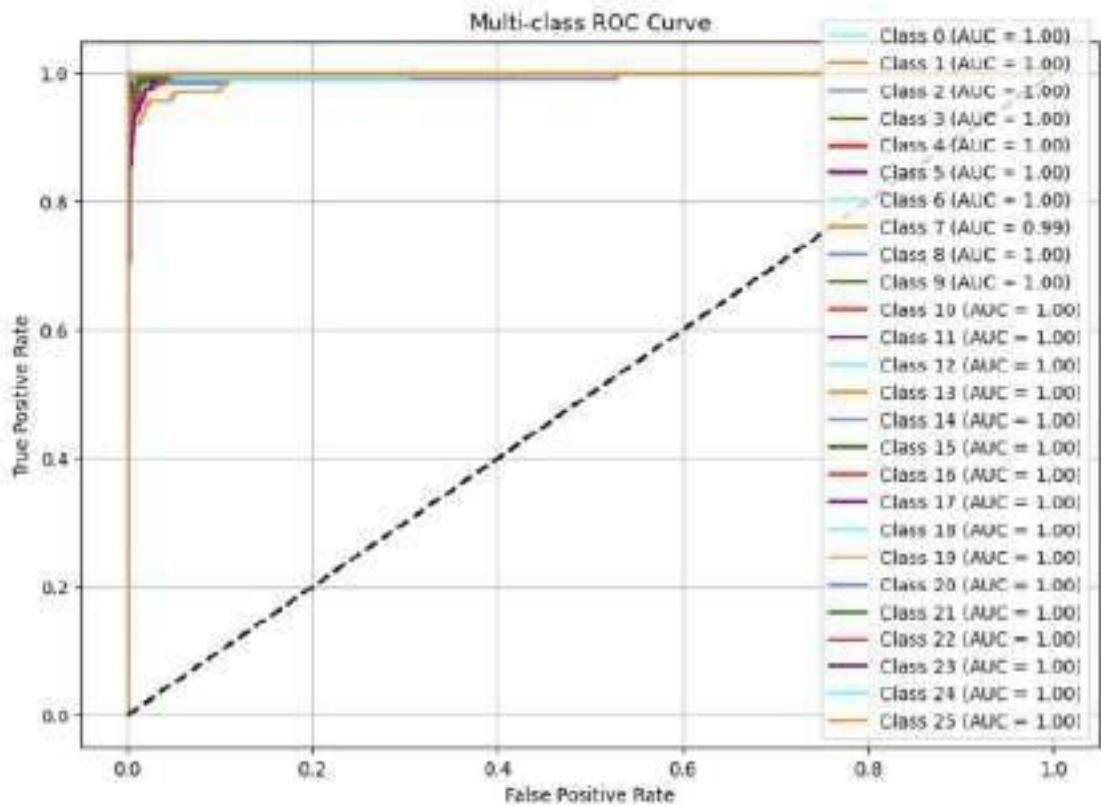
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# 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

URBAN EDGE 07.09.2

S.No.	CGPA	Interactiveness	Communication Skills	Prac.Knowledge
1	≥ 9	Yes	Good	Good
2	< 9	No	Moderate	Good
3	≥ 9	No	Moderate	Average
4	≥ 9	No	Moderate	Average
5	≥ 9	Yes	Moderate	Good

$[CGPA]$

```

graph TD
    CGPA[CGPA] --> GEq9[≥ 9]
    CGPA --> L9[< 9]
    GEq9 --> Interactiveness[Interactiveness]
    GEq9 --> JobOfferYes[Job Offer : Yes]
    L9 --> JobOfferNo[Job Offer : No]
    Interactiveness --> Good[Good]
    Interactiveness --> Moderate[Moderate]
    Good --> JobOfferYes1[Job Offer : Yes]
    Good --> JobOfferNo1[Job Offer : No]
    Moderate --> PracKnowledge[Practical Knowledge]
    PracKnowledge --> Average[Average]
    Average --> JobOfferYes2[Job Offer : Yes]
    Average --> JobOfferNo2[Job Offer : No]
  
```

Scanned with CamScanner

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("\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:
```

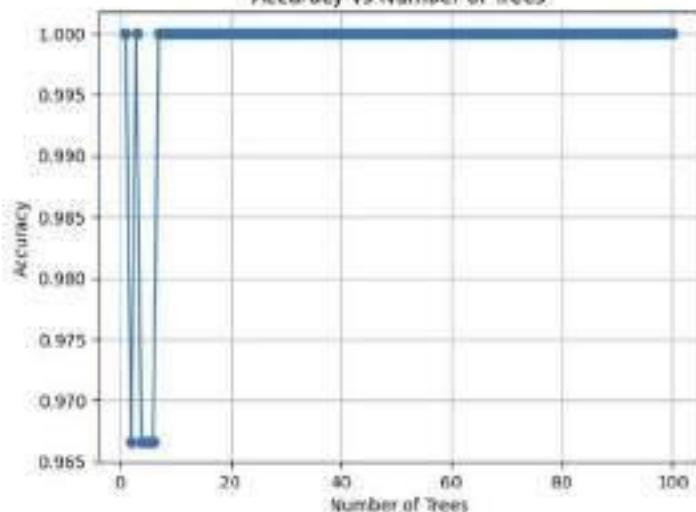
```
[[10  0  0]  
 [ 0  9  0]  
 [ 0  0 11]]
```

```
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

URBAN EDGE 07, 05, 25

Lab - P

Implement Boosting ensemble method on a given dataset.

- Deliver stump calculation steps for the attribute CGPA.

CGPA	Interactions	Practical Knowledge	Communication	Total Profit
≥ 9	Yes	Good	Good	Yes
< 9	No	Good	Moderate	Yes
≥ 9	No	Average	Moderate	No
< 9	No	Average	Good	No
≥ 9	Yes	Good	Moderate	Yes
≥ 9	Yes	Good	Moderate	Yes

- Initial Setup.

1. Data Points (N): 6

2. Initial weights (w_i): $1/N = 1/6 \approx 0.1667$

3. Possible splits : < 9 vs ≥ 9

- Error Calculation

- Split 1 (CGPA < 9):

- Predict 'Yes' for < 9
- 1 error (instance 4: < 9 but 'No')
- Error = 0.1667

- Split 2 (CGPA ≥ 9):

- Predict 'Yes' for ≥ 9
- 3 errors (instances 3, 5, 6)
- Error = 0.5

• Chosen stump:

If CGPA < 9 then Predict 'Yes' else Predict 'No'

Stump weight (α)

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

$$= \frac{1}{2} \ln \left(\frac{1 - 0.1667}{0.1667} \right) = 0.8047$$

• Weight update

- Correct cases: new weight = 0.0754

- Incorrect case (instance 4): new weight = 0.3684

- After normalization.

Correct = 0.10

Incorrect = 0.49

→ Observation.

Default accuracy = 0.8271

Default confusion matrix:

$$\begin{bmatrix} 1022 & 387 \\ 2138 & 14063 \end{bmatrix}$$

using 100 trees to in each batch.

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
    )
    cv_scores.append(scores.mean())

# Plot CV scores vs n_estimators
plt.figure(figsize=(10, 6))
plt.plot(tree_counts, cv_scores, marker='o')
plt.title("Cross-validation scores vs Number of estimators")
plt.xlabel("Number of estimators (n)")
plt.ylabel("Cross-validation accuracy")
plt.grid(True)
plt.show()
```

```
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()

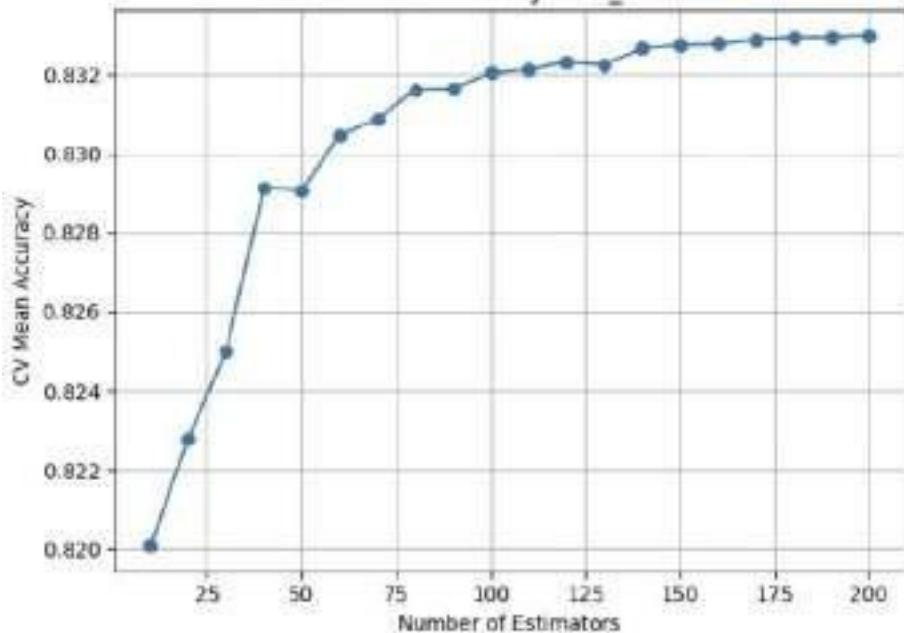
# Plot confusion matrix for best model
cm = confusion_matrix(y_test, y_best)
labels = best_model.named_steps['clf'].classes_
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
plt.figure()
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Best AdaBoost Model')
plt.tight_layout()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

```

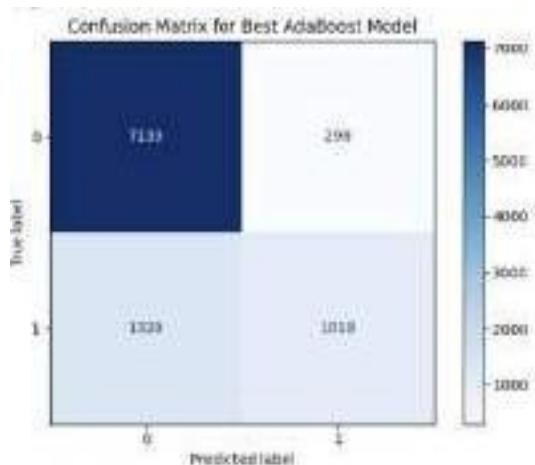
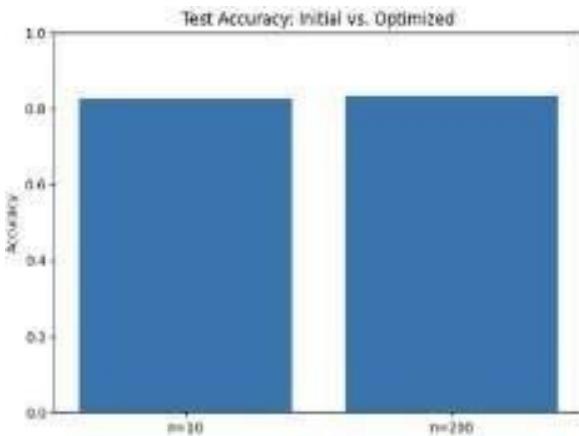
```
Dataset shape: (48842, 7)

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=0.8261
n_estimators=20: CV mean accuracy=0.8220
n_estimators=30: CV mean accuracy=0.8258
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=0.8328
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=0.8338
```

AdaBoost CV Accuracy vs. n_estimators



```
Best CV accuracy=0.8338 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			
Build K-Means algorithm to cluster a set of data stored in a .CSV file.			
0,4	(1,1)	(5,2)	
1,1	0,0	$\sqrt{52}$	C ₁
1.5,2	$\sqrt{1.25}$	$\sqrt{27.25}$	C ₁
3,4	$\sqrt{13}$	$\sqrt{13}$	C ₂
5,7	$\sqrt{52}$	0	C ₂
2.5,5	$\sqrt{27.25}$	$\sqrt{6.25}$	C ₂
4.5,5	$\sqrt{28.25}$	$\sqrt{4.25}$	C ₂
3.5,4.5	$\sqrt{18.5}$	$\sqrt{6.5}$	C ₂
$C_1 = \left(\frac{1+1.5}{2}, \frac{1+2}{2} \right) = (1.25, 1.5)$			
$C_2 = (3.5, 5.1)$			
(1.25, 1.5) (3.5, 5.1) clusters			
(1,1)	0.57	5.25	C ₁
(1.5,2)	0.56	4.04	C ₂
(3,4)	2.50	1.42	C ₂
(5,7)	6.10	2.16	C ₂
(2.5,5)	2.85	0.41	C ₂
(4.5,5)	3.32	0.61	C ₂
(3.5,4.5)	2.63	0.85	C ₂
Clusters = (1,1) (1.5,2) (3,4) (5,7) (3.5,5)			

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.
    """
    fit KMeans with k clusters, return labels and fitted model.
```

```

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')

```

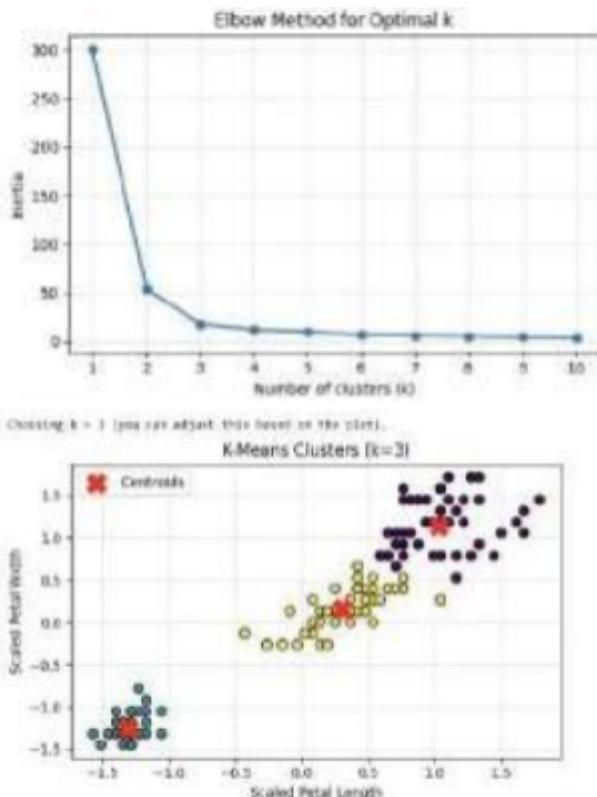
```

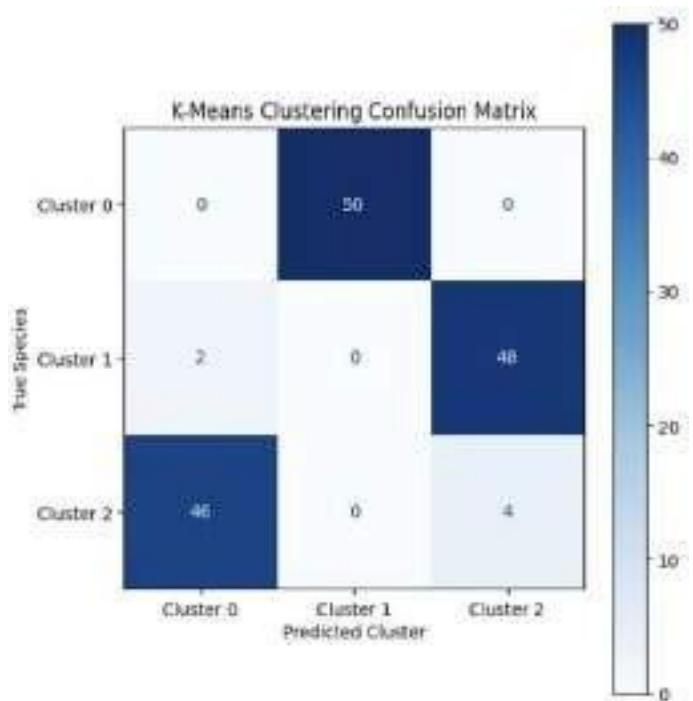
plt.ylabel('Scaled Petal Width')
plt.title(f'K-Means Clusters (k={optimal_k})')
plt.legend()
plt.grid(True, linestyle='--',
alpha=0.5) plt.tight_layout()
plt.show()

```

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

```
if __name__ == "__main__": main()
```





LABORATORY PROGRAM – 11

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

OBSERVATION BOOK

	LAB - 10	URBAN EDGE 07.05.25
	Given data , reduce dimension from 2 to 1	
	$x_1 \quad 4 \quad 8 \quad 13 \quad 7$ $x_2 \quad 11 \quad 9 \quad 5 \quad 14$	
	Eigen $\lambda_1 = 30.3849$ $\lambda_2 = 6.6151$	
	$z = [-4.30535, 3.73635, 5.64305, -5.12405]$	
	Model Accuracy	
	Before	After
	logistic regression : 0.8833	0.8667
	SVM : 0.8778	0.8556
	Random Forest : 0.8889	0.8722
		DATA 1-5-2025
	Scanned with CamScanner	

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
```