VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Navya Billalar (1BM22CS175)

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 **Navya Billalar (1BM22CS175),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty Incharge Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	 **Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

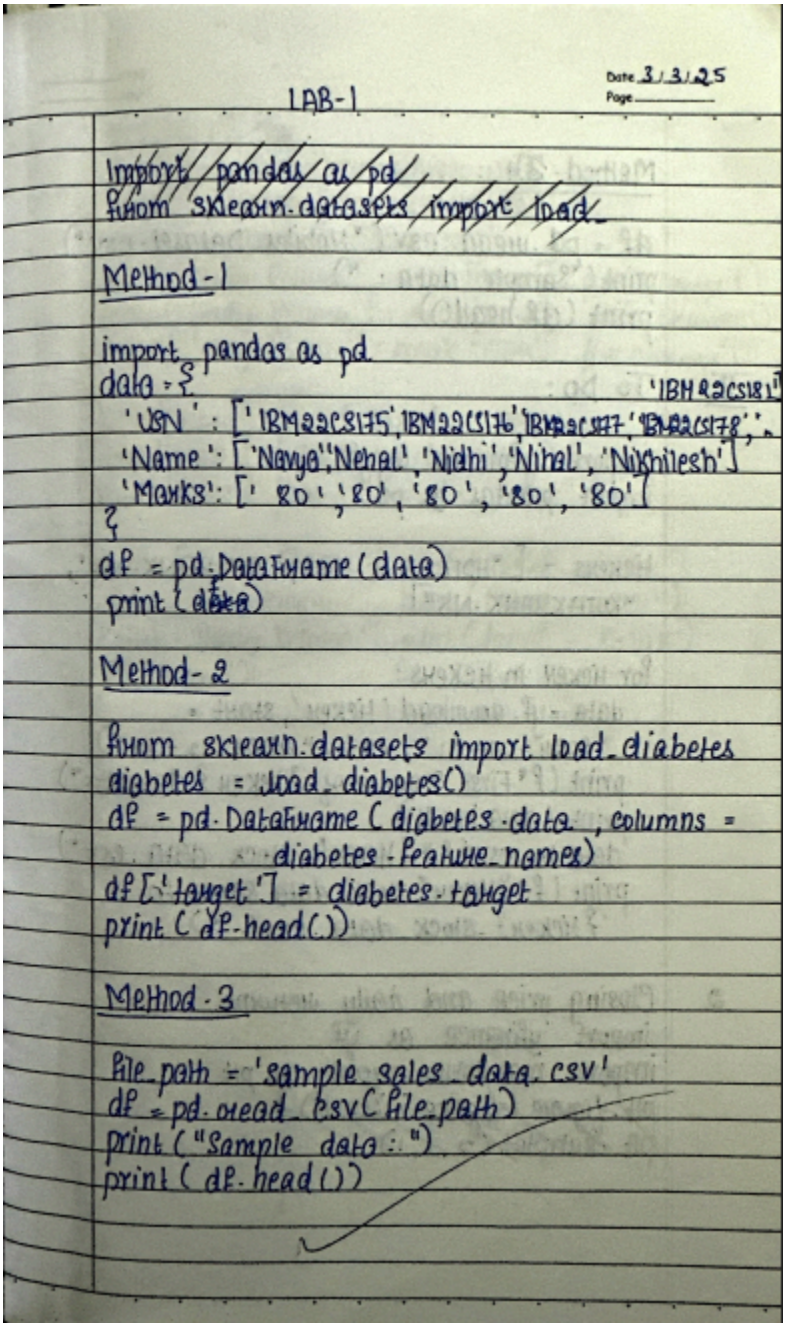
Index

Github Link:
https://github.com/navya-08/ML_LAB/tree/main

Program 1

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

Observation:



LAB-1

Date 3/3/25
Page

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

Method-1

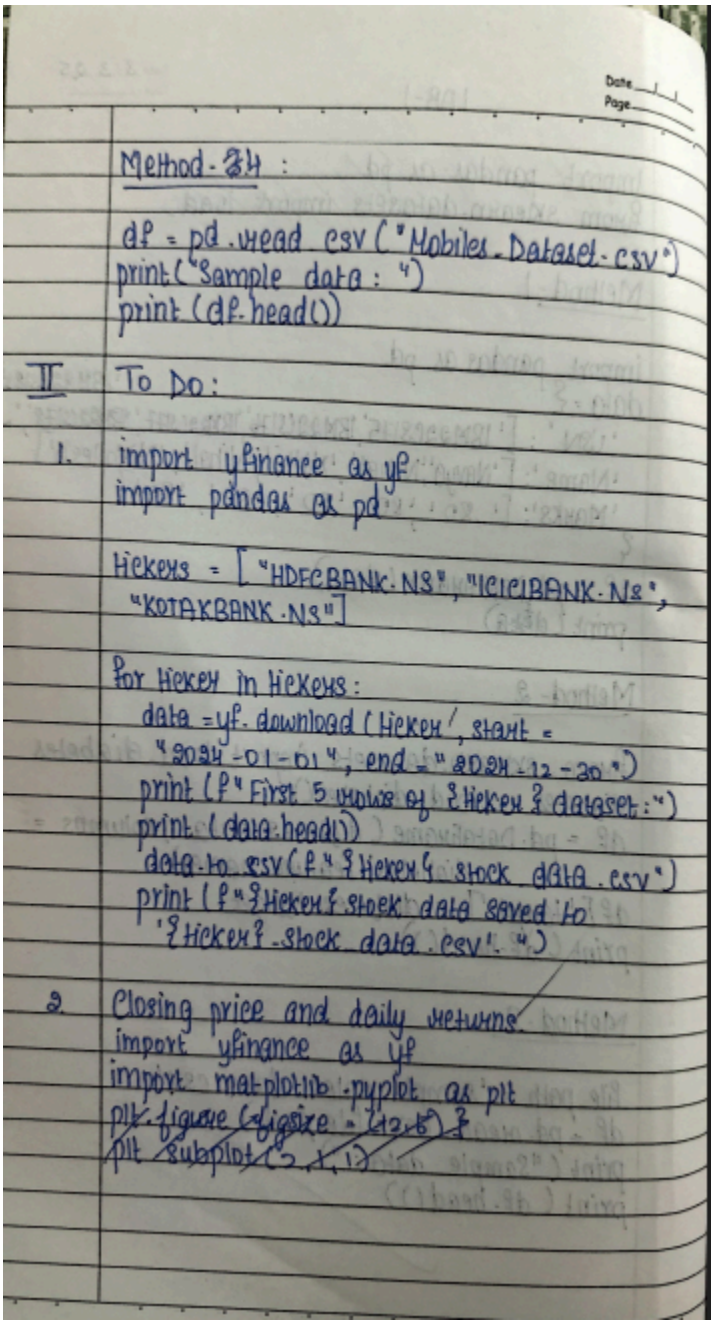
```
import pandas as pd
data = {
  'USN' : ['1BM22CS175','1BM22CS176','1BM22CS177','1BM22CS178',
  'Name': ['Navya','Nehal','Nidhi','Nihal','Nishilesh']
  'Marks': ['80','80','80','80','80']
}
df = pd.DataFrame(data)
print(data)
```

Method-2

```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns =
     diabetes.feature_names)
df['target'] = diabetes.target
print(df.head())
```

Method-3

```
file_path = 'sample_sales_data.csv'
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
```



Method - 2 :

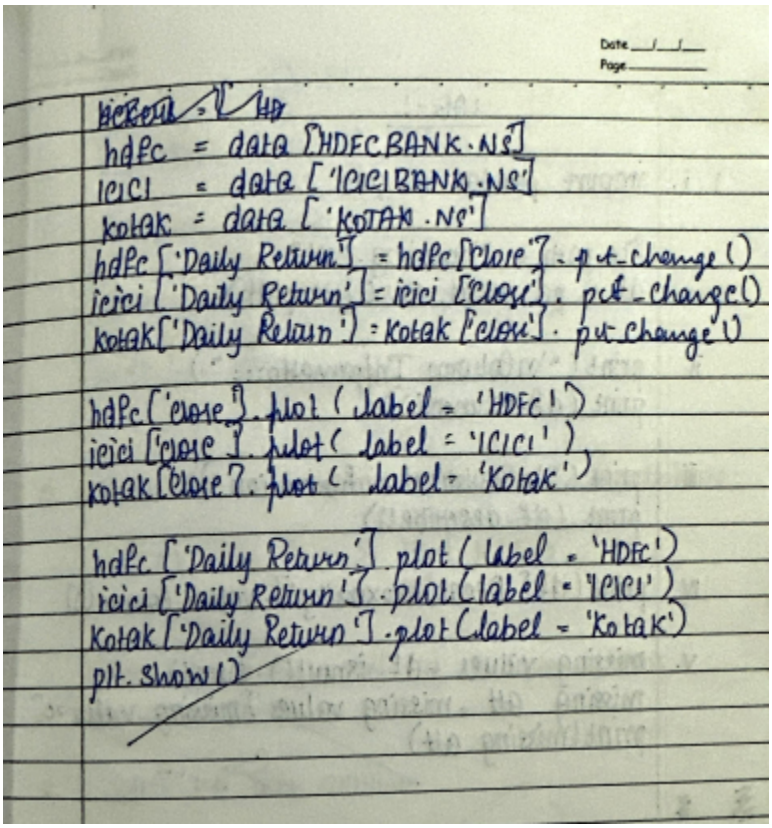
```
df = pd.read_csv("Mobiles_Dataset.csv")
print("Sample data :")
print(df.head())
```

II To Do:

1.
```
import yfinance as yf
import pandas as pd

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

for ticker in Tickers:
    data = yf.download(ticker, start =
    "2024-01-01", end = "2024-12-30")
    print(f"First 5 rows of {ticker} dataset:")
    print(data.head())
    data.to_csv(f"{ticker}_stock_data.csv")
    print(f"{ticker} stock data saved to
    '{ticker}_stock_data.csv'.")
```

2. Closing price and daily returns

```
import yfinance as yf
import matplotlib.pyplot as plt
plt.figure(figsize = (12,6))
plt.subplot(2,1,1)
```



Code:

```
import pandas as pd

# Reading data from a CSV file

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Evangline'],

'USN': ['1BM22CS025', '1BM22CS030', '1BM22CS035', '1BM22CS040', '1BM22CS045'],

'Marks': [25, 30, 35, 40, 45]

}

df = pd.DataFrame(data)

print("Sample data:")

print(df.head())
```

```
Sample data:
        Name          USN  Marks
0      Alice  1BM22CS025     25
1        Bob  1BM22CS030     30
2    Charlie  1BM22CS035     35
3      David  1BM22CS040     40
4  Evangline  1BM22CS045     45
```

```
from sklearn.datasets import load_diabetes

dia = load_diabetes()

df = pd.DataFrame(dia.data, columns=dia.feature_names)

df['target'] = dia.target

print("Sample data:")

print(df.head())
```

Output:

```
Sample data:
        age       sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6  target
0 -0.002592  0.019907 -0.017646   151.0
1 -0.039493 -0.068332 -0.092204    75.0
2 -0.002592  0.002861 -0.025930   141.0
3  0.034309  0.022688 -0.009362   206.0
4 -0.002592 -0.031988 -0.046641   135.0
```

```
import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

print("\nShape of the dataset:")
```

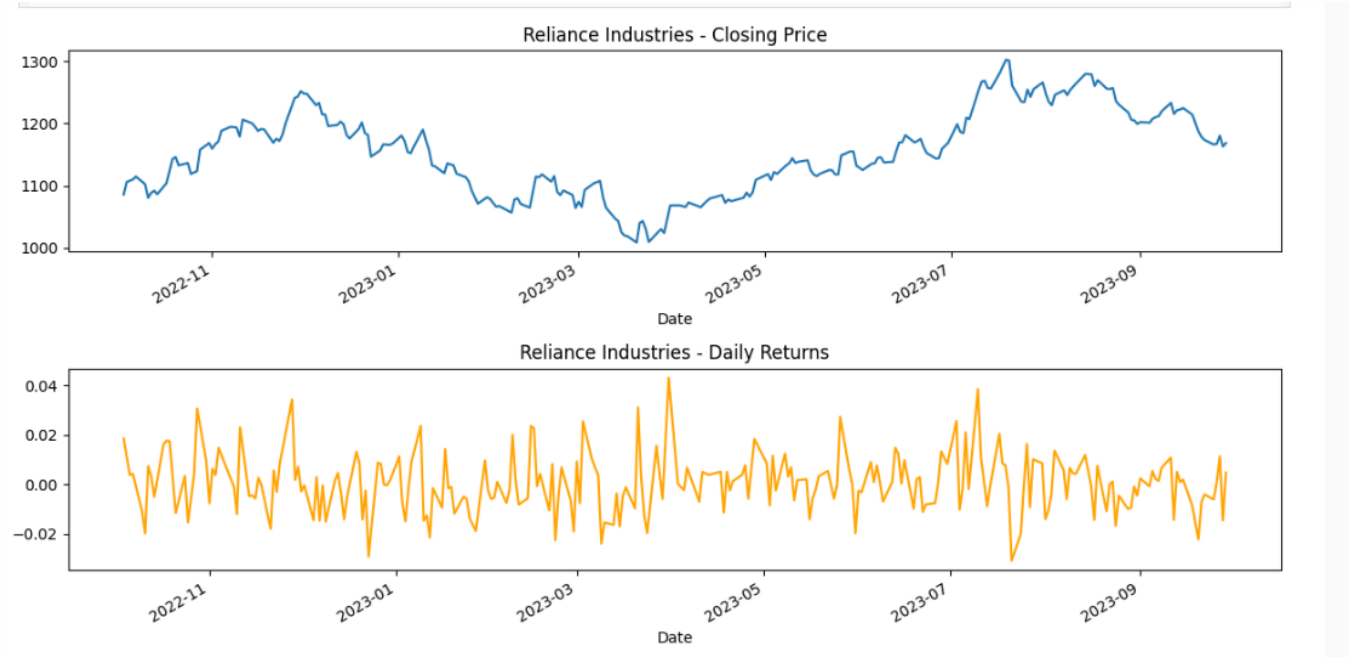
```
print(data.shape)

# Check column names

print("\nColumn names:")

print(data.columns)

# Summary statistics for a specific stock (e.g., Reliance)

reliance_data = data['RELIANCE.NS']

print("\nSummary statistics for Reliance Industries:")

print(reliance_data.describe())

# Calculate daily returns

# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe

reliance_data = data['RELIANCE.NS'].copy()

# Now, apply the calculation

reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
```

Output:

```
Shape of the dataset:
(247, 15)

Column names:
MultiIndex([('RELIANCE.NS',   'Open'),
            ('RELIANCE.NS',   'High'),
            ('RELIANCE.NS',    'Low'),
            ('RELIANCE.NS',  'Close'),
            ('RELIANCE.NS', 'Volume'),
            (     'TCS.NS',   'Open'),
            (     'TCS.NS',   'High'),
            (     'TCS.NS',    'Low'),
            (     'TCS.NS',  'Close'),
            (     'TCS.NS', 'Volume'),
            (    'INFY.NS',   'Open'),
            (    'INFY.NS',   'High'),
            (    'INFY.NS',    'Low'),
            (    'INFY.NS',  'Close'),
            (    'INFY.NS', 'Volume')],
           names=['Ticker', 'Price'])

Summary statistics for Reliance Industries:
Price         Open         High          Low        Close        Volume
count   247.000000   247.000000   247.000000   247.000000  2.470000e+02
mean   1155.033899  1163.758985  1144.612976  1154.002433  1.316652e+07
std      65.890843    66.876907    65.755901    66.726021  6.754099e+06
min    1015.178443  1017.470038   999.137216  1008.876526  3.370033e+06
25%    1106.532938  1111.081861  1092.347974  1104.997559  8.717141e+06
50%    1155.424265  1163.078198  1146.716157  1155.240967  1.158959e+07
75%    1202.667031  1209.102783  1193.235594  1201.447937  1.530302e+07
max    1297.045129  1308.961472  1281.920577  1302.476196  5.708188e+07
```

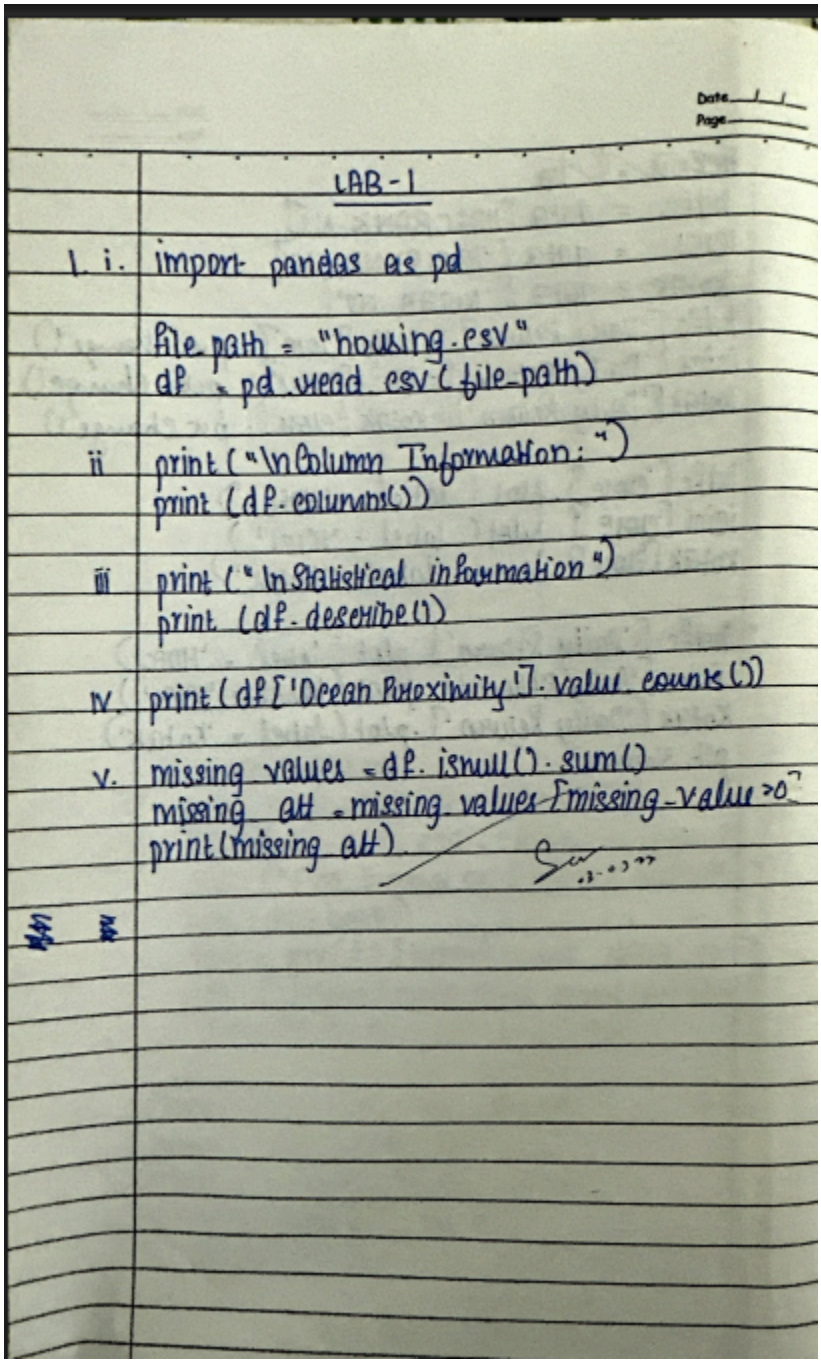
```
# Plot the closing price and daily returns

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

plt.subplot(2, 1, 1)

reliance_data['Close'].plot(title="Reliance Industries - Closing Price")

plt.subplot(2, 1, 2)

reliance_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')

plt.tight_layout()

plt.show()
```

Output:



Program 2

Observation:



LAB - 1

1. i. import pandas as pd

file_path = "housing.csv"
df = pd.read_csv(file_path)

ii print("\nColumn Information:")
print(df.columns())

iii print("\nStatistical information")
print(df.describe())

iv. print(df['Ocean Proximity'].value_counts())

v. missing_values = df.isnull().sum()
missing_all = missing_values[missing_value>0]
print(missing_all)

Code:

i. To load .csv file into the data frame

```
import pandas as pd

file_path = '/content/housing.csv'

df = pd.read_csv(file_path)

print("Sample data:")

print(df.head())

print("\n")
```

output:

```
Sample data:
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

ii. To display information of all columns

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

Output:

```
<bound method DataFrame.info of         longitude  latitude  housing_median_age  total_rooms  total_bedrooms
0         -122.23     37.88                41.0        880.0           129.0
1         -122.22     37.86                21.0       7099.0          1106.0
2         -122.24     37.85                52.0       1467.0           190.0
3         -122.25     37.85                52.0       1274.0           235.0
4         -122.25     37.85                52.0       1627.0           280.0
...           ...       ...                 ...          ...             ...
20635     -121.09     39.48                25.0       1665.0           374.0
20636     -121.21     39.49                18.0        697.0           150.0
20637     -121.22     39.43                17.0       2254.0           485.0
20638     -121.32     39.43                18.0       1860.0           409.0
20639     -121.24     39.37                16.0       2785.0           616.0

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
...               ...
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[20640 rows x 10 columns]>
```

iii. To display the count of unique labels for “Ocean Proximity” column

print(df['ocean_proximity'].value_counts())

Output:

```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

iv. To display which attributes (columns) in a dataset have missing values count greater than zero

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

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

V.To display statistical information of all numerical

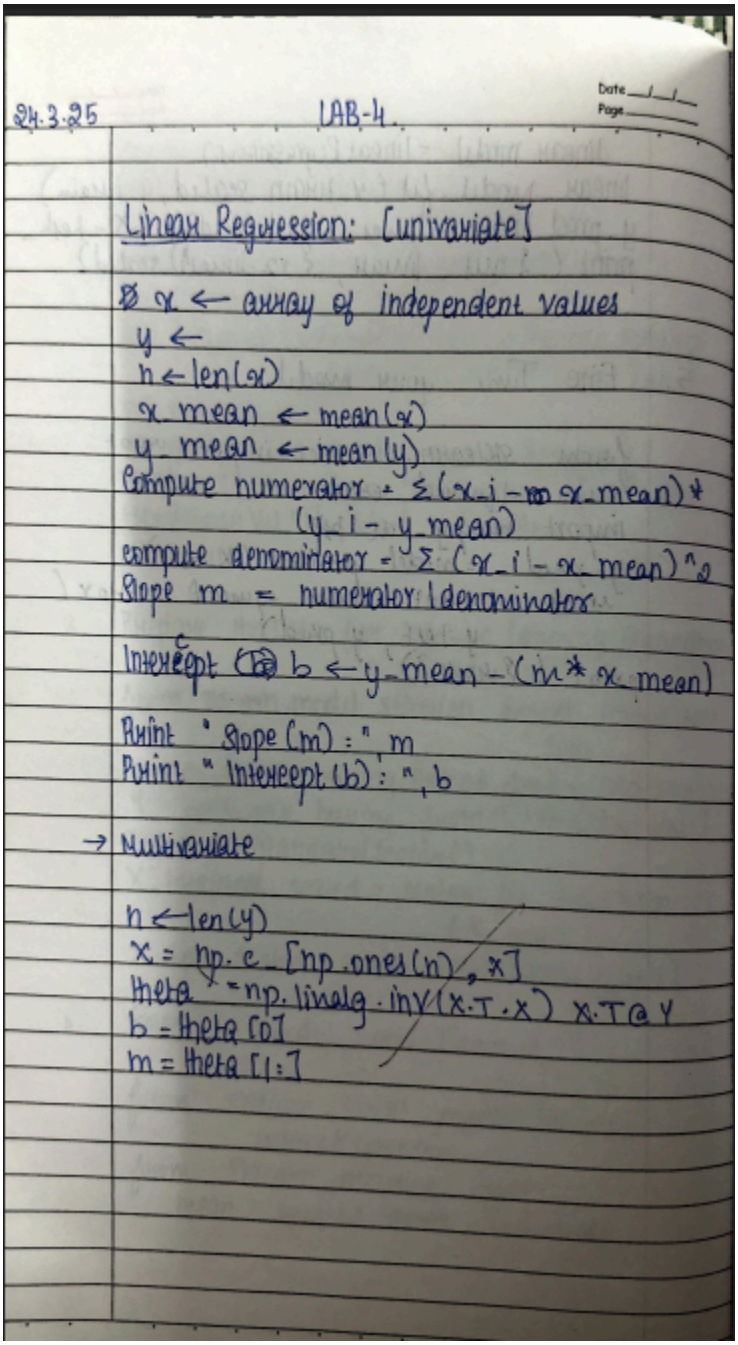
print(df.describe())

Output:

```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000
```

Program 3

Observation:



24.3.25 LAB-4

Linear Regression: [univariate]

x ← array of independent values
y ←
n ← len(x)
x_mean ← mean(x)
y_mean ← mean(y)
Compute numerator = Σ(x_i - x_mean) * (y_i - y_mean)
compute denominator = Σ (x_i - x_mean) ^2
Slope m = numerator / denominator

Intercept b ← y_mean - (m * x_mean)

Print "Slope (m) : ", m
Print "Intercept (b) : ", b

→ Multivariate

n ← len(y)
X = np.c_[np.ones(n), x]
theta = np.linalg.inv(X.T.X) X.T@Y
b = theta[0]
m = theta[1:]

Code:

Linear Regression:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()

X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)

X = X[['MedInc', 'AveRooms']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print("Linear Regression Results:")
print(results.head())
```

Output :

```
Linear Regression Results:
        Actual  Predicted
20046  0.47700   1.162302
3024   0.45800   1.499135
15663  5.00001   1.955731
20484  2.18600   2.852755
9814   2.78000   2.001677
```

2. Multiple regression

Code:

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing
california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
X = X[['MedInc', 'AveRooms']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
# Print the actual vs predicted values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

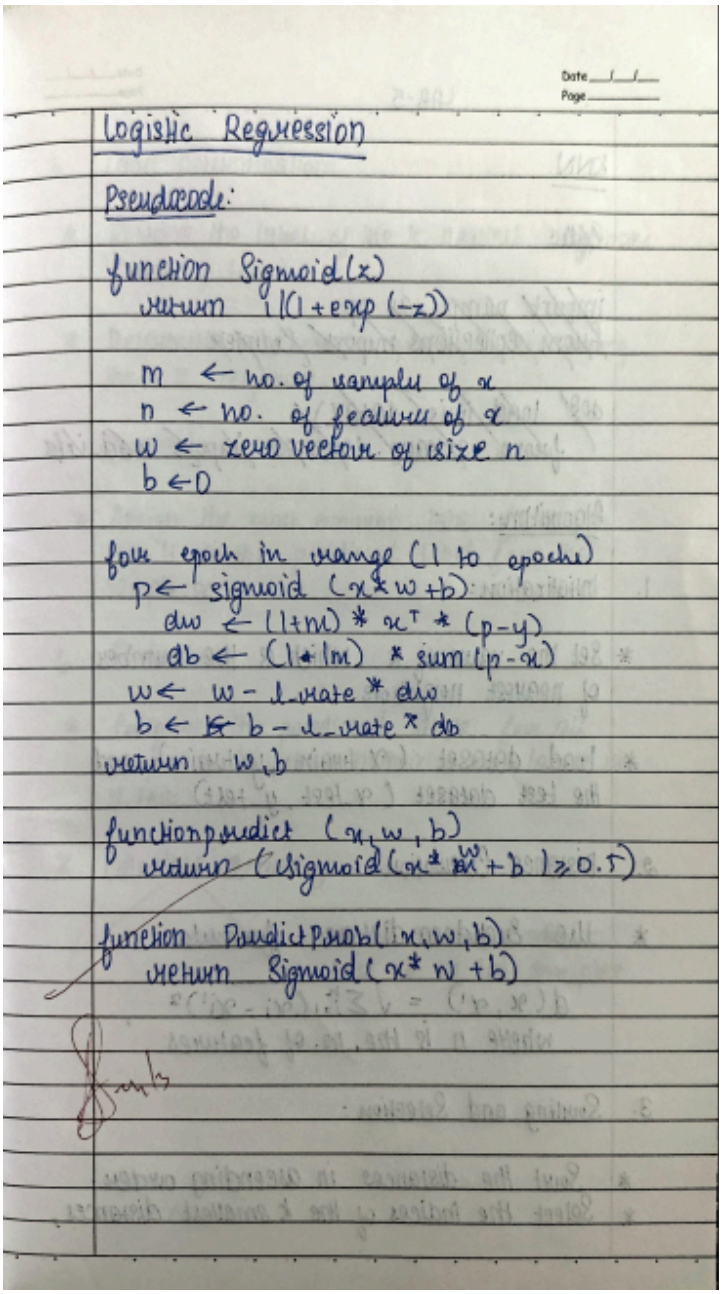
Output:

```
        Actual  Predicted
20046  0.47700   1.162302
3024   0.45800   1.499135
15663  5.00001   1.955731
20484  2.18600   2.852755
9814   2.78000   2.001677
```

Program 4

Logistic regression

Observation:



Logistic Regression

Pseudocode:

```
function Sigmoid(z)
    return 1/(1+exp(-z))

    m ← no. of samples of x
    n ← no. of features of x
    w ← zero vector of size n
    b ← 0

for epoch in range (1 to epochs)
    p ← sigmoid (x*w+b)
      dw ← (1/m) * x^T * (p-y)
      db ← (1/m) * sum(p-x)
    w ← w - l_rate * dw
    b ← b - l_rate * db
return w,b

functionpredict (x, w, b)
    return (sigmoid(x* w+b) >= 0.5)

function PredictProb(x,w,b)
    return Sigmoid(x* w +b)
```

Code:

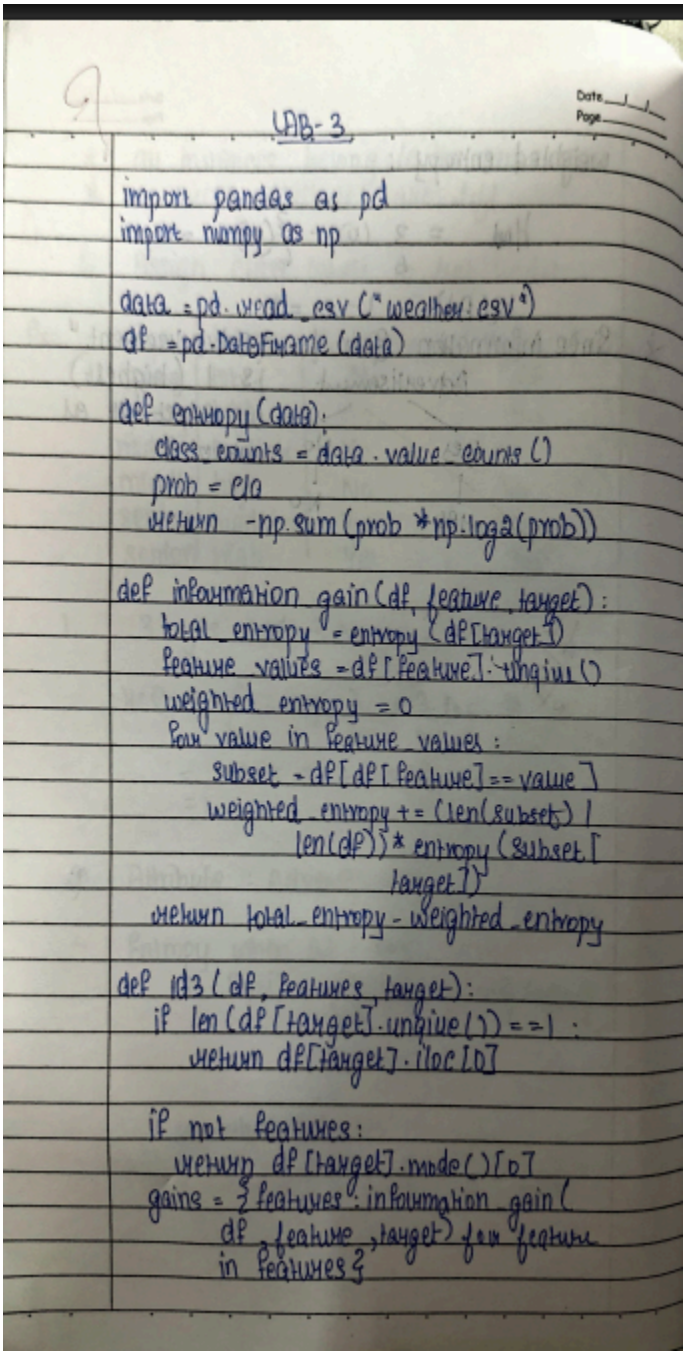
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)
X = X[y.isin([0, 1])]  # Select only classes 0 and 1
y = y[y.isin([0, 1])]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print("Logistic Regression Results:")
print(results.head())
```

output:

```
Logistic Regression Results:
    Actual  Predicted
83       1          1
53       1          1
70       1          1
45       0          0
44       0          0
```

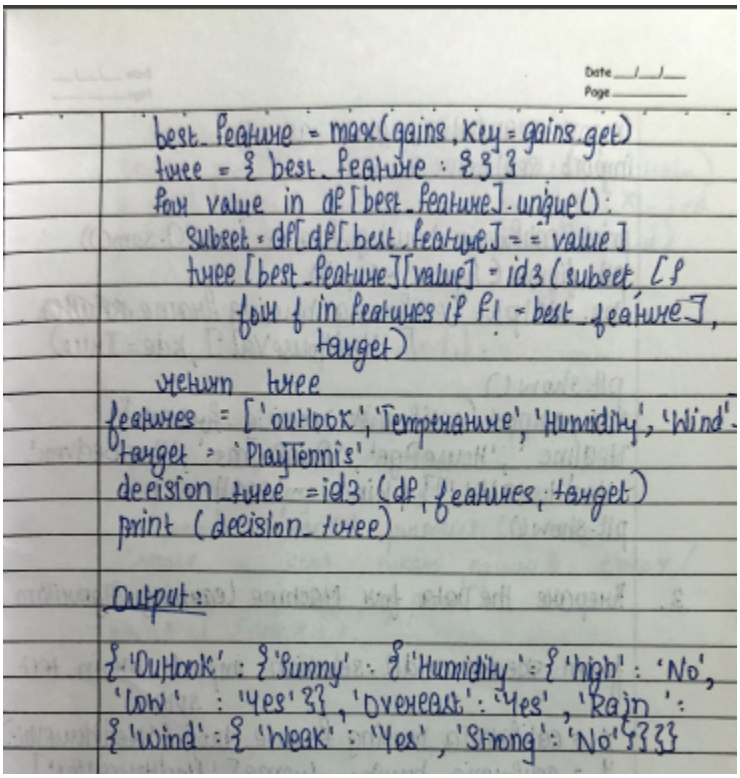
Program 5

Observation:



LAB-3

```
import pandas as pd
import numpy as np

data = pd.read_csv("weather.csv")
df = pd.DataFrame(data)

def entropy(data):
    class_counts = data.value_counts()
    prob = c/a
    return -np.sum(prob * np.log2(prob))

def information_gain(df, feature, target):
    total_entropy = entropy(df[target])
    feature_values = df[feature].unique()
    weighted_entropy = 0
    for value in feature_values:
        subset = df[df[feature] == value]
        weighted_entropy += (len(subset) /
            len(df)) * entropy(subset[
                target])
    return total_entropy - weighted_entropy

def id3(df, features, target):
    if len(df[target].unique()) == 1:
        return df[target].iloc[0]

    if not features:
        return df[target].mode()[0]
    gains = {feature: information_gain(
        df, feature, target) for feature
        in features}
```



Code:

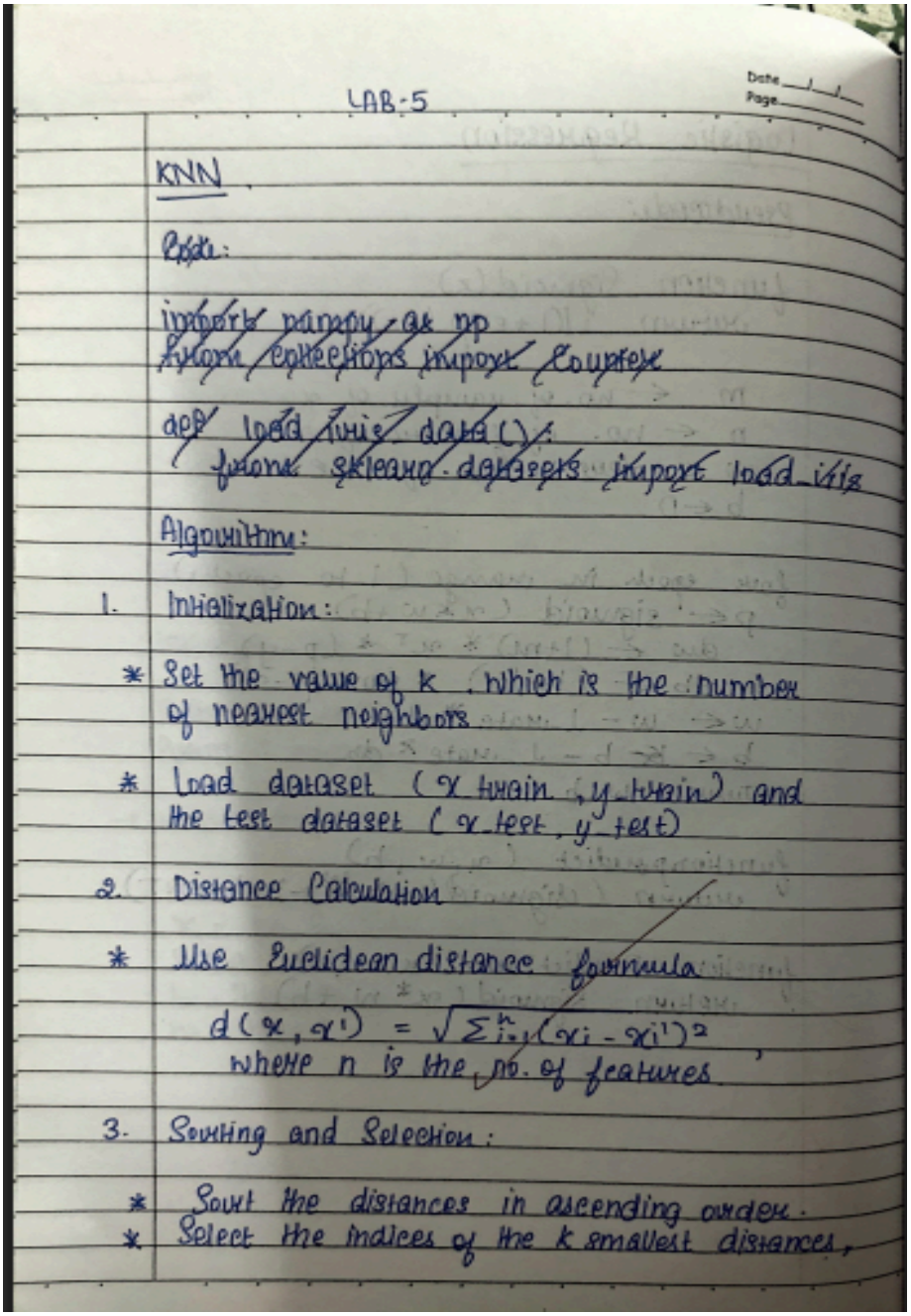
```
import numpy as np
import pandas as pd
from graphviz import Digraph


def entropy(dataset):
    class_counts = dataset.iloc[:, -1].value_counts()
    prob = class_counts / len(dataset)
    return -np.sum(prob * np.log2(prob))


def information_gain(dataset, feature):
    total_entropy = entropy(dataset)
    feature_values = dataset[feature].value_counts()
    weighted_entropy = 0
    for value, count in feature_values.items():
        subset = dataset[dataset[feature] == value]
        weighted_entropy += (count / len(dataset)) * entropy(subset)
    return total_entropy - weighted_entropy


def best_feature(dataset):
    features = dataset.columns[:-1]
    best_info_gain = -1
    best_feature = None
    for feature in features:
        info_gain = information_gain(dataset, feature)
```

```
        if info_gain > best_info_gain:
            best_info_gain = info_gain
            best_feature = feature
    return best_feature


def id3(dataset, max_depth=None, depth=0):
    if len(dataset.iloc[:, -1].unique()) == 1:
        return dataset.iloc[0, -1]
    if len(dataset.columns) == 1:
        return dataset.iloc[:, -1].mode()[0]
    if max_depth is not None and depth >= max_depth:
        return dataset.iloc[:, -1].mode()[0]
    best = best_feature(dataset)
    tree = {best: {}}
    for value in dataset[best].unique():
        subset = dataset[dataset[best] == value]
        tree[best][value] = id3(subset.drop(columns=[best]), max_depth=max_depth, depth=depth+1)
    return tree


def create_tree_diagram(tree, dot=None, parent_name="Root", parent_value=""):
    if dot is None:
        dot = Digraph(format="png", engine="dot")

    # Recursively add nodes to the graph
    if isinstance(tree, dict):
        for feature, branches in tree.items():
            feature_name = f"{parent_name}_{feature}"
            dot.node(feature_name, feature)
            dot.edge(parent_name, feature_name, label=parent_value)

            for value, subtree in branches.items():
                value_name = f"{feature_name}_{value}"
                dot.node(value_name, f"{feature}: {value}")
                dot.edge(feature_name, value_name, label=str(value))

                # Recurse for each subtree
                create_tree_diagram(subtree, dot, value_name, str(value))
    else:
        dot.node(parent_name + "_class", f"Class: {tree}")
        dot.edge(parent_name, parent_name + "_class", label="Leaf")

    return dot


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', 'Cool','Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal',
'High', 'Normal', 'High'],
```

```
    'Windy': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',
'Weak', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}
df = pd.DataFrame(data)

tree = id3(df, max_depth=3)


dot = create_tree_diagram(tree)

dot.render("decision_tree", view=True)
```

Program 6

Observation:



LAB-5

Date / /
Page

KNN

Code:

~~import numpy as np~~
~~from collections import Counter~~

~~def load_iris_data():~~
~~from sklearn.datasets import load_iris~~

Algorithm:

1. Initialization:

* Set the value of k, which is the number of nearest neighbors
* Load dataset (x_train, y_train) and the test dataset (x_test, y_test)

2. Distance Calculation

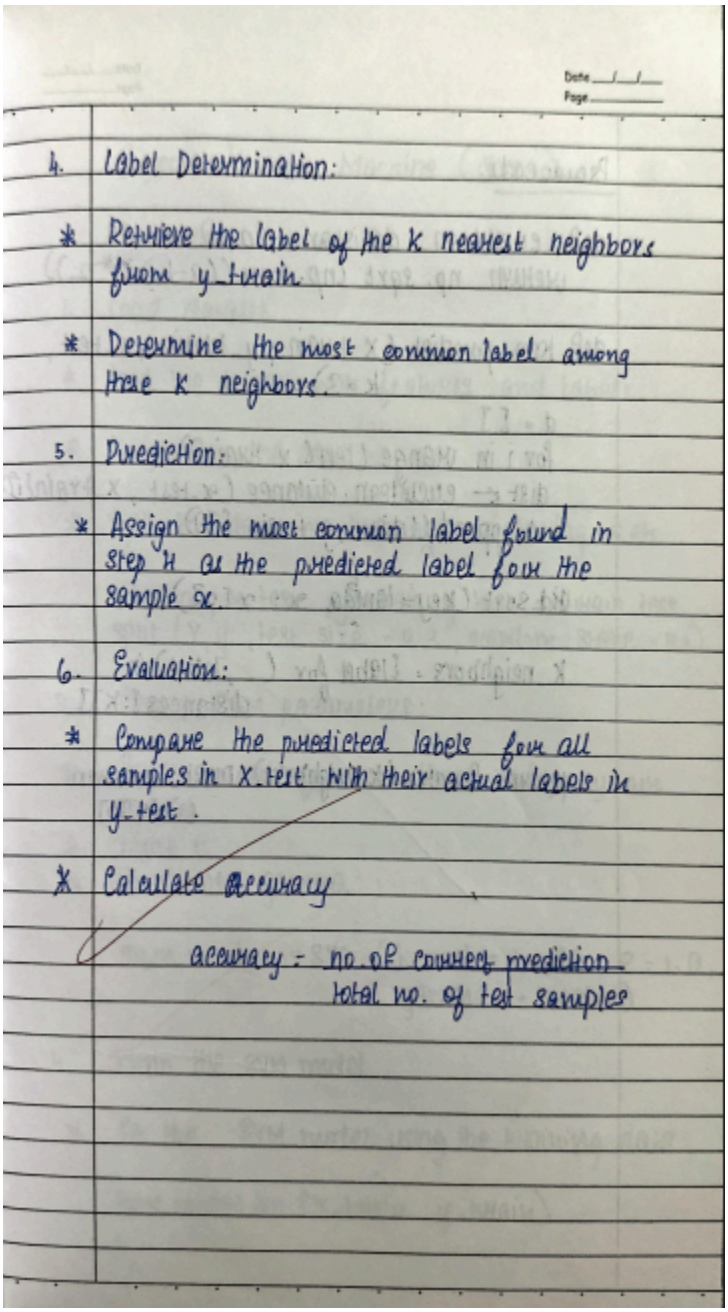
* Use Euclidean distance formula

$$d(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x_i')^2}$$

where n is the no. of features.

3. Sorting and Selection:

* Sort the distances in ascending order.
* Select the indices of the k smallest distances,



4. Label Determination:

* Retrieve the label of the k nearest neighbors from y_train.

* Determine the most common label among these k neighbors.

5. Prediction:

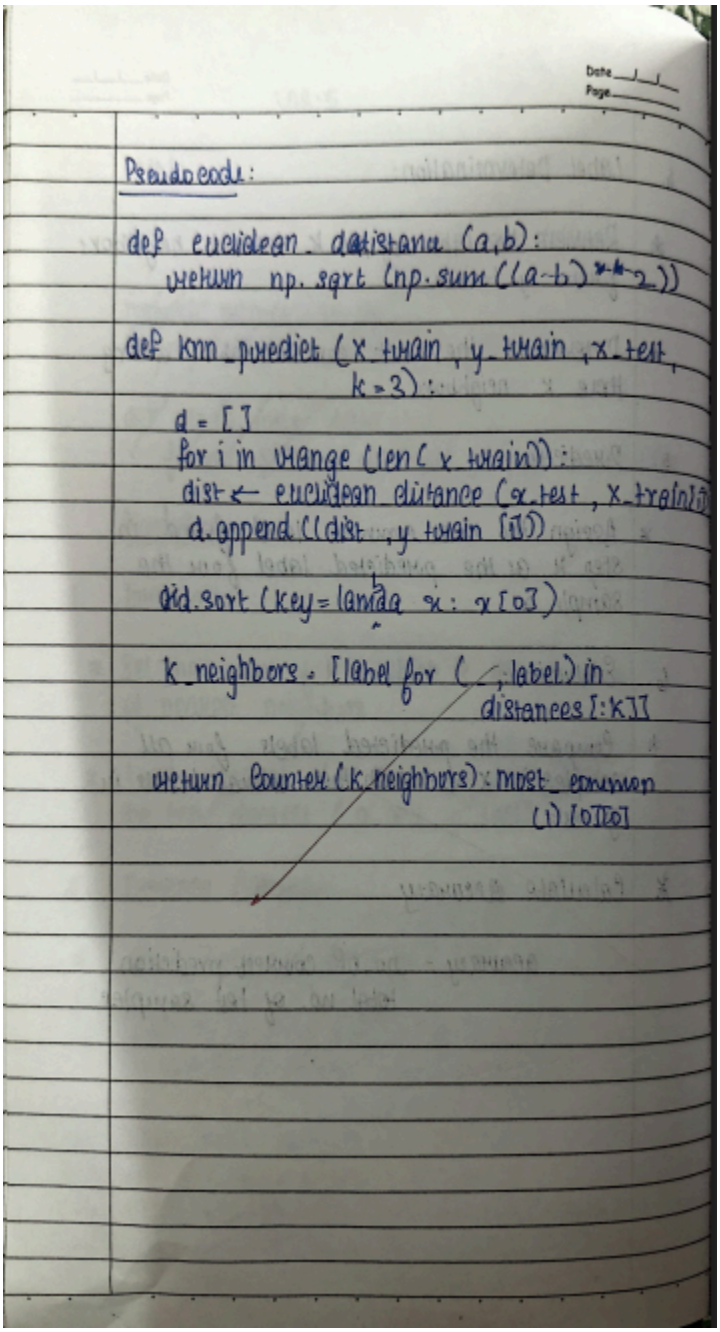
* Assign the most common label found in step 4 as the predicted label for the sample x.

6. Evaluation:

* Compare the predicted labels for all samples in X_test with their actual labels in y_test.

* Calculate accuracy

accuracy = no. of correct prediction / total no. of test samples



Code:

```
# Importing necessary libraries
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
```

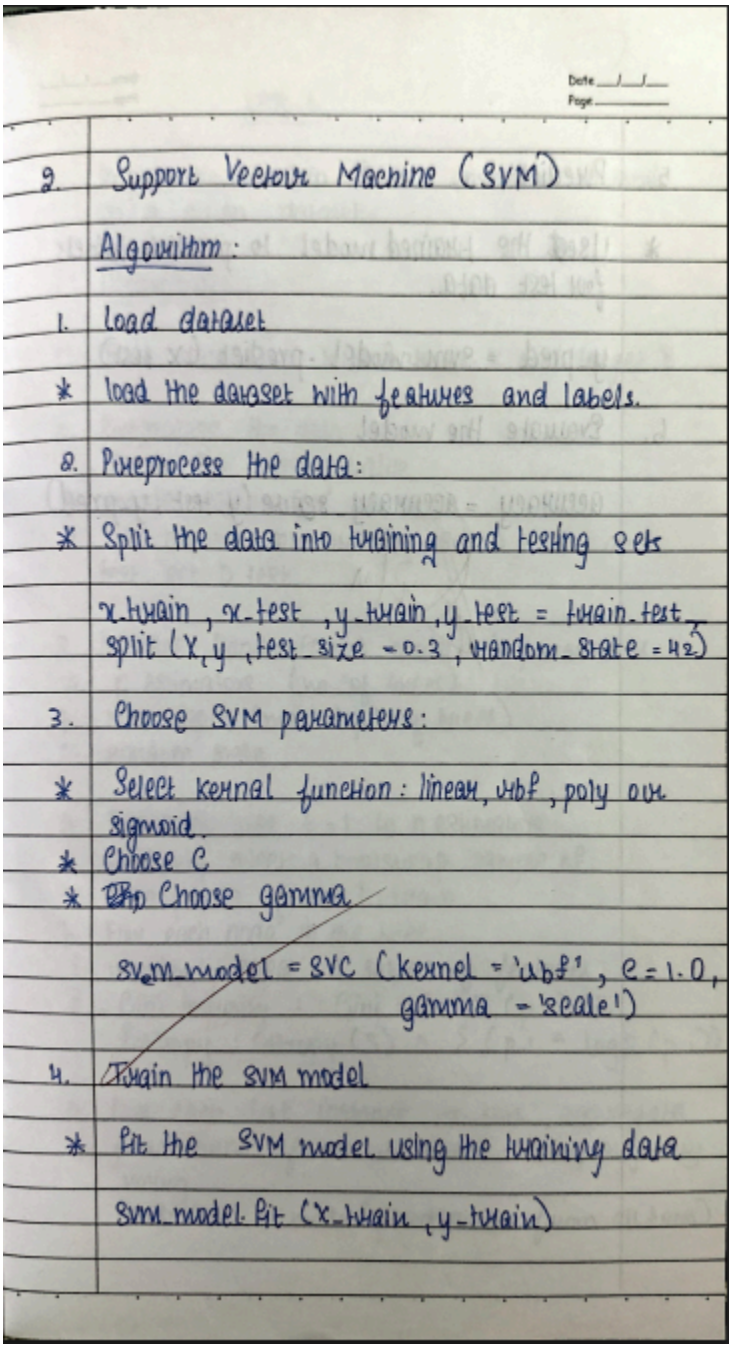
```
y = iris.target  # Target labels

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

# Create the KNN classifier (k=3 for this example)
knn = KNeighborsClassifier(n_neighbors=3)

# Fit the model
knn.fit(X_train, y_train)

# Predict using the test set
y_pred_knn = knn.predict(X_test)

# Evaluate the model's accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Model Accuracy: {accuracy_knn * 100:.2f}%")
```

Output:

```
KNN Model Accuracy: 100.00%
```

Program 7

Observation:



2. Support Vector Machine (SVM)

Algorithm:

1. load dataset

* load the dataset with features and labels.

2. Preprocess the data:

* Split the data into training and testing sets

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

3. Choose SVM parameters:

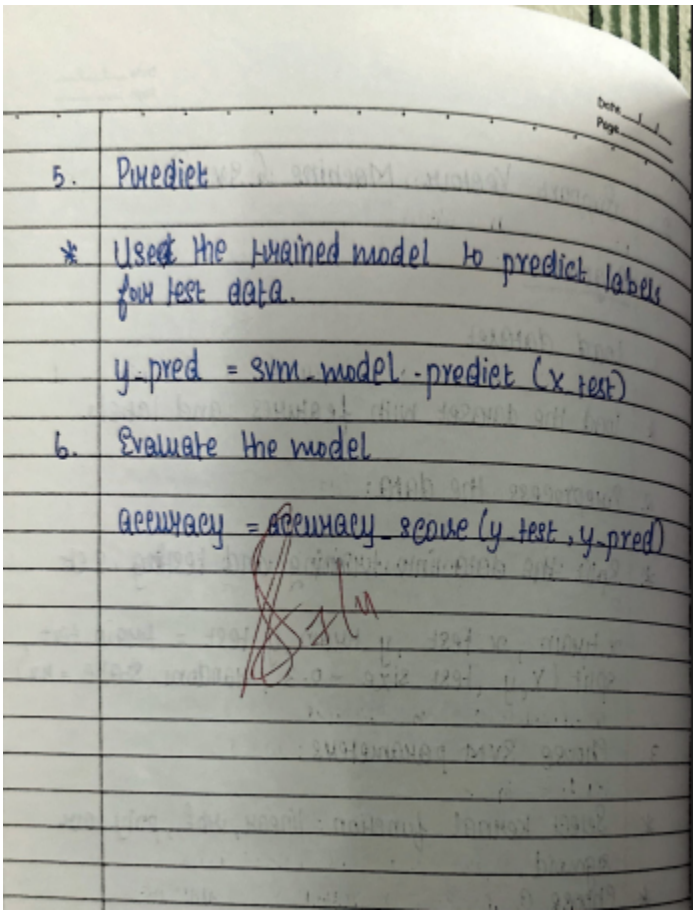
* Select kernel function: linear, rbf, poly or sigmoid.
* Choose C
* Choose gamma

```
svm_model = SVC(kernel = 'rbf', C = 1.0, gamma = 'scale')
```

4. Train the SVM model

* Fit the SVM model using the training data

```
svm_model.fit(x_train, y_train)
```



Code:

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Target labels

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

# Create the SVM classifier (using a linear kernel for this example)
svm = SVC(kernel='linear')

# Fit the model
svm.fit(X_train, y_train)
```

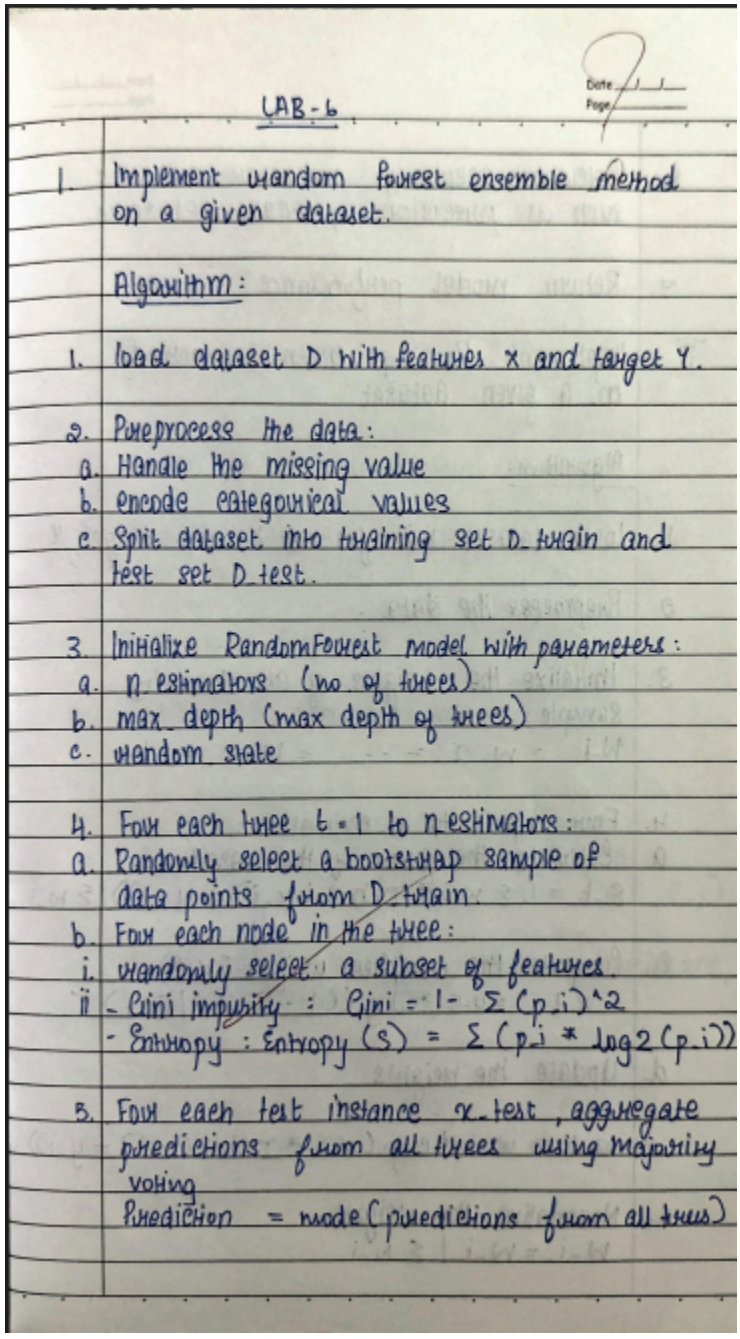
```
# Predict using the test set
y_pred_svm = svm.predict(X_test)

# Evaluate the model's accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Model Accuracy: {accuracy_svm * 100:.2f}%")
```

Output:

```
SVM Model Accuracy: 100.00%
```

Program 8

Observation:

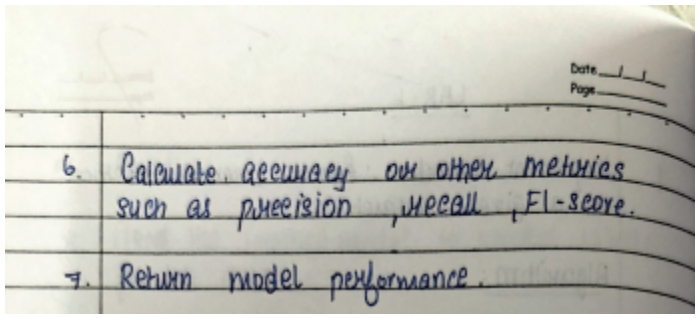


LAB - 6

1. Implement random forest ensemble method on a given dataset.

Algorithm:

1. load dataset D with features X and target Y.
2. Preprocess the data:
   a. Handle the missing value
   b. encode categorical values
   c. Split dataset into training set D_train and test set D_test.
3. Initialize RandomForest model with parameters:
   a. n_estimators (no of trees)
   b. max_depth (max depth of trees)
   c. random state
4. For each tree t = 1 to n_estimators:
   a. Randomly select a bootstrap sample of data points from D_train.
   b. For each node in the tree:
      i. randomly select a subset of features.
      ii. - Gini impurity : Gini = 1 - Σ(p_i)^2
      - Entropy : Entropy(S) = Σ(p_i * log2(p_i))
5. For each test instance x_test, aggregate predictions from all trees using majority voting
   Prediction = mode(predictions from all trees)



6. Calculate accuracy or other metrics such as precision, recall, F1-score.
7. Return model performance.

Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

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

# Initialize Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

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

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Random Forest Classifier Accuracy: {accuracy:.4f}')
```

output:

```
Random Forest Classifier Accuracy: 1.0000
```

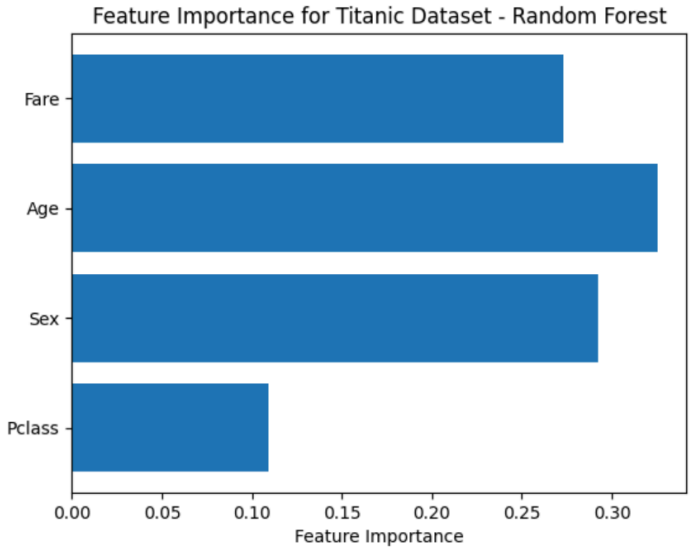
Random Forest on a Kaggle Dataset (Visualization)

Code:

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

# Load Titanic dataset (Replace URL with the Kaggle Titanic dataset URL or local file)
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)

# Preprocessing: Select features and target variable
data = data.dropna(subset=['Survived', 'Pclass', 'Sex', 'Age', 'Fare'])  # Drop missing values
X = data[['Pclass', 'Sex', 'Age', 'Fare']]
X['Sex'] = X['Sex'].map({'male': 0, 'female': 1})  # Encode 'Sex' column as binary
y = data['Survived']
```

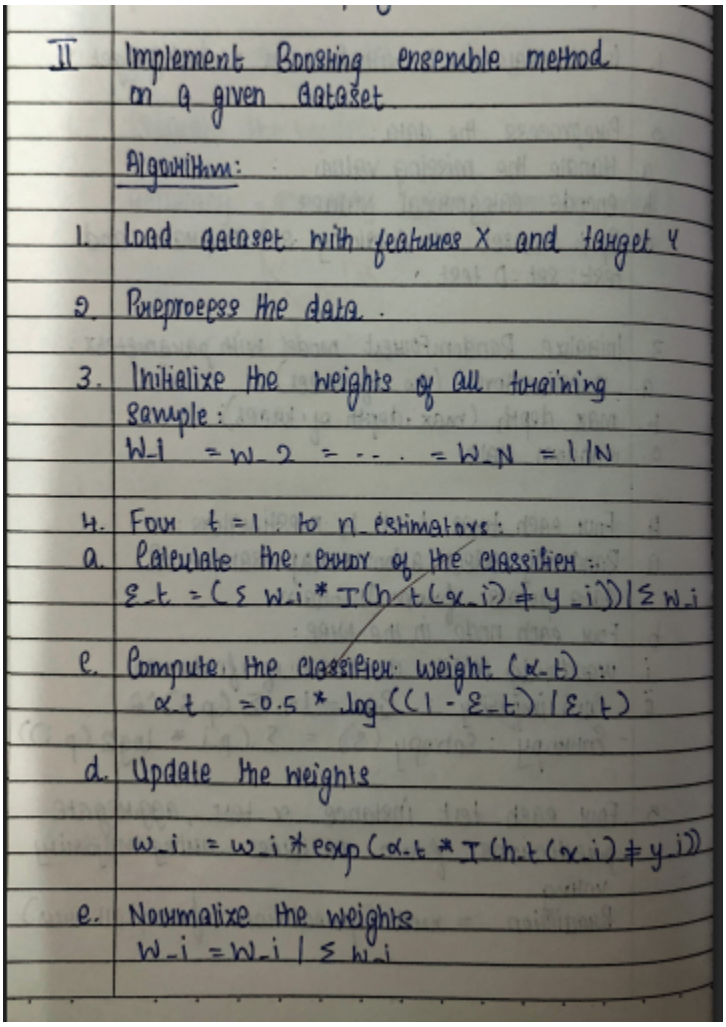
```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

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

# Make predictions and calculate accuracy
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Random Forest Classifier Accuracy on Titanic Dataset: {accuracy:.4f}')


# Feature importance visualization
importances = clf.feature_importances_
features = X.columns

# Plot feature importance
plt.barh(features, importances)
plt.xlabel('Feature Importance')
plt.title('Feature Importance for Titanic Dataset - Random Forest')
plt.show()
```

Output:



Program 9

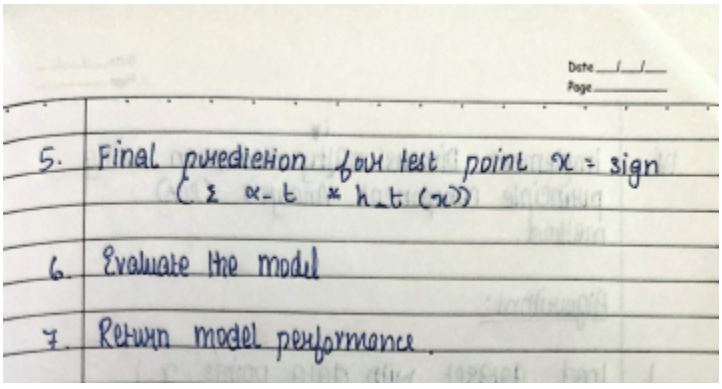
Observation:



II Implement Boosting ensemble method on a given dataset.

Algorithm:

1. Load dataset with features X and target Y
2. Preprocess the data.
3. Initialize the weights of all training sample:
   $W_1 = W_2 = \ldots = W_N = 1/N$
4. For $t = 1$ to n_estimators:
   a. Calculate the error of the classifier:
   $\varepsilon_t = (\sum w_i * I(h_t(x_i) \neq y_i)) / \sum w_i$
   e. Compute the classifier weight (α_t):
   $\alpha_t = 0.5 * \log((1 - \varepsilon_t) / \varepsilon_t)$
   d. Update the weights
   $w_i = w_i * \exp(\alpha_t * I(h_t(x_i) \neq y_i))$
   e. Normalize the weights
   $w_i = w_i / \sum w_i$



Date __/__/__
Page _____

5. Final prediction for test point x = sign $(\sum \alpha_t * h_t(x))$
6. Evaluate the model
7. Return model performance.

Code:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
```

```
X = iris.data
y = iris.target

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

# Initialize Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(n_estimators=100, random_state=42)

# Train the model
gb_clf.fit(X_train, y_train)

# Make predictions
y_pred = gb_clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Gradient Boosting Classifier Accuracy on Iris Dataset: {accuracy:.4f}')
```

Output:

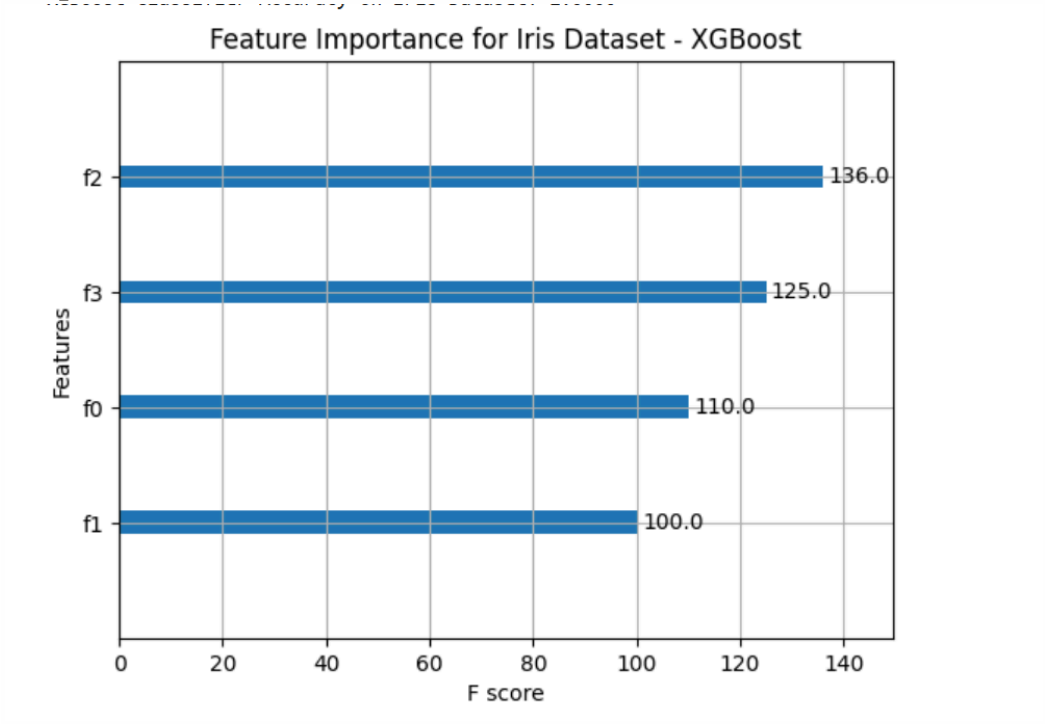
```
Gradient Boosting Classifier Accuracy on Iris Dataset: 1.0000
```

Boosting Ensemble (XGBoost) on Iris Dataset (Visualization)

```
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

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

# Initialize XGBoost Classifier
xgb_clf = xgb.XGBClassifier(n_estimators=100, random_state=42)

# Train the model
xgb_clf.fit(X_train, y_train)

# Make predictions
y_pred = xgb_clf.predict(X_test)

# Calculate accuracy
accuracy = (y_pred == y_test).mean()
```

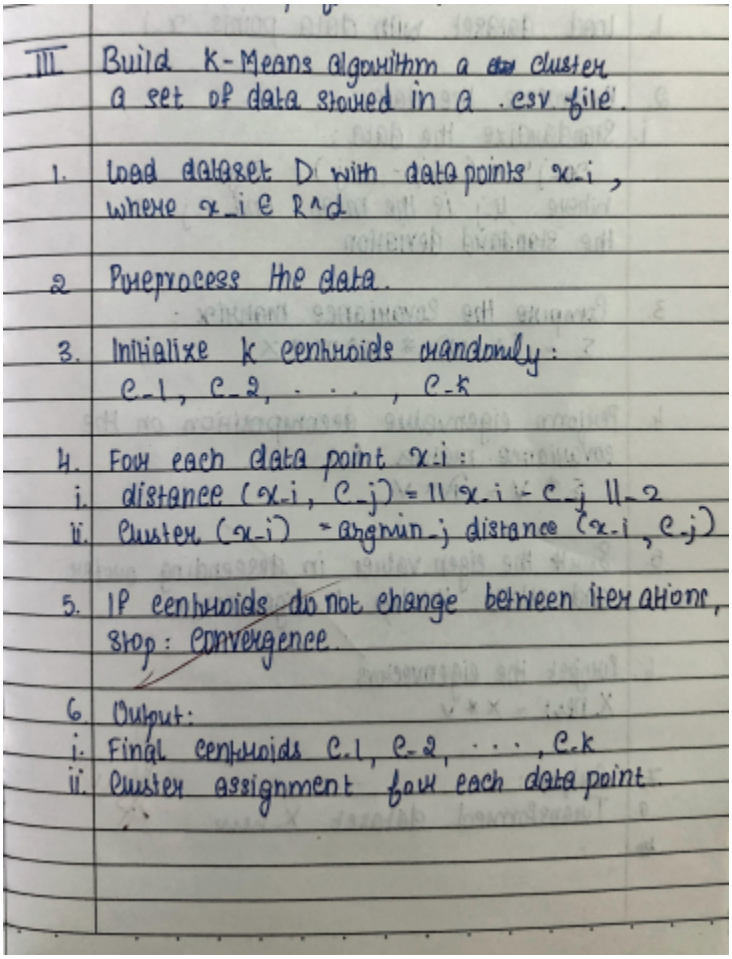
```
print(f'XGBoost Classifier Accuracy on Iris Dataset: {accuracy:.4f}')

# Feature importance visualization
xgb.plot_importance(xgb_clf, importance_type='weight', max_num_features=10)
plt.title('Feature Importance for Iris Dataset - XGBoost')
plt.show()
```

Output:



Program 10

Observation:



Code:

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.model_selection import train_test_split

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Perform K-means clustering (3 clusters for 3 species in Iris)
kmeans = KMeans(n_clusters=3, random_state=42)
y_kmeans = kmeans.fit_predict(X)

# Since K-means is an unsupervised algorithm, the cluster labels don't directly match the target labels
# To calculate accuracy, we map cluster labels to actual species labels
# Here we assume that the true labels are known and compare them with the clusters' output

# We use the mode (most frequent) of the true class labels for each cluster to match them
from scipy.stats import mode
```

```
cluster_labels = np.zeros_like(y_kmeans)
for i in range(3):  # 3 clusters
    mask = (y_kmeans == i)
    cluster_labels[mask] = mode(y[mask])[0]


# Calculate accuracy by comparing the mapped cluster labels with true labels
accuracy = accuracy_score(y, cluster_labels)
print(f'K-means Clustering Accuracy on Iris Dataset: {accuracy:.4f}')
```

output:

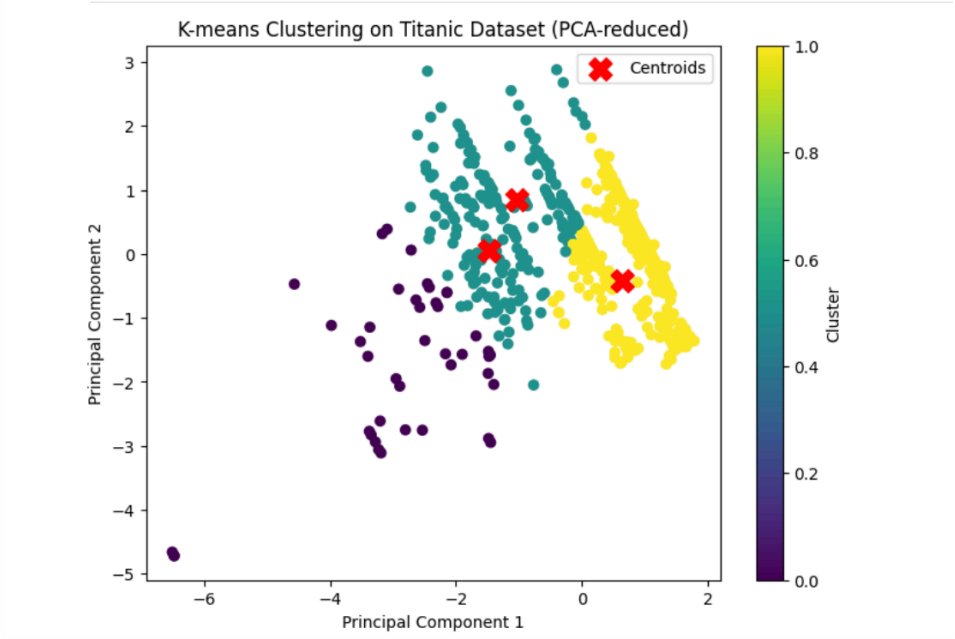
```
K-means Clustering Accuracy on Iris Dataset: 0.8867
```

K-means on a Kaggle Dataset (Visualization)

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# Load Titanic dataset (You can replace the URL with your own Kaggle dataset link)
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)

# Preprocessing: Selecting relevant features and dropping missing values
data = data.dropna(subset=['Pclass', 'Age', 'Fare'])
X = data[['Pclass', 'Age', 'Fare']]

# Normalize the data (optional, but helps with clustering)
X = (X - X.mean()) / X.std()

# Apply K-means clustering (let's assume we use 3 clusters for this example)
kmeans = KMeans(n_clusters=3, random_state=42)
y_kmeans = kmeans.fit_predict(X)

# Apply PCA for 2D visualization (reduce to 2 components)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot the clusters in 2D
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X',
label='Centroids')
plt.title('K-means Clustering on Titanic Dataset (PCA-reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
```

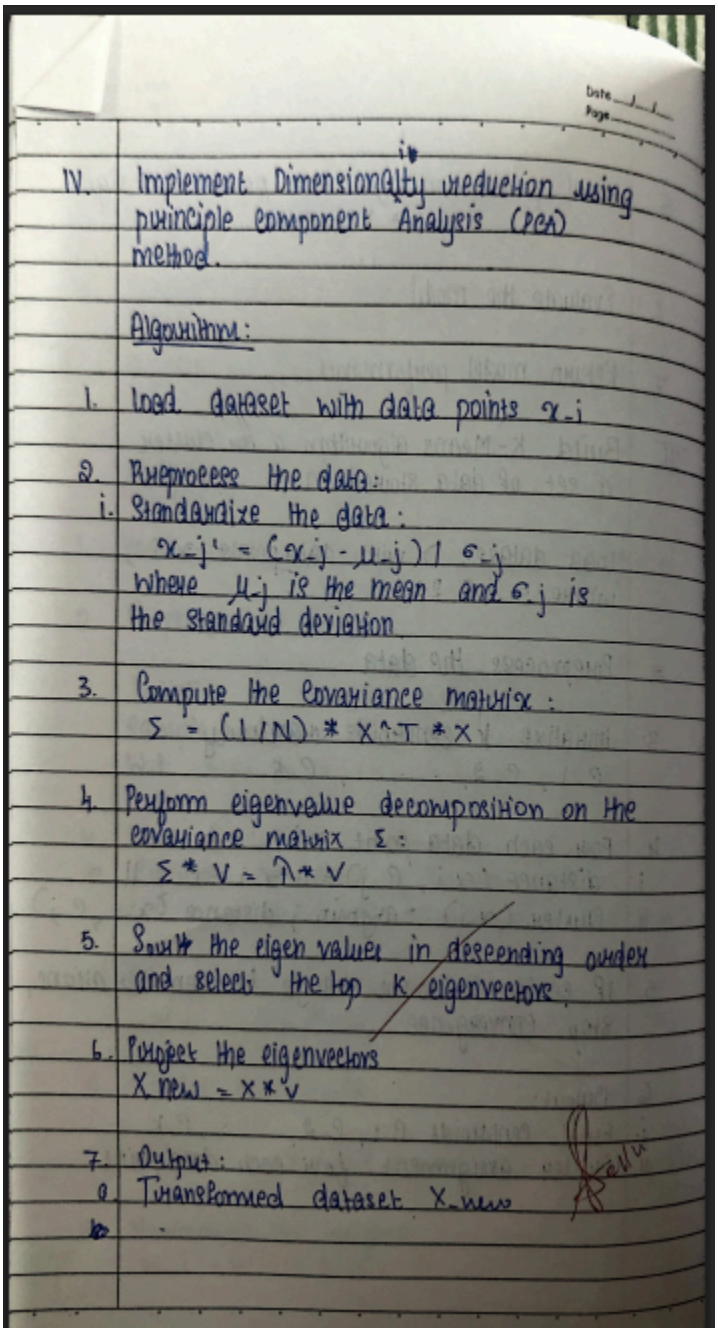
```
plt.legend()
plt.show()
```

Output:



Program 11

Observation:



IV. Implement Dimensionality reduction using principle component Analysis (PCA) method.

Algorithm:

1. Load dataset with data points x_i
2. Preprocess the data:
   i. Standardize the data:
   $x\_j' = (x\_j - \mu\_j) / \sigma\_j$
   where μ_j is the mean and σ_j is the standard deviation
3. Compute the covariance matrix:
   $\Sigma = (1/N) * X^T * X$
4. Perform eigenvalue decomposition on the covariance matrix Σ:
   $\Sigma * V = \lambda * V$
5. Sort the eigen values in descending order and select the top K eigenvectors.
6. Project the eigenvectors
   X_new = X * V
7. Output:
   a. Transformed dataset X_new
   b.

Code:

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X = iris.data
```

```
y = iris.target

# Apply PCA to reduce data to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

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

# Train a Logistic Regression model
clf = LogisticRegression(max_iter=200)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy after PCA on Iris Dataset: {accuracy:.4f}')
```

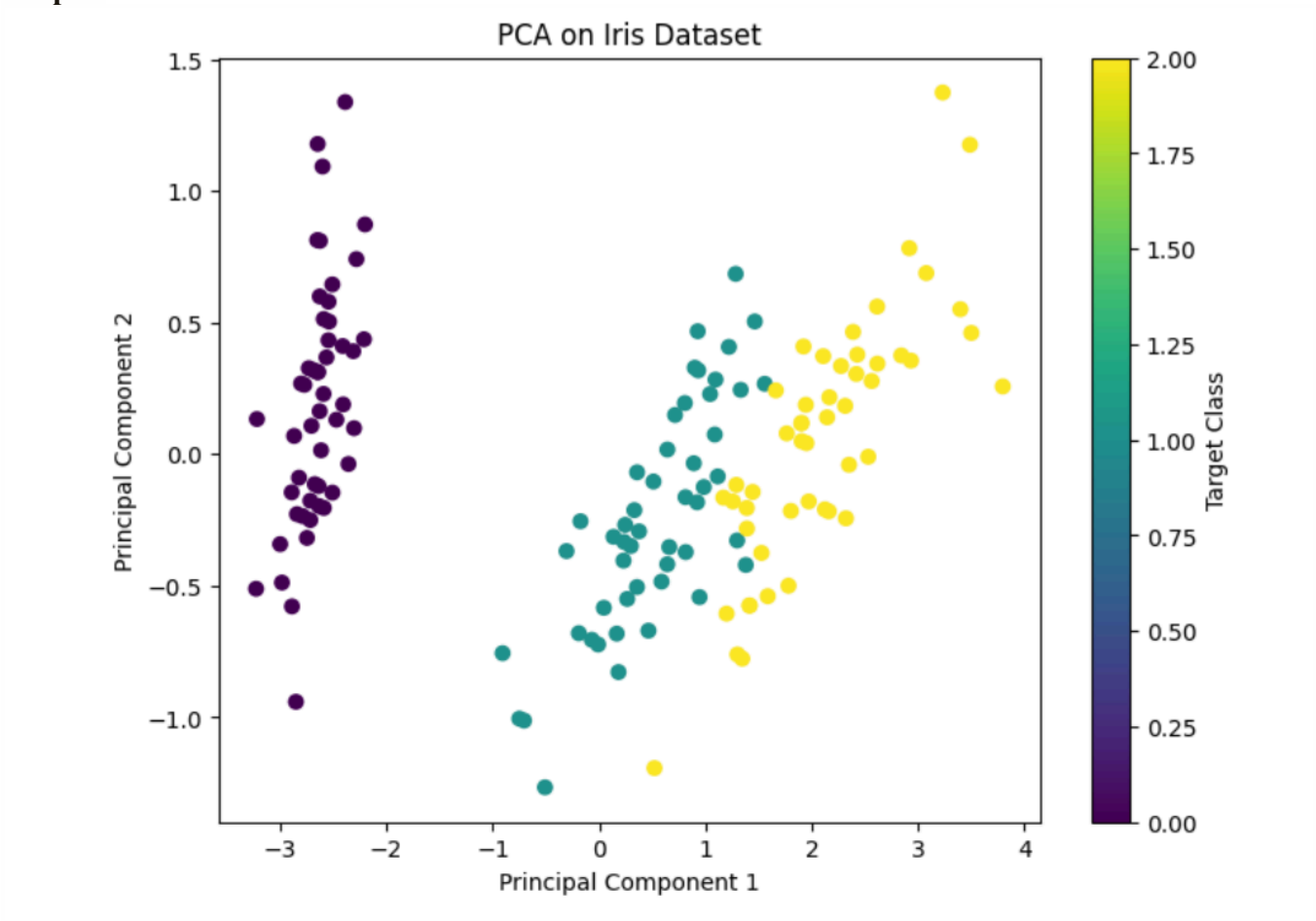
Output:

```
Accuracy after PCA on Iris Dataset: 1.0000
```

PCA on Iris Dataset (Visualization)

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Apply PCA to reduce data to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot the PCA-reduced data
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.title('PCA on Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Target Class')
plt.show()
```

Output:


PCA on Iris Dataset
Principal Component 2
Principal Component 1
Target Class
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00