VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Anup Vaidya(1BM22CS047)

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 Anup Vaidya(1BM22CS047)**,** 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 a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

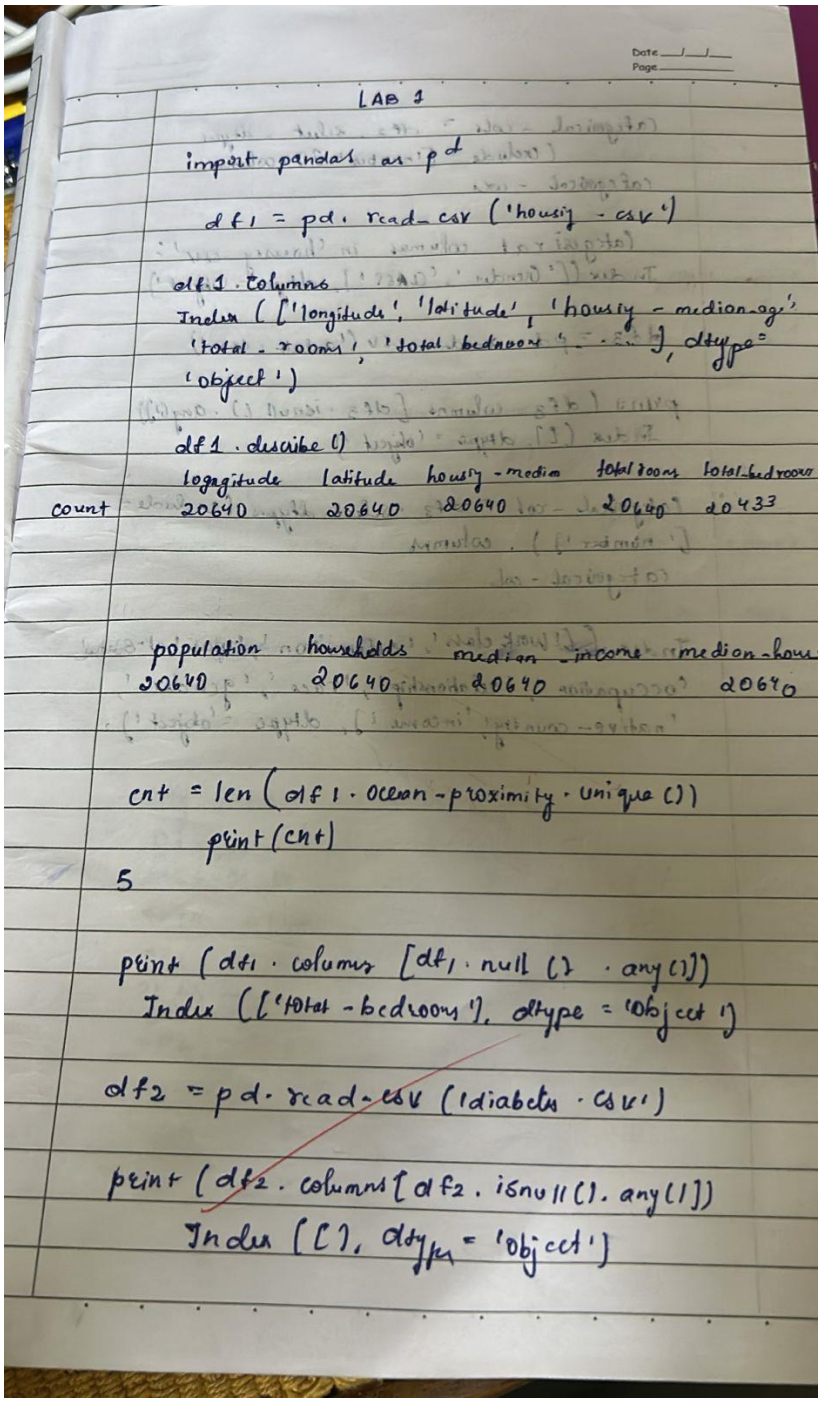
Lab Faculty Incharge	
Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

Program 1

Write a python program to import and export data using Panda's library functions

Screenshot



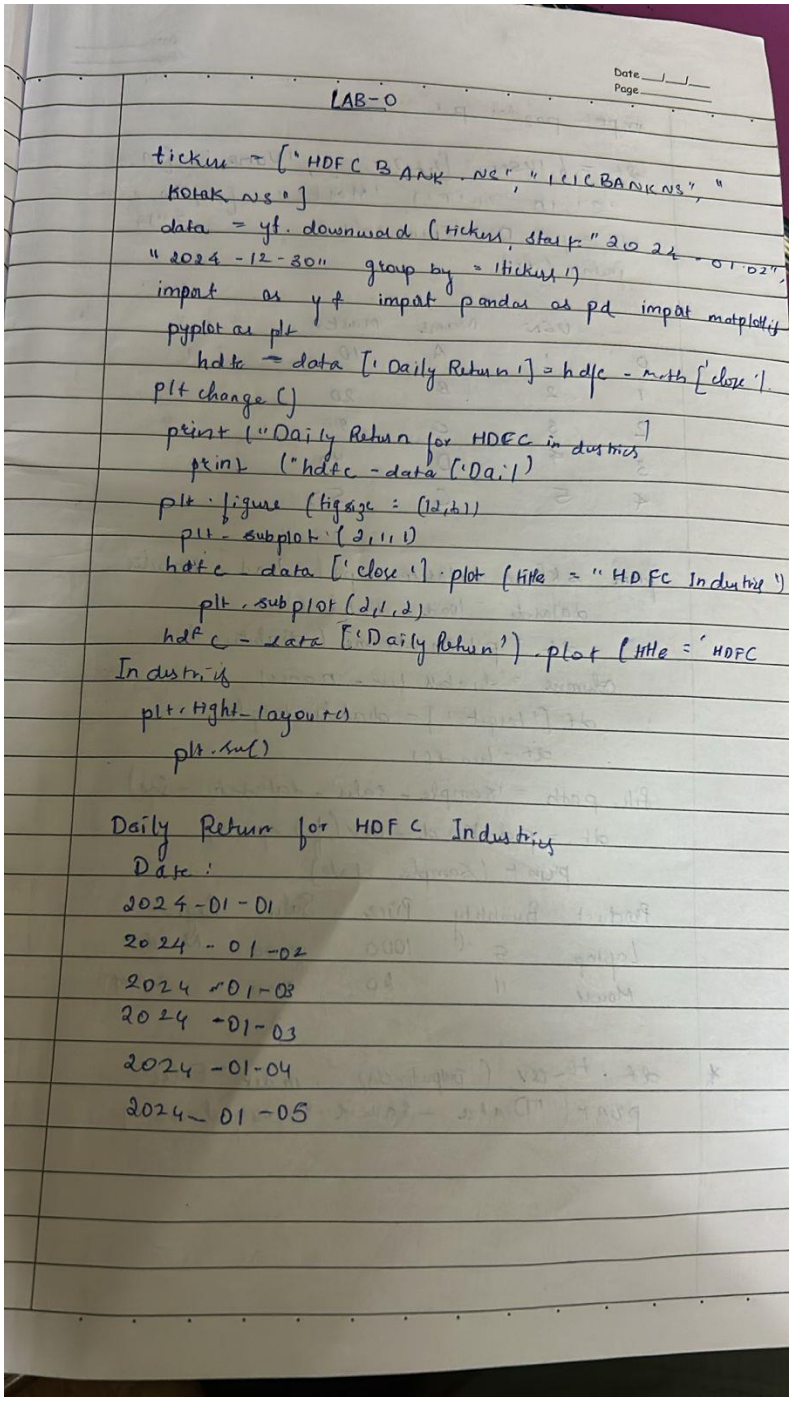
Code:

```
import pandas as pd
try:
    df = pd.read_csv('input.csv')
    print("Data imported successfully!\n")
    print(df)
except FileNotFoundError:
    print("The file 'input.csv' was not found.")
df["Processed"] = True
df.to_csv('output.csv', index=False)
print("\nData exported successfully to 'output.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshots:



```
LAB-0

tickers = ['HDFC BANK.NS', 'ICICBANK.NS', 'KOTAK.NS']
data = yf.download(tickers, start="2022-01-01",
"2024-12-30" group by = 'ticker')
import as yf import pandas as pd import matplotlib
pyplot as plt
hdfc = data['Daily Return'] = hdfc.pct['close'].
plt.change()
print("Daily Return for HDFC in dustries
print("hdfc-data['Dai']
plt.figure(figsize=(12,2))
plt.subplot(2,1,1)
hdfc data['close'].plot(title="HDFC Industries")
plt.subplot(2,1,2)
hdfc-data['Daily Return'].plot(title='HDFC
Industries
plt.tight_layout()
plt.show()

Daily Return for HDFC Industries
Date:
2024-01-01
2024-01-02
2024-01-03
2024-01-03
2024-01-04
2024-01-05
```

Code

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler


data = {

    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', None],

    'Age': [25, 30, np.nan, 35, 29, 40],
```

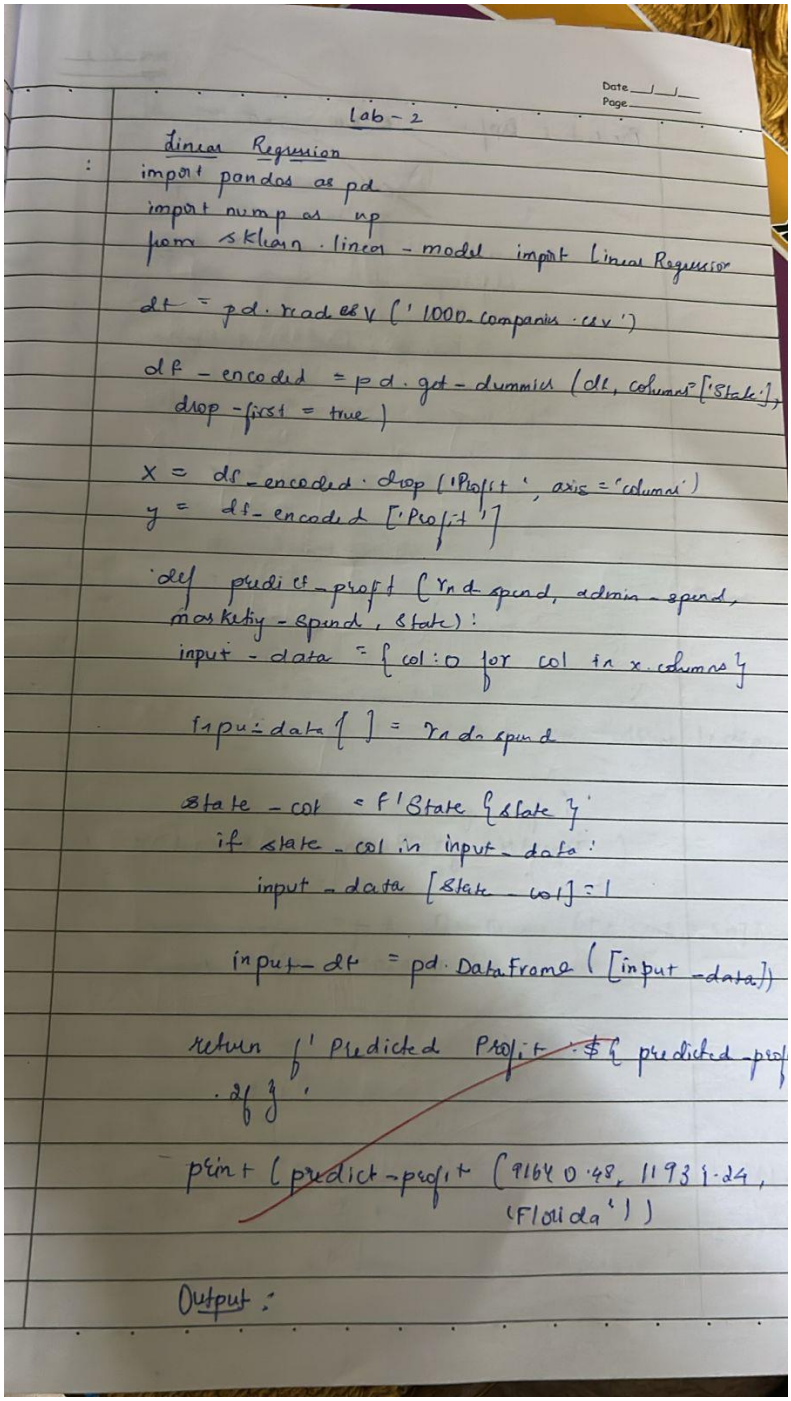
```
    'Department': ['HR', 'IT', 'Finance', 'IT', 'HR', 'Finance'],
    'Salary': [50000, 60000, 58000, 62000, np.nan, 52000]
}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Salary'].fillna(df['Salary'].median(), inplace=True)
df['Name'].fillna('Unknown', inplace=True)

le = LabelEncoder()
df['Department_Encoded'] = le.fit_transform(df['Department'])

df.drop_duplicates(inplace=True)

df.rename(columns={'Salary': 'Monthly_Salary'}, inplace=True)

df['Age'] = df['Age'].astype(int)

scaler = MinMaxScaler()
df['Salary_Normalized'] = scaler.fit_transform(df[['Monthly_Salary']])

standard_scaler = StandardScaler()
```

```
df['Age_Standardized'] = standard_scaler.fit_transform(df[['Age']])


print("\nPreprocessed DataFrame:\n", df)
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots

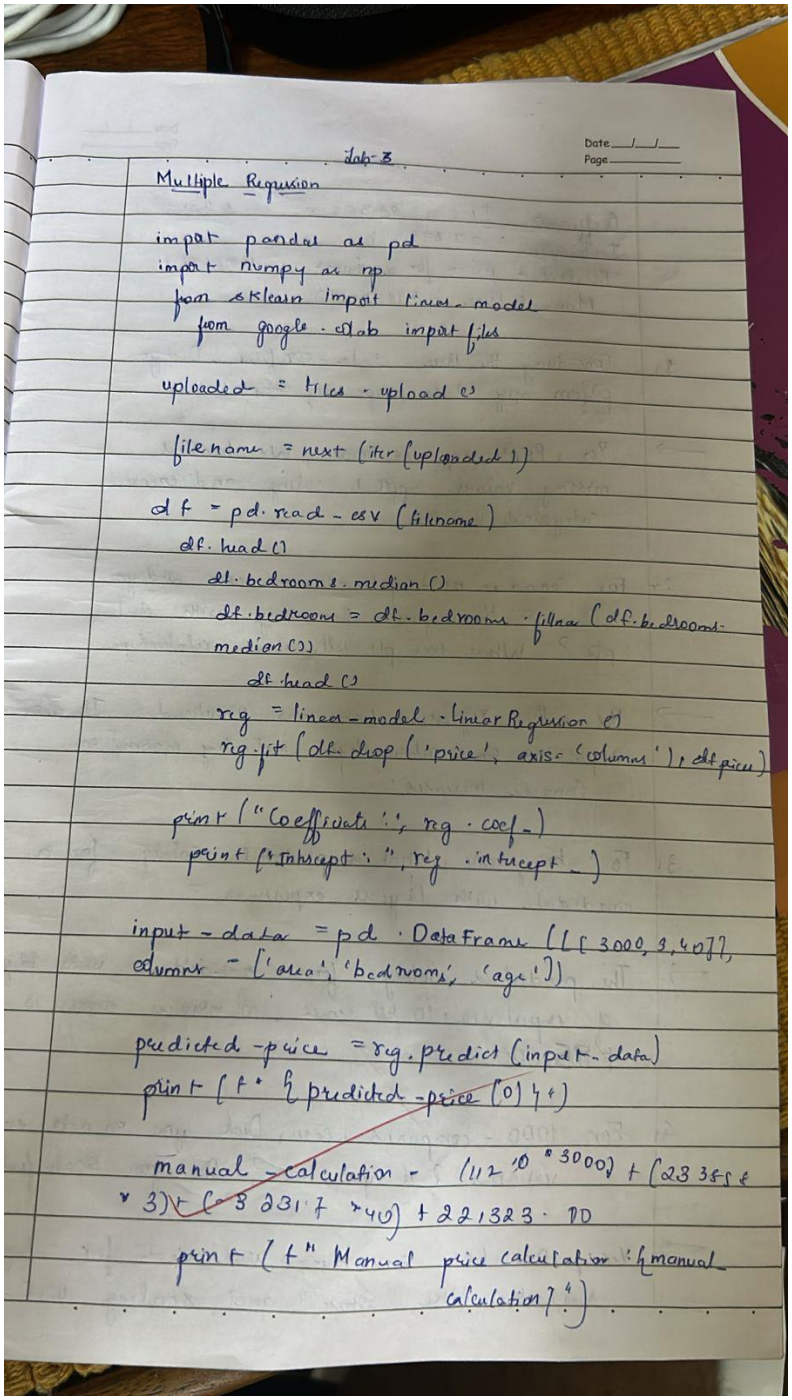


Lab - 2

Linear Regression

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

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

df_encoded = pd.get_dummies(df, columns=['State'],
drop_first = True)

X = df_encoded.drop('Profit', axis='columns')
y = df_encoded['Profit']

def predict_profit(rnd_spend, admin_spend,
marketing_spend, state):
input_data = {col:0 for col in X.columns}

input_data['R&D Spend'] = rnd_spend

state_col = f'State_{state}'
if state_col in input_data:
    input_data[state_col] = 1

input_df = pd.DataFrame([input_data])

return f'Predicted Profit: ${predicted_prof
.2f}'

print(predict_profit(91640.48, 11931.24,
'Florida'))
```

Output :



Code

Linear Regression

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

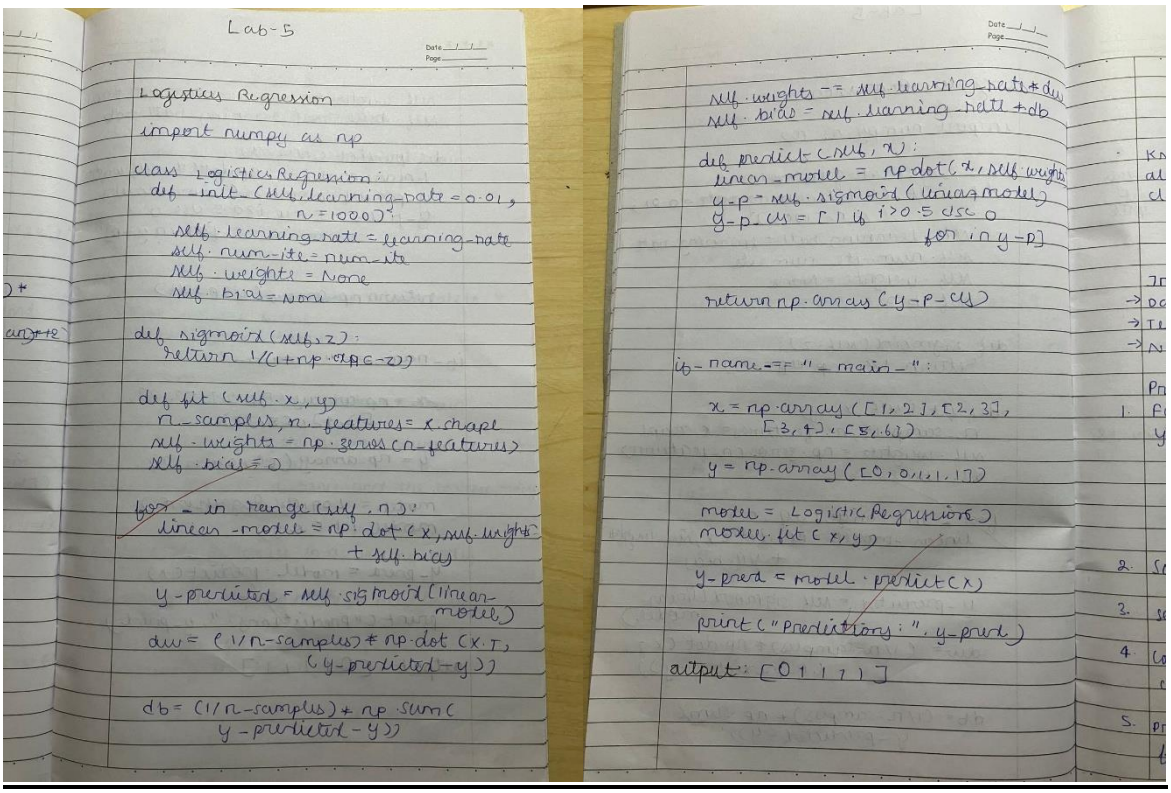
```
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target

# Use only one feature for simple linear regression (e.g., RM = average number of rooms)
X = df[['RM']]
y = df['PRICE']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
# Predict
y_pred = lr.predict(X_test)

# Output
print("Linear Regression Results")
print("Coefficients:", lr.coef_)
print("Intercept:", lr.intercept_)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Plot
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red')
plt.xlabel('Average Number of Rooms (RM)')
plt.ylabel('House Price')
plt.title('Simple Linear Regression')
plt.show()

# Multiple Linear Regression

# Use all features
X = df.drop('PRICE', axis=1)
y = df['PRICE']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train model
mlr = LinearRegression()
mlr.fit(X_train, y_train)

# Predict
y_pred = mlr.predict(X_test)

# Output
print("\nMultiple Linear Regression Results")
print("Coefficients:", mlr.coef_)
print("Intercept:", mlr.intercept_)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot’s



Code

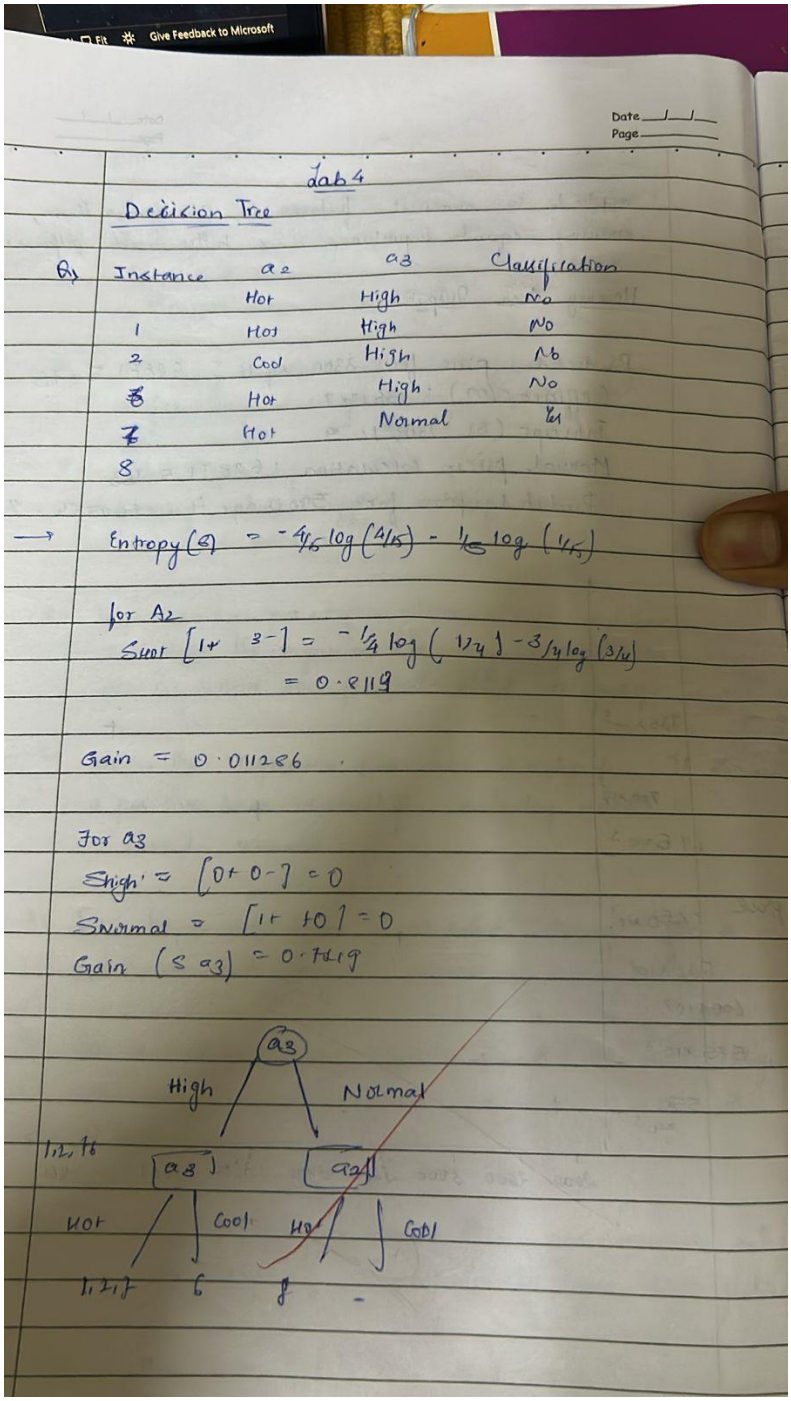
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
```

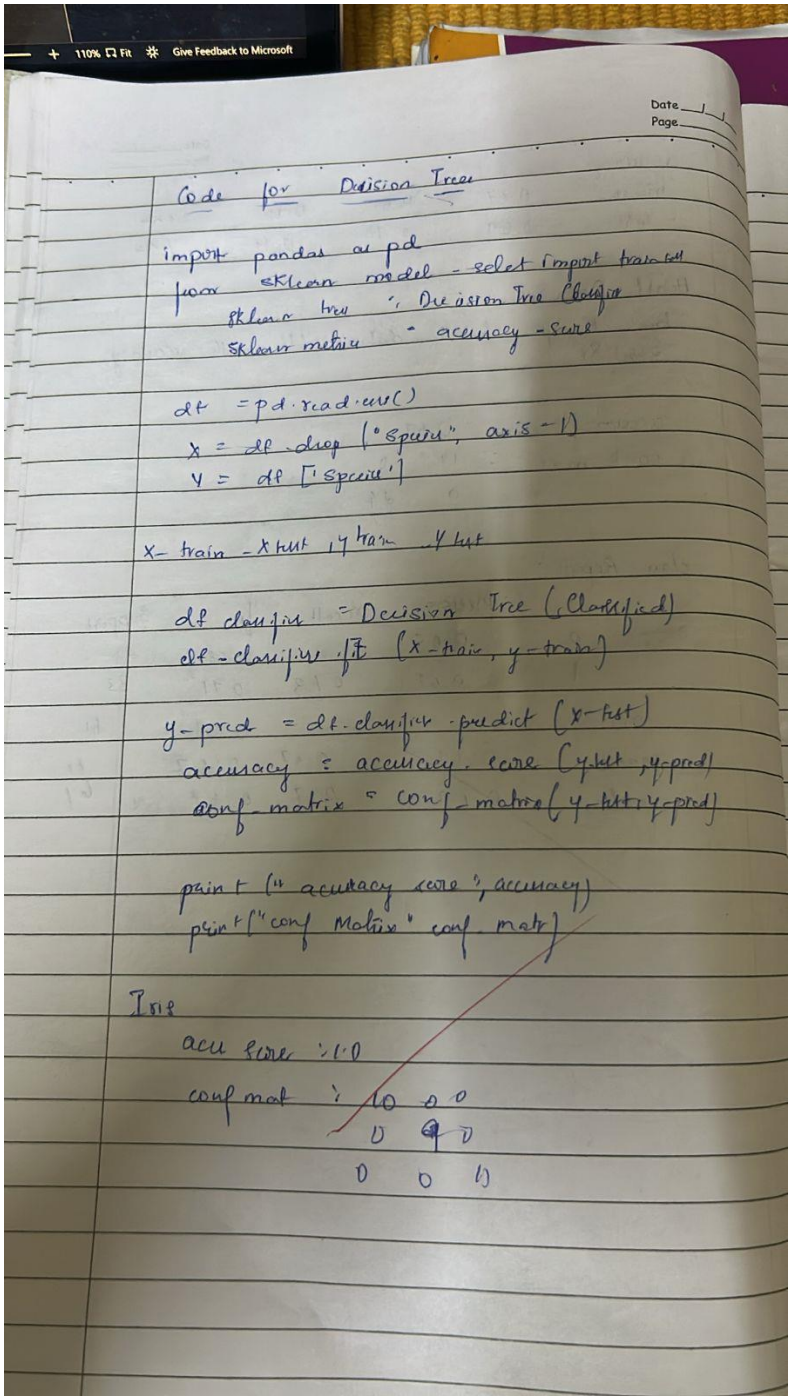
```
df_binary = df[df['species'] != 2]  # Remove class 2 (Virginica)

X = df_binary.iloc[:, :-1]   # Features

y = df_binary['species']      # Target (0 or 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)

# Step 5: Predict and evaluate

y_pred = model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nAccuracy:", accuracy_score(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Program 5

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

Screenshots





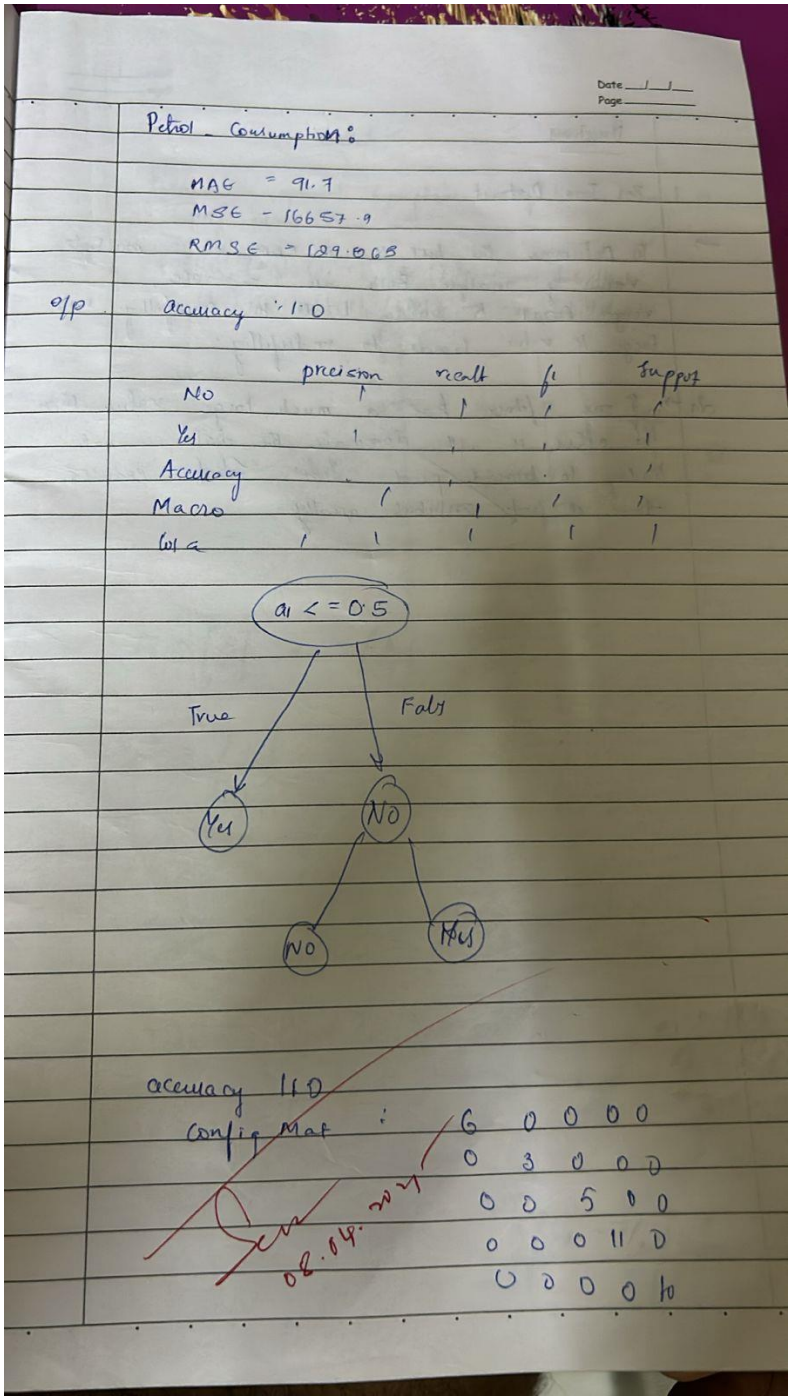
Code for Decision Tree

```
import pandas as pd
from sklearn model - select import train test
    sklearn tree ; Decision Tree Classifier
    sklearn metrics - accuracy - score

df = pd.read.csv()
x = df.drop ("Species", axis = 1)
y = df ['Species']

x- train - x test , y train - y test

df classifier = Decision Tree (, Classified)
df - classifier.fit (x-train, y-train)

y-pred = df.classifier.predict (x-test)
accuracy = accuracy. score (y test , y-pred)
conf_matrix = conf_matrix( y-test, y-pred)

print ("accuracy score", accuracy)
print ("conf Matrix" conf. matr)
```

Iris

acc score : 1.0

conf mat : 10 0 0
0 9 0
0 0 11



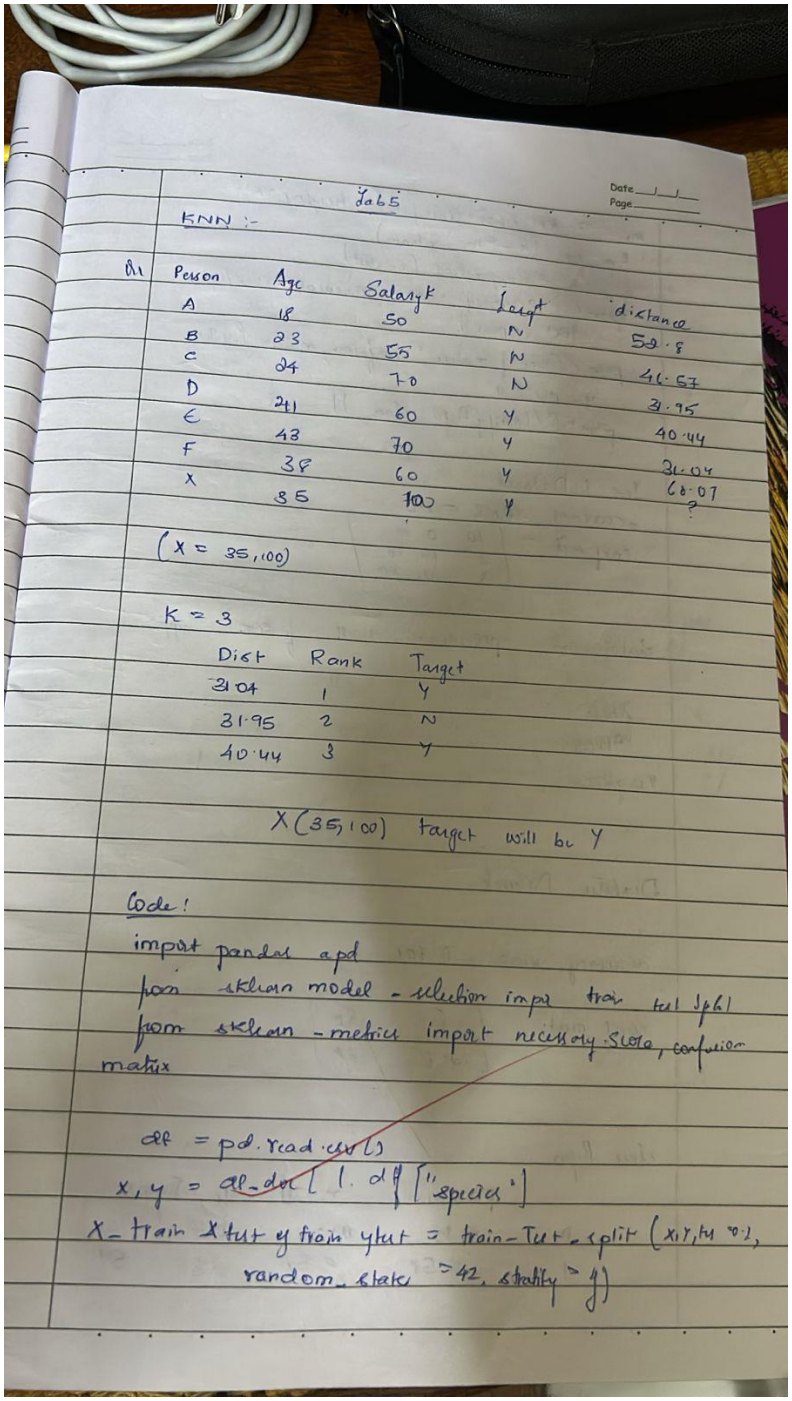
Code

import pandas as pd

```
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
data = {
   'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast',
               'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],
   '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'],
   'Wind': ['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)
le = LabelEncoder()
for column in df.columns:
   df[column] = le.fit_transform(df[column])

# Step 3: Separate features and label
X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']
clf = DecisionTreeClassifier(criterion='entropy')  # ID3 uses 'entropy'
```

```
clf = clf.fit(X, y)

print("\nDecision Tree Rules:")

tree_text = tree.export_text(clf, feature_names=X.columns.tolist())

print(tree_text)

# Example: Outlook=Rain, Temperature=Mild, Humidity=High, Wind=Weak

# Encode input sample with same label encoding order used earlier

sample = pd.DataFrame({

    'Outlook': [le.transform(['Rain'])[0]],

    'Temperature': [le.transform(['Mild'])[0]],

    'Humidity': [le.transform(['High'])[0]],

    'Wind': [le.transform(['Weak'])[0]]

})


# Predict

prediction = clf.predict(sample)

result = 'Yes' if prediction[0] == 1 else 'No'

print(f"\nPrediction for new sample (Rain, Mild, High, Weak): {result}")
```

Program 6 Build KNN Classification model for a given dataset



Lab5

KNN :-

Q1

Person	Age	Salary k	Loyal	distance
A	18	50	N	59.8
B	23	55	N	41.57
C	24	70	N	31.95
D	21	60	Y	40.44
E	43	70	Y	31.04
F	38	60	Y	68.07
X	35	100	Y	?

(X = 35, 100)

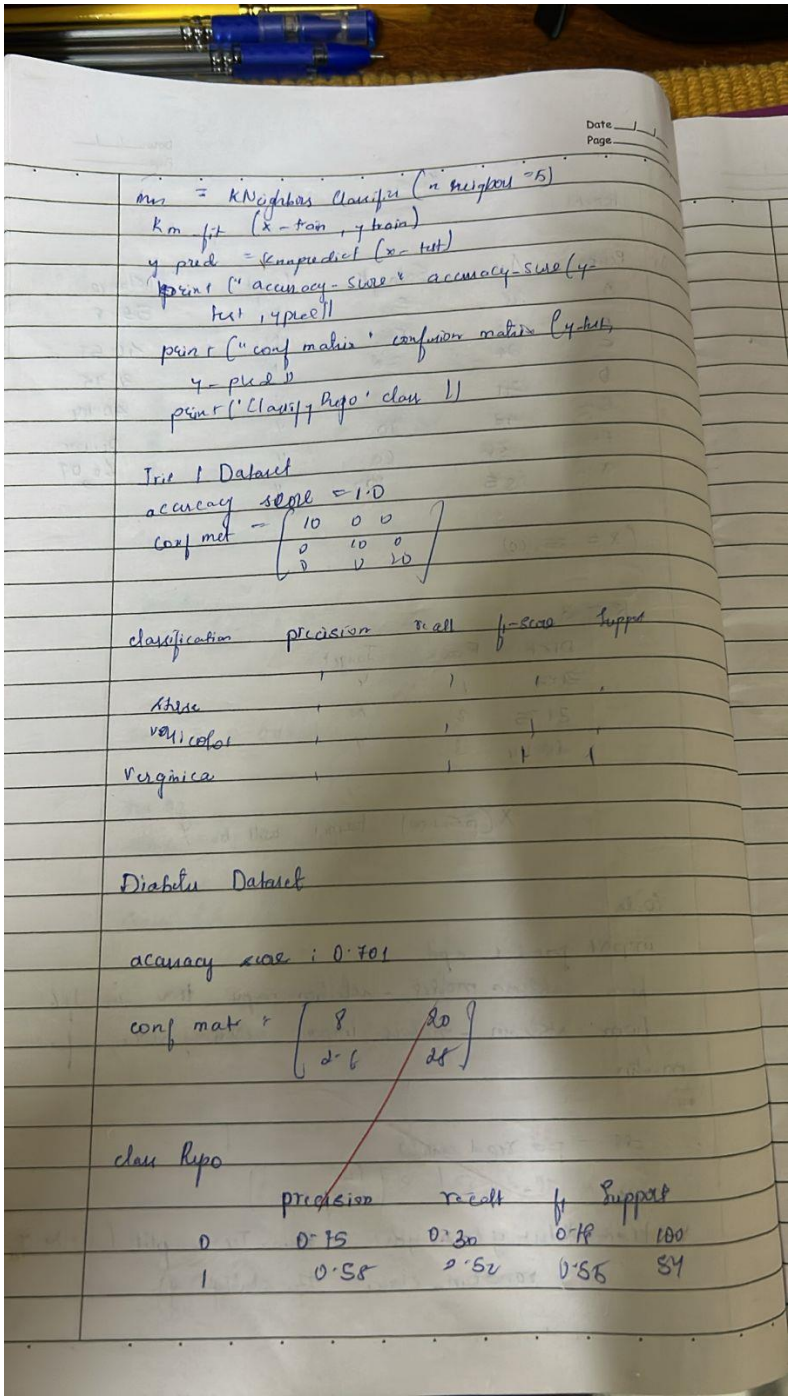
K = 3

Dist	Rank	Target
31.04	1	Y
31.95	2	N
40.44	3	Y

X(35, 100) target will be Y

Code :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

df = pd.read_csv()
x, y = df.drop(['species'])
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
        random_state=42, stratify=y)
```



Code

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Step 1: Load the Iris dataset

iris = load_iris()

X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
y = pd.Series(iris.target)

# Step 2: Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Build KNN model (k = 3)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Step 4: Predict and evaluate
y_pred = knn.predict(X_test)

# Results
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 7

Build Support vector machine model for a given dataset

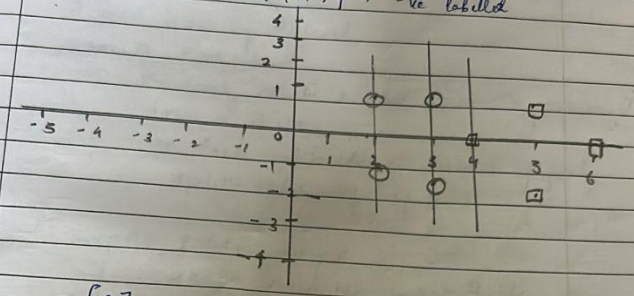
Screenshots

Date / /
Page

Lab 7 :-

Draw an optimal hyperplane using linear sum to classify points.

{ (1,1), (2,1), (1,-1), (2,-1) } → +ve labelled
{ (4,0), (5,1), (5,-1), (6,0) } → -ve labelled



$s_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ $s_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$ $s_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

$\bar{s_1} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$ $\bar{s_2} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$ $\bar{s_3} = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

$\alpha_1 \bar{s_1} \bar{s_1} + \alpha_2 \bar{s_2} \bar{s_1} + \alpha_3 \bar{s_3} \bar{s_1} = +1$
$\alpha_1 \bar{s_1} \bar{s_2} + \alpha_2 \bar{s_2} \bar{s_2} + \alpha_3 \bar{s_3} \bar{s_2} = +1$
$\alpha_1 \bar{s_1} \bar{s_3} + \alpha_2 \bar{s_2} \bar{s_3} + \alpha_3 \bar{s_3} \bar{s_3} = -1$

$\alpha_1 (6) + \alpha_2 (4) + \alpha_3 (9) = +1$
$\alpha_1 (4) + \alpha_2 (6) + \alpha_3 (9) = +1$
$\alpha_1 (9) + \alpha_2 (9) + \alpha_3 (13) = -1$

$\alpha_1 = 13/4$
$\alpha_2 = 13/4$
$\alpha_3 = -7/2$

$w = \alpha_1 \bar{s_1} + \alpha_2 \bar{s_2} + \alpha_3 \bar{s_3}$

$= 13/4 \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} + 13/4 \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} - 7/2 \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 3 \end{bmatrix} = -\begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix}$

Code

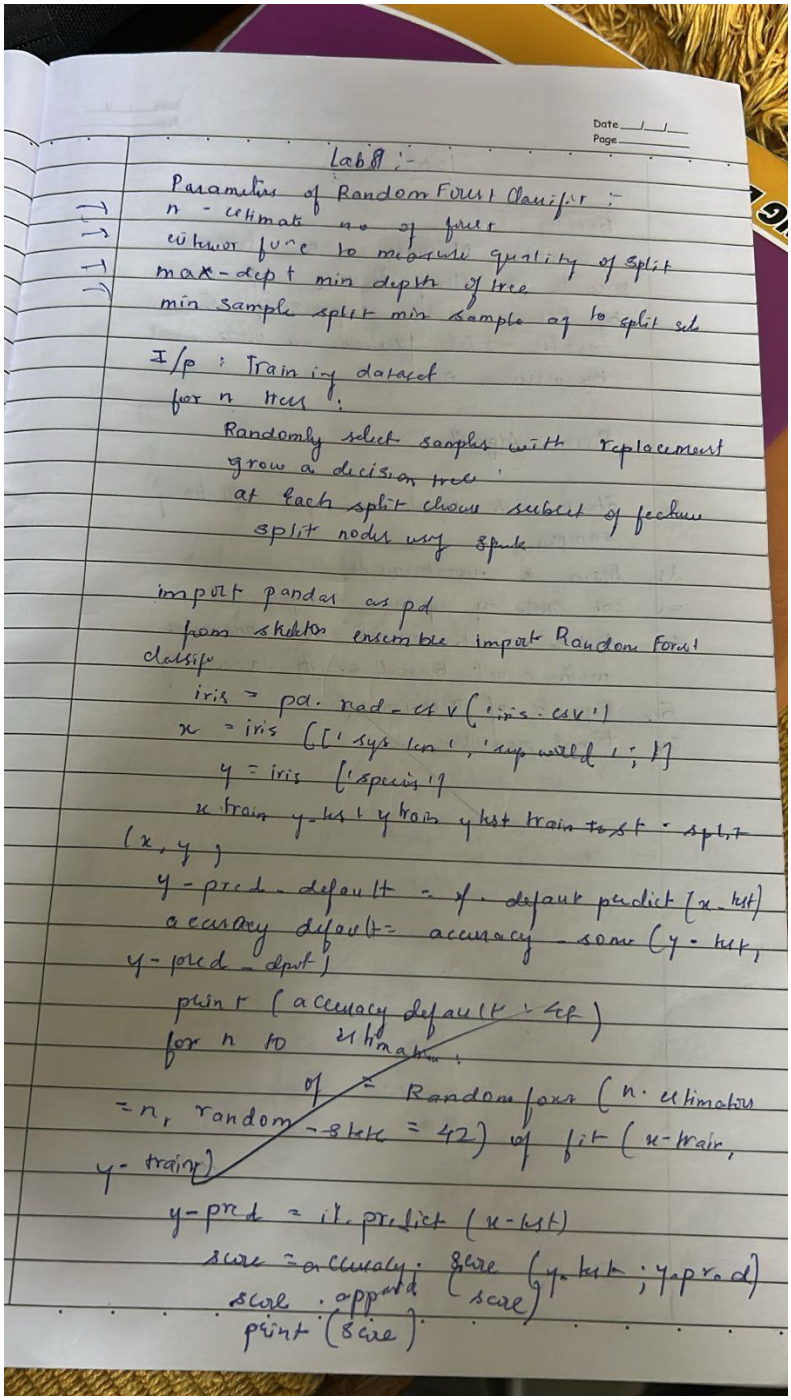
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# Step 1: Load the Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# Step 2: Split the data into train and test sets (80% train, 20% test)
```

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

# Step 3: Build and train the SVM model (linear kernel)
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Step 4: Predict and evaluate
y_pred = svm_model.predict(X_test)

# Output results
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 8

Implement Random Forest ensemble method on a given dataset

Screenshots



Lab 8 :-

Parameters of Random Forest Classifier :

- n - estimators no of trees
- criterion fun to measure quality of split
- max-depth min depth of tree
- min sample split min sample req to split node

I/p : Training dataset

for n trees :

Randomly select samples with replacement

grow a decision tree

at each split choose subset of features

split nodes using split

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
iris = pd.read_csv('iris.csv')
x = iris[['sepal length', 'sepal width']]
y = iris['species']
x_train, x_test, y_train, y_test = train_test_split(x, y)
y_pred_default = rf_default.predict(x_test)
accuracy_default = accuracy_score(y_test, y_pred_default)
print('accuracy default:', ap)
for n in estimators:
    rf = RandomForest(n_estimators=n, random_state=42).fit(x_train, y_train)
    y_pred = rf.predict(x_test)
    score = accuracy_score(y_test, y_pred)
    score.append(score)
    print(score)
```

Code

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


# Step 1: Load Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)
```

```
    # Step 2: Split into train and test sets (80% train, 20% test)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Step 3: Train Random Forest model
    rf = RandomForestClassifier(n_estimators=100, random_state=42)  # 100 trees
    rf.fit(X_train, y_train)

    # Step 4: Predict and evaluate
    y_pred = rf.predict(X_test)

    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshots

Date / /
Page

Lab 9 :-

Boosting :-

Combine Multiple weak learners to create a strong learner. It works by training models sequentially where each model focuses on errors made by previous one.

Basic Algorithm :-

1. Start with equal weight for all training samples.
2. Train a weak model.
3. Calculate error & update sample weights.
4. Add weak model to ensemble with a wt based on its accuracy.
5. Repeat of n estimators.
6. Final prediction

Code

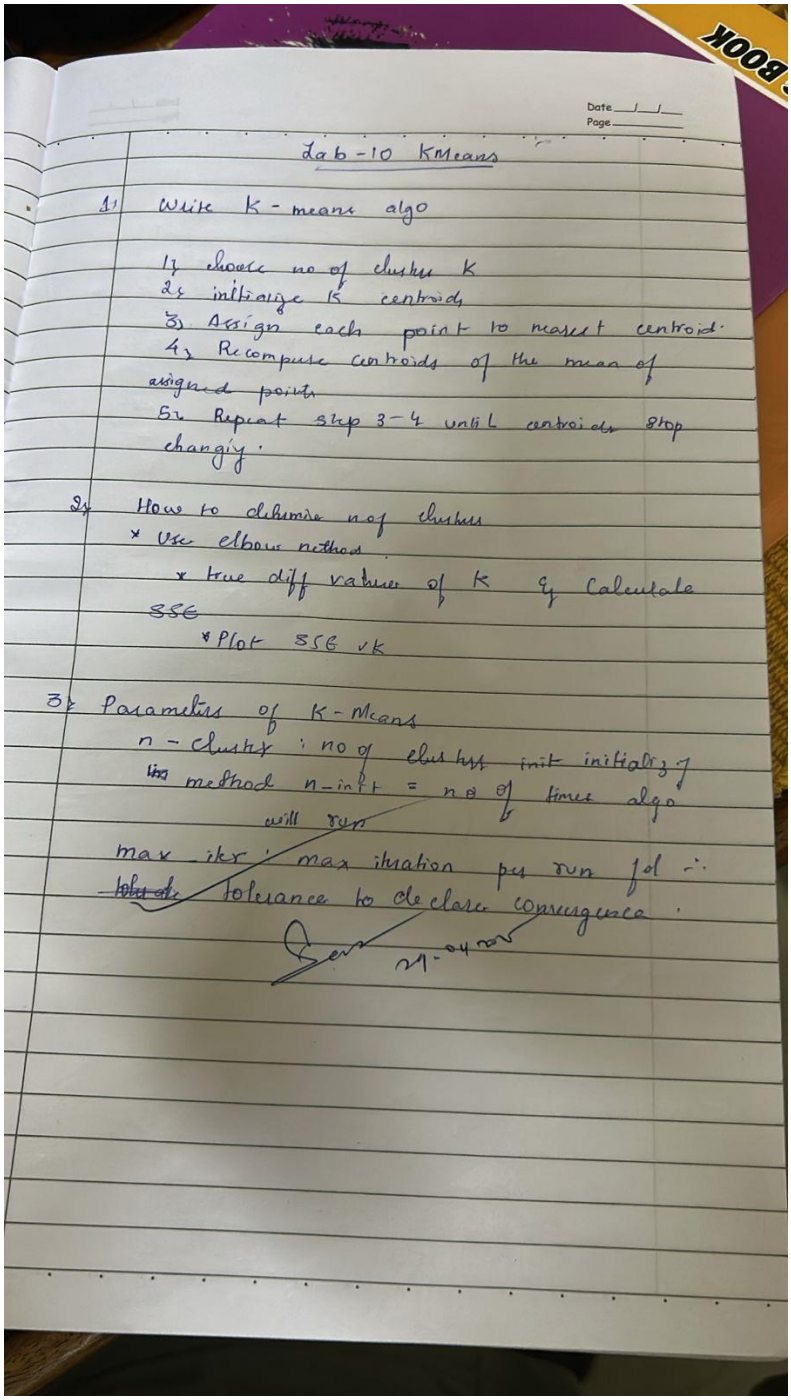
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.tree import DecisionTreeClassifier

# Step 1: Load the Iris dataset
iris = load_iris()
```

```
X = pd.DataFrame(iris.data, columns=iris.feature_names)

y = pd.Series(iris.target)


# Step 2: Split into train and test sets (80% train, 20% test)

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


# Step 3: Build AdaBoost model with DecisionTreeClassifier as base estimator

base_estimator = DecisionTreeClassifier(max_depth=1)

model = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50, learning_rate=1.0,
random_state=42)

model.fit(X_train, y_train)


# Step 5: Predict and evaluate

y_pred = model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))

print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Program 10

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

Screenshots



Date __/__/__
Page

Lab-10 KMeans

1) Write K-means algo

1) choose no of clusters K
2) initialize K centroids
3) Assign each point to nearest centroid.
4) Recompute centroids of the mean of assigned points
5) Repeat step 3-4 until centroids stop changing.

2) How to determine no of clusters
* Use elbow method.
* Run diff values of K & Calculate SSE
* Plot SSE vK

3) Parameters of K-Means
n-clusters : no of clusters init initializing method n_init = no of times algo will run
max_iter : max iteration per run for ~~one~~ tolerance to declare convergence.

Code

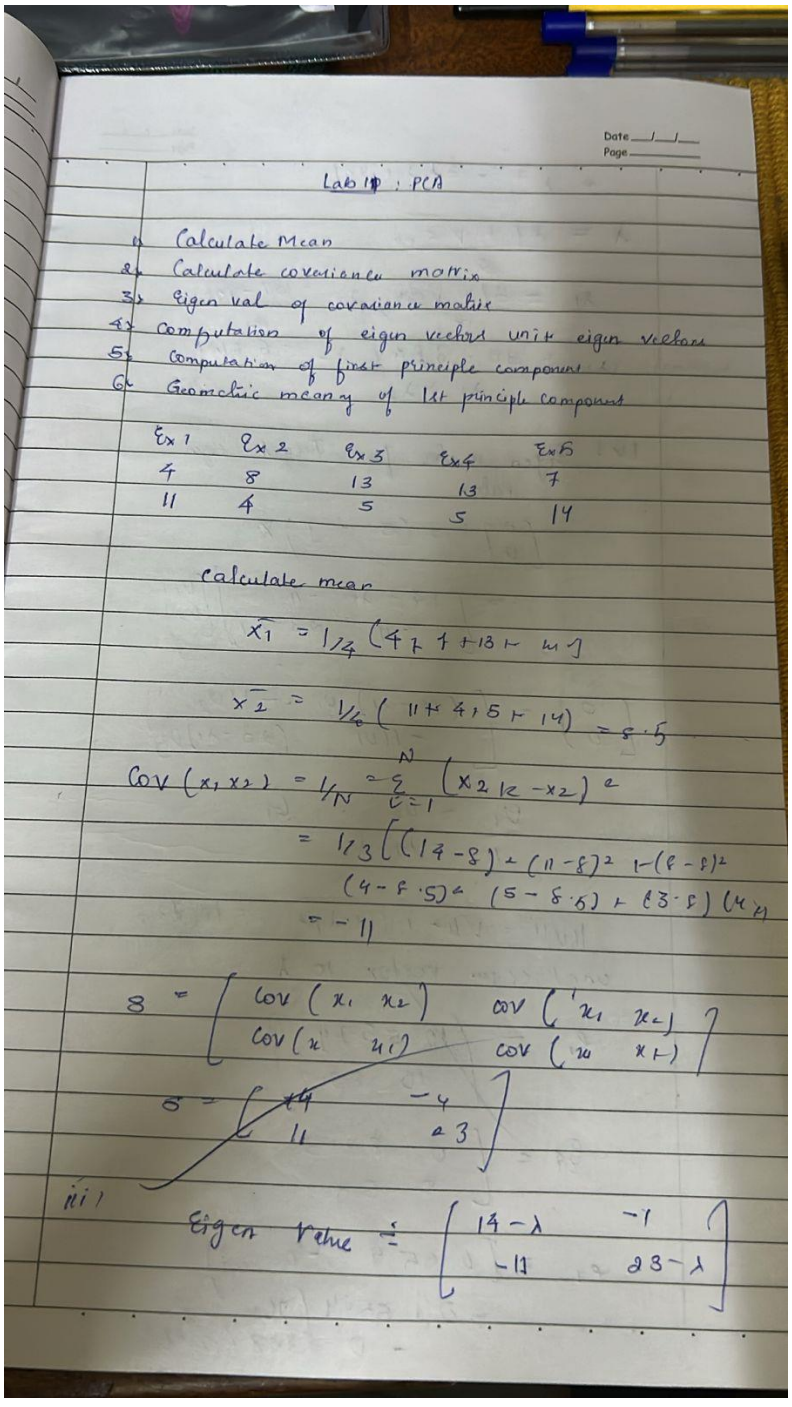
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Step 1: Load dataset from CSV
df = pd.read_csv('your_dataset.csv')  # Replace with your file path

# Optional: View first few rows
print("Data Preview:\n", df.head())
```

```
# Step 2: Select relevant numeric columns for clustering
# You can specify specific columns like: df[['column1', 'column2']]
X = df.select_dtypes(include='number')

# Step 3: Scale the data (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: Apply K-Means clustering (e.g., 3 clusters)
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Step 5: Print cluster centers
print("Cluster Centers:\n", kmeans.cluster_centers_)

# Optional Step 6: Visualize (works well for 2D or PCA-reduced data)
if X.shape[1] >= 2:
    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis')
    plt.title("K-Means Clustering")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show(
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshots



Date / /
Page

Lab 10 : PCA

1. Calculate Mean
2. Calculate covariance matrix
3. Eigen val of covariance matrix
4. Computation of eigen vectors unit eigen vectors
5. Computation of first principle component
6. Geometric meaning of 1st principle component

Ex 1	Ex 2	Ex 3	Ex 4	Ex 5
4	8	13	13	7
11	4	5	5	14

calculate mean

$\bar{x_1} = 1/4 (4 + 7 + 13 + 4)$

$\bar{x_2} = 1/6 (11 + 4 + 5 + 14) = 8.5$

$$Cov(x_1, x_2) = \frac{1}{N} \sum_{i=1}^{N} (x_{2i} - \bar{x_2})^2$$

$= 1/3 [(14 - 8) + (11 - 8)^2 + (8 - 8)^2 (4 - 8.5)^2 + (5 - 8.5) + (3 - 8) (4 - 1)$

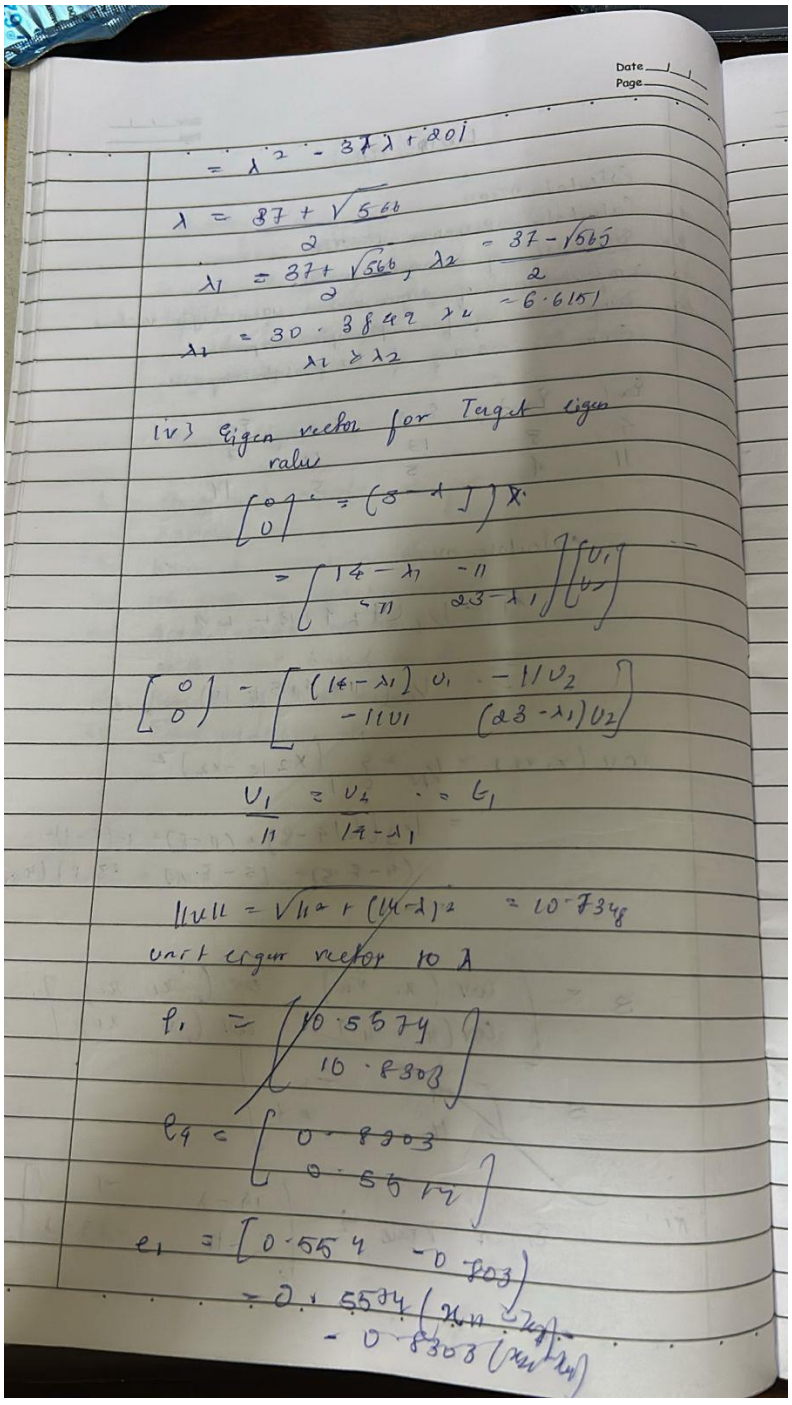
$= -11$

$$S = \begin{bmatrix} Cov(x_1, x_2) & Cov(x_1, x_2) \\ Cov(x, x_i) & Cov(x, x_1) \end{bmatrix}$$

$$S = \begin{bmatrix} 14 & -4 \\ 11 & 23 \end{bmatrix}$$

iii)

Eigen value $= \begin{bmatrix} 14 - \lambda & -1 \\ -11 & 23 - \lambda \end{bmatrix}$



Code

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

```
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt


# Step 1: Load the Iris dataset

iris = load_iris()

X = iris.data
```

```
y = iris.target
feature_names = iris.feature_names

# Step 2: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA (reduce to 2 components for visualization)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 4: Plot the 2D PCA result
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=60)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA - Iris Dataset")
plt.legend(handles=scatter.legend_elements()[0], labels=iris.target_names)
plt.grid(True)
plt.show()

# Explained variance
print("Explained variance ratio:", pca.explained_variance_ratio_)
```