# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

## LAB REPORT

### on

Machine Learning (23CS6PCMAL)

#### Submitted by

**Akash K S (1BM22CS028)**

#### 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 **Akash K S (1BM22CS028),** 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.

Dr. Kavitha Sooda Professor & HOD

Department of CSE, BMSCE

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE

# Index

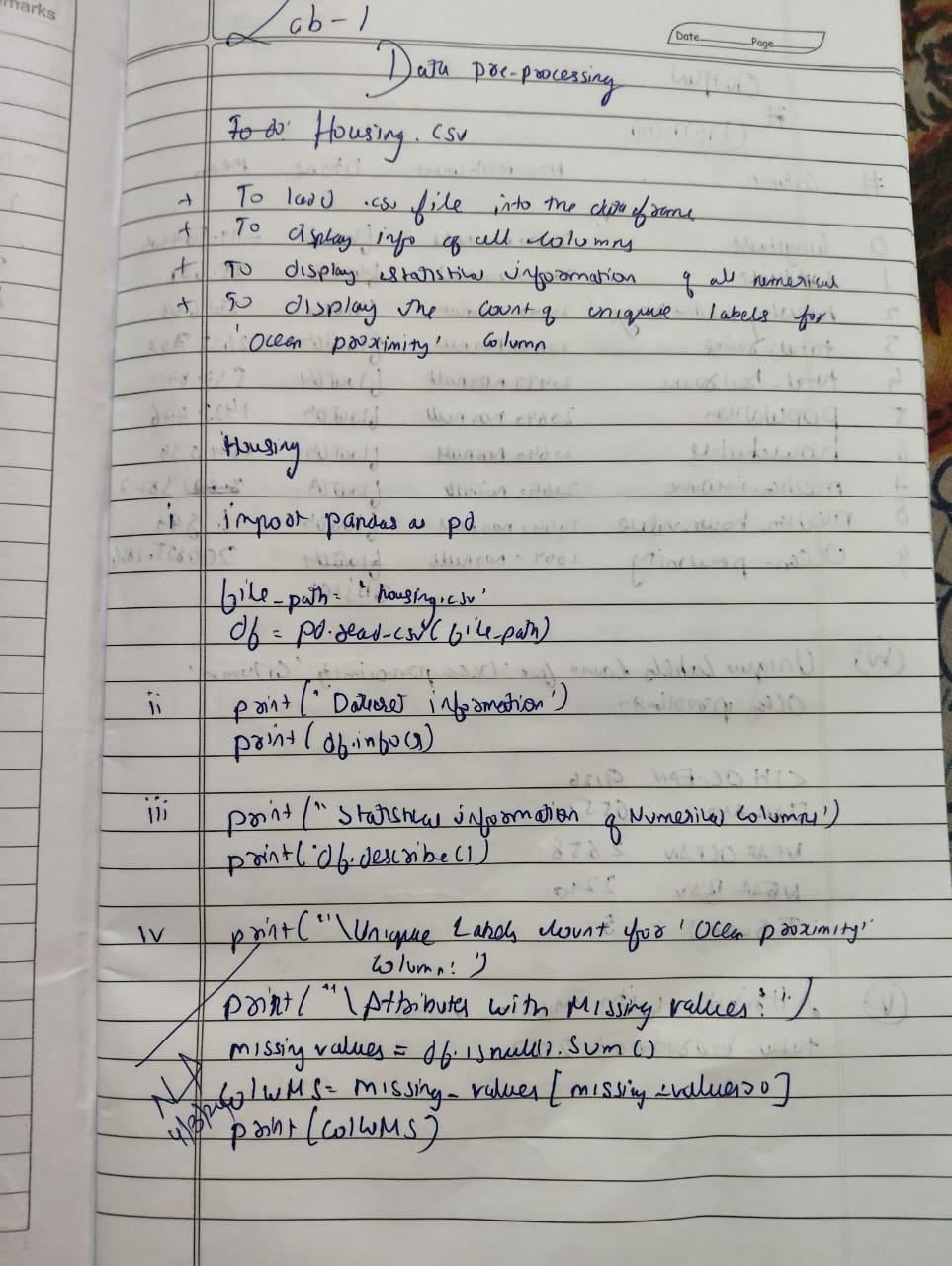
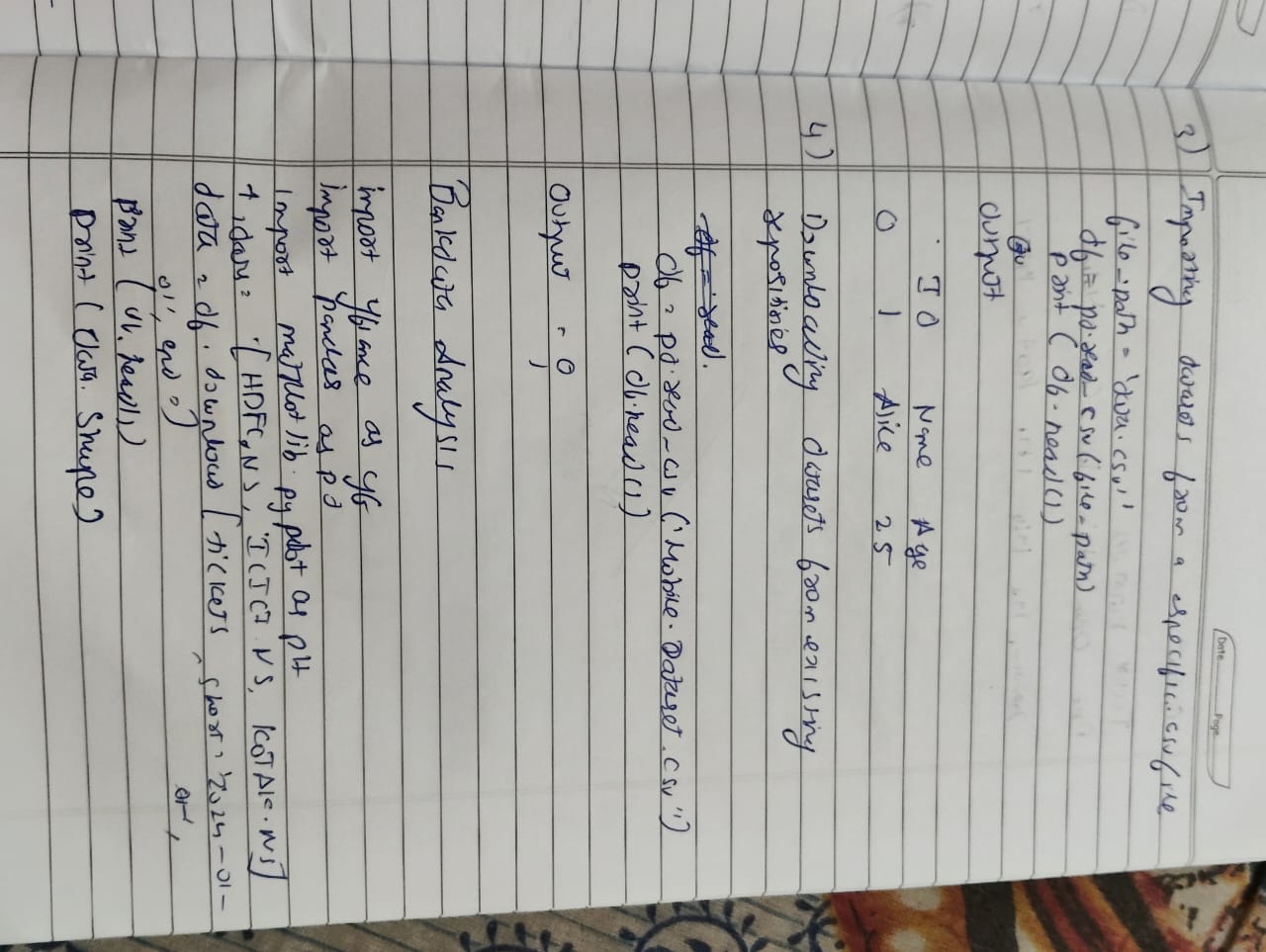
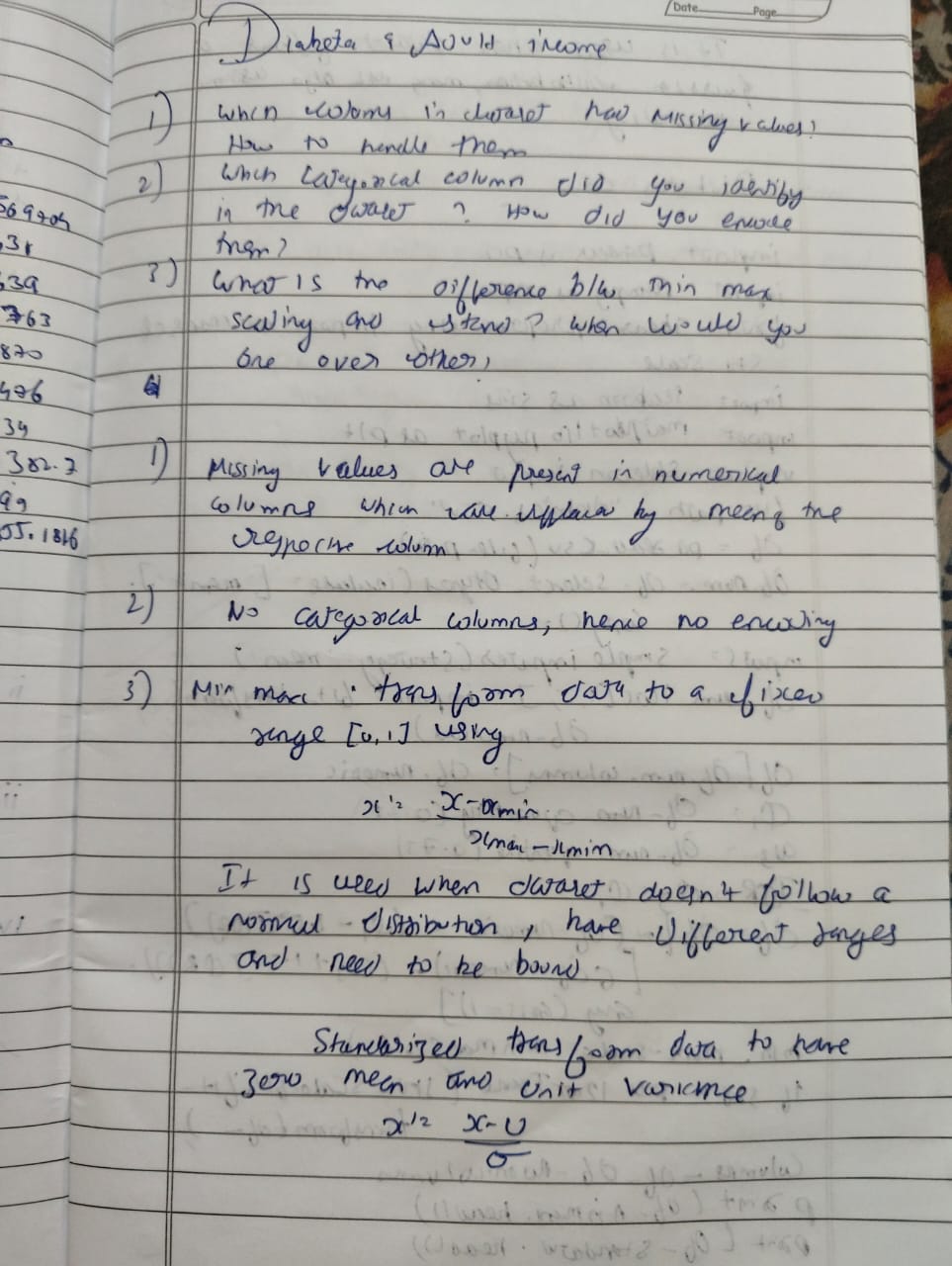
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 21-2-2025 | Write a python program to import and export data using Pandas library functions | 4 |
| 2 | 3-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 8 |
| 3 | 10-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 15 |
| 4 | 17-3-2025 | Build Logistic Regression Model for a given dataset | 25 |
| 5 | 24-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 28 |
| 6 | 7-4-2025 | Build KNN Classification model for a given dataset. | 34 |
| 7 | 21-4-2025 | Build Support vector machine model for a given dataset | 47 |
| 8 | 5-5-2025 | Implement Random forest ensemble method on a given dataset. | 55 |
| 9 | 5-5-2025 | Implement Boosting ensemble method on a given dataset. | 59 |
| 10 | 12-5-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 64 |
| 11 | 12-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 72 |

Github Link: https://github.com/AKASHKS06/ML\_Lab\_Sem6

Program 1

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

Screenshot

Code:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from scipy import stats

#\*\*Diabetes Dataset\*\*

df=pd.read\_csv('/content/Dataset of Diabetes .csv')

df.head()

df.shape

print(df.info())

# Summary statistics

print(df.describe())

missing\_values=df.isnull().sum()

print(missing\_values[missing\_values > 0])

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.")

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())

#\*\*Adult Income Dataset\*\*

df1=pd.read\_csv('/content/adult.csv')

df1.head()

df1.shape

print(df1.info())

# Summary statistics

print(df.describe())

missing\_values=df1.isnull().sum()

print(missing\_values[missing\_values > 0])

categorical\_cols = df1.select\_dtypes(include=['object']).columns

print("Categorical columns identified:", categorical\_cols)

if len(categorical\_cols) > 0:

df1 = pd.get\_dummies(df1, columns=categorical\_cols, drop\_first=True)

print("\nDataFrame after one-hot encoding:")

print(df.head())

else:

print("\nNo categorical columns found in the dataset.")

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import pandas as pd

numerical\_cols = df1.select\_dtypes(include=['number']).columns

scaler = MinMaxScaler()

df\_minmax = df1.copy() # Create a copy to avoid modifying the original

df\_minmax[numerical\_cols] = scaler.fit\_transform(df1[numerical\_cols])

scaler = StandardScaler()

df\_standard = df1.copy()

df\_standard[numerical\_cols] = scaler.fit\_transform(df1[numerical\_cols])

print("\nDataFrame after Min-Max Scaling:")

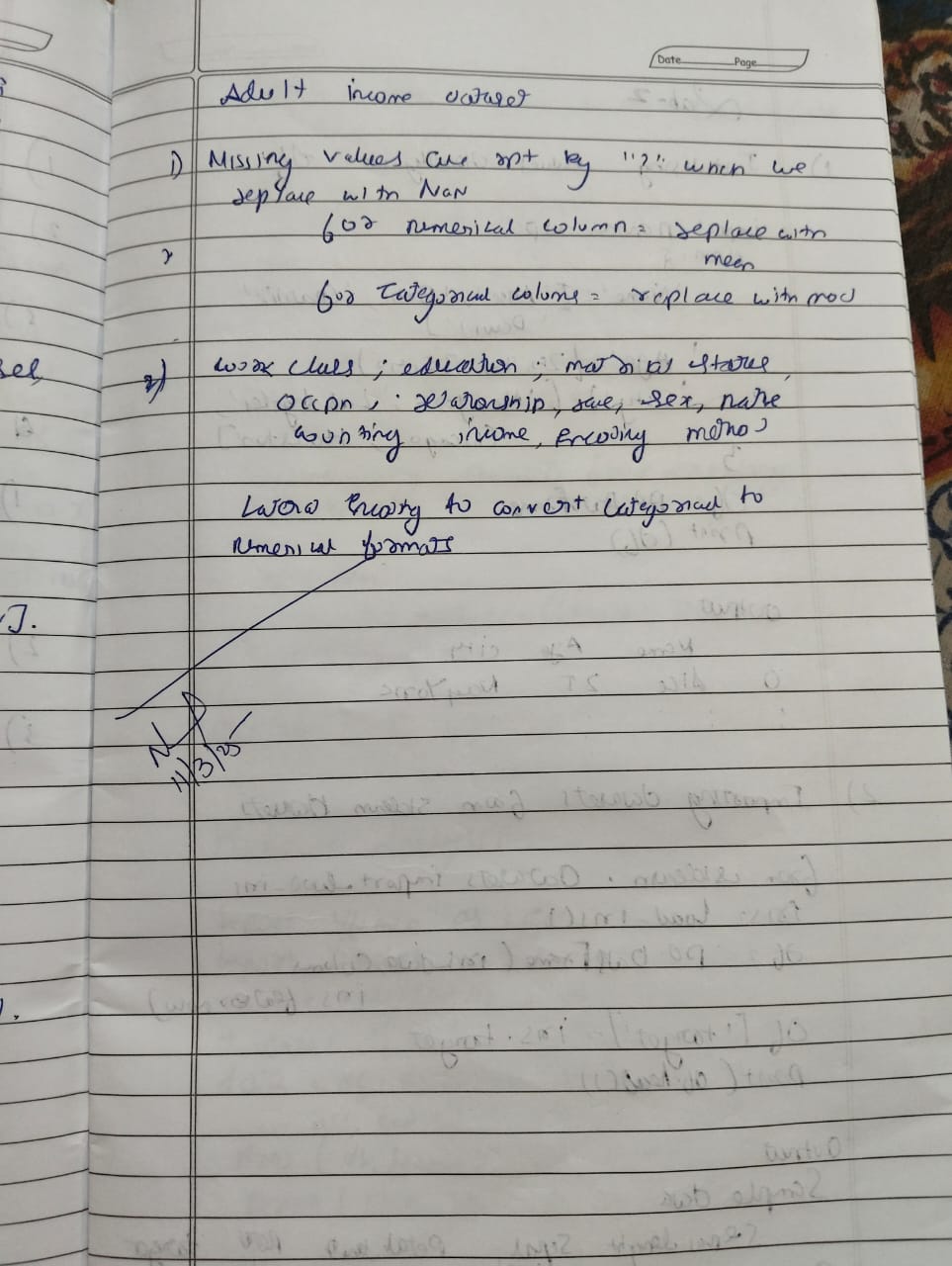
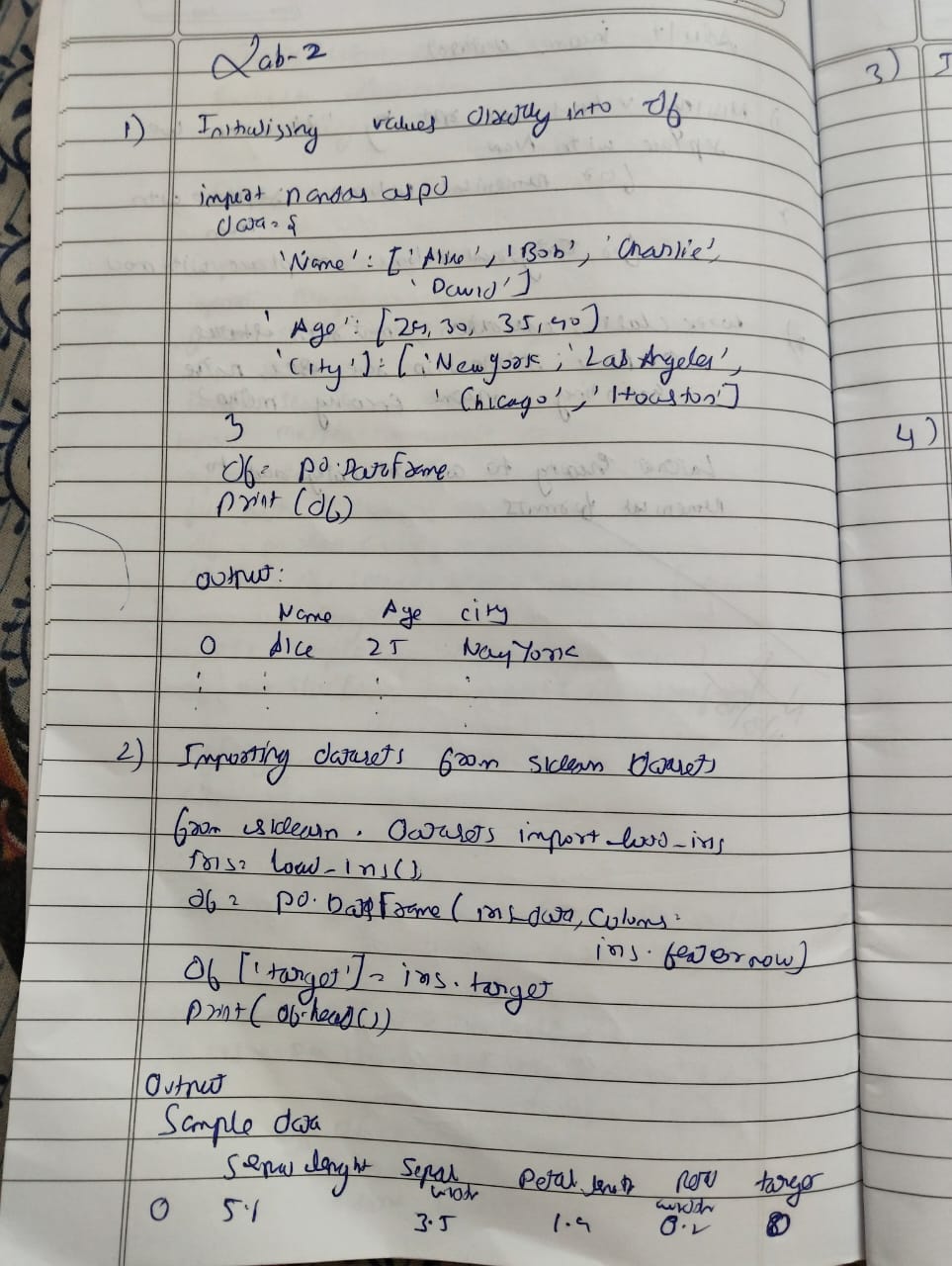
print(df\_minmax.head())

print("\nDataFrame after Standardization:")

print(df\_standard.head())

PROGRAM 2 Demonstrate various data pre-processing techniques for a given dataset

Screenshot



Code

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

df=pd.read\_csv('housing.csv')

df.head(2)

df.describe()

df.info()

sns.histplot(df['median\_income'], kde=True, color='green')

sns.histplot(df['housing\_median\_age'])

from sklearn.model\_selection import train\_test\_split

X = df.drop("median\_house\_value", axis=1)

y = df["median\_house\_value"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,random\_state=42)

X = df.drop("median\_house\_value", axis=1)

y = df["median\_house\_value"]

df["income\_cat"] = pd.cut(df["median\_house\_value"],

bins=[0, 100000, 200000, 300000, 400000, np.inf],

labels=[1, 2, 3, 4, 5])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42,

stratify=df["income\_cat"])

train\_set = X\_train.copy()

train\_set["median\_house\_value"] = y\_train

train\_set.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,s=train\_set["population"]/100, label="population",figsize=(10,7), c="median\_house\_value", cmap=plt.get\_cmap("jet"),

colorbar=True)

plt.legend()

numerical\_columns = df.select\_dtypes(include=['float64', 'int64'])

correlation\_matrix = numerical\_columns.corr()

print(correlation\_matrix["median\_house\_value"].sort\_values(ascending=False))

df.plot(kind="scatter", x="median\_income", y="median\_house\_value", alpha=0.1)

# Combine 'median\_income' and 'households'

df["income\_households"] = df["median\_income"] \* df["households"]

numerical\_columns = df.select\_dtypes(include=['float64', 'int64'])

correlation\_matrix = numerical\_columns.corr()

print(correlation\_matrix["median\_house\_value"].sort\_values(ascending=False))

df.plot(kind="scatter", x="income\_households", y="median\_house\_value", alpha=0.1)

plt.show()

missing\_values = df.isnull().sum()

print(missing\_values[missing\_values > 0])

h=df

h.dropna(subset=["total\_bedrooms"])

from sklearn.preprocessing import OneHotEncoder

df1=pd.read\_csv('housing.csv')

hc=df1[["ocean\_proximity"]]

encoder=OneHotEncoder()

hc\_encoded=encoder.fit\_transform(hc).toarray()

hc\_1hot\_df = pd.DataFrame(hc\_encoded, columns=encoder.get\_feature\_names\_out(hc.columns))

hc\_1hot\_df.head()

Feature scaling is crucial in machine learning for several reasons, particularly when using algorithms that are sensitive to the scale of features. Here's a breakdown of its importance:

1. \*\*Improved Performance of Distance-Based Algorithms:\*\*

2. \*\*Faster Convergence of Gradient Descent:\*\*

3. \*\*Improved Regularization:\*\*

4. \*\*Better Interpretation of Coefficients:\*\*

5. \*\*Numerical Stability:\*\*

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 = df1.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),

])

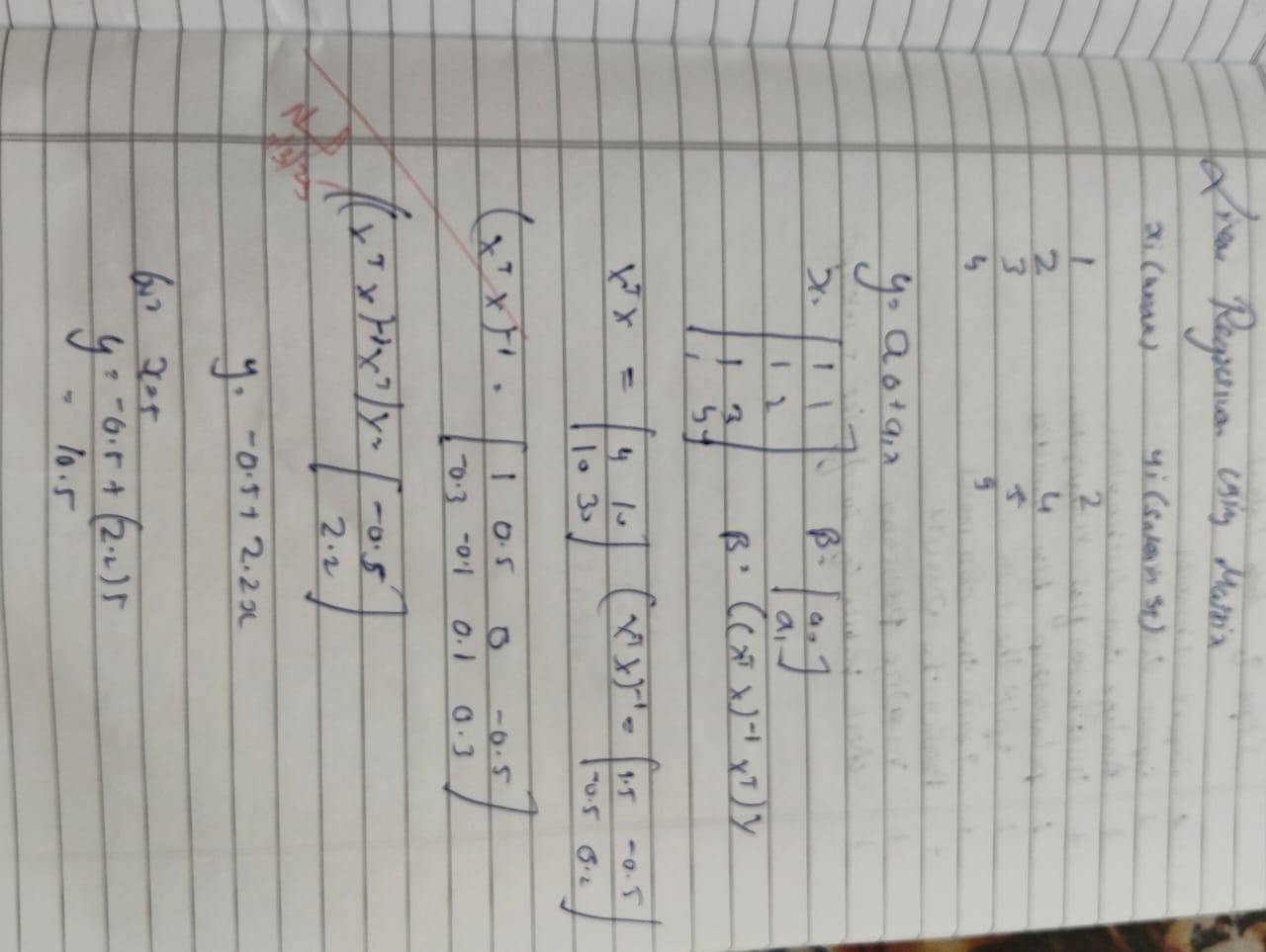
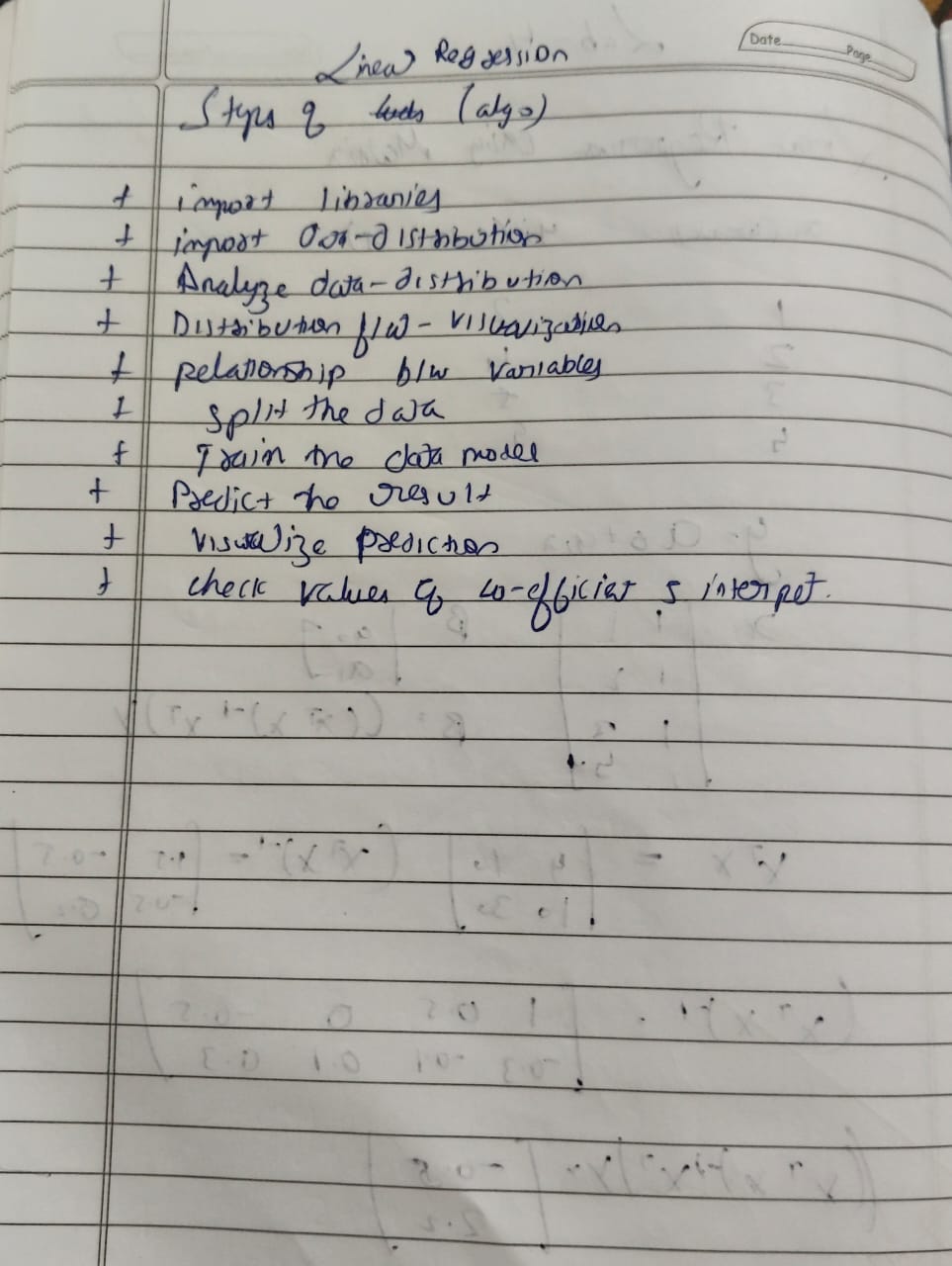
# Process the dataset using the pipeline

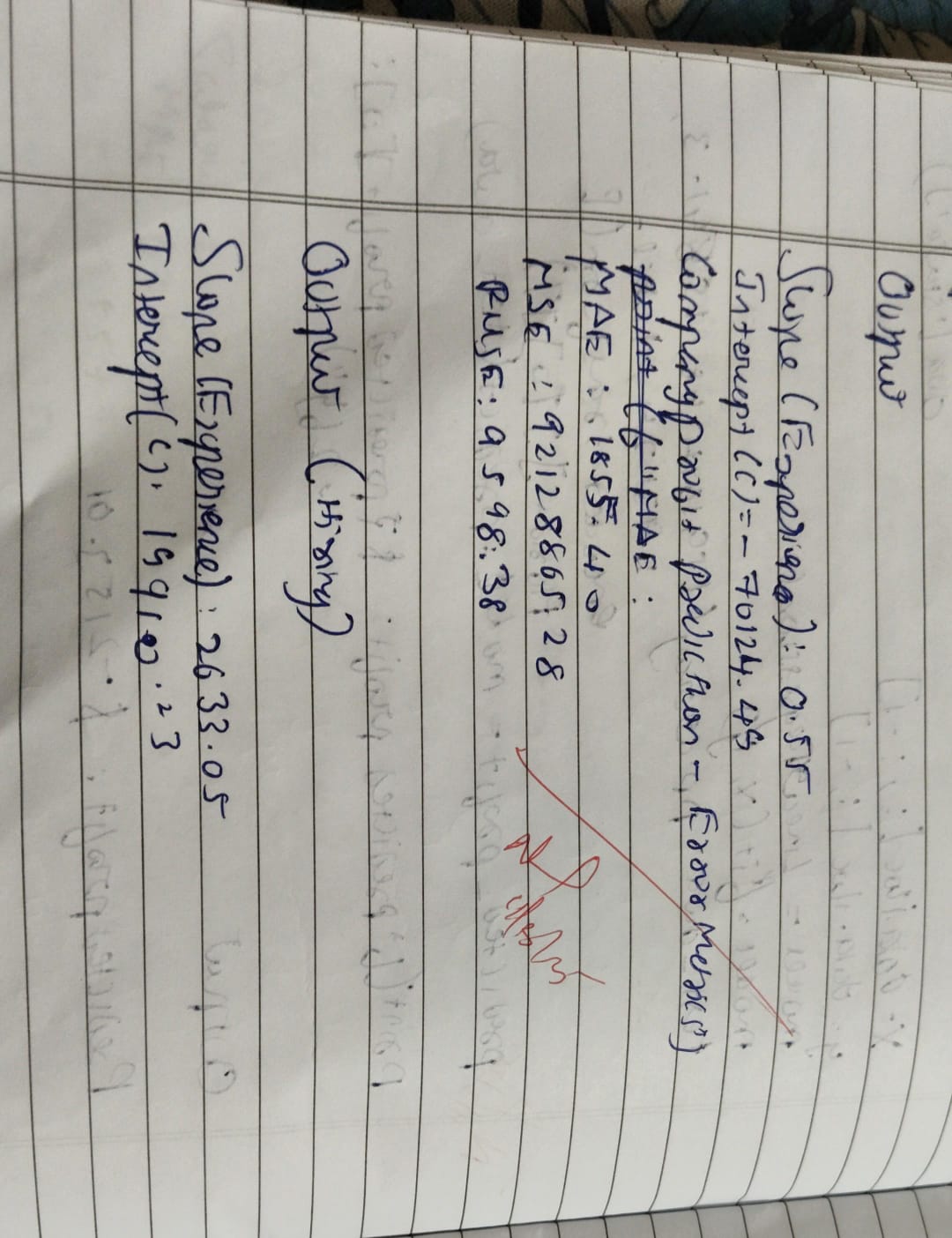
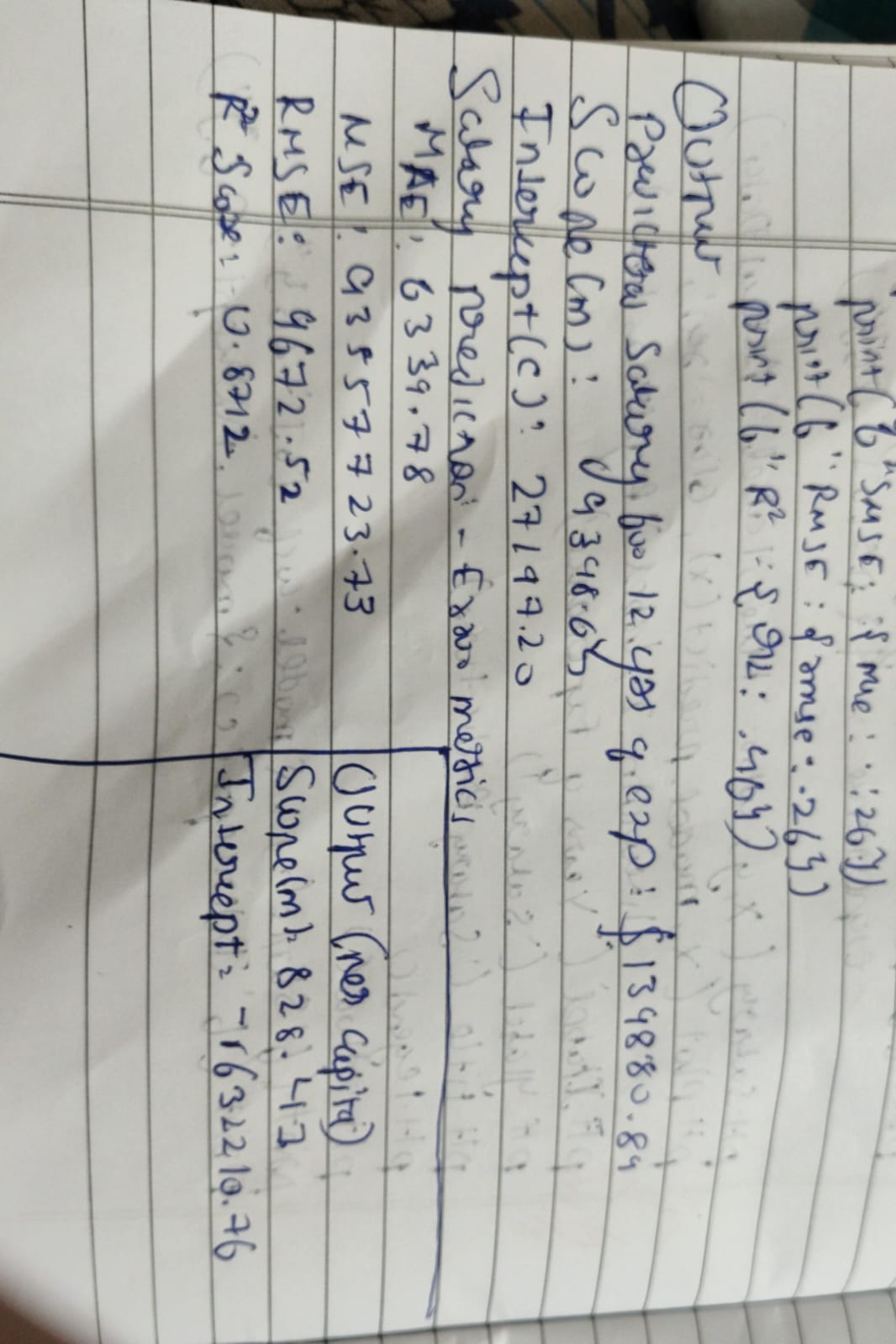
housing\_prepared = full\_pipeline.fit\_transform(housing)

print("Shape of processed data:", housing\_prepared.shape)

PROGRAM 3 Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

Code

# -\*- coding: utf-8 -\*-

import pandas as pd

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

df = pd.read\_csv('/content/housing\_area\_price.csv')

df

# Commented out IPython magic to ensure Python compatibility.

# %matplotlib inline

plt.xlabel('area')

plt.ylabel('price')

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

new\_df = df.drop('price',axis='columns')

new\_df

price = df.price

price

# Create linear regression object

reg = linear\_model.LinearRegression()

reg.fit(new\_df,price)

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

reg.predict([[3300]])

reg.coef\_

reg.intercept\_

"""Y = m \* X + b (m is coefficient and b is intercept)"""

3300\*135.78767123 + 180616.43835616432

"""(1) Predict price of a home with area = 5000 sqr ft"""

reg.predict([[5000]])

# -\*- coding: utf-8 -\*-

import pandas as pd

import numpy as np

from sklearn import linear\_model

df = pd.read\_csv('/content/homeprices\_Multiple\_LR.csv')

df

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

df.bedrooms.median()

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

df

reg = linear\_model.LinearRegression()

reg.fit(df.drop('price',axis='columns'),df.price)

reg.coef\_

reg.intercept\_

"""Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old"""

reg.predict([[3000, 3, 40]])

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

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Load the dataset

df1 = pd.read\_csv('/content/canada\_per\_capita\_income.csv')

# Prepare the data

X = df1.year.values.reshape(-1, 1) # Features (year)

y = df1['per capita income (US$)'] # Target (per capita income)

# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)

# Predict per capita income for 2020

year\_2020 = [[2020]]

predicted\_income = model.predict(year\_2020)

print(f"Predicted per capita income for Canada in 2020: {predicted\_income[0]:.2f}")

import pandas as pd

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Load the dataset (canada\_per\_capita\_income.csv)

df1 = pd.read\_csv('/content/canada\_per\_capita\_income.csv')

# Prepare the data

X = df1.year.values.reshape(-1, 1) # Features (year)

y = df1['per capita income (US$)'] # Target (per capita income)

# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)

# Create the plot

plt.figure(figsize=(8, 6))

plt.scatter(X, y, color='blue', label='Data Points') # Now using the correct X and y

plt.plot(X, model.predict(X), color='red', label='Regression Line')

plt.xlabel('Year')

plt.ylabel('Per Capita Income (US$)')

plt.title('Per Capita Income in Canada over Time')

plt.legend()

plt.grid(True)

plt.show()

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.impute import SimpleImputer

# Load the dataset

df = pd.read\_csv('/content/salary.csv')

# Prepare the data

X = df.iloc[:, :-1].values # Features (years of experience)

y = df.iloc[:, 1].values # Target (salary)

# Impute missing values with the mean

imputer = SimpleImputer(strategy='mean') # Create an imputer object with strategy as mean

X = imputer.fit\_transform(X) # Fit and transform the imputer on feature data 'X'

# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)

# Predict salary for 12 years of experience

years\_experience = [[12]]

predicted\_salary = model.predict(years\_experience)

print(f"Predicted salary for 12 years of experience: {predicted\_salary[0]:.2f}")

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.impute import SimpleImputer

# Load the dataset

df = pd.read\_csv('/content/hiring.csv')

# Handle missing values

# Convert 'experience' column to numeric, replacing non-numeric with NaN

df['experience'] = pd.to\_numeric(df['experience'], errors='coerce')

imputer = SimpleImputer(strategy='mean')

df['experience'] = imputer.fit\_transform(df[['experience']])

df['test\_score(out of 10)'] = imputer.fit\_transform(df[['test\_score(out of 10)']])

# Prepare the data

X = df.drop('salary($)', axis='columns')

y = df['salary($)']

# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)

# Predict salaries for the given candidates

candidate1 = [[2, 9, 6]]

candidate2 = [[12, 10, 10]]

predicted\_salary1 = model.predict(candidate1)

predicted\_salary2 = model.predict(candidate2)

print(f"Predicted salary for candidate 1: ${predicted\_salary1[0]:.2f}")

print(f"Predicted salary for candidate 2: ${predicted\_salary2[0]:.2f}")

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.compose import ColumnTransformer

# Load the dataset

df = pd.read\_csv('/content/1000\_Companies.csv')

# Separate features (X) and target (y)

X = df.iloc[:, :-1].values

y = df.iloc[:, 4].values

# Encode categorical data (State)

labelencoder = LabelEncoder()

X[:, 3] = labelencoder.fit\_transform(X[:, 3])

ct = ColumnTransformer(

transformers=[('encoder', OneHotEncoder(), [3])],

remainder='passthrough'

)

X = ct.fit\_transform(X)

# Avoid dummy variable trap (remove one encoded column)

X = X[:, 1:]

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Create and train the multiple linear regression model

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

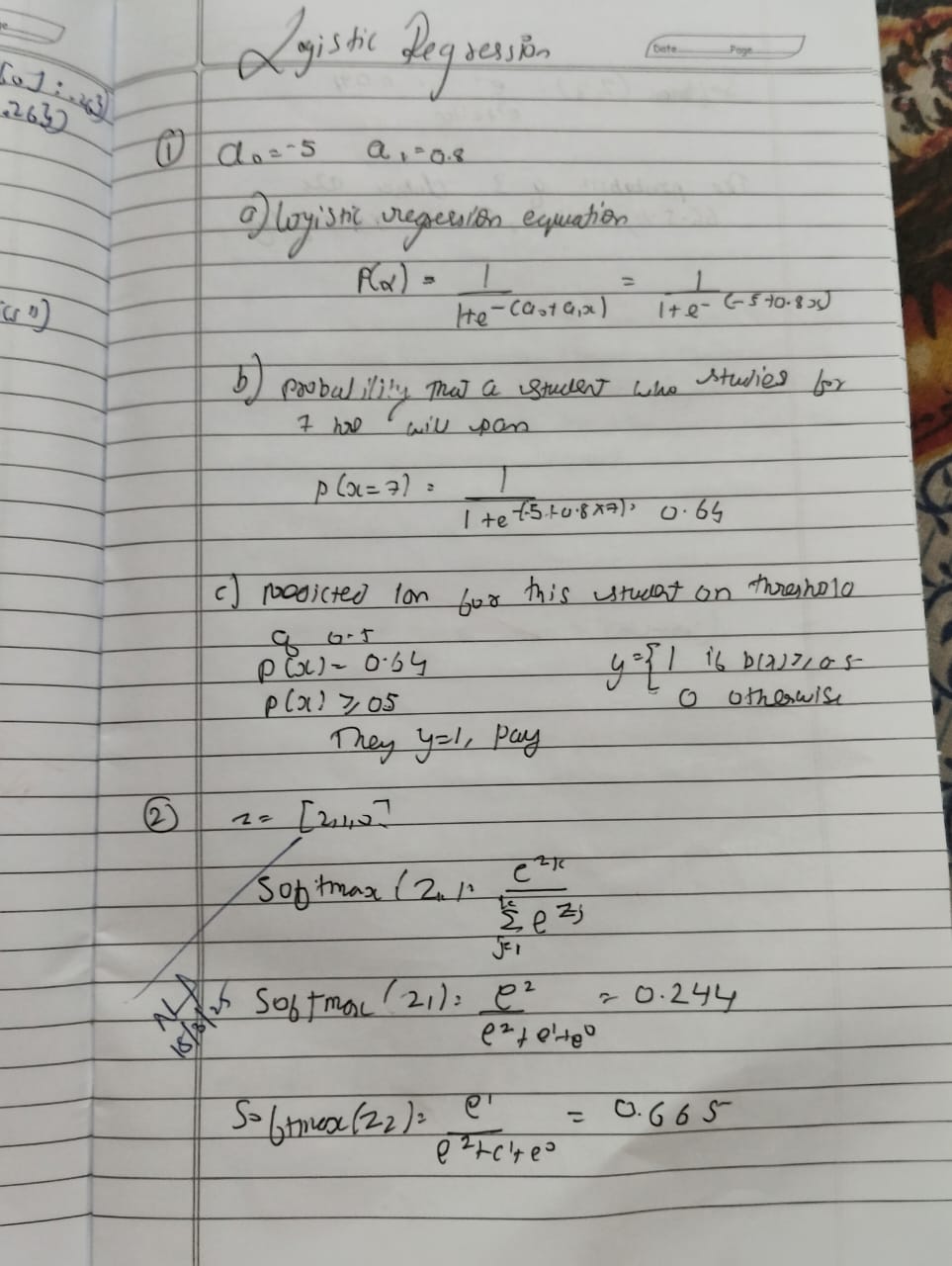
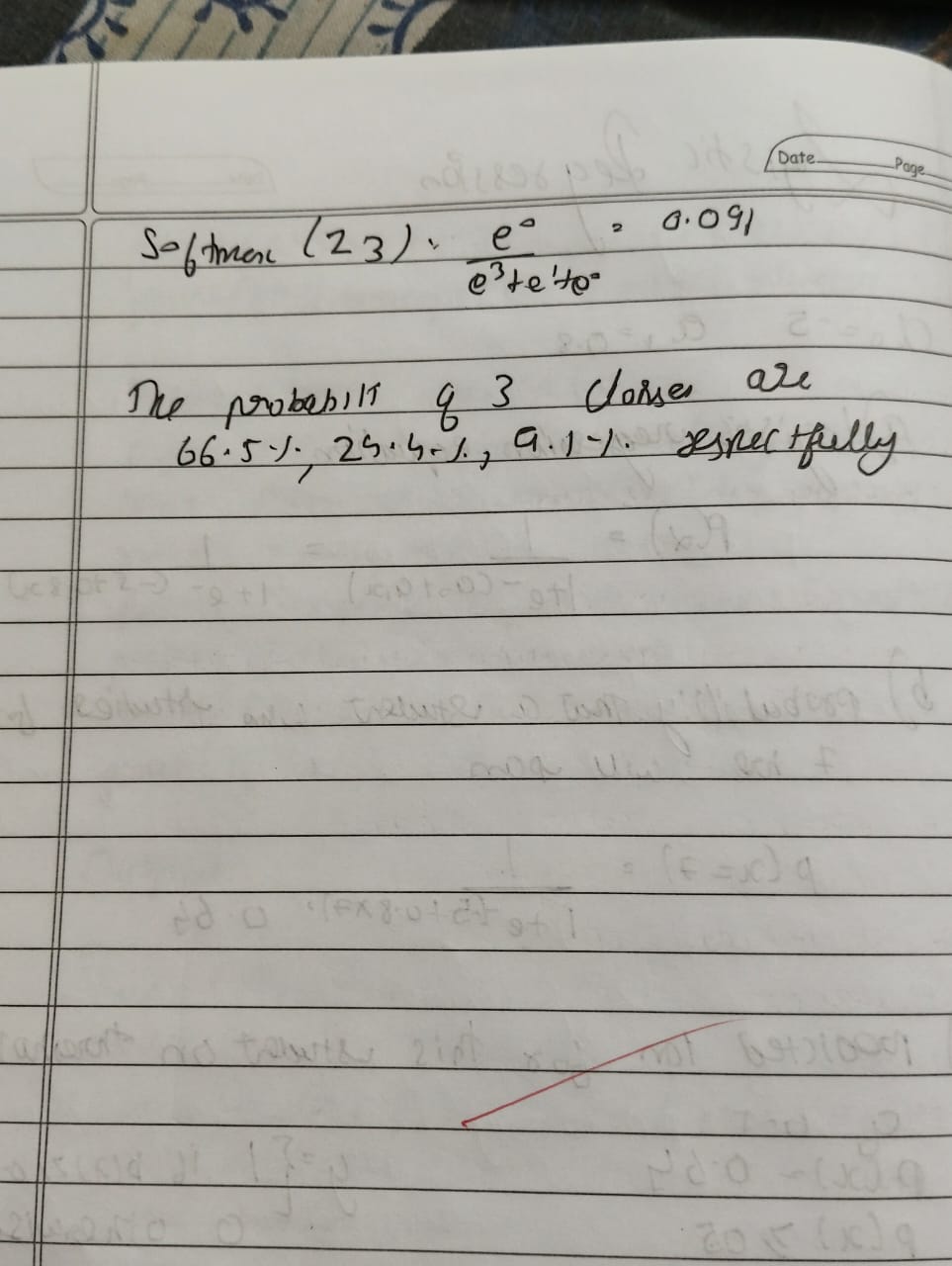
# Predict profit for the given values

new\_prediction = regressor.predict([[1, 0, 91694.48, 515841.3, 11931.24]])

print(f"Predicted Profit: {new\_prediction[0]:.2f}")

PROGRAM 4 Build Logistic Regression Model for a given dataset

Screenshot

Code

import pandas as pd

import numpy as np

df=pd.read\_csv("/content/HR\_comma\_sep.csv")

df.head(3)

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

print(df.groupby('left').mean(numeric\_only=True))

print(df.groupby('salary').mean(numeric\_only=True))

import matplotlib.pyplot as plt

pd.crosstab(df.salary,df.left).plot(kind='bar')

plt.title('Employee Retention vs Salary')

plt.xlabel('Salary')

plt.ylabel('Number of Employees')

plt.show()

pd.crosstab(df.Department,df.left).plot(kind='bar')

plt.title('Employee Retention vs Department')

plt.xlabel('Department')

plt.ylabel('Number of Employees')

plt.show()

salary\_dummies = pd.get\_dummies(df.salary, prefix="salary")

dept\_dummies = pd.get\_dummies(df.Department, prefix="dept")

df\_with\_dummies = pd.concat([df, salary\_dummies, dept\_dummies], axis=1)

df\_with\_dummies = df\_with\_dummies.drop(['salary', 'Department'], axis=1)

X\_features = ['satisfaction\_level', 'last\_evaluation', 'number\_project', 'average\_montly\_hours', 'time\_spend\_company', 'Work\_accident', 'promotion\_last\_5years'] + list(salary\_dummies.columns) + list(dept\_dummies.columns)

X = df\_with\_dummies[X\_features]

y = df\_with\_dummies.left

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X\_train, y\_train)

from sklearn.metrics import accuracy\_score

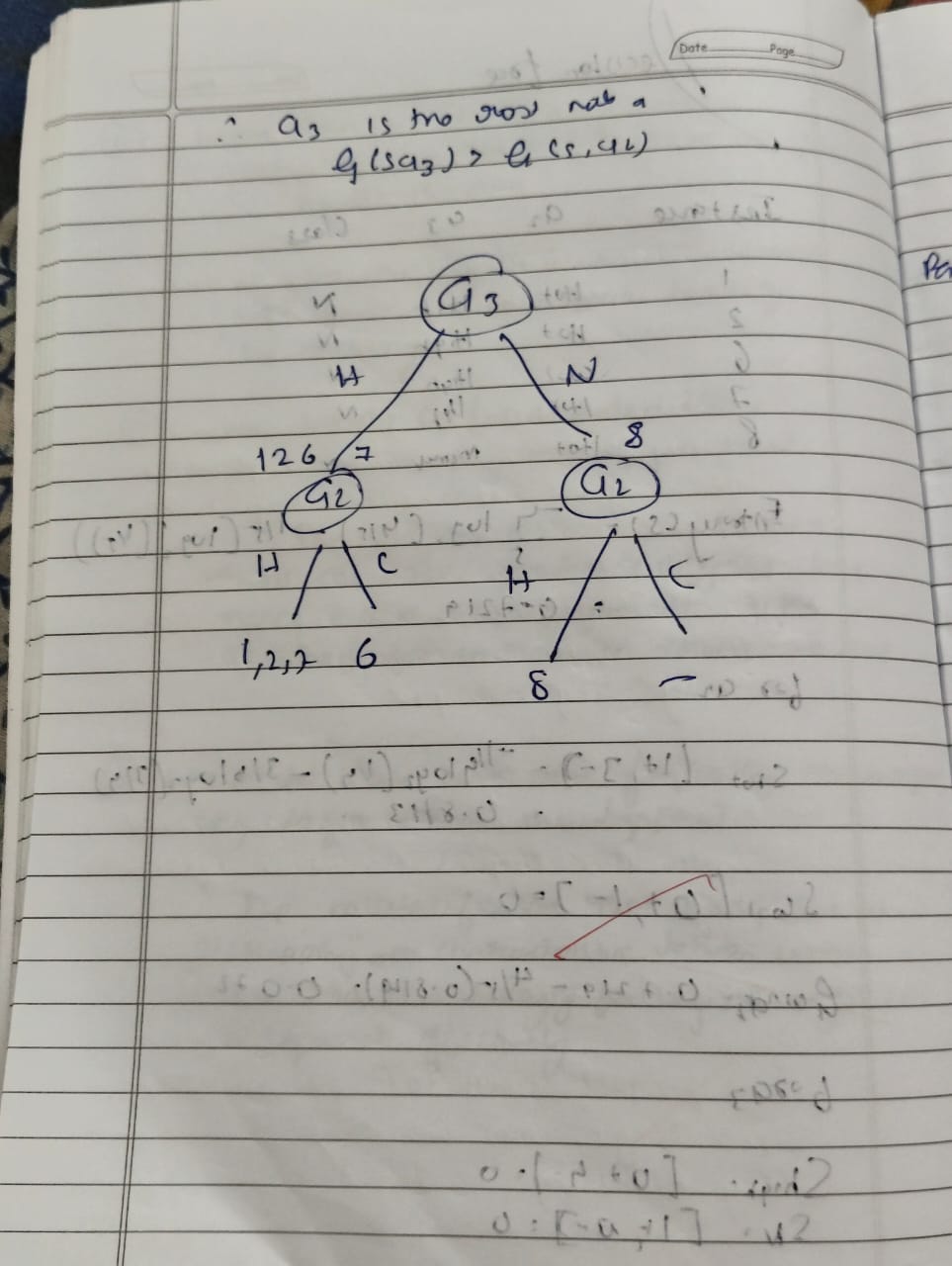
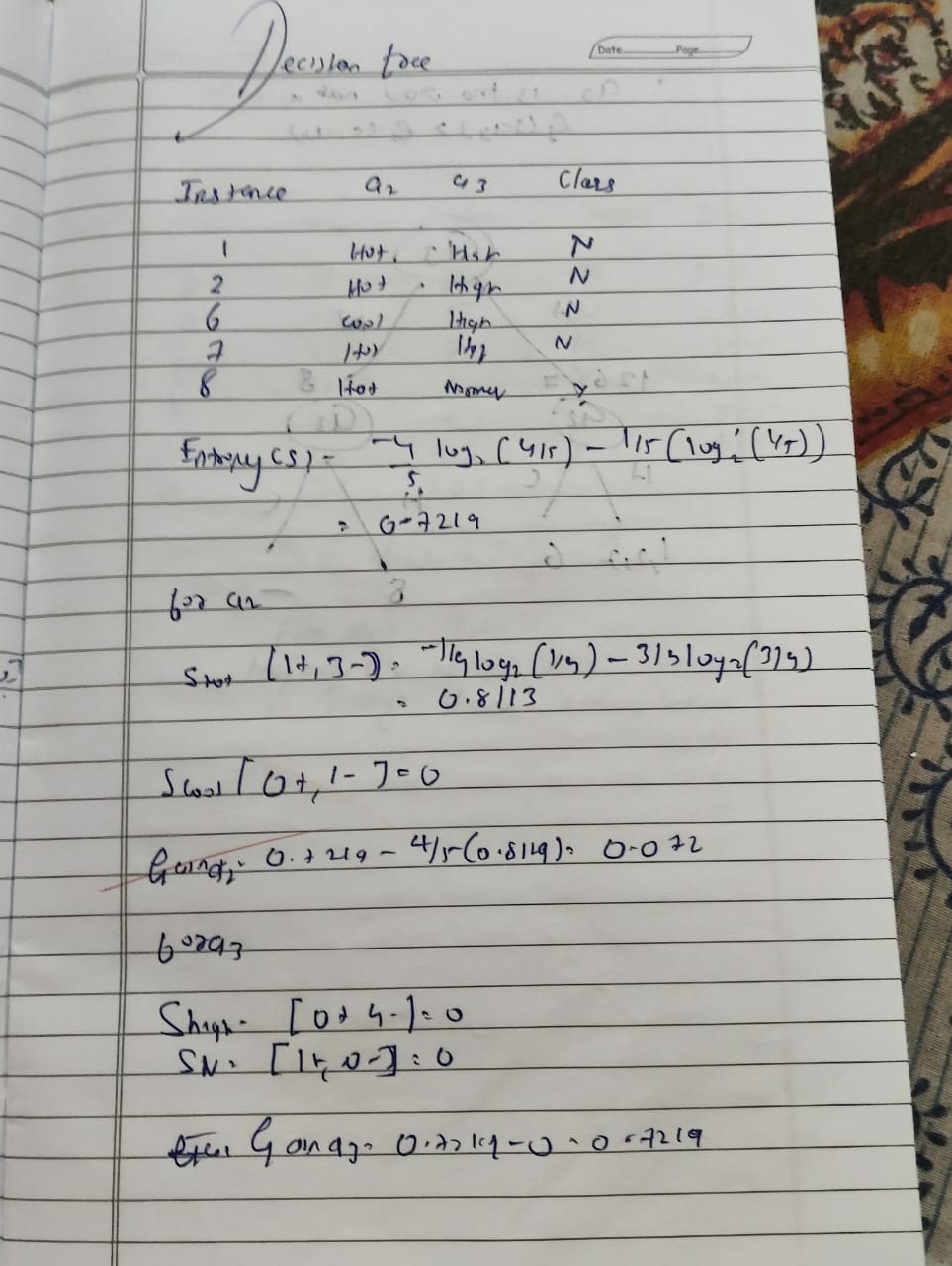
y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy of the model:", accuracy)

PROGRAM 5 Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Screenshot



Code

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn import tree

import matplotlib.pyplot as plt

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf\_matrix)

plt.figure(figsize=(12, 8))

tree.plot\_tree(clf, feature\_names=iris.feature\_names, class\_names=iris.target\_names, filled=True)

plt.show()

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn import tree

import matplotlib.pyplot as plt

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf\_matrix)

plt.figure(figsize=(12, 8))

tree.plot\_tree(clf, feature\_names=iris.feature\_names, class\_names=iris.target\_names, filled=True)

plt.show()

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import numpy as np # import numpy

data = pd.read\_csv("petrol\_consumption.csv")

X = data[['Petrol\_tax', 'Average\_income', 'Paved\_Highways',

'Population\_Driver\_licence(%)']]

y = data['Petrol\_Consumption']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42)

regressor = DecisionTreeRegressor()

regressor.fit(X\_train, y\_train)

y\_pred = regressor.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

print("Root Mean Squared Error:", rmse)

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

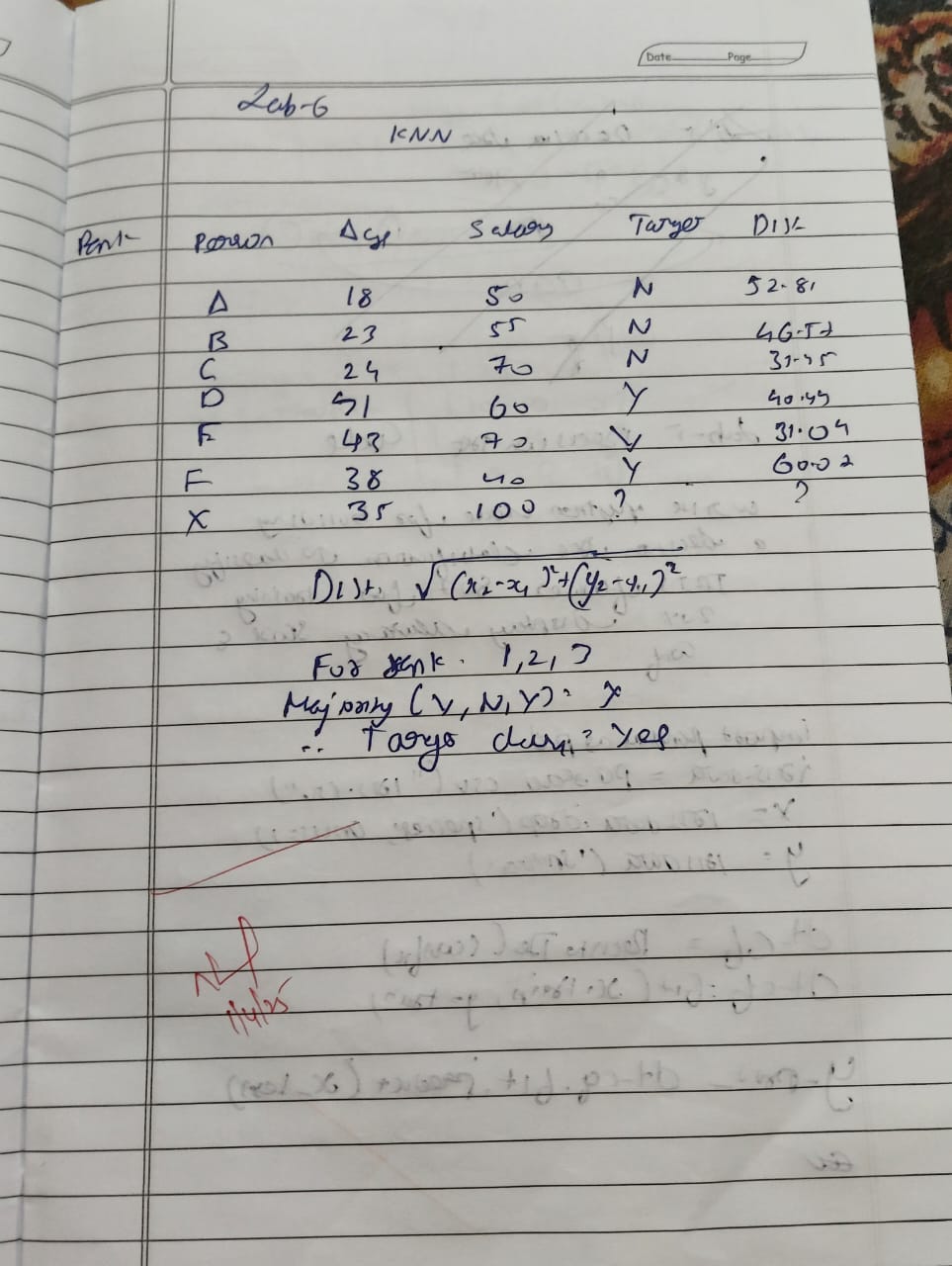
# Assuming 'data' is your original pandas DataFrame

plot\_tree(regressor, feature\_names=data[['Petrol\_tax', 'Average\_income', 'Paved\_Highways', 'Population\_Driver\_licence(%)']].columns, filled=True, rounded=True)

plt.show()

PROGRAM 6 Build KNN Classification model for a given dataset.

Screenshot



Code

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

import seaborn as sns

import matplotlib.pyplot as plt

try:

data = pd.read\_csv('/content/iris (1).csv')

except FileNotFoundError:

print("Error: 'iris.csv' not found. Please upload the file to your Colab environment.")

exit()

X = data.drop('species', axis=1)

y = data['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

print("Accuracy Score:", accuracy\_score(y\_test, y\_pred))

print("\nConfusion Matrix:")

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=knn.classes\_, yticklabels=knn.classes\_)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.preprocessing import StandardScaler

import seaborn as sns

import matplotlib.pyplot as plt

try:

diabetes = pd.read\_csv('diabetes.csv')

except FileNotFoundError:

print("Error: 'diabetes.csv' not found. Please ensure the file is in the current directory.")

exit()

X = diabetes.drop('Outcome', axis=1)

y = diabetes['Outcome']

scaler = StandardScaler()

X = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(cm)

sns.heatmap(cm, annot=True, fmt="d")

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

try:

heart = pd.read\_csv('heart.csv')

except FileNotFoundError:

print("Error: 'heart.csv' not found. Please ensure the file is in the current directory.")

exit()

X = heart.drop('target', axis=1)

y = heart['target']

scaler = StandardScaler()

X = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

best\_k = 1

best\_accuracy = 0

for k in range(1, 21):

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_k = k

print(f"Best k: {best\_k} with accuracy {best\_accuracy}")

knn = KNeighborsClassifier(n\_neighbors=best\_k)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(cm)

sns.heatmap(cm, annot=True, fmt="d")

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import classification\_report, confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

print(classification\_report(y\_test, y\_pred))

# prompt: For Iris dataset

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

# rate. Give theory

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

# Load the Iris dataset

try:

data = pd.read\_csv('/content/iris (1).csv')

except FileNotFoundError:

print("Error: 'iris (1).csv' not found. Please upload the file to your Colab environment.")

exit()

# Prepare the data

X = data.drop('species', axis=1)

y = data['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Scale the data (important for KNN)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Find the optimal k value

error\_rates = []

for k in range(1, 31): # Test k values from 1 to 30

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

error\_rates.append(1 - accuracy\_score(y\_test, y\_pred)) # Error rate = 1 - accuracy

# Plot error rates

plt.figure(figsize=(10, 6))

plt.plot(range(1, 31), error\_rates, color='blue', linestyle='dashed', marker='o',

markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

plt.show()

# Theory for choosing k:

# The optimal 'k' value minimizes the error rate.

# Very small k (e.g., 1) can lead to overfitting, being too sensitive to noise.

# Very large k (e.g., 30) can lead to underfitting, smoothing out the decision boundaries too much.

# We seek a k that balances these extremes, as shown by the error rate plot.

#Select k based on the minimum error rate observed in the plot

best\_k = error\_rates.index(min(error\_rates)) + 1 #Add 1 as the index starts from 0

# Train and evaluate the model with the best k

knn = KNeighborsClassifier(n\_neighbors=best\_k)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

# Evaluate the model

print("Accuracy Score:", accuracy\_score(y\_test, y\_pred))

print("\nConfusion Matrix:")

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=knn.classes\_, yticklabels=knn.classes\_)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Load data

df = pd.read\_csv('/content/iris (1).csv')

X = df.iloc[:, :-1]

y = df.iloc[:, -1]

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

# Store accuracy and error rate

accuracy = []

error\_rate = []

# Try k from 1 to 20

for k in range(1, 21):

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

preds = knn.predict(X\_test)

acc = accuracy\_score(y\_test, preds)

accuracy.append(acc)

error\_rate.append(1 - acc)

# Plot

plt.figure(figsize=(10,5))

plt.plot(range(1, 21), accuracy, label='Accuracy')

plt.plot(range(1, 21), error\_rate, label='Error Rate')

plt.xlabel('K Value')

plt.ylabel('Rate')

plt.title('K vs Accuracy and Error Rate')

plt.legend()

plt.show()

import pandas as pd

from sklearn.preprocessing import StandardScaler

# Load data

df = pd.read\_csv('/content/diabetes.csv')

X = df.drop('Outcome', axis=1) # Features

y = df['Outcome'] # Target

# Perform scaling

scaler = StandardScaler()

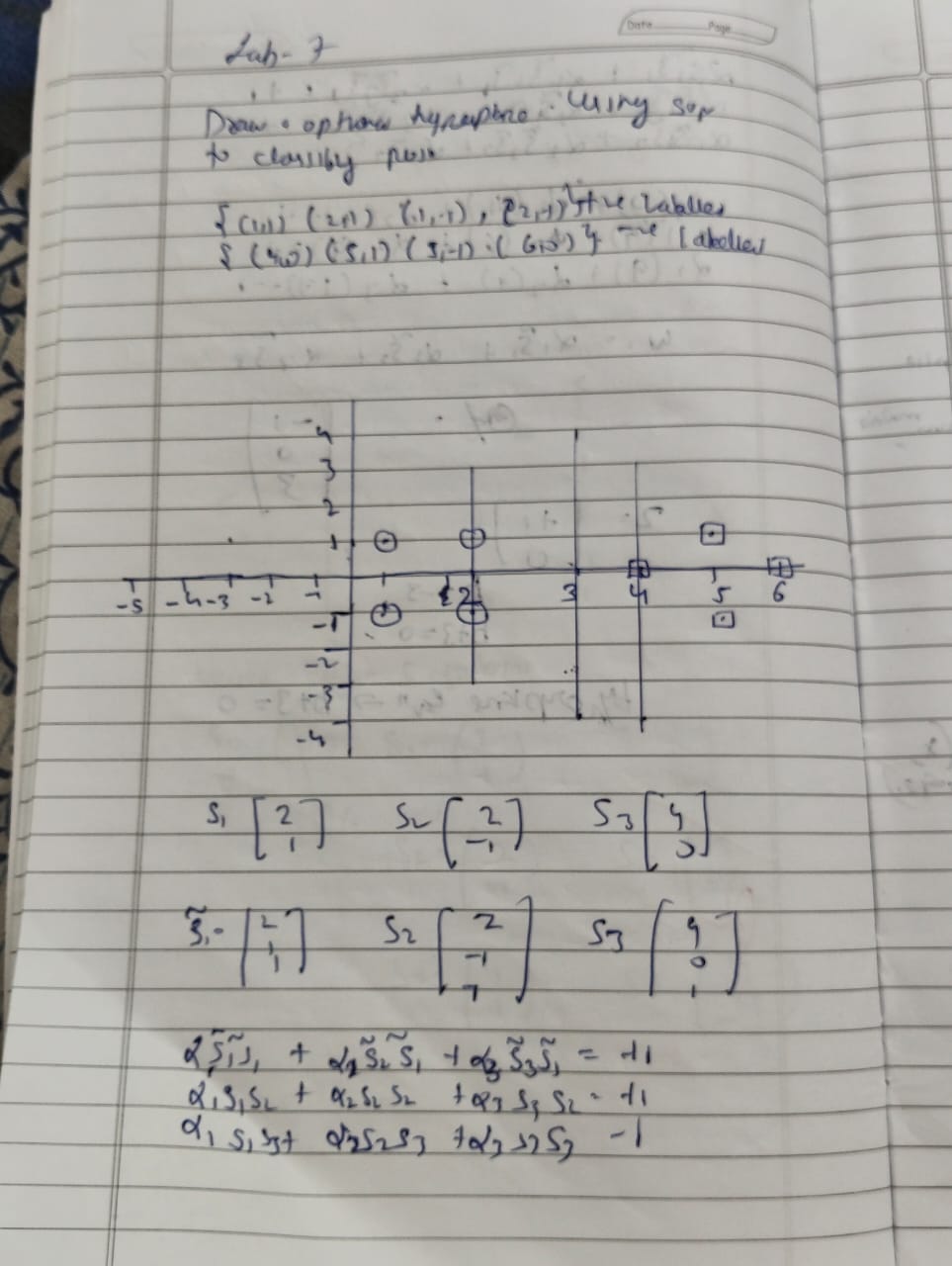
X\_scaled = scaler.fit\_transform(X)

# Convert back to DataFrame (optional)

X\_scaled\_df = pd.DataFrame(X\_scaled, columns=X.columns)

PROGRAM 7 Build Support vector machine model for a given dataset

Screenshot



Code

import numpy as np

import matplotlib.pyplot as plt

positive\_class = np.array([[4, 1], [4, -1], [6, 0]])

negative\_class = np.array([[1, 0], [0, 1], [0, -1]])

plt.figure(figsize=(8, 6))

plt.scatter(positive\_class[:, 0], positive\_class[:, 1], color='red', label='Positive Class', s=100, edgecolors='black')

plt.scatter(negative\_class[:, 0], negative\_class[:, 1], color='blue', label='Negative Class', s=100, edgecolors='black')

all\_points = np.concatenate([positive\_class, negative\_class])

labels = ["(4,1)", "(4,-1)", "(6,0)", "(1,0)", "(0,1)", "(0,-1)"]

for i, txt in enumerate(labels):

plt.annotate(txt, (all\_points[i][0], all\_points[i][1]), textcoords="offset points", xytext=(0,5), ha='center', fontsize=10)

x\_values = np.linspace(-1, 7, 100)

y\_values = np.zeros\_like(x\_values)

plt.plot(x\_values, y\_values, color='black', linestyle='--', label='Optimal Hyperplane (y = 0)')

plt.plot(x\_values, y\_values + 1, color='gray', linestyle=':', label='Margin at y = 1')

plt.plot(x\_values, y\_values - 1, color='gray', linestyle=':', label='Margin at y = -1')

plt.title('Optimal Hyperplane for SVM (Visual Approximation)', fontsize=14)

plt.xlabel('x1')

plt.ylabel('x2')

plt.xlim(-1, 7)

plt.ylim(-2, 2)

plt.axhline(0, color='black',linewidth=0.5)

plt.axvline(0, color='black',linewidth=0.5)

plt.legend()

plt.grid(True)

plt.show()

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

data = pd.read\_csv('/content/iris (1) (1).csv')

X = data.drop('species', axis=1)

y = data['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

svm\_rbf = SVC(kernel='rbf')

svm\_rbf.fit(X\_train, y\_train)

y\_pred\_rbf = svm\_rbf.predict(X\_test)

accuracy\_rbf = accuracy\_score(y\_test, y\_pred\_rbf)

cm\_rbf = confusion\_matrix(y\_test, y\_pred\_rbf)

print("SVM with RBF Kernel:")

print("Accuracy:", accuracy\_rbf)

print("Confusion Matrix:\n", cm\_rbf)

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_rbf, annot=True, fmt='d', cmap='Blues',

xticklabels=data['species'].unique(),

yticklabels=data['species'].unique())

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix (RBF Kernel)')

plt.show()

svm\_linear = SVC(kernel='linear')

svm\_linear.fit(X\_train, y\_train)

y\_pred\_linear = svm\_linear.predict(X\_test)

accuracy\_linear = accuracy\_score(y\_test, y\_pred\_linear)

cm\_linear = confusion\_matrix(y\_test, y\_pred\_linear)

print("\nSVM with Linear Kernel:")

print("Accuracy:", accuracy\_linear)

print("Confusion Matrix:\n", cm\_linear)

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_linear, annot=True, fmt='d', cmap='Blues',

xticklabels=data['species'].unique(),

yticklabels=data['species'].unique())

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix (Linear Kernel)')

plt.show()

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, auc

import seaborn as sns

from sklearn.preprocessing import label\_binarize

from sklearn.multiclass import OneVsRestClassifier

data = pd.read\_csv('/content/letter-recognition.csv') # Replace with the correct path if necessary

X = data.drop('letter', axis=1)

y = data['letter']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

svm\_classifier = SVC(kernel='rbf', probability=True) # probability=True is needed for ROC curve

svm\_classifier.fit(X\_train, y\_train)

y\_pred = svm\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

cm = confusion\_matrix(y\_test, y\_pred)

print("SVM Classifier:")

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", cm)

plt.figure(figsize=(10, 8))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y), yticklabels=np.unique(y))

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

y\_test\_bin = label\_binarize(y\_test, classes=np.unique(y))

n\_classes = y\_test\_bin.shape[1]

classifier = OneVsRestClassifier(SVC(kernel='rbf', probability=True))

classifier.fit(X\_train, y\_train)

y\_score = classifier.predict\_proba(X\_test)

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test\_bin[:, i], y\_score[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test\_bin.ravel(), y\_score.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

plt.figure(figsize=(8, 6))

plt.plot(fpr["micro"], tpr["micro"],

label='micro-average ROC curve (area = {0:0.2f})'

''.format(roc\_auc["micro"]))

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Micro-averaged ROC Curve')

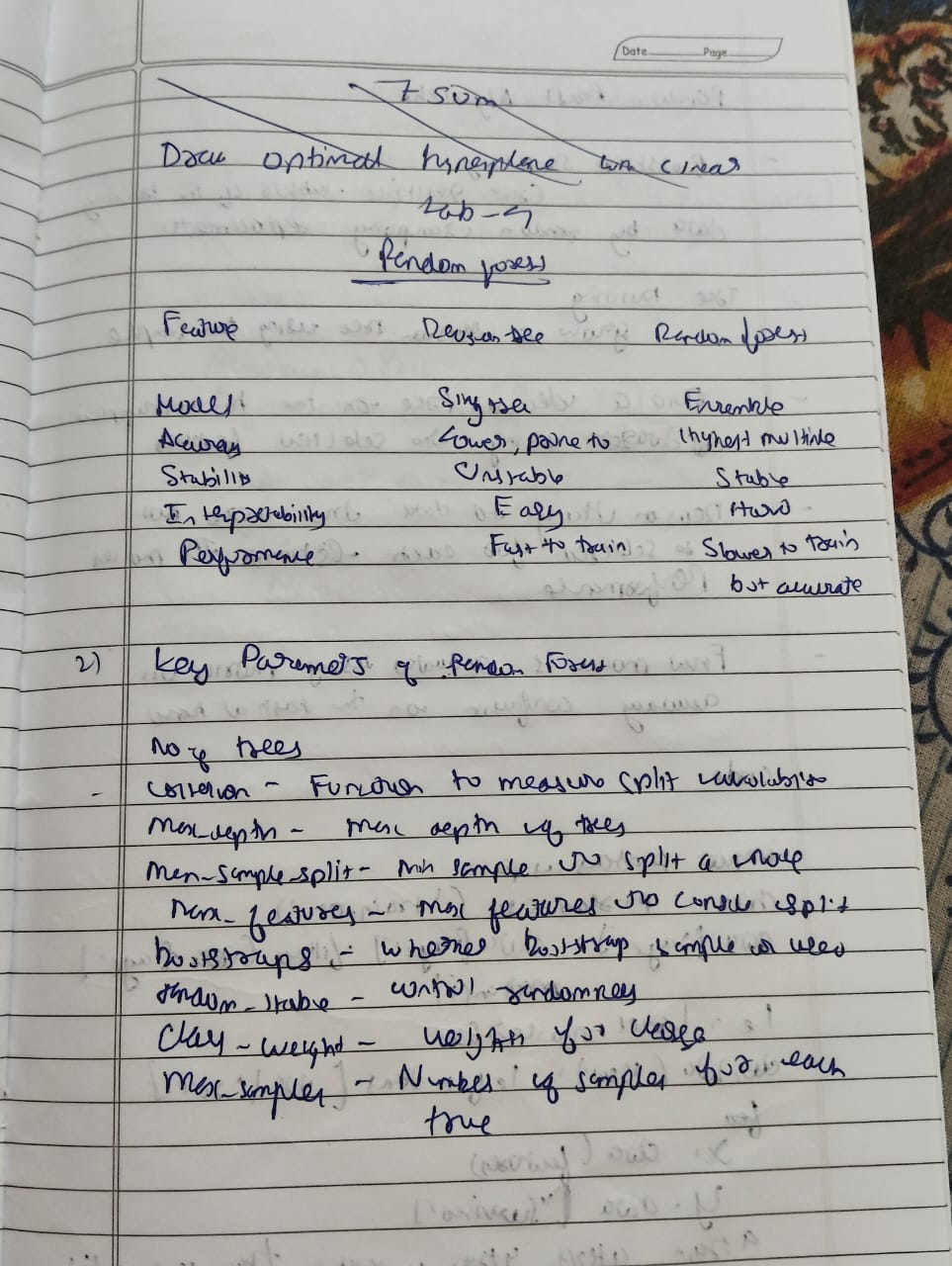
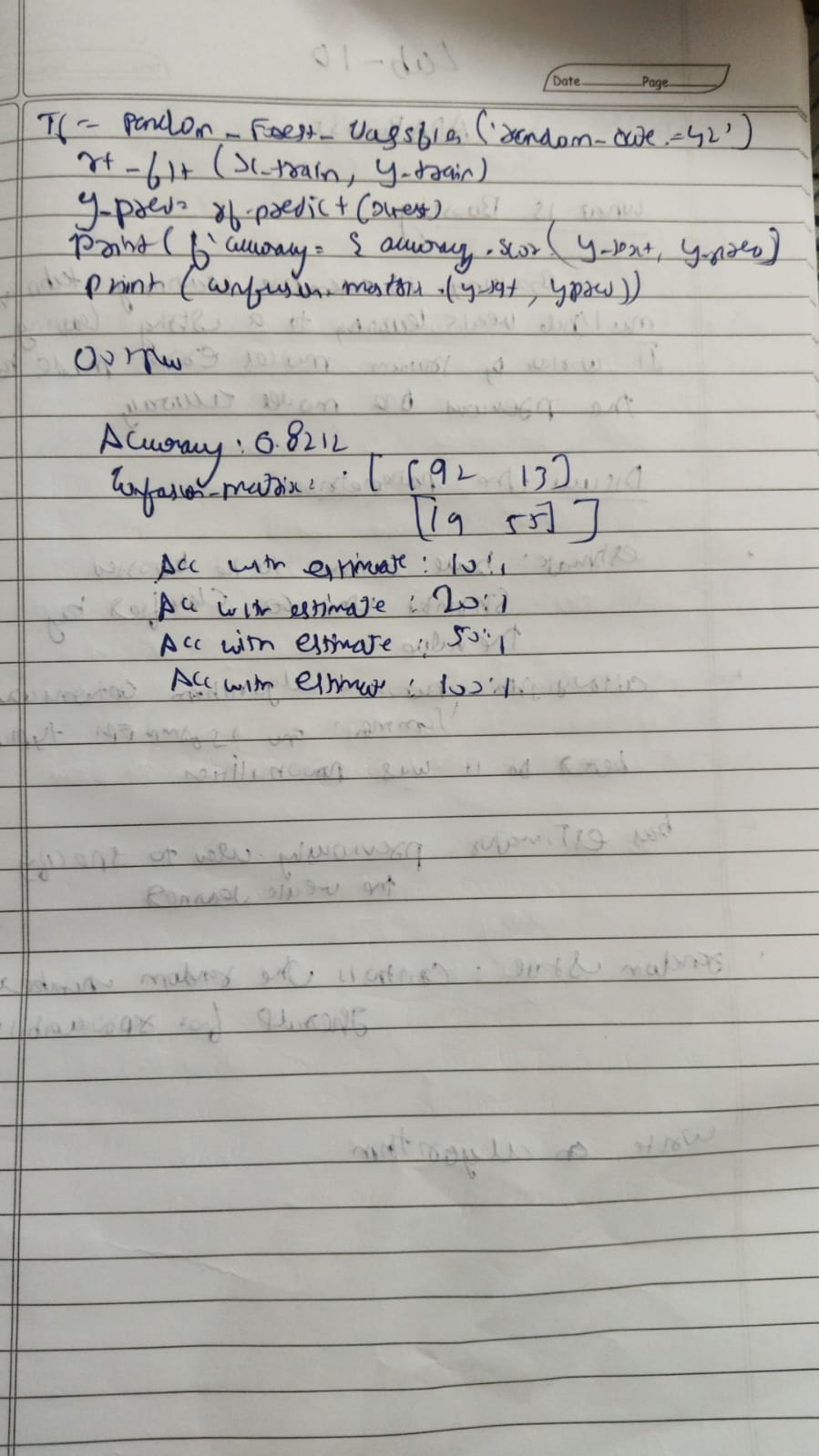
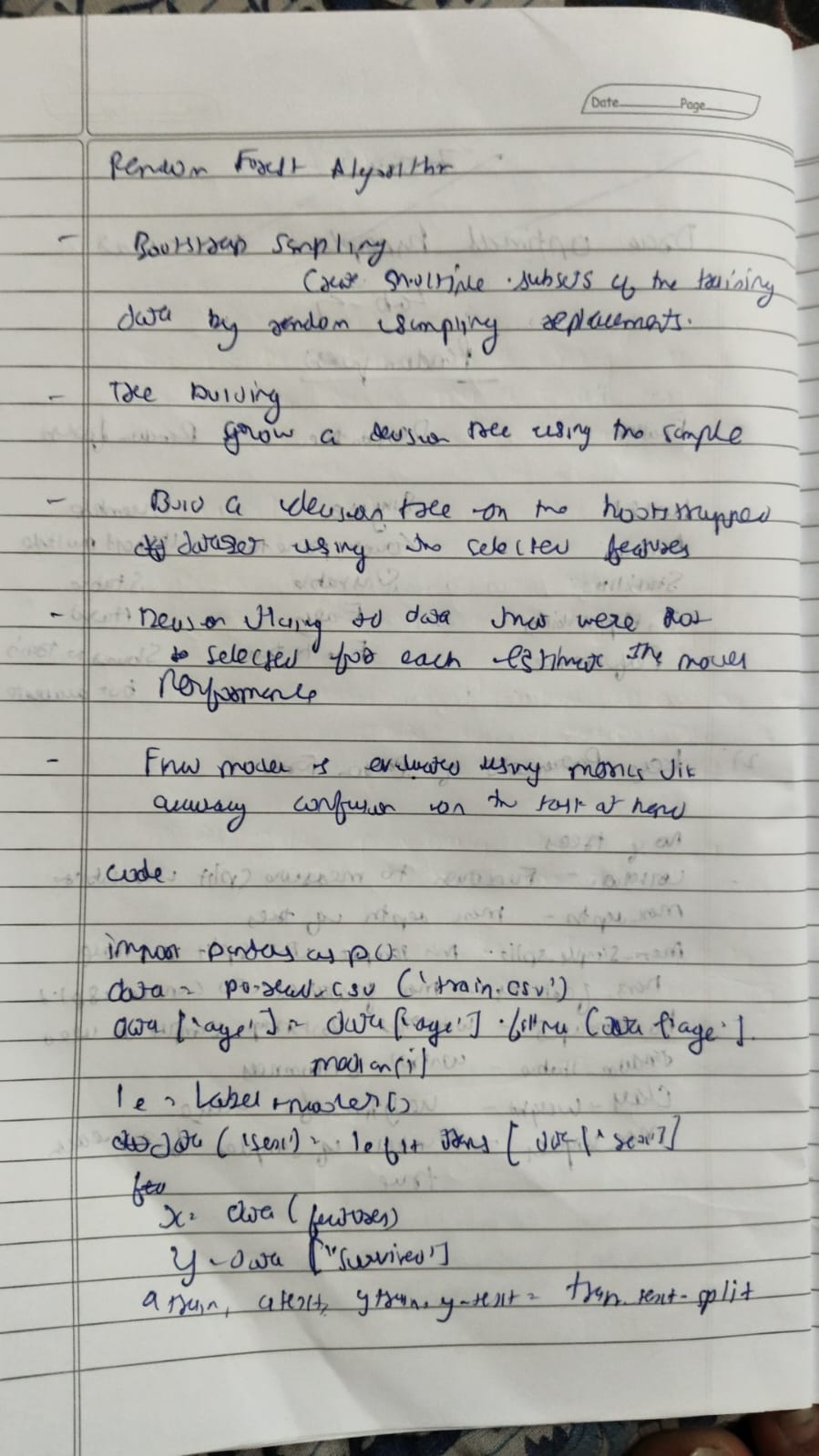
plt.legend(loc="lower right")

plt.show()

print(f"Micro-averaged AUC: {roc\_auc['micro']}")

PROGRAM 8 Implement Random forest ensemble method on a given dataset.

Screenshot

Code

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('/content/iris (1).csv')

# Prepare features and target

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

y = df['species']

# Split 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)

# Build Random Forest with default n\_estimators (10)

rf\_default = RandomForestClassifier(n\_estimators=10, random\_state=42)

rf\_default.fit(X\_train, y\_train)

y\_pred\_default = rf\_default.predict(X\_test)

# Measure accuracy

default\_score = accuracy\_score(y\_test, y\_pred\_default)

print(f"Default RF accuracy (n\_estimators=10): {default\_score:.4f}")

# Fine-tune the number of trees

scores = []

n\_range = range(1, 101)

for n in n\_range:

rf = RandomForestClassifier(n\_estimators=n, random\_state=42)

rf.fit(X\_train, y\_train)

y\_pred = rf.predict(X\_test)

score = accuracy\_score(y\_test, y\_pred)

scores.append(score)

# Find the best score and number of trees

best\_score = max(scores)

best\_n = n\_range[scores.index(best\_score)]

print(f"Best RF accuracy: {best\_score:.4f} with n\_estimators={best\_n}")

# Optional: Plot accuracy vs number of estimators

plt.figure(figsize=(10, 6))

plt.plot(n\_range, scores, marker='o')

plt.title('Random Forest Accuracy vs Number of Trees')

plt.xlabel('Number of Trees (n\_estimators)')

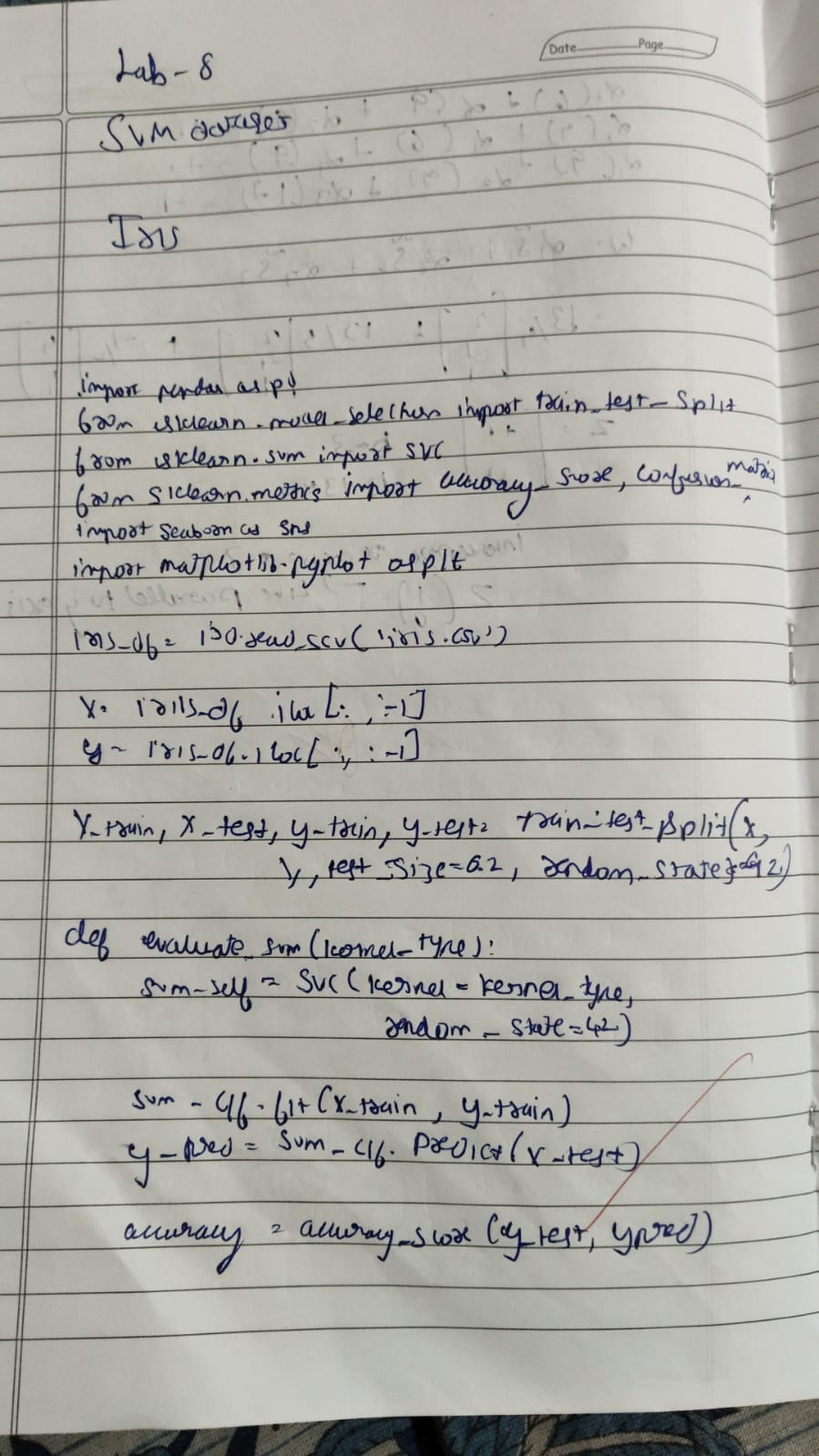
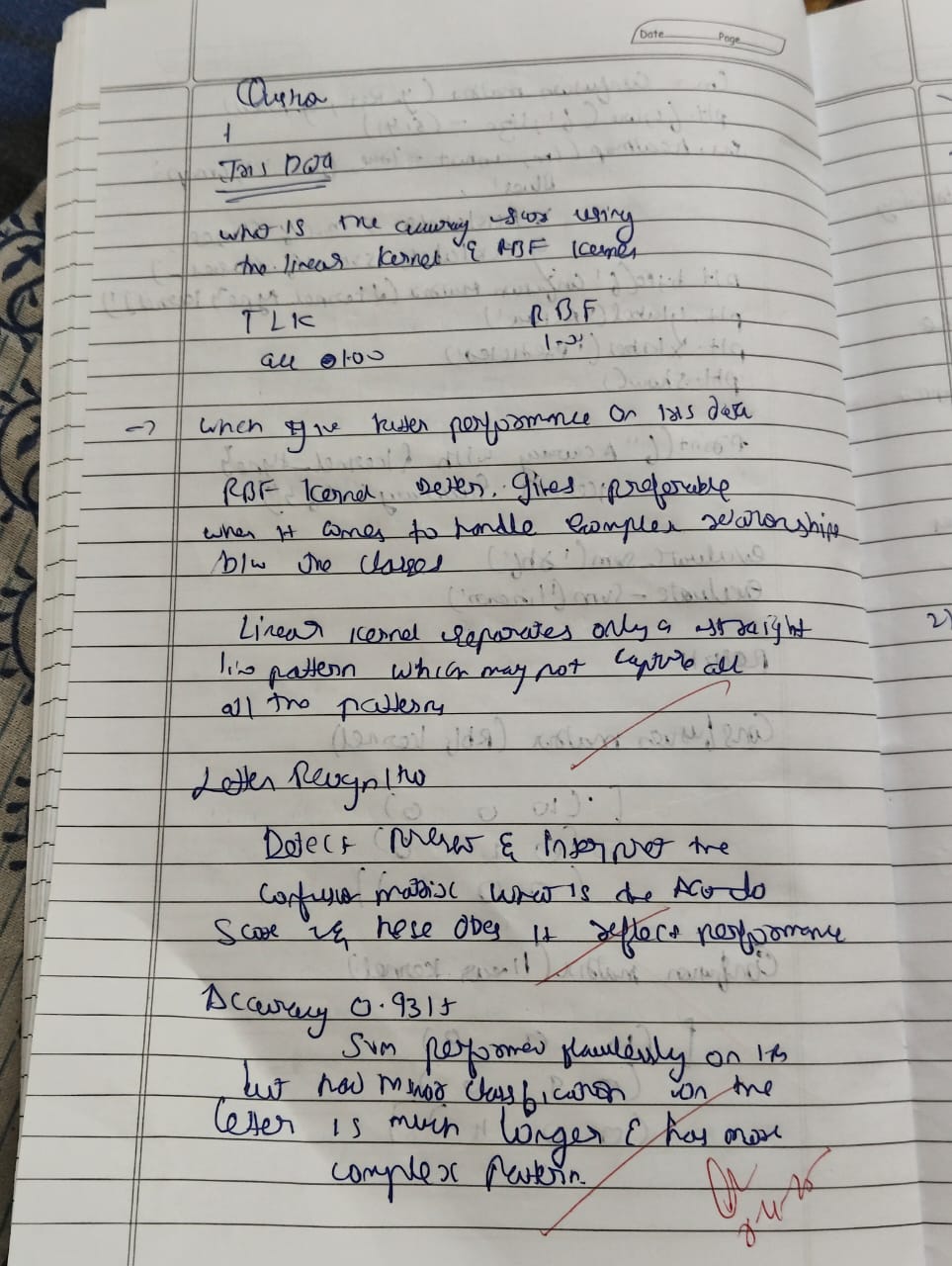
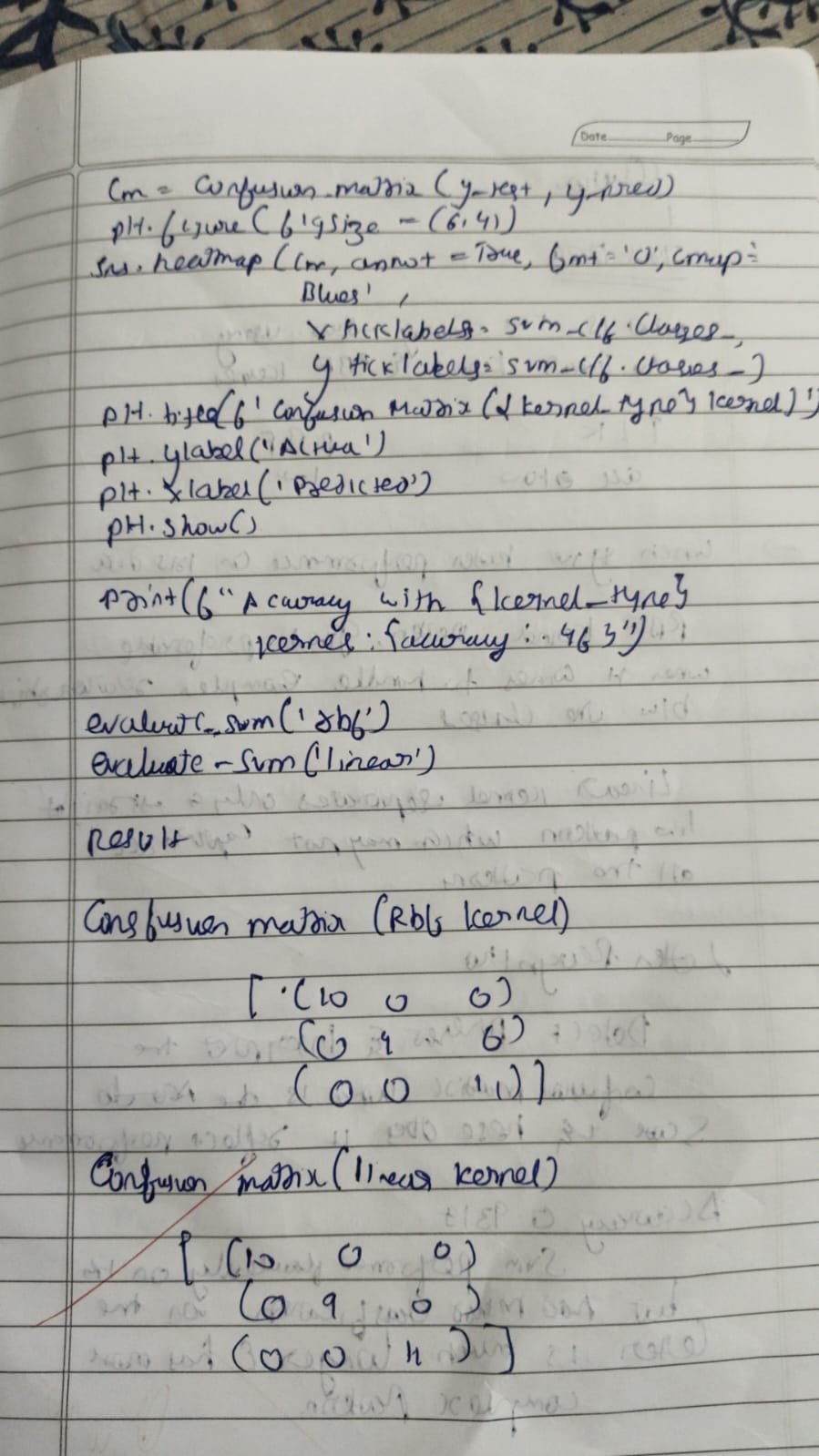
plt.ylabel('Accuracy')

plt.grid(True)

plt.show()

PROGRAM 9 Implement Boosting ensemble method on a given dataset.

Screenshot

Code

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

# Load dataset

df = pd.read\_csv("/content/income.csv")

# Drop rows with missing values

df.dropna(inplace=True)

# Encode categorical columns

label\_encoders = {}

for column in df.select\_dtypes(include=['object']).columns:

le = LabelEncoder()

df[column] = le.fit\_transform(df[column])

label\_encoders[column] = le

# Separate features and target

X = df.drop(columns=['income\_level'], errors='ignore', axis=1)

y = df['income\_level']

# Split into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# AdaBoost with 10 estimators

model\_10 = AdaBoostClassifier(n\_estimators=10, random\_state=42)

model\_10.fit(X\_train, y\_train)

y\_pred\_10 = model\_10.predict(X\_test)

score\_10 = accuracy\_score(y\_test, y\_pred\_10)

print(f"Accuracy with 10 estimators: {score\_10:.4f}")

# Fine-tune number of estimators

best\_score = 0

best\_n = 0

estimators\_range = list(range(10, 201, 10))

scores = []

for n in estimators\_range:

model = AdaBoostClassifier(n\_estimators=n, random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

score = accuracy\_score(y\_test, y\_pred)

scores.append(score)

print(f"n\_estimators={n}, Accuracy={score:.4f}")

if score > best\_score:

best\_score = score

best\_n = n

print(f"\nBest Accuracy: {best\_score:.4f} using {best\_n} estimators")

# Plot accuracy vs number of estimators

plt.figure(figsize=(7, 4))

plt.plot(estimators\_range, scores, marker='o', linestyle='-', color='blue')

plt.title("Accuracy vs Number of Estimators (AdaBoost)")

plt.xlabel("Number of Estimators (Trees)")

plt.ylabel("Accuracy")

plt.grid(True)

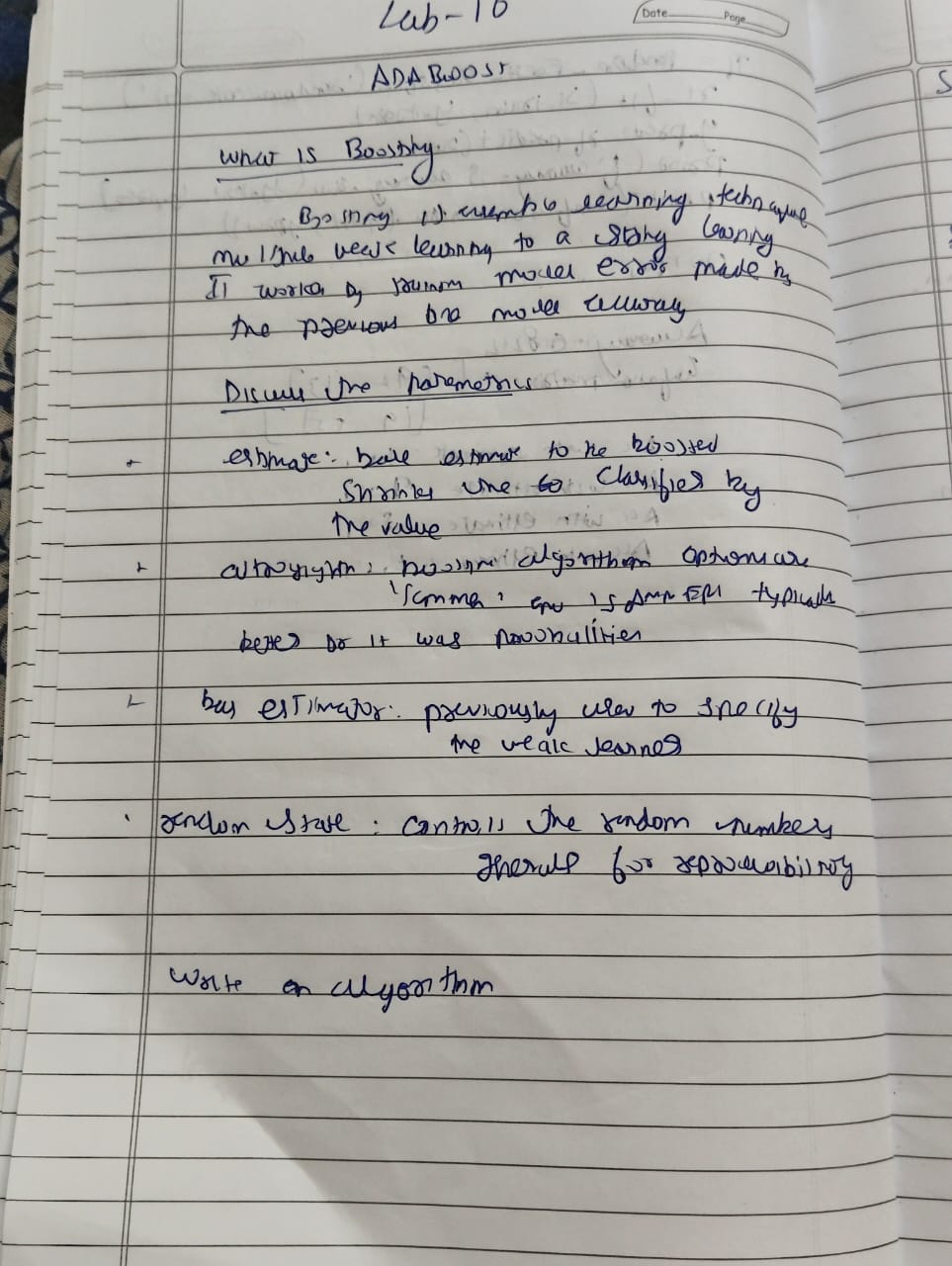
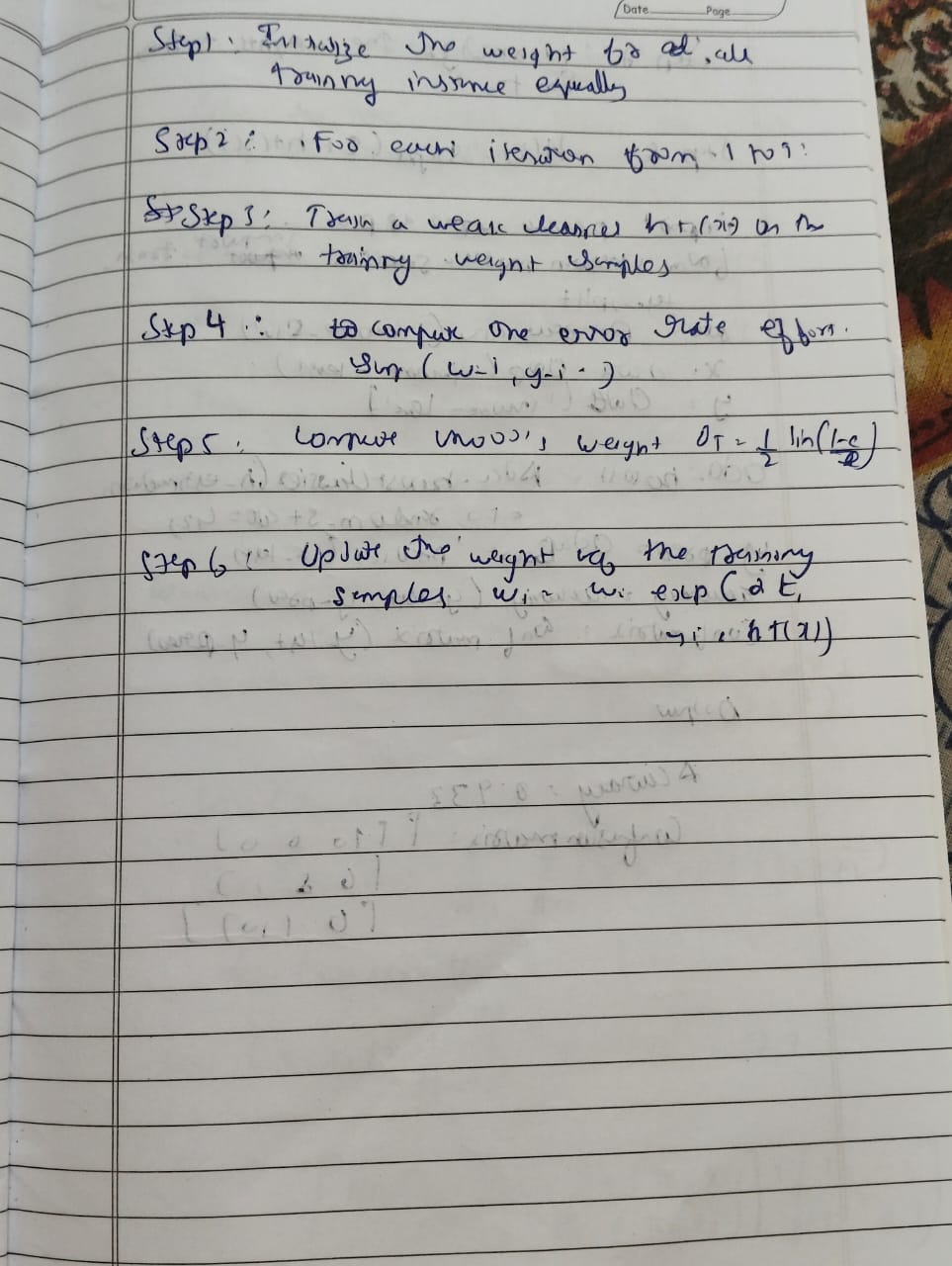
plt.xticks(estimators\_range)

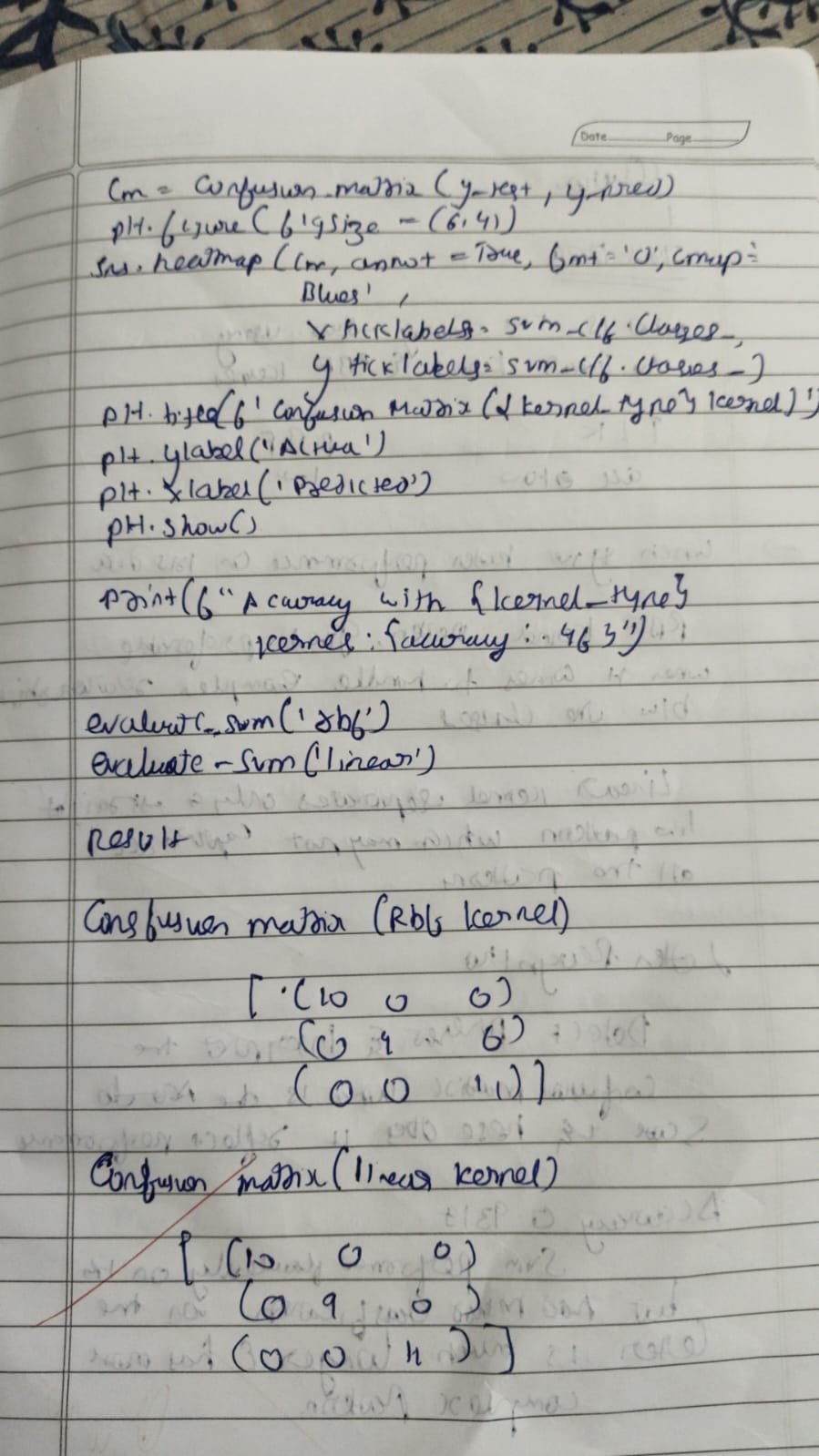
plt.tight\_layout()

plt.show()

PROGRAM 10 Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Screenshot



Code

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy\_score

from scipy.stats import mode

import matplotlib.pyplot as plt

# Step 1: Generate sample data and save to CSV

np.random.seed(42)

names = [f"Person\_{i}" for i in range(50)]

ages = np.random.randint(20, 60, 50)

income = np.random.randint(30000, 120000, 50)

df = pd.DataFrame({'Name': names, 'Age': ages, 'Income': income})

df.to\_csv("income.csv", index=False)

# Step 2: Load the data

data = pd.read\_csv("income.csv")

# Drop 'Name' and extract features

X = data[['Age', 'Income']]

# Step 3: Split the data

X\_train, X\_test = train\_test\_split(X, test\_size=0.2, random\_state=42)

# Step 4: Perform scaling

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Step 5: Plot SSE vs number of clusters (Elbow method)

sse = []

k\_range = range(1, 11)

for k in k\_range:

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(X\_train\_scaled)

sse.append(kmeans.inertia\_)

plt.figure(figsize=(8, 4))

plt.plot(k\_range, sse, marker='o')

plt.xlabel('Number of clusters')

plt.ylabel('SSE (Inertia)')

plt.title('Elbow Method For Optimal k')

plt.grid(True)

plt.show()

# Step 6: Choose optimal number of clusters (say 3) and fit model

optimal\_k = 3

kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)

kmeans.fit(X\_train\_scaled)

# Predict on test data

predictions = kmeans.predict(X\_test\_scaled)

# Note: There's no ground truth labels, but for demonstration,

# we can try assigning true clusters (via KMeans on full data)

# and see if predicted clusters align

# Fit on full data to assign pseudo-labels

full\_kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)

true\_clusters = full\_kmeans.fit\_predict(scaler.fit\_transform(X))

# Align predicted clusters using majority voting (only for demonstration)

# Match predicted labels to closest true labels

def map\_clusters(true\_labels, pred\_labels):

labels = np.zeros\_like(pred\_labels)

for i in range(optimal\_k):

mask = (pred\_labels == i)

if np.sum(mask) == 0:

continue

labels[mask] = mode(true\_labels[mask])[0]

return labels

mapped\_preds = map\_clusters(true\_clusters[X\_test.index], predictions)

accuracy = accuracy\_score(true\_clusters[X\_test.index], mapped\_preds)

print(f"Approximate Clustering Accuracy: {accuracy:.2f}")

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

# Step 1: Load Iris dataset

iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df['target'] = iris.target

# Keep only petal length and petal width

X = df[['petal length (cm)', 'petal width (cm)']].values

# Step 2: Check impact of scaling

# Try without scaling

sse\_unscaled = []

for k in range(1, 11):

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(X)

sse\_unscaled.append(kmeans.inertia\_)

# Now scale the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

sse\_scaled = []

for k in range(1, 11):

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(X\_scaled)

sse\_scaled.append(kmeans.inertia\_)

# Step 3: Plot Elbow Comparison (Scaled vs Unscaled)

plt.figure(figsize=(10, 5))

plt.plot(range(1, 11), sse\_unscaled, marker='o', label='Unscaled')

plt.plot(range(1, 11), sse\_scaled, marker='s', label='Scaled')

plt.title('Elbow Method (Petal Features Only)')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('SSE (Inertia)')

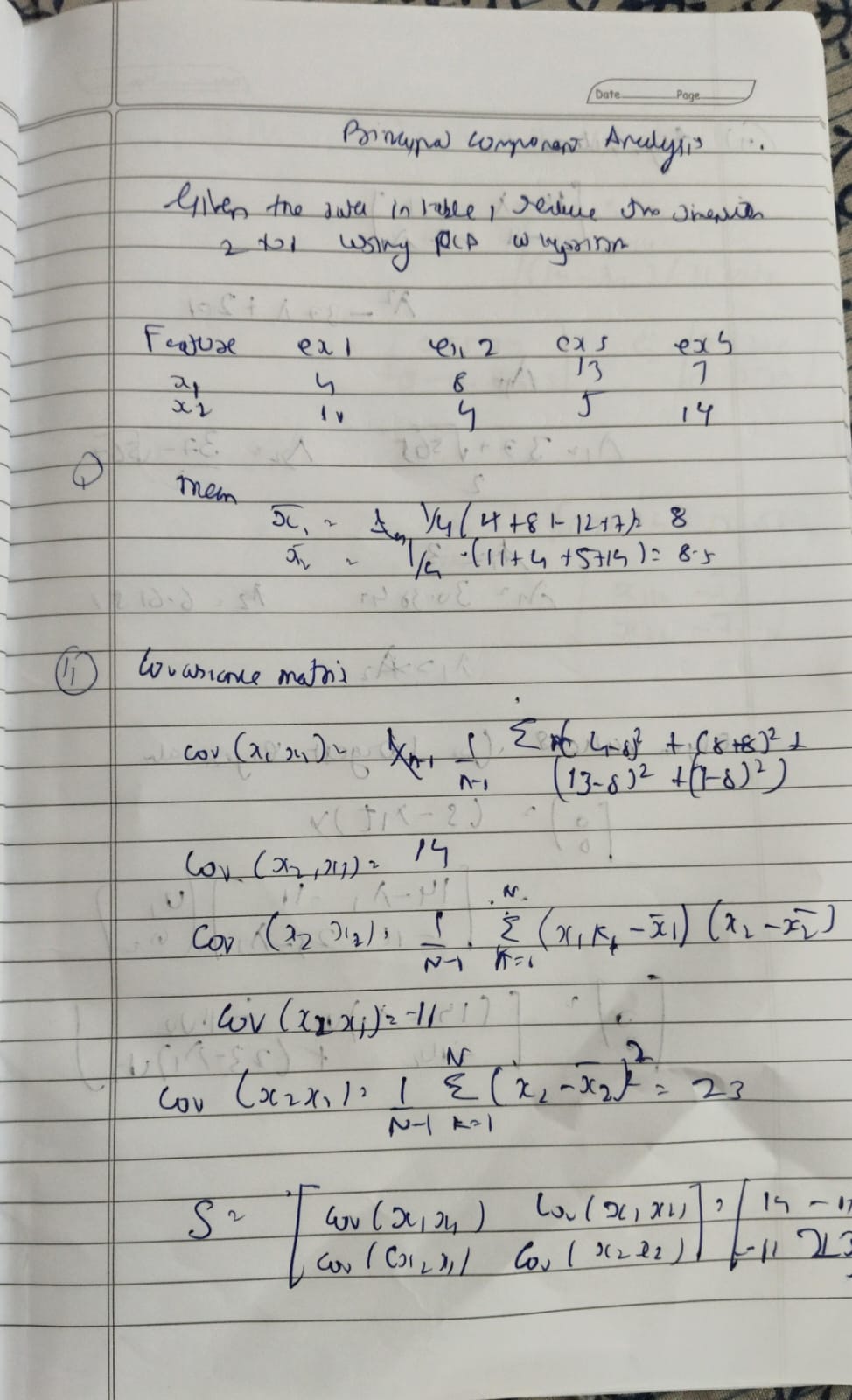
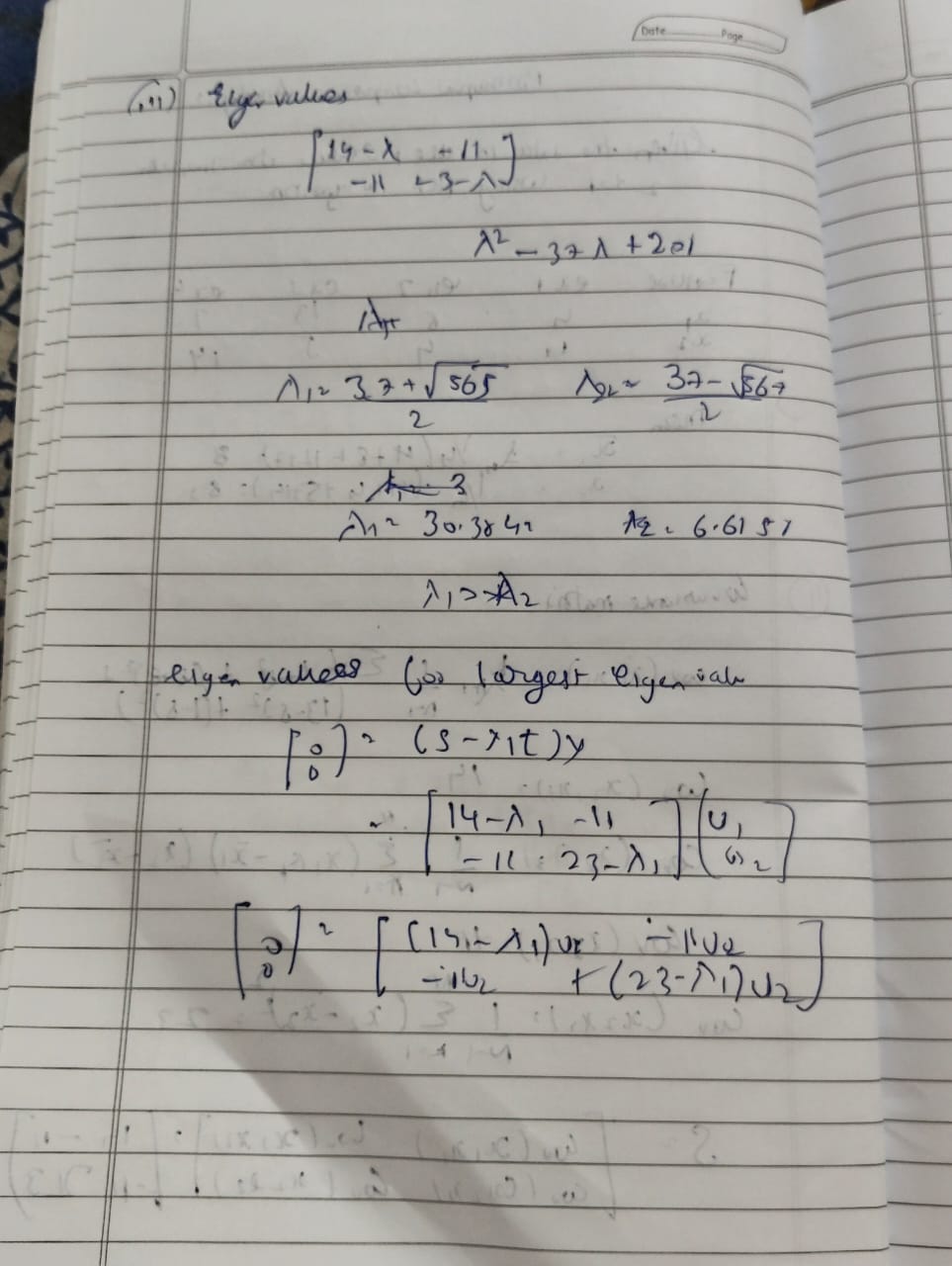
plt.legend()

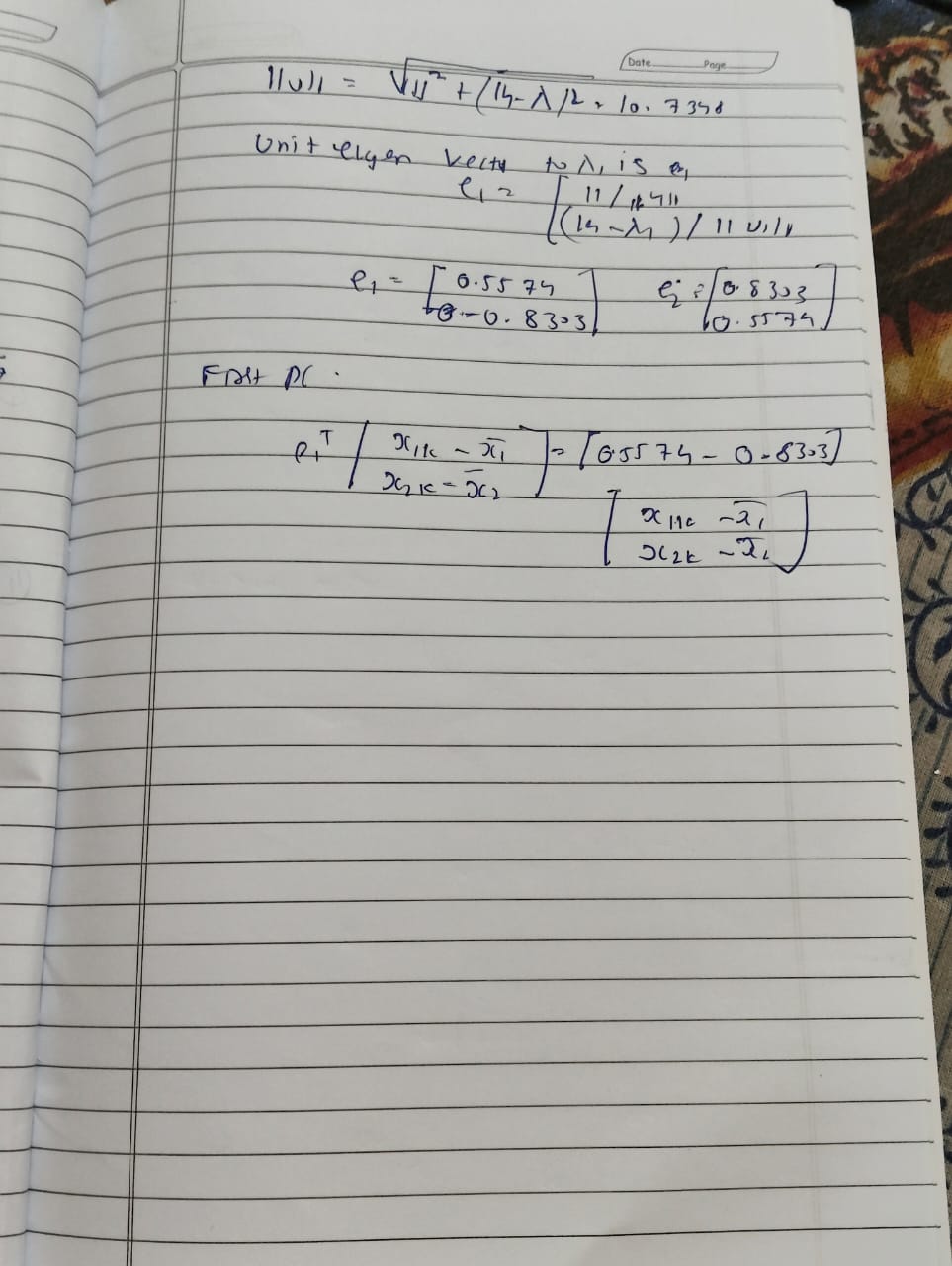
plt.grid(True)

plt.show()

PROGRAM 11 Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Screenshot



Code

<a href="https://colab.research.google.com/github/mukund166/ML\_Lab\_1BM22CS166/blob/main/1BM22CS166\_Lab\_10\_PCA.ipynb" target="\_parent"><img src="https://colab.research.google.com/assets/colab-badge.svg" alt="Open In Colab"/></a>

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy\_score

# 1. Load data

df = pd.read\_csv("heart.csv")

# 2. Label‑encode binary text columns

le = LabelEncoder()

for col in ["Sex", "ExerciseAngina"]:

df[col] = le.fit\_transform(df[col])

# 3. Separate features and target

X = df.drop("HeartDisease", axis=1)

y = df["HeartDisease"]

# 4. Build preprocessing pipeline:

# - One‑hot for multi‑category columns (using sparse\_output=False)

# - passthrough the rest

# - then scale everything

cat\_cols = ["ChestPainType", "RestingECG", "ST\_Slope"]

preprocessor = Pipeline([

("onehot", ColumnTransformer([

("ohe", OneHotEncoder(sparse\_output=False, drop="first"), cat\_cols)

], remainder="passthrough")),

("scaler", StandardScaler())

])

# 5. Apply preprocessing

X\_proc = preprocessor.fit\_transform(X)

# 6. Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_proc, y, test\_size=0.2, random\_state=42

)

# 7. Define models

models = {

"SVM": SVC(random\_state=42),

"LogisticRegression": LogisticRegression(max\_iter=1000, random\_state=42),

"RandomForest": RandomForestClassifier(random\_state=42)

}

# 8. Train & evaluate before PCA

print("=== Accuracies BEFORE PCA ===")

scores\_before = {}

for name, clf in models.items():

clf.fit(X\_train, y\_train)

preds = clf.predict(X\_test)

acc = accuracy\_score(y\_test, preds)

scores\_before[name] = acc

print(f"{name:17s}: {acc:.4f}")

# 9. Apply PCA (retain 95% variance)

pca = PCA(n\_components=0.95, random\_state=42)

X\_train\_pca = pca.fit\_transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

print(f"\nPCA retained {pca.n\_components\_} components, "

f"explained variance = {pca.explained\_variance\_ratio\_.sum():.4f}\n")

# 10. Train & evaluate after PCA

print("=== Accuracies AFTER PCA ===")

scores\_after = {}

for name, clf in models.items():

clf.fit(X\_train\_pca, y\_train)

preds = clf.predict(X\_test\_pca)

acc = accuracy\_score(y\_test, preds)

scores\_after[name] = acc

print(f"{name:17s}: {acc:.4f}")