# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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



## LAB REPORT

### on

Machine Learning (23CS6PCMAL)

#### Submitted by

**Akshith M Naik (1BM22CS033)**

#### 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 **Akshith M Naik (1BM22CS033),** 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

Dr. Seema Patil

Assistant Professor Department of CSE, BMSCE

# Index

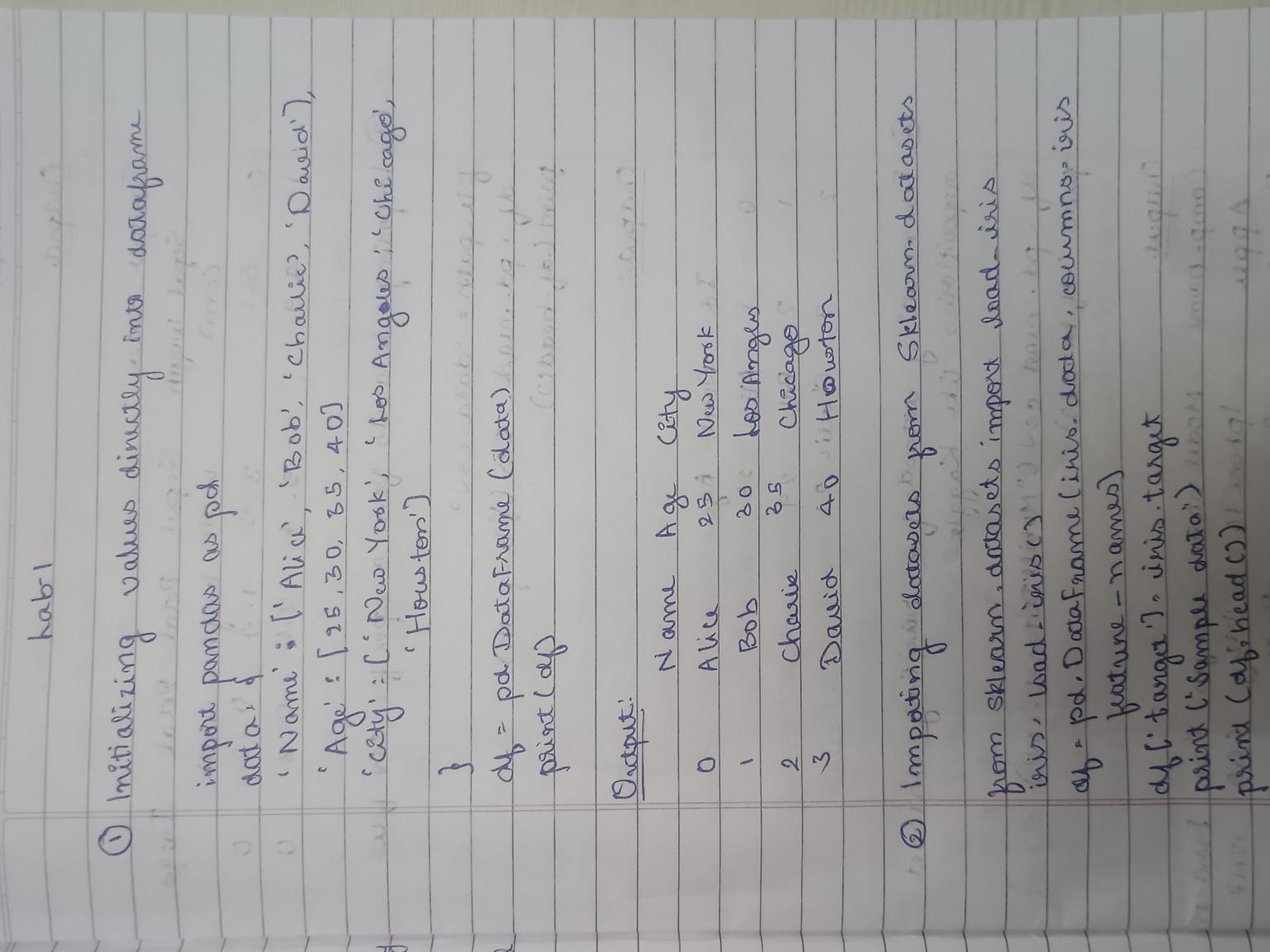
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 4-3-2025 | Write a python program to import and export data using Pandas library functions | 5-8 |
| 2 | 11-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 9-13 |
| 3 | 18-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 14-17 |
| 4 | 1-4-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 18-21 |
| 5 | 8-4-2025 | Build Logistic Regression Model for a given dataset | 22-25 |
| 6 | 15-4-2025 | Build KNN Classification model for a given dataset. | 26-30 |
| 7 | 15-4-2025 | Build Support vector machine model for a given dataset | 31-34 |
| 8 | 22-4-2025 | Implement Random forest ensemble method on a given dataset. | 35-37 |
| 9 | 22-4-2025 | Implement Boosting ensemble method on a given dataset. | 38-41 |
| 10 | 29-4-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 42-46 |
| 11 | 29-4-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 47-52 |

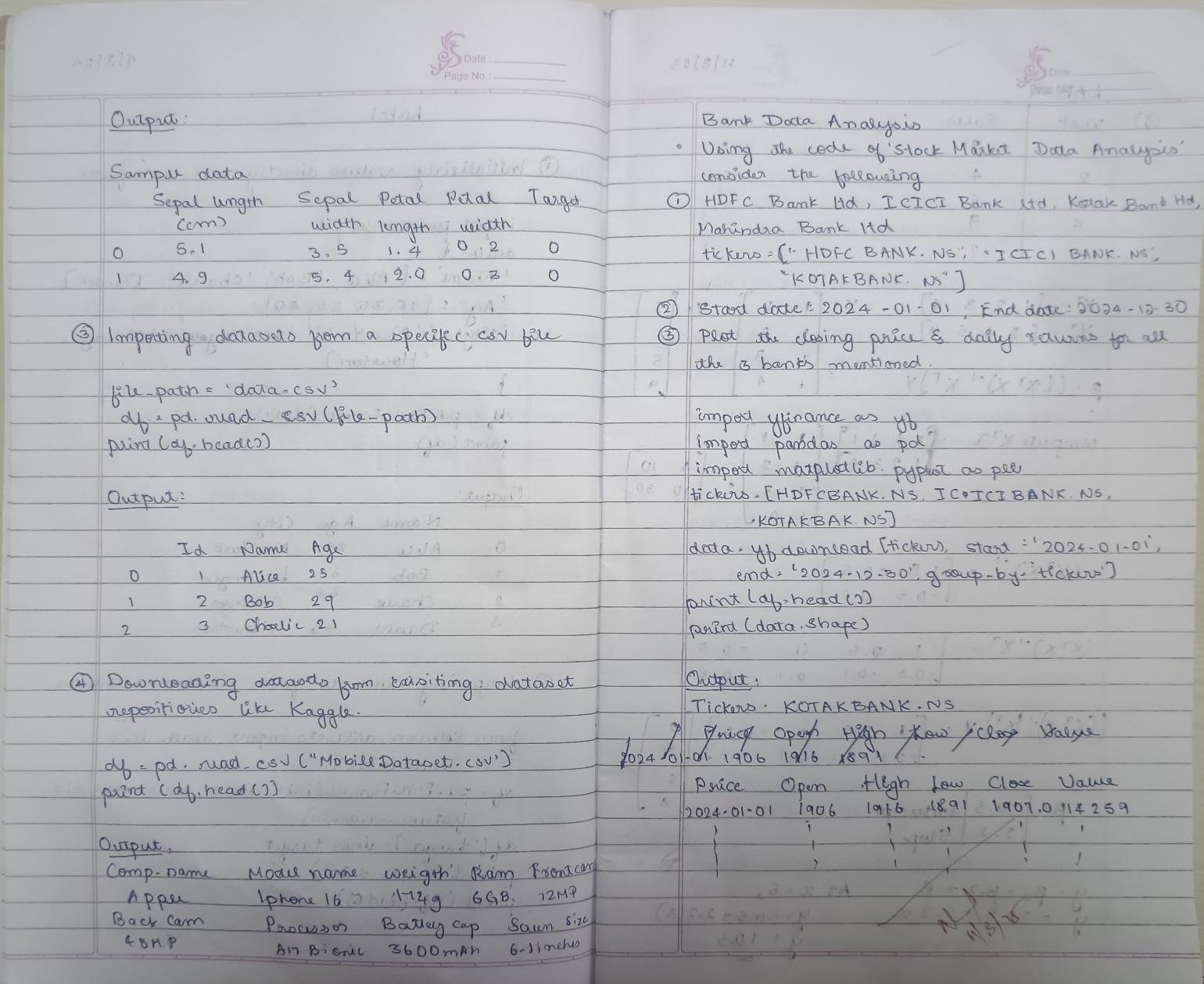
Github Link: <https://github.com/Akshith033/6thSem-ML-Lab>

##### Program 1

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

Screenshot





Code:

import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

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

data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group\_by='ticker')

print("First 5 rows of the dataset:")

print(data.head())

print("\nShape of the dataset:")

print(data.shape)

print("\nColumn names:")

print(data.columns)

hdfc\_data = data['HDFCBANK.NS']

print("\nSummary statistics for HDFC Bank:")

print(hdfc\_data.describe())

hdfc\_data['Daily Return'] = hdfc\_data['Close'].pct\_change()

icici\_data = data['ICICIBANK.NS']

print("\nSummary statistics for ICICI Bank:")

print(icici\_data.describe())

icici\_data['Daily Return'] = icici\_data['Close'].pct\_change()

kotak\_data = data['KOTAKBANK.NS']

print("\nSummary statistics for Kotak Mahindra Bank:")

print(kotak\_data.describe())

kotak\_data['Daily Return'] = kotak\_data['Close'].pct\_change()

plt.figure(figsize=(14, 10))

plt.subplot(3, 2, 1)

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

plt.subplot(3, 2, 2)

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

plt.subplot(3, 2, 3)

icici\_data['Close'].plot(title="ICICI Bank - Closing Price")

plt.subplot(3, 2, 4)

icici\_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='orange')

plt.subplot(3, 2, 5)

kotak\_data['Close'].plot(title="Kotak Mahindra Bank - Closing Price")

plt.subplot(3, 2, 6)

kotak\_data['Daily Return'].plot(title="Kotak Mahindra Bank - Daily Returns", color='orange')

plt.tight\_layout()

plt.show()

hdfc\_data.to\_csv('hdfc\_bank\_data.csv')

icici\_data.to\_csv('icici\_bank\_data.csv')

kotak\_data.to\_csv('kotak\_bank\_data.csv')

print("\nHDFC Bank data saved to 'hdfc\_bank\_data.csv'.")

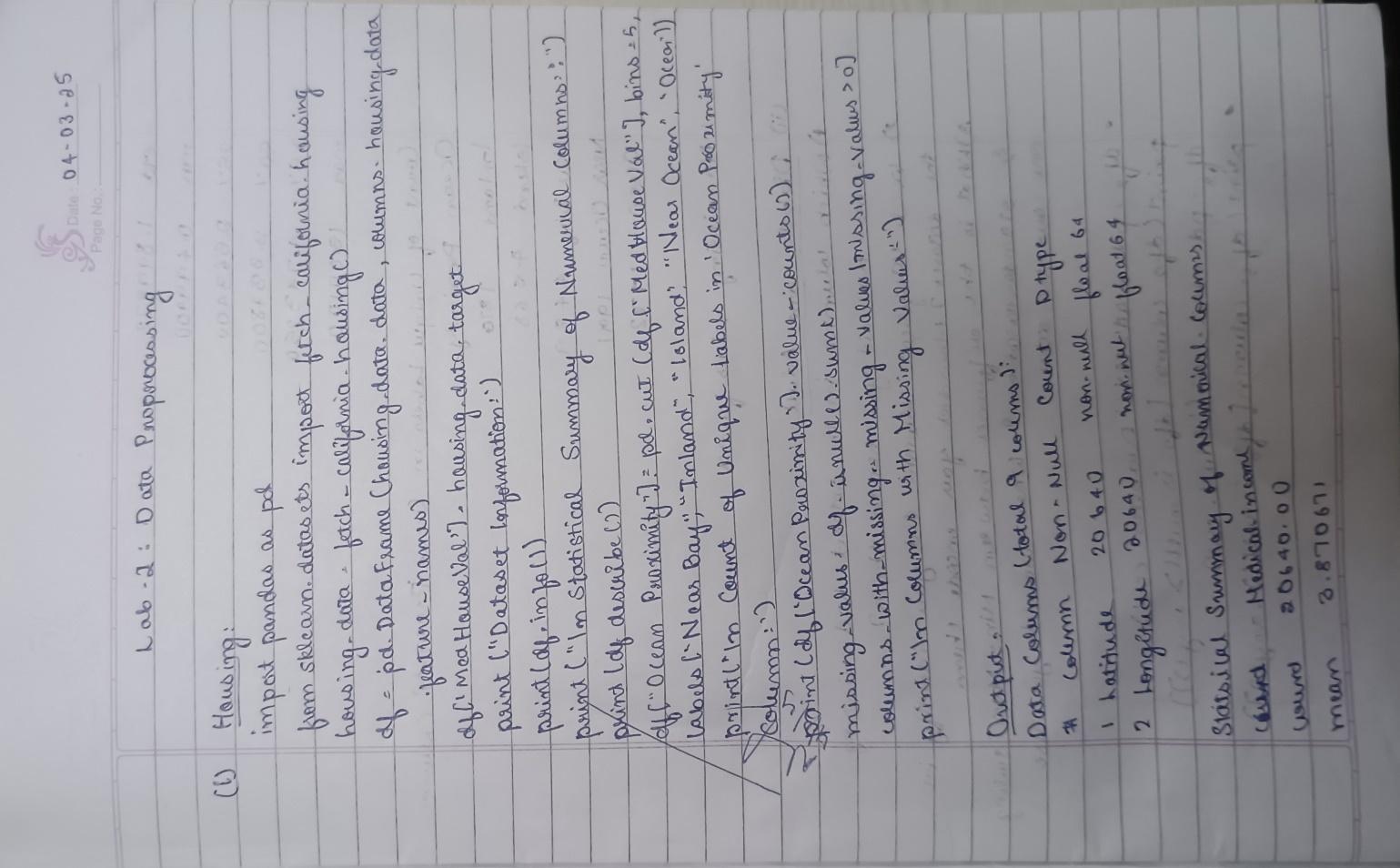
print("ICICI Bank data saved to 'icici\_bank\_data.csv'.")

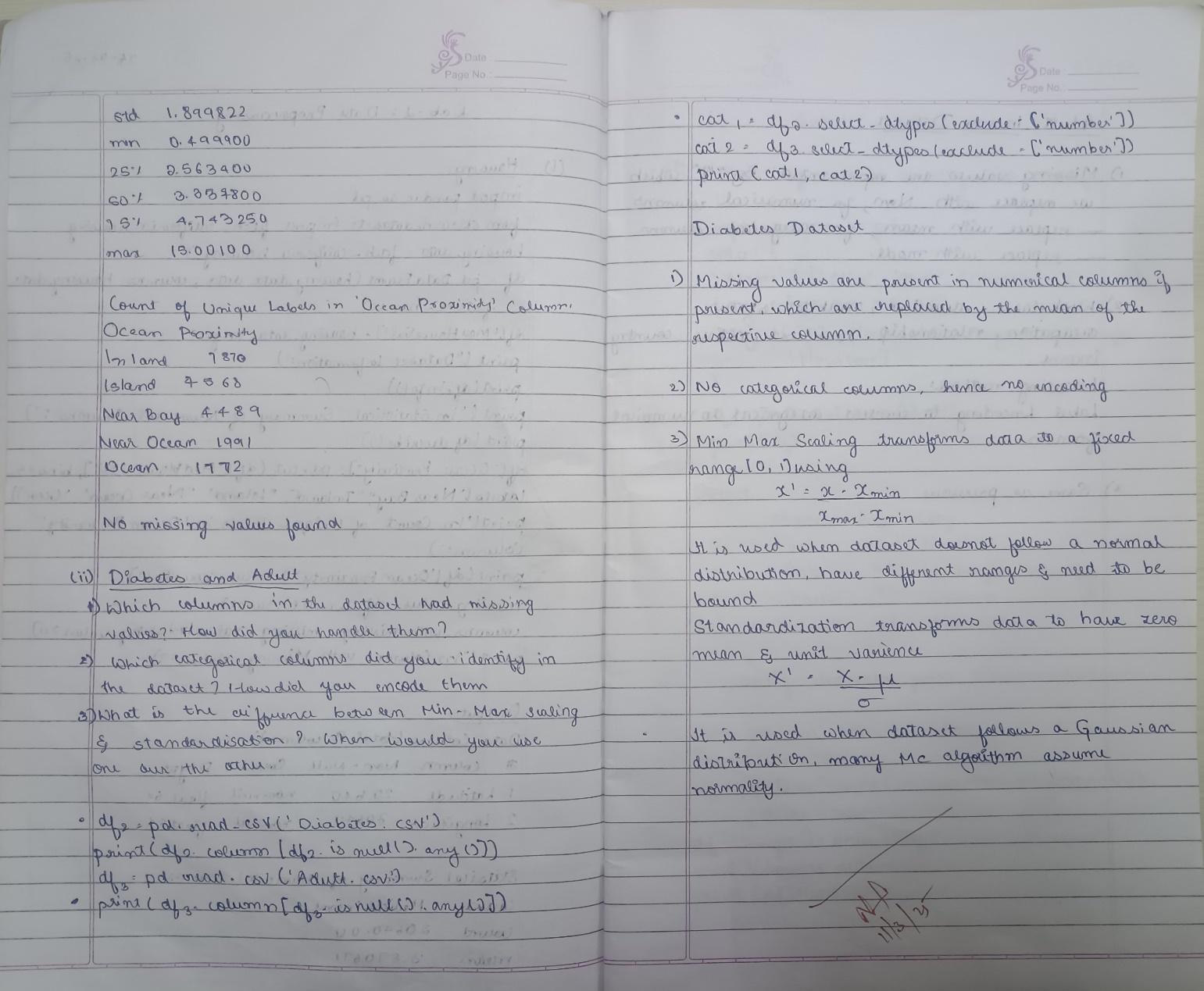
print("Kotak Bank data saved to 'kotak\_bank\_data.csv'.")

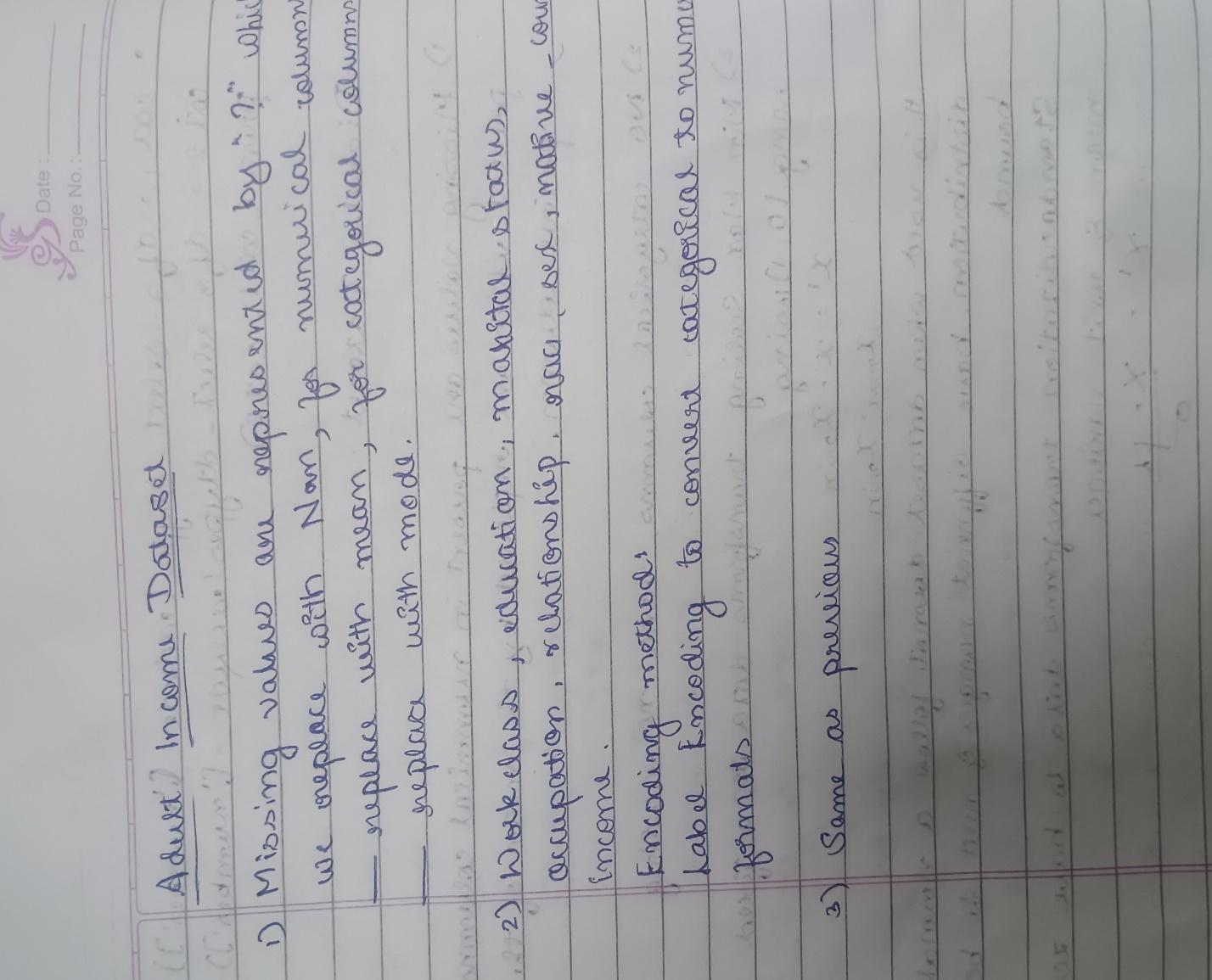
##### Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot







Code:

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

import seaborn as sns

import matplotlib.pyplot as plt

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

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

print("Diabetes Dataset:")

print(diabetes\_data.head())

print("\nAdult Income Dataset:")

print(adult\_income\_data.head())

diabetes\_numerical\_cols = diabetes\_data.select\_dtypes(include=[np.number]).columns

diabetes\_categorical\_cols = diabetes\_data.select\_dtypes(include=[object]).columns

diabetes\_imputer\_num = SimpleImputer(strategy='median')

diabetes\_data[diabetes\_numerical\_cols] = diabetes\_imputer\_num.fit\_transform(diabetes\_data[diabetes\_numerical\_cols])

diabetes\_imputer\_cat = SimpleImputer(strategy='most\_frequent')

diabetes\_data[diabetes\_categorical\_cols] = diabetes\_imputer\_cat.fit\_transform(diabetes\_data[diabetes\_categorical\_cols])

adult\_income\_numerical\_cols = adult\_income\_data.select\_dtypes(include=[np.number]).columns

adult\_income\_categorical\_cols = adult\_income\_data.select\_dtypes(include=[object]).columns

adult\_income\_imputer\_num = SimpleImputer(strategy='median')

adult\_income\_data[adult\_income\_numerical\_cols] = adult\_income\_imputer\_num.fit\_transform(adult\_income\_data[adult\_income\_numerical\_cols])

adult\_income\_imputer\_cat = SimpleImputer(strategy='most\_frequent')

adult\_income\_data[adult\_income\_categorical\_cols] = adult\_income\_imputer\_cat.fit\_transform(adult\_income\_data[adult\_income\_categorical\_cols])

categorical\_columns\_adult = adult\_income\_data.select\_dtypes(include=['object']).columns

label\_encoder = LabelEncoder()

for col in categorical\_columns\_adult:

    adult\_income\_data[col] = label\_encoder.fit\_transform(adult\_income\_data[col])

def detect\_and\_remove\_outliers(df):

    numerical\_df = df.select\_dtypes(include=[np.number])

    Q1 = numerical\_df.quantile(0.25)

    Q3 = numerical\_df.quantile(0.75)

    IQR = Q3 - Q1

    return df[~((numerical\_df < (Q1 - 1.5 \* IQR)) | (numerical\_df > (Q3 + 1.5 \* IQR))).any(axis=1)]

diabetes\_data\_cleaned = detect\_and\_remove\_outliers(diabetes\_data)

adult\_income\_data\_cleaned = detect\_and\_remove\_outliers(adult\_income\_data)

min\_max\_scaler = MinMaxScaler()

diabetes\_numerical\_cols = diabetes\_data\_cleaned.select\_dtypes(include=[np.number]).columns

diabetes\_data\_normalized = diabetes\_data\_cleaned.copy()

diabetes\_data\_normalized[diabetes\_numerical\_cols] = min\_max\_scaler.fit\_transform(diabetes\_data\_cleaned[diabetes\_numerical\_cols])

adult\_income\_numerical\_cols = adult\_income\_data\_cleaned.select\_dtypes(include=[np.number]).columns

adult\_income\_data\_normalized = adult\_income\_data\_cleaned.copy()

adult\_income\_data\_normalized[adult\_income\_numerical\_cols] = min\_max\_scaler.fit\_transform(adult\_income\_data\_cleaned[adult\_income\_numerical\_cols])

standard\_scaler = StandardScaler()

diabetes\_data\_standardized = diabetes\_data\_cleaned.copy()

diabetes\_data\_standardized[diabetes\_numerical\_cols] = standard\_scaler.fit\_transform(diabetes\_data\_cleaned[diabetes\_numerical\_cols])

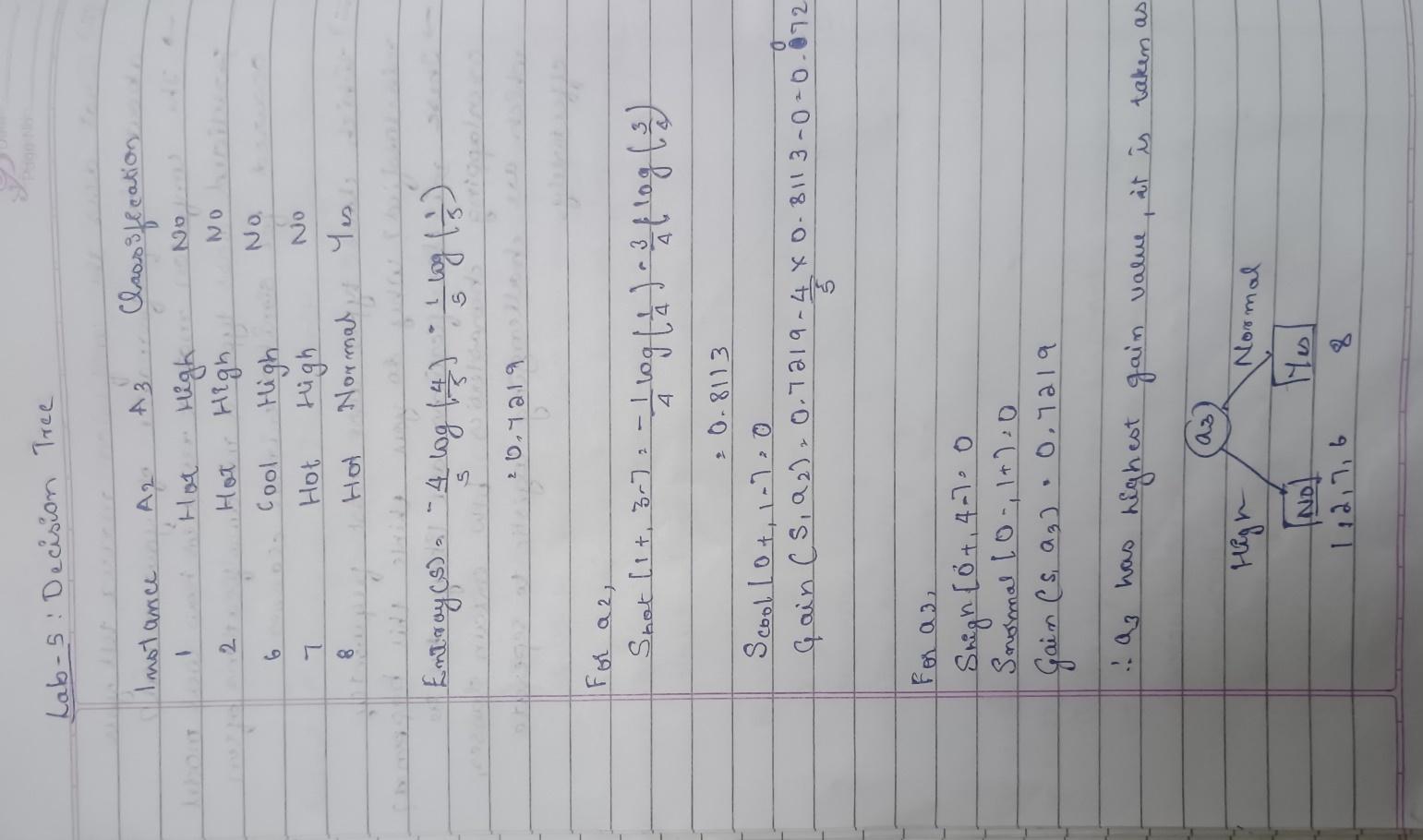
adult\_income\_data\_standardized = adult\_income\_data\_cleaned.copy()

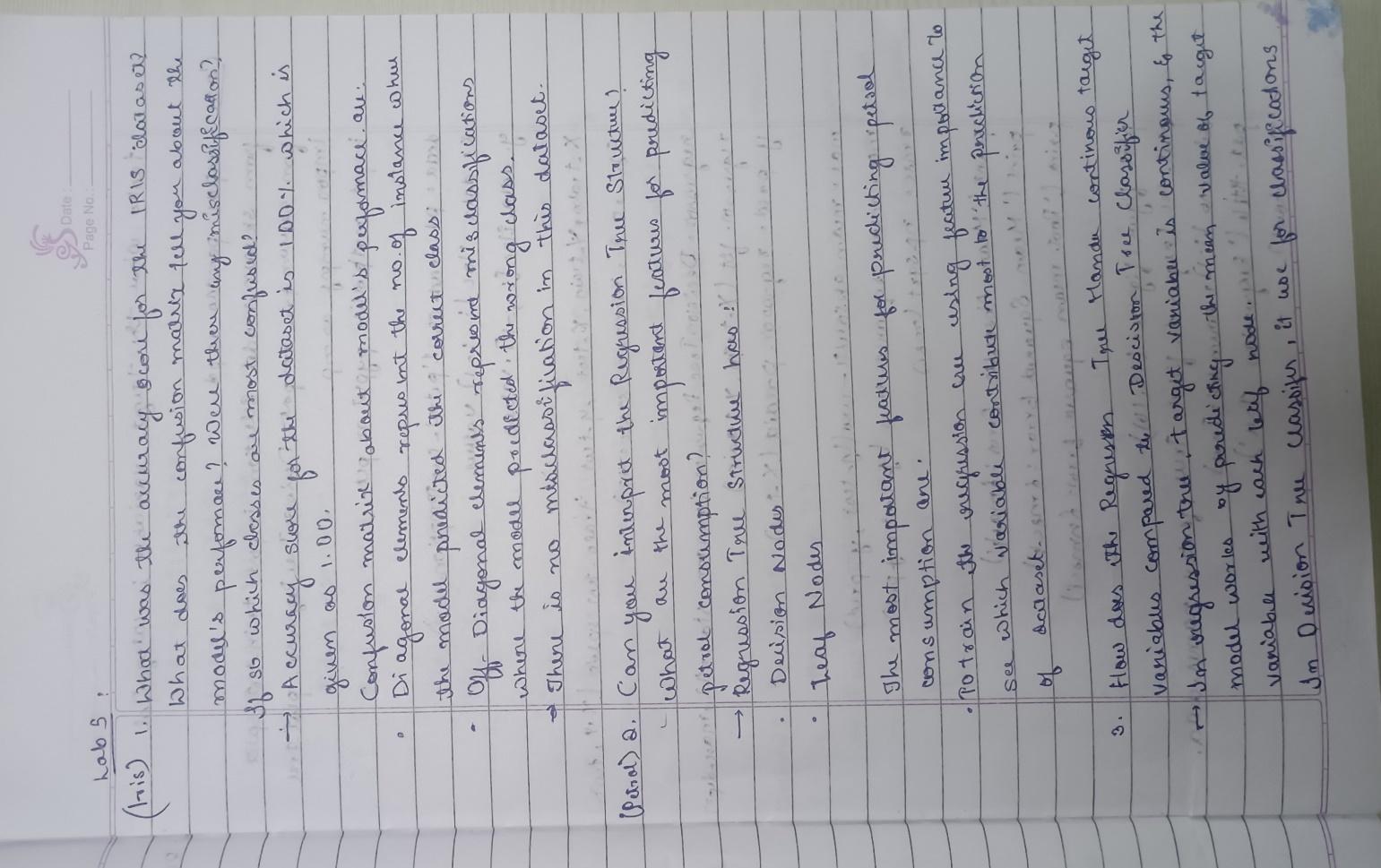
adult\_income\_data\_standardized[adult\_income\_numerical\_cols] = standard\_scaler.fit\_transform(adult\_income\_data\_cleaned[adult\_income\_numerical\_cols])

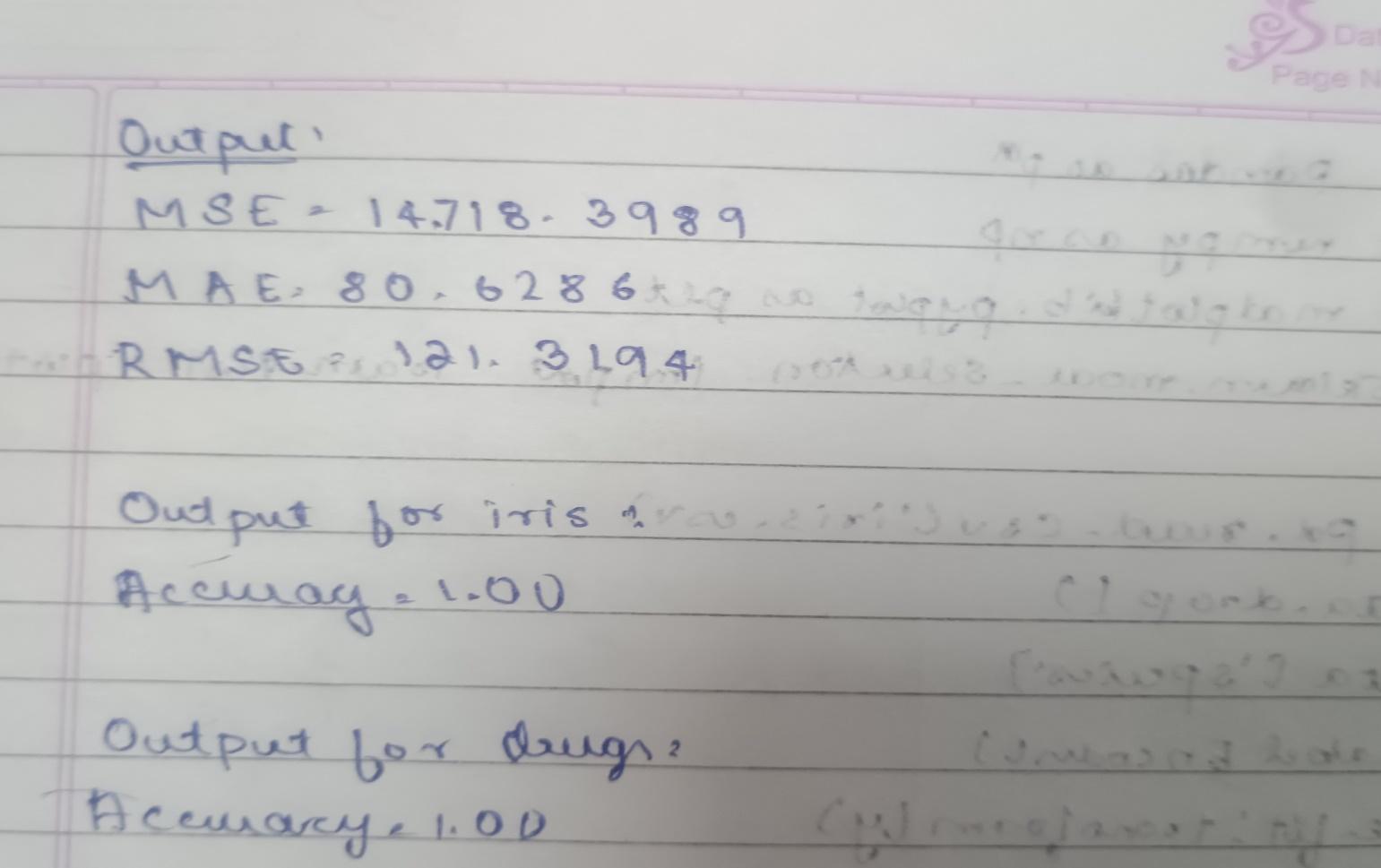
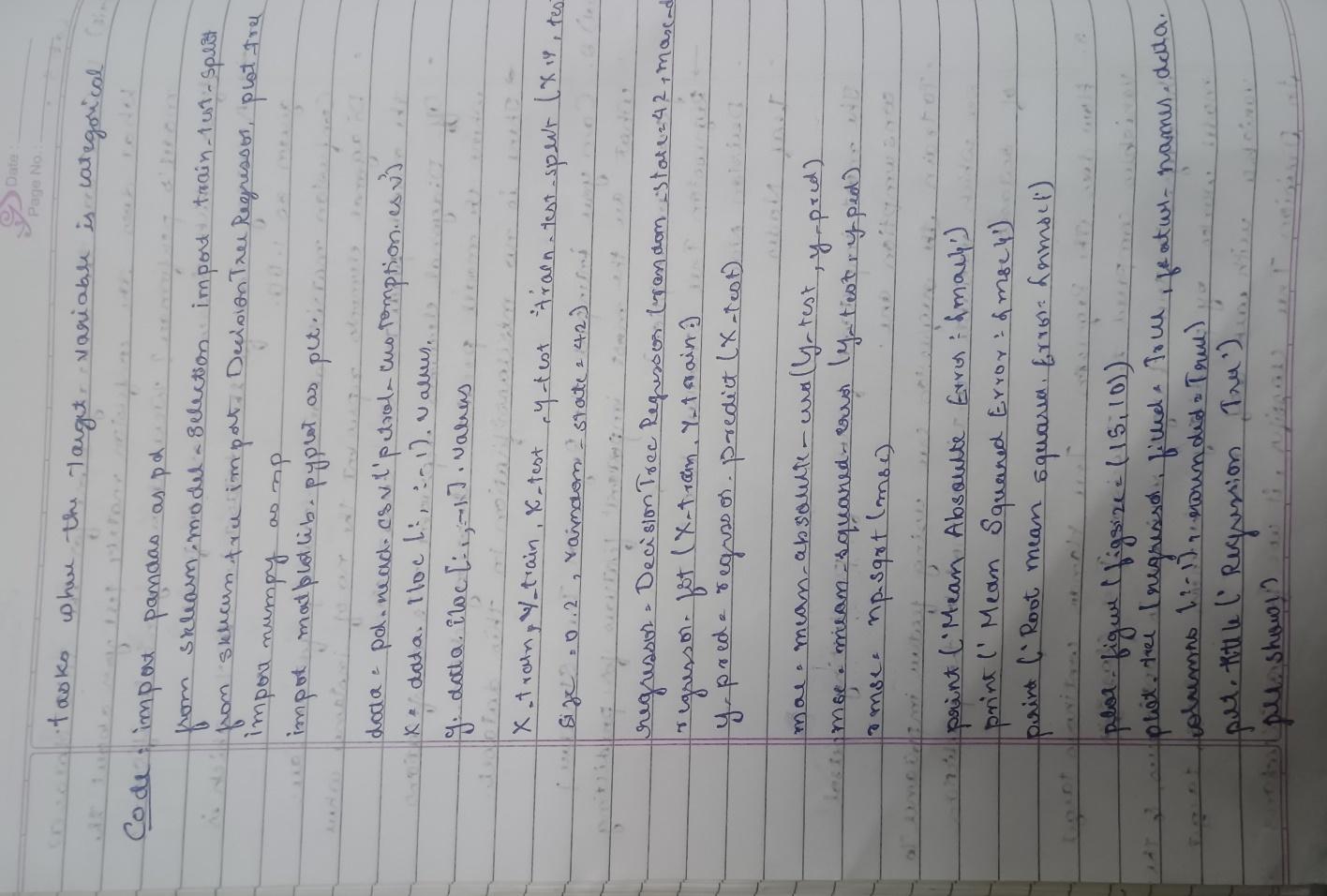
##### Program 3

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

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.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot\_tree

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, mean\_absolute\_error, mean\_squared\_error

from sklearn.preprocessing import LabelEncoder

iris = pd.read\_csv("/content/iris (4).csv")

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

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

X\_iris = iris.iloc[:, :-1]

y\_iris = iris.iloc[:, -1]

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

dtc = DecisionTreeClassifier()

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

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

print("Decision Tree Classification for IRIS Dataset:")

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

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

X\_drug = drug.iloc[:, :-1]

y\_drug = drug.iloc[:, -1]

le = LabelEncoder()

for col in X\_drug.select\_dtypes(include=['object']).columns:

    X\_drug[col] = le.fit\_transform(X\_drug[col])

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

dtc = DecisionTreeClassifier()

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

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

print("\nDecision Tree Classification for Drug Dataset:")

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

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

X\_petrol = petrol.iloc[:, :-1]

y\_petrol = petrol.iloc[:, -1]

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

dtr = DecisionTreeRegressor()

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

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

print("\nDecision Tree Regression for Petrol Consumption:")

print("Mean Absolute Error:", mean\_absolute\_error(y\_test, y\_pred))

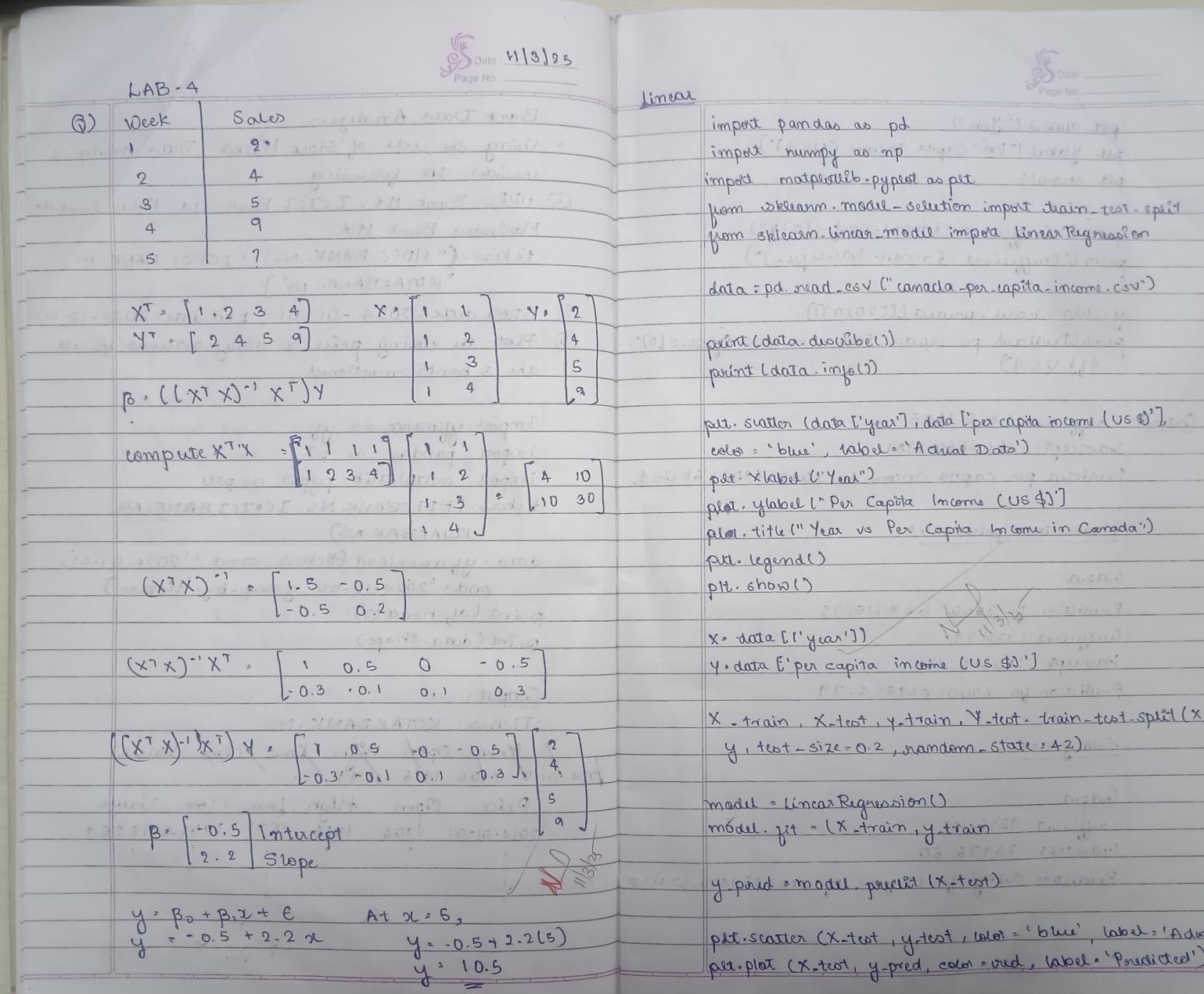
print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

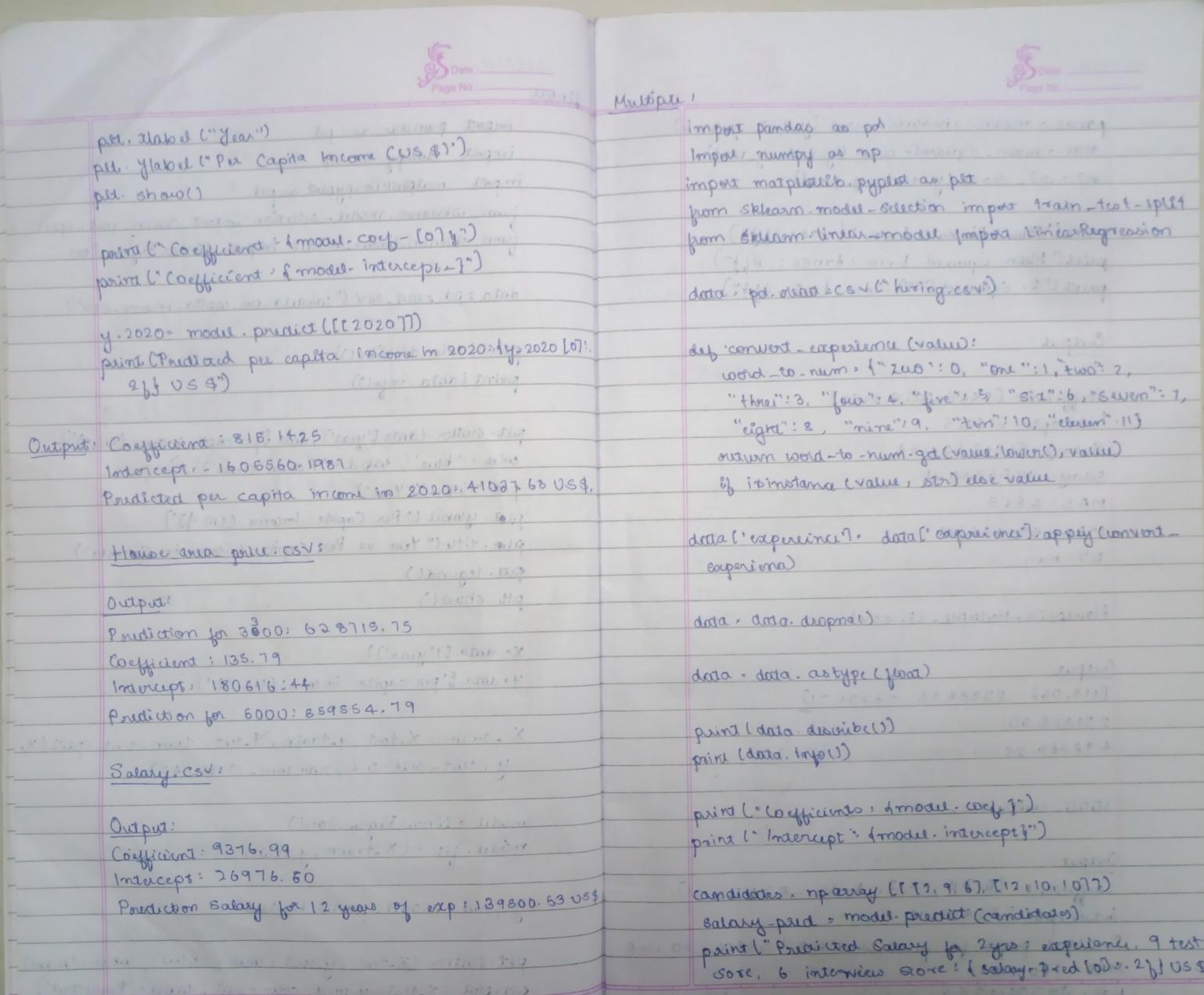
print("Root Mean Squared Error:", np.sqrt(mean\_squared\_error(y\_test, y\_pred)))

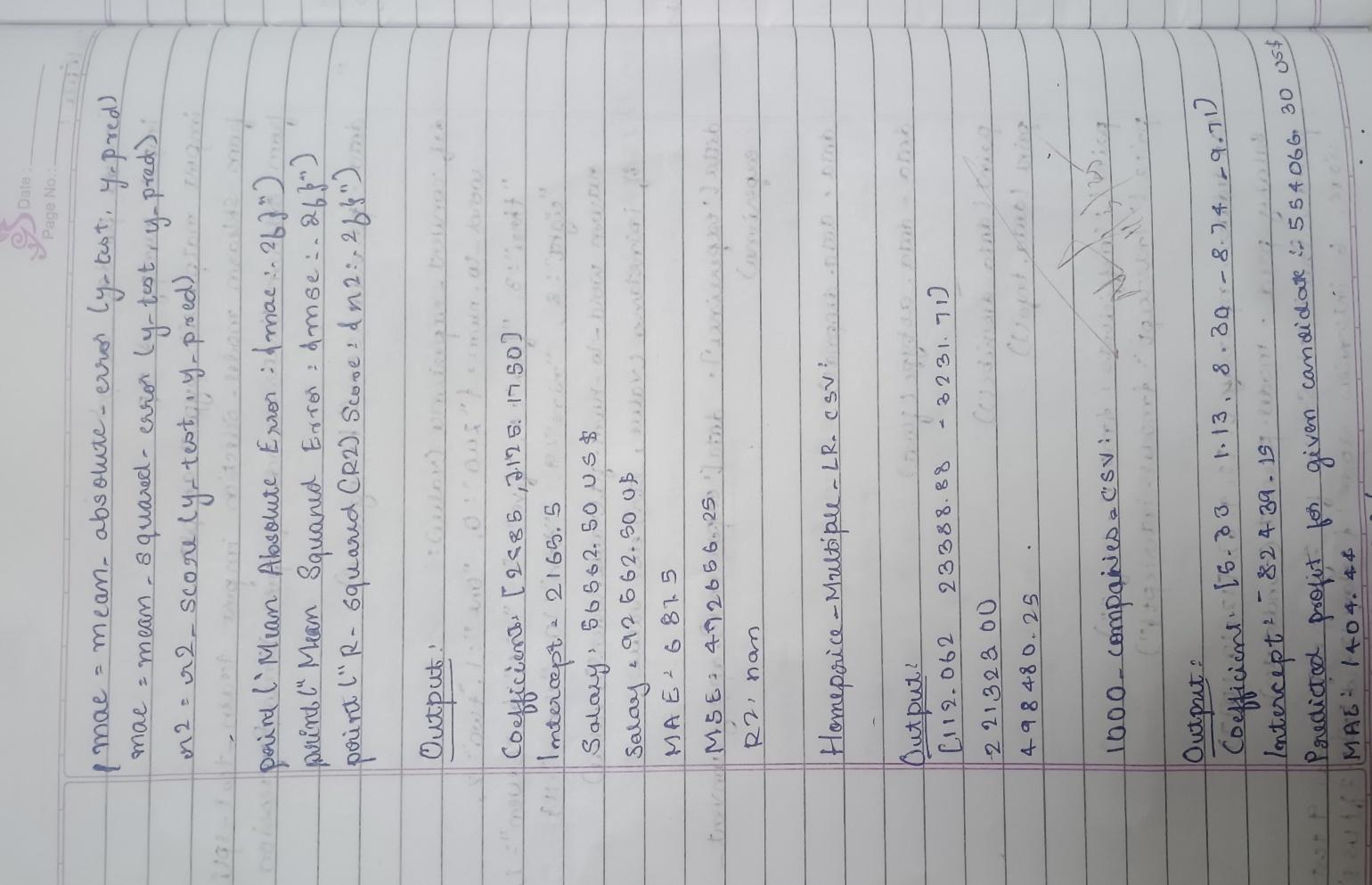
##### Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot







Code:

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import mean\_absolute\_error

import matplotlib.pyplot as plt

hiring\_data = pd.read\_csv('hiring.csv')

print(hiring\_data.head())

hiring\_data = hiring\_data.dropna()

experience\_mapping = {

    'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'seven': 7, 'eight': 8,

    'nine': 9, 'ten': 10, 'eleven': 11, 'twelve': 12, 'thirteen': 13, 'fourteen': 14,

}

hiring\_data['experience'] = hiring\_data['experience'].replace(experience\_mapping)

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

if hiring\_data['experience'].isnull().any():

    print("Warning: There are still non-numeric values in the 'experience' column.")

    hiring\_data = hiring\_data.dropna(subset=['experience'])

X\_hiring = hiring\_data[['experience', 'test\_score(out of 10)', 'interview\_score(out of 10)']]

y\_hiring = hiring\_data['salary($)']

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

regressor\_hiring = LinearRegression()

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

candidate\_1 = np.array([[2, 9, 6]])

candidate\_2 = np.array([[12, 10, 10]])

salary\_1 = regressor\_hiring.predict(candidate\_1)

salary\_2 = regressor\_hiring.predict(candidate\_2)

print(f"Predicted salary for candidate 1 (2 yr experience, 9 test score, 6 interview score): {salary\_1[0]}")

print(f"Predicted salary for candidate 2 (12 yr experience, 10 test score, 10 interview score): {salary\_2[0]}")

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

print(companies\_data.head())

companies\_data = companies\_data.dropna()

label\_encoder = LabelEncoder()

companies\_data['State'] = label\_encoder.fit\_transform(companies\_data['State'])

X\_companies = companies\_data[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]

y\_companies = companies\_data['Profit']

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

regressor\_companies = LinearRegression()

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

input\_data = np.array([[91694.48, 515841.3, 11931.24, label\_encoder.transform(['Florida'])[0]]])

predicted\_profit = regressor\_companies.predict(input\_data)

print(f"Predicted profit for the given inputs (Florida State): {predicted\_profit[0]}")

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

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

print(f"Mean Absolute Error for Salary Prediction: {mae\_hiring}")

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

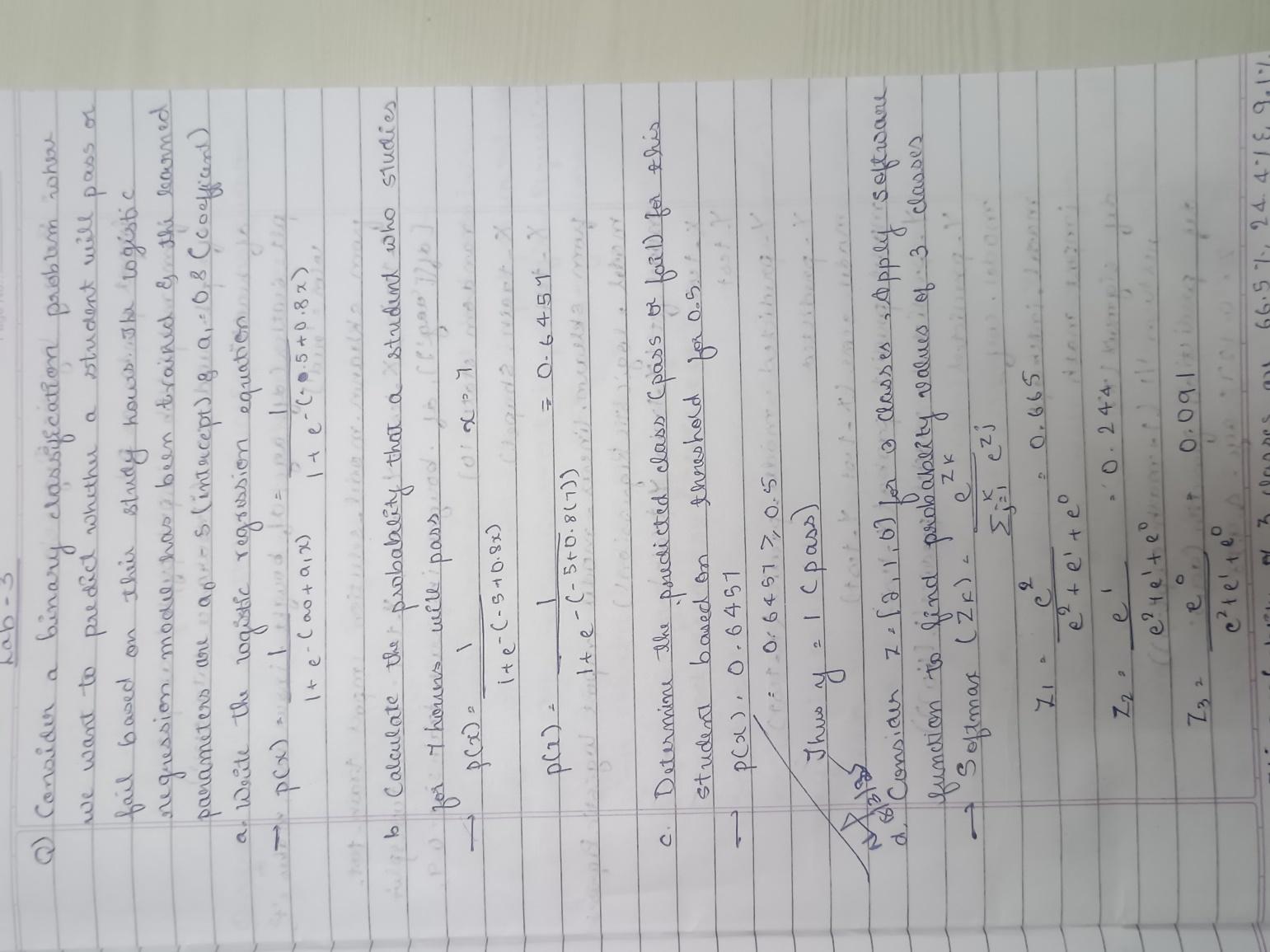
mae\_companies = mean\_absolute\_error(y\_test\_companies, y\_pred\_companies)

print(f"Mean Absolute Error for Profit Prediction: {mae\_companies}")

##### Program 5

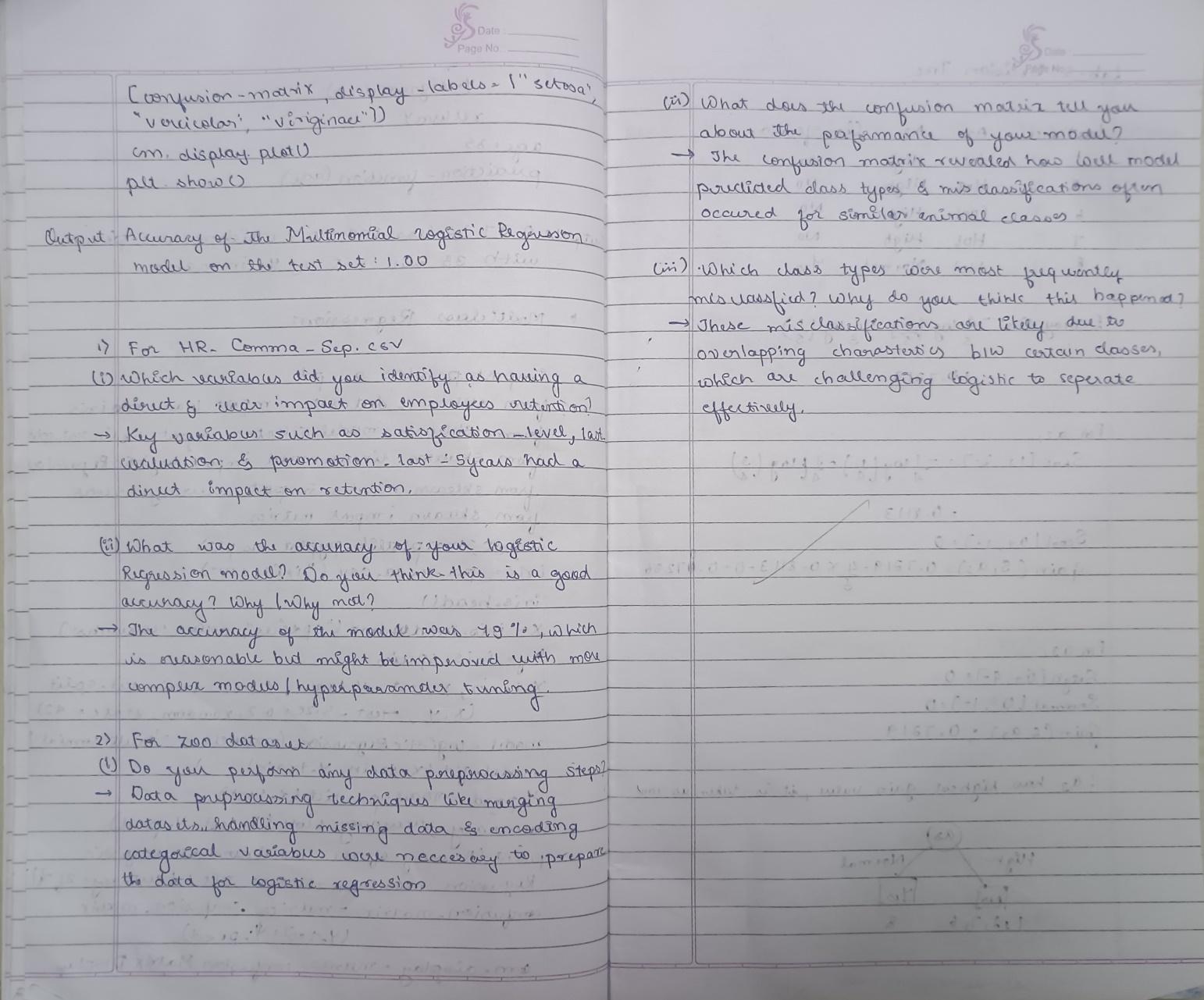
Build Logistic Regression Model for a given dataset

Screenshot









Code:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

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

file\_path = 'HR\_comma\_sep.csv'

data = pd.read\_csv(file\_path)

print(data.info())

print(data.head())

print(data.describe())

plt.figure(figsize=(8, 5))

sns.countplot(x='salary', hue='left', data=data)

plt.title('Impact of Salary on Employee Retention')

plt.xlabel('Salary')

plt.ylabel('Count')

plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])

plt.show()

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

sns.countplot(x='Department', hue='left', data=data)

plt.title('Impact of Department on Employee Retention')

plt.xlabel('Department')

plt.ylabel('Count')

plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])

plt.xticks(rotation=45)

plt.show()

data\_encoded = pd.get\_dummies(data, columns=['salary', 'Department'], drop\_first=True)

print(data\_encoded.info())

X = data\_encoded.drop('left', axis=1)

y = data\_encoded['left']

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

scaler = StandardScaler()

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

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

logreg = LogisticRegression(max\_iter=1000)

logreg.fit(X\_train\_scaled, y\_train)

y\_pred = logreg.predict(X\_test\_scaled)

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

print(f"Accuracy of the Logistic Regression Model: {accuracy \* 100:.2f}%")

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

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['Stayed', 'Left'], yticklabels=['Stayed', 'Left'])

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

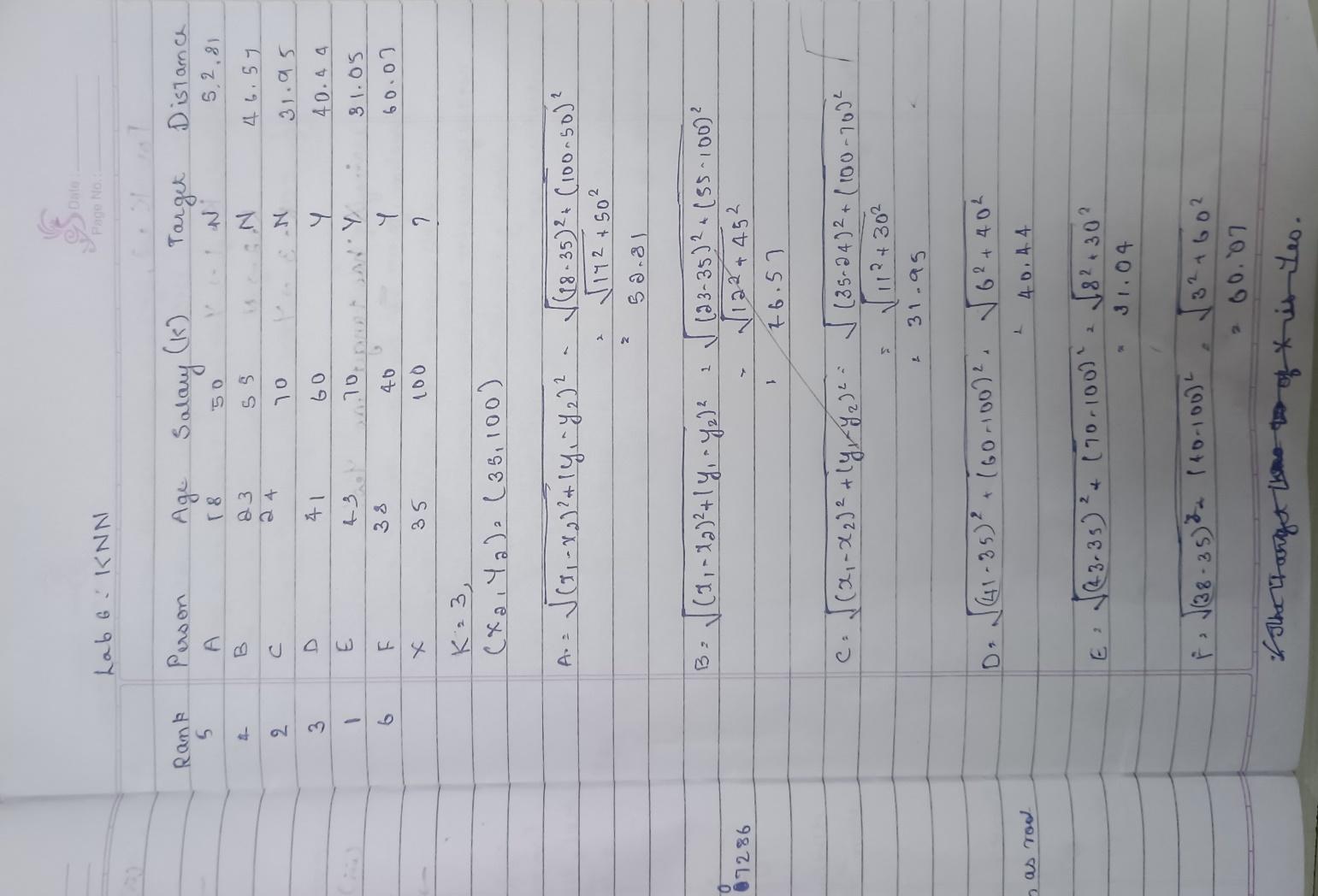
plt.ylabel('Actual')

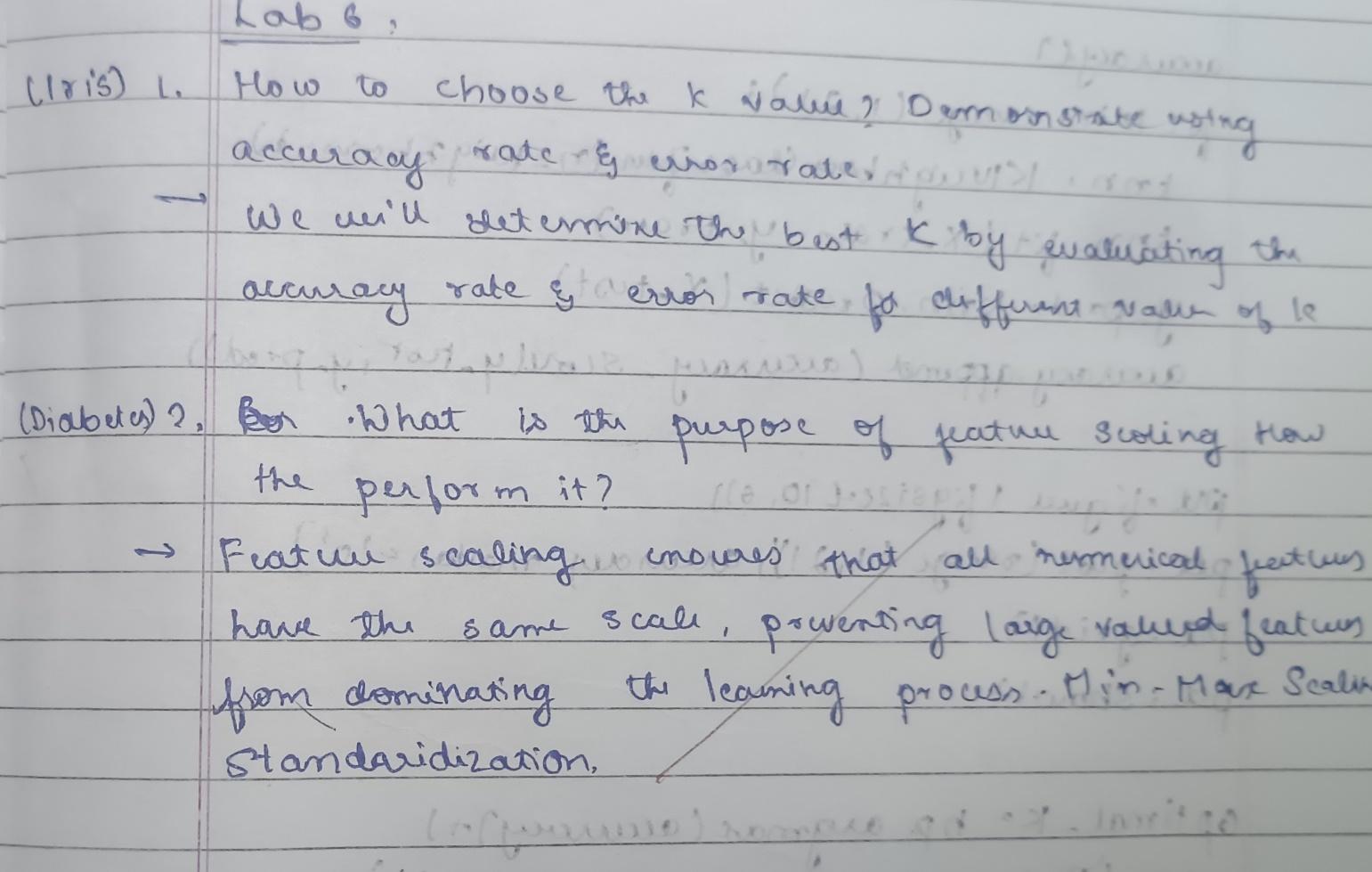
plt.show()

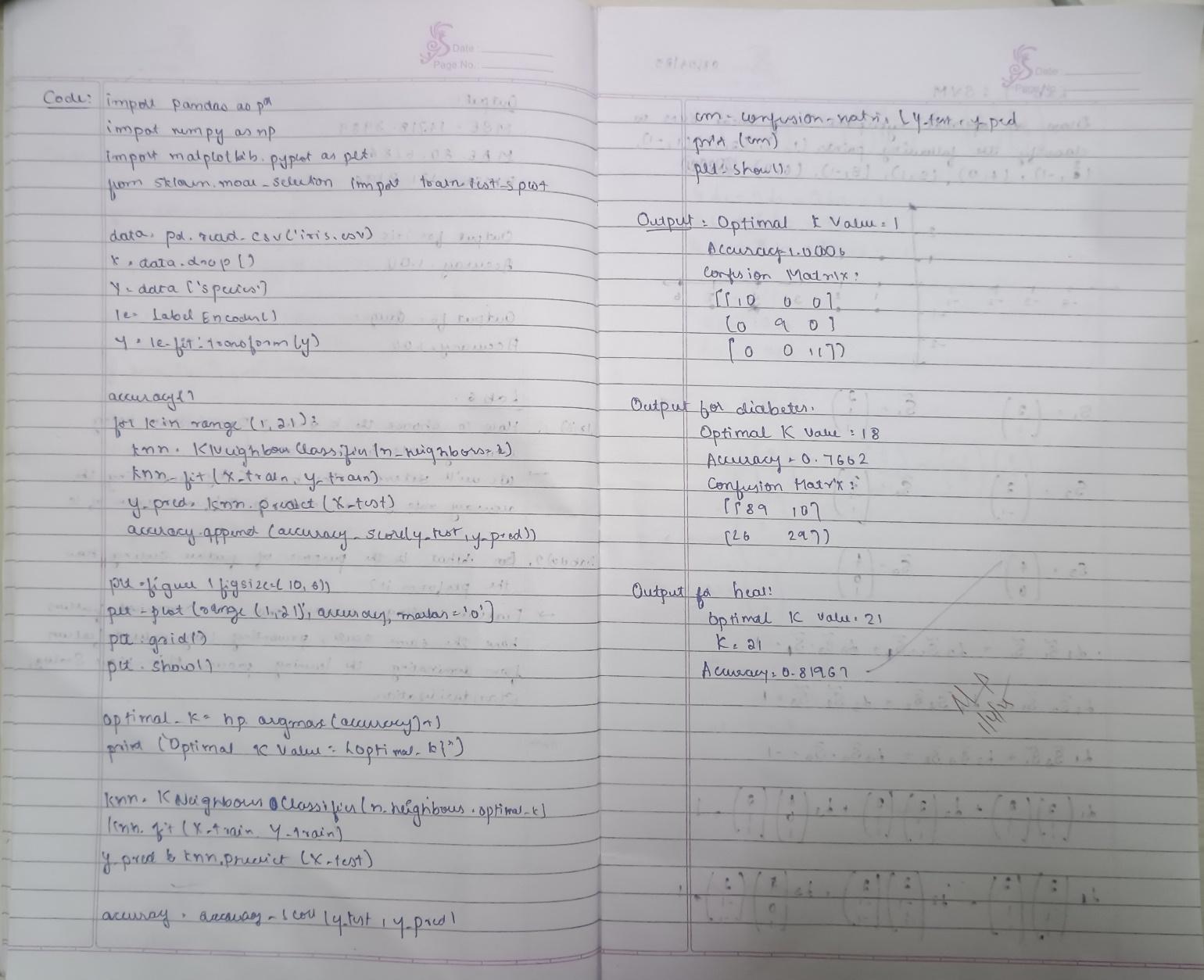
Program 6

Build KNN Classification model for a given dataset.

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.neighbors import KNeighborsClassifier

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

import matplotlib.pyplot as plt

import seaborn as sns

iris\_df = pd.read\_csv('/content/iris (3).csv')

print(iris\_df.head())

X\_iris = iris\_df.drop(columns=['species'])

y\_iris = iris\_df['species']

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

scaler = StandardScaler()

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

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

knn\_iris = KNeighborsClassifier(n\_neighbors=3)

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

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

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

print(f"Accuracy on Iris test data: {accuracy\_iris \* 100:.2f}%")

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

sns.heatmap(cm\_iris, annot=True, fmt="d", cmap="Blues", xticklabels=knn\_iris.classes\_, yticklabels=knn\_iris.classes\_)

plt.title("Confusion Matrix for Iris Dataset")

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

print("Classification Report for Iris Dataset:")

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

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

print(diabetes\_df.head())

X\_diabetes = diabetes\_df.drop(columns=['Outcome'])

y\_diabetes = diabetes\_df['Outcome']

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

scaler = StandardScaler()

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

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

knn\_diabetes = KNeighborsClassifier(n\_neighbors=5)

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

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

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

print(f"Accuracy on Diabetes test data: {accuracy\_diabetes \* 100:.2f}%")

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

sns.heatmap(cm\_diabetes, annot=True, fmt="d", cmap="Blues", xticklabels=knn\_diabetes.classes\_, yticklabels=knn\_diabetes.classes\_)

plt.title("Confusion Matrix for Diabetes Dataset")

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

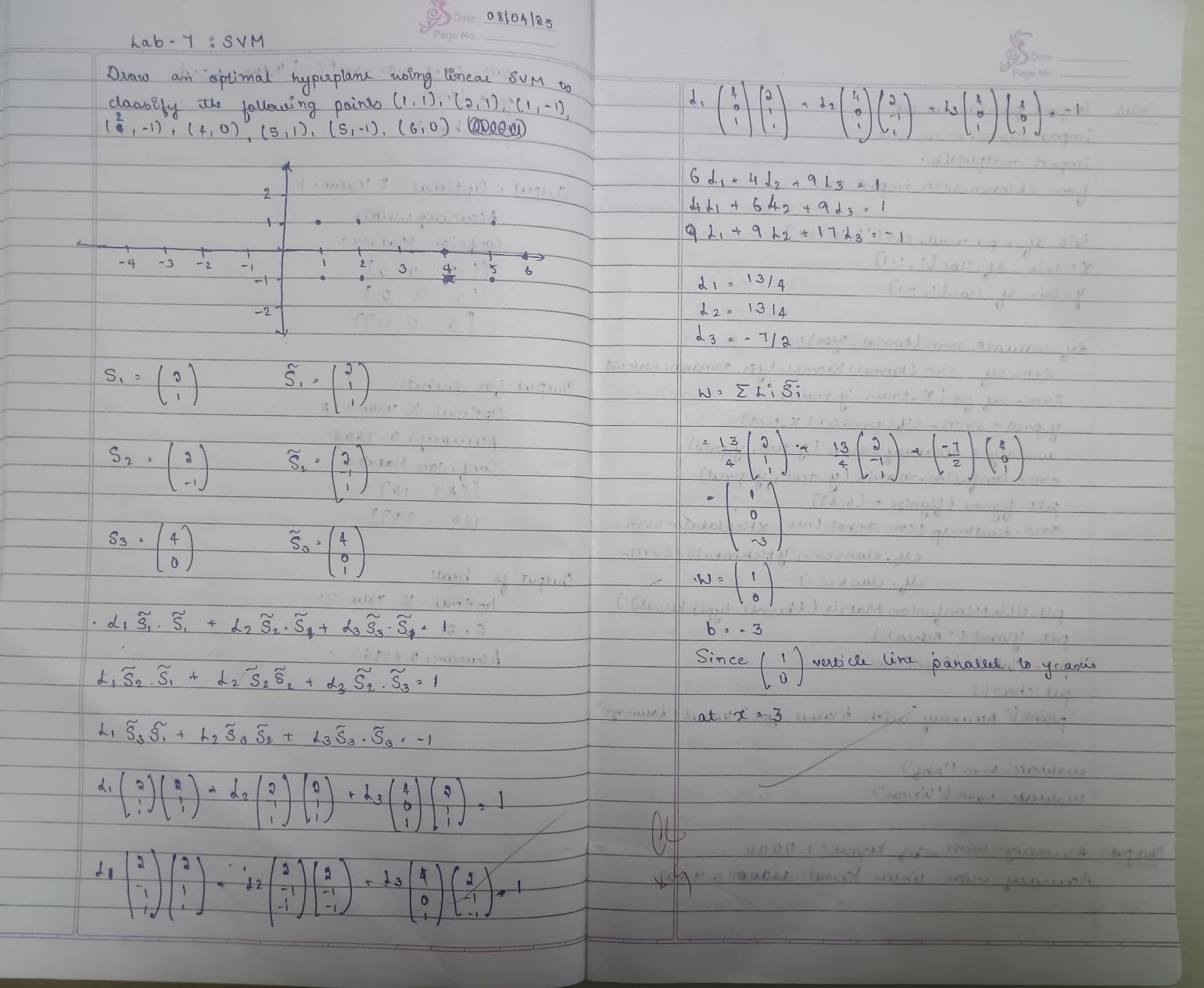
print("Classification Report for Diabetes Dataset:")

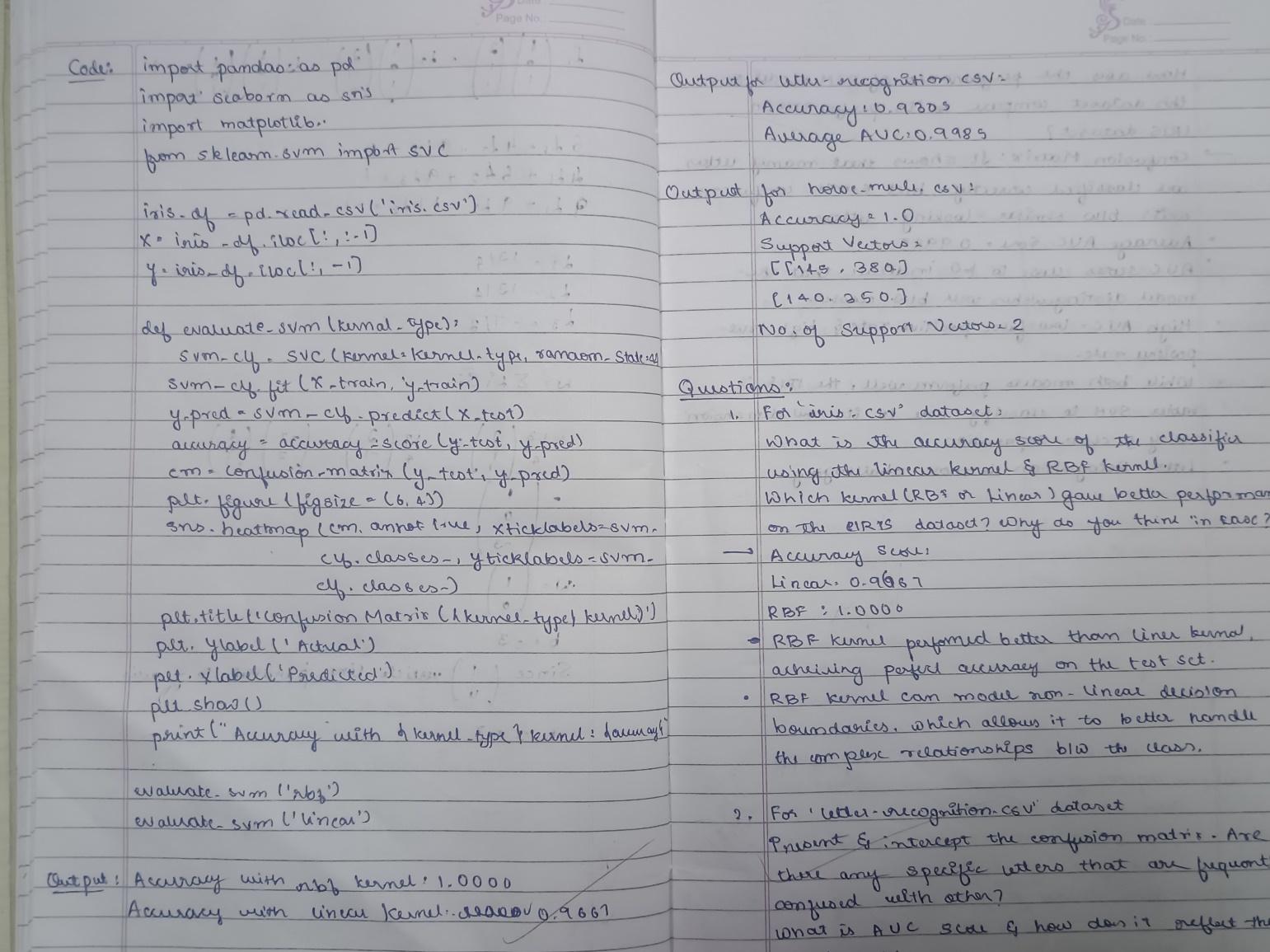
print(classification\_report(y\_test\_diabetes, y\_pred\_diabetes))

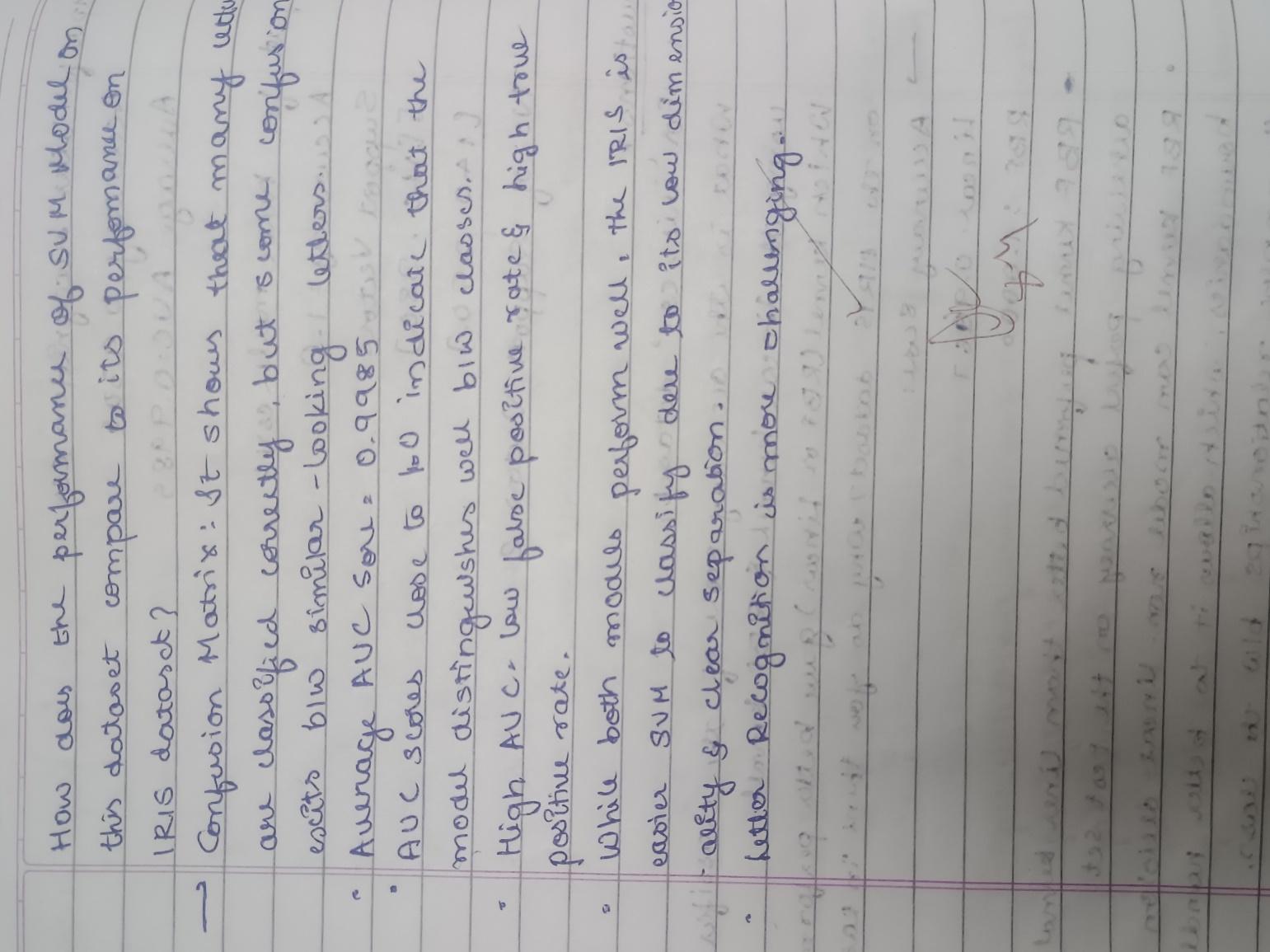
##### Program 7

Build Support vector machine model for a given dataset

Screenshot







Code:

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\_auc\_score, roc\_curve

from sklearn.preprocessing import LabelEncoder, label\_binarize

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

df = pd.read\_csv("/content/letter-recognition.csv")

top\_classes = df['letter'].value\_counts().head(5).index.tolist()

df = df[df['letter'].isin(top\_classes)]

X = df.iloc[:, 1:]

y = df.iloc[:, 0]

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

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

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

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

svm\_model = SVC(kernel='linear', probability=True)

svm\_model.fit(X\_train, y\_train.argmax(axis=1))

y\_score = svm\_model.predict\_proba(X\_test)

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

y\_true = y\_test\_bin.argmax(axis=1)

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

print("Confusion Matrix:\n", confusion\_matrix(y\_true, y\_pred))

plt.figure()

for i in range(n\_classes):

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

    auc = roc\_auc\_score(y\_test\_bin[:, i], y\_score[:, i])

    plt.plot(fpr, tpr, label=f"{label\_encoder.inverse\_transform([i])[0]} AUC={auc:.2f}")

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

plt.title("ROC Curve (Top 5 Classes)")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend(loc="lower right")

plt.tight\_layout()

plt.show()

macro\_auc = roc\_auc\_score(y\_test\_bin, y\_score, average="macro")

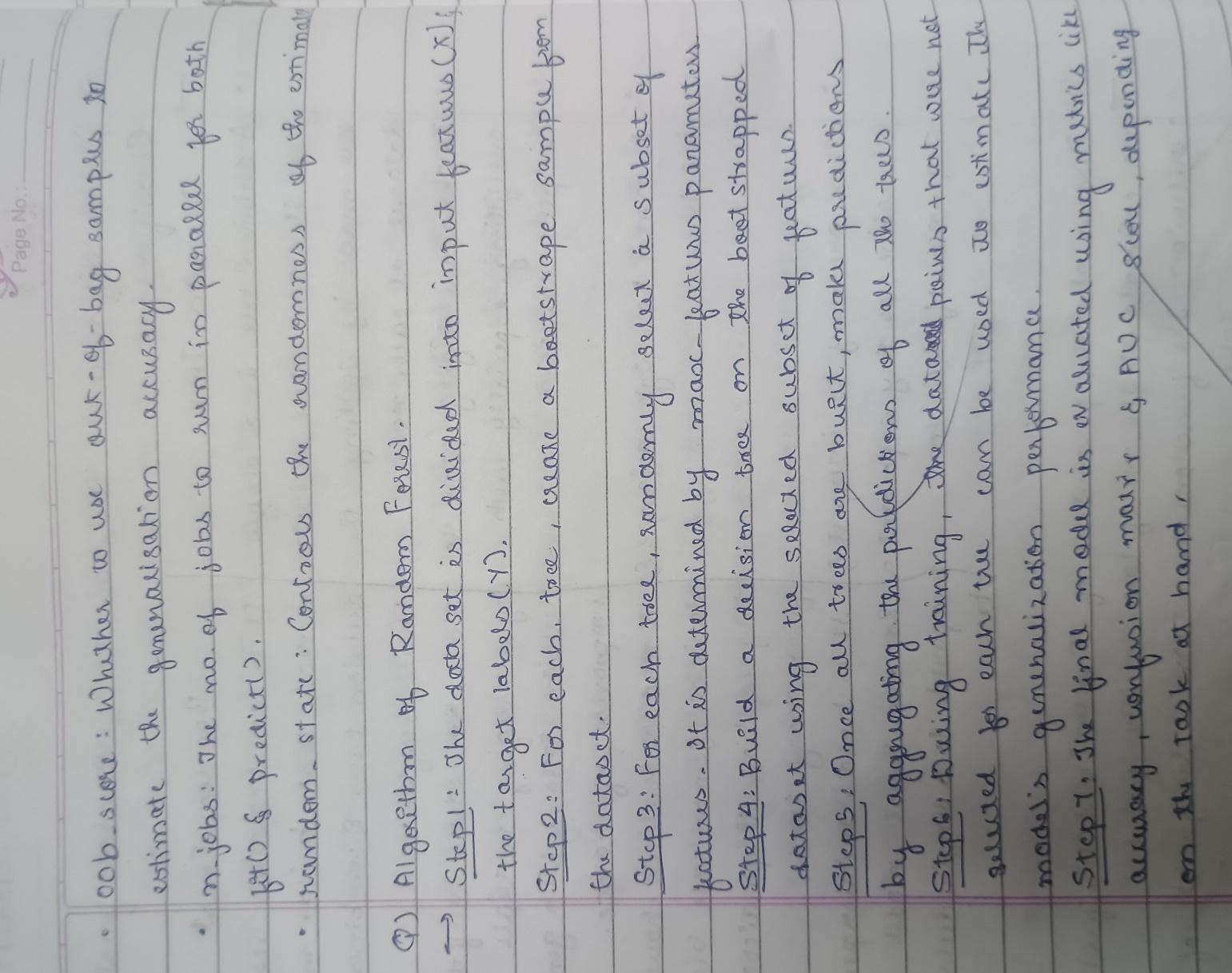
print("Macro AUC Score:", macro\_auc)

##### 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, confusion\_matrix

from sklearn import preprocessing

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

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

y = df.iloc[:, -1]

for column in X.columns:

    if X[column].dtype == 'object':

      le = preprocessing.LabelEncoder()

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

if y.dtype == 'object':

  le = preprocessing.LabelEncoder()

  y = le.fit\_transform(y)

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

rf\_classifier = RandomForestClassifier(random\_state=42)

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

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

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

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

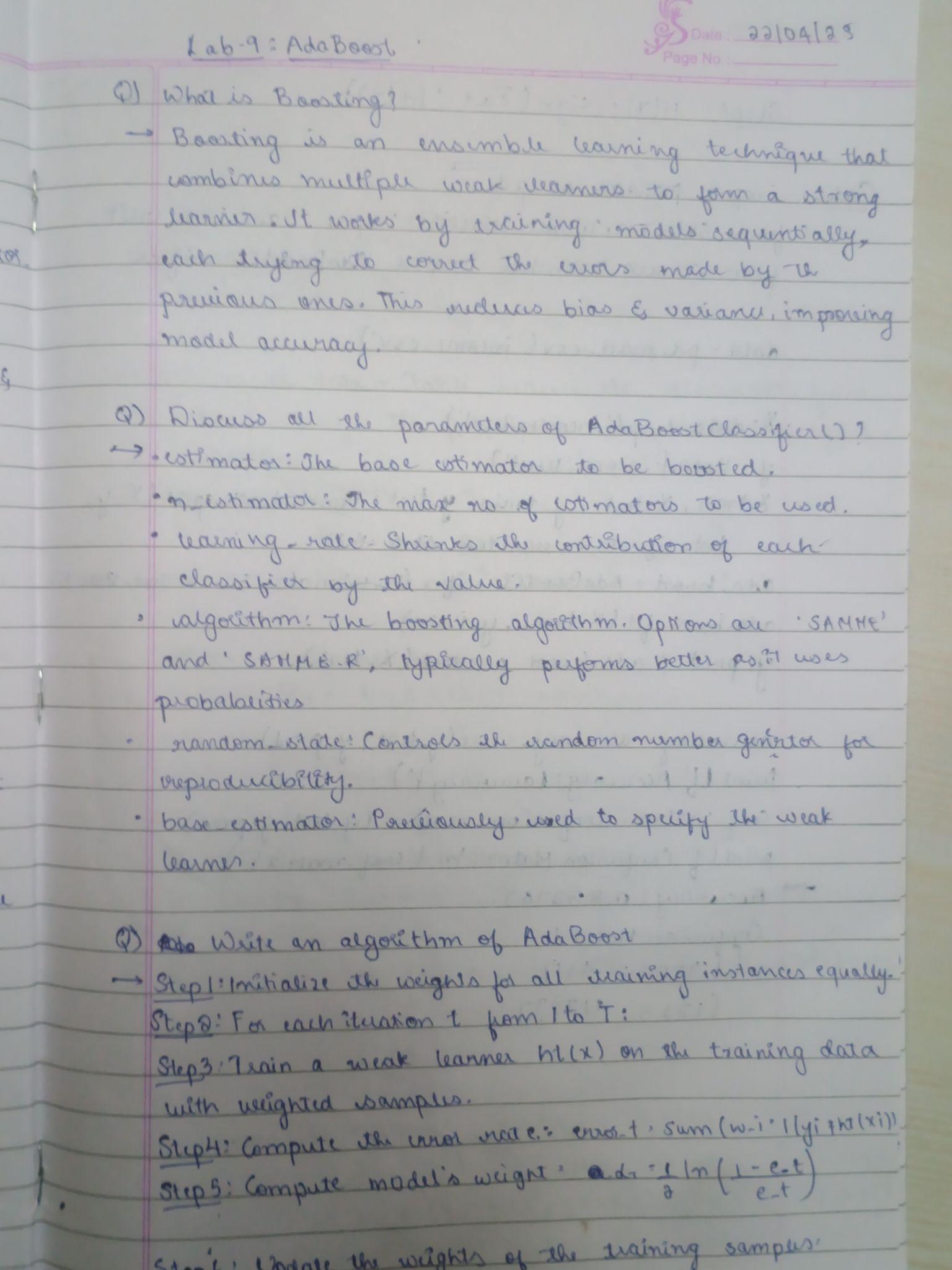
print(f"Accuracy: {accuracy}")

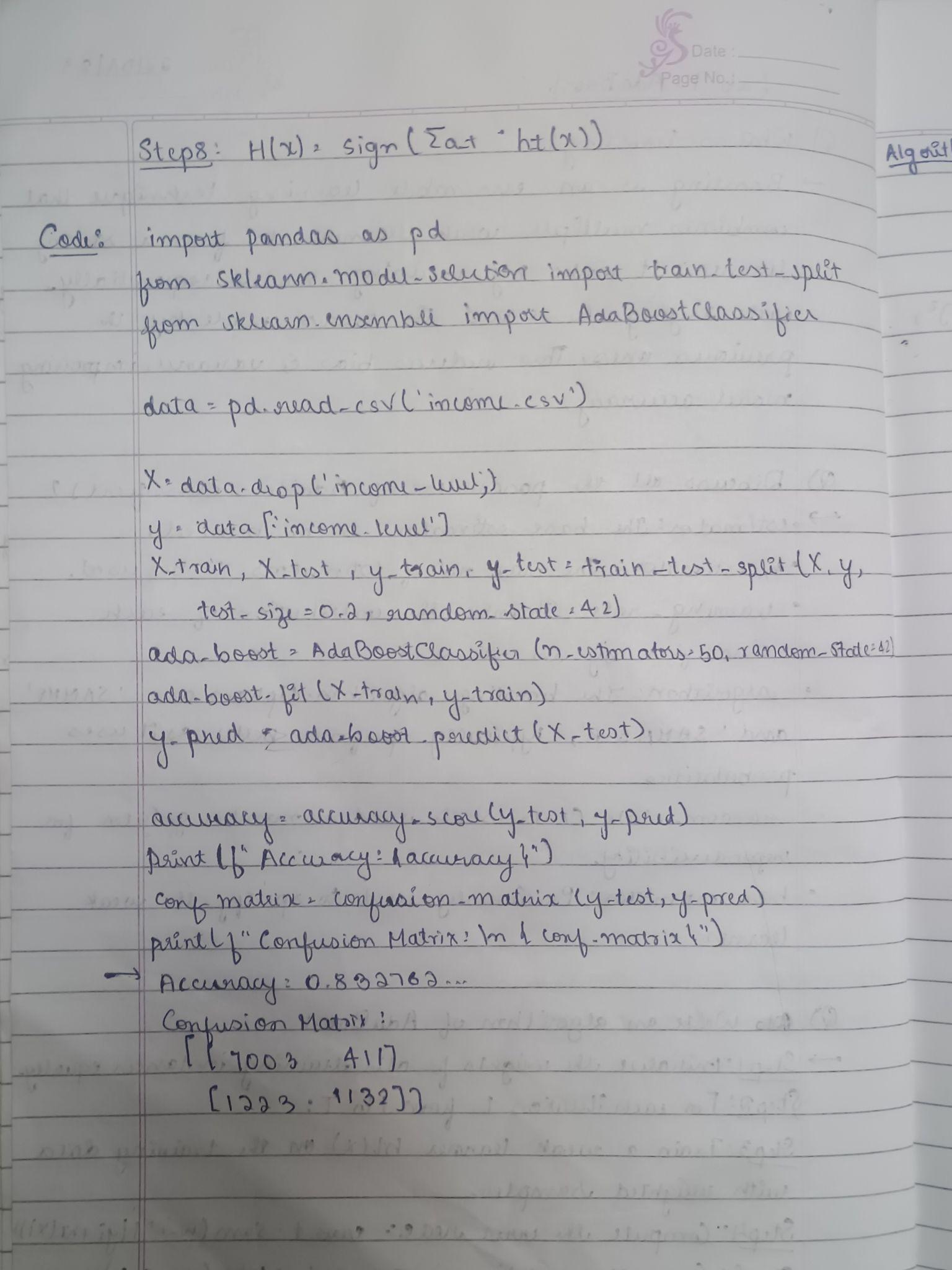
print(f"Confusion Matrix:\n{conf\_matrix}")

##### Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot





Code:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import accuracy\_score

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)

results = []

n\_estimators\_list = [10, 50, 100]

learning\_rates = [0.01, 0.1, 1]

for n in n\_estimators\_list:

    for lr in learning\_rates:

        tree\_base = DecisionTreeClassifier(max\_depth=1)

        model = AdaBoostClassifier(estimator=tree\_base, n\_estimators=n, learning\_rate=lr, random\_state=42)

        model.fit(X\_train, y\_train)

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

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

        results.append({

            'Base': 'DecisionTree',

            'n\_estimators': n,

            'learning\_rate': lr,

            'Accuracy': acc

        })

for n in n\_estimators\_list:

    for lr in learning\_rates:

        log\_reg\_base = LogisticRegression(max\_iter=1000)

        model = AdaBoostClassifier(estimator=log\_reg\_base, n\_estimators=n, learning\_rate=lr, random\_state=42)

        model.fit(X\_train, y\_train)

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

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

        results.append({

            'Base': 'LogisticRegression',

            'n\_estimators': n,

            'learning\_rate': lr,

            'Accuracy': acc

        })

results\_df = pd.DataFrame(results)

print(results\_df)

import seaborn as sns

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

sns.barplot(x='n\_estimators', y='Accuracy', hue='Base', data=results\_df, ci=None)

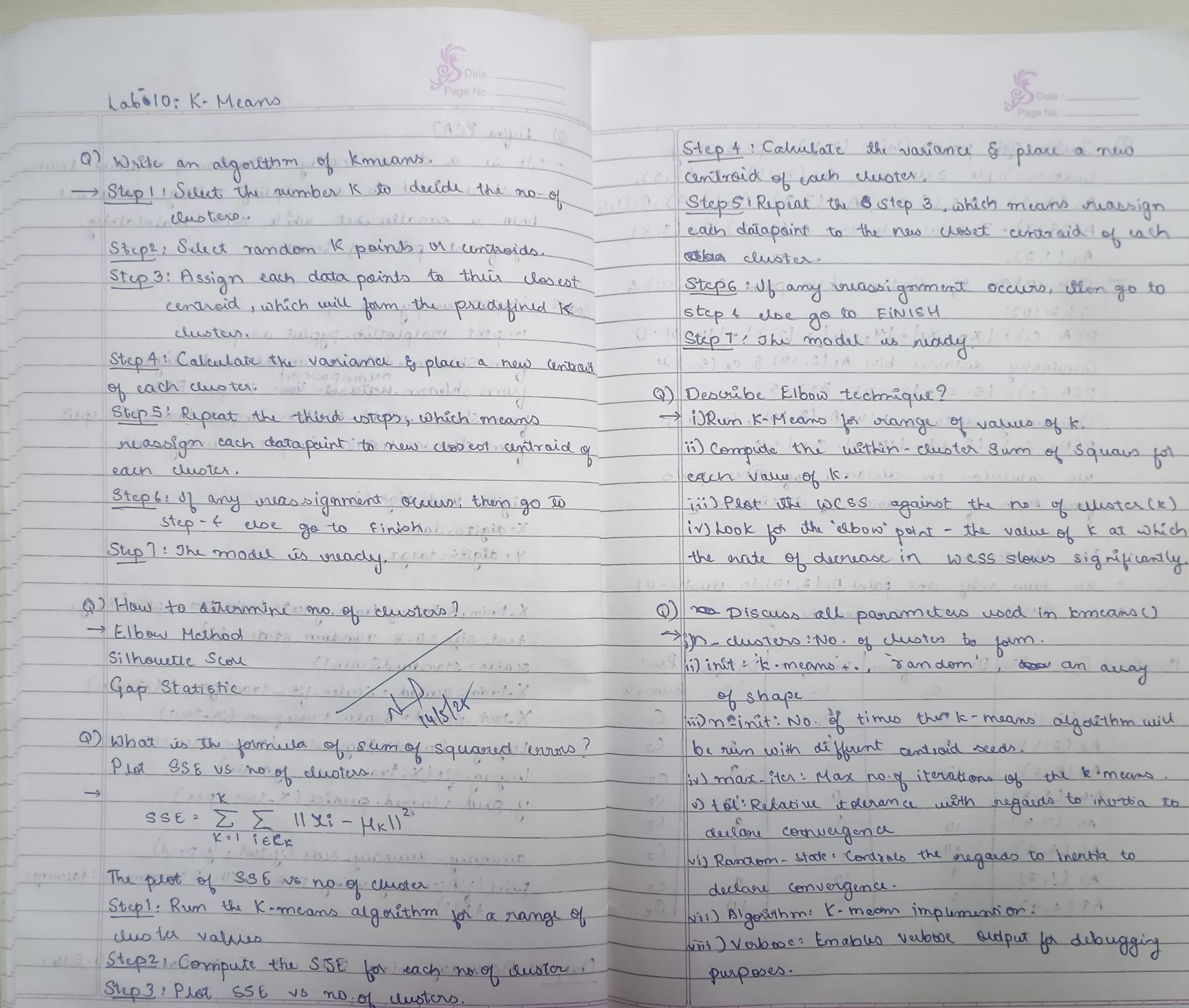
plt.title('AdaBoost Accuracy with Different Estimators and n\_estimators')

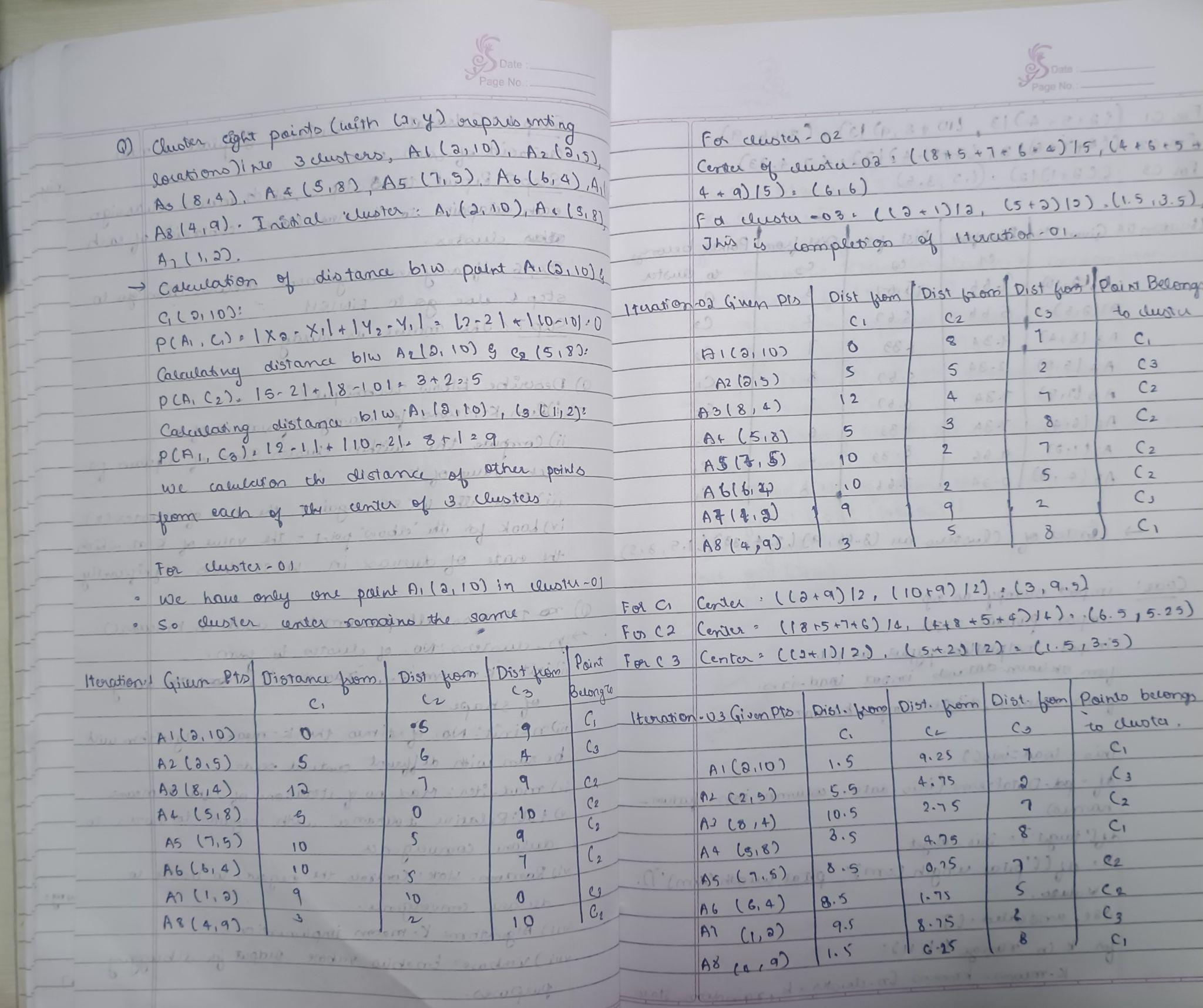
plt.show()

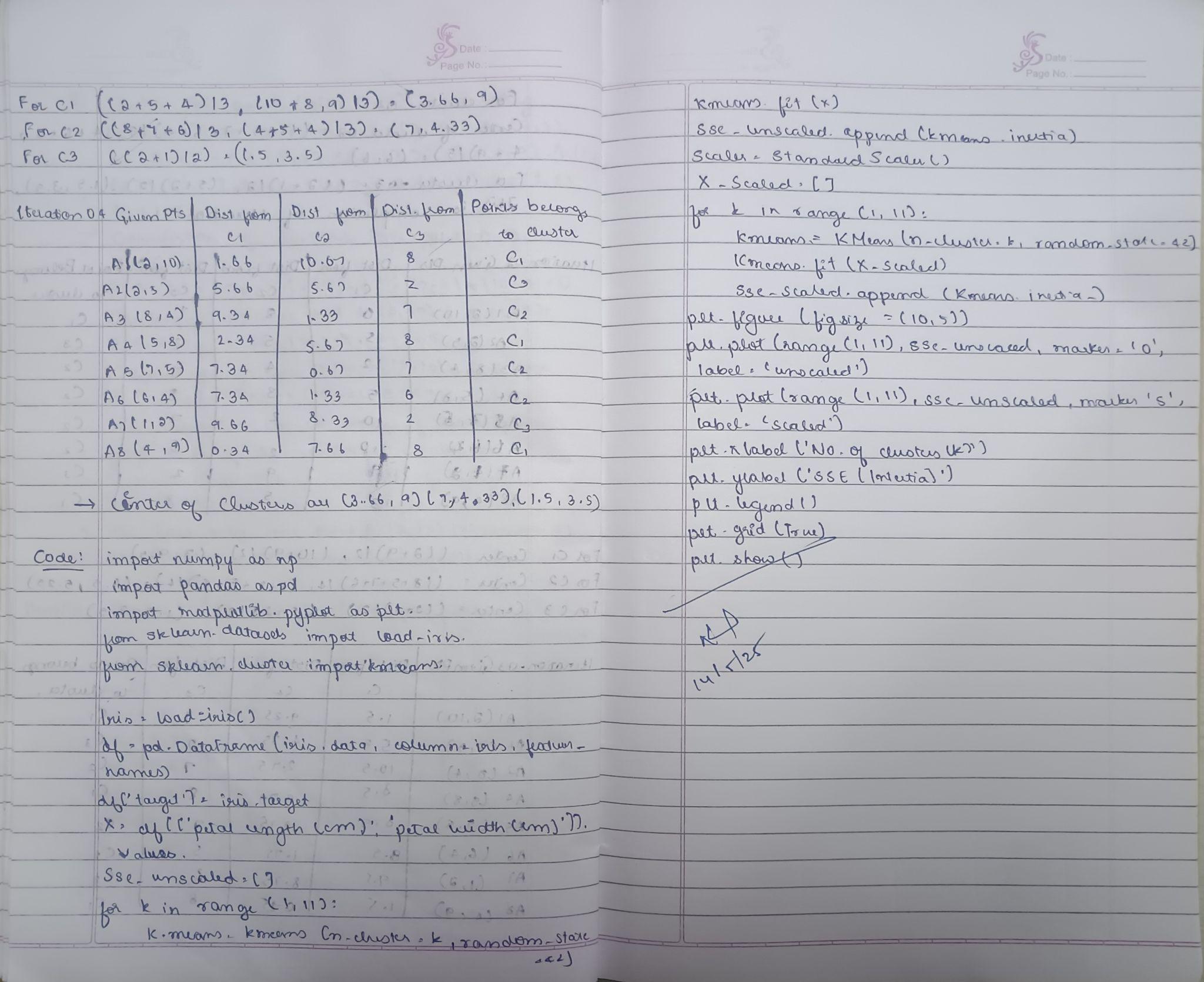
##### Program 10

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

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

data = {

    'Name': [f'Person\_{i+1}' for i in range(50)],

    'Age': np.random.randint(18, 70, size=50),

    'Income': np.random.randint(20000, 120000, size=50)

}

df = pd.DataFrame(data)

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

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

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

scaler = StandardScaler()

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

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

sse = []

k\_range = range(1, 11)

for k in k\_range:

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

    kmeans.fit(X\_train)

    sse.append(kmeans.inertia\_)

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

plt.title('SSE vs Number of Clusters')

plt.xlabel('Number of Clusters')

plt.ylabel('Sum of Squared Errors (SSE)')

plt.show()

optimal\_k = 3

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

kmeans.fit(X\_train)

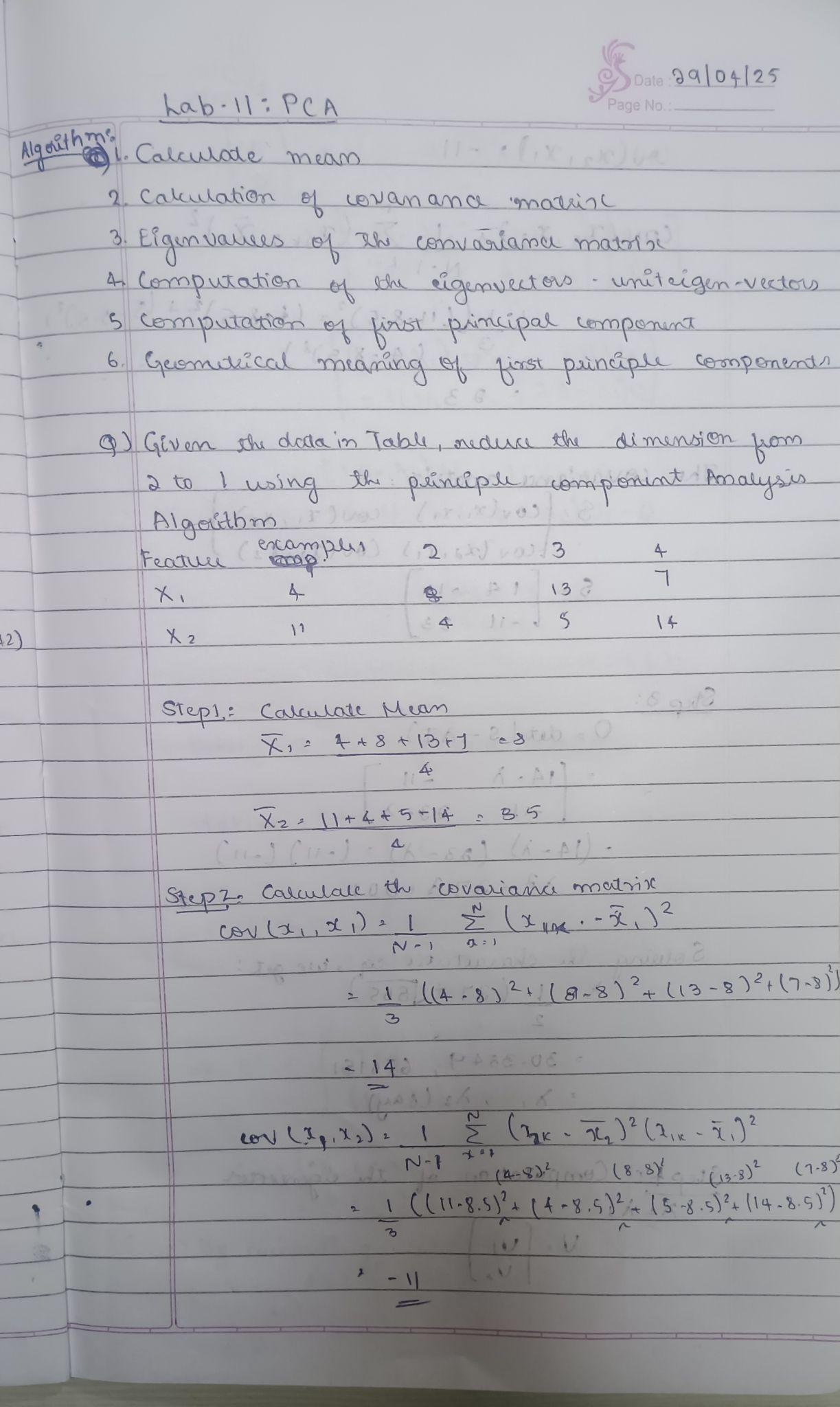
y\_pred = kmeans.predict(X\_test)

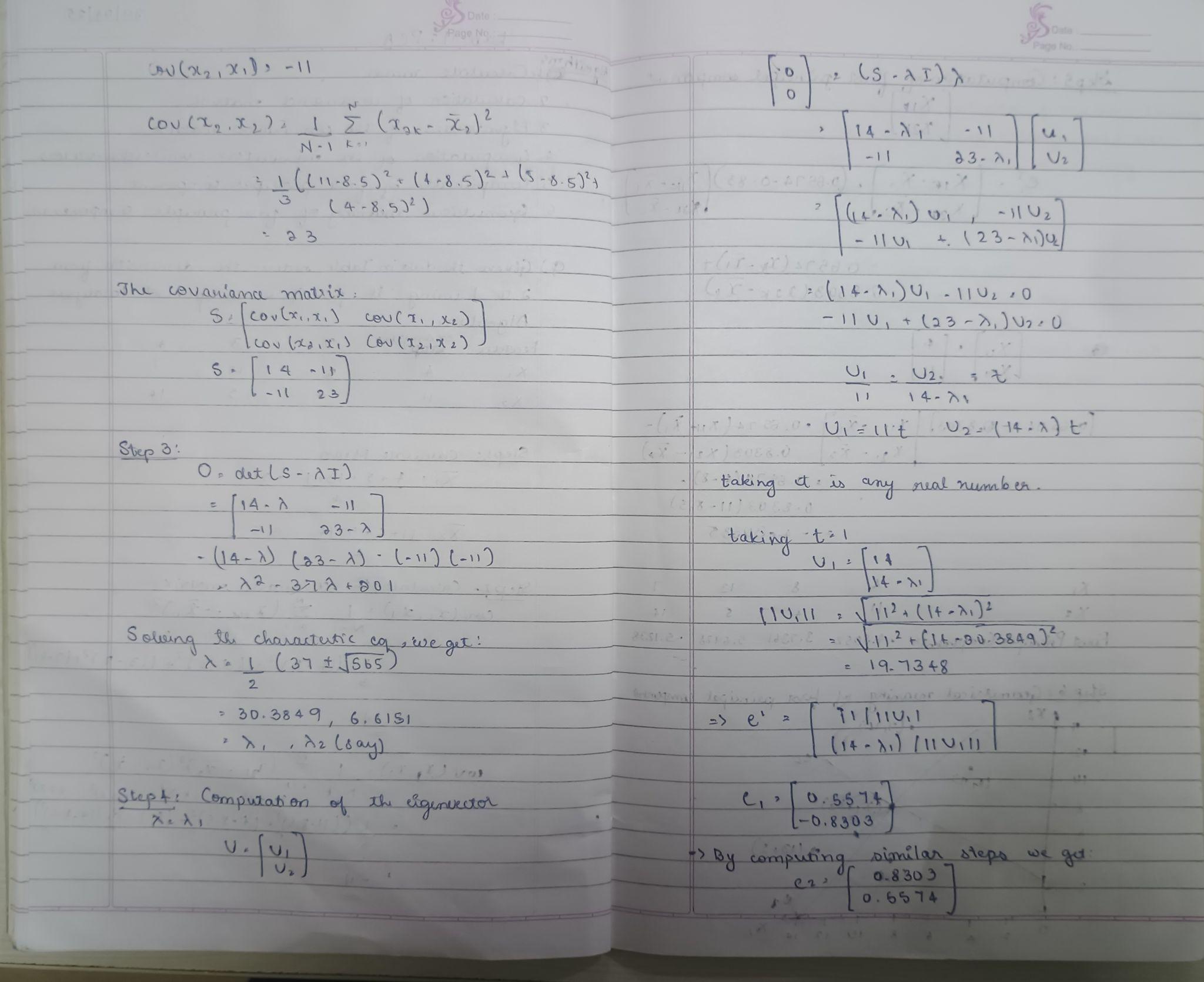
print(f'Predicted Clusters for Test Data: {y\_pred}')

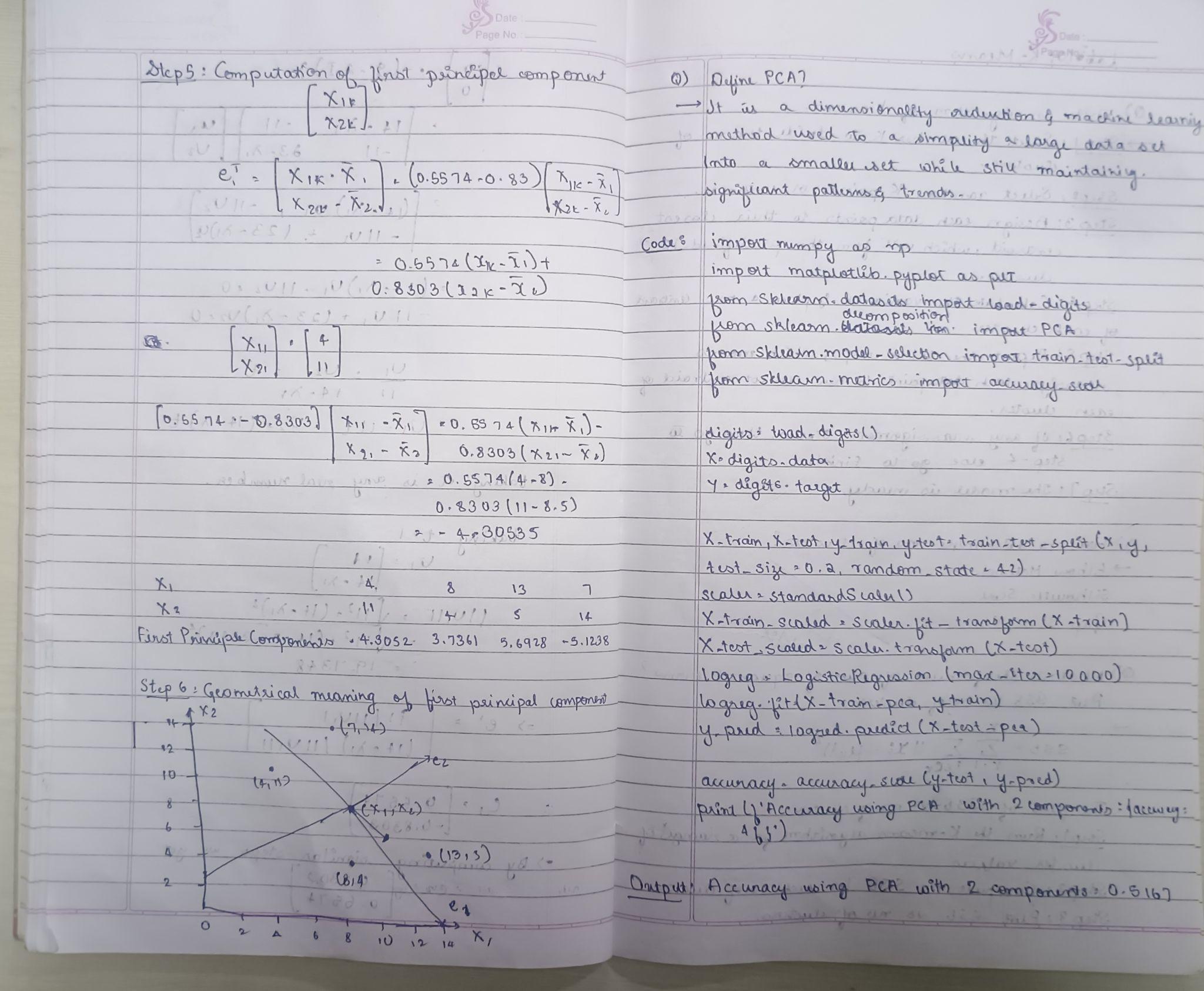
##### Program 11

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

Screenshot







Code:

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, LabelEncoder

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

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn.decomposition import PCA

from scipy import stats

df = pd.read\_csv('heart (2).csv')

z\_scores = np.abs(stats.zscore(df.select\_dtypes(include=[np.number])))

df\_no\_outliers = df[(z\_scores < 3).all(axis=1)]

df\_cleaned = df\_no\_outliers.copy()

for col in df\_cleaned.select\_dtypes(include='object').columns:

    df\_cleaned[col] = LabelEncoder().fit\_transform(df\_cleaned[col])

X = df\_cleaned.drop('HeartDisease', axis=1)

y = df\_cleaned['HeartDisease']

scaler = StandardScaler()

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

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

models = {

    "Logistic Regression": LogisticRegression(max\_iter=1000),

    "Random Forest": RandomForestClassifier(),

    "SVM": SVC()

}

print("Accuracy without PCA:")

for name, model in models.items():

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

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

    print(f"{name}: {acc:.4f}")

pca = PCA(n\_components=5)

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

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

print("\nAccuracy with PCA:")

for name, model in models.items():

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

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

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

    print(f"{name}: {acc:.4f}")