# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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



## LAB REPORT

### on

Machine Learning (23CS6PCMAL)

#### Submitted by

**Rachana N (1BM23CS416)**

#### 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 **Rachana N (1BM23CS416),** who is a 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:** Ms. Saritha A N

Assistant Professor Department of CSE, BMSCE

# Index

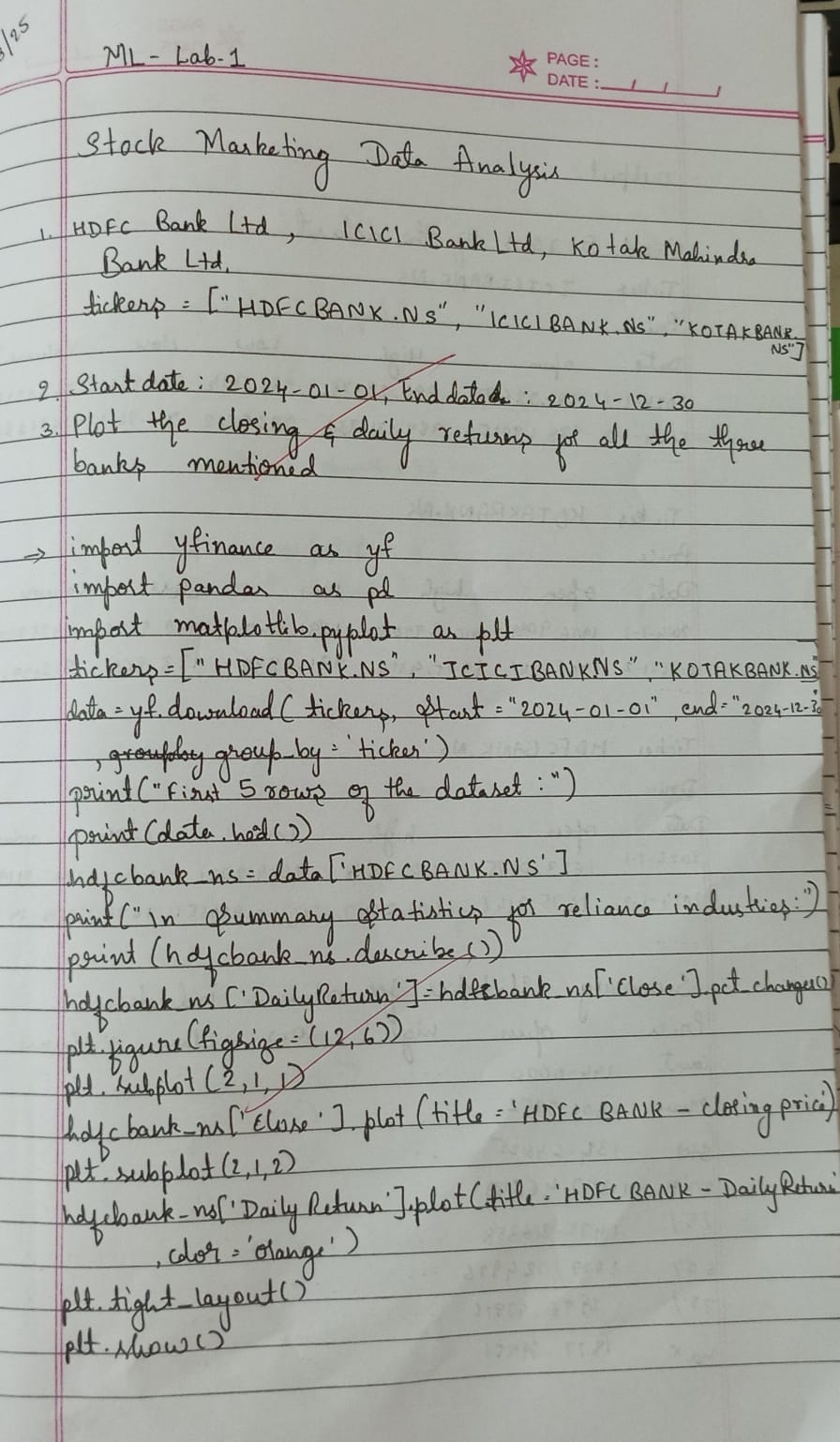
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 21-2-2025 | Write a python program to import and export data using Pandas library functions |  |
| 2 | 3-3-2025 | Demonstrate various data pre-processing techniques for a given dataset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

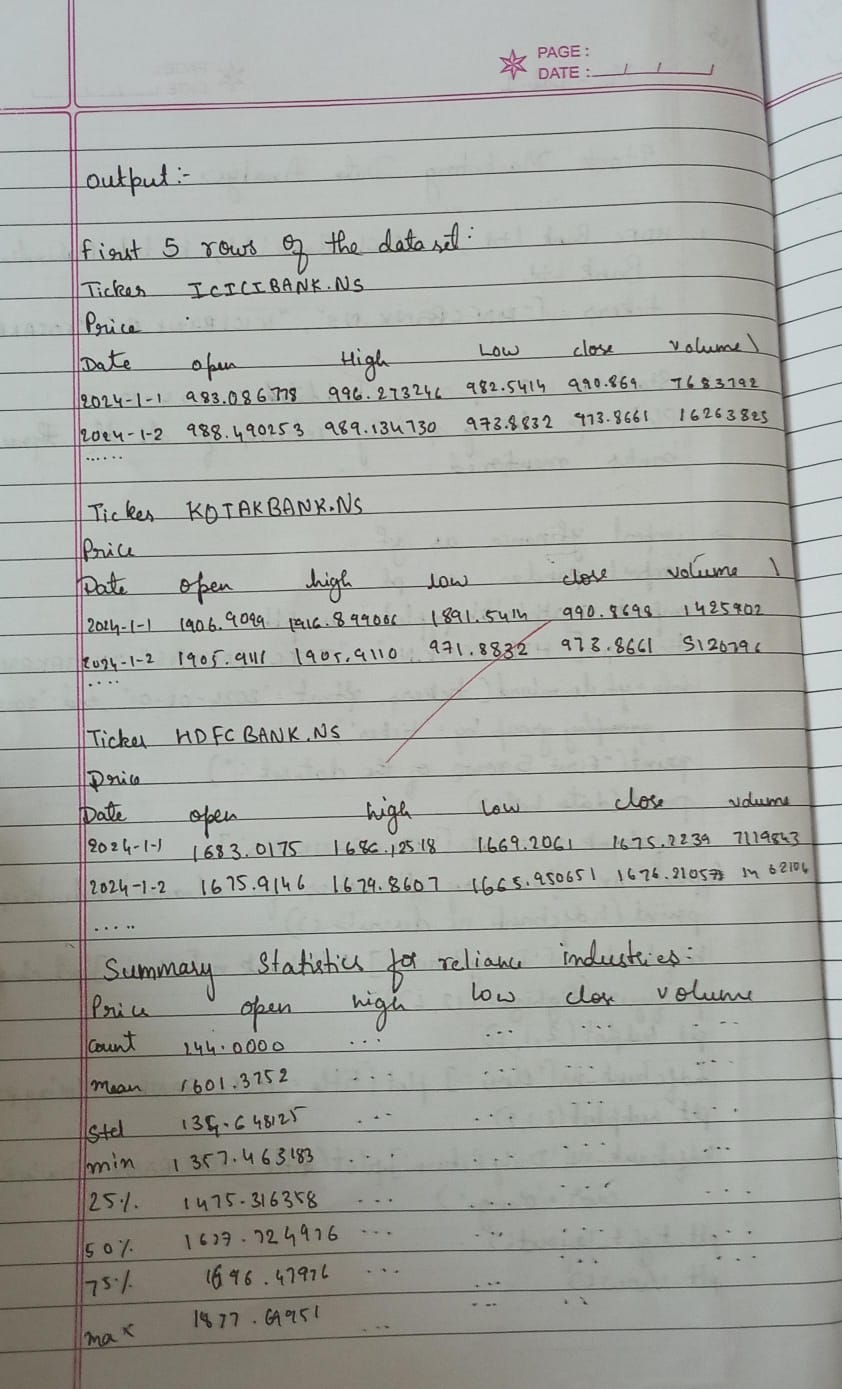
**Github Link: https://github.com/Rachanan2608/ML**

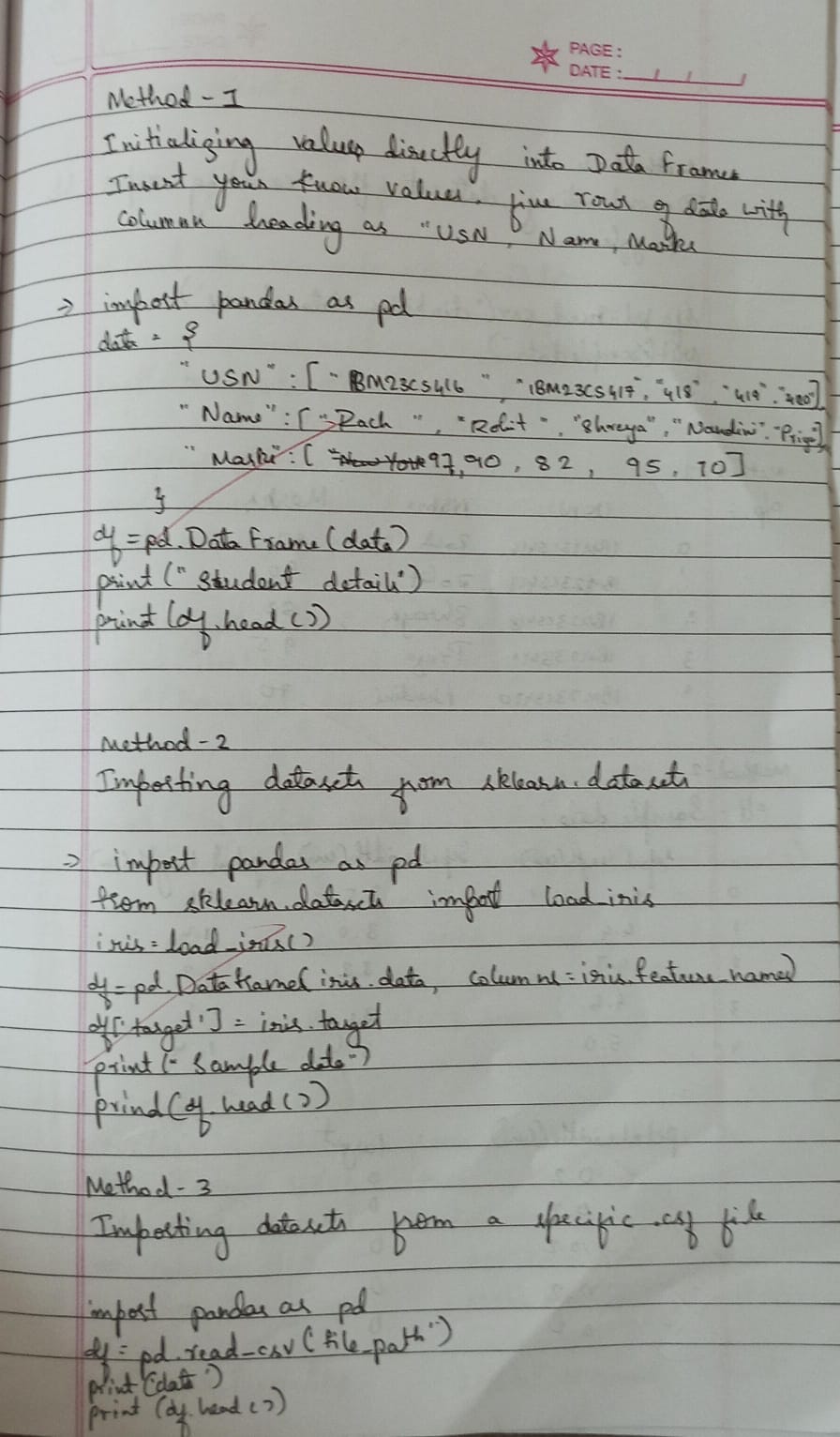
**Program 1**

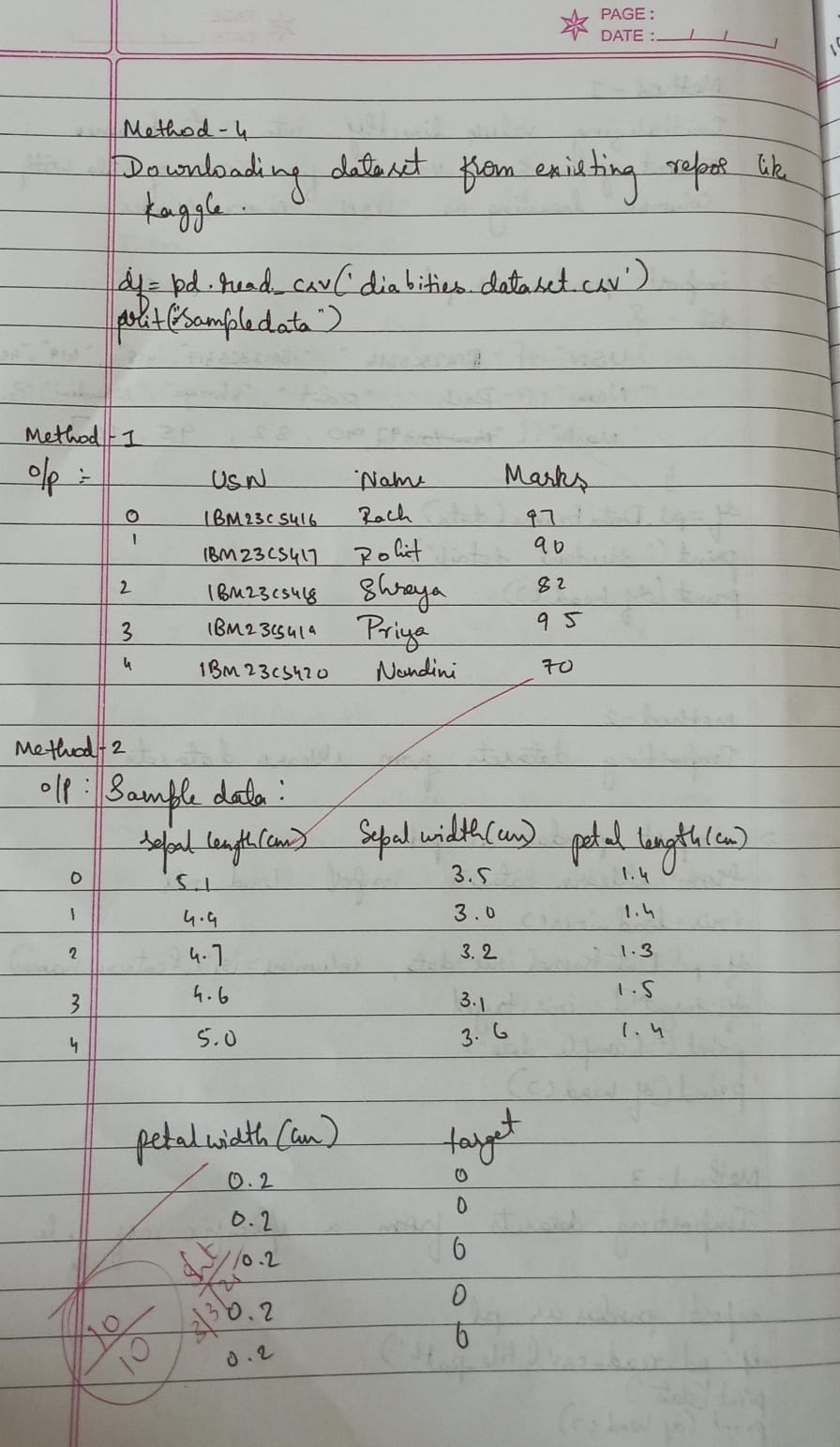
Write a python program to import and export data using Pandas library functions.

**Screenshot:**









**Code:**

import pandas as pd

data = {

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

'Age': [25, 30, 35, 40],

'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']

}

df = pd.DataFrame(data)

print("Sample data:")

print(df.head())

from sklearn.datasets import load\_iris

iris = load\_iris()

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

df['target'] = iris.target

print("Sample data:")

print(df.head())

from sklearn.datasets import load\_iris

iris = load\_iris()

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

df['target'] = iris.target

print("Sample data:")

print(df.head())

file\_path = 'mobiles-dataset-2025.csv'

df = pd.read\_csv(file\_path, encoding='latin-1') # or 'cp1252' or other suitable encoding

print("Sample data:")

print(df.head())

import pandas as pd

data = {

'USN': ['IS001','IS002','IS003','IS004','IS005'],

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

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

}

df = pd.DataFrame(data)

print("Sample data:")

print(df.head())

file\_path = 'sample\_sales\_data.csv'

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

print("Sample data:")

print(df.head())

print("\n")

df = pd.read\_csv("/content/dataset-of-diabetes .csv",encoding='latin-1')

print("Sample data:")

print(df.head())

print("\n")

df =pd.read\_csv('sample\_sales\_data.csv')

print("Sample data:")

print(df.head())

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

print("Data saved to output.csv")

sales\_df =pd.read\_csv('sample\_sales\_data.csv')

print("Sample data:")

print(sales\_df.head())

sales\_by\_region =sales\_df.groupby('Region')['Sales'].sum()

print("\nTotal sales by region:")

print(sales\_by\_region)

best\_selling\_products =sales\_df.groupby('Product')['Quantity'].sum().sort\_values(ascending=False)

print("\nBest-selling products by quantity:")

print(best\_selling\_products)

sales\_by\_region.to\_csv('sales\_by\_region.csv')

best\_selling\_products.to\_csv('best\_selling\_products.csv')

print("Data saved to sales\_by\_region.csv and best\_selling\_products.csv")

import yfinance as yf

import matplotlib.pyplot as plt

tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]

data = yf.download(tickers, start="2022-10-01", end="2023-10-01",

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)

print("\n")

reliance\_data = data['RELIANCE.NS']

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

print(reliance\_data.describe())

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

print("\n")

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

plt.subplot(2, 1, 1)

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

plt.subplot(2, 1, 2)

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

plt.tight\_layout()

plt.show()

reliance\_data.to\_csv('reliance\_stock\_data.csv')

tickers = ["HDFCBANK.NS", "ICICI.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)

print("\n")

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

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

print(reliance\_data.describe())

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

print("\n")

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

plt.subplot(2, 1, 1)

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

plt.subplot(2, 1, 2)

reliance\_data['Daily Return'].plot(title="HDFCIndustries - Daily Returns", color='red')

plt.tight\_layout()

plt.show()

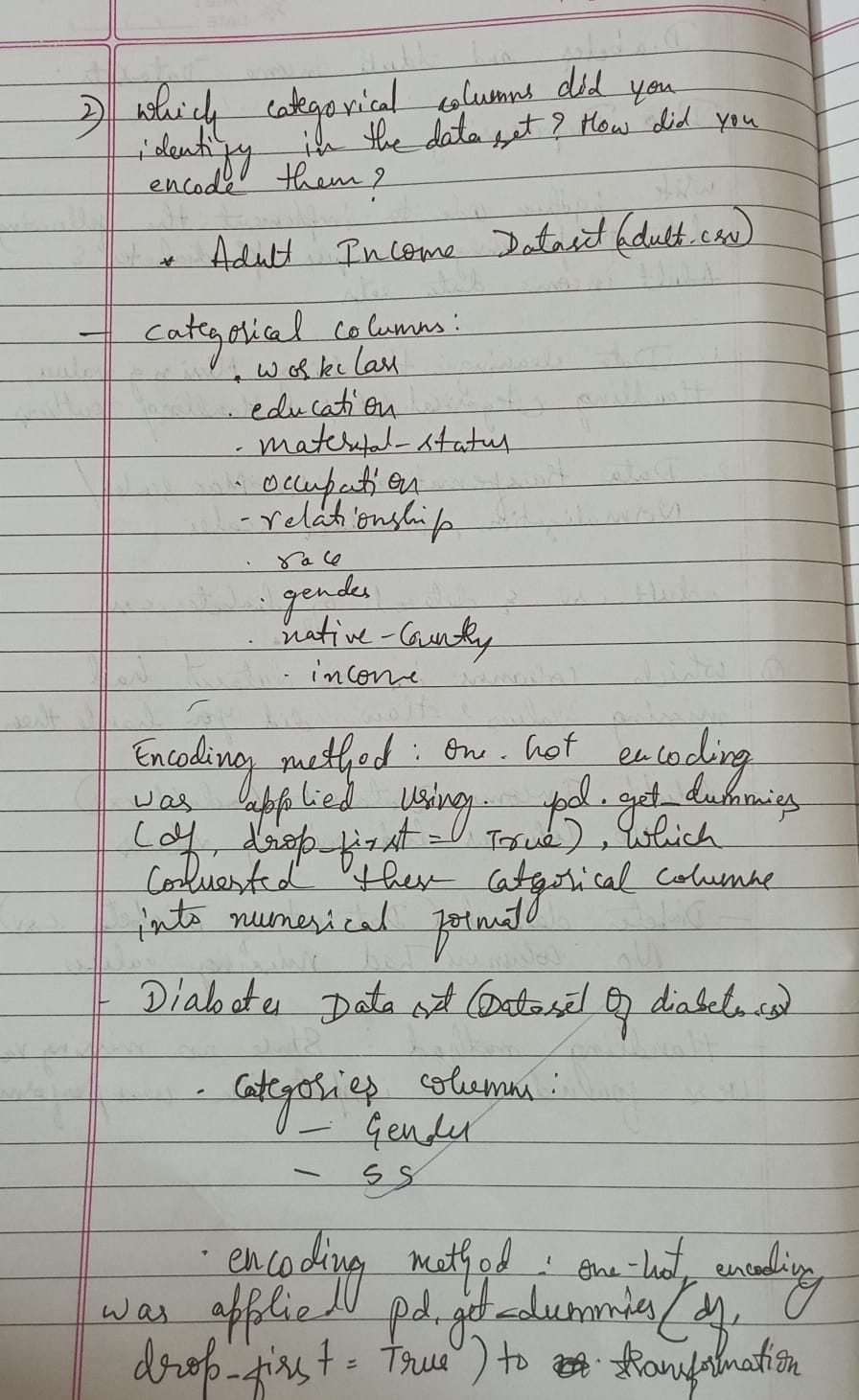
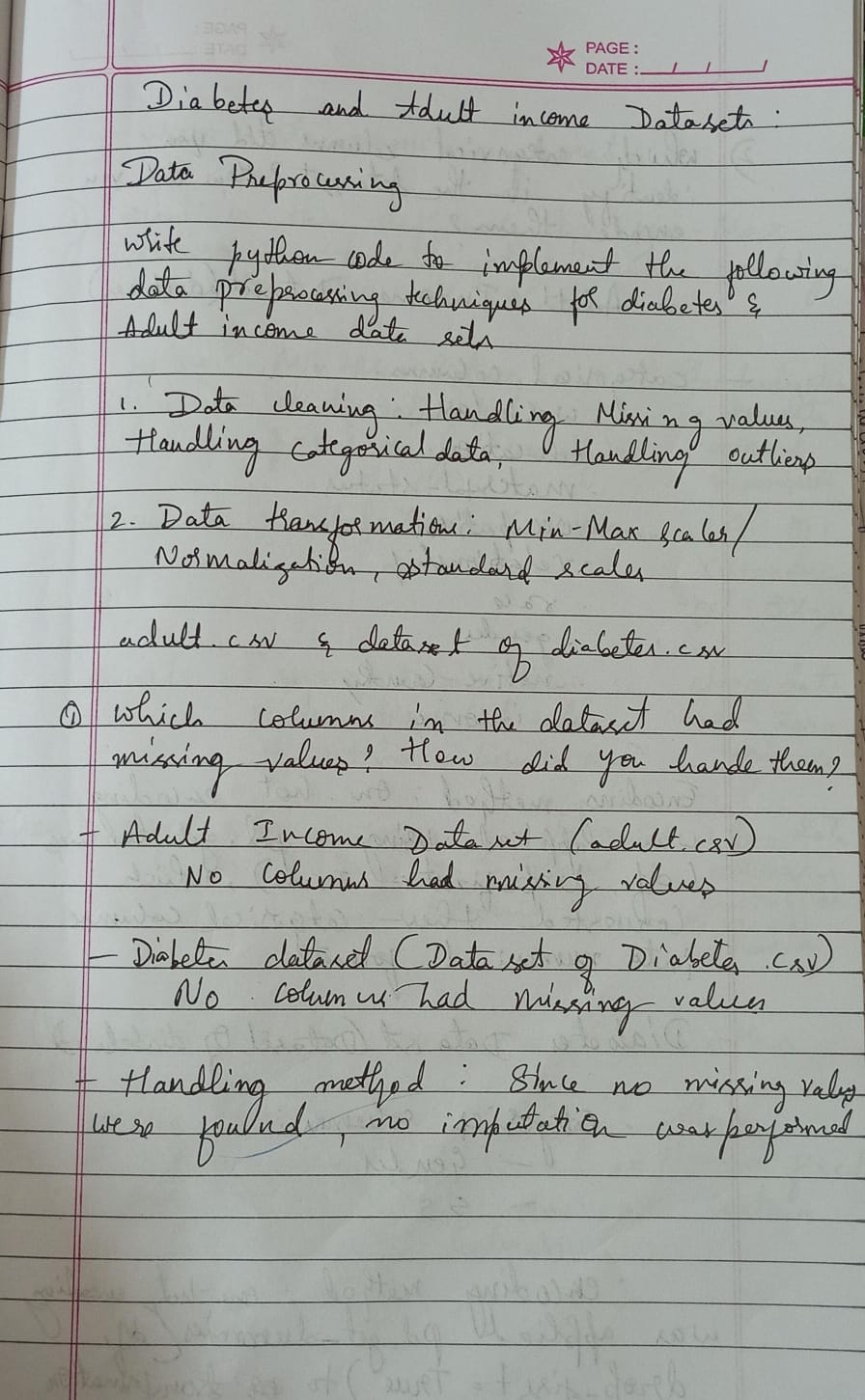
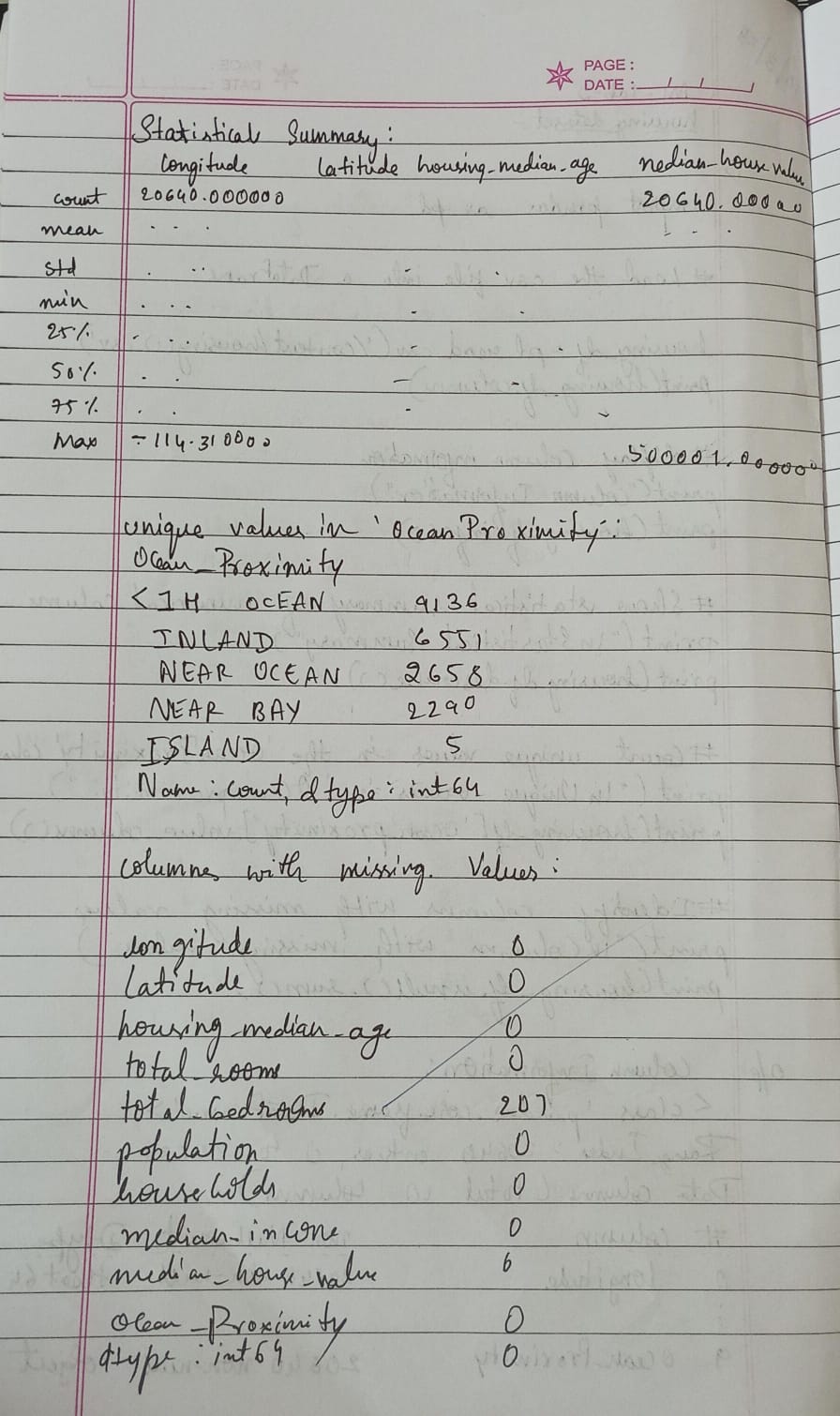
reliance\_data.to\_csv('hdfc\_stock\_data.csv')

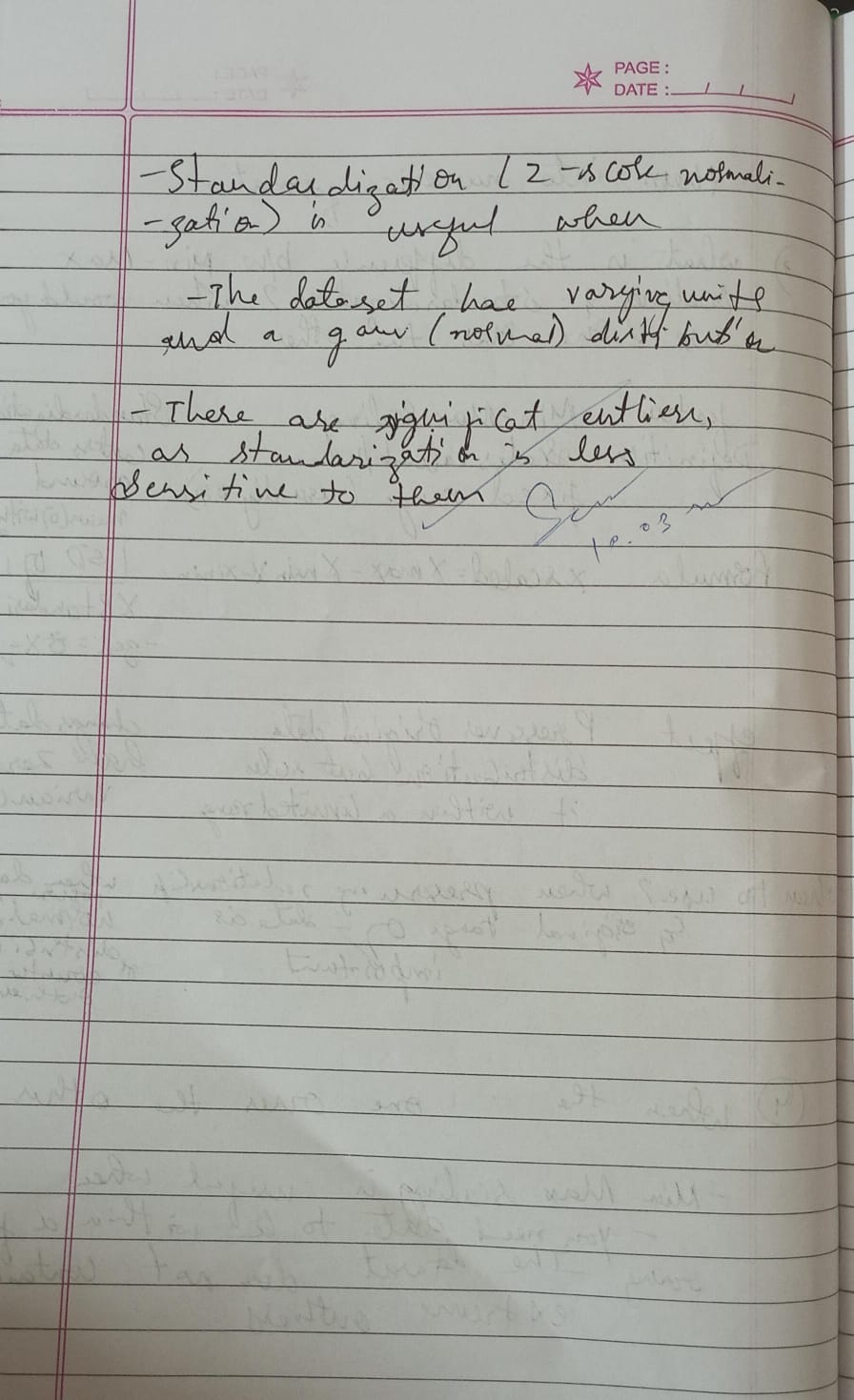
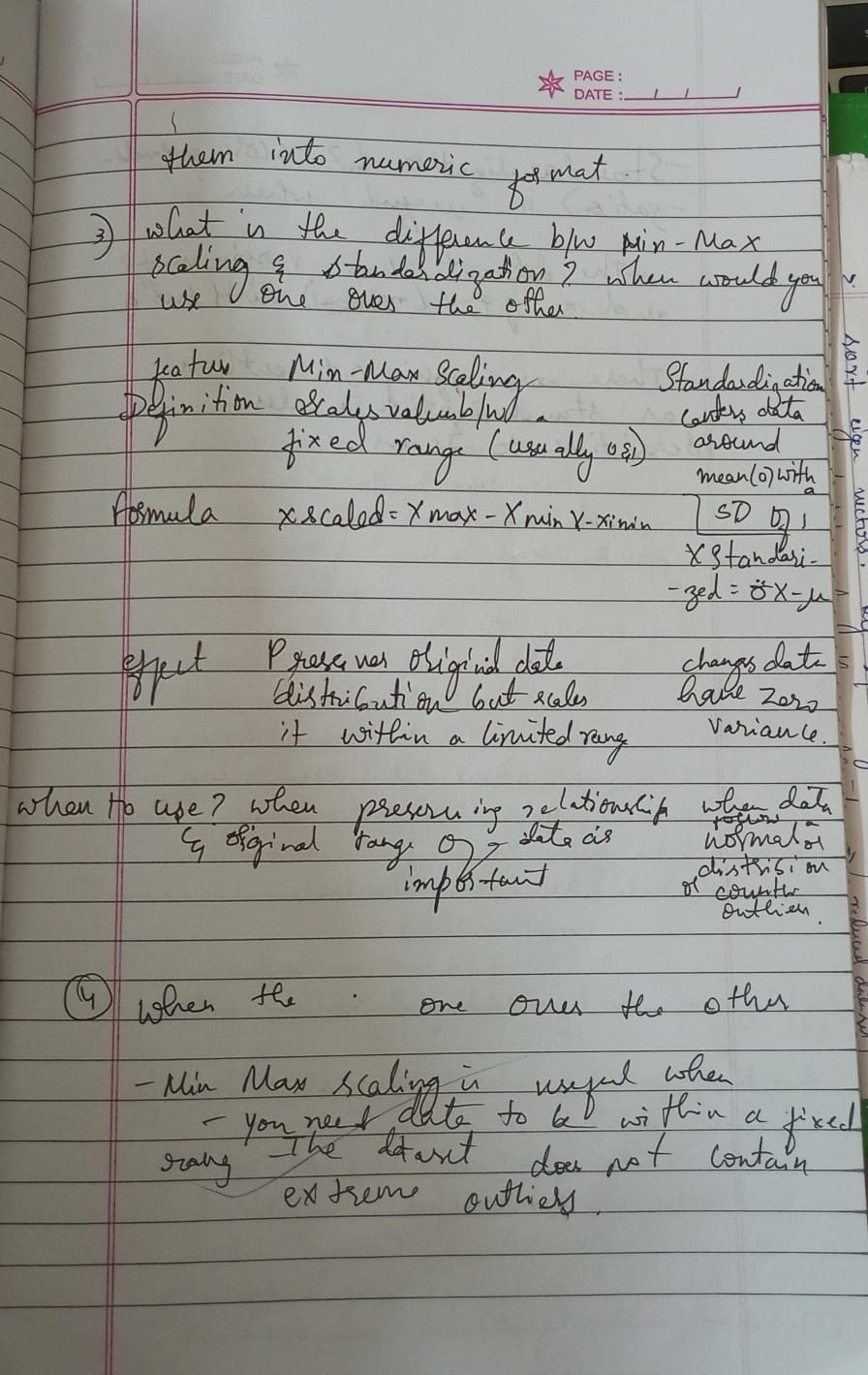
print("\nhdfc stock data saved to 'hdfc\_stock\_data.csv'.")

**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

try:

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

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

except FileNotFoundError:

print("Error: Please upload 'diabetes.csv' and 'adult.csv' to your Google Colab environment.")

exit()

diabetes\_df.head(10)

adult\_df.head(10)

diabetes\_df.shape

adult\_df.shape

*#Handling Missing Values*

diabetes\_numeric\_cols = diabetes\_df.select\_dtypes(include=[np.number]).columns

diabetes\_categorical\_cols = diabetes\_df.select\_dtypes(exclude=[np.number]).columns

adult\_numeric\_cols = adult\_df.select\_dtypes(include=[np.number]).columns

adult\_categorical\_cols = adult\_df.select\_dtypes(exclude=[np.number]).columns

diabetes\_numeric\_imputer = SimpleImputer(strategy='mean')

adult\_numeric\_imputer = SimpleImputer(strategy='mean')

diabetes\_df[diabetes\_numeric\_cols] = diabetes\_numeric\_imputer.fit\_transform(diabetes\_df[diabetes\_numeric\_cols])

adult\_df[adult\_numeric\_cols] = adult\_numeric\_imputer.fit\_transform(adult\_df[adult\_numeric\_cols])

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

adult\_categorical\_imputer = SimpleImputer(strategy='most\_frequent')

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

adult\_df[adult\_categorical\_cols] = adult\_categorical\_imputer.fit\_transform(adult\_df[adult\_categorical\_cols])

print("Missing values in Diabetes dataset after imputation:")

print(diabetes\_df.isnull().sum())

print("Missing values in Adult Income dataset after imputation:")

print(adult\_df.isnull().sum())

adult\_df.replace("?", np.nan, inplace=True)

print("Missing values in Adult Income dataset after replacing '?':")

print(adult\_df.isnull().sum())

from sklearn.impute import SimpleImputer

*# Identify numeric and categorical columns*

adult\_numeric\_cols = adult\_df.select\_dtypes(include=[np.number]).columns

adult\_categorical\_cols = adult\_df.select\_dtypes(exclude=[np.number]).columns

*# Handle missing values in numeric columns using mean imputation*

adult\_numeric\_imputer = SimpleImputer(strategy='mean')

adult\_df[adult\_numeric\_cols] = adult\_numeric\_imputer.fit\_transform(adult\_df[adult\_numeric\_cols])

*# Handle missing values in categorical columns using most frequent imputation*

adult\_categorical\_imputer = SimpleImputer(strategy='most\_frequent')

adult\_df[adult\_categorical\_cols] = adult\_categorical\_imputer.fit\_transform(adult\_df[adult\_categorical\_cols])

print("Missing values in Adult Income dataset after imputation:")

print(adult\_df.isnull().sum())

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

*# Encode categorical columns in Diabetes dataset*

for col in diabetes\_categorical\_cols:

diabetes\_df[col] = label\_encoder.fit\_transform(diabetes\_df[col])

*# Encode categorical columns in Adult Income dataset*

for col in adult\_categorical\_cols:

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

print("Encoded columns in Diabetes dataset:")

print(diabetes\_df.head())

print("Encoded columns in Adult Income dataset:")

print(adult\_df.head())

*#Handling outliers*

def remove\_outliers(df):

Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

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

return df\_no\_outliers

diabetes\_df\_no\_outliers = remove\_outliers(diabetes\_df)

adult\_df\_no\_outliers = remove\_outliers(adult\_df)

print("Diabetes dataset shape after removing outliers:", diabetes\_df\_no\_outliers.shape)

print("Adult Income dataset shape after removing outliers:", adult\_df\_no\_outliers.shape)

*#Min-max scaling*

from sklearn.preprocessing import MinMaxScaler

min\_max\_scaler = MinMaxScaler()

diabetes\_scaled\_minmax = pd.DataFrame(min\_max\_scaler.fit\_transform(diabetes\_df\_no\_outliers), columns=diabetes\_df\_no\_outliers.columns)

adult\_scaled\_minmax = pd.DataFrame(min\_max\_scaler.fit\_transform(adult\_df\_no\_outliers), columns=adult\_df\_no\_outliers.columns)

print("Diabetes dataset after Min-Max scaling:")

print(diabetes\_scaled\_minmax.head())

print("Adult Income dataset after Min-Max scaling:")

print(adult\_scaled\_minmax.head())

*# Initialize Standard Scaler*

from sklearn.preprocessing import StandardScaler

standard\_scaler = StandardScaler()

diabetes\_scaled\_standard = pd.DataFrame(standard\_scaler.fit\_transform(diabetes\_df\_no\_outliers), columns=diabetes\_df\_no\_outliers.columns)

adult\_scaled\_standard = pd.DataFrame(standard\_scaler.fit\_transform(adult\_df\_no\_outliers), columns=adult\_df\_no\_outliers.columns)

print("Diabetes dataset after Standard scaling:")

print(diabetes\_scaled\_standard.head())

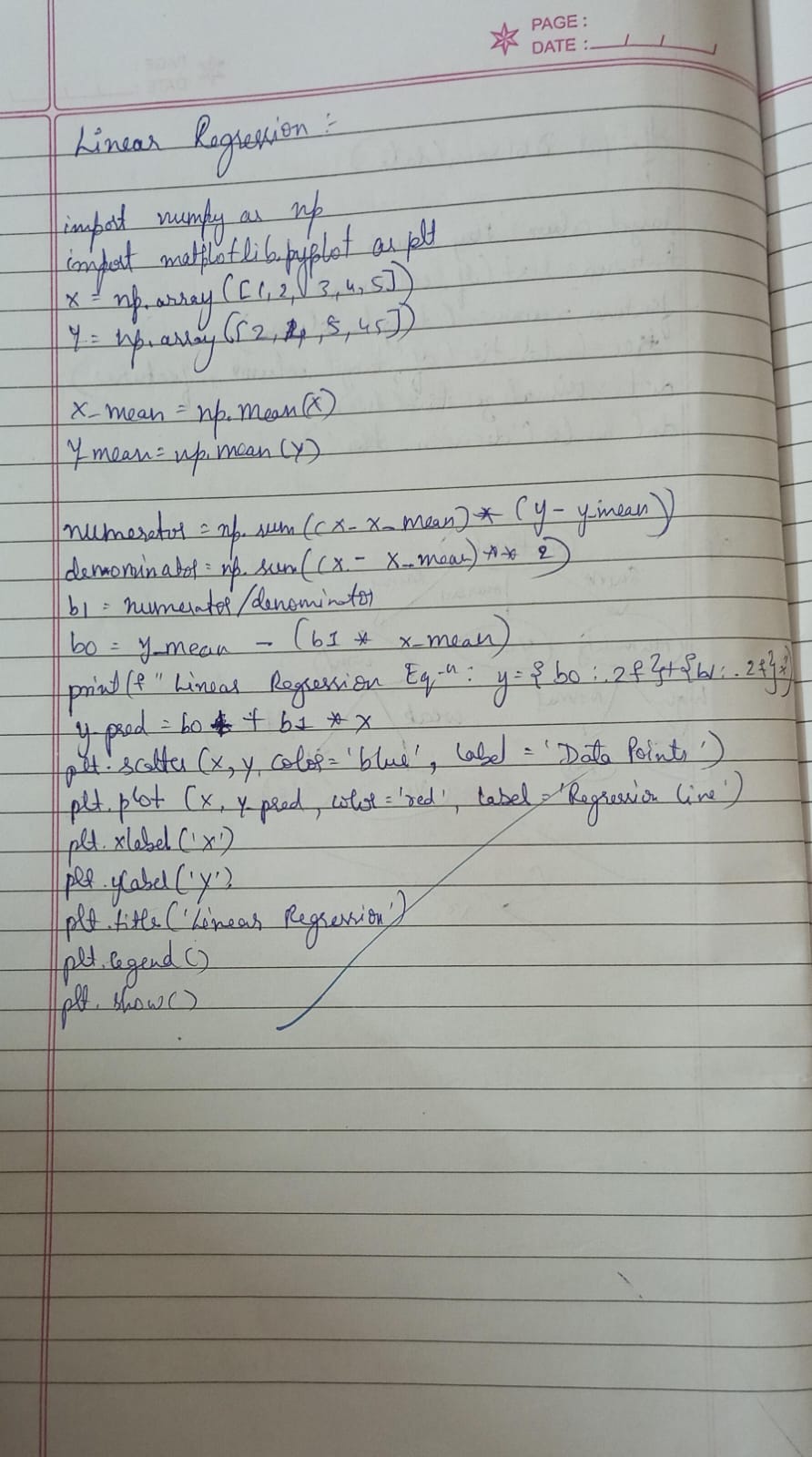
print("Adult Income dataset after Standard scaling:")

print(adult\_scaled\_standard.head())

**Program 3**

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset.

**Screenshot:**



**Code:**

import pandas as pd

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

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

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

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\_

3300\*135.78767123 + 180616.43835616432

*#(2) Predict price of a home with area = 5000 sqr ft*

reg.predict([[5000]])

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

df.bedrooms.median()

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

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

df = pd.read\_csv('canada\_per\_capita\_income.csv')

print(df.head())

X = df[['year']]

y = df['per capita income (US$)']

reg = LinearRegression()

reg.fit(X, y)

predicted\_income\_2020 = reg.predict([[2020]])

print(f"Predicted per capita income for Canada in 2020: {predicted\_income\_2020[0]:.2f}")

plt.scatter(X, y, color='blue')

plt.plot(X, reg.predict(X), color='red')

plt.xlabel('Year')

plt.ylabel('Per Capita Income')

plt.title('Per Capita Income in Canada Over the Years')

plt.show()

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

print(df.head())

print("Missing values in the dataset:")

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

df['YearsExperience'] = df['YearsExperience'].fillna(df['YearsExperience'].median())

print("\nMissing values after filling:")

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

X = df[['YearsExperience']]

y = df['Salary']

reg = LinearRegression()

reg.fit(X, y)

predicted\_salary\_12\_years = reg.predict([[12]])

print(f"\nPredicted salary for an employee with 12 years of experience: ${predicted\_salary\_12\_years[0]:,.2f}")

plt.scatter(X, y, color='blue')

plt.plot(X, reg.predict(X), color='red')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Salary vs. Years of Experience')

plt.show()

def convert\_to\_numeric(value):

word\_to\_num = {

'zero': 0, '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,

'fifteen': 15

}

return word\_to\_num.get(value.lower(), value) if isinstance(value, str) else value

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

print(df.head())

df\_hiring['experience'] = df\_hiring['experience'].apply(convert\_to\_numeric)

df\_hiring['experience'].fillna(0, inplace=True)

df\_hiring['test\_score(out of 10)'].fillna(df\_hiring['test\_score(out of 10)'].median(), inplace=True)

df\_hiring['interview\_score(out of 10)'].fillna(df\_hiring['interview\_score(out of 10)'].median(), inplace=True)

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

y\_hiring = df\_hiring['salary($)']

reg\_hiring = LinearRegression()

reg\_hiring.fit(X\_hiring, y\_hiring)

candidates = np.array([[2, 9, 6], [12, 10, 10]])

predicted\_salaries = reg\_hiring.predict(candidates)

for i, candidate in enumerate(candidates):

print(f"\nPredicted salary for candidate with {candidate[0]} yrs experience, {candidate[1]} test score, {candidate[2]} interview score: {predicted\_salaries[i]:.2f} USD")

plt.scatter(y\_hiring, reg\_hiring.predict(X\_hiring), color='blue', label='Predicted vs Actual')

plt.xlabel("Actual Salary")

plt.ylabel("Predicted Salary")

plt.title("Actual vs Predicted Salary")

plt.legend()

plt.show()

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

print(df.head())

label\_encoder = LabelEncoder()

df\_companies['State'] = label\_encoder.fit\_transform(df\_companies['State'])

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

y\_companies = df\_companies['Profit']

df\_companies.fillna(df\_companies.median(), inplace=True)

reg\_companies = LinearRegression()

reg\_companies.fit(X\_companies, y\_companies)

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

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

print(f"Predicted profit: {predicted\_profit[0]:.2f} USD")

plt.scatter(y\_companies, reg\_companies.predict(X\_companies), color='blue', label='Predicted vs Actual')

plt.xlabel("Actual Profit")

plt.ylabel("Predicted Profit")

plt.title("Actual vs Predicted Profit")

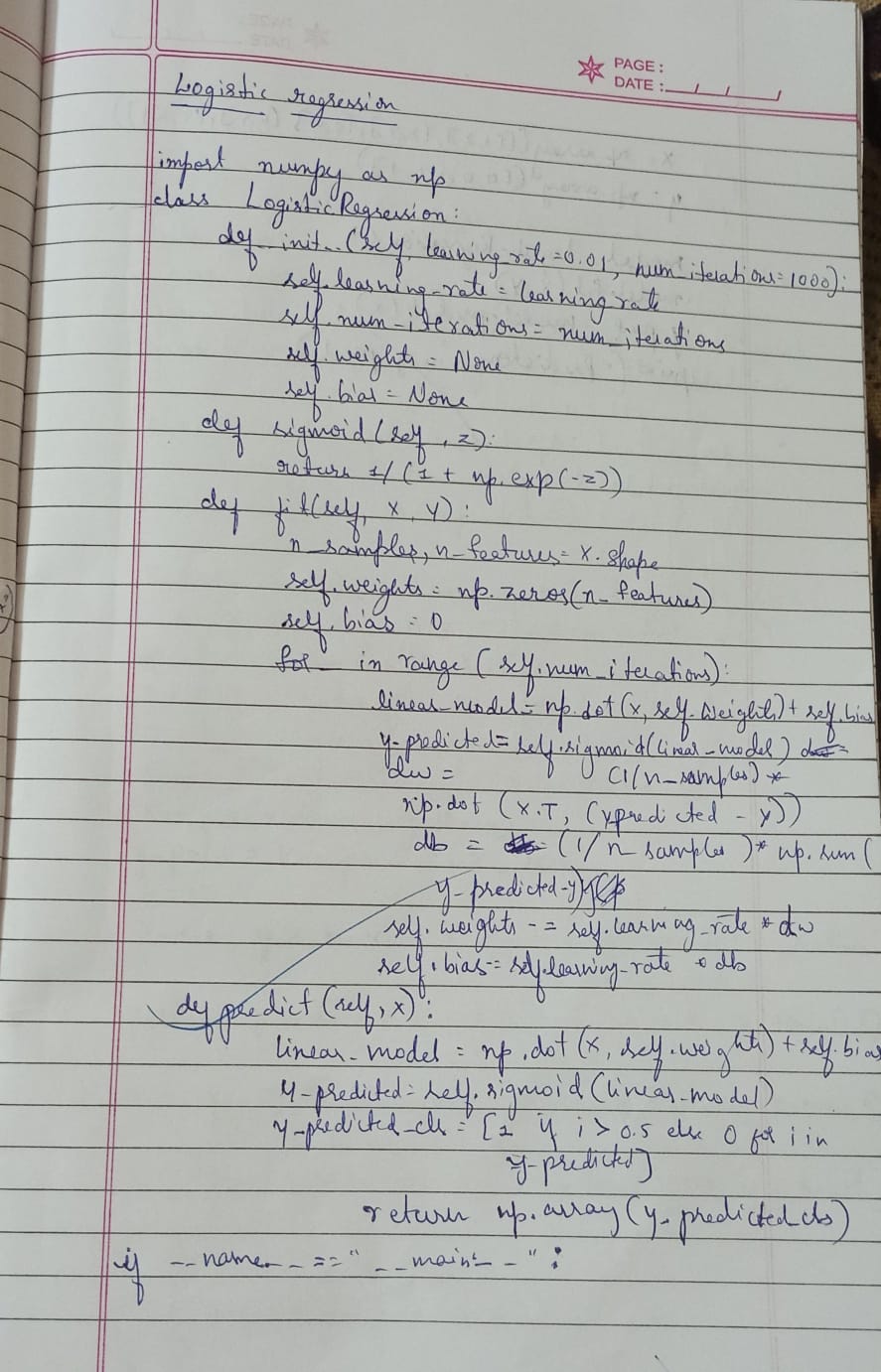
plt.legend()

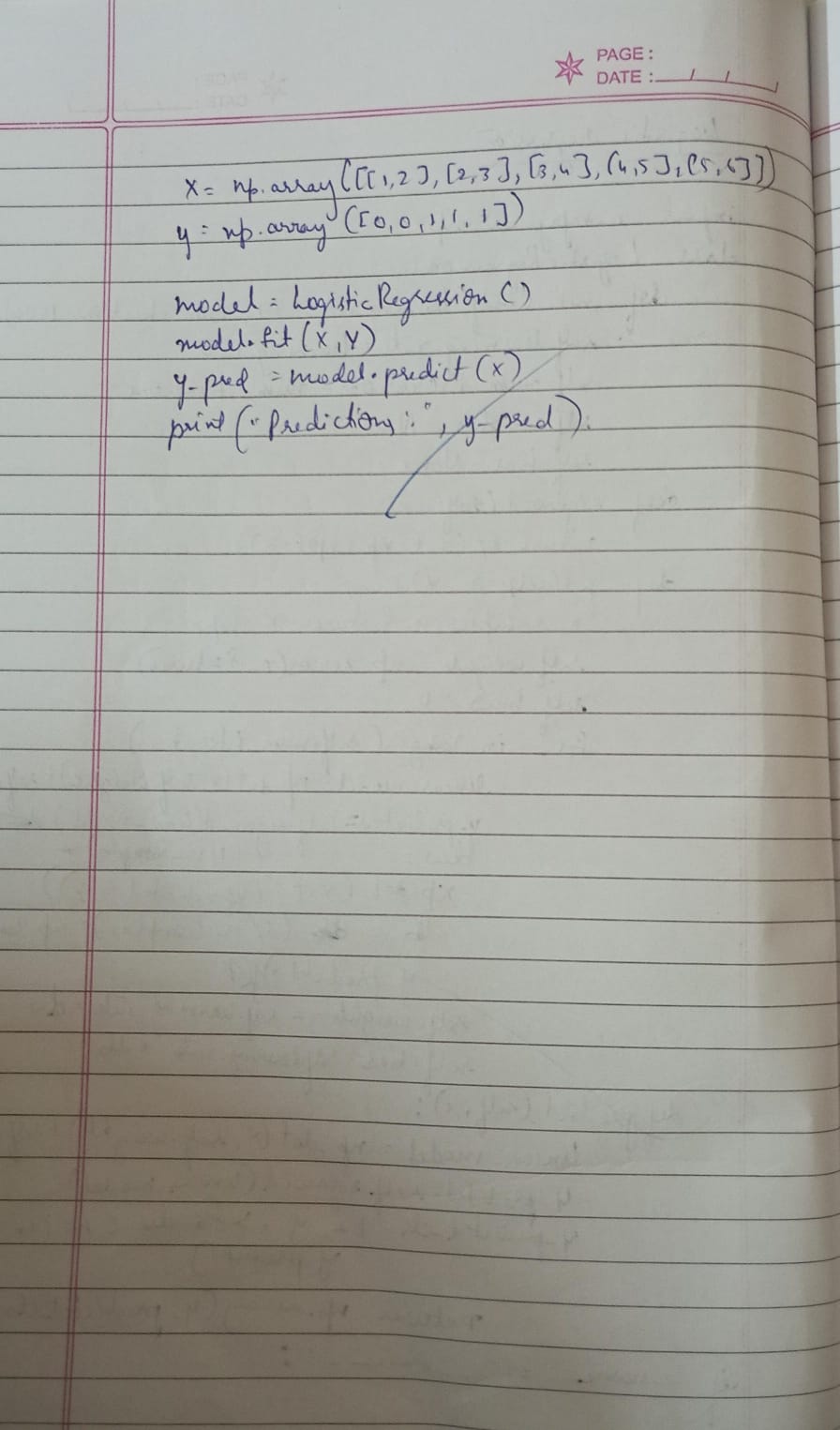
plt.show()

**Program 4**

Build Logistic Regression Model for a given dataset.

**Screenshot:**

****

****

**Code:**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

print(df.info())

numericCols = df.select\_dtypes(include=['float64', 'int64']).columns

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

sns.heatmap(df[numericCols].corr(), annot=True, cmap='coolwarm', fmt='.2f')

plt.title("Correlation Matrix (Numeric Features)")

plt.show()

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

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

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

plt.xlabel("Salary Level")

plt.ylabel("Employee Count")

plt.show()

import pandas as pd

df = pd.read\_csv("zoo-data.csv")

print(df.info())

print(df.head())

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

df.drop(columns=['animal\_name'], inplace=True)

X = df.drop(columns=['class\_type'])

y = df['class\_type']

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

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

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

scaler = StandardScaler()

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

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

logreg = LogisticRegression(max\_iter=200, multi\_class='multinomial', solver='lbfgs')

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

from sklearn.metrics import accuracy\_score

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

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

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

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

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

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=logreg.classes\_)

disp.plot(cmap=plt.cm.Blues)

plt.title("Confusion Matrix for Zoo Animal Classification")

plt.show()

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

pred\_classes = [class\_mapping[pred] for pred in y\_pred]

print("Predicted Classes:", pred\_classes)

import seaborn as sns

import matplotlib.pyplot as plt

sns.countplot(x='class\_type', data=df)

plt.title("Class Distribution of Animals in Zoo Dataset")

plt.xlabel("Class Type")

plt.ylabel("Count")

plt.show()

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

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

class\_labels = [class\_mapping[num] for num in logreg.classes\_]

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=class\_labels)

disp.plot(cmap=plt.cm.Blues)

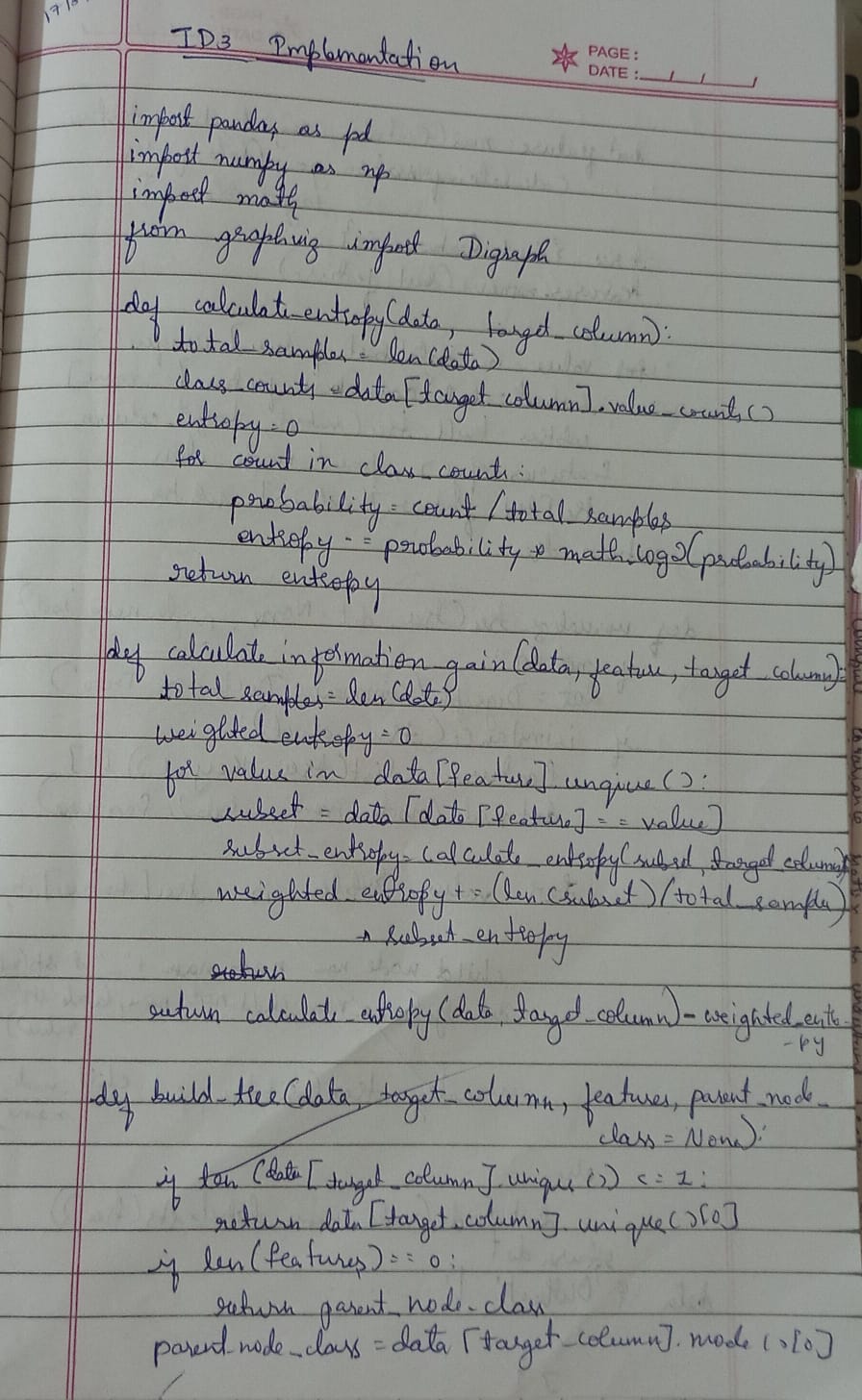
plt.title("Confusion Matrix with Class Names")

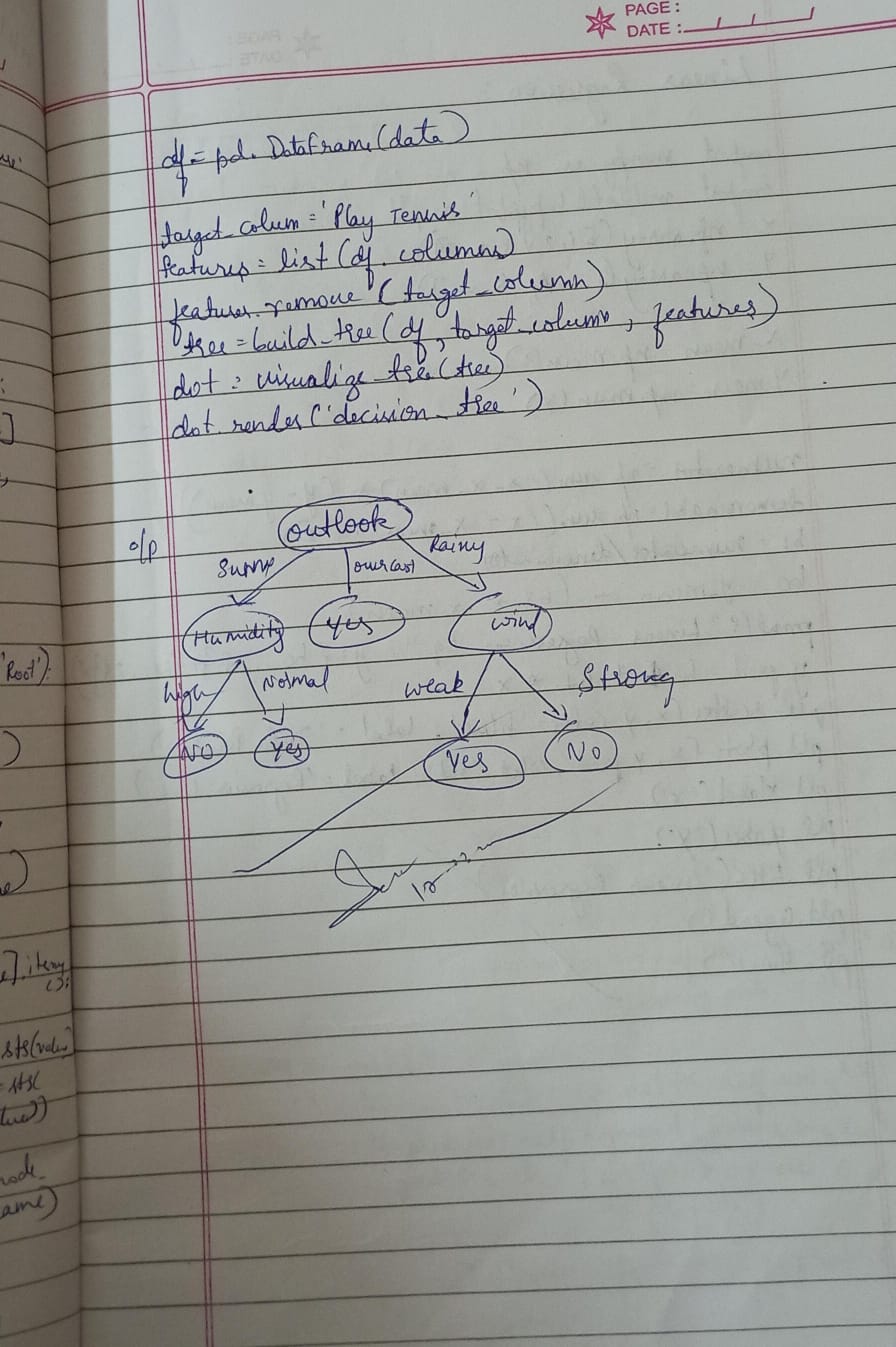
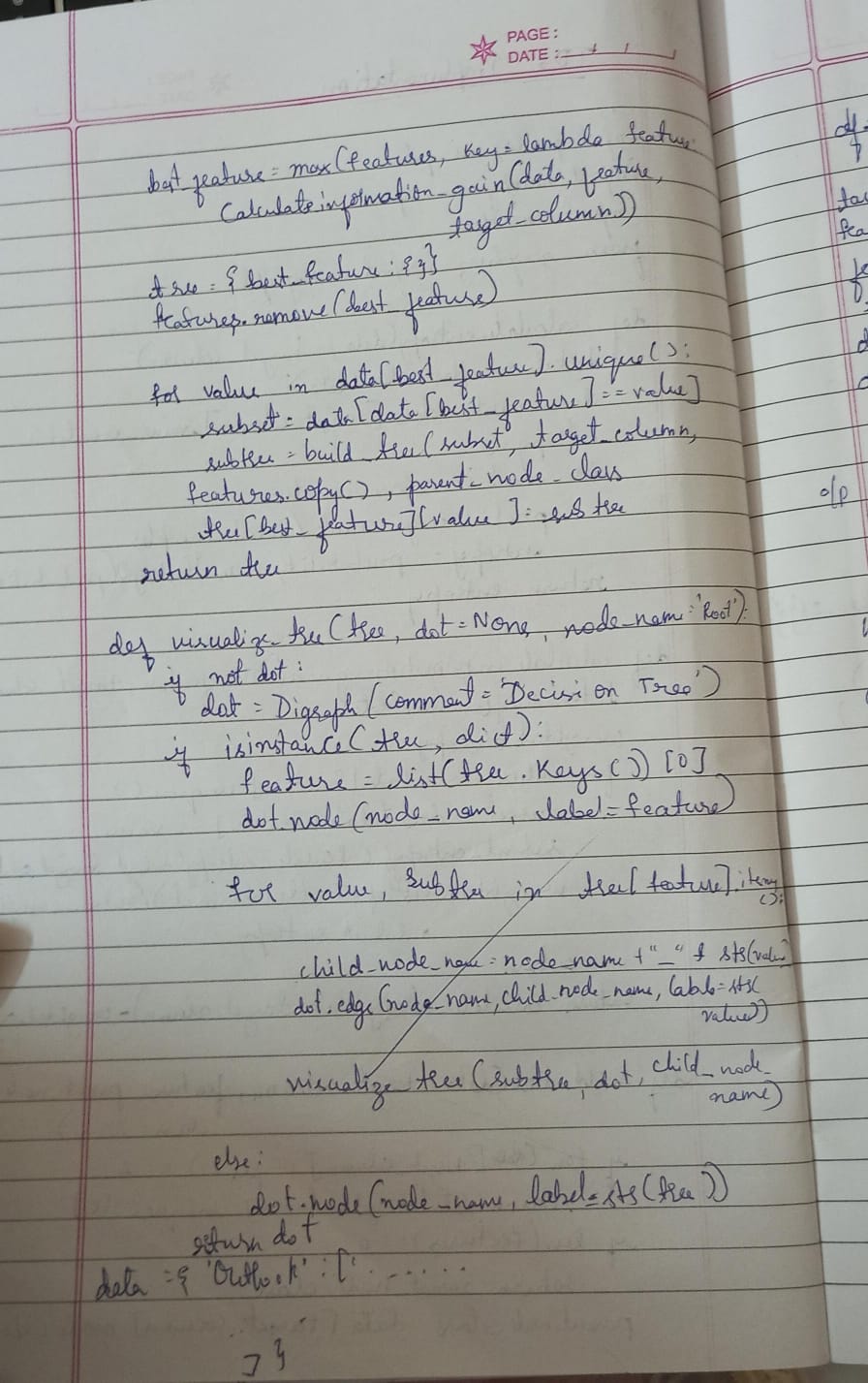
plt.show()

**Program 5**

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 seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score, f1\_score

from sklearn.preprocessing import LabelEncoder

def train\_and\_evaluate\_iris():

iris\_df = pd.read\_csv("iris.csv")

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

y = iris\_df["species"]

y\_le = LabelEncoder()

y = 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)

model = DecisionTreeClassifier(random\_state=42)

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

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

*# Evaluating the model*

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

prec = precision\_score(y\_test, y\_pred, average='weighted')

rec = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

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

print("IRIS Dataset Classification:")

print(f"Accuracy Score: {acc:.4f}")

print(f"Precision Score: {prec:.4f}")

print(f"Recall Score: {rec:.4f}")

print(f"F1 Score: {f1:.4f}")

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=y\_le.classes\_, yticklabels=y\_le.classes\_)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix: iris.csv")

plt.show()

train\_and\_evaluate\_iris()

def train\_and\_evaluate\_drug():

drug\_df = pd.read\_csv("drug.csv")

categorical\_features = ["Sex", "BP", "Cholesterol"]

label\_encoders = {}

for col in categorical\_features:

le = LabelEncoder()

drug\_df[col] = le.fit\_transform(drug\_df[col])

label\_encoders[col] = le

X = drug\_df.drop(columns=["Drug"])

y = drug\_df["Drug"]

y\_le = LabelEncoder()

y = 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)

model = DecisionTreeClassifier(random\_state=42)

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

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

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

prec = precision\_score(y\_test, y\_pred, average='weighted')

rec = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

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

print("Drug Dataset Classification:")

print(f"Accuracy Score: {acc:.4f}")

print(f"Precision Score: {prec:.4f}")

print(f"Recall Score: {rec:.4f}")

print(f"F1 Score: {f1:.4f}")

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=y\_le.classes\_, yticklabels=y\_le.classes\_)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix: drug.csv")

plt.show()

train\_and\_evaluate\_drug()

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.tree import DecisionTreeRegressor, plot\_tree

from sklearn.preprocessing import StandardScaler

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

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

X = petrol\_df.drop(columns=["Petrol\_Consumption"])

y = petrol\_df["Petrol\_Consumption"]

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)

model = DecisionTreeRegressor(max\_depth=5, random\_state=42)

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

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

print("Petrol Consumption Regression:")

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

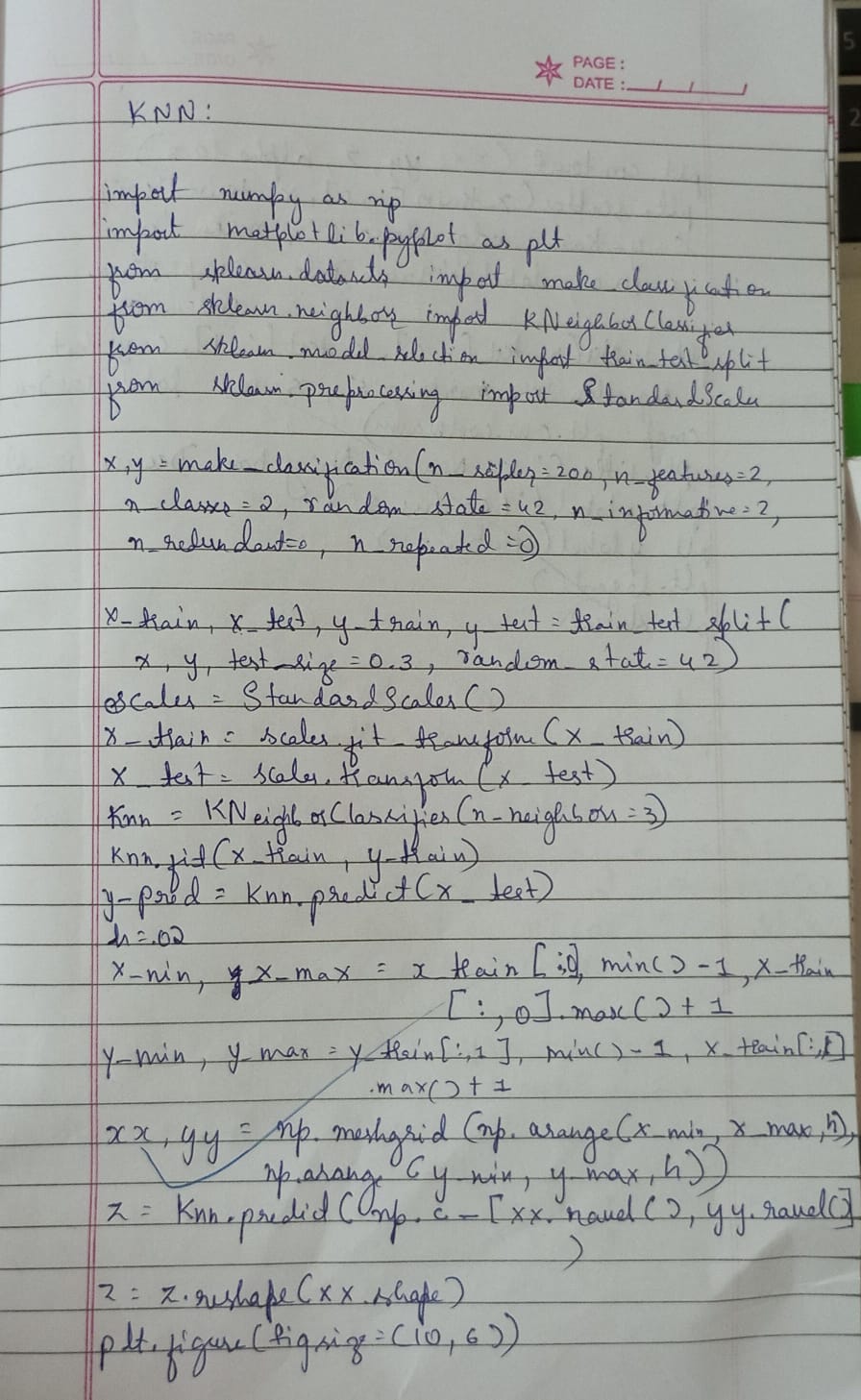
print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred))

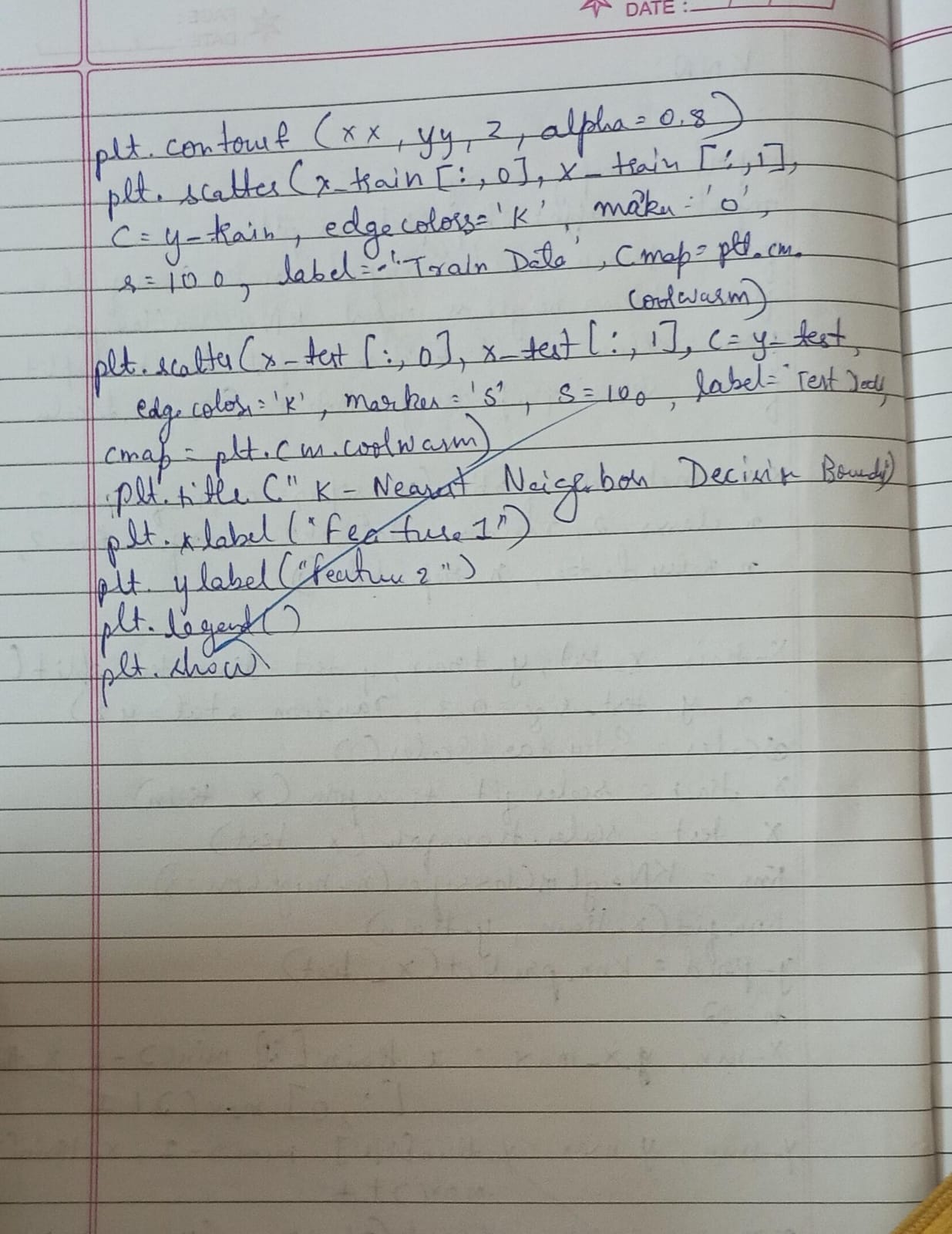
print("Root Mean Squared Error (RMSE):", np.sqrt(mean\_squared\_error(y\_test, y\_pred)))

**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.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

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

import seaborn as sns

import matplotlib.pyplot as plt

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

le = LabelEncoder()

iris\_df['species'] = le.fit\_transform(iris\_df['species'])

X = iris\_df.drop('species', axis=1)

y = iris\_df['species']

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

error\_rates = []

accuracies = []

k\_values = range(1, 10)

for k in k\_values:

knn = KNeighborsClassifier(n\_neighbors=k)

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

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

error = 1 - accuracy\_score(y\_test, y\_pred\_k)

error\_rates.append(error)

accuracies.append(accuracy\_score(y\_test, y\_pred\_k))

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(k\_values, accuracies, marker='o', color='blue')

plt.title("Accuracy vs K")

plt.xlabel("K Value")

plt.ylabel("Accuracy")

plt.subplot(1, 2, 2)

plt.plot(k\_values, error\_rates, marker='o', color='red')

plt.title("Error Rate vs K")

plt.xlabel("K Value")

plt.ylabel("Error Rate")

plt.tight\_layout()

plt.show()

best\_k = k\_values[accuracies.index(max(accuracies))]

print(f"Best K: {best\_k} with Accuracy: {max(accuracies):.2f}")

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

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

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

*# Evaluation*

print("\n=== Final Evaluation on IRIS Dataset ===")

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

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, labels=[0, 1, 2], target\_names=le.classes\_))

*# Confusion Matrix*

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

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

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

plt.title("Confusion Matrix - IRIS")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

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

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

y = df['Outcome']

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

)

accuracy\_scores = []

k\_range = range(1, 21)

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors=k)

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

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

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

accuracy\_scores.append(acc)

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

plt.plot(k\_range, accuracy\_scores, marker='o', color='purple')

plt.title("Accuracy vs K (Diabetes Dataset)")

plt.xlabel("K Value")

plt.ylabel("Accuracy")

plt.xticks(k\_range)

plt.grid()

plt.show()

best\_k = k\_range[accuracy\_scores.index(max(accuracy\_scores))]

print(f"Best K: {best\_k} with Accuracy: {max(accuracy\_scores):.2f}")

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

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

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

print("=== Final Evaluation (Diabetes Dataset) ===")

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

print("\nClassification Report:")

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

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=['No Diabetes', 'Diabetes'], yticklabels=['No Diabetes', 'Diabetes'])

plt.title("Confusion Matrix - Diabetes")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

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

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

y = heart\_df['target']

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

)

accuracy\_scores = []

k\_range = range(1, 21)

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors=k)

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

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

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

accuracy\_scores.append(acc)

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

plt.plot(k\_range, accuracy\_scores, marker='o', color='red')

plt.title("Accuracy vs K (Heart Dataset)")

plt.xlabel("K Value")

plt.ylabel("Accuracy")

plt.xticks(k\_range)

plt.grid()

plt.show()

best\_k = k\_range[accuracy\_scores.index(max(accuracy\_scores))]

print(f"Best K: {best\_k} with Accuracy: {max(accuracy\_scores):.2f}")

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

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

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

print("=== Final Evaluation (Heart Dataset) ===")

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

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=['No Disease', 'Disease']))

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=['No Disease', 'Disease'], yticklabels=['No Disease', 'Disease'])

plt.title("Confusion Matrix - Heart Disease")

plt.xlabel("Predicted")

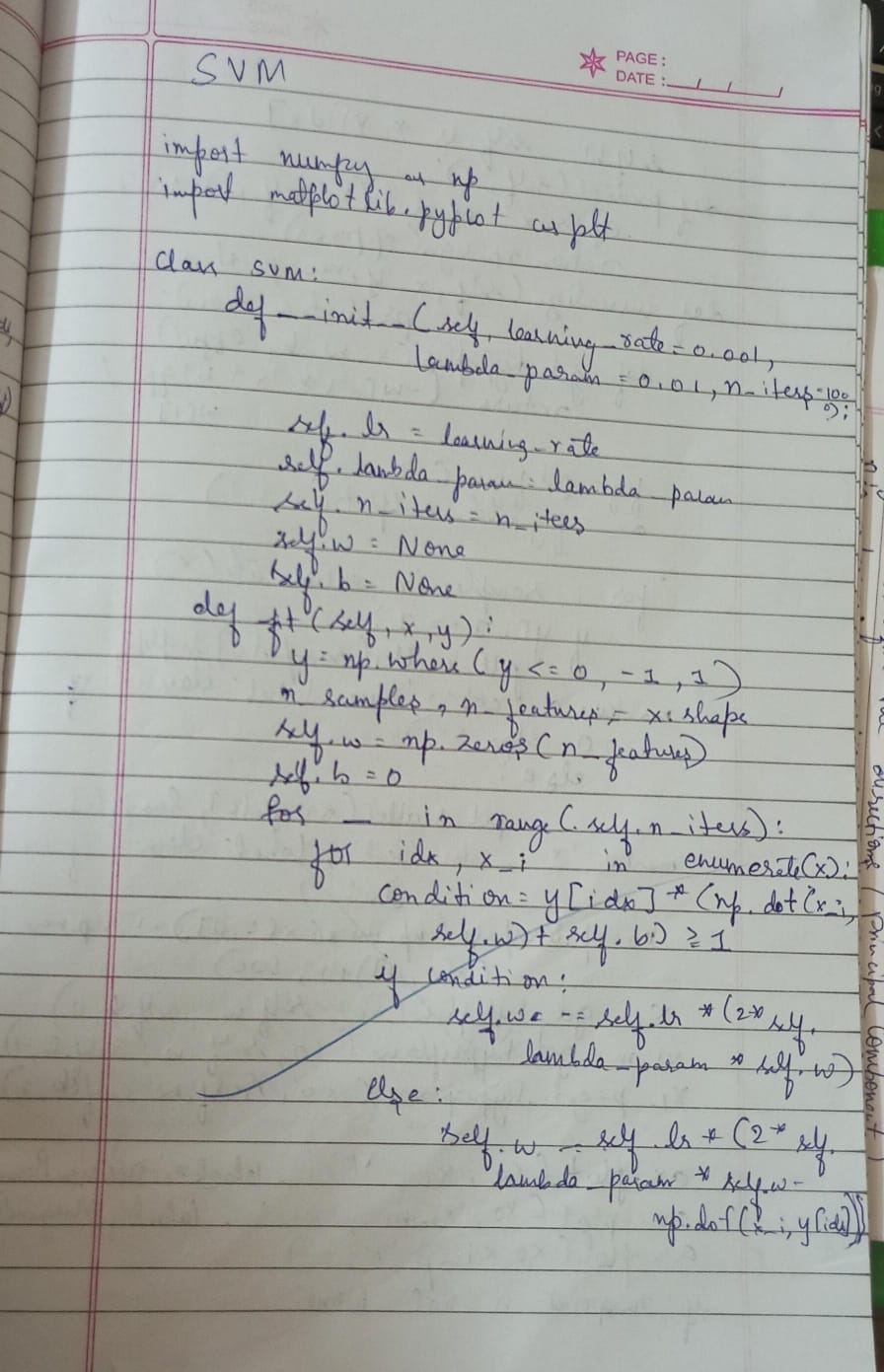
plt.ylabel("Actual")

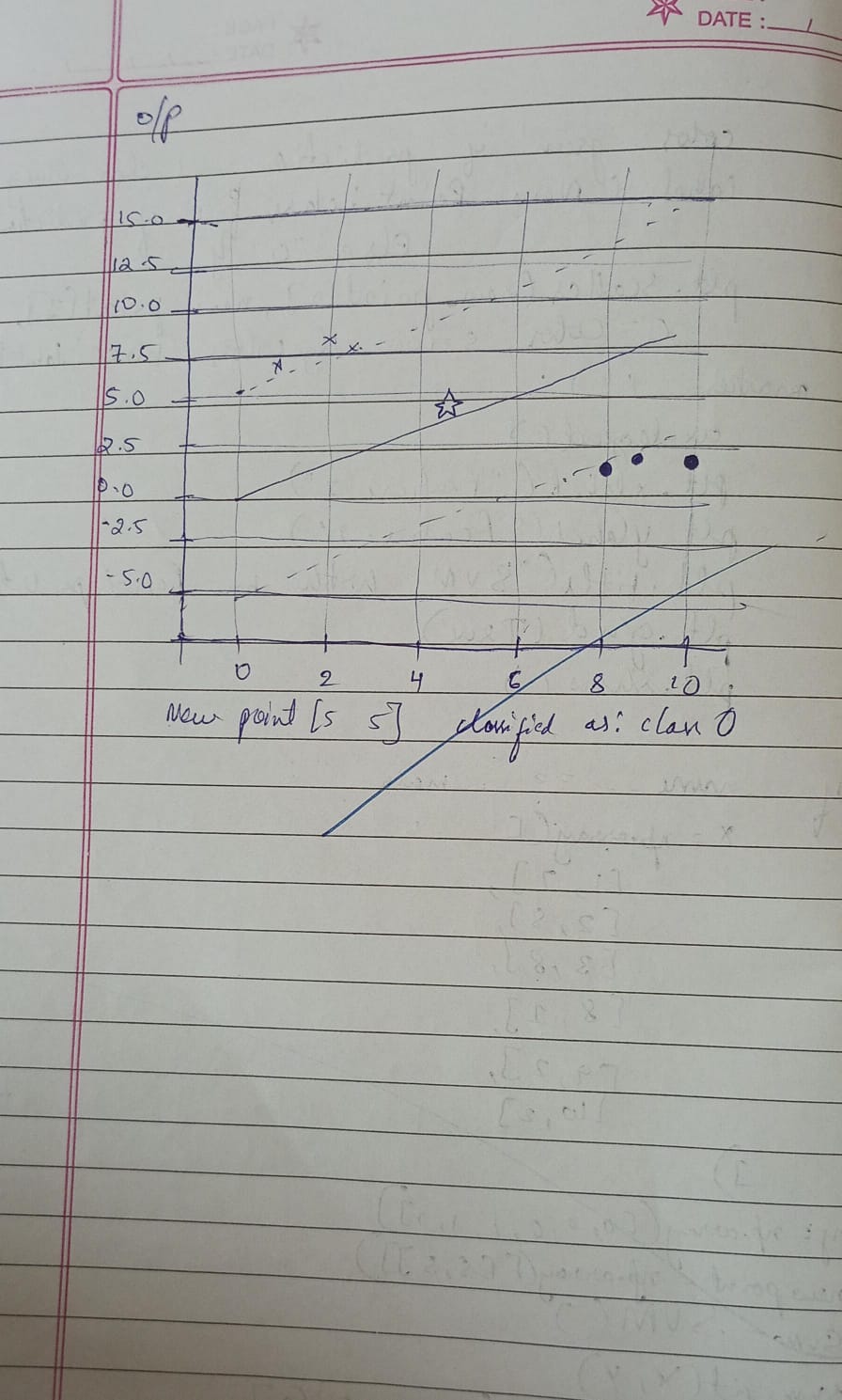
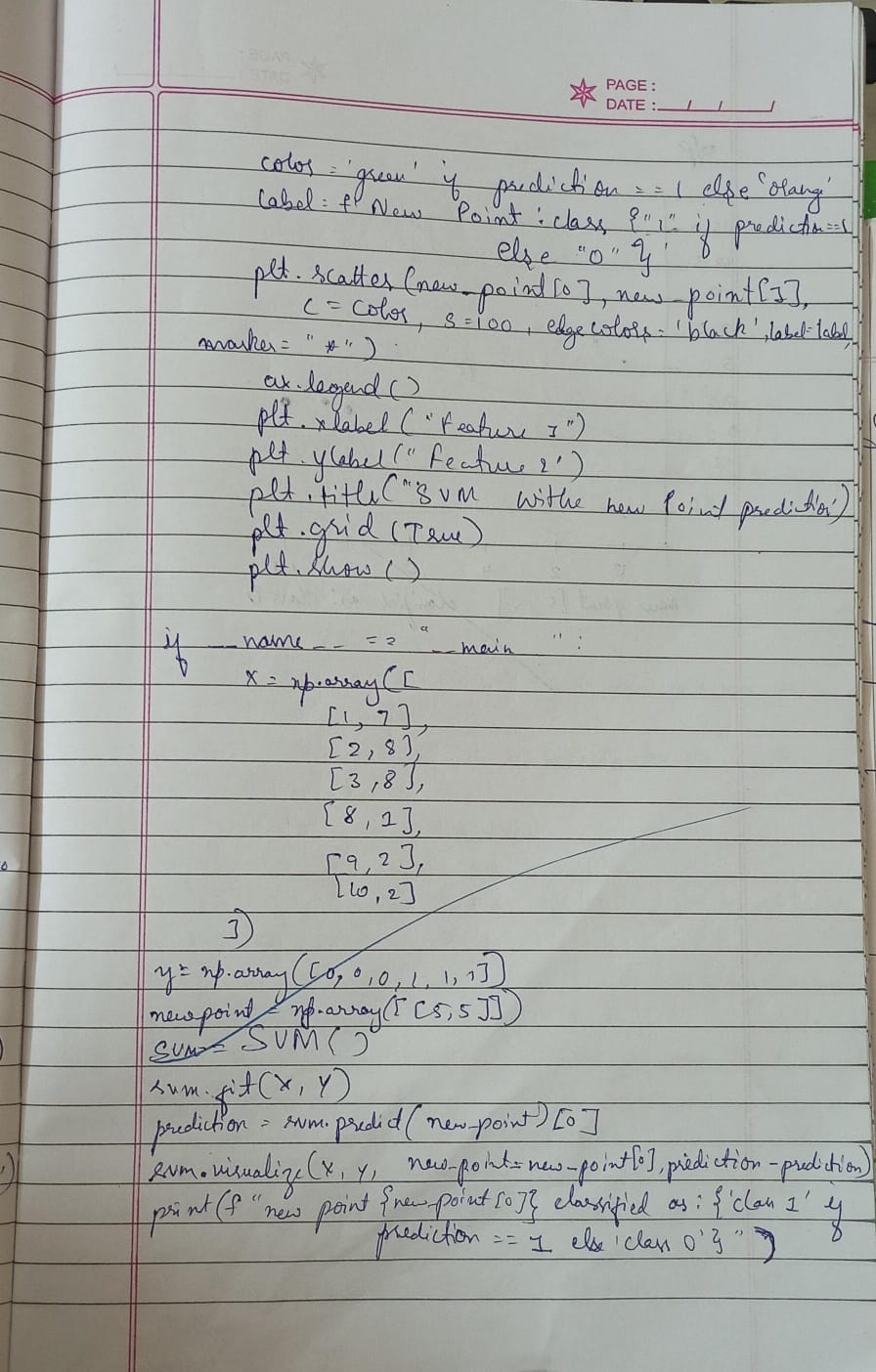
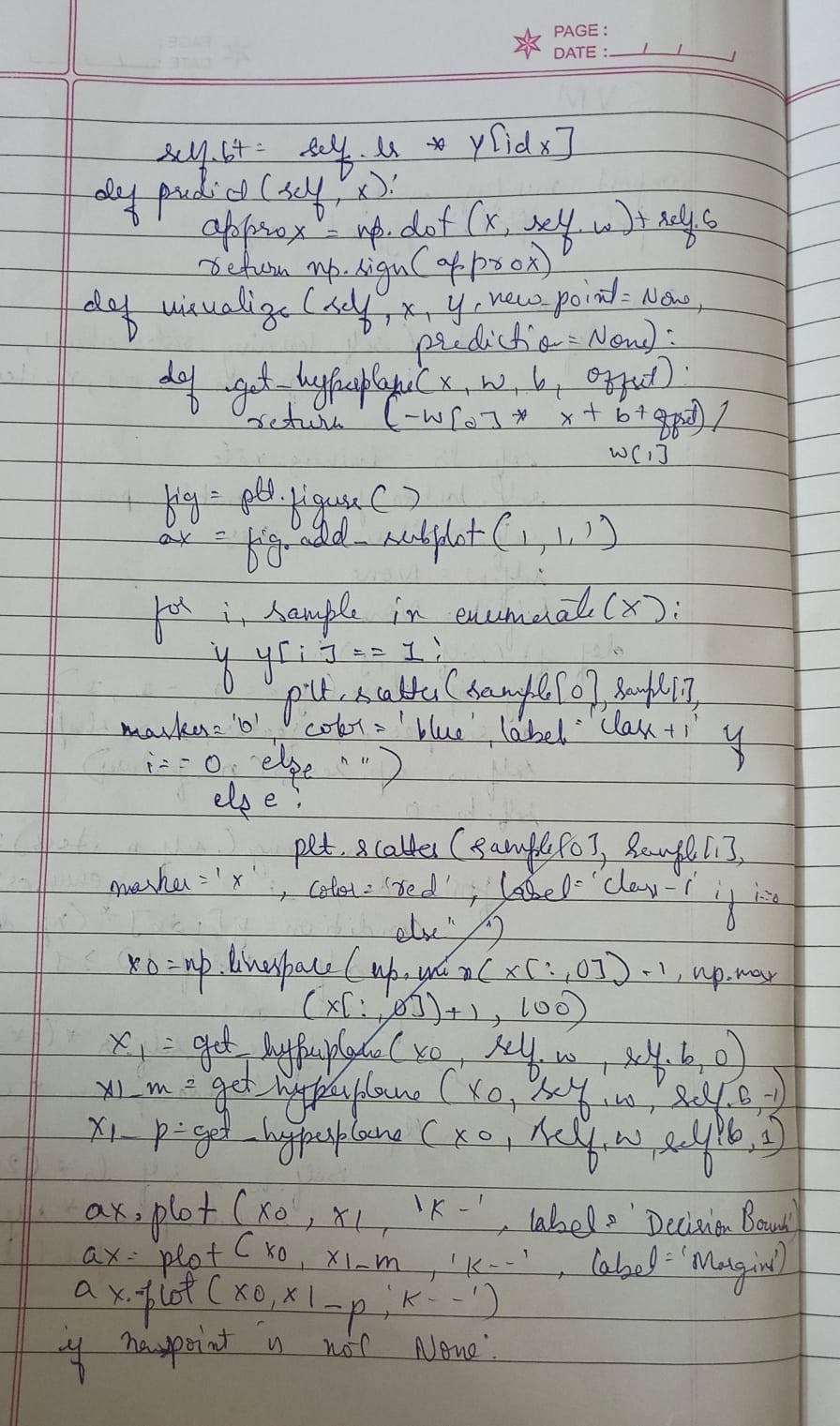
plt.show()

**Program 7**

Build Support vector machine 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 LabelEncoder, StandardScaler

from sklearn.svm import SVC

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

from sklearn.preprocessing import label\_binarize

import matplotlib.pyplot as plt

import seaborn as sns

iris = pd.read\_csv("iris.csv")

label\_encoder = LabelEncoder()

iris['species'] = label\_encoder.fit\_transform(iris['species'])

class\_names\_iris = label\_encoder.classes\_

X\_iris = iris.drop('species', axis=1)

y\_iris = iris['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)

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

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

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

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

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

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

sns.heatmap(cm\_linear, annot=True, fmt='d', cmap='Blues', xticklabels=class\_names\_iris, yticklabels=class\_names\_iris)

plt.title(f'IRIS SVM Linear Kernel\nAccuracy: {acc\_linear:.2f}')

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.tight\_layout()

plt.show()

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

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

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

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

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

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

sns.heatmap(cm\_rbf, annot=True, fmt='d', cmap='Greens', xticklabels=class\_names\_iris, yticklabels=class\_names\_iris)

plt.title(f'IRIS SVM RBF Kernel\nAccuracy: {acc\_rbf:.2f}')

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.tight\_layout()

plt.show()

letters = pd.read\_csv("letter-recognition.csv")

X\_letters = letters.drop('letter', axis=1)

y\_letters = letters['letter']

label\_encoder\_letters = LabelEncoder()

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

class\_names\_letters = label\_encoder\_letters.classes\_

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

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

scaler\_letters = StandardScaler()

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

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

svm\_letters = SVC(kernel='rbf', probability=True)

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

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

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

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

plt.figure(figsize=(14, 12))

sns.heatmap(cm\_letters, annot=True, fmt='d', cmap='Purples',

xticklabels=class\_names\_letters,

yticklabels=class\_names\_letters,

annot\_kws={"size": 8},

cbar=True)

plt.title(f'Letter Recognition - SVM RBF Kernel\nAccuracy: {acc\_letters\*100:.2f}%', fontsize=16)

plt.xlabel("Predicted Label", fontsize=14)

plt.ylabel("True Label", fontsize=14)

plt.xticks(rotation=45)

plt.yticks(rotation=0)

plt.tight\_layout()

plt.show()

y\_test\_binarized = label\_binarize(y\_test\_letters, classes=np.arange(len(class\_names\_letters)))

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

auc\_score = roc\_auc\_score(y\_test\_binarized, y\_score, average='macro')

fpr = dict()

tpr = dict()

for i in range(len(class\_names\_letters)):

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

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

for i in range(0, len(class\_names\_letters), 4): # Plot every 4th class

plt.plot(fpr[i], tpr[i], lw=1.5, label=f'Class {class\_names\_letters[i]}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title(f"Multi-Class ROC Curve (Macro AUC = {auc\_score:.6f})")

plt.legend(loc="lower right", fontsize='small')

plt.grid()

plt.tight\_layout()

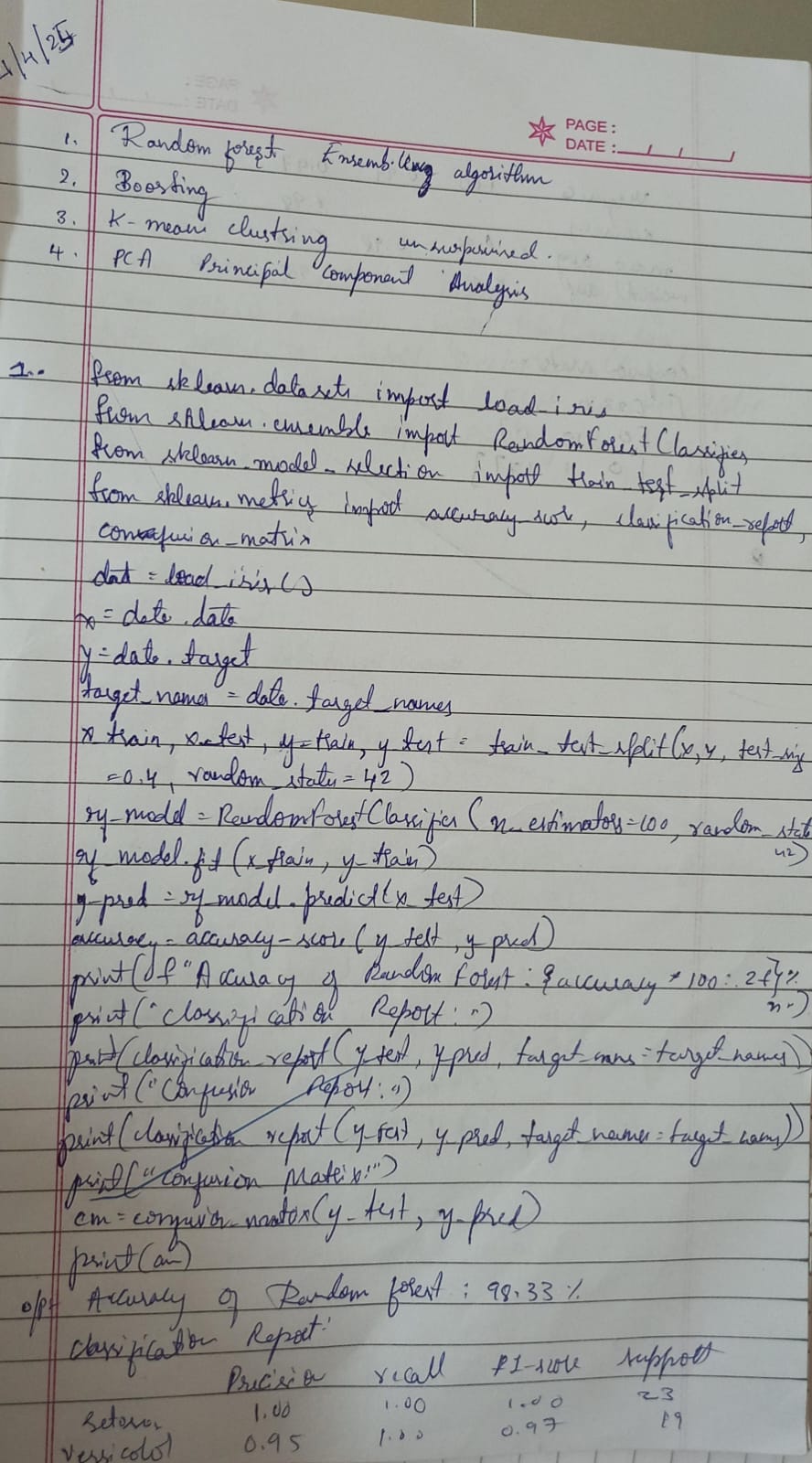
plt.show()

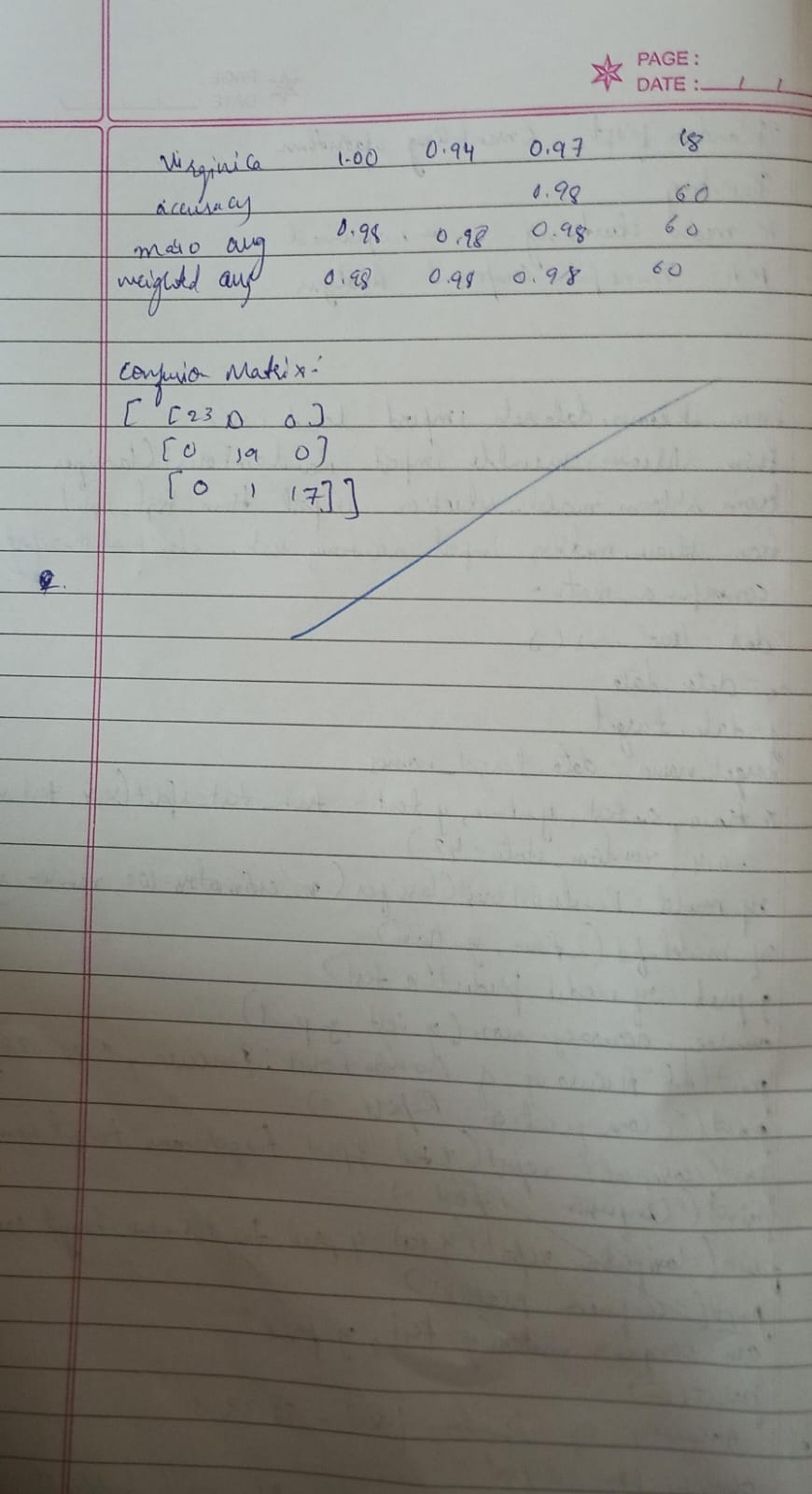
print(f"Exact AUC Score = {auc\_score}")

**Program 8**

Implement Random forest ensemble method on a given dataset.

**Screenshot:**





**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix

iris\_df = pd.read\_csv("iris.csv")

X = iris\_df.drop('species', axis=1)

y = iris\_df['species']

le = LabelEncoder()

y\_encoded = le.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.3, random\_state=42)

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

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

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

print("Random Forest Accuracy with 10 trees:", accuracy\_score(y\_test, y\_pred))

scores = []

n\_range = range(1, 101)

best\_model = None

best\_preds = None

for n in n\_range:

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

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

preds = model.predict(X\_test)

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

scores.append(acc)

if acc == max(scores):

best\_model = model

best\_preds = preds

best\_score = max(scores)

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

print(f"Best Random Forest Accuracy: {best\_score:.4f} with {best\_n} trees")

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

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

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

plt.xlabel('Number of Trees')

plt.ylabel('Accuracy')

plt.grid(True)

plt.show()

cm = confusion\_matrix(y\_test, best\_preds)

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes\_, yticklabels=le.classes\_)

plt.title(f"Confusion Matrix for Best Random Forest Model ({best\_n} Trees)")

plt.xlabel("Predicted")

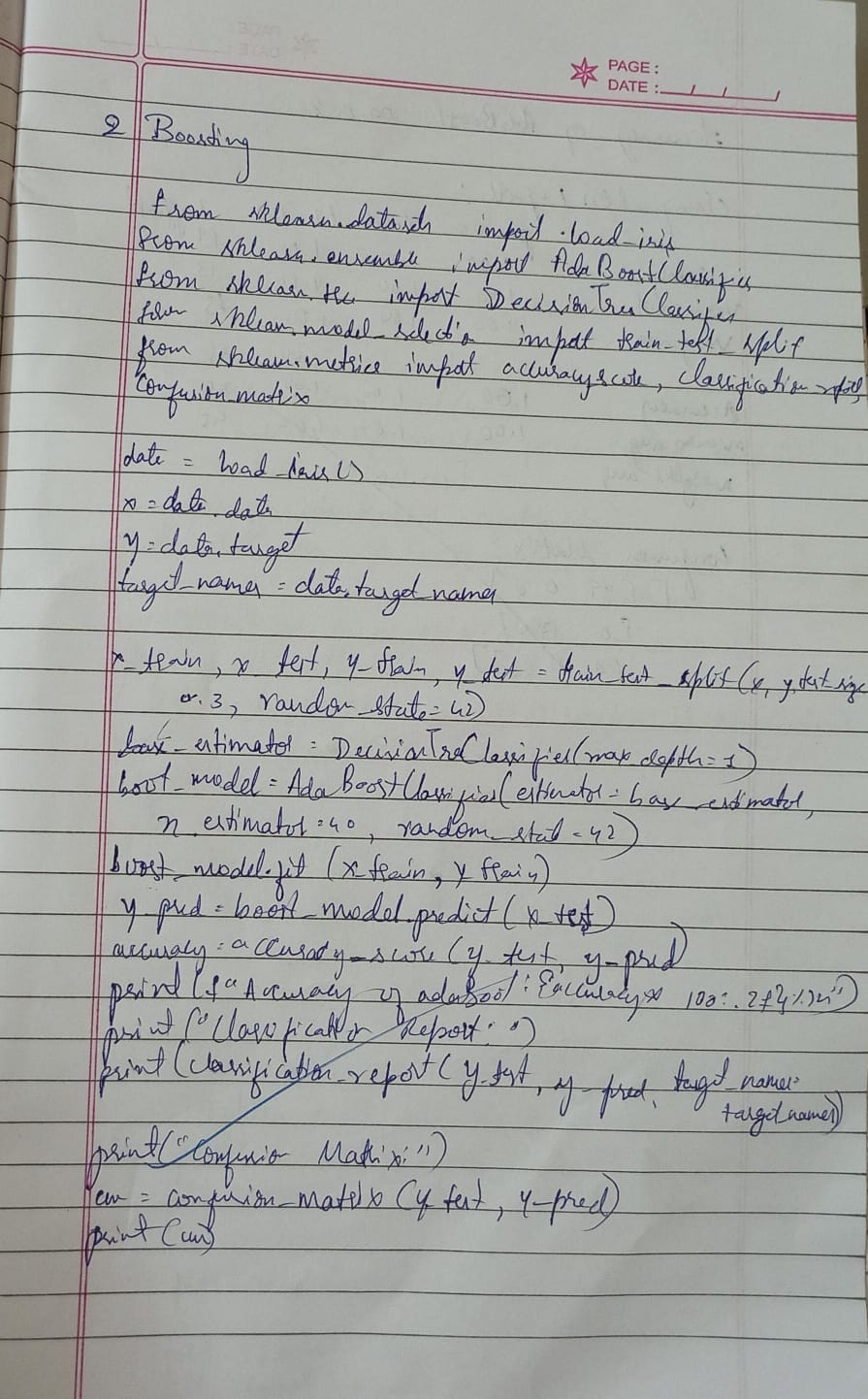
plt.ylabel("Actual")

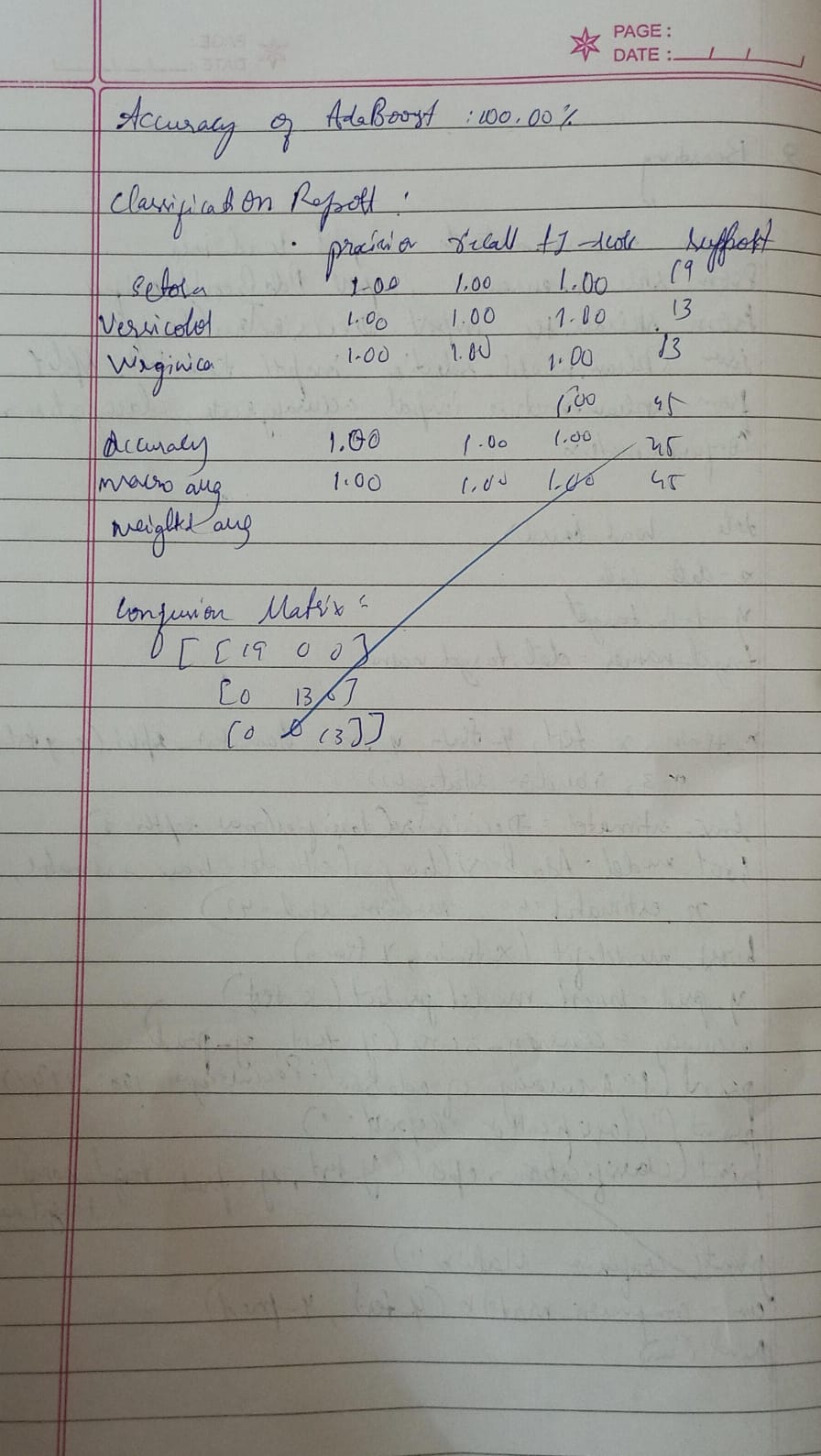
plt.show()

**Program 9**

Implement Boosting ensemble method on a given dataset.

**Screenshot:**





**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import AdaBoostClassifier

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

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix

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

X\_income = income\_df.drop('income\_level', axis=1)

y\_income = income\_df['income\_level']

X\_train\_i, X\_test\_i, y\_train\_i, y\_test\_i = train\_test\_split(X\_income, y\_income, test\_size=0.3, random\_state=42)

ada\_model = AdaBoostClassifier(n\_estimators=10, random\_state=42)

ada\_model.fit(X\_train\_i, y\_train\_i)

y\_pred\_i = ada\_model.predict(X\_test\_i)

print("AdaBoost Accuracy with 10 estimators:", accuracy\_score(y\_test\_i, y\_pred\_i))

scores\_ada = []

n\_range\_ada = range(1, 51)

best\_model\_ada = None

best\_preds\_ada = None

for n in n\_range\_ada:

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

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

preds = model.predict(X\_test\_i)

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

scores\_ada.append(acc)

if acc == max(scores\_ada):

best\_model\_ada = model

best\_preds\_ada = preds

best\_score\_ada = max(scores\_ada)

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

print(f"Best AdaBoost Accuracy: {best\_score\_ada:.4f} with {best\_n\_ada} estimators")

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

plt.plot(n\_range\_ada, scores\_ada, marker='o', linestyle='-', color='orange')

plt.title('AdaBoost Accuracy vs Number of Estimators (Income Dataset)')

plt.xlabel('Number of Estimators')

plt.ylabel('Accuracy')

plt.grid(True)

plt.show()

cm\_ada = confusion\_matrix(y\_test\_i, best\_preds\_ada)

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

sns.heatmap(cm\_ada, annot=True, fmt='d', cmap='Oranges', xticklabels=[0, 1], yticklabels=[0, 1])

plt.title(f"Confusion Matrix for Best AdaBoost Model ({best\_n\_ada} Estimators)")

plt.xlabel("Predicted")

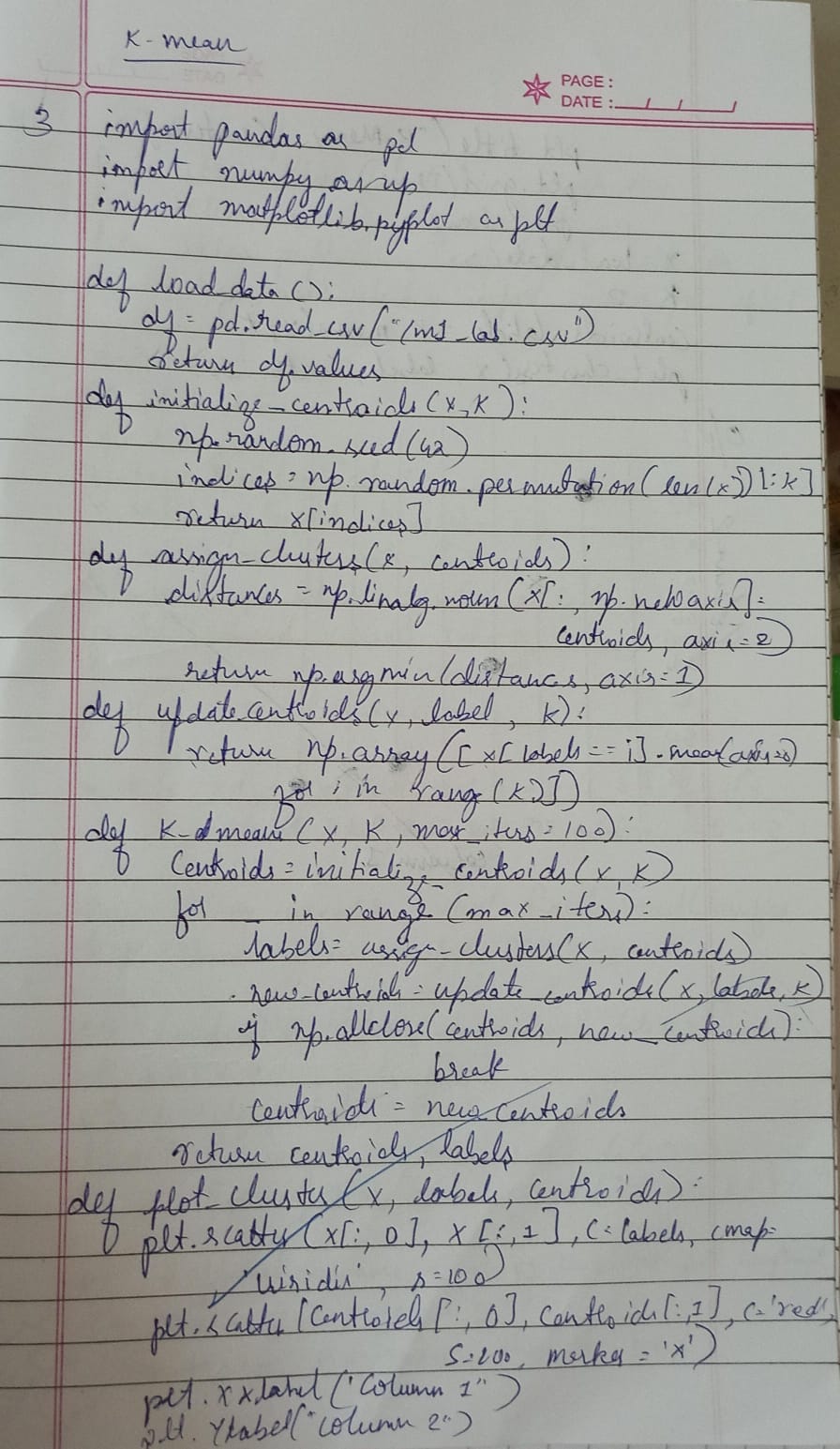
plt.ylabel("Actual")

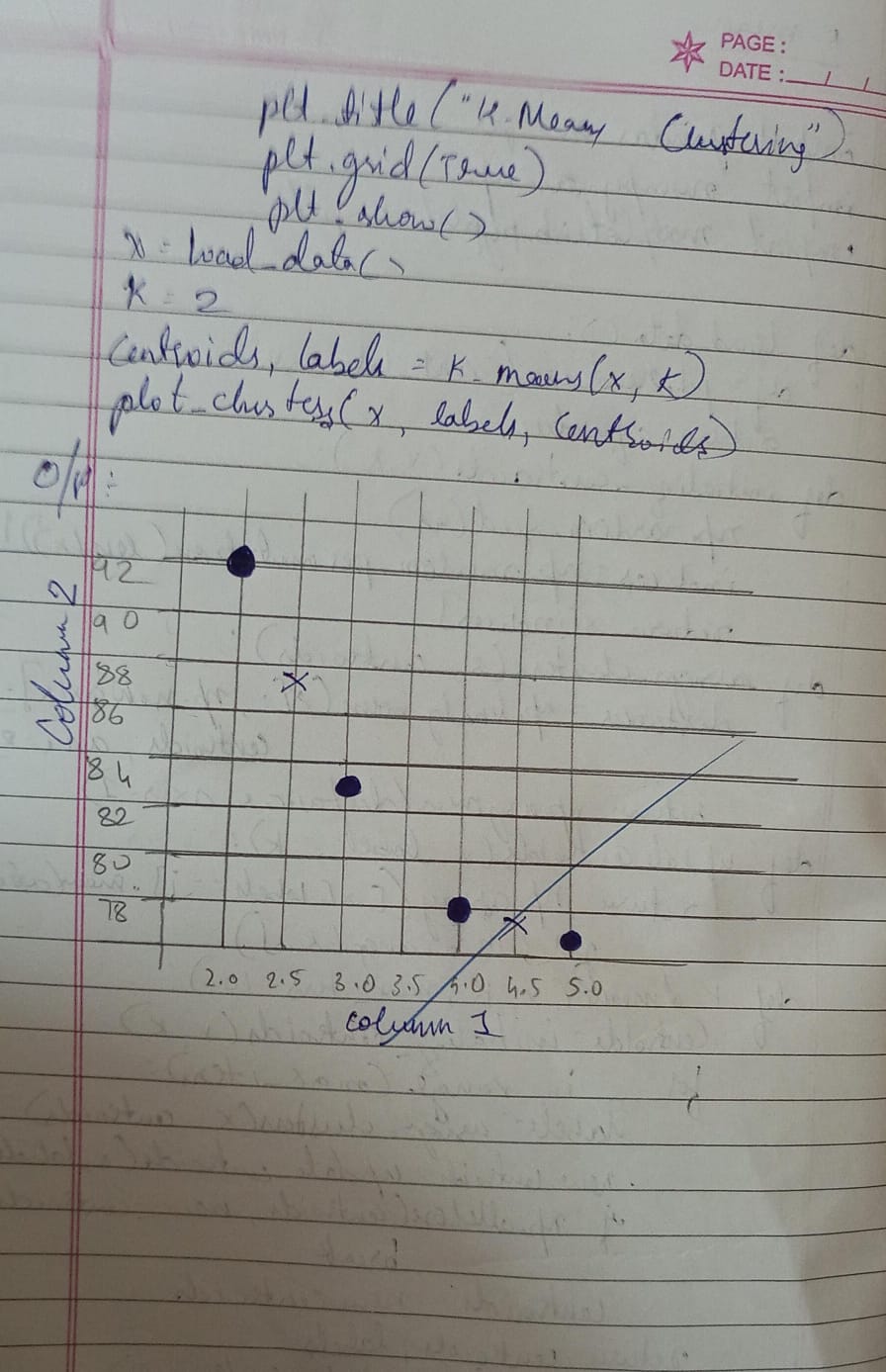
plt.show()

**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.model\_selection import train\_test\_split

from scipy import stats

import seaborn as sns

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

from sklearn.metrics import accuracy\_score

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

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

df1=pd.read\_csv("iris.csv")

df1.head()

df = df1.drop(['sepal\_length','sepal\_width','species'],axis=1)

scaler = StandardScaler()

scaled\_df = scaler.fit\_transform(df)

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

kmeans.fit(scaled\_df)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

plt.title('Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

kmeans = KMeans(n\_clusters=3, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

pred\_y = kmeans.fit\_predict(scaled\_df)

df['cluster'] = pred\_y

plt.scatter(df['petal\_length'], df['petal\_width'], c=df['cluster'])

plt.title('Clusters of Iris Flowers')

plt.xlabel('Petal Length')

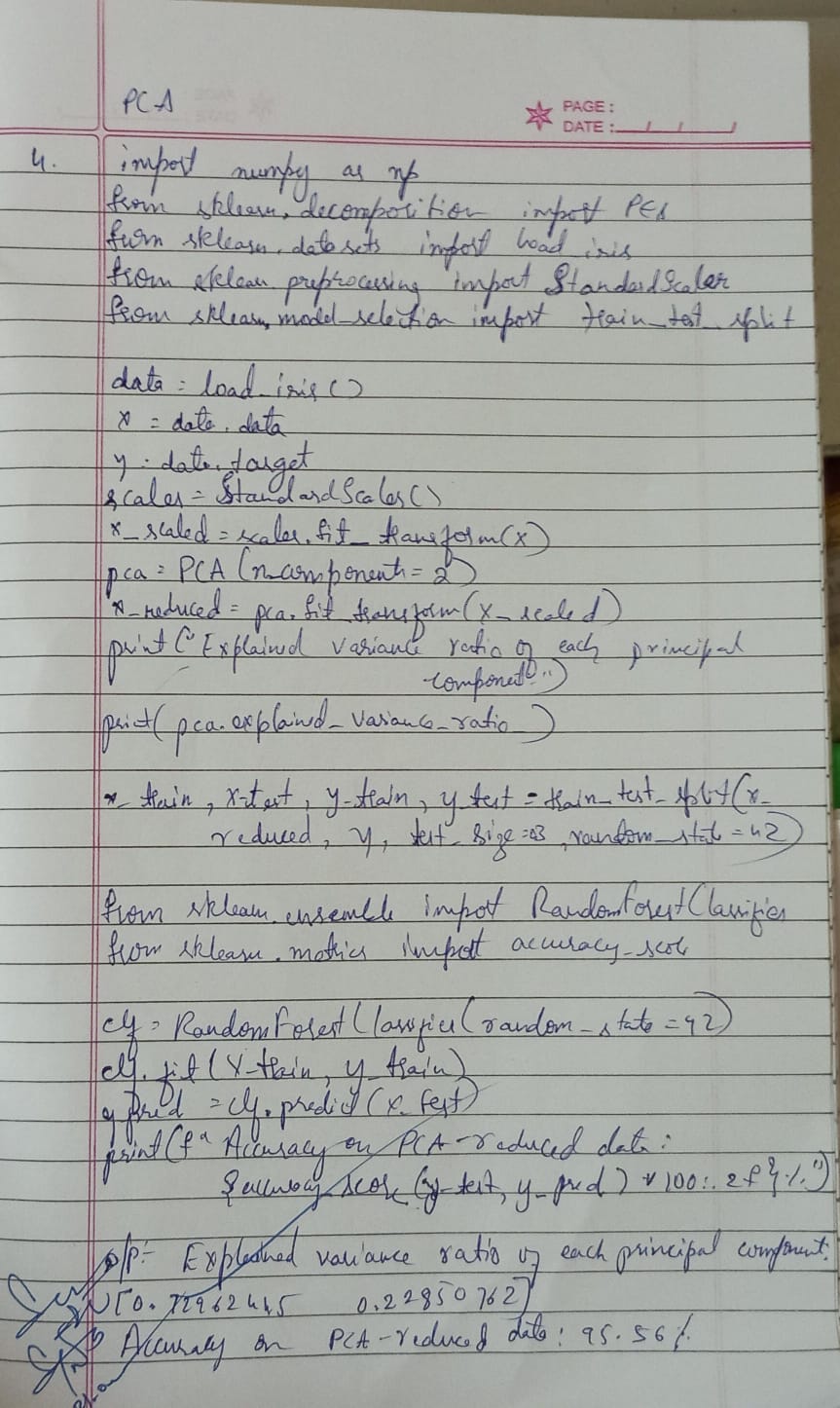
plt.ylabel('Petal Width')

plt.show()

**Program 11**

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

**Screenshot:**



**Code:**

from google.colab import files

heart=files.upload()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from scipy import stats

import seaborn as sns

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

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

from sklearn.metrics import accuracy\_score

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

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

df1=pd.read\_csv("heart.csv")

df1.head()

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

label\_encoder = LabelEncoder()

for col in text\_cols:

df1[col] = label\_encoder.fit\_transform(df1[col])

print(df1.head())

X = df1.drop('HeartDisease', axis=1)

y = df1['HeartDisease']

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

scaler = StandardScaler()

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

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

*# Support Vector Machine*

svm\_model = SVC(kernel='linear', random\_state=42)

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

svm\_predictions = svm\_model.predict(X\_test)

svm\_accuracy = accuracy\_score(y\_test, svm\_predictions)

print(f"SVM Accuracy: {svm\_accuracy}")

*# Logistic Regression*

lr\_model = LogisticRegression(random\_state=42)

lr\_model.fit(X\_train, y\_train)

lr\_predictions = lr\_model.predict(X\_test)

lr\_accuracy = accuracy\_score(y\_test, lr\_predictions)

print(f"Logistic Regression Accuracy: {lr\_accuracy}")

*# Random Forest*

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

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

rf\_predictions = rf\_model.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, rf\_predictions)

print(f"Random Forest Accuracy: {rf\_accuracy}")

models = {

"SVM": svm\_accuracy,

"Logistic Regression": lr\_accuracy,

"Random Forest": rf\_accuracy}

best\_model = max(models, key=models.get)

print(f"\nBest Model: {best\_model} with accuracy {models[best\_model]}")

pca = PCA(n\_components=0.95)

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

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

svm\_model\_pca = SVC(kernel='linear', random\_state=42)

svm\_model\_pca.fit(X\_train\_pca, y\_train)

svm\_predictions\_pca = svm\_model\_pca.predict(X\_test\_pca)

svm\_accuracy\_pca = accuracy\_score(y\_test, svm\_predictions\_pca)

print(f"SVM Accuracy (with PCA): {svm\_accuracy\_pca}")

lr\_model\_pca = LogisticRegression(random\_state=42)

lr\_model\_pca.fit(X\_train\_pca, y\_train)

lr\_predictions\_pca = lr\_model\_pca.predict(X\_test\_pca)

lr\_accuracy\_pca = accuracy\_score(y\_test, lr\_predictions\_pca)

print(f"Logistic Regression Accuracy (with PCA): {lr\_accuracy\_pca}")

rf\_model\_pca = RandomForestClassifier(random\_state=42)

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

rf\_predictions\_pca = rf\_model\_pca.predict(X\_test\_pca)

rf\_accuracy\_pca = accuracy\_score(y\_test, rf\_predictions\_pca)

print(f"Random Forest Accuracy (with PCA): {rf\_accuracy\_pca}")

models\_pca = {

"SVM": svm\_accuracy\_pca,

"Logistic Regression": lr\_accuracy\_pca,

"Random Forest": rf\_accuracy\_pca}

best\_model\_pca = max(models\_pca, key=models\_pca.get)

print(f"\nBest Model (with PCA): {best\_model\_pca} with accuracy {models\_pca[best\_model\_pca]}")