**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

Machine Learning (23CS6PCMAL)

***Submitted by***

**Pooja Gaikwad (1BM22CS194)**

***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 **Pooja Gaaikwad (1BM22CS194),** 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.

Geetha N Dr. Kavitha Sooda

Assistant Professor Professor & HOD

Department of CSE , BMSCE 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 | 11 |
| 4 | 17-3-2025 | Build Logistic Regression Model for a given dataset | 16 |
| 5 | 24-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 20 |
| 6 | 7-4-2025 | Build KNN Classification model for a given dataset. | 24 |
| 7 | 21-4-2025 | Build Support vector machine model for a given dataset | 27 |
| 8 | 5-5-2025 | Implement Random forest ensemble method on a given dataset. | 31 |
| 9 | 5-5-2025 | Implement Boosting ensemble method on a given dataset. | 33 |
| 10 | 12-5-2025 | Build k-Means algorithm to cluster a set of data stored in a  .CSV file. | 35 |
| 11 | 12-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 37 |

**Github Link:**

https://github.com/poojagaikwad10/Machine-Learning

**Program 1**

**Write a python program to import and export data using Pandas library functions 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()) file\_path = 'data.csv'

df = pd.read\_csv(file\_path) print("Sample data:") print(df.head())

print("\n")

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())

from sklearn.datasets import load\_diabetes iris = load\_diabetes()

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

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'.")

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)

print("\n")

reliance\_data = data['ICICIBANK.NS'] print("\nSummary statistics for ICICI 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="ICICI Industries - Closing Price") plt.subplot(2, 1, 2)

reliance\_data['Daily Return'].plot(title="ICICI Industries - Daily Returns", color='BLACK') plt.tight\_layout()

plt.show() reliance\_data.to\_csv('icici\_stock\_data.csv')

print("\nicici stock data saved to 'icici\_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['KOTAKBANK.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="KOTAK Industries - Closing Price") plt.subplot(2, 1, 2)

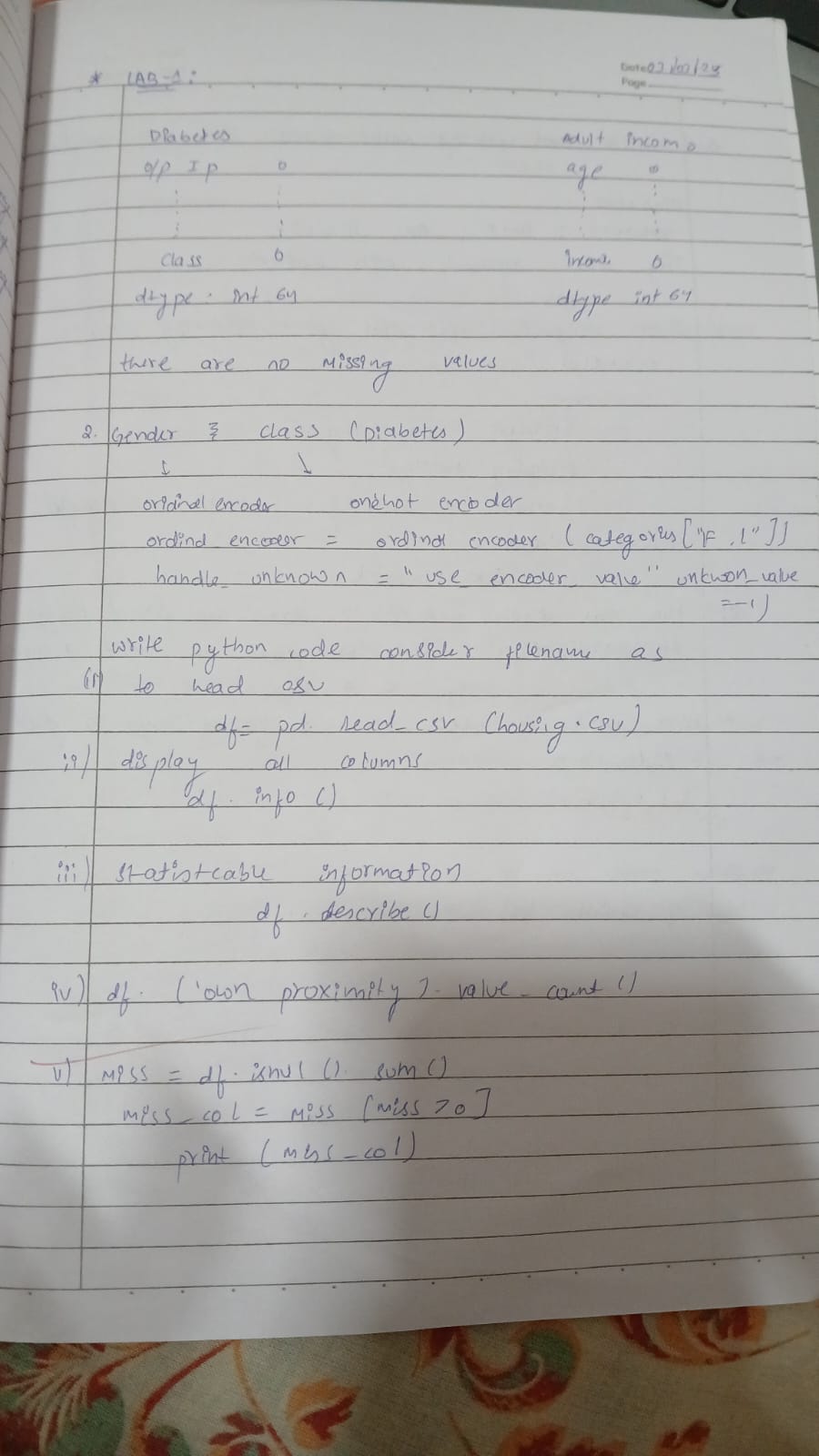
reliance\_data['Daily Return'].plot(title="kotak Industries - Daily Returns", color='red') plt.tight\_layout()

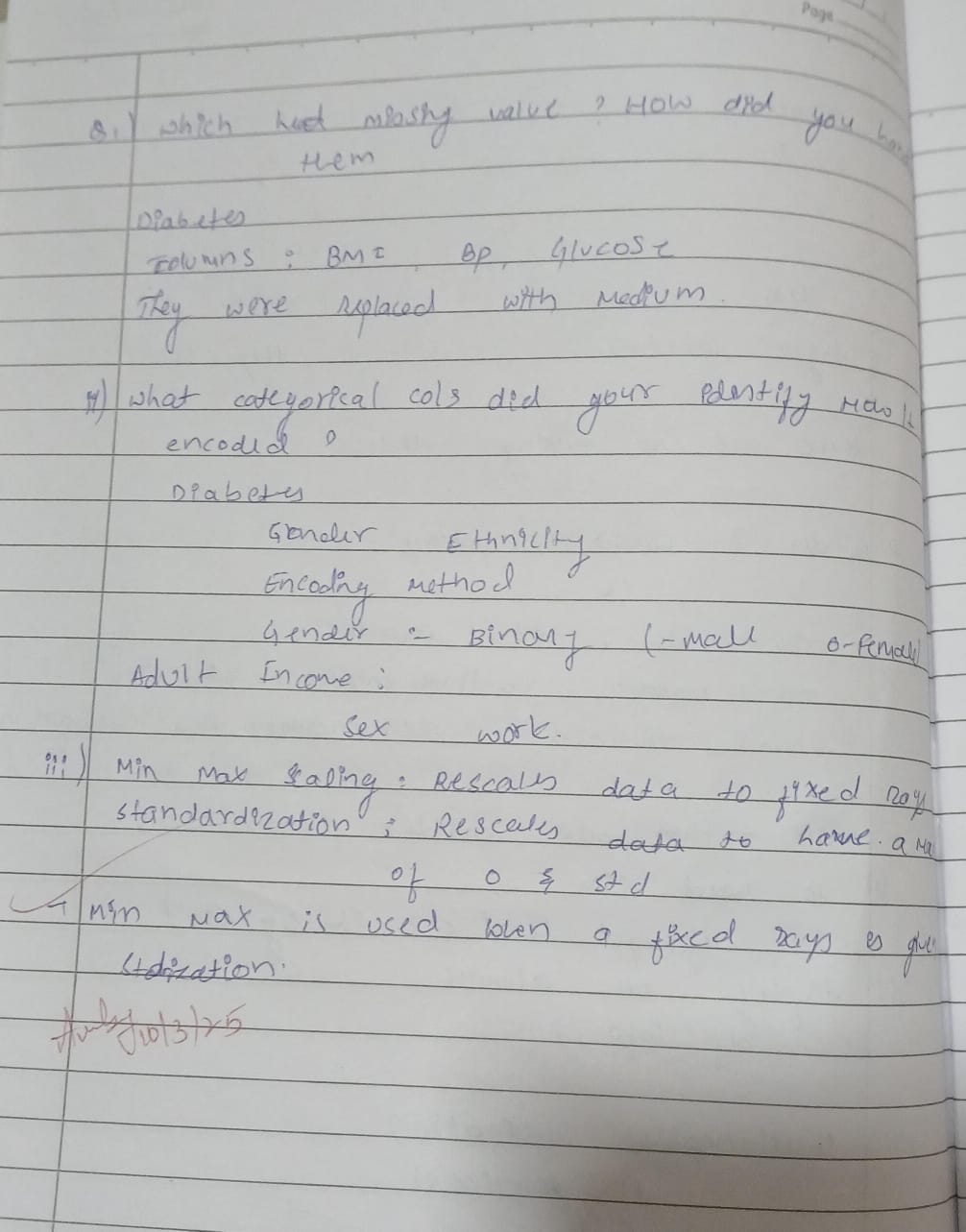
plt.show() reliance\_data.to\_csv('kotak\_stock\_data.csv')

print("\nkotak stock data saved to 'kotak\_stock\_data.csv'.")

**Program 2**

**Demonstrate various data pre-processing techniques for a given dataset Screenshot:**





**Code:**

from google.colab import files diabetes=files.upload()

from google.colab import files adult\_income=files.upload()

df1=pd.read\_csv("Dataset of Diabetes .csv") df1.head()

df2=pd.read\_csv("adult.csv") df2.head()

df1.info() df2.info()

df1.describe() df2.describe()

missing\_values1 = df1.isnull().sum() print(missing\_values1) missing\_values2 = df2.isnull().sum() print(missing\_values2)

df1['Gender'] = df1['Gender'].replace('f', 'F') ordinal\_encoder = OrdinalEncoder(categories=[["F", "M"]])

df1["Gender\_Encoded"] = ordinal\_encoder.fit\_transform(df1[["Gender"]]) onehot\_encoder = OneHotEncoder()

encoded\_data = onehot\_encoder.fit\_transform(df1[["CLASS"]]) encoded\_array = encoded\_data.toarray()

encoded\_df = pd.DataFrame(encoded\_array, columns=onehot\_encoder.get\_feature\_names\_out(["CLASS"])) df\_encoded = pd.concat([df1, encoded\_df], axis=1)

df1 = pd.concat([df1, encoded\_df], axis=1) df1.drop("CLASS", axis=1, inplace=True) df1.drop("Gender", axis=1, inplace=True) print(df2.head())

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder df\_copy2 = df2

ordinal\_encoder = OrdinalEncoder(categories=[["Male","Female"]]) df\_copy2["Gender\_Encoded"] = ordinal\_encoder.fit\_transform(df\_copy2[["gender"]]) print(df\_copy2[["gender","Gender\_Encoded"]])

onehot\_encoder = OneHotEncoder() encoded\_data =

onehot\_encoder.fit\_transform(df2[["occupation","workclass","education","marital-status","relationship","race","n ative-country","income"]])

encoded\_array = encoded\_data.toarray() encoded\_df = pd.DataFrame(encoded\_array,

columns=onehot\_encoder.get\_feature\_names\_out(["occupation","workclass","education","marital-status","relatio nship","race","native-country","income"]))

df\_encoded = pd.concat([df\_copy2, encoded\_df], axis=1)

df\_encoded.drop("gender", axis=1, inplace=True) df\_encoded.drop("occupation", axis=1, inplace=True) df\_encoded.drop("workclass", axis=1, inplace=True) df\_encoded.drop("education", axis=1, inplace=True) df\_encoded.drop("marital-status", axis=1, inplace=True) df\_encoded.drop("relationship", axis=1, inplace=True) df\_encoded.drop("race", axis=1, inplace=True) df\_encoded.drop("native-country", axis=1, inplace=True) df\_encoded.drop("income", axis=1, inplace=True) print(df\_encoded. head())

normalizer = MinMaxScaler()

df\_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per-week"]] = normalizer.fit\_transform(df\_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per-week"]

])

df\_encoded.head()

normalizer = MinMaxScaler()

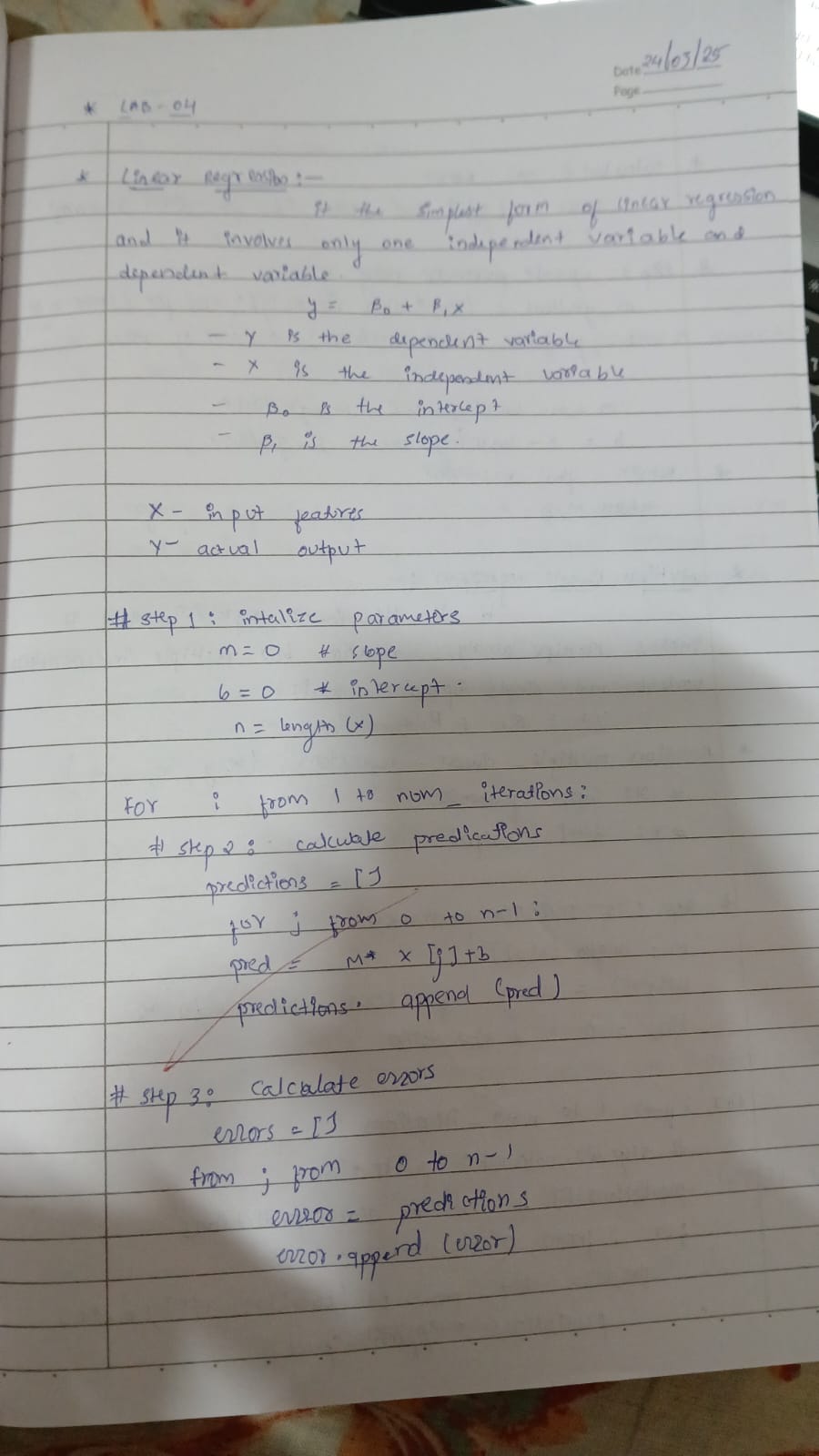
df1[["No\_Pation","AGE","Urea","Cr" , "HbA1c" , "Chol","TG","HDL","LDL","VLDL","BMI"]] =

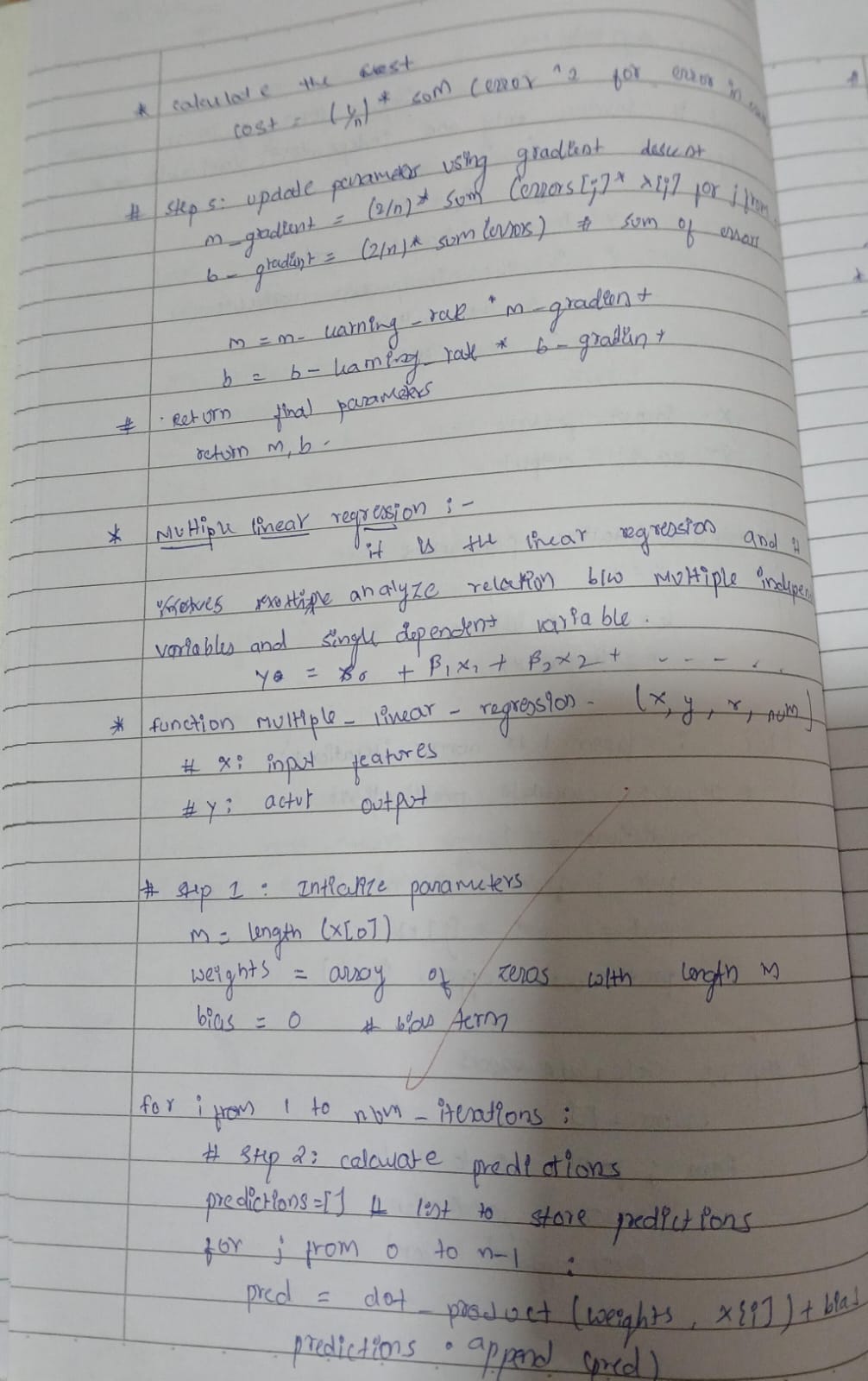
normalizer.fit\_transform(df1[["No\_Pation","AGE","Urea","Cr" , "HbA1c" , "Chol","TG","HDL","LDL","VLDL","BMI"]])

df1.head()

**Program 3**

**Implement Linear and Multi-Linear Regression algorithm using appropriate dataset Screenshot:**





**Code:**

from google.colab import files per\_capita\_income=files.upload()

from google.colab import files salary=files.upload()

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

from sklearn import linear\_model

df1=pd.read\_csv("canada\_per\_capita\_income.csv") df1.head()

df2=pd.read\_csv("salary.csv") df2.head()

df2.YearsExperience.median()

df2.YearsExperience = df2.YearsExperience.fillna(df2.YearsExperience.median()) df2

plt.xlabel("year")

plt.ylabel("per capita income (US$)") plt.scatter(df1.year, df1['per capita income (US$)'])

plt.xlabel("YearsExperience") plt.ylabel("Salary") plt.scatter(df2.YearsExperience, df2.Salary)

reg1 = linear\_model.LinearRegression() reg1.intercept\_

df2.YearsExperience.median()

df2.YearsExperience = df2.YearsExperience.fillna(df2.YearsExperience.median()) df2

plt.xlabel("year")

plt.ylabel("per capita income (US$)") plt.scatter(df1.year, df1['per capita income (US$)'])

plt.xlabel("YearsExperience") plt.ylabel("Salary") plt.scatter(df2.YearsExperience, df2.Salary)

reg1 = linear\_model.LinearRegression() reg1.intercept\_

reg1.predict([[2020]])

reg2 = linear\_model.LinearRegression() reg2.fit(df2.drop('Salary', axis='columns'), df2['Salary']) reg2.coef\_

reg2.intercept\_ reg2.predict([[12]])

from google.colab import files hiring=files.upload()

from google.colab import files companies=files.upload()

df3=pd.read\_csv("hiring.csv") df3.head()

df4=pd.read\_csv("1000\_Companies.csv") df4.head()

df3.isnull().sum()

df4.isnull().sum()

df3\_copy = df3.copy()

experience\_mapping = {'two': 2, 'three': 3, 'five': 5, 'seven': 7, 'ten': 10, 'eleven': 11} df3\_copy['experience'] = df3\_copy['experience'].map(experience\_mapping) median\_experience = df3\_copy['experience'].median()

df3\_copy['experience'] = df3\_copy['experience'].fillna(median\_experience) df3\_copy

df3\_copy['test\_score(out of 10)'] = df3\_copy['test\_score(out of 10)'].fillna(df3\_copy['test\_score(out of 10)'].mean())

reg3 = linear\_model.LinearRegression() reg3.fit(df3\_copy.drop('salary($)', axis='columns'), df3\_copy['salary($)']) reg3.coef\_

reg3.intercept\_ reg3.predict([[2,9,6]])

reg3.predict([[12,10,10]])

ohe = OneHotEncoder(sparse\_output=False, handle\_unknown='ignore') state\_encoded = ohe.fit\_transform(df4[['State']])

state\_encoded\_df = pd.DataFrame(state\_encoded, columns=ohe.get\_feature\_names\_out(['State']))

df4 = pd.concat([df4, state\_encoded\_df], axis=1).drop(columns=['State']) print(df4)

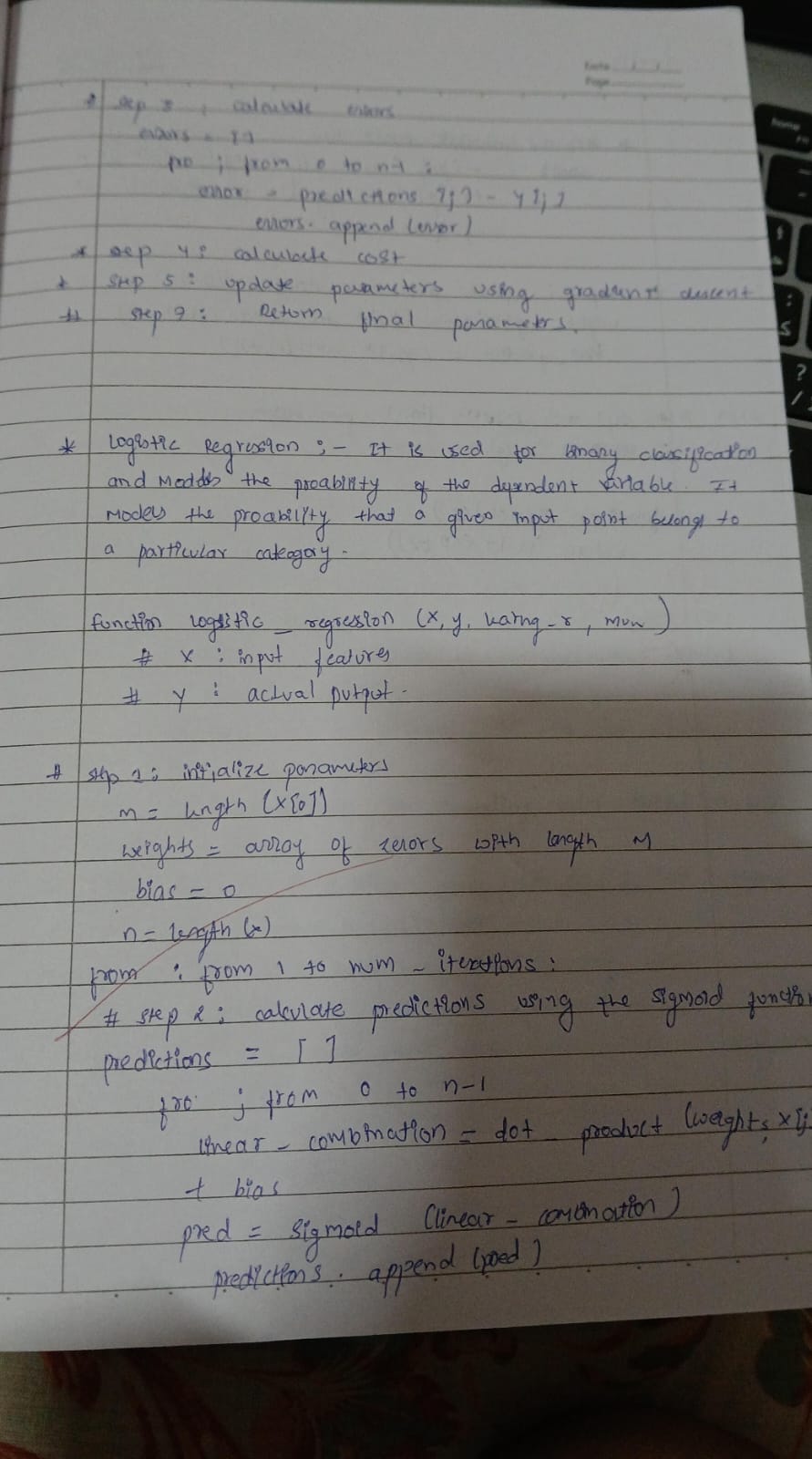
reg4 = linear\_model.LinearRegression() reg4.fit(df4.drop('Profit',axis='columns'),df4.Profit) print(reg4.coef\_)

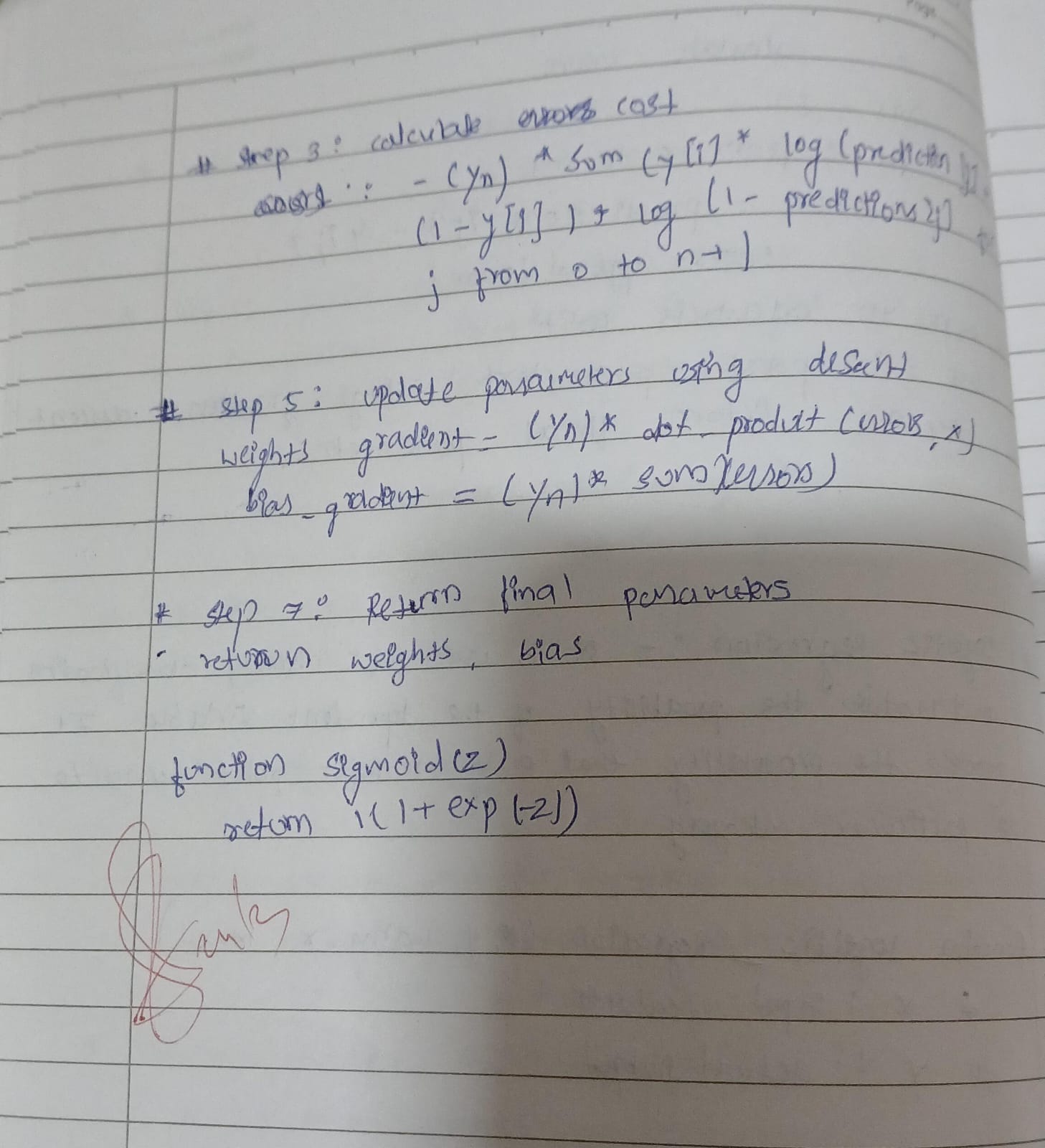
print(reg4.intercept\_)

reg4.predict([[91694.48, 515841.3, 11931.24,0,1,0]])

**Program 4 Build Logistic Regression Model for a given dataset**

**Screenshot:**





**Code:**

from google.colab import files hr=files.upload()

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

from sklearn import linear\_model import seaborn as sns

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn.metrics import accuracy\_score

df1=pd.read\_csv("HR\_comma\_sep.csv") df1.head()

df1.isnull().sum() plt.figure(figsize=(12, 6))

sns.barplot(x='Department', y='left', data=df1) plt.title('Employee Retention Rate by Department') plt.xlabel('Department')

plt.ylabel('Proportion of Employees Left') plt.xticks(rotation=45, ha='right') plt.show()

ohe = OneHotEncoder(handle\_unknown='ignore', sparse\_output=False) department\_encoded = ohe.fit\_transform(df1[['Department']]) department\_encoded\_df = pd.DataFrame(department\_encoded, columns=ohe.get\_feature\_names\_out(['Department']))

df1 = pd.concat([df1, department\_encoded\_df], axis=1) df1 = df1.drop('Department', axis=1)

ordinal\_encoder = OrdinalEncoder(categories=[['low', 'medium', 'high']], dtype=np.int64) salary\_encoded = ordinal\_encoder.fit\_transform(df1[['salary']])

df1['salary\_encoded'] = salary\_encoded df1 = df1.drop('salary', axis=1) df1.head()

correlation\_matrix = df1.corr() plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f") plt.title('Correlation Matrix of Features')

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

sns.barplot(x='salary\_encoded', y='left', data=df1) plt.title('Impact of Employee Salary on Retention') plt.xlabel('Salary Level (Encoded)') plt.ylabel('Proportion of Employees Left') plt.show()

df\_copy = df1[['number\_project', 'average\_montly\_hours', 'time\_spend\_company', 'left','salary\_encoded', 'satisfaction\_level','Work\_accident']]

df\_copy.head()

X = df\_copy.drop('left', axis=1) y = df\_copy['left']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test)

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

print(f"Accuracy of the Logistic Regression model: {accuracy}")

from google.colab import files zoodata=files.upload() zootype=files.upload()

zoo\_data = pd.read\_csv('zoo-data.csv') zoo\_class = pd.read\_csv('zoo-class-type.csv')

merged\_data = pd.merge(zoo\_data, zoo\_class, left\_on='class\_type', right\_on='Class\_Number') merged\_data = merged\_data.drop(['Animal\_Names', 'Number\_Of\_Animal\_Species\_In\_Class', 'Class\_Number','class\_type','animal\_name'], axis=1)

X = merged\_data.drop('Class\_Type', axis=1) y = merged\_data['Class\_Type'] print(merged\_data.head())

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print(f"Accuracy: {accuracy}")

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

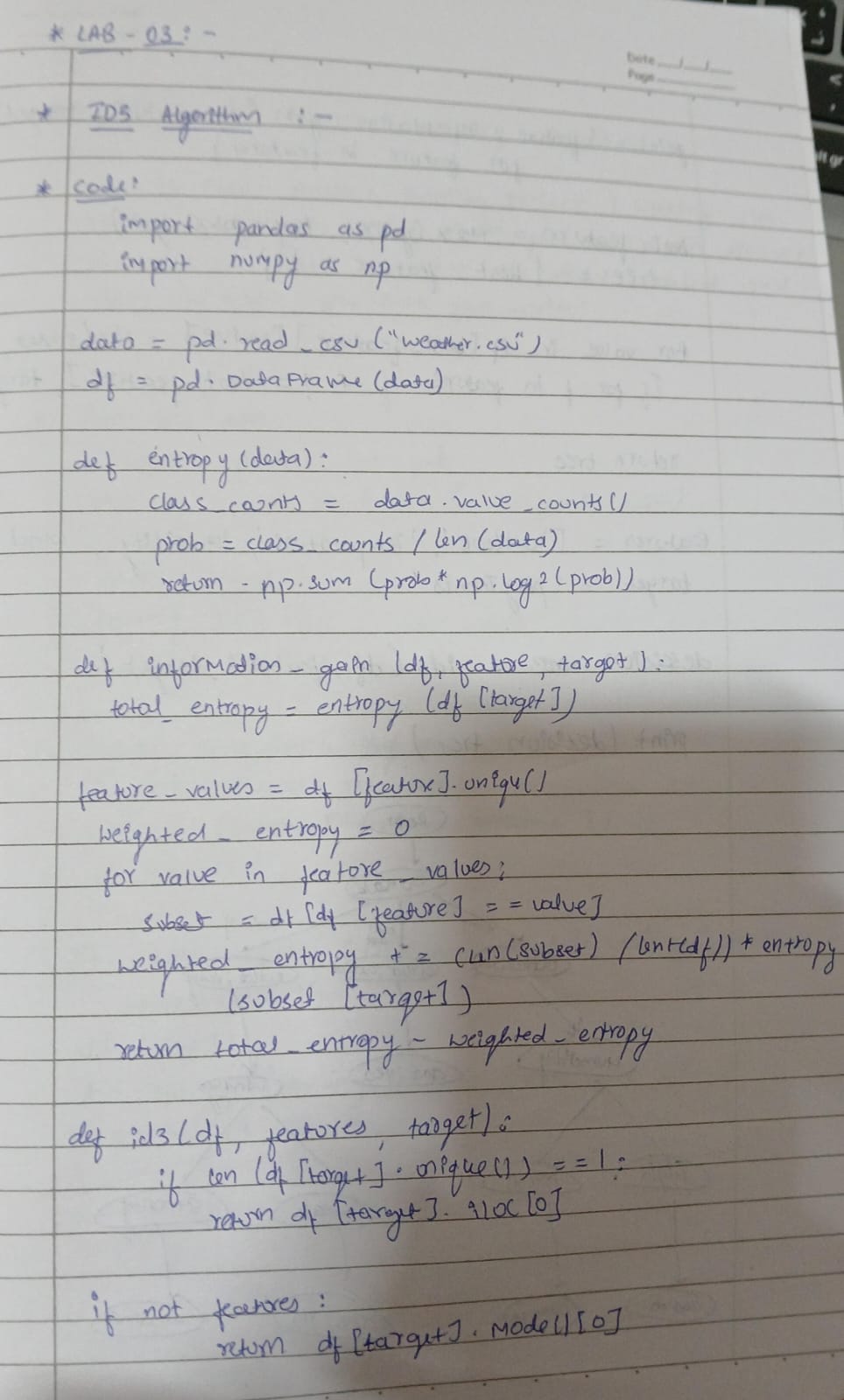
disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(y\_test)) disp.plot(cmap="Blues", values\_format="d")

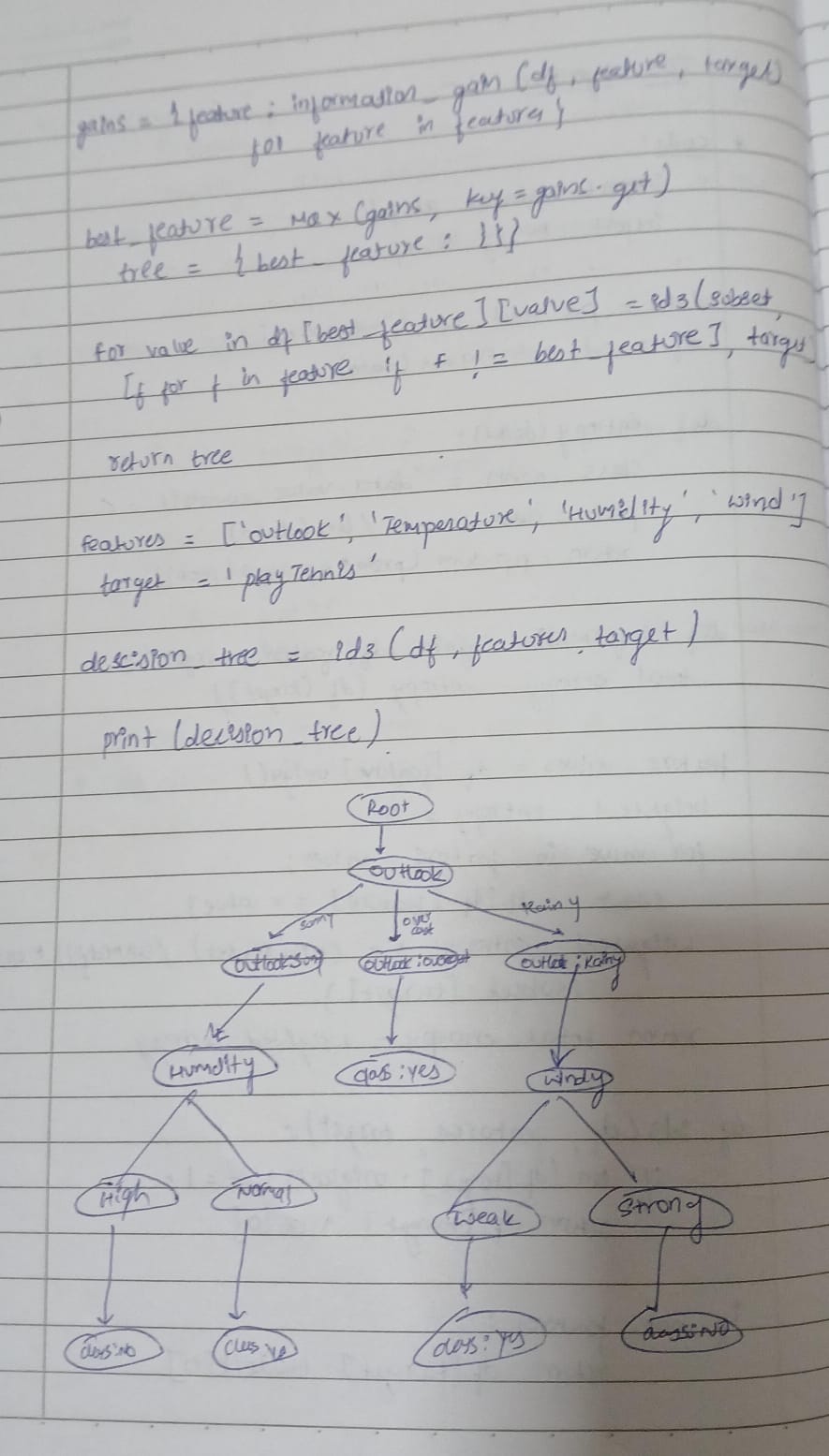
plt.title("Confusion Matrix") plt.show()

**Program 5**

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

**Screenshot:**





**Code:**

from google.colab import files iris=files.upload() df1=pd.read\_csv("iris.csv") df1.head()

df1.isnull().sum()

X = df1.drop('species', axis=1) y = df1['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) clf = DecisionTreeClassifier(criterion='entropy')

clf.fit(X\_train, y\_train) y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print(f'Accuracy: {accuracy:.2f}') print(classification\_report(y\_test, y\_pred)) plt.figure(figsize=(12, 8))

plot\_tree(clf, filled=True, feature\_names=X.columns, class\_names=y.unique()) plt.show()

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

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=clf.classes\_) cmap = plt.cm.get\_cmap('PuBuGn')

disp.plot(cmap=cmap) plt.show()

drug=files.upload() df2=pd.read\_csv("drug.csv") df2.head()

df2.isnull().sum()

label\_encoders = {}

for column in df2.columns: le = LabelEncoder()

df2[column] = le.fit\_transform(df2[column]) label\_encoders[column] = le

X = df2.drop('Drug', axis=1) y = df2['Drug']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) clf = DecisionTreeClassifier(criterion='entropy')

clf.fit(X\_train, y\_train) y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print(f'Accuracy: {accuracy:.2f}') print(classification\_report(y\_test, y\_pred)) plt.figure(figsize=(12, 8))

plot\_tree(clf, filled=True, feature\_names=X.columns, class\_names=[str(c) for c in y.unique()]) plt.show()

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

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

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

pc=files.upload() df3=pd.read\_csv("petrol\_consumption.csv") df3.head()

df3.isnull().sum()

X = df3.drop('Petrol\_Consumption', axis=1) y = df3['Petrol\_Consumption']

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

regressor.fit(X\_train, y\_train) y\_pred = regressor.predict(X\_test)

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

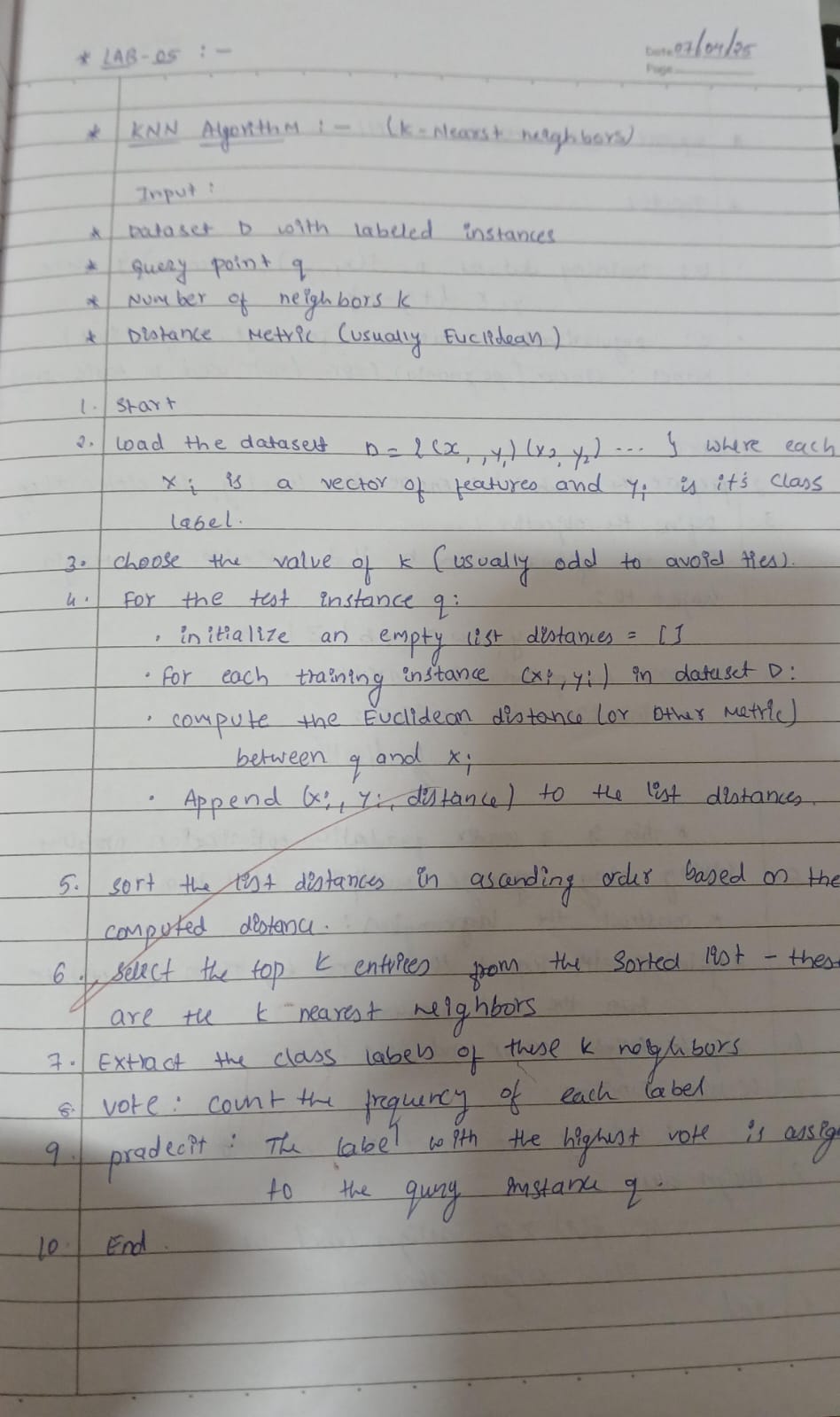
mae = mean\_absolute\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse:.2f}') print(f'Root Mean Squared Error: {rmse:.2f}') print(f'Mean Absolute Error: {mae:.2f}') print(f'R-squared: {r2:.2f}') plt.figure(figsize=(30, 30))

plot\_tree(regressor, filled=True, feature\_names=X.columns, fontsize=10) plt.show()

**Program 6 Build KNN Classification model for a given dataset.**

**Screenshot:**



**Code:**

from google.colab import files iris=files.upload() df1=pd.read\_csv("iris (2).csv") df1.head()

df1.isnull().sum()

X = df1.drop('species', axis=1) y = df1['species']

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, 11):

knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(X\_train, y\_train)

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

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

print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}: {1-accuracy}")

if accuracy > best\_accuracy: best\_accuracy = accuracy best\_k = k

print(f"Best k value: {best\_k}")

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)

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()

diabetes=files.upload() df2=pd.read\_csv("diabetes.csv") df2.head()

df2.isnull().sum()

from sklearn.preprocessing import StandardScaler scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(df2.drop('Outcome', axis=1))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, df2['Outcome'], test\_size=0.2, random\_state=42) best\_k = 1

best\_accuracy = 0 for k in range(1, 11):

knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(X\_train, y\_train)

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

accuracy = accuracy\_score(y\_test, y\_pred) print(f"Accuracy for k={k}: {accuracy}") if accuracy > best\_accuracy:

best\_accuracy = accuracy best\_k = k

print(f"Best k value: {best\_k}")

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("Accuracy:", accuracy)

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

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

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

plt.xlabel("Predicted") plt.ylabel("Actual") plt.title("Confusion Matrix") plt.show() print("\nClassification Report:")

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

heart=files.upload() df3=pd.read\_csv("heart.csv") df3.head()

df3.isnull().sum()

X = df3.drop('target', axis=1) y = df3['target']

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, 11):

knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(X\_train, y\_train)

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

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

print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}: {1-accuracy}") if accuracy > best\_accuracy:

best\_accuracy = accuracy best\_k = k

print(f"Best k value: {best\_k}")

knn = KNeighborsClassifier(n\_neighbors=optimal\_k) knn.fit(X\_train, y\_train)

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

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

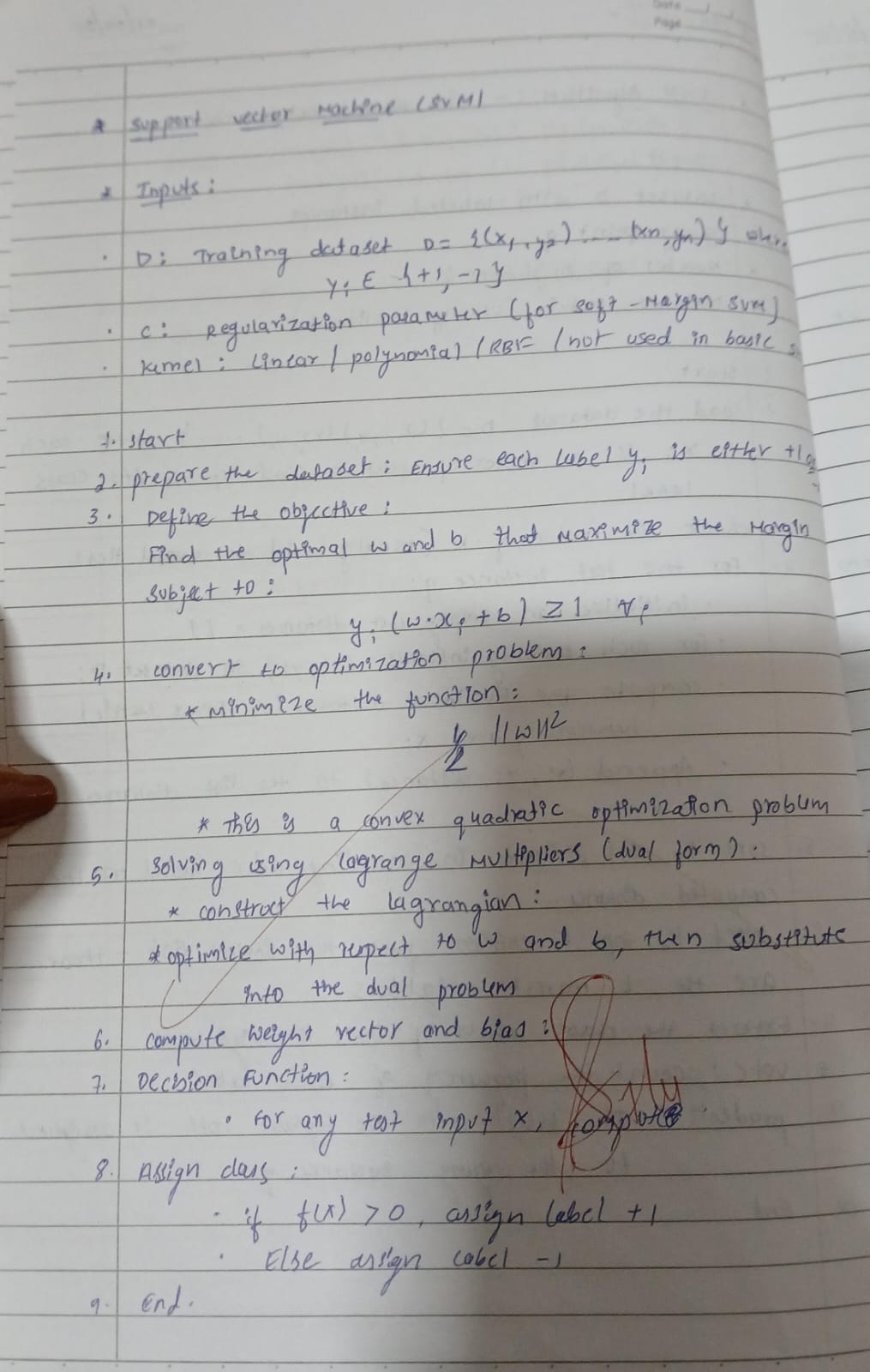
cm = confusion\_matrix(y\_test, y\_pred) print(cm)

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()

**Program 7 Build Support vector machine model for a given dataset Screenshot:**



**Code:**

from google.colab import files iris=files.upload() df1=pd.read\_csv("iris (1).csv") df1.head()

X = df1.drop('species', axis=1) y = df1['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) rbf\_svm = SVC(kernel='rbf')

rbf\_svm.fit(X\_train, y\_train) rbf\_y\_pred = rbf\_svm.predict(X\_test) print("RBF Kernel SVM:")

print("Accuracy:", accuracy\_score(y\_test, rbf\_y\_pred)) cm = confusion\_matrix(y\_test, rbf\_y\_pred) sns.heatmap(cm, annot=True, fmt='d',cmap="Blues") plt.title('Confusion Matrix for RBF Kernel SVM') plt.xlabel('Predicted')

plt.ylabel('True') plt.show()

print(classification\_report(y\_test, rbf\_y\_pred)) linear\_svm = SVC(kernel='linear') linear\_svm.fit(X\_train, y\_train)

linear\_y\_pred = linear\_svm.predict(X\_test) print("\nLinear Kernel SVM:")

print("Accuracy:", accuracy\_score(y\_test, linear\_y\_pred)) cm = confusion\_matrix(y\_test, linear\_y\_pred) sns.heatmap(cm, annot=True, fmt='d',cmap="Blues") plt.title('Confusion Matrix for Linear Kernel SVM') plt.xlabel('Predicted')

plt.ylabel('True') plt.show()

print(classification\_report(y\_test, linear\_y\_pred)) letter=files.upload()

df2=pd.read\_csv("letter-recognition.csv") df2.head()

X = df2.drop('letter', axis=1) y = df2['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='linear', probability=True)

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

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

cm = confusion\_matrix(y\_test, y\_pred) plt.figure(figsize=(10,10))

sns.heatmap(cm, annot=True, fmt='d', cmap="Blues") plt.title('Confusion Matrix for SVM') plt.xlabel('Predicted')

plt.ylabel('True') plt.show()

lb = LabelBinarizer() lb.fit(y\_test)

y\_test\_lb = lb.transform(y\_test)

y\_pred\_prob = svm\_classifier.predict\_proba(X\_test) fpr = {}

tpr = {} thresh ={}

roc\_auc = dict()

n\_class = y\_test\_lb.shape[1] for i in range(n\_class):

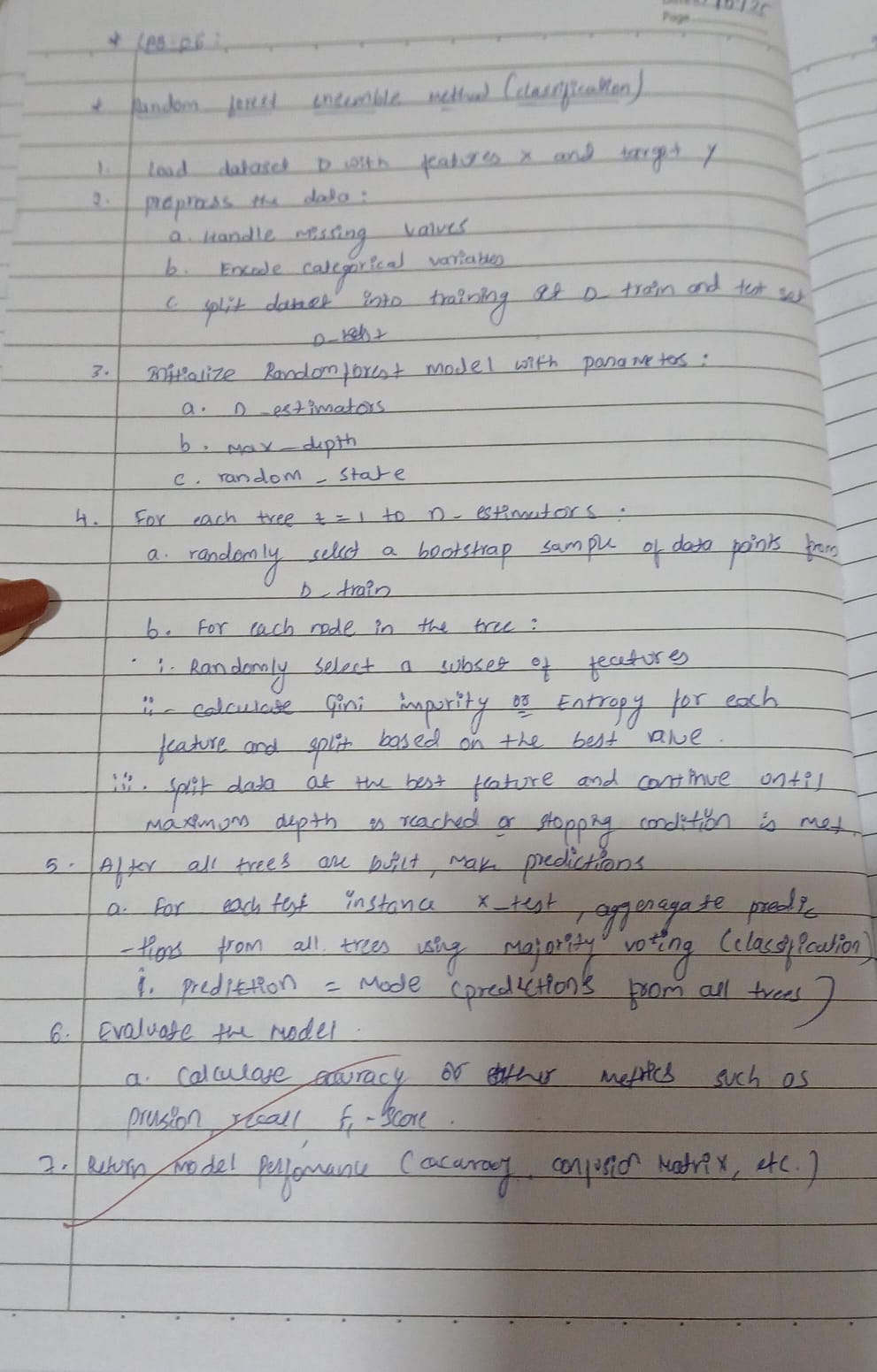
fpr[i], tpr[i], thresh[i] = roc\_curve(y\_test\_lb[:,i], y\_pred\_prob[:,i]) roc\_auc[i] = auc(fpr[i], tpr[i])

plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label='SVM (AUC = %0.2f)' % roc\_auc[0]) plt.title('ROC Curve for Class 0')

plt.xlabel('False Positive Rate') plt.ylabel('True Positive rate') plt.legend(loc='best') plt.show()

print(f"AUC score for class 0: {roc\_auc[0]}")

**Program 8 Implement Random forest ensemble method on a given dataset**



**Code:**

from google.colab import files iris=files.upload() df1=pd.read\_csv("iris (4).csv") df1.head()

X = df1.drop('species', axis=1) y = df1['species']

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

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

default\_accuracy = accuracy\_score(y\_test, y\_pred) print(f"Accuracy with default n\_estimators: {default\_accuracy}") best\_accuracy = 0

best\_n\_estimators = 0

for n\_estimators in range(1, 101):

rf\_classifier = RandomForestClassifier(n\_estimators=n\_estimators, random\_state=0) rf\_classifier.fit(X\_train, y\_train)

y\_pred = rf\_classifier.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred) if accuracy > best\_accuracy:

best\_accuracy = accuracy best\_n\_estimators = n\_estimators

print(f"\nBest accuracy: {best\_accuracy} achieved with n\_estimators = {best\_n\_estimators}") cm = confusion\_matrix(y\_test, y\_pred)

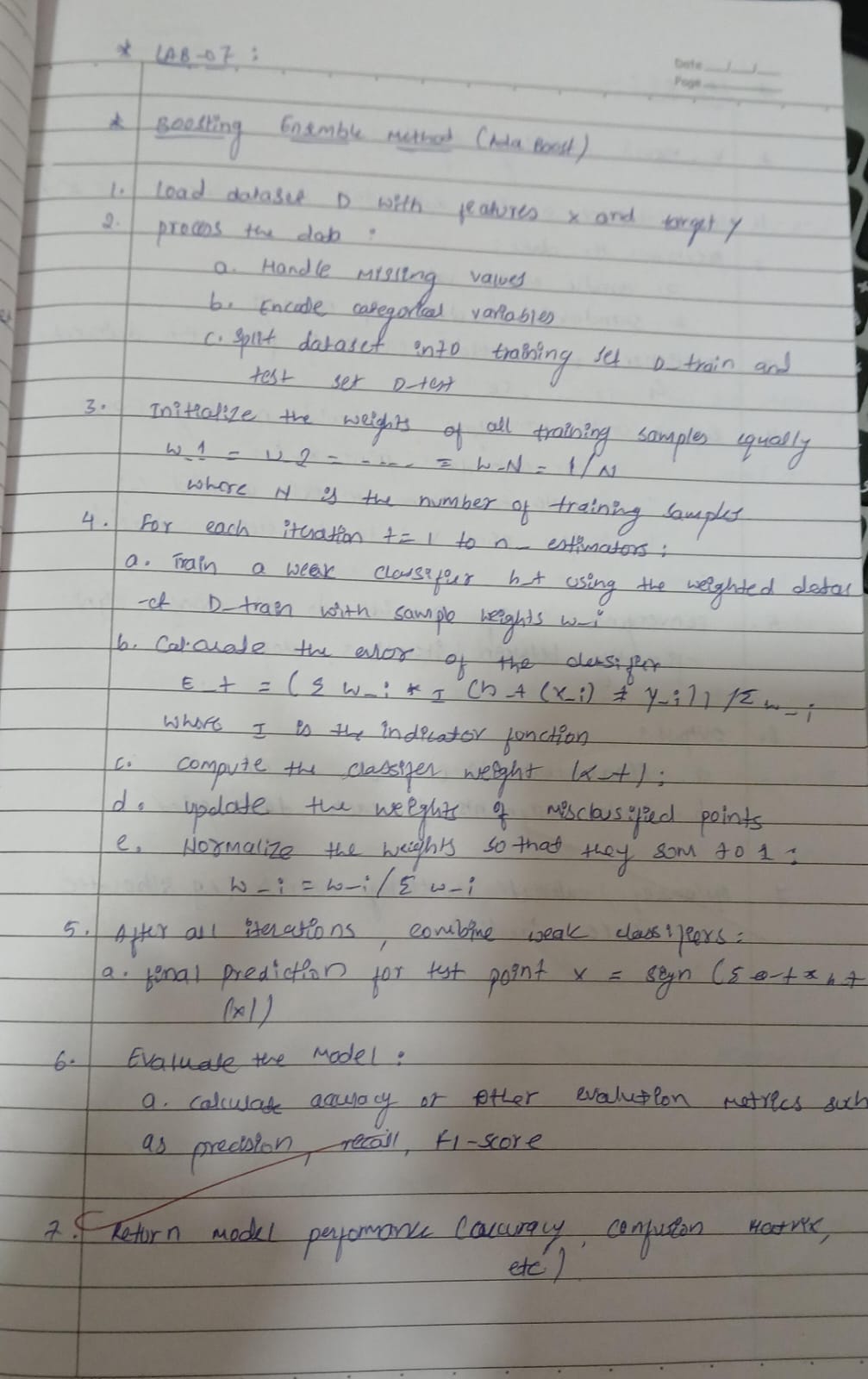
plt.figure(figsize=(8, 6))

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

plt.xlabel("Predicted") plt.ylabel("Actual") plt.title("Confusion Matrix") plt.show()

**Program 9 Implement Boosting ensemble method on a given dataset**

**Screenshot:**



**Code:**

from google.colab import files income=files.upload() df1=pd.read\_csv("income.csv") df1.head()

X = df1.drop('income\_level', axis=1)y = df1['income\_level'] X = pd.get\_dummies(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42) abc = AdaBoostClassifier(n\_estimators=10, random\_state=42)

abc.fit(X\_train, y\_train) y\_pred = abc.predict(X\_test)

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

print(f"Initial AdaBoost accuracy (10 trees): {accuracy}") param\_grid = {'n\_estimators': [50, 100, 150, 200]}

grid\_search = GridSearchCV(AdaBoostClassifier(random\_state=42), param\_grid, cv=5, scoring='accuracy') grid\_search.fit(X\_train, y\_train)

print(f"Best parameters: {grid\_search.best\_params\_}") print(f"Best cross-validation score: {grid\_search.best\_score\_}") best\_abc = grid\_search.best\_estimator\_

y\_pred\_best = best\_abc.predict(X\_test) best\_accuracy = accuracy\_score(y\_test, y\_pred\_best)

print(f"Accuracy of the best model on the test set: {best\_accuracy}") cm = confusion\_matrix(y\_test, y\_pred\_best)

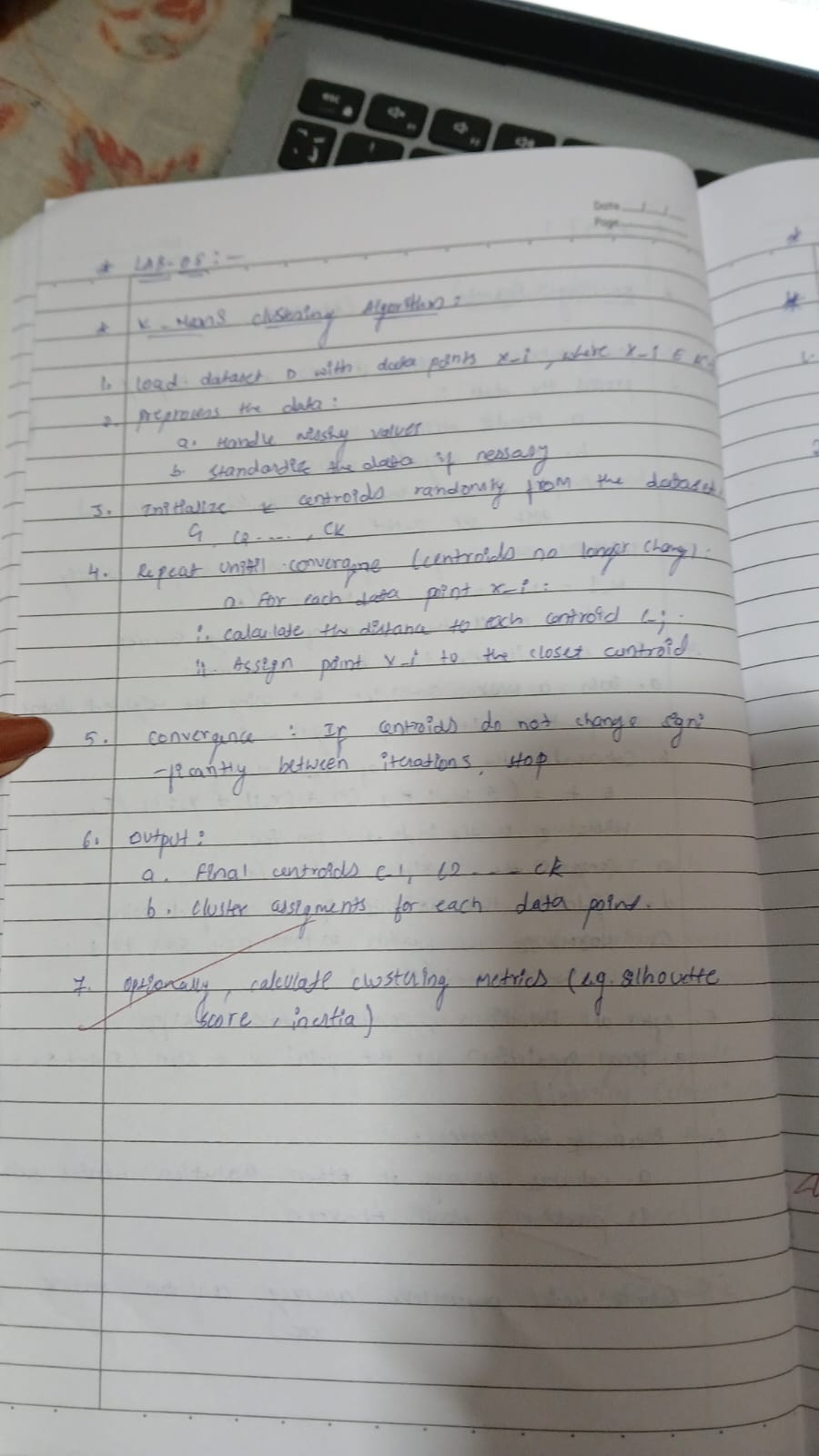
plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])

plt.xlabel('Predicted') plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show()

**Program 10**

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



**Code:**

from google.colab import files iris=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.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 (4).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)

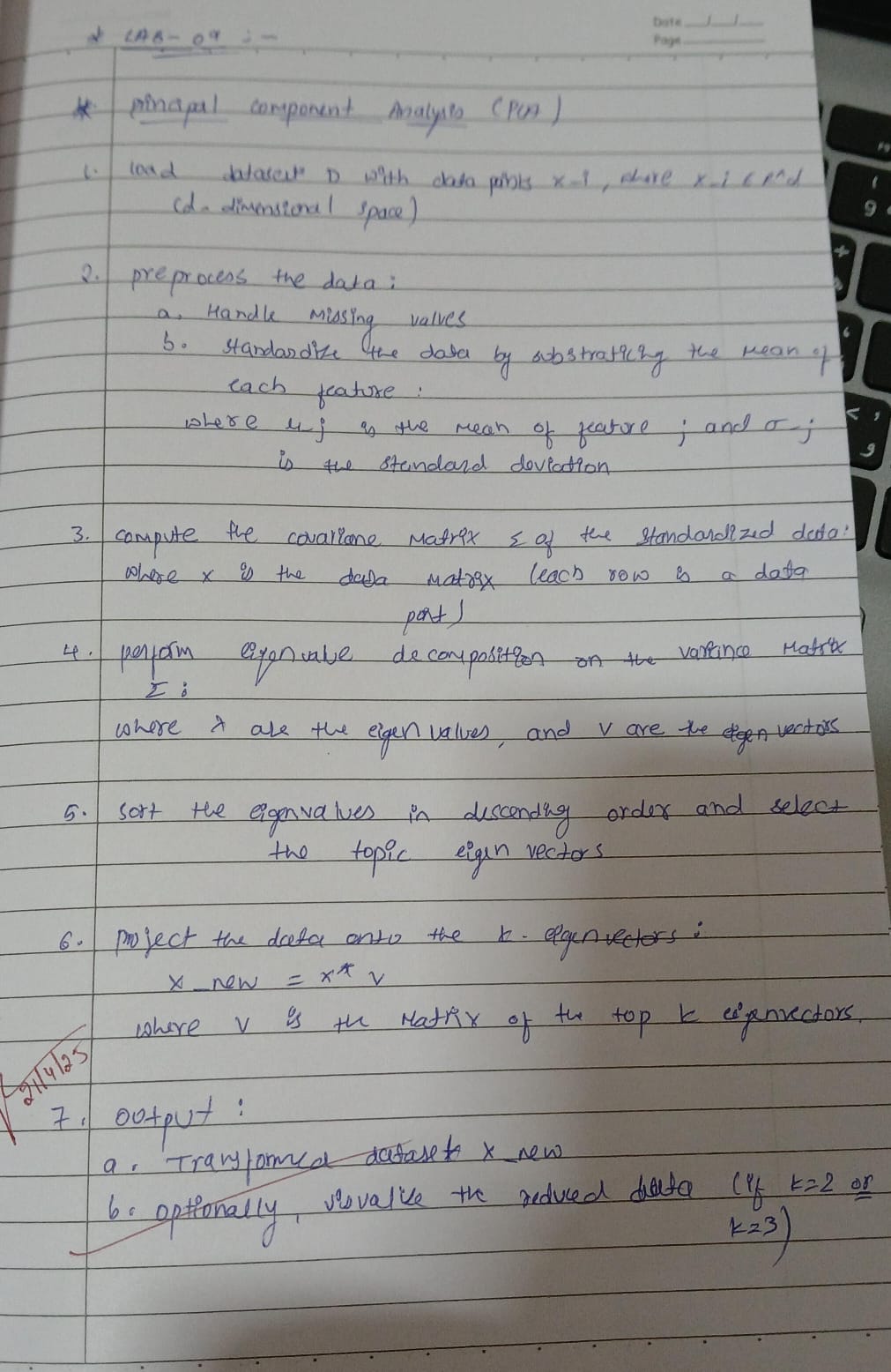
df['cluster'] = pred\_y

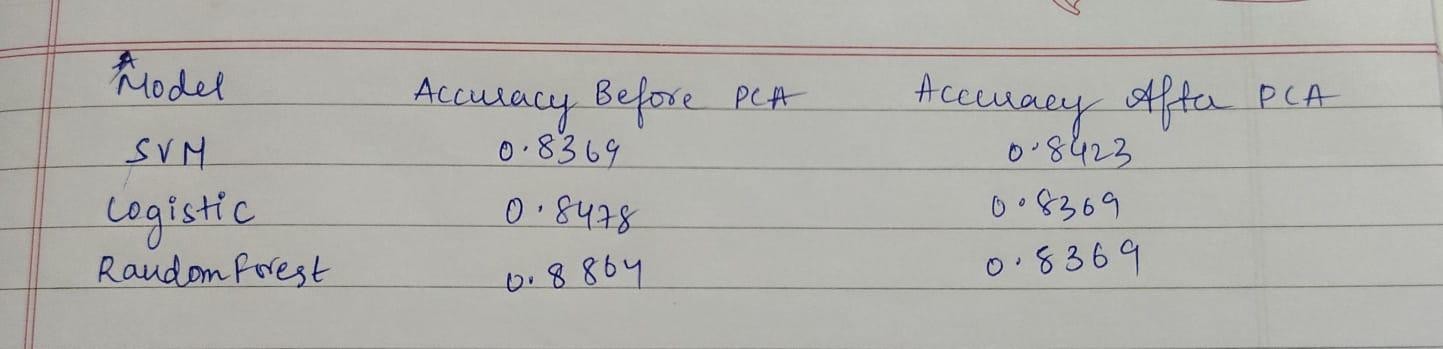
plt.scatter(df['petal\_length'], df['petal\_width'], c=df['cluster']) plt.title('Clusters of Iris Flowers')

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 (1).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]}")