# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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

****

## LAB REPORT

### on

Machine Learning (23CS6PCMAL)

#### Submitted by

**Shruti Khandelia (1BM22CS274)**

#### 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 **Shruti Khandelia (1BM22CS274),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

**Dr. Kavitha Sooda** Professor & HOD

Department of CSE, BMSCE

**Lab Faculty Incharge**

Name: **Sunayana S**

Assistant Professor Department of CSE, BMSCE

# Index

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 5-3-2025 | Write a python program to import and export data using Pandas library functions | 4 |
| 2 | 5-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 7 |
| 3 | 19-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 10 |
| 4 | 2-4-2025 | Build Logistic Regression Model for a given dataset | 13 |
| 5 | 12-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample | 17 |
| 6 | 2-4-2025 | Build KNN Classification model for a given dataset | 20 |
| 7 | 9-4-2025 | Build Support vector machine model for a given dataset | 25 |
| 8 | 7-5-2025 | Implement Random Forest ensemble method on a given dataset | 27 |
| 9 | 7-5-2025 | Implement Boosting ensemble method on a given dataset | 29 |
| 10 | 7-5-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file | 32 |
| 11 | 7-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method | 34 |

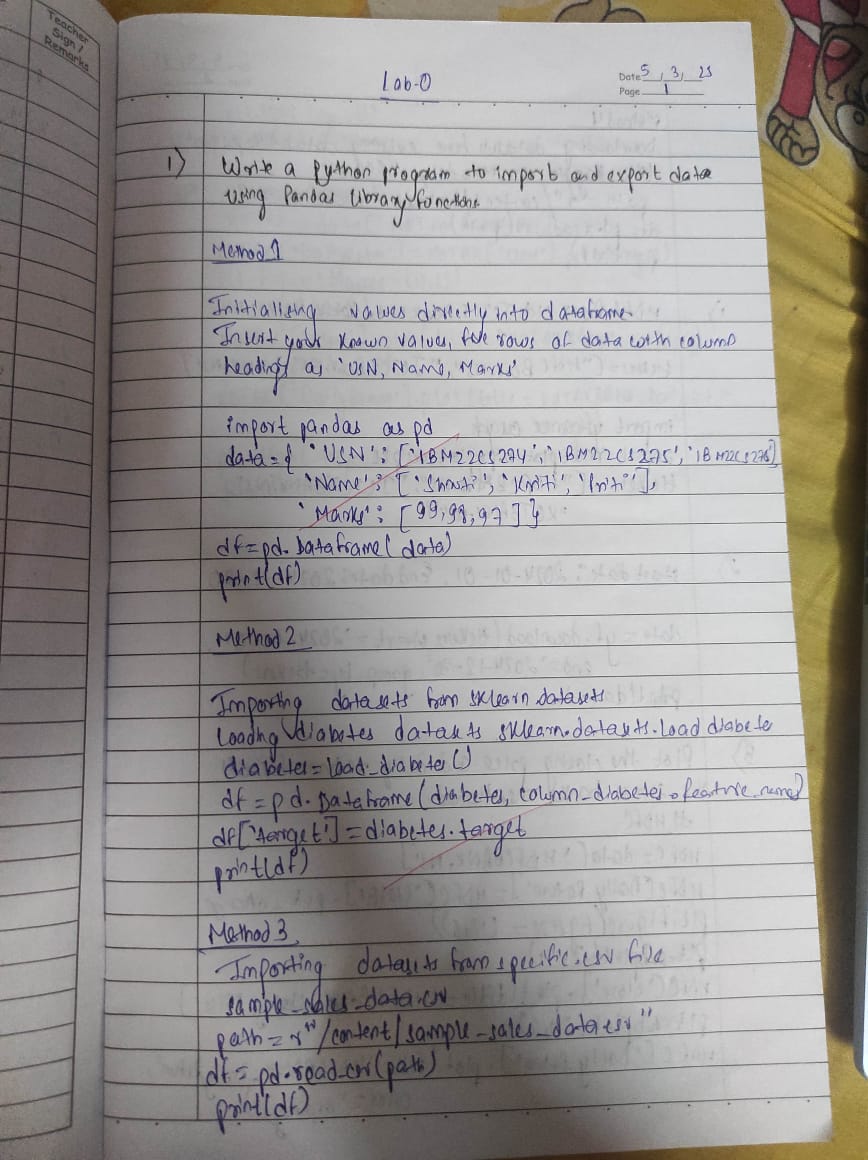
**Github Link:**

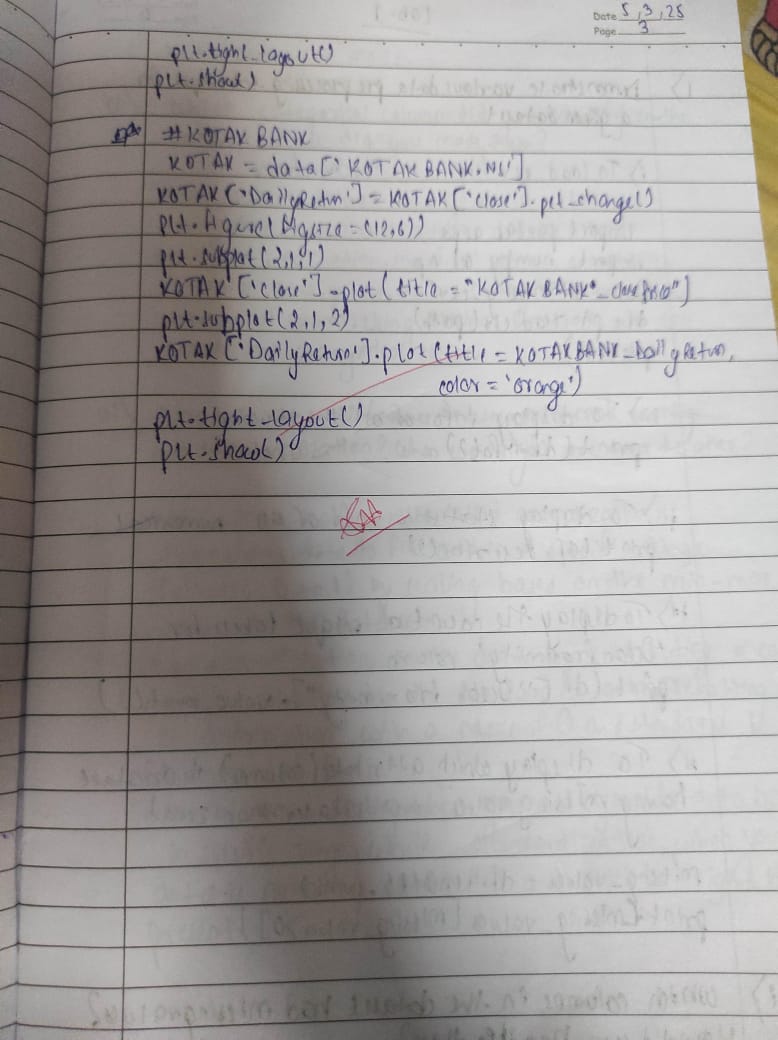
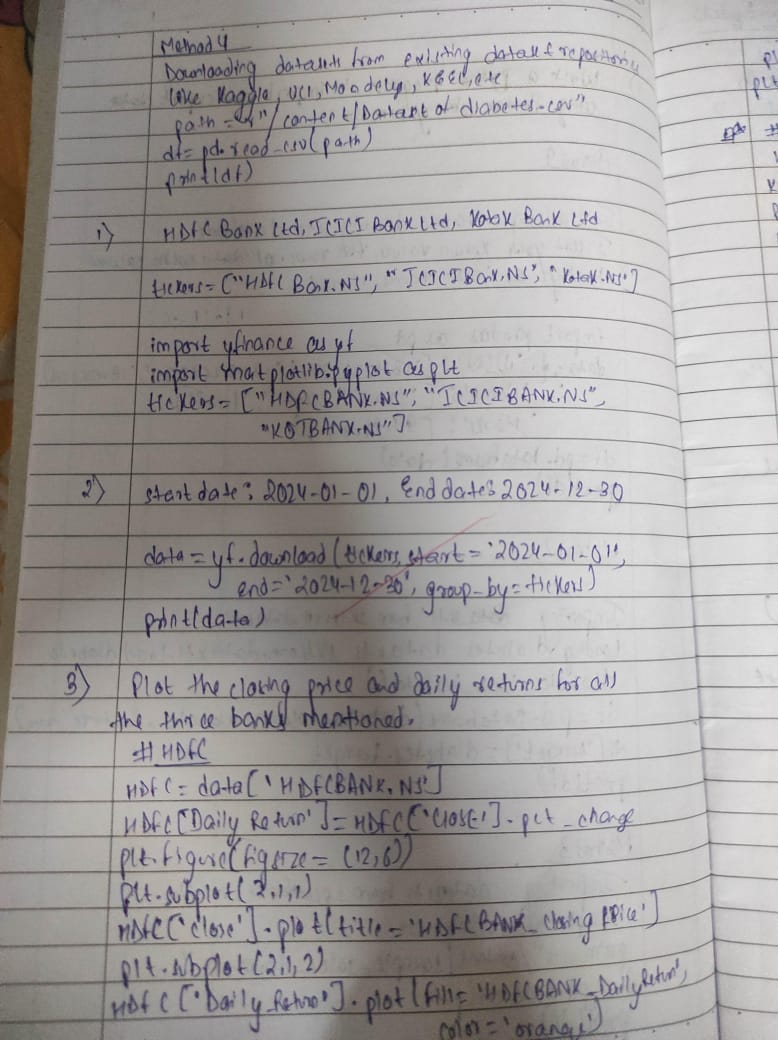
**https://github.com/shrutikhandelia/ML**

##### Program 1

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

Screenshot





Code:

import pandas as pd

data={

'USN':['1BM22CS001','1BM22CS002','1BM22CS003','1BM22CS004','1BM22CS005'],

'Name':['Ankita','Anita','Amit','Anish','Arun'],

'Marks':[99,56,96,85,45]

}

df=pd.DataFrame(data)

print(df)

from sklearn.datasets import load\_diabetes

data=load\_diabetes()

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

df['target']=data.target

print(df)

path=r"/content/sample\_sales\_data.csv"

df=pd.read\_csv(path)

print(df)

path=r"/content/Dataset of Diabetes .csv"

df=pd.read\_csv(path)

print(df.head())

import yfinance as yf

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=tickers)

print(data)

*#HDFCBANK*

HDFC=data['HDFCBANK.NS']

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

print(HDFC)

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

plt.subplot(2,1,1)

HDFC['Close'].plot(title='HDFC BANK - Closing Price')

plt.subplot(2,1,2)

HDFC['Daily Return'].plot(title='HDFC BANK - Daily Return',color='orange')

plt.tight\_layout()

plt.show()

*#ICICIBANK*

ICICI=data['ICICIBANK.NS']

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

print(ICICI)

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

plt.subplot(2,1,1)

ICICI['Close'].plot(title='ICICI BANK - Closing Price')

plt.subplot(2,1,2)

ICICI['Daily Return'].plot(title='ICIC BANK - Daily Return',color='orange')

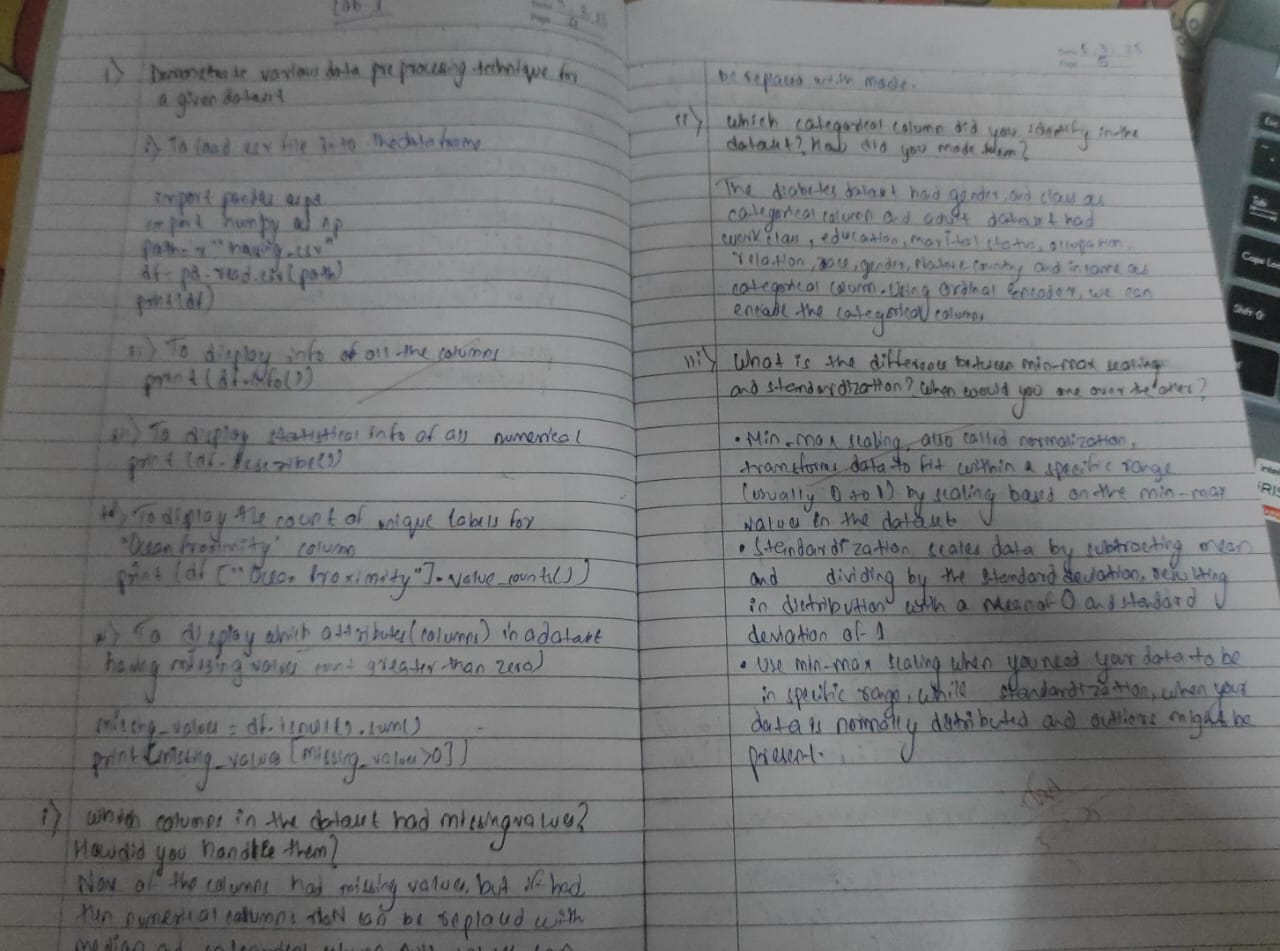
plt.tight\_layout()

plt.show()

##### Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



Code:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from scipy import stats

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

df.head(10)

df.shape

print(df.info())

print(df.describe())

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

*# Display columns with missing values*

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

*#Set the values to some value (zero, the mean, the median, etc.).*

*# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary*

imputer1 = SimpleImputer(strategy="median")

imputer2 = SimpleImputer(strategy="mean")

df\_copy=df

*# Step 2: Fit the imputer on the "Age" and "Salary"column*

*# Note: SimpleImputer expects a 2D array, so we reshape the column*

imputer1.fit(df\_copy[["AGE"]])

imputer2.fit(df\_copy[["BMI"]])

*# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column*

df\_copy["AGE"] = imputer1.transform(df[["AGE"]])

df\_copy["BMI"] = imputer2.transform(df[["BMI"]])

*# Verify that there are no missing values left*

print(df\_copy["AGE"].isnull().sum())

print(df\_copy["BMI"].isnull().sum())

*#Handling Categorical Attributes*

*#Using Ordinal Encoding for gender COlumn and One-Hot Encoding for City Column*

*# Initialize OrdinalEncoder*

ordinal\_encoder = OrdinalEncoder(categories=[["M", "F","f"]])

*# Fit and transform the data*

df\_copy["Gender\_Encoded"] = ordinal\_encoder.fit\_transform(df\_copy[["Gender"]])

*# Initialize OneHotEncoder*

onehot\_encoder = OneHotEncoder()

*# Fit and transform the "City" column*

encoded\_data = onehot\_encoder.fit\_transform(df[["CLASS"]])

*# Convert the sparse matrix to a dense array*

encoded\_array = encoded\_data.toarray()

*# Convert to DataFrame for better visualization*

encoded\_df = pd.DataFrame(encoded\_array, columns=onehot\_encoder.get\_feature\_names\_out(["CLASS"]))

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

df\_encoded.drop("Gender", axis=1, inplace=True)

df\_encoded.drop("CLASS", axis=1, inplace=True)

print(df\_encoded. head())

normalizer = MinMaxScaler()

df\_encoded[['BMI']] = normalizer.fit\_transform(df\_encoded[['BMI']])

df\_encoded.head()

scaler = StandardScaler()

df\_encoded[['AGE']] = scaler.fit\_transform(df\_encoded[['AGE']])

df\_encoded.head()

df\_encoded\_copy1=df\_encoded

df\_encoded\_copy2=df\_encoded

df\_encoded\_copy3=df\_encoded

Q1 = df\_encoded\_copy1['BMI'].quantile(0.25)

Q3 = df\_encoded\_copy1['BMI'].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

df\_encoded\_copy1['BMI'] = np.where(df\_encoded\_copy1['BMI'] > upper\_bound, upper\_bound,

np.where(df\_encoded\_copy1['BMI'] < lower\_bound, lower\_bound, df\_encoded\_copy1['BMI']))

print(df\_encoded\_copy1.head())

df\_encoded\_copy2['BMI\_zscore'] = stats.zscore(df\_encoded\_copy2['BMI'])

df\_encoded\_copy2['BMI'] = np.where(df\_encoded\_copy2['BMI\_zscore'].abs() > 3, np.nan, df\_encoded\_copy2['BMI']) *# Replace outliers with NaN*

print(df\_encoded\_copy2.head())

df\_encoded\_copy3['BMI\_zscore'] = stats.zscore(df\_encoded\_copy3['BMI'])

median\_salary = df\_encoded\_copy3['BMI'].median()

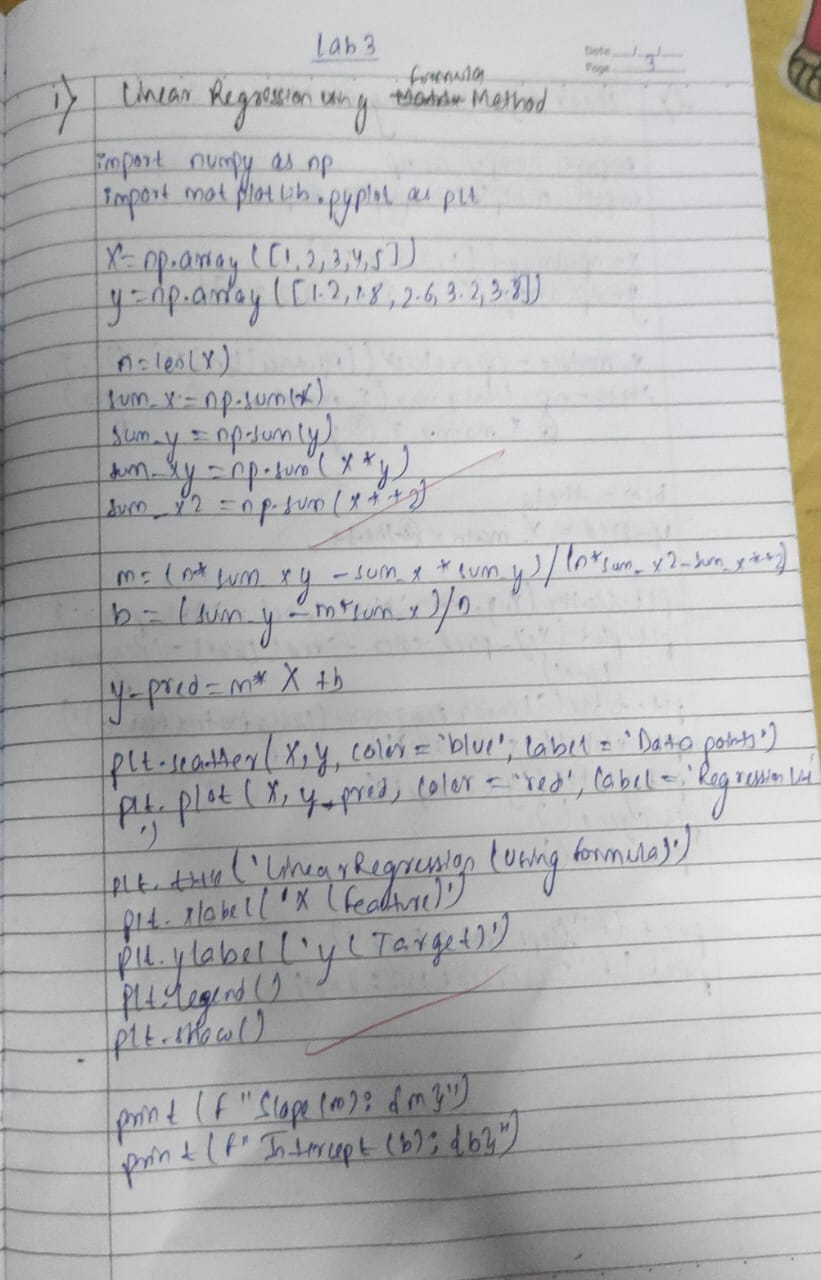
df\_encoded\_copy3['BMI'] = np.where(df\_encoded\_copy3['BMI\_zscore'].abs() > 3, median\_salary, df\_encoded\_copy3['BMI'])

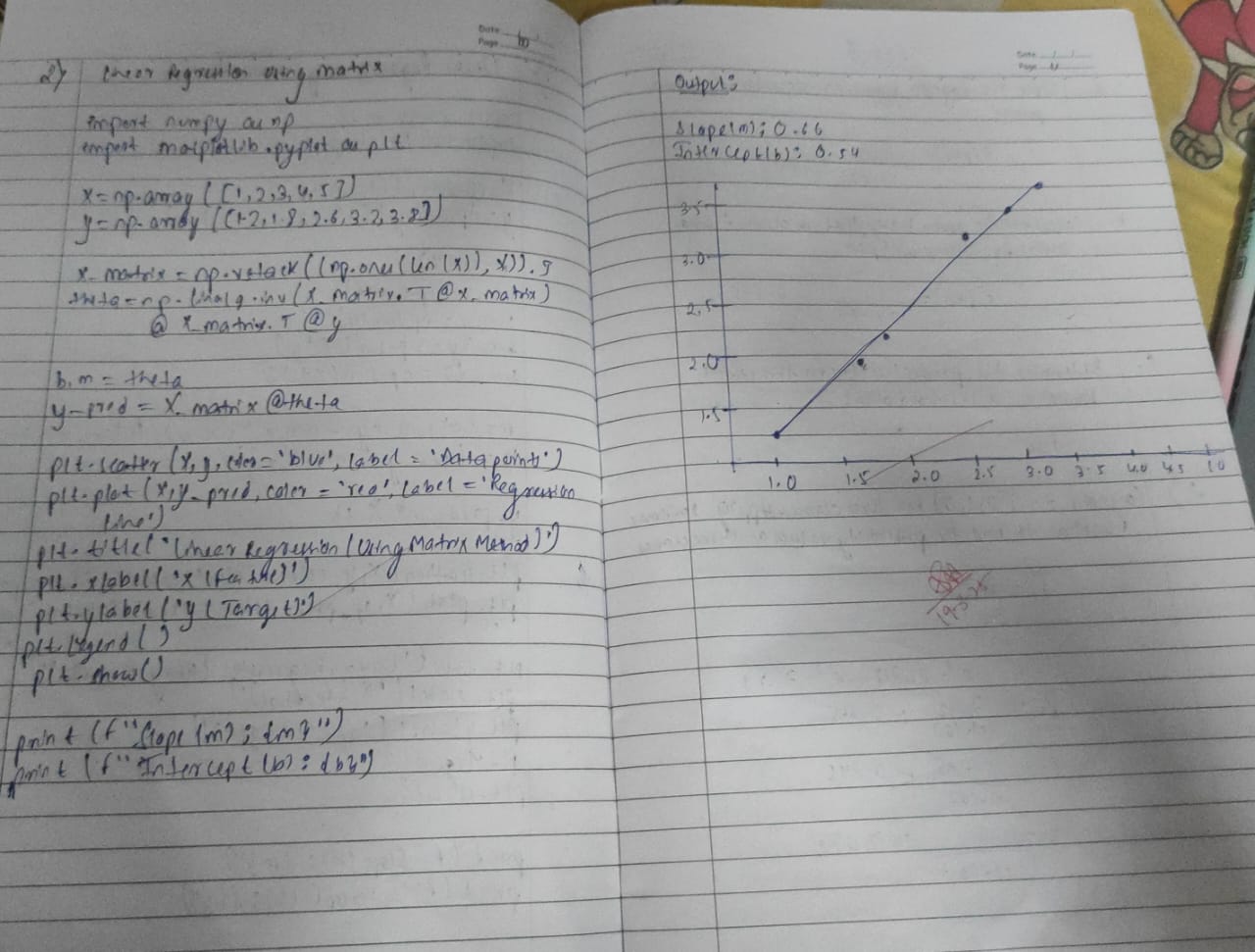
print(df\_encoded\_copy3.head())

##### Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot





Code:

import pandas as pd

import matplotlib.pyplot as plt

data={"X":[1,2,3,4,5],

"Y":[1.2,1.8,2.6,3.2,3.8]}

df=pd.DataFrame(data)

df

Xi=df["X"].mean()

Yi=df["Y"].mean()

df["Xi^2"]=[Xi\*\*2 for Xi in df["X"]]

Xisq=df["Xi^2"].mean()

xiyi=[]

x=df["X"]

y=df["Y"]

for i in range(len(x)):

xiyi.append(x[i]\*y[i])

df["XiYi"]=xiyi

print(df["XiYi"])

XiYi2=df["XiYi"].mean()

print(XiYi2)

a1 = (df["XiYi"].sum() - len(df) \* Xi \* Yi) / (df["X"].apply(lambda x: x\*\*2).sum() - len(df) \* Xi\*\*2)

a0 = Yi - a1 \* Xi

x=9

Y=a0+a1\*x

print(Y)

plt.scatter(df["X"], df["Y"], color='blue', label='Data points') *# Scatter plot of original data*

plt.plot(df["X"], a0 + a1 \* df["X"], color='red', label='Regression Line') *# Correct regression line*

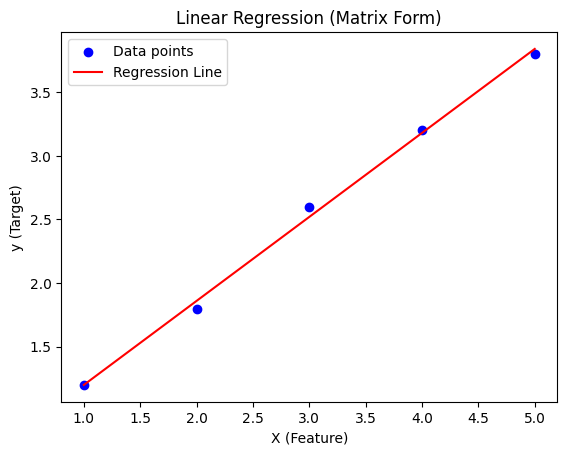
plt.title('Linear Regression (Matrix Form)')

plt.xlabel('X (Feature)')

plt.ylabel('y (Target)')

plt.legend()

plt.show()



import numpy as np

import matplotlib.pyplot as plt

X = np.array([1, 2, 3, 4])

y = np.array([1,3,4,8])

X\_matrix = np.c\_[np.ones(len(X)), X]

theta = np.linalg.inv(X\_matrix.T @ X\_matrix) @ X\_matrix.T @ y

b, m = theta

y\_pred = m \* X + b

print(f"Slope (m): {m}")

print(f"Intercept (b): {b}")

Slope (m): 2.2000000000000006

Intercept (b): -1.5

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

plt.plot(X, y\_pred, color='red', label='Regression Line')

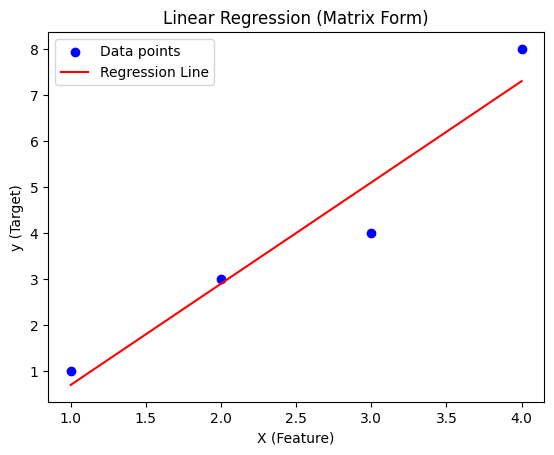
plt.title('Linear Regression (Matrix Form)')

plt.xlabel('X (Feature)')

plt.ylabel('y (Target)')

plt.legend()

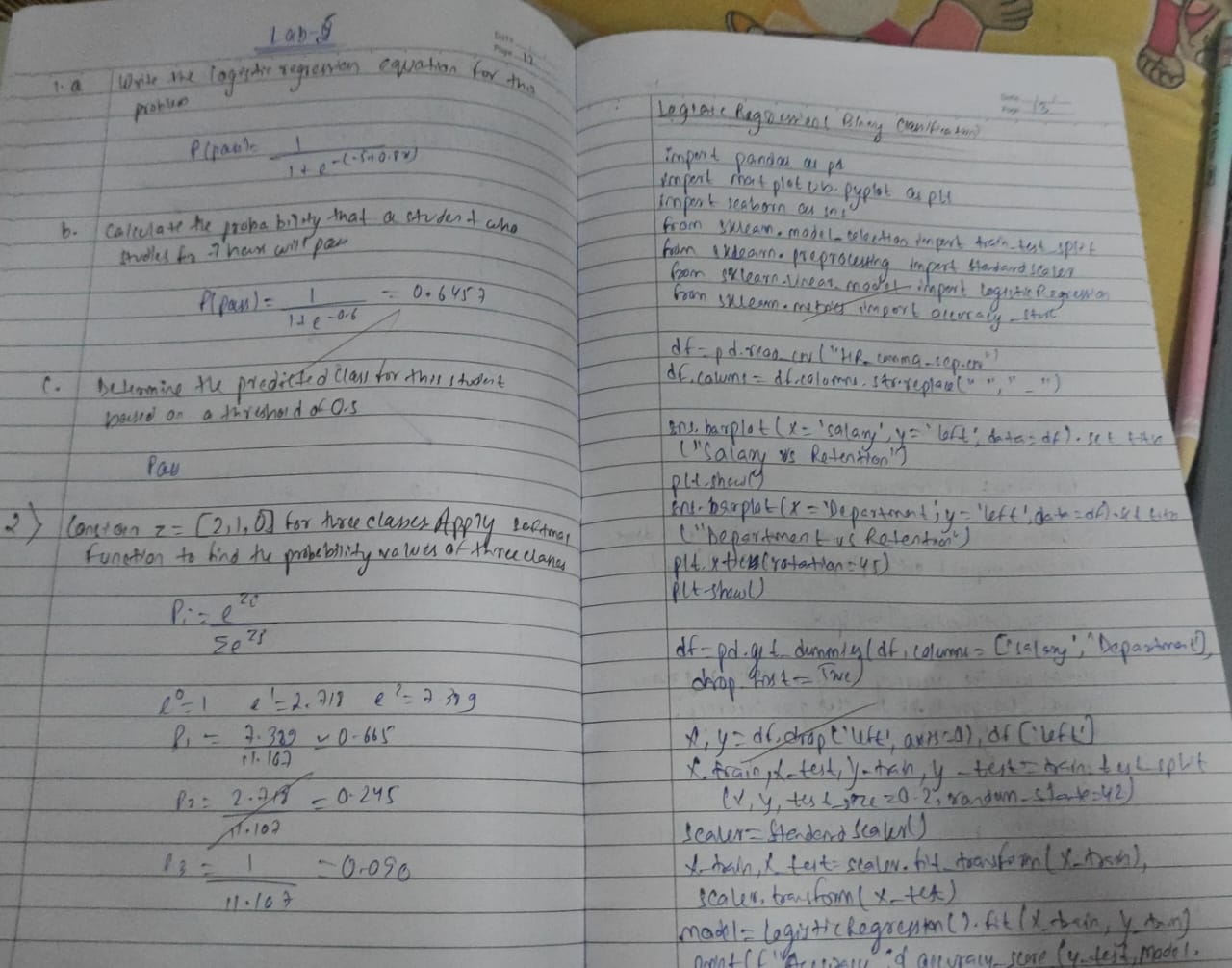
plt.show()

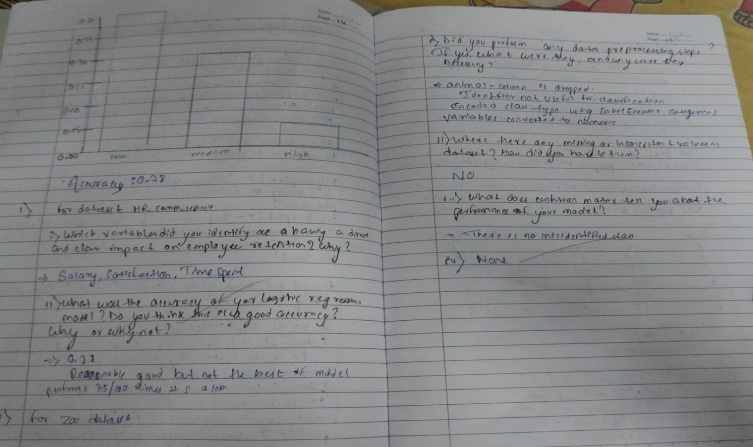


##### Program 4

Build Logistic Regression Model for a given dataset

Screenshot





Code:

import pandas as pd

import matplotlib.pyplot as plt

import math

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

from sklearn.linear\_model import LogisticRegression

*# Load dataset*

df = pd.read\_csv("/content/HR\_comma\_sep (2).csv")

*# Scatter plot: Employee satisfaction vs Retention*

plt.scatter(df.satisfaction\_level, df.left, marker='+', color='red')

plt.xlabel("Satisfaction Level")

plt.ylabel("Left (1) / Stayed (0)")

plt.title("Impact of Satisfaction Level on Employee Retention")

plt.show()

*# Define features (X) and target (y)*

X = df[['satisfaction\_level']]

y = df['left']

*# Split dataset (90% train, 10% test)*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.9, random\_state=10)

*# Train logistic regression model*

model = LogisticRegression()

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

*# Predictions*

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

*# Model Accuracy*

print(f"Model Accuracy: {model.score(X\_test, y\_test):.4f}")

*# Probability predictions*

print("Predicted Probabilities:")

print(model.predict\_proba(X\_test))

*# Predict for a specific satisfaction level (e.g., 0.4)*

predicted\_status = model.predict([[0.4]])

print(f"Prediction for Satisfaction Level 0.4: {'Left' if predicted\_status[0] == 1 else 'Stayed'}")

*# Logistic function*

def sigmoid(x):

return 1 / (1 + math.exp(-x))

*# Custom prediction function*

m, b = model.coef\_[0][0], model.intercept\_[0]

def prediction\_function(satisfaction):

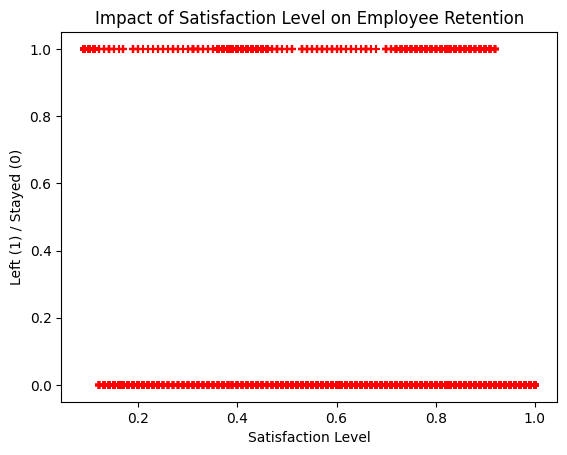
z = m \* satisfaction + b

y = sigmoid(z)

return y

satisfaction\_test = 0.4

print(f"Sigmoid Prediction for Satisfaction Level {satisfaction\_test}: {prediction\_function(satisfaction\_test):.4f}")



Model Accuracy: 0.7707

Predicted Probabilities:

[[0.81879598 0.18120402]

[0.64435551 0.35564449]

[0.67008191 0.32991809]

...

[0.85026544 0.14973456]

[0.93858587 0.06141413]

[0.90306111 0.09693889]]

Prediction for Satisfaction Level 0.4: Stayed

Sigmoid Prediction for Satisfaction Level 0.4: 0.3644

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

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

import matplotlib.pyplot as plt

*# Load the Zoo dataset*

file\_path = "/content/zoo-data (1).csv"

zoo\_data = pd.read\_csv(file\_path)

*# Drop the 'animal\_name' column as it is not a relevant feature*

X = zoo\_data.drop(['animal\_name', 'class\_type'], axis=1) *# Features*

y = zoo\_data['class\_type'] *# Target variable*

*# Split the dataset into 80% training and 20% testing*

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

*# Initialize the Logistic Regression model for multi-class classification*

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

*# Train the model*

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

*# Make predictions*

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

*# Calculate accuracy*

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

print(f"Accuracy of the Multinomial Logistic Regression model: {accuracy:.2f}")

*# Compute confusion matrix*

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

*# Adjust display labels to match actual present labels in the test set*

unique\_classes\_in\_test = sorted(y\_test.unique())

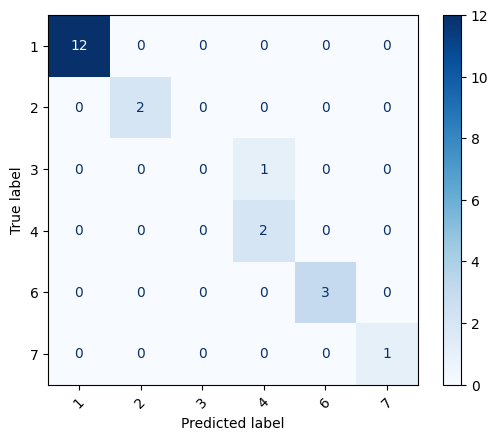
*# Display confusion matrix*

cm\_display = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=unique\_classes\_in\_test)

cm\_display.plot(cmap='Blues', xticks\_rotation=45)

plt.show()

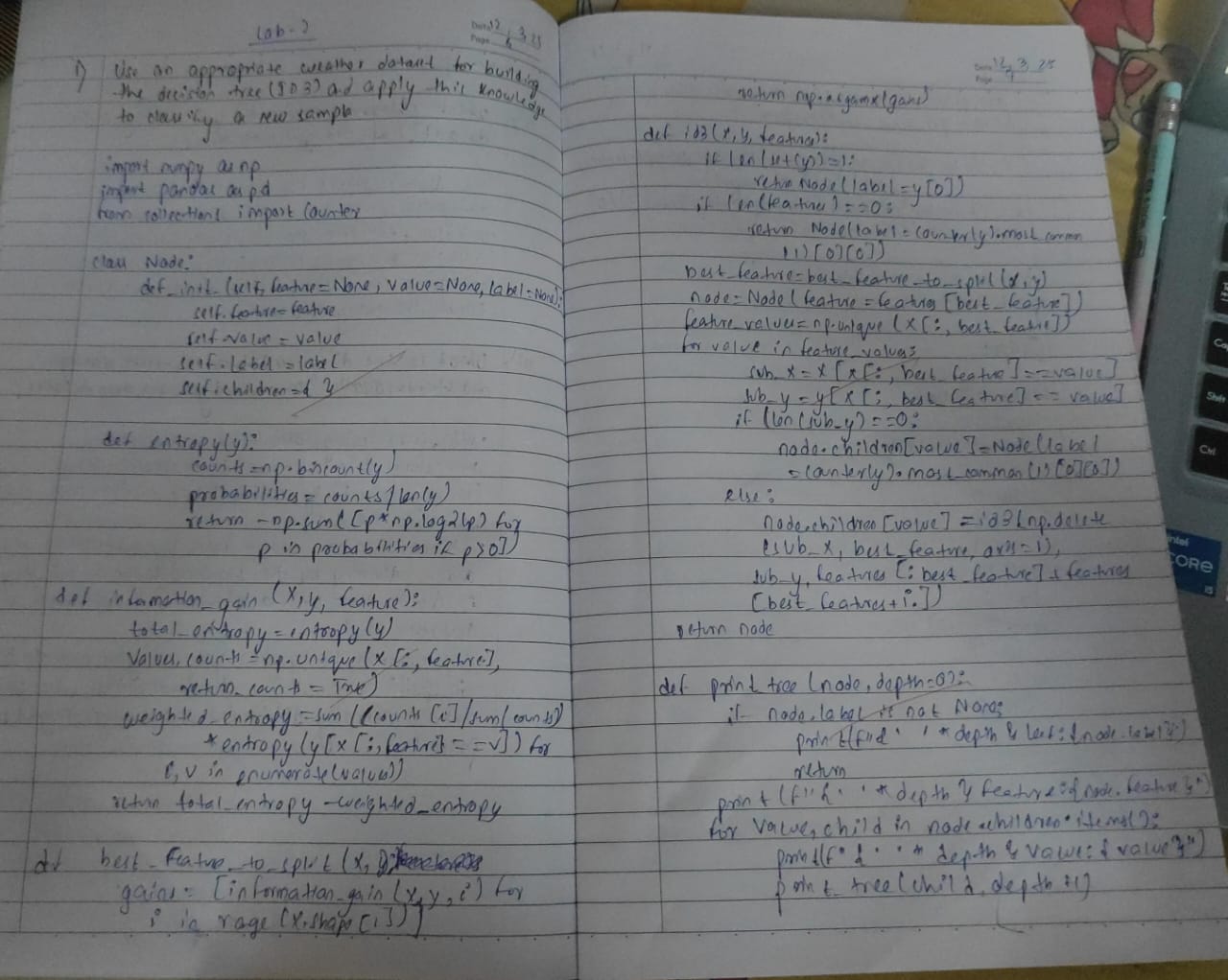
Accuracy of the Multinomial Logistic Regression model: 0.95

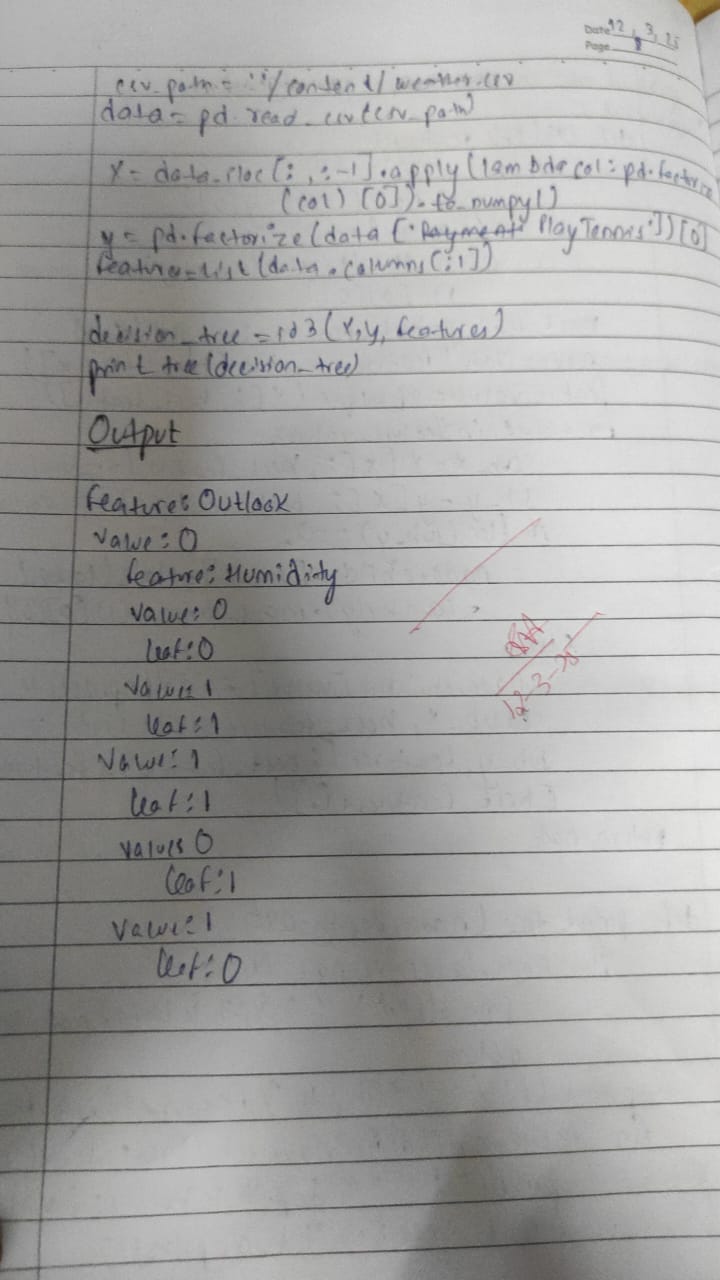


##### Program 5

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

Screenshot





Code:

import numpy as np

import pandas as pd

from collections import Counter

class Node:

def \_\_init\_\_(self, feature=None, value=None, label=None):

self.feature = feature *# Attribute to split on*

self.value = value *# Value of the attribute*

self.label = label *# Label if it’s a leaf node*

self.children = {} *# Dictionary of child nodes*

def entropy(y):

counts = np.bincount(y)

probabilities = counts / len(y)

return -np.sum([p \* np.log2(p) for p in probabilities if p > 0])

def information\_gain(X, y, feature):

total\_entropy = entropy(y)

values, counts = np.unique(X[:, feature], return\_counts=True)

weighted\_entropy = sum((counts[i] / sum(counts)) \* entropy(y[X[:, feature] == v]) for i, v in enumerate(values))

return total\_entropy - weighted\_entropy

def best\_feature\_to\_split(X, y):

gains = [information\_gain(X, y, i) for i in range(X.shape[1])]

return np.argmax(gains)

def id3(X, y, features):

if len(set(y)) == 1:

return Node(label=y[0])

if len(features) == 0:

return Node(label=Counter(y).most\_common(1)[0][0])

best\_feature = best\_feature\_to\_split(X, y)

node = Node(feature=features[best\_feature])

feature\_values = np.unique(X[:, best\_feature])

for value in feature\_values:

sub\_X = X[X[:, best\_feature] == value]

sub\_y = y[X[:, best\_feature] == value]

if len(sub\_y) == 0:

node.children[value] = Node(label=Counter(y).most\_common(1)[0][0])

else:

node.children[value] = id3(np.delete(sub\_X, best\_feature, axis=1), sub\_y, features[:best\_feature] + features[best\_feature+1:])

return node

if node.label is not None:

print(f"{' ' \* depth}Leaf: {node.label}")

return

print(f"{' ' \* depth}Feature: {node.feature}")

for value, child in node.children.items():

print(f"{' ' \* depth}Value: {value}")

print\_tree(child, depth + 1)

*# Example dataset*

data = pd.DataFrame({

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

})

X = data.iloc[:, :-1].apply(lambda col: pd.factorize(col)[0]).to\_numpy()

y = pd.factorize(data['PlayTennis'])[0]

features = list(data.columns[:-1])

decision\_tree = id3(X, y, features)

print\_tree(decision\_tree)

Feature: Outlook

Value: 0

Feature: Humidity

Value: 0

Leaf: 0

Value: 1

Leaf: 1

Value: 1

Leaf: 1

Value: 2

Feature: Wind

Value: 0

Leaf: 1

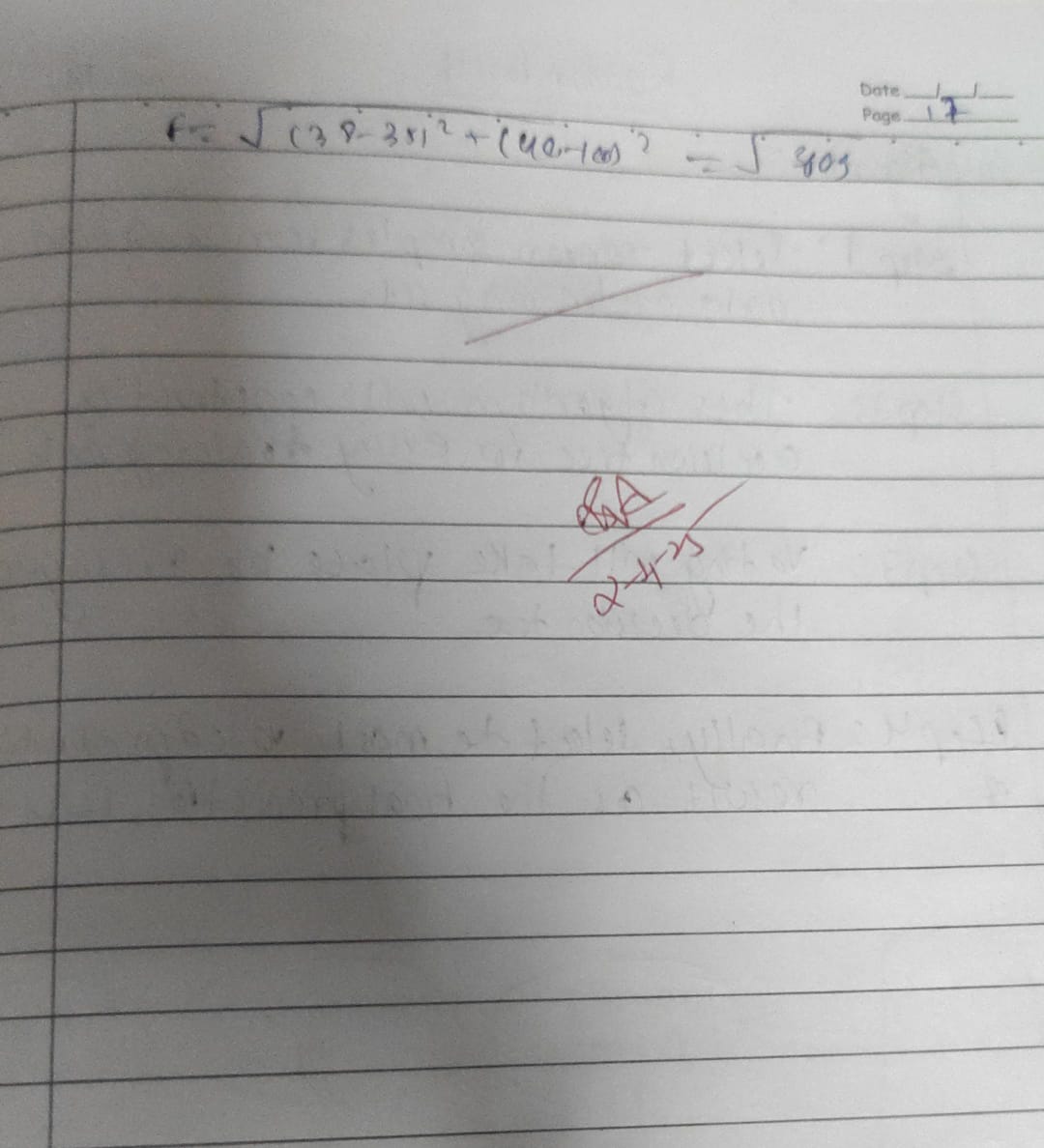
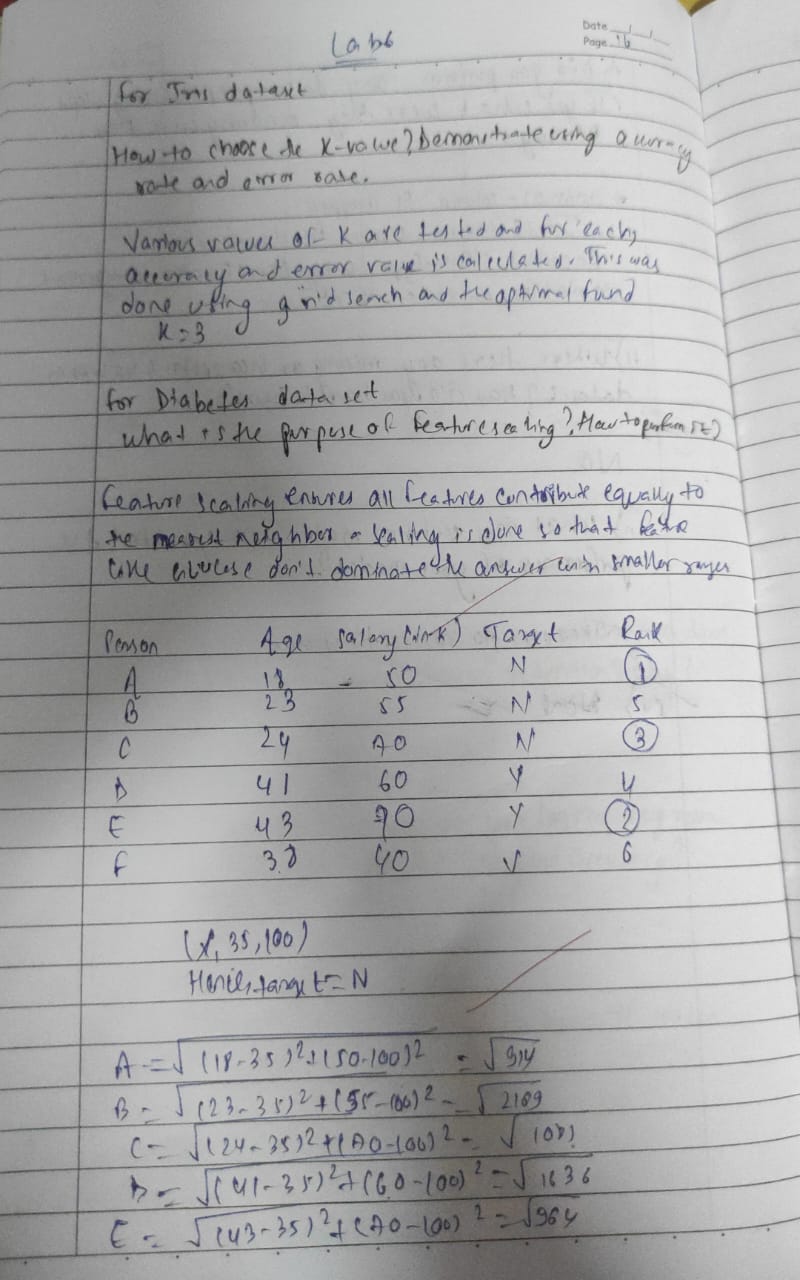
Value: 1

Leaf: 0

##### Program 6

Build KNN Classification model for a given dataset

Screenshot



Code:

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

*# Function to train and evaluate KNN model*

def knn\_classification(data\_path, target\_column, dataset\_name, k=5):

*# Load dataset*

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

*# Split features and target*

X = df.drop(columns=[target\_column])

y = df[target\_column]

*# Split data into training and testing sets*

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

*# Feature scaling for better performance*

scaler = StandardScaler()

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

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

*# Train KNN model*

model = KNeighborsClassifier(n\_neighbors=k)

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

*# Make predictions*

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

*# Evaluate model*

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

print(f'Accuracy of KNN on {dataset\_name} dataset: {accuracy:.4f}')

print("Classification Report:")

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

*# Confusion matrix*

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

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

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title(f'Confusion Matrix - {dataset\_name}')

plt.show()

*# Run KNN classification on both datasets*

knn\_classification('/content/iris (3).csv', 'species', 'Iris', k=5)

knn\_classification('/content/diabetes.csv', 'Outcome', 'Diabetes', k=5)

Accuracy of KNN on Iris dataset: 1.0000

Classification Report:

precision recall f1-score support

setosa 1.00 1.00 1.00 10

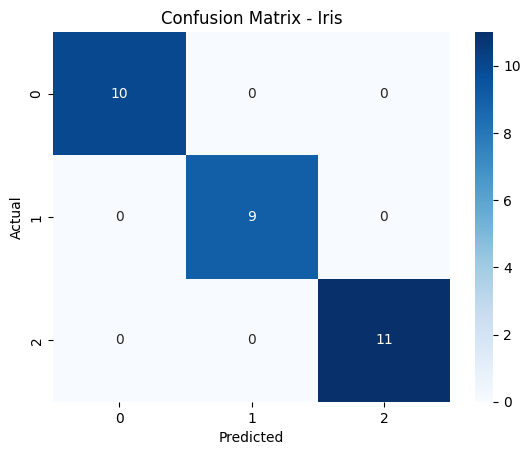
versicolor 1.00 1.00 1.00 9

virginica 1.00 1.00 1.00 11

accuracy 1.00 30

macro avg 1.00 1.00 1.00 30

weighted avg 1.00 1.00 1.00 30



Accuracy of KNN on Diabetes dataset: 0.6948

Classification Report:

precision recall f1-score support

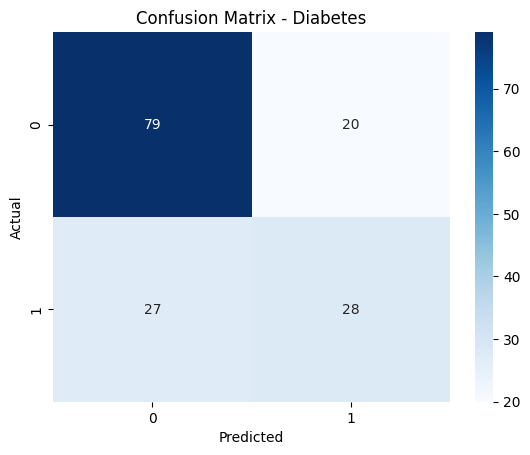
0 0.75 0.80 0.77 99

1 0.58 0.51 0.54 55

accuracy 0.69 154

macro avg 0.66 0.65 0.66 154

weighted avg 0.69 0.69 0.69 154



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.neighbors import KNeighborsClassifier

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

*# Load dataset*

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

*# Define features and target*

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

y = df['target']

*# Split data*

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

*# Feature scaling*

scaler = StandardScaler()

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

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

*# Find the best K value*

k\_values = range(1, 21)

accuracy\_scores = []

for k in k\_values:

model = KNeighborsClassifier(n\_neighbors=k)

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

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

accuracy\_scores.append(accuracy\_score(y\_test, y\_pred))

best\_k = k\_values[np.argmax(accuracy\_scores)]

print(f'Best K value: {best\_k}')

*# Train model with best K*

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

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

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

*# Evaluate model*

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

print(f'Accuracy with best K ({best\_k}): {accuracy:.4f}')

print("Classification Report:")

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

*# Confusion matrix*

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

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

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title(f'Confusion Matrix - KNN (K={best\_k})')

plt.show()

*# Plot K values vs. Accuracy*

plt.plot(k\_values, accuracy\_scores, marker='o')

plt.xlabel('K Value')

plt.ylabel('Accuracy')

plt.title('K Value vs Accuracy')

plt.show()

Best K value: 7

Accuracy with best K (7): 0.9180

Classification Report:

precision recall f1-score support

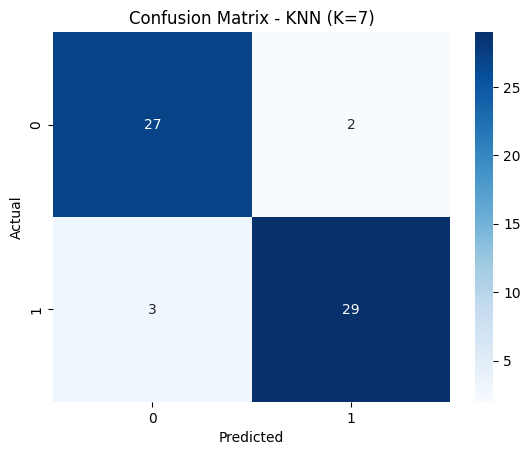
0 0.90 0.93 0.92 29

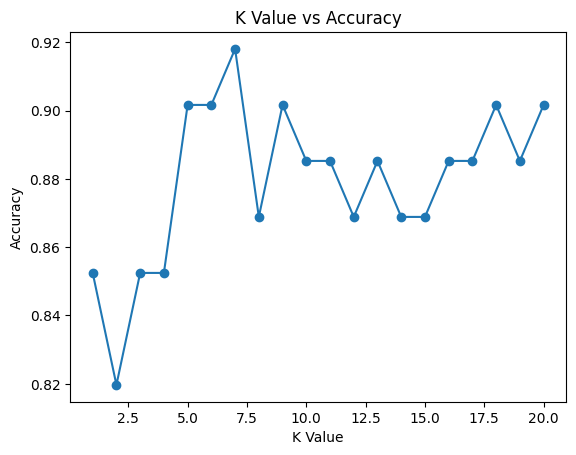
1 0.94 0.91 0.92 32

accuracy 0.92 61

macro avg 0.92 0.92 0.92 61

weighted avg 0.92 0.92 0.92 61





##### Program 7

Build Support vector machine model for a given dataset

Code:

import numpy as np

import matplotlib.pyplot as plt

*# Define the Linear SVM class*

class LinearSVM:

def \_\_init\_\_(self, learning\_rate=0.001, reg\_strength=0.1, num\_iterations=1000):

self.learning\_rate = learning\_rate

self.reg\_strength = reg\_strength

self.num\_iterations = num\_iterations

def fit(self, X, y):

*# Initialize weights and bias*

num\_samples, num\_features = X.shape

self.W = np.zeros(num\_features) *# Weights*

self.b = 0 *# Bias*

*# Gradient Descent*

for \_ in range(self.num\_iterations):

*# Compute the margin (decision function)*

margins = 1 - y \* (np.dot(X, self.W) + self.b)

*# Compute gradient*

dw = -2 \* np.dot(X.T, (y \* (margins > 0))) / num\_samples + 2 \* self.reg\_strength \* self.W

db = -2 \* np.sum(y \* (margins > 0)) / num\_samples

*# Update weights and bias*

self.W -= self.learning\_rate \* dw

self.b -= self.learning\_rate \* db

def predict(self, X):

*# Make predictions*

return np.sign(np.dot(X, self.W) + self.b)

*# Generate toy data (binary classification)*

np.random.seed(42)

num\_samples = 100

X = np.random.randn(num\_samples, 2)

y = np.ones(num\_samples)

y[X[:, 0] < X[:, 1]] = -1 *# Assign different class based on condition*

*# Train the Linear SVM*

svm = LinearSVM(learning\_rate=0.001, reg\_strength=0.1, num\_iterations=1000)

svm.fit(X, y)

*# Predict*

y\_pred = svm.predict(X)

*# Visualize the decision boundary*

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')

ax = plt.gca()

xlim = ax.get\_xlim()

ylim = ax.get\_ylim()

xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100), np.linspace(ylim[0], ylim[1], 100))

Z = svm.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')

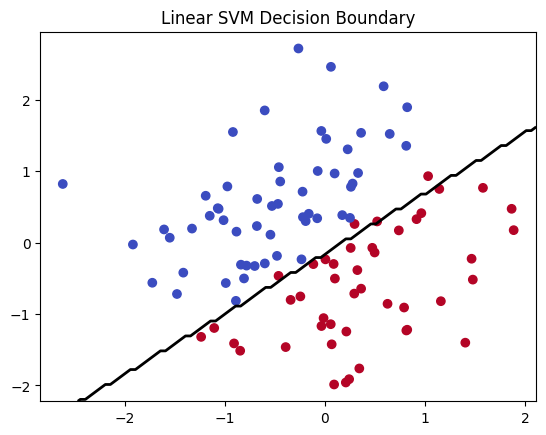
plt.title("Linear SVM Decision Boundary")

plt.show()

*# Print accuracy (simple comparison)*

accuracy = np.mean(y\_pred == y)

print(f"Accuracy: {accuracy \* 100:.2f}%")

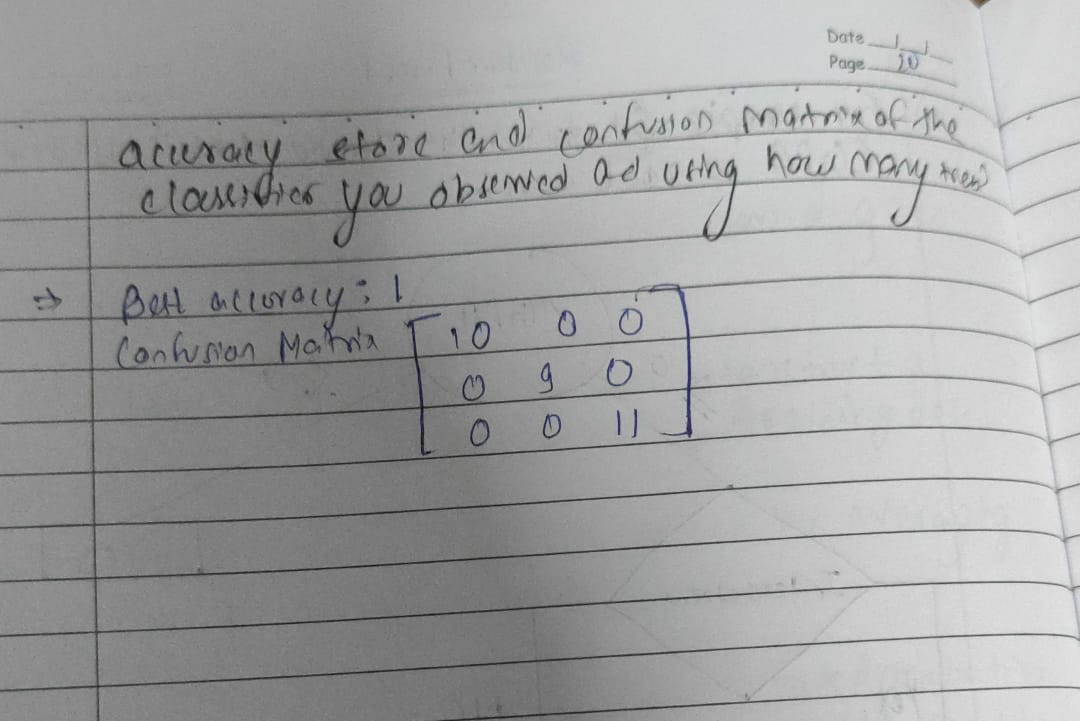
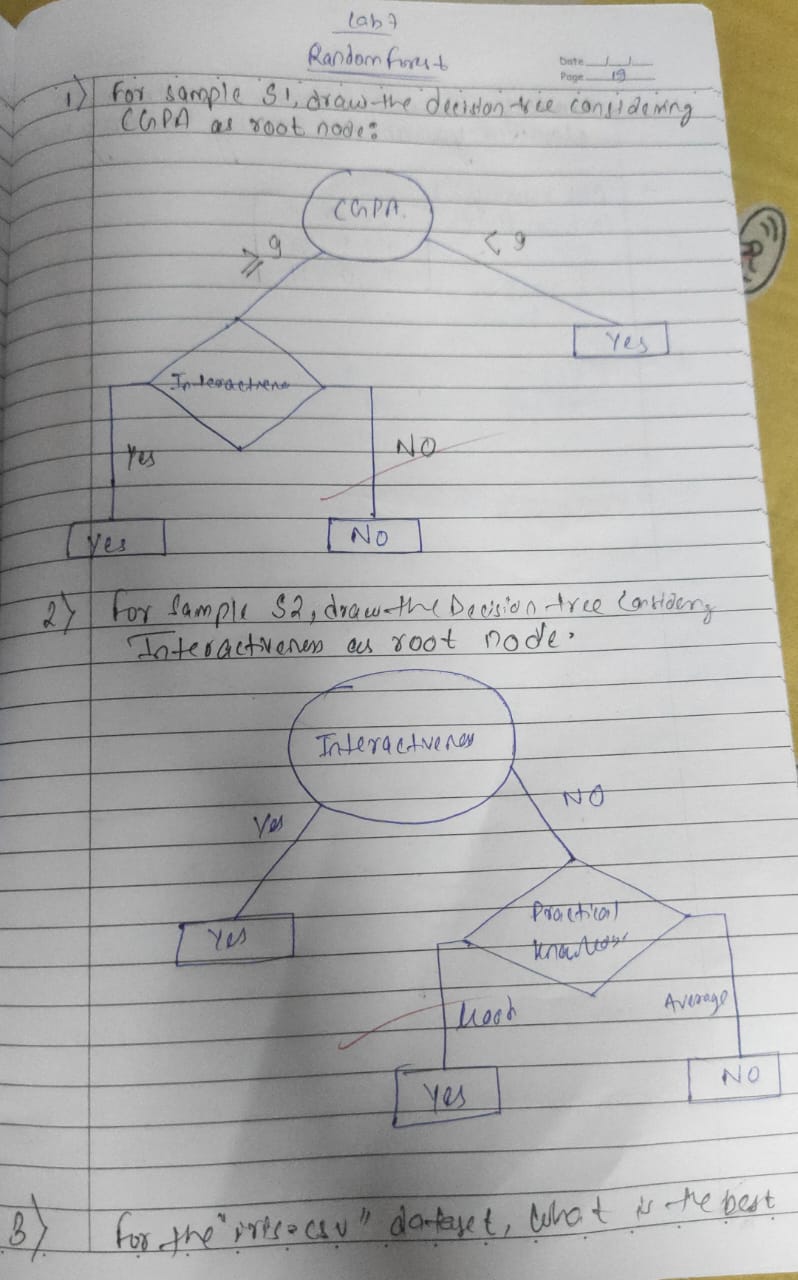
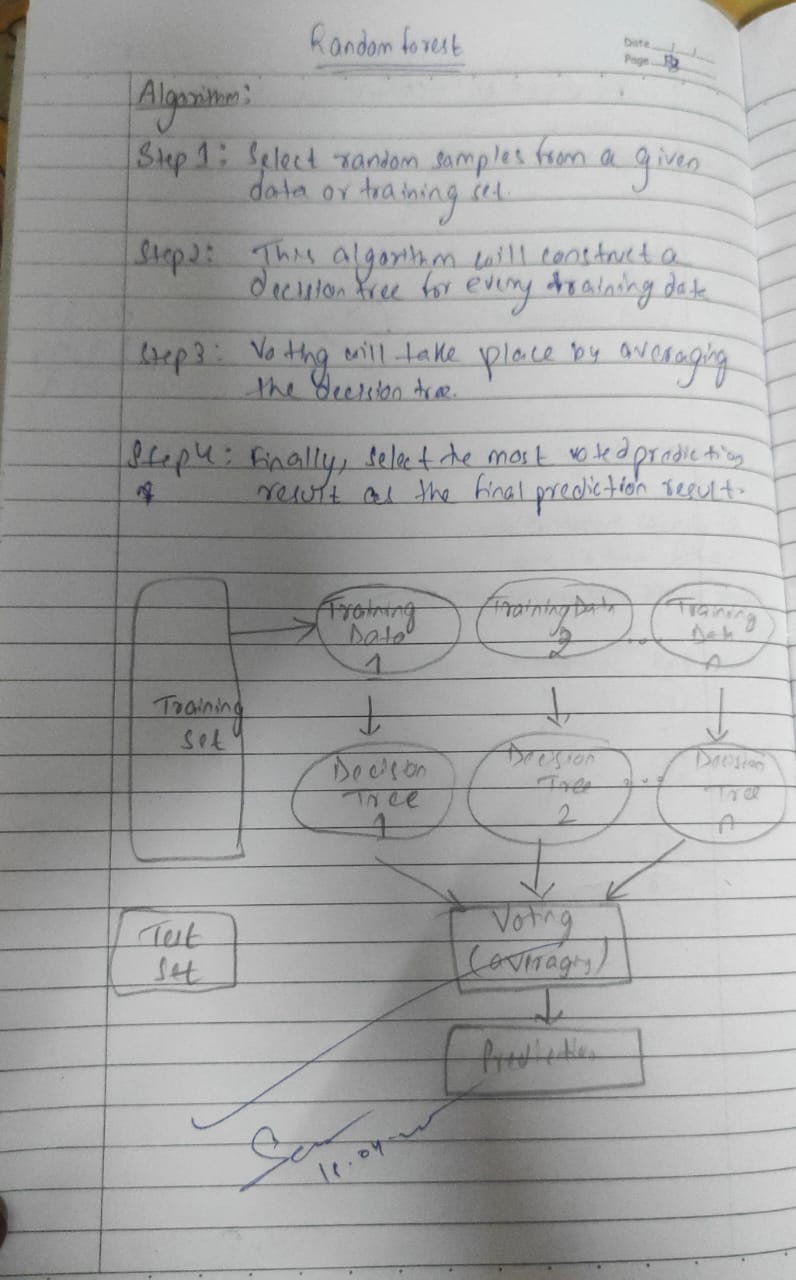


Accuracy: 96.00%

##### Program 8

Implement Random forest ensemble method on a given dataset

Screenshot



Code:

import pandas as pd

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

*# Load the iris dataset from CSV*

df = pd.read\_csv("/content/iris (2).csv")

*# Assuming last column is the label*

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

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

*# Split into training and test sets (70% train, 30% test)*

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

*# 1. Train RF Classifier with default n\_estimators=10*

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

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

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

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

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

best\_accuracy = 0

best\_n = 0

accuracies = []

for n in range(1, 101):

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

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

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

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

accuracies.append(acc)

if acc > best\_accuracy:

best\_accuracy = acc

best\_n = n

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

*# Plot accuracy vs. number of trees*

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

plt.plot(range(1, 101), accuracies, marker='o')

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

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

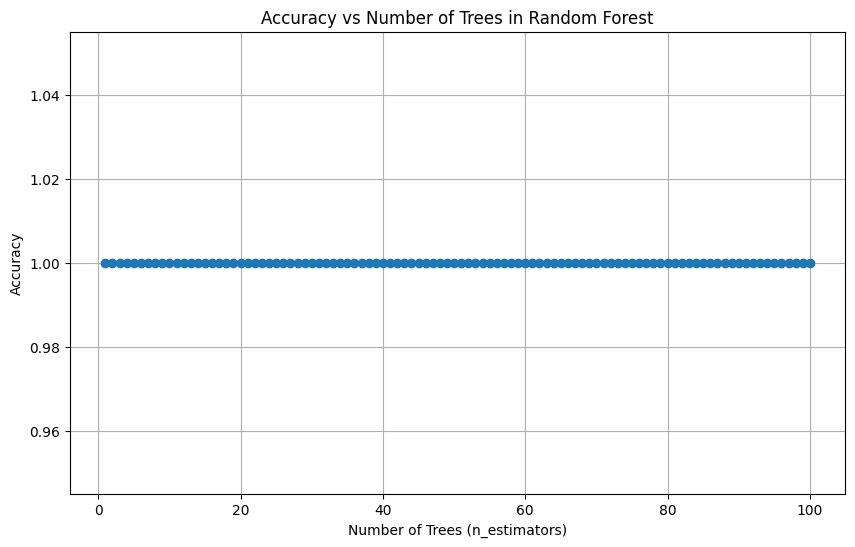
plt.ylabel("Accuracy")

plt.grid(True)

plt.show()

Default RF Accuracy (n\_estimators=10): 1.0000

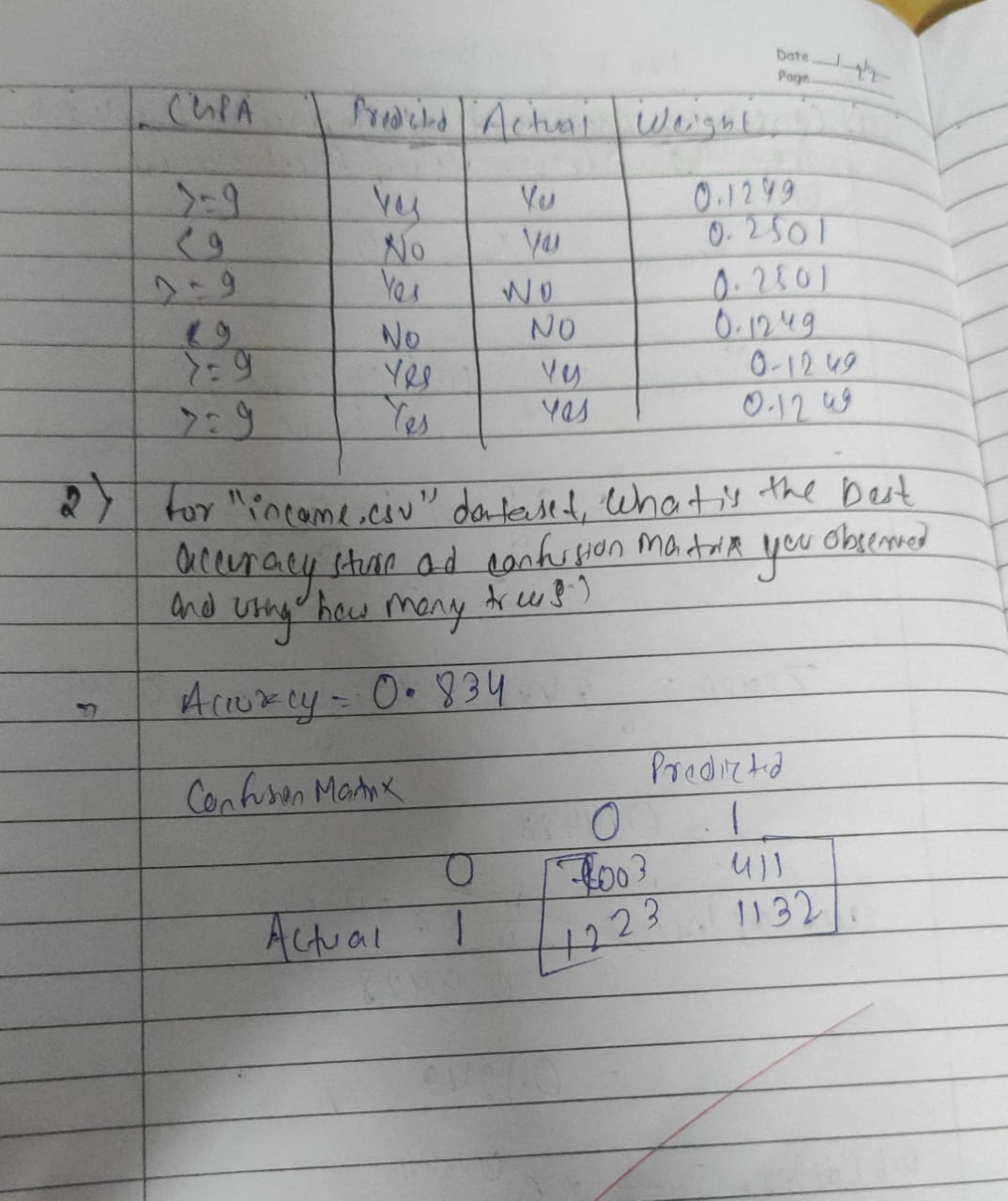
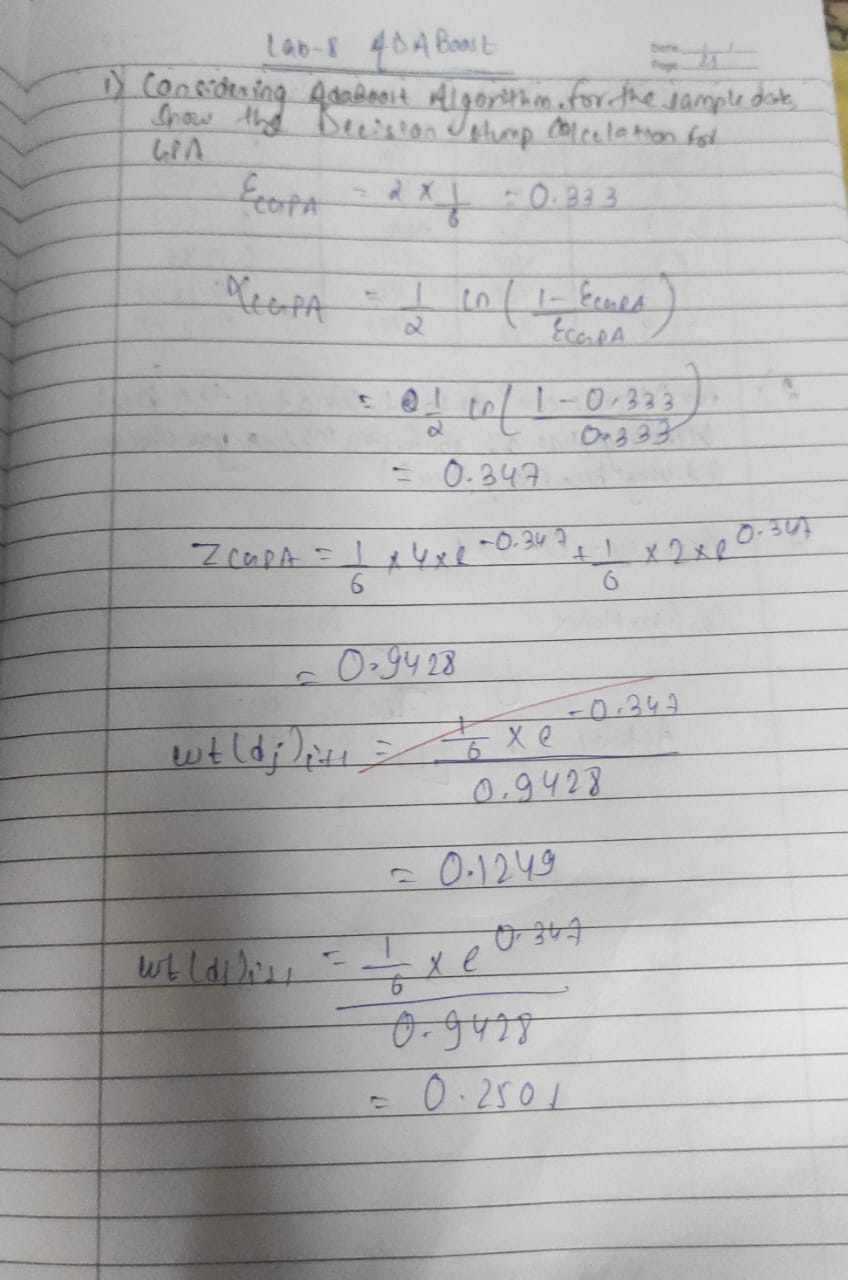
Best RF Accuracy: 1.0000 with n\_estimators = 1



##### 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

from sklearn.ensemble import AdaBoostClassifier

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

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

*# Step 1: Load the dataset*

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

*# Step 2: Split into features and target*

X = df.drop(columns=['income\_level'])

y = df['income\_level']

*# Step 3: Train-test split*

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

*# Step 4: AdaBoost with 10 estimators*

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

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

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

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

conf\_matrix\_10 = confusion\_matrix(y\_test, y\_pred\_10)

print("Accuracy with 10 estimators:", round(accuracy\_10, 4))

print("Confusion Matrix (10 estimators):\n", conf\_matrix\_10)

*# Step 5: Fine-tune number of trees (1 to 50)*

best\_accuracy = 0

best\_n = 0

accuracies = []

for n in range(1, 51):

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

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

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

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

accuracies.append(acc)

if acc > best\_accuracy:

best\_accuracy = acc

best\_n = n

print(f"\nBest Accuracy: {round(best\_accuracy, 4)} with n\_estimators = {best\_n}")

*# Step 6: Plot accuracy vs. number of estimators*

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

plt.plot(range(1, 51), accuracies, marker='o', linestyle='-', color='blue')

plt.title('Accuracy vs Number of Trees (n\_estimators)')

plt.xlabel('Number of Trees')

plt.ylabel('Accuracy')

plt.grid(True)

plt.tight\_layout()

plt.show()

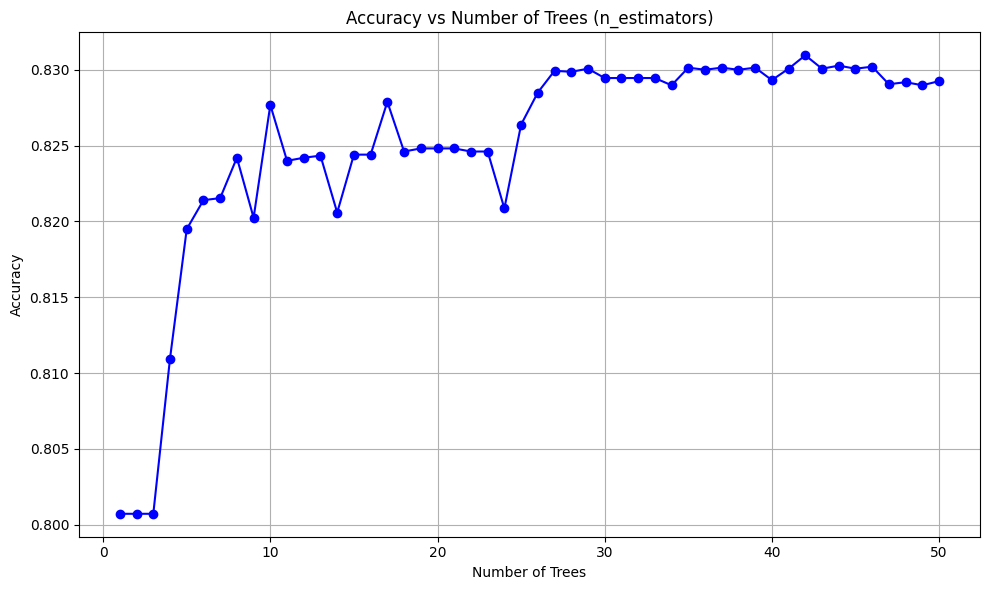
Accuracy with 10 estimators: 0.8277

Confusion Matrix (10 estimators):

[[10722 387]

[ 2138 1406]]

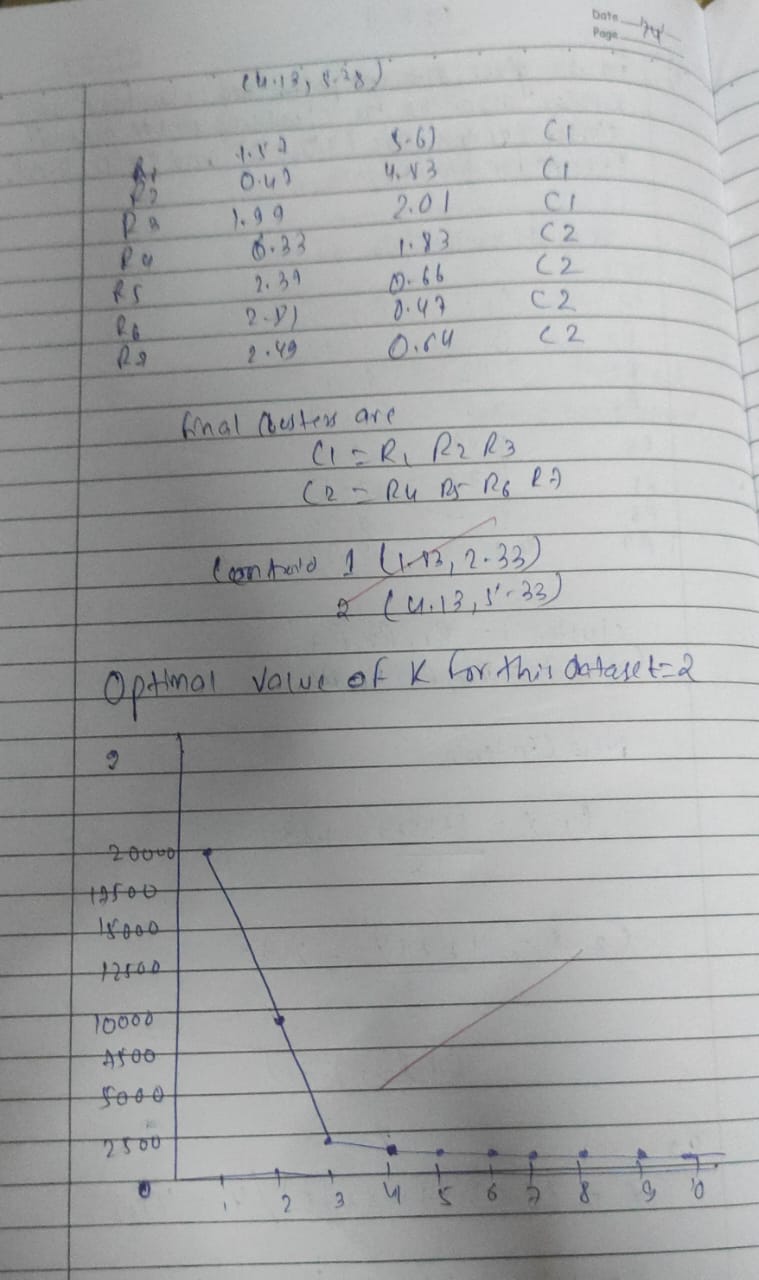
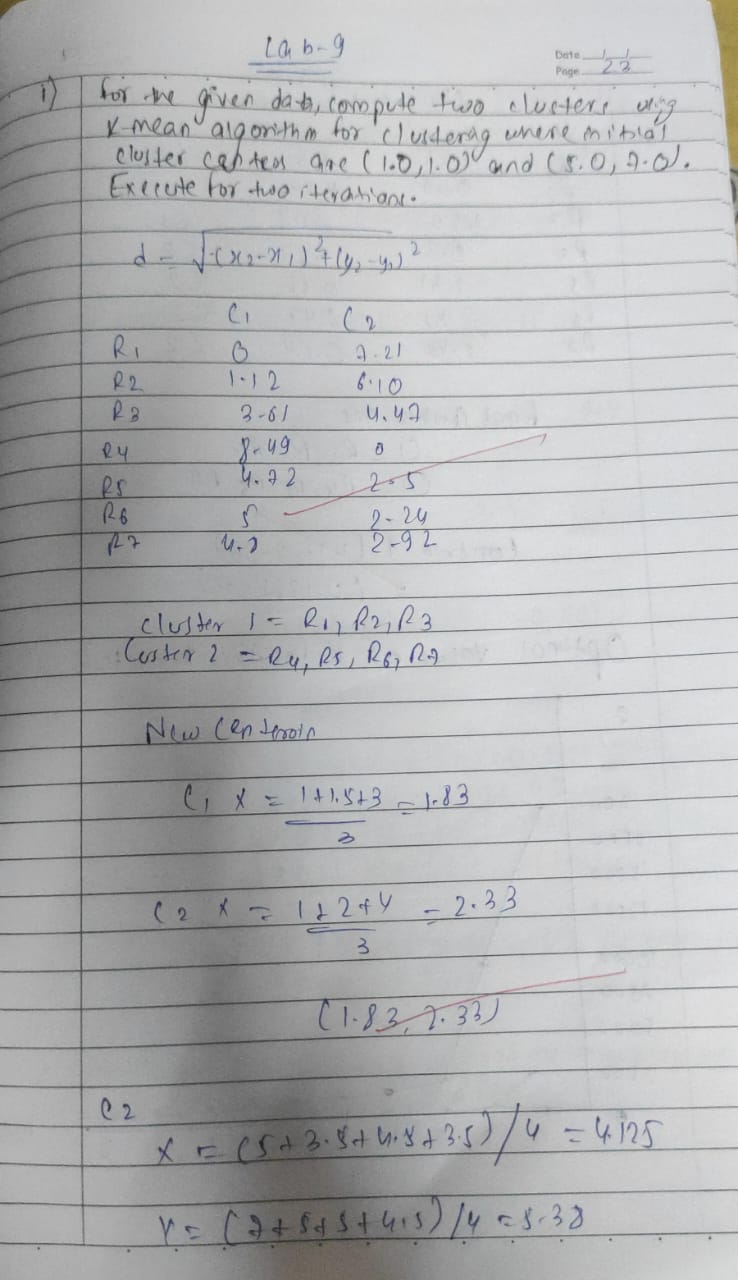
Best Accuracy: 0.831 with n\_estimators = 42



##### Program 10

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

Screenshot



Code:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

*# Load the dataset*

df = pd.read\_csv("/content/iris (2).csv")

*# Select only petal length and petal width*

X = df[['petal\_length', 'petal\_width']]

*# Optional: Standardize the data*

scaler = StandardScaler()

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

*# Elbow method to determine optimal k*

inertia = []

k\_range = range(1, 11)

for k in k\_range:

model = KMeans(n\_clusters=k, random\_state=42, n\_init=10)

model.fit(X\_scaled)

inertia.append(model.inertia\_)

*# Plot the elbow graph*

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

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

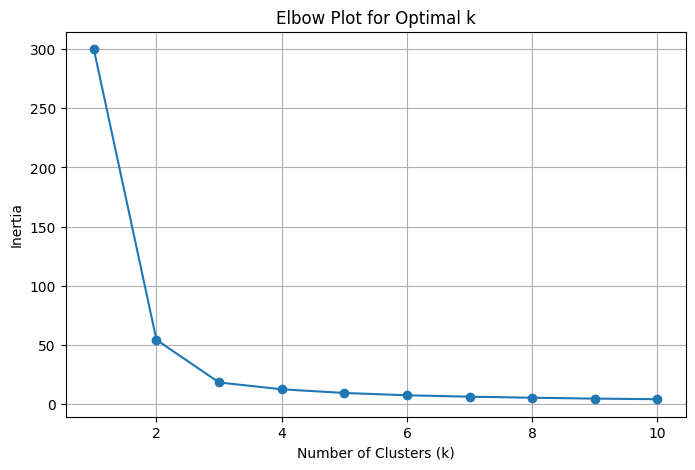
plt.title('Elbow Plot for Optimal k')

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

plt.ylabel('Inertia')

plt.grid(True)

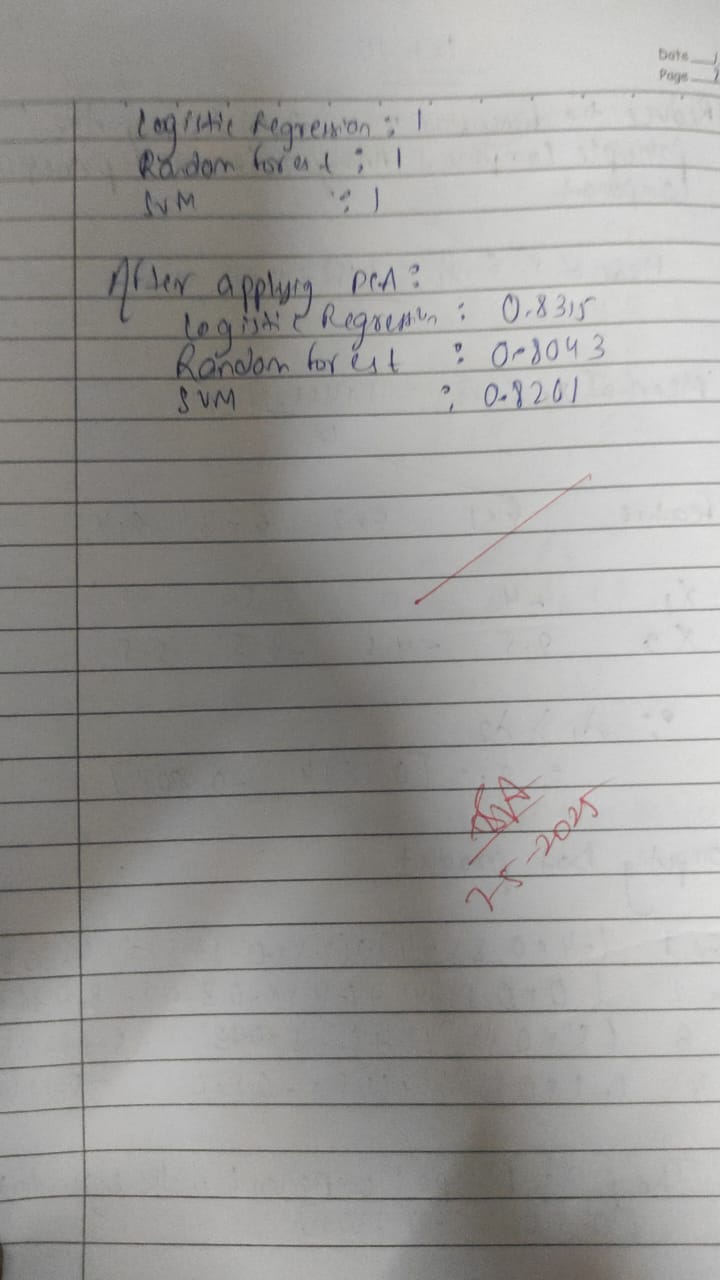
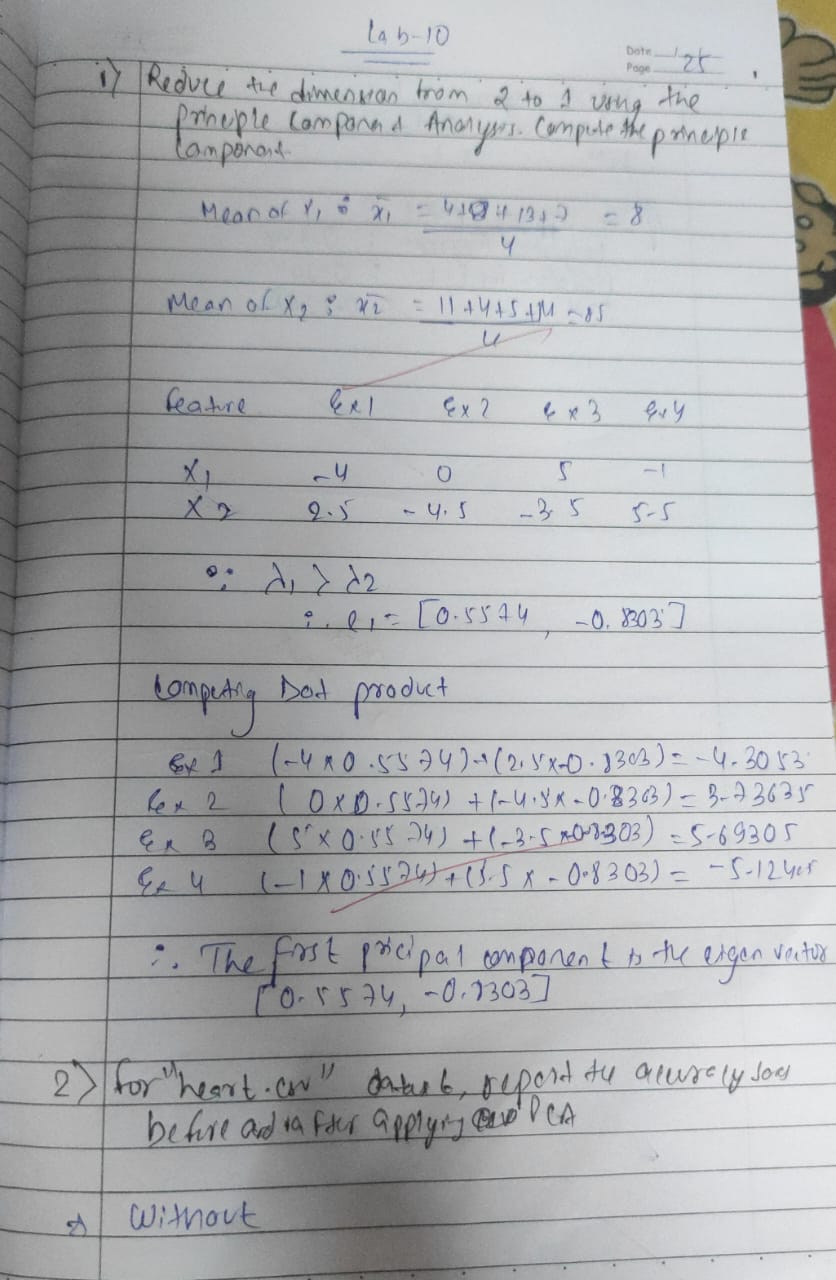
plt.show()



##### Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot



Code:

import pandas as pd

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from sklearn.svm import SVC

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy\_score

*# Load dataset*

df = pd.read\_csv("/content/heart (1).csv") *# Update to match your file path if needed*

*# Define features and target*

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

y = df['HeartDisease']

*# Identify categorical columns*

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

*# Encode categorical columns*

for col in categorical\_cols:

if X[col].nunique() == 2:

X[col] = LabelEncoder().fit\_transform(X[col])

else:

X = pd.get\_dummies(X, columns=[col])

*# Scale features*

scaler = StandardScaler()

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

*# Train-test split*

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

*# Initialize models*

models = {

'SVM': SVC(),

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

'Random Forest': RandomForestClassifier()

}

*# Train and evaluate models (without PCA)*

print("🔍 Accuracy without PCA:")

for name, model in models.items():

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

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

print(f"{name}: {accuracy\_score(y\_test, y\_pred):.4f}")

*# Apply PCA (reduce to 5 components)*

pca = PCA(n\_components=5)

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

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

*# Train and evaluate models (with PCA)*

print("\n📉 Accuracy with PCA:")

for name, model in models.items():

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

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

print(f"{name}: {accuracy\_score(y\_test\_pca, y\_pred\_pca):.4f}")

🔍 Accuracy without PCA:

SVM: 0.8804

Logistic Regression: 0.8533

Random Forest: 0.8859

📉 Accuracy with PCA:

SVM: 0.8424

Logistic Regression: 0.8641

Random Forest: 0.8533