# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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



## LAB REPORT

### on

Machine Learning (23CS6PCMAL)

#### Submitted by

**Abhishek Yadav (1BM22CS009)**

#### 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

## Feb-2025 to June-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 **Abhishek Yadav (1BM22CS009),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

|  |  |
| --- | --- |
| Dr. Seema Patil  Assistant Professor  Department of CSE, BMSCE | Dr. Kavitha Sooda  Professor & HOD  Department of CSE, BMSCE |

# Index

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 4-03-2025 | Write a python program to import and export data using Pandas library functions | 1-4 |
| 2 | 4-03-2025 | Demonstrate various data pre-processing techniques for a given dataset | 5-7 |
| 3 | 11-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 8-16 |
| 4 | 18-3-2025 | Build Logistic Regression Model for a given dataset | 17-20 |
| 5 | 25-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 21-22 |
| 6 | 01-4-2025 | Build KNN Classification model for a given dataset. | 23-24 |
| 7 | 08-4-2025 | Build Support vector machine model for a given dataset | 25-27 |
| 8 | 15-5-2025 | Implement Random forest ensemble method on a given dataset. | 28-30 |
| 9 | 15-5-2025 | Implement Boosting ensemble method on a given dataset. | 31-33 |
| 10 | 29-5-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 34-36 |
| 11 | 29-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 37-41 |

Github Link:

##### <https://github.com/AbhishekCS22/6A-ML-LAB-Batch1>

##### Program 1

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

**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('/content/Dataset of Diabetes .csv')

df.head()

df.shape

print(df.info())

*# Summary statistics*

print(df.describe())

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

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

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

if len(categorical\_cols) > 0:

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

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

print(df.head())

else:

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

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import pandas as pd

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

scaler = MinMaxScaler()

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

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

scaler = StandardScaler()

df\_standard = df.copy()

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

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

print(df\_minmax.head())

print("\nDataFrame after Standardization:")

print(df\_standard.head())

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

df1.head()

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import pandas as pd

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

scaler = MinMaxScaler()

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

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

scaler = StandardScaler()

df\_standard = df1.copy()

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

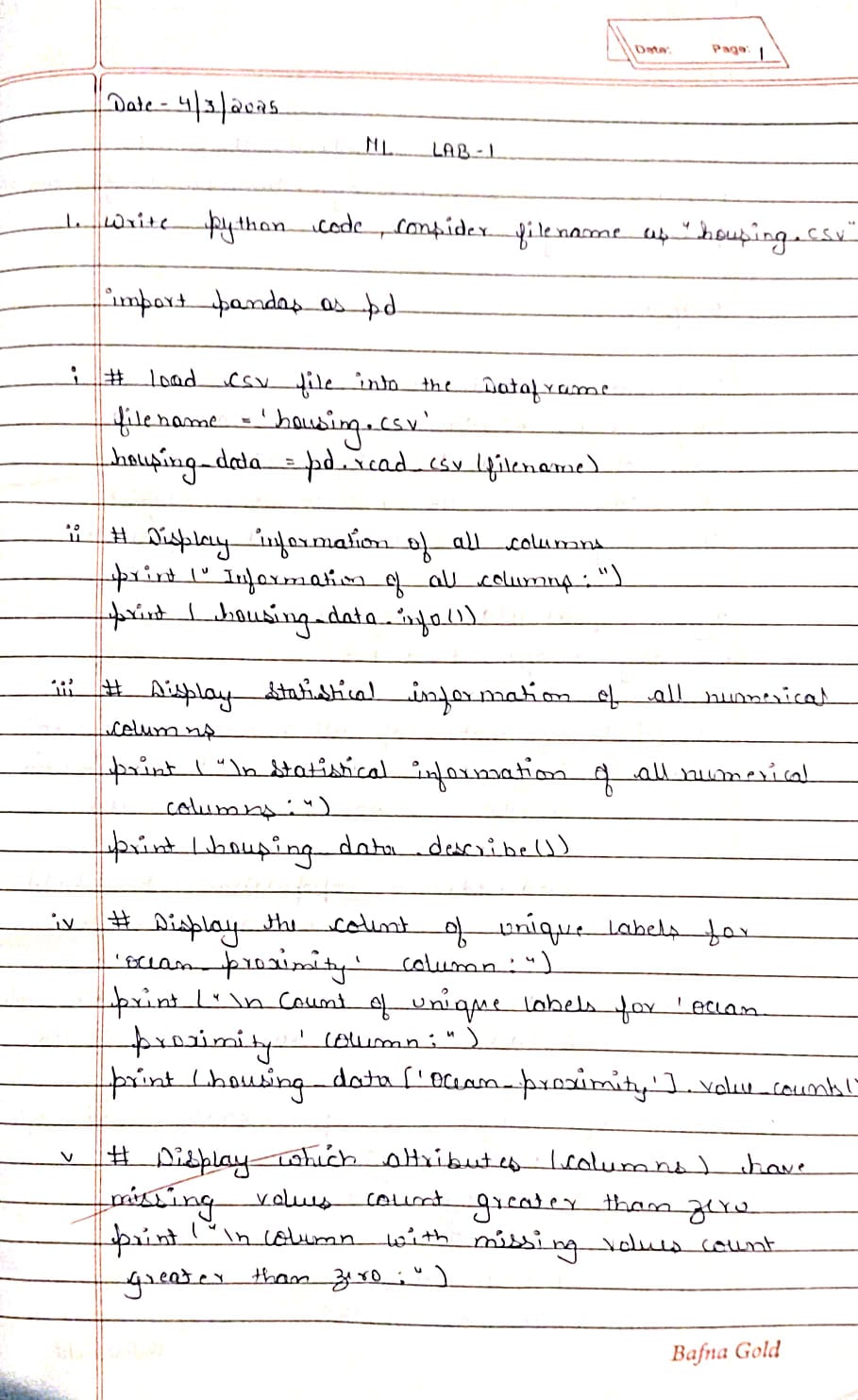
print("\nDataFrame after Min-Max Scaling:")

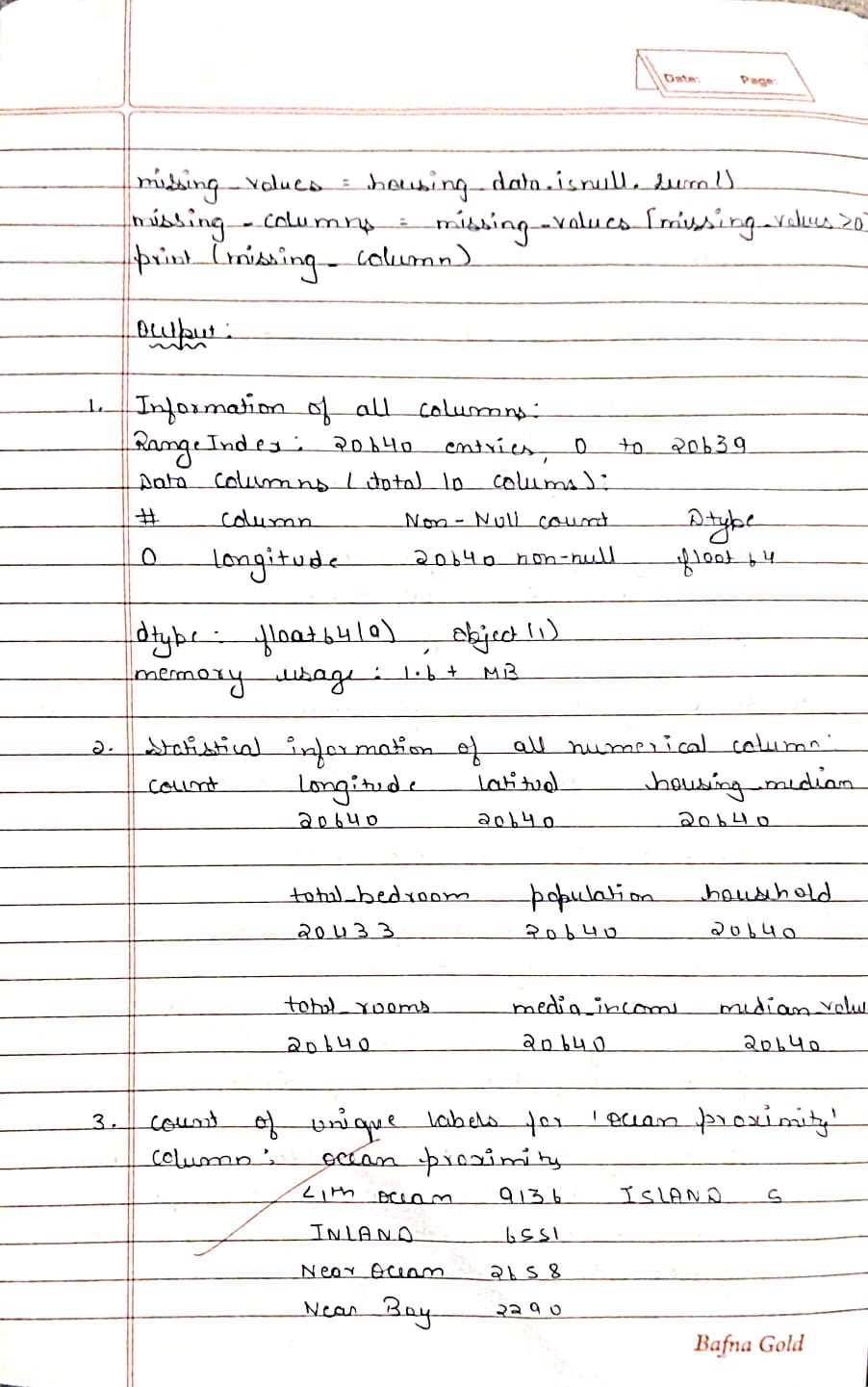
print(df\_minmax.head())

print("\nDataFrame after Standardization:")

print(df\_standard.head())

**Observation:**

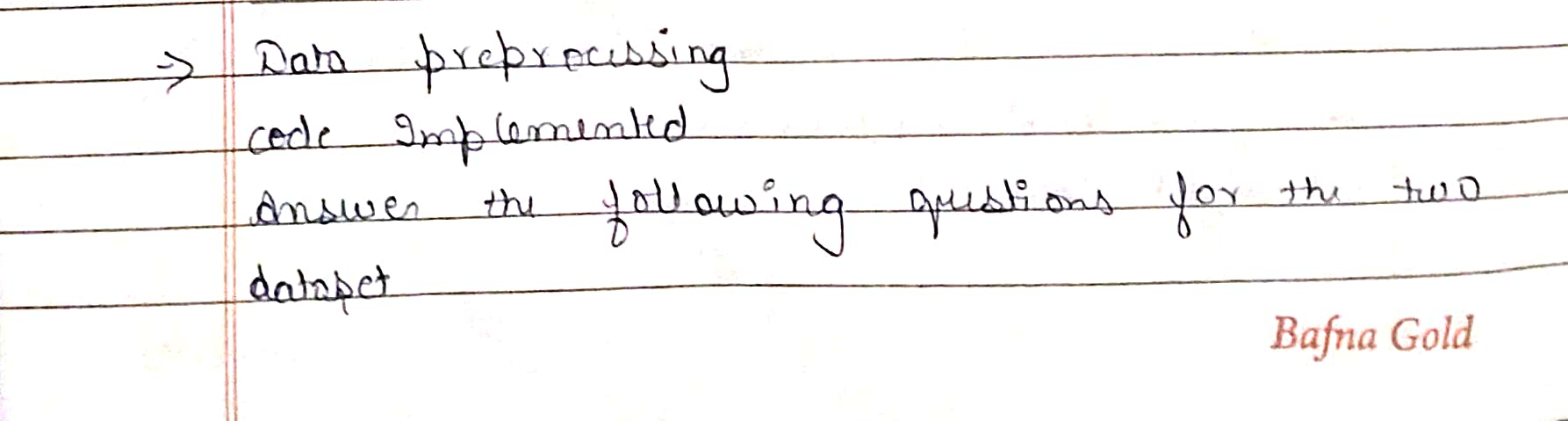


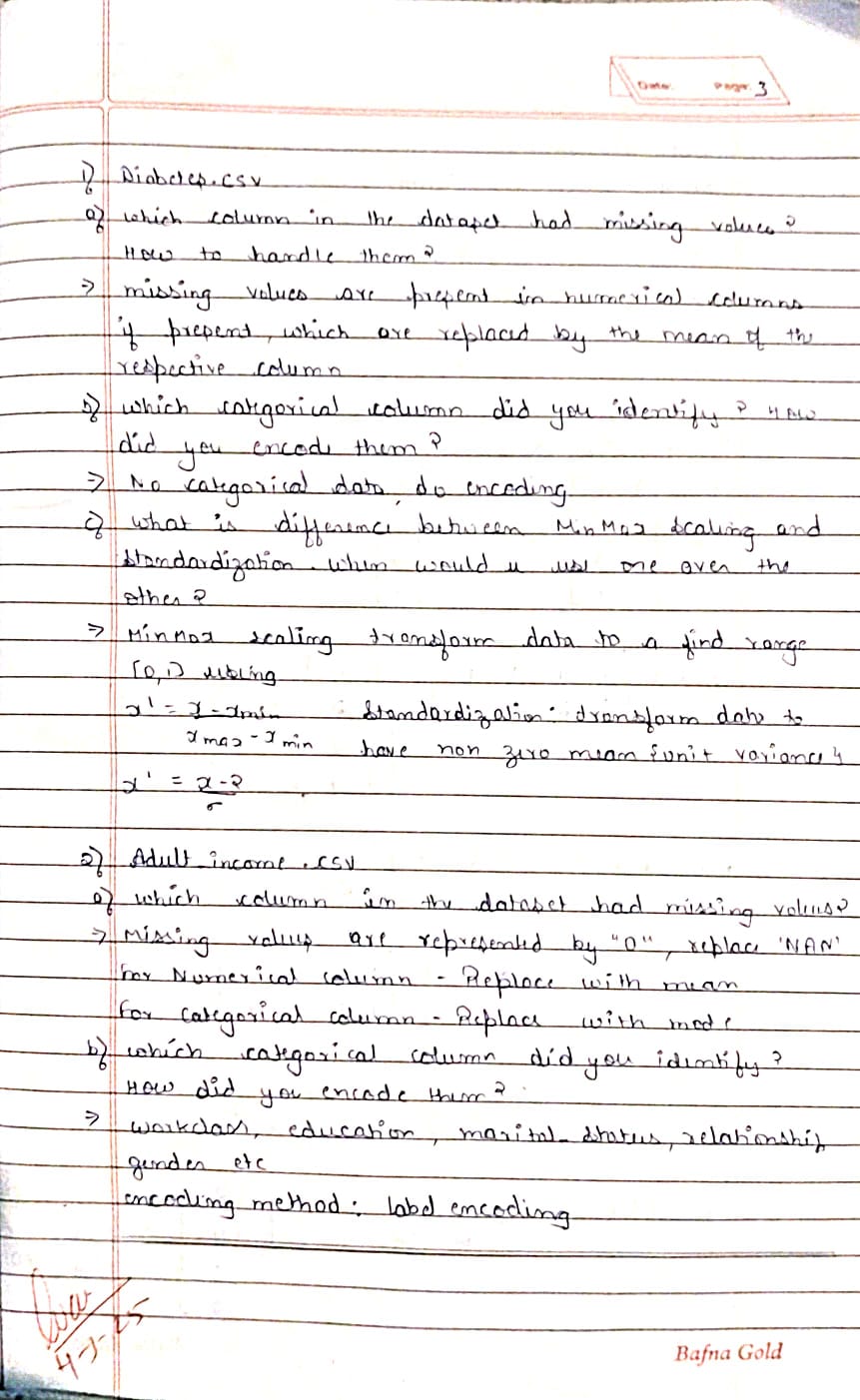


**Program 2**

Demonstrate various data pre-processing techniques for a given dataset.

**Observation:**

****

****

**Code:**

import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder

from google.colab import files

*# Upload Files Manually in Google Colab*

uploaded = files.upload()

*# Load the datasets (replace filenames accordingly after uploading)*

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

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

*# --- Data Cleaning ---*

*# Handling Missing Values: Fill numerical columns with median, categorical with mode*

for df in [diabetes\_df, adult\_df]:

for col in df.columns:

if df[col].isnull().sum() > 0:

if df[col].dtype == "object":

df[col].fillna(df[col].mode()[0], inplace=True)

else:

df[col].fillna(df[col].median(), inplace=True)

*# Handling Outliers: Capping values beyond 1.5\*IQR*

for df in [diabetes\_df, adult\_df]:

for col in df.select\_dtypes(include=np.number).columns:

Q1, Q3 = df[col].quantile(0.25), df[col].quantile(0.75)

IQR = Q3 - Q1

lower, upper = Q1 - 1.5 \* IQR, Q3 + 1.5 \* IQR

df[col] = np.clip(df[col], lower, upper)

*# --- Handling Categorical Data ---*

for df in [diabetes\_df, adult\_df]:

categorical\_cols = df.select\_dtypes(include="object").columns

for col in categorical\_cols:

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

*# --- Data Transformations ---*

scaler\_minmax = MinMaxScaler()

scaler\_standard = StandardScaler()

for df in [diabetes\_df, adult\_df]:

numerical\_cols = df.select\_dtypes(include=np.number).columns

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

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

*# Save processed datasets*

diabetes\_df.to\_csv("processed\_diabetes.csv", index=False)

adult\_df.to\_csv("processed\_adult.csv", index=False)

*# Download processed files*

files.download("processed\_diabetes.csv")

files.download("processed\_adult.csv")

import pandas as pd

from google.colab import files

*# Upload Files Manually in Google Colab*

uploaded = files.upload()

*# Load the datasets*

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

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

*# Check for missing values*

missing\_diabetes = diabetes\_df.isnull().sum()

missing\_adult = adult\_df.isnull().sum()

*# Display columns with missing values*

print("Missing values in Diabetes Dataset:")

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

print("\nMissing values in Adult Income Dataset:")

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

print("Missing Values Count in Diabetes Dataset:")

print(missing\_diabetes)

print("\nMissing Values Count in Adult Income Dataset:")

print(missing\_adult)

categorical\_diabetes = diabetes\_df.select\_dtypes(include="object").columns.tolist()

categorical\_adult = adult\_df.select\_dtypes(include="object").columns.tolist()

*# Display categorical columns*

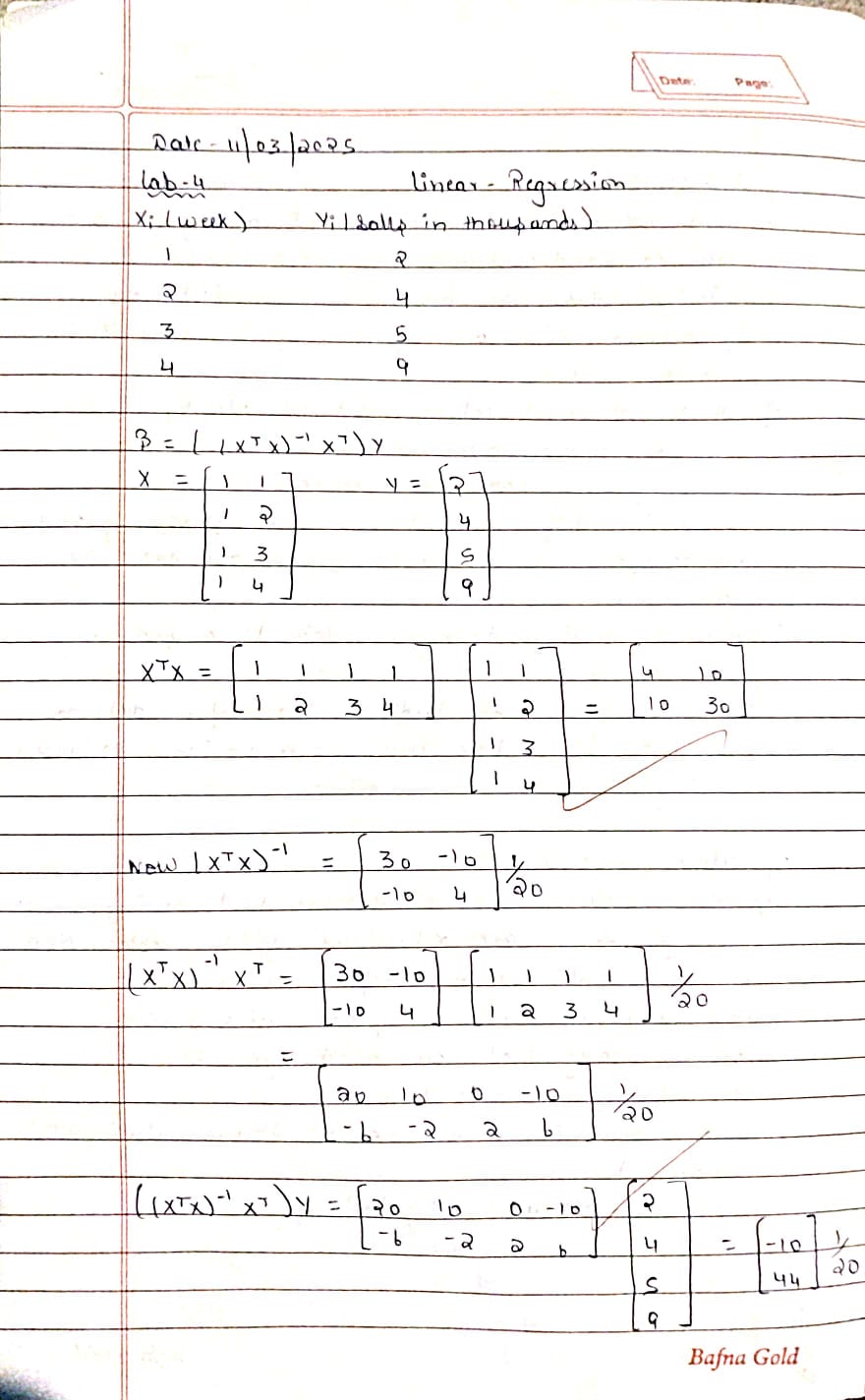
print("Categorical Columns in Diabetes Dataset:", categorical\_diabetes)

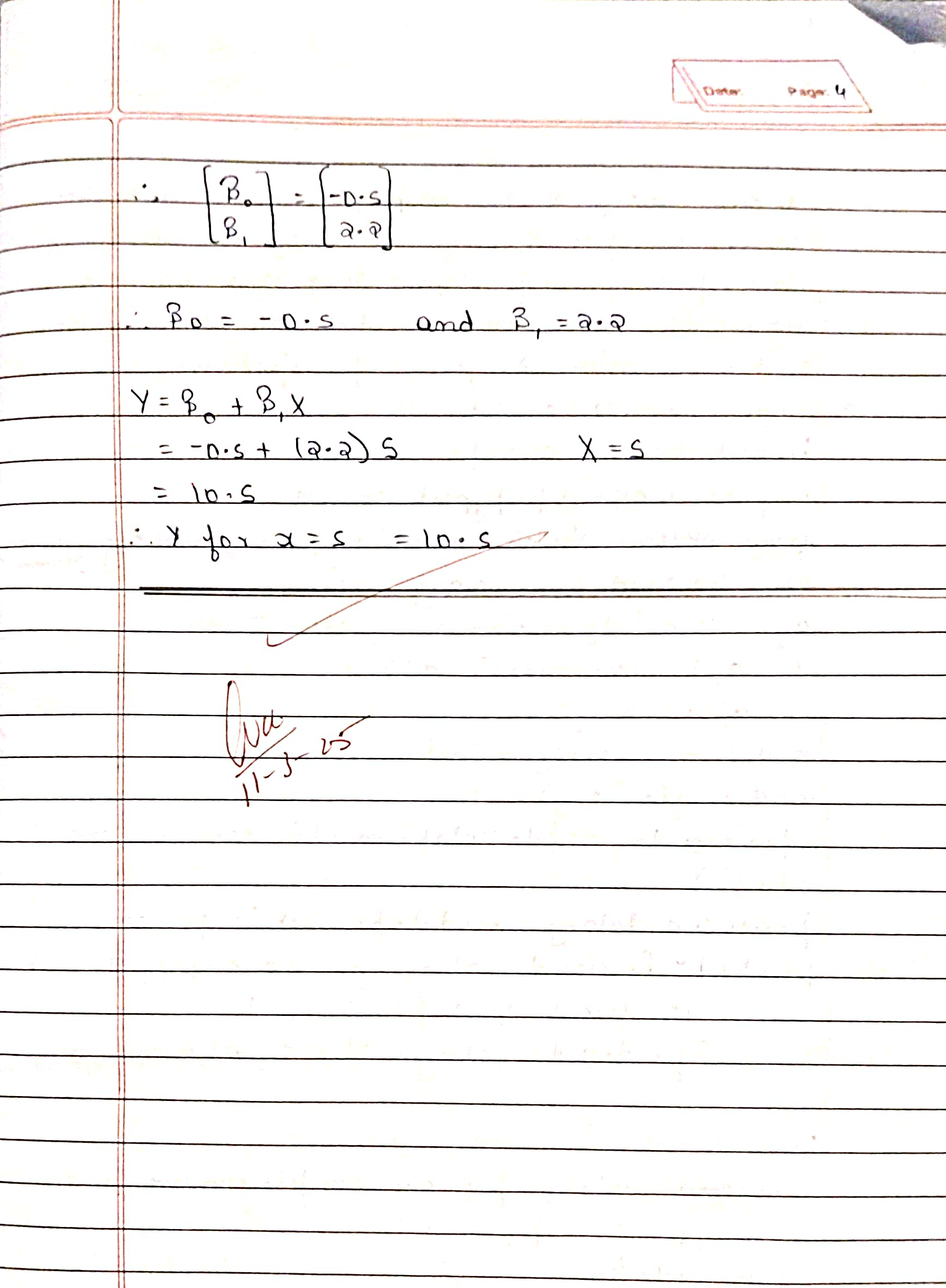
print("\nCategorical Columns in Adult Income Dataset:", categorical\_adult)

**Program 3**

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

**Observation:**





**Code:**

import numpy as np

# Given data

# x: Week numbers

# y: Sales in thousands

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

y = np.array([2, 4, 5, 9])

# Construct the design matrix X by adding a column of ones (for the intercept)

X = np.column\_stack((np.ones(x.shape[0]), x))

# Compute the coefficients using the formula: beta = (X^T X)^(-1) X^T y

XtX = X.T.dot(X)            # Compute X^T X

XtX\_inv = np.linalg.inv(XtX)  # Invert X^T X

XtY = X.T.dot(y)            # Compute X^T y

beta = XtX\_inv.dot(XtY)     # Compute beta

# Display the computed coefficients

print("Computed coefficients (beta):", beta)

import matplotlib.pyplot as plt

# ... (previous code)

# Generate points for the regression line

x\_line = np.linspace(x.min(), x.max(), 100)  # Create 100 points for a smooth line

y\_line = beta[0] + beta[1] \* x\_line         # Calculate y-values for the line

# Plot the data points

plt.scatter(x, y, label='Data Points', color='blue')

# Plot the regression line

plt.plot(x\_line, y\_line, label='Linear Regression', color='red')

# Customize the plot

plt.xlabel('Week Number (x)')

plt.ylabel('Sales (thousands) (y)')

plt.title('Linear Regression Plot')

plt.legend()  # Show the legend

plt.grid(True)  # Show the grid

# Display the plot

plt.show()

import numpy as np

# Given data

x = np.array([8, 10, 12])

y = np.array([10, 13, 16])

# Construct the design matrix X (adding a column of ones for the intercept)

X = np.column\_stack((np.ones(x.shape[0]), x))

# Compute beta using the normal equation: beta = (X^T X)^(-1) X^T y

XtX = X.T.dot(X)

XtX\_inv = np.linalg.inv(XtX)

XtY = X.T.dot(y)

beta = XtX\_inv.dot(XtY)

# Extract coefficients

beta0, beta1 = beta

print("Intercept (beta0):", beta0)

print("Slope (beta1):", beta1)

# Predict the price for a 20-inch pizza

x\_new = 20

y\_pred = beta0 + beta1 \* x\_new

print("Predicted price for a 20-inch pizza: $", y\_pred)

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Load the data

income\_data = pd.read\_csv("canada\_per\_capita\_income.csv")

# Assumed data columns: 'Year' and 'PerCapitaIncome'

print("Canada Income Data Head:")

print(income\_data.head())

# Prepare feature and target

X\_income = income\_data[["year"]]     # Predictor variable: Year

y\_income = income\_data["per capita income (US$)"]  # Target variable: Per capita income

# Build and train the linear regression model

model\_income = LinearRegression()

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

# Predict per capita income for the year 2020

predicted\_income = model\_income.predict([[2020]])

print("\nPredicted per capita income for Canada in 2020:", predicted\_income[0])

import matplotlib.pyplot as plt

# ... (previous code)

# Predict per capita income for the year 2020

predicted\_income = model\_income.predict([[2020]])

print("\nPredicted per capita income for Canada in 2020:", predicted\_income[0])

# Plot the data points and the regression line

plt.scatter(X\_income, y\_income, color='blue', label='Actual Data')

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

# Plot the prediction for 2020

plt.scatter(2020, predicted\_income[0], color='green', label='Prediction for 2020')

# Customize the plot

plt.xlabel('Year')

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

plt.title('Canada Per Capita Income Prediction')

plt.legend()

plt.grid(True)

# Display the plot

plt.show()

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Load the salary data

salary\_data = pd.read\_csv("salary.csv")

print(income\_data.head())

# Check for null values and handle them (e.g., imputation or removal)

if salary\_data.isnull().values.any():

    print("Null values found in the salary dataset. Handling null values...")

    # Example: Fill null values with the mean of the 'YearsExperience' column

    salary\_data['YearsExperience'].fillna(salary\_data['YearsExperience'].mean(), inplace=True)

    # Other options: Remove rows with nulls or use more sophisticated imputation methods

# Prepare feature and target

X\_salary = salary\_data[["YearsExperience"]]  # Predictor variable: Years of Experience

y\_salary = salary\_data["Salary"]            # Target variable: Salary

# Build and train the linear regression model

model\_salary = LinearRegression()

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

# Predict salary for an employee with 12 years of experience

predicted\_salary = model\_salary.predict([[12]])

print("\nPredicted salary for an employee with 12 years of experience:", predicted\_salary[0])

import matplotlib.pyplot as plt

# Plot the data points and the regression line

plt.scatter(X\_salary, y\_salary, color='blue', label='Actual Data')

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

# Plot the prediction for 12 years of experience

plt.scatter(12, predicted\_salary[0], color='green', label='Prediction for 12 years')

# Customize the plot

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Salary Prediction based on Experience')

plt.legend()

plt.grid(True)

# Display the plot

plt.show()

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

# Read the CSV file (ensure the file is uploaded in your Colab environment)

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

# Rename columns for convenience

df.columns = ['experience', 'test\_score', 'interview\_score', 'salary']

print("Original Data:")

print(df)

# Define a mapping for text to numeric conversion for the 'experience' column

num\_map = {

    "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

}

# Function to convert experience values to numeric

def convert\_experience(x):

    try:

        return float(x)

    except:

        x\_lower = str(x).strip().lower()

        return num\_map.get(x\_lower, np.nan)

# Convert the 'experience' column using the mapping

df['experience'] = df['experience'].apply(convert\_experience)

# Convert 'test\_score', 'interview\_score', and 'salary' to numeric (coerce errors to NaN)

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

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

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

print("\nData After Conversion:")

print(df)

# Fill missing values in numeric columns using the column mean

df['experience'].fillna(df['experience'].mean(), inplace=True)

df['test\_score'].fillna(df['test\_score'].mean(), inplace=True)

df['interview\_score'].fillna(df['interview\_score'].mean(), inplace=True)

print("\nData After Filling Missing Values:")

print(df)

# Prepare the feature matrix X and target vector y

X = df[['experience', 'test\_score', 'interview\_score']]

y = df['salary']

# Build and train the Multiple Linear Regression model

model = LinearRegression()

model.fit(X, y)

# Predict salaries for the given candidate profiles

# Candidate 1: 2 years of experience, 9 test score, 6 interview score

candidate1 = np.array([[2, 9, 6]])

predicted\_salary1 = model.predict(candidate1)

# Candidate 2: 12 years of experience, 10 test score, 10 interview score

candidate2 = np.array([[12, 10, 10]])

predicted\_salary2 = model.predict(candidate2)

import matplotlib.pyplot as plt

# Create the plot

plt.figure(figsize=(10, 6))  # Adjust figure size for better visualization

plt.scatter(df['experience'], y, color='blue', label='Actual Salary') #Plot actual salary against years of experience

# Plot the regression line (this is an approximation since it's a multi-variable regression)

# You can visualize a single feature against the predicted salary

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

# Highlight predictions

plt.scatter(candidate1[0, 0], predicted\_salary1, color='green', label='Candidate 1 Prediction')

plt.scatter(candidate2[0, 0], predicted\_salary2, color='purple', label='Candidate 2 Prediction')

# Add labels and title

plt.xlabel("Years of Experience")

plt.ylabel("Salary")

plt.title("Salary Prediction based on Experience, Test Score, Interview Score")

# Add a legend

plt.legend()

plt.grid(True)

plt.show()

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

# Read the CSV file (ensure the file is uploaded in your Colab environment)

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

# Display the first few rows

print("Original Data:")

print(df.head())

# --- Data Preprocessing ---

# For numeric columns, fill missing values with the column mean

numeric\_cols = ["R&D Spend", "Administration", "Marketing Spend", "Profit"]

for col in numeric\_cols:

    df[col].fillna(df[col].mean(), inplace=True)

# For the categorical column 'State', fill missing values with a placeholder

df["State"].fillna("Unknown", inplace=True)

# Confirm that missing values are handled

print("\nMissing Values After Processing:")

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

# Separate the features and target variable

features = ["R&D Spend", "Administration", "Marketing Spend"] + \

           [col for col in df\_encoded.columns if col.startswith("State\_")]

X = df\_encoded[features]

y = df\_encoded["Profit"]

# --- Prediction for a New Company ---

# Given sample data:

# R&D Spend = 91694.48, Administration = 515841.3, Marketing Spend = 11931.24, State = 'Florida'

new\_company = pd.DataFrame({

    "R&D Spend": [91694.48],

    "Administration": [515841.3],

    "Marketing Spend": [11931.24],

    "State": ["Florida"]

})

# One-hot encode the 'State' column using the same strategy as training data

new\_company\_encoded = pd.get\_dummies(new\_company, columns=["State"], drop\_first=True)

# Align the new data's columns with the training features (fill missing columns with 0)

new\_company\_encoded = new\_company\_encoded.reindex(columns=X.columns, fill\_value=0)

# Predict the profit using the trained model

predicted\_profit = model.predict(new\_company\_encoded)

print("\nPredicted Profit for the New Company: $", round(predicted\_profit[0], 2))

import matplotlib.pyplot as plt

# Assuming 'df\_encoded', 'features', 'X', 'y', 'model', 'new\_company\_encoded', and 'predicted\_profit' are defined from the previous code

# Create the plot

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

# Scatter plot of actual profits vs. R&D Spend

plt.scatter(df\_encoded["R&D Spend"], y, color='blue', label='Actual Profit')

# Plot the regression line (approximation for visualization)

plt.plot(df\_encoded["R&D Spend"], model.predict(X), color='red', label='Regression Line')

# Highlight the new company's prediction

plt.scatter(new\_company\_encoded["R&D Spend"], predicted\_profit, color='green', label='New Company Prediction')

# Add labels and title

plt.xlabel("R&D Spend")

plt.ylabel("Profit")

plt.title("Profit Prediction based on R&D Spend")

# Add a legend

plt.legend()

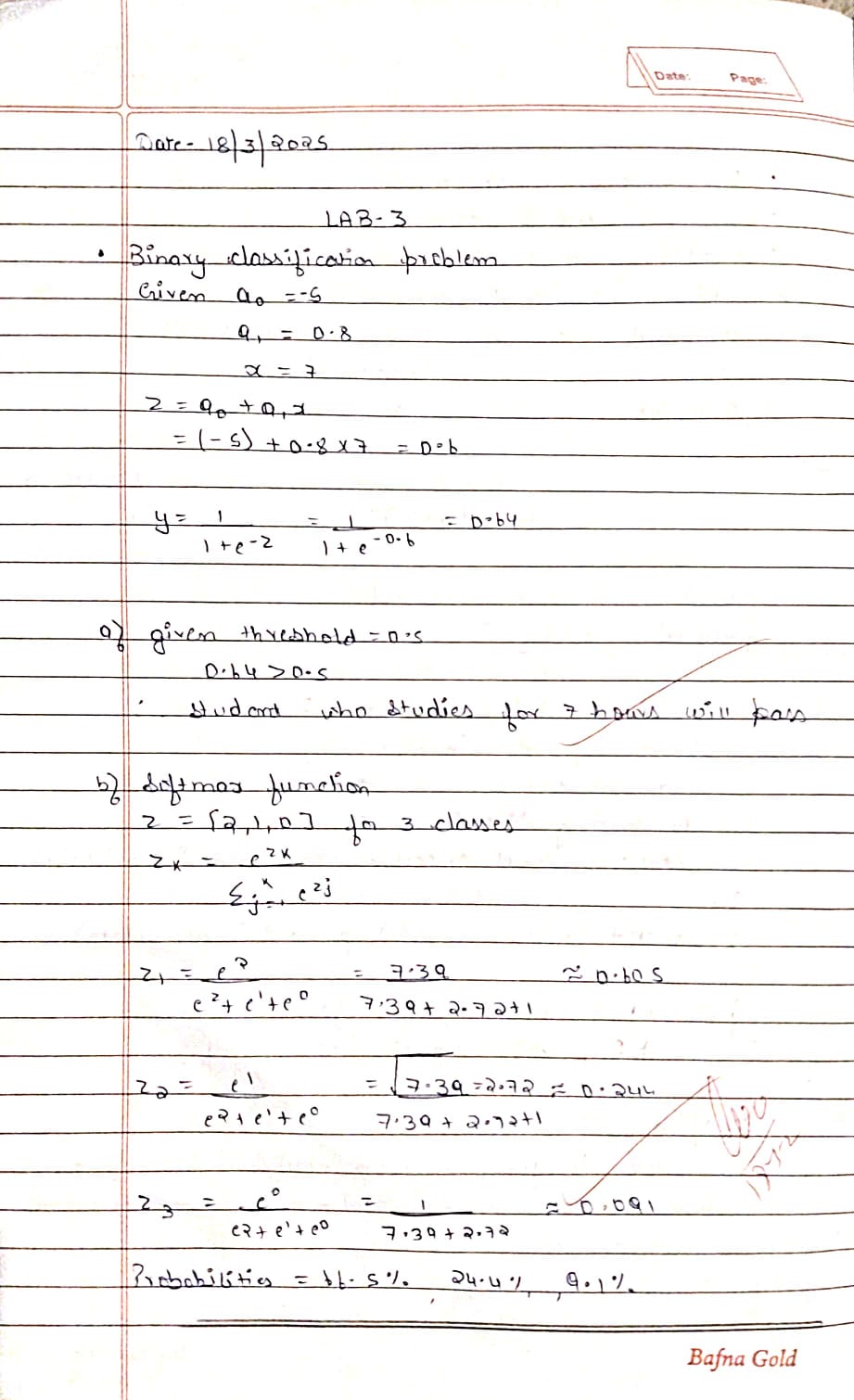
plt.grid(True)

plt.show()

**Program 4**

Build Logistic Regression Model for a given dataset.

**Observation:**



**Code:**

**Hr.csv**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LogisticRegression

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

# Load dataset

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

# Basic Info

print("Dataset Info:")

print(df.info())

print("\nFirst few rows:")

print(df.head())

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

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

sns.barplot(x='Department', y='satisfaction\_level', data=df)

# plt.title('Salary vs Employee Retention')

plt.xlabel('Departments')

plt.ylabel('Satisfaction level')

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# Encode categorical variables (drop\_first avoids dummy variable trap)

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

plt.figure(figsize=(15, 8))

sns.heatmap(df\_encoded.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

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

sns.countplot(x='salary', hue='left', data=df, order=['low', 'medium', 'high'])

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

plt.xlabel('Salary Level')

plt.ylabel('Number of Employees')

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

plt.show()

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

# Calculate the correlation matrix

correlation\_matrix = df\_encoded.corr()

# Extract the correlation with 'left' (employee retention)

correlation\_with\_left = correlation\_matrix['left'].sort\_values(ascending=False)

# Display the correlation

print(correlation\_with\_left)

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

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

# Title and labels

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

plt.xlabel('Department')

plt.ylabel('Number of Employees')

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

plt.xticks(rotation=45)  # Rotate department names for readability

plt.show()

# Step 1: Preprocess the data

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

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

# Load the dataset

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

# Select important features and encode categorical variable

df\_encoded = pd.get\_dummies(df, columns=['salary'], drop\_first=True)  # This encodes salary (low -> low salary column)

# Step 2: Define features (X) and target (y)

X = df\_encoded[['satisfaction\_level', 'time\_spend\_company', 'salary\_low']]  # Using low salary as a feature

y = df\_encoded['left']  # Target variable (whether the employee left or stayed)

# Step 3: Train-test split

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

# Step 4: Build and train the logistic regression model

model = LogisticRegression(max\_iter=1000)

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

# Step 5: Make predictions

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

# Step 6: Evaluate the model

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

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

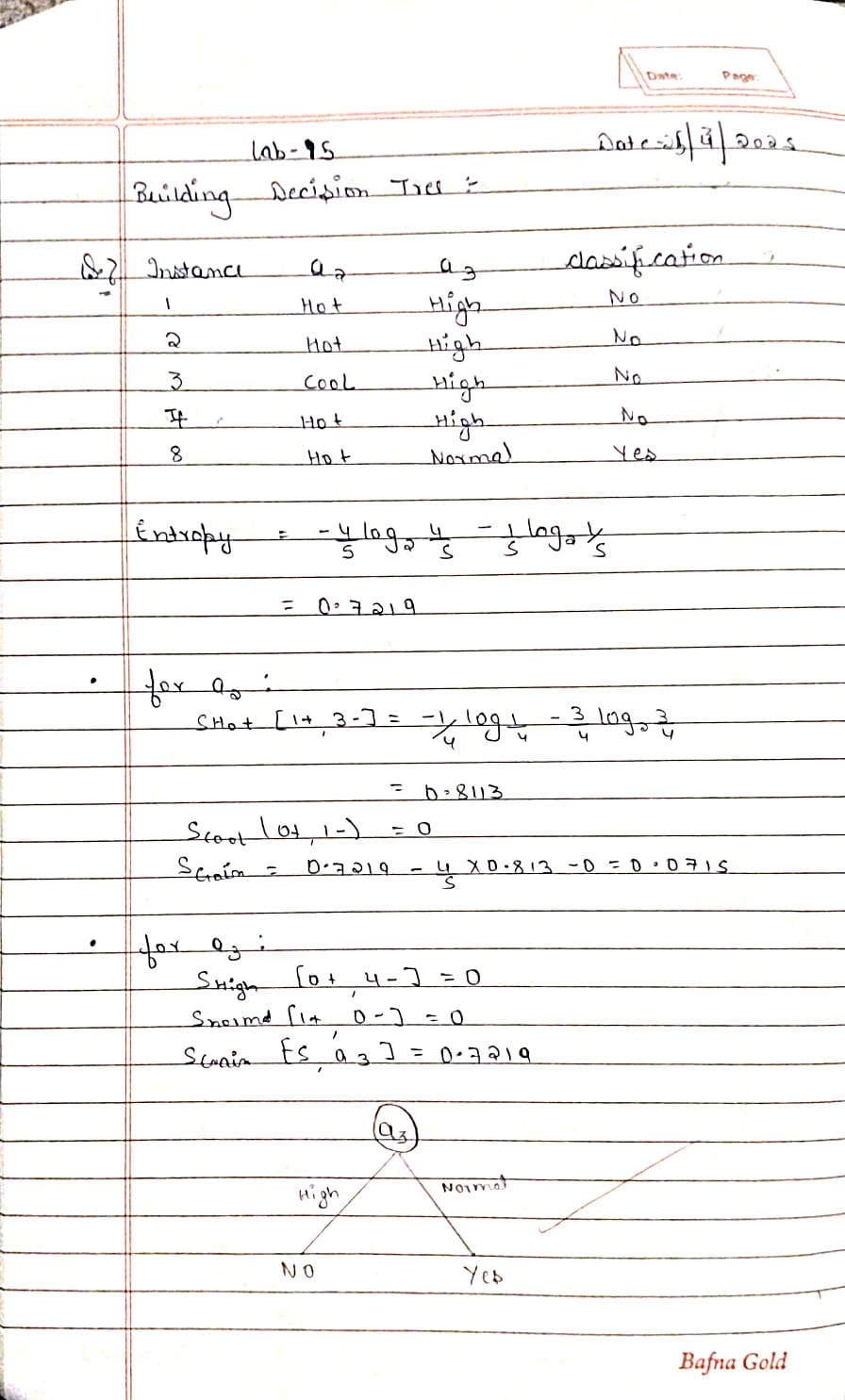
print("\nClassification Report:")

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

**Program 5**

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

**Observation:**



**Code:**

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

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

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

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

data = {

    'a1': [True, True, False, False, False, True, True, True, False, False],

    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],

    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],

    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']

}

df = pd.DataFrame(data)

df.head()

label\_encoders = {}

for column in df.columns:

    le = LabelEncoder()

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

    label\_encoders[column] = le

df.head()

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

y = df['Classification']

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

clf = DecisionTreeClassifier(criterion='entropy')

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

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

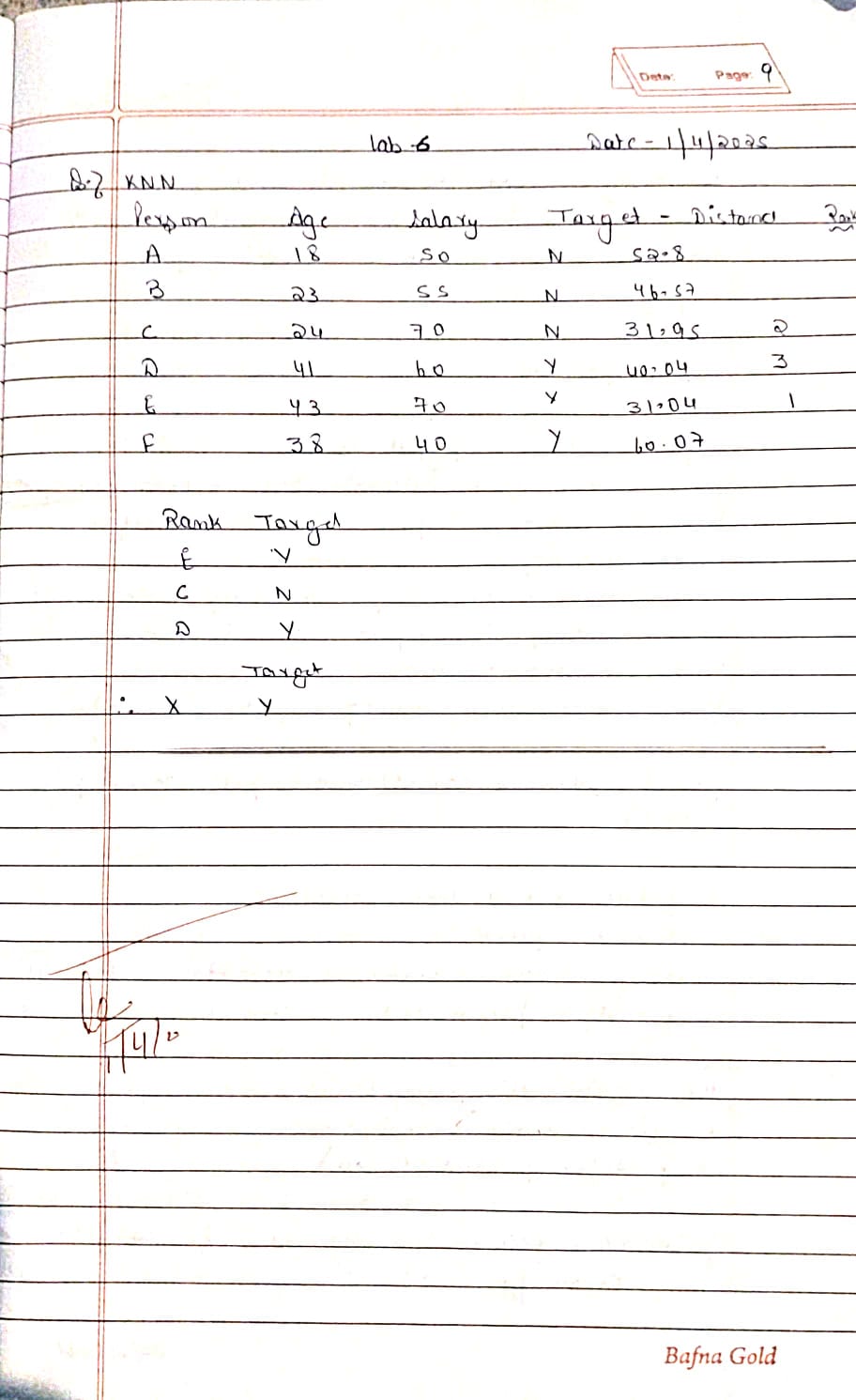
accuracy = accuracy\_score(y\_test, y\_pred)

accuracy

**Program 6**

Build KNN Classification model for a given dataset.

**Observation:**



**Code:**

import pandas as pd

import numpy as np

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

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import OrdinalEncoder, StandardScaler

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

data.head()

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

y = data.iloc[:, -1]

ss = StandardScaler()

X[["Pregnancies"]] = ss.fit\_transform(X[["Pregnancies"]])

X[["Glucose"]] = ss.fit\_transform(X[["Glucose"]])

X[["BloodPressure"]] = ss.fit\_transform(X[["BloodPressure"]])

X[["SkinThickness"]] = ss.fit\_transform(X[["SkinThickness"]])

X[["Insulin"]] = ss.fit\_transform(X[["Insulin"]])

X[["BMI"]] = ss.fit\_transform(X[["BMI"]])

X[["DiabetesPedigreeFunction"]] = ss.fit\_transform(X[["DiabetesPedigreeFunction"]])

X[["Age"]] = ss.fit\_transform(X[["Age"]])

X.head()

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

knn = KNeighborsClassifier()

param\_grid = {"n\_neighbors": [1, 3, 5, 7, 9]}

grid = GridSearchCV(estimator = knn, param\_grid = param\_grid, cv = 5, scoring = "accuracy")

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

grid.best\_params\_

best = grid.best\_estimator\_

best

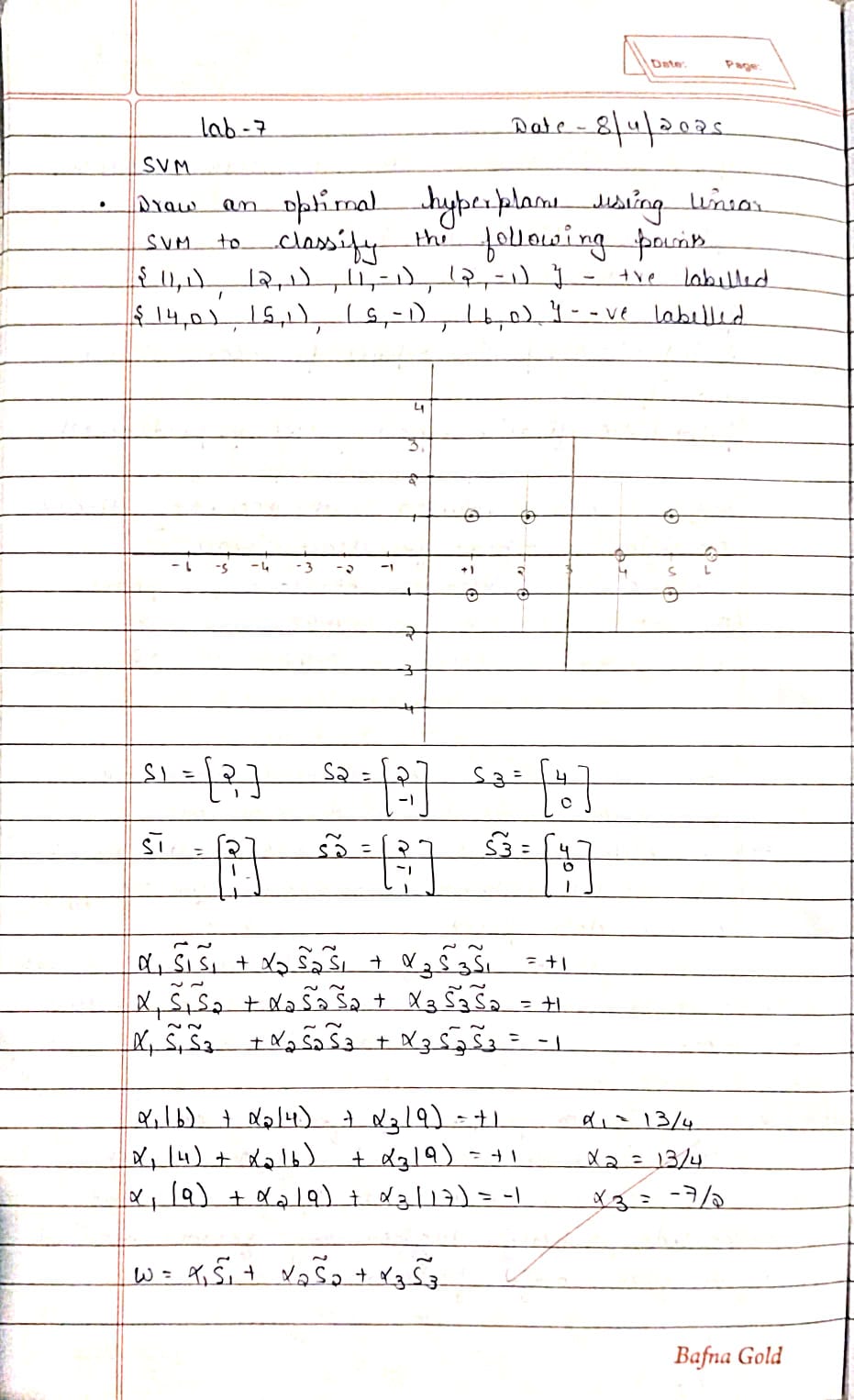
y\_pred = best.predict(X\_test)

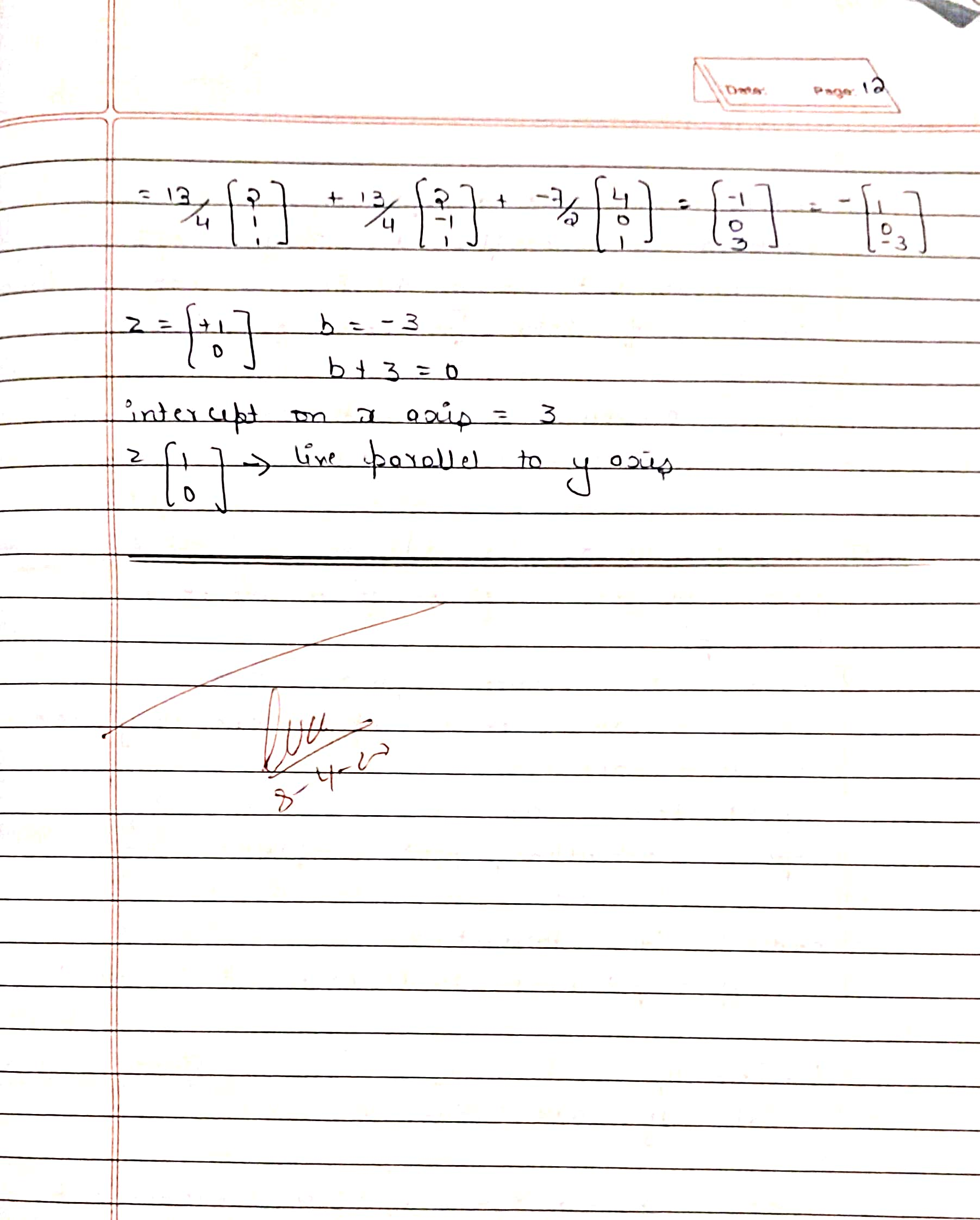
accuracy\_score(y\_test, y\_pred)

**Program 7**

Build Support vector machine model for a given dataset.

**Observation:**





**Code:**

**Iris.csv**

import pandas as pd

from sklearn.datasets import load\_digits

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

from sklearn.svm import SVC

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

from sklearn.preprocessing import OrdinalEncoder

data = pd.read\_csv("iris (1).csv")

data.head()

oe = OrdinalEncoder()

data[["species"]] = oe.fit\_transform(data[["species"]])

data.head()

y = data.iloc[:, -1]

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

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

rbf\_model = SVC(kernel='rbf')

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

rbf\_model.score(X\_test,y\_test)

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

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

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

linear\_model = SVC(kernel='linear')

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

linear\_model.score(X\_test,y\_test)

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

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

**Digits.csv**

import pandas as pd

from sklearn.datasets import load\_digits

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

from sklearn.svm import SVC

digits = load\_digits()

digits.target

dir(digits)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('target',axis='columns'), df.target, test\_size=0.3)

rbf\_model = SVC(kernel='rbf')

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

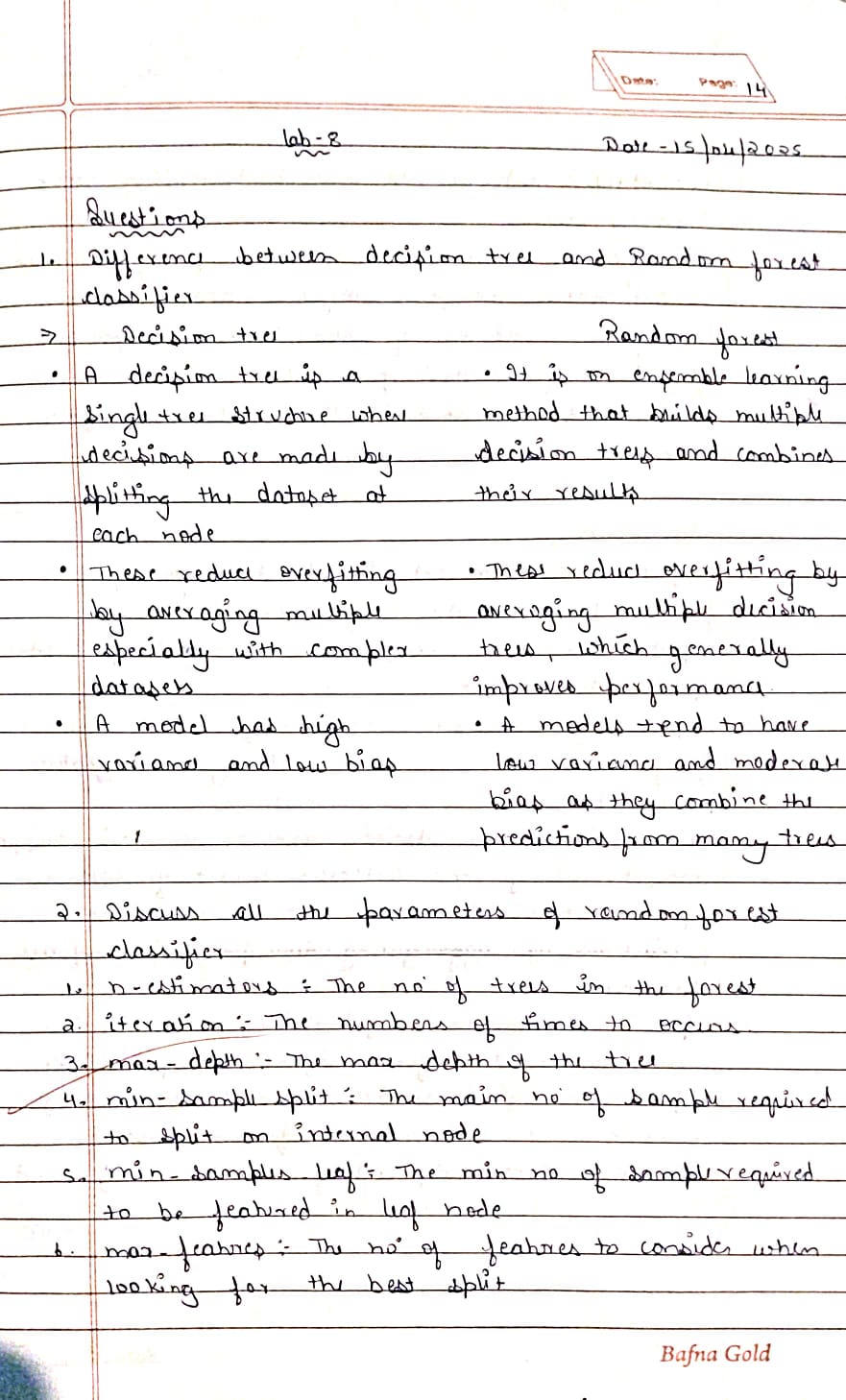
linear\_model = SVC(kernel='linear')

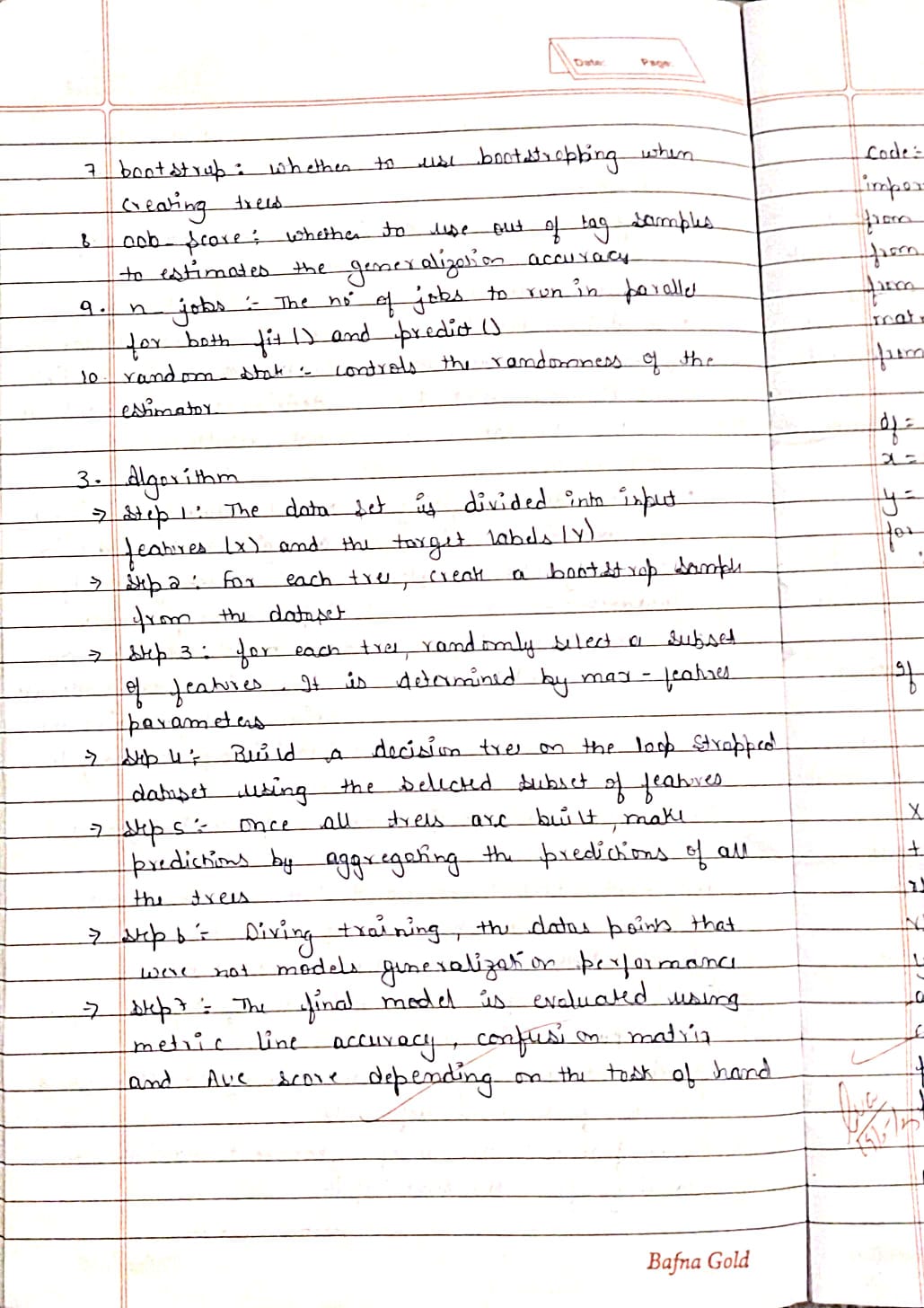
linear\_model.fit(X\_train,y\_train)

**Program 8**

Implement Random forest ensemble method on a given dataset.

**Observation:**





**Code:**

import numpy as np

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

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

import matplotlib.pyplot as plt

from sklearn.preprocessing import OrdinalEncoder

data = pd.read\_csv("iris (2).csv")

data.head()

oe = OrdinalEncoder()

data[["species"]] = oe.fit\_transform(data[["species"]])

data.head()

y = data.iloc[:, -1]

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

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

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

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

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

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

accuracy

n\_estimators\_list = [10, 50, 100, 200, 500, 1000]

accuracies = []

for n in n\_estimators\_list:

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

    rf.fit(X\_train, y\_train)

    y\_pred = rf.predict(X\_test)

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

    accuracies.append(accuracy)

    print(f"Accuracy with n\_estimators={n}: {accuracy:.4f}")

plt.plot(n\_estimators\_list, accuracies, marker='o')

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

plt.ylabel('Accuracy')

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

plt.show()

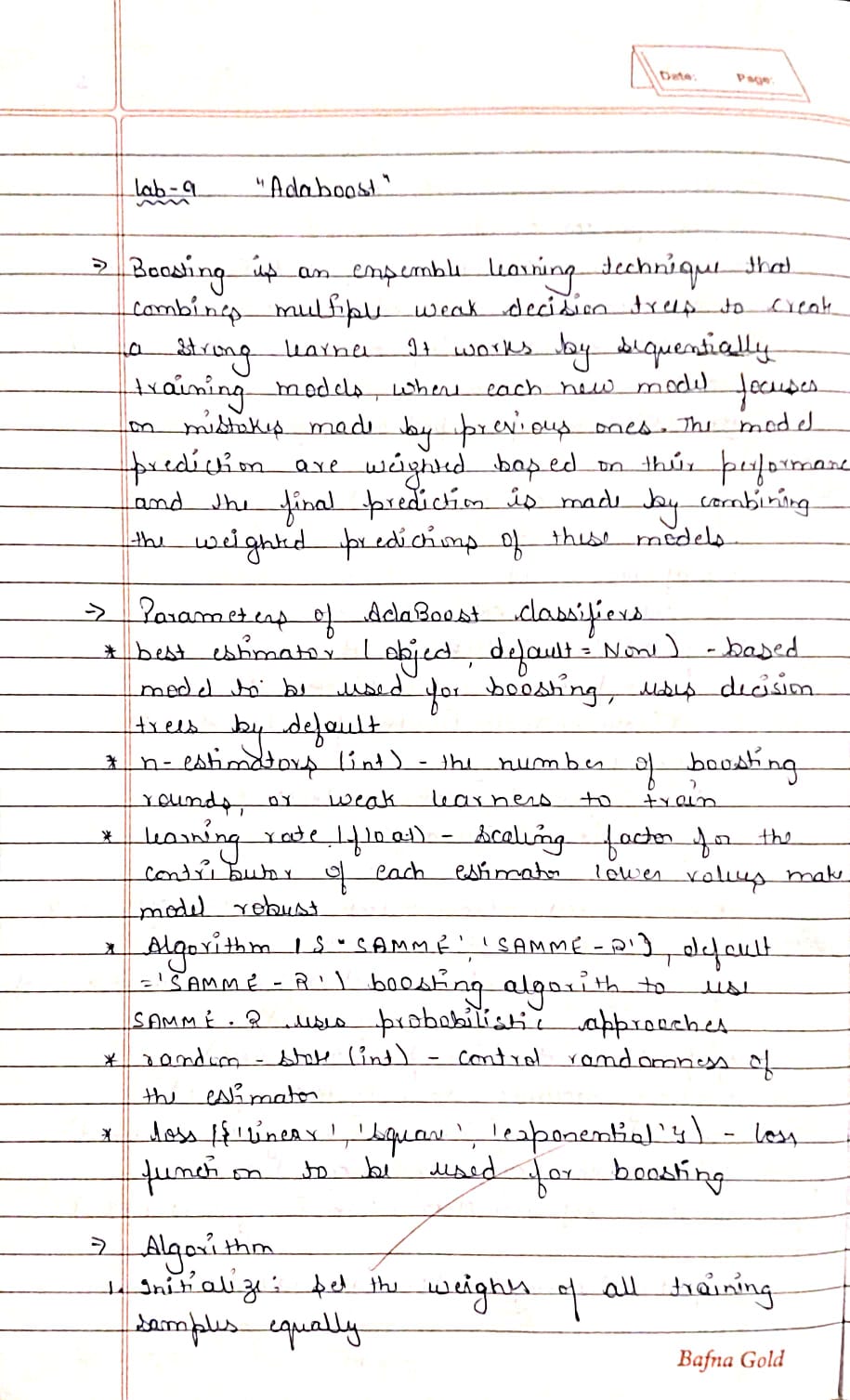
optimal\_n\_estimators = n\_estimators\_list[np.argmax(accuracies)]

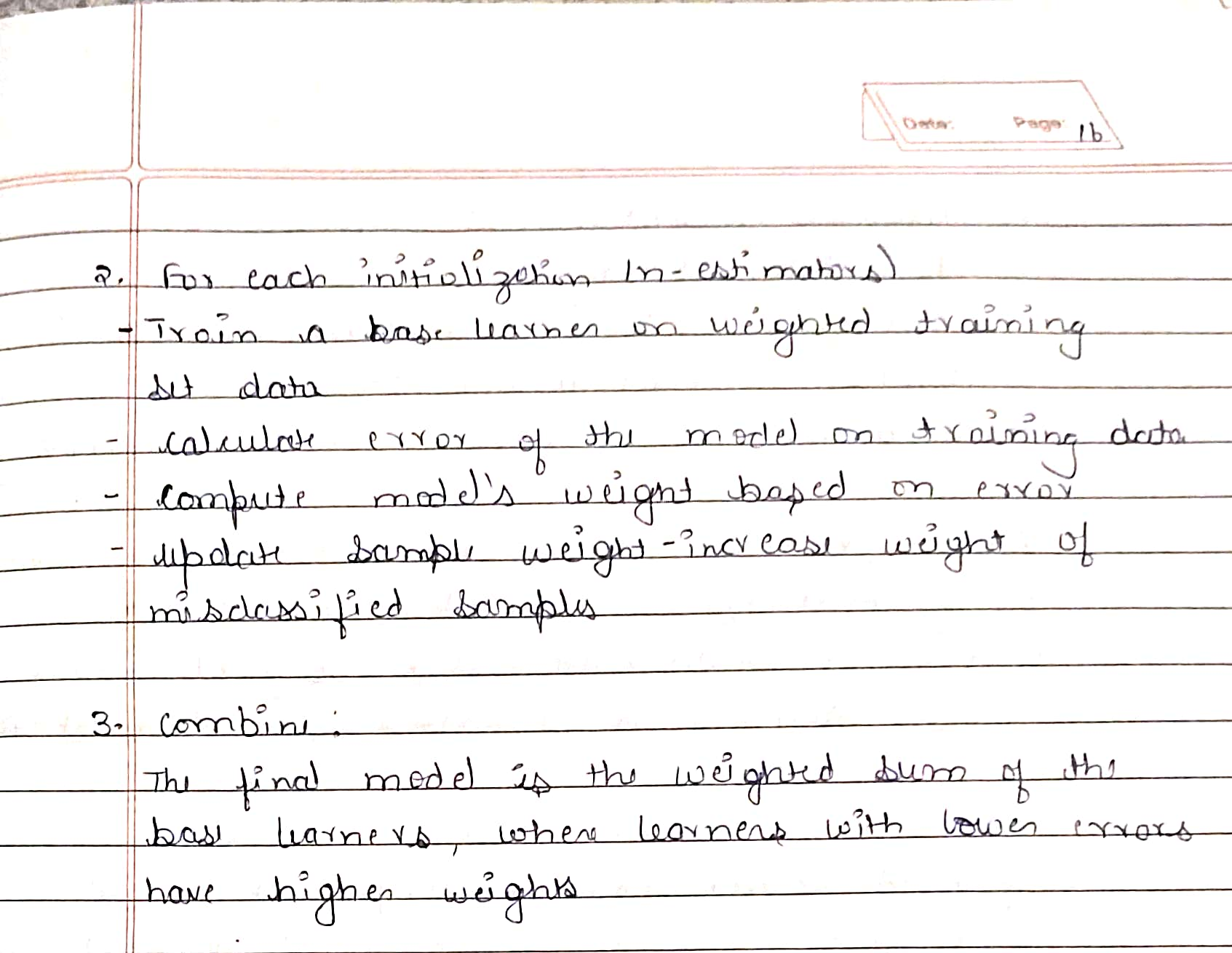
print(f"Best accuracy is obtained with n\_estimators={optimal\_n\_estimators}")

**Program 9**

Implement Boosting ensemble method on a given dataset.

**Observation:**





**Code:**

import numpy as np

import pandas as pd

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

from sklearn.ensemble import AdaBoostClassifier

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

import matplotlib.pyplot as plt

from sklearn.preprocessing import OrdinalEncoder

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

data.head()

y = data.iloc[:, -1]

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

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

rf = AdaBoostClassifier(n\_estimators=1000, random\_state=42)

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

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

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

accuracy

n\_estimators\_list = [10, 50, 100, 200, 500, 1000]

accuracies = []

for n in n\_estimators\_list:

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

    rf.fit(X\_train, y\_train)

    y\_pred = rf.predict(X\_test)

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

    accuracies.append(accuracy)

    print(f"Accuracy with n\_estimators={n}: {accuracy:.4f}")

plt.plot(n\_estimators\_list, accuracies, marker='o')

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

plt.ylabel('Accuracy')

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

plt.show()

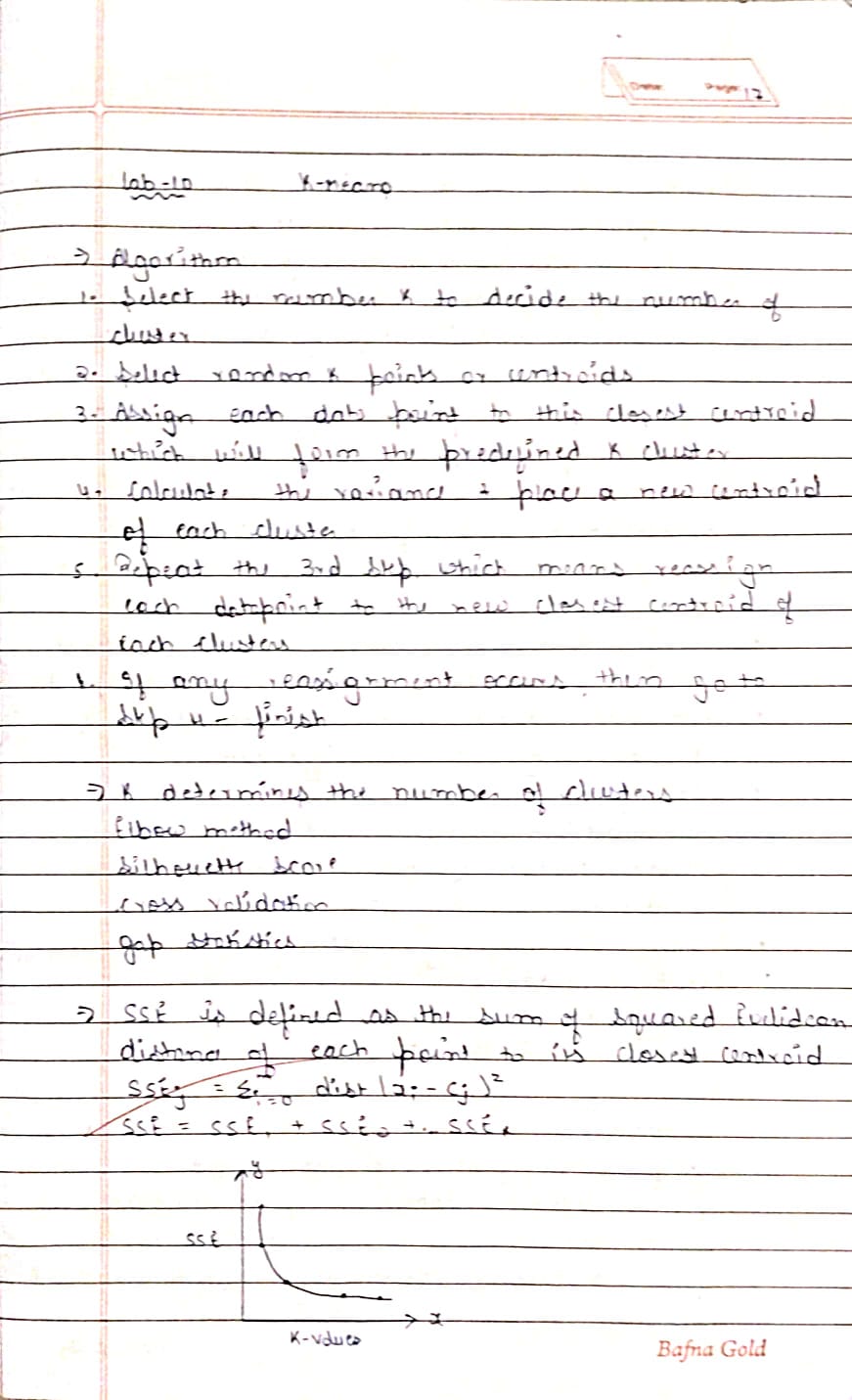
optimal\_n\_estimators = n\_estimators\_list[np.argmax(accuracies)]

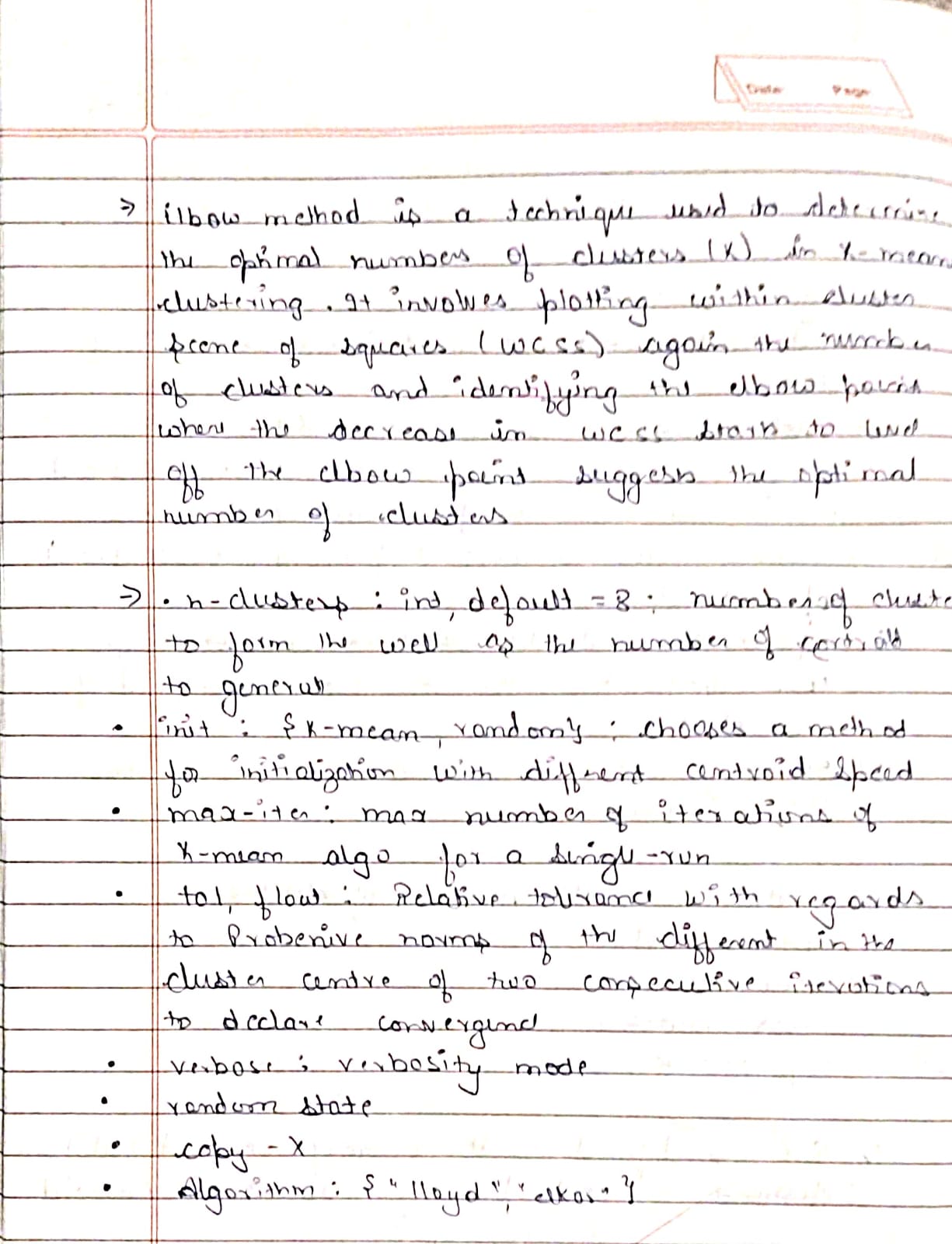
print(f"Best accuracy is obtained with n\_estimators={optimal\_n\_estimators}")

**Program 10**

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

**Observation:**





**==Code:**

# Import required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Load dataset

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

# Use only petal\_length and petal\_width

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

# Scale the features (helps with KMeans)

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:

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

    kmeans.fit(X\_scaled)

    inertia.append(kmeans.inertia\_)

# Plot the elbow curve

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

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

plt.title("Elbow Method for Optimal k")

plt.xlabel("Number of clusters (k)")

plt.ylabel("Inertia (Within-Cluster Sum of Squares)")

plt.grid(True)

plt.show()

# Find optimal k using "elbow" (visually)

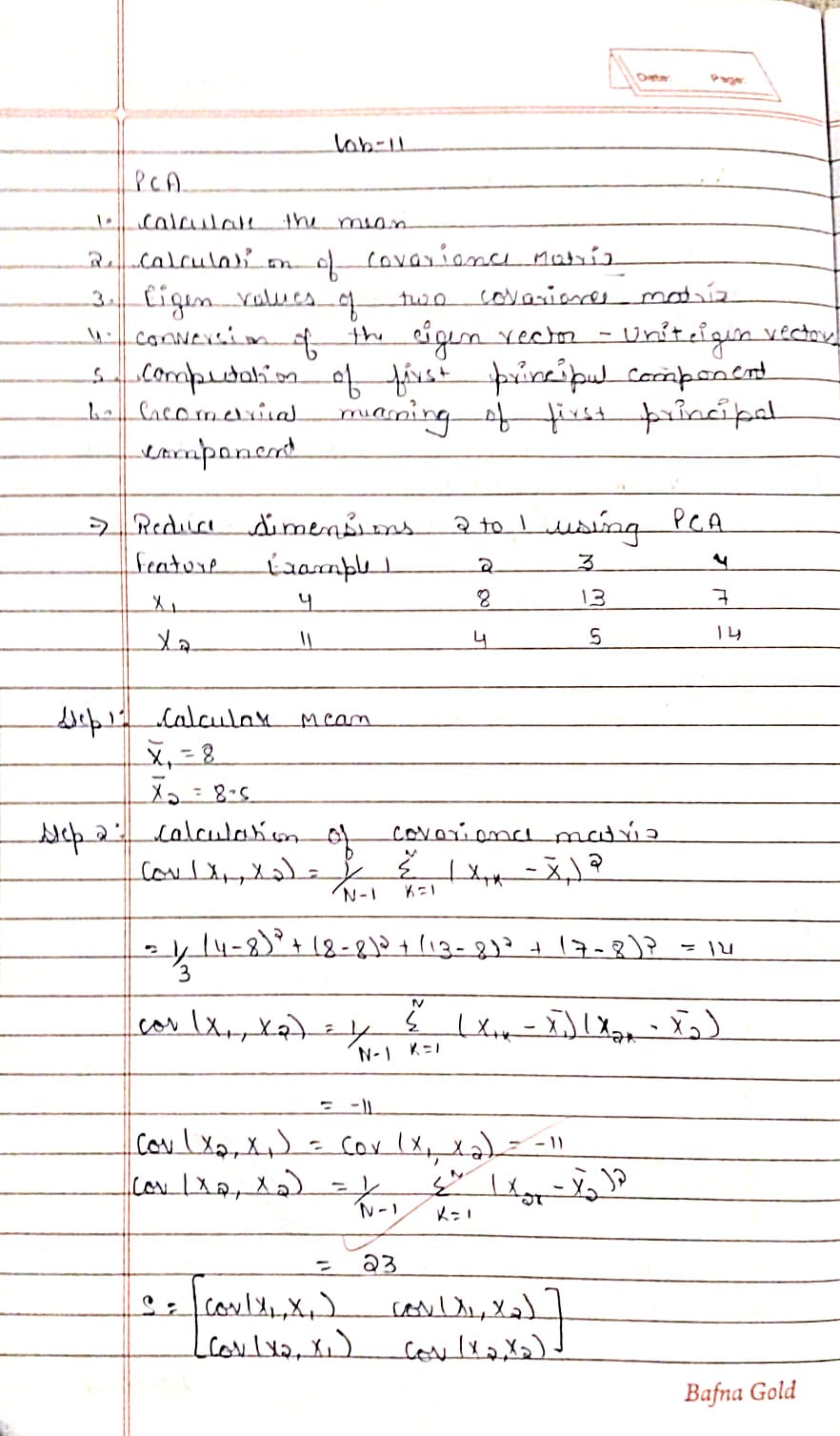
optimal\_k = 3  # for IRIS, elbow is usually at 3

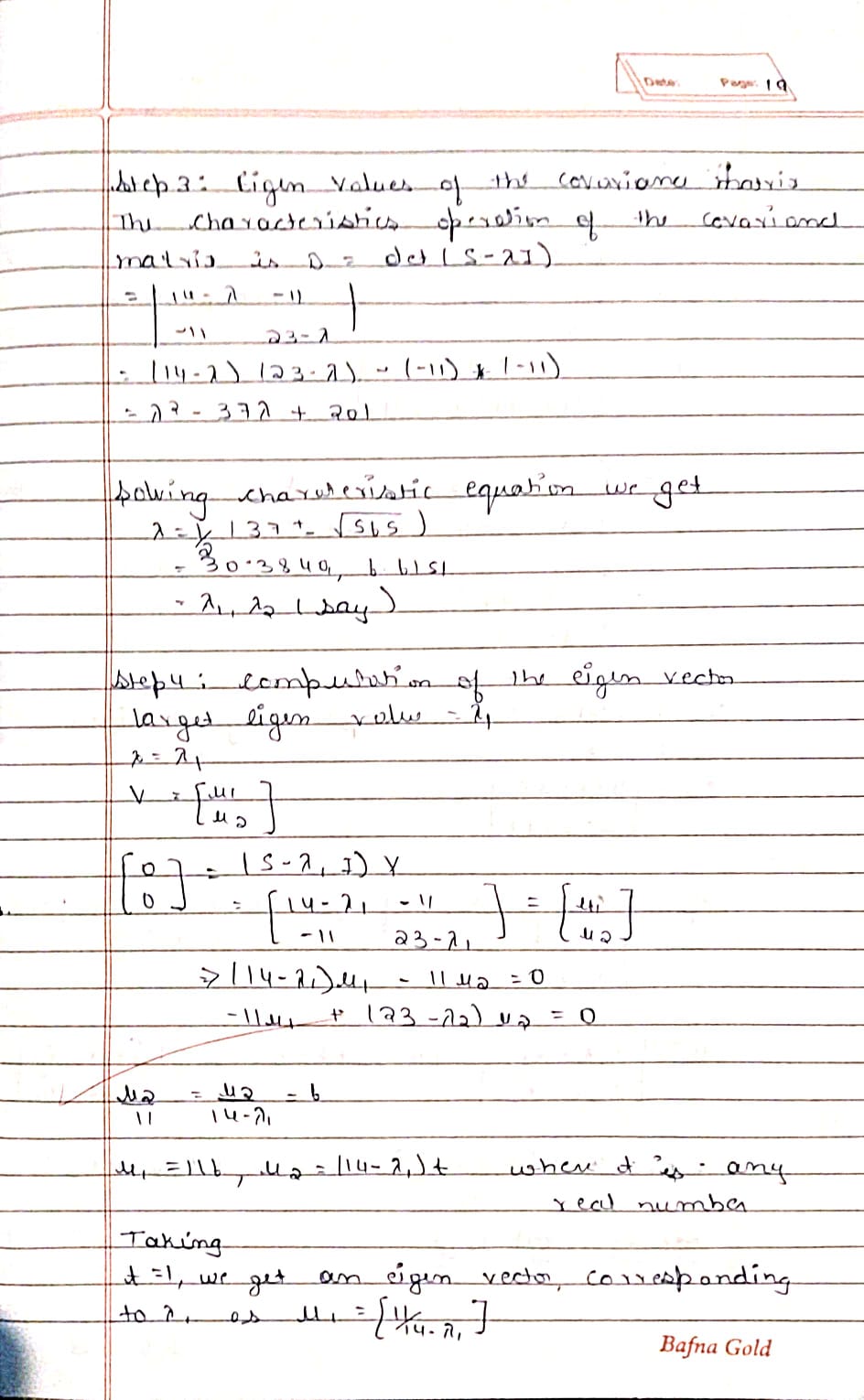
print(f"Optimal number of clusters (k): {optimal\_k}")

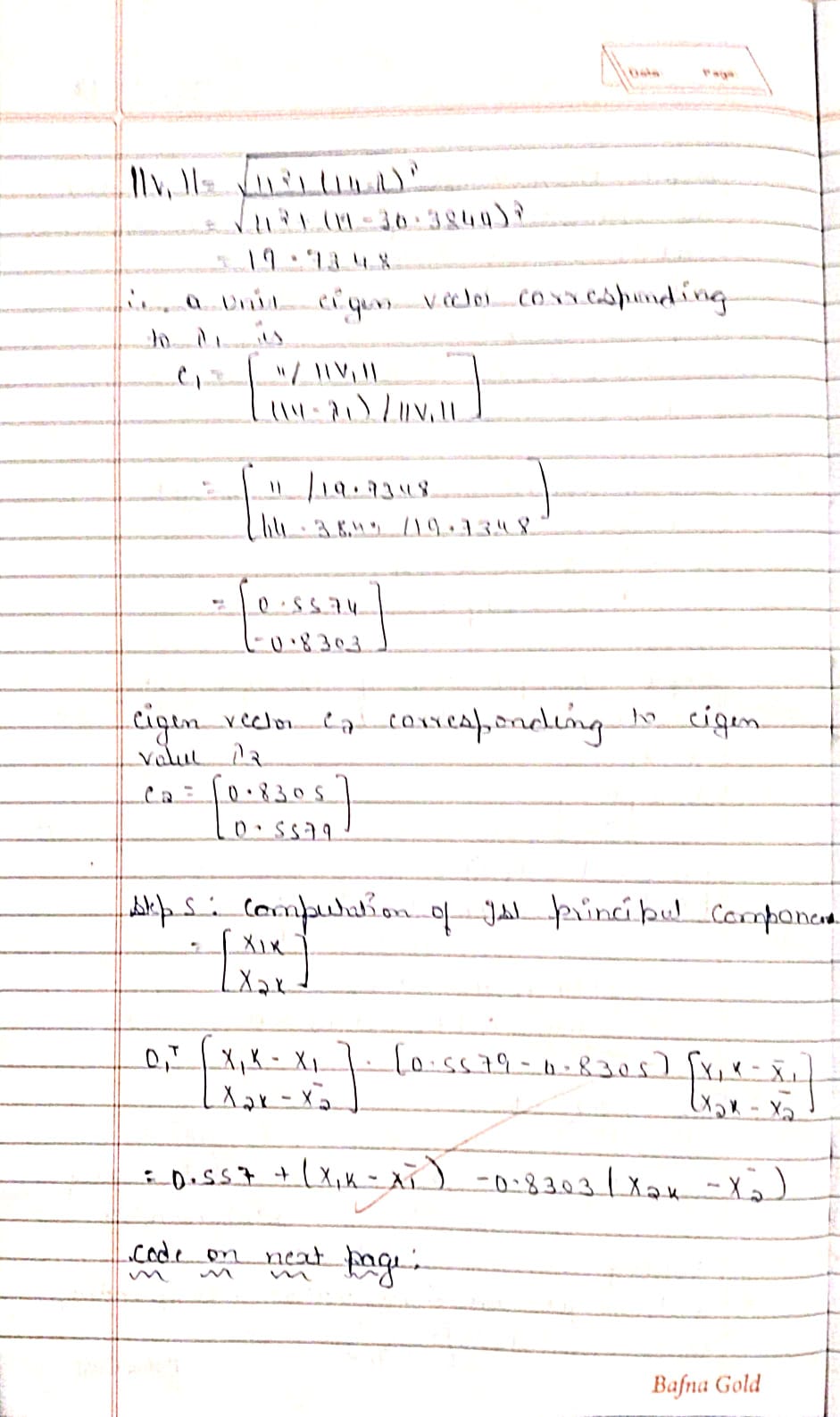
**Program 11**

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

**Observation:**







**Code:**

# Importing necessary libraries

import pandas as pd

import numpy as np

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.metrics import accuracy\_score

from sklearn.decomposition import PCA

# Load dataset

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

# Separate features and target

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

y = df["HeartDisease"]

# Identify categorical columns

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

# Label Encode binary categorical columns

label\_enc = LabelEncoder()

for col in cat\_cols:

    if X[col].nunique() == 2:

        X[col] = label\_enc.fit\_transform(X[col])

        cat\_cols.remove(col)

# One-hot encode remaining categorical columns

X = pd.get\_dummies(X, columns=cat\_cols)

# Train-test split

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

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

# Initialize models

models = {

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

    "SVM": SVC(),

    "Random Forest": RandomForestClassifier()

}

# Store accuracy scores

accuracy\_before\_pca = {}

accuracy\_after\_pca = {}

# Training and evaluating models before PCA

for name, model in models.items():

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

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

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

    accuracy\_before\_pca[name] = acc

# Apply PCA

pca = PCA(n\_components=0.95)  # retain 95% variance

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

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

# Training and evaluating models after PCA

for name, model in models.items():

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

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

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

    accuracy\_after\_pca[name] = acc

# Print accuracy comparison

print("Model Accuracy Comparison (Before vs After PCA):")

print(f"{'Model':<20} {'Before PCA':<15} {'After PCA':<15}")

for name in models.keys():

    print(f"{name:<20} {accuracy\_before\_pca[name]:<15.4f} {accuracy\_after\_pca[name]:<15.4f}")