# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT**
**on**

# Machine Learning (23CS6PCMAL)

***Submitted by***

**AKSHARA SINGA (1BM22CS029)**

***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 **Akshara Singa(1BM22CS029),** 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.

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

# Index



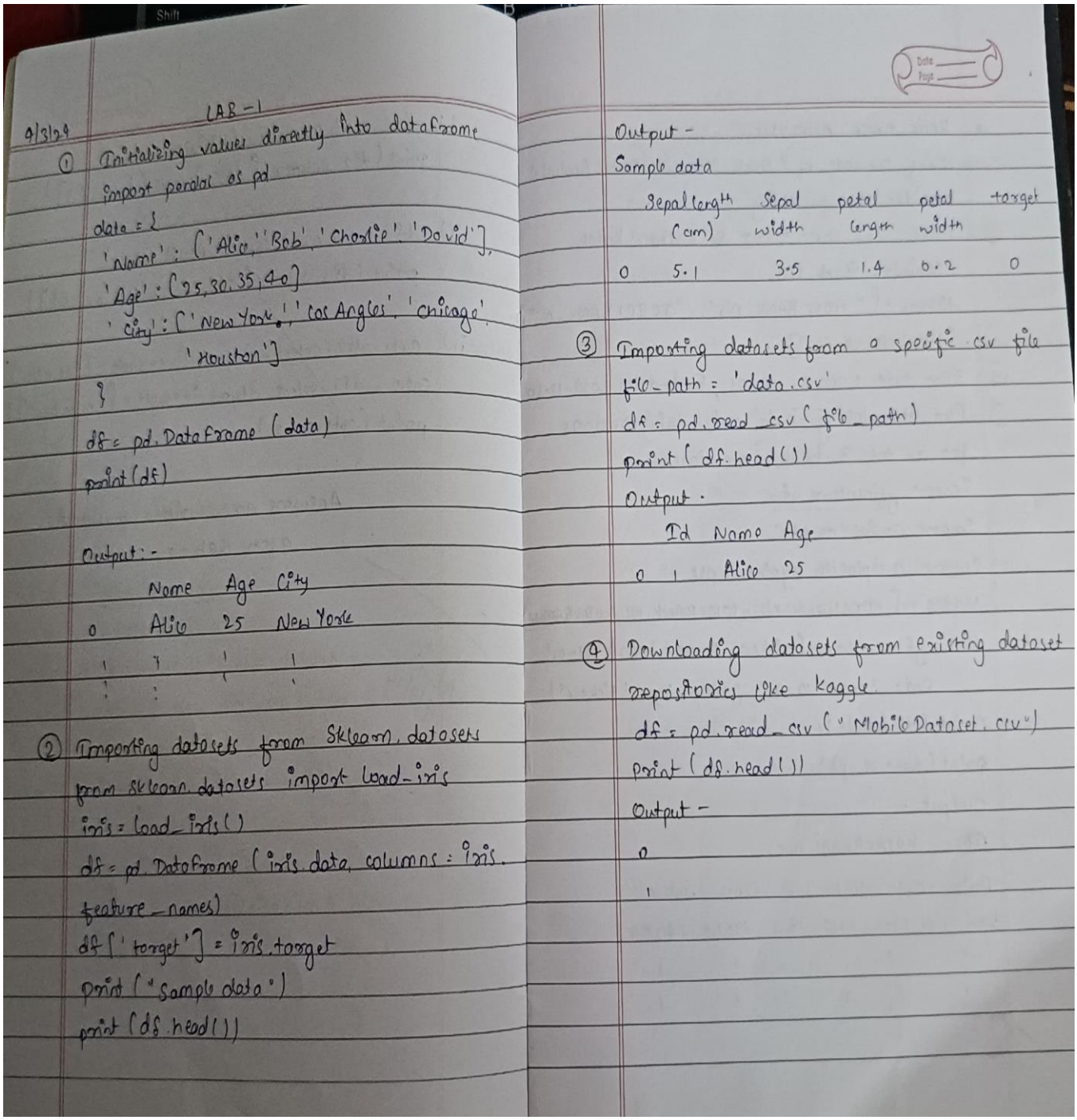
Github Link:

https://github.com/Singa-akshara/6A-ML-LAB-Batch2

## Program 1

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

**Screenshot:**



9/3/24 LAB - 1

① Initializing values directly into dataframe

```
import pandas as pd
data = {
'Name': ['Alice', 'Bob', 'Charlie', 'David'],
'Age': [25, 30, 35, 40]
'City': ['New York', 'Los Angles', 'Chicago',
        'Houston']
}
df = pd.DataFrame(data)
print(df)
```

Output:-

| | Name | Age | City |
|---|---|---|---|
| 0 | Alice | 25 | New York |
| : | : | : | : |

② Importing datasets from Sklearn.datasets

```
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())
```

Output -

Sample data

| | Sepal length (cm) | Sepal width | petal length | petal width | target |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |

③ Importing datasets from a specific csv file

```
file_path = 'data.csv'
df = pd.read_csv(file_path)
print(df.head())
```

Output:

| | Id | Name | Age |
|---|---|---|---|
| 0 | 1 | Alice | 25 |

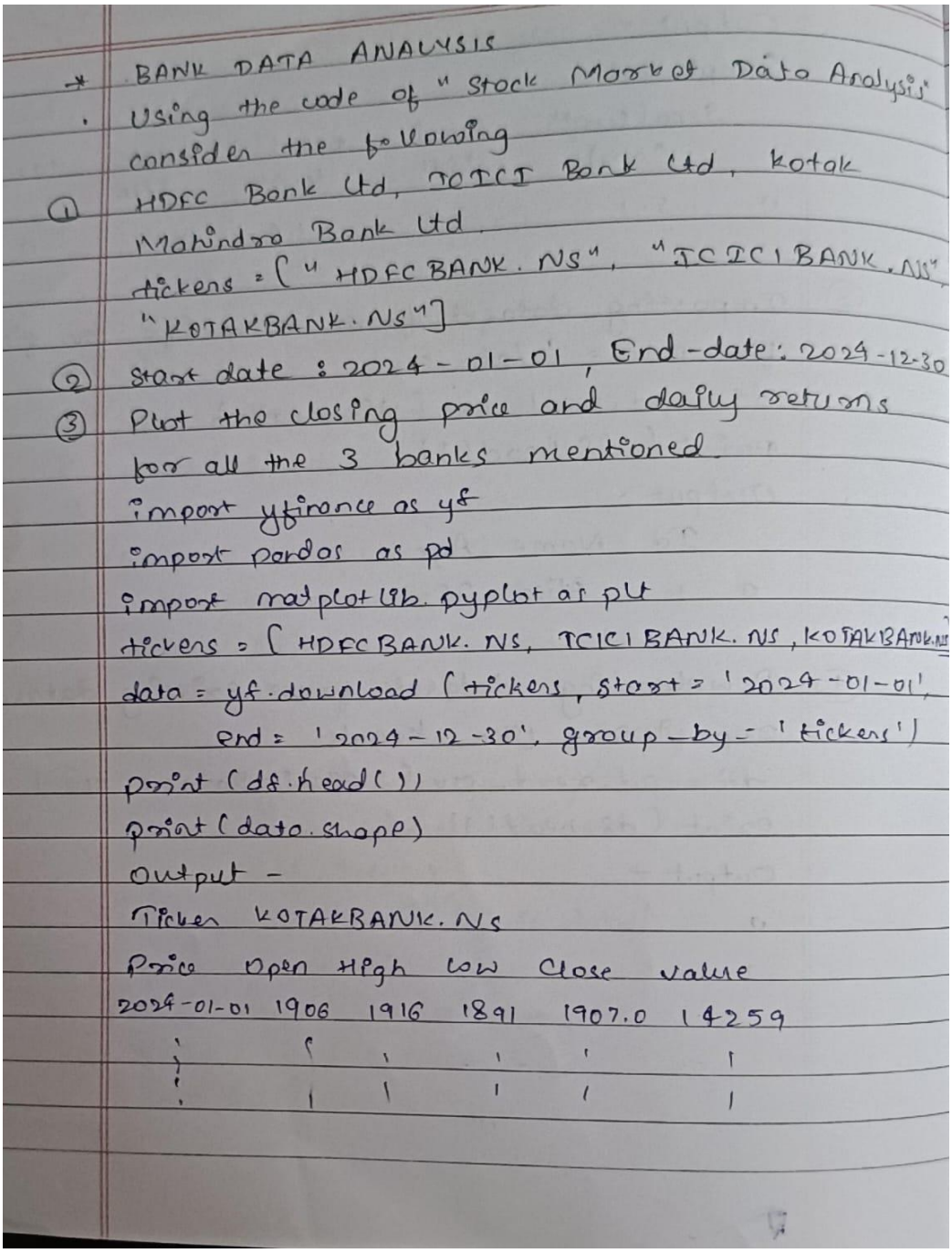
④ Downloading datasets from existing dataset repositories like kaggle

```
df = pd.read_csv("Mobile Dataset.csv")
print(df.head())
```

Output -

0

1




**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:")
```

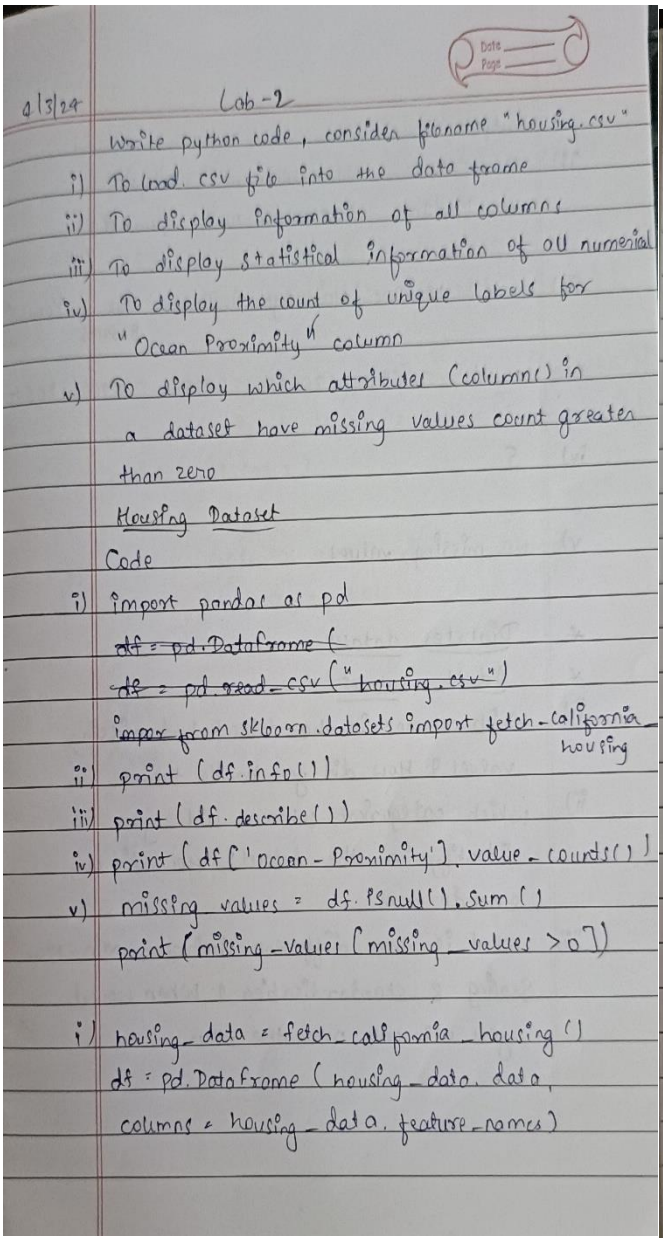
```
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="HDFC Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="HDFCIndustries - 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:**



9/3/24 Lab-2

Write python code, consider filename "housing.csv"

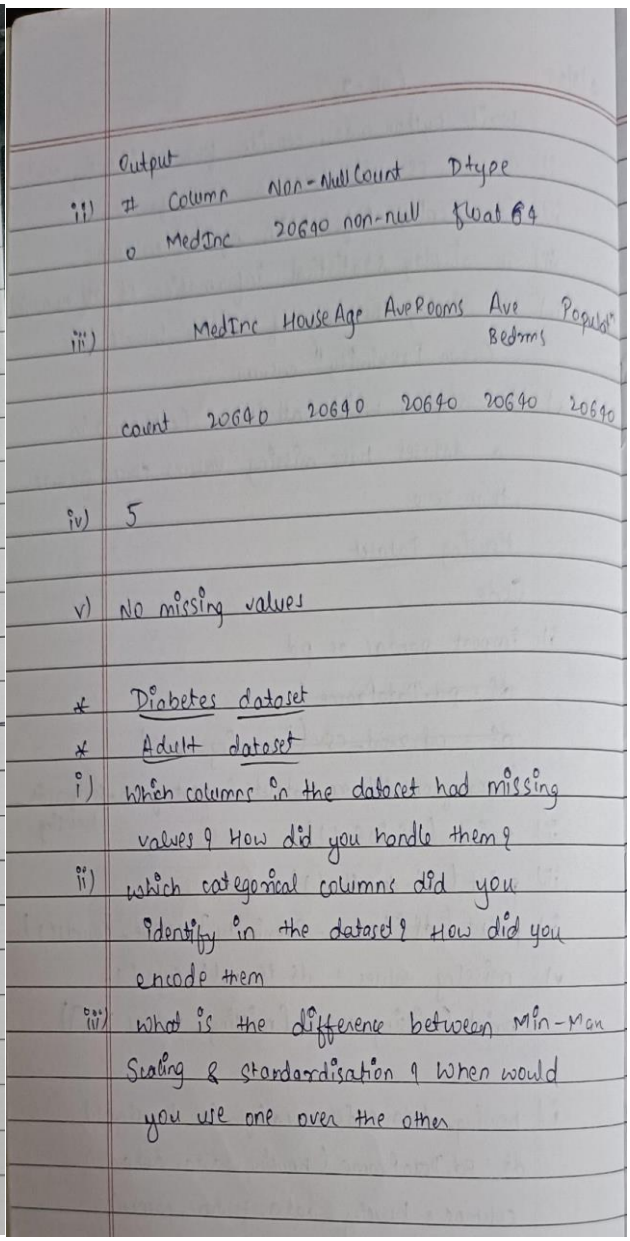
i) To load .csv file into the data frame
ii) To display information of all columns
iii) To display statistical information of all numerical
iv) To display the count of unique labels for "Ocean Proximity" column
v) To display which attributes (columns) in a dataset have missing values count greater than zero

Housing Dataset

Code

i) import pandas as pd
~~df = pd.DataFrame(~~
~~df = pd.read_csv("housing.csv")~~
~~impor from sklearn.datasets import fetch_california_~~ housing
ii) print (df.info())
iii) print (df.describe())
iv) print (df['Ocean_Proximity'].value_counts())
v) missing_values = df.isnull().sum()
print (missing_values[missing_values > 0])

i) housing_data = fetch_california_housing()
df = pd.DataFrame(housing_data.data, columns = housing_data.feature_names)



Output

ii)

| # | Column | Non-Null Count | Dtype |
|---|---|---|---|
| 0 | MedInc | 20640 non-null | float64 |

iii)

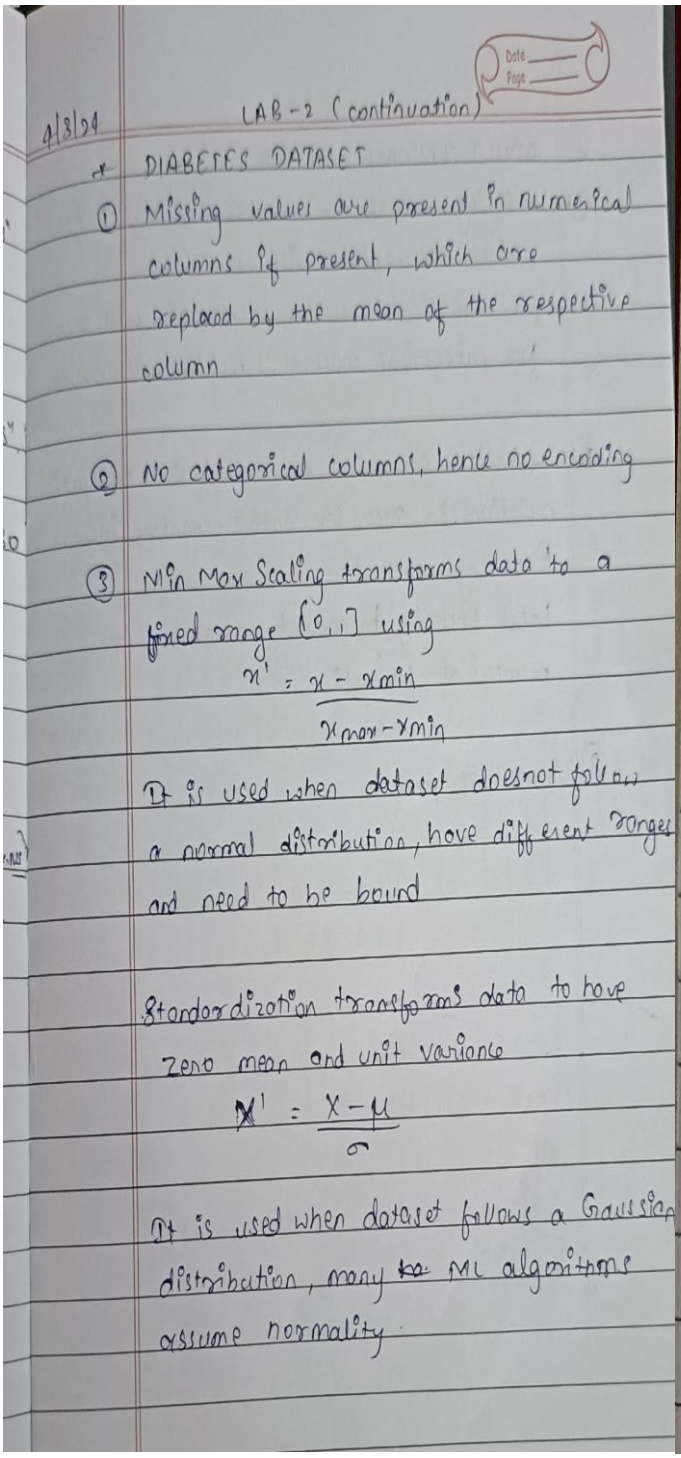
| | MedInc | HouseAge | AveRooms | AveBedrms | Popula |
|---|---|---|---|---|---|
| count | 20640 | 20640 | 20640 | 20640 | 20640 |

iv) 5

v) No missing values

* Diabetes dataset
* Adult dataset

i) Which columns in the dataset had missing values? How did you handle them?
ii) Which categorical columns did you identify in the dataset? How did you encode them
iii) What is the difference between Min-Max Scaling & Standardisation? When would you use one over the other



4/3/24 LAB-2 (continuation)

* DIABETES DATASET

① Missing values are present in numerical columns if present, which are replaced by the mean of the respective column

② No categorical columns, hence no encoding

③ Min Max Scaling transforms data to a fixed range [0,1] using

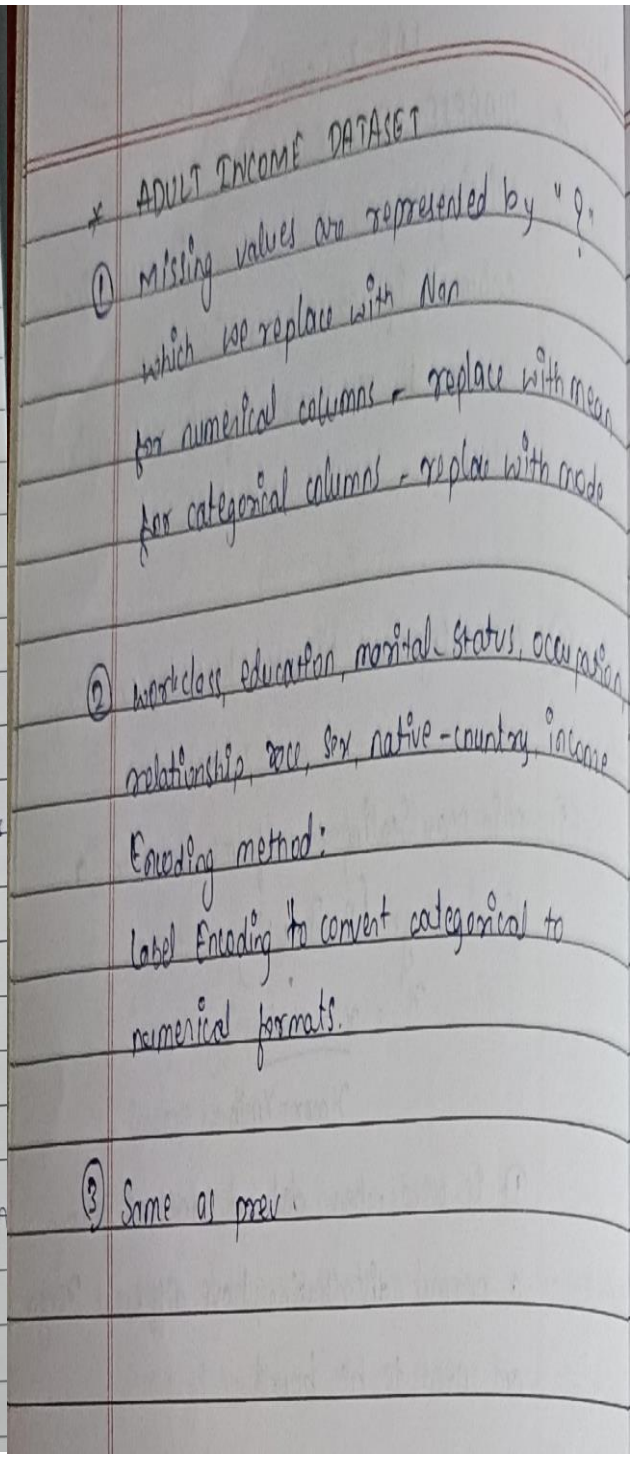
$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

It is used when dataset doesnot follow a normal distribution, have different ranges and need to be bound

Standardization transforms data to have zero mean and unit variance

$$X' = \frac{X - \mu}{\sigma}$$

It is used when dataset follows a Gaussian distribution, many ML algorithms assume normality.



* ADULT INCOME DATASET

① Missing values are represented by "?" which we replace with Nan
for numerical columns - replace with mean
for categorical columns - replace with mode

② workclass, education, marital_status, occupation, relationship, race, sex, native-country, income
Encoding method:
Label Encoding to convert categorical to numerical formats.

③ Same as prev.

**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")
```

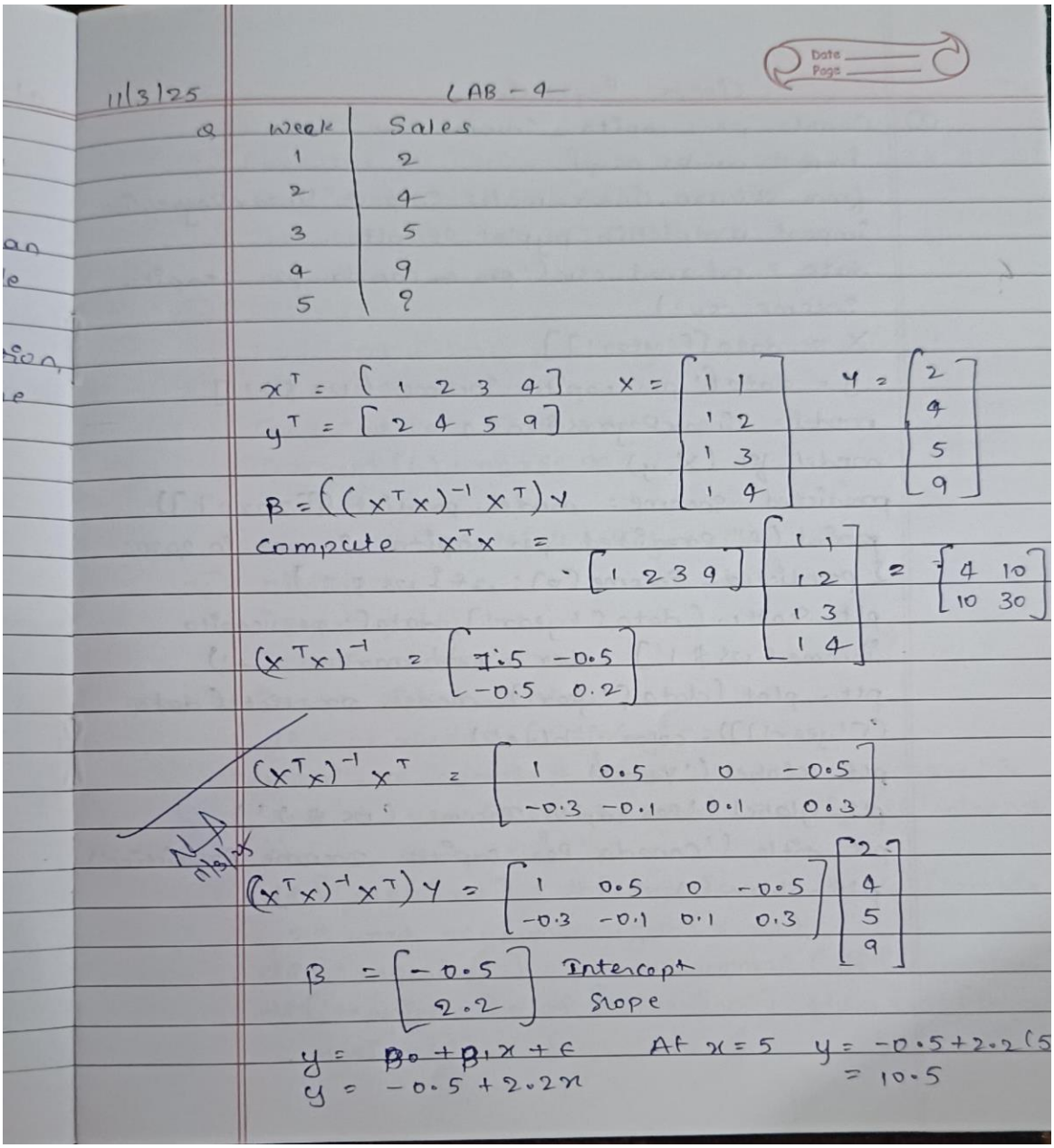
```
df1.head()

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, nplace=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:**



## Linear Regression

① Canada_per_capita_income.csv

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
data = pd.read_csv('Canada_per_capita_income.csv')
X = data[['year']]
y = data['per capita income (US$)']
model = LinearRegression()
model.fit(X,y)
predicted_income = model.predict([[2020]])
print(f"Predicted per capita income in 2020: {predicted_income[0]:.2f} US$")
plt.scatter(data['year'], data['per capita income (US$)'], color='red', marker='+')
plt.plot(data['year'], model.predict(data[['year']]), color='blue')
plt.xlabel('Year')
plt.ylabel('Per capita Income (US$)')
plt.title('Canada Per Capita Income Prediction')
plt.show()
```

Output

Predicted per capita income in 2020: 41288.69 US$

Coefficient (m): 828.47

Intercept (b): -1632210.76

② Salary.csv

Predicted Salary for 12 years of experience:
$ 139049.67 .

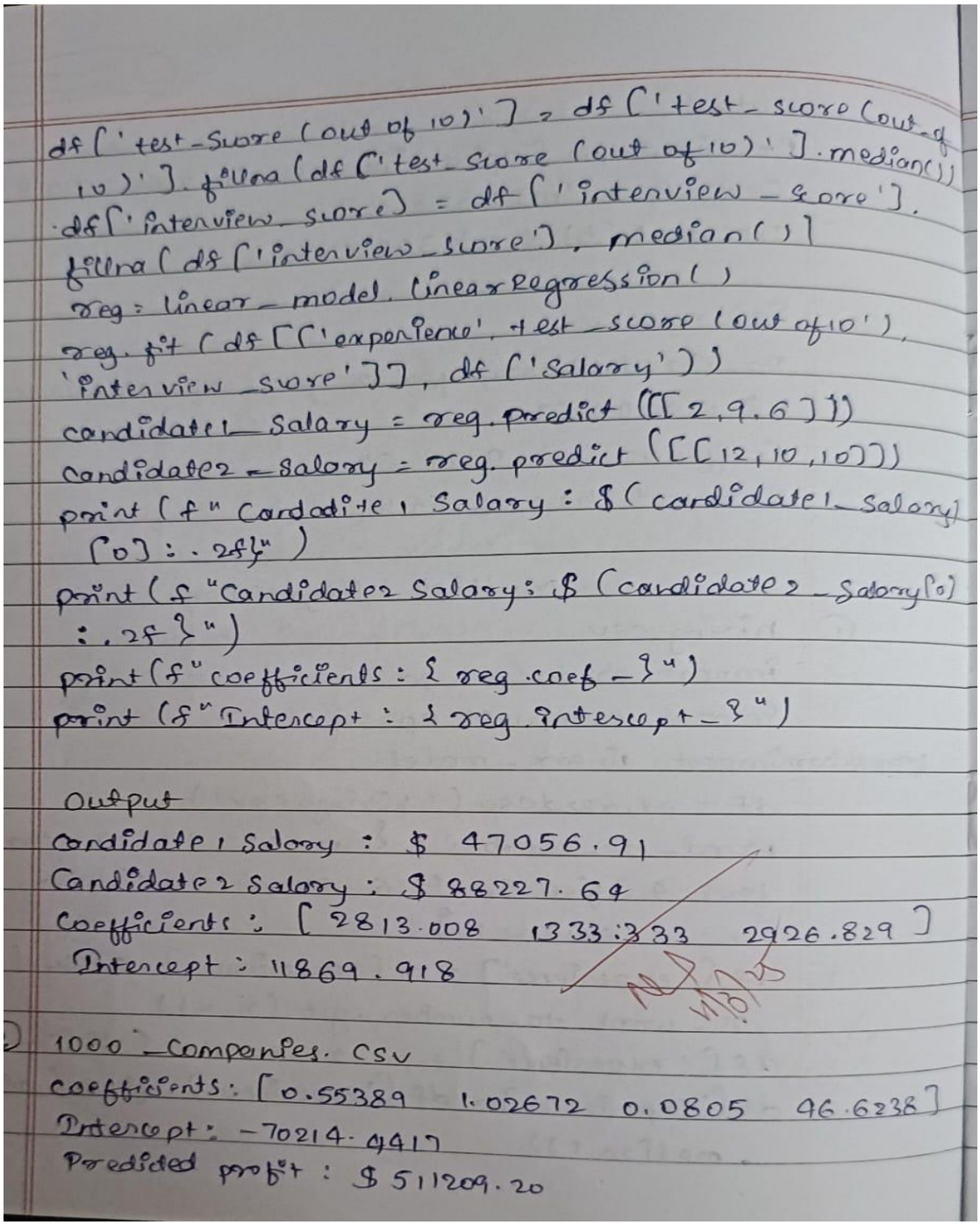
Coefficient (m): 9398.69

Intercept (b): 26265.99

## Multiple Regression

① hiring.csv

```
import pandas as pd
import numpy as np
from sklearn import linear_model
df = pd.read_csv('hiring.csv')
word_to_number = { one:1, two:2, three:3, four:4, five:5, six:6, seven:7, eight:8, nine:9, ten:10, eleven:11 }
df['experience'] = df['experience'].apply(lambda x: word_to_number.get(x,x))
df['experience'] = pd.to_numeric(df['experience'])
df['experience'] = df['experience'].fillna(df['experience'].median())
```




**Code:**

```
from google.colab import files
per_capita_income=files.upload()

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

import pandas as pd
```

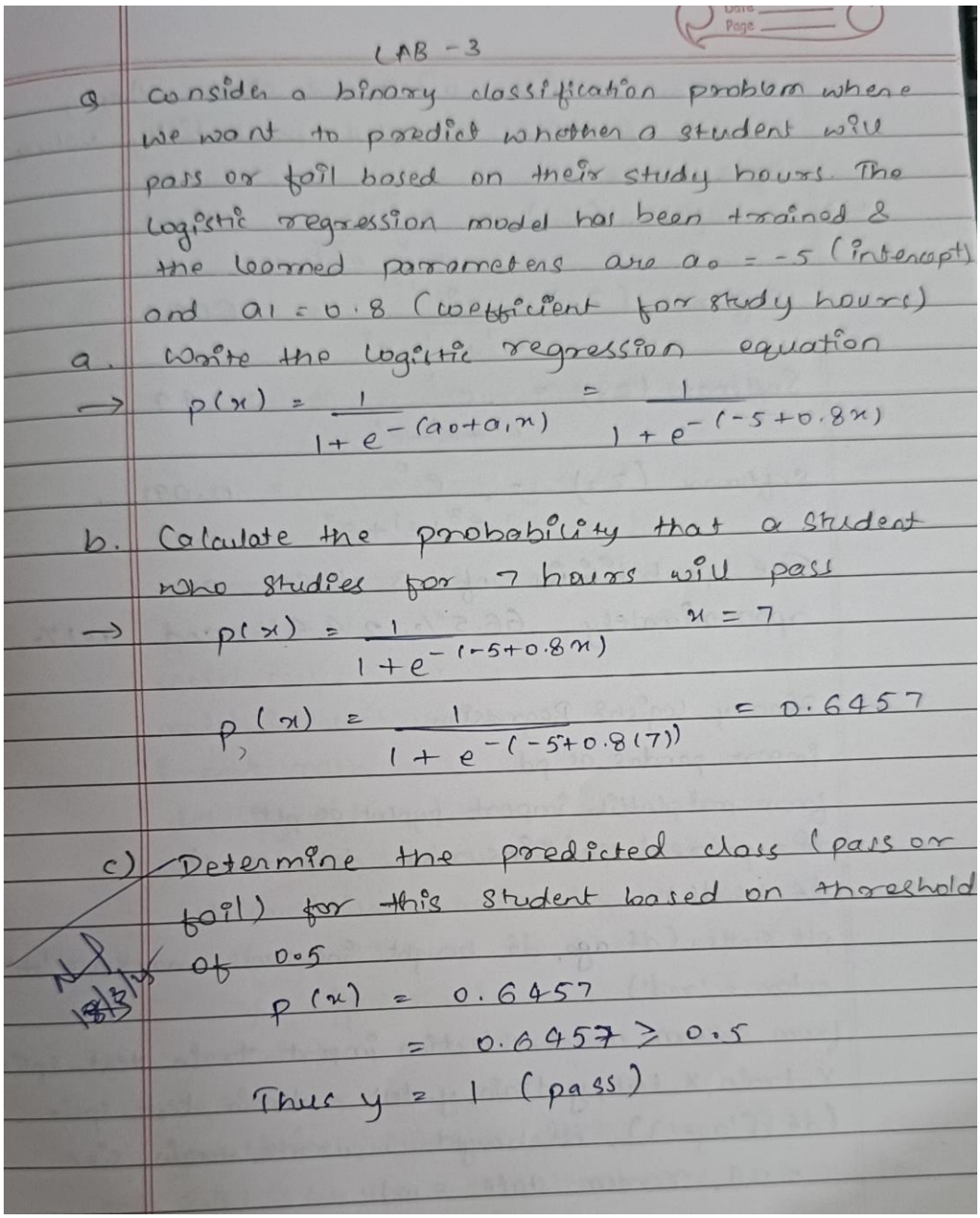
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import
train_test_split  from sklearn.impute
import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder,
OneHotEncoder  from sklearn.preprocessing import
StandardScaler, MinMaxScaler  from scipy import stats
from sklearn import linear_model

df1=pd.read_csv("canada_per_capita_inc
ome.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_
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:**



LAB - 3

Q Consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The logistic regression model has been trained & the learned parameters are $a_0 = -5$ (intercept) and $a_1 = 0.8$ (coefficient for study hours)

a. Write the logistic regression equation

→ $p(x) = \frac{1}{1+e^{-(a_0+a_1 x)}} = \frac{1}{1+e^{-(-5+0.8x)}}$

b. Calculate the probability that a student who studies for 7 hours will pass

→ $p(x) = \frac{1}{1+e^{-(-5+0.8x)}}$ $x = 7$

$p(x) = \frac{1}{1+e^{-(-5+0.8(7))}} = 0.6457$

c) Determine the predicted class (pass or fail) for this student based on threshold of 0.5

$p(x) = 0.6457$

$= 0.6457 \geq 0.5$

Thus $y = 1$ (pass)

2. Consider $z = [2, 1, 0]$ for 3 classes. Apply Softmax function to find probability values of 3 classes

→ $Softmax(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}$

$Softmax(z_1) = \frac{e^2}{e^2+e^1+e^0} = 0.665$

$Softmax(z_2) = \frac{e^1}{e^2+e^1+e^0} = 0.244$

$Softmax(z_3) = \frac{e^0}{e^2+e^1+e^0} = 0.091$

The probabilities of 3 classes are approximately 66.5%, 24.4% and 9.1%.

* Binary Logistic Regression

```
import pandas as pd
from matplotlib import pyplot as plt
df = pd.read_csv("insurance_data.csv")
df.head()
plt.scatter(df.age, df.bought_insurance, marker='+', color='red')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']], df.bought_insurance, train_size=0.9, random_state=10)
X_train.shape
X_test
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_test
y_predicted = model.predict(X_test)
y_predicted
model.score(X_test, y_test)
model.predict_proba(X_test)
y_predicted = model.predict([[60]])
y_predicted
model.coef_
model.intercept_
import math
def sigmoid(x):
    return 1/(1+math.exp(-x))
def prediction_function(age):
    z = 0.127 * age - 4.973
    y = sigmoid(z)
    return y
age = 35
prediction_function(age)
```

→ 0.37 is less than 0.5 which means person with 35 will not buy insurance

* Logistic Regression - Multiclass

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt
iris = pd.read_csv("iris.csv")
iris.head()
x = iris.drop('species', axis='columns')
y = iris.species
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
model = LogisticRegression(multi_class='multinomial')
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=["Setosa", "Versicolor", "Virginica"])
cm_display.plot()
plt.show()
```

Output:

Accuracy of the Multinomial Logistic Regression model on the test set: 1.00

1. For dataset file "HR_comma_sep.csv"

i) Which variables did you identify as having a direct & clear impact on employee retention? Why

→ Key variables such as satisfaction_level, last_evaluation and promotion_last_5years had a direct impact on retention

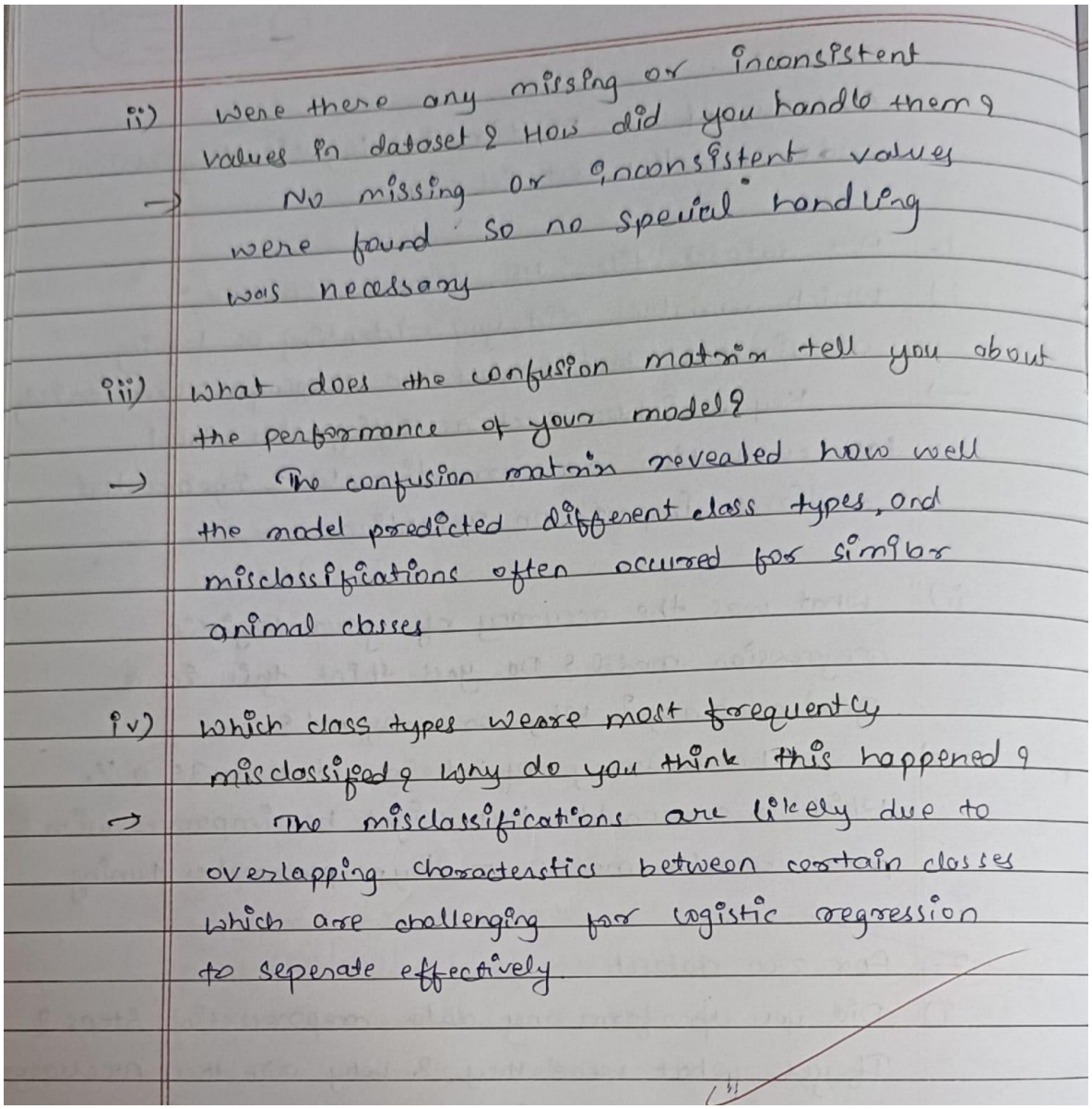
ii) What was the accuracy of your logistic regression model & Do you think this is a good accuracy? Why or why not?

→ The accuracy of the model was 75.9 % which is reasonable but might be improved with more complex models or hyperparameter tuning

2. For zoo dataset

i) Did you perform any data preprocessing steps? If yes, what were they, & why are they necessary

→ Data preprocessing steps like merging datasets, handling missing data, and encoding categorical variables were necessary to prepare the data for logistic regression




**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
```

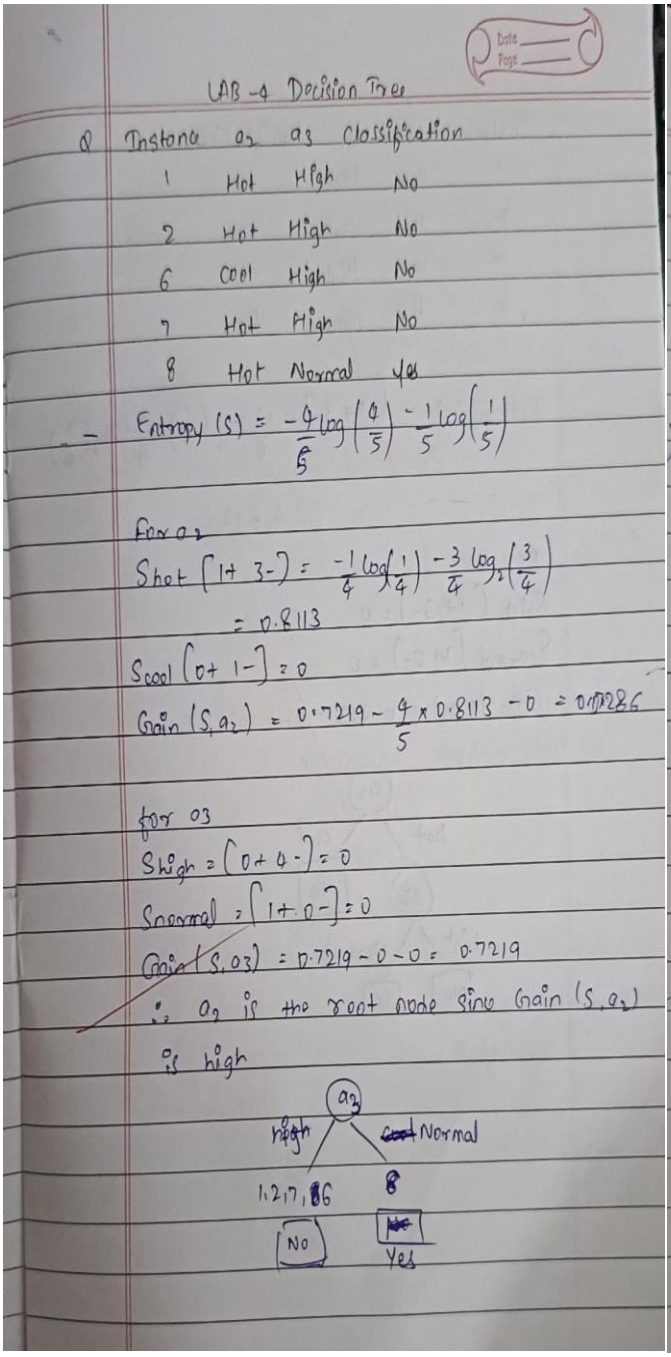
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder,
OneHotEncoder  from sklearn.preprocessing import
StandardScaler, MinMaxScaler
from scipy import stats
from sklearn import linear_model
import seaborn as sns
from sklearn.model_selection import train_test_split
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:**



LAB-4 Decision Tree

| | $a_1$ | $a_2$ | Classification |
|---|---|---|---|
| 1 | Hot | High | No |
| 2 | Hot | High | No |
| 6 | Cool | High | No |
| 7 | Hot | High | No |
| 8 | Hot | Normal | Yes |

$$\text{Entropy}(S) = -\frac{4}{5}\log\left(\frac{4}{5}\right) - \frac{1}{5}\log\left(\frac{1}{5}\right)$$

for $a_2$

$$S_{hot}[1+,3-] = -\frac{1}{4}\log\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) = 0.8113$$

$$S_{cool}[0+,1-] = 0$$

$$\text{Gain}(S,a_2) = 0.7219 - \frac{4}{5} \times 0.8113 - 0 = 0.07286$$

for $a_3$

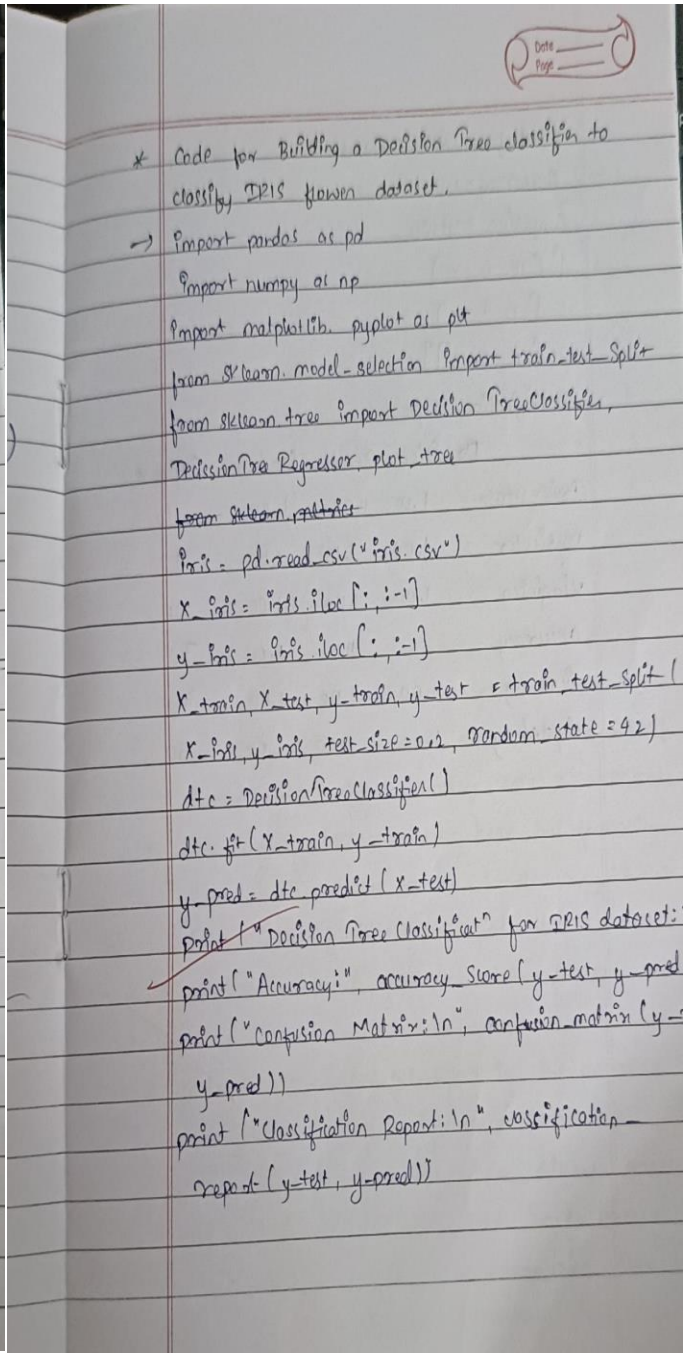
$$S_{high} = [0+,4-] = 0$$

$$S_{normal} = [1+,0-] = 0$$

$$\text{Gain}(S,a_3) = 0.7219 - 0 - 0 = 0.7219$$

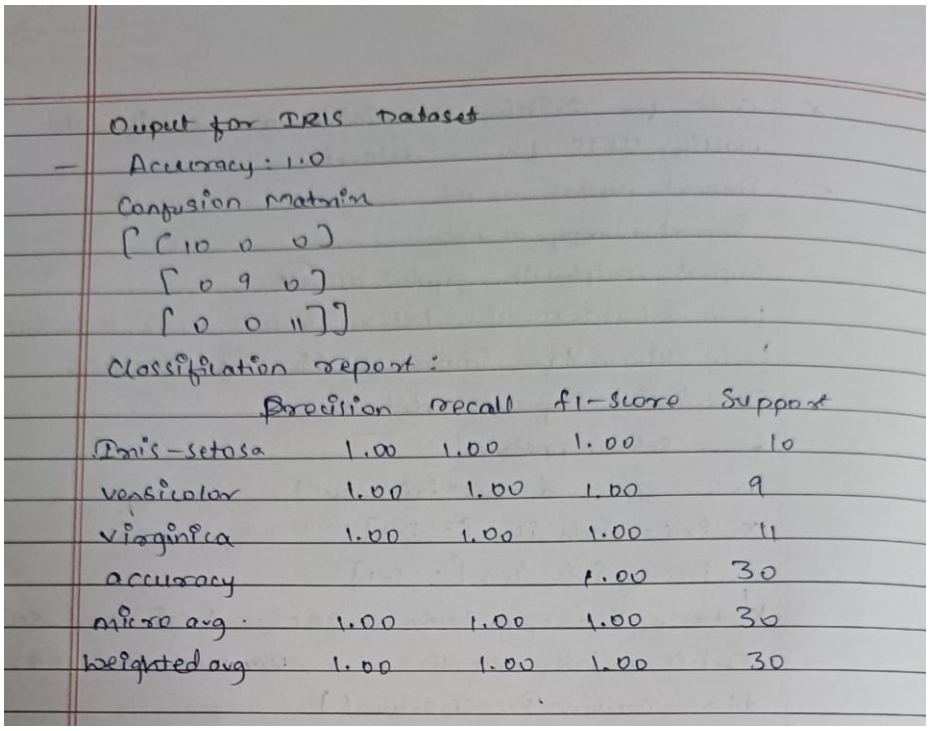
$\therefore a_3$ is the root node since Gain $(S,a_3)$ is high



* Code for Building a Decision Tree classifier to classify IRIS flower dataset.

→ import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn.metrics
iris = pd.read_csv("iris.csv")
X_iris = iris.iloc[:,:-1]
y_iris = iris.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)
print("Decision Tree Classifier for IRIS dataset:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))



Ouput for IRIS Dataset

Accuracy: 1.0

Confusion Matrix

[[10 0 0]
 [0 9 0]
 [0 0 11]]

Classification report:

| | Precision | recall | f1-score | Support |
|---|---|---|---|---|
| Iris-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 |
| micro avg | 1.00 | 1.00 | 1.00 | 30 |
| weighted avg | 1.00 | 1.00 | 1.00 | 30 |

**Code:**

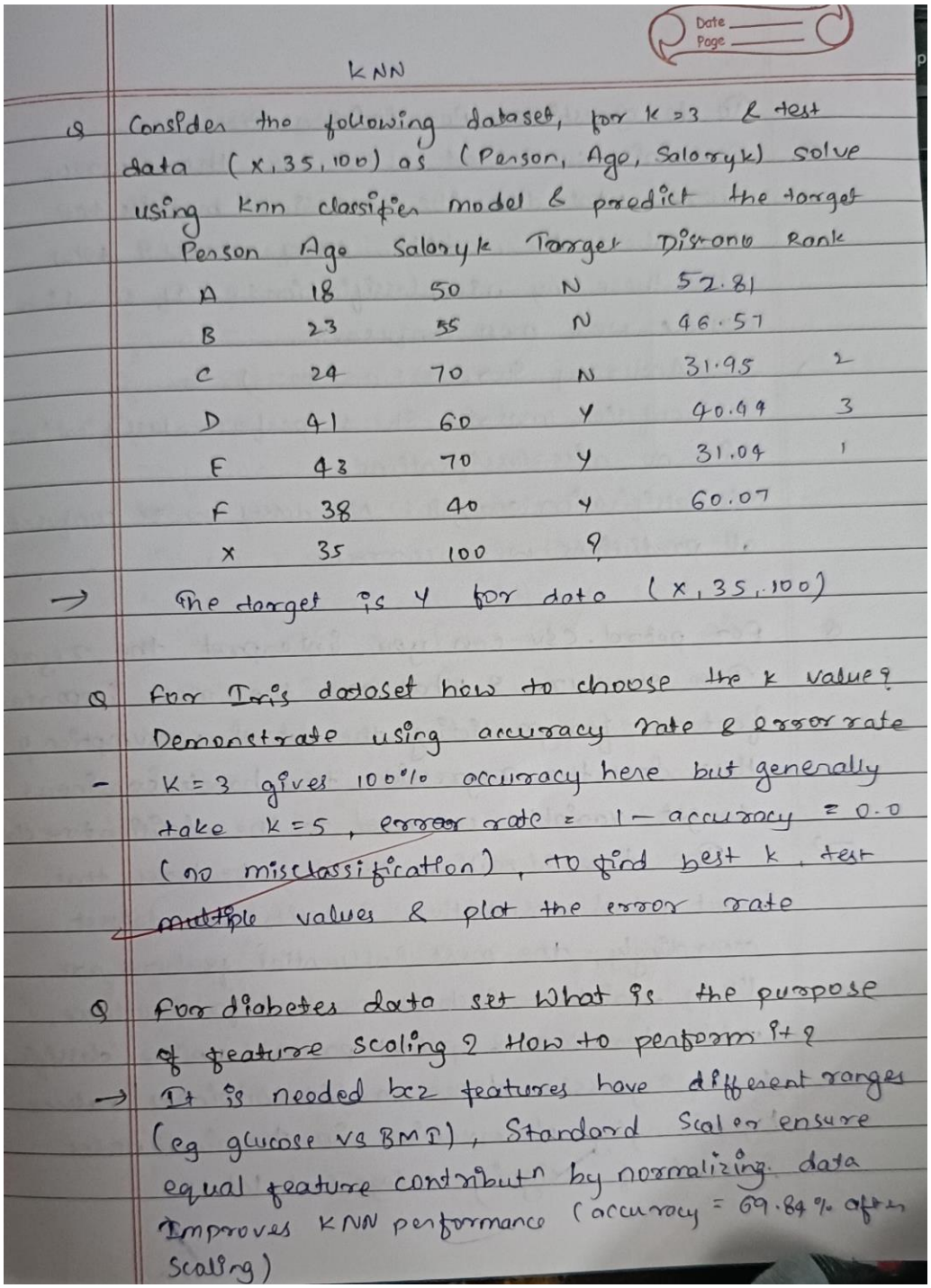
```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris.csv")
df1.head()

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 :**



KNN

Q1 Consider the following dataset, for k=3 & test data (x, 35, 100) as (Person, Age, Salary k) solve using knn classifier model & predict the target

| Person | Age | Salary k | Target | Distance | Rank |
|---|---|---|---|---|---|
| A | 18 | 50 | N | 52.81 | |
| B | 23 | 35 | N | 46.57 | |
| C | 24 | 70 | N | 31.95 | 2 |
| D | 41 | 60 | Y | 40.99 | 3 |
| F | 43 | 70 | Y | 31.04 | 1 |
| F | 38 | 40 | Y | 60.07 | |
| X | 35 | 100 | ? | | |

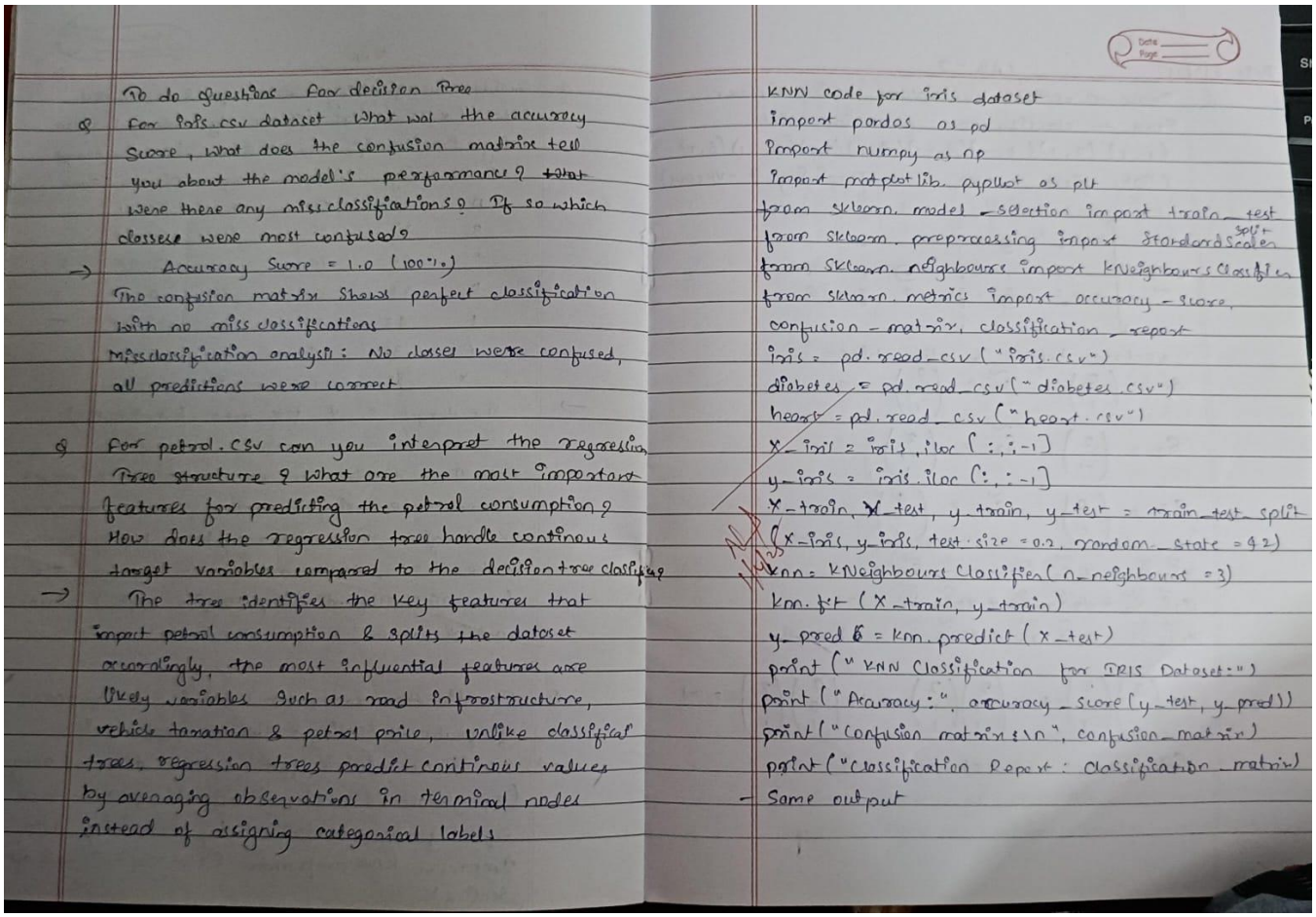
→ The target is Y for data (x, 35, 100)

Q For Iris dataset how to choose the k value? Demonstrate using accuracy rate & error rate

- k=3 gives 100% accuracy here but generally take k=5, error rate = 1 − accuracy = 0.0 (no misclassification), to find best k, test multiple values & plot the error rate

Q For diabetes data set what is the purpose of feature scaling? How to perform it?

→ It is needed bcz features have different ranges (eg glucose vs BMI), Standard Scaler ensure equal feature contribution by normalizing data Improves KNN performance (accuracy = 69.84% after scaling)




**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)
```

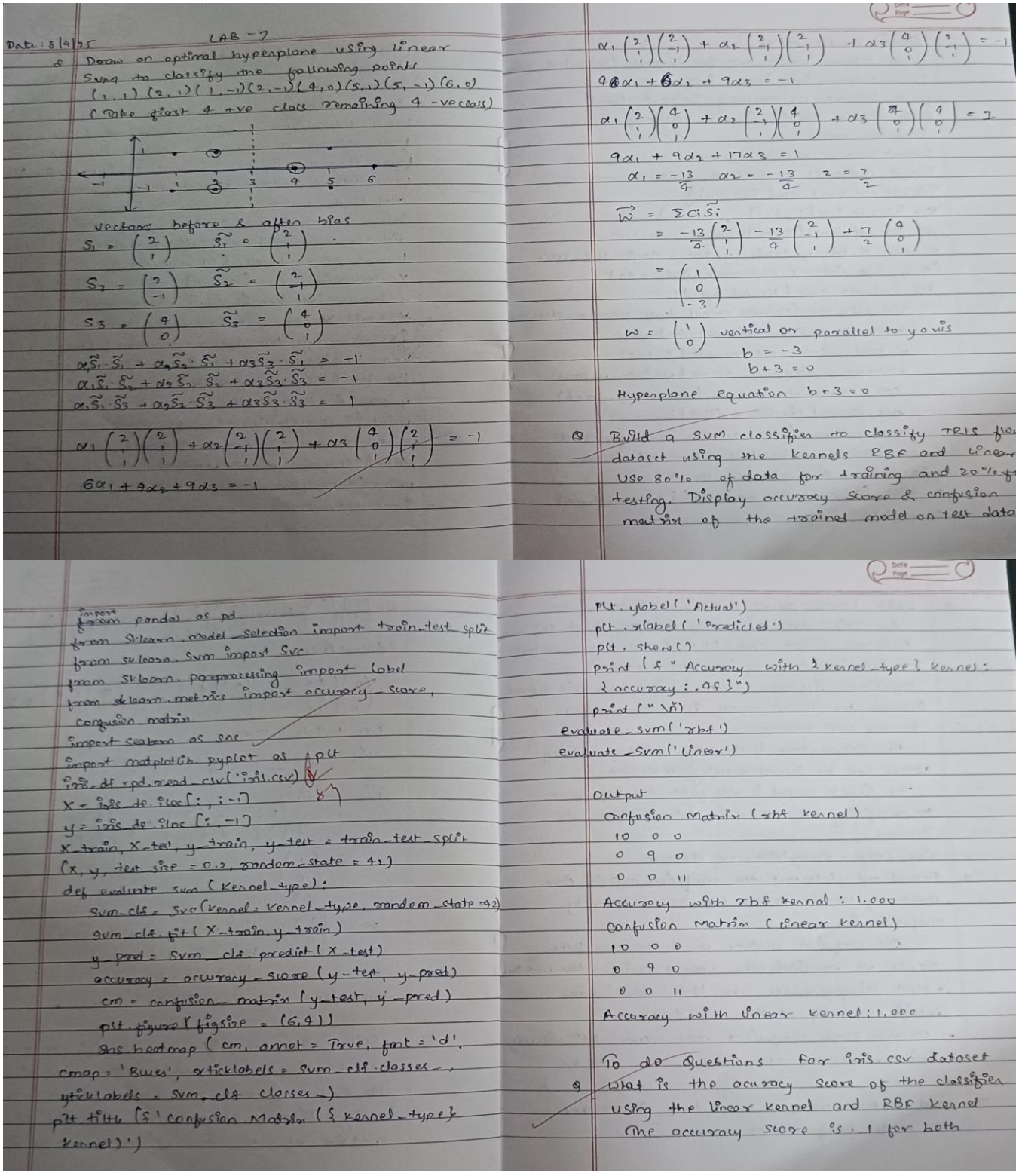
```
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            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()
```

# Program7

## Build Support vector machine model for a given dataset

## Screenshot:



Date: 8/4/25 LAB - 7

Q Draw an optimal hyperplane using linear SVM to classify the following points
(1,1) (2,1) (1,-1) (2,-1) (4,0) (5,1) (5,-1) (6,0)
(Take first 4 +ve class remaining 4 -ve class)



vectors before & after bias

$S_1 = \begin{pmatrix}2\\1\end{pmatrix}$ $\tilde{S_1} = \begin{pmatrix}2\\1\\1\end{pmatrix}$

$S_2 = \begin{pmatrix}2\\-1\end{pmatrix}$ $\tilde{S_2} = \begin{pmatrix}2\\-1\\1\end{pmatrix}$

$S_3 = \begin{pmatrix}4\\0\end{pmatrix}$ $\tilde{S_3} = \begin{pmatrix}4\\0\\1\end{pmatrix}$

$\alpha_1\tilde{S_1}\cdot\tilde{S_1} + \alpha_2\tilde{S_2}\cdot\tilde{S_1} + \alpha_3\tilde{S_3}\cdot\tilde{S_1} = -1$
$\alpha_1\tilde{S_1}\cdot\tilde{S_2} + \alpha_2\tilde{S_2}\cdot\tilde{S_2} + \alpha_3\tilde{S_3}\cdot\tilde{S_2} = -1$
$\alpha_1\tilde{S_1}\cdot\tilde{S_3} + \alpha_2\tilde{S_2}\cdot\tilde{S_3} + \alpha_3\tilde{S_3}\cdot\tilde{S_3} = 1$

$\alpha_1\begin{pmatrix}2\\1\\1\end{pmatrix}\begin{pmatrix}2\\1\\1\end{pmatrix} + \alpha_2\begin{pmatrix}2\\-1\\1\end{pmatrix}\begin{pmatrix}2\\1\\1\end{pmatrix} + \alpha_3\begin{pmatrix}4\\0\\1\end{pmatrix}\begin{pmatrix}2\\1\\1\end{pmatrix} = -1$

$6\alpha_1 + 4\alpha_2 + 9\alpha_3 = -1$

$\alpha_1\begin{pmatrix}2\\1\\1\end{pmatrix}\begin{pmatrix}2\\-1\\1\end{pmatrix} + \alpha_2\begin{pmatrix}2\\-1\\1\end{pmatrix}\begin{pmatrix}2\\-1\\1\end{pmatrix} + \alpha_3\begin{pmatrix}4\\0\\1\end{pmatrix}\begin{pmatrix}2\\-1\\1\end{pmatrix} = -1$

$4\alpha_1 + 6\alpha_2 + 9\alpha_3 = -1$

$\alpha_1\begin{pmatrix}2\\1\\1\end{pmatrix}\begin{pmatrix}4\\0\\1\end{pmatrix} + \alpha_2\begin{pmatrix}2\\-1\\1\end{pmatrix}\begin{pmatrix}4\\0\\1\end{pmatrix} + \alpha_3\begin{pmatrix}4\\0\\1\end{pmatrix}\begin{pmatrix}4\\0\\1\end{pmatrix} = 1$

$9\alpha_1 + 9\alpha_2 + 17\alpha_3 = 1$

$\alpha_1 = \frac{-13}{4}$ $\alpha_2 = \frac{-13}{4}$ $\alpha_3 = \frac{7}{2}$

$\vec{w} = \sum c_i\vec{s_i}$

$= \frac{-13}{4}\begin{pmatrix}2\\1\\1\end{pmatrix} - \frac{13}{4}\begin{pmatrix}2\\-1\\1\end{pmatrix} + \frac{7}{2}\begin{pmatrix}4\\0\\1\end{pmatrix}$

$= \begin{pmatrix}1\\0\\-3\end{pmatrix}$

$w = \begin{pmatrix}1\\0\end{pmatrix}$ vertical or parallel to y axis

$b = -3$

$b + 3 = 0$

Hyperplane equation $b + 3 = 0$

Q Build a SVM classifier to classify IRIS flo dataset using the kernels RBF and linear Use 80% of data for training and 20% of testing. Display accuracy score & confusion matrix of the trained model on test data

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import label
from sklearn.metrics import accuracy_score,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
iris_df = pd.read_csv('iris1.csv')
X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size = 0.2, random_state = 42)
def evaluate_svm(kernel_type):
    svm_clf = SVC(kernel = kernel_type, random_state =42)
    svm_clf.fit(X_train, y_train)
    y_pred = svm_clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize = (6,4))
    sns.heatmap(cm, annot = True, fmt = 'd',
cmap = 'Blues', xticklabels = svm_clf.classes_,
yticklabels = svm_clf.classes_)
    plt.title(f'Confusion Matrix ({kernel_type}
Kernel)')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
    print(f"Accuracy with {kernel_type} Kernel:
{accuracy:.0f}")
    print("\n")
evaluate_svm('rbf')
evaluate_svm('linear')
```

Output

Confusion Matrix (rbf kernel)

10 0 0
0 9 0
0 0 11

Accuracy with rbf kernal : 1.000

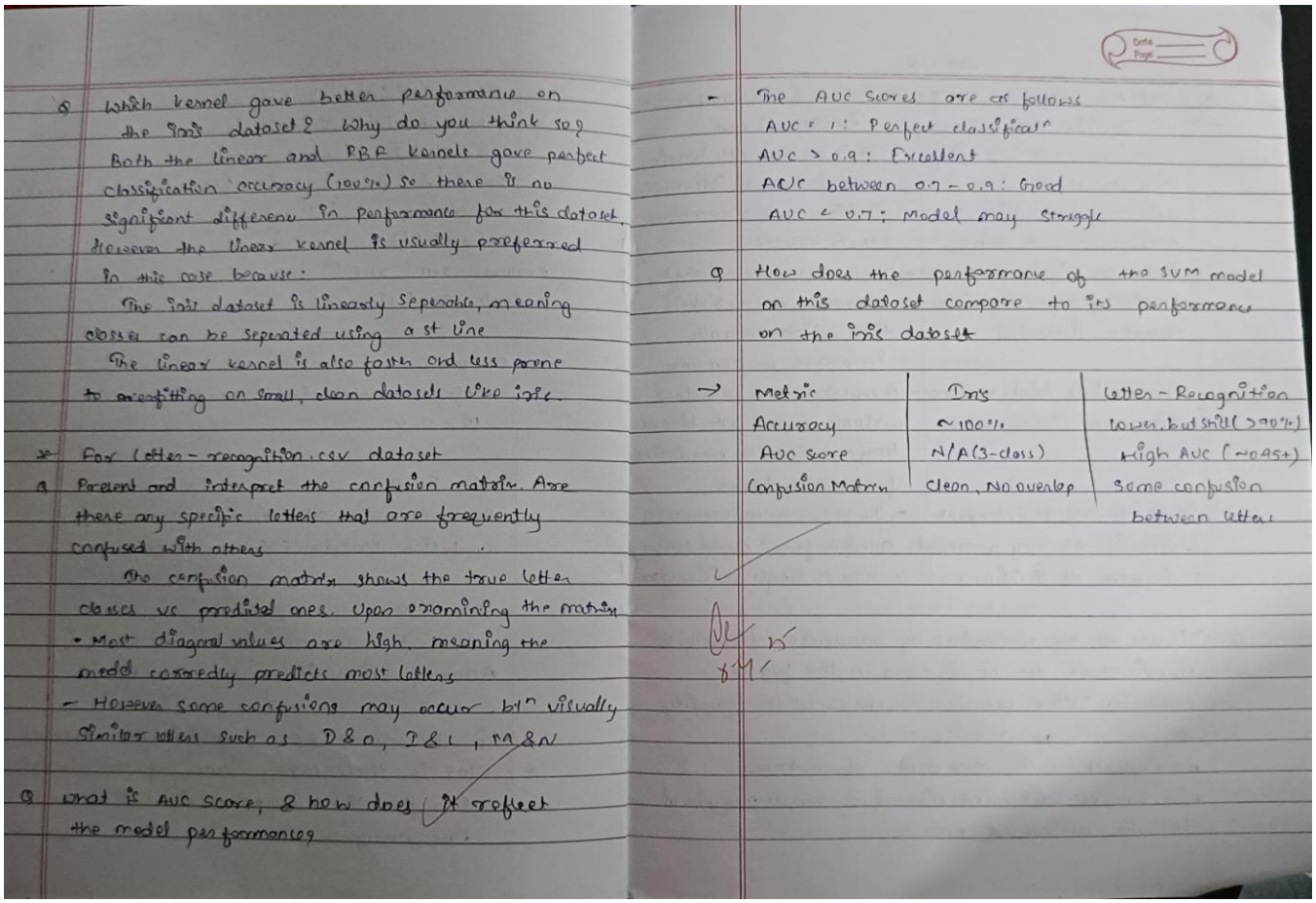
Confusion Matrix (linear kernel)

10 0 0
0 9 0
0 0 11

Accuracy with linear kernel: 1.000

To do Questions for iris csv dataset

Q What is the accuracy score of the classifier using the linear kernel and RBF kernel
The accuracy score is 1 for both




**Code:**

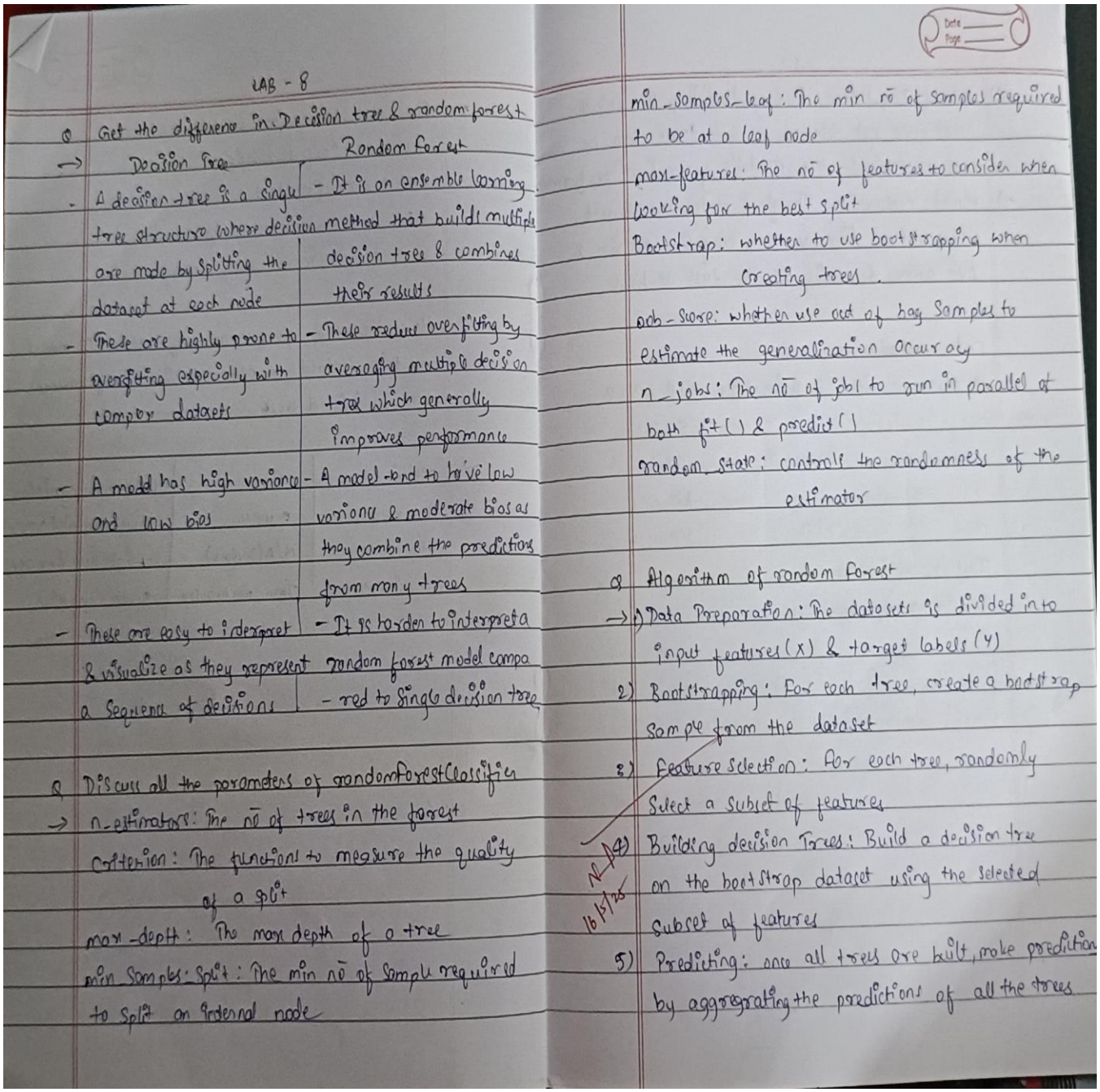
```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (1).csv")
df1.head()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
rbf_y_pred = 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**

**Screenshot:**



LAB - 8

Q Get the difference in Decision tree & random forest

→

| Decision Tree | Random Forest |
|---|---|
| - A decision tree is a single tree structure where decision are made by splitting the dataset at each node | - It is an ensemble learning method that build multiple decision tree & combines their results |
| - These are highly prone to overfitting especially with complex datasets | - These reduce overfitting by averaging multiple decision tree which generally improves performance |
| - A model has high variance and low bias | - A model tend to have low variance & moderate bias as they combine the predictions from many trees |
| - These are easy to interpret & visualize as they represent a sequence of decisions | - It is harder to interpret a random forest model compared to single decision tree |

Q Discuss all the parameters of randomforestClassifier

→ n_estimators: The nō of trees in the forest

criterion: The function to measure the quality of a split

max_depth: The max depth of a tree

min_samples_split: The min nō of sample required to split an internal node

min_samples_leaf: The min nō of samples required to be at a leaf node

max_features: The nō of features to consider when looking for the best split

Bootstrap: whether to use bootstrapping when creating trees.

oob_score: whether use out of bag samples to estimate the generalization accuracy

n_jobs: The nō of jobs to run in parallel of both fit() & predict()

random_state: controls the randomness of the estimator

Q Algorithm of random forest

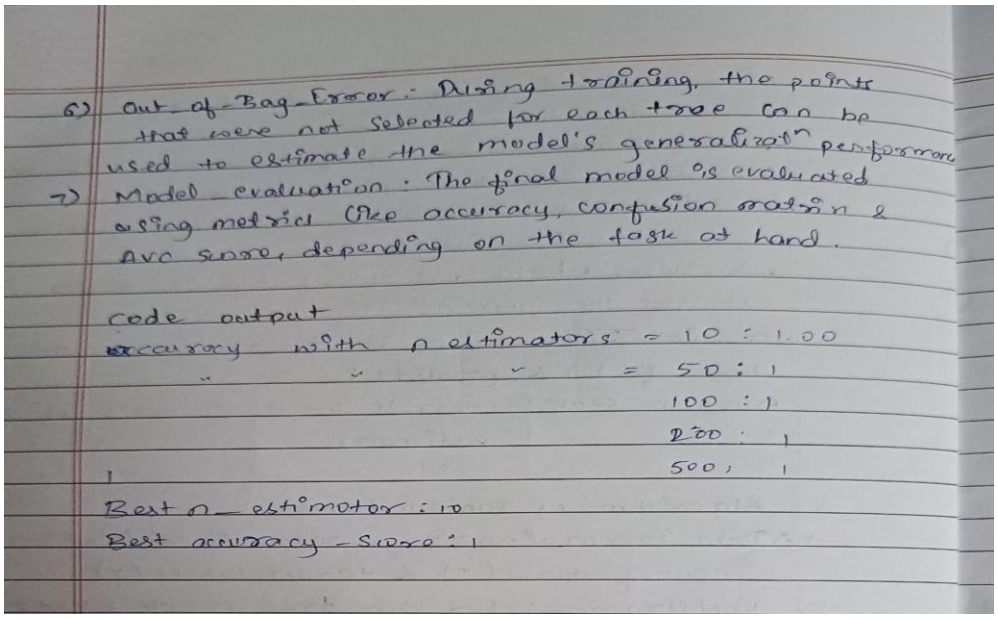
→ 1) Data Preparation: The dataset is divided into to input features (x) & target labels (y)

2) Bootstrapping: For each tree, create a bootstrap sample from the dataset

3) Feature selection: For each tree, randomly select a subset of features

4) Building decision Trees: Build a decision tree on the bootstrap dataset using the selected subset of features

5) Predicting: once all trees are built, make prediction by aggregating the predictions of all the trees



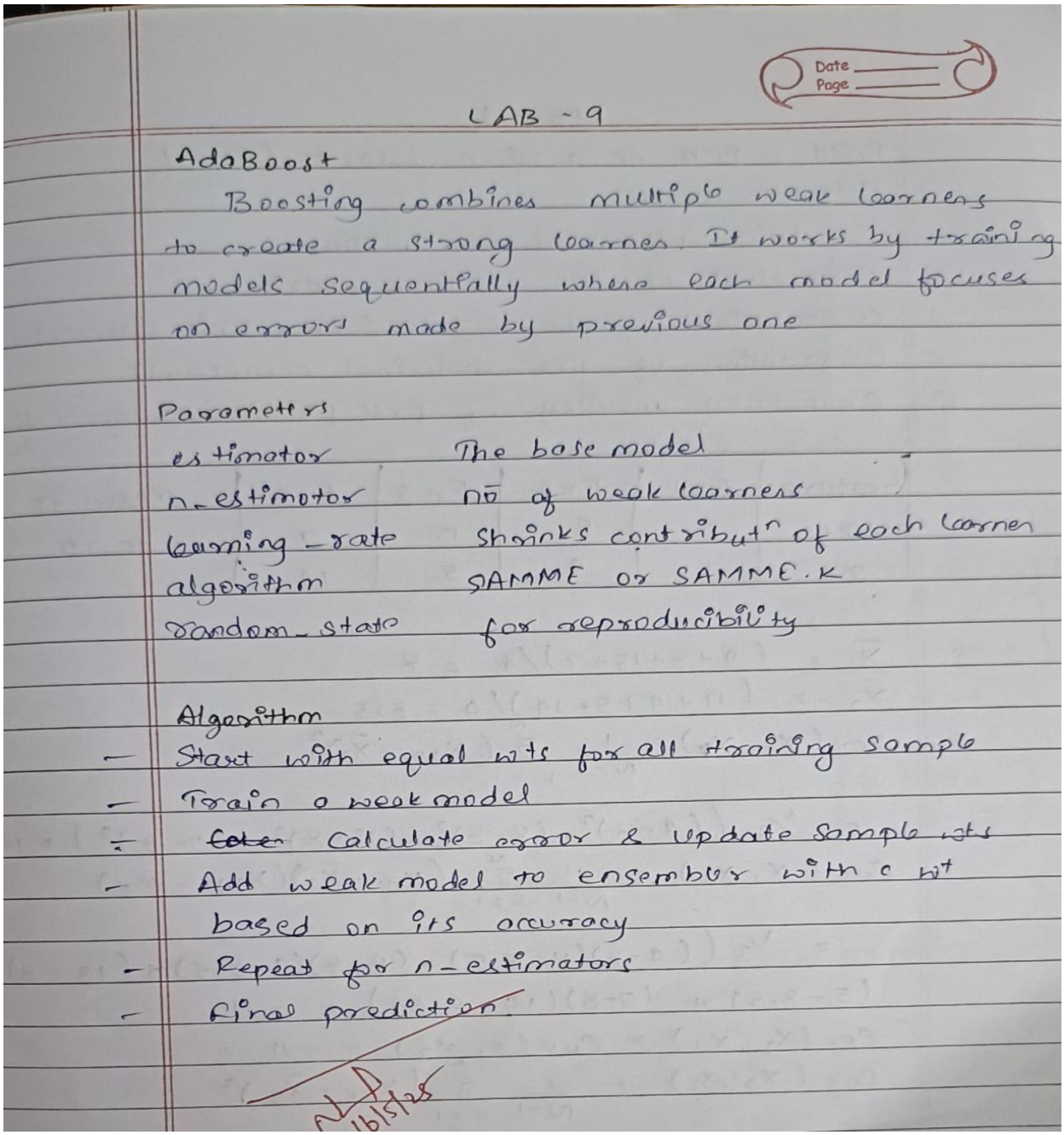

**Code:**

```
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:**



LAB - 9

AdaBoost

Boosting combines multiple weak learners to create a strong learner. It works by training models sequentially where each model focuses on errors made by previous one

Parameters

| | |
|---|---|
| estimator | The base model |
| n_estimator | no of weak learners |
| learning_rate | shrinks contribut^n of each learner |
| algorithm | SAMME or SAMME.R |
| random_state | for reproducibility |

Algorithm

- Start with equal wts for all training sample
- Train a weak model
- Calculate error & update sample wts
- Add weak model to ensembler with a wt based on its accuracy
- Repeat for n-estimators
- Final prediction

16/5/25

**Code:**

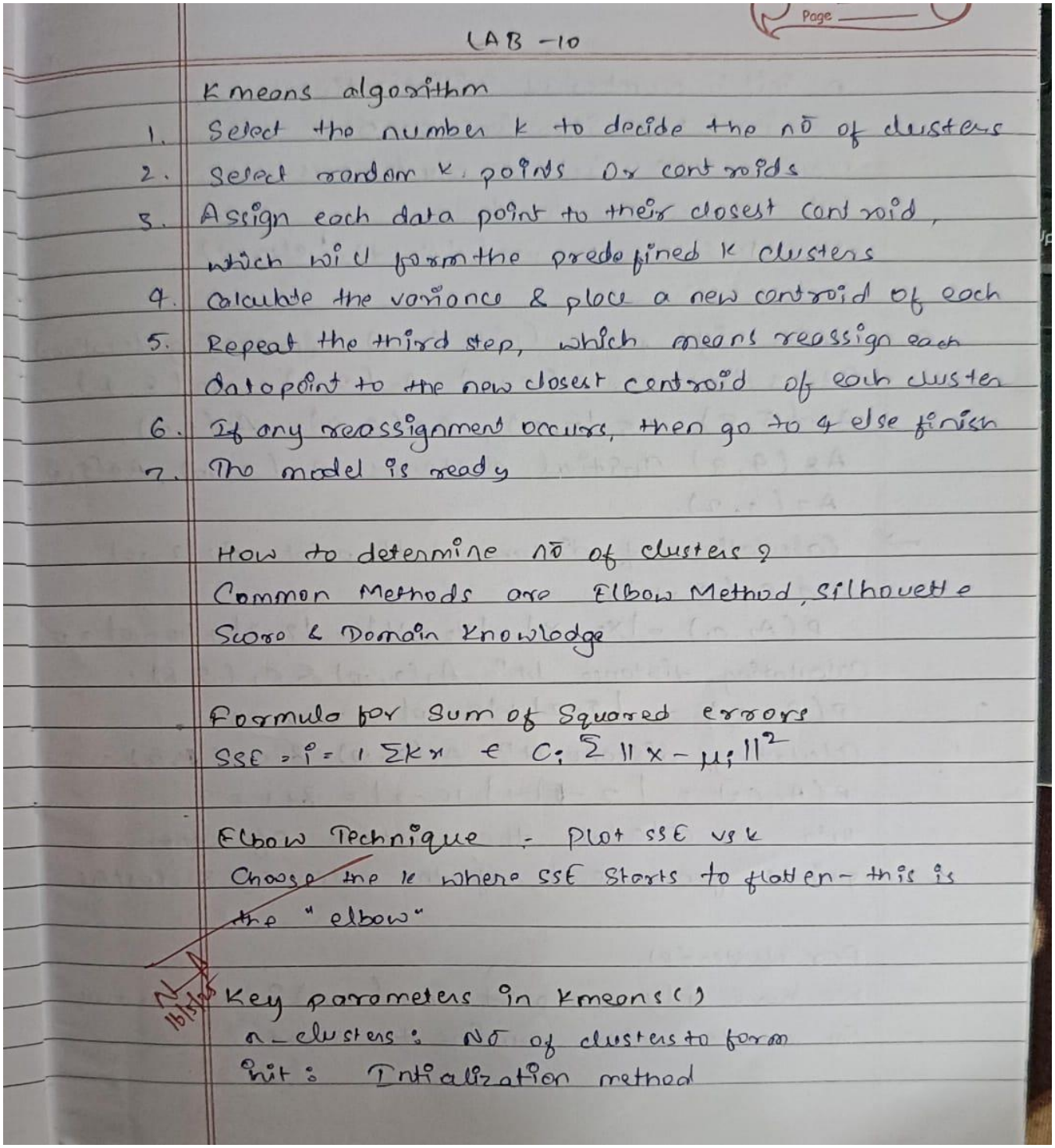
```
from google.colab import files
income=files.upload()
df1=pd.read_csv("income.csv")
df1.head()
X = df1.drop('income_level', axis=1)
```

```
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:**



LAB - 10

Kmeans algorithm

1. Select the number k to decide the no of clusters
2. Select random k points or centroids
3. Assign each data point to their closest centroid, which will form the predefined k clusters
4. Calculate the variance & place a new centroid of each
5. Repeat the third step, which means reassign each datapoint to the new closest centroid of each cluster
6. If any reassignment occurs, then go to 4 else finish
7. The model is ready

How to determine no of clusters ?
Common Methods are Elbow Method, Silhouette Score & Domain Knowledge

Formula for Sum of Squared errors
$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$

Elbow Technique : Plot SSE vs k
Choose the k where SSE starts to flatten - this is the "elbow"

Key parameters in Kmeans()
n_clusters : No of clusters to form
init : Initialization method

n_init : Number of initialization
max_iter : Max iterations per run
random_state : Control randomness
tol : convergence threshold
algorithm : Algo to use ['lloyd', 'elkan']

Cluster eight points (with (x,y) representing locat
into 3 clusters. A1(2,10), A2(2,5), A3(8,4)
A4(5,8), A5(7,5), A6(6,4), A7(1,2),
A8(4,9) Initial clusters: A1(2,10), A4(5
A7(1,2)

→ Calculation of distance betn point A1(2,10)
and C1(2,10):
$P(A_1, C_1) = |x_2 - x_1| + |y_2 - y_1| = |2-2| + |10-10|$
Calculating distance btn A1(2,10) & C2(5,8):
$P(A_1, C_2) = |5-2| + |8-10| = 3+2 = 5$
Calculating distance btn A1(2,10) & C3(4,2)
$P(A_1, C_3) = |2-4| + |10-2| = 8+1 = 9$
Like we calculate the distance of other
points from each of the center of 3 clust

For cluster-01
- We have only one point A1(2,10) in cluster
- So cluster center remains the same

| Given Pts | Distance from $C_1$ | Dist from $C_2$ | Dist from $C_3$ | Point Belong to |
|---|---|---|---|---|
| A1(2,10) | 0 | 5 | 9 | C1 |
| A2(2,5) | 5 | 6 | 4 | C3 |
| A3(8,4) | 12 | 7 | 9 | C2 |
| A4(5,8) | 5 | 0 | 10 | C2 |
| A5(7,5) | 10 | 5 | 9 | C2 |
| A6(6,4) | 10 | 5 | 7 | C2 |
| A7(1,2) | 9 | 10 | 0 | C3 |
| A8(4,9) | 3 | 2 | 10 | C2 |

For cluster-02
Center of cluster-02 = ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6,6)
For cluster-03
Center of cluster-03 = ((2+1)/2, (5+2)/2) = (1.5, 3.5), This is completion of Iteration-01

Iteration-02

| Given Pts | Dist from $C_1$ | Dist from $C_2$ | Dist from $C_3$ | Point Belongs to cluster |
|---|---|---|---|---|
| A1(2,10) | 0 | 8 | 7 | C1 |
| A2(2,5) | 5 | 5 | 2 | C3 |
| A3(8,4) | 12 | 4 | 7 | C2 |
| A4(5,8) | 5 | 3 | 8 | C2 |
| A5(7,5) | 10 | 2 | 7 | C2 |
| A6(6,4) | 10 | 2 | 5 | C2 |
| A7(1,2) | 9 | 9 | 2 | C3 |
| A8(4,9) | 3 | 5 | 8 | C1 |

For C1 Center = ((2+4)/2, (10+9)/2) = (3, 9.5)
For C2 Center = ((8+5+7+6)/4, (4+8+5+4)/4) = (6.5, 5.25)
For C3 Center = ((1+2)/2, (5+2)/2) = (1.5, 3.5)

Iteration-03

| Given Pts | Distance from $C_1$ | Dist from $C_2$ | Dist from $C_3$ | Point belongs to cluster |
|---|---|---|---|---|
| A1(2,10) | 1.5 | 9.25 | 7 | C1 |
| A2(2,5) | 5.5 | 4.75 | 2 | C3 |
| A3(8,4) | 10.5 | 2.75 | 7 | C2 |
| A4(5,8) | 3.5 | 4.75 | 8 | C1 |
| A5(7,5) | 8.5 | 0.75 | 7 | C2 |
| A6(6,4) | 8.5 | 1.75 | 5 | C2 |
| A7(1,2) | 9.5 | 8.75 | 2 | C3 |
| A8(4,9) | 1.5 | 6.25 | 8 | C1 |

For C1 (2+5+4)/3 = 3.66, (10+8+9)/3 = 9
For C2 (8+7+6)/3 = 7, (4+5+4)/3 = 4.33
For C3 (2+1)/2 = 1.5, 3.5

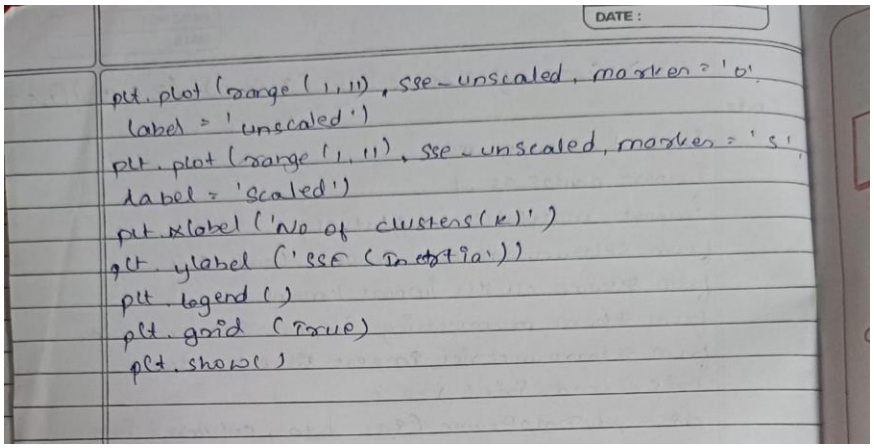
Iteration-04

| Given Pts | Dfrom $C_1$ | Dfrom $C_2$ | Dfrom $C_3$ | Point |
|---|---|---|---|---|
| A1(2,10) | 1.66 | 10.67 | 8 | C1 |
| A2(2,5) | 5.66 | 5.67 | 2 | C3 |
| A3(8,4) | 9.34 | 1.33 | 7 | C2 |
| A4(5,8) | 2.34 | 5.67 | 8 | C1 |
| A5(7,5) | 7.34 | 0.67 | 7 | C2 |
| A6(6,4) | 7.34 | 1.33 | 6 | C2 |
| A7(1,2) | 9.66 | 8.33 | 2 | C3 |
| A8(4,9) | 0.34 | 7.66 | 8 | C1 |

→ Centers of clusters are (3.66,9), (7,4.33), (1.5,3.5)

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
Iris = load_iris()
df = pd.DataFrame(Iris.data, columns = Iris.feature_names)
df['target'] = Iris.target
X = df[['petal length (cm)', 'petal width (cm)']].values
sse_unscaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    sse_unscaled.append(kmeans.inertia_)
Scaler = StandardScaler()
X_scaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    sse_scaled.append(kmeans.inertia_)
plt.figure(figsize=(10,5))
plt.
```



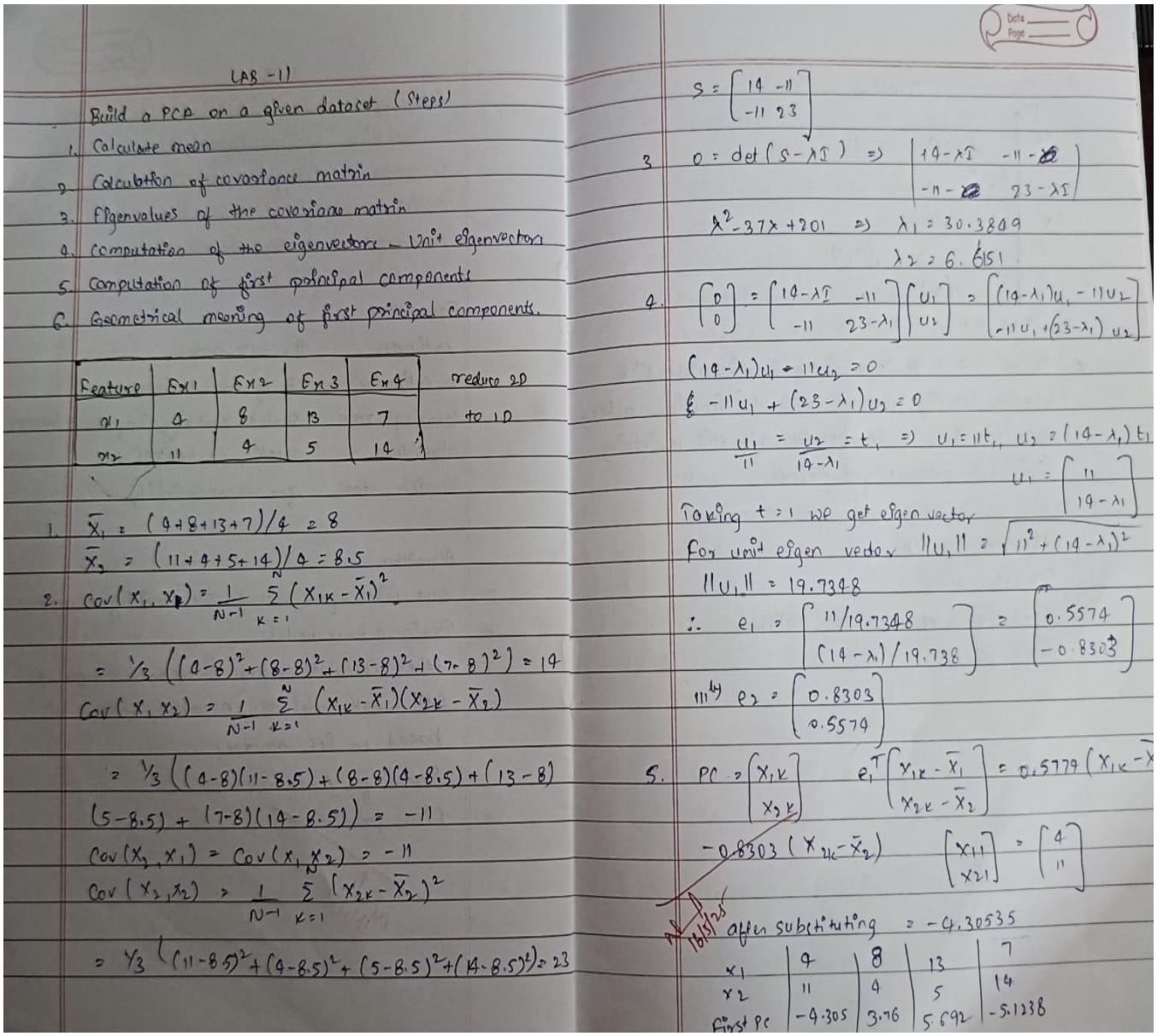

**Code:**

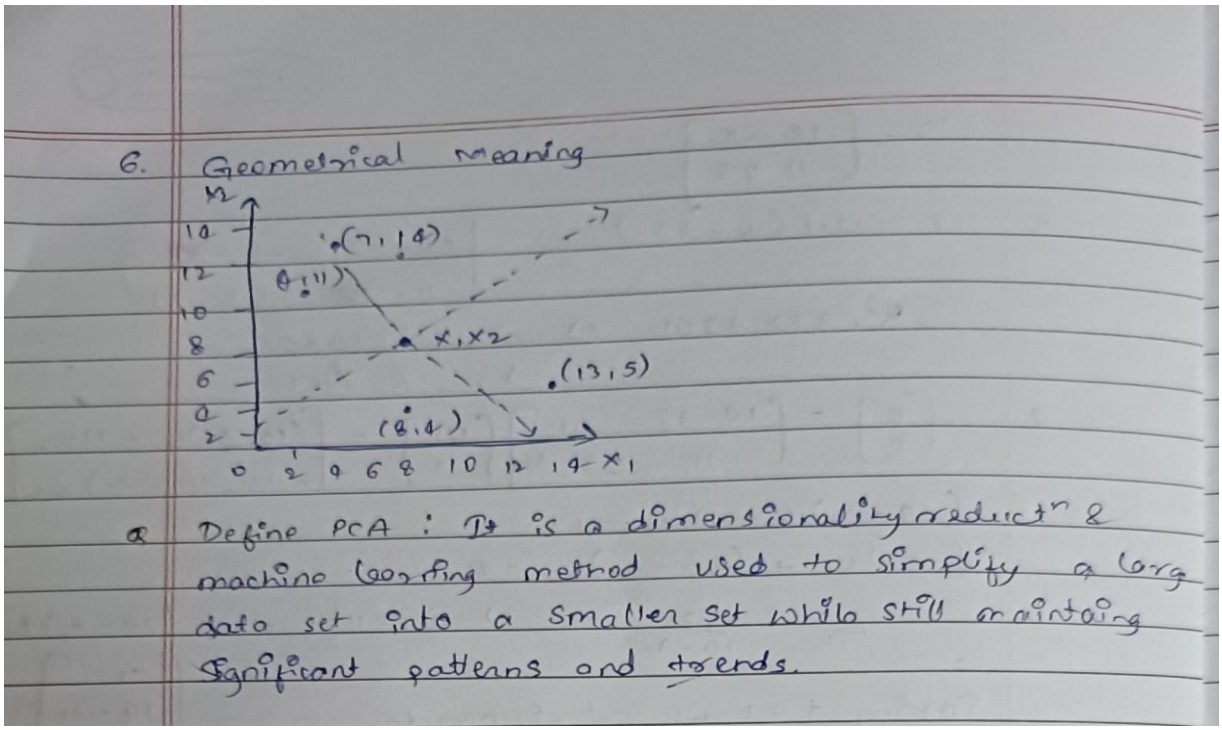
```
from google.colab import files
iris=files.upload()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import 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]}")
```