VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum- 590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

ARBAJ WADAGERA (1BM22CS051)

in partial fulfilment 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
February 2025 – June 25

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 **Arbaj Wadagera (1BM22CS051),** who is Bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Laboratory report has been approved as it satisfies the academic requirements in respect of a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty InCharge's:	
Saritha A. N Assistant Professor Department of CSE BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

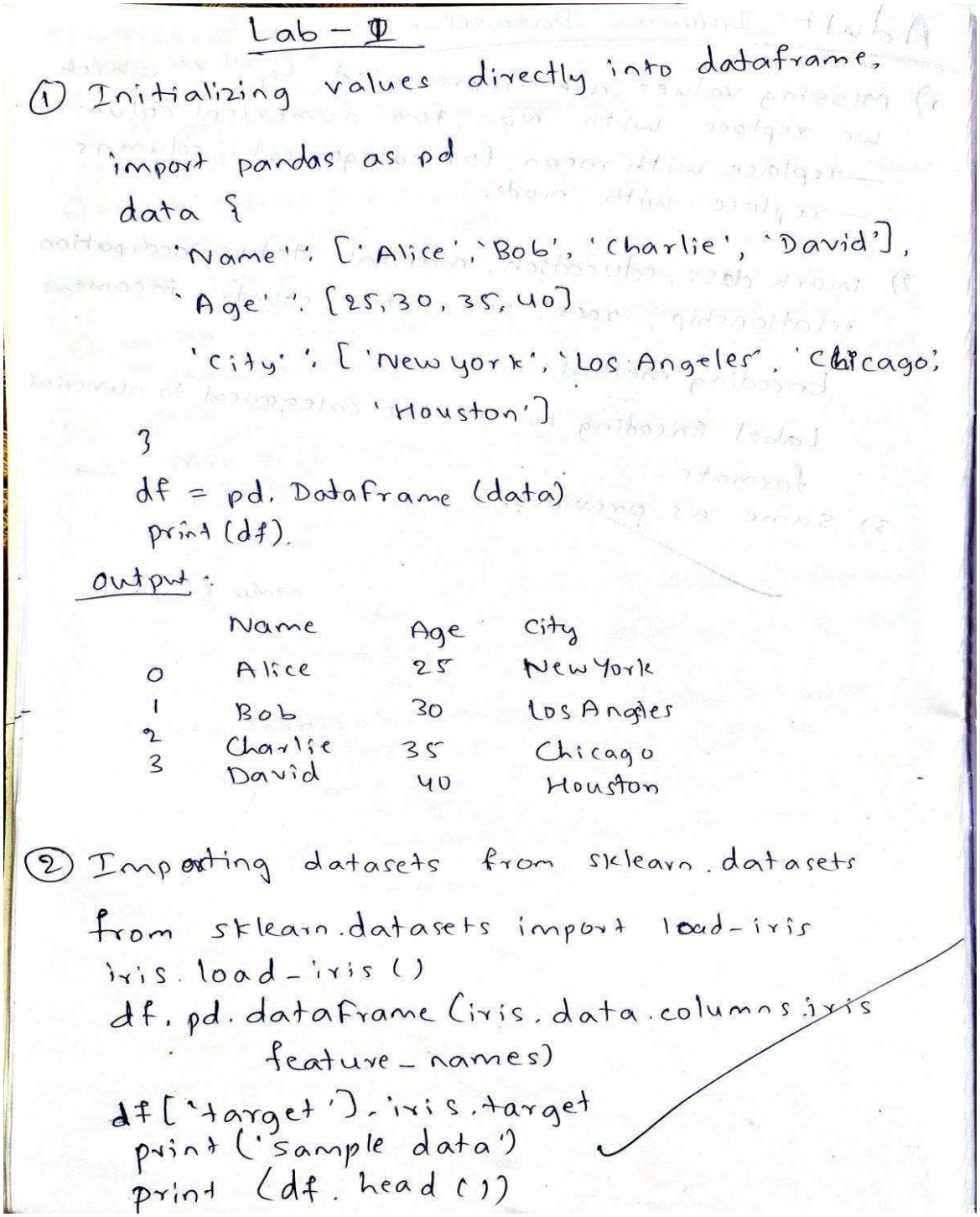
INDEX

GitHub Link: https://github.com/Arbaj-Wadagera/6thSem-ML-Lab

Program 1

Write a python program to import and export data using Panda's library functions

Screenshot:



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 Angeles', 'Chicago',
             'Houston']
}
df = pd.DataFrame (data)
print (df).
```

output:

	Name	Age	City
0	Alice	25	New York
1	Bob	30	Los Angles
2	Charlie	35	Chicago
3	David	40	Houston

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
1	4.9	5.4	2.0	0.3	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
1	2	Bob	29
2	3	Charlie	21

(4) Downloading datasets from existing dataset repositiories like kaggle

```
df. pd. read- csv ("MobileDataset.csv")
print (df.head (1)
```

output

Comp. name	modelname	weight	Ram	Front Camera
Apple	iphone 16	144g	6GB	12mp
Back Cam	Processor	Battery cap	source size	
48mp	A17 Bionic	3600 mah	6-Inches	

Bank - Data Analysis

- using the code of 'stock Market Data Analysis' consider the following.

① HDFC Bank Ltd, ICICI Bank Ltd, Kotak Bank Ltd, Mahindra Bank Ltd

tickers = ["HDFCBANK.NS", "ICICI BANK.NS", "KOTAKBANK.NS"]

② Start date = 2024-01-01 End date = 2024-12-30

③ Plot the closing price & daily returns for all the 3 banks mentioned.

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers [HDFCBANK.NS, ICICI BANK.NS,
         KOTAK BANK.NS]
data.yfdownload [tickers, start: '2024-01-01',
  end '2024-12-30', group_by.tickers].
print (df.head())
print (data.shape)
```

output

Tickers	KOTAK BANK.NS				
Price	Open	High	Low	Close	Volume
2024-01-01	1906	1916	1891	1903.0	14259

Code:

```
from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()

df['target'] = iris.target
df
import kagglehub


# Download latest version
path = kagglehub.dataset_download("abdulmalik1518/mobiles-dataset-2025")


print("Path to dataset files:", path)


df = pd.read_csv("/content/Mobiles_Dataset_(2025).csv", encoding='latin-1') # or 'ISO-8859-1', or
'cp1252'
df.head()
df['Company Name']


data = {"USN" : ['1', "2", "3"], "Name" : ["A", "B", "C"]}
df = pd.DataFrame(data)
df
from sklearn.datasets import load_diabetes
```

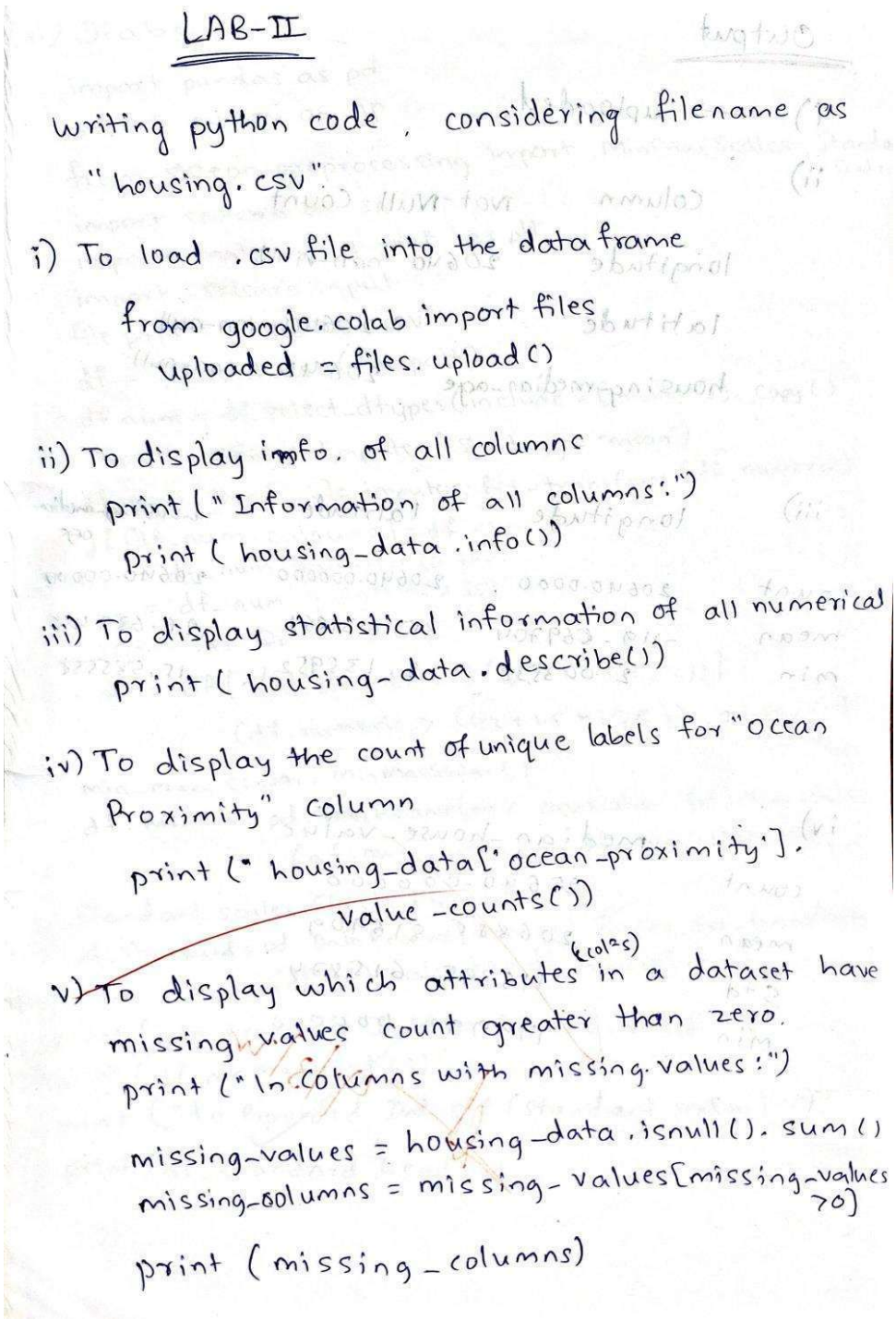
```
diabetes = load_diabetes()

df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)

df.head()

df.columns

df = pd.read_csv("/content/Dataset_of_Diabetes .csv")

df.head()

import yfinance as yf import pandas as pd

import matplotlib.pyplot as plt



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

# Fetch historical data for the last 1 year

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


# Display the first 5 rows of the dataset

print("First 5 rows of the dataset:")


print(data.head())

print("\nShape of the dataset:")

print(data.shape)
```

```
# Summary statistics for a specific stock (e.g., Reliance)

reliance_data = data['RELIANCE.NS']

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

print(reliance_data.describe())

# Calculate daily returns

reliance_data['Daily Return'] = reliance_data['Close'].pct_change()


# Plot the closing price and daily returns

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()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



LAB-II

Writing python code, considering filename as "housing.csv".

i) To load .csv file into the data frame

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

ii) To display info. of all columns

```
print("Information of all columns:")
print(housing_data.info())
```

iii) To display statistical information of all numerical

```
print(housing_data.describe())
```

iv) To display the count of unique labels for "ocean Proximity" column

```
print("housing_data['ocean_proximity'].
      value_counts())
```

v) To display which attributes (cols) in a dataset have missing values count greater than zero.

```
print("In columns with missing values:")
missing_values = housing_data.isnull().sum()
missing_columns = missing_values[missing_values
                                 >0]
print(missing_columns)
```

Output

i) ⟶ uploaded

ii)

Column	Not-Null Count
longitude	20640 non-Null
latitude	20640 non-null
housing-median-age	20640 non-null

iii)

	longitude	latitude	housing-median-age
count	20640.0000	20640.000000	20640.00000
mean	-119.569704	35.631861	28.639486
min	2.003532	2.135952	15.585558

iv)

	median_house_value
count	20640.000000
mean	206855.816909
std	115395.615874
min	149999.000000

4/3/24

(ii) Diabetes.csv

```
import pandas as pd
import numpy as np
from skleon.preprocessing import MinMaxScales, Standa
                                                    scale.
import seaborn as sn
import matplotlib.pyplot as plt
import sklearn.input
file.path = "diabetes.csv"
df = pd.read_csv(file_path)
df.num = df.select_dtypes(include = ['number']).copy()
imputer = SimpleImputer( Strategy = "mean")
df_mean.iloc[:,:]= imputer.fit_transform (df_numeric)
df[df_num.colouns] = df.numeric
Q1 = df_num.quantile(0.25)
Q3 = df_num.quantile(0.75)
IQR = Q3-Q1
df = df[~((df_numeric < (Q1 - 1.5 * IQR))]
        (df_numeric > (Q3 + 1.5 * IQR)) .any(axis=1)

min_max_scalar = MinMaxScalar()
df_minmax= pd.DataFrame(min.maxscalar.fit.transform
                (af_num., columns = df_num.columns)

Standard_scales.StandardScaler()
df_Standard = pd.DataFrame(standard_scales.fit_transform
                (df_num) columns = df_num.columns)

print(" In Processed Dataset (minmax scaled):")
print (df_minmax.head())
print ("In Provided Dataset (standard scalar):")
print (df_standard.head())
```

a) which columns in the dataset had missing values. How to handle them

Ans:- missing values are present in numericals columns if present, which are replaced by the mean of the respective column.

b) which categorical columns did you identify? How did you encode them?

Ans:- No categorical column, hence no encoding.

c) what is the difference b/w min max scaling & standardization? when would you use one over the other?

Ans:- Min max scaling transform data to a final range [0, 1] using

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

used when: - dataset does not follow a normal dist.
- features have diff. ranges & used to be bound

Standardization transforms data to have zero mean & unit variance

$$x' = \frac{x - r}{\sigma}$$

used when: - dataset follows a Gaussian dist.
- many ML algos. 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) work class, education, marital status, occupation, relationship, race, sex, native, country income.

   Encoding method:

   Label Encoding to convert categorical to numerical formats

3) same as previous.

Code:

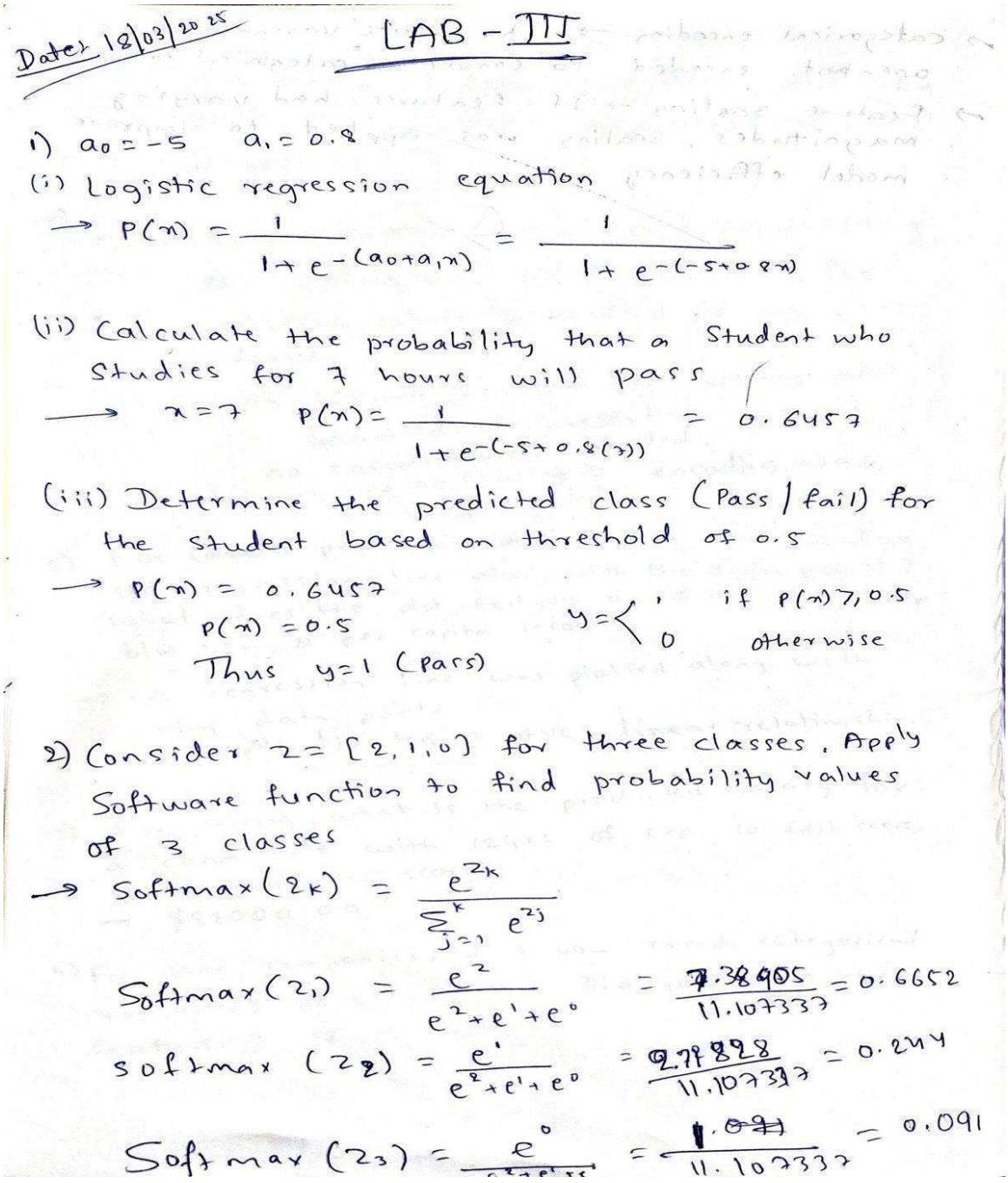
```
import pandas as pd

import numpy as np

# Load dataset

df = pd.read_csv("data.csv")

print(df.head())

# Check missing values

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


# Drop rows with missing values

df_cleaned = df.dropna()


# Or fill missing values with mean/median

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

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


# For nominal categories

df = pd.get_dummies(df, columns=['Gender', 'Country'], drop_first=True
# For ordinal categories

from sklearn.preprocessing import OrdinalEncoder

encoder = OrdinalEncoder()

df[['Education_Level']] = encoder.fit_transform(df[['Education_Level']])
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization (Z-score)

scaler = StandardScaler()
```

```
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])

# Min-Max Normalization

minmax = MinMaxScaler()

df[['Age', 'Salary']] = minmax.fit_transform(df[['Age', 'Salary']])

# Using IQR method

Q1 =df['Salary'].quantile(0.25)

Q3 = df['Salary'].quantile(0.75)

IQR = Q3 - Q1

df = df[(df['Salary'] >= Q1 - 1.5*IQR) & (df['Salary'] <= Q3 + 1.5*IQR)]

df ['Age_Salary_Ratio'] = df['Age'] / df['Salary']



# Drop irrelevant columns

df.drop(['User_ID', 'Name'], axis=1, inplace=True)



# Correlation-based filtering

correlation_matrix = df.corr()



print(correlation_matrix)

from sklearn.model_selection import train_test_spl
X = df.drop('Purchased', axis=1) y = df['Purchased']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

Program 3

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

Screenshot:



Date: 18/03/2025

LAB - III

1) $a_0 = -5 \quad a_1 = 0.8$

(i) Logistic regression equation

$\rightarrow P(x) = \frac{1}{1+e^{-(a_0+a_1x)}} = \frac{1}{1+e^{-(-5+0.8x)}}$

(ii) Calculate the probability that a Student who Studies for 7 hours will pass

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

(iii) Determine the predicted class (Pass / fail) for the student based on threshold of 0.5

$\rightarrow P(x) = 0.6457$

$P(x) = 0.5$

Thus $y = 1$ (Pass)

$y = \begin{cases} 1 & \text{if } P(x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

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

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

$\text{Softmax}(z_1) = \frac{e^2}{e^2+e^1+e^0} = \frac{7.38905}{11.107337} = 0.6652$

$\text{softmax}(z_2) = \frac{e^1}{e^2+e^1+e^0} = \frac{2.71828}{11.107337} = 0.244$

$\text{Softmax}(z_3) = \frac{e^0}{\quad} = \frac{1.0}{11.107337} = 0.091$

The probabilities of the 3 classes are approximately 66.5%, 24.4%, 19.1%.

18.03.2022

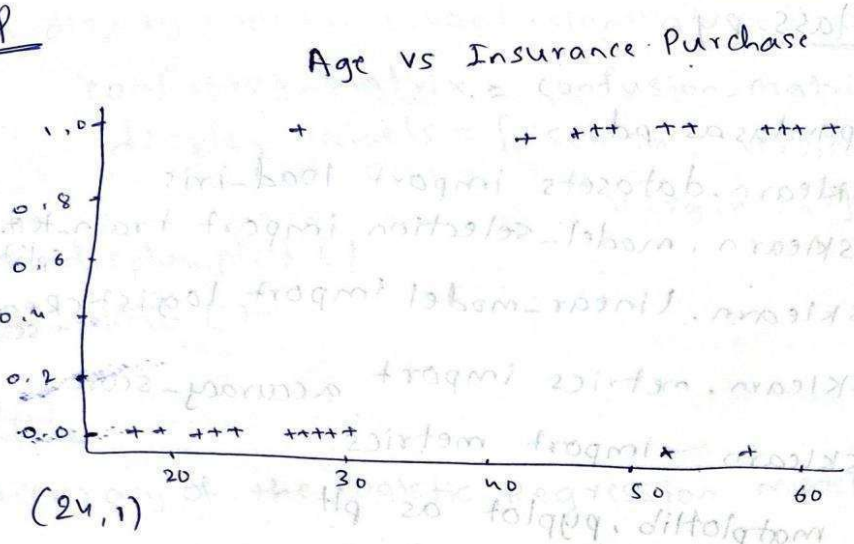
Logistic Regression (lab - executed codes)

*Binary.py

```
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)
```

Op



(24, 1)

	age
7	60
5	56
18	19
7	1
5	1
18	0

Name: bought_insurance

Model Accuracy: 1.0

Predicted Probilities: [[0.06470655 0.93529345]
[0.10327333 0.89672667]
[0.92775258 0.07224742]

Prediction for age 60: [1]

model coefficient: [[0.1274065]]

model Intercept: [-4.97339194]

Prediction for age 35: 0.3709834769552775

*Multiclass.py

```
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_sta-
                _te=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 Multinominal Logistic Regress
    -ion 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 Logistic Regression model: 0.95

Confusion matrix

True class \ Predicted class	mammal	bird	Reptile	Fish	Amphibian	Bug	Invertebrat
mammal	12	0	0	0	0	0	
Bird	0	2	0	0	0	0	
Reptile	0	0	0	0	0	1	
fish	0	0	0	2	0	0	
Amphibian	0	0	0	0	3	0	
Bug Invertebrat	0	0	0	0	0	1	

Answering the Questions

1) For dataset file "HR_comma_sep.csv"

i) which variable did you identify as having a direct and clear impact on employee retention? why?

→ The variables with a direct & clear impact on employee retention are

- Satisfaction level
- Time Spent at the Company
- Promotion in the last 5 years
- Average monthly hrs
- work accident

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 L-R model is 75.9 %.

- This is reasonable accuracy but not perfect
- The model shows a relatively high no. of false negatives, indicating that some employees who left were incorrectly predicted to stay.

2) For 200 dataset

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

→ Preprocessing steps performed.

- Merged the 200 datasets using class_type as the key.
- Removed non-numeric columns before standardizing the features.
- Scaled the features using Standard Scalar for better model performance.
- Split the data into training & testing sets

(ii) Were there any missing or inconsistent values in the dataset? How did you handle them?

→ No missing or inconsistent values were found, so no special handling was necessary

(iii) What does the confusion matrix tell you about the performance of your model?

→ The model achieved 93.5% accuracy, showing good performance

(iv) which class type were most frequently misclassified? why do you think this happened?

→ The class is Class 3, where all samples were incorrectly classified.

- This may have occured due to the small no. of samples, making it difficult for the model to learn its characteristics
- Class 7 was also partially misclassified, likely due to overlapping features with Class 6.

Code:

```
import pandas as pd

import numpy as np

from graphviz import Digraph

# Calculate Entropy

def entropy(data):

    class_probabilities = data.iloc[:, -1].value_counts(normalize=True)

    return -np.sum(class_probabilities * np.log2(class_probabilities))


# Calculate Information Gain
def information_gain(data, feature):

    total_entropy = entropy(data)

    feature_values = data[feature].unique()

    weighted_entropy = 0

    for value in feature_values:

        subset = data[data[feature] == value]

        weighted_entropy += (len(subset) / len(data)) * entropy(subset)

    return total_entropy - weighted_entropy


# Find the best feature to split the data

def best_feature(data):

    features = data.columns[:-1]  # Exclude the target column

    gains = {feature: information_gain(data, feature) for feature in features}
    return max(gains, key=gains.get)

# Create the decision tree

def id3(data, features=None):

    if len(data.iloc[:, -1].unique()) == 1:  # All data points belong to the same class
```

```
        return data.iloc[:, -1].iloc[0]

    if len(features) == 0:  # No more features to split on
        return data.iloc[:, -1].mode()[0]

    best = best_feature(data)
    tree = {best: {}}

    new_features = features.copy()
    new_features.remove(best)

    for value in data[best].unique():
        subset = data[data[best] == value]
        tree[best][value] = id3(subset, new_features)

    return tree
# Function to classify new examples based on the decision tree
def classify(tree, example):
    if not isinstance(tree, dict):
        return tree
    feature = list(tree.keys())[0]
    value = example[feature]
    return classify(tree[feature][value], example)
# Function to visualize the decision tree using Graphviz
def create_tree_diagram(tree, dot=None, parent_name="Root", parent_value=""):
    if dot is None:
        dot = Digraph(format="png", engine="dot")
```

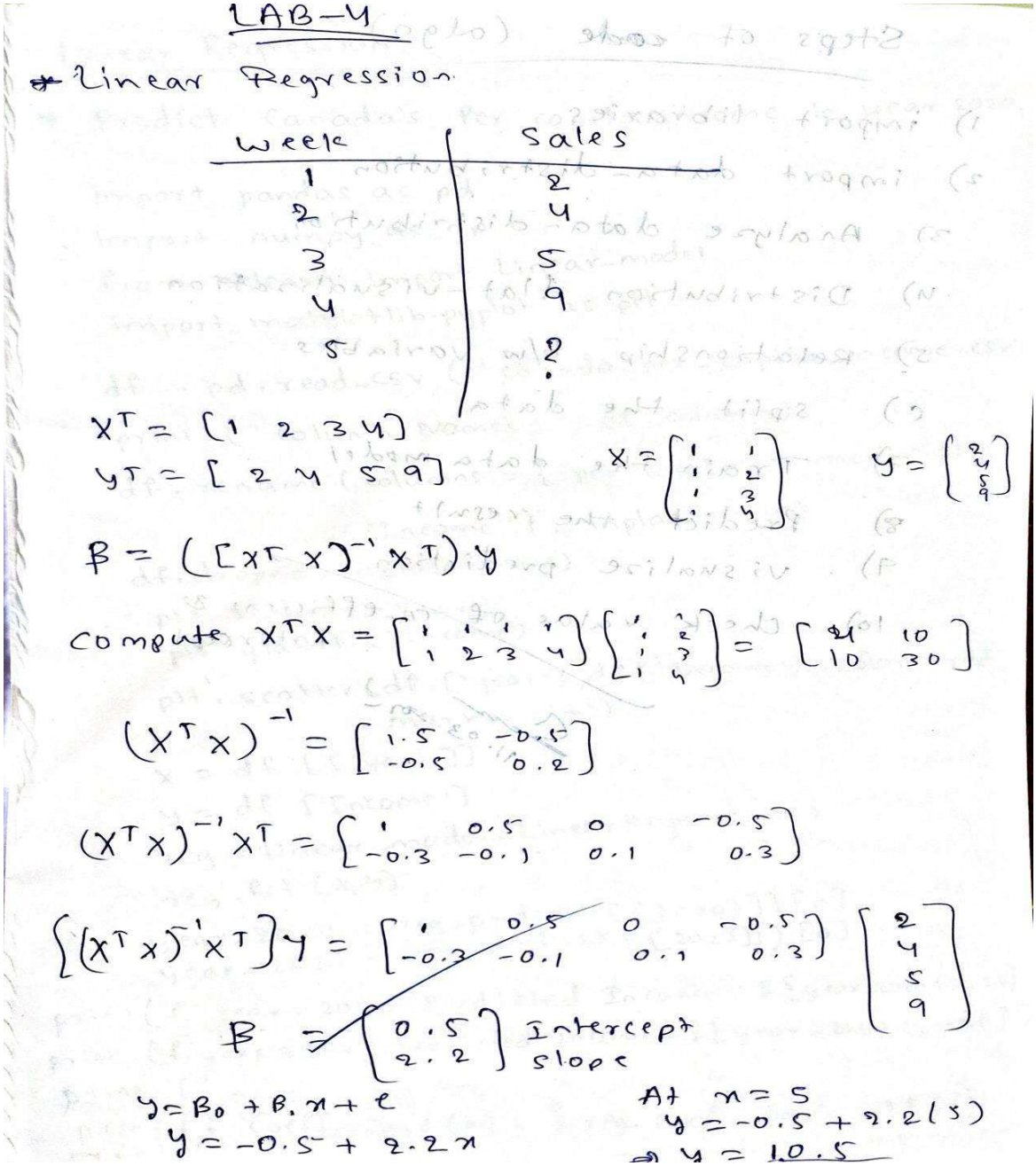
```
    if isinstance(tree, dict):  # Tree node
        for feature, branches in tree.items():
            feature_name = f"{parent_name}_{feature}"
            dot.node(feature_name, feature)
            dot.edge(parent_name, feature_name, label=parent_value)

            for value, subtree in branches.items():
                value_name = f"{feature_name}_{value}"
                dot.node(value_name, f"{feature}: {value}")
                dot.edge(feature_name, value_name, label=str(value))
                # Recurse for each subtree
                create_tree_diagram(subtree, dot, value_name, str(value))
    else:  # Leaf node
        dot.node(parent_name + "_class", f"Class: {tree}")
        dot.ede(parent_name, parent_name + "_class", label="Leaf")
    return dot

# Example usage
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'Low', 'High', 'Low', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong', 'Weak'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
```

```
})

# Train the decision tree
tree = id3(data, features=list(data.columns[:-1]))
print("Decision Tree:", tree)
# Classify a new example
example = {'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'Low', 'Wind': 'Strong'}
prediction = classify(tree, example)
print("Prediction for the example:", prediction)


# Visualize the decision tree
dot = create_tree_diagram(tree)
dot.render("decision_tree", view=True)  # This will generate and open the tree diagram
```

Program 4

Implement Linear and Multi-Linear Regression algorithm for appropriate dataset

Screenshot:



Steps of code (algo)

1) import libraries
2) import data-distribution
3) Analyze data-distribution
4) Distribution flat-visualization
5) Relationship b/w variables
6) split the data
7) Train the data model
8) Predict the result
9) visualize prediction
10) check value of co-efficient & intercept.

11.03.2021

Linear Regression.

* Predict Canada's Per capita income in year 2020.

```
import pandas as pd
import numpy as np
from sklearn import Linear-model
import matplotlib.pyplot as plt

df = pd.read-csv ('canada-per-capita-income.csv')
print ("Column Names:", df.columns)
df.rename (columns = {'per capita income (US$)':
        'Income'}, inplace =True)
df.dropna(inplace =True)
plt.xlabel ('Year')
plt.ylabel ('Income')
plt.scatter(df ['year'], df['Income'], color='red',
        marker='+')
x = df [['year']]
y = df ['Income']
reg = linear_model.LinearRegression()
reg.fit (x, y)
year-2020 = reg.predict ([[2020]]) [0]
year-2027 = reg.predict([[2027]]) [0]
print(f'year=2020, Predicted Income = ${year-2020:,.2f}')
print (f'year=2027, Predicted Income = ${year-2027:,.2f}')
print ("- - - - - - - - - - - - - - - - -")
print (f'Coefficient (m): {reg.coef_[0]:,.2f}')
print ("- - - - - - - - - - - - - - - - -")
```

```
print(f'Intercept(b): {reg.intercept_:.2f}')
print("-----------")
plt.plot(df['year'], reg.predict(x), color='blue')
plt.show()
```

output

```
Column Names: Index(['year', 'per capita Income (US $)'], dtype='object')
```

Year = 2020, Predicted Income = $41,288.69

Year = 2027, Predicted Income = $47,087.95

Coefficient (m): 828.47

Intercept (b): -1,632,210.76

Multiple Linear Regression

```
#Hiring
import pandas as pd
import numpy as np
from sklearn import linear_model
df = pd.read_csv('hiring.csv')
print(df.isnull().sum())
df['test-score(out of 10)'].fillna(df['test-score(out of 10)'].median(), inplace=True)
df['experience'] = df['experience'].astype(str)
```

```
df_encoded = pd.get_dummies(df, columns=['exper
                -ience'], drop_first=True)

X = df_encoded.drop('salary($)', axis='columns')
y = df_encoded['salary($)']

reg = linear_model.LinearRegression()
reg.fit(X,y)

def predict_salary(exp,test_score, interview_score):
    exp_col = f'experience_{exp}'
    input_data = {col : 0 for col in X.columns}
    if exp_col in input_data:
        input_data[exp_col] = 1
    input_data['test_score(out of 10)'] = test_score
    input_data['interview_score(out of 10)']
        = interview_score
    input_df = pd.DataFrame([input_data])
    predicted_salary = reg.predict(input_df)[0]
    return f'Predicted Salary: ${predicted_
                salary:,.2f}'

print(predict_salary('twelve', 10, 10))
print(predict_salary('two', 9.6))
```

output

```
experience                      2
test_score(out of 10)           1
interview_score(out of 10)      0
salary($)                       0
```

Predicted Salary : $67,803.92

Predicted Salary : $63,509.80

~~Questions~~

* Linear_regression_housing_area_price.py

Output

saving housing_area_price.csv

predicted price for 3300 sqr ft : 628715.753424657534

Coefficient (m) = 135.78767123287673

Intercept (b) : 180616.43835616432

Manual price calculation for 3300 sqr ft : 628715.753424657534

Predicted Price for 500 sqr ft : 859554.7945205479)

* multiple_lr_homeprice.py

Output

Coefficients : [112.06244194 23388.88007794 -3231.71790863]

Intercept : 221323.00186540396

Predicted price for a home with 3000 sqr ft, 3 bedrooms, 40 years old : 498408.2515803067

Manual Price calculation : 498408.25157403

Linear - Regression (Salary.csv)

output

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

YearsExperience	2
Salary	0
Years Experience	0
Salary	0

YearsExperience = 12, Predicted Income = $139,980.89

Coefficient (m) = 9,398.64

Intercept (b) = 27,197.20

Multiple - Regression (1000-companies.csv)

output

Predicted Profit : $510,570.99

Questions:

1) Considering 3 data .csv files, did you perform any data preprocessing steps? If yes, why?

⇒ • canada_per_capita_income.csv → NO encoding or scaling was needed. Checked for missing values & handled if any were found.

Salary.csv → Verified for missing values, handled if present.

hiring → no scaling & encoding was needed, no encoding was needed.

2) For canada per capita income did you visualize the regression line along with the data points? What does the plot tell you about the relationship blw you & per capita income

→ • A regression line was plotted along with the data points
The plot indicated a +ve linear relationship.

3) For hiring, what is the predicted salary for a candidate with 12 yrs of exp. 10 test score & 10 interview score?
→ $82000.00

4) For 1000 companies, did you encode categorical variables? If yes, how? Did you scale the features? If yes, why?

→ categorical encoding → The 'state' variable was one-hot encoded to convert categorical to numerical

→ Feature scaling → If features had varying magnitudes, scaling was applied to improve model efficiency.

Code:

Linear Regression

```
import pandas as pd

df = pd.read_csv("/content/tvmarketing.csv")

df


# Visualise the relationship between the features and the response using scatterplots

df.plot(x='TV',y='Sales',kind='scatter')


from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(df['TV'], df['Sales'], test_size=0.2, random_state=42)

from sklearn.linear_model import LinearRegression model = LinearRegression()
model.fit(x_train.values.reshape(-1, 1), y_train) y_train
model.coef_

model.intercept_
```

MultiLinearRegression

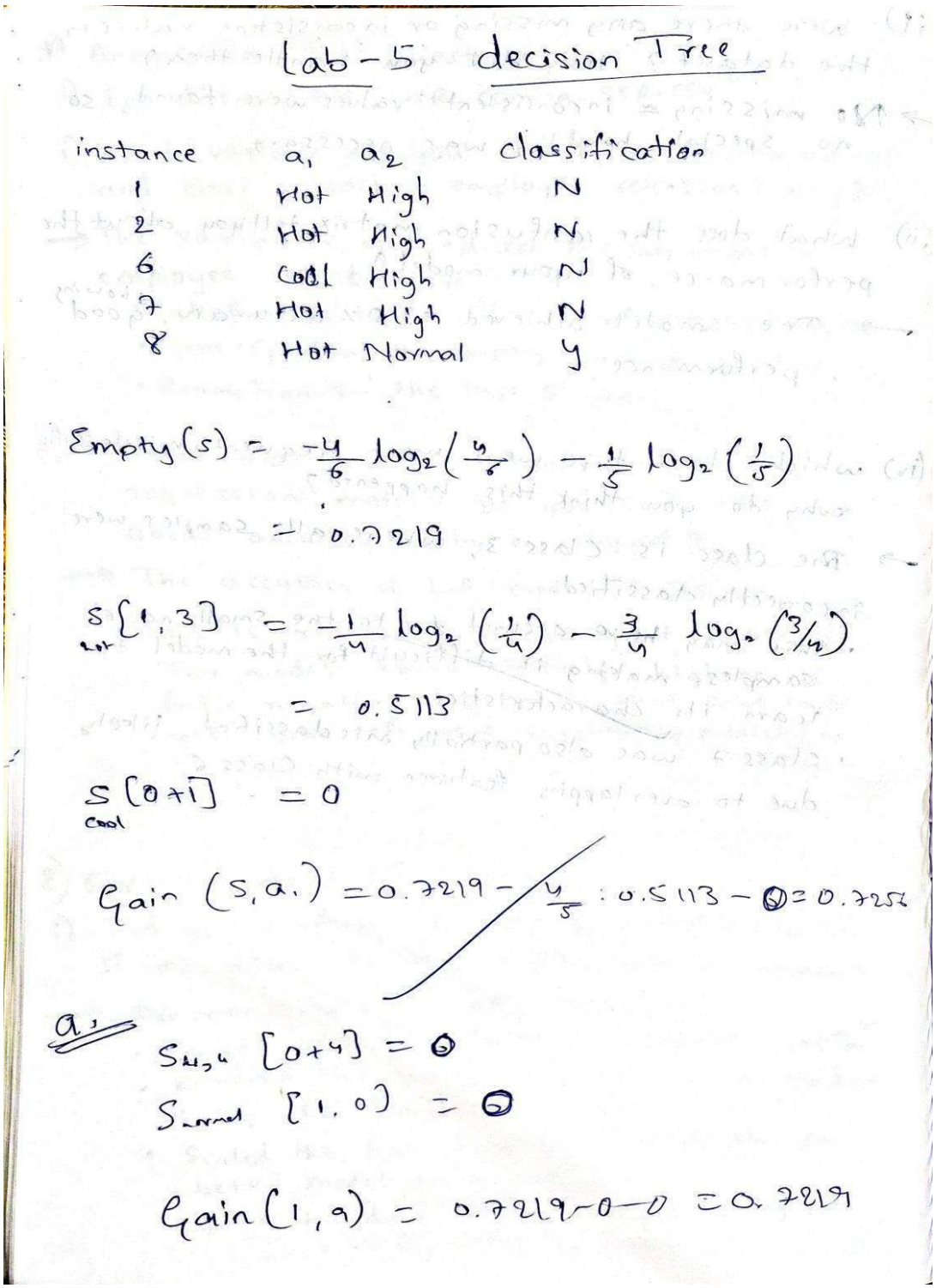
```
import pandas as pd

# Step 2 : import data

house = pd.read_csv('https://github.com/YBIFoundation/Dataset/raw/main/Boston.csv')

# display first 5 rows house.head()


y = house['MEDV']
```

```
X = house.drop(['MEDV'],axis=1)

# Step 4 : train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.7, random_state=2529)

# Step 5 : select model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
# Step 6 : train or fit model
model.fit(X_train,y_train)
model.intercept_

model.coef_
```

Program 5

Build Logistic Regression Model for a given dataset

Screenshot:



Lab-5 decision Tree

instance	a_1	a_2	classification
1	Hot	High	N
2	Hot	High	N
6	Cool	High	N
7	Hot	High	N
8	Hot	Normal	Y

$Empty(S) = -\frac{4}{6}\log_2\left(\frac{4}{6}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right)$

$= 0.7219$

$S_{Hot}[1,3] = -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right)$

$= 0.5113$

$S_{Cool}[0+1] = 0$

$Gain(S, a_1) = 0.7219 - \frac{4}{5} : 0.5113 - 0 = 0.7258$

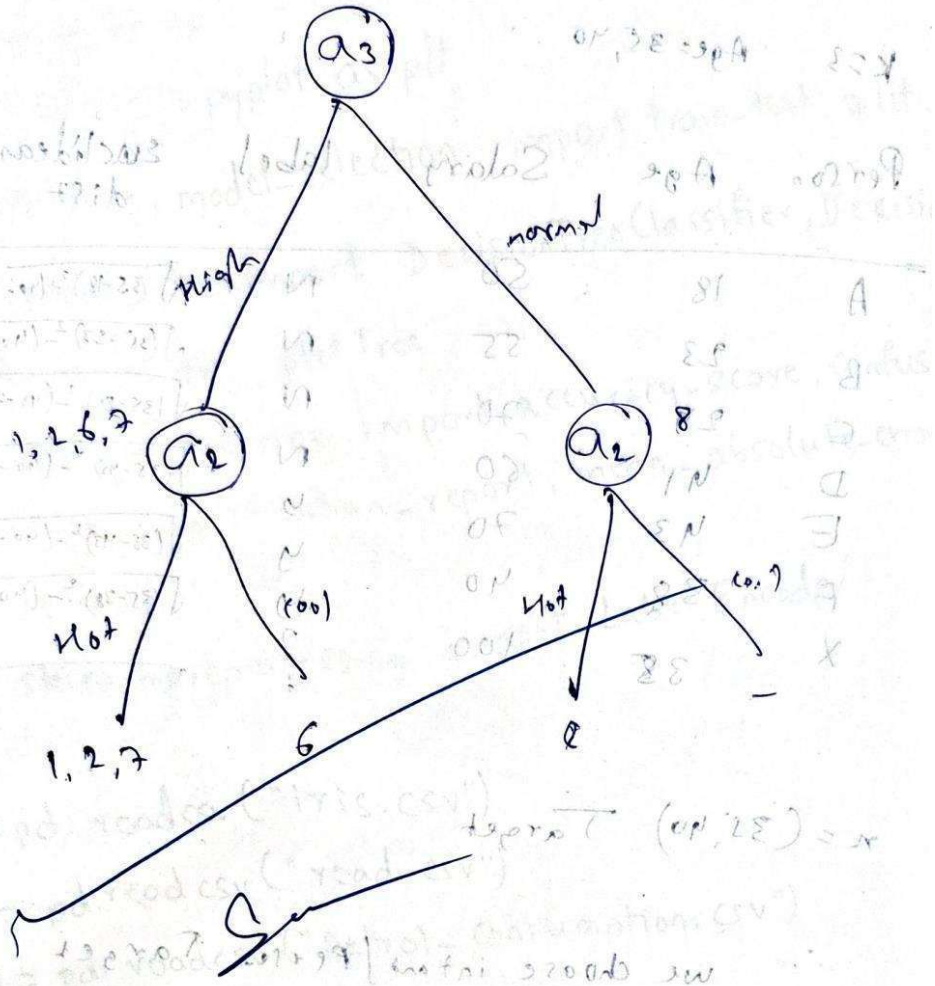
a_1

$S_{High}[0+4] = 0$

$S_{Normal}[1,0] = 0$

$Gain(1, a_2) = 0.7219 - 0 - 0 = 0.7219$

(i) a_3 is root node ∵ Since $\ell_4(S_0, a_3)$ is higher



lab-5

```
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, DecisionTree Regressioner, plotTree
from sklearn.metrics import accuracy_score, confusion matrix, classification_report, mean_absolute_error, mean_squared_error.
from sklearn.preprocessing import LabelEncoder

iris = pd.readcsv("iris.csv")
drug = pd.read csv("read.csv")
petrol = pd.readcsv("petrol_consumption.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("Accuracy", accuracy_score(y_test, y_pred))
X_drug = drug.iloc[:, :-1]
y_drug = drug.iloc[:, :-1]
le = labelEncoder().
```

```
for col in x_drug.select_dtypes(includes=['object']).
  columns:
      X_drug[col] = le.fit_transform(X_drug[col])

X_train, x_test, y_train, y_test = train_test_split(
    x_drug, y_drug, test_size=0.2, random=42)

dtc = Decision tree classifier()
dtc.fit(x_train, y_train)
 y_pred = dtc.predict(x_test)
print("Accuracy", accuracy_score(y_test, y_pred)

 X_petrol = petrol.iloc[:,:-1]
 y_petrol = petrol.iloc[:,-1]

 X_train, x_test, y_train, y_test = train_test_split
 [x_petrol, y_petrol, test_size=0.2, random=42)

 dtr = DecisionTree regressor()
 dtr.fit(x_train, y_train)
print[("Mean Absolute Error:", Mean_Absolute_Error
      (y_test, y_pred)
print("mean squared Error:", Mean_squared_Error
      (y_test, y_pred))
```

output

```
Iris dataset
Accuracy : 1.0
Drug dataset
Accuracy : 1.0
petrol Dataset

Mean Absolute Error: 91.6
mean Squared Error : 17609.8
Root Mean Squared Error : 132701 9
```

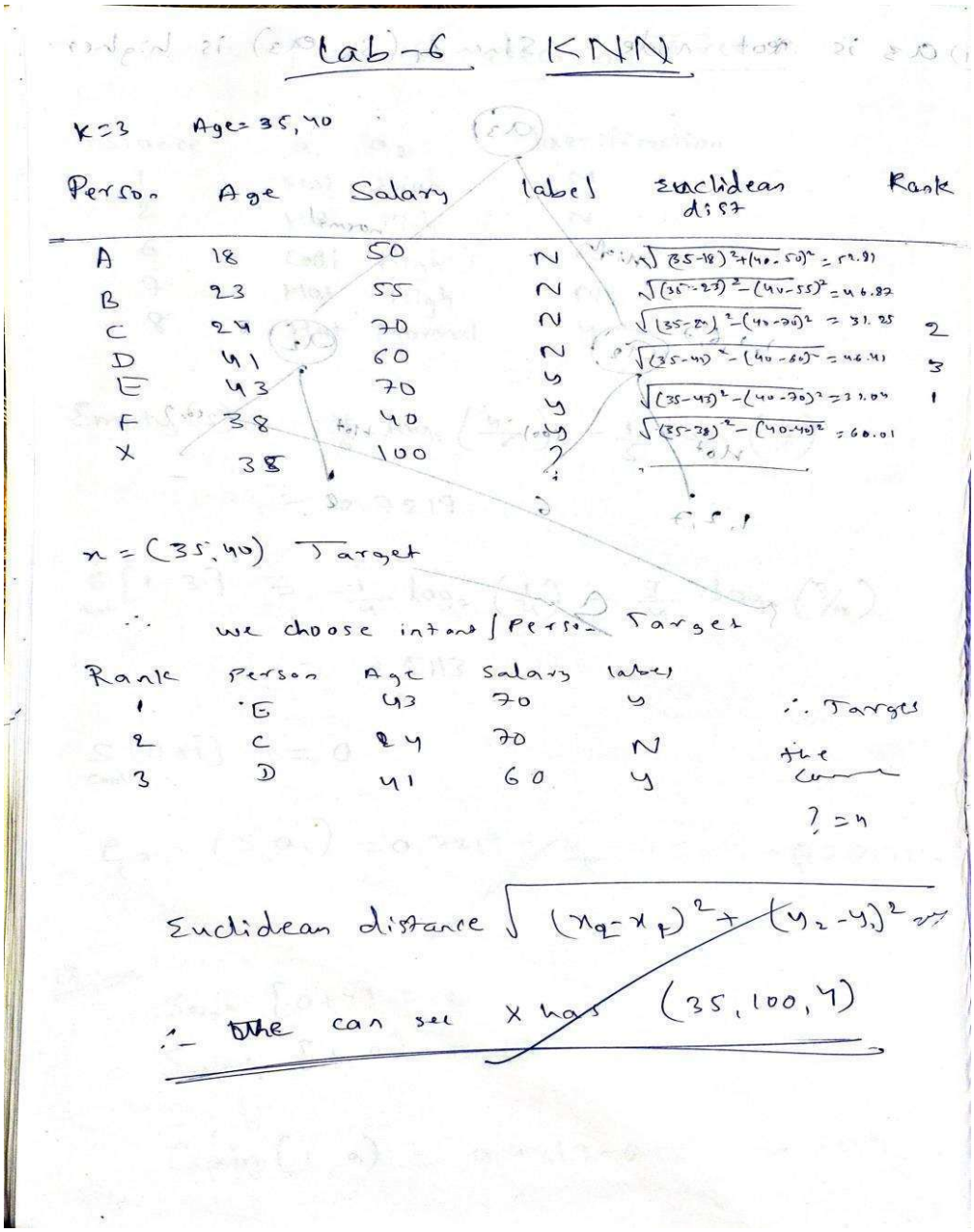
Code:

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# Load sample dataset (binary classification - Iris with only 2 classes)
iris = load_iris()
X = iris.data[iris.target != 2]
y = iris.target[iris.target != 2]

# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)


# Predict and evaluate
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Program 6

Build KNN Classification model for a given dataset

Screenshot:



lab 6-6

* imports are same as previous program.

```
heart_df = pd.read_csv('heart.csv')
print(heart_df.head())
X = heart_df.drop(columns=['target'])
y = heart_df['target']

X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size=0.2, random_state=42)
Scaler = StandardScaler()
X_train_scaled = Scaler.fit_transform(X_train)
X_test_scaled = Scaler.transform(X_test)

k_values = range(1,21)
best_k = 1
best_score = 0
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    score = accuracy_score(y_test, ypred)
if score > best_score:
        best_score = score
        best_k = k
print(f"Best k value: {best_k} with accuracy:
        {best_score*100:.25}%")
```

```
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(X_train_scaled, y_train)
y_pred_best = best_knn.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_predict_best)
print(f"Accuracy on test dates: {accuracy*100:.2f}%")
cm = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap
="Blues", xticklabels=best_knn_classes,
yticklabels=best_knn_classes)
```

Output

best k value 7 with accuracy = 91.80%

Accuracy on test data: 91.80%

Code:

KNN

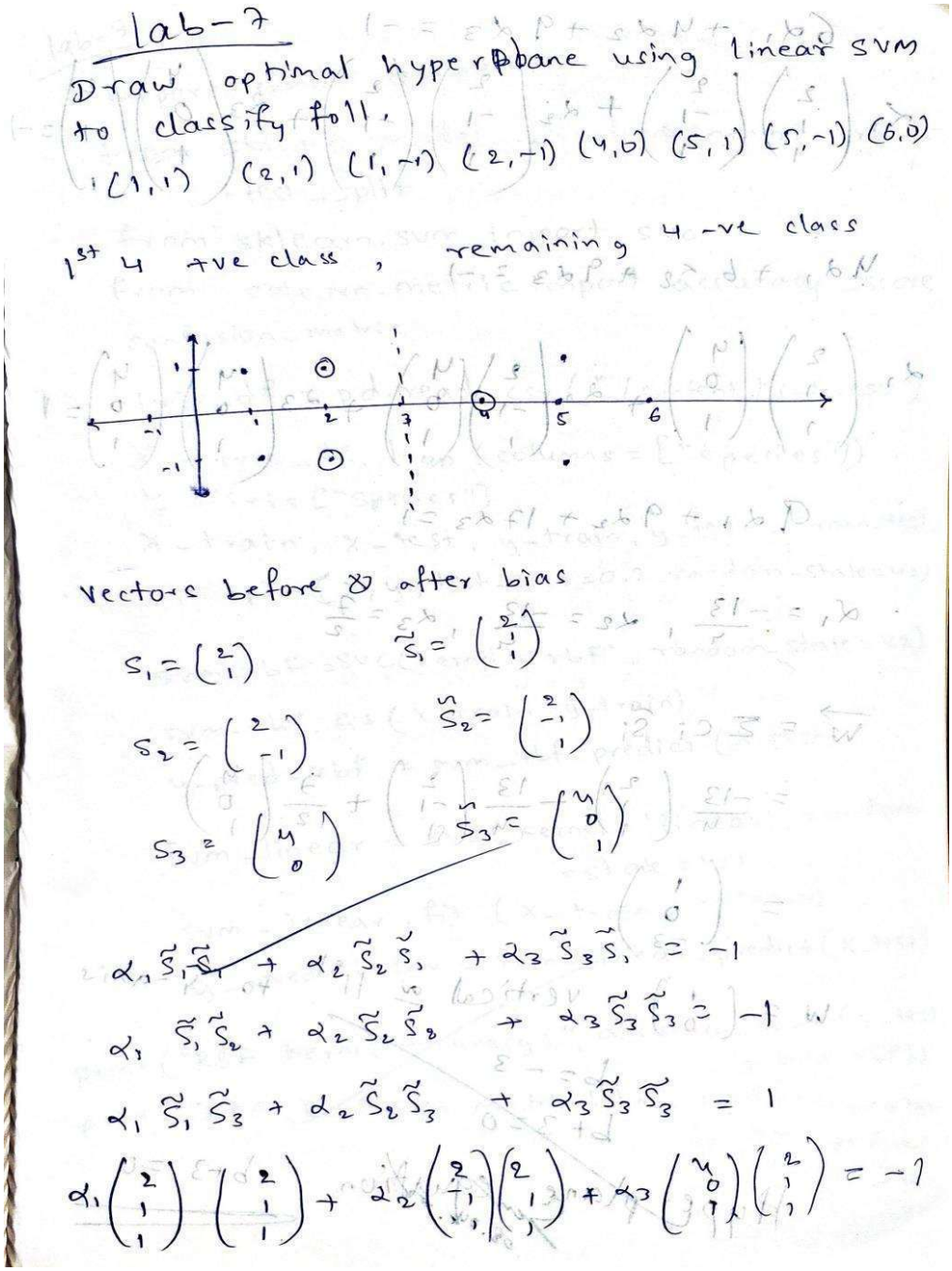
```
import numpy as np
from collections import Counter

class KNN:
def __init__(self, k=3): self.k = k
    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)

    def euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2) ** 2))


    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        # Compute distances to all training points
        distances = [self.euclidean_distance(x, x_train) for x_train in self.X_train]

        # Get indices of k nearest neighbors
        k_indices = np.argsort(distances)[:self.k]


        # Get the labels of those neighbors
        k_nearest_labels = [self.y_train[i] for i in k_indices]
```

```
        # Return the most common label
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

# Sample dataset (like a mini version of Iris)
X_train = [[1, 2], [2, 3], [3, 1], [6, 5], [7, 7], [8, 6]]
y_train = [0, 0, 0, 1, 1, 1]

# Test data
X_test = [[5, 5], [1, 1]]

# Using the KNN modelh
knn = KNN(k=3)
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)
print("Predictions:", predictions)
```

Program 7

Build Support vector machine model for a given dataset

Screenshot:



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

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

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

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

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

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

$$\overrightarrow{w} = \Sigma c_i \tilde{s}_i$$

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

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

$w = \begin{pmatrix}1\\0\end{pmatrix}$ vertical $\perp$ to 4-axis

$b = -3$

$b + 3 = 0$

Hyperplane equation $b + 3 = 0$

08.04.

lab-7

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import svc
from sklearn.metric import accuracy_score
confusion_matrix
iris_df = pd.read_csv("/content/iris.csv")
X = iris_df.drop(columns=["species"])
y = iris["Species"]
X_train, X_test, y_train, y_test = train_test_
    split(X, y, test_size=0.2, random_state=42)

svm_rbf = svc(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)

svm_linear = svc(kernel='linear', random_
                 state=42)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)

print("RBF kernel Accuracy:", accuracy_score(y_test,
                                    y_pred_rbf))
print("RBF Confusion matrix:\n", confusion_matrix
                    (y_test, y_pred_rbf))
```

```
print("\nLinear kernel Accuracy:", accuracy_score(
                    y_test, y_pred_linear))
print("Linear Confusion Matrix:\n", confusion_
                    matrix(y_test, y_pred_linear))
```

O/P

RBF kernel Accuracy: 1.0

RBF Confusion Matrix:

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

Linear kernel Accuracy: 1.0

Linear Confusion Matrix:

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

Q Answer

* for "Iris.csv" dataset

1) which is the accuracy score of the classifier using the linear kernel & RBF kernel

- linear → 1.0
- RBF → 1.0

} Accuracy

2) Better kernel :→ Both performed equally well with 100% accuracy. This is likely bz the Iris dataset is relatively small & well seperated, making it easy for both kernels to classify correctly

3) which kernel (RBF or linear) gave better performance? why do you think this is the case?

* for "letter_recognition.csv" dataset

1) Present & Interpret the confusion matrix, Are there any specific letters that are frequently confused with others?

2) what is the AUC (Area under the Curve) score, and how does it reflect the model performance?

3) How does the performance of the svm model of this dataset compare to its performance on the IRIS dataset?

Ans

1) Accuracy : 0.9315 (93.15%)

Top 5 most confused letters are

- P → 53
- B → 52
- M → 49
- D → 45
- U → 44

These letters might have similar features, leading to confusion.

② AUC Score → 0.9981

Almost perfect classification performance

③ Comparision to Iris dataset

→ SVM performed flawlessly on Iris, but had minor misclassifications on the letter dataset

→ The letter dataset is much larger & has most complex patterns, making classification harder than the simple 3-class Iris dataset.

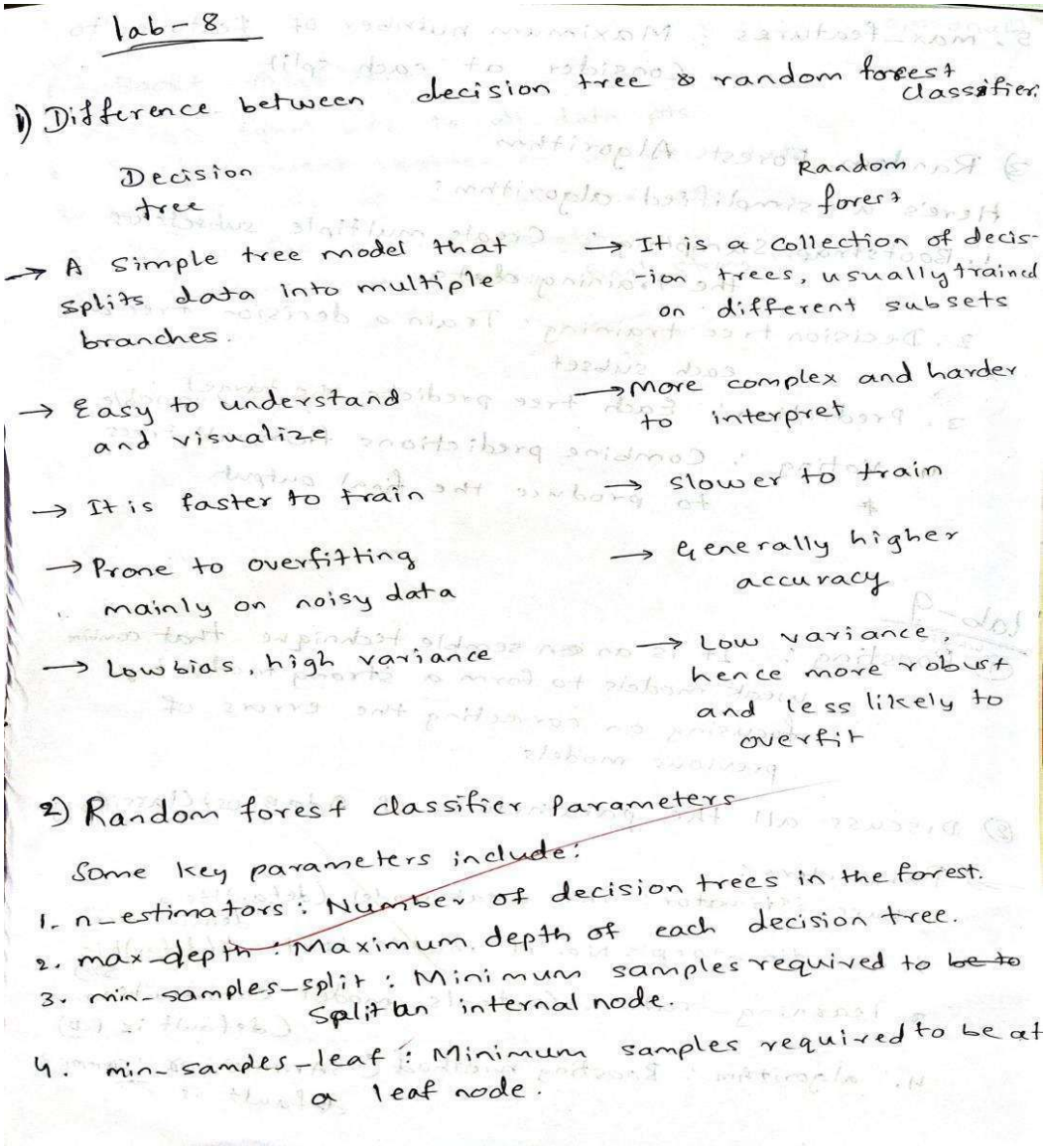
Code:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA


# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
# For binary classification (class 0 vs 1)
X = X[y != 2]
y = y[y != 2]
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train SVM
clf = SVC(kernel='linear')  # Try 'rbf', 'poly', etc.
clf.fit(X_train, y_train)

# Accuracy
print("Test Accuracy:", clf.score(X_test, y_test))
```

Program 8

Implement Random Forest ensemble method on a given dataset

Screenshot:



lab - 8

1) Difference between decision tree & random forest classifier

Decision tree	Random forest
→ A simple tree model that splits data into multiple branches.	→ It is a collection of decision trees, usually trained on different subsets
→ Easy to understand and visualize	→ More complex and harder to interpret
→ It is faster to train	→ Slower to train
→ Prone to overfitting mainly on noisy data	→ Generally higher accuracy
→ Low bias, high variance	→ Low variance, hence more robust and less likely to overfit

2) Random forest classifier Parameters

Some key parameters include:

1. n_estimators : Number of decision trees in the forest.
2. max_depth : Maximum depth of each decision tree.
3. min_samples_split : Minimum samples required to ~~be to~~ Split an internal node.
4. min_samples_leaf : Minimum samples required to be at a leaf node.

5. max_features : Maximum number of features to Consider at each split.

③ Random Forest Algorithm

Here's a simplified algorithm:

1. Bootstrap Sampling : Create multiple subsets of the training data.
2. Decision tree training : Train a decision tree on each subset
3. Prediction : Each tree predicts the target variable.
4. Voting : Combine predictions from all trees to produce the final output

lab-9

① what is Boosting : It is an ensemble technique that combine weak models to form a strong model by focusing on correcting the errors of previous models

② Discuss all the parameters of AdaBoostClassifier()

→ Parameters :

1. base_estimator : The weak model (default is a decision tree)
2. n_estimators : No. of models to train (default is 50)
3. learning_rate : Controls model contribution (default is 1.0)
4. algorithm : Boosting method ('SAMME' or 'SAMME.R' default is ↗

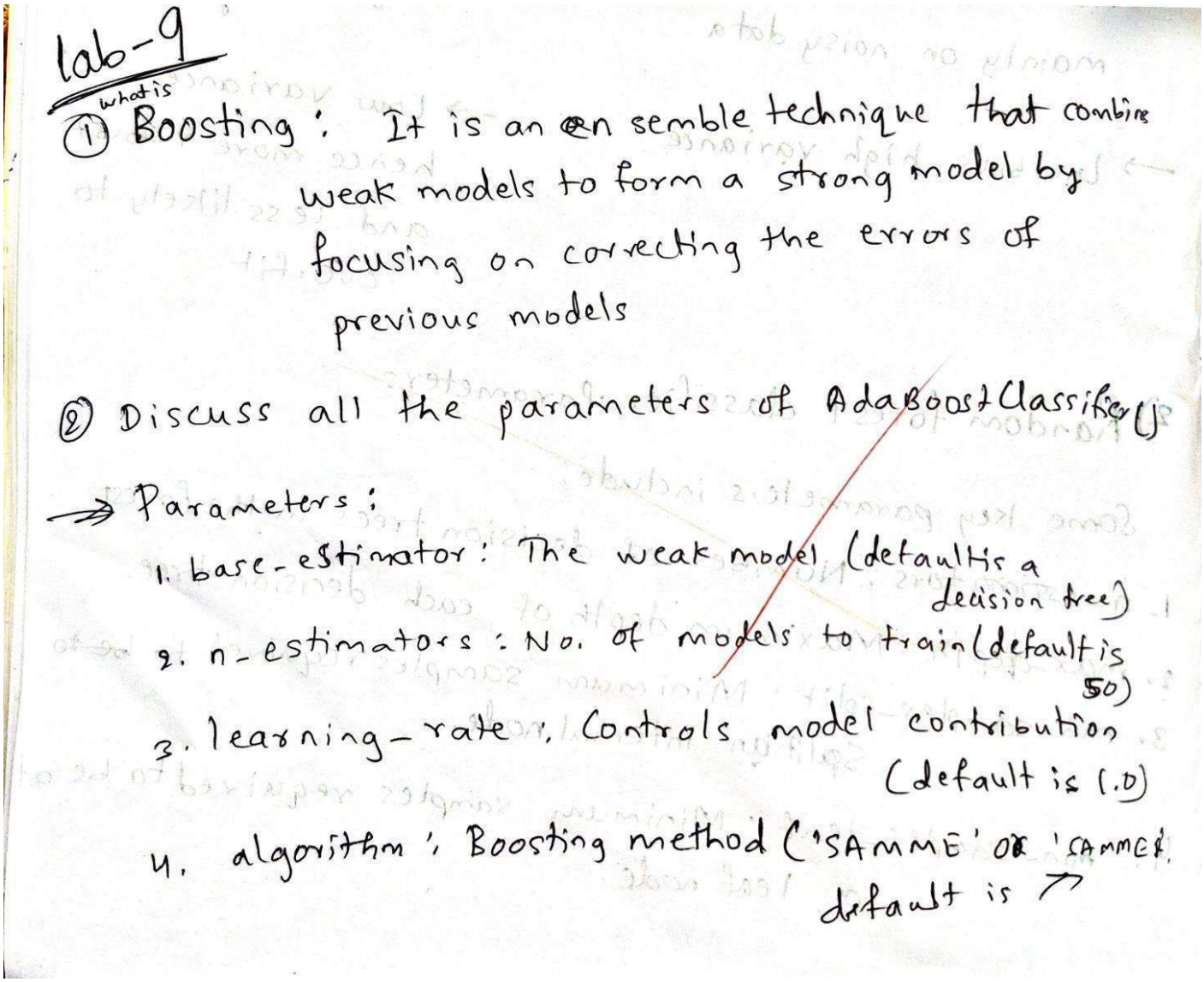
Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# Load sample dataset
iris = load_iris()
X, y = iris.data, iris.target


# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:



5. random_state : Ensures reproducibility (optional)

(3) write AdaBoost Algorithm:

1. Assign equal wts. to all data pts.
2. Train a weak learner on weighted data
3. Increase wts. of misclassified pts.
4. Repeat for several iterations
5. Combine models into a final strong learner.

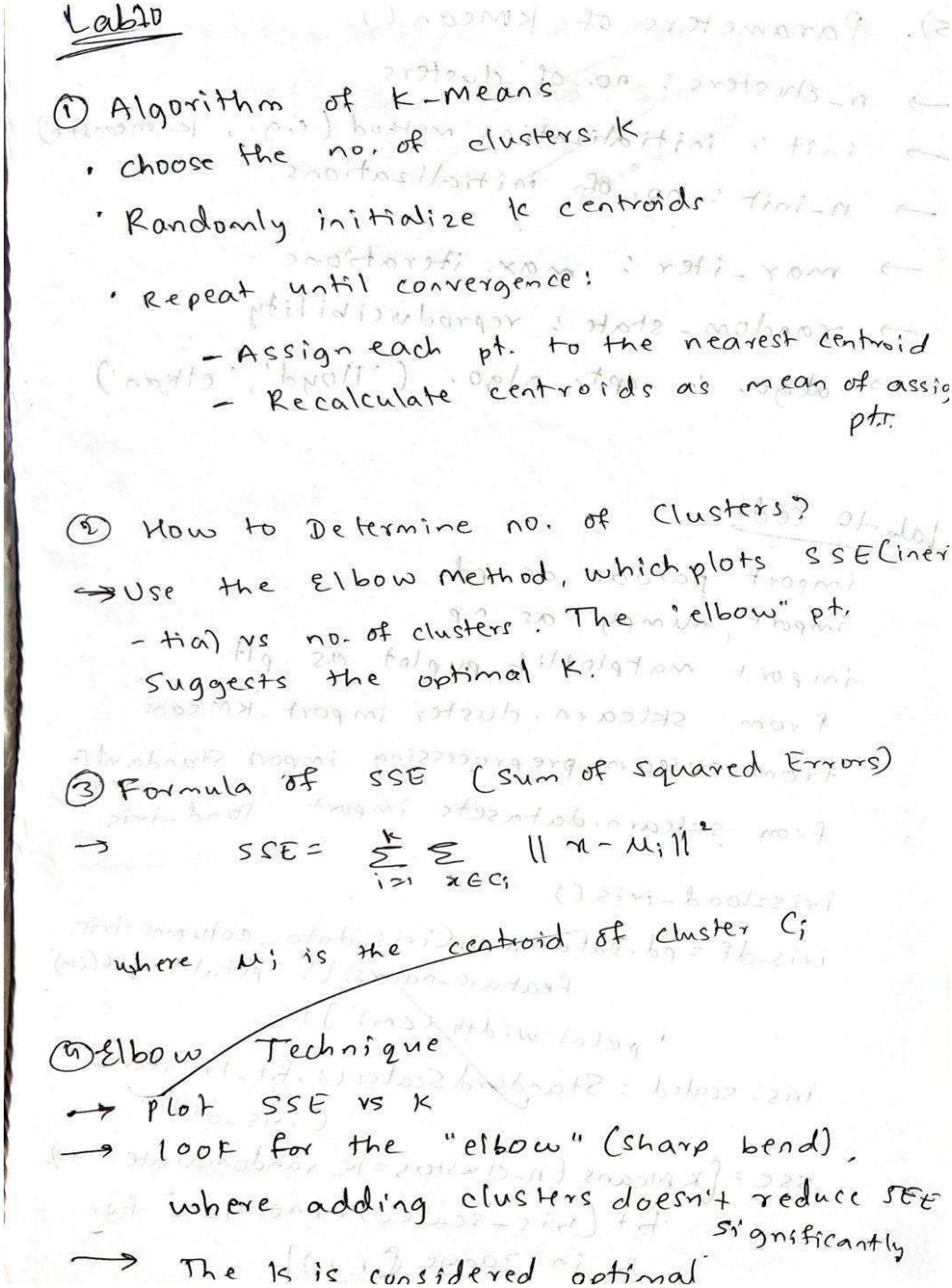
Code:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# For AdaBoost, we'll use binary classification
# Convert to binary (setosa vs. not-setosa)
y = (y == 0).astype(int)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train AdaBoost
model = AdaBoostClassifier(n_estimators=50, learning_rate=1.0, random_state=42)
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
print("AdaBoost Accuracy (sklearn):", accuracy_score(y_test, y_pred))
```

Program 10

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

Screenshot:



Lab10

① Algorithm of K-means
- Choose the no. of clusters K
- Randomly initialize k centroids
- Repeat until convergence:
  - Assign each pt. to the nearest centroid
  - Recalculate centroids as mean of assig pts.

② How to Determine no. of Clusters?
→ Use the Elbow method, which plots SSE(inertia) vs no. of clusters. The "elbow" pt. suggests the optimal K.

③ Formula of SSE (Sum of squared Errors)
→ $SSE = \sum_{i=1}^{K} \sum_{x \in C_i} \| x - \mu_i \|^2$

where μ_i is the centroid of cluster C_i

④ Elbow Technique
→ Plot SSE vs K
→ look for the "elbow" (sharp bend), where adding clusters doesn't reduce SSE significantly
→ The K is considered optimal

5). Parameters of KMean()

→ n_clusters : no. of clusters
→ init : initialization method (e.g., 'k-means++')
→ n_init : no. of initializations
→ max_iter : max. iterations
→ random_state : reproducibility
→ algo. : opt. algo. ('lloyd', 'elkan')

Lab-10 code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

iris=load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.
                       feature_names)[['petal length(cm)',
'petal width (cm)']]
iris_scaled = StandardScaler().fit_transform
                       (iris_df)
sse=[KMeans(n_clusters=k, random_state=42).
     fit(iris_scaled).inertia_ for
     k in range(1,11)]
```

```
plt.plot(range(1,11), sse, marker='o')
plt.title("Elbow Method : SSE")
```

29.04.

Code:

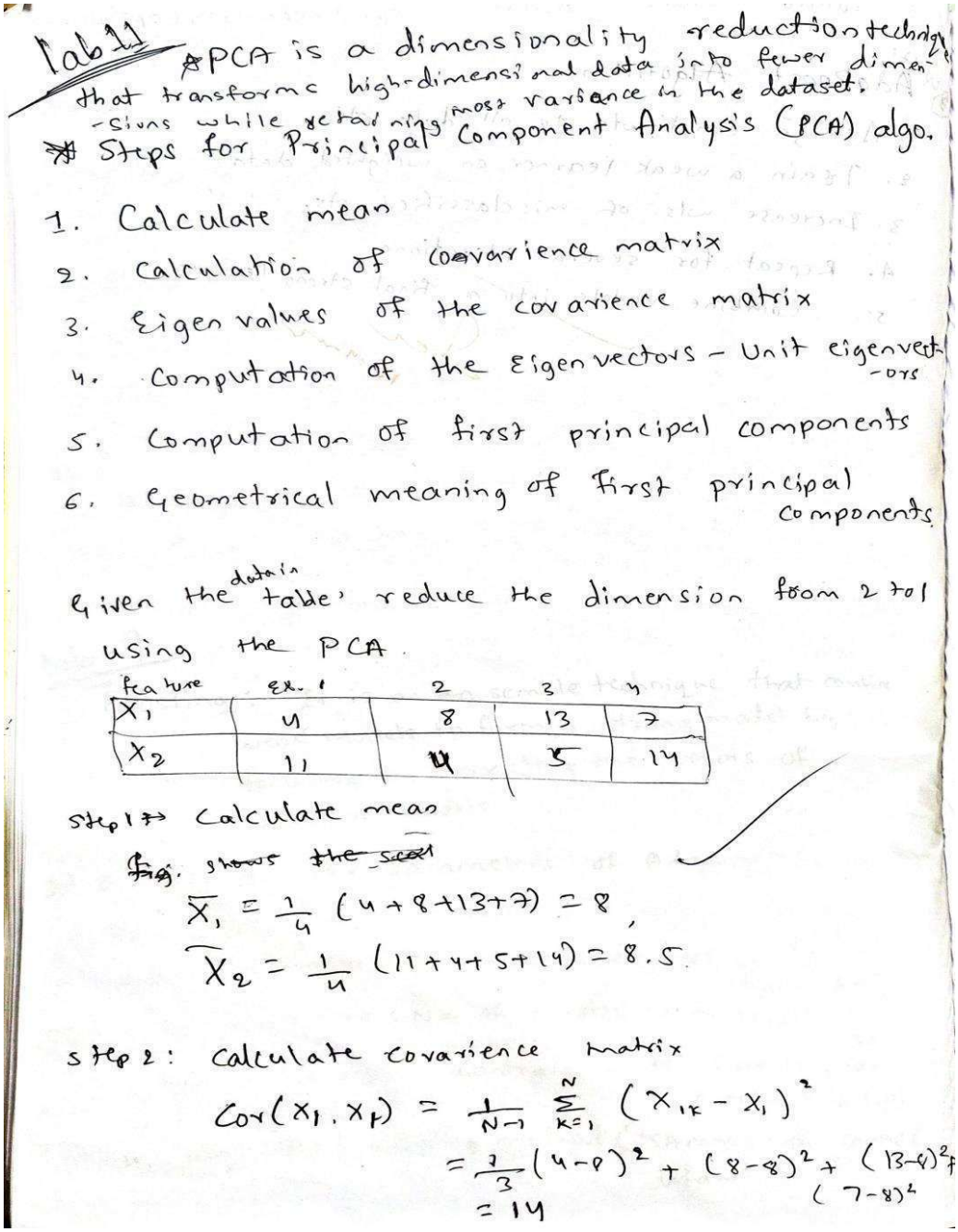
```
import pandas as pd

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris # Import load_iris


# Step 1: Load the Iris dataset directly

iris = load_iris()

# Create a DataFrame from the data and target

data = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Add the target column for potential reference, though not used for clustering

data['target'] = iris.target



# Step 2: Extract only numeric columns (or select required features)

# All features in the Iris dataset are numeric

X = data[iris.feature_names].values # Use the feature names to select columns


# Step 3: Apply KMeans

# Adjust n_clusters based on the expected number of clusters in your data (3 for Iris)

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10) # Added n_init to suppress future
warnings

data['Cluster'] = kmeans.fit_predict(X)


# Step 4: Plot clusters (for 2D data)
# Iris data has 4 features. We will plot the first two features for visualization.

if X.shape[1] >= 2:
```

```
    plt.scatter(X[:, 0], X[:, 1], c=data['Cluster'], cmap='viridis')
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color='red', marker='x', s=200)
    plt.title("K-Means Clustering of Iris Dataset")
    plt.xlabel(iris.feature_names[0])  # Label with actual feature name
    plt.ylabel(iris.feature_names[1])  # Label with actual feature name
    plt.show()
else:
    print("Cannot plot clustering results directly for data with less than 2 features.")
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:



lab 11

* PCA is a dimensionality reduction technique that transforms high-dimensional data into fewer dimensions while retaining most variance in the dataset.

** Steps for Principal Component Analysis (PCA) algo.

1. Calculate mean
2. Calculation of covarience matrix
3. Eigen values of the covarience matrix
4. Computation of the Eigen vectors - Unit eigenvectors
5. Computation of first principal components
6. Geometrical meaning of first principal components

Given the data in table, reduce the dimension from 2 to 1 using the PCA.

feature	Ex. 1	2	3	4
X_1	4	8	13	7
X_2	11	4	5	14

step 1 → Calculate mean

Fig. shows the scatter

$\bar{X_1} = \frac{1}{4}(4+8+13+7) = 8$

$\bar{X_2} = \frac{1}{4}(11+4+5+14) = 8.5$

step 2: Calculate covarience matrix

$Cov(X_1, X_1) = \frac{1}{N-1}\sum_{k=1}^{N}(X_{1k} - \bar{X_1})^2$

$= \frac{1}{3}(4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2$

$= 14$

$$Cov(X_1, X_2) = \frac{1}{N-1} \sum_{k=1}^{n} (X_{1k} - \bar{X}_1)(X_{2k} - \bar{X}_2)$$

$$= \frac{1}{3}\Big((4-8)(11-8.5) + (8-8)(4-8.5) + (13-8)(5-8.5) + (7-8)(14-8.5)\Big)$$

$$= -11$$

$$Cov(X_2, X_1) = Cov(X_1, X_2)$$

$$= -11$$

$$Cov(X_2, X_2) = \frac{1}{N-1} \sum_{k=1}^{N} (X_{2k} - \bar{X}_2)^2$$

$$= \frac{1}{3}\Big((11-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2\Big)$$

$$= 23$$

Eigenvalues of the covariance matrix

The cov. matrix is

$$S = \begin{bmatrix} Cov(X_1, X_1) & Cov(X_1, X_2) \\ Cov(X_2, X_1) & Cov(X_2, X_2) \end{bmatrix}$$

$$= \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$$

$\neq$

Step 3: Eigen values of cov. matrix

The char. eqn of cov matrix is

$$0 = \det(S - \lambda I)$$

$$= \begin{bmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{bmatrix}$$

$$= (14-\lambda)(23-\lambda) - (-11) \times (-11)$$

$$= \lambda^2 - 37\lambda + 201$$

giving the char. equ^n we get,

$$\lambda = \frac{1}{2}(37 \pm \sqrt{561})$$

$$= 30.3849,\ 6.6151$$

$$= \lambda_1, \lambda_2 \text{ (say)}$$

Step 4: Computation of eigenvectors

→ The eigenvector corr. to $\lambda_* = \lambda_1$ is a vector

$$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

satisfying the foll. equ^{n}s:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda_1 I) X$$

$$= \begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

$$= \begin{bmatrix} (14-\lambda_1) u_1 - 11 u_2 \\ -11 u_1 + (23-\lambda_1) u_2 \end{bmatrix}$$

this is equiv to following 2 equations:

$$(14-\lambda_1)u_1 - 11u_2 = 0$$
$$-11u_1 + (23-\lambda_1)u_2 = 0$$

using theory of system of linear eqns, we note that these eqns are not independent & sol^ns are given by.

$$\frac{u_1}{11} = \frac{u_2}{14-\lambda_1} = t,$$

that is,

$$u_1 = 11t, \quad u_2 = (14-\lambda_1)t,$$

where t is any real number

Taking $t=1$, we get an eigenvector corresp. to λ_1 as

$$U_1 = \begin{bmatrix} 11 \\ 14-\lambda_1 \end{bmatrix}$$

to find a unit eigenvector, we compute the length of X_1 which is given by

$$\|U_1\| = \sqrt{11^2 + (14-\lambda_1)^2}$$
$$= \sqrt{11^2 + (14-30.3849)^2}$$
$$= 19.7348$$

∴, a unit eigenvector corr. to λ_1 is

$$e_1 = \begin{bmatrix} 11 / \|U_1\| \\ (14-\lambda_1) / \|U_1\| \end{bmatrix}$$

$$= \begin{bmatrix} 11 / 19.7348 \\ (14 - 30.3849) / 19.7348 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$$

By carrying out similar computations, the unit eigenvector e_2 corr. to the eigenvalue $\lambda = \lambda_2$ can be shown to be $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

Step 5: Computation of I^{st} principal components

let,

$$\begin{bmatrix} X_{1k} \\ X_{2k} \end{bmatrix}$$

be the k^{th} sample in the above table (dataset). the I^{st} principal component of this ex. is given by (here "T" denotes the transpose of the matrix)

$$e_1^T \begin{bmatrix} X_{1k} - \bar{X}_1 \\ X_{2k} - \bar{X}_2 \end{bmatrix} = \begin{bmatrix} 0.5574 & -0.8303 \end{bmatrix} \begin{bmatrix} X_{1k} - \bar{X}_1 \\ X_{2k} - \bar{X}_2 \end{bmatrix}$$

$$= 0.5574 (X_{1k} - \bar{X}_1) - 0.8303 (X_{2k} - \bar{X}_2)$$

Code:

```
import pandas as pd

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt


# Load dataset

data = pd.read_csv("your_data.csv")  # Replace with your file

X = data.select_dtypes(include=['float64', 'int64'])


# Step 1: Standardize

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Step 2: Apply PCA

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)


# Print explained variance ratio

print("Explained variance ratio:", pca.explained_variance_ratio_)


# Visualize

plt.scatter(X_pca[:, 0], X_pca[:, 1], c='blue', alpha=0.5)

plt.title("PCA - 2D Projection")

plt.xlabel("Principal Component 1")
```

```
plt.ylabel("Principal  Component 2")

plt.show()
```

Choose Files Student.csv

- **Student.csv**(text/csv) - 41 bytes, last modified: 5/18/2025 - 100% done

```
Saving Student.csv to Student.csv
Uploaded file: Student.csv
```

	Hours_Studied	Passes
0	1	0
1	2	0
2	3	1
3	4	1
4	5	1

```
Numerical features found: ['Hours_Studied', 'Passes']
```