VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Abhinav Raghu (1BM22CS005)

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 **Abhinav Raghu (1BM22CS005),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

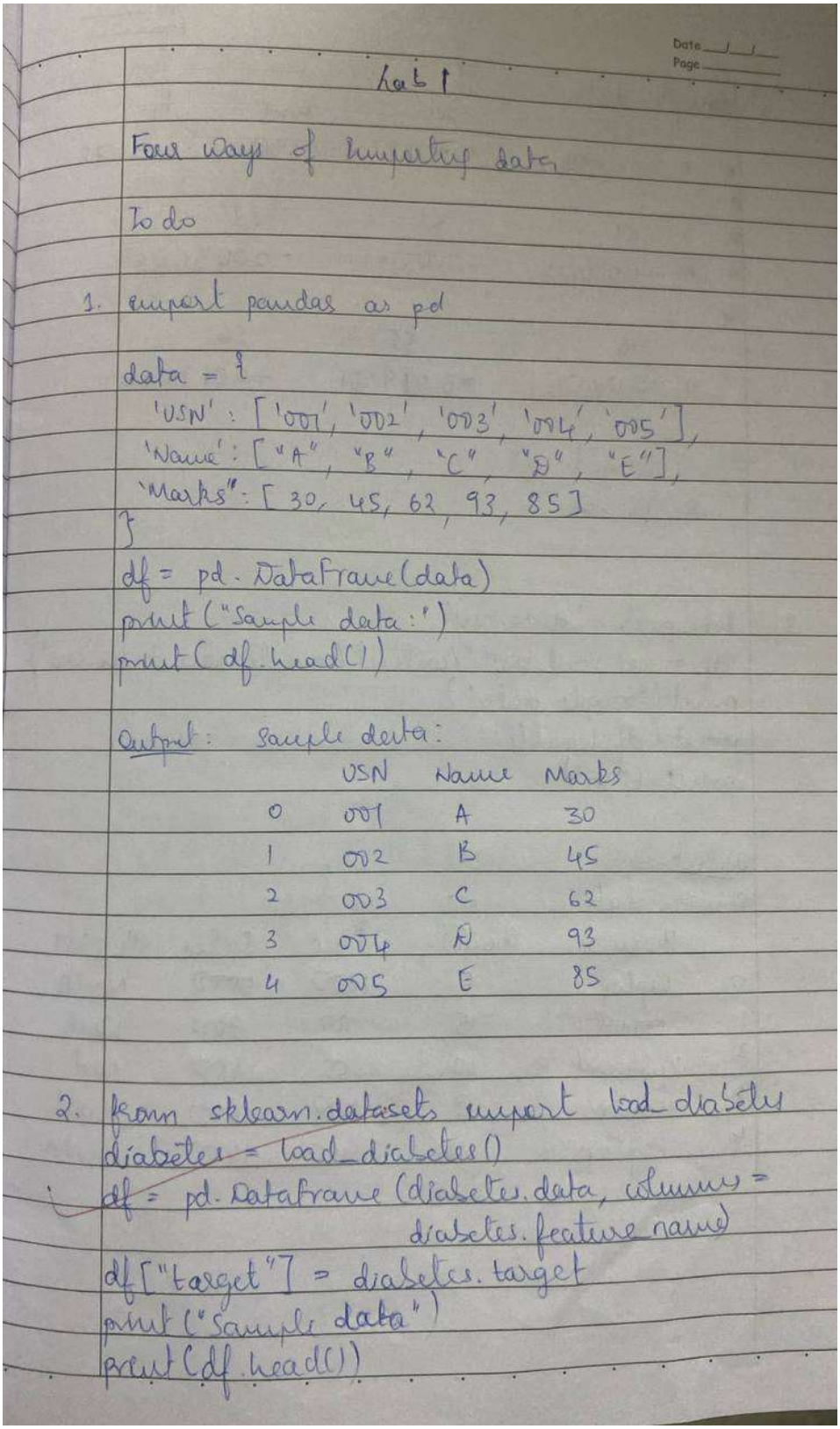
Dr. Seema Patil Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link:
https://github.com/AbR04/6thSem-ML-Lab

Program 1

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



Lab 1

Four ways of importing data

To do

1. import pandas as pd

```
data = {
    'USN': ['001', '002', '003', '004', '005'],
    'Name': ["A", "B", "C", "D", "E"],
    'Marks': [30, 45, 62, 93, 85]
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

Output: Sample data:

	USN	Name	Marks
0	001	A	30
1	002	B	45
2	003	C	62
3	004	D	93
4	005	E	85

2. from sklearn.datasets import load_diabetes

```
diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns =
                  diabetes.feature_name)
df["target"] = diabetes.target
print("Sample data")
print(df.head())
```

Sample data:

	age	sex	bmi	bp
0	0.038076	0.050680	0.061696	0.021872

	s1	s2	s3
0	-0.044223	-0.034821	-0.043401

	s4	s5	s6
0	-0.002592	0.019907	-0.017646

	target
0	151.0

3.
```
file_path = 'data.csv'
df = pd.read_csv("/content/sample_sales_data.csv")
print("Sample data:")
print(df.head())
print("\n")
```

Output:

Sample data

	Product	Quantity	Price	Sales	Region
0	Laptop	5	1000	5000	North
1	Mouse	15	20	300	West
2	Keyboard	10	50	500	East
3	Monitor	8	200	1600	South
4	Laptop	12	950	11400	North

Q4
```
df = pd.read_csv("/content/Dataset of Diabetes.csv",
                 encoding="ISO-8859-1")
print("Sample data:")
print(df.head())
```

Output:

Sample data:

	ID	NO_Patient	Gender	AGE	Urea	Cr
0	502	17975	F	50	4.7	46

	HbA1c	Chol	TG	HDL	LDL	VLDL
0	4.9	4.2	0.9	2.4	1.4	0.5

	BMI	CLASS
0	24.0	N

Analyzing Sales Dataset

1. Reading sales data

```
sales_df = pd.read_csv('sales_data.csv')
print("First few rows of the sales data:")
print(sales_df.head())
```

Output: First few rows of the sales data

	Product	Quantity	Price	Sales	Region
0	Laptop	5	1000	5000	North

Date / /
Page

2.1 Grouping by Region

```
sales_by_region = sales_df.groupby('Region')['Sales'].sum()
print("\n Total sales by region:")
print(sales_by_region)
```

output:

```
Total sales by region
Region
East     770
North    1640
South    3070
West     650
Name: Sales,
```

2.2 # Grop by product and calculating total quantity sold

```
best_selling_products = sales_df.groupby('Product')['Quantity'].sum().sort_values(ascending=False)
print("\n Best-selling products by quantity:")
print(best_selling_products)
```

Output: Best selling products by quantity:

```
Product
Mouse       29
Laptop      17
Keyboard    16
Monitor     15
Name: Quantity
```

Date / /
Page

3. Saving the analysis Results to a CSV File

```
# Saving the sales by region data to CSV File
sales_by_region.to_csv('sales_by_region.csv')

# Saving the best-selling products data to a csv fi
best_selling_products.to_csv("best_selling_product.c
print("\n Analysis results saved to CSV files."
```

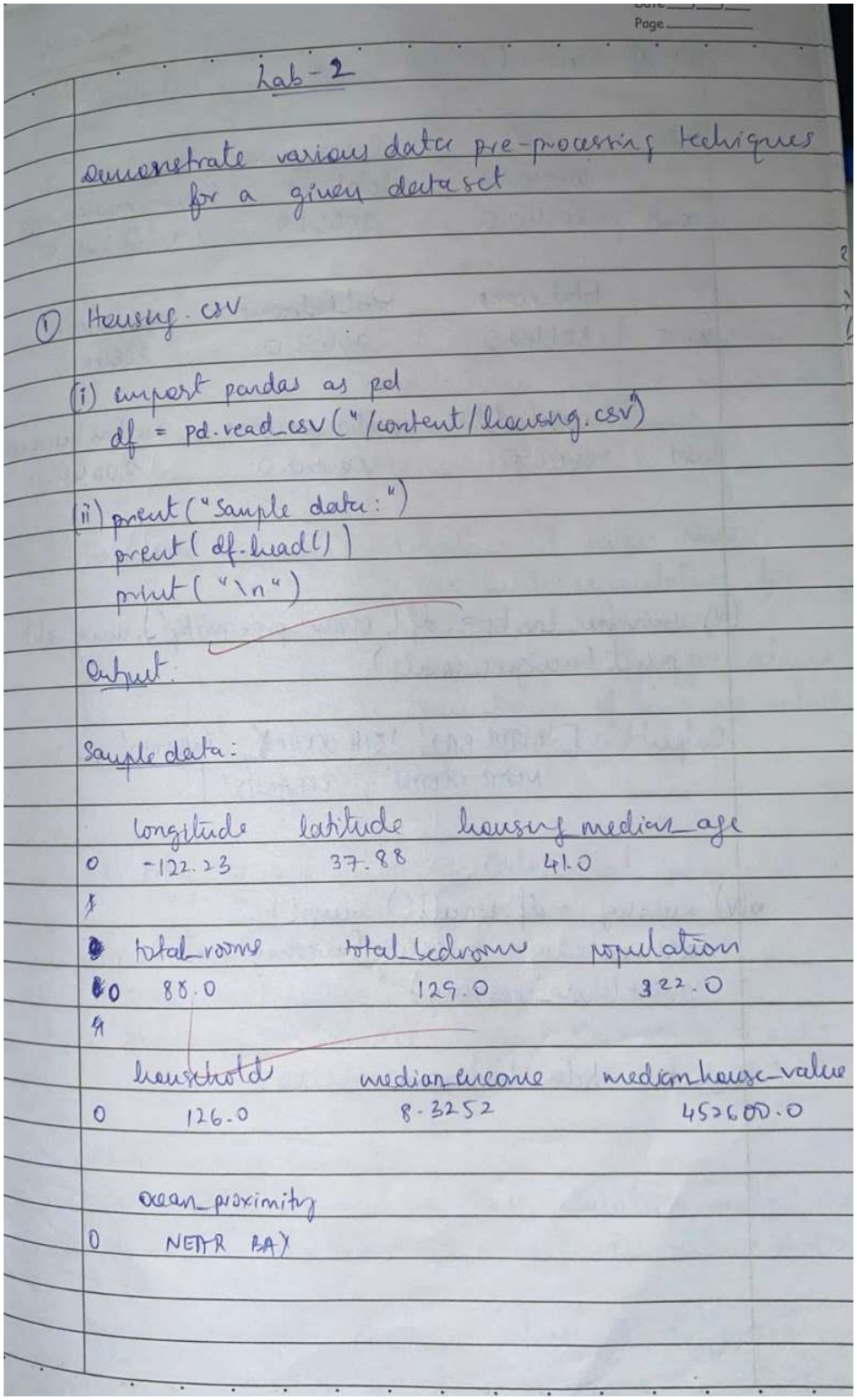
Output: Analysis results saved to CSV files

Code:

```
import pandas as pd

data = {
'USN': ['001', '002', '003', '004', "005"],
'Name': ["A", "B", "C", "D", "E"],
'Marks': [30, 45, 62, 93, 85]
}

df = pd.DataFrame(data)

print("Sample data:")
print(df.head())

#Export

import pandas as pd
# Reading data from a CSV file
df = pd.read_csv('sample_sales_data.csv')

print(df.head())
# Writing the DataFrame to a CSV file
df.to_csv('output.csv', index=False)
print("Data saved to output.csv")
```

Program 2

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



Lab - 2

Demonstrate various data pre-processing techniques for a given dataset

① Housing.csv

(i) import pandas as pd
df = pd.read_csv("/content/housing.csv")

(ii) print("Sample data:")
print(df.head())
print("\n")

Output:

Sample data:

	longitude	latitude	housing_median_age
0	-122.23	37.88	41.0

	total_rooms	total_bedrooms	population
0	88.0	129.0	322.0

	households	median_income	median_house_value
0	126.0	8.3252	452600.0

	ocean_proximity
0	NEAR BAY

Date __/__/__
Page

② Diabetes and Adult Income

1. Handling missing values:

Diabetes Dataset: There are no missing values in any columns.

Adult Income Dataset: The workclass, occupation and native country columns have missing values represented as "?"

Handling Strategy:
- Replace '?' with NaN
- Use mode imputation for categorical columns
- Drop rows or use predictive imputation if missing values are substantial.

2. Identifying and Encoding Categorical Columns

Diabetes Dataset:
- categorical columns: Gender and class
- encoding:
  - Gender: Convert F and M to 0 and 1
  - class: Convert N and Y to 0 and 1

Adult Income Dataset:
- categorical columns: workclass, education, marital-status, occupation, relationship, race, gender, native-country, and income
- Encoding: Label different values for different subcategories.

(iii) df.describe()

output:

	longitude	latitude	housing_median_age
count	20640.0	20640.0	20640.0

	total_rooms	total_bedrooms	population
count	20640.0	20433.0	20640.0

	households	median_income	median_house_val
count	20640.0	20640.0	20640.0

(iv) unique_counts = df['ocean_proximity'].unique()
print(unique_counts)

Output: ['NEAR BAY', '<1H OCEAN', 'INLAND',
'NEAR OCEAN', 'ISLAND']

(v) missing = df.isnull().sum()
cols_missing = missing[missing > 0]
print(cols_missing)

Output: total_bedrooms 207

Date __/__/__
Page

3. Min-Max Scaling vs. Standardisation

- Min-Max Scaling
  - → Scales values to a fixed range (0,1)
  - → Formula:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

  - → Used when data has fixed range and no outliers

- Standardization
  - → Centers data around mean = 0 and variance = 1
  - → Formula:

$$X_{standardized} = \frac{X - \mu}{\sigma}$$

  - → Used when data has different ranges or contains outliers

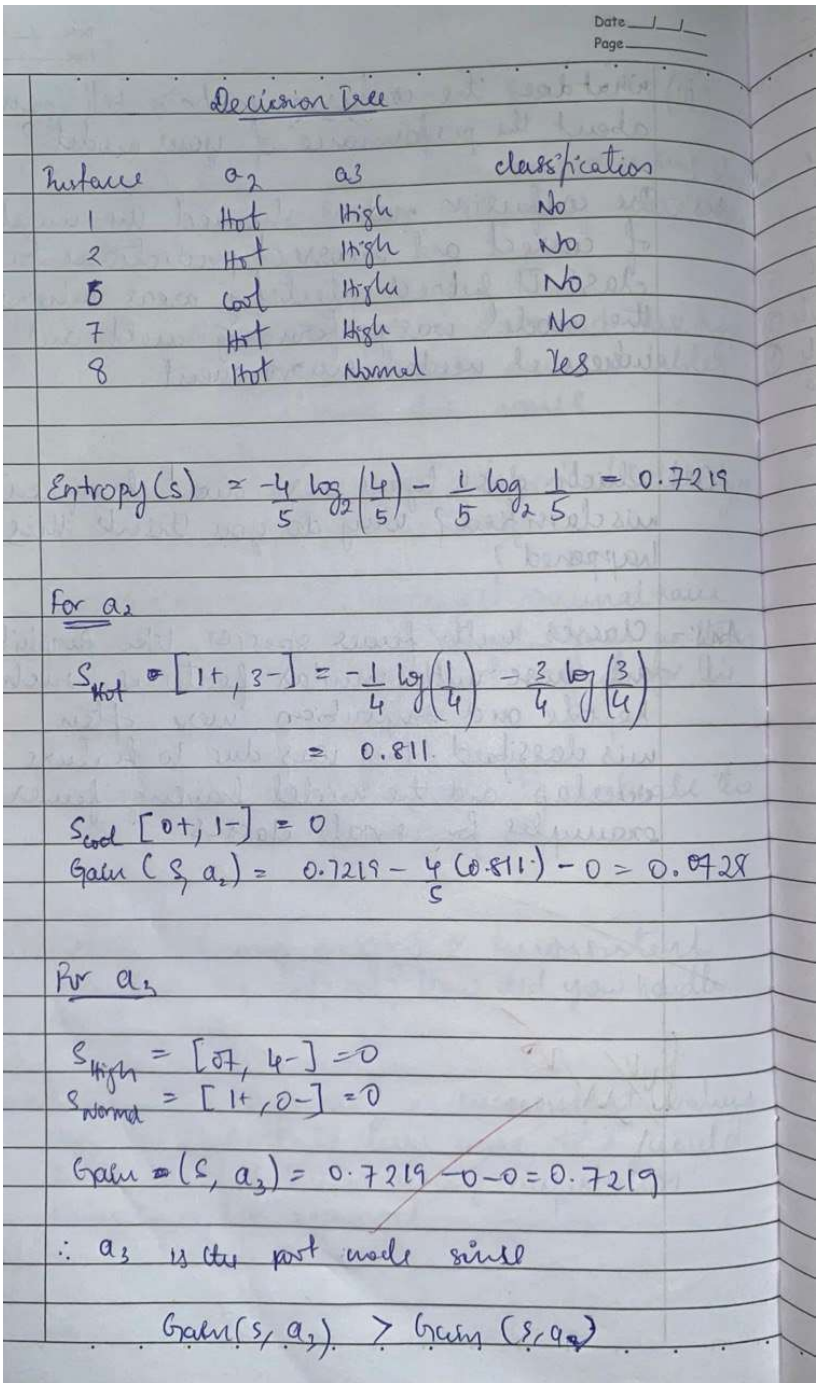
Code

```
# Step 1: Import required libraries
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder

# Step 2: Load the Adult Income dataset (Ensure it's uploaded to Colab)
file_path = "adult.csv"  # Update the path if needed
df = pd.read_csv(file_path)

# Step 3: Handling Missing Values (Replacing '?' with NaN and imputing mode for categorical, mean for numerical)
df.replace("?", np.nan, inplace=True)

# Handling numerical missing values with mean
num_imputer = SimpleImputer(strategy="mean")
df[df.select_dtypes(include=['number']).columns] =
num_imputer.fit_transform(df.select_dtypes(include=['number']))

# Handling categorical missing values with mode
cat_imputer = SimpleImputer(strategy="most_frequent")
df[df.select_dtypes(include=['object']).columns] =
cat_imputer.fit_transform(df.select_dtypes(include=['object']))

# Step 4: Handling Categorical Data (Encoding)
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Step 5: Handling Outliers (Removing values beyond 1.5*IQR)
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]

# Step 6: Data Transformation
# Min-Max Normalization
min_max_scaler = MinMaxScaler()
df_minmax = pd.DataFrame(min_max_scaler.fit_transform(df), columns=df.columns)

# Standardization (Z-score normalization)
standard_scaler = StandardScaler()
df_standard = pd.DataFrame(standard_scaler.fit_transform(df), columns=df.columns)
```

```
# Step 7: Display Processed Data
print("\nProcessed Adult Income Dataset (Min-Max Scaled):")
print(df_minmax.head())

print("\nProcessed Adult Income Dataset (Standard Scaled):")
print(df_standard.head())
```

Program 3

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



Decision Tree

Instance	a_2	a_3	classification
1	Hot	High	No
2	Hot	High	No
5	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes

$$Entropy(S) = -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\frac{1}{5} = 0.7219$$

For a_2

$$S_{Hot} = [1+, 3-] = -\frac{1}{4}\log\left(\frac{1}{4}\right) - \frac{3}{4}\log\left(\frac{3}{4}\right) = 0.811$$

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

$$Gain(S, a_2) = 0.7219 - \frac{4}{5}(0.811) - 0 = 0.0728$$

For a_3

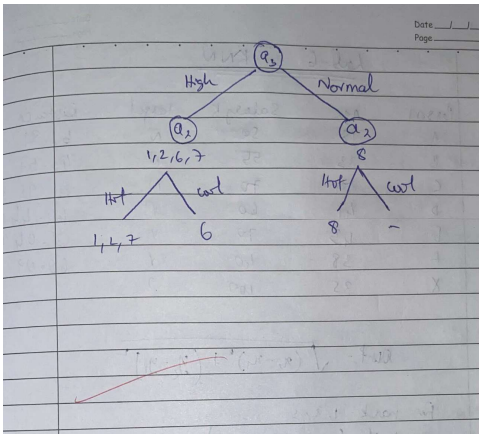
$$S_{High} = [0+, 4-] = 0$$

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

$$Gain(S, a_3) = 0.7219 - 0 - 0 = 0.7219$$

$\therefore a_3$ is the root node since

$$Gain(S, a_3) > Gain(S, a_2)$$



Date __/__/__
Page

Decision tree code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score,
confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

try:
    df = pd.read_csv("drug.csv")

except FileNotFoundError:
    print("Error: 'drug.csv' not found. Please
          make sure the file is in the correct
          directory.")
    exit()

categorical_cols = ['Sex', 'BP', 'Cholesterol']
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

X = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']]
y = df['Drug']

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

dt_classifier = DecisionTreeClassifier(random_state=42)
```

Date __/__/__
Page

```
dt_classifier.fit(X_train, y_train)
y_pred = dt_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion:", confusion)
print("classification Report:\n", report)
```

Output:

Confusion Matrix on Test Data:

```
[[6 0 0 0 0]
 [0 3 0 0 0]
 [0 0 5 0 0]
 [0 0 0 11 0]
 [0 0 0 0 15]]
```

classification Report on Test Data:

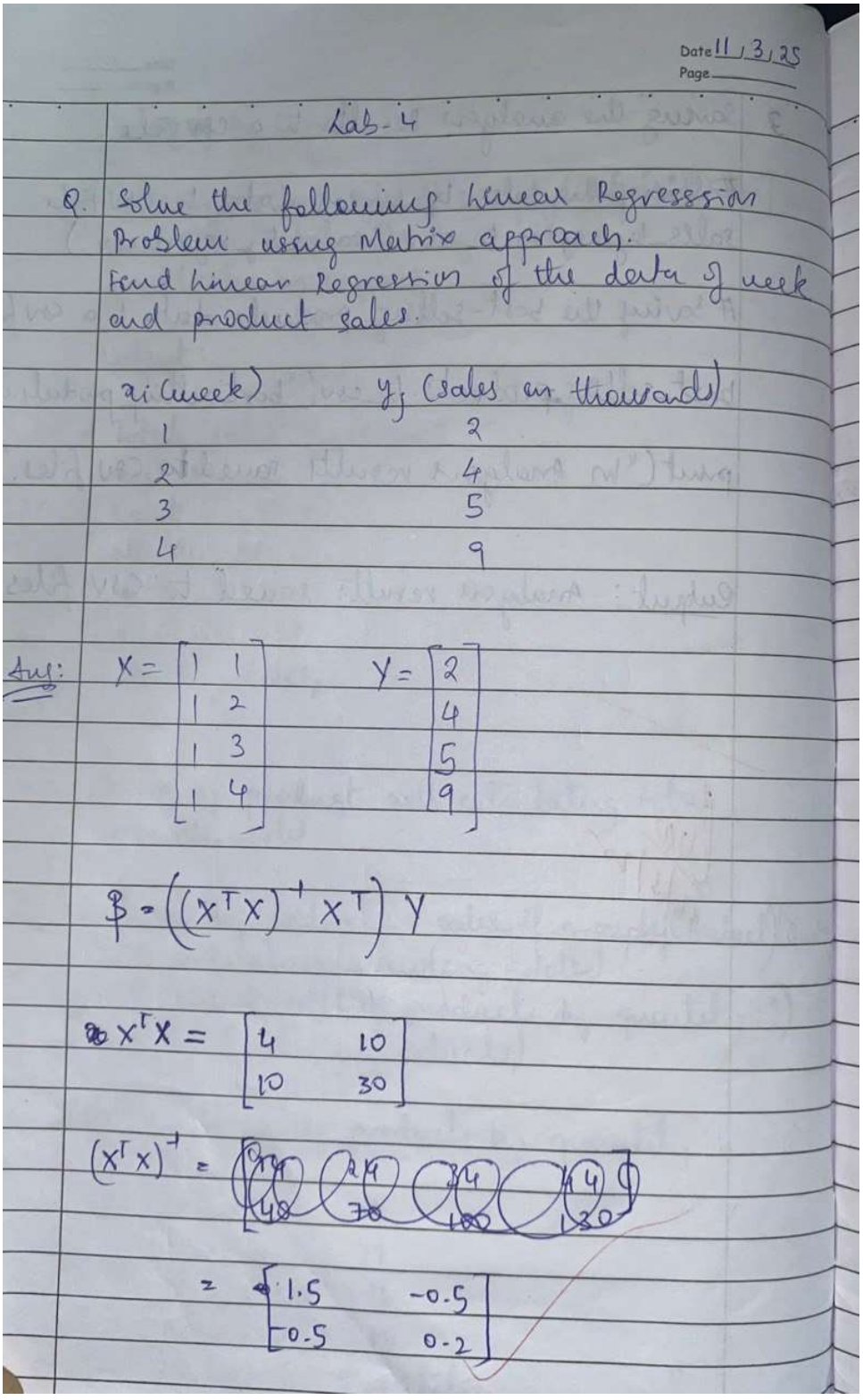
	precision	recall	f1-score	support
drugA	1.00	1.00	1.00	6

Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt

# Load the dataset
try:
    df = pd.read_csv("iris.csv")
except FileNotFoundError:
    print("Error: 'iris.csv' not found. Please make sure the file is in the correct directory.")
    exit()

# Separate features (X) and target (y)
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the classifier
dt_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = dt_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Display the results
print("Accuracy Score on Test Data:", accuracy)
print("\nConfusion Matrix on Test Data:\n", confusion)
print("\nClassification Report on Test Data:\n", report)

# Visualize the Decision Tree in the Colab interpreter
plt.figure(figsize=(12, 8))
plot_tree(dt_classifier, filled=True, feature_names=X.columns, class_names=y.unique(),
rounded=True)
plt.show()
```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset



Date 11/3/25
Page

Lab-4

Q. Solve the following linear Regression Problem using Matrix approach.
Find linear regression of the data of week and product sales.

x_i (week)	y_i (sales in thousands)
1	2
2	4
3	5
4	9

Ans:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \qquad Y = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$

$$\beta = \left((X^T X)^{-1} X^T\right) Y$$

$$X^T X = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

$$= \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

Date ___/___/___
Page _____

$$(X^T X^{-1})^{-1} X^T = \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix}$$

$$\left((X^T X)^{-1} X^T\right) Y = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

$\therefore \beta_0 = -0.5$

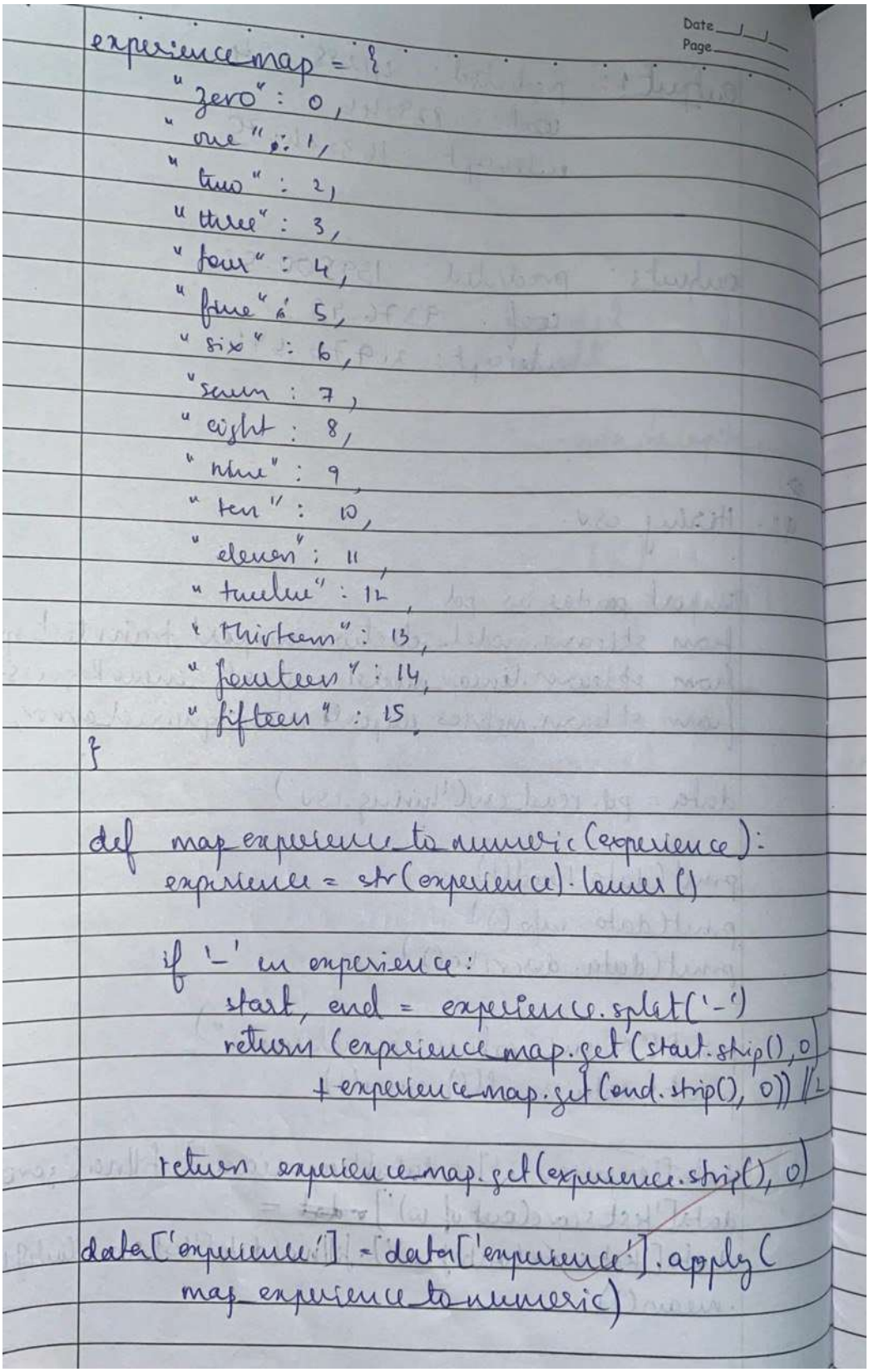
$\beta_1 = 2.2$

∴ for week 5: using $Y = \beta_0 + \beta_1 X + \varepsilon$

$Y = -0.5 + 2.2(5)$

$= 10.5$

Ans
11/03/25



```
experience_map = {
    "zero": 0,
    "one": 1,
    "two": 2,
    "three": 3,
    "four": 4,
    "five": 5,
    "six": 6,
    "seven": 7,
    "eight": 8,
    "nine": 9,
    "ten": 10,
    "eleven": 11,
    "twelve": 12,
    "thirteen": 13,
    "fourteen": 14,
    "fifteen": 15,
}

def map_experience_to_numeric(experience):
    experience = str(experience).lower()

    if '-' in experience:
        start, end = experience.split('-')
        return (experience_map.get(start.strip(), 0)
                + experience_map.get(end.strip(), 0)) // 2

    return experience_map.get(experience.strip(), 0)

data['experience'] = data['experience'].apply(
    map_experience_to_numeric)
```

Linear Regression

Q1. Canada's per capita income

Ans:-

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('/content/canada_per_capita_income
plt.xlabel('year')
plt.ylabel('per capita income (US$)')
plt.scatter(df.year, df['per capita income (US$)'],
            color='red', marker='+')

new_df = df.drop('per capita income (US$)',
                 axis='columns')
new_df
cpi = df['per capita income (US$)']

reg = linear_model.LinearRegression()
reg.fit(new_df, cpi)

reg.predict([[2020]])

reg.coef_

reg.intercept_
```

Output 1: predicted: 41288.694
coef: 828.46
intercept: -1632210.75

output 3: predicted: 139500.53
coef: 9376.99
intercept: 26976.596

Q2. Hiring.csv.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

data = pd.read_csv('hiring.csv')

print(data.head())
print(data.info())
print(data.describe())

print("Missing values per column:")
print(data.isnull().sum())

data['experience'] = data['experience'].fillna('zero')
data['test_score(out of 10)'] = data['test_score(out of 10)'].fillna(data['test_score(out of 10)'].mean())
```

```
X = data[['experience', 'test score(out of 10)',
          'interview_score(out of 10)']]

Y = data['salary($)']

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

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Mean Squared Error:", mean_squared_error
      (y_test, y_pred))

print("R-squared:", r2_score(y_test, y_pred))

candidate_1 = pd.DataFrame([[2,9,6]],
              columns=['experience',
              'test_score(out of 10)', 'interview_score
              (out of 10)'])

candidate_2 = pd.DataFrame([[12, 10, 10]],
              columns=['experience',
              'test_score(out of 10)',
              'interview_score(out of 10)'])

salary_1 = model.predict(candidate_1)
salary_2 = model.predict(candidate_2)
```

```
print(f"Predicted salary for candidate 1
      : {salary_1[0]}")

print(f"Predicted salary for candidate 2
      : {salary_2[0]}")

print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {model.coef_}")
```

Output 1:

Predicted salary for candidate 1: 54974.266
Predicted salary for candidate 2: 90717.173
Intercept: 20722.33
Coefficients: [2716.4 2126.9 1612.7]

Output 2:

Predicted profit for the candidate
(R&D: 91694.48
Admin: 515841.3
Marketing: 11931.24
State: Florida) : 554900.68

Intercept: -82639.33
Coefficients: [0.534 1.139 0.052 -42.87]

Code

Linear

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the dataset
data = pd.read_csv("salary.csv")

# Handle missing values by removing rows with NaN
data = data.dropna()

# Analyze data distribution
print(data.describe())
print(data.info())

# Distribution plot visualization
plt.scatter(data['YearsExperience'], data['Salary'], color='blue', label='Actual Data')
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Experience vs Salary")
plt.legend()
plt.show()

# Relationship between variables
X = data[['YearsExperience']]
y = data['Salary']

# Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict the results
y_pred = model.predict(X_test)

# Visualize prediction
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
```

```
plt.title("Prediction of Salary")
plt.legend()
plt.show()

# Check values of coefficient and intercept
print(f'Coefficient: {model.coef_[0]}")
print(f"Intercept: {model.intercept_}")

# Predict salary for an employee with 12 years of experience
salary_12_years = model.predict([[12]])
print(f'Predicted salary for 12 years of experience: {salary_12_years[0]:.2f} US$")

# Calculate errors
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")
```

Multi-Linear

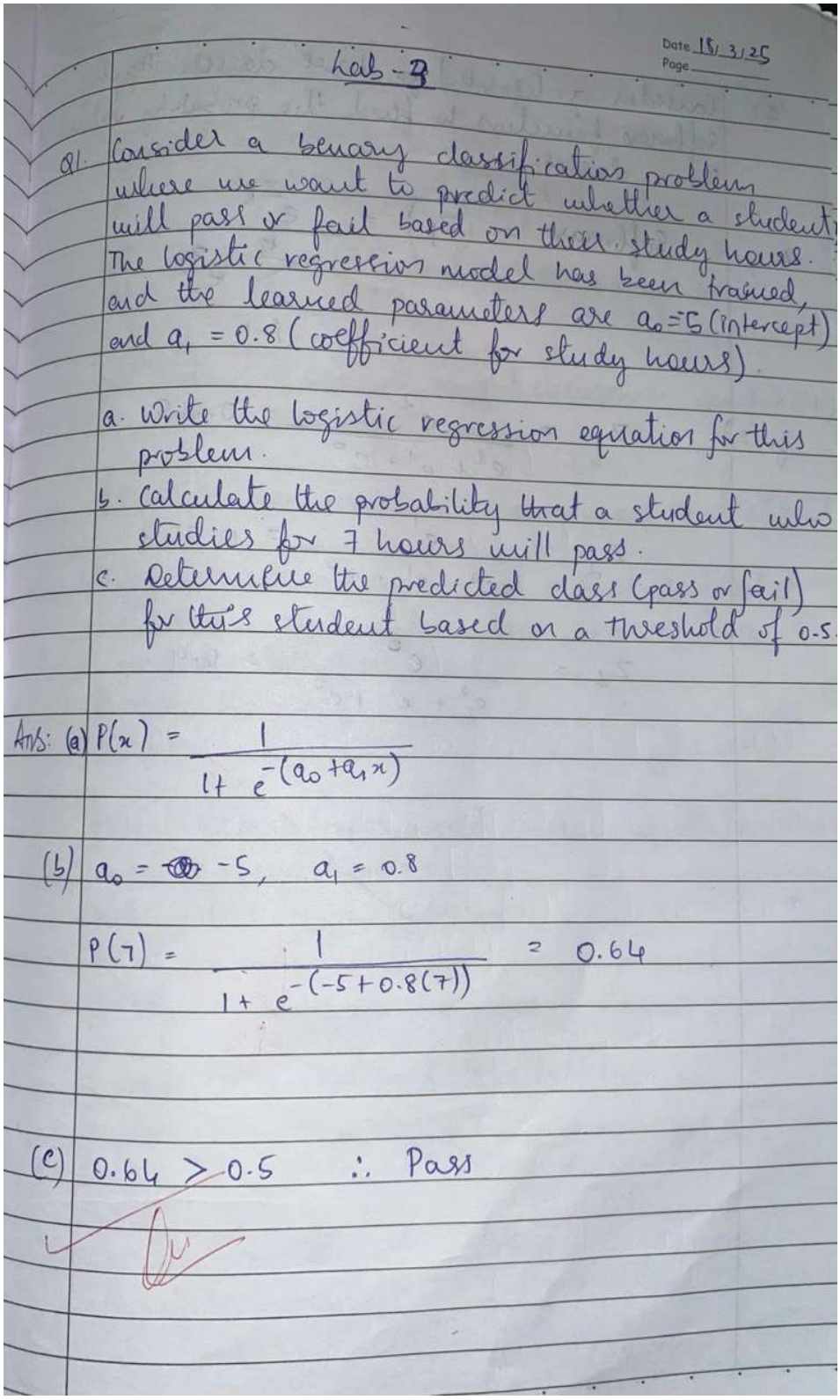
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv("hiring.csv")

# Function to convert experience from words to numbers
def convert_experience(value):
    word_to_num = {"zero": 0, "one": 1, "two": 2, "three": 3, "four": 4, "five": 5, "six": 6,
                   "seven": 7, "eight": 8, "nine": 9, "ten": 10, "eleven": 11, "twelve": 12}
    return word_to_num.get(value.lower(), value) if isinstance(value, str) else value

# Apply conversion
data['experience'] = data['experience'].apply(convert_experience)

# Handle missing values by removing rows with NaN
data = data.dropna()
```

```
# Convert all columns to numeric
data = data.astype(float)

# Analyze data distribution
print(data.describe())
print(data.info())

# Distribution plot visualization
plt.scatter(data['experience'], data['salary($)'], color='blue', label='Actual Data')
plt.xlabel("Experience (Years)")
plt.ylabel("Salary ($)")
plt.title("Experience vs Salary")
plt.legend()
plt.show()

# Relationship between variables
X = data[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y = data['salary($)']

# Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict the results
y_pred = model.predict(X_test)

# Visualize prediction
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel("Actual Salary")
plt.ylabel("Predicted Salary")
plt.title("Salary Prediction")
plt.legend()
plt.show()

# Check values of coefficients and intercept
print(f"Coefficients: {model.coef_}")
print(f"Intercept: {model.intercept_}")

# Predict salary for given candidates
candidates = np.array([[2, 9, 6], [12, 10, 10]])
salary_predictions = model.predict(candidates)
print(f"Predicted salary for 2 yrs experience, 9 test score, 6 interview score:
{salary_predictions[0]:.2f} US$")
print(f"Predicted salary for 12 yrs experience, 10 test score, 10 interview score:
```

```
{salary_predictions[1]:.2f} US$")

# Calculate errors
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")
```

Program 5

Build Logistic Regression Model for a given dataset



Lab - 3

Date 15/3/25
Page

Q1. 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, and the learned parameters are $a_0 = -5$ (intercept) and $a_1 = 0.8$ (coefficient for study hours).

a. Write the logistic regression equation for this problem.

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

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

Ans: (a) $P(x) = \frac{1}{1 + e^{-(a_0 + a_1 x)}}$

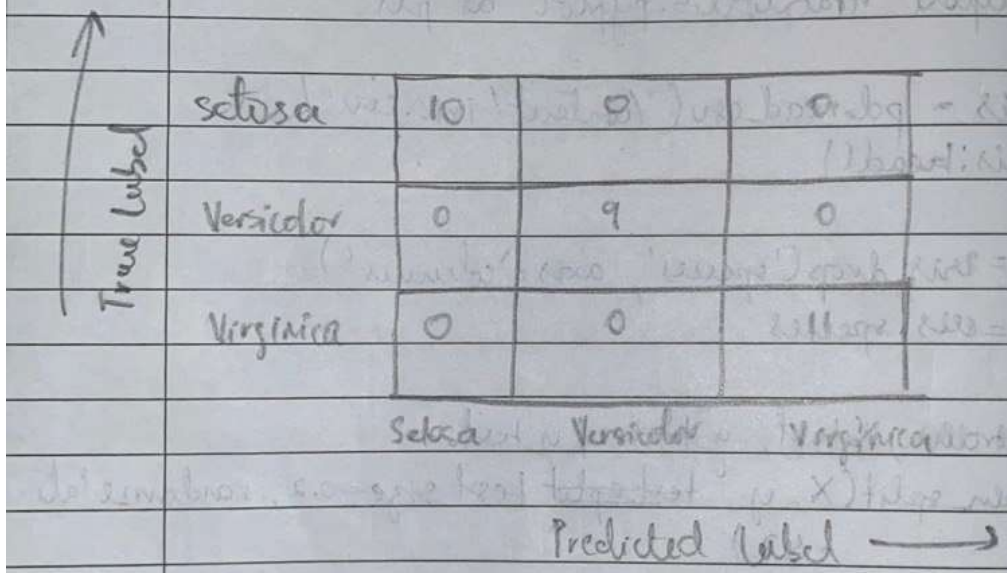
(b) $a_0 = -5, \quad a_1 = 0.8$

$P(7) = \frac{1}{1 + e^{-(-5 + 0.8(7))}} = 0.64$

(c) $0.64 > 0.5 \quad \therefore$ Pass

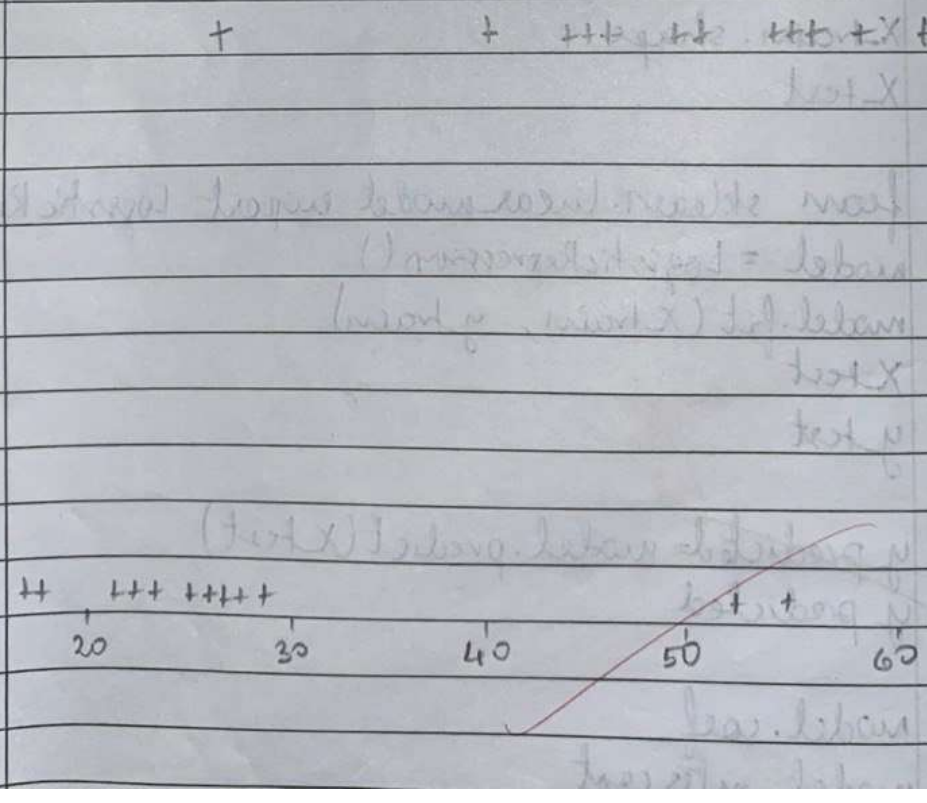
```
confusion_matrix = confusion_matrix, display_labels
                 = ["Setosa", "Versicolor", "Virginica"])

cm.display_plot()
plt.show()
```

Output :



```
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)
```

output:



Linear Regression (Multi class)

```
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("/content/iris.csv")
iris.head()

X = iris.drop('species', axis='columns')
y = iris.species

X_train, X_test, y_train, y_test =
train_split(X, y, ~~test_split~~ 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(
```

Date __/__/__
Page _____

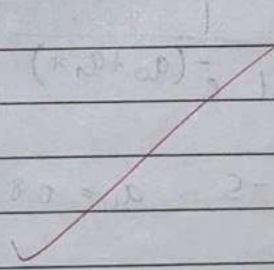
Q2. Consider $z = [2, 1, 0]$ for three classes. Apply softmax function to find the probabity values of three classes.

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

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

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

$$z_3 = \frac{e^0}{e^2 + e^1 + e^0} = 0.09$$



84

Date __/__/__
Page _____

Logistics Regression (Binary)

```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv("/content/insurance_data.csv")
df.head()

plt.scatter(df.age, df.boughtinsurance, 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.boughtinsurance, 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.coef_
model.intercept_
```

Date __/__/__
Page _____

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

Ans:- The confusion matrix showed the number of correct and incorrect predictions for each class. It helped identify areas where the model was performing well and where it needed improvement.

(iv) Which class types were most frequently misclassified? Why do you think this happened?

Ans:- Classes with fewer species, like Amphibian and those with similar features such as Reptile and Amphibian were often misclassified. This was due to feature overlap and the model having fewer examples for small classes.

Date / /
Page

1. For dataset file "HR_comma_sep.csv"

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

Ans:- • Satisfaction Level: How satisfaction leads to higher turnover

• Average Monthly Hours: Longer working hours often indicate burnout increasing turnover.

• Number of Projects: Too many projects can overwhelm employees leading to resignation.

• Time Spent at Company: Longer tenure often correlates with retention

• Promotion in Last 5 Years: Lack of promotion can make employees feel stagnant, prompting them to leave.

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

Ans:- The logistic regression model's accuracy would likely be around 80-85%. This is good, but it's important to evaluate further with metrics like precision, recall, and confusion matrix.

2. Zoo dataset

(i) Did you perform any data preprocessing step? If yes, what were they, and why were they necessary?

Ans- Merging datasets: Merged zoo data and class_types to associate numeric labels with class names

Label Encoding: Converted categorical class type to numeric labels.

Dropping Non-Numeric columns: Removed animal_name and Animal_Names as they were not useful for the model

Train-Test split: Split the data to evaluate the model performance

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

Ans- There were no missing or inconsistent values in the dataset. If there were, I would have handled them using imputation techniques or removal.

Code

Binary Classification

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

# Step 1: Load the dataset
df = pd.read_csv("/content/HR_comma_sep.csv")

# Step 2: Inspect the first few rows and basic info
print(df.head())
print(df.info())  # Check for missing values and data types

# Step 3: Check for missing values (None found as per your output)
print(df.isnull().sum())

# Step 4: Convert categorical columns (Department and Salary) to numeric using LabelEncoder
label_encoder = LabelEncoder()

# 'Department' and 'salary' are categorical, so we encode them
df['Department'] = label_encoder.fit_transform(df['Department'])
df['salary'] = label_encoder.fit_transform(df['salary'])

# Step 5: Exploratory Data Analysis (EDA)

# 5.1: Univariate Analysis - Check for distribution of 'left' column (employee retention)
sns.countplot(x='left', data=df)
plt.title('Employee Retention Distribution')
plt.show()

# 5.2: Distribution of other numerical features
sns.histplot(df['satisfaction_level'], kde=True)
plt.title('Satisfaction Level Distribution')
plt.show()

sns.histplot(df['last_evaluation'], kde=True)
plt.title('Last Evaluation Distribution')
plt.show()

# 5.3: Correlation heatmap to see correlations between features
```

```
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Step 6: Visualize key features vs 'left' (employee retention)

# 6.1: Satisfaction level vs retention
sns.boxplot(x='left', y='satisfaction_level', data=df)
plt.title('Satisfaction Level vs Retention')
plt.show()

# 6.2: Number of projects vs retention
sns.boxplot(x='left', y='number_project', data=df)
plt.title('Number of Projects vs Retention')
plt.show()

# 6.3: Average monthly hours vs retention
sns.boxplot(x='left', y='average_montly_hours', data=df)
plt.title('Average Monthly Hours vs Retention')
plt.show()

# Step 7: Feature Selection and Impact Analysis
# Checking the correlation between different features and 'left' (target)
print(correlation_matrix['left'])

# Step 8: Train a logistic regression model to predict employee retention
# Feature Selection (dropping the 'left' target variable)
X = df.drop('left', axis=1)
y = df['left']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a logistic regression model
model = LogisticRegression(max_iter=1000)  # max_iter increased to avoid convergence warning
model.fit(X_train, y_train)

# Step 9: Evaluate the model
y_pred = model.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Step 10: Feature importance (Impact on retention)
```

```
# Get feature coefficients from the logistic regression model
feature_importance = pd.DataFrame({'Feature': X.columns, 'Importance': model.coef_[0]})
feature_importance = feature_importance.sort_values(by='Importance', ascending=False)
print(feature_importance)

# Optional: Predict retention for a new employee (e.g., with some given features)
new_data = pd.DataFrame({
   'satisfaction_level': [0.80],
   'last_evaluation': [0.90],
   'number_project': [5],
   'average_montly_hours': [250],
   'time_spend_company': [5],
   'Work_accident': [0],
   'promotion_last_5years': [0],
   'Department': [1],  # Assuming 1 corresponds to 'sales' after label encoding
   'salary': [2]  # Assuming 2 corresponds to 'high' salary after label encoding
})

prediction = model.predict(new_data)
print(f"Prediction for new data (0: stayed, 1: left): {prediction[0]}")
```

Multiclass Classification

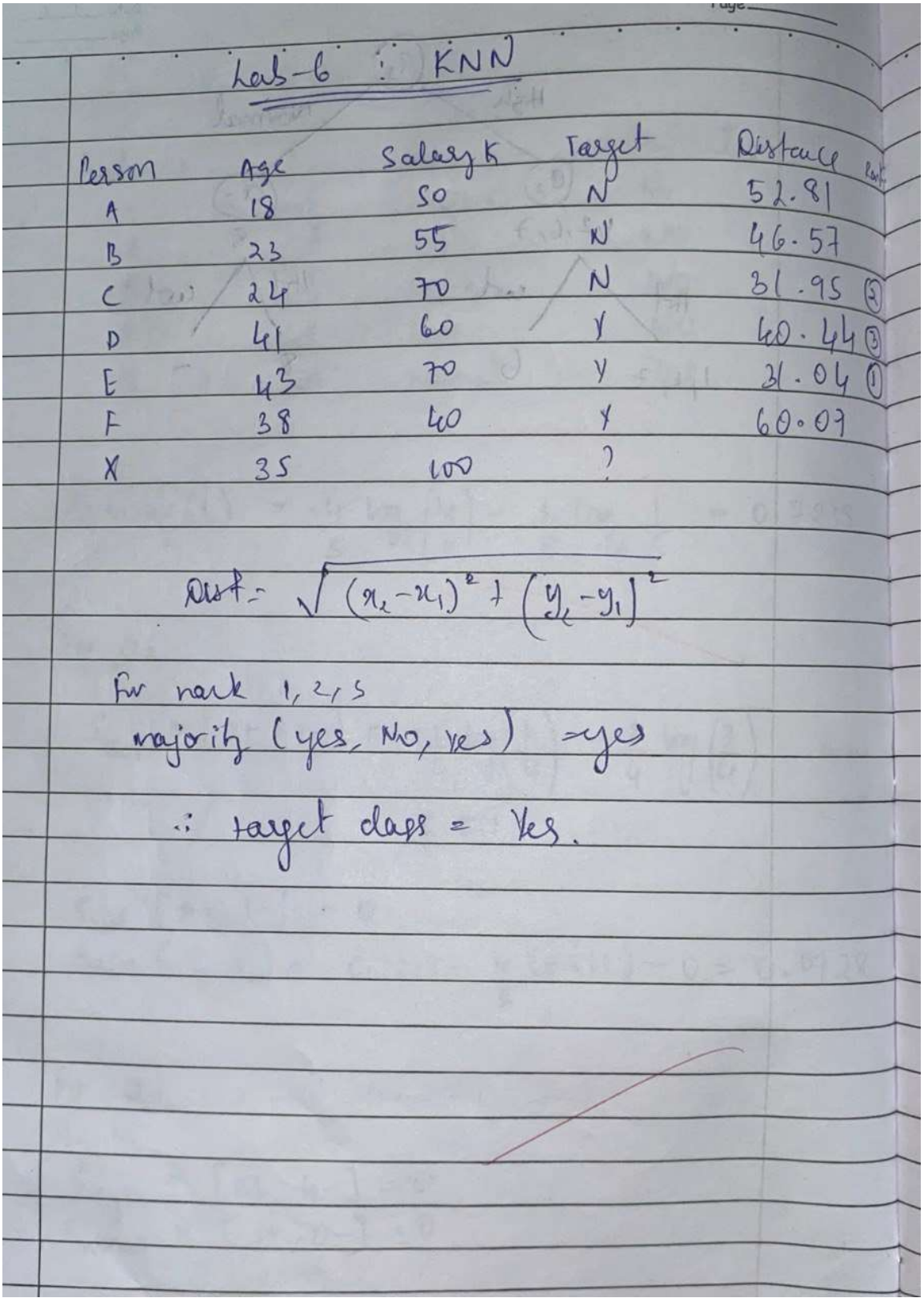
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Step 1: Load the datasets
zoo_data = pd.read_csv('zoo-data.csv')
class_type = pd.read_csv('zoo-class-type.csv')

# Display the first few rows to understand the structure
print(zoo_data.head())
print(class_type.head())

# Check the column names of zoo_data to ensure the correct column names are used
print("Zoo data columns:", zoo_data.columns)

# Step 2: Merge the zoo_data and class_type to get the 'class_type' column
# We are assuming 'class_type' from zoo_data matches 'Class_Type' in class_type file
```

```
zoo_data = zoo_data.merge(class_type[['Class_Type', 'Class_Number']], how='left',
left_on='class_type', right_on='Class_Number')

# Now, 'class_type' in zoo_data is mapped to the actual 'Class_Type' text
# Drop unnecessary columns from the merged DataFrame (e.g., 'Class_Number' as we already have
'class_type')
zoo_data = zoo_data.drop(columns=['Class_Number'])

# Step 3: Handle categorical data (Encode class_type if it's categorical)
# We already have 'class_type' in numeric form in zoo_data. If it was not numeric, we would encode
it.
# Let's check and encode class_type if necessary

# Encode class_type (if not already encoded) using LabelEncoder
label_encoder = LabelEncoder()
zoo_data['class_type'] = label_encoder.fit_transform(zoo_data['Class_Type'])

# Step 4: Drop non-numeric columns (e.g., 'animal_name', 'Animal_Names') and prepare feature set
(X) and target variable (y)
# Ensure to drop columns that are not part of the numeric features and the target variable

# Print column names again to ensure correctness
print("Columns before dropping non-numeric columns:", zoo_data.columns)

# Drop the columns that are non-numeric (we'll drop 'animal_name' and 'Animal_Names' if they
exist)
# If 'Animal_Names' is not a column in zoo_data, it won't throw an error now
X = zoo_data.drop(columns=['animal_name', 'Class_Type', 'Animal_Names'], errors='ignore')  #
'errors="ignore"' will prevent KeyError if the column doesn't exist
y = zoo_data['class_type']  # Target variable (encoded class_type)

# Step 5: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 6: Build the Logistic Regression model
model = LogisticRegression(max_iter=1000, multi_class='ovr')  # 'ovr' for one-vs-rest classification
model.fit(X_train, y_train)

# Step 7: Make predictions on the test data
y_pred = model.predict(X_test)

# Step 8: Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Logistic Regression model: {accuracy:.4f}')

# Step 9: Plot the Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_,
yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Program 6

Build KNN Classification model for a given dataset.



Lab-6 : KNN

Person	Age	Salary k	Target	Distance	Rank
A	18	50	N	52.81	
B	23	55	N	46.57	
C	24	70	N	31.95	2
D	41	60	Y	40.44	3
E	43	70	Y	31.04	1
F	38	40	Y	60.07	
X	35	100	?		

$$\text{dist} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

For rank 1, 2, 5

majority (yes, No, Yes) = yes

∴ target class = Yes.

Date __/__/____
Page _____

KNN

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbours import KNeighboursClassifier
from sklearn.metrics import accuracy_score,
confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score

try:
    df = pd.read_csv("diabetes.csv")
except FileNotFoundError:
    print("Error")
    exit()

X = df.drop('Outcome', axis=1)
y = df['Outcome']

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=4
                 stratify=y)

scaler = StandardScaler()
X_train_scaled & = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

K_values = list(range(1, 21, 2))
cv_scores = []
```

```
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_scaled,
             y_train, cv=5, scoring='accuracy')
    cv_scores.append(scores.mean())

best_k = k_values[cv_scores.index(max(cv_scores))]

knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
knn_classifier.fit(X_train_scaled, y_train)

y_pred = knn_classifier.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"In KNN Classifier with K = {best_k} (Scaled Data)")
print("Accuracy Score on test Data:", accuracy)
print("\n Confusion Matrix on Test Data: \n", confusion)
print("\n Classification Report on Test Data: \n", report)
```

Output:

- Confusion Matrix on Test Data: [[84 16]
  [23 31]]

Classification Report

	precision	recall	f1-score	support
0	0.79	0.84	0.81	100
1	0.66	0.57	0.61	54
accuracy			0.75	154
macro avg	0.72	0.71	0.71	154
weighted avg	0.74	0.75	0.74	154

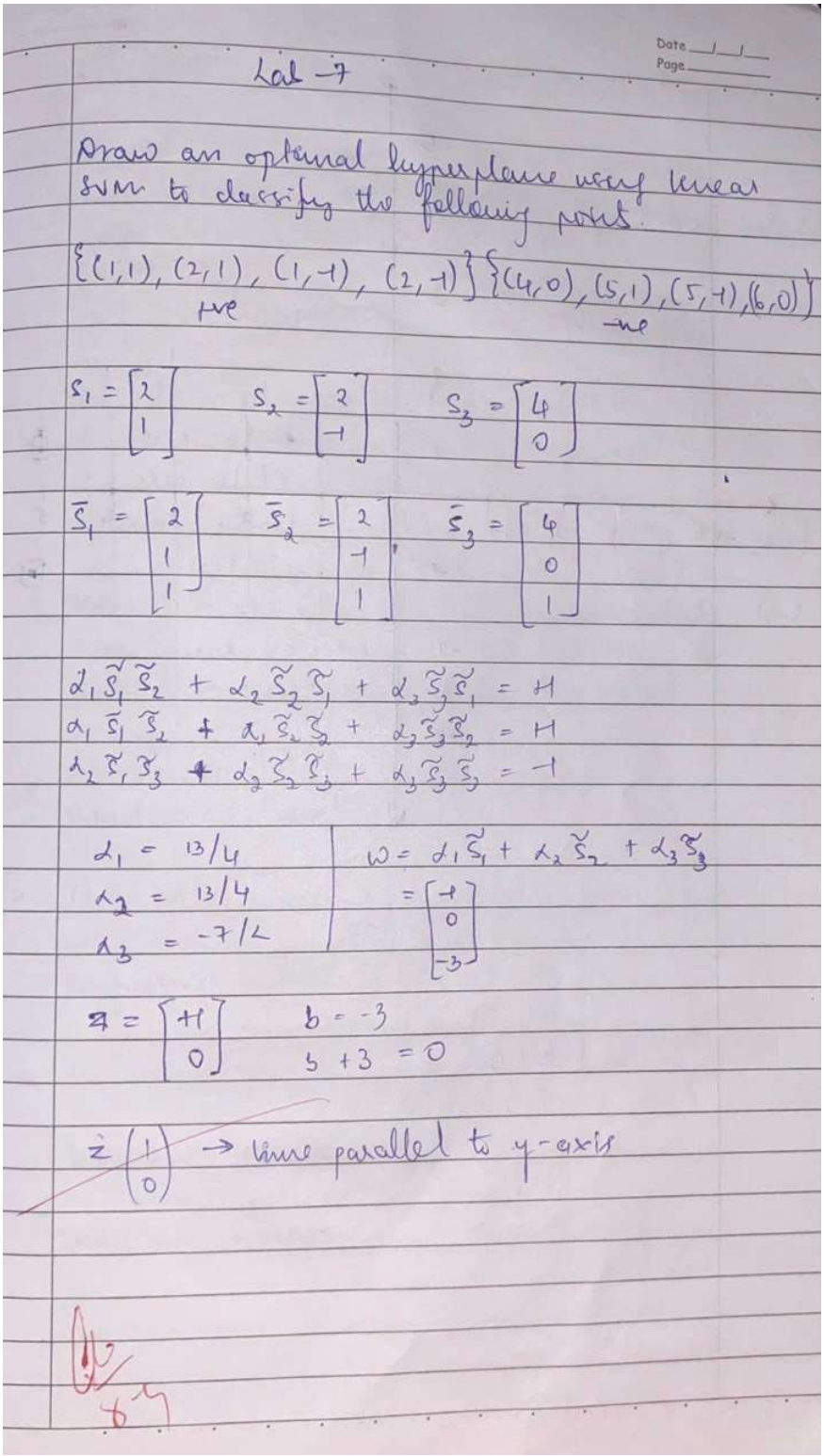
Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_score

# Load the dataset
try:
    df = pd.read_csv("iris.csv")
except FileNotFoundError:
    print("Error: 'iris.csv' not found. Please make sure the file is in the correct directory.")
    exit()

# Separate features (X) and target (y)
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Choose an appropriate k value using cross-validation
k_values = list(range(1, 21))  # Try k values from 1 to 20
cv_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=5, scoring='accuracy')
    cv_scores.append(scores.mean())

# Plot the cross-validation scores to help choose k
plt.figure(figsize=(10, 6))
plt.plot(k_values, cv_scores, marker='o')
plt.title('Cross-Validation Accuracy vs. K Value')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Mean Cross-Validation Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# Based on the plot, choose the best k value
# Let's assume the plot suggests k=5 is a good choice (this might vary)
best_k = 5

# Build and train the KNN classifier with the chosen k
```

```
knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
knn_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Display the results
print(f"\nK-Nearest Neighbors Classifier with K = {best_k}")
print("Accuracy Score on Test Data:", accuracy)
print("\nConfusion Matrix on Test Data:\n", confusion)
print("\nClassification Report on Test Data:\n", report)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',
            xticklabels=df['species'].unique(), yticklabels=df['species'].unique())
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Program 7

Build Support vector machine model for a given dataset



Lab -7

Draw an optimal hyperplane using linear SVM to classify the following points.

$\{(1,1), (2,1), (1,-1), (2,-1)\}$ +ve $\{(4,0), (5,1), (5,-1), (6,0)\}$ -ve

$$s_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad s_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \quad s_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$\bar{s}_1 = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} \quad \bar{s}_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \quad \bar{s}_3 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$$

$$\alpha_1 \tilde{s}_1 \tilde{s}_2 + \alpha_2 \tilde{s}_2 \tilde{s}_1 + \alpha_3 \tilde{s}_3 \tilde{s}_1 = +1$$
$$\alpha_1 \tilde{s}_1 \tilde{s}_2 + \alpha_2 \tilde{s}_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3 \tilde{s}_2 = +1$$
$$\alpha_1 \tilde{s}_1 \tilde{s}_3 + \alpha_2 \tilde{s}_2 \tilde{s}_3 + \alpha_3 \tilde{s}_3 \tilde{s}_3 = -1$$

$\alpha_1 = 13/4$
$\alpha_2 = 13/4$
$\alpha_3 = -7/2$

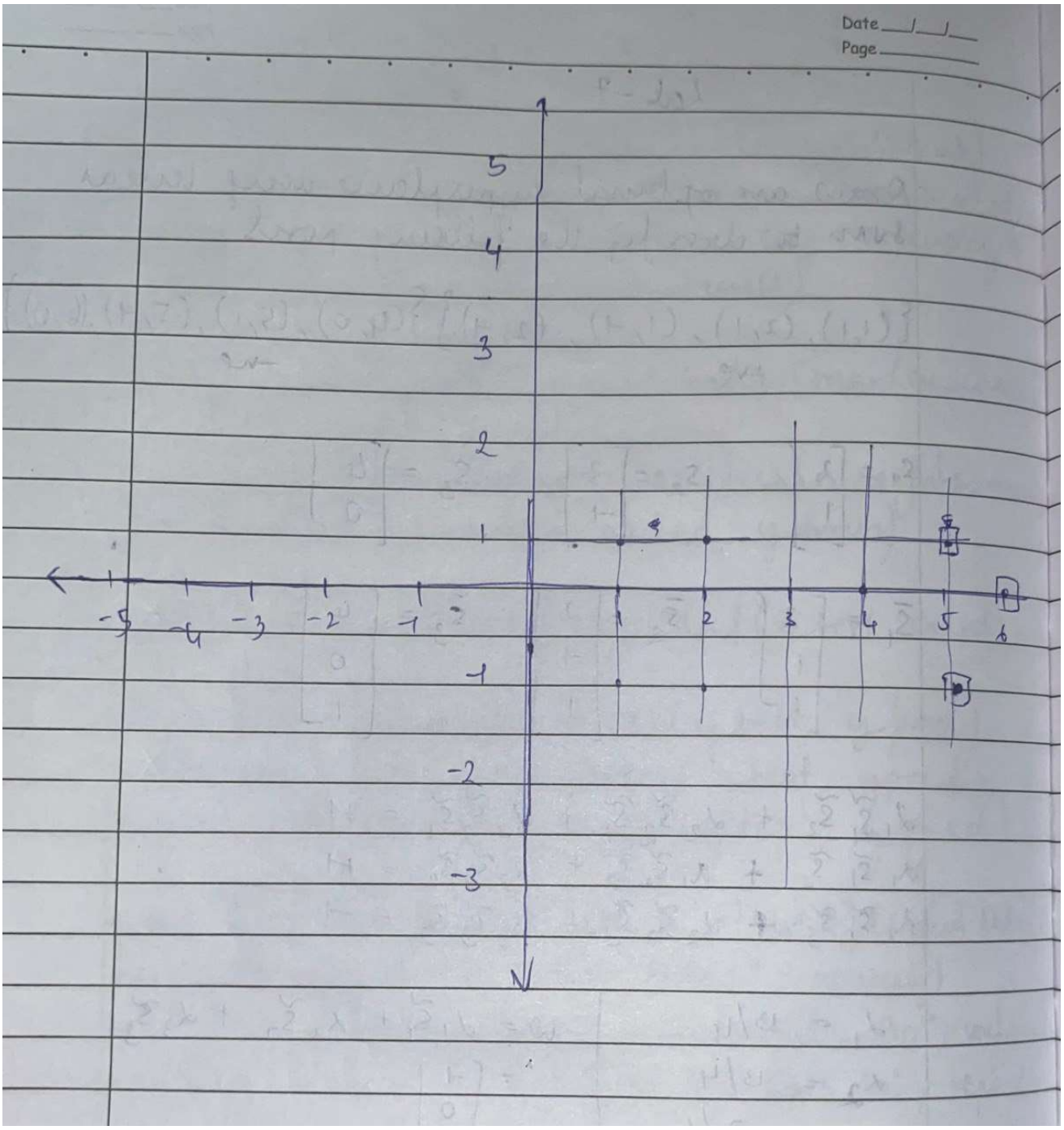
$w = \alpha_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3$

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

$$w = \begin{bmatrix} +1 \\ 0 \end{bmatrix} \quad b = -3$$

$x + 3 = 0$

$x \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ → line parallel to y-axis


Date
Page
5
4
3
2
1
-1
-2
-3
-5
-4
-3
-2
-1
1
2
3
4
5
6

code SVM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.SVM import SVC
from sklearn.metrics import accuracy_score,
confusion_matrix

iris_df = pd.read_csv("iris.csv")
x = iris_df.drop(columns = ['species'])
y = iris_df["species"]
x_train, x_test, y_train, y_test = train_test_split
        test_size=0.2, random_state =42)
svm_rbf = SVC(kernel = "rbf", random_state = 42)
svm.linear.fit(x_train, y_train)
y_pred = svm.linear_predict(x_test)

print(y_pred)
print(pred_linear)
```

O/p: RBF SVM accuracy : 1.0

confusion matrix : $\begin{bmatrix} 10 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 11 \end{bmatrix}$

linear svm accuracy : 1.0

Confusion matrix : $\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$

Date __/__/__
Page ____

Questions

1. Both RBF & linear gave the same accuracy score for iris dataset

2. The accuracy score : 96.8% ⇒ AUC model performed very well for letter-recognition.csv

The accuracy score for iris.csv was 1.00 as it was very small & clean dataset.

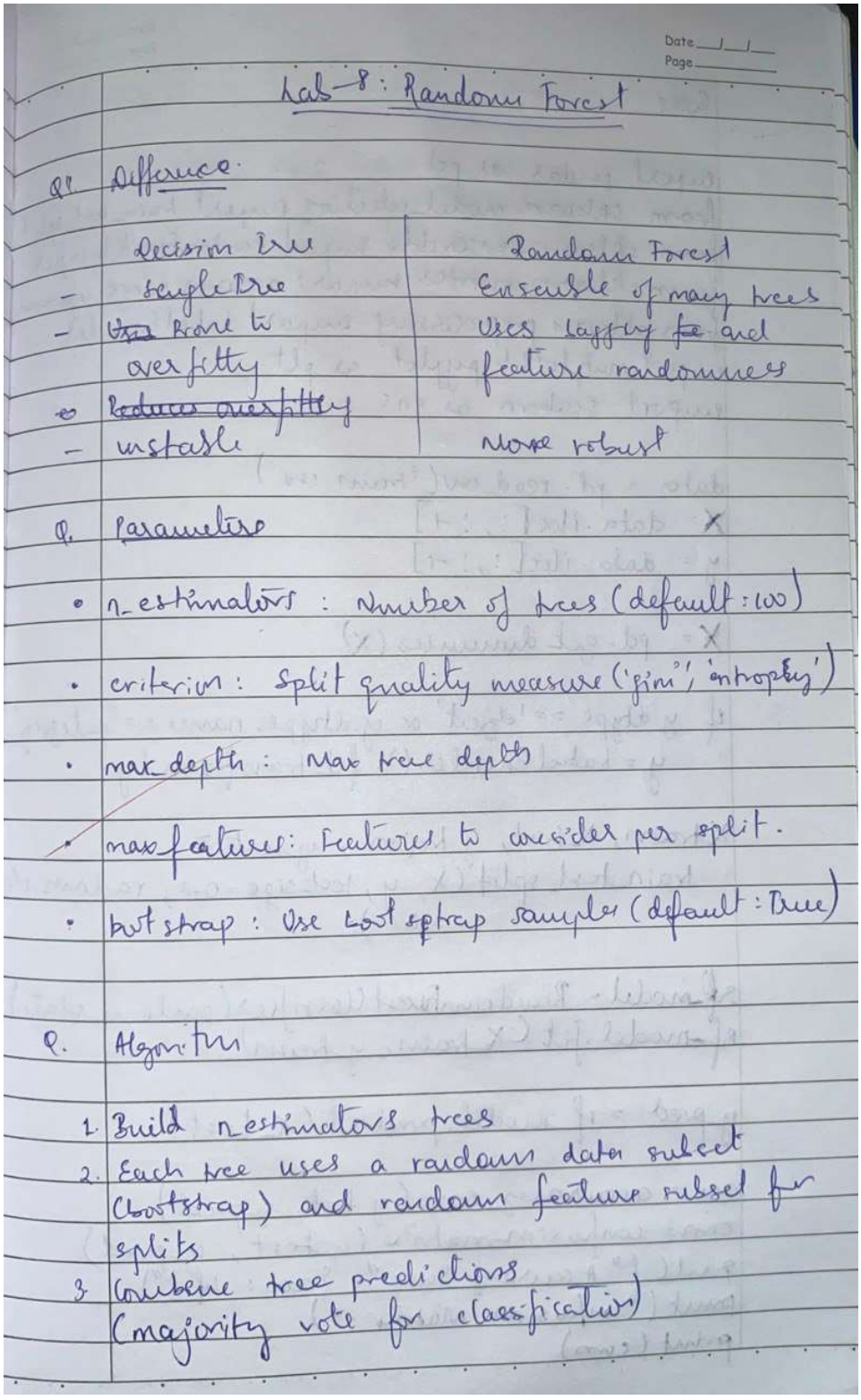
Code

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load your local dataset (assuming it's in CSV format)
# Replace 'path_to_your_iris_data.csv' with the actual file path of your dataset
file_path = 'iris.csv'  # Example: 'C:/datasets/iris.csv'
data = pd.read_csv(file_path)

# Check the first few rows to confirm it loaded correctly
print(data.head())

# If necessary, adjust column names or data format (e.g., ensure correct column names)
# Assuming columns: sepal_length, sepal_width, petal_length, petal_width, species

# Assign features (X) and target (y)
X = data.drop('species', axis=1)  # Features (all columns except 'species')
y = data['species']  # Target (species)

# Split the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the SVM with Linear Kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)

# Make predictions on the test data
y_pred_linear = svm_linear.predict(X_test)

# Calculate accuracy and confusion matrix for linear kernel
accuracy_linear = accuracy_score(y_test, y_pred_linear)
conf_matrix_linear = confusion_matrix(y_test, y_pred_linear)

# Initialize and train the SVM with RBF Kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)

# Make predictions on the test data
y_pred_rbf = svm_rbf.predict(X_test)

# Calculate accuracy and confusion matrix for RBF kernel
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
```

```
conf_matrix_rbf = confusion_matrix(y_test, y_pred_rbf)

# Displaying accuracy scores
print(f"Accuracy (Linear Kernel): {accuracy_linear * 100:.2f}%")
print(f"Accuracy (RBF Kernel): {accuracy_rbf * 100:.2f}%")

# Displaying confusion matrices
print("\nConfusion Matrix (Linear Kernel):")
print(conf_matrix_linear)
print("\nConfusion Matrix (RBF Kernel):")
print(conf_matrix_rbf)

# Plotting confusion matrices using seaborn heatmap for better visualization
def plot_confusion_matrix(cm, title):
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=data['species'].unique(),
yticklabels=data['species'].unique())
    plt.title(title)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Plot confusion matrices
plot_confusion_matrix(conf_matrix_linear, "Confusion Matrix (Linear Kernel)")
plot_confusion_matrix(conf_matrix_rbf, "Confusion Matrix (RBF Kernel)")
```

Program 8

Implement Random Forest ensemble method on a given dataset.



Lab-8: Random Forest

Q. Difference.

Decision Tree	Random Forest
- Single Tree	Ensemble of many trees
- Prone to overfitting	Uses bagging and feature randomness
- Reduce overfitting	
- unstable	More robust

Q. Parameters

- n_estimators : Number of trees (default: 100)
- criterion : Split quality measure ('gini', 'entropy')
- max_depth : Max tree depth
- max_features: Features to consider per split.
- bootstrap : Use bootstrap sampler (default: True)

Q. Algorithm

1. Build n_estimators trees
2. Each tree uses a random data subset (bootstrap) and random feature subset for splits
3. Combine tree predictions (majority vote for classification)

Date __/__/__
Page ____

Code :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt.
import seaborn as sns

data = pd.read_csv('train.csv')
X = data.iloc[:, :-1]
y = data.iloc[:, :-1]

X = pd.get_dummies(X)

if y.dtype == 'object' or y.dtype.name == 'category':
    y = LabelEncoder().fit_transform(y)

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

rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

y_pred = rf_modul.predict(X_test)

acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print(f"Accuracy Score" {acc:.4f}")
print("Confusion Matrix :")
print(cm)
```

Date __/__/__
Page _____

output:

Accuracy score : 0.7263

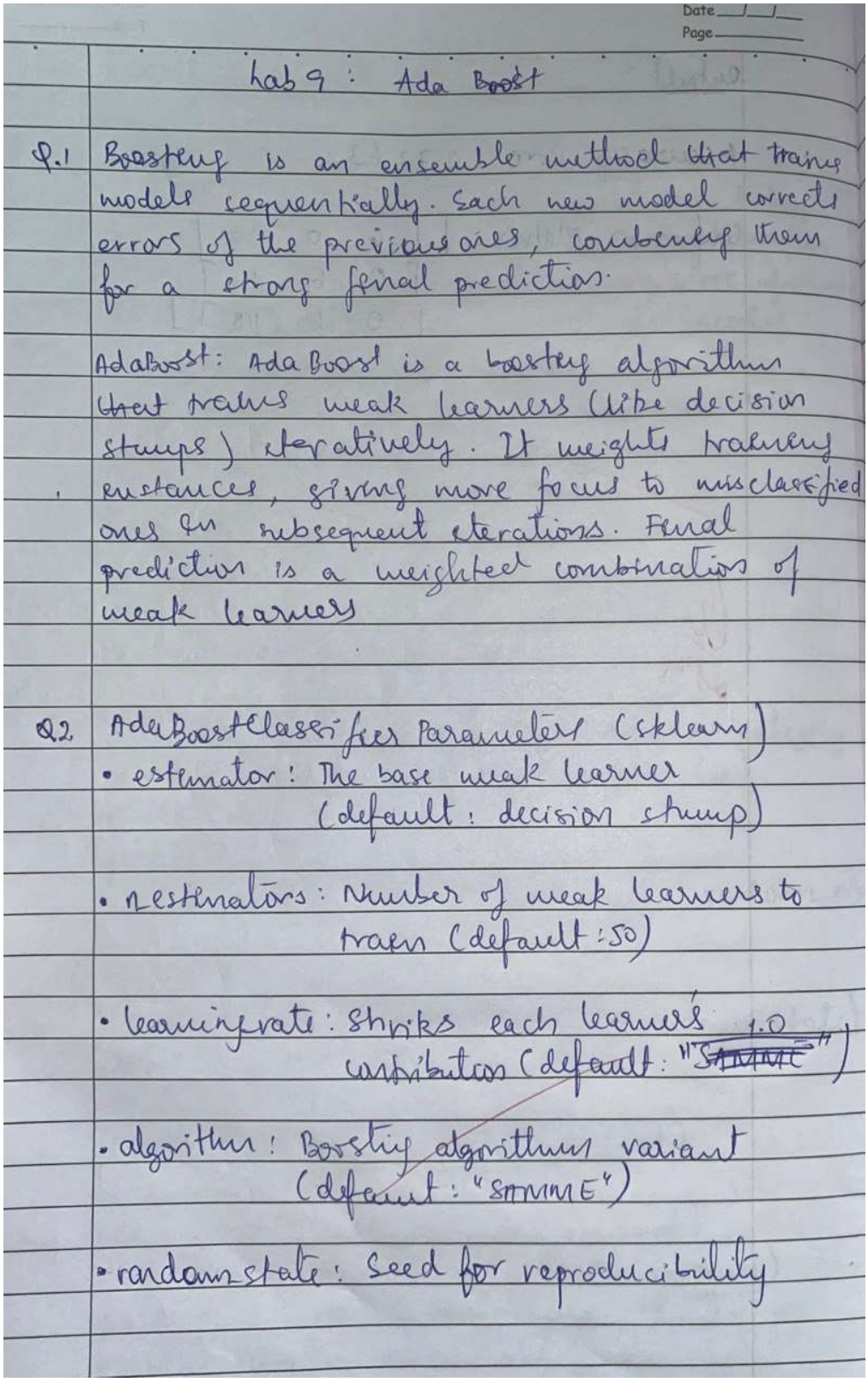
Confusion Matrix : [[[6 0 37]
[0 6 11]
[0 1 118]]]

Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
data = pd.read_csv('train.csv')
# Separate features and target (Assume last column is target)
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Encode categorical features in X
X = pd.get_dummies(X)

# Encode the target if it's categorical
if y.dtype == 'object' or y.dtype.name == 'category':
    y = LabelEncoder().fit_transform(y)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Random Forest model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Predictions
y_pred = rf_model.predict(X_test)

# Evaluate
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Accuracy Score: {acc:.4f}")
print("Confusion Matrix:")
print(cm)

# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

Program 9

Implement Boosting ensemble method on a given dataset.



Lab 9 : Ada Boost

Q.1 Boosting is an ensemble method that trains models sequentially. Each new model corrects errors of the previous ones, combining them for a strong final prediction.

AdaBoost: AdaBoost is a boosting algorithm that trains weak learners (like decision stumps) iteratively. It weights training instances, giving more focus to misclassified ones in subsequent iterations. Final prediction is a weighted combination of weak learners

Q2. AdaBoostClassifier Parameters (sklearn)

- estimator: The base weak learner (default: decision stump)
- n_estimators: Number of weak learners to train (default: 50)
- learning_rate: Shrinks each learners contribution (default: 1.0)
- algorithm: Boosting algorithm variant (default: "SAMME")
- random_state: Seed for reproducibility

Date / /
Page

Algorithm

1. Start with equal instance weights
2. Repeat for a set number of learners:
   - Train a weak learner on weighted data
   - Calculate learner error and its weight (α)
   - Increase weights for misclassified instances
   - Normalize instance weights.
3. Combine learner predictions using their α weights.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score,
                            confusion_matrix

df = pd.read_csv("income.csv")
df.columns = df.columns.str.strip()

if df['income_level'].dtype == 'object':
    le = LabelEncoder()
    df['income_level']
    = le.fit_transform(df['income_level'])
```

```
X = df.drop('income_level', axis=1)
y = df['income_level']

Xtrain, Xtest, ytrain, ytest =
train_test_split(X, y, test_size=0.2,
                 random_state=42)

model = AdaBoostClassifier(n_estimators=100,
                           random_state=42)
model.fit(Xtrain, ytrain)

ypred = model.predict(Xtest)

accuracy = accuracy_score(ytest, ypred)
conf_matrix = confusion_matrix(ytest, ypred)

print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
```

Output:

```
Accuracy: 0.8328
Confusion Matrix:
[[7117  297
 [1336 1019]]
```

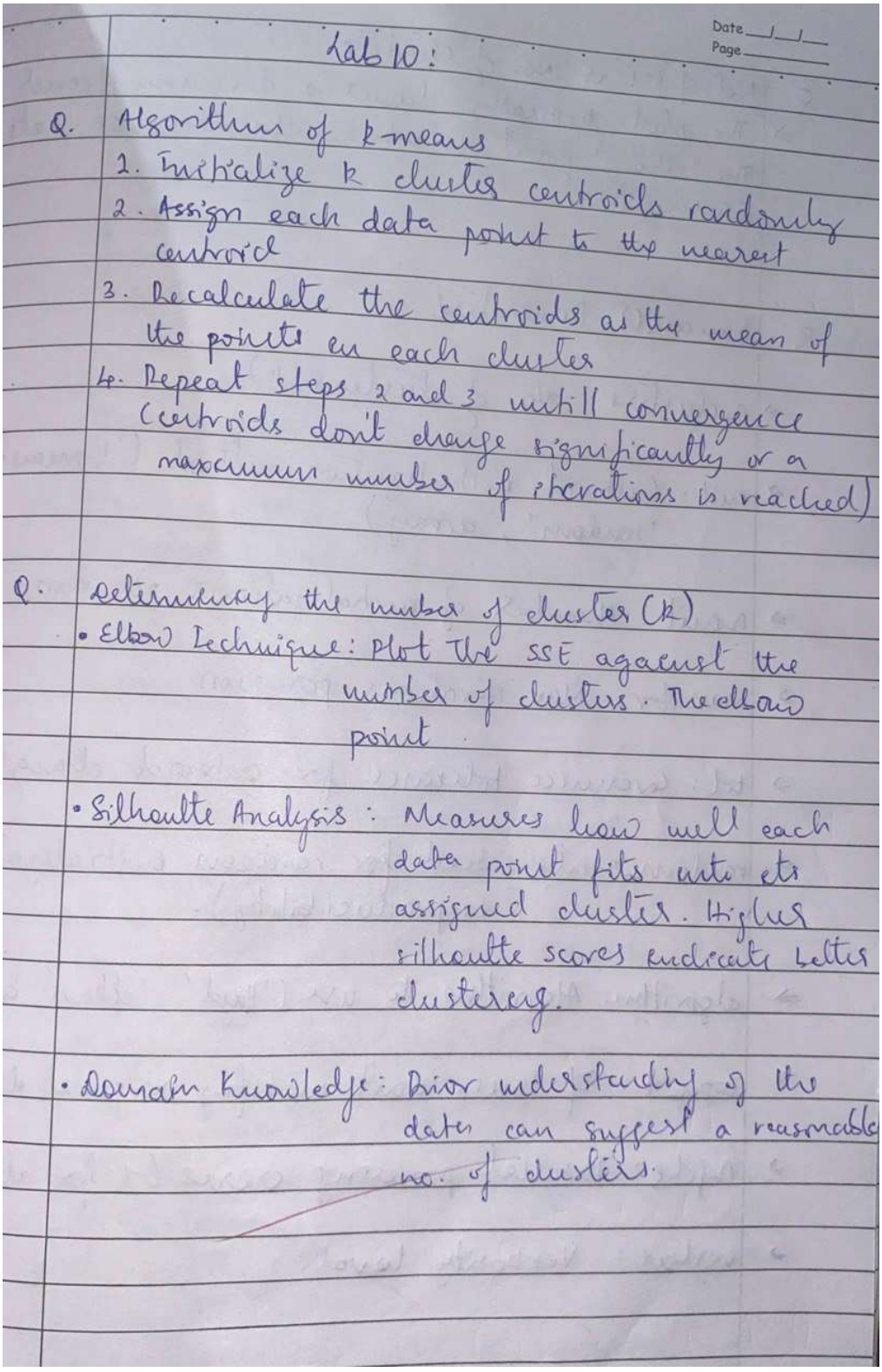
Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load Iris dataset for example
data = load_iris()
X = data.data
y = data.target

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

# Hyperparameters to tune
n_estimators_list = [10, 50, 100]
learning_rates = [0.1, 0.5, 1.0]

# AdaBoost with DecisionTreeClassifier base estimator
print("\nAdaBoost with DecisionTreeClassifier base estimator:")
for n in n_estimators_list:
    for lr in learning_rates:
        base_dt = DecisionTreeClassifier(max_depth=1)  # Decision stump
        ada_dt = AdaBoostClassifier(estimator=base_dt, n_estimators=n, learning_rate=lr,
random_state=42)
        ada_dt.fit(X_train, y_train)
        y_pred = ada_dt.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        print(f"  n_estimators={n}, learning_rate={lr:.1f} => Accuracy: {acc:.4f}")

# AdaBoost with LogisticRegression base estimator
print("\nAdaBoost with LogisticRegression base estimator:")
for n in n_estimators_list:
    for lr in learning_rates:
        base_lr = LogisticRegression(max_iter=1000)
        ada_lr = AdaBoostClassifier(estimator=base_lr, n_estimators=n, learning_rate=lr,
random_state=42)
        ada_lr.fit(X_train, y_train)
        y_pred = ada_lr.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        print(f"  n_estimators={n}, learning_rate={lr:.1f} => Accuracy: {acc:.4f}")

# GradientBoostingClassifier as an alternative to AdaBoost
```

```
print("\nGradient Boosting with Logistic Regression-like behavior:")
for n in n_estimators_list:
    for lr in learning_rates:
        gb = GradientBoostingClassifier(n_estimators=n, learning_rate=lr, random_state=42)
        gb.fit(X_train, y_train)
        y_pred = gb.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        print(f"  n_estimators={n}, learning_rate={lr:.1f} => Accuracy: {acc:.4f}")
```

Program 10

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



Lab 10:

Q. Algorithm of k-means

1. Initialize k cluster centroids randomly
2. Assign each data point to the nearest centroid
3. Recalculate the centroids as the mean of the points in each cluster
4. Repeat steps 2 and 3 until convergence (centroids don't change significantly or a maximum number of iterations is reached)

Q. Determining the number of cluster (k)

- Elbow Technique: Plot the SSE against the number of clusters. The elbow point
- Silhoutte Analysis: Measures how well each data point fits into its assigned cluster. Higher silhoutte scores indicate better clustering.
- Domain Knowledge: Prior understanding of the data can suggest a reasonable no. of clusters.

Q. Plot of SSE vs No. of clusters

⇒ The plot typically shows a decreasing trend. The "elbow" point is where the decrease rate diminishes

Q. kmeans() Parameters

→ n_clusters : No. of clusters (k)

→ init : Centroid initialization method ('k-means++', 'random', array)

⇒ n_init : Number of initializations per run.

→ max_iter : Max iterations per run.

→ tol : Convergence tolerance for centroid change.

→ random_state : Seed for random initialization (reproducibility).

⇒ algorithm : Algorithm to use ('lloyd', 'elkan', 'auto', 'full')

⇒ copy_x : If True, don't modify original data

→ n_jobs : Parallel processing cores (-1 for all)

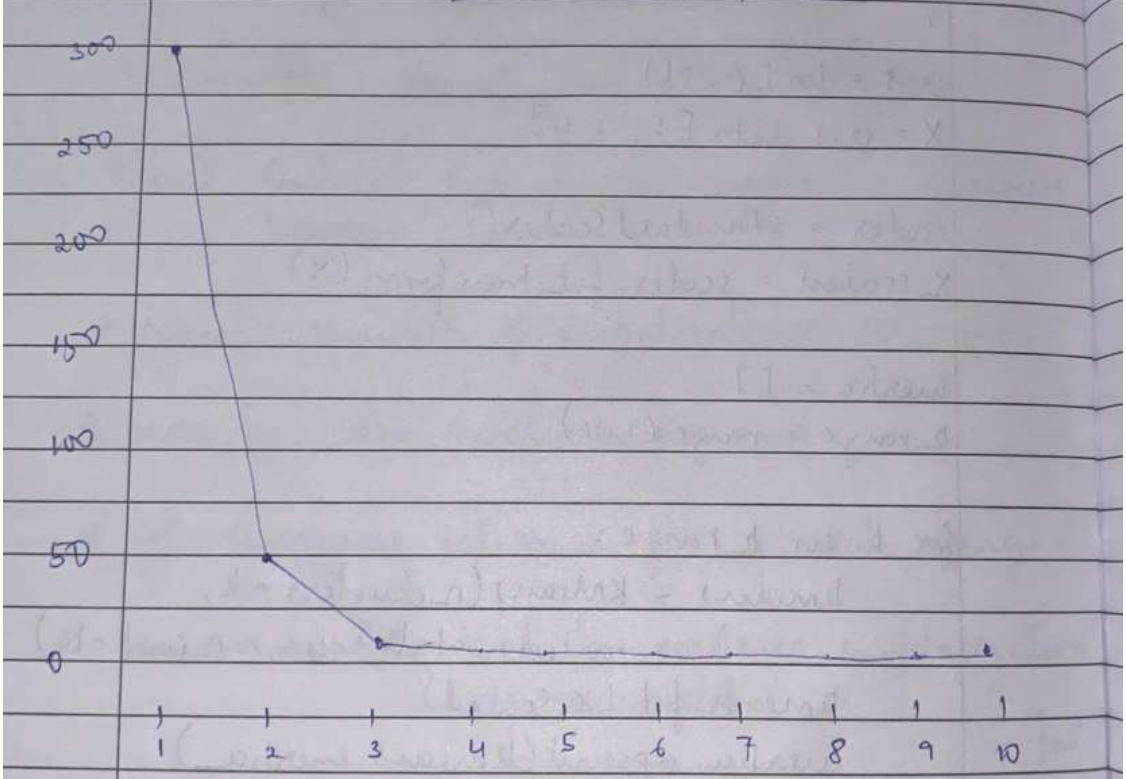
→ verbose : Verbosity level.

Code:

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

iris = load_iris()
X = iris.data[:, 2:4]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters = k,
                    random_state = 42, n_init = 10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize = (10, 6))
plt.plot(k_range, inertia, marker = 'o',
         linestyle = '-')
plt.title('Elbow Method for Optimal k')
plt.xlabel
plt.ylabel
plt.xticks
plt.grid
plt.show()
optimal_k = 3
```

print(f"Based on the elbow plot, the estimated optimal number of clusters (k) is: {optimal_k}")

output:

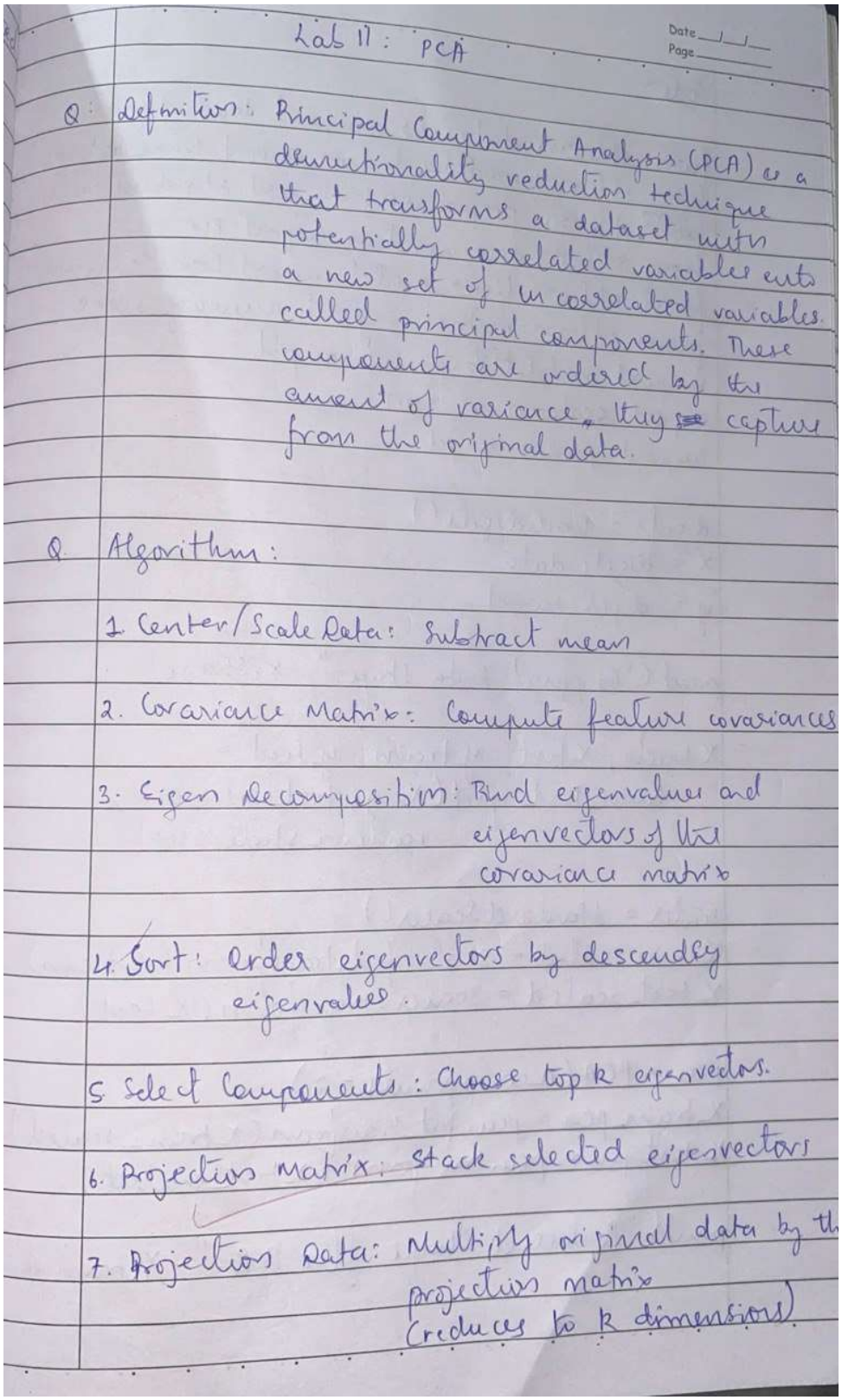


Code

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

# 1. Load the iris flower dataset and select petal features
iris = load_iris()
X = iris.data[:, 2:4]  # Petal length and width

# 2. Figure out if scaling would help
# Scaling is generally beneficial for KMeans as it's distance-based.
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Draw elbow plot and find the optimal value of k
inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(k_range, inertia, marker='o', linestyle='-')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.xticks(k_range)
plt.grid(True)
plt.show()

# Interpretation of the elbow plot:
# The "elbow" is the point where the rate of decrease in inertia starts to slow down.
# This point suggests a good trade-off between minimizing within-cluster variance
# and having a reasonable number of clusters.
# For the Iris dataset with petal length and width, you'll likely observe an elbow
# around k=2 or k=3.

# Based on the plot, determine the optimal k. For demonstration, let's assume it's 3.
optimal_k = 3
print(f"Based on the elbow plot, the estimated optimal number of clusters (k) is: {optimal_k}")

# You can now proceed to train a KMeans model with the determined optimal k
# and visualize the clusters.
```

Program 11

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



Lab 11 : PCA

Date __/__/__
Page

Q: Definition: Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms a dataset with potentially correlated variables into a new set of uncorrelated variables called principal components. These components are ordered by the amount of variance they capture from the original data.

Q Algorithm:

1. Center/Scale Data: Subtract mean
2. Covariance Matrix: Compute feature covariances
3. Eigen Decomposition: Find eigenvalues and eigenvectors of the covariance matrix
4. Sort: Order eigenvectors by descending eigenvalues.
5. Select Components: Choose top k eigenvectors.
6. Projection Matrix: Stack selected eigenvectors
7. Projection Data: Multiply original data by the projection matrix (reduces to k dimensions)

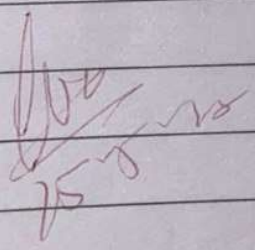
Code:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd.

digits = load_digits()
X = digits.data
y = digits.target

print("Original Data shape:", X.shape)

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)

pca = PCA(n_components = 2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Reduced shape after PCA:", X_train_pca.shape)
```

```
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_pca, y_train)

y_pred = log_reg.predict(X_test_pca)

accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy using PCA (2 component)
      and Logistic Regression", round
      round(accuracy * 100, 2), "%.")
```

Output

```
Original Data shape : (1797, 64)
Reduced shape after PCA : (1437, 2)
```



Code

```
# Required Libraries
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# -------------------------------------
# Step 1: Load the digits dataset
# -------------------------------------
digits = load_digits()
X = digits.data     # Shape: (1797, 64)
y = digits.target   # 10 classes: 0 to 9

print("Original Data Shape:", X.shape)

# -------------------------------------
# Step 2: Train-Test Split (80% train, 20% test)
# -------------------------------------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# -------------------------------------
# Step 3: Feature Scaling (Standardization)
# -------------------------------------
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -------------------------------------
# Step 4: Apply PCA with 2 components
# -------------------------------------
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Reduced Shape after PCA:", X_train_pca.shape)

# -------------------------------------
# Step 5: Train Logistic Regression on PCA-reduced data
# -------------------------------------
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_pca, y_train)
```

```
# Predict on test data
y_pred = log_reg.predict(X_test_pca)

# ------------------------------------
# Step 6: Accuracy Score
# ------------------------------------
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy using PCA (2 components) and Logistic Regression:", round(accuracy * 100, 2),
"%")

# ------------------------------------
# Optional: Visualize 2D PCA Representation
# ------------------------------------
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_test_pca[:, 0], y=X_test_pca[:, 1], hue=y_test, palette='tab10', legend='full')
plt.title("Digits Dataset Visualized in 2D using PCA")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```