VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Samarth Kumar Dubey (1BM22CS235)

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 **,** who is bonafide student Sagar bangari(1BM22CS231) 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.

Lab Faculty Incharge Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

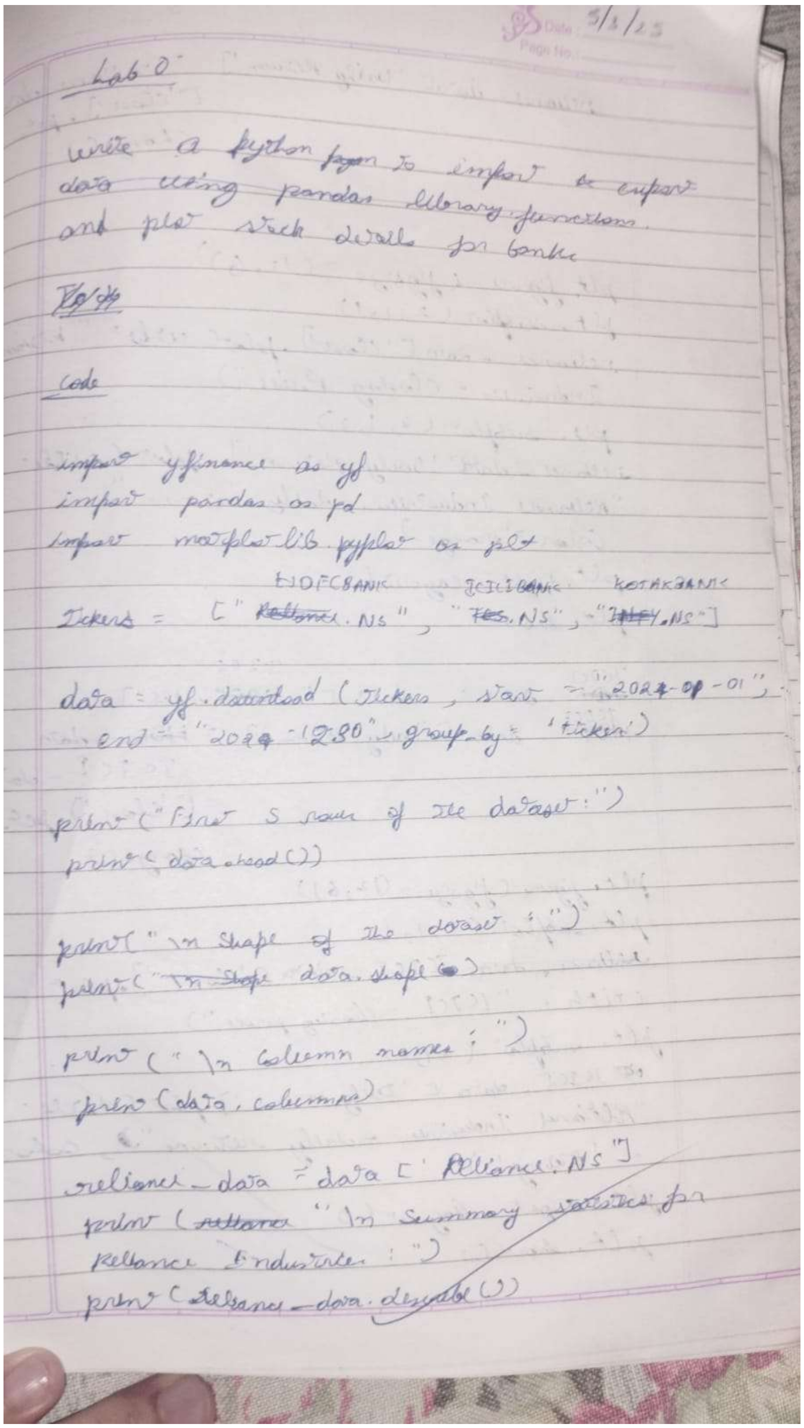
Index

Github Link:
https://github.com/Samarth512/ML_LAB

Program 1

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

Screenshot:



Date 5/3/25

Lab 0

Write a python ~~prgm~~ to import & export data using pandas library functions and plot stock details for banks

Code

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start="2024-09-01",
    end="2024-12-30", group_by='ticker')

print("First 5 rows of the dataset:")
print(data.head())

print("\n Shape of the dataset:")
print(data.shape)

print("\n Column names:")
print(data.columns)

reliance_data = data['Reliance.NS']
print("\n Summary statistics for
Reliance Industries:")
print(reliance_data.describe())
```

Date:
Page No.:

```
reliance_data['Daily Return'] = reliance_data
                                ['Close'].pct_
                                change()

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()

ICICI_data = data["ICICI.NS"]
ICICI_data['Daily Return'] = ICICI_data
                                ['Close'].pct_
                                change()

plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
ICICI_data['Close'].plot
(title="ICICI - Closing price")
plt.subplot(2,1,2)
ICICI_data['Daily Returns'].plot(title=
"Reliance Industries - daily returns", color
='orange')
plt.tight_layout()
plt.show()
```

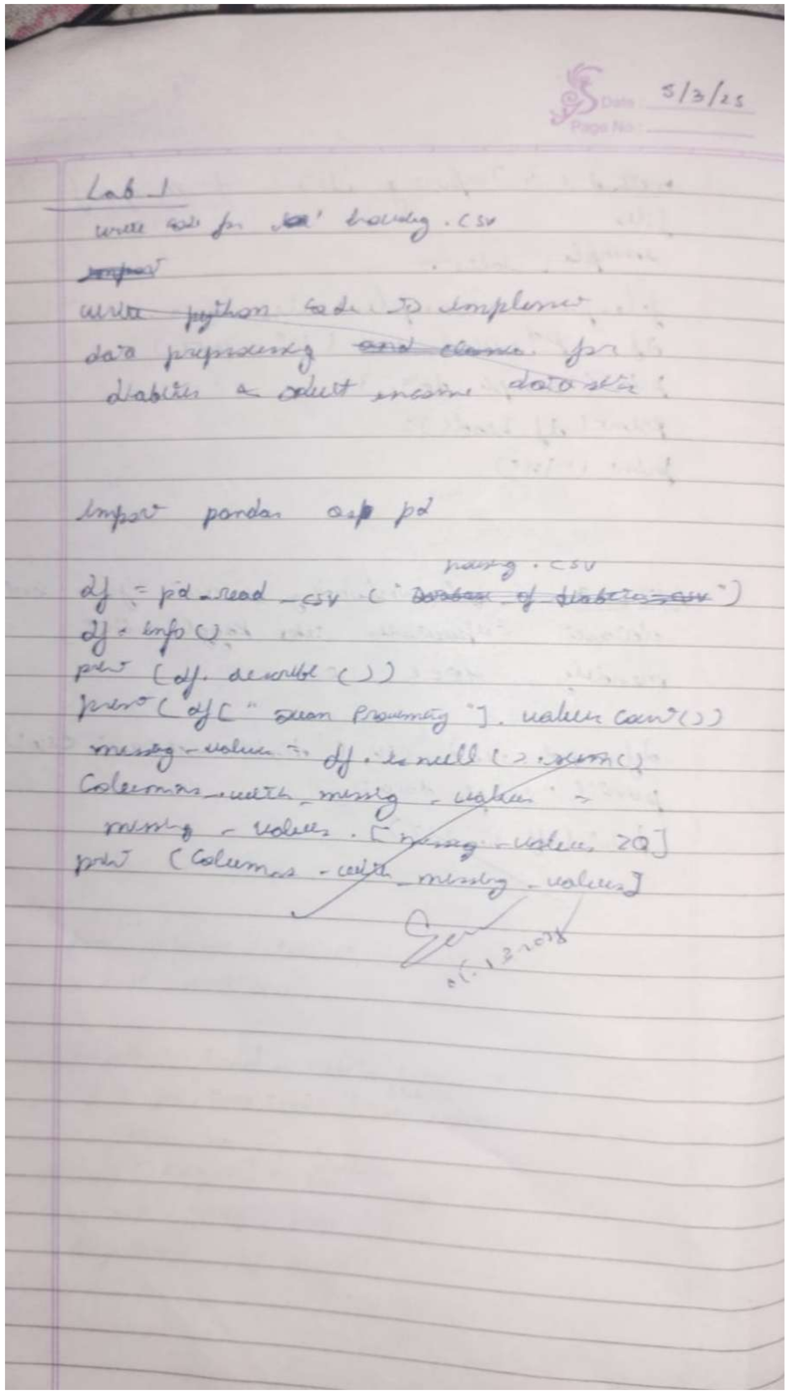
Code:

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')

hdfc_data = data['HDFCBANK.NS']
print("\nSummary statistics for HDFC bank :")
print(hdfc_data.describe())

hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc_data['Close'].plot(title="HDFC bank - Closing Price")
plt.subplot(2, 1, 2)
hdfc_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()


icici_data = data['ICICIBANK.NS']
print("\nSummary statistics for ICICI bank :")
print(icici_data.describe())

icici_data['Daily Return'] = icici_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
icici_data['Close'].plot(title="ICICI bank - Closing Price")
plt.subplot(2, 1, 2)
icici_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

```
      kotak_data = data['KOTAKBANK.NS']
      print("\nSummary statistics for KOTAK bank :")
      print(kotak_data.describe())

      kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
      plt.figure(figsize=(12, 6))
      plt.subplot(2, 1, 1)
      kotak_data['Close'].plot(title="KOTAK bank - Closing Price")
      plt.subplot(2, 1, 2)
      kotak_data['Daily Return'].plot(title="KOTAK Bank  - Daily Returns",
color='orange')
      plt.tight_layout()
      plt.show()
```

Program 2

Screenshot:



Date 5/3/25

Lab 1

write code for ~~...~~ housing.csv

~~import~~

write ~~python~~ code to implement data preprocessing ~~and cleaning~~ for diabetes & adult income datasets

```
import pandas as pd

df = pd.read_csv("housing.csv")
df.info()
print(df.describe())
print(df["Ocean Proximity"].value_counts())
missing_values = df.isnull().sum()
columns_with_missing_values = missing_values[missing_values > 0]
print(columns_with_missing_values)
```

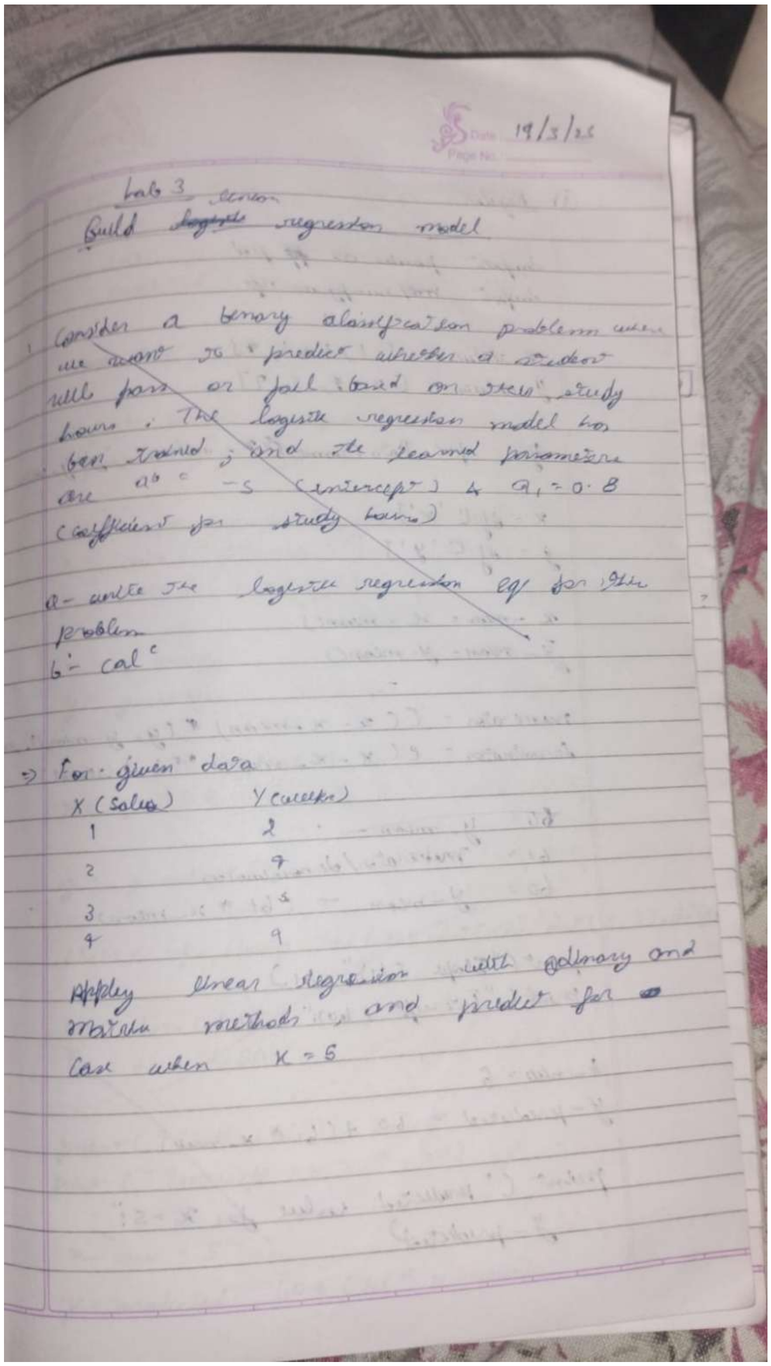
Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder



# Load dataset

diabetes = pd.read_csv('diabetes.csv')  # Replace with actual file path



diabetes.drop(columns=['ID', 'No_Pation'], inplace=True)  # Drop unnecessary columns



diabetes.dropna(inplace=True)  # Handle missing values



diabetes[['Gender', 'CLASS']] = diabetes[['Gender', 'CLASS']].apply(LabelEncoder().fit_transform)  # Encode categorical data



diabetes = diabetes[(np.abs((diabetes.select_dtypes(include=[np.number]) - diabetes.mean()) / diabetes.std()) < 3).all(axis=1)]  # Remove outliers



scaler = MinMaxScaler()

diabetes_scaled = pd.DataFrame(scaler.fit_transform(diabetes.drop(columns=['CLASS'])), columns=diabetes.columns[:-1])

diabetes_scaled['CLASS'] = diabetes['CLASS'].values  # Add target column back
```

```
print(diabetes_scaled.head())
```

Program 3

Screenshot:



Date: 19/3/26

Lab 3

Build ~~logistic~~ linear regression model

~~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) & $a_1 = 0.8$ (coefficient for study hours)~~

~~a- write the logistic regression eq for the problem~~

~~b:- cal^c~~

=> For given data

X (Sales)	Y (weeks)
1	2
2	4
3	5
4	9

Apply linear regression with ordinary and matrix methods and predict for case when $x = 6$

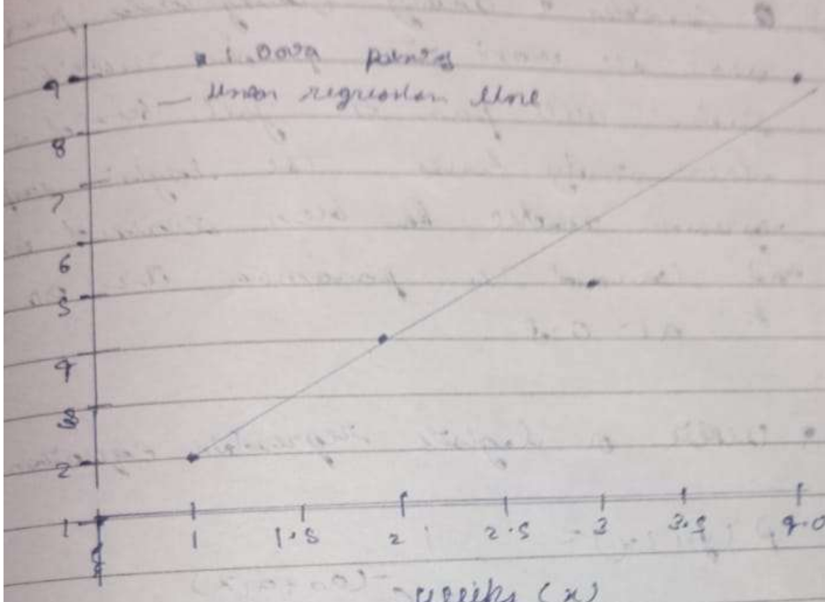
(1) Regular

```
import pandas as ~~pp~~ pd
import ~~mat~~ numpy as np

x_values = [1, 2, 3, 4]
y_values = [2, 4, 5, 9]

df = pd.DataFrame({x:x_values, y:y_values})

x = df['x']
y = df['y']

x_mean = x.mean()
y_mean = y.mean()

numerator = ((x - x_mean) * (y - y_mean)).sum()
denominator = ((x - x_mean) ** 2).sum()

~~b0 = y_mean~~
b1 = numerator/denominator
b0 = y_mean - (b1 * x_mean)

print("Slope (b1):", b1)
print("Intercept (b0):", b0)

x_new = 5
y_predicted = b0 + (b1 * x_new)

print("predicted value for x = 5:",
   y_predicted)
```

O/P

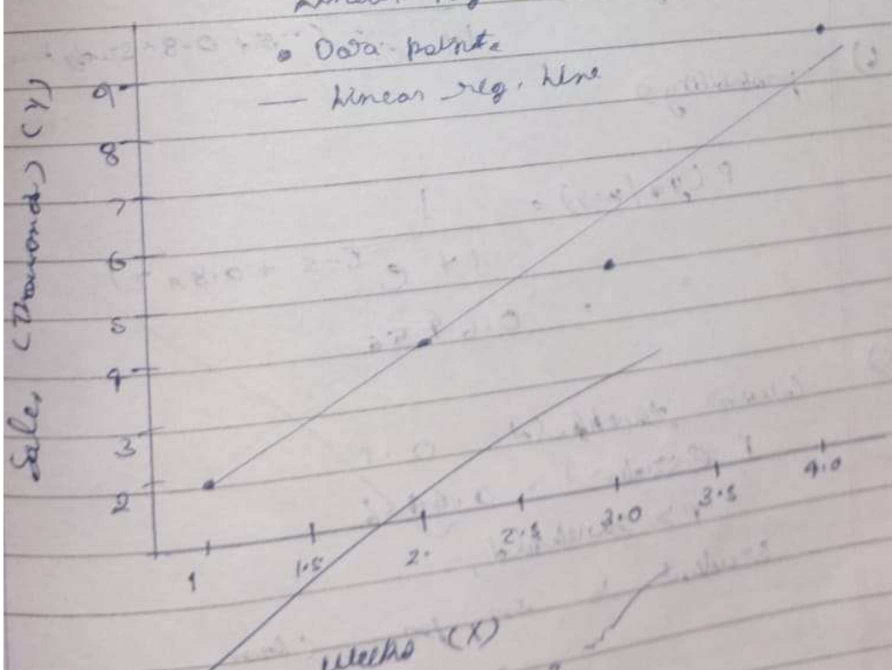
for regular

Linear Regression : Weeks vs Sales



for marks

Linear Regression : Weeks vs Sales



19.02

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt



# Load the dataset from CSV file

dataset = pd.read_csv('/content/sales.csv')



# Display the first few rows of the dataset

print(dataset.head())



# Assuming the CSV has columns 'Week' and 'Sales' (adjust based on your actual column names)

weeks = dataset['xi(week)'].values

sales = dataset['yi(Sales in thousands)'].values



# Reshaping weeks for matrix operations (make it a column vector)

X = weeks.reshape(-1, 1)

y = sales.reshape(-1, 1)



# Add a column of ones to X to account for the intercept (b)

X_b = np.c_[np.ones((len(X), 1)), X]  # The "1" is for the bias term (intercept)
```

```python
# Compute theta using the normal equation: θ = (X^T X)^(-1) X^T y

theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)


# Extract slope (m) and intercept (b) from theta

b = theta[0]

m = theta[1]


# Print the regression line equation

print(f"The regression equation is: y = {m[0]:.2f}x + {b[0]:.2f}")


# Predict the sales for 7th and 9th weeks

week_7 = 7

week_9 = 9


predicted_sales_7 = m * week_7 + b

predicted_sales_9 = m * week_9 + b


print(f"Predicted sales for the 7th week: {predicted_sales_7[0]:.2f}
thousand")

print(f"Predicted sales for the 9th week: {predicted_sales_9[0]:.2f}
thousand")
```

```
# Plot the data points and the regression line

plt.scatter(weeks, sales, color='blue', label='Data Points')

plt.plot(weeks, m * weeks + b, color='red', label=f'Linear Regression: y = {m[0]:.2f}x + {b[0]:.2f}')

plt.xlabel('Weeks')

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

plt.title('Sales Data and Linear Regression')

plt.legend()

plt.show()
```

Program 4

Screenshot:

.../4/25

Lab 4

Logistic regression

1) 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 on and learned the parameters are $a_0 = -5$ & $a_1 = 0.8$

2) Write a logistic regression equation

$$P(y=1 \mid x) = \frac{1}{1+e^{-(a_0+a_1 x)}}$$

where x = study hours

$$\Rightarrow P(y=1 \mid x) = \frac{1}{1+e^{-(-5+0.8 \times \text{study hrs})}}$$

probability ⇒

$$P(y=1 \mid x=7) = \frac{1}{1+e^{-(-5+0.8\times 7)}}$$

$$= 0.6456$$

Given threshold = 0.5

P (student) = 0.6456

P > threshold,

Student is in pass class

2) Given classes $\Rightarrow$ $z = [2, 1, 0]$

A. $e^{z} \Rightarrow [e^{2}, e^{1}, e^{0}]$

$\Rightarrow [7.389, 2.718, 1]$

$\Sigma e^{z} \Rightarrow 11.107$

Probabilities $\Rightarrow$ ~~11.109~~ $[0.665, 0.244, 0.09]$

$\Rightarrow$ For dataset file "HR_comma_sep.csv".

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

A. Key variable are

$\Rightarrow$ Satisfaction level $\Rightarrow$ Strong negative correlation

$\Rightarrow$ Time spent in company $\Rightarrow$ Positive correlation

$\Rightarrow$ Work accident $\Rightarrow$ Negative correlation

$\Rightarrow$ Salary $\Rightarrow$ lower salary leads to more leaves

$\Rightarrow$ Department $\Rightarrow$ Retention varies for departments

$\Rightarrow$ Model accuracy $\Rightarrow$ Accuracy is 78.58% which is concerning and indicates of an imbalanced dataset, many leaves are missed.

Class ~~repeat~~ ~~was~~ Big was misclassified once. So could be because

a) Limited amount of data

b) Feature similarity in Big and Bird class.

Code:

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score



# Load dataset

file_path = "/content/HR_comma_sep"

df = pd.read_csv(file_path)



# Exploratory Data Analysis

# Plot bar chart showing impact of salaries on retention

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

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

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

plt.xlabel("Salary Level")

plt.ylabel("Number of Employees")

plt.legend(["Stayed", "Left"])

plt.show()
```

```
# Plot bar chart showing correlation between department and employee
retention

plt.figure(figsize=(10, 5))

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

plt.title("Department-wise Employee Retention")

plt.xlabel("Department")

plt.ylabel("Number of Employees")

plt.xticks(rotation=45)

plt.legend(["Stayed", "Left"])

plt.show()



# Selecting key features based on analysis

X = df[['satisfaction_level', 'time_spend_company', 'Work_accident',
'salary']]

X = pd.get_dummies(X, columns=['salary'], drop_first=True)  # Convert
categorical 'salary' to numerical

y = df['left']



# Splitting data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.9,
random_state=10)



# Train logistic regression model

model = LogisticRegression(max_iter=1000)
```

```python
model.fit(X_train, y_train)



# Predictions

y_predicted = model.predict(X_test)



# Model accuracy

accuracy = accuracy_score(y_test, y_predicted)

print(f"Model Accuracy: {accuracy:.4f}")



# Plotting actual vs predicted values

plt.figure(figsize=(6, 4))

sns.heatmap(pd.crosstab(y_test, y_predicted), annot=True, fmt='d',
cmap='Blues')

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()



# Predict probability of an employee leaving

def predict_leave_probability(features):

    return model.predict_proba([features])[0][1]  # Probability of leaving



# Example prediction
```

```
example_employee = [[0.5, 3, 0, 1, 0]]  # Sample feature values with one-hot encoded salary

probability = predict_leave_probability(example_employee[0])

print(f"Probability of leaving: {probability:.4f}")
```

Program 5

Screenshot:

Date 2/4/25

Lab 5

KNN classification model

=> Consider the following dataset, for K=3 and test data (X, 35, 100) as (Person, Age, Salary K) solve using KNN classifier model and predict the target.

Person	Age	Salary K	Target
A	18	50	N
B	23	55	N
C	24	70	N
D	41	60	Y
E	43	70	Y
F	38	40	Y
X	35	100	?

=> Making a distance table

Age	Salary K	Target	Distance
18	50	N	52.81
23	55	N	46.57
24	70	N	31.953
41	60	Y	40.44
43	70	Y	31.048
38	40	Y	60.075
~~35~~	~~100~~		

Closest neighbours -> E, C, D

Classes => [Y, N, Y]

=> majority = Y, so (X, 35, 100) belongs

For iris dataset

How to choose the K value? Determine using accuracy rate & error rate.

Steps are

i) split dataset into training and testing set.

ii) Train K-NN with different K values like 1, 3, 5, 7 ...

iii) Calculate accuracy

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{Total prediction}} \times 100$$

iv) Calculate error rate => Error rate = 1 - accuracy rate

v) plot accuracy of K

For diabetes dataset

What is feature scaling and purpose of it? How to perform it?

Purpose =>

i) Diabetes dataset has features like glucose, BMI, age, which have different ranges

ii) Machine learning algos perform better when features are on similar scale

=> It helps increase convergence speed & prevents bias from dominant features

To perform, we have two ways =>

i) <u>Min-Max scaling</u> =>

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

scales b/w 0 to 1

) <u>Standardization</u>

Converts to mean 0 & standard deviation 1

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

09.04.2024

Code:

```
import pandas as pd

import numpy as np



# Sample weather dataset

data = {

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

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

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

    'Windy': ['No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 
'Yes', 'Yes', 'No', 'Yes'],

    'Play Tennis?': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 
'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}



# Convert to DataFrame

df = pd.DataFrame(data)



# Function to calculate entropy

def entropy(target):

    # Get the counts of each class
```

```python
    class_counts = target.value_counts()

    # Calculate the entropy using the formula

    probabilities = class_counts / len(target)

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


# Function to calculate information gain

def information_gain(data, feature, target):

    # Calculate the entropy of the whole dataset

    entropy_before = entropy(target)


    # Get the unique values of the feature

    feature_values = data[feature].unique()


    # Calculate the weighted entropy after the split

    weighted_entropy = 0

    for value in feature_values:

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

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


    # Information gain is the reduction in entropy

    return entropy_before - weighted_entropy
```

```python
# Function to print entropy and information gain for each feature

def print_entropy_and_gain(data, features, target):

    print("\nEntropy and Information Gain for each feature:")

    for feature in features:

        gain = information_gain(data, feature, target)

        ent = entropy(target)

        print(f"Feature: {feature} | Entropy: {ent:.4f} | Information Gain: {gain:.4f}")



# Function to build the decision tree recursively

def build_tree(data, target, features):

    # Base case: If all target values are the same, return a leaf node

    if len(target.unique()) == 1:

        return target.iloc[0]



    # Base case: If no features left to split, return the majority class

    if len(features) == 0:

        return target.mode()[0]



    # Calculate information gain for each feature

    gains = {feature: information_gain(data, feature, target) for feature in features}

```

```
    # Find the feature with the highest information gain
    best_feature = max(gains, key=gains.get)

    # Create the tree node with the best feature
    tree = {best_feature: {}}

    # Get the unique values of the best feature
    feature_values = data[best_feature].unique()

    # Recursively build the tree for each subset of the data
    for value in feature_values:
        subset_data = data[data[best_feature] == value]
        subset_target = target[data[best_feature] == value]

        # Remove the best feature from the list of features for the next level
        remaining_features = [f for f in features if f != best_feature]

        # Build the subtree for the subset
        subtree = build_tree(subset_data, subset_target, remaining_features)

        # Add the subtree to the tree
        tree[best_feature][value] = subtree
```

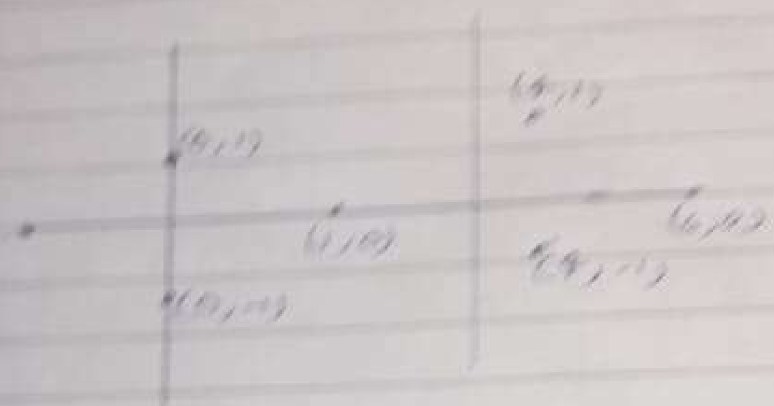
```python
    return tree


# Function to print the tree in a visually structured way
def print_tree(tree, indent=""):
    if isinstance(tree, dict):
        for feature, branches in tree.items():
            print(f"{indent}{feature}:")
            for value, subtree in branches.items():
                print(f"{indent}  {value} ->", end=" ")
                print_tree(subtree, indent + "    ")
    else:
        print(f"{indent}{tree}")


# Target variable
target = df['Play Tennis?']


# Features
features = ['Outlook', 'Temperature', 'Humidity', 'Windy']


# Step 1: Print entropy and information gain for each feature
print_entropy_and_gain(df, features, target)
```

```python
# Step 2: Build the decision tree
tree = build_tree(df, target, features)

# Step 3: Print the decision tree (formatted)
print("\nDecision Tree:")
print_tree(tree, indent="    ")
```

Program 6

Screenshot:

Points (4,1), (4,-1) & (6,0) belong to positive class; points (1,0), (0,1) and (0,-1) belong to negative class. Draw an optimal hyperplane.



$s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $s_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$ $s_3 = \begin{pmatrix} 4 \\ -1 \end{pmatrix}$

$\tilde{s}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$, $\tilde{s}_2 = \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}$ $\tilde{s}_3 = \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}$

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

On solving, $\alpha_1 = -\frac{22}{9}$, $\alpha_2 = \frac{7}{18}$, $\alpha_3 = \frac{7}{18}$

$w = \sum \alpha_i \tilde{s}_i$

$= \begin{pmatrix} 2/9 \\ 0 \\ [illegible] \end{pmatrix}$

$z = (\frac{2}{3}) x_1 + (3) x_2 + \frac{-3}{3} = 6$

$2x_1 = 5$

$x_1 = 2.5$

boundary at $x_1 = 2.5$

Q. Iris .csv dataset, what is accuracy of classifier using linear and RBF kernel? which of the two has better performance?

Linear kernel = 100%

RBF kernel = 100%

Generally RBF perform better but since we had small dataset, we get equal performance.

RBF is more suitable as iris dataset is non linear.

Q. "letter recognition .csv" dataset, present & interpret the confusion matrix. Are there any specific letters frequently confused with others? How does the SVM model perform compared to Iris dataset.

~~[illegible]~~ The confusion matrix shows how each letter was predicted [illegible]

Code:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report



# Load the Iris dataset

iris = pd.read_csv("/content/iris.csv")

X_iris = iris.drop(columns=['species'])

y_iris = iris['species']



# Split the dataset into training and testing sets

X_train_iris, X_test_iris, y_train_iris, y_test_iris =
train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)



# Choose the best k value (testing k from 1 to 20)

k_values = range(1, 21)

accuracy_list = []
```

```python
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_iris, y_train_iris)
    y_pred = knn.predict(X_test_iris)
    accuracy_list.append(accuracy_score(y_test_iris, y_pred))

best_k_iris = k_values[np.argmax(accuracy_list)]
print(f"Optimal k value for Iris dataset: {best_k_iris}")

# Train KNN with the best k value
knn_iris = KNeighborsClassifier(n_neighbors=best_k_iris)
knn_iris.fit(X_train_iris, y_train_iris)
y_pred_iris = knn_iris.predict(X_test_iris)

# Display Accuracy and Confusion Matrix for Iris dataset
print("Iris Dataset Accuracy:", accuracy_score(y_test_iris, y_pred_iris))
print("Confusion Matrix for Iris:")
print(confusion_matrix(y_test_iris, y_pred_iris))
print("Classification Report for Iris:")
print(classification_report(y_test_iris, y_pred_iris))

# Load the Diabetes dataset
```

```python
diabetes = pd.read_csv("/content/diabetes.csv")

X_diabetes = diabetes.drop(columns=['Outcome'])

y_diabetes = diabetes['Outcome']



# Split into training and testing

X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes =
train_test_split(X_diabetes, y_diabetes, test_size=0.2, random_state=42)



# Feature Scaling

scaler = StandardScaler()

X_train_diabetes = scaler.fit_transform(X_train_diabetes)

X_test_diabetes = scaler.transform(X_test_diabetes)



# Train KNN Classifier for Diabetes dataset

best_k_diabetes = 7  # Assume 7 as a reasonable k value (can be tuned
further)

knn_diabetes = KNeighborsClassifier(n_neighbors=best_k_diabetes)

knn_diabetes.fit(X_train_diabetes, y_train_diabetes)

y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)



# Display Accuracy and Confusion Matrix for Diabetes dataset

print("Diabetes Dataset Accuracy:", accuracy_score(y_test_diabetes,
y_pred_diabetes))

print("Confusion Matrix for Diabetes:")
```

```
print(confusion_matrix(y_test_diabetes, y_pred_diabetes))

print("Classification Report for Diabetes:")

print(classification_report(y_test_diabetes, y_pred_diabetes))
```

Program 7

Screenshot:

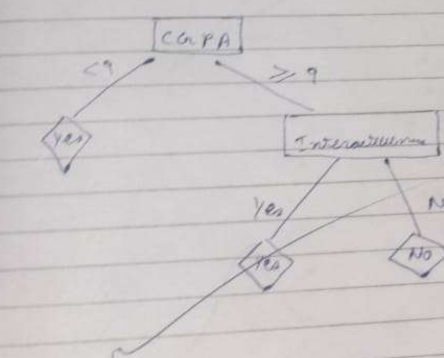
7/3/25

Lab 7

Q. Implement Random forest ensemble method.

Q. For sample S1, draw decision tree considering CGPA as root

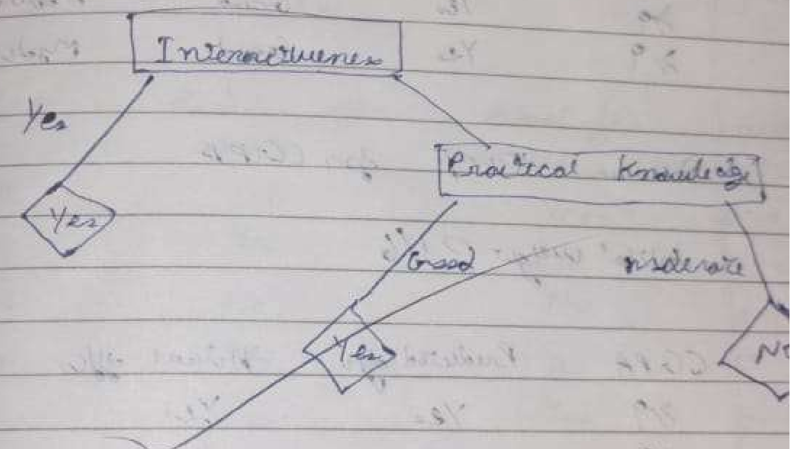
Sno	CGPA	Interactiveness	Comm^n skill	Practical Knowledge	Job offer
1	≥ 9	Yes	Good	Good	Yes
2	< 9	NO	Moderate	Good	Yes
3	≥ 9	No	Moderate	Average	No
4	≥ 9	No	Moderate	Average	No
5	≥ 9	Yes	Moderate	Good	Yes



Date: 7/6/15
Page No.

sample SA

S.No	CGPA	Interactiveness	Comm^n skills	Prac. Know	Job
	<9	No	Moderate	Good	Yes
2	≥9	NO	Moderate	Average	No
3	≥9	No	Moderate	Average	No
3	≥9	Yes	Moderate	Good	Yes
5	≥9	Yes	Moderate	Good	Yes



Code:

```
import numpy as np

import matplotlib.pyplot as plt



class SVM:

    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):

        self.lr = learning_rate

        self.lambda_param = lambda_param

        self.n_iters = n_iters

        self.w = None

        self.b = None



    def fit(self, X, y):

        y = np.where(y <= 0, -1, 1) # Convert labels to -1 and 1

        n_samples, n_features = X.shape

        self.w = np.zeros(n_features)

        self.b = 0



        for _ in range(self.n_iters):

            for idx, x_i in enumerate(X):

                condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1

                if condition:
```

```
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                    self.b += self.lr * y[idx]


    def predict(self, X):
        approx = np.dot(X, self.w) + self.b
        return np.sign(approx)


    def visualize(self, X, y, new_point=None, prediction=None):
        def get_hyperplane(x, w, b, offset):
            return (-w[0] * x + b + offset) / w[1]


        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)


        # Plot existing data points
        for i, sample in enumerate(X):
            if y[i] == 1:
                plt.scatter(sample[0], sample[1], marker='o', color='blue', label='Class +1' if i == 0 else "")
            else:
```

```
                plt.scatter(sample[0], sample[1], marker='x', color='red',
label='Class -1' if i == 0 else "")


        # Plot decision boundary

        x0 = np.linspace(np.min(X[:, 0])-1, np.max(X[:, 0])+1, 100)

        x1 = get_hyperplane(x0, self.w, self.b, 0)

        x1_m = get_hyperplane(x0, self.w, self.b, -1)

        x1_p = get_hyperplane(x0, self.w, self.b, 1)


        ax.plot(x0, x1, 'k-', label='Decision Boundary')

        ax.plot(x0, x1_m, 'k--', label='Margins')

        ax.plot(x0, x1_p, 'k--')


        # Plot the new point

        if new_point is not None:

            color = 'green' if prediction == 1 else 'orange'

            label = f'New Point: Class {"1" if prediction == 1 else "0"}'

            plt.scatter(new_point[0], new_point[1], c=color, s=100,
edgecolors='black', label=label, marker='*')


        ax.legend()

        plt.xlabel("Feature 1")

        plt.ylabel("Feature 2")
```

```python
        plt.title("SVM with New Point Prediction")
        plt.grid(True)
        plt.show()



if __name__ == "__main__":
    # Training data
    X = np.array([
        [1, 0],
        [0, 1],
        [0, -1],
        [4, -1],
        [4, 1],
        [6, 0]
    ])
    y = np.array([0, 0, 0, 1, 1, 1]) # 0 -> -1, 1 -> +1

    # New point to classify
    new_point = np.array([[5, 5]])

    # Train and predict
```

```
    svm = SVM()

    svm.fit(X, y)

    prediction = svm.predict(new_point)[0]



    # Visualize

    svm.visualize(X, y, new_point=new_point[0], prediction=prediction)



    # Print prediction

    print(f"New point {new_point[0]} classified as: {'Class 1' if prediction
== 1 else 'Class 0'}")
```

Program 8

Screenshot:

Lab 8

Considering Adaboost Algorithm, for following sample data, show decision stumps.

CGPA	Internship	Proj. Know.	Comm. skill	Job offer
≥ 9	Yes	Good	Good	Yes
< 9	No	Good	Moderate	Yes
≥ 9	No	Average	Moderate	No
< 9	No	Average	Good	No
≥ 9	Yes	Good	Moderate	Yes
≥ 9	Yes	Good	Moderate	Yes

Decision stump for CGPA

Initial weight => 1/6

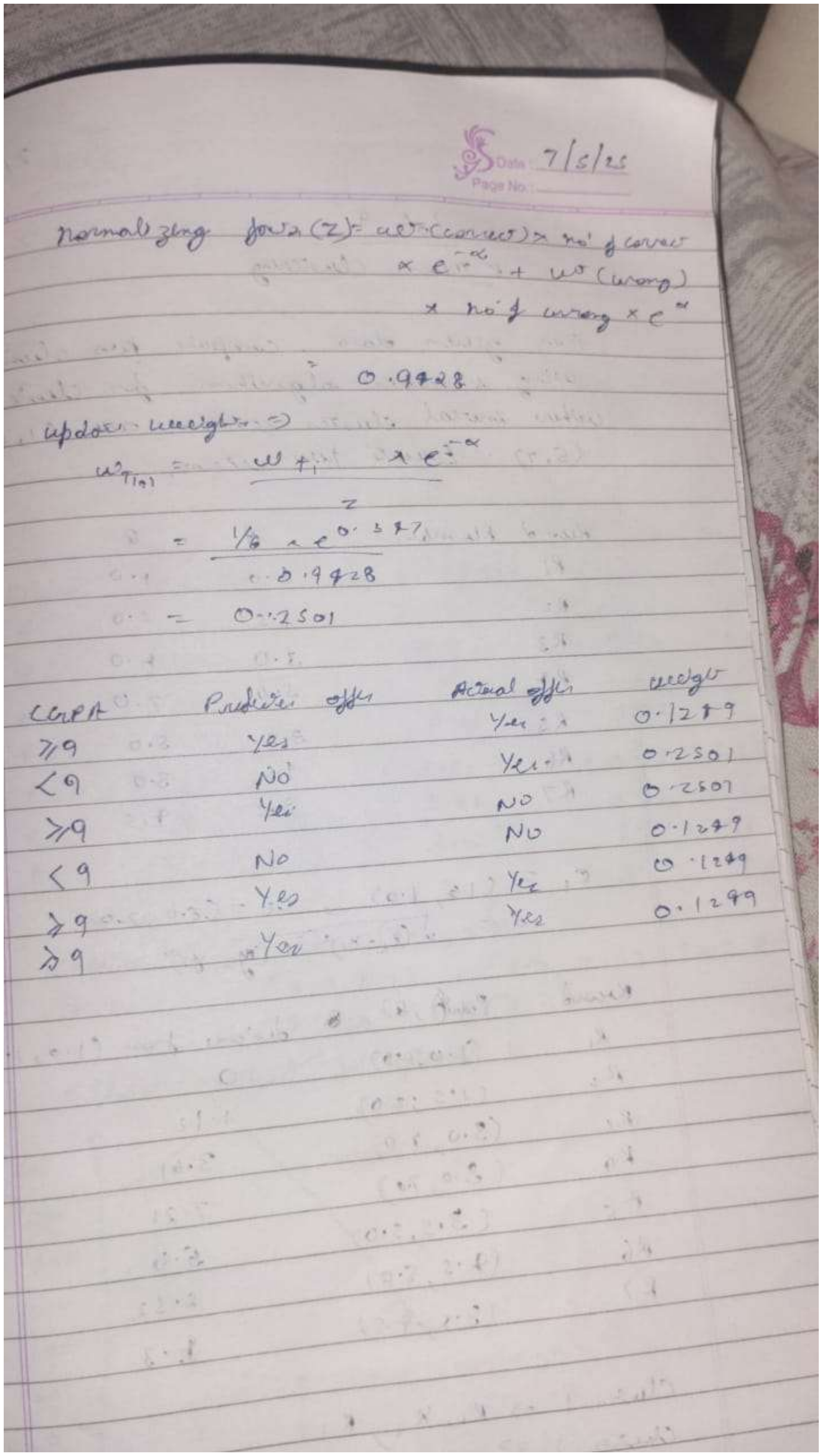
CGPA	Predicted offer	Actual offer	weight
≥ 9	Yes	Yes	1/6
< 9	No	Yes	1/6
≥ 9	Yes	No	1/6
< 9	No	No	1/6
≥ 9	Yes	Yes	1/6
≥ 9	Yes	Yes	1/6

weighted error => ϵ = no. of incorrect × weight of incorrect

$= 2 \times 1/6 = 0.333$

weight of classifier $\alpha \Rightarrow \frac{1}{2} \ln\left(\frac{1-\epsilon}{\epsilon}\right)$

$\Rightarrow 0.347$



Date: 7/5/25

Page No:

normalizing factor (Z) = w° (correct) × no. of correct × $e^{-\alpha}$ + w° (wrong) × no. of wrong × e^{α}

= 0.9428

update weights ⇒

$w_{T(n)} = \dfrac{w_{T_1} \times e^{-\alpha}}{Z}$

$= \dfrac{1/6 \times e^{0.347}}{0.9428}$

= 0.2501

CGPA	Predicted offer	Actual offer	weight
≥9	Yes	Yes	0.1249
<9	No	Yes	0.2501
≥9	Yes	No	0.2501
<9	No	No	0.1249
≥9	Yes	Yes	0.1249
≥9	Yes	Yes	0.1249
≥9	Yes		

Code:

```python
import pandas as pd
```

```python
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns


# Load dataset

df = pd.read_csv("/content/iris (4).csv")


# Features and target

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

y = df.iloc[:, -1]


# Train/test split

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


# ---------------------------------------------

# 1. Default Random Forest with 10 estimators

# ---------------------------------------------

clf_default = RandomForestClassifier(n_estimators=10, random_state=42)

clf_default.fit(X_train, y_train)

y_pred_default = clf_default.predict(X_test)
```

```
default_accuracy = accuracy_score(y_test, y_pred_default)

default_cm = confusion_matrix(y_test, y_pred_default)


print("=== Default Model (10 trees) ===")

print(f"Accuracy: {default_accuracy:.4f}")

print("Confusion Matrix:")

print(default_cm)


# ---------------------------------------------

# 2. Fine-tune number of trees for best accuracy

# ---------------------------------------------

best_score = 0

best_n = 0

best_model = None

best_cm = None


scores = []

n_estimators_range = range(1, 101)


for n in n_estimators_range:

    clf = RandomForestClassifier(n_estimators=n, random_state=42)

    clf.fit(X_train, y_train)
```

```
    y_pred = clf.predict(X_test)

    score = accuracy_score(y_test, y_pred)

    scores.append(score)


    if score > best_score:

        best_score = score

        best_n = n

        best_model = clf

        best_cm = confusion_matrix(y_test, y_pred)


# Print best result

print("\n=== Tuned Model ===")

print(f"Best Accuracy: {best_score:.4f}")

print(f"Best Number of Trees: {best_n}")

print("Confusion Matrix:")

print(best_cm)


# Plot accuracy vs number of trees

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

plt.plot(n_estimators_range, scores, marker='o')

plt.xlabel("Number of Trees")

plt.ylabel("Accuracy")
```

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

plt.grid(True)

plt.show()



# Plot best confusion matrix

plt.figure(figsize=(6, 4))

sns.heatmap(best_cm, annot=True, fmt="d", cmap="Blues",
xticklabels=y.unique(), yticklabels=y.unique())

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Best Confusion Matrix")

plt.show()
```

Program 9

Screenshot:

7/5/25

Lab 9
K Mean Clustering

For given data, compute two clusters using k-means algorithm for clustering where initial cluster centers are (1, 1) & (5, 7). Execute two iteration.

Record Number	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

$C_1 = (1.0, 1.0)$, $C_2 = (5.0, 7.0)$

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

Record	Point (A, B)	distance from (1.0, 1.0)
R_1	(1.0, 1.0)	0
R_2	(1.5, 2.0)	1.12
R_3	(3.0, 4.0)	3.61
R_4	(5.0, 7.0)	7.21
R_5	(3.5, 5.0)	5.0
R_6	(4.5, 5.0)	5.32
R_7	(3.5, 4.5)	4.3

Cluster 1 => R_1, R_2, R_3
Cluster 2 => R_4, R_5, R_6, R_7

$C_1 \Rightarrow \left(\frac{1 + 1.5 + 3}{3}, \frac{1 + 2 + 4}{3} \right)$

$\Rightarrow (1.83, 2.33)$

$C_2 \Rightarrow \left(\frac{5 + 3.5 + 4.5 + 3.5}{4}, \frac{7.0 + 5.0 + 5.0 + 4.5}{4} \right)$

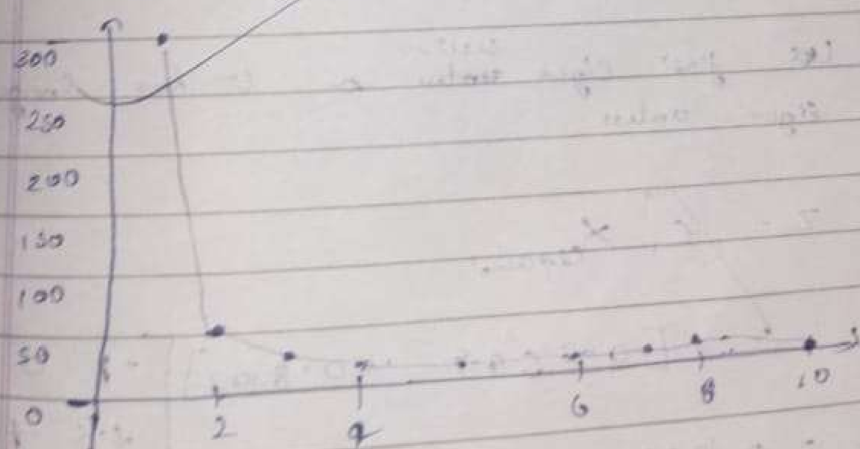
$\Rightarrow (4.125, 5.375)$

Round	Point (A, B)	distance (1.83, 2.33)	dist (4.12, 5.38)	Cl
R1	(1.0, 1.0)	1.57	5.62	C1
R2	(1.5, 2.0)	0.47	4.52	C1
R3	(3.0, 4.0)	2.12	1.63	C2
R4	(5.0, 7.0)	5.71	1.85	C2
R5	(3.5, 5.0)	3.57	0.78	C2
R6	(4.5, 5.0)	3.42	0.53	C2
R7	(3.5, 4.5)	3.07	1.01	C2

cluster 1 $\Rightarrow$ (1, 1), (1.5, 2.0)

cluster 2 $\Rightarrow$ (3.0, 4.0), (5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)

Elbow method for optimal K



Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns



# Load dataset

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



# Feature and target split

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

y = df.iloc[:, -1]



# Handle categorical variables (if any)

X = pd.get_dummies(X)

y = pd.factorize(y)[0]



# Train/test split

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

```
# ------------------------

# 1. Default AdaBoost model (10 estimators)

# ------------------------

clf_default = AdaBoostClassifier(n_estimators=10, random_state=42)

clf_default.fit(X_train, y_train)

y_pred_default = clf_default.predict(X_test)

default_accuracy = accuracy_score(y_test, y_pred_default)

default_cm = confusion_matrix(y_test, y_pred_default)


print("=== Default AdaBoost Model (10 estimators) ===")

print(f"Accuracy: {default_accuracy:.4f}")

print("Confusion Matrix:")

print(default_cm)



# ------------------------

# 2. Fine-tune n_estimators

# ------------------------

best_score = 0

best_n = 0

best_cm = None

scores = []
```

```python
for n in range(1, 101):

    clf = AdaBoostClassifier(n_estimators=n, random_state=42)

    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    score = accuracy_score(y_test, y_pred)

    scores.append(score)


    if score > best_score:

        best_score = score

        best_n = n

        best_cm = confusion_matrix(y_test, y_pred)


print("\n=== Best AdaBoost Model ===")

print(f"Best Accuracy: {best_score:.4f}")

print(f"Best Number of Estimators: {best_n}")

print("Confusion Matrix:")

print(best_cm)


# Plot accuracy vs number of estimators

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

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

```
plt.xlabel("Number of Estimators")

plt.ylabel("Accuracy")

plt.title("AdaBoost Accuracy vs Number of Estimators")

plt.grid(True)

plt.show()



# Plot best confusion matrix

plt.figure(figsize=(6, 4))

sns.heatmap(best_cm, annot=True, fmt="d", cmap="Blues")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Best Confusion Matrix")

plt.show()
```

Program 10

Screenshot:

7/5/25

Lab 9

K mean Clustering

For given data, compute two clusters using k-means algorithm for clustering where initial cluster centers are (1, 1) & (5, 7). Execute two iterations.

Record Number	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

$C_1 = (1.0, 1.0)$, $C_2 = (5.0, 7.0)$

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

Record	Point (A, B)	distance from (1.0, 1.0)
R_1	(1.0, 1.0)	0
R_2	(1.5, 2.0)	1.12
R_3	(3.0, 4.0)	3.61
R_4	(5.0, 7.0)	7.21
R_5	(3.5, 5.0)	5.0
R_6	(4.5, 5.0)	5.32
R_7	(3.5, 4.5)	4.3

Cluster 1 => R_1, R_2, R_3

Cluster 2 => R_4, R_5, R_6, R_7

$C_1 \rightarrow \left(\frac{1 + 1.5 + 3}{3} , \frac{1 + 2 + 4}{3} \right)$

$\rightarrow (1.83, 2.33)$

$C_2 \Rightarrow \left(\frac{5 + 3.5 + 4.5 + 3.5}{4} , \frac{7.0 + 5.0 + 5.0 + 4.5}{4} \right)$

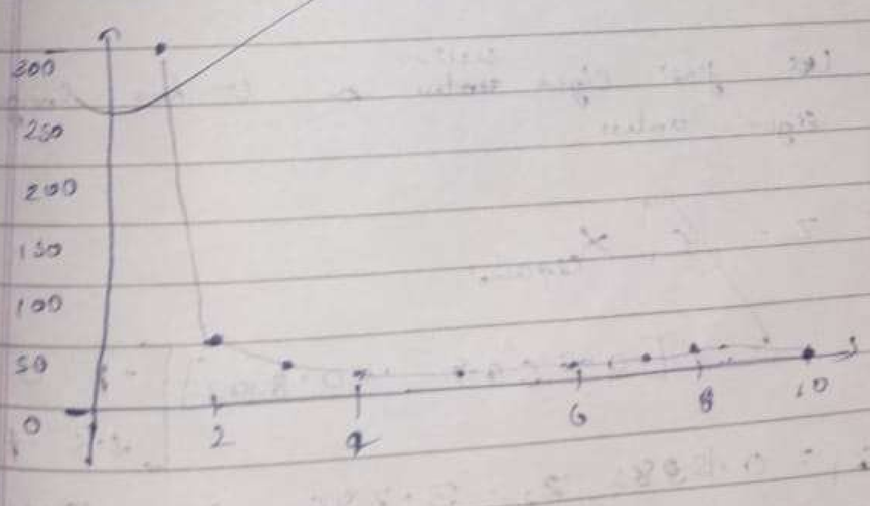
$\Rightarrow (4.125, 5.375)$

Round	Point (A, B)	distance (1.83, 2.33)	(4.125, 5.375) dist	Cluster
R1	(1.0, 1.0)	1.52	5.62	C1
R2	(1.5, 2.0)	0.47	4.51	C1
R3	(3.0, 4.0)	2.12	1.65	C2
R4	(5.0, 7.0)	3.71	1.85	C2
R5	(3.5, 5.0)	3.53	0.78	C2
R6	(4.5, 5.0)	3.42	0.53	C2
R7	(3.5, 4.5)	3.07	1.01	C2

Cluster 1 ⇒ (1, 1), (1.5, 2.0)

Cluster 2 → (3.0, 4.0), (5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)

Elbow method for optimal K



Code:

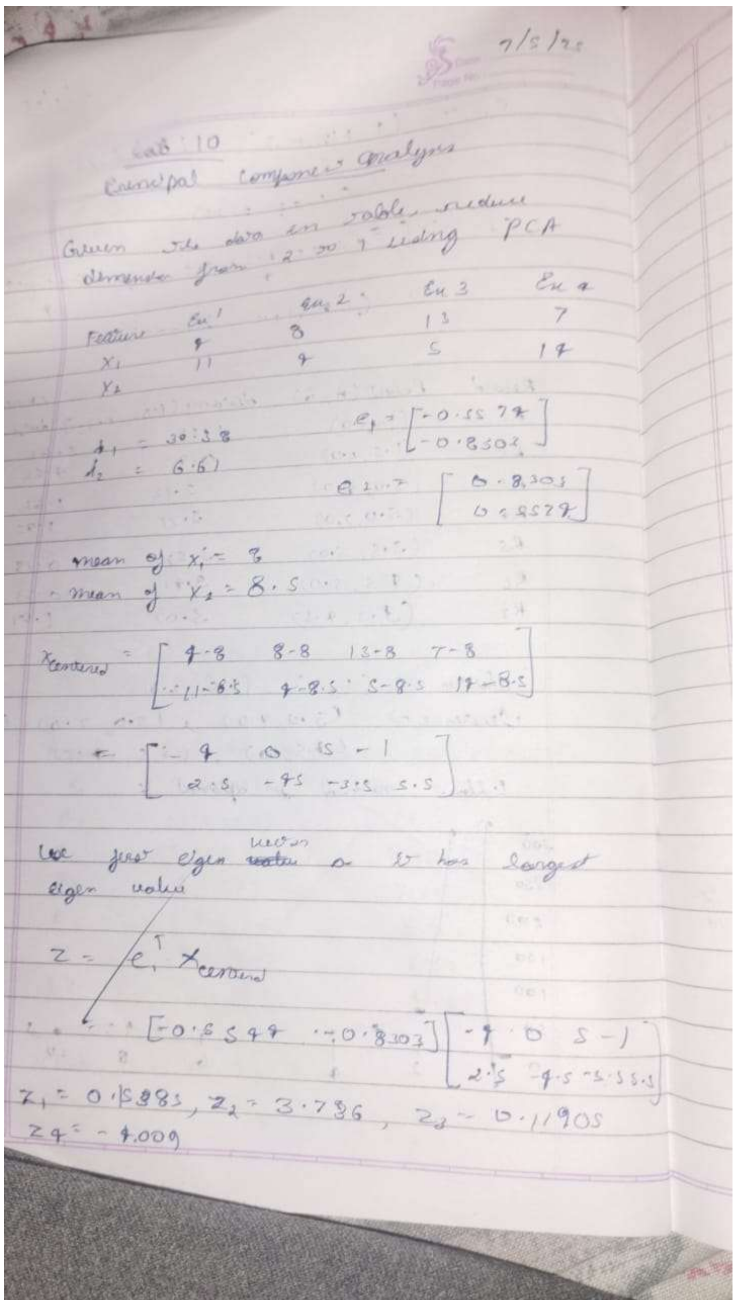
```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import numpy as np



# Load dataset

df = pd.read_csv("iris (4).csv")  # Replace with your actual filename if
needed



# Use only petal length and petal width

X = df[['petal_length', 'petal_width']]



# Feature scaling

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)



# Elbow method to find optimal K

inertia = []

k_range = range(1, 11)

for k in k_range:

    kmeans = KMeans(n_clusters=k, random_state=42)
```

```
    kmeans.fit(X_scaled)

    inertia.append(kmeans.inertia_)



# Automatically determine the elbow point (optional)

diff = np.diff(inertia)

diff_r = np.diff(diff)

optimal_k = np.argwhere(diff_r > -0.1)[0][0] + 2  # add 2 due to second derivative shift



# Plot elbow graph with optimal K marked

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

plt.plot(k_range, inertia, 'bo-')

plt.axvline(x=optimal_k, color='red', linestyle='--', label=f'Optimal k = {optimal_k}')

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

plt.ylabel('Inertia')

plt.title('Elbow Method For Optimal k')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()



# Output the optimal k
```

```
print(f"Optimal number of clusters (k): {optimal_k}")
```

Program 11

Screenshot:



7/5/25

Lab 10

Principal Component Analysis

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

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

$\lambda_1 = 30.38$

$\lambda_2 = 6.61$

$e_1 = \begin{bmatrix} -0.5574 \\ -0.8303 \end{bmatrix}$

$e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

mean of $x_1 = 8$

mean of $x_2 = 8.5$

$X_{centered} = \begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.5 & 4-8.5 & 5-8.5 & 14-8.5 \end{bmatrix}$

$= \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$

Use first eigen ~~vector~~ vector as it has largest eigen value

$Z = e_1^T X_{centered}$

$\begin{bmatrix} -0.5574 & -0.8303 \end{bmatrix} \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$

$Z_1 = 0.15385$, $Z_2 = 3.736$, $Z_3 = 0.11905$

$Z_4 = -4.009$

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score




df = pd.read_csv("heart (1).csv")




categorical_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina',
'ST_Slope']

df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)




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

y = df["HeartDisease"]
```

```python
scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)



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



models = {

    "SVM": SVC(),

    "Logistic Regression": LogisticRegression(max_iter=1000),

    "Random Forest": RandomForestClassifier()

}



accuracy_before_pca = {}

for name, model in models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy_before_pca[name] = accuracy_score(y_test, y_pred)



pca = PCA(n_components=0.95)

X_pca = pca.fit_transform(X_scaled)



X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y,
test_size=0.2, random_state=42)
```

```python
accuracy_after_pca = {}

for name, model in models.items():

    model.fit(X_train_pca, y_train_pca)

    y_pred_pca = model.predict(X_test_pca)

    accuracy_after_pca[name] = accuracy_score(y_test_pca, y_pred_pca)


print(" Accuracy BEFORE PCA:")

for name, acc in accuracy_before_pca.items():

    print(f"{name}: {acc:.4f}")


print("\nAccuracy AFTER PCA:")

for name, acc in accuracy_after_pca.items():

    print(f"{name}: {acc:.4f}")


print(f"\nOriginal features: {X.shape[1]}")

print(f"Features after PCA: {X_pca.shape[1]}")
```