VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Shashank C (1BM22CS254)

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 **Shashank C (1BM22CS254),** 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.

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

Index

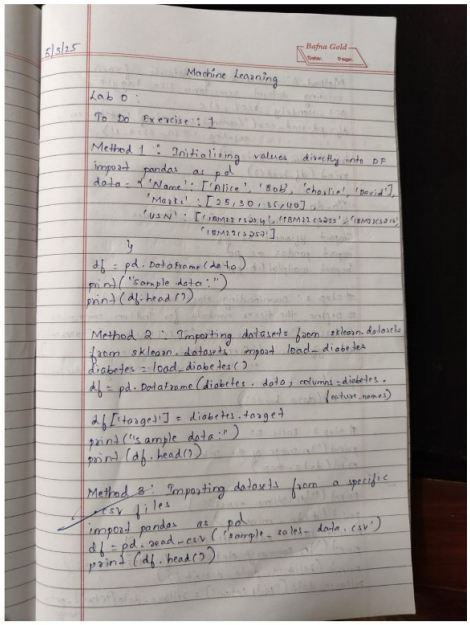
Link:

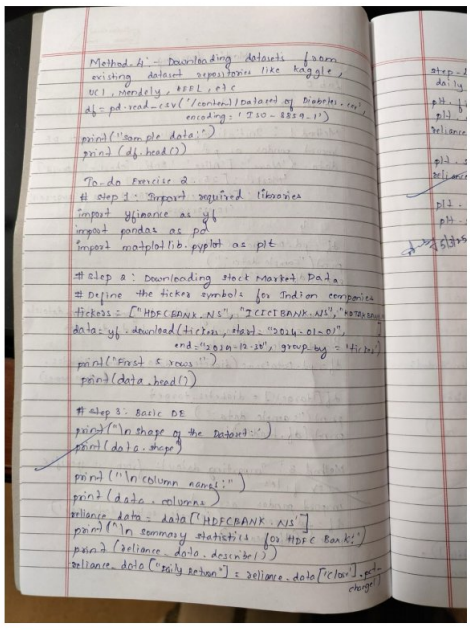
https://github.com/ShashankCS254/Machine_Learning-LAB.git

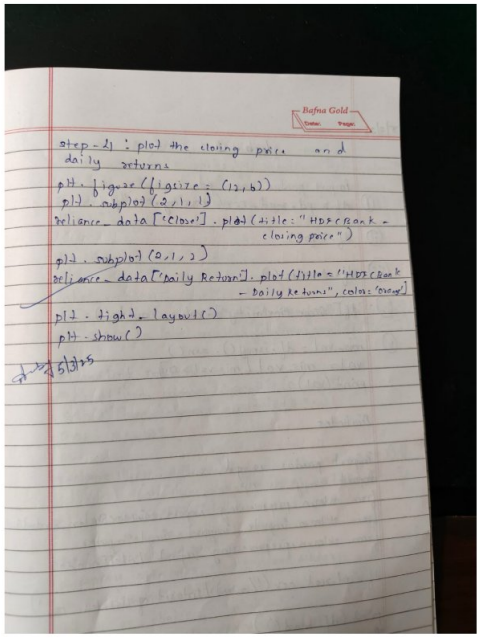
Program 1

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

Screenshot







Code:

```
#To do 2
# Step 1: Import required libraries
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# Step 2: Downloading Stock Market Data
# Define the ticker symbols for Indian companies
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

# Fetch historical data for the last 1 year
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')

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

# Step 3: Basic Data Exploration
# Check the shape of the dataset
print("\nShape of the dataset:")
print(data.shape)

# Check column names
#print("\nColumn names:")
#print(data.columns)

# Summary statistics for a specific stock HDFC
HDFC_data = data['HDFCBANK.NS']
print("\nSummary statistics for HDFC Industries:")
```

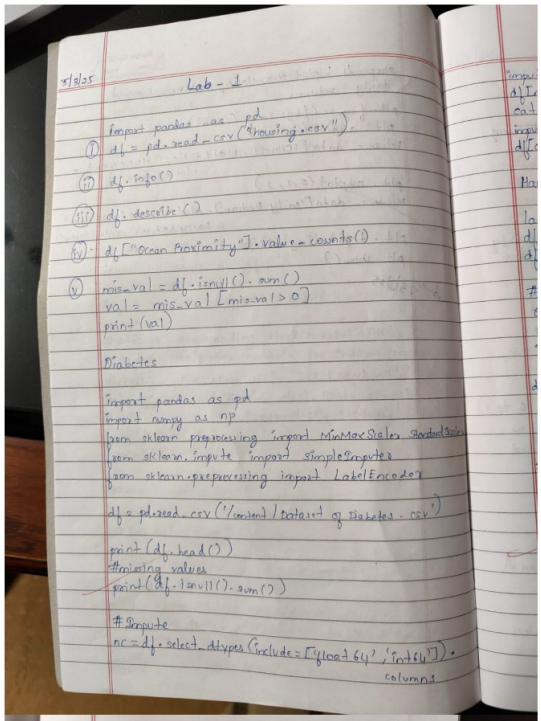
```
print(HDFC_data.describe())

# Calculate daily returns
HDFC_data['Daily Return'] = HDFC_data['Close'].pct_change()

# Plot the closing price and daily returns
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="HDFC Bank - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

# Summary statistics for a specific stock ICICI
ICICI_data = data['ICICIBANK.NS']
print("\nSummary statistics for ICICI Bank:")
print(ICICI_data.describe())

# Calculate daily returns
ICICI_data['Daily Return'] = ICICI_data['Close'].pct_change()

# Plot the closing price and daily returns
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()

# Summary statistics for a specific stock KOTAK
KOTAK_data = data['KOTAKBANK.NS']
print("\nSummary statistics for KOTAK Bank:")
print(ICICI_data.describe())

# Calculate daily returns
KOTAK_data['Daily Return'] = KOTAK_data['Close'].pct_change()

# Plot the closing price and daily returns
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()
```

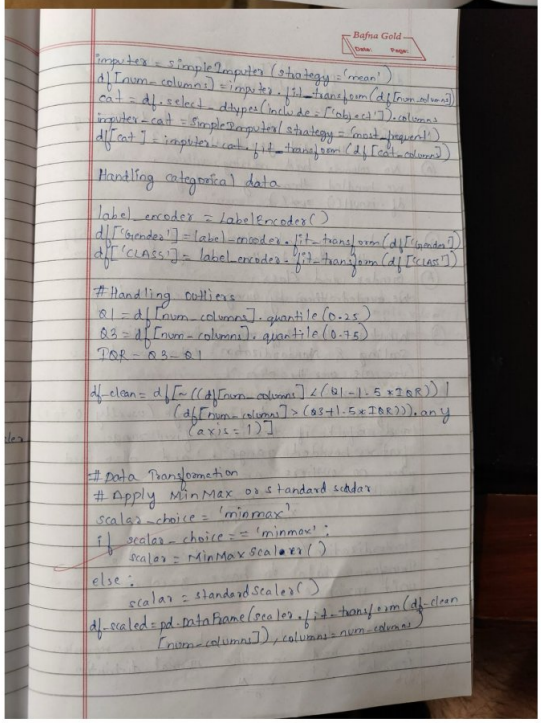
Program 2

Demonstrate various data pre-processing techniques for a given dataset

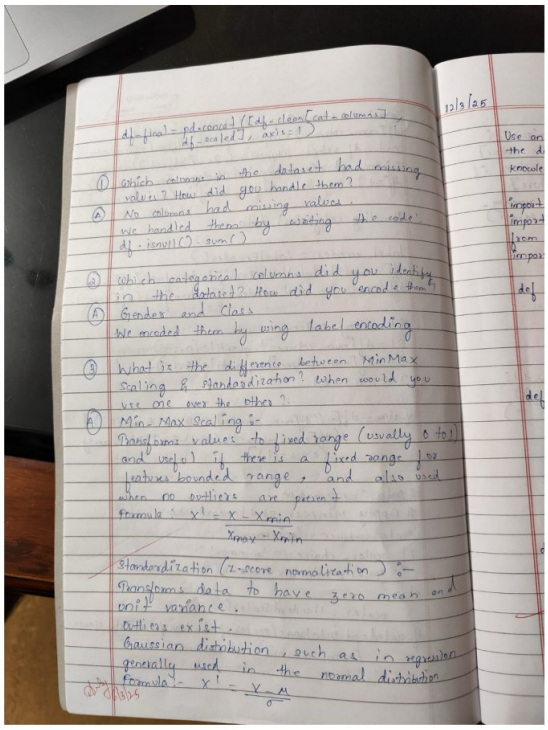
Screenshot



```
Lab - 1

import pandas as pd
(i) df = pd.read_csv("housing.csv")
(ii) df.info()
(iii) df.describe()
(iv) df["ocean_proximity"].value_counts()
(v) mis_val = df.isnull().sum()
val = mis_val[mis_val>0]
print(val)

Diabetes

import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('/content/Dataset of Diabetes .csv')

print(df.head())
#missing values
print(df.isnull().sum())

#Impute
nc = df.select_dtypes(include=['float64','int64']).
columns
```



```
imputer = SimpleImputer(strategy='mean')
df[num_columns] = imputer.fit_transform(df[num_columns])
cat = df.select_dtypes(include=['object']).columns
imputer_cat = SimpleImputer(strategy='most_frequent')
df[cat] = imputer_cat.fit_transform(df[cat_columns])

Handling categorical data

label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['CLASS'] = label_encoder.fit_transform(df['CLASS'])

#Handling outliers
Q1 = df[num_columns].quantile(0.25)
Q3 = df[num_columns].quantile(0.75)
IQR = Q3-Q1

df_clean = df[~((df[num_columns] < (Q1-1.5*IQR)) |
(df[num_columns] > (Q3+1.5*IQR))).any
(axis=1)]

#Data Transformation
#Apply MinMax or Standard scaler
scaler_choice = 'minmax'
if scaler_choice == 'minmax':
    scaler = MinMaxScaler()
else:
    scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_clean
[num_columns]), columns = num_columns)
```



Code:

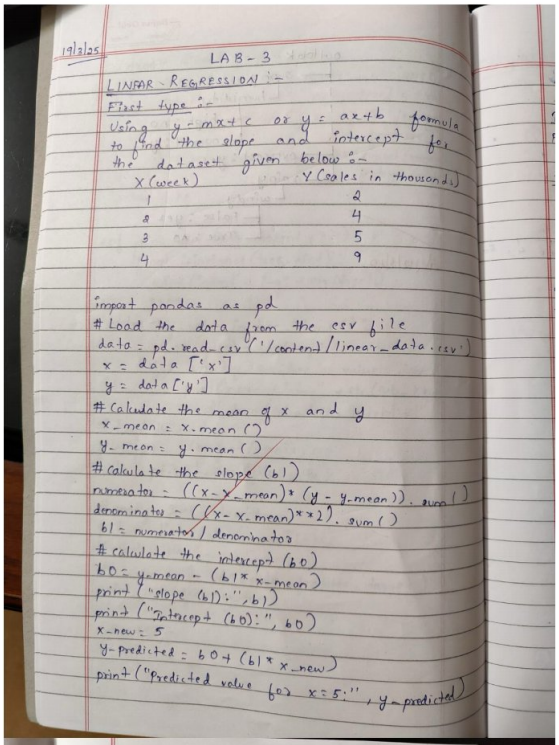
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('/content/Dataset of Diabetes .csv')

# Display first few rows
print("First few rows of the dataset:")
print(df.head())

# --- Step 1: Handling Missing Values ---
# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Impute missing values for numerical columns with mean
num_columns = df.select_dtypes(include=['float64', 'int64']).columns
imputer = SimpleImputer(strategy='mean')
df[num_columns] = imputer.fit_transform(df[num_columns])

# Impute missing values for categorical columns with the mode
cat_columns = df.select_dtypes(include=['object']).columns
```

```
imputer_cat = SimpleImputer(strategy='most_frequent')
df[cat_columns] = imputer_cat.fit_transform(df[cat_columns])

# --- Step 2: Handling Categorical Data ---
# Encode 'Gender' column using LabelEncoder
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])

# Encode 'CLASS' column using LabelEncoder (assuming 'N' and 'Y' are the categories)
df['CLASS'] = label_encoder.fit_transform(df['CLASS'])

# --- Step 3: Handling Outliers ---
# Detect and remove outliers using IQR method (Interquartile Range)
Q1 = df[num_columns].quantile(0.25)
Q3 = df[num_columns].quantile(0.75)
IQR = Q3 - Q1

# Filter out rows where any of the numerical columns have outliers
df_clean = df[~((df[num_columns] < (Q1 - 1.5 * IQR)) | (df[num_columns] > (Q3 + 1.5 *
IQR))).any(axis=1)]

# --- Step 4: Data Transformations ---
# Apply Min-Max Scaler or Standard Scaler
scaler_choice = 'minmax'  # Change to 'standard' for StandardScaler

if scaler_choice == 'minmax':
    scaler = MinMaxScaler()
else:
    scaler = StandardScaler()

# Scale the numerical columns
df_scaled = pd.DataFrame(scaler.fit_transform(df_clean[num_columns]), columns=num_columns)

# Join the scaled data back with the categorical columns
df_final = pd.concat([df_clean[cat_columns], df_scaled], axis=1)

# Display the cleaned and scaled data
print("\nCleaned and Scaled Data:")
print(df_final.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



LAB - 3

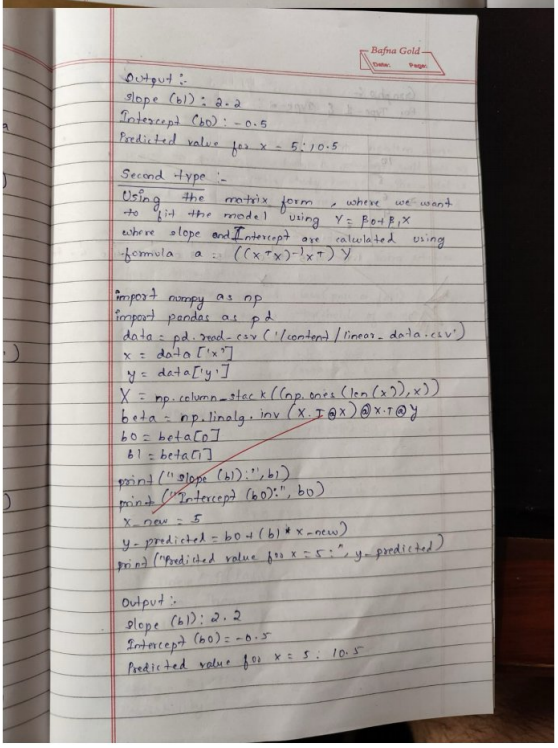
LINEAR REGRESSION :-

First type :-

Using $y = mx + c$ or $y = ax + b$ formula to find the slope and intercept for the dataset given below :-

X (week)	Y (Sales in thousands)
1	2
2	4
3	5
4	9

```
import pandas as pd
# Load the data from the csv file
data = pd.read_csv('/content/linear_data.csv')
x = data['x']
y = data['y']
# Calculate the mean of x and y
x_mean = x.mean()
y_mean = y.mean()
# Calculate the slope (b1)
numerator = ((x - x_mean) * (y - y_mean)).sum()
denominator = ((x - x_mean)**2).sum()
b1 = numerator / denominator
# Calculate the intercept (b0)
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)
```



Output :

Slope (b1) : 2.2

Intercept (b0) : -0.5

Predicted value for x = 5 : 10.5

Second type :-

Using the matrix form, where we want to fit the model using $Y = \beta_0 + \beta_1 X$ where slope and Intercept are calculated using formula $a = ((X^T X)^{-1} X^T) Y$

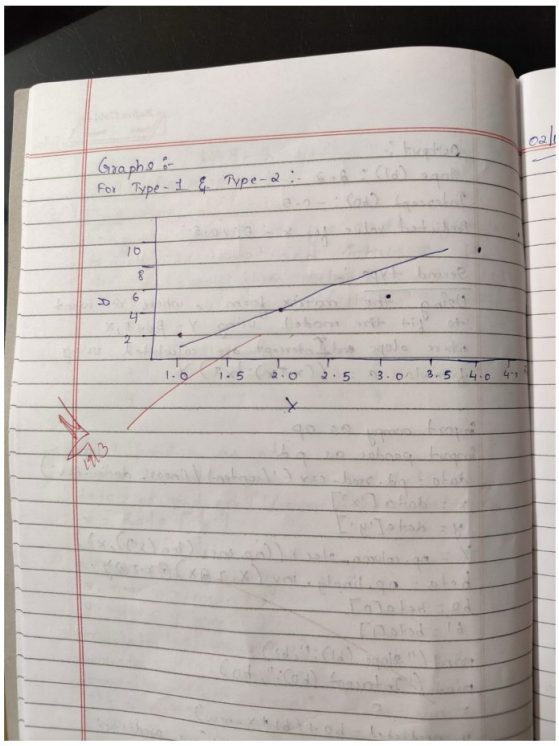
```
import numpy as np
import pandas as pd
data = pd.read_csv('/content/linear_data.csv')
x = data['x']
y = data['y']
X = np.column_stack((np.ones(len(x)), x))
beta = np.linalg.inv(X.T @ X) @ X.T @ y
b0 = beta[0]
b1 = beta[1]
print("Slope (b1):", b1)
print("Intercept (b0):", b0)
x_new = 5
y_predicted = b0 + (b1 * x_new)
print("Predicted value for x = 5:", y_predicted)
```

Output :

Slope (b1) : 2.2

Intercept (b0) : -0.5

Predicted value for x = 5 : 10.5



Code:

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

# Load the data from the CSV file
data = pd.read_csv('/content/linear_data.csv')
x = data['x']
y = data['y']

# Calculate the mean of x and y
x_mean = x.mean()
y_mean = y.mean()

# Calculate the slope (b1)
numerator = ((x - x_mean) * (y - y_mean)).sum()
denominator = ((x - x_mean)**2).sum()
b1 = numerator / denominator

# Calculate the intercept (b0)
b0 = y_mean - (b1 * x_mean)

# Print the slope and intercept
print("Slope (b1):", b1)
```

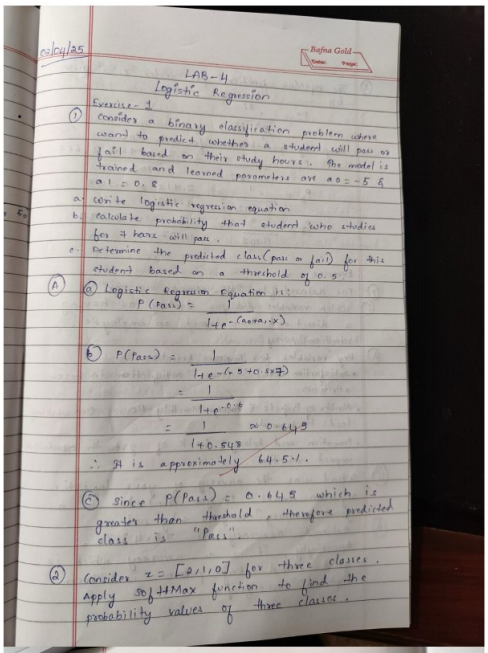
```
print("Intercept (b0):", b0)

# Predictive model for x = 5
x_new = 5
y_predicted = b0 + (b1 * x_new)

# Print the predicted value
print("Predicted value for x = 5:", y_predicted)

# Plot the data points and the fitted line
plt.scatter(x, y, color='blue', label='Data Points')  # Plot the actual data
plt.plot(x, b0 + b1 * x, color='red', label='Fitted Line')  # Plot the regression line
plt.scatter(x_new, y_predicted, color='green', label=f'Prediction for x={x_new}')  # Prediction point
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression Fit')
plt.legend()
plt.show()

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the data from the CSV file
data = pd.read_csv('/content/linear_data.csv')
x = data['x']
y = data['y']

# Create the design matrix X (add a column of 1s for the intercept)
X = np.column_stack((np.ones(len(x)), x))

# Calculate the coefficients using the matrix method
beta = np.linalg.inv(X.T @ X) @ X.T @ y

# Extract the intercept (b0) and slope (b1)
b0 = beta[0]
b1 = beta[1]

# Print the slope and intercept
print("Slope (b1):", b1)
print("Intercept (b0):", b0)

# Predictive model for x = 5
x_new = 5
```

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

# Print the predicted value
print("Predicted value for x = 5:", y_predicted)

# Plot the data points and the fitted line
plt.scatter(x, y, color='blue', label='Data Points')  # Plot the actual data
plt.plot(x, b0 + b1 * x, color='red', label='Fitted Line')  # Plot the regression line
plt.scatter(x_new, y_predicted, color='green', label=f'Prediction for x={x_new}')  # Prediction point
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression Fit (Matrix Method)')
plt.legend()
plt.show()
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot



LAB-4
Logistic Regression

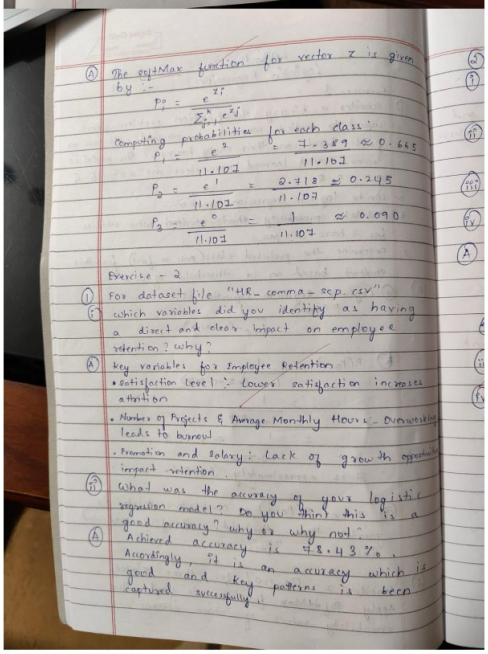
Exercise-1
① consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The model is trained and learned parameters are a0 = -5 & a1 = 0.8
a. write logistic regression equation
b. calculate probability that student who studies for 7 hours will pass.
c. Determine the predicted class (pass or fail) for this student based on a threshold of 0.5

(A) ① Logistic Regression Equation is:
$P(\text{Pass}) = \frac{1}{1+e^{-(a_0+a_1 \cdot x)}}$

(B) $P(\text{Pass}) = \frac{1}{1+e^{-(-5+0.8x7)}}$
$= \frac{1}{1+e^{-0.6}}$
$= \frac{1}{1+0.548} \approx 0.645$
∴ It is approximately 64.5%.

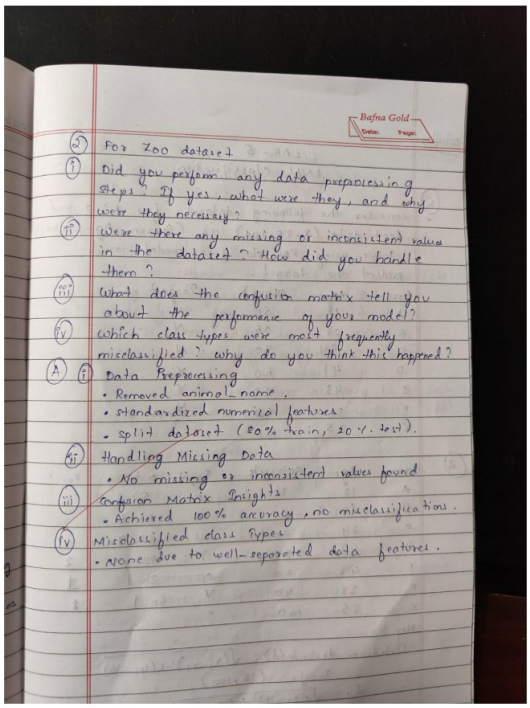
(C) ② Since P(Pass) = 0.645 which is greater than threshold, therefore predicted class is "Pass".

② Consider z = [0,1,0] for three classes. Apply Soft Max function to find the probability values of three classes.



(A) The softMax function for vector z is given by :-
$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$
Computing probabilities for each class:
$P_1 = \frac{e^2}{11.107} = \frac{7.389}{11.107} \approx 0.665$
$P_2 = \frac{e^1}{11.107} = \frac{2.718}{11.107} \approx 0.245$
$P_3 = \frac{e^0}{11.107} = \frac{1}{11.107} \approx 0.090$

Exercise - 2
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?
(A) key variables for employee Retention
• Satisfaction level :- Lower satisfaction increases attrition
• Number of Projects & Average Monthly Hours :- Overworking leads to burnout
• Promotion and Salary :- Lack of growth opportunities impact retention.
(ii) What was the accuracy of your logistic regression model? Do you think this is a good accuracy? why or why not?
(A) Achieved accuracy is 78.43%. Accordingly, it is an accuracy which is good and key patterns is been captured successfully.



Code:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
df = pd.read_csv('/content/HR_comma_sep (2).csv')

# Step 1: Exploratory Data Analysis (EDA)

# Check for missing values and basic statistics
print(df.isnull().sum())
print(df.describe())

# Check the data types of columns
print(df.dtypes)

# Inspect the first few rows of the data
print(df.head())
```

```
# Step 2: Plotting bar charts to visualize the relationship between salary, department, and retention

# Plot bar chart for salary vs retention
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='salary', hue='left')
plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary')
plt.ylabel('Count')
plt.show()

# Plot bar chart for department vs retention
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='Department', hue='left')
plt.title('Impact of Department on Employee Retention')
plt.xlabel('Department')
plt.ylabel('Count')
plt.show()

# Step 3: Preprocessing data for Logistic Regression
# For simplicity, we will use 'satisfaction_level', 'last_evaluation', 'number_project',
# 'average_montly_hours', 'time_spend_company', 'Work_accident', 'promotion_last_5years',
'department', 'salary'

# Convert categorical variables to numerical using one-hot encoding
df_encoded = pd.get_dummies(df, columns=['Department', 'salary'], drop_first=True)

# Step 4: Define Features and Target
X = df_encoded.drop(columns=['left'])  # Features
y = df_encoded['left']  # Target variable (whether employee left the company)

# 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=0.2, random_state=42)

# Step 6: Scale the features (Logistic Regression works better with scaled data)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 7: Build Logistic Regression Model
model = LogisticRegression(random_state=42)
model.fit(X_train_scaled, y_train)

# Step 8: Make Predictions
y_pred = model.predict(X_test_scaled)
```

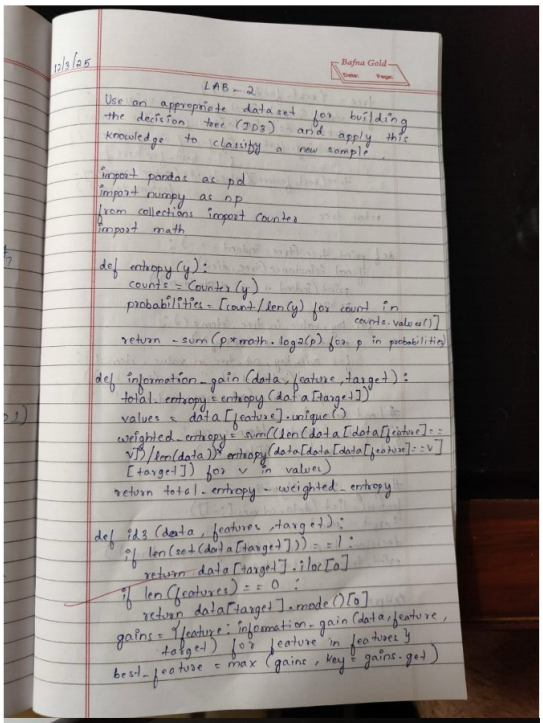
```
# Step 9: Evaluate the Model's Performance

# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Logistic Regression model: {accuracy:.4f}')

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Classification Report
print('Classification Report:')
print(classification_report(y_test, y_pred))


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

# Load datasets
zoo_data_path = "/content/zoo-data (1).csv"
class_type_path = "/content/zoo-class-type (1).csv"

zoo_data = pd.read_csv(zoo_data_path)
class_type = pd.read_csv(class_type_path)

# Drop the 'animal_name' column
zoo_data_cleaned = zoo_data.drop(columns=['animal_name'])

# Split into features (X) and target (y)
X = zoo_data_cleaned.drop(columns=['class_type'])
y = zoo_data_cleaned['class_type']

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

# Standardize features
```

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

# Train Logistic Regression Model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=200)
model.fit(X_train_scaled, y_train)

# Predictions
y_pred = model.predict(X_test_scaled)

# Model Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Model Accuracy: {accuracy:.2f}')

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y),
yticklabels=np.unique(y))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix (Accuracy: {accuracy:.2f})')
plt.show()
```

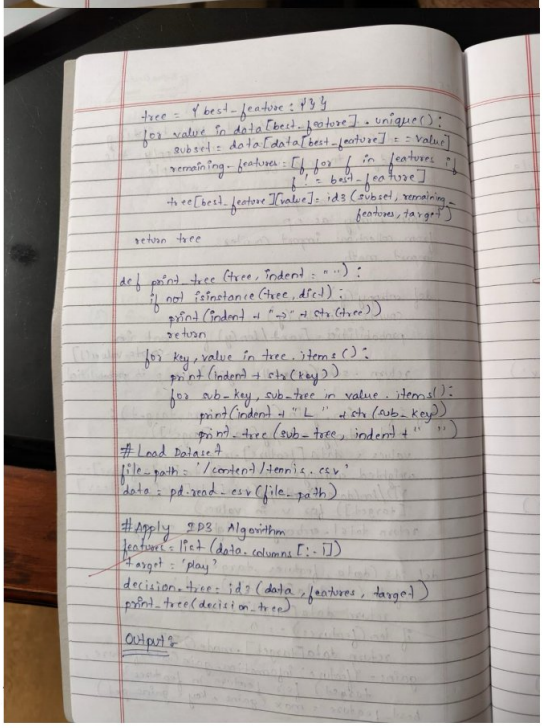
Program 5

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

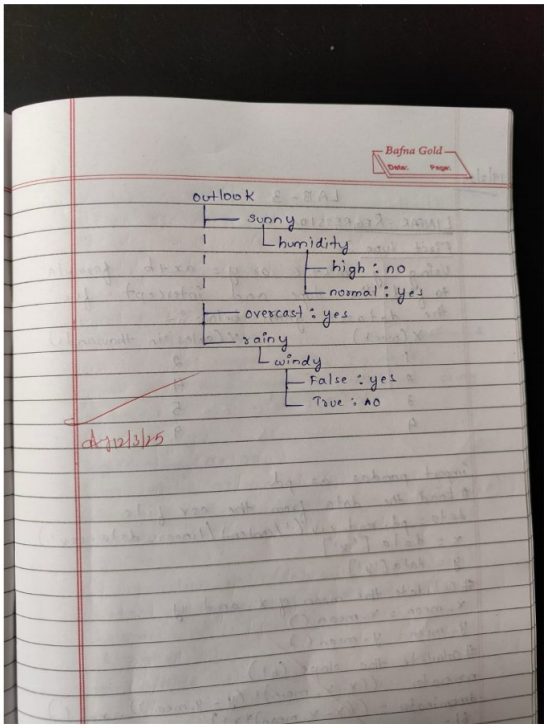
Screenshot



```
LAB - 2
Use an appropriate dataset for building the decision tree (ID3) and apply this knowledge to classify a new sample

import pandas as pd
import numpy as np
from collections import Counter
import math

def entropy(y):
    counts = Counter(y)
    probabilities = [count/len(y) for count in counts.values()]
    return -sum(p*math.log2(p) for p in probabilities)

def information_gain(data, feature, target):
    total_entropy = entropy(data[target])
    values = data[feature].unique()
    weighted_entropy = sum((len(data[data[feature]==v])/len(data))*entropy(data[data[feature]==v][target]) for v in values)
    return total_entropy - weighted_entropy

def id3(data, features, target):
    if len(set(data[target]))==1:
        return data[target].iloc[0]
    if len(features)==0:
        return data[target].mode()[0]
    gains = {feature: information_gain(data, feature, target) for feature in features}
    best_feature = max(gains, key=gains.get)
```



```
    tree = {best_feature:{}}
    for value in data[best_feature].unique():
        subset = data[data[best_feature]==value]
        remaining_features = [f for f in features if f!=best_feature]
        tree[best_feature][value] = id3(subset, remaining_features, target)
    return tree

def print_tree(tree, indent=""):
    if not isinstance(tree, dict):
        print(indent + "->" + str(tree))
        return
    for key, value in tree.items():
        print(indent + str(key))
        for sub_key, sub_tree in value.items():
            print(indent + " L " + str(sub_key))
            print_tree(sub_tree, indent + "  ")
# Load Dataset
file_path = '/content/tennis.csv'
data = pd.read_csv(file_path)

#Apply ID3 Algorithm
features = list(data.columns[:-1])
target = 'play'
decision_tree = id3(data, features, target)
print_tree(decision_tree)

Output:
```



Code:

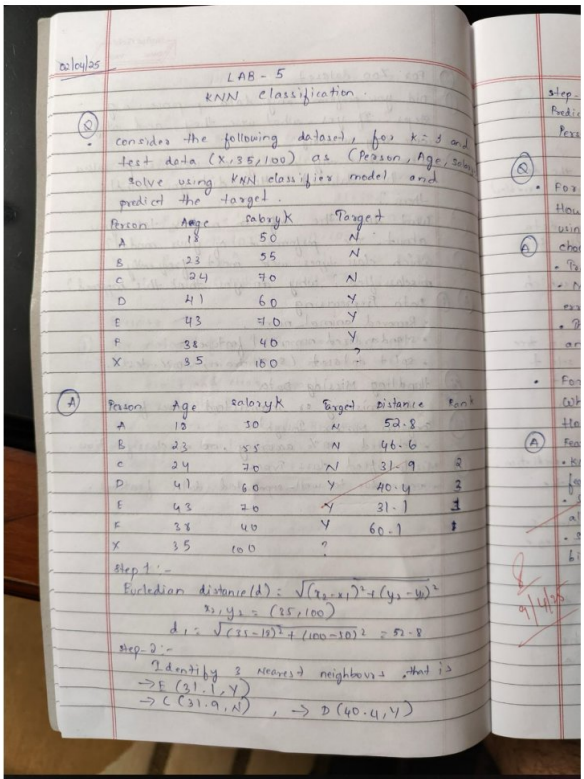
```
import pandas as pd
import numpy as np
from collections import Counter
import math

# Function to calculate entropy
def entropy(y):
    counts = Counter(y)
    probabilities = [count / len(y) for count in counts.values()]
    return -sum(p * math.log2(p) for p in probabilities)

# Function to calculate information gain
def information_gain(data, feature, target):
    total_entropy = entropy(data[target])
    values = data[feature].unique()
    weighted_entropy = sum((len(data[data[feature] == v]) / len(data)) * entropy(data[data[feature] == v]
[target]) for v in values)
    return total_entropy - weighted_entropy

# Recursive ID3 algorithm to build the decision tree
def id3(data, features, target):
    # If all target values are the same, return the label
    if len(set(data[target])) == 1:
```

```
        return data[target].iloc[0]

    # If no features left, return the most common label
    if len(features) == 0:
        return data[target].mode()[0]

    # Choose the best feature based on information gain
    gains = {feature: information_gain(data, feature, target) for feature in features}
    best_feature = max(gains, key=gains.get)

    # Create tree node
    tree = {best_feature: {}}

    # Split dataset and recurse for each value of the best feature
    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value]
        remaining_features = [f for f in features if f != best_feature]
        tree[best_feature][value] = id3(subset, remaining_features, target)

    return tree

# Function to print decision tree in a readable format
def print_tree(tree, indent=""):
    if not isinstance(tree, dict):
        print(indent + "→ " + str(tree))
        return

    for key, value in tree.items():
        print(indent + str(key))
        for sub_key, sub_tree in value.items():
            print(indent + "  └─ " + str(sub_key))
            print_tree(sub_tree, indent + "    ")

# Load dataset
file_path = '/content/tennis.csv'
data = pd.read_csv(file_path)

# Apply ID3 algorithm
features = list(data.columns[:-1])  # All columns except the target
target = 'play'
decision_tree = id3(data, features, target)

# Print the decision tree
print_tree(decision_tree)
```

Program 6

Build KNN Classification model for a given dataset

Screenshot



LAB - 5
KNN classification

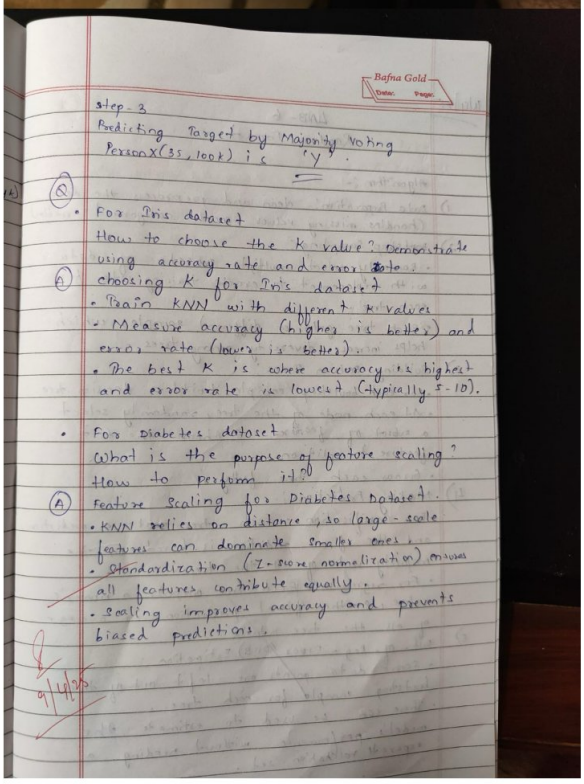
Consider the following dataset, for k=3 and test data (X, 35, 100) as (Person, Age, Salary). Solve using KNN classifier model and predict the target.

Person	Age	salaryK	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	?

Person	Age	salaryK	Target	distance	rank
A	18	50	N	52.8	
B	23	55	N	46.6	
C	24	70	N	31.9	2
D	41	60	Y	40.4	3
E	43	70	Y	31.1	1
F	38	40	Y	60.1	
X	35	100	?		

Step 1:-
Euclidean distance(d): $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
$x_1, y_1 = (35, 100)$
$d_1 = \sqrt{(35-18)^2+(100-50)^2} = 52.8$

Step 2:-
Identify 3 Nearest neighbours, that is
→ E (31.1, Y)
→ C (31.9, N), → D (40.4, Y)



Bafna Gold

Step 3
Predicting Target by Majority voting
Person X(35, 100) is 'Y'.

- For Iris dataset
How to choose the K value? Demonstrate using accuracy rate and error rate.

choosing K for Iris dataset
- Train KNN with different K values
- Measure accuracy (higher is better) and error rate (lower is better).
- The best K is where accuracy is highest and error rate is lowest (typically 5-10).

- For Diabetes dataset
What is the purpose of feature scaling? How to perform it?

Feature Scaling for Diabetes dataset.
- KNN relies on distance, so large-scale features can dominate smaller ones.
- Standardization (Z-score normalization) ensures all features contribute equally.
- Scaling improves accuracy and prevents biased predictions.

9/4/25

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
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 datasets
iris_data = pd.read_csv("/content/iris (3).csv")
diabetes_data = pd.read_csv("/content/diabetes (1).csv")

# 1. KNN Classifier for IRIS Dataset
# Splitting features and target
X_iris = iris_data.drop(columns=['species'])
y_iris = iris_data['species']

# Train-test split (80% train, 20% test)
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)

# Train KNN model (choosing K=5 as an example)
knn_iris = KNeighborsClassifier(n_neighbors=5)
knn_iris.fit(X_train_iris, y_train_iris)

# Predictions
y_pred_iris = knn_iris.predict(X_test_iris)

# Accuracy and Confusion Matrix
accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
conf_matrix_iris = confusion_matrix(y_test_iris, y_pred_iris)

print("Iris Dataset Classification Report:")
print(classification_report(y_test_iris, y_pred_iris))
print(f"Iris Dataset Accuracy: {accuracy_iris:.2f}")

# Plot Confusion Matrix for IRIS
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_iris, annot=True, cmap='Blues', fmt='d', xticklabels=np.unique(y_iris),
yticklabels=np.unique(y_iris))
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.title('Confusion Matrix - IRIS Dataset')
plt.show()

# 2. KNN Classifier for Diabetes Dataset
# Splitting features and target
X_diabetes = diabetes_data.drop(columns=['Outcome'])
y_diabetes = diabetes_data['Outcome']

# Train-test split (80% train, 20% test)
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 (Standardization)
scaler = StandardScaler()
X_train_diabetes_scaled = scaler.fit_transform(X_train_diabetes)
X_test_diabetes_scaled = scaler.transform(X_test_diabetes)

# Train KNN model (choosing K=5 as an example)
knn_diabetes = KNeighborsClassifier(n_neighbors=5)
knn_diabetes.fit(X_train_diabetes_scaled, y_train_diabetes)

# Predictions
y_pred_diabetes = knn_diabetes.predict(X_test_diabetes_scaled)

# Accuracy and Confusion Matrix
accuracy_diabetes = accuracy_score(y_test_diabetes, y_pred_diabetes)
conf_matrix_diabetes = confusion_matrix(y_test_diabetes, y_pred_diabetes)

print("Diabetes Dataset Classification Report:")
print(classification_report(y_test_diabetes, y_pred_diabetes))
print(f"Diabetes Dataset Accuracy: {accuracy_diabetes:.2f}")

# Plot Confusion Matrix for Diabetes
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_diabetes, annot=True, cmap='Reds', fmt='d', xticklabels=[0,1],
yticklabels=[0,1])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Diabetes Dataset')
plt.show()
```

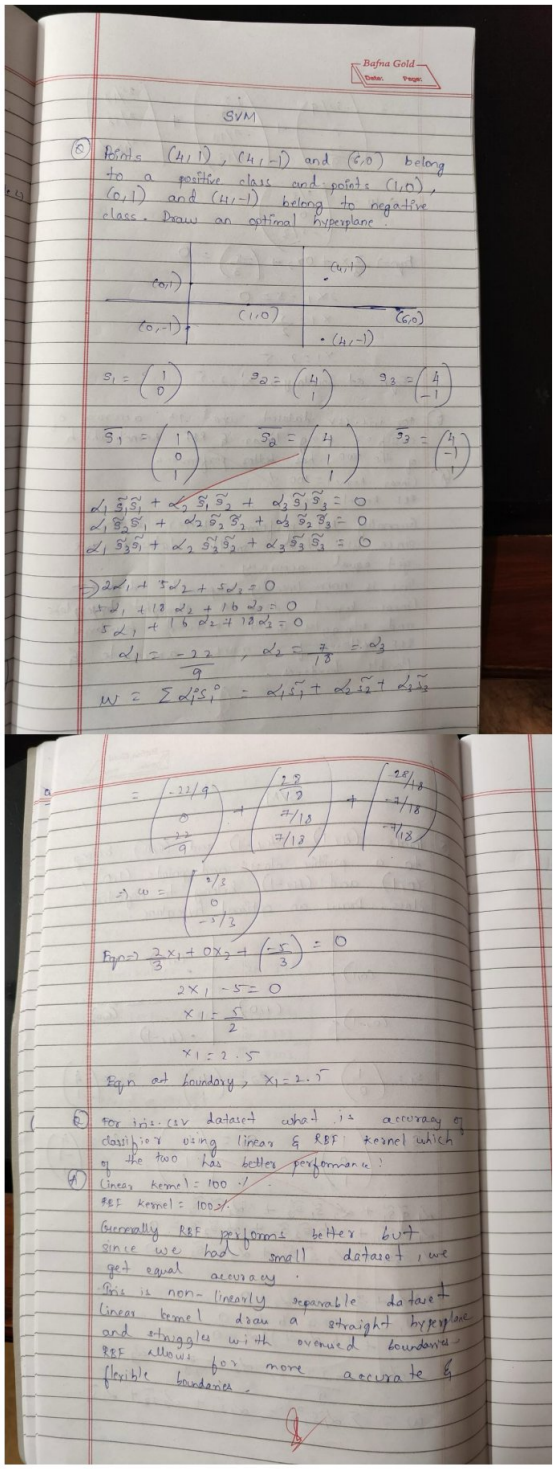
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,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

# Load the dataset
heart_data = pd.read_csv("/content/heart (1).csv")

# Splitting features and target
X_heart = heart_data.drop(columns=['target'])
y_heart = heart_data['target']

# Train-test split (80% train, 20% test)
X_train_heart, X_test_heart, y_train_heart, y_test_heart = train_test_split(X_heart, y_heart,
test_size=0.2, random_state=42)

# Feature Scaling (Standardization)
scaler = StandardScaler()
X_train_heart_scaled = scaler.fit_transform(X_train_heart)
X_test_heart_scaled = scaler.transform(X_test_heart)

# Finding the best K value
k_values = range(1, 21)
accuracy_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_heart_scaled, y_train_heart)
    y_pred = knn.predict(X_test_heart_scaled)
    accuracy_scores.append(accuracy_score(y_test_heart, y_pred))

# Plot accuracy vs. K value
plt.figure(figsize=(8, 5))
plt.plot(k_values, accuracy_scores, marker='o', linestyle='dashed', color='b')
plt.xlabel('K Value')
plt.ylabel('Accuracy Score')
plt.title('Finding the Optimal K for KNN')
plt.show()

# Best K value
```

```
best_k = k_values[np.argmax(accuracy_scores)]
print(f"Best K value: {best_k}")

# Train final model with the best K
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train_heart_scaled, y_train_heart)
y_pred_heart = knn_best.predict(X_test_heart_scaled)

# Accuracy Score
accuracy_heart = accuracy_score(y_test_heart, y_pred_heart)
print(f"Heart Disease Dataset Accuracy: {accuracy_heart:.2f}")

# Confusion Matrix
conf_matrix_heart = confusion_matrix(y_test_heart, y_pred_heart)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_heart, annot=True, cmap='Blues', fmt='d', xticklabels=[0,1], yticklabels=[0,1])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Heart Disease Dataset')
plt.show()

# Classification Report
print("Classification Report:")
print(classification_report(y_test_heart, y_pred_heart))
```

Program 7

Build Support vector machine model for a given dataset

Screenshot



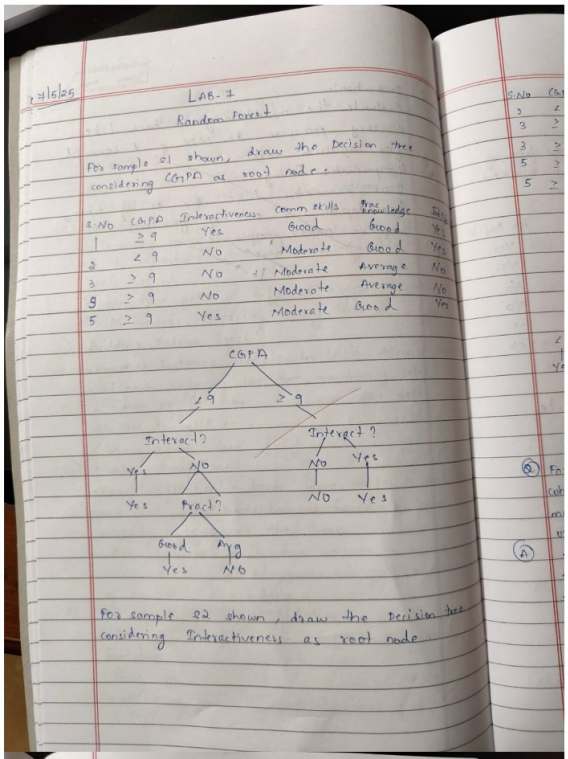
Code:

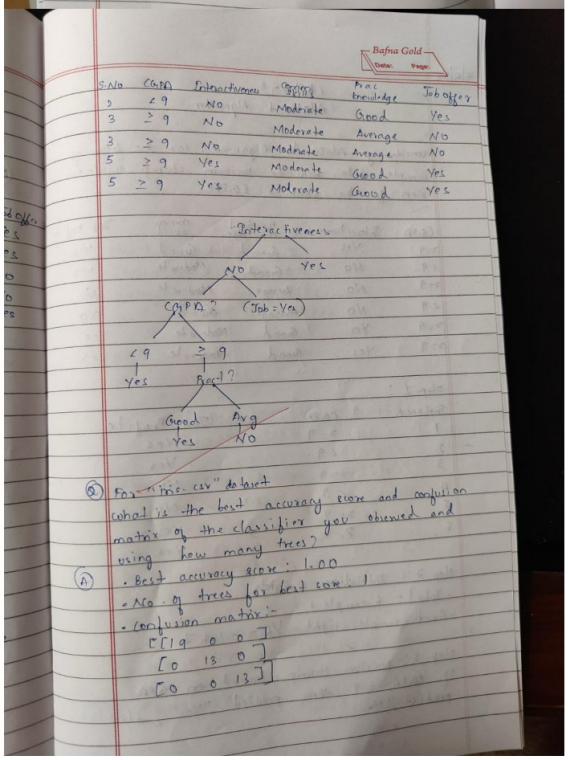
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix

# Load dataset
iris = pd.read_csv("iris (1).csv")

# Features and Labels
X = iris.drop('species', axis=1)
y = LabelEncoder().fit_transform(iris['species'])

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

# RBF Kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
pred_rbf = svm_rbf.predict(X_test)
print("RBF Kernel Accuracy:", accuracy_score(y_test, pred_rbf))
print("RBF Confusion Matrix:\n", confusion_matrix(y_test,
pred_rbf))

# Linear Kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
pred_linear = svm_linear.predict(X_test)
print("Linear Kernel Accuracy:", accuracy_score(y_test,
pred_linear))
print("Linear Confusion Matrix:\n", confusion_matrix(y_test,
pred_linear))
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot





Code:

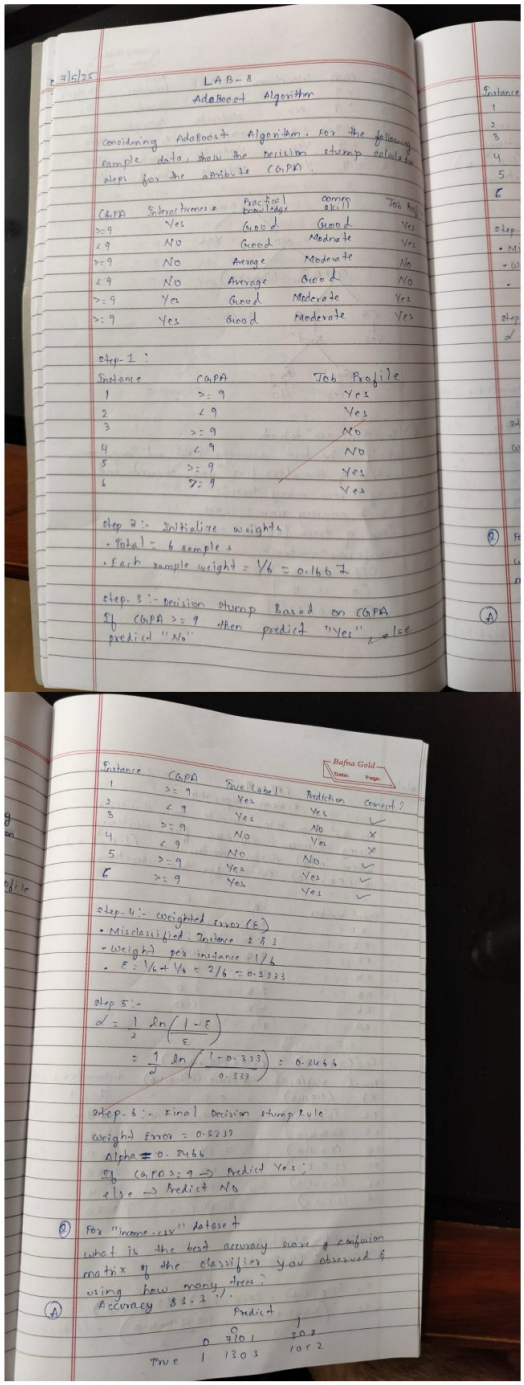
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("/content/iris (4).csv")  # replace with your path if needed

# Prepare features and labels
X = df.drop(columns='species')
y = df['species']

# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train with default n_estimators = 10
rf_default = RandomForestClassifier(random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default RF Accuracy (10 trees): {default_score:.2f}")

# Fine-tuning number of trees
n_estimators_range = range(1, 101)
scores = []

for n in n_estimators_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))

# Best score and corresponding number of trees
best_score = max(scores)
best_n = n_estimators_range[scores.index(best_score)]
print(f"Best Accuracy: {best_score:.2f} with {best_n} trees")

# Plotting accuracy vs number of trees
plt.figure(figsize=(10, 6))
```

```
plt.plot(n_estimators_range, scores, marker='o')
plt.axhline(y=default_score, color='r', linestyle='--', label=f'Default (10 trees): {default_score:.2f}')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy Score')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot



Code:

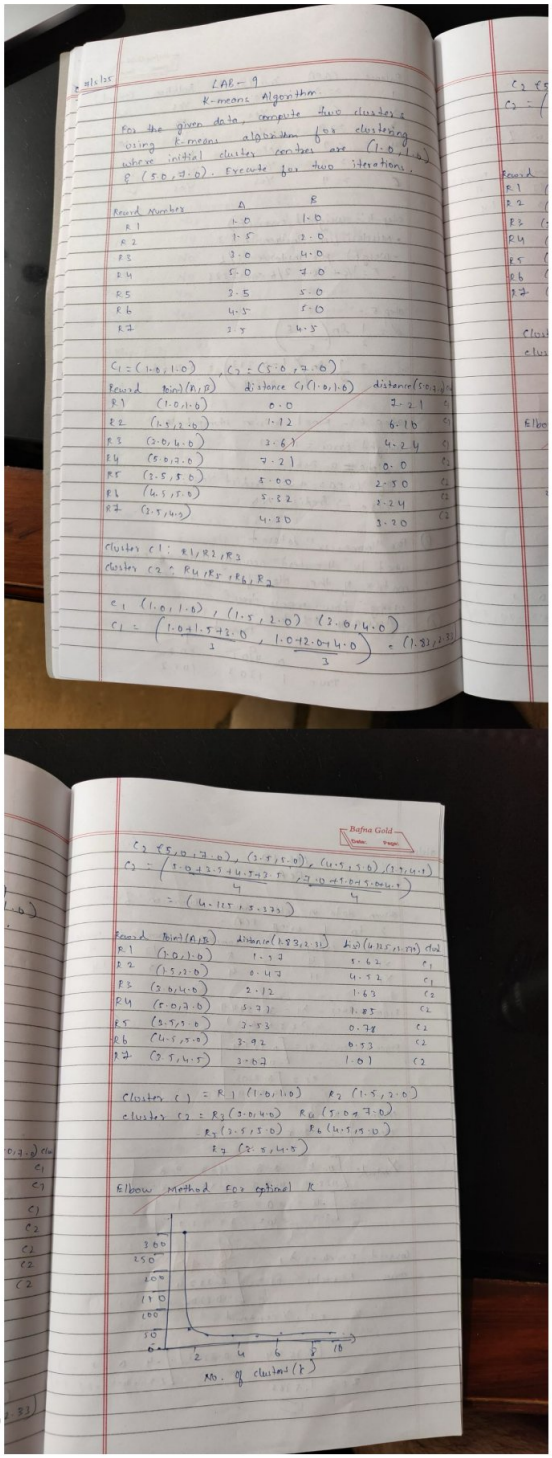
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("/content/income.csv")  # Make sure the CSV is in your working directory

# Define features and target
X = df.drop(columns='income_level')
y = df['income_level']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train AdaBoost with default n_estimators = 10
ada_default = AdaBoostClassifier(n_estimators=10, random_state=42)
ada_default.fit(X_train, y_train)
y_pred_default = ada_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default Accuracy (10 estimators): {default_score:.2f}")

# Fine-tune number of estimators
scores = []
n_estimators_range = range(1, 101)

for n in n_estimators_range:
    ada = AdaBoostClassifier(n_estimators=n, random_state=42)
    ada.fit(X_train, y_train)
    y_pred = ada.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))

# Best result
best_score = max(scores)
best_n = n_estimators_range[scores.index(best_score)]
print(f"Best Accuracy: {best_score:.2f} with {best_n} estimators")

# Confusion matrix
ada_best = AdaBoostClassifier(n_estimators=best_n, random_state=42)
ada_best.fit(X_train, y_train)
y_pred_best = ada_best.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred_best)
```

```
print("Confusion Matrix:\n", conf_matrix)

# Plotting accuracy vs number of estimators
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, scores, marker='o')
plt.axhline(y=default_score, color='r', linestyle='--', label=f'Default (10): {default_score:.2f}')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('AdaBoost Accuracy vs Number of Estimators')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Program 10

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

Screenshot



Code:

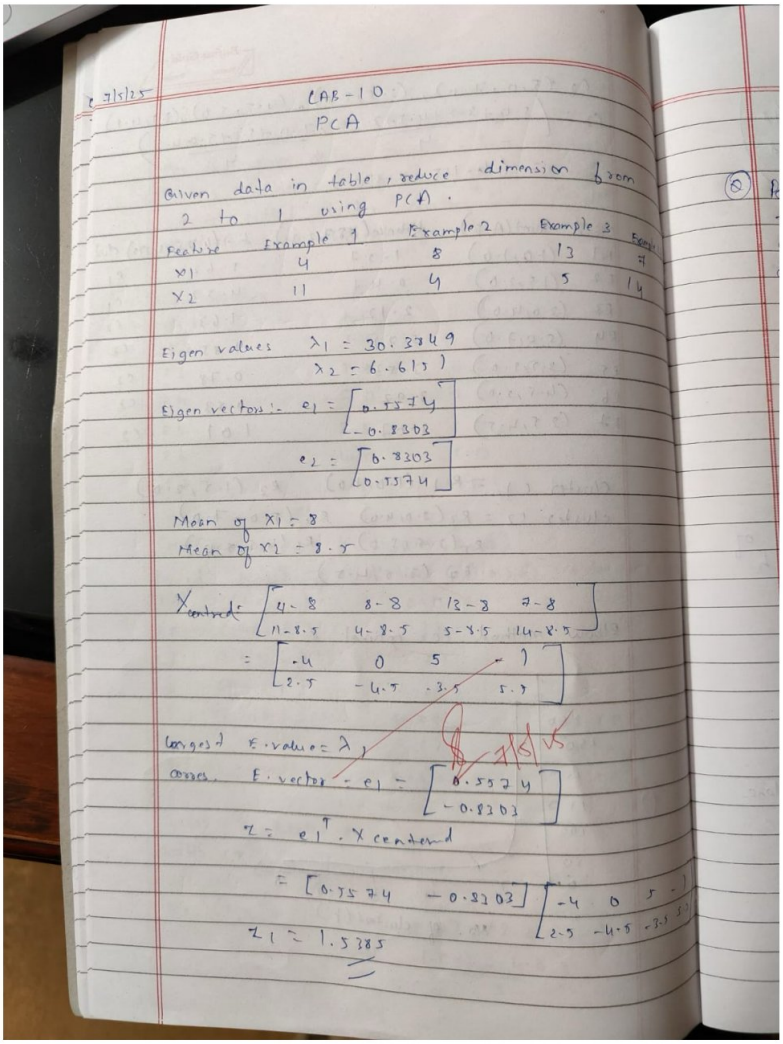
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

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

# Step 2: Select only petal length and width features
X = df[['petal_length', 'petal_width']]

# Step 3: Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: 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_)

# Step 5: Plot Elbow Graph
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, 'bo-')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia (Within-Cluster Sum of Squares)')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.show()

# Step 6: Build final model with optimal k (visually identify from elbow plot)
# You can replace 3 with the optimal k you see from the elbow plot
optimal_k = 3
kmeans_final = KMeans(n_clusters=optimal_k, random_state=42)
labels = kmeans_final.fit_predict(X_scaled)

# Step 7: Add cluster labels to original data
df['Cluster'] = labels

# Step 8: Plot the clustered data
```

```
plt.figure(figsize=(8, 5))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', s=50)
plt.scatter(kmeans_final.cluster_centers_[:, 0], kmeans_final.cluster_centers_[:, 1],
            s=200, c='red', marker='X', label='Centroids')
plt.xlabel('Petal Length (scaled)')
plt.ylabel('Petal Width (scaled)')
plt.title(f'K-Means Clustering with k={optimal_k}')
plt.legend()
plt.grid(True)
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot



Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
```

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv("/content/heart.csv")

# Identify categorical columns
cat_cols = df.select_dtypes(include=['object']).columns.tolist()

# Label Encoding for binary categories, One-Hot Encoding for others
for col in cat_cols:
    if df[col].nunique() == 2:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
    else:
        df = pd.get_dummies(df, columns=[col], drop_first=True)

# Features and Target
X = df.drop(columns='target') if 'target' in df.columns else df.iloc[:, :-1]
y = df['target'] if 'target' in df.columns else df.iloc[:, -1]

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

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train different models
models = {
    "SVM": SVC(),
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier()
}

print("=== Accuracy Without PCA ===")
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
```

```
    y_pred = model.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name}: {acc:.4f}")

# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% variance
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("\n=== Accuracy With PCA (95% variance) ===")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name} with PCA: {acc:.4f}")
```