VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

SHREE SANKET (1BM22CS261)

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 **Shree sanket (1BM22CS261),** 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. Saritha A N** Assistant Professor Department of CSE, BMSCE	 **Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

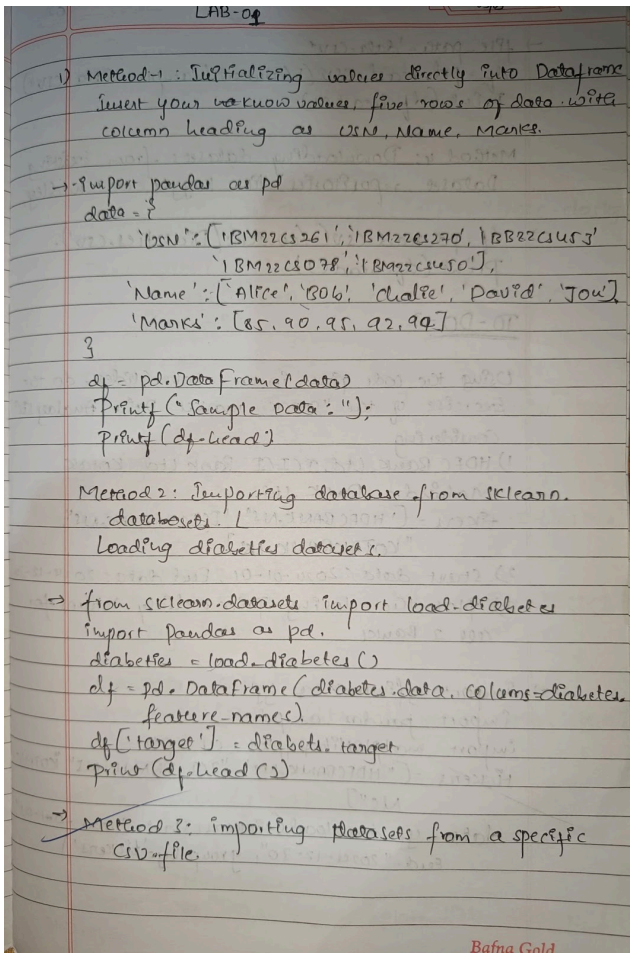
Github Link:

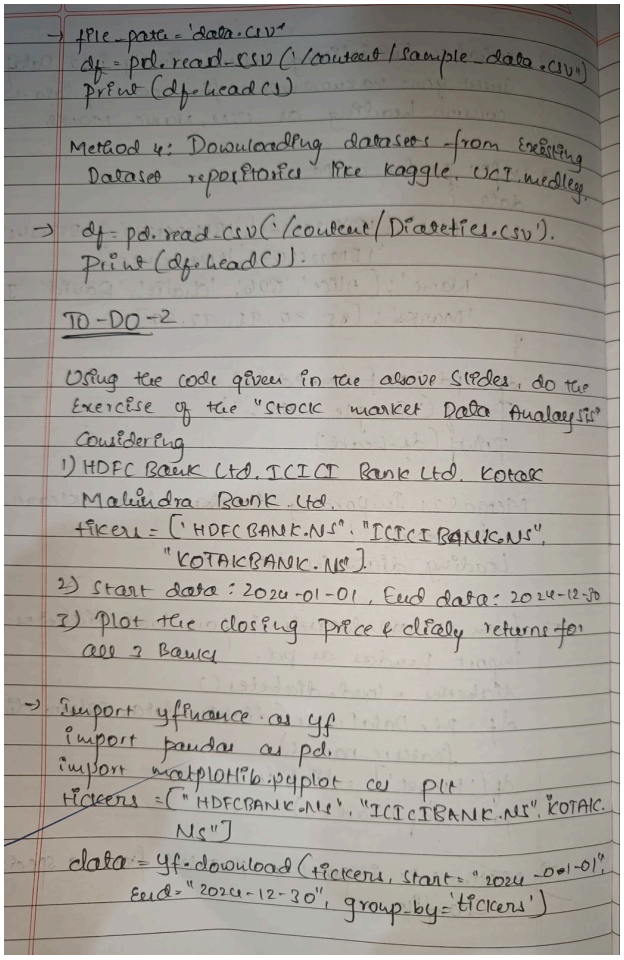
https://github.com/shreesanket/ML-LAB

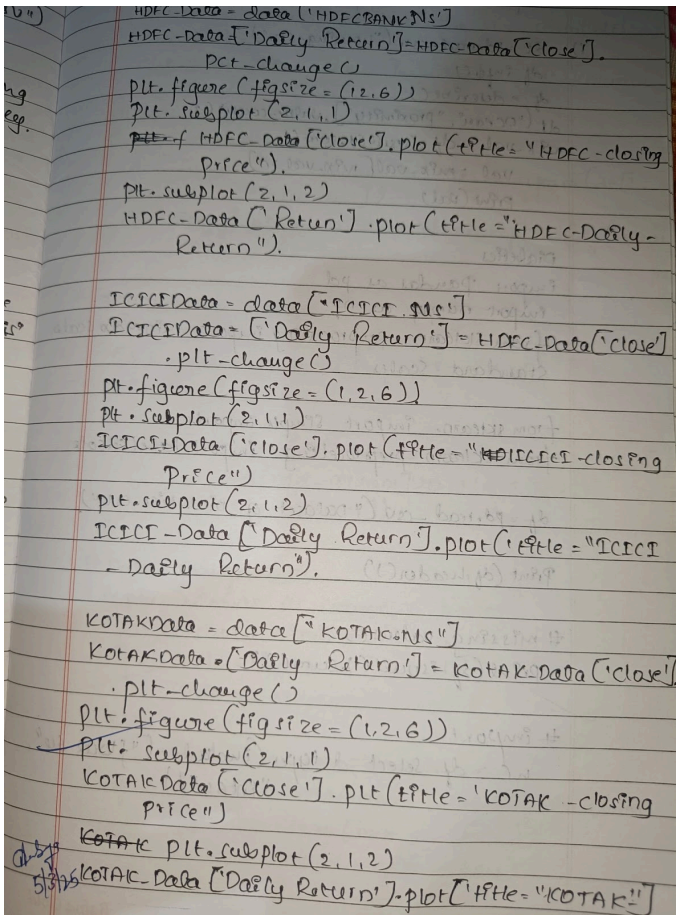
Program 1

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

Screenshot:







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="2023-10-01", end="2024-10-01", group_by='ticker')


HDFC_Data = data['HDFCBANK.NS']

HDFC_Data['Daily Return'] = HDFC_Data['Close'].pct_change()
```

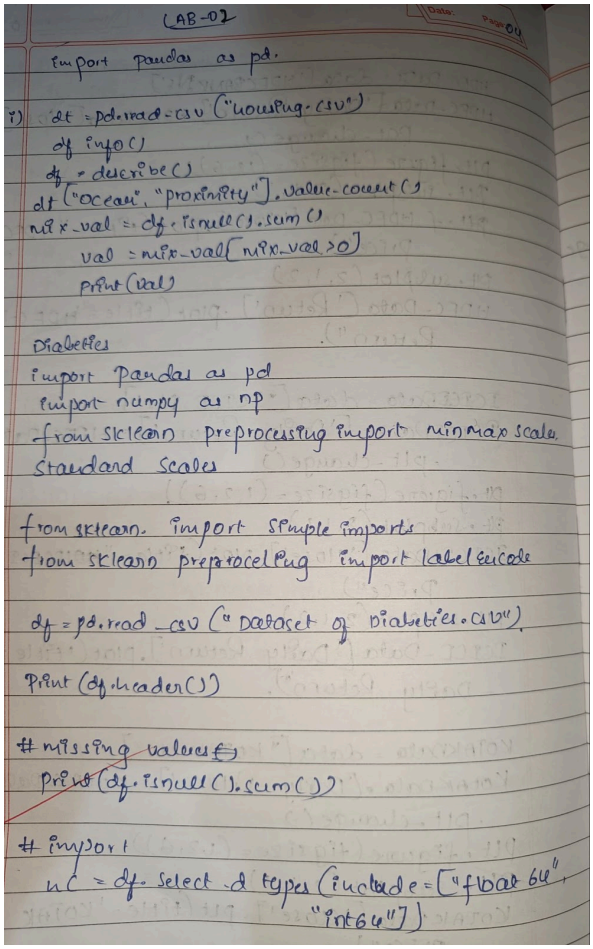
```
plt.figure(figsize=(12, 6))

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="HDFC Bank - Daily Returns", color='orange')

plt.tight_layout()

plt.show()


ICICIBANK_Data = data['ICICIBANK.NS']

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


plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)

ICICIBANK_Data['Close'].plot(title="ICICIBANK_Data Bank - Closing Price")

plt.subplot(2, 1, 2)

ICICIBANK_Data['Daily Return'].plot(title="ICICIBANK_Data Bank - Daily Returns",
color='orange')

plt.tight_layout()

plt.show()


KOTAKBANK_Data = data['KOTAKBANK.NS']

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


plt.figure(figsize=(12, 6))
```

```
plt.subplot(2, 1, 1)

KOTAKBANK_Data['Close'].plot(title="KOTAKBANK_Data Bank - Closing Price")

plt.subplot(2, 1, 2)

KOTAKBANK_Data['Daily Return'].plot(title="KOTAKBANK_Data Bank - Daily Returns",
color='orange')

plt.tight_layout()

plt.show()
```

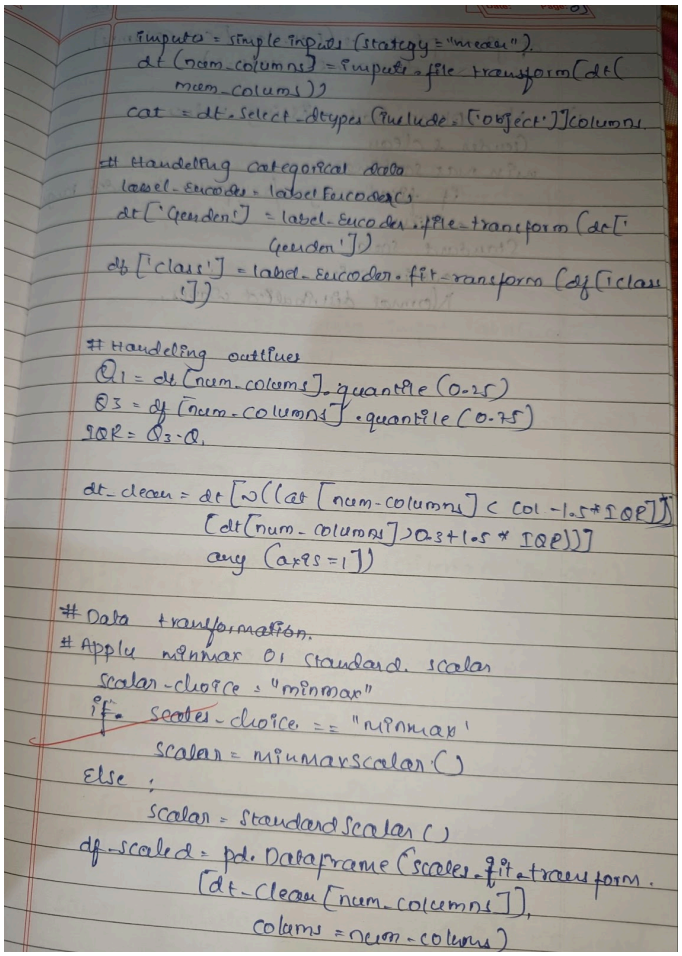
Program 2

Demonstrate various data pre-processing techniques for a given dataset

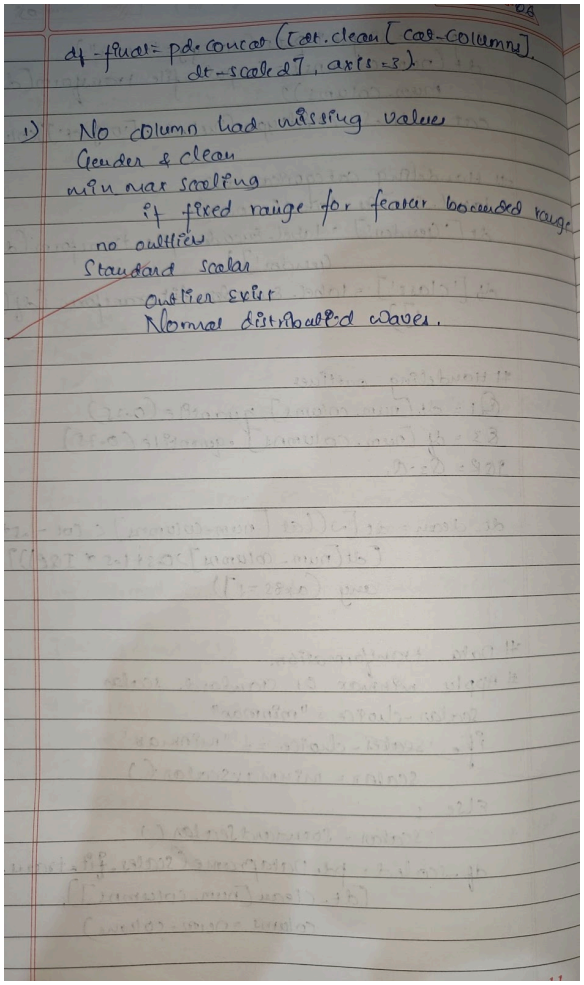
Screenshot :



```
LAB-02
import pandas as pd.
i) dt = pd.read_csv("housing.csv")
df.info()
df.describe()
dt["ocean_proximity"].value_counts()
nix_val = df.isnull().sum()
val = nix_val[nix_val>0]
print(val)

Diabetics
import pandas as pd
import numpy as np
from sklearn preprocessing import minmax scale,
Standard Scales

from sklearn. import simple imports
from sklearn preprocelling import labelencode

df = pd.read_csv("Dataset of Diabetics.csv").

Print(df.header())

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

# import
nC = df.select_d types(include=["float 64",
                               "int64"])
```



```
imputer = simple imputer(strategy="median")
dt[num_columns] = imputer.file_transform(dt[
    num_columns])
cat = dt.select_dtypes(include=['object']).columns

# Handelling categorical data
label_encoder = label Encoder()
dt['Gender'] = label_encoder.file_transform(dt['
                Gender'])
df['class'] = label_encoder.fit_transform(df['class
              '])

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

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

# Data transformation.
# Apply minmax or standard. scalar
scalar_choice = "minmax"
if scales_choice == "minmax"
    scalar = minmaxscalar()
Else:
    scalar = StandardScalar()
df_scaled = pd.Dataframe(scales.fit.transform.
            [dt_clean[num_columns]],
            colums = num_columns)
```



Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder

from sklearn.impute import SimpleImputer

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline


diabetes_df = pd.read_csv("/content/Dataset of Diabetes .csv")
```

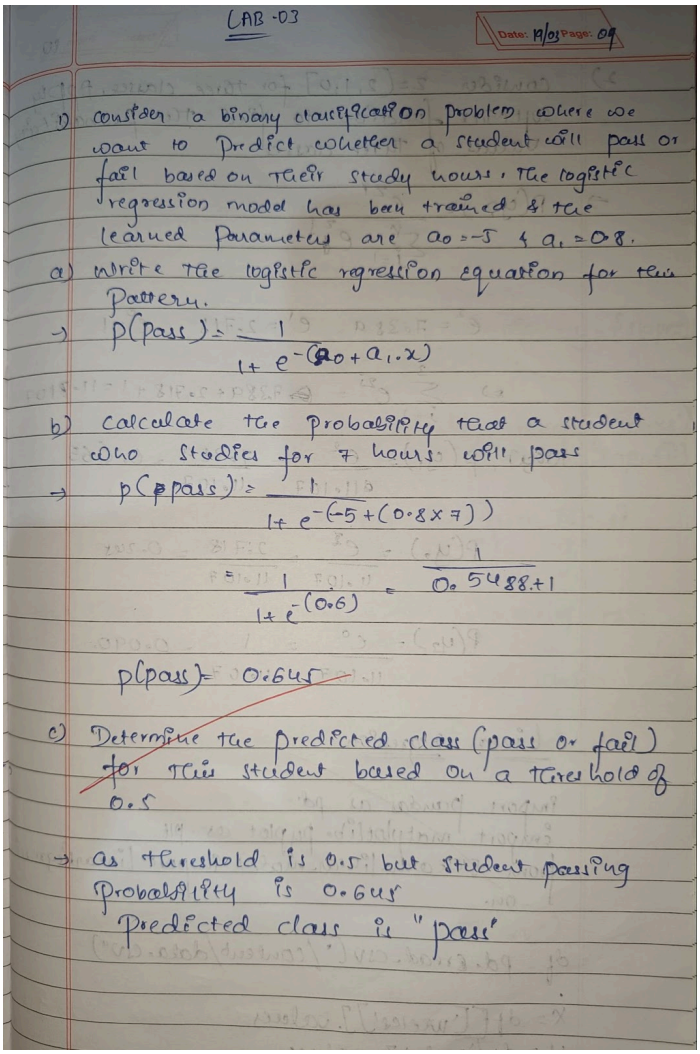
```
adult_income_df = pd.read_csv("/content/adult.csv")


def preprocess_data(df):
    missing_values = df.isnull().sum()
    print("Columns with missing values:\n", missing_values[missing_values > 0])

    numerical_cols = df.select_dtypes(include=["float64", "int64"]).columns
    categorical_cols = df.select_dtypes(include=["object"]).columns

    imputer = SimpleImputer(strategy='most_frequent')
    df[categorical_cols] = imputer.fit_transform(df[categorical_cols])

    imputer = SimpleImputer(strategy='median')
    df[numerical_cols] = imputer.fit_transform(df[numerical_cols])

    print("\nCategorical Columns:\n", categorical_cols)

    encoder = OneHotEncoder(drop='first', sparse_output=False)
    encoded_categorical_data = encoder.fit_transform(df[categorical_cols])

    encoded_categorical_df = pd.DataFrame(encoded_categorical_data,
columns=encoder.get_feature_names_out(categorical_cols))

    df = df.drop(categorical_cols, axis=1)
    df = pd.concat([df, encoded_categorical_df], axis=1)
```

```
    from scipy import stats
    z_scores = np.abs(stats.zscore(df[numerical_cols]))
    df = df[(z_scores < 3).all(axis=1)]

    min_max_scaler = MinMaxScaler()
    df[numerical_cols] = min_max_scaler.fit_transform(df[numerical_cols])

    standard_scaler = StandardScaler()
    df[numerical_cols] = standard_scaler.fit_transform(df[numerical_cols])

    return df

diabetes_df = preprocess_data(diabetes_df)
print("\nPreprocessed Diabetes Dataset:")
print(diabetes_df.head())

adult_income_df = preprocess_data(adult_income_df)
print("\nPreprocessed Adult Income Dataset:")
print(adult_income_df.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:


LAB -03

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 & the learned parameters are $a_0 = -5$ & $a_1 = 0.8$.

a) Write the logistic regression equation for this Pattern.

→ $P(\text{Pass}) = \frac{1}{1 + e^{-(a_0 + a_1 \cdot x)}}$

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

→ $P(\text{pass}) = \frac{1}{1 + e^{-(-5 + (0.8 \times 7))}}$

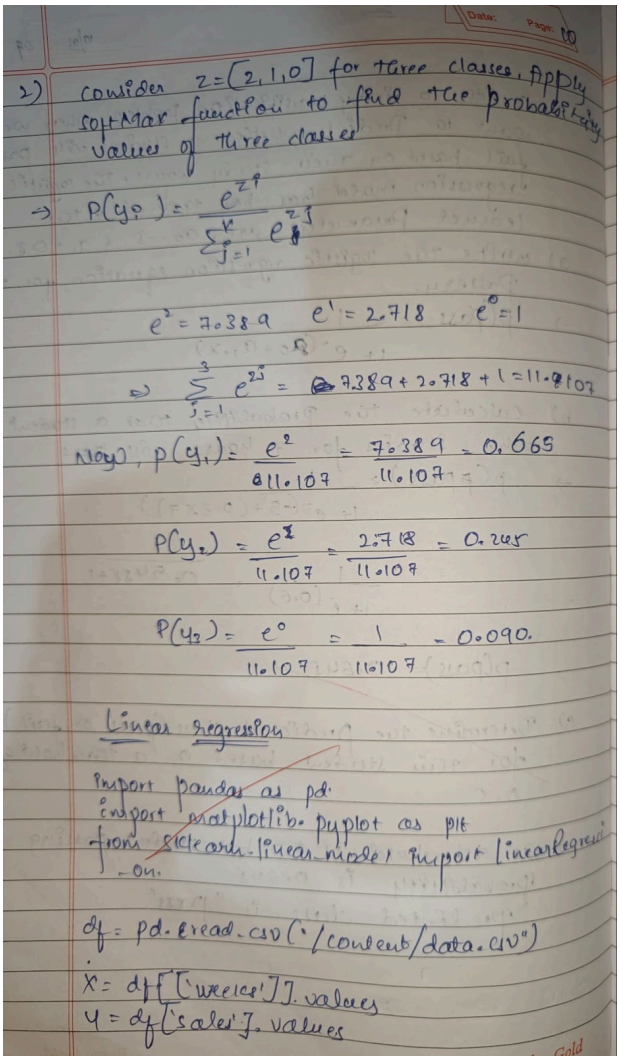
$= \frac{1}{1 + e^{-(0.6)}} = \frac{1}{0.5488 + 1}$

$P(\text{pass}) = 0.645$

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

→ as threshold is 0.5 but student passing probability is 0.645

Predicted class is "pass"


2) consider $z = [2, 1, 0]$ for three classes. Apply softMax function to find the probability values of three classes

→ $P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

$e^2 = 7.389 \quad e^1 = 2.718 \quad e^0 = 1$

$\sum_{j=1}^{3} e^{z_j} = 7.389 + 2.718 + 1 = 11.107$

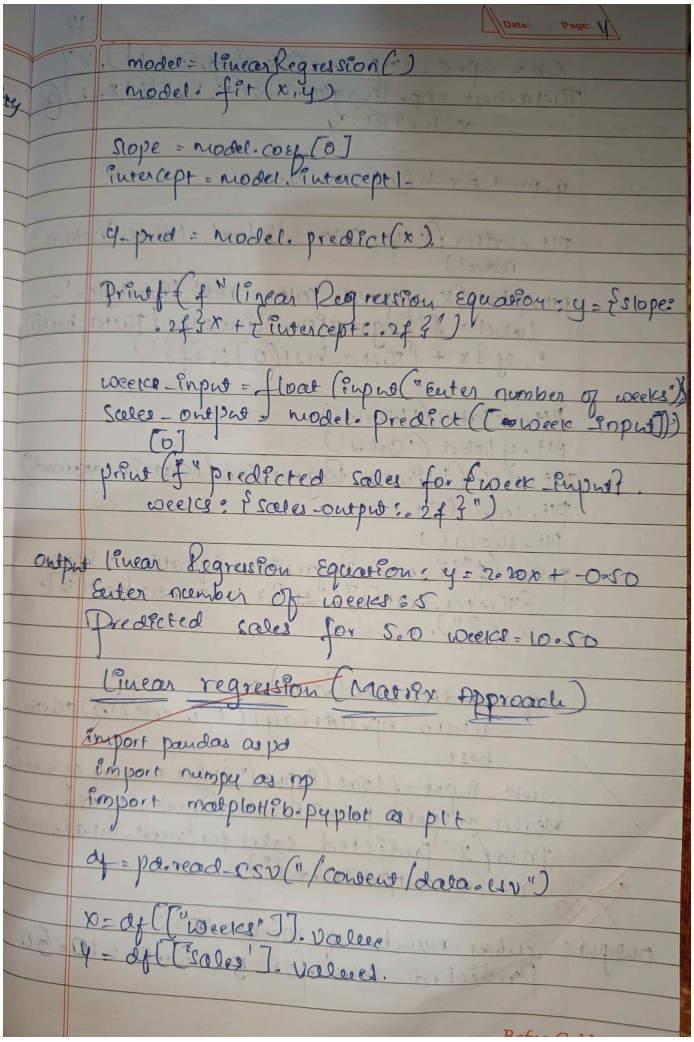
Now, $P(y_1) = \frac{e^2}{11.107} = \frac{7.389}{11.107} = 0.665$

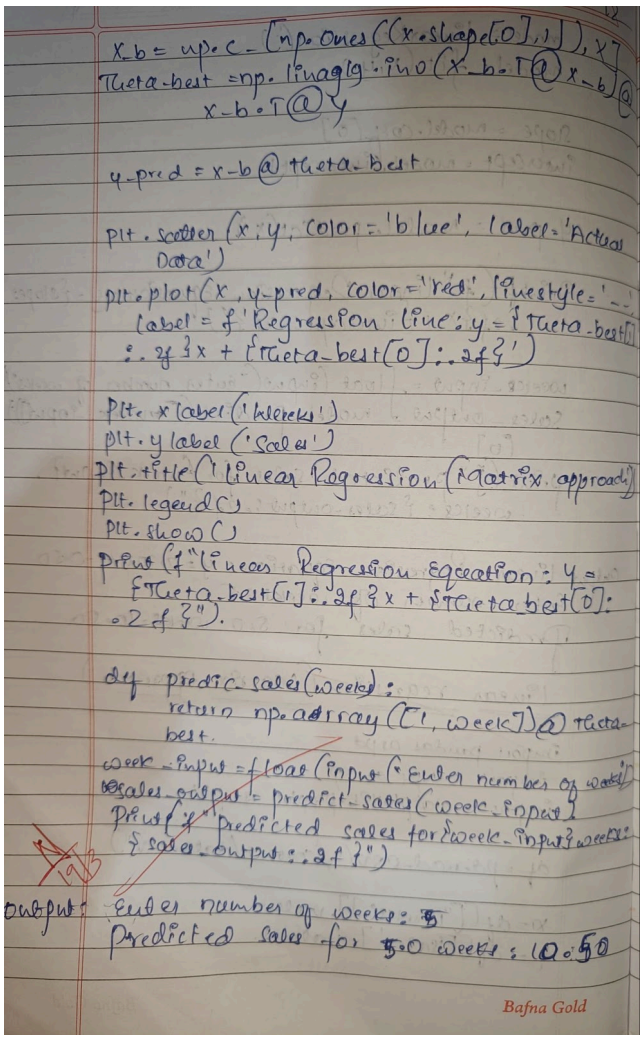
$P(y_2) = \frac{e^1}{11.107} = \frac{2.718}{11.107} = 0.245$

$P(y_3) = \frac{e^0}{11.107} = \frac{1}{11.107} = 0.090$

Linear Regression

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
df = pd.read_csv("/content/data.csv")
X = df[['weight']].values
y = df['sales'].values
```





Code:

Linear regression

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

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

X = df[['weeks']].values
y = df['sales'].values

model = LinearRegression()
model.fit(X, y)

slope = model.coef_[0]
```

```
intercept = model.intercept_

y_pred = model.predict(X)

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', linestyle='--', label=f'Regression Line: y = {slope:.2f}x + {intercept:.2f}')
plt.xlabel('Weeks')
plt.ylabel('Sales')
plt.title('Linear Regression')
plt.legend()
plt.show()

print(f"Linear Regression Equation: y = {slope:.2f}x + {intercept:.2f}")

week_input = float(input("Enter number of weeks: "))
sales_output = model.predict([[week_input]])[0]
print(f"Predicted sales for {week_input} weeks: {sales_output:.2f}")
```

Multi-linear regression

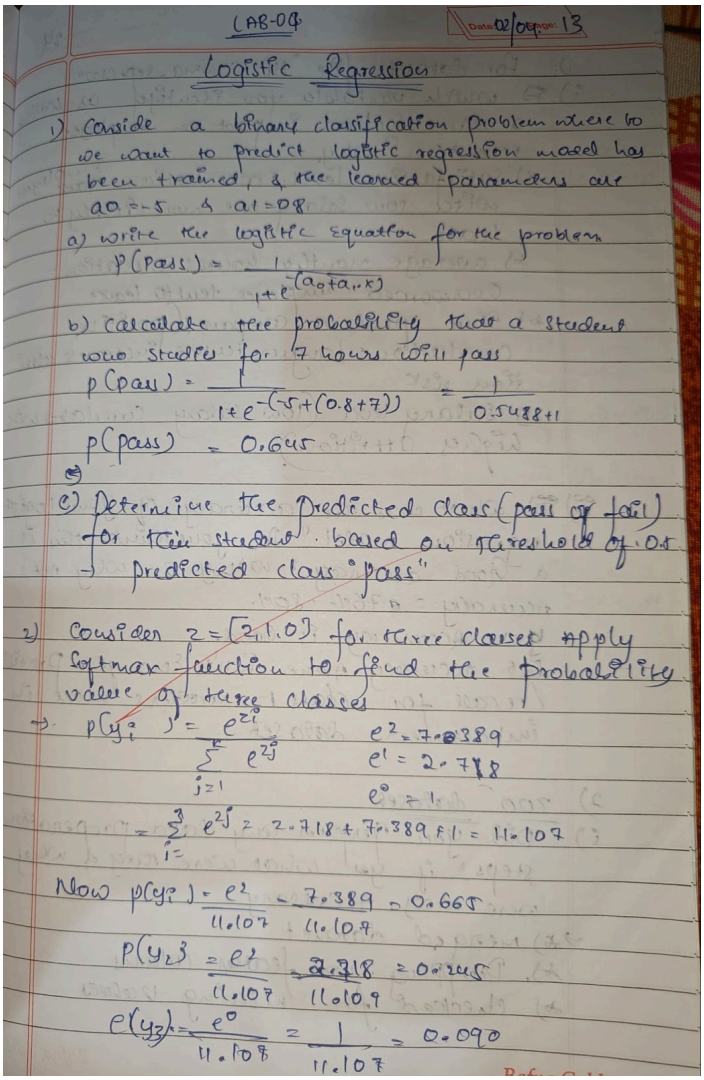
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

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

X = df[['weeks']].values
y = df['sales'].values

X_b = np.c_[np.ones((X.shape[0], 1)), X]
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y

y_pred = X_b @ theta_best

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', linestyle='--', label=f'Regression Line: y = {theta_best[1]:.2f}x + {theta_best[0]:.2f}')
plt.xlabel('Weeks')
plt.ylabel('Sales')
plt.title('Linear Regression (Matrix Approach)')
plt.legend()
plt.show()

print(f"Linear Regression Equation: y = {theta_best[1]:.2f}x + {theta_best[0]:.2f}")
```

```
def predict_sales(week):
    return np.array([1, week]) @ theta_best

week_input = float(input("Enter number of weeks: "))
sales_output = predict_sales(week_input)
print(f"Predicted sales for {week_input} weeks: {sales_output:.2f}")
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:



LAB-04

Logistic Regression

1) Conside a binary classification problem where to we want to predict logistic regression model has been trained, & the learned parameters are a0 = -5 & a1 = 0.8

a) write the logistic equation for the problem

$$p(\text{Pass}) = \frac{1}{1+e^{-(a_0+a_1 x)}}$$

b) calculate the probability that a student who studies for 7 hours will pass

$$p(\text{pass}) = \frac{1}{1+e^{-(-5+(0.8*7))}} = \frac{1}{0.5488+1}$$

$$p(\text{pass}) = 0.645$$

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

→ predicted class "pass"

2) Consider z = [2, 1, 0] for three classes apply Softmax function to find the probability value of three classes

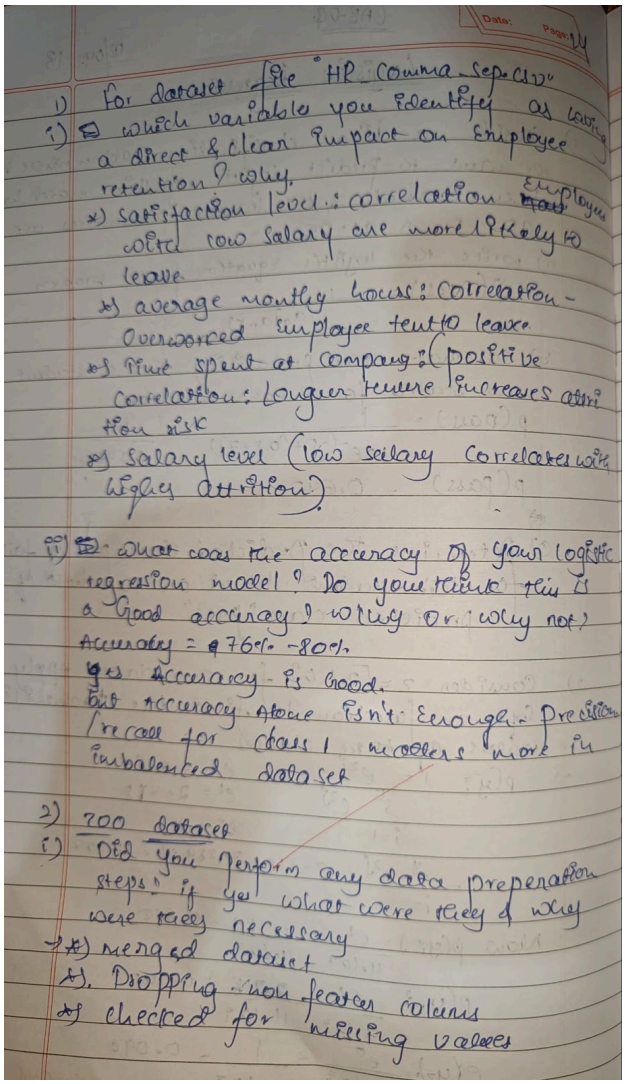
$$p(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \qquad e^2 = 7.389 \quad e^1 = 2.718 \quad e^0 = 1$$

$$= \sum_{i=}^{3} e^{z_j} = 2.718 + 7.389 + 1 = 11.107$$

Now $p(y_i) = \frac{e^2}{11.107} = \frac{7.389}{11.107} = 0.665$

$$p(y_2) = \frac{e^1}{11.107} = \frac{2.718}{11.107} = 0.245$$

$$e(y_3) = \frac{e^0}{11.108} = \frac{1}{11.107} = 0.090$$



1) For dataset file "HR_comma_sep.csv"

i) which variable you identified as having a direct & clear impact on Employee retention? why.

*) satisfaction level: correlation Employees with low salary are more likely to leave

*) average monthly hours: Correlation - Overworked employee tend to leave

*) Time spent at company: (positive correlation: longer tenure increases attrition risk

*) Salary level (low salary correlates with higher attrition)

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

Accuracy = 76% - 80%

Yes Accuracy is Good.

But Accuracy Alone isn't Enough. Precision / recall for class 1 matters more in imbalanced dataset

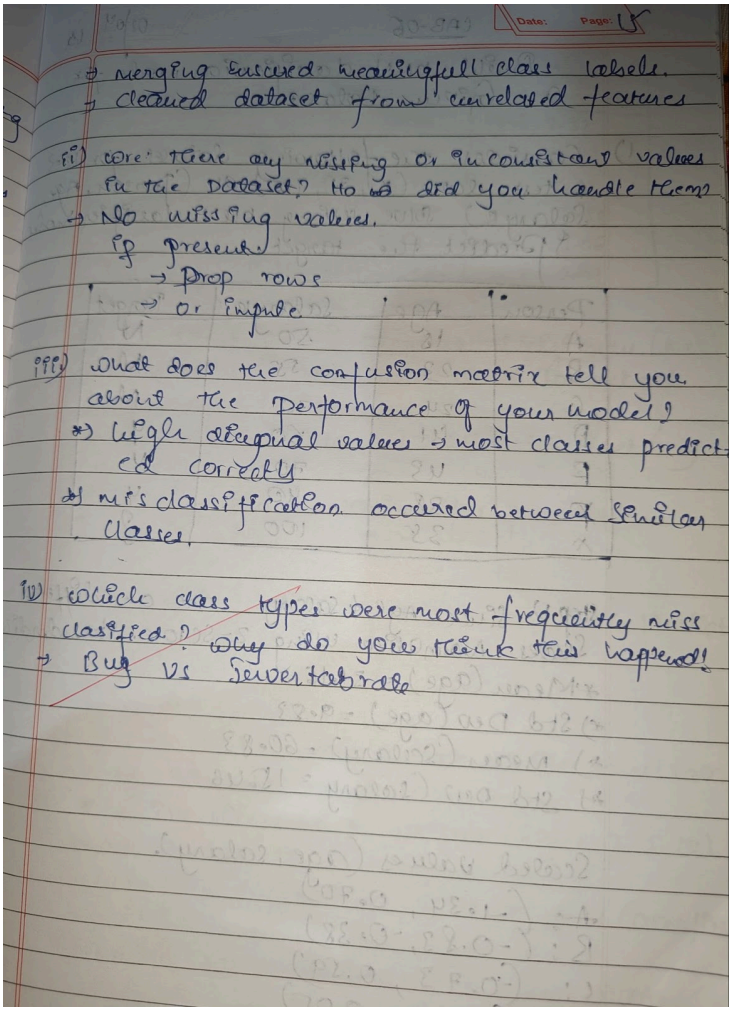
2) 200 datasets

i) Did you perform any data preperation steps? if yes what were they & why were they necessary

*) merged dataset

*) Dropping non feature columns

*) checked for missing values



Code:

Binary logistic regression

```
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.bought_insurance, marker='+', color='red')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']], df.bought_insurance, train_size=0.9, random_state=10)
X_train.shape

X_test
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

model.fit(X_train, y_train)

X_test

y_test

y_predicted = model.predict(X_test)
y_predicted

model.score(X_test, y_test)

model.predict_proba(X_test)

y_predicted = model.predict([[60]])
y_predicted

model.coef_

model.intercept_

import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

def prediction_function(age):
  z = 0.127 * age - 4.973
  y = sigmoid(z)
  return y

age = 35
prediction_function(age)
```

Multiclass logistic regression

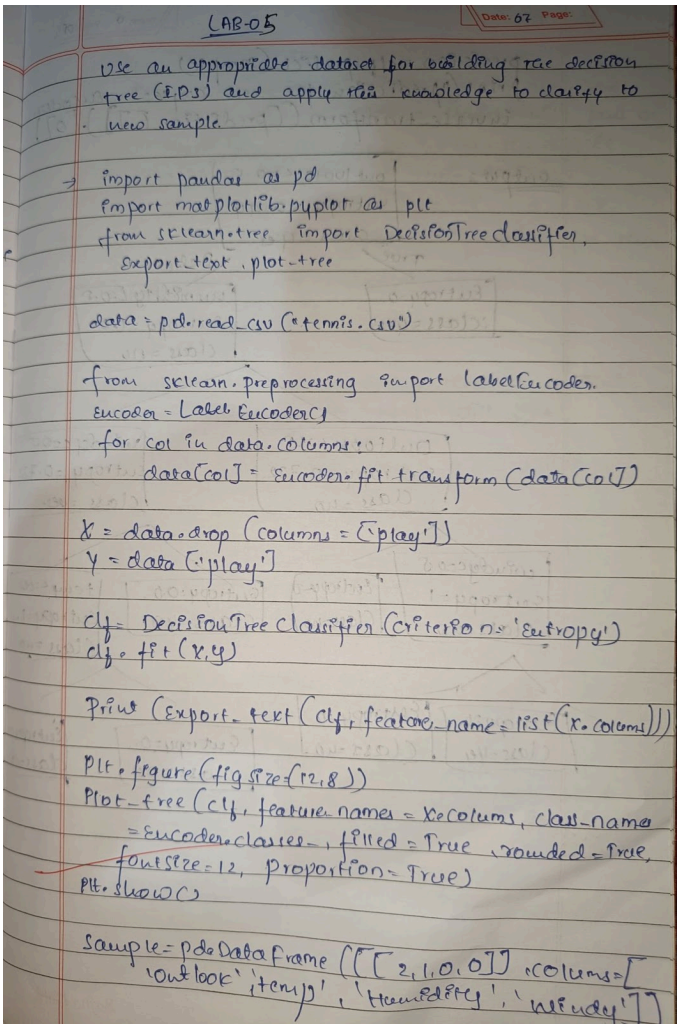
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt

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

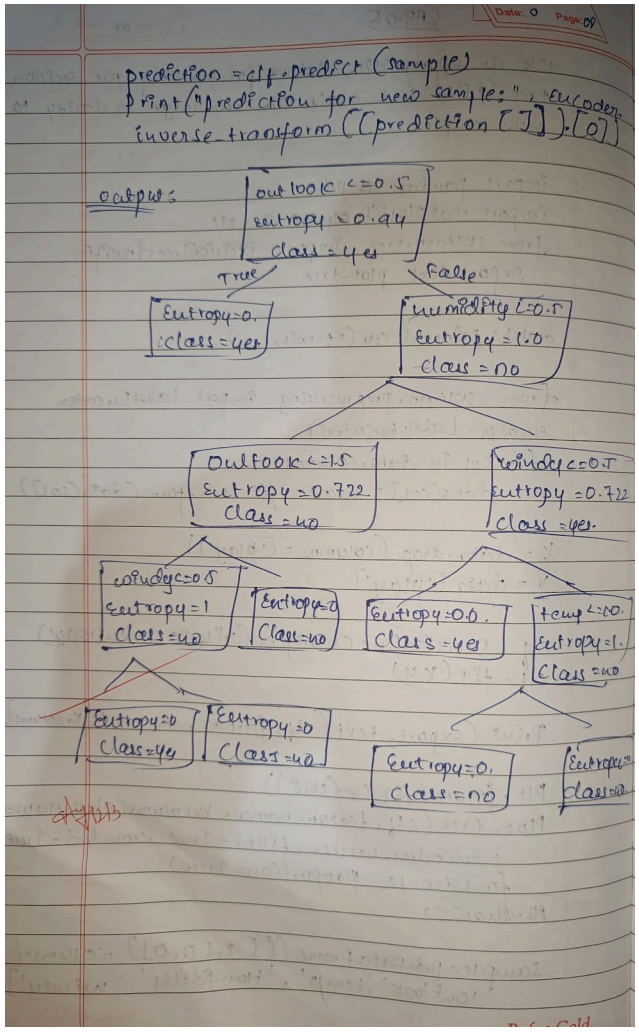
```
iris.head()

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

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

model = LogisticRegression(multi_class='multinomial')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,
display_labels=["Setosa", "Versicolor", "Virginica"])

cm_display.plot()
plt.show()
```

Program 5

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

Screenshot:





Code:

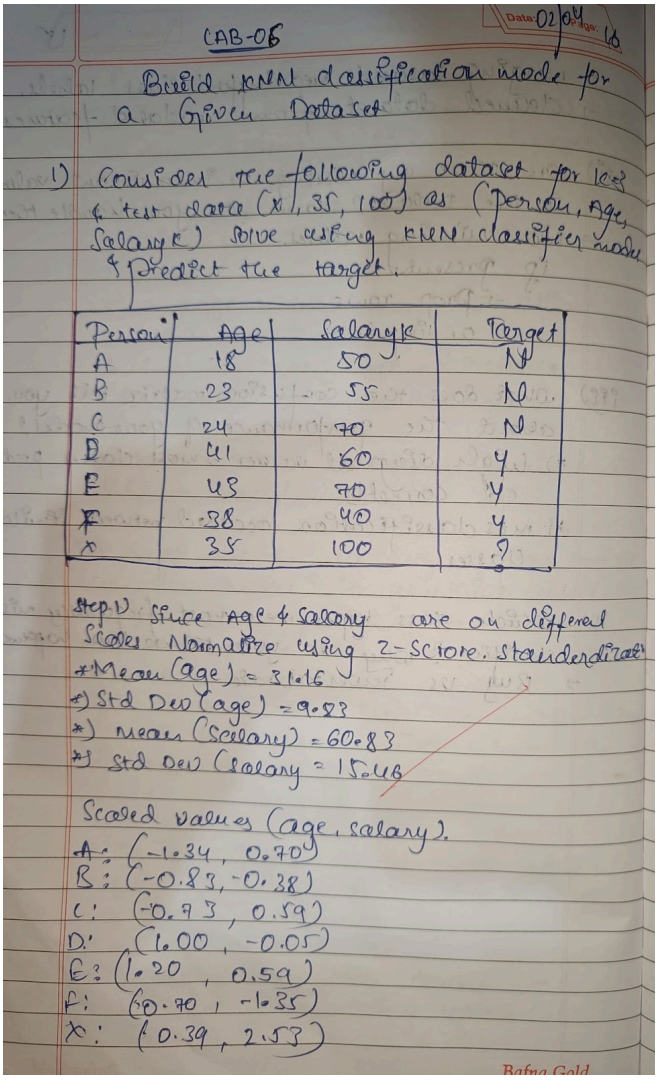
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
import matplotlib.pyplot as plt

data = pd.read_csv('/content/seattle-weather.csv')
data_cleaned = data.drop('date', axis=1)

label_encoder = LabelEncoder()
data_cleaned['weather'] = label_encoder.fit_transform(data_cleaned['weather'])

X = data_cleaned.drop('weather', axis=1)
y = data_cleaned['weather']

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

id3_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
id3_classifier.fit(X_train, y_train)

plt.figure(figsize=(20, 12))
tree.plot_tree(id3_classifier, feature_names=X.columns, class_names=list(label_encoder.classes_),
filled=True, fontsize=10)
plt.title("ID3 Decision Tree Classifier (Enhanced Clarity)")
plt.show()
```

Program 6

Build KNN Classification model for a given dataset

Screenshot:



LAB-06

Build KNN classification model for a Given Dataset

1) Consider the following dataset for test data (x, 35, 100) as (Person, Age, Salary,) Solve using KNN classifier model & predict the target.

Person	Age	Salary	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	?

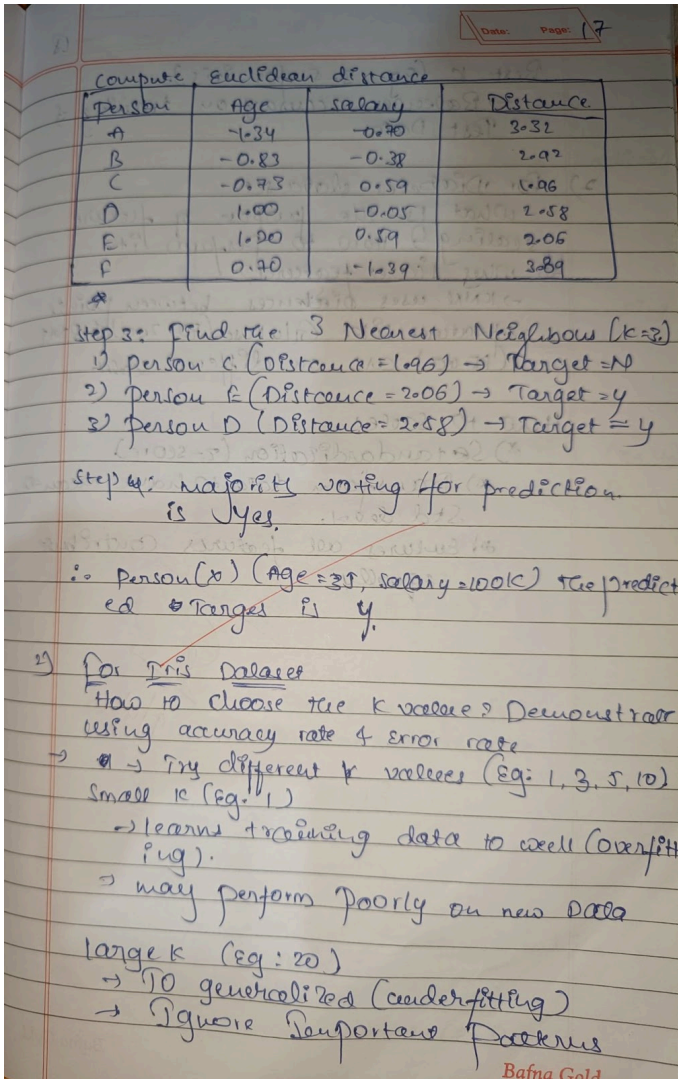
Step 1) Since Age & Salary are on different scales Normalize using z-score. standerdized

*) Mean (age) = 31.16
*) Std Dev (age) = 9.83
*) Mean (Salary) = 60.83
*) Std Dev (Salary) = 15.48

Scaled values (age, salary).

A: (-1.34, 0.70)
B: (-0.83, -0.38)
C: (-0.73, 0.59)
D: (1.00, -0.05)
E: (1.20, 0.59)
F: (0.70, -1.35)
X: (0.39, 2.53)

Bafna Gold



Compute Euclidean distance

Person	Age	salary	Distance
A	-1.34	-0.70	3.32
B	-0.83	-0.38	2.92
C	-0.73	0.59	1.96
D	1.00	-0.05	2.58
E	1.20	0.59	2.06
F	0.70	-1.39	3.89

Step 3: Find the 3 Nearest Neighbours (k=3)

1) person C (Distance = 1.96) → Target = N
2) person E (Distance = 2.06) → Target = Y
3) person D (Distance = 2.58) → Target = Y

Step 4: majority voting for prediction is Yes.

∴ Person (x) (Age = 35, salary = 100k) the predicted Target is Y.

2) For Iris Dataset

How to choose the k value? Demonstrate using accuracy rate & error rate

→ a) → Try different k values (Eg: 1, 3, 5, 10)

Small k (Eg. 1)

→ learns training data to well (Overfitting).

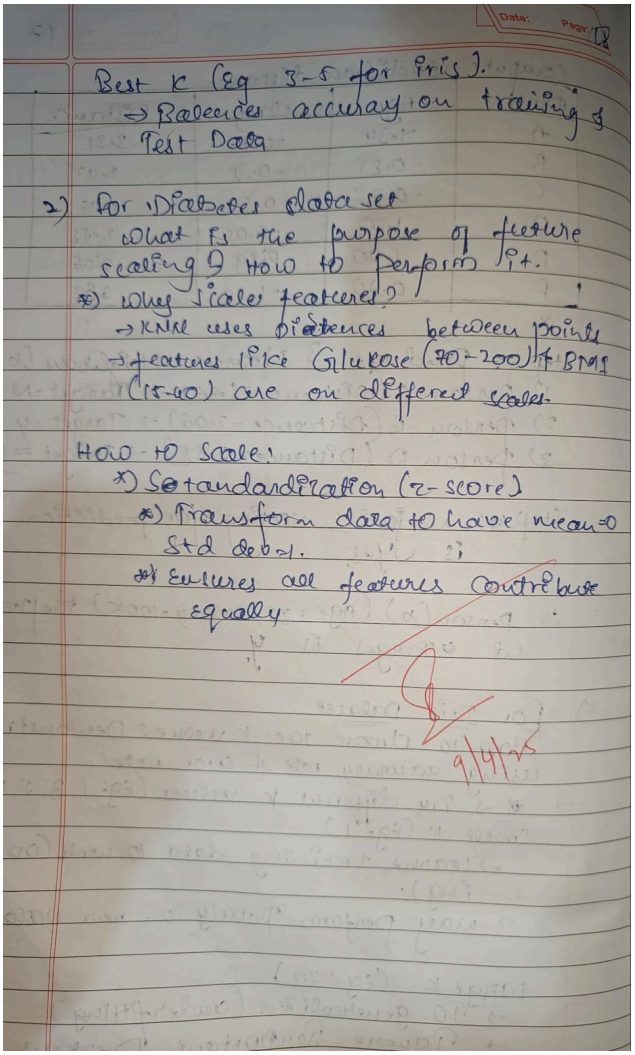
→ may perform poorly on new data

large k (Eg: 20)

→ TO generalized (underfitting)

→ Ignore Important Patterns

Bafna Gold



Code:

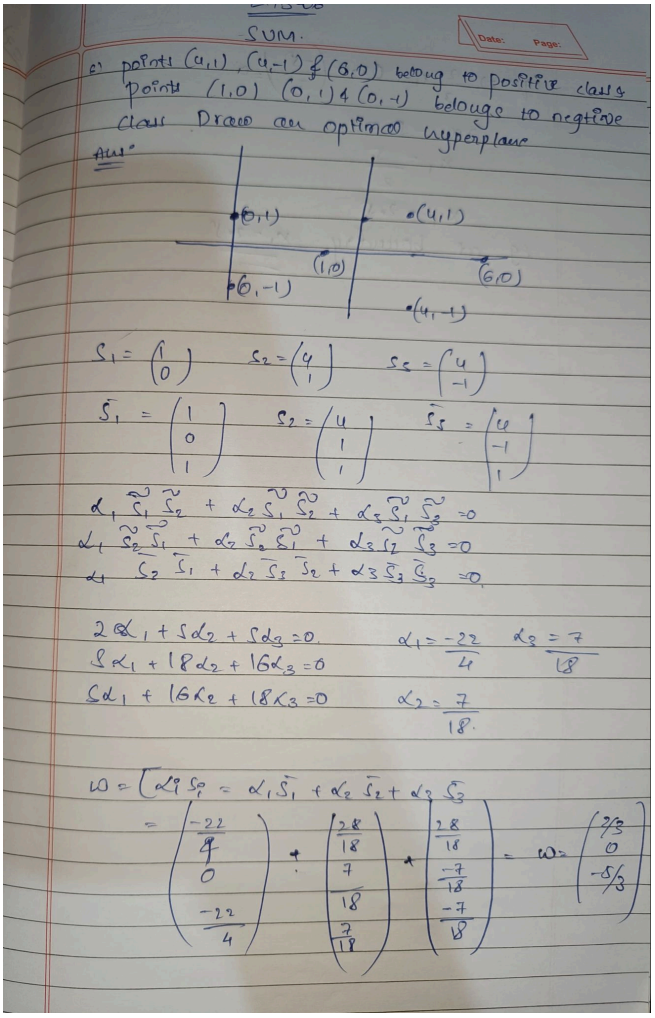
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,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

heart_df = pd.read_csv("/content/heart.csv")
X_heart = heart_df.iloc[:, :-1].values
y_heart = heart_df.iloc[:, -1].values

X_train_heart, X_test_heart, y_train_heart, y_test_heart = train_test_split(
    X_heart, y_heart, test_size=0.20, random_state=42
```

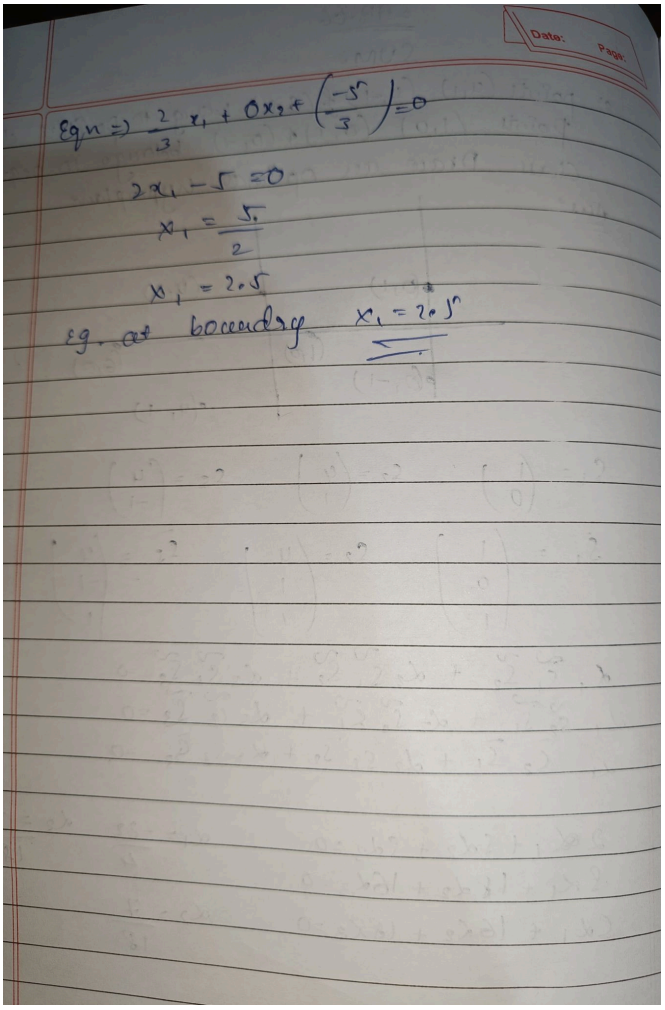
```
)

scaler = StandardScaler()
X_train_heart = scaler.fit_transform(X_train_heart)
X_test_heart = scaler.transform(X_test_heart)

param_grid = {'n_neighbors': range(1, 21)}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_heart, y_train_heart)
best_k = grid_search.best_params_['n_neighbors']

knn_heart = KNeighborsClassifier(n_neighbors=best_k)
knn_heart.fit(X_train_heart, y_train_heart)
y_pred_heart = knn_heart.predict(X_test_heart)

print(f"[HEART RESULTS - Best k = {best_k}]")
print("Accuracy:", accuracy_score(y_test_heart, y_pred_heart))
print("Confusion Matrix:\n", confusion_matrix(y_test_heart, y_pred_heart))
print("Classification Report:\n", classification_report(y_test_heart, y_pred_heart))

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test_heart, y_pred_heart),
            annot=True, fmt="d", cmap="Blues",
            xticklabels=["No Disease", "Disease"],
            yticklabels=["No Disease", "Disease"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

k_values = range(1, 21)
accuracies = [grid_search.cv_results_['mean_test_score'][i-1] for i in k_values]
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o')
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.title("Accuracy vs. k Value (Cross-Validated)")
plt.grid()
plt.show()
```

Program 7

Build Support vector machine model for a given dataset

Screenshot:





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
```

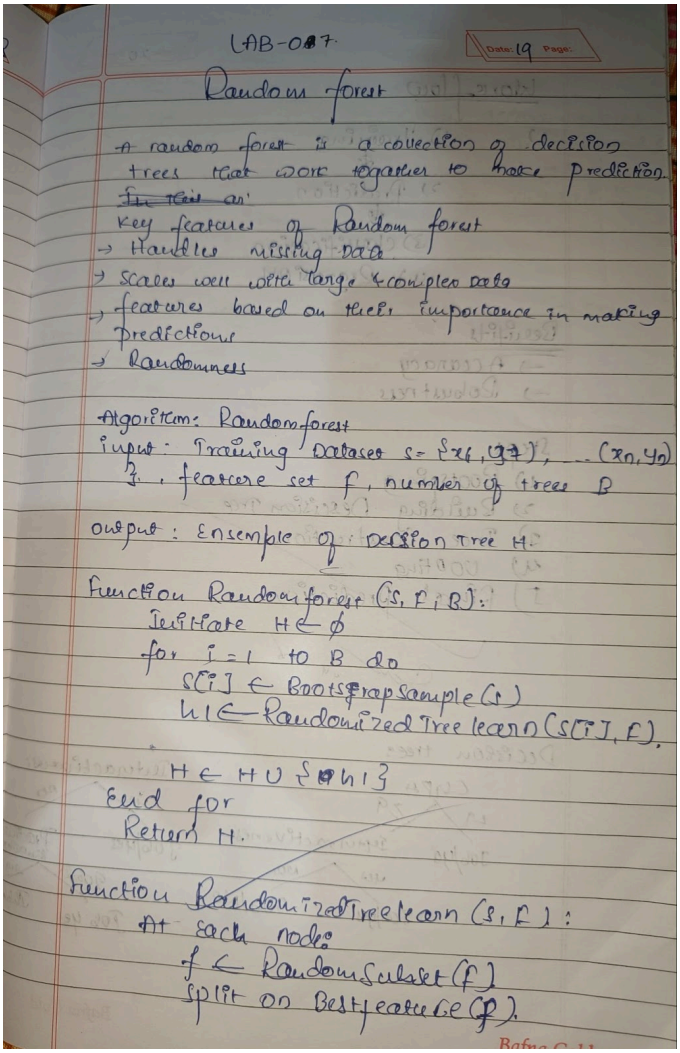
```
        self.b = None

    def fit(self, X, y):
        y = np.where(y <= 0, -1, 1)
        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)

        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 "")

        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--')

        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='*')
```

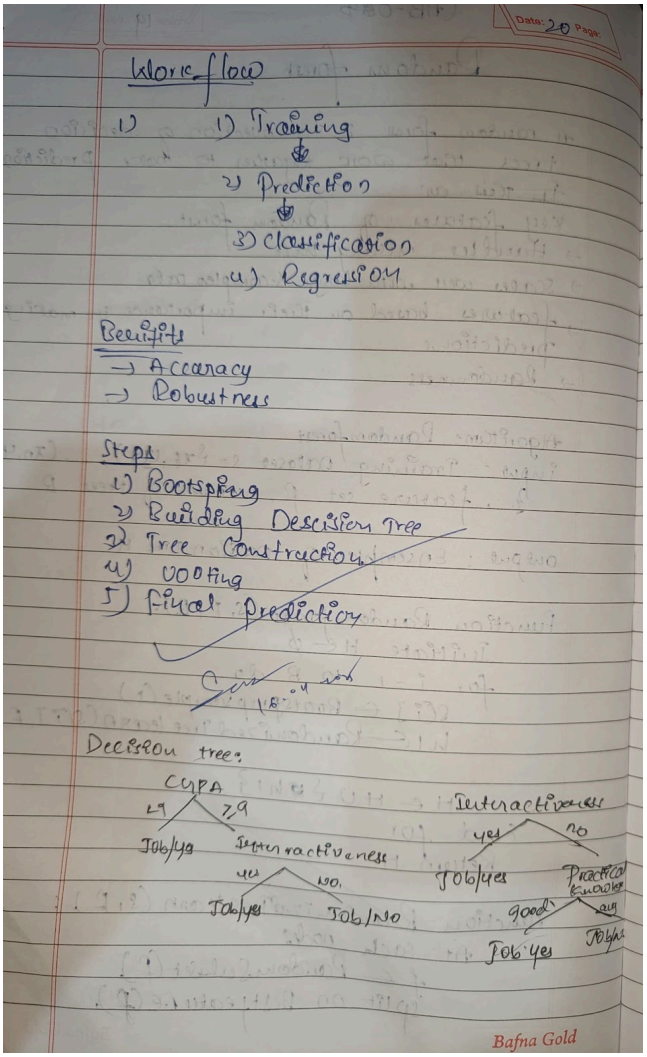
```
        ax.legend()
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
        plt.title("SVM with New Point Prediction")
        plt.grid(True)
        plt.show()

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

    new_point = np.array([[5, 5]])

    svm = SVM()
    svm.fit(X, y)
    prediction = svm.predict(new_point)[0]

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

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

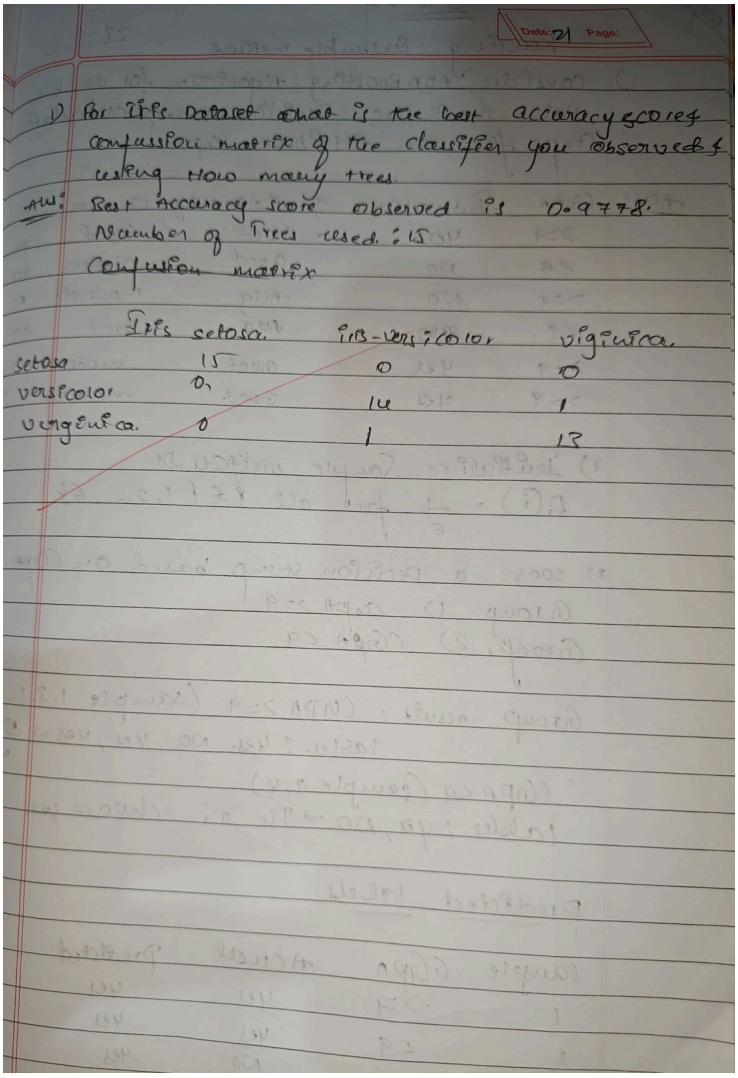
Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:







Code:

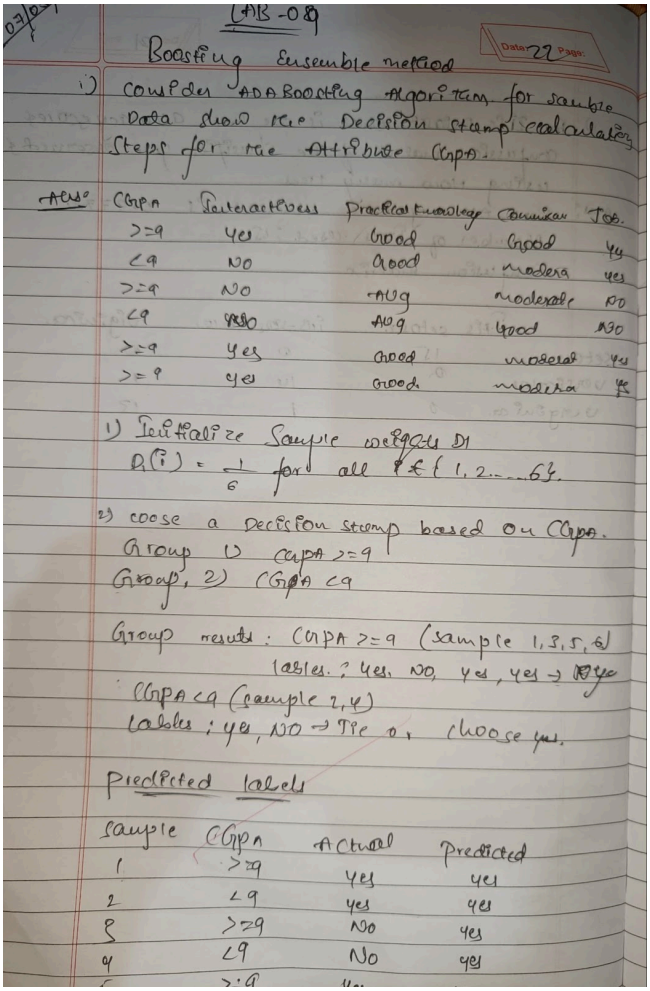
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

iris_data = pd.read_csv('/content/iris.csv')

X = iris_data.drop('species', axis=1)
y = iris_data['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

rf_default = RandomForestClassifier(random_state=42)
rf_default.fit(X_train, y_train)
```

```
y_pred_default = rf_default.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred_default)

print(f"Accuracy with default n_estimators (10): {default_accuracy:.4f}")

n_estimators_list = range(1, 101)
accuracy_scores = []

for n in n_estimators_list:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)
    if n % 20 == 0:
        print(f"n_estimators: {n}, Accuracy: {accuracy:.4f}")

best_accuracy = max(accuracy_scores)
best_n = n_estimators_list[accuracy_scores.index(best_accuracy)]

print(f"\nBest accuracy: {best_accuracy:.4f} achieved with {best_n} trees")

plt.figure(figsize=(10, 6))
plt.plot(n_estimators_list, accuracy_scores)
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.axvline(x=best_n, color='r', linestyle='--', label=f'Best n_estimators: {best_n}')
plt.legend()
plt.grid(True)
plt.show()
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:



Lab-09

Boosting Ensemble method

1) Consider AdaBoosting Algorithm for sample Data show the Decision stump calculation steps for the Attribute CGPA.

Actual	CGPA	Interactiveness	Practical knowledge	Communication	Job
	>=9	Yes	Good	Good	Yes
	<9	No	Good	Moderate	Yes
	>=9	No	Avg	Moderate	No
	<9	No	Avg	Good	No
	>=9	Yes	Good	Moderate	Yes
	>=9	Yes	Good	Moderate	Yes

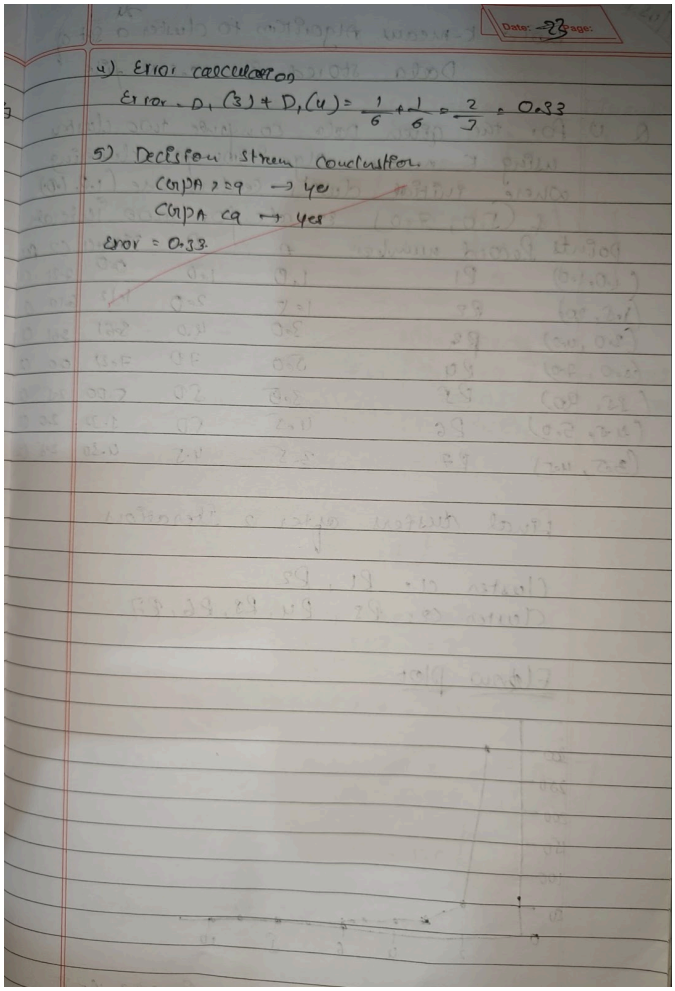
1) Initialize Sample weights D1
$D(i) = \frac{1}{6}$ for all $i \in \{1, 2, \ldots 6\}$.

2) Choose a Decision stump based on CGPA.
Group 1) CGPA >=9
Group 2) CGPA <9

Group results: CGPA >=9 (sample 1,3,5,6)
labels: Yes, No, Yes, Yes → Yes
CGPA <9 (sample 2,4)
Labels: Yes, No → Tie or choose yes.

Predicted labels

Sample	CGPA	Actual	Predicted
1	>=9	Yes	Yes
2	<9	Yes	Yes
3	>=9	No	Yes
4	<9	No	Yes



4) Error calculation
Error $= D_1(3) + D_1(4) = \frac{1}{6} + \frac{1}{6} = \frac{2}{6} = 0.33$

5) Decision stream conclusion.
CGPA >=9 → Yes
CGPA <9 → Yes
Error = 0.33.

Code:

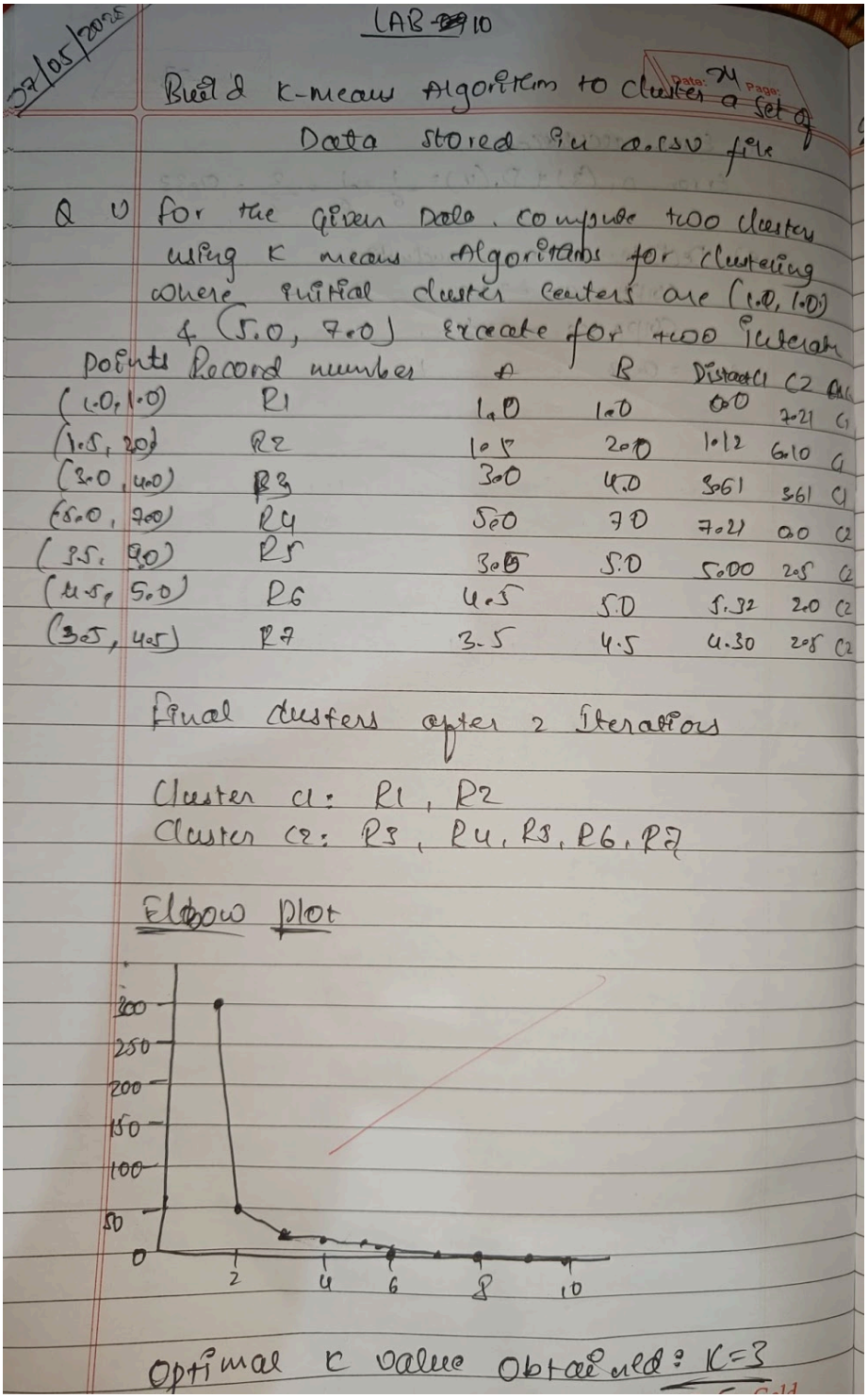
```
import pandas as pd
import numpy as np
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

df = pd.read_csv('/content/income.csv')
```

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

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

base_estimator = DecisionTreeClassifier(max_depth=1)

ada = AdaBoostClassifier(estimator=base_estimator, n_estimators=10, random_state=42)
ada.fit(X_train, y_train)
y_pred = ada.predict(X_test)
score_10 = accuracy_score(y_test, y_pred)
print(f"Accuracy with 10 estimators: {score_10:.4f}")

n_estimators_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200]
train_scores = []
test_scores = []

for n in n_estimators_range:
    ada = AdaBoostClassifier(estimator=base_estimator, n_estimators=n, random_state=42)
    ada.fit(X_train, y_train)
    train_pred = ada.predict(X_train)
    train_acc = accuracy_score(y_train, train_pred)
    train_scores.append(train_acc)
    test_pred = ada.predict(X_test)
    test_acc = accuracy_score(y_test, test_pred)
    test_scores.append(test_acc)
    print(f"n_estimators: {n:3d} | Train Accuracy: {train_acc:.4f} | Test Accuracy: {test_acc:.4f}")

best_n = n_estimators_range[np.argmax(test_scores)]
best_score = max(test_scores)
print(f"\nBest performance: {best_score:.4f} with {best_n} estimators")

plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, train_scores, label='Train Accuracy', marker='o', color='#00bcd4')
plt.plot(n_estimators_range, test_scores, label='Test Accuracy', marker='o', color='#ffcc80')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('AdaBoost Performance vs Number of Estimators')
plt.axvline(x=best_n, color='r', linestyle='--', label=f'Best n_estimators={best_n}')
plt.legend()
plt.grid(True)
plt.show()
```

Program 10

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

Screenshot:



27/05/2025

LAB 10

Build K-means Algorithm to cluster a set of Data stored in a .csv file

Q) For the given Data, compute 100 clusters using K means Algorithms for clustering where initial cluster centers are (1.0, 1.0) & (5.0, 7.0) Execute for 100 iterations

Points	Record number	x	y	Distance C1	C2	Cluster
(1.0, 1.0)	R1	1.0	1.0	0.0	7.21	C1
(1.5, 2.0)	R2	1.5	2.0	1.12	6.10	C1
(3.0, 4.0)	R3	3.0	4.0	3.61	3.61	C1
(5.0, 7.0)	R4	5.0	7.0	7.21	0.0	C2
(3.5, 5.0)	R5	3.5	5.0	5.00	2.5	C2
(4.5, 5.0)	R6	4.5	5.0	5.32	2.0	C2
(3.5, 4.5)	R7	3.5	4.5	4.30	2.5	C2

Final clusters after 2 iterations

Cluster C1: R1, R2

Cluster C2: R3, R4, R5, R6, R7

Elbow plot

Optimal K value obtained: K=3

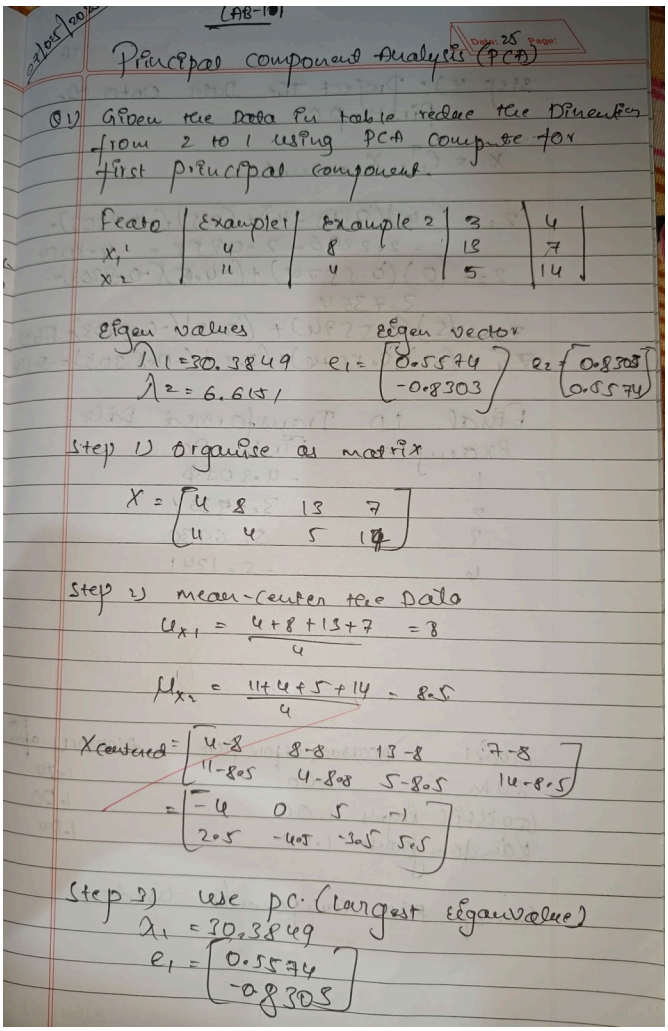
Code:

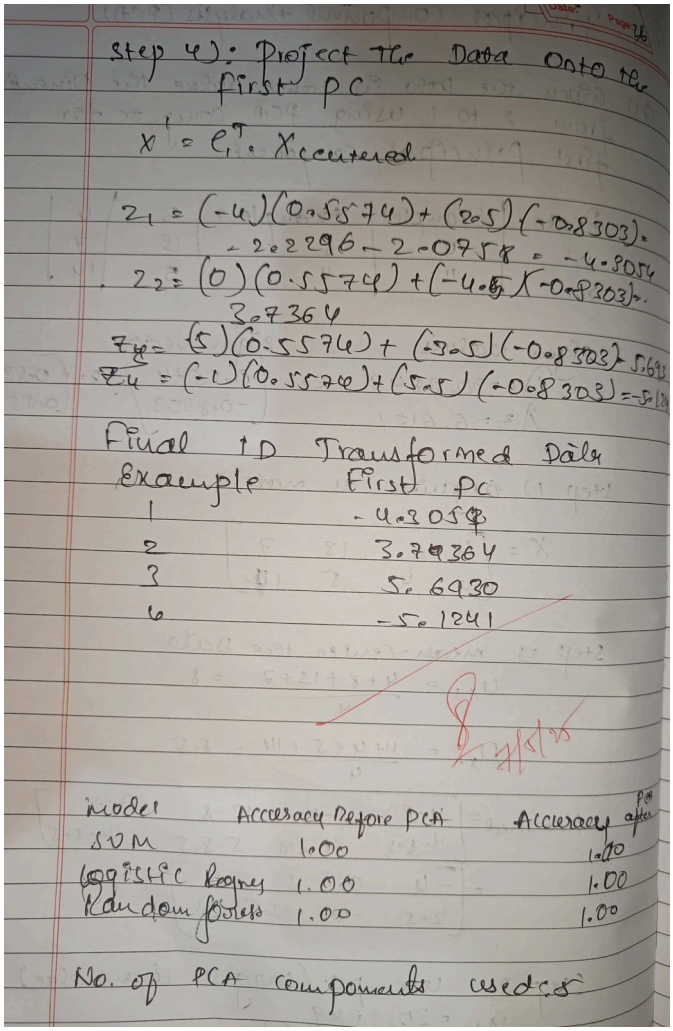
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

iris_df = pd.read_csv('/content/iris.csv')

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

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)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.show()

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)

iris_df['cluster'] = kmeans.labels_

print(iris_df.head())
```

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, StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.decomposition import PCA
```

```
df = pd.read_csv('/content/heart.csv')

binary_cols = ['Sex', 'ExerciseAngina']
multi_cols = ['ChestPainType', 'RestingECG', 'ST_Slope']

le = LabelEncoder()
for col in binary_cols:
    df[col] = le.fit_transform(df[col])

df = pd.get_dummies(df, columns=multi_cols)

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

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

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

models = {
    'SVM': SVC(random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42)
}

print("Model Performance WITHOUT PCA:")
best_acc = 0
best_model = None

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}: Accuracy = {acc:.4f}")
    if acc > best_acc:
        best_acc = acc
        best_model = name
    if name == 'Random Forest':
        print("Confusion Matrix (Random Forest):")
        print(confusion_matrix(y_test, y_pred))

pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("\nModel Performance WITH PCA:")
```

```
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} (PCA): Accuracy = {acc:.4f}")

print(f"\nBest model without PCA: {best_model} (Accuracy: {best_acc:.4f})")
print(f"Number of PCA components: {pca.n_components_}")
```