

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT
on**

Machine Learning (23CS6PCMAL)

Submitted by

Shreya Bharamanna Patil (1BM23CS420)

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 **Shreya Bharamanna Patil (1BM23CS420)**, 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 a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty Incharge	
Name: Sunayana S Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Sl. No.	Date	Experiment Title	Page No.
1	5-3-2025	Write a python program to import and export data using Pandas library functions	4
2	5-3-2025	Demonstrate various data pre-processing techniques for a given dataset	7
3	19-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	10
4	2-4-2025	Build Logistic Regression Model for a given dataset	13
5	12-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	17
6	2-4-2025	Build KNN Classification model for a given dataset	20
7	9-4-2025	Build Support vector machine model for a given dataset	25
8	7-5-2025	Implement Random Forest ensemble method on a given dataset	27
9	7-5-2025	Implement Boosting ensemble method on a given dataset	29
10	7-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file	32
11	7-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	34

Github Link:

<https://github.com/PatilShreya22/6thSem-ML-Lab>

Program 1

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

Screenshot

Lab-0
PAGE NO. 1
DATE: 05/03/2025

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

To Do:

Method 1 : Initializing values directly into DataFrame
Insert your known values, Five rows of data with column headings as 'USN', 'Name', 'Marks'?

```
→ Import pandas as pd  
data = { 'USN' : [ '1BM22CB001', '1BM22CB002',  
'1BM22CB003', '1BM22CB004', '1BM22-  
CB005' ],  
'Name' : [ 'Ankita', 'Aniket', 'Anish', 'Anil',  
'Anita' ],  
'Marks' : [ 98, 90, 45, 86, 78 ] }  
df = pd.DataFrame(data)  
print(df)
```

Method 2 : Importing datasets from sklearn.datasets
loading diabetes datasets sklearn.datasets.load_diabetes

```
→ import pandas as pd  
diabetes = load_diabetes()  
df = pd.DataFrame(diabetes, columns = diabetes.  
feature_names)  
df['target'] = diabetes.target  
print(df)
```

Method 3 : Importing Datasets from specific .csv
file sample_sales_data.csv

```
→ path = r'content/sample_sales_data.csv'  
df = pd.read_csv(path)  
print(df)
```

Lab-1
PAGE NO. 2
DATE: 05/03/2025

Method 4 : Downloading datasets from existing
dataset repositories like kaggle, UCI, Mendeley,
KEEL, etc.

```
→ path = r'content/Dataset of Diabetes.csv'  
df = pd.read_csv(path)  
print(df)
```

To Do :

1. HDFC Bank Ltd., ICICI Bank Ltd., Kotak Mahindra
Bank Ltd.
Tickers = ["HDFCBANK.NS", "ICICIBANK.NS",
"KOTAKBANK.NS"]
- import yfinance as yf
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAK-
BANK.NS"]
2. Start date : 2024-01-01, End date : 2024-12-30
→ data = yf.download(tickers, start = '2024-01-01',
end = '2024-12-30', group_by = tickers)
print(data)
3. Plot the closing price and daily returns for all
the three banks mentioned

```
→ # HDFC BANK  
HDFC = data[HDFC] data['HDFCBANK.NS']  
HDFC['Daily Return'] = HDFC['Close'].pct_change()  
plt.figure(figsize = (10,6))  
plt.subplot(2,1,1)  
HDFC['Close'].plot(title = 'HDFC BANK - Closing Price')  
plt.subplot(2,1,2)  
HDFC['Daily Return'].plot(title = 'HDFC BANK -
```

→ # KOTAK BANK
KOTAK = data['KOTAKBANK.NS']
KOTAK['Daily Return'] = KOTAK['Close'].pct_change()
plt.figure(figsize = (10,6))
plt.subplot(2,1,1)
KOTAK['Close'].plot(title = 'KOTAK BANK - Close
Price')
plt.subplot(2,1,2)
KOTAK['Daily Return'].plot(title = 'KOTAK BANK -
Daily Return', color = 'orange')
plt.tight_layout()
plt.show()

→ # ICICI BANK
ICICI = data[ICICI] data['ICICIBANK.NS']
ICICI['Daily Return'] = ICICI['Close'].pct_change()
plt.figure(figsize = (10,6))
plt.subplot(2,1,1)
ICICI['Close'].plot(title = 'ICICI BANK - Closing Price')
plt.subplot(2,1,2)
ICICI['Daily Return'].plot(title = 'ICICI BANK -

Code:

```
import pandas as pd

data={
    'USN':['1BM22CS001','1BM22CS002','1BM22CS003','1BM22CS004','1BM22CS005'],
    'Name':['Ankita','Anita','Amit','Anish','Arun'],
    'Marks':[99,56,96,85,45]
}

df=pd.DataFrame(data)
print(df)

from sklearn.datasets import load_diabetes
data=load_diabetes()
df=pd.DataFrame(data.data,columns=data.feature_names)
df['target']=data.target
print(df)

path=r"/content/sample_sales_data.csv"
df=pd.read_csv(path)
print(df)

path=r"/content/Dataset of Diabetes .csv"
df=pd.read_csv(path)
print(df.head())

import yfinance as yf
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=tickers)
print(data)

#HDFCBANK
HDFC=data['HDFCBANK.NS']
HDFC['Daily Return']=HDFC['Close'].pct_change()
print(HDFC)

plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
HDFC['Close'].plot(title='HDFC BANK - Closing Price')
plt.subplot(2,1,2)
HDFC['Daily Return'].plot(title='HDFC BANK - Daily Return',color='orange')
```

```
plt.tight_layout()
plt.show()

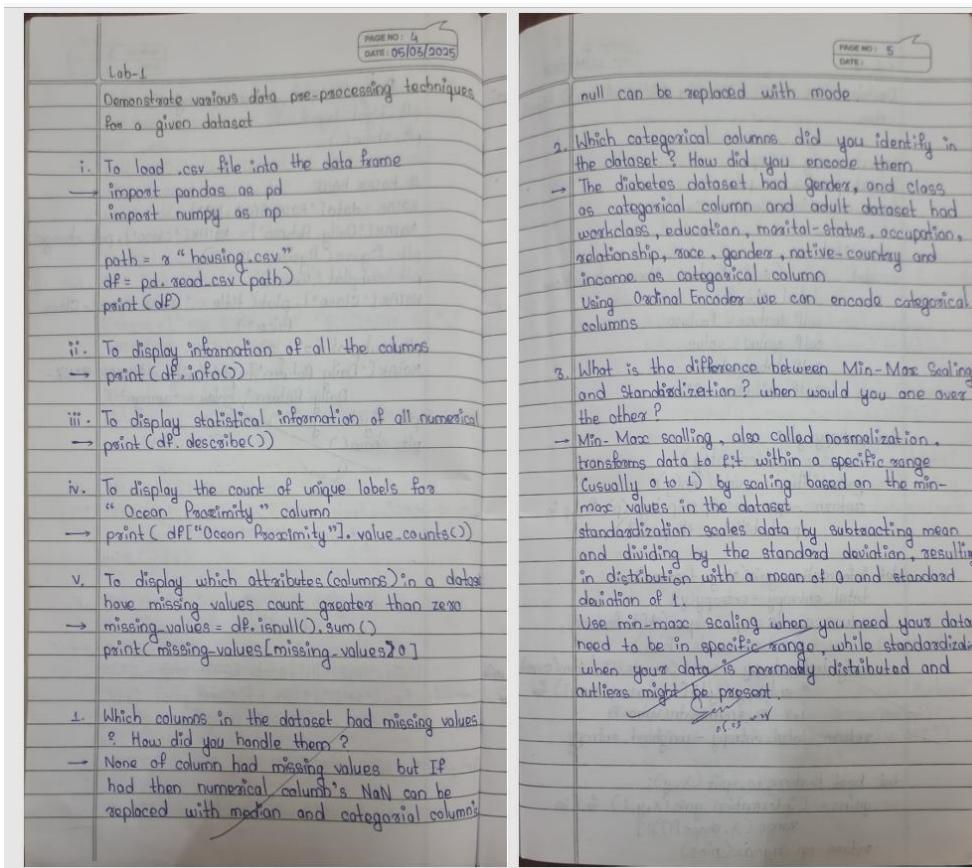
#ICICIBANK
ICICI=data['ICICIBANK.NS']
ICICI['Daily Return']=ICICI['Close'].pct_change()
print(ICICI)

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

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
df = pd.read_csv(r"/content/Dataset of Diabetes .csv")
df.head(10)
df.shape
print(df.info())
print(df.describe())
missing_values = df.isnull().sum()
```

```

# Display columns with missing values
print(missing_values[missing_values > 0])

#Set the values to some value (zero, the mean, the median, etc.).
# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean strategy for Salary
imputer1 = SimpleImputer(strategy="median")
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary" column
# Note: SimpleImputer expects a 2D array, so we reshape the column
imputer1.fit(df_copy[["AGE"]])
imputer2.fit(df_copy[["BMI"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary" column
df_copy["AGE"] = imputer1.transform(df[["AGE"]])
df_copy["BMI"] = imputer2.transform(df[["BMI"]])

# Verify that there are no missing values left
print(df_copy["AGE"].isnull().sum())
print(df_copy["BMI"].isnull().sum())
#Handling Categorical Attributes
#Using Ordinal Encoding for gender Column and One-Hot Encoding for City Column

# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories=[["M", "F", "f"]])
# Fit and transform the data
df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])

# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column
encoded_data = onehot_encoder.fit_transform(df[["CLASS"]])

# Convert the sparse matrix to a dense array
encoded_array = encoded_data.toarray()

# Convert to DataFrame for better visualization
encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df_copy, encoded_df], axis=1)

df_encoded.drop("Gender", axis=1, inplace=True)
df_encoded.drop("CLASS", axis=1, inplace=True)

print(df_encoded.head())
normalizer = MinMaxScaler()

```

```

df_encoded[['BMI']] = normalizer.fit_transform(df_encoded[['BMI']])
df_encoded.head()
scaler = StandardScaler()
df_encoded[['AGE']] = scaler.fit_transform(df_encoded[['AGE']])
df_encoded.head()
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['BMI'].quantile(0.25)
Q3 = df_encoded_copy1['BMI'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['BMI'] = np.where(df_encoded_copy1['BMI'] > upper_bound, upper_bound,
                                    np.where(df_encoded_copy1['BMI'] < lower_bound, lower_bound, df_encoded_copy1['BMI']))

print(df_encoded_copy1.head())
df_encoded_copy2['BMI_zscore'] = stats.zscore(df_encoded_copy2['BMI'])
df_encoded_copy2['BMI'] = np.where(df_encoded_copy2['BMI_zscore'].abs() > 3, np.nan, df_encoded_copy2['BMI'])
# Replace outliers with NaN
print(df_encoded_copy2.head())
df_encoded_copy3['BMI_zscore'] = stats.zscore(df_encoded_copy3['BMI'])
median_salary = df_encoded_copy3['BMI'].median()
df_encoded_copy3['BMI'] = np.where(df_encoded_copy3['BMI_zscore'].abs() > 3, median_salary,
df_encoded_copy3['BMI'])
print(df_encoded_copy3.head())

```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

PAGE NO: 8
DATE: 13/03/2020

Lab - 3

Linear Regression

```

import pandas as pd
import matplotlib.pyplot as plt

data = {'X': [1, 2, 3, 4, 5], 'Y': [1, 2, 3, 8, 2, 6, 3, 2, 3, 8]}

df = pd.DataFrame(data)
X1 = df['X'].mean()
Y1 = df['Y'].mean()
df['X1**2'] = [X1**2 for X1 in df['X']]
X1_sq = df['X1**2'].mean()
x1_sq = []
x1 = df['X1']
y1 = df['Y1']

for i in range(len(x1)):
    x1_sq.append(x1[i]**2)
df['X1**2'] = x1_sq
X1_sq2 = df['X1**2'].mean()

a1 = (df['X1*Y1'].sum() - len(df)*X1*Y1)/
      (df['X1'].apply(lambda x: x**2).sum() -
       len(df)*X1**2)
a0 = Y1 - a1*X1

a = int(input())
y = a0 + a1*a
print(y)

plt.scatter(df['X'], df['Y'], color = 'blue', label =
            'Data points')
plt.plot(df['X'], a0 + a1*df['X'], color = 'red', label =
            'Fitted line')

```

PAGE NO: 15
DATE: _____

"Regression Line")
plt. title ("Linear Regression (Matrix Form)")
plt. xlabel ("X (Features)")
plt. ylabel ("Y (Target)")
plt. legend()
plt. show()

Linear Regression

• Data Points

Regression Line

Output:
Slope (m): 2.20
Intercept (b): -1.5

Code:

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

data={"X":[1,2,3,4,5],
      "Y":[1.2,1.8,2.6,3.2,3.8]}
df=pd.DataFrame(data)
df

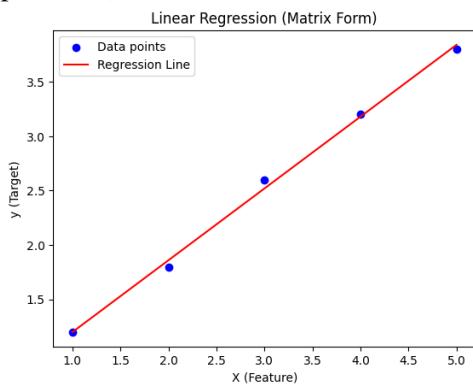
Xi=df["X"].mean()
Yi=df["Y"].mean()

df["Xi^2"]=[Xi**2 for Xi in df["X"]]
Xisq=df["Xi^2"].mean()

xiyi=[]
x=df["X"]
y=df["Y"]
for i in range(len(x)):
    xiyi.append(x[i]*y[i])
df["XiYi"]=xiyi
print(df["XiYi"])
XiYi2=df["XiYi"].mean()
print(XiYi2)
a1 = (df["XiYi"].sum() - len(df) * Xi * Yi) / (df["X"].apply(lambda x: x**2).sum() - len(df) * Xi**2)
a0 = Yi - a1 * Xi

x=9
Y=a0+a1*x
print(Y)

plt.scatter(df["X"], df["Y"], color='blue', label='Data points') # Scatter plot of original data
plt.plot(df["X"], a0 + a1 * df["X"], color='red', label='Regression Line') # Correct regression line
plt.title('Linear Regression (Matrix Form)')
plt.xlabel('X (Feature)')
plt.ylabel('y (Target)')
plt.legend()
plt.show()
```



```

import numpy as np
import matplotlib.pyplot as plt

X = np.array([1, 2, 3, 4])
y = np.array([1, 3, 4, 8])

X_matrix = np.c_[np.ones(len(X)), X]

theta = np.linalg.inv(X_matrix.T @ X_matrix) @ X_matrix.T @ y

b, m = theta

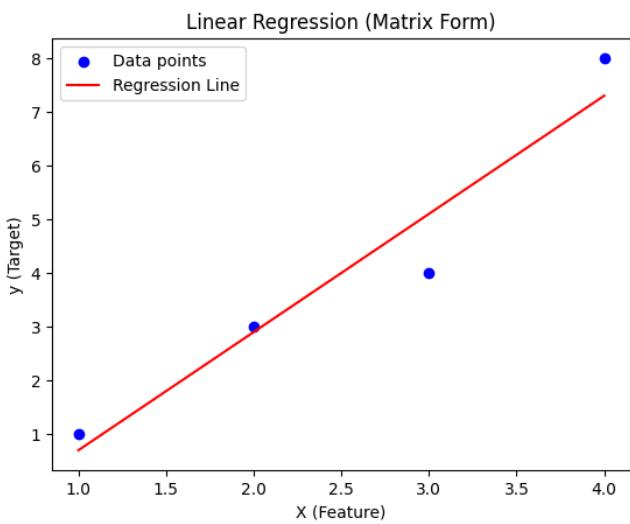
y_pred = m * X + b

print(f"Slope (m): {m}")
print(f"Intercept (b): {b}")

Slope (m): 2.2000000000000006
Intercept (b): -1.5

plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.title('Linear Regression (Matrix Form)')
plt.xlabel('X (Feature)')
plt.ylabel('y (Target)')
plt.legend()
plt.show()

```



Program 4

Build Logistic Regression Model for a given dataset

Screenshot

<p>Lab 4 Logistic Regression</p> <p>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 build and the learned parameters are $a_0 = -5$ (Intercept) and $a_1 = 0.8$ (Coefficient for study hours).</p> <p>a) Write the logistic regression equation for this problem.</p> <p>b) Calculate the probability that a student who studies for 7 hours will pass.</p> <p>c) Determine the predicted class (Pass/Fail) for this student based on a threshold of 0.5</p> <p>→ Logistic regression = $\frac{1}{1 + e^{-(a_0 + a_1 x)}}$</p> $a_0 = -5 \quad a_1 = 0.8$ <p>$P(\text{Pass} x) = \frac{1}{1 + e^{-(5 + 0.8x)}}$</p> $x = a_0 + a_1 x$ $= -5 + (0.8 \times 7)$ $= 0.6$ <p>$P(\text{Pass} 7) = \frac{1}{1 + e^{-0.6}}$</p> $= 0.6456 \approx 64.56\%$ <p>Predicted class is pass as $0.6456 > 0.5$</p> <p>2. Consider $z = [2, 1, 0]$ for these classes. Apply SoftMax function to find the probability values of these classes.</p>	<p>PAGE NO: 43 DATE: 03/04/2020</p> <p>SoftMax (z_1) = $\frac{e^{z_1}}{\sum_{j=1}^3 e^{z_j}}$</p> $z = [2, 1, 0]$ <p>SoftMax (z_1) = $\frac{e^2}{e^2 + e^1 + e^0} = 0.665$</p> <p>SoftMax ($z_2$) = $\frac{e^1}{e^2 + e^1 + e^0} = 0.2447$</p> <p>SoftMax ($z_3$) = $\frac{e^0}{e^2 + e^1 + e^0} = 0.0903$</p> <p>The probabilities of these classes = 66.5%, 24.47%, 9%</p> <p>Q. After building the logistic regression model, write the answers for the following questions</p> <p>a) For dataset file "HR-Comma-Sep.csv"</p> <ul style="list-style-type: none"> Which variables did you identify as having a direct and clear impact on employee retention? why? Variables impacting employee retention? The satisfaction level has a strong impact on retention of employee with lower satisfaction are more likely to leave. Average monthly hours and number of projects also influence retention. Overworked or underutilized employees may leave. Salary level and promotion employees with low salary and no promotions are more likely to leave. <p>b) What are the accuracy of your logistic regression model? Do you think this is a good accuracy? Why or Why not?</p>
--	---

<p>PAGE NO: 44 DATE:</p> <p>→ The logistic regression model achieved an accuracy of $x\%$.</p> <p>For accuracy -</p> <ul style="list-style-type: none"> If above 80% → Yes, it's a good model If below 70% → It may need improvement If around 75% → decent but can be improved <p>2. For Zoo dataset</p> <p>a) Did you perform any data preprocessing steps? If yes, what were they, and why were they necessary?</p> <ul style="list-style-type: none"> → Dropped 'animal_name' → Not useful for classification → Split data → 80% training, 20% testing Used multinomial Logistic Regression for multiclass classification. <p>b) Were there any missing or inconsistent values in the dataset? How did you handle them?</p> <ul style="list-style-type: none"> → No explicit missing values found. If present, they could be handled by filling or dropping rows. <p>c) What does the confusion matrix tell you about the performance of your model?</p> <ul style="list-style-type: none"> It shows how well the model classified each class. High diagonal values = good predictions, off-diagonal values = misclassifications. <p>d) Which class type was most frequently misclassified? Why do you think this happened?</p> <ul style="list-style-type: none"> Similar feature values across some classes. Possible class imbalance. Logistic Regression assumes linear decision boundaries, which may not fully capture complex relationships.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

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

# Scatter plot: Employee satisfaction vs Retention
plt.scatter(df.satisfaction_level, df.left, marker='+', color='red')
plt.xlabel("Satisfaction Level")
plt.ylabel("Left (1) / Stayed (0)")
plt.title("Impact of Satisfaction Level on Employee Retention")
plt.show()

# Define features (X) and target (y)
X = df[['satisfaction_level']]
y = df['left']

# Split dataset (90% train, 10% test)
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()
model.fit(X_train, y_train)

# Predictions
y_predicted = model.predict(X_test)

# Model Accuracy
print(f"Model Accuracy: {model.score(X_test, y_test):.4f}")

# Probability predictions
print("Predicted Probabilities:")
print(model.predict_proba(X_test))

# Predict for a specific satisfaction level (e.g., 0.4)
predicted_status = model.predict([[0.4]])
print(f"Prediction for Satisfaction Level 0.4: {'Left' if predicted_status[0] == 1 else 'Stayed'}")

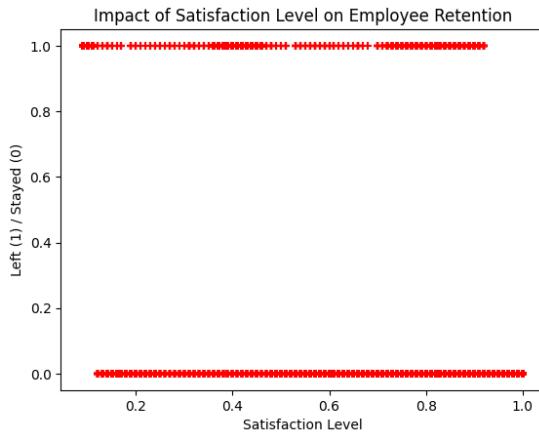
# Logistic function
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

```

# Custom prediction function
m, b = model.coef_[0][0], model.intercept_[0]
def prediction_function(satisfaction):
    z = m * satisfaction + b
    y = sigmoid(z)
    return y

satisfaction_test = 0.4
print(f"Sigmoid Prediction for Satisfaction Level {satisfaction_test}:
{prediction_function(satisfaction_test):.4f}")

```



Model Accuracy: 0.7707

Predicted Probabilities:

[[0.81879598 0.18120402]

[0.64435551 0.35564449]

[0.67008191 0.32991809]

...

[0.85026544 0.14973456]

[0.93858587 0.06141413]

[0.90306111 0.09693889]]

Prediction for Satisfaction Level 0.4: Stayed

Sigmoid Prediction for Satisfaction Level 0.4: 0.3644

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

```

Load the Zoo dataset

file_path = "/content/zoo-data (1).csv"

zoo_data = pd.read_csv(file_path)

Drop the 'animal_name' column as it is not a relevant feature

X = zoo_data.drop(['animal_name', 'class_type'], axis=1) # Features

```

y = zoo_data['class_type'] # Target variable

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

# Initialize the Logistic Regression model for multi-class classification
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=200)
# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Multinomial Logistic Regression model: {accuracy:.2f}")

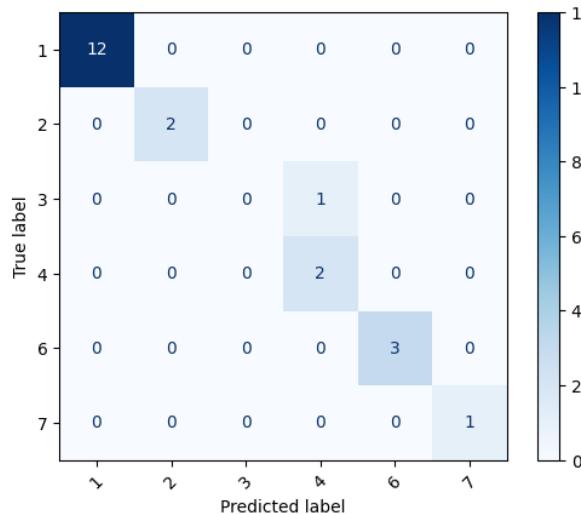
# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Adjust display labels to match actual present labels in the test set
unique_classes_in_test = sorted(y_test.unique())

# Display confusion matrix
cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=unique_classes_in_test)
cm_display.plot(cmap='Blues', xticks_rotation=45)
plt.show()

```

Accuracy of the Multinomial Logistic Regression model: 0.95



Program 5

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

Screenshot

PAGE NO: 6 DATE: 10/03/2023

```

Lab-2
Decision Tree : ID3 (Iterative Dichotomizer 3) Algorithm
import numpy as np
import pandas as pd
from collections import Counter

class Node:
    def __init__(self, feature = None, value = None, label = None):
        self.feature = feature
        self.value = value
        self.label = label
        self.children = {}

def entropy(y):
    counts = np.bincount(y)
    probabilities = counts / len(y)
    return -np.sum([p * np.log2(p) for p in probabilities if p > 0])

def information_gain(x, y, feature):
    total_entropy = entropy(y)
    values, counts = np.unique(x[:, feature], return_counts=True)
    weighted_entropy = sum([counts[i] / sum(counts) * entropy(y[x[:, feature] == v]) for i, v in enumerate(values)])
    return total_entropy - weighted_entropy

def best_feature_to_split(x, y):
    gains = [information_gain(x, y, i) for i in range(x.shape[1])]
    return np.argmax(gains)

```

PAGE NO: 7 DATE:

```

def id3(x, y, features):
    if len(set(y)) == 1:
        return Node(label = y[0])
    if len(features) == 0:
        return Node(label = Counter(y).most_common(1)[0][0])
    best_feature = best_feature_to_split(x, y)
    node = Node(feature = features[best_feature])
    feature_values = np.unique(x[:, best_feature])
    for value in feature_values:
        sub_x = x[x[:, best_feature] == value]
        sub_y = y[x[:, best_feature] == value]
        if len(sub_y) == 0:
            node.children[value] = Node(label = Counter(y).most_common(1)[0][0])
        else:
            node.children[value] = id3(sub_x, sub_y, features[best_feature + 1:])
    return node

def print_tree(node, depth=0):
    if node.label is not None:
        print(f"{' ' * depth}Leaf: {node.label}")
        return
    print(f"{' ' * depth}Feature: {node.feature}")
    for value, child in node.children.items():
        print(f"{' ' * depth}Value: {value} {child}")
        print_tree(child, depth + 1)

```

```

# Example dataset
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Sunny', 'Overcast', 'Rain', 'Sunny'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Mild', 'Mild', 'Mild', 'Mild', 'Cool', 'Cool', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Cool'],
    'Humidity': ['High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'Normal', 'Normal', 'High', 'High', 'High', 'High', 'Normal', 'Normal', 'High', 'High'],
    'Wind': ['Weak', 'Weak', 'Strong', 'Weak', 'Strong', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak', 'Strong', 'Weak', 'Strong', 'Weak', 'Weak'],
    'Play Tennis': ['No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes']
})

X = data.iloc[:, :-1].apply(lambda col: pd.factorize(col)[0]).to_numpy()
y = pd.factorize(data['Play Tennis'])[0]
features = list(data.columns[:-1])

decision_tree = id3(X, y, features)
print_tree(decision_tree)

```

Output:

```

Feature : Outlook
Value : 0
Feature : Humidity
Value : 0
Leaf : 0
Value : 1
Leaf : 1
Value : 2
Leaf : 2
Value : 3
Leaf : 3
Value : 4
Leaf : 4
Value : 5
Leaf : 5
Value : 6
Leaf : 6
Value : 7
Leaf : 7
Value : 8
Leaf : 8
Value : 9
Leaf : 9
Value : 10
Leaf : 10
Value : 11
Leaf : 11
Value : 12
Leaf : 12
Value : 13
Leaf : 13
Value : 14
Leaf : 14
Value : 15
Leaf : 15
Value : 16
Leaf : 16
Value : 17
Leaf : 17
Value : 18
Leaf : 18
Value : 19
Leaf : 19
Value : 20
Leaf : 20
Value : 21
Leaf : 21
Value : 22
Leaf : 22
Value : 23
Leaf : 23
Value : 24
Leaf : 24
Value : 25
Leaf : 25
Value : 26
Leaf : 26
Value : 27
Leaf : 27
Value : 28
Leaf : 28
Value : 29
Leaf : 29
Value : 30
Leaf : 30
Value : 31
Leaf : 31
Value : 32
Leaf : 32
Value : 33
Leaf : 33
Value : 34
Leaf : 34
Value : 35
Leaf : 35
Value : 36
Leaf : 36
Value : 37
Leaf : 37
Value : 38
Leaf : 38
Value : 39
Leaf : 39
Value : 40
Leaf : 40
Value : 41
Leaf : 41
Value : 42
Leaf : 42
Value : 43
Leaf : 43
Value : 44
Leaf : 44
Value : 45
Leaf : 45
Value : 46
Leaf : 46
Value : 47
Leaf : 47
Value : 48
Leaf : 48
Value : 49
Leaf : 49
Value : 50
Leaf : 50
Value : 51
Leaf : 51
Value : 52
Leaf : 52
Value : 53
Leaf : 53
Value : 54
Leaf : 54
Value : 55
Leaf : 55
Value : 56
Leaf : 56
Value : 57
Leaf : 57
Value : 58
Leaf : 58
Value : 59
Leaf : 59
Value : 60
Leaf : 60
Value : 61
Leaf : 61
Value : 62
Leaf : 62
Value : 63
Leaf : 63
Value : 64
Leaf : 64
Value : 65
Leaf : 65
Value : 66
Leaf : 66
Value : 67
Leaf : 67
Value : 68
Leaf : 68
Value : 69
Leaf : 69
Value : 70
Leaf : 70
Value : 71
Leaf : 71
Value : 72
Leaf : 72
Value : 73
Leaf : 73
Value : 74
Leaf : 74
Value : 75
Leaf : 75
Value : 76
Leaf : 76
Value : 77
Leaf : 77
Value : 78
Leaf : 78
Value : 79
Leaf : 79
Value : 80
Leaf : 80
Value : 81
Leaf : 81
Value : 82
Leaf : 82
Value : 83
Leaf : 83
Value : 84
Leaf : 84
Value : 85
Leaf : 85
Value : 86
Leaf : 86
Value : 87
Leaf : 87
Value : 88
Leaf : 88
Value : 89
Leaf : 89
Value : 90
Leaf : 90
Value : 91
Leaf : 91
Value : 92
Leaf : 92
Value : 93
Leaf : 93
Value : 94
Leaf : 94
Value : 95
Leaf : 95
Value : 96
Leaf : 96
Value : 97
Leaf : 97
Value : 98
Leaf : 98
Value : 99
Leaf : 99
Value : 100
Leaf : 100
Value : 101
Leaf : 101
Value : 102
Leaf : 102
Value : 103
Leaf : 103
Value : 104
Leaf : 104
Value : 105
Leaf : 105
Value : 106
Leaf : 106
Value : 107
Leaf : 107
Value : 108
Leaf : 108
Value : 109
Leaf : 109
Value : 110
Leaf : 110
Value : 111
Leaf : 111
Value : 112
Leaf : 112
Value : 113
Leaf : 113
Value : 114
Leaf : 114
Value : 115
Leaf : 115
Value : 116
Leaf : 116
Value : 117
Leaf : 117
Value : 118
Leaf : 118
Value : 119
Leaf : 119
Value : 120
Leaf : 120
Value : 121
Leaf : 121
Value : 122
Leaf : 122
Value : 123
Leaf : 123
Value : 124
Leaf : 124
Value : 125
Leaf : 125
Value : 126
Leaf : 126
Value : 127
Leaf : 127
Value : 128
Leaf : 128
Value : 129
Leaf : 129
Value : 130
Leaf : 130
Value : 131
Leaf : 131
Value : 132
Leaf : 132
Value : 133
Leaf : 133
Value : 134
Leaf : 134
Value : 135
Leaf : 135
Value : 136
Leaf : 136
Value : 137
Leaf : 137
Value : 138
Leaf : 138
Value : 139
Leaf : 139
Value : 140
Leaf : 140
Value : 141
Leaf : 141
Value : 142
Leaf : 142
Value : 143
Leaf : 143
Value : 144
Leaf : 144
Value : 145
Leaf : 145
Value : 146
Leaf : 146
Value : 147
Leaf : 147
Value : 148
Leaf : 148
Value : 149
Leaf : 149
Value : 150
Leaf : 150
Value : 151
Leaf : 151
Value : 152
Leaf : 152
Value : 153
Leaf : 153
Value : 154
Leaf : 154
Value : 155
Leaf : 155
Value : 156
Leaf : 156
Value : 157
Leaf : 157
Value : 158
Leaf : 158
Value : 159
Leaf : 159
Value : 160
Leaf : 160
Value : 161
Leaf : 161
Value : 162
Leaf : 162
Value : 163
Leaf : 163
Value : 164
Leaf : 164
Value : 165
Leaf : 165
Value : 166
Leaf : 166
Value : 167
Leaf : 167
Value : 168
Leaf : 168
Value : 169
Leaf : 169
Value : 170
Leaf : 170
Value : 171
Leaf : 171
Value : 172
Leaf : 172
Value : 173
Leaf : 173
Value : 174
Leaf : 174
Value : 175
Leaf : 175
Value : 176
Leaf : 176
Value : 177
Leaf : 177
Value : 178
Leaf : 178
Value : 179
Leaf : 179
Value : 180
Leaf : 180
Value : 181
Leaf : 181
Value : 182
Leaf : 182
Value : 183
Leaf : 183
Value : 184
Leaf : 184
Value : 185
Leaf : 185
Value : 186
Leaf : 186
Value : 187
Leaf : 187
Value : 188
Leaf : 188
Value : 189
Leaf : 189
Value : 190
Leaf : 190
Value : 191
Leaf : 191
Value : 192
Leaf : 192
Value : 193
Leaf : 193
Value : 194
Leaf : 194
Value : 195
Leaf : 195
Value : 196
Leaf : 196
Value : 197
Leaf : 197
Value : 198
Leaf : 198
Value : 199
Leaf : 199
Value : 200
Leaf : 200
Value : 201
Leaf : 201
Value : 202
Leaf : 202
Value : 203
Leaf : 203
Value : 204
Leaf : 204
Value : 205
Leaf : 205
Value : 206
Leaf : 206
Value : 207
Leaf : 207
Value : 208
Leaf : 208
Value : 209
Leaf : 209
Value : 210
Leaf : 210
Value : 211
Leaf : 211
Value : 212
Leaf : 212
Value : 213
Leaf : 213
Value : 214
Leaf : 214
Value : 215
Leaf : 215
Value : 216
Leaf : 216
Value : 217
Leaf : 217
Value : 218
Leaf : 218
Value : 219
Leaf : 219
Value : 220
Leaf : 220
Value : 221
Leaf : 221
Value : 222
Leaf : 222
Value : 223
Leaf : 223
Value : 224
Leaf : 224
Value : 225
Leaf : 225
Value : 226
Leaf : 226
Value : 227
Leaf : 227
Value : 228
Leaf : 228
Value : 229
Leaf : 229
Value : 230
Leaf : 230
Value : 231
Leaf : 231
Value : 232
Leaf : 232
Value : 233
Leaf : 233
Value : 234
Leaf : 234
Value : 235
Leaf : 235
Value : 236
Leaf : 236
Value : 237
Leaf : 237
Value : 238
Leaf : 238
Value : 239
Leaf : 239
Value : 240
Leaf : 240
Value : 241
Leaf : 241
Value : 242
Leaf : 242
Value : 243
Leaf : 243
Value : 244
Leaf : 244
Value : 245
Leaf : 245
Value : 246
Leaf : 246
Value : 247
Leaf : 247
Value : 248
Leaf : 248
Value : 249
Leaf : 249
Value : 250
Leaf : 250
Value : 251
Leaf : 251
Value : 252
Leaf : 252
Value : 253
Leaf : 253
Value : 254
Leaf : 254
Value : 255
Leaf : 255
Value : 256
Leaf : 256
Value : 257
Leaf : 257
Value : 258
Leaf : 258
Value : 259
Leaf : 259
Value : 260
Leaf : 260
Value : 261
Leaf : 261
Value : 262
Leaf : 262
Value : 263
Leaf : 263
Value : 264
Leaf : 264
Value : 265
Leaf : 265
Value : 266
Leaf : 266
Value : 267
Leaf : 267
Value : 268
Leaf : 268
Value : 269
Leaf : 269
Value : 270
Leaf : 270
Value : 271
Leaf : 271
Value : 272
Leaf : 272
Value : 273
Leaf : 273
Value : 274
Leaf : 274
Value : 275
Leaf : 275
Value : 276
Leaf : 276
Value : 277
Leaf : 277
Value : 278
Leaf : 278
Value : 279
Leaf : 279
Value : 280
Leaf : 280
Value : 281
Leaf : 281
Value : 282
Leaf : 282
Value : 283
Leaf : 283
Value : 284
Leaf : 284
Value : 285
Leaf : 285
Value : 286
Leaf : 286
Value : 287
Leaf : 287
Value : 288
Leaf : 288
Value : 289
Leaf : 289
Value : 290
Leaf : 290
Value : 291
Leaf : 291
Value : 292
Leaf : 292
Value : 293
Leaf : 293
Value : 294
Leaf : 294
Value : 295
Leaf : 295
Value : 296
Leaf : 296
Value : 297
Leaf : 297
Value : 298
Leaf : 298
Value : 299
Leaf : 299
Value : 300
Leaf : 300
Value : 301
Leaf : 301
Value : 302
Leaf : 302
Value : 303
Leaf : 303
Value : 304
Leaf : 304
Value : 305
Leaf : 305
Value : 306
Leaf : 306
Value : 307
Leaf : 307
Value : 308
Leaf : 308
Value : 309
Leaf : 309
Value : 310
Leaf : 310
Value : 311
Leaf : 311
Value : 312
Leaf : 312
Value : 313
Leaf : 313
Value : 314
Leaf : 314
Value : 315
Leaf : 315
Value : 316
Leaf : 316
Value : 317
Leaf : 317
Value : 318
Leaf : 318
Value : 319
Leaf : 319
Value : 320
Leaf : 320
Value : 321
Leaf : 321
Value : 322
Leaf : 322
Value : 323
Leaf : 323
Value : 324
Leaf : 324
Value : 325
Leaf : 325
Value : 326
Leaf : 326
Value : 327
Leaf : 327
Value : 328
Leaf : 328
Value : 329
Leaf : 329
Value : 330
Leaf : 330
Value : 331
Leaf : 331
Value : 332
Leaf : 332
Value : 333
Leaf : 333
Value : 334
Leaf : 334
Value : 335
Leaf : 335
Value : 336
Leaf : 336
Value : 337
Leaf : 337
Value : 338
Leaf : 338
Value : 339
Leaf : 339
Value : 340
Leaf : 340
Value : 341
Leaf : 341
Value : 342
Leaf : 342
Value : 343
Leaf : 343
Value : 344
Leaf : 344
Value : 345
Leaf : 345
Value : 346
Leaf : 346
Value : 347
Leaf : 347
Value : 348
Leaf : 348
Value : 349
Leaf : 349
Value : 350
Leaf : 350
Value : 351
Leaf : 351
Value : 352
Leaf : 352
Value : 353
Leaf : 353
Value : 354
Leaf : 354
Value : 355
Leaf : 355
Value : 356
Leaf : 356
Value : 357
Leaf : 357
Value : 358
Leaf : 358
Value : 359
Leaf : 359
Value : 360
Leaf : 360
Value : 361
Leaf : 361
Value : 362
Leaf : 362
Value : 363
Leaf : 363
Value : 364
Leaf : 364
Value : 365
Leaf : 365
Value : 366
Leaf : 366
Value : 367
Leaf : 367
Value : 368
Leaf : 368
Value : 369
Leaf : 369
Value : 370
Leaf : 370
Value : 371
Leaf : 371
Value : 372
Leaf : 372
Value : 373
Leaf : 373
Value : 374
Leaf : 374
Value : 375
Leaf : 375
Value : 376
Leaf : 376
Value : 377
Leaf : 377
Value : 378
Leaf : 378
Value : 379
Leaf : 379
Value : 380
Leaf : 380
Value : 381
Leaf : 381
Value : 382
Leaf : 382
Value : 383
Leaf : 383
Value : 384
Leaf : 384
Value : 385
Leaf : 385
Value : 386
Leaf : 386
Value : 387
Leaf : 387
Value : 388
Leaf : 388
Value : 389
Leaf : 389
Value : 390
Leaf : 390
Value : 391
Leaf : 391
Value : 392
Leaf : 392
Value : 393
Leaf : 393
Value : 394
Leaf : 394
Value : 395
Leaf : 395
Value : 396
Leaf : 396
Value : 397
Leaf : 397
Value : 398
Leaf : 398
Value : 399
Leaf : 399
Value : 400
Leaf : 400
Value : 401
Leaf : 401
Value : 402
Leaf : 402
Value : 403
Leaf : 403
Value : 404
Leaf : 404
Value : 405
Leaf : 405
Value : 406
Leaf : 406
Value : 407
Leaf : 407
Value : 408
Leaf : 408
Value : 409
Leaf : 409
Value : 410
Leaf : 410
Value : 411
Leaf : 411
Value : 412
Leaf : 412
Value : 413
Leaf : 413
Value : 414
Leaf : 414
Value : 415
Leaf : 415
Value : 416
Leaf : 416
Value : 417
Leaf : 417
Value : 418
Leaf : 418
Value : 419
Leaf : 419
Value : 420
Leaf : 420
Value : 421
Leaf : 421
Value : 422
Leaf : 422
Value : 423
Leaf : 423
Value : 424
Leaf : 424
Value : 425
Leaf : 425
Value : 426
Leaf : 426
Value : 427
Leaf : 427
Value : 428
Leaf : 428
Value : 429
Leaf : 429
Value : 430
Leaf : 430
Value : 431
Leaf : 431
Value : 432
Leaf : 432
Value : 433
Leaf : 433
Value : 434
Leaf : 434
Value : 435
Leaf : 435
Value : 436
Leaf : 436
Value : 437
Leaf : 437
Value : 438
Leaf : 438
Value : 439
Leaf : 439
Value : 440
Leaf : 440
Value : 441
Leaf : 441
Value : 442
Leaf : 442
Value : 443
Leaf : 443
Value : 444
Leaf : 444
Value : 445
Leaf : 445
Value : 446
Leaf : 446
Value : 447
Leaf : 447
Value : 448
Leaf : 448
Value : 449
Leaf : 449
Value : 450
Leaf : 450
Value : 451
Leaf : 451
Value : 452
Leaf : 452
Value : 453
Leaf : 453
Value : 454
Leaf : 454
Value : 455
Leaf : 455
Value : 456
Leaf : 456
Value : 457
Leaf : 457
Value : 458
Leaf : 458
Value : 459
Leaf : 459
Value : 460
Leaf : 460
Value : 461
Leaf : 461
Value : 462
Leaf : 462
Value : 463
Leaf : 463
Value : 464
Leaf : 464
Value : 465
Leaf : 465
Value : 466
Leaf : 466
Value : 467
Leaf : 467
Value : 468
Leaf : 468
Value : 469
Leaf : 469
Value : 470
Leaf : 470
Value : 471
Leaf : 471
Value : 472
Leaf : 472
Value : 473
Leaf : 473
Value : 474
Leaf : 474
Value : 475
Leaf : 475
Value : 476
Leaf : 476
Value : 477
Leaf : 477
Value : 478
Leaf : 478
Value : 479
Leaf : 479
Value : 480
Leaf : 480
Value : 481
Leaf : 481
Value : 482
Leaf : 482
Value : 483
Leaf : 483
Value : 484
Leaf : 484
Value : 485
Leaf : 485
Value : 486
Leaf : 486
Value : 487
Leaf : 487
Value : 488
Leaf : 488
Value : 489
Leaf : 489
Value : 490
Leaf : 490
Value : 491
Leaf : 491
Value : 492
Leaf : 492
Value : 493
Leaf : 493
Value : 494
Leaf : 494
Value : 495
Leaf : 495
Value : 496
Leaf : 496
Value : 497
Leaf : 497
Value : 498
Leaf : 498
Value : 499
Leaf : 499
Value : 500
Leaf : 500
Value : 501
Leaf : 501
Value : 502
Leaf : 502
Value : 503
Leaf : 503
Value : 504
Leaf : 504
Value : 505
Leaf : 505
Value : 506
Leaf : 506
Value : 507
Leaf : 507
Value : 508
Leaf : 508
Value : 509
Leaf : 509
Value : 510
Leaf : 510
Value : 511
Leaf : 511
Value : 512
Leaf : 512
Value : 513
Leaf : 513
Value : 514
Leaf : 514
Value : 515
Leaf : 515
Value : 516
Leaf : 516
Value : 517
Leaf : 517
Value : 518
Leaf : 518
Value : 519
Leaf : 519
Value : 520
Leaf : 520
Value : 521
Leaf : 521
Value : 522
Leaf : 522
Value : 523
Leaf : 523
Value : 524
Leaf : 524
Value : 525
Leaf : 525
Value : 526
Leaf : 526
Value : 527
Leaf : 527
Value : 528
Leaf : 528
Value : 529
Leaf : 529
Value : 530
Leaf : 530
Value : 531
Leaf : 531
Value : 532
Leaf : 532
Value : 533
Leaf : 533
Value : 534
Leaf : 534
Value : 535
Leaf : 535
Value : 536
Leaf : 536
Value : 537
Leaf : 537
Value : 538
Leaf : 538
Value : 539
Leaf : 539
Value : 540
Leaf : 540
Value : 541
Leaf : 541
Value : 542
Leaf : 542
Value : 543
Leaf : 543
Value : 544
Leaf : 544
Value : 545
Leaf : 545
Value : 546
Leaf : 546
Value : 547
Leaf : 547
Value : 548
Leaf : 548
Value : 549
Leaf : 549
Value : 550
Leaf : 550
Value : 551
Leaf : 551
Value : 552
Leaf : 552
Value : 553
Leaf : 553
Value : 554
Leaf : 554
Value : 555
Leaf : 555
Value : 556
Leaf : 556
Value : 557
Leaf : 557
Value : 558
Leaf : 558
Value : 559
Leaf : 559
Value : 560
Leaf : 560
Value : 561
Leaf : 561
Value : 562
Leaf : 562
Value : 563
Leaf : 563
Value : 564
Leaf : 564
Value : 565
Leaf : 565
Value : 566
Leaf : 566
Value : 567
Leaf : 567
Value : 568
Leaf : 568
Value : 569
Leaf : 569
Value : 570
Leaf : 570
Value : 571
Leaf : 571
Value : 572
Leaf : 572
Value : 573
Leaf : 573
Value : 574
Leaf : 574
Value : 575
Leaf : 575
Value : 576
Leaf : 576
Value : 577
Leaf : 577
Value : 578
Leaf : 578
Value : 579
Leaf : 579
Value : 580
Leaf : 580
Value : 581
Leaf : 581
Value : 582
Leaf : 582
Value : 583
Leaf : 583
Value : 584
Leaf : 584
Value : 585
Leaf : 585
Value : 586
Leaf : 586
Value : 587
Leaf : 587
Value : 588
Leaf : 588
Value : 589
Leaf : 589
Value : 590
Leaf : 590
Value : 591
Leaf : 591
Value : 592
Leaf : 592
Value : 593
Leaf : 593
Value : 594
Leaf : 594
Value : 595
Leaf : 595
Value : 596
Leaf : 596
Value : 597
Leaf : 597
Value : 598
Leaf : 598
Value : 599
Leaf : 599
Value : 600
Leaf : 600
Value : 601
Leaf : 601
Value : 602
Leaf : 602
Value : 603
Leaf : 603
Value : 604
Leaf : 604
Value : 605
Leaf : 605
Value : 606
Leaf : 606
Value : 607
Leaf : 607
Value : 608
Leaf : 608
Value : 609
Leaf : 609
Value : 610
Leaf : 610
Value : 611
Leaf : 611
Value : 612
Leaf : 612
Value : 613
Leaf : 613
Value : 614
Leaf : 614
Value : 615
Leaf : 615
Value : 616
Leaf : 616
Value : 617
Leaf : 617
Value : 618
Leaf : 618
Value : 619
Leaf : 619
Value : 620
Leaf : 620
Value : 621
Leaf : 621
Value : 622
Leaf : 622
Value : 623
Leaf : 623
Value : 624
Leaf : 624
Value : 625
Leaf : 625
Value : 626
Leaf : 626
Value : 627
Leaf : 627
Value : 628
Leaf : 628
Value : 629
Leaf : 629
Value : 630
Leaf : 630
Value : 631
Leaf : 631
Value : 632
Leaf : 632
Value : 633
Leaf : 633
Value : 634
Leaf : 634
Value : 635
Leaf : 635
Value : 636
Leaf : 636
Value : 637
Leaf : 637
Value : 638
Leaf : 638
Value : 639
Leaf : 639
Value : 640
Leaf : 640
Value : 641
Leaf : 641
Value : 642
Leaf : 642
Value : 643
Leaf : 643
Value : 644
Leaf : 644
Value : 645
Leaf : 645
Value : 646
Leaf : 646
Value : 647
Leaf : 647
Value : 648
Leaf : 648
Value : 649
Leaf : 649
Value : 650
Leaf : 650
Value : 651
Leaf : 651
Value : 652
Leaf : 652
Value : 653
Leaf : 653
Value : 654
Leaf : 654
Value : 655
Leaf : 655
Value : 656
Leaf : 656
Value : 657
Leaf : 657
Value : 658
Leaf : 658
Value : 659
Leaf : 659
Value : 660
Leaf : 660
Value : 661
Leaf : 661
Value : 662
Leaf : 662
Value : 663
Leaf : 663
Value : 664
Leaf : 664
Value : 665
Leaf : 665
Value : 666
Leaf : 666
Value : 667
Leaf : 667
Value : 668
Leaf : 668
Value : 669
Leaf : 669
Value : 670
Leaf : 670
Value : 671
Leaf : 671
Value : 672
Leaf : 672
Value : 673
Leaf : 673
Value : 674
Leaf : 674
Value : 675
Leaf : 675
Value : 676
Leaf : 676
Value : 677
Leaf : 677
Value : 678
Leaf : 678
Value : 679
Leaf : 679
Value : 680
Leaf : 680
Value : 681
Leaf : 681
Value : 682
Leaf : 682
Value : 683
Leaf : 683
Value : 684
Leaf : 684
Value : 685
Leaf : 685
Value : 686
Leaf : 686
Value : 687
Leaf : 687
Value : 688
Leaf : 688
Value : 689
Leaf : 689
Value : 690
Leaf : 690
Value : 691
Leaf : 691
Value : 692
Leaf : 692
Value : 693
Leaf : 693
Value : 694
Leaf : 694
Value : 695
Leaf : 695
Value : 696
Leaf : 696
Value : 697
Leaf : 697
Value : 698
Leaf : 698
Value : 699
Leaf : 699
Value : 700
Leaf : 700
Value : 701
Leaf : 701
Value : 702
Leaf : 702
Value : 703
Leaf : 703
Value : 704
Leaf : 704
Value : 705
Leaf : 705
Value : 706
Leaf : 706
Value : 707
Leaf : 707
Value : 708
Leaf : 708
Value : 709
Leaf : 709
Value : 710
Leaf : 710
Value : 711
Leaf : 711
Value : 712
Leaf : 712
Value : 713
Leaf : 713
Value : 714
Leaf : 714
Value : 715
Leaf : 715
Value : 716
Leaf : 716
Value : 717
Leaf : 717
Value : 718
Leaf : 718
Value : 719
Leaf : 719
Value : 720
Leaf : 720
Value : 721
Leaf : 721
Value : 722
Leaf : 722
Value : 723
Leaf : 723
Value : 724
Leaf : 724
Value : 725
Leaf : 725
Value : 726
Leaf : 726
Value : 727
Leaf : 727
Value : 728
Leaf : 728
Value : 729
Leaf : 729
Value : 730
Leaf : 730
Value : 731
Leaf : 731
Value : 732
Leaf : 732
Value : 733
Leaf : 733
Value : 734
Leaf : 734
Value : 735
Leaf : 735
Value : 736
Leaf : 736
Value : 737
Leaf : 737
Value : 738
Leaf : 738
Value : 739
Leaf : 739
Value : 740
Leaf : 740
Value : 741
Leaf : 741
Value : 742
Leaf : 742
Value : 743
Leaf : 743
Value : 744
Leaf : 744

```

Code:

```
import numpy as np
import pandas as pd
from collections import Counter
class Node:
    def __init__(self, feature=None, value=None, label=None):
        self.feature = feature # Attribute to split on
        self.value = value # Value of the attribute
        self.label = label # Label if it's a leaf node
        self.children = {} # Dictionary of child nodes

    def entropy(y):
        counts = np.bincount(y)
        probabilities = counts / len(y)
        return -np.sum([p * np.log2(p) for p in probabilities if p > 0])

    def information_gain(X, y, feature):
        total_entropy = entropy(y)
        values, counts = np.unique(X[:, feature], return_counts=True)
        weighted_entropy = sum((counts[i] / sum(counts)) * entropy(y[X[:, feature] == v]) for i, v in enumerate(values))
        return total_entropy - weighted_entropy

    def best_feature_to_split(X, y):
        gains = [information_gain(X, y, i) for i in range(X.shape[1])]
        return np.argmax(gains)

    def id3(X, y, features):
        if len(set(y)) == 1:
            return Node(label=y[0])
        if len(features) == 0:
            return Node(label=Counter(y).most_common(1)[0][0])
        best_feature = best_feature_to_split(X, y)
        node = Node(feature=features[best_feature])
        feature_values = np.unique(X[:, best_feature])
        for value in feature_values:
            sub_X = X[X[:, best_feature] == value]
            sub_y = y[X[:, best_feature] == value]
            if len(sub_y) == 0:
                node.children[value] = Node(label=Counter(y).most_common(1)[0][0])
            else:
                node.children[value] = id3(np.delete(sub_X, best_feature, axis=1), sub_y, features[:best_feature] + features[best_feature+1:])
        return node
        if node.label is not None:
            print(f"{' ' * depth}Leaf: {node.label}")
        return
    print(f"{' ' * depth}Feature: {node.feature}")
```

```

for value, child in node.children.items():
    print(f"{' ' * depth}Value: {value}")
    print_tree(child, depth + 1)
# Example dataset
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny',
    'Overcast', 'Overcast', 'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal',
    'High', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',
    'Weak', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
})
X = data.iloc[:, :-1].apply(lambda col: pd.factorize(col)[0]).to_numpy()
y = pd.factorize(data['PlayTennis'])[0]
features = list(data.columns[:-1])
decision_tree = id3(X, y, features)
print_tree(decision_tree)

```

Feature: Outlook

Value: 0

 Feature: Humidity

 Value: 0

 Leaf: 0

 Value: 1

 Leaf: 1

 Value: 1

 Leaf: 1

 Value: 2

 Feature: Wind

 Value: 0

 Leaf: 1

 Value: 1

 Leaf: 0

Program 6

Build KNN Classification model for a given dataset

Screenshot

Lab 5

PAGE NO: 15

Build KNN classification model for a given dataset

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

Person	Age	Salary	k	Target	Distance	Rank
A	18	50	N	52.83	5	
B	23	55	N	46.58	4	
C	24	70	N	31.96	2	
D	41	60	Y	40.44	3	
E	43	70	Y	31.06	1	
F	38	40	Y	60.08	6	
X	35	100				

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

A = $\sqrt{(35 - 18)^2 + (100 - 50)^2} = 52.83$
B = $\sqrt{(35 - 23)^2 + (100 - 55)^2} = 46.58$
C = $\sqrt{(35 - 24)^2 + (100 - 70)^2} = 31.96$
D = $\sqrt{(35 - 41)^2 + (100 - 60)^2} = 40.44$
E = $\sqrt{(35 - 43)^2 + (100 - 70)^2} = 31.06$
F = $\sqrt{(35 - 38)^2 + (100 - 40)^2} = 60.08$

K=3 \rightarrow X=Yes

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

To determine the best K value, Low K (eg: 1-3): High variance, more overfitting. High K (eg: 10+): More bias, risk of underfitting. Best K: Found by plotting accuracy v & k and error rates.

PAGE NO: 16

v & k, selecting the k with highest accuracy and lowest error.

For diabetes dataset: What is the purpose of feature scaling? How to perform it?

→ Features -

- Equal influence of all features
- Faster convergence in training
- Better performance and accuracy

To perform scaling - Use StandardScaler() in scikit-learn which transform data into a standard normal distribution.

Code:

```
import pandas as pd
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

# Function to train and evaluate KNN model
def knn_classification(data_path, target_column, dataset_name, k=5):
    # Load dataset
    df = pd.read_csv(data_path)
```

```

# Split features and target
X = df.drop(columns=[target_column])
y = df[target_column]

# Split 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)

# Feature scaling for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train KNN model
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of KNN on {dataset_name} dataset: {accuracy:.4f}')
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - {dataset_name}')
plt.show()

# Run KNN classification on both datasets
knn_classification('/content/iris (3).csv', 'species', 'Iris', k=5)
knn_classification('/content/diabetes.csv', 'Outcome', 'Diabetes', k=5)

```

Accuracy of KNN on Iris dataset: 1.0000

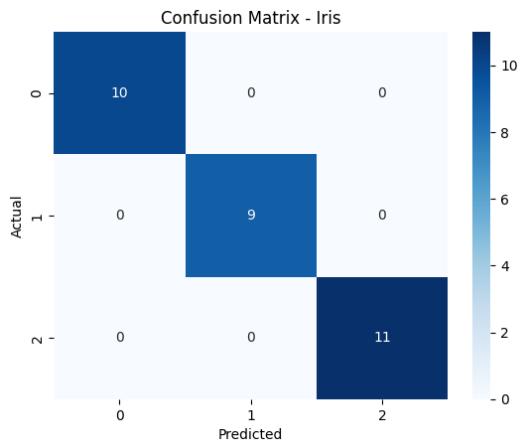
Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11

accuracy		1.00	30	
macro avg	1.00	1.00	1.00	30

weighted avg 1.00 1.00 1.00 30



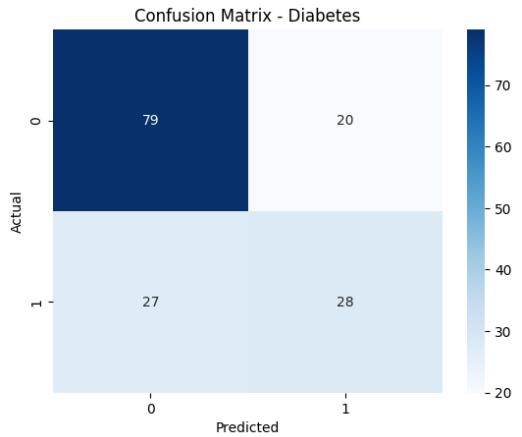
Accuracy of KNN on Diabetes dataset: 0.6948

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.75	0.80	0.77	99
1	0.58	0.51	0.54	55

	accuracy		0.69	154
macro avg	0.66	0.65	0.66	154
weighted avg	0.69	0.69	0.69	154



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

# Define features and target
X = df.drop(columns=['target']) # Assuming 'target' is the classification column
y = df['target']

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

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Find the best K value
k_values = range(1, 21)
accuracy_scores = []
for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))

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

# Train model with best K
best_model = KNeighborsClassifier(n_neighbors=best_k)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy with best K ({best_k}): {accuracy:.4f}')
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - KNN (K={best_k})')
plt.show()

# Plot K values vs. Accuracy
plt.plot(k_values, accuracy_scores, marker='o')

```

```

plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.title('K Value vs Accuracy')
plt.show()

```

Best K value: 7

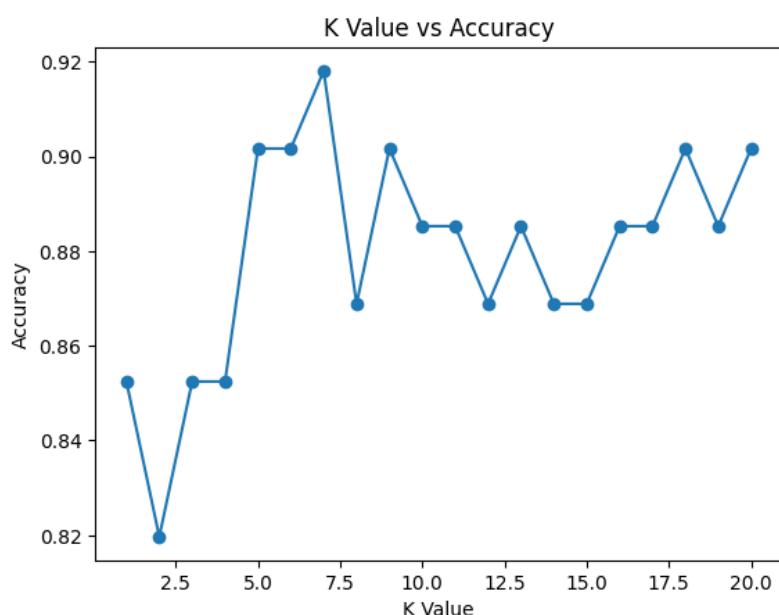
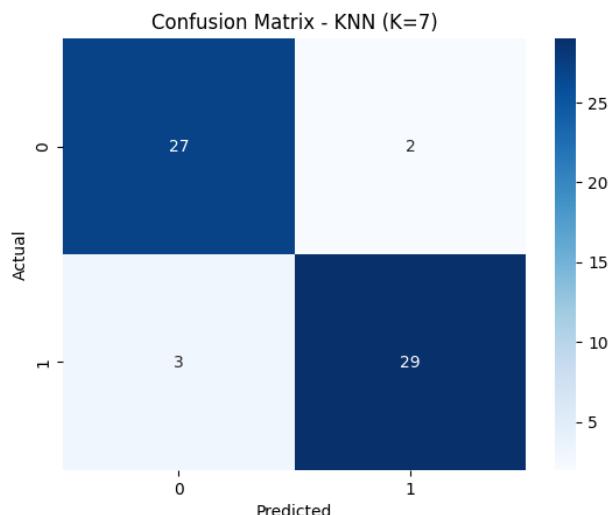
Accuracy with best K (7): 0.9180

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.90	0.93	0.92	29
1	0.94	0.91	0.92	32

accuracy		0.92	61	
macro avg	0.92	0.92	0.92	61
weighted avg	0.92	0.92	0.92	61



Program 7

Build Support vector machine model for a given dataset

Code:

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

# Define the Linear SVM class
class LinearSVM:
    def __init__(self, learning_rate=0.001, reg_strength=0.1, num_iterations=1000):
        self.learning_rate = learning_rate
        self.reg_strength = reg_strength
        self.num_iterations = num_iterations

    def fit(self, X, y):
        # Initialize weights and bias
        num_samples, num_features = X.shape
        self.W = np.zeros(num_features) # Weights
        self.b = 0 # Bias

        # Gradient Descent
        for _ in range(self.num_iterations):
            # Compute the margin (decision function)
            margins = 1 - y * (np.dot(X, self.W) + self.b)
            # Compute gradient
            dw = -2 * np.dot(X.T, (y * (margins > 0))) / num_samples + 2 * self.reg_strength * self.W
            db = -2 * np.sum(y * (margins > 0)) / num_samples

            # Update weights and bias
            self.W -= self.learning_rate * dw
            self.b -= self.learning_rate * db

    def predict(self, X):
        # Make predictions
        return np.sign(np.dot(X, self.W) + self.b)

# Generate toy data (binary classification)
np.random.seed(42)
num_samples = 100
X = np.random.randn(num_samples, 2)
y = np.ones(num_samples)
y[X[:, 0] < X[:, 1]] = -1 # Assign different class based on condition

# Train the Linear SVM
svm = LinearSVM(learning_rate=0.001, reg_strength=0.1, num_iterations=1000)
svm.fit(X, y)
```

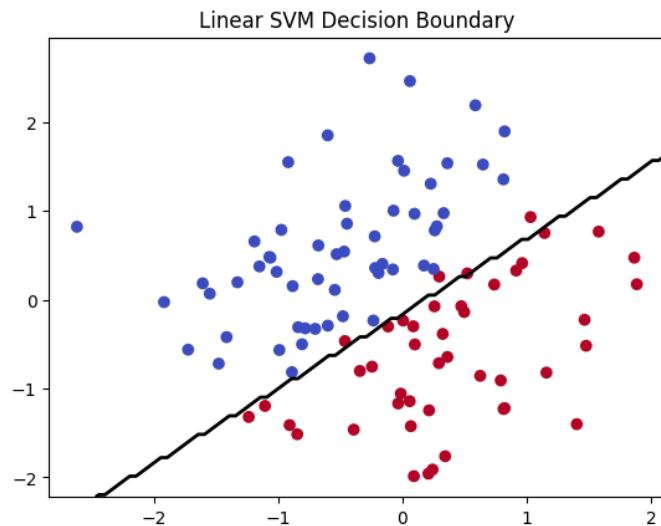
```

# Predict
y_pred = svm.predict(X)

# Visualize the decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100), np.linspace(ylim[0], ylim[1], 100))
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
plt.title("Linear SVM Decision Boundary")
plt.show()

# Print accuracy (simple comparison)
accuracy = np.mean(y_pred == y)
print(f"Accuracy: {accuracy * 100:.2f}%")

```

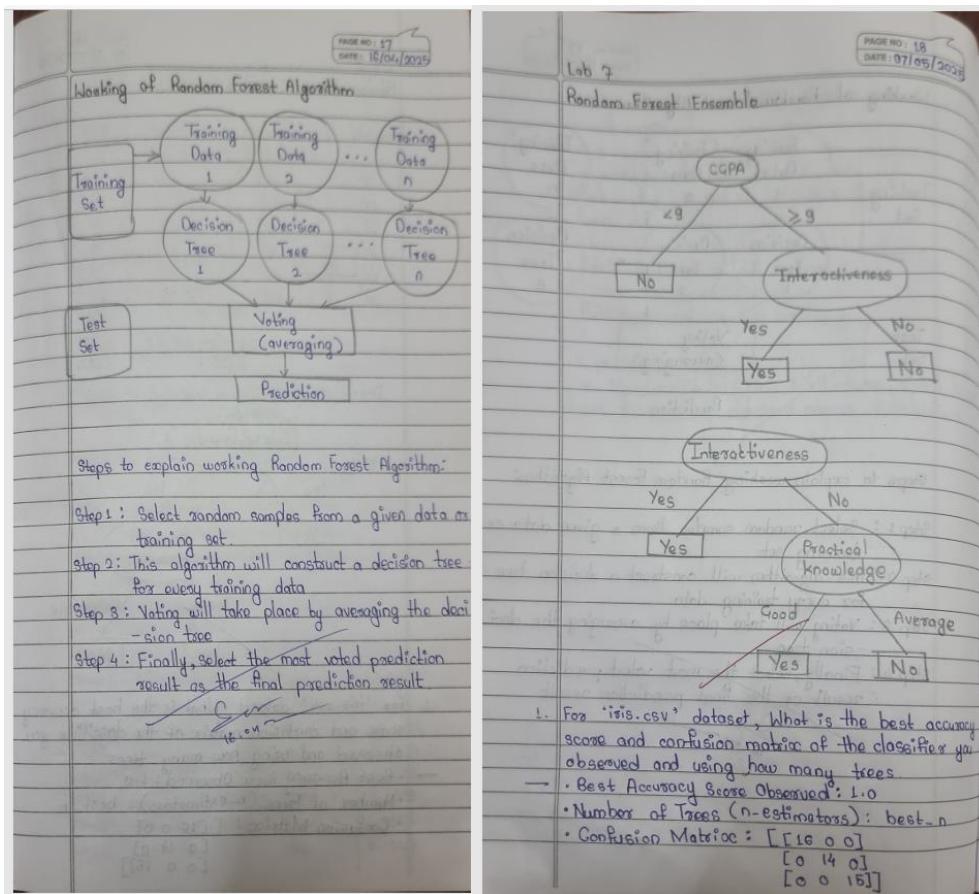


Accuracy: 96.00%

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot



Code:

```
import pandas as pd
import numpy as np
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 iris dataset from CSV
df = pd.read_csv("/content/iris (2).csv")
```

```
# Assuming last column is the label
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

```
# Split into training and test sets (70% train, 30% test)
```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# 1. Train RF Classifier with default n_estimators=10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
accuracy_default = accuracy_score(y_test, y_pred_default)

print(f"Default RF Accuracy (n_estimators=10): {accuracy_default:.4f}")

best_accuracy = 0
best_n = 0
accuracies = []

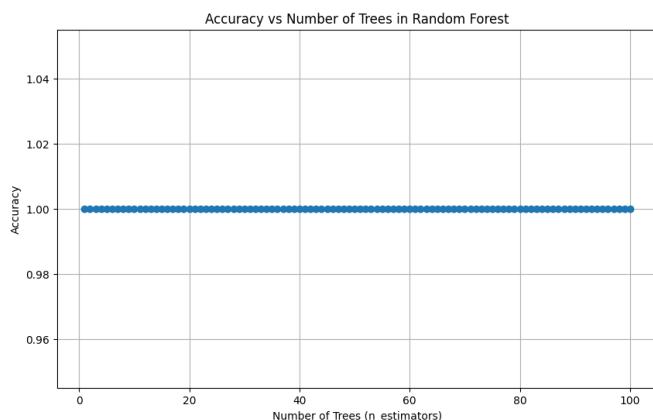
for n in range(1, 101):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    if acc > best_accuracy:
        best_accuracy = acc
        best_n = n

print(f"Best RF Accuracy: {best_accuracy:.4f} with n_estimators = {best_n}")
# Plot accuracy vs. number of trees
plt.figure(figsize=(10, 6))
plt.plot(range(1, 101), accuracies, marker='o')
plt.title("Accuracy vs Number of Trees in Random Forest")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()

```

Default RF Accuracy (n_estimators=10): 1.0000
 Best RF Accuracy: 1.0000 with n_estimators = 1



Program 9

Implement Boosting ensemble method on a given dataset

Screenshot

Lab 8

PAGE NO: 19

DATE:

AdaBoost Algorithm

CGPA	Predicted	Actual	Weight
Job Offer	Job Offer	Job Offer	
>=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

$\hat{e}_{CGPA} = 2 * \frac{1}{6} = 0.333$

$\alpha_{CGPA} = \frac{1}{2} \times \ln \left(\frac{1 - 0.333}{0.333} \right) = 0.347$

$\hat{z}_{CGPA} = \frac{1}{6} * 4 * e^{-0.347} + \frac{1}{6} * 2 * e^{0.347} = 0.9428$

$Wt(d_j)_{i+1} = \frac{\frac{1}{6} * e^{-0.347}}{0.9428} = 0.1249$

$Wt(d_j)_{i+1} = \frac{\frac{1}{6} * e^{0.347}}{0.9428} = 0.2501$

CGPA	Predicted	Actual	Weight
Job Offer	Job Offer	Job Offer	
>=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

PAGE NO: 20

DATE:

1. For 'income.csv' dataset, What is the best accuracy score and confusion matrix of the classifier you observed and using how many trees?

→ Accuracy with 10 estimators: 0.8277

Confusion Matrix (10 estimators):

$\begin{bmatrix} 10722 & 387 \\ 2138 & 1406 \end{bmatrix}$

Best Accuracy: 0.881 with n-estimators=42

Code:

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

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

```
# Step 2: Split into features and target
X = df.drop(columns=['income_level'])
y = df['income_level']
```

```
# Step 3: Train-test split
```

```

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

# Step 4: AdaBoost with 10 estimators
model_10 = AdaBoostClassifier(n_estimators=10, random_state=42)
model_10.fit(X_train, y_train)
y_pred_10 = model_10.predict(X_test)
accuracy_10 = accuracy_score(y_test, y_pred_10)
conf_matrix_10 = confusion_matrix(y_test, y_pred_10)

print("Accuracy with 10 estimators:", round(accuracy_10, 4))
print("Confusion Matrix (10 estimators):\n", conf_matrix_10)

# Step 5: Fine-tune number of trees (1 to 50)
best_accuracy = 0
best_n = 0
accuracies = []

for n in range(1, 51):
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    if acc > best_accuracy:
        best_accuracy = acc
        best_n = n

print(f"\nBest Accuracy: {round(best_accuracy, 4)} with n_estimators = {best_n}")

# Step 6: Plot accuracy vs. number of estimators
plt.figure(figsize=(10, 6))
plt.plot(range(1, 51), accuracies, marker='o', linestyle='-', color='blue')
plt.title('Accuracy vs Number of Trees (n_estimators)')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.grid(True)
plt.tight_layout()
plt.show()

```

Accuracy with 10 estimators: 0.8277

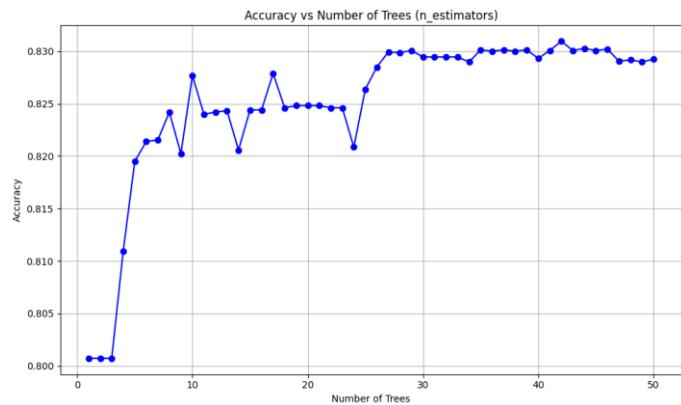
Confusion Matrix (10 estimators):

```

[[10722  387]
 [ 2138 1406]]

```

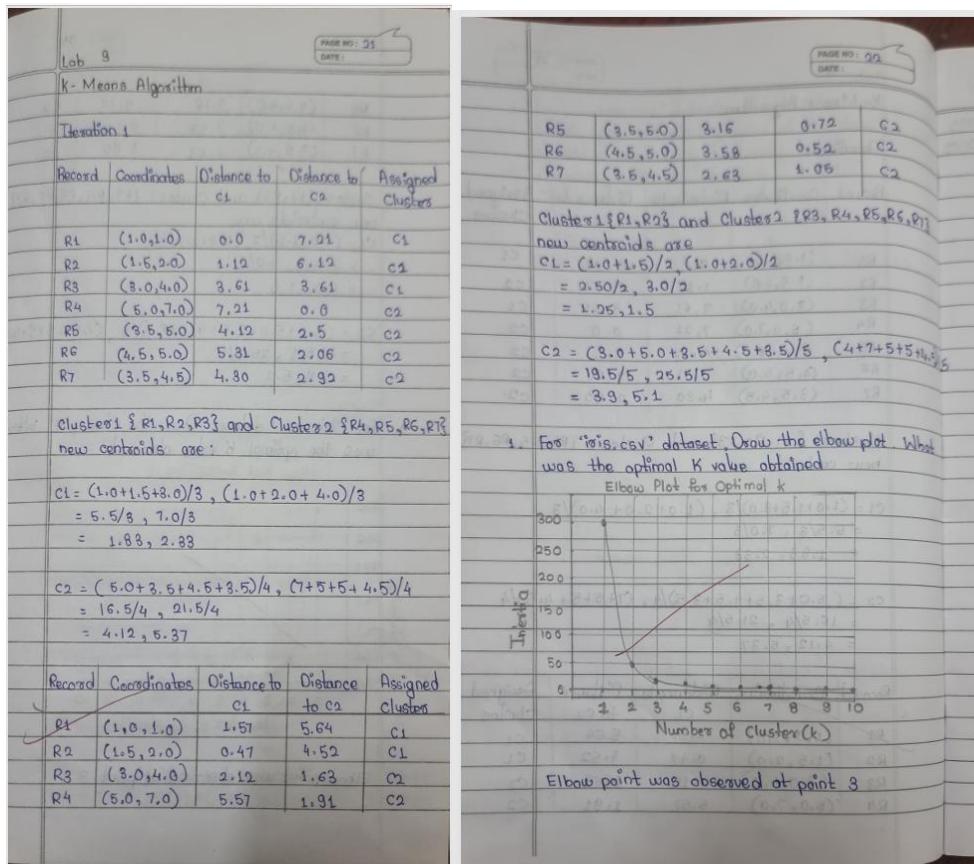
Best Accuracy: 0.831 with n_estimators = 42



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
```

```
# Load the dataset
df = pd.read_csv("/content/iris (2).csv")
```

```
# Select only petal length and petal width
X = df[['petal_length', 'petal_width']]
```

```
# Optional: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

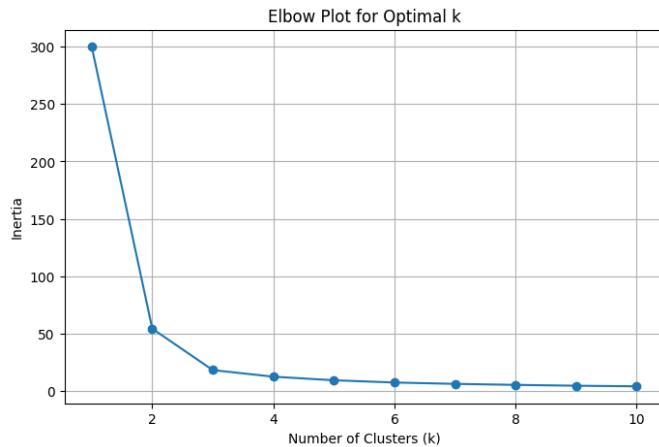
```

# Elbow method to determine optimal k
inertia = []
k_range = range(1, 11)

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

# Plot the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Plot for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()

```



Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot

Page 23 (Left):

Lab 10
Principle Component Analysis
PAGE NO: 23
DATE:

Feature Eg_1 Eg_2 Eg_3 Eg_4

x_1	4	8	13	7
x_2	11	4	5	14

Eigen Values
 $\lambda_1 = 30.8849$
 $\lambda_2 = 6.6151$

Eigen Vectors
 $e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$
 $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

$\rightarrow \text{matrix} C = \begin{bmatrix} 4 & 8 & 13 & 7 \\ 11 & 4 & 5 & 14 \end{bmatrix}$

mean centre the data
 $\text{mean } x_1 = \frac{4+8+13+7}{4} = 8$
 $\text{mean } x_2 = \frac{11+4+5+14}{4} = 8.5$

$\text{Y}_{\text{centered}} = \begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.5 & 4-8.5 & 5-8.5 & 14-8.5 \end{bmatrix} = \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$

$Z = e_1^T \cdot Y_{\text{centered}}$

$z_1 = (0.5574)(-4) + (-0.8303)(2.5) = -4.3035$
 $z_2 = (0.8303)(0) + (0.5574)(-4.5) = 3.73635$

Page 24 (Right):

PAGE NO: 24
DATE:

$z_3 = (0.5574)(5) + (-0.8303)(-3.5) = 5.69305$
 $z_4 = (0.5574)(-1) + (-0.8303)(5.5) = -5.12405$

$Z = [-4.3035, 3.73635, 5.69305, -5.12405]$

Model Accuracy without PCA
SVM: 0.8804
Logistic Regression: 0.8533
Random Forest: 0.8859

Model Accuracy with PCA
SVM: 0.8424
Logistic Regression: 0.8641
Random Forest: 0.8533

7/15/2025

Code:

```

import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
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 dataset
df = pd.read_csv("/content/heart (1).csv") # Update to match your file path if needed

# Define features and target
X = df.drop('HeartDisease', axis=1)

```

```

y = df['HeartDisease']

# Identify categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()

# Encode categorical columns
for col in categorical_cols:
    if X[col].nunique() == 2:
        X[col] = LabelEncoder().fit_transform(X[col])
    else:
        X = pd.get_dummies(X, columns=[col])

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

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

# Train and evaluate models (without PCA)
print("🔍 Accuracy without PCA:")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{name}: {accuracy_score(y_test, y_pred):.4f}")

# Apply PCA (reduce to 5 components)
pca = PCA(n_components=5)
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)

# Train and evaluate models (with PCA)
print("\n🧩 Accuracy with PCA:")
for name, model in models.items():
    model.fit(X_train_pca, y_train_pca)
    y_pred_pca = model.predict(X_test_pca)
    print(f"{name}: {accuracy_score(y_test_pca, y_pred_pca):.4f}")

```

🔍 Accuracy without PCA:
 SVM: 0.8804
 Logistic Regression: 0.8533

Random Forest: 0.8859

Accuracy with PCA:

SVM: 0.8424

Logistic Regression: 0.8641

Random Forest: 0.8533