

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on **Machine Learning (23CS6PCMAL)**

Submitted by

Sindhuja Narasimhan (1BM22CS279)

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 **Sindhuja Narasimhan (1BM22CS279)**, 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: **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	21-2-2025	Write a python program to import and export data using Pandas library functions	1
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	4
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	6
4	17-3-2025	Build Logistic Regression Model for a given dataset	9
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	15
6	7-4-2025	Build KNN Classification model for a given dataset	18
7	21-4-2025	Build Support vector machine model for a given dataset	22
8	5-5-2025	Implement Random forest ensemble method on a given dataset	24
9	5-5-2025	Implement Boosting ensemble method on a given dataset	26
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file	29
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	31

Github Link:

https://github.com/sindhuja279/ML_Lab.git

Program 1

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

Screenshot

The image shows two pages of handwritten notes from a notebook. The left page is dated 05/03/23 and titled 'Lab 0'. It contains code examples for importing data and printing headshots. The right page is dated 05/03/23 and titled 'TD DO2'. It contains code examples for downloading stock data, grouping by tickers, and plotting closing prices for HDFC, ICICI, and Kotak banks.

Lab 0

1. Import pandas as pd
df = pd.DataFrame({
 'USN': [101, 102, 103],
 'Name': ['Alicia', 'Bob', 'Charlie', 'David'],
 'Marks': [25, 30, 35, 40]
})
af = pd.DataFrame(df)
print(af.head())
2. df
3. df = pd.read_csv('diabetes.csv')
df = df.drop(['Pregnancies'], axis=1)
df = df.drop(['SkinThickness'], axis=1)
df = df.drop(['Insulin'], axis=1)
df = df.drop(['Age'], axis=1)
df = df.drop(['Diabetes'], axis=1)
df = df.drop(['Glucose'], axis=1)
4. df = pd.read_csv('dataset_of_diabetes.csv')
print(df.head())

TD DO2

1. import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
2. filepaths = ['HDFCBANK.NS', 'ICICIBANK.NS', 'KOTAKBANK.NS']
data = yf.download(filepaths, start='2024-01-01', end='2024-01-31', group_by='ticker')
print(data.head())
3. hdfc_data = data['HDFCBANK.NS']
icici_data = data['ICICIBANK.NS']
kotak_data = data['KOTAKBANK.NS']
print(hdfc_data.describe())
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc_data['Close'].plot(title='HDFC Closing Price')
plt.subplot(2, 1, 2)
hdfc_data['Daily Return'].plot(title='HDFC Daily Return', color='orange')
plt.tight_layout()
plt.show()
6. Similarly for Kotak & ICICI bank

```

Code:
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())

import pandas as pd
data={
    'USN':[100,101,102,103],
    'Name':['Alice','Bob','Charlie','David'],
    'Marks':[25,30,35,40],
}
df=pd.DataFrame(data)
print(df.head())


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


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

df = pd.read_csv('/content/Dataset_of_Diabetes.csv')
print("Sample data:")
print(df.head())


import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')

print("First 5 rows of the dataset:")

```

```

print(data.head())

hdfc_data=data['HDFCBANK.NS']
icici_data=data['ICICIBANK.NS']
kotak_data=data['KOTAKBANK.NS']

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

print("\nSummary statistics for Reliance Industries:")
print(icici_data.describe())
icici_data['Daily Return'] = icici_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
icici_data['Close'].plot(title="ICICI - Closing Price")
plt.subplot(2, 1, 2)
icici_data['Daily Return'].plot(title="ICICI - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

print("\nSummary statistics for Reliance Industries:")
print(kotak_data.describe())
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
kotak_data['Close'].plot(title="kotak - Closing Price")
plt.subplot(2, 1, 2)
kotak_data['Daily Return'].plot(title="kotak - Daily Returns", color='orange')
plt.tight_layout()
plt.show()

```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot

05/03/25 CLASSMATE Date _____ Page _____

Lab 3 - Data Preprocessing

1. import pandas as pd
import numpy as np
2. df = pd.read_csv('content/housing.csv')
3. print(df)
4. df.describe()
5. df['ocean_proximity'].value_counts()
df['Ocean proximity'].unique()
6. mv = df['ocean_proximity'].sum()
cmv = mv/len(df)
print(cmv)

Diabetes & Adult dataset

1. df4 = pd.read_csv('content/dataset_of_diabetes.csv')
df4.head()
2. df5 = pd.read_csv('content/adult.csv')
df5.head()
3. missing_col = df5.columns[df5.isnull().sum() > 0]
print(missing_col.adult.to_list())
4. categorical = df5.select_dtypes(include=['object']).columns
print(categorical.to_list())
5. missing_diabetes = df5.columns[df5.isnull().sum() > 0]
print(missing_diabetes.to_list())

15/03/25 CLASSMATE Date _____ Page _____

Q1 Categorical df = df5.select_dtypes(include=['object']).columns
print(categorical.to_list())

Q2 Difference b/w Min-Max Scaling & Standardization
→ Min-max → also called normalization transform
data to fit within a specific range (0 → 1).

Standardization scales data by subtracting the mean and dividing by standard deviation.

Q3 If the column in dataset has missing value we can replace categorical value with mode and numerical value with median.

Q4 In diabetes column we have class which in adult categorical column we work class, same we use ordinal encoder to encode categorical columns

Encoder

Code:

```
import pandas as pd
import numpy as np
df = pd.read_csv('/content/housing.csv')
print(df)

df.describe()

df['ocean_proximity'].value_counts()

df['ocean_proximity'].nunique()

mv=df.isnull().sum()
cmv=mv[mv>0]
print(cmv)

df4=pd.read_csv('/content/adult.csv')
df4.head()

df5=pd.read_csv('/content/Dataset_of_Diabetes.csv')
df5.head()

missing_cols_adult = df4.columns[df4.isnull().sum() > 0]
print(f"Columns with missing values in 'adult.csv': {missing_cols_adult.tolist()}")

missing_cols_diabetes = df5.columns[df5.isnull().sum() > 0]
print(f"Columns with missing values in 'Dataset of Diabetes .csv': {missing_cols_diabetes.tolist()}")

categorical_cols_adult = df4.select_dtypes(include=['object']).columns
print(f"Categorical columns in 'adult.csv': {categorical_cols_adult.tolist()}")

categorical_cols_diabetes = df5.select_dtypes(include=['object']).columns
print(f"Categorical columns in 'Dataset of Diabetes .csv': {categorical_cols_diabetes.tolist()}")
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

19/3/2022

Lab 3: Linear Regression.

1) import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

x = np.array([1, 2, 3, 2.7, 2.8, 2.9])
y = np.array([1, 3, 2, 3])
model = LinearRegression()
model.fit(x, y)
intercept = model.intercept_
coefficient = model.coef_[0]
print(f"Intercept (0): {intercept}")
print(f"Coefficient (1): {coefficient}")
x_new = np.array([2.5])
y_pred = model.predict(x_new)

print(f"Predicted y for x=5: {y_pred[0]}")
plt.scatter(x, y, color='blue', label='Data')

y_line = model.predict(x)
plt.plot(x, y_line, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression')
plt.legend()
plt.show()

2) Intercept (0): -1.4938...
Coefficient (1): 2.129...
Predicted y for x=5: 9.5

Scatter plot showing data points and a linear regression line.

2) import numpy as np
import matplotlib.pyplot as plt

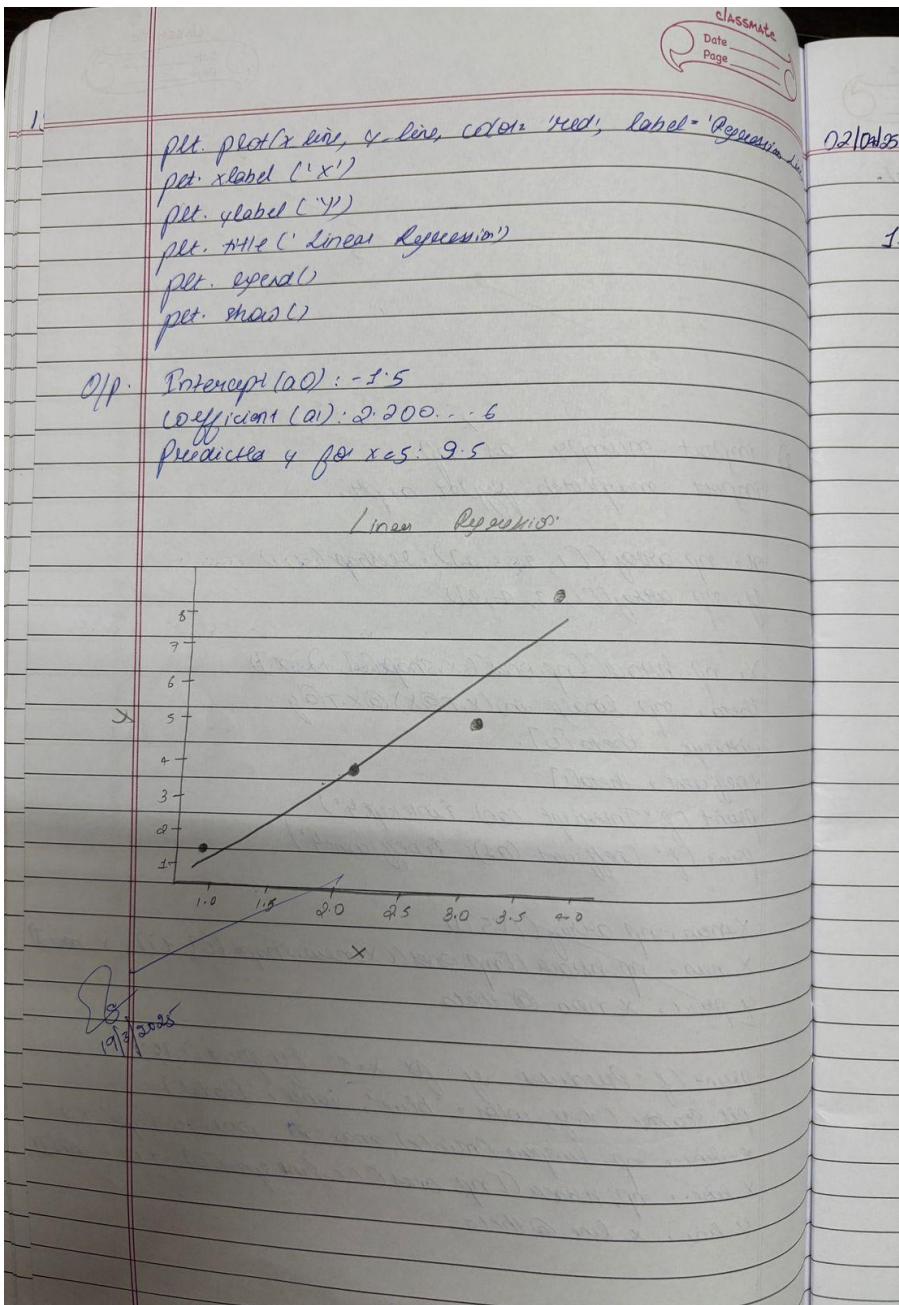
x = np.array([1, 2, 3, 2.7, 2.8, 2.9])
y = np.array([1, 3, 2, 3])

X = np.ones((np.ones(x).shape[0], 2))
theta = np.linalg.inv(X.T @ X) @ X.T @ y
intercept = theta[0]
coefficient = theta[1]
print(f"Intercept (0): {intercept}")
print(f"Coefficient (1): {coefficient}")

x_new = np.array([2.5])
x_new = np.ones((np.ones(x_new).shape[0], 1))
y_pred = x_new @ theta

print(f"Predicted y for x=5: {y_pred[0]}")
plt.scatter(x, y, color='blue', label='Data')

x_line = np.linspace(min(x), max(x), 100).reshape(-1, 1)
y_line = np.dot(np.ones((x_line.shape[0], 1)), x_line)
y_line = x_line @ theta



Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
x = np.array([1, 2, 3, 4]).reshape(-1, 1)
y = np.array([1, 3, 4, 8])
model = LinearRegression()
model.fit(x, y)
intercept = model.intercept_
coefficient = model.coef_[0]
print(f"Intercept (a0): {intercept}")
print(f"Coefficient (a1): {coefficient}")
x_new = np.array([[5]])
y_pred = model.predict(x_new)
print(f"Predicted y for x=5: {y_pred[0]}")
plt.scatter(x, y, color='blue', label='Data')
y_line = model.predict(x)
plt.plot(x, y_line, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression')
plt.legend()
plt.show()

import numpy as np
import matplotlib.pyplot as plt
x = np.array([1, 2, 3, 4]).reshape(-1, 1)
y = np.array([1, 3, 4, 8])
X = np.hstack([np.ones((x.shape[0], 1)), x])
theta = np.linalg.inv(X.T @ X) @ X.T @ y
intercept = theta[0]
coefficient = theta[1]
print(f"Intercept (a0): {intercept}")
print(f"Coefficient (a1): {coefficient}")
x_new = np.array([[5]])
X_new = np.hstack([np.ones((x_new.shape[0], 1)), x_new])
y_pred = X_new @ theta

print(f"Predicted y for x=5: {y_pred[0]}")
plt.scatter(x, y, color='blue', label='Data')
x_line = np.linspace(min(x), max(x), 100).reshape(-1, 1)
X_line = np.hstack([np.ones((x_line.shape[0], 1)), x_line])
y_line = X_line @ theta

plt.plot(x_line, y_line, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression')
plt.legend()
plt.show()
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot

22/02/25

classmate
Date _____
Page _____

data + Logistic Regression

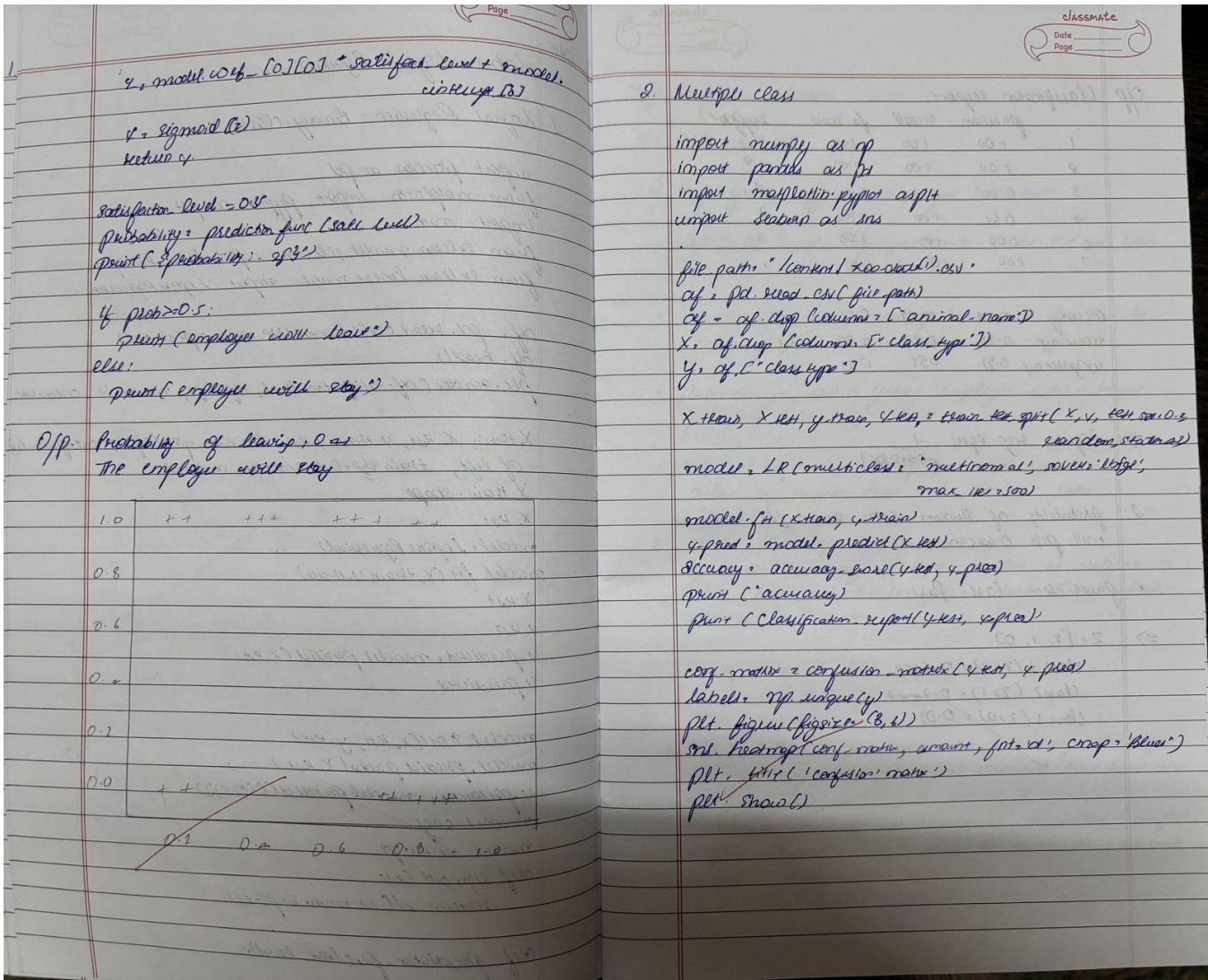
1. Logistic Regression - Binary class

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

df = pd.read_csv('content/HR_comma_sep.csv')
df.head()
plt.scatter(df['satisfaction_level'], df['left', marker='^', color='red'])

X_train, X_test, y_train, y_test, train_size, split(df['satisfaction_level'],
                                                df['left'], train_size=0.9, random_state=10)
X_train.shape
X_test
model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_pred = model.predict(X_test)
y_predicted
model.score(X_test, y_test)
model.predict_proba(X_test)
y_predicted, model.predict([[0.5]])
model.coef_
model.intercept_
def sigmoid(x):
    return 1/(1 + math.exp(-x))

def prediction_func(left_level):
```



Q1 Classification report						
	Precision	Recall	F1-Score	Support		
1	1.00	1.00	1.00	12		
2	1.00	1.00	1.00	2		
3	0.00	0.00	0.00	1		
4	0.67	1.00	0.80	2		
5	1.00	1.00	1.00	3		
6	1.00	1.00	1.00	1		
7	1.00	1.00	1.00	1		
Accuracy	0.73	0.73	0.73	21		
Macro Avg	0.78	0.78	0.78	21		
Weighted Avg	0.91	0.91	0.91	21		

Q1 Logistic Reg Egn: $\hat{P} = \frac{1}{1 + e^{-(Z + b)}}$

3. Probability of student who studies for 7 hours will pass 0.6253726257%

4. Predicted class: Pass.

5. $Z = [2, 1, 0]$

Class 1 ($Z+2$) = 0.6657

Class 2 ($Z+1$) = 0.2447

Class 3 ($Z+0$) = 0.09

Q2 Confusion Matrix

Q2 HR dataset:

- Key factor affecting employee retention.
- Satisfaction level - employees with lower satisfaction are more likely to leave.
- Number of projects & monthly hours - too few or too many projects/hours increase attrition rate.
- Performance in last 5 years - employees with recent poor performance tend to leave.
- Salary & department - employees with low salaries & certain departments show high attrition rates.

Model Accuracy & Evaluation

If the accuracy is high model is reliable. If it's low the dataset might need better feature selection or another model.

Accuracy achieved: 86.1

Q2 Zoo dataset:

- Yes data preprocessing was necessary. The steps include encoding the categorical variable (Class Type) into multiple categories.
- Removed animal name column as it is not useful.
- Split dataset into training-test split to evaluate model performance.

- CLASSMATE
Date _____
Page _____
- ii) checked for missing values before training. If any were found they would be handled by removal or imputation.
 - iii) The confusion matrix shows how well the model predicts each class. Diagonal values indicate correct prediction, while off-diagonal values express misclassifications.
 - iv) The most misclassified class could be some animal classes as it may be confuse due to overlapping features. Also the data may lack distinct features for certain groups.

Code:

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

# Load HR dataset
df = pd.read_csv("/content/HR_comma_sep (1).csv")
df.head()

plt.scatter(df.satisfaction_level, df.left, marker='+', color='red')

# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['satisfaction_level']], df.left, train_size=0.9,
random_state=10)
X_train.shape

X_test

# Training logistic regression model
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([[0.5]])

# model.coef_ indicates value of m in y=m*x + b equation
model.coef_

# model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

# Define sigmoid function and do the math manually
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def prediction_function(satisfaction_level):
    z = model.coef_[0][0] * satisfaction_level + model.intercept_[0]
    y = sigmoid(z)
    return y

satisfaction_level = 0.35
```

```

probability = prediction_function(satisfaction_level)
print(f"Probability of leaving: {probability:.2f}")

if probability >= 0.5:
    print("The employee will leave.")
else:
    print("The employee will stay.")

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

# Load dataset
file_path = "/content/zoo-data (1).csv"
df = pd.read_csv(file_path)

# Drop the animal_name column as it is not a feature
df = df.drop(columns=["animal_name"])

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

# 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)

# Train multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=500)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))

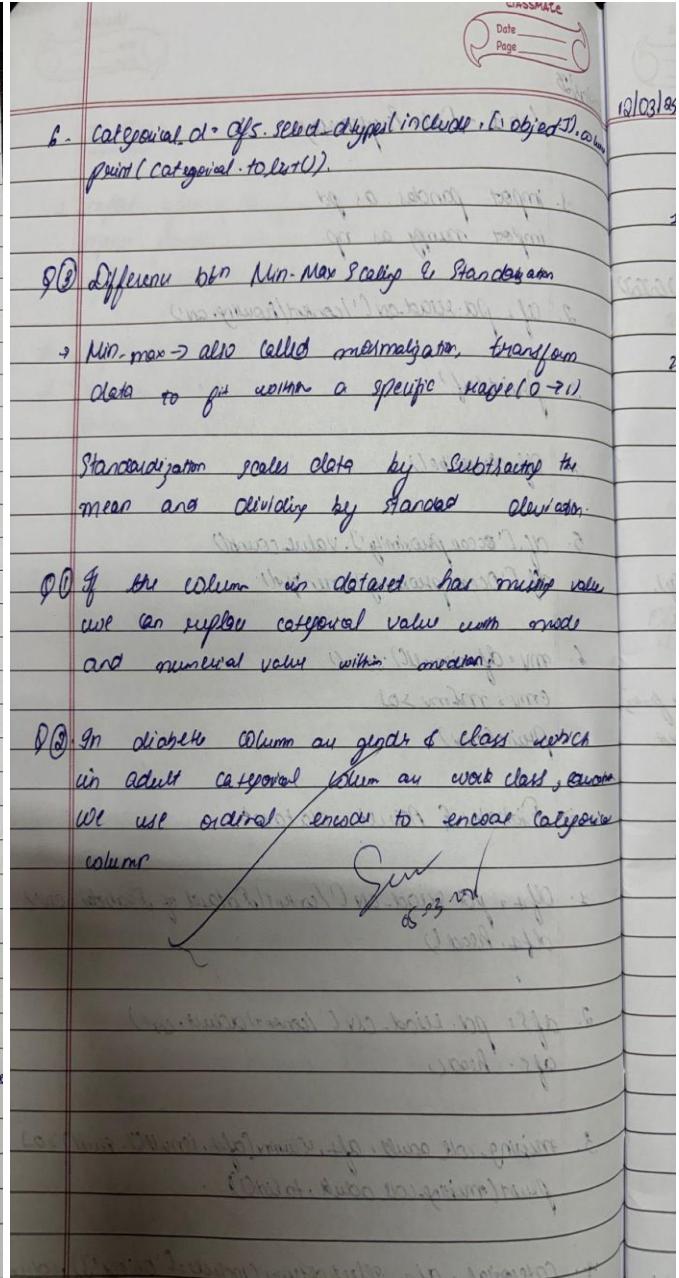
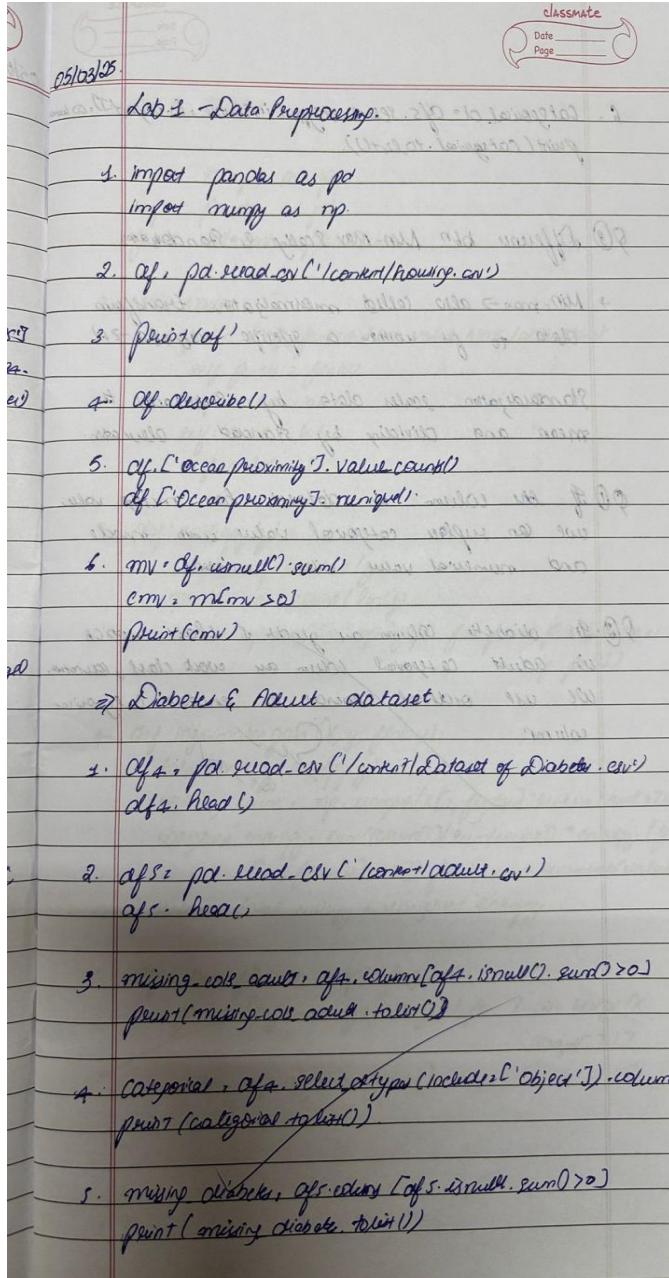
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
labels = np.unique(y)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

```

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 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)

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

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+1)}Value: {value}")
        print_tree(child, depth+1)

```

```

print(f"{' ' * depth}Value: {value}")
print_tree(child, depth + 1)

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', 'Mild', 'Cool', '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)

```

Program 6

Build KNN Classification model for a given dataset

Screenshot

CLASSMATE
Date _____
Page _____

02/04/2023

Lab 5: KNN.

1. $K=3$ and test data ($x = 35, y = 100$) as (Person, Age, Salary).

P	A	K	T	d.
A	18	50	N	52.81
B	23	55	N	46.57
C	24	70	N	31.95
D	41	60	V	40.95
E	43	70	V	31.05
F	38	40	V	60.07
X	35	100	V	

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

$K=1$ E - 31.05 ~~Y~~ NO V
 $K=2$ C - 31.95 ~~N~~ NO V
 $K=3$ D - 40.95 ~~V~~ NO V

Predicted class for (35, 100) as ~~Y~~ V

For this dataset.

Various values of K are tested and for each accuracy and error rate is calculated. This was done using grid search and the optimal found at $K=3$

For diabetes.

Feature scaling ensures all features contribute equally to the nearest neighbor. Scaling is done so that feature like glucose age don't dominate the cone with smaller range.

Code:

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

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

# Split features and labels
X = iris.iloc[:, :-1] # Features: all columns except last
y = iris.iloc[:, -1] # Labels: last column (species)

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

# Initialize and train the KNN model
k = 5 # Choosing k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt="d",
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Iris Dataset")
plt.show()
```

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

```

from sklearn.metrics import accuracy_score, confusion_matrix

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

# Split features and labels
X = diabetes.iloc[:, :-1] # Features: all columns except last
y = diabetes.iloc[:, -1] # Labels: last column (diabetic or not)

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

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

# Initialize and train the KNN model
k = 5 # Choosing k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt="d",
            xticklabels=["Non-Diabetic", "Diabetic"], yticklabels=["Non-Diabetic", "Diabetic"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Diabetes Dataset")
plt.show()

```

```

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

```

```

# Load dataset
df = pd.read_csv('/content/heart (1).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()

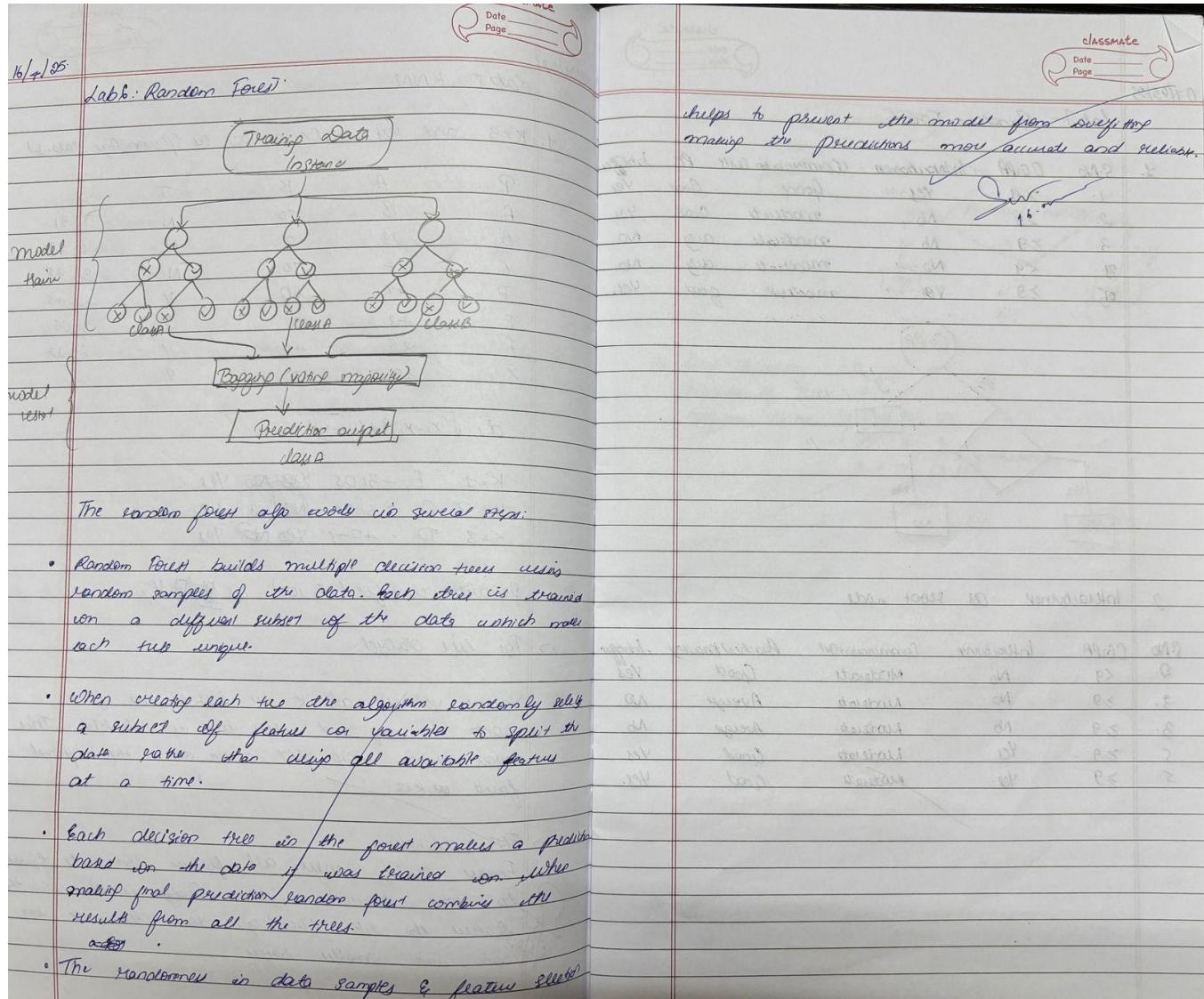
# Plot K values vs. Accuracy
plt.plot(k_values, accuracy_scores, marker='o')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.title('K Value vs Accuracy')
plt.show()

```

Program 7

Build Support vector machine model for a given dataset

Screenshot



Code:

```
# Load the important packages
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

#Build the model
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
# Trained the model
svm.fit(X, y)

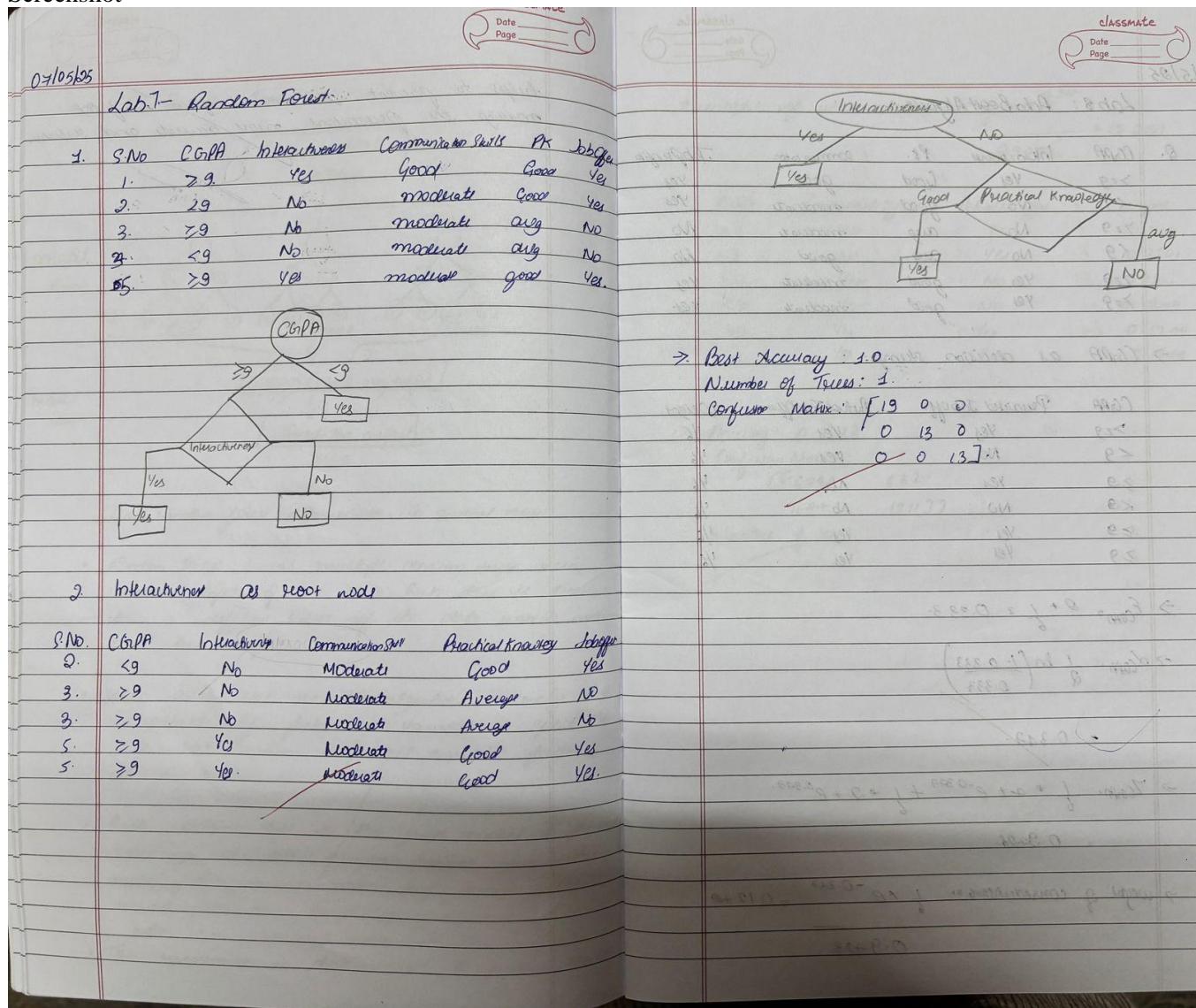
# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    cmap=plt.cm.Spectral,
    alpha=0.8,
    xlabel=cancer.feature_names[0],
    ylabel=cancer.feature_names[1],
)

# Scatter plot
plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot



Code:

```

import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the iris dataset
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Split features and target
X = df.drop('species', axis=1)
y = df['species']

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

best_score = 0
best_n = 0
best_cm = None

# Try different numbers of trees
for n in range(1, 101):
    clf = RandomForestClassifier(n_estimators=n, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    if acc > best_score:
        best_score = acc
        best_n = n
        best_cm = confusion_matrix(y_test, y_pred)

print(f"Best Accuracy: {best_score}")
print(f"Number of Trees: {best_n}")
print("Confusion Matrix:")
print(best_cm)

```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot

7/5/95

Lab 8: AdaBoost Algorithm.

Q. CGPA Interactions Pk. communication JobOffer

>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

→ CGPA as decision stump.

CGPA	Predicted JobOffer	Actual JobOffer	Weight.
>9	Yes	Yes	1/6
<9	No	Yes	1/6
>9	Yes	No	1/6
<9	No	No	1/6
>9	Yes	Yes	1/6
>9	Yes	Yes	1/6

→ $E_{CGPA} = \frac{2}{6} + \frac{1}{6} = 0.333$.

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

→ 0.342.

→ $Z_{CGPA} = \frac{1}{6} + 2 + e^{-0.342} + \frac{1}{6} + 2 + e^{0.342}$.

→ 0.9728.

→ weight of correctly classified = $\frac{1}{6} \times e^{-0.342} \approx 0.1249$.

0.9728

→ weight of misclassified instances: $\frac{1}{6} \times e^{0.342} \approx 0.2501$.

0.9728

CGPA PredictedJobOffer ActualJobOffer Weight.

>9	Yes	Yes	0.1249
<9	No	Yes	0.2501
>9	Yes	No	0.2501
<9	No	No	0.1249
>9	Yes	Yes	0.1249
>9	Yes	Yes	0.1249

→ Accuracy: 0.8182

Confusion Matrix:

11 G782	632
11 AGT	1211 J7

Number of trees:

1	2	3
2	3	4
3	4	5
4	5	6
5	6	7
6	7	8
7	8	9
8	9	10

Code:

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

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

# Encode categorical variables (if any)
label_encoders = {}
for column in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Split dataset into features and target
X = df.drop("income_level", axis=1) # replace 'income' with the actual target column name if different
y = df["income_level"]

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

# Train AdaBoost with default n_estimators=10
ada_default = AdaBoostClassifier(n_estimators=10, random_state=42)
ada_default.fit(X_train, y_train)
y_pred_default = ada_default.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred_default)

print(f"Default Accuracy (10 estimators): {default_accuracy:.4f}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_default))

# Tune number of estimators
scores = []
n_estimators_range = range(1, 101)

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

# Best accuracy and corresponding n_estimators
best_score = max(scores)
best_n_estimators = n_estimators_range[scores.index(best_score)]

# Final model with best n_estimators
```

```
ada_best = AdaBoostClassifier(n_estimators=best_n_estimators, random_state=42)
ada_best.fit(X_train, y_train)
y_pred_best = ada_best.predict(X_test)
conf_matrix_best = confusion_matrix(y_test, y_pred_best)

print(f"\nBest Accuracy: {best_score:.4f} using {best_n_estimators} estimators")
print("Best Confusion Matrix:")
print(conf_matrix_best)

# Optional: Plot accuracy vs number of estimators
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, scores, marker='o')
plt.title("AdaBoost Accuracy vs Number of Estimators")
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

Program 10

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

Screenshot

7/15/08 Lab 9 - K-Means.			
Record Number	A	B	
R1	1.0	1.0	
R2	1.5	2.0	
R3	3.0	4.0	
R4	5.0	7.0	
R5	3.5	5.0	
R6	4.5	5.0	
R7	3.5	4.5	

Iteration 1.			
Record	$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$	distance to C1(1,1)	distance to C2(5,3)
R1	0.00	7.21	C3
R2	3.12	6.10	C1
R3	3.61	4.24	C1
R4	7.21	0.00	C2
R5	5.00	2.50	C2
R6	5.82	2.24	C2
R7	4.61	8.92	C2

Clustered: R1, R2, R3
Mean A: $(1.0 + 1.5 + 3.0) / 3 = 1.83$
Mean B: $(1.0 + 2.0 + 4.0) / 3 = 2.33$
New C1: $(1.83, 2.33)$.

Cluster C2: R4, R5, R6, R7
Mean A: $(3.0 + 3.5 + 4.5 + 3.5) / 4 = 4.125$
Mean B: $(5.0 + 5.0 + 5.0 + 4.0) / 4 = 5.375$
New C2: $(4.125, 5.375)$.

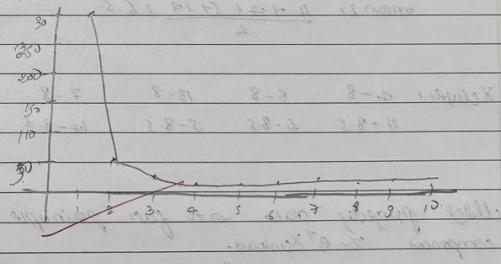
Iteration 2.

Record	$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$	distance to C1 (1.83, 2.33)	distance to C2 (5.375, 5.375)	assigned cluster
R1	1.87	5.91	C1	
R2	0.49	4.70	C1	
R3	1.89	1.63	C2	
R4	3.18	1.84	C2	
R5	2.93	0.70	C2	
R6	3.36	0.47	C2	
R7	2.50	0.39	C2	

Final clusters after 2 iterations.

C1: R1, R2
C2: R3, R4, R5, R6, R7.

? graph with elbow point K.



Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
# Select only petal length and width
X = df[['petal length (cm)', 'petal width (cm)']]
# Optional: Scale the features (important for distance-based algorithms like K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Elbow method to determine optimal number of clusters
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_)
# Plot the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
# From the elbow plot, we choose k = 3 (typical for Iris dataset)
optimal_k = 3
kmeans_final = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans_final.fit_predict(X_scaled)
# Add cluster labels to original data
df['Cluster'] = clusters
# Plotting the clusters
plt.figure(figsize=(8, 5))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap='viridis', s=50)
plt.scatter(kmeans_final.cluster_centers_[:, 0], kmeans_final.cluster_centers_[:, 1],
            c='red', s=200, alpha=0.7, marker='X', label='Centroids')
plt.title('K-Means Clustering (k=3) on Iris Petal Features')
plt.xlabel('Petal Length (scaled)')
plt.ylabel('Petal Width (scaled)')
plt.legend()
plt.grid(True)
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot

7/15/25
lab-10 PCA method.

Feature: $\begin{bmatrix} x_1 & x_2 \end{bmatrix}$ $\begin{bmatrix} e_1 & e_2 & e_3 & e_4 \end{bmatrix}$
 $x_1 = \begin{bmatrix} 4 & 8 & 13 & 7 \end{bmatrix}$
 $x_2 = \begin{bmatrix} 11 & 4 & 5 & 10 \end{bmatrix}$

Eigenvalue: $\lambda_1 = 30.3829$ Eigenvector: $e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$
 $\lambda_2 = 6.615$ $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

\Rightarrow data: $\begin{bmatrix} 4 & 8 & 13 & 7 \end{bmatrix}$.
matrix: $\begin{bmatrix} 11 & 4 & 5 & 10 \end{bmatrix}$

1) mean cluster the data
mean 1: $\frac{4+8+13+7}{4} = 8$
mean 2: $\frac{11+4+5+10}{4} = 6.5$.

$X_{\text{cluster}} = \begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.5 & 4-8.5 & 5-8.5 & 10-8.5 \end{bmatrix}$

Using projecting data onto first principle component $Z = e_1^T X_{\text{centered}}$.

$Z_1 = (0.5574)(-4) + (-0.8303)(2.5) = -4.30545$
 $Z_2 = (0.5574)(0) + (-0.8303)(-4) = 3.73635$
 $Z_3 = (0.5574)(5) + (-0.8303)(-3.5) = 5.69305$

Date: 7/15/25
Page: 1/1

$Z = [-4.30545, 3.73635, 5.69305, -5.12405]$

$\therefore Z = [-4.30545, 3.73635, 5.69305, -5.12405]$

→ Model accuracy:

Before	After
Logistic: 0.8833	0.8663
Linear: 0.8776	0.8516
Random: 0.8009	0.8722

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 (2).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}')
```