VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Sinchana R (1BM22CS78)

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 **Sinchana R (1BM22CS78),** 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: **Sheetal V A** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

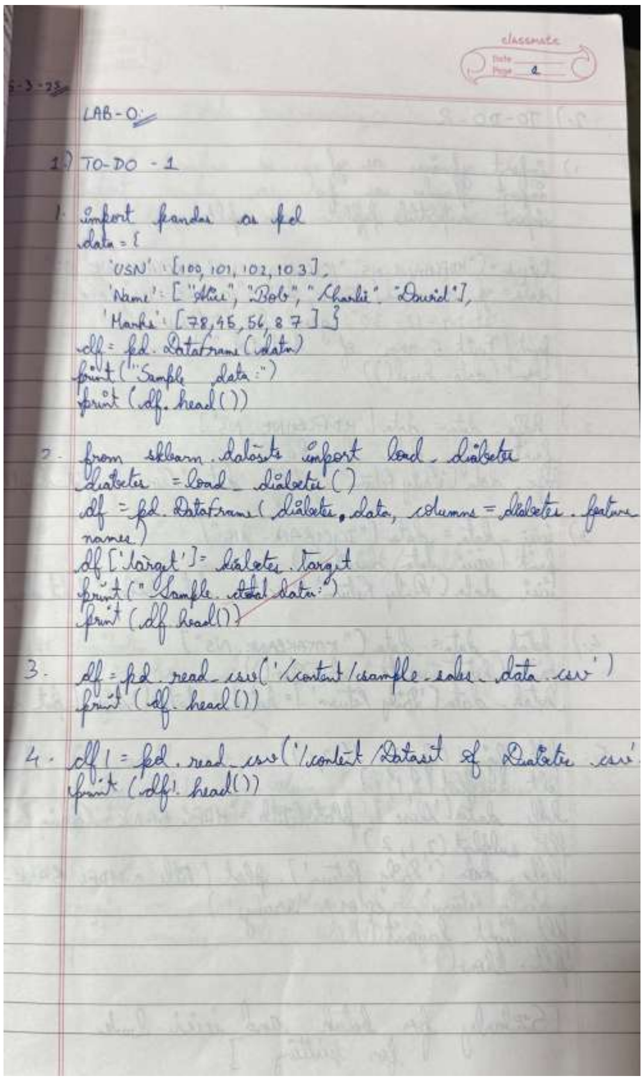
Index

Github Link:
https://github.com/sinchana-08/ML_LAB

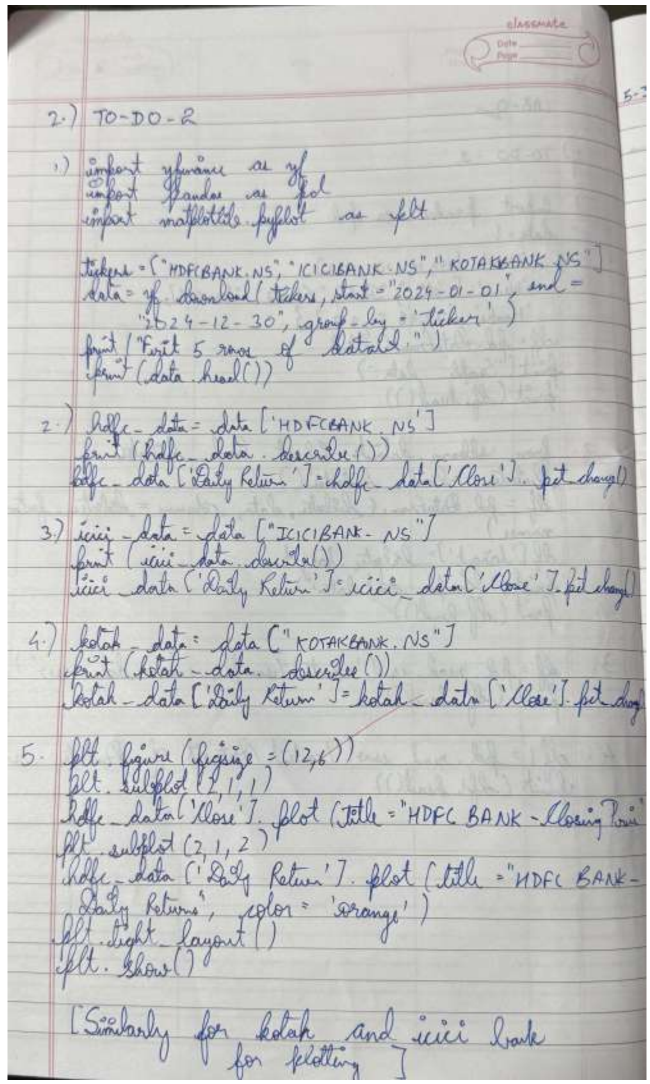
Program 1

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

Screenshot



```
5-3-25

LAB-0

1) TO-DO - 1

1. import pandas as pd
data = {
    'USN' : [100, 101, 102, 103],
    'Name' : ["Alice", "Bob", "Charlie", "David"],
    'Marks' : [78, 45, 56, 87] }
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())

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

3. df = pd.read_csv('/content/sample_sales_data.csv')
print(df.head())

4. df1 = pd.read_csv('/content/Dataset of Diabetes.csv')
print(df1.head())
```



```
2.) TO-DO-2

1.) 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 dataset:")
print(data.head())

2.) hdfc_data = data['HDFCBANK.NS']
print(hdfc_data.describe())
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

3.) icici_data = data["ICICIBANK.NS"]
print(icici_data.describe())
icici_data['Daily Return'] = icici_data['Close'].pct_change()

4.) kotak_data = data["KOTAKBANK.NS"]
print(kotak_data.describe())
kotak_data['Daily Return'] = kotak_data['Close'].pct_chang

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

[Similarly for kotak and icici bank
        for plotting ]
```

Code:

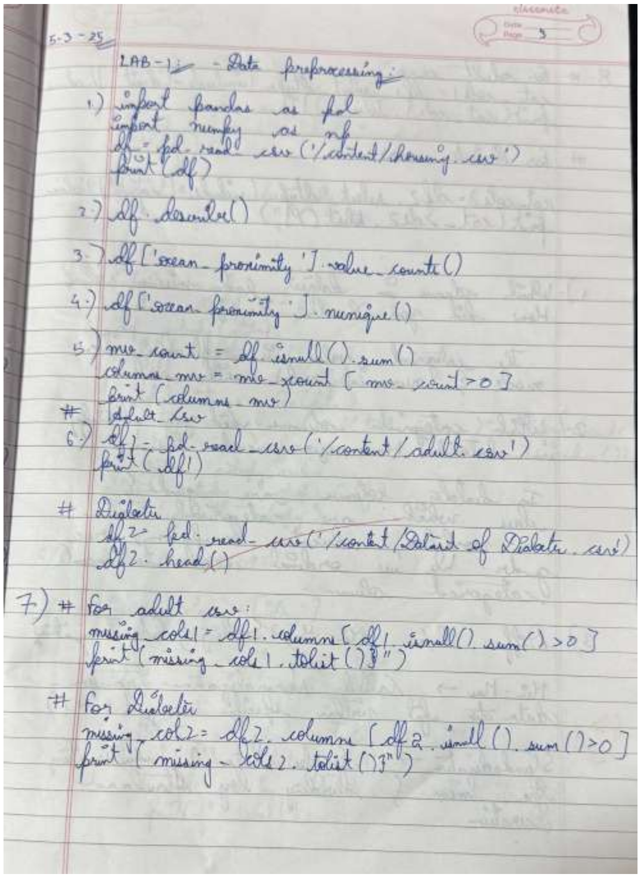
```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())

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

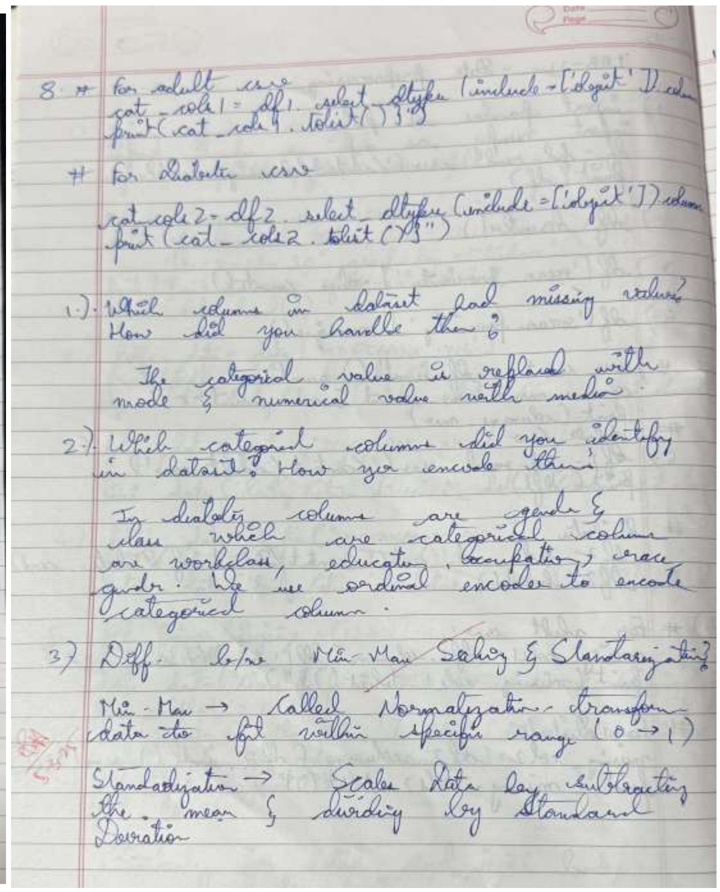
Screenshot



5-3-25

LAB-1 - Data preprocessing:

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

2.) df.describe()

3.) df['ocean_proximity'].value_counts()

4.) df['ocean_proximity'].nunique()

5.) mv_count = df.isnull().sum()
    columns_mv = mv_count[mv_count>0]
    print(columns_mv)
#   Adult csv
6.) df1 = pd.read_csv('/content/adult.csv')
    print(df1)

#   Diabetes
    df2 = pd.read_csv('/content/Dataset of Diabetes.csv')
    df2.head()

7) # for adult csv:
    missing_cols1 = df1.columns[df1.isnull().sum()>0]
    print(missing_cols1.tolist()}")

#  For Diabetes
    missing_col2 = df2.columns[df2.isnull().sum()>0]
    print(missing_cols2.tolist()}")
```



```
8. # for adult csv
    cat_cols1 = df1.select_dtypes(include=['object']).colum
    print(cat_cols1.tolist()}")

#  for Diabetes csv

    cat_cols2 = df2.select_dtypes(include=['object']).colum
    print(cat_cols2.tolist()}")
```

1.) Which columns in dataset had missing values? How did you handle them?

The categorical value is replaced with mode & numerical value with median.

2.) Which categorical columns did you identify in dataset? How you encode them?

In diabetes columns are gender & class which are categorical column are workclass, education, occupation, race, gender. We use ordinal encoder to encode categorical column.

3) Diff. b/w Min Max Scaling & Standardization?

Min-Max → Called Normalization transform data to fit within specific range (0 → 1)

Standardization → Scales Data by subtracting the mean & dividing by Standard Deviation

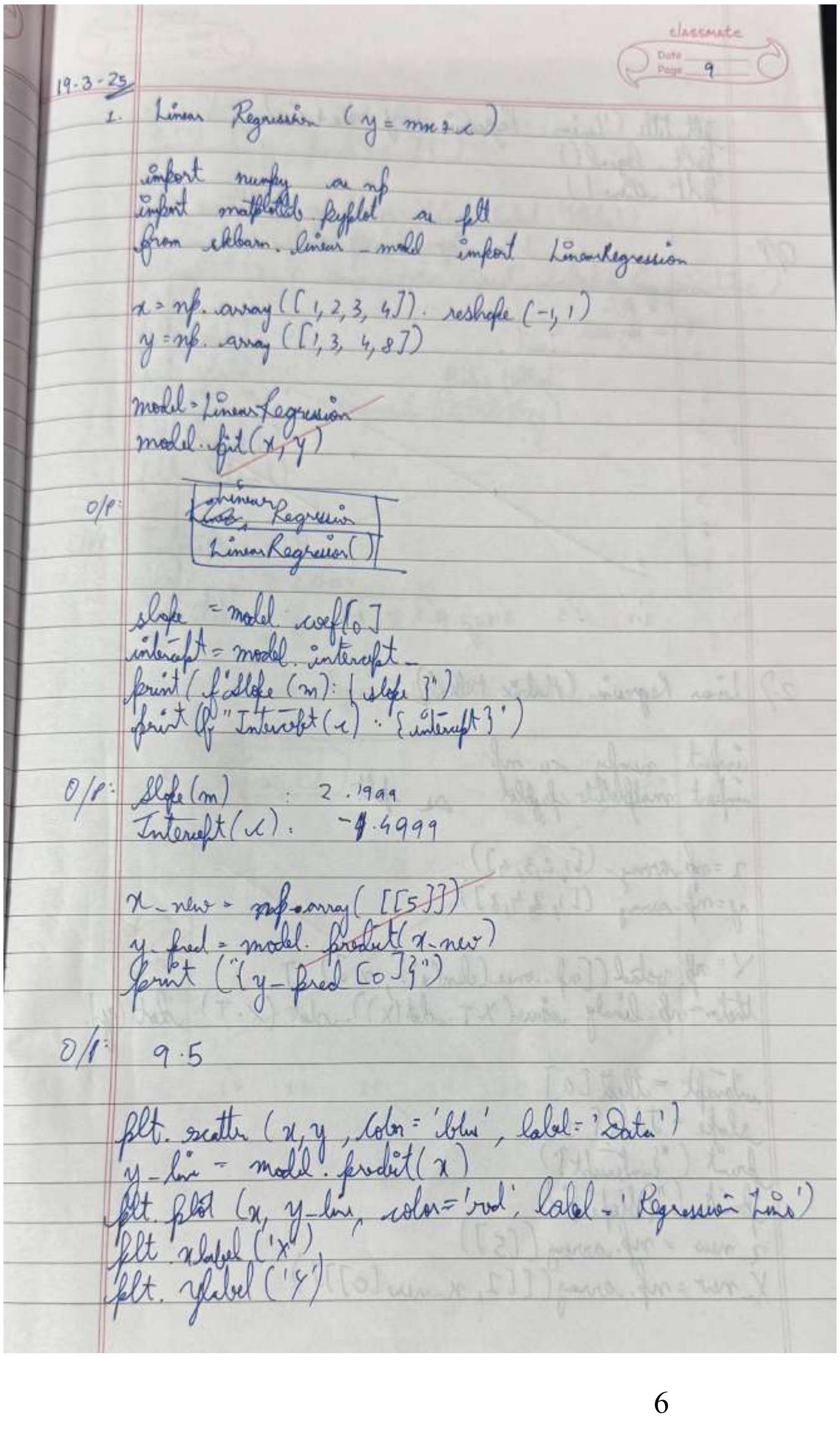
Code:

```
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


19-3-25

Page 9

1. Linear Regression (y = mx + c)

```
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)
```

O/P:

```
LinearRegression
LinearRegression()
```

```
slope = model.coef[0]
intercept = model.intercept_
print(f"Slope (m): {slope}")
print(f"Intercept(c): {intercept}")
```

O/P:

```
Slope(m)     : 2.1999
Intercept(c) : -1.4999
```

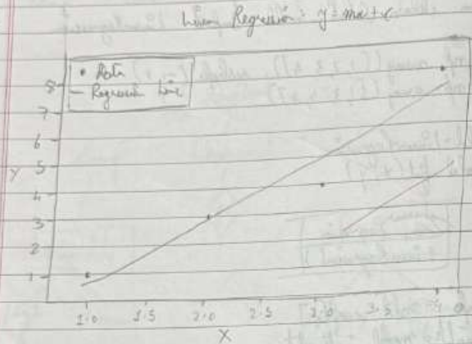
```
x_new = np.array([[5]])
y_pred = model.predict(x_new)
print(f"{y_pred[0]}")
```

O/P: 9.5

```
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: y = mx + c')
plt.legend()
plt.show()
```

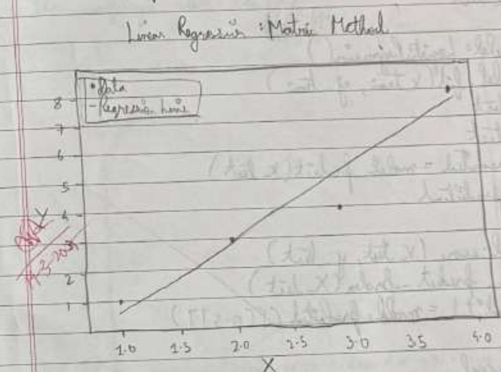
O/P:



2.) Linear Regression (Matrix Method)

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

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

X = np.vstack([np.ones(len(x)), x]).T
theta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

intercept = theta[0]
slope = theta[1]
print("{intercept}")
print("{slope}")
x_new = np.array([5])
X_new = np.array([[1, x_new[0]]])
y_pred = X_new.dot(theta)
print("{y_pred[0]}")

plt.scatter(x, y, color='blue', label='Data')
y_line = X.dot(theta)
plt.plot(x, y_line, color='red', label='Regression Line')

plt.xlabel('x')
plt.ylabel('y')
plt.title("Linear Regression: Matrix Method")
plt.legend()
plt.show()
```

O/P: Intercept : -1.5
Slope : 2.2000
Predicted y for x=5 : 9.5000



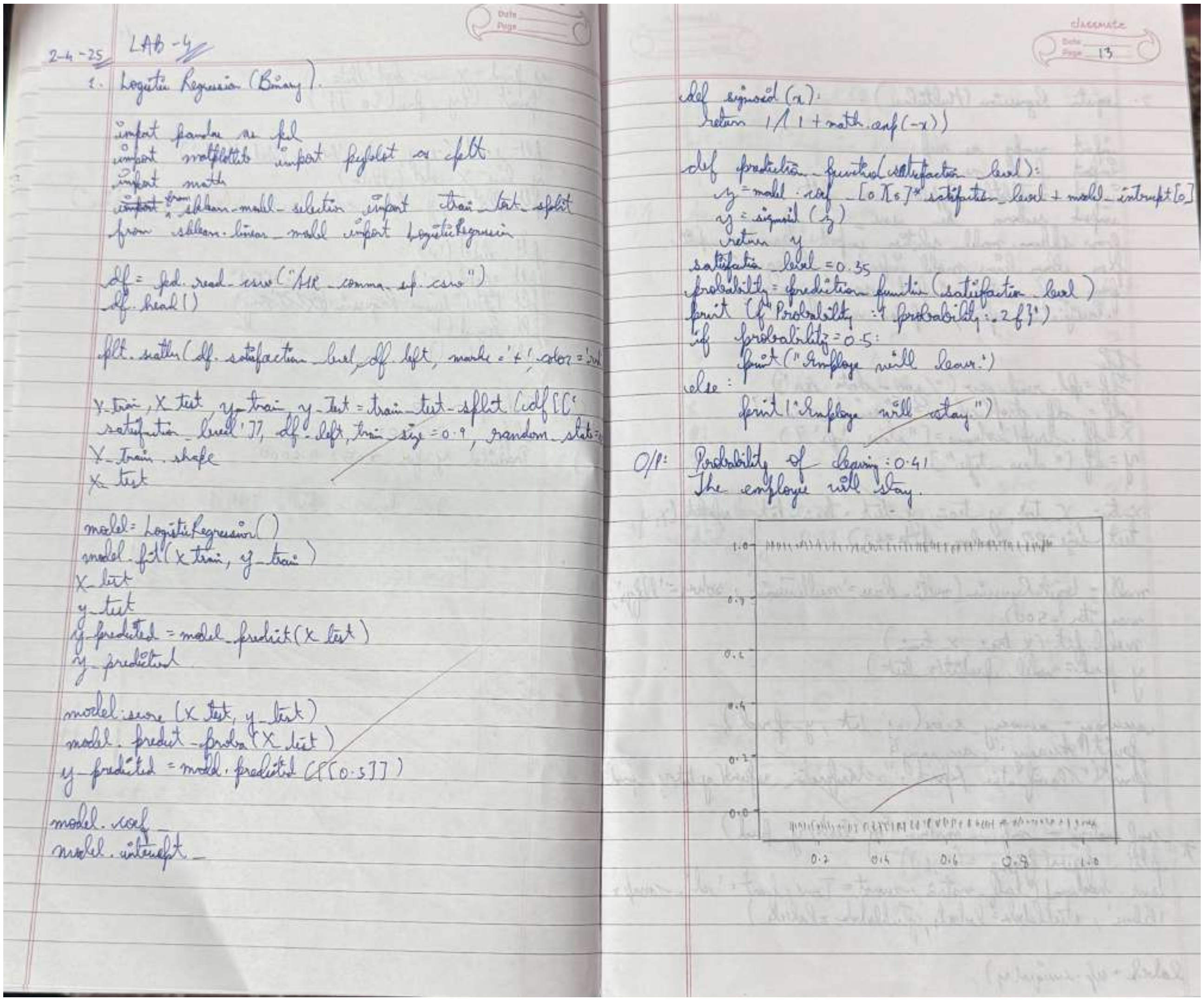
Code:

```
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



7. Logistic Regression (Multiclass).

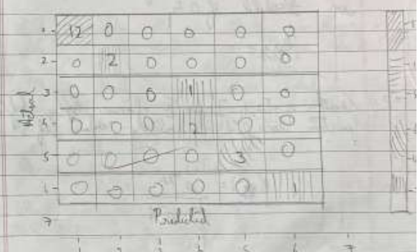
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

df = pd.read_csv("/zoo-data.csv")
df = df.drop(columns=["animal_name"])
X = df.drop(columns=["class_type"])
y = df["class_type"]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=500)
model.fit(x_train, x_train)
y_pred = model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:", classification_report(y_test, y_pred

* conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)

* labels = np.unique(y).
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

O/P: Accuracy: 0.9523809

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
6	1.00	1.00	1.00	3
7	1.00	1.00	1.00	1
accuracy			0.95	21
macro avg	0.78	0.83	0.80	21
weighted avg	0.92	0.95	0.93	21



1. Consider binary classification problem where we want to predict whether a student will pass or fail based on their study hours. Logistic regression model has been trained and learned parameters are $a_0 = 5$ (intercept) & $a_1 = 0.1$ (coeff. of study hours)

a.) Write logistic regression eq.

$$\frac{1}{1+e^{-(a_0+a_1x)}} = \frac{1}{1+e^{-(5+0.1x)}}$$

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

$$\frac{1}{1+e^{-(5+0.1(7))}} = 0.6457 = 64.57\%$$

c.) Determine predicted class for this student based on threshold of 0.5

Pass

2.) Consider $z = [2, 1, 0]$ for 3 classes. Apply Softmax function to find probability values of 3 classes.

$$P(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{3} e^{z_j}}$$

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

Sum $= 7.389 + 2.718 + 1 = 11.107$

$$P(z_1) = \frac{e^2}{Sum} = \frac{7.389}{11.107} = 0.665$$

$$P(z_2) = \frac{e^1}{Sum} = \frac{2.718}{11.107} = 0.245$$

$$P(z_3) = \frac{e^0}{Sum} = \frac{1}{11.107} = 0.090$$

$P[z] = [0.665, 0.245, 0.090]$

1. For dataset HR_comma_sep.csv:

i). Which variable did you identify as having direct & clear impact on employee retention? Why?

→ Satisfaction Level because employees with lower satisfaction level are more likely to leave (1 means left, 0 mean stay in dataset). This is visualized with scatter plot to show people leaving because of low satisfaction.

ii). What was the accuracy of the model? Do you think this is a good accuracy? Why or why not?

→ Accuracy obtained using model.score(x_test, y_te

→ Good accuracy: if 80-90%

→ If it's too high (~99%) may indicate overfitting, low (50-60%), indicate underfitting.

2. For your dataset:

i) Did you perform data preprocessing steps? If yes, what and why were they necessary?

→ Yes.

→ Dropped animal-name column because it wasn't useful for prediction.

→ ~~Drop~~ Separated features (x) and target (y) for model training.

(ii) Were there missing or inconsistent values?

→ No.

(iii) What does confusion matrix tell you about performance of model?

→ It shows how many predictions were correct & how many were misclassified. It shows which classes the model struggles to predict.

(iv) Which class types were most frequent misclassifi[ed]? Why?

→ Classes having high off-diagonal value in confusion matrix were misclassified ~~d~~ due to similarities b/w the classes

Code:

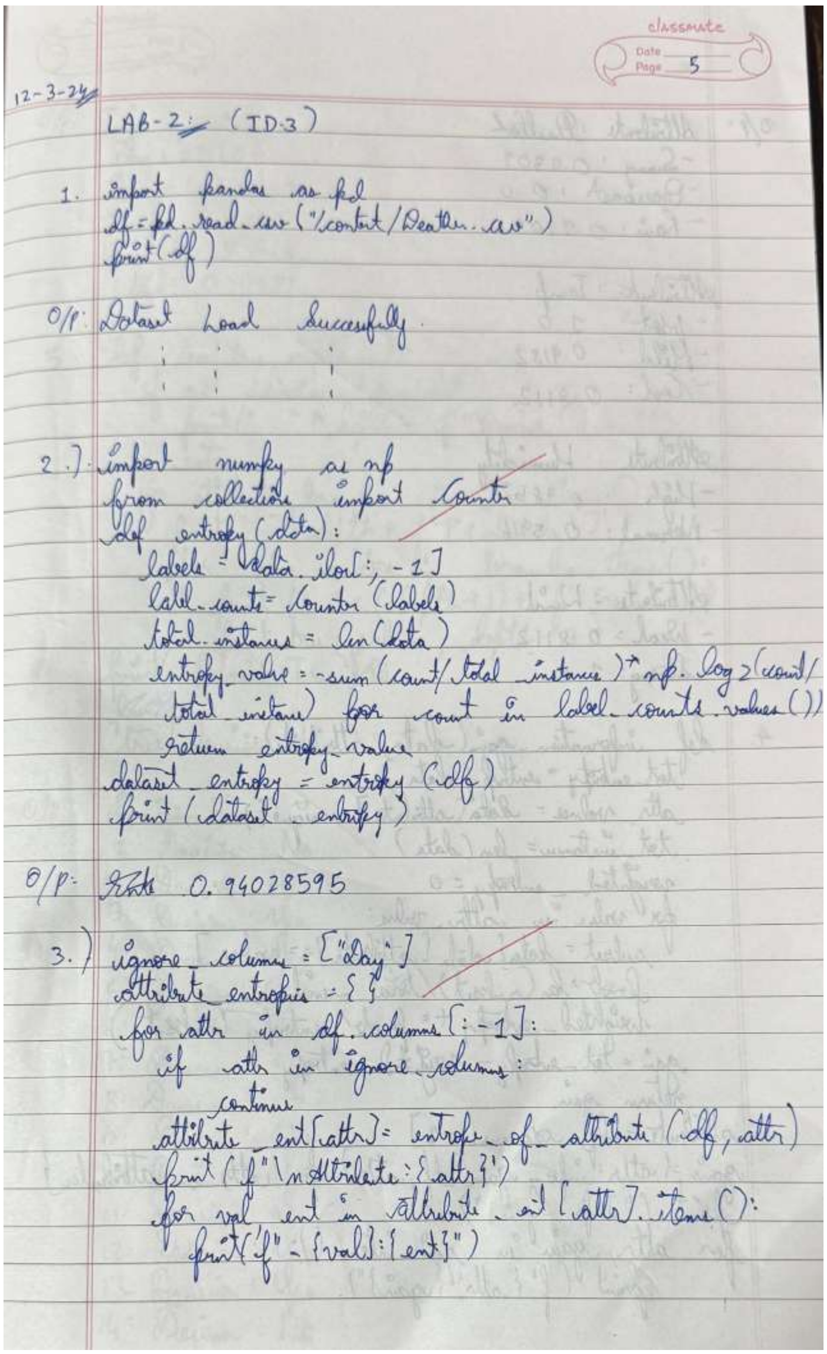
```
import pandas as pd
from matplotlib import pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# 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



classmate

Date

Page 5

12-3-24

LAB-2: (ID-3)

1. import pandas as pd
df = pd.read_csv("/content/Weather.csv")
print(df)

O/P: Dataset Load Succesfully.

2.) import numpy as np
from collections import Counter
def entropy(data):
labels = data.iloc[:, -1]
label_counts = Counter(labels)
total_instances = len(data)
entropy_value = -sum(count/total_instances) * np.log2(count/total_instances) for count in label_counts.values())
return entropy_value
dataset_entropy = entropy(df)
print(dataset_entropy)

O/P: 0.94028595

3.) ignore_columns = ["Day"]
attribute_entropies = {}
for attr in df.columns[:-1]:
if attr in ignore_columns:
continue
attribute_ent[attr] = entropy_of_attribute(df, attr)
print(f"\nAttribute: {attr}")
for val, ent in attribute_ent[attr].items():
print(f" - {val}: {ent}")

Page 7

O/P: Attribute : Outlook
- Sunny : 0.9709
- Overcast : 0.0
- Rain : 0.9709

Attribute : Temp
- Hot : 1.0
- Mild : 0.9182
- Cool : 0.8112

Attribute : Humidity
- High : 0.9852
- Normal : 0.5916

Attribute : Wind
- Weak : 0.8112
- Strong : 1.0

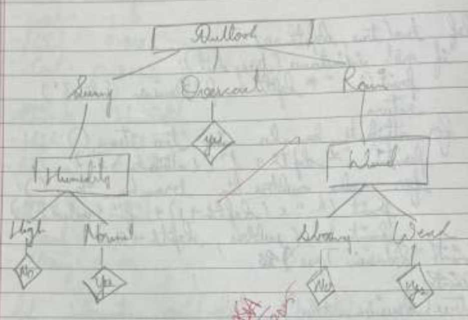
4.
```
def information_gain(data, attribute):
    tot_entropy = entropy(data)
    attr_values = data[attribute].unique()
    tot_instance = len(data)
    weighted_entropy = 0
    for value in attr_values:
        subset = data[data[attribute] == value]
        prob = len(subset)/total_instances
        weighted_entropy += prob * entropy(subset)
    gain = tot_entropy - weighted_entropy
    return gain
attributes = df.columns[:-1]
gains = {attr: info_gain(df, attr) for attr in attributes}
print("Info Gain:")
for attr, gain in gains.items():
    print(f"{attr}{gain}")
```

O/P: Info gain:
Day : 0.9402
Outlook : 0.2469
Temp : 0.0292
Humidity : 0.1518
Wind : 0.0481

5.
```
def tree(tree, depth=0):
    if not isinstance(tree, dict):
        print(" " * depth + f"Decision : {tree}")
        return
    for attribute, branches in tree.items():
        print(" " * depth + f"[{attribute}]")
        for value, subtree in branches.items():
            print(" " * (depth+1) + f"{value}:")
            print_tree(subtree, depth+2)
print("Decision Tree")
tree(decision_tree)
```

O/P: Decision Tree:
1: Decision : No
2: Decision : No
3: Decision : Yes
4: Decision : Yes
5: Decision : Yes
6: Decision : No
7: Decision : Yes
8: Decision : No
9: Decision : Yes
10: Decision : Yes
11: Decision : Yes
12: Decision : Yes
13: Decision : Yes
14: Decision : No

{'Outlook' : {'Sunny' : {humidity : {'high' : 'no',
'normal' : 'yes' }},
{'Overcast' : 'yes',
'Rainy' : {'Wind' : {'Weak' : 'yes', 'Strong' : 'No'}}}}

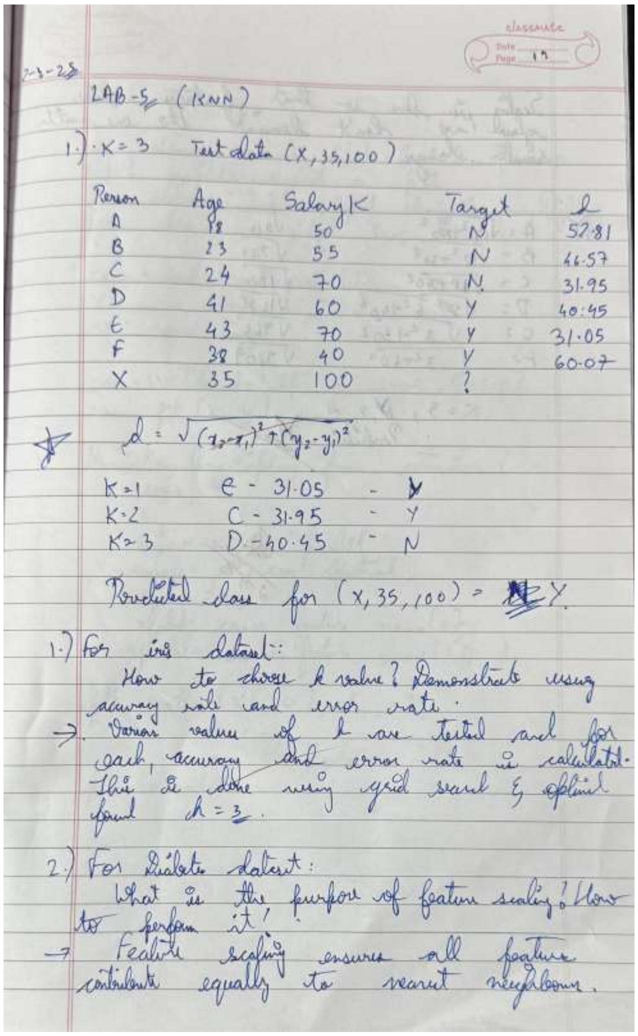


Code:
```
import numpy as np
import pandas as pd
from collections import Counter

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}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', '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



2-3-25

LAB-5 (KNN)

1.) K = 3 Test data (X, 35, 100)

Person	Age	Salary K	Target	d
A	18	50	N	57.81
B	23	55	N	46.57
C	24	70	N	31.95
D	41	60	Y	40.45
E	43	70	Y	31.05
F	38	40	Y	60.07
X	35	100	?	

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

K=1 E - 31.05 - Y

K=2 C - 31.95 - Y

K=3 D - 40.45 - N

Predicted class for (X, 35, 100) = Y

1.) For iris dataset:

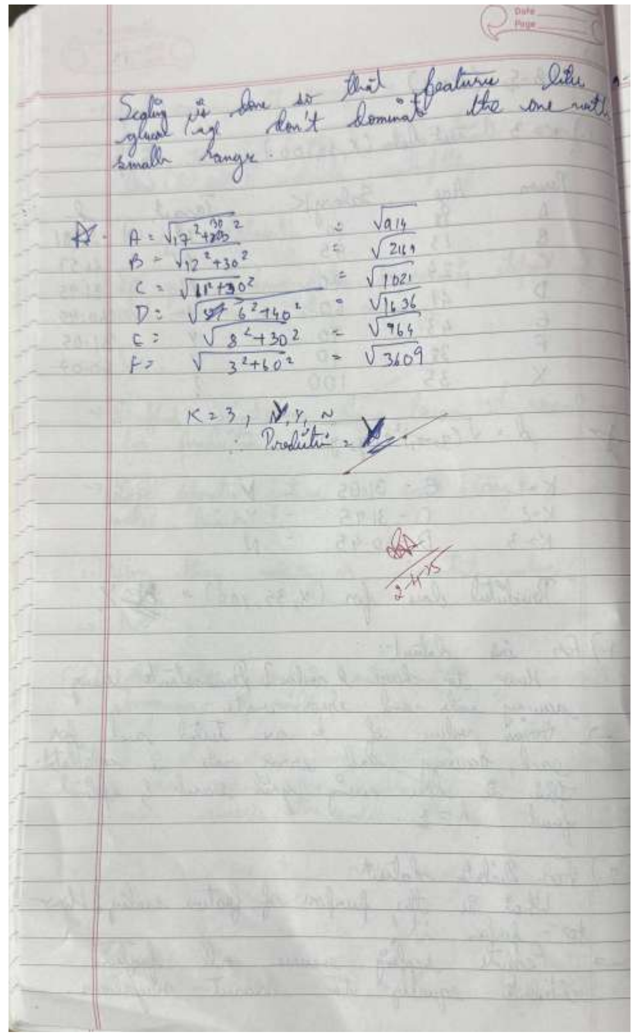
How to choose k value? Demonstrate using accuracy rate and error rate.

→ Various values of k are tested and for each, accuracy and error rate is calculated. This is done using grid search & optimal found k = 3.

2.) For Diabetes dataset:

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

→ Feature scaling ensures all features contribute equally to nearest neighbours.



Scaling is done so that features like glucose (age) don't dominate the one with smaller range.

A = $\sqrt{17^2 + 30^2}$ = $\sqrt{914}$

B = $\sqrt{12^2 + 30^2}$ = $\sqrt{244}$

C = $\sqrt{11^2 + 30^2}$ = $\sqrt{1021}$

D = $\sqrt{6^2 + 40^2}$ = $\sqrt{1636}$

E = $\sqrt{8^2 + 30^2}$ = $\sqrt{964}$

F = $\sqrt{3^2 + 60^2}$ = $\sqrt{3609}$

K = 3, Y, Y, N

∴ Prediction = Y

2-4-25

Code:

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

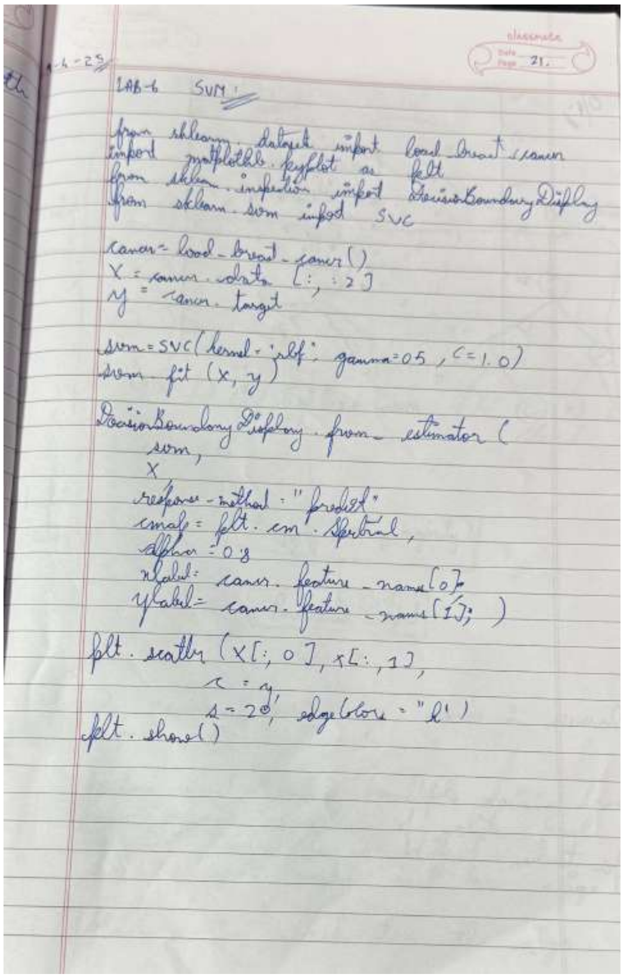
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# 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')
```

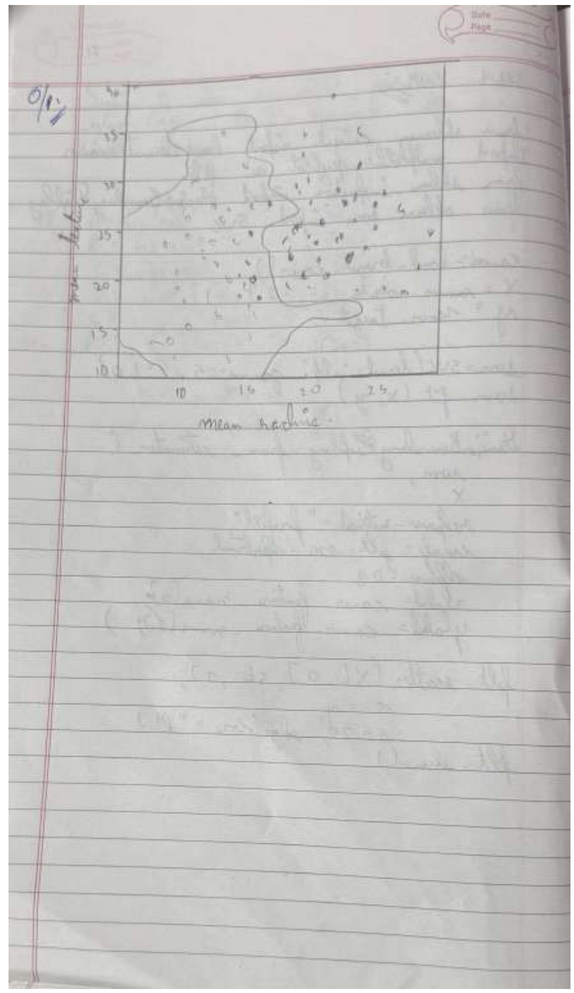
```python
# Define features and target
X = df.drop(columns=['target'])  # Assuming 'target' is the classification column
y = df['target']

# 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



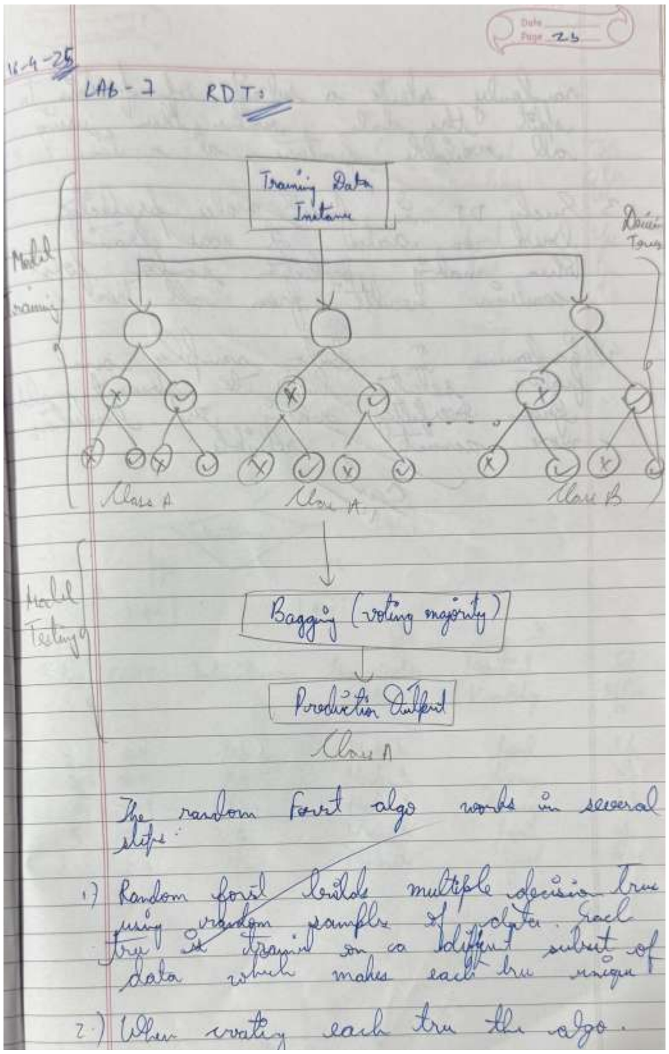


Code:

```
# 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

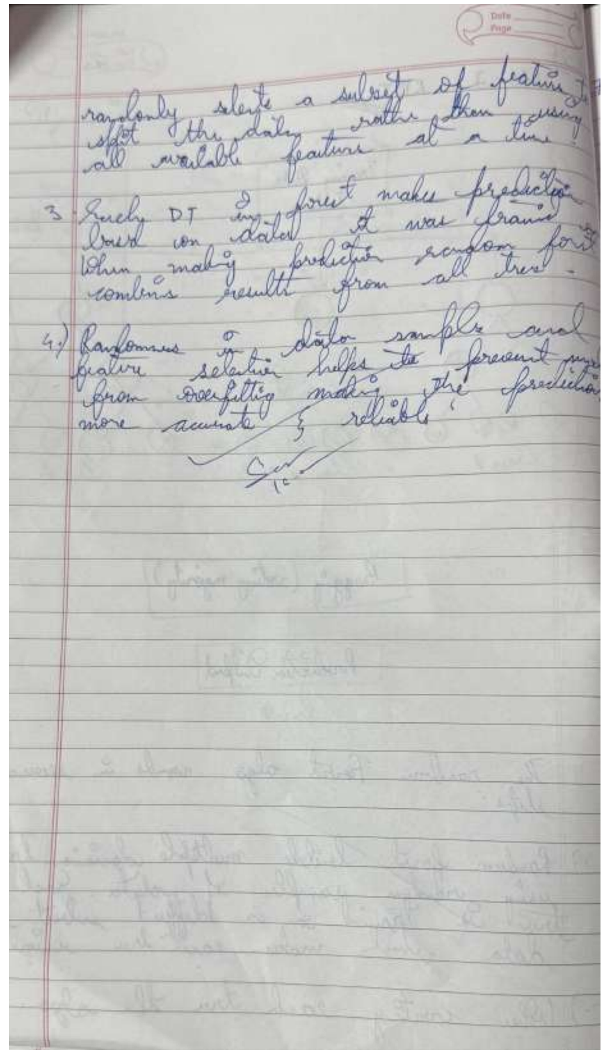

16-4-25

LAB - 7 RDT:

The random forest algo works in several steps:

1) Random forest builds multiple decision trees using random samples of data. Each tree is trained on a different subset of data which makes each tree unique.

2) When creating each tree the algo.


randomly selects a subset of features to split the data rather than using all available features at a time.

3. Each DT in forest makes predictions based on data it was trained on. When making prediction random forest combines results from all trees.

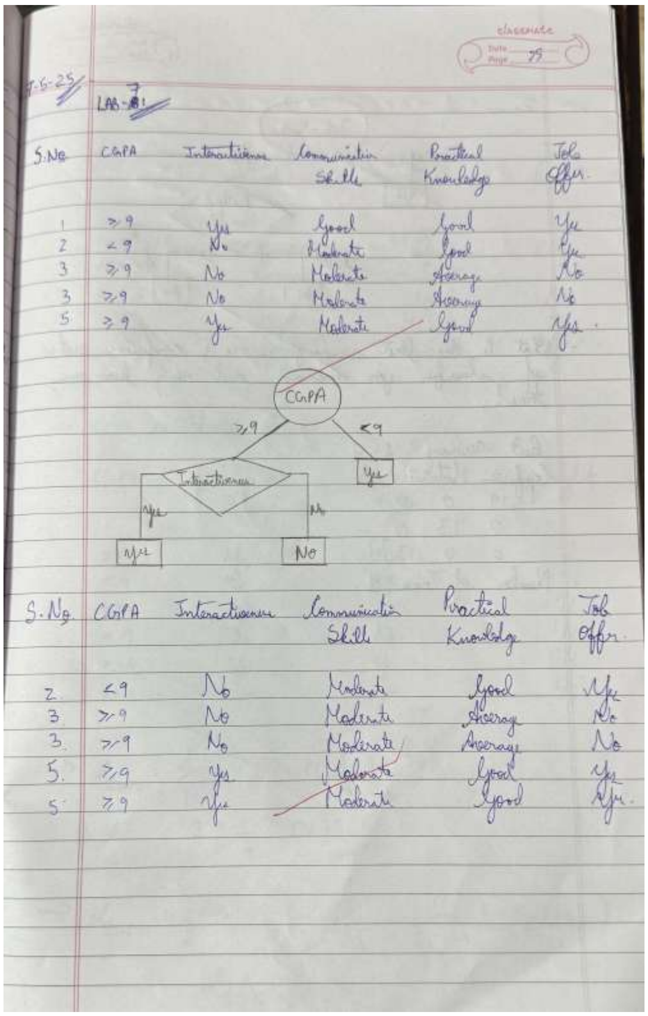
4) Randomness in data samples and feature selection helps to prevent model from overfitting making the predictions more accurate & reliable.

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

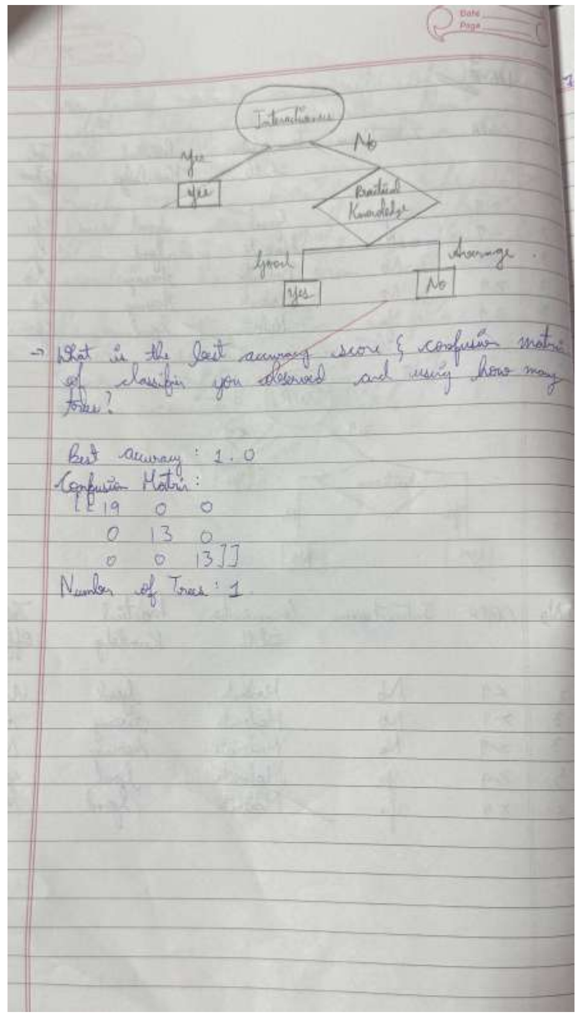


7-5-25

LAB-7

S.No	CGPA	Interactiveness	Communication Skills	Practical Knowledge	Job Offer
1	≥ 9	Yes	Good	Good	Yes
2	< 9	No	Moderate	Good	Yes
3	≥ 9	No	Moderate	Average	No
3	≥ 9	No	Moderate	Average	No
5	≥ 9	Yes	Moderate	Good	Yes

S.No	CGPA	Interactiveness	Communication Skills	Practical Knowledge	Job Offer
2	< 9	No	Moderate	Good	Yes
3	≥ 9	No	Moderate	Average	No
3	≥ 9	No	Moderate	Average	No
5	≥ 9	Yes	Moderate	Good	Yes
5	≥ 9	Yes	Moderate	Good	Yes



→ What is the best accuracy score & confusion matrix of classifier you observed and using how many trees?

Best accuracy : 1.0

Confusion Matrix :

[[19 0 0
0 13 0
0 0 13]]

Number of Trees : 1

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)]
```

```python
# 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

25-25

LAB-9: K-Means

Record Number	A	B
R_1	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: $d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

Record	Distance to $C_1(1,1)$	$d_2(5,7)$	Assigned ~~distance~~ cluster
R_1	0.00	7.21	C_1
R_2	1.12	6.10	C_1
R_3	3.61	4.21	C_1
R_4	7.21	0.00	C_2
R_5	5.00	2.50	C_2
R_6	5.37	2.84	C_2
R_7	4.61	2.92	C_2

→ Update Cluster Centroid:

Cluster C_1: R_1, R_2, R_3
Mean A $= (1.0 + 1.5 + 3.0)/3 = 1.83$
Mean B $= (1.0 + 2.0 + 4.0)/3 = 2.33$
~~Mean C~~
New C_1 = (1.83, 2.33)

Cluster C_2: R_4, R_5, R_6, R_7
Mean A $= (5.0 + 3.5 + 4.5 + 3.5)/4 = 4.125$
Mean B $= (7.0 + 5.0 + 5.0 + 4.5)/4 = 5.375$
New C_2 = (4.13, 5.38)

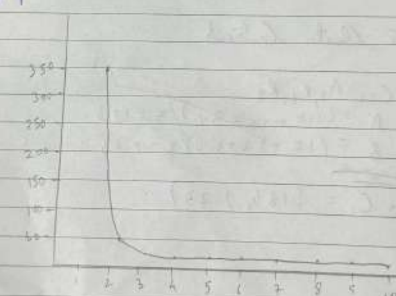
→ Iteration 2:

Record	d to C_1 (1.83, 2.33)	d to C_2 (4.13, 5.38)	Assigned
R_1	1.57	5.71	C_1
R_2	0.47	4.70	C_1
R_3	1.97	1.63	C_2
R_4	3.18	1.84	C_2
R_5	2.93	6.20	C_2
R_6	3.36	0.47	C_2
R_7	2.50	0.79	C_2

→ Final Cluster:

Cluster 1: R_1, R_2
Cluster 2: R_3, R_4, R_5, R_6, R_7

→ Graph:



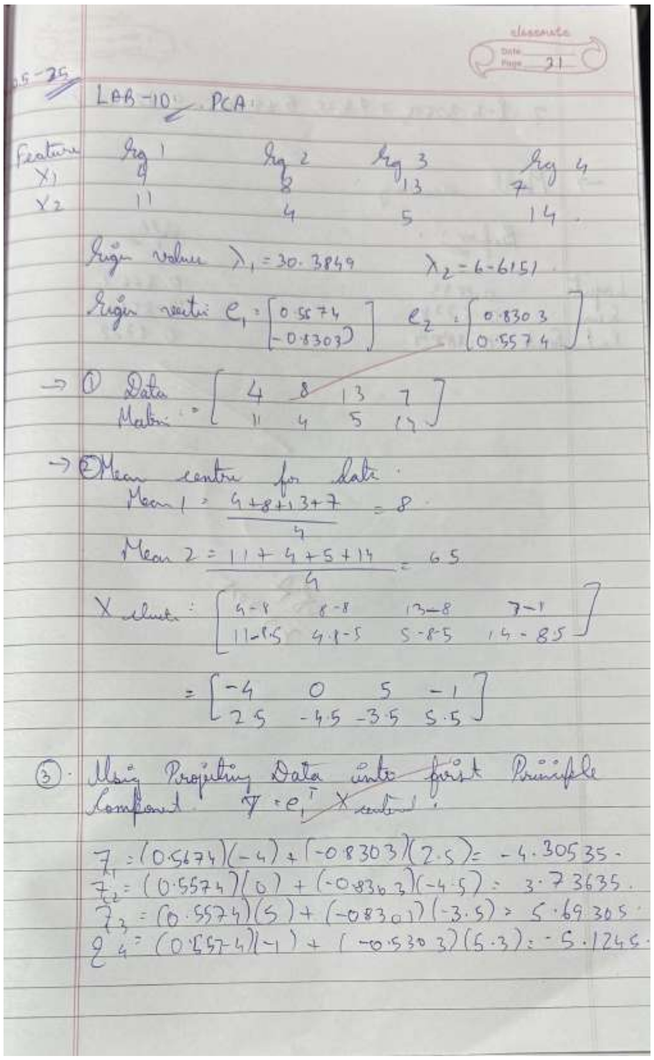
Code:

```
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



LAB-10 PCA

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

Eigen value $\lambda_1 = 30.3849$ $\lambda_2 = 6.6151$

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

→ ① Data Matrix = $\begin{bmatrix} 4 & 8 & 13 & 7 \\ 11 & 4 & 5 & 14 \end{bmatrix}$

→ ② Mean centre for data.

Mean 1 = $\frac{4+8+13+7}{4} = 8$

Mean 2 = $\frac{11+4+5+14}{4} = 6.5$

X centre: $\begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.5 & 4-8.5 & 5-8.5 & 14-8.5 \end{bmatrix}$

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

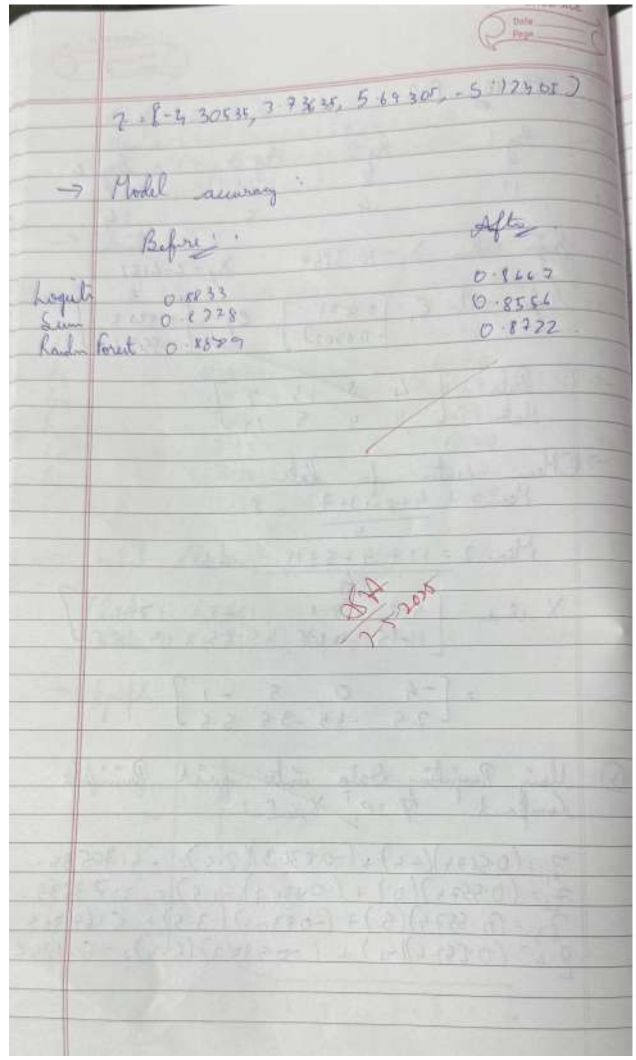
③ Using Projecting Data into first Principle Component $Z = e_1^T X_{centred}$.

$Z_1 = (0.5674)(-4) + (-0.8303)(2.5) = -4.30535$.

$Z_2 = (0.5574)(0) + (-0.8303)(-4.5) = 3.73635$.

$Z_3 = (0.5574)(5) + (-0.8303)(-3.5) = 5.69305$.

$Z_4 = (0.5574)(-1) + (-0.5303)(5.3) = -5.1245$.



$Z = [-4.30535, 3.73635, 5.69305, -5.12405]$

→ Model accuracy:

	Before	After
Logistic	0.8833	0.8667
Svm	0.8778	0.8556
Random Forest	0.8889	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}")
```