VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

SHASHANK SP (1BM22CS256)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

Sep-2024 to Jan-2025

B.M.S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **SHASHANK SP (1BM22CS256),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

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

Index

Github Link:

https://github.com/shashanksp2003/machine-learning-1BM22CS256

Program 1

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

Screenshot:

Lab-0

method-1

```
Import pandas as pd
data={
    'name' :['Alice', 'Bob', 'charlie', 'David']
    'usn' : [1,2,3,4]
    'make', [94,92,99,95]
    }
df = pd.Dataframe (data)
print(data head ())
```

method-2

```
diabetes = load_diabetes ()
df = pd. Dataframe ( diabetes )
print (df. head ())
```

method=3

```
df =pd.read_csv ('/content/sample-sales.csv')
df.head()
```

method-4

```
df = pd.read_csv ('/content/Dataset of diabetes.csv')
df.head ()
```

Code:

```
import pandas as pd

data = {

    'Name': ['Alice', 'Bob', 'Charlie'],

    'Age': [25, 30, 22],
```

'Department': ['HR', 'Finance', 'IT']

}

df = pd.DataFrame(data) df.to_csv('data.csv',

index=False) print("Sample data exported to

'data.csv'.")

imported_df = pd.read_csv('data.csv') print("\nImported

Data from 'data.csv':") print(imported_df)

imported_df['Age'] = imported_df['Age'] + 1

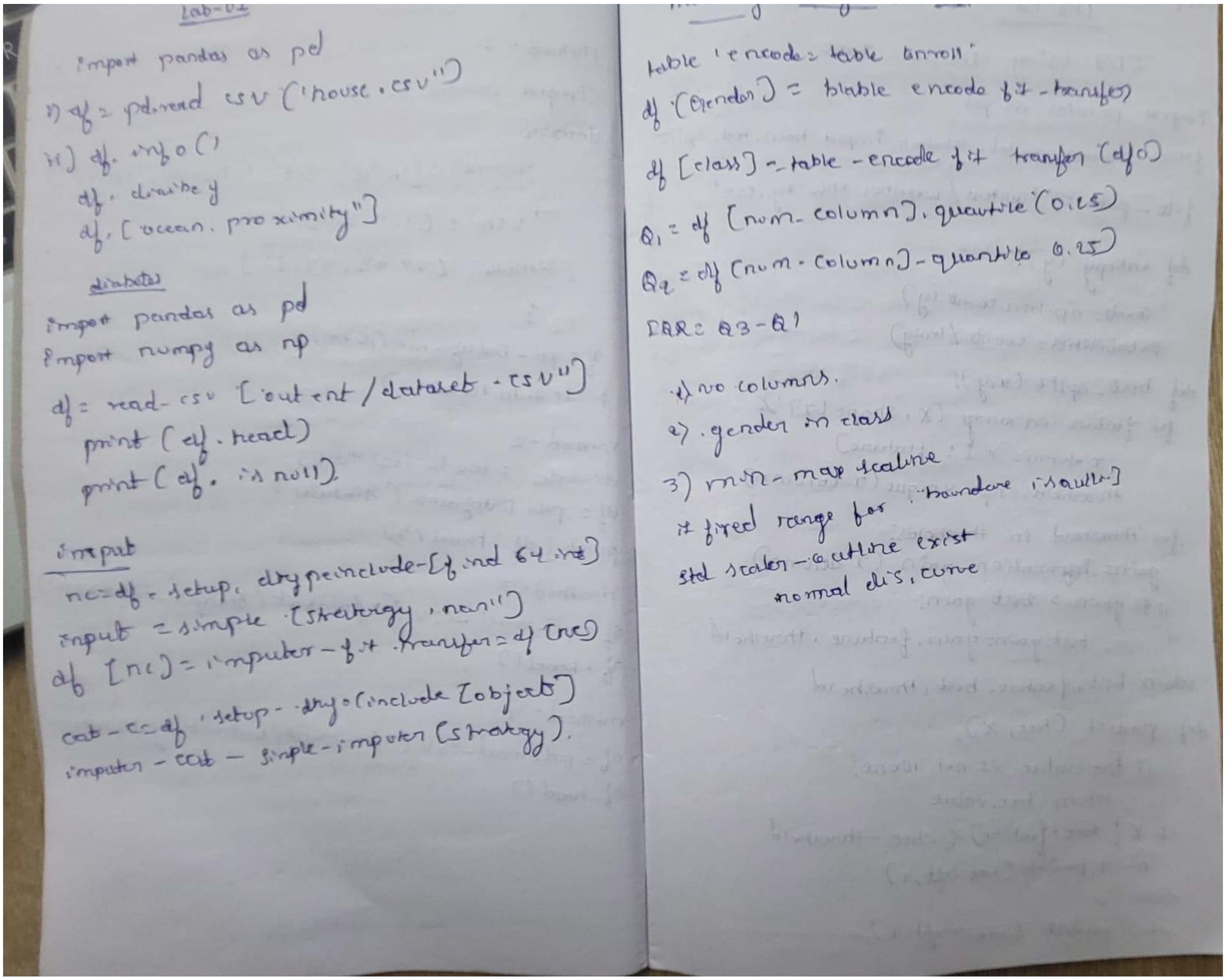
imported_df.to_csv('updated_data.csv', index=False)

print("\nUpdated data exported to 'updated_data.csv'.")

Program 2

Demonstrate various data pre-processing techniques for a given dataset.

Screenshot:



Lab-02

import pandas as pd
i) df = pd.read csv ("house.csv")
ii) df. info()
df. describe()
df. ["ocean. proximity"]

diabetes

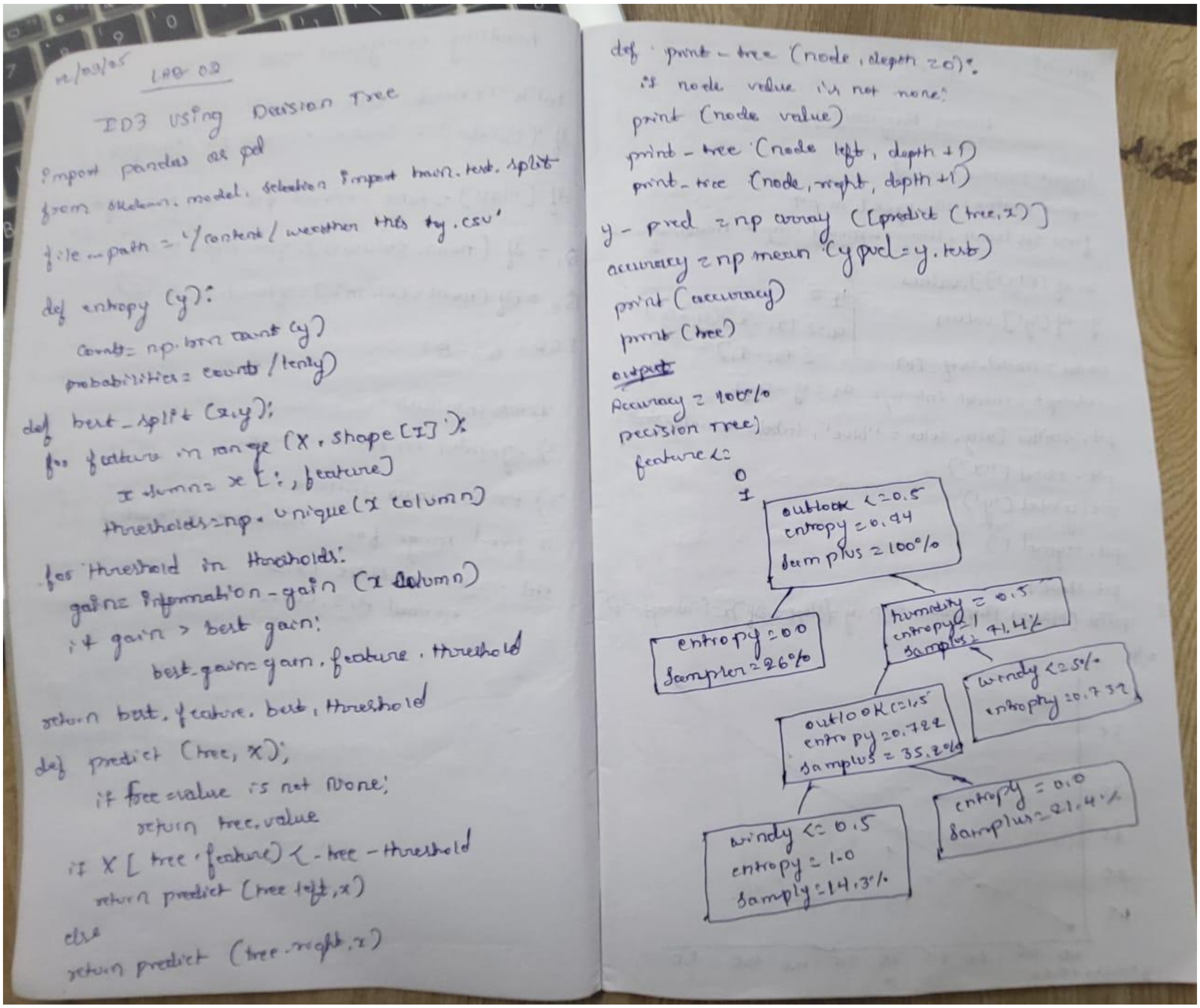
import pandas as pd
import numpy as np
df = read-csv ["out ent / dataset . csv"]
print (df. head)
print (df. is null)

input

nc=df. setup. dtype include-[float 64 int]
input = simple (strategy, nan")
df [nc] = imputer - fit . transfer = df (nc)
cat - c = df . setup - dtype (include [object]
imputer - cat - simple - imputer (strategy).

table 'encode= table enroll"
df ('Gender') = blable encode fit - transfer
df [class] = table - encode fit transfer (df())
Q1 = df [num- column]. quantile (0.25)
Q2 = df [num - column] - quantile 0.25)
IQR = Q3 - Q1

1) no columns.
2) . gender in class
3) min - max scaling
it fixed range for boundary issue]
std scaler - outline exist
normal dis, curve



Code:

import pandas as pd

df = pd.read_csv('/content/Dataset_of_Diabetes .csv')

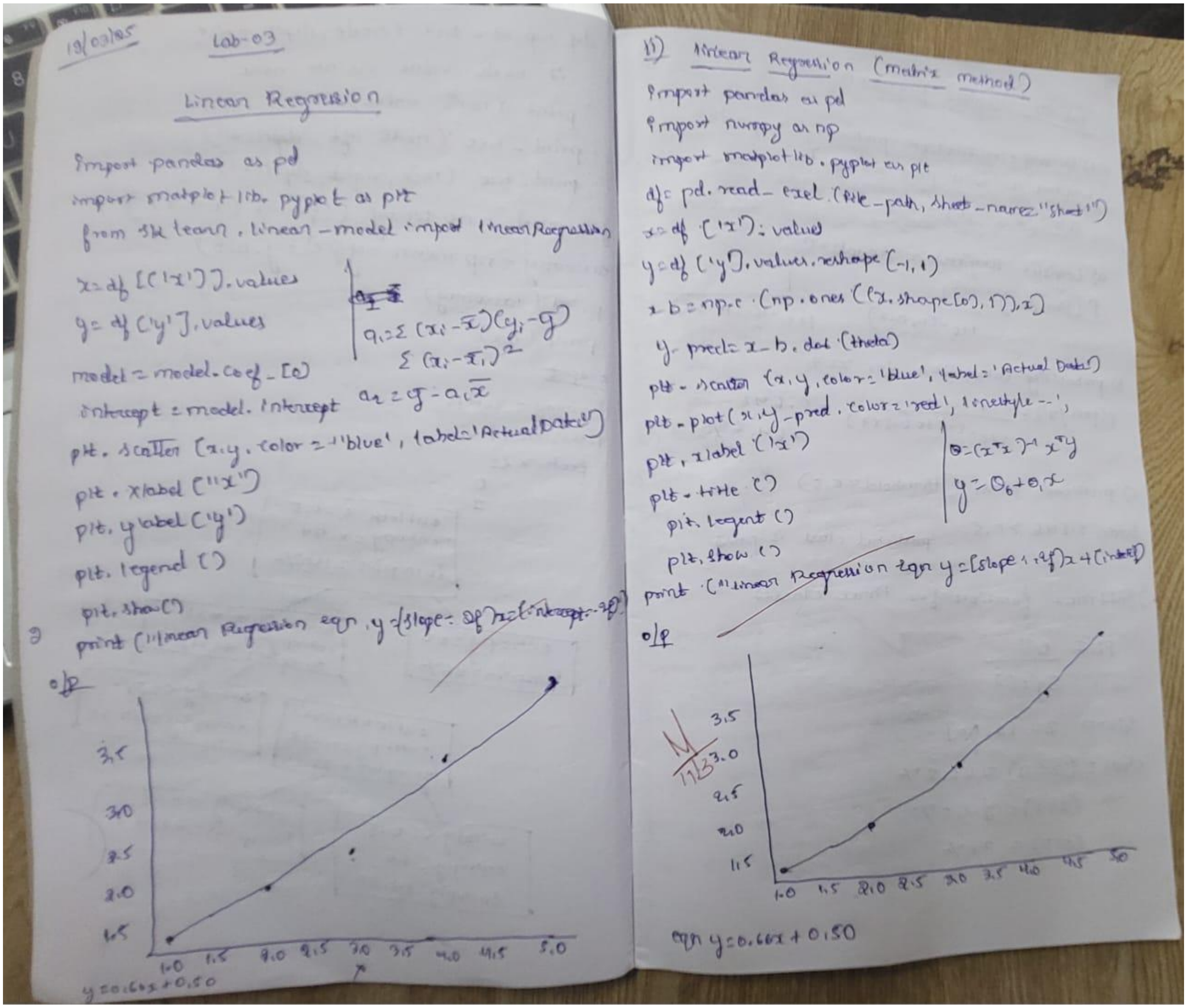
print(df.head())

```
df.isnull.sum()

from sklearn.preprocessing import OneHotEncoder categorical_cols =

df.select_dtypes(include=['object']).columns print("Categorical columns:",

categorical_cols)

encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore') encoded_data

= encoder.fit_transform(df[categorical_cols])

encoded_df = pd.DataFrame(encoded_data,
columns=encoder.get_feature_names_out(categorical_cols))

df = pd.concat([df, encoded_df], axis=1)

df.drop(categorical_cols, axis=1, inplace=True) df.head()

Q1 = df['AGE'].quantile(0.25)  Q3 =

df['AGE'].quantile(0.75)

print(Q1,Q3)

IQR = Q3 - Q1

lower_bound   =   Q1   -   1.5   *   IQR

upper_bound   =   Q3   +   1.5   *   IQR

print(lower_bound,upper_bound)

outliers = df[(df['AGE'minmax_scaler = MinMaxScaler() standard_scaler =

StandardScaler()

numerical_features = ['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']

df[numerical_features] = minmax_scaler.fit_transform(df[numerical_features]) df[numerical_features] =

standard_scaler.fit_transform(df[numerical_features])

print(df)] < lower_bound) | (df['AGE'] > upper_bound)]

print(outliers['AGE'])
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:



19/03/25 Lab-03

Linear Regression

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
x = df[['x']].values
y = df['y'].values
model = model.coef_[0]
intercept = model.intercept_
plt.scatter(x, y, color='blue', label='Actual Data')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
print("linear Regression eqn, y={slope:.2f}x+{intercept:.2f}")
```

$$a_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \qquad a_2 = \bar{y} - a_1\bar{x}$$

o/p

$y = 0.66x + 0.50$

ii) Linear Regression (matrix method)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_excel(file_path, sheet_name="Sheet1")
x = df['x'].values
y = df['y'].values.reshape(-1,1)
x_b = np.c_[np.ones((x.shape[0],1)), x]
y_pred = x_b.dot(theta)
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x, y_pred, color='red', linestyle='--',
plt.xlabel('x')
plt.title()
plt.legend()
plt.show()
print("Linear Regression eqn y={slope:.2f}x+{intercept}")
```

$$\theta = (X^T X)^{-1} X^T y \qquad y = \theta_0 + \theta_1 x$$

o/p

eqn $y = 0.66x + 0.50$

lab-04 Logistic Regression

1) Binary classification problem to predict whether a will pass or fail

$a_0 = -5$ $a_1 = 0.8$

a) Logistic Regression equation

$$P(\text{pass}/x) = \frac{1}{1+e^{-(a_0 + a_1 x)}} = \frac{1}{1+e^{-1(-5+0.8x)}}$$

b) probability calculation for $x=7$

$$p(\text{pass}/7) = \frac{1}{1+e^{-(-5+0.8\times7)}} = 0.646$$
$$= 64.6\%$$

c) predicted class (threshold $= 0.5$)

Since $0.646 > 0.5$ predicted class is pass

) Softmax function for three classes

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Given $z = [2, 1, 0]$

Class I $(z=2) = 66.5\%$

$(z=1) = 24.5\%$

$(z=0) = 9.0\%$

1) For "HR comma - sep.csv"

i) Variables impacting employee selection

→ Satisfaction level: low satisfication, high chance of leaving

→ Average monthly hours: Extreme working hours, more likely to leave;

→ Salary level: Low salary, high chances of exiting

ii) Model Accuracy & evaluation

→ Accuracy: 78.8%

2) Zoo Dataset

i) Data preprocessing steps:

→ Removed animal-name attribute since it was not relevant for classification

→ Standardized numeric features standard scaler

ii) Handling missing & inconsistent values

→ no missing values found

iii) The confusion matrix provided insights into which classes were predicted correctly & which one were classified

iv) Some similar classes were classified wrong due to overlapping characteristics

Code:

```
import pandas as pd

import numpy as np

from sklearn import linear_model import

matplotlib.pyplot as plt

df = pd.read_csv('/content/housing_area_price.csv')

plt.xlabel('area')

plt.ylabel('price')

plt.scatter(df.area,df.price,color='red',marker='+')

new_df = df.drop('price',axis='columns') new_df

price = df.price

price

reg = linear_model.LinearRegression()

reg.fit(new_df,price) reg.predict([[3300]])

reg.coef_ reg.intercept_

3300*135.78767123 + 180616.43835616432

reg.predict([[5000]])
```

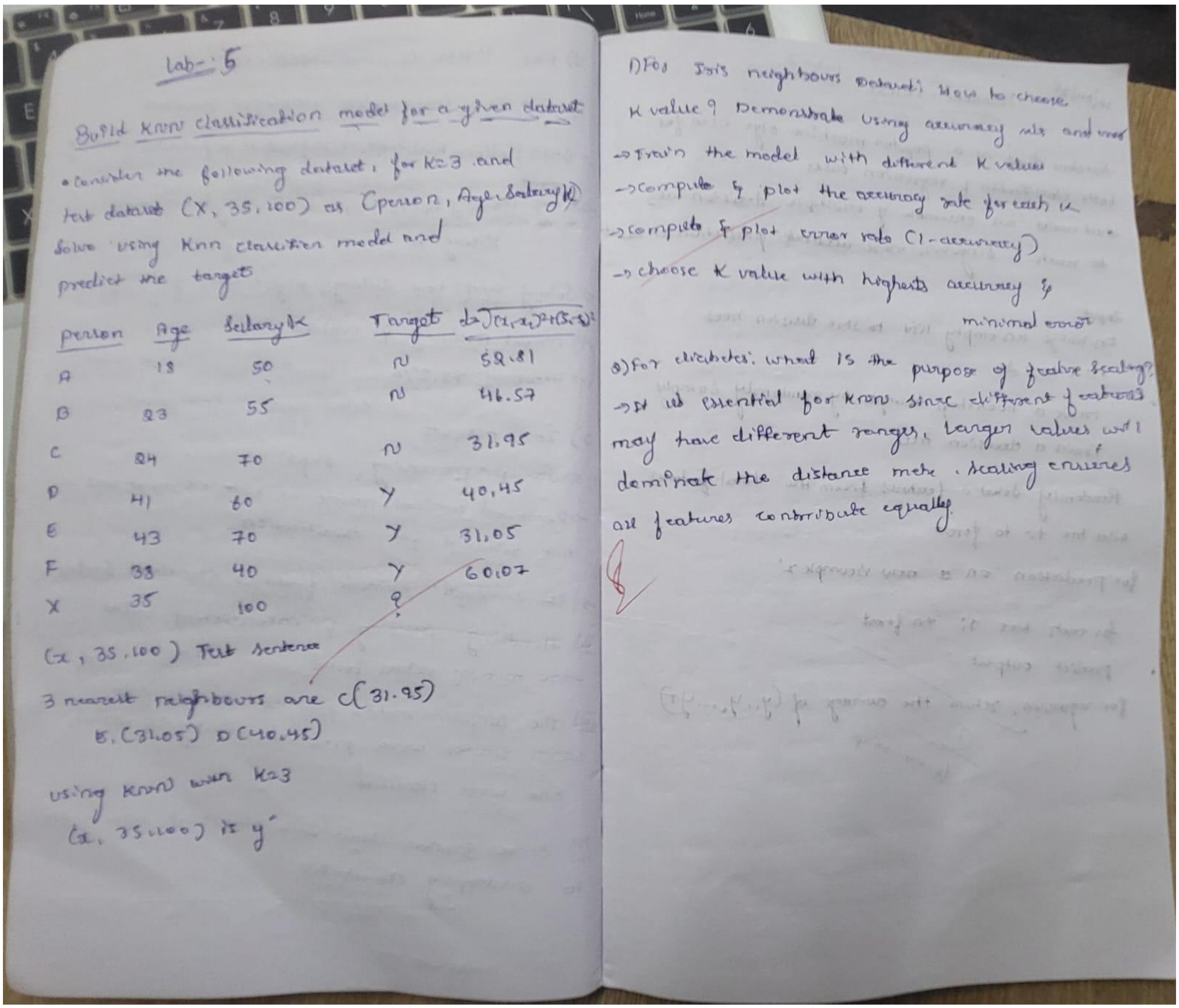
```
df = pd.read_csv('/content/canada_per_capita_income.csv') new_df =
df.drop('per_capita_income',axis='columns')
reg = linear_model.LinearRegression() per_capita_income
= y = df['per_capita_income'].values
reg.fit(new_df,per_capita_income)
print(reg.coef_)
print(reg.intercept_)
predicted_income = reg.predict([[2020]])
print("predicted Income in the year 2020:" , predicted_income)
plt.scatter(df['year'], per_capita_income, color='blue', label='Data Points')
plt.plot(df['year'], reg.predict(new_df), color='red', label='Regression Line')
plt.xlabel('Year')
plt.ylabel('Per Capita Income (US$)') plt.title('Regression
Line: Per Capita Income vs Year') plt.legend()
plt.show()
df = pd.read_csv('/content/salary.csv')
df.YearsExperience.median()
df.YearsExperience = df.YearsExperience.fillna(df.YearsExperience.median()) reg
= linear_model.LinearRegression()
reg.fit(df.drop('Salary',axis='columns'),df.Salary)
print(reg.coef_)
print(reg.intercept_)
print("Predicted Salary of Person with 12 years of Experience: ",reg.predict([[12]]))
```

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

{

        'one':1,'two':2,'three':3,'four':4,'five':5,'six':6,'seven':7,'eight':8,'nine':9,'ten':10,'eleven':11,'twelve':12

         }

experience_map = df['experience'] = df['experience'].map(experience_map)

df.test_score = df.test_score.fillna(df.test_score.median())

df.experience = df.experience.fillna(df.experience.median()) reg

= linear_model.LinearRegression()

reg.fit(df.drop('salary',axis='columns'),df.salary) print(reg.coef_)

print(reg.intercept_)

print("Predicted Salary of Person with 2 years of Experience, 9 test score, 6 interview score:
",reg.predict([[2, 9, 6]]))

print("Predicted Salary of Person with 12 years of Experience, 10 test score, 10 interview score:
",reg.predict([[12, 10, 10]]))

df = pd.read_csv('/content/1000_Companies.csv')

experience_map = {

         'New York':1,'California':2,'Florida':3

         }

experience_map = df['State'] = df['State'].map(experience_map) reg

= linear_model.LinearRegression()

reg.fit(df.drop('Profit',axis='columns'),df.Profit)

print(reg.coef_)

print(reg.intercept_)

print(reg.predict([[91694.48, 515841.3, 11931.24,3]]))
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:


Lab-5

Build KNN classification model for a given dataset

• Consider the following dataset, for K=3 and test dataset (X, 35, 100) as (person, Age, Salary(K)) solve using Knn classifier model and predict the target

person	Age	Salary K	Target	$d=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$
A	18	50	N	52.81
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	33	40	Y	60.07
X	35	100	?	

(x, 35, 100) Test sentence

3 nearest neighbours are C(31.95) E.(31.05) D(40.45)

using KNN with K=3
(x, 35, 100) is y

1) For Iris neighbours dataset: How to choose K value? Demonstrate using accuracy rate and error

-> Train the model with different K values
-> Compute & plot the accuracy rate for each K
-> Compute & plot error rate (1-accuracy)
-> choose K value with highest accuracy & minimal error

2) For diabetes: what is the purpose of feature scaling?

-> It is essential for KNN since different features may have different ranges, larger values will dominate the distance metric. Scaling ensures all features contribute equally.

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 OneHotEncoder, StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix #

Load dataset
```

```
file_path = "/content/HR_comma_sep.csv" df =
pd.read_csv(file_path)
# Exploratory Data Analysis (EDA)
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show() plt.figure(figsize=(6,4))
sns.countplot(x="left", data=df, palette="Set2")
plt.title("Employee Retention Distribution")
plt.xlabel("Left Company (1 = Yes, 0 = No)")
plt.ylabel("Count")
plt.show()
```

```
# Impact of salary on employee retention

plt.figure(figsize=(8,5))

sns.countplot(x="salary", hue="left", data=df, palette="muted")

plt.title("Impact of Salary on Employee Retention")

plt.xlabel("Salary Level")

plt.ylabel("Count")

plt.legend(title="Left Company", labels=["Stayed", "Left"])

plt.show()

# Correlation between department and employee retention

plt.figure(figsize=(12,5))

sns.countplot(x="Department",   hue="left",   data=df,   palette="pastel")

plt.title("Correlation  between  Department  and  Employee  Retention")

plt.xlabel("Department")

plt.ylabel("Count") plt.xticks(rotation=45)

plt.legend(title="Left Company", labels=["Stayed", "Left"])

plt.show()

# Selecting important features

features = ["satisfaction_level", "time_spend_company", "number_project",
"average_montly_hours", "salary", "Department"]

X = df[features] y =

df["left"]

# One-hot encode categorical variables

X = pd.get_dummies(X, columns=["salary", "Department"], drop_first=True)
```

```
# Standardize numerical features

scaler = StandardScaler()

X.iloc[:, :4] = scaler.fit_transform(X.iloc[:, :4])

# Split dataset into training and testing sets (80-20 split)

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

Train logistic regression model

model = LogisticRegression()

model.fit(X_train, y_train)

# Predict and measure accuracy

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")

# Plot confusion matrix

plt.figure(figsize=(6,5))

sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Stayed", "Left"],
yticklabels=["Stayed", "Left"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()




zoo_data = pd.read_csv("/content/zoo-data.csv")
```

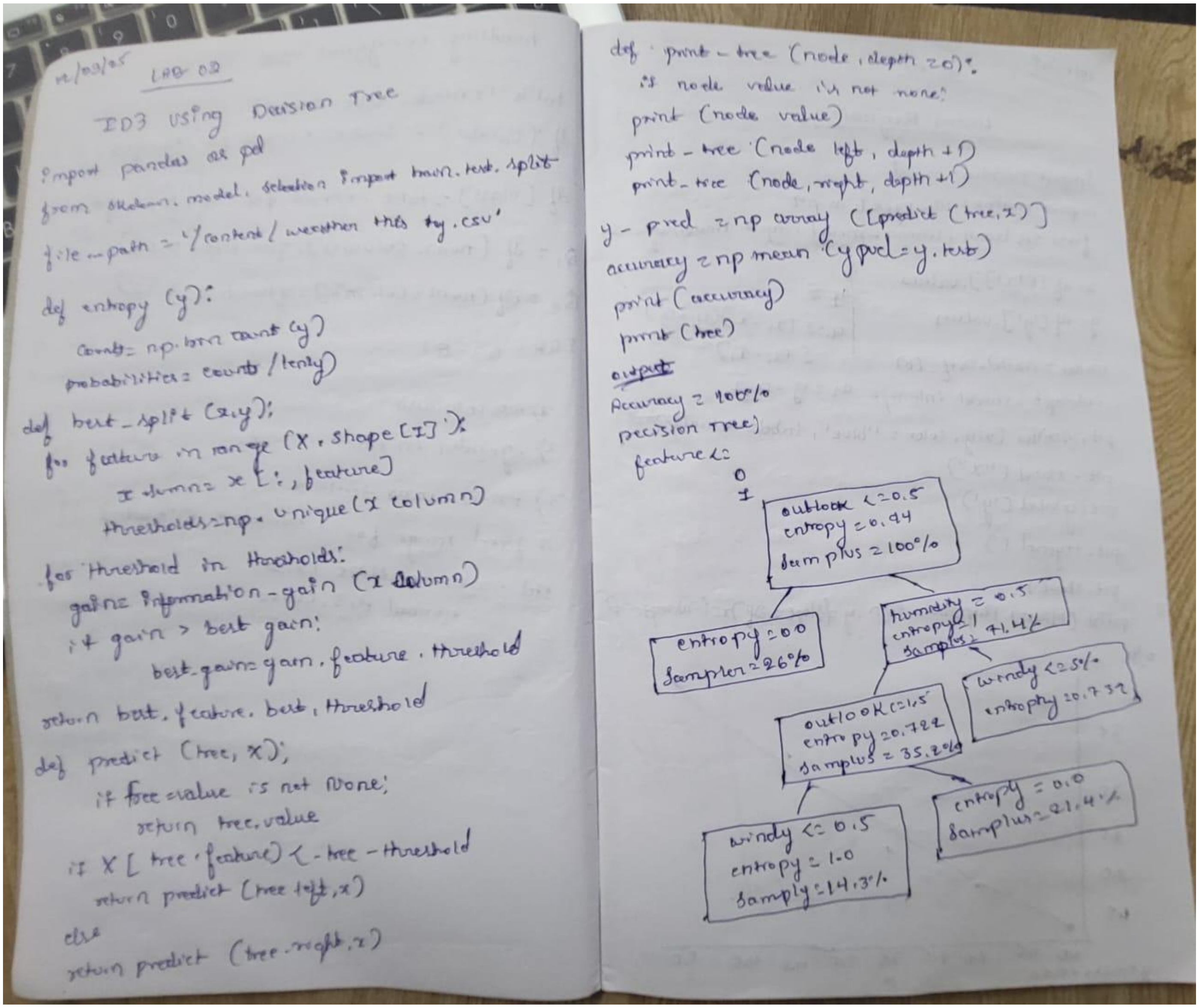
```
zoo_classes = pd.read_csv("/content/zoo-class-type.csv") #

Merge datasets on class_type if needed

    if 'class_type' in zoo_data.columns and 'class_type' in zoo_classes.columns:

        zoo_data = zoo_data.merge(zoo_classes, on='class_type', how='left')

# Separate features and target variable

X = zoo_data.drop(columns=['class_type', 'animal_name']) # Assuming 'animal_name' is
non-numeric

y = zoo_data['class_type']

# Split 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, stratify=y) #

Scale the features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Train Logistic Regression model

model = LogisticRegression(multi_class='ovr', solver='lbfgs', max_iter=200)

model.fit(X_train, y_train)

# Predictions

y_pred = model.predict(X_test) #

Evaluate accuracy

accuracy = accuracy_score(y_test, y_pred) print(f"Model

Accuracy: {accuracy:.2f}")
```

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred) plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y),
yticklabels=np.unique(y))

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("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:



11/03/25 LAB 02

ID3 using Decision Tree

```
import pandas as pd
from sklearn.model.selection import train.test.split
file_path = '/content/weather this ty.csv'

def entropy (y):
    counts= np.bin count (y)
    probabilities = counts / len(y)

def best_split (x,y):
for feature in range (X.shape[1]):
    x column= x [:, feature]
    thresholds=np.unique(x column)

for threshold in thresholds:
  gain= information_gain (x column)
  if gain > best gain:
       best_gain= gain, feature, threshold

return best, feature, best, threshold

def predict (tree, x):
    if tree value is not None;
      return tree.value
  if X [tree.feature) <-tree-threshold
    return predict (tree left, x)
  else
  return predict (tree.right, x)

def print_tree (node, depth =0):
  if node value is not none:
  print (node value)
  print_tree (node left, depth +1)
  print_tree (node, right, depth +1)

y-pred =np array ([predict (tree, x)])
accuracy =np mean (y pred= y.test)
print (accuracy)
print (tree)
```

output

Accuracy = 100%

Decision Tree)

feature <=

0

1

outlook <=0.5
entropy = 0.94
samplus = 100%

entropy = 0.0
samples = 26%

humidity = 0.5
entropy = 1
samples = 71.4%

windy <= 5%
entropy = 0.732

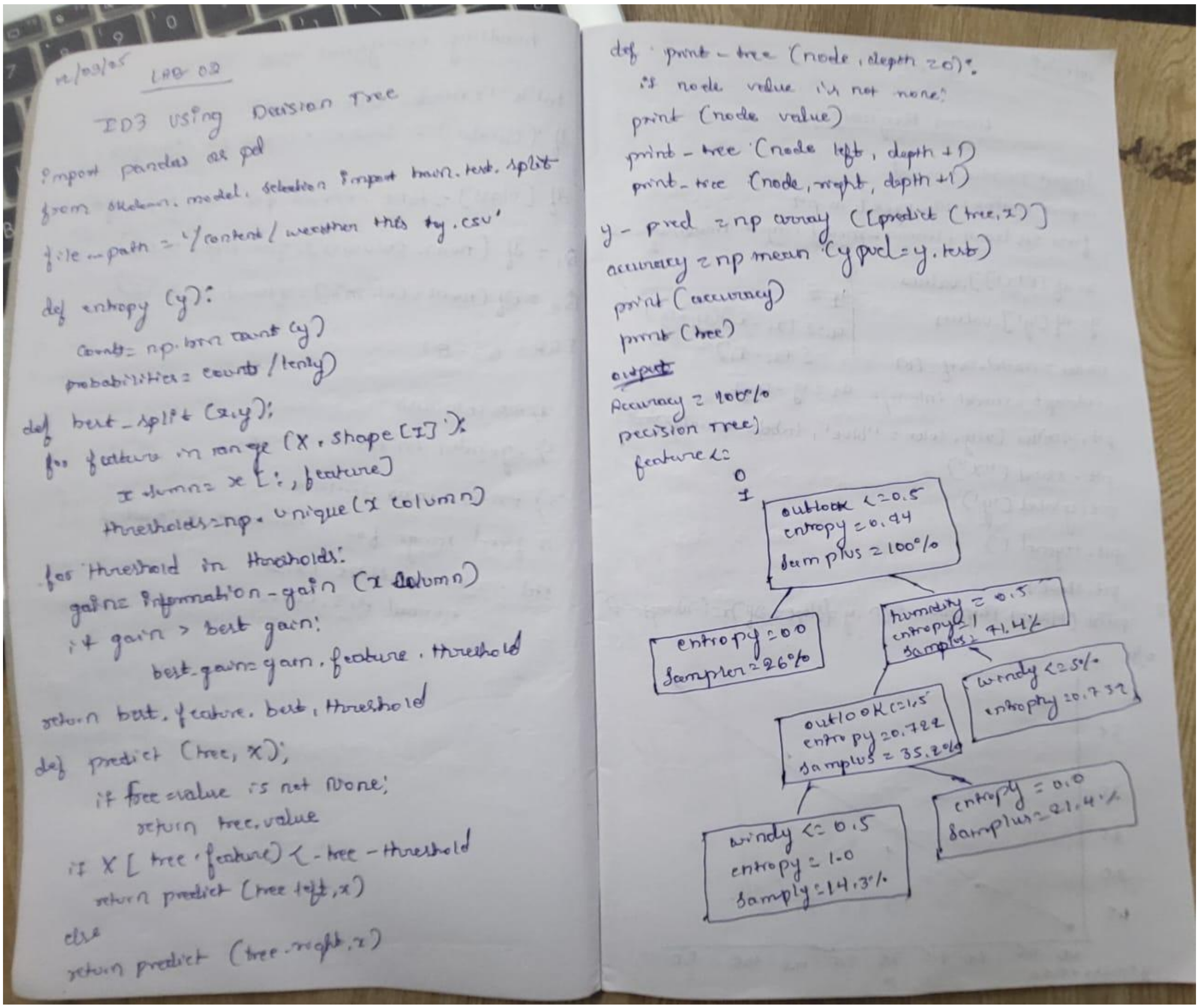
outlook <=1.5
entropy = 0.722
samplus = 35.2%

entropy = 0.0
samplus = 21.4%

windy <= 0.5
entropy = 1.0
samply = 14.3%

Code:

import pandas as pd



```
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

file_path = "/content/iris.csv"

df = pd.read_csv(file_path)

# Separate features and target

X = df.drop(columns=['species']) y =

df['species']

# Encode target labels

y = LabelEncoder().fit_transform(y)

# Split data into training (80%) and testing (20%)

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

Create and train the DecisionTree classifier

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

# Make predictions on the test set y_pred =

clf.predict(X_test)

# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
# Generate confusion matrix

cm = confusion_matrix(y_test, y_pred) #
```

Plot confusion matrix

plt.figure(figsize=(6,5))

sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=df['species'].unique(), yticklabels=df['species'].unique())

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

label_encoders = {}

for column in df.columns:

le = LabelEncoder()

df[column] = le.fit_transform(df[column])

label_encoders[column] = le

Split the dataset into features and target X =

df.drop('species', axis=1)

y = df['species']

Initialize the Decision Tree Classifier with entropy as the criterion clf =

DecisionTreeClassifier(criterion='entropy')

Train the classifier

clf.fit(X_train, y_train) #

Make predictions

y_pred = clf.predict(X_test) #

Evaluate the classifier

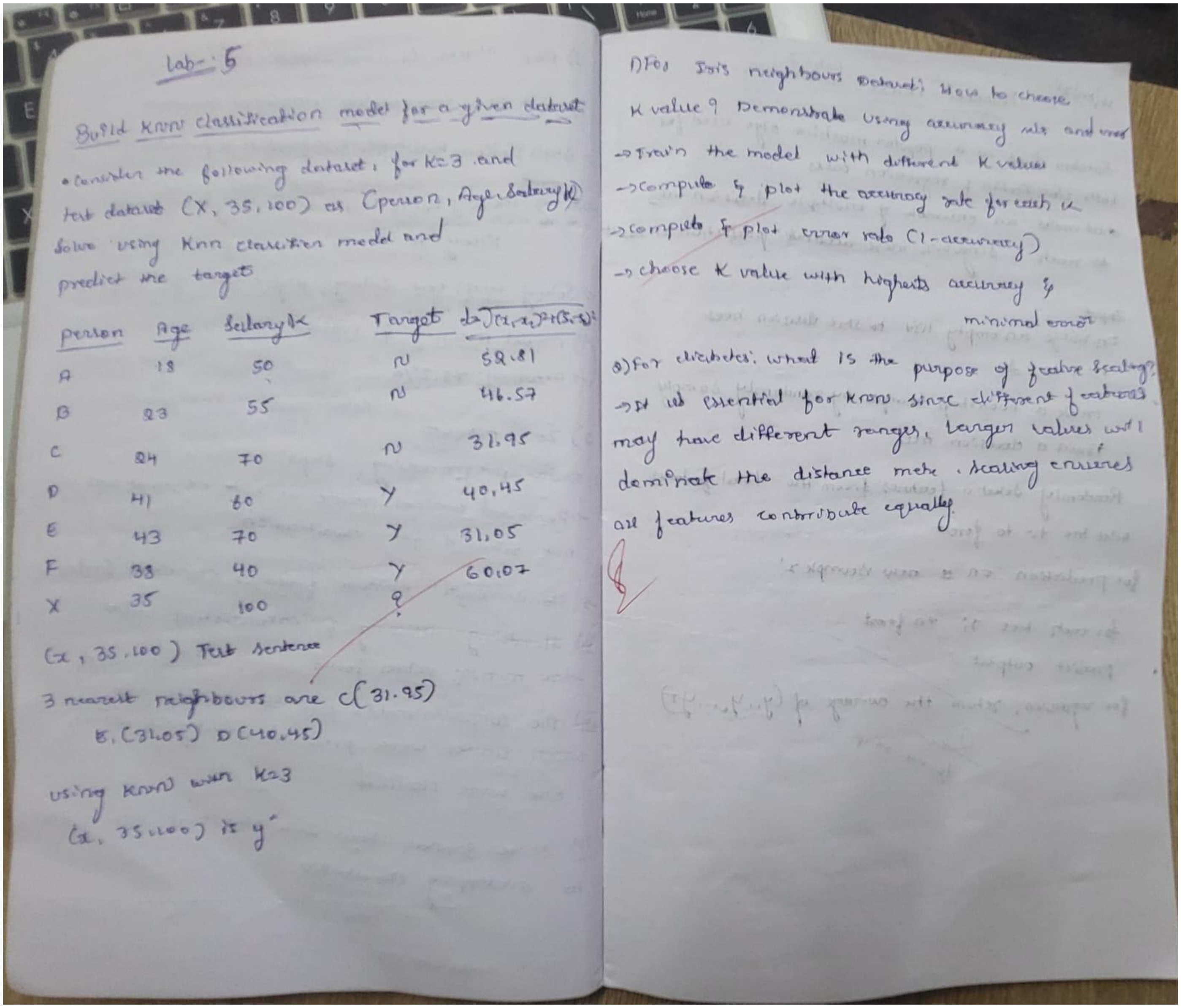
accuracy = accuracy_score(y_test, y_pred)

```
print(f'Accuracy: {accuracy:.2f}')

print(classification_report(y_test, y_pred, target_names=['Iris-setosa',
'Iris-versicolor','Iris-virginica']))

# Optionally, visualize the decision tree

from sklearn.tree import plot_tree import

matplotlib.pyplot as plt

plt.figure(figsize=(12,8))

plot_tree(clf, filled=True, feature_names=X.columns, class_names=['Setosa', 'Versicolor','Virginica'])

plt.show()

file_path = "/content/drug.csv" df =

pd.read_csv(file_path)

# Encode categorical features categorical_cols

= ['Sex', 'BP', 'Cholesterol']

df[categorical_cols] = df[categorical_cols].apply(LabelEncoder().fit_transform) #

Separate features and target

X = df.drop(columns=['Drug']) y =

df['Drug']

# Encode target labels

y = LabelEncoder().fit_transform(y)

# Split data into training (80%) and testing (20%)

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

Create and train the DecisionTree classifier

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

# Make predictions on the test set y_pred =
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot:



Code:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split
```

from sklearn.preprocessing import StandardScaler from

sklearn.neighbors import KNeighborsClassifier

Lab- 5

Build KNN classification model for a given dataset

• Consider the following dataset, for K=3 and test dataset (X, 35, 100) as (person, Age, Salary K) solve using Knn classifier model and predict the target

person	Age	Salary K	Target	$d = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$
A	18	50	N	52.81
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	?	

(x, 35, 100) Test sentence

3 nearest neighbours are C(31.95) E(31.05) D(40.45)

using Knn with K=3

(x, 35, 100) is y

1) For Iris neighbours Dataset: How to choose K value? Demonstrate using accuracy rate and error

→ Train the model with different K values

→ Compute & plot the accuracy rate for each K

→ Compute & plot error rate (1-accuracy)

→ choose K value with highest accuracy & minimal error

2) For diabetes: what is the purpose of feature scaling?

→ It is essential for Knn since different features may have different ranges. Larger values will dominate the distance metric. Scaling ensures all features contribute equally.

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score import

matplotlib.pyplot as plt

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

df.head()

# Separate features and labels X

= df.drop('species', axis=1) y =

df['species']

# Split the data into training and testing sets

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

Feature Scaling

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Find the best k value by plotting error rate

error_rate = []


for i in range(1, 31):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train, y_train)


pred_i = knn.predict(X_test)

 error_rate.append(np.mean(pred_i != y_test)) #

Plotting error rates
plt.figure(figsize=(12,6))

   plt.plot(range(1,31), error_rate, color='blue', linestyle='dashed', marker='o',
```

```
             markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate') plt.show()
# Choose k with minimum error
optimal_k = error_rate.index(min(error_rate)) + 1
print(f"Optimal K value: {optimal_k}")
# Train the model with optimal k
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
# Predict the test set results
y_pred = knn.predict(X_test) #
Evaluate the model
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
class_report = classification_report(y_test, y_pred)


acc_score = accuracy_score(y_test, y_pred)

print("\nClassification Report:\n", class_report)

print("\nAccuracy Score:", acc_score)


 diabetes_df = pd.read_csv('/content/diabetes.csv')

# Display first few rows

diabetes_df.head()

# Separate features and target

X = diabetes_df.drop('Outcome', axis=1)

# Assuming 'Outcome' is the target variable based on common diabetes datasets

y = diabetes_df['Outcome']

# 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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Finding the best k value

error_rate = []

for i in range(1, 31):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train, y_train)

    pred_i = knn.predict(X_test)
```

```
    error_rate.append(np.mean(pred_i != y_test))
# Plotting error rates
plt.figure(figsize=(12,6))
plt.plot(range(1,31), error_rate, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()
# Choose optimal k
optimal_k = error_rate.index(min(error_rate)) + 1
print(f"Optimal K value: {optimal_k}")
# Train the model with optimal k
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
# Predict the test set results
y_pred = knn.predict(X_test)
# Evaluate the model
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
plt.show()
acc_score = accuracy_score(y_test, y_pred)
print("\nAccuracy Score:", acc_score)


heart_df = pd.read_csv('/content/heart.csv')
# Display first few rows
heart_df.head()
# Separate features and target
X = heart_df.drop('target', axis=1)
y = heart_df['target']
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Find the best k value
error_rate = []
acc_scores = []
for i in range(1, 31):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    acc_scores.append(accuracy_score(y_test, pred_i))
    error_rate.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(12,6))
plt.plot(range(1,31), acc_scores, color='green', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
optimal_k = acc_scores.index(max(acc_scores)) + 1
print(f"Optimal K value: {optimal_k}")
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
# Predict the test set results
y_pred = knn.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)
acc_score = accuracy_score(y_test, y_pred)
print("\nAccuracy Score:", acc_score)
```

Program 7

Build Support vector machine model for a given dataset.

Code:

```
import pandas as pd import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns
```

```
import matplotlib.pyplot as plt

df1=pd.read_csv("/content/iris.csv")



print("Iris\n",df1.head())

X_iris = df1.drop('species', axis=1) y_iris

= df1['species']

X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2,
random_state=42)

# Linear Kernel SVM

svm_linear = SVC(kernel='linear', random_state=42)

svm_linear.fit(X_train_iris, y_train_iris)

# RBF Kernel SVM

svm_rbf = SVC(kernel='rbf', random_state=42)

svm_rbf.fit(X_train_iris, y_train_iris)

y_pred_linear = svm_linear.predict(X_test_iris)

y_pred_rbf = svm_rbf.predict(X_test_iris)

# Accuracy and Confusion Matrix for Linear Kernel accuracy_linear =

accuracy_score(y_test_iris, y_pred_linear) conf_matrix_linear =

confusion_matrix(y_test_iris, y_pred_linear)

# Accuracy and Confusion Matrix for RBF Kernel accuracy_rbf

= accuracy_score(y_test_iris, y_pred_rbf) conf_matrix_rbf =

confusion_matrix(y_test_iris, y_pred_rbf) # Display Results

print(f"Linear Kernel Accuracy: {accuracy_linear}")

print(f"RBF Kernel Accuracy: {accuracy_rbf}")

# Confusion Matrices
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(conf_matrix_linear, annot=True, fmt='d', cmap='Blues', ax=ax1)

ax1.set_title("Linear Kernel Confusion Matrix")
```

```
ax1.set_xlabel('Predicted')

ax1.set_ylabel('Actual')

sns.heatmap(conf_matrix_rbf, annot=True, fmt='d', cmap='Blues', ax=ax2)

ax2.set_title("RBF Kernel Confusion Matrix")

ax2.set_xlabel('Predicted')

ax2.set_ylabel('Actual') plt.show()


df2=pd.read_csv("/content/letter-recognition.csv")

print("Letter-Recognition\n",df2.head())

X_letter = df2.drop('letter', axis=1) y_letter

= df2['letter']

y_letter = y_letter.astype('category').cat.codes

X_train_letter, X_test_letter, y_train_letter, y_test_letter = train_test_split(X_letter, y_letter,
test_size=0.2, random_state=42)

# Linear Kernel SVM for Letter Recognition

svm_linear_letter = SVC(kernel='linear', random_state=42, probability=True)

svm_linear_letter.fit(X_train_letter, y_train_letter)

# RBF Kernel SVM for Letter Recognition

svm_rbf_letter = SVC(kernel='rbf', random_state=42, probability=True)

svm_rbf_letter.fit(X_train_letter, y_train_letter)

y_pred_linear_letter = svm_linear_letter.predict(X_test_letter) y_pred_rbf_letter

= svm_rbf_letter.predict(X_test_letter) accuracy_linear_letter =

accuracy_score(y_test_letter, y_pred_linear_letter)

conf_matrix_linear_letter = confusion_matrix(y_test_letter, y_pred_linear_letter) accuracy_rbf_letter

= accuracy_score(y_test_letter, y_pred_rbf_letter)
```

conf_matrix_rbf_letter = confusion_matrix(y_test_letter, y_pred_rbf_letter) print(f"Linear Kernel Accuracy (Letter-recognition): {accuracy_linear_letter}") print(f"RBF Kernel Accuracy (Letter-recognition): {accuracy_rbf_letter}")

Confusion Matrices

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(25, 12)) sns.heatmap(conf_matrix_linear_letter, annot=True, fmt='d', cmap='Blues', ax=ax1) ax1.set_title("Linear Kernel Confusion Matrix")

ax1.set_xlabel('Predicted')

ax1.set_ylabel('Actual')

sns.heatmap(conf_matrix_rbf_letter, annot=True, fmt='d', cmap='Blues', ax=ax2) ax2.set_title("RBF Kernel Confusion Matrix")

ax2.set_xlabel('Predicted')

ax2.set_ylabel('Actual')

plt.show()

Plotting ROC curve for Linear Kernel

fpr, tpr, thresholds = roc_curve(y_test_letter, svm_linear_letter.predict_proba(X_test_letter)[:, 1], pos_label=1)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

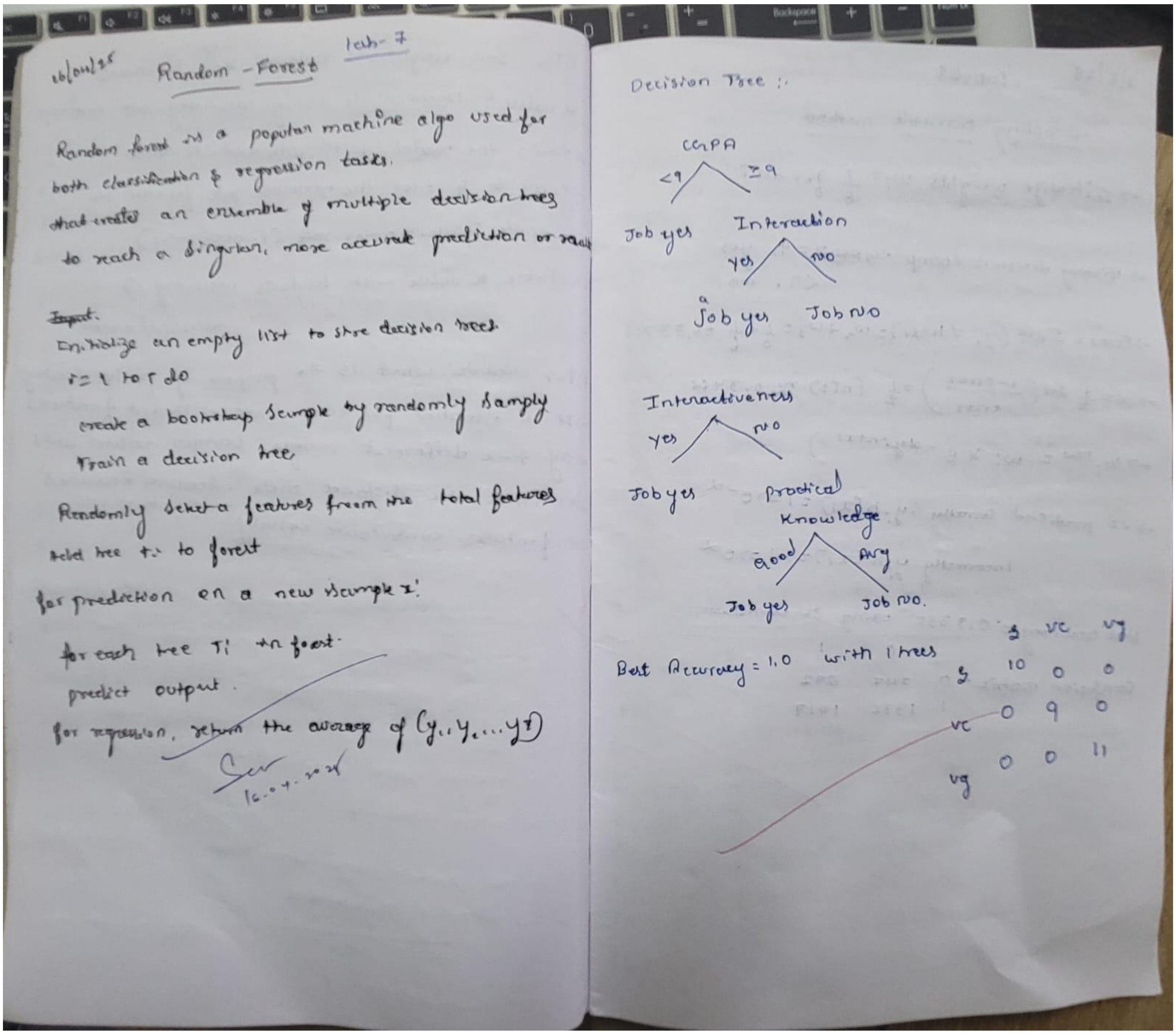
plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:



16/04/25 Random - Forest lab- 7

Random forest is a popular machine algo used for both classification & regression tasks. that creates an ensemble of multiple decision trees to reach a singular, more accurate prediction or result

~~Input.~~

Initialize an empty list to store decision trees.

i = 1 to T do

create a bootstrap sample by randomly sampling

Train a decision tree

Randomly select a features from the total features

Add tree Ti to forest

for prediction on a new sample x:

for each tree Ti in forest.

predict output.

for regression, return the average of (y1, y2, ... yT)

16.04.2025

Decision Tree :

CGPA

<9 ≥9

Job yes Interaction

yes no

Job yes Job No

Interactiveness

yes no

Job yes practical knowledge

Good Avg

Job yes Job No.

Best Accuracy = 1.0 with 1 trees

	g	vc	vg
g	10	0	0
vc	0	9	0
vg	0	0	11

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split from

sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report import

matplotlib.pyplot as plt

import seaborn as sns #

Load the dataset

file_path = '/content/iris.csv' data =

pd.read_csv(file_path)

print("Columns:", data.columns)

# Assume last column is target, others are features X

= data.iloc[:, :-1]

y = data.iloc[:, -1] #

Split dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42) # 1

Build Random Forest 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) score_default

= accuracy_score(y_test, y_pred_default)
```

```
# Show confusion matrix
cm = confusion_matrix(y_test, y_pred_default)
print("\nConfusion Matrix (default 10 trees):")
print(cm)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (n_estimators=10)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print("\nClassification Report:")
print(classification_report(y_test, y_pred_default)) #
Show feature importance
importances = rf_default.feature_importances_
feature_names = X.columns
feat_importances = pd.Series(importances, index=feature_names)
feat_importances.sort_values().plot(kind='barh', figsize=(8,6))
plt.title('Feature Importances (n_estimators=10)')
plt.show() best_score =
0
best_n = 0
scores = []
```

```
n_values = range(1, 101, 5)  # Try from 1 to 100 in steps of 5 for

n in n_values:

        rf = RandomForestClassifier(n_estimators=n, random_state=42)

        rf.fit(X_train, y_train)

        y_pred = rf.predict(X_test)

        score = accuracy_score(y_test, y_pred)

        scores.append(score)

        print(f"n_estimators={n}, Accuracy: {score:.4f}")

        if score > best_score:

            best_score = score

            best_n = n

print(f"\nBest accuracy {best_score:.4f} achieved with n_estimators={best_n}") #

Plot scores vs. number of trees

plt.figure(figsize=(10,6)) plt.plot(n_values,

scores, marker='o')

plt.xlabel('Number of Trees (n_estimators)')

plt.ylabel('Accuracy Score')

plt.title('Random Forest Accuracy vs. Number of Trees')

plt.grid(True)

plt.show()
```

Program 9

Implement Boosting ensemble method on a dataset.

Screenshot:

7/5/25 Lab-08

Boosting ensemble method

→ Initialize weights $w_i = \frac{1}{6}$ for all i

→ Binary decision stump CGPA >= 9, yes
<9, no.

→ Error = $\sum w_i (y_i \neq h(x_i)) = w_2 + w_3 = \frac{1}{6} + \frac{1}{6} = 0.333$

→ $\alpha = \frac{1}{2} \ln\left(\frac{1-\text{error}}{\text{error}}\right) = \frac{1}{2} (\ln(2)) \approx 0.3466$

→ $w_i^{new} = w_i \times e^{-\alpha y_i h(x_i)}$

→ if predicted correctly $y_i h(x_i) = 1 \Rightarrow e^{-\alpha}$

incorrectly $y_i h(x_i) = -1 \Rightarrow e^{\alpha}$

Best accuracy = 0.8335 using 80 estimators.

Confusion matrix:

	0	1
0	7117	297
1	1336	1019

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report import

matplotlib.pyplot as plt

import seaborn as sns #

Load the dataset
```

```
file_path = '/content/income.csv'

data = pd.read_csv(file_path)

# Inspect columns

print("Columns:", data.columns)

# Assume last column is target, others are features X

= data.iloc[:, :-1]

y = data.iloc[:, -1]


# Split dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42) # 1

Build AdaBoost with 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)


score_default = accuracy_score(y_test, y_pred_default)

print(f"n_estimators=10, Accuracy: {score_default:.4f}") #

Show confusion matrix

cm = confusion_matrix(y_test, y_pred_default)

print("\nConfusion Matrix (n_estimators=10):")

print(cm)

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

```
plt.title('Confusion Matrix (n_estimators=10)')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# Show classification report  print("\nClassification

Report:") print(classification_report(y_test,

y_pred_default)) # 2 Fine-tune number of estimators

best_score = 0

best_n = 0

scores = []

n_values = range(10, 201, 10)  # Try from 10 to 200 in steps of 10 for n

in n_values:

    ada = AdaBoostClassifier(n_estimators=n, random_state=42)
    ada.fit(X_train, y_train)

    y_pred = ada.predict(X_test)

    score = accuracy_score(y_test, y_pred)

    scores.append(score)

    print(f"n_estimators={n}, Accuracy: {score:.4f}")

    if score > best_score:

        best_score = score

        best_n = n

print(f"\nBest accuracy {best_score:.4f} achieved with n_estimators={best_n}") #

Plot scores vs. number of estimators
```

```
plt.figure(figsize=(10,6))
plt.plot(n_values, scores, marker='o') plt.xlabel('Number

of Estimators (n_estimators)') plt.ylabel('Accuracy

Score')

plt.title('AdaBoost Accuracy vs. Number of Estimators')

plt.grid(True)

plt.show()
```

Program 10

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

Screenshot:

Lab-09 7/5/25

K-means Algorithm to cluster set of data

Initial cluster

C1: (1.0, 1.0) C2: (5.0, 7.0)

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

Point	C1	C2	Assignment
(1,1)	0	7.21	C1
(1.5,2)	1.12	6.10	C1
(3,4)	3.61	4.24	C1
(5,7)	7.21	0	C2
(3.5,5)	4.72	1.80	C2
(4.5,5)	5.32	2.24	C2
(3.5,4.5)	4.30	2.50	C2

C1: R1, 2, 3

C2 = R4, 5, 6, 7

C1 = (1.83, 2.33)

C2 = (4.125, 5.375)

Iteration 2.

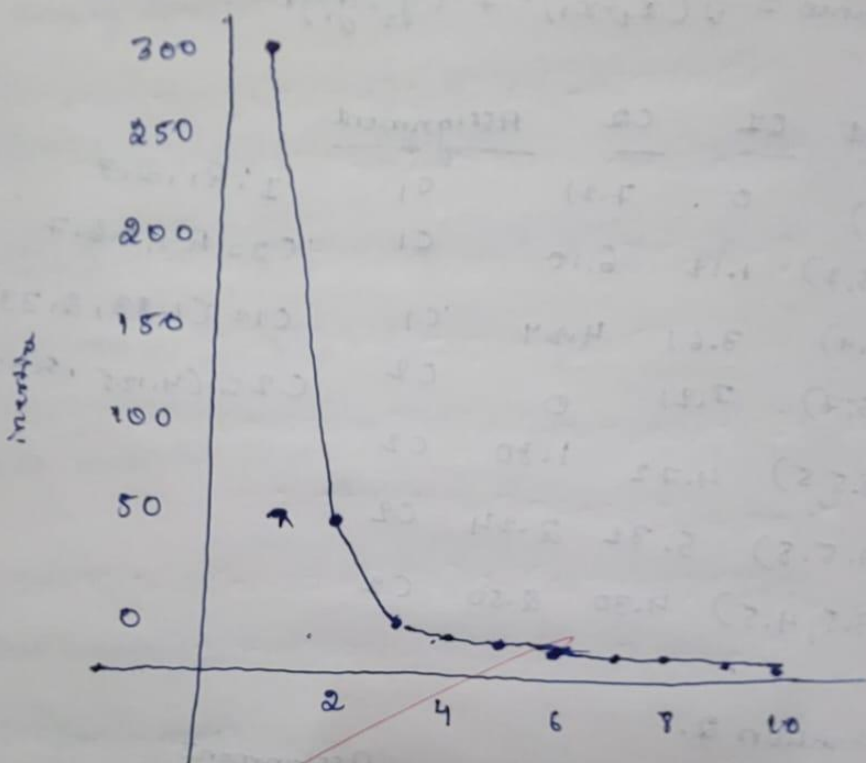
Point	C1	C2	Assignment
(1,1)	1.57	5.45	C1
(1.5,2)	0.47	4.20	C1
(3,4)	1.95	1.26	C2
(5,7)	5.40	1.78	C2
(3.5,5)	2.04	0.67	C2
(4.5,5)	2.25	0.44	C2
(3.5,4.5)	2.03	0.82	C2

Final

C1: R1, 2

C2: R3, R4, R5, R6, R7

Elbow plot



7/8/25

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

```
from scipy import stats

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score from

sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

df1=pd.read_csv("iris.csv")

df1.head()

df = df1.drop(['sepal_length','sepal_width','species'],axis=1)

scaler = StandardScaler()

scaled_df = scaler.fit_transform(df) wcss =

[]

for i in range(1, 11):

    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)

    kmeans.fit(scaled_df)

    wcss.append(kmeans.inertia_)
```

```
plt.plot(range(1, 11), wcss)

plt.title('Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)

pred_y = kmeans.fit_predict(scaled_df)

df['cluster'] = pred_y

plt.scatter(df['petal_length'],    df['petal_width'],    c=df['cluster'])

plt.title('Clusters of Iris Flowers')

plt.xlabel('Petal      Length')

plt.ylabel('Petal       Width')

plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:

Dimensionally Reduction using principal Component Analysis

Given the data, reduce the dimension. from 2 to 1 using PCA compute for first PCA

Feature	example 1	example 2	example 3	example 4
x_1	4	8	13	7
x_2	11	4	5	14

Given

2 features x_1, x_2 with 4 example

$$\begin{bmatrix} 4 & 8 & 13 & 7 \\ 11 & 4 & 5 & 14 \end{bmatrix}$$

eigen values: $\lambda_1 = 30.3849$, $\lambda_2 = 6.6151$

eigen vectors: $e_1 = \begin{bmatrix} 0.5574 \\ 0.8303 \end{bmatrix}$ $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

Reduce dimensionally from 2D to 1D

1) Centre the data:-

$x_1 = 4+8+13+7/4 = 8.0$

$x_2 = 11+4+5+14/4 = 8.5$

Centred data

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

Step-2 :- project the centered data on the first principal component.

use dot product

$z_i = e_1 \cdot$ centered. data

Compute each projection

let $e_1 = [0.5574 - 0.8307]^T$

(i) Ex - 1: $z = (0.5574)(-4) + (-0.8303)$

$(2.5) = -2.2296 - 2.07575 = -6.30535$

Ex. 2: $z = (0.9934)(0) + (-0.8)(-4.5)$

$+ 3.23635 = 3.73635$

Ex 3: $z = (0.5574)(5) + (-0.8303) \cdot (-3.5)$

$= 2.781 + 2.90605 = 5.6805$

Ex-4: $z = (0.5574)(-1) + (0.8303)(5.5)$

$= -0.5574 - 4.56665 = 5.12405$

Final answer

projected data along first principal component

$$\begin{bmatrix} -4.30135 \\ 3.73635 \\ 5.69305 \\ 5.12405 \end{bmatrix}$$

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from scipy import stats

import seaborn as sns
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

df1=pd.read_csv("heart.csv")

df1.head()

text_cols = df1.select_dtypes(include=['object']).columns

label_encoder = LabelEncoder()

for col in text_cols:

  df1[col] = label_encoder.fit_transform(df1[col])

print(df1.head())

X = df1.drop('HeartDisease', axis=1)

y = df1['HeartDisease']

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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

```
# Support Vector Machine
svm_model = SVC(kernel='linear', random_state=42)

svm_model.fit(X_train, y_train)

svm_predictions = svm_model.predict(X_test)

svm_accuracy = accuracy_score(y_test, svm_predictions)

print(f"SVM Accuracy: {svm_accuracy}")

# Logistic Regression

lr_model = LogisticRegression(random_state=42)

lr_model.fit(X_train, y_train)

lr_predictions = lr_model.predict(X_test)

lr_accuracy = accuracy_score(y_test, lr_predictions)

print(f"Logistic Regression Accuracy: {lr_accuracy}")

# Random Forest

rf_model = RandomForestClassifier(random_state=42)

rf_model.fit(X_train, y_train)

rf_predictions = rf_model.predict(X_test)

rf_accuracy = accuracy_score(y_test, rf_predictions)

print(f"Random Forest Accuracy: {rf_accuracy}")

models = {

"SVM": svm_accuracy,

"Logistic Regression": lr_accuracy,

"Random Forest": rf_accuracy
```

```
}
best_model = max(models, key=models.get)
print(f"\nBest Model: {best_model} with accuracy {models[best_model]}")
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
svm_model_pca = SVC(kernel='linear', random_state=42)
svm_model_pca.fit(X_train_pca, y_train)
svm_predictions_pca = svm_model_pca.predict(X_test_pca)
svm_accuracy_pca = accuracy_score(y_test, svm_predictions_pca)
print(f"SVM Accuracy (with PCA): {svm_accuracy_pca}")
lr_model_pca = LogisticRegression(random_state=42)
lr_model_pca.fit(X_train_pca, y_train)
lr_predictions_pca = lr_model_pca.predict(X_test_pca)
lr_accuracy_pca = accuracy_score(y_test, lr_predictions_pca)
print(f"Logistic Regression Accuracy (with PCA): {lr_accuracy_pca}")
rf_model_pca = RandomForestClassifier(random_state=42)
rf_model_pca.fit(X_train_pca, y_train)
rf_predictions_pca = rf_model_pca.predict(X_test_pca)
rf_accuracy_pca = accuracy_score(y_test, rf_predictions_pca)
print(f"Random Forest Accuracy (with PCA): {rf_accuracy_pca}")
models_pca = {
"SVM": svm_accuracy_pca,
```

```
"Logistic Regression": lr_accuracy_pca,

"Random Forest": rf_accuracy_pca

    }

best_model_pca = max(models_pca, key=models_pca.get)

print(f"\nBest Model (with PCA): {best_model_pca} with accuracy
{models_pca[best_model_pca]}")
```