# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**

**LAB REPORT**
**on**

# Machine Learning (23CS6PCMAL)

***Submitted by***

**NITHYA LAKSHMI V (1BM22CS186)**

***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 **NITHYA LAKSHMI V (1BM22CS186),** 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.

| | |
|---|---|
| Ms. Saritha A N<br>Assistant Professor<br>Department of CSE, BMSCE | Dr. Kavitha Sooda<br>Professor & HOD<br>Department of CSE, BMSCE |

# Index

Github Link:
https://github.com/Nithya1909/ML-LAB

## Program 1

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

Code:

```
import pandas as pd

# Method-1: Initializing values directly into DataFrame

data_method1 = {'USN': ['1JS17CS001', '1JS17CS002', '1JS17CS003',

'1JS17CS004', '1JS17CS005'],

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

'Marks': [90, 85, 92, 78, 88]}

df_method1 = pd.DataFrame(data_method1)

print("Method-1:")

print(df_method1)

print("-" * 20)

# Method-2: Importing datasets from sklearn.datasets

from sklearn.datasets import load_diabetes

diabetes_data = load_diabetes()

df_method2 = pd.DataFrame(data=diabetes_data.data,

columns=diabetes_data.feature_names)

df_method2['target'] = diabetes_data.target

print("Method-2:")

print(df_method2.head())

print("-" * 20)

# Method-3: Importing datasets from a specific .csv file

try:
```

```
df_method3 = pd.read_csv('sample_sales_data.csv')
print("Method-3:")
print(df_method3.head())
print("-" * 20)
except FileNotFoundError:
print("sample_sales_data.csv not found. Please upload the file.")
print("-" * 20)
import yfinance as yf
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
start_date = "2024-01-01"
end_date = "2024-12-30"
data = yf.download(tickers, start=start_date, end=end_date)
closing_prices = data['Close']
daily_returns = closing_prices.pct_change().dropna()
plt.figure(figsize=(12, 6))
closing_prices.plot()
plt.title('Closing Prices (2024)')
plt.xlabel('Date')
plt.ylabel('Price (INR)')
plt.grid(True)
plt.show()
plt.figure(figsize=(12, 6))
daily_returns.plot()
plt.title('Daily Returns (2024)')
plt.xlabel('Date')
```

```python
plt.ylabel('Daily Return')
plt.grid(True)
plt.show()
```

## **Program 2**

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

Code:

```
from google.colab import files

uploaded = files.upload()



uploaded = files.upload()

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from scipy import stats

import pandas as pd
df1=pd.read_csv("adult.csv")

print(df1.head())

df2=pd.read_csv("Dataset of Diabetes .csv")

print(df2.head())

df1["education"].value_counts()

ordinal_encoder = OrdinalEncoder(categories=[["HS-grad",

"Some-college","Bachelors","Masters","Assoc-voc","11th","Assoc-acdm","10th","7t

h-8th","Prof-school","9th","12th","Doctorate","5th-6th","1st-4th","Preschool"]]

)

df1["Education_Encoded"] = ordinal_encoder.fit_transform(df1[["education"]])

onehot_encoder = OneHotEncoder()

encoded_data =

onehot_encoder.fit_transform(df1[["gender","relationship","workclass","occupati

on","race","native-country","income","marital-status"]])
```

```
encoded_array = encoded_data.toarray()


encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["gender","relationship","workclas
s","occupation","race","native-country","income","marital-status"]))
df_encoded = pd.concat([df1, encoded_df], axis=1)
df_encoded.drop(["education","gender","workclass","relationship","occupation","
race","native-country","income","marital-status"], axis=1, inplace=True)


print(df_encoded.head())
normalizer = MinMaxScaler()
df_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per
-week"]] =
normalizer.fit_transform(df_encoded[["fnlwgt","educational-num","capital-gain",
"capital-loss","hours-per-week"]])
df_encoded.head()
df2.isnull().sum()
df2['Gender'] = df2['Gender'].replace('f', 'F')
ordinal_encoder = OrdinalEncoder(categories=[["F", "M"]])
df2["Gender_Encoded"] = ordinal_encoder.fit_transform(df2[["Gender"]])
onehot_encoder = OneHotEncoder()
encoded_data = onehot_encoder.fit_transform(df2[["CLASS"]])
encoded_array = encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df2, encoded_df], axis=1)
```

```
df2 = pd.concat([df2, encoded_df], axis=1)

df2.drop("CLASS", axis=1, inplace=True)

df2.drop("Gender", axis=1, inplace=True)

print(df2.head())

normalizer = MinMaxScaler()

df_encoded[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,

"Chol","TG","HDL","LDL","VLDL","BMI"]] =

normalizer.fit_transform(df_encoded[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,

"Chol","TG","HDL","LDL","VLDL","BMI"]])

df_encoded.head()
```

# Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:

LAB-4

Simple linear Regression

```
function linear reg (x, y, r, num)

   m=0
   b=0
   n = length(x)

   for i ← 1 to num
      pred = [ ]
      for i ∈ 0 to n-1
         pred = m[i] + b
         append pre to pred

      errors = [ ]
      for j ← 0 to n-1
         error = pred[j] - y[j]
         error.append(error)

   cost = (1/n) * sum (error 2 for error in errors)

   m gradient = 2/n x sum (error[j] x x[j] for j ← 0 to n-1)

   b_gradient = 2/n x sum(error)

   m = m - m_gradient x r
   b = b - b_gradient x r

   return m, b
```

## Algo
## Linear Regression

1. Intialise m & b = 0
2. make prediction mX+b
3. Find error by y - y_pred
4. update m & b using gradient descent
5. Repeat 4 steps for num-iteration

## Algo logistics Regression

1. m be the length of features initialize to 0.
2. n = no of data points
3. Intialize weight & bias to 0
4. Make predictions weight X + bias
5. Use Sigmoid function for pred.
6. Find errors using pred - y[i]
7. Update weight & bias using gradient descent
8. Repeat steps & return weight & bias

## Algo multilinear Regression

Same as linear but with multiple features.

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')

df

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)


print(reg.coef_)

print(reg.intercept_)

reg.predict([[5000]])

df_mlr=pd.read_csv('/content/homeprices_Multiple_LR.csv')

df_mlr

df_mlr.bedrooms.median()

df_mlr.bedrooms = df_mlr.bedrooms.fillna(df_mlr.bedrooms.median())

print(df_mlr)

reg = linear_model.LinearRegression()

reg.fit(df_mlr.drop('price',axis='columns'),df_mlr.price)
```

```
reg.coef_

reg.intercept_

#Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old

reg.predict([[3000, 3, 40]])

112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384


canada=pd.read_csv('/content/canada_per_capita_income.csv')

canada

plt.xlabel('year')

plt.ylabel('per capita income (US$)')

canada.rename(columns={'per capita income (US$)':'income'},inplace=True)

plt.scatter(canada.year,canada.income ,color='red',marker='+')

new_df_canada = canada.drop('income',axis='columns')

new_df_canada.sample(5)

income = canada.income

income.sample(5)

reg = linear_model.LinearRegression()

reg.fit(new_df_canada,income)

print(reg.coef_)

print(reg.intercept_)

reg.predict([[2020]])

hiring=pd.read_csv('/content/hiring.csv')

hiring

hiring['experience'].fillna(0, inplace=True)

hiring['test_score(out of 10)'].fillna(hiring['test_score(out of 10)'].mean(),

inplace=True)
```

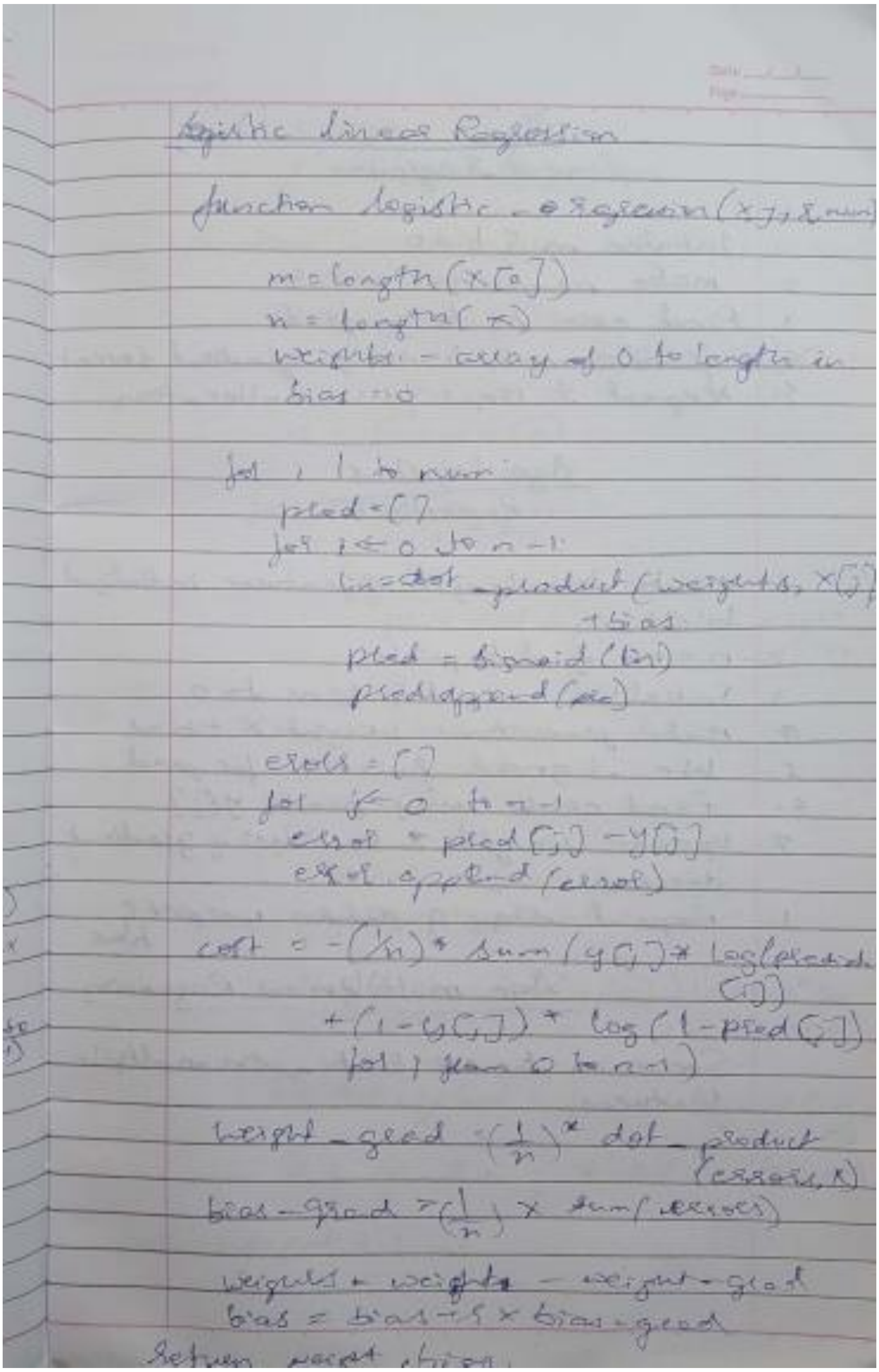
```
def convert_to_int(word):

word_dict = {'one':1, 'two':2, 'three':3, 'four':4, 'five':5, 'six':6,

'seven':7, 'eight':8,

'nine':9, 'ten':10, 'eleven':11, 'twelve':12, 'zero':0, 0: 0}

return word_dict[word]

hiring['experience'] = hiring['experience'].apply(lambda x : convert_to_int(x))
hiring

reg = linear_model.LinearRegression()

hiring.rename(columns={'salary($)':'salary'},inplace=True)

reg.fit(hiring.drop('salary',axis='columns'),hiring.salary)

print(reg.coef_)

print(reg.intercept_)

#What is the predicted salary for a candidate with 12 years of experience, 10

test score, and 10 interview score?

reg.predict([[12, 10, 10]])


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

comp

comp.isnull().sum()

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

state_encoded = ohe.fit_transform(comp[['State']])

state_encoded_df = pd.DataFrame(state_encoded,

columns=ohe.get_feature_names_out(['State']))

comp = pd.concat([comp, state_encoded_df], axis=1).drop(columns=['State'])

print(comp)

reg = linear_model.LinearRegression()
```

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

print(reg.coef_)

print(reg.intercept_)


reg.predict([[91694.48, 515841.3, 11931.24,0,1,0]])
```

## **Program 4**

Build Logistic Regression Model for a given dataset.

Screenshot:



```
Logistic Linear Regression

function logistic_regression(x, y, num)

    m = length(x[0])
    n = length(x)
    weights = array of 0 to length m
    bias = 0

    for i 1 to num
        pred = []
        for j <- 0 to n-1
            lin = dot_product(weights, x[j])
                       + bias
            pred = sigmoid(lin)
            pred.append(pred)

        errors = []
        for j <- 0 to n-1
            error = pred[j] - y[j]
            error append(error)

        cost = -(1/n) * sum(y[j] * log(pred[j])
               + (1 - y[j]) * log(1 - pred[j])
               for j from 0 to n-1)

        weight_grad = (1/n) * dot_product
                                 (errors, x)
        bias_grad = (1/n) x sum(errors)

        weights = weights - weight_grad
        bias = bias - r x bias_grad
return weight, bias
```

Code:

```
import pandas as pd

from matplotlib import pyplot as plt

from seaborn import regplot

import seaborn as sns

df1=pd.read_csv('HR_comma_sep.csv')

df1.sample(10)

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

sns.barplot(x='salary_encoded', y='left', data=df1)

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

plt.xlabel('Salary Level (Encoded)')

plt.ylabel('Proportion of Employees Left')

plt.show()

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

sns.barplot(x='Department', y='left', data=df1)

plt.title('Employee Retention Rate by Department')

plt.xlabel('Department')

plt.ylabel('Proportion of Employees Left')

plt.xticks(rotation=45, ha='right')

plt.show()

from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

import numpy as np

import seaborn as sns

ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)

department_encoded = ohe.fit_transform(df1[['Department']])

department_encoded_df = pd.DataFrame(department_encoded,
```

```
columns=ohe.get_feature_names_out(['Department']))

df1 = pd.concat([df1, department_encoded_df], axis=1)

df1 = df1.drop('Department', axis=1)

ordinal_encoder = OrdinalEncoder(categories=[['low', 'medium', 'high']],

dtype=np.int64)

salary_encoded = ordinal_encoder.fit_transform(df1[['salary']])

df1['salary_encoded'] = salary_encoded

df1 = df1.drop('salary', axis=1)

df1.head()

correlation_matrix = df1.corr()

plt.figure(figsize=(12, 10))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
```

```
plt.title('Correlation Matrix of Features')

plt.show()

correlation_threshold = 0.1

correlated_features = correlation_matrix['left'].abs() > correlation_threshold

highly_correlated_features =

correlated_features[correlated_features].index.tolist()

new_df = df1[highly_correlated_features]

print(new_df.head())


from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


X = new_df.drop('left', axis=1)

y = new_df['left']

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

random_state=42)

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")

example_data = X_test.iloc[0].values.reshape(1, -1)

prediction = model.predict(example_data)


import pandas as pd
```

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

zoo_data = pd.read_csv('zoo-data.csv')

zoo_class = pd.read_csv('zoo-class-type.csv')


merged_data = pd.merge(zoo_data, zoo_class, left_on='class_type',

right_on='Class_Number')

merged_data = merged_data.drop(['Animal_Names',

'Number_Of_Animal_Species_In_Class',

'Class_Number','class_type','animal_name'], axis=1)


X = merged_data.drop('Class_Type', axis=1)
y = merged_data['Class_Type']

print(merged_data.head())


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

random_state=42)


model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")


from sklearn.metrics import confusion_matrix

import seaborn as sns
```

```
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

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

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",

xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))

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:

LAB-2

ID3 Algorithm

ID3 algorithm is a popular decision tree algorithm used in machine learning. It aims to build decision tree by iteratively selecting the best attribute to split the data based on information gain.

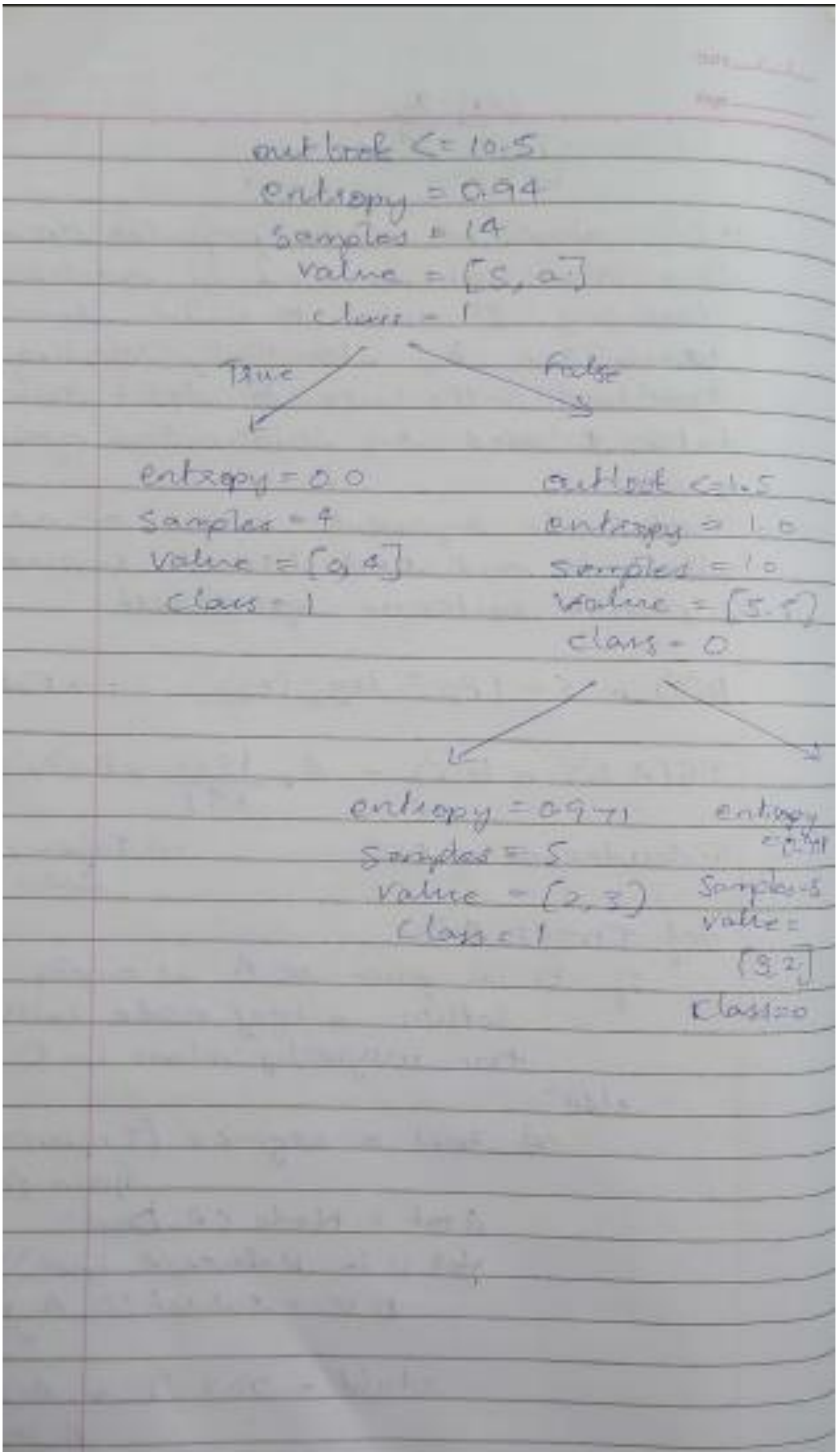
Each node represents a test on an attribute and each branch represent a possible outcome of the test.

$H(S) = \Sigma - (P_i * \log_2 (P_i))$ → Entropy

$IG(A, D) = H(S) - \Sigma_v \frac{|S_v|}{|S|} \times H(S_v)]$ → Information Gain

Pseudocode of ID3

```
def ID3(D,A):
    if D is pure or A is empty
        return a leaf node with
        the majority class in D
    else:
        A_best = argmax (Information
                 Gain (D,A)
        root = Node (A_best)
        for v in values(A_best):
            D_v = subset (D, A_best
                  v)
            child = ID3 (D_v, A -
                    A_best)
            root.add_child(v, child)
    return root
```


outlook <= 10.5
entropy = 0.94
samples = 14
value = [5, 9]
class = 1
True
False
entropy = 0.0
samples = 4
value = [0, 4]
class = 1
outlook <= 1.5
entropy = 1.0
samples = 10
value = [5, 5]
class = 0
entropy = 0.971
samples = 5
value = [2, 3]
class = 1
entropy = 0.971
samples = 5
value = [3, 2]
class = 0

17/03/25 LAB-3

ID3 - Code Implementation without using sklearn library.

```
import pandas as pd
import numpy as np

data = pd.readcsv("weather.csv")
df = pd.DataFrame(data)

def entropy(data):
    class_counts = data.value_counts()
    prob = class_counts/len(data)
    return -np.sum(prob*np.log2(prob))

def information_gain(df, feature, target):
        total_entropy = entropy(df[target])
        feature_value = df[feature].unique
        weighted_entropy = 0
        for value in feature_values:
                subset = df[df[feature]==
                                        value]
                weighted_entropy += (len
                                    (subset)/
                                    len(df))
                                   * entropy
                                (subset[target])
    return total_entropy - weighted_entropy

def id3(df, features, target):
    if len(df[target].unique()) == 1:
        return df[target].iloc[0]
```

```
if not features:
        return df[target].mode()[0]
gains = {feature: information_gain
             (df, feature, target) for
          feature in features}
best_feature = max(gains, key=gains.get)
tree = {best_feature: {}}

for value in df[best_failure].unique():
  subset = df[df[best_failure] == value]
  tree[best_feature][value] = id3(subset,
               [f for f in features if
                      f != best_feature],
                    target)
      return valuetree

features = ['outlook', 'temperature',
              'Humidity', 'wind']
target = 'PlayTennis'
dec_tree = id3(df, features, target)
print(dec_tree)
```

OUTPUT

```
{'Outlook': {'Sunny':
   {Humidity: {'high': 'No', 'low': 'yes'}},
   'overcast': 'Yes', 'Rain': {'wind':
                {'weak': 'Yes',
                 'Strong': 'No'}}}}
```

Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
data = {
'a1': [True, True, False, False, False, True, True, True, False, False],
'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool',
'Cool'],
'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High',
'Normal', 'Normal', 'High'],
'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes',
'Yes', 'Yes']
}
data
df = pd.DataFrame(data)
label_encoders = {}
for column in df.columns:
le = LabelEncoder()
df[column] = le.fit_transform(df[column])
label_encoders[column] = le
df
X = df.drop('Classification', axis=1)
y = df['Classification']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

```
random_state=42)


clf = DecisionTreeClassifier(criterion='entropy')

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

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

print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))

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=['No', 'Yes'])

plt.show()


iris=pd.read_csv("/content/iris - Copy.csv")

iris

le = LabelEncoder()

iris["species"] = le.fit_transform(iris["species"])

label_encoders[column] = le

iris

X = iris.drop('species', axis=1)

y = iris['species']

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

random_state=42)

clf = DecisionTreeClassifier(criterion='entropy')

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
```

```
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',]))

from sklearn.metrics import confusion_matrix

import seaborn as sns

cm = confusion_matrix(y_test, y_pred)

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'],

yticklabels=['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

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

plot_tree(clf, filled=True, feature_names=X.columns,

class_names=['Iris-setosa', 'Iris-versicolor','Iris-virginica'])

plt.show()


drug=pd.read_csv("/content/drug - Copy.csv")

print(drug)

drug["Drug"].unique()

le = LabelEncoder()

drug["Sex"] = le.fit_transform(drug["Sex"])

drug["BP"] = le.fit_transform(drug["BP"])

drug["Cholesterol"] = le.fit_transform(drug["Cholesterol"])
```

```
#drug["Drug"] = le.fit_transform(drug["Drug"])

label_encoders[column] = le

drug

X = drug.drop('Drug', axis=1)

y = drug['Drug']

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

clf = DecisionTreeClassifier(criterion='entropy')

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

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

print(classification_report(y_test, y_pred, target_names=['drugY', 'drugC',
'drugX', 'drugA', 'drugB']))

from sklearn.metrics import confusion_matrix

import seaborn as sns

cm = confusion_matrix(y_test, y_pred)

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['drugY', 'drugC', 'drugX', 'drugA', 'drugB'],
yticklabels=['drugY', 'drugC', 'drugX', 'drugA', 'drugB'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()


petrol=pd.read_csv("/content/petrol_consumption - Copy.csv")
```

```
petrol

X = petrol.drop('Petrol_Consumption', axis=1)

y = petrol['Petrol_Consumption']

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

random_state=42)

clf = DecisionTreeRegressor()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

from sklearn.tree import DecisionTreeRegressor

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error, mean_squared_error


mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = mean_squared_error(y_test, y_pred)**0.5

print(f'Mean Absolute Error: {mae}')

print(f'Mean Squared Error: {mse}')

print(f'Root Mean Squared Error: {rmse}')

plt.figure(figsize=(30, 30))

plot_tree(clf, filled=True, feature_names=X.columns, fontsize=10)

plt.show()
```

## Program 6

Build KNN Classification model for a given dataset.

Screenshot:

7/4/25 LAB-5

KNN algorithm

1. choose the number of neighbors (k)

eg :- K = 3

2. Calculate distance from the test points to all training points

use a distance metric like :
Euclidean distance :

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

3. Sort the distances in ascending order of the Euclidean's distance

4. Select the k nearest neighbors
Pick the top k points with the smallest distance.

5. Do a majority vote (for classification)
→ Count the labels of the k neighbors.
→ The label with the most votes is the predicted class.

(for Regression) : average of the k nearest neighbors

Code:

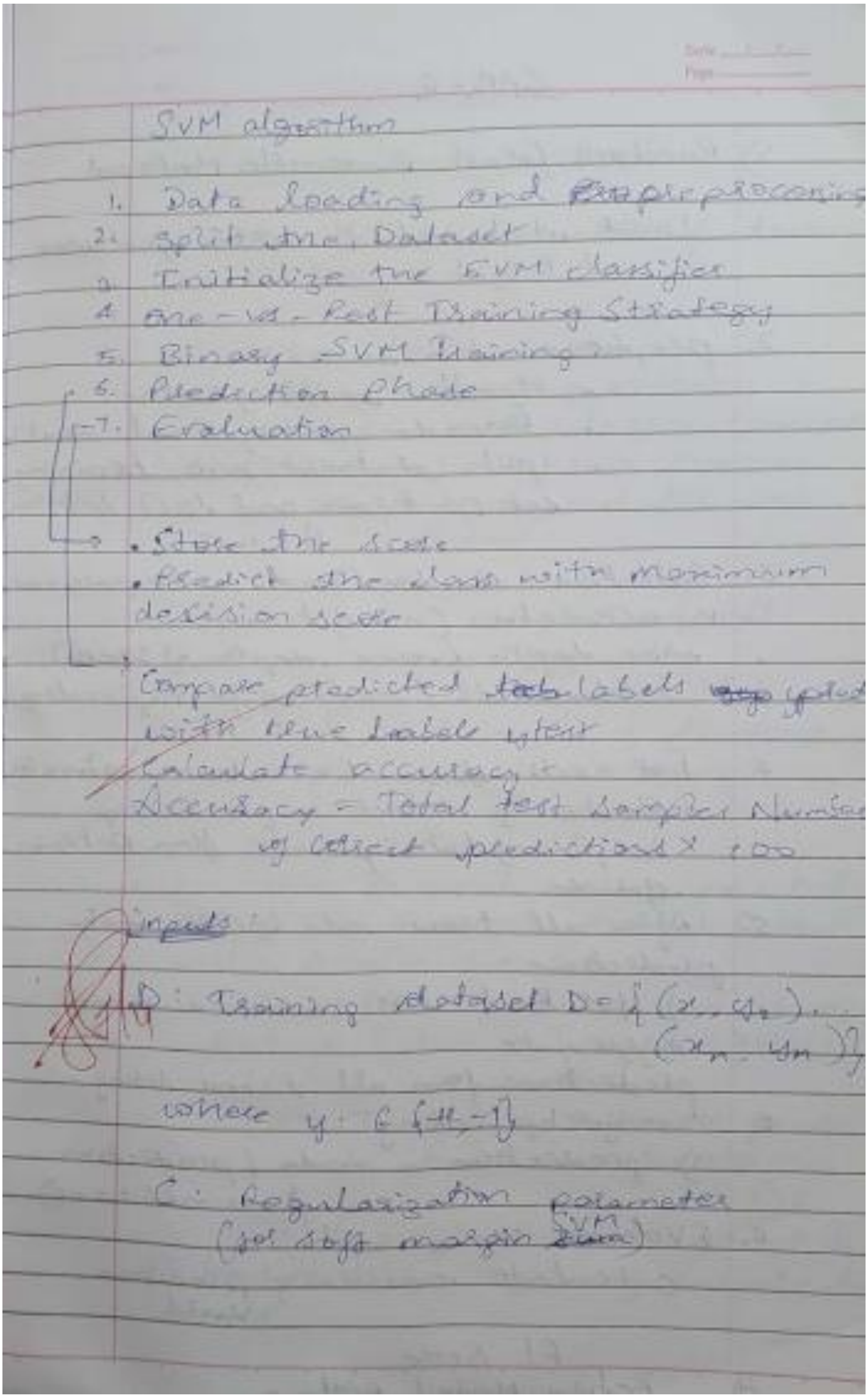
```
import pandas as pd
iris=pd.read_csv('iris.csv')
iris.head()
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

X = iris.drop('species', axis=1)
y = iris['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
best_k = 1
best_accuracy = 0
for k in range(1, 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}:
    {1-accuracy}")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
```

```
        best_k = k
print(f"Best k value: {best_k}")


knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues',xticklabels=iris['species'].unique(),
yticklabels=iris['species'].unique())
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()


print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

## Program 7

Build Support vector machine model for a given dataset

Screenshot:



SVM algorithm

1. Data loading and ~~Pre~~preprocessing
2. Split the Dataset
3. Initialize the SVM classifier
4. One-vs-Rest Training Strategy
5. Binary SVM Training
6. Prediction phase
7. Evaluation

- Store the score
- Predict the class with maximum decision score

Compare predicted labels (pred) with true label y_test

Calculate accuracy

Accuracy = Total test samples Number of correct predictions × 100

Inputs:

D : Training dataset $D = \{(x_1, y_1), \ldots (x_n, y_n)\}$

where $y_i \in \{+1, -1\}$

C : Regularization parameter (for soft margin SVM)

Code:

```
import pandas as pd

iris=pd.read_csv('iris.csv')

iris.head()

iris.isnull().sum()
import pandas as pd

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

iris = pd.read_csv('iris.csv')

X = iris.drop('species', axis=1)

y = iris['species']

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

random_state=42)

svm_rbf = SVC(kernel='rbf', gamma='scale')

svm_rbf.fit(X_train, y_train)

y_pred_rbf = svm_rbf.predict(X_test)

accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

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

cm_rbf = confusion_matrix(y_test, y_pred_rbf)

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

sns.heatmap(cm_rbf, annot=True, fmt="d", cmap="Blues",

xticklabels=iris['species'].unique(),

yticklabels=iris['species'].unique())

plt.title("Confusion Matrix (RBF Kernel)")
```

```
plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)

y_pred_linear = svm_linear.predict(X_test)

accuracy_linear = accuracy_score(y_test, y_pred_linear)

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

cm_linear = confusion_matrix(y_test, y_pred_linear)

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

sns.heatmap(cm_linear, annot=True, fmt="d", cmap="Blues",

xticklabels=iris['species'].unique(),

yticklabels=iris['species'].unique())

plt.title("Confusion Matrix (Linear Kernel)")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()


df2=pd.read_csv('letter-recognition.csv')

df2.head()

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, classification_report,

confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt
```

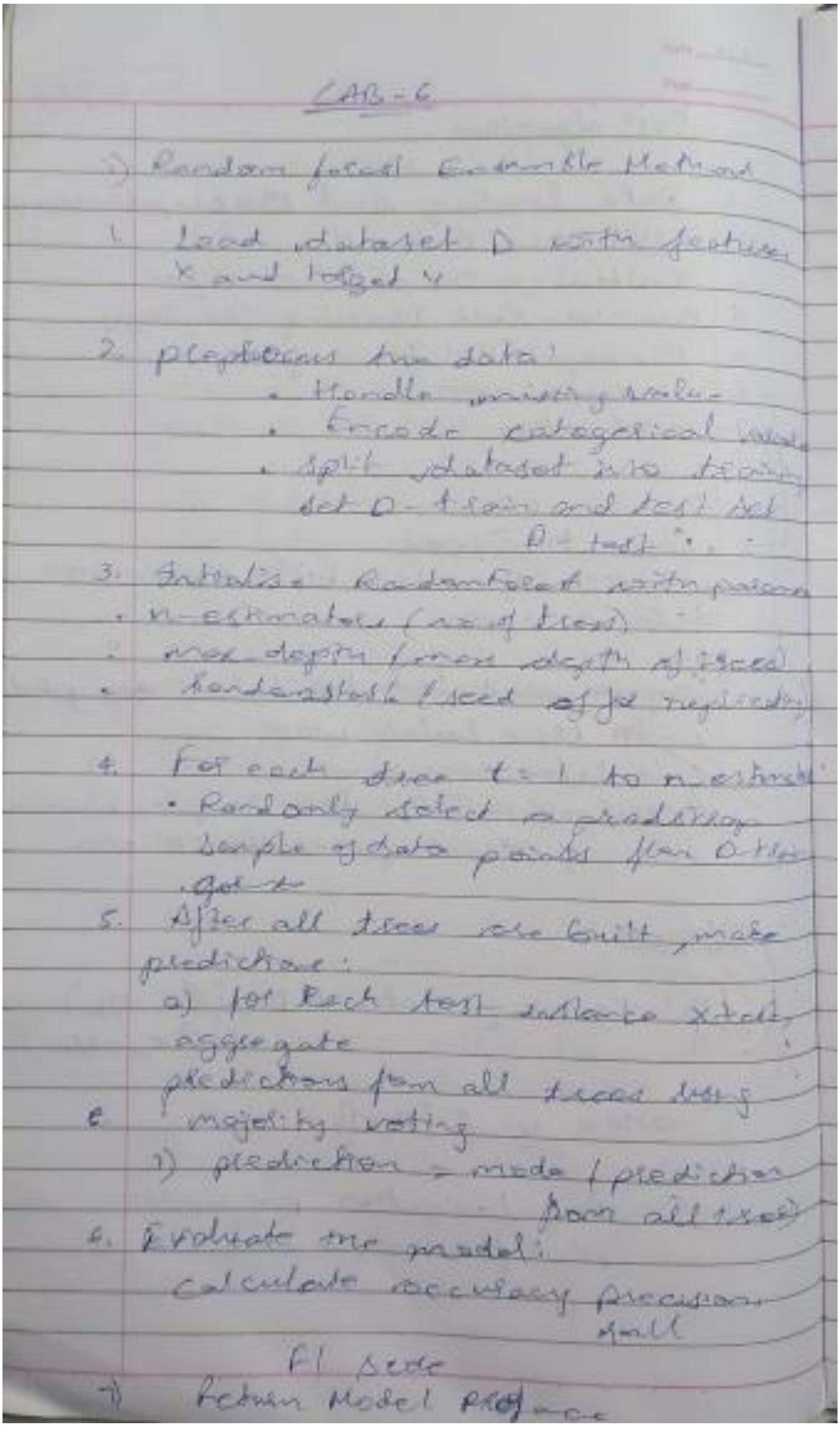
```
from sklearn.preprocessing import LabelBinarizer

X = df2.drop('letter', axis=1)

y = df2['letter']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

random_state=42)

svm_classifier = SVC(kernel='linear', probability=True)

svm_classifier.fit(X_train, y_train)


y_pred = svm_classifier.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)

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

plt.title('Confusion Matrix for SVM')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

lb = LabelBinarizer()

lb.fit(y_test)

y_test_lb = lb.transform(y_test)

y_pred_prob = svm_classifier.predict_proba(X_test)

fpr = {}

tpr = {}

thresh ={}

roc_auc = dict()

n_class = y_test_lb.shape[1]

for i in range(n_class):
```

```
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test_lb[:,i], y_pred_prob[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])
plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label='SVM (AUC =
%0.2f)' % roc_auc[0])
plt.title('ROC Curve for Class 0')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
print(f"AUC score for class 0: {roc_auc[0]}")
```

**Program 8**

Implement Random forest ensemble method on a given dataset.

Screenshot:



LAB - 6

1) Random forest Ensemble Method

1. Load dataset D with features X and target y

2. preprocess the data:
   - Handle missing values
   - Encode categorical values
   - Split dataset into training set D-train and test set D-test

3. Initialize RandomForest with params
   - n-estimators (no of trees)
   - max depth (max depth of trees)
   - randomstate (seed of for replicating)

4. For each tree t = 1 to n-estimators
   - Randomly select a bootstrap sample of data points from D-train set

5. After all trees are built, make predictions:
   a) for each test instance X test, aggregate predictions from all trees using majority voting
   1) prediction = mode (predictions from all trees)

6. Evaluate the model:
   calculate accuracy precision recall F1 score

7) Return Model performance

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

data = pd.read_csv('iris.csv')
X = data.drop('species', axis=1)

y = data['species']

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

random_state=42)

rf_classifier = RandomForestClassifier(random_state=42)

rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

default_accuracy = accuracy_score(y_test, y_pred)

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


best_n_estimators = 10

best_accuracy = default_accuracy

for n_estimators in range(1, 101):

    rf_classifier = RandomForestClassifier(n_estimators=n_estimators,

random_state=42)

    rf_classifier.fit(X_train, y_train)

    y_pred = rf_classifier.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    if accuracy > best_accuracy:

            best_accuracy = accuracy

            best_n_estimators = n_estimators
```

```
print(f"Best accuracy: {best_accuracy} achieved with n_estimators:
{best_n_estimators}")
from sklearn.metrics import precision_score, recall_score, f1_score
best_rf_classifier = RandomForestClassifier(n_estimators=best_n_estimators,
random_state=42)
best_rf_classifier.fit(X_train, y_train)
best_y_pred = best_rf_classifier.predict(X_test)
precision = precision_score(y_test, best_y_pred, average='weighted')
recall = recall_score(y_test, best_y_pred, average='weighted')
f1 = f1_score(y_test, best_y_pred, average='weighted')
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, best_y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
    xticklabels=data['species'].unique(),
    yticklabels=data['species'].unique())
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

## Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot:

LAB-7

Boosting Ensemble Method

1. Read dataset D with features X and target Y

2. preprocess the data
   a. Handle missing values
   b. Encode ~~cat~~ categorical variab[illegible]
   c. Split dataset into training set D-train and test set D-test

3. Initialise the weights of all training sample equally
   $w1 = w2 = \ldots = wN = 1/N$
   where ~~[illegible]~~ N is the no of training sample

4. For each iteration $t = 1$ to n-estimators:
   a. i. Train a weak classifier h-t using the weighted dataset D-train with sample weights w-i
   b. calculate the error of the classifi[illegible]
      $E\text{-}t = (\Sigma w_i \times I(h_t(x_i) \neq y_i)) / \Sigma w_i$
      where I is the indicator functio[illegible]
   c) Compute the classifier weight (α-t):
      $\alpha\text{-}t = 0.5 * \log((1 - E\text{-}t) / E\text{-}t)$
   d) Update the weights of misclassified points

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv('income.csv')


label_encoders = {}
for column in df.columns:
    if df[column].dtype == 'object':
        le = LabelEncoder()
        df[column] = le.fit_transform(df[column])
        label_encoders[column] = le

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


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


abc = AdaBoostClassifier(n_estimators=10, random_state=0)
abc.fit(X_train, y_train)
y_pred = abc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with n_estimators=10: {accuracy}")
```

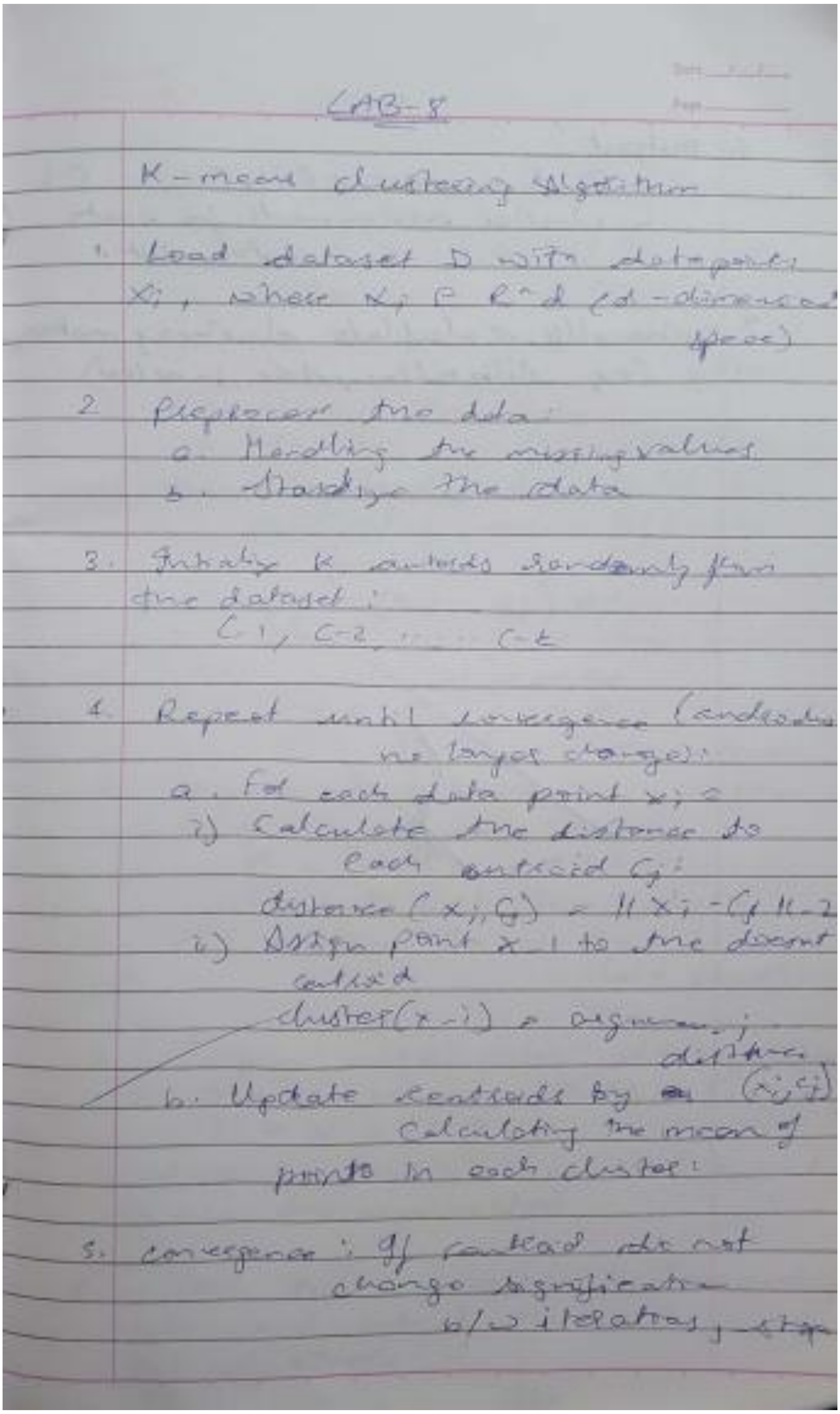
```
best_accuracy = 0
best_n_estimators = 0
for n_estimators in range(1, 101):
    abc = AdaBoostClassifier(n_estimators=n_estimators, random_state=0)
    abc.fit(X_train, y_train)
    y_pred = abc.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_n_estimators = n_estimators
print(f"\nBest accuracy: {best_accuracy} achieved with
n_estimators={best_n_estimators}")
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


abc = AdaBoostClassifier(n_estimators=73, random_state=0)
abc.fit(X_train, y_train)
y_pred = abc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with n_estimators=73: {accuracy}")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
plt.figure(figsize=(8, 6))
```

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

xticklabels=["<=50K", ">50K"], yticklabels=["<=50K", ">50K"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix Heatmap")

plt.show()
```

## Program 10

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

Screenshot:



LAB-8

K-means clustering Algorithm

1. Load dataset D with datapoints $x_i$, where $x_i \in R^d$ (d-dimensional space)

2. Preprocess the data:
   a. Handling the missing values
   b. Standardize the data

3. Initialize K centroids randomly from the dataset:
   $C_1, C_2, \ldots, C_k$

4. Repeat until convergence (centroids no longer change):
   a. For each data point $x_i$:
      i) Calculate the distance to each centroid $C_j$:
      $\text{distance}(x_i, C_j) = \| x_i - C_j \|_2$
      ii) Assign point $x_i$ to the closest centroid
      $\text{cluster}(x_i) = \text{argmin}_j \ \text{distance}(x_i, C_j)$
   b. Update centroids by calculating the mean of points in each cluster:

5. Convergence: If centroid do not change significantly b/w iterations, stop

6. Output :
   a. Find centroids C1, C2, ... Ck
   b. cluster assignments for each data point

7. Optionally calculate clustering metrics (eg. Silhouette, elbow method)

Code:

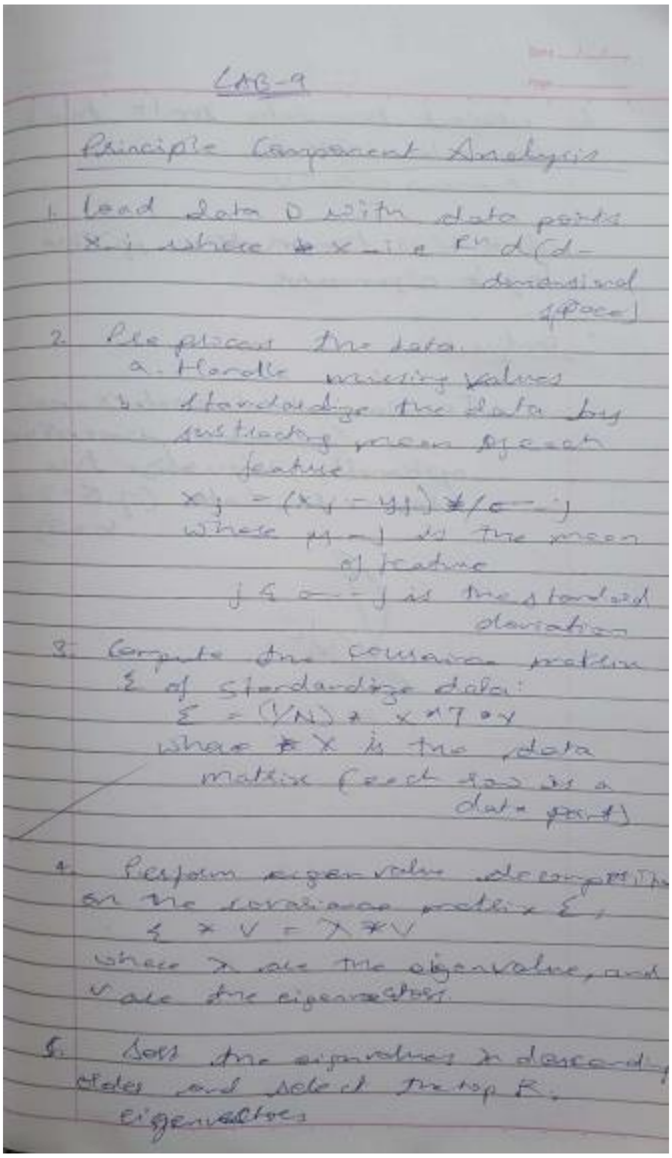
```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

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

df = df[['petal_width', 'petal_length']]

scaler = MinMaxScaler()

df[['petal_width', 'petal_length']] = scaler.fit_transform(df[['petal_width',
```

```python
'petal_length']])
sse = []
k_rng = range(1, 10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df)
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng, sse)
plt.show()
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(df)
df['cluster'] = kmeans.labels_
plt.scatter(df['petal_width'], df['petal_length'], c=df['cluster'],
cmap='viridis')
plt.xlabel('Petal Width')
plt.ylabel('Petal Length')
plt.title('K-Means Clustering of Iris Flowers')
plt.show()
```

## **Program 11**

Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Screenshot:


LAB-9

Principle Component Analysis

1. Load data D with data points $x_i$ where $x_i \in R^d$ (d-dimensional space)

2. Preprocess the data.
   a. Handle missing values
   b. Standardize the data by subtracting mean of each feature
   $x_j = (x_j - \mu_j) / \sigma_j$
   where $\mu_j$ is the mean of feature j & $\sigma_j$ is the standard deviation

3. Compute the covariance matrix $\Sigma$ of standardize data:
   $\Sigma = (1/N) * X^T \cdot X$
   where X is the data matrix (each row is a data point)

4. Perform eigenvalue decomposition on the covariance matrix $\Sigma$,
   $\Sigma * V = \lambda * V$
   where $\lambda$ are the eigenvalue, and V are the eigenvectors.

5. Sort the eigenvalues $\lambda$ in descending order and select the top k eigenvectors

6. project the data onto the k eigenvectors (PCA):
$X\_new = X * V$

where V is the matrix of the top k eigenvectors

- **output**:

Transformed dataset X_new (reduced dimensions)
optionally, visualize the reduced data (if K=2 or K=3)

16/5

Code:

```
import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

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

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

le = LabelEncoder()

ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
for col in text_cols:

    df[col + '_le'] = le.fit_transform(df[col])

    ohe_results = ohe.fit_transform(df[[col]])

    df_ohe = pd.DataFrame(ohe_results, columns=[f"{col}_{i}" for i in

range(ohe_results.shape[1])])

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

df = df.drop(text_cols, axis=1)

print(df.head())

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, MinMaxScaler

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

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

le = LabelEncoder()

ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)

for col in text_cols:

df[col + '_le'] = le.fit_transform(df[col])

ohe_results = ohe.fit_transform(df[[col]])

df_ohe = pd.DataFrame(ohe_results, columns=[f"{col}_{i}" for i in
```

```
range(ohe_results.shape[1])])
df = pd.concat([df, df_ohe], axis = 1)
df = df.drop(text_cols, axis=1)
scaler = MinMaxScaler()
scaled_values = scaler.fit_transform(df)
df_scaled = pd.DataFrame(scaled_values, columns=df.columns)
print(df_scaled.head())
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.metrics import accuracy_score
X = df_scaled.drop('target', axis=1)
y = df_scaled['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
svm_model = SVC()
lr_model = LogisticRegression()
rf_model = RandomForestClassifier()
models = {'SVM': svm_model,
'Logistic Regression': lr_model,
'Random Forest': rf_model
}
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    results[name] = accuracy

best_model = max(results, key=results.get)

best_accuracy = results[best_model]

print("Model Accuracies:")

for name, accuracy in results.items():

    print(f"{name}: {accuracy}")

print(f"\nBest Model: {best_model} with accuracy: {best_accuracy}")
from sklearn.decomposition import PCA

pca = PCA(n_components=10)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)

results_pca = {}

for name, model in models.items():

    model.fit(X_train_pca, y_train)

    y_pred_pca = model.predict(X_test_pca)

    accuracy_pca = accuracy_score(y_test, y_pred_pca)

    results_pca[name] = accuracy_pca

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

best_accuracy_pca = results_pca[best_model_pca]

print("\nModel Accuracies after PCA:")

for name, accuracy in results_pca.items():

    print(f"{name}: {accuracy}")

print(f"\nBest Model after PCA: {best_model_pca} with accuracy:

{best_accuracy_pca}")
```