VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Nandini A T (1BM23CS411)

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 **Nandini A T (1BM23CS411),** who is a 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.

Rajeshwari Madli Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

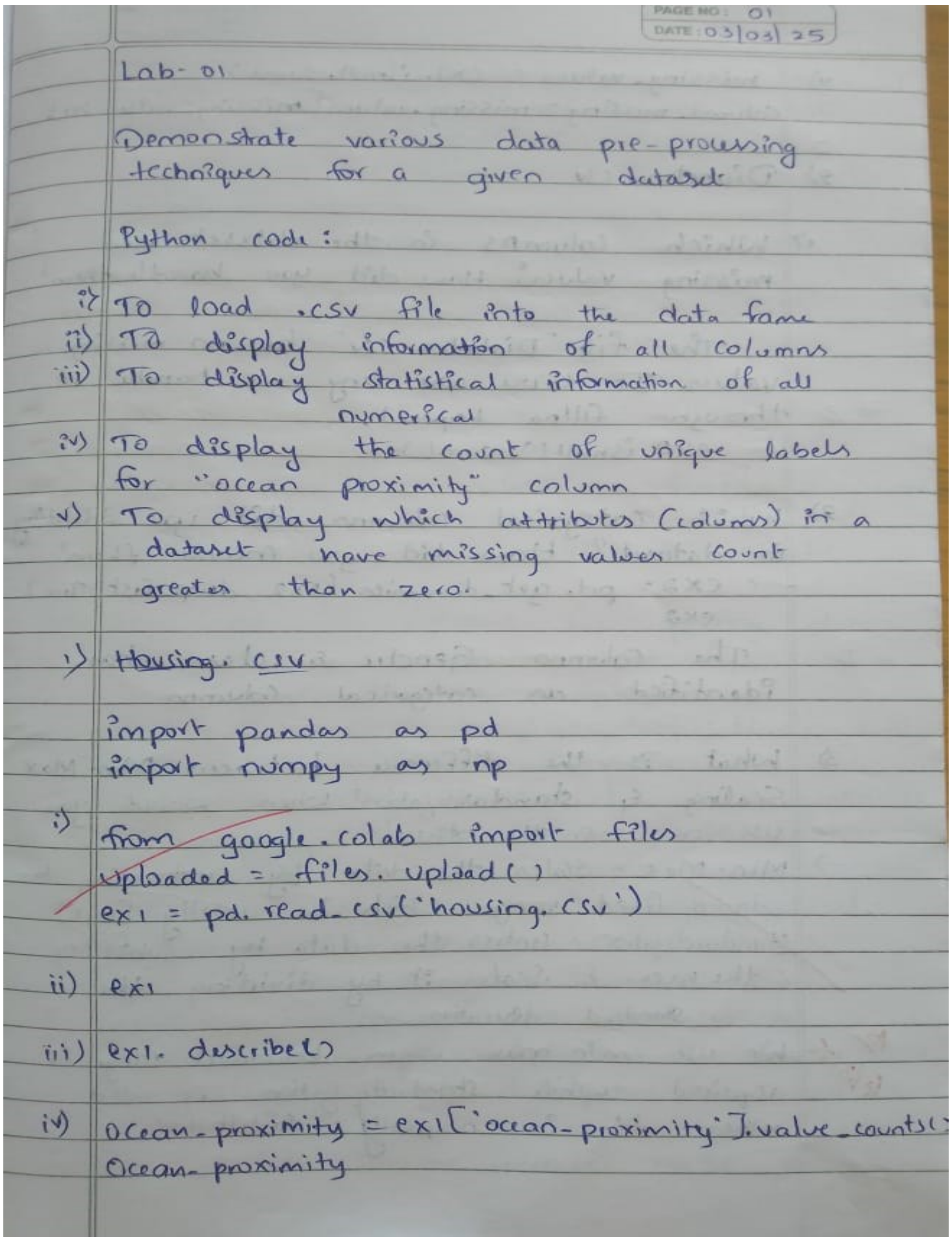
Index

Github Link:
https://github.com/NandiniAT/ML.git

Program 1

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

Screenshot:



PAGE NO: 01
DATE: 03/03/25

Lab- 01

Demonstrate various data pre-processing techniques for a given dataset

Python code :

i) To load .csv file into the data frame
ii) To display information of all columns
iii) To display statistical information of all numerical
iv) To display the count of unique labels for "ocean proximity" column
v) To display which attributes (colums) in a dataset have missing values count greater than zero.

1) Housing. csv

```
import pandas as pd
import numpy as np
```

i)
```
from google.colab import files
uploaded = files.upload( )
ex1 = pd.read_csv('housing.csv')
```

ii)
```
ex1
```

iii)
```
ex1.describe()
```

iv)
```
ocean_proximity = ex1['ocean_proximity'].value_counts()
ocean_proximity
```

PAGE NO: 2
DATE:

v) missing_values = ex1.isnull.sum()
Colums_missing = missing_values[missing_values>0].

2) Diabetes.csv

i) Which columns in the dataset had missing values? How did you handle them?

the file "Diabetes.csv" has no missing values. If there are any we handle it through fillna keyword,
ex2.isnull().sum()

2) Which categorical columns did you identify in dataset? How did you encode them?
→ ex2 = pd.get_dummies(ex2, drop_first=True)
ex2

The columns Gender & class has identified as categorical columns.

3) What is the difference between Min-Max Scaling & standardization? When would you use one over the other?
→ Min-Max - Scales the data by ~~subtracting the~~ to a fixed range which is usually [0,1]
Standardization - centres the data by subtracting the mean & Scales it by dividing the standard deviation.

6/3/25

We use min-max when bounded range is required while standardization is used when data is normally distributed.

Code:

Housing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
from google.colab import files
uploaded=files.upload()
ex1=pd.read_csv('housing.csv')
ex1
ex1.describe()
Ocean_proximity=ex1['ocean_proximity'].value_counts()
Ocean_proximity
```

```
missing_values = ex1.isnull().sum()
columns_missing_values = missing_values[missing_values > 0]
columns_missing_values
```

Diabetes

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats

from google.colab import files
uploaded=files.upload()

ex2=pd.read_csv('Dataset of Diabetes .csv')
ex2
ex2.describe()
# Handling missing values
ex2.isnull().sum()
#Handling categorical data
ex2 = pd.get_dummies(ex2, drop_first=True)
ex2
#Handling Outliers using IQR method (only for numerical columns)
print("Handling Outliers...")
numeric_cols = ex2.select_dtypes(include=[np.number]).columns
Q1 = ex2[numeric_cols].quantile(0.25)
Q1
Q3 = ex2[numeric_cols].quantile(0.75)
Q3
IQR = Q3 - Q1
IQR
lower_bound = Q1 - 1.5 * IQR
lower_bound
upper_bound = Q3 + 1.5 * IQR
upper_bound
ex2 = ex2[~((ex2[numeric_cols] < lower_bound) | (ex2[numeric_cols] > upper_bound)).any(axis=1)]
ex2
ex2.isnull().sum()
ex2.boxplot("AGE")
#Min-Max Scaling (Normalization)
print("Applying Min-Max Scaling...")
scaler = MinMaxScaler()
scaler
df_scaled = pd.DataFrame(scaler.fit_transform(ex2[numeric_cols]), columns=numeric_cols)
df_scaled
#Standard Scaling
print("Applying Standard Scaling...")
scaler = StandardScaler()
scaler
print("\nCleaned Data:")
(ex2.head())
print("\nNormalized Data:")
print(df_scaled.head())
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:

PAGE NO: 3
DATE: 10/03/25

Lab 2

Demonstrate the steps to build a ML model that predicts the median housing price using the given dataset

1. print (df.info()).
   print (df.describe())

2. print x label as 'columns'
   print y label as 'frequency'
   plot histogram for each plot
   interpretation:
   median income shows median income values of dataset indicates high/low income
   house median age shows distribution of median house age in different blocks.

3. test set is created by splitting data into training & testing set by random sampling. (or) stratified sampling.
   random sampling: samples datapoints randomly without considering distribution.
   stratified sampling: divides data into homogenous sub-groups based on some specific feature.

4. 'ocean_proximity' indicated geographical feature in dataset.

5. median_income feature correlates to maximum graph indicates positive correlation capped house values, dataspread & few outliers.

As median income increases median house values also increases +ve correlation there are a few outliers

6. Features that could be combined to improve correlation could be:
- rooms per household = total_rooms / households.
- bedrooms per room = total_bedrooms / total_rooms.
- population per household = population / household

conclusion - Correlation has remained constant even after combining features.

7. Features that need to be cleaned include total bedrooms, ocean-proximity, median house value cleaning of data is done by checking missing values, encoding categorical values, features scaling, outlier handling.

8. The ocean-proximity column is categorical that can be converted to numerical data by using One hot encoding to convert where the data is reshaped, transformed & then converted.

9. Feature scaling is highly important due to various factors such as
* improved model performance
* faster convergence
* Enhanced interpretability
* prevention of numerical issues

→ Common scaling techniques are:
* min-max scaling
* standardization
* robust scaling

PAGE NO: 5
DATE:

10. Numerical pipeline is done by imputation (fills missing value with median value), custom transformations (adding features), scaling categorical pipelining is done by one hot encoding.

10/3/25

Code:

```
from google.colab import files
diabetes=files.upload()
from google.colab import files
adult_income=files.upload()
df1=pd.read_csv("Dataset of Diabetes .csv")
df1.head()
df2=pd.read_csv("adult.csv")
df2.head()
df1.info()
df2.info()
```

```
df1.describe()
df2.describe()
missing_values1=df1.isnull().sum()
print(missing_values1)
missing_values2 = df2.isnull().sum()
print(missing_values2)
df1['Gender']=df1['Gender'].replace('f', 'F')
ordinal_encoder =OrdinalEncoder(categories=[["F",
"M"]])
df1["Gender_Encoded"] = ordinal_encoder.fit_transform(df1[["Gender"]])
onehot_encoder = OneHotEncoder()
encoded_data = onehot_encoder.fit_transform(df1[["CLASS"]])
encoded_array = encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df1, encoded_df], axis=1)
df1 = pd.concat([df1, encoded_df], axis=1)
df1.drop("CLASS", axis=1, inplace=True)
df1.drop("Gender", axis=1, inplace=True)
print(df2.head())
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
df_copy2 = df2
ordinal_encoder = OrdinalEncoder(categories=[["Male","Female"]])
df_copy2["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy2[["gender"]])
print(df_copy2[["gender","Gender_Encoded"]])

onehot_encoder = OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df2[["occupation","workclass","education","marital-status","relationship","race","n
ative-country","income"]])
encoded_array = encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["occupation","workclass","education","marital-status","relatio
nship","race","native-country","income"]))
df_encoded = pd.concat([df_copy2, encoded_df], axis=1)

df_encoded.drop("gender", axis=1, inplace=True)
df_encoded.drop("occupation", axis=1, inplace=True)
df_encoded.drop("workclass", axis=1, inplace=True)
df_encoded.drop("education", axis=1, inplace=True)
df_encoded.drop("marital-status", axis=1, inplace=True)
df_encoded.drop("relationship", axis=1, inplace=True)
df_encoded.drop("race", axis=1, inplace=True)
df_encoded.drop("native-country", axis=1, inplace=True)
df_encoded.drop("income", 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()
```

```
normalizer = MinMaxScaler()
df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" , "Chol","TG","HDL","LDL","VLDL","BMI"]] =
normalizer.fit_transform(df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,
"Chol","TG","HDL","LDL","VLDL","BMI"]])
df1.head()
```

<u>**Program 3**</u>

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:

PAGE NO : 6
DATE : 17/03/25

Lab-3

1. Implement Linear & Multi-Linear Regression algorithm using appropriate dataset

1. Solve the following Linear Regression Problem using Matrix approach. Find Linear Regression of the data of week & product sales.

x_i (Week)	y_j (Sales in Thousands)
1	2
2	4
3	5
4	9

$$\beta = (x^T x)^{-1} x^T) Y.$$

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

$$\rightarrow X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \qquad Y = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$

$$X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

$$X^T Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix} = \begin{bmatrix} 20 \\ 61 \end{bmatrix}$$

Inverse of $x^T x$

$$(x^T x)^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

→ compute $\beta = ((x^T x)^{-1} \cdot x^T) y$

$$= \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix} \begin{bmatrix} 20 \\ 61 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

Regression Line Equation $y = -0.5 + 2.2x$.

2. Simple Linear Regression

Diameter (X)	Price (Y)
8	10
10	13
12	16

Predict the price of 20 inch pizza using the data given

	X	Y	$X_i \cdot X_i$	$X_i Y_i$
	8	10	64	80
	10	13	100	130
	12	16	144	192
n =	30	39	308	402
$\bar{y}$ =	30/3=10	13	102.67	134

$$a_1 = \frac{(\overline{XY}) - (\bar{x})(\bar{y})}{(\overline{x_i^2}) - (\bar{x})^2}$$

$$= \frac{134 - (10)(13)}{(102.67) - 10^2} = \frac{134 - 130}{2.67} = \frac{4}{2.67} = 1.49$$

$a_1 \approx 1.5$

$a_0 = (\bar{y}) - a_1 \times \bar{x}$
$= 13 - 1.5 \times 10$
$= -2$

$y = -2 + 1.5x$

The prediction of price of 20 is

$y = -2 + 1.5(20)$
$y = 28$

After building the regression models, write the following questions.

1. Data Preprocessing Steps:
* Missing Values: Handled by imputing or dropping missing data
* Scaling: Applied to features with different scales to ensure uniformity
* Encoding: Categorical variables were encoded to convert them to numeric

2. Regression Line for Canada per-capita income. csv
Yes, the Regression line was plotted

3. Predicted Salary for Hiring Dataset
For a candidate with 12 years of experience, a 10 in test score & 10 in interview the model predicted a salary of $64629.625

4. Encoding: "State" was encoded using one-hot encoding. The rest of features were scaled to ensure consistent range & prevent larger features from dominating model

Code:

```
from google.colab import files
per_capita_income=files.upload()

from google.colab import files
salary=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
from sklearn import linear_model

df1=pd.read_csv("canada_per_capita_income.csv")
df1.head()

df2=pd.read_csv("salary.csv")
df2.head()
df2.YearsExperience.median()
df2.YearsExperience = df2.YearsExperience.fillna(df2.YearsExperience.median())
df2

plt.xlabel("year")
plt.ylabel("per capita income (US$)")
plt.scatter(df1.year, df1['per capita income (US$)'])

plt.xlabel("YearsExperience")
plt.ylabel("Salary")
plt.scatter(df2.YearsExperience, df2.Salary)

reg1 = linear_model.LinearRegression()
reg1.intercept_
reg1.predict([[2020]])

reg2 = linear_model.LinearRegression()
reg2.fit(df2.drop('Salary', axis='columns'), df2['Salary'])
reg2.coef_
reg2.intercept_
reg2.predict([[12]])

from google.colab import files
hiring=files.upload()

from google.colab import files
companies=files.upload()

df3=pd.read_csv("hiring.csv")
df3.head()

df4=pd.read_csv("1000_Companies.csv")
df4.head()
```

```
df3.isnull().sum()
df4.isnull().sum()

df3_copy = df3.copy()
experience_mapping = {'two': 2, 'three': 3, 'five': 5, 'seven': 7, 'ten': 10, 'eleven': 11}
df3_copy['experience'] = df3_copy['experience'].map(experience_mapping)
median_experience = df3_copy['experience'].median()
df3_copy['experience'] = df3_copy['experience'].fillna(median_experience)
df3_copy
df3_copy['test_score(out of 10)'] = df3_copy['test_score(out of 10)'].fillna(df3_copy['test_score(out of
10)'].mean())
reg3 = linear_model.LinearRegression()
reg3.fit(df3_copy.drop('salary($)', axis='columns'), df3_copy['salary($)'])
reg3.coef_
reg3.intercept_
reg3.predict([[2,9,6]])
reg3.predict([[12,10,10]])
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
state_encoded = ohe.fit_transform(df4[['State']])
state_encoded_df = pd.DataFrame(state_encoded, columns=ohe.get_feature_names_out(['State']))

df4 = pd.concat([df4, state_encoded_df], axis=1).drop(columns=['State'])
print(df4)
reg4 = linear_model.LinearRegression()
reg4.fit(df4.drop('Profit',axis='columns'),df4.Profit)
print(reg4.coef_)
print(reg4.intercept_)
reg4.predict([[91694.48, 515841.3, 11931.24,0,1,0]])
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:

PAGE NO: 9
DATE: 24/03/25

Lab-04 Logistic Regression

Consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The logistic regression model has been trained & the learned parameters are $a_0 = -5$ (intercept) & $a_1 = 0.8$ (coefficient for study hours)

1. Write logistic regression equation.

$$P(pass|x) = \frac{1}{1+e^{-(a_0+a_1x)}}$$

Given: $a_0 = -5$, $a_1 = 0.8$

$$p(pass|x) = \frac{1}{1+e^{-(-5+0.8x)}}$$

Calculate probability for $x = 7$ hours

$$p(pass|7) = \frac{1}{1+e^{-(-5+0.8\times 7)}}$$

Step1: compute linear combination

$z = a_0 + a_1 x$

$z = -5+0.8\times 7 = -5+5.6 = 0.6$

Step2: Plug z into logistic funⁿ

$$P(pass|7) = \frac{1}{1+e^{-0.6}}$$

Step3: Compute $e^{-0.6}$

$e^{-0.6} \approx 0.5488$

Step4: Calculate probability

$$P(pass|7) = \frac{1}{1+0.5488} = \frac{1}{1.5488} \approx 0.6457$$

So, the probability that student will pass approximately is 64.57%

PAGE NO: 10
DATE:

3. Determine the predicted class using a threshold of 0.5

If $P(pass|x) \geq 0.5$, predict pass

If $P(pass|x) < 0.5$, predict fail

for $x = 7$ hours

$P(pass|7) \approx 0.645 \geq 0.5$

Thus, the predict class is Pass.

4. Consider $z = [2, 1, 0]$ for 3 classes. Apply soft max funⁿ to find prob. of 3 classes.

Solⁿ: $$\text{softMax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$z = [2, 1, 0]$

1. Exponentiate each value

$e^z = [e^2, e^1, e^0] = [7.3891, 2.7183, 1]$

2. Sum up the exponentiated values:

$sum = 7.3891 + 2.7183 + 1 = 11.1074$

3. Divide each exponentiated value by sum:

$\text{softmax}(z_0) = \frac{7.3891}{11.1074} \approx 0.6652$ class 0

$\text{softmax}(z_1) = \frac{2.7183}{11.1074} \approx 0.2447$ class 1

$\text{softmax}(z_2) = \frac{1}{11.1074} \approx 0.0901$ class 2

→ For dataset file "HR-comma-sep.csv".

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

* The key variables impact the employees retention are: Satisfaction level - lower satisfaction level increases
* Time spent in company - employees 5+ years tend to leave

PAGE NO: 11
DATE:

* salary - low salaries lead to higher turnover
* No. of projects & avg monthly hours - very high or low values affect retention.

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

The accuracy of logistic regression model is 78.40%. This accuracy is overall good. But the model still needs an improvement. The model captures key factors.

R 7/4/25

Code:

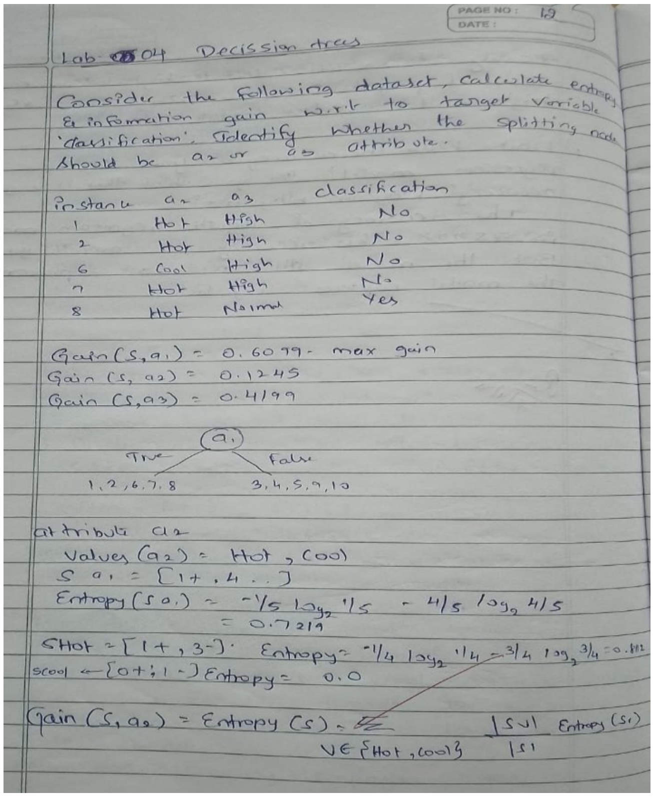
```
from google.colab import files
hr=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
from sklearn import linear_model
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df1=pd.read_csv("HR_comma_sep.csv")
df1.head()
df1.isnull().sum()
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()

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()
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()
```

```
df_copy = df1[['number_project', 'average_montly_hours', 'time_spend_company', 'left','salary_encoded',
'satisfaction_level','Work_accident']]
df_copy.head()
X = df_copy.drop('left', axis=1)
y = df_copy['left']
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 of the Logistic Regression model: {accuracy}")

from google.colab import files
zoodata=files.upload()
zootype=files.upload()

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}")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap="Blues", values_format="d")
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:



PAGE NO: 12

Lab 04 Decission trees

Consider the following dataset, calculate entropy & information gain w.r.t to target variable 'classification'. Identify whether the splitting node should be a_2 or a_3 attribute.

instance	a_2	a_3	classification
1	Hot	High	No
2	Hot	High	No
6	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes

$Gain(S, a_1) = 0.6099$ - max gain

$Gain(S, a_2) = 0.1245$

$Gain(S, a_3) = 0.4199$

a_1

True: 1, 2, 6, 7, 8

False: 3, 4, 5, 9, 10

attribute a_2

Values (a_2) = Hot, Cool

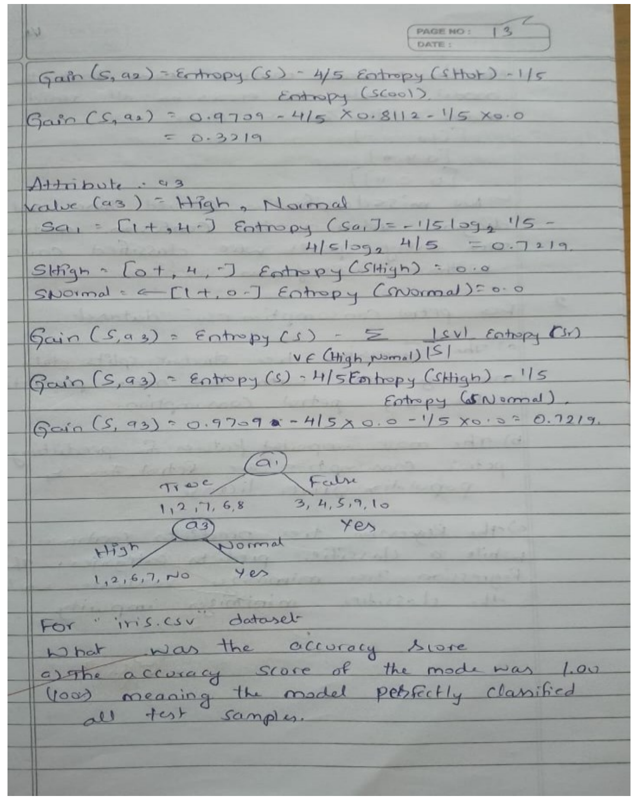
$S_{a_1} = [1+, 4-]$

$Entropy(S_{a_1}) = -1/5 \log_2 1/5 - 4/5 \log_2 4/5 = 0.7219$

$S_{Hot} = [1+, 3-]$. $Entropy = -1/4 \log_2 1/4 - 3/4 \log_2 3/4 = 0.8112$

$S_{Cool} \leftarrow [0+, 1-]$ $Entropy = 0.0$

$$Gain(S, a_2) = Entropy(S) - \sum_{v \in \{Hot, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$



PAGE NO: 13

$Gain(S, a_2) = Entropy(S) - 4/5\ Entropy(S_{Hot}) - 1/5\ Entropy(S_{Cool})$

$Gain(S, a_2) = 0.9709 - 4/5 \times 0.8112 - 1/5 \times 0.0 = 0.3219$

Attribute: a_3

value (a_3) = High, Normal

$S_{a_1} = [1+, 4-]$ $Entropy(S_{a_1}) = -1/5 \log_2 1/5 - 4/5 \log_2 4/5 = 0.7219$

$S_{High} = [0+, 4-]$ $Entropy(S_{High}) = 0.0$

$S_{Normal} \leftarrow [1+, 0-]$ $Entropy(S_{Normal}) = 0.0$

$$Gain(S, a_3) = Entropy(S) - \sum_{v \in (High, Normal)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$Gain(S, a_3) = Entropy(S) - 4/5\ Entropy(S_{High}) - 1/5\ Entropy(S_{Normal})$

$Gain(S, a_3) = 0.9709 - 4/5 \times 0.0 - 1/5 \times 0.0 = 0.7219$

a_1

True: 1, 2, 7, 6, 8 → a_3

False: 3, 4, 5, 9, 10 → Yes

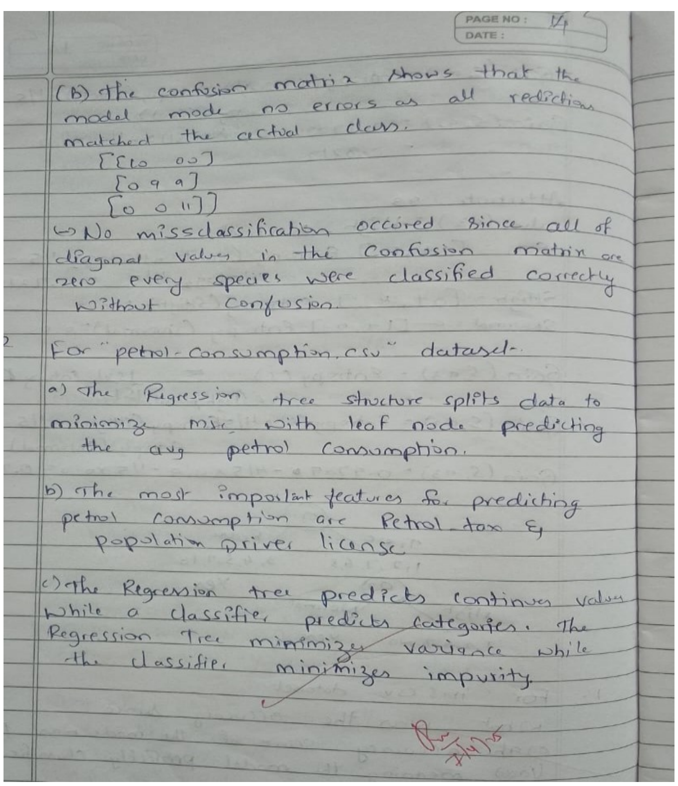
a_3 High: 1, 2, 6, 7, No

a_3 Normal: Yes

For "iris.csv" dataset

What was the accuracy score

a) The accuracy score of the mode was 1.00 (100) meaning the model perfectly classified all test samples.



PAGE NO: 14

(B) The confusion matrix shows that the model made no errors as all predictions matched the actual class.

[[10 0 0]
[0 9 0]
[0 0 11]]

→ No missclassification occured since all of diagonal values in the confusion matrix are zero every species were classified correctly without confusion.

2 For "petrol-consumption.csv" dataset.

a) The Regression tree structure splits data to minimize mse with leaf node predicting the avg petrol consumption.

b) The most important features for predicting petrol consumption are Petrol tax & population driver license.

c) The Regression tree predicts continues values while a classifier predicts categories. The Regression Tree minimizes variance while the classifier minimizes impurity.

Code:

```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris.csv")
df1.head()

df1.isnull().sum()

X = df1.drop('species', axis=1)
y = df1['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))
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=y.unique())
plt.show()

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
cmap = plt.cm.get_cmap('PuBuGn')
disp.plot(cmap=cmap)
plt.show()

drug=files.upload()
df2=pd.read_csv("drug.csv")
df2.head()
df2.isnull().sum()

label_encoders = {}
for column in df2.columns:
    le = LabelEncoder()
    df2[column] = le.fit_transform(df2[column])
    label_encoders[column] = le
X = df2.drop('Drug', axis=1)
y = df2['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))
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=[str(c) for c in y.unique()])
plt.show()

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
```

```
cmap = plt.cm.Blues
disp.plot(cmap=cmap)
plt.show()

pc=files.upload()
df3=pd.read_csv("petrol_consumption.csv")
df3.head()
df3.isnull().sum()
X = df3.drop('Petrol_Consumption', axis=1)
y = df3['Petrol_Consumption']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'Mean Absolute Error: {mae:.2f}')
print(f'R-squared: {r2:.2f}')
plt.figure(figsize=(30, 30))
plot_tree(regressor, filled=True, feature_names=X.columns, fontsize=10)
plt.show()
```

<u>**Program 6**</u>

Build KNN Classification model for a given dataset.

Screenshot:

PAGE NO: 15
DATE: 5/7/0/25

Lab - 05

KNN classification Model

* Consider the following dataset. for K = 3 & test data (x, 35, 100) as (Person, Age, Salary k) solve using KNN classifier model & predict the target

Person	Age	Salary k	Target
A	18	50	N
B	23	55	N
C	24	70	N
D	41	60	Y
E	43	70	Y
F	38	40	Y
x	35	100	?

distance		rank
$\sqrt{(35-18)^2 + (100-50)^2}$	$= 52.81$	5
$\sqrt{(35-23)^2 + (100-55)^2}$	$= 46.57$	4
$\sqrt{(35-24)^2 + (100-70)^2}$	$= 31.95$	2
$\sqrt{(35-41)^2 + (100-60)^2}$	$= 40.45$	3
$\sqrt{(35-43)^2 + (100-70)^2}$	$= 31.05$	1
$\sqrt{(35-38)^2 + (100-40)^2}$	$= 61.07$	6

~~$\sqrt{(35-35)^2}$~~

K = 1 target = Y

K = 2 target = N

K = 3 target = Y

target for Person X = Y

i) ~~for iris dataset~~

PAGE NO: 16
DATE:

* How to choose the k value? Demonstrate using accuracy rate & error rate.

Soln: Best k value in iris data set =
k values from 1 to 20 are tested for accuracy and error rate the k value of 3 was choosen because of high accuracy & low error rate.

2. For diabetes dataset
* What is the purpose of feature scaling? how to perform it?

Soln: Feature scaling in diabetes to ensure that all the features like glucose, age, insulin are on the same scale so that KNN does not get biased by large numerical values.

Code:

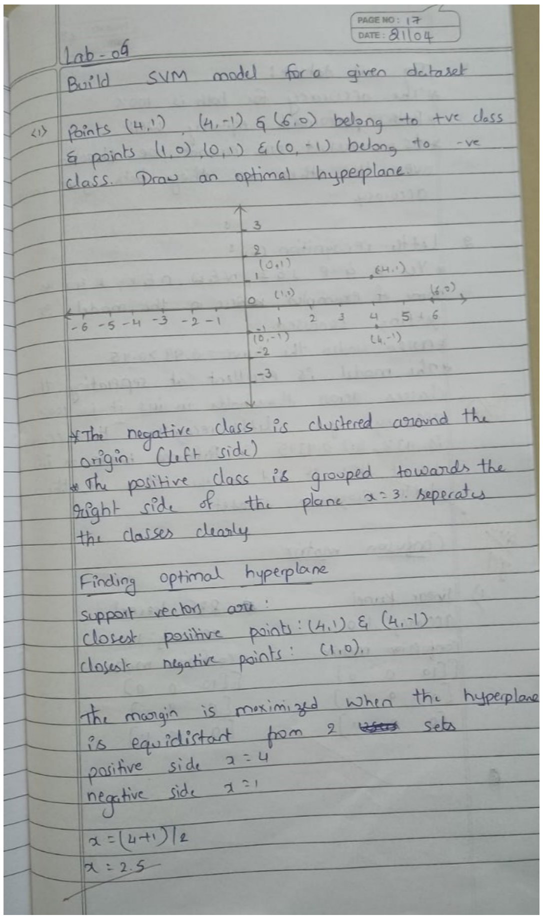
```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (2).csv")
df1.head()
df1.isnull().sum()
X = df1.drop('species', axis=1)
y = df1['species']

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)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

diabetes=files.upload()
df2=pd.read_csv("diabetes.csv")
df2.head()
df2.isnull().sum()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df2.drop('Outcome', axis=1))
X_train, X_test, y_train, y_test = train_test_split(X_scaled, df2['Outcome'], 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}")
    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')


plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
print("\nClassification Report:")
print(classification_report(y_test,  y_pred))
heart=files.upload()
df3=pd.read_csv("heart.csv")
df3.head()
df3.isnull().sum()
X = df3.drop('target', axis=1)
y = df3['target']
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=optimal_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Program 7

Build Support vector machine model for a given dataset

Screenshot:



PAGE NO: 17
DATE: 21/04

Lab - 09

Build SVM model for a given dataset

<1> Points (4,1), (4,-1) & (6,0) belong to +ve class & points (1,0) (0,1) & (0,-1) belong to -ve class. Draw an optimal hyperplane

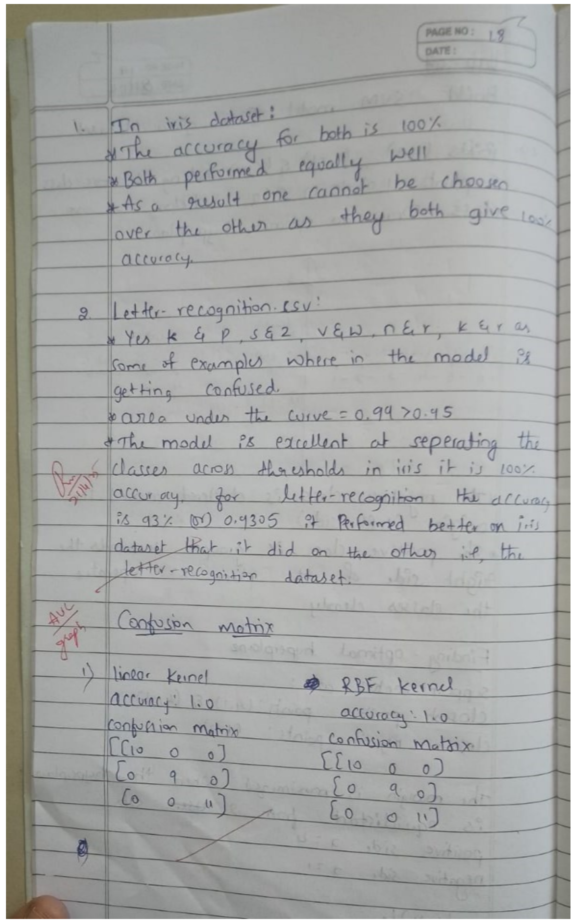
* The negative class is clustered around the origin (left side)
* The positive class is grouped towards the right side of the plane x=3 seperates the classes clearly

Finding optimal hyperplane

Support vectors are:
closest positive points: (4,1) & (4,-1)
closest negative points: (1,0).

The margin is maximized when the hyperplane is equidistant from 2 sets
positive side x=4
negative side x=1

x=(4+1)/2
x=2.5



PAGE NO: 18
DATE:

1. In iris dataset:
* The accuracy for both is 100%
* Both performed equally well
* As a result one cannot be choosen over the other as they both give 100% accuracy.

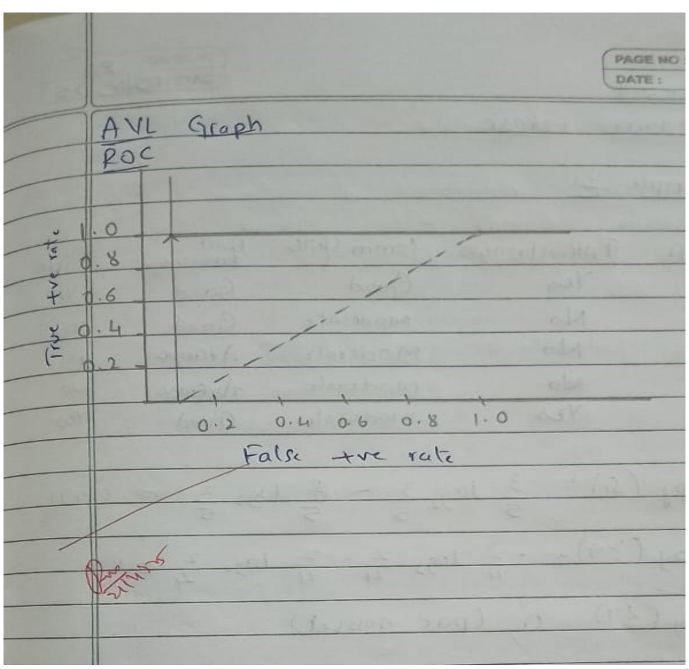
2. Letter-recognition.csv:
* Yes k & p, s & z, v & w, n & r, k & r as some of examples where in the model is getting confused.
* area under the curve = 0.99 > 0.95
* The model is excellent at seperating the classes across thresholds in iris it is 100% accuracy. for letter-recognition the accuracy is 93% (or) 0.9305 it performed better on iris dataset that it did on the other i.e, the letter-recognition dataset.

AUC graph

Confusion matrix

1) linear kernel
accuracy: 1.0
confusion matrix
[[10 0 0]
[0 9 0]
[0 0 11]]

RBF kernel
accuracy: 1.0
confusion matrix
[[10 0 0]
[0 9 0]
[0 0 11]]



PAGE NO
DATE:

AVL Graph
ROC

True +ve rate: 1.0, 0.8, 0.6, 0.4, 0.2
0.2 0.4 0.6 0.8 1.0
False +ve rate

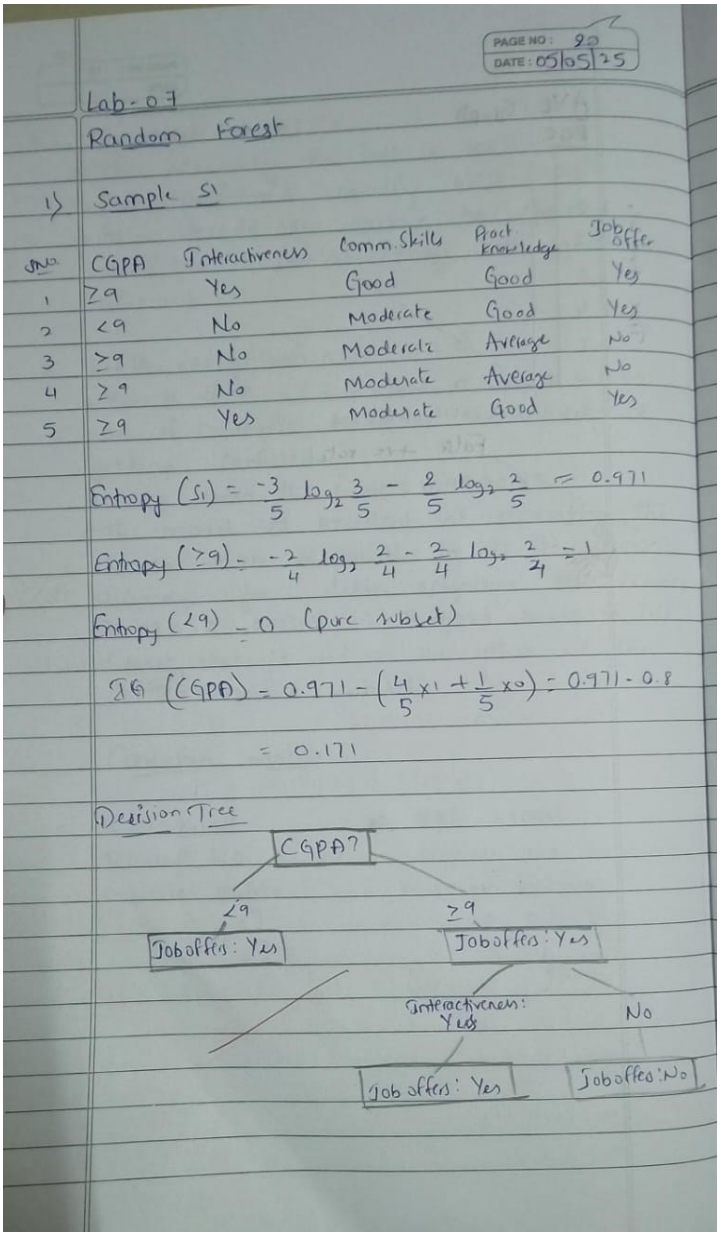
Code:

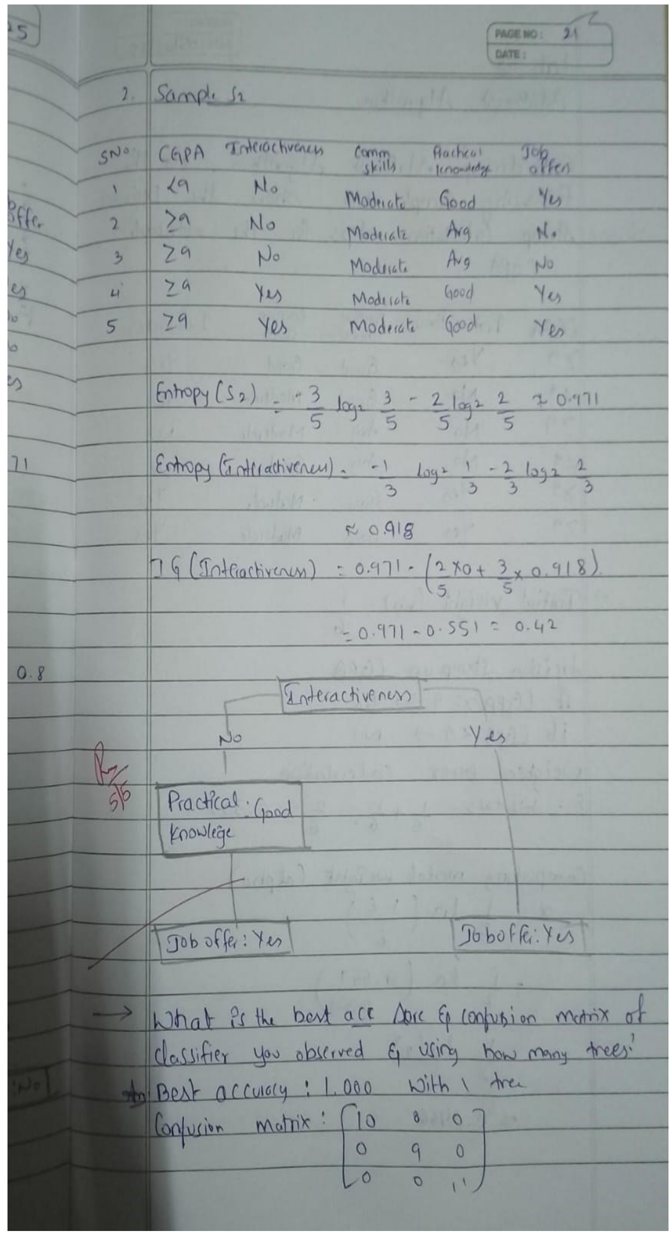
```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (1).csv")
df1.head()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
rbf_y_pred = rbf_svm.predict(X_test)
print("RBF Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, rbf_y_pred))
cm = confusion_matrix(y_test, rbf_y_pred)
sns.heatmap(cm, annot=True, fmt='d',cmap="Blues")
plt.title('Confusion Matrix for RBF Kernel SVM')
plt.xlabel('Predicted')

plt.ylabel('True')
plt.show()
print(classification_report(y_test, rbf_y_pred))
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
linear_y_pred = linear_svm.predict(X_test)
print("\nLinear Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, linear_y_pred))
cm = confusion_matrix(y_test, linear_y_pred)
sns.heatmap(cm, annot=True, fmt='d',cmap="Blues")
plt.title('Confusion Matrix for Linear Kernel SVM')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print(classification_report(y_test, linear_y_pred))
letter=files.upload()
df2=pd.read_csv("letter-recognition.csv")
df2.head()
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)
plt.figure(figsize=(10,10))
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:





Code:
```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (4).csv")
df1.head()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
rf_classifier = RandomForestClassifier(random_state=0)
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: {default_accuracy}")
best_accuracy = 0
best_n_estimators = 0
for n_estimators in range(1, 101):
    rf_classifier = RandomForestClassifier(n_estimators=n_estimators, random_state=0)
    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"\nBest accuracy: {best_accuracy} achieved with n_estimators = {best_n_estimators}")
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 9

Implement Boosting ensemble method on a given dataset

Screenshot:

DATE : 05/05/25

Lab-08

AdaBoost Algorithm

Considering AdaBoost Algorithm. For the following sample data, show the Decision stump calculations steps for the attribute CGPA

CGPA	Interactiveness	Practical knowledge	Comm skills	Job Profile
≥9	Yes	Good	Good	Yes
<9	No	Good	Moderate	Yes
≥9	No	Avg	Moderate	No
<9	No	Avg	Good	No
≥9	Yes	Good	Moderate	Yes
≥9	Yes	Good	Moderate	Yes

→ Initial weight $(w_i) = \frac{1}{6}$

decision stump on CGPA

if CGPA >= 9 → Yes

if CGPA < 9 → No

weighted error calculation

$\varepsilon = w_2 + w_5 = \frac{1}{6} + \frac{1}{6} = \frac{2}{6} = 0.333$

Computing model weight (alpha)

$\alpha = \frac{1}{2} \ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$

$= \frac{1}{2} \ln\left(\frac{0.667}{0.333}\right)$

$= \frac{1}{2} \ln(2)$

≈ 0.3466

$z_{CGPA} = \frac{1}{6} \times 4 \times e^{-0.3466} + \frac{1}{6} \times 2 \times e^{0.3466}$

$= 0.9428$

2. Best accuracy score & confusion matrix of classifier & how many trees?

Best accuracy score = 0.8385

with n estimation = 80

Confusion matrix = $\begin{bmatrix} 7130 & 284 \\ 1343 & 1012 \end{bmatrix}$

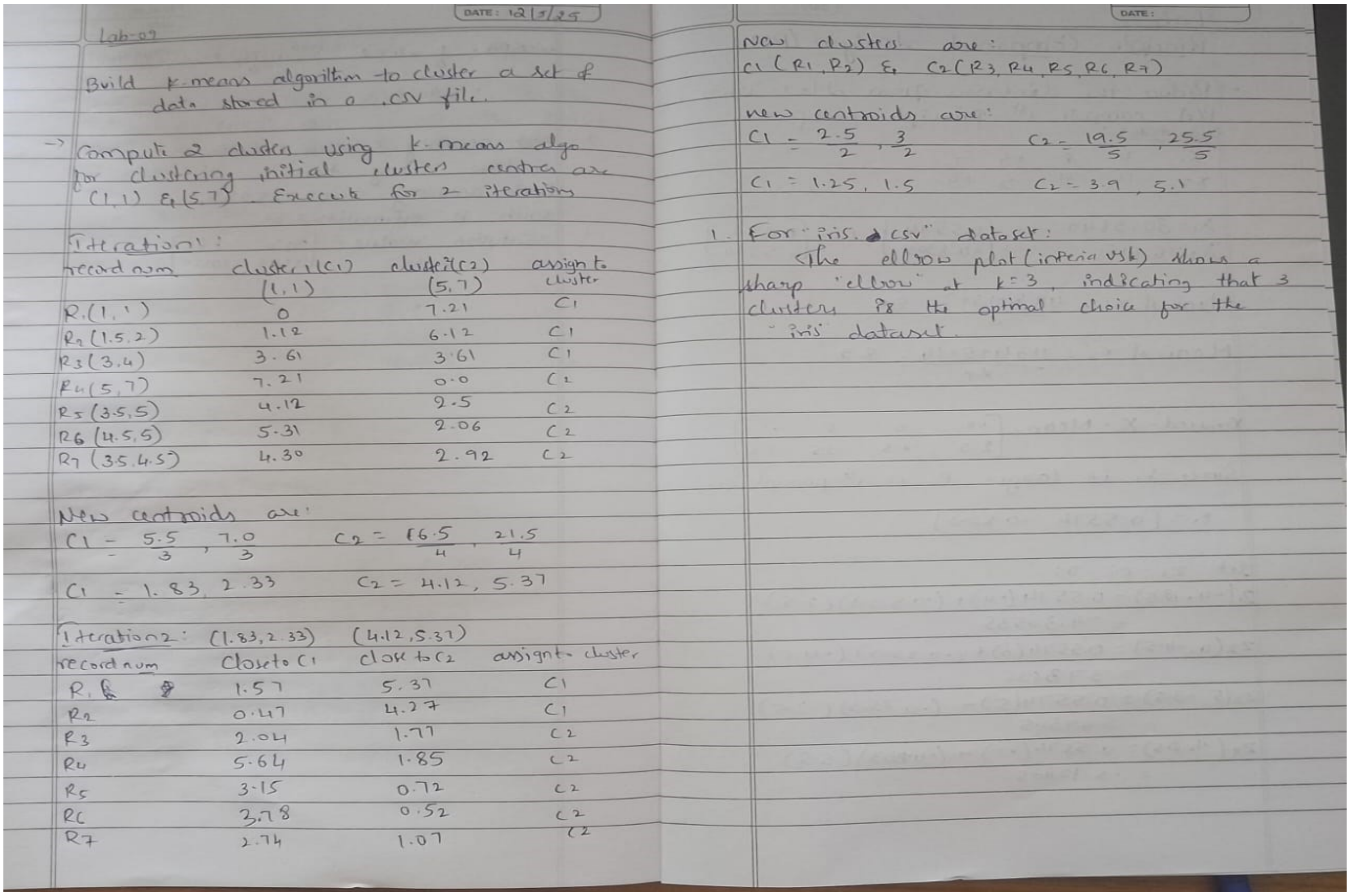
Code:

```
from google.colab import files
income=files.upload()
df1=pd.read_csv("income.csv")
df1.head()
X = df1.drop('income_level', axis=1)

y = df1['income_level']
X = pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
abc = AdaBoostClassifier(n_estimators=10, random_state=42)
abc.fit(X_train, y_train)
y_pred = abc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Initial AdaBoost accuracy (10 trees): {accuracy}")
param_grid = {'n_estimators': [50, 100, 150, 200]}
grid_search = GridSearchCV(AdaBoostClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_}")
best_abc = grid_search.best_estimator_
y_pred_best = best_abc.predict(X_test)
best_accuracy = accuracy_score(y_test, y_pred_best)
print(f"Accuracy of the best model on the test set: {best_accuracy}")
cm = confusion_matrix(y_test, y_pred_best)
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')
plt.show()
```

Program 10

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

Screenshot:



Lab-07 DATE: 12/3/25

Build k-means algorithm to cluster a set of data stored in a .csv file.

→ Compute 2 clusters using k-means algo for clustering, initial cluster centres are (1,1) & (5,7). Execute for 2 iterations

Iteration1:

record num	cluster1(C1) (1,1)	cluster2(C2) (5,7)	assign to cluster
R1(1,1)	0	7.21	C1
R2(1.5,2)	1.12	6.12	C1
R3(3,4)	3.61	3.61	C1
R4(5,7)	7.21	0.0	C2
R5(3.5,5)	4.12	2.5	C2
R6(4.5,5)	5.31	2.06	C2
R7(3.5,4.5)	4.30	2.92	C2

New centroids are:

$C1 = \frac{5.5}{3}, \frac{7.0}{3}$ $C2 = \frac{16.5}{4}, \frac{21.5}{4}$

$C1 = 1.83, 2.33$ $C2 = 4.12, 5.37$

Iteration2: (1.83, 2.33) (4.12, 5.37)

record num	Close to C1	close to C2	assign to cluster
R1	1.57	5.37	C1
R2	0.47	4.27	C1
R3	2.04	1.77	C2
R4	5.64	1.85	C2
R5	3.15	0.72	C2
R6	3.78	0.52	C2
R7	2.74	1.07	C2

New clusters are:

C1 (R1, R2) & C2 (R3, R4, R5, R6, R7)

new centroids are:

$C1 = \frac{2.5}{2}, \frac{3}{2}$ $C2 = \frac{19.5}{5}, \frac{25.5}{5}$

$C1 = 1.25, 1.5$ $C2 = 3.9, 5.1$

1. For "iris.csv" dataset:

The elbow plot (inertia vs k) shows a sharp "elbow" at k=3, indicating that 3 clusters is the optimal choice for the "iris" dataset.

Code:

```
from google.colab import files
iris=files.upload()
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 (4).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:

Lab-10

Principle Component Analysis (PCA)

→ Reduce the dimension from 2 to 1 using PCA compute 1st principle component

Data matrix:

x_1	4	8	13	7
x_2	11	4	5	14

$\lambda_1 = 30.3849$ $e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$, $e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

$\lambda_2 = 6.6151$

Mean of $x_1 = \frac{4+8+13+7}{4} = 8$

Mean of $x_2 = \frac{11+4+5+14}{4} = 8.5$

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

Since λ_1 is larger, e_1 is 1st principle component

$e_1 = [0.5574, -0.8303]$

let $z_i = e_i \cdot x_i$

$z_1(-4, 2.5) = 0.5574(-4) + (-0.8303)(2.5)$
$= -4.30535$

$z_2(0, -4.5) = 0.5574(0) + (-0.8303)(-4.5)$
$= 3.73635$

$z_3(5, -3.5) = 0.5574(5) + (-0.8303)(-3.5)$
$= 5.69305$

$z_4(-1, 5.5) = 0.5574(-1) + (-0.8303)(5.5)$
$= -5.12405$

$z = \begin{bmatrix} -4.305 \\ 3.736 \\ 5.693 \\ -5.124 \end{bmatrix}$

1. For "heart.csv" dataset:

* Accuracy before PCA:
  - Logistic regression: 0.9016
  - SVM: 0.8525
  - Random Forest: 0.8361

* Accuracy after PCA:
  - Logistic regression: 0.8689
  - SVM: 0.8689
  - Random Forest: 0.8852

Code:

```
from google.colab import files
heart=files.upload()

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 (1).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]}")
```