VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum- 590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Jeevan A (1BM22CS119)

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

February 2025 – July 25

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 **Jeevan A (1BM22CS119),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Laboratory report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Srushti C S	Dr. Kavitha Sooda
Assistant Professor	Professor & HOD
Department of CSE, BMSCE	Department of CSE, BMSCE

INDEX

Github Link: Jeevan-017/ML-LAB

LABORATORY PROGRAM – 1

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

OBSERVATION BOOK

Date 03/03/25
Page

LABORATORY - 01

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

(i) To load .csv file into the data frame

```
⇒ import pandas as pd

filename = "housing.csv"
df = pd.read_csv(filename)
```

(ii) To display information of all column

```
⇒ print ("Information of all columns")
df.info()
```

(iii) To display the statistical information of all numerical column

```
⇒ print (df.describe())
```

(iv) To display the count of unique labels for "Ocean Proximity" column

```
⇒ print (df['Ocean Proximity'].value_counts())
```

(v) To display which columns in a dataset have missing values count greater than zero

```
⇒ print (df.isnull().sum() [df.isnull().sum > 0])
```

Dataset : "Diabetes.csv" and "Adult.csv"

Questions :-

1. Which column in the dataset had missing values? How did you handle them

```
⇒ print ("Missing values in Dataset diabetes:")
print (diabetes_df.isnull().sum())
```

The code checks for missing values (NaN) in each column of the diabetes dataset and prints the count of missing values for each column.

Numerical columns are filled with the mean of the column, and categorical column with the mode.

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

```
⇒ categ_cols = adult_income_df.select_dtypes
               (include=['object']).columns.tolist()
```

3. What is the difference between Min-Max scaling and standardization? When would you use one over the other?

Page

⇒ Min-Max scaling scales the data to a specific range (usually 0 to 1). It's sensitive to outliers.

Standardization transforms data to have a mean of 0 and std dev of 1. It's less sensitive to outliers.

Usage :-

- Min-Max : Use when algorithms are sensitive to feature magnitudes and when data has no significant outliers.
- Standardization : For gaussian data.

3/3/25

CODE WITH OUTPUT

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

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
0	502	17975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
1	735	34221	M	26	4.5	62	4.9	3.7	1.4	1.1	2.1	0.6	23.0	N
2	420	47975	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
3	680	87656	F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	24.0	N
4	504	34223	M	33	7.1	46	4.9	4.9	1.0	0.8	2.0	0.4	21.0	N

```
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)
if len(categorical_cols) > 0:
        df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
        print("\nDataFrame after one-hot encoding:")
         print(df.head())
        else:
         print("\nNo categorical columns found in the dataset.")
```

```
Categorical columns identified: Index(['Gender', 'CLASS'], dtype='object')

DataFrame after one-hot encoding:
    ID  No_Pation  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL   BMI  \
0  502      17975   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
1  735      34221   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6  23.0
2  420      47975   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
3  680      87656   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5  24.0
4  504      34223   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4  21.0

   Gender_M  Gender_f  CLASS_N  CLASS_P  CLASS_Y  CLASS_Y
0     False     False    False    False    False     False
1      True     False    False    False    False     False
2     False     False    False    False    False     False
3     False     False    False    False    False     False
4      True     False    False    False    False     False
```

LABORATORY PROGRAM – 2

Demonstrate various data pre-processing techniques for a given dataset

OBSERVATION BOOK

LABORATORY — 02

Steps to build a ML- model that predict the housing price using the California housing price dataset

1. Perform the describe and info steps

```
housingData = pd.read_csv(FilePath)
housingData.describe()
housingData.info()
```

2. Indicate what does histogram indicate on median income and house_median_age

⇒ The histogram is right - skewed, meaning that most households have a lower median income.

3.⇒ house_median_age ⇒ The distribution is bimodal with two peaks.

The presence of sharp spikes suggest that certain constructions booms at specific times.

3. Write the difference B/W random and stratified test set

⇒ Random test set :- It splits the data randomly without considering feature distribution.

Stratified test set :- It splits data while preserving the distribution of key features.

4. What does the graph indicate w.r.t housing prices and location

=> Housing prices prices are strongly location dependent, with higher prices usually for locations closer to coastal cities and urban areas.

5. Analyze what the housing price graph indicate

=> Areas with higher median income tends to have more expensive homes.

6. List the features that could be combined to improve correlation.

=> The features/attributes that can be added :-

•

• Rooms Per Household
• Bedrooms Per Room
• Population Per Household

7. List the features that needs to be cleaned

=> total_bedroom has missing values
Ocean_proximity has a categorical feature
median_house value has capped value.

8. Discuss the importance of feature scaling

=> • Mainly used for achieving model convergence
• Can achieve regularization
• Prevents features with large values from overpowering others.
• Stable training

10/3/25

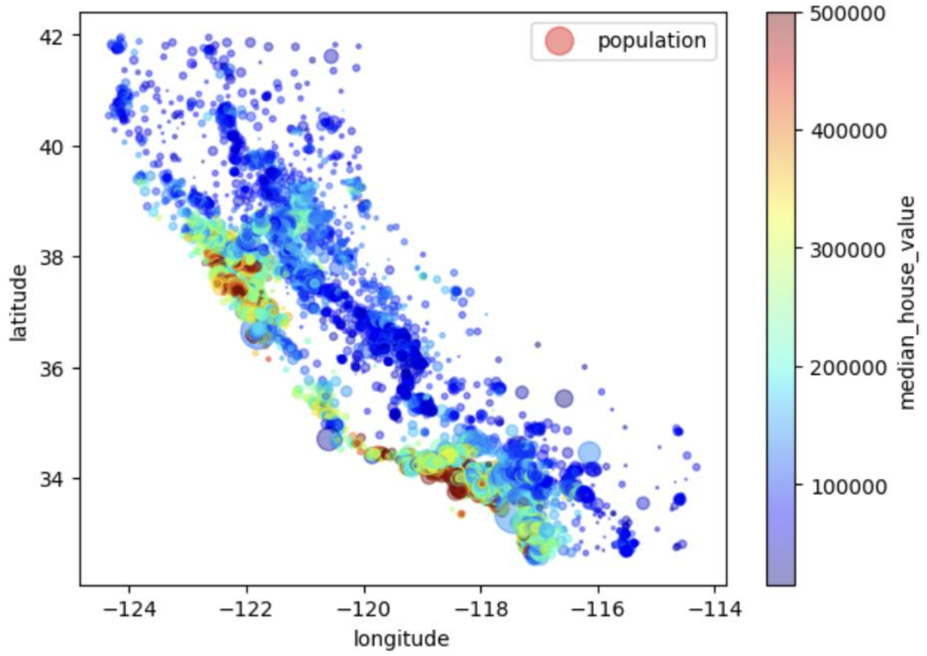
CODE WITH OUTPUT

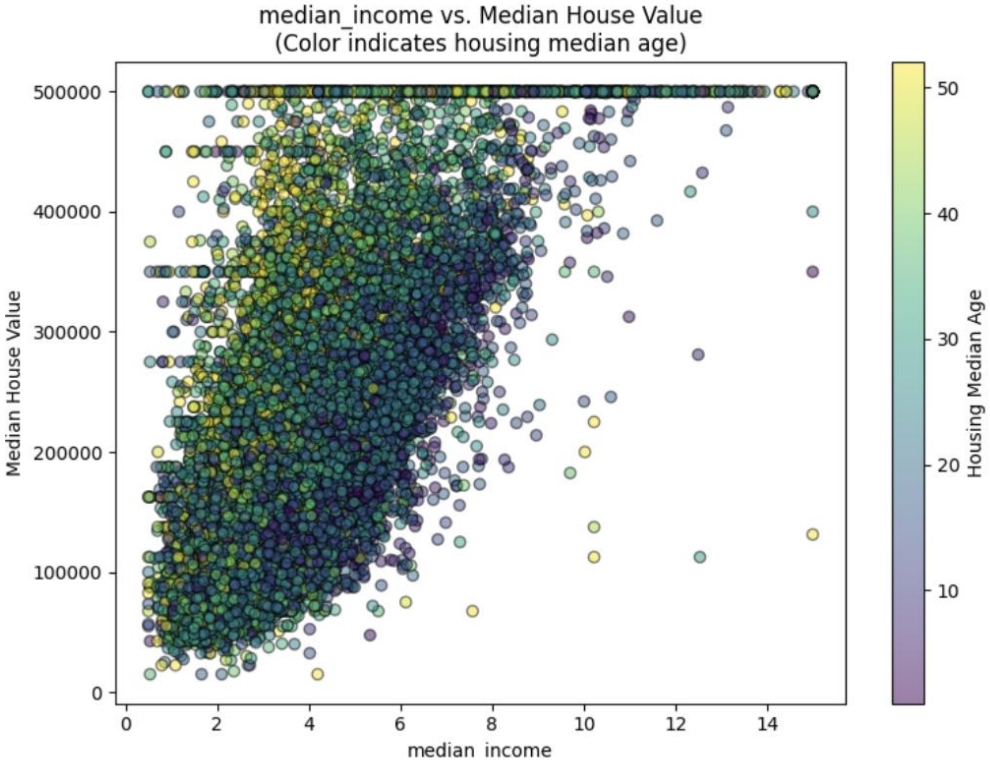
```
# Load the dataset into a pandas DataFrame
df = pd.read_csv('housing.csv')

# Display descriptive statistics
df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatior
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.00000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.47674
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.46212
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.00000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.00000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.00000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.00000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.00000

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit

# Load the dataset
```

```python
housing = pd.read_csv('housing.csv')

# For this demonstration, consider only 'median_income' and 'median_house_value'
housing_selected = housing[['median_income', 'median_house_value']].copy()

# Random split: This splits the data randomly without preserving any specific distribution.
train_set_random, test_set_random = train_test_split(housing_selected, test_size=0.2,
random_state=42)

# For stratified sampling, first create an income category.
housing_selected['income_cat'] = pd.cut(housing_selected['median_income'],
                                        bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                                        labels=[1, 2, 3, 4, 5])

# Use StratifiedShuffleSplit to ensure the income distribution is preserved in both sets.
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing_selected, housing_selected['income_cat']):
    strat_train_set = housing_selected.loc[train_index]
    strat_test_set = housing_selected.loc[test_index]

# Remove the temporary income category attribute.
for dataset in (strat_train_set, strat_test_set):
    dataset.drop("income_cat", axis=1, inplace=True)
```






```
from sklearn.preprocessing import OneHotEncoder

# Extract the categorical attribute
housing_cat = housing[["ocean_proximity"]]

# Perform one-hot encoding
encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat).toarray()

# Create a DataFrame for the encoded features
housing_cat_1hot_df = pd.DataFrame(housing_cat_1hot,
                                   columns=encoder.get_feature_names_out(["ocean_proximity"]))
housing_cat_1hot_df.head()
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

# Custom transformer to add engineered attributes
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        # Assumes X is a NumPy array with the following columns:
        # total_rooms (index 3), total_bedrooms (index 2), population (index 4), households
(index 5)
        rooms_per_household = X[:, 3] / X[:, 5]
```

```python
        population_per_household = X[:, 4] / X[:, 5]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, 2] / X[:, 3]
            return np.c_[X, rooms_per_household, population_per_household,
bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

# Identify numerical and categorical columns
num_attribs = housing.drop("ocean_proximity", axis=1).columns  # All numeric columns
cat_attribs = ["ocean_proximity"]

# Build numerical pipeline: impute missing values, add new attributes, then scale
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

# Build the full pipeline combining numerical and categorical processing
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])

# Process the dataset using the pipeline
housing_prepared = full_pipeline.fit_transform(housing)
print("Shape of processed data:", housing_prepared.shape)
```

LABORATORY PROGRAM – 3

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

OBSERVATION BOOK

Date 17/03/25

LABORATORY – 03

Implementing Linear & Multiple Regression Algorithm

(Q1) ⇒ Find the Linear regression of the data of week and Product.

x_i (week)	y_i (sales in thousands)
1	2
2	4
3	5
4	9

:- $Y = \beta_0 + \beta_1 X$

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

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

$$X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}^{-1} = \begin{bmatrix} 0.75 & -0.25 \\ -0.25 & 0.1 \end{bmatrix}$$

$$X^T Y = \begin{bmatrix} 20 \\ 52 \end{bmatrix}$$

$$\beta = \begin{bmatrix} 0.75 & -0.25 \\ -0.25 & 0.1 \end{bmatrix} \begin{bmatrix} 20 \\ 52 \end{bmatrix}$$

$\beta_0 = -0.5$ $\quad$ $\beta = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$

$\beta_1 = 2.2$

$\Rightarrow Y = -0.5 + 2.2X$

(2)

$$X = \begin{bmatrix} 1 & 8 \\ 1 & 10 \\ 1 & 12 \end{bmatrix} \quad Y = \begin{bmatrix} 10 \\ 13 \\ 16 \end{bmatrix}$$

Predict the prize of 20 inch pizza using the given data.

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

$$(X^T)X^{-1} = \begin{bmatrix} 77/6 & -5/4 \\ -5/4 & 1/8 \end{bmatrix}$$

$$((X^T X)^{-1} X^T) Y = \begin{bmatrix} -2 \\ 1.5 \end{bmatrix} = \beta$$

Least error

$Y = \beta_0 + \beta_1 X$

$Y = (1.5)X + (-2)$

For $X = 20$, $\quad Y = (1.5) \times 20 + (-2)$

$Y = 28$

Questions:-

① Data Preprocessing steps performed :-

:- Missing values :- Filled with column mean
Used fillna() to replace with col mean.

One-hot Encoding :- Converted text to numbers

~~Scaling~~

② Relation between year and per capita income :-

- The plot showed strong linear rel" between year and per capita income.
- Hence positive correlation

③ Predicted salary : $137, 493.22

④ Encoded "State" using 'One-hot Encoding'

① In "salary.csv", filled the missing values in "Years Experience" column with ffill.

In "hiring.csv", filled "test_score" column missing value with the column mean.

In "hiring.csv" performed One-hot encoding for "experience" column.

CODE WITH OUTPUT

```
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, classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Load the iris dataset (make sure iris.csv is in the working directory)
iris = pd.read_csv("iris.csv")
# Assuming the last column is the target (species) and the rest are features.
X = iris.iloc[:, :-1]
y = iris.iloc[:, -1]

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

# Initialize and train the Decision Tree classifier
clf_iris = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_iris.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred_iris = clf_iris.predict(X_test)
accuracy_iris = accuracy_score(y_test, y_pred_iris)
conf_matrix_iris = confusion_matrix(y_test, y_pred_iris)

print("IRIS Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_iris)
print("Confusion Matrix:\n", conf_matrix_iris)
print("Classification Report:\n", classification_report(y_test, y_pred_iris))

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf_iris, filled=True, feature_names=X.columns, class_names=clf_iris.classes_)
plt.title("Decision Tree for IRIS Dataset")
plt.show()
```

```
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, classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Load the drug dataset (make sure drug.csv is in the working directory)
drug = pd.read_csv("drug.csv")

# Since the target column is 'Drug', drop it from the features
```

```python
X_drug = drug.drop('Drug', axis=1)
y_drug = drug['Drug']

# If there are categorical features, perform necessary encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# Encode features that are categorical
for col in X_drug.select_dtypes(include='object').columns:
    X_drug[col] = le.fit_transform(X_drug[col])
# Also encode the target variable if necessary
y_drug = le.fit_transform(y_drug)

# Split the data (80% training, 20% testing)
X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_drug, y_drug, test_size=0.2,
random_state=42)

# Initialize and train the Decision Tree classifier using entropy criterion
clf_drug = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_drug.fit(X_train_d, y_train_d)

# Make predictions and evaluate the model
y_pred_drug = clf_drug.predict(X_test_d)
accuracy_drug = accuracy_score(y_test_d, y_pred_drug)
conf_matrix_drug = confusion_matrix(y_test_d, y_pred_drug)

print("Drug Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_drug)
print("Confusion Matrix:\n", conf_matrix_drug)
print("Classification Report:\n", classification_report(y_test_d, y_pred_drug))

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf_drug, filled=True, feature_names=X_drug.columns,
          class_names=[str(cls) for cls in clf_drug.classes_])
plt.title("Decision Tree for Drug Dataset")
plt.show()
```

```
Drug Dataset Decision Tree Classifier
Accuracy: 1.0
Confusion Matrix:
 [[ 6  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  5  0  0]
 [ 0  0  0 11  0]
 [ 0  0  0  0 15]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00        15

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

LABORATORY PROGRAM – 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

OBSERVATION BOOK

Date __/__/__
Page ____

- Accuracy is 80% ⇒ accuracy of the model is good and acceptable as it correctly predicts employee retention is 8 out of 10 class.

Decision Tree :-

1. Choose the best splitting attribute (a2 / a3)

:- 1. Entropy calculation :-

4 - No's ; 1 - Yes

Entropy $= -\left(\frac{4}{5}\log_2\frac{4}{5} + \frac{1}{5}\log_2\frac{1}{5}\right)$

$= 0.722$

2. Information gain for "a2" :-

Hot (4 sam) : 3 No's , 1 Yes

Entropy = 0.811

Cool (1 sample) ⇒ Entropy = 0

Entropy (a2) $= \left(\frac{4}{5} \times 0.811\right) + \left(\frac{1}{5} \times 0\right) =$

0.649

}

Page ____

3. Information gain for "a3"

High → 4 No's ∴ Entropy = 0

Normal → 1 Yes

∴ Entropy (Normal) = 0

Entropy (a3) $= \left(\frac{4}{5} \times 0 + \frac{1}{5} \times 0\right) = 0$

Information gain $= 0.722 - 0$

$= 0.722$

∴ I. gain (a3) = 0.722
I. gain (a2) = 0.649

∴ Best splitting attribute is "a3".

⇒ For "iris.csv" dataset, accuracy is 81%. Confusion matrix identifies ~~the~~ 81/100 times properly - i.e, true positives & 19/100 false positives & false negatives.

⇒ For "Petrol_consumption.csv" dataset, accuracy is 87%.

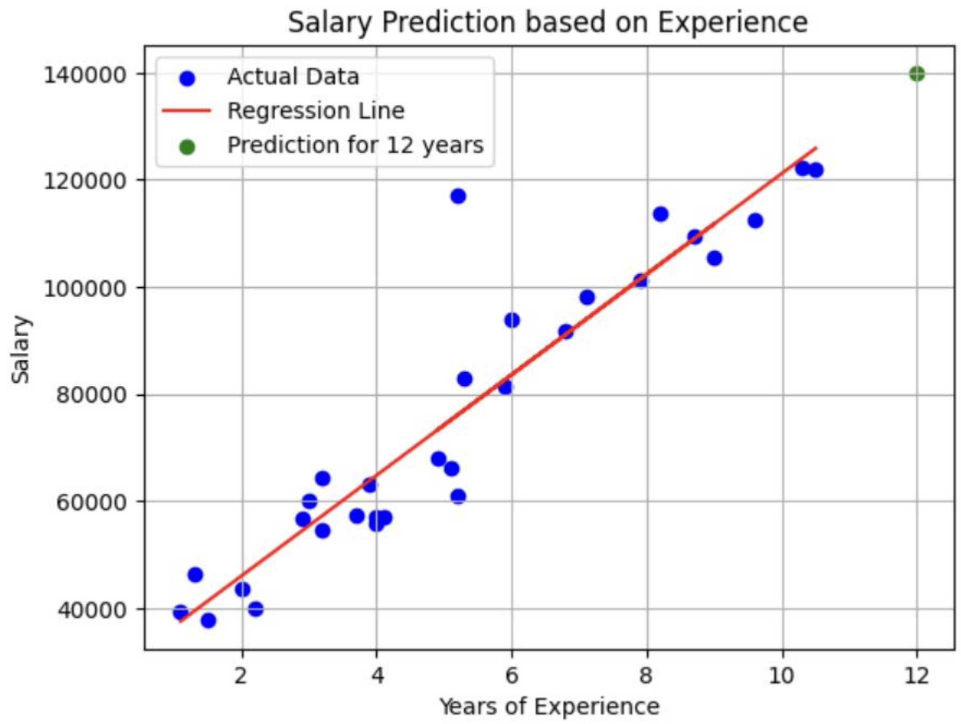
7/4/25

CODE WITH OUTPUT

```
import pandas as pd
from sklearn.linear_model import LinearRegression
# Load the data
income_data = pd.read_csv("canada_per_capita_income.csv")
# Assumed data columns: 'Year' and 'PerCapitaIncome'
print("Canada Income Data Head:")
print(income_data.head())
# Prepare feature and target
X_income = income_data[["year"]]     # Predictor variable: Year
y_income = income_data["per capita income (US$)"]
# Build and train the linear regression model
model_income = LinearRegression()
model_income.fit(X_income, y_income)

# Predict per capita income for the year 2020
predicted_income = model_income.predict([[2020]])

print("\nPredicted per capita income for Canada in 2020:", predicted_income[0])

# Plot the data points and the regression line
plt.scatter(X_income, y_income, color='blue', label='Actual Data')
plt.plot(X_income, model_income.predict(X_income), color='red', label='Regression Line')

# Plot the prediction for 2020
plt.scatter(2020, predicted_income[0], color='green', label='Prediction for 2020')

# Customize the plot
plt.xlabel('Year')
plt.ylabel('Per Capita Income (US$)')
plt.title('Canada Per Capita Income Prediction')
plt.legend()
plt.grid(True)

# Display the plot
plt.show()



import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression

# Load the salary data
salary_data = pd.read_csv("salary.csv")
print(income_data.head())

# Prepare feature and target
X_salary = salary_data[["YearsExperience"]]  # Predictor variable: Years of Experience
y_salary = salary_data["Salary"]
```

```
# Build and train the linear regression model
model_salary = LinearRegression()
model_salary.fit(X_salary, y_salary)

import matplotlib.pyplot as plt
# Plot the data points and the regression line
plt.scatter(X_salary, y_salary, color='blue', label='Actual Data')
plt.plot(X_salary, model_salary.predict(X_salary), color='red', label='Regression Line')

# Plot the prediction for 12 years of experience
plt.scatter(12, predicted_salary[0], color='green', label='Prediction for 12 years')

# Customize the plot
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Salary Prediction based on Experience')
plt.legend()
plt.grid(True)

# Display the plot
plt.show()
```

```
Predicted salary for an employee with 12 years of experience: 139980.88923969213
```




```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

# Read the CSV file (ensure the file is uploaded in your Colab environment)
```

```python
df = pd.read_csv("hiring.csv")

# Rename columns for convenience
df.columns = ['experience', 'test_score', 'interview_score', 'salary']

print("Original Data:")
print(df)
# Function to convert experience values to numeric
def convert_experience(x):
    try:
        return float(x)
    except:
        x_lower = str(x).strip().lower()
        return num_map.get(x_lower, np.nan)

# Convert the 'experience' column using the mapping
df['experience'] = df['experience'].apply(convert_experience)

# Convert 'test_score', 'interview_score', and 'salary' to numeric (coerce errors to NaN)
df['test_score'] = pd.to_numeric(df['test_score'], errors='coerce')
df['interview_score'] = pd.to_numeric(df['interview_score'], errors='coerce')
df['salary'] = pd.to_numeric(df['salary'], errors='coerce')

print("\nData After Conversion:")
print(df)

# Fill missing values in numeric columns using the column mean
df['experience'].fillna(df['experience'].mean(), inplace=True)
df['test_score'].fillna(df['test_score'].mean(), inplace=True)
df['interview_score'].fillna(df['interview_score'].mean(), inplace=True)

print("\nData After Filling Missing Values:")
print(df)

# Prepare the feature matrix X and target vector y
X = df[['experience', 'test_score', 'interview_score']]
y = df['salary']

# Build and train the Multiple Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Predict salaries for the given candidate profiles
# Candidate 1: 2 years of experience, 9 test score, 6 interview score
candidate1 = np.array([[2, 9, 6]])
predicted_salary1 = model.predict(candidate1)

# Candidate 2: 12 years of experience, 10 test score, 10 interview score
candidate2 = np.array([[12, 10, 10]])
predicted_salary2 = model.predict(candidate2)
```

```python
print("\nPredicted Salary for Candidate (2 yrs, 9 test, 6 interview): $",
round(predicted_salary1[0], 2))
print("Predicted Salary for Candidate (12 yrs, 10 test, 10 interview): $",
round(predicted_salary2[0], 2))
import matplotlib.pyplot as plt

# Create the plot
plt.figure(figsize=(10, 6))  # Adjust figure size for better visualization
plt.scatter(df['experience'], y, color='blue', label='Actual Salary') #Plot actual salary against
years of experience

# Plot the regression line (this is an approximation since it's a multi-variable regression)
# You can visualize a single feature against the predicted salary
plt.plot(df['experience'], model.predict(X), color='red', label='Regression Line')

# Highlight predictions
plt.scatter(candidate1[0, 0], predicted_salary1, color='green', label='Candidate 1 Prediction')
plt.scatter(candidate2[0, 0], predicted_salary2, color='purple', label='Candidate 2 Prediction')

# Add labels and title
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Salary Prediction based on Experience, Test Score, Interview Score")

# Add a legend
plt.legend()
plt.grid(True)
plt.show()
```

LABORATORY PROGRAM – 5

Build Logistic Regression Model for a given dataset

OBSERVATION BOOK

⇒ Logistic Regression

• Question 1 :–

$m = 0.8 \qquad c = -5$

(a) $\text{Sigmoid}(z) = \dfrac{1}{1 + e^{-(mx+c)}}$

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

(b) Given : $x = 7$

$$\therefore z = \frac{1}{1 + e^{-0.6}} = 0.6456$$

(c) Keeping threshold as 0.5, and $z = 0.6456$, the predicted class is "PASS".

(Q) Which variables have a direct and clear impact on employee retention?

:– • Satisfaction level has a strong negative correlation with employee retention.

• Promotion in last 5 years affects the stability of the job as they are likely to leave.

Date __/__/__
Page ____

- Accuracy is 80% ⇒ accuracy of the model is good and acceptable as it correctly predicts employee retention is 8 out of 10 class.

Decision Tree :-

1. Choose the best splitting attribute (a2 / a3).

∴ 1. Entropy calculation:-

4 - No's ; 1 - Yes

$$\text{Entropy} = -\left(\frac{4}{5}\log_2\frac{4}{5} + \frac{1}{5}\log_2\frac{1}{5}\right)$$

$$= 0.722$$

2. Information gain for "a2" :-

Hot (4 rows) : 3 No's , 1 Yes

Entropy = 0.811

Cool (1 sample) ⇒ Entropy = 0

$$\text{Entropy (a2)} = \left(\frac{4}{5} \times 0.811\right) + \left(\frac{1}{5} \times 0\right) =$$

0.649

3

Page ____

3. Information gain for "a3"

High → 4 No's ∴ Entropy = 0

Normal → 1 Yes

∴ Entropy (Normal) = 0

$$\text{Entropy (a3)} = \left(\frac{4}{5} \times 0 + \frac{1}{5} \times 0\right) = 0$$

Information gain = 0.722 − 0

= 0.722

∴ I.gain (a3) = 0.722

I.gain (a2) = 0.649

∴ Best splitting attribute is "a3".

⇒ For "iris.csv" dataset, accuracy is 81%. Confusion matrix identifies ~~the~~ 81/100 times properly – i.e, true positives & 19/100 false positives & false negatives.

⇒ For "Petrol_consumption.csv" dataset, accuracy is 87%.

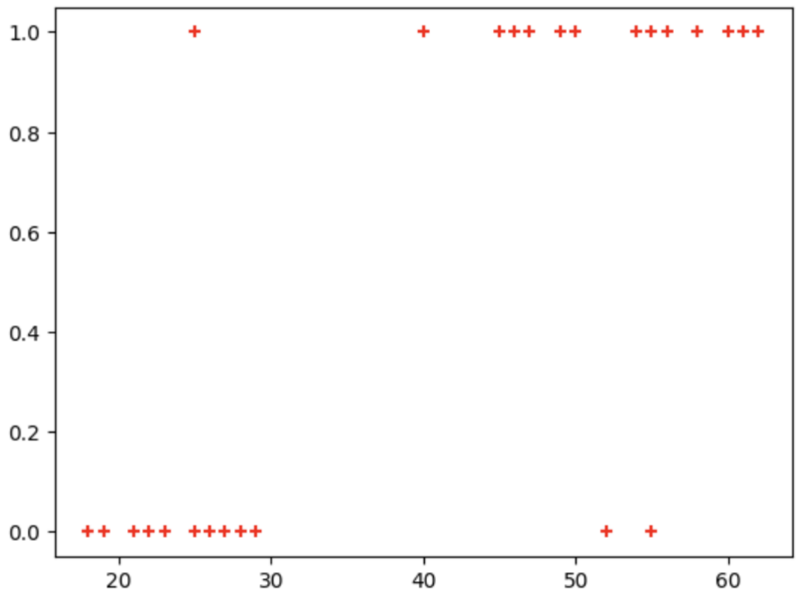
7/4/25

CODE WITH OUTPUT

```
import pandas as pd
from matplotlib import pyplot as plt
# %matplotlib inline
#"%matplotlib inline" will make your plot outputs appear and be stored within the notebook.

df = pd.read_csv("insurance_data.csv")
df.head()

plt.scatter(df.age,df.bought_insurance,marker='+',color='red')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df[['age']],df.bought_insurance,train_size=0.9,random_state=10)
X_train.shape

X_test

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

model.fit(X_train, y_train)

X_test

y_test

y_predicted = model.predict(X_test)
y_predicted

model.score(X_test,y_test)

model.predict_proba(X_test)

y_predicted = model.predict([[60]])
y_predicted

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

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

#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

def prediction_function(age):
   z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
```

```
    y = sigmoid(z)
    return y

age = 35
prediction_function(age)

"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""
```

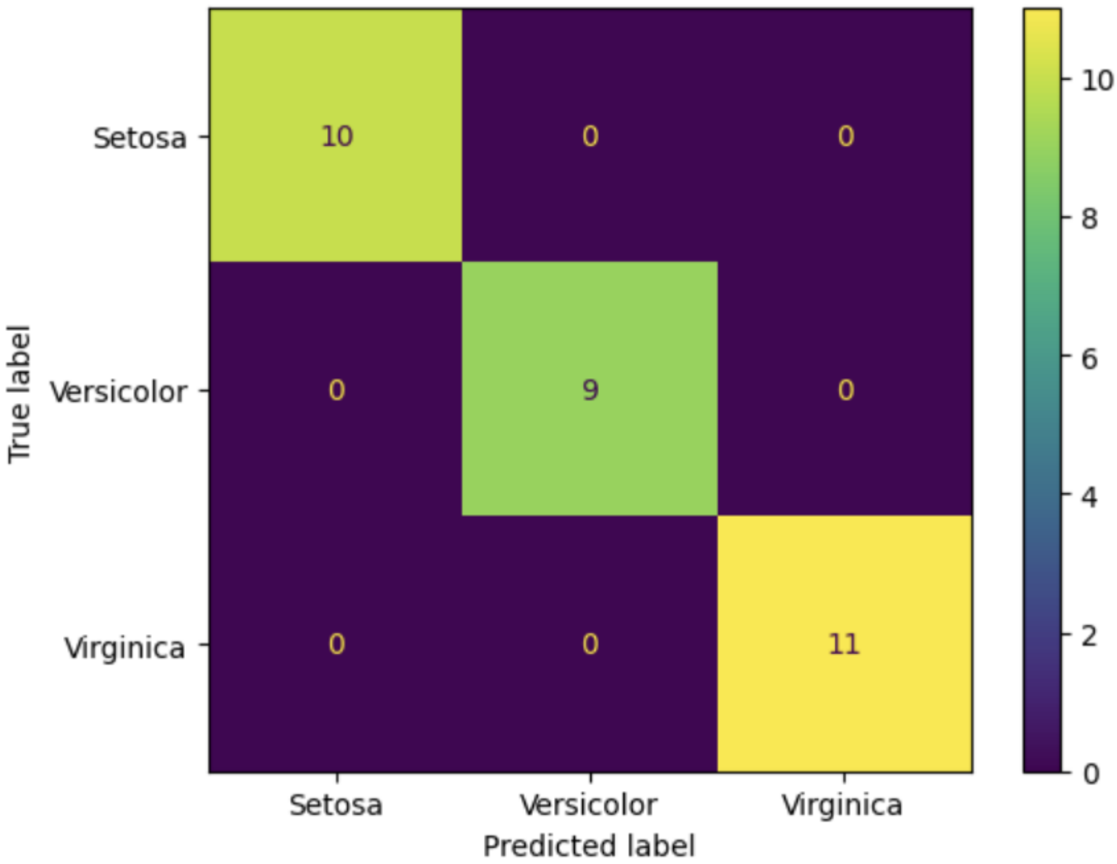
```
'0.37 is less than 0.5 which means person with 35 will not buy the insurance'
```




```
# Import necessary libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt

# Load the Iris dataset
iris =  pd.read_csv("iris.csv")
iris.head()

X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, petal width)
y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

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

# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver
model = LogisticRegression(multi_class='multinomial')
```

```
# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)

# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set:
{accuracy:.2f}")

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = ["Setosa", "Versicolor", "Virginica"])

cm_display.plot()
plt.show()
```

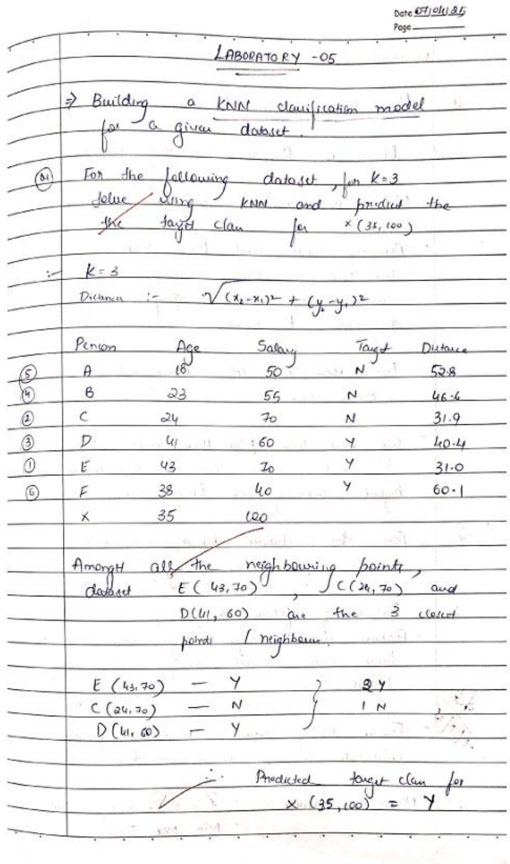
```
Accuracy of the Multinomial Logistic Regression model on the test set: 1.00
```



LABORATORY PROGRAM – 6

Build KNN Classification model for a given dataset.

OBSERVATION BOOK



Date 07/04/25
Page

LABORATORY -05

⇒ Building a KNN classification model for a given dataset.

Q1 For the following dataset, for K=3 solve using KNN and predict the target class for x (35, 100)

:- K = 3

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

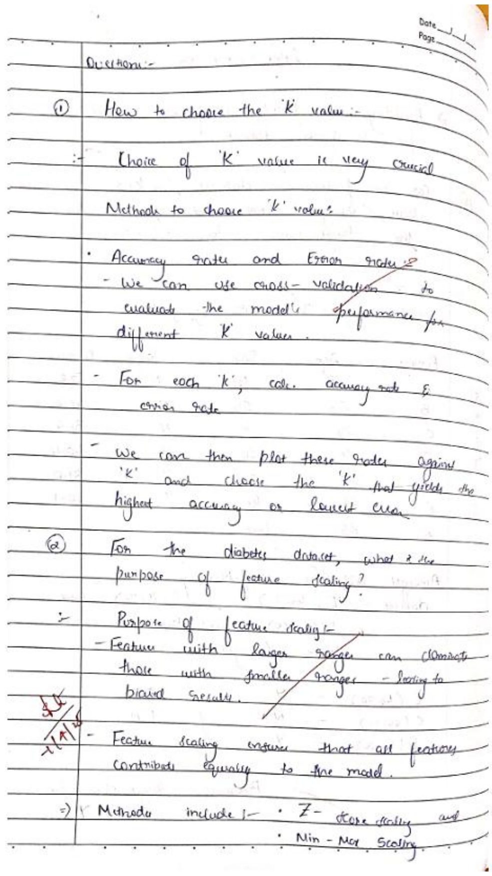
	Person	Age	Salary	Target	Distance
5	A	18	50	N	52.8
4	B	23	55	N	46.6
2	C	24	70	N	31.9
3	D	41	60	Y	40.4
1	E	43	70	Y	31.0
6	F	38	40	Y	60.1
	X	35	100		

Amongst all the neighbouring points, dataset E (43, 70), C (24, 70) and D(41, 60) are the 3 closest points / neighbours.

E (43, 70) — Y
C (24, 70) — N
D (41, 60) — Y

2 Y
1 N

∴ Predicted target class for x (35, 100) = Y



Date
Page

Questions:-

1 How to choose the 'K' value:-

:- Choice of 'K' value is very crucial

Methods to choose 'k' value:

- Accuracy rates and Error rates:
  - We can use cross-validation to evaluate the model's performance for different 'K' values.
  - For each 'K', calc. accuracy rate & error rate
  - We can then plot these rates against 'K' and choose the 'K' that yields the highest accuracy or lowest error

2 For the diabetes dataset, what is the purpose of feature scaling?

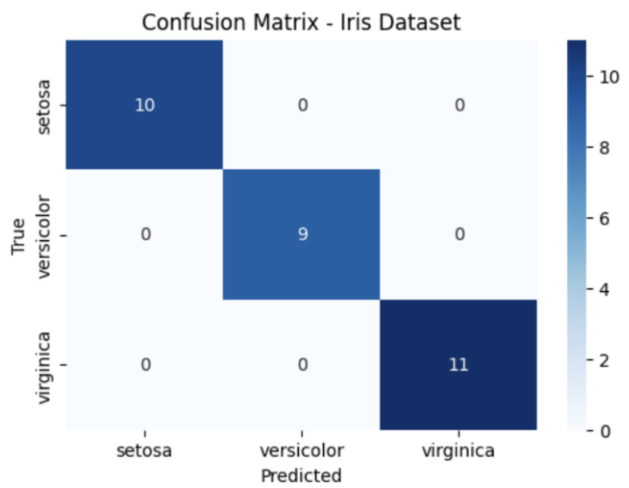
:- Purpose of feature scaling:-
- Features with larger ranges can dominate those with smaller ranges - leading to biased results.
- Feature scaling ensures that all features contribute equally to the model.

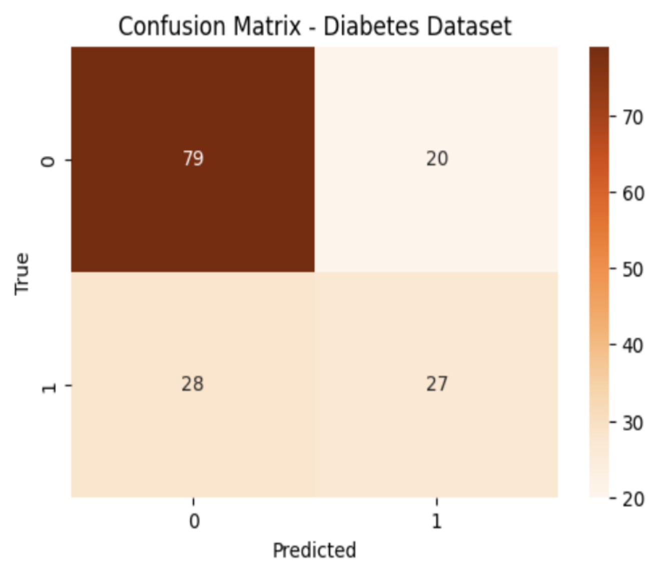
⇒ Methods include :- • Z- score scaling and
• Min - Max Scaling.

CODE WITH OUTPUT

```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# For model building and evaluation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# -------------------------- Part 1: IRIS Dataset -------------------------- #
# Load the iris dataset (ensure iris.csv is in the same directory or provide correct path)
iris_df = pd.read_csv("iris.csv")

# Separate features and target
X_iris = iris_df.drop("species", axis=1)
```

```
y_iris = iris_df["species"]

# Split the data (80% training, 20% testing)
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
)

# Choose a value for k; here K=3 is used as an example.
knn_iris = KNeighborsClassifier(n_neighbors=3)

# Train the model on training data
knn_iris.fit(X_train_iris, y_train_iris)

# Predict on test data
y_pred_iris = knn_iris.predict(X_test_iris)

# Calculate accuracy score
acc_iris = accuracy_score(y_test_iris, y_pred_iris)
print("IRIS Dataset Accuracy Score:", acc_iris)
# Compute confusion matrix and classification report
cm_iris = confusion_matrix(y_test_iris, y_pred_iris)
print("\nIRIS Dataset Confusion Matrix:\n", cm_iris)
```





LABORATORY PROGRAM – 7

Build Support vector machine model for a given dataset

OBSERVATION BOOK

Date 21/04/25

Page

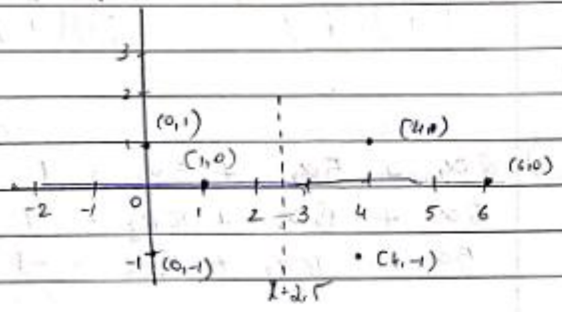
LABORATORY — 06

⇒ Support Vector Machines:-

Draw an optimal hyperplane for the points (4,1), (4,-1), and (6,0) belong to positive class and points (1,0), (0,1) and (0,-1) belong to negative class.

$$\left.\begin{matrix}(4,1)\\(4,-1)\\(6,0)\end{matrix}\right\}\ \text{Positive class}$$

$$\left.\begin{matrix}(1,0)\\(0,1)\\(0,-1)\end{matrix}\right\}\ \text{Negative class}$$



⇒ Support vectors are

(1,0) — Positive class

(4,1) — Neg class

(4,-1) — Neg class

$$\left\{ S_1 = \begin{pmatrix}1\\0\end{pmatrix} \quad S_2 = \begin{pmatrix}4\\1\end{pmatrix} \quad S_3 = \begin{pmatrix}4\\-1\end{pmatrix} \right\}$$

Augmented support vectors:-

$\Rightarrow \left\{ s_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad s_2 = \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix} \quad s_3 = \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix} \right\}$

$\Rightarrow \alpha_1 s_1 s_1 + \alpha_2 s_2 s_1 + \alpha_3 s_3 s_1 = +1$

$\alpha_1 s_1 s_2 + \alpha_2 s_2 s_2 + \alpha_3 s_3 s_2 = -1$

$\alpha_1 s_1 s_3 + \alpha_2 s_2 s_3 + \alpha_3 s_3 s_3 = -1$

$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = +1$

$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix} = -1$

$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}\begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix} = -1$

$\Rightarrow 2\alpha_1 + 5\alpha_2 + 5\alpha_3 = 1$

$5\alpha_1 + 18\alpha_2 + 16\alpha_3 = -1$

$5\alpha_1 + 16\alpha_2 + 18\alpha_3 = -1$

$\Rightarrow x = 3.5$

① For iris.csv

Linear Kernel accuracy = 1.0

RBF Kernel accuracy = 1.0

Performance of both is same.
Similar performance is likely because the dataset is linearly separable for most parts.

② Confusion Matrix shows strong diagonal values, but some cells b/w similar looking letters is visible

Ex:- O and D   M and N
I and L   V and Y

Accuracy score - 0.995

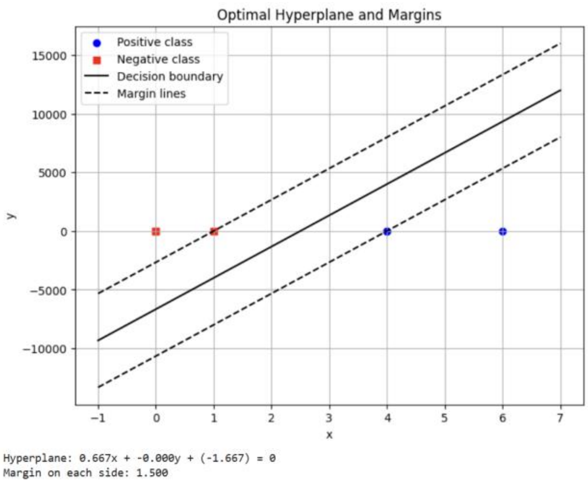
→ Model has good performance across all classes.

SVM Model perf. on iris dataset = 1.0
Letter dataset ~ 0.945

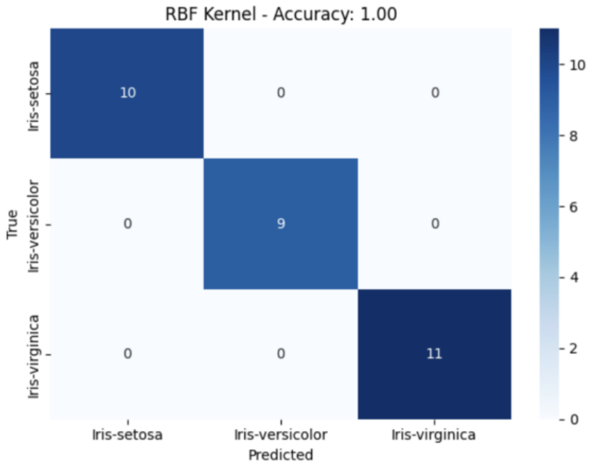
4.5
21/4/25

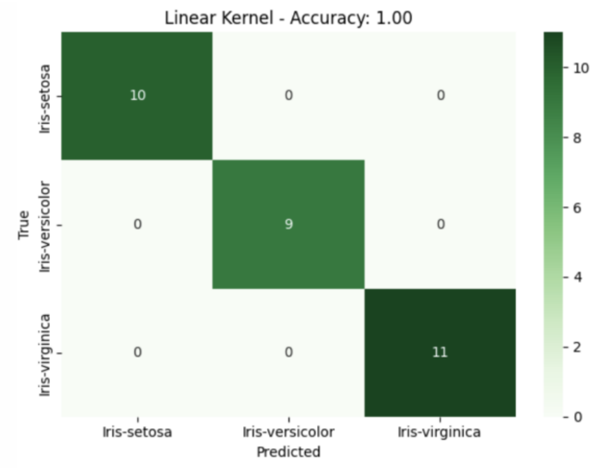
CODE WITH OUTPUT

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

# Data points
X = np.array([[4, 1], [4, -1], [6, 0], [1, 0], [0, 1], [0, -1]])
y = np.array([1, 1, 1, -1, -1, -1])

# Fit linear SVM with a very large C to approximate hard-margin
clf = SVC(kernel='linear', C=1e6)
clf.fit(X, y)

# Extract model parameters
w = clf.coef_[0]
b = clf.intercept_[0]

# Compute decision boundary and margins
xx = np.linspace(-1, 7, 500)
yy = -(w[0] * xx + b) / w[1]

# Margin offset: distance = 1/||w||
margin = 1 / np.linalg.norm(w)
yy_down = yy - np.sqrt(1 + (w[0] / w[1])**2) * margin
yy_up   = yy + np.sqrt(1 + (w[0] / w[1])**2) * margin

# Plotting
plt.figure(figsize=(8, 6))
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='blue', marker='o', label='Positive class')
plt.scatter(X[y == -1, 0], X[y == -1, 1], c='red', marker='s', label='Negative class')
plt.plot(xx, yy, 'k-', label='Decision boundary')
plt.plot(xx, yy_down, 'k--', label='Margin lines')
plt.plot(xx, yy_up, 'k--')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Optimal Hyperplane and Margins')
plt.grid(True)
plt.show()

# Print hyperplane equation
print(f"Hyperplane: {w[0]:.3f}x + {w[1]:.3f}y + ({b:.3f}) = 0")
print(f"Margin on each side: {margin:.3f}")
```



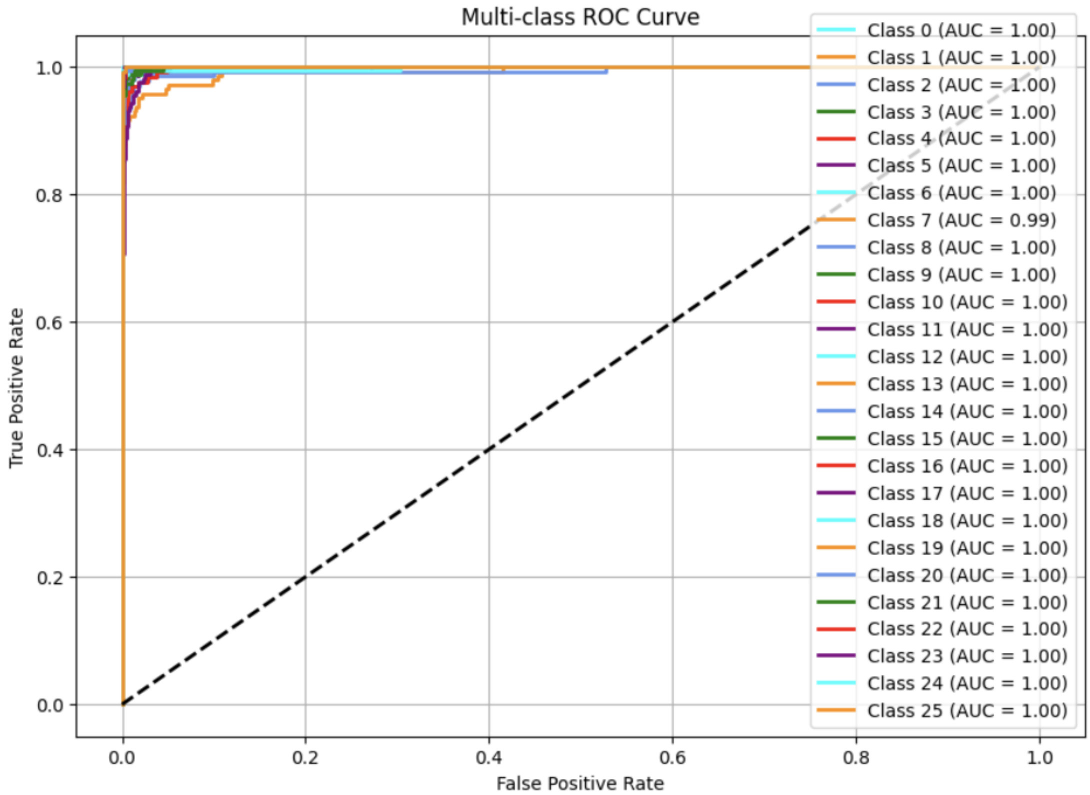

```
import pandas as pd

# Load both datasets
iris_df = pd.read_csv("/content/iris.csv")
# 1. IRIS DATASET - SVM with RBF and Linear Kernels
X_iris = iris_df.drop("species", axis=1)
y_iris = iris_df["species"]

# Encode labels
le_iris = LabelEncoder()
y_iris_encoded = le_iris.fit_transform(y_iris)

# Split dataset
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris_encoded,
test_size=0.2, random_state=42)

# Train models
svm_rbf = SVC(kernel='rbf')
svm_linear = SVC(kernel='linear')

svm_rbf.fit(X_train_iris, y_train_iris)
svm_linear.fit(X_train_iris, y_train_iris)

# Predictions
y_pred_rbf = svm_rbf.predict(X_test_iris)
y_pred_linear = svm_linear.predict(X_test_iris)

# Accuracy and Confusion Matrix
acc_rbf = accuracy_score(y_test_iris, y_pred_rbf)
acc_linear = accuracy_score(y_test_iris, y_pred_linear)
cm_rbf = confusion_matrix(y_test_iris, y_pred_rbf)
cm_linear = confusion_matrix(y_test_iris, y_pred_linear)
```






```
# Load dataset
letter_df = pd.read_csv("/content/letter-recognition.csv")  # Update path if needed
letter_df['letter'] = LabelEncoder().fit_transform(letter_df['letter'])

# Split features and labels
X = letter_df.drop('letter', axis=1)
y = letter_df['letter']

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

# Standardize
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train SVM
svm = SVC(kernel='rbf', probability=True)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
y_prob = svm.predict_proba(X_test)

# Accuracy and Confusion Matrix
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ROC and AUC (one-vs-rest)
y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```python
# Plot ROC Curve
plt.figure(figsize=(10, 7))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'red', 'purple'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f'Class {i} (AUC = {roc_auc[i]:0.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multi-class ROC Curve")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



LABORATORY PROGRAM – 8

Implement Random forest ensemble method on a given dataset.

OBSERVATION BOOK

Date 05/05/25
Page

LABORATORY - 07

Random Forest

→ Decision Tree :-

Sl No.	CGPA	Interactiveness	Com. Skills	Practical Knowledge	[illegible]
01	≥9	Yes	Good	Good	Yes
02	<9	No	Moderate	Good	Yes
03	≥9	No	Moderate	Avg	No
04	≥9	No	Moderate	Avg	No
05	≥9	Yes	Moderate	Good	Yes

:- Root Node - CGPA

CGPA

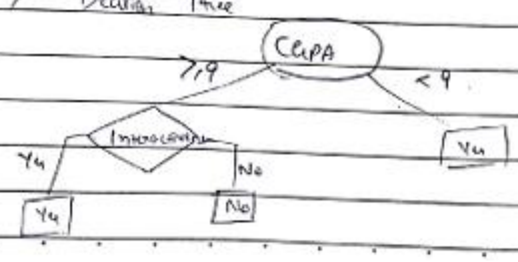
≥9 / \ <9

Splitting the data based on CGPA :-

(i) CGPA ≥ 9 :
Records - 1, 2, 5

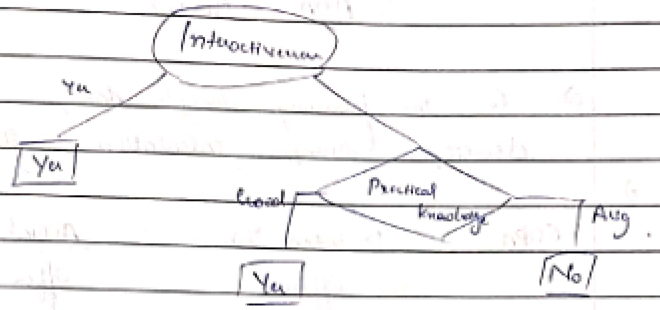
(ii) CGPA < 9 :
Records : 3, 4

:) Decision Tree



Page

Decision Tree for Sample S2 :-



=> The best accuracy score is 1.0 & confusion matrix is

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$$

=> For this dataset, number of trees is used to is

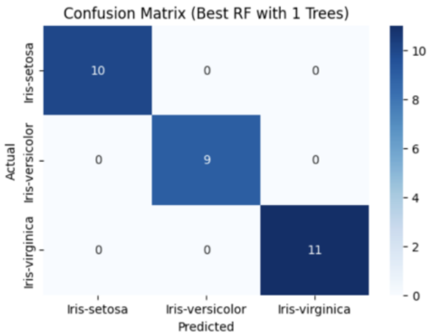
n = 34

best accuracy is 1.0.

CODE WITH OUTPUT

```
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
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("iris.csv")  # Adjust filename if needed

# Prepare data
X = df.drop(columns=["species"])  # Assuming 'species' is the target column
y = df["species"]

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

# Default Random Forest with 10 trees
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
acc_default = accuracy_score(y_test, y_pred_default)
conf_matrix_default = confusion_matrix(y_test, y_pred_default)

print(f"Default RF (10 trees) Accuracy: {acc_default}")
print("Confusion Matrix:\n", conf_matrix_default)

# Try different numbers of trees to find the best
best_acc = 0
best_n = 10
acc_list = []

for n in range(1, 101):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    acc_list.append((n, acc))
    if acc > best_acc:
        best_acc = acc
        best_n = n
        best_conf_matrix = confusion_matrix(y_test, y_pred)

print(f"\nBest Accuracy: {best_acc} using {best_n} trees")
print("Best Confusion Matrix:\n", best_conf_matrix)
# Plot accuracy vs number of trees
x_vals, y_vals = zip(*acc_list)
plt.plot(x_vals, y_vals, marker='o')
plt.title("Accuracy vs Number of Trees")
plt.xlabel("Number of Trees")
```

```
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```



LABORATORY PROGRAM – 9

Implement Boosting ensemble method on a given dataset.

OBSERVATION BOOK

Date 05/05/25

Page

LABORATORY – 08

ADA BOOST

=> For the following sample data, show the decision stump calculation steps for CGPA

CGPA	Predicted Job Offer	Actual Job Offer	Weight
>=9	Yes	Yes	1/6
<9	No	Yes	1/6
>=9	Yes	No	1/6
<9	No	No	1/6
>=9	Yes	Yes	1/6
>=9	Yes	Yes	1/6

$$\mathcal{E}_i = \sum_{i=1}^{N} I_i(d_i)\, w_t(d_i)$$

$$\mathcal{E}_{CGPA} = 2 \times \frac{1}{6} = 0.3333$$

$$\alpha_{CGPA} = \frac{1}{2} \ln \frac{(1 - \mathcal{E}_{CGPA})}{\mathcal{E}_{CGPA}}$$

$$= \frac{1}{2} \ln \frac{(1 - 0.333)}{0.333}$$

$$\alpha_{CGPA} = 0.347$$

$$Z_{CGPA} = \frac{1}{6} \times 4 \times e^{-0.347} + \frac{1}{6} \times 2 \times e^{0.347}$$

$$= 0.9428$$

Date __/__/__
Page ______

$$wt(d_j)_{i+1} = \frac{wt(d_j)_{CGPA} \text{ of correct instance} \times e^{-\alpha_{CGPA}}}{Z_{CGPA}}$$

$$wt(d_j)_{i+1} = \frac{\frac{1}{6} \times e^{-0.[illegible]}}{0.9428} = 0.1249$$

$$wt(d_j)_{i+1} = \frac{\frac{1}{6} \times e^{0.[illegible]}}{0.9428} = 0.2501$$

CGPA	Predicted Job Offer	Actual Job Offer	Weight
>=9	Yes	Yes	0.1249
<9	No	Yes	0.2501
>=9	Yes	No	0.2501
<9	No	No	0.1249
>=9	Yes	Yes	0.1249
>=9	Yes	Yes	0.1249

Best accuracy score achieved is 83.40 % with n = 500.

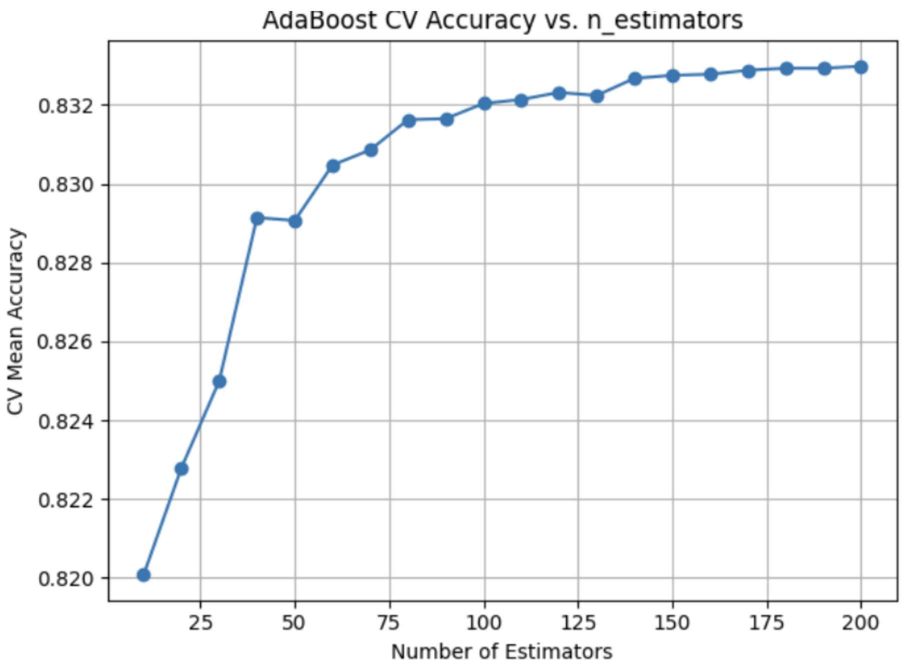
5/5/23

CODE WITH OUTPUT

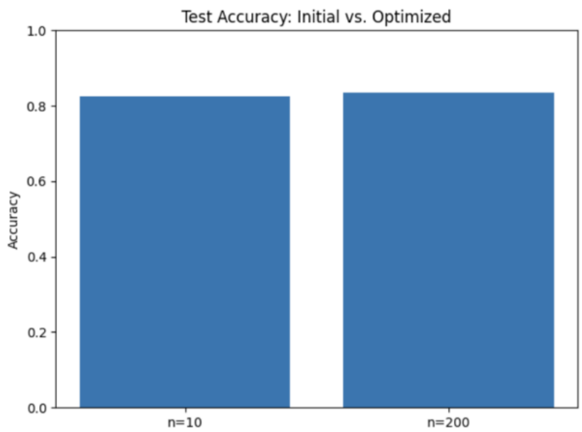
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay

# Load dataset
data = pd.read_csv('income.csv')

# Display basic info
print("First five rows:")
print(data.head())
print(f"\nDataset shape: {data.shape}")

# Define features and target
target_column = 'income_level'
y = data[target_column]
X = data.drop(columns=[target_column])

# Identify categorical vs numerical columns
categorical_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
print(f"\nNumerical columns: {numerical_cols}")
print(f"Categorical columns: {categorical_cols}")

# Preprocessor: scale numericals, one-hot encode categoricals
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ]
)

# Initial AdaBoost model with 10 estimators
pipeline = Pipeline([
    ('preprocess', preprocessor),
    ('clf', AdaBoostClassifier(n_estimators=10, random_state=42))
])

# Split into train/test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Train and evaluate initial model
pipeline.fit(X_train, y_train)
```

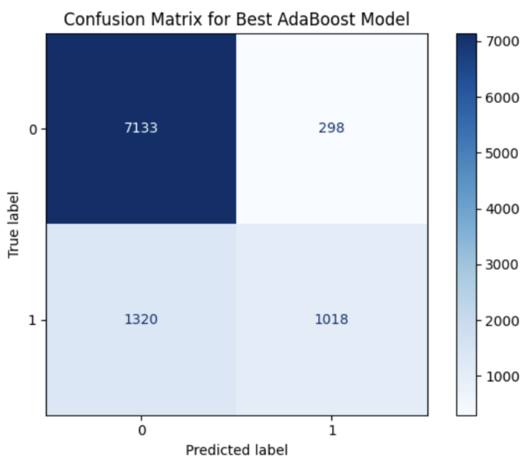
```python
y_pred = pipeline.predict(X_test)
initial_acc = accuracy_score(y_test, y_pred)
print(f"Initial test accuracy (n_estimators=10): {initial_acc:.4f}")

# Hyperparameter tuning: find best n_estimators
tree_counts = list(range(10, 201, 10))  # 10,20,...,200
cv_scores = []
for n in tree_counts:
    model = Pipeline([
        ('preprocess', preprocessor),
        ('clf', AdaBoostClassifier(n_estimators=n, random_state=42))
    ])
    scores = cross_val_score(
        model, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1
    )
    mean_score = scores.mean()
    cv_scores.append(mean_score)
    print(f"n_estimators={n}: CV mean accuracy={mean_score:.4f}")

# Plot CV accuracy vs. number of estimators
plt.figure()
plt.plot(tree_counts, cv_scores, marker='o')
plt.title('AdaBoost CV Accuracy vs. n_estimators')
plt.xlabel('Number of Estimators')
plt.ylabel('CV Mean Accuracy')
plt.grid(True)
plt.tight_layout()
plt.show()

# Determine optimal number of trees
best_score = max(cv_scores)
best_n = tree_counts[cv_scores.index(best_score)]
print(f"\nBest CV accuracy={best_score:.4f} with n_estimators={best_n}")

# Retrain and evaluate best model
best_model = Pipeline([
    ('preprocess', preprocessor),
    ('clf', AdaBoostClassifier(n_estimators=best_n, random_state=42))
])
best_model.fit(X_train, y_train)
y_best = best_model.predict(X_test)
best_test_acc = accuracy_score(y_test, y_best)
print(f"Test accuracy with best n_estimators ({best_n}): {best_test_acc:.4f}")

# Plot comparison of initial vs. best test accuracy
plt.figure()
plt.bar(['n=10', f'n={best_n}'], [initial_acc, best_test_acc])
plt.title('Test Accuracy: Initial vs. Optimized')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.tight_layout()
```

```
plt.show()

# Plot confusion matrix for best model
cm = confusion_matrix(y_test, y_best)
labels = best_model.named_steps['clf'].classes_
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
plt.figure()
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Best AdaBoost Model')
plt.tight_layout()
plt.show()
```




```
Best CV accuracy=0.8330 with n_estimators=200
Test accuracy with best n_estimators (200): 0.8344
```

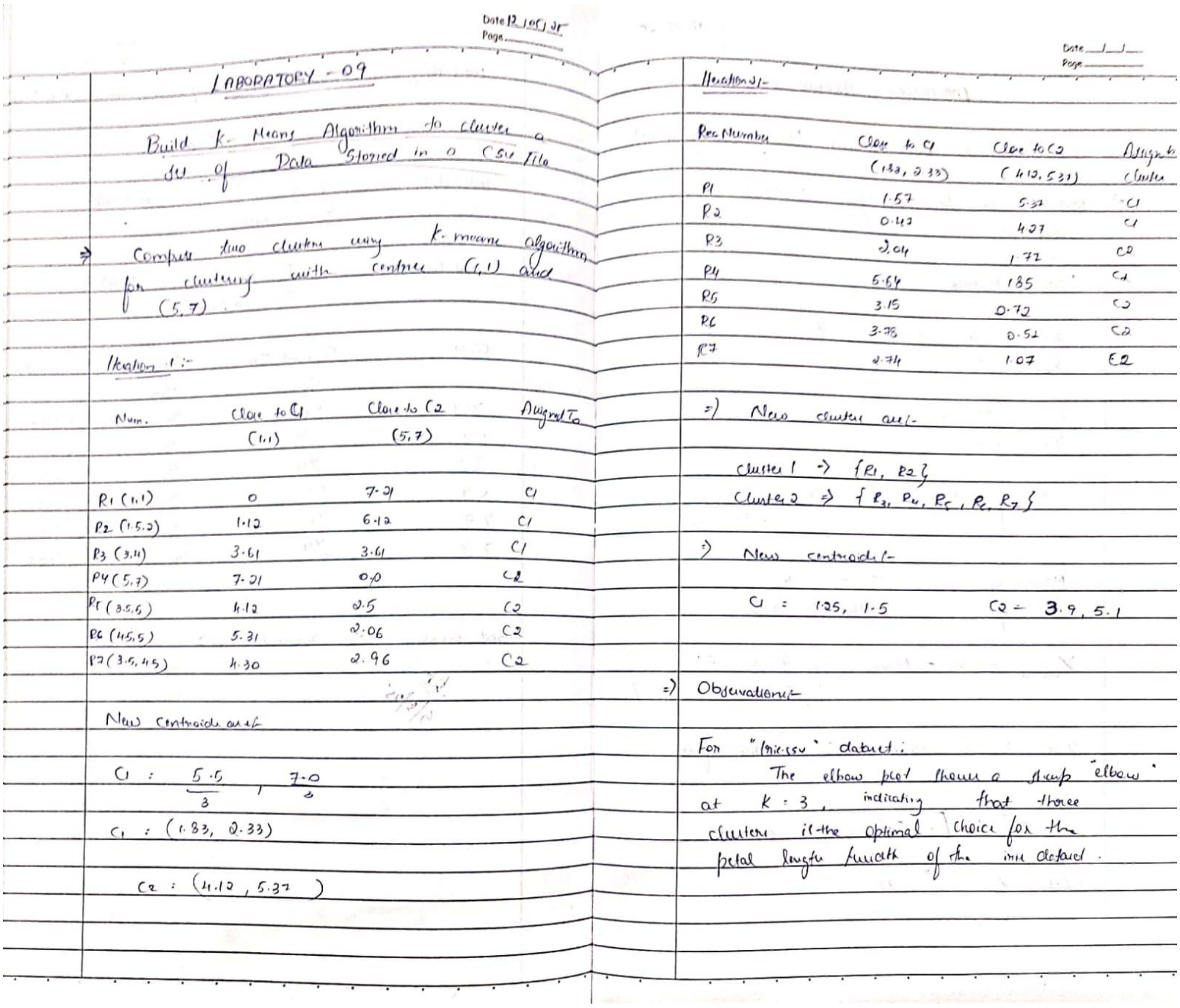




LABORATORY PROGRAM – 10

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

OBSERVATION BOOK


Date 12/05/25

LABORATORY - 09

Build K-Means Algorithm to cluster a set of Data Stored in a CSV File

⇒ Compute two clusters using k-means algorithm for clustering with centres (1,1) and (5,7)

Iteration 1 :-

Num.	Close to C1 (1,1)	Close to C2 (5,7)	Assigned To
P1 (1,1)	0	7.21	C1
P2 (1.5,2)	1.12	6.12	C1
P3 (3,4)	3.61	3.61	C1
P4 (5,7)	7.21	0	C2
P5 (3.5,5)	4.12	2.5	C2
P6 (4.5,5)	5.31	2.06	C2
P7 (3.5,4.5)	4.30	2.96	C2

New centroids are:

C1 : $\frac{5.5}{3}$, $\frac{7.0}{3}$

C1 : (1.83, 2.33)

C2 : (4.12, 5.37)

Iteration 2 :-

Rec Number	Close to C1 (1.83, 2.33)	Close to C2 (4.12, 5.37)	Assigned to cluster
P1	1.57	5.37	C1
P2	0.47	4.27	C1
P3	2.04	1.72	C2
P4	5.64	1.85	C2
P5	3.15	0.72	C2
P6	3.78	0.52	C2
P7	2.74	1.07	C2

⇒ New clusters are:

Cluster 1 → {P1, P2}

Cluster 2 → {P3, P4, P5, P6, P7}

⇒ New centroids :-

C1 : 1.25, 1.5   C2 = 3.9, 5.1

⇒ Observations :-

For "iris.csv" dataset :

The elbow plot shows a sharp "elbow" at k = 3, indicating that three clusters is the optimal choice for the petal length/width of the iris dataset.

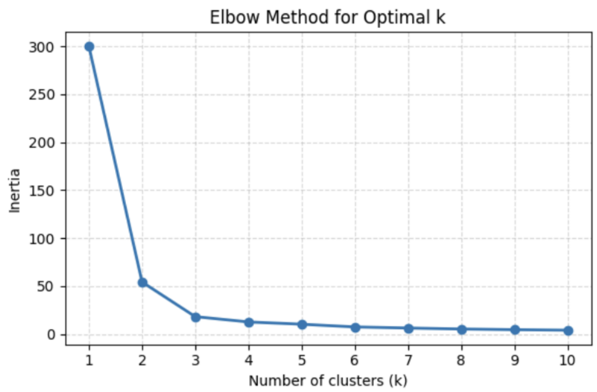
CODE WITH OUTPUT

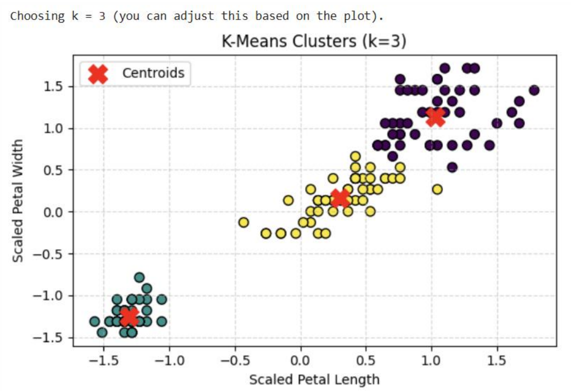
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def load_data(csv_path='iris.csv'):
    """
    Try loading from csv_path; if not found, load via sklearn.
    Expects columns: sepal_length, sepal_width, petal_length, petal_width, species.
    Returns DataFrame with a 'species' column.
    """
    try:
```

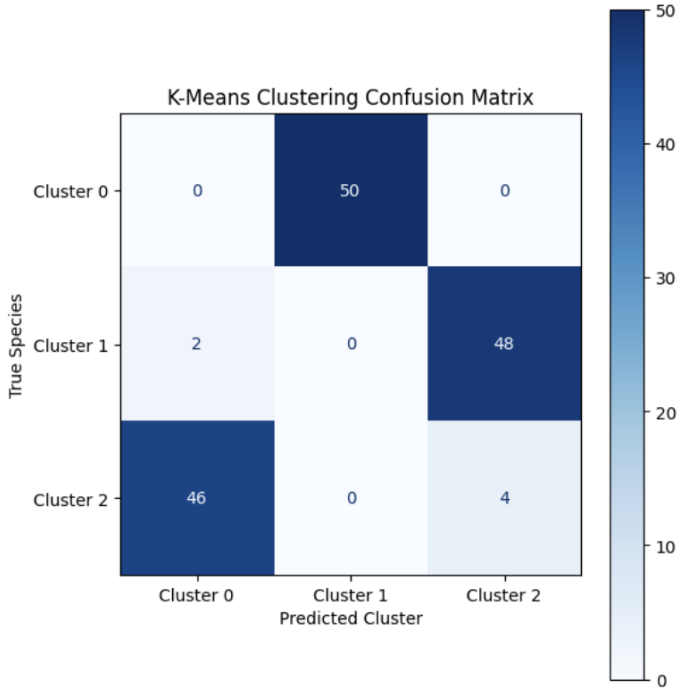
```python
        df = pd.read_csv(csv_path)
        # Fixed typo here: use c.strip().replace, not ace()
        df.columns = [c.strip().replace(' ', '_') for c in df.columns]
    except FileNotFoundError:
        iris = load_iris()
        df = pd.DataFrame(
            data=np.c_[iris['data'], iris['target']],
            columns=iris['feature_names'] + ['target']
        )
        df.columns = [c.strip().replace(' (cm)', '').replace(' ', '_')
                      for c in df.columns]
        df['species'] = df['target'].map(lambda x: iris['target_names'][int(x)])
    return df

def preprocess(df):
    """
    Select only petal_length & petal_width, then standard-scale.
    Returns scaled numpy array.
    """
    X = df[['petal_length', 'petal_width']].values
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, scaler

def plot_elbow(X_scaled, max_k=10):
    """
    Compute KMeans inertia for k=1..max_k and plot the elbow curve.
    Returns list of inertias.
    """
    inertias = []
    ks = range(1, max_k + 1)
    for k in ks:
        km = KMeans(n_clusters=k, random_state=42)
        km.fit(X_scaled)
        inertias.append(km.inertia_)
    plt.figure(figsize=(6, 4))
    plt.plot(ks, inertias, 'o-', linewidth=2)
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal k')
    plt.xticks(ks)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()
    return inertias

def run_kmeans(X_scaled, k):
    """
    Fit KMeans with k clusters, return labels and fitted model.
    """
    km = KMeans(n_clusters=k, random_state=42)
```

```
    labels = km.fit_predict(X_scaled)
    return km, labels

def plot_confusion(df, labels, k):
    """
    Builds and displays a confusion matrix comparing true species vs. cluster.
    """
    species_names = df['species'].unique()
    species_to_num = {name: idx for idx, name in enumerate(species_names)}
    true_nums = df['species'].map(species_to_num)

    cm = confusion_matrix(true_nums, labels)
    disp = ConfusionMatrixDisplay(
        confusion_matrix=cm,
        display_labels=[f"Cluster {i}" for i in range(k)]
    )
    fig, ax = plt.subplots(figsize=(6, 6))
    disp.plot(ax=ax, cmap='Blues', colorbar=True)
    ax.set_xlabel('Predicted Cluster')
    ax.set_ylabel('True Species')
    plt.title('K-Means Clustering Confusion Matrix')
    plt.tight_layout()
    plt.show()

    cm_df = pd.DataFrame(
        cm,
        index=[f"True: {name}" for name in species_names],
        columns=[f"Cluster {i}" for i in range(k)]
    )
    print("\nConfusion Matrix (counts):")
    print(cm_df)

def main():
    # 1) Load data
    df = load_data('iris.csv')
    if 'species' not in df.columns:
        print("Error: 'species' column not found.")
        return

    # 2) Preprocess
    X_scaled, scaler = preprocess(df)

    # 3) Elbow plot to decide k
    print("Generating elbow plot to find optimal k...")
    inertias = plot_elbow(X_scaled, max_k=10)

    # 4) From the elbow you'll typically see a bend at k=3
    optimal_k = 3
    print(f"Choosing k = {optimal_k} (you can adjust this based on the plot).")

    # 5) Run K-Means and assign clusters
```

```python
    km_model, labels = run_kmeans(X_scaled, optimal_k)
    df['cluster'] = labels

    # 6) Visualize clusters in feature space
    plt.figure(figsize=(6, 4))
    plt.scatter(
        X_scaled[:, 0], X_scaled[:, 1],
        c=labels, cmap='viridis', edgecolor='k', s=50
    )
    centroids = km_model.cluster_centers_
    plt.scatter(
        centroids[:, 0], centroids[:, 1],
        marker='X', c='red', s=200, label='Centroids'
    )
    plt.xlabel('Scaled Petal Length')
    plt.ylabel('Scaled Petal Width')
    plt.title(f'K-Means Clusters (k={optimal_k})')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

    # 7) Confusion matrix vs. true species
    plot_confusion(df, labels, optimal_k)

if __name__ == "__main__":
    main()
```



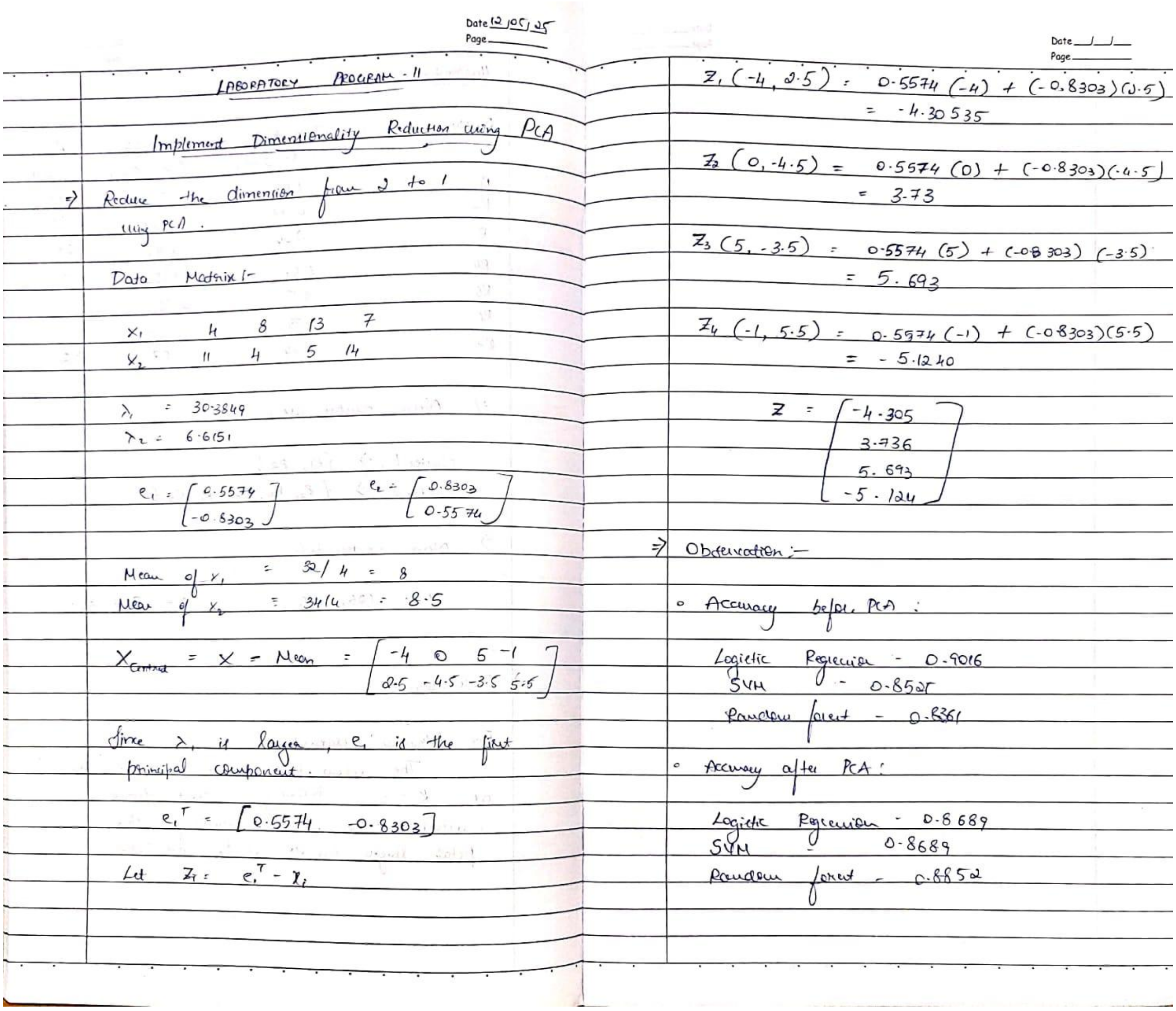



K-Means Clustering Confusion Matrix
True Species
Predicted Cluster
Cluster 0
Cluster 1
Cluster 2
0
50
0
2
0
48
46
0
4
50
40
30
20
10
0

LABORATORY PROGRAM – 11

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

OBSERVATION BOOK



Date 12/05/25

LABORATORY PROGRAM - 11

Implement Dimensionality Reduction using PCA

⇒ Reduce the dimension from 2 to 1 using PCA.

Data Matrix :-

x_1	4	8	13	7
x_2	11	4	5	14

$\lambda_1 = 30.3849$

$\lambda_2 = 6.6151$

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

Mean of x_1 = 32/4 = 8

Mean of x_2 = 34/4 = 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 the first principal component.

$e_1^T = [0.5574 \quad -0.8303]$

Let $Z_1 = e_1^T - x_1$

$Z_1(-4, 2.5) = 0.5574(-4) + (-0.8303)(2.5) = -4.30535$

$Z_2(0, -4.5) = 0.5574(0) + (-0.8303)(-4.5) = 3.73$

$Z_3(5, -3.5) = 0.5574(5) + (-0.8303)(-3.5) = 5.693$

$Z_4(-1, 5.5) = 0.5574(-1) + (-0.8303)(5.5) = -5.1240$

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

⇒ Observation :-

- 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 WITH OUTPUT

```
import pandas as pd

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

# Step 3: Split Features and Target
X = df.drop("target", axis=1)
y = df["target"]

# Step 4: Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

```
categorical_features = ["cp", "thal", "slope"]
numerical_features = [col for col in X.columns if col not in categorical_features]

preprocessor = ColumnTransformer(transformers=[
    ("num", StandardScaler(), numerical_features),
    ("cat", OneHotEncoder(), categorical_features)
])

# Step 5: Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "SVM": SVC(),
    "Random Forest": RandomForestClassifier()
}

# Step 7: Train and Evaluate Models (Before PCA)
print("Accuracy Before PCA:")
results = {}
for name, model in models.items():
    pipeline = Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("classifier", model)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"{name}: {acc:.4f}")

from sklearn.decomposition import PCA

print("\nAccuracy After PCA (n_components=5):")
pca_results = {}

for name, model in models.items():
    pipeline_pca = Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("pca", PCA(n_components=5)),
        ("classifier", model)
    ])
    pipeline_pca.fit(X_train, y_train)
    y_pred_pca = pipeline_pca.predict(X_test)
```

```
acc_pca = accuracy_score(y_test, y_pred_pca)
pca_results[name] = acc_pca
print(f"{name}: {acc_pca:.4f}")
```

```
Accuracy Before PCA:
Logistic Regression: 0.9016
SVM: 0.8525
Random Forest: 0.8361

Accuracy After PCA (n_components=5):
Logistic Regression: 0.8689
SVM: 0.8689
Random Forest: 0.8852
```