**Syllabus for B. Tech. III Year II Semester**

**Computer Science and Engineering (AI & ML)**

**MACHINE LEARNING LAB**

**B.Tech. III Year II Sem.**  L   T    P   C

0    0    3   1.5

**Code: 9LC65**

**Course Objective:**

The objective of this lab is to get an overview of the various machine learning

Techniques and can demonstrate them using python.

**Course Outcomes:**

Understand modern notions in predictive data analysis

Select data, model selection, model complexity and identify the trends

Understand a range of machine learning algorithms along with their strengths and weaknesses

Build predictive models from data and analyze their performance

**LIST OF EXPERIMENTS (MACHINE LEARNING):**

1. Write a python program to compute  Central Tendency Measures: Mean, Median, Mode  Measure of Dispersion: Variance, Standard Deviation
2. Write a Python program example to use basic Libraries such as Statistics, Math, Numpy and Scipy on a data set
3. Write a Python program to compute ML applications using of Python Libraries Pandas and Matplotlib
4. Write a Python program to implement Simple Linear Regression
5. Implementation of Multiple Linear Regression for House Price Prediction using sklearn
6. Implementation of Decision tree using sklearn and its parameter tuning
7. Implementation of KNN using sklearn
8. Implementation of Logistic Regression using sklearn
9. Implementation of K-Means Clustering
10. Performance analysis of Classification Algorithms on a specific dataset (Mini Project)

**TEXT BOOK:**

**1.** Stephen Marsland, ―Machine Learning – An Algorithmic Perspective, Second Edition, Chapman and Hall/CRC Machine Learning and Pattern Recognition Series, 2014.

**2.** Tom M Mitchell, ―Machine Learning, First Edition, McGraw Hill Education, 2013.

**REFERENCE BOOK:**

**1.** Peter Flach, ―Machine Learning: The Art and Science of Algorithms that Make Sense of Data, First Edition, Cambridge University Press, 2012.

2. Jason Bell, ―Machine learning – Hands on for Developers and Technical Professionals‖, First Edition, Wiley, 2014.

3. Ethem Alpaydin, ―Introduction to Machine Learning 3e (Adaptive Computation and Machine Learning Series), Third Edition, MIT Press, 2014.

**LIST OF EXPERIMENTS AND THEIR CODE**

**EXPERIMENT - 1**

**Write a python program to compute  Central Tendency Measures: Mean, Median, Mode  Measure of Dispersion: Variance, Standard Deviation**

# Import numpy and pandas libraries

import numpy as np  
import pandas as pd

# Define a sample dataset as a numpy array

data = np.array([10, 15, 20, 25, 30, 35, 40, 45, 50])

#Calculate the mean, median, and mode using numpy functions

mean = np.mean(data)  
median = np.median(data)  
mode = np.bincount(data).argmax()

# Calculate the variance and standard deviation using numpy functions

variance = np.var(data)  
std\_dev = np.std(data)

variance

**output:**

**166.66666666666666**

std\_dev

**output:**

12.909944487358056

# Print the results

print("The mean of the data is", mean)  
print("The median of the data is", median)  
print("The mode of the data is", mode)  
print("The variance of the data is", variance)  
print("The standard deviation of the data is", std\_dev)

**output:**

The mean of the data is 30.0

The median of the data is 30.0

The mode of the data is 10

The variance of the data is 166.66666666666666

The standard deviation of the data is 12.909944487358056

**OR**

Use Pandas Also

#Import pandas library

import pandas as pd

#Define a sample dataset as a pandas data frame

df = pd.DataFrame({"data": [10, 15, 20, 25, 30, 35, 40, 45, 50]})

#Calculate the mean, median, and mode using pandas methods

mean = df["data"].mean()  
median = df["data"].median()  
mode = df["data"].mode()

# Calculate the variance and standard deviation using pandas methods

variance = df["data"].var()  
std\_dev = df["data"].std()

variance

**output:**

166.66666666666666

std\_dev

**output:**

12.909944487358056

#Print the results

print("The mean of the data is", mean)

print("The median of the data is", median)

print("The mode of the data is", mode)

print("The variance of the data is", variance)

print("The standard deviation of the data is", std\_dev)

**output:**

The mean of the data is 30.0

The median of the data is 30.0

The mode of the data is 10

The variance of the data is 166.66666666666666

The standard deviation of the data is 12.909944487358056

**EXPERIMENT - 2**

**Write Python program example to use basic Libraries such as Statistics, Math, Numpy and Scipy on a data set**

**Sol). Hypothesis Testing with SciPy**:

1. **Suppose you have two samples (e.g., control group and treatment group) and want to compare their means. You can perform a t-test using SciPy.**

import numpy as np

import scipy.stats as stats

# Generate two sample datasets

control\_group = np.random.normal(10, 2, 50)

treatment\_group = np.random.normal(12, 2, 50)

# Perform an independent t-test

t\_statistic, p\_value = stats.ttest\_ind(control\_group, treatment\_group)

if p\_value < 0.05:

print("Reject null hypothesis: Significant difference between groups.")

else:

print("Fail to reject null hypothesis: No significant difference detected.")

Sol). Reject null hypothesis: Significant difference between groups.

**OR**

1. **Probability Distributions with SciPy**: **You can explore various probability distributions using SciPy. For example, let’s generate random samples from a normal distribution and plot the histogram.**

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

# Generate random samples from a normal distribution

data = np.random.normal(0, 1, 1000)

# Plot the histogram

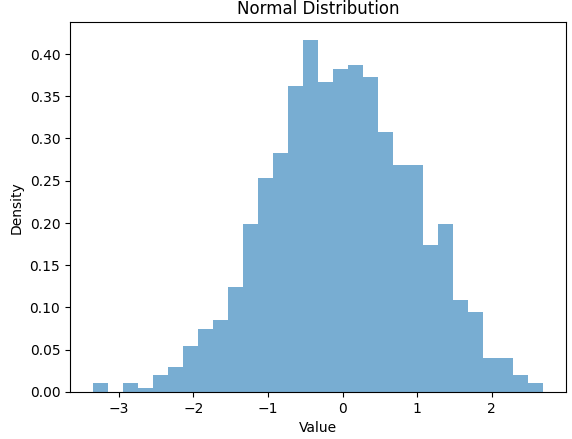
plt.hist(data, bins=30, density=True, alpha=0.6)

plt.xlabel("Value")

plt.ylabel("Density")

plt.title("Normal Distribution")

plt.show()



**OR**

1. **Perform basic mathematical operations like sum, product, and square root using numpy. Additionally, we use scipy.stats.pearsonr to calculate the correlation coefficient between two datasets**.

import numpy as np

from scipy.stats import pearsonr

# Generating two sets of data

data\_set1 = np.array([1, 2, 3, 4, 5])

data\_set2 = np.array([2, 4, 6, 8, 10])

# Mathematical operations using numpy

sum\_result = np.sum(data\_set1)

product\_result = np.prod(data\_set2)

sqrt\_result = np.sqrt(data\_set1)

print(f"Data Set 1: {data\_set1}")

print(f"Data Set 2: {data\_set2}")

print(f"Sum of Data Set 1: {sum\_result}")

print(f"Product of Data Set 2: {product\_result}")

print(f"Square Root of Data Set 1: {sqrt\_result}")

# Correlation calculation using scipy

correlation\_coefficient, p\_value = pearsonr(data\_set1, data\_set2)

print(f"Correlation Coefficient: {correlation\_coefficient:.2f}")

print(f"P-value: {p\_value:.4f}")

Sol). Correlation Coefficient: 1.00

P-value: 0.0000

**OR**

**d) It creates a sample dataset of 100 random numbers using the np.random. randn function** **from NumPy. The code then uses the numpy library to calculate the minimum, maximum, sum, and product of the dataset. Finally, it uses the scipy library to calculate the skewness and kurtosis of the dataset.**

import statistics

import math

import numpy as np

from scipy import stats

# Create a sample dataset

data = np.random.randn(100)

# Use numpy library to calculate minimum, maximum, sum, product

minimum = np.min(data)

maximum = np.max(data)

sum = np.sum(data)

product = np.prod(data)

# Use scipy library to calculate skewness, kurtosis

skewness = stats.skew(data)

kurtosis = stats.kurtosis(data)

# Print the results

print("Minimum:", minimum)

print("Maximum:", maximum)

print("Sum:", sum)

print("Product:", product)

print("Skewness:", skewness)

print("Kurtosis:", kurtosis)

sol).

Mean: 0.02154287718483102

Median: 0.09093786852985018

Mode: 0.5405153610325202

Standard deviation: 0.9491071378282817

Variance: 0.900804359076593

Minimum: -2.2308403479361667

Maximum: 2.468584599859246

Sum: 2.1542877184830993

Product: 2.4663208225411206e-27

Skewness: -0.048484625127097614

Kurtosis: -0.5763325175902998

**EXPERIMENT - 3**

**Write a Python program to compute ML applications using of Python Libraries Pandas and Matplotlib**

**NAÏVE BAYESIAN CLASIFIER**

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

import matplotlib.pyplot as plt

# Load Iris dataset as an example

iris = load\_iris()

X = iris.data

y = iris.target

# Create a DataFrame for visualization

iris\_df = pd.DataFrame(X, columns=iris.feature\_names)

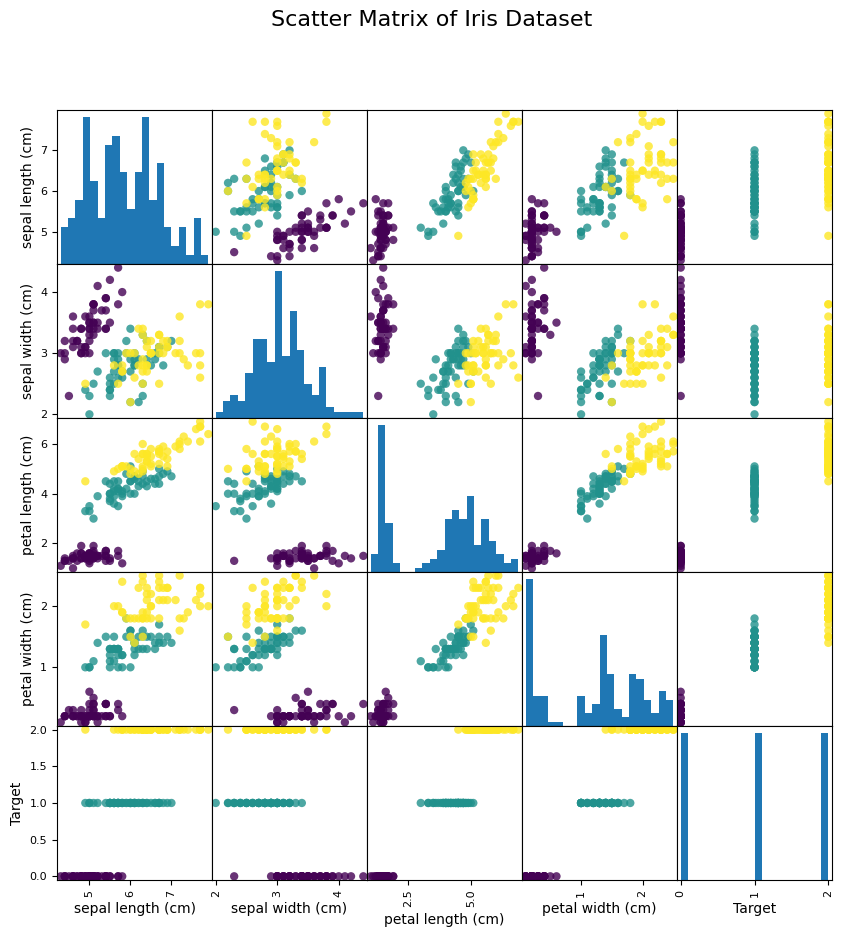
iris\_df['Target'] = y

# Visualize the data by plotting a scatter matrix

pd.plotting.scatter\_matrix(iris\_df, c=iris\_df['Target'], figsize=(10, 10), marker='o', hist\_kwds={'bins': 20}, alpha=0.8)

plt.suptitle('Scatter Matrix of Iris Dataset', size=16)

plt.show()

****

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a Naive Bayes model

nb\_model = GaussianNB()

# Train the model

nb\_model.fit(X\_train, y\_train)



# Make predictions on the test set

y\_pred = nb\_model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

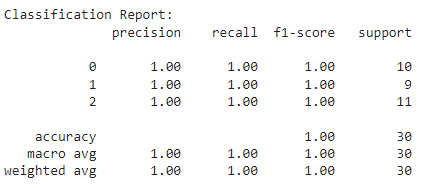
print("Accuracy:", accuracy)

Accuracy: 1.0

# Display classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))



**EXPERIMENT - 4**

**Write a Python program to implement Simple Linear Regression**

**ALGORITHM:**

Step 1: Create Database for Linear Regression

Step 2:Finding Hypothesis of inear Regression

Step 3:Training a Linear Regression model

Step 4:Evaluating the model

Step 5: Scikit-learn implementation

Step 6: End

**PROGRAM**

import numpy as np

import matplotlib.pyplot as plt

# Generate sample data

X = np.random.rand(100) \* 10

Y = 2 \* X + np.random.normal(0, 1, 100)

# Fit a linear regression model

coeffs = np.polyfit(X, Y, 1)

slope, intercept = coeffs

# Plot the data points and regression line

plt.scatter(X, Y, label="Data")

plt.plot(X, slope \* X + intercept, color='red', label="Regression Line")

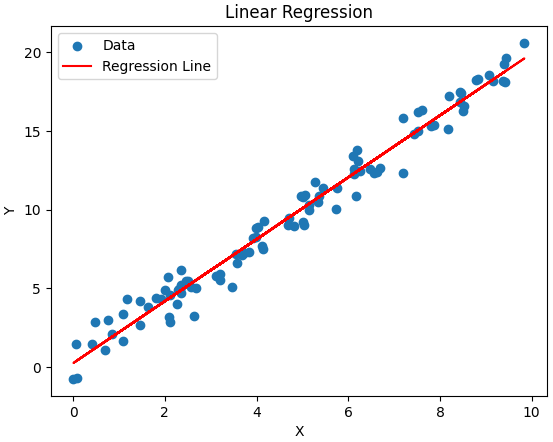
plt.xlabel("X")

plt.ylabel("Y")

plt.title("Linear Regression")

plt.legend()

plt.show()



**EXPERIMENT – 5**

**Implementation of Multiple Linear Regression for House Price Prediction using sklearn**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Load the Boston Housing dataset

url = "https://raw.githubusercontent.com/scikit-learn/scikit-learn/main/sklearn/datasets/data/boston\_house\_prices.csv"

data = pd.read\_csv(url, header=None)

# Display the first few rows of the dataset

print(data.head())

0 1 2 3 4 5 6 7 8 9 10 \

0 506 13 NaN NaN NaN NaN NaN NaN NaN NaN NaN

1 CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO

2 0.00632 18 2.31 0 0.538 6.575 65.2 4.09 1 296 15.3

3 0.02731 0 7.07 0 0.469 6.421 78.9 4.9671 2 242 17.8

4 0.02729 0 7.07 0 0.469 7.185 61.1 4.9671 2 242 17.8

11 12 13

0 NaN NaN NaN

1 B LSTAT MEDV

2 396.9 4.98 24

3 396.9 9.14 21.6

4 392.83 4.03 34.7

# Drop two rows by index

data = data.drop([0, 1])

print(data.columns.to\_list())

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

# Assuming the actual column names are as you mentioned

data.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

# Display the original DataFrame

print("Original DataFrame:")

print(data)

Original DataFrame:

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO \

2 0.00632 18 2.31 0 0.538 6.575 65.2 4.09 1 296 15.3

3 0.02731 0 7.07 0 0.469 6.421 78.9 4.9671 2 242 17.8

4 0.02729 0 7.07 0 0.469 7.185 61.1 4.9671 2 242 17.8

5 0.03237 0 2.18 0 0.458 6.998 45.8 6.0622 3 222 18.7

6 0.06905 0 2.18 0 0.458 7.147 54.2 6.0622 3 222 18.7

.. ... .. ... ... ... ... ... ... .. ... ...

503 0.06263 0 11.93 0 0.573 6.593 69.1 2.4786 1 273 21

504 0.04527 0 11.93 0 0.573 6.12 76.7 2.2875 1 273 21

505 0.06076 0 11.93 0 0.573 6.976 91 2.1675 1 273 21

506 0.10959 0 11.93 0 0.573 6.794 89.3 2.3889 1 273 21

507 0.04741 0 11.93 0 0.573 6.03 80.8 2.505 1 273 21

B LSTAT MEDV

2 396.9 4.98 24

3 396.9 9.14 21.6

4 392.83 4.03 34.7

5 394.63 2.94 33.4

6 396.9 5.33 36.2

.. ... ... ...

503 391.99 9.67 22.4

504 396.9 9.08 20.6

505 396.9 5.64 23.9

506 393.45 6.48 22

507 396.9 7.88 11.9

[506 rows x 14 columns]

# Display the DataFrame after dropping the row

print("\nDataFrame after dropping the specified row:")

print(data\_dropped)

DataFrame after dropping the specified row:

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO \

0 506 13 NaN NaN NaN NaN NaN NaN NaN NaN NaN

2 0.00632 18 2.31 0 0.538 6.575 65.2 4.09 1 296 15.3

3 0.02731 0 7.07 0 0.469 6.421 78.9 4.9671 2 242 17.8

4 0.02729 0 7.07 0 0.469 7.185 61.1 4.9671 2 242 17.8

5 0.03237 0 2.18 0 0.458 6.998 45.8 6.0622 3 222 18.7

.. ... .. ... ... ... ... ... ... ... ... ...

503 0.06263 0 11.93 0 0.573 6.593 69.1 2.4786 1 273 21

504 0.04527 0 11.93 0 0.573 6.12 76.7 2.2875 1 273 21

505 0.06076 0 11.93 0 0.573 6.976 91 2.1675 1 273 21

506 0.10959 0 11.93 0 0.573 6.794 89.3 2.3889 1 273 21

507 0.04741 0 11.93 0 0.573 6.03 80.8 2.505 1 273 21

B LSTAT MEDV

0 NaN NaN NaN

2 396.9 4.98 24

3 396.9 9.14 21.6

4 392.83 4.03 34.7

5 394.63 2.94 33.4

.. ... ... ...

503 391.99 9.67 22.4

504 396.9 9.08 20.6

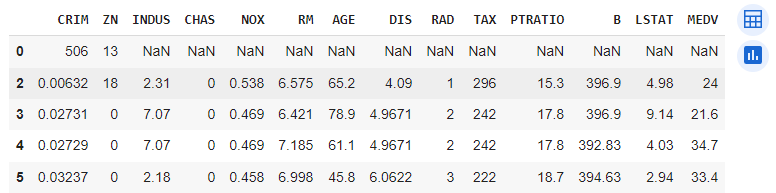
505 396.9 5.64 23.9

506 393.45 6.48 22

507 396.9 7.88 11.9

[507 rows x 14 columns]

data\_dropped.head()

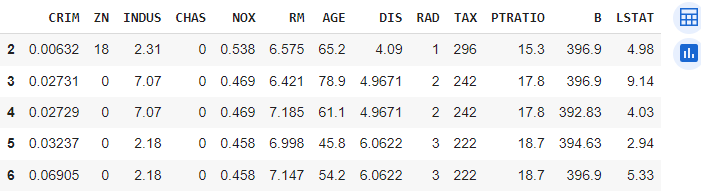


# Split the dataset into features (X) and target variable (y)

X = data.drop('MEDV', axis=1)

y = data['MEDV']

X.head()



y.head()

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train.head()

X\_test.head()

y\_train.head()

y\_test.head()

# Create a linear regression model

model = LinearRegression()

# Train the model

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

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

print(f'R-squared Score: {r2}')

Mean Squared Error: 24.291119474973478

R-squared Score: 0.6687594935356326

!pip install seaborn

import seaborn as sns

# Plotting the predicted vs actual values

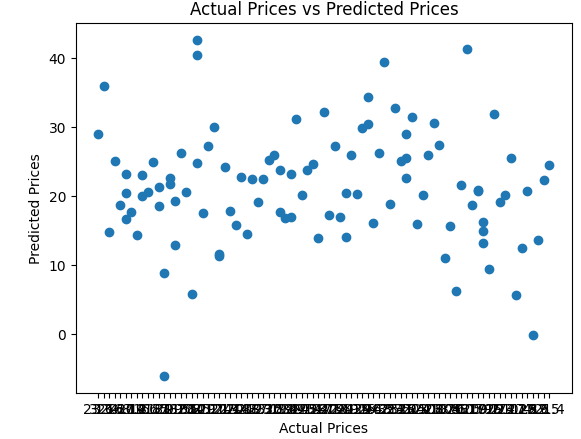
plt.scatter(y\_test, y\_pred)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual Prices vs Predicted Prices")

plt.show()



import seaborn as sns

import matplotlib.pyplot as plt

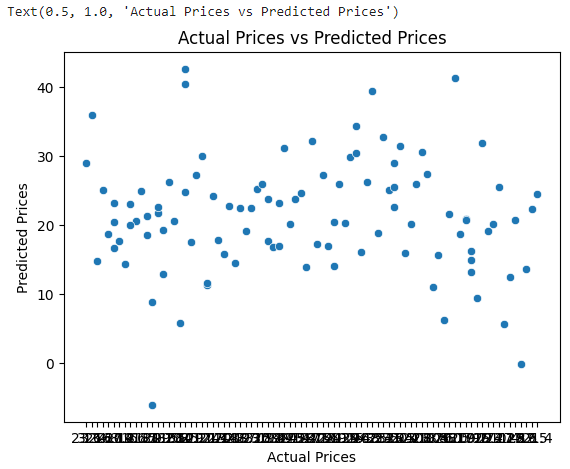
# Create a scatter plot using Seaborn

sns.scatterplot(x=y\_test, y=y\_pred)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual Prices vs Predicted Prices")



# Convert y\_test and y\_pred to numeric arrays

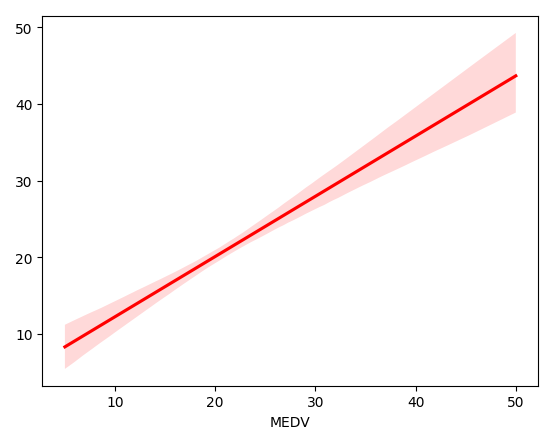
y\_test = pd.to\_numeric(y\_test)

y\_pred = pd.to\_numeric(y\_pred)

# Add a regression line to the scatter plot

sns.regplot(x=y\_test, y=y\_pred, scatter=False, color='red')

plt.show()



# Convert y\_test and y\_pred to numeric arrays

y\_test = pd.to\_numeric(y\_test)

y\_pred = pd.to\_numeric(y\_pred)

# Add a regression line to the scatter plot

sns.regplot(x=y\_test, y=y\_pred, scatter=True, color='red')

# Create a scatter plot using Seaborn

#sns.scatterplot(x=y\_test, y=y\_pred)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual Prices vs Predicted Prices")

plt.show()



**EXPERIMENT – 6**

**Implementation of Decision tree using sklearn and its parameter tuning**

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report

# Load Iris dataset as an example

iris = load\_iris()

X = iris.data

y = iris.target

# Split the dataset into training and testing sets

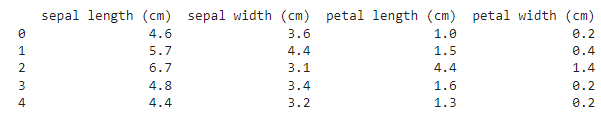
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Convert NumPy array to Pandas DataFrame

X\_train\_df = pd.DataFrame(X\_train, columns=iris.feature\_names)

# Display the first few rows

print(X\_train\_df.head())

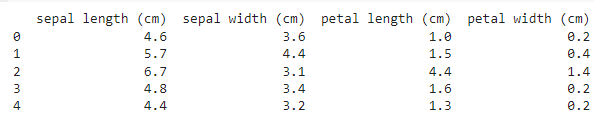


# Convert NumPy array to Pandas DataFrame

X\_test\_df = pd.DataFrame(X\_test, columns=iris.feature\_names)

# Display the first few rows

print(X\_train\_df.head())



print(y\_train.shape)

(120,)

print(len(iris.feature\_names))

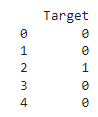
4

# Convert 1D array to Pandas DataFrame

y\_train\_df = pd.DataFrame({'Target': y\_train})

# Display the first few rows

print(y\_train\_df.head())

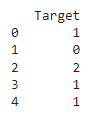


# Convert 1D array to Pandas DataFrame

y\_test\_df = pd.DataFrame({'Target': y\_test})

# Display the first few rows

print(y\_test\_df.head())



# Create a Decision Tree model

model = DecisionTreeClassifier()

# Define the hyperparameter grid for tuning

param\_grid = {

'criterion': ['gini', 'entropy'],

'max\_depth': [None, 5, 10, 15],

'min\_samples\_split': [2, 5, 10],

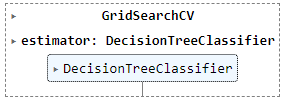
'min\_samples\_leaf': [1, 2, 4]

}

# Perform GridSearchCV for hyperparameter tuning

grid\_search = GridSearchCV(model, param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)



# Get the best parameters from the grid search

best\_params = grid\_search.best\_params\_

print("Best Hyperparameters:", best\_params)

Best Hyperparameters: {'criterion': 'entropy', 'max\_depth': None, 'min\_samples\_leaf': 4, 'min\_samples\_split': 2}

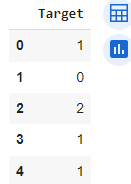
# Use the best model to make predictions

best\_model = grid\_search.best\_estimator\_

y\_pred = best\_model.predict(X\_test)

y\_pred\_df = pd.DataFrame({'Target': y\_pred})

y\_pred\_df.head()



# Evaluate the model

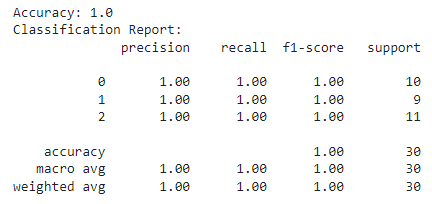
accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

# Display classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))



**EXPERIMENT-7**

**Implementation of KNN using sklearn**

**ALGORITHM**

1. Load the Iris dataset using load\_iris() from scikit-learn.
2. Split the dataset into training and testing sets using train\_test\_split.
3. Create a KNN model with k=3 using KNeighborsClassifier.
4. Train the model on the training set using fit.
5. Make predictions on the test set using predict.
6. Evaluate the model's performance using accuracy and a classification report.

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report

# Load Iris dataset as an example

iris = load\_iris()

X = iris.data

y = iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a KNN model with k=3

knn\_model = KNeighborsClassifier(n\_neighbors=3)

# Train the model

knn\_model.fit(X\_train, y\_train)



# Make predictions on the test set

y\_pred = knn\_model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

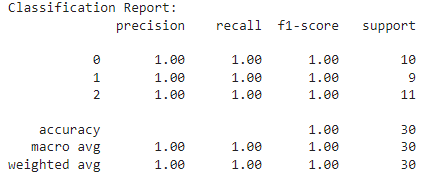
print("Accuracy:", accuracy)

Accuracy: 1.0

# Display classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred)



**EXPERIMENT – 8**

**Implementation of Logistic Regression using sklearn**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Sample dataset (you can replace this with your own dataset)

data = {'feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

        'feature2': [0, 1, 0, 1, 0, 1, 0, 1, 1, 0],

        'target': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]}

df = pd.DataFrame(data)

# Visualize the relationship between features

plt.scatter(df['feature1'], df['feature2'], c=df['target'], cmap='viridis')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('Sample Dataset - Feature 1 vs Feature 2')

plt.show()

# Split the dataset into features (X) and target variable (y)

X = df[['feature1', 'feature2']]

y = df['target']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train a Logistic Regression classifier

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set

predictions = model.predict(X\_test)

# Evaluate the accuracy of the model

accuracy = accuracy\_score(y\_test, predictions)

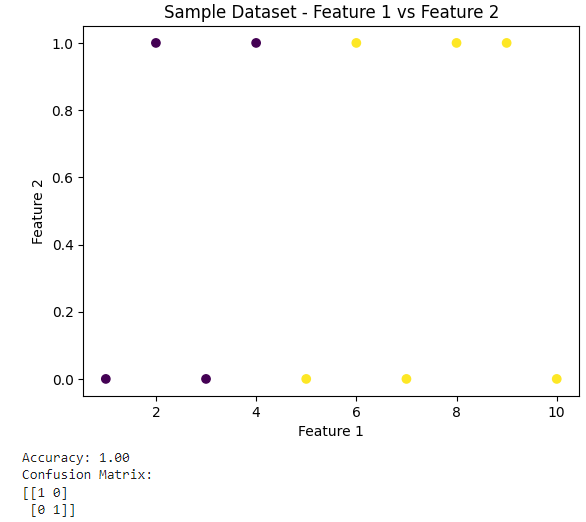
conf\_matrix = confusion\_matrix(y\_test, predictions)

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

print("Confusion Matrix:")

print(conf\_matrix)

Sol).



**EXPERIMENT – 9**

**Implementation of K-Means Clustering**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Load Iris dataset as an example

iris = load\_iris()

X = iris.data[:, :2]  # Using only the first two features (sepal length and width)

y = iris.target

# Standardize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Create a DataFrame for visualization

iris\_df = pd.DataFrame(X\_scaled, columns=['sepal length (cm)', 'sepal width (cm)'])

# Visualize the standardized data

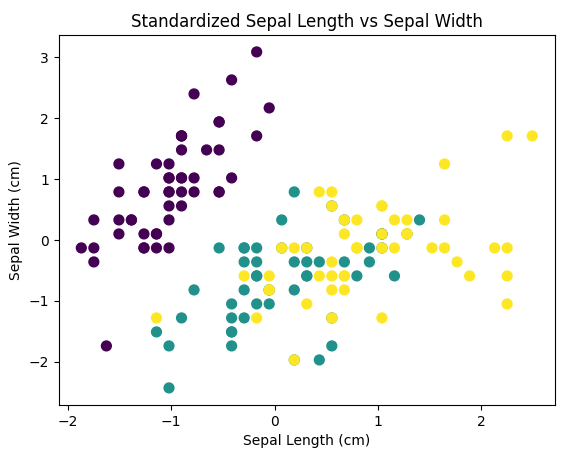
plt.scatter(iris\_df['sepal length (cm)'], iris\_df['sepal width (cm)'], c=y, cmap='viridis', s=50)

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.title('Standardized Sepal Length vs Sepal Width')

plt.show()



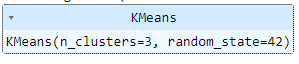
# Apply K-Means clustering with k=3

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans.fit(X\_scaled)

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/\_kmeans.py:870: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning

warnings.warn(



# Get the cluster centers and labels

cluster\_centers = kmeans.cluster\_centers\_

labels = kmeans.labels\_

# Visualize the clusters

plt.scatter(iris\_df['sepal length (cm)'], iris\_df['sepal width (cm)'], c=labels, cmap='viridis', s=50)

plt.scatter(cluster\_centers[:, 0], cluster\_centers[:, 1], c='red', marker='X', s=200, label='Cluster Centers')

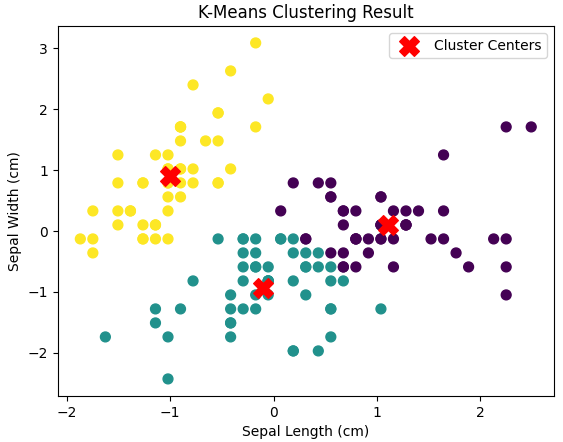
plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.title('K-Means Clustering Result')

plt.legend()

plt.show()



**EXPERIMENT – 10**

**Performance analysis of Classification Algorithms on a specific dataset (Mini Project)**

Give Two Weeks of Time to Students to submit Project