**LAB FILE**

**MACHINE LEARNING LAB**

**(ML‐407P)**

Student Name: Lakshay Sharma

Roll No: 02396402721

Semester: 7th

Group: 7-CSE-AIML-II-C

Faculty name: Dr. Sachin Gupta



Department Of Computer Science & Engineering

Maharaja Agrasen Institute of Technology, PSP area, Sector – 22, Rohini,

New Delhi – 110085

(Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)

**2024**



## MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

**VISION**

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

## MISSION

The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:

**Engineering Hardware – Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and

encouraging inquisitive nature.

**Life – Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

**Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

**Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

**Digitization of Learning Processes**

The Institute provides seamless opportunities for innovative learning in all Engineering and Management disciplines through digitization of learning processes using analysis, synthesis, simulation, graphics,

tutorials and related tools to create a platform for multi-disciplinary approach.

**Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities so that they become successful entrepreneurs and responsible citizens.



## MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

**COMPUTER SCIENCE & ENGINEERING DEPARTMENT**

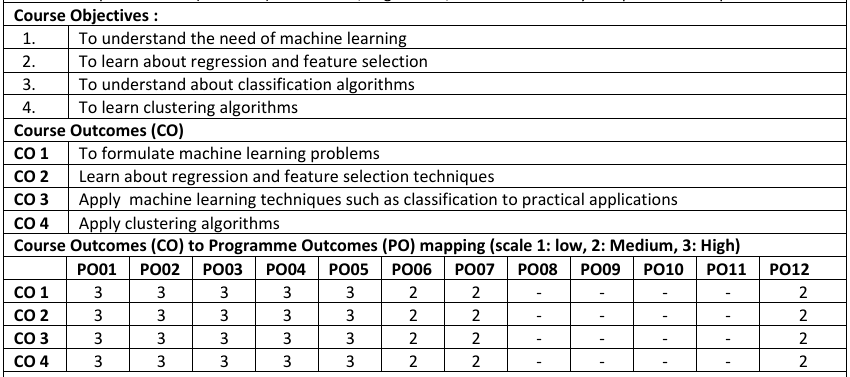
# VISION

“To be centre of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values.”

# MISSION

“To foster an open, multidisciplinary and highly collaborative research environment to produce world-class engineers capable of providing innovative solutions to real life problems and fulfil societal needs.”

**Course Objectives and Outcomes**



Department of Computer Science and Engineering Rubrics for Lab Assessment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | | **0** | **1** | **2** | **3** |
| **Missing** | **Inadequate** | **Needs Improvement** | **Adequate** |
| R1 | Is able to identify the problem to be solved and define the objectives of the experiment. | No mention is made of the problem to be solved. | An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable. | The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors. | The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors. |
| R2 | Is able to design a reliable experiment that solves the problem. | The experiment does not solve the problem. | The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution. | The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable  solution. | The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution. |
| R3 | Is able to communicate the details of an experimental procedure clearly and completely. | Diagrams are missing and/or experimental procedure is missing or extremely vague. | Diagrams are present but unclear and/or experimental procedure is present but important details are missing. | Diagrams and/or experimental procedure are present but with minor omissions or vague details. | Diagrams and/or experimental procedure are clear and complete. |
| R4 | Is able to record and represent data in a meaningful way. | Data are either absent or incomprehensible. | Some important data are absent or incomprehensible. | All important data are present, but recorded in a way that requires some effort to comprehend. | All important data are present, organized and recorded clearly. |
| R5 | Is able to make a judgment about the results of the experiment. | No discussion is presented about the results of the experiment . | A judgment is made about the results, but it is not reasonable or coherent. | An acceptable judgment is made about the result, but the reasoning is flawed or incomplete. | An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered. |

# PRACTICAL RECORD

**PAPER CODE : ML‐407P**

Name of Student : Lakshay Sharma

University roll : 02396402721

Semester : 7th (4th year)

Group : 7-CSE-AIML-II-C

## PRACTICAL DETAILS

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exp. no** | **Experiment Name** | **Date of performance** | **Date of checking** | **R1 (3)** | **R2 (3)** | **R3 (3)** | **R4 (3)** | **R5 (3)** | **Total Marks (15)** | **Signature** |
| 1. |  |  |  |  |  |  |  |  |  |  |
| 2. |  |  |  |  |  |  |  |  |  |  |
| 3. |  |  |  |  |  |  |  |  |  |  |
| 4. |  |  |  |  |  |  |  |  |  |  |
| 5. |  |  |  |  |  |  |  |  |  |  |

Experiments according to the lab syllabus prescribed by GGSIPU

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exp. no** | **Experiment Name** | **Date of Performance** | **Date of checking** | **R1 (3)** | **R2 (3)** | **R3 (3)** | **R4 (3)** | **R5 (3)** | **Total Marks (15)** | **Signature** |
| 6. |  |  |  |  |  |  |  |  |  |  |
| 7. |  |  |  |  |  |  |  |  |  |  |
| 8. |  |  |  |  |  |  |  |  |  |  |
| 9. |  |  |  |  |  |  |  |  |  |  |
| 10. |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

# EXPERIMENT

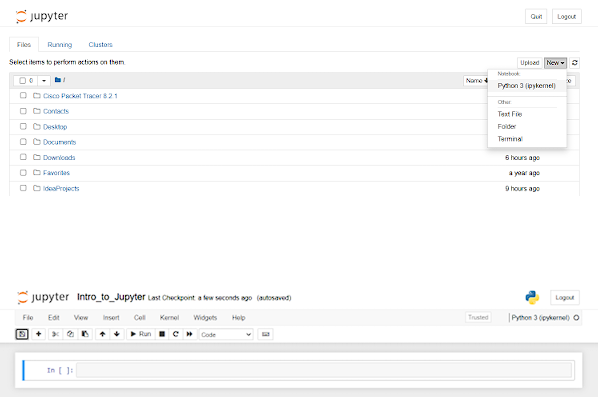
### AIM: Introduction to JUPYTER IDE and its libraries Pandas and NumPy

**1.1. Launch Jupyter Notebook on your computer.**

**1.2. Create a new notebook by clicking on the "New" button and selecting "Python 3" from the dropdown menu.**

**1.3. Familiarize yourself with the interface: cells, toolbar, menu options, etc.**

**1.4. Rename your notebook to "Intro\_to\_Jupyter.ipynb".**



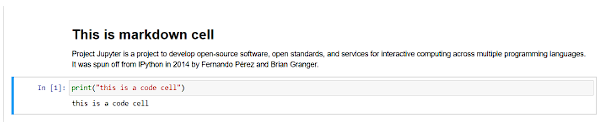
**2.1. Understand the two main types of cells: Code and Markdown.**

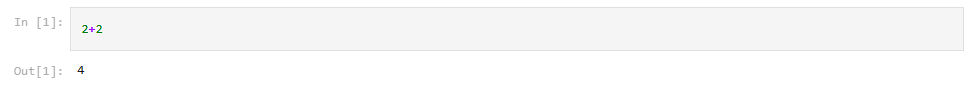
**2.2. Create a new Markdown cell and write a brief introduction to Jupyter Notebooks.**

**2.3. Create a new Code cell and write a simple Python expression (e.g., 2 + 2).**

**2.4. Execute the Code cell by pressing Shift + Enter and observe the output.**

**2.5. Experiment with adding, deleting, and moving cells within the notebook.**



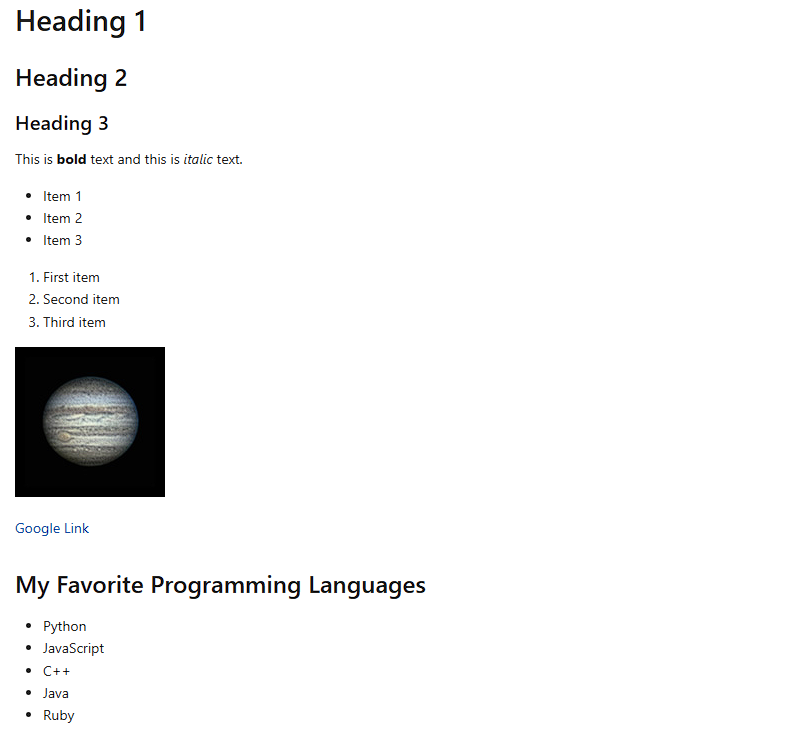


**3.1. Learn basic Markdown syntax for text formatting (e.g., headings, bold, italic).**

**3.2. Practice creating Markdown cells with formatted text and headings.**

**3.3. Insert images and hyperlinks using Markdown syntax.**

**3.4. Create a Markdown cell with a list of your favorite programming languages**

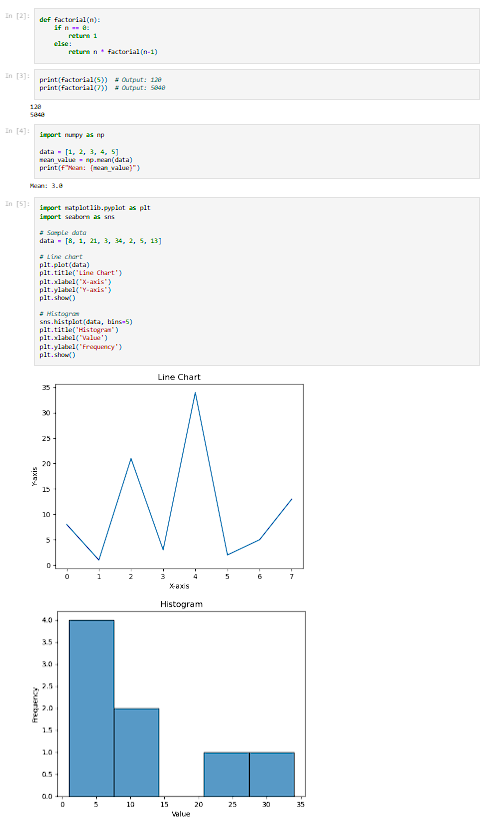


**4.1. Write a Python function that calculates the factorial of a given number.**

**4.2. Test your function with different input values and observe the results.**

**4.3. Import a Python library (e.g., NumPy) and use it to perform a mathematical operation (e.g., calculate mean, median).**

**4.4. Visualize data using Matplotlib or Seaborn libraries within a Jupyter Notebook.**



**5.1. Understand what a kernel is in the context of Jupyter Notebooks.**

**5.2. Learn how to switch kernels in a Jupyter Notebook.**

**5.3. If available, switch your notebook's kernel to a different programming language (e.g., R, Julia) and run a simple command.**

**5.4. Reset the kernel and clear the outputs. Observe how this affects your notebook.**

In the context of Jupyter Notebooks, a kernel is a computational engine that executes the code contained in the notebook document. Each notebook is associated with a specific kernel, which can be Python, R, Julia, or other languages, depending on the setup.

To switch kernels in a Jupyter Notebook, you can follow these steps:

1. Click on the "Kernel" menu at the top of the notebook interface.
2. Select "Change Kernel" from the dropdown menu.
3. Choose the desired kernel from the list of available kernels.

**6.1. Explore different file formats for exporting your Jupyter Notebook: PDF, HTML, slideshows, etc.**

**6.2. Export your notebook as a PDF and HTML file.**

**6.3. Discuss when different export formats might be useful.**

Jupyter Notebooks offer various file formats for exporting your work, including PDF, HTML,

slideshows, and more. Each format serves different purposes and may be useful in different scenarios.

To export your notebook as a PDF and HTML file:

1. Click on "File" in the Jupyter Notebook interface.
2. Navigate to "Download as" in the dropdown menu.
3. Select "PDF" to export the notebook as a PDF file.
4. Similarly, select "HTML" to export the notebook as an HTML file.

Discussing When Different Export Formats Might be Useful:

* PDF: Exporting a notebook to PDF format is useful when you need a static and printable version of your notebook. It's commonly used for sharing reports, documentation, or academic papers where formatting needs to be preserved.
* HTML: Exporting a notebook to HTML format is useful when you want to share your work online or embed it within a website. HTML files retain interactivity, allowing viewers to interact with code cells and visualizations. It's also helpful for creating online tutorials or interactive documentation
* Slideshows: Jupyter Notebooks can also be exported as slideshows using tools like RISE. This format is useful for presenting your work in a slideshow format, making it easy to navigate through different sections of the notebook during a presentation.
* Markdown: While not a direct export format, Jupyter Notebooks support exporting cells as Markdown. This can be useful for converting code and text into Markdown format for use in other documents or README files.
* Choosing the appropriate export format depends on your intended audience and how you plan to share or present your work. Each format has its own advantages and use cases, so it's helpful to be familiar with the options available in Jupyter Notebooks.

**7.1. Learn about nbviewer, an online tool to view Jupyter Notebooks.**

**7.2. Upload your notebook to a public GitHub repository.**

**7.3. Use nbviewer to view your notebook online and share the link with on the document you submit online.**

**8.1. Learn about interactive widgets in Jupyter and their purpose.**

**8.2. Create a simple interactive widget (e.g., a slider to input a number).**

**8.3. Link the widget to a Python function (e.g., modifying a plot based on the widget's input).**

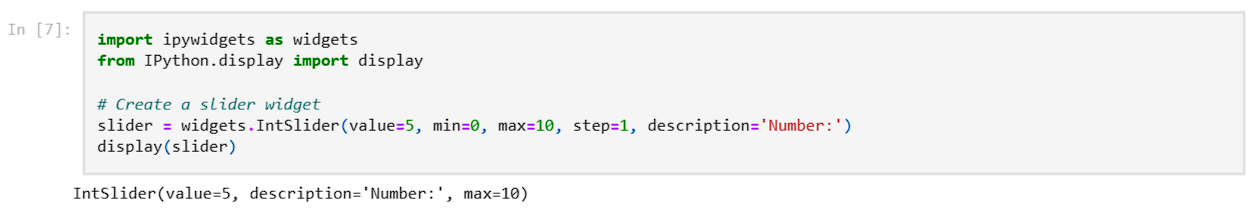
**9.1. Learn about Jupyter Notebook extensions (e.g., Jupyter\_contrib\_nbextensions).**

**9.2. Install any chosen extension and demonstrate its functionality in your notebook.**

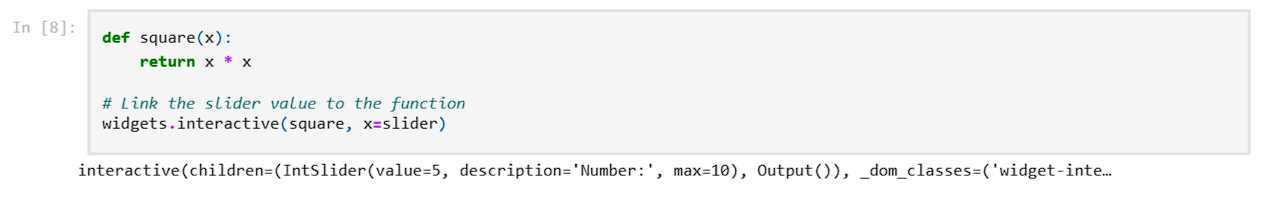
*Understanding Interactive Widgets:*

Interactive widgets in Jupyter Notebook allow users to create interactive graphical user interfaces (GUIs) directly within the notebook environment. These widgets enable users to interact with and manipulate data, visualizations, and other elements dynamically, enhancing the exploratory data analysis process and enabling more engaging presentations.

*Creating a simple widget:*



*Linking a widget to a python function:*



*Learning About Nbviewer:*

Nbviewer is an online service that allows you to view Jupyter Notebooks without needing to run Jupyter locally. It renders the notebooks stored in public GitHub repositories or shared Gists, providing a convenient way to share and view notebooks online.

*Uploading your notebook to a public Github repository:*

To upload your notebook to a public GitHub repository, follow these steps:

1. Create a new GitHub repository or use an existing one.
2. Upload your Jupyter Notebook file (.ipynb extension) to the repository.3.
3. Make sure the repository is set to public so that it can be accessed by nbviewer.

*Using Nbviewer to view your notebook online:*

To view your notebook using nbviewer:

1. Open your web browser and go to nbviewer.
2. Enter the URL of your notebook's GitHub repository or Gist in the provided field.
3. Click on "Go!" to view your notebook rendered by nbviewer.

After viewing your notebook in nbviewer, you can copy the URL from the browser's address bar and share it with others. This link will allow anyone to view your notebook online without needing to download or run it locally.

# EXPERIMENT

### AIM: Program to demonstrate Simple Linear Regression

**CODE:**

1. **A program for linear regression model using ML**

*# DATA IMPORT*

import tensorflow as tf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

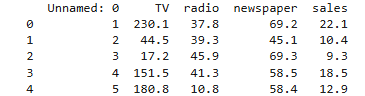
from sklearn.metrics import mean\_squared\_error

*# Load the Advertising Data*

url = "https://raw.githubusercontent.com/selva86/datasets/master/Advertising.csv"

data = pd.read\_csv(url)

print(data.head())



*# Prepare the data*

X1 = data['TV'].values

X2 = data['radio'].values

X3 = data['newspaper'].values

y = data['sales'].values

*# Split the data*

X1\_train, X1\_test, y\_train, y\_test = train\_test\_split(X1, y, test\_size=0.2, random\_state=42)

X2\_train, X2\_test, \_, \_ = train\_test\_split(X2, y, test\_size=0.2, random\_state=42)

X3\_train, X3\_test, \_, \_ = train\_test\_split(X3, y, test\_size=0.2, random\_state=42)

*# Define the models*

model1 = tf.keras.Sequential([

*# tf.keras.layers.Dense(128, input\_shape=(1,)),*

tf.keras.layers.Dense(1, input\_shape=(1,))

])

model2 = tf.keras.Sequential([

tf.keras.layers.Dense(1, input\_shape=(1,))

*# tf.keras.layers.Dense(1, input\_shape=(X2\_train.shape[1],))*

])

model3 = tf.keras.Sequential([

tf.keras.layers.Dense(1, input\_shape=(1,))

*# tf.keras.layers.Dense(1, input\_shape=(X3\_train.shape[1],))*

])

*# Compile the models*

model1.compile(optimizer='adam', loss='mean\_squared\_error')

model2.compile(optimizer='adam', loss='mean\_squared\_error')

model3.compile(optimizer='adam', loss='mean\_squared\_error')

*# Train the models*

history1 = model1.fit(X1\_train, y\_train, epochs=100, validation\_split=0.2, verbose=0)

history2 = model2.fit(X2\_train, y\_train, epochs=100, validation\_split=0.2, verbose=0)

history3 = model3.fit(X3\_train, y\_train, epochs=100, validation\_split=0.2, verbose=0)

*# Evaluate the models*

loss1 = model1.evaluate(X1\_test, y\_test)

loss2 = model2.evaluate(X2\_test, y\_test)

loss3 = model3.evaluate(X3\_test, y\_test)

print(f"Test Loss of TV model: {loss1}")

print(f"Test Loss of Radio model: {loss2}")

print(f"Test Loss of Newspaper model: {loss3}")

*# Plot training and validation loss*

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

*# Training plot for TV advertising budget vs Sales*

axes[0].plot(history1.history['loss'], label='Train Loss')

axes[0].plot(history1.history['val\_loss'], label='Validation Loss')

axes[0].set\_xlabel('Epochs')

axes[0].set\_ylabel('Loss')

axes[0].legend()

axes[0].set\_title('TV vs Sales - Training and Validation Loss')

*# Training plot for Radio advertising budget vs Sales*

axes[1].plot(history2.history['loss'], label='Train Loss')

axes[1].plot(history2.history['val\_loss'], label='Validation Loss')

axes[1].set\_xlabel('Epochs')

axes[1].set\_ylabel('Loss')

axes[1].legend()

axes[1].set\_title('Radio vs Sales - Training and Validation Loss')

*# Training plot for Newspaper advertising budget vs Sales*

axes[2].plot(history3.history['loss'], label='Train Loss')

axes[2].plot(history3.history['val\_loss'], label='Validation Loss')

axes[2].set\_xlabel('Epochs')

axes[2].set\_ylabel('Loss')

axes[2].legend()

axes[2].set\_title('Newspaper vs Sales - Training and Validation Loss')

*# Adjust layout to prevent overlap*

plt.tight\_layout()

plt.show()

*# Plot the regression lines*

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

*# Regression plot for TV advertising budget vs Sales*

y\_pred1 = model1.predict(X1\_test)

axes[0].scatter(X1\_test, y\_test, color='blue', label='Actual')

axes[0].plot(X1\_test, y\_pred1, color='red', label='Predicted')

axes[0].set\_xlabel('TV Budget')

axes[0].set\_ylabel('Sales')

axes[0].legend()

axes[0].set\_title('TV vs Sales - Regression Line')

*# Regression plot for Radio advertising budget vs Sales*

y\_pred2 = model2.predict(X2\_test)

axes[1].scatter(X2\_test, y\_test, color='blue', label='Actual')

axes[1].plot(X2\_test, y\_pred2, color='red', label='Predicted')

axes[1].set\_xlabel('Radio Budget')

axes[1].set\_ylabel('Sales')

axes[1].legend()

axes[1].set\_title('Radio vs Sales - Regression Line')

*# Regression plot for Newspaper advertising budget vs Sales*

y\_pred3 = model3.predict(X3\_test)

axes[2].scatter(X3\_test, y\_test, color='blue', label='Actual')

axes[2].plot(X3\_test, y\_pred3, color='red', label='Predicted')

axes[2].set\_xlabel('Newspaper Budget')

axes[2].set\_ylabel('Sales')

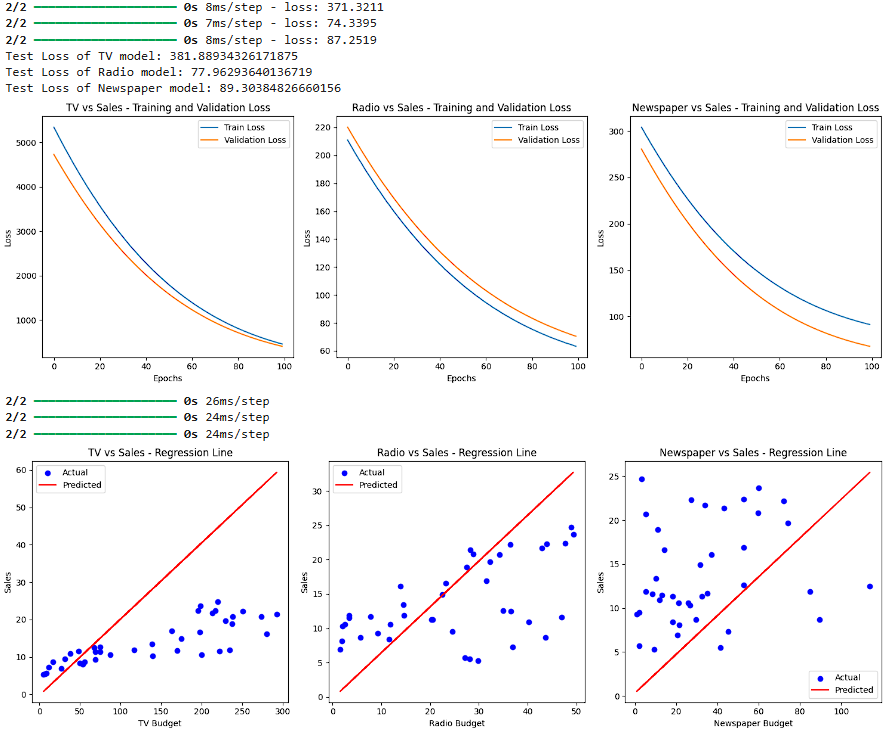
axes[2].legend()

axes[2].set\_title('Newspaper vs Sales - Regression Line')

*# Adjust layout to prevent overlap*

plt.tight\_layout()

plt.show()



1. **A program for linear regression model using scikit-learn but no machine learning**

*# Prepare the data*

X1 = data[['TV']].values

X2 = data[['radio']].values

X3 = data[['newspaper']].values

y = data['sales'].values

*# Split the data*

X1\_train, X1\_test, y\_train, y\_test = train\_test\_split(X1, y, test\_size=0.2, random\_state=42)

X2\_train, X2\_test, \_, \_ = train\_test\_split(X2, y, test\_size=0.2, random\_state=42)

X3\_train, X3\_test, \_, \_ = train\_test\_split(X3, y, test\_size=0.2, random\_state=42)

*# Define the models*

model1 = LinearRegression()

model2 = LinearRegression()

model3 = LinearRegression()

*# Train the models*

model1.fit(X1\_train, y\_train)

model2.fit(X2\_train, y\_train)

model3.fit(X3\_train, y\_train)

*# Predict and evaluate the models*

y\_pred1 = model1.predict(X1\_test)

y\_pred2 = model2.predict(X2\_test)

y\_pred3 = model3.predict(X3\_test)

mse1 = mean\_squared\_error(y\_test, y\_pred1)

mse2 = mean\_squared\_error(y\_test, y\_pred2)

mse3 = mean\_squared\_error(y\_test, y\_pred3)

print(f"Mean Squared Error of TV model: {mse1}")

print(f"Mean Squared Error of Radio model: {mse2}")

print(f"Mean Squared Error of Newspaper model: {mse3}")

*# Plot the training loss*

*# Note: `scikit-learn` doesn't provide training loss directly like TensorFlow.*

*# Plot the regression lines*

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

*# Regression plot for TV advertising budget vs Sales*

axes[0].scatter(X1\_test, y\_test, color='blue', label='Actual')

axes[0].plot(X1\_test, y\_pred1, color='red', label='Predicted')

axes[0].set\_xlabel('TV Budget')

axes[0].set\_ylabel('Sales')

axes[0].legend()

axes[0].set\_title('TV vs Sales - Regression Line')

*# Regression plot for Radio advertising budget vs Sales*

axes[1].scatter(X2\_test, y\_test, color='blue', label='Actual')

axes[1].plot(X2\_test, y\_pred2, color='red', label='Predicted')

axes[1].set\_xlabel('Radio Budget')

axes[1].set\_ylabel('Sales')

axes[1].legend()

axes[1].set\_title('Radio vs Sales - Regression Line')

*# Regression plot for Newspaper advertising budget vs Sales*

axes[2].scatter(X3\_test, y\_test, color='blue', label='Actual')

axes[2].plot(X3\_test, y\_pred3, color='red', label='Predicted')

axes[2].set\_xlabel('Newspaper Budget')

axes[2].set\_ylabel('Sales')

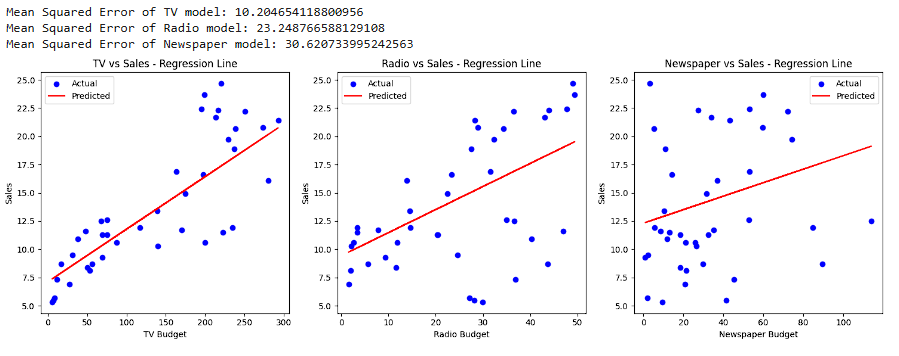
axes[2].legend()

axes[2].set\_title('Newspaper vs Sales - Regression Line')

*# Adjust layout to prevent overlap*

plt.tight\_layout()

plt.show()



1. **A program without using scikit-learn or machine learning for Linear regression**

**A. Using Machine Learning**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def update\_weights(x, y, w, b, lr, m\_w, v\_w, m\_b, v\_b, t, beta1=0.9, beta2=0.999, epsilon=1e-8):

"""

Update weights and bias using the Adam optimizer.

"""

n = len(x)

x = x.flatten()

# Compute gradients

predictions = w \* x + b

d\_w = -2 \* np.sum(x \* (y - predictions))

d\_b = -2 \* np.sum(y - predictions)

# Update moving averages of the gradients

m\_w = beta1 \* m\_w + (1 - beta1) \* d\_w

v\_w = beta2 \* v\_w + (1 - beta2) \* (d\_w \*\* 2)

m\_b = beta1 \* m\_b + (1 - beta1) \* d\_b

v\_b = beta2 \* v\_b + (1 - beta2) \* (d\_b \*\* 2)

# Compute bias-corrected estimates

m\_w\_hat = m\_w / (1 - beta1 \*\* t)

v\_w\_hat = v\_w / (1 - beta2 \*\* t)

m\_b\_hat = m\_b / (1 - beta1 \*\* t)

v\_b\_hat = v\_b / (1 - beta2 \*\* t)

# Update weights and bias

w -= lr \* m\_w\_hat / (np.sqrt(v\_w\_hat) + epsilon)

b -= lr \* m\_b\_hat / (np.sqrt(v\_b\_hat) + epsilon)

return w, b, m\_w, v\_w, m\_b, v\_b

def loss\_fn(x, y, w, b):

"""

Calculate mean squared error loss.

"""

n = len(x)

x = x.flatten()

predictions = w \* x + b

e = np.sum((y - predictions) \*\* 2)

return e / n

def pred(x, w, b):

"""

Prediction function.

"""

return w \* x + b

def training(x, y, w, b, lr, epochs):

"""

Train the model and record loss history.

"""

loss\_history = []

# Initialize Adam optimizer variables

m\_w = 0

v\_w = 0

m\_b = 0

v\_b = 0

for t in range(1, epochs + 1):

w, b, m\_w, v\_w, m\_b, v\_b = update\_weights(x, y, w, b, lr, m\_w, v\_w, m\_b, v\_b, t)

loss = loss\_fn(x, y, w, b)

loss\_history.append(loss)

# if t % 10 == 0:

# print(f"epoch={t} weight={w:.4f} bias={b:.4f} cost={loss:.4f}")

return w, b, loss\_history

# Prepare the data

url = "https://raw.githubusercontent.com/selva86/datasets/master/Advertising.csv"

data = pd.read\_csv(url)

tv = data['TV'].values

radio = data['radio'].values

newspaper = data['newspaper'].values

sales = data['sales'].values

inputs = [tv, radio, newspaper]

titles = ['TV', 'Radio', 'Newspaper']

# Create subplots

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for i, x in enumerate(inputs):

weight = 0

bias = 0

lr = 0.005 # Learning rate

epochs = 150

w, b, loss\_history = training(x, sales, weight, bias, lr, epochs)

# Predict y values

y\_pred = pred(x, w, b)

# Plot

axes[i].scatter(x, sales, color='blue', label='Actual')

axes[i].plot(x, y\_pred, color='red', label='Predicted')

axes[i].set\_xlabel(f'{titles[i]} Budget')

axes[i].set\_ylabel('Sales')

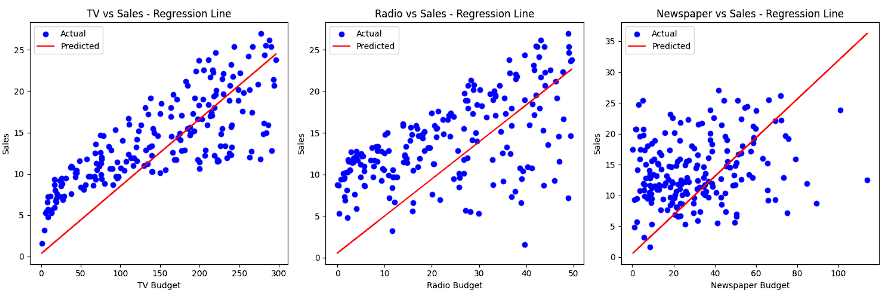
axes[i].legend()

axes[i].set\_title(f'{titles[i]} vs Sales - Regression Line')

# Adjust layout

plt.tight\_layout()

plt.show()



**B. Using Co-efficient of Regression**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def calculate\_coefficients(x, y):

"""

Calculate the slope and intercept for linear regression manually.

"""

*# Flatten x to be a 1D array*

x = x.flatten()

*# Mean of x and y*

x\_mean = np.mean(x)

y\_mean = np.mean(y)

*# Calculating coefficients*

numerator = np.sum((x - x\_mean) \* (y - y\_mean))

denominator = np.sum((x - x\_mean) \*\* 2)

slope = numerator / denominator

intercept = y\_mean - slope \* x\_mean

return slope, intercept

def predict(x, slope, intercept):

"""

Predict y values using the slope and intercept.

"""

return slope \* x + intercept

*# Prepare the data*

url = "https://raw.githubusercontent.com/selva86/datasets/master/Advertising.csv"

data = pd.read\_csv(url)

tv = data['TV'].values

radio = data['radio'].values

newspaper = data['newspaper'].values

sales = data['sales'].values

inputs = [tv, radio, newspaper]

titles = ['TV', 'Radio', 'Newspaper']

*# Create subplots*

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for i, x in enumerate(inputs):

*# Calculate coefficients*

slope, intercept = calculate\_coefficients(x, sales)

*# Predict y values*

y\_pred = predict(x, slope, intercept)

*# Plot*

axes[i].scatter(x, sales, color='blue', label='Actual')

axes[i].plot(x, y\_pred, color='red', label='Predicted')

axes[i].set\_xlabel(f'{titles[i]} Budget')

axes[i].set\_ylabel('Sales')

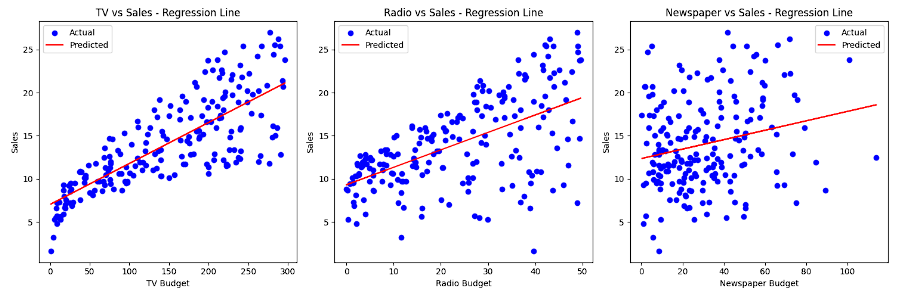
axes[i].legend()

axes[i].set\_title(f'{titles[i]} vs Sales - Regression Line')

*# Adjust layout*

plt.tight\_layout()

plt.show()



1. **What do the Coefficients of Regression signify?**

The coefficients in a linear regression model represent the estimated change in the dependent variable for a one-unit change in the corresponding independent variable, holding all other variables constant. Specifically, in the context of the California housing dataset, each coefficient signifies how much the median house value is expected to change with a one-unit change in a specific feature (e.g., median income or average rooms per dwelling). The intercept represents the predicted value of the dependent variable when all independent variables are zero.

# EXPERIMENT

### AIM: Program to demonstrate Logistic Regression

**CODE:**

*# Import necessary libraries*

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import (

accuracy\_score,

confusion\_matrix,

classification\_report,

roc\_curve,

roc\_auc\_score,

)

*# Load the Advertising Data*

url = "https://raw.githubusercontent.com/selva86/datasets/master/Advertising.csv"

data = pd.read\_csv(url)

*# Let's categorize sales above the median as '1' (High) and below or equal as '0' (Low)*

median\_sales = data['sales'].median()

data['sales\_category'] = np.where(data['sales'] > median\_sales, 1, 0)

*# Features and target variable*

X = data[['TV', 'radio', 'newspaper']]

y = data['sales\_category']

*# Function to perform logistic regression on a single feature*

def logistic\_regression\_analysis(feature\_name):

X\_feature = data[[feature\_name]]

*# Split the data*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_feature, y, test\_size=0.2, random\_state=42

)

*# Define and train the model*

model = LogisticRegression()

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

*# Make predictions*

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

y\_pred\_proba = model.predict\_proba(X\_test)[:, 1]

*# Evaluate the model*

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

cm = confusion\_matrix(y\_test, y\_pred)

cr = classification\_report(y\_test, y\_pred)

auc = roc\_auc\_score(y\_test, y\_pred\_proba)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

*# Print evaluation metrics*

print(f"--- Logistic Regression using {feature\_name} ---")

print(f"Accuracy: {accuracy:.4f}")

print(f"AUC Score: {auc:.4f}")

print("\nConfusion Matrix:")

print(cm)

print("\nClassification Report:")

print(cr)

*# Plot ROC Curve*

plt.figure(figsize=(4, 3))

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.4f})', color='darkorange')

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

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title(f'ROC Curve - {feature\_name}')

plt.legend(loc='lower right')

plt.show()

*# Plot data points and decision boundary*

plt.figure(figsize=(4,3))

sns.scatterplot(x=X\_test[feature\_name], y=y\_test, hue=y\_pred, palette='coolwarm')

plt.xlabel(f'{feature\_name} Budget')

plt.ylabel('Sales Category')

plt.title(f'Logistic Regression Decision Boundary - {feature\_name}')

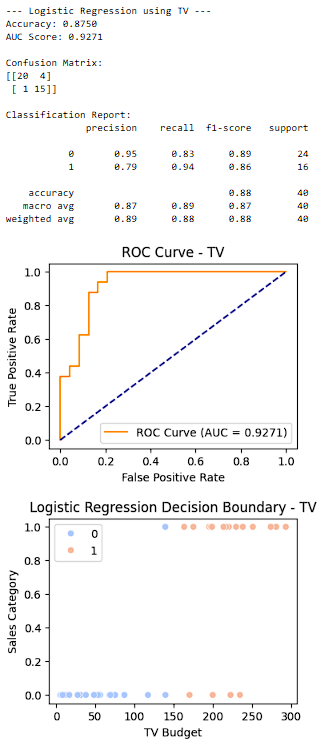
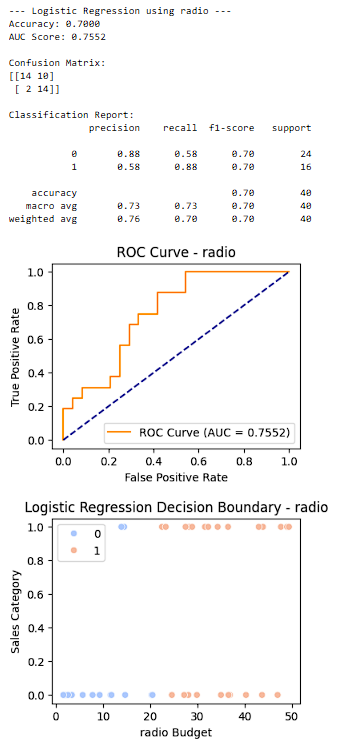
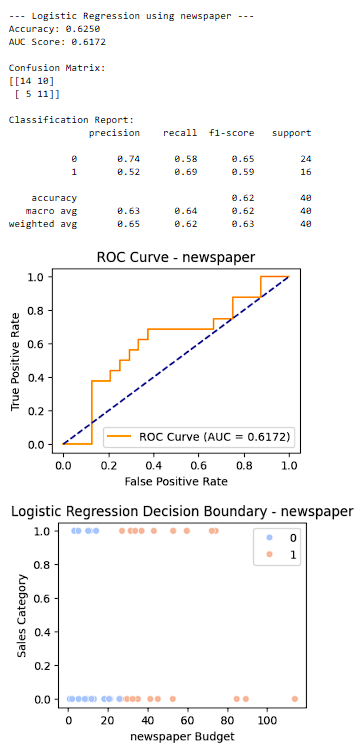
plt.show()

*# Perform logistic regression analysis on each feature*

for feature in ['TV', 'radio', 'newspaper']:

logistic\_regression\_analysis(feature)

**OUTPUT:**

# EXPERIMENT

### AIM: Program to demonstrate Decision Tree – ID3 Algorithm

**CODE:**

**1. Using the DecisionTreeClassifier from sklearn**

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

import matplotlib.pyplot as plt

*# Load the Advertising Data*

url = "https://raw.githubusercontent.com/selva86/datasets/master/Advertising.csv"

data = pd.read\_csv(url)

*# Create a binary target variable (High Sales vs Low Sales)*

median\_sales = data['sales'].median()

data['sales\_label'] = (data['sales'] > median\_sales).astype(int)

*# Define features and target*

X = data[['TV', 'radio', 'newspaper']]

y = data['sales\_label']

*# Split the dataset into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

from sklearn.metrics import accuracy\_score

*# Initialize the DecisionTreeClassifier with Gini index*

clf\_gini = DecisionTreeClassifier(criterion='gini', random\_state=42)

clf\_gini.fit(X\_train, y\_train)

*# Predict on the test set*

y\_pred\_gini = clf\_gini.predict(X\_test)

*# Evaluate accuracy*

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

print(f"Accuracy (Gini Index): {accuracy\_gini:.2f}")

*# Plot the resulting decision tree*

plt.figure(figsize=(15, 10))

plot\_tree(clf\_gini, filled=True, feature\_names=X.columns, class\_names=['Low', 'High'])

plt.title("Decision Tree (Gini Index)")

plt.show()

*# Initialize the DecisionTreeClassifier with Entropy*

clf\_entropy = DecisionTreeClassifier(criterion='entropy', random\_state=42)

clf\_entropy.fit(X\_train, y\_train)

*# Predict on the test set*

y\_pred\_entropy = clf\_entropy.predict(X\_test)

*# Evaluate accuracy*

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

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

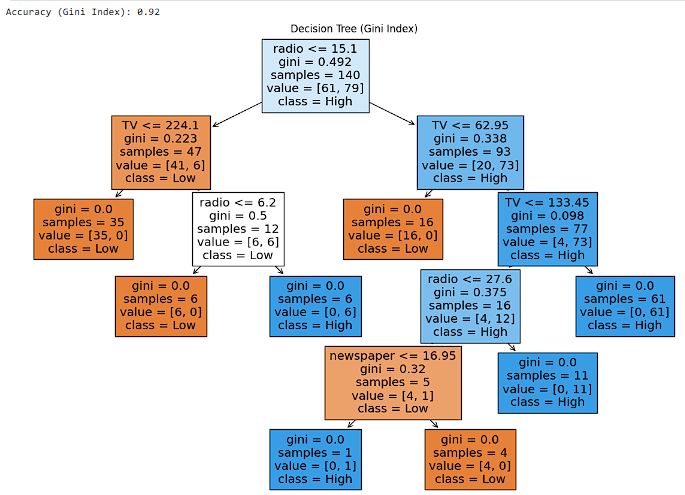
*# Plot the resulting decision tree*

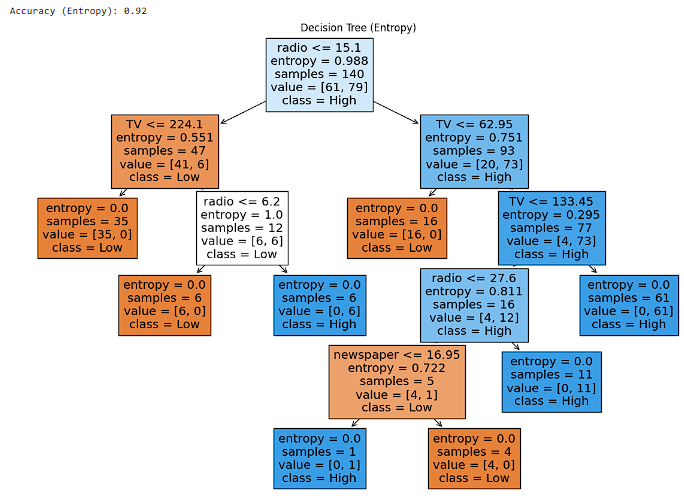
plt.figure(figsize=(15, 10))

plot\_tree(clf\_entropy, filled=True, feature\_names=X.columns, class\_names=['Low', 'High'])

plt.title("Decision Tree (Entropy)")

plt.show()





**2. Creating a Decision Tree Without sklearn**

import pandas as pd

import numpy as np

import math

from collections import Counter

*# Function to calculate entropy*

def calculate\_entropy(y):

class\_counts = Counter(y)

total\_samples = len(y)

entropy = 0.0

for cls in class\_counts:

p = class\_counts[cls] / total\_samples

entropy -= p \* math.log2(p)

return entropy

*# Function to calculate information gain*

def calculate\_information\_gain(X, y, feature):

base\_entropy = calculate\_entropy(y)

values, counts = np.unique(X[feature], return\_counts=True)

weighted\_entropy = 0.0

for i, value in enumerate(values):

subset\_y = y[X[feature] == value]

subset\_entropy = calculate\_entropy(subset\_y)

weighted\_entropy += (counts[i] / sum(counts)) \* subset\_entropy

information\_gain = base\_entropy - weighted\_entropy

return information\_gain

*# Load the Advertising Data*

url = "https://raw.githubusercontent.com/selva86/datasets/master/Advertising.csv"

data = pd.read\_csv(url)

*# Create a binary target variable*

median\_sales = data['sales'].median()

data['sales\_label'] = (data['sales'] > median\_sales).astype(int)

*# Define a small subset of the data*

subset = data[['TV', 'radio', 'newspaper', 'sales\_label']].head(10)

X = subset[['TV', 'radio', 'newspaper']]

y = subset['sales\_label']

*# Calculate information gain for each feature*

info\_gain\_tv = calculate\_information\_gain(X, y, 'TV')

info\_gain\_radio = calculate\_information\_gain(X, y, 'radio')

info\_gain\_newspaper = calculate\_information\_gain(X, y, 'newspaper')

print(f"Information Gain for TV: {info\_gain\_tv}")

print(f"Information Gain for Radio: {info\_gain\_radio}")

print(f"Information Gain for Newspaper: {info\_gain\_newspaper}")

*# Determine the root node*

if info\_gain\_tv > info\_gain\_radio and info\_gain\_tv > info\_gain\_newspaper:

root\_node = 'TV'

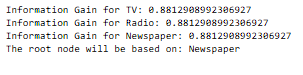
elif info\_gain\_radio > info\_gain\_tv and info\_gain\_radio > info\_gain\_newspaper:

root\_node = 'Radio'

else:

root\_node = 'Newspaper'

print(f"The root node will be based on: {root\_node}")



**3. Decision Trees for Classification and Regression**

from sklearn.tree import DecisionTreeRegressor

*# Define features and target*

X = data[['TV', 'radio', 'newspaper']]

y = data['sales']

*# Split the dataset into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

*# Initialize the DecisionTreeRegressor*

regressor = DecisionTreeRegressor(random\_state=42)

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

*# Predict on the test set*

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

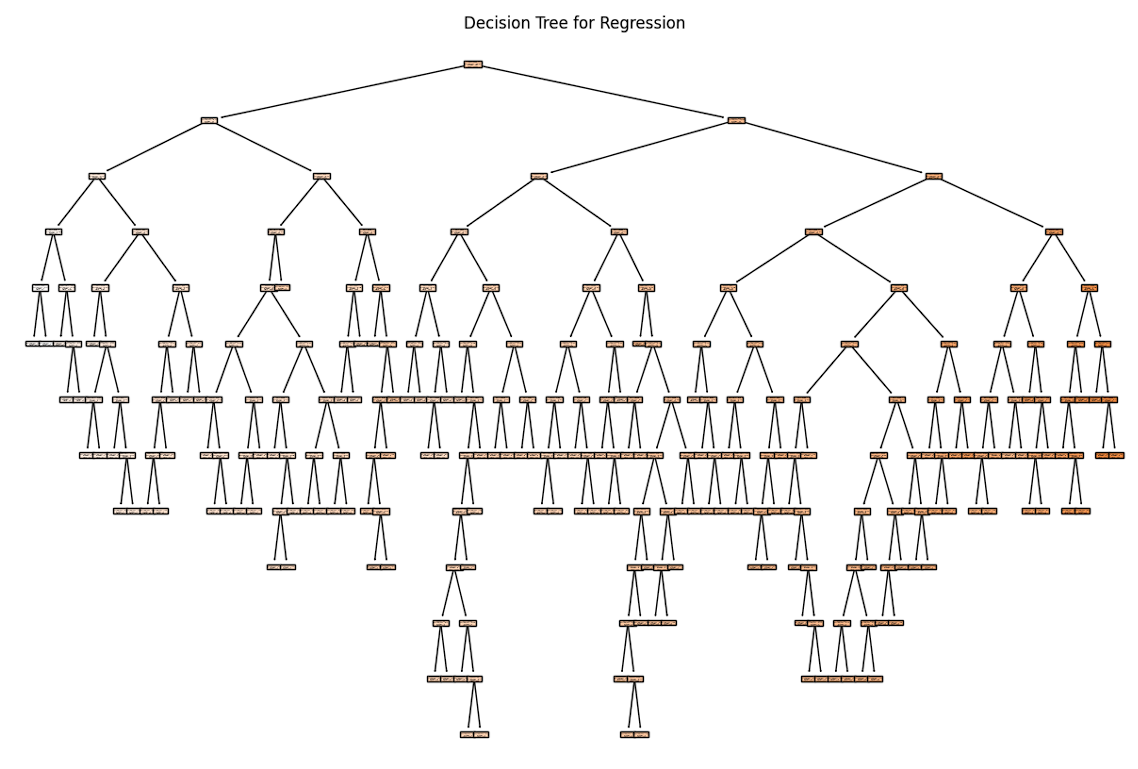
*# Plot the resulting decision tree*

plt.figure(figsize=(15, 10))

plot\_tree(regressor, filled=True, feature\_names=X.columns)

plt.title("Decision Tree for Regression")

plt.show()



# EXPERIMENT

### AIM: Program to demonstrate K-Nearest Neighbor flowers classification

**CODE:**

**1. KNN Classification with Scikit-Learn:**

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

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=75)

k\_values = [5, 8, 9, 15, 26, 45, 78, 95]

score=[]

for k in k\_values:

knn = KNeighborsClassifier(n\_neighbors=k)

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

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

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

score.append(accuracy)

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

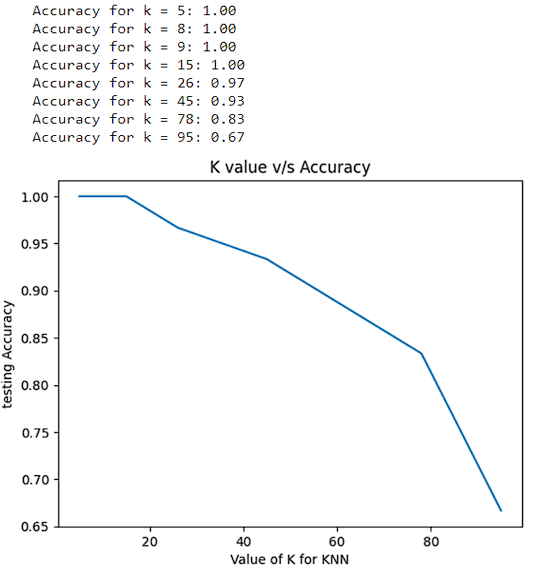
import matplotlib.pyplot as plt

plt.plot(k\_values, score)

plt.title("K value v/s Accuracy")

plt.xlabel('Value of K for KNN')

plt.ylabel('testing Accuracy')



**2. Custom KNN Implementation with Different Distance Metrics:**

import numpy as np

def euclidean\_distance(x1, x2):

return np.sqrt(np.sum((x1 - x2) \*\* 2))

def manhattan\_distance(x1, x2):

return np.sum(np.abs(x1 - x2))

def minkowski\_distance(x1, x2, p):

return np.power(np.sum(np.power(np.abs(x1 - x2), p)), 1 / p)

def chebyshev\_distance(x1, x2):

return np.max(np.abs(x1 - x2))

class CustomKNN:

def \_\_init\_\_(self, n\_neighbors=5, distance\_metric="euclidean"):

self.n\_neighbors = n\_neighbors

self.distance\_metric = distance\_metric

def fit(self, X\_train, y\_train):

self.X\_train = X\_train

self.y\_train = y\_train

def predict(self, X\_test):

y\_pred = []

for x\_test in X\_test:

distances = []

for i in range(len(self.X\_train)):

if self.distance\_metric == "euclidean":

distance = euclidean\_distance(x\_test, self.X\_train[i])

elif self.distance\_metric == "manhattan":

distance = manhattan\_distance(x\_test, self.X\_train[i])

elif self.distance\_metric == "minkowski":

distance = minkowski\_distance(x\_test, self.X\_train[i], p=3)

elif self.distance\_metric == "chebyshev":

distance = chebyshev\_distance(x\_test, self.X\_train[i])

distances.append((distance, i))

distances.sort()

neighbors = [self.y\_train[i] for \_, i in distances[:self.n\_neighbors]]

y\_pred.append(max(set(neighbors), key=list(neighbors).count))

return y\_pred

*# Use the custom KNN implementation with different distance metrics*

knn\_euclidean = CustomKNN(5, distance\_metric="euclidean")

knn\_manhattan = CustomKNN(5, distance\_metric="manhattan")

knn\_chebyshev = CustomKNN(5, distance\_metric="chebyshev")

knn\_minkowski = CustomKNN(5, distance\_metric="minkowski")

*# Train and evaluate the models*

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

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

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

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

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

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

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

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

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

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

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

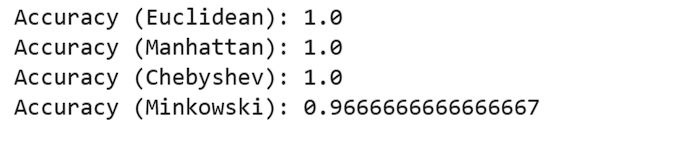
accuracy\_minkowski = accuracy\_score(y\_test, y\_pred\_minkowski)

print("Accuracy (Euclidean):", accuracy\_euclidean)

print("Accuracy (Manhattan):", accuracy\_manhattan)

print("Accuracy (Chebyshev):", accuracy\_chebyshev)

print("Accuracy (Minkowski):", accuracy\_minkowski)



**3. Applying Custom KNN to Pima Indians Diabetes Database:**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

*# Load the dataset*

data = pd.read\_csv("diabetes.csv")

*# Handle missing values (if necessary)*

*# data = data.fillna(data.mean())*

*# Standardize features*

scaler = StandardScaler()

X = scaler.fit\_transform(data.drop("Outcome", axis=1))

y = data["Outcome"]

*# 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=47)

*# Create a KNN classifier with optimized parameters*

knn = KNeighborsClassifier(n\_neighbors=5, algorithm='ball\_tree') *# Use ball tree for efficiency*

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

*# Make predictions and evaluate the model*

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

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

print("Accuracy:", accuracy)

# 

# EXPERIMENT

### AIM: Program to demonstrate Naïve‐ Bayes Classifier.

**CODE:**

from sklearn.datasets import make\_classification

X, y = make\_classification(

n\_features=6,

n\_classes=3,

n\_samples=800,

n\_informative=2,

random\_state=1,

n\_clusters\_per\_class=1,

)

import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1], c=y, marker="\*")

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size=0.33, random\_state=125

)

from sklearn.naive\_bayes import GaussianNB

*# Build a Gaussian Classifier*

model = GaussianNB()

*# Model training*

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

*# Predict Output*

predicted = model.predict([X\_test[6]])

print("Actual Value:", y\_test[6])

print("Predicted Value:", predicted[0])

from sklearn.metrics import (

accuracy\_score,

confusion\_matrix,

ConfusionMatrixDisplay,

f1\_score,

)

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

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

f1 = f1\_score(y\_pred, y\_test, average="weighted")

print("Accuracy:", accuray)

print("F1 Score:", f1)

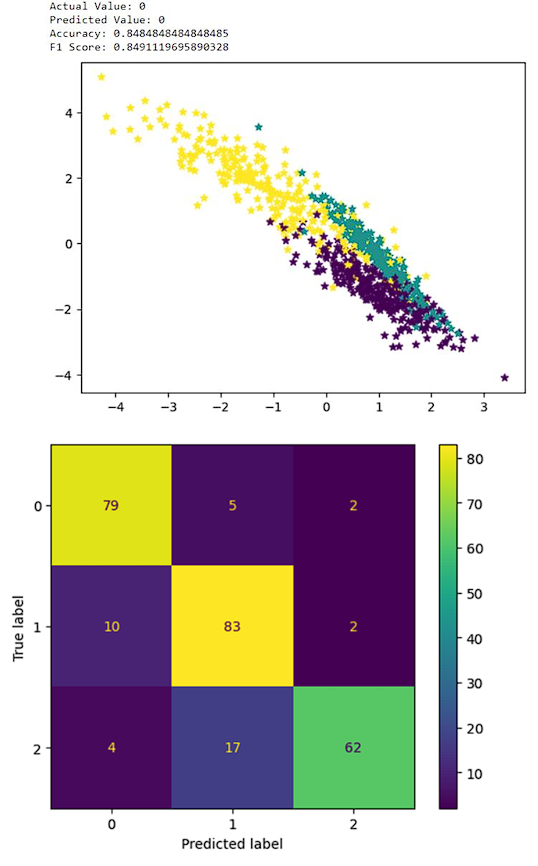
labels = [0,1,2]

cm = confusion\_matrix(y\_test, y\_pred, labels=labels)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=labels)

disp.plot();

**OUTPUT:**

****

# EXPERIMENT

### AIM: Program to demonstrate PCA and LDA on Iris dataset.

**CODE:**

**1. Apply the PCA and LDA techniques on the IRIS dataset from sklearn - are both applicable?**

# Importing Necessary Liberaries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

# Load the IRIS dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Standardize the features

scaler = StandardScaler()

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

# Apply PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Apply LDA

lda = LinearDiscriminantAnalysis(n\_components=2)

X\_lda = lda.fit\_transform(X\_scaled, y)

*# Plot PCA results*

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

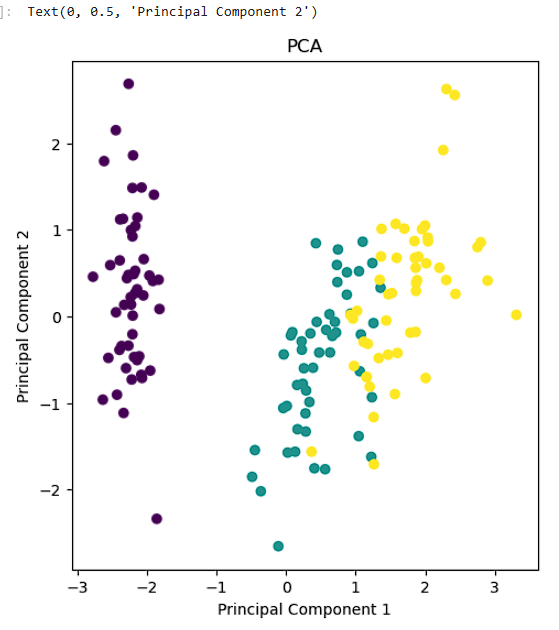
plt**.**subplot(1, 2, 1)

plt**.**scatter(X\_pca[:, 0], X\_pca[:, 1], c**=**y, cmap**=**'viridis')

plt**.**title('PCA')

plt**.**xlabel('Principal Component 1')

plt**.**ylabel('Principal Component 2')



*# Plot LDA results*

plt**.**subplot(1, 2, 2)

plt**.**scatter(X\_lda[:, 0], X\_lda[:, 1], c**=**y, cmap**=**'viridis')

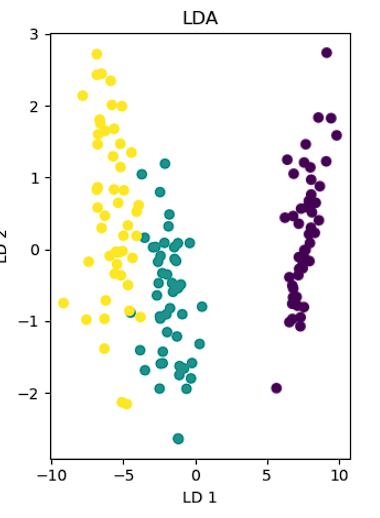
plt**.**title('LDA')

plt**.**xlabel('LD 1')

plt**.**ylabel('LD 2')

plt**.**tight\_layout()

plt**.**show()



Yes, both Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are applicable to the IRIS dataset from sklearn. Since the IRIS dataset includes both features and class labels, we can apply both PCA and LDA techniques to it.

In summary, both PCA and LDA can be applied to the IRIS dataset, but the choice between them depends on the specific goals of analysis. If the goal is purely dimensionality reduction or visualization, PCA might suffice. If the goal is classification or maximizing class separability, LDA could be more appropriate.

**2. Explain the difference between two approaches.**

PCA (Principal Component Analysis) and LDA (Linear Discriminant Analysis) are both dimensionality reduction techniques, but they have different underlying principles and objectives:

1. **PCA (Principal Component Analysis):**
   * Objective: PCA aims to find the orthogonal axes (principal components) that maximize the variance in the data. It doesn't consider class labels.
   * Unsupervised: PCA is an unsupervised technique, meaning it doesn't use class information during the dimensionality reduction process.
   * Maximizes variance: PCA seeks to capture the directions of maximum variance in the data, making it useful for reducing the dimensionality of the feature space while retaining as much variance as possible.
   * Applicability: PCA is widely used for data visualization, noise reduction, and general-purpose dimensionality reduction when the main goal is to represent the data in a lower-dimensional space without specific consideration of class boundaries.
2. **LDA (Linear Discriminant Analysis):**
   * Objective: LDA aims to find the linear combinations of features that best separate different classes in the data. It explicitly considers class labels.
   * Supervised: LDA is a supervised technique because it uses class labels to maximize class separability.
   * Maximizes class separability: LDA seeks to project the data onto a lower-dimensional space while maximizing the separation between different classes. It considers both within-class and between-class scatter.
   * Applicability: LDA is particularly useful when the goal is classification or when the main interest lies in maximizing the separability between different classes in the reduced-dimensional space. It can help improve the performance of classifiers by providing more discriminative features.

**3. Using Examples From the Colab Notebook and applying PCA and LDA with a brief explanation of the Results.**

**3.1 PCA on Iris Dataset**

import numpy as np

from sklearn import datasets

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data[:, :2]

y = iris.target

# Center the data

X\_meaned = X - np.mean(X, axis=0)

# Calculate the covariance matrix and eigendecomposition

cov\_mat = np.cov(X\_meaned, rowvar=False)

eig\_vals, eig\_vecs = np.linalg.eig(cov\_mat)

# Sort components by explained variance (eigenvalues)

idx = np.argsort(eig\_vals)[::-1]

eig\_vecs = eig\_vecs[:, idx]

eig\_vals = eig\_vals[idx]

# Select top 2 components (capturing most of the variance)

W = eig\_vecs[:, :2]

# Project the data onto the new space

X\_reduced = np.dot(X\_meaned, W)

# Plot the results

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

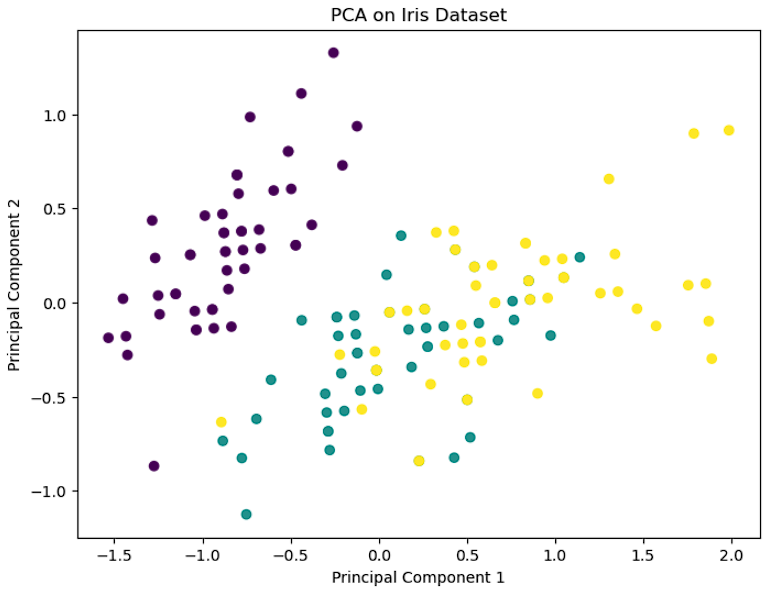
plt.scatter(X\_reduced[:, 0], X\_reduced[:, 1], c=y, cmap='viridis')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('PCA on Iris Dataset')

plt.show()



**Explanation**

* In the resulting scatter plot, we observe clusters of data points representing different species of iris flowers.
* The separation between clusters suggests that the first two principal components effectively capture much of the variability in the dataset.
* Since PCA is an unsupervised technique, the clusters are based solely on patterns of variance in the data and do not necessarily correspond to the true class labels.
* However, we can see some degree of separation between the clusters, indicating that the selected principal components are informative in distinguishing between different species of iris flowers based on their sepal measurements.

**3.2 PCA with Eigenvectors**

import numpy as np

from sklearn import datasets

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data[:, :2]

y = iris.target

# Center the data

X\_meaned = X - np.mean(X, axis=0)

# Calculate the covariance matrix and eigendecomposition

cov\_mat = np.cov(X\_meaned, rowvar=False)

eig\_vals, eig\_vecs = np.linalg.eig(cov\_mat)

# Sort components by explained variance (eigenvalues)

idx = np.argsort(eig\_vals)[::-1]

eig\_vecs = eig\_vecs[:, idx]

eig\_vals = eig\_vals[idx]

# Select top 2 components (capturing most of the variance)

W = eig\_vecs[:, :2]

# Project the data onto the new space

X\_reduced = np.dot(X\_meaned, W)

# Define scaling factors for visualizing eigenvectors (adjust as needed)

scale1 = 3

scale2 = 2

# Plot the results

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

plt.scatter(X\_reduced[:, 0], X\_reduced[:, 1], c=y, cmap='viridis', alpha=0.7, label='Data')

# Plot the eigenvectors with arrows

plt.arrow(0, 0, scale1\*eig\_vecs[:, 0][0], scale1\*eig\_vecs[:, 0][1], color='red', linewidth=2, label='PC1')

plt.arrow(0, 0, scale2\*eig\_vecs[:, 1][0], scale2\*eig\_vecs[:, 1][1], color='blue', linewidth=2, label='PC2')

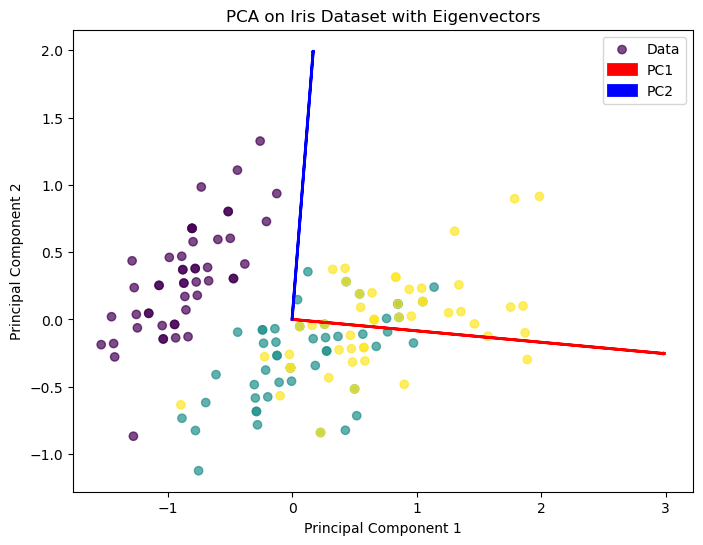
plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('PCA on Iris Dataset with Eigenvectors')

plt.legend()

plt.show()



**Interpretation:**

* In the resulting scatter plot, we observe clusters of data points representing different species of iris flowers, similar to the previous visualization.
* Additionally, we now see two arrows (red and blue) representing the principal components (PC1 and PC2) in the same plot.
* The direction of each arrow indicates the orientation of the corresponding principal component in the original feature space.
* The length of each arrow represents the magnitude or importance of the corresponding principal component.
* PC1 and PC2 capture the directions of maximum variance in the data, and their lengths indicate how much of the total variance they explain.
* The visualization of eigenvectors helps us understand how the principal components are aligned with the original features and how they contribute to the variability in the dataset.

**3.3 PCA on Height and Hand Span**

import numpy as np

# Define the mean and standard deviation for height and hand span

mean\_height = 170

std\_height = 10

mean\_span = 18

std\_span = 8

# Generate correlated data using a linear relationship with some noise

covariance = 0.8

np.random.seed(10)

height = np.random.normal(mean\_height, std\_height, 100)

noise = np.random.normal(0, 1, 100)

hand\_span = covariance \* height + mean\_span + std\_span \* noise

In [4]:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

# Define parameters

num\_people = 100

mean\_height = 170

std\_height = 10

mean\_span = 18

std\_span = 8

covariance = 0.8

# Generate correlated data

np.random.seed(10)

height = np.random.normal(mean\_height, std\_height, num\_people)

noise = np.random.normal(0, 1, num\_people)

hand\_span = covariance \* height + mean\_span + std\_span \* noise

# Create scatter plot

plt.scatter(height, hand\_span)

plt.xlabel('Height')

plt.ylabel('Hand Span')

plt.title('Original Data (Height vs. Hand Span)')

plt.show()

# Apply PCA for dimensionality reduction

pca = PCA(n\_components=1)

pca.fit(np.c\_[height, hand\_span])

# Transformed data

data\_transformed = pca.transform(np.c\_[height, hand\_span])

# Print original and transformed feature matrices

print("Original feature matrix:\n", np.c\_[height, hand\_span])

print("\nTransformed feature matrix:\n", data\_transformed)

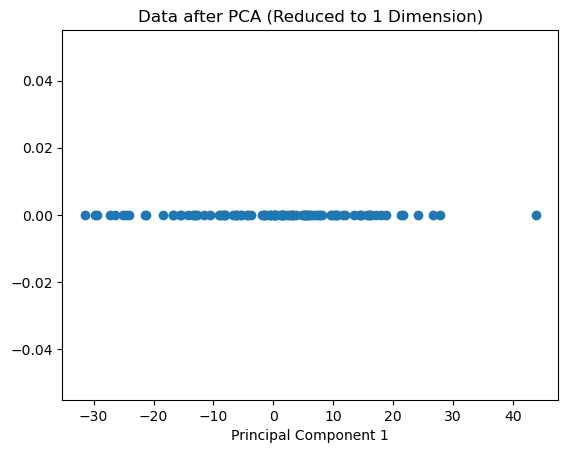
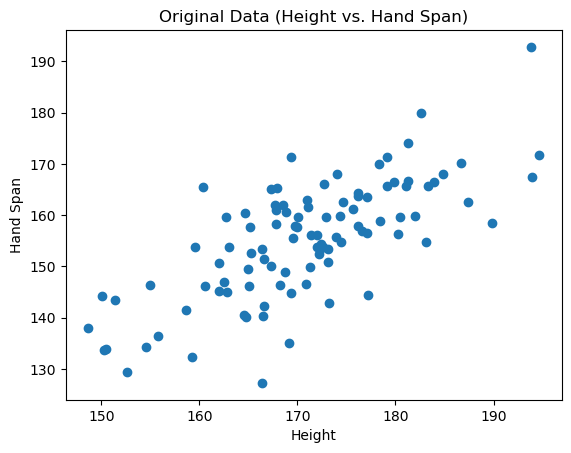
# Visualize transformed data (single component)

plt.scatter(data\_transformed, np.zeros\_like(data\_transformed))

plt.xlabel('Principal Component 1')

plt.title('Data after PCA (Reduced to 1 Dimension)')

plt.show()



**Interpretation:**

The provided code generates synthetic data for a hypothetical scenario where there's a correlation between height and hand span measurements. Then, Principal Component Analysis (PCA) is applied for dimensionality reduction to visualize the data in a lower-dimensional space.

* In the scatter plot of the original data, we observe a trend where hand span tends to increase with height, reflecting the underlying linear relationship.
* After applying PCA, the transformed data is visualized along the single principal component.
* The spread of points along the principal component captures the variability in the original data, with individuals positioned based on their overall height-hand span relationship.
* By reducing the data to a single dimension, PCA effectively summarizes the variability in the original data while retaining the essential patterns.

**3.4 LDA & PCA (Reducing to 2 Dimensions)**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Apply LDA (reducing to 2 dimensions for visualization)

lda = LinearDiscriminantAnalysis(n\_components=2)

X\_lda = lda.fit\_transform(X, y)

# Project data using PCA for visualization comparison (optional)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

# Plot the data with class labels

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

markers = ['s', 'x', 'o']

for i, target\_name in enumerate(iris.target\_names):

plt.scatter(X\_lda[:, 0][y == i], X\_lda[:, 1][y == i], marker=markers[i], label=target\_name)

plt.xlabel('LD1')

plt.ylabel('LD2')

plt.title('Iris Dataset with LDA (Reduced to 2 Dimensions)')

plt.legend()

plt.show()

# Optional: Plot with PCA for comparison

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

for i, target\_name in enumerate(iris.target\_names):

plt.scatter(X\_pca[:, 0][y == i], X\_pca[:, 1][y == i], marker=markers[i], label=target\_name)

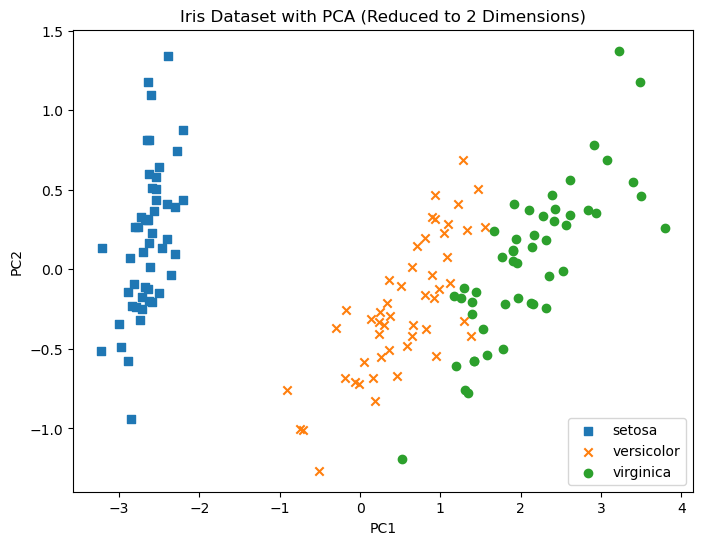
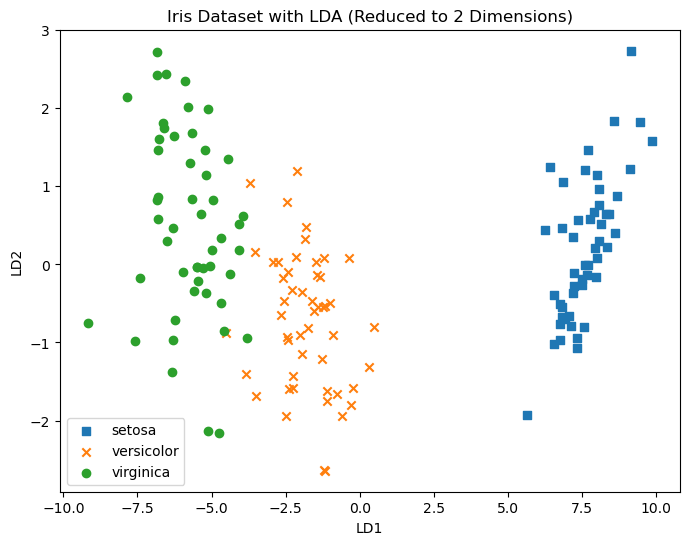
plt.xlabel('PC1')

plt.ylabel('PC2')

plt.title('Iris Dataset with PCA (Reduced to 2 Dimensions)')

plt.legend()

plt.show()



**Interpretation:**

The provided code performs dimensionality reduction on the Iris dataset using Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA), both reducing the data to two dimensions for visualization purposes.

* The LDA visualization demonstrates that the two linear discriminants effectively separate the three classes of iris flowers based on the provided features.
* In contrast, the PCA visualization captures the overall variance in the dataset but doesn't explicitly optimize for class separation.
* LDA is more suitable when the goal is to maximize class separability, as seen in this case with the clear separation between classes in the LDA plot.
* PCA, while useful for capturing overall variance and reducing dimensionality, might not always be optimal for class-based tasks like classification or discrimination.

# EXPERIMENT

### AIM: Program to demonstrate DBSCAN clustering algorithm

**CODE:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.metrics import silhouette\_score

from sklearn.cluster import DBSCAN

iris = datasets.load\_iris()

X = iris.data

eps = 0.7

min\_samples = 10

metric = 'euclidean'

dbscan = DBSCAN(eps=eps, min\_samples=min\_samples, metric=metric)

dbscan.fit(X)

labels = dbscan.labels\_

core\_samples\_mask = np.zeros\_like(labels, dtype=bool)

core\_samples\_mask[dbscan.core\_sample\_indices\_] = True

n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)

n\_noise\_ = list(labels).count(-1)

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

unique\_labels = set(labels)

colors = [plt.cm.Spectral(each)

for each in np.linspace(0, 1, len(unique\_labels))]

for k, col in zip(unique\_labels, colors):

if k == -1:

col = [0, 0, 0, 1]

class\_member\_mask = (labels == k)

xy = X[class\_member\_mask & core\_samples\_mask]

plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),

markeredgecolor='k', markersize=14)

xy = X[class\_member\_mask & ~core\_samples\_mask]

plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),

markeredgecolor='k', markersize=6)

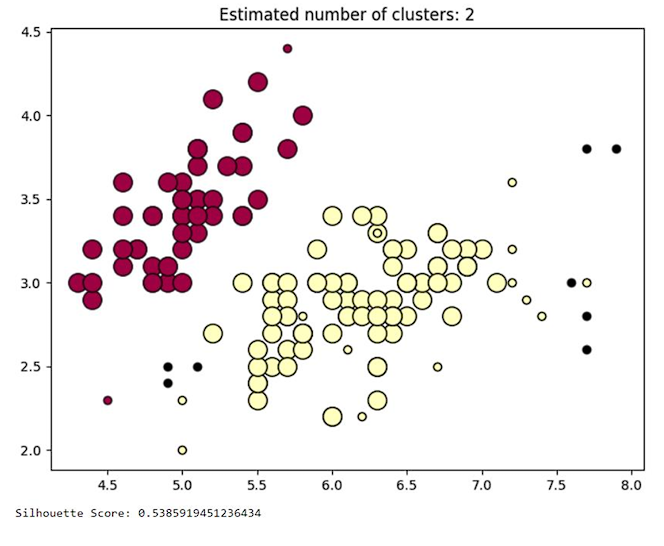
plt.title('Estimated number of clusters: %d' % n\_clusters\_)

plt.show()

silhouette\_avg = silhouette\_score(X, labels)

print("Silhouette Score:", silhouette\_avg)

**OUTPUT:**

****

# EXPERIMENT

### AIM: Program to demonstrate K-Medoid clustering algorithm

**CODE:**

import numpy as np

from sklearn.metrics.pairwise import pairwise\_distances\_argmin

from sklearn.datasets import make\_blobs

import matplotlib.pyplot as plt

X, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60,random\_state=0)

n\_clusters = 4

max\_iter = 300

n\_samples = X.shape[0]

cluster\_centers\_idx = np.random.choice(n\_samples, n\_clusters,replace=False)

cluster\_centers = X[cluster\_centers\_idx]

for \_ in range(max\_iter):

closest\_medoids = pairwise\_distances\_argmin(X, cluster\_centers)

for i in range(n\_clusters):

cluster\_points = X[closest\_medoids == i]

new\_medoid =cluster\_points[np.argmin(pairwise\_distances\_argmin(cluster\_points, cluster\_points))]

cluster\_centers[i] = new\_medoid

labels = pairwise\_distances\_argmin(X, cluster\_centers)

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')

plt.scatter(cluster\_centers[:, 0], cluster\_centers[:, 1], marker='^', c='red', s=100, label='Medoids')

plt.title('K-Medoids Clustering')

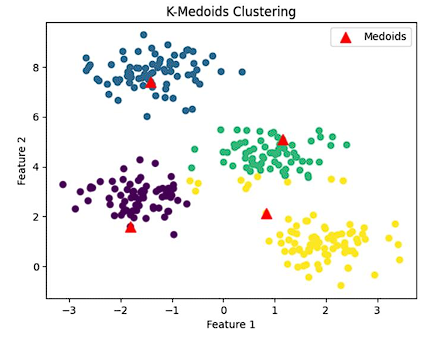
plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()

**OUTPUT:**



# EXPERIMENT

### AIM: Program to demonstrate K-Means Clustering Algorithm on Handwritten Dataset.

**CODE:**

import numpy as np

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.datasets import load\_digits

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import mode

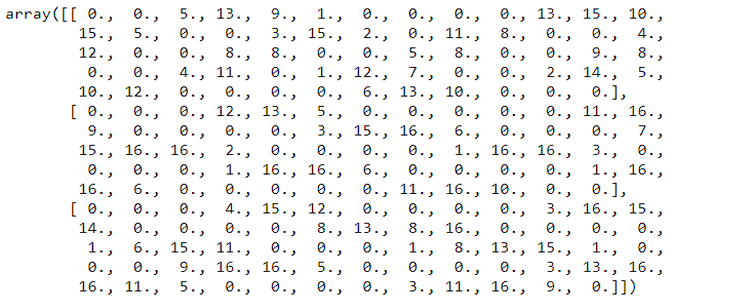
from sklearn.metrics import accuracy\_score, confusion\_matrix

digits = load\_digits()

digits.data.shape

digits.keys()

digits.data[0:3]



k\_means = KMeans(n\_clusters=10,random\_state=0)

clusters = k\_means.fit\_predict(digits.data)

k\_means.cluster\_centers\_.shape

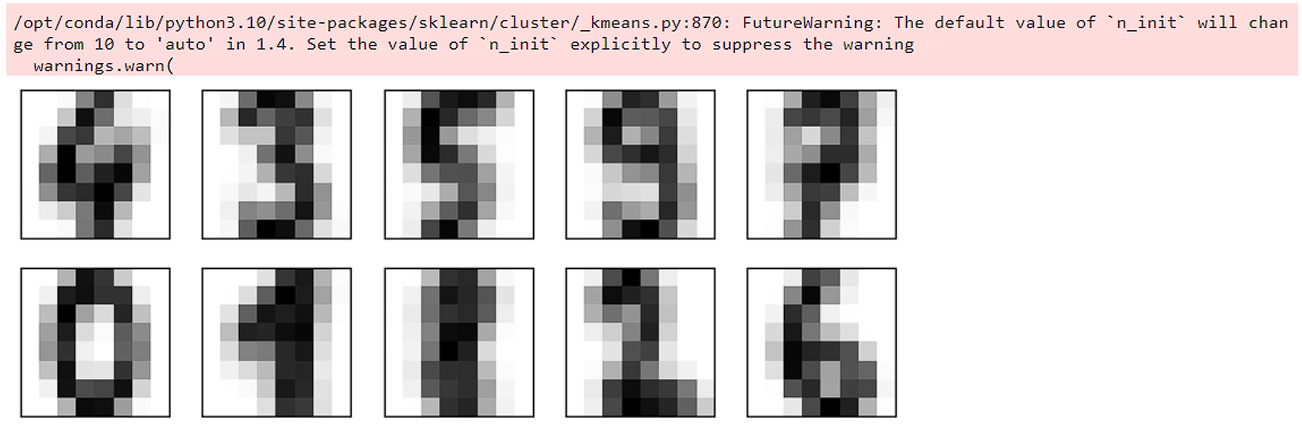
fig, ax = plt.subplots(2, 5, figsize = (8,3))

centers = k\_means.cluster\_centers\_.reshape(10,8,8)

for axi, center in zip(ax.flat, centers):

axi.set(xticks = [], yticks = [])

axi.imshow(center, interpolation='nearest',cmap = plt.cm.binary)



labels = np.zeros\_like(clusters)

print(f"The labels are : {labels}")

print(f"\nThe size of labels is : {labels.shape}")

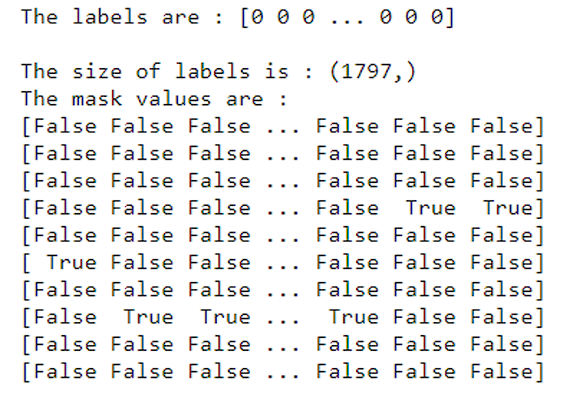
print("The mask values are : ")

for i in range(10):

mask = (clusters == i)

print(mask)

labels[mask] = mode(digits.target[mask])[0]



accuracy = "{:.2f}".format(accuracy\_score(digits.target, labels)\*100)

print(f"Accuracy: {accuracy}%")

matrix = confusion\_matrix(digits.target, labels)

sns.heatmap(matrix.T, square = True, annot = True,

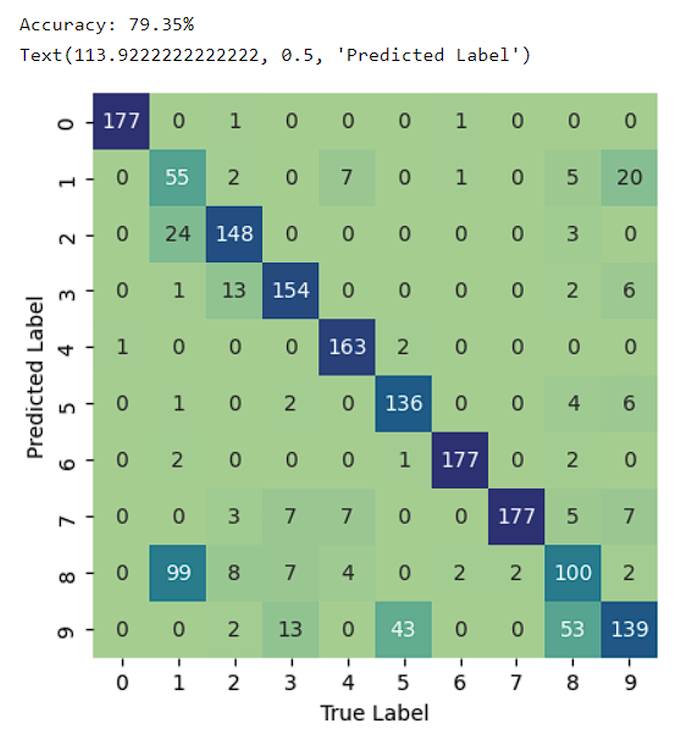
fmt = 'd', cbar=False, cmap="crest",

xticklabels=digits.target\_names,

yticklabels=digits.target\_names)

plt.xlabel('True Label')

plt.ylabel("Predicted Label")



# EXPERIMENT

### AIM: Program to demonstrate SVM on IRIS data set for classification.

**CODE:**

**1. Apply SVM on IRIS data set from sklearn for classification.:**

from sklearn.datasets import load\_iris

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

import numpy as np

import matplotlib.pyplot as plt

iris = load\_iris()

X = iris.data[:, :2]

y = iris.target

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')

svm\_classifier.fit(X\_train, y\_train)

y\_pred = svm\_classifier.predict(X\_test)

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

print("Accuracy:", accuracy)

*# Plot decision boundaries*

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02), np.arange(y\_min, y\_max, 0.02))

Z = svm\_classifier.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

*# Plot decision boundaries and data points*

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

plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')

plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap='coolwarm', edgecolors='k',

label='Training Data')

plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap='coolwarm', marker='x', s=80, label='Test Data')

plt.xlabel('Feature 1')

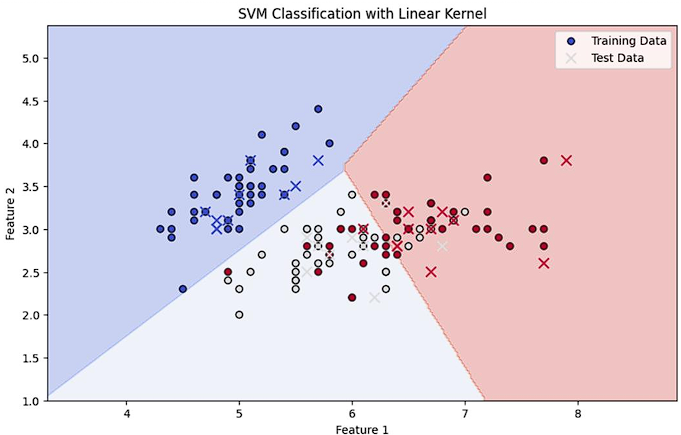
plt.ylabel('Feature 2')

plt.title('SVM Classification with Linear Kernel')

plt.legend()

plt.show()





**2. Visualize data classification with different kernels**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

from sklearn.svm import SVC

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

n\_features = 2

n\_informative = 2

n\_redundant = 0

n\_repeated = 0

X, y = make\_classification(n\_samples=100, n\_features=n\_features,

n\_informative=n\_informative, n\_redundant=n\_redundant, n\_repeated=n\_repeated, n\_classes=2, n\_clusters\_per\_class=1, random\_state=42)

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

svm\_linear = SVC(kernel='linear')

svm\_poly = SVC(kernel='poly')

svm\_rbf = SVC(kernel='rbf')

svm\_linear.fit(X\_train, y\_train)

svm\_poly.fit(X\_train, y\_train)

svm\_rbf.fit(X\_train, y\_train)

h = 0.02

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

plt.figure(figsize=(15, 5))

Z\_linear = svm\_linear.predict(np.c\_[xx.ravel(), yy.ravel()])

Z\_linear = Z\_linear.reshape(xx.shape)

plt.subplot(1, 3, 1)

plt.contourf(xx, yy, Z\_linear, alpha=0.3, cmap='coolwarm')

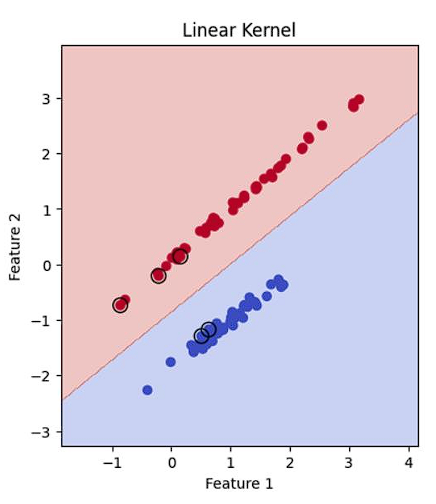
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')

plt.scatter(svm\_linear.support\_vectors\_[:, 0], svm\_linear.support\_vectors\_[:, 1], s=100, facecolors='none', edgecolors='k')

plt.title('Linear Kernel')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')



Z\_poly = svm\_poly.predict(np.c\_[xx.ravel(), yy.ravel()])

Z\_poly = Z\_poly.reshape(xx.shape)

plt.subplot(1, 3, 2)

plt.contourf(xx, yy, Z\_poly, alpha=0.3, cmap='coolwarm')

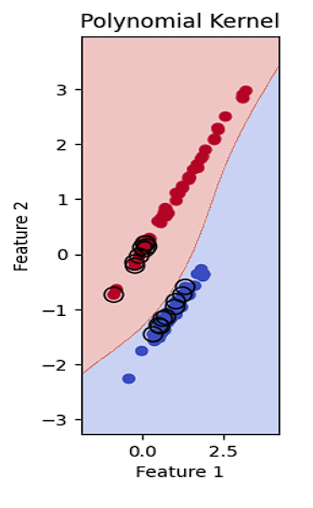
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')

plt.scatter(svm\_poly.support\_vectors\_[:, 0], svm\_poly.support\_vectors\_[:, 1], s=100, facecolors='none', edgecolors='k')

plt.title('Polynomial Kernel')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')



Z\_rbf = svm\_rbf.predict(np.c\_[xx.ravel(), yy.ravel()])

Z\_rbf = Z\_rbf.reshape(xx.shape)

plt.subplot(1, 3, 3)

plt.contourf(xx, yy, Z\_rbf, alpha=0.3, cmap='coolwarm')

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm')

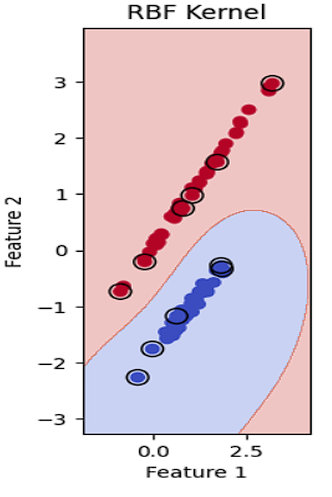
plt.scatter(svm\_rbf.support\_vectors\_[:, 0], svm\_rbf.support\_vectors\_[:, 1], s=100,

facecolors='none', edgecolors='k')

plt.title('RBF Kernel')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')



**3. Explain the different types of kernels, and also explain the process to choose the appropriate kernel for a dataset classification**

**Linear Kernel:**

* Simplest kernel, suitable for linearly separable data.
* Computes the dot product between data points.

**Polynomial Kernel:**

* Introduces a polynomial transformation to the data, allowing for nonlinear decision boundaries.
* Parameters: degree (polynomial order), coef0 (constant term).

**Radial Basis Function (RBF) Kernel:**

* Most commonly used kernel, suitable for nonlinearly separable data. Computes a Gaussian radial basis function centered at each data point.
* Parameter: gamma (inverse bandwidth).

**Sigmoid Kernel:**

* Similar to the logistic function, often used in neural networks.
* Parameters: coef0 (constant term), gamma (scale).

**Choosing the Appropriate Kernel:**

1. Linearly Separable Data: Use the linear kernel for simplicity and efficiency.
2. Nonlinearly Separable Data: Experiment with polynomial, RBF, and sigmoid kernels to find the best fit.
3. Consider Computational Cost: The polynomial and RBF kernels can be computationally expensive for large datasets.
4. Start with RBF: RBF is often a good starting point due to its flexibility.
5. Tune Hyperparameters: Use techniques like grid search or random search to find optimal hyperparameter values for each kernel.
6. Cross-Validation: Evaluate the performance of different kernels using cross-validation to avoid overfitting.