**Program 01**

**1.Program on data wrangling: Combining and merging datasets, Reshaping and Pivoting**

**Source code:**

# Import necessary libraries

import pandas as pd

importnumpy as np

# 1. Combining and Merging Datasets

# Create two sample DataFrames

sales\_data\_1 = pd.DataFrame({

'OrderID': [1, 2, 3, 4],

'Product': ['Laptop', 'Tablet', 'Smartphone', 'Headphones'],

'Sales': [1200, 800, 1500, 300]

})

sales\_data\_2 = pd.DataFrame({

'OrderID': [3, 4, 5, 6],

'Product': ['Smartphone', 'Headphones', 'Smartwatch', 'Tablet'],

'Sales': [1500, 300, 200, 900]

})

# Display the DataFrames

print("Sales Data 1:\n", sales\_data\_1)

print("\nSales Data 2:\n", sales\_data\_2)

# Merge DataFrames based on 'OrderID' using an inner join

merged\_data = pd.merge(sales\_data\_1, sales\_data\_2, on='OrderID', how='inner', suffixes=('\_left', '\_right'))

print("\nMerged Data (Inner Join):\n", merged\_data)

# Concatenate the DataFrames vertically

combined\_data = pd.concat([sales\_data\_1, sales\_data\_2], ignore\_index=True)

print("\nCombined Data (Concatenated Vertically):\n", combined\_data)

# 2. Reshaping Data with Melt

# Create a sample DataFrame for reshaping

reshaping\_data = pd.DataFrame({

'Month': ['Jan', 'Feb', 'Mar'],

'Product\_A': [100, 150, 130],

'Product\_B': [90, 80, 120]

})

print("\nReshaping Data (Original):\n", reshaping\_data)

# Melt the DataFrame to reshape it from wide to long format

melted\_data = pd.melt(reshaping\_data, id\_vars=['Month'], var\_name='Product', value\_name='Sales')

print("\nMelted Data (Long Format):\n", melted\_data)

# 3. Pivoting Data

# Create a sample DataFrame for pivoting

pivot\_data = pd.DataFrame({

'Month': ['Jan', 'Jan', 'Feb', 'Feb', 'Mar', 'Mar'],

'Product': ['Product\_A', 'Product\_B', 'Product\_A', 'Product\_B', 'Product\_A', 'Product\_B'],

'Sales': [100, 90, 150, 80, 130, 120]

})

print("\nPivot Data (Original):\n", pivot\_data)

# Pivot the DataFrame to reshape it back to wide format

pivoted\_data = pivot\_data.pivot(index='Month', columns='Product', values='Sales')

print("\nPivoted Data (Wide Format):\n", pivoted\_data)

# 4. Handling Missing Data

# Introduce some missing values

pivoted\_data.loc['Feb', 'Product\_A'] = np.nan

pivoted\_data.loc['Mar', 'Product\_B'] = np.nan

print("\nPivoted Data with Missing Values:\n", pivoted\_data)

# Fill missing values with the mean of each column

filled\_data = pivoted\_data.fillna(pivoted\_data.mean())

print("\nFilled Data (Missing Values Handled):\n", filled\_data)

# 5. Summary Statistics

print("\nSummary Statistics of Filled Data:\n", filled\_data.describe())

**Sample output:**

Sales Data 1:

OrderIDProduct Sales

0 1 Laptop 1200

1 2 Tablet 800

2 3 Smartphone 1500

3 4 Headphones 300

Sales Data 2:

OrderIDProduct Sales

0 3 Smartphone 1500

1 4 Headphones 300

2 5 Smartwatch 200

3 6 Tablet 900

Merged Data (Inner Join):

OrderIDProduct\_leftSales\_leftProduct\_rightSales\_right

0 3 Smartphone 1500 Smartphone 1500

1 4 Headphones 300 Headphones 300

Combined Data (Concatenated Vertically):

OrderIDProduct Sales

0 1 Laptop 1200

1 2 Tablet 800

2 3 Smartphone 1500

3 4 Headphones 300

4 3 Smartphone 1500

5 4 Headphones 300

6 5 Smartwatch 200

7 6 Tablet 900

Reshaping Data (Original):

Month Product\_AProduct\_B

0 Jan 100 90

1 Feb 150 80

2 Mar 130 120

Melted Data (Long Format):

Month Product Sales

0 Jan Product\_A 100

1 Feb Product\_A 150

2 Mar Product\_A 130

3 Jan Product\_B 90

4 Feb Product\_B 80

5 Mar Product\_B 120

Pivot Data (Original):

Month Product Sales

0 Jan Product\_A 100

1 Jan Product\_B 90

2 Feb Product\_A 150

3 Feb Product\_B 80

4 Mar Product\_A 130

5 Mar Product\_B 120

Pivoted Data (Wide Format):

Product Product\_AProduct\_B

Month

Jan 100.0 90.0

Feb 150.0 80.0

Mar 130.0 120.0

Pivoted Data with Missing Values:

Product Product\_AProduct\_B

Month

Jan 100.0 90.0

Feb NaN 80.0

Mar 130.0 NaN

Filled Data (Missing Values Handled):

Product Product\_AProduct\_B

Month

Jan 100.0 90.0

Feb 126.7 80.0

Mar 130.0 110.0

Summary Statistics of Filled Data:

Product\_AProduct\_B

count 3.0 3.0

mean 118.9 93.3

std 15.6 15.3

min 100.0 80.0

25% 113.3 85.0

50% 126.7 90.0

75% 128.3 100.0

max 130.0 110.0

**Program 02**

**2.a. Program on Data Transformation: String Manipulation.**

**Source code:**

String Manipulation:

# Sample text to work with

text = " Hello, World! Welcome to Python programming. "

# 1. Strip leading and trailing spaces

clean\_text = text.strip()

print(f"Original Text: '{text}'")

print(f"Text after stripping spaces: '{clean\_text}'")

# 2. Convert the text to uppercase

upper\_text = clean\_text.upper()

print(f"\nText in uppercase: '{upper\_text}'")

# 3. Convert the text to lowercase

lower\_text = clean\_text.lower()

print(f"\nText in lowercase: '{lower\_text}'")

# 4. Count occurrences of a substring (e.g., "o")

count\_o = clean\_text.count("o")

print(f"\nNumber of occurrences of 'o': {count\_o}")

# 5. Replace a word in the string

replaced\_text = clean\_text.replace("Python", "Data Science")

print(f"\nText after replacing 'Python' with 'Data Science': '{replaced\_text}'")

# 6. Find the position of a word in the string

position\_world = clean\_text.find("World")

print(f"\nPosition of 'World' in the text: {position\_world}")

# 7. Split the text into words (by default on spaces)

words = clean\_text.split()

print(f"\nList of words in the text: {words}")

# 8. Join the words back into a single string

joined\_text = " ".join(words)

print(f"\nText after joining words: '{joined\_text}'")

# 9. Check if the text starts with "Hello"

starts\_with\_hello = clean\_text.startswith("Hello")

print(f"\nDoes the text start with 'Hello'? {starts\_with\_hello}")

# 10. Check if the text ends with a specific word (e.g., "programming.")

ends\_with\_programming = clean\_text.endswith("programming.")

print(f"\nDoes the text end with 'programming.'? {ends\_with\_programming}")

**Sample output:**

Original Text: ' Hello, World! Welcome to Python programming. '

Text after stripping spaces: 'Hello, World! Welcome to Python programming.'

Text in uppercase: 'HELLO, WORLD! WELCOME TO PYTHON PROGRAMMING.'

Text in lowercase: 'hello, world! welcome to python programming.'

Number of occurrences of 'o': 5

Text after replacing 'Python' with 'Data Science': 'Hello, World! Welcome to Data Science programming.'

Position of 'World' in the text: 7

List of words in the text: ['Hello,', 'World!', 'Welcome', 'to', 'Python', 'programming.']

Text after joining words: 'Hello, World! Welcome to Python programming.'

Does the text start with 'Hello'? True

Does the text end with 'programming.'? True

**Explanation:**

Python program involves various **string manipulation** techniques using built-in string functions such as:

* Convert the string to uppercase.
* Convert the string to lowercase.
* Replace a specific substring with another.
* Split the string into a list of words.
* Reverse the string.
* Remove leading and trailing whitespaces.
* Check if the string starts with a specific substring.
* Count occurrences of a substring.
* Capitalize the first letter of the string.
* Find the position of a substring.

Python offers several built-in methods to manipulate strings, such as:

* **Splitting** (split()): Breaks a string into a list.
* **Joining** (join()): Combines a list of strings into a single string.
* **Replacing** (replace()): Replaces a substring with another substring.
* **Case transformation** (lower(), upper()): Converts the string to lowercase or uppercase.
* **Slicing**: Extracting parts of strings by specifying start and end positions

**2.b. Program on Data Transformation: Regular Expressions**

**Source code:**

import re

# Sample text

text = """

John's email is [email protected]. He said, "Python is awesome!!" It's a great language.

Another email: [email protected].

"""

# 1. Remove special characters except for spaces and email-related characters.

# Using regex to remove non-alphabetic characters and non-email symbols

clean\_text = re.sub(r"[^a-zA-Z0-9@\.\s]", "", text)

print("Text after removing special characters:")

print(clean\_text)

# 2. Convert the text to lowercase

clean\_text = clean\_text.lower()

print("\nText after converting to lowercase:")

print(clean\_text)

# 3. Replace multiple spaces with a single space

clean\_text = re.sub(r"\s+", " ", clean\_text)

print("\nText after replacing multiple spaces:")

print(clean\_text)

# 4. Extract all words starting with a vowel (a, e, i, o, u)

vowel\_words = re.findall(r"\b[aeiouAEIOU]\w+", clean\_text)

print("\nWords starting with a vowel:")

print(vowel\_words)

# 5. Replace email addresses with '[email protected]'

masked\_text = re.sub(r"\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b", "[email protected]", clean\_text)

print("\nText after replacing emails:")

print(masked\_text)

**Explanation:**

**Regular Expressions:**

* **Definition** A regular expression (RegEx) is a sequence of characters that defines a search pattern, mainly for matching strings.
* **Functionality** Python's re module allows you to check if a string matches a specific pattern defined by a regular expression, letting you find, replace, or split strings.
* **Usage** Regular expressions are powerful for pattern matching, enabling operations like validating email formats, searching for specific sequences, and more.
* **Searching**: Finding whether a pattern exists in a string (re.search()).
* **Matching**: Checking if a string fully matches a pattern (re.match()).
* **Finding all matches**: Extracting all occurrences of a pattern in a string (re.findall()).
* **Substituting**: Replacing parts of a string that match a pattern (re.sub()).

**Sample Output:**

Text after removing special characters:

johns email is john.doe@example.com. he said python is awesome its a great language another email jane@example.com

Text after converting to lowercase:

johns email is john.doe@example.com. he said python is awesome its a great language another email jane@example.com

Text after replacing multiple spaces:

johns email is john.doe@example.com. he said python is awesome its a great language another email jane@example.com

Words starting with a vowel:

['email', 'is', 'awesome', 'another', 'email']

Text after replacing emails:

johns email is [email protected]. he said python is awesome its a great language another email [email protected]

### Explanation :

1. **Remove Special Characters**:
   * We use re.sub() to substitute all characters that are not alphabetic, numeric, or email-related symbols (like @ and .). The pattern [^a-zA-Z0-9@\.\s] matches any character not within the alphabet or specified symbols.
2. **Convert to Lowercase**:
   * The built-in method lower() is used to convert all characters to lowercase, standardizing the text for further processing.
3. **Replace Multiple Spaces**:
   * Regex \s+ matches one or more whitespace characters. We replace them with a single space, ensuring uniform spacing between words.
4. **Extract Words Starting with a Vowel**:
   * Using re.findall(), we search for all words starting with a vowel using the regex pattern \b[aeiouAEIOU]\w+. Here, \b ensures that we are matching the beginning of a word.
5. **Replace Email Addresses**:
   * The regex pattern \b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b is designed to match email addresses. We replace any matches with the string [email protected].

**Program 03**

**3.Program on Time series: GroupBy Mechanics to display in data vector, multivariate time series and forecasting formats**

**Source code:**

import pandas as pd

importnumpy as np

importmatplotlib.pyplot as plt

fromstatsmodels.tsa.holtwinters import ExponentialSmoothing

# Create sample time series data

np.random.seed(42)

date\_range = pd.date\_range(start="2022-01-01", end="2023-01-01", freq="D")

data = pd.DataFrame({

"Date": date\_range,

"Value\_A": np.random.normal(100, 10, len(date\_range)),

"Value\_B": np.random.normal(200, 20, len(date\_range)),

})

# Set Date as the index

data.set\_index("Date", inplace=True)

# GroupBy Mechanics

defgroupby\_mechanics(data):

print("\n--- GroupBy Mechanics ---")

# Group data by month and calculate mean

grouped = data.resample('M').mean()

print(grouped)

return grouped

# Data Formats: Vector and Multivariate

defdata\_formats(data):

print("\n--- Data Formats ---")

# Display data as vector

print("\nVector Format:")

print(data["Value\_A"].head())

# Display multivariate time series

print("\nMultivariate Time Series:")

print(data.head())

# Forecasting Example

deftime\_series\_forecasting(data):

print("\n--- Forecasting ---")

# Select a single column for forecasting

ts = data["Value\_A"]

# Train-Test Split

train = ts[:int(0.8 \* len(ts))]

test = ts[int(0.8 \* len(ts)):]

# Fit the Holt-Winters Exponential Smoothing model

model = ExponentialSmoothing(train, seasonal="add", seasonal\_periods=30).fit()

# Forecast for the test period

forecast = model.forecast(len(test))

# Plot results

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

plt.plot(train, label="Train")

plt.plot(test, label="Test")

plt.plot(forecast, label="Forecast")

plt.legend()

plt.title("Time Series Forecasting")

plt.show()

# Main function

if \_\_name\_\_ == "\_\_main\_\_":

print("--- Time Series Data ---")

print(data.head())

# Grouping Mechanics

monthly\_data = groupby\_mechanics(data)

# Data Formats

data\_formats(data)

# Time Series Forecasting

time\_series\_forecasting(data)

**Sample output:**

--- Time Series Data ---

Value\_AValue\_B

Date

2022-01-01 104.967142 204.481850

2022-01-02 98.617357 200.251848

2022-01-03 106.476885 201.953522

2022-01-04 115.230299 184.539804

2022-01-05 97.658466 200.490203

--- GroupBy Mechanics ---

Value\_AValue\_B

Date

2022-01-31 97.985125 202.137470

2022-02-28 98.568317 204.960833

2022-03-31 100.439383 194.956405

2022-04-30 99.797484 198.429574

2022-05-31 99.161855 199.020262

2022-06-30 102.912924 192.752508

2022-07-31 100.983406 199.844253

2022-08-31 99.784632 201.134556

2022-09-30 99.089296 203.720687

2022-10-31 100.649960 198.150774

2022-11-30 102.325711 199.427682

2022-12-31 99.467543 197.195680

2023-01-31 95.987795 180.432544

--- Data Formats ---

Vector Format:

Date

2022-01-01 104.967142

2022-01-02 98.617357

2022-01-03 106.476885

2022-01-04 115.230299

2022-01-05 97.658466

Name: Value\_A, dtype: float64

Multivariate Time Series:

Value\_AValue\_B

Date

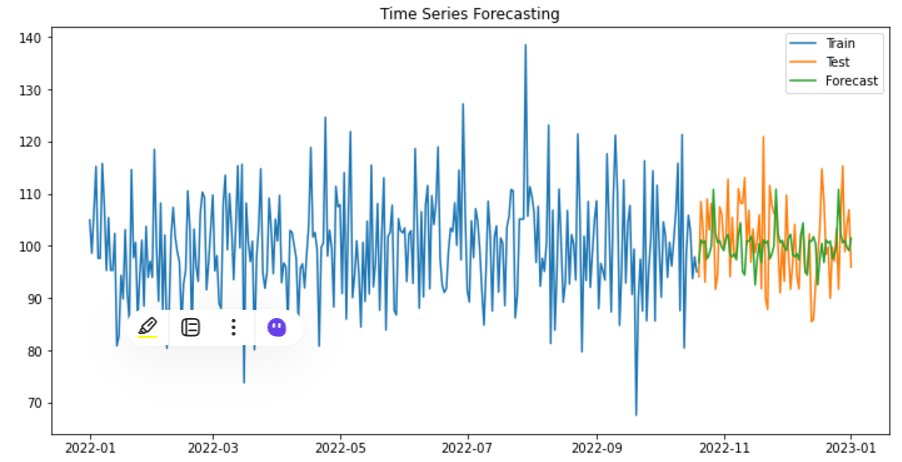
2022-01-01 104.967142 204.481850

2022-01-02 98.617357 200.251848

2022-01-03 106.476885 201.953522

2022-01-04 115.230299 184.539804

2022-01-05 97.658466 200.490203--- Forecasting ---



**Program 04**

**4. Program to measure central tendency and measures of dispersion: Mean, Median, Mode, Standard Deviation, Variance, Mean deviation and Quartile deviation for a frequency distribution/data**

**Source code:**

# Import necessary libraries

importnumpy as np

import pandas as pd

defcalculate\_statistics(data, frequencies):

# Create a DataFrame for the frequency distribution

df = pd.DataFrame({'Value': data, 'Frequency': frequencies})

# Calculate the total number of observations

total = df['Frequency'].sum()

# Calculate mean

df['Weighted\_Value'] = df['Value'] \* df['Frequency']

mean = df['Weighted\_Value'].sum() / total

# Calculate median

cumulative\_frequency = df['Frequency'].cumsum()

median\_index = cumulative\_frequency.searchsorted(total / 2)

median = df['Value'][median\_index]

# Calculate mode

mode = df['Value'][df['Frequency'].idxmax()]

# Calculate variance and standard deviation

variance = np.average((df['Value'] - mean) \*\* 2, weights=df['Frequency'])

std\_deviation = np.sqrt(variance)

# Calculate mean deviation

mean\_deviation = np.average(np.abs(df['Value'] - mean), weights=df['Frequency'])

# Calculate quartile deviation

q1 = np.percentile(data, 25)

q3 = np.percentile(data, 75)

quartile\_deviation = (q3 - q1) / 2

return {

'Mean': mean,

'Median': median,

'Mode': mode,

'Variance': variance,

'Standard Deviation': std\_deviation,

'Mean Deviation': mean\_deviation,

'Quartile Deviation': quartile\_deviation

}

# Get user input for data and frequencies

data\_input = input("Enter the data values separated by commas (e.g., 10, 20, 30): ")

frequencies\_input = input("Enter the corresponding frequencies separated by commas (e.g., 1, 2, 3): ")

# Convert input strings to lists of integers

data = list(map(int, data\_input.split(',')))

frequencies = list(map(int, frequencies\_input.split(',')))

# Calculate statistics

statistics = calculate\_statistics(data, frequencies)

# Display the results

for stat, value in statistics.items():

print(f"{stat}: {value:.2f}")

**Sample output:**

**Data values**: 10, 20, 20, 30, 30, 30, 40, 50, 50, 60

**Frequencies**: 1, 2, 3, 1, 1, 1, 1, 2, 2, 1

Mean:36.00

Median:30.00

Mode:30.00

Variance:167.78

Standard Deviation:12.93

Mean Deviation:11.70

Quartile Deviation:10.00

**Explanation:**

**1. Mean (Average)**

* **What it is**: The mean is what we usually call the average. It tells us what the "typical" number is when we have a bunch of numbers.
* **How to find it**: To find the mean, we add up all the numbers and then divide by how many numbers there are.
* **Example**: If you have 2 candies, 3 candies, and 5 candies, you first add them up (2 + 3 + 5 = 10) and then divide by how many groups of candies there are (3). So, the mean is 10÷3=3.3310 ÷ 3 = 3.3310÷3=3.33.

**2. Median**

* **What it is**: The median is the middle number in a list of numbers. If we lined up all our numbers from smallest to largest, the median would be the one right in the middle.
* **How to find it**: If there’s an odd number of numbers, the median is the one in the middle. If there’s an even number, we take the two middle numbers, add them together, and divide by 2.
* **Example**: For the numbers 2, 3, 5, the median is 3. For 2, 3, 4, 5, the median is (3+4)÷2=3.5 (3 + 4) ÷ 2 = 3.5(3+4)÷2=3.5.

**3. Mode**

* **What it is**: The mode is the number that appears the most in a list. It tells us which number is the most popular.
* **How to find it**: We look at our list of numbers and see which one shows up the most times.
* **Example**: If you have candies like this: 2, 2, 3, 4, the mode is 2 because it appears more times than any other number.

**4. Variance**

* **What it is**: Variance tells us how spread out the numbers are. If the numbers are all close to the mean, the variance is small. If they’re very different from each other, the variance is big.
* **How to find it**: We calculate how far each number is from the mean, square those differences, and then average them.
* **Example**: If the numbers are 2, 3, and 4, they are all close to the mean (3), so the variance is small. If the numbers are 1, 5, and 10, they are more spread out, so the variance is larger.

**5. Standard Deviation**

* **What it is**: Standard deviation is just a fancy word for how much the numbers vary from the mean. It’s like the square root of the variance.
* **How to find it**: We take the square root of the variance.
* **Example**: If the variance tells us how far apart the candies are from the average size, the standard deviation gives us a way to understand that distance in the same units as the candies.

**6. Mean Deviation**

* **What it is**: Mean deviation tells us, on average, how far each number is from the mean. It helps us see how different the numbers are from the average number.
* **How to find it**: We take the absolute value of the differences between each number and the mean, and then find the average of those differences.
* **Example**: If the mean is 3 and the numbers are 2, 3, and 4, the distances from the mean are 1 (for 2), 0 (for 3), and 1 (for 4). The mean deviation would be (1+0+1)÷3=0.67 (1 + 0 + 1) ÷ 3 = 0.67(1+0+1)÷3=0.67.

**7. Quartile Deviation**

* **What it is**: Quartile deviation measures how spread out the middle half of the data is. It uses the first quartile (Q1) and the third quartile (Q3) to find the "interquartile range."
* **How to find it**: We first find Q1 and Q3, then subtract Q1 from Q3, and divide by 2.
* **Example**: If Q1 is 2 and Q3 is 4, the quartile deviation is (4−2)÷2=1 (4 - 2) ÷ 2 = 1(4−2)÷2=1. This means the middle half of the candies is spread out over 1 candy.

**Program 05**

**5.Program to perform cross validation for a given dataset to measure Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and R2 Error using Validation Set, Leave One Out Cross-Validation(LOOCV) and K-fold Cross-Validation approaches**

**Source code:**

# Import necessary libraries

importnumpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split, KFold, LeaveOneOut

fromsklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from sklearn.linear\_model import LinearRegression

fromsklearn.datasets import fetch\_california\_housing

# Load the California housing dataset

data = fetch\_california\_housing()

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

y = pd.Series(data.target)

# Function to calculate and display metrics

defdisplay\_metrics(y\_true, y\_pred):

rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

mae = mean\_absolute\_error(y\_true, y\_pred)

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

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

print(f"Mean Absolute Error (MAE): {mae:.4f}")

print(f"R² Score: {r2:.4f}")

returnrmse, mae, r2

# Validation Set Approach

defvalidation\_set\_approach(X, y):

print("Validation Set Approach:")

# Split the dataset into training (80%) and validation (20%) sets

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

# Initialize and train the model

model = LinearRegression()

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

# Make predictions on the validation set

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

# Display metrics

display\_metrics(y\_val, y\_pred)

# Leave-One-Out Cross-Validation (LOOCV) Approach

defloocv\_approach(X, y):

print("Leave-One-Out Cross-Validation (LOOCV):")

loo = LeaveOneOut()

y\_true, y\_pred = [], []

# Loop through each sample using LOOCV

fortrain\_index, test\_index in loo.split(X):

X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]

y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

# Initialize and train the model

model = LinearRegression()

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

# Make prediction for the single test sample

y\_pred.append(model.predict(X\_test)[0])

y\_true.append(y\_test.iloc[0])

# Display metrics

display\_metrics(y\_true, y\_pred)

# K-Fold Cross-Validation Approach

defkfold\_approach(X, y, k=5):

print(f"{k}-Fold Cross-Validation Approach:")

kf = KFold(n\_splits=k, shuffle=True, random\_state=42)

y\_true, y\_pred = [], []

# Loop through each fold

fortrain\_index, test\_index in kf.split(X):

X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]

y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

# Initialize and train the model

model = LinearRegression()

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

# Make predictions on the test set

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

y\_true.extend(y\_test)

# Display metrics

display\_metrics(y\_true, y\_pred)

# Main function to run all approaches

def main():

print("Cross-Validation for RMSE, MAE, and R²:\n")

validation\_set\_approach(X, y)

print("\n")

loocv\_approach(X, y)

print("\n")

kfold\_approach(X, y, k=5) # You can change k for different K-Fold Cross-Validation

# Execute the main function

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Sample output:**

Cross-Validation for RMSE, MAE, and R²:

Validation Set Approach:

Root Mean Squared Error (RMSE): 0.7456

Mean Absolute Error (MAE): 0.5332

R² Score: 0.5758

Leave-One-Out Cross-Validation (LOOCV):

Root Mean Squared Error (RMSE): 0.7268

Mean Absolute Error (MAE): 0.5317

R² Score: 0.6033

5-Fold Cross-Validation Approach:

Root Mean Squared Error (RMSE): 0.7284

Mean Absolute Error (MAE): 0.5317

R² Score: 0.6015

**Program 06**

**6.Program to display Normal, Binomial Poisson, Bernoulli disrtibutions for a given frequency distribution**

**Source code:**

Import numpy as np

Import mat plotlib.pyplot as plt

From scipy.stats import norm, binom, poisson, bernoulli

defget\_user\_data():

# Get the frequency distribution input from the user

data\_input = input("Enter the data values separated by commas (e.g., 10, 20, 30): ")

frequencies\_input = input("Enter the corresponding frequencies separated by commas (e.g., 2, 3, 4): ")

# Convert the inputs into lists of integers

data = list(map(int, data\_input.split(',')))

frequencies = list(map(int, frequencies\_input.split(',')))

return data, frequencies

defplot\_normal\_distribution(data, frequencies):

# Fit and plot Normal distribution

mean = np.mean(data)

std\_dev = np.std(data)

x = np.linspace(min(data), max(data), 100)

pdf = norm.pdf(x, mean, std\_dev)

plt.plot(x, pdf, 'r-', lw=2, label='Normal Distribution')

plt.title('Normal Distribution')

plt.xlabel('Value')

plt.ylabel('Probability Density')

plt.show()

defplot\_binomial\_distribution(data, frequencies):

# Fit and plot Binomial distribution (assuming n is max(data) and p is mean/len(data))

n = max(data)

p = np.mean(data) / n

x = np.arange(0, n+1)

pmf = binom.pmf(x, n, p)

plt.bar(x, pmf, alpha=0.7, color='b', label='Binomial Distribution')

plt.title('Binomial Distribution')

plt.xlabel('Value')

plt.ylabel('Probability')

plt.show()

defplot\_poisson\_distribution(data, frequencies):

# Fit and plot Poisson distribution (lambda is the mean of the data)

lam = np.mean(data)

x = np.arange(0, max(data)+1)

pmf = poisson.pmf(x, lam)

plt.bar(x, pmf, alpha=0.7, color='g', label='Poisson Distribution')

plt.title('Poisson Distribution')

plt.xlabel('Value')

plt.ylabel('Probability')

plt.show()

defplot\_bernoulli\_distribution(data, frequencies):

# Assuming binary outcome for Bernoulli

success\_prob = np.mean(data) / max(data)

x = [0, 1]

pmf = bernoulli.pmf(x, success\_prob)

plt.bar(x, pmf, alpha=0.7, color='purple', label='Bernoulli Distribution')

plt.title('Bernoulli Distribution')

plt.xlabel('Value')

plt.ylabel('Probability')

plt.show()

defanalyze\_distributions(data, frequencies):

print("Analyzing Normal Distribution:")

plot\_normal\_distribution(data, frequencies)

print("Analyzing Binomial Distribution:")

plot\_binomial\_distribution(data, frequencies)

print("Analyzing Poisson Distribution:")

plot\_poisson\_distribution(data, frequencies)

print("Analyzing Bernoulli Distribution:")

plot\_bernoulli\_distribution(data, frequencies)

# Main program

data, frequencies = get\_user\_data()

analyze\_distributions(data, frequencies)

**Explanation:**

**1. Normal Distribution**

* **What it is**: The Normal distribution (also called the Gaussian distribution) is the most common probability distribution. It looks like a bell-shaped curve and is symmetrical around the mean.
* **Real-world analogy**: Imagine you’re measuring the height of a group of people. Most people will have heights around the average (mean), and fewer people will be much taller or much shorter. The majority of the data clusters around the average with fewer extreme values on either side.
* **Key Points**:
  + Symmetrical bell curve.
  + Mean = Median = Mode.
  + Many natural phenomena, like people's heights, shoe sizes, or IQ scores, follow a Normal distribution.

**2. Binomial Distribution**

* **What it is**: The Binomial distribution is used to model the number of successful outcomes in a fixed number of independent trials, where each trial has two possible outcomes (like flipping a coin: heads or tails).
* **Real-world analogy**: Imagine you flip a coin 10 times. If you’re trying to figure out how many heads you’ll get, the Binomial distribution helps you calculate the probability of getting 0 heads, 1 head, 2 heads, all the way up to 10 heads.
* **Key Points**:
  + Two possible outcomes: success or failure (like heads or tails).
  + Fixed number of trials (e.g., flipping a coin 10 times).
  + Probability of success (like getting heads) stays the same for each trial.

**3. Poisson Distribution**

* **What it is**: The Poisson distribution is used to model the probability of a given number of events happening in a fixed interval of time or space, where these events occur independently of each other.
* **Real-world analogy**: Imagine you’re running a bakery, and you want to know the probability of a certain number of customers arriving in the next hour. The Poisson distribution can help you predict this based on the average number of customers you usually get in an hour.
* **Key Points**:
  + The events are random and independent.
  + It’s used when you’re counting the number of occurrences over a fixed period or space (e.g., customers arriving, phone calls received).
  + Average rate of occurrence (λ or lambda) is constant.

**4. Bernoulli Distribution**

* **What it is**: The Bernoulli distribution models the probability of a single trial with two possible outcomes, typically referred to as success (1) or failure (0).
* **Real-world analogy**: Suppose you're tossing a coin one time. The outcome can only be heads or tails (success or failure). The Bernoulli distribution helps model this scenario.
* **Key Points**:
  + There is just one trial (e.g., one coin flip).
  + The probability of success is constant.
  + The result is binary: either success (1) or failure (0).

**Key Differences:**

* **Normal Distribution**: Continuous and symmetrical. It models things like people's heights, weights, and test scores where the data is spread around the mean in a bell-shaped curve.
* **Binomial Distribution**: Discrete. It models how many times a success happens out of a fixed number of trials, such as how many heads you get when you flip a coin 10 times.
* **Poisson Distribution**: Discrete. It models how many times an event happens in a fixed amount of time or space (e.g., the number of cars passing by in an hour or phone calls received).
* **Bernoulli Distribution**: Discrete and binary. It models a single yes/no outcome (success or failure) for one event, such as one coin flip.

### Visualizing Distributions:

* **Normal Distribution**: Think of a bell curve. Most values are clustered around the mean, and fewer values are farther away.
* **Binomial Distribution**: The histogram of outcomes peaks around the most likely number of successes (like getting 5 heads out of 10 coin flips).
* **Poisson Distribution**: It shows how the likelihood of different numbers of events drops off quickly (e.g., it’s rare to get 10 customers in 5 minutes, but more common to get 1 or 2).
* **Bernoulli Distribution**: It’s simple and binary, like flipping a coin. The bar chart has just two bars: one for success (1) and one for failure (0).

**Program 07**

**7.Program to implement one sample, two sample and paired sample t-tests for sample data and analyzethe results.**

**Source code:**

# Import necessary libraries

importnumpy as np

import pandas as pd

fromscipy import stats

# Sample data for demonstration

# One-sample test: A group of exam scores

exam\_scores = np.array([85, 87, 90, 78, 88, 95, 82, 79, 94, 91])

# Two-sample test: Scores of two different groups

group\_A = np.array([85, 89, 88, 90, 93, 85, 84, 79, 90, 87])

group\_B = np.array([82, 86, 85, 87, 92, 80, 81, 78, 89, 85])

# Paired-sample test: Before and after treatment scores of the same group

before\_treatment = np.array([82, 84, 88, 78, 80, 85, 90, 79, 87, 83])

after\_treatment = np.array([85, 87, 89, 81, 83, 88, 92, 82, 89, 86])

# Function to perform one-sample t-test

defone\_sample\_ttest(data, population\_mean):

t\_stat, p\_value = stats.ttest\_1samp(data, population\_mean)

returnt\_stat, p\_value

# Function to perform two-sample t-test (independent samples)

deftwo\_sample\_ttest(group1, group2):

t\_stat, p\_value = stats.ttest\_ind(group1, group2)

returnt\_stat, p\_value

# Function to perform paired-sample t-test

defpaired\_sample\_ttest(before, after):

t\_stat, p\_value = stats.ttest\_rel(before, after)

returnt\_stat, p\_value

# Analyze results of the t-tests

defanalyze\_ttest\_results(t\_stat, p\_value, alpha=0.05):

print(f"T-statistic: {t\_stat}")

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

ifp\_value< alpha:

print("Result: The null hypothesis is rejected (statistically significant difference).")

else:

print("Result: The null hypothesis cannot be rejected (no statistically significant difference).")

# One-sample t-test: Compare exam scores with a population mean (e.g., 85)

print("One-Sample T-Test:")

t\_stat, p\_value = one\_sample\_ttest(exam\_scores, 85)

analyze\_ttest\_results(t\_stat, p\_value)

print()

# Two-sample t-test: Compare the means of two independent groups

print("Two-Sample T-Test:")

t\_stat, p\_value = two\_sample\_ttest(group\_A, group\_B)

analyze\_ttest\_results(t\_stat, p\_value)

print()

# Paired-sample t-test: Compare before and after treatment of the same group

print("Paired-Sample T-Test:")

t\_stat, p\_value = paired\_sample\_ttest(before\_treatment, after\_treatment)

analyze\_ttest\_results(t\_stat, p\_value)

**Sample Output:**

One-Sample T-Test:

T-statistic: 1.0189950494649807

P-value: 0.3348142605778697

Result: The null hypothesis cannot be rejected (no statistically significant difference).

Two-Sample T-Test:

T-statistic: 1.3547090246981803

P-value: 0.19227122007981406

Result: The null hypothesis cannot be rejected (no statistically significant difference).

Paired-Sample T-Test:

T-statistic: -11.758942438532781

P-value: 9.151111215642479e-07

Result: The null hypothesis is rejected (statistically significant difference).

​

**Program 08**

**8.Program to implement One-way and Two-way ANOVA tests and analyze the results**

**Source code:**

importnumpy as np

import pandas as pd

fromscipy.stats import f\_oneway

importstatsmodels.api as sm

fromstatsmodels.formula.api import ols

# Function for One-way ANOVA

defone\_way\_anova(data, groups, response):

"""

Perform one-way ANOVA.

:param data: DataFrame containing the dataset

:param groups: Column name for grouping variable

:param response: Column name for response variable

"""

grouped\_data = [group[response].values for \_, group in data.groupby(groups)]

f\_stat, p\_value = f\_oneway(\*grouped\_data)

print("\nOne-way ANOVA Results:")

print(f"F-statistic: {f\_stat:.4f}, p-value: {p\_value:.4f}")

ifp\_value< 0.05:

print("Reject the null hypothesis: Significant difference among group means.")

else:

print("Fail to reject the null hypothesis: No significant difference among group means.")

# Function for Two-way ANOVA

deftwo\_way\_anova(data, response, factor1, factor2):

"""

Perform two-way ANOVA.

:param data: DataFrame containing the dataset

:param response: Column name for response variable

:param factor1: Column name for first factor

:param factor2: Column name for second factor

"""

formula = f"{response} ~ C({factor1}) + C({factor2}) + C({factor1}):C({factor2})"

model = ols(formula, data).fit()

anova\_table = sm.stats.anova\_lm(model, typ=2) # Type II ANOVA

print("\nTwo-way ANOVA Results:")

print(anova\_table)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Example dataset for One-way ANOVA

data\_one\_way = pd.DataFrame({

"Group": np.repeat(['A', 'B', 'C'], 10),

"Score": np.concatenate([

np.random.normal(loc=50, scale=5, size=10),

np.random.normal(loc=55, scale=5, size=10),

np.random.normal(loc=60, scale=5, size=10)

])

})

# Perform One-way ANOVA

one\_way\_anova(data\_one\_way, groups="Group", response="Score")

# Example dataset for Two-way ANOVA

data\_two\_way = pd.DataFrame({

"Factor1": np.repeat(['Low', 'Medium', 'High'], 6),

"Factor2": np.tile(['Type1', 'Type2'], 9),

"Response": np.concatenate([

np.random.normal(loc=50, scale=5, size=6),

np.random.normal(loc=55, scale=5, size=6),

np.random.normal(loc=60, scale=5, size=6)

])

})

# Perform Two-way ANOVA

two\_way\_anova(data\_two\_way, response="Response", factor1="Factor1", factor2="Factor2")

**Sample output:**

One-way ANOVA Results:

F-statistic: 25.1121, p-value: 0.0000

Reject the null hypothesis: Significant difference among group means.

Two-way ANOVA Results:

sum\_sqdf F PR(>F)

C(Factor1) 159.837203 2.0 2.086902 0.166807

C(Factor2) 56.395083 1.0 1.472636 0.248276

C(Factor1):C(Factor2) 39.362026 2.0 0.513927 0.610731

Residual 459.544039 12.0NaNNaN

**Program 09**

**9.Program to implement correlation, rank correlation and regression and plot x-y plot and heat maps of correlation matrices.**

**Source code:**

# Import required libraries

importnumpy as np

import pandas as pd

importseaborn as sns

importmatplotlib.pyplot as plt

fromscipy.stats import spearmanr

from sklearn.linear\_model import LinearRegression

fromsklearn.metrics import mean\_squared\_error

# Generate sample data (or load your dataset here)

np.random.seed(42) # For reproducibility

x = np.random.rand(100) \* 100 # Random values for x

y = 2.5 \* x + np.random.normal(0, 25, 100) # Linear relation with noise

# Convert data into a DataFrame

data = pd.DataFrame({'X': x, 'Y': y})

# Compute Correlation

pearson\_corr = data.corr(method='pearson') # Pearson Correlation

spearman\_corr, \_ = spearmanr(data['X'], data['Y']) # Spearman Rank Correlation

# Linear Regression

X = data['X'].values.reshape(-1, 1) # Reshape for sklearn

Y = data['Y'].values

model = LinearRegression()

model.fit(X, Y)

Y\_pred = model.predict(X)

regression\_coeff = model.coef\_[0] # Slope

regression\_intercept = model.intercept\_ # Intercept

mse = mean\_squared\_error(Y, Y\_pred)

# Print statistical results

print("Pearson Correlation Coefficient Matrix:")

print(pearson\_corr)

print("\nSpearman Rank Correlation Coefficient:", spearman\_corr)

print("\nLinear Regression Equation: Y = {:.2f}X + {:.2f}".format(regression\_coeff, regression\_intercept))

print("Mean Squared Error (MSE):", mse)

# Plot X-Y scatter plot with regression line

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

plt.scatter(data['X'], data['Y'], color='blue', label='Data Points')

plt.plot(data['X'], Y\_pred, color='red', label='Regression Line')

plt.title('X-Y Scatter Plot with Regression Line')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.show()

# Plot heatmap of correlation matrix

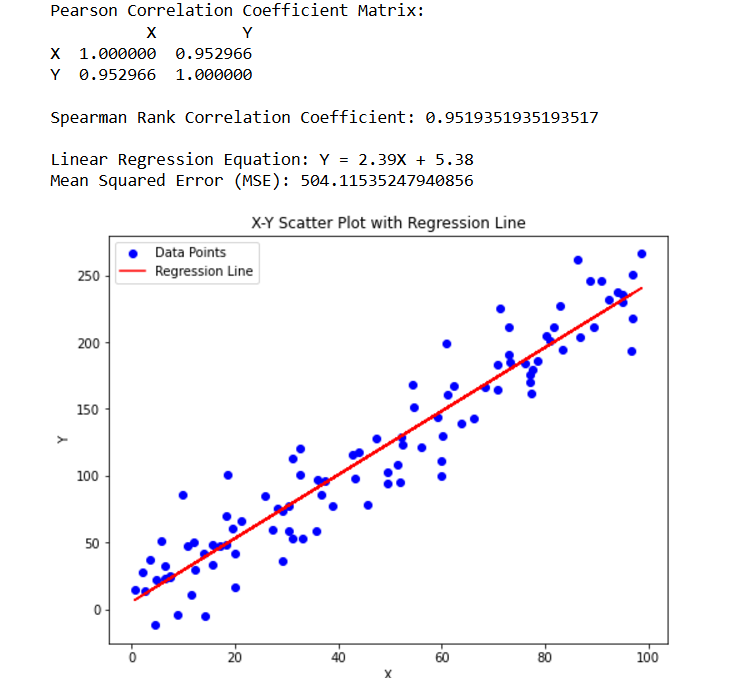
plt.figure(figsize=(6, 5))

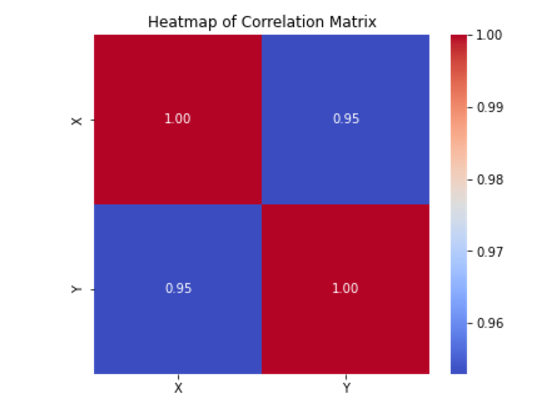
sns.heatmap(pearson\_corr, annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Heatmap of Correlation Matrix')

plt.show()

# Note: To use this script with a custom dataset, replace the sample data generation section with loading your dataset using pandas (e.g., pd.read\_csv).





**Program 10**

10**.Program to implement PCA for Wisconsin dataset, visualize and analyze the results**.

**Source code:**

# Import necessary libraries

importnumpy as np

import pandas as pd

importmatplotlib.pyplot as plt

importseaborn as sns

fromsklearn.datasets import load\_breast\_cancer

fromsklearn.decomposition import PCA

fromsklearn.preprocessing import StandardScaler

# Load the Wisconsin Breast Cancer dataset

data = load\_breast\_cancer()

X = data.data # Features

y = data.target # Target variable (0 = malignant, 1 = benign)

feature\_names = data.feature\_names

target\_names = data.target\_names

# Standardize the data (important for PCA)

scaler = StandardScaler()

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

# Apply PCA

pca = PCA(n\_components=2) # Reduce to 2 dimensions for visualization

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

# Get explained variance ratio for each component

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

# Create a DataFrame for visualization

pca\_df = pd.DataFrame(X\_pca, columns=['PCA1', 'PCA2'])

pca\_df['Target'] = y

# Plot the PCA results

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

sns.scatterplot(data=pca\_df, x='PCA1', y='PCA2', hue='Target', palette='Set1', alpha=0.8)

plt.title('PCA of Wisconsin Breast Cancer Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend(target\_names)

plt.grid()

plt.show()

# Plot explained variance ratio

plt.figure(figsize=(8, 5))

plt.bar(range(1, 3), explained\_variance\_ratio, tick\_label=['PCA1', 'PCA2'], color='skyblue')

plt.title('Explained Variance Ratio of PCA Components')

plt.xlabel('Principal Components')

plt.ylabel('Variance Explained')

plt.show()

# Full PCA with all components for analysis

pca\_full = PCA()

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

cumulative\_variance = np.cumsum(pca\_full.explained\_variance\_ratio\_)

# Plot cumulative explained variance

plt.figure(figsize=(8, 5))

plt.plot(range(1, len(cumulative\_variance) + 1), cumulative\_variance, marker='o', linestyle='--', color='b')

plt.title('Cumulative Explained Variance')

plt.xlabel('Number of Principal Components')

plt.ylabel('Cumulative Variance Explained')

plt.grid()

plt.show()

# Print key insights

print("PCA Analysis of Wisconsin Breast Cancer Dataset")

print("-------------------------------------------------")

print(f"Explained Variance (PCA1): {explained\_variance\_ratio[0]:.4f}")

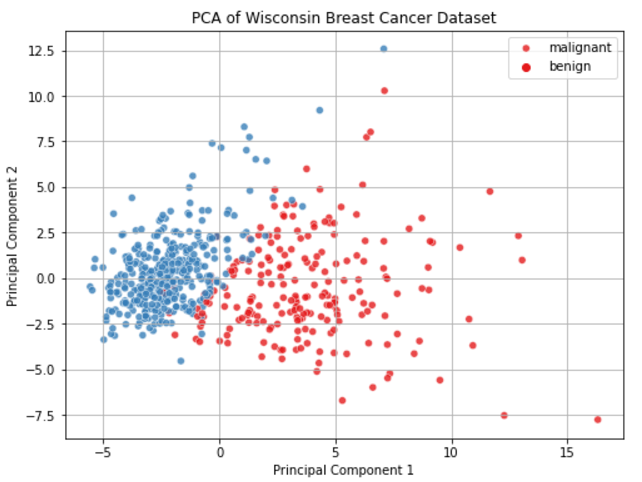
print(f"Explained Variance (PCA2): {explained\_variance\_ratio[1]:.4f}")

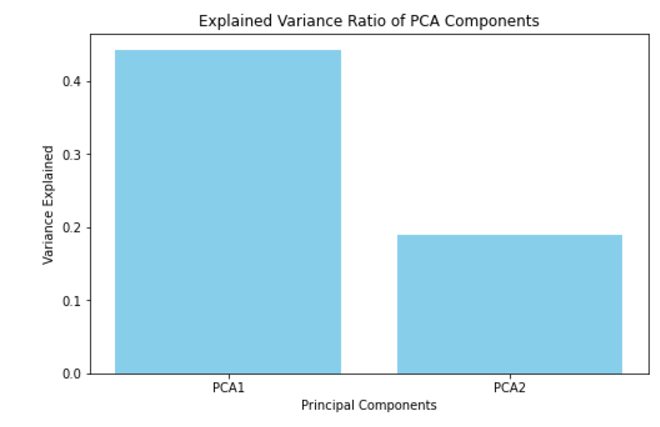
print("Cumulative Variance Explained by All Components:")

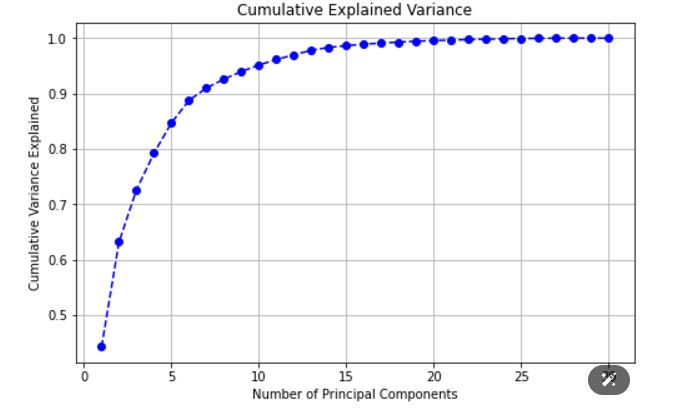
fori, cum\_var in enumerate(cumulative\_variance, start=1):

print(f" Component {i}: {cum\_var:.4f}")

**Sample Output:**







PCA Analysis of Wisconsin Breast Cancer Dataset

-------------------------------------------------

Explained Variance (PCA1): 0.4427

Explained Variance (PCA2): 0.1897

Cumulative Variance Explained by All Components:

Component 1: 0.4427

Component 2: 0.6324

Component 3: 0.7264

Component 4: 0.7924

Component 5: 0.8473

Component 6: 0.8876

Component 7: 0.9101

Component 8: 0.9260

Component 9: 0.9399

Component 10: 0.9516

Component 11: 0.9614

Component 12: 0.9701

Component 13: 0.9781

Component 14: 0.9834

Component 15: 0.9865

Component 16: 0.9892

Component 17: 0.9911

Component 18: 0.9929

Component 19: 0.9945

Component 20: 0.9956

Component 21: 0.9966

Component 22: 0.9975

Component 23: 0.9983

Component 24: 0.9989

Component 25: 0.9994

Component 26: 0.9997

Component 27: 0.9999

Component 28: 1.0000

Component 29: 1.0000

Component 30: 1.0000

**Program 11**

**11.Program to implement the working of linear discriminant analysis using iris dataset and visualize the results**.

**Source code:**

# Import necessary libraries

importnumpy as np

import pandas as pd

importmatplotlib.pyplot as plt

importseaborn as sns

fromsklearn.datasets import load\_iris

fromsklearn.discriminant\_analysis import LinearDiscriminantAnalysis

fromsklearn.preprocessing import StandardScaler

# Load the Iris dataset

data = load\_iris()

X = data.data # Features

y = data.target # Target variable (0, 1, 2)

target\_names = data.target\_names # Class names

# Standardize the data (LDA benefits from scaling)

scaler = StandardScaler()

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

# Apply Linear Discriminant Analysis (LDA)

lda = LinearDiscriminantAnalysis(n\_components=2) # Reduce to 2 components for visualization

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

# Create a DataFrame for LDA-transformed data

lda\_df = pd.DataFrame(X\_lda, columns=['LDA1', 'LDA2'])

lda\_df['Target'] = y

# Plot the LDA results in 2D space

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

sns.scatterplot(data=lda\_df, x='LDA1', y='LDA2', hue='Target', palette='Set1', style='Target', s=100)

plt.title('LDA of Iris Dataset')

plt.xlabel('Linear Discriminant 1')

plt.ylabel('Linear Discriminant 2')

plt.legend(title='Class', labels=target\_names)

plt.grid()

plt.show()

# Print key insights

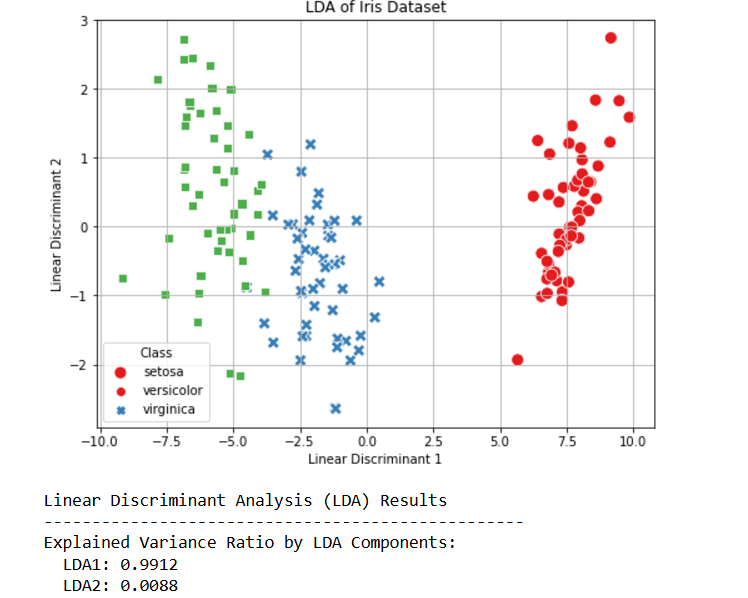
print("Linear Discriminant Analysis (LDA) Results")

print("--------------------------------------------------")

print("Explained Variance Ratio by LDA Components:")

fori, ratio in enumerate(lda.explained\_variance\_ratio\_, start=1):

print(f" LDA{i}: {ratio:.4f}")

**Sample Output:** 

**Program 12**

**12.Program to Implement multiple linear regression using iris dataset, visualize and analyse the results.**

**Source code:**

# Import necessary libraries

Import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

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

from sklearn.linear\_model import Linear Regression

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

# Load the Iris dataset

data = load\_iris()

X = pd.DataFrame(data.data, columns=data.feature\_names) # Features

y = X['petal length (cm)'] # Let's predict 'petal length' as the dependent variable

X = X.drop(columns=['petal length (cm)']) # Remove 'petal length' from independent variables

# 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)

# Apply Multiple Linear Regression

model = LinearRegression()

model.fit(X\_train, y\_train) # Train the model

# Predict 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 model performance metrics

print("Multiple Linear Regression Results")

print("----------------------------------")

print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"R-squared (R²): {r2:.4f}")

print("\nModel Coefficients:")

for feature, coef in zip(X.columns, model.coef\_):

print(f" {feature}: {coef:.4f}")

print(f"Intercept: {model.intercept\_:.4f}")

# Visualize actual vs predicted values

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

plt.scatter(y\_test, y\_pred, color='blue', alpha=0.7)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linewidth=2, linestyle='--')

plt.title('Actual vs Predicted Values (Test Set)')

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.grid()

plt.show()

# Pair plot to explore relationships in the dataset

sns.pairplot(pd.DataFrame(data.data, columns=data.feature\_names), diag\_kind='kde')

plt.suptitle('Pairplot of Iris Dataset Features', y=1.02)

plt.show()

**Sample Output**:

