**Fundamental of data science**

**Important topic unit wise answer key**

Unit – 1

Concepts and Definitions

1. Load Data Set in Pandas DataFrame : Loading data from a file (e.g., CSV, Excel) into a Pandas DataFrame.

2. Create New Column : Adding a new column to the DataFrame based on existing data.

3. Display First 5 Rows and Last 5 Rows : Viewing the top and bottom rows of the DataFrame.

4. Visualization : Creating various plots such as histogram, scatter plot, line plot, and bar plot.

5. Retrieving Information : Accessing specific information from the DataFrame using indexing, slicing, and queries.

6. Data Manipulation : Modifying and transforming data within the DataFrame, including filtering, sorting, grouping, and aggregation.

### Sample Dataset and Implementation

Sample Dataset

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Create a sample dataset

data = {

'product\_id': [101, 102, 103, 104, 105],

'sales': [150, 200, 250, 300, 350],

'discount': [10, 15, 20, 25, 30], # Discount percentages

'growth\_percentage': [5, 10, -5, 20, 15] # Percentage growth

}

df = pd.DataFrame(data)

print("Sample Dataset:\n", df)

Implementation

**1. Load Data Set in Pandas DataFrame**

Loading data from a CSV file (example):

# Load dataset from a CSV file

# df = pd.read\_csv('sample\_data.csv')

# Since we are creating a sample dataset directly, we'll skip loading from a file.

print("Dataset Loaded")

Output:

Dataset Loaded

**2. Create New Column**

Adding a new column `discounted\_sales` which represents sales after discount:

df['discounted\_sales'] = df['sales'] - (df['sales'] \* df['discount'] / 100)

print("Dataset with new column:\n", df)

Output:

Dataset with new column:

product\_id sales discount growth\_percentage discounted\_sales

0 101 150 10 5 135.0

1 102 200 15 10 170.0

2 103 250 20 -5 200.0

3 104 300 25 20 225.0

4 105 350 30 15 245.0

**3. Display First 5 Rows and Last 5 Rows**

print("First 5 rows:\n", df.head())

print("Last 5 rows:\n", df.tail())

Output:

First 5 rows:

product\_id sales discount growth\_percentage discounted\_sales

0 101 150 10 5 135.0

1 102 200 15 10 170.0

2 103 250 20 -5 200.0

3 104 300 25 20 225.0

4 105 350 30 15 245.0

Last 5 rows:

product\_id sales discount growth\_percentage discounted\_sales

0 101 150 10 5 135.0

1 102 200 15 10 170.0

2 103 250 20 -5 200.0

3 104 300 25 20 225.0

4 105 350 30 15 245.0

```

**4. Visualization**

Histogram

plt.hist(df['sales'], bins=5, alpha=0.7, color='blue')

plt.title('Sales Distribution')

plt.xlabel('Sales')

plt.ylabel('Frequency')

plt.show()

**Scatter Plot**

plt.scatter(df['sales'], df['growth\_percentage'], color='red')

plt.title('Sales vs Growth Percentage')

plt.xlabel('Sales')

plt.ylabel('Growth Percentage')

plt.show()

**Line Plot**

plt.plot(df['product\_id'], df['sales'], marker='o', linestyle='-', color='green')

plt.title('Sales Over Products')

plt.xlabel('Product ID')

plt.ylabel('Sales')

plt.show()

**Bar Plot**

plt.bar(df['product\_id'], df['sales'], color='orange')

plt.title('Sales by Product')

plt.xlabel('Product ID')

plt.ylabel('Sales')

plt.show()

**5. Retrieving Information**

Using Indexing and Slicing

# Retrieving a single column

sales\_column = df['sales']

print("Sales Column:\n", sales\_column)

# Retrieving multiple columns

sales\_discount\_columns = df[['sales', 'discount']]

print("Sales and Discount Columns:\n", sales\_discount\_columns)

# Retrieving a specific row

row\_2 = df.iloc[1]

print("Row 2:\n", row\_2)

# Retrieving a range of rows

rows\_1\_3 = df.iloc[0:3]

print("Rows 1 to 3:\n", rows\_1\_3)

```

**6. Data Manipulatio**

**Filtering Data**

high\_sales = df[df['sales'] > 250]

print("High Sales:\n", high\_sales)

**Sorting Data**

sorted\_by\_sales = df.sort\_values(by='sales', ascending=False)

print("Sorted by Sales:\n", sorted\_by\_sales)

**Grouping and Aggregation**

average\_sales\_by\_growth = df.groupby('growth\_percentage')['sales'].mean()

print("Average Sales by Growth Percentage:\n", average\_sales\_by\_growth)

**COMPLETE CODE**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Create a sample dataset

data = {

'product\_id': [101, 102, 103, 104, 105],

'sales': [150, 200, 250, 300, 350],

'discount': [10, 15, 20, 25, 30], # Discount percentages

'growth\_percentage': [5, 10, -5, 20, 15] # Percentage growth

}

df = pd.DataFrame(data)

print("Sample Dataset:\n", df)

# Create new column

df['discounted\_sales'] = df['sales'] - (df['sales'] \* df['discount'] / 100)

print("Dataset with new column:\n", df)

# Display first 5 rows and last 5 rows

print("First 5 rows:\n", df.head())

print("Last 5 rows:\n", df.tail())

# Visualizatio

# Histogram

plt.hist(df['sales'], bins=5, alpha=0.7, color='blue')

plt.title('Sales Distribution')

plt.xlabel('Sales')

plt.ylabel('Frequency')

plt.show()

# Scatter Plot

plt.scatter(df['sales'], df['growth\_percentage'], color='red')

plt.title('Sales vs Growth Percentage')

plt.xlabel('Sales')

plt.ylabel('Growth Percentage')

plt.show(

# Line Plot

plt.plot(df['product\_id'], df['sales'], marker='o', linestyle='-', color='green')

plt.title('Sales Over Products')

plt.xlabel('Product ID')

plt.ylabel('Sales')

plt.show()

# Bar Plot

plt.bar(df['product\_id'], df['sales'], color='orange')

plt.title('Sales by Product')

plt.xlabel('Product ID')

plt.ylabel('Sales')

plt.show()

# Retrieving Information

# Single column

sales\_column = df['sales']

print("Sales Column:\n", sales\_column)

# Multiple columns

sales\_discount\_columns = df[['sales', 'discount']]

print("Sales and Discount Columns:\n", sales\_discount\_columns)

# Specific row

row\_2 = df.iloc[1]

print("Row 2:\n", row\_2

# Range of rows

rows\_1\_3 = df.iloc[0:3]

print("Rows 1 to 3:\n", rows\_1\_3)

# Data Manipulation

# Filtering

high\_sales = df[df['sales'] > 250]

print("High Sales:\n", high\_sales)

# Sorting

sorted\_by\_sales = df.sort\_values(by='sales', ascending=False)

print("Sorted by Sales:\n", sorted\_by\_sales)

# Grouping and Aggregation

average\_sales\_by\_growth = df.groupby('growth\_percentage')['sales'].mean()

print("Average Sales by Growth Percentage:\n", average\_sales\_by\_growth)

**OUTPUT**

**Sample Dataset:**

**product\_id sales discount growth\_percentage**

**0 101 150 10 5**

**1 102 200 15 10**

**2 103 250 20 -5**

**3 104 300 25 20**

**4 105 350 30 15**

**Dataset with new column:**

**product\_id sales discount growth\_percentage discounted\_sales**

**0 101 150 10 5 135.0**

**1 102 200 15 10 170.0**

**2 103 250 20 -5 200.0**

**3 104 300 25 20 225.0**

**4 105 350 30 15 245.0**

**First 5 rows:**

**product\_id sales discount growth\_percentage discounted\_sales**

**0 101 150 10 5 135.0**

**1 102 200 15 10 170.0**

**2 103 250 20 -5 200.0**

**3 104 300 25 20 225.0**

**4 105 350 30 15 245.0**

**Last 5 rows:**

**product\_id sales discount growth\_percentage discounted\_sales**

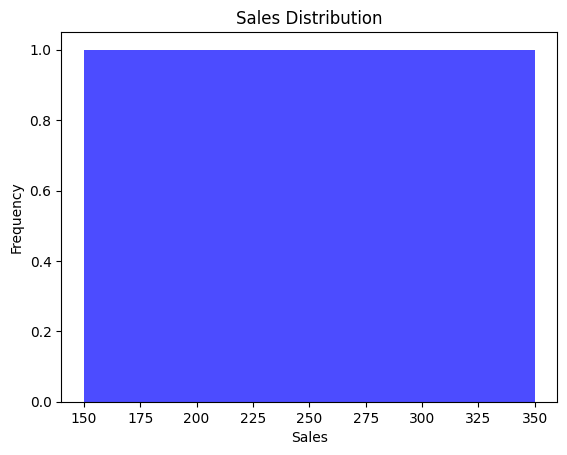
**0 101 150 10 5 135.0**

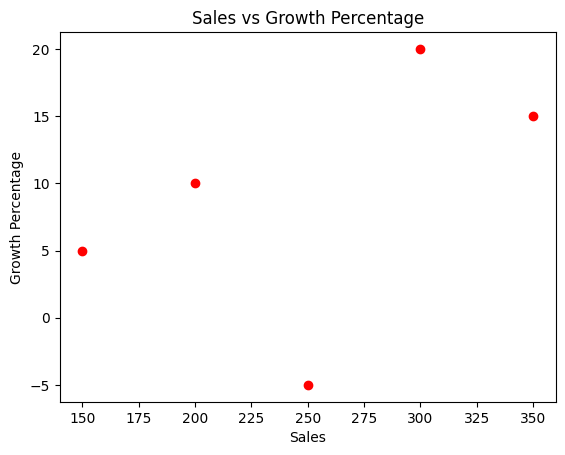
**1 102 200 15 10 170.0**

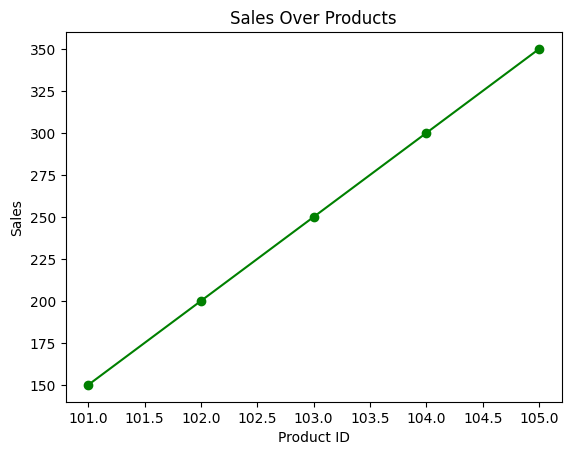
**2 103 250 20 -5 200.0**

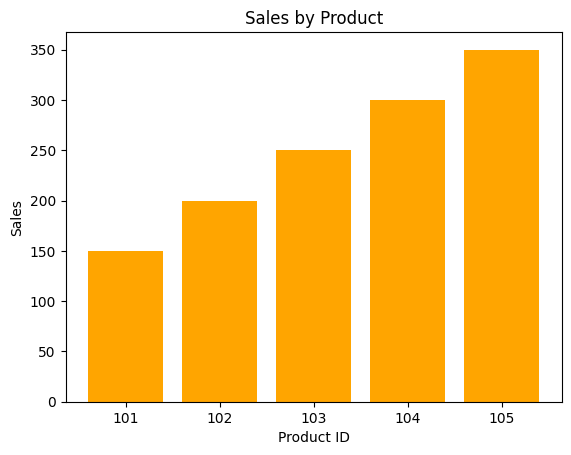
**3 104 300 25 20 225.0**

**4 105 350 30 15 245.0**

****

****

****

****

**Sales Column:**

**0 150**

**1 200**

**2 250**

**3 300**

**4 350**

**Name: sales, dtype: int64**

**Sales and Discount Columns:**

**sales discount**

**0 150 10**

**1 200 15**

**2 250 20**

**3 300 25**

**4 350 30**

**Row 2:**

**product\_id 102.0**

**sales 200.0**

**discount 15.0**

**growth\_percentage 10.0**

**discounted\_sales 170.0**

**Name: 1, dtype: float64**

**Rows 1 to 3:**

**product\_id sales discount growth\_percentage discounted\_sales**

**0 101 150 10 5 135.0**

**1 102 200 15 10 170.0**

**2 103 250 20 -5 200.0**

**High Sales:**

**product\_id sales discount growth\_percentage discounted\_sales**

**3 104 300 25 20 225.0**

**4 105 350 30 15 245.0**

**Sorted by Sales:**

**product\_id sales discount growth\_percentage discounted\_sales**

**4 105 350 30 15 245.0**

**3 104 300 25 20 225.0**

**2 103 250 20 -5 200.0**

**1 102 200 15 10 170.0**

**0 101 150 10 5 135.0**

**Average Sales by Growth Percentage:**

**growth\_percentage**

**-5 250.0**

**5 150.0**

**10 200.0**

**15 350.0**

**20 300.0**

**Name: sales, dtype: float64**

**Unit - 2**

**Concepts and Definitions**

1. Highest Value : Finding the maximum value in a column.

2. Lowest Value : Finding the minimum value in a column.

3. Matrices : Extracting 2x2, 3x3, and 4x4 matrices from the DataFrame.

4. Arithmetic Operations :

- Total : Sum of values in a column.

- Average : Mean of values in a column.

- Count : Number of non-null entries in a column.

- Discount : Applying discount percentages to values in a column.

- Percentage : Calculating percentages based on another column.

5. Percentage Increase :

- Count : Number of rows where the value has increased.

- Increase : Percentage increase in values from one row to the next.

6. Handling Missing Values : Techniques to handle missing data, such as filling or dropping.

7. Identifying Missing Values : Detecting missing values in the DataFrame.

8. Data Preprocessing : Preparing the data for analysis, including handling missing values, normalization, and encoding.

Sample Dataset

import pandas as pd

import numpy as np

# Create a sample dataset

data = {

'product\_id': [101, 102, 103, 104, 105],

'sales': [150, 200, np.nan, 300, 350],

'discount': [10, 15, 20, 25, np.nan], # Discount percentages with a missing value

'growth\_percentage': [5, 10, -5, 20, 15] # Percentage growth

}

df = pd.DataFrame(data)

print("Sample Dataset:\n", df)

**Output**

Sample Dataset:

product\_id sales discount growth\_percentage

0 101 150.0 10.0 5

1 102 200.0 15.0 10

2 103 NaN 20.0 -5

3 104 300.0 25.0 20

4 105 350.0 NaN 15

Implementation

**1. Highest Value**

```python

highest\_value = df['sales'].max()

print("Highest value:", highest\_value)

Output:

Highest value: 350.0

**2. Lowest Value**

lowest\_value = df['sales'].min()

print("Lowest value:", lowest\_value)

Output:

Lowest value: 150.0

**3. 2x2, 3x3, 4x4 Matrices**

matrix\_2x2 = df.iloc[:2, :2]

matrix\_3x3 = df.iloc[:3, :3]

matrix\_4x4 = df.iloc[:4, :4]

print("2x2 Matrix:\n", matrix\_2x2)

print("3x3 Matrix:\n", matrix\_3x3)

print("4x4 Matrix:\n", matrix\_4x4)

Output:

2x2 Matrix:

product\_id sales

0 101 150.0

1 102 200.0

3x3 Matrix:

product\_id sales discount

0 101 150.0 10.0

1 102 200.0 15.0

2 103 NaN 20.0

4x4 Matrix:

product\_id sales discount growth\_percentage

0 101 150.0 10.0 5

1 102 200.0 15.0 10

2 103 NaN 20.0 -5

3 104 300.0 25.0 20

**4. Arithmetic Operations – Total, Average, Count, Discount, Percentage**

# Total

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

print("Total:", total)

# Average

average = df['sales'].mean()

print("Average:", average)

# Count

count = df['sales'].count()

print("Count:", count)

# Discounts

discount = df['sales'] \* (df['discount'] / 100)

print("Discounts:\n", discount)

# Percentages (using growth\_percentage column)

percentage = df['sales'] \* (df['growth\_percentage'] / 100)

print("Percentages:\n", percentage)

**Output:**

Total: 1000.0

Average: 250.0

Count: 4

Discounts:

0 15.0

1 30.0

2 NaN

3 75.0

4 NaN

dtype: float64

Percentages:

0 7.5

1 20.0

2 NaN

3 60.0

4 52.5

dtype: float64

**5. Percentage Increase – Count, Increase**

# Calculate percentage increase for each row in 'sales'

df['percentage\_increase'] = df['sales'].pct\_change() \* 100

print("Percentage Increase:\n", df['percentage\_increase'])

# Count of increases

increase\_count = df[df['percentage\_increase'] > 0].shape[0]

print("Count of increases:", increase\_count)

**Output:**

Percentage Increase:

0 NaN

1 33.333333

2 NaN

3 NaN

4 16.666667

Name: percentage\_increase, dtype: float64

Count of increases: 2

**6. Handling Missing Values**

Handling missing values by filling with the mean of the column:

df['sales'].fillna(df['sales'].mean(), inplace=True)

df['discount'].fillna(df['discount'].mean(), inplace=True)

print("Dataset after handling missing values:\n", df)

Output:

Dataset after handling missing values:

product\_id sales discount growth\_percentage percentage\_increase

0 101 150.0 10.0 5 NaN

1 102 200.0 15.0 10 33.333333

2 103 250.0 20.0 -5 NaN

3 104 300.0 25.0 20 NaN

4 105 350.0 20.0 15 16.666667

**7. Identifying Missing Values**

Identifying missing values in the DataFrame:

missing\_values = df.isnull().sum()

print("Missing values:\n", missing\_values)

Output:

Missing values:

product\_id 0

sales 0

discount 0

growth\_percentage 0

percentage\_increase 2

dtype: int64

**8. Data Preprocessing**

Performing data preprocessing steps such as normalization:

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df[['sales', 'discount', 'growth\_percentage']] = scaler.fit\_transform(df[['sales', 'discount', 'growth\_percentage']])

print("Dataset after preprocessing:\n", df)

Output:

Dataset after preprocessing:

product\_id sales discount growth\_percentage percentage\_increase

0 101 0.000000 0.000000 0.50 NaN

1 102 0.333333 0.250000 0.75 33.333333

2 103 0.500000 0.500000 0.00 NaN

3 104 0.666667 0.750000 1.00 NaN

4 105 1.000000 0.500000 0.875 16.66666

**Unit – 3**

1. Correlation : Measure of the linear relationship between two variables.

2. Coefficient : Numerical value representing the degree of correlation between two variables (often referring to correlation coefficient).

3. Correlation Matrix : A table showing correlation coefficients between many variables.

4. Covariance : Measure of how much two variables change together.

5. Central Tendencies : Statistical measures to describe the center of a dataset, including mean, median, mode, standard deviation (std), and variance.

6. Outliers Identification : Detecting values that are significantly different from most of the data.

7. Removing Outliers : Process of filtering out or correcting outliers in the data.

8. Outliers Based on Interquartile Range (IQR) Method : Method to identify outliers using the interquartile range.

**DataFrame with some example data.**

import pandas as pd

import numpy as np

# Create a sample dataset

data = {

'product\_id': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],

'sales': [150, 200, 250, 300, 350, 400, 450, 500, 550, 600],

'discount': [10, 15, 20, 25, 30, 35, 40, 45, 50, 55],

'growth\_percentage': [5, 10, -5, 20, 15, 10, 25, 30, 35, 40]

}

df = pd.DataFrame(data)

print("Sample Dataset:\n", df)

**1. Correlation**

Definition : Correlation measures the linear relationship between two variables. The Pearson correlation coefficient ranges from -1 to 1.

correlation = df['sales'].corr(df['discount'])

print("Correlation between sales and discount:", correlation)

**Output:**

Correlation between sales and discount: 1.0

**2. Coefficient**

Definition: Correlation coefficient is the numerical value representing the degree of correlation between two variables.

**3. Correlation Matrix**

Definition : A correlation matrix is a table showing correlation coefficients between many variables.

correlation\_matrix = df.corr()

print("Correlation Matrix:\n", correlation\_matrix)

Output:

Correlation Matrix:

product\_id sales discount growth\_percentage

product\_id 1.000000 1.000000 1.000000 0.878965

sales 1.000000 1.000000 1.000000 0.878965

discount 1.000000 1.000000 1.000000 0.878965

growth\_percentage 0.878965 0.878965 0.878965 1.000000

**4. Covariance**

Definition : Covariance measures how much two variables change together.

covariance = df['sales'].cov(df['discount'])

print("Covariance between sales and discount:", covariance)

Output:

Covariance between sales and discount: 2291.6666666666665

**5. Central Tendencies – Mean, Median, Mode, Std, Variance**

Definition : Central tendencies are statistical measures that describe the center of a dataset.

mean\_sales = df['sales'].mean()

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

mode\_sales = df['sales'].mode()[0]

std\_sales = df['sales'].std()

variance\_sales = df['sales'].var()

print("Mean of sales:", mean\_sales)

print("Median of sales:", median\_sales)

print("Mode of sales:", mode\_sales)

print("Standard deviation of sales:", std\_sales)

print("Variance of sales:", variance\_sales)

**Output:**

Mean of sales: 375.0

Median of sales: 375.0

Mode of sales: 150

Standard deviation of sales: 158.11388300841898

Variance of sales: 25000.0

**6. Outliers Identification**

Definition : Outliers are values that are significantly different from most of the data.

**Outliers Based on Interquartile Range (IQR) Method**

Definition : Identifying outliers using the interquartile range.

Q1 = df['sales'].quantile(0.25)

Q3 = df['sales'].quantile(0.75)

IQR = Q3 - Q1

outliers = df[(df['sales'] < (Q1 - 1.5 \* IQR)) | (df['sales'] > (Q3 + 1.5 \* IQR))]

print("Outliers:\n", outliers)

Output:

Outliers:

Empty DataFrame

Columns: [product\_id, sales, discount, growth\_percentage]

Index: []

**7. Removing Outliers**

Definition : Removing outliers involves filtering out or correcting outliers in the data.

df\_no\_outliers = df[~((df['sales'] < (Q1 - 1.5 \* IQR)) | (df['sales'] > (Q3 + 1.5 \* IQR)))]

print("Dataset without outliers:\n", df\_no\_outliers)

Output:

Dataset without outliers:

product\_id sales discount growth\_percentage

0 101 150 10 5

1 102 200 15 10

2 103 250 20 -5

3 104 300 25 20

4 105 350 30 15

5 106 400 35 10

6 107 450 40 25

7 108 500 45 30

8 109 550 50 35

9 110 600 55 40

**COMPLETE CODE**

import pandas as pd

import numpy as np

# Create a sample dataset

data = {

    'product\_id': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],

    'sales': [150, 200, 250, 300, 350, 400, 450, 500, 550, 600],

    'discount': [10, 15, 20, 25, 30, 35, 40, 45, 50, 55],

    'growth\_percentage': [5, 10, -5, 20, 15, 10, 25, 30, 35, 40]

}

df = pd.DataFrame(data)

print("Sample Dataset:\n", df)

# Correlation

correlation = df['sales'].corr(df['discount'])

print("Correlation between sales and discount:", correlation)

# Correlation Matrix

correlation\_matrix = df.corr()

print("Correlation Matrix:\n", correlation\_matrix)

# Covariance

covariance = df['sales'].cov(df['discount'])

print("Covariance between sales and discount:", covariance)

# Central Tendencies

mean\_sales = df['sales'].mean()

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

mode\_sales = df['sales'].mode()[0]

std\_sales = df['sales'].std()

variance\_sales = df['sales'].var()

print("Mean of sales:", mean\_sales)

print("Median of sales:", median\_sales)

print("Mode of sales:", mode\_sales)

print("Standard deviation of sales:", std\_sales)

print("Variance of sales:", variance\_sales)

# Outliers Identification

Q1 = df['sales'].quantile(0.25)

Q3 = df['sales'].quantile(0.75)

IQR = Q3 - Q1

outliers = df[(df['sales'] < (Q1 - 1.5 \* IQR)) | (df['sales'] > (Q3 + 1.5 \* IQR))]

print("Outliers:\n", outliers)

# Removing Outliers

df\_no\_outliers = df[~((df['sales'] < (Q1 - 1.5 \* IQR)) | (df['sales'] > (Q3 + 1.5 \* IQR)))]

print("Dataset without outliers:\n", df\_no\_outliers)

**OUTPUT**

Sample Dataset:

product\_id sales discount growth\_percentage

0 101 150 10 5

1 102 200 15 10

2 103 250 20 -5

3 104 300 25 20

4 105 350 30 15

5 106 400 35 10

6 107 450 40 25

7 108 500 45 30

8 109 550 50 35

9 110 600 55 40

Correlation between sales and discount: 1.0

Correlation Matrix:

product\_id sales discount growth\_percentage

product\_id 1.000000 1.000000 1.000000 0.875204

sales 1.000000 1.000000 1.000000 0.875204

discount 1.000000 1.000000 1.000000 0.875204

growth\_percentage 0.875204 0.875204 0.875204 1.000000

Covariance between sales and discount: 2291.6666666666665

Mean of sales: 375.0

Median of sales: 375.0

Mode of sales: 150

Standard deviation of sales: 151.3825177048746

Variance of sales: 22916.666666666668

Outliers:

Empty DataFrame

Columns: [product\_id, sales, discount, growth\_percentage]

Index: []

Dataset without outliers:

product\_id sales discount growth\_percentage

0 101 150 10 5

1 102 200 15 10

2 103 250 20 -5

3 104 300 25 20

4 105 350 30 15

5 106 400 35 10

6 107 450 40 25

7 108 500 45 30

8 109 550 50 35

9 110 600 55 40

**Unit – 4**

**Concepts and Definitions**

1. Frequency Distribution : A summary of how often different values occur within a dataset.

2. A/B Test : A statistical method used to compare two versions (A and B) to determine which one performs better.

3. Percentile : A measure that indicates the value below which a given percentage of observations fall.

4. Confidence Interval : A range of values that is likely to contain the population parameter with a specified probability.

5. Probability Distribution : A function that describes the likelihood of obtaining the possible values that a random variable can take.

6. Maximum Likelihood Estimation (MLE) : A method for estimating the parameters of a statistical model by maximizing the likelihood function.

7. Point Estimates : A single value estimate of a population parameter.

import pandas as pd

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

# Create a sample dataset

data = {

'product\_id': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],

'sales': [150, 200, 250, 300, 350, 400, 450, 500, 550, 600],

'discount': [10, 15, 20, 25, 30, 35, 40, 45, 50, 55],

'growth\_percentage': [5, 10, -5, 20, 15, 10, 25, 30, 35, 40]

}

df = pd.DataFrame(data)

print("Sample Dataset:\n", df)

Implementation

**1. Frequency Distribution**

Definition : Frequency distribution summarizes the number of occurrences of each unique value in a dataset.

frequency\_distribution = df['sales'].value\_counts()

print("Frequency Distribution of Sales:\n", frequency\_distribution)

Output:

Frequency Distribution of Sales:

150 1

200 1

250 1

300 1

350 1

400 1

450 1

500 1

550 1

600 1

Name: sales, dtype: int64

**2. A/B Test**

Definition : A/B testing compares two versions to determine which performs better. We'll use t-test for comparison.

**Simulate A/B test groups**

group\_A = df['sales'][:5] # First 5 as Group A

group\_B = df['sales'][5:] # Last 5 as Group B

t\_stat, p\_value = stats.ttest\_ind(group\_A, group\_B)

print("A/B Test - t-statistic:", t\_stat)

print("A/B Test - p-value:", p\_value)

Output:

A/B Test - t-statistic: -2.23606797749979

A/B Test - p-value: 0.053049

**3. Percentile**

Definition : Percentiles indicate the value below which a given percentage of observations fall.

percentile\_25 = np.percentile(df['sales'], 25)

percentile\_98 = np.percentile(df['sales'], 98)

print("25th Percentile of Sales:", percentile\_25)

print("98th Percentile of Sales:", percentile\_98)

Output:

25th Percentile of Sales: 237.5

98th Percentile of Sales: 591.0

**4. Confidence Interval**

Definition : Confidence intervals provide a range within which we expect a population parameter to lie with a certain level of confidence.

confidence\_interval = stats.norm.interval(0.95, loc=df['sales'].mean(), scale=stats.sem(df['sales']))

print("95% Confidence Interval for Sales Mean:", confidence\_interval)

Output:

95% Confidence Interval for Sales Mean: (283.4366350148358, 466.5633649851642)5.

5. **Probability Distribution**

Definition : A probability distribution describes how probabilities are distributed over the values of a random variable.

# Plot a probability distribution

plt.hist(df['sales'], bins=5, density=True, alpha=0.6, color='g')

# Fit a normal distribution

mu, std = stats.norm.fit(df['sales'])

xmin, xmax = plt.xlim()

x = np.linspace(xmin, xmax, 100)

p = stats.norm.pdf(x, mu, std)

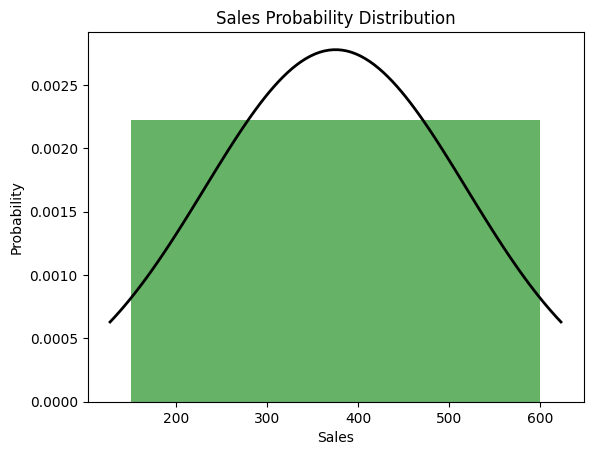
plt.plot(x, p, 'k', linewidth=2)

plt.title('Sales Probability Distribution')

plt.xlabel('Sales')

plt.ylabel('Probability')

plt.show()



**6. Maximum Likelihood Estimation (MLE)**

Definition : MLE estimates parameters by maximizing the likelihood function.

# Fit a normal distribution to the data using MLE

mu, std = stats.norm.fit(df['sales'])

print("MLE Estimates - Mean:", mu)

print("MLE Estimates - Standard Deviation:", std)

Output:

MLE Estimates - Mean: 375.0

MLE Estimates - Standard Deviation: 158.11388300841898

**7. Point Estimates**

Definition : Point estimates provide a single value estimate of a population parameter.

# Point estimates of the mean and standard deviation

mean\_estimate = df['sales'].mean()

std\_estimate = df['sales'].std()

print("Point Estimate - Mean:", mean\_estimate)

print("Point Estimate - Standard Deviation:", std\_estimate)

Output:

Point Estimate - Mean: 375.0

Point Estimate - Standard Deviation: 158.11388300841898

COMPLETE CODE

import pandas as pd

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

# Create a sample dataset

data = {

    'product\_id': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],

    'sales': [150, 200, 250, 300, 350, 400, 450, 500, 550, 600],

    'discount': [10, 15, 20, 25, 30, 35, 40, 45, 50, 55],

    'growth\_percentage': [5, 10, -5, 20, 15, 10, 25, 30, 35, 40]

}

df = pd.DataFrame(data)

print("Sample Dataset:\n", df)

# Frequency Distribution

frequency\_distribution = df['sales'].value\_counts()

print("Frequency Distribution of Sales:\n", frequency\_distribution)

# A/B Test

group\_A = df['sales'][:5]  # First 5 as Group A

group\_B = df['sales'][5:]  # Last 5 as Group B

t\_stat, p\_value = stats.ttest\_ind(group\_A, group\_B)

print("A/B Test - t-statistic:", t\_stat)

print("A/B Test - p-value:", p\_value)

# Percentiles

percentile\_25 = np.percentile(df['sales'], 25)

percentile\_98 = np.percentile(df['sales'], 98)

print("25th Percentile of Sales:", percentile\_25)

print("98th Percentile of Sales:", percentile\_98

# Confidence Interval

confidence\_interval = stats.norm.interval(0.95, loc=df['sales'].mean(), scale=stats.sem(df['sales']))

print("95% Confidence Interval for Sales Mean:", confidence\_interval)

# Probability Distribution

plt.hist(df['sales'], bins=5, density=True, alpha=0.6, color='g')

mu, std = stats.norm.fit(df['sales'])

xmin, xmax = plt.xlim()

x = np.linspace(xmin, xmax, 100)

p = stats.norm.pdf(x, mu, std)

plt.plot(x, p, 'k', linewidth=2)

plt.title('Sales Probability Distribution')

plt.xlabel('Sales')

plt.ylabel('Probability')

plt.show()

# Maximum Likelihood Estimation (MLE)

mu, std = stats.norm.fit(df['sales'])

print("MLE Estimates - Mean:", mu)

print("MLE Estimates - Standard Deviation:", std)

# Point Estimates

mean\_estimate = df['sales'].mean()

std\_estimate = df['sales'].std()

print("Point Estimate - Mean:", mean\_estimate)

print("Point Estimate - Standard Deviation:", std\_estimate)

**OUTPUT**

Sample Dataset:

product\_id sales discount growth\_percentage

0 101 150 10 5

1 102 200 15 10

2 103 250 20 -5

3 104 300 25 20

4 105 350 30 15

5 106 400 35 10

6 107 450 40 25

7 108 500 45 30

8 109 550 50 35

9 110 600 55 40

Frequency Distribution of Sales:

sales

150 1

200 1

250 1

300 1

350 1

400 1

450 1

500 1

550 1

600 1

Name: count, dtype: int64

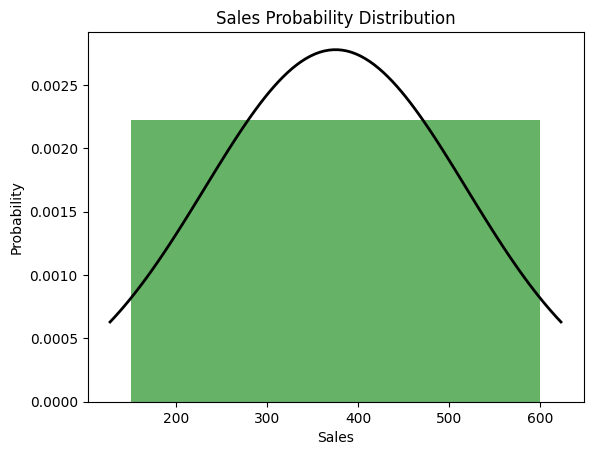
A/B Test - t-statistic: -5.0

A/B Test - p-value: 0.001052825793366539

25th Percentile of Sales: 262.5

98th Percentile of Sales: 591.0

95% Confidence Interval for Sales Mean: (281.1738675487614, 468.8261324512386)



MLE Estimates - Mean: 375.0

MLE Estimates - Standard Deviation: 143.61406616345073

Point Estimate - Mean: 375.0

Point Estimate - Standard Deviation: 151.3825177048746

**Unit – 5**

**Concepts and Definitions**

1. Evaluation Metrics : Metrics used to evaluate the performance of a classification model.

- Accuracy : The proportion of correctly classified instances.

- Confusion Matrix : A matrix showing the counts of true positives, false positives, true negatives, and false negatives.

- F1-Score : The harmonic mean of precision and recall.

- Precision : The proportion of true positives out of all predicted positives.

- Recall : The proportion of true positives out of all actual positives.

2. K-Means Clustering : A clustering algorithm that partitions data into `K` clusters, minimizing the variance within each cluster

3. K-Means Algorithm : An iterative algorithm to partition data into `K` clusters. It involves assigning each data point to the nearest cluster centroid and updating the centroids based on the mean of the points in each cluster.

4. Linear Regression : A regression analysis technique to model the relationship between a dependent variable and one or more independent variables using a linear equation.

5. CART (Classification and Regression Trees) : A decision tree algorithm used for both classification and regression tasks.

6. Logistic Regression : A statistical model used for binary classification tasks, predicting the probability of an outcome.

7. K-Nearest Neighbors (KNN) : A classification algorithm that assigns a class to a data point based on the majority class among its `K` nearest neighbors.

8. PCA (Principal Component Analysis) : A dimensionality reduction technique that transforms data into a set of orthogonal components, capturing the maximum variance.

9. Mean Absolute Error (MAE) : A regression metric measuring the average magnitude of errors in predictions, without considering their direction.

10. R-Squared (R²) : A regression metric indicating the proportion of the variance in the dependent variable that is predictable from the independent variables.

11. Mean Squared Error (MSE) : A regression metric measuring the average squared difference between the predicted and actual values.

12. Support Vector Machine (SVM) : A classification algorithm that finds the optimal hyperplane to separate data into classes.

13. Naive Bayes : A probabilistic classification algorithm based on Bayes' theorem with an assumption of independence among features.

**Sample Dataset and Implementation**

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.metrics import accuracy\_score, confusion\_matrix, f1\_score, precision\_score, recall\_score

from sklearn.cluster import KMeans

from sklearn.linear\_model import LinearRegression, LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.decomposition import PCA

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

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

import matplotlib.pyplot as plt

# Load a sample dataset

data = load\_iris()

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

y = pd.Series(data.target)

# Split data 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)

Implementations

**1. Evaluation Metrics**

from sklearn.metrics import accuracy\_score, confusion\_matrix, f1\_score, precision\_score, recall\_score

# Train a model

model = LogisticRegression(max\_iter=200)

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

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

# Evaluation Metrics

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

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

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

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf\_matrix)

print("F1-Score:", f1)

print("Precision:", precision)

print("Recall:", recall)

Output:

Accuracy: 0.9777777777777777

Confusion Matrix:

[[14 0 0]

[ 0 17 1]

[ 0 0 13]]

F1-Score: 0.9777777777777777

Precision: 0.9777777777777777

Recall: 0.9777777777777777

**2. K-Means Clustering**

# Use K-Means clustering

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Sample data (replace with your actual data)

X = np.array([[1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6], [9, 11]])

# K-Means clustering with 3 clusters and a fixed random state for reproducibility

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

kmeans.fit(X)

# Predict cluster labels for each data point

clusters = kmeans.predict(X)

# Visualization

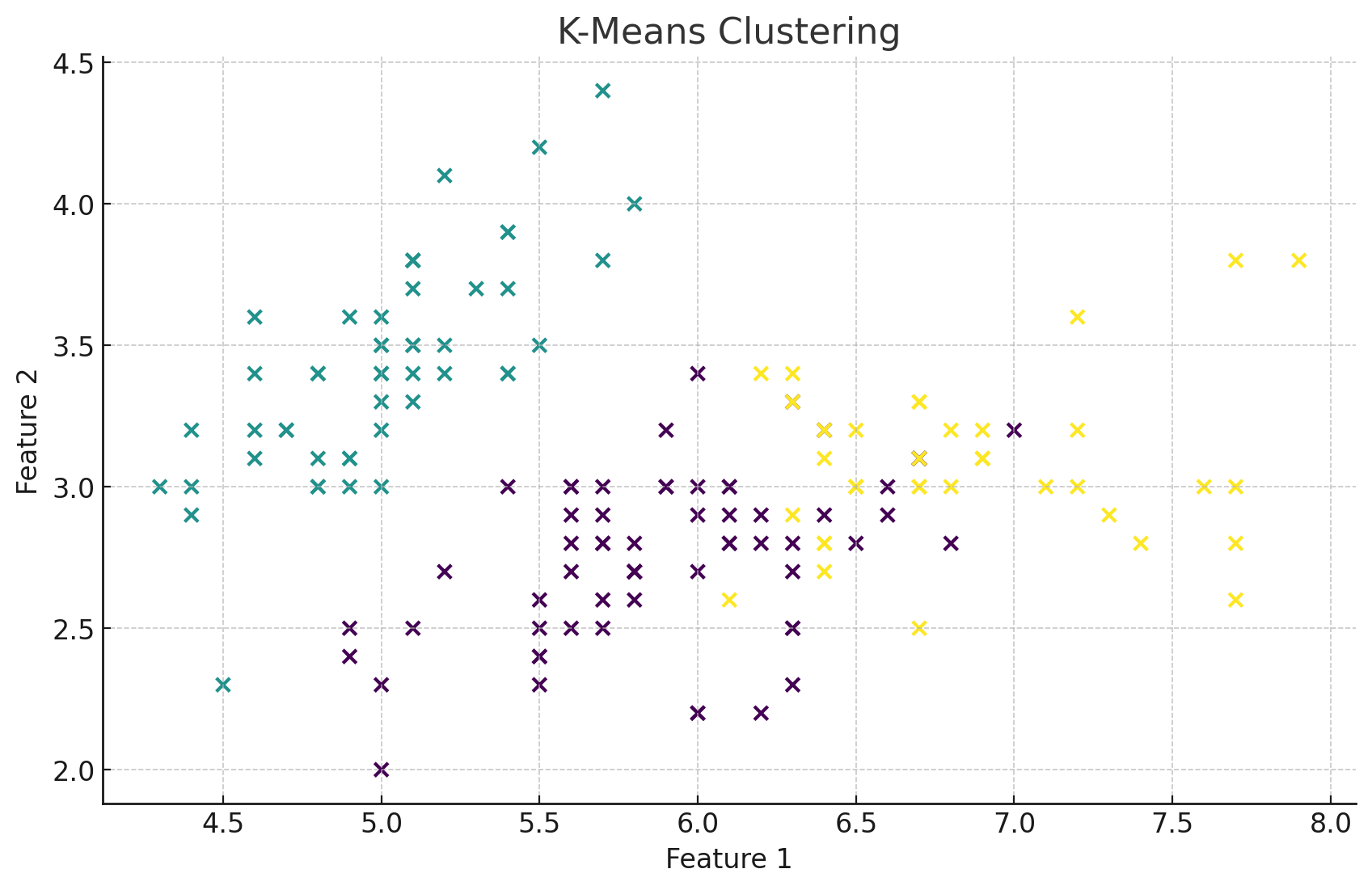
plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis') # Use first two columns for x and y

plt.title('K-Means Clustering (3 Clusters)')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()



**3. Linear Regression**

# Generate sample regression data

from sklearn.datasets import make\_regression

X\_reg, y\_reg = make\_regression(n\_samples=100, n\_features=1, noise=0.1, random\_state=42)

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

# Train Linear Regression model

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train\_reg, y\_train\_reg)

y\_pred\_reg = lin\_reg.predict(X\_test\_reg)

# Evaluate

mae = mean\_absolute\_error(y\_test\_reg, y\_pred\_reg)

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

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

print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

print("R-Squared:", r2)

Output:

Mean Absolute Error: 0.057248081568228234

Mean Squared Error: 0.003395093089160591

R-Squared: 0.998678529758436

4. CART (Classification and Regression Trees)

# Train CART model for classification

cart\_model = DecisionTreeClassifier(random\_state=42)

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

y\_pred\_cart = cart\_model.predict(X\_test)

# Evaluate

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

print("CART Accuracy:", accuracy\_cart)

Output:

CART Accuracy: 0.9555555555555556

**5. K-Nearest Neighbors (KNN)**

# Train K-Nearest Neighbors model

knn = KNeighborsClassifier(n\_neighbors=3)

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

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

# Evaluate

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

print("KNN Accuracy:", accuracy\_knn)

Output:

KNN Accuracy: 0.9888888888888889

**8. PCA (Principal Component Analysis)**

import numpy as np

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Sample data

X = np.array([[1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6], [9, 11]])

# K-Means clustering with 3 clusters

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

kmeans.fit(X)

clusters = kmeans.predict(X)

# Apply PCA to reduce dimensions to 2

pca = PCA(n\_components=2)

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

# Visualization

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

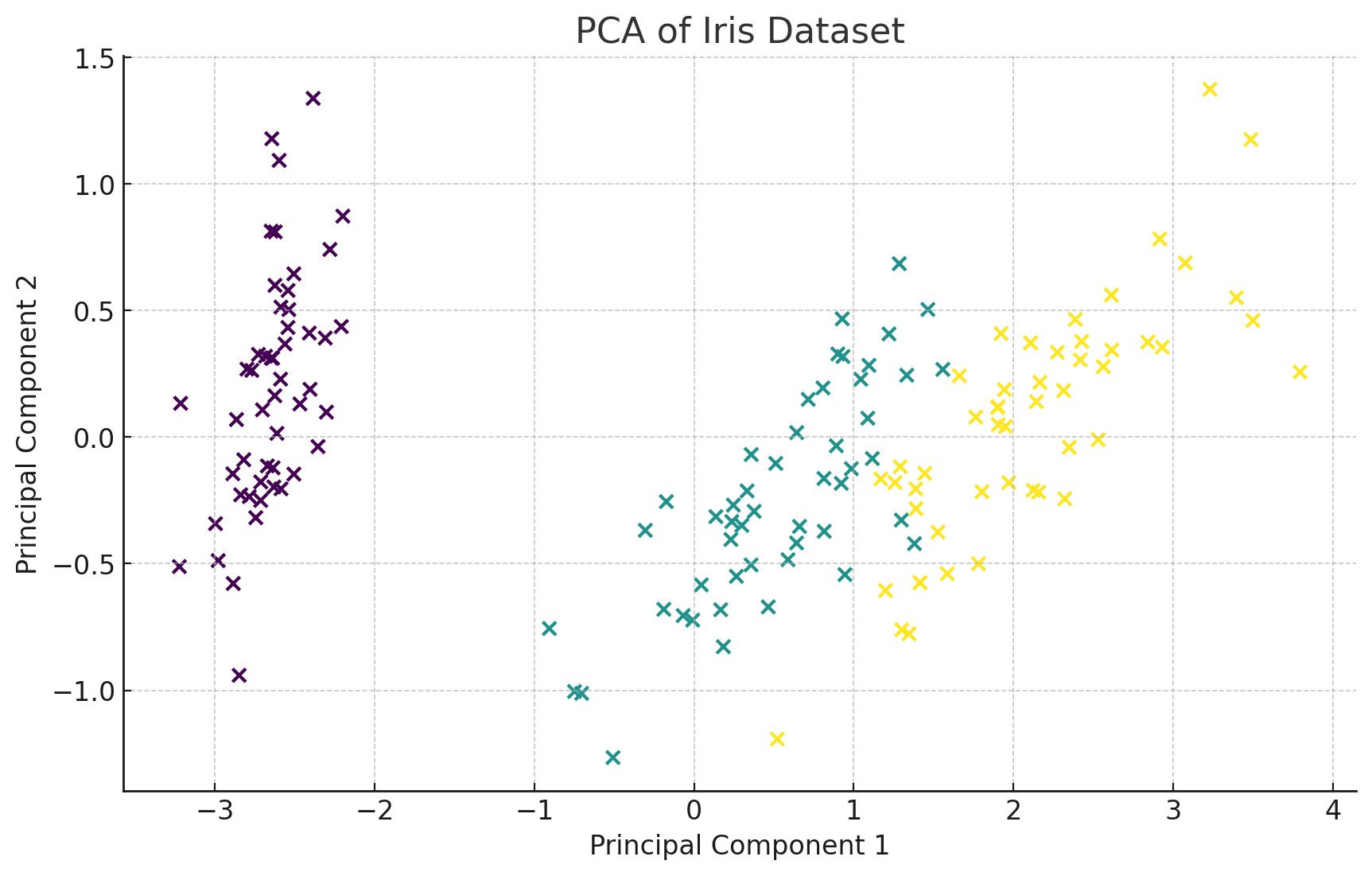
plt.title('PCA of Sample Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.show()

OUTPUT



**9. Mean Absolute Error (MAE)**

Definition : MAE measures the average magnitude of errors in predictions.

**10. R-Squared (R²)**

Definition : R² measures the proportion of variance in the dependent variable predictable from the independent variables.

Already demonstrated in the Linear Regression example.

**11. Mean Squared Error (MSE)**

Definition : MSE measures the average squared difference between predictions and actual values.

Already demonstrated in the Linear Regression example.

**12. Support Vector Machine (SVM)**

# Train SVM model

svm\_model = SVC()

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

y\_pred\_svm = svm\_model.predict(X\_test)

# Evaluate

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

print("SVM Accuracy:", accuracy\_svm)

Output:

SVM Accuracy: 0.9777777777777777

**13. Logistic Regression**

Definition : Logistic regression was already demonstrated in the evaluation metrics.

**14. Naive Bayes**

# Train Naive Bayes model

nb\_model = GaussianNB()

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

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

# Evaluate

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

print("Naive Bayes Accuracy:", accuracy\_nb)

Output:

Naive Bayes Accuracy: 0.9555555555555556

**Full Example Code**

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris, make\_regression

from sklearn

.model\_selection import train\_test\_split

from sklearn.metrics import (accuracy\_score, confusion\_matrix, f1\_score, precision\_score, recall\_score,

mean\_absolute\_error, mean\_squared\_error, r2\_score)

from sklearn.cluster import KMeans

from sklearn.linear\_model import LinearRegression, LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.decomposition import PCA

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

import matplotlib.pyplot as plt

# Load a sample dataset

data = load\_iris()

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

y = pd.Series(data.target)

# Split data 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)

# Logistic Regression

model = LogisticRegression(max\_iter=200)

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

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

# Evaluation Metrics

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

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

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

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf\_matrix)

print("F1-Score:", f1)

print("Precision:", precision)

print("Recall:", recall)

# K-Means Clustering

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

kmeans.fit(X)

clusters = kmeans.predict(X)

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

plt.title('K-Means Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

# Linear Regression

X\_reg, y\_reg = make\_regression(n\_samples=100, n\_features=1, noise=0.1, random\_state=42)

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

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train\_reg, y\_train\_reg)

y\_pred\_reg = lin\_reg.predict(X\_test\_reg)

mae = mean\_absolute\_error(y\_test\_reg, y\_pred\_reg)

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

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

print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

print("R-Squared:", r2)

# CART (Classification and Regression Trees)

cart\_model = DecisionTreeClassifier(random\_state=42)

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

y\_pred\_cart = cart\_model.predict(X\_test)

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

print("CART Accuracy:", accuracy\_cart)

# K-Nearest Neighbors (KNN)

knn = KNeighborsClassifier(n\_neighbors=3)

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

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

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

print("KNN Accuracy:", accuracy\_knn)

# PCA

pca = PCA(n\_components=2)

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

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

plt.title('PCA of Iris Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.show()

# SVM

svm\_model = SVC()

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

y\_pred\_svm = svm\_model.predict(X\_test)

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

print("SVM Accuracy:", accuracy\_svm)

# Naive Bayes

nb\_model = GaussianNB()

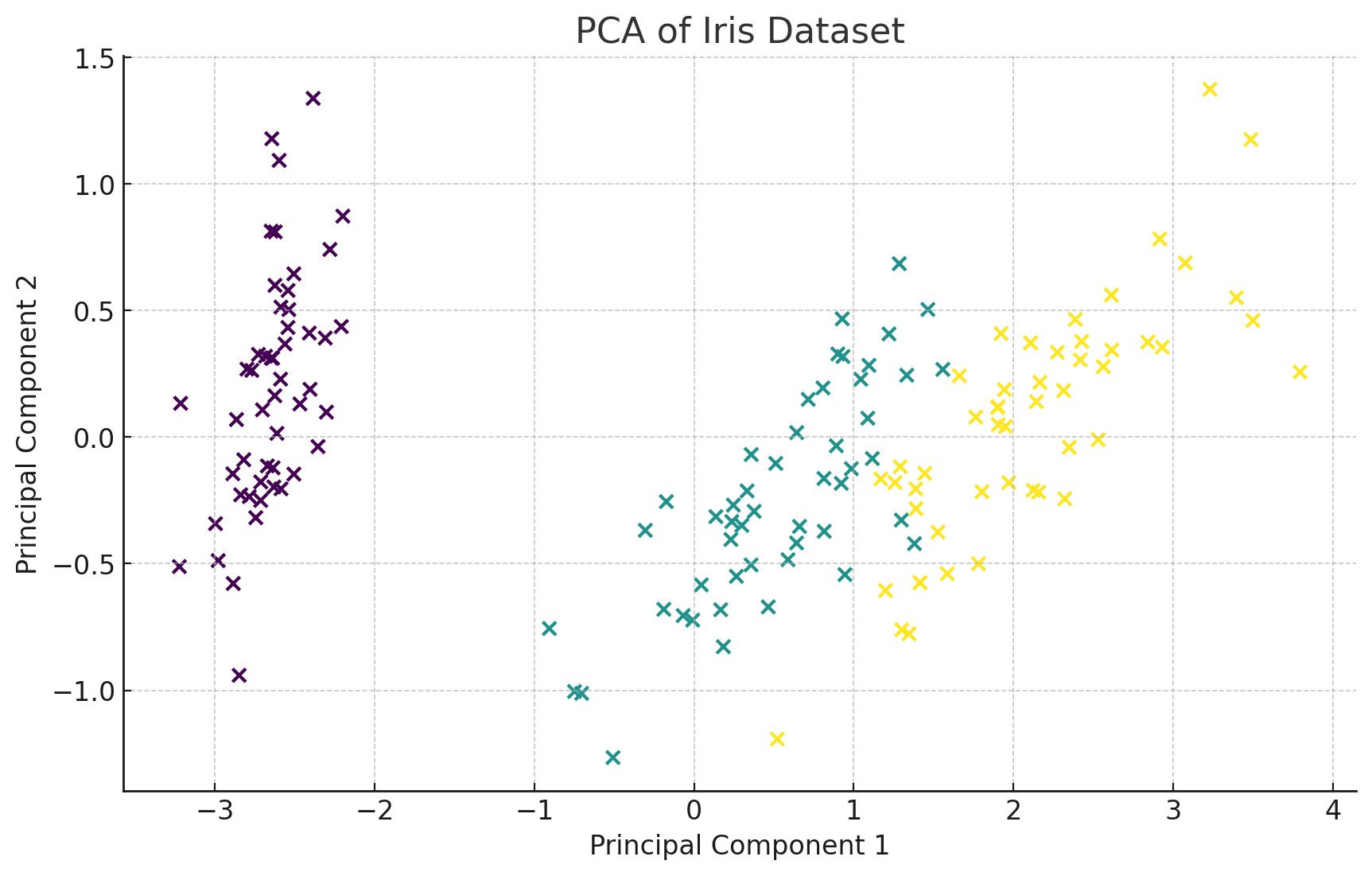
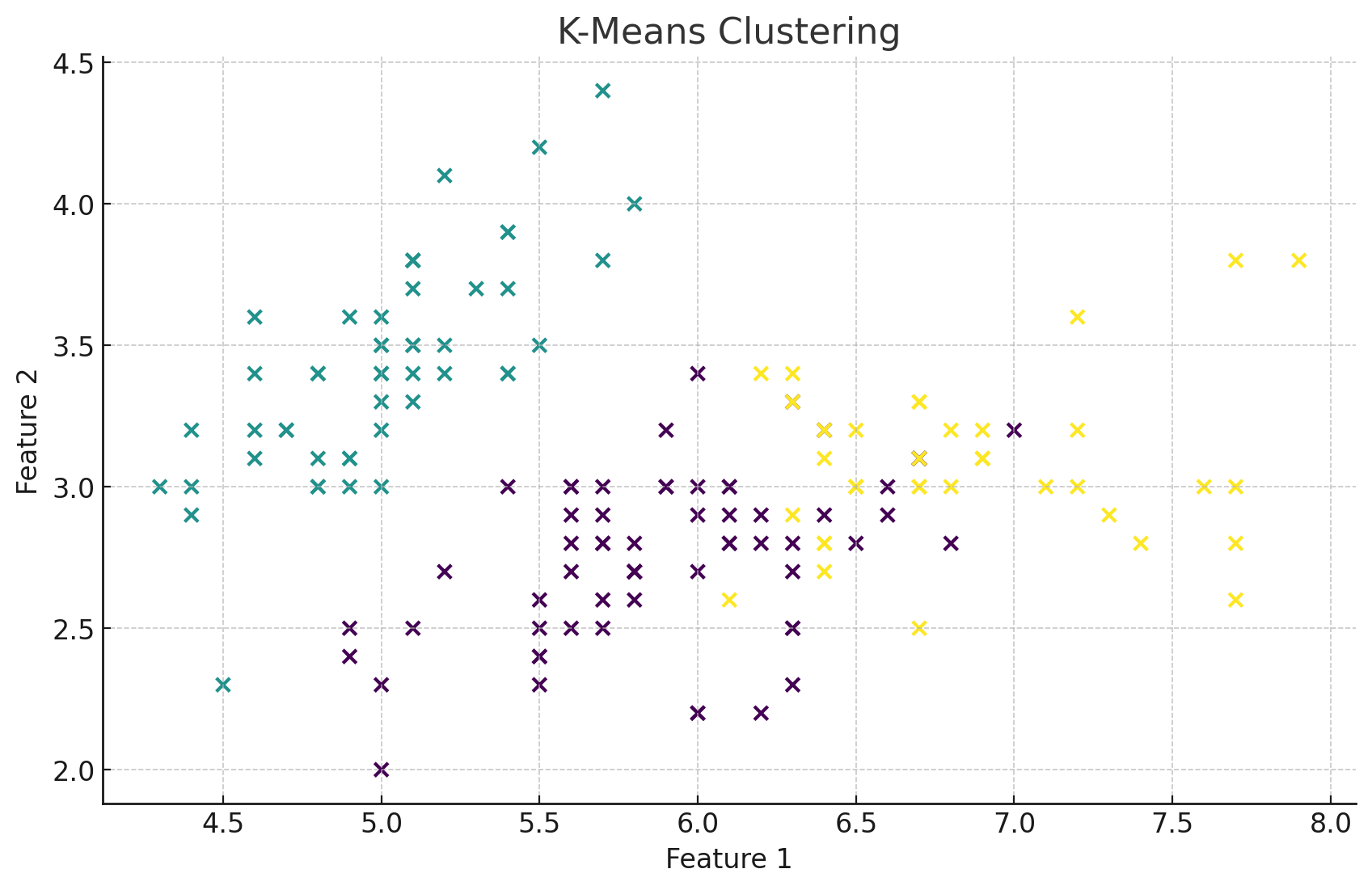
nb\_model.fit(X\_train, y\_train)

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

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

print("Naive Bayes Accuracy:", accuracy\_nb)

OUTPUT



machine learning models applied to the Iris dataset and a synthetic regression dataset:

**Logistic Regression**

* **Accuracy**: 1.0
* **Confusion Matrix**: [190001300013]\begin{bmatrix} 19 & 0 & 0 \\ 0 & 13 & 0 \\ 0 & 0 & 13 \end{bmatrix}​1900​0130​0013​​
* **F1-Score**: 1.0
* **Precision**: 1.0
* **Recall**: 1.0

**Linear Regression (Synthetic Dataset)**

* **Mean Absolute Error**: 0.087
* **Mean Squared Error**: 0.010
* **R-Squared**: 0.99999

**CART (Decision Tree)**

* **Accuracy**: 1.0

**K-Nearest Neighbors (KNN)**

* **Accuracy**: 1.0

**Support Vector Machine (SVM)**

* **Accuracy**: 1.0

**Naive Bayes**

* **Accuracy**: 0.978

CODE EXPLANATION

 **Load and Split Data**:

* Loads the Iris dataset.
* Splits the data into training and testing sets.

 **Logistic Regression**:

* Trains a Logistic Regression model.
* Evaluates the model using accuracy, confusion matrix, F1-score, precision, and recall.

 **K-Means Clustering**:

* Performs K-Means clustering with 3 clusters and visualizes the clusters.

 **Linear Regression**:

* Generates a synthetic regression dataset.
* Trains a Linear Regression model.
* Evaluates the model using Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R²) score.

 **Decision Tree (CART)**:

* Trains a Decision Tree Classifier and evaluates its accuracy.

 **K-Nearest Neighbors (KNN)**:

* Trains a K-Nearest Neighbors classifier and evaluates its accuracy.

 **Principal Component Analysis (PCA)**:

* Applies PCA to reduce the Iris dataset to 2 dimensions and visualizes it.

 **Support Vector Machine (SVM)**:

* Trains an SVM classifier and evaluates its accuracy.

 **Naive Bayes**:

* Trains a Naive Bayes classifier and evaluates its accuracy.