ಕಿಷ್ಕಿಂದ ವಿಶ್ವವಿದ್ಯಾಲಯ

KISHKINDA UNIVERSITY

(A State Private University Established by the Karnataka Act No. 20 of 2023)

Advancing Knowledge Transforming Live

Department of

Computer Science & Engineering

KISHKINDA UNIVERSITY

(A State Private University Established by the Karnataka Act No. 20 of 2023)

Faculty of Engineering & Technology

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LABORATORY MANUAL

PYTHON FOR DATA VISUALIZATION AND ANALYSIS

BE V Semester: 2024-25

Signature of Course Coordinator **Signature of HOD**

FACULTY OF ENGINEERING & TECHNOLOGY

Vision of the University

To develop an ethical and a competent global workforce with an inquisitive mind.

Mission of the University

Mission 1: To equip the students with professional skills, ethical values and competency.

Mission 2: To offer state of the art programs by collaborating with industry, academia and society.

Mission 3: To undertake collaborative & inter disciplinary research for overall development.

Mission 4: To develop leaders and entrepreneurs for the real and sustainable world

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision of the department

To be a leading center of excellence in computer science education and research, dedicated to producing innovative, ethical, managerial and inquisitive professionals capable of addressing global challenges.

Mission of the department

Mission 1: Offer a cutting-edge curriculum that equips students with the knowledge and skills to excel in the rapidly evolving field of computer science.

Mission 2: Foster a vibrant research environment that encourages collaboration and breakthroughs in technology, driving advancements that address global challenges.

Mission 3: Equip students with a strong ethical and managerial foundation, ensuring they become responsible technologists dedicated to societal and environmental well-being.

Program educational objectives

After 3-5 years of graduation, the graduates will be able to

PEO1: Graduates will create innovative solutions to global challenges using their computer science expertise.

PEO2: Graduates will lead with integrity, making decisions that benefit society and the environment.

PEO3: Graduates will pursue ongoing learning and research to stay at the forefront of technological advancements.

Program outcomes

Engineering graduates will be able to:

PO1: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Demonstrate knowledge understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

At the end of 4 years, Computer Science and Engineering graduates will be able to:

PSO1: Develop proficiency in applying programming principles, algorithms, and modern software engineering practices to design, develop, and deploy robust software solutions across various platforms.

PSO2: Cultivate skills in employing data analysis techniques, machine learning models, and AI tools to extract insights, drive innovation, and support decision-making in real-world applications.

PSO3: Acquire proficiency to design, implement, and manage complex computer-based systems, ensuring seamless integration of hardware and software components to meet the specified needs.

TABLE OF CONTENTS

Course Outcomes:

- Solve simple problems using python data structures, decision making and iterative statements.
- Efficiently analyze data using NumPy for real-world applications.
- Manage time series, missing data, and hierarchical indexing using Pandas.
- Create effective visualizations with Matplotlib and Seaborn to communicate data insights.
- Apply learned techniques in hands-on mini projects with real-world datasets.

Program 1

Write a python program for NumPy Array Creation and Aggregations

Aim

To write a Python program for creating NumPy arrays and performing aggregation operations such as sum, mean, min, max, and standard deviation.

Objectives

1. To understand the concept of NumPy arrays.
2. To learn different methods of creating NumPy arrays.
3. To perform aggregation operations like sum, mean, min, max, and standard deviation.
4. To analyze results of array computations efficiently using NumPy.

1. Array Creation

NumPy arrays can be created in several ways:

- **From Python Lists/Tuples:**
  Using `np.array()` to convert lists or tuples into arrays.
  Example: `np.array([1, 2, 3])`
- **Using Range Functions:**
  - `np.arange(start, stop, step)` → Creates evenly spaced values within a range.
  - `np.linspace(start, stop, num)` → Creates evenly spaced values between start and stop with fixed number of elements.
- **Special Arrays:**
  - `np.zeros(shape)` → Creates an array filled with zeros.
  - `np.ones(shape)` → Creates an array filled with ones.
  - `np.eye(n)` → Creates an identity matrix of size `n x n`.
  - `np.random.rand()` → Generates arrays with random values.

2. Aggregation Functions

Aggregation refers to performing mathematical/statistical operations across array elements. NumPy provides optimized functions that work efficiently on large datasets.

- **Summation (`np.sum`)** → Adds all elements in the array.
- **Mean (`np.mean`)** → Calculates average of elements.
- **Minimum & Maximum (`np.min`, `np.max`)** → Finds smallest and largest element.

- **Standard Deviation (`np.std`)** → Measures the spread of data from the mean.
- **Other Functions:**
  - `np.median()` → Finds median value.
  - `np.prod()` → Product of all elements.
  - `np.cumsum()` → Cumulative sum of elements.

1)Write a python program for NumPy Array Creation and Aggregations

```
import numpy as np

# Create a 1D array

arr = np.array([2, 5, 8, 1, 9, 3, 7])

print("Original 1D array:", arr)

# Create a 2D array

matrix = np.array([[10, 15, 20],

                   [25, 30, 35],

                   [40, 45, 50]])

print("\nOriginal 2D matrix:\n", matrix)

# Perform aggregations

print("\n--- Aggregations ---")

print("Sum of all elements:", np.sum(arr))

print("Mean of all elements:", np.mean(arr))

print("Minimum of the matrix:", np.min(matrix))

print("Maximum of each column:", np.max(matrix, axis=0))

print("Standard deviation of each row:", np.std(matrix, axis=1))
```

output

```
Original 1D array: [2 5 8 1 9 3 7]
Original 2D matrix:
 [[10 15 20]
 [25 30 35]
 [40 45 50]]

--- Aggregations ---
Sum of all elements: 35
Mean of all elements: 5.0
Minimum of the matrix: 10
Maximum of each column: [40 45 50]
Standard deviation of each row: [4.0824829 4.0824829 4.0824829]
```

Program 2

Write a python program for Broadcasting and Boolean Masking

Aim

To write a Python program demonstrating **Broadcasting** and **Boolean Masking** operations using NumPy arrays.

Objectives

1. To understand the concept of **broadcasting** in NumPy arrays.
2. To apply **boolean masking** for filtering array elements.
3. To practice vectorized operations without explicit loops.
4. To analyze how broadcasting and masking improve **efficiency in numerical computations**.

Theory

1. Broadcasting

- **Definition:** Broadcasting in NumPy allows arrays of different shapes to be combined in arithmetic operations.
- Instead of explicitly replicating smaller arrays, NumPy **automatically expands** them to match the shape of the larger array.
- This reduces **memory usage** and improves **execution speed**.

Broadcasting Rules:

1. If two arrays differ in dimensions, the smaller array is **stretched** to match.
2. If one dimension has size `1`, it can be repeated to match the other array.
3. If dimensions are not compatible (and cannot be broadcasted), NumPy raises an error

2) **Write a python program for Broadcasting and Boolean Masking**

This program demonstrates broadcasting to perform operations on arrays of different shapes and uses boolean masking for filtering data.

```
import numpy as np

# Create a 2D array

data = np.random.randint(50, 100, size=(5, 4))

print("Original data (5x4):\n", data)

# Broadcasting: Add a 1D array to each row of the 2D array

# The 1D array is "broadcast" across the rows

adjustment = np.array([5, -2, 0, 3])

adjusted_data = data + adjustment

print("\nData after broadcasting:\n", adjusted_data)

# Boolean Masking: Find values greater than 80

high_values_mask = data > 80

print("\nBoolean mask (values > 80):\n", high_values_mask)

# Apply the mask to filter the data

high_values = data[high_values_mask]

print("\nValues from the original data that are > 80:", high_values)
```

output

```
Data after broadcasting:
 [[104  87  70  72]
 [ 59  72  59  94]
 [ 98  60  60  79]
 [ 62  91  75  59]
 [ 72  66  78  77]]

Boolean mask (values > 80):
 [[ True  True False False]
 [False False False  True]
 [ True False False False]
 [False  True False False]
 [False False False False]]

Values from the original data that are > 80: [99 89 91 93 93]
```

Program 3

Write a python program for Data Indexing and Handling Missing Data

Aim

To study and implement data indexing operations and techniques for handling missing data using the Python Pandas library.

Objectives

1. To learn how to index and select data in a Pandas DataFrame.
2. To understand methods for detecting missing values.
3. To explore strategies for handling missing data (removal, filling, interpolation).
4. To demonstrate practical examples of indexing and missing data handling.

Theory

Data Indexing in Pandas:

- Pandas provides powerful indexing capabilities using .loc[], .iloc[], and direct column selection.
- loc[] → Label-based indexing (select rows/columns by labels).
- iloc[] → Position-based indexing (select rows/columns by integer positions).
- Direct indexing allows selection of specific columns or filtering with conditions.

Handling Missing Data:

- Real-world datasets often contain missing values (NaN).
- Pandas provides functions to detect, remove, or replace missing values.
- Detection: isnull(), notnull().
- Removal: dropna() removes rows/columns with missing values.
- Imputation (Filling): fillna() replaces NaN with specific values, mean, median, or forward/backward fill.
- Interpolation: Estimates missing values based on trends.

3) Write a python program for Data Indexing and Handling Missing Data

This program shows how to create a DataFrame, access data using various indexing methods, and handle missing values.

```
import pandas as pd

import numpy as np

# Create a DataFrame with some missing data (NaN)

data = {'City': ['London', 'Paris', 'Tokyo', 'Sydney', 'New York'],

        'Population': [8.9, np.nan, 14.0, 5.3, 8.4],

        'Area': [1572, 105, 2194, np.nan, 784]}

cities_df = pd.DataFrame(data)

print("Original DataFrame:\n", cities_df)

# Indexing and Selection

print("\n--- Indexing and Selection ---")

print("Population column:\n", cities_df['Population'])

print("\nRow at index 2 (Tokyo):\n", cities_df.iloc[2])

print("\nRows for 'London' and 'New York':\n", cities_df.loc[[0, 4], ['City', 'Area']])

# Handling Missing Data

print("\n--- Handling Missing Data ---")

# Drop rows with any missing values

df_dropped = cities_df.dropna()

print("DataFrame after dropping rows with NaN:\n", df_dropped)

# Fill missing values with a specific value

df_filled = cities_df.fillna(0)

print("\nDataFrame after filling NaN with 0:\n", df_filled)
```

output

```
City  Population    Area
0    London         8.9  1572.0
1     Paris         NaN   105.0
2     Tokyo        14.0  2194.0
3    Sydney         5.3     NaN
4  New York         8.4   784.0

--- Indexing and Selection ---
Population column:
 0     8.9
1     NaN
2    14.0
3     5.3
4     8.4
Name: Population, dtype: float64

Row at index 2 (Tokyo):
 City           Tokyo
Population      14.0
Area          2194.0
Name: 2, dtype: object

Rows for 'London' and 'New York':
        City    Area
0    London  1572.0
4  New York   784.0

--- Handling Missing Data ---
```

DataFrame after dropping rows with NaN:

	City	Population	Area
0	London	8.9	1572.0
2	Tokyo	14.0	2194.0
4	New York	8.4	784.0

DataFrame after filling NaN with 0:

	City	Population	Area
0	London	8.9	1572.0
1	Paris	0.0	105.0
2	Tokyo	14.0	2194.0
3	Sydney	5.3	0.0
4	New York	8.4	784.0

Program 4

Write a python program for Combining Datasets and Grouping

Aim

To learn how to combine multiple datasets using different techniques and to perform grouping operations for data aggregation using **Pandas** in Python.

Objectives

1. To understand methods for combining datasets: concatenation, merging, and joining.
2. To perform grouping operations using groupby() for data analysis.
3. To explore aggregation functions such as sum(), mean(), and count().
4. To demonstrate practical examples of dataset combination and grouping.

Theory

1. Combining Datasets:

- Real-world data often comes from multiple sources that must be combined. Pandas provides several ways:
    - **Concatenation (pd.concat)** → Stacks DataFrames either row-wise or column-wise.
    - **Merging (pd.merge)** → Combines datasets based on common columns/keys (like SQL JOIN).
    - **Joining (df.join)** → Joins two DataFrames on index or key column.

2. Grouping and Aggregation:

- The groupby() function is used to split data into groups based on some criteria, apply an operation (aggregate, transform, filter), and combine results.
- Common aggregation functions: sum(), mean(), count(), min(), max().
- Useful for analyzing datasets (e.g., average sales by region, total marks per student).

4)Write a python program for Combining Datasets and Grouping

This program demonstrates how to combine data from multiple DataFrames and use the groupby() function for data aggregation.

```
import pandas as pd
# Create two DataFrames
df1 = pd.DataFrame({'key': ['A', 'B', 'C', 'D'],
                    'value1': [1, 2, 3, 4]})
df2 = pd.DataFrame({'key': ['B', 'D', 'E', 'F'],
                    'value2': [5, 6, 7, 8]})
# Combining Datasets
print("--- Combining Datasets ---")
# Merge the two DataFrames
merged_df = pd.merge(df1, df2, on='key', how='inner')
print("Inner merge of df1 and df2:\n", merged_df)
# Grouping and Aggregation
print("\n--- Grouping and Aggregation ---")
data = {'Category': ['A', 'B', 'A', 'B', 'A'],
        'Value': [10, 20, 15, 25, 30]}
df_group = pd.DataFrame(data)
# Group by 'Category' and calculate the mean
grouped_data = df_group.groupby('Category').mean()
print("Grouped data by category, showing mean:\n", grouped_data)
```

--- Combining Datasets ---

Inner merge of df1 and df2:

	key	value1	value2
0	B	2	5
1	D	4	6

--- Grouping and Aggregation ---

Grouped data by category, showing mean:

Category	Value
A	18.333333
B	22.500000

Program 5

Write a python program for Combining Datasets and Grouping

Aim

To plot simple line and scatter graphs using the **Matplotlib** library in Python.

Objectives

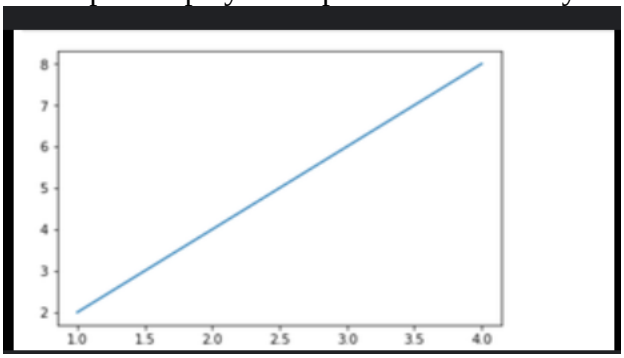
1. To understand the basics of data visualization with Matplotlib.
2. To plot data using **line plots**.
3. To represent data points using **scatter plots**.
4. To compare and interpret data visually.

Theory

- **Data Visualization** is the graphical representation of information and data.
- **Matplotlib** is a popular Python library for creating 2D plots and graphs.

Line Plot:

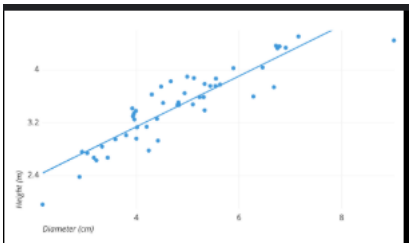
- A line plot displays data points connected by straight lines.



- Useful for showing trends over time or continuous data.
- Implemented with `plt.plot(x, y)`.

Scatter Plot:

- A scatter plot shows individual data points as dots.
- Useful for analyzing the relationship between two variables.
- Implemented with `plt.scatter(x, y)`.

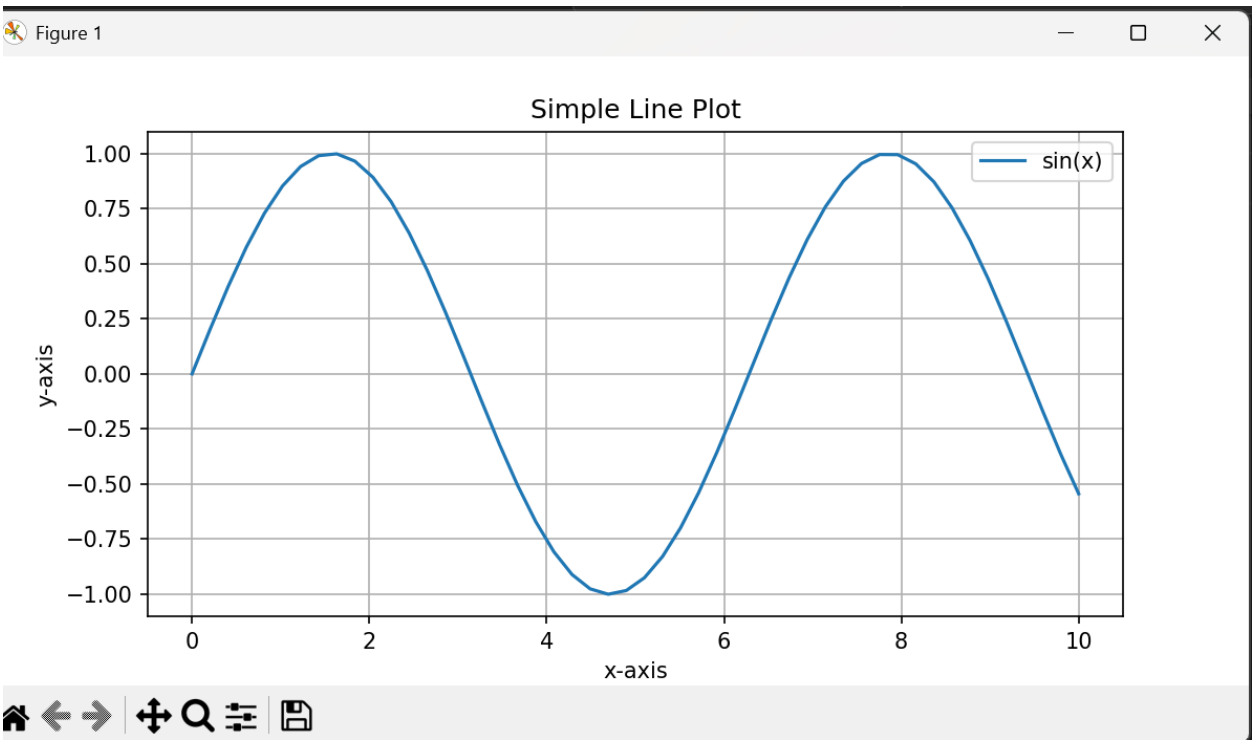


5)Write a python program for Simple Line and Scatter Plots

This program creates basic line and scatter plots to visualize trends and relationships in data.

```
import matplotlib.pyplot as plt
import numpy as np
# Data for plotting
x = np.linspace(0, 10, 50)
y = np.sin(x)
# Simple Line Plot
plt.figure(figsize=(8, 4))
plt.plot(x, y, label='sin(x)')
plt.title('Simple Line Plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend()
plt.grid(True)
plt.show()

# Simple Scatter Plot
rng = np.random.default_rng(42) # for reproducibility
x_scatter = rng.random(100)
y_scatter = rng.random(100)
```

```
plt.figure(figsize=(8, 4))
plt.scatter(x_scatter, y_scatter)
plt.title('Simple Scatter Plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```



Program 6

Write a python program for Histograms and Density Plots

Aim

To plot histograms and density plots using Python libraries **Matplotlib** and **Seaborn**.

Objectives

1. To understand histogram visualization for frequency distribution of data.
2. To learn about density plots (KDE – Kernel Density Estimation) as a smooth alternative to histograms.
3. To implement histograms and density plots using Python.
4. To compare histogram and density plots for the same dataset.

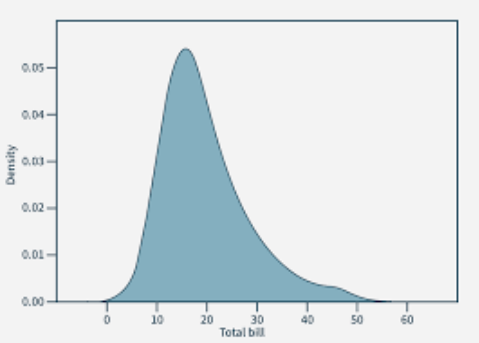
Theory

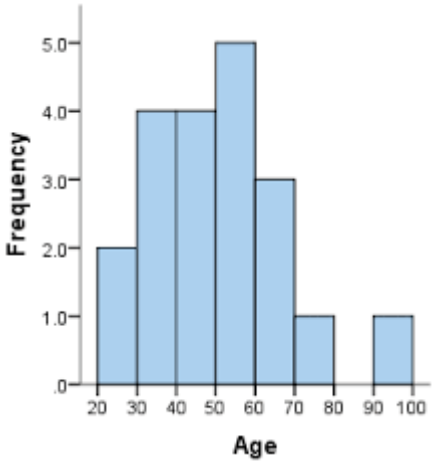
Histogram:

- A histogram groups numerical data into bins (intervals) and displays the frequency of values in each bin.
- Useful for understanding the distribution and spread of data.
- Implemented with plt.hist(data, bins, color, edgecolor).

Density Plot (KDE – Kernel Density Estimation):

- A density plot is a smoothed version of a histogram.
- It estimates the probability density function of a continuous variable.
- Implemented with sns.kdeplot(data) in Seaborn.



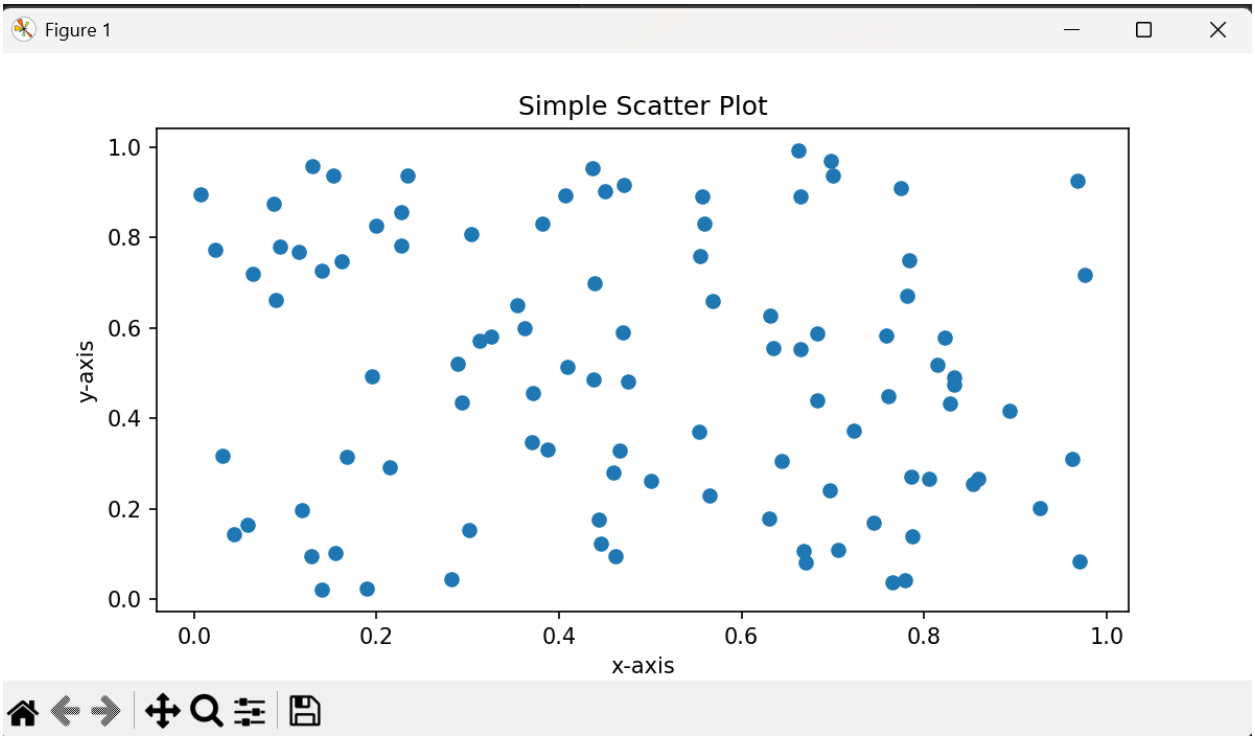


6) Write a python program for Histograms and Density Plots

This program demonstrates how to visualize data distributions using histograms and kernel density estimation plots.

```
import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns # Seaborn is often used for density plots and histograms

# Generate random data

data = np.random.randn(1000)

# Histogram

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

plt.hist(data, bins=30, alpha=0.7, color='skyblue', edgecolor='black')

plt.title('Histogram of Random Data')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.show()

# Density Plot (Kernel Density Estimate)

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

```
sns.kdeplot(data, fill=True, color='purple')
plt.title('Density Plot of Random Data')
plt.xlabel('Value')
plt.ylabel('Density')
plt.show()
```



Program 7

Write a python program for Multiple Subplots and Plot Customization

Aim

To create multiple subplots in a single figure and apply customization options for better visualization using **Matplotlib**.

Objectives

1. To understand how to create multiple subplots using `plt.subplot()` and `plt.subplots()`.
2. To learn customization of plots (titles, labels, legends, colors, line styles, grids).
3. To display different types of plots in one figure.
4. To enhance readability and presentation of plots.

Theory

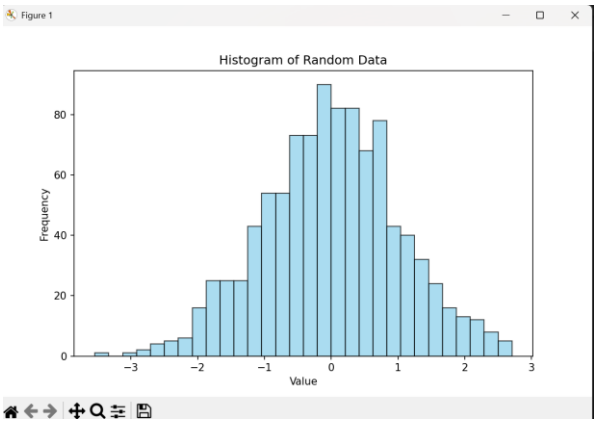
- **Subplots:**
  - Subplots allow multiple plots in one figure for comparison.
  - `plt.subplot(rows, cols, index)` → Adds subplots by grid position.
  - `plt.subplots(rows, cols)` → Creates a grid of subplots with shared customization options.
- **Customization:**
  - Titles: `plt.title("Title")`
  - Axis labels: `plt.xlabel()`, `plt.ylabel()`
  - Legends: `plt.legend()`
  - Line styles: `linestyle`, `color`, `marker`
  - Grids: `plt.grid(True)`

7)Write a python program for Multiple Subplots and Plot Customization

This program shows how to create multiple plots in a single figure and customize them with titles, legends, and gridlines.

```
import matplotlib.pyplot as plt

import numpy as np

# Data for subplots

x1 = np.linspace(0, 10, 100)
```

```
y1 = np.sin(x1)
y2 = np.cos(x1)
# Create a figure with a 2x1 grid of subplots
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(8, 8))
# Plot on the first subplot
axes[0].plot(x1, y1, color='blue', linestyle='--', label='sin(x)')
axes[0].set_title('Sinusoidal Plot')
axes[0].set_xlabel('x-axis')
axes[0].set_ylabel('y-axis')
axes[0].legend()
axes[0].grid(True)
# Plot on the second subplot
axes[1].plot(x1, y2, color='green', marker='o', markersize=3, label='cos(x)')
axes[1].set_title('Cosinusoidal Plot')
axes[1].set_xlabel('x-axis')
axes[1].set_ylabel('y-axis')
axes[1].legend()
axes[1].grid(True)
# Adjust layout to prevent titles from overlapping
plt.tight_layout()
plt.show()
```



Program 8

Write a python program for Three-Dimensional Plotting

Aim

To visualize data in three dimensions (3D) using **Matplotlib's mplot3d toolkit**.

Objectives

1. To understand how to create 3D plots in Python.
2. To implement 3D line, scatter, and surface plots.
3. To explore applications of 3D visualization in data analysis.
4. To customize 3D plots with labels, titles, and colors.

Theory

- **Matplotlib's `mplot3d` toolkit** enables three-dimensional data visualization.
- **Types of 3D plots:**
  - **3D Line Plot (`ax.plot3D`)** → Shows data along three axes.
  - **3D Scatter Plot (`ax.scatter3D`)** → Plots individual points in 3D space.
  - **3D Surface Plot (`ax.plot_surface`)** → Displays a continuous surface over a 2D grid.

8) Write a python program for Three-Dimensional Plotting

This program demonstrates a basic three-dimensional scatter plot, useful for visualizing data with three variables.

```
import matplotlib.pyplot as plt

import numpy as np

# Matplotlib requires a specific import for 3D plotting

from mpl_toolkits.mplot3d import Axes3D

# Generate random data for x, y, and z axes

rng = np.random.default_rng(42)

x_3d = rng.standard_normal(100)

y_3d = rng.standard_normal(100)

z_3d = rng.standard_normal(100)
```

 # Create a figure and add a 3D subplot

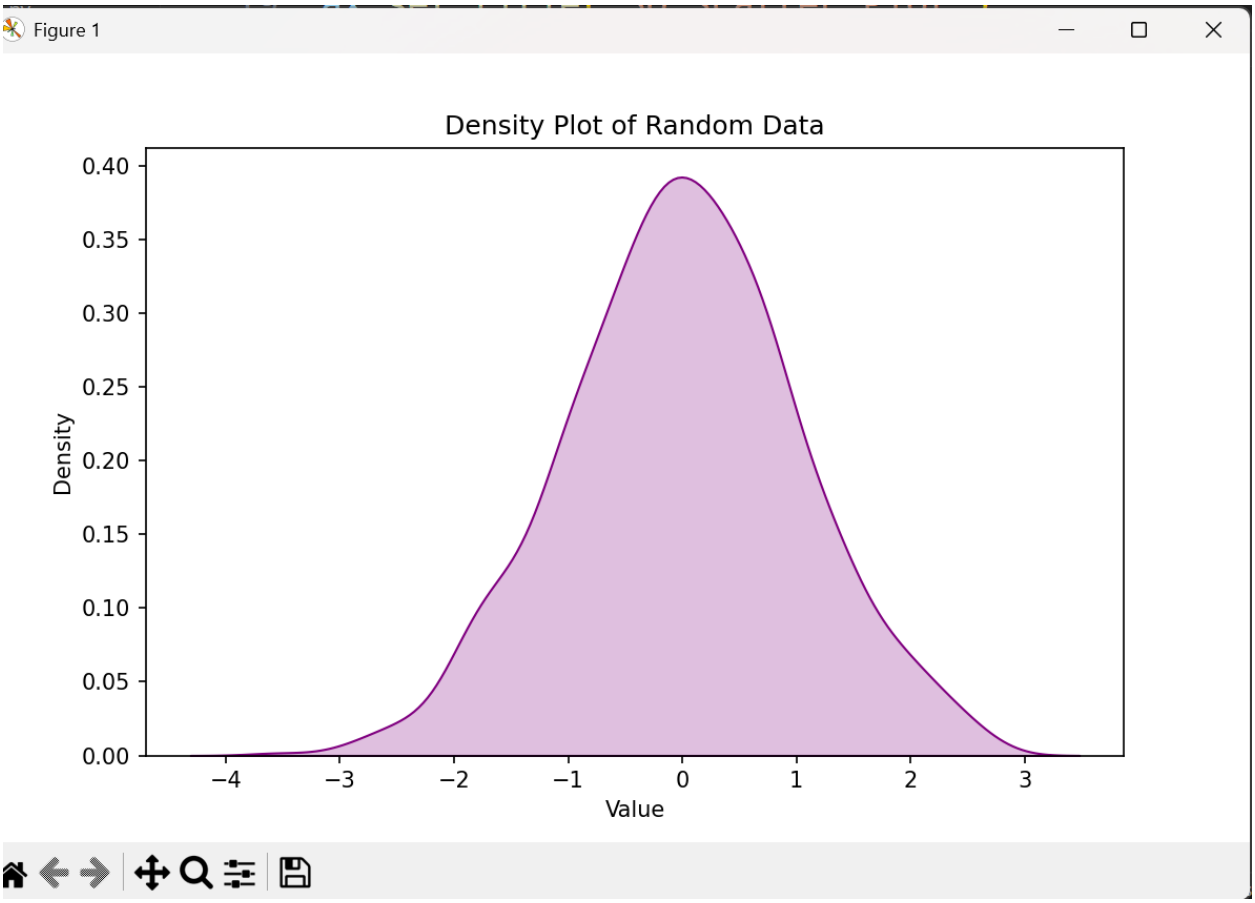
```
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
```

 # Create the 3D scatter plot

```
ax.scatter(x_3d, y_3d, z_3d, c=z_3d, cmap='viridis')
```

 # Set labels and title

```
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.set_title('3D Scatter Plot')
plt.show()
```



Program 9

Write a python program for Seaborn Fundamentals - Distribution and Relationship Plots

Aim

To visualize data distributions and relationships between variables using **Seaborn** plots.

Objectives

1. To understand Seaborn as a statistical data visualization library.
2. To plot **distribution plots** (histogram, KDE, box, violin).
3. To plot **relationship plots** (scatter, line, pair plot).
4. To interpret datasets using visual insights.

Theory

- **Seaborn** is built on top of Matplotlib and provides a high-level interface for attractive and informative statistical graphics.

1. Distribution Plots:

- Show how data is distributed.
- Examples:
    - histplot() – Histogram for frequency.
    - kdeplot() – Kernel Density Estimation (smooth curve).
    - boxplot() – Displays quartiles and outliers.
    - violinplot() – Combines boxplot with KDE.

2. Relationship Plots:

- Show how two or more variables are related.
- Examples:
    - scatterplot() – Plots points between two variables.
    - lineplot() – Shows trends/lines.
    - pairplot() – Multi-variable scatter and distribution visualization.

9) Write a python program for Seaborn Fundamentals - Distribution and Relationship Plots

This program covers fundamental Seaborn plots like a **histogram** for visualizing a single variable's distribution and a **pairplot** for exploring relationships between multiple variables.

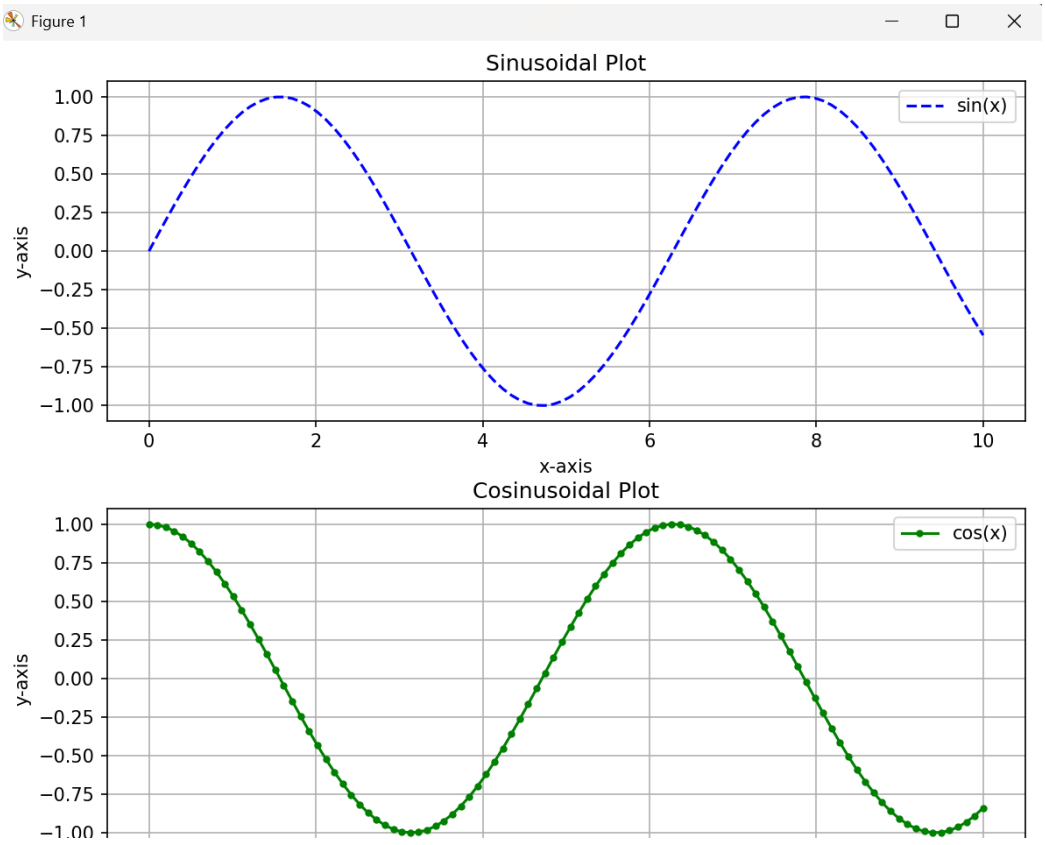
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Create a sample DataFrame
rng = np.random.default_rng(42)
data = {
    'A': rng.normal(loc=10, scale=2, size=100),
    'B': rng.normal(loc=50, scale=10, size=100),
    'C': rng.exponential(scale=3, size=100),
}
df = pd.DataFrame(data)
print("Sample DataFrame head:\n", df.head())
# Plot 1: Histogram (Distribution)
# Use a histogram to visualize the distribution of variable 'A'
plt.figure(figsize=(8, 6))
sns.histplot(data=df, x='A', kde=True)
plt.title('Distribution of Variable A')
plt.xlabel('Values of A')
plt.ylabel('Frequency')
plt.show()
# Plot 2: Pairplot (Relationships)
# Use a pairplot to show the relationship between all numeric variables
# It creates scatter plots for each pair and a histogram for each individual variable
```

```
sns.pairplot(df)
plt.suptitle('Pairplot of All Variables', y=1.02)
plt.show()
```

 # Plot 3: Boxplot (Distribution & Outliers)

 # A boxplot to visualize the distribution and potential outliers of a single variable

```
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, y='B')
plt.title('Boxplot of Variable B')
plt.ylabel('Values of B')
plt.show()
```



Write a python program for Mini Project - Analyzing a Real-World Dataset

Aim

To perform data analysis and visualization on a real-world dataset using **Python (Pandas, Matplotlib, Seaborn)**.

Objectives

1. To load and explore a dataset using Pandas.
2. To handle missing values and basic preprocessing.
3. To perform **statistical analysis** and **grouping**.
4. To visualize data using histograms, scatter plots, boxplots, and correlation heatmaps.
5. To gain meaningful insights from real-world data.

Dataset Used

We will use the **Iris dataset** (famous dataset in Machine Learning) which contains:

- **Features:** Sepal Length, Sepal Width, Petal Length, Petal Width
- **Target:** Species of flower (Setosa, Versicolor, Virginica)

10) Write a python program for Mini Project - Analyzing a Real-World Dataset

This mini-project demonstrates the integration of **Pandas, Matplotlib, and Seaborn** to perform exploratory data analysis on the well-known **Iris dataset**. It uses various plots to understand the data's structure and relationships.

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load the Iris dataset using Seaborn's built-in function

iris_df = sns.load_dataset('iris')

print("Iris Dataset head:\n", iris_df.head())

# 1. Visualize petal length distribution by species

# We'll use a violin plot which combines a boxplot and a kernel density estimate
```

```
plt.figure(figsize=(10, 7))
sns.violinplot(x='species', y='petal_length', data=iris_df)
plt.title('Distribution of Petal Length by Species')
plt.xlabel('Species')
plt.ylabel('Petal Length (cm)')
plt.show()
# 2. Explore the relationships between variables using a pairplot
# The 'hue' parameter colors points by species, revealing group relationships
sns.pairplot(iris_df, hue='species')
plt.suptitle('Pairplot of Iris Dataset', y=1.02)
plt.show()
# 3. Create a heatmap of the correlation matrix
# This shows the linear relationships between all pairs of numeric variables
plt.figure(figsize=(8, 6))
correlation_matrix = iris_df.drop('species', axis=1).corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap of Iris Dataset')
plt.show(
```