1. What is NumPy, and why is it widely used in Python?

NumPy, short for Numerical Python, is a powerful open-source library in Python that is primarily used for numerical and scientific computing.

Ease of Use: NumPy's syntax is straightforward and intuitive, making it accessible for both beginners and experienced programmers.

Community and Documentation: Being one of the oldest libraries for numerical computing in Python, NumPy has a large community and extensive documentation, which makes it easier to find resources and support.

Interoperability: NumPy arrays can be easily converted to and from other data types, such as Python lists and Pandas DataFrames, facilitating data manipulation and analysis.

Support for Large Datasets: NumPy is optimized for performance and can handle large datasets efficiently, which is crucial for data science and machine learning applications.

Standard in Scientific Computing: Many scientific and engineering applications rely on NumPy, making it a standard tool in the field. Its widespread adoption means that many tutorials, courses, and resources are available for learning.

2 How does broadcasting work in NumPy?

Broadcasting is a powerful feature in NumPy that allows for arithmetic operations to be performed on arrays of different shapes and sizes without the need for explicit replication of data. This capability makes it easier to write concise and efficient code when working with arrays. Here’s how broadcasting works in NumPy:

Basic Principles of Broadcasting Array Shapes: When performing operations on two arrays, NumPy compares their shapes element-wise, starting from the trailing dimensions (i.e., the rightmost dimensions).

Dimension Compatibility: Two dimensions are compatible when:

They are equal, or One of them is 1 (in which case it can be "stretched" to match the other dimension). Stretching: If the dimensions of the arrays do not match, NumPy will "stretch" the smaller array across the larger array to make their shapes compatible. This does not involve copying data; instead, it creates a view that allows for the operation to be performed as if the smaller array were replicated.

3 What is a Pandas DataFrame?

A Pandas DataFrame is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure provided by the Pandas library in Python. It is one of the most commonly used data structures in data analysis and manipulation, and it is particularly well-suited for ha structured data.

Key Features of a Pandas DataFrame: Tabular Structure: A DataFrame is similar to a table in a database or a spreadsheet, consisting of rows and columns. Each column can hold different data types (e.g., integers, floats, strings).

Labeled Axes: Each row and column in a DataFrame can be labeled with an index. This allows for easy access and manipulation of data using labels rather than just integer-based indexing.

Size-Mutable: You can easily add or remove columns and rows from a DataFrame, making it flexible for data manipulation.

4.Explain the use of the groupby() method in Pandas?

The groupby() method in Pandas is a powerful tool used for splitting a DataFrame into groups based on some criteria, allowing for the aggregation, transformation, or filtering of data. It is particularly useful for summarizing data and performing operations on subsets of data.

Common Use Cases

Aggregation: You can use groupby() to compute summary statistics for each group, such as mean, sum, count, etc.

Multiple Aggregations: You can apply multiple aggregation functions at once using the agg() method.

Transformations: You can also use groupby() to perform transformations on the data, such as normalizing values within each group.

5. Why is Seaborn preferred for statistical visualizations?

Seaborn is preferred for statistical visualizations due to its high-level interface that simplifies the creation of complex visualizations with minimal code. It enhances Matplotlib's capabilities by providing built-in themes and statistical functionalities, making it easier to visualize data distributions and relationships. ### Key Advantages of Seaborn for Statistical Visualizations

High-Level Interface: Seaborn offers a user-friendly API that allows users to create complex visualizations with less code compared to Matplotlib. This reduces the amount of boilerplate code needed for plotting.

Built-in Themes: Seaborn comes with several aesthetically pleasing themes and color palettes, which enhance the visual appeal of the plots. This makes it easier to create professional-looking graphics without extensive customization.

Statistical Functions: Seaborn provides a variety of built-in statistical functions that facilitate the visualization of data distributions and relationships. This includes functions for creating box plots, violin plots, and pair plots, which are essential for exploratory data analysis.

7. What is a heatmap, and when should it be used?

A heatmap is a data visualization technique that uses color to represent the values of a matrix or a two-dimensional dataset. It provides a way to visualize complex data in a more intuitive format, allowing viewers to quickly identify patterns, correlations, and trends within the data.

When to Use a Heatmap: Heatmaps are particularly useful in the following scenarios:

Correlation Analysis: Heatmaps are commonly used to visualize correlation matrices, where the strength of the relationship between pairs of variables is represented. This helps in identifying which variables are positively or negatively correlated.

Data Density Visualization: Heatmaps can be used to visualize the density of data points in a two-dimensional space, such as in geographical data or user activity on a website. This helps in identifying hotspots or areas of high activity.

Comparative Analysis: When comparing multiple categories or groups across different variables, heatmaps can provide a clear visual representation of how each category performs relative to others.

Time Series Data: Heatmaps can be used to visualize time series data, where time is represented on one axis and another variable on the other axis. This can help in identifying trends over time.

8. What does the term “vectorized operation” mean in NumPy?

In NumPy, the term "vectorized operation" refers to the ability to perform operations on entire arrays (or large datasets) at once, rather than using explicit loops to iterate through individual elements. This approach leverages the underlying implementation of NumPy, which is optimized for performance, allowing for efficient computation and memory usage

9 How does Matplotlib differ from Plotly?

Matplotlib and Plotly are both popular libraries for data visualization in Python, but they have different features, capabilities, and use cases. Here are the key differences between Matplotlib and Plotly:

1. Type of Visualizations Matplotlib:

Primarily designed for static visualizations. It excels at creating a wide range of 2D plots, including line plots, bar charts, histograms, scatter plots, and more. While it has some capabilities for creating 3D plots, its primary strength lies in 2D visualizations. Plotly:

Designed for interactive visualizations. It allows users to create dynamic and interactive plots that can be embedded in web applications or Jupyter notebooks.

2. Interactivity Matplotlib:

Primarily focuses on static plots. While it does have some interactive capabilities (e.g., zooming and panning in certain backends), it is not inherently designed for interactivity. Users car

interactive plots using additional libraries like mpld3 or matplotlib.widgets, but this requires extra effort. Plotly:

Built from the ground up for interactivity. Users can hover over data points to see tooltips, zoom in and out, and pan around the plot without any additional coding. Supports features like dropdowns, sliders, and buttons to create complex interactive dashboards.

3. Ease of Use Matplotlib:

Has a steeper learning curve for beginners due to its extensive customization options and the need to manage figure and axis objects explicitly. Requires more lines of code to achieve complex visualizations compared to Plotly. Plotly:

Generally easier to use for creating interactive visualizations. The syntax is often more intuitive, and it requires fewer lines of code to create complex plots. Provides a high-level interface that abstracts many of the details involved in creating interactive visualizations.

10.What is the significance of hierarchical indexing in Pandas?

Hierarchical indexing, also known as multi-level indexing, is a powerful feature in Pandas that allows you to work with data that has multiple levels of indexing. This feature is particularly useful for organizing and analyzing complex datasets that have multiple dimensions or categories. Here are the key significances and benefits of hierarchical indexing in Pandas:

1. Multi-dimensional Data Representation Hierarchical indexing allows you to represent multi-dimensional data in a two-dimensional DataFrame or Series. This is particularly useful when dealing with datasets that have multiple categorical variables. For example, you can have a DataFrame indexed by both "Country" and "Year," allowing you to easily analyze data across these two dimensions.
2. Enhanced Data Organization Hierarchical indexing helps in organizing data in a more structured way. It allows you to group related data together, making it easier to navigate and manipulate. For instance, you can group sales data by "Region" and "Product," which helps in analyzing sales performance across different regions and products simultaneously.
3. Simplified Data Selection and Slicing With hierarchical indexing, you can easily select and slice data at different levels of the index. This allows for more intuitive data retrieval. For example, you can access all data for a specific country or year without needing to filter the entire DataFrame.

11. What is the role of Seaborn’s pairplot() function?

The pairplot() function in Seaborn is a powerful tool for visualizing relationships between multiple variables in a dataset. It creates a grid of scatter plots (or other types of plots) for each pair of variables in a DataFrame, allowing for a comprehensive exploration of the relationships and distributions of the data. Here are the key roles and features of the pairplot()

function:

1. Visualizing Pairwise Relationships The primary role of pairplot() is to visualize pairwise relationships between all numerical variables in a dataset. It generates a matrix of plots, where each cell in the matrix represents a scatter plot of two variables. This helps in identifying correlations, trends, and potential outliers.
2. Diagonal Plots for Distribution On the diagonal of the pairplot, you can visualize the distribution of each variable. By default, pairplot() uses histograms or kernel density estimates (KDE) to show the distribution of each variable. This provides insight into the distribution characteristics (e.g., normality, skewness) of individual variables.
3. Color Coding by Categorical Variables pairplot() allows you to color-code the points in the scatter plots based on a categorical variable. This is useful for visualizing how different categories relate to each other across multiple dimensions. By using the hue parameter, you can easily distinguish between different groups in the dataset.
4. Faceting

The pairplot() function can also create separate plots for different subsets of the data using the hue parameter. This allows for a more detailed analysis of how relationships differ across categories.

12 What is the purpose of the describe() function in Pandas?

The describe() function in Pandas is a powerful and convenient method used to generate descriptive statistics of a DataFrame or Series. It provides a quick overview of the central tendency, dispersion, and shape of the dataset's distribution, which is essential for exploratory data analysis (EDA). Here are the key purposes and features of the describe() function:

1. Summary Statistics The primary purpose of the describe() function is to compute and return summary statistics for the numerical columns in a DataFrame. By default, it provides the following statistics:

Count: The number of non-null entries in each column. Mean: The average value of the entries in each column. Standard Deviation (std): A measure of the amount of variation or dispersion in the dataset. Minimum (min): The smallest value in each column. 25th Percentile (25%): The value below which 25% of the data falls (first quartile). 50th Percentile (50%): The median value, which is the middle value of the dataset. 75th Percentile (75%): The value below which 75% of the data falls (third quartile). Maximum (max): The largest value in each column.

2. Handling Different Data Types The describe() function can be used with both numerical and categorical data. When applied to a DataFrame containing categorical columns, it provides different statistics, such as:

Count: The number of non-null entries. Unique: The number of unique values in the colum
The most frequently occurring value. Frequency: The count of the most frequently occurr

value.

13 Why is handling missing data important in Pandas?

Handling missing data is a crucial aspect of data analysis and preprocessing in Pandas (and in data science in general) for several reasons. Missing data can significantly impact the quality of your analysis, the performance of machine learning models, and the validity of your conclusions. Here are the key reasons why handling missing data is important:

1. Data Integrity and Quality Accuracy of Analysis: Missing data can lead to biased or incorrect results. If certain observations are missing, the analysis may not accurately reflect the underlying patterns in the data. Statistical Validity: Many statistical methods assume that the data is complete. Missing values can violate these assumptions, leading to unreliable statistical inferences.
2. Impact on Machine Learning Models Model Performance: Most machine learning algorithms cannot handle missing values directly. If missing data is not addressed, it can lead to errors during model training or prediction, or the model may perform poorly. Feature Engineering: Missing values can affect the creation of new features. If not handled properly, the derived features may also contain missing values, complicating the modeling process.
3. Bias Introduction Systematic Bias: If the missing data is not random (i.e., missing data is related to the value of the variable itself or other variables), it can introduce systematic bias into the analysis. This can lead to misleading conclusions. Loss of Information: Ignoring or improperly handling missing data can result in the loss of valuable information that could be critical for understanding the dataset.

14. What are the benefits of using Plotly for data visualization?

01 Interactive Visualizations Dynamic Features:

Plotly creates interactive plots that allow users to hover over data points to see tooltips, zoom in and out, and pan around the plot. This interactivity enhances user engagement and makes it easier to explore complex datasets. Customizable Interactivity: Users can add buttons, sliders, and dropdowns to create dynamic dashboards that allow for real-time data exploration and filtering.

2 Wide Range of Plot Types Diverse Visualization Options: Plotly supports a wide variety of plot types, including line charts, scatter plots, bar charts, histograms, box plots, heatmaps, 3D plots, geographical maps, and more. This versatility allows users to choose the most appropriate visualization for their data.

Complex Visualizations: Plotly can handle complex visualizations, such as contour plots, surface plots, and network graphs, which may be challenging to create with other librarie

3. High-Quality Graphics Publication-Ready Figures: Plotly generates high-quality, aesthetically pleasing graphics that are suitable for presentations, reports, and publications. The visualizations are rendered in vector format, ensuring clarity and scalability. Custom Styling: Users can customize the appearance of plots, including colors, fonts, and layout, to match branding or personal preferences.

4. Ease of Use

Intuitive Syntax: Plotly's API is designed to be user-friendly, making it easy for both beginners and experienced users to create complex visualizations with minimal code.

Integration with Pandas: Plotly works seamlessly with Pandas DataFrames, allowing users to create visualizations directly from their data without extensive data manipulation.

5. Web-Based and Cross-Platform Web Integration: Plotly visualizations can be easily embedded in web applications, dashboards, and Jupyter notebooks. This makes it a great choice for creating interactive data-driven applications.

Cross-Platform Compatibility: Plotly visualizations can be viewed in any modern web browser, making them accessible across different devices and platforms.

15.How does NumPy handle multidimensional arrays?

NumPy is a powerful library in Python that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. Here's how NumPy handles multidimensional arrays:

1. N-Dimensional Arrays (ndarray) Core Data Structure: The primary data structure in NumPy for handling multidimensional data is the ndarray (N-dimensional array). An ndarray can have any number of dimensions, which allows for the representation of scalars (0D), vectors (1D), matrices (2D), and higher-dimensional data (3D, 4D, etc.). Homogeneous Data: All elements in a NumPy array must be of the same data type, which allows for efficient storage and computation.
2. Creating Multidimensional Arrays NumPy provides several functions to create multidimensional arrays:

Using np.array(): You can create an array from a nested list (or tuple) to form a multidimensional array.

3. Indexing and Slicing NumPy provides powerful indexing and slicing capabilities for multidimensional arrays:

Basic Indexing: You can access elements using a tuple of indices. For example, to access an element in a 2D array:

16 What is the role of Bokeh in data visualization?

Bokeh is an interactive and open-source Python library designed for creating visualizations such as plots, dashboards, and applications. Its key features include the ability to create highly interactive graphs, support for streaming data, and the capability to build complex data dashboards for user exploration.

1. Interactive Visualizations

User Engagement: Bokeh allows users to create interactive plots that enable viewers to explore data dynamically. This includes features like zooming, panning, and hovering over data points to reveal additional information.

Widgets and Tools: It provides various tools and widgets that facilitate user interaction, such as sliders, dropdowns, and buttons, allowing users to manipulate the visualizations in real-time.

2. Integration with Web Technologies

Web-Based Applications: Bokeh visualizations can be easily embedded into web applications, making it suitable for creating dashboards that can be shared and accessed via web browsers.

Support for Jupyter Notebooks: Bokeh works seamlessly with Jupyter notebooks, allowing data scientists to create and display interactive visualizations alongside their code and data analysis.

3. Data Handling Capabilities Streaming Data: Bokeh supports real-time data streaming, which is essential for applications that require live updates, such as financial dashboards or IoT data monitoring.

Efficient Data Sources: It can handle large datasets efficiently through various data sources, including ColumnDataSource, which allows for easy updates and modifications to the data being visualized.

4. Customization and Flexibility Customizable Plots: Users can create a wide range of visualizations, from simple line plots to complex multi-faceted dashboards, with extensive customization options for aesthetics and layout.

Integration with Other Libraries: Bokeh can be used alongside other Python libraries like NumPy, Pandas, and SciPy, enhancing its capabilities for data manipulation and analysis.

17.Explain the difference between apply() and map() in Pandas?

1. Functionality and Scope

apply():

The apply() function is used to apply a function along the axis of a DataFrame (rows or columns) or to a Series.

It can take a function that operates on an entire row or column, making it more versatile for complex operations.

It can be used with both DataFrames and Series.

map():

The map() function is primarily used with Series. It applies a function element-wise to each value in the Series.

It is generally used for simpler operations, such as transforming or mapping values based on a function, dictionary, or Series.

It cannot be used directly on DataFrames.

2. Input and Output

apply():

The input can be a function that takes a Series or DataFrame as an argument, and the output can be a Series, DataFrame, or scalar, depending on the function applied. It can return different shapes based on the function used. map():

The input is typically a function that takes a single value and returns a single value. It can also take a dictionary or Series for mapping values.

The output is always a Series of the same length as the input Series.

3. Use Cases

apply():

Use apply() when you need to perform operations that require access to multiple columns or when you want to apply a function that operates on entire rows or columns.

Example: Calculating the sum of two columns or applying a custom function that requires multiple inputs.

map():

Use map() when you want to transform or map values in a Series based on a function, dictionary, or another Series.

Example: Replacing values in a Series or applying a simple transformation to each element.

18. What are some advanced features of NumPy? NumPy is a powerful library for numerical computing in Python, and it offers a wide range of advanced features that enhance its capabilities for handling arrays and performing mathematical operations. Here are some of the advanced features of NumPy:

1.. Broadcasting Definition: Broadcasting is a powerful mechanism that allows NumPy to perform arithmetic operations on arrays of different shapes. It automatically expands the smaller array across the larger array to make their shapes compatible.

2.. Advanced Indexing and Slicing Boolean Indexing: You can use boolean arrays to index into other arrays, allowing for conditional selection of elements.

3.. Linear Algebra Functions Matrix Operations: NumPy provides a comprehensive set of functions for linear algebra, including matrix multiplication, determinants, eigenvalues, and

solving linear systems.

19. How does Pandas simplify time series analysis?

Pandas is a powerful library in Python that provides extensive support for time series analysis, making it easier for data scientists and analysts to work with time-based data. Here are several ways in which Pandas simplifies time series analysis:

1. Date and Time Data Types

DatetimeIndex: Pandas provides a specialized DatetimeIndex that allows for efficient indexing and manipulation of time series data. This index can handle various date and time formats, making it easy to work with time-based data. Datetime Conversion: Pandas has built-in functions like pd.to_datetime() that convert strings or other formats into datetime objects, simplifying the process of preparing time series data.

2. Resampling and Frequency Conversion Resampling:

   Pandas allows you to easily change the frequency of your time series data using the resample() method. This is useful for aggregating data (e.g., converting daily data to monthly averages) or downsampling/upscaling time series.

   3. Time Series Indexing and Slicing Indexing: With a DatetimeIndex, you can easily index and slice time series data using date ranges. This allows for straightforward selection of data for specific time periods

04 Time Series Operations Date Arithmetic: Pandas supports date arithmetic, allowing you to perform operations like adding or subtracting time intervals from dates easily.

20 What is the role of a pivot table in Pandas?

A pivot table in Pandas is a powerful tool for data analysis that allows you to summarize and reorganize data in a DataFrame. It provides a way to aggregate and transform data, making it easier to analyze and visualize complex datasets. Here are the key roles and features of pivot tables in Pandas:

1. Data Summarization

Aggregation: Pivot tables allow you to aggregate data based on one or more keys (columns). You can compute various summary statistics (like sum, mean, count, etc.) for different groups of data.

Multi-dimensional Analysis: You can create multi-dimensional summaries by specifying multiple index and column variables, enabling you to analyze data across different dimensions.

2. Data Reshaping Reorganizing Data: Pivot tables reshape data from a long format to a wide format, making it easier to read and interpret. This is particularly useful when you want to compare values across different categories.

Creating a Matrix: Pivot tables create a matrix-like structure where you can see the relationships between different variables, with one variable represented in rows and another in columns.

3. Flexible Grouping

Custom Grouping: You can group data by one or more columns, allowing for flexible analysis based on different categories or attributes.

Hierarchical Indexing: Pivot tables can create hierarchical indices (multi-level indices) for rows and columns, which is useful for organizing complex datasets.

4. Handling Missing Data Filling Missing Values: Pivot tables can handle missing data by allowing you to specify how to fill in gaps (e.g., using fill_value parameter) when aggregating data.

21.Why is NumPy’s array slicing faster than Python’s list slicing?

NumPy's array slicing is generally faster than Python's list slicing due to several key factors related to how NumPy is implemented and how it handles data. Here are the main reasons for this performance difference:

1. Contiguous Memory Allocation

Memory Layout: NumPy arrays are stored in contiguous blocks of memory, which allows for efficient access and manipulation of data. This contiguous memory layout enables NumPy to take advantage of low-level optimizations and cache efficiency. Data Type Homogeneity: All elements in a NumPy array are of the same data type, which allows for more efficient memory usage and faster computations compared to Python lists, which can store mixed data types.

2. Vectorized Operations

Element-wise Operations: NumPy is designed for vectorized operations, meaning that operations can be applied to entire arrays at once without the need for explicit loops. This is achieved through optimized C and Fortran code under the hood, which is much faster than Python's interpreted loops.

Batch Processing: When slicing a NumPy array, the operation is performed in a batch manner, leveraging low-level optimizations that are not available in Python's list slicing.

3. No Overhead of Python Objects

Lightweight Data Structures: NumPy arrays are implemented as a single data structure in memory, while Python lists are collections of pointers to Python objects. This means that accessing elements in a NumPy array involves less overhead than accessing elements in a list, where each element is a separate object.

Reduced Function Call Overhead: NumPy operations are implemented in C, which reduces the overhead associated with Python function calls and allows for faster execution.

22 What are some common use cases for Seaborn?

Seaborn is a powerful Python data visualization library built on top of Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. It is particularly well-suited for visualizing complex datasets and making statistical graphics easier to create and interpret.

Here are some common use cases for Seaborn:

1. Exploratory Data Analysis (EDA) Visualizing Distributions: Seaborn provides functions like sns.histplot(), sns.kdeplot(), and sns.boxplot() to visualize the distribution of a dataset, helping to identify patterns, outliers, and the overall shape of the data
2. Comparing Groups Categorical Data Visualization: Seaborn excels at visualizing categorical data through functions like sns.barplot(), sns.countplot(), and sns.boxplot(), which allow for easy comparison of different groups.

3.3. Correlation Analysis Heatmaps: Seaborn's sns.heatmap() function is commonly used to visualize correlation matrices, making it easy to identify relationships between multiple variables in a dataset.

01.How do you create a 2D NumPy array and calculate the sum of each row?

```
import numpy as np

# Step 2: Create a 2D NumPy array
array_2d = np.array([[1, 2, 3],
                     [4, 5, 6],
                     [7, 8, 9]])

# Step 3: Calculate the sum of each row
row_sums = np.sum(array_2d, axis=1)

# Print the results
print("2D Array:")
print(array_2d)
print("\nSum of each row:")
print(row_sums)
```

```
2D Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Sum of each row:
[ 6 15 24]
```

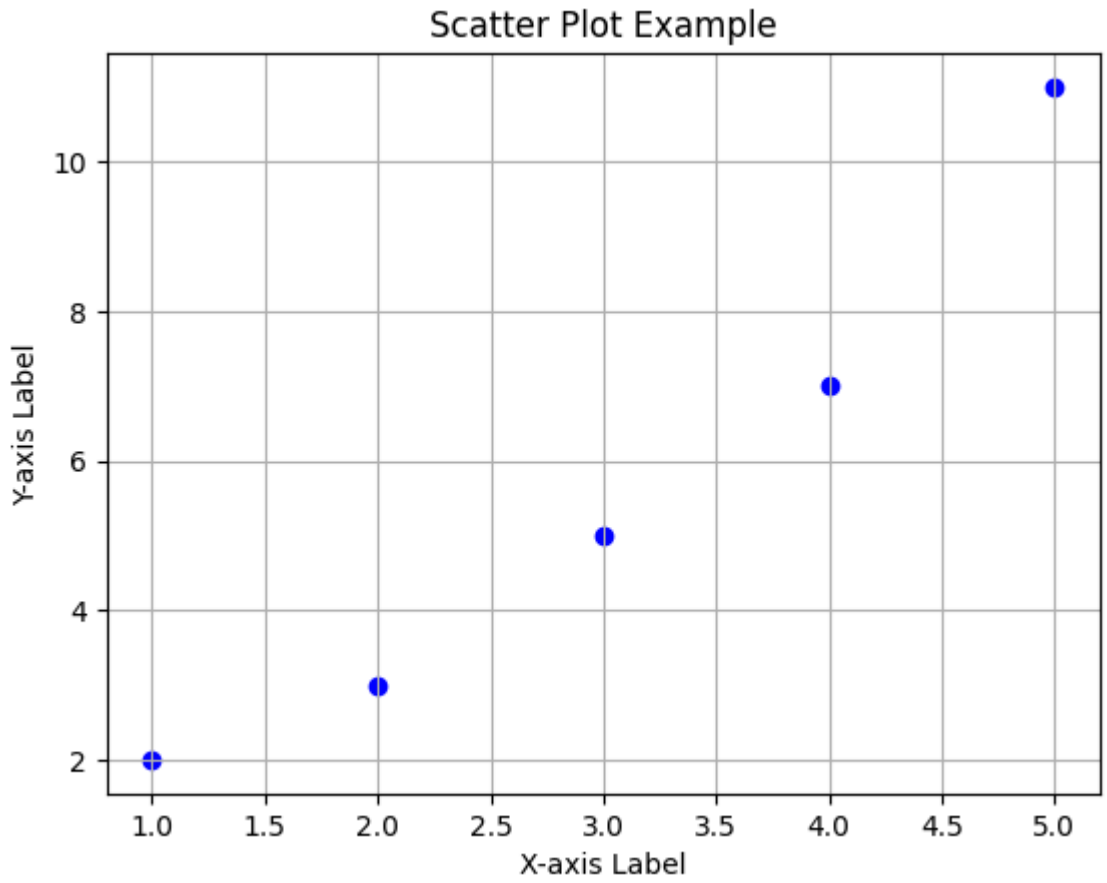
##2.Write a Pandas script to find the mean of a specific column in a DataFrame?

```
import pandas as pd

# Step 2: Create a DataFrame
```

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [24, 30, 22, 35],
    'Salary': [50000, 60000, 55000, 70000]
}

df = pd.DataFrame(data)

# Step 3: Calculate the mean of the 'Salary' column
mean_salary = df['Salary'].mean()

# Print the results
print("DataFrame:")
print(df)
print("\nMean Salary:")
print(mean_salary)
```

```
DataFrame:
      Name  Age  Salary
0    Alice   24   50000
1      Bob   30   60000
2  Charlie   22   55000
3    David   35   70000

Mean Salary:
58750.0
```

```
#3 Create a scatter plot using Matplotlib?

import matplotlib.pyplot as plt

# Step 3: Prepare Data
# Example data
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# Step 4: Create the Scatter Plot
plt.scatter(x, y, color='blue', marker='o')

# Adding titles and labels
plt.title('Scatter Plot Example')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')

# Show the plot
plt.grid(True)  # Optional: Add a grid for better readability
plt.show()
```

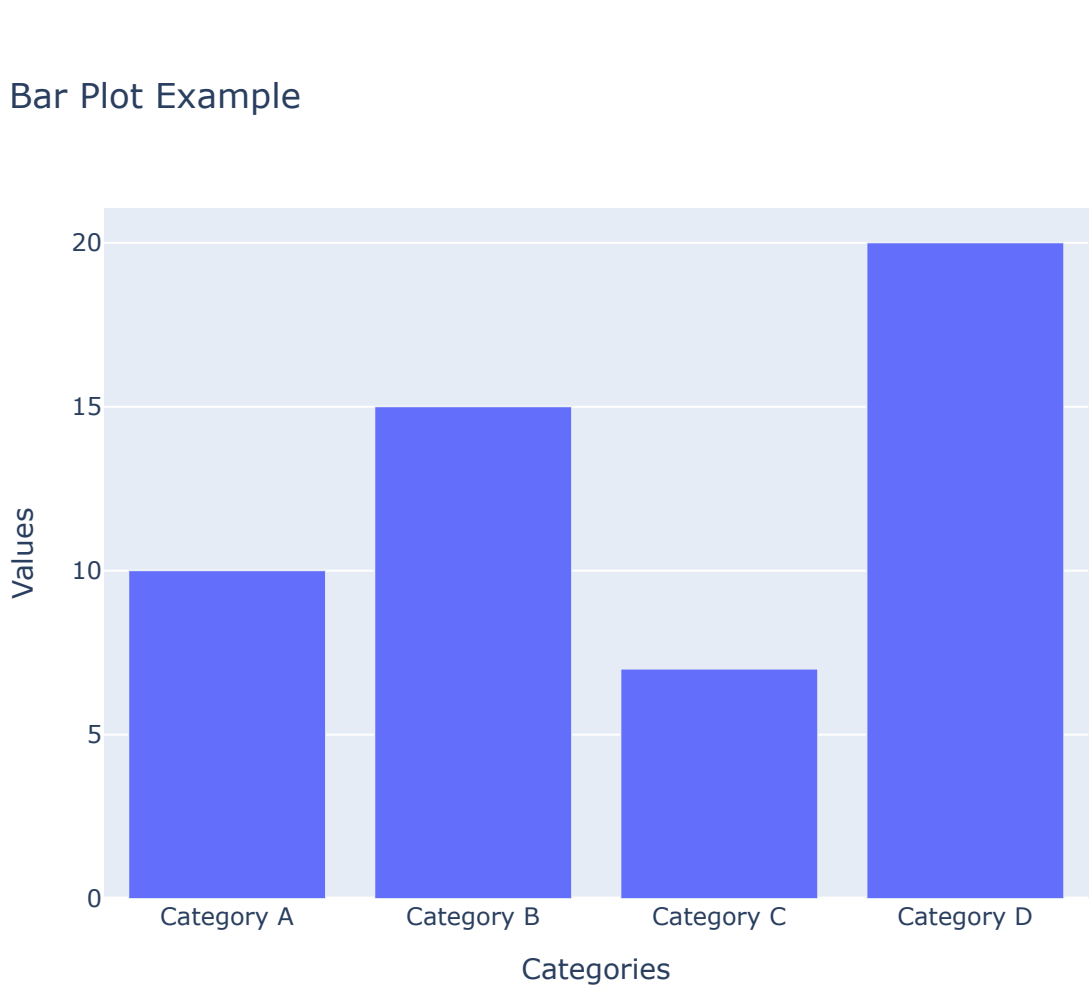



```
#4How do you calculate the correlation matrix using Seaborn and visualize it with a heatm

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv("path_to_your_dataset.csv")

# Compute correlation matrix
correlation_matrix = data.corr(numeric_only=True)

# Visualize the correlation matrix with a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', square=True)
plt.title('Correlation Heatmap')
plt.show()
```

5 Generate a bar plot using Plotly?

```
import plotly.graph_objects as go

# Step 3: Prepare Data
# Example data
categories = ['Category A', 'Category B', 'Category C', 'Category D']
values = [10, 15, 7, 20]

# Step 4: Create the Bar Plot
```

```
fig = go.Figure(data=[go.Bar(x=categories, y=values)])

# Adding titles and labels
fig.update_layout(
    title='Bar Plot Example',
    xaxis_title='Categories',
    yaxis_title='Values'
)

# Show the plot
fig.show()
```



6. Create a DataFrame and add a new column based on an existing column?

```
import pandas as pd

# Step 3: Create a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [24, 30, 22, 35],
    'Salary': [50000, 60000, 55000, 70000]
}

df = pd.DataFrame(data)
```

```
# Step 4: Add a new column based on an existing column
# For example, let's create a new column 'Salary After Tax' which is 80% of the original
df['Salary After Tax'] = df['Salary'] * 0.8

# Print the DataFrame
print("DataFrame with New Column:")
print(df)
```

```
DataFrame with New Column:
      Name  Age  Salary  Salary After Tax
0    Alice   24   50000           40000.0
1      Bob   30   60000           48000.0
2  Charlie   22   55000           44000.0
3    David   35   70000           56000.0
```

7. Write a program to perform element-wise multiplication of two NumPy arrays?

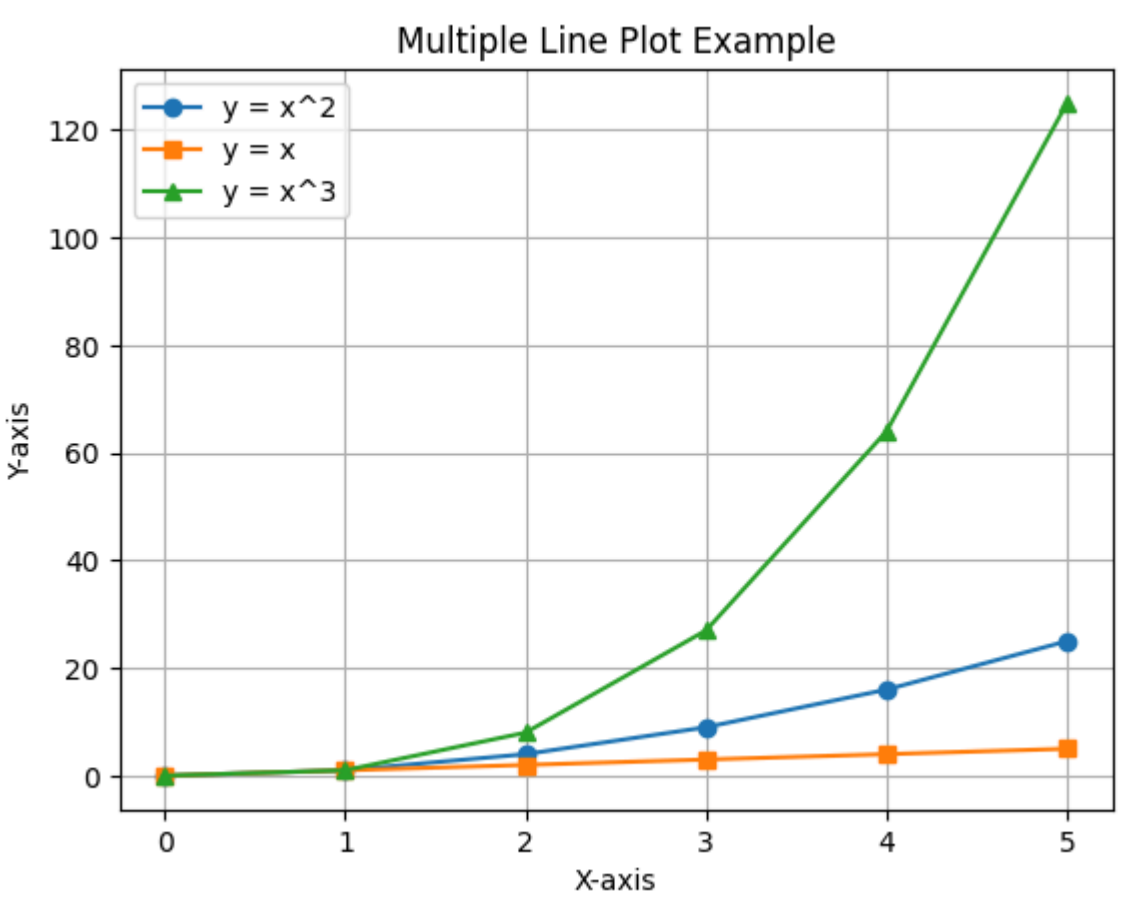
```
import numpy as np

# Step 3: Create two NumPy arrays
array1 = np.array([1, 2, 3, 4])
array2 = np.array([5, 6, 7, 8])

# Step 4: Perform element-wise multiplication
result = array1 * array2

# Print the result
print("Array 1:", array1)
print("Array 2:", array2)
print("Element-wise multiplication result:", result)
```

```
Array 1: [1 2 3 4]
Array 2: [5 6 7 8]
Element-wise multiplication result: [ 5 12 21 32]
```

8 Create a line plot with multiple lines using Matplotlib?

```
import matplotlib.pyplot as plt

# Step 3: Prepare Data
# Example data
x = [0, 1, 2, 3, 4, 5]
y1 = [0, 1, 4, 9, 16, 25]  # y = x^2
y2 = [0, 1, 2, 3, 4, 5]     # y = x
y3 = [0, 1, 8, 27, 64, 125]  # y = x^3

# Step 4: Create the Line Plot
plt.plot(x, y1, label='y = x^2', marker='o')  # Line 1
plt.plot(x, y2, label='y = x', marker='s')     # Line 2
plt.plot(x, y3, label='y = x^3', marker='^')    # Line 3

# Adding titles and labels
```

```
plt.title('Multiple Line Plot Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

# Show legend
plt.legend()

# Show the plot
plt.grid(True)  # Optional: Add a grid for better readability
plt.show()
```



9.Generate a Pandas DataFrame and filter rows where a column value is greater than a threshold?

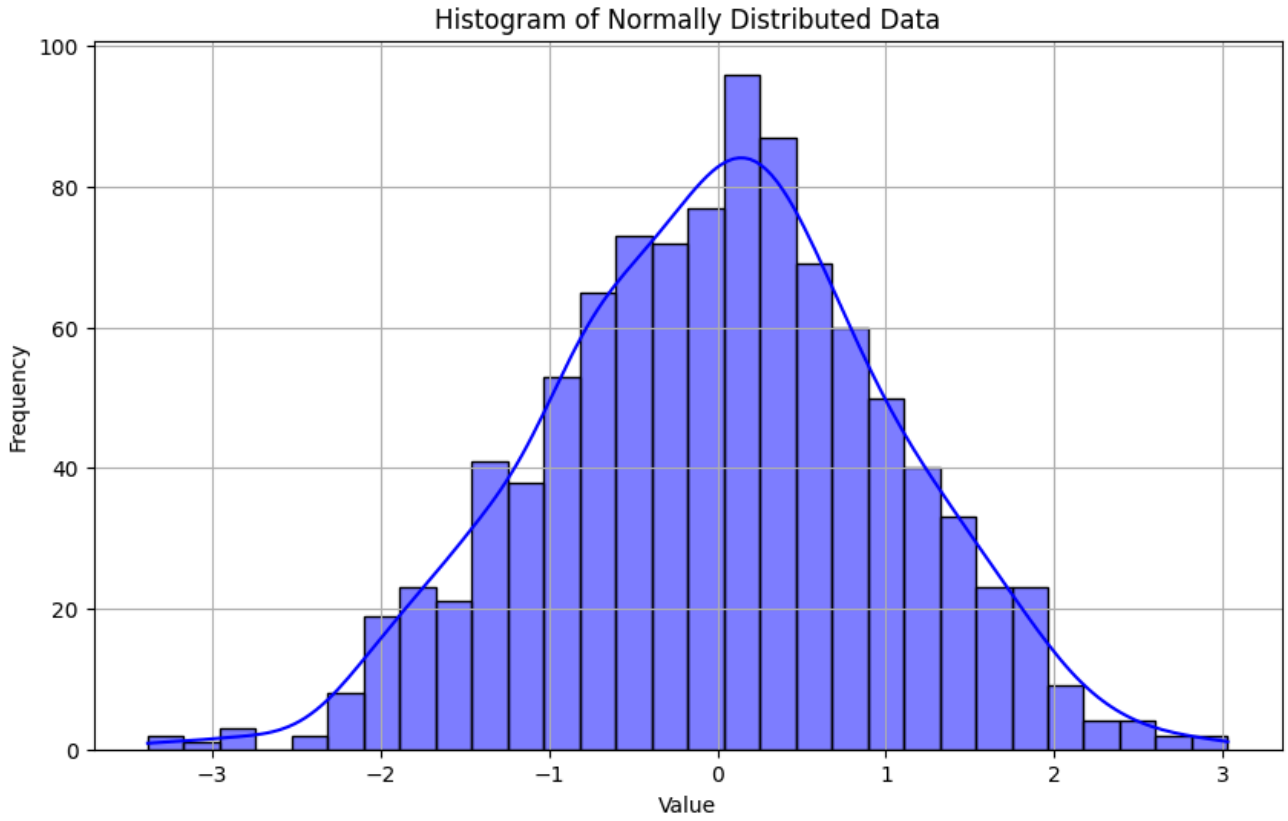
```
import pandas as pd

# Step 3: Create a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [24, 30, 22, 35],
    'Salary': [50000, 60000, 55000, 70000]
}

df = pd.DataFrame(data)

# Step 4: Filter rows where 'Salary' is greater than a threshold
threshold = 55000
filtered_df = df[df['Salary'] > threshold]
```

```
# Print the original DataFrame and the filtered DataFrame
print("Original DataFrame:")
print(df)
print("\nFiltered DataFrame (Salary > 55000):")
print(filtered_df)
```

```
Original DataFrame:
      Name  Age  Salary
0    Alice   24   50000
1      Bob   30   60000
2  Charlie   22   55000
3    David   35   70000

Filtered DataFrame (Salary > 55000):
    Name  Age  Salary
1    Bob   30   60000
3  David   35   70000
```

10.Create a histogram using Seaborn to visualize a distribution?

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Step 3: Prepare Data
# Generate random data
data = np.random.normal(loc=0, scale=1, size=1000)  # Normal distribution

# Step 4: Create the Histogram
plt.figure(figsize=(10, 6))  # Optional: Set the figure size
sns.histplot(data, bins=30, kde=True, color='blue', edgecolor='black')

# Adding titles and labels
plt.title('Histogram of Normally Distributed Data')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Show the plot
plt.grid(True)  # Optional: Add a grid for better readability
plt.show()
```



11.Perform matrix multiplication using NumPy?

```
import numpy as np

# Step 3: Create two matrices
# Matrix A (2x3)
A = np.array([[1, 2, 3],
              [4, 5, 6]])

# Matrix B (3x2)
B = np.array([[7, 8],
              [9, 10],
              [11, 12]])

# Step 4: Perform matrix multiplication
# Method 1: Using the @ operator
result1 = A @ B

# Method 2: Using np.dot()
result2 = np.dot(A, B)

# Print the results
print("Matrix A:")
print(A)
print("\nMatrix B:")
print(B)
print("\nResult of A @ B:")
print(result1)
print("\nResult of np.dot(A, B):")
print(result2)
```

```
Matrix A:
[[1 2 3]
 [4 5 6]]

Matrix B:
[[ 7  8]
 [ 9 10]
 [11 12]]

Result of A @ B:
[[ 58  64]
 [139 154]]

Result of np.dot(A, B):
[[ 58  64]
 [139 154]]
```

12. Use Pandas to load a CSV file and display its first 5 rows?

```
#import pandas as pd

# Step 3: Load the CSV file
# Replace 'your_file.csv' with the path to your CSV file
# For example, if you have a file named 'data.csv' in the same directory, use 'da
#df = pd.read_csv('your_file.csv')
```

```
# Step 4: Display the first 5 rows
print("First 5 rows of the DataFrame:")
print(df.head())
```

```
First 5 rows of the DataFrame:
      Name  Age  Salary
0    Alice   24   50000
1      Bob   30   60000
2  Charlie   22   55000
3    David   35   70000
```

13. Create a 3D scatter plot using Plotly.

```
import pandas as pd
import plotly.express as px

# Load the dataset
#movies = pd.read_csv('data.csv')

# Filter out rows with missing values
#filtered_movies = movies.dropna(subset=['runtime', 'audience_rating', 'tomatometer_ratin

# Create the 3D scatter plot
#fig = px.scatter_3d(
    #filtered_movies,
   #x='runtime',
    #y='audience_rating',
    #z='tomatometer_rating',
    #color='genres',
    #title='3D Scatter Plot of Movie Runtime vs Ratings',
    #labels={
        #'runtime': 'Runtime (minutes)',
        #'audience_rating': 'Audience Rating (%)',
        #'tomatometer_rating': 'Tomatometer Rating (%)'



# Customize the layout
fig.update_layout(
    legend=dict(
        x=1.05,
        y=0.5,
        title=None,
        font=dict(size=8),
        itemsizing='constant',
        bgcolor='rgba(255, 255, 255, 0.7)',
```