


Q1.What is NumPy, and why is it widely used in Python ?

- NumPy is a powerful Python library primarily used for numerical and scientific computing. it is widely used in Python:-
- Performance: NumPy operations are implemented in C, making them significantly faster than equivalent operations in pure Python, especially for large arrays.
- Versatility: It offers a comprehensive suite of mathematical functions, including linear algebra, Fourier transforms, and random number generation, facilitating complex computations.
- Integration: NumPy serves as the foundation for many other scientific libraries in Python, such as SciPy, pandas, and scikit-learn, making it integral to the scientific Python ecosystem.
- Ease of Use: With its intuitive syntax and extensive documentation, NumPy is accessible to both beginners and experienced programmers.
- Community Support: Being open-source, it has a large and active community, ensuring continuous development and support.

Q2.How does broadcasting work in NumPy ?

- Ans:-  How Broadcasting Works
- When performing operations on two arrays, NumPy compares their shapes element-wise, starting from the trailing dimensions. Two dimensions are compatible when:
 - They are equal, or
 - One of them is 1

If these conditions are met, NumPy stretches the smaller array across the larger one so that they have compatible shapes. This process does not involve actual data replication, which makes broadcasting memory-efficient.


Q3.What is a Pandas DataFrame ?

- A Pandas DataFrame is a two-dimensional, size-mutable, and heterogeneous tabular data structure in Python, equipped with labeled axes (rows and columns). It is a core component of the Pandas library, widely used for data manipulation and analysis.

Key Features of a Pandas DataFrame

- Two-Dimensional Structure: Data is organized in rows and columns, similar to a spreadsheet or SQL table. Pandas
- Labeled Axes: Both rows and columns have labels, allowing for intuitive data selection and alignment.
- Heterogeneous Data: Each column can contain data of different types (e.g., integers, floats, strings).
- Size-Mutable: You can add or remove columns and rows as needed.
- Rich Functionality: Supports a wide range of operations, including data alignment, aggregation, filtering, and merging.

Q4.Explain the use of the groupby() method in Pandas ?

- Ans  The groupby() method in Pandas is a powerful tool for grouping and analyzing data based on specific criteria. It follows the "split-apply-combine" strategy, which involves splitting the data into groups, applying a function to each group independently, and then combining the results into a new DataFrame.

How groupby() Works

- Splitting: The data is divided into groups based on the values in one or more columns.
- Applying: A function is applied to each group separately. This function can be
- Aggregation: Computing summary statistics like sum, mean, or count.
- Transformation: Performing operations that return an object of the same size as the group.
- Filtration: Filtering out groups based on a condition. [GeeksforGeeks](#)
- Combining: The results of these operations are combined into a new DataFrame.

Q5.Why is Seaborn preferred for statistical visualizations ?

- Ans:- ♦ Seaborn is a Python data visualization library built on top of Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Why Seaborn Excels at Statistical Visualizations

1. High-Level Interface for Complex Plots Seaborn simplifies the creation of complex statistical plots with concise syntax. Functions like `relplot()`, `catplot()`, and `pairplot()` allow users to generate intricate visualizations with minimal code.
2. Seamless Integration with Pandas Designed to work closely with Pandas DataFrames, Seaborn allows direct plotting of data without extensive preprocessing. This integration streamlines the workflow for data analysis and visualization.
3. Built-In Themes and Color Palettes Seaborn offers a variety of built-in themes and color palettes, such as `darkgrid` and `deep`, which enhance the aesthetics of plots. These presets help in creating visually appealing and professional-looking visualizations effortlessly.
4. Advanced Statistical Plotting Functions The library provides specialized functions for statistical plotting, including `sns.histplot()` for distributions, `sns.boxplot()` for categorical data, and `sns.heatmap()` for correlation matrices. These functions are tailored for statistical data visualization, making it easier to understand data distributions and relationships.
5. Facilitates Exploratory Data Analysis (EDA) Seaborn's capabilities are particularly beneficial for EDA, allowing analysts to uncover patterns, trends, and relationships within datasets through visual means. Its functions support the creation of informative plots that aid in data interpretation..

Q6.What are the differences between NumPy arrays and Python lists ?

- ♦ Ans:- NumPy arrays and Python lists are both used to store collections of data, but they differ significantly in terms of structure, performance, and functionality. Here's a detailed comparison:

1. Data Type Consistency

- Python Lists: Can store elements of different data types within the same list. For example:

```
py_list = [1, 3.14, 'hello', True]
```
- NumPy Arrays: Require all elements to be of the same data type (homogeneous). If you attempt to create an array with mixed types, NumPy will upcast to a compatible type to maintain consistency. For example:

```
import numpy as np
np_array = np.array([1, 2, 3])
```

This array will have a uniform data type, such as `int64`.

2. Performance and Memory Efficiency

- Python Lists: Store references to objects, which can lead to higher memory usage and slower performance for numerical operations.
- NumPy Arrays: Store data in contiguous memory blocks, enabling efficient access and manipulation. This design leads to faster computations and reduced memory consumption, especially for large datasets

3. Functionality and Operations

- Python Lists: Do not support element-wise operations inherently. Operations like addition or multiplication need to be performed using loops or list comprehensions.
- NumPy Arrays: Support vectorized operations, allowing element-wise computations without explicit loops. For example:

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = a + b # Output: array([5, 7, 9])
```

4. Flexibility vs. Specialization

- Python Lists: Offer greater flexibility, allowing storage of heterogeneous data types and dynamic resizing. They are suitable for general-purpose programming tasks.
- NumPy Arrays: Are specialized for numerical and scientific computing. They provide a wide range of mathematical functions and are optimized for performance, making them ideal for tasks involving large numerical datasets.

5. Use Cases

- Python Lists: Best suited for applications requiring flexibility, such as storing mixed data types or when the dataset size is relatively small.
- NumPy Arrays: Preferred for numerical computations, data analysis, and situations where performance and memory efficiency are critical. They are commonly used in scientific computing, machine learning, and data processing tasks

Q7.What is a heatmap, and when should it be used ?

- ♦ Ans:-A heatmap is a data visualization technique that represents numerical values using a color gradient, allowing for quick identification of patterns, correlations, and anomalies within a dataset. Typically, higher values are depicted with warmer colors (e.g.,

reds and oranges), while lower values are shown with cooler colors (e.g., blues and greens). This method is particularly effective for visualizing large datasets and understanding complex data relationships.

◆ When to Use a Heatmap

- Heatmaps are especially useful in the following scenarios:
- Analyzing Correlations: Visualizing correlation matrices to identify relationships between variables.
- Website User Behavior: Tracking user interactions such as clicks, scrolls, and mouse movements to optimize web design and user experience.
- Geospatial Analysis: Displaying data density or intensity over geographical areas, such as population density or crime rates. Wikipedia
- Biological Data: Representing gene expression levels or other biological measurements across different conditions or samples.
- Time-Series Data: Visualizing patterns over time, such as activity levels throughout days or weeks.

Q8.What does the term “vectorized operation” mean in NumPy ?

- Ans:- ◆ In NumPy, a vectorized operation refers to performing operations on entire arrays without the need for explicit loops. This approach leverages optimized, pre-compiled code (often written in low-level languages like C) to execute operations efficiently over sequences of data. Vectorization enhances performance by reducing the overhead associated with Python loops and by utilizing CPU-level optimizations.

⚙️ How Vectorized Operations Work

NumPy arrays are homogeneous, meaning all elements are of the same data type. This uniformity allows NumPy to delegate operations to optimized C code, enabling fast execution. For instance, adding two arrays element-wise can be done succinctly:

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = a + b # Output: array([5, 7, 9])
```

🚀 Benefits of Vectorization

- Performance: Vectorized operations are typically much faster than their loop-based counterparts due to reduced interpreter overhead and optimized low-level implementations.
- Conciseness: Code becomes more readable and concise, as operations are expressed in a high-level manner.
- Memory Efficiency: Operations are performed in-place when possible, reducing memory usage.

Q9.How does Matplotlib differ from Plotly ?

- Ans:- ◆ Matplotlib and Plotly are both powerful Python libraries for data visualization, each with its own strengths and ideal use cases. Here's a detailed comparison to help you understand their differences:
- 🍷 Matplotlib: The Classic Visualization Library
 - Overview: Matplotlib is a foundational library in Python's data visualization ecosystem, known for its ability to create static, high-quality plots.

Key Features:

- Static Visualizations: Generates non-interactive plots suitable for publications and reports.
- Customization: Offers granular control over plot elements, including fonts, colors, and line styles.
- Integration: Widely used in conjunction with libraries like NumPy, Pandas, and Seaborn.

Output Formats: Supports various formats such as PNG, PDF, and SVG.

Use Cases:

- Academic research and publications.
- Static reporting and documentation.
- Situations requiring detailed customization of plots

🌐 Plotly: Interactive and Modern Visualizations

Overview: Plotly is a modern library designed for creating interactive and web-ready visualizations.


Key Features:

- Interactivity: Provides features like zooming, panning, and hover tooltips out of the box.
- Ease of Use: Simplifies the creation of complex plots with concise syntax.
- Web Integration: Ideal for dashboards and web applications, especially when combined with frameworks like Dash.
- Aesthetics: Delivers visually appealing plots with minimal configuration.


Use Cases:

- Interactive dashboards and web applications.
- Data exploration and analysis in Jupyter notebooks.
- Presentations requiring dynamic visualizations.

Q10. What is the significance of hierarchical indexing in Pandas ?

- Ans:-  Significance of Hierarchical Indexing
- Efficient Data Representation: Hierarchical indexing allows you to represent multi-dimensional data in a compact and readable format, making it easier to understand and analyze.
- Advanced Data Selection: With a MultiIndex, you can perform more complex data selections using `.loc[]` and `.xs()`, enabling you to access data at different levels of the hierarchy. Programiz
- GroupBy Operations: It enhances the functionality of `groupby()` operations, allowing for aggregation and transformation across multiple levels of data.
- Reshaping Data: Hierarchical indexing facilitates reshaping operations like pivoting and stacking, enabling you to transform data between wide and long formats.
- Time Series Analysis: It's particularly useful in time series analysis where data is indexed by multiple time periods and categories, allowing for more granular analysis.

Q11.What is the role of Seaborn's pairplot() function ?

-  Seaborn's `pairplot()` function is a powerful tool for visualizing pairwise relationships in a dataset, making it an essential component in exploratory data analysis (EDA). It facilitates the examination of interactions between multiple variables simultaneously, providing insights into correlations, distributions, and potential patterns within the data.

What Is pairplot():-

The `pairplot()` function creates a grid of scatter plots for each pair of numerical variables in a dataset, with histograms or kernel density estimates (KDEs) along the diagonal to show the univariate distribution of each variable. This comprehensive visualization allows for the quick identification of relationships and distributions across variables.

Key Features

- Pairwise Scatter Plots: Visualizes relationships between each pair of variables, helping to identify correlations and trends.
- Diagonal Distribution Plots: Displays univariate distributions of variables, providing insights into their individual characteristics.
- Categorical Grouping: Using the `hue` parameter, you can color-code the plots based on a categorical variable, facilitating the comparison of distributions and relationships across different categories. GeeksforGeeks
- Customization: Offers various parameters to customize the plots, such as `kind` for choosing the type of plot (e.g., scatter, KDE), `height` for adjusting the size of the plots, and `corner` to remove the upper (off-diagonal) portion of the grid for a cleaner look.

Example Usage

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

# Load dataset
df = sns.load_dataset('iris')

# Create pairplot
sns.pairplot(df, hue='species', kind='scatter', height=2.5)
plt.show()
```

- Identifying Relationships: Quickly spot linear or nonlinear relationships between variables. Analytics Vidhya
- Detecting Correlations: Visualize correlations to inform feature selection for modeling.
- Recognizing Outliers: Easily identify outliers that may require further investigation or preprocessing.
- Understanding Distributions: Gain insights into the distribution of individual variables, aiding in the selection of appropriate statistical methods.

Q12.What is the purpose of the describe() function in Pandas ?

- ♦ Ans:- The describe() function in Pandas is a powerful tool for quickly generating summary statistics of a DataFrame or Series. It provides insights into the central tendency, dispersion, and shape of the data's distribution, excluding missing values (NaNs)

Key Features of describe()

-Numerical Data: By default, describe() computes the following statistics for each numeric column:

- count: Number of non-null entries.
- mean: Average value.
- std: Standard deviation.
- min: Minimum value.
- 25%: 25th percentile (first quartile).
- 50%: 50th percentile (median).
- 75%: 75th percentile (third quartile).
- max: Maximum value.

-Categorical Data: When applied to object (string) columns, describe() returns:

- count: Number of non-null entries.
- unique: Number of unique values.
- top: Most frequent value.
- freq: Frequency of the most common value.

-Customization: You can customize the output using parameters like percentiles, include, and exclude to control which statistics and data types are included in the summary.

Q13.Why is handling missing data important in Pandas?

- 🧠 Why Handling Missing Data Matters
- Ensures Accurate Analysis: Missing values can skew statistical measures like mean, median, and standard deviation, leading to misleading conclusions.
- Prevents Errors in Computations: Operations involving missing data can result in errors or unintended behavior, affecting the integrity of your analysis. Data Science Dojo Discussions
- Improves Model Performance: Machine learning algorithms often require complete data; missing values can hinder model training and prediction accuracy.
- Facilitates Data Cleaning: Identifying and addressing missing data is a fundamental step in data preprocessing, ensuring a clean dataset for analysis. .

Common Representations of Missing Data in Pandas

-Pandas recognizes several representations for missing data:

- NaN (Not a Number): A special floating-point value from NumPy, commonly used for missing numerical data.
- None: A Python singleton object, often used for missing values in object-type arrays.
- NaT (Not a Time): Used for missing datetime or timedelta data. Pandas
- A nullable missing value introduced in Pandas 1.0, compatible with integer, boolean, and string data types.

Q14.What are the benefits of using Plotly for data visualization?

- Plotly is a powerful open-source data visualization library that enables the creation of interactive, web-ready visualizations with minimal code. Its integration with Python, R, and JavaScript makes it a versatile tool for data scientists, analysts, and developers.

✔ Key Benefits of Using Plotly

1 .Interactive Visualizations

- Plotly excels in creating interactive charts that allow users to zoom, pan, hover, and click to explore data dynamically. This interactivity enhances user engagement and facilitates deeper insights during exploratory data analysis. GeeksforGeeks.

2. Wide Range of Chart Types Plotly supports a diverse array of chart types, including:

- Line, bar, and scatter plots
- 3D surface and scatter plots
- Heatmaps and choropleths
- Box plots, violin plots, and more.

3.Ease of Use

- With libraries like plotly.express, creating complex visualizations becomes straightforward. For instance, generating a scatter plot with a regression line can be achieved in just one line of code:

```
import plotly.express as px
fig = px.scatter(data_frame=df, x='x_column', y='y_column', trendline='ols')
fig.show()
```

4.Seamless Integration with Dash

- Plotly integrates seamlessly with Dash, a Python framework for building analytical web applications. This combination allows for the creation of interactive dashboards that can be deployed and shared easily.

5.High Customizability

- Plotly offers extensive customization options, enabling users to adjust colors, labels, annotations, and more. This flexibility ensures that visualizations can be tailored to meet specific requirements and preferences

Q15.How does NumPy handle multidimensional arrays?

- NumPy efficiently handles multidimensional arrays through its core data structure, the ndarray (N-dimensional array). This structure allows for the storage and manipulation of homogeneous data in a contiguous memory block, enabling high-performance computations.

📊 Structure of a NumPy ndarray

- Shape: A tuple indicating the size of the array along each dimension. For example, an array with shape (3, 4, 5) has 3 dimensions with sizes 3, 4, and 5 respectively.
- ndim: The number of dimensions (axes) of the array. In the above example, ndim would be 3. GeeksforGeeks
- dtype: Specifies the data type of the array's elements, such as float64 or int32.
- strides: A tuple indicating the number of bytes to step in each dimension when traversing the array. This allows for efficient memory access patterns.

These attributes enable NumPy to perform operations on arrays without the need for explicit loops, leveraging vectorized computations for speed and efficiency.

📊 Operations on Multidimensional Arrays

- NumPy supports a wide range of operations on multidimensional arrays, including:
- Element-wise operations: Perform arithmetic operations between arrays of the same shape.
- Broadcasting: Automatically expand arrays of smaller dimensions to match larger ones during operations.
- Aggregation functions: Compute summary statistics like sum(), mean(), and max() along specified axes.

Linear algebra operations: Perform matrix multiplication, transposition, and other linear algebra computations.

Q.16.What is the role of Bokeh in data visualization?

- Bokeh is a powerful and flexible Python library designed for creating interactive, web-ready visualizations. It enables data scientists and developers to build dynamic plots, dashboards, and applications that can be embedded in web pages or run as standalone apps.

🎯 Key Features of Bokeh

1.Interactive Visualizations

Bokeh excels at producing interactive plots with tools like zoom, pan, hover, and selection. These features allow users to explore data dynamically, enhancing the depth of analysis. For instance, you can create scatter plots, line charts, and bar graphs that respond to user

inputs, facilitating a more engaging data exploration experience.

2. Versatile Output Options

Bokeh supports multiple output formats, including:

- Jupyter Notebooks: Ideal for exploratory data analysis within interactive environments.
- Standalone HTML Files: Useful for sharing visualizations without requiring a server.
- Bokeh Server Applications: Enables the creation of interactive web applications with real-time data updates.
- Integration with Web Frameworks: Bokeh plots can be embedded into Flask or Django applications, allowing for seamless integration into existing web projects

3. Layered API

- Bokeh offers a layered API structure to cater to users with varying levels of expertise:
- `bokeh.plotting`: A high-level interface resembling Matplotlib, suitable for quick and easy plot creation.
- `bokeh.models`: Provides low-level access for fine-grained control over plot components, enabling the construction of complex visualizations.

4. Integration with Data Science Tools

- Bokeh integrates seamlessly with popular Python data science libraries such as NumPy, Pandas, and SciPy. This compatibility allows for efficient data manipulation and visualization within a unified workflow.

5. Real-Time Data Streaming

- With Bokeh's streaming capabilities, you can build visualizations that update in real-time. This feature is particularly useful for monitoring live data feeds, such as financial market trends or sensor data streams.

Q17. Explain the difference between `apply()` and `map()` in Pandas?

- In Pandas, `apply()` and `map()` are both used to apply functions to data, but they differ in scope and flexibility.
- `map()`: Element-wise Transformation for Series
 - Applicable to: Series objects only.
 - Functionality: Applies a function to each element in the Series individually. Spark By {Examples}
 - Accepts: A function, dictionary, or another Series.

Use Cases:

- Transforming or mapping values in a Series.
- Replacing values using a dictionary.

Example:

```
import pandas as pd
```

```
s = pd.Series([1, 2, 3]) s_mapped = s.map(lambda x: x ** 2) # Applies the function to each element
```

- `apply()`: Flexible Function Application
 - Applicable to: Both Series and DataFrame objects.

Functionality:

- For Series: Applies a function to each element.
- For DataFrame: Applies a function along an axis (rows or columns)
- Accepts: Callable functions only.

Use Cases:

- Performing row-wise or column-wise operations in a DataFrame.
- Applying complex functions that may not be element-wise.

Example:

```
import pandas as pd
```



```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df_applied = df.apply(lambda x: x.sum(), axis=0) # Sums each column
```

Q18. What are some advanced features of NumPy?

- NumPy offers a suite of advanced features that enhance its efficiency and flexibility for numerical computations. Here's an overview of some key capabilities:

Broadcasting


- Broadcasting allows NumPy to perform arithmetic operations on arrays of different shapes by automatically expanding them to compatible shapes. This eliminates the need for explicit loops and enables efficient vectorized computations. Medium.

Structured Arrays

- Structured arrays enable the storage of heterogeneous data types within a single array, similar to records or rows in a database. This is particularly useful for datasets where each element contains multiple fields of different types.

Fancy Indexing

Fancy indexing allows for the selection of array elements using integer arrays, boolean masks, or other arrays, providing powerful and flexible data retrieval methods.

 **Vectorization** Vectorization refers to the process of replacing explicit loops with array expressions to perform element-wise operations. This leads to more concise code and significant performance improvements.

Universal Functions (ufuncs)

Universal functions are functions that operate element-wise on arrays, supporting array broadcasting, type casting, and several other standard features. They provide a fast and efficient way to perform element-wise operations.

Linear Algebra Operations

NumPy includes a comprehensive set of linear algebra functions, such as matrix multiplication, eigenvalue computation, and singular value decomposition, facilitating advanced mathematical computations.

Q19. How does Pandas simplify time series analysis?

- Pandas simplifies time series analysis by offering a comprehensive suite of tools tailored for handling temporal data. Here's how it streamlines the process:

1. Time-Aware Data Structures

- DatetimeIndex:** Pandas allows setting datetime objects as indices, enabling intuitive time-based indexing and slicing. For instance, you can retrieve data for a specific date range using `df['2021-01':'2021-03']`.
- Timestamp and Timedelta:** These classes facilitate precise time point and duration representations, supporting arithmetic operations like adding days or calculating differences between dates.

2. Resampling and Frequency Conversion

- Resampling:** Pandas provides the `resample()` method to convert time series data to different frequencies. For example, converting daily data to monthly averages is straightforward.
- Frequency Conversion:** You can easily change the frequency of your time series data, such as from daily to weekly, using built-in methods.

3. Rolling and Expanding Windows

- Rolling Windows:** The `rolling()` function enables computation of moving statistics like rolling means or sums, which are essential for smoothing time series data.
- Expanding Windows:** The `expanding()` function allows calculations over an expanding window, useful for cumulative statistics.

4. Date Range Generation and Shifting

- Date Range Generation:** Use `pd.date_range()` to create a range of dates with a specified frequency, aiding in time series creation and analysis.
- Shifting:** The `shift()` method moves data forward or backward in time, which is helpful for computing differences or lagged values.

5. Time Series Visualization

- Pandas integrates seamlessly with Matplotlib, allowing for quick and easy plotting of time series data. By calling the `plot()` method on a time-indexed DataFrame or Series, you can generate informative visualizations.

Q20.What is the role of a pivot table in Pandas?

- The role of a pivot table in pandas is to reshape and summarize data within a DataFrame, enabling users to analyze data from different perspectives. It transforms data by aggregating values based on one or more index and column labels. This functionality is similar to pivot tables in spreadsheet software, offering a flexible way to explore data relationships and patterns.

Pivot tables are particularly useful for:

Data summarization:

- They can aggregate data using functions like sum, mean, count, or custom functions, providing a concise overview of data trends. Data reshaping:
- Pivot tables can convert data from a long format to a wide format, making it easier to compare different variables. Multidimensional analysis:
- They allow for the examination of data across multiple dimensions, revealing insights that might be hidden in raw data. Report generation:
- Pivot tables can be used to create structured reports and summaries for presentations or further analysis.

Q21.Why is NumPy's array slicing faster than Python's list slicing?

- NumPy's array slicing is significantly faster than Python's list slicing due to several key architectural and implementation differences:

1.Contiguous Memory Allocation

- NumPy arrays are stored in contiguous blocks of memory, meaning all elements are laid out sequentially. This layout enhances cache locality, allowing the CPU to efficiently prefetch and access data, leading to faster operations. Data Leads Future
- In contrast, Python lists are arrays of pointers to objects scattered throughout memory. Accessing elements requires dereferencing these pointers, which is less efficient due to potential cache misses.

2.Homogeneous Data Types

- NumPy arrays enforce a single data type for all elements, enabling optimized, low-level operations. This uniformity allows for efficient vectorized computations and reduces overhead.
- Python lists can contain elements of varying data types, necessitating type checks and dynamic dispatch during operations, which slows down performance.

3.Slicing Returns Views, Not Copies

- When slicing a NumPy array, the result is a view of the original data, not a separate copy. This approach avoids unnecessary data duplication and speeds up operations.
- Conversely, slicing a Python list creates a new list and copies the data, which incurs additional memory and time overhead.

4.Implementation in Optimized C Code

- NumPy's core operations, including slicing, are implemented in optimized C code. This low-level implementation minimizes the overhead associated with Python's dynamic typing and interpreter, resulting in faster execution. Stack Overflow.
- Python lists, being high-level abstractions, rely on the interpreter for operations, which introduces additional overhead.

5.Vectorized Operations

- NumPy supports vectorized operations, allowing batch processing of data without explicit Python loops. This capability leverages optimized C routines and CPU features like SIMD (Single Instruction, Multiple Data) to perform operations more efficiently.

Python lists require explicit loops for element-wise operations, which are slower due to interpreter overhead.

Q.22.What are some common use cases for Seaborn?

- Seaborn is a powerful Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn is particularly well-suited for exploring and understanding complex datasets.

1. Exploratory Data Analysis (EDA)

- Seaborn simplifies EDA by offering functions that allow quick visualization of data distributions and relationships. For instance, `pairplot()` can display pairwise relationships in a dataset, helping to identify patterns and correlations.

2. Visualizing Statistical Relationships

- Seaborn provides tools to visualize statistical relationships between variables. Functions like `lplot()` and `regplot()` can plot data and fit regression models, making it easier to understand trends and correlations.

3. Analyzing Distributions

- Understanding the distribution of data is crucial in statistics. Seaborn offers functions like `histplot()`, `kdeplot()`, and `displot()` to visualize univariate and bivariate distributions, aiding in identifying skewness, modality, and outliers.

4. Categorical Data Visualization


- Seaborn excels at visualizing categorical data. Functions such as `boxplot()`, `violinplot()`, `swarmplot()`, and `barplot()` allow for detailed comparison between categories, highlighting differences in distributions and central tendencies.

5. Multi-Plot Grids

- For datasets with multiple variables, Seaborn's `FacetGrid` and `PairGrid` enable the creation of multi-plot grids. These grids facilitate the visualization of relationships across subsets of data, making it easier to detect patterns and interactions.

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

```
import numpy as np
arr = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
row_sums = np.sum(arr, axis=1)
print("Sum of each row:", row_sums)
```


 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

```
# Create a sample DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'Salary': [50000, 60000, 70000, 80000]
}
```

```
df = pd.DataFrame(data)
```

```
mean_salary = df['Salary'].mean()
print(f"Mean Salary: {mean_salary}")
```

 Mean Salary: 65000.0

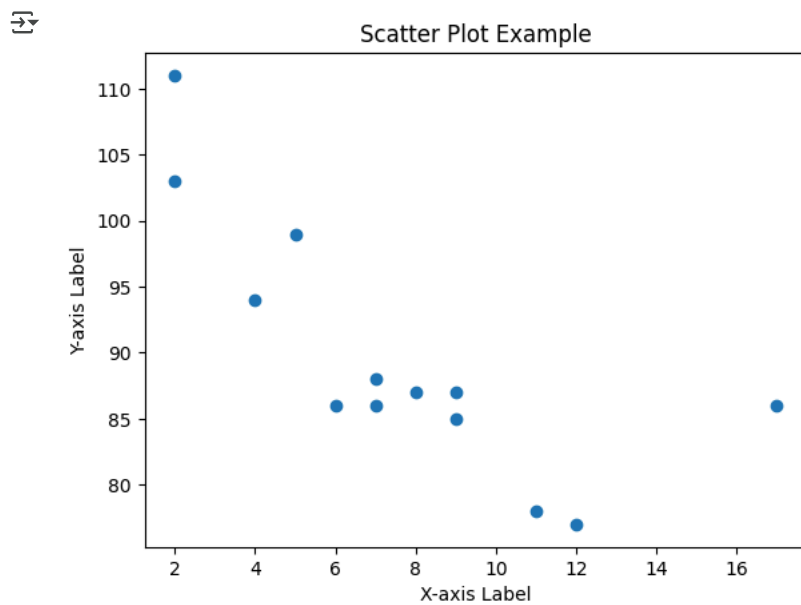
#3 Create a scatter plot using Matplotlib.

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

x = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6])
y = np.array([99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86])

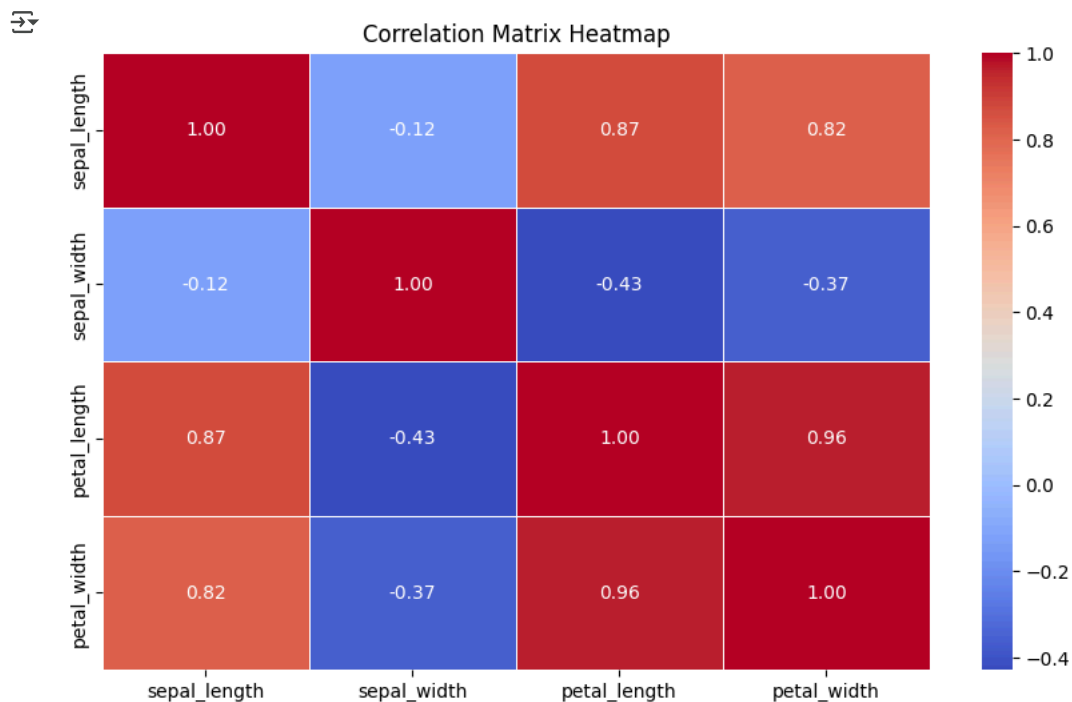
plt.scatter(x, y)

plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Scatter Plot Example')
plt.show()
```



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

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset('iris')
corr_matrix = df.corr(numeric_only=True)
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```



#5. Generate a bar plot using Plotly.

```
import plotly.graph_objects as go

categories = ['Python', 'SQL', 'Excel', 'Power BI']
values = [85, 75, 90, 70]

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

```
fig.update_layout(
    title='Skill Proficiency Levels',
    xaxis_title='Skills',
    yaxis_title='Proficiency (%)',
    template='plotly_white'
)

fig.show()
```


 [Show hidden output](#)

#6. Create a DataFrame and add a new column based on an existing column.
import pandas as pd

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Score': [85, 42, 77, 90]
}
df = pd.DataFrame(data)

df['Result'] = df['Score'].apply(lambda x: 'Pass' if x >= 50 else 'Fail')

print(df)
```




| | Name | Score | Result |
|---|---------|-------|--------|
| 0 | Alice | 85 | Pass |
| 1 | Bob | 42 | Fail |
| 2 | Charlie | 77 | Pass |
| 3 | David | 90 | Pass |

#7. Write a program to perform element-wise multiplication of two NumPy arrays.
import numpy as np

```
array1 = np.array([1, 2, 3, 4])
array2 = np.array([10, 20, 30, 40])

result = array1 * array2

print("Array 1:", array1)
print("Array 2:", array2)
print("Element-wise Multiplication:", result)
```



```
Array 1: [1 2 3 4]
Array 2: [10 20 30 40]
Element-wise Multiplication: [ 10  40  90 160]
```

#8. Create a line plot with multiple lines using Matplotlib.
import matplotlib.pyplot as plt

```
x = [1, 2, 3, 4, 5]
y1 = [10, 20, 25, 30, 35]
y2 = [5, 15, 20, 25, 30]

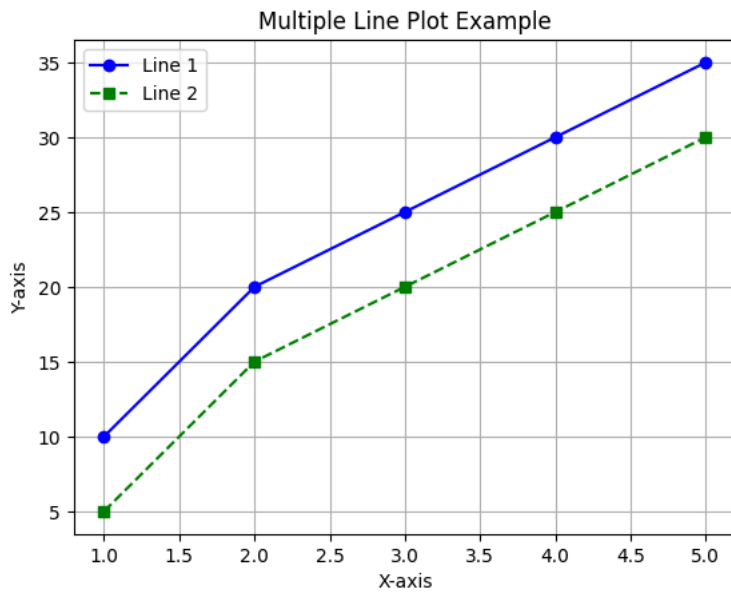
plt.plot(x, y1, label='Line 1', color='blue', marker='o')
plt.plot(x, y2, label='Line 2', color='green', linestyle='--', marker='s')

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

plt.legend()

plt.grid(True)

plt.show()
```



#9. Generate a Pandas DataFrame and filter rows where a column value is greater than a threshold.
import pandas as pd

```
data = {  
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],  
    'Marks': [88, 45, 72, 39, 95]  
}  
df = pd.DataFrame(data)  
  
filtered_df = df[df['Marks'] > 50]  
  
print(filtered_df)
```



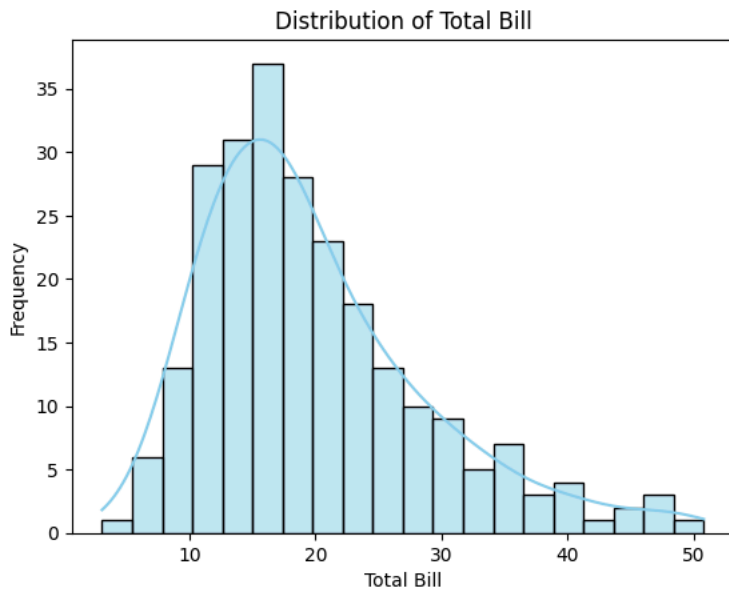
| | Name | Marks |
|---|---------|-------|
| 0 | Alice | 88 |
| 2 | Charlie | 72 |
| 4 | Eve | 95 |

#10. Create a histogram using Seaborn to visualize a distribution.
import seaborn as sns
import matplotlib.pyplot as plt

tips = sns.load_dataset('tips')

sns.histplot(data=tips, x='total_bill', bins=20, kde=True, color='skyblue')

plt.title('Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Frequency')
plt.show()



```
#11.Perform matrix multiplication using NumPy.
import numpy as np

A = np.array([[1, 2],
              [3, 4]])

B = np.array([[5, 6],
              [7, 8]])

result = np.dot(A, B)
print("Matrix A:\n", A)
print("Matrix B:\n", B)
print("Matrix Multiplication Result:\n", result)
```



```
Matrix A:
[[1 2]
 [3 4]]
Matrix B:
[[5 6]
 [7 8]]
Matrix Multiplication Result:
[[19 22]
 [43 50]]
```

```
#12. Use Pandas to load a CSV file and display its first 5 rows.
import pandas as pd
try:
    df = pd.read_csv('your_file.csv')

    print(df.head())

except FileNotFoundError:
    print("Error: The file 'your_file.csv' was not found. Please check the file path.")
except Exception as e:
    print(f"An error occurred: {e}")
```



```
Error: The file 'your_file.csv' was not found. Please check the file path.
```

```
#13. Create a 3D scatter plot using Plotly.
import plotly.graph_objects as go

x = [1, 2, 3, 4, 5]
y = [10, 15, 13, 17, 14]
z = [5, 6, 7, 8, 9]

fig = go.Figure(data=[go.Scatter3d(
    x=x,
    y=y,
    z=z,
    mode='markers',
    marker=dict(
```

```
        size=8,  
        color=z,  
        colorscale='Viridis',  
        opacity=0.8  
    )  
    ))  
  
fig.update_layout(  
    title='3D Scatter Plot Example',  
    scene=dict(  
        xaxis_title='X Axis',  
        yaxis_title='Y Axis',  
        zaxis_title='Z Axis'  
    ),  
    margin=dict(l=0, r=0, b=0, t=30)  
)  
  
fig.show()
```



3D Scatter Plot Example

