

ChatGPT

In a tabular format, tell me everything about matplotlib, seaborn, plotly. add useful internal module too.

Here's a comparison of **Matplotlib**, **Seaborn**, and **Plotly**, with additional useful internal modules for each library:

Aspect	Matplotlib	Seaborn	Plotly
Overview	A low-level, highly customizable plotting library in Python.	Built on top of Matplotlib, but provides a higher-level, more user-friendly interface.	An interactive, web-based plotting library with beautiful plots, suitable for dashboards and data visualization.
Type	Static plots, can be converted to interactive with other libraries.	Primarily static plots, but can work with interactive elements.	Primarily interactive plots for web-based applications.
Purpose	General-purpose data visualization.	Statistical data visualization, focusing on data analysis and distribution plots.	Interactive, high-quality, web-based visualizations for dashboards.
Plotting Style	Manual control over plot elements, flexible.	Attractive default themes, simplifies complex plots.	Interactive plots with zoom, pan, and hover features.
Syntax	Requires more boilerplate code.	Simple syntax, more abstracted.	Simple syntax with complex options for interactivity.
Themes/Styles	Offers customization with themes like <code>ggplot</code> , <code>seaborn</code> , <code>bmh</code> , etc.	Uses <code>matplotlib</code> under the hood but comes with custom, aesthetically pleasing themes.	Multiple built-in themes; includes more advanced styling for web-based features.
Plot Types	Line, scatter, bar, histograms, 3D, etc.	Scatterplots, bar plots, box plots, violin plots, heatmaps, etc.	Line, bar, scatter, bubble, pie, maps, 3D, and more interactive plots.
Interactivity	Basic interactivity (via <code>mpl_interactions</code> or	Limited interactivity but can use	Advanced interactivity with hover effects,

Aspect	Matplotlib	Seaborn	Plotly
	<code>matplotlib.widgets</code>).	<code>matplotlib</code> tools.	zoom, pan, animations, and tooltips.
Integration	Works with many Python libraries like NumPy, pandas, SciPy, etc.	Built on top of Matplotlib, integrates seamlessly with pandas and NumPy.	Integrates well with pandas, NumPy, and external libraries like Dash for app building.
Exporting	Can save plots as images (PNG, SVG, PDF, etc.).	Can save plots as images (PNG, SVG, PDF, etc.).	Exports to various formats like PNG, JPG, PDF, and interactive HTML files.
Community/Support	Well-established with extensive documentation.	Built on top of Matplotlib with strong community support.	Growing community, especially with web-based and dashboard-based applications.

Useful Internal Modules

Matplotlib

- `pyplot` (`matplotlib.pyplot`) – The main interface for creating plots.
- `axes` (`matplotlib.axes`) – Controls the layout and the scaling of the plot area.
- `figure` (`matplotlib.figure`) – Manages the creation of a figure (canvas).
- `colors` (`matplotlib.colors`) – Contains color maps and other color-related utilities.
- `patches` (`matplotlib.patches`) – Shapes like circles, rectangles, etc.
- `animation` (`matplotlib.animation`) – For creating animations.

Seaborn

- `seaborn.axes_style()` – Functions to change the default style of the plots (e.g., dark, whitegrid).
- `seaborn.load_dataset()` – Used for loading sample datasets for quick testing and examples.
- `seaborn.palettes` – Used for displaying the palette of colors.
- `seaborn.set_context()` – Adjusts the scale of the plot for different environments (paper, talk, poster).

- `seaborn.relplot()` – A high-level function for plotting relational data.
- `seaborn.catplot()` – High-level interface for categorical plots.

Plotly

- `plotly.graph_objects` – Main module for creating and customizing plots.
- `plotly.express` – Higher-level interface for simpler plotting.
- `plotly.subplots` – Module for creating subplots in Plotly.
- `plotly.io` – Module to handle input/output operations like saving and loading plots.
- `plotly.utils` – Utility functions like `PlotlyJSONEncoder` for serialization.
- `plotly.colors` – Provides tools for defining colors and palettes.

Each library has its strengths depending on the use case:

- **Matplotlib** for detailed, static, customizable plots.
- **Seaborn** for quick statistical visualization with good aesthetics.
- **Plotly** for creating interactive, web-based plots.

Give more details and explain everything. And give examples

Let's dive deeper into **Matplotlib**, **Seaborn**, and **Plotly**, providing more details and examples to explain the features, modules, and how to use them effectively.

Matplotlib

Overview

- **Matplotlib** is a highly customizable, low-level plotting library for Python. It's often considered the foundation of most Python plotting tools.
- **Matplotlib** works with basic, static plots (such as line charts, histograms, and bar charts), but it can be extended to create interactive plots using additional libraries.
- It is a powerful library when you need to have precise control over your plots, like customizing every detail of the graph.

Key Concepts

- **Figure:** Represents the entire window or canvas where all plot elements (axes, labels, etc.) are drawn.

- **Axes:** Represents a single plot or graph (e.g., a line plot or a scatter plot).
- **Artist:** A term used for any drawable object in the figure, like text, lines, shapes, etc.

Examples

1. Basic Plot

python

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y) # Line plot
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Basic Line Plot')
plt.show()
```

- Here, `plt.plot(x, y)` creates a simple line plot. You can customize axes and title using `xlabel()`, `ylabel()`, and `title()`.

2. Scatter Plot

python

```
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

plt.scatter(x, y, color='red') # Scatter plot
plt.show()
```

3. Bar Plot

python

```
x = ['A', 'B', 'C', 'D']
y = [10, 15, 7, 10]

plt.bar(x, y, color='blue')
plt.show()
```

Useful Modules

- **pyplot** (`matplotlib.pyplot`): The main interface for creating plots.
 - **colors** (`matplotlib.colors`): This module includes color maps and functions for managing colors.
 - **patches** (`matplotlib.patches`): For creating basic shapes like circles, rectangles, and polygons.
 - **animation** (`matplotlib.animation`): For creating dynamic and animated plots.
-

Seaborn

Overview

- **Seaborn** is a higher-level data visualization library built on top of **Matplotlib**. It's designed to make statistical plotting easier and more attractive.
- **Seaborn** integrates seamlessly with pandas DataFrames and simplifies the process of creating complex plots like violin plots, box plots, and pair plots.
- It also offers better default themes and color palettes, making it easier to create visually appealing graphs.

Key Concepts

- **Style**: Seaborn comes with built-in styles (e.g., `darkgrid`, `white`, etc.) that enhance the aesthetics of the plots.
- **Palette**: Colors in Seaborn can be easily customized through built-in color palettes like `deep`, `muted`, `dark`, etc.
- **Axes-level functions**: These functions generate plots based on an axes object.

Examples

1. Basic Plot with Seaborn

python

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

```
# Load a dataset from seaborn's collection
tips = sns.load_dataset("tips")

# Create a scatter plot
sns.scatterplot(x="total_bill", y="tip", data=tips)
plt.show()
```

2. Bar Plot

python

```
sns.barpplot(x="day", y="total_bill", data=tips)
plt.show()
```

3. Heatmap

python

```
flight_data = sns.load_dataset("flights")
pivot_data = flight_data.pivot_table(index="month", columns="year",
values="passengers")

sns.heatmap(pivot_data, annot=True, cmap="coolwarm")
plt.show()
```

4. Pair Plot (for exploring pairwise relationships)

python

```
sns.pairplot(tips)
plt.show()
```

Useful Modules

- **axes_style():** Function to set the style of the plots (e.g., `seaborn.set_style("whitegrid")`).
- **load_dataset():** Loads Seaborn's sample datasets, which are useful for testing and examples.
- **relplot():** High-level function for creating relational plots (scatter, line, etc.).
- **catplot():** High-level function for categorical plots like box plots, violin plots, etc.

Plotly

Overview

- **Plotly** is a powerful library for creating interactive plots that can be embedded in web applications or shared as standalone HTML files. It's built for interactive visualization, where users can zoom, pan, and hover to view tooltips.
- Unlike **Matplotlib** and **Seaborn**, **Plotly** plots are interactive by default. It also provides smooth animations and custom visualizations.

Key Concepts

- **Graph Objects:** A flexible approach to building complex figures where each element (traces, layout, annotations) is defined explicitly.
- **Express:** A high-level interface for simpler and quicker plotting with fewer lines of code.
- **Subplots:** Allows you to create multiple plots within a single figure.

Examples

1. Basic Line Plot with Plotly Express

python

```
import plotly.express as px

df = px.data.gapminder() # Example dataset
fig = px.line(df, x="year", y="lifeExp", title="Life Expectancy Over Time")
fig.show()
```

2. Scatter Plot

python

```
fig = px.scatter(df, x="gdpPercap", y="lifeExp", color="continent", title="GDP vs Life Expectancy")
fig.show()
```

3. Bar Plot

python

```
fig = px.bar(df, x="continent", y="pop", title="Population by Continent")
fig.show()
```

4. Interactive 3D Scatter Plot

python

```
fig = px.scatter_3d(df, x="gdpPercap", y="lifeExp", z="pop", color="continent")
fig.show()
```

Useful Modules

- **graph_objects**: Main module to build custom plots (e.g., `plotly.graph_objects.Figure`).
- **express**: Provides a higher-level, more simplified interface for making quick plots.
- **subplots**: Creates a grid of plots (e.g., for subplots in a multi-panel layout).
- **colors**: Defines built-in color scales (e.g., `plotly.colors.sequential`).

Comparison Summary

Feature	Matplotlib	Seaborn	Plotly
Use Case	Custom static plots, academic/research plotting.	Statistical visualization, easier and prettier plots.	Interactive, web-based data visualizations.
Customization	Very high – control over every plot element.	Moderate – fewer options but much easier to use.	Moderate – easy for interactivity, but limited customization.
Interactivity	Limited (needs additional libraries).	Limited interactivity.	Fully interactive (zoom, hover, pan, etc.).
Ease of Use	Requires more code for complex plots.	Simplifies the process of statistical plots.	Very simple for quick interactive plots.
Integration	Works with numpy, pandas, and other scientific libraries.	Built on top of matplotlib, works with pandas too.	Built for integration with web apps (Dash, Flask, etc.).

Matplotlib is best suited for creating static, detailed plots. **Seaborn** is great for statistical plots and simplicity, while **Plotly** shines when interactivity is required for web-based applications.

Tell me everything with explanation and examples about Numpy, Pandas, Matplotlib, Seaborn, Plotly, Bokeh. You should add the most useful modules and functions methods associated and use practical use cases and examples. i have should find everything about these topics when I refer to these notes. use the official documentation as well.

Let's go through **Numpy**, **Pandas**, **Matplotlib**, **Seaborn**, **Plotly**, and **Bokeh** in detail, covering the most important aspects, modules, and functions. I'll also provide practical use cases and examples for each library, ensuring you have everything you need when you refer to these notes.

Numpy

Overview:

NumPy (Numerical Python) is a powerful library for numerical computing. It provides support for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Key Concepts:

- **Array:** The central object in NumPy, which is more efficient than lists in Python for numerical operations.
- **Vectorization:** A technique where operations are performed on entire arrays instead of individual elements, leading to faster execution.
- **Broadcasting:** Allows NumPy to perform operations on arrays of different shapes.

Commonly Used Functions/Methods:

1. Creating Arrays:

- `np.array()`: Creates an array from a Python list.

```
python
```

```
import numpy as np
arr = np.array([1, 2, 3, 4])
```

- `np.zeros()`, `np.ones()`, `np.empty()`: Used to create arrays initialized to zero, one, or with uninitialized values.

python

```
zeros_arr = np.zeros((2, 3)) # 2x3 array of zeros
ones_arr = np.ones((3, 3))  # 3x3 array of ones
```

- `np.arange()`: Returns an array with evenly spaced values.

python

```
arr = np.arange(0, 10, 2) # [0, 2, 4, 6, 8]
```

2. Array Operations:

- **Element-wise operations:**

python

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

- **Matrix multiplication (`np.dot()`):**

python

```
mat1 = np.array([[1, 2], [3, 4]])
mat2 = np.array([[5, 6], [7, 8]])
result = np.dot(mat1, mat2)
```

3. Statistical Functions:

- `np.mean()`, `np.median()`, `np.std()`, `np.var()`: Calculate mean, median, standard deviation, and variance.

python

```
data = np.array([1, 2, 3, 4, 5])
mean_val = np.mean(data) # 3.0
```

4. Array Indexing & Slicing:

python

```
arr = np.array([1, 2, 3, 4, 5])
arr[1:4] # [2, 3, 4]
arr[::2] # [1, 3, 5]
```

Practical Use Case:

Numerical computations in data science: Suppose you have a dataset of test scores and you need to compute the mean and standard deviation of the scores. NumPy provides efficient ways to perform these operations.

python

```
scores = np.array([89, 78, 92, 85, 88, 76, 95])
mean_score = np.mean(scores) # 86.14
std_deviation = np.std(scores) # 6.29
```

Pandas

Overview:

Pandas is a high-level data manipulation library for Python, built on top of **NumPy**. It provides data structures like **DataFrame** and **Series** for handling structured data.

Key Concepts:

- **DataFrame:** Two-dimensional, size-mutable, and potentially heterogeneous tabular data structure.
- **Series:** A one-dimensional labeled array capable of holding any data type.

Commonly Used Functions/Methods:

1. Creating DataFrames and Series:

- `pd.DataFrame()` : Creates a DataFrame from a dictionary, list, or NumPy array.

python

```
import pandas as pd
data = {'Name': ['Alice', 'Bob'], 'Age': [24, 27]}
df = pd.DataFrame(data)
```

2. Data Selection:

- **Selecting columns:**

python

```
df['Age'] # Returns the 'Age' column
```

- **Selecting rows by index:**

python

```
df.iloc[0] # Selects the first row
df.loc[0] # Selects row based on label
```

3. Data Aggregation:

- `df.groupby()` : Group data by certain columns and apply aggregation functions.

python

```
df.groupby('Name')['Age'].mean()
```

4. Handling Missing Data:

- `df.isna()`, `df.dropna()`, `df.fillna()` : Detect, drop, or replace missing data.

python

```
df.dropna() # Drops rows with missing values
df.fillna(0) # Fills missing values with 0
```

5. Merging and Joining:

- `pd.merge()` : Merge two DataFrames based on a key.

python

```
df1 = pd.DataFrame({'key': ['A', 'B'], 'value': [1, 2]})
df2 = pd.DataFrame({'key': ['A', 'C'], 'value': [3, 4]})
merged_df = pd.merge(df1, df2, on='key')
```

Practical Use Case:

Cleaning and analyzing a dataset: Suppose you have sales data with missing values, and you want to clean it up and analyze the sales by product.

python

```
sales = pd.DataFrame({
    'Product': ['A', 'B', 'C', 'D'],
    'Sales': [100, None, 300, 200]
})

# Fill missing sales with 0
sales['Sales'].fillna(0, inplace=True)

# Calculate total sales
total_sales = sales['Sales'].sum()
```

Matplotlib

Overview:

Matplotlib is the foundational plotting library in Python. It is highly customizable and used for creating static plots such as line plots, bar plots, and histograms.

Key Concepts:

- **Figure:** The overall window or canvas for the plot.
- **Axes:** The plot area, which contains the actual graph elements like lines or bars.
- **Artist:** Any object that can be drawn on the plot (e.g., lines, text, shapes).

Commonly Used Functions/Methods:

1. Basic Plot:

python

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]
plt.plot(x, y)
plt.show()
```

2. Customizing Plots:

- Adding labels, title, and grid:

python

```
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Basic Plot')
plt.grid(True)
plt.show()
```

3. Subplots:

python

```
fig, axs = plt.subplots(1, 2) # 1 row, 2 columns
axs[0].plot(x, y)
axs[1].bar(x, y)
plt.show()
```

4. Saving a Plot:

python

```
plt.plot(x, y)
plt.savefig('plot.png')
```

Practical Use Case:

Visualizing trends: You may want to visualize the sales over time using a line plot.

python

```
months = ['Jan', 'Feb', 'Mar', 'Apr']
sales = [150, 200, 250, 300]
```

```
plt.plot(months, sales)
plt.xlabel('Month')
plt.ylabel('Sales')
plt.title('Sales Over Time')
plt.show()
```

Seaborn

Overview:

Seaborn is built on top of **Matplotlib** and simplifies the creation of beautiful, informative statistical graphics.

Key Concepts:

- **Statistical Plots:** Includes functions for creating plots like histograms, violin plots, and box plots with minimal code.
- **Themes:** Seaborn comes with built-in themes to enhance the visual appeal of plots.
- **Facet Grids:** Helps create multi-plot grids for visualizing data distributions across categories.

Commonly Used Functions/Methods:

1. Categorical Plots:

python

```
import seaborn as sns
sns.boxplot(x='species', y='sepal_length', data=iris)
```

2. Distribution Plots:

python

```
sns.histplot(data, kde=True)
```

3. Pair Plot:

python

```
sns.pairplot(iris)
```

4. Correlation Heatmap:

```
python
```

```
corr = iris.corr()  
sns.heatmap(corr, annot=True)
```

Practical Use Case:

Exploring relationships: Visualizing relationships between multiple variables in a dataset.

```
python
```

```
sns.pairplot(iris, hue='species')  
plt.show()
```

Plotly

Overview:

Plotly is a powerful library for creating interactive, web-ready visualizations, supporting a wide variety of plots, from basic to complex.

Key Concepts:

- **Graph Objects:** Provides full control over the plot elements.
- **Express:** A high-level interface to make complex plots with less code.
- **Interactivity:** Built for creating plots with hover effects, zooming, and more.

Commonly Used Functions/Methods:

1. Basic Plot:

```
python
```

```
import plotly.express as px  
fig = px.scatter(df, x="gdpPercap", y="lifeExp")
```



```
fig.show()
```

2. 3D Plot:

```
python
```

```
fig = px.scatter_3d(df, x="gdpPercap", y="lifeExp", z="pop")  
fig.show()
```

3. Bar Plot:

```
python
```

```
fig = px.bar(df, x="continent", y="pop")  
fig.show()
```

Practical Use Case:

Interactive dashboards: Displaying interactive plots for data analysis and presentation.

```
python
```

```
fig = px.bar(df, x='Category', y='Value', title="Category-wise Value Distribution")  
fig.show()
```

Bokeh

Overview:

Bokeh is a library for creating interactive plots and dashboards, with a focus on modern web browsers.

Key Concepts:

- **Layouts:** Arrange multiple plots in a web page layout.
- **Widgets:** Interactive controls (like sliders, dropdowns) to update plots dynamically.

Commonly Used Functions/Methods:

1. Basic Plot:

python

```
from bokeh.plotting import figure, show
p = figure(title="Simple Plot", x_axis_label='X', y_axis_label='Y')
p.line([1, 2, 3, 4], [4, 6, 2, 3], legend_label="Line", line_width=2)
show(p)
```

2. Interactive Plot with Widgets:

python

```
from bokeh.layouts import column
from bokeh.models import Slider

slider = Slider(start=0, end=10, value=1, step=0.1, title="Value")
layout = column(slider, plot)
show(layout)
```

Practical Use Case:

Interactive visualizations: Use Bokeh to create plots that can be embedded into web applications or dashboards.

python

```
from bokeh.plotting import figure, output_file, save
output_file("line_plot.html")
p = figure(title="Line Plot Example", x_axis_label='X', y_axis_label='Y')
p.line([1, 2, 3, 4], [10, 20, 30, 40], line_color="green")
save(p)
```

Summary:

- **NumPy:** Efficient numerical array operations (vectorization, broadcasting).
- **Pandas:** Data manipulation with DataFrames (aggregation, merging, cleaning).
- **Matplotlib:** Basic static plots with customizability.
- **Seaborn:** Statistical plots and enhanced visualizations.
- **Plotly:** Interactive, web-based plots.

- **Bokeh:** Interactive visualizations with web integration.

These libraries are powerful tools in data science, visualization, and web applications. By understanding their strengths and practical applications, you'll be able to tackle a wide variety of data visualization tasks.

```
from bokeh.layouts import column
from bokeh.models import Slider

slider = Slider(start=0, end=10, value=1, step=0.1,
               title="Value")
layout = column(slider, plot)
show(layout)
```

Its not working. plot not defined in line 5

You're right! The error is occurring because the `plot` variable has not been defined before it's used in the `column()` layout function.

To fix the issue, you need to create a plot using Bokeh's `figure()` function and then add it to the layout alongside the `Slider`. Here's the corrected code:

Corrected Code:

python

```
from bokeh.plotting import figure, show
from bokeh.layouts import column
from bokeh.models import Slider

# Create a plot
plot = figure(title="Simple Plot", x_axis_label='X', y_axis_label='Y')
plot.line([1, 2, 3, 4], [4, 6, 2, 3], legend_label="Line", line_width=2)

# Create a slider
slider = Slider(start=0, end=10, value=1, step=0.1, title="Value")

# Combine the slider and plot into a layout
layout = column(slider, plot)

# Show the layout
show(layout)
```

Explanation:

1. `figure()` : This function creates a Bokeh plot. In this case, the plot is a simple line plot defined by the `x` and `y` values `[1, 2, 3, 4]` and `[4, 6, 2, 3]` respectively.
2. `Slider()` : Creates a slider widget that allows you to interact with the plot (you can extend this functionality further if needed).
3. `column()` : The `column` layout arranges the `slider` and `plot` vertically on the page.
4. `show(layout)` : Displays the interactive layout in a browser.

Next Steps:

To make the slider interact with the plot (e.g., change the plot's data based on the slider value), you would need to add a callback function to update the plot dynamically. Let me know if you want help with that as well!

Make a cheat sheet note on all the functions and useful features of all the concepts

Here's a **cheat sheet** summarizing key functions and useful features for **NumPy**, **Pandas**, **Matplotlib**, **Seaborn**, **Plotly**, and **Bokeh**. These are essential for quick reference and working efficiently with these libraries.

NumPy Cheat Sheet

1. Creating Arrays

- `np.array(list)` : Convert Python list into a NumPy array.

python

```
arr = np.array([1, 2, 3])
```

- `np.zeros(shape)` : Create an array filled with zeros.

python

```
arr = np.zeros((2, 3))
```

- `np.ones(shape)` : Create an array filled with ones.

python

```
arr = np.ones((2, 3))
```

- `np.arange(start, stop, step)` : Create an array with evenly spaced values.

python

```
arr = np.arange(0, 10, 2)
```

2. Array Operations

- `np.add(a, b)` , `a + b` : Element-wise addition of arrays.
- `np.subtract(a, b)` , `a - b` : Element-wise subtraction.
- `np.multiply(a, b)` , `a * b` : Element-wise multiplication.
- `np.dot(a, b)` : Matrix multiplication.

python

```
result = np.dot(a, b)
```

3. Array Manipulation

- `arr.shape` : Get the shape of an array.

python

```
arr.shape # (2, 3)
```

- `arr.reshape(new_shape)` : Reshape the array.

python

```
arr = arr.reshape(3, 2)
```

- `arr.flatten()` : Flatten a multi-dimensional array into one dimension.

4. Statistical Operations

- `np.mean(arr)` : Mean of the array.

- `np.median(arr)` : Median of the array.
- `np.std(arr)` : Standard deviation of the array.
- `np.var(arr)` : Variance of the array.
- `np.min(arr)` , `np.max(arr)` : Minimum/Maximum value of the array.

5. Random Numbers

- `np.random.rand(n)` : Generate an array of random numbers between 0 and 1.
- `np.random.randn(n)` : Generate an array of random numbers from a normal distribution.

Pandas Cheat Sheet

1. Creating DataFrames

- `pd.DataFrame(data)` : Create a DataFrame from a dictionary, list, or NumPy array.

python

```
df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
```

2. Data Selection

- `df['col_name']` : Select a column.

python

```
df['A']
```

- `df.loc[row_index]` : Select rows by label.

python

```
df.loc[0] # First row
```

- `df.iloc[row_index]` : Select rows by position.

python

```
df.iloc[0] # First row
```

3. Data Aggregation

- `df.groupby('col_name')` : Group by a column.

```
python
```

```
grouped = df.groupby('col_name').mean()
```

- `df.mean()`, `df.sum()`, `df.count()` : Basic aggregation functions.

4. Handling Missing Data

- `df.isna()` : Detect missing values.
- `df.dropna()` : Drop rows with missing values.
- `df.fillna(value)` : Replace missing values with a specific value.

```
python
```

```
df.fillna(0)
```

5. Merging and Joining

- `pd.merge(df1, df2, on='key')` : Merge two DataFrames on a key.

```
python
```

```
merged_df = pd.merge(df1, df2, on='key')
```

6. Sorting

- `df.sort_values('col_name')` : Sort DataFrame by column.

```
python
```

```
df.sort_values('Age')
```

1. Basic Plotting

- `plt.plot(x, y)` : Basic line plot.

python

```
plt.plot(x, y)
plt.show()
```

- `plt.scatter(x, y)` : Scatter plot.

python

```
plt.scatter(x, y)
```

2. Customization

- `plt.title('Title')` : Add title.
- `plt.xlabel('X-axis label')`, `plt.ylabel('Y-axis label')` : Add axis labels.
- `plt.grid(True)` : Show grid lines.

3. Subplots

- `plt.subplots(nrows, ncols)` : Create multiple plots.

python

```
fig, ax = plt.subplots(2, 2)
```

4. Saving Plots

- `plt.savefig('filename.png')` : Save the plot to a file.

python

```
plt.savefig('plot.png')
```

Seaborn Cheat Sheet

1. Basic Plots

- `sns.lineplot(x, y)` : Line plot.
- `sns.barplot(x, y)` : Bar plot.
- `sns.boxplot(x, y)` : Box plot.
- `sns.histplot(data)` : Histogram.

2. Advanced Plots

- `sns.pairplot(df)` : Pairwise relationships plot.

python

```
sns.pairplot(df)
```

- `sns.heatmap(data)` : Heatmap for correlation.

python

```
sns.heatmap(df.corr(), annot=True)
```

3. Categorical Data

- `sns.catplot(x, y, data=df, kind='bar')` : Categorical plot (bar, box, etc.).

python

```
sns.catplot(x="day", y="total_bill", data=tips, kind="box")
```

4. Styling

- `sns.set_style('whitegrid')` : Set plot style.
- `sns.set_palette('deep')` : Set color palette.

Plotly Cheat Sheet

1. Basic Plots

- `px.scatter(df, x, y)` : Scatter plot.

python

```
fig = px.scatter(df, x="gdpPercap", y="lifeExp")
fig.show()
```

- `px.bar(df, x, y)` : Bar plot.

```
python
```

```
fig = px.bar(df, x="continent", y="pop")
```

2. 3D Plots

- `px.scatter_3d(df, x, y, z)` : 3D scatter plot.

```
python
```

```
fig = px.scatter_3d(df, x="gdpPercap", y="lifeExp", z="pop")
fig.show()
```

3. Subplots

- `make_subplots()` : Create subplots with multiple charts.

```
python
```

```
from plotly.subplots import make_subplots
fig = make_subplots(rows=1, cols=2)
```

4. Interactivity

- `fig.update_layout()` : Update layout properties like titles, axes, etc.

```
python
```

```
fig.update_layout(title="GDP vs Life Expectancy")
```

Bokeh Cheat Sheet

1. Basic Plot

- `figure()` : Create a new plot.

python

```
from bokeh.plotting import figure, show
plot = figure(title="Simple Plot", x_axis_label='X', y_axis_label='Y')
plot.line([1, 2, 3, 4], [4, 6, 2, 3], legend_label="Line", line_width=2)
show(plot)
```

2. Widgets

- `Slider(start, end, value)` : Create a slider.

python

```
from bokeh.models import Slider
slider = Slider(start=0, end=10, value=1, step=0.1, title="Value")
```

- `TextInput(value)` : Create a text input widget.

python

```
from bokeh.models import TextInput
text_input = TextInput(value="default text")
```

3. Layouts

- `column()`, `row()` : Arrange plots and widgets vertically or horizontally.

python

```
from bokeh.layouts import column
layout = column(slider, plot)
show(layout)
```

4. Interactive Plot

- `update()` : Dynamically update plot based on widget interaction.

Summary of Key Functions

- **NumPy**: Array creation, vectorized operations, random number generation.

- **Pandas:** DataFrame manipulation, grouping, aggregation, merging, and handling missing data.
- **Matplotlib:** Static plots, customization, subplots, and saving plots.
- **Seaborn:** Statistical plots, customization, and categorical plots.
- **Plotly:** Interactive plots, 3D plots, and subplots.
- **Bokeh:** Interactive plots, layouts, and widgets.

This cheat sheet covers key functionalities and provides examples for quick reference. You can dive deeper into each function by referring to the official documentation of the libraries.