Matplotlib

Matplotlib is a powerful visualization library in python that allows you to create a wide range of plots and charts. In this cheat sheet, we will explore some of the most commonly used functions and techniques in Matplotlib
to import the `matplotlib.pyplot` module:

```
import matplotlib.pyplot as plt
```

Displaying Plots in Matplotlib:

- Matplotlib provides versatile tools for creating visualizations in Python, offering support for a wide range of plot types and extensive customization options.
- Viewing Matplotlib plots is context-based, with different best practices for plotting from a script, an IPython shell, or a Jupyter notebook.
- In a script, the `plt.show()` command is crucial for displaying plots and should only be used once per Python session, placed at the end of the script.
- Interactive plotting in an IPython shell can be enabled by specifying Matplotlib mode with the `%matplotlib` magic command.
- The Jupyter Notebook allows for interactive plotting using the `%matplotlib` command, with options to produce either interactive or static plots embedded within the notebook.

- **Matplotlib Object Hierarchy**:

  - Matplotlib plots are structured hierarchically:
    - **Figure**: Outermost container for a plot, containing one or more Axes.
    - **Axes**: Represents an individual plot with elements like axis, tick marks, lines, and labels.

- **Matplotlib APIs**:

  - **Pyplot API**:
    - MATLAB-style interface for quick, state-based plotting.
    - Uses functions like `plt.plot()` and `plt.subplot()` to create and modify plots.
    - Implicitly manages figures and axes, making it easy for quick visualizations.
    - Useful for interactive plotting and simple visualizations.

- **Object-Oriented (OO) API**:
    - More powerful and flexible than the Pyplot API.
    - Relies on explicit Figure and Axes objects.
    - Allows fine-grained control over plot elements and layout.
    - Suited for complex plotting scenarios requiring precise customization and multiple subplots.

- **Advantages of Each API**:

    - **Pyplot API**:
        - Ideal for quick, straightforward plots.
        - Simplifies interactive plotting and visualization tasks.
        - Convenient for beginners and rapid prototyping.
    - **Object-Oriented API**:
        - Offers greater control and customization.
        - Enables creation of complex layouts and multi-panel plots.
        - Supports advanced features like shared axes, inset plots, and customized legends.

- **Usage Recommendations**:

    - Choose **Pyplot API** for quick exploratory data analysis and simple plotting tasks.
    - Prefer **Object-Oriented API** for projects requiring detailed control over plot elements, complex layouts, and reusable plotting functions.

- **Formatting and Styling**:

    - Both APIs support formatting options for plot styles, colors, markers, and line types.
    - Formats are inspired by MATLAB, providing flexibility in plot appearance and clarity.

- **Best Practices**:

    - Avoid mixing Pyplot and Object-Oriented APIs within the same script for clarity and maintainability.
    - Use Pyplot for rapid prototyping and quick visual feedback, and transition to Object-Oriented API for production-level plots and customization.

- **Figure and Subplots in Matplotlib**:

    - **Figure**: Top-level container for all plot elements. Created with `plt.figure()`.
    - **Subplots**: Divisions within a figure where plots reside, defined using `fig.add_subplot()`.

- **Creating Plots**:

- Use `plt.plot()` or `ax.plot()` to create line plots, where data points are connected by lines.
- Specify x and y coordinates explicitly or use implicit numbering for x-values.

- **Multiline Plots**:

  - Multiple plots can be displayed in the same figure using successive `plt.plot()` or `ax.plot()` commands.
  - All plots are shown together when `plt.show()` is called.

- **Parts of a Plot**:

  - **Title**: Describes the purpose or content of the plot.
  - **Legend**: Explains the meaning of symbols or colors used in the plot.
  - **Grid**: A network of equally spaced horizontal and vertical lines that help align data points.
  - **Axis**: Vertical (y-axis) and horizontal (x-axis) lines that establish the scale of the plot.
  - **Labels**: Text that identifies each axis and its units.

- **Saving Plots**:

  - Use `fig.savefig('filename.format')` to save a figure as an image file (e.g., PNG, JPEG, PDF).
  - File format is determined by the extension provided in the filename.

- **Different Plot Types**:

  - **Line Plot**: Connects data points with straight lines to show trends over time or space.
  - **Scatter Plot**: Represents individual data points as dots or markers to show relationships between variables.
  - **Histogram**: Displays the distribution of numerical data using bars of varying heights.
  - **Bar Chart**: Uses rectangular bars to compare quantities across different categories.
  - **Pie Chart**: Displays proportions of a whole using wedges of a circle.
  - **Boxplot**: Summarizes the distribution of a dataset using median, quartiles, and outliers.
  - **Area Chart**: Fills the area between lines and the x-axis with color or shading to emphasize changes over time.