Matplotlib Cheatsheet

Navigate the world of data visualization with this comprehensive Matplotlib cheatsheet, covering fundamental plotting, customization, and advanced techniques.

Getting Started & Basics

Installation & Import

Install:

```
pip install matplotlib
```

Standard Import:

```
import matplotlib.pyplot as plt
import numpy as np # Often used with Matplotlib
```

Enable interactive plots (e.g., in Jupyter):

```
%matplotlib inline # For static images
# %matplotlib notebook # For interactive plots
```

Check version:

```
import matplotlib
print(matplotlib.__version__)
```

Clearing the current figure/axes:

```
plt.clf() # Clear figure
plt.cla() # Clear axes
plt.close() # Close figure window
plt.close('all') # Close all figure windows
```

The Matplotlib Figure Hierarchy

Figure: The top-level container for all plot elements. It holds one or more Axes.

Axes: The area where the data is plotted. Each Axes has an x-axis, y-axis (and potentially z-axis), titles, labels, and tick marks. A Figure can contain multiple Axes (subplots).

Axis: These are the number-line objects that control the limits of the graph (e.g., x-axis, y-axis). They contain ticks and ticklabels.

Artist: Everything visible on the Figure is an Artist (Title, Labels, Lines, Text, etc.). Most Artists are tied to an Axes.

Recommended Practice: Use the Object-Oriented (OO) interface where you explicitly create Figure and Axes objects, rather than relying solely on the state-based `pyplot` interface.

Basic Plotting with pyplot

Simple Line Plot	`plt.plot([1, 2, 3, 4])`
	`plt.ylabel('some numbers')`
	`plt.show()`

Plotting x vs y	`plt.plot([1, 2, 3, 4], [1, 4, 9, 16])` `plt.show()`

Plotting multiple lines	`t = np.arange(0., 5., 0.2)`
	`plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')`
	`plt.show()`

Creating a new figure	`plt.figure(figsize=(8, 6), dpi=100)`
Showing the plot	`plt.show()`
Adding grid	`plt.grid(True)`

Creating Figures & Axes (OO Interface)

Create Figure and one Axes	`fig, ax = plt.subplots()`
Create Figure and multiple Axes (2x2 grid)	`fig, axes = plt.subplots(2, 2)` # axes is a 2D numpy array of Axes objects
Set size and DPI	`fig, ax = plt.subplots(figsize=(10, 5), dpi=150)`
Plotting on an Axes object	`ax.plot([1, 2, 3, 4], [1, 4, 9, 16])` `fig.show() # Use fig.show() or plt.show()`

Adding a single Axes to an existing Figure	`fig = plt.figure()`
	`ax = fig.add_subplot(111) # 1 row, 1 col, 1st plot`
	`ax.plot([1, 2, 3])`
	`plt.show()`

Adding Titles, Labels, Legends

Figure Title (`pyplot`)	`plt.suptitle('Overall Title')`
Axes Title (`pyplot`)	`plt.title('Plot Title')`
X/Y Labels (`pyplot`)	`plt.xlabel('X-axis Label')` `plt.ylabel('Y-axis Label')`
Figure Title (OO)	`fig.suptitle('Overall Title')`
Axes Title (OO)	`ax.set_title('Plot Title')`
X/Y Labels (OO)	`ax.set_xlabel('X-axis Label')` `ax.set_ylabel('Y-axis Label')`
Adding Legend (`pyplot`)	`plt.plot(x, y, label='Data Series 1')` `plt.legend()`
Adding Legend (OO)	`ax.plot(x, y, label='Data Series 1')` `ax.legend()`

Line Plot Customization

Basic plot with customizations	`plt.plot(x, y, color='green', linestyle='dashed', linewidth=3, marker='o', markerfacecolor='blue', markersize=12)`

Short-hand format string	`# color marker linestyle`
	`plt.plot(x, y, 'ro-') # Red circles with solid line`
	`plt.plot(x, y, 'b-.') # Blue dots with dash-dot line`

Line Style options	`'-'` (solid), `'--'` (dashed), `'-.'` (dash-dot), `':'` (dotted)
Marker Style options	`'.'` (point), `','` (pixel), `'o'` (circle), `'v'` (triangle down), `'^'` (triangle up), `'<'` (triangle left), `'>'` (triangle right), `'s'` (square), `'p'` (pentagon), `'*'` (star), `'+'` (plus), `'x'` (x), `'D'` (diamond), `'h'` (hexagon1), `'H'` (hexagon2), `'_'` (hline), `'|'` (vline)
Color options	`'b'` (blue), `'g'` (green), `'r'` (red), `'c'` (cyan), `'m'` (magenta), `'y'` (yellow), `'k'` (black), `'w'` (white) Hex codes: `'#1f77b4'` Named colors: `'skyblue'`, `'indianred'` etc. Tableau Colors: `'tab:blue'`, `'tab:orange'`, etc.
Adjusting opacity (alpha)	`plt.plot(x, y, alpha=0.5) # 50% transparent`

Plot Types & Data Visualization

Scatter Plots

Basic Scatter Plot (`pyplot`)	`plt.scatter(x, y)` `plt.show()`

Scatter Plot with size and color variation (`pyplot`)	`# s=size, c=color (can be array)`
	`plt.scatter(x, y, s=sizes, c=colors, alpha=0.5)`
	`plt.colorbar(label='Color Intensity')`
	`plt.show()`

Basic Scatter Plot (OO)	`fig, ax = plt.subplots()`
	`ax.scatter(x, y)`
	`plt.show()`

Scatter Plot with size and color variation (OO)	`fig, ax = plt.subplots()`
	`scatter = ax.scatter(x, y, s=sizes, c=colors, alpha=0.5)`
	`fig.colorbar(scatter, ax=ax, label='Color Intensity')`
	`plt.show()`

Adding markers/styles (similar to `plot`)	`plt.scatter(x, y, marker='x', color='red')`

Bar Charts

Basic Vertical Bar Chart (`pyplot`)	`categories = ['A', 'B', 'C']`
	`values = [10, 15, 7]`
	`plt.bar(categories, values)`
	`plt.show()`

Basic Horizontal Bar Chart (`pyplot`)	`plt.barh(categories, values)` `plt.show()`
Customizing Bars	`plt.bar(categories, values,` `color='skyblue', width=0.6)`

Basic Bar Chart (OO)	`fig, ax = plt.subplots()`
	`ax.bar(categories, values)`
	`plt.show()`

Grouped Bar Chart Example	`labels = ['G1', 'G2', 'G3']`
	`men_means = [20, 35, 30]`
	`women_means = [25, 32, 34]`
	`width = 0.35`
	`x = np.arange(len(labels))`
	`fig, ax = plt.subplots()`
	`rects1 = ax.bar(x - width/2,`
	`men_means, width, label='Men')`
	`rects2 = ax.bar(x + width/2,`
	`women_means, width, label='Women')`
	`ax.set_xticks(x)`
	`ax.set_xticklabels(labels)`
	`ax.legend()`
	`plt.show()`

Stacked Bar Chart Example	`labels = ['G1', 'G2', 'G3']`
	`men_means = [20, 35, 30]`
	`women_means = [25, 32, 34]`
	`fig, ax = plt.subplots()`
	`ax.bar(labels, men_means, label='Men')`
	`ax.bar(labels, women_means,`
	`bottom=men_means, label='Women')`
	`ax.legend()`
	`plt.show()`

Histograms

Basic Histogram (`pyplot`)	`data = np.random.randn(1000)`
	`plt.hist(data, bins=30)`
	`plt.show()`

Customizing Bins & Density	`# bins can be int or sequence`
	`# density=True for normalized histogram`
	`plt.hist(data, bins='auto', density=True,`
	`color='lightblue', edgecolor='black')`
	`plt.show()`

Basic Histogram (OO)	`fig, ax = plt.subplots()`
	`data = np.random.randn(1000)`
	`ax.hist(data, bins=30)`
	`plt.show()`

Plotting multiple histograms	`data1 = np.random.normal(0, 1, 1000)`
	`data2 = np.random.normal(3, 2, 1000)`
	`plt.hist(data1, bins=30, alpha=0.5,`
	`label='Data 1')`
	`plt.hist(data2, bins=30, alpha=0.5,`
	`label='Data 2')`
	`plt.legend()`
	`plt.show()`

Box Plots

Basic Box Plot (`pyplot`)	`data = [np.random.rand(100),`
	`np.random.rand(100) * 2]`
	`plt.boxplot(data)`
	`plt.show()`

Horizontal Box Plot	`plt.boxplot(data, vert=False)` `plt.show()`

Customizing Box Plots	`# showmeans=True, showfliers=False`
	`plt.boxplot(data, patch_artist=True,`
	`medianprops=dict(color='red'))`
	`plt.show()`

Basic Box Plot (OO)	`fig, ax = plt.subplots()`
	`ax.boxplot(data)`
	`plt.show()`

Pie Charts

Basic Pie Chart (`pyplot`)	`sizes = [15, 30, 45, 10]`
	`labels = ['A', 'B', 'C', 'D']`
	`plt.pie(sizes, labels=labels, autopct='%1.1f%%')`
	`plt.axis('equal') # Equal aspect ratio ensures pie is circular.`
	`plt.show()`

Customizing Pie Chart	`explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (B)`
	`plt.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)`
	`plt.axis('equal')`
	`plt.show()`

Basic Pie Chart (OO)	`fig, ax = plt.subplots()`
	`ax.pie(sizes, labels=labels, autopct='%1.1f%%')`
	`ax.axis('equal')`
	`plt.show()`

Images & Heatmaps

Displaying an image (`pyplot`)	`# img = mpimg.imread('stinkbug.png')`
	`# plt.imshow(img)`
	`# plt.show()`

Displaying an array as image/heatmap (`pyplot`)	`import numpy as np`
	`data = np.random.rand(10, 10)`
	`plt.imshow(data, cmap='hot', interpolation='nearest')`
	`plt.colorbar(label='Value')`
	`plt.show()`

Displaying an array as image/heatmap (OO)	`fig, ax = plt.subplots()`
	`data = np.random.rand(10, 10)`
	`im = ax.imshow(data, cmap='viridis')`
	`fig.colorbar(im, ax=ax, label='Value')`
	`plt.show()`

Setting aspect ratio	`plt.imshow(data, aspect='auto') # Or 'equal'`
Common colormaps (`cmap`)	`'viridis'`, `'plasma'`, `'inferno'`, `'magma'`, `'cividis'`, `'Greys'`, `'Purples'`, `'Blues'`, `'Greens'`, `'Oranges'`, `'Reds'`, `'YlOrRd'`, `'YlGnBu'`, `'coolwarm'`, `'RdBu'`, `'seismic'`, `'terrain'`, `'ocean'`, `'hot'`

Customization & Layout

Colors, Markers, Linestyles

Setting color by name or hex	`ax.plot(x, y, color='steelblue')` `ax.scatter(x, y, c='#FF5733')`
Setting marker style	`ax.plot(x, y, marker='s', markersize=8)` `ax.scatter(x, y, marker='^')`
Setting linestyle	`ax.plot(x, y, linestyle='--')` `ax.plot(x, y, ls=':')`
Setting linewidth	`ax.plot(x, y, linewidth=2)` `ax.plot(x, y, lw=1.5)`
Setting opacity	`ax.plot(x, y, alpha=0.7)` `ax.scatter(x, y, alpha=0.3)`

Using colormaps for line segments	`points = np.array([x, y]).T.reshape(-1, 1, 2)`
	`segments = np.concatenate([points[:-1], points[1:]], axis=1)`
	`from matplotlib.collections import LineCollection`
	`from matplotlib.colors import ListedColormap, Normalize`
	
	`# Create a colormap`
	`vals = np.linspace(0, 10, len(y))`
	
	`fig, ax = plt.subplots()`
	`norm = Normalize(vmin=vals.min(), vmax=vals.max())`
	`lc = LineCollection(segments, cmap='viridis', norm=norm)`
	`lc.set_array(vals)`
	`lc.set_linewidth(2)`
	`line = ax.add_collection(lc)`
	`fig.colorbar(line, ax=ax)`
	`ax.autoscale_view()`
	`plt.show()`

Axis Limits, Ticks, Scales

Set X/Y limits (`pyplot`)	`plt.xlim(0, 10)` `plt.ylim(-5, 5)`
Set X/Y limits (OO)	`ax.set_xlim(0, 10)` `ax.set_ylim(-5, 5)`
Set X/Y ticks (`pyplot`)	`plt.xticks([0, 2, 4, 6, 8, 10])` `plt.yticks([-5, 0, 5])`
Set X/Y ticks (OO)	`ax.set_xticks([0, 2, 4, 6, 8, 10])` `ax.set_yticks([-5, 0, 5])`
Set tick labels	`ax.set_xticklabels(['Low', 'Mid', 'High'])`

Logarithmic scale	`ax.set_xscale('log')`
	`ax.set_yscale('log')`
	`# or plt.xscale('log') etc.`

Adding minor ticks	`ax.minorticks_on()`

Annotations and Text

Add simple text (`pyplot`)	`plt.text(x_coord, y_coord, 'Some Text', fontsize=12)` `plt.show()`
Add simple text (OO)	`ax.text(x_coord, y_coord, 'Some Text', color='red')` `plt.show()`
Add annotation with arrow (`pyplot`)	`plt.annotate('Peak', xy=(peak_x, peak_y), xytext=(peak_x+1, peak_y+10), arrowprops=dict(facecolor='black', shrink=0.05))` `plt.show()`
Add annotation with arrow (OO)	`ax.annotate('Peak', xy=(peak_x, peak_y), xytext=(peak_x+1, peak_y+10), arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=.2'))` `plt.show()`

Text alignment (`ha`, `va`)	`ax.text(x, y, 'Left Aligned', ha='left')`
	`ax.text(x, y, 'Centered', ha='center', va='center')`
	`ax.text(x, y, 'Top Aligned', va='top')`

Using TeX for text	`ax.set_xlabel('Time ($s$)', fontsize=12)`
	`ax.set_title('Motion of a Particle: $v(t) = at^2 + b$', fontsize=14)`
	`# Ensure you have a TeX distribution installed or use mathtext`
	`plt.rcParams['text.usetex'] = True # Requires LaTeX install`
	`plt.rcParams['text.usetex'] = False # Use mathtext`

Working with Subplots

Using `plt.subplot()` (Grid layout)	`# 1 row, 2 cols, 1st plot`
	`plt.subplot(1, 2, 1)`
	`plt.plot(x, y1)`
	`# 1 row, 2 cols, 2nd plot`
	`plt.subplot(1, 2, 2)`
	`plt.plot(x, y2)`
	`plt.show()`

Using `fig.add_subplot()` (More control)	`fig = plt.figure()`
	`ax1 = fig.add_subplot(2, 1, 1)`
	`# 2 rows, 1 col, 1st plot`
	`ax1.plot(x, y1)`
	`ax2 = fig.add_subplot(2, 1, 2)`
	`# 2 rows, 1 col, 2nd plot`
	`ax2.plot(x, y2)`
	`plt.show()`

Using `plt.subplots()` (Recommended OO approach)	`# Creates figure and 2x2 grid of axes`
	`fig, axes = plt.subplots(2, 2)`
	`axes[0, 0].plot(x, y1)`
	`axes[0, 1].plot(x, y2)`
	`axes[1, 0].plot(x, y3)`
	`axes[1, 1].plot(x, y4)`
	`plt.show()`

Sharing axes	`fig, axes = plt.subplots(1, 2, sharey=True) # Share Y axis` `# or sharex=True, shareboth=True`
Adjusting layout to prevent overlap	`plt.tight_layout()` `# or fig.tight_layout()`
Adding space between subplots	`plt.subplots_adjust(wspace=0.4, hspace=0.4)` `# or fig.subplots_adjust(...)`

Twin Axes

Creating a twin Y-axis (`pyplot`)	`fig, ax1 = plt.subplots()`
	`ax1.plot(x, data1, 'g-')`
	`ax1.set_xlabel('Time')`
	`ax1.set_ylabel('Data 1', color='g')`
	`ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis`
	`ax2.plot(x, data2, 'b-')`
	`ax2.set_ylabel('Data 2', color='b')`
	`plt.show()`

Creating a twin X-axis (less common)	`fig, ax1 = plt.subplots()`
	`ax1.plot(x1, y, 'g-')`
	`ax1.set_ylabel('Data')`
	`ax2 = ax1.twiny()`
	`ax2.plot(x2, y, 'b-')`
	`ax2.set_xlabel('Other Time Scale')`
	`plt.show()`

Saving Plots

Saving the current figure (`pyplot`)	`plt.savefig('my_plot.png')`
Saving a specific figure (OO)	`fig.savefig('my_figure.pdf')`
Specifying format (inferred from extension)	Formats: `'png'`, `'pdf'`, `'svg'`, `'jpg'`, `'jpeg'`, `'tif'`, `'tiff'`, `'eps'`, `'ps'`, `'raw'`, `'rgba'`
Controlling DPI	`plt.savefig('high_res.png', dpi=300)`
Transparent background	`plt.savefig('transparent.png', transparent=True)`
Removing whitespace around plot	`plt.savefig('tight.png', bbox_inches='tight')`

Advanced Topics & Best Practices

Recommended Practice: OO Interface

The `pyplot` interface is good for interactive sessions and simple plots. However, for more complex plots, scripting, and embedding plots in GUI applications, the Object-Oriented (OO) interface is recommended.

With OO, you work directly with `Figure` and `Axes` objects, giving you more explicit control over plot elements.

`pyplot` functions (like `plt.plot()`) implicitly create a Figure and Axes if none exist, and plot on the 'current' Axes. OO methods (like `ax.plot()`) are called directly on the `Axes` object.

Example comparison:

pyplot:

```
plt.plot(x, y)
plt.title('Title')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

OO:

```
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title('Title')
ax.set_xlabel('X')
ax.set_ylabel('Y')
plt.show()
```

The OO version is clearer when dealing with multiple subplots or more complex figures.

Using Styles

Listing available styles	`print(plt.style.available)`
Using a specific style	`plt.style.use('seaborn-v0_8-darkgrid') # Use before creating figure/axes`
Using multiple styles (layered)	`plt.style.use(['dark_background', 'presentation'])`

Applying a style temporarily	`with plt.style.context('ggplot'):`
	`  plt.plot(x, y)`
	`  plt.title('Styled Plot')`
	`plt.show()`
	`# Plotting outside the block reverts to default or previously set style`

Resetting style	`plt.style.use('default')`

Handling Dates & Times

Importing modules	`import matplotlib.dates as mdates` `import datetime`

Creating date data	`# Using datetime objects`
	`dates = [datetime.datetime(2023, 1, 1) + datetime.timedelta(days=i) for i in range(10)]`
	`values = np.random.rand(10)`

Basic date plot	`fig, ax = plt.subplots()`
	`ax.plot(dates, values)`
	`# Matplotlib handles conversion automatically`
	`plt.show()`

Formatting date ticks	`fig, ax = plt.subplots()`
	`ax.plot(dates, values)`
	`# Format X-axis to show date`
	`ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))`
	`# Rotate tick labels`
	`plt.xticks(rotation=45, ha='right')`
	`plt.tight_layout()`
	`plt.show()`

Setting date intervals for major/minor ticks	`ax.xaxis.set_major_locator(mdates.AutoDateLocator()) # Auto`
	`ax.xaxis.set_major_locator(mdates.MonthLocator()) # Major ticks every month`
	`ax.xaxis.set_minor_locator(mdates.DayLocator(interval=5)) # Minor ticks every 5 days`

Customizing Matplotlibrc

Matplotlib uses a configuration file (`matplotlibrc`) to customize default plot settings.

Location:
Find the path to your active `matplotlibrc` file:

```
import matplotlib
print(matplotlib.matplotlib_fname())
```

Edit this file (or create a copy in your current directory) to change defaults like figure size, font size, line width, colors, etc.

Example entry in `matplotlibrc`:

```
figure.figsize : 8, 6    # figure size in inches
lines.linewidth : 2      # line width in points
axes.titlesize : 16      # fontsize of the axes title
```

You can also change settings programmatically for the current session using `plt.rcParams`:

```
plt.rcParams['figure.figsize'] = [10, 8]
plt.rcParams['lines.linewidth'] = 3
plt.rcParams['font.size'] = 14
```

Tips for Large Data

Downsampling: Reduce the number of data points plotted, especially for line plots (e.g., only plot every Nth point or average points in bins).

Using `plt.scatter` with `s=1` and `alpha`: For many overlapping points, a small size and transparency can reveal density better than `plt.plot` or default `plt.scatter` settings.

Hexbin Plots: Good for showing density of points in 2D. `plt.hexbin(x, y, gridsize=50, cmap='viridis')`

2D Histograms: Similar to hexbin, shows density in rectangular bins. `plt.hist2d(x, y, bins=50, cmap='viridis')`

Aggregating Data: Group data into bins and plot aggregated statistics (mean, count, etc.) as a bar chart or line plot.

Using Libraries like Datashader: For extremely large datasets, libraries like Datashader can perform rasterization off-GPU before passing aggregated data to Matplotlib for rendering.

Common Issues & Troubleshooting

Plots not showing: Ensure you call `plt.show()` or `fig.show()`. In interactive environments like Jupyter, `%matplotlib inline` or `%matplotlib notebook` is often needed.

Plots overlapping: Use `plt.tight_layout()` or `fig.tight_layout()` to automatically adjust subplot parameters for a tight layout. Manually adjust using `plt.subplots_adjust()` or `fig.subplots_adjust()`.

Labels/Titles too small/large: Adjust `fontsize` parameter in the relevant function (`set_title`, `set_xlabel`, `legend`, `text`, etc.) or set defaults using `plt.rcParams` or `matplotlibrc`.

Incorrect date format: Use `matplotlib.dates` locators and formatters (e.g., `mdates.DateFormatter('%Y-%m-%d')`) on the axis.

Legends not appearing: Ensure you add the `label` argument to your plotting calls (`ax.plot(..., label='series')`) and call `ax.legend()` or `plt.legend()`.

Saving blank plots: Call `plt.savefig()` BEFORE `plt.show()` when using the `pyplot` interface, as `plt.show()` might clear the figure depending on the backend.