

Lesson 17: Matplotlib Basics

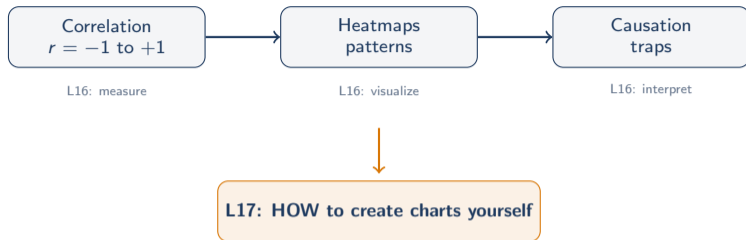
Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

Previously on Data Science...



L16 showed you: correlation heatmaps, scatter plots, rolling charts. **L17 asks:** “How do I *build* those myself?”

You have been reading charts for 4 lessons – now you learn to write them

Learning Objectives

After this lesson, you will be able to:

1. Explain the figure/axes architecture in matplotlib
2. Create line, bar, scatter, and histogram charts from scratch
3. Customize styles, colors, labels, and annotations
4. Apply matplotlib to financial data (price, volume, returns)
5. Build multi-panel figures with subplots

Matplotlib is the foundation – every Python visualization library builds on it

From Chart Reader to Chart Writer

You have been **READING** charts. Now you learn to **WRITE** them.

Think of matplotlib as learning to paint:

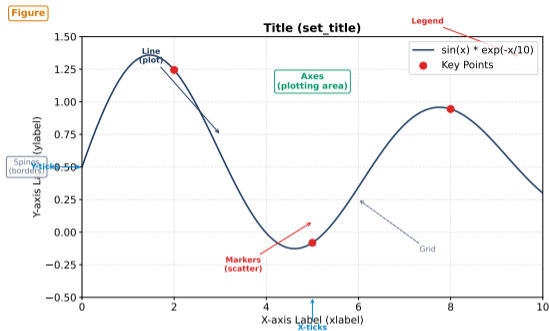
- The **figure** is your canvas
- The **axes** are the frames you paint inside
- **Colors, labels, annotations** are your brushes

Today's mission:

Master the 4 core chart types that cover 90% of data visualization needs.

Line, bar, histogram, scatter – four tools, infinite stories

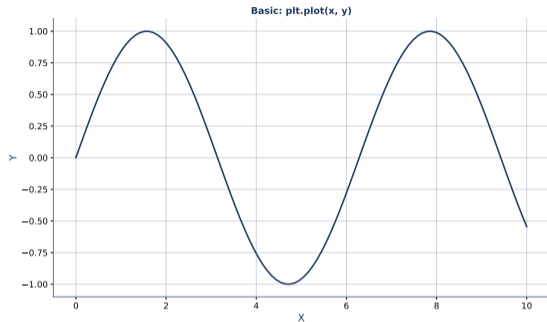
The Canvas: Figure and Axes



- **Figure** = entire window (canvas)
- **Axes** = one plot area (where data lives)
- `fig, ax = plt.subplots()` creates both at once

Always think: figure contains axes, axes contain your data

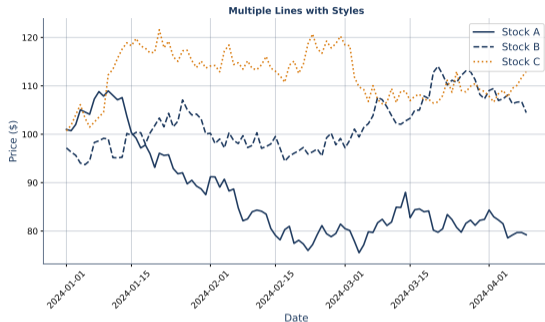
Line Plot: The Simplest Chart



`ax.plot(x, y)` – one line of code, one line on screen.

Line plots show how a single variable changes over a continuous axis

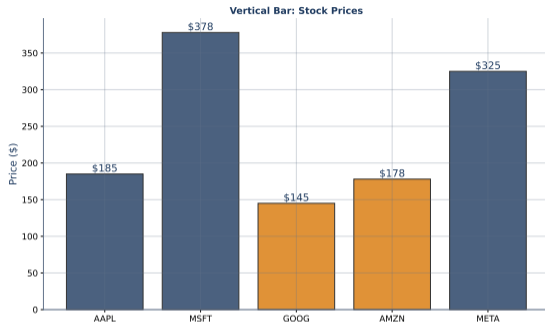
Overlaying Multiple Series



Call `ax.plot()` multiple times on the same axes to overlay data.

Add a legend so viewers know which line is which

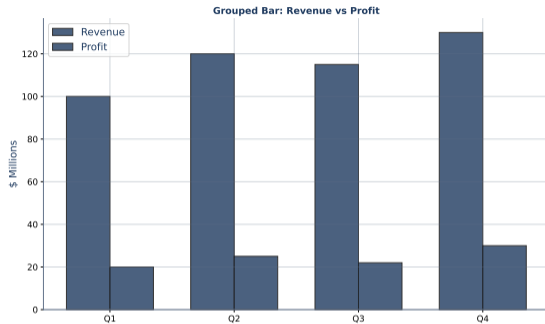
Bar Chart: Comparing Categories



`ax.bar(categories, values)` – one bar per category.

Bar charts answer: which category is biggest, smallest, or different?

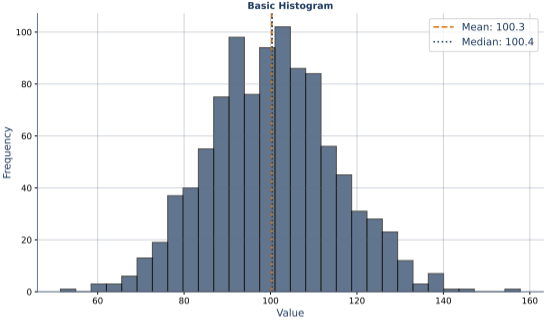
Grouped Bar Chart: Side-by-Side Comparison



Offset bar positions with `np.arange()` and `width` to group them.

Grouped bars compare multiple series across the same categories

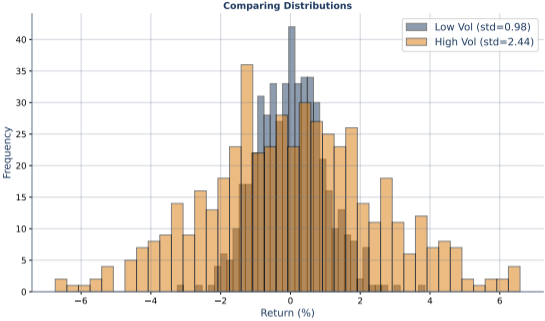
Histogram: Seeing Distributions



`ax.hist(data, bins=30)` – bins determine resolution.

Histograms answer: what is the shape of my data?

Overlaying Distributions



Use `alpha=0.5` for transparency when overlaying histograms.

Compare distributions by overlaying with different colors and transparency

Checkpoint: Which Chart?

Quick Check

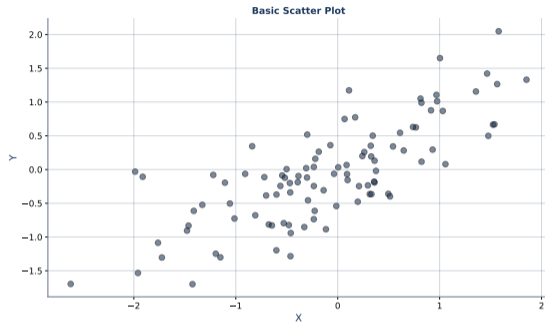
Match the goal to the chart type:

1. Show a trend over time → ?
2. Compare category sizes → ?
3. Show the shape of a distribution → ?

Answers: 1. Line 2. Bar 3. Histogram

Choosing the right chart type is half the battle

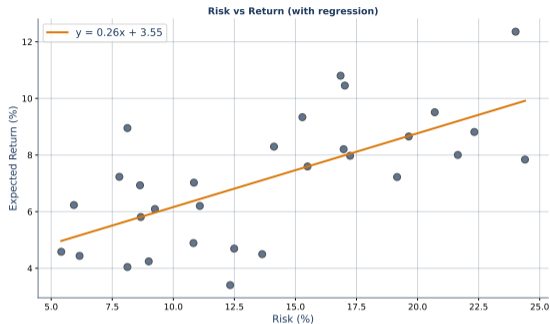
Scatter Plot: Relationships



`ax.scatter(x, y)` – each dot is one observation.

Scatter plots reveal relationships, clusters, and outliers

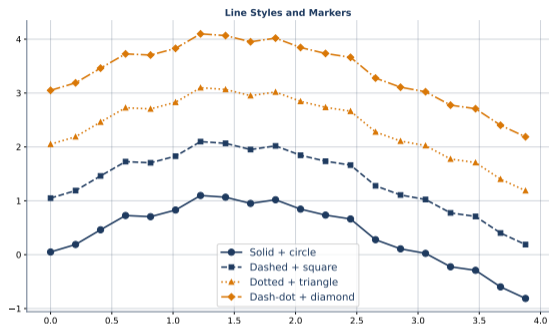
Scatter + Trend Line



Add `np.polyfit()` trend line to quantify the relationship.

A regression line turns a visual pattern into a number

Customization: Styles, Colors, Markers



- `linestyle`: solid, dashed, dotted, dash-dot
- `color`: named, hex, or RGB
- `marker`: o, s, ^, D, and dozens more

Good defaults first, customize only when it adds clarity

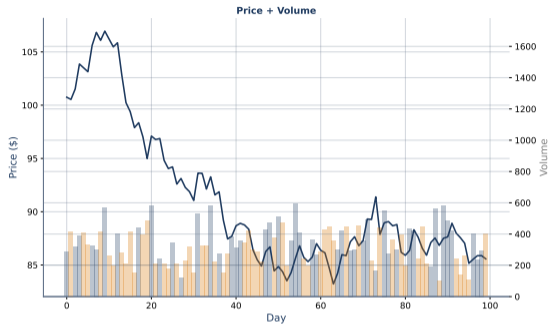
Annotations: Highlight What Matters



```
ax.annotate('text', xy=(...), arrowprops={...})
```

Annotations guide the viewer's eye to the insight

Finance: Price and Volume



Dual-axis chart: price on left axis, volume bars on right axis.

`twinx()` creates a second y-axis sharing the same x-axis

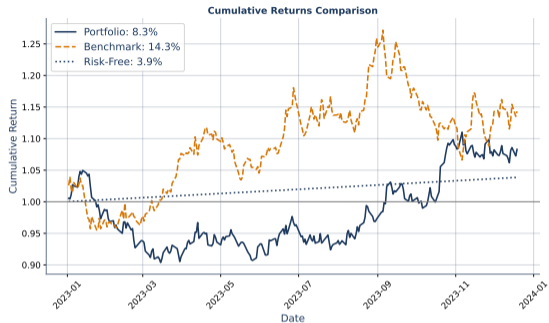
Finance: Rolling Sharpe Ratio



Rolling window calculations track performance stability over time.

A steady Sharpe above 1.0 is better than a volatile Sharpe averaging 2.0

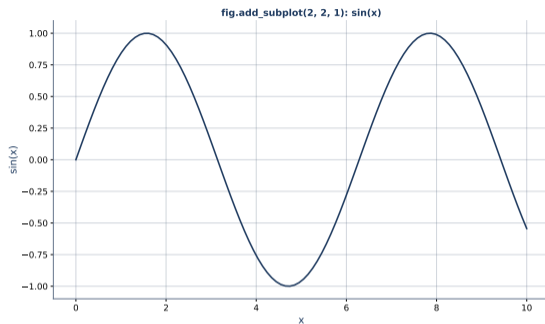
Finance: Cumulative Returns



`(1 + returns).cumprod()` turns daily returns into a growth curve.

Cumulative return charts are how investors compare strategies

Subplots: Multiple Charts in One Figure



`fig.add_subplot(rows, cols, index)` or
`fig, axes = plt.subplots(1, 2)` for a grid.

Subplots let you tell a multi-part story in one figure

Hands-On: Build Your First Dashboard

Task: Create a 4-panel financial dashboard.

1. Generate 252 days of random stock returns with `np.random.normal()`
2. **Panel 1:** Line plot of cumulative returns
3. **Panel 2:** Histogram of daily returns with mean line
4. **Panel 3:** Bar chart of monthly returns
5. **Panel 4:** Scatter plot of return vs volume (simulated)

Stretch goal: Add annotations marking the best and worst days.

Hands-on: 10 minutes – use `plt.subplots(2, 2)` for the grid

The Art of Showing Data



You



"Every chart tells a story."

Make it a good one.

"A chart is not decoration. It is an argument made visible."

The best chart is the one your audience understands in 5 seconds

Key Takeaways

What you now know:

1. **Figure + Axes** = matplotlib's two-layer architecture
2. **Line plots** for trends, **bar charts** for categories, **histograms** for distributions, **scatter plots** for relationships
3. **Customization** (colors, styles, annotations) adds clarity, not decoration
4. **Finance charts** (price/volume, Sharpe, cumulative returns) use the same 4 tools
5. **Subplots** combine multiple views into one coherent story

You now own the core toolkit – L18 adds statistical intelligence with seaborn

Coming Up: L18 – Seaborn Statistical Plots

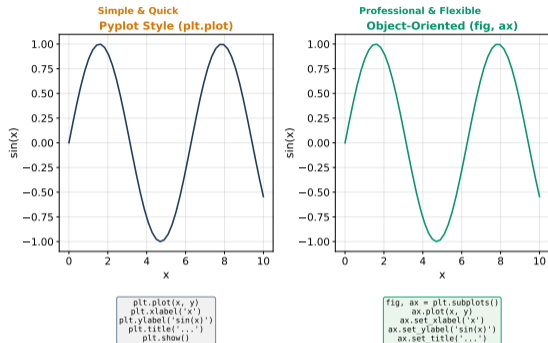
Matplotlib is powerful but verbose. Seaborn adds elegance.

- Distribution plots (KDE, violin, box) in one line
- Statistical regression with confidence bands
- Correlation heatmaps with automatic formatting



Seaborn is matplotlib with a PhD in statistics and a sense of style

Self-Study: Pyplot vs Object-Oriented API



- `plt.plot()` = quick and simple (pyplot interface)
- `fig, ax = plt.subplots()` = full control (OO interface)
- Use OO for anything beyond a quick sketch

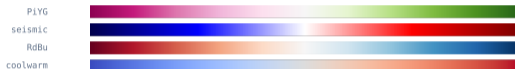
The OO API gives you explicit control over every element

Self-Study: Colormap Gallery

Sequential (magnitude)



Diverging (correlation)



Qualitative (categories)

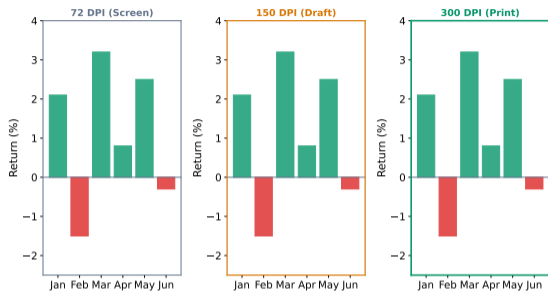


```
Usage: plt.imshow(data, cmap="viridis") | sns.heatmap(data, cmap="coolwarm")
```

- Sequential: single-hue gradients (e.g., Blues, Reds)
- Diverging: two-hue with neutral center (e.g., coolwarm, RdBu)
- Qualitative: distinct colors for categories (e.g., Set2, tab10)

Choose colormaps based on data type – sequential for magnitude, diverging for deviation

Self-Study: DPI and Resolution



Low quality
Pixelated edges

Medium quality
Acceptable clarity

High quality
Crisp & sharp

Recommendation: Use dpi=300 for publications, dpi=150 for presentations

- 72 DPI: screen only, blurry when printed
- 150 DPI: good for presentations
- 300 DPI: publication quality

Use dpi=300 and bbox_inches='tight' for professional output

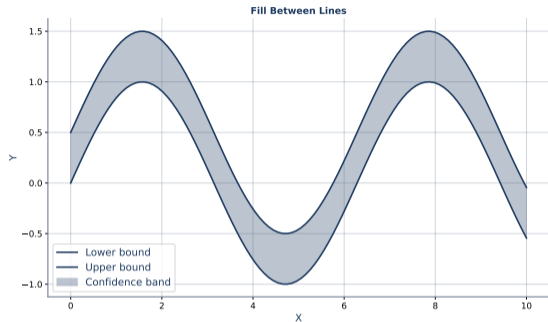
Self-Study: Line with Markers



- Add markers to highlight individual data points
- Useful for small datasets where each point matters

Markers make data points visible – use sparingly with large datasets

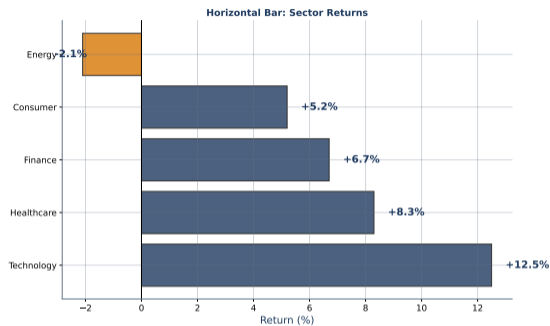
Self-Study: Fill Between



- `ax.fill_between(x, y1, y2, alpha=0.3)` shades between curves
- Great for confidence bands, ranges, and area comparisons

Fill between creates visual weight – use it for emphasis

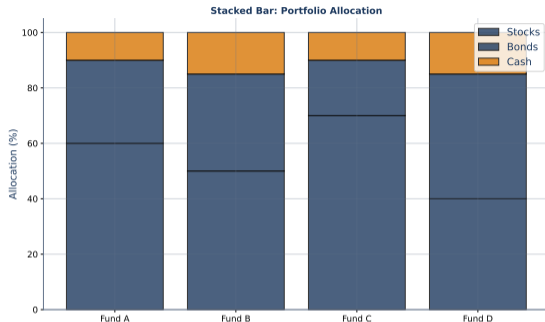
Self-Study: Horizontal Bar Chart



- `ax.barh()` for horizontal bars
- Better when category names are long

Horizontal bars are easier to read when you have many categories

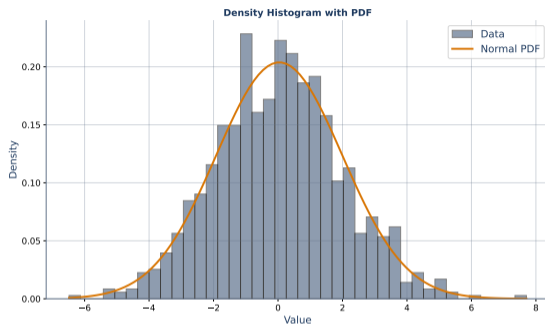
Self-Study: Stacked Bar Chart



- Shows composition within each category
- Use `bottom=` parameter to stack bars

Stacked bars show part-to-whole relationships

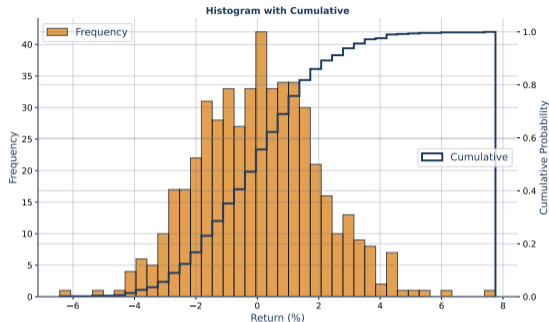
Self-Study: Density Histogram



- `density=True` normalizes to probability density
- Area under curve sums to 1.0

Density normalization lets you overlay a PDF curve for comparison

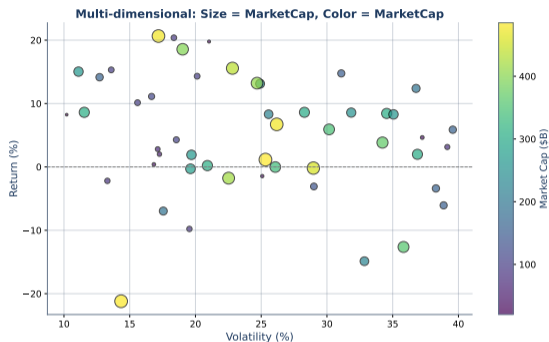
Self-Study: Cumulative Histogram



- `cumulative=True` shows running total
- Useful for finding percentiles and VaR thresholds

The cumulative histogram is the empirical CDF

Self-Study: Multidimensional Scatter



- Encode extra dimensions with size (`s=`) and color (`c=`)
- Up to 4 variables in a single 2D plot

Size and color add information without adding axes

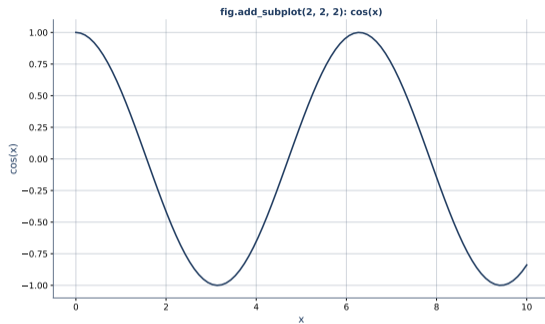
Self-Study: Grouped Scatter



- Loop over groups, plot each with different color/marker
- Essential for comparing clusters or categories

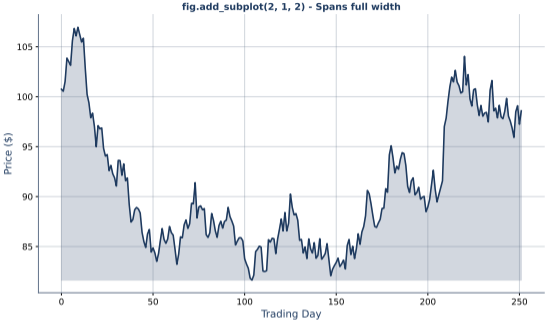
Color-coded groups reveal whether the relationship differs by category

Self-Study: add_subplot (cos)



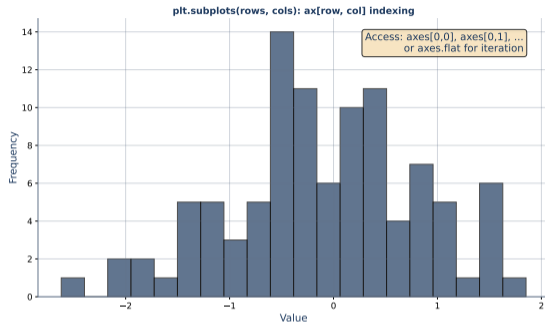
Multiple axes in one figure – each subplot is independent

Self-Study: Spanning Subplot



Subplots can span multiple grid positions for emphasis

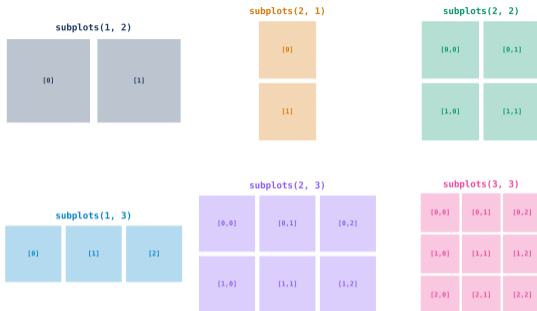
Self-Study: Subplots Grid Iteration



- `fig, axes = plt.subplots(2, 2)` creates a 2x2 grid
- Iterate with `axes.flat` or index with `axes[row, col]`

`plt.subplots()` is the modern, preferred way to create grids

Self-Study: Subplot Layout Patterns

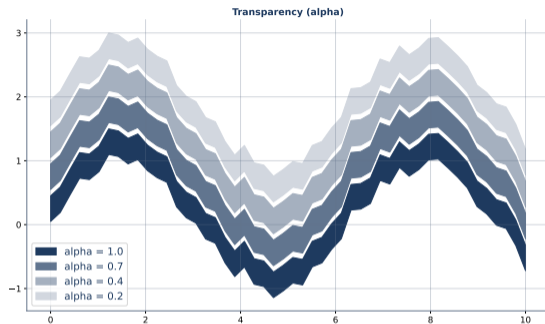


Usage: `fig, axes = plt.subplots(rows, cols)` | Access: `axes[row, col]` or `axes[idx]`

- GridSpec for unequal subplot sizes
- `plt.tight_layout()` prevents overlap

GridSpec gives you pixel-level control over subplot placement

Self-Study: Transparency (Alpha)



- $\alpha=0.5$ makes overlapping data visible
- Essential for scatter plots with many points

Transparency reveals density in overlapping data

Self-Study: Axis Customization
























- `set_xlim()`, `set_ylim()` control visible range
- `set_xticks()`, `set_xticklabels()` control tick marks

Custom axes focus the viewer on the data range that matters

Self-Study: Marker Types Reference

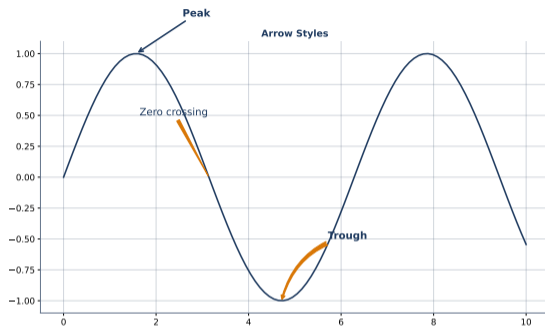
Matplotlib Marker Reference

Common:						
	'o'	's'	'^'	'v'	'd'	'*''
Lines:						
	'+'	'x'	' '	'_'		
Filled:						
	'p'	'h'	'B'	'<'	'>'	
Special:						
	'.''	'*''	'Y'	'Y'	'Y'	'Y'

Usage: plt.plot(x, y, marker='o', markersize=10)

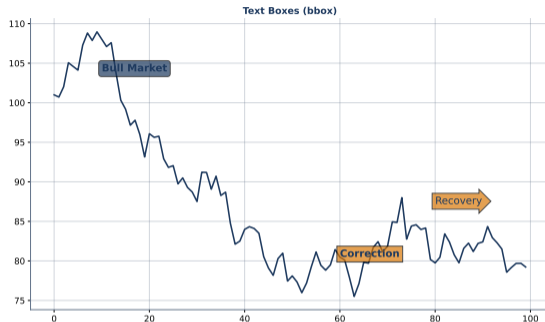
Reference card: matplotlib supports 30+ marker styles

Self-Study: Arrow Styles for Annotations



Different arrowprops styles for different annotation needs

Self-Study: Text Boxes



- `bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5)`
- Use text boxes for key statistics or callouts

Boxed text stands out from the chart – use for key metrics

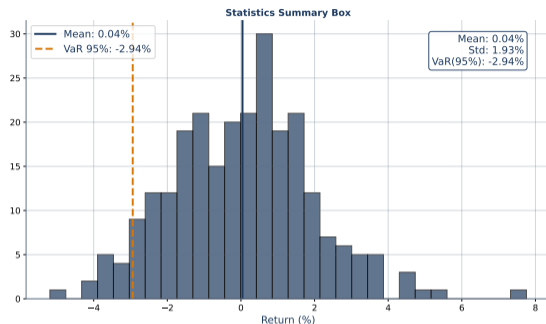
Self-Study: Event Markers



- `ax.axvline()` for vertical event lines
- Label events with `ax.text()` or annotations

Event markers connect chart patterns to real-world causes

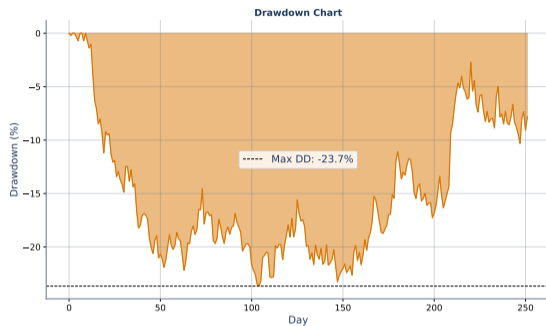
Self-Study: Statistics Box



- Embed key metrics directly in the chart
- `ax.text()` with `transform=ax.transAxes` for relative positioning

A statistics box turns a chart into a self-contained report

Self-Study: Drawdown Chart



- Drawdown = current value / peak value - 1
- Always negative or zero – shows losses from peak

Drawdown charts reveal risk that cumulative return charts hide