

***Rotman***

# INTRO TO DATA VISUALIZATION

Part II Intro to Matplotlib – Concepts and Basic Plots

September 3, 2024 Prepared by Jay Cao / [TDMDAL](https://tdmdal.github.io)

Website: <https://tdmdal.github.io/mma-dv-2024/>



Rotman School of Management  
UNIVERSITY OF TORONTO

# Python Visualization Package Landscape (1)

- [Matplotlib](#) & derivatives
  - [Pandas.DataFrame.plot](#) (simple plots directly from pandas dataframes)
  - [Seaborn](#) (high-level interface; good-looking and modern default graphics)
  - [Plotnine](#) (an implementation of *the grammar of graphics*; R ggplot2-like syntax)
  - [Cartopy](#), [geoplot](#), [geopandas](#)'s [plot\(\)](#) (geospatial visualization)
  - Many more, [a full list](#)
- [plotly](#), [plotly express](#), & [plotly dash](#)
  - Modern-looking, interactive web-based charts; support dashboard;
  - Built on [plotly.js](#), a javascript plotting library.

# Python Visualization Package Landscape (2)

- [HoloViz](#)
  - A set of high-level tools
    - e.g., hvPlot (interactive plot), Panel (dashboard), GeoViews (geospatial plot), etc.
  - Built on [Bokeh](#) (yet another Python plotting tool), Matplotlib, Plotly
- [Lets-plot](#)
  - yet another implementation of the grammar of graphics
  - by JetBrains, the [PyCharm](#) IDE developer
- [Vega-Altair](#)
  - a declarative visualization library based on the [Vega-Lite](#) grammar
  - Declarative (say what you want) vs imperative (say how to get what you want by step-by-step instructions)
- Many more...

# Why Matplotlib

- General purpose
  - Flexible/low-level enough to plot almost anything you want
- Highly customizable plots
  - You could consider using [seaborn](#) for quick and good-looking default plots if not much customization is needed
- Integrate well with other packages (because its many derivatives)
  - E.g., pandas, [geopandas](#), etc.
- Good documentation and community support

# Our Plan to Learn Matplotlib

- Understand the principles/fundamentals
  - Matplotlib library basic architecture
  - Anatomy of a figure & object hierarchy
  - Two coding styles/approaches
- Work through two main notebooks
  - Basic plots
  - From default to publication-ready – how to refine/customize a plot
- Won't take you through syntax for each type of plots

# Matplotlib Software Architecture



## Script (`matplotlib.pyplot`)

- A light wrapper/interface to artist layer
- Beginner friendly function calls for simple plots



## Artist (`matplotlib.artist.Artist`)

- All visual elements in a figure
- Primitive artists: Line2D, Rectangle, Text, etc.
- Container artists: Figure, Axes, Axis, etc.



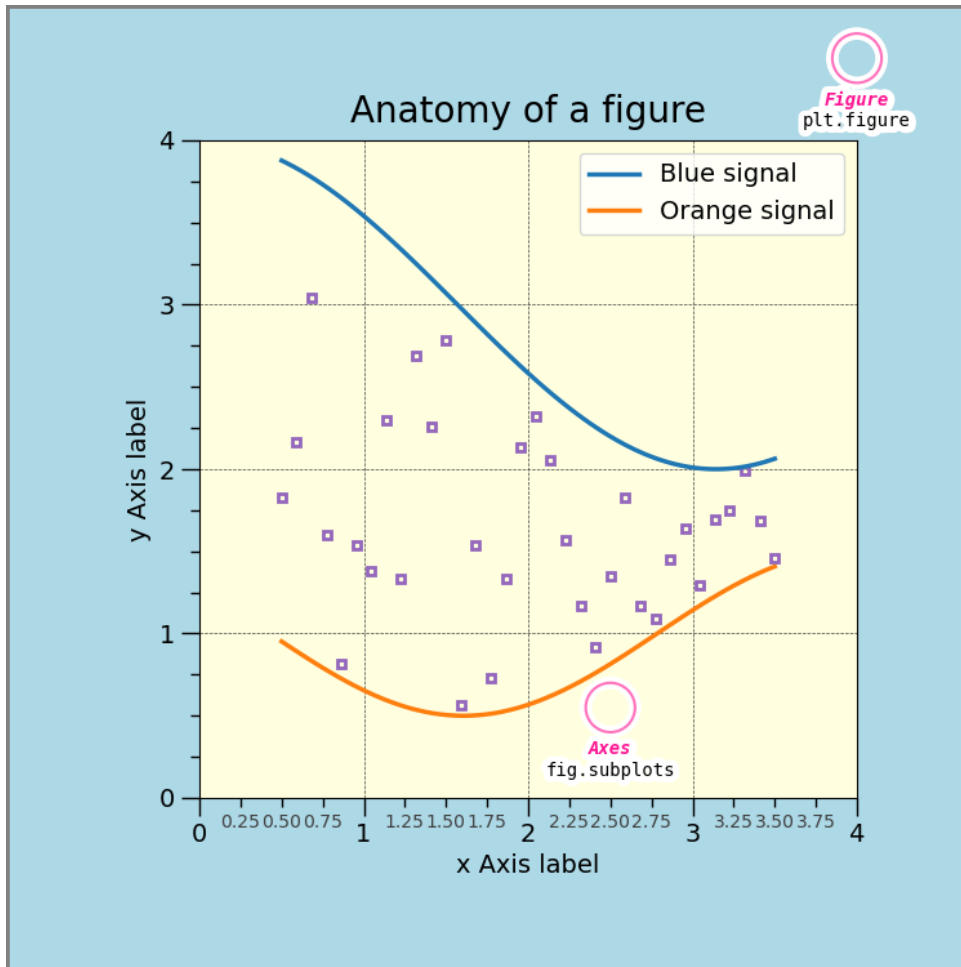
## Backend (`matplotlib.backend_bases`)

- FigureCanvasBase (“canvas”)
- RenderBase (“paintbrush”)
- ...

Represent and  
manipulate a  
Figure.

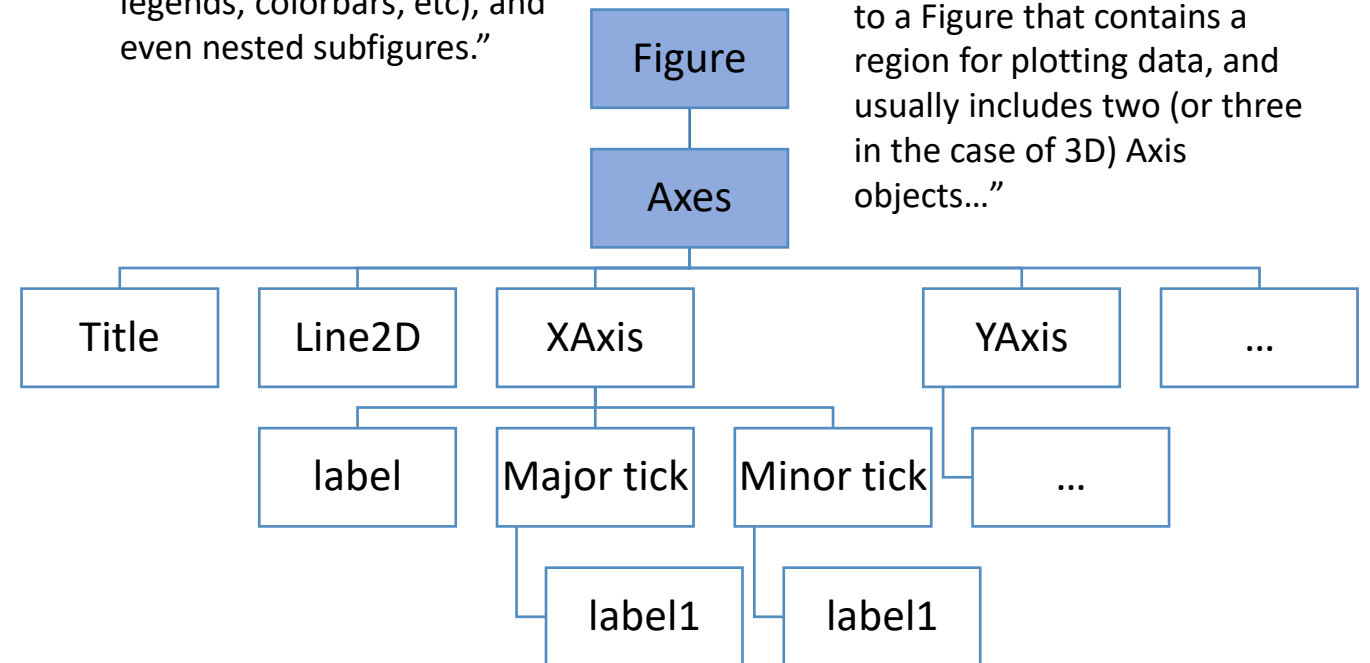
Rendering a Figure  
on different  
hardware & software  
settings.

# Anatomy of a Figure & Object Hierarchy (1)



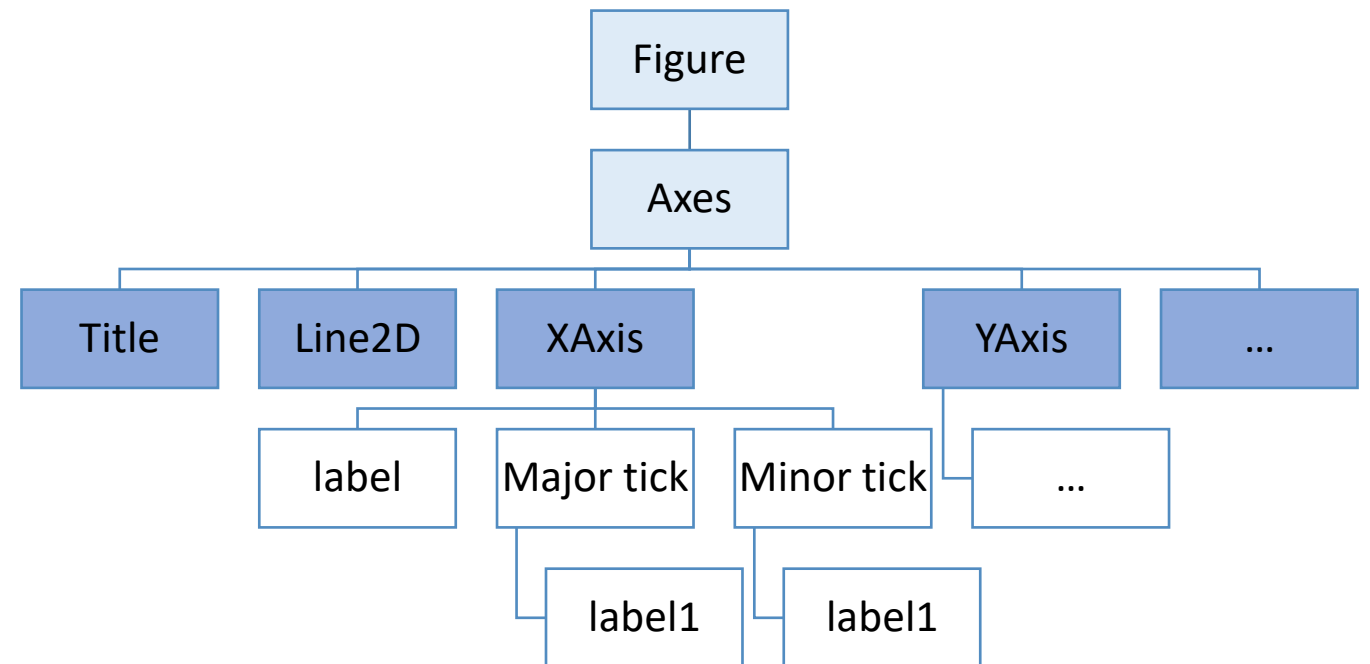
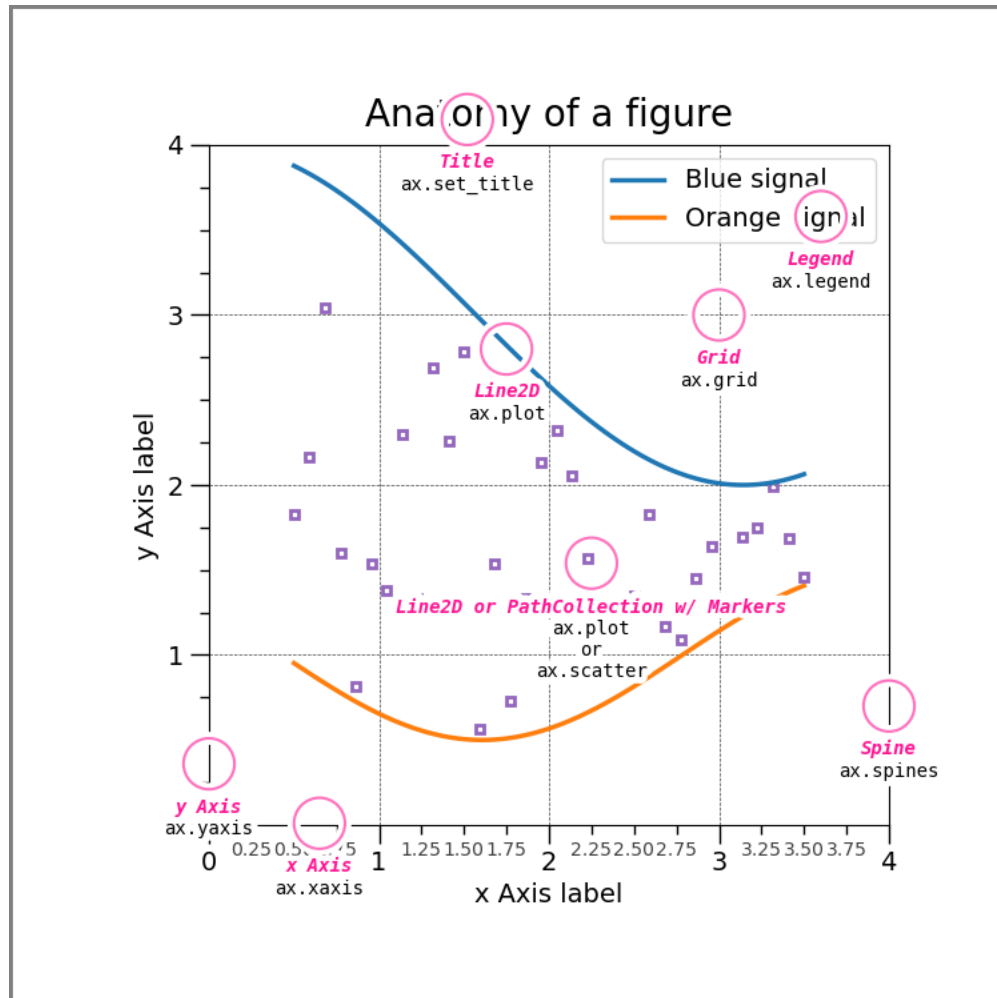
“The Figure keeps track of all the child Axes, a group of 'special' Artists (titles, figure legends, colorbars, etc), and even nested subfigures.”

“An Axes is an Artist attached to a Figure that contains a region for plotting data, and usually includes two (or three in the case of 3D) Axis objects...”



Note: Only a subset of the objects are shown in this diagram.

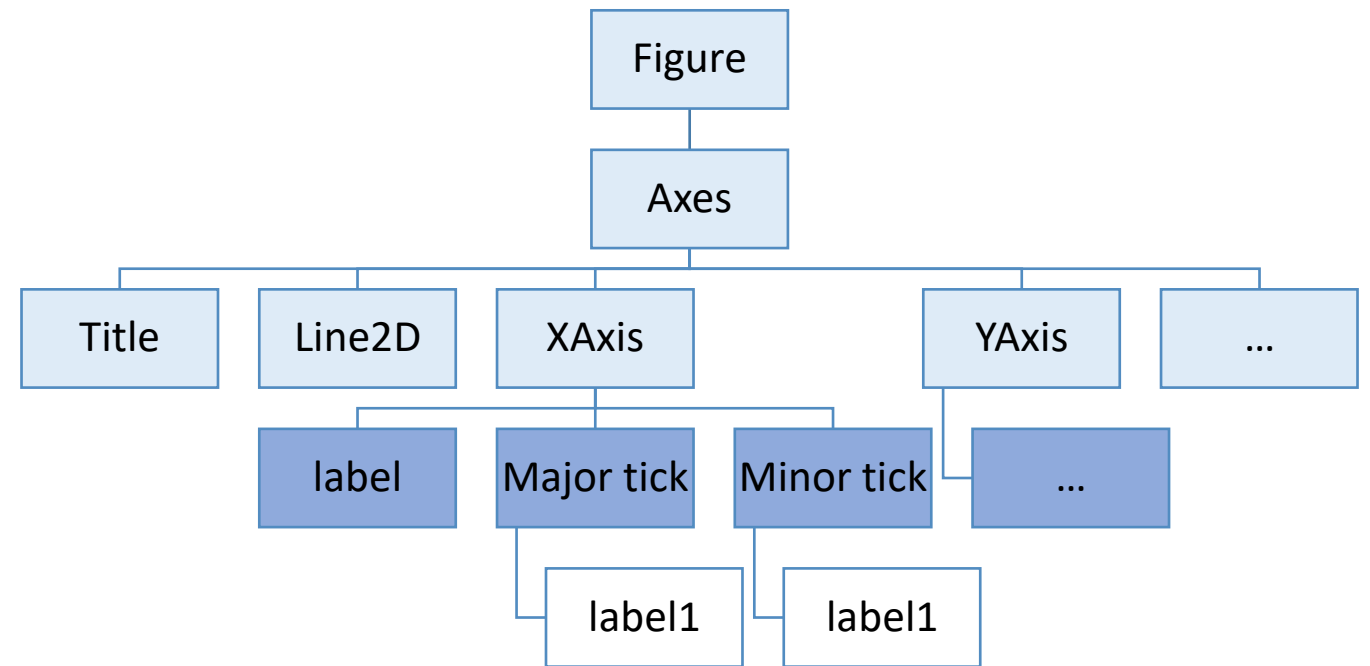
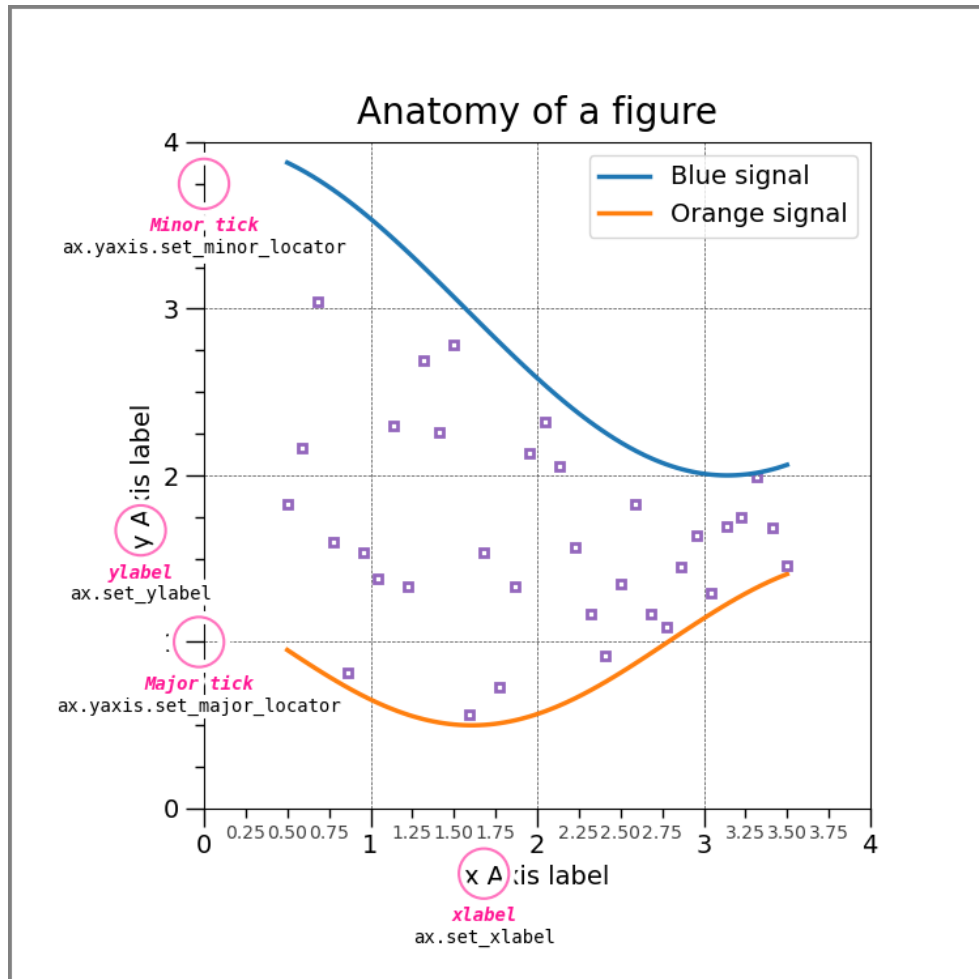
# Anatomy of a Figure & Object Hierarchy (2)



Note: Only a subset of the objects are shown in this diagram.

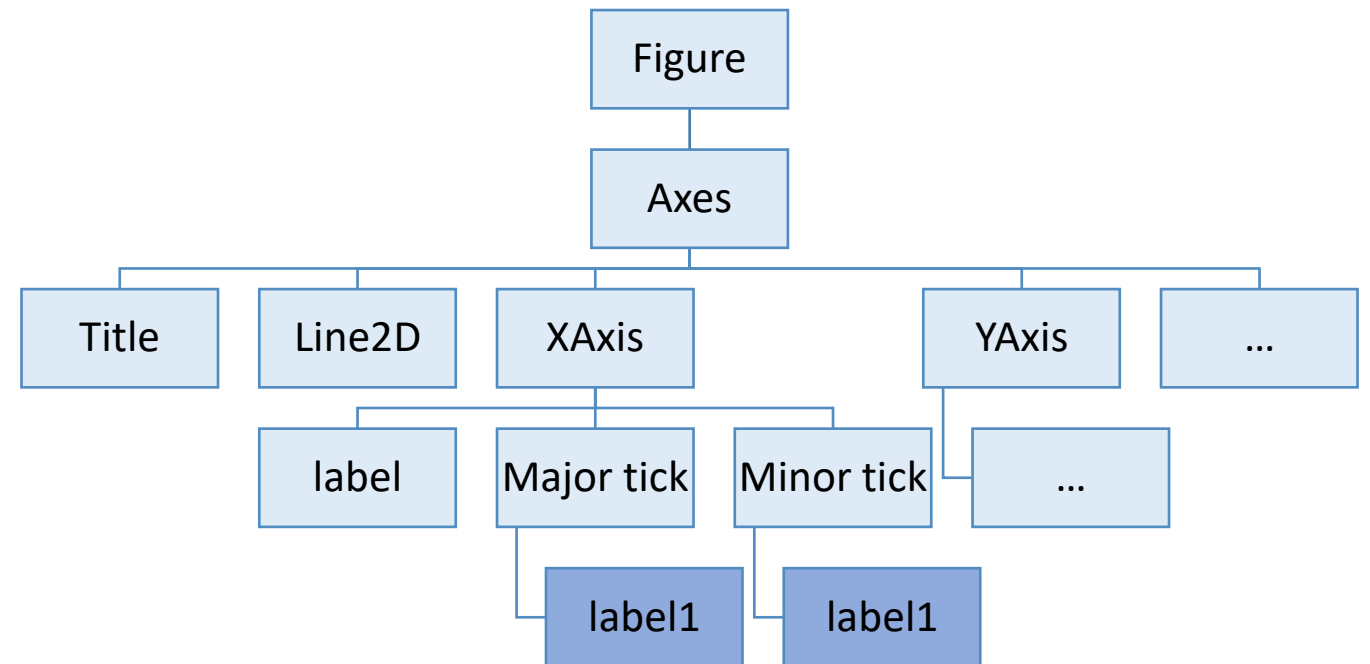
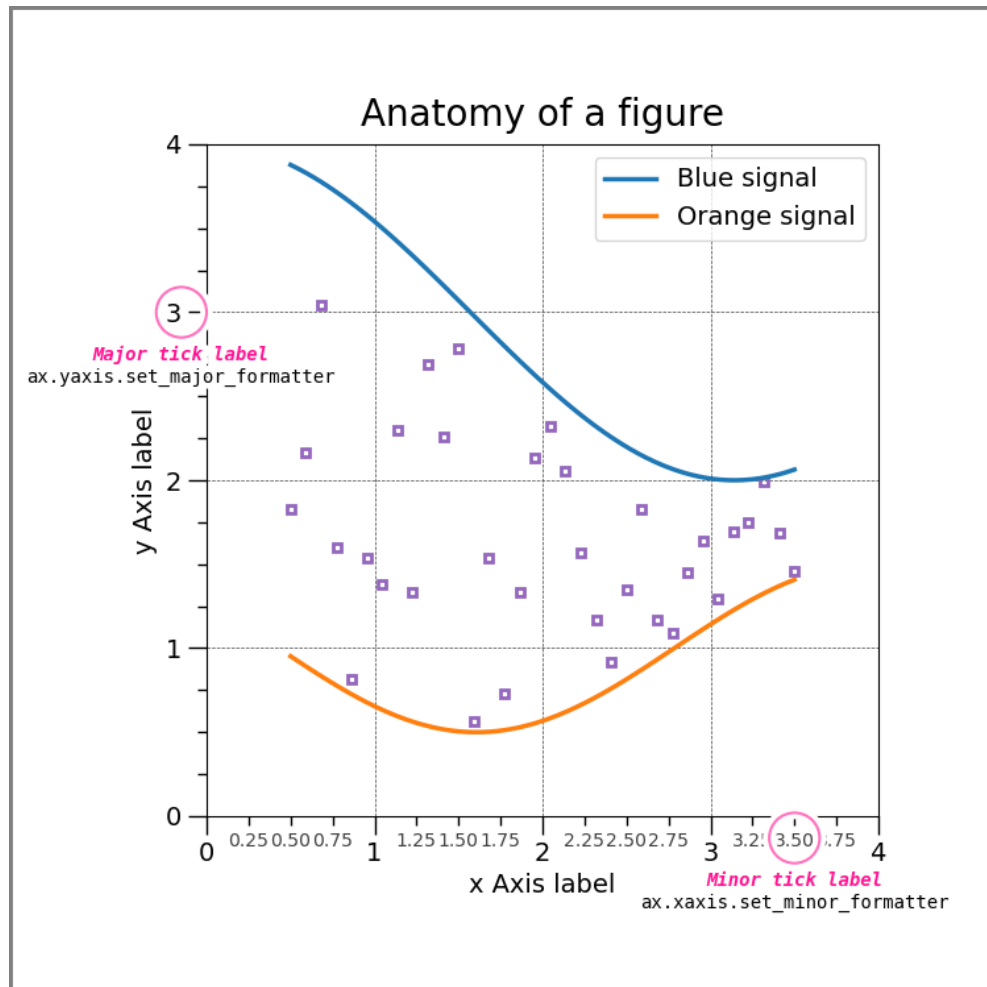


# Anatomy of a Figure & Object Hierarchy (3)



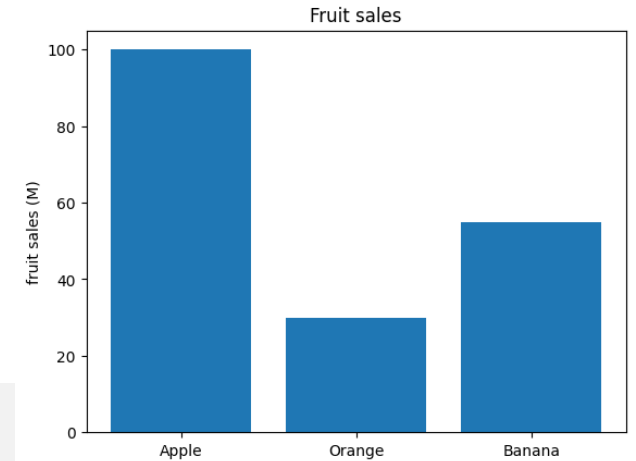
Note: Only a subset of the objects are shown in this diagram.

# Anatomy of a Figure & Object Hierarchy (4)



Note: Only a subset of the objects are shown in this diagram.

# Two Coding Styles/Approaches



```
import pandas as pd
import matplotlib.pyplot as plt
```

```
# create a simple dataset for plotting
df = pd.DataFrame(data = {
    'fruits': ['Apple', 'Orange', 'Banana'],
    'sales': [100, 30, 55],
})
```

```
# Approach 1 (script layer; implicit; stateful;)
```

```
# call plt.bar() function from matplotlib.pyplot
# implicitly create figure and axes
```

```
→ plt.bar(x=df["fruits"], height=df["sales"])
```

```
# call pyplot level functions to set title and ylabel
# implicitly refer to the current axes' title and ylabel
```

```
→ plt.title("Fruit sales")
→ plt.ylabel("fruit sales (M)")
```

```
# show the plot
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
# create a simple dataset for plotting
df = pd.DataFrame(data = {
    'fruits': ['Apple', 'Orange', 'Banana'],
    'sales': [100, 30, 55],
})
```

```
# Approach 2 (artist layer; explicit; stateless; OOP;)
```

```
# create a figure and an axes (subplot)
```

```
→ fig, ax = plt.subplots()
```

```
# explicitly call the bar method of the axes just created
```

```
→ ax.bar(x=df["fruits"], height=df["sales"])
```

```
# explicitly call the set_xyz() methods of the axes instance
```

```
→ ax.set_title("Fruit sales")
→ ax.set_ylabel("fruit sales (M)")
```

```
# show the plot
plt.show()
```

# The Artist Layer Approach - Preferred

- Figure and Subplots/Axes
  - Create a figure with 2x2 grid of Axes: `fig, ax = plt.subplots(2, 2)`
  - Set figure-level properties/objects: `fig.set_facecolor()`, `fig.suptitle()`
- Manipulate objects below an Axes (using method calls)
  - 1-layer below, e.g., `ax.plot()`, `ax.set_title()`, `ax.set_facecolor()`
  - 2-layer below, e.g., `ax.set_ylabel()`, `ax.yaxis.set_major_locator()`
  - 3-layer below, e.g., `ax.yaxis.set_major_formatter()`

# The Artist Layer Approach – Subplots

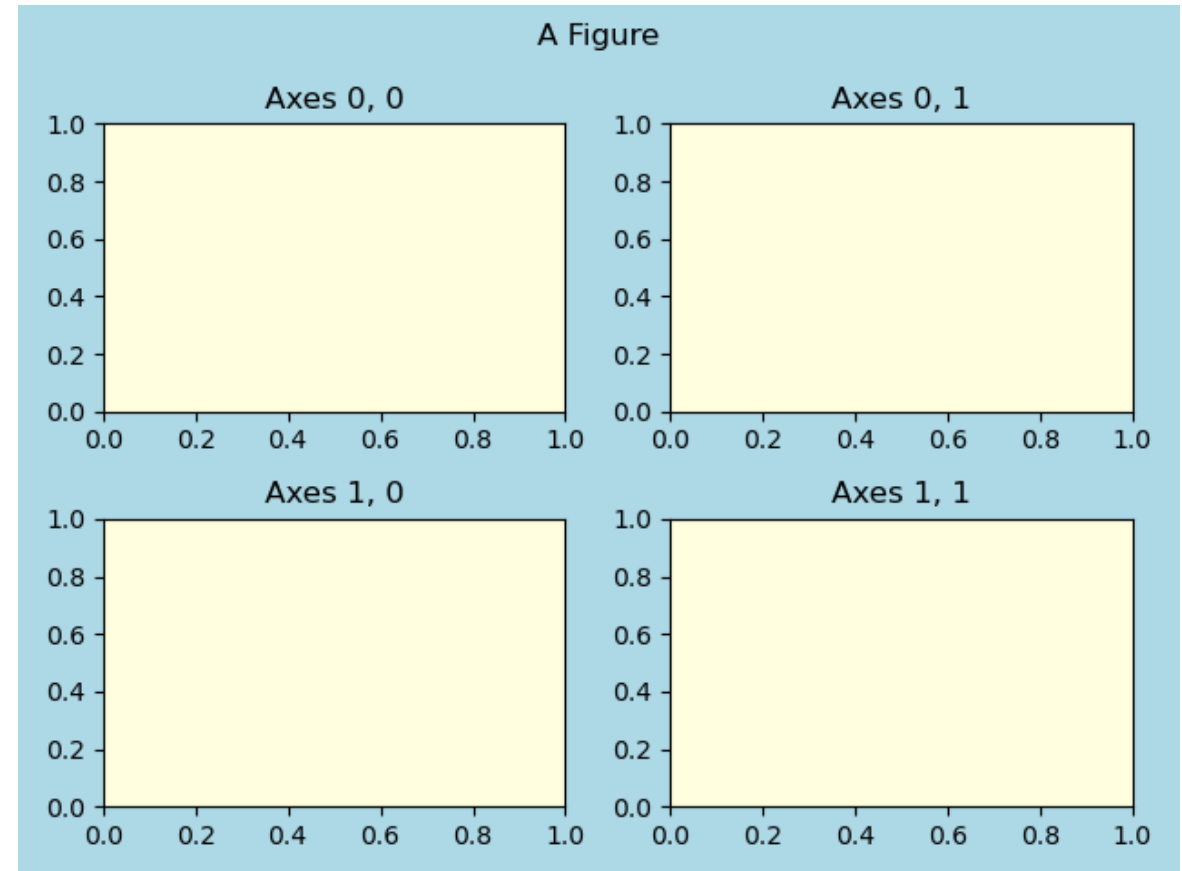
```
import matplotlib.pyplot as plt

# create a figure with 2x2 grid of subplots
→ fig, axes = plt.subplots(nrows=2, ncols=2)

# set the title of the figure
fig.set_facecolor('lightblue')
fig.suptitle('A Figure')

# set the title of each subplot
for row in range(2):
    for col in range(2):
        → axes[row, col].set_title(f'Axes {row}, {col}')
        axes[row, col].set_facecolor('lightyellow')

# set layout and display the plot
plt.tight_layout()
plt.show()
```



# The Artist Layer Approach – An Example

```
import matplotlib.pyplot as plt

# create a simple dataset for plotting
df = pd.DataFrame(data = {
    'fruits': ['Apple', 'Orange', 'Banana'],
    'sales': [80, 30, 55],
})

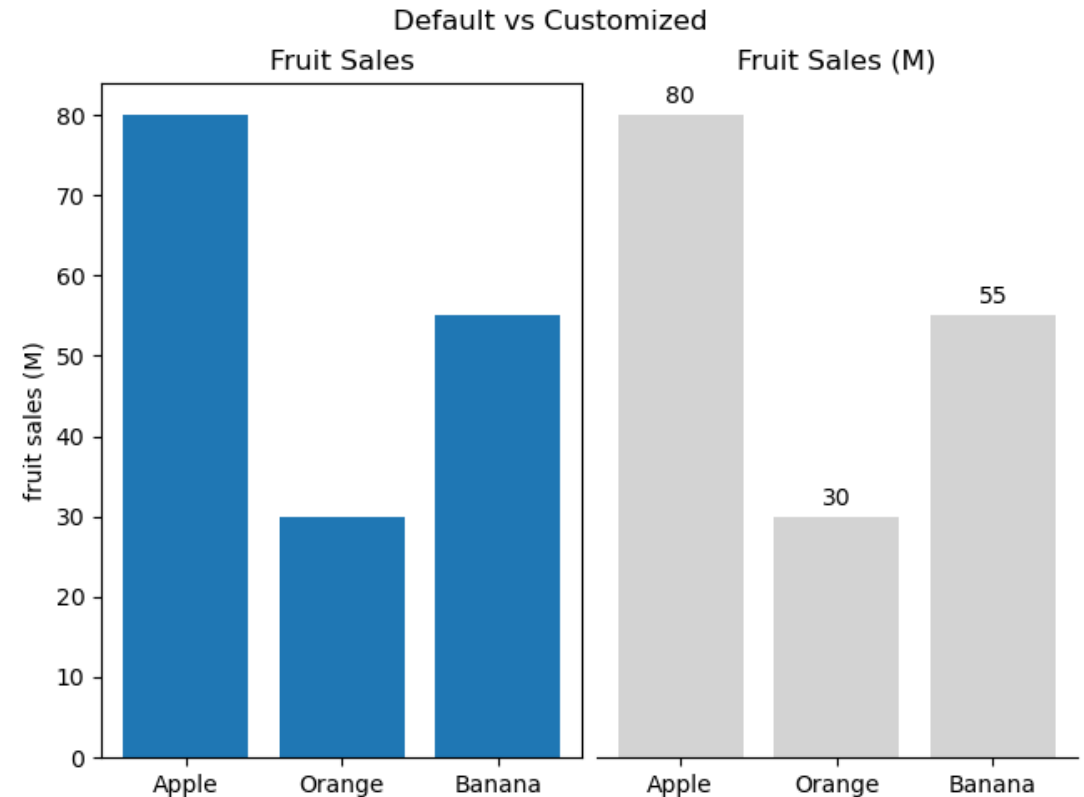
# Approach 2 gives cleaner code when having more
# than one subplot
→ fig, axes = plt.subplots(nrows=1, ncols=2,
                           layout="constrained")

# Left subplot (Fruit Sales)
axes[0].bar(x=df["fruits"], height=df["sales"])
axes[0].set_ylabel("fruit sales (M)")
axes[0].set_title("Fruit Sales")

# Right subplot (Fruit Sales (M))
rects = axes[1].bar(x=df["fruits"], height=df["sales"],
                    color="lightgray")
axes[1].bar_label(rects, padding=3)
axes[1].spines[["top", "right", "left"]].set_visible(False)
axes[1].yaxis.set_visible(False)
axes[1].set_title("Fruit Sales (M)")

fig.suptitle("Default vs Customized")

plt.show()
```



# Hands-on: Basic Plots

- The gapminder dataset
- Matplotlib or Seaborn
- A few basic plots
  - Bar plot and grouped bar plot
  - Line plot
  - Histogram
  - Scatter plot
- Bubble plot (a taste of customizing a default plot)