Matplotlib/Seaborn

Prof.Dr. Bahadır AKTUĞ
803400815021 Machine Learning with Python

**Compiled from sources given in the references.*

Matplotlib

- Matplotlib is the most popular graphics module used in Python.
- Matplotlib, line many other libraries, relies on Numpy module by design for input and output.
- Matplotlib module is also based on Pylab graphics which resembles Matlab graphics routines for the most part.
- Matplotlib offers two interfaces:
  - Pyplot interface with basic control for quick and easy plotting,
  - Object-oriented interface for a more comprehensive plotting with full-control
- Both interfaces have their own ups and downs.

Pyplot Interface & Advantages

- This interface is a simple and quick way to create graphs with Matplotlib, which is based on a series of functions that you can call to create and modify graphs.
- Some of its advantages are:
  - **Greater ease of use**: The pyplot interface is very intuitive and easy to use. You just have to call a series of functions to create and modify a graph, which makes it very easy to create basic graphs.
  - **Less code**: Due to its simplicity, the pyplot interface requires less code than the object interface to creating basic graphs.
  - **More excellent compatibility**: The pyplot interface is compatible with more versions of Matplotlib than the object interface, making it more reliable.

Pyplot Interface & Disadvantages

- However, the ***pyplot interface*** **also has some disadvantages** that you should keep in mind:
  - **Less flexibility**: The pyplot interface is less flexible than the object-oriented interface, as it does not give you as much control over the details of the figure and axes.
  - **Lower ability to create complex charts**: If you want to create more complex and customized charts, such as 3D scatter plots or charts with multiple axes, the pyplot interface may not be the best option.
  - **Lower ability to modify already created charts**: If you want to modify a chart already created with the pyplot interface, it may be more difficult than with the object-oriented interface.
- Overall, the **pyplot interface** is an e**xcellent choice** for **quick and simple tasks**, such as creating basic charts and exploring data. But what if we want to create more complex charts? In that case, the pyplot interface may not be the best option. While it is very simple and easy to use, it is less flexible than the object-oriented interface.

Object-Oriented Interface

- The Matplotlib ***object interface*** provides **more control** and **flexibilit**y over the final chart
- By using it, you can create more complex and customized charts, as you can access and modify each element of the chart individually.
- Additionally, **this interface is more consistent with the rest of the Python library** and is considered the “official” way to use Matplotlib.
- **Here is an example** that shows how the object interface can be more powerful and flexible than the pyplot interface:

Object-Oriented Interface

- *Imagine that you have two sets of data that you want to compare on a chart.*
  - *The first set of data represents the number of products sold by your company in each month of the year and the second set of data represents the number of hours worked by each employee in each month.*
  - *You want to see if there is any relationship between the number of products sold and the number of hours worked.*
- To do this, you can **create a dual-axis** chart using Matplotlib.
- The first axis will be used to show the number of products sold and the second axis will be used to show the number of hours worked.

Object-Oriented Interface

```
import matplotlib.pyplot as plt
# Data for the plot
months = ['January', 'February', 'March', 'April']
products_sold = [110, 90, 70, 60]
hours_worked = [190, 170, 150, 130]
# Create a figure and an axis
fig, ax = plt.subplots()
# Add titles to the axes and the plot
ax.set_xlabel('Month')
ax.set_ylabel('Products sold', c = 'b')

# Plot the line for products sold data
ax.plot(months, products_sold, color='b', marker='o', linestyle='--')
# Create a second axis and plot the line for hours worked data
ax2 = ax.twinx()
ax2.plot(months, hours_worked, color='r', marker='s', linestyle='-.')
ax2.set_ylabel('Hours worked', c = 'r')
plt.title('Sales and hours worked per month')
# Show the plot
plt.show()
```

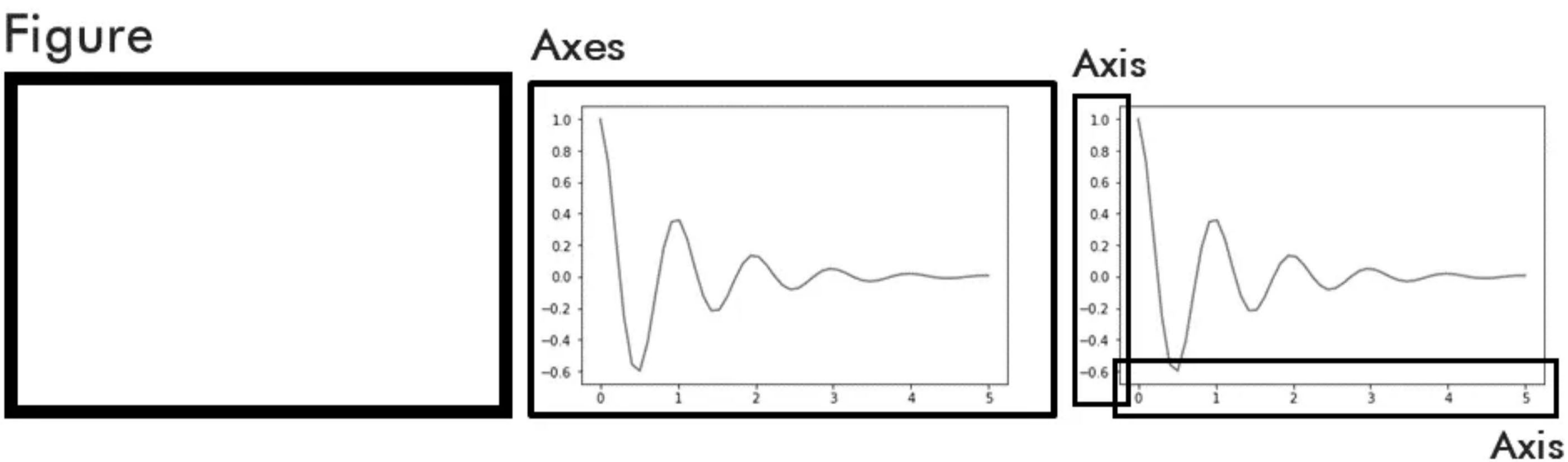
Object-Oriented Interface

- In addition to the greater flexibility and control it offers, there are **some other advantages** to using the Matplotlib ***object interface***:
  - **Consistency with the rest of Python**: The Matplotlib object interface uses the object-oriented programming style that is common in Python, making it more consistent with the rest of the library.
  - **Greater performance**: The object interface is faster and more efficient than the pyplot interface, as it does not have to do as much work behind the scenes.
  - **Greater ease of use in the long term**: While it may be a bit harder to use at first, in the long term the object interface is easier to maintain and extend.
  - **Greater flexibility for complex plots**: If you need to create more complex and customized plots, the object interface is the best option. It allows you to access and modify each element of the plot independently, giving you greater control over the final appearance.

- here are three issues that cause confusion:
  - The somewhat awkward nomenclature used for plots.
  - The co-existence of *two* plotting interfaces which I'll call the *pyplot approach* and the *object-oriented style*.
  - Plot manipulation methods in the two interfaces that have *similar* but *different* names.

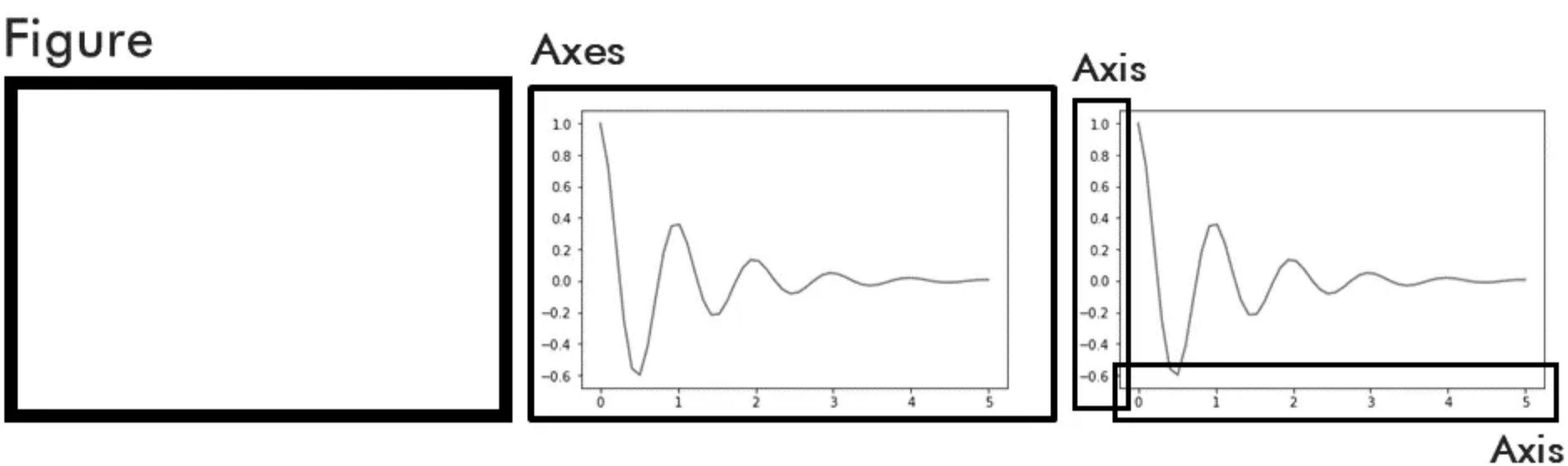
Matplotlib Objects - Figure

- Plots in Matplotlib are held within a Figure object. This is a blank canvas that represents the *top-level container* for all plot elements. Besides providing the canvas on which the plot is drawn, the Figure object also controls things like the size of the plot, its aspect ratio, the spacing between multiple plots drawn on the same canvas, and the ability to output the plot as an image.



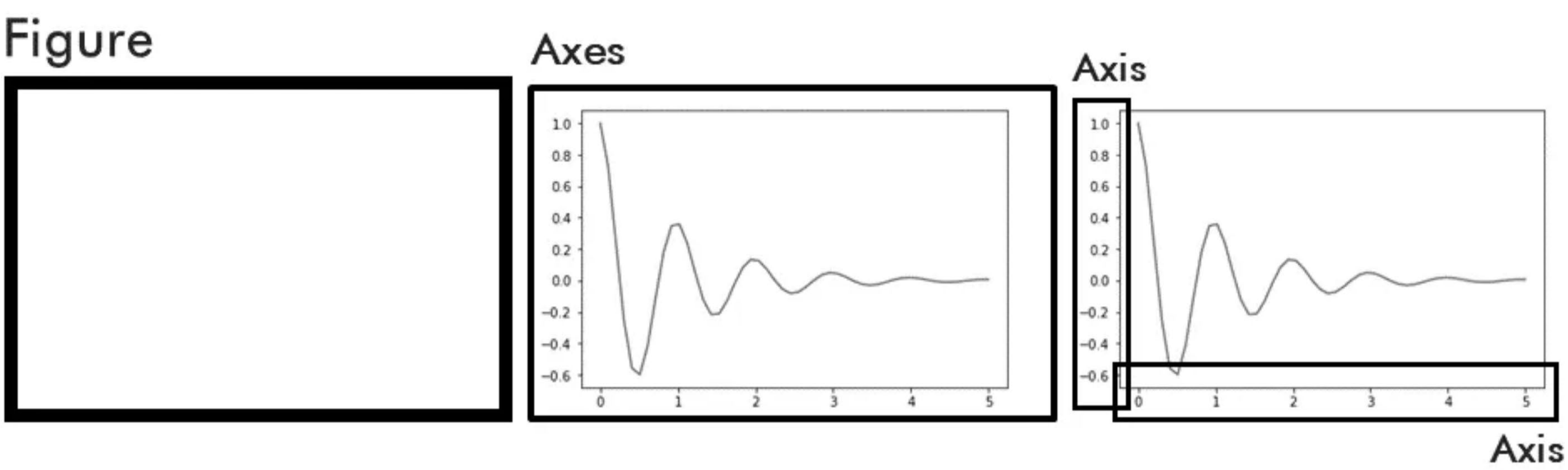
Matplotlib Objects - Axes

- The plots themselves are represented by the Axes class.
- This class includes most of the figure elements, such as lines, polygons, markers (points), text, titles, and so on, as well as the methods that act on them. It also sets the coordinate system. A Figure object can contain multiple Axes objects, but each Axes object can belong to only one Figure.



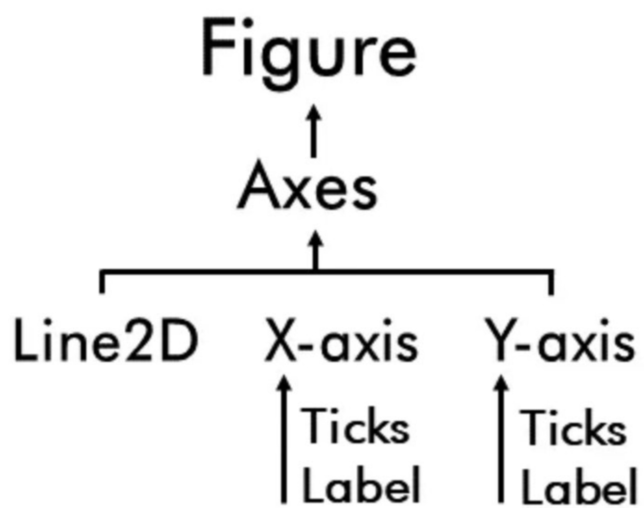
Matplotlib Objects - Axis

- The Axes object should not be confused with the Axis element that represents the numerical values on, say, the x- or y-axis of a chart.
- This includes the tick marks, labels, and limits. All these elements are contained within the Axes class.

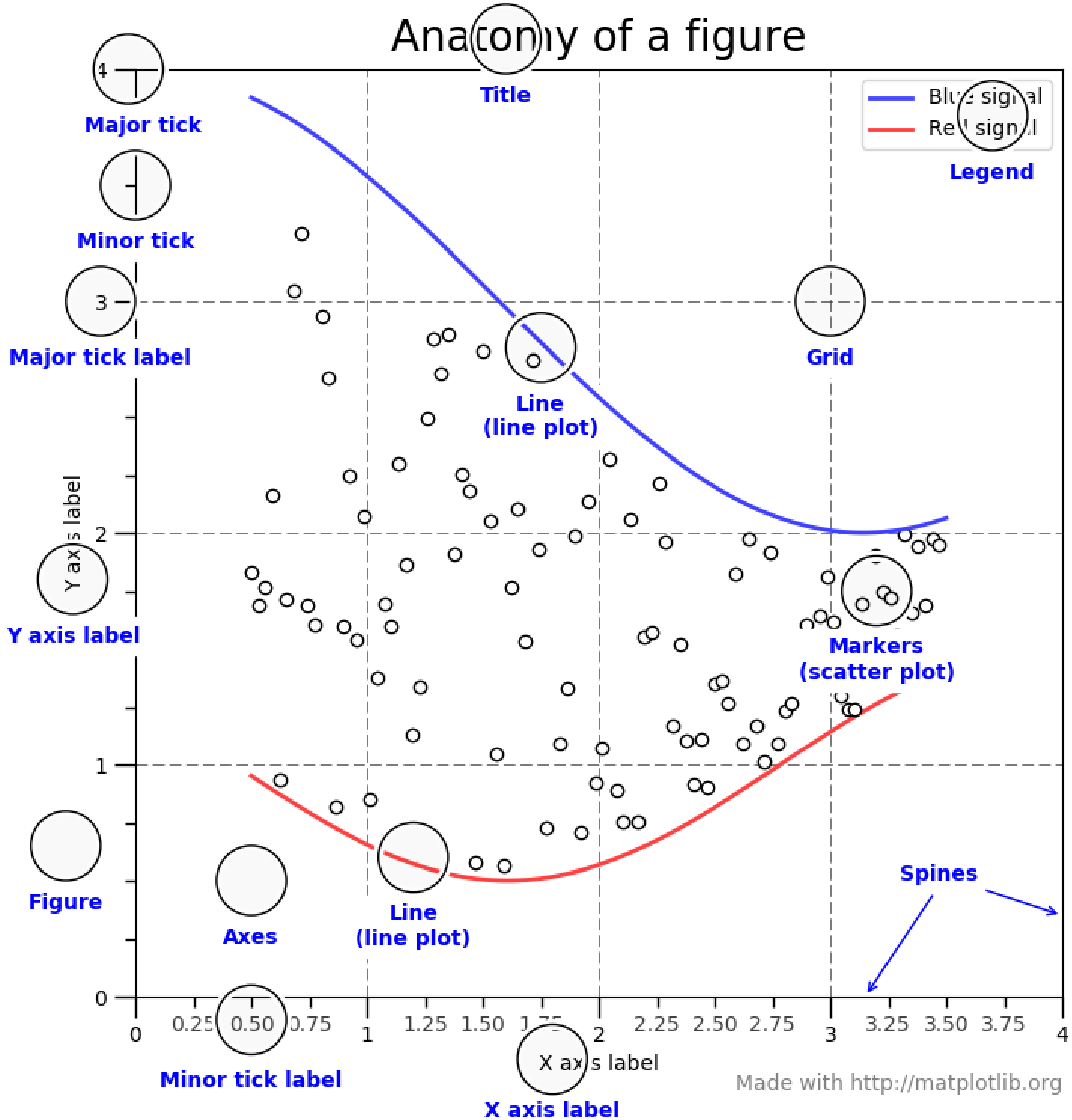


Matplotlib Objects - Hierarchy

- The Matplotlib Objects exist within a hierarchical structure.
- The lowest layer includes elements such as each axis, the axis tick marks and labels, and the curve (Line2D). The highest level is the Figure object, which serves as a container for everything below it.

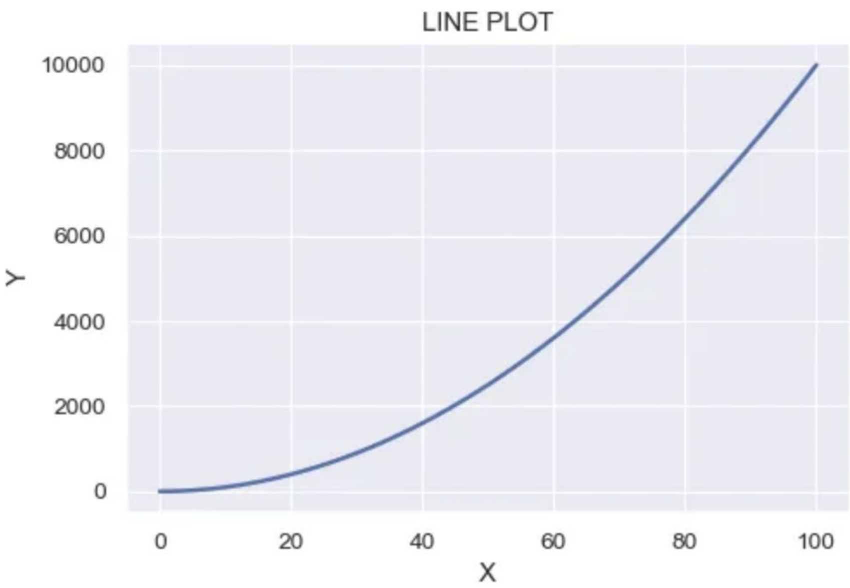


Matplotlib – Parameters & Terminology



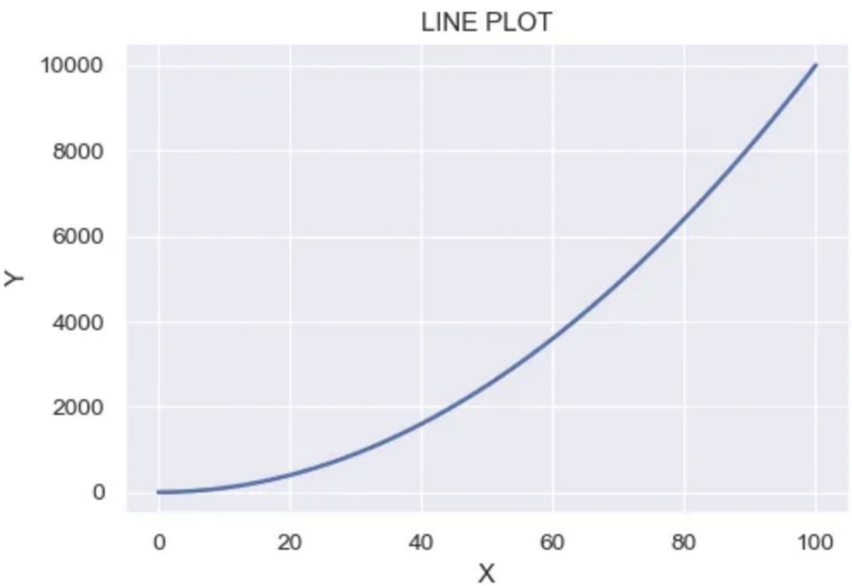
Pyplot Interface

```
import matplotlib.pyplot as plt
# Generate data
x = np.linspace(0, 100, 100)
y = x**2
# plot data
plt.plot(x, y)
# add title and axes label
plt.title("LINE PLOT")
plt.xlabel("X")
plt.ylabel("Y")
# Show plot
plt.show()
```



Object-Oriented Interface

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
# Define plot style
sns.set_theme()
# Create figure and axes
fig, ax = plt.subplots(dpi = 90)
# Generate data
x = np.linspace(0, 100, 100)
y = x**2
# Plot data
ax.plot(x, y, lw = 2)
# Add title and axes label
ax.set_title("LINE PLOT")
ax.set_xlabel("X")
ax.set_ylabel("Y")
# Show plot
plt.show()
```



Pyplot Interface – Plot Creation Methods

Useful Plot Creation Methods - pyplot

Method	Description	Example
`bar`	Make a bar chart	`plt.bar(x, height, width=0.8)`
`barh`	Make a horizontal bar chart	`plt.barh(x, height)`
`contour`	Draw a contour map	`plt.contour(X, Y, Z)`
`contourf`	Draw a filled contour map	`plt.contourf(X, Y, Z, cmap='Greys')`
`hist`	Make a 2D histogram	`plt.hist(x, bins)`
`pie`	Display a pie chart	`plt.pie(x=[8, 80, 9], labels=['A', 'B', 'C'])`
`plot`	Plot data as lines/markers	`plt.plot(x, y, 'r+')  # Red crosses`
`Polar`	Make a polar plot	`plt.polar(theta, r, 'bo')  # Blue dots`
`Scatter`	Make a scatterplot	`plt.scatter(x, y, marker='o')`
`stem`	Plot vertical lines to y coordinate	`plt.stem(x, y)`

Pyplot Interface – Plot Manipulation Methods - I

Method	Description	Example
annotate	Add text, arrows to Axes	`plt.annotate('text', (x, y))`
axis	Set axis properties (min, max)	`plt.axis([xmin, xmax, ymin, ymax])`
axhline	Add a horizontal line	`plt.axhline(y_loc, lw=5)`
axvline	Add a vertical line	`plt.axvline(x_loc, lw=3, c='red')`
close	Close a plot	`plt.close()`
draw	Update if interactive mode off	`plt.draw()`
figure	Create or activate a figure	`plt.figure(figsize=(4.0, 6.0))`
grid	Add grid lines	`plt.grid()`
imshow	Display data as an image	`pic = plt.imread('img.png')` `plt.imshow(pic, cmap='gray'))`
legend	Place a legend on the Axes	`plt.plot(data, label='Data')` `plt.legend()`
loglog	Use log scaling on each axis	`plt.loglog()`
minorticks_off	Remove minor ticks from axis	`plt.minorticks_off()`
minorticks_on	Display minor ticks on axis	`plt.minorticks_on()`
savefig	Save as *.jpg, .png, .pdf,* and so on	`plt.savefig('filename.jpg')`

Pyplot Interface – Plot Manipulation Methods - II

Method	Description	Example
semilogx	Use log scaling on x-axis	plt.semilogx()
semiology	Use log scaling on y-axis	plt.semilogy()
set_cmap	Set colormap	plt.set_cmap('Greens')
show	Show plot run from terminal or when interactive mode is off	plt.show()
subplot	Create subplots on a figure	plt.subplot(nrows, ncols, index)
text	Add text to the Axes	plt.text(x, y, 'text')
tight_layout	Adjust padding in subplots	plt.tight_layout(pad=3)
title	Add a title to the Axes	plt.title('text')
xkcd	Turn on xkcd sketch-style*	plt.xkcd()
xlabel	Set the x-axis label	plt.xlabel('text')
xlim	Set x-axis limits	plt.xlim(xmin, xmax)
xticks	Set tick information	plt.xticks([0, 2], rotation=30)
ylabel	Set the y-axis label	plt.ylabel('text')
ylim	Set y-axis limits	plt.ylim(ymin, ymax)
yticks	Set tick information	plt.yticks([0, 2], rotation=30)

OO Interface – Plot Creation Methods

Useful Plot Creation Methods – Object-oriented Style

Method	Description	Example
`bar`	Make a bar chart	`ax.bar(x, height)`
`barh`	Make a horizontal bar chart	`ax.barh(x, height)`
`contour`	Draw a contour map	`ax.contour(X, Y, Z)`
`contourf`	Draw a filled contour map	`ax.contourf(X, Y, Z, cmap='Greys')`
`hist`	Make a 2D histogram	`ax.hist(x, bins)`
`pie`	Display a pie chart	`ax.pie(x=[8, 80, 9], labels=['A', 'B', 'C'])`
`plot`	Plot data as lines/markers	`ax.plot(x, y, 'r+') # Red crosses`
`polar`	Make a polar plot	`fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})` `ax.plot(theta, r, 'bo') # Blue dots`
`scatter`	Make a scatterplot	`ax.scatter(x, y, marker='o')`
`stem`	Plot vertical lines to y coordinate	`ax.stem(x, y)`

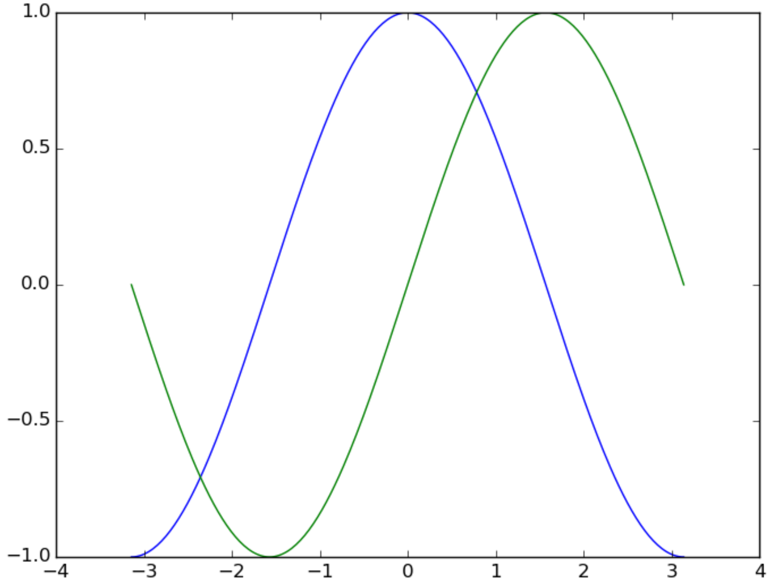
OO Interface – Plot Manipulation Methods

Useful Plot Manipulation Methods – Object-oriented Style

Method	Description	Example
`add_subplot`	Add or retrieve an Axes	`ax = fig.add_subplot(2, 2, 1)`
`close()`	Close a figure	`plt.close(fig2)`
`colorbar`	Add a colorbar to an Axes	`fig.colorbar(image, ax=ax)`
`constrained_layout`	Auto-adjust fit of subplots	`fig, ax = plt.subplots(constrained_layout=True)`
`gca`	Get the current Axes instance on the current figure	`fig.gca()`
`savefig`	Save as *.jpg*, *.png*, *.pdf*, and so on	`fig.savefig('filename.jpg')`
`set_size_inches`	Set Figure size in inches	`fig.set_size_inches(6, 4)`
`set_dpi`	Set Figure dots per inch	`fig.set_dpi(200) # Default is 100.`
`show`	Show plot run from terminal or when interactive mode is off	`plt.show()`
`subplots`	Create Figure with Axes	`fig, ax = plt.subplots(2, 2)`
`suptitle`	Add a super title to a Figure	`fig.suptitle('text')`
`tight_layout`	Auto-adjust subplots fit	`fig.tight_layout()`

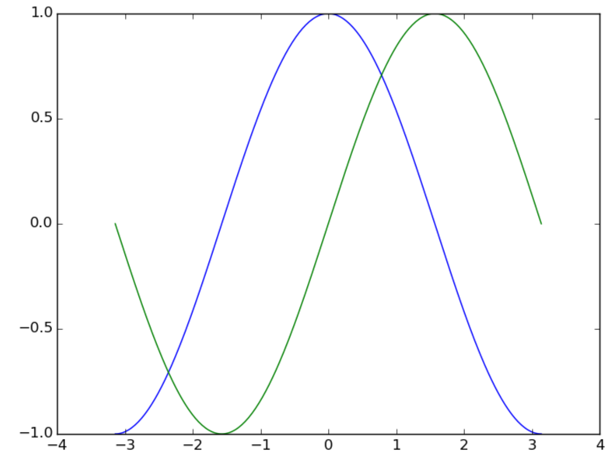
Matplotlib – First Plot

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C)
plt.plot(X, S)
plt.show()
```

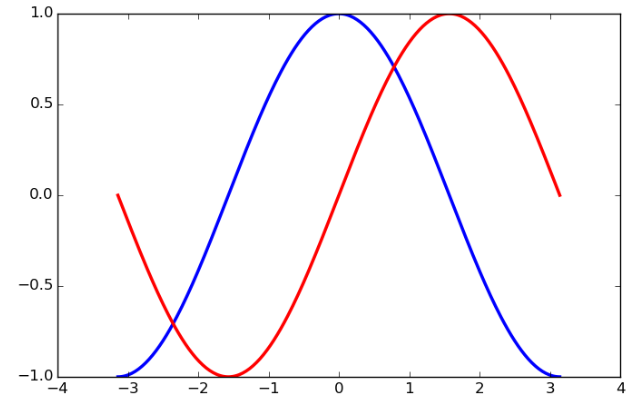


Matplotlib – Default Plot Parameters

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6), dpi=80)
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")
plt.xlim(-4.0, 4.0)
plt.xticks(np.linspace(-4, 4, 9, endpoint=True))
plt.ylim(-1.0, 1.0)
plt.yticks(np.linspace(-1, 1, 5, endpoint=True))
plt.show()
```



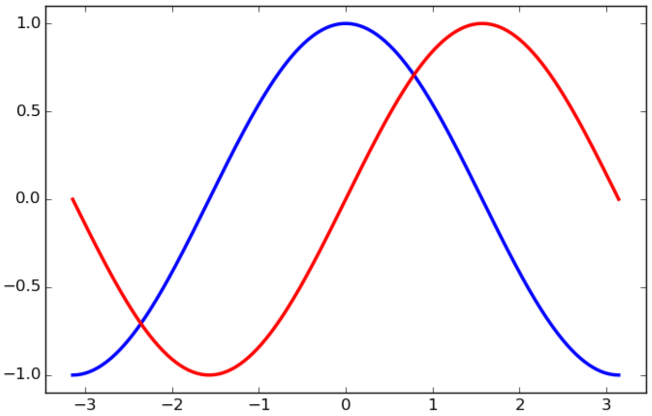
Matplotlib – Colors & LineStyles




```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5), dpi=80)
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
plt.xlim(-4.0, 4.0)
plt.xticks(np.linspace(-4, 4, 9, endpoint=True))
plt.ylim(-1.0, 1.0)
plt.yticks(np.linspace(-1, 1, 5, endpoint=True))
plt.show()
```

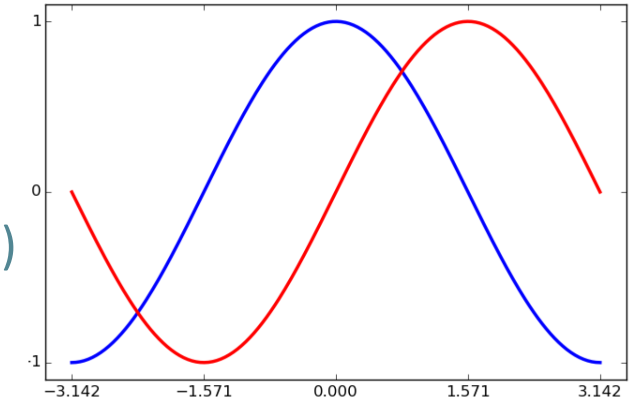
Matplotlib – Axis Limits

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5), dpi=80)
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
S = np.sin(X)
C = np.cos(X)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.ylim(C.min() * 1.1, C.max() * 1.1)
plt.show()
```

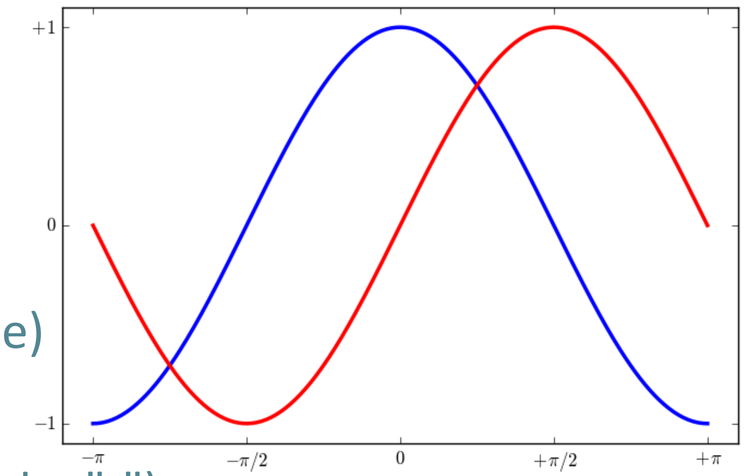


Matplotlib – Axis Ticks

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5), dpi=80)
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
S = np.sin(X)
C = np.cos(X)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.ylim(C.min() * 1.1, C.max() * 1.1)
plt.yticks([-1, 0, +1])
plt.show()
```



Matplotlib – Axis Ticks/Labels




```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5), dpi=80)
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi], [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.ylim(C.min() * 1.1, C.max() * 1.1)
plt.yticks([-1, 0, +1], [r'$-1$', r'$0$', r'$+1$'])
plt.show()
```

Methods with Axes objects - I

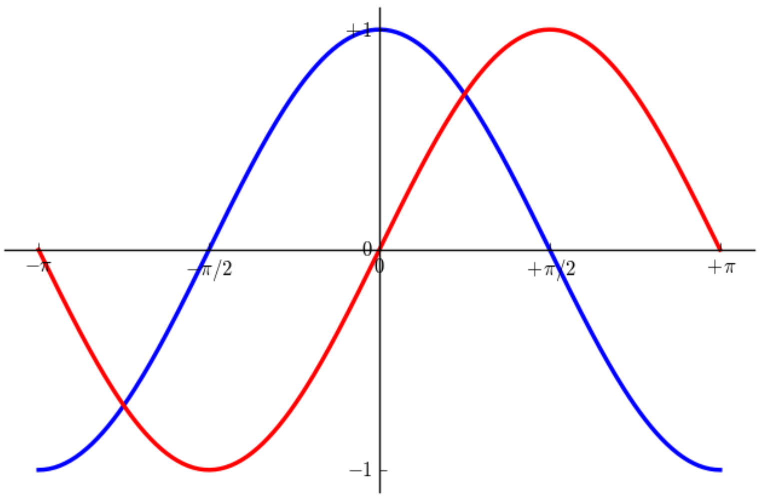
Method	Description	Example
annotate	Add text and arrows to Axes	ax.annotate('text', xy=(5, 2))
axis	Get or set axis properties	ax.axis([xmin, xmax, ymin, ymax])
axhline	Add a horizontal line	ax.axhline(y_loc, lw=5)
axvline	Add a vertical line	ax.axvline(x_loc, lw=3, c='red')
grid	Add grid lines	ax.grid()
imshow	Display data as an image	pic = plt.imread('img.png') ax.imshow(pic, cmap='gray'))
legend	Place a legend on the Axes	ax.plot(data, label='Data') ax.legend()
loglog	Use log scaling on each axis	ax.loglog()
minorticks_on	Display minor ticks on axis	ax.yaxis.get_ticklocs(minor=True) ax.minorticks_on()
minorticks_off	Remove minor ticks from axis	plt.minorticks_off()
semilogx	Use log scaling on x-axis	ax.semilogx()
semiology	Use log scaling on y-axis	ax.semilogy()
set	Set multiple properties at once	ax.set(title, ylabel, xlim, alpha)
set_title()	Set the Axes title	ax.set_title('text', loc='center')

Methods with Axes objects - II

Method	Description	Example
set_xticks()	Set x-axis tick marks	`xticks = np.arange(0, 100, 10)` `ax.set_xticks(xticks)`
set_yticks()	Set y-axis tick marks	`yticks = np.arange(0, 100, 10)` `ax.set_yticks(yticks)`
set_xticklabels	Set x-axis labels after calling set_xticks()	`labels = [a', 'b', 'c', 'd']` `ax.set_xticklabels(labels)`
set_yticklabels	Set y-axis labels after calling set_yticks()	`ax.set_yticklabels([1, 2, 3, 4])`
tick_params	Change ticks, labels, and grid	`ax.tick_params(labelcolor= 'red')`
twinx	New y-axis with shared x-axis	`ax.twinx()`
twiny	New x-axis with shared y-axis	`ax.twiny()`
set_xlabel()	Set label for x-axis	`ax.set_xlabel('text', loc='left')`
set_ylabel()	Set label for y-axis	`ax.set_ylabel('text', loc='top')`
set_xlim()	Set limits of x-axis	`ax.set_xlim(-5, 5)`
set_ylim()	Set limits of y-axis	`ax.set_ylim(0, 10)`
set_xscale()	Set the x-axis scale	`ax.set_xscale('log')`
set_yscale()	Set the y-axis scale	`ax.set_yscale('linear')`
text	Add text to the Axes	`ax.text(x, y, 'text')`
xaxis.grid()	Add x-axis grid lines	`ax.xaxis.grid(True, which='major')`
yaxis.grid()	Add y-axis grid lines	`ax.yaxis.grid(True, which='minor')`

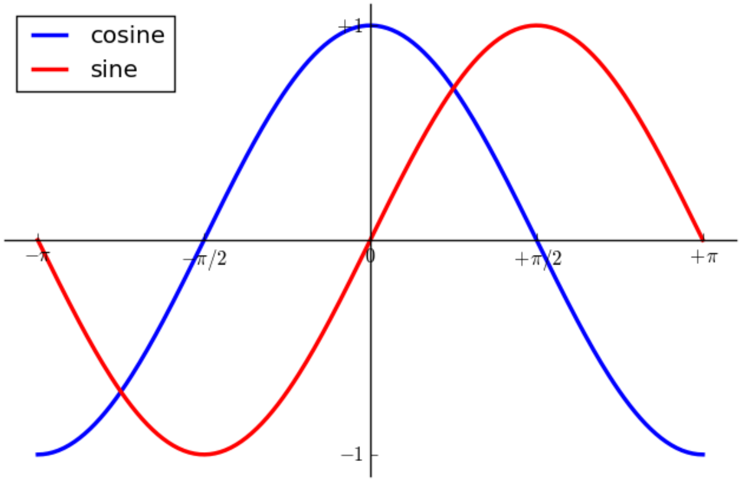
Matplotlib – Position of Axes

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5), dpi=80)
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
C, S = np.cos(X) , np.sin(X)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi], [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.ylim(C.min() * 1.1, C.max() * 1.1)
plt.yticks([-1, 0, +1], [r'$-1$', r'$0$', r'$+1$'])
plt.show()
```



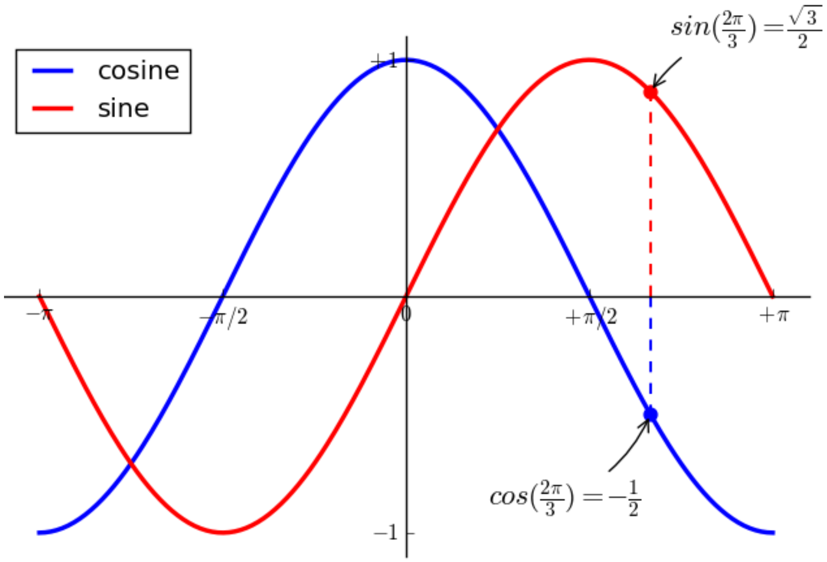
Matplotlib – Legend

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5), dpi=80)
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
C,S = np.cos(X), np.sin(X)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosine")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-", label="sine")
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi], [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.ylim(C.min() * 1.1, C.max() * 1.1)
plt.yticks([-1, +1], [r'$-1$', r'$+1$'])
plt.legend(loc='upper left')
plt.show()
```



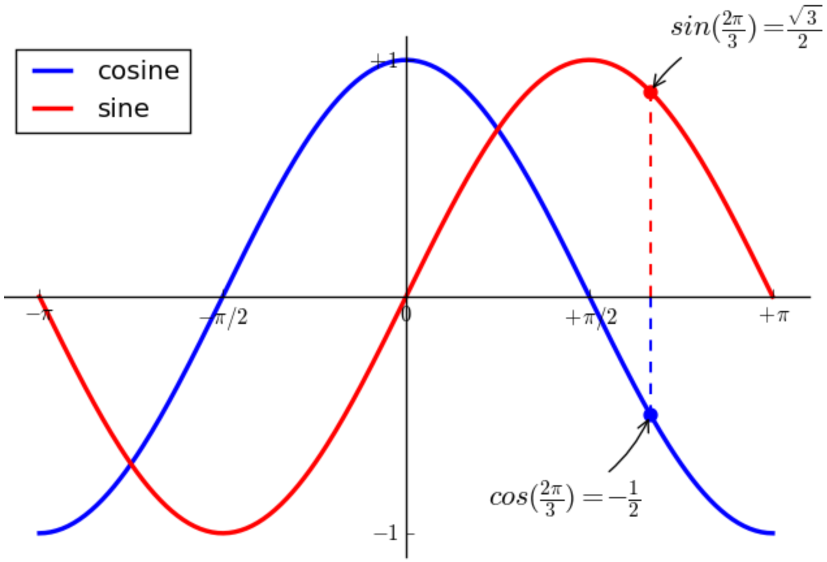
Matplotlib – Annotation

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5), dpi=80)
plt.subplot(111)
.
.
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')
plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
xy=(t, np.sin(t)), xycoords='data',
xytext=(+10, +30), textcoords='offset points', fontsize=16,
arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
.
.
plt.scatter([t, ], [np.sin(t), ], 50, color='red')
plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$', xy=(t, np.cos(t)),
xycoords='data', xytext=(-90, -50), textcoords='offset points',
fontsize=16,  arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
plt.show()
```



Matplotlib – Annotation

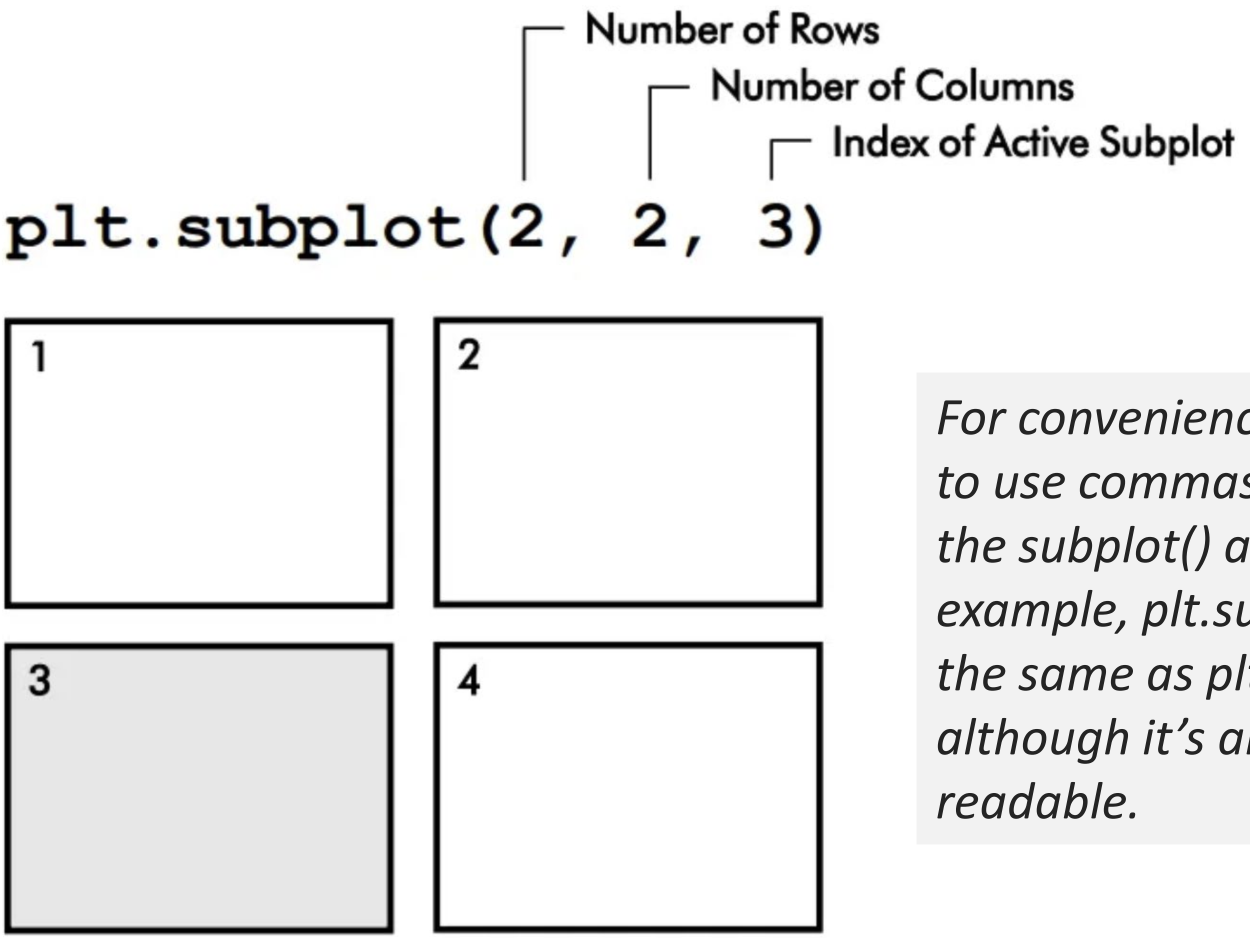
```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5), dpi=80)
plt.subplot(111)
.
.
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')
plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
xy=(t, np.sin(t)), xycoords='data',
xytext=(+10, +30), textcoords='offset points', fontsize=16,
arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
.
.
plt.scatter([t, ], [np.sin(t), ], 50, color='red')
plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$', xy=(t, np.cos(t)),
xycoords='data', xytext=(-90, -50), textcoords='offset points',
fontsize=16,  arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
plt.show()
```



Subplots

- The subplots will be arranged in a grid, and the first two arguments passed to the subplot() method specify the dimensions of this grid.
- The first argument represents the number of rows in the grid, the second is the number of columns, and the third argument is the index of the active subplot.
- The active subplot is the one you are currently plotting in when you call a method like plot() or scatter(). Unlike most things in Python, the first index is 1, not 0.
- Matplotlib uses a concept called the “current figure” to keep track of which Axes is currently being worked on. For example, when you call plt.plot(), pyplot creates a new “current figure” Axes to plot on. When working with multiple subplots, the index argument tells pyplot which subplot represents the “current figure.”

Pyplot Interface - Subplots

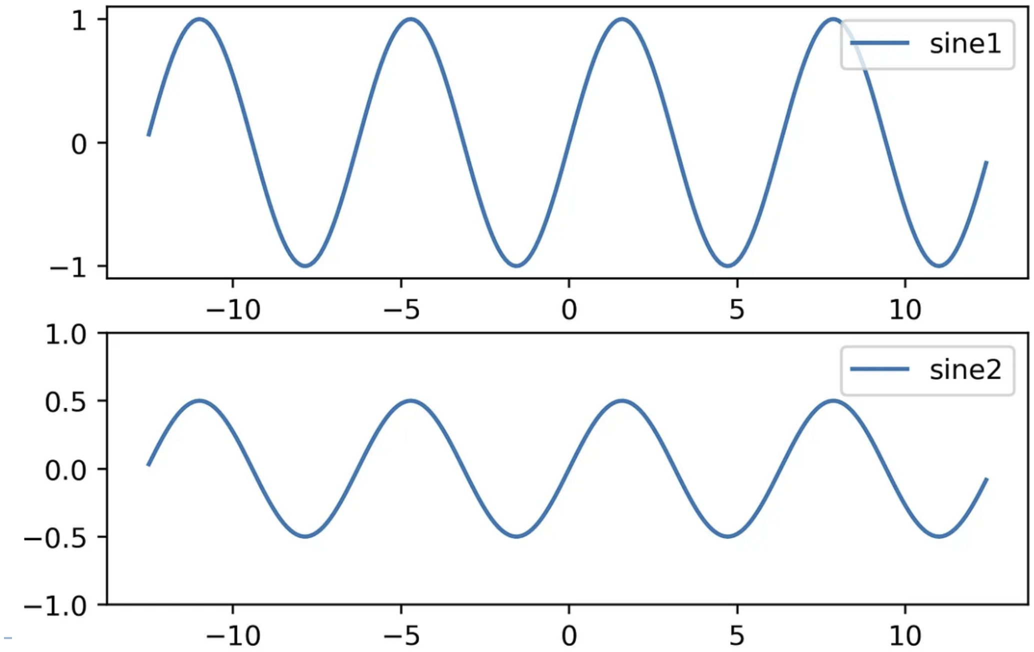


For convenience, you don't need to use commas with the subplot() arguments. For example, plt.subplot(223) works the same as plt.subplot(2, 2, 3), although it's arguably less readable.

Pyplot Interface - Subplots

```
plt.subplot(2, 1, 1)
plt.plot(time, amplitude, label='sine1')
plt.legend(loc='upper right')

plt.subplot(2, 1, 2)
plt.ylim(-1, 1)
plt.plot(time, amplitude_halved, label='sine2')
plt.legend(loc='best')
```



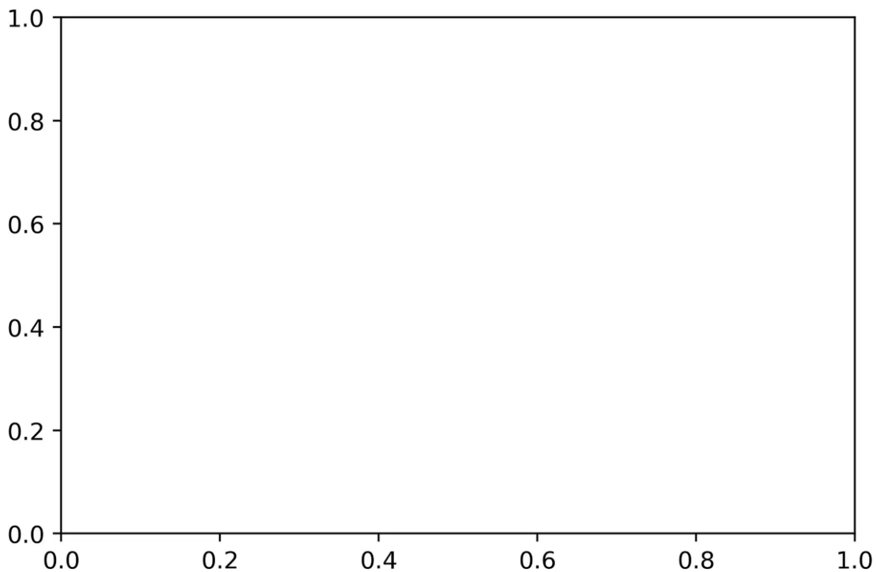
OO Interface - Subplots

- Like the pyplot approach, the object-oriented style supports the use of subplots.
- Although there are multiple ways to assign subplots to Figure and Axes objects, the plt.subplots() method is convenient and returns a NumPy array that lets you select subplots using standard indexing or with unique names such as axs[0, 0] or ax1.
- Another benefit is that you can preview the subplots' geometry prior to plotting any data.

OO Interface - Subplots

- Calling plt.subplots() with no arguments generates a single empty plot.
- Technically, this produces a 1×1 AxesSubplot object.

    fig, ax = plt.subplots()

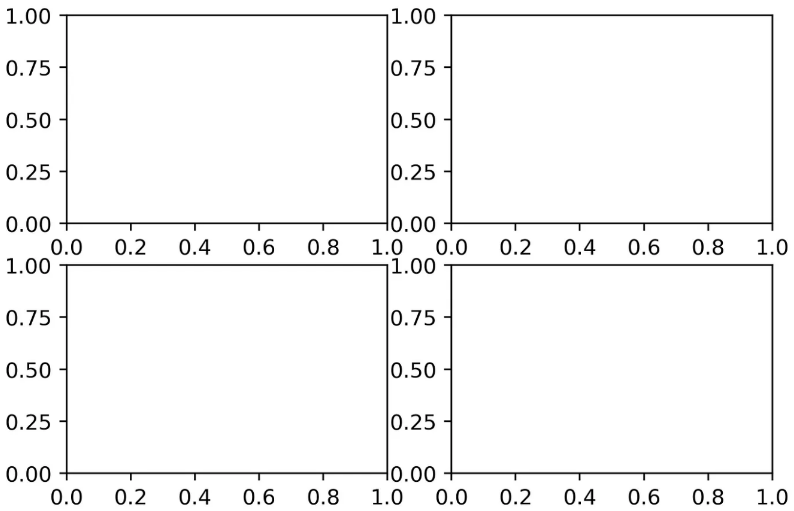


The object-oriented method for creating subplots is spelled subplots, *whereas the pyplot approach uses* subplot. *You can remember this by associating the simplest technique, pyplot, with the shortest name.*

OO Interface - Subplots

- Producing multiple subplots works like the plt.subplot() method, only without an index argument for the active subplot. The first argument indicates the number of rows; the second specifies the number of columns.
- By convention, multiple Axes are given the plural name, axs, rather than axes so as to avoid confusion with a single instance of Axes.

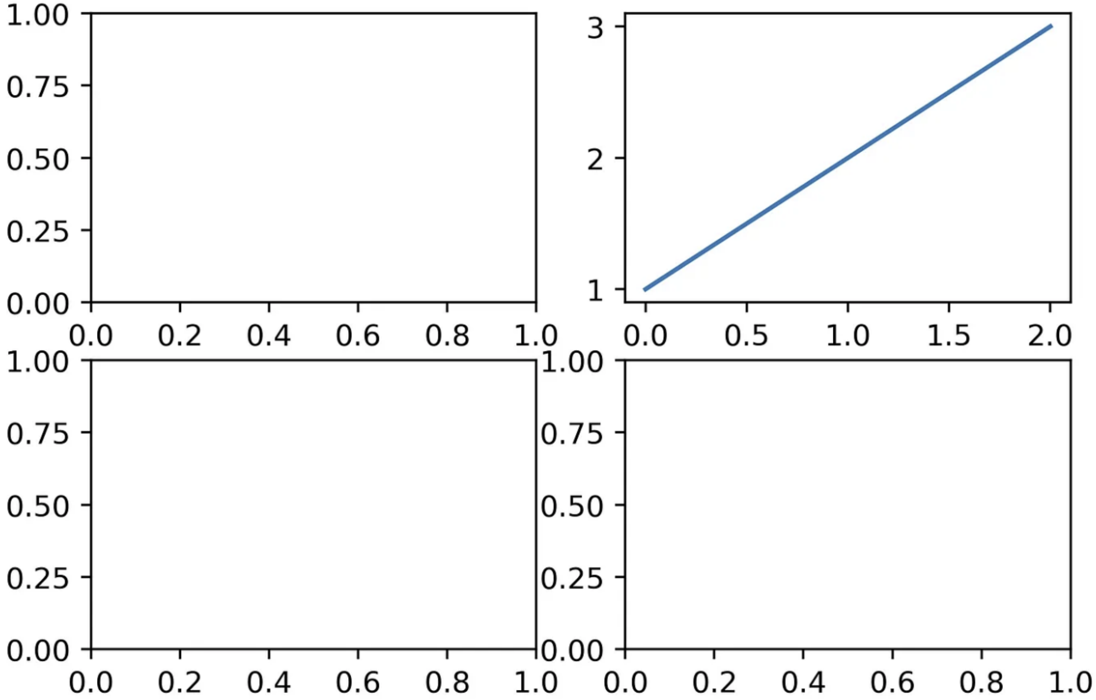
```
fig, axs = plt.subplots(2, 2)
```



OO Interface - Subplots

- To activate a subplot, you can use its index. In this example, we plot on the second subplot in the first row:

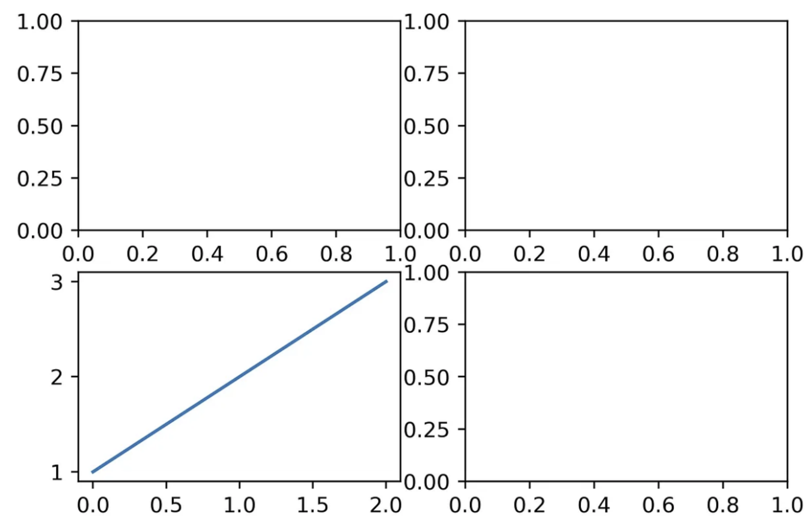
```
fig, axs = plt.subplots(2, 2)
axs[0, 1].plot([1, 2, 3])
```



OO Interface - Subplots

- Alternatively, you can name and store the subplots individually by using tuple unpacking for multiple Axes. Each row of subplots will need to be in its own tuple. You can then select a subplot using a name, versus a less-readable index:

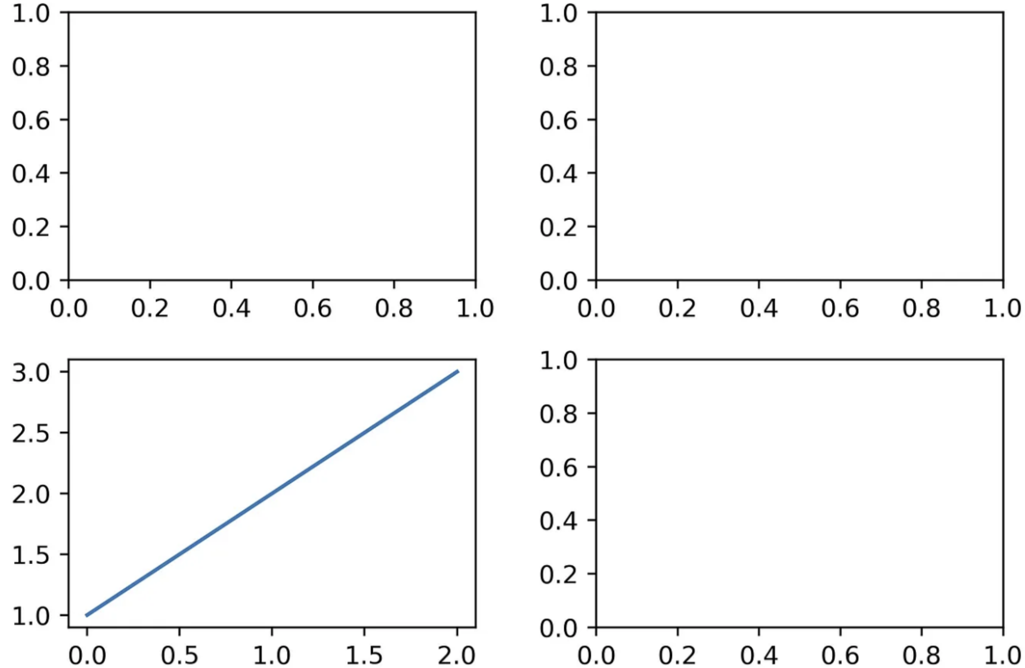
```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
ax3.plot([1, 2, 3])
```



OO Interface - Subplots

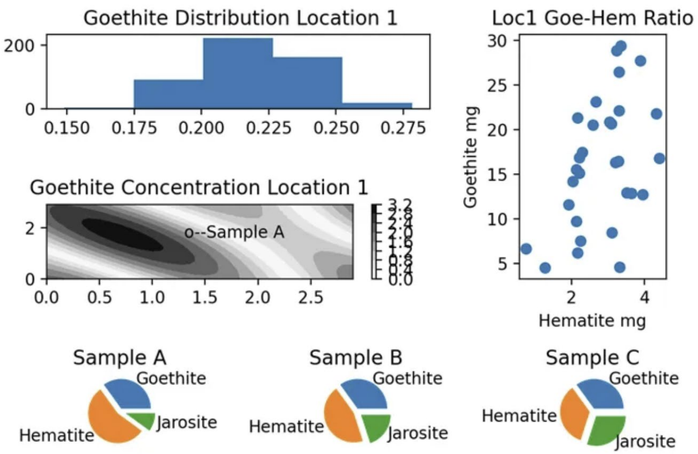
- In both the pyplot approach and object-oriented style, you can add whitespace around the subplots by calling the tight_layout() method on the Figure object:

```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
ax3.plot([1, 2, 3])
fig.tight_layout()
```



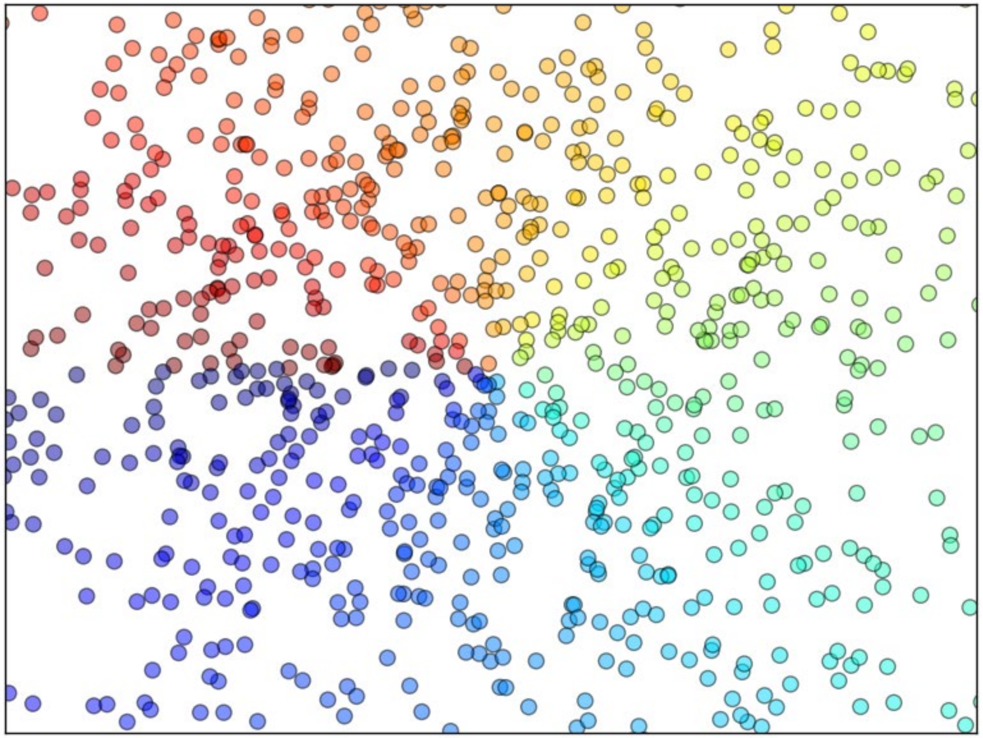
Alternative Ways to Make Subplots

- No matter which technique you use, there are higher-level alternatives to help you split a figure into a grid of subareas. This, in turn, helps you create subplots that have different widths and heights. The resulting multipaneled displays are useful for summarizing information in presentations and reports.
- Among these paneling tools are Matplotlib's GridSpec module and its subplot_mosaic() method. Here's an example built with GridSpec:



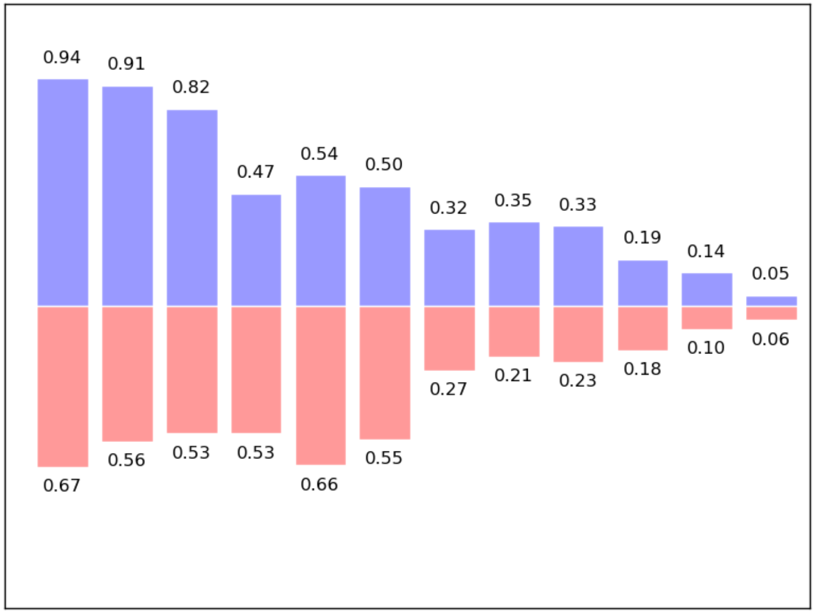
Matplotlib - Scatter Plots

```
import numpy as np
import matplotlib.pyplot as plt
n = 1024
X = np.random.normal(0, 1, n)
Y = np.random.normal(0, 1, n)
T = np.arctan2(Y, X)
plt.axes([0.025, 0.025, 0.95, 0.95])
plt.scatter(X, Y, s=75, c=T, alpha=.5)
plt.xlim(-1.5, 1.5)
plt.xticks(())
plt.ylim(-1.5, 1.5)
plt.yticks(())
plt.show()
```



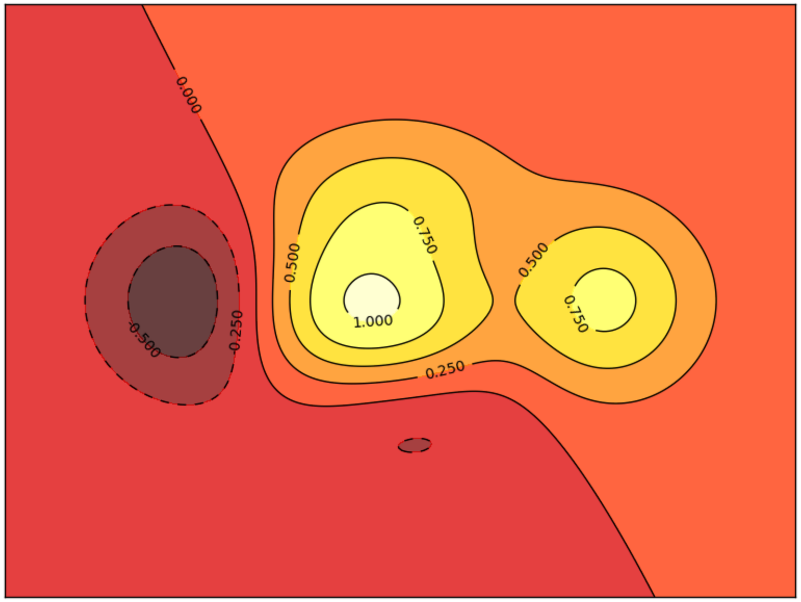
Matplotlib - Bar Plots

```
import numpy as np
import matplotlib.pyplot as plt
n = 12
X = np.arange(n)
Y1 = (1 - X / float(n)) * np.random.uniform(0.5, 1.0, n)
Y2 = (1 - X / float(n)) * np.random.uniform(0.5, 1.0, n)
plt.axes([0.025, 0.025, 0.95, 0.95])
plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')
for x, y in zip(X, Y1):
plt.text(x + 0.4, y + 0.05, '%.2f' % y, ha='center', va= 'bottom')
for x, y in zip(X, Y2):
plt.text(x + 0.4, -y - 0.05, '%.2f' % y, ha='center', va= 'top')
plt.xlim(-.5, n)
plt.xticks(())
plt.ylim(-1.25, 1.25)
plt.yticks(())
plt.show()
```



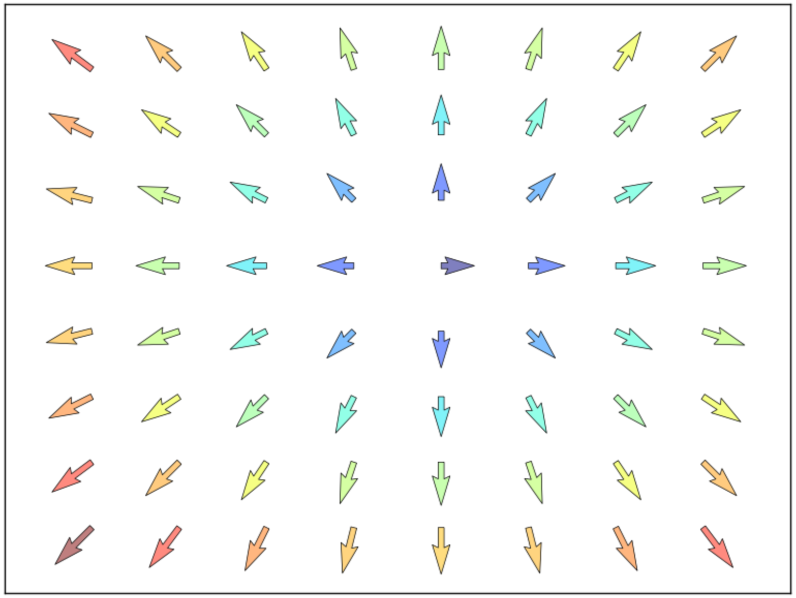
Matplotlib – Contour Plots

```
import numpy as np
import matplotlib.pyplot as plt
def f(x,y):
return (1 - x / 2 + x**5 + y**3) * np.exp(-x**2 -y**2)
n = 256
x = np.linspace(-3, 3, n)
y = np.linspace(-3, 3, n)
X,Y = np.meshgrid(x, y)
plt.axes([0.025, 0.025, 0.95, 0.95])
plt.contourf(X, Y, f(X, Y), 8, alpha=.75, cmap=plt.cm.hot)
C = plt.contour(X, Y, f(X, Y), 8, colors='black', linewidth=.5)
plt.clabel(C, inline=1, fontsize=10)
plt.xticks(())
plt.yticks(())
plt.show()
```



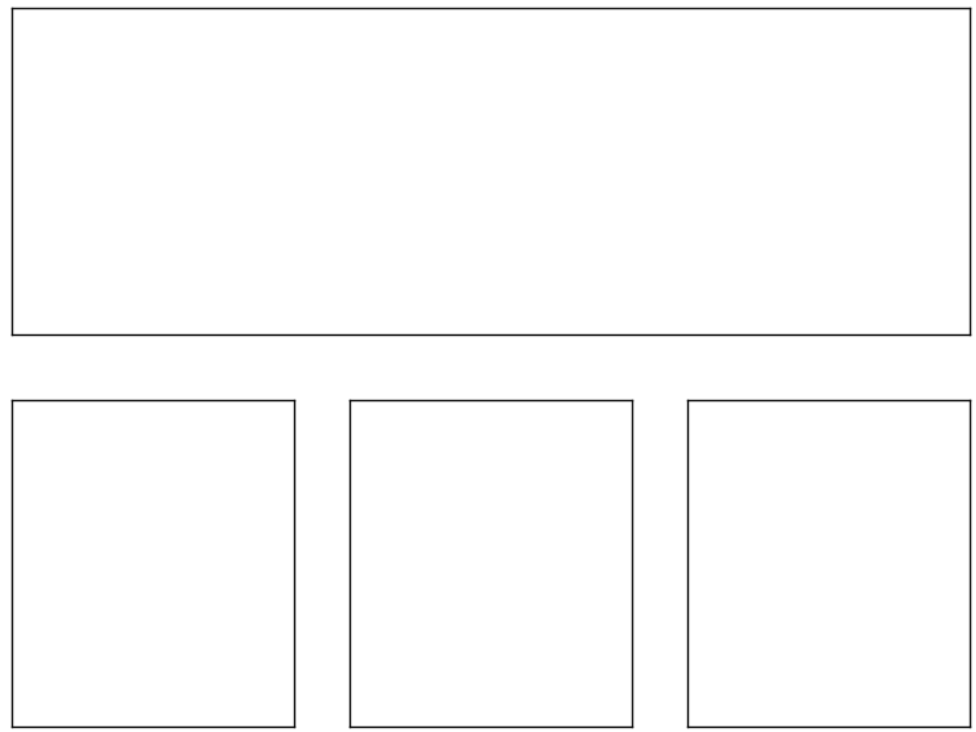
Matplotlib – Vector Field Plots

```
import numpy as np
import matplotlib.pyplot as plt
n = 8
X, Y = np.mgrid[0:n, 0:n]
T = np.arctan2(Y - n / 2., X - n/2.)
R = 10 + np.sqrt((Y - n / 2.0) ** 2 + (X - n / 2.0) ** 2)
U, V = R * np.cos(T), R * np.sin(T)
plt.axes([0.025, 0.025, 0.95, 0.95])
plt.quiver(X, Y, U, V, R, alpha=.5)
plt.quiver(X, Y, U, V, edgecolor='k', facecolor='None', linewidth=.5)
plt.xlim(-1, n)
plt.xticks(())
plt.ylim(-1, n)
plt.yticks(())
plt.show()
```



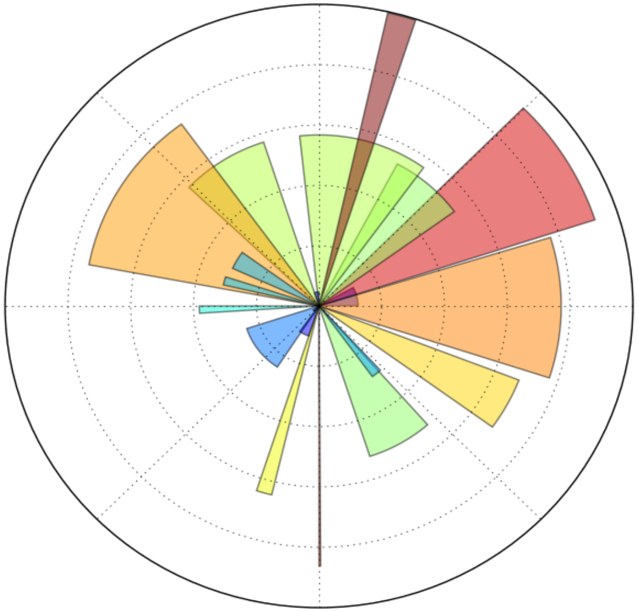
Matplotlib – Multiple/Sub Plots

```
import matplotlib.pyplot as plt
fig = plt.figure()
fig.subplots_adjust(bottom=0.025, left=0.025, top = 0.975, right=0.975)
plt.subplot(2, 1, 1)
plt.xticks(()), plt.yticks(())
plt.subplot(2, 3, 4)
plt.xticks(())
plt.yticks(())
plt.subplot(2, 3, 5)
plt.xticks(())
plt.yticks(())
plt.subplot(2, 3, 6)
plt.xticks(())
plt.yticks(())
plt.show()
```



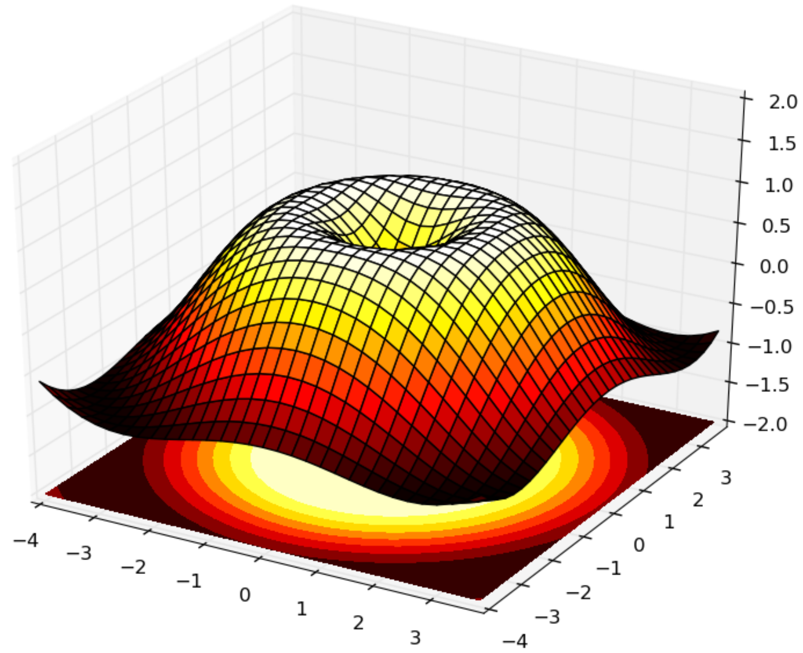
Matplotlib – Polar Plots

```
import numpy as np
import matplotlib.pyplot as plt
ax = plt.axes([0.025, 0.025, 0.95, 0.95], polar=True)
N = 20
theta = np.arange(0.0, 2 * np.pi, 2 * np.pi / N)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
bars = plt.bar(theta, radii, width=width, bottom=0.0)
for r,bar in zip(radii, bars):
bar.set_facecolor(plt.cm.jet(r/10.))
bar.set_alpha(0.5)
ax.set_xticklabels([])
ax.set_yticklabels([])
plt.show()
```



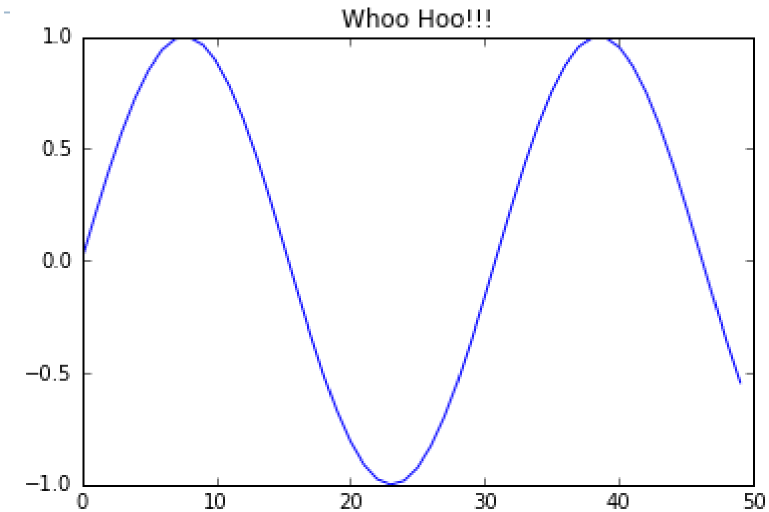
Matplotlib – 3D Surface Plots

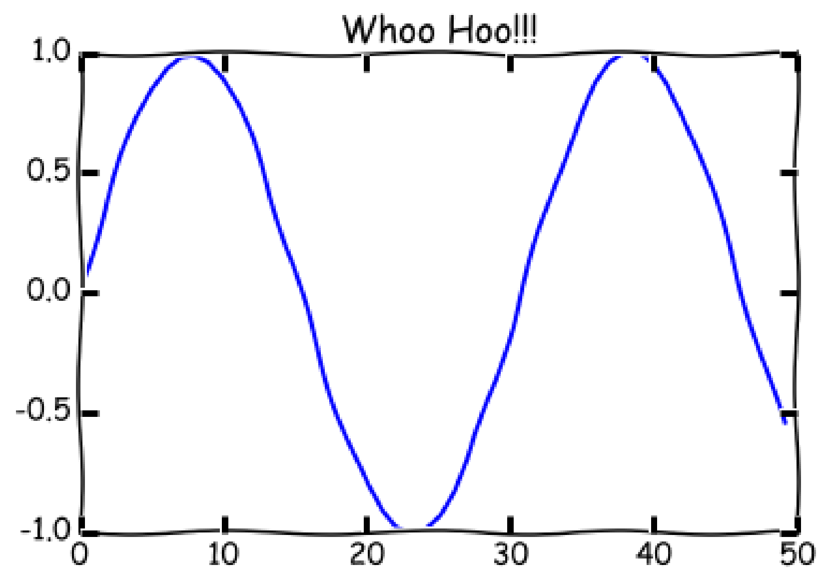
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X ** 2 + Y ** 2)
Z = np.sin(R)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.hot)
ax.contourf(X, Y, Z, zdir='z', offset=-2, cmap=plt.cm.hot)
ax.set_zlim(-2, 2)
plt.show()
```



XKCD Style Plots

```
import numpy as np
import matplotlib.pyplot as plt
plt.xkcd()
plt.plot(np.sin(np.linspace(0, 10)))
plt.title('Whoo Hoo!!!')
```

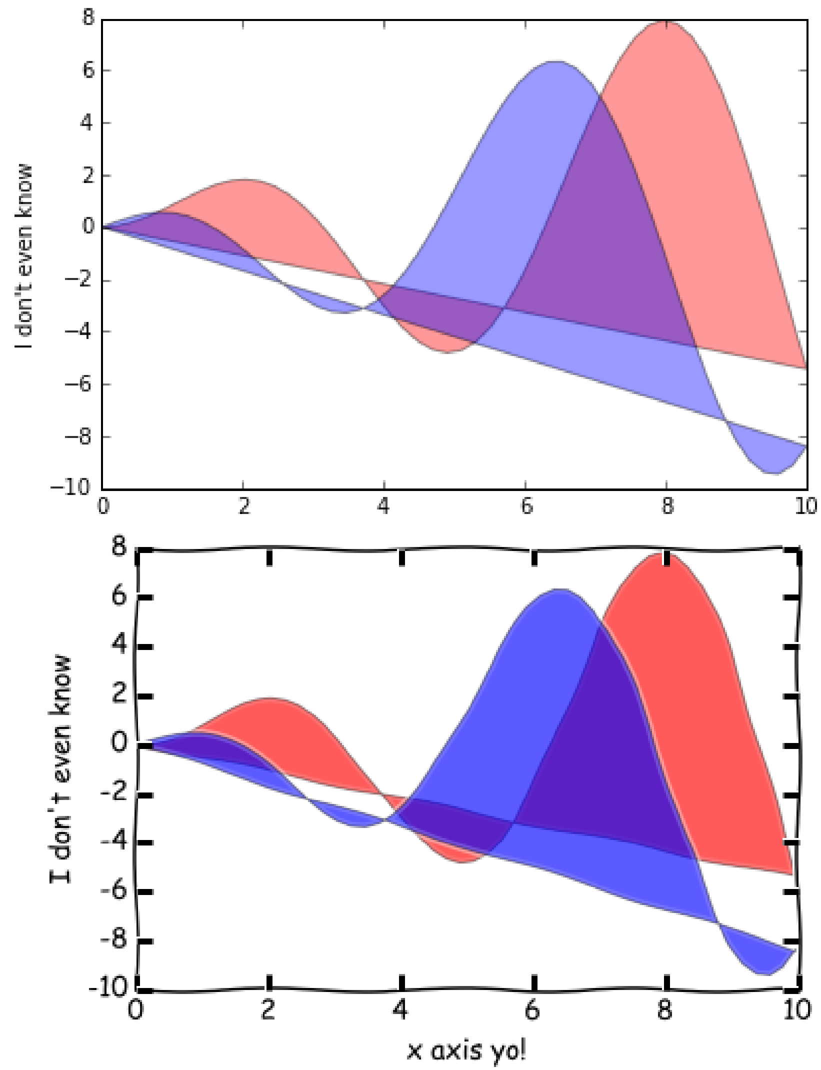




XKCD Style Plots

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10)
y1 = x * np.sin(x)
y2 = x * np.cos(x)

plt.xkcd()
plt.fill(x, y1, 'red', alpha=0.4)
plt.fill(x, y2, 'blue', alpha=0.4)
plt.xlabel('x axis yo!')
plt.ylabel("I don't even know")
```



Seaborn

- Seaborn is a library for making statistical graphics in Python.
- It builds on top of matplotlib and integrates closely with pandas data structures.
- Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.
- Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

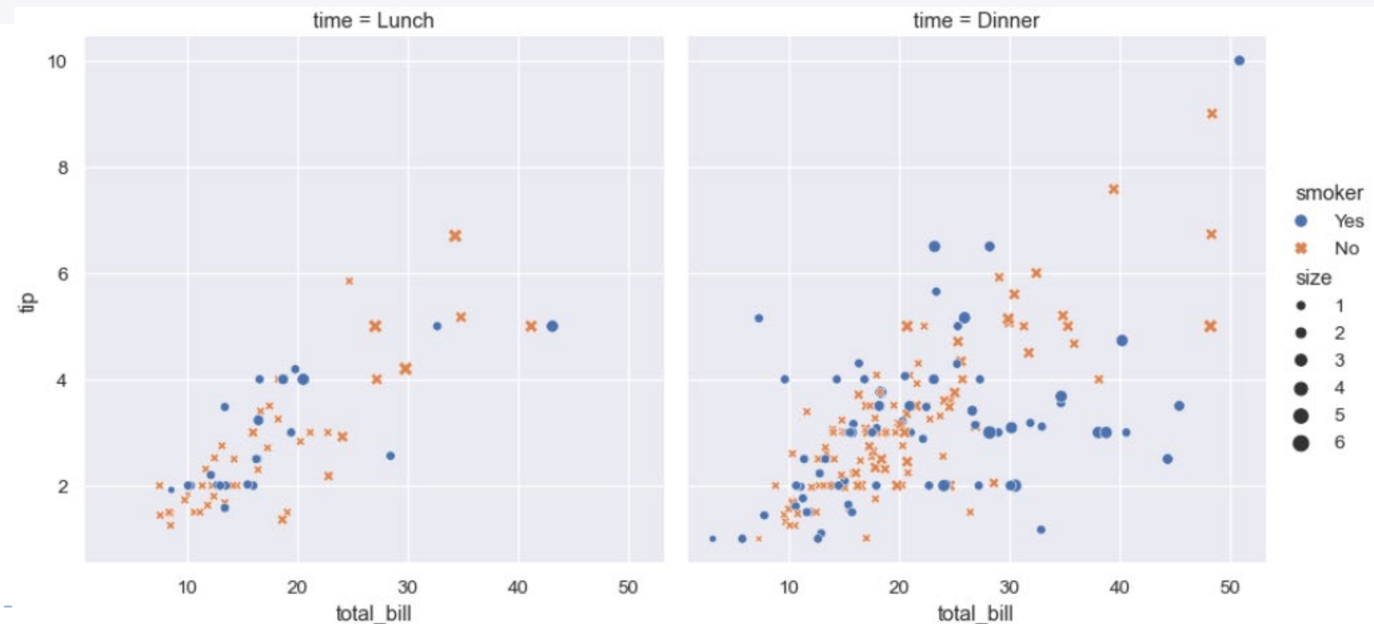
Why use Seaborn

- Seaborn allows you to make beautiful visualizations with very short codes.
- If you use pandas for your data analysis, it is a perfect match for you.
- Optimized for statistical analysis.
- It is a well-known and widespread tool among data scientists.
- Integrated with Pandas dataframes/series
- Many statistical and categorical analyses are readily available
- It can be used as an analysis tool rather than merely a graphics module

Seaborn – First Plot

```python
# Import seaborn
import seaborn as sns
# Apply the default theme
sns.set_theme()
# Load an example dataset
tips = sns.load_dataset("tips")
# Create a visualization
sns.relplot( data=tips, x="total_bill", y="tip", col="time", hue="smoker", style="smoker",
size="size", )
```

1	total_bill	tip	sex	smoker	day	time	size
2	16.99	1.01	Female	No	Sun	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3
4	21.01	3.5	Male	No	Sun	Dinner	3
5	23.68	3.31	Male	No	Sun	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4
7	25.29	4.71	Male	No	Sun	Dinner	4
8	8.77	2	Male	No	Sun	Dinner	2



Seaborn Tips Dataset

total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2
10.33	1.67	Female	No	Sun	Dinner	3
16.29	3.71	Male	No	Sun	Dinner	3
16.97	3.5	Female	No	Sun	Dinner	3
20.65	3.35	Male	No	Sat	Dinner	3
17.92	4.08	Male	No	Sat	Dinner	2
20.29	2.75	Female	No	Sat	Dinner	2
15.77	2.23	Female	No	Sat	Dinner	2
39.42	7.58	Male	No	Sat	Dinner	4

Seaborn – First Plot – Steps in Detail

- First, we need to import Seaborn. By convention it is nearly always imported as sns

```python
# Import seaborn
import seaborn as sns
```

- Seaborn uses matplotlib rcParam system for themes.

```python
# Apply the default theme
sns.set_theme()
```

- Seaborn imports data as Pandas Dataframe

```python
# Load an example dataset
tips = sns.load_dataset("tips")
```

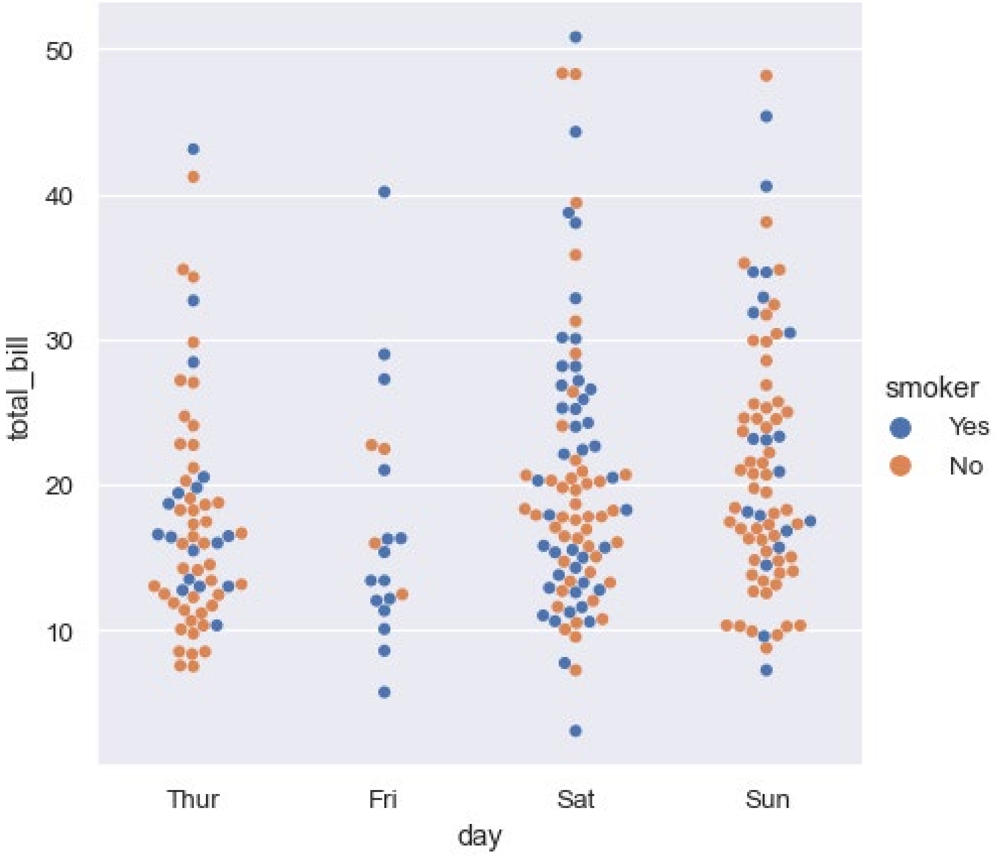
- Using a single call, we can show the relationship between five variables in the data by using relplot() command.

```python
# Create a visualization
sns.relplot( data=tips, x="total_bill", y="tip", col="time", hue="smoker",
style="smoker",
size="size", )
```

Seaborn – Categorical Data Plots

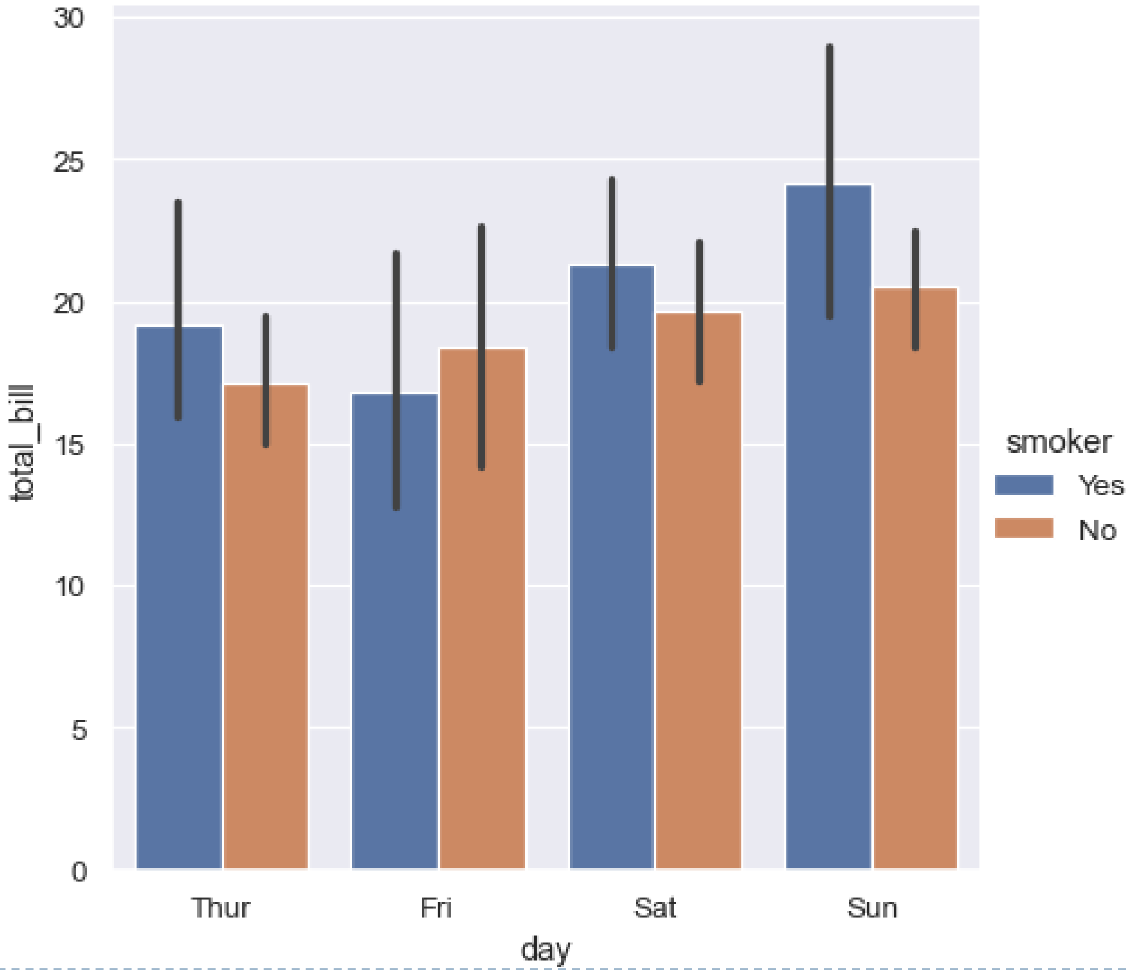
- One of the specialized plot types in seaborn is about visualizing categorical data.
- They can be accessed through catplot().

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



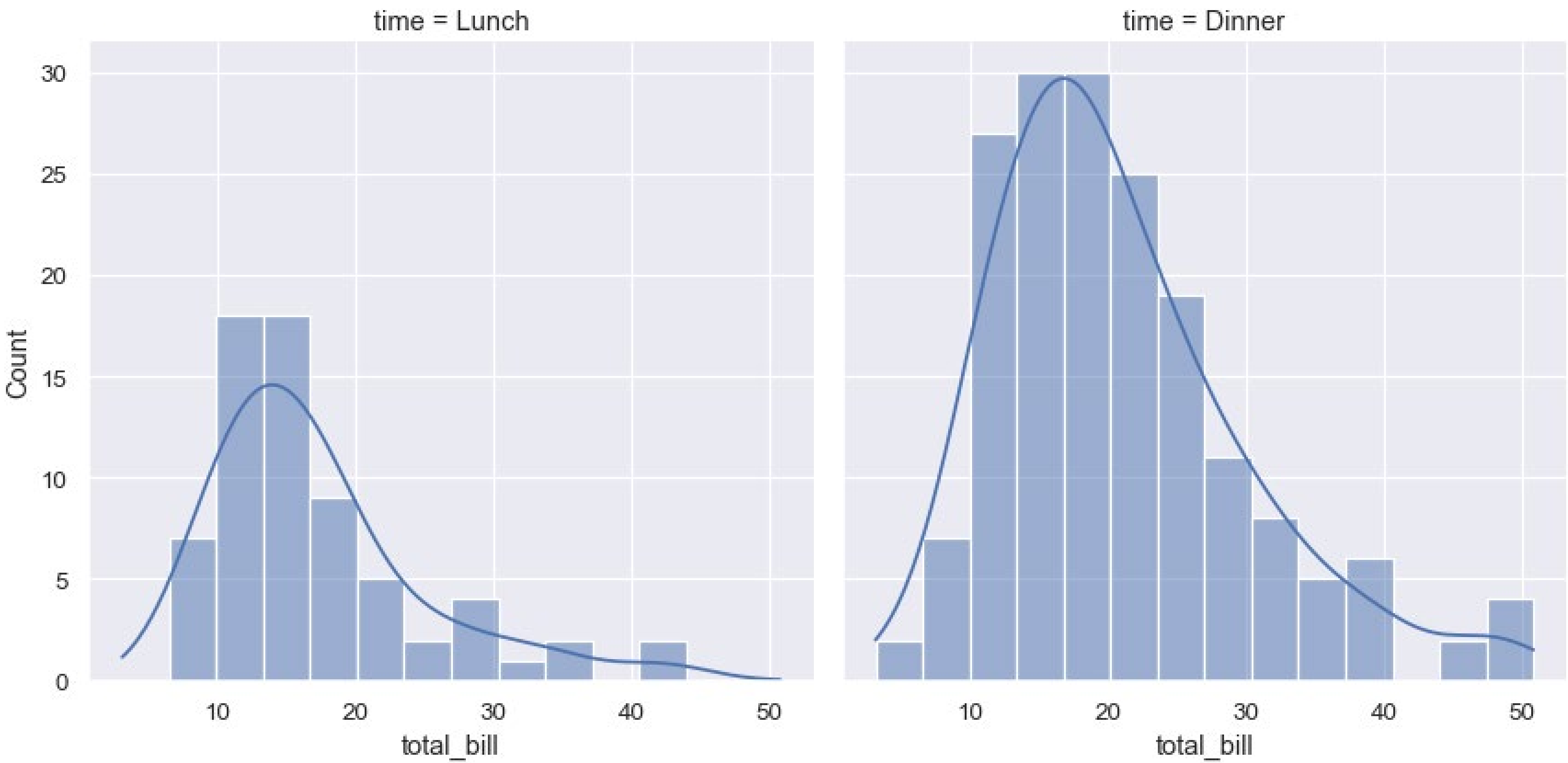
Seaborn – Categorical Data Plots

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



Informative distributional summaries

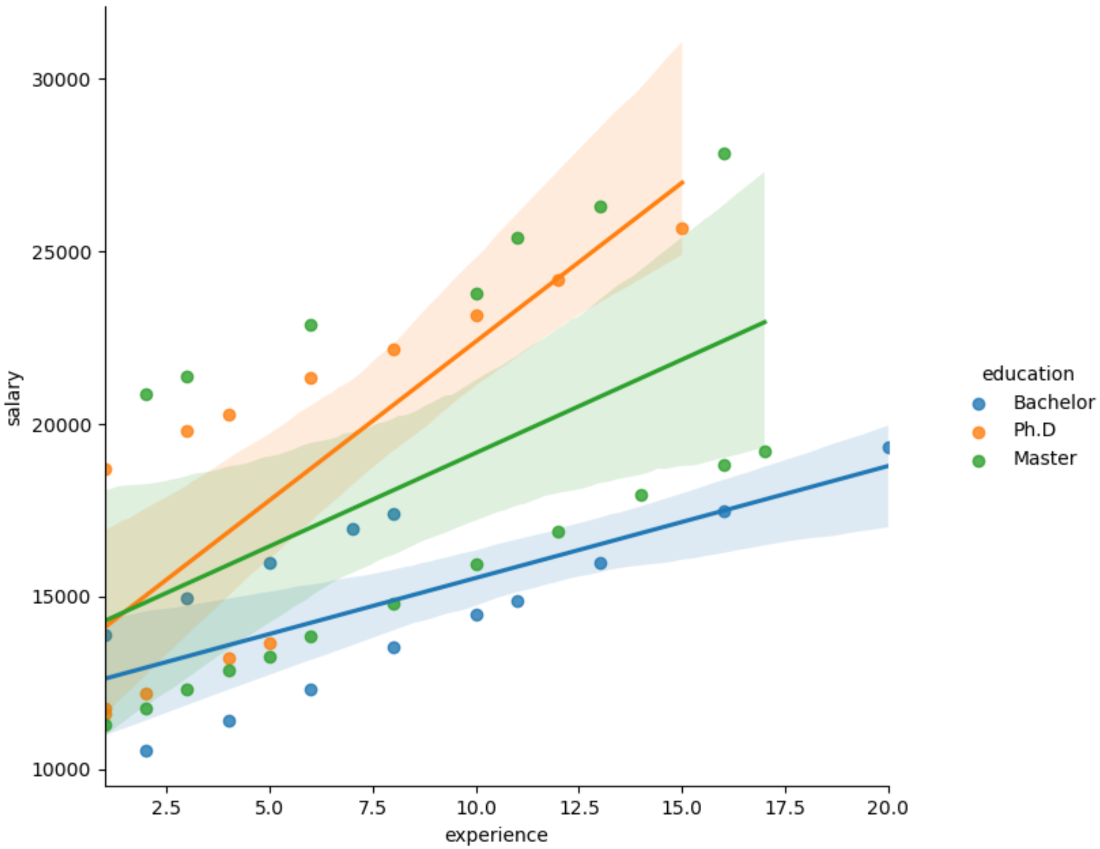
```
sns.displot(data=tips, x="total_bill", col="time", kde=True)
```



Statistical estimation and error bars

- Linear Model (regression)

```
sns.lmplot(x="experience", y="salary", hue="education", data=salary)
```

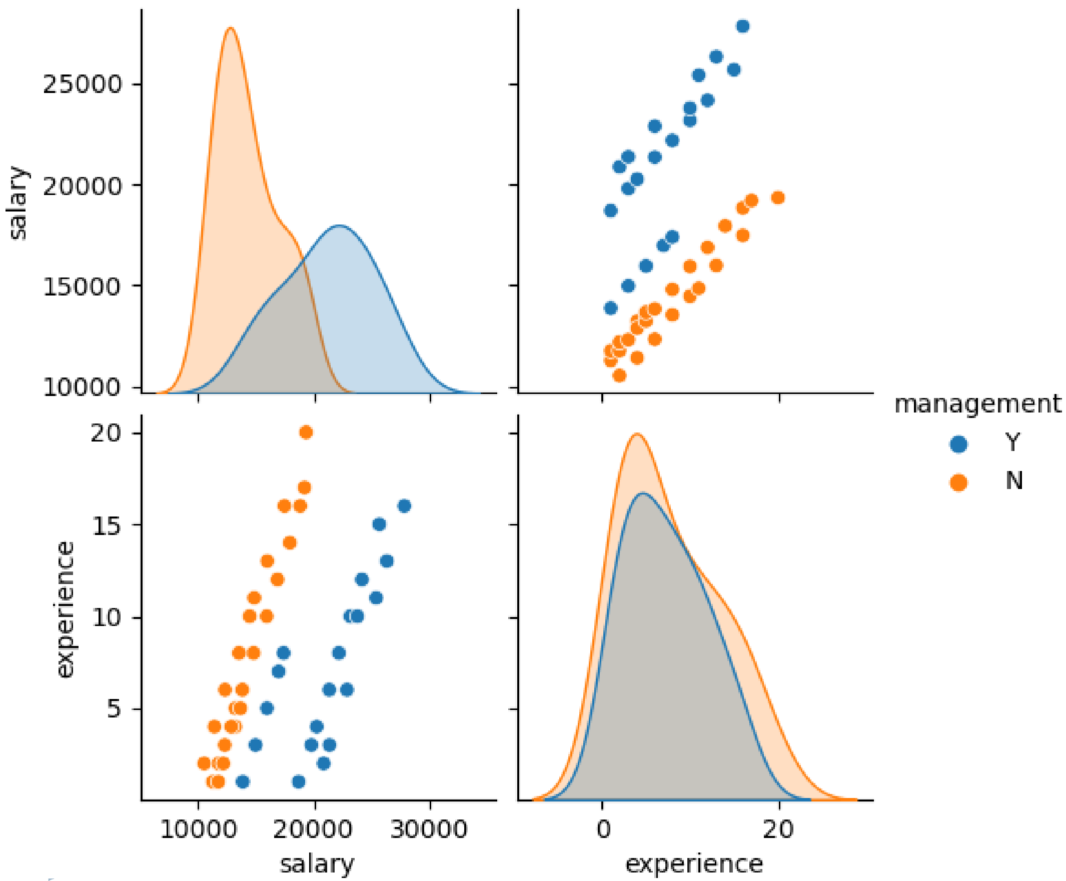



```
salary,experience,education,management
13876,1,Bachelor,Y
11608,1,Ph.D,N
18701,1,Ph.D,Y
11283,1,Master,N
11767,1,Ph.D,N
20872,2,Master,Y
11772,2,Master,N
10535,2,Bachelor,N
12195,2,Ph.D,N
12313,3,Master,N
14975,3,Bachelor,Y
21371,3,Master,Y
19800,3,Ph.D,Y
11417,4,Bachelor,N
20263,4,Ph.D,Y
13231,4,Ph.D,N
12884,4,Master,N
13245,5,Master,N
```

Pair Plots

- Pairwise distribution of a parameter w.r.t. other parameters

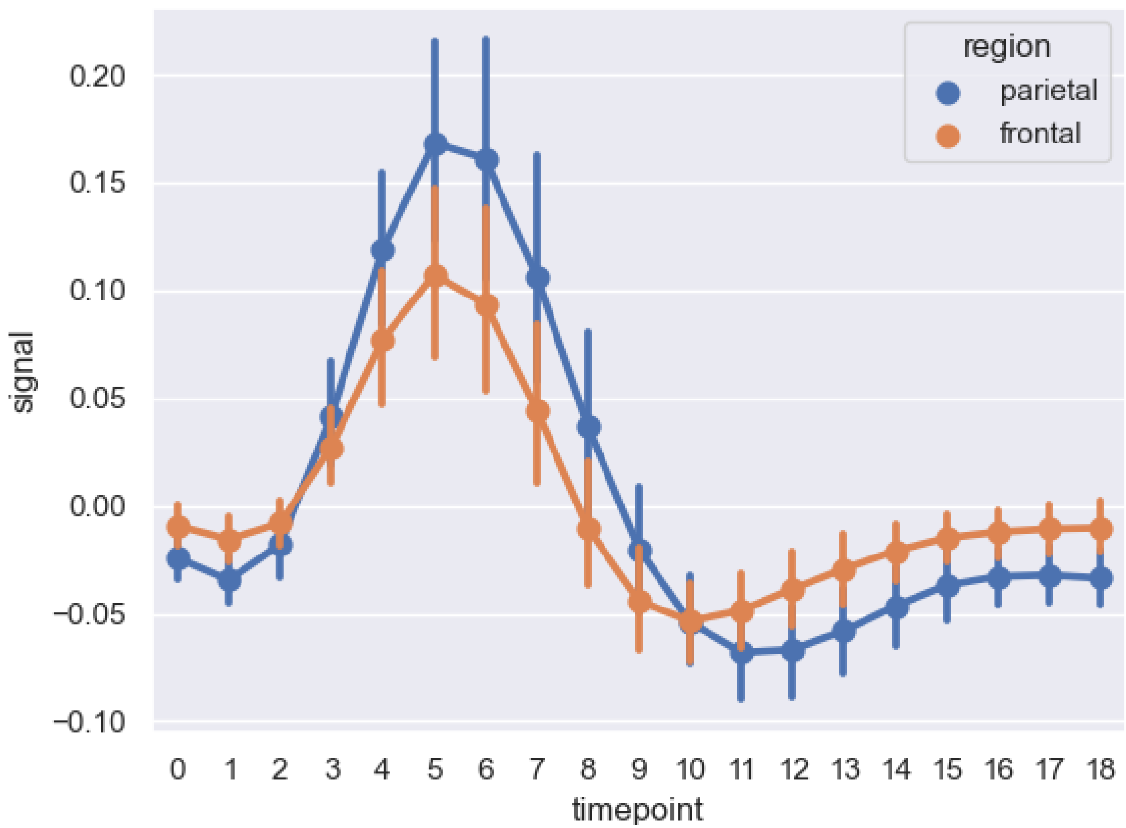
```
sns.pairplot(salary, hue="management")
```




```
salary,experience,education,management
13876,1,Bachelor,Y
11608,1,Ph.D,N
18701,1,Ph.D,Y
11283,1,Master,N
11767,1,Ph.D,N
20872,2,Master,Y
11772,2,Master,N
10535,2,Bachelor,N
12195,2,Ph.D,N
12313,3,Master,N
14975,3,Bachelor,Y
21371,3,Master,Y
19800,3,Ph.D,Y
11417,4,Bachelor,N
20263,4,Ph.D,Y
13231,4,Ph.D,N
12884,4,Master,N
13245,5,Master,N
```

Time Series

```
import seaborn as sns
sns.set(style="darkgrid")
fmri = sns.load_dataset("fmri")
sns.pointplot(x="timepoint", y="signal", hue="region", style="event", data=fmri)
```



subject	timepoint	event	region	signal
s13	18	stim	parietal	-0.017551581538
s5	14	stim	parietal	-0.0808829319505
s12	18	stim	parietal	-0.0810330187333
s11	18	stim	parietal	-0.04613439017519999
s10	18	stim	parietal	-0.0379702032642
s9	18	stim	parietal	-0.10351309616
s8	18	stim	parietal	-0.0644081947232
s7	18	stim	parietal	-0.0605262017124
s6	18	stim	parietal	-0.00702856091007
s5	18	stim	parietal	-0.0405568546157
s4	18	stim	parietal	-0.048812199946599986
s3	18	stim	parietal	-0.0471481458275
s2	18	stim	parietal	-0.08662295949179999
s1	18	stim	parietal	-0.0466590461638
s0	18	stim	parietal	-0.0755699759477
s13	17	stim	parietal	-0.00826462526111
s12	17	stim	parietal	-0.08851175012250001
s7	9	stim	parietal	0.058896545297
s10	17	stim	parietal	-0.016846516627
s9	17	stim	parietal	-0.12157375579000003
s8	17	stim	parietal	-0.0762871207962

References

1 https://seaborn.pydata.org/
2 https://duchesnay.github.io/pystatsml/scientific_python/scipy_matplotlib.html
3 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
4 Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/.
5 Millman, K.J., Aivazis, M. (2011). Python for Scientists and Engineers, Computing in Science & Engineering, 13, 9-12.
6 https://duchesnay.github.io/pystatsml/scientific_python
7 Travis E. Oliphant (2007). Python for Scientific Computing, Computing in Science & Engineering, 9, 10-20.
8 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
9 http://www.diveintopython.net/
10 https://docs.python.org/3/tutorial/
11 http://www.python-course.eu
12 https://developers.google.com/edu/python/
13 http://learnpythonthehardway.org/book/
14 https://medium.com/codex/matplotlib-pyplot-or-the-object-interface-find-out-which-is-best-for-your-projects-b9402269bf1e
15 https://medium.com/towards-data-science/demystifying-matplotlib-3895ab229a63