

Rushabh Prajapati  
Dhruti Patel



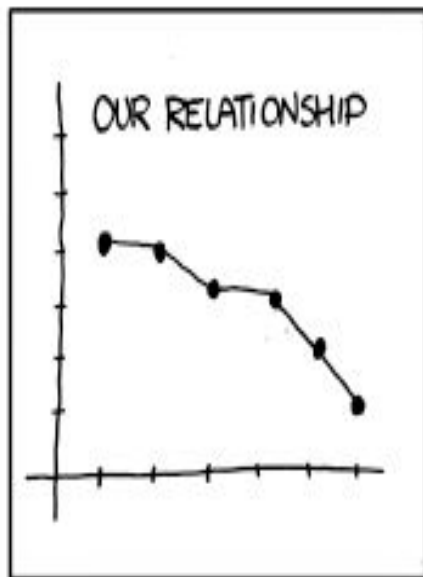
# Matplotlib

## Pyplot & Plotting



I THINK WE SHOULD  
GIVE IT ANOTHER SHOT.

WE SHOULD BREAK  
UP, AND I CAN  
PROVE IT.



HUH.



MAYBE YOU'RE RIGHT.

I KNEW DATA WOULD CONVINCE YOU.  
NO, I JUST THINK I CAN DO  
BETTER THAN SOMEONE WHO  
DOESN'T LABEL HER AXES.



# What is Data Visualization?

Data visualization is the process of translating large data sets and metrics into charts, graphs and other visuals.

The resulting visual representation of data makes it easier to identify and share real-time trends, outliers, and new insights about the information represented in the data.

Visuals help to easily understand the complex problem.

Data visualization in python is perhaps one of the most utilized features for data science with python in today's day and age.

The libraries in python come with lots of different features that enable users to make highly customized, elegant, and interactive plots



# What is Matplotlib?

Matplotlib is one of the most powerful tools for data visualization in Python.

2-D plotting library that helps in visualizing figures.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code.

- makes heavy use of NumPy (other extension codes)
- can be used in python scripts, lpython notebooks, etc.
- backends for generating vector or pixel graphics, PDFs, postscript etc.
- some of the available diagram types: plots, histograms, bar charts, scatterplots ...

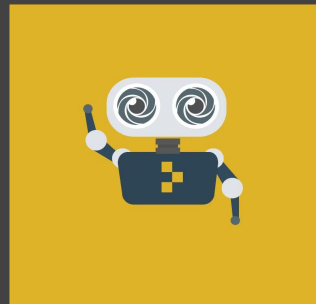
Matplotlib is divided into three main parts:

1. **pylab interface** to create MATLAB like figures, provided by matplotlib.pylab
2. **matplotlib frontend** or *matplotlib API*
3. **matplotlib backends** are the renderers for different kinds of formats, device-dependent drawing devices

# How to install Matplotlib?

## [Installation Guide](#)

- Installation Guide for installing matplotlib and other useful resources
  - PIP
  - IPYTHON SHELL
  - PYTHON



# How to work with Matplotlib?

Before using matplotlib, we need to import the package. This can be done using the **'import'** method.. **PyPlot** is the graphical module in matplotlib which is mostly used for data visualisation, **importing PyPlot is sufficient to work around data visualisation.**

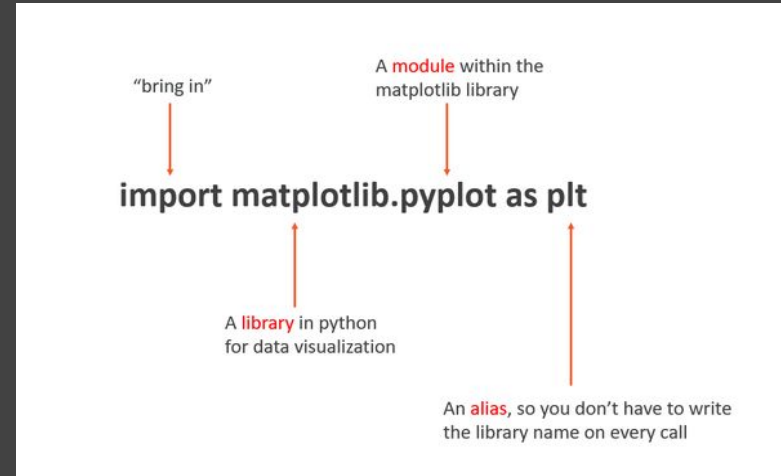
## How?

```
# import matplotlib library as mpl
```

```
import matplotlib as mpl
```

```
#import the pyplot module from matplotlib as plt (short  
name used for referring the object)
```

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



# Matplotlib, PyPlot and Python

**Python** : used for web development, mathematics and statistical analysis.

**Matplotlib package**, which is developed using python is very widely used for data visualisations.

**PyPlot** is a module in matplotlib which provides MATLAB like interface. MATLAB is heavily used for statistical analysis in the manufacturing industry. MATLAB is a licensed software and requires a significant amount of money to buy and use, whereas PyPlot is an open-source module and gives similar functionality as MATLAB using python. Just to conclude PyPlot has been seen as a replacement of MATLAB in the context of open source.

**How can we create high quality scientific graphs using matplotlib and what are some of the alternatives?**





# Black hole Image Using Matplotlib

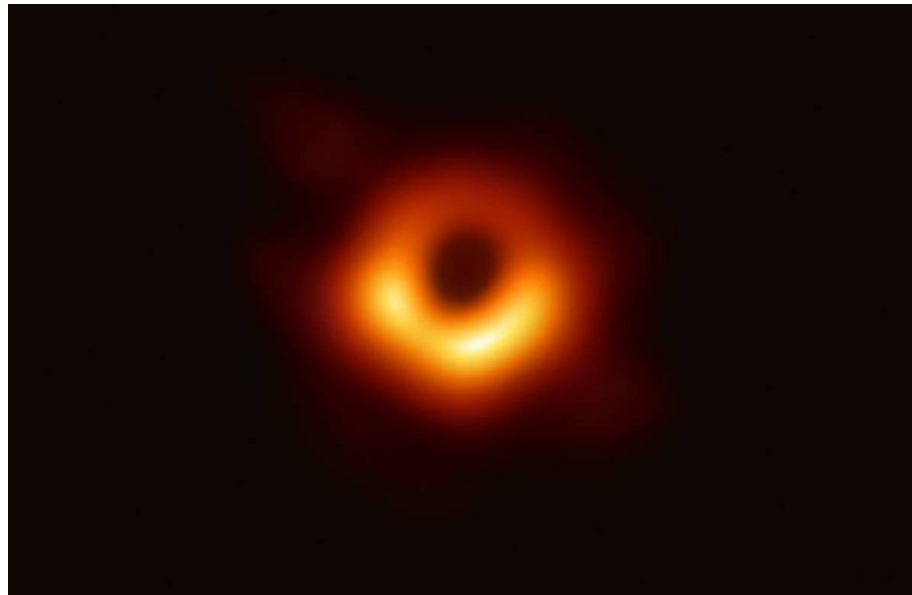


It has been used to display the first ever photograph of a black hole and to illustrate the existence of gravitational waves .

[First M87 Event Horizon Telescope Results. III. Data Processing and Calibration,](#)

Calibration and reduction of Event Horizon Telescope (EHT) 1.3 mm radio wavelength observations of the supermassive black hole candidate at the center of the radio galaxy M87 and the quasar 3C 279, taken during the 2017 April 5–11 observing campaign.

[Source](#)

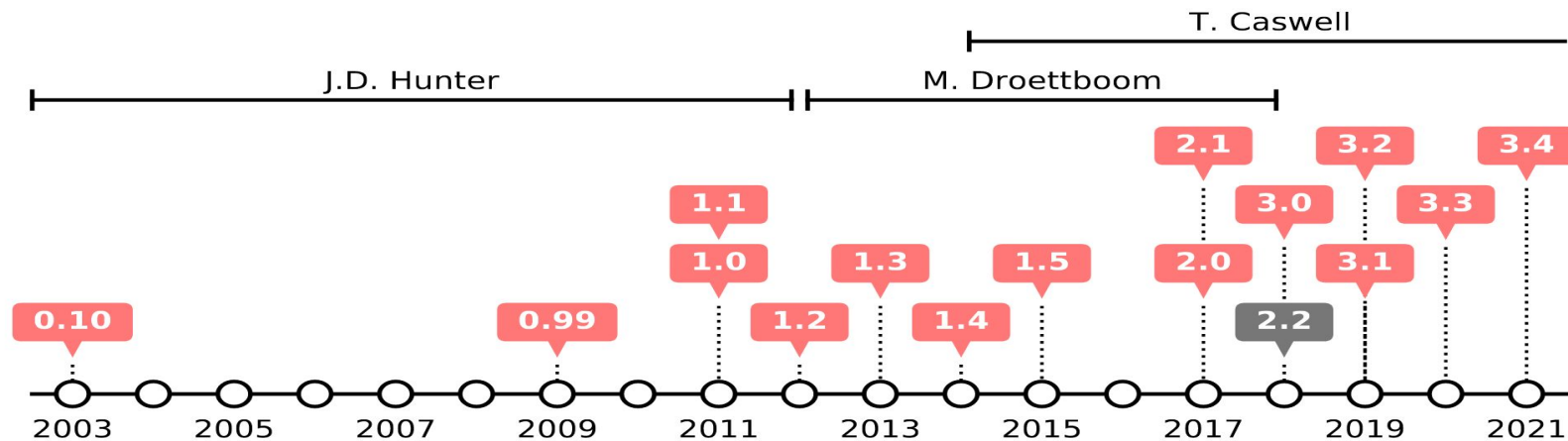




Matplotlib is both a versatile and powerful library that allows you to design very high quality figures, suitable for scientific publishing. It offers both a simple and intuitive interface (pyplot) as well as an object oriented architecture that allows you to tweak anything within a figure.

Note that, it can also be used as a regular graphic library in order to design non-scientific figures, as we'll see throughout this book.

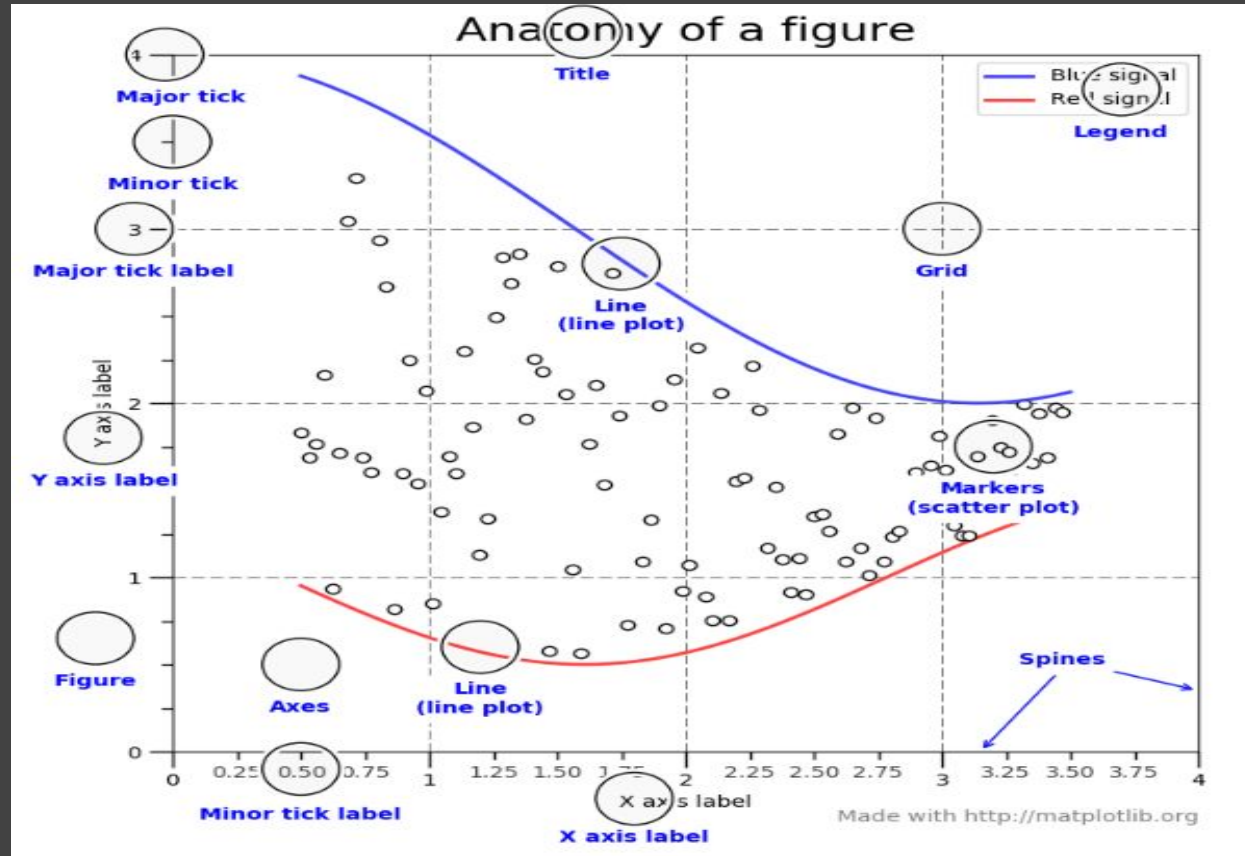
For example, the Matplotlib timeline figure (below) is simply made of a line with markers and some styled annotations.



# FUNDAMENTALS

The figure contains the overall window where plotting happens, contained within the figure are where actual graphs are plotted.

Every Axes has an x-axis and y-axis for plotting. And contained within the axes are titles, ticks, labels associated with each axis.



# Anatomy of a Figure Continued ..

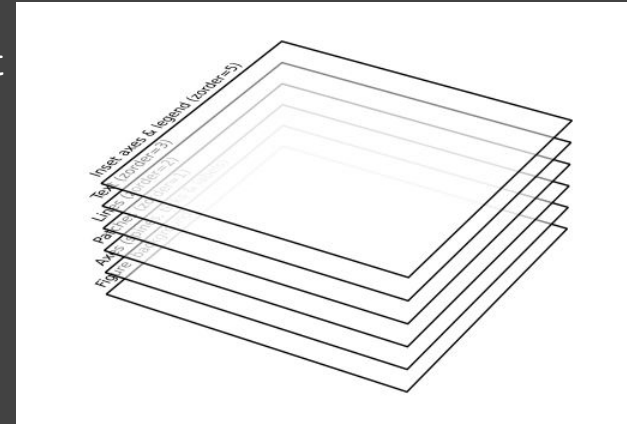


Matplotlib library is built upon Numpy. It is a tree-like hierarchical structure which consists of objects that makeup each of these plots.

A '**Figure**' in Matplotlib can be understood as the outermost storage for a graph. This '**Figure**' can contains multiple '**Axes**' objects. '**Axes**' object is NOT the plural form of '**Axis**' in this case.

'**Axes**' can be understood as a part of '**Figure**', a subplot. It can be used to manipulate every part of the graph inside it. A '**Figure**' object in an Matplotlib is a box that stores one or more '**Axes**' objects.

Under '**Axes**' comes the tick marks, lines, legends, and text boxes in the hierarchy. Every object in the Matplotlib can be manipulated



# Mastering the Defaults and Customization

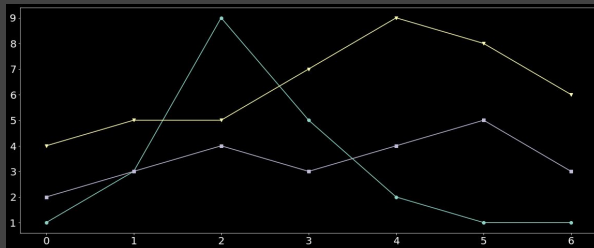
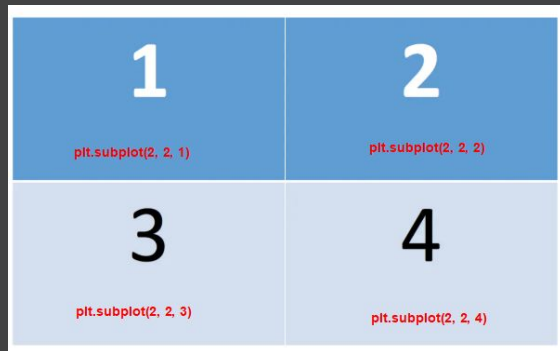
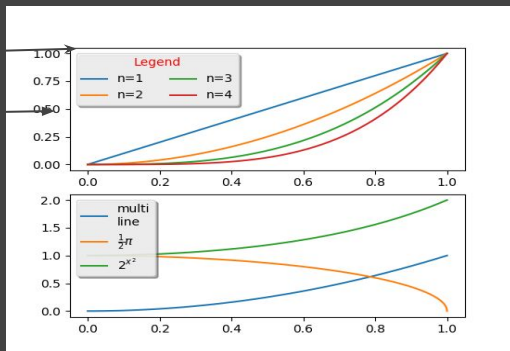
Customizing plot legends

Custom Subplots

Stylesheets

Advanced Plotting

Pylustrator





# To create Better quality graphs

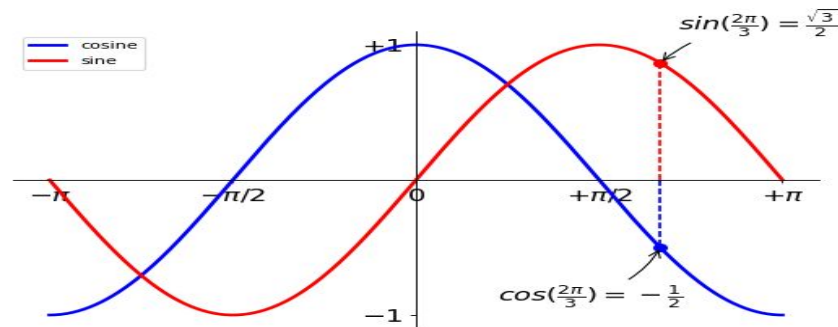
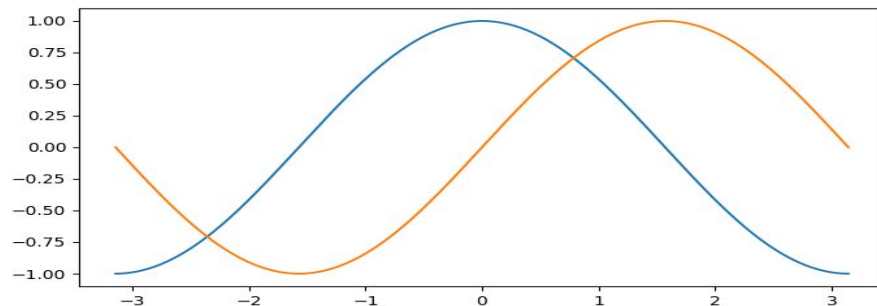
## Do Not Trust the Defaults

Defaults are good enough for any plot but they are best for none.

All plots require at least some manual tuning of the different settings to better express the message, be it for making a precise plot more salient to a broad audience, or to choose the best colormap for the nature of the data.

Upper Figure: Default Settings

Lower Figure: Manual Tweaks



# Customizing plot legends



```
import matplotlib.pyplot as plt
plt.style.use('classic')
import numpy as np
x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
```

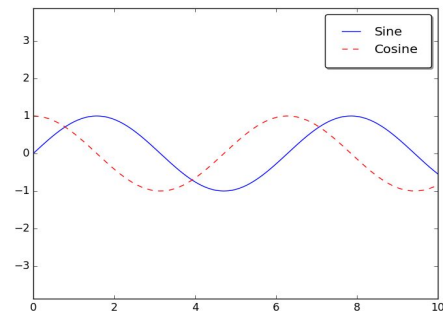
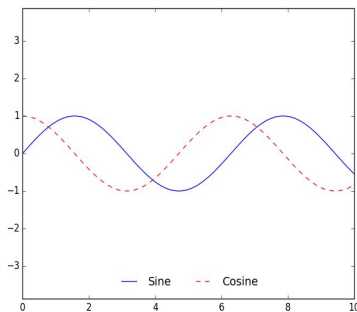
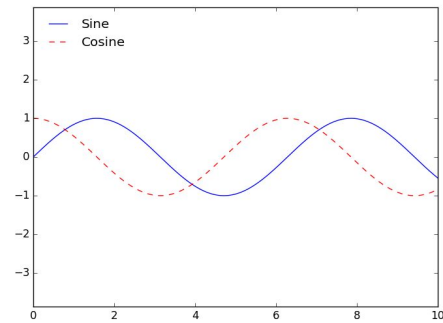
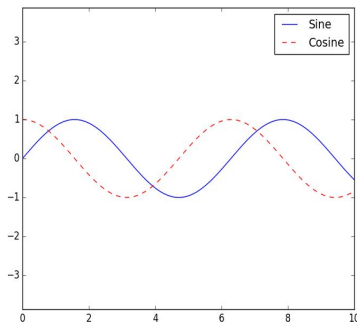
```
# # Default Position
ax.plot(x, np.sin(x), '-b-', label='Sine')
ax.plot(x, np.cos(x), '--r-', label='Cosine')
ax.axis('equal')
leg = ax.legend()
```

```
#----- Upper Left Legend-----
ax.legend(loc='upper left', frameon=False)
```

```
#-----
# We can use the ncol command to specify the number of columns in the legend:
ax.legend(frameon=False, loc='lower center', ncol=2)
```

```
# # We can use a rounded box (fancybox) or add a shadow, change the transparency
# # (alpha value) of the frame, or change the padding around the text
ax.legend(fancybox=True, framealpha=1, shadow=True, borderpad=1)
```

```
plt.show()
```





# Multiple Plot Legends

```
# Sometimes when designing a plot you'd like to add multiple legends
# to the same axes. Unfortunately, Matplotlib does not make this easy:
# via the standard legend interface, it is only possible to create a
# single legend for the entire plot. If you try to create a second
# legend using plt.legend() or ax.legend(), it will simply override
# the first one. We can work around this by creating a new legend artist
# from scratch, and then using the lower-level ax.add_artist() method to
# manually add the second artist to the plot:
```

```
import matplotlib.pyplot as plt
plt.style.use('classic')
import numpy as np
fig, ax = plt.subplots()
```

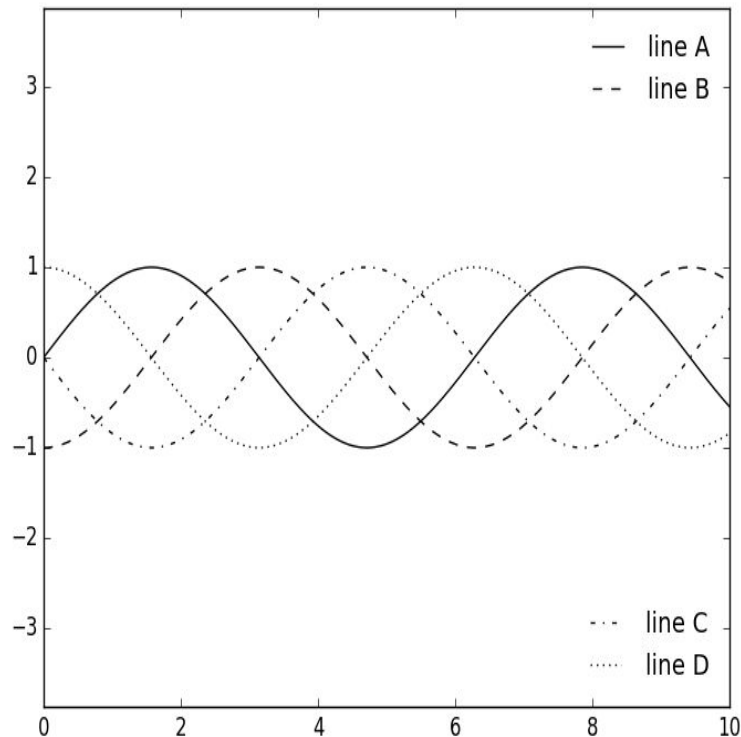
```
lines = []
styles = ['-', '--', '-.', ':']
x = np.linspace(0, 10, 1000)
```

```
for i in range(4):
    lines += ax.plot(x, np.sin(x - i * np.pi / 2),
                    styles[i], color='black')
ax.axis('equal')
```

```
# specify the lines and labels of the first legend
ax.legend(lines[:2], ['line A', 'line B'],
         loc='upper right', frameon=False)
```

```
# Create the second legend and add the artist manually.
from matplotlib.legend import Legend
leg = Legend(ax, lines[2:], ['line C', 'line D'],
            loc='lower right', frameon=False)
ax.add_artist(leg)
```

```
plt.show()
```







# Customizing Subplots

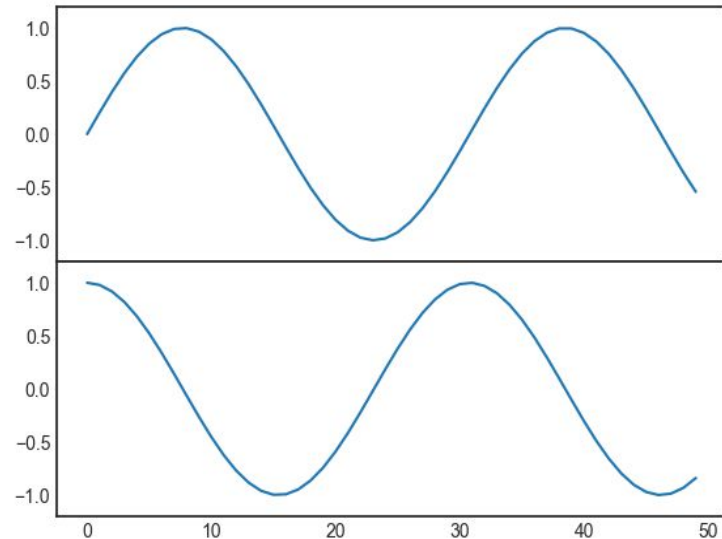
```
fig = plt.figure()

# These numbers represent [left, bottom, width, height]
# in the figure coordinate system, which ranges from 0.1
# at the bottom left of the figure to 0.5 at the top right
# of the figure.

ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4],
                   xticklabels=[], ylim=(-1.2, 1.2))
ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4],
                   ylim=(-1.2, 1.2))

x = np.linspace(0, 10)
ax1.plot(np.sin(x))
ax2.plot(np.cos(x))


plt.show()
```



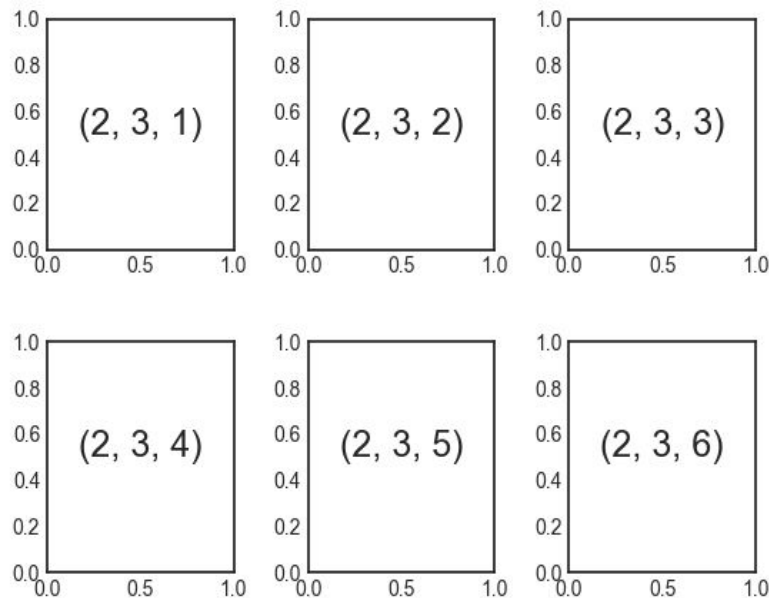
Vertically stacked axes example



# Suplots Continued...

Code > Data Science > Custom Plots >  grid-subplot.py > ...

```
1  # Aligned columns or rows of subplots are a common-enoug
2  # need that Matplotlib has several convenience routines
3  # that make them easy to create. The lowest level of these
4  # is plt.subplot(), which creates a single subplot within
5  # grid. As you can see, this command takes three integer
6  # arguments—the number of rows, the number of columns, and
7  # the index of the plot to be created in this scheme, which
8  # runs from the upper left to the bottom right:
9  |
10 import matplotlib.pyplot as plt
11 plt.style.use('seaborn-white')
12 import numpy as np
13
14 for i in range(1, 7):
15     plt.subplot(2, 3, i)
16     plt.text(0.5, 0.5, str((2, 3, i)),
17             fontsize=18, ha='center')
18
19
20 # Adjusted Spacing
21 fig = plt.figure()
22 fig.subplots_adjust(hspace=0.4, wspace=0.4)
23 for i in range(1, 7):
24     ax = fig.add_subplot(2, 3, i)
25     ax.text(0.5, 0.5, str((2, 3, i)),
26           fontsize=18, ha='center')
27
28 plt.show()
```



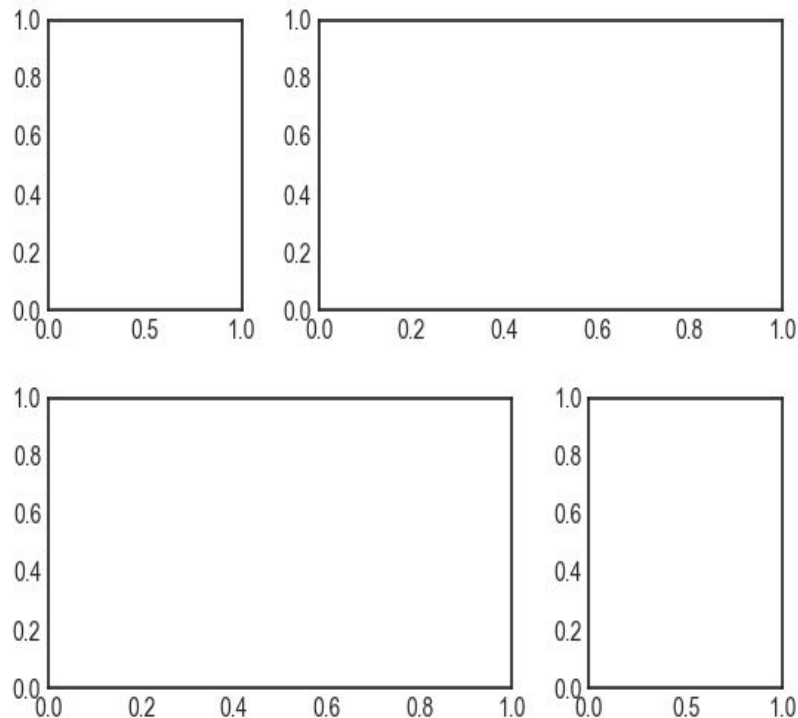
plt.subplot: Simple Grids of Subplots



## plt.GridSpec: More Complicated Arrangements

Code > Data Science > Custom Plots > complex-subplots.py > ...

```
1 import matplotlib.pyplot as plt
2 plt.style.use('seaborn-white')
3 import numpy as np
4
5 # To go beyond a regular grid to subplots that span multiple
6 # rows and columns, plt.GridSpec() is the best tool.
7 # The plt.GridSpec() object does not create a plot by
8 # itself; it is simply a convenient interface that is
9 # recognized by the plt.subplot() command. For example,
10 # a gridspec for a grid of two rows and three columns with some
11 # specified width and height space looks like this:
12
13
14 grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
15
16 # From this we can specify subplot locations and extents using
17 # the familiar Python slicing syntax:
18 plt.subplot(grid[0, 0])
19 plt.subplot(grid[0, 1:])
20 plt.subplot(grid[1, :2])
21 plt.subplot(grid[1, 2])
22
23 plt.show()
```

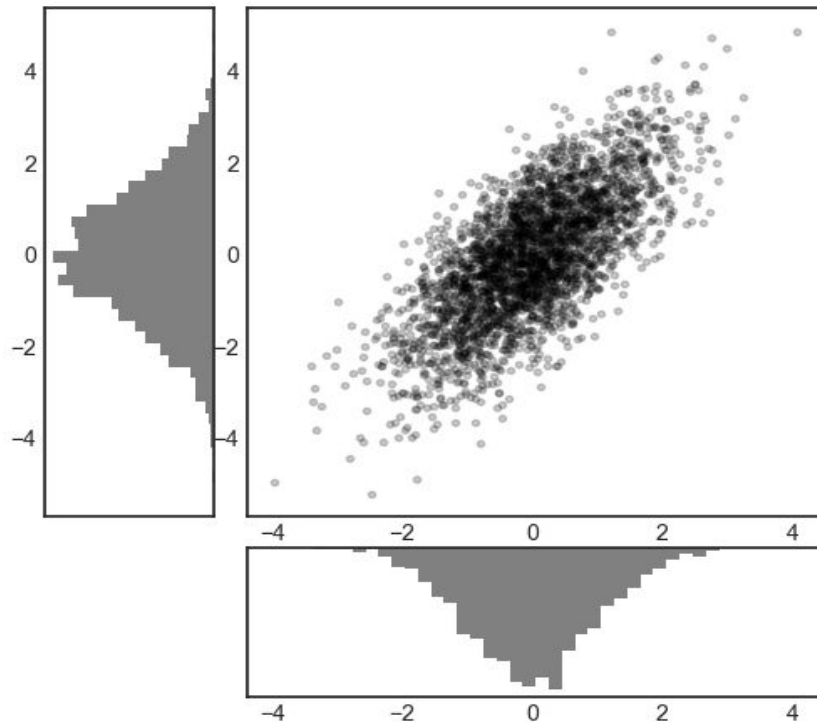




# Visualizing multidimensional distributions with plt.GridSpec

Code > Data Science > Custom Plots > complex-data-viz-subplots.py > ...

```
1 import matplotlib.pyplot as plt
2 plt.style.use('seaborn-white')
3 import numpy as np
4
5 # Create some normally distributed data
6 mean = [0, 0]
7 cov = [[1, 1], [1, 2]]
8 x, y = np.random.multivariate_normal(mean, cov, 3000).T
9
10 # Set up the axes with gridspec
11 fig = plt.figure(figsize=(6, 6))
12 grid = plt.GridSpec(4, 4, hspace=0.2, wspace=0.2)
13 main_ax = fig.add_subplot(grid[:-1, 1:])
14 y_hist = fig.add_subplot(grid[:-1, 0], xticklabels=[], sharey=main_ax)
15 x_hist = fig.add_subplot(grid[-1, 1:], yticklabels=[], sharex=main_ax)
16
17 # scatter points on the main axes
18 main_ax.plot(x, y, 'ok', markersize=3, alpha=0.2)
19
20 # histogram on the attached axes
21 x_hist.hist(x, 40, histtype='stepfilled',
22            orientation='vertical', color='gray')
23 x_hist.invert_yaxis()
24
25 y_hist.hist(y, 40, histtype='stepfilled',
26            orientation='horizontal', color='gray')
27 y_hist.invert_xaxis()
28
29 plt.show()
```



# Stylesheets

The version 1.4 release of Matplotlib in August 2014 added a very convenient style module, which includes a number of new default stylesheets, as well as the ability to create and package your own styles. These stylesheets are formatted similarly to the **.matplotlibrc** files mentioned earlier, but must be named with a **.mplstyle** extension.

Even if you don't create your own style, the stylesheets included by default are extremely useful. The available styles are listed in **plt.style.available**

Code > Style Sheets > sheets.py

```
1 import matplotlib.pyplot as plt
2 plt.style.use('classic')
3 import numpy as np
4
5 print(plt.style.available[:5])
6
```

PROBLEMS

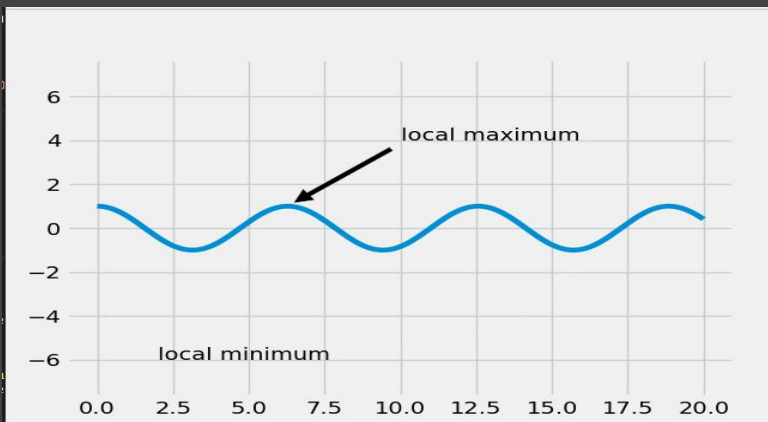
OUTPUT

DEBUG CONSOLE

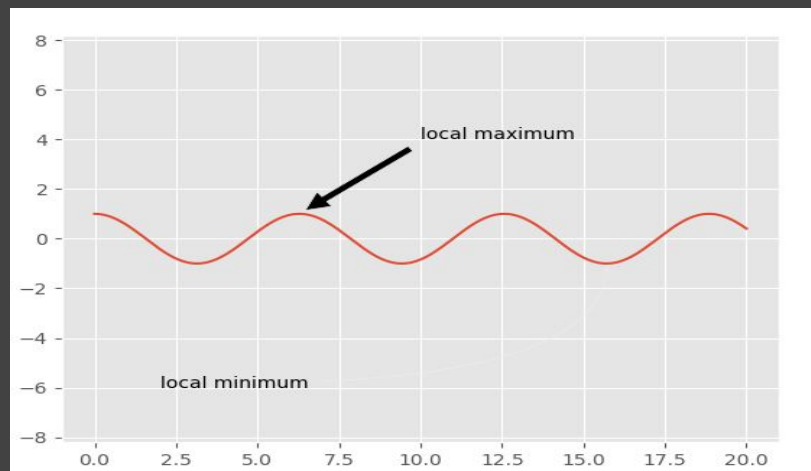
TERMINAL

```
rusve@Synapse MINGW64 /r/MacEwan/CMPT/CMPT340/Project/matplotlibTutorial/Code/Data Science
$ C:/Users/rusve/AppData/Local/Programs/Python/Python39/python.exe "r:/MacEwan/CMPT/CMPT340
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background']
```

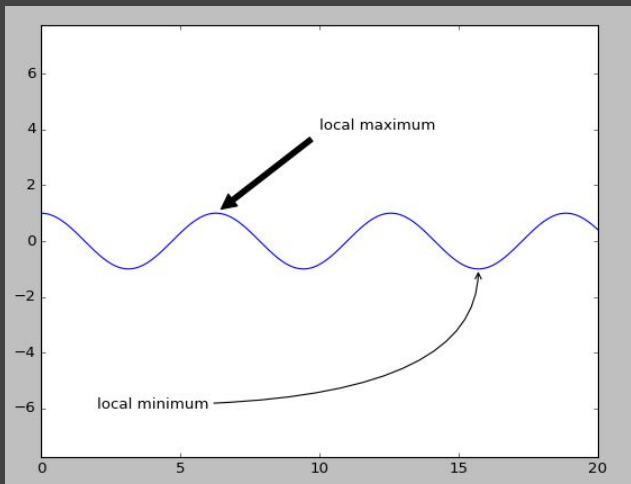
fivethirtyeight



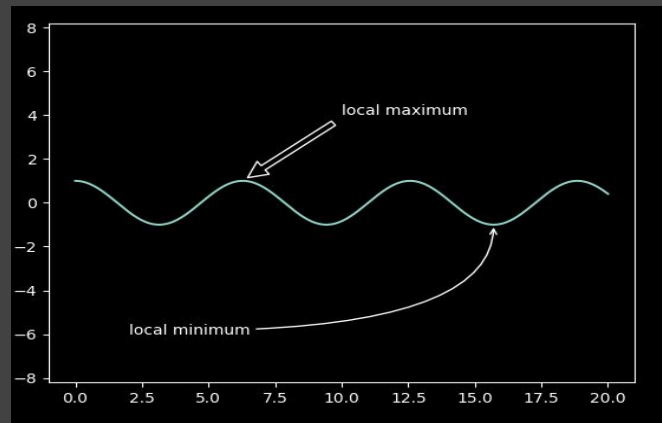
ggplot



Classic



Dark  
Background



With all of these built-in options for various plot styles, Matplotlib becomes much more useful for both interactive visualization and creation of figures for publication.

Each time Matplotlib loads, it defines a runtime configuration (rc) containing the default styles for every plot element you create. This configuration can be adjusted at any time using the `plt.rc` convenience routine.

Matplotlib also has stylesheets inspired by the Seaborn library.



# Advanced Plotting

Animation - [Animation Framework](#)

Going 3D - [Matplotlib 3D - Interface](#)

Graphing Library

Beyond its usage for scientific visualization, matplotlib is also a comprehensive library for creating static, animated, and interactive visualizations in

PYthon as written on the website . Said differently, matplotlib is a graphic

library that can be used for virtually any purpose, even though the perfor-

mance may vary greatly from one application to the other, depending on

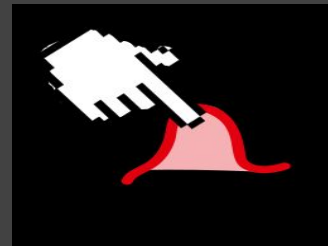
the complexity of the rendering.

Architecture & optimization

a deep understanding of the matplotlib architecture is not necessary to use it, it is nonetheless useful to know a bit of its architecture to optimize either speed, memory and even rendering.



# Pylustrator



Pylustrator is a software to prepare your figures for publication in a reproducible way. This means you receive a figure representing your data and alongside a generated code file that can exactly reproduce the figure as you put them in the publication, without the need to readjust things in external programs.

## Installation

Just get pylustrator over the pip installation:

**pip install pylustrator**

The package depends on:

numpy, matplotlib, PyQt5, qtpy, QtAwesome, scikit-image, natsort



## **Saving**

To save the figure press `ctrl+s` or select `File->Save`. This will generate code that corresponds to the changes you made to the figure and past it into your script file or your jupyter notebook cell. The code will be pasted directly over the `plt.show()` command that started the editor or, if there already is a generated code block for this figure, it will replace the existing code block.

## **Color editor**

Pylustrator comes with a powerful color editor which allows to test different color configurations for your figure easily

## **Tick Editor**

To edit the ticks of an axes click on the tick icon. There, a windows opens that allows to set major and minor ticks every line in the edit window corresponds to one tick.



# DEMO

# Get the right tool

There exist many tools that can make your life easier when creating figures, and knowing a few of them can save you a lot of time.

Whether drawing a graph, designing a schema of your experiment, or plotting some data, there are open-source tools for you. They're just waiting to be found and used.

- **Matplotlib**
- **R**
- **Inkscape**
- [D3.js](#)
- [Cytoscape](#)
- [Dash Plotly](#)
- [Seaborn](#)
- [Pandas](#)

# Question/Comments?

If you have any questions or comments please let us know!

Check out the github repository and try some of the code and Visualizations.

If you find any issues please report them.

Thank you for your time!

# References

Gerum, R., (2020). **pylustrator: code generation for reproducible figures for publication**. Journal of Open Source Software, 5(51), 1989. [doi:10.21105/joss.01989](https://doi.org/10.21105/joss.01989)

<https://github.com/matplotlib/AnatomyOfMatplotlib>

<https://matplotlib.org/>