

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

[Log in](#)[Create Account](#)

**Karlijn Willems**  
February 22nd, 2017

PYTHON +1

# Matplotlib Tutorial: Python Plotting

This Matplotlib tutorial takes you through the basics Python data visualization: the anatomy of a plot, pyplot and pylab, and much more

Humans are very visual creatures: we understand things better when we see things visualized. However, the step to presenting analyses, results or insights can be a bottleneck: you might not even know where to start or you might have already a right format in mind, but then questions like “Is this the right way to visualize the insights that I want to bring to my audience?” will have definitely come across your mind.

When you're working with the Python plotting library Matplotlib, the first step to answering the above questions is by building up knowledge on topics like:

- The [anatomy of a Matplotlib plot](#): what is a subplot? What are the Axes? What exactly is a figure?
- [Plot creation](#), which could raise questions about what module you exactly need to import (pylab or pyplot?), how you exactly should go about initializing the figure and the Axes of your plot, how to use matplotlib in Jupyter notebooks, etc.

• [Plotting routines](#), from simple ways to plot your data to more advanced ways of visualizing

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

example, pdf files, clear the axes, clear the figure or close the plot, etc.

- Lastly, you'll briefly cover two ways in which you can [customize Matplotlib](#): with style sheets and the rc settings.

(To practice matplotlib interactively, try the free Matplotlib chapter at the start of this [Intermediate Python course](#) or see DataCamp's [Viewing 3D Volumetric Data With Matplotlib](#) tutorial to learn how to work with matplotlib's event handler API.)

## What Does A Matplotlib Python Plot Look Like?

At first sight, it will seem that there are quite some components to consider when you start plotting with this Python data visualization library. You'll probably agree with me that it's confusing and sometimes even discouraging seeing the amount of code that is necessary for some plots, not knowing where to start yourself and which components you should use.

Luckily, this library is very flexible and has a lot of handy, built-in defaults that will help you out tremendously. As such, you don't need much to get started: you need to make the necessary imports, prepare some data, and you can start plotting with the help of the `plot()` function! When you're ready, don't forget to show your plot using the `show()` function.

Look at this example to see how easy it really is:

script.py

```
1 # Import the necessary packages and module
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # Prepare the data
6 x = np.linspace(0, 10, 100)
```

IPython Shell

In [1]: |

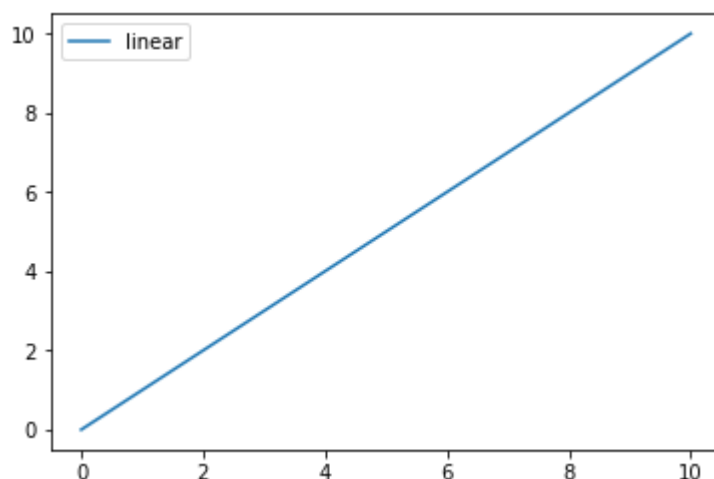
[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

**Run**

**Note** that you import the `pyplot` module of the `matplotlib` library under the alias `plt`.

Congrats, you have now successfully created your first plot! Now let's take a look at the resulting plot in a little bit more detail:



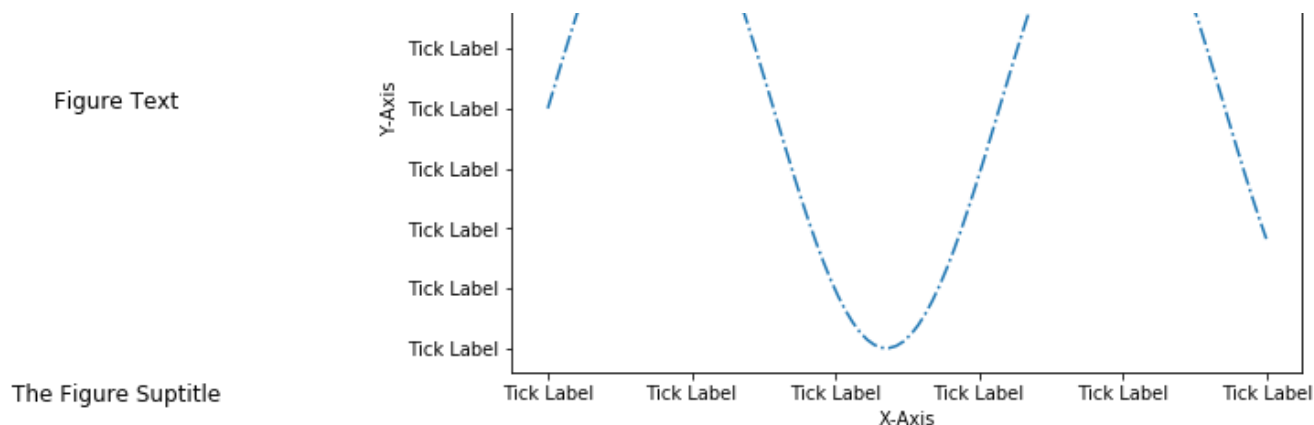
Great, isn't it?

What you can't see on the surface is that you have -maybe unconsciously- made use of the built-in defaults that take care of the creation of the underlying components, such as the Figure and the Axes. You'll read more about these defaults in the section that deals with the differences between `pylab` and `pyplot`.

For now, you'll understand that working with `matplotlib` will already become a lot easier when you understand how the underlying components are instantiated. Or, in other words,

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs



In essence, there are two big components that you need to take into account:

- The **Figure** is the overall window or page that everything is drawn on. It's the top-level component of all the ones that you will consider in the following points. You can create multiple independent Figures. A Figure can have several other things in it, such as a subtitle, which is a centered title to the figure. You'll also find that you can add a legend and color bar, for example, to your Figure.
- To the figure you add **Axes**. The Axes is the area on which the data is plotted with functions such as `plot()` and `scatter()` and that can have ticks, labels, etc. associated with it. This explains why Figures can contain multiple Axes.

**Tip:** when you see, for example, `plt.xlim`, you'll call `ax.set_xlim()` behind the covers. All methods of an Axes object exist as a function in the `pyplot` module and vice versa. Note that mostly, you'll use the functions of the `pyplot` module because they're much cleaner, at least for simple plots!

You'll see what "clean" means when you take a look at the following pieces of code. Compare, for example, this piece of code:

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

```
plt.show()
```

Run

With the piece of code below:

<pre>script.py 1 import matplotlib.pyplot as plt 2 plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='blue', linewidth=3) 3 plt.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 12, 13], color='darkgreen', marker='^') 4 plt.xlim(0.5, 4.5) 5 plt.show()</pre>	<pre>IPython Shell In [1]:</pre>
Run	

The second code chunk is definitely cleaner, isn't it?

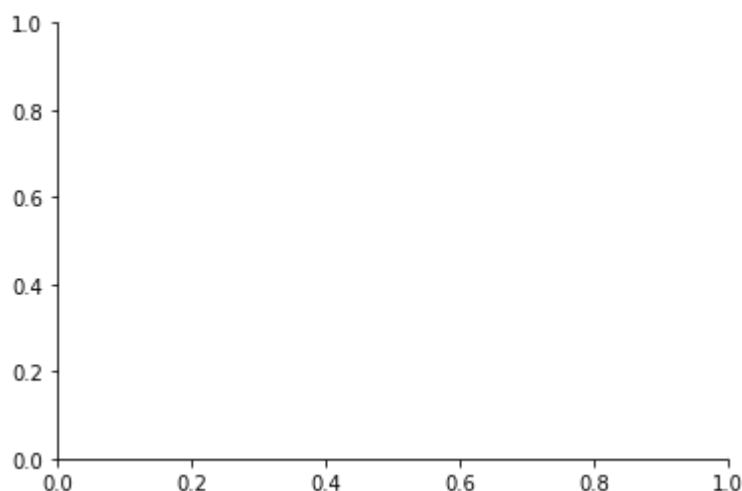
**Note** that the above code examples come from the [Anatomy of Matplotlib Tutorial](#) by Benjamin Root.

However, if you have multiple axes, it's still better to make use of the first code chunk because it's always better to prefer explicit above implicit code! In such cases, you want to make use of the `Figure` object, not the `plt` object.

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

- Spines are lines that connect the axis tick marks and that designate the boundaries of the data area. In other words, they are the simple black square that you get to see when you don't plot any data at all but when you have initialized the Axes, like in the picture below:



You see that the right and top spines are set to invisible.

**Note** that you'll sometimes also read about Artist objects, which are virtually all objects that the package has to offer to users like yourself. Everything drawn using Matplotlib is part of the Artist module. The containers that you will use to plot your data, such as Axis, Axes and Figure, and other graphical objects such as text, patches, etc. are types of Artists.

For those who have already got some coding experience, it might be good to check out and study the code examples that you find in the [Matplotlib gallery](#).

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs



**R, Python & Data Science Online!**

**Get Started for Free**

## Matplotlib, pyplot and pylab: how are they related?

First off, you'll already know Matplotlib by now. When you talk about "Matplotlib", you talk about the whole Python data visualization package. This should not come to you as a big surprise :)

Secondly, `pyplot` is a module in the `matplotlib` package. That's why you often see `matplotlib.pyplot` in code. The module provides an interface that allows you to implicitly and automatically create figures and axes to achieve the desired plot.

This is especially handy when you want to quickly plot something without instantiating any Figures or Axes, as you saw in the example in the first section of this tutorial. You see, you haven't explicitly specified these components, yet you manage to output a plot that you have even customized! The defaults are initialized and any customizations that you do, will be done with the current Figure and Axes in mind.

Lastly, `pylab` is another module, but it gets installed alongside the `matplotlib` package. It bulk imports `pyplot` and the `numpy` library and was generally recommended when you were working with arrays, doing mathematics interactively and wanted access to plotting features.

You might still see this popping up in older tutorials and examples of `matplotlib`, but its use is no longer recommended, especially not when you're using the IPython kernel in your

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

1. ✓

**Note** that also when you're not working in a Jupyter notebook, you'll still need to choose a different backend, depending on your use case. In other words, if you don't want to embed plots inside a notebook, but you rather want to embed them into graphical user interfaces, in batch scripts or web application servers, etc., you will need to specify the backend that you want to use. However, this topic is outside the scope of this tutorial; Instead, the tutorial assumes that you will be using Matplotlib to save your images to your local file system.

## Data For Matplotlib Plots

As you have read in one of the previous sections, Matplotlib is often used to visualize analyses or calculations. That's why the first step that you have to take in order to start plotting in Python yourself is to consider revising NumPy, the Python library for scientific computing.

Scientific computing might not really seem of much interest, but when you're doing data science you'll find yourself working a lot with data that is stored in arrays. You'll need to perform operations on them, inspect your arrays and manipulate them so that you're working with the (subset of the) data that is interesting for your analysis and that is in the right format, etc.


In short, you'll find NumPy extremely handy when you're working with this data visualization library. If you're interested in taking a NumPy tutorial to start well-prepared, go and take DataCamp's [tutorial](#) and make sure to have your copy of our [NumPy cheat sheet](#) close!

Of course, arrays are not the only thing that you pass to your plotting functions; There's also the possibility to, for example, pass Python lists. If you would like to know more about Python lists, consider checking out our [Python list tutorial](#) or the free [Intro to Python for Data Science](#) course.

[Want to leave a comment?](#)



Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

<pre>script.py 1  # Import `pyplot` 2  import matplotlib.pyplot as plt 3 4  # Initialize a Figure 5  fig = plt.figure() 6 7  # Add Axes to the Figure 8  fig.add_axes([0,0,1,1])</pre>	<pre>IPython Shell In [1]:  </pre>
<div>Run </div>	

Easy, isn't it?

## What Is A Subplot?

You have seen all components of a plot and you have initialized your first figure and Axes, but to make things a bit more complicated, you'll sometimes see subplots pop up in code.

Now, don't get discouraged just yet!

You use subplots to set up and place your Axes on a regular grid. So that means that in most cases, Axes and subplot are synonymous, they will designate the same thing. When you do call subplot to add Axes to your figure, do so with the `add_subplots()` function. There is, however, a difference between the `add_axes()` and the `add_subplots()` function, but you'll learn more about this later on in the tutorial.

Consider the following example:

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

```
8 # Set up axes
9 ax = fig.add_subplot(111)
10
11 # Scatter the data
12 ax.scatter(np.linspace(0, 1, 5), np.linspace(0, 1, 5))
13
14 # Show the plot
15 plt.show()
```

Run

You see that the `add_subplot()` function in itself also poses you with a challenge, because you see `add_subplots(111)` in the above code chunk.

What does `111` mean?

Well, `111` is equal to `1, 1, 1`, which means that you actually give three arguments to `add_subplot()`. The three arguments designate the number of rows (1), the number of columns (1) and the plot number (1). So you actually make one subplot.

**Note** that you can really go bananas with this function when you are using this function, especially when you're just starting out with this library and you keep on forgetting for what the three numbers stand.

Consider the following commands and try to envision what the plot will look like and how many Axes your Figure will have: `ax = fig.add_subplot(2, 2, 1)`.

Got it?

That's right, your Figure will have four axes in total arranged in a structure that has two

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

The difference between `fig.add_axes()` and `fig.add_subplot()` doesn't lie in the result: they both return an Axes object. However, they do differ in the mechanism that is used to add the axes: you pass a list to `add_axes()` which is the lower left point, the width and the height. This means that the axes object is positioned in absolute coordinates.

In contrast, the `add_subplot()` function doesn't provide the option to put the axes at a certain position: it does, however, allow the axes to be situated according to a subplot grid, as you have seen in the section above.

In most cases, you'll use `add_subplot()` to create axes; Only in cases where the positioning matters, you'll resort to `add_axes()`. Alternatively, you can also use `subplots()` if you want to get one or more subplots at the same time. You'll see an example of how this works in the next section.

## How To Change The Size of Figures

Now that you have seen how to initialize a Figure and Axes from scratch, you will also want to know how you can change certain small details that the package sets up for you, such as the figure size.

Let's say you don't have the luxury to follow along with the defaults and you want to change this. How do you set the size of your figures manually?

Like everything with this package, it's pretty easy, but you need to know first what to change.

Add an argument `figsize` to your `plt.figure()` function of the `pyplot` module; You just have to specify a tuple with the width and height of your figure in inches, just like this `plt.figure(figsize=(3,4))`, for it to work.

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

```
2 import matplotlib.pyplot as plt
3
4 # Initialize the plot
5 fig = plt.figure(figsize=(20,10))
6 ax1 = fig.add_subplot(121)
7 ax2 = fig.add_subplot(122)
8
9 # or replace the three lines of code above
  following line:
10 #fig, (ax1, ax2) = plt.subplots(1,2, figsi
11
12 # Plot the data
13 ax1.bar([1,2,3],[3,4,5])
14 ax2.barh([0.5,1,2.5],[0,1,2])
15
16 # Show the plot
17 plt.show()
```

**Run**

## Working With Pyplot: Plotting Routines

Now that all is set for you to start plotting your data, it's time to take a closer look at some plotting routines. You'll often come across functions like `plot()` and `scatter()`, which either draw points with lines or markers connecting them, or draw unconnected points, which are scaled or colored.

But, as you have already seen in the example of the first section, you shouldn't forget to pass the data that you want these functions to use!

These functions are only the bare basics. You will need some other functions to make sure your plots look awesome:

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

ax.vline()	Vertical line across axes
ax.fill()	Filled polygons
ax.fill_between()	Fill between y-values and 0
ax.stackplot()	Stack plot

If you're curious how you can use these functions to plot your data, consider the following example. **Note** that the `x` and `y` variables have already been loaded in for you:

script.py

```

1  # Import `pyplot` from `matplotlib`
2  import matplotlib.pyplot as plt
3
4  # Initialize the plot
5  fig = plt.figure()
6  ax1 = fig.add_subplot(131)
7  ax2 = fig.add_subplot(132)
8  ax3 = fig.add_subplot(133)
9
10 # Plot the data
11 ax1.bar([1,2,3],[3,4,5])
12 ax2.barh([0.5,1,2.5],[0,1,2])
13 ax2.axhline(0.45)
14 ax1.axvline(0.65)
15 ax3.scatter(x,y)
16
17 # Show the plot
18 plt.show()

```

IPython Shell
In [1]: |

Run

Most functions speak for themselves because the names are quite clear. But that doesn't mean that you need to limit yourself: for example, the `fill_between()` function is perfect for those who want to create area plots, but they can also be used to create a stacked line

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

When you move on and you start to work with vector fields or data distributions, you might want to check out the following functions:

<code>ax.arrow()</code>	Arrow
<code>ax.quiver()</code>	2D field of arrows
<code>ax.streamplot()</code>	2D vector fields
<code>ax.hist()</code>	Histogram
<code>ax.boxplot()</code>	Boxplot
<code>ax.violinplot()</code>	Violinplot

Note of course that you probably won't use all of the functions listed in these tables; It really depends on your data and your use case. If you're totally new to data science, you might want to check out the statistical plotting routines first!

On the other hand, when you work with 2-D or n-D data, you might also find yourself in need of some more advanced plotting routines, like these ones:

<code>ax.pcolor()</code>	Pseudocolor plot
<code>ax.pcolormesh()</code>	Pseudocolor plot
<code>ax.contour()</code>	Contour plot
<code>ax.contourf()</code>	Filled contour plot
<code>ax.clabel()</code>	Labeled contour plot

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

If you're working with images or 2D data, for example, you might also want to check out `imshow()` to show images in your subplots. For a practical example of how to use the `imshow()` function, go to DataCamp's [scikit-learn tutorial](#).

The examples in the tutorial also make clear that this data visualization library is really the cherry on the pie in the data science workflow: you have to be quite well-versed in general Python concepts, such as lists and control flow, which can come especially handy if you want to automate the plotting for a great number of subplots. If you feel like revising these concepts, consider taking the free [introduction to Python for data science](#) course.

## Customizing Your PyPlot

A lot of questions about this package come from the fact that there are a lot of things that you can do to personalize your plots and make sure that they are unique: besides adjusting the colors, you also have the option to change markers, linestyle and linewidths, add text, legend and annotations, and change the limits and layout of your plots.

It's exactly the fact that there is an endless range of possibilities when it comes to these plots that makes it difficult to set out some things that you need to know when you start working on this topic.

Great tips that you should keep in the back of your mind are not only the [gallery](#), which contains many real-life examples that are already coded for you and which you can use, but also the documentation, which can tell you more about the arguments that you can pass to certain functions to adjust visual features.

Also keep in mind that there are multiple solutions for one problem and that you learn most of this stuff when you're getting your hands dirty with the package itself and when you run

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

```
1  # Import `pyplot` from `matplotlib`
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # Initialize the plot
6  fig = plt.figure()
7  ax1 = fig.add_subplot(131)
8  ax2 = fig.add_subplot(132)
9  ax3 = fig.add_subplot(133)
10
11 # Plot the data
12 ax1.bar([1,2,3],[3,4,5])
13 ax2.barh([0.5,1,2.5],[0,1,2])
14 ax2.axhline(0.45)
15 ax1.axvline(0.65)
16 ax3.scatter(np.linspace(0, 1, 5), np.linspace(0, 1, 5))
17
18 # Delete `ax3`
19 fig.delaxes(ax3)
20
21 # Show the plot
22 plt.show()
```

In [1]: |

Run



**Note** that you can restore a deleted axes by adding `fig.add_axes(ax)` right after `fig.delaxes(ax3)`.

## How To Put The Legend Out of the Plot

There are a number of ways to address this question, but mostly all come back to the arguments that you can provide to `legend()`:

- You can specify the `loc` or `location` argument to something like center left or upper right, which ensures that your legend does not fall in the Axes or subplot area.

[Want to leave a comment?](#)



Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

To change your plot title and axes labels, you can follow one of the following approaches, depending on which container of which you want to make use:

- The easiest way to set these things right is by using `ax.set(title="A title", xlabel="x", ylabel="y")` or `ax.set_xlim()`, `ax.set_ylim()` or `ax.set_title()`.
- If you want to work with the figure, you might also resort to `fig.suptitle()` to add a title to your plot.
- If you're making use of the default settings that the package has to offer, you might want to use `plt.title()`, `plt.xlabel()`, `plt.ylabel()`.
- Define your own style sheet or change the default `matplotlibrc` settings. Read more about this [here](#).

## How To Fix The Plot Layout

A thing to consider when you're using subplots to build up your plot is the `tight_layout` function, which will help you to make sure that the plots fit nicely in your figure. You ideally call it after you have plotted your data and customized your plot; So that's right before you call `plt.show()` that you should use `plt.tight_layout()`.

Additionally, you might also be interested to use `subplots_adjust()`, which allows you to manually set the width and height reserved for blank space between subplots, and also fix the left and right sides, and the top and bottom of the subplots.

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs



**R, Python & Data Science Online!**

Get Started for Free

## Showing, Saving And Closing Your Plot

After you have done all the necessary customizations, you will want to show your plot because, as you will have noticed from working in the terminal, you just get to see that an object is made, but you never see the nice plot every time you make adjustments.

In the first example of this tutorial, this was implicitly done;

Do you remember? It's this piece of code:

script.py

```
1  # Import the necessary packages and module
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  # Prepare the data
6  x = np.linspace(0, 10, 100)
7
8  # Plot the data
9  plt.plot(x, x, label='linear')
10
11 # Add a legend
12 plt.legend()
13
14 # Show the plot
15 plt.show()
```

IPython Shell

In [1]: |

Run

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

can you clear the image so that you can start anew? The following short sections will cover these questions.

## How To Save A Plot To An Image File

You can easily save a figure to, for example, a png file by making use of `plt.savefig()`. The only argument you need to pass to this function is the file name, just like in this example:

```
# Save Figure
plt.savefig("foo.png")

# Save Transparent Figure
plt.savefig("foo.png", transparent=True)
```

By just executing this line of code, you'll save the plot that you have made to an image file instead of displaying it.

## How To Save A Plot To A Pdf File

If you want to save multiple plots to a pdf file, you want to make use of the pdf backend, which you can easily import:

```
# Import PdfPages
from matplotlib.backends.backend_pdf import PdfPages

# Initialize the pdf file
pp = PdfPages('multipage.pdf')

# Save the figure to the file
pp.savefig()
```

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

library for the first time, it might be weird at start because you can, of course, shut down the GUI window that appears, but that's usually not the way you want to handle things, because it doesn't always work as well when you're working on several things at a time.

You have to explicitly tell Matplotlib to close down the plot that you've been working on so that you can move on. There are three functions that will come in handy once you're at this point:

- Use `plt.cla()` to clear an axis,
- `plt.clf()` to clear the entire figure, and
- `plt.close()` to close a window that has popped up to show you your plot.

## Customizing Matplotlib

By now, you're already familiar with some basic options to customize your plots. But what if the customizations that you want to make situate more on a library level instead of a plot level? In such cases, also, you don't need to panic: Matplotlib offers you several options to adjust some of the internal workings. This section will just cover two options, namely style sheets and rc settings.

If you want to know more, definitely check out [this page](#).

### How To Use A ggplot2 Style

For the R enthusiasts among you, Matplotlib also offers you the option to set the style of the plots to `ggplot`. You can easily do this by running the following piece of code:

```
# Import `pyplot`  
import matplotlib.pyplot as plt
```

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

having configs as executables: they are automatically run and configure settings, for example. You can read more about it [here](#). Matplotlib has such an rc file to which you can make adjustments dynamically and statically.

To dynamically change default rc settings, you can use the `rcParams` variable:

script.py	IPython Shell
<pre>1  # Import the necessary packages and module 2  import matplotlib as mpl 3  import matplotlib.pyplot as plt 4  import numpy as np 5 6  # Uncomment following line to see the effect 7  #mpl.rcParams['lines.linewidth'] = 5 8 9  # Prepare the data 10 x = np.linspace(0, 10, 100) 11 12 # Plot the data 13 plt.plot(x, x, label='linear') 14 15 # Add a legend 16 plt.legend() 17 18 # Show the plot 19 plt.show()</pre>	<pre>In [1]:  </pre>
<b>Run</b> ●	

You just adjusted the line width in the example above, but you can also change figure size and dpi, line width, color and style, axes, axis and grid properties, text and font properties, ...

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

```
matplotlib.matplotlib_fname()
```

Next, you can pull up the file and start playing with the settings!

## Continue Learning

Congratulations! You have gone through today's Matplotlib tutorial successfully! There is still much to learn, but you're definitely ready to go out on your own and create your own amazing plots. Don't miss out on DataCamp's [Matplotlib cheat sheet](#) that can help you to make plots in no time, step by step.

If you're eager to discover more from Matplotlib, consider checking out DataCamp's [Viewing 3D Volumetric Data With Matplotlib](#) tutorial to learn how to work with matplotlib's event handler API or this tutorial, in which you'll learn all about [animating your plots](#).

If you're ready to start exploring interactive data visualizations with Python, consider taking DataCamp's [Bokeh course](#) and don't miss our [Bokeh cheat sheet](#).

▲  
83

💬  
13



### RELATED POSTS

PYTHON +4

[Python Exploratory Data Analysis Tutorial](#)

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in **2 days 08 hrs 39 mins 02 secs**

DATA VISUALIZATION +1

## Viewing 3D Volumetric Data With Matplotlib

**Juan Nunez-Iglesias**  
April 19th, 2017

PYTHON +1

## Python Seaborn Tutorial For Beginners

**Karlijn Willems**  
August 10th, 2017

### COMMENTS

**Ali Alharbi**

13/03/2018 12:54 AM

when we run Figures, don't show a complete figures

Thank you for the easy explanation

▲ 4 ← **REPLY**

Ozan Serttas

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

## In "How To Set Plot Title And Axes Labels"



*The easiest way to set these things right is by using `ax.set(title="A title", xlabel="x", ylabel="y")` or `ax.set_xlim()`, `ax.set_ylim()` or `ax.set_title()`.*

Maybe `ax.set_xlabel()`, `ax.set_ylabel()` nor `ax.set_xlim()`, `ax.set_ylim()`

▲ 4   ← REPLY

**Ambar Singh**

27/05/2018 10:05 AM

thanks for a good intro

▲ 1   ← REPLY

**Krisan Haria**

01/06/2018 03:47 AM

If i create many figures how do I save them to one pdf document?

▲ 3   ← REPLY

**Jose Torres**

01/06/2018 04:44 AM

Thanks for this tutorial!

▲ 1   ← REPLY

**sara jones**

17/08/2018 10:22 PM

Great Tutorial!

[Want to leave a comment?](#)



Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in 2 days 08 hrs 39 mins 02 secs

**Yash Thaker**

06/09/2018 11:12 PM

Amazing tutorial, Karlijn! Eager to explore more tutorials by you.

▲ 1 ↩ REPLY

**Juan Valera**

12/11/2018 06:08 AM

Thanks ! Great tutorial, clear and easy explanation.

▲ 1 ↩ REPLY

**naveen ganpat**

15/12/2018 03:53 PM

Dear Karlijn,

I enjoy your blogs/tutorials a lot. Recently I started working with the Datacamp Lite to insert some excercises. I was wondereing how did you manage to get two of these apps on the same page. Perhaps this is not the right place to ask this question but I am not able to see a community group to ask technical questions about the Datacamp Light. Would you be able to point me to the right direction? Many thanks!

▲ 1 ↩ REPLY

**durga prasad**

23/12/2018 05:11 PM

it is great website thankyou

▲ 1 ↩ REPLY

**Kaveh Yousefi**

20/01/2019 06:12 PM

Thanks for your tutorial. Your simple and to-the-point instructions make our lives easier!

[Want to leave a comment?](#)

Subscribe now. Save 67% on DataCamp and commit to learning data science and analytics this year.  
Offer ends in **2 days 08 hrs 39 mins 02 secs**

▲ 1 ↩ REPLY

 [Subscribe to RSS](#)



[About](#) [Terms](#) [Privacy](#)

Want to leave a comment?