Photo by Markus Winkler on Unsplash

Figures, plots & subplots: A simple cheatsheet for plotting graphs & images in Python

Takashi Nakamura, PhD
Dec 2, 2019 · 4 min read

As a former electrical engineering student, my ‘go-to’ language has always been Matlab. Matlab is great for numerical analysis (including implementing deep learning models with recent updates); however, Matlab is not free.

During my undergraduate studies, I learnt Python. Python is one of the most popular programming languages, especially in the field of data science; it has many built-in functions and modules to facilitate data analysis.

In fact, my ‘go-to’ language has recently been shifting to Python. I am falling in love with Python. Python is absolutely great, but I always struggle with making figures using `matplotlib` — which is one of the most commonly used Python modules for plotting figures, graphs, charts, etc.

For me (or perhaps many people), the confusing part of `matplotlib` is that the following four statements are not particularly intuitive:

1. plot: `matplotlib.pyplot`
2. subplot: `matplotlib.pyplot.subplots`
3. fig: `matplotlib.figure.Figure`
4. axis: `matplotlib.axes.Axes`

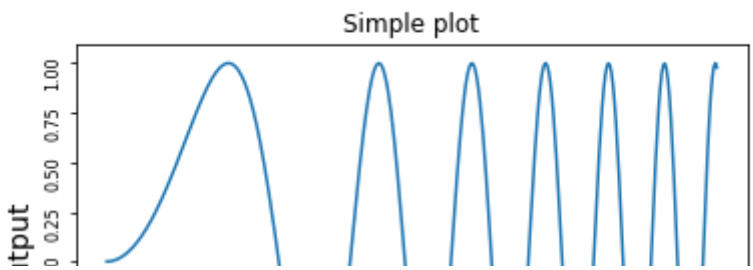
This blog post will be my future 'cheat sheet' for `matplotlib`.

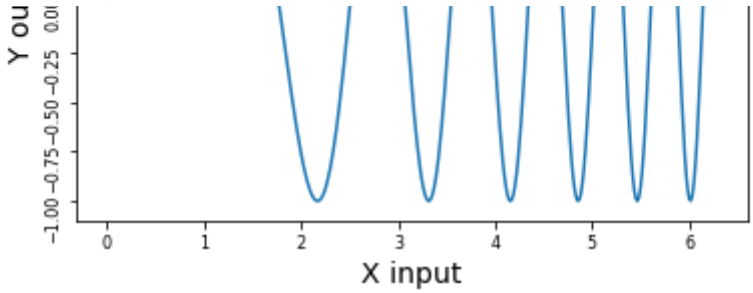
Simple plot

Let's start with a simple example. We have numerical input (`x`) and output (`y`) values. We can plot this with `plt.plot()` and save the plot with `plt.savefig()`.

```
import numpy as np
import matplotlib.pyplot as plt

# Input x, output y
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)
# Plot
plt.plot(x, y)
# Change the fontsize of ticks
plt.xticks(fontsize=8)
plt.yticks(fontsize=8, rotation=90)
# Add label
plt.xlabel('X input', fontsize=14)
plt.ylabel('Y output', fontsize=14)
plt.title('Simple plot')
# Save plot
plt.savefig('Simple_plot.png')
```

simple_plot_1.py hosted with ♡ by GitHub view raw





A simple plot

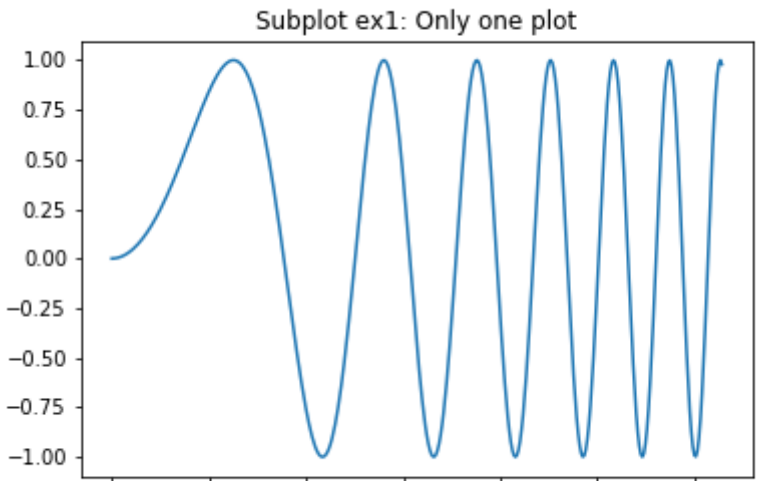
Subplots

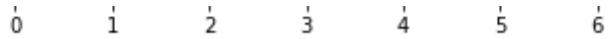
A single subplot

Let's take a look at `matplotlib.subplots`. Note, we are going to save the **figure** with `fig.savefig()`, whereas we used `fig.savefig()` on the previous example in order to save the **plot**.

```
import numpy as np
import matplotlib.pyplot as plt

# Input x, output y
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

# Creates fig and ax from subplots().
# But only create a single plot
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title('Subplot ex1: Only one plot')
# Save figure
fig.savefig('Subplot_ex1.png')
```

subplots_ex1.py hosted with ♡ by GitHub view raw





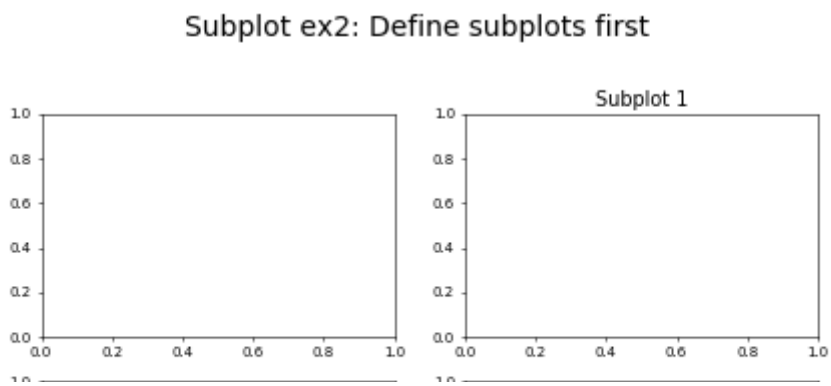
Subplot example 1

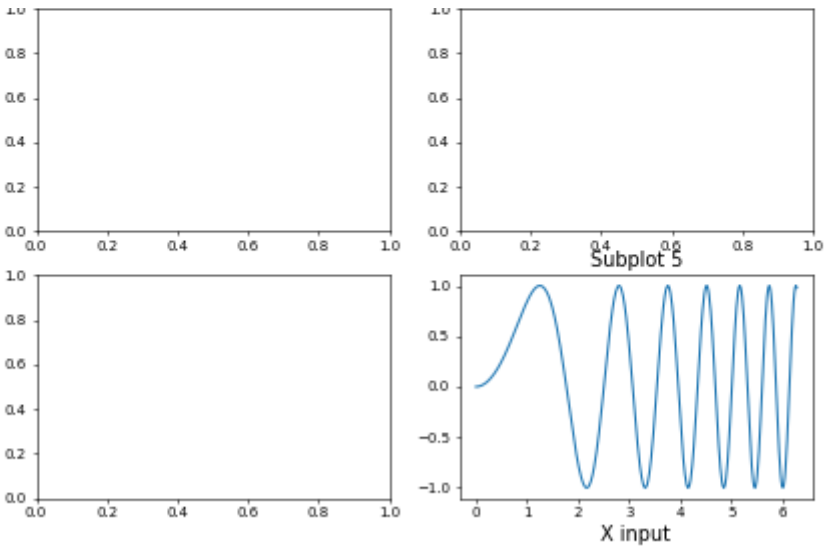
Subplots (Define the number of subplots first)

This time, we are going to create multiple plots on a single figure. We define the number of plots at the beginning.

```
import numpy as np
import matplotlib.pyplot as plt

# Input x, output y
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

# make multiple subplots.
# you can define the size of figure and dpi (dot per inch, defalt dpi=72)
my_dpi = 50
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(10, 10), dpi=my_dpi)
print(fig)
print(axes)

# title for entire figure
fig.suptitle('Subplot ex2: Define subplots first', fontsize=20)

# edit subplots
axes[0, 1].set_title('Subplot 1', fontsize=14)

axes[2, 1].plot(x, y)
axes[2, 1].set_xlabel('X input', fontsize=14)
axes[2, 1].set_title('Subplot 5', fontsize=14)

# Save figure
fig.savefig('Subplot_ex2.png')
```

subplots_ex2.py hosted with ♡ by GitHub view raw





Subplot example 2

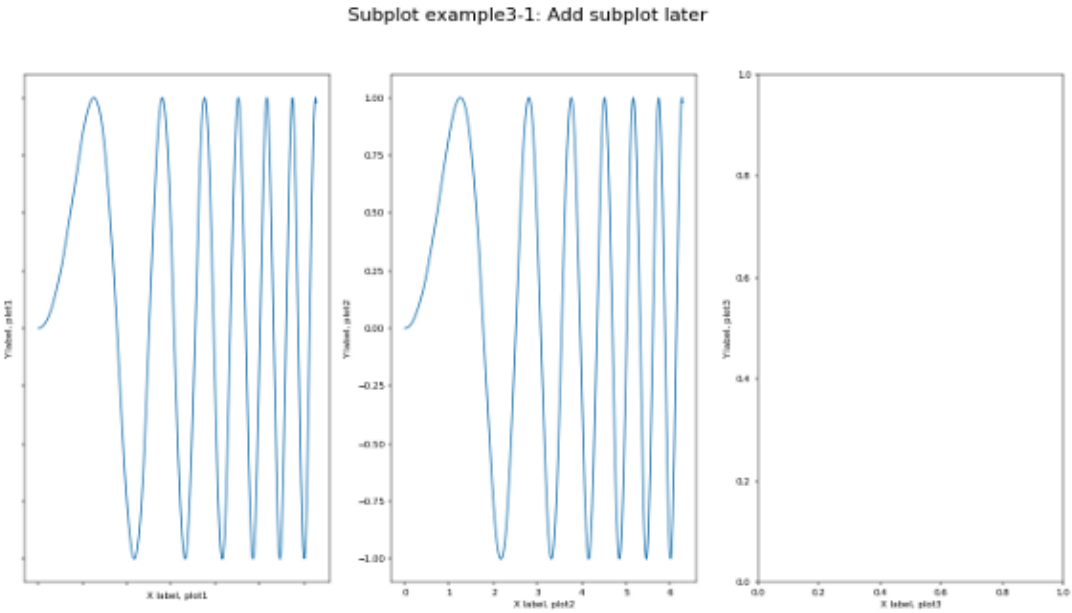
Subplots (Not define the number of plots first, but add plots one-by-one)

We are able to create a figure without defining the number of plots a priori. In this case, we need to define the location of plots.

```
import numpy as np
import matplotlib.pyplot as plt

# Input x, output y
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

# Creates figure first
my_dpi = 40
fig = plt.figure(figsize=(20, 10), dpi=my_dpi)
print(fig)

fig.suptitle('Subplot example3-1: Add subplot later', fontsize=20)

# Add plots
ax1 = fig.add_subplot(1, 3, 1)
ax1.plot(x, y)
ax1.set_xlabel('X label, plot1')
ax1.set_ylabel('Y label, plot1')
ax1.set_xticklabels('')
ax1.set_yticklabels('')

ax2 = fig.add_subplot(1, 3, 2)
ax2.plot(x, y)
ax2.set_xlabel('X label, plot2')
```

```
ax2.set_ylabel('Y label, plot2')

ax3 = fig.add_subplot(1, 3, 3)
ax3.set_xlabel('X label, plot3')
ax3.set_ylabel('Y label, plot3')

fig.savefig('Subplot_ex3.png')
```

subplots_ex3.py hosted with ♡ by GitHub view raw



Subplot example 3

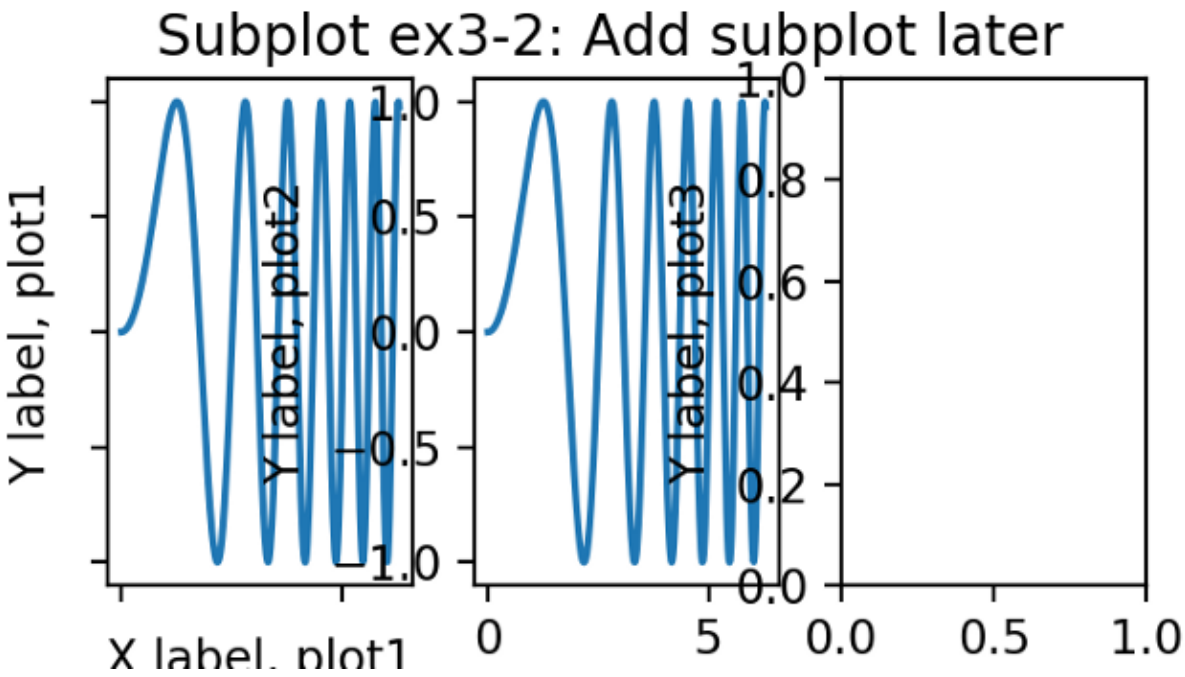
Subplots (Same size, but change the dots per inch)

You may notice in the figure above that the axes and ticks are too small to read. We are able to change the font size, but also the dpi. The size of the figure is `figsize` × `dpi`. The previous figure size was 800×400, which is because `figsize` × `dpi` = (20, 10)×40 = (800, 400). The figure size of this following example is also 800×400, but `figsize` =(4, 2) and `dpi` =200. In other words, (4, 2)×200 = (800, 400).

```
import numpy as np
import matplotlib.pyplot as plt

# Input x, output y
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)

# Creates figure first
my_dpi = 200
fig = plt.figure(figsize=(4, 2), dpi=my_dpi)
```

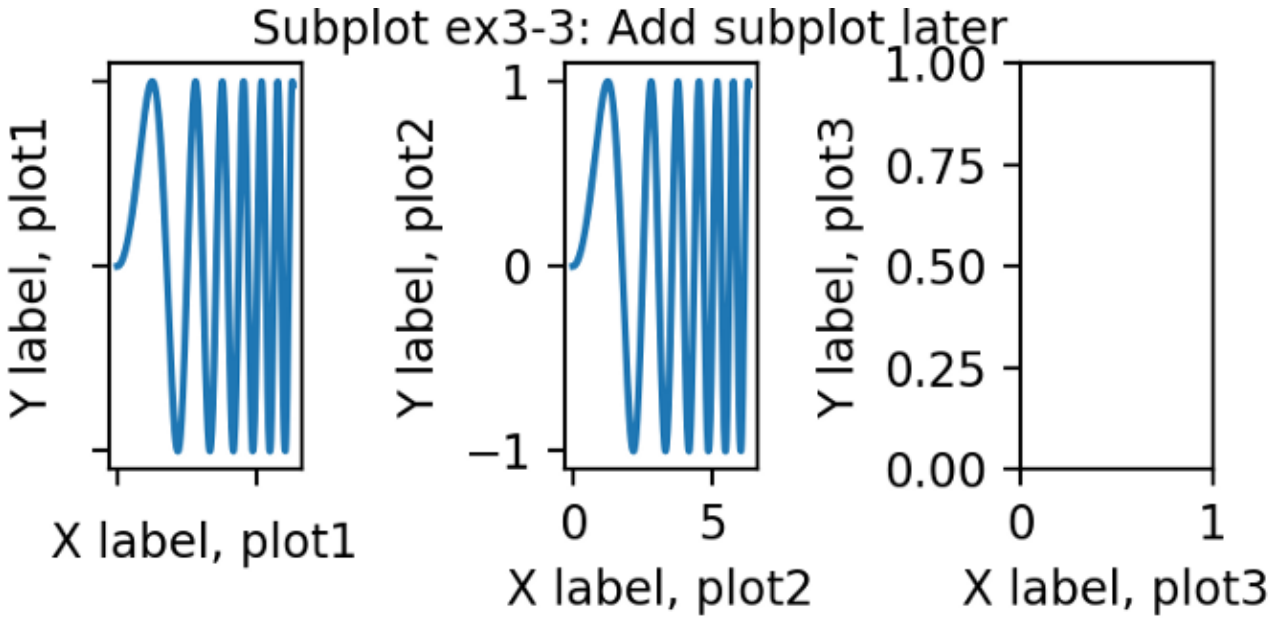
```
print(fig)

fig.suptitle('Subplot ex3-2: Add subplot later')

# Add plot
ax1 = fig.add_subplot(1, 3, 1)
ax1.plot(x, y)
ax1.set_xlabel('X label, plot1')
ax1.set_ylabel('Y label, plot1')
ax1.set_xticklabels('')
ax1.set_yticklabels('')

# Add plot
ax2 = fig.add_subplot(1, 3, 2)
ax2.plot(x, y)
ax2.set_xlabel('X label, plot2')
ax2.set_ylabel('Y label, plot2')

# Add plot
ax3 = fig.add_subplot(1, 3, 3)
ax3.set_xlabel('X label, plot3')
ax3.set_ylabel('Y label, plot3')

fig.savefig('Subplot_ex4.png')
```

subplot_ex4.py hosted with ♡ by GitHub view raw



Subplot example 4, high dpi

If you add another line, `plt.tight_layout()` , the layout becomes much neat.



Subplot example 5, tight layout

Plot image files

We have been playing around with subplots for a while. Finally, let's try to plot images. In Python, there are multiple ways to open image files, for example:

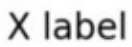
1. `matplotlib mpimg`
2. `Pillow Image`
3. `OpenCV cv2`

There are some nuances with the syntax for each module, and we must be especially careful when we open image files via `cv2` and plot them. On top of this, being a computer vision enthusiast, I use PyTorch often. The following example covers how to tensorise and de-tensorise image files.

```
import numpy as np
import matplotlib.pyplot as plt
# image modules
from PIL import Image
import matplotlib.image as mpimg
import cv2
# PyTorch
import torch
from torch.utils import data
from torchvision import transforms
# other module
```

```
import os

my_dpi = 200
fig = plt.figure(figsize=(6, 6), dpi=my_dpi)

# ============ AX1 ============
# PIL Image
ax1 = fig.add_subplot(2, 3, 1)
ax1.set_title("PIL Image")
ax1.set_xlabel('X label')
ax1.set_ylabel('Y label')
ax1.set_xticks([])
ax1.set_yticks([])
pil_img = Image.open(os.path.join('my_data', 'img1.jpg'))
ax1.imshow(pil_img)

# ============ AX2 ============
# mpimg image
ax2 = fig.add_subplot(2, 3, 2)
ax2.set_title("mpimg image")
ax2.set_xticks([])
ax2.set_yticks([])
mpimg_img = mpimg.imread(os.path.join('my_data', 'img2.jpg'))
ax2.imshow(mpimg_img)

# ============ AX3 ============
# CV2 image (default)
ax3 = fig.add_subplot(2, 3, 3)
ax3.set_title("CV2 image (default)")
ax3.set_xticks([])
ax3.set_yticks([])
opencv_img = cv2.imread(os.path.join('my_data', 'img3.jpg'))
ax3.imshow(opencv_img)

# ============ AX4 ============
# CV2 image (transform)
ax4 = fig.add_subplot(2, 3, 4)
ax4.set_title("CV2 image (transform)")
ax4.set_xticks([])
ax4.set_yticks([])
cv2_img = cv2.imread(os.path.join('my_data', 'img3.jpg'))
mod_cv2_img = cv2.cvtColor(cv2_img, cv2.COLOR_BGR2RGB)
ax4.imshow(mod_cv2_img)

# ============ AX5 ============
# CV2 image (transform)
ax5 = fig.add_subplot(2, 3, 5)
ax5.axis('off')
```

```
cv2_img = cv2.imread(os.path.join('my_data', 'img3.jpg'))
mod_cv2_img = cv2.cvtColor(cv2_img, cv2.COLOR_BGR2RGB)
ax5.imshow(mod_cv2_img)

# ============ AX6 ============
# PIL image. With PyTorch, tensorise and de-tensorise
ax6 = fig.add_subplot(2, 3, 6)
ax6.set_title("PIL image (tensor)")
ax6.set_xlabel('X label')
ax6.set_ylabel('Y label')
ax6.set_xticklabels('')
ax6.set_yticklabels('')
pil_image = Image.open(os.path.join('my_data', 'img1.jpg'))
tensor_image = transforms.ToTensor()(pil_image)
de_tensor_image = transforms.ToPILImage()(tensor_image)
ax6.imshow(de_tensor_image)

fig.savefig("saved_images.jpg", dpi=DPI, bbox_inches='tight')
```



Plot images via different modules

Summary

Key ideas are 1) we are able to make a plot via `subplots`, and 2) we are able to make a figure comprising of multiple plots via `subplots`.

Hope this article helps you as much as it has helped me!

Data Science