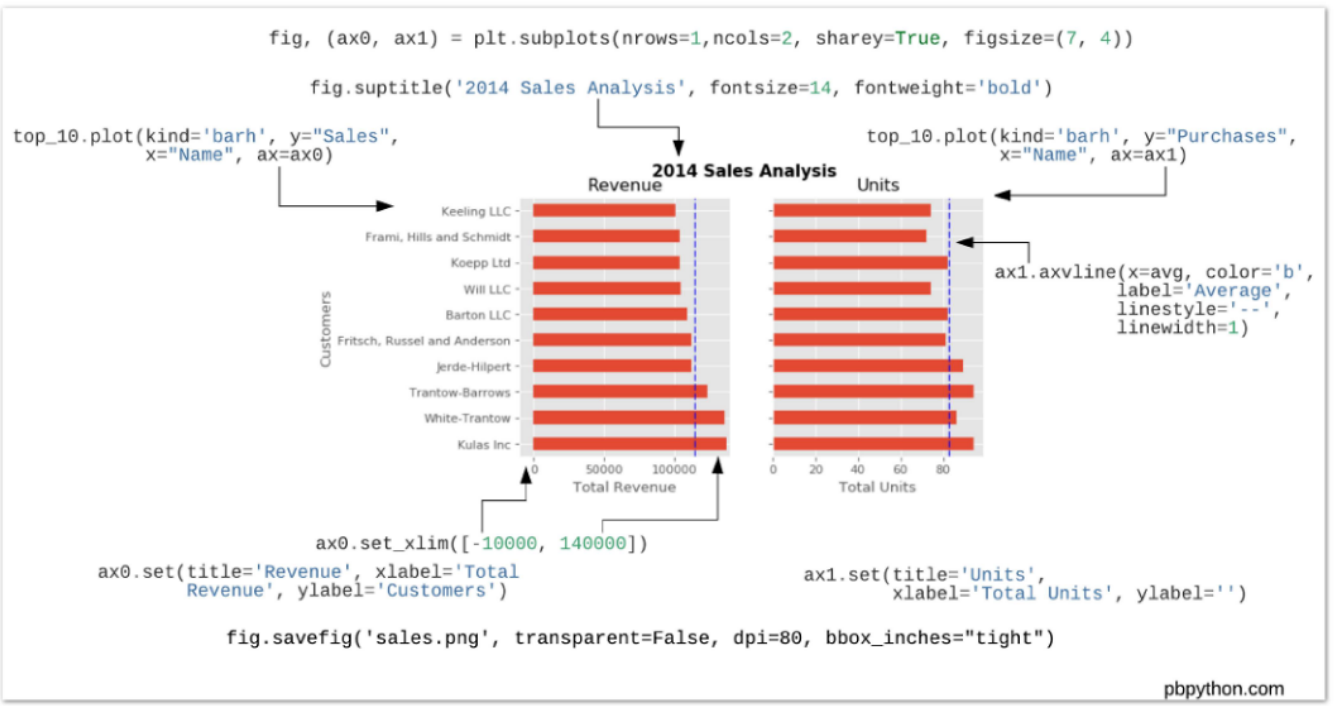
Practical Business Python

Taking care of business, one python script at a time

Tue 25 April 2017

Effectively Using Matplotlib

Posted by Chris Moffitt in articles



Introduction

The python visualization world can be a frustrating place for a new user. There are many different options and choosing the right one is a challenge. For example, even after 2 years, this article is one of the top posts that lead people to this site. In that article, I threw some shade at matplotlib and dismissed it during the analysis. However, after using tools such as pandas, scikit-learn, seaborn and the rest of the data science stack in python - I think I was a little premature in dismissing matplotlib. To be honest, I did not quite understand it and how to use it effectively in my workflow.

Now that I have taken the time to learn some of these tools and how to use them with matplotlib, I have started to see matplotlib as an indispensable tool. This post will show how I use matplotlib and provide some recommendations for users getting started or users who have not taken the time to learn matplotlib. I do firmly believe matplotlib is an essential part of the python data science stack and hope this article will help people understand how to use it for their own visualizations.

Why all the negativity towards matplotlib?

In my opinion, there are a couple of reasons why matplotlib is challenging for the new user to learn.

First, matplotlib has two interfaces. The first is based on MATLAB and uses a state-based interface. The second option is an an object-oriented interface. The why's of this dual approach are outside the scope of this post but *knowing* that there are two approaches is vitally important when plotting with matplotlib.

The reason two interfaces cause confusion is that in the world of stack overflow and tons of information available via google searches, new users will stumble across multiple solutions to problems that look somewhat similar but are not the same. I can speak from experience. Looking back on some of my old code, I can tell that there is a mishmash of matplotlib code - which is confusing to me (even if I wrote it).

Key Point
New matplotlib users should learn and use the object oriented interface.

Another historic challenge with matplotlib is that some of the default style choices were rather unattractive. In a world where R could generate some really cool plots with ggplot, the matplotlib options tended to look a bit ugly in comparison. The good news is that matplotlib 2.0 has much nicer styling capabilities and ability to theme your visualizations with minimal effort.

The third challenge I see with matplotlib is that there is confusion as to when you should use pure matplotlib to plot something vs. a tool like pandas or seaborn that is built on top of matplotlib. Anytime there can be more than one way to do something, it is challenging for the new or infrequent user to follow the right path. Couple this confusion with the two different API's and it is a recipe for frustration.

Why stick with matplotlib?

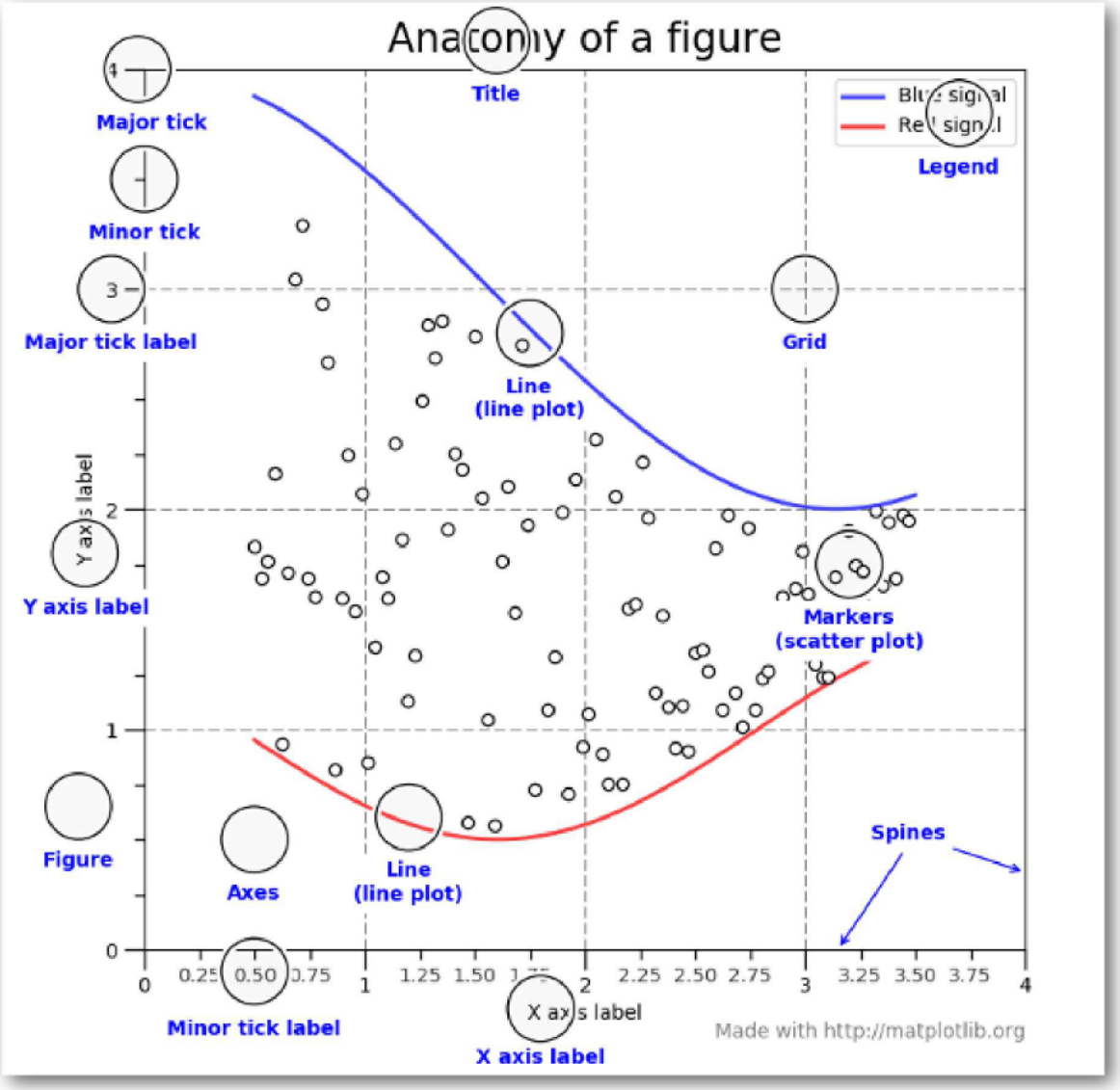
Despite some of these issues, I have come to appreciate matplotlib because it is extremely powerful. The library allows you to create almost any visualization you could imagine. Additionally, there is a rich ecosystem of python tools built around it and many of the more advanced visualization tools use matplotlib as the base library. If you do any work in the python data science stack, you will need to develop some basic familiarity with how to use matplotlib. That is the focus of the rest of this post - developing a basic approach for effectively using matplotlib.

Basic Premises

If you take nothing else away from this post, I recommend the following steps for learning how to use matplotlib:

1. Learn the basic matplotlib terminology, specifically what is a `Figure` and an `Axes` .
2. Always use the object-oriented interface. Get in the habit of using it from the start of your analysis.
3. Start your visualizations with basic pandas plotting.
4. Use seaborn for the more complex statistical visualizations.
5. Use matplotlib to customize the pandas or seaborn visualization.

This graphic from the matplotlib faq is gold. Keep it handy to understand the different terminology of a plot.



Most of the terms are straightforward but the main thing to remember is that the `Figure` is the final image that may contain 1 or more axes. The `Axes` represent an individual plot. Once you understand what these are and how to access them through the object oriented API, the rest of the process starts to fall into place.

The other benefit of this knowledge is that you have a starting point when you see things on the web. If you take the time to understand this point, the rest of the matplotlib API will start to make sense. Also, many of the advanced python packages like seaborn and ggplot rely on matplotlib so understanding the basics will make those more powerful frameworks much easier to learn.

Finally, I am not saying that you should avoid the other good options like ggplot (aka ggpy), bokeh, plotly or altair. I just think you'll need a basic understanding of matplotlib + pandas + seaborn to start. Once you understand the basic visualization stack, you can explore the other options and make informed choices based on your needs.

Getting Started

The rest of this post will be a primer on how to do the basic visualization creation in pandas and customize the most common items using matplotlib. Once you understand the basic process, further customizations are relatively straightforward.

I have focused on the most common plotting tasks I encounter such as labeling axes, adjusting limits, updating plot titles, saving figures and adjusting legends. If you would like to follow along, the notebook includes additional detail that should be helpful.

To get started, I am going to setup my imports and read in some data:

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter

df = pd.read_excel("https://github.com/chris1610/pbpython/blob/master/data/sample-sal
df.head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2014-01-01 10:00:47
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22
4	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2014-01-01 23:26:55

The data consists of sales transactions for 2014. In order to make this post a little shorter, I'm going to summarize the data so we can see the total number of purchases and total sales for the top 10 customers. I am also going to rename columns for clarity during plots.

```
top_10 = (df.groupby('name')['ext price', 'quantity'].agg({'ext price': 'sum', 'quant
          .sort_values(by='ext price', ascending=False))[:10].reset_index()
top_10.rename(columns={'name': 'Name', 'ext price': 'Sales', 'quantity': 'Purchases'}
```

Here is what the data looks like.

	Name	Purchases	Sales
0	Kulas Inc	94	137351.96
1	White-Trantow	86	135841.99
2	Trantow-Barrows	94	123381.38
3	Jerde-Hilpert	89	112591.43
4	Fritsch, Russel and Anderson	81	112214.71
5	Barton LLC	82	109438.50
6	Will LLC	74	104437.60
7	Koepp Ltd	82	103660.54
8	Frami, Hills and Schmidt	72	103569.59
9	Keeling LLC	74	100934.30

Now that the data is formatted in a simple table, let's talk about plotting these results as a bar chart.

As I mentioned earlier, matplotlib has many different styles available for rendering plots. You can see which ones are available on your system using `plt.style.available` .

```
plt.style.available
```

```
['seaborn-dark',
 'seaborn-dark-palette',
 'fivethirtyeight',
 'seaborn-whitegrid',
 'seaborn-darkgrid',
 'seaborn',
 'bmh',
 'classic',
 'seaborn-colorblind',
 'seaborn-muted',
 'seaborn-white',
 'seaborn-talk',
 'grayscale',
 'dark_background',
 'seaborn-deep',
 'seaborn-bright',
 'ggplot',
 'seaborn-paper',
 'seaborn-notebook',
 'seaborn-poster',
 'seaborn-ticks',
 'seaborn-pastel']
```

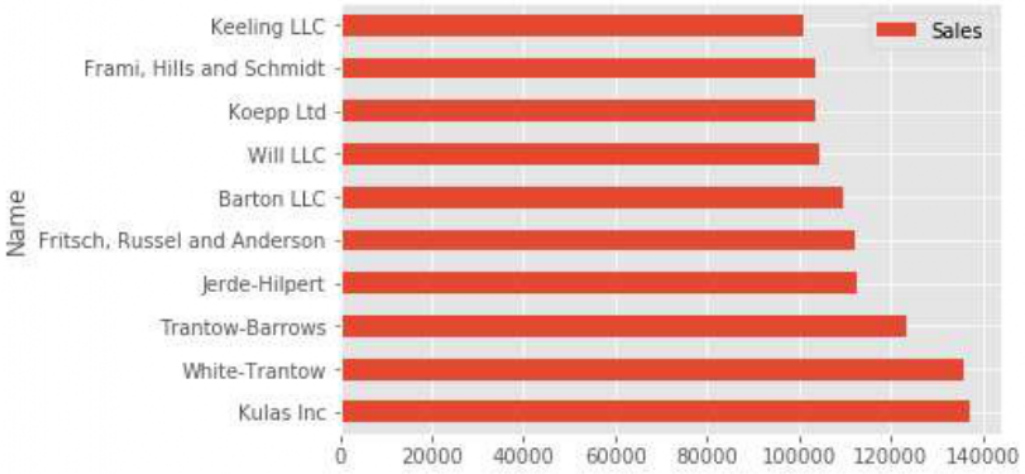
Using a style is as simple as:

```
plt.style.use('ggplot')
```

I encourage you to play around with different styles and see which ones you like.

Now that we have a nicer style in place, the first step is to plot the data using the standard pandas plotting function:

```
top_10.plot(kind='barh', y="Sales", x="Name")
```



The reason I recommend using pandas plotting first is that it is a quick and easy way to prototype your visualization. Since most people are probably already doing some level of data manipulation/analysis in pandas as a first step, go ahead and use the basic plots to get started.

Customizing the Plot

Assuming you are comfortable with the gist of this plot, the next step is to customize it. Some of the customizations (like adding titles and labels) are very simple to use with the pandas `plot` function. However, you will probably find yourself needing to move outside of that functionality at some point. That's why I recommend getting in the habit of doing this:

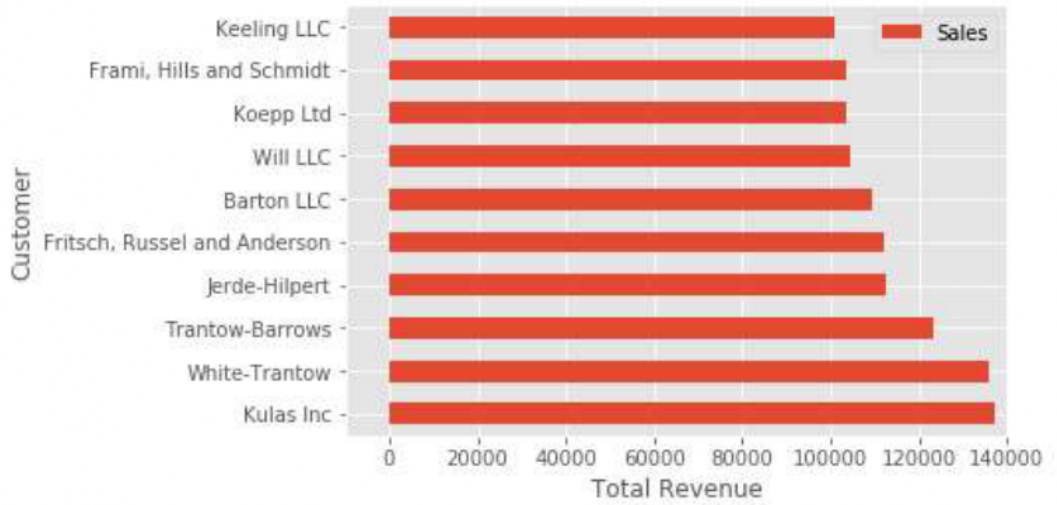
```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
```

The resulting plot looks exactly the same as the original but we added an additional call to `plt.subplots()` and passed the `ax` to the plotting function. Why should you do this? Remember when I said it is critical to get access to the axes and figures in matplotlib? That's what we have accomplished here. Any future customization will be done via the `ax` or `fig` objects.

We have the benefit of a quick plot from pandas but access to all the power from matplotlib now. An example should show what we can do now. Also, by using this naming convention, it is fairly straightforward to adapt others' solutions to your unique needs.

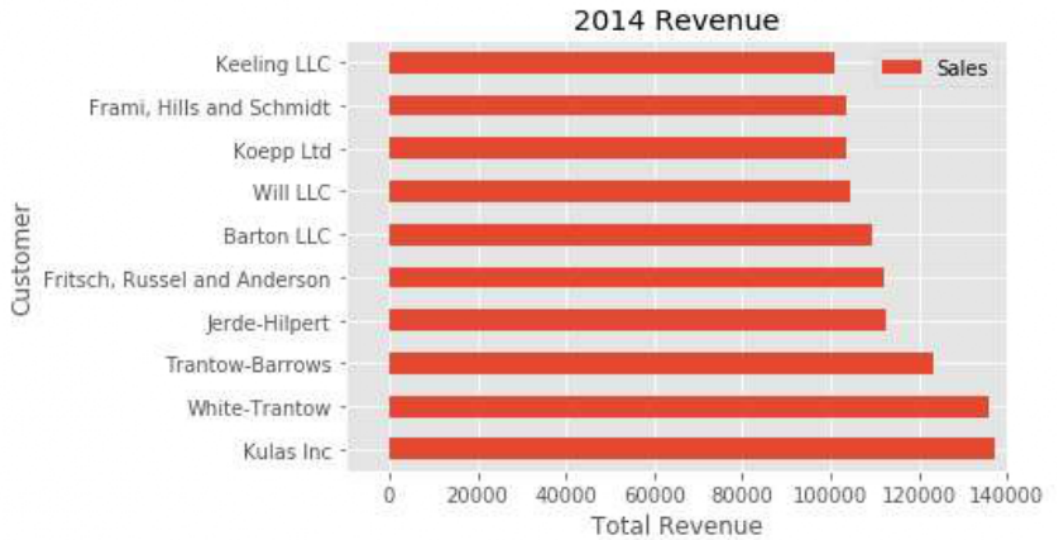
Suppose we want to tweak the x limits and change some axis labels? Now that we have the axes in the `ax` variable, we have a lot of control:

```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set_xlabel('Total Revenue')
ax.set_ylabel('Customer');
```



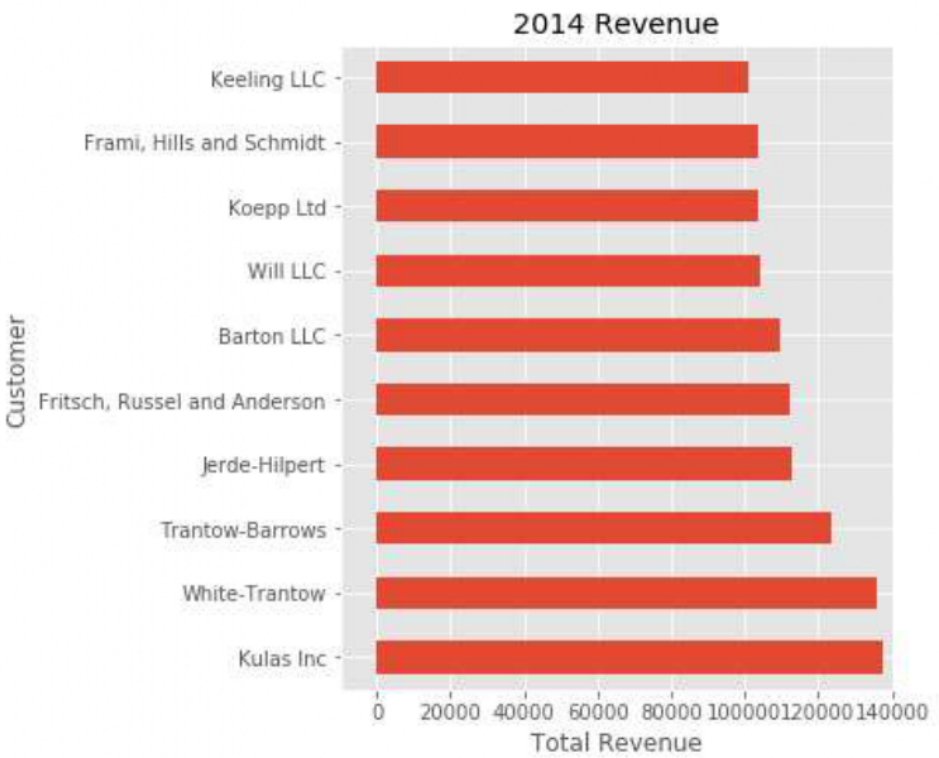
Here's another shortcut we can use to change the title and both labels:

```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')
```



To further demonstrate this approach, we can also adjust the size of this image. By using the `plt.subplots()` function, we can define the `figsize` in inches. We can also remove the legend using `ax.legend().set_visible(False)`

```
fig, ax = plt.subplots(figsize=(5, 6))
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue')
ax.legend().set_visible(False)
```



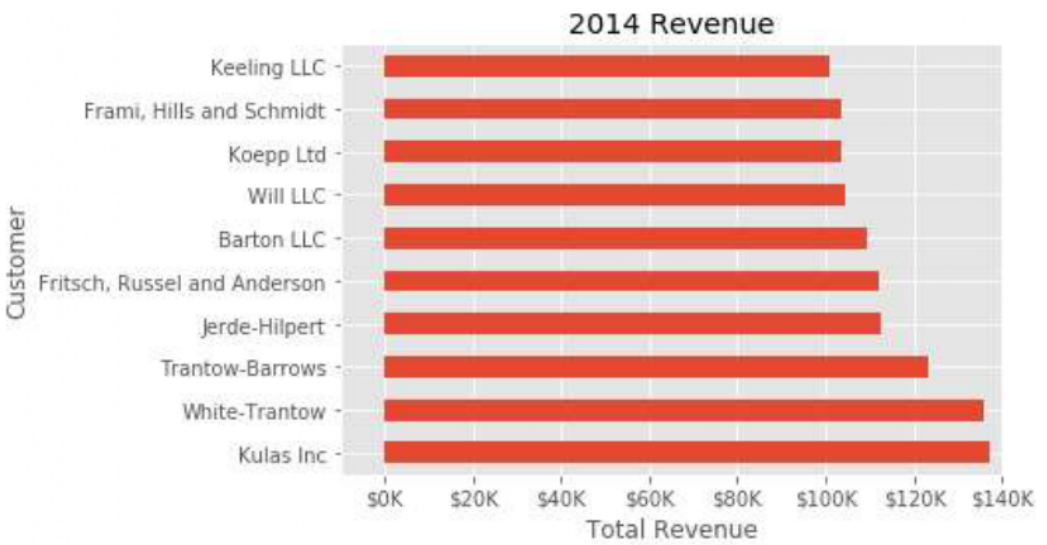
There are plenty of things you probably want to do to clean up this plot. One of the biggest eye sores is the formatting of the Total Revenue numbers. Matplotlib can help us with this through the use of the `FuncFormatter`. This versatile function can apply a user defined function to a value and return a nicely formatted string to place on the axis.

Here is a currency formatting function to gracefully handle US dollars in the several hundred thousand dollar range:

```
def currency(x, pos):
    'The two args are the value and tick position'
    if x >= 1000000:
        return '${:1.1f}M'.format(x*1e-6)
    return '${:1.0f}K'.format(x*1e-3)
```

Now that we have a formatter function, we need to define it and apply it to the x axis. Here is the full code:

```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')
formatter = FuncFormatter(currency)
ax.xaxis.set_major_formatter(formatter)
ax.legend().set_visible(False)
```

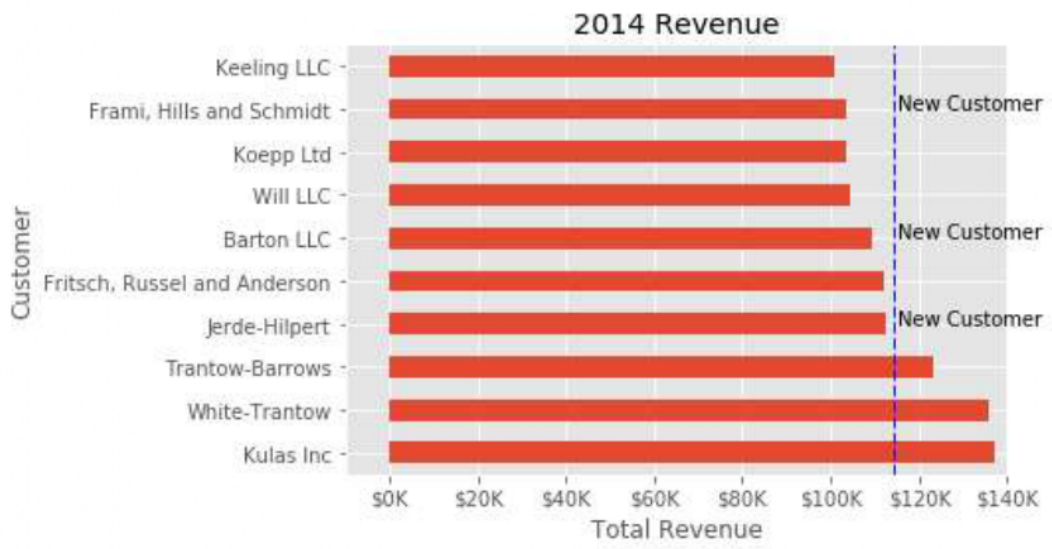


That's much nicer and shows a good example of the flexibility to define your own solution to the problem.

The final customization feature I will go through is the ability to add annotations to the plot. In order to draw a vertical line, you can use `ax.axvline()` and to add custom text, you can use `ax.text()` .

For this example, we'll draw a line showing an average and include labels showing three new customers. Here is the full code with comments to pull it all together.

```
# Create the figure and the axes
fig, ax = plt.subplots()

# Plot the data and get the averaged
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
avg = top_10['Sales'].mean()

# Set limits and labels
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')

# Add a line for the average
ax.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

# Annotate the new customers
for cust in [3, 5, 8]:
    ax.text(115000, cust, "New Customer")

# Format the currency
formatter = FuncFormatter(currency)
ax.xaxis.set_major_formatter(formatter)

# Hide the legend
ax.legend().set_visible(False)
```



While this may not be the most exciting plot it does show how much power you have when following this approach.

Figures and Plots

Up until now, all the changes we have made have been with the individual plot. Fortunately, we also have the ability to add multiple plots on a figure as well as save the entire figure using various options.

If we decided that we wanted to put two plots on the same figure, we should have a basic understanding of how to do it. First, create the figure, then the axes, then plot it all together. We can accomplish this using `plt.subplots()` :

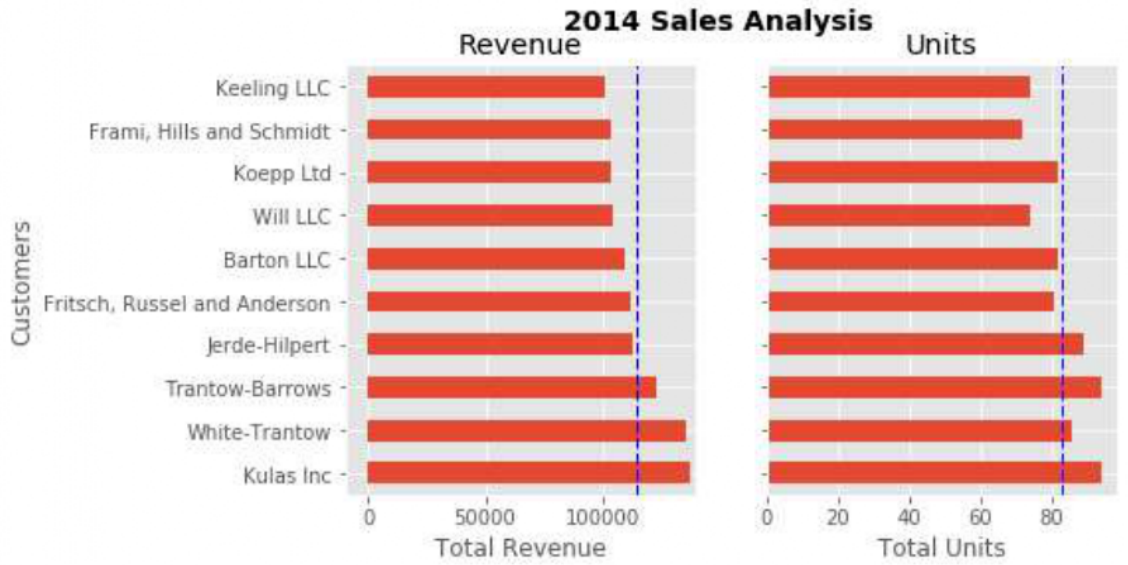
```
fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(7, 4))
```

In this example, I'm using `nrows` and `ncols` to specify the size because this is very clear to the new user. In sample code you will frequently just see variables like 1,2. I think using the named parameters is a little easier to interpret later on when you're looking at your code.

I am also using `sharey=True` so that the yaxis will share the same labels.

This example is also kind of nifty because the various axes get unpacked to `ax0` and `ax1` . Now that we have these axes, you can plot them like the examples above but put one plot on `ax0` and the other on `ax1` .

```
# Get the figure and the axes
fig, (ax0, ax1) = plt.subplots(nrows=1,ncols=2, sharey=True, figsize=(7, 4))
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax0)
ax0.set_xlim([-10000, 140000])
ax0.set(title='Revenue', xlabel='Total Revenue', ylabel='Customers')

# Plot the average as a vertical line
avg = top_10['Sales'].mean()
ax0.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

# Repeat for the unit plot
top_10.plot(kind='barh', y="Purchases", x="Name", ax=ax1)
avg = top_10['Purchases'].mean()
ax1.set(title='Units', xlabel='Total Units', ylabel='')
ax1.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

# Title the figure
fig.suptitle('2014 Sales Analysis', fontsize=14, fontweight='bold');

# Hide the legends
ax1.legend().set_visible(False)
ax0.legend().set_visible(False)
```



Up until now, I have been relying on the jupyter notebook to display the figures by virtue of the `%matplotlib inline` directive. However, there are going to be plenty of times where you have the need to save a figure in a specific format and integrate it with some other presentation.

Matplotlib supports many different formats for saving files. You can use `fig.canvas.get_supported_filetypes()` to see what your system supports:

```
fig.canvas.get_supported_filetypes()
```

```
{'eps': 'Encapsulated Postscript',
 'jpeg': 'Joint Photographic Experts Group',
 'jpg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

Since we have the `fig` object, we can save the figure using multiple options:

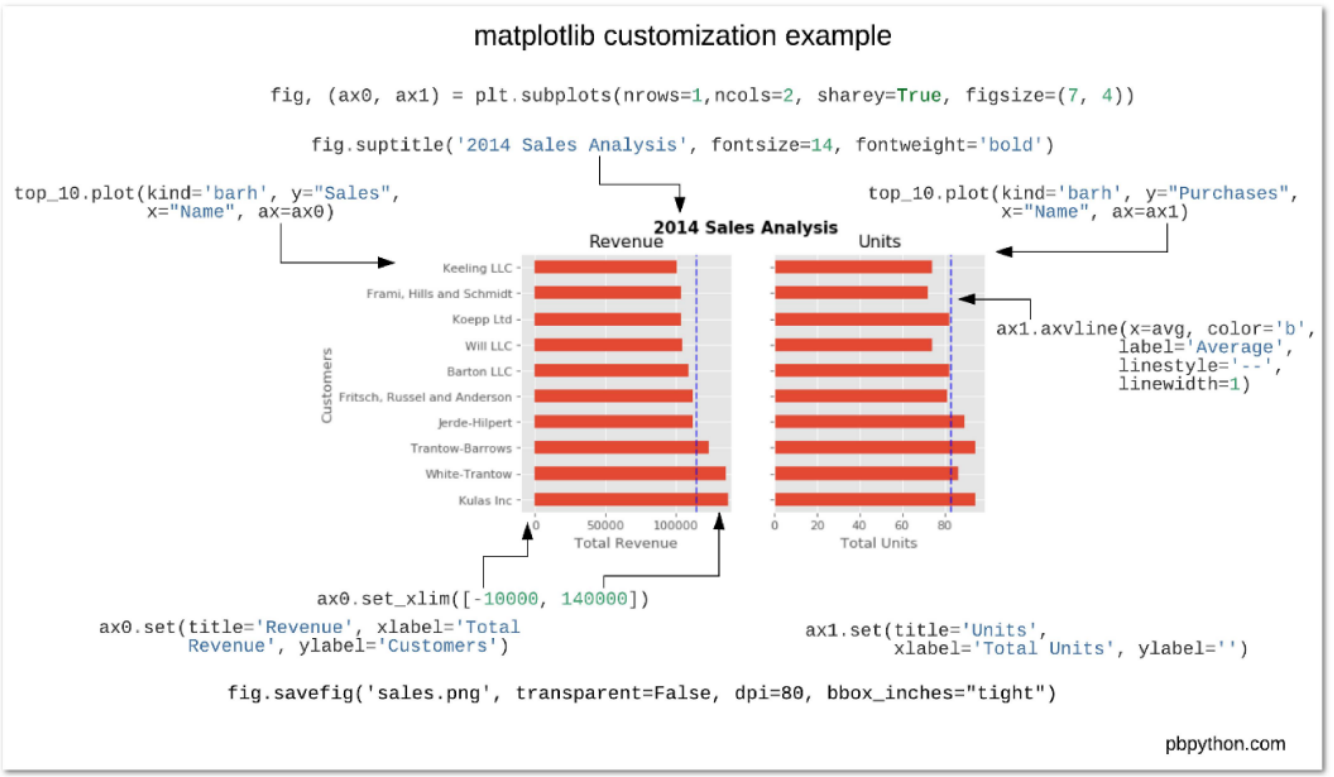
```
fig.savefig('sales.png', transparent=False, dpi=80, bbox_inches="tight")
```

This version saves the plot as a png with opaque background. I have also specified the dpi and `bbox_inches="tight"` in order to minimize excess white space.

Conclusion

Hopefully this process has helped you understand how to more effectively use matplotlib in your daily data analysis. If you get in the habit of using this approach when doing your analysis, you should be able to quickly find out how to do whatever you need to do to customize your plot.

As a final bonus, I am including a quick guide to unify all the concepts. I hope this helps bring this post together and proves a handy reference for future use.



← Understanding the Transform Function in Pandas

How Accurately Can Prophet Project Website Traffic? →

Tags pandas matplotlib

Comments

ALSO ON **PRACTICAL BUSINESS PYTHON**

Exploring an Alternative to Jupyter Notebooks ...

4 years ago · 25 comments

The Jupyter notebook file format has some downsides that can be mitigated with ...

Case Study: Automating Excel File Creation ...

3 years ago · 13 comments

Case Study showing how to break an Excel file into multiple files and email to ...

Reading HTML tables with Pandas

3 years ago · 14 comments

This article describes how to read HTML tables from Wikipedia or other sites ...

Rea Stru

3 yea

This to us to re

Denis Akhiyarov → Neal McBurnett
7 years ago

`%matplotlib notebook` magic already provides tooltip in right lower corner with info on data points when hovering over them. This is very limited, but still addresses the main use case (x & y coordinates).

1 0 Reply • Share ›

Georgy Ayzel → Denis Akhiyarov
7 years ago

One of the core possibility to play with mpl figure is to use %matplotlib notebook notation instead of standard %matplotlib inline

1 0 Reply • Share ›

Denis Akhiyarov → Georgy Ayzel
7 years ago

exactly, but you can also play with the figure much more in the qt backend, when the matplotlib figure is displayed in a popup window. What I would like to is switch between notebook and qt backends with a click of a button.

0 0 Reply • Share ›

Chris Moffitt Mod → Denis Akhiyarov
7 years ago

I agree as well. I have not played with the notebook magic yet but am definitely interested in learning more about that functionality.

0 0 Reply • Share ›

Denis Akhiyarov → Chris Moffitt
7 years ago

notebook magic is much nicer than inline, especially if you need to zoom in/out on the plots without re-running the plotting code.

0 0 Reply • Share ›

bul
7 years ago

Hello Chris, This is an amazing piece about matplotlib. Have you ever worked on facet grids ?
I created this graph is R. Can you get me a similar kind of graph in Python/Pandas? Please advise.

View – uploads.disquscdn.com

4 0 Reply • Share ›

Chris Moffitt Mod → bul

7 years ago

When I need to use facet grids, I typically use seaborn - http://seaborn.pydata.org/i...

There is a FacetGrid documentation page here - http://seaborn.pydata.org/g...

The examples are also a good place to start- http://seaborn.pydata.org/g...

However, if you have experience with R, you may really like plotnine - https://plotnine.readthedoc...

plotnine is almost a direct translation of ggplot2 so you may be able to get up and running more quickly. It is a relatively new project but it looks very promising.

3 0 Reply • Share ›

B

bul → Chris Moffitt

7 years ago

Thanks Chris. But, all these examples in website shows numbers in x axis and y axis. But, i have strings in y axis and numbers in x axis . I am unable to find any link related to that.
Any help is appreciated please

57 0 Reply • Share ›

H

Henrik Eckermann → bul

6 years ago

Hey bul,

is there active development going for this and especially will there be in the future?

0 0 Reply • Share ›

pauloneves

7 years ago

Just to say thanks for the nice writeup. I've been doing dirty plots from examples in the web. This post really clarified all the basic concepts for me.

4 1 Reply • Share ›

C

Chandra Lingam

2 years ago

Outstanding article! Thank you!

2 0 Reply • Share ›

Galina Isaeva

4 years ago

Thank you for such a brilliant article!
Matplotlib is much more clear for me now.

However, I have a question when try to repeat your code from the very beginning.
Why in pd.read_excel argument ?raw=true is used? I mean:
df = pd.read_excel("https://github.com/chris161...")

I tried to load that file as usually only by address, with "no ?raw=true" part and failed. I did:
df = pd.read_excel("https://github.com/chris161...")
Error I faced sounds "XLRDError: Unsupported format, or corrupt file: Expected BOF record; found b'\n\n\n\n\n\n

1 0 Reply • Share ›

Chris Moffitt Mod → Galina Isaeva

4 years ago

Thank you. I am glad you found the article helpful.

It might be easier to understand if you look at the github page here

You can see that github does not know how to render an Excel file. There is a link to download the file using ?raw=True which is what I used in the script.

The other option is to download the file locally on your machine and run from there.

0 0 Reply • Share ›

rosebud

4 years ago edited

Can empathise with all the points here.

However one final question - the docs refer to pyplot as the 'NOT object oriented interface' yet this is what is imported as plt and used in the following object oriented code.... confused!

EDIT found this:
"In many cases you will create a Figure and one or more Axes using pyplot.subplots and from then on only work on these objects. However, it's also possible to create Figures explicitly (e.g. when including them in GUI applications)."

does this mean that people are using the MATLAB interface to get started then switching to OO?

1 0 Reply • Share ›

Chris Moffitt Mod → rosebud

4 years ago

Yes. I think you are correct. Sometimes the MATLAB inspired interface is the way to

get started but use OO for the actual manipulations.

0 0 Reply • Share ›

rosebud → Chris Moffitt

4 years ago

Thank you for clarifying Chris. I think your article in combination with this one (in particular the sections after the 'anatomy of a figure' diagram) covers all the bases!

2 0 Reply • Share ›

陈

陈俊杰

6 years ago

Great work!

Can you give us a tutorial of Seaborn? The documents of it was too hard to understand.. Thanks in advance..

1 0 Reply • Share ›

Chris Moffitt Mod → 陈俊杰

6 years ago

Thank you. I will add it to the list of potential topics to cover. Thanks for bringing up the request.

1 0 Reply • Share ›

T

Theodore Petrou

6 years ago

There is no need to create a figure with `plt.subplots` beforehand. pandas returns the matplotlib object after a call to `plot`. You can just do this.
ax = top_10.plot(kind='barh', y="Sales", x="Name")

1 0 Reply • Share ›

Chris Moffitt Mod → Theodore Petrou

6 years ago

Yes, that is true. But if you want to save your figure or do something at the "fig" level, don't you need to create the subplot?

I could be wrong but that was my rationale for the recommendation.

0 0 Reply • Share ›

T

Theodore Petrou → Chris Moffitt

6 years ago

No. You can get the figure from ax.figure.

0 0 Reply • Share ›

Chris Moffitt Mod → Theodore Petrou
6 years ago

Ah. Good to know. Thanks for clarifying.

1 0 Reply • Share ›

陈俊杰
6 years ago

very helpful. But how can I plot the chart like below with matplotlib? View – uploads.disquscdn.com

1 2 Reply • Share ›

Lincoln Frias
7 years ago

Awesome, very helpful. Thanks!

1 2 Reply • Share ›

Priyanshu Sahu
a year ago

Great Post

0 0 Reply • Share ›

janmeppe
3 years ago

This post is amazing. Matplotlib is very confusing for beginners. Thank you for sharing! Small typo "indivudual" btw!

0 0 Reply • Share ›

Chris Moffitt Mod → janmeppe
3 years ago

Glad you found it useful. Thanks for pointing out the typo. I'll fix it. Thanks!

0 0 Reply • Share ›

Iteml
3 years ago

I have a question. If I run the following code (on Julyter Notebook):

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

%matplotlib inline

df = pd.read_excel("https://github.com/chris161...")
```

```
top_10 = (df.groupby('name')['ext price', 'quantity'].agg({'ext price': 'sum', 'quantity':
'count'}).sort_values(by='ext price', ascending=False))[:10].reset_index()
top_10.rename(columns={'name': 'Name', 'ext price': 'Sales', 'quantity': 'Purchases'},
inplace=True)

fig, ax = plt.subplots()
ax.set_xlim([-10000, 140000])
ax.set_xlabel('Total Revenue')
ax.set_ylabel('Customer')
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
```

see more

0 0 Reply • Share ›

Chris Moffitt Mod → Iteml

3 years ago

The easiest way to think about it is that the Figure is a container that has one or more Axes. If there is only one Axes then it is a 1:1 relationship. However you can have many Axes within a single Figure.

So, when you type 'fig' you are displaying the top level figure that contains the top_10 plot.

Does that help?

0 0 Reply • Share ›

L **Iteml** → Chris Moffitt

3 years ago

Yes, that makes sense from a conceptual point of view!

But from a Python code point of view, I don't find it intuitive. Usually the arguments that are used when I call an object's method do not modify the arguments themselves.

0 0 Reply • Share ›

L **Iteml**

4 years ago

Thanks for this post. It was very helpful. I was also VERY confused because of the mishmash of the object-oriented API Vs the state-based.

I have one question: when I have the code below and run it on Jupyter Notebook:

```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
```

```
ax.set_xlabel('Total Revenue')
ax.set_ylabel('Customer');
```

why is this actually showing the figure? There is no "print" or "show" or similar at the end of the block.

0 0 Reply • Share ›

Chris Moffitt Mod → Iteml

4 years ago

Glad you found the article helpful.

I imagine you have a line of code (maybe at the top of your notebook) that says %matplotlib inline

This is a "magic command" and ensures that matplotlib figures are displayed in a notebook even if you don't use show.

This is very similar to viewing a variable in a a notebook. If you type df.head() you can see the output without explicitly using print. Hope that helps.

0 0 Reply • Share ›

L

Iteml → Chris Moffitt

4 years ago

Hi Chris,
I actually do not have the magic command for %matplotlib inline. I started a brand new kernel from scratch with just the following code:

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

top_10 = (df.groupby('name')['ext price', 'quantity'].agg({'ext price': 'sum',
'quantity': 'count'}).sort_values(by='ext price', ascending=False))
[:10].reset_index()
top_10.rename(columns={'name': 'Name', 'ext price': 'Sales', 'quantity':
'Purchases'}, inplace=True)

fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set_xlabel('Total Revenue')
ax.set_ylabel('Customer');
```

Even with the semicolon (which usually prevents Jupyter Notebook from evaluating expressions unless explicitly states) the plot shows when I run the block. I am puzzled.

0 0 Reply • Share ›

Chris Moffitt Mod ↗ Iteml

4 years ago

FYI - I tweeted about this and sure enough a recent change to Jupyter notebook means you don't need to use %matplotlib inline to display plots! News to me so thanks for bringing this up.

0 0 Reply • Share ›

L

Iteml ↗ Chris Moffitt

4 years ago

:)

0 0 Reply • Share ›

Chris Moffitt Mod ↗ Iteml

4 years ago

Interesting. I just tried something similar in a new notebook and I did not need to use %matplotlib inline either. It just displayed the output. Something must have changed somewhere with notebooks. I'll do some investigation.

0 0 Reply • Share ›

N

N Srinivasan

4 years ago

An excellent article! Clarifies a lot of confusion! Thanks!!

0 0 Reply • Share ›

X

Xian Liu

5 years ago

Cool. I wish I could have read this post before using Matplotlib. So much time did I spend on google.

0 0 Reply • Share ›

S

Sander van den Oord

6 years ago

Great article! Just one addition. I don't really like the funcformatter, it's hard to remember, so to change the ticklabels I usually use (as an example):
ax.set_yticklabels(u'{:0.0f}%'.format(ytick) for ytick in ax.get_yticks())

0 0 Reply • Share ›

T

Tebogo Mogaleemang

6 years ago

This is a great post. It has put a lot of issues into proper context for me. Thanks!

0 0 Reply • Share ›

Пёрдни в Ухо
6 years ago edited

Hello. I want to translate this article in Russian and posted in my blog with link to your site. Ok?

0 0 Reply • Share ›

Chris Moffitt Mod → Пёрдни в Ухо
6 years ago

Yes. Absolutely, please feel free to translate and link back to the original.

Thanks!

0 0 Reply • Share ›

Carlos
7 years ago

How to run de code?
I writed:

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter

df = pd.read_excel("sample-salesv3.xlsx")
df.head()

top_10 = (df.groupby('name')['ext price', 'quantity'].agg({'ext price': 'sum', 'quantity': 'count'})
.sort_values(by='ext price', ascending=False))[:10].reset_index()
top_10.rename(columns={'name': 'Name', 'ext price': 'Sales', 'quantity': 'Purchases'},
inplace=True)
plt.style.use('ggplot')
top_10.plot(kind='barh', y="Sales", x="Name")
```

But only return :
Process finished with exit code 0

0 0 Reply • Share ›

Chris Moffitt Mod → Carlos
7 years ago

You need to run this in a jupyter notebook and include %matplotlib inline or add plt.show() at the end of your script.

Subscribe to the mailing list

Email address

Subscribe

Social

Github
Twitter
LinkedIn

Submit a Topic

Suggest a topic for a post

Popular

Pandas Pivot Table Explained
Common Excel Tasks Demonstrated in Pandas
Overview of Python Visualization Tools
Guide to Encoding Categorical Values in Python
Overview of Pandas Data Types

Article Roadmap

Feeds

Atom Feed

Disclosure

We are a participant in the Amazon Services LLC Associates Program, an affiliate advertising program designed to provide a means for us to earn fees by linking to Amazon.com and affiliated sites.