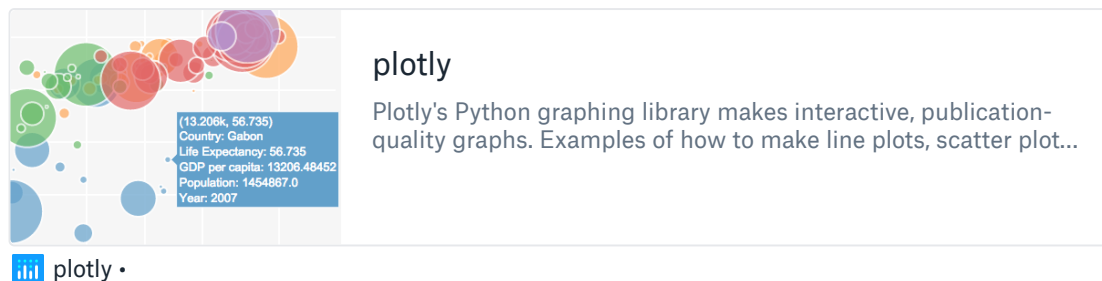


Graphing and Mapping with Plot.ly

Plot.ly is a graphing service that you can work with from Python. It allows you to create many different kinds of graphs, including interactive graphs. It is extremely powerful and supports pretty sophisticated graphs.

Previously, plot.ly only worked in an online mode, where your graphs would automatically get uploaded into your plot.ly account, but since the release of plot.ly 4 in the summer of 2019, you can create plot.ly graphs completely locally.

Plot.ly supports a ton of different graph types. Let's take a look:



Let's play around with some simple charts to get a sense of what plotly does.

Working with Plotly from Python

Plotly supports graphing from a number of programming languages, including python.

To be able to use Plotly from your python programs, you will need to install the plotly module:

```
pip3 install plotly.
```

If you ever want to use Plotly from Jupyter Notebook, the basic directions for setting that up are found here: <https://plot.ly/python/getting-started/>

OK, let's create a very simple bar graph with plotly. Create a new python file with the following three lines:

```
1 import plotly.graph_objects as go
2 fig = go.Figure(data=go.Bar(y=[2, 3, 1]))
3 fig.write_html('first_figure.html', auto_open=True)
```

Run the program. If all goes well, plotly created an html file in your working directory containing the graph and automatically opened it in your browser. Kinda cool, ha?

One more example of basic plotting functionality. For this, you'll need to install numpy:

```
pip3 install numpy
```

Here is the code:

```
1 import numpy as np
2 import plotly.graph_objs as go
3
4 # Create random data with numpy
5 N = 1000
6 random_x = np.random.randn(N)
7 random_y = np.random.randn(N)
8
9 # Create a trace
10 trace = go.Scatter(
11     x = random_x,
12     y = random_y,
13     mode = 'markers'
14 )
15 data = [trace]
16 fig = go.Figure(data=data)
17 fig.write_html('scatter.html', auto_open=True)
18
```

A note on the syntax here. We could have written this program slightly more concisely like this:

```
1 import numpy as np
2 import plotly.graph_objs as go
3
4 # Create random data with numpy
5 N = 1000
6 random_x = np.random.randn(N)
7 random_y = np.random.randn(N)
8
9 fig = go.Figure(data=go.Scatter(x=random_x, y=random_y, mode="markers"))
10 fig.write_html('scatter.html', auto_open=True)
```

The reason we used the longer form is that the specification of the trace (the thing that your Scatter, Bar, and other types of graph objects return), can get pretty complicated. It can be easier to keep track of all the arguments that need to be passed by pulling that part of the code out.

Let's deconstruct what's going on here, since most of the plots are made in a very similar way:

- The `graph_objs` module which we import at the start of the file contains classes for different kinds of graphs that plotly knows how to make.
- You create a graph trace—the information that will be graphed—by instantiating a particular graph type (scatter, bar, etc.) with arguments that include the data that you want plotted and specification of how that data should be displayed.
 - For instance, in our scatter plot, we told it to use markers as the plotting “mode.” Most graph types have a lot of different parameters you can set to tell plotly exactly how you want your data to be graphed.
- You then create a Figure object by passing it the trace as an argument. The figure object represents the actual graph that is generated.
 - For some graph types, the figure object will also need to be provided with a layout, which tells plotly how exactly to display this graph. We'll see how to do that in just a bit.
- you can then use display this figure object in different ways, including, as we have been doing here, by saving it as an html that is opened in the browser.
 - If you don't need to permanently store the graph (in html, in our examples above), you could replace the last line in the two programs

above with just this: `fig.show()` That will still open the browser to show you an interactive graph, but it won't save it permanently.

Tutorials for creating other types of graphs using python are here:

<https://plot.ly/python/>

Using Plotly Maps

For the rest of the class, we'll focus on Plotly's mapping functionality, since this is what you need both for Project 2 and for your homework this week.

From the python tutorial page, go to the Maps section, "More Maps" > "Scatter Plots on Maps"

Since the tutorial is written for pandas users (pandas is a python module for doing high-performance data analytics), we will modify the code slightly to not include this dependency.

First, you need to get the csv file used in the tutorial onto your local machine. You can download the file [here](#). Put it in a new directory and create a new python file (say, called airports.py)

Documentation for the python csv module:

<https://docs.python.org/3/library/csv.html>

To understand how plotly maps work, let's start at the end and work backwards.

Here's the punchline from the tutorial:

```
1 fig = go.Figure(data=go.Scattergeo(  
2     lon = df['long'],  
3     lat = df['lat'],  
4     text = df['text'],  
5     mode = 'markers',  
6     marker_color = df['cnt'],  
7 ))  
8  
9 fig.update_layout(  

```

```
10     title = 'Most trafficked US airports<br>(Hover for airp  
ort names)',  
11     geo_scope='usa',  
12 )  
13 fig.show()
```

- What we see here is that create a figure by passing it, as a data argument, trace generated by instantiating a Scattergeo graph type.
 - This object itself is created by passing it the information for coordinates that need to mapped, and information for how those points should be displayed.
- We then specify the layout of the graph by invoking the `update_layout()` method of our figure object. The layout specifies how the graph overall should be formatted.
- We then display the figure.

There is a cheat sheet for how these different elements need to be specified for different types of plotly graphs. You can find it here:

https://images.plot.ly/plotly-documentation/images/python_cheat_sheet.pdf

Ok, let's rewrite the tutorial program without using pandas:
First, we need the data:

```
1  import plotly.graph_objs as go  
2  import csv  
3  
4  f = open('2011_february_us_airport_traffic.csv')  
5  csv_data = csv.reader(f)  
6  
7  lat_vals = []  
8  lon_vals = []  
9  text_vals = []  
10 for row in csv_data:  
11     if row[0] != 'iata':  
12         lat_vals.append(row[5])  
13         lon_vals.append(row[6])
```

```
14     text_vals.append(row[0])
```

Now, we'll rewrite the graphing code to use the data lists we read in from the csv file:

```
1  fig = go.Figure(data=go.Scattergeo(  
2      lon = lon_vals,  
3      lat = lat_vals,  
4      text = text_vals,  
5      mode = 'markers',  
6      marker_color = 'red',  
7  ))
```

Now we can put everything together and graph the airports:

```
1  import plotly.graph_objs as go  
2  import csv  
3  
4  f = open('2011_february_us_airport_traffic.csv')  
5  csv_data = csv.reader(f)  
6  
7  lat_vals = []  
8  lon_vals = []  
9  text_vals = []  
10 for row in csv_data:  
11     if row[0] != 'iata':  
12         lat_vals.append(row[5])  
13         lon_vals.append(row[6])  
14         text_vals.append(row[0])  
15 fig = go.Figure(data=go.Scattergeo(  
16     lon = lon_vals,  
17     lat = lat_vals,  
18     text = text_vals,  
19     mode = 'markers',  
20     marker_color = 'red',  
21 ))
```

```

22
23 fig.update_layout(
24     title = 'Most trafficked US airports<br>(Hover for
airport names)',
25     geo_scope='usa',
26 )
27 fig.show()

```

Scaling and centering the map

- Let's work with a subset of the data. Let's look only at California airports. Here is the csv file containing just the CA airports: [CA-airports.csv](#).
- Change your program to open the new CSV instead of the old one, and save it as "ca-airports.py"
- Run it.
- it works, but we are wasting a lot of space on showing the whole country when our data is just for California.

Can we get the map to show just California?

We will need to set the "range" for the map coordinates, so the map only shows the part we are interested in. We can do this by finding the extremes of our data. We'll need minimum and maximum longs and lats in our data. We can find these manually (you learned how to do that in 506!), or we can just use the `min()` and `max()` functions that operate on lists. Let's just do that for conciseness:

```

1 min_lat = float(min(lat_vals))
2 max_lat = float(max(lat_vals))
3 min_lon = float(min(lon_vals))
4 max_lon = float(max(lon_vals))

```

(we have to convert the min and max values to floats since they are read from the file as strings.)

Now we need to use this information to set the range of the axes. We do this by adding parameters to our `layout` object. Plotly wants this as a list:

```

1 lat_axis = [min_lat, max_lat]
2 lon_axis = [max_lon, min_lon]

```

Which we will now add, along some other formatting, to the `layout` object we will use to create our map:

```

1 layout = dict(
2     title = 'California airports<br>(Hover for airport
names)',
3     geo = dict(
4         scope='usa',
5         projection=dict( type='albers usa' ),
6         showland = True,
7         landcolor = "rgb(250, 250, 250)",
8         subunitcolor = "rgb(100, 217, 217)",
9         countrycolor = "rgb(217, 100, 217)",
10        lataxis = {'range': lat_axis},
11        lonaxis = {'range': lon_axis},
12        countrywidth = 3,
13        subunitwidth = 3
14    ))

```

now we just need to update the figure layout by passing it this layout object:

```

1 fig.update_layout(layout)
2 fig.show()

```

Here is the program with these changes:

```

1 import plotly.graph_objs as go
2 import csv
3
4 f = open('CA-airports.csv')
5 csv_data = csv.reader(f)
6
7 lat_vals = []
8 lon_vals = []
9 text_vals = []
10 for row in csv_data:
11     if row[0] != 'iata':
12         lat_vals.append(row[5])
13         lon_vals.append(row[6])

```



```
14         text_vals.append(row[0])
15
16 min_lat = float(min(lat_vals))
17 max_lat = float(max(lat_vals))
18 min_lon = float(min(lon_vals))
19 max_lon = float(max(lon_vals))
20
21 lat_axis = [min_lat, max_lat]
22 lon_axis = [max_lon, min_lon]
23
24 layout = dict(
25     title = 'California airports<br>(Hover for airport name
26 s)',
27     geo = dict(
28         scope='usa',
29         projection=dict( type='albers usa' ),
30         showland = True,
31         landcolor = "rgb(250, 250, 250)",
32         subunitcolor = "rgb(100, 217, 217)",
33         countrycolor = "rgb(217, 100, 217)",
34         lataxis = {'range': lat_axis},
35         lonaxis = {'range': lon_axis},
36         countrywidth = 3,
37         subunitwidth = 3
38     ))
39
40 fig = go.Figure(data=go.Scattergeo(
41     lon = lon_vals,
42     lat = lat_vals,
43     text = text_vals,
44     mode = 'markers',
45     marker_color = 'red',
46 ))
```

```

47 fig.update_layout(layout)
48 fig.show()

```

- Running the program, we see things are a little weird.
- We changed the map scale, but our data is not in frame. Turns out we need to center the map as well. To do this, we need to add the "center" key to the layout object, which will be focused on the middle point of our data:

```

1 center_lat = (min_lat + max_lat) / 2
2 center_lon = (min_lon + max_lon) / 2

```

And we now add update the layout object to include the center paramter:

```

1 layout = dict(
2     title = 'US airports<br>(Hover for airport names)',
3     geo = dict(
4         scope='usa',
5         projection=dict( type='albers usa' ),
6         showland = True,
7         landcolor = "rgb(250, 250, 250)",
8         subunitcolor = "rgb(100, 217, 217)",
9         countrycolor = "rgb(217, 100, 217)",
10        lataxis = {'range': lat_axis},
11        lonaxis = {'range': lon_axis},
12        center= {'lat': center_lat, 'lon': center_lon
13    },
14        countrywidth = 3,
15        subunitwidth = 3
16    ),
17 )

```

Looking better:

One thing though: our map feels a little crowded. The extreme values are right at the edge of the map, and not even fully displayed. Let's add a bit of padding. Here's a super simple way to do that (we could get fancier as well, but this will do for our purposes):

- We know that longitudes range from -180 degrees to 180 degrees, and that latitudes range from -90 degrees to 90 degrees.
- Given that all airports are relatively close, we can probably just add 1 degree of space in each direction and be fine.
- To do this, we just need to pass a slightly modified range to `layout`

```
1 lat_axis = [min_lat - 1, max_lat + 1]
2 lon_axis = [min_lon - 1, max_lon + 1]
```

this will give us something like this:

That looks better.

Here's that whole program:

```
1 import plotly.graph_objs as go
2 import csv
3
4 f = open('CA-airports.csv')
5 csv_data = csv.reader(f)
6
7 lat_vals = []
8 lon_vals = []
9 text_vals = []
10 for row in csv_data:
11     if row[0] != 'iata':
12         lat_vals.append(row[5])
13         lon_vals.append(row[6])
14         text_vals.append(row[0])
15
16 min_lat = float(min(lat_vals))
17 max_lat = float(max(lat_vals))
```

```
18 min_lon = float(min(lon_vals))
19 max_lon = float(max(lon_vals))
20
21 lat_axis = [min_lat -1, max_lat + 1]
22 lon_axis = [max_lon + 1, min_lon -1]
23
24 center_lat = (min_lat + max_lat) / 2
25 center_lon = (min_lon + max_lon) / 2
26
27
28 layout = dict(
29     title = 'California airports<br>(Hover for airport name
30 s)',
31     geo = dict(
32         scope='usa',
33         projection=dict( type='albers usa' ),
34         showland = True,
35         landcolor = "rgb(250, 250, 250)",
36         subunitcolor = "rgb(100, 217, 217)",
37         countrycolor = "rgb(217, 100, 217)",
38         lataxis = {'range': lat_axis},
39         lonaxis = {'range': lon_axis},
40         center= {'lat': center_lat, 'lon': center_lon },
41         countrywidth = 3,
42         subunitwidth = 3
43     ))
44
45 fig = go.Figure(data=go.Scattergeo(
46     lon = lon_vals,
47     lat = lat_vals,
48     text = text_vals,
49     mode = 'markers',
50     marker_color = 'red',
51 ))
```

```
51  
52 fig.update_layout(layout)  
53 fig.show()
```

Now let's see how we would plot 2 different types of data on the map

Let's go back to our program that maps all U.S. airports

Now, let's split our data into two sets by separating "big" airports from "small" ones.

We'll take this part of the code:

```
1 lat_vals = []  
2 lon_vals = []  
3 text_vals = []  
4 for row in csv_data:  
5     if row[0] != 'iata':  
6         lat_vals.append(row[5])  
7         lon_vals.append(row[6])  
8         text_vals.append(row[0])
```

and turn it into this:

```
1 big_lat_vals = []  
2 big_lon_vals = []  
3 big_text_vals = []  
4 small_lat_vals = []  
5 small_lon_vals = []  
6 small_text_vals = []  
7 for row in csv_data:  
8     if row[0] != 'iata':  
9         traffic = int(row[7])  
10        lat = row[5]  
11        lon = row[6]  
12        text = row[0]  
13        if traffic > 1000:  
14            big_lat_vals.append(row[5])  
15            big_lon_vals.append(row[6])
```

```
16         big_text_vals.append(row[0])
17     else:
18         small_lat_vals.append(row[5])
19         small_lon_vals.append(row[6])
20         small_text_vals.append(row[0])
```

And now we need to create a data object (that we will pass to the figure) that contains two traces instead of one—one for big airports and one for small airports.

Note that each of the traces will be a dictionary (since the content of the data lists are dictionaries). We will get something like this:

```
1  trace1 = dict(
2      type = 'scattergeo',
3      locationmode = 'USA-states',
4      lon = big_lon_vals,
5      lat = big_lat_vals,
6      text = big_text_vals,
7      mode = 'markers',
8      marker = dict(
9          size = 15,
10         symbol = 'star',
11         color = 'red'
12     ))
13 trace2 = dict(
14     type = 'scattergeo',
15     locationmode = 'USA-states',
16     lon = small_lon_vals,
17     lat = small_lat_vals,
18     text = small_text_vals,
19     mode = 'markers',
20     marker = dict(
21         size = 8,
22         symbol = 'circle',
```

```
23         color = 'blue'
24     ))
25
26 data = [trace1, trace2]
```

Now we just need to pass this object as an argument when we create our figure:

```
fig = go.Figure(data=data)
```

Now try running it!

Boom.

Here's that whole program:

```
1  import plotly.graph_objs as go
2  import csv
3
4  f = open('2011_february_us_airport_traffic.csv')
5  csv_data = csv.reader(f)
6
7  big_lat_vals = []
8  big_lon_vals = []
9  big_text_vals = []
10 small_lat_vals = []
11 small_lon_vals = []
12 small_text_vals = []
13 for row in csv_data:
14     if row[0] != 'iata':
15         traffic = int(row[7])
16         lat = row[5]
17         lon = row[6]
18         text = row[0]
```

```
19         if traffic > 1000:
20             big_lat_vals.append(row[5])
21             big_lon_vals.append(row[6])
22             big_text_vals.append(row[0])
23         else:
24             small_lat_vals.append(row[5])
25             small_lon_vals.append(row[6])
26             small_text_vals.append(row[0])
27
28 trace1 = dict(
29     type = 'scattergeo',
30     locationmode = 'USA-states',
31     lon = big_lon_vals,
32     lat = big_lat_vals,
33     text = big_text_vals,
34     mode = 'markers',
35     marker = dict(
36         size = 15,
37         symbol = 'star',
38         color = 'red'
39     ))
40 trace2 = dict(
41     type = 'scattergeo',
42     locationmode = 'USA-states',
43     lon = small_lon_vals,
44     lat = small_lat_vals,
45     text = small_text_vals,
46     mode = 'markers',
47     marker = dict(
48         size = 8,
49         symbol = 'circle',
50         color = 'blue'
51     ))
```



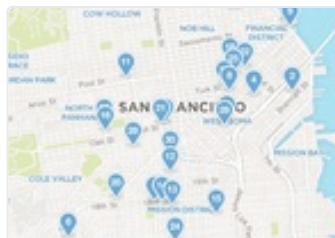
```

52
53 data = [trace1, trace2]
54
55 fig = go.Figure(data=data)
56
57 fig.update_layout(
58     title = 'Most trafficked US airports<br>(Hover for airport names)',
59     geo_scope='usa',
60 )
61 fig.show()

```


Mapping with Mapbox

Mapbox is map data provider that allows us to graph data onto maps that have a lot more detail (e.g., streets, lakes, etc.) than is possible with default Plotly maps. Mapbox is used in a very similar way to the default maps. Directions for how to use it can be found [here](#):



Scatter Plots on Mapbox

How to make scatter plots on Mapbox maps in Python.

 Scatter Plots on Mapbox •

Here we are going to just modify our original airports map program to get it to work with Mapbox.

To do this:

- We will need a Mapbox API key. You can get it from [this file](#), which you will also use for your homework this week. We can import the information by adding `from secret import *` to the start of our file
- We will need to change the “**type**” key in the data dictionary from “scattergeo” to “scattermapbox”
- And we’ll need to modify the layout object a little. This is how it’s going to look:

```

1 layout = dict(

```

```

2         title = 'US airports on Mapbox<br>(Hover for airport names)',
3         autosize=True,
4         showlegend = False,
5         mapbox=dict(
6             accesstoken=MAPBOX_TOKEN,
7             bearing=0,
8             center=dict(
9                 lat=38,
10                lon=-94
11            ),
12            pitch=0,
13            zoom=3,
14        ),
15    )

```

- finally, we need to pass this layout object to our figure:
`fig.update_layout(layout)`
- The key thing to pay attention to here is the “mapbox” dictionary that is an element of the layout object. This is where Mapbox specific stuff needs to go. Everything outside of that dictionary are standard elements of Plotly’s layout objects, that are documented in Plotly’s reference guide. The stuff in the “mapbox” dictionary, however, is how you send information to the Mapbox API to get it to format the map the way you want it to look.
 - Note the “center” key. We are currently positioning the center of the map to see the whole of the United States, but you could also calculate the center of the map, like we did with the California airports.
 - The “zoom” key tells Mapbox how much to zoom in to the map. If you took it out completely, it would show you the whole world. The bigger the zoom, the more it zooms in. 4 zooms in too much to see the whole country, and 2 is a little too wide (try it). Hence 3 in our code.
 - You can see the meaning of the other options, such as pitch, in the Mapbox documentation: <https://www.mapbox.com/mapbox-gl-js/style-spec/>

Everything else stays the same as in our original airports file. When we now run it, we get this:

Notice that when you zoom in, you can get all the way down to the street level.

Here is the complete code for the Mapbox example:

```
1 from secret import *
2 import plotly.graph_objs as go
3 import csv
4
5 f = open('2011_february_us_airport_traffic.csv')
6 csv_data = csv.reader(f)
7
8 lat_vals = []
9 lon_vals = []
10 text_vals = []
11 for row in csv_data:
12     if row[0] != 'iata':
13         lat_vals.append(row[5])
14         lon_vals.append(row[6])
15         text_vals.append(row[0])
16
17 layout = dict(
18     title = 'US airports on Mapbox<br>(Hover for airport na
19     mes)',
20     autosize=True,
21     showlegend = False,
22     mapbox=dict(
23         accesstoken=MAPBOX_TOKEN,
24         bearing=0,
25         center=dict(
26             lat=38,
27             lon=-94
28         ),
29         pitch=0,
30         zoom=3,
```

```
30     ),
31 )
32
33 fig = go.Figure(data=go.Scattermapbox(
34     lon = lon_vals,
35     lat = lat_vals,
36     text = text_vals,
37     mode = 'markers',
38     marker_color = 'red',
39 ))
40
41 fig.update_layout(layout)
42 fig.show()
```

That's it! If there's time left, work on Project 2 or your individual project and ask for help if you get stuck!

Mapbox example with both big and small airports:



bigsmall
源代码

<https://www.dropbox.com/s/qo4qasj3dc01dh/bigsmall.py?dl=0>

These are very useful documents. For some reason they are really hard to find on the plotly documentation site:

- <https://plot.ly/python/user-guide/>
- <https://plot.ly/python/reference/>

These are the pages we looked at in class:

- <https://plot.ly/python/getting-started/>
- <https://plot.ly/python/>
- <https://plot.ly/python/bar-charts/>
- <https://plot.ly/python/scatter-plots-on-maps/>
- <https://plot.ly/python/scattermapbox/>
- https://images.plot.ly/plotly-documentation/images/python_cheat_sheet.pdf

- <https://www.mapbox.com/mapbox-gl-js/style-spec/>