PIC 16, Winter 2018 – Preparation 8M

Assigned 2/21/2018. To be completed by class 2/26/2018.

Intended Learning Outcomes

By the end of this preparatory assignment, students should be able to:

- use Plotly to produce simple Box, Scatter, and Histogram plots;
- publish plots to the web and local files;
- strip data from existing Plotly plots;
- adjust the appearance of plots as desired using online editing tools;
- learn, as needed, how to edit the appearance of plots programmatically;
- refer to the Plotly Python Library for plotting solutions.

Tasks

- ☐ Follow Getting Started with Plotly for Python. You can skip "Online Plot Privacy" with the understanding that all your online plots will be publicly accessible (unless you pay for an account). You can also skip the Special Instructions for Plotly On-Premise Users. Try running the example code for creating both online and offline plots. I recommend using Jupyter Notebook so that plots display inline rather than in your browser. Note: when installing, try using `conda install plotly` rather than `pip install plotly`. This will ensure that Plotly is set up for your Anaconda distribution of Python rather than another distribution on your computer. This is especially important for OS X users.
- ☐ Plotly servers only allow a certain number of API calls (plot generations) per day. Try spreading this work over a few days so you don't reach the limit.
- ☐ Follow the Plotly User Guide in Python. Again, using a Jupyter notebook is more convenient because the plots will appear inline.
- ☐ I find the documentation difficult to follow. Maybe it's something about the formatting of the website. Maybe it's because the examples look so long. Maybe the organization is poor. To help myself keep things straight and maybe to help you, here's my attempt at a tutorial.
    - Plotly is used for creating attractive, interactive "Figures" or "Plots". These Figures appear as charts/graphs in a web browser.
    - Plotly is somewhat unusual in that these Figures are typically generated by and stored on Plotly's servers (online). You can view and modify them by logging into Plotly's web interface. Plotly also generates a URL that anyone can use to view your public plots. With a free account, you can only keep one private plot, so this means that most of your plots are publicly accessible.
    - We will be generating Figures using Python. In Python code, Figures are represented as instances of the `Figure` class, which you can `import` from the `plotly.graph_objs` module.
    - There are two important functions for making a `Figure` object appear as a plot in your web browser.
        - `iplot` is for use within a Jupyter Notebook and makes the plot appear inline.
        - `plot` is for use elsewhere (e.g. Spyder) and makes the plot appear in a browser window.

      Both of these functions can be imported from the `plotly.plotly` module for online plotting or the `plotly.offline` module for offline plotting. The arguments they require are the same:
        - the first argument is the `Figure` object to be shown, and

- you should also specify a value for the `filename` parameter, which is the name you will use to refer to the Figure in Plotly's web interface. Each figure with a unique `filename` gets a unique URL. If you re-plot a figure with the same name, it keeps the same URL; you've just changed the plot that is there.

Assuming you `import plotly.plotly as py`, you can call `plot`/`iplot` like: `py.plot(fig, filename = "MyPlot")`

- Of course, before you do that, you'll beed to have a `Figure` object to plot. A `Figure` object is composed of two parts:
  - The Data – all the information you are trying to present graphically, and
  - The Layout – specifications about the figure independent of the data.

  In Python, you represent your Data using an instance of the `Data` class and your Layout using an instance of the `Layout` class, both of which you can `import` from the `plotly.graph_objs` module. In fact, all the other classes we're going to look at in the tutorial are "Graph Objects", so all of them are imported from `plotly.graph_objs`.
- A `Layout` object is similar to a Python dictionary. It has key/value pairs that specify different aspects of the Figure's Layout. `Layout` objects are like dictionaries that only understand certain keys, and each key expects a certain type of value. Here are some keys and notes about acceptable values that you should memorize:
  - `title` – For this key, the value must be a string to appear at the top of the plot.
  - `xaxis`, `yaxis` – A single string does not suffice, because there are many aspects of the x- and y- axes that you can control using this key. Acceptable values are a `XAxis`/`YAxis` object or equivalent Python dictionary. This allows you to specify not only the label that will appear next to each axis (the axis `title`) but options about the plot's grid, etc…

  An example Layout is:

```
layout = Layout(title = "Hello",
                xaxis = XAxis(title = "WattupX"),
                yaxis = {title: "WattupY", showLegend = True})
```

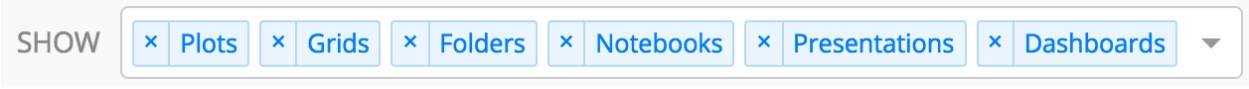
  Note a few things:
  - The `Layout` object constructor is used by naming parameters and assigning values. The resulting `Layout` object is much like a dictionary with key/value pairs and actually can be replaced by a regular dictionary with the appropriate key names and value types.
  - After you've created a Layout, you can access its keys as you would those of a regular dictionary, like `layout["title"] = "Hello2".` Unlike a regular dictionary, you can also access them with a dot notation: `layout.title = "Hello3"`. This is also true of other dictionary-like graph objects.
  - There are many other keys/parameters you can set besides those shown here. You don't need to memorize them all. When you need to do something you don't yet know how to do, do some research online. Find a relevant example in Plotly's documentation. Or, modify your plot using Plotly's web interface. Then you can see the keys/values it used to make the changes by viewing the code/JSON used to generate the plot.
  - The `XAxis` object is also a dictionary-like object, and the initializer is used in basically the same fashion as the `Layout` constructor: by naming parameters and assigning values. See the documentation for acceptable keys and their corresponding value types.
  - Instead of creating a `YAxis` object, we have substituted an appropriate dictionary. You don't need to use Plotly graph objects since they're basically just lists or dictionaries. They just add some convenient features.

You will find that there are many dictionary-like classes (such as `Layout`, `XAxis`, and `YAxis`) in `plotly.graph_objs`. You can instantiate and initialize them all in the same way: by naming parameters and assigning values. Equivalently, you can create a regular Python dictionary yourself and use that in place of the object. You just need to know the names of the keys and the appropriate types of values.

- A `Data` object is similar to a Python *list*, not a dictionary. This list-like object contains several Traces. Let's talk about Traces, then come back to `Data`.
- A Trace is a set of data to be represented in a plot along with some specifications for how it should appear. Traces are represented by dictionary-like objects such as `Scatter` (for scatter plots), `Histogram` (for histograms), and `Bar` (for bar charts). You already know how to create dictionary-like objects from `plotly.graph_objs`. You just need to know what keys and values are appropriate for Traces. The most common are:
  - `x` – a Python list of datum x-coordinates,
  - `y` – a Python list of datum y-coordinates,
  - `name` – a string describing the trace; this will appear in the plot's legend, and
  - `marker` – a dictionary with keys like `color`, `symbol`, and `size` that describe the symbol used to represent each datum at its coordinates.

  See the examples in the Plotly documentation to learn about other key/value pairs. An example Trace is:
  ```
  trace = Scatter(x=[1,2,3], y=[4,5,6], marker={'color': 'red',
  'symbol': 'x', 'size': "10"}, name= 'Series 1')
  ```
- A `Data` object is a list-like object containing Traces. If you only have one trace, like one `Scatter` object, you initialize the `Data` object with a Python list containing only one element, your `Scatter` object. The nice thing about `Data` being list-like is that you can create plots with several kinds of traces on it at once, like a scatter plot over a bar chart. The `Data` object would simply be initialized with a list containing both a `Scatter` Trace and a `Bar` Trace. For instance, a `Data` object can be created like:
  ```
  data=Data([scatter_trace, bar_trace])
  ```
- Finally, we create our `Figure` object, mentioned earlier. A `Figure` object is dictionary-like, and it expects keys `data` and `layout`.
  ```
  fig = Figure(data = data, layout = layout)
  ```
  Incidentally, we've named our `Data` and `Layout` objects the same as the keys of the dictionary, hence `data` (the key) `= data` (the object). Now we can `plot` the figure.

- ☐ You might also find the Preparation 17 Notebook on CCLE helpful. Open it in Jupyter.
- ☐ Visit your Plotly account online and go to "My Files". Find a file you have worked on in the tutorial and select "Edit Graph". Alternatively, click "Edit Chart" at the bottom right of one of your plots in Jupyter. Try to figure out how to change the title, axis labels, grid lines, and colors. When you're done, you can "View JSON" to see the attributes you've changed, which will allow you to make the same changes in Python code. For the most part, the user interface is pretty intuitive, so the best way to learn these features is by trial and error, but please ask for help in lecture if you get stuck!
- ☐ Something I find confusing about "My Files" online is that it shows two entries for every plot you have created: one is the plot itself, and the other with "Grid" at the end of the name containing, which contains only the data (no graph). You can show only Plots by de-selecting everything else in the control at the top that looks like:



- ☐ Take a look at the solutions available in the Plotly Python Library Examples. In the future, this is a good place to start when you need to create a plot!

- A few useful functions that are not included in the tutorial (but are demonstrated in the notebook):
  - `plotly.plotly.get_figure(user_name, plot_number)` returns a `Figure` object given a Plotly user's username as a string and the unique number associated with their plot, both of which can be gleaned from a plot's URL. For instance, one of my plots is available at:
    https://plot.ly/~mdhaber/74/plot-update/
    My username is `mdhaber` and the plot's number is `74`, so I can get the entire `Figure` object representing the Figure like:
    ```
    fig = py.get_figure("mdhaber",74)
    ```
  - `Figure` objects have methods. One of them is `get_data()`, and it returns the Figure's `Data` object. Combined with the `get_figure` function, if you know a Figure's URL, you can get the raw data used to produce it.
  - The `plotly.plotly.image.save_as` function can be used to save a Figure as an image file, like a `.png`. Note: Image file formats are not created equal. `.png` is typically a far more appropriate format for saving graphs than `.jpg` (whereas `.jpg`s are far better for photos.) Please use `.png` for plots in this class.
  - See Get Requests in Python and Static Image Export as needed for more information about these features.