Beginner's Python Cheat Sheet - Plotly

What is Plotly?

Data visualization involves exploring data through visual representations. Plotly helps you make visually appealing representations of the data you're working with. Plotly is particularly well suited for visualizations that will be presented online, because it supports interactive elements.

Installing Plotly

Plotly runs on all systems, and can be installed in one line.

Installing Plotly

```
$ python -m pip install --user plotly
```

Line graphs, scatter plots, and bar graphs

To make a plot with Plotly, you specify the data and then pass it to a graph object. The data is stored in a list, so you can add as much data as you want to any graph. In offline mode, the output should open automatically in a browser window.

Making a line graph

A line graph is a scatter plot where the points are connected. Plotly generates JavaScript code to render the plot file. If you're curious to see the code, open the squares.html file in a text editor after running this program.

```
from plotly.graph_objs import Scatter
from plotly import offline

# Define the data.
x_values = list(range(11))
squares = [x**2 for x in x_values]

# Pass the data to a graph object, and store it
#   in a list.
data = [Scatter(x=x_values, y=squares)]

# Pass the data and a filename to plot().
offline.plot(data, filename='squares.html')
```

Making a scatter plot

To make a scatter plot, use the `mode='markers'` argument to tell Plotly to only display the markers.

```
data = [Scatter(x=x_values, y=squares,
        mode='markers')]
```

Line graphs, scatter plots, and bar graphs (cont.)

Making a bar graph

To make a bar graph, pass your data to the `Bar()` graph object.

```
from plotly.graph_objs import Bar
--snip--

data = [Bar(x=x_values, y=squares)]

# Pass the data and a filename to plot().
offline.plot(data, filename='squares.html')
```

Adding a title and labels

Using layout objects

The `Layout` class allows you to specify titles, labels, and other formatting directives for your visualizations.

```
from plotly.graph_objs import Scatter, Layout
from plotly import offline

x_values = list(range(11))
squares = [x**2 for x in x_values]

data = [Scatter(x=x_values, y=squares)]

# Add a title, and a label for each axis.
title = 'Square Numbers'
x_axis_config = {'title': 'x'}
y_axis_config = {'title': 'Square of x'}

my_layout = Layout(title=title,
    xaxis=x_axis_config, yaxis=y_axis_config)

offline.plot(
        {'data': data, 'layout': my_layout},
        filename='squares.html')
```

Specifying complex data

Data as a dictionary

Plotly is highly customizable, and most of that flexibility comes from representing data and formatting directives as a dictionary. Here is the same data from the previous examples, defined as a dictionary.

Defining the data as a dictionary also allows you to specify more information about each series. Anything that pertains to a specific data series such as markers, lines, and point labels, goes in the data dictionary. Plotly has several ways of specifying data, but internally all data is represented in this way.

```
data = [{
    'type': 'scatter',
    'x': x_values,
    'y': squares,
    'mode': 'markers',
}]
```

Multiple plots

You can include as many data series as you want in a visualization. To do this, create one dictionary for each data series, and put these dictionaries in the data list. Each of these dictionaries is referred to as a trace in the Plotly documentation.

Plotting squares and cubes

Here we use the `'name'` attribute to set the label for each trace.

```
from plotly.graph_objs import Scatter
from plotly import offline

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]

data = [
    {
        # Trace 1: squares
        'type': 'scatter',
        'x': x_values,
        'y': squares,
        'name': 'Squares',
    },
    {
        # Trace 2: cubes
        'type': 'scatter',
        'x': x_values,
        'y': cubes,
        'name': 'Cubes',
    },
]

offline.plot(data,
        filename='squares_cubes.html')
```

Online resources

The Plotly documentation is extensive and well-organized. Start with the overview at plotly.com/python/. Here you can see an example of all the basic chart types, and click on any example to see a relevant tutorial.

Then take a look at the Python Figure Reference, at plotly.com/python/reference/. Check out the Figure Data Structure in Python page as well, at plotly.com/python/figure-structure/.

Specifying complex layouts

You can also specify the layout of your visualization as a dictionary, which gives you much more control of the overall layout.

Layout as a dictionary

Here is the same layout we used earlier, written as a dictionary. Simple elements such as the title of the chart are just key-value pairs. More complex elements such as axes, which can have many of their own settings, are nested dictionaries.

```
my_layout = {
    'title': 'Square Numbers',
    'xaxis': {
        'title': 'x',
    },
    'yaxis': {
        'title': 'Square of x',
    },
}
```

A more complex layout

Here is a layout for the same data, with more specific formatting directives in the data and layout dictionaries.

```
from plotly.graph_objs import Scatter
from plotly import offline

x_values = list(range(11))
squares = [x**2 for x in x_values]

data = [{
    'type': 'scatter',
    'x': x_values,
    'y': squares,
    'mode': 'markers',
    'marker': {
        'size': 10,
        'color': '#6688dd',
    },
}]

my_layout = {
    'title': 'Square Numbers',
    'xaxis': {
        'title': 'x',
        'titlefont': {'family': 'monospace'},
    },
    'yaxis': {
        'title': 'Square of x',
        'titlefont': {'family': 'monospace'},
    },
}

offline.plot(
       {'data': data, 'layout': my_layout},
       filename='squares.html')
```

Specifying complex layouts (cont.)

Using a colorscale

Colorscales are often used to show variations in large datasets. In Plotly, colorscales are set in the marker dictionary, nested inside a data dictionary.

```
data = [{
    'type': 'scatter',
    'x': x_values,
    'y': squares,
    'mode': 'markers',
    'marker': {
        'colorscale': 'Viridis',
        'color': squares,
        'colorbar': {'title': 'Value'},
    },
}]
```

Using Subplots

It's often useful to have multiple plots share the same axes. This is done using the subplots module.

Adding subplots to a figure

To use the `subplots` *module, make a figure to hold all the charts that will be made. Then use the* `add_trace()` *method to add each data series to the overall figure.*

For more help, see the documentation at plot.ly/python/subplots/.

```
from plotly.subplots import make_subplots
from plotly.graph_objects import Scatter
from plotly import offline

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]

# Make two subplots, sharing a y-axis.
fig = make_subplots(rows=1, cols=2,
        shared_yaxes=True)

data = {
    'type': 'scatter',
    'x': x_values,
    'y': squares,
}
fig.add_trace(data, row=1, col=1)

data = {
    'type': 'scatter',
    'x': x_values,
    'y': cubes,
}
fig.add_trace(data, row=1, col=2)

offline.plot(fig, filename='subplots.html')
```

Plotting global datasets

Plotly has a variety of mapping tools. For example, if you have a set of points represented by latitude and longitude, you can create a scatter plot of those points overlaying a map.

The scattergeo chart type

Here's a map showing the location of three of the higher peaks in North America. If you hover over each point, you'll see its location and the name of the mountain.

```
from plotly import offline

# Points in (lat, lon) format.
peak_coords = [
    (63.069, -151.0063),
    (60.5671, -140.4055),
    (46.8529, -121.7604),
]

# Make matching lists of lats, lons,
#   and labels.
lats = [pc[0] for pc in peak_coords]
lons = [pc[1] for pc in peak_coords]
peak_names = ['Denali', 'Mt Logan',
        'Mt Rainier']

data = [{
    'type': 'scattergeo',
    'lon': lons,
    'lat': lats,
    'marker': {
        'size': 20,
        'color': '#227722',
    },
    'text': peak_names,
}]

my_layout = {
    'title': 'Selected High Peaks',
    'geo': {
        'scope': 'north america',
        'showland': True,
        'showocean': True,
        'showlakes': True,
        'showrivers': True,
    },
}

offline.plot(
        {'data': data, 'layout': my_layout},
        filename='peaks.html')
```

More cheat sheets available at

ehmatthes.github.io/pcc_2e/