Visualizing Data with MatPlotLib

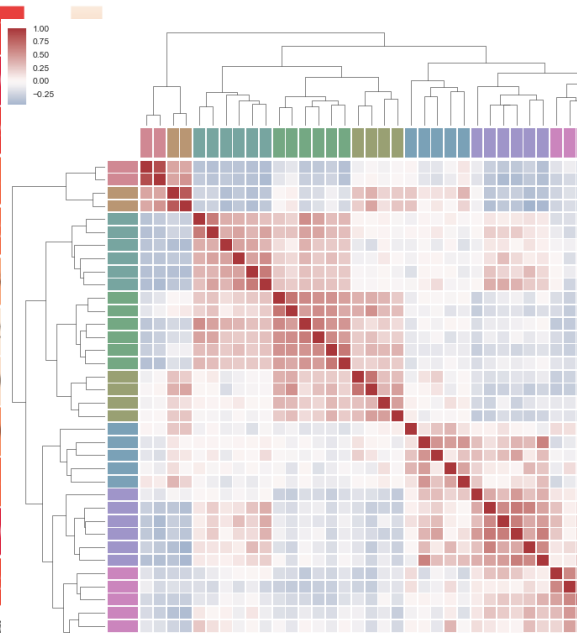
What is matplotlib?

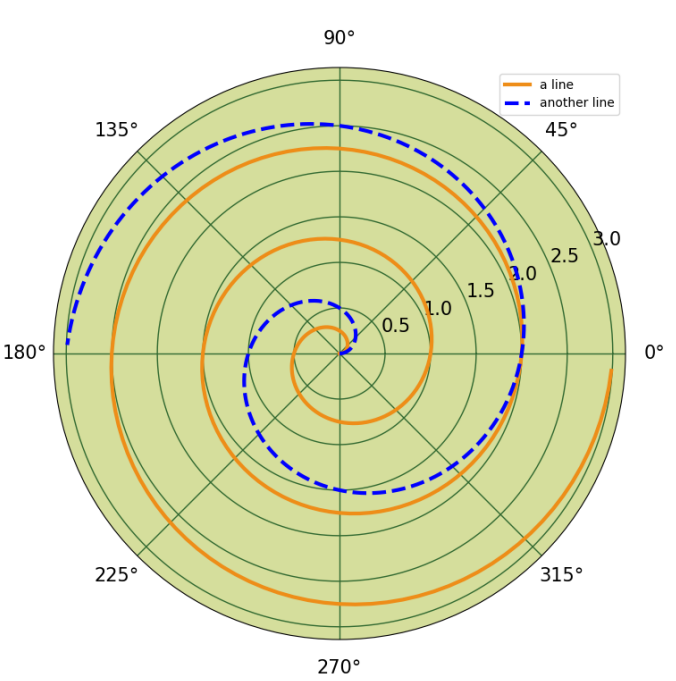
- An open-source Python graphing library officially launched in 2003.

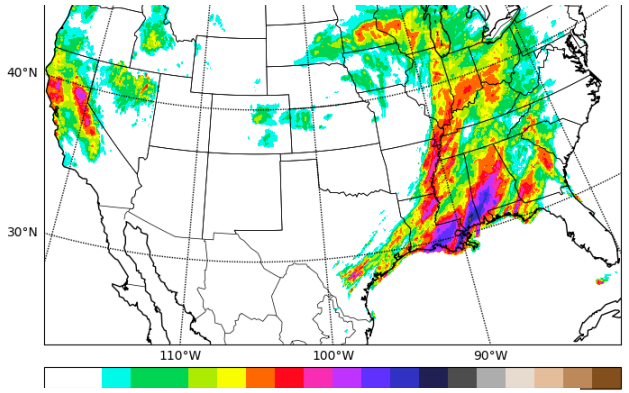
- Ability to generate a variety of different types of graphs from simple to complex, as well as drawing capabilities and interactive capabilities similar to MATLAB

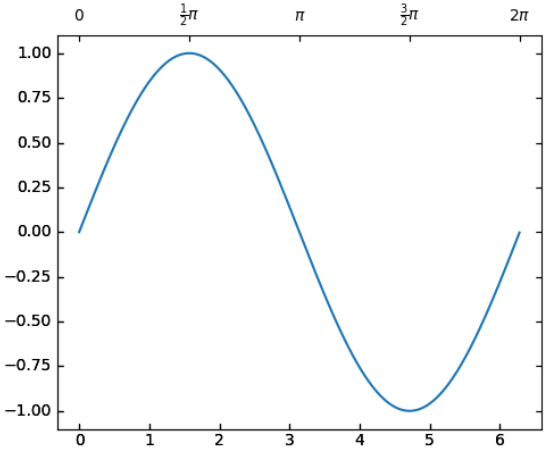
- Many third-party open-source Python libraries have a matplotlib dependancy as well, some of which we will cover in this tutorial
    - **Pandas**
    - **Seaborn**
    - Sci-kit learn (great for machine learning)
    - NetworkX
    - ggplot (based on R's ggplot2 library)
    - **IPython**
    - StarCluster
    - Bokeh (interactive plots for web-dev; doesn't require matplotlib but is compatible with it)

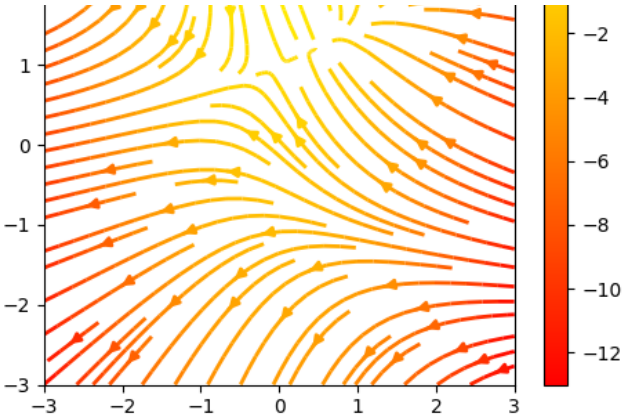

January 112 115 145 171 196 204 242 284 315 340 360 41
February 118 126 150 180 196 188 233 277 301 318 342 39
March 132 141 178 193 236 235 267 317 356 362 406 41
April 129 135 163 181 235 227 269 313 348 348 396 46
May 121 125 172 183 229 234 270 318 355 363 420 47
June 135 149 178 218 243 264 315 374 422 435 472 53
July 148 170 199 230 264 302 364 413 465 491 548 62
August 148 170 199 242 272 293 347 405 467 505 559 60
September 136 158 184 209 237 259 312 355 404 404 463 50
October 119 133 162 191 211 229 274 306 347 359 407 46
November 104 114 146 172 180 203 237 271 305 310 362 39
December 118 140 166 194 201 229 278 306 336 337 405 43
1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 196
year

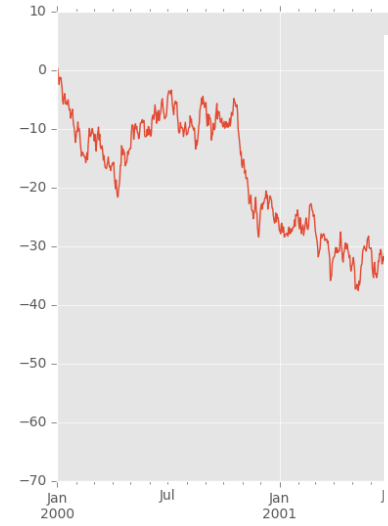


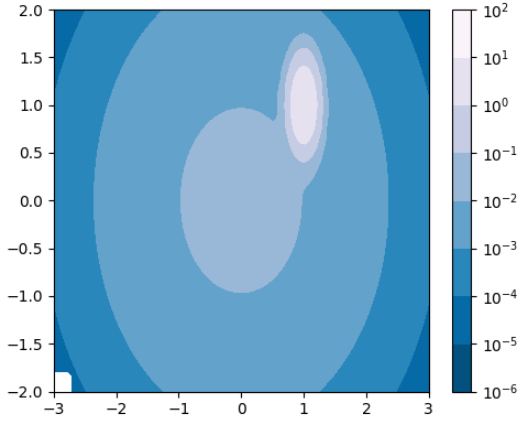

90°
135°
45°
180°
0°
225°
315°
270°
a line
another line
0.5
1.0
1.5
2.0
2.5
3.0

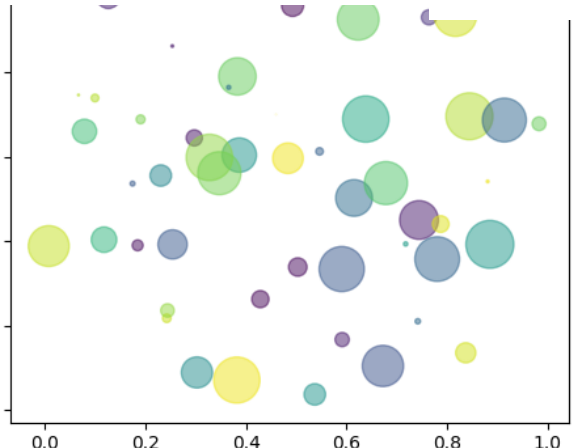

40°N
30°N
110°W
100°W
90°W

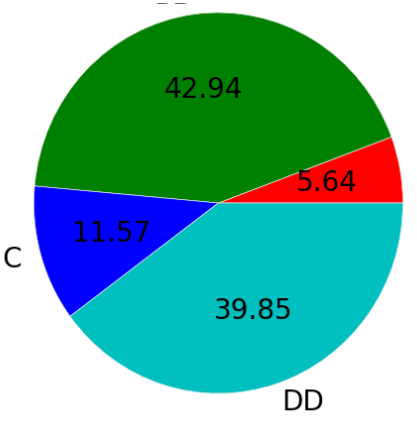

0
½π
π
3/2π
2π
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00




10
0
−10
−20
−30
−40
−50
−60
−70
Jan 2000
Jul
Jan 2001






42.94
5.64
11.57
39.85
AA
C
DD

The matplotlib API

- Essentially 3 different matplotlib APIs exist that make scripting in matplotlib easy for the user:
    1) pylab
    2) pyplot
    3) object-oriented interface

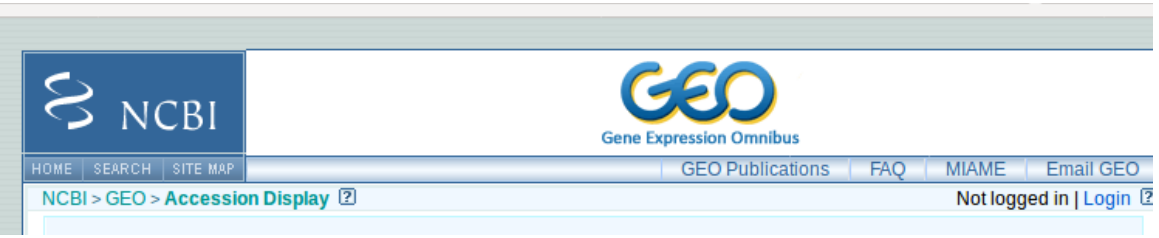
We are going to focusing on ***pyplot*** for this tutorial. Why?

First and foremost, pylab is deprecated. Additionally, the pyplot API is easy to use and is built to handle visualizations easily and quickly. Additionally it has similar behaviors as pylab which was meant to be used as a substitute for MATLAB graphing, with similar syntax format.

Although pyplot is also object-oriented, the API takes care of the direct creation of graphing objects on the backend so the user does not have to deal with scripting and building these objects, unlike the object-oriented interface where the user must create these objects manually and it is non-interactive, so it cannot be used with IPython or Jupyter. So unless you need super fine-artistic details and customization of your graphs, **pyplot is much easier and will generally meet the needs of most users!**

Data Sets

- There are several free and publicly available complex dataset that can be used for prototyping code/algorithms/visualizations.
- All data sets used in this tutorial have come from the one of the following websites:

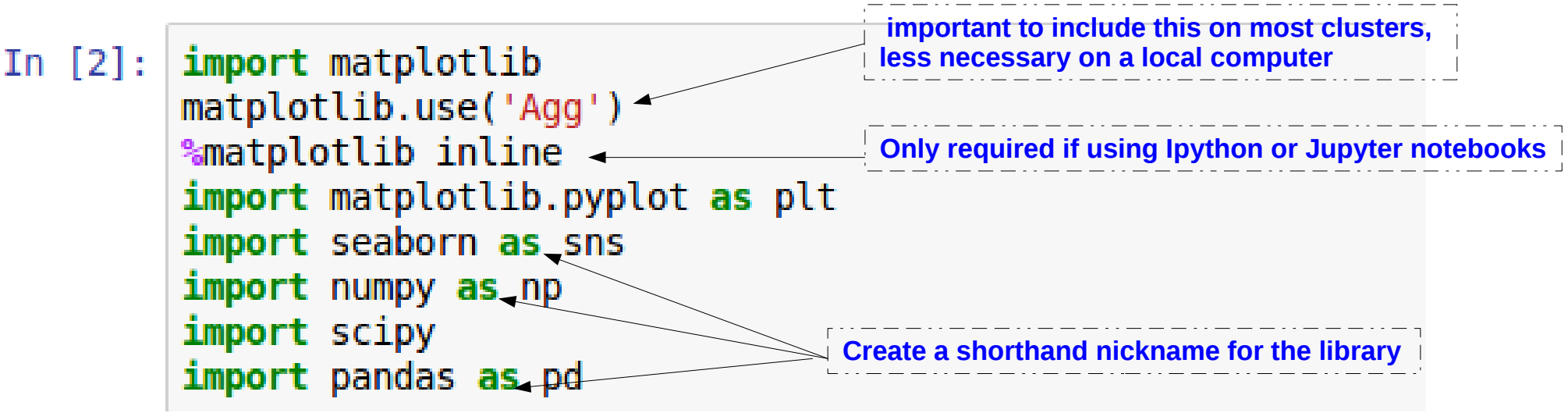


Installation and backend set up

For this tutorial all libraries will already be installed on the cluster, however for the future matplotlib requires the several dependencies. The easiest way to install this locally will be to use your OS package manager or pip.

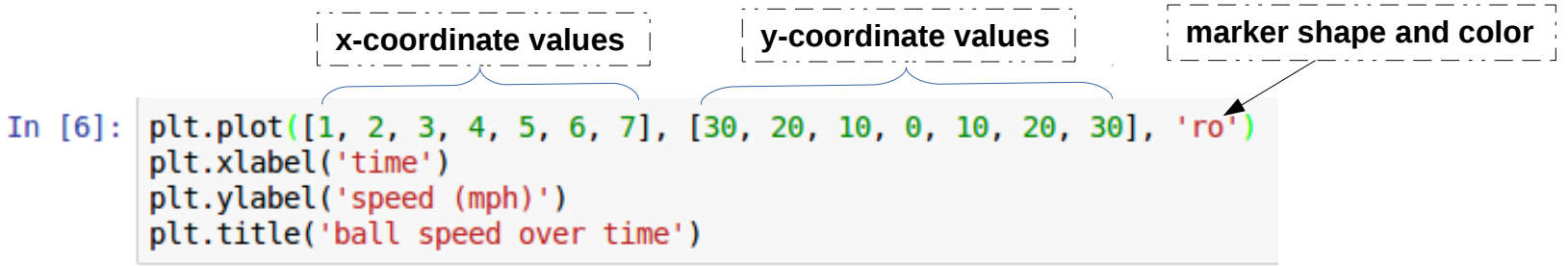
https://matplotlib.org/users/installing.html

This list of imports should be placed at the top of your Python scripts or should be executed first:



Generating simple plots

- Open script basic_test_plots.py, run lines 10-14



- Python graphs the arrays in elemental order, so the first element of x-coordiate values matches the first element of y-coordiate values. Hence, the lists/arrays must be of same length. In this case, we could expect the following points plotted:

  (1, 30), (2, 20), (3, 10), (4, 0), (5, 10), (6, 20), (7, 30) , the marker shape and color tells python to plot red (‘r’) circles (‘o’)

There are tables to the right that describes the keywords for some of the basic colors and symbols. There are many more and you can even use html hex strings and shades by specifying the ‘color’ keyword argument

code	color
b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black
w	white

code	marker symbol
.	point
,	pixel
o	circle
^	Up triangle
<	Right triange
+	plus
x	x
D	diamond
H	Hexagon
s	square
8	octagon
p	pentagon
*	star
P	Filled plus

Generating simple plots

- Now, run lines 17-28

```
In [12]: time_points = np.arange(0, 10, 1)
         response_1 = [0, .2, .4, .8, 1.6, 3.2, 3.2, 3.2, 3.2, 3.2]
         response_2 = [0, .1, .2, .4, .8, 1.6, 3.2, 6.4, 12, 12]
         response_3 = [0, .1, .2, .3, .4, .8, 1.6, 1.6, 1.6,  1.6]
         plt.plot(time_points, response_1, 'g*', label='trial1')
         plt.plot(time_points, response_2, 'rH', label='trial2')
         plt.plot(time_points, response_3, 'bv', label='trial3')
         plt.ylabel('response')
         plt.xlabel('time (min)')
         plt.title('Drug response over time')
         plt.legend(loc='upper left')
```

- How do we get dashed and continuous lines instead? Follow the directions on lines 31-34

- Do you want to save your plot to a file?

Plotting large complex datasets

- Many people will already have a spreadsheet or table of various metrics they may want to visualize so typing all of these into the code we have before is not a feasible option.

- Additionally, they may want other graph styles besides basic scatter and line plots. What do we do?

- ***Also, do we really want to hard code or loop through all data? Time consuming!***

Histograms, bar graphs, and pie charts

- Open the histogram_barplot_piechart.py script. Let's use a modified and cleaned up dataset on the Game of Thrones

```
In [4]: data
```

Out[4]:

	name	year	battle_number	attacker_king	defender_king	attacker_1	defender_1	attacker_outcome	battle_type	major_death	major_ca
0	Battle of the Golden Tooth	298	1	Joffrey/Tommen Baratheon	Robb Stark	Lannister	Tully	win	pitched battle	1	0
1	Battle at the Mummer's Ford	298	2	Joffrey/Tommen Baratheon	Robb Stark	Lannister	Baratheon	win	ambush	1	0
2	Battle of Riverrun	298	3	Joffrey/Tommen Baratheon	Robb Stark	Lannister	Tully	win	pitched battle	0	1

- We are interested in knowing how many battles occurred per year. How do we do this? Run lines 13-14

```
In [2]: # bargraph/histogram example
        # read in csv file, infers if header is available
        data = pd.read_csv('GOT_battles.csv')
        sns.countplot(x='year', data=data, palette='cubehelix')
```

Only x is required, no y values because sns.countplot() uses the total number of occurrences for each group of x

name of dataframe

color scheme (we will discuss this further later in the presentation)

Histograms, bar graphs, and pie charts

- What if we want to plot the graph in descending order? We will have to do a little coding...Run lines 16-24

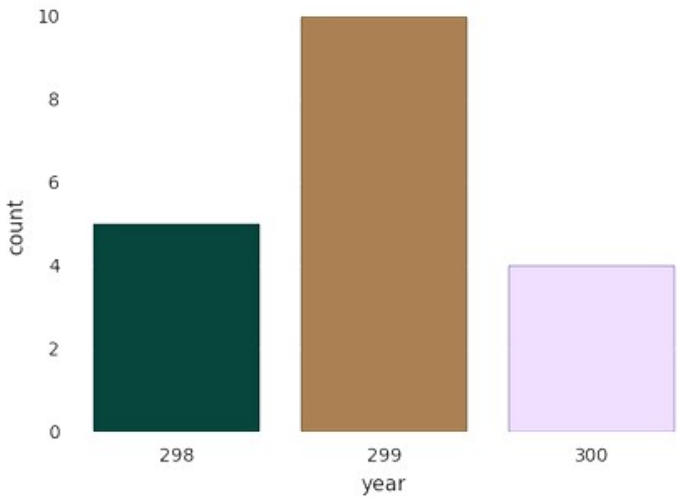
```
In [ ]: # another bargraph/histogram examples
order_names=[]
most=collections.Counter(data['attacker_1']).most_common()
for tuples in most:
    order_names.append(tuples[0])
graph = sns.countplot(x='attacker_1', data=data, order=order_names)

for item in graph.get_xticklabels():
    item.set_rotation(45)
graph.set_title('Fight starter', fontsize=20)
```

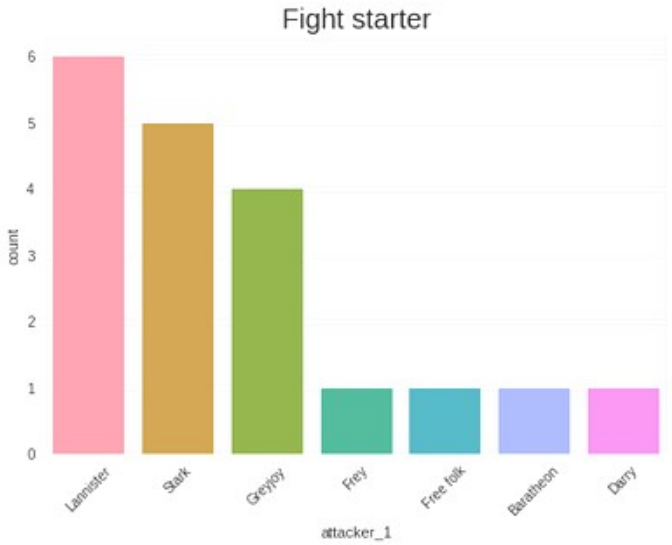
Creates a dictionary of this in descending order or occurrences

Put keys in an ordered list

By specifying the order keyword pass in the list

Rotates the labels 45 degrees along the x-axis so not cluttered





- When plotting with seaborn, unfortunately, setting the axis rotation can be a little tricker than with pandas since it does not have a rotation keyword
- Exercise, reorder the x-axis in the year vs total battles plot to be in ascending order (Line 28)

Pie charts in pandas

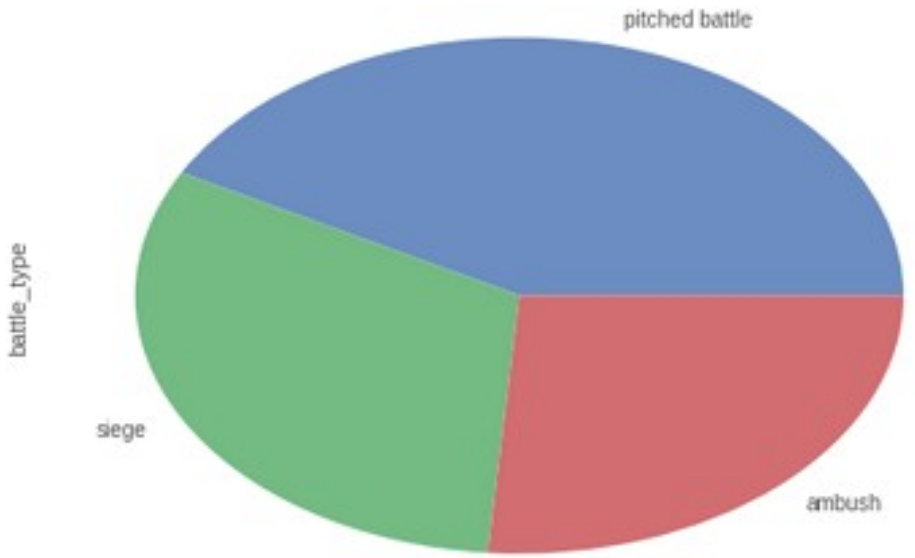
- Using the same dataset, let's make a pie chart of what percentage of battles of which type. Run lines 32-33. This will produce the most basic pie chart.

```
In [2]: # pie chart using pandas
        battle_locs = data['battle_type'].value_counts() #
        battle_locs.plot.pie()
```

```
In [4]: battle_locs

Out[4]: pitched battle    8
        siege             6
        ambush            5
        Name: battle_type, dtype: int64
```

The pandas pie chart feature is not "smart" enough to actually count the occurrences of each battle type, so it is necessary to count the values and store it as a dataframe

This is the new data object after we apply .value_counts() to the column 'battle_type'. Ultimately, this is the object where we apply the plot.pie() function



- However, this does not tell us a lot of information and it is pretty plain and more of an oval shape. Can we do better?

Pie charts in pandas...FANCY!

- By adding a few built-in keywords, we can make the same pie chart with more information and more aesthetically pleasing. Run lines 36-39

Plot the percentage to 2 decimal places

Piece out a slice of the pie

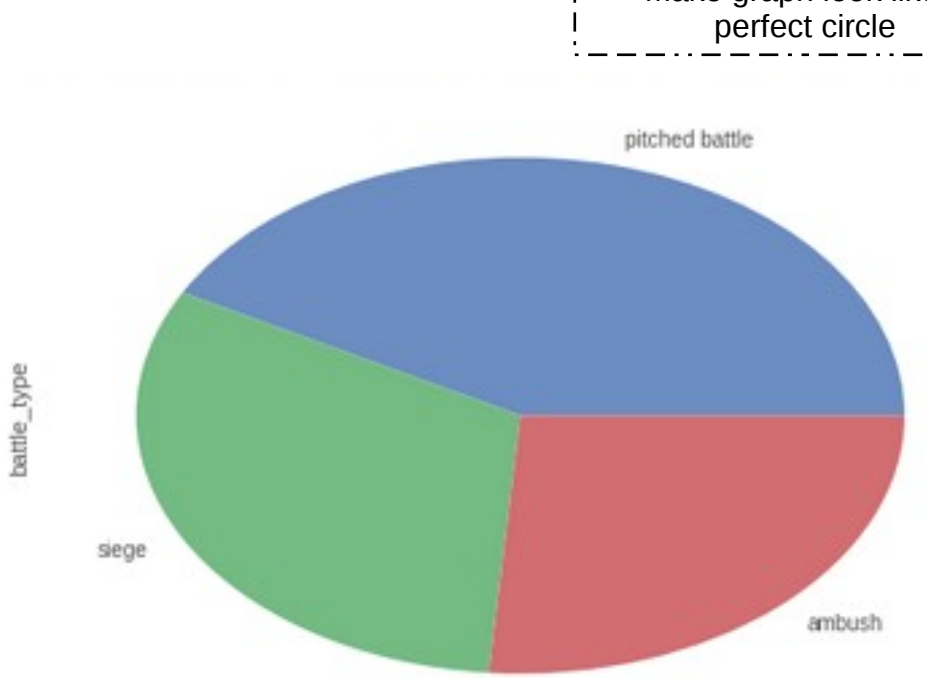
Place a shadow

```
In [3]: # fancier pie chart
        battle_locs.plot(kind='pie', autopct='%.2f%%', explode=(0,0,0.2), startangle=180, shadow=True,
                         colors=('#FFFF00', '#9900CC', '#99FF99'))
        plt.title('Percentage of battle at specifed locations')
        plt.ylabel('')
        plt.axis('equal')
```

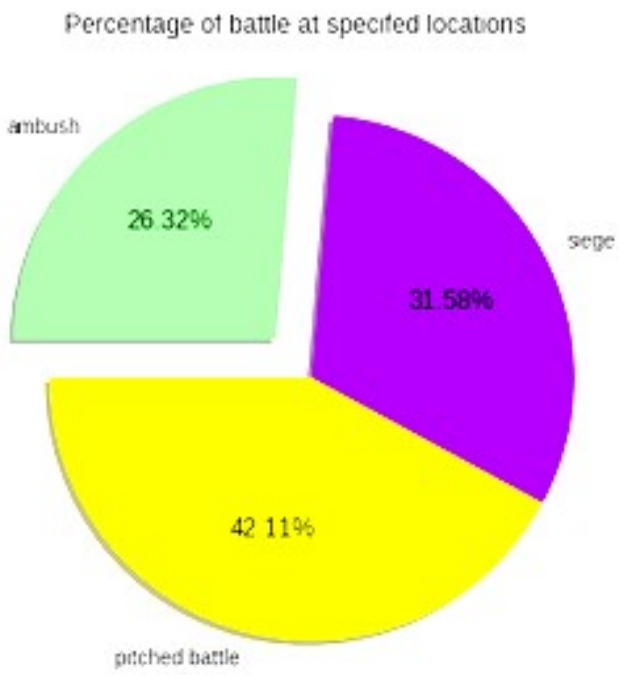
Degree of circle where you would like to start

Color pieces of the circle

Make graph look like a perfect circle



vs



- Exercise Generate a new pie plot of wins vs losses when the attacker initiates a battle. See lines 43-44

General plotting syntax with pandas

- The plotting functions used in pandas can all generally be called using the following command:

NameOfDataframe.plot(kind="pickKeyword")

The following keywords are acceptable for the "kind" parameter:

- line
- bar
- barh
- hist
- box
- kde
- density
- area
- pie
- scatter
- hexbin

- No matter which kind of plot you choose, you can always add the following keywords (not an exhaustive list) : title, xticks, yticks, xlim, ylim, fontsize, colormap, legend, grid, style… and many others.

- It gives the user the convenient option of making one command call (assuming the dataframe is configured and organized correctly)

Heatmaps and clustermaps

- Seaborn generates heatmaps and clustermaps with minimal code
- To illustrate, will use a cleaned dataset of a quality-of-life survey from the European Foundation for the Improvement of Living and Working Conditions

- Open script heatmap_clustermap.py and run lines 10-12

```
In [22]: eur_qol = pd.read_csv('european_quality_of_life.csv')
         order = eur_qol.pivot('Country', 'Gender', 'Mean')
```

Reshapes dataframe for compatibility with seaborn graphs

- Compare the difference between eur_qol and order:

Out[23]:

	Country	Gender	Mean	Total
0	AT	Both	7.3	1041
1	AT	Female	7.3	570
2	AT	Male	7.3	471

Out[8]:

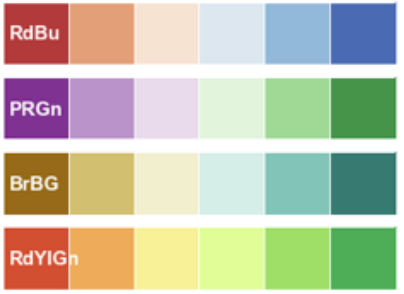
Gender	Both	Female	Male
Country			
AT	7.3	7.3	7.3
BE	7.8	7.8	7.8
BG	5.8	5.8	5.8

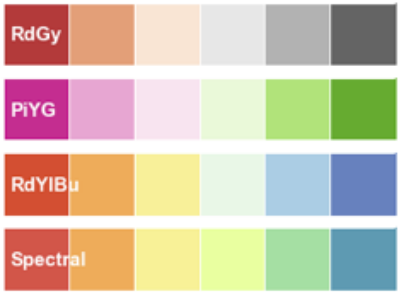
Duplicate 'Country' rows are collapsed to represent the y-axis, similarly with 'Gender' to represent to x-axis

Mean is now structured to represent the cells of the clustermap

Heatmap and clustermap beautification and customization

- Change font size
- Choose color palettes
- Change spacing and shape
- Add cell annotation
- Choose clustering method and distance metrics (clustermap)





Or create your own by creating a Python list of keyword colors or html codes and using:
sns.set_palette(your_list)
sns.palplot(sns.color_palette())

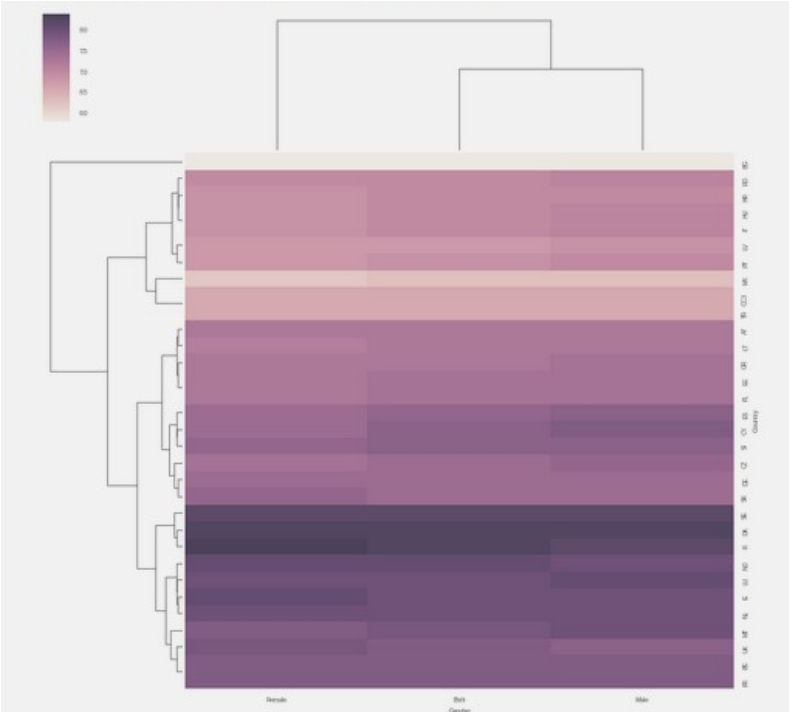
Palette images from http://jose-coto.com/styling-with-seaborn

Heatmaps and clustermaps

- Let’s generate the most basic heatmap and clustermap from this data
- Open script heatmap_clustermap.py and run line 13 and then line 14

```
In [29]: sns.clustermap(order)

In [30]: sns.heatmap(order)
```

- You can see the difference between the clustermap and the heatmap is the clustering of data with the most similar cells and including a dendrogram to illustrate this



Example heatmaps

- Run lines 18-19

```
In [38]: sns.set(font_scale = 1.0)
         sns.heatmap(order, linewidths=0.25, cmap='Blues', square=True)
```

Whitespace to leave between cells

Set colormap color

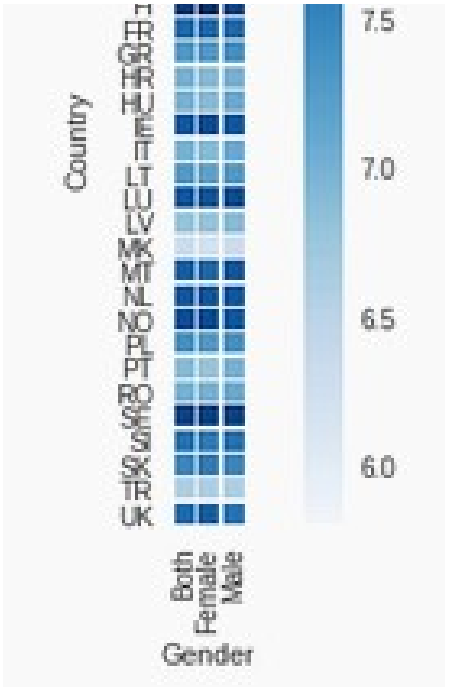
Draws cells perfectly square instead of rectangle

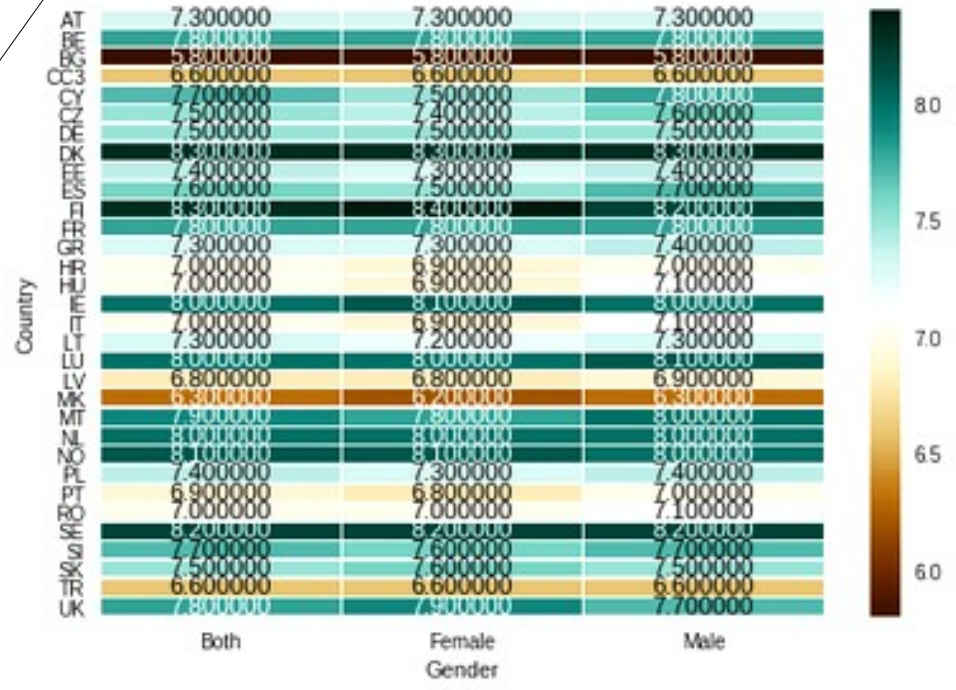
- Run lines 22-23

```
In [43]: sns.set(font_scale = 1.0)
         sns.heatmap(order, annot=True, fmt= 'f', linewidths=0.50, cmap='BrBG')
```

Annotate cells with values

Tells Python how to format cells





Example clustermaps

```
In [35]: sns.set(font_scale = 1.0)
         sns.clustermap(order, linewidths=0.25, cmap='coolwarm', method = 'centroid', metric = 'hamming')
```

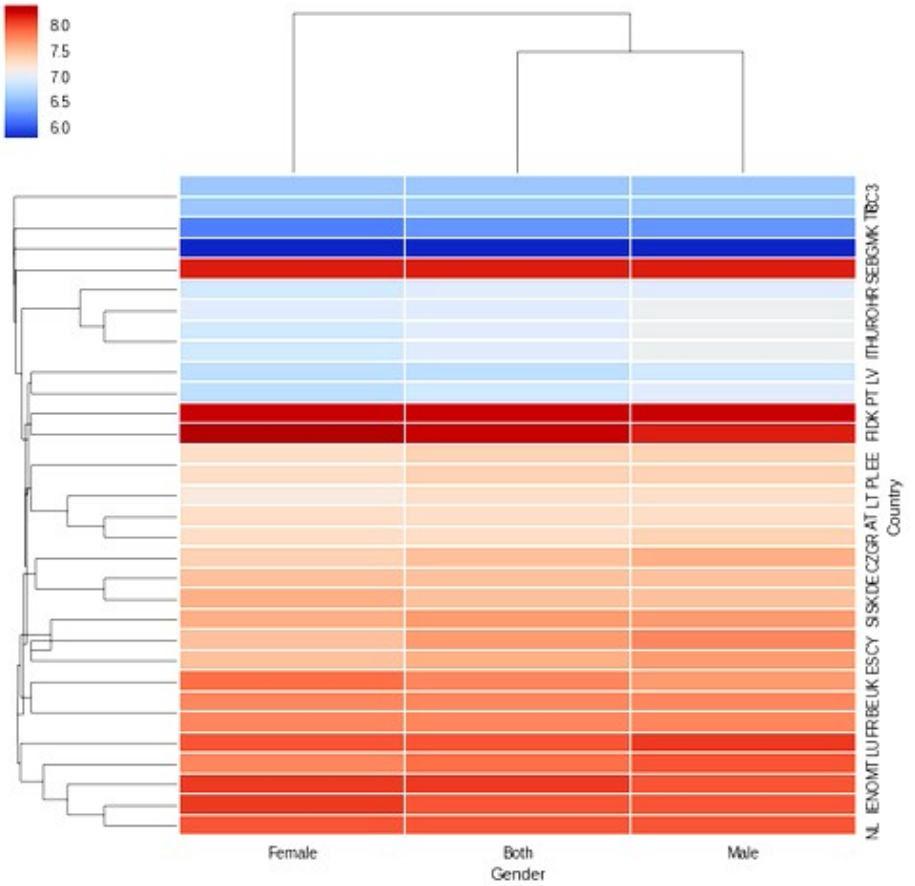
Determine cluster membership method

Determine clustering method and sample similarity metric

- Run lines 25-27

metric	
barycurtis	canberra
chebysev	cityblock
correlation	cosine
dice	euclidean
hamming	jaccard
kulsinski	mahalanobis
matching	minkowski
rogerstanimoto	russellrao
seuclidean	sokalmichener
sokalsneath	sqeuclidean
yule	

method
single
complete
average
weighted
centroid
median
ward



- Exercise line 30. Create your own heatmap using differential expression data from GEO (GSE27233_GEO_subset.txt) and use a diverging color palette centered around 0. Hint: use center=0 as a keyword argument

Grouped box plots

- Let's take a peek at a very small and modified version of stock market data.

```
In [43]: stock_data
```

Out[43]:

	Month	Open	High	Low	Close	Day
0	August	181.47	181.47	179.90	180.27	MON
1	August	181.00	181.39	180.46	180.56	MON
2	August	180.12	180.33	179.21	179.25	MON
3	August	178.67	180.09	178.32	179.87	MON

- Open the script grouped_plots.py and run lines 11-12

```
In [4]: stock_data = pd.read_table('all_stocks_1yr_modified.csv')
        sns.boxplot(x="Month", y="Open", hue="Day", data=stock_data)
```

Main group categories

Dependent variable measurements

Determines how to subgroup data

Name of dataframe

- Some other options you can include in the .boxplot() method are: orient, color, order, palette, saturation, width, dodge, fliersize, linewidth, whis, notch
- Another cool thing you can do with the seaborn boxplot is to overlay a strip plot/swarm plot on top of the boxplots. Run lines 18-19 together.

```
In [4]: sns.boxplot(x="Month", y="Open", data=stock_data)
        sns.swarmplot(x="Month", y="Open", data=stock_data, color='k')
```

Grouped boxplots in pandas vs seaborn

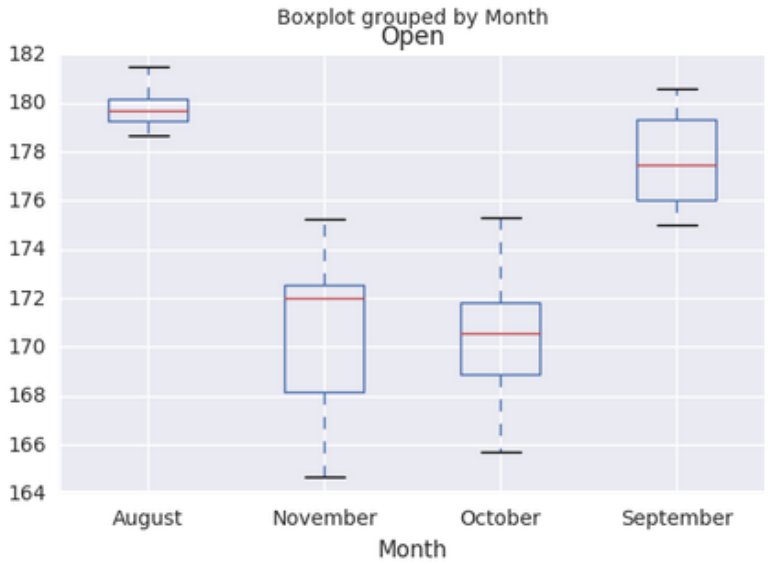
- Open the script grouped_plots.py and run line 23

Graphs statistics for the y-axis and boxplot

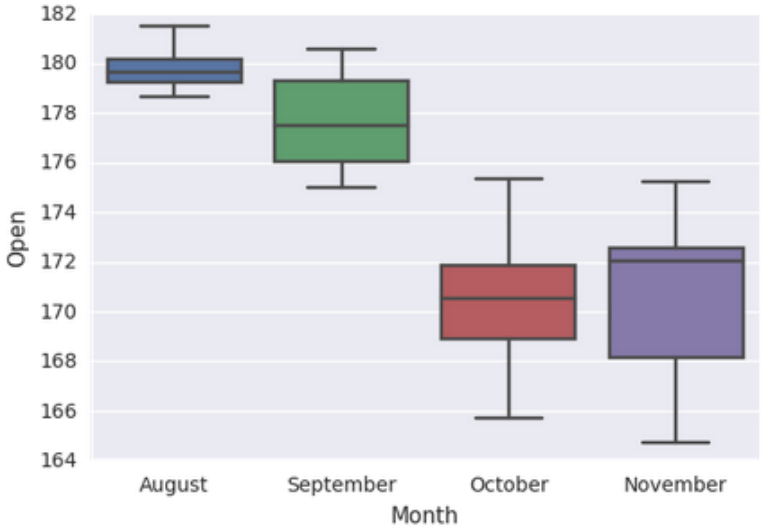
Determines what groups to separate data out on the x-axis

Name of dataframe

```
In [39]: stock_data.boxplot(column='Open', by='Month')

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7f18c09023d0>
```




```
In [44]: sns.boxplot(x="Month", y="Open", data=stock_data)

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x7f18c0fe0d50>
```



- Exercise on line 28. Read in the facebook dataset and create a set of box plots where the column ‘Type’ are the groups on the x-axis and the y-axis will represent the total number of people that have both liked and engaged with a post. Use the notch parameter and set it to true and use the orient parameter and set it to ‘h’. Use any color scheme you would like and set the font equal to 2.0 NOTE! ***This dataset is semi-colon delimited!!***

Differences between Pandas and Seaborn

Pandas

- Easier formatting axis positioning (fontsize and rotation)
- Pie chart option is easier
- Also capable of cleaning up “dirty data”
- Can plot data in one command call with several built-in keyword arguments

Seaborn

- No need to “count” data
- Easy to get visually complex and pleasing graphs
- Not made for cleaning data
- Can create heatmaps and clustermaps

Fancier Plots (made easy!)

- Open fancy_plots.py