PYTHON DATA VISUALIZATIONS

Content from Jose Portilla's Udemy course *Learning Python for Data Analysis and Visualization*
https://www.udemy.com/learning-python-for-data-analysis-and-visualization/
Notes by Michael Brothers, available on http://github.com/mikebrothers/data-science/

Table of Contents

Note: except where noted, code & output are Python v2.7 on Jupyter Notebooks

MATPLOTLIB

Scatter plot with wine data downloaded from UC Irvine's Machine Learning Archive:

```
import numpy as np
import pandas as pd
from pandas import Series,DataFrame
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/'
```

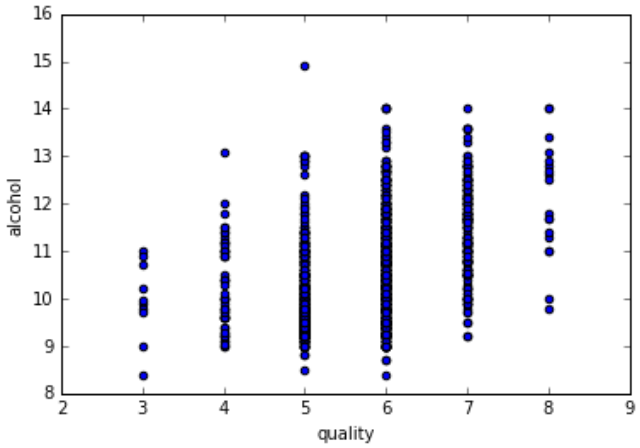
the file 'winequality-red.csv' was saved to the jupyter notebook directory

```
dframe_wine = pd.read_csv('winequality-red.csv',sep=';')
```

note the separator

```
%matplotlib inline
dframe_wine.plot(kind='scatter',x='quality',y='alcohol');
```

did not have to import matplotlib



Bar plot with errorbars (see http://matplotlib.org/examples/api/barchart_demo.html)

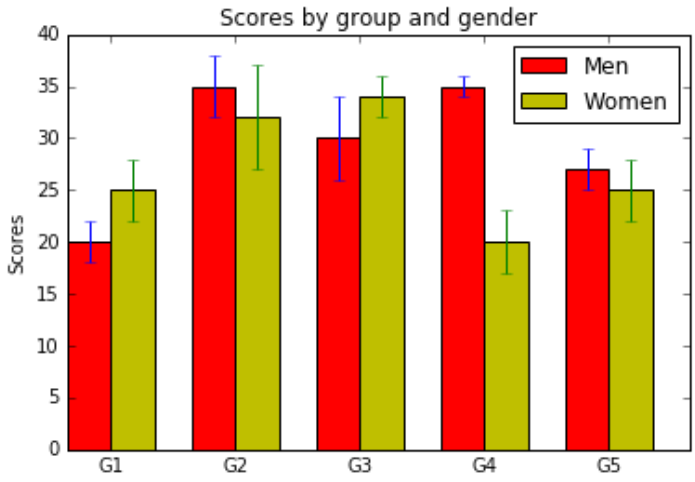
```
import numpy as np
import matplotlib.pyplot as plt
N = 5
ind = np.arange(N)      # the x locations for the groups
width = 0.35            # the width of the bars
fig, ax = plt.subplots()

menMeans = (20, 35, 30, 35, 27)
menStd = (2, 3, 4, 1, 2)
rects1 = ax.bar(ind, menMeans, width, color='r', yerr=menStd)
womenMeans = (25, 32, 34, 20, 25)
womenStd = (3, 5, 2, 3, 3)
rects2 = ax.bar(ind + width, womenMeans, width, color='y', yerr=womenStd)
```

add axis labels, title and axis tickmarks

```
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks(ind + width)
ax.set_xticklabels(('G1', 'G2', 'G3', 'G4', 'G5'))
ax.legend((rects1[0], rects2[0]), ('Men', 'Women'))
```

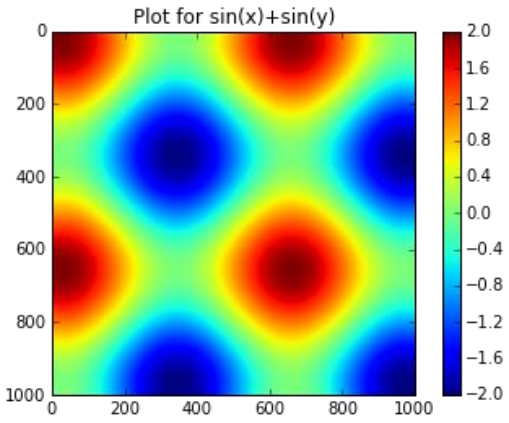
you can add data labels above each bar (I chose not to in the plot below):

```
def autolabel(rects):
    # attach some text labels
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                '%d' % int(height),
                ha='center', va='bottom')
autolabel(rects1)
autolabel(rects2)

plt.show()              if not using %matplotlib inline in iPython
```



3D Graphical Analysis:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline                          display the plot immediately
points = np.arange(-5,5,0.01)               grab an array of 1000 datapoints
dx,dy=np.meshgrid(points,points)            create the grid
z = (np.sin(dx) + np.sin(dy))               set an evaluating function

plt.imshow(z)                               plot the array
                                            NOTE: plots the positions 1-1000, not the values -5 to 5
plt.colorbar()                              add a colorbar & title
plt.title("Plot for sin(x)+sin(y)")         add a chart title
```



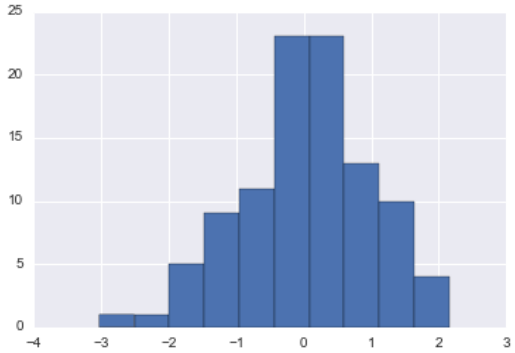
Histograms:

```
from numpy.random import randn
import matplotlib.pyplot as plt
%matplotlib inline
```

for generating random number datasets (normal distribution)

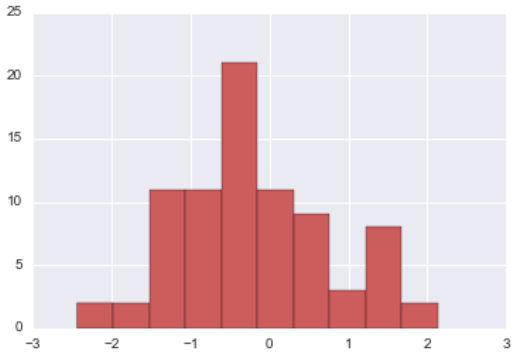
so that plots appear in the iPython Notebook

```
dataset1 = randn(100)
plt.hist(dataset1)
(array([  1.,   1.,   5.,   9.,  11.,  23.,  23.,  13.,  10.,   4.]),
 array([-3.03051447, -2.5119968 , -1.99347913, -1.47496147, -0.9564438 ,
        -0.43792613,  0.08059154,  0.5991092 ,  1.11762687,  1.63614454,
         2.1546622 ]),
 <a list of 10 Patch objects>)
```



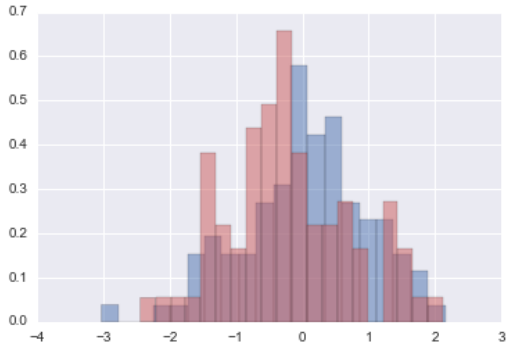
data grouped into 10 bins by default,
with 11 equally spaced borders (min to max)

```
dataset2 = randn(80)
plt.hist(dataset2,color='indianred')
```



dataset2 is set to indianred for clarity

```
plt.hist(dataset1,normed=True,alpha=0.5,bins=20)
plt.hist(dataset2,normed=True,color='indianred',alpha=0.5,bins=20)
```



plot both histograms in the same Jupyter notebook cell

normed=True normalizes the data
(since we have two different-sized datasets)

alpha=0.5 sets the transparency

SEABORN LIBRARIES

Required dependencies: numpy, scipy, matplotlib, pandas; **Recommended**: statsmodels, patsy

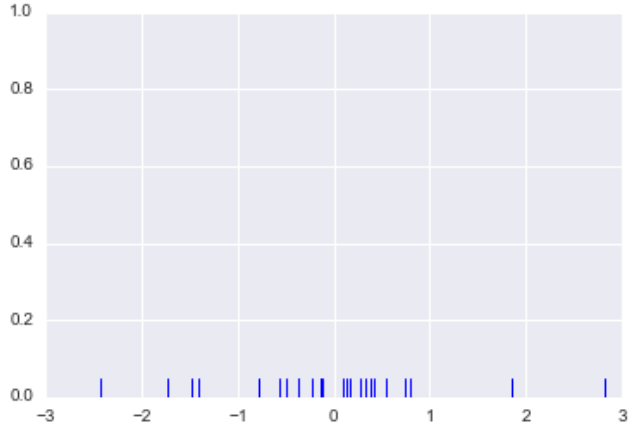
Standard imports:

```
import numpy as np
import pandas as pd
from numpy.random import randn          for generating random number datasets (normal distribution)
from scipy import stats                 the numpy stats library
import matplotlib as mpl                plotting modules and libraries
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline                      so that plots appear in the iPython Notebook
```

NOTE: matplotlib throws a UserWarning: axes.color_cycle is deprecated and replaced with axes.prop_cycle

Rug Plots

```
dataset = randn(25)
sns.rugplot(dataset)                    plots a simple row of tic marks along the x-axis
```



Histograms using factorplot

Note: Histograms are already part of matplotlib: `plt.hist(dataset)`

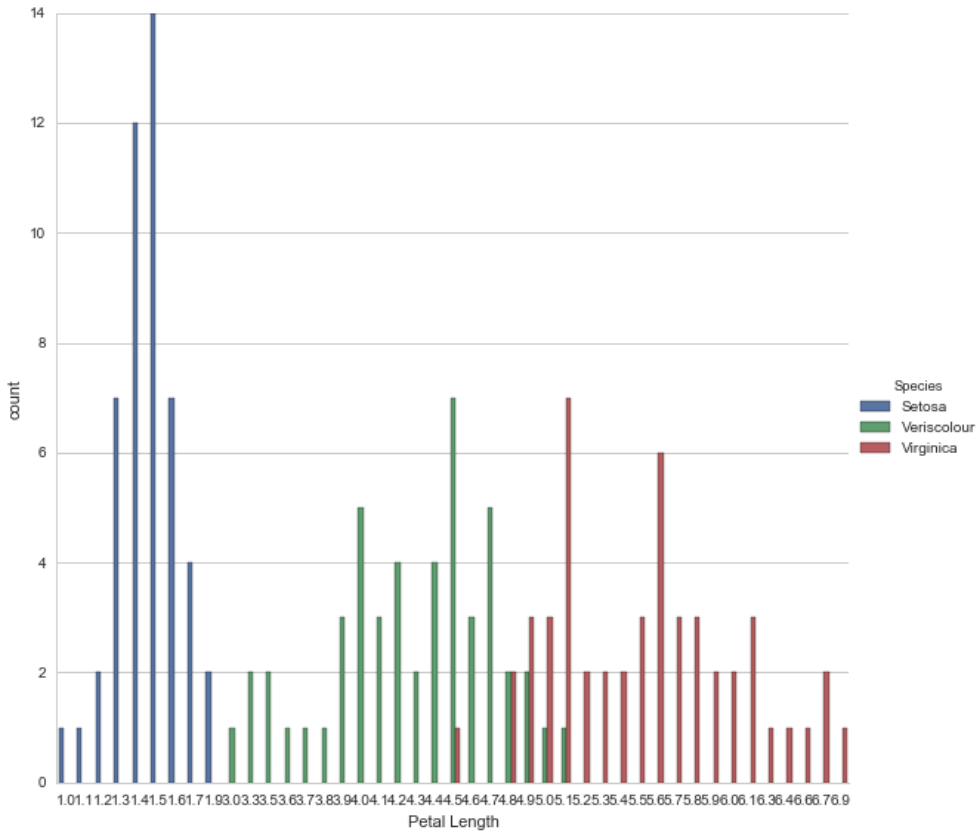
Seaborn's factorplot lets you choose between histograms, point plots, violin plots, etc.

Also, the "hue" argument makes it easy to compare multiple variables simultaneously.

Unfortunately, sorting columns appropriately can be a challenge.

The following example makes use of the Iris flower data set included in Seaborn:

```
xorder = np.apply_along_axis(sorted, 0, iris['Petal Length'].unique())
sns.factorplot('Petal Length', data=iris, order=xorder, size=8, hue='Species',
                        kind='count');          Note: without size=8, the x-axis labels overlap
```

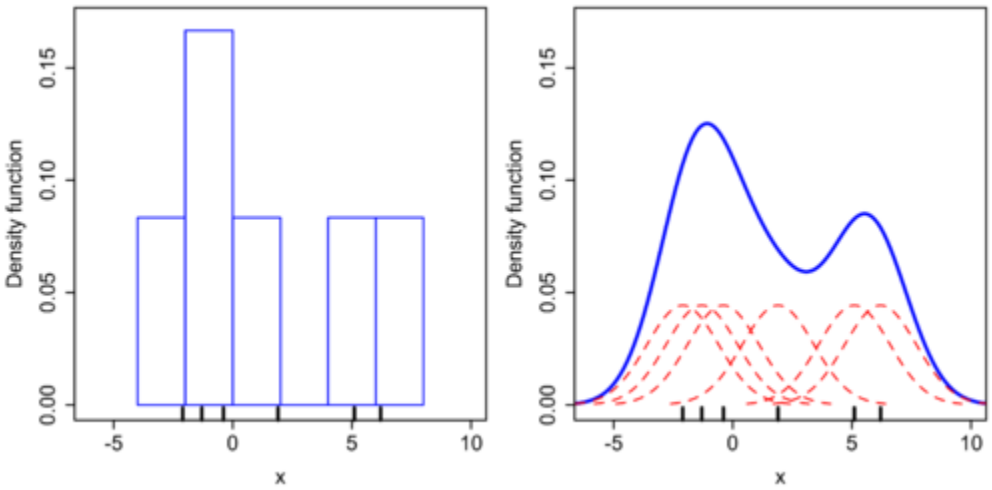


For more info: https://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.factorplot.html

KDE Plots (Kernel Density Estimation Plots)

KDE's are a tool for representing Probability Density Functions (PDF's)

For a full description of how to generate plots by hand and how easily Seaborn does it, refer to the Jupyter notebook



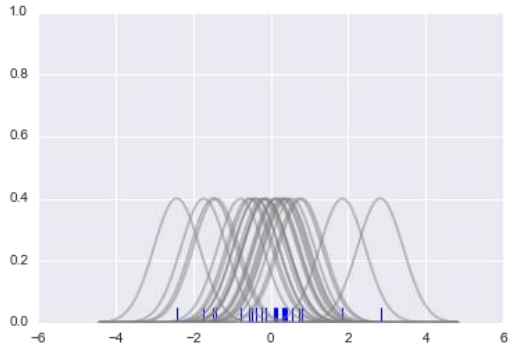
Source: https://en.wikipedia.org/wiki/Kernel_density_estimation

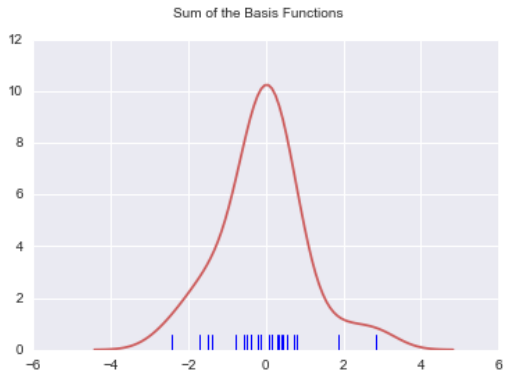
KDE Plots, cont'd

By hand:

```
dataset = randn(25)                          take a random (normal distribution) sample set.
```

```
sns.rugplot(dataset)                         make a rug plot
x_min = dataset.min() - 2                    Set up the x-axis for the plot
x_max = dataset.max() + 2
x_axis = np.linspace(x_min,x_max,100)        set 100 equally spaced points from x_min to x_max
bandwidth = ((4*dataset.std()**5)/(3*len(dataset)))**.2    =Silverman's rule of thumb
kernel_list = []                             Create an empty kernel list
for data_point in dataset:                   Plot each basis function
  # Create a kernel for each point and append to list
    kernel = stats.norm(data_point,bandwidth).pdf(x_axis)
    kernel_list.append(kernel)
  #Scale for plotting
    kernel = kernel / kernel.max()
    kernel = kernel * .4
    plt.plot(x_axis,kernel,color = 'grey',alpha=0.5)
plt.ylim(0,1)          SEE PLOT BELOW LEFT
```

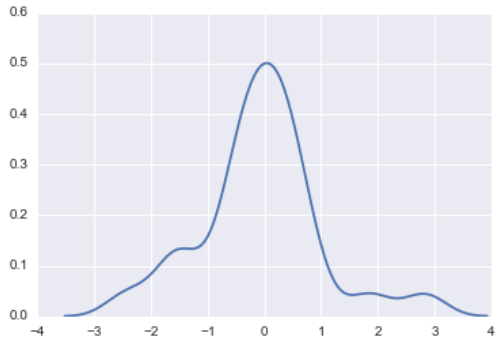




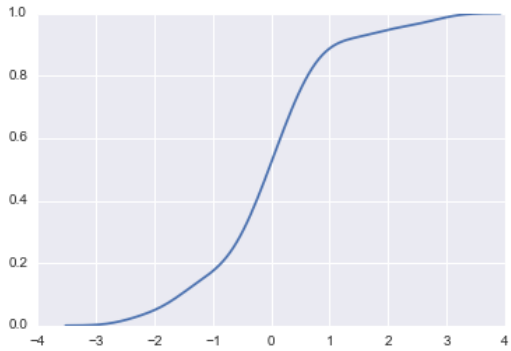

```
sum_of_kde = np.sum(kernel_list,axis=0)
fig = plt.plot(x_axis,sum_of_kde,color='indianred')
sns.rugplot(dataset)
plt.suptitle("Sum of the Basis Functions")   SEE PLOT ABOVE RIGHT
```

Using Seaborn:

```
sns.kdeplot(dataset)
```



```
sns.kdeplot(dataset,cumulative=True)
```



Cumulative Distribution Function (CDF)

Seaborn allows you to quickly change bandwidth, kernels, orientation, and a number of other parameters. Seaborn also supports multivariate density estimation. See jupyter notebook for more info.

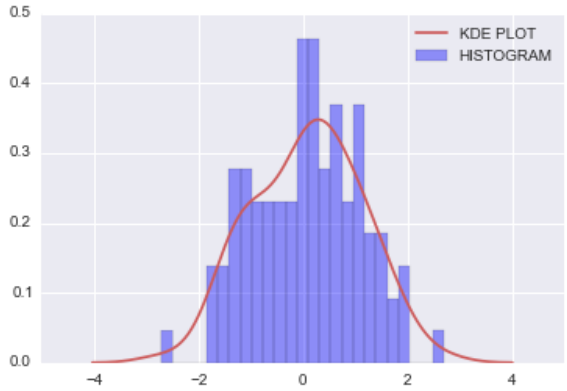
Combined Plots (kde, hist, rug) using distplot

```
sns.distplot(dataset,bins=25)
```
by default, a KDE over a histogram

```
sns.distplot(dataset,rug=True,hist=False)
```
here a rug and a KDE

To control specific plots in distplot, use a `[plot]_kws` argument with dictionaries:

```
sns.distplot(dataset,bins=25,
             kde_kws={'color':'indianred','label':'KDE PLOT'},
             hist_kws={'color':'blue','label':"HISTOGRAM"})
```



Seaborn's distplot can be used on Series as well as DataFrames

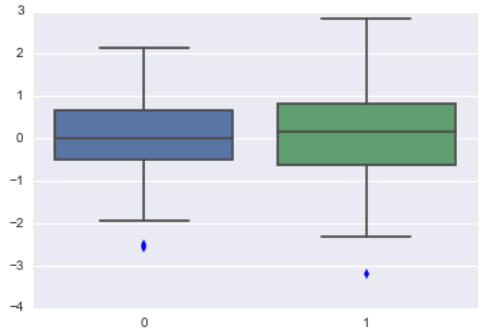
Box & Whisker Plots

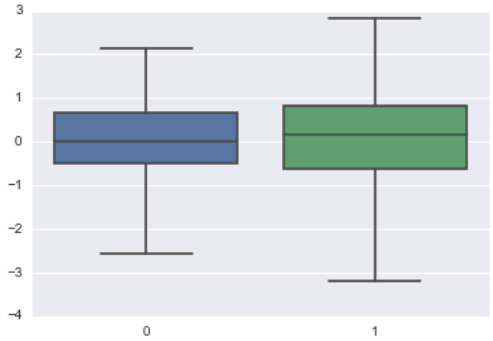
Box plots are another tool for representing Probability Density Functions (PDF's)

```
data1 = randn(100)
data2 = randn(100)
```

OLD: `sns.boxplot([data1,data2])`

NEW: `sns.boxplot(data=[data1,data2]);` with Seaborn v0.6.0





To absorb outliers into the whiskers (above right):

```
sns.boxplot(data=[data1,data2], whis=np.inf)
```

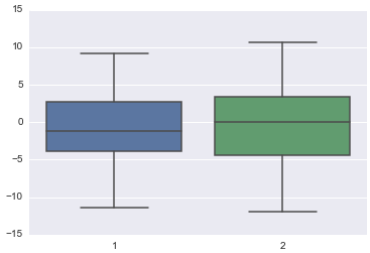
To set horizontal:

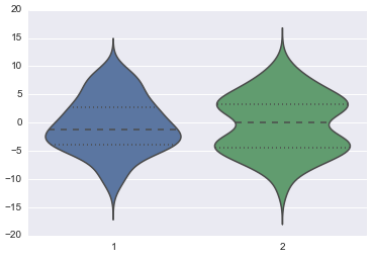
```
sns.boxplot([data1,data2], whis=np.inf, vert=False)
```

Violin Plots with `sns.violinplot(data=[data1,data2])`

May reveal what a box plot doesn't by incorporating some of the functionality of KDE plots

Refer to Jupyter notebook for an explanation of the math behind these two datasets.

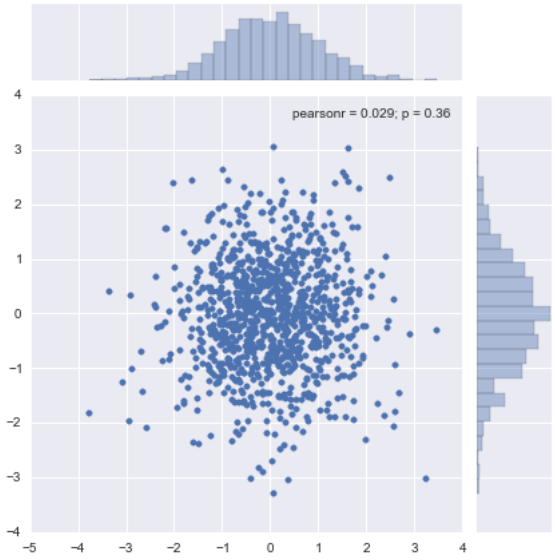




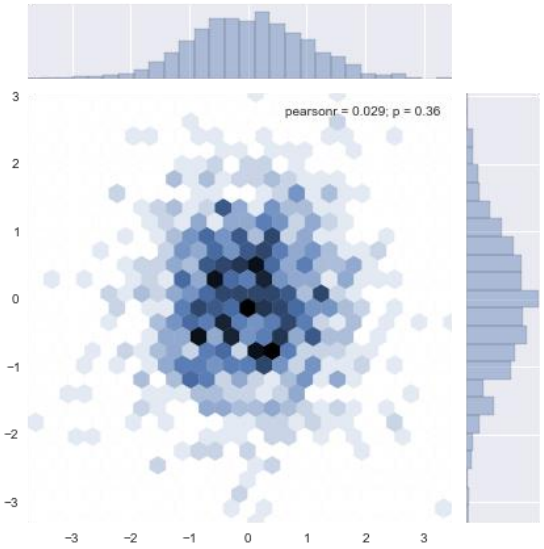
Joint Plots

```
data1 = randn(1000)
data2 = randn(1000)
```

```
sns.jointplot(data1,data2)
```




```
sns.jointplot(data1,data2,kind='hex')
```



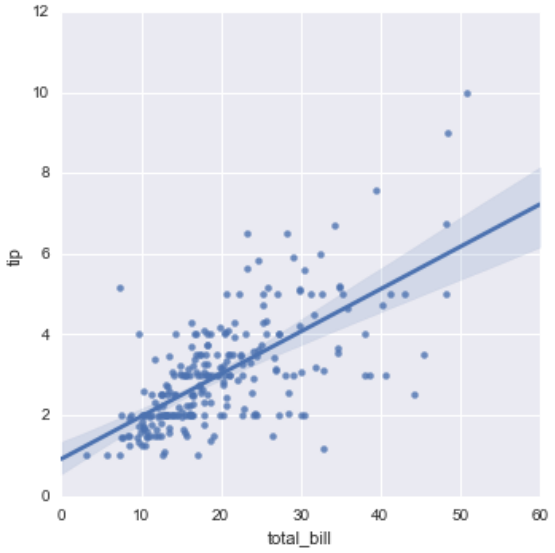
Regression Plots

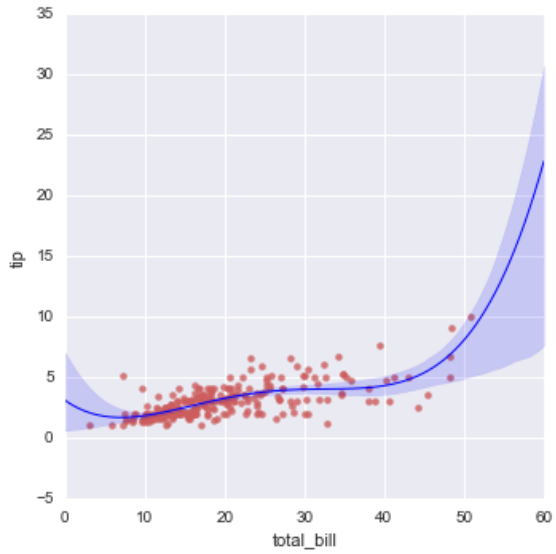
```
tips = sns.load_dataset("tips")
```

load a Seaborn sample dataset

```
sns.lmplot(x,y,data)
sns.lmplot("total_bill","tip",tips);
```

scatter plot with linear regression line & confidence interval






```
sns.lmplot("total_bill","tip",tips,order=4, scatter_kws={"color":"indianred"},
           line_kws={"linewidth": 1, "color": "blue"})
```

ABOVE RIGHT

Refer to the online documentation & jupyter notebook for more on adjusting the confidence interval, plotting discrete variables, jittering, removing the regression line, and using hue & markers to define subsets along a column.

Seaborn even supports loca regression (LOESS) with the argument `lowess=True`.

For lower level regression plots, use `sns.regplot(x,y,data)`. These can be tied to other plots.

Heatmaps

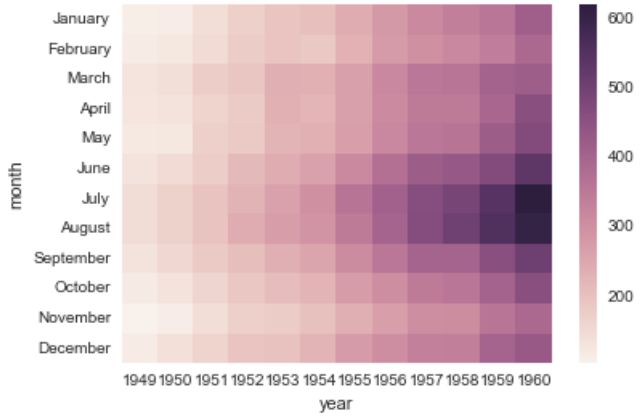
```
flight_dframe = sns.load_dataset('flights')
```
load a Seaborn sample dataset

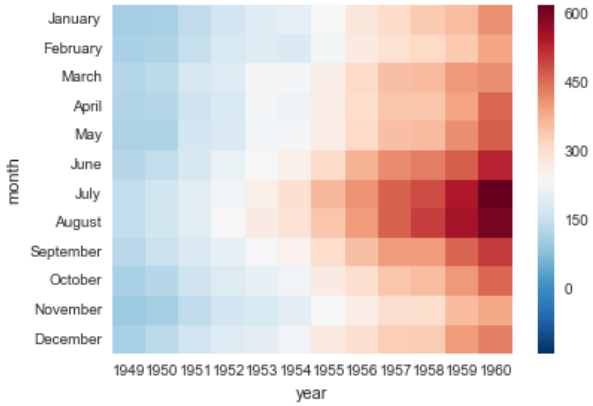
Pivot the data to make it more usable (index=month, columns=year, fill=passengers):

```
flight_dframe = flight_dframe.pivot("month","year","passengers")
```

Note: unlike the lecture notebook, dframe now sorts months in date order, not alphabetically.

```
sns.heatmap(flight_dframe);
```





<u>You can add fill data with</u>

```
sns.heatmap(flight_dframe,annot=True,fmt='d')
```

<u>You can specify a "center" for the colormap with</u>

```
sns.heatmap(flight_dframe,center=flight_dframe.loc['January',1955])
```
ABOVE RIGHT

<u>Heatmap() can be added onto a subplot axis to create more informative figures:</u>

```
f,(axis1,axis2) = plt.subplots(2,1)
```
figure "f" will have two rows, one column

Since yearly_flights is a weird format, we'll have to grab the values we want with a Series, then put them in a dframe

```
yearly_flights = flight_dframe.sum()
years = pd.Series(yearly_flights.index.values)
years = pd.DataFrame(years)
flights = pd.Series(yearly_flights.values)
flights = pd.DataFrame(flights)
```

Make the dframe and name columns

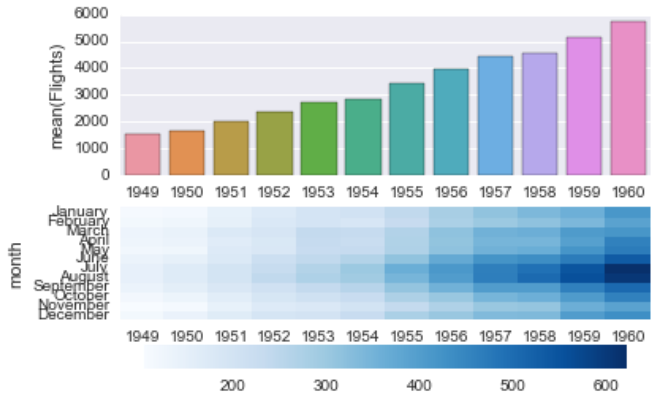
```
year_dframe = pd.concat((years,flights),axis=1)
year_dframe.columns = ['Year','Flights']
```

Create the bar plot on top

```
sns.barplot('Year',y='Flights',data=year_dframe, ax=axis1)
```

Create the heatmap on bottom

```
sns.heatmap(flight_dframe,cmap='Blues',ax=axis2,
      cbar_kws={"orientation": "horizontal"})
```
places the colorbar horizontally



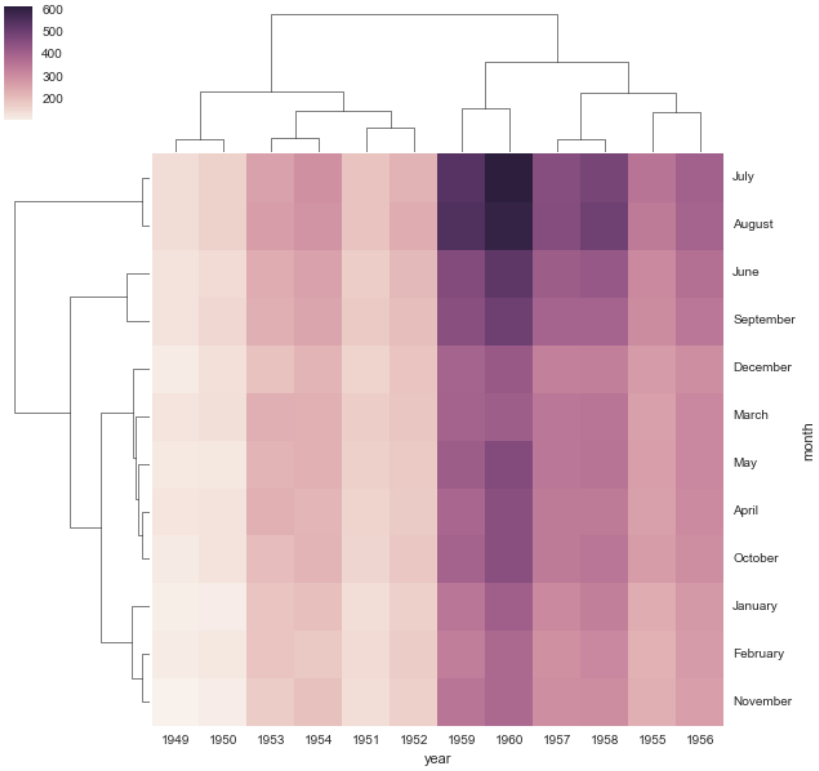
Clustered Matrices

In the lecture notebook, clustermaps helped reveal the summer trend, whereas the latest Seaborn version did so by default. Also, newer clustermaps aligned the month labels vertically instead of horizontally.

A workaround (suggested by a classmate):

```
cg = sns.clustermap(flight_dframe)
```
original code in lecture

```
plt.setp(cg.ax_heatmap.yaxis.get_majorticklabels(), rotation=0);
```




```
sns.clustermap(flight_dframe,col_cluster=False)
```
unclusters the columns

<u>You can set a standard scale (since the number of flights increase every year):</u>

```
sns.clustermap(flight_dframe,standard_scale=1)
```
standardize by columns (year)

```
sns.clustermap(flight_dframe,standard_scale=0)
```
…or standardize by rows (month)

<u>You can normalize rows by their Z-score:</u>

```
sns.clustermap(flight_dframe,z_score=1)
```

This subtracts the mean and divides by the STD of each column, so the rows have a mean of 0 and a variance of 1.

OTHER USEFUL TOOLS:

How to Save a DataFrame as a Figure (.png file)
http://stackoverflow.com/questions/19726663/how-to-save-the-pandas-dataframe-series-data-as-a-figure

How to open a webpage inside Jupyter

```
website = "http://docs.scipy.org/doc/numpy/reference/ufuncs.html#available-ufuncs"
import webbrowser
webbrowser.open(website)
```

How to watch a YouTube Video inside Jupyter

```
from IPython.display import YouTubeVideo
YouTubeVideo('UqYde-LULfs')
```

this is a brief tutorial on k-nearest neighbors