**PYTHON DATA VISUALIZATIONS**

Table of Contents

[**MATPLOTLIB** 2](#_Toc443140544)

[**Scatter plot** 2](#_Toc443140545)

[**Bar plot with errorbars** 3](#_Toc443140546)

[**3D Graphical Analysis:** 4](#_Toc443140547)

[**Histograms:** 5](#_Toc443140548)

[**SEABORN LIBRARIES** 6](#_Toc443140549)

[**Rug Plots** 6](#_Toc443140550)

[**Histograms using factorplot** 7](#_Toc443140551)

[**Combined Plots (kde, hist, rug) using distplot** 9](#_Toc443140552)

[**Box & Whisker Plots** 9](#_Toc443140553)

[**Violin Plots** 9](#_Toc443140554)

[**Joint Plots** 10](#_Toc443140555)

[**Regression Plots** 10](#_Toc443140556)

[**Heatmaps** 11](#_Toc443140557)

[**Clustered Matrices** 12](#_Toc443140558)

[**OTHER USEFUL TOOLS:** 13](#_Toc443140559)

[**How to Save a DataFrame as a Figure (.png file)** 13](#_Toc443140560)

[**How to open a webpage inside Jupyter** 13](#_Toc443140561)

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

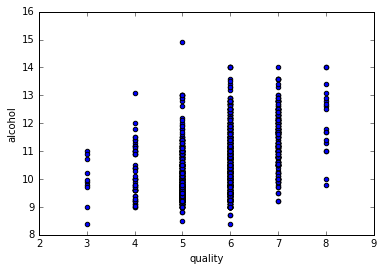
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/'

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 did not have to import matplotlib

dframe\_wine.plot(kind='scatter',x='quality',y='alcohol');



**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,

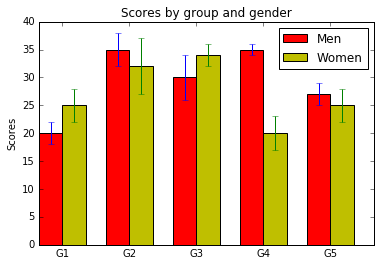
'%d' % int(height),

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

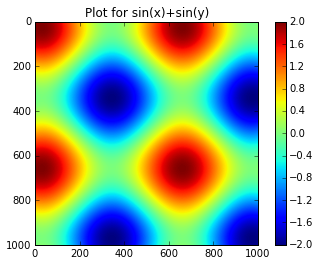
dx,dy=np.meshgrid(points,points) create the grid

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 for generating random number datasets (normal distribution)

import matplotlib.pyplot as plt

%matplotlib inline so that plots appear in the iPython Notebook

dataset1 = randn(100)

plt.hist(dataset1)

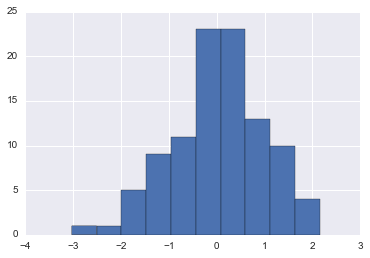
(array([ 1., 1., 5., 9., 11., 23., 23., 13., 10., 4.]),

array([**-3.03051447**, -2.5119968 , -1.99347913, -1.47496147, -0.9564438 ,

-0.43792613, 0.08059154, 0.5991092 , 1.11762687, 1.63614454,

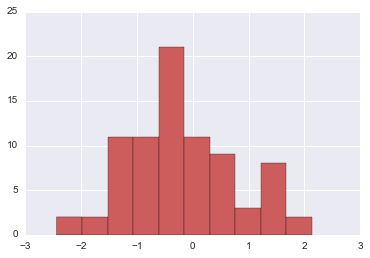
**2.1546622** ]),

<a list of 10 Patch objects>)



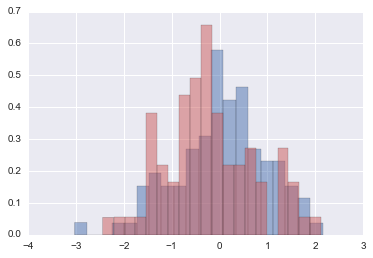
data grouped into 10 bins by default,   
with 11 equally spaced borders (min to max)

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

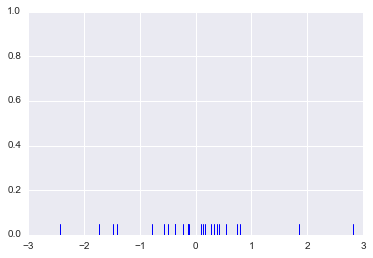
%matplotlib inline so that plots appear in the iPython Notebook

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.

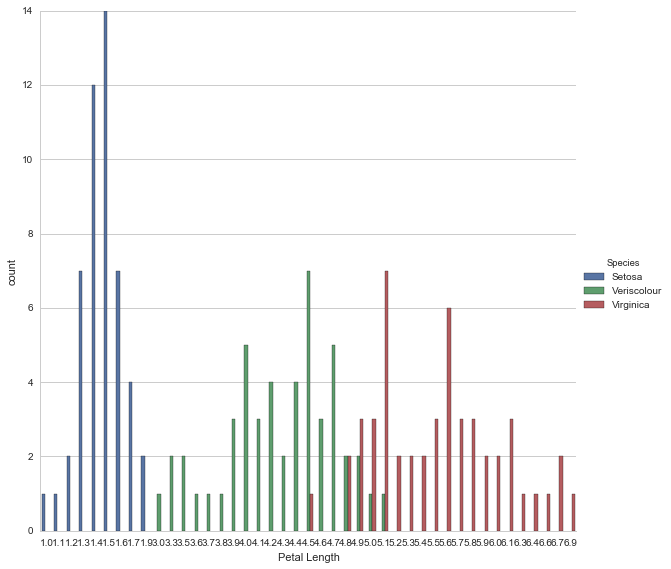
Unfortunately, sorting columns appropriately can be a challenge.

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

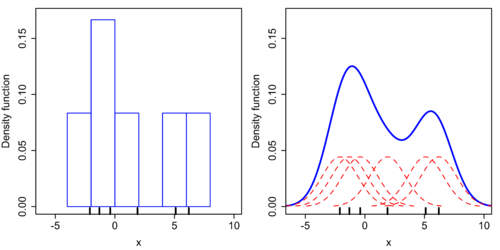


For more info: <https://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.factorplot.html>

**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)

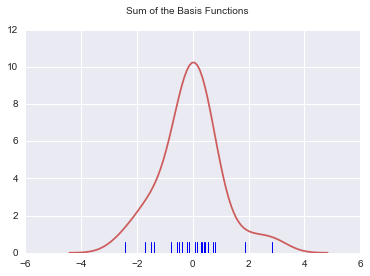
#Scale for plotting

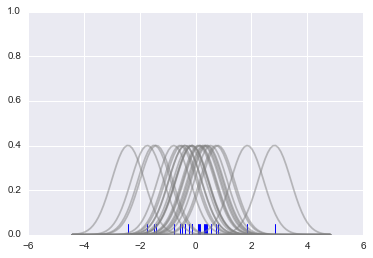
kernel = kernel / kernel.max()

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)

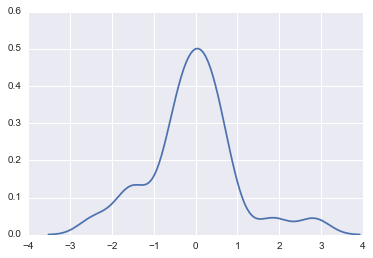
fig = plt.plot(x\_axis,sum\_of\_kde,color='indianred')

sns.rugplot(dataset)

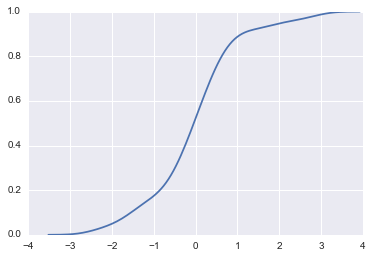
plt.suptitle("Sum of the Basis Functions") SEE PLOT ABOVE RIGHT

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

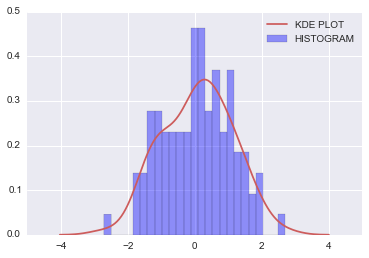
sns.distplot(dataset,rug=True,hist=False) here a rug and a KDE

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**

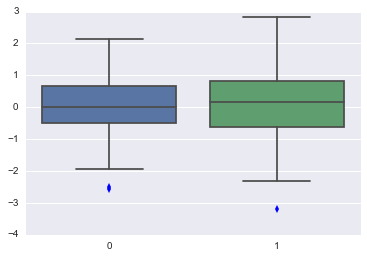
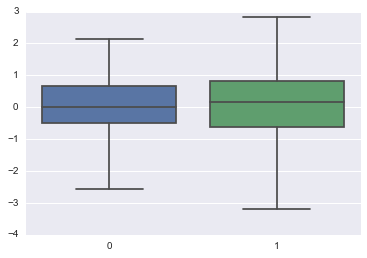
Box plots are another tool for representing Probability Density Functions (PDF's)

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:

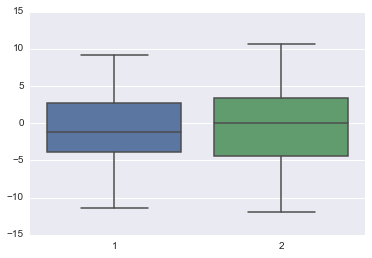
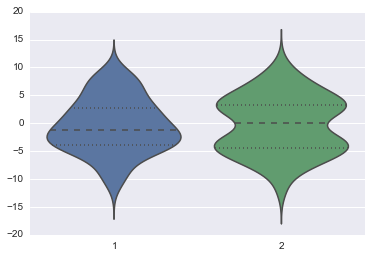
sns.boxplot([data1,data2], whis=np.inf, vert=False)

**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.

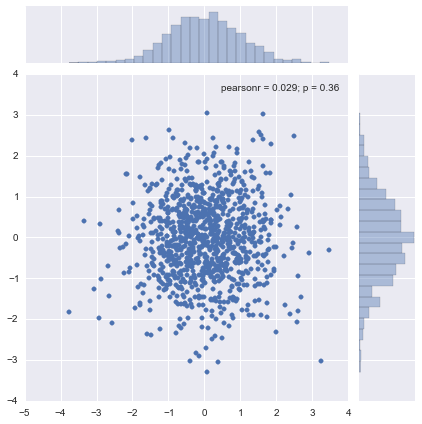
 

**Joint Plots**

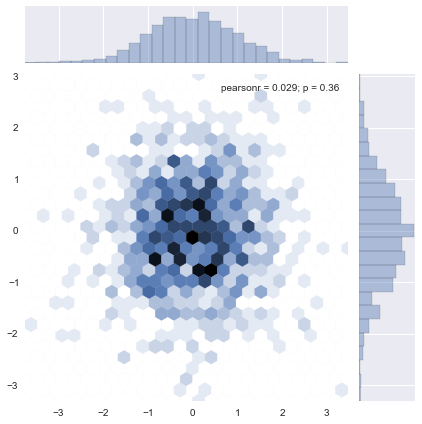
data1 = randn(1000)

data2 = randn(1000)

sns.jointplot(data1,data2)



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

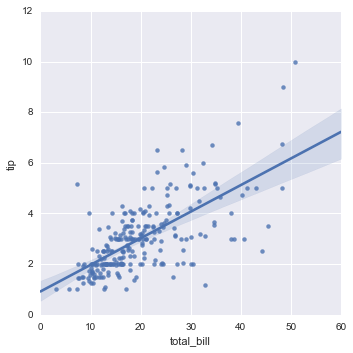
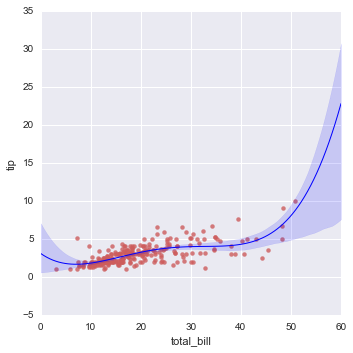


**Regression Plots**

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**

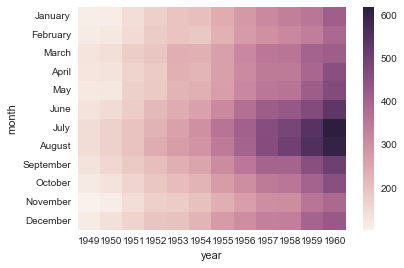
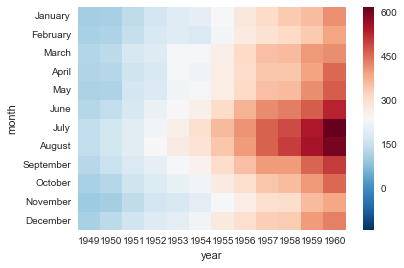
flight\_dframe = sns.load\_dataset('flights') load a Seaborn sample dataset

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);



You can add fill data with

sns.heatmap(flight\_dframe,annot=True,fmt='d')

You can specify a "center" for the colormap with sns.heatmap(flight\_dframe,center=flight\_dframe.loc['January',1955]) **ABOVE RIGHT**

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

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)

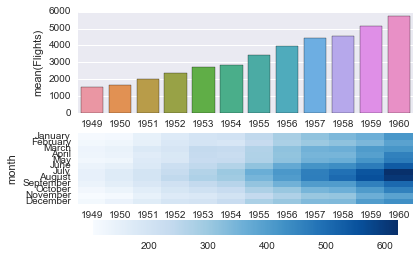
year\_dframe.columns = ['Year','Flights']

*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



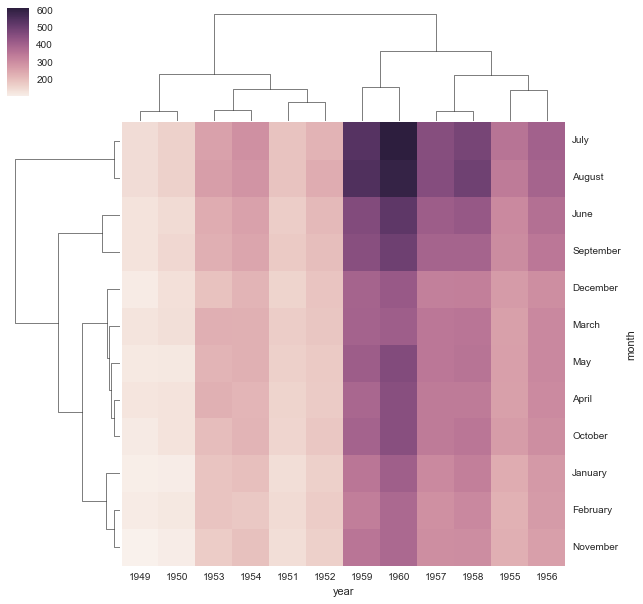
**Clustered Matrices**

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

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

sns.clustermap(flight\_dframe,standard\_scale=1) standardize by columns (year)

sns.clustermap(flight\_dframe,standard\_scale=0) …or standardize by rows (month)

You can normalize rows by their Z-score:

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