04-Finance Project - Solutions

September 2, 2019

___

1 Finance Data Project - Solutions

In this data project we will focus on exploratory data analysis of stock prices. Keep in mind, this project is just meant to practice your visualization and pandas skills, it is not meant to be a robust financial analysis or be taken as financial advice. ____ ** NOTE: This project is extremely challenging because it will introduce a lot of new concepts and have you looking things up on your own (we'll point you in the right direction) to try to solve the tasks issued. Feel free to just go through the solutions lecture notebook and video as a "walkthrough" project if you don't want to have to look things up yourself. You'll still learn a lot that way! ** ____ We'll focus on bank stocks and see how they progressed throughout the financial crisis all the way to early 2016.

1.1 Get the Data

In this section we will learn how to use pandas to directly read data from Google finance using pandas!

First we need to start with the proper imports, which we've already laid out for you here.

*Note: You'll need to install pandas-datareader for this to work! Pandas datareader allows you to read stock information directly from the internet Use these links for install guidance (**pip install pandas-datareader**), or just follow along with the video lecture.*

1.1.1 The Imports

Already filled out for you.

```
In [1]: from pandas_datareader import data, wb
        import pandas as pd
        import numpy as np
        import datetime
        %matplotlib inline
```

1.2 Data

We need to get data using pandas datareader. We will get stock information for the following banks: * Bank of America * CitiGroup * Goldman Sachs * JPMorgan Chase * Morgan Stanley * Wells Fargo

** Figure out how to get the stock data from Jan 1st 2006 to Jan 1st 2016 for each of these banks. Set each bank to be a separate dataframe, with the variable name for that bank being its ticker symbol. This will involve a few steps:** 1. Use datetime to set start and end datetime objects. 2. Figure out the ticker symbol for each bank. 2. Figure out how to use datareader to grab info on the stock.

** Use this documentation page for hints and instructions (it should just be a matter of replacing certain values. Use google finance as a source, for example:**

```
# Bank of America
BAC = data.DataReader("BAC", 'google', start, end)
```

1.2.1 WARNING: MAKE SURE TO CHECK THE LINK ABOVE FOR THE LATEST WORKING API. "google" MAY NOT ALWAYS WORK.

```
In [2]: start = datetime.datetime(2006, 1, 1)
        end = datetime.datetime(2016, 1, 1)
```

```
In [3]: # Bank of America
        BAC = data.DataReader("BAC", 'google', start, end)

        # CitiGroup
        C = data.DataReader("C", 'google', start, end)

        # Goldman Sachs
        GS = data.DataReader("GS", 'google', start, end)

        # JPMorgan Chase
        JPM = data.DataReader("JPM", 'google', start, end)

        # Morgan Stanley
        MS = data.DataReader("MS", 'google', start, end)

        # Wells Fargo
        WFC = data.DataReader("WFC", 'google', start, end)
```

```
In [4]: # Could also do this for a Panel Object
        df = data.DataReader(['BAC', 'C', 'GS', 'JPM', 'MS', 'WFC'],'google', start, end)
```

** Create a list of the ticker symbols (as strings) in alphabetical order. Call this list: tickers**

```
In [5]: tickers = ['BAC', 'C', 'GS', 'JPM', 'MS', 'WFC']
```

** Use pd.concat to concatenate the bank dataframes together to a single data frame called bank_stocks. Set the keys argument equal to the tickers list. Also pay attention to what axis you concatenate on.**

```
In [6]: bank_stocks = pd.concat([BAC, C, GS, JPM, MS, WFC],axis=1,keys=tickers)
```

** Set the column name levels (this is filled out for you):**

```
In [7]: bank_stocks.columns.names = ['Bank Ticker','Stock Info']
```

** Check the head of the bank_stocks dataframe.**

```
In [8]: bank_stocks.head()
```

```
Out[8]: Bank Ticker    BAC                                          C                          \
        Stock Info    Open   High    Low  Close    Volume   Open   High    Low  Close
        Date
        2006-01-03   46.92  47.18  46.15  47.08  16296700  490.0  493.8  481.1  492.9
        2006-01-04   47.00  47.24  46.45  46.58  17757900  488.6  491.0  483.5  483.8
        2006-01-05   46.58  46.83  46.32  46.64  14970900  484.4  487.8  484.0  486.2
        2006-01-06   46.80  46.91  46.35  46.57  12599800  488.8  489.0  482.0  486.2
        2006-01-09   46.72  46.97  46.36  46.60  15620000  486.0  487.4  483.0  483.9

        Bank Ticker              ...       MS                                    WFC   \
        Stock Info     Volume    ...      Open   High    Low  Close   Volume   Open
        Date                     ...
        2006-01-03   1537660     ...     57.17  58.49  56.74  58.31  5377000  31.60
        2006-01-04   1871020     ...     58.70  59.28  58.35  58.35  7977800  31.80
        2006-01-05   1143160     ...     58.55  58.59  58.02  58.51  5778000  31.50
        2006-01-06   1370250     ...     58.77  58.85  58.05  58.57  6889800  31.58
        2006-01-09   1680740     ...     58.63  59.29  58.62  59.19  4144500  31.68

        Bank Ticker
        Stock Info    High    Low  Close    Volume
        Date
        2006-01-03   31.98  31.20  31.90  11016400
        2006-01-04   31.82  31.36  31.53  10871000
        2006-01-05   31.56  31.31  31.50  10158000
        2006-01-06   31.78  31.38  31.68   8403800
        2006-01-09   31.82  31.56  31.68   5619600

        [5 rows x 30 columns]
```

2 EDA

Let's explore the data a bit! Before continuing, I encourage you to check out the documentation on Multi-Level Indexing and Using .xs. Reference the solutions if you can not figure out how to use .xs(), since that will be a major part of this project.

** What is the max Close price for each bank's stock throughout the time period?**

```
In [9]: bank_stocks.xs(key='Close',axis=1,level='Stock Info').max()
```

```
Out[9]: Bank Ticker
        BAC      54.90
        C       564.10
        GS      247.92
        JPM      70.08
        MS       89.30
        WFC      58.52
        dtype: float64
```

** Create a new empty DataFrame called returns. This dataframe will contain the returns for each bank's stock. returns are typically defined by:**

$$r_t = \frac{p_t - p_{t-1}}{p_{t-1}} = \frac{p_t}{p_{t-1}} - 1$$

```
In [10]: returns = pd.DataFrame()
```

** We can use pandas pct_change() method on the Close column to create a column representing this return value. Create a for loop that goes and for each Bank Stock Ticker creates this returns column and set's it as a column in the returns DataFrame.**

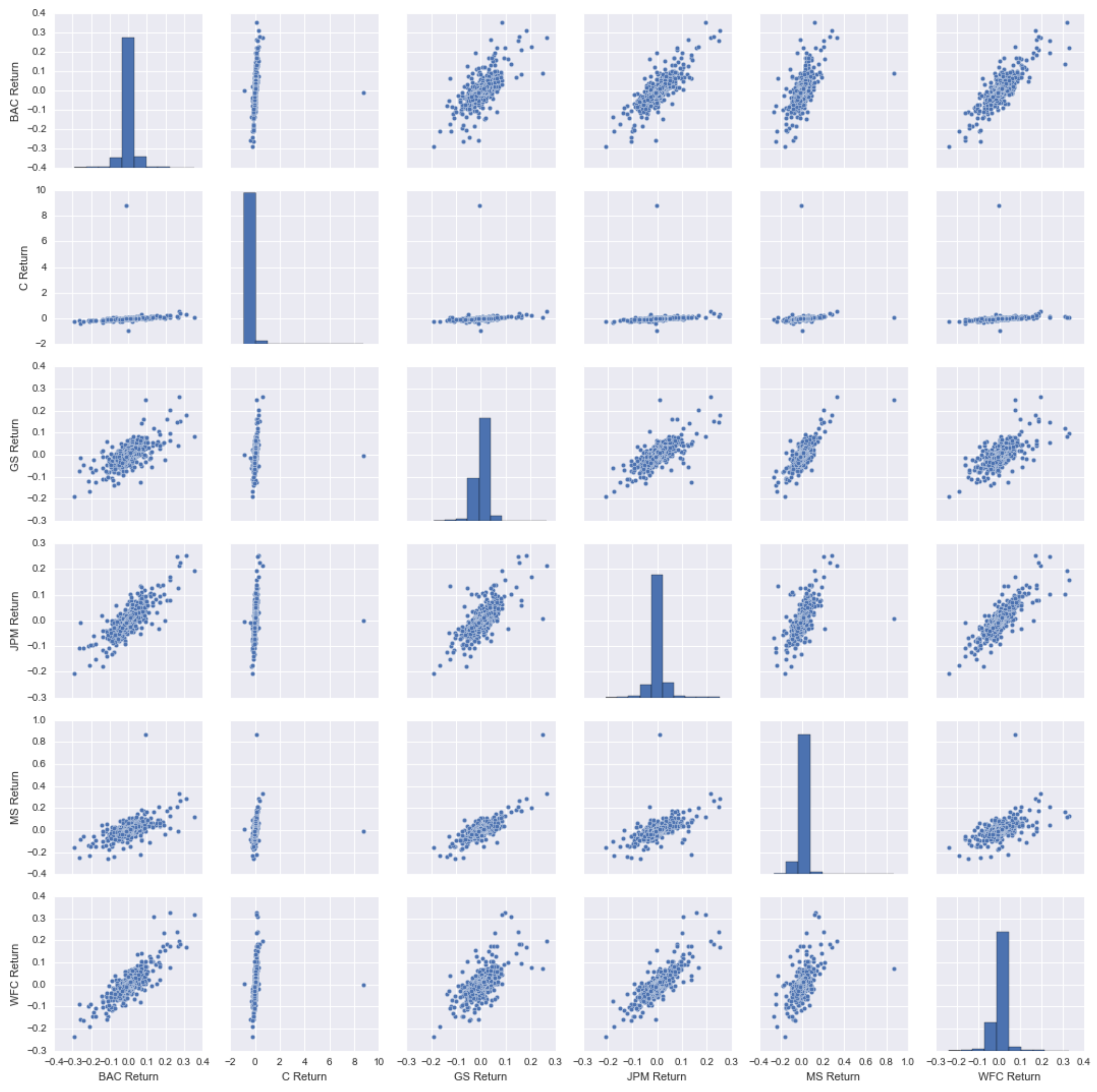
```
In [11]: for tick in tickers:
             returns[tick+' Return'] = bank_stocks[tick]['Close'].pct_change()
         returns.head()
```

```
Out[11]:             BAC Return  C Return  GS Return  JPM Return  MS Return  WFC Return
         Date
         2006-01-03         NaN       NaN        NaN         NaN        NaN         NaN
         2006-01-04   -0.010620 -0.018462  -0.013812   -0.014183   0.000686   -0.011599
         2006-01-05    0.001288  0.004961  -0.000393    0.003029   0.002742   -0.000951
         2006-01-06   -0.001501  0.000000   0.014169    0.007046   0.001025    0.005714
         2006-01-09    0.000644 -0.004731   0.012030    0.016242   0.010586    0.000000
```

** Create a pairplot using seaborn of the returns dataframe. What stock stands out to you? Can you figure out why?**

```
In [13]: #returns[1:]
         import seaborn as sns
         sns.pairplot(returns[1:])
```

```
Out[13]: <seaborn.axisgrid.PairGrid at 0x113fb4da0>
```



Background on Citigroup's Stock Crash available here.

You'll also see the enormous crash in value if you take a look a the stock price plot (which we do later in the visualizations.)

** Using this returns DataFrame, figure out on what dates each bank stock had the best and worst single day returns. You should notice that 4 of the banks share the same day for the worst drop, did anything significant happen that day?**

```
In [14]: # Worst Drop (4 of them on Inauguration day)
         returns.idxmin()
```

```
Out[14]: BAC Return    2009-01-20
         C Return      2011-05-06
         GS Return     2009-01-20
         JPM Return    2009-01-20
         MS Return    2008-10-09
         WFC Return   2009-01-20
         dtype: datetime64[ns]
```

** You should have noticed that Citigroup's largest drop and biggest gain were very close to one another, did anythign significant happen in that time frame? **

Citigroup had a stock split.

```
In [15]: # Best Single Day Gain
         # citigroup stock split in May 2011, but also JPM day after inauguration.
         returns.idxmax()

Out[15]: BAC Return   2009-04-09
         C Return     2011-05-09
         GS Return    2008-11-24
         JPM Return   2009-01-21
         MS Return    2008-10-13
         WFC Return   2008-07-16
         dtype: datetime64[ns]
```

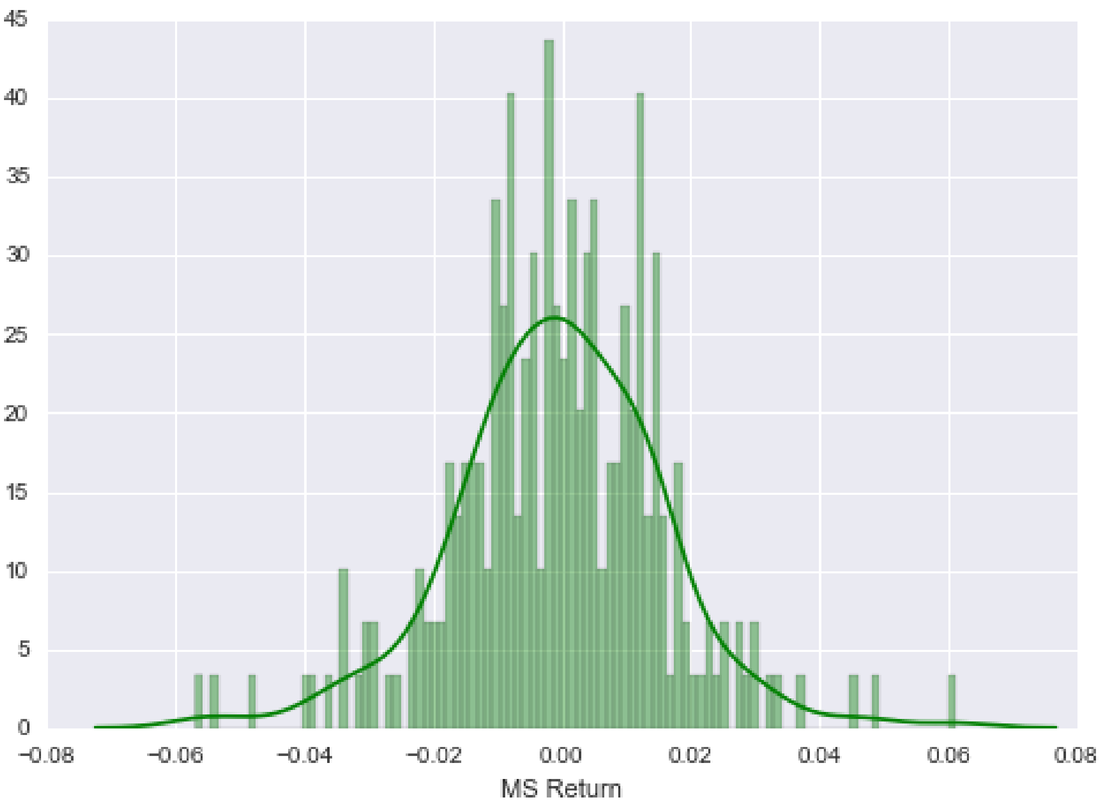
** Take a look at the standard deviation of the returns, which stock would you classify as the riskiest over the entire time period? Which would you classify as the riskiest for the year 2015?**

```
In [16]: returns.std() # Citigroup riskiest

Out[16]: BAC Return    0.036650
         C Return      0.179969
         GS Return     0.025346
         JPM Return    0.027656
         MS Return     0.037820
         WFC Return    0.030233
         dtype: float64

In [17]: returns.ix['2015-01-01':'2015-12-31'].std() # Very similar risk profiles, but Morgan

Out[17]: BAC Return    0.016163
         C Return      0.015289
         GS Return     0.014046
         JPM Return    0.014017
         MS Return     0.016249
         WFC Return    0.012591
         dtype: float64
```

** Create a distplot using seaborn of the 2015 returns for Morgan Stanley **

```
In [18]: sns.distplot(returns.ix['2015-01-01':'2015-12-31']['MS Return'],color='green',bins=10

/Users/marci/anaconda/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:20: Vi
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x11cc84828>
```

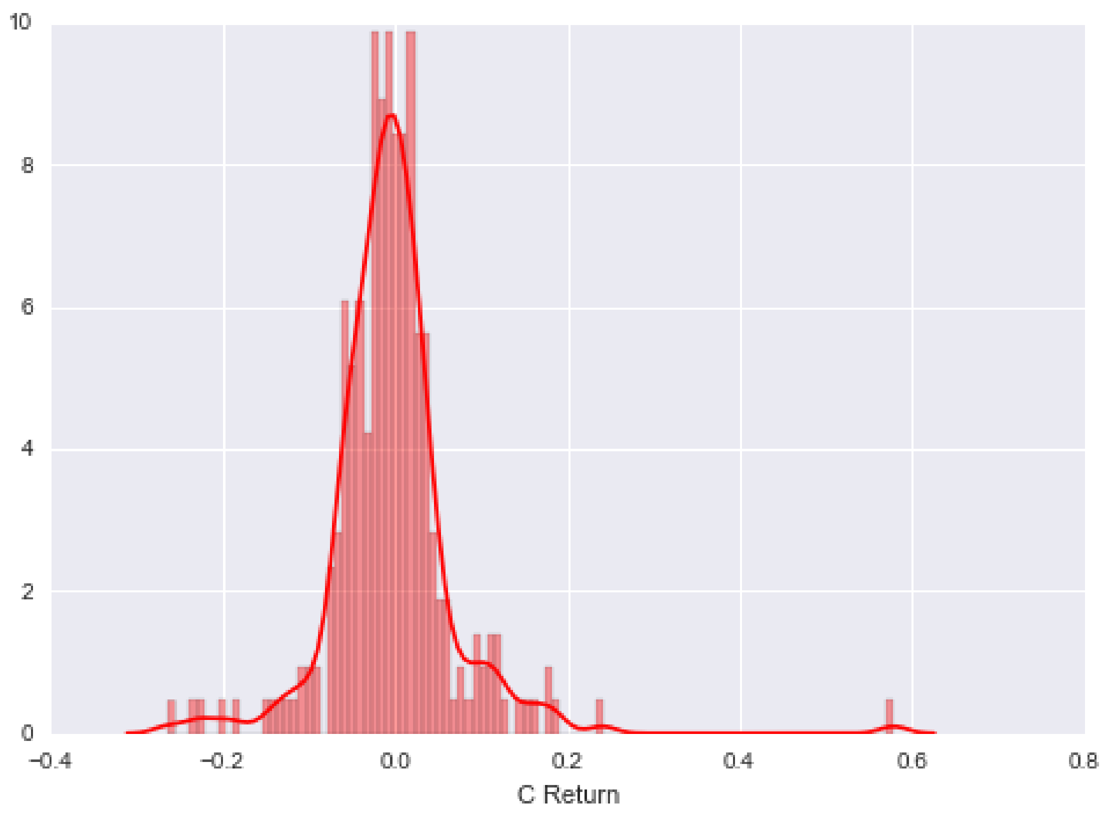


** Create a distplot using seaborn of the 2008 returns for CitiGroup **

```
In [19]: sns.distplot(returns.ix['2008-01-01':'2008-12-31']['C Return'],color='red',bins=100)
```

```
/Users/marci/anaconda/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:20: Vi
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x11efb9518>
```



---

3 More Visualization

A lot of this project will focus on visualizations. Feel free to use any of your preferred visualization libraries to try to recreate the described plots below, seaborn, matplotlib, plotly and cufflinks, or just pandas.

3.0.1 Imports

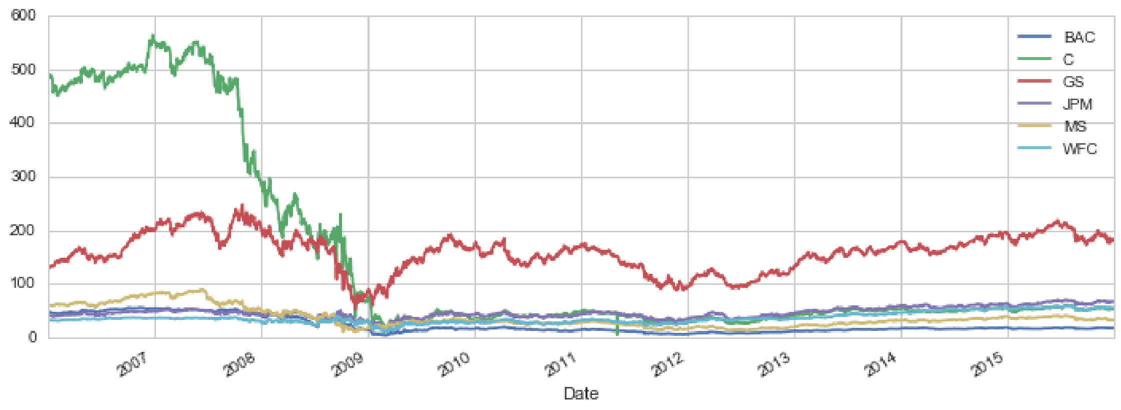
```
In [20]: import matplotlib.pyplot as plt
         import seaborn as sns
         sns.set_style('whitegrid')
         %matplotlib inline

         # Optional Plotly Method Imports
         import plotly
         import cufflinks as cf
         cf.go_offline()
```

```
<IPython.core.display.HTML object>
```

** Create a line plot showing Close price for each bank for the entire index of time. (Hint: Try using a for loop, or use .xs to get a cross section of the data.)**

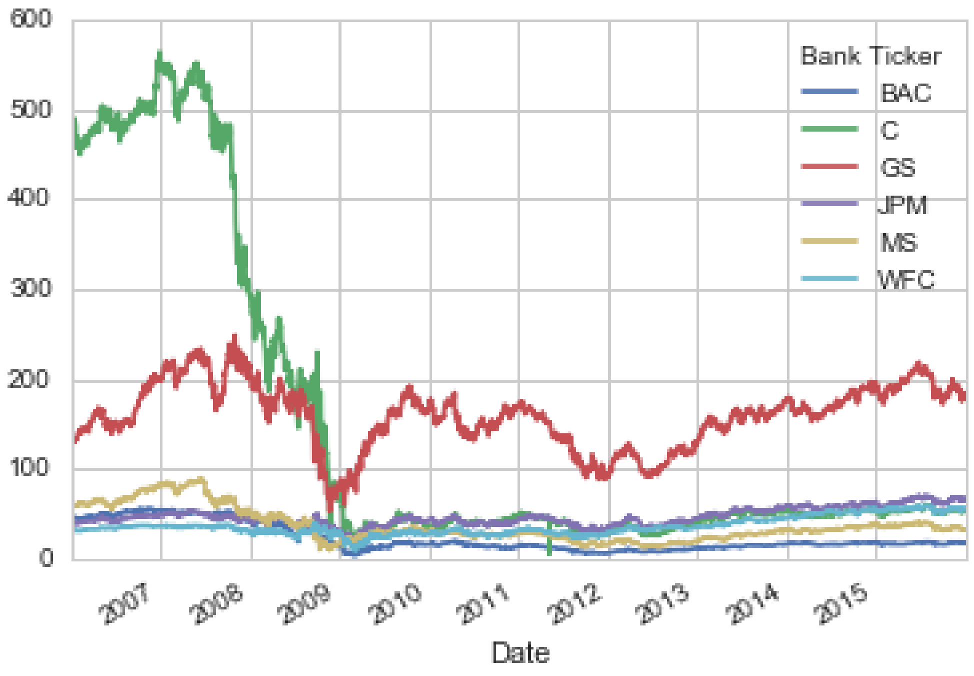
```
In [21]: for tick in tickers:
             bank_stocks[tick]['Close'].plot(figsize=(12,4),label=tick)
         plt.legend()
```

```
Out[21]: <matplotlib.legend.Legend at 0x116137748>
```



```
In [22]: bank_stocks.xs(key='Close',axis=1,level='Stock Info').plot()
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x11f7bd908>
```



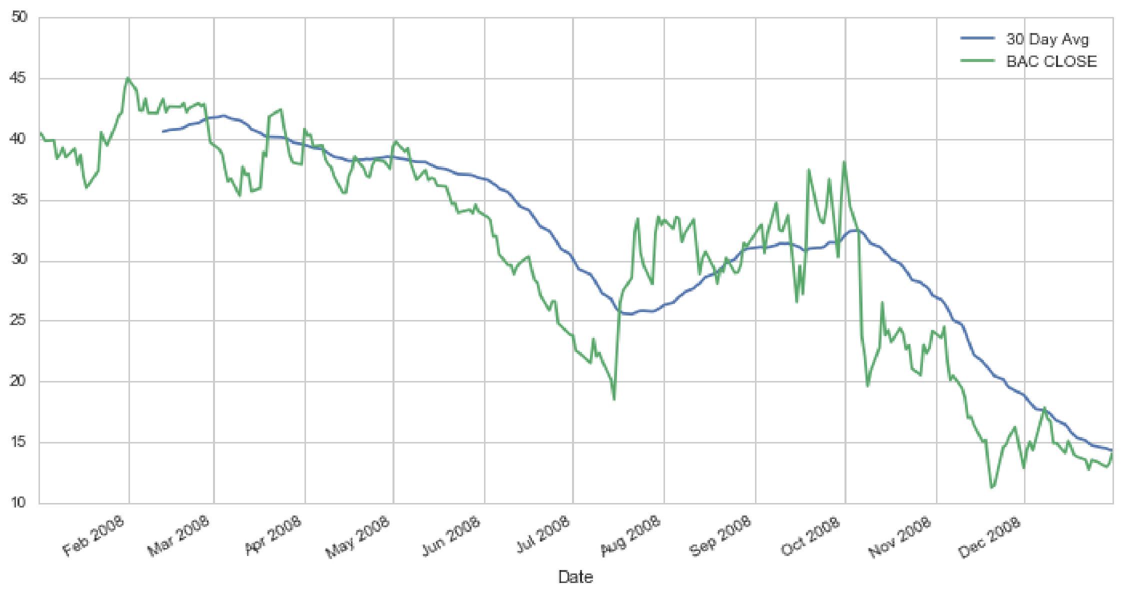
```
In [23]: # plotly
         bank_stocks.xs(key='Close',axis=1,level='Stock Info').iplot()

<IPython.core.display.HTML object>
```

3.1 Moving Averages

Let's analyze the moving averages for these stocks in the year 2008.

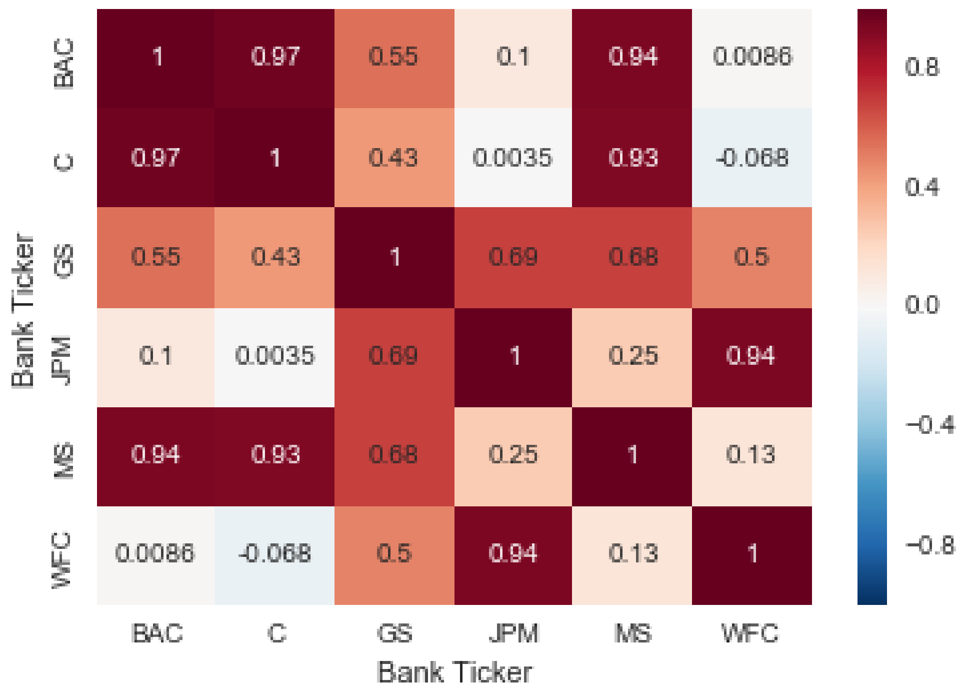
** Plot the rolling 30 day average against the Close Price for Bank Of America's stock for the year 2008**

```
In [24]: plt.figure(figsize=(12,6))
         BAC['Close'].ix['2008-01-01':'2009-01-01'].rolling(window=30).mean().plot(label='30 Da
         BAC['Close'].ix['2008-01-01':'2009-01-01'].plot(label='BAC CLOSE')
         plt.legend()

Out[24]: <matplotlib.legend.Legend at 0x11f966cf8>
```



** Create a heatmap of the correlation between the stocks Close Price.**

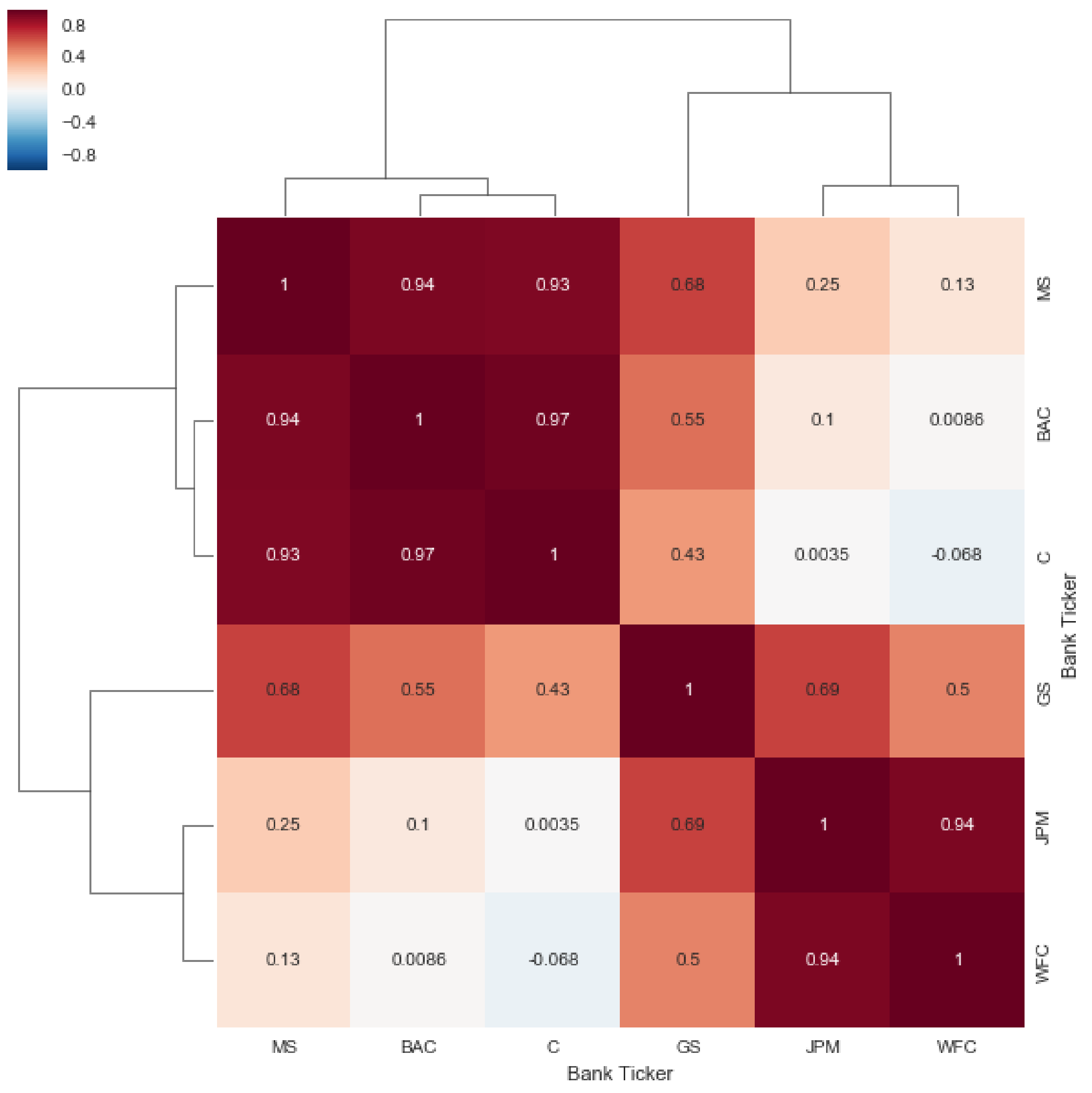
```
In [25]: sns.heatmap(bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr(),annot=True)

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x12045e2b0>
```



** Optional: Use seaborn's clustermap to cluster the correlations together:**

```
In [26]: sns.clustermap(bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr(),annot=Tru
```

```
Out[26]: <seaborn.matrix.ClusterGrid at 0x1204755c0>
```



```
In [27]: close_corr = bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr()
         close_corr.iplot(kind='heatmap',colorscale='rdylbu')
```

```
<IPython.core.display.HTML object>
```

4 Part 2 (Optional)

In this second part of the project we will rely on the cufflinks library to create some Technical Analysis plots. This part of the project is experimental due to its heavy reliance on the cuffinks project, so feel free to skip it if any functionality is broken in the future.

** Use .iplot(kind='candle) to create a candle plot of Bank of America's stock from Jan 1st 2015 to Jan 1st 2016.**

```
In [28]: BAC[['Open', 'High', 'Low', 'Close']].ix['2015-01-01':'2016-01-01'].iplot(kind='candl
```

```
<IPython.core.display.HTML object>
```

** Use .ta_plot(study='sma') to create a Simple Moving Averages plot of Morgan Stanley for the year 2015.**

```
In [29]: MS['Close'].ix['2015-01-01':'2016-01-01'].ta_plot(study='sma',periods=[13,21,55],titl
```

```
<IPython.core.display.HTML object>
```

Use .ta_plot(study='boll') to create a Bollinger Band Plot for Bank of America for the year 2015.

```
In [30]: BAC['Close'].ix['2015-01-01':'2016-01-01'].ta_plot(study='boll')
```

```
<IPython.core.display.HTML object>
```

5 Great Job!

Definitely a lot of more specific finance topics here, so don't worry if you didn't understand them all! The only thing you should be concerned with understanding are the basic pandas and visualization oeprations.