Finance Data Project

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.

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.*

The Imports

Already filled out for you.

In [1]:
```
#pip install pandas_datareader
```

In [2]:
```
import pandas_datareader.data as web

import pandas as pd
import numpy as np
import datetime
%matplotlib inline
```

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.
3. 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)
```

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

We also provide pickle file in the article lecture right before the video lectures.

In [3]:
```
start = datetime.datetime(2006, 1, 1)
end = datetime.datetime(2016, 1, 1)
```

In [4]:
```
# Scraping data from internet
# Bank of America
BAC = web.DataReader("BAC", 'yahoo', start, end)

# CitiGroup
C = web.DataReader("C", 'yahoo', start, end)

# Goldman Sachs
GS = web.DataReader("GS", 'yahoo', start, end)

# JPMorgan Chase
JPM = web.DataReader("JPM", 'yahoo', start, end)

# Morgan Stanley
MS = web.DataReader("MS", 'yahoo', start, end)

# Wells Fargo
WFC = web.DataReader("WFC", 'yahoo', start, end)
```

In [5]:
```
# Could also do this for a Panel Object
df = web.DataReader(['BAC', 'C', 'GS', 'JPM', 'MS', 'WFC'],'yahoo', start, end)
```

In [6]:
```
df
```

Out[6]:

Attributes	Adj Close						Close				...			Open			
Symbols	BAC	C	GS	JPM	MS	WFC	BAC	C	GS	JPM	...	GS	JPM	MS	WFC	BAC	C
Date																	
2006-01-03	34.106041	370.934509	104.412071	26.663109	36.364140	20.523809	47.080002	492.899994	128.869995	40.189999	...	126.699997	39.830002	57.169998	31.600000	16296700.0	1537600.0
2006-01-04	33.743816	364.086212	102.969894	26.509205	36.389069	20.285755	46.580002	483.799988	127.089996	39.619999	...	127.349998	39.779999	58.700001	31.799999	17757900.0	1870960.0
2006-01-05	33.787266	365.892365	102.929398	26.589512	36.488869	20.263241	46.639999	486.200012	127.040001	39.740002	...	126.000000	39.610001	58.549999	31.500000	14970700.0	1143160.0
2006-01-06	33.736572	365.892365	104.387749	26.776842	36.526283	20.382265	46.570000	486.200012	128.839996	40.020000	...	127.290001	39.919998	58.770000	31.580000	12599800.0	1370210.0
2006-01-09	33.758297	364.161621	105.643616	27.211746	36.912933	20.379047	46.599998	483.899994	130.389999	40.669998	...	128.500000	39.880001	58.630001	31.674999	15619400.0	1680740.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2015-12-24	15.430058	46.007008	165.441620	56.782124	28.227333	45.933815	17.270000	52.709999	182.470001	66.599998	...	182.270004	66.620003	32.570000	54.970001	29369400.0	4671200.0
2015-12-28	15.304976	45.718975	164.670898	56.594566	27.957930	45.816509	17.129999	52.380001	181.619995	66.379997	...	181.770004	66.290001	32.360001	54.549999	41777500.0	8761700.0
2015-12-29	15.438995	46.242680	166.402649	57.182861	28.288172	46.327637	17.280001	52.980000	183.529999	67.070000	...	182.820007	66.839996	32.439999	55.110001	45670400.0	10155100.0
2015-12-30	15.233494	45.649151	165.024536	56.773621	28.010065	45.992474	17.049999	52.299999	182.009995	66.589996	...	183.279999	67.040001	32.500000	55.270000	35066400.0	8763300.0
2015-12-31	15.036935	45.169086	163.410629	56.296162	27.645065	45.548393	16.830000	51.750000	180.229996	66.029999	...	181.160004	66.190002	31.910000	54.509998	47153000.0	11281800.0

2517 rows × 36 columns

Create a list of the ticker symbols (as strings) in alphabetical order. Call this list: tickers (A ticker symbol or stock symbol is an abbreviation used to uniquely identify publicly traded shares of a particular stock on a particular stock market)

In [7]:
```
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 [8]:
```
bank_stocks = pd.concat([BAC, C, GS, JPM, MS, WFC],axis=1,keys=tickers)
bank_stocks
```

Out[8]:

	BAC									C	...		MS					
	High	Low	Open	Close	Volume	Adj Close	High	Low	Open	Close	...	Open	Close	Volume	Adj Close	High	Low	C
Date																		
2006-01-03	47.180000	46.150002	46.919998	47.080002	16296700.0	34.106041	493.799988	481.100006	490.000000	492.899994	...	57.169998	58.310001	5377000.0	36.364140	31.975000	31.195000	31.60
2006-01-04	47.240002	46.450001	47.000000	46.580002	17757900.0	33.743816	491.000000	483.500000	488.600006	483.799988	...	58.700001	58.349998	7977800.0	36.389069	31.820000	31.365000	31.79
2006-01-05	46.830002	46.320000	46.580002	46.639999	14970700.0	33.787266	487.799988	484.000000	484.399994	486.200012	...	58.549999	58.509998	5778000.0	36.488869	31.555000	31.309999	31.50
2006-01-06	46.910000	46.349998	46.799999	46.570000	12599800.0	33.736572	489.000000	482.000000	488.799988	486.200012	...	58.770000	58.570000	6889800.0	36.526283	31.775000	31.385000	31.58
2006-01-09	46.970001	46.360001	46.720001	46.599998	15619400.0	33.758297	487.399994	483.000000	486.000000	483.899994	...	58.630001	59.189999	4144500.0	36.912933	31.825001	31.555000	31.67
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2015-12-24	17.379999	17.219999	17.320000	17.270000	29369400.0	15.430058	52.970001	52.450001	52.480000	52.709999	...	32.570000	32.480000	2798200.0	28.227333	55.090000	54.709999	54.97
2015-12-28	17.230000	16.980000	17.219999	17.129999	41777500.0	15.304976	52.570000	51.959999	52.570000	52.380001	...	32.360001	32.169998	5420300.0	27.957930	54.779999	54.169998	54.54
2015-12-29	17.350000	17.160000	17.250000	17.280001	45670400.0	15.438995	53.220001	52.740002	52.759998	52.980000	...	32.439999	32.549999	6388200.0	28.288172	55.349998	54.990002	55.11
2015-12-30	17.240000	17.040001	17.200001	17.049999	35066400.0	15.233494	52.939999	52.250000	52.840000	52.299999	...	32.500000	32.230000	5057200.0	28.010065	55.310001	54.790001	55.27
2015-12-31	17.070000	16.830000	17.010000	16.830000	47153000.0	15.036935	52.389999	51.750000	52.070000	51.750000	...	31.910000	31.809999	8154300.0	27.645065	54.950001	54.220001	54.50

2517 rows × 36 columns

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

In [9]:
```
bank_stocks.columns.names = ['Bank Ticker','Stock Info']
```

Check the head of the bank_stocks dataframe.

In [10]:
```
bank_stocks.head(1)
```

Out[10]:

Bank Ticker	BAC									C	...		MS							
Stock Info	High	Low	Open	Close	Volume	Adj Close	High	Low	Open	Close	...	Open	Close	Volume	Adj Close	High	Low	Open	Close	Volu
Date																				
2006-01-03	47.18	46.150002	46.919998	47.080002	16296700.0	34.106041	493.799988	481.100006	490.0	492.899994	...	57.169998	58.310001	5377000.0	36.36414	31.975	31.195	31.6	31.9	110164(

1 rows × 36 columns

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 [11]:
```
bank_stocks.xs(key='Close',axis=1,level='Stock Info').max()
```

Out[11]:
```
Bank Ticker
BAC     54.900002
C      564.099976
GS     247.919998
JPM     70.080002
MS      89.300003
WFC     58.520000
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 [12]:
```
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 [13]:
```
for tick in tickers:
    returns[tick+' Return'] = bank_stocks[tick]['Close'].pct_change()
returns.head()
```

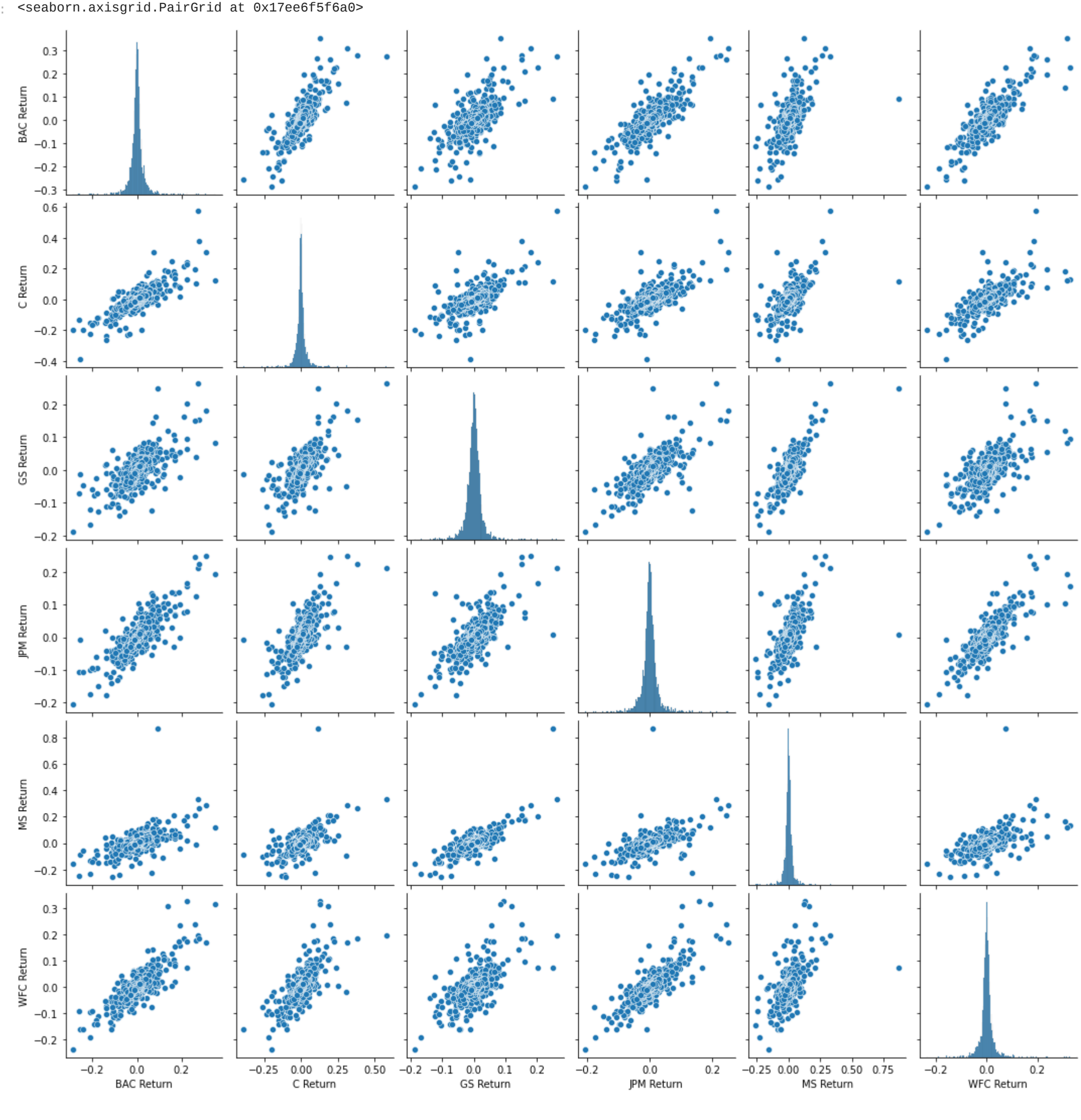
Out[13]:

	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	-0.001110
2006-01-06	-0.001501	0.000000	0.014169	0.007046	0.001025	0.005874
2006-01-09	0.000644	-0.004731	0.012030	0.016242	0.010586	-0.000158

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

In [14]:
```
#returns[1:]
import seaborn as sns
sns.pairplot(returns[1:])
```

Out[14]: <seaborn.axisgrid.PairGrid at 0x17ee6f5f6a0>



- See solution for details about Citigroup behavior....

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 [15]:
```
# Worst Drop (4 of them on Inauguration day)
returns.idxmin()
```

Out[15]:
```
BAC Return   2009-01-20
C Return     2009-02-27
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?

- See Solution for details

In [16]:
```
# Best Single Day Gain
# citigroup stock split in May 2011, but also JPM day after inauguration.
returns.idxmax()
```

Out[16]:
```
BAC Return   2009-04-09
C Return     2008-11-24
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 [17]:
```
returns.std() # Citigroup riskiest
```

Out[17]:
```
BAC Return    0.036647
C Return      0.038672
GS Return     0.025390
JPM Return    0.027667
MS Return     0.037819
WFC Return    0.030238
dtype: float64
```

In [18]:
```
returns.iloc[0:365].std() # Very similar risk profiles, but Morgan Stanley or BofA
```

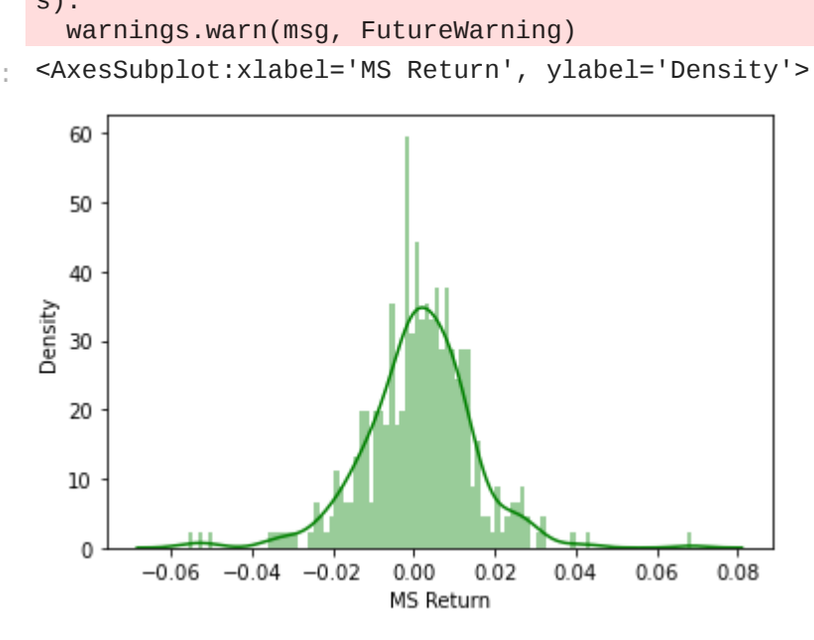
Out[18]:
```
BAC Return    0.008293
C Return      0.009784
GS Return     0.015271
JPM Return    0.010730
MS Return     0.013535
WFC Return    0.008684
dtype: float64
```

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

In [19]:
```
sns.distplot(returns.iloc[0:365]['MS Return'],color='green',bins=100)
```

```
C:\Users\Yasin\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

Out[19]: <AxesSubplot:xlabel='MS Return', ylabel='Density'>

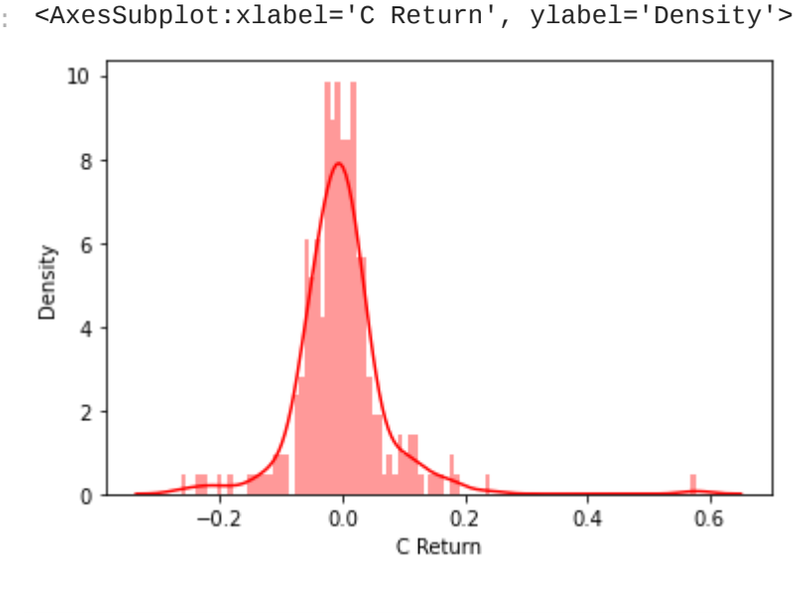


Create a distplot using seaborn of the 2008 returns for CitiGroup

In [20]:
```
sns.distplot(returns.loc['2008-01-01':'2008-12-31']['C Return'],color='red',bins=100)
```

```
C:\Users\Yasin\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

Out[20]: <AxesSubplot:xlabel='C Return', ylabel='Density'>



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.

Imports

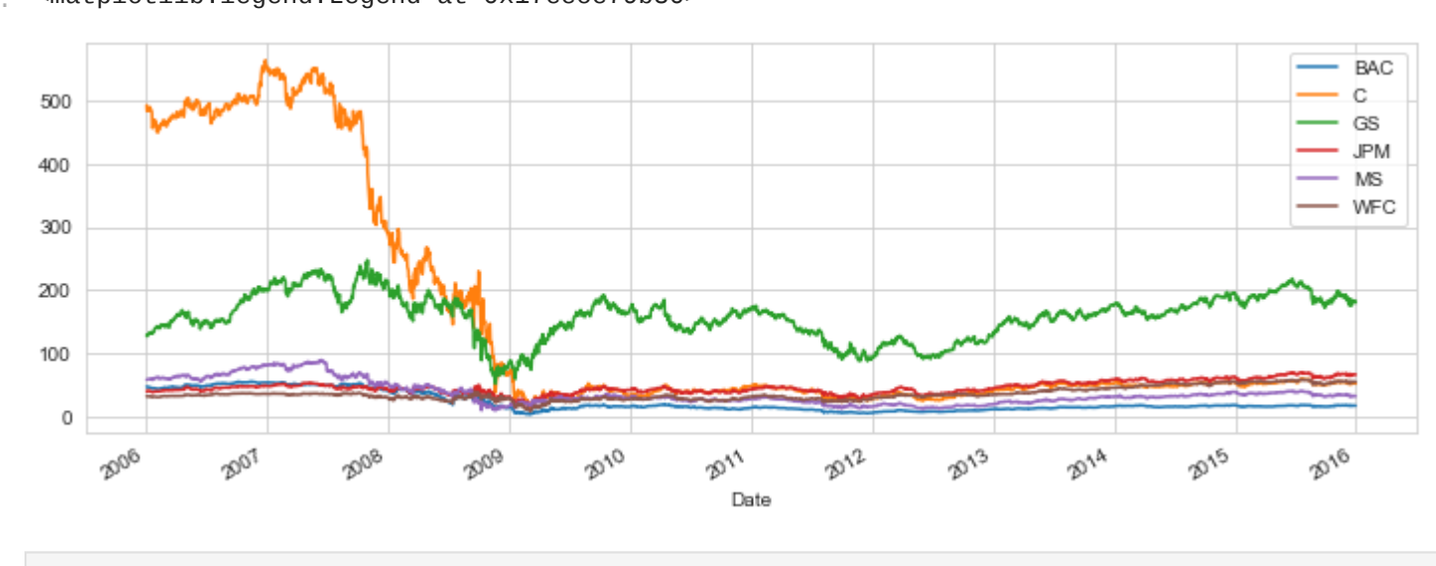
In [21]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

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

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 [22]:
```
for tick in tickers:
    bank_stocks[tick]['Close'].plot(figsize=(12,4),label=tick)
plt.legend()
```

Out[22]: <matplotlib.legend.Legend at 0x17eece79b80>



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

Out[23]: <AxesSubplot:xlabel='Date'>

In [24]:
```
# plotly
bank_stocks.xs(key='Close',axis=1,level='Stock Info').iplot()
```

Export to plot.ly »

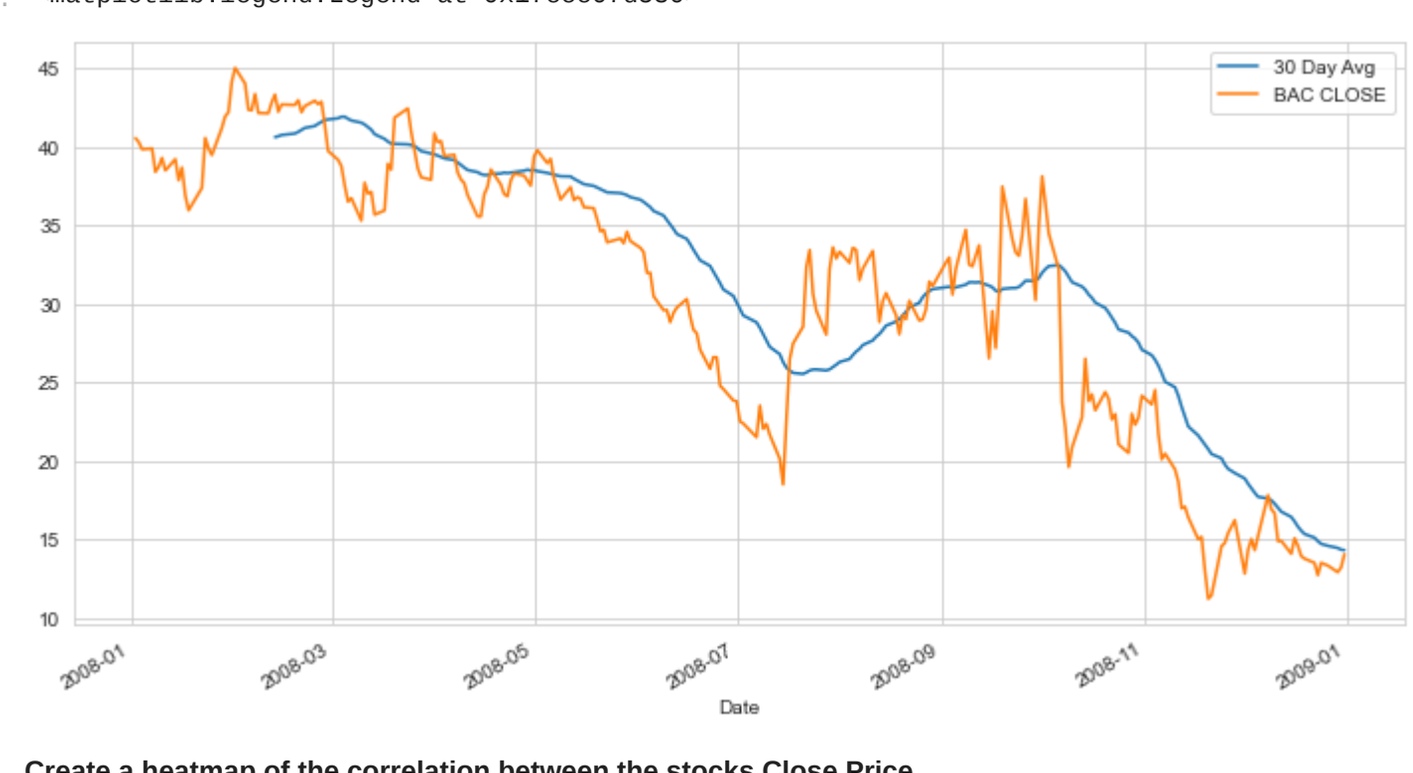
Moving Averages

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 [25]:
```
plt.figure(figsize=(12,6))
BAC['Close'].loc['2008-01-01':'2009-01-01'].rolling(window=30).mean().plot(label='30 Day Avg')
BAC['Close'].loc['2008-01-01':'2009-01-01'].plot(label='BAC CLOSE')
plt.legend()
```

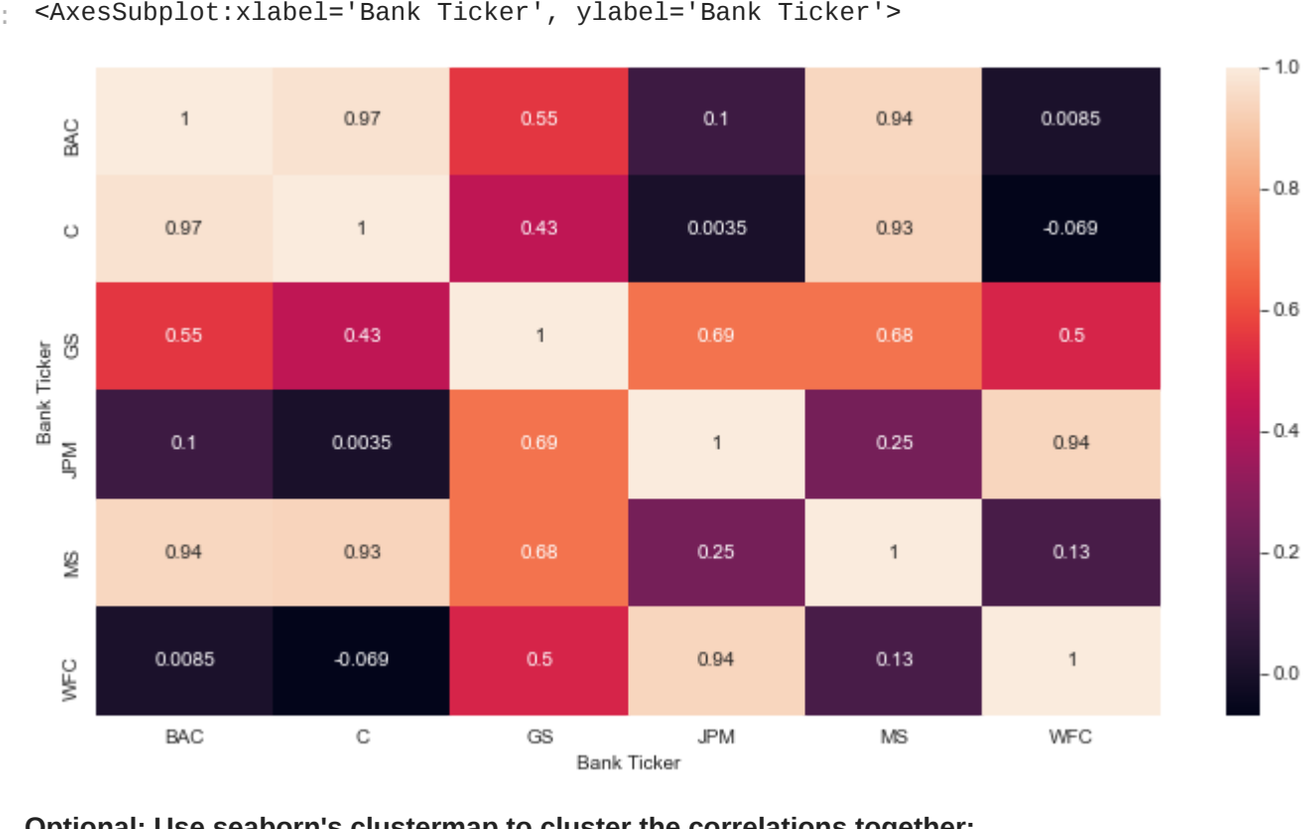
Out[25]: <matplotlib.legend.Legend at 0x17eee07d580>



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

In [26]:
```
plt.figure(figsize=(12,6))
sns.heatmap(bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr(),annot=True)
```

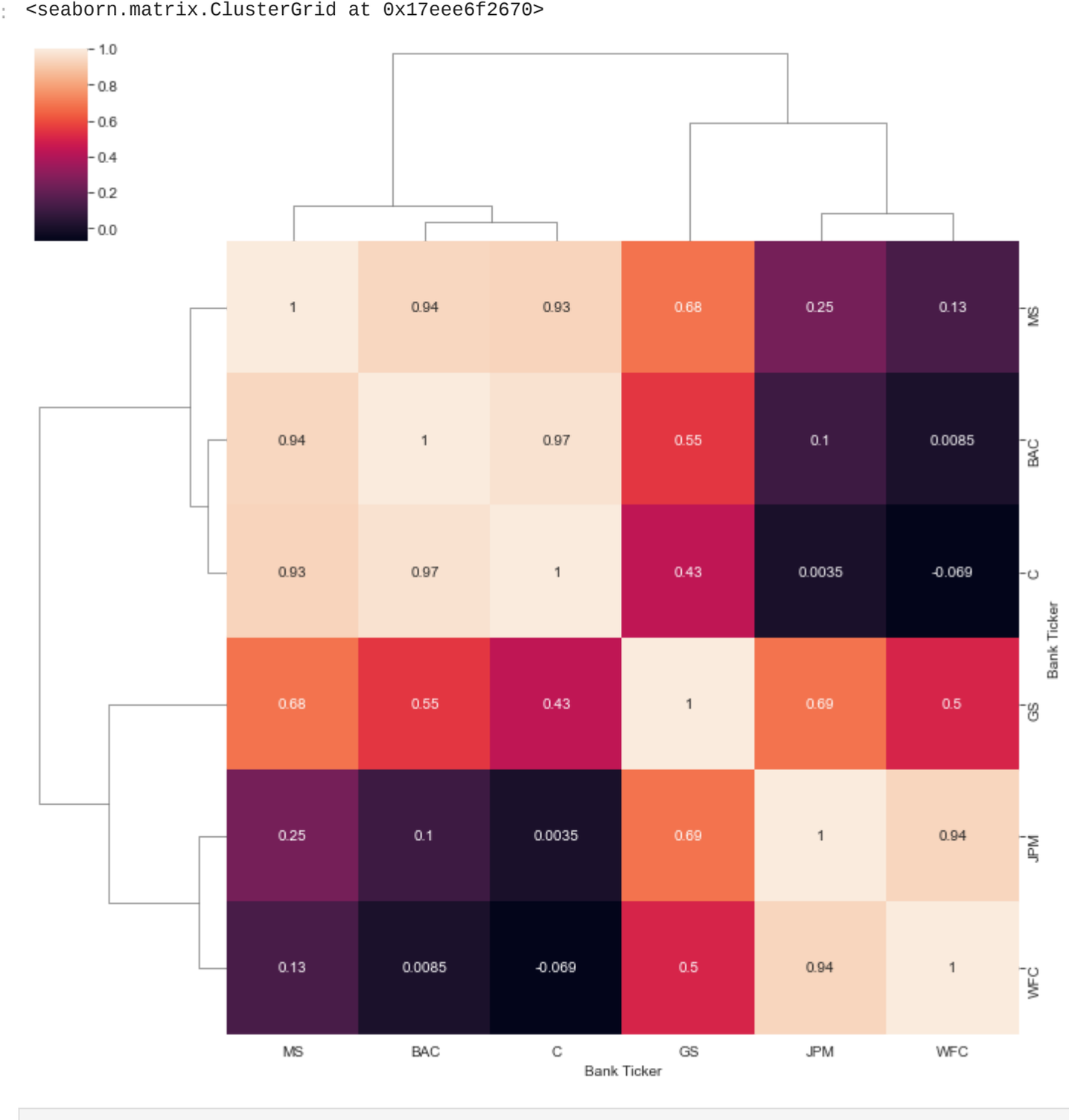
Out[26]: <AxesSubplot:xlabel='Bank Ticker', ylabel='Bank Ticker'>



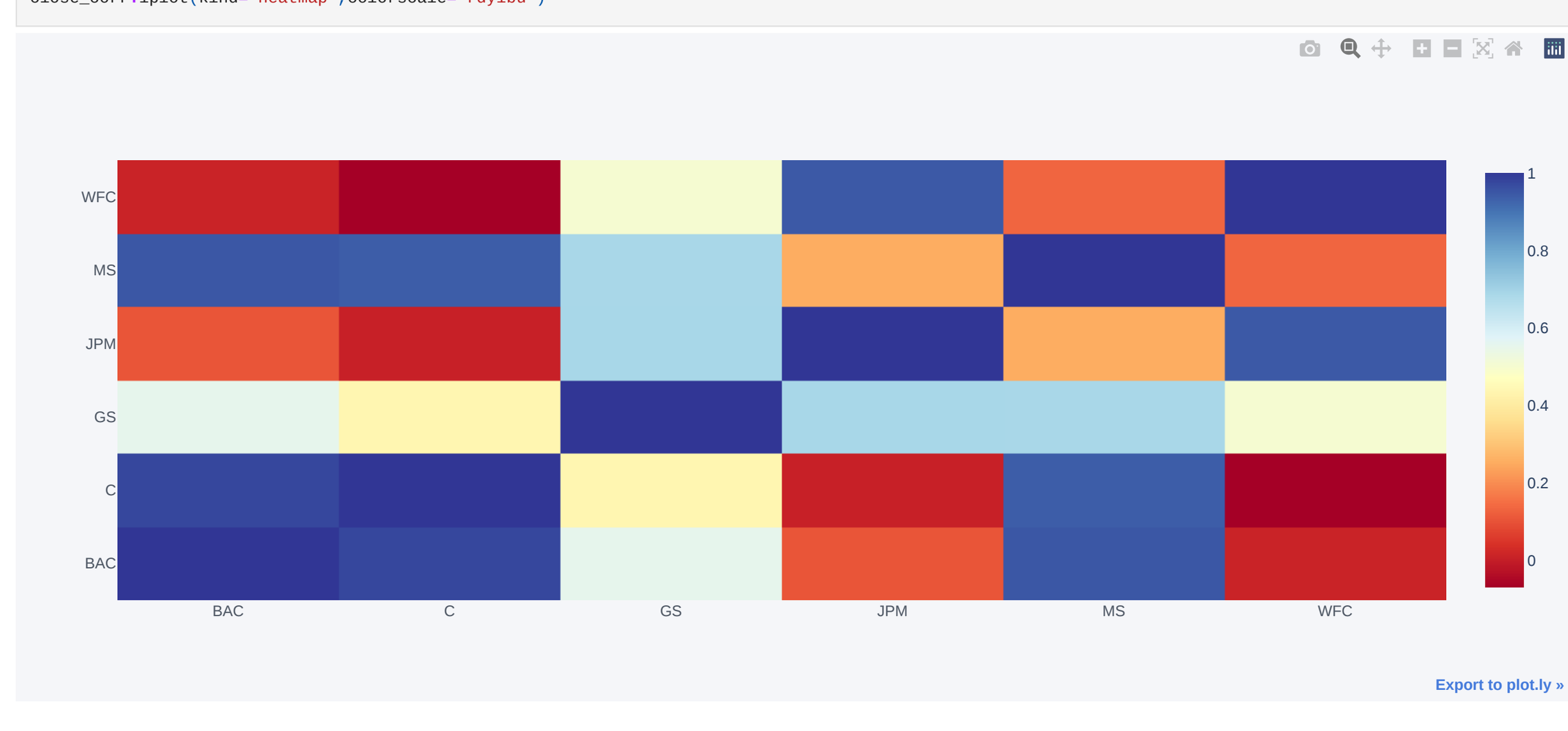
Optional: Use seaborn's clustermap to cluster the correlations together:

In [27]:
```
sns.clustermap(bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr(),annot=True)
```

Out[27]: <seaborn.matrix.ClusterGrid at 0x17eee6f2670>



In [28]:
```
close_corr = bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr()
close_corr.iplot(kind='heatmap',colorscale='rdylbu')
```



Export to plot.ly »

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 [29]:
```
BAC[['Open', 'High', 'Low', 'Close']].loc['2015-01-01':'2016-01-01'].iplot(kind='candle')
```

Export to plot.ly »

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

In [30]:
```
MS['Close'].loc['2015-01-01':'2016-01-01'].ta_plot(study='sma',periods=[13,21,55],title='Simple Moving Averages')
```

Export to plot.ly »

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

In [31]:
```
BAC['Close'].loc['2015-01-01':'2016-01-01'].ta_plot(study='boll')
```

Export to plot.ly »

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.