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 [38]:
```
from pandas_datareader import data, wb
import pandas as pd
import numpy as np
import datetime
import seaborn as sns
%matplotlib inline
```

In [39]:
```
from pandas_datareader import data, wb
import yfinance as yf
yf.pdr_override()
```

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 [9]:
```
#We pass in the year, the month, the day
#This code is taken online from Google Finance website

start = datetime.datetime(2006,1,1)
end = datetime.datetime(2016,1,1)
```

In [20]:
```
# Bank of America
BAC = data.get_data_yahoo('BAC', start, end)

# CitiGroup
C = data.get_data_yahoo('C', start, end)

# Goldman Sachs
GS = data.get_data_yahoo('GS', start, end)

# JPMorgan Chase
JPM = data.get_data_yahoo('JPM', start, end)

# Morgan Stanley
MS = data.get_data_yahoo('MS', start, end)

# Wells Fargo
WFC = data.get_data_yahoo('WFC', start, end)
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

In [21]:
```
#Shows all the info down

BAC
```

Out[21]:

Date	Open	High	Low	Close	Adj Close	Volume
2006-01-03 00:00:00-05:00	46.919998	47.180000	46.150002	47.080002	33.170319	16296700
2006-01-04 00:00:00-05:00	47.000000	47.240002	46.450001	46.580002	32.818035	17757900
2006-01-05 00:00:00-05:00	46.580002	46.830002	46.320000	46.639999	32.860313	14970700
2006-01-06 00:00:00-05:00	46.799999	46.910000	46.349998	46.570000	32.810989	12599800
2006-01-09 00:00:00-05:00	46.720001	46.970001	46.360001	46.599998	32.832115	15619400
...	...	...	...	...	...	...
2015-12-24 00:00:00-05:00	17.320000	17.379999	17.219999	17.270000	15.006726	29369400
2015-12-28 00:00:00-05:00	17.219999	17.230000	16.980000	17.129999	14.885073	41777500
2015-12-29 00:00:00-05:00	17.250000	17.350000	17.160000	17.280001	15.015417	45670400
2015-12-30 00:00:00-05:00	17.200001	17.240000	17.040001	17.049999	14.815557	35066400
2015-12-31 00:00:00-05:00	17.010000	17.070000	16.830000	16.830000	14.624387	47153000

2517 rows × 6 columns

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

In [22]:
```
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 [23]:
```
bank_stocks = pd.concat([BAC,C,GS,JPM,MS,WFC],axis=1,keys=tickers)
```

In [24]:
```
#Now let's display our bank stocks

bank_stocks.head()
```

Out[24]:

	BAC						C				...	
	Open	High	Low	Close	Adj Close	Volume	Open	High	Low	Close	...	Low
Date												
2006-01-03 00:00:00-05:00	46.919998	47.180000	46.150002	47.080002	33.170319	16296700	490.000000	493.799988	481.100006	492.899994	...	56.740002
2006-01-04 00:00:00-05:00	47.000000	47.240002	46.450001	46.580002	32.818035	17757900	488.600006	491.000000	483.500000	483.799988	...	58.349998
2006-01-05 00:00:00-05:00	46.580002	46.830002	46.320000	46.639999	32.860313	14970700	484.399994	487.799988	484.000000	486.200012	...	58.020000
2006-01-06 00:00:00-05:00	46.799999	46.910000	46.349998	46.570000	32.810989	12599800	488.799988	489.000000	482.000000	486.200012	...	58.049999
2006-01-09 00:00:00-05:00	46.720001	46.970001	46.360001	46.599998	32.832115	15619400	486.000000	487.399994	483.000000	483.899994	...	58.619999

5 rows × 36 columns

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

In [25]:
```
bank_stocks.columns.names = ['Bank Ticker','Stock Info']
```

Check the head of the bank_stocks dataframe.

In [26]:
```
bank_stocks.head()
```

Out[26]:

Bank Ticker	BAC						C				...	
Stock Info	Open	High	Low	Close	Adj Close	Volume	Open	High	Low	Close	...	Low
Date												
2006-01-03 00:00:00-05:00	46.919998	47.180000	46.150002	47.080002	33.170319	16296700	490.000000	493.799988	481.100006	492.899994	...	56.740002
2006-01-04 00:00:00-05:00	47.000000	47.240002	46.450001	46.580002	32.818035	17757900	488.600006	491.000000	483.500000	483.799988	...	58.349998
2006-01-05 00:00:00-05:00	46.580002	46.830002	46.320000	46.639999	32.860313	14970700	484.399994	487.799988	484.000000	486.200012	...	58.020000
2006-01-06 00:00:00-05:00	46.799999	46.910000	46.349998	46.570000	32.810989	12599800	488.799988	489.000000	482.000000	486.200012	...	58.049999
2006-01-09 00:00:00-05:00	46.720001	46.970001	46.360001	46.599998	32.832115	15619400	486.000000	487.399994	483.000000	483.899994	...	58.619999

5 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 [27]:
```
#for tick in tickers:
    #bank_stocks[tick]
#Let's say we just want one which is from Bank of America

bank_stocks['BAC']['Close'].max()
```

Out[27]:
```
54.900001525878906
```

In [31]:
```
#Now let's get to all and each bank's stock max close price
for tick in tickers:
    print(tick,bank_stocks[tick]['Close'].max())
```

```
BAC 54.900001525878906
C 564.0999755859375
GS 247.9199981689453
JPM 70.08000183105469
MS 89.30000305175781
WFC 58.52000045776367
```

In [32]:
```
#Another method would be
bank_stocks.xs(key='Close',axis=1,level='Stock Info').max()
```

Out[32]:
```
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 [33]:
```
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 [40]:
```
for tick in tickers:
    returns[tick + 'Return'] = bank_stocks[tick]['Close'].pct_change()
```

In [41]:
```
returns.head()
```

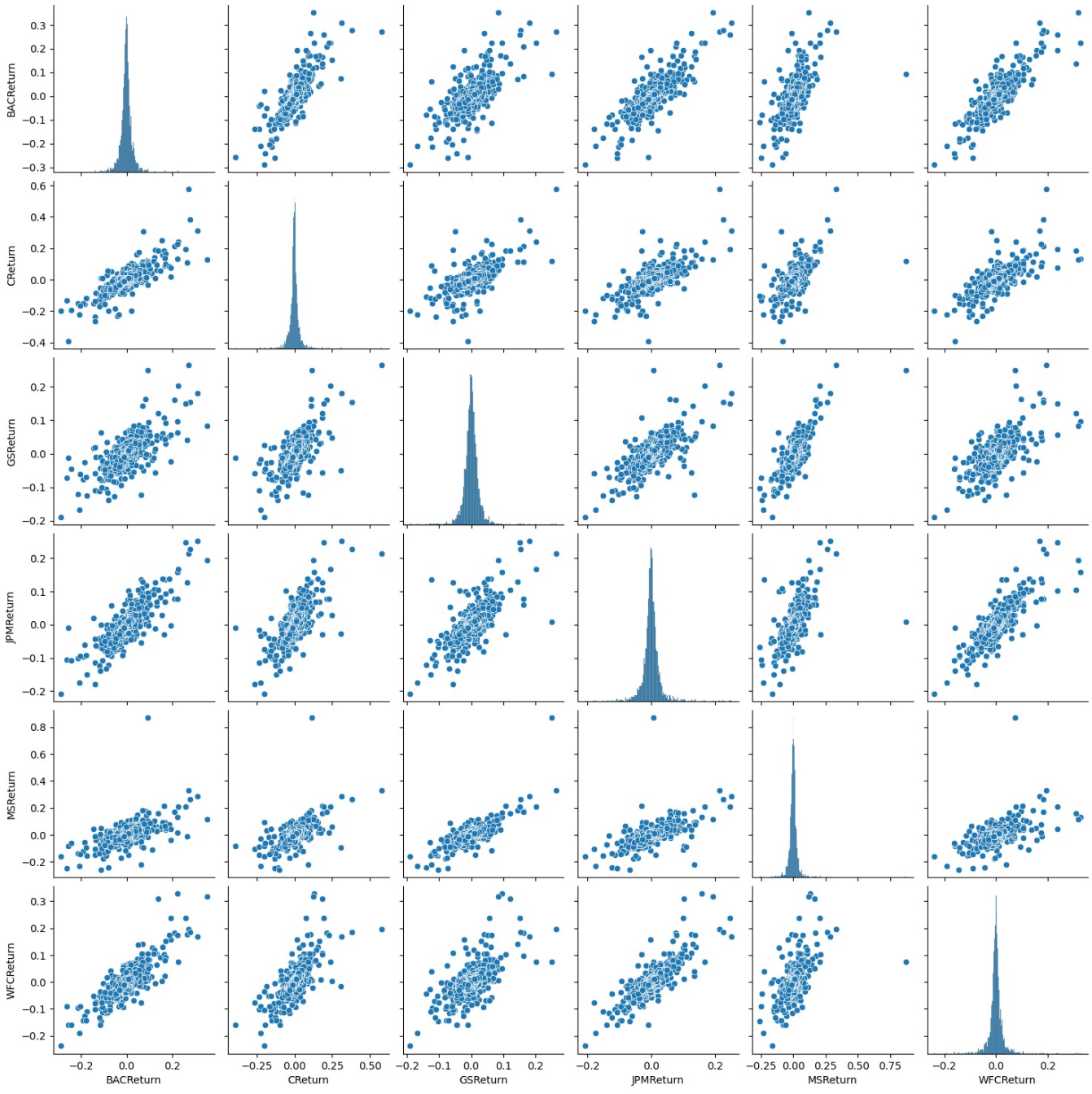
Out[41]:

Date	BACReturn	CReturn	GSReturn	JPMReturn	MSReturn	WFCReturn
2006-01-03 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN
2006-01-04 00:00:00-05:00	-0.010620	-0.018462	-0.013812	-0.014183	0.000686	-0.011599
2006-01-05 00:00:00-05:00	0.001288	0.004961	-0.000393	0.003029	0.002742	-0.001110
2006-01-06 00:00:00-05:00	-0.001501	0.000000	0.014169	0.007046	0.001025	0.005874
2006-01-09 00:00:00-05:00	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 [42]:
```
sns.pairplot(returns[1:])
```

Out[42]:
```
<seaborn.axisgrid.PairGrid at 0x13e893acdf0>
```



- 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 [45]:
```
#To get the min value of these returns,

returns.idxmin()
```

Out[45]:
```
BACReturn   2009-01-20 00:00:00-05:00
CReturn     2009-02-27 00:00:00-05:00
GSReturn    2009-01-20 00:00:00-05:00
JPMReturn   2009-01-20 00:00:00-05:00
MSReturn    2008-10-09 00:00:00-04:00
WFCReturn   2009-01-20 00:00:00-05:00
dtype: datetime64[ns, America/New_York]
```

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 [76]:
```
returns.idxmax()
```

Out[76]:
```
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 [47]:
```
#A good measure and note for your feature endeavors is
# The standard deviation of the returns of a stock is a good measure of it's volatility
# The higher the STD of returns of the stock, the riskier it is. This means it's volatile asl.
# Here's is how we calculate the following;

returns .std()

#So we can see that the riskiest is MS and the most stable is GS
```

Out[47]:
```
BACReturn    0.036647
CReturn      0.038672
GSReturn     0.025390
JPMReturn    0.027667
MSReturn     0.037819
WFCReturn    0.030238
dtype: float64
```

In [50]:
```
returns.loc['2015-01-01':'2015-12-31'].std()

#Riskiest is BAC, least risky is WFC
```

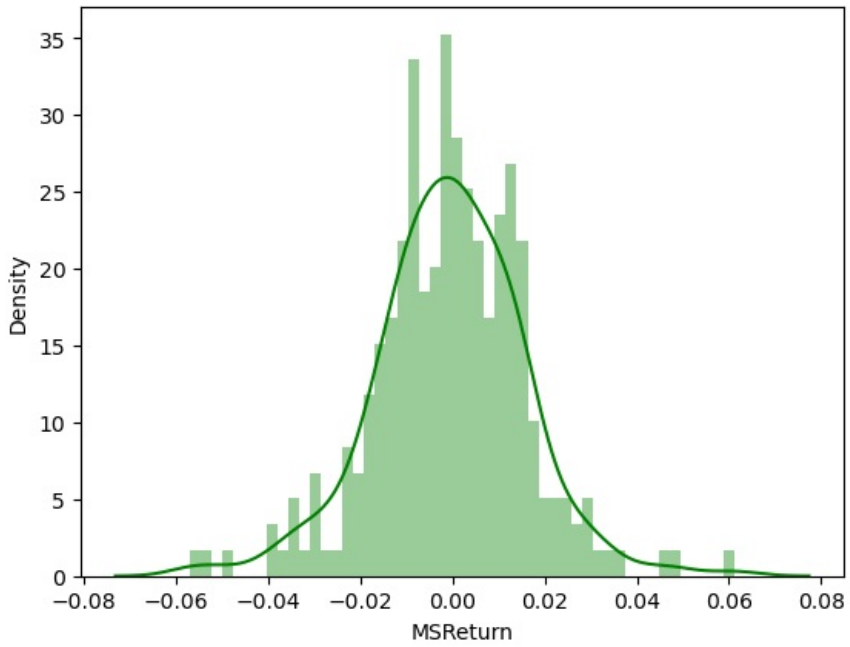
Out[50]:
```
BACReturn    0.016163
CReturn      0.015289
GSReturn     0.014046
JPMReturn    0.014017
MSReturn     0.016249
WFCReturn    0.012591
dtype: float64
```

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

In [52]:
```
sns.distplot(returns.loc['2015-01-01':'2015-12-31']['MSReturn'],color='green',bins=50)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprec
ated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[52]:
```
<AxesSubplot:xlabel='MSReturn', ylabel='Density'>
```

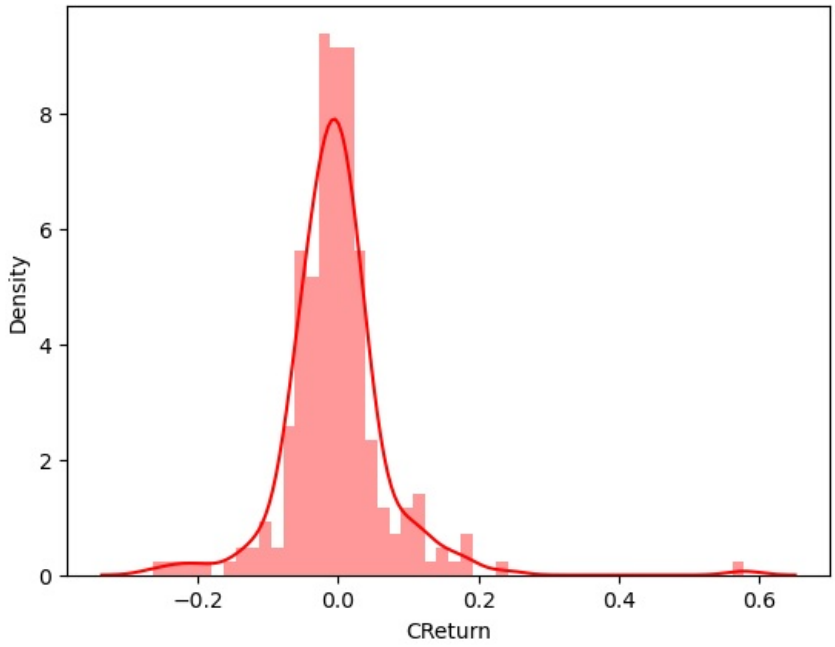


Create a distplot using seaborn of the 2008 returns for CitiGroup

In [54]:
```
sns.distplot(returns.loc['2008-01-01':'2008-12-31']['CReturn'],color='red',bins=50)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprec
ated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[54]:
```
<AxesSubplot:xlabel='CReturn', 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

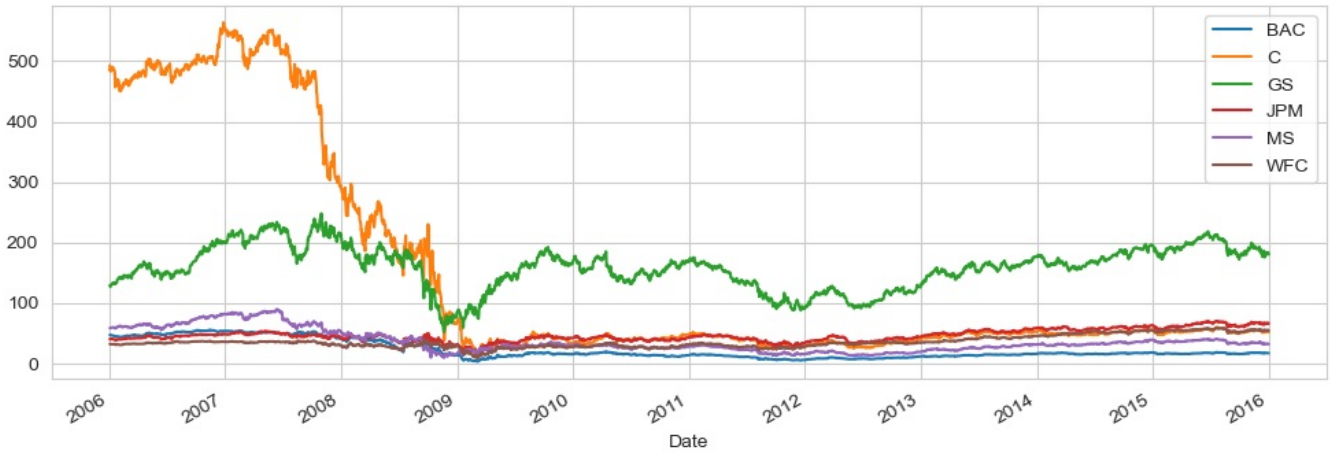
In [55]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

# 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.)

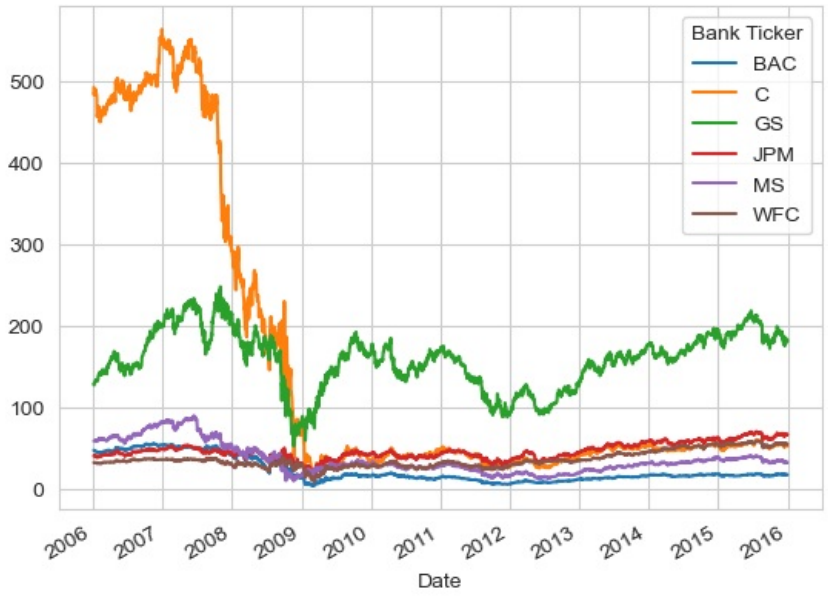
In [56]:
```
for tick in tickers:
    bank_stocks[tick]['Close'].plot(label=tick,figsize=(12,4))
plt.legend()
```

Out[56]:
```
<matplotlib.legend.Legend at 0x13e8f837b50>
```



In [58]:
```
#Another method
bank_stocks.xs(key='Close',axis=1,level='Stock Info').plot()
```

Out[58]:
```
<AxesSubplot:xlabel='Date'>
```



In [59]:
```
#3rd method - iplot method
bank_stocks.xs(key='Close',axis=1,level='Stock Info').iplot()
```

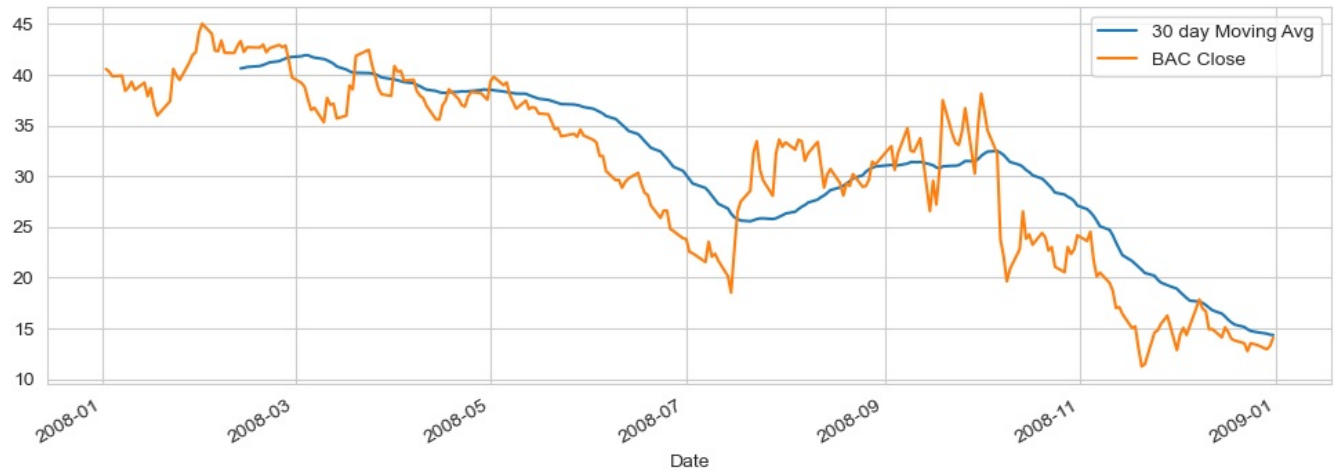
Moving Averages

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 [62]:
```
plt.figure(figsize=(12,4))
BAC['Close'].loc['2008-01-01':'2009-01-01'].rolling(window=30).mean().plot(label='30 day Moving Avg')
BAC['Close'].loc['2008-01-01':'2009-01-01'].plot(label='BAC Close')
plt.legend()
```

Out[62]:
```
<matplotlib.legend.Legend at 0x13e9291a3a0>
```



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

In [63]:
```
#sns.heatmap() - First let's find

bank_stocks.xs(key='Close',axis=1,level='Stock Info')
```

Out[63]:

Bank Ticker	BAC	C	GS	JPM	MS	WFC
Date						
2006-01-03 00:00:00-05:00	47.080002	492.899994	128.869995	40.189999	58.310001	31.900000
2006-01-04 00:00:00-05:00	46.580002	483.799988	127.089996	39.619999	58.349998	31.530001
2006-01-05 00:00:00-05:00	46.639999	486.200012	127.040001	39.740002	58.509998	31.495001
2006-01-06 00:00:00-05:00	46.570000	486.200012	128.839996	40.020000	58.570000	31.680000
2006-01-09 00:00:00-05:00	46.599998	483.899994	130.389999	40.669998	59.189999	31.674999
...	...	...	...	...	...	...
2015-12-24 00:00:00-05:00	17.270000	52.709999	182.470001	66.599998	32.480000	54.820000
2015-12-28 00:00:00-05:00	17.129999	52.380001	181.619995	66.379997	32.169998	54.680000
2015-12-29 00:00:00-05:00	17.280001	52.980000	183.529999	67.070000	32.549999	55.290001
2015-12-30 00:00:00-05:00	17.049999	52.299999	182.009995	66.589996	32.230000	54.889999
2015-12-31 00:00:00-05:00	16.830000	51.750000	180.229996	66.029999	31.809999	54.360001

2517 rows × 6 columns

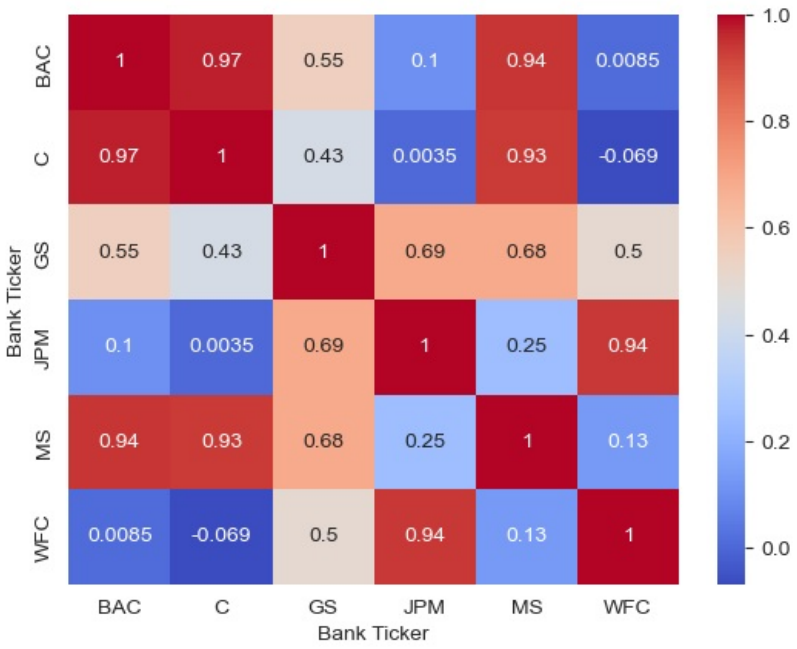
In [64]:
```
bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr()
```

Out[64]:

Bank Ticker	BAC	C	GS	JPM	MS	WFC
Bank Ticker						
BAC	1.000000	0.971516	0.550898	0.103874	0.944218	0.008542
C	0.971516	1.000000	0.434123	0.003515	0.933609	-0.068536
GS	0.550898	0.434123	1.000000	0.685286	0.683792	0.499897
JPM	0.103874	0.003515	0.685286	1.000000	0.250427	0.940269
MS	0.944218	0.933609	0.683792	0.250427	1.000000	0.131835
WFC	0.008542	-0.068536	0.499897	0.940269	0.131835	1.000000

In [67]:
```
#Now Heatmap;
sns.heatmap(bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr(), annot = True, cmap='coolwarm')
```

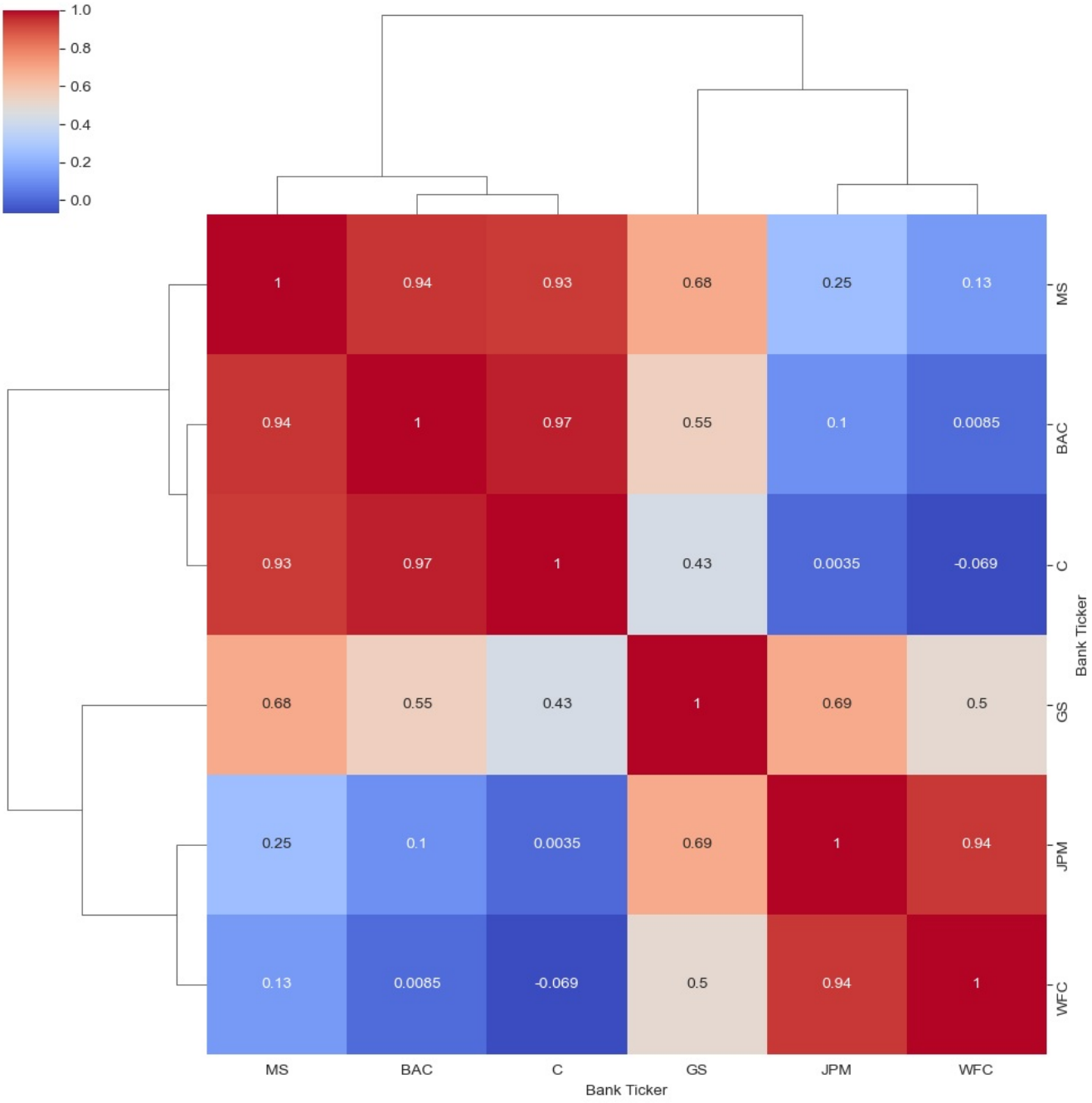
Out[67]:
```
<AxesSubplot:xlabel='Bank Ticker', ylabel='Bank Ticker'>
```



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

In [68]:
```
sns.clustermap(bank_stocks.xs(key='Close',axis=1,level='Stock Info').corr(), annot = True, cmap='coolwarm')
```

Out[68]:
```
<seaborn.matrix.ClusterGrid at 0x13e8f88fb80>
```



Part 3 (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.

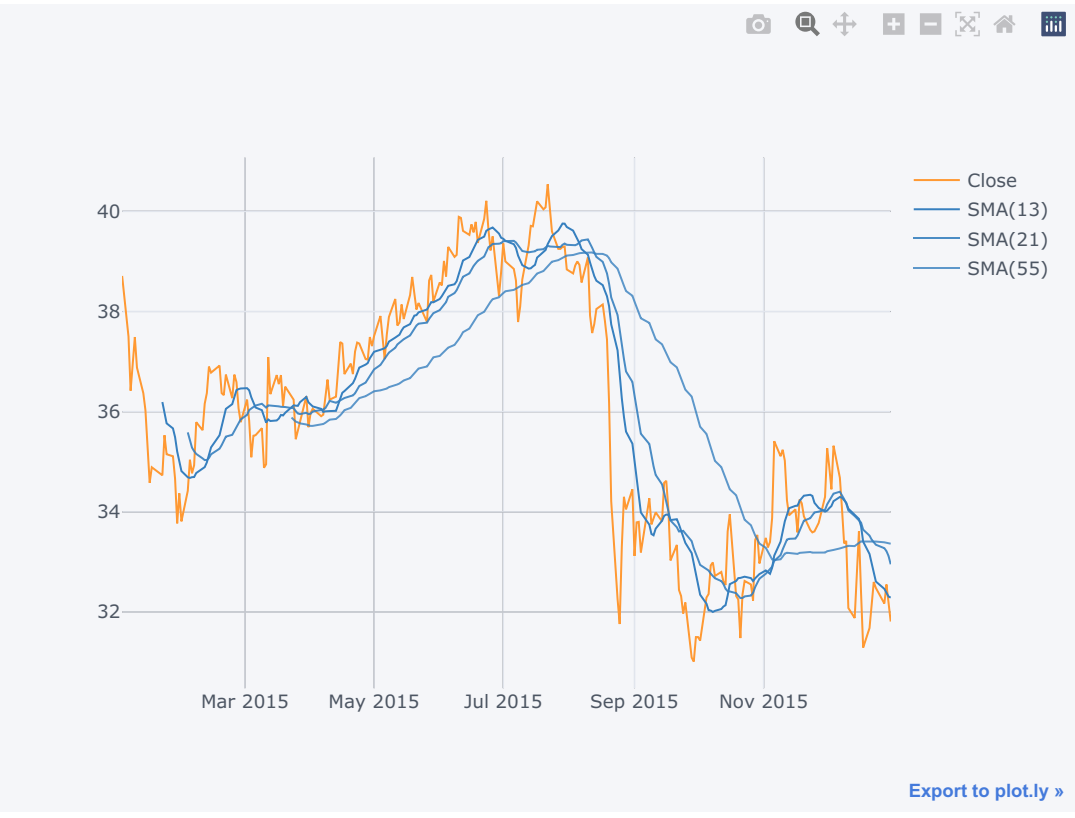
In [76]:
```
#USE THIS TO CREATE YOUR OWN CANDLESTICK PATTERNS

bac15 = BAC[['Open','High','Low','Close']].loc['2015-01-01':'2016-01-01']
bac15.iplot(kind='candle')
```

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

In [77]:
```
#SMA

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



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

In [78]:
```
#Bollinger Bands

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

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.