

Anirudh Dhawan - Finance Project

July 28, 2025

1 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. ** _____ 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.

```
[10]: import pandas_datareader as pdr
from pandas_datareader import data, wb
import pandas as pd
import numpy as np
import datetime
from datetime 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)
```

1.3 ##### WARNING: MAKE SURE TO CHECK THE LINK ABOVE FOR THE LATEST WORKING API. “google” MAY NOT ALWAYS WORK.

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

```
[11]: start = datetime(2006, 1, 1)  
end = datetime(2016, 1, 1)
```

```
[15]: df = pd.read_pickle('all_banks.pkl')
```

```
[21]: df
```

```
[21]: Bank Ticker      BAC  
Stock Info      Open   High    Low  Close     Volume      C          \  
Date  
2006-01-03    46.92  47.18  46.15  47.08  16296700  490.00  493.80  481.10  
2006-01-04    47.00  47.24  46.45  46.58  17757900  488.60  491.00  483.50  
2006-01-05    46.58  46.83  46.32  46.64  14970900  484.40  487.80  484.00  
2006-01-06    46.80  46.91  46.35  46.57  12599800  488.80  489.00  482.00  
2006-01-09    46.72  46.97  46.36  46.60  15620000  486.00  487.40  483.00  
...       ...  ...  ...  ...  ...  ...  ...  ...  ...  
2015-12-24    17.32  17.38  17.22  17.27  29373415  52.48  52.97  52.45  
2015-12-28    17.22  17.23  16.98  17.13  41777497  52.57  52.57  51.96  
2015-12-29    17.25  17.35  17.16  17.28  45670376  52.76  53.22  52.74  
2015-12-30    17.20  17.24  17.04  17.05  35066378  52.84  52.94  52.25  
2015-12-31    17.01  17.07  16.83  16.83  47152968  52.07  52.39  51.75
```

```
Bank Ticker      MS  
Stock Info      Close  Volume  ...  Open   High    Low  Close     Volume          \  
Date  
...  
2006-01-03    492.90  1537660  ...  57.17  58.49  56.74  58.31  5377000  
2006-01-04    483.80  1871020  ...  58.70  59.28  58.35  58.35  7977800  
2006-01-05    486.20  1143160  ...  58.55  58.59  58.02  58.51  5778000  
2006-01-06    486.20  1370250  ...  58.77  58.85  58.05  58.57  6889800  
2006-01-09    483.90  1680740  ...  58.63  59.29  58.62  59.19  4144500
```

...												
2015-12-24	52.71	4671254	...	32.57	32.71	32.44	32.48	2798163				
2015-12-28	52.38	8761743	...	32.36	32.36	31.95	32.17	5420280				
2015-12-29	52.98	10155134	...	32.44	32.70	32.32	32.55	6388244				
2015-12-30	52.30	8763337	...	32.50	32.64	32.20	32.23	5057162				
2015-12-31	51.75	11281771	...	31.91	32.30	31.77	31.81	8154307				

Bank Ticker	WFC				
Stock Info	Open	High	Low	Close	Volume
Date					
2006-01-03	31.60	31.98	31.20	31.90	11016400
2006-01-04	31.80	31.82	31.36	31.53	10871000
2006-01-05	31.50	31.56	31.31	31.50	10158000
2006-01-06	31.58	31.78	31.38	31.68	8403800
2006-01-09	31.68	31.82	31.56	31.68	5619600
...
2015-12-24	54.97	55.09	54.71	54.82	4999417
2015-12-28	54.55	54.78	54.17	54.68	8288841
2015-12-29	55.11	55.35	54.99	55.29	7894876
2015-12-30	55.27	55.31	54.79	54.89	8016893
2015-12-31	54.51	54.95	54.22	54.36	10929767

[2517 rows x 30 columns]

```
[22]: df.columns
#Bank of America --> BAC
#Citigroup --> C
#Goldman Sachs --> GS
#JPMorgan Chase --> JPM
#Morgan Stanley --> MS
#Wells Fargo --> WFC
```

```
[22]: MultiIndex([('BAC',      'Open'),
                  ('BAC',      'High'),
                  ('BAC',      'Low'),
                  ('BAC',      'Close'),
                  ('BAC',      'Volume'),
                  ('C',        'Open'),
                  ('C',        'High'),
                  ('C',        'Low'),
                  ('C',        'Close'),
                  ('C',        'Volume'),
                  ('GS',       'Open'),
                  ('GS',       'High'),
                  ('GS',       'Low'),
                  ('GS',       'Close'),
                  ('GS',       'Volume'),
```

```

        ('JPM',      'Open'),
        ('JPM',      'High'),
        ('JPM',      'Low'),
        ('JPM',      'Close'),
        ('JPM',      'Volume'),
        ('MS',       'Open'),
        ('MS',       'High'),
        ('MS',       'Low'),
        ('MS',       'Close'),
        ('MS',       'Volume'),
        ('WFC',      'Open'),
        ('WFC',      'High'),
        ('WFC',      'Low'),
        ('WFC',      'Close'),
        ('WFC',      'Volume')],
    names=['Bank Ticker', 'Stock Info'])

```

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

```
[51]: tickers: list[str] = ['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.**

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

```
[52]: bank_stocks = df
```

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

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

```
[54]: bank_stocks.head()
```

	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	High			
Date													
2006-01-03	1537660	...	57.17	58.49	56.74	58.31	5377000	31.60	31.98				
2006-01-04	1871020	...	58.70	59.28	58.35	58.35	7977800	31.80	31.82				

```
2006-01-05    1143160 ... 58.55 58.59 58.02 58.51 5778000 31.50 31.56
2006-01-06    1370250 ... 58.77 58.85 58.05 58.57 6889800 31.58 31.78
2006-01-09    1680740 ... 58.63 59.29 58.62 59.19 4144500 31.68 31.82
```

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

[5 rows x 30 columns]

```
[55]: bank_stocks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2517 entries, 2006-01-03 to 2015-12-31
Data columns (total 30 columns):
 #  Column          Non-Null Count  Dtype  
--- 
 0  (BAC, Open)    2517 non-null   float64
 1  (BAC, High)   2517 non-null   float64
 2  (BAC, Low)    2517 non-null   float64
 3  (BAC, Close)  2517 non-null   float64
 4  (BAC, Volume) 2517 non-null   int64  
 5  (C, Open)     2517 non-null   float64
 6  (C, High)    2517 non-null   float64
 7  (C, Low)     2517 non-null   float64
 8  (C, Close)   2517 non-null   float64
 9  (C, Volume)  2517 non-null   int64  
 10 (GS, Open)   2517 non-null   float64
 11 (GS, High)  2517 non-null   float64
 12 (GS, Low)   2517 non-null   float64
 13 (GS, Close) 2517 non-null   float64
 14 (GS, Volume) 2517 non-null   int64  
 15 (JPM, Open)  2517 non-null   float64
 16 (JPM, High) 2517 non-null   float64
 17 (JPM, Low)  2517 non-null   float64
 18 (JPM, Close) 2517 non-null   float64
 19 (JPM, Volume) 2517 non-null   int64  
 20 (MS, Open)   2517 non-null   float64
 21 (MS, High)  2517 non-null   float64
 22 (MS, Low)   2517 non-null   float64
 23 (MS, Close) 2517 non-null   float64
 24 (MS, Volume) 2517 non-null   int64  
 25 (WFC, Open) 2517 non-null   float64
```

```

26 (WFC, High)    2517 non-null   float64
27 (WFC, Low)     2517 non-null   float64
28 (WFC, Close)   2517 non-null   float64
29 (WFC, Volume)  2517 non-null   int64
dtypes: float64(24), int64(6)
memory usage: 609.6 KB

```

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](#).

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

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

```

[56]: 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:**

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

```
[57]: 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.**

```

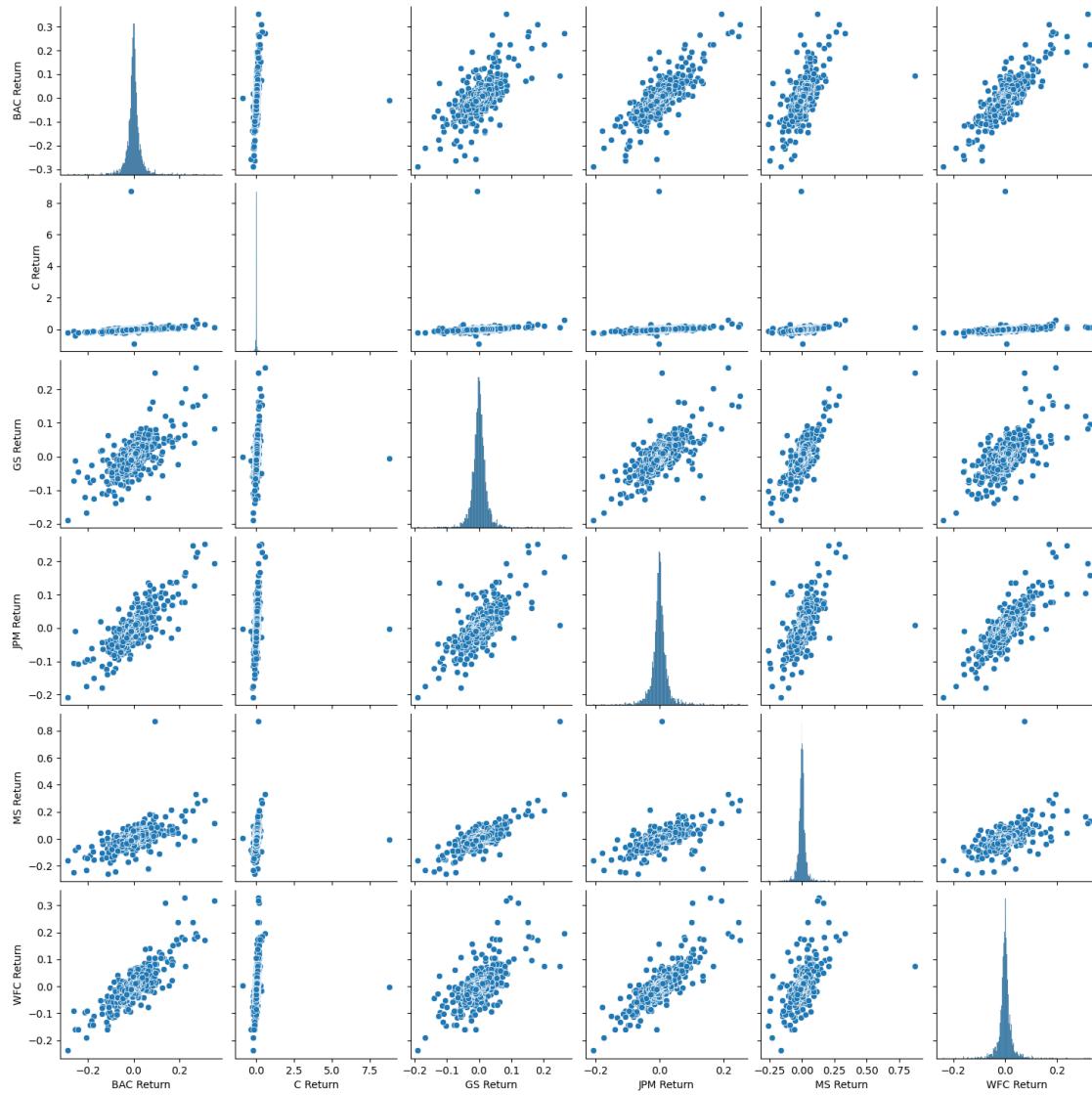
[58]: for ticker in tickers:
        returns[ticker + ' Return'] = bank_stocks[ticker]['Close'].pct_change()
returns.head()

```

	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?**

```
[62]: import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.pairplot(returns[1:])
plt.show()
```



** 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?**

```
[67]: returns.idxmin()
#BAC, GS, JPM, and WFC all had their worst drop on inauguration day
```

```
[67]: 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 anything significant happen in that time frame? **

```
[86]: #Citigroup had a stock split
```

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

```
[68]: 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]
```

```
[69]: returns
```

```
[69]:          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
...
2015-12-24 -0.004037  0.001520 -0.002624 -0.001948 -0.003681 -0.003997
2015-12-28 -0.008107 -0.006261 -0.004658 -0.003303 -0.009544 -0.002554
2015-12-29  0.008757  0.011455  0.010516  0.010395  0.011812  0.011156
2015-12-30 -0.013310 -0.012835 -0.008282 -0.007157 -0.009831 -0.007235
2015-12-31 -0.012903 -0.010516 -0.009780 -0.008410 -0.013031 -0.009656
```

[2517 rows x 6 columns]

```
[70]: returns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2517 entries, 2006-01-03 to 2015-12-31
Data columns (total 6 columns):
```

```

#   Column      Non-Null Count  Dtype  
---  --  
0   BAC Return    2516 non-null   float64 
1   C Return      2516 non-null   float64 
2   GS Return     2516 non-null   float64 
3   JPM Return    2516 non-null   float64 
4   MS Return     2516 non-null   float64 
5   WFC Return    2516 non-null   float64 

dtypes: float64(6)
memory usage: 137.6 KB

```

** 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?**

```
[75]: returns.std()
#Citigroup poses higher risk because it has the highest variation about the mean indicating volatility.
```

```
[75]: 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
```

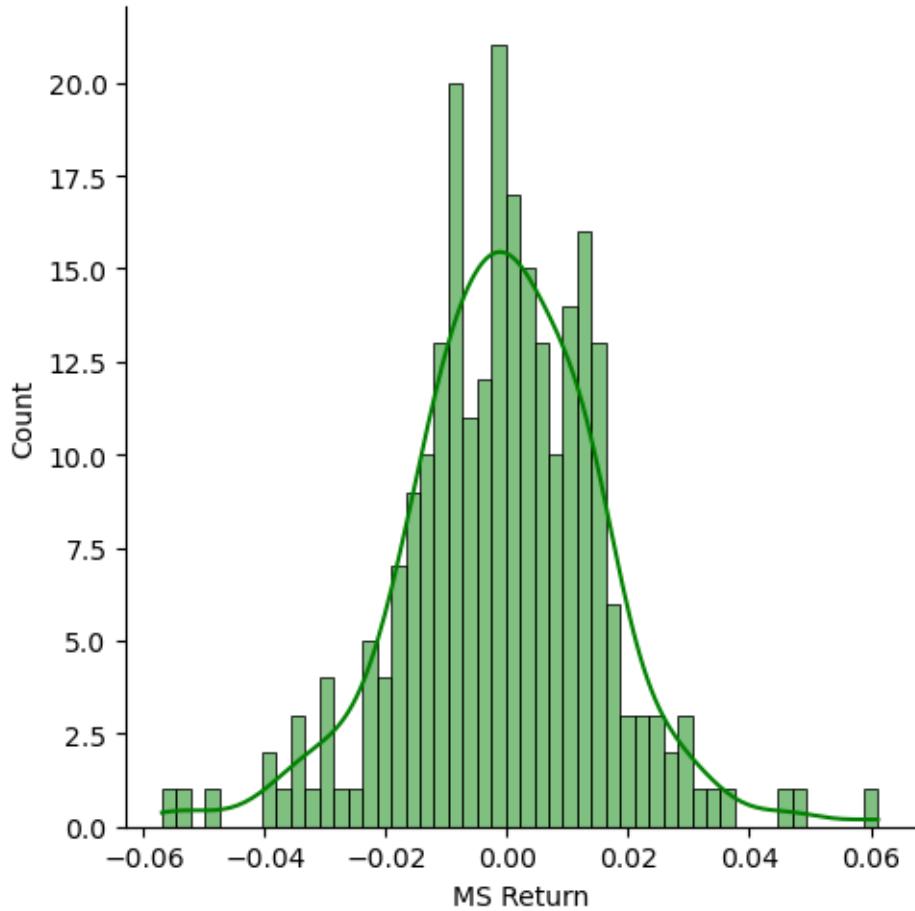
```
[77]: returns['2015-01-01':'2015-12-31'].std()
#Overall risk is about the same for every stock in 2015, but BOFA and MS have the highest variation about the mean
```

```
[77]: 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
```

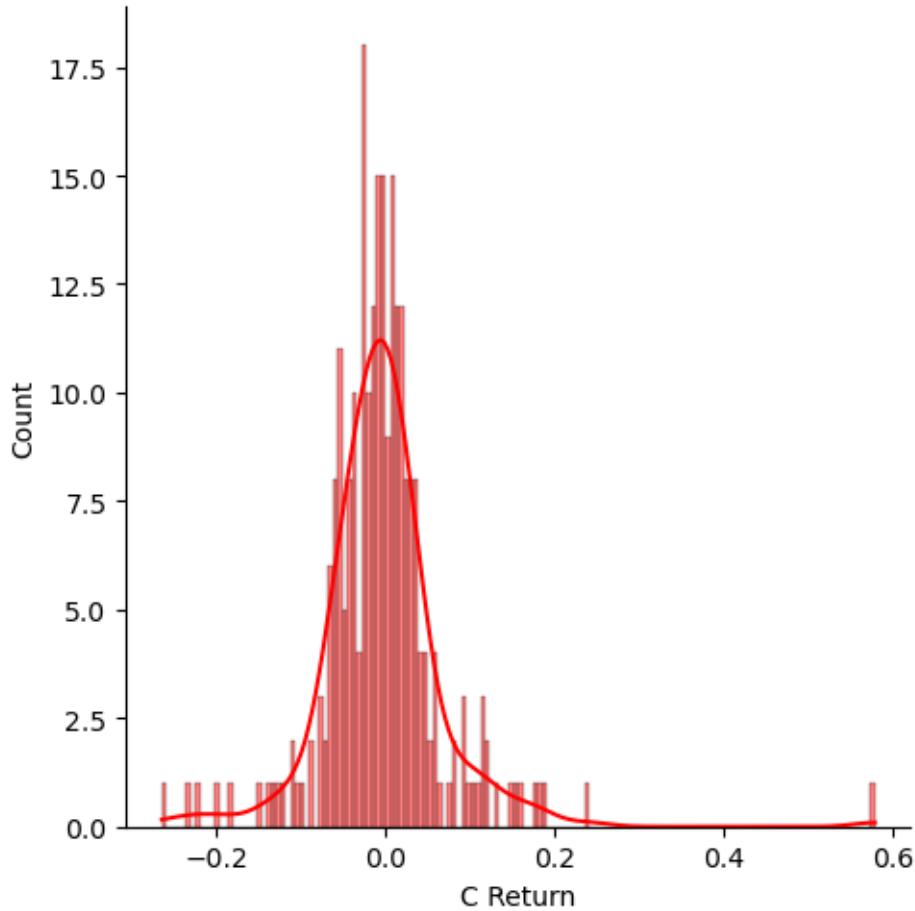
** Create a distplot using seaborn of the 2015 returns for Morgan Stanley **

```
[83]: sns.distplot(returns['2015-01-01':'2015-12-31']['MS Return'], kde=True, bins=50, color='Green')
plt.show()
```



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

```
[84]: sns.distplot(returns['2008-01-01':'2008-12-31']['C_Return'], kde=True, bins=150, color='Red')
plt.show()
```



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

```
[90]: import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
import plotly.express as px

# Optional Plotly Method Imports
import plotly
```

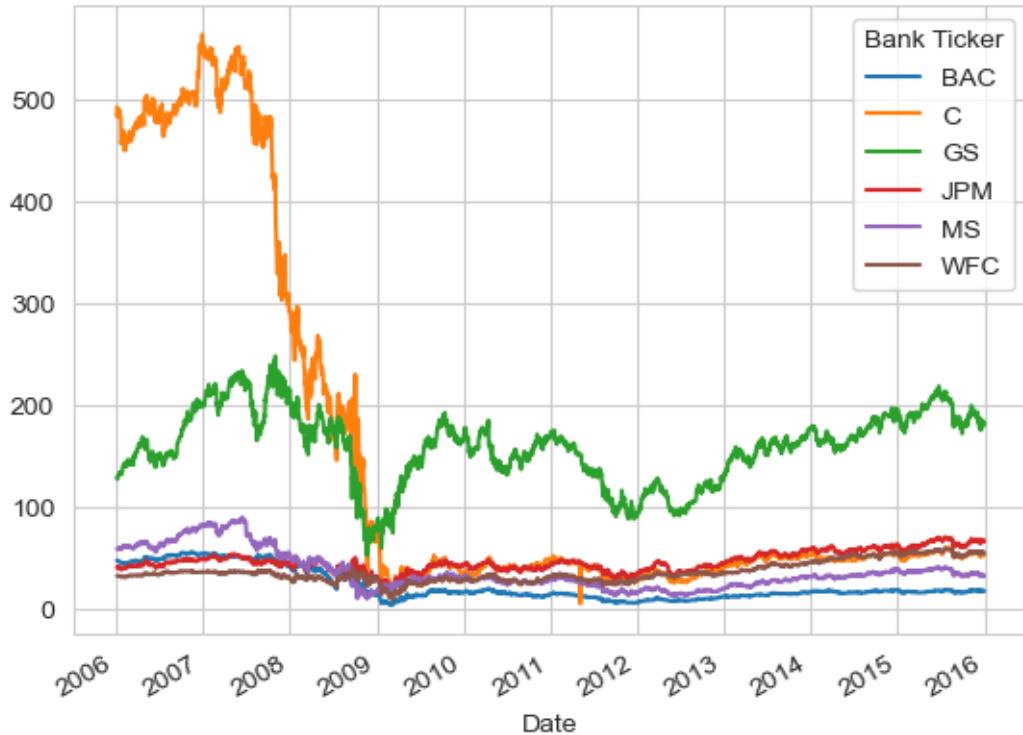
```

import cufflinks as cf
cf.go_offline()
from plotly import __version__
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
#init_notebook_mode(connected=True)

```

** 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.)**

[95]: (bank_stocks.xs(key='Close', axis=1, level='Stock Info')).plot()
plt.show()



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**

[157]: bac_close = df['BAC']['Close']
bac_2008 = bac_close.loc['2008-01-01':'2009-02-01']
bac_rolling_avg = bac_2008.rolling(window=30).mean()

```

plt.figure(figsize=(12,6))
bac_2008.plot(label='BAC Close', color='green')
bac_rolling_avg.plot(label='30 Day Average', color='blue')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()

```



```

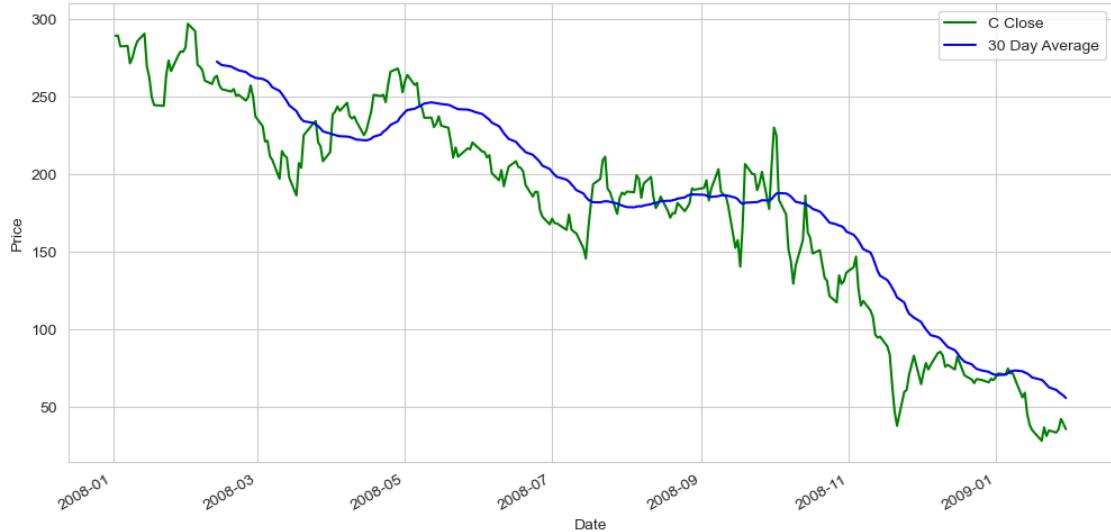
[160]: c_close = df['C']['Close']
c_2008 = c_close.loc['2008-01-01':'2009-02-01']
c_rolling_avg = c_2008.rolling(window=30).mean()

```

```

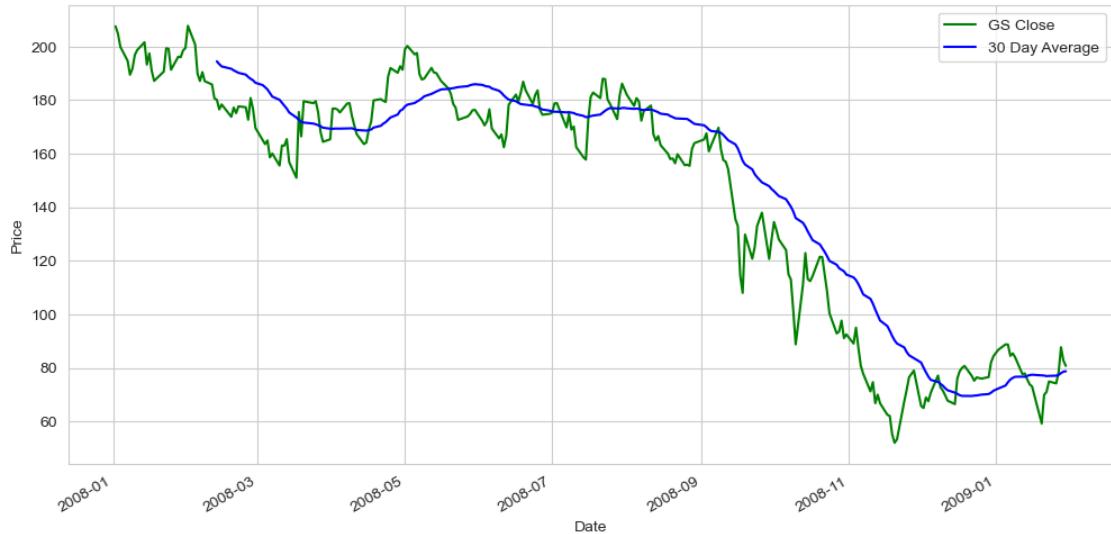
plt.figure(figsize=(12,6))
c_2008.plot(label='C Close', color='green')
c_rolling_avg.plot(label='30 Day Average', color='blue')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()

```



```
[161]: gs_close = df['GS']['Close']
gs_2008 = gs_close.loc['2008-01-01':'2009-02-01']
gs_rolling_avg = gs_2008.rolling(window=30).mean()
```

```
plt.figure(figsize=(12,6))
gs_2008.plot(label='GS Close', color='green')
gs_rolling_avg.plot(label='30 Day Average', color='blue')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```



```
[162]: jpm_close = df['JPM']['Close']
jpm_2008 = jpm_close.loc['2008-01-01':'2009-02-01']
jpm_rolling_avg = jpm_2008.rolling(window=30).mean()

plt.figure(figsize=(12,6))
jpm_2008.plot(label='JPM Close', color='green')
jpm_rolling_avg.plot(label='30 Day Average', color='blue')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```



```
[163]: ms_close = df['MS']['Close']
ms_2008 = ms_close.loc['2008-01-01':'2009-02-01']
ms_rolling_avg = ms_2008.rolling(window=30).mean()

plt.figure(figsize=(12,6))
ms_2008.plot(label='MS Close', color='green')
ms_rolling_avg.plot(label='30 Day Average', color='blue')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```



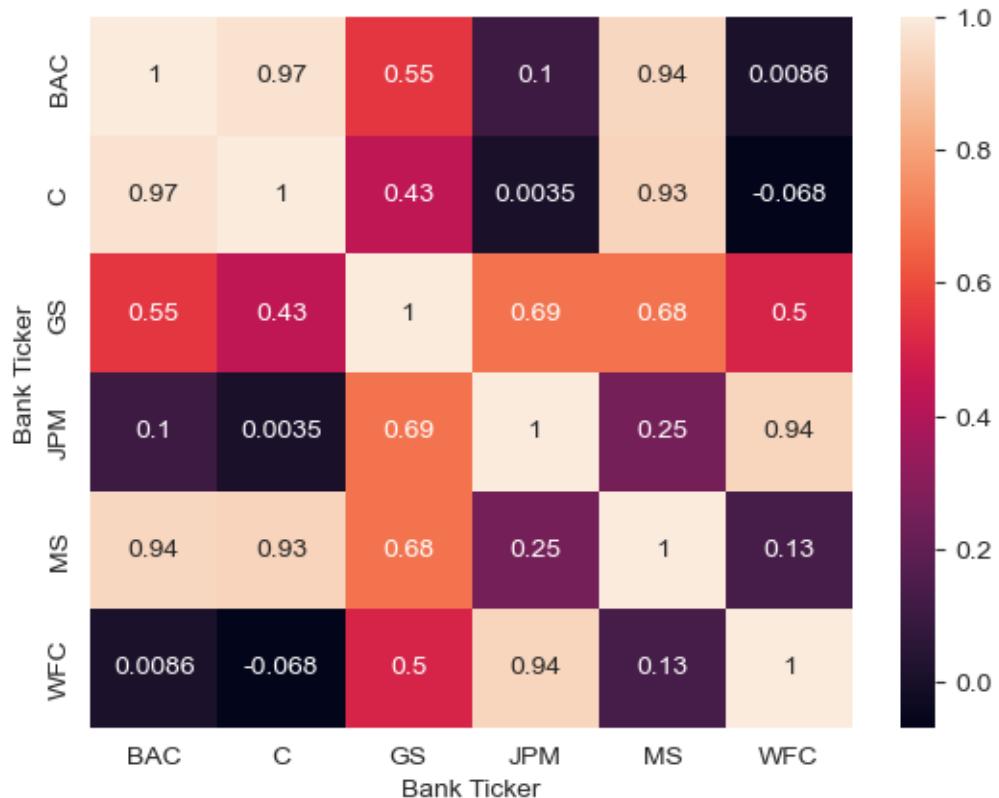
```
[164]: wfc_close = df['WFC']['Close']
wfc_2008 = wfc_close.loc['2008-01-01':'2009-02-01']
wfc_rolling_avg = wfc_2008.rolling(window=30).mean()
```

```
plt.figure(figsize=(12,6))
wfc_2008.plot(label='WFC Close', color='green')
wfc_rolling_avg.plot(label='30 Day Average', color='blue')
plt.legend()
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```



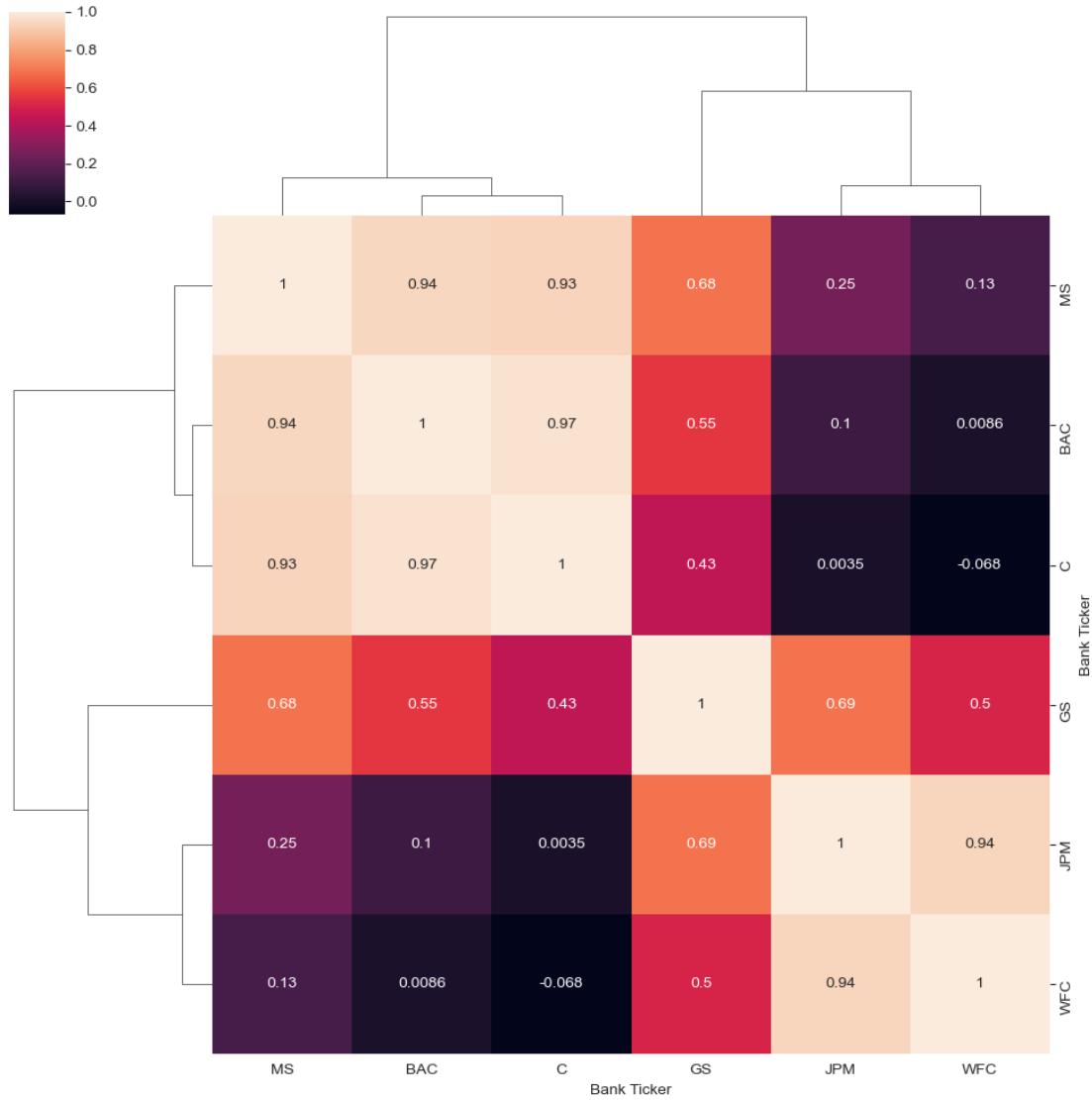
** Create a heatmap of the correlation between the stocks Close Price.**

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



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

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



4 Part 2 Synthesis of Findings

BAC and C suffered the steepest declines — both fell over 60% in 2008

WFC had relatively milder decline, supported by stronger fundamentals and bailout participation.

JPM had less volatility but still showed a steady decline post-Lehman

MS and GS had a moderate decline until hitting the post-Lehman collapse period triggering a more dramatic decline.

The 30-day moving average was effective in highlighting long-term trends and helped differentiate momentary volatility from sustained crashes.

[]: