Finance_Taking_Subjects

October 5, 2024

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.

[6]:

```
pip install yfinance
```

```
Requirement already satisfied: yfinance in
/home/fischer/anaconda3/lib/python3.11/site-packages (0.2.28)
Requirement already satisfied: pandas>=1.3.0 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (2.2.1)
Requirement already satisfied: numpy>=1.16.5 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (1.25.2)
Requirement already satisfied: requests>=2.31 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (4.9.1)
```

```
Requirement already satisfied: appdirs>=1.4.4 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (2022.7)
Requirement already satisfied: frozendict>=2.3.4 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (2.3.8)
Requirement already satisfied: beautifulsoup4>=4.11.1 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (4.12.2)
Requirement already satisfied: html5lib>=1.1 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: six>=1.9 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: tzdata>=2022.7 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
pandas>=1.3.0->yfinance) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
requests>=2.31->yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
requests>=2.31->yfinance) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in
/home/fischer/anaconda3/lib/python3.11/site-packages (from
requests>=2.31->yfinance) (2023.11.17)
Note: you may need to restart the kernel to use updated packages.
```

[1]:

```python
import os
import yfinance as yf
```

[2]:

```python
import pandas_datareader.data as web
from pandas_datareader import data, wb
import pandas as pd
import numpy as np
import datetime
```

```
yf.pdr_override()
%matplotlib inline
%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.

[3]:

```
#start = datetime.datetime(2006, 1, 1)
#end = datetime.datetime(2024, 1, 1)
```

[11]:

```
# Bank of America
BAC = yf.download("BAC",  start="2006-1-1", end="2024-1-1")
#BAC = yf.Ticker("AAPL", 'yahoo', start, end)

# CitiGroup
C = yf.download("C",  start="2006-1-1", end="2024-1-1")

# Goldman Sachs
GS = yf.download("GS", start="2006-1-1", end="2024-1-1")

# JPMorgan Chase
JPM = yf.download("JPM",start="2006-1-1", end="2024-1-1")

# Morgan Stanley
MS = yf.download("MS", start="2006-1-1", end="2024-1-1")

# Wells Fargo
WFC = yf.download("WFC",  start="2006-1-1", end="2024-1-1")
```

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

```
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')


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

/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')


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

/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')


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

/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')


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

/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')


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

/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
```

[6]:

```python
data = yf.download("AAPL", start="2022-01-01", end="2022-12-31")
print(data)
```

```
[*********************100%%**********************]  1 of 1 completed
                  Open        High         Low       Close   Adj Close  \
Date
2022-01-03  177.830002  182.880005  177.710007  182.009995  179.273605
2022-01-04  182.630005  182.940002  179.119995  179.699997  176.998367
```

```
2022-01-05  179.610001  180.169998  174.639999  174.919998  172.290207
2022-01-06  172.699997  175.300003  171.639999  172.000000  169.414108
2022-01-07  172.889999  174.139999  171.029999  172.169998  169.581573
…                  …           …           …           …           …
2022-12-23  130.919998  132.419998  129.639999  131.860001  130.631363
2022-12-27  131.380005  131.410004  128.720001  130.029999  128.818420
2022-12-28  129.669998  131.029999  125.870003  126.040001  124.865593
2022-12-29  127.989998  130.479996  127.730003  129.610001  128.402328
2022-12-30  128.410004  129.949997  127.430000  129.929993  128.719330

               Volume
Date
2022-01-03  104487900
2022-01-04   99310400
2022-01-05   94537600
2022-01-06   96904000
2022-01-07   86709100
…                 …
2022-12-23   63814900
2022-12-27   69007800
2022-12-28   85438400
2022-12-29   75703700
2022-12-30   77034200

[251 rows x 6 columns]
```

```
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
```

[10]:

```python
apple = yf.Ticker("AAPL")
print(apple)  # General information about Apple Inc.
```

```
yfinance.Ticker object <AAPL>
```

[4]:

```python
# Could also do this for a Panel Object
from pandas_datareader import data as pdr

df = pdr.get_data_yahoo(['BBAS3.SA', 'SANB4.SA', 'ITUB4.SA', 'BBDC4.
 ↪SA','AAPL','MSFT','META'], start='2008-01-01',end='2023-01-01')
```

```
[                       0%%                       ]
```

```
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
```

```
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
```

```
[********************* 43%%                      ]  3 of 7 completed
```

```
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
```

```
[*********************100%%**********************]  7 of 7 completed
```

```
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
/home/fischer/anaconda3/lib/python3.11/site-packages/yfinance/utils.py:771:
FutureWarning: The 'unit' keyword in TimedeltaIndex construction is deprecated
and will be removed in a future version. Use pd.to_timedelta instead.
  df.index += _pd.TimedeltaIndex(dst_error_hours, 'h')
```

[5]:
```
df.head()
```

[5]:
```
           Adj Close                                                  \
                AAPL  BBAS3.SA   BBDC4.SA  ITUB4.SA META       MSFT
Date
2008-01-02  5.876341  5.322039  11.270322  7.942460  NaN  25.489273
2008-01-03  5.879056  5.241672  11.040893  7.792808  NaN  25.597826
2008-01-04  5.430278  5.180949  10.911146  7.847555  NaN  24.881344
2008-01-07  5.357594  5.304179  11.113206  7.847555  NaN  25.047791
2008-01-08  5.164871  5.413121  11.283357  7.920557  NaN  24.208290


                          Close                        … Open             \
              SANB4.SA     AAPL BBAS3.SA   BBDC4.SA  … META       MSFT
Date                                                 …
2008-01-02 -29.529142  6.958571   14.900  22.030840  …  NaN  35.790001
2008-01-03 -29.529142  6.961786   14.675  21.577751  …  NaN  35.220001
2008-01-04 -27.068382  6.430357   14.505  21.324188  …  NaN  35.189999
2008-01-07 -27.068382  6.344286   14.850  21.719082  …  NaN  34.549999
2008-01-08 -29.529142  6.116071   15.155  22.051622  …  NaN  34.709999
```

```
                             Volume                                         \
             SANB4.SA          AAPL    BBAS3.SA   BBDC4.SA   ITUB4.SA META
Date
2008-01-02  12.007016  1.079179e+09   9456600.0        0.0  6442543.0  NaN
2008-01-03  12.007016  8.420664e+08  10427400.0        0.0  7212266.0  NaN
2008-01-04  11.506723  1.455832e+09   8461600.0  5694095.0  7374122.0  NaN
2008-01-07  11.506723  2.072193e+09   5468400.0        0.0  7597580.0  NaN
2008-01-08  11.506723  1.523816e+09   6253200.0        0.0  5372057.0  NaN


                  MSFT SANB4.SA
Date
2008-01-02  63004200.0   3997.0
2008-01-03  49599600.0   5996.0
2008-01-04  72090800.0  11992.0
2008-01-07  80164300.0  17989.0
2008-01-08  79148300.0   9994.0

[5 rows x 42 columns]
```

[6]:

```
df.tail()
```

[6]:

```
             Adj Close                                               \
                  AAPL   BBAS3.SA   BBDC4.SA   ITUB4.SA        META
Date
2022-12-26         NaN  15.610377  12.670477  22.203974         NaN
2022-12-27  128.818405  15.091951  12.610474  22.078074  116.529137
2022-12-28  124.865593  15.362241  12.996242  22.527725  115.272934
2022-12-29  128.402344  15.388828  12.987670  22.482761  119.899002
2022-12-30  128.719330        NaN        NaN        NaN  119.978760

                                        Close                      …  \
                  MSFT   SANB4.SA        AAPL   BBAS3.SA BBDC4.SA  …
Date                                                               …
2022-12-26         NaN  13.518714         NaN  17.615000    14.78  …
2022-12-27  233.600662  13.491586  130.029999  17.030001    14.71  …
2022-12-28  231.205109  13.572971  126.040001  17.334999    15.16  …
2022-12-29  237.593246  13.509672  129.610001  17.365000    15.15  …
2022-12-30  236.420120        NaN  129.929993        NaN      NaN  …

                  Open                           Volume              \
                  META        MSFT SANB4.SA        AAPL    BBAS3.SA
Date
2022-12-26         NaN         NaN    14.95         NaN   9105400.0
2022-12-27  117.930000  238.699997    14.97  69007800.0  17731600.0
2022-12-28  116.250000  236.889999    14.92  85438400.0  21450400.0
```

```
2022-12-29  116.400002  235.649994    15.07  75703700.0  18673000.0
2022-12-30  118.160004  238.210007      NaN  77034200.0         NaN


              BBDC4.SA    ITUB4.SA        META        MSFT  SANB4.SA
Date
2022-12-26  13937200.0  15730800.0         NaN         NaN  104100.0
2022-12-27  78235200.0  17203600.0  21392300.0  16688600.0  104500.0
2022-12-28  45117800.0  22696400.0  19612500.0  17457100.0   99900.0
2022-12-29  41911700.0  24799700.0  22366200.0  19770700.0   93500.0
2022-12-30         NaN         NaN  19583800.0  21938500.0       NaN

[5 rows x 42 columns]
```

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

[14]:

```
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.**

[15]:

```
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):**

[16]:

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

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

[17]:

```
bank_stocks.head()
```

[17]:

```
Bank Ticker        BAC                                                       \
Stock Info        Open       High        Low      Close  Adj Close    Volume
Date
2006-01-03   46.919998  47.180000  46.150002  47.080002  31.544907  16296700
2006-01-04   47.000000  47.240002  46.450001  46.580002  31.209888  17757900
2006-01-05   46.580002  46.830002  46.320000  46.639999  31.250111  14970700
2006-01-06   46.799999  46.910000  46.349998  46.570000  31.203192  12599800
2006-01-09   46.720001  46.970001  46.360001  46.599998  31.223289  15619400

Bank Ticker           C                                      …         MS  \
Stock Info         Open        High         Low       Close  …        Low
Date                                                         …
2006-01-03   490.000000  493.799988  481.100006  492.899994  …  56.740002
2006-01-04   488.600006  491.000000  483.500000  483.799988  …  58.349998
2006-01-05   484.399994  487.799988  484.000000  486.200012  …  58.020000
2006-01-06   488.799988  489.000000  482.000000  486.200012  …  58.049999
```

```
2006-01-09   486.000000  487.399994  483.000000  483.899994  …  58.619999

Bank Ticker                                        WFC                      \
Stock Info        Close   Adj Close    Volume       Open        High         Low
Date
2006-01-03   58.310001  32.661312  5377000  31.600000  31.975000  31.195000
2006-01-04   58.349998  32.683716  7977800  31.799999  31.820000  31.365000
2006-01-05   58.509998  32.773346  5778000  31.500000  31.555000  31.309999
2006-01-06   58.570000  32.806938  6889800  31.580000  31.775000  31.385000
2006-01-09   59.189999  33.154236  4144500  31.674999  31.825001  31.555000

Bank Ticker
Stock Info        Close   Adj Close     Volume
Date
2006-01-03   31.900000  18.979553  11016400
2006-01-04   31.530001  18.759413  10870000
2006-01-05   31.495001  18.738594  10158000
2006-01-06   31.680000  18.848661   8403800
2006-01-09   31.674999  18.845688   5619600

[5 rows x 36 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?**

[18]:

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

[18]:

```
Bank Ticker
BAC     54.900002
C      564.099976
GS     423.850006
JPM    171.779999
MS     108.730003
WFC     65.930000
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:**

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

[19]:

```
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.**

[20]:

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

[20]:

```
            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?**

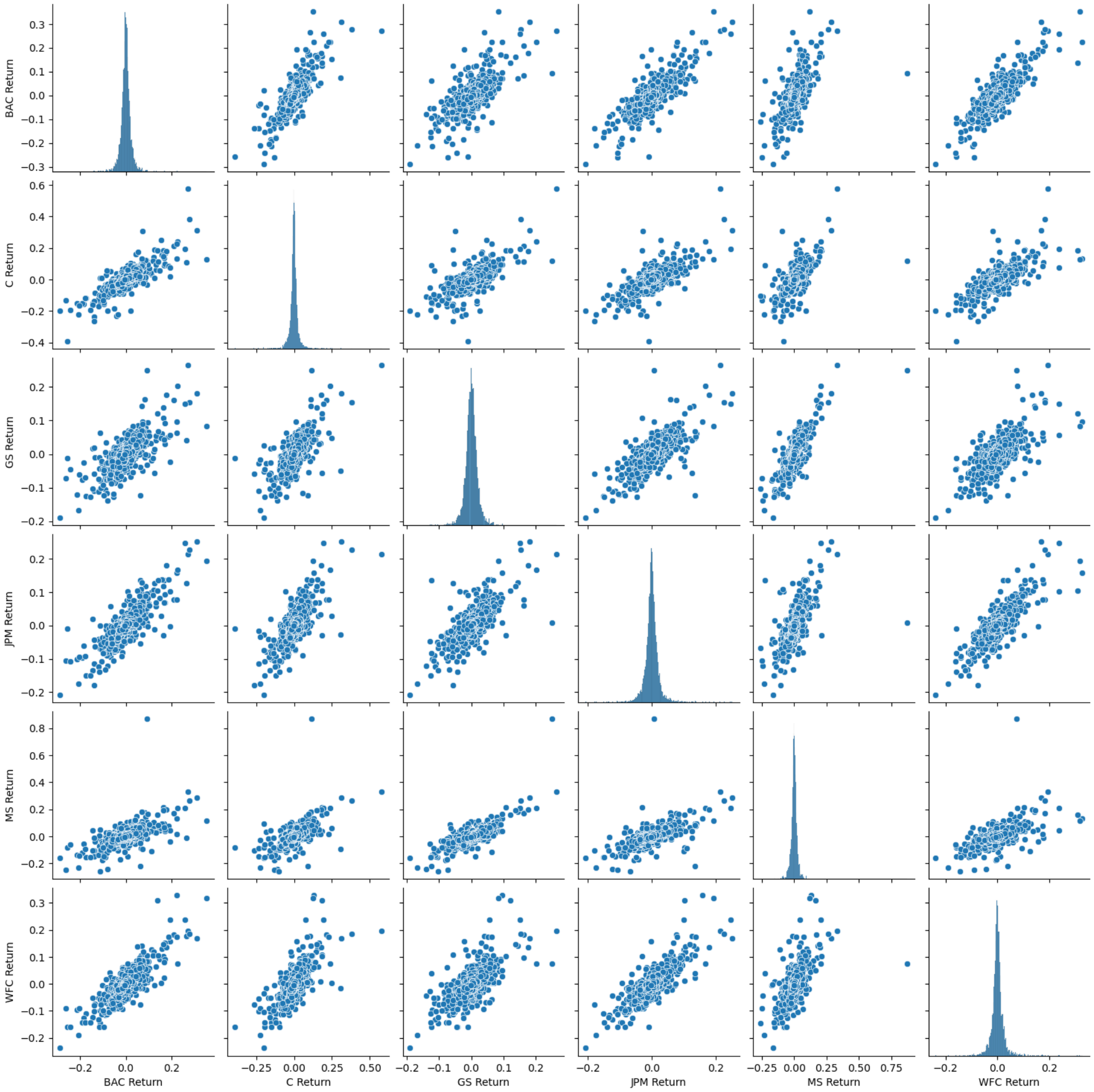
[21]:

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

```
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
```

```
  with pd.option_context('mode.use_inf_as_na', True):
```

[21]:
```
<seaborn.axisgrid.PairGrid at 0x788a52870490>
```



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?**

[22]:
```
# Worst Drop (4 of them on Inauguration day)
returns.idxmin()
```

```
[22]: 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? **

Citigroup had a stock split.

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

```
[23]: 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?**

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

```
[24]: BAC Return    0.030557
      C Return      0.032251
      GS Return     0.022715
      JPM Return    0.023828
      MS Return     0.031324
      WFC Return    0.026434
      dtype: float64
```

```
[26]: returns.loc['2015-01-01':'2015-12-31'].std() # Very similar risk profiles, but␣
        ↪Morgan Stanley or BofA
```

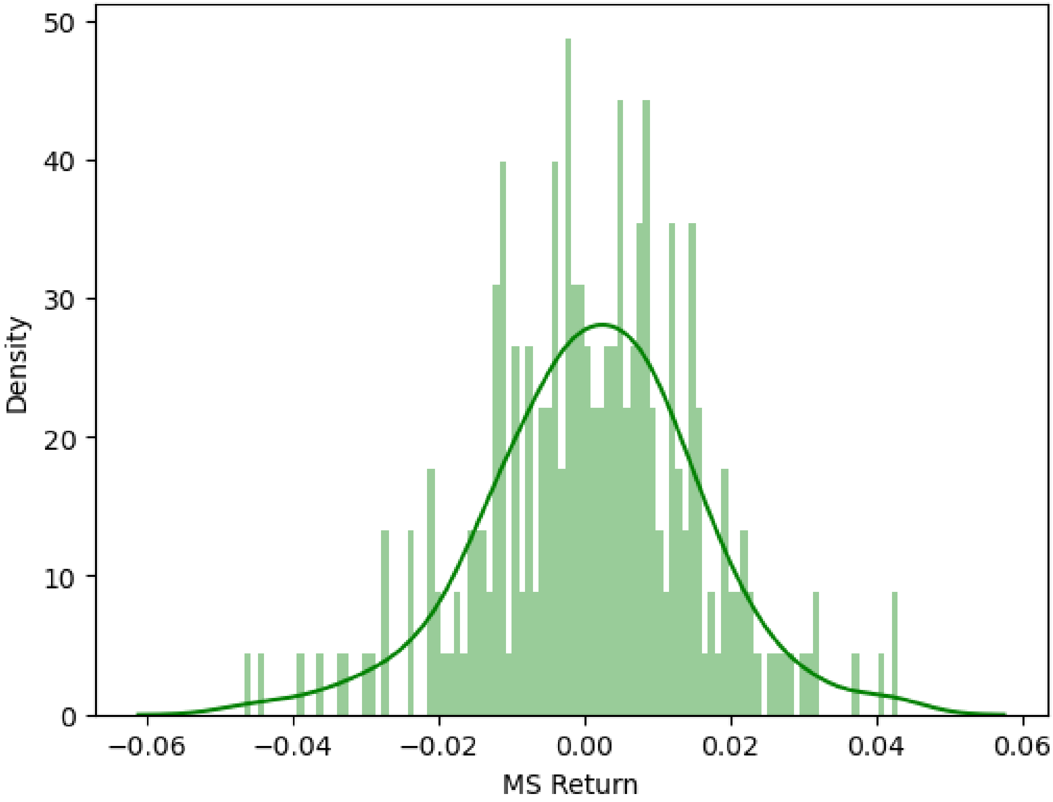
```
[26]: BAC Return    0.016163
      C Return      0.015289
      GS Return     0.014046
      JPM Return    0.014017
      MS Return     0.016249
      WFC Return    0.012591
      dtype: float64
```

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

```
[27]: sns.distplot(returns.loc['2019-01-01':'2019-12-31']['MS␣
       ↪Return'],color='green',bins=100)
```

```
/tmp/ipykernel_5254/1391145458.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(returns.loc['2019-01-01':'2019-12-31']['MS
Return'],color='green',bins=100)
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
[27]: <Axes: xlabel='MS Return', ylabel='Density'>
```

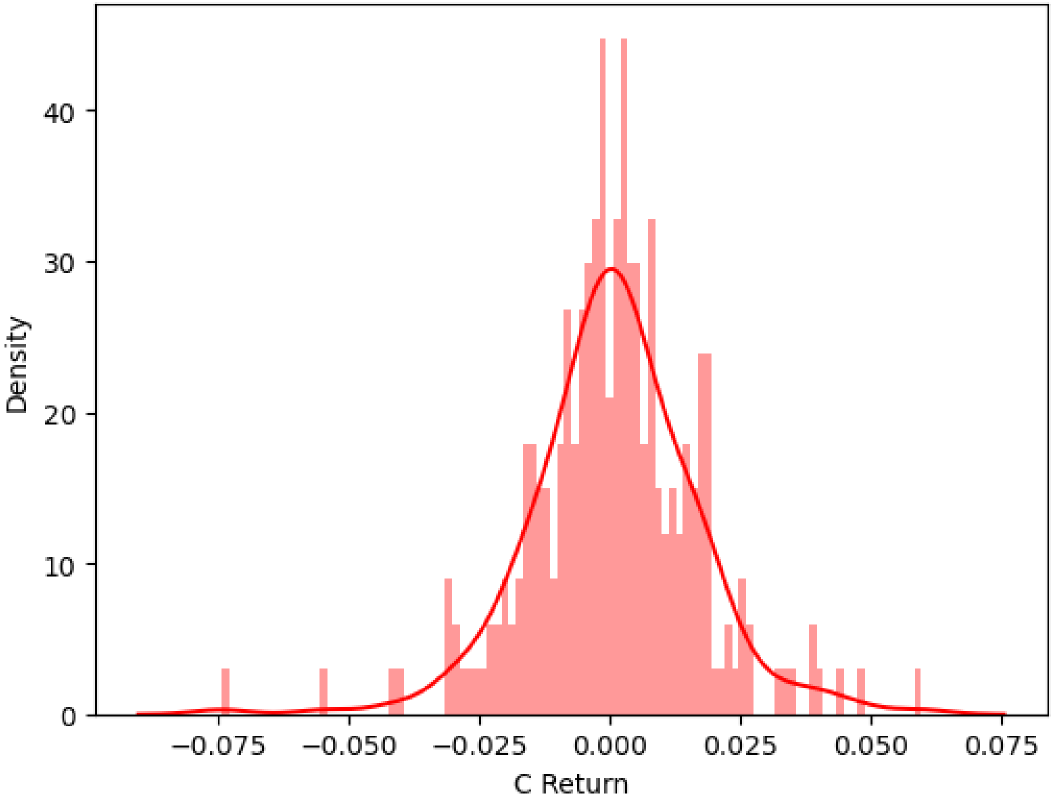


** Create a distplot using seaborn of the 2008 returns for CitiGroup **

```
[28]: sns.distplot(returns.loc['2023-01-01':'2023-12-31']['C␣
       ↪Return'],color='red',bins=100)
```

```
/tmp/ipykernel_5254/3892473082.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(returns.loc['2023-01-01':'2023-12-31']['C
Return'],color='red',bins=100)
/home/fischer/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
[28]: <Axes: xlabel='C Return', ylabel='Density'>
```



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

[29]:

```
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()
```

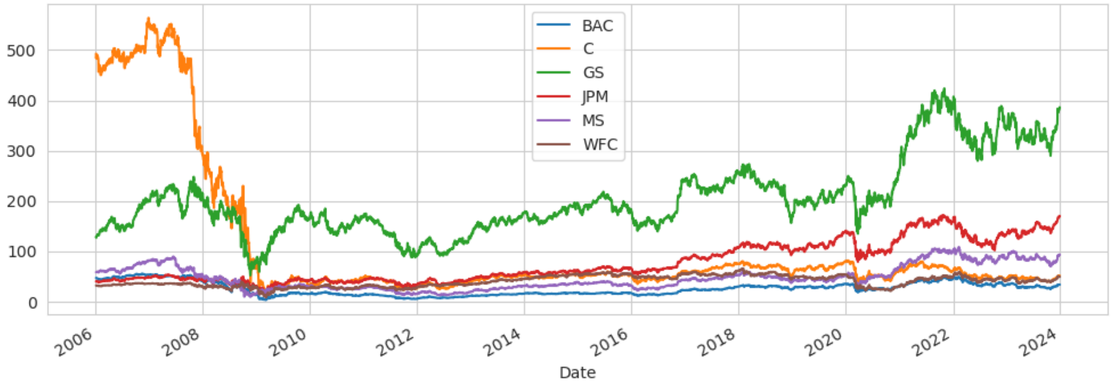
** Create a line plot showing Close price for each bank for the entire index of time. (Hint: Try using a for loop, or use .xs to get a cross section of the data.)**

[30]:

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

[30]:

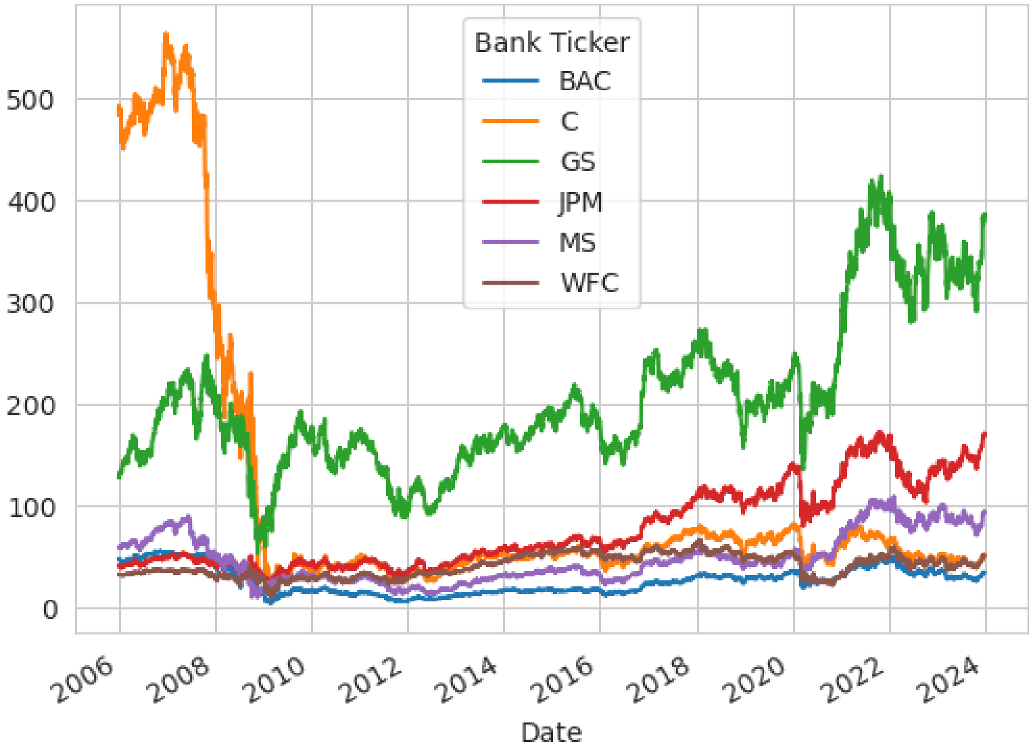
```
<matplotlib.legend.Legend at 0x788a44d64b10>
```



[31]:

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

```
[31]: <Axes: xlabel='Date'>
```



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

```
/home/fischer/anaconda3/lib/python3.11/site-
packages/cufflinks/plotlytools.py:117: FutureWarning:

DatetimeIndex.format is deprecated and will be removed in a future version.
Convert using index.astype(str) or index.map(formatter) instead.
```

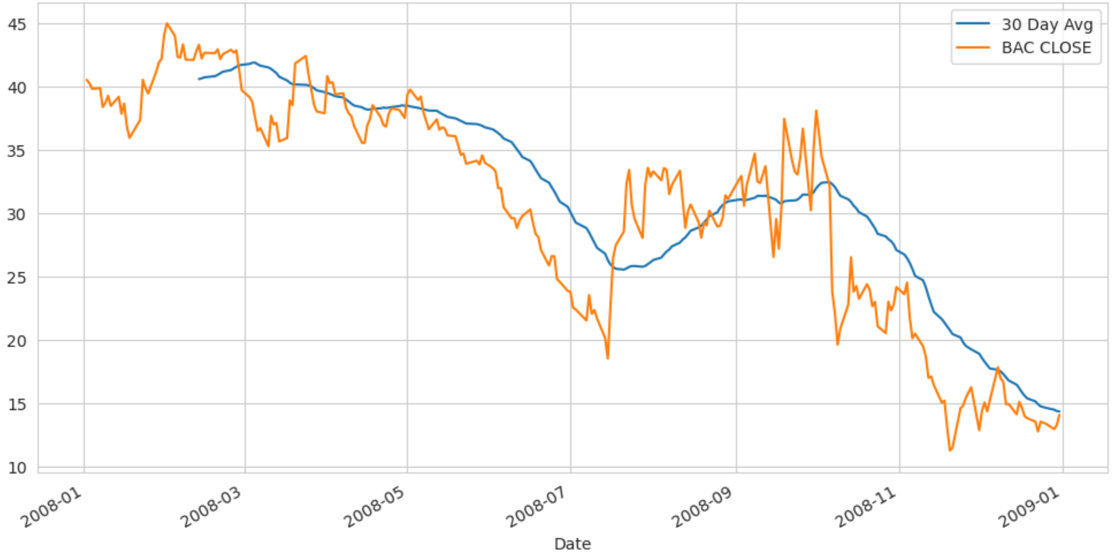
3.1 Moving Averages

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**

```
[34]: 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()
```

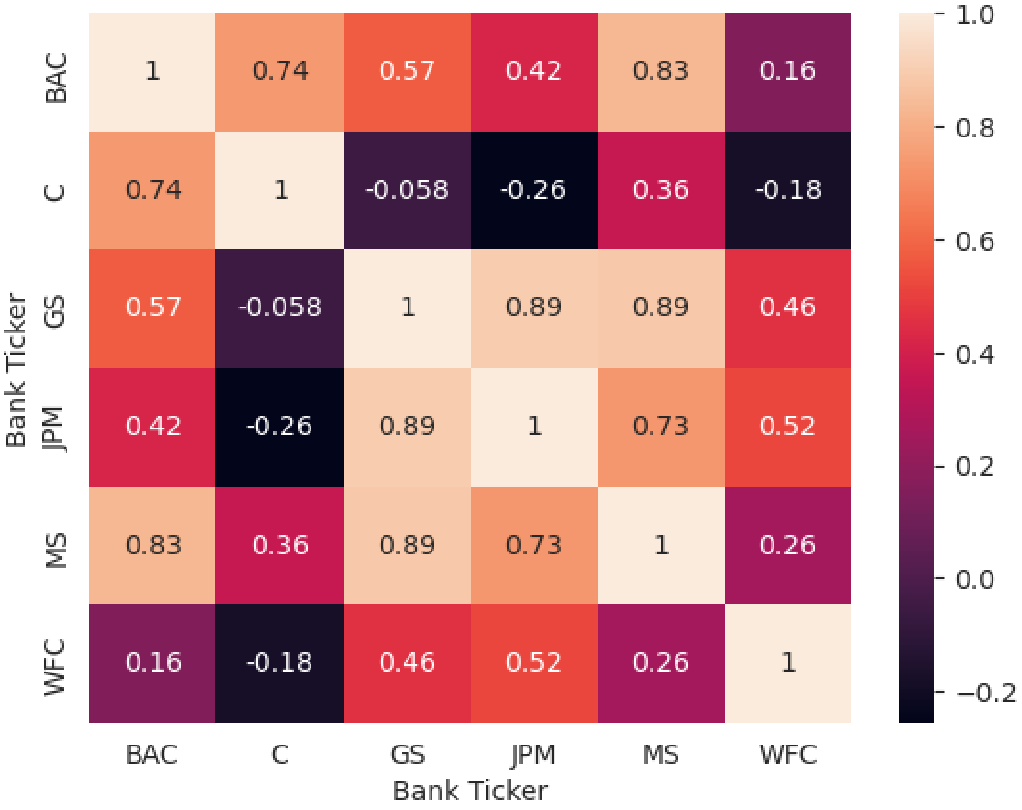
```
[34]: <matplotlib.legend.Legend at 0x788a4425e4d0>
```



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

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

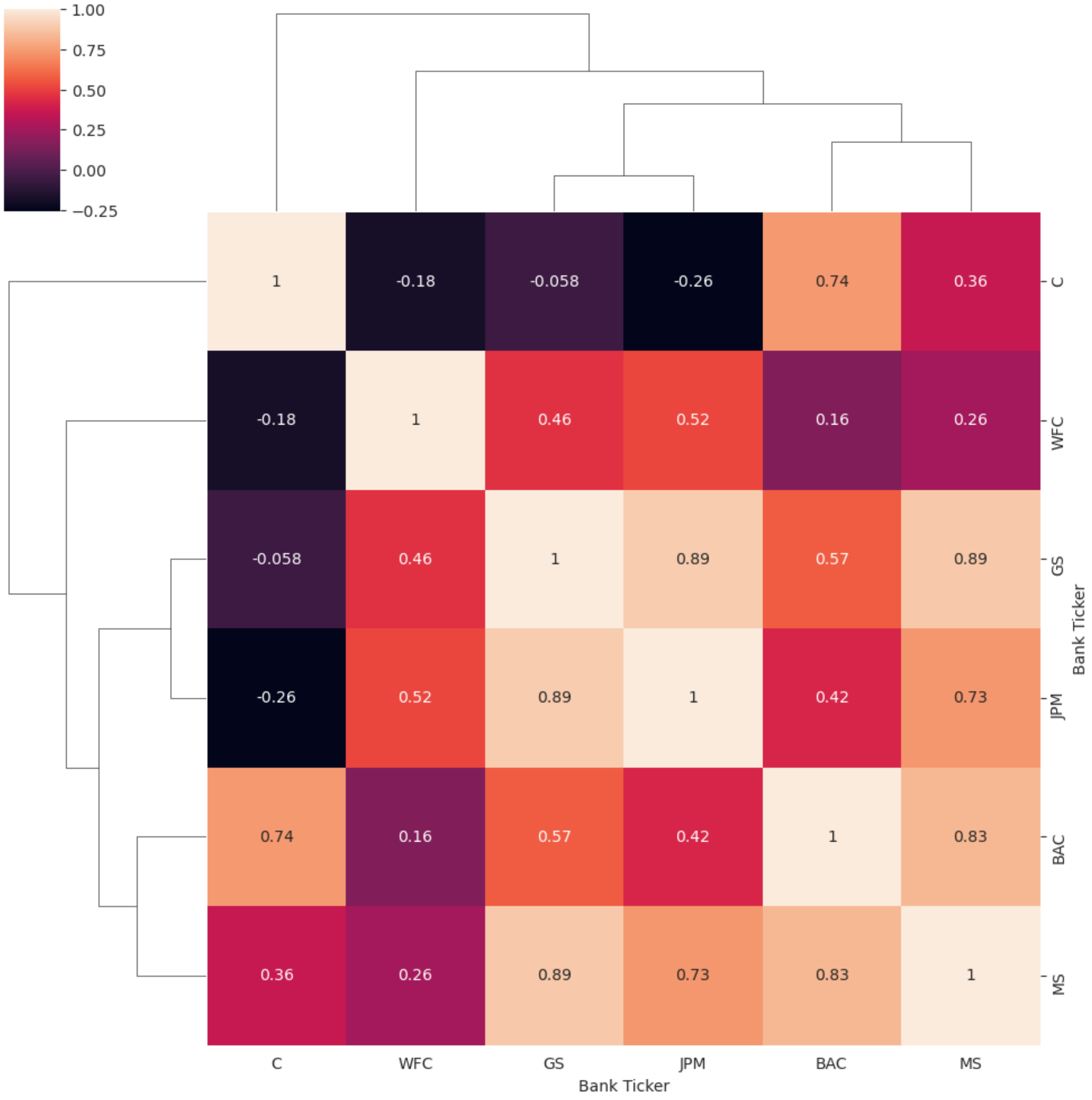
```
[35]: <Axes: xlabel='Bank Ticker', ylabel='Bank Ticker'>
```



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

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

```
[36]: <seaborn.matrix.ClusterGrid at 0x788a442ecc50>
```



[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.**

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

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

```
[38]: MS['Close'].loc['2022-01-01':'2023-01-01'].
       ↪ta_plot(study='sma',periods=[13,21,55],title='Simple Moving Averages')
```

```
/home/fischer/anaconda3/lib/python3.11/site-
packages/cufflinks/plotlytools.py:117: FutureWarning:

DatetimeIndex.format is deprecated and will be removed in a future version.
Convert using index.astype(str) or index.map(formatter) instead.

/home/fischer/anaconda3/lib/python3.11/site-
packages/cufflinks/plotlytools.py:117: FutureWarning:

DatetimeIndex.format is deprecated and will be removed in a future version.
Convert using index.astype(str) or index.map(formatter) instead.
```

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

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

```
/home/fischer/anaconda3/lib/python3.11/site-
packages/cufflinks/plotlytools.py:117: FutureWarning:

DatetimeIndex.format is deprecated and will be removed in a future version.
Convert using index.astype(str) or index.map(formatter) instead.
```