Finding good pairs

↙ 1 backlink

Here we look at the various statistical analysis methods employed for findings the right pairs to trade.

The code below uses the API of Alpha Vintage to obtain historical and real-time data for markets. They support BSE, and until a year ago, they had support for NSE as well. For this article, we are using examples that we got from various websites on the web. Alpha Vintage API has a freemium model where we get a rate limit of 5 requests per minute and 500 requests per day.

▼ ***Correlation*** between stocks is the first indicator that we are going to look at.

What is correlation?

A correlation is a relationship between two sets of data.

For example, in the equity markets, you may notice that stocks like Microsoft (MSFT) and Apple (AAPL) both tend to rise and fall at the same time. The price behavior between the two stocks is not an exact match, but there is enough similarity to say there is a relationship. In this scenario, we can say MSFT and AAPL have a positive correlation.

Further, there are often relationships across markets, such as equities and bonds or precious metals. We often also see a correlation between financial instruments and economic data or even sentiment indicators.

Why do correlations matter?

There are several reasons why correlations are important, here a few benefits of tracking them in the markets –

1. **Insights** – keeping track of different relationships can provide insight into where the markets are headed. A good example is when the markets turned sharply lower in late February due to the Coronavirus escalation. The price of gold, which is known as an asset investors turn to when their mood for risky investment sours, rose sharply the trading day before the big initial drop in stocks. It acted as a warning signal for those equity traders mindful of the inverse correlation between the two.
2. **Strength in correlated moves** – It's much easier to assess trends when there is a correlated move. In other words, if a bulk of the tech stocks on your watchlist are rising, it's probably safe to say the sector is bullish or that there is strong demand.
3. **Diversification** – To make sure you have some diversification in your portfolio, it's a good idea to make sure the assets within it aren't all strongly correlated to each other.
4. **Signal confirmation** – Let's say you want to buy a stock because your analysis shows that it is bullish. You could analyze another stock with a positive correlation to make sure it provides a similar signal.

Correlation doesn't imply causation

A popular saying among the statistics crowd is “correlation does not imply causation.” It comes up often, and it’s important to understand its meaning. Essentially, correlations can provide valuable insights, but you’re bound to come across situations that might imply a correlation where a relationship does not exist.

As an example, data has shown a sharp rise in Netflix subscribers due to the lockdown that followed the Coronavirus escalation. The idea is that people are forced to stay at home and therefore are more likely to watch tv.

The same scenario has resulted in a rise in electricity bills. People are using more electricity at home compared to when they were at work all day.

If you were blindly comparing the rise in Netflix subscribers versus the rise in electricity usage during the month of lockdown, you might reasonably conclude that the two have a relationship.

However, having some perspective on the manner, it is clear that the two are not related and that it is not likely that fluctuations in one will impact the other moving forward. Rather, the lockdown, an external variable, is the causation for both of these trends.

What is a correlation coefficient?

We’ve discussed that fluctuations in the stock prices of Apple and Microsoft tend to have a relationship. You might then notice other tech companies also correlate well with the two.

But not all relationships are equal, and the correlation coefficient can help assess the strength of a correlation.

There are a few different ways of calculating a correlation coefficient, but the most popular methods result in a number between -1 and +1.

The closer the number is to +1, the stronger the relationship. If the figure is close to -1, it indicates that there is a strong inverse relationship.

In the finance world, an inverse relationship is where one asset rises while the other drops. Stocks and gold prices have a long-standing inverse relationship.

The closer the correlation coefficient is to zero, the more likely it is that the two variables being compared don’t have any relationship to each other.

Breaking down the math to calculate the correlation coefficient

$$r = \frac{\sum_(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_(x_i - \overline{x})^2 \sum_(y_i - \overline{y})^2}}$$

The above formula is what's used to calculate a correlation coefficient using the Pearson method. We will break down this formula.

There are libraries available that can do this automatically, but the following example will show how we can make the calculation manually.

We will start by creating a dataset. We can use the Numpy library to create some random data for us. Here is the code:

```
import numpy as np import pandas as pd df =
pd.DataFrame(np.random.randint(0,10,size=(5, 2)),
columns=list('xy'))
```

The image below shows what my DataFrame looks like. If you're following along, the data will look different for you as Numpy is filling in random numbers. But the format should look the same.

```
import numpy as np
import pandas as pd
df = pd.DataFrame(np.random.randint(0,10,size=(5, 2)), columns=list('xy'))
```

```
df
```

	x	y
0	2	1
1	4	3
2	8	3
3	1	5
4	1	8

Now that we have a dataset let's move on to the formula. We will start by separating the first part of the formula.

$$\sum (x_i - \overline{x})(y_i - \overline{y})$$

We can break this down further.

$x_i - \overline{x}$

For the formula above, we need to take each value of x and subtract it by the mean of x.

We can use the `mean()` function in Pandas to create the mean for us. Like this:

```
df.x.mean()
```

But we still need to subtract the mean from x. And we also need to temporarily store this information somewhere. Let's create a new column for that and call it step1.

```
df['step1'] = df.x - df.x.mean()
```

This is what our DataFrame looks like at this point.

```
df['step1'] = df.x - df.x.mean()
```

```
df
```

	x	y	step1
0	2	1	-1.2
1	4	3	0.8
2	8	3	4.8
3	1	5	-2.2
4	1	8	-2.2

Now that we have the calculations needed for the first step. Let's keep going.

$$y_i - \overline{y}$$

The second step involves doing the same thing for the y column.

```
df['step2'] = df.y - df.y.mean()
```

That's easy enough, what's next?

$$(x_i - \overline{x})(y_i - \overline{y})$$

The formula tells us that we need to take all the values we gathered in step 1 and multiply them by the values in step 2. We will store this in a new column labeled step3.

```
df['step3'] = df.step1 * df.step2
```

This is what the DataFrame looks like at this point:

We can now move on to the last operation in this part of the formula.

```
df['step3'] = df.step1 * df.step2
```

$$\sum_{df}(x_i - \overline{x})(y_i - \overline{y})$$

This means we need to add up all the values from the previous step.

```
step4 = df.step3.sum()
```

Great, we have summed up the values and have stored it in a variable called step4. We will come back to this later. For now, we can start on the second part of the formula.

$$\sqrt{\sum(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}$$

We have already found in the following in step1, so we can use that data. We will store this data in a new column labeled step5.

```
df['step5'] = df.step1 ** 2
```

The next part of the formula tells us to do the same thing for the y values.

$$(y_i - \overline{y})^2$$

We can take the values that we created in step 2 and square them.

```
df['step6'] = df.step2 ** 2
```

This is what our DataFrame looks like at this point:

```
df['step5'] = df.step1 ** 2
```

```
df['step6'] = df.step2 ** 2
```

```
df
```

	x	y	step1	step2	step3	step5	step6
0	2	1	-1.2	-3.0	3.6	1.44	9.0
1	4	3	0.8	-1.0	-0.8	0.64	1.0
2	8	3	4.8	-1.0	-4.8	23.04	1.0
3	1	5	-2.2	1.0	-2.2	4.84	1.0
4	1	8	-2.2	4.0	-8.8	4.84	16.0

Let's look at the next part of the formula:

$$\sum (x_i - \overline{x})^2 \sum (y_i - \overline{y})^2$$

This tells us that we have to take the sum of what we did in step 5 and multiply it with the sum of what we did in step 6.

```
step7 = df.step5.sum() * df.step6.sum()
```

Let's keep going, almost there!

$$\sqrt{\sum_{i=1}^{n} (x_i - \overline{x})^2 (y_i - \overline{y})^2}$$

The last portion of this part is to simply take the square root of the figure from our previous step. We can use the Numpy library to calculate the square root.

```
step8 = np.sqrt(step7)
```

Now that we've done that, all that is left is to take the answer from the first part of the formula and divide it by the answer in the second part.

```
step4/step8
```

And there you have it, we've manually calculated a correlation coefficient. To make sure that the calculation is correct, we can use the `corr()` function which is built into Pandas to calculate the coefficient.

```
df.x.corr(df.y)
```

Here is our final result. Your correlation coefficient will be different, but it should match the output from the Pandas calculation.

```
step4/step8
```

```
-0.4164613615708485
```

```
df.x.corr(df.y)
```

```
-0.4164613615708485
```

How to calculate the correlation coefficients for a watchlist?

Calculating a correlation coefficient in Python is quite simple as there are several libraries that can do the heavy lifting for you. In this guide, we will be using python and the libraries from this GitHub repository.

Step one – Gathering and cleaning up historical data

We are using the Alpha Vantage library in this step.

```
import pandas as pd from alpha_vantage.timeseries import
TimeSeries
```

Our first step is to import the Pandas library as we will be using it to store our data and calculate the correlation coefficient. We've also imported the Timeseries class from the alpha_vantage library, which will retrieve historical data.

We have exported our watchlist to a CSV file so in the next step we will import it and convert it to a list format. There are several ways to read a CSV file in Python but since we are already using Pandas, we might as well use it here rather than importing another library just for this step.

If you don't have your watchlist in CSV format, you can just as easily create a Python list that includes the tickers within your watchlist.

```
#grab tickers from csv file watchlist_df =
pd.read_csv('watchlist.csv', header=None) watchlist =
watchlist_df.iloc[0].tolist()
```

We now have a Python list of the five stock tickers we will use in this example. Our next step is to iterate through the watchlist and download historical data.

```
#instantiate TimeSeries class from alpha_vantage library app =
TimeSeries(output_format='pandas')
```

First, we instantiate the Timeseries class from the alpha_vantage library. We've passed through a parameter here so that the output will be a Pandas dataframe. This will save a lot of time having to format the data.

```
#itter through watchlist and retrieve daily price data stocks_df
= pd.DataFrame() for ticker in watchlist: alphav_df =
app.get_daily_adjusted(ticker) alphav_df = alphav_df[0]
alphav_df.columns = [i.split(' ')[1] for i in alphav_df.columns]
stocks_df[ticker] = alphav_df['adjusted'].pct_change()
```

Next, we iterate through our Python list of stock tickers and call the Alpha Vantage API for each ticker's data. But before doing that, we'll create an empty Pandas dataframe that we can append data to.

What we've done is taken the 'adjusted' column, which is the adjusted daily close, and appended it to our `stocks_df` dataframe. Note the additional `pct_change()` function. This will normalize our data by converting the price data to a percentage return. This is what our dataframe looks like at this point.

date	AAPL	MSFT	GLD	XOM	NFLX
2020-05-15	-0.023019	-0.009464	0.007622	-0.073666	0.003557
2020-05-14	0.005947	-0.014359	-0.005612	0.007143	-0.026949
2020-05-13	-0.006106	-0.004321	-0.008772	-0.008747	-0.008327
2020-05-12	0.012222	0.015355	-0.009531	0.052230	-0.014717
2020-05-11	0.011560	0.023177	-0.003874	0.016670	0.020147

Now we have a nicely formated time-series dataframe in less than 20 lines of code!

Step Two – Calculating the correlation coefficient

Now that we have our data, we can easily check the correlation coefficient between any stocks within our dataframe. Here is how we check the correlation between AAPL and MSFT.

```
print(stocks_df.AAPL.corr(stocks_df.MSFT))
```

What we've done here is taken the column of adjusted closing prices for AAPL and compared it with the column for MSFT. To access a single column, we specify the name of the dataframe and column like so:

```
print(stocks_df.AAPL)
```

alternatively, we can also access it like this:

```
```

```
print(stocks_df['AAPL'])
```

When dealing with a single column we are no longer working with a dataframe. Rather, we are working with a Pandas series. The basic syntax for calculating the correlation between different series is as follows:

```
Series.corr(other_series)
```

In our example, we found a correlation coefficient of 0.682 between AAPL and MSFT. Remember, the closer to 1, the higher the positive correlation. So in this example, there is a very strong correlation between these two stocks.

Let's take a look at the correlation between Apple and Netflix:

```
print(stocks_df.AAPL.corr(stocks_df.NFLX)
```

The correlation coefficient is -0.152. It's quite close to zero, which indicates that there was no correlation between these two stocks. At least during that time period.

There are three main methods used in calculating the correlation coefficient: Pearson, Spearman, and Kendall. We will discuss these methods in a bit more detail later on in the guide.

By default, Pandas will use the Pearson method. You can pass through different methods as parameters if you desire to do so. Here is an example of a calculation using the Spearman method:

```
print(stocks_df.AAPL.corr(stocks_df.NFLX, method='spearman')
```

And this is how you would get the correlation coefficient using the Kendall method:

```
print(stocks_df.AAPL.corr(stocks_df.NFLX, method='kendall'))
```

Correlation of returns versus prices

We calculated the percentage return between each price point in our dataset and ran our correlation function on that rather than calculating it on the raw data itself. We do this to get a more accurate correlation coefficient.

The reasoning behind it is that it standardizes the data which is beneficial no matter which calculation method you use.

If you're using the Spearman or Kendall method, which utilizes a ranking system, returns data will remove some of the extremes from your dataset, which can otherwise influence the entire ranking system.

The Pearson method doesn't use a ranking system but heavily relies on the mean of your data set. Using returns data narrows the range of your dataset, which in turn puts more emphasis on deviations from the mean, resulting in higher accuracy.

```
values_x = [10, 11, 13, 16, 17, 4, 5, 6] values_y = [10, 11, 13, 16, 17, 18, 19, 20]
```

Take a look at the above two datasets as an example.

Notice how they both have almost the same data? The difference is that `values_x` dropped off sharply in the third last value from 17 to 4. However, it continued to rise by one in the last two values, the same way `values_y` did.

This type of behavior can often happen in the markets. For example, a stock might have reported earnings, which caused a sharp but temporary drop in its price. But aside from the momentary drop, the overall fluctuations in the stock price have not changed much at all compared to other correlated stocks.

The ranking systems used in correlation calculations, however, will view the momentary decline differently. It will assign an arbitrarily low value to the last three values in values_x since they are the lowest in the dataset. At the same time, it will rank the last three values in values_y as the largest.

This creates a major discrepancy that will ultimately cause our correlation coefficient to be much lower than it should be.

In a non-ranking system such as the Pearson method, the last three values will drag down the mean value for the entire dataset.

If we take the returns instead, we are comparing how much one value fluctuated relative to the value before it.

In that case, there would have been a major decline when the values in values_x dropped from 17 to 4, but the divergence in correlation stops there

as both the data sets rose in value in the last two places.

How to create a time-series dataset in Pandas?

A time-series is simply a dataset that follows regular, timed intervals. The previous example, where we had data for five stocks, is a good example of a time-series dataset.

Further, Pandas intuitively lined up price data when we merged all five stocks into one dataframe based on the date column, which all of our data had in common. This column then acts as an index for our data.

We can just as easily create a dataframe with a time-series index from scratch. The next example will show how to do that with data we have saved in a CSV file.

```
import pandas as pd TSLA_df = pd.read_csv('TSLA.CSV')
print(TSLA_df)
```

Here we've imported price data for TSLA based on 15-minute intervals. In other words, 15-minute bars for TSLA.

	date	open	high	low	close	volume
0	2020-04-24 16:00:00	725.2200	727.5700	723.8000	725.0400	573985.0
1	2020-04-24 15:45:00	725.6745	730.7300	724.5801	725.1801	648799.0
2	2020-04-24 15:30:00	723.5400	725.8116	722.1700	725.8116	287570.0
3	2020-04-24 15:15:00	722.1550	727.0000	720.6100	723.4800	438345.0
4	2020-04-24 15:00:00	725.9800	727.5700	721.2500	722.1100	420652.0

Notice that Pandas has created a generic index rather than using the date column. We can change that with two methods. Either we manually set the index like this:

```
TSLA_df.set_index('date', inplace=True)
```

Or, we can pass in a parameter in the prior function where we imported the data.

```
TSLA_df = pd.read_csv('TSLA.CSV', index_col=0)
```

This will tell Pandas to use the first column in the CSV data as the index which in price data will typically be your date or time data.

date	open	high	low	close	volume
2020-04-24 16:00:00	725.2200	727.5700	723.8000	725.0400	573985.0
2020-04-24 15:45:00	725.6745	730.7300	724.5801	725.1801	648799.0
2020-04-24 15:30:00	723.5400	725.8116	722.1700	725.8116	287570.0
2020-04-24 15:15:00	722.1550	727.0000	720.6100	723.4800	438345.0
2020-04-24 15:00:00	725.9800	727.5700	721.2500	722.1100	420652.0

Next we will check the data type for our newly-created index.

```
print(TSLA_df.index[:4])
```

```
Index(['2020-04-24 16:00:00', '2020-04-24 15:45:00', '2020-04-24 15:30:00',
       '2020-04-24 15:15:00'],
      dtype='object', name='date')
```

As you can see, the dtype shows the index as an object. We can convert it to a DateTime like so:

```
TSLA_df.index = pd.to_datetime(TSLA_df.index)
```

If we check the index again, we will now see the dtype as 'datetime64[ns]' which is what we are after.

```
DatetimeIndex(['2020-04-24 16:00:00', '2020-04-24 15:45:00',
               '2020-04-24 15:30:00', '2020-04-24 15:15:00'],
              dtype='datetime64[ns]', name='date', freq=None)
```

When importing a CSV file, we can pass through `parse_dates=True` into the `pd.read_csv()` function to automatically parse the dates as a DateTime object.

```
TSLA_df = pd.read_csv('TSLA.CSV', index_col=0, parse_dates=True)
```

We did it manually in this example just to illustrate how it can be done in the event you are creating a dataframe using other methods than from a CSV.

What is a correlation matrix?

The previous examples have shown how to calculate a correlation coefficient for two stocks. But if we have a dataframe full of stocks? Surely there has to be an easier way to get the coefficient for everything in the dataframe?

That's where the correlation matrix comes in. It is a table or a matrix that will display the correlation coefficient for everything in the dataframe. To create this, simply type your dataframe name, followed by `.corr()`. Or in our example, `stocks_df.corr()`.

	AAPL	MSFT	GLD	XOM	NFLX
AAPL	1.000000	0.681712	-0.533378	0.762428	-0.151615
MSFT	0.681712	1.000000	-0.331221	0.445689	0.452704
GLD	-0.533378	-0.331221	1.000000	-0.348389	-0.054419
XOM	0.762428	0.445689	-0.348389	1.000000	-0.343662
NFLX	-0.151615	0.452704	-0.054419	-0.343662	1.000000

Here we have our correlation matrix. The first column in the first row is the correlation between AAPL and AAPL, which obviously will have the highest correlation when comparing data with itself.

Looking at this matrix, we can easily see that the correlation between Apple (AAPL) and Exxon Mobile (XOM) is the strongest while the correlation between Netflix (NFLX) and AAPL is the weakest.

Further, there is fairly notable negative correlation between AAPL and GLD which is an ETF that tracks gold prices.

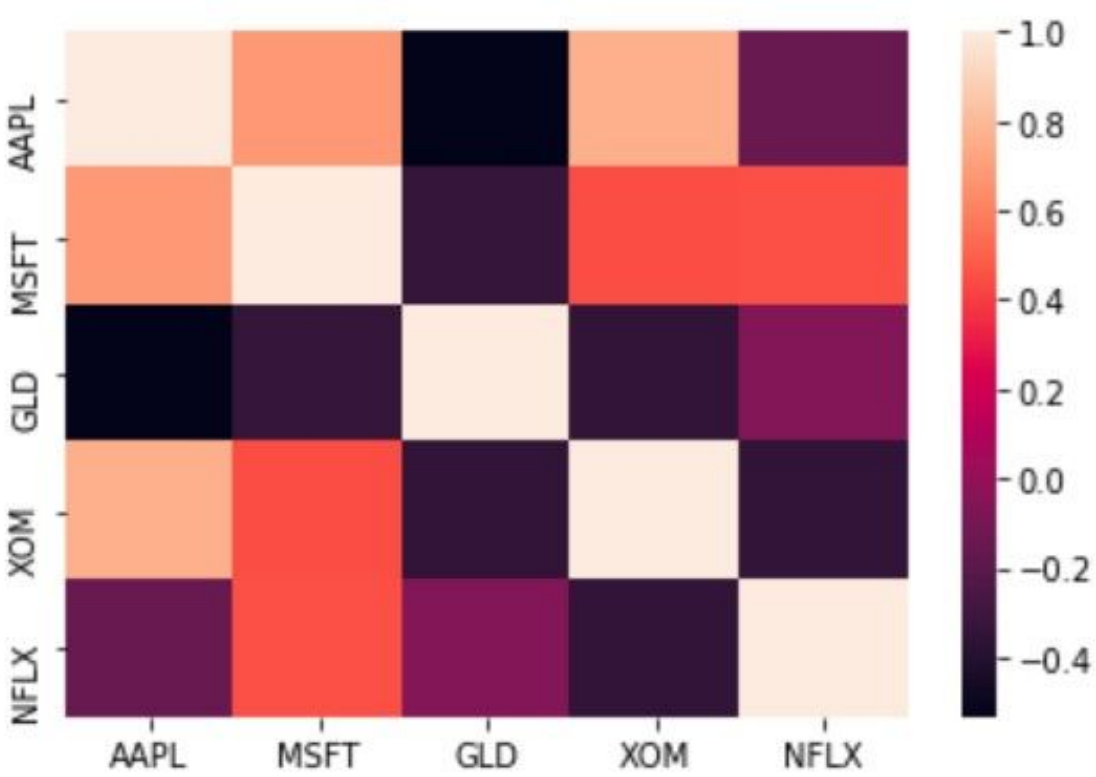
We can also create a heatmap. This will allow us to visualize the correlation between the different stocks.

To do this, we will use the Seaborn library, which is a great tool for plotting and charting. It is built on top of the popular matplotlib library and does all the

heavy lifting involved in creating a plot.

```
import seaborn as sns import matplotlib.pyplot as plt ax =
sns.heatmap(stocks_df.corr()) plt.show()
```

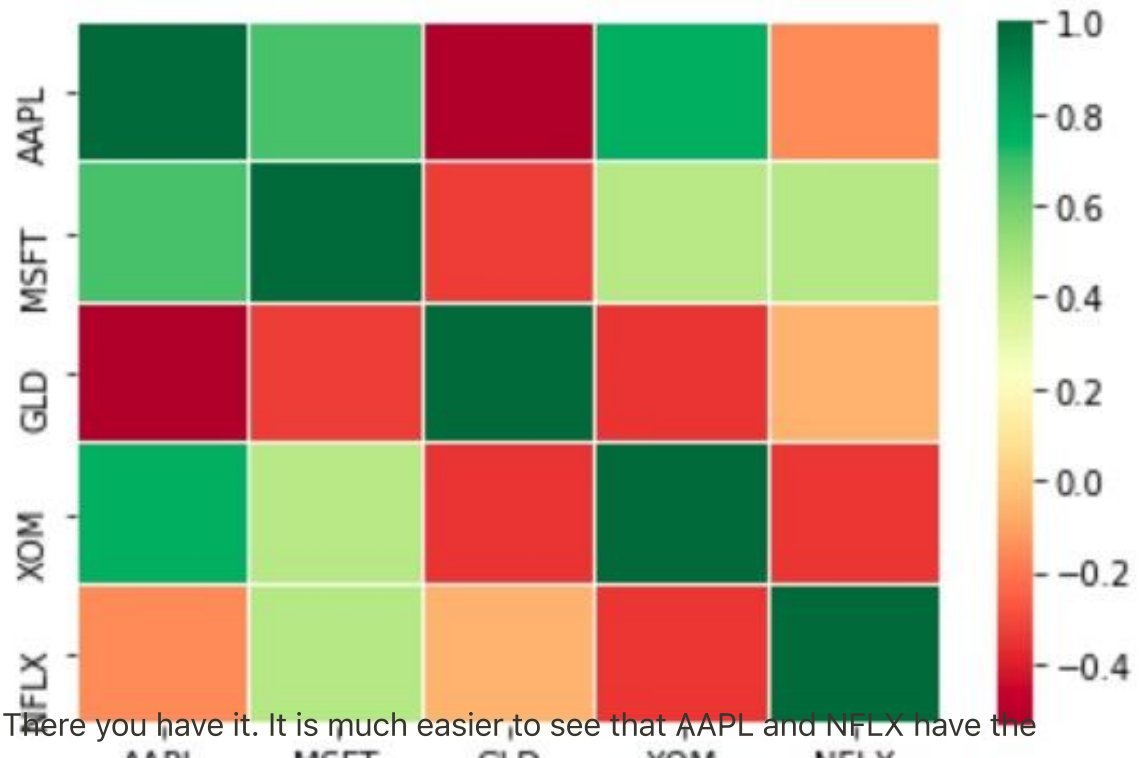
Here we've imported the library and called the heatmap function to display the heatmap. At this stage, we've only passed through the correlation matrix dataframe.



We can now assess the strength in correlation based on color, and there is a useful guide on the right-hand side. But since we are used to seeing things in red and green in the finance world, let's customize it a bit.

```
ax = sns.heatmap(stocks_df.corr(), cmap='RdYlGn', linewidths=.1)
plt.show()
```

The above code snippet sets the Red, Yellow, Green values to cmap which defines our colors. We have also passed through a line width of .1 to create a bit of space between the boxes just to improve the visual aesthetics.



There you have it. It is much easier to see that AAPL and NFLX have the weakest correlation. We can also easily see that GLD has a negative correlation with all of the other assets.

How to use a correlation matrix in practice?

You can use a correlation matrix to filter out stocks for various reasons quickly. Maybe you're already in a trade, and you don't want to trade other instruments with a strong correlation. Another reason might be to check other strongly correlated instruments to ensure your analysis produces a similar signal.

As an example, say you've already taken a long position in AAPL. Now your automated trading algo is sending you a signal to buy MSFT. This is very likely to happen since we've already determined that the two have a strong correlation with each other.

In this case, you might want to skip that trade because it is only increasing your risk exposure. In other words, when the correlation is that high, it's not all that different from just doubling up your exposure in AAPL, and that is something to avoid.

In the same way, we can also confirm if our signal is strong enough to act on. For example, let's say we are trading a breakout strategy, and we buy a stock when it exceeds more than one standard deviation from its average.

We get a signal to buy NFLX. We can see what stock is most closely correlated with NFLX to determine if it has also exceeded one standard deviation from its

average. We can use the `idxmax()` function from Pandas to figure out the

```
nflx_corr_df = stocks_df.corr().NFLX
print(nflx_corr_df.idxmax())
```

But wait, we already know that the highest correlation is going to be with NFLX itself, it produces a correlation of 1. So we want to filter for correlations less than 1.

```
print(nflx_corr_df[ nflx_corr_df < 1 ].idxmax())
```

The above code returns 'MSFT'. Now we can check where Microsoft is trading relative to its standard deviation. If it is trading below it, we can even wait until it exceeds it to give us a stronger signal on our original NFLX buy signal.

In the same manner, we can easily check for inverse correlations with NFLX as follows

```
print(nflx_corr_df.idxmin())
```

This returned 'XOM'. If our analysts gives us a bearish signal for XOM it would once again provide more conviction on our bullish NFLX trade.

As we move along, we will introduce the concepts of error ratio and cointegration.

Now, we are grabbing the list of S&P 500 companies. Wikipedia keeps an up to date list and we can use Pandas to scrape it.

```
import pandas as pd from alpha_vantage.timeseries import TimeSeries fr
time import sleep stock_list =
pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companie
```

We now have a DataFrame of all the companies in the S&P 500.

	Symbol	Security	SEC filings	GICS Sector	GICS Sub Industry	Headquarters Location	Date first added	CIK	Founded
0	MMM	3M Company	reports	Industrials	Industrial Conglomerates	St. Paul, Minnesota	1976-08-09	66740	1902
1	ABT	Abbott Laboratories	reports	Health Care	Health Care Equipment	North Chicago, Illinois	1964-03-31	1800	1888
2	ABBV	AbbVie Inc.	reports	Health Care	Pharmaceuticals	North Chicago, Illinois	2012-12-31	1551152	2013 (1888)
3	ABMD	ABIOMED Inc	reports	Health Care	Health Care Equipment	Danvers, Massachusetts	2018-05-31	815094	1981

The list can be filtered for bank stocks only as that is the sector we will focus on.

To find these stocks we will filter the GICS Sub Industry column for rows that contain the term **Diversified Banks**.

```
banks = stock_list[stock_list['GICS Sub Industry'].str.contains('Diversified Banks')]
```

Great, we now have a DataFrame of only bank stocks.

	Symbol	Security	SEC filings	GICS Sector	GICS Sub Industry	Headquarters Location	Date first added	CIK	Founded
62	BAC	Bank of America Corp	reports	Financials	Diversified Banks	Charlotte, North Carolina	1976-06-30	70858	1998 (1923 / 1874)
108	C	Citigroup Inc.	reports	Financials	Diversified Banks	New York, New York	1988-05-31	831001	1998
118	CMA	Comerica Inc.	reports	Financials	Diversified Banks	Dallas, Texas	1995-12-01	28412	1849
267	JPM	JPMorgan Chase & Co.	reports	Financials	Diversified Banks	New York, New York	1975-06-30	19617	2000 (1799 / 1871)
455	USB	U.S. Bancorp	reports	Financials	Diversified Banks	Minneapolis, Minnesota	NaN	36104	1968
485	WFC	Wells Fargo	reports	Financials	Diversified Banks	San Francisco, California	1976-06-30	72971	1852

We don't need all that information, the only thing we need are the ticker symbols. So let's save all the tickers to a list.

```
tickers = banks.Symbol.to_list
```

With our ticker symbols saved in a list, we can iterate through the list and query the Alpha Vantage API for daily price data.

```
stocks_df = pd.DataFrame() for ticker in tickers: alphav_df =
app.get_daily_adjusted(ticker) alphav_df = alphav_df[0]
alphav_df.columns = [i.split(' ')[1] for i in alphav_df.columns]
stocks_df[ticker] = alphav_df['adjusted'].pct_change() sleep(12)
```

We will save data for all of the tickers into a DataFrame. Note that we are converting the data to show the daily gain or loss in percentage format using the `pct_change()` function from Pandas.

This will provide better accuracy in our next step where we calculate the correlation coefficient.

This is what our stocks DataFrame looks like at this point.

date	BAC	C	CMA	JPM	USB	WFC
2020-09-22	0.029235	0.034409	0.019484	0.016498	0.031594	0.035918
2020-09-21	0.022139	0.014784	0.040646	0.011032	0.016016	0.016490
2020-09-18	0.030241	0.021170	0.051474	0.031896	0.040376	0.045341
2020-09-17	0.005553	0.014935	0.002952	0.002135	0.007177	-0.000796
2020-09-16	0.009862	0.010762	0.010056	0.011567	0.007126	0.023895

We can create a correlation matrix from the data

```
stocks_df.corr()
```

This is what it looks like.

	BAC	C	CMA	JPM	USB	WFC
BAC	1.000000	0.920478	0.927203	0.943053	0.937353	0.900835
C	0.920478	1.000000	0.854148	0.894971	0.870065	0.860617
CMA	0.927203	0.854148	1.000000	0.904612	0.938394	0.911366
JPM	0.943053	0.894971	0.904612	1.000000	0.935053	0.894732
USB	0.937353	0.870065	0.938394	0.935053	1.000000	0.920740
WFC	0.900835	0.860617	0.911366	0.894732	0.920740	1.000000

From here, we can try and pick out a pair with a strong correlation to see if the relationship is appropriate for pairs trading.

In this case, the matrix is small enough for us to eyeball the best pair. But let's automate this part too, in case we have to deal with a much larger correlation matrix down the road.

```
cor = stocks_df.corr()
```

In the code above, we start by saving the correlation table to a variable. We then want the largest number in the correlation matrix.

We first want to remove any perfect correlations which are denoted by 1. This is simply the coefficient of the stock to itself, not very useful. We can change any instances of 1 to 0.

```
cor[cor==1] = 0
```

Now we can determine which coefficient is the largest. The idxmax function will return the largest, except it will do so for each column. If we stack the DataFrame first, it will return only the pair with the strongest correlation.

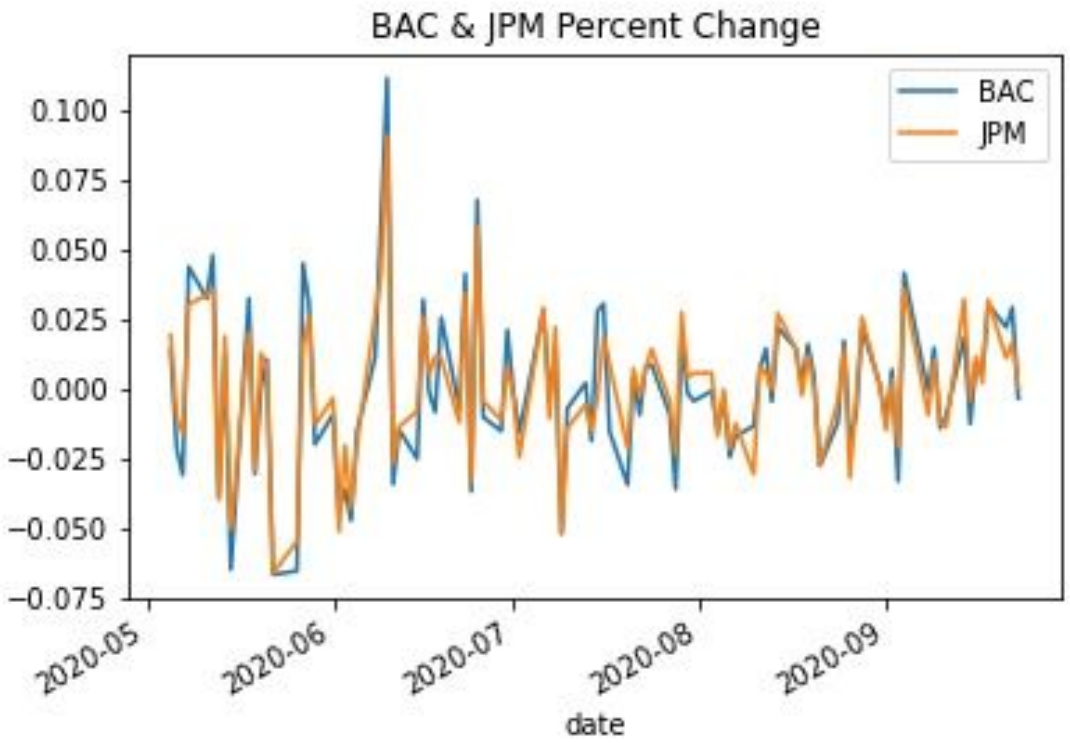
```
cor.stack().idxmax()
```

In this case, the code returned a tuple containing BAC and JPM.

Now that we know that BAC and JPM have the strongest correlation in our table, let's quickly plot a chart to visually confirm it.

```
%matplotlib inline bac_jpm = pd.concat([stocks_df.BAC,
stocks_df.JPM], axis=1) bac_jpm.plot()
```

We've created another DataFrame with only the BAC and JPM data. We then used the plot() function from Pandas to display a chart in our Jupyter notebook.



Visually, it looks like a good correlation, just as our correlation coefficient had suggested.

But a good correlation alone does not imply that the two bank stocks are cointegrated.

Now, a concept called ***cointegration*** is going to be introduced.

Correlation and cointegration are two very different things. We are only using correlation to try and narrow down our focus.

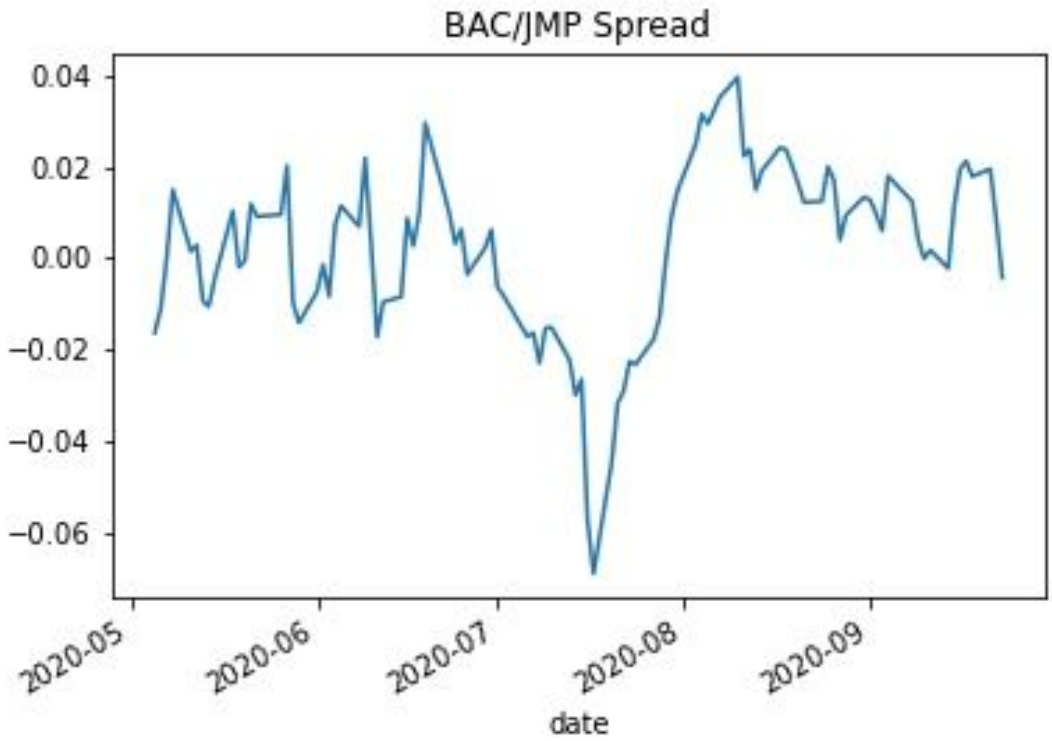
There could be a scenario where one stock consistently outperforms another. In that case, the two stocks will still have a fairly high correlation, but the synthetic pair will not be stationary or cointegrated.

Let's dig a little deeper to try and find out if this pair is suitable for a cointegration strategy.

The next step is to calculate the spread between BAC and JPM and plot it. After all, that is what we will be trading. If the two stocks are cointegrated, the chart should have some consistency in the variance from its mean.

```
spread = stocks_df.BAC - stocks_df.JPM spread.cumsum().plot()
```

To calculate the spread, we need to subtract one stock from another. This is because we are using percent change data. If we were using closing prices, we would have divided one by the other to derive the spread.



The chart above does not show consistency in variance.

There is enough evidence here that the spread is mean-reverting between BAC and JPM. There was a larger than usual dip in July, but overall, the spread seems to revert to the 0 level continuously. This could be a good candidate for a cointegration strategy.

This should have given an intuitive idea of cointegration. Now, we are going to look at rigorous statistical methods that are employed to find the cointegration between pairs.

▼ ***Statistical methods to find cointegration***

There are several methods to test for cointegration, we will focus on explaining three that are commonly used.

The first method we will explain is called the Augmented Dickey-Fuller test or, in short, ADF test. Here's the explanation of the ADF test from Wikipedia.

Augmented Dickey-Fuller test

In statistics and econometrics, an **augmented Dickey–Fuller test** (**ADF**) tests the null hypothesis that a unit root is present in a time series sample. The alternative hypothesis is different depending on which version of the test is used, but is usually stationarity or trend-stationarity. It is an augmented version of the Dickey–Fuller test for a larger and more complicated set of time series models.

The augmented Dickey–Fuller (ADF) statistic, used in the test, is a negative number. The more negative it is, the stronger the rejection of the hypothesis that there is a unit root at some level of confidence.

Testing procedure

The testing procedure for the ADF test is the same as for the Dickey–Fuller test but it is applied to the model.

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \cdots + \delta_{p-1} \Delta y_{t-p+1} + \varepsilon_t$$

where α is a constant, β the coefficient on a time trend and p the lag order of the autoregressive process. Imposing the constraints $\alpha = 0$ and $\beta = 0$ corresponds to modeling a random walk and using the constraint $\beta = 0$ corresponds to modeling a random walk with a drift. Consequently, there are three main versions of the test, analogous to the ones discussed on Dickey–Fuller test (see that page for a discussion on dealing with uncertainty about including the intercept and deterministic time trend terms in the test equation.)

By including lags of the order p the ADF formulation allows for higher-order autoregressive processes. This means that the lag length p has to be determined when applying the test. One possible approach is to test down from high orders and examine the t-values on coefficients. An alternative approach is to examine information criteria such as the Akaike information

criterion, Bayesian information criterion, or the Hannan–Quinn information criterion.
The unit root test is then carried out under the null hypothesis $\gamma = 0$ against the alternative hypothesis of $\gamma < 0$. Once a value for the test statistic

$$\mathrm{DF}_\tau = \frac{\hat{\gamma}}{\mathrm{SE}(\hat{\gamma})}$$

is computed it can be compared to the relevant critical value for the Dickey-Fuller test. As this test is asymmetrical, we are only concerned with the negative values of our test statistic DF_τ. If the calculated test statistic is less (more negative) than the critical value, then the null hypothesis of $\gamma = 0$ is rejected and no unit root is present.

Intuition

The intuition behind the test is that if the series is characterized by a unit root process then the lagged level of the series (y_{t-1}) will provide no relevant information in predicting the change in y_t besides the one obtained in the lagged changes (Δy_{t-k}). In this case, the $\gamma = 0$ and the null hypothesis is not rejected. In contrast, when the process has no unit root, it is stationary and hence exhibits reversion to the mean - so the lagged level will provide relevant information in predicting the change of the series and the null of a unit root will be rejected.

More rigorous mathematical analysis

The parts above are directly quoted from the Wikipedia article of ADF test.

However, a more intensive mathematical analysis of Dickey and Fuller tests, as well as the augmented one, can be found in this linked article, pages 9-12.

Now, let's have look at another statistical method for finding cointegration known as Engle–Granger two-step method.

Engle–Granger two-step method

If x_t and y_t are non-stationary and Order of integration d=1, then a linear combination of them must be stationary for some value of β and u_t. In other

words:

$$y_t - \beta x_t = u_t$$

where u_t is stationary.

If we knew β , we could just test it for stationarity with something like a Dickey–Fuller test and be done. But because we don't know β , we must estimate this first, generally by using ordinary least squares[clarification needed], and then run our stationarity test on the estimated u_t series, often denoted $\hat{u}_t$.

A second regression is then run on the first differenced variables from the first regression, and the lagged residuals $\hat{u}_{t-1}$ is included as a regressor.

More rigorous mathematical analysis

The parts above are directly quoted from the Wikipedia article of ADF test.

However, a more intensive mathematical analysis of The Engle and Granger approach can be read from the linked article.

The code below will use the statsmodel python library.

The first statistical method we explained was the ADF test. We can use the `adfuller` method from the statsmodels library in order to run this.

The second method we explained is the Engle-Granger two-step method. We can use the `coint` method from statsmodels library in order to run this.

We will start by importing both methods.

```
from statsmodels.tsa.stattools import coint, adfuller
```

We will run our spread Dataframe through the adfuller method. This will let us know if our data is stationary or not.

Recall that if two assets combined produce a stationary time series, then those two assets are considered to be cointegrated.

But before running the test, we need to clean up our data. The statsmodels library currently does not handle missing data.

We can use the dropna function from Pandas to delete any empty rows.

```
spread.dropna(inplace=True)
```

Next, we will save the results from the adfuller method to a variable called adf_results.

```
adf_results = adfuller(spread)
```

The output might seem a bit confusing, so the next part of the code cleans it up a bit so it is easier to read.

```
print(f'ADF Test Statistic: {adf_results[0]}') print(f'PValue:
{adf_results[1]}') print(f'Number of lags used: {adf_results[2]}')
print(f'Number of observations: {adf_results[3]}') print('Critical
Values:') for k,v in adf_results[4].items(): print(f'{k}: {v}')
```

And here are our results.

```
ADF Test Statistic: -9.925655849625182
PValue: 2.9117504611317605e-17
Number of lags used: 0
Number of observations: 98
Critical Values:
1%: -3.498909760601449 6
5%: -2.891516256916761
10%: -2.582760441482715 7
```

The first thing we will look at is the P-Value. It should be below a certain threshold. Some discretion can be used to determine exactly what that threshold is.

Commonly used thresholds are either 5% or 1% (0.05 or 0.01). In this case, the P-Value falls well below the threshold.

Next, we will look at the test statistic which is roughly -9.9. We can compare this to the Critical Values provided by the test.

Since our test statistic is less than the 1% critical value of -3.5, we can assume that our data is stationary with 99% certainty.

In other words, this test tells us that we have found a cointegrated pair based on the data we have supplied.

The next test we will run is the `coint` test from the statsmodels library.

In the last test, we used the spread DataFrame that we created by subtracting percentage change data from one stock to another.

This test requires the percentage change data from both stocks. We have this available already in our `stocks_df` DataFrame.

Once again, we will start by removing any empty rows. Then we will save the results from the test to a variable.

```
stocks_df.dropna(inplace=True) coint_results = coint(stocks_df.BAC,
stocks_df.JPM)
```

We can then output the results to the screen.

```
print(f'T-statistic of unit-root test on residuals
{coint_results[0]}') print(f'PValue: {coint_results[1]}')
print(f'Critical Values: {coint_results[2]}')
```

In this test, we are only focused on the P-Value. Just like the Augmented Dickey-Fuller test, if the value is below a certain threshold, we can consider the data to be cointegrated.

```
T-statistic of unit-root test on residuals -10.070276026622652
PValue: 1.6507830421961935e-16
Critical Values: [-4.01168502 -3.39918839 -3.08801077]
```

The P-Value, in this case, is well below 0.01 which lets us know that our two stocks are cointegrated.

We now have both visual confirmation from our graphs and statistical confirmation that this strategy is suitable for a cointegration strategy.

However, there is one other important consideration. Recall that the ADF test showed us the number of observations was 98.

This means we tested 98 trading days worth of data. This is a rather small sample.

It may very well be that the pair shows cointegration during some periods and not others. We can run further tests on a larger sample to gain more insight.

We can also customize our data to an extent.

For example, let's say it is determined that this spread moves well beyond the norm during earnings reports, and it is not profitable to trade during that time. So we can avoid these periods of an anomaly.

Here is a summary of the things we have covered so far in Trading Systems

1. We looked at the overviews of different kind of Quantitative trading strategies that include:
    - Alternative Data
    - Arbitrage
    - Pairs Trading
    - High Frequency Trading
2. Understood the idea behind Pair Trading.
3. Looked at advantages as well as disadvantages of Pairs Trading
4. Looked at statistical methods to find good pairs. We have also shown the different libraries that can be utilized to make a trading system. The statistical method we explained is for correlation and cointegration.
5. Looked at a practical example of finding correlation and testing for cointegration using code.

Now, using what we have discussed so far, we can successfully find pairs that can be traded. But, there is a vital part of the quantitative that is yet to be discussed. It is the back-testing of the strategy we have made.

Backtesting

What is backtesting and what's our goal?

Backtesting involves testing your strategy against historical data to see how it would have performed.

A backtest can reveal a lot about a strategy. It can provide insight into how profitable a strategy might be and point out areas where the strategy can be improved.

While it is natural to focus on profits, a backtest will show how a strategy performs during bad times and how it handles drawdowns.

It's normal for a strategy to go through periods where it doesn't make money. If it doesn't, it might even be a warning that something is not right.

A common pitfall in backtesting is making modifications to improve the PnL only to find the strategy underperforming once deployed live.

To avoid this, the historical data can be split into two parts. This is known as in-sample data and out of sample data.

Backtesting platforms

There are several platforms out there and each has its own strengths and weaknesses.

Like many aspects of this process, picking the right backtesting framework comes down to preference to a degree.

This is because the developers that have created backtesting frameworks all have different approaches to the market.

Here is a list of the more popular backtesters:

- Backtrader
- Zipline
- Pine Script (TradingView)
- MetaTrader 4
- MetaTrader 5
- Quantopian
- Quantconnect

Of these, the one I have looked into closely is Backtrader. Its API is vast, and I think it's better to leave the links to the articles that help understand back-testing with Backtrader rather than trying to make an excerpt of it.

List of articles I found to helpful and to the point:

1. Intro to Statistical Arbitrage in Crypto — Pairs Trading [Part8]

   This article explains statistical ideas along with code that shows backtesting of some crypto pair trading strategies

2. Trading Strategy: Back testing with Backtrader

   A very brief introductory article on backtesting with Backtrader

3. MultiData Strategy

   An article on the official website of Backtrader that talks about dealing with data from two different stocks. Essentially, it discusses the backtesting of a pair trading strategy with code snippets.