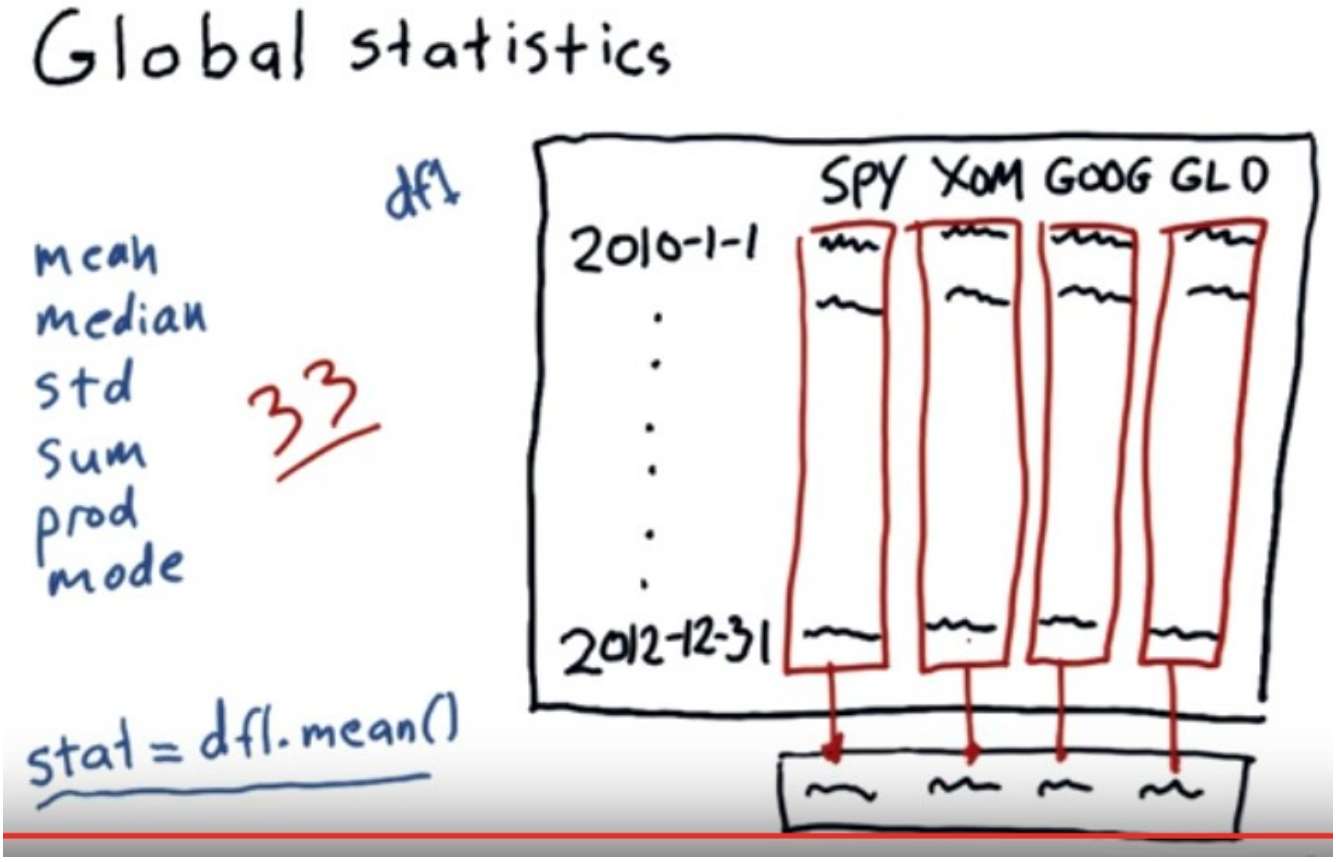
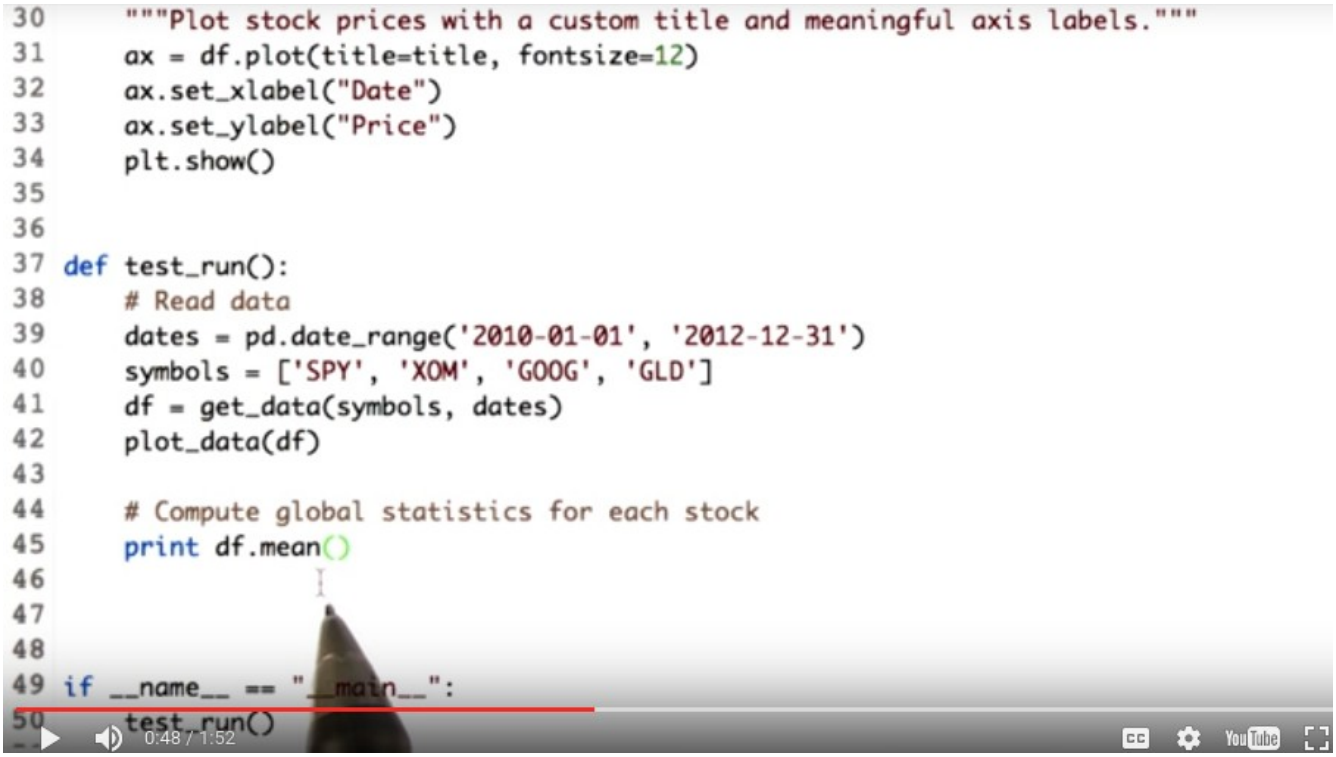
1. Are you ready, Dave? >> Ready for what, Professor? >> We're going to start some serious number crunching now. >> What do you mean? >> In this lesson, we're going to unleash the power of Python. We're going to show folks some tools that enable them to calculate all kinds of important statistics on time series data. >> What are we waiting for? >> Let's go.



2. In this lesson, we're going to take a look at the various kinds of statistics that we can take on time series data. Let's start first with global statistics. Consider our trusty data frame DF1 with columns for SPY, XOM, Google, and Gold. We can take the mean of each of these columns very simply with a statement like this. This statement will take the mean of each column and put it in the appropriate location of a new one-dimensional or row-wise of the array. Now because this is a data frame, and remember, a data frame augments NumPy and provides a lot more functionality. It's sort of in the array on steroids. Now we get lots and lots of functions we can access in this way. We already mentioned mean in addition to mean we've got median, standard deviation, sum, prod, mode. All together there's at least 33 global statistics you can compute in this way. And they're always adding more. Let me hand it over to Dave and she's going to show you how to do this in code.



3. Let's do some coding to get an idea of what professor just explained. Starting with defining our symbols list, having symbols like SPY, XOM, GOOG, and GLD. We then move ahead to build our dataframe df just like we did in couple of lessons before. So df is our final dataframe. Now let's start computing statistics. First we compute mean. We need mean of stock prices for each symbol. And dataframe.mean will do this for us. As professor explained, it computes mean for each column. And our columns denote one stock each. So we get mean for all stocks in just one line of code. So to compute the mean, we just called the name of the data frame df.mean.

```
SPY     121.711206
XOM      73.296221
GOOG    568.941941
GLD     142.603279
dtype: float64
```

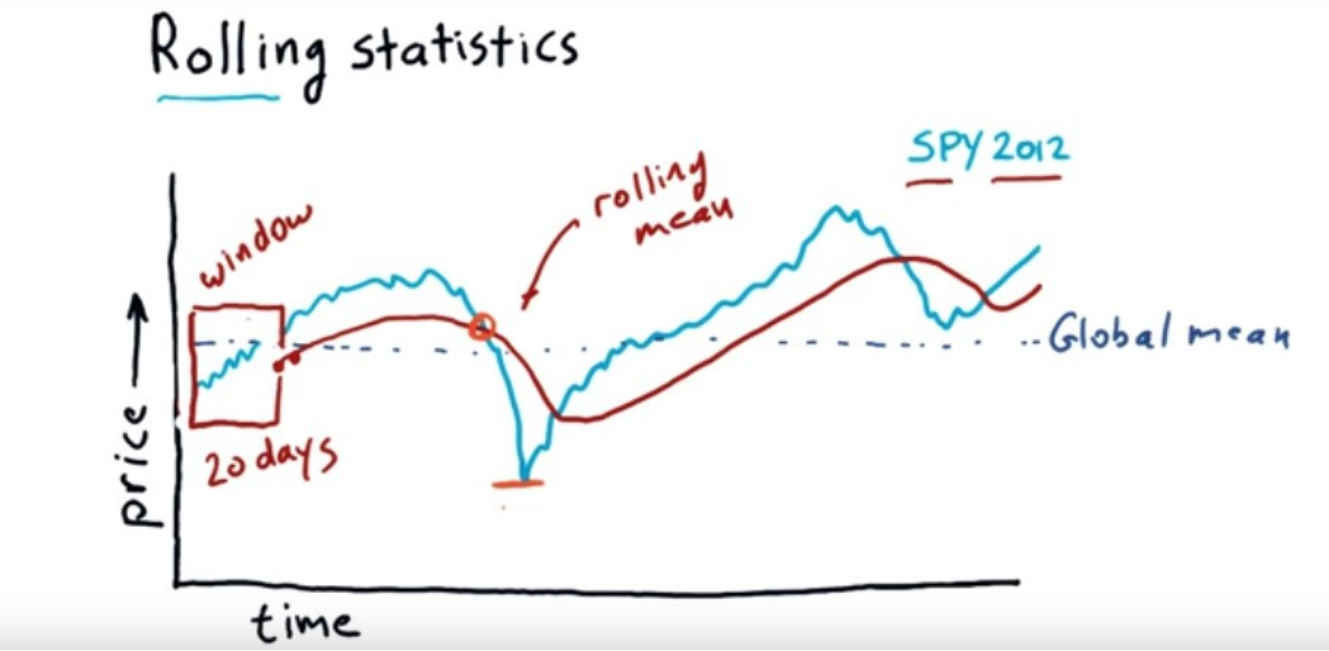
Let's check the output. Note how Pandas prints the mean for each symbol properly labeled. Also, here's the graph with all the symbols and their data.

```
44      # Compute global statistics for each stock
45      print df.median()
```

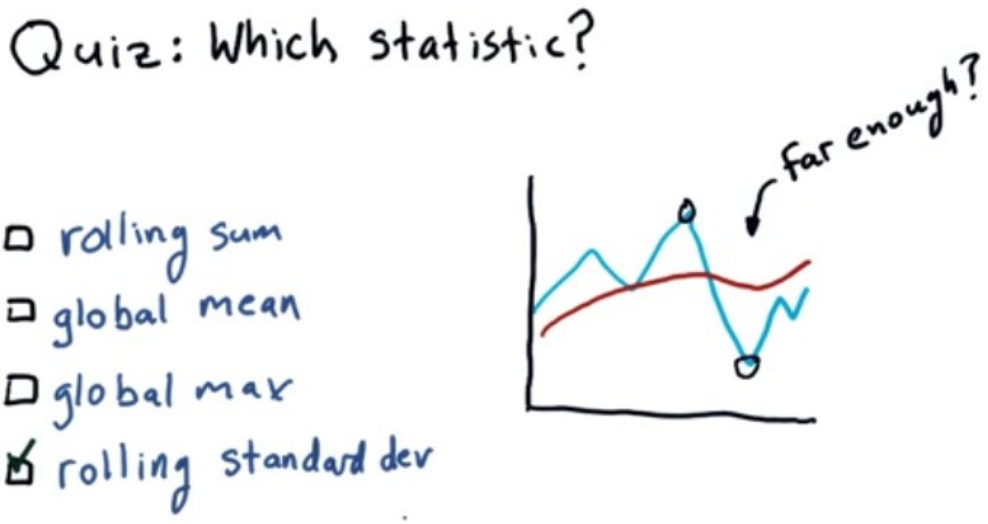
Similarly, we can compute median and standard deviation. Let's do it. We compute the median of the data frame by calling the median function. Remember the difference between the mean and the median. Mean is the average of a set of values that is the total sum divided by number of values. Whereas median refers to the value in the middle when they are all sorted.

```
44      # Compute global statistics for each stock
45      print df.std()
```

Now let's try standard deviation. We compute the standard deviation by calling the function std over the data frame. Let's check the output. Mathematically, standard deviation is the square root of variance. But more intuitively, it is a measure of deviation from central value. Here, the central value is the mean. A higher standard of deviation like here for Google indicates that the stock prices has varied a lot over time.

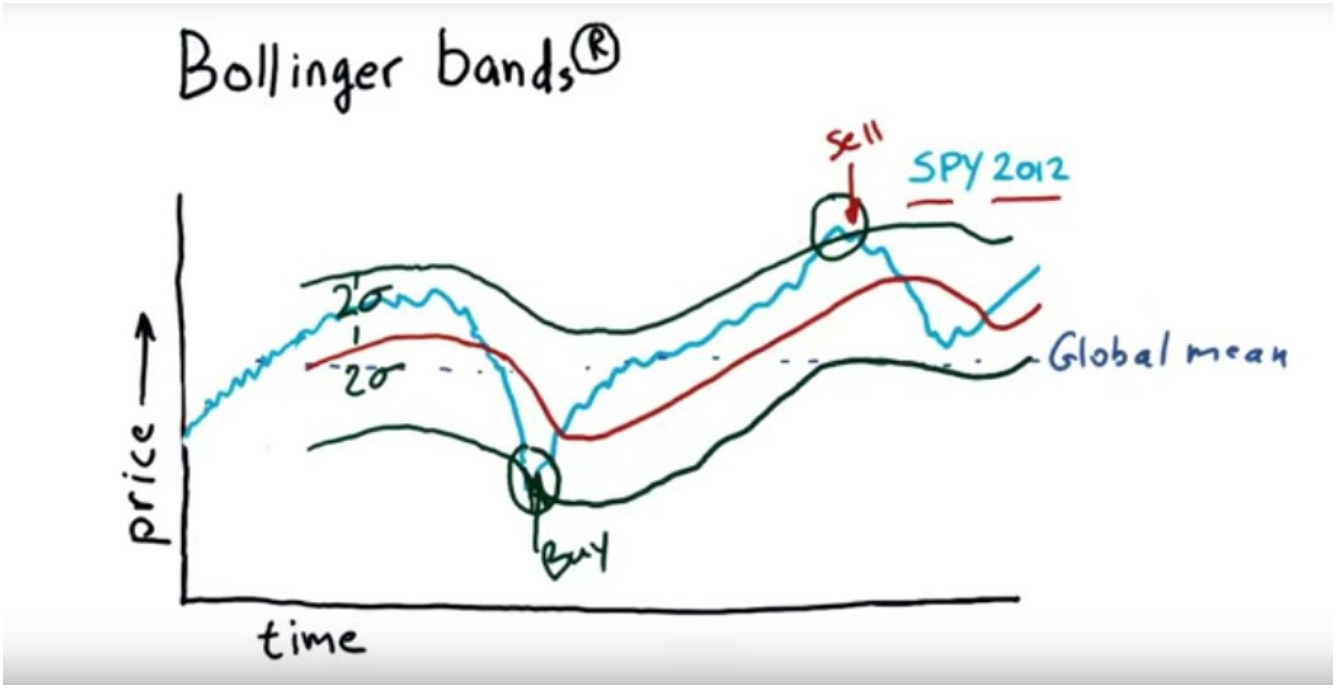


4. We're going to introduce a new kind of statistic now called rolling statistics, and as opposed to just taking the mean across the whole period of time we take sort of a snapshot over windows. I'll show you what that means in just a moment. Now on that last slide, we computed a global mean, which would be something about like this on this data. A rolling mean is a little bit different and here's how it works. Let's suppose we're going to take a 20 day rolling mean. We go, starting from here, 20 days, it's right about here. And then we take the mean of all that data behind us. We can draw a little box around that. This is called the window. In our case, it's 20 days. So we average all these values, and we get one mean, which is this point. We then move the window forward one day and we take another mean. Here's our next mean, which is a little bit higher. Now if we do that everyday over this entire year, so this is S and P 500 over the year 2012, we get something that looks about like this. You can see essentially, that it's a line that follows the day-to-day values of whatever it is we're tracking, but it lags a little bit. It's sort of a smoothed and lagged line. And this is called the rolling mean. We can compute statistics like this, just like the rolling mean. We could do standard deviation. We could do mode, median and so on. All of those statistics I showed you just a moment ago can also be used as rolling statistics. In the next mini course we are going to spend a lot of time talking about technical indicators, and this is actually one of them this rolling mean, it's called by technical analysts a simple moving average. And one thing they look at is places where the price crosses through the rolling average. So, in this case, the price is moving down through the 20 day mean. Now a hypothesis that I'm not saying I support, but a hypothesis that many who conduct technical analysis, is that this rolling mean may be a good representation of sort of the true underlying price of a stock, and that significant deviations from that, like this one (最下端紅色橫線那裡) here eventually result in a return to the mean. So if you can look for, say significant deviations like this one, you might find say a buying opportunity here. A challenge though, is to know when is that deviation significant enough that you should pay attention to it.



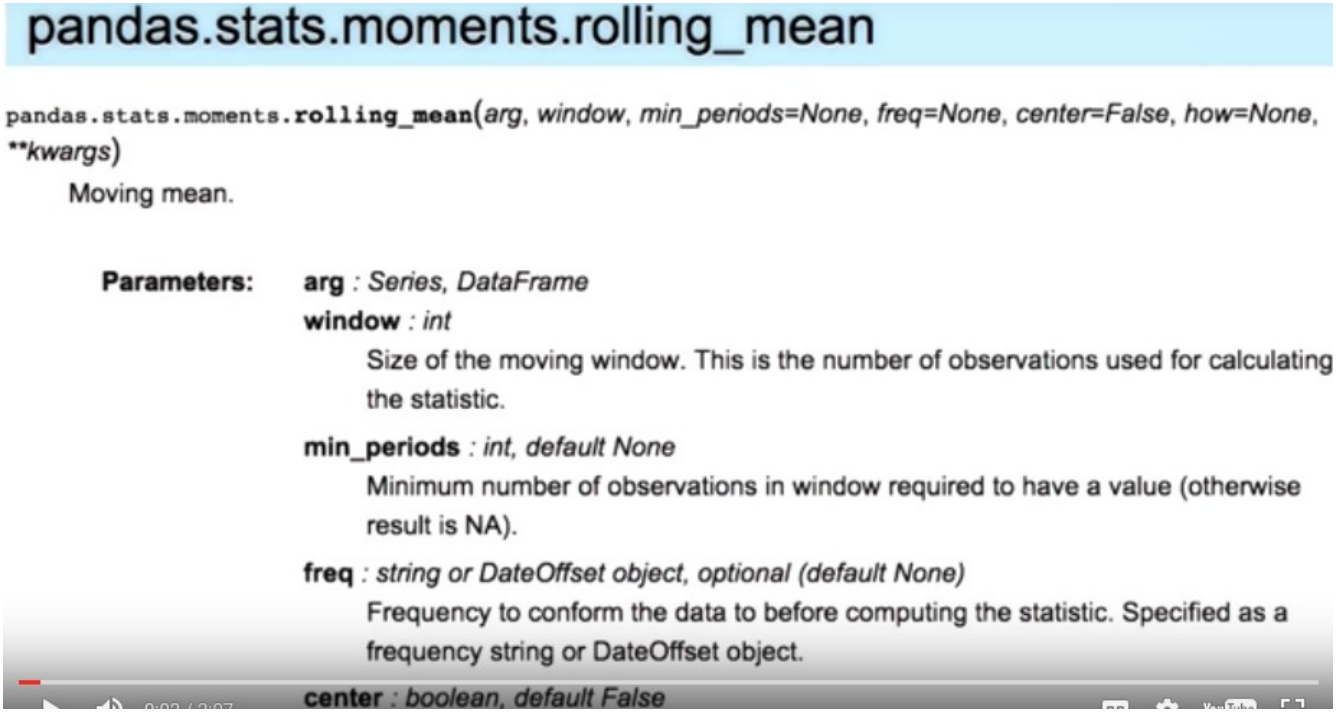
5. Assume we're using a rolling mean, and we're tracking the price here in blue. And we're looking for an opportunity to find when the prices diverged significantly far from the rolling mean that it might be an opportunity for, say, a buy signal or a sell signal. How can we decide that we're far enough away from the mean that we should consider something like that? So the question is, which statistic might we use to discover this? Here are a few options. Give it some thought, and check the box you think makes the most sense.

6. The answer is rolling standard deviation, and we'll show you why in the next note.



7. Returning to that question of how can we know if a deviation from the rolling mean is significant enough to warrant a trading signal, we need some way of measuring that. And John Bollinger, in the 1980s, came up with something he calls Bollinger bands. And whenever you mention that you have to put a little R there, because he has registered Bollinger bands as a trademark. If you don't do that,

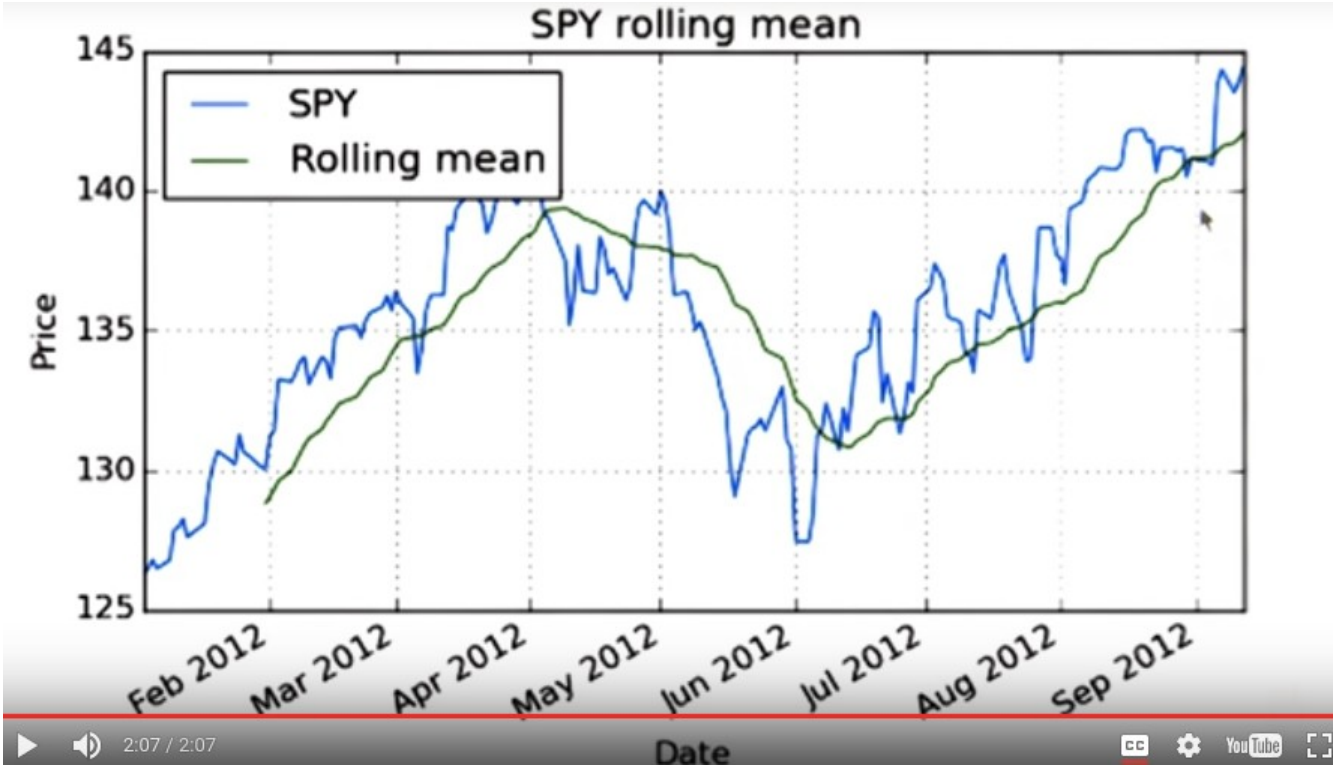
they'll come after you. [LAUGH] Anyhow, how might we measure that? What Bollinger observed was that we ought to take a look at the recent volatility of the stock. And if it's very volatile, we might discard movements above and below the mean. Whereas if it's not very volatile, a similarly sized movement maybe we should pay attention to. His idea then was to add a band 2 standard deviations above and 2 standard deviations below. Now I'm not going to make any comment as to how effective this method is. That's something for us to assess in the next mini course. But the theory anyways is that when you see excursions up to 2 sigma or 2 standard deviations away from the mean, you should pay attention. And in particular, if we drop below that and then up back through it, that is potentially right there a buy signal, because the hypothesis there is that we've gone quite far from the simple moving average and we're now moving back towards it. So if you buy there, you should anticipate positive returns as it climbs back through the average. Similarly, here where you see it punch through the top and then go back down through, that's potentially a sell signal. And as you can see, in this particular case, if we had bought here and sold there, we would've done great. But if you look at many, many examples of this, it's not always so great. So don't run off and start trading, but just be aware that this is an example of a technical indicator, and how you might involve it in a trading strategy. Dave is now going to show you how to read in data like this, compute a rolling mean, and chart it. And once again, I want to repeat that I'm not necessarily endorsing technical analysis here, although I think it can be very powerful. Just introducing some of these concepts to you. And again, in our later mini course, we're going to talk a lot about these approaches. Okay, here's to you, Dave.



8. For working with time series data, pandas provide a number of functions to compute moving statistics. We use rolling mean function to compute the rolling mean of the SPY. Note that rolling mean is not a DataFrame method but it is a function with the pandas library. So we wouldn't be able to call def.rollingmean. Instead we pass in a set of values for which rolling mean has to be calculated as the first parameter.

```
29 def plot_data(df, title="Stock prices"):
30     """Plot stock prices with a custom title and meaningful axis labels."""
31     ax = df.plot(title=title, fontsize=12)
32     ax.set_xlabel("Date")
33     ax.set_ylabel("Price")
34     plt.show()
35
36
37 def test_run():
38     # Read data
39     dates = pd.date_range('2012-01-01', '2012-12-31')
40     symbols = ['SPY']
41     df = get_data(symbols, dates)
42
43     # Plot SPY data, retain matplotlib axis object
44     ax = df['SPY'].plot(title="SPY rolling mean", label='SPY')
45
46     # Compute rolling mean using a 20-day window
47     rm_SPY = pd.rolling_mean(df['SPY'], window=20)
48
49     # Add rolling mean to same plot
50     rm_SPY.plot(label='Rolling mean', ax=ax)
51
52     # Add axis labels and legend
53     ax.set_xlabel("Date")
54     ax.set_ylabel("Price")
55     ax.legend(loc='upper left')
56     plt.show()
```

Now let's go for this. Firstly, let's get SPY data in our data frame for the year 2012. We also go ahead and plot the SPY data. Notice that we retain the matplotlib axis object so that we can add to it later on. Next we call the rolling mean function from pandas library, and pass in two parameters. As explained before the first parameter would be the values for which the rolling mean has to be calculated. Hence we pass our data frame containing SPY values. The next parameter is the window size, for which the mean will be calculated. We use a period of 20 days. This will return a series consisting of the rolling mean. It is always good to visualize the rolling mean. So we plot the series using the plot function. This time, while plotting the rolling mean, we pass in the matplotlib access object so that it gets added to the existing plot. Notice that we specified a label is equal to rolling mean. This will be used to create a plot legend. Let's add the legend and some access labels to our plot. So here we add our legend to the upper left corner of the plot at the X label and the Y label. Finally we are all set to view the plot.



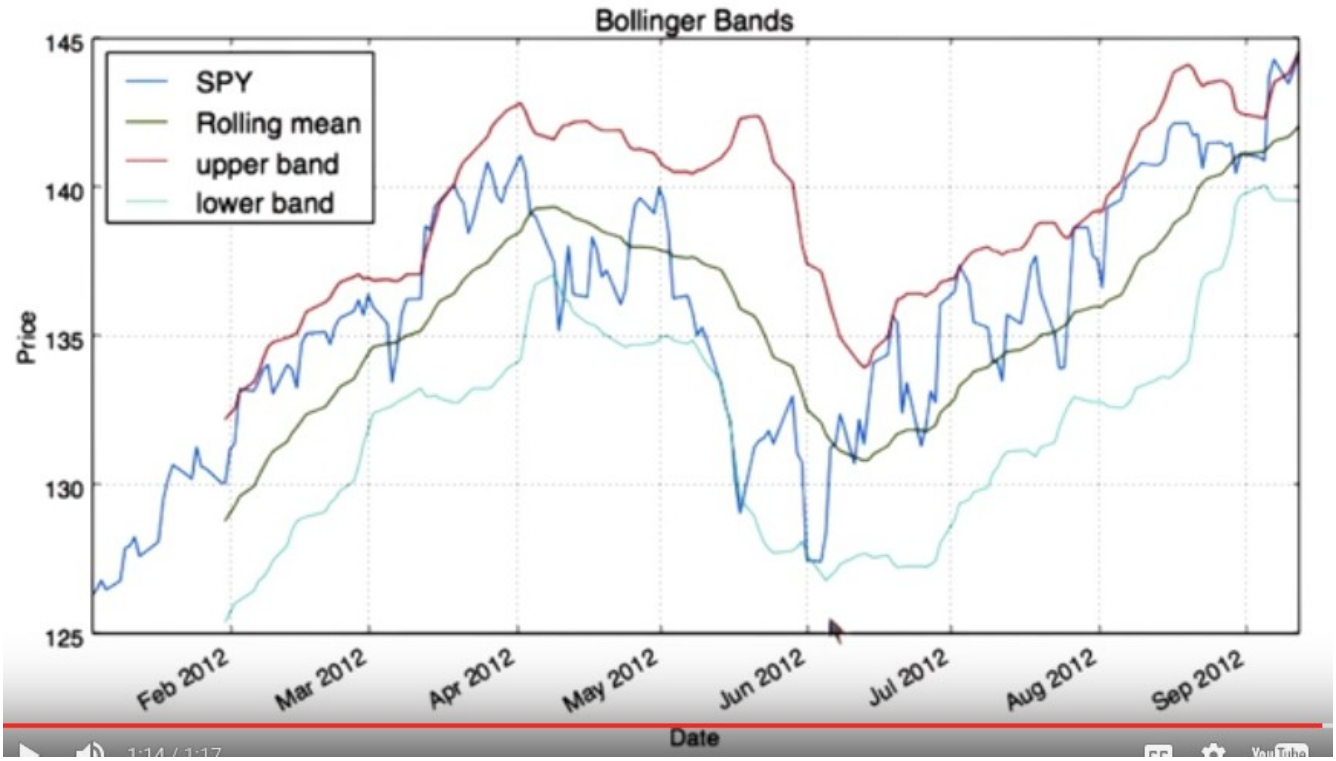
Observe that the rolling mean has missing initial values. The reason is that we defined a window period of 20 days, so the first 20 days there are no values. Also notice how it follows the movement of the raw prices, and is also less spiky.

9. It's quiz time. Professor explained to you how to get Bollinger Bands and now you get to try it yourself. Here's the data frame containing the stock prices for SPY for year 2012. Now computing Bollinger Bands consists of three main steps. First, compute rolling mean followed by computing rolling standard deviation. And then, finally, computing the values for the upper and the lower bands. We want you to implement one function for each step. You can call each function in this manner. Note that in this case, we use a window size of 20 for calculating rolling statistics. But we should be able to vary this. Finally, we plot the original prices, rolling mean, and the Bollinger Bands. Let me start you out with one of the functions. Here is how I would implement get_rolling_mean. Now go ahead and write code to compute the rolling standard deviation and calculate the upper and the lower Bollinger Bands. Wondering how to compute rolling standard deviation? Check out the trusty panels documentation for that. Now refer back to the previous video if you forgot how to calculate Bollinger Bands.

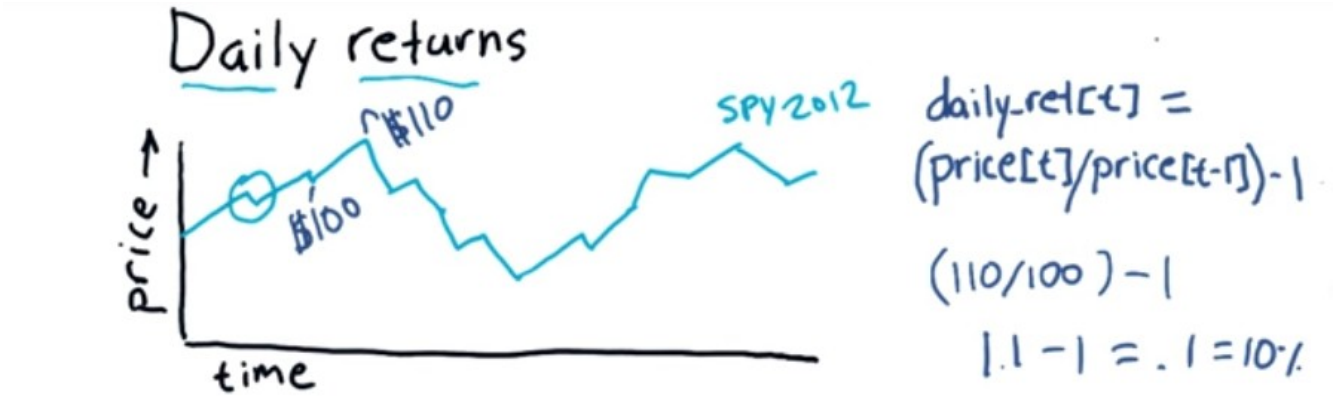
```python
def get_rolling_mean(values, window):
    """Return rolling mean of given values, using specified window size."""
    return pd.rolling_mean(values, window=window)


def get_rolling_std(values, window):
    """Return rolling standard deviation of given values, using specified window siz
    return pd.rolling_std(values, window=window)


def get_bollinger_bands(rm, rstd):
    """Return upper and lower Bollinger Bands."""
    upper_band = rm + rstd * 2
    lower_band = rm - rstd * 2
    return upper_band, lower_band
```

```python
def test_run():
    # Read data
    dates = pd.date_range('2012-01-01', '2012-12-31')
    symbols = ['SPY']
    df = get_data(symbols, dates)

    # Compute Bollinger Bands
    # 1. Compute rolling mean
    rm_SPY = get_rolling_mean(df['SPY'], window=20)

    # 2. Compute rolling standard deviation
    rstd_SPY = get_rolling_std(df['SPY'], window=20)

    # 3. Compute upper and lower bands
    upper_band, lower_band = get_bollinger_bands(rm_SPY, rstd_SPY)

    # Plot raw SPY values, rolling mean and Bollinger Bands
    ax = df['SPY'].plot(title="Bollinger Bands", label='SPY')
    rm_SPY.plot(label='Rolling mean', ax=ax)
    upper_band.plot(label='upper band', ax=ax)
    lower_band.plot(label='lower band', ax=ax)
```

10. Note how we calculated the rolling mean. The rolling standard deviation can be computed in a very similar way. Pandas provide a function called rolling_std to do this job. We simply pass in the values and the window size. Now, onto Bollinger bands. Recall that upper bound is two standard deviation above the rolling mean. Let's type this in our code. Here, we add 2 times the value of the rolling standard deviation to the rolling mean. Though the mean and the standard deviation values are in the form of series, the mathematics still works. It is similar to the arithmetic operation on numpy arrays, which is done element-wise. Next, let's calculate lower_band in a similar way. Here, I subtracted 2 times the rolling standard deviation values from the rolling mean. Note that will return the values for the two bands together. These are received back when the function is called.

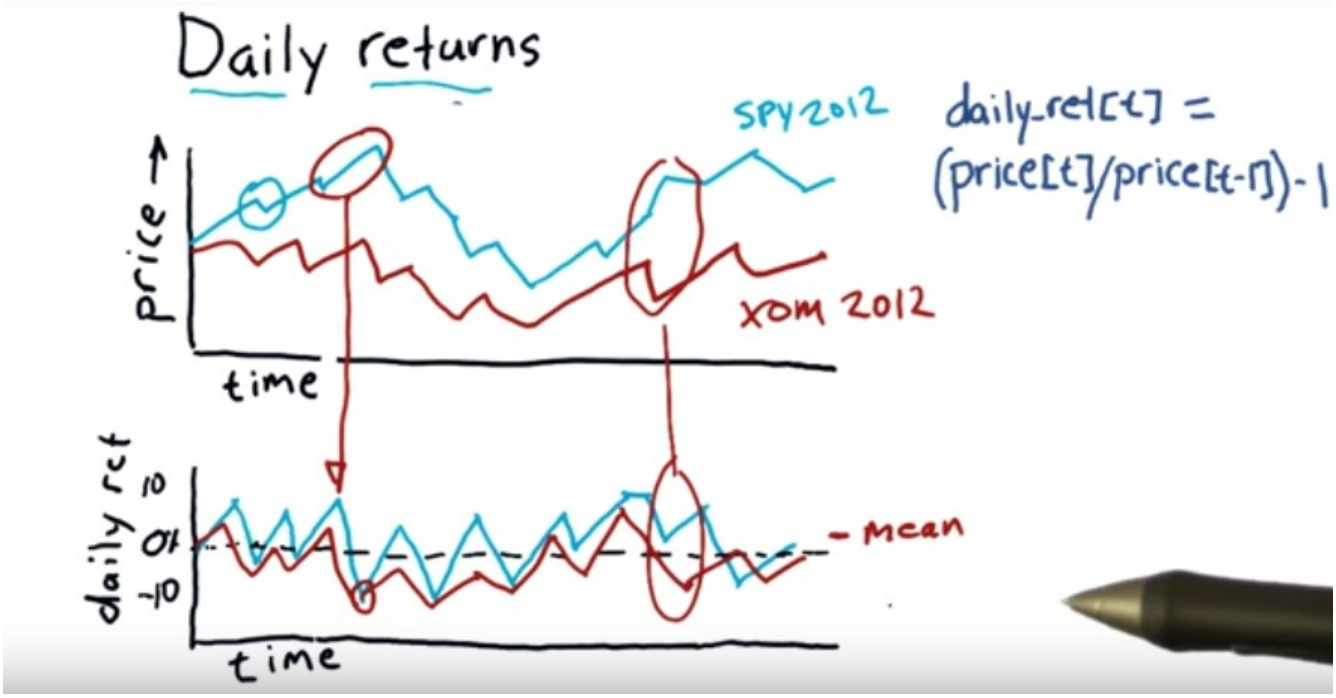


Let's see if a function computes Bollinger bands correctly. Looks good to me. Observe the selling and the buying points. You can play with the window size and see how the bands change. You could also try computing bands at different deviation away from the rolling mean.



11. Now we're going to look at something called Daily Returns. Daily returns are one of the most important statistics used in financial analysis. So let's consider first here this time series, S&P 500 in 2012. What daily returns are is simply how much did the price go up or down on a particular day? So, for instance, on this day it went down a little. On the next day it went up a lot. Daily returns are calculated easily using a simple equation here. So the daily return for day t, let's say today, is simply today's stock price divided by yesterdays' stock price, minus one. Let me show you an example. Let's suppose on this particular day the price went from $100 yesterday to $110 today. The daily return then, for that day, is (110/100)- 1, or 1.1- 1 = .1, which is 10%. So that's how we calculate daily returns. Now one thing to remember is this is a kind of statement you might put in a for loop where you iterate over individual days. Don't do that. Use the NumPy syntax we showed you, where you can do this in a

single statement with no for loops.

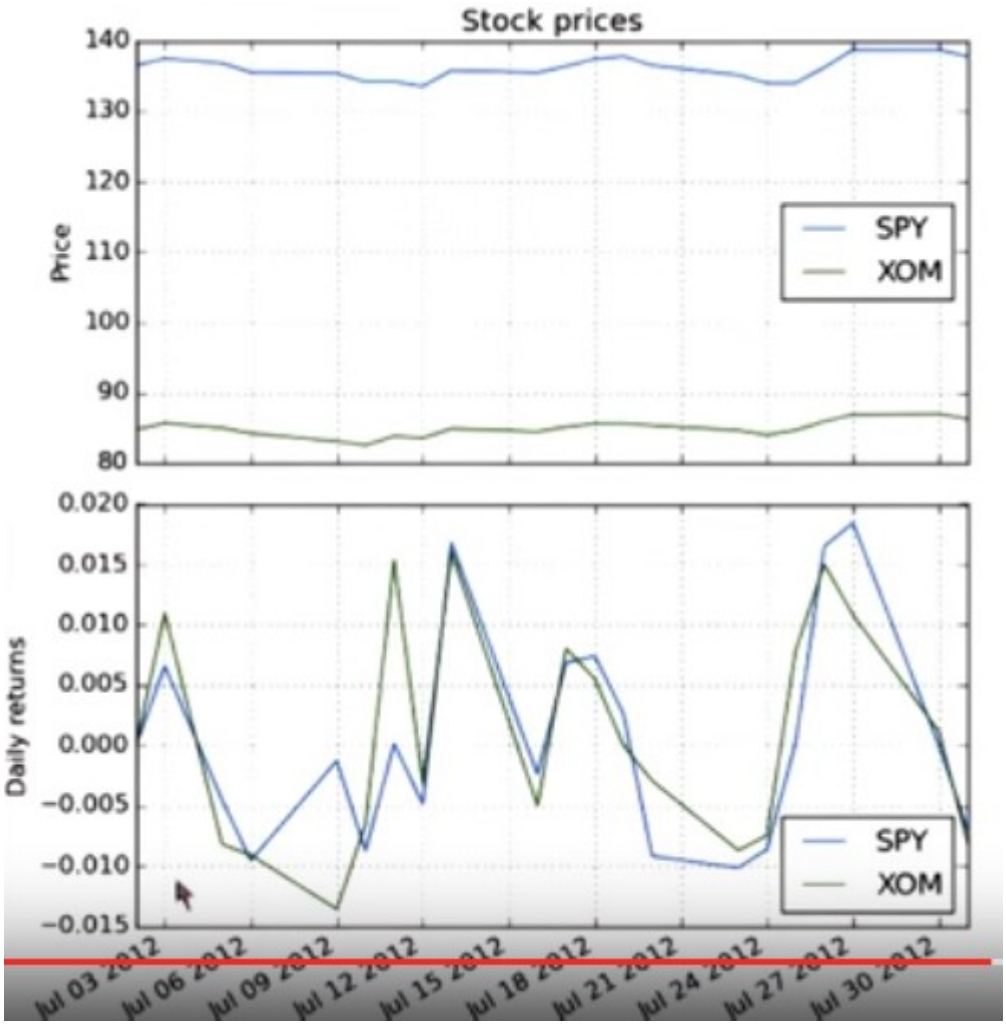


Here's what a chart for daily returns might look like. Everything is scaled now from -10% to +10%. And what we see here is the daily return for each day. If it was a positive return, of course it's positive, and negative if it were negative. Remember the day when we calculated we had a positive return of 10% that corresponds to that point right here. And for instance, here on the next day we had negative daily return. That corresponds to that point right here. Key thing to remember here is this is a line that sort of zigs and zags, usually close to zero. And if you were to, say, take the mean of all these values because we've had a generally upward moving trend here, our mean would probably be a little bit positive, above zero. Where looking at daily returns can be really important and revealing is to compare daily returns between different stocks or different assets. So, for example we might compare how Exxon moves in comparison to S&P 500. As one example, if you take a look at this section here you can see that when S&P 500 went up, Exxon went down and that's revealed here in this section of the daily returns. We're going to spend a lot of time in some future lessons, looking at how these statistics, specifically how daily returns between different assets, can be revealing. Dave is going to show you now in Python, how to calculate these daily return values. Here's to you, Dave.

12. Ta da! It's quiz time again. Can you write a function to compute daily return values? It should take a data frame as input. Apply the formula to calculate daily returns. Use proper slicing and indexing to avoid having to loop over each value. Note that the return data frame must contain the same column labels and the same number of rows as the given data frame. Which means if there are any missing or unknown values, replace them with zero.

```
31      ax = df.plot(title=title, fontsize=12)
32      ax.set_xlabel(xlabel)
33      ax.set_ylabel(ylabel)
34      plt.show()
35
36
37  def compute_daily_returns(df):
38      """Compute and return the daily return values."""
39      daily_returns = df.copy()  # copy given DataFrame to match size and column names
40      # compute daily returns for row 1 onwards
41      daily_returns[1:] = (df[1:] / df[:-1].values) - 1
42      daily_returns.ix[0, :] = 0  # set daily returns for row 0 to 0
43      return daily_returns
44
45
46  def test_run():
47      # Read data
48      dates = pd.date_range('2012-07-01', '2012-07-31')  # one month only
49      symbols = ['SPY','XOM']
50      df = get_data(symbols, dates)
51      plot_data(df)
```

13. First we make a copy of the data frame, where we can save computed values. Dataframe.copy will help us with that. For the next part, let's consider this. Suppose we want daily returns for date at index T, then we need to divide the value at index T by the value at index T minus 1. And subtract 1. We want to do that for all the dates, starting with index 1. Now let's code this. Here, df[1:] picks all the rows from 1 till the end. And df[:-1] picks all the rows from 0 till 1 less than the end. This operation cannot be done at index zero since we do not have the price of the stock prior to this day. So we set the values at the zero throw to all zeros. Finally, we return this data frame. You must be wondering why did we use dot values attribute of one of the intermediate data frames. The reason is to access the underlying numpy array. This is necessary because when given two data frames, Pandas will try to match each row based on index when performing element wise arithmetic operations. So all our effort in shifting the values by one will be lost if we do not use .values attribute. Okay, now let's run this.

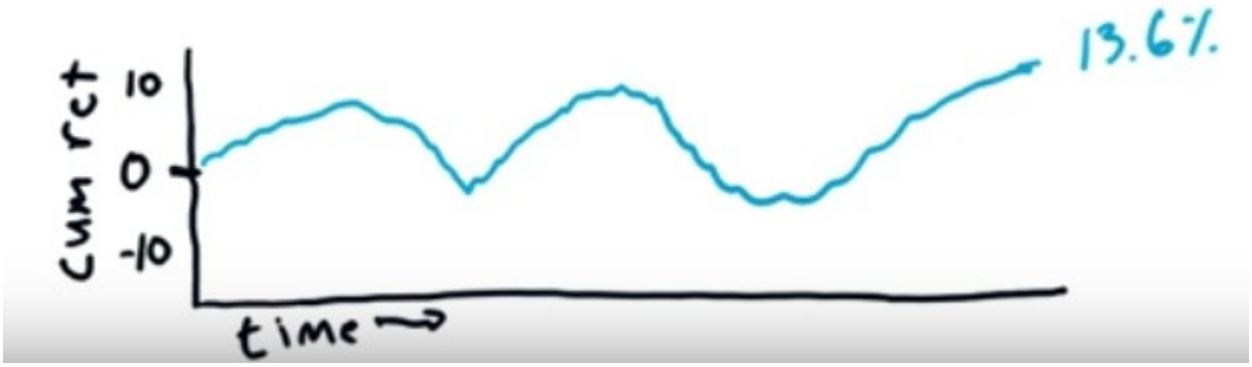


Here is what the daily returns look like compared to the original stock prices. As you can see, the original prices of SPY and XOM are quite different. However since the daily returns are implicitly normalized, they show up at a comparable scale. Each daily return value is either positive or negative fraction related to the previous day's value. This reveals that Exxon or Exxon Mobil actually matches ups and downs of the SPY quite closely.

```
37 def compute_daily_returns(df):
38     """Compute and return the daily return values."""
39     daily_returns = (df / df.shift(1)) - 1  # much easier with Pandas!
40     daily_returns.ix[0, :] = 0  # Pandas leaves the 0th row full of NaNs
41     return daily_returns
```

There is another way to compute daily returns. This time, directly using Pandas data frame. Here is how we can do it using Pandas data frame function, shift. Note that we still have to replace the values at the zero true with zeroes. The reason for doing this is, Pandas leaves these unknown values as 9 by default. Now let's check the output. As you can see the result is same as before.

14. One last important statistic on a stock that's important is called cumulative returns. So let's consider S&P 500, again, back in 2012. Now in 2012, the S&P 500 started the year at $125, and it ended the year at $142. When you listen to the news you hear things like, for the year 2012 S&P 500 gained 13.6%. That is cumulative return. You don't hear them say over 2012 S&P 500 went from $125 to $142. So how do you calculate these cumulative returns? It's really easy. Here's the equation. The cumulative return for a particular day, t, is just today's price divided by the price at the beginning. So price of zero is over here, and the price of any particular day, say would be here. And we can calculate the value like this. Now the cumulative return for the whole period is where t is this last day. So let's consider the example we've got here. To calculate the cumulative return for this whole year. It's the price at the end, divided by the price at the beginning, minus one. Turns out 142/125 is 1.136- 1 gives us .136 Which is equal to 13.6%. So our cumulative return for the ETF SPY was 13.6%.



We can calculate and chart cumulative returns just like we did earlier for daily returns, except now the plot is showing us the cumulative return of course instead of the data return. Note that the shape of the chart is the same as the price chart. It's just now it's normalized, and in fact this equation is exactly our normalization equation. So that is how to calculate and plot cumulative returns. We're not going to have Dave show you how to do that, you're on your own there, now that you know how to do daily returns, it shouldn't be that tough. Okay, that's it for this lesson, we'll see you again soon.