How to Generate these Popular Stock Terms using Python

Sicong Zhao Jan 16, 2020 · 6 min read

In the previous tutorial, we have introduced intrinsic value and 2 ways to download financial data (download here). Although very important, intrinsic value is not the only metric that value investors would consider.

In fact, a lot of important metrics are also easy to compute, which might serve as a better starting point especially if you do not feel confident about using Python or Pandas. But if you are already familiar with the tech and only care about intrinsic value, please feel free to skip to the next tutorial.

In this tutorial, we will cover the definition of some stock terms and then retrieve or calculate them from the financial data we have collected using Python. You can find the complete code at the end of this post.

Python. You can find the complete code at the end of this post.

The learning objectives are:

- Learn what are some important terms for value investing
- Learn to load financial data into Pandas
- Learn how to retrieve and calculate the terms using Pandas

. . .

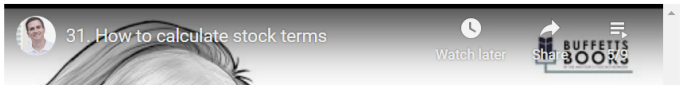
1. Stock Terms that will be covered

The following is the stock terms we will cover in this tutorial.

- Shares Outstanding
- EPS (Earning Per Share)
- Dividend Rate
- Dividend Yield
- Debt/Equity Ratio
- Book Value Per Share
- ROE (Return on Equity)
- Current Ratio
- P/E (Price to Earning)
- P/BV (Price to Book Value)

If you are already familiar with these terms, feel free to skip this part. If not, below is a video that covers all of these terms, you might want to watch it.

Just 2 quick notes: (1) The part about 'Equity Growth' (from 17:30 to 18:36) is not accurate. You might skip since it will not be covered in this section. (2) The definition of Current Ratio (from 20:27 to 22:23) is not correct, please refer to this Investopedia post.



. . .

2. Preparation

Codes are written in Python, you can use your local environment to do exercise. For the communication/demonstration purpose, I will upload my code to Google Colab and share it with you in this and next tutorial. The Google Colab is Jupyter Notebook running on Google's server, so we do not need to worry about the compliance issue.

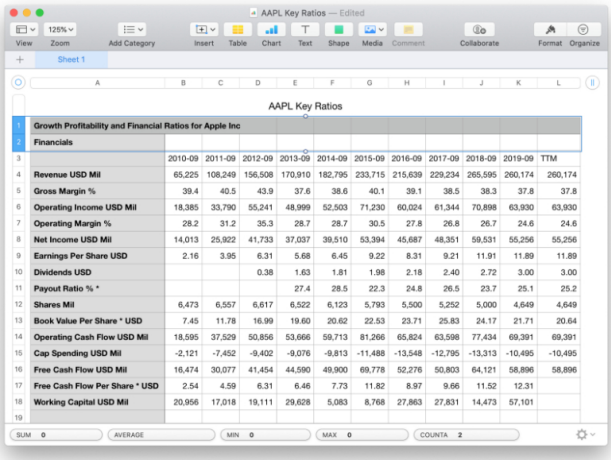
. . .

3. Load Data and Explore the Data

Pandas works like Excel. To do any calculation we need firstly load the data.

```
aapl = pd.read_csv('./stock_list/AAPL Key Ratios.csv', skiprows=2,
index_col='Unnamed: 0')
```

In this line of code, '*./stock_list/AAPL Key Ratios.csv*' is the file path, you need to substitute it with yours. Two parameters worth mentioning: 'skiprows=2' tells Pandas to ignore the first 2 rows. As you can see in fig.1 below, the first 2 rows are subtitles. Through 'index_col='Unnamed: 0", Pandas knows the first column should be index. (Yes, the column name '*Unnamed: 0*' is annoying, but it just represents 'the first unnamed column'. As you can see, if we ignore the first 2 rows, there is no column name specified for the first column.)



	2010-09	2011-09	2012-09	2013-09	2014-09	2015-09	2016-09	2017-09	2018-09	2019-09	TTM
Revenue USD Mil	65,225	108,249	156,508	170,910	182,795	233,715	215,639	229,234	265,595	260,174	260,174
Gross Margin %	39.4	40.5	43.9	37.6	38.6	40.1	39.1	38.5	38.3	37.8	37.8
Operating Income USD Mil	18,385	33,790	55,241	48,999	52,503	71,230	60,024	61,344	70,898	63,930	63,930
Operating Margin %	28.2	31.2	35.3	28.7	28.7	30.5	27.8	26.8	26.7	24.6	24.6
Net Income USD Mil	14,013	25,922	41,733	37,037	39,510	53,394	45,687	48,351	59,531	55,256	55,256
Earnings Per Share USD	2.16	3.95	6.31	5.68	6.45	9.22	8.31	9.21	11.91	11.89	11.89
Dividends USD			0.38	1.63	1.81	1.98	2.18	2.40	2.72	3.00	3.00
Payout Ratio % *				27.4	28.5	22.3	24.8	26.5	23.7	25.1	25.2
Shares Mil	6,473	6,557	6,617	6,522	6,123	5,793	5,500	5,252	5,000	4,649	4,649
Book Value Per Share * USD	7.45	11.78	16.99	19.60	20.62	22.53	23.71	25.83	24.17	21.71	20.64
Operating Cash Flow USD Mil	18,595	37,529	50,856	53,666	59,713	81,266	65,824	63,598	77,434	69,391	69,391
Cap Spending USD Mil	-2,121	-7,452	-9,402	-9,076	-9,813	-11,488	-13,548	-12,795	-13,313	-10,495	-10,495
Free Cash Flow USD Mil	16,474	30,077	41,454	44,590	49,900	69,778	52,276	50,803	64,121	58,896	58,896
Free Cash Flow Per Share * USD	2.54	4.59	6.31	6.46	7.73	11.82	8.97	9.66	11.52	12.31	
Working Capital USD Mil	20,956	17,018	19,111	29,628	5,083	8,768	27,863	27,831	14,473	57,101	

Fig.1 The structure of financial data

Then we can take a look at our dataframe:

```
aapl.head()
```

Using *.head()* function we can observe the first 5 rows of our data (Fig.2). It looks good.

	2010-09	2011-09	2012-09	2013-09	2014-09	2015-09	2016-09	2017-09	2018-09	2019-09	TTM
Revenue USD Mil	65,225	108,249	156,508	170,910	182,795	233,715	215,639	229,234	265,595	260,174	260,174
Gross Margin %	39.4	40.5	43.9	37.6	38.6	40.1	39.1	38.5	38.3	37.8	37.8
Operating Income USD Mil	18,385	33,790	55,241	48,999	52,503	71,230	60,024	61,344	70,898	63,930	63,930
Operating Margin %	28.2	31.2	35.3	28.7	28.7	30.5	27.8	26.8	26.7	24.6	24.6
Net Income USD Mil	14,013	25,922	41,733	37,037	39,510	53,394	45,687	48,351	59,531	55,256	55,256

Fig.2 Output of 'appl.head()'

. . .

4. Get Terms From Data

It turns out the financial data we collected in the previous tutorial is very comprehensive, it contains most of the terms in our list. Let's learn to retrieve them directly from our data, just so we can use them for further computation.

(1) Shares Outstanding

Let's firstly get the Shares Outstanding in Sep. 2019. The corresponding index name is 'Shares Mil', which is an abbreviation of 'Shares Outstanding in Million'. We can use function *.loc()* to locate the data. The first argument specifies the index, and the second specifies the column.

```
aapl.loc['Shares Mil', '2019-09']

# output:
# '4,649'
```

This result is fine except its format is string, not number. Which is not eligible for mathematical calculation. We can convert it to a number in the following 2 steps.

```
appl_share_str = aapl.loc['Shares Mil', '2019-09'].replace(',','')
appl_share = float(appl_share_str)
print(appl_share)

# Output:
# 4649.0
```

Instead of getting one data point, we might want the entire historical data. The following command would give us the complete historical 'Shares Outstanding'.

```
aapl.loc['Shares Mil', ]

# Output:
# 2010-09    6,473
# 2011-09    6,557
# 2012-09    6,617
# 2013-09    6,522
# 2014-09    6,123
# 2015-09    5,793
# 2016-09    5,500
# 2017-09    5,252
# 2018-09    5,000
# 2019-09    4,649
# TTM        4,649
# Name: Shares Mil, dtype: object
```

Compared with the code above, it simply leaves the second argument of *loc()* function blank.

In the output, 'dtype' means the data type. Similarly, we want it to be float for computational convenience. So, we can use the following code:

```
aapl.loc['Shares Mil', ].str.replace(',','').astype(float)

# Output:
# 2010-09    6,473
# 2011-09    6,557
# 2012-09    6,617
# 2013-09    6,522
# 2014-09    6,123
# 2015-09    5,793
# 2016-09    5,500
# 2017-09    5,252
# 2018-09    5,000
# 2019-09    4,649
# TTM        4,649
# Name: Shares Mil, dtype: float64
```

(2) EPS (Earning Per Share)

We can do exactly the same thing as above. But in order to improve efficiency, let's write a function and reuse it for the following terms.

```
def search_value(index_name, date):
    return float(aapl.loc[index_name, date])

def historical_value(index_name):
    return aapl.loc[index_name, ].astype(float)
```

So, we could use these two functions to retrieve data. The index name of EPS in our data is 'Earnings Per Share USD'.

```
search_value('Earnings Per Share USD', '2019-09')

# Output
# 11.89
```

The following code would update the *aapl* dataframe.

```
aapl.loc['Earnings Per Share USD'] = historical_value('Earnings Per
Share USD')
```

(3) Other Terms

Below is a dictionary lists the index name of each stock term in our dataframe. I am not going to do the exercise for the other stock terms here, you are encouraged to do this exercise.

5. Calculate Stock Terms (P/E and P/BV)

Our data does not contain P/E (Price to Earning) and P/BV (Price to Book Value), so we calculate them.

(1) Get stock price

Based on the definition of these two terms, we need to get the stock price in order to compute them.

I have introduced IEX Cloud at the beginning, that's a good choice. But here we use a package named *yfinance* to retrieve stock price because it's free and easy to use.

Since *yfinance* is a 3rd party library, we need to install it before we use it. You can find documentation and the installation instruction here.

After installation, we can have our stock price in this way:

```
import yfinance as yf
apple = yf.Ticker('AAPL')
apple_price = apple.history(period='max')
```

We have load historical price data of Apple into *apple_price* in the form of Pandas dataframe. We can take a look using *head()* function.

```
apple_price.head()
```

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1980-12-12	0.41	0.41	0.41	0.41	117258400	0.0	0.0
1980-12-15	0.39	0.39	0.39	0.39	43971200	0.0	0.0
1980-12-16	0.36	0.36	0.36	0.36	26432000	0.0	0.0
1980-12-17	0.37	0.37	0.37	0.37	21610400	0.0	0.0
1980-12-18	0.38	0.38	0.38	0.38	18362400	0.0	0.0

Fig.3 The output of 'apple_price.head()'

(2) Calculate P/E (Price to Earning)

Firstly, let's consider what data to use for P/E. Earning is from the financial statement, which remains constant until the next financial report. Price changes every day. As we care mostly about the latest P/E, so we will calculate it using Earning Per Share from the latest financial records and yesterday's close price for Apple.

```
latest_price = apple_price.loc['2020-01-14','Close']
latest_eps = search_value('Earnings Per Share USD', '2019-09')
latest_PtoE = latest_price/latest_eps
print(latest_PtoE)

# Output:
# 26.297729184188395
```

(3) Calculate P/BV (Price to Book Value)

Similarly, we care most about the latest P/BV, so we will calculate it using Book Value Per Share from the latest financial records and yesterday's close price for Apple.

```
latest_eps = search_value('Book Value Per Share * USD', '2019-09')
latest_PtoBV = latest_price/latest_eps
print(latest_PtoBV)

# Output:
# 14.402579456471672
```

6. Summary

Thanks for reaching the end, I hope you have found something meaningful to you. If the Python or Pandas component in this tutorial annoyed you, maybe go back to the course description and use the materials listed to get more practice. If you feel easy about the content, then it's a good time to start the next tutorial.

As mentioned above, I have pasted the code below. In the code, I also reformatted the data structure to make it more clean and readable. You can click the 'Open in Colab' button, and play around these codes online.

If you have any suggestions or feedbacks, please feel free to leave a comment or email me at sz163@duke.edu.

Previous tutorial: **1.Collecting financial data for fundamental analysis**

Next tutorial: **3. How to calculate the intrinsic value**

Finance | Stock Market | Data Science | Fundamental Analysis | Quantitative Finance