QuantDare (https://quantdare.com)

PYTHON

Calculate monthly returns…with Pandas

mgreco
(https://quantdare.com/author/mgreco/) *27/09/2017*

Calculating returns on a price series is one of the most basic calculations in finance, but it can become a headache when we want to do aggregations for weeks, months, years, etc. In Python, the Pandas library makes this aggregation very easy to do, but if we don't pay attention we could still make mistakes. Assuming that we want the return of the whole month, and we are not interested, for example, in the returns accumulated so far. These latter returns require that they be normalized to be comparable with the other returns.

Simple motivation

Given some prices on business days, you can get the trailing returns per month for the situation that we want to calculate. With this in mind, I'd like to describe how to avoid miscalculating monthly returns.

Note

Before you start, you may need some dependencies to do this:

```
# to install dependencies with anaconda installed
$ conda install pandas numpy pandas-datareader seaborn matplotlib
# else
```

```
4   $ pip install pandas numpy pandas-datareader seaborn matplotlib
```

install.sh hosted with ❤ by **GitHub** **view raw**

Let's get to work

We're going to work with the shares of "Banco do Brazil SA." Our first step is to download yahoo finance data using pandas_datareader:

```
1   >>> today = '20170926'  # to make static this script.
2   >>> tckr = 'BBAS3.SA'  # Banco do Brasil SA
3   # download data
4   >>> data = pdr.get_data_yahoo(tckr, 2014, today)
5   >>> data = data.asfreq('B')  # add frequency needed for some pandas functionalities releated wit
6   >>> data.columns = data.columns.map(lambda col: col.lower())
```

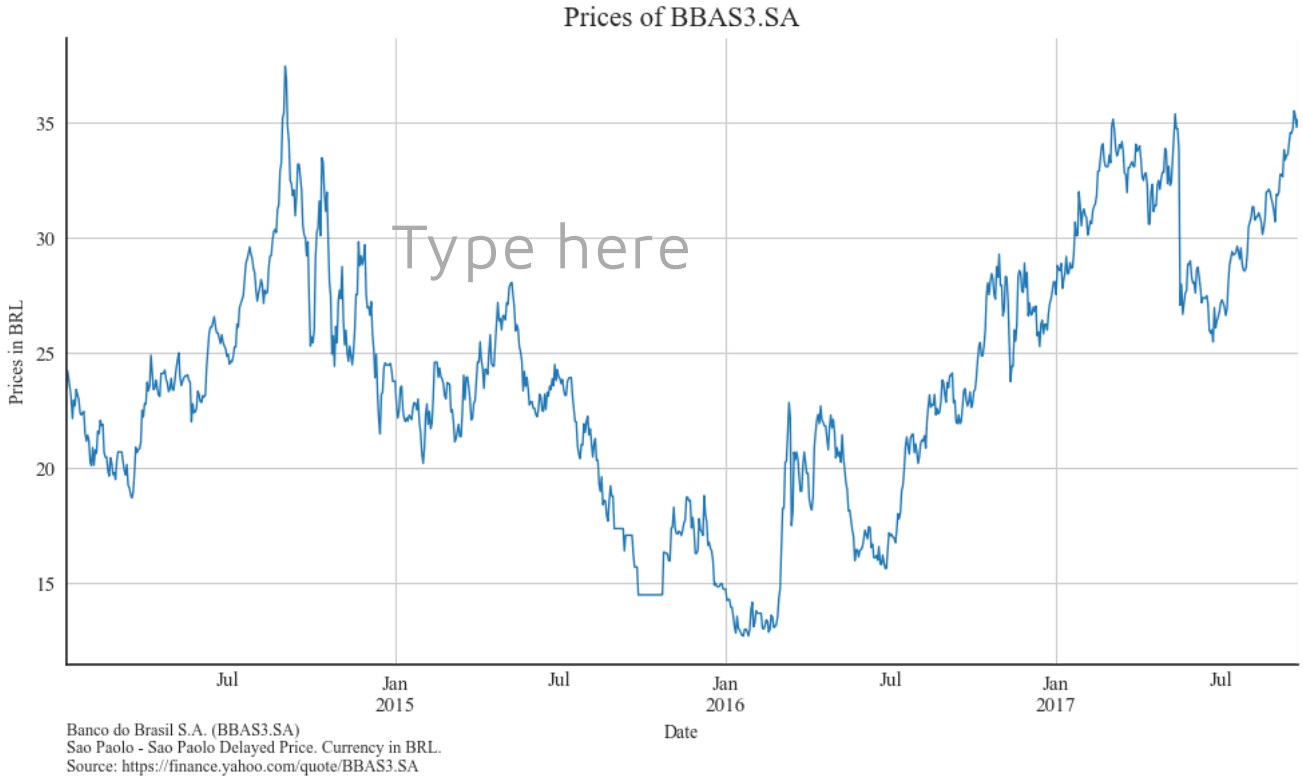
code1.py hosted with ❤ by **GitHub** **view raw**

Let's see what the data looks like:

```
1   >>> data
2                  open        high         low       close   adj close       volu
3   Date
4   20140102   24.450001   24.480000   23.950001   24.000000   18.998478
5   20140103   24.020000   24.250000   23.850000   24.250000   19.196381
6   20140106   24.020000   24.049999   23.610001   23.889999   18.911404
7   20140107   23.900000   24.100000   23.420000   23.480000   18.586851
8   20140108   23.500000   23.570000   22.969999   23.010000   18.214798
9   ....
10  20170920   34.660000   35.130001   34.310001   34.759998   34.759998
11  20170921   35.400002   35.970001   35.220001   35.520000   35.520000
12  20170922   35.369999   35.419998   35.020000   35.299999   35.299999
13  20170925   35.619999   35.740002   34.759998   34.799999   34.799999
14  20170926   35.160000   35.279999   34.950001   35.160000   35.160000
```

code2.py hosted with ❤ by **GitHub** **view raw**

If we plot the closing prices, we'll see this:



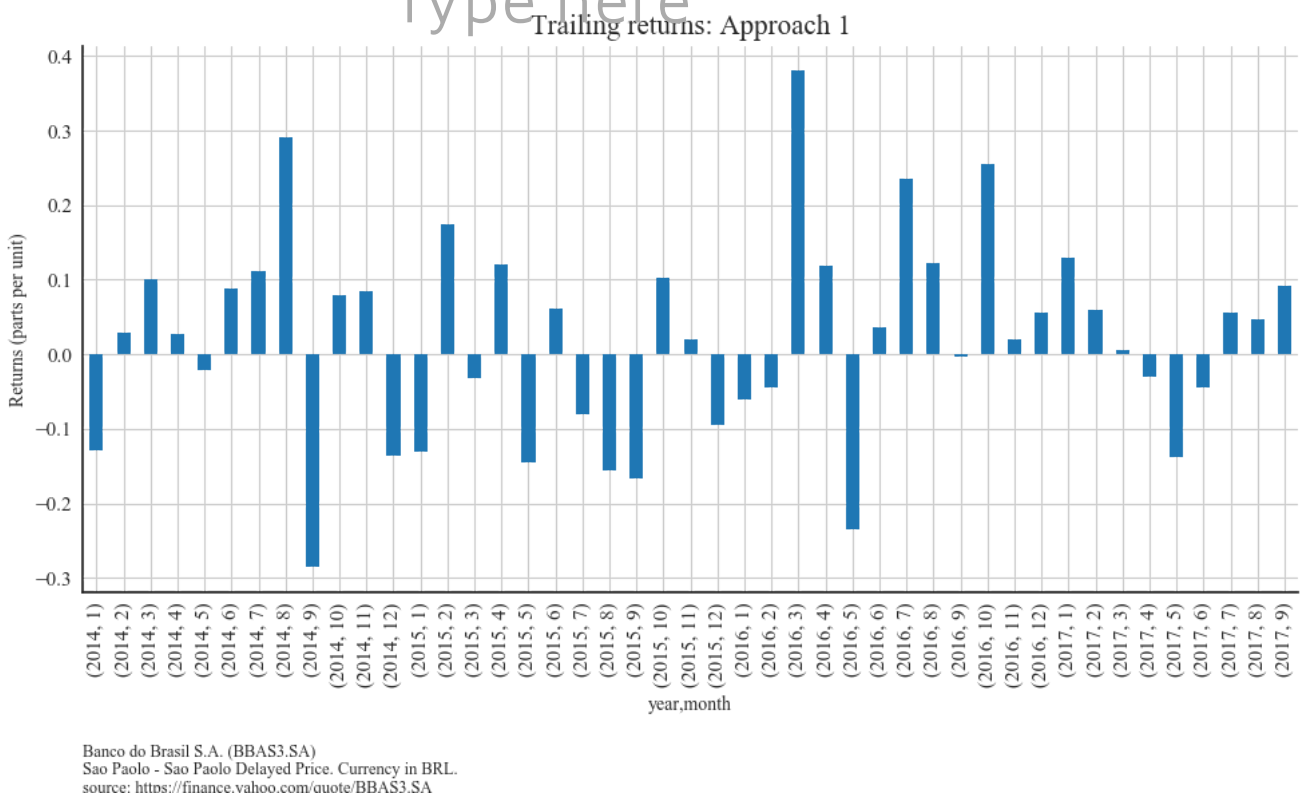
Now we'll work with closing prices. We're going to calculate the monthly returns, so we can do the following*:

* At the end of this post you will find the auxiliary functions used in the code, such as "total_return"

```
1
2   # Approach 1: starting from prices
3   # ================================
4   >>> approach1 = results_storage.groupby(['year', 'month'], )['close'].apply(total_return)
5   >>> approach1.tail(12)
6
7   year  month
8   2016  10       0.254927
9         11       0.020043
10        12       0.055222
11  2017  1        0.129630
12        2        0.058861
13        3        0.005060
14        4       -0.030115
15        5       -0.137291
16        6       -0.044223
17        7        0.056312
```

```
18         8        0.047782
19         9        0.102540
20   Name: close, dtype: float64
```

code3.py hosted with ❤ by **GitHub** **view raw**



Banco do Brasil S.A. (BBAS3.SA)
Sao Paolo - Sao Paolo Delayed Price. Currency in BRL.
source: https://finance.yahoo.com/quote/BBAS3.SA

The problem of this approximation is that it leaves out one day in the calculation of each monthly return, as it only takes into account the prices that belong to the month in question and completely omits all other information.

For the calculation to be correct, you must include the closing price on the day *before* the first day of the month, i. e. the last day of the previous month.

We can see it with an example: if we select month 8 of 2017, and see the prices that have been used to calculate returns, we will see that the series starts on August 1st and ends on August 31st. We have left out July 31st!, this happens every month.

```
1   >>> select_idx = (2017, 8)
2   >>> idx_approach1 = results_storage.groupby(['year', 'month'])['close'].groups[select_idx]
3   >>> last_group = results_storage.loc[idx_approach1]
4
```

```
5                    close       year    month   day     week_day week_day_name
6   Date
7   20170801        29.299999       2017    8       1       1       Tuesday
8   20170802        30.480000       2017    8       2       2       Wednesday
9   20170803        30.680000       2017    8       3       3       Thursday
10  20170804        30.870001       2017    8       4       4       Friday
11  20170807        31.350000       2017    8       7       0       Monday
12  ...
13  20170825        32.000000       2017    8       25      4       Friday
14  20170828        31.700001       2017    8       28      0       Monday
15  20170829        31.490000       2017    8       29      1       Tuesday
16  20170830        31.170000       2017    8       30      2       Wednesday
17  20170831        30.700001       2017    8       31      3       Thursday
```


It should also be mentioned that the last month (the current month) is not comparable with the rest of the months since it has not yet finished.

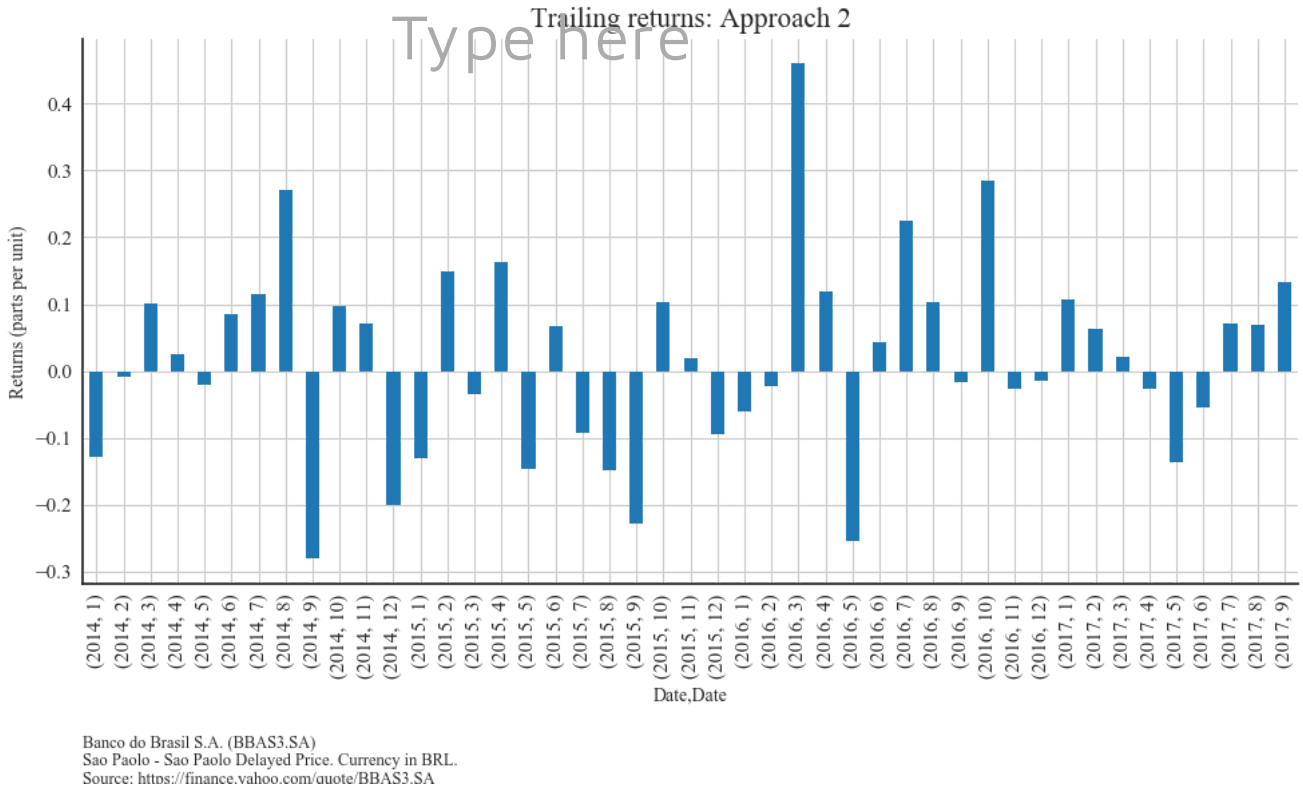
A second approach

Okay, as a second approach to incorporate the previous data, I could calculate the returns first, *then* group and calculate the total return with that series:

```
1   # Approach 2: starting from daily returns
2   # =======================================
3   >>> r = prices.pct_change()
4   >>> approach2 = r.groupby((r.index.year, r.index.month))\
5   >>>              .apply(total_return_from_returns)
6   >>> approach2.tail(12)
7
8   Date  Date
9   2016  10      0.284649
10        11     -0.026972
11        12     -0.014386
12  2017  1       0.107512
13        2       0.063966
14        3       0.020242
15        4      -0.027243
16        5      -0.137291
17        6      -0.054340
18        7       0.070896
19        8       0.069686
```

```
20         9       0.145277
21   Name: close, dtype: float64
```

code5.py hosted with ❤ by **GitHub** **view raw**



Banco do Brasil S.A. (BBAS3.SA)
Sao Paolo - Sao Paolo Delayed Price. Currency in BRL.
Source: https://finance.yahoo.com/quote/BBAS3.SA

However, all that glitters is not gold; this approximation has a problem in the first value. We still don't have the price before the one needed to make the calculation. The remaining months would be correctly calculated with the exception of the last return (current month), which is again not comparable with the rest.

The third approach

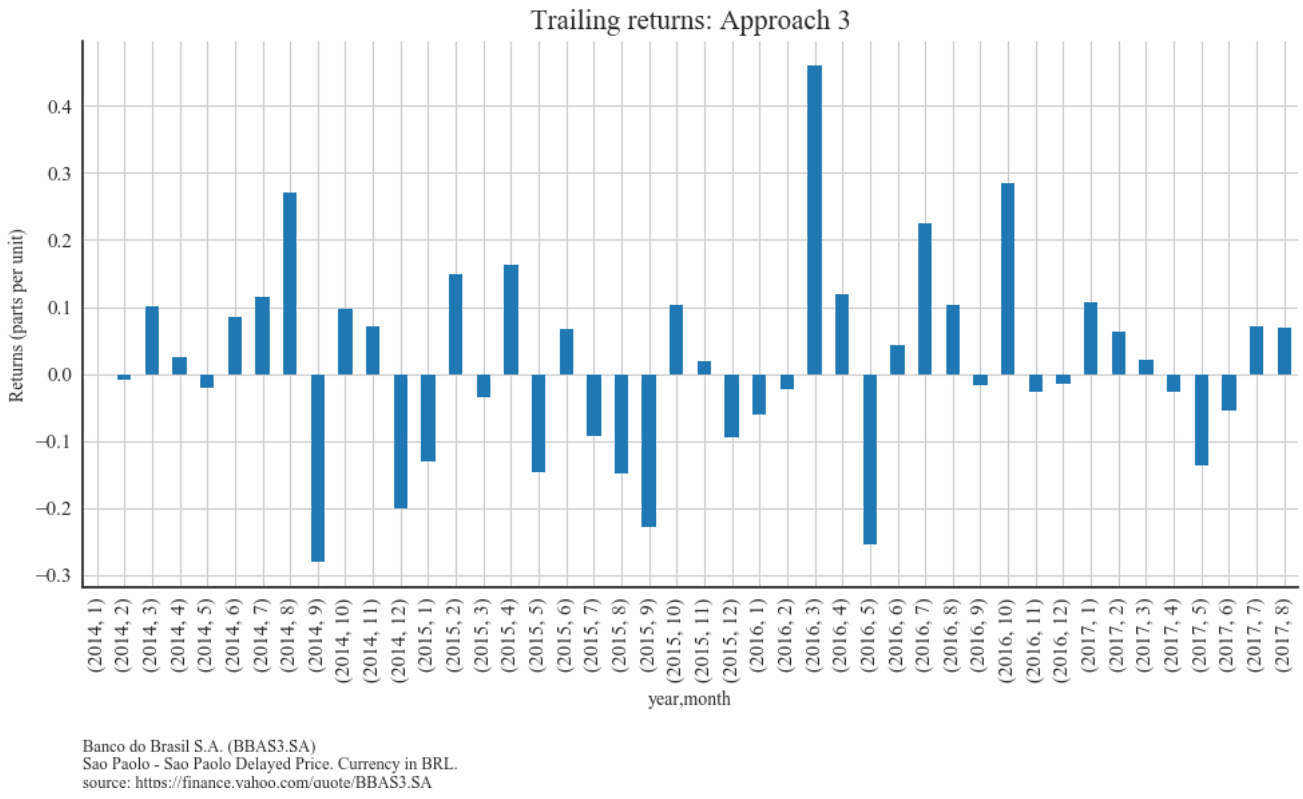
So, what can we do? The most correct result would come from first decimating the price series by taking only the last working day of the month, then grouping by year and month, and with the resulting series calculate, finally, the returns.


```
1   # Approach 3: Now yes, the definitive approach
2   # ============================================
3   >>> approach3 = results_storage.asfreq('BM').\
```

```
>>>                             .set_index(['year', 'month'])\
>>>                             .close\
>>>                             .pct_change()
>>> approach3.tail(12)

year  month
2016  9       -0.017665
      10       0.284649
      11      -0.026972
      12      -0.014386
2017  1        0.107512
      2        0.063966
      3        0.020242
      4       -0.027243
      5       -0.137291
      6       -0.054340
      7        0.070896
      8        0.069686
Name: close, dtype: float64
```

code6.py hosted with ❤ by **GitHub** **view raw**



Banco do Brasil S.A. (BBAS3.SA)
Sao Paolo - Sao Paolo Delayed Price. Currency in BRL.
source: https://finance.yahoo.com/quote/BBAS3.SA

The good thing about this approximation is that it doesn't return values for the first or last month with prices available, which can save us some calculation errors.

Conclusion

It's unfortunately very easy to make mistakes with these kinds of calculations. Pandas makes things much simpler, but sometimes can also be a double-edged sword. Nothing like a quick reading to avoid those potential mistakes. As we can see on the plot, we can underestimate or overestimate the returns obtained.

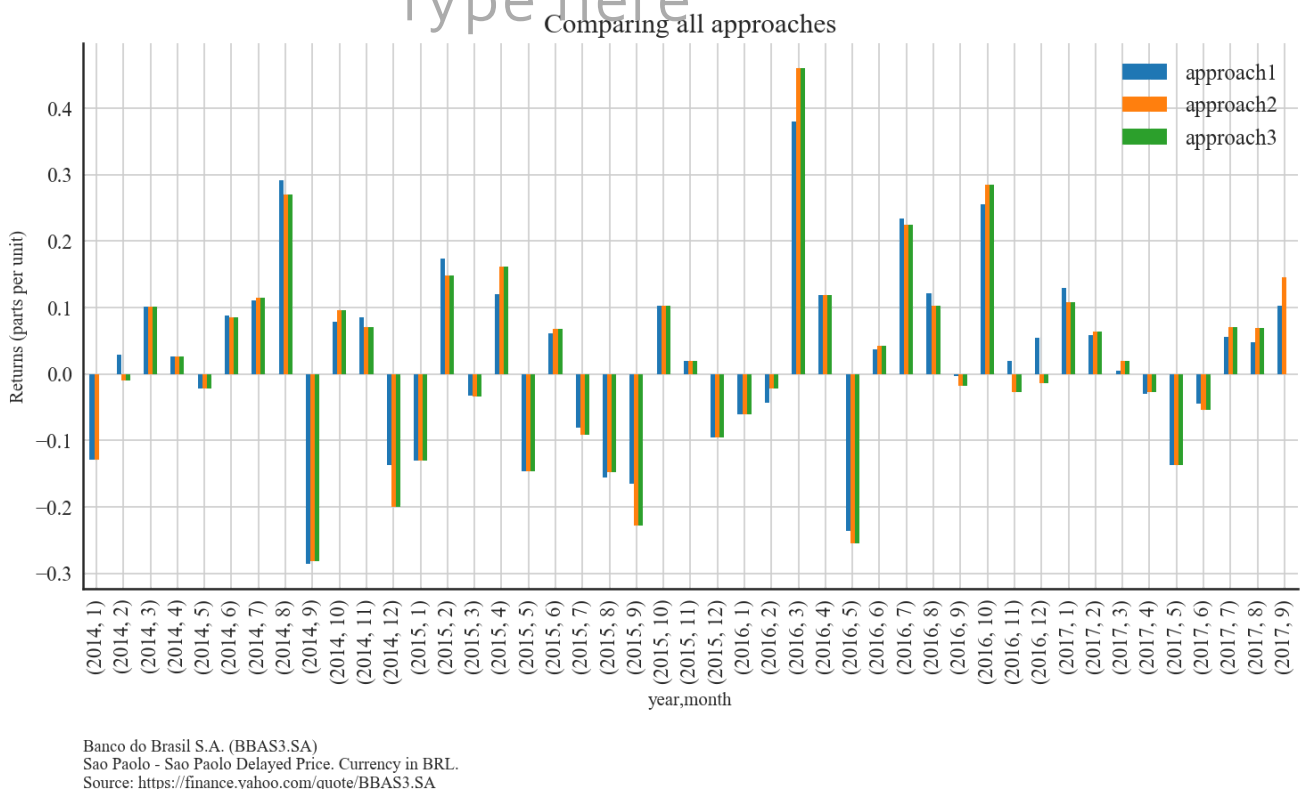
```
# Comparing all approaches
# ========================
>>> all_approaches = pd.concat([approach1, approach2, approach3], axis=1,
>>>                            keys=['approach1', 'approach2', 'approach3'])

>>> all_approaches

                 approach1       approach2       approach3
year    month
2014    1        -0.129583       -0.129583       NaN
        2        0.028827        -0.009095       -0.009095
        3        0.101449        0.101449        0.101449
        4        0.026778        0.025877        0.025877
        5        -0.021377       -0.021377       -0.021377
        6        0.087527        0.085627        0.085627
        7        0.111467        0.115493        0.115493
        8        0.291743        0.269841        0.269841
        9        -0.285916       -0.281250       -0.281250
        10       0.078988        0.096047        0.096047
        11       0.085130        0.071042        0.071042
        12       -0.136578       -0.199663       -0.199663
...
2016    10       0.254927        0.284649        0.284649
        11       0.020043        -0.026972       -0.026972
        12       0.055222        -0.014386       -0.014386
2017    1        0.129630        0.107512        0.107512
        2        0.058861        0.063966        0.063966
        3        0.005060        0.020242        0.020242
        4        -0.050115       -0.072242       -0.072242
        5        -0.137291       -0.137291       -0.137291
        6        -0.044223       -0.054340       -0.054340
```

```
32          7       0.056312        0.070896        0.070896
33          8       0.047782        0.069686        0.069686
34          9       0.102540        0.145277        NaN
```

code7.py hosted with ❤ by **GitHub** **view raw**



Banco do Brasil S.A. (BBAS3.SA)
Sao Paolo - Sao Paolo Delayed Price. Currency in BRL.
Source: https://finance.yahoo.com/quote/BBAS3.SA

- Approximation 1, gives us some miscalculations.
- Approximation 2, is a little closer to what we are looking for, but it has values that we should not use.
- Approximation 3, is the best method to use for this calculation.

Appendix: useful functions

```
1   # useful functions
2   # ================
3   def total_return(prices):
4       """Retuns the return between the first and last value of the DataFrame.
5       Parameters
6       ----------
7       prices : pandas.Series or pandas.DataFrame
8       Returns
```

```
    -------
    total_return : float or pandas.Series
        Depending on the input passed returns a float or a pandas.Series.
    """
    return prices.iloc[-1] / prices.iloc[0] - 1


def total_return_from_returns(returns):
    """Retuns the return between the first and last value of the DataFrame.
    Parameters
    ----------
    returns : pandas.Series or pandas.DataFrame
    Returns
    -------
    total_return : float or pandas.Series
        Depending on the input passed returns a float or a pandas.Series.
    """
    return (returns + 1).prod() - 1


def plot_this(df, title, figsize=None, ylabel='',
              output_file='imgs/fig_rets_approach1.png', bottom_adj=0.25,
              txt_ymin=-0.4, bar=False):
    if bar:
        ax = df.plot.bar(title=title, figsize=figsize)
    else:
        ax = df.plot(title=title, figsize=figsize)
    sns.despine()
    plt.ylabel(ylabel)
    plt.tight_layout()
    plt.text(0, txt_ymin, asset_info, transform=ax.transAxes, fontsize=9)
    plt.gcf().subplots_adjust(bottom=bottom_adj)
    plt.savefig(output_file, **kw_save)
```

aux_functions.py hosted with ❤ by **GitHub** **view raw**

Bonus

I have prepared a notebook with the complete code so that it can be executed and played with: Check it out here!

RELATED POSTS

Type here

(https://quantdare.com/robust-strategies-are-an-ally/)

(https://quantdare.com/robust-strategies-are-an-ally/)

ALL (HTTPS://QUANTDARE.COM/CATEGORY/ALL/)

Is robustness an ally?

(https://quantdare.com/robust-strategies-are-an-ally/)

Javier Cárdenas

(https://quantdare.com/author/javier-cardenas/)

(https://quantdare.com/create-your-own-deep-learning-framework-using-numpy/)

(https://quantdare.com/create-your-own-deep-learning-framework-using-numpy/)

ALL (HTTPS://QUANTDARE.COM/CATEGORY/ALL/)

Create your own Deep Learning framework using Numpy

(https://quantdare.com/create-your-own-deep-learning-framework-using-numpy/)

Alejandro Pérez

(https://quantdare.com/author/aperez/)

(https://quantdare.com/factor-exposure-the-turn-of-the-screw/)

(https://quantdare.com/factor-exposure-the-turn-of-the-screw/)

ALL (HTTPS://QUANTDARE.COM/CATEGORY/ALL/)

Factor Exposure: The Turn of The Screw

(https://quantdare.com/factor-exposure-the-turn-of-the-screw/)

J. González

(https://quantdare.com/author/jgonzalezomega/)

(https://quantdare.com/have-you-tried-to-calculate-derivatives-using-tensorflow-2/)

(https://quantdare.com/have-you-tried-to-calculate-derivatives-using-tensorflow-2/)

ALL (HTTPS://QUANTDARE.COM/CATEGORY/ALL/)

Have you tried to calculate derivatives using TensorFlow 2?

(https://quantdare.com/have-you-tried-to-calculate-derivatives-using-tensorflow-2/)
mgreco
(https://quantdare.com/author/mgreco/)

Join the discussion...

1 1 0 2

Ettore Errazuriz

Why not Df.resample("M").last().pct_change(1) ?

Reply 1 year ago

Mgreco (Https://Quantdare.Com/Author/Mgreco/)

Hello Ettore Errazuriz, sorry for the late reply. Yes, It seems to be a good point, although it seems to work well it fails in the same way as "approach 2", at the end it gives us a return that shouldn't be there. I've updated the notebook picking up your proposal, you can have a look at it here (https://github.com/mmngreco/quantdare_posts/blob/master/calcular_retornos/agregate_returns.ipynb), and/or run it online here (https://beta.mybinder.org/v2/gh/mmngreco/quantdare_posts/master?filepath=calcular_retornos/agregate_returns.ipynb). Thank you very much for you question!! 😉

Reply 1 year ago