Creating a Time Series Object - Appendix

Hello everybody!

For most of the course, we use a fixed data set. The reason is simple - we want to be plugging in the same values for both the simple tests we do at the beginning of the course and the complicated analysis towards the end.

As you already know, this data set ends in 2018, so we wanted to show you how to update it if you wish so. We'll be scraping the data off of Yahoo Finance using a very convenient library created by Ran Aroussi. Of course, using an updated version of the data will probably yield different results, so have that in mind if you decide to proceed with the new data.

We're going to use a completely new package for this task, called "yfinance". Of course, the name comes from Yahoo Finance, where we're taking the values from.

Since this library is not pre-installed in Anaconda, we need to pip – install it first. If you need a hint on how to do that, please refer back to the lecture titled– Installing the Necessary Packages.

Once you're ready, please use Jupyter to open the Python Notebook file attached to this article.

Starting from the top, let's quickly go over the code before us and see why we use it.

The first cell clearly imports the relevant package we need to load the data.

```
In [1]: # Importing the necessary package
        import yfinance
```

The second cell imports another new library we haven't seen before. Actually, this code just ignores the warnings Python might send in case it thinks you're using the data incorrectly. Don't worry, because we'll talk more about this in the next section of the course.

```
In [2]: # Ignoring warning messages
        import warnings
        warnings.filterwarnings("ignore")
```

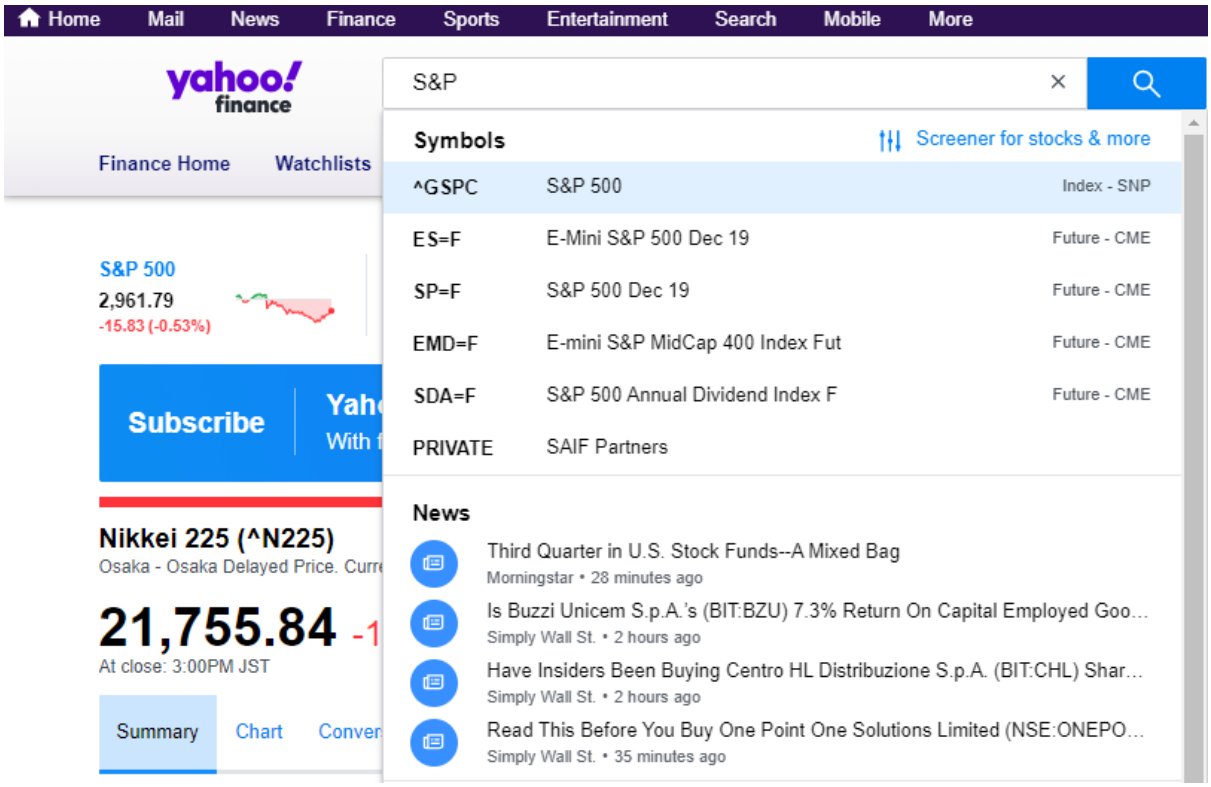
Next, we're using the download method to scrape our data from the Yahoo Finance webpage. The comments on screen represent what each argument does and how to use it properly.

```
In [3]: # Using the .download() method to get our data

        raw_data = yfinance.download (tickers = "^GSPC ^FTSE ^N225 ^GDAXI", start = "1994-01-07", end = "2019-09-27", interval = "1d", gr

        # tickers -> The time series we are interested in - (in our case, these are the S&P, FTSE, NIKKEI and DAX)
        # start -> The starting date of our data set
        # end -> The ending date of our data set (at the time of upload, this is the current date)
        # interval -> The distance in time between two recorded observations. Since we're using daily closing prices, we set it equal to
        # group_by -> The way we want to group the scraped data. Usually we want it to be "ticker", so that we have all the information
        # auto_adjust -> Automatically adjuss the closing prices for each period.
        # treads - > Whether to use threads for mass downloading.
```

Take a moment to examine the comments in cell 3 of the Python notebook, before we briefly go over what they do.

Okay, so the “tickers” argument takes the names of the time series we wish to include. However, the names of the indexes don’t always match the way they’re stored in Yahoo Finance. For instance, the S&P 500 isn’t expressed with SPX as we’re used to, but rather “^GSPC”.

Thus, if you ever want to find the way a market index or a stock is stored in Yahoo Finance, simply go to the website and use the search bar as shown below. The bolded representation on the left gives us information on how these are stored and how we should type them up within the quotation marks.



Then, we have the start and end arguments, which are pretty much self-explanatory. The only important factor here is the format of the dates. We’re using a YYYY/MM/DD format, so make sure to enter the correct start and end dates if you wish to switch them up.

The interval argument sets the frequency of the data. Since we’re using daily values, we set it equal to 1 day.

```
In [3]:
N225 ^GDAXI", start = "1994-01-07", end = "2019-09-27", interval = "1d", group_by = 'ticker', auto_adjust = True, treads = True)

in our case, these are the S&P, FTSE, NIKKEI and DAX)

 of upload, this is the current date)
ed observations. Since we're using daily closing prices, we set it equal to "1d", which indicates 1 day.
ta. Usually we want it to be "ticker", so that we have all the information about a time series in 1 variable.
ces for each period.
ing.

[**********100%**********]  4 of 4 downloaded
```

And the group_by argument groups all the data we’re importing based on the time series (tickers) it belongs to.

What about the auto_adjust column? Well, it simply replaces the closing prices with the adjusted closing prices based on the Open-High-Closing method. If we leave it as “False”, which is the default

value, we'll have Closing and Adjusted Closing Prices for each period of each time series. Therefore, we're just limiting surplus data, since the adjusted closing prices is what we use in our analysis anyway.

Lastly, the treads argument is related to how we download the data when we're dealing with massive amounts of data. Usually, leaving it as True is preferable.

Moving on to cell number 4. Here, we're just creating a copy of the data set, so that we don't have to scrape it anew if we happen to remove or alter elements by accident.

```
In [4]: # Creating a back up copy in case we remove/alter elements of the data by mistake
        df_comp = raw_data.copy()
```

Since notation like "^GDAXI" or "^N225" can be extremely confusing, we decide to add new columns to our data frame with the names we're familiar with from our original data set. Of course, as stated before, we're only using the closing prices, so we solely need the "Close" column of each time series.

```
In [5]: # Adding new columns to the data set
        df_comp['spx'] = df_comp['^GSPC'].Close
        df_comp['dax'] = df_comp['^GDAXI'].Close
        df_comp['ftse'] = df_comp['^FTSE'].Close
        df_comp['nikkei'] = df_comp['^N225'].Close
```

What we do next is to remove the first elements of each time series, because of how the download method is coded. Due to the fact that the closing and opening times vary when the data is stored, the dataset always starts 1 period before the "start" argument we set.

```
In [6]: df_comp = df_comp.iloc[1:] # Removing the first elements, since we always start 1 period before the first, due to time zone diff
        del df_comp['^N225']  # Removing the original tickers of the data set
        del df_comp['^GSPC']
        del df_comp['^GDAXI']
        del df_comp['^FTSE']
        df_comp=df_comp.asfreq('b') # Setting the frequency of the data
        df_comp=df_comp.fillna(method='ffill') # Filling any missing values
```

After taking care of that, we remove the surplus data. Since we already stored the closing prices in the new columns we created, we can get rid of the original series we scraped from the site.

Of course, the last two lines of the 6th cell once again set the frequency and handle any missing values.

Now, the 7th cell in the code is only there to see how we've done so far. The head method helps us make sure the initial elements of our data set are the same as the ones from the CSV file. The tail method is there to make sure we've correctly included all the data up to the period we are interested in.

In [7]:
```
print (df_comp.head()) # Displaying the first 5 elements to make sure the data was scrapped correctly
print (df_comp.tail()) # Making sure of the last day we're including in the series
```

```
               spx      dax    ftse    nikkei

Date
1994-01-07  469.90  2224.95  3446.0  18124.01
1994-01-10  475.27  2225.00  3440.6  18443.44
1994-01-11  474.13  2228.10  3413.8  18485.25
1994-01-12  474.17  2182.06  3372.0  18793.88
1994-01-13  472.47  2142.37  3360.0  18577.26
                spx       dax    ftse    nikkei

Date
2019-09-20  2992.07  12468.01  7344.9  22079.09
2019-09-23  2991.78  12342.33  7326.1  22079.09
2019-09-24  2966.60  12307.15  7291.4  22098.84
2019-09-25  2984.87  12234.18  7290.0  22020.15
2019-09-26  2977.62  12288.54  7351.1  22048.24
```

Of course, in practice, we also need to remove the surplus data and split it into a training and a testing set. Luckily, we just learned how to do that in the previous lecture, so you can complete that final step on your own.

Okay!

Now that you know how to get whatever data you wish from Yahoo Finance, you're able to play around and use any data you are interested in, without being confined to the one we provide you with.

For more details on using this new package, please check out the following link https://pypi.org/project/yfinance/

Thank you for reading and good luck!

Viktor