HOME ASSIGNMENT 3

DUE DATE: 4/6/20

CONCEPT:

- In this work you will learn a few tools for making predictions. This is a highly relevant and active field, and at the bottom there are a few links for extra reading for those of you interested in this topic.
- You will learn that computer predictions are usually based on learning from a dataset, and then making predictions.
- You will get a first experience at working with an API and getting data from online sources via python.

WHAT YOU MUST TO DO:

You will implement a class named 'PortfolioBuilder'. Its constructor takes no arguments. You will implement three public methods:

- **get_daily_data** - The method will access a public API and will get end of day stocks data. It will return the stocks data. More information below.
- **find_universal_portfolio –** The method will decide on a portfolio for the next day according to previously retrieved data. The method will use the universal portfolio algorithm. More information below.
- **find_exponential_gradient_portfolio** - The method will decide on a portfolio for the next day according to previously retrieved data. The method will use the exponential gradient algorithm. More information below.

SUBMISSION & GRADING

Submission:

- **Submission instructions are <u>very</u> clear. Failure to comply in any way will fail your code resulting in a grade of zero.**
- You must submit a **<u>single zip archive</u>**. This is regardless of the question how many files you put in that archive.
- The zip file archive **<u>must</u>** be named according to your id number.
- The archive must contain at least one file.
- This file **<u>must</u>** be named *your_id_number.py* where *your_id_number* is your id number, for instance 123456789.py
- The archive may or may not contain additional files. This is up to you.

Grading system:

- The grading system is very simple. It will unzip your archive into a directory and **look for a file name with your id number**. It will import this file, call the constructor of 'PortfolioBuilder', and call the required methods.
- If you wish to divide your work into several files you are welcome to do so. However, additional imports are your responsibility, and should happen within the main file with your id as a name, since this is the only file we import directly.
- The zip archive should not in any case contain sub-directories. These will fail to be imported correctly and fail your code.

WHAT YOU CAN AND CAN'T DO

Just like in real life no one will tell you how to solve your problem. We give you a task and leave the details to you. We do not impose any constraints (except a few necessary ones). You are welcomed to:

- Define how many classes, methods and properties you want. This is your business, not ours.
- You may divide your work into multiple files (according to submission instructions)
- You can import whatever you want, we don't care (except for limitations below).
- You may even look for code snippets or complete solutions online. This is a common and legitimate practice in real life.

Limitations:

- You can implement whatever you want, but we will call only the specified functions with the specified arguments. Make sure it is enough for your code to run properly.
- Regarding imports:
  - As stated above, you're allowed to use all imports.
  - However, this will require us first to install via pip the desired modules.
  - The template py file uploaded to Moodle, contains imports of all the modules which the course staff will ensure are available to you on the testing machine. Some of these will surely be useful to you.
  - Beyond these, you may import any built-in module which doesn't require installation via pip.
  - **If you feel these aren't enough you are welcomed to contact the course staff via the forum and ask for more modules. Only modules with a PyPI distribution (modules installable by pip) will be allowed. An additional module will be allowed to use only after a written approval by course staff.**
  - **A list of approved modules appears in Moodle**

Words of caution:

- If you use an unapproved module, your import statement will result in an exception and code crashing. This will not happen on your machine, if you did install it, and will therefore be undetected at the testing stage. **Make sure you work on a project which has exactly the approved modules installed.**
- Python modules have versions, and sometimes they are not backwards compatible. Our testing code will always use the most recent version (which will be published on Moodle). If you use codes snippets found online make sure no compatability issues arise.

- Be careful when using code found online!
    - I can assure you partial solutions for this task can be found online. In fact, I found some, but none of them worked out of the box.
    - If you use such a code, you're responsible for the results, not the person who wrote the original code.

GET DAILY DATA:

The method '**get_daily_data**' has the following prototype:

```
def get_daily_data(self, tickers_list: List[str], start_date: date, end_date: date = date.today()) -> pandas.DataFrame:
```

where:

- tickers_list – is a list of strings containing stock tickers. For instance ['GOOG', 'MSFT', 'AAPL'] for google, Microsoft, and apple.
- start_date – a date object from the datetime built-in module. Stands for the first date of the query. For example date(2018, 12, 31)
- end_date – a date object from the datetime built-in module. Stands for the last date of the query. For example date(2019, 12, 31)
- return value – returns a data frame which is a type of a table, where each row is a single date, and each column stands for a company. Values are quotes for adjusted end of day prices.
- Return type - pandas.DataFrame, more on that in links below.

BUT HOW DO I GET DATA, AND WHAT IS AN API?

API

You must get stock market data online (where else would you get?). For that, we use an API (application programming interface), which in our case, means using a python library, that someone else wrote for us. This library knows how to access a certain web server and make requests for data. If the server is nice, it then returns data to us for processing.

So how do I get data?

There are many APIs from which you can get data (specific tools listed below). Basically:

1. You choose one
2. You import it
3. You call the relevant function of the module
4. You wait (wait time can vary from few seconds to several minutes, depending on the server, your internet connection, and the amount of requested data.)
5. The function returns you the data.

I list here two modules which I think might be convenient for this task. You may use others, in accordance with the import limitations stated above.

- pandas-datareader – This is a well-documented, and heavily used module which allows access to several API's. Example on how to read data from the alpha-vantage API: example. Note that the documentation doesn't state yahoo finance formally as an option, however, it works. Here is example using yahoo finance: example (just remove the session argument in line 8).

- yfinance – Not as well documented, simple.
- There are other alternatives, if you prefer another one follow above instructions regarding additional imports.

Words of caution:

- There are many vendors of stock market data. Most of them will not give it for free.
- If you try any provider other than yahoo finance you must, register for an API_KEY which is like a password to access the API. If you use a provider which requires an API_KEY you **must hardcode the key** so that the testing system can access the data. **Thus, we strongly suggest the use of the free yahoo finance to avoid potential problems**.
- Additionally, some providers will give you very limited data for free (eg. Quandl), while others will severely limit you. For instance, alpha-vantage allows you to only query one stock each time, and up to 5 per minute.

You need:

- You need a provider which allows you to query at least 25 tickers in two minutes for this task.
- Again, use of yahoo finance is suggested due to simplicity and as it is free.

REQUIRED VALIDATION:

- The course staff will not try deliberately to make your code fail. All input will make sense and won't contain errors.

- However, in real life errors do occur. In our case potential issues may arise due to problematic internet connection, or missing data in the server.

- If any of the following occurs:

  - You request data from the server and receive any exception

  - You request data and receive a response, but the data contains null values (nan) if you're using the pandas dataframe.

  You must raise a ValueError exception. The testing code will catch it and give you up to 5 attempts with different tickers before failing you.

STOCK MARKET BASICS

- In the stock market many companies are traded. Each company has its own stock.
- Each company's stock has a different price. This price changes a lot for many reasons.
- I have a certain amount of money that I wish to invest, from here on we call it capital.
- I can divide the capital in many ways and buy different amounts of each stock. We will call each such choice to divide the capital a portfolio.
- From here on portfolios are denoted as vectors $\vec{b}_t$ where each element represents the fraction of capital invested in a stock. The index t will stand for time, i.e. in which day was this portfolio

used. Since portfolios contain capital fractions as elements, they must be summable to one, i.e.: $\sum_j (\vec{b}_t)_j = 1$.

- We define the vector $\vec{x}_t$ as: $(\vec{x}_t)_j = \frac{price\ of\ stock\ j\ at\ time\ t}{price\ of\ stock\ j\ at\ \ time\ t-1}$.
- We define the wealth (how much money we have at day T) as:

$$S_T(\vec{b}) = S_0 \prod_{t=1}^{T} \vec{b}_t \cdot \vec{x}_t$$

  From here on we assume $S_0 = 1$.
- Examples:
  - I have vectors of prices for some stocks at days 0 and 1 (adjusted closing prices):

$$\vec{p}_0 = (p_{0,1}, p_{0,2}, p_{0,3}) = (100,200,300)$$
$$\vec{p}_1 = (p_{1,1}, p_{1,2}, p_{1,3}) = (80,260,300)$$

    The first stock lost 20% of its value, the second gained 30%, and the third didn't move.
  - Thus, we get:

$$\vec{x}_1 = \left(\frac{p_{1,1}}{p_{0,1}}, \frac{p_{1,2}}{p_{0,2}}, \frac{p_{1,3}}{p_{0,3}}\right) = (0.8,1.3,1)$$

  - Let us assume we bought stocks using the following division of capital: 20%, 20%, 60% for the first, second and third stocks respectively. Then:

$$\vec{b} = (0.2,0.2,0.6)$$

  - Our wealth at the end of the first day will be:

$$S_1 = S_0 * \vec{b}_1 \cdot \vec{x}_1 = 1.02 * S_0$$

    Thus, after one day we're richer by 2%.

UNIVERSAL PORTFOLIO ALGORITHM:

The universal portfolio algorithm (invented by Cover at 1991), allows you to calculate the preferable portfolio given a set of previous stocks prices. The portfolio for the next time period is as follows:

$$\vec{b}_{t+1} = \frac{\sum_{\omega \in \Omega} \vec{b}^{\omega} S_t(\vec{b}^{\omega})}{\sum_{\omega \in \Omega} S_t(\vec{b}^{\omega})}$$

What it means:

1. Cover decided to do the following; given a set of possible stocks, find all possible portfolios (how many are there?). Denote the set of possible portfolios by Ω.
2. Now for each of these portfolios $\vec{b}^{\omega}$ (ω is not a power, just an upper index to distinguish from time), compute the wealth **assuming this portfolio was used for all days** up to day t.
3. Compute the above expression to decide on the portfolio for next day.
4. The algorithm is initialized (at the first day) with the portfolio divided equally between all stocks.

Two issues:

1. If we allow to buy half a stock (which cover allowed in his model), we have an infinite number of possible portfolios. In your implementation, you will allow "jumps" of $\frac{1}{a}$ between holdings on each stock, where a is an integer argument of the function, described later.

2. The time complexity of the suggested solution is NP-hard, as it is exponential in the number of stocks (Can you tell why?). Therefore, trying to solve it for many stocks, will take very long, meaning **FOREVER**. I solved it for 5 stocks which took several minutes. There are more efficient solutions, but they are beyond our scope.

EXPONENTIAL GRADIENT

The exponential gradient algorithm (also known as multiplicative updates), has a simpler update rule:

$$(\vec{b}_{t+1})_j = \frac{(\vec{b}_t)_j \exp\left(\frac{\eta(\vec{x}_t)_j}{\vec{b}_t \cdot \vec{x}_t}\right)}{\sum_k (\vec{b}_t)_k \exp\left(\frac{\eta(\vec{x}_t)_k}{\vec{b}_t \cdot \vec{x}_t}\right)}$$

In the above:

- η is a constant parameter called the learning rate which will be an argument of the function.
- Noe that η is a function of x, this is just a multiplication between the constant η and the j-th element of the vector $\vec{x}_t$.
- Hint: This expression is not as complicated to compute as it seems.
- This algorithm is much more efficient (why?), and can solve in several minutes for 100 stocks with ease.
- In the first day divide the portfolio equally between all stocks.

FIND_UNIVERSAL_PORTFOLIO

The method '**find_universal_portfolio**' has the following prototype:

```
def find_universal_portfolio(self, portfolio_quantization: int = 20) -> List[float]:
```

where:

- portfolio_quantization – is an integer (denoted by a above), describing the jumps in the portfolio calculation (see algorithm description above).
- Given a dataset of stock prices for t days, obtained before using the 'get_daily_data' method, do the following. For each day $1 \leq \tilde{t} \leq t$ the function does the following:
    - Consider all days in the interval $[0, \tilde{t}]$.
    - Using the above information, compute the universal portfolio for day $\tilde{t} + 1$
    - Compute the wealth which would have been attained, if the experiment would have ended at the end of day $\tilde{t}$.
- return value – The function returns the attained wealth (how much money we made) for each day in the interval $[1, t]$ by using the algorithm, as a list of floats.
- Return type – List of floats.

FIND_EXPONENTIAL_GRADIENT_PORTFOLIO

The method ‘**find_exponential_gradient_portfolio**’ has the following prototype:

```
def find_exponential_gradient_portfolio(self, learn_rate: float = 0.5) -> List[float]:
```

where:

- learn_rate – is a float, denoted by η, in the algorithm description.
- Given a dataset of stock prices for t days, obtained before using the ‘get_daily_data’ method, do the following. For each day $1 \leq \tilde{t} \leq t$ the function does the following:
  - Consider all days in the interval $[0, \tilde{t}]$.
  - Using the above information, compute the exponential gradient portfolio for day $\tilde{t} + 1$
  - Compute the wealth which would have been attained, if the experiment would have ended at the end of day $\tilde{t}$.
- return value – The function returns the attained wealth (how much money we made) for each day in the interval $[1, t]$ by using the algorithm, as a list of floats.
- Return type – List of floats.

ADDIOTNAL TIPS

- A file called ‘requirements.txt’ lists all modules suggested at the time of publishing this document. You can install all allowed modules as well as their dependencies by running a single command in the terminal. Within PyCharm, open the terminal at the bottom. Run the following command: *pip install -r requirements.txt*
- This will install all allowed modules and dependencies. The file will be updated, as more modules becomes approved following your requests.

REFERENCES:

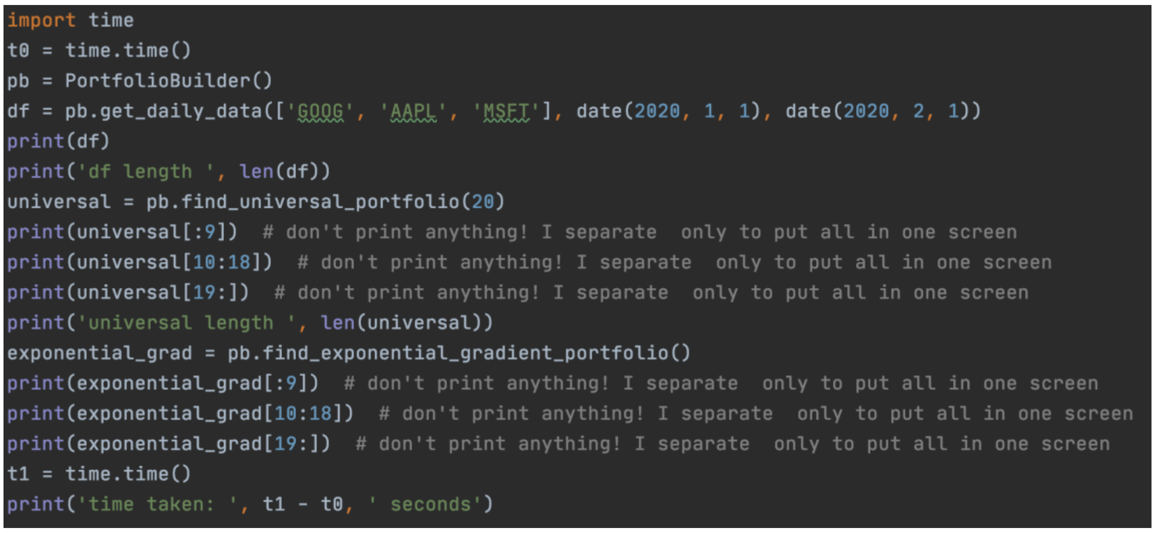
Python modules:

- NumPy
- Pandas
- matplotlib – Not really needed, only if you want to plot graphs, to see how well your algorithms did
- pandas-datareader

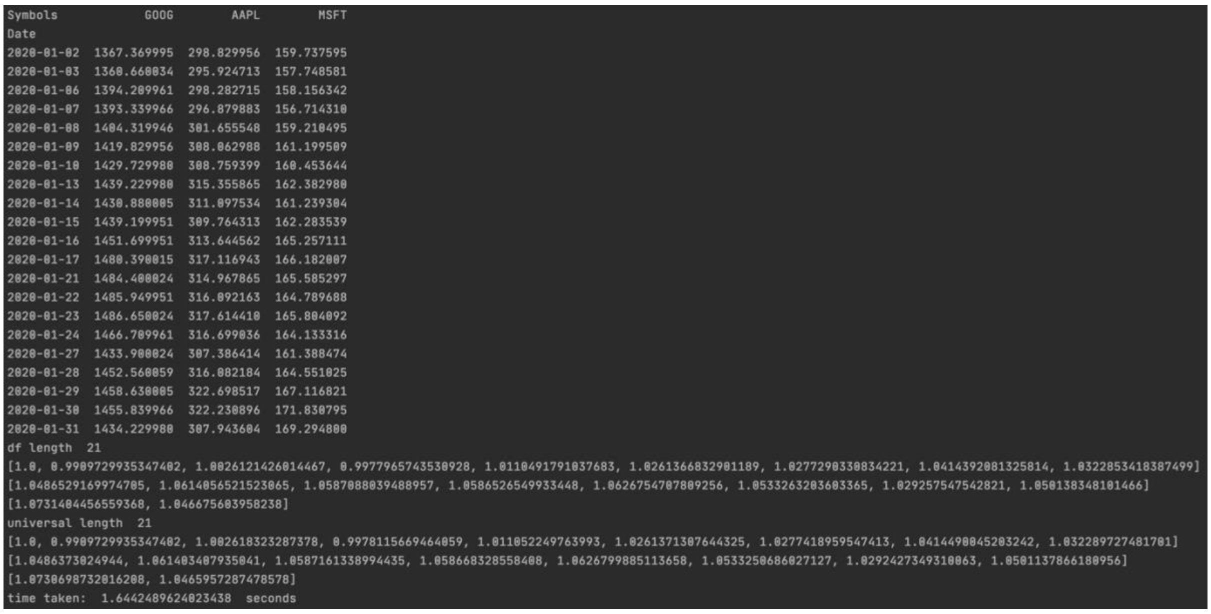
Quantitative finance (Mostly for students interested in further reading):

- http://thomasorton.info/regret_min_survey.pdf
- http://arxiv.org/abs/1212.2129
- Online Algorithms for the Portfolio Selection Problem - a book freely downloadable (legally), if you’re within the campus network, or connected via VPN

EXAMPLES



```
import time
t0 = time.time()
pb = PortfolioBuilder()
df = pb.get_daily_data(['GOOG', 'AAPL', 'MSFT'], date(2020, 1, 1), date(2020, 2, 1))
print(df)
print('df length ', len(df))
universal = pb.find_universal_portfolio(20)
print(universal[:9])  # don't print anything! I separate  only to put all in one screen
print(universal[10:18])  # don't print anything! I separate  only to put all in one screen
print(universal[19:])  # don't print anything! I separate  only to put all in one screen
print('universal length ', len(universal))
exponential_grad = pb.find_exponential_gradient_portfolio()
print(exponential_grad[:9])  # don't print anything! I separate  only to put all in one screen
print(exponential_grad[10:18])  # don't print anything! I separate  only to put all in one screen
print(exponential_grad[19:])  # don't print anything! I separate  only to put all in one screen
t1 = time.time()
print('time taken: ', t1 - t0, ' seconds')
```



```
Symbols            GOOG        AAPL        MSFT
Date
2020-01-02  1367.369995  298.829956  159.737595
2020-01-03  1360.660034  295.924713  157.748581
2020-01-06  1394.209961  298.282715  158.156342
2020-01-07  1393.339966  296.879883  156.714310
2020-01-08  1404.319946  301.655548  159.210495
2020-01-09  1419.829956  308.062988  161.199509
2020-01-10  1429.729980  308.759399  160.453644
2020-01-13  1439.229980  315.355865  162.382980
2020-01-14  1430.880005  311.097534  161.239304
2020-01-15  1439.199951  309.764313  162.283539
2020-01-16  1451.699951  313.644562  165.257111
2020-01-17  1480.390015  317.116943  166.182007
2020-01-21  1484.400024  314.967865  165.585297
2020-01-22  1485.949951  316.092163  164.789688
2020-01-23  1486.650024  317.614410  165.804092
2020-01-24  1466.709961  316.699036  164.133316
2020-01-27  1433.900024  307.386414  161.388474
2020-01-28  1452.560059  316.082184  164.551025
2020-01-29  1458.630005  322.698517  167.116821
2020-01-30  1455.839966  322.230896  171.830795
2020-01-31  1434.229980  307.943604  169.294800
df length  21
[1.0, 0.9909729935347402, 1.0026121426014467, 0.9977965743530928, 1.0110491791037683, 1.0261366832901189, 1.0277290330834221, 1.0414392081325814, 1.0322853418387499]
[1.0486529169974705, 1.0614056521523065, 1.0587088039488957, 1.0586526549933448, 1.0626754707809256, 1.0533263203603365, 1.029257547542821, 1.050138348101466]
[1.0731404456559368, 1.0466756039582381]
universal length  21
[1.0, 0.9909729935347402, 1.002618323287378, 0.9978115669464059, 1.011052249763993, 1.0261371307644325, 1.0277418959547413, 1.0414490045203242, 1.032289727481701]
[1.0486373024944, 1.061403407935041, 1.0587161338994435, 1.058668328558408, 1.0626799885113658, 1.0533250686027127, 1.0292427349310063, 1.0501137866180956]
[1.0730698732016208, 1.0465957287478578]
time taken:  1.6442489624023438  seconds
```