National Technical University of Athens

Department of Electrical Engineering & Computer Science

Multiple Criteria Decision Analysis

Project

Georgiou Dimitrios (03115106)

<dgeorgiou3@gmail.com>

June 2019

1 Introduction

1.1 *Preface*

The Multiple Criteria Decision Analysis (MCDA), which is a combination of the following fields:

- Field I Operations Research
- Field II Business Analytics
- Field III Informatics

is growing rapidly especially since the number of publications and papers are always increasing. In that direction, conflicting criteria are typical in evaluating different options upon a multicriteria problem we would like to resolve.

Numerous MCDA methodologies exist in many different variations, and have already successfully applied in all kinds of scientific domains. Despite the existence of a variety of research papers, the methodologies are still difficult to be exploited by non expert researchers. We deal with applied field of research so we should coherently understand every step of each methodology.

As far as our **project** is concerned, we will deal with portfolio management, as we are interested in getting high returns but at the same time minimizing our risk. We will analyze our investing horizon in the next subsection.

1.2 *Investing Horizon*

Investing is a globally accepted practice, in which we should firstly determine the characteristics of different types of investments and then match them with our individual needs and specific objectives.

In order to secure a well-established set of lifestyle, financial security, return, etc. the portfolio planned should meet theses lifetime needs. Being primarily focused on my education, managed investing is the best option for me. A manager will handle all of the details of my investment management (formation of portfolio allocation, etc.).

Investment Objective

The main reason I would like to invest is to generate an additional source of income, financing future needs (mainly buying a home) and last but not least financing a business idea we currently run with some classmates, combined with other types of debt-free funding (eg. crowdfunding, friends etc...)

Evaluating Risk Appetite

- Being 21 years old, 75% financial dependency to my parents, zero level of engagement (but consider myself as a passive investor), investible capital low
- 15 % knowledge about investment products, 60% inclination to learn, no other portfolio held before
- Time horizon: Willing to commit an investment program for ¿15 years to achieve my goals, with low liquidity requirements in the near future, medium sensitivity to tax savings
- Investment attitude 25% willingness towards risk-taking, 25% ability to cope up with small notional short-term losses in return for high long-term returns.

Summarizing, I consider myself as a **conservative investor**, with main objective to preserve the capital and receive regular income. In spite of being optimistic about the long-term prospects of the economy, I would prefer investments that would provide safety of principal and moderate income, so any fluctuations of more than 20% would concern me. This is closely connected with lowest rate of income return and yield.

So, ideally, we could invest in governmental bonds and corporate bonds, preferred shares with lower investment ratings and finally, debt or money market mutual funds (income schemes, FMPs, etc...). So, concluding the 2 main objectives of:

- **Capital Preservation**

- **Income**

lead to the seeking of low/medium degree of risk, but also the generation of income (or dividends)

On the other hand, stocks have the potential to generate higher returns than bonds, so whoever investor is willing to take on greater risks and would prefer to benefit of having partial ownership in a company, and the potential of rising stock price, would be better off investing in stocks

1.3 *Securities Concept*

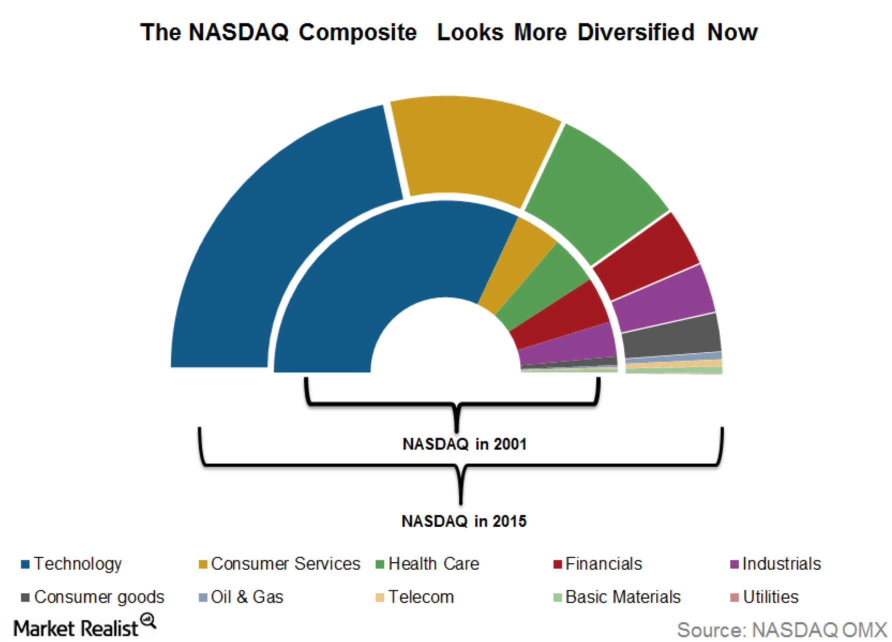
However, within this project we will deal with stocks traded in NASDAQ and which are part of the technology sector.

Tech Stocks

Tech stocks have been among the best performing stocks over the last decades, a trend that is likely to continue. The technology sector is an inescapably huge investment opportunity for both corporate America and Wall Street. More than anything, technology companies are associated with innovation and invention. Investors expect considerable expenditures on research and development by technology companies, but also a steady stream of growth fueled by a pipeline of innovative new products, services, and features.

NASDAQ

So let's focus on the **tech stocks traded in NASDAQ** and for which there are information provided by *investing.com*. NASDAQ is a leading provider of trading, clearing, exchange technology company services. Through its diverse portfolio of solutions, NASDAQ enables customers to **plan, optimize and execute** their business vision using proven technologies that provide transparency for navigating today's global capital markets. *Its technology powers more than 70 marketplaces in 50 countries, and 1 in 10 of the world's securities transactions.* NASDAQ is home to more than 3,500 listed companies with a market value of approximately $9.5 trillion and more than 10,000 corporate clients



MARKET BENCHMARK

Exclusively using the SP 500 Index, which is a good gage of the performance of the US stock market, would not be a strong measure of how a diversified, multi-asset portfolio looks in terms of risk and return. **However**, the Standard Poor's 500 Index is the most commonly used benchmark for determining the state of the overall economy. The SP 500 has become the leading stock index due to its broader scope. Many hedge funds compare their annual performance to the SP 500 – seeking to realize alpha in excess of the index's returns.

Our observations

1. Not only won't we deal with we low-risk securities, but on the contrary we will try to give some insights upon **stocks which are characterized as high-risk**, and more specifically on **tech stocks** that are of the highest risk since technology nowadays is still expanding exponentially and the trends come and go

all the time, leading to a real price & risk prediction challenge[1].
So let's try applying our preferences of our investing horizon (via the use of suitable weights) in tech stocks

2. Since the basic question is: **Investing or Trading ?**. Everyone that has spent maximum 1 hour searching upon these 2 definitions, will be familiar with the concept of *trade stocks, especially tech, that are of high volatility and have real fluctuations.*. However, in order for anyone to use the methodology followed in this project to conclude in a kernel of some stocks, and solve a **goal programming** problem later on, the kernel/result could be used for **long term investing and not for trading, it is not recommended.**
3. **We'll use SP 500 as a market benchmark to calculate risk-based results**

1.4 *Project Perspectives*

GOAL **Securities selection**

The initial perspective of this project was the implementation and the application of the following methods in a pool of stocks (~ 50) of a sector of our preference and analyzed upon criteria also of our preference, to come up with a small kernel of choices that would represent our result.

- ELECTREE I with veto
- TOPSIS
- PROMETHE

In general, portfolio management is composed of a variety of problems such as but not limited to:

- asset allocation,
- portfolio optimization based on some criteria,
- securities selection
- group decision making etc.

For the sake of completeness, we will expand the initial perspective of **securities selection** to:

GOAL **Group decision Making (2 DMs)** $\longrightarrow$ **Securities selection**

PART A : [Group Decision Making problem]

The goal of this part, is to preprocess data from 2 sources/decision makers to bring them into suitable processing forms, combine their normalized results and reach a consensus of x stocks ($20 < x < 30$), to further analyze them in **PART B**

The sources used are the following:

- *investing.com*
- *yahoo finance*

Let's focus on the preprocess applied to each one of the previous sources.

- **DM1: Investing.com**
  1. Download 91 stocks of NASDAD of technology sector as 3 .csv files, which include results of **fundamental, technical and performance** analysis

[1] In such occasions, **judgemental analysis** is considered really important (by consulting experts, or apply Natural Language Techniques (NLP)) except from statistical analysis which was only applied in the context of this project.

2. Import & preprocess these 3 .csv files, combine the information provided for each stock into a single final suitable dataframe.
3. Remove criteria, (or alternative dataframe's columns) that are not of utmost importance of our scopes. Additionally, remove stocks for which no data are provided by DM2. We are left with 89 stocks

- **DM2: Yahoo Finance**
The important difference with previous DM, is that here **we only use** risk-adjusted historical closed price, in order to calculate risk metrics and returns [2]. The edited dataframe will include these information, whereas for the rest criteria we will use DM1's data

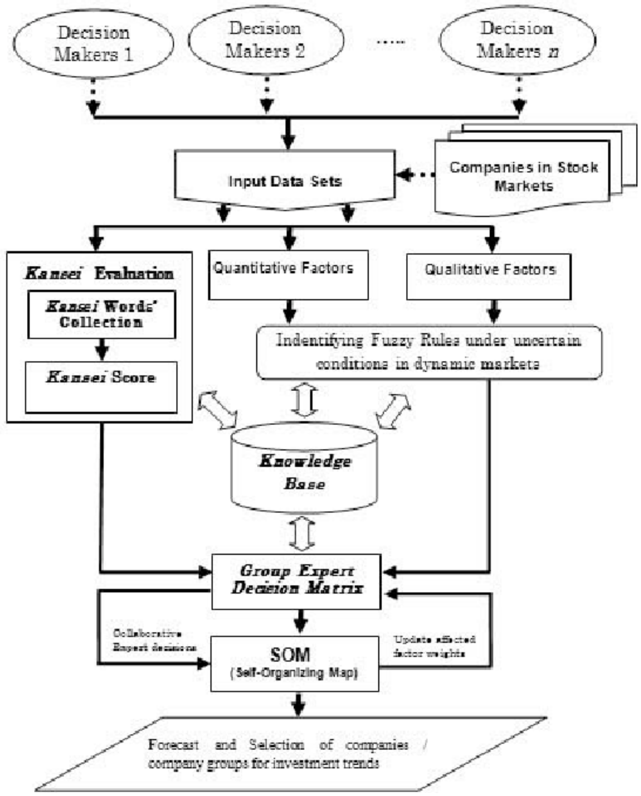
1. Pull Adjusted closing prices with Pandas datareader for all the 89 stocks
2. **Fundamenal Analysis**
   a) Calculate daily and annualized returns (1 Year, 3 Year, YTD)
   b) Calculate alpha and beta by using the CAMPP model. Additionally calculate r^2, sharpe ratio, treynor ratio and f-test ratio (Risk-based ratios)
   c) Based on the previous analysis drop all the stocks considering their bad performance on these metrics. **We are left with 39 stocks**
3. **Technical Analysis**: For each one of all these 39 stocks with good risk performance, train a LSTM model, which will predict last week and month's stock's movement. More specifically:
   a) Use the 80% of the 756 days historical adjusted closed prices, as the train set.
   b) Construct the LSTM model, which has 3 prices memory "looking back"
   c) Predict the adjusted closed prices for the rest 20% of the days, and keep separately the results for this month and this week.
   d) For the weekly and monthly predictions calculate Mean Error score (and not another metric since '-' is necessary in our occasion for a proper transformation) by finding the differences between predictions and real stock movement
   e) **Normalize** the mean errors in the range (-1,1), and translate this result into (0-4) range, as a representtation of a recommendation chart, and final result of our technical analysis
4. After these 2 analysis, it's time to combine the results into DM2's suitable dataframe.

- Now, we are finally ready to compare the results of DM1, DM2 and apply *group decision analysis*. More specifically:

1. We apply a weighted normalization to both DM1, DM2 results, calculating also their score for each stock.
2. We find the average, the standard deviation, the total score and the consensus for each stock, for both DM1 and DM2.
3. Finally, since we are dealing with group decision, we observed that a "good" standard deviation threshold would be $TH = 0.8$, and we excluded all stocks with standard deviation above that. **We are therefore left with 22 stocks**, which have a great consensus of at least 15 %

$$\textbf{91 stocks} \xrightarrow{\text{Exceptions removal}} \textbf{89 stocks} \xrightarrow{\text{Bad risk performance}} \textbf{39 stocks} \xrightarrow{\text{Group DM consensus \& std performance}} \textbf{22 stocks}$$

[2]It is of high importance for every investor tot be able to export his own metrics, and generally make security analysis

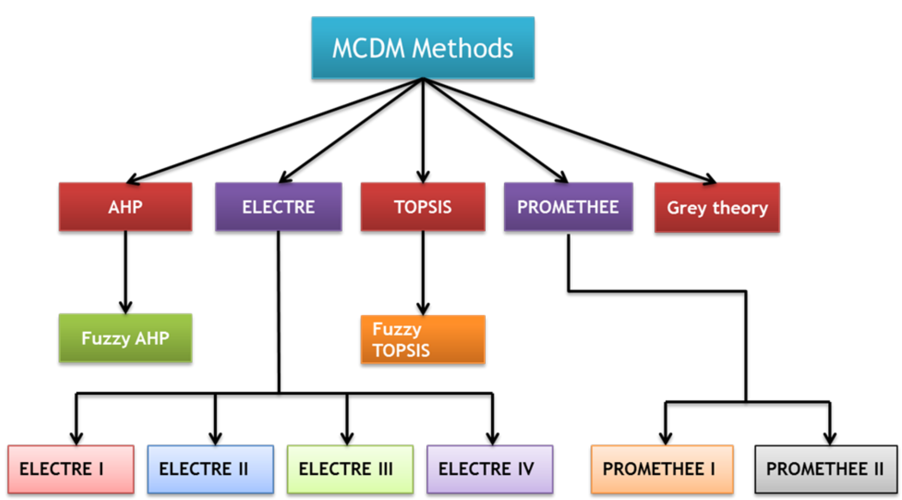


PART B : [Securities selection problem]

Generally about the methods

There exist many methods to solve multiobjective optimization problems. Methods which introduce some preference information into the solution process are commonly known as multiple criteria decision making methods. When using so called interactive methods, the decision maker (DM) takes an active part in an iterative solution process by expressing preference information at several iterations. According to the given preferences, the solution process is updated at each iteration and one or several new solutions are generated. This iterative process continues until the DM is sufficiently satisfied with one of the solutions found.

Many interactive methods have been proposed and they differ from each other e.g. in the way preferences are expressed and how the preferences are utilized when new solutions.



Goal

The goal of this part, is to take as input the 22 selected based on their risk and group decision making performance, and apply the methodologies mentioned in the beginning of this subsection, and finally reach a kernel of 6 or less stocks as the top n alternatives. [3]

[3]This result-subset could be a new input for an asset allocation/goal programming problem. It could be very interesting

22 stocks $\xrightarrow{\text{Securities Selection}}$ **5-7 stocks**

2 Main Chapter

2.1 *Group Decision Making*

- How can we expand the project perspective?
  - Firstly, we searched upon the database of the *investing.com* and we observed that the technology stocks belonging to NASDAQ exchange were 91.
  - Since the number of the stocks was huge to apply immediately **securities selection techniques**, we thought that we should somehow, based one some criteria, to eliminate those not relating to our scopes. So, applying a fundamental analysis upon the information provided by *investing.com*, and find our own normalized risk metrics and eliminate those not according to our low risk horizon, was the first idea.

FUNDAMENTAL ANALYSIS						PERFORMANCE ANALYSIS			TECHNICAL ANALYSIS	
Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
	our analysis (use investing.com data)									
	investing.com									

  - The next idea was to *not let the investing.com analysis unexploited.* We decided to use both investing.com, as DM1, and our analysed data, as DM2 and resolve a group decision problem, reaching a consensus of a subset of stocks. It is worth mentioning that the value of resolving a group decision problem is the diversity of each DM's estimation on each criterion. In that direction, we decided to combine our analyzed data (on *yahoo finance* adjusted closed prices) results with *investitng.com* to create a diversified DM2, enough different from DM1.

FUNDAMENTAL ANALYSIS						PERFORMANCE ANALYSIS			TECHNICAL ANALYSIS	
Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
	our analysis (use investing.com data)									
	investing.com									

+

FUNDAMENTAL ANALYSIS						PERFORMANCE ANALYSIS			TECHNICAL ANALYSIS	
Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
	our analysis (use yahoo finance data)									
	investing.com									

  - To enhance the prementioned argument, and since technical analysis is of high subjectivity, we build and train a LSTM neural network model for each and every of these 91 stocks (approximately ~ 45 minutes of training) to produce our own weekly and monthly prediction results for the stocks (via examing the ME metric between our prediction and the actual adjusted closed prices, and rescale (0-4)). So, we finally create descent diversified DM2 results.

FUNDAMENTAL ANALYSIS						PERFORMANCE ANALYSIS			TECHNICAL ANALYSIS	
Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
	our analysis (use investing.com data)									
	investing.com									

+

FUNDAMENTAL ANALYSIS						PERFORMANCE ANALYSIS			TECHNICAL ANALYSIS	
Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
	our analysis (use yahoo finance data)									
	investing.com									

- Steps taken?

  1. We imported all the necessary libraries in order to work with DataFrames, keras LSTM models, HTML api's etc..
  2. We scraped over *investitng.com* and download .csv data for fundamental,performance and technical analysis results for all 91 stocks.
  3. **Investitng.com**
     (a) **Tokenization Process:** We needed first of all to combine the data into a single file, easily manageable. So, after preprocessing the .csv files carefully to **tokenize** the information correctly, we create the new *stocks_investingcom* DataFrame.

- Translate abbreviations **eg.**We should translate 'B', 'M' of the Market Cap as $10^{12}, 10^{9}$ respectively
- Convert yield percentages into decimals
- Int conversion to floats

[a.csv]

(b) We also dropped criteria (columns) that we thought that are not necessary for the scopes of this project and we handle exceptions, by dropping stocks for which no info are provided by DM2 yahoo finance later on

	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
0	Apple	9.124400e+11	17.01	2.584900e+11	29290000.0	11.67	1.23	25.83	6.42	108.71	4	4
1	Microsoft	1.000000e+12	30.14	1.222100e+11	23690000.0	4.48	1.23	32.35	31.96	168.48	4	4
2	Intel	2.132400e+11	10.86	7.084000e+10	23770000.0	4.36	0.83	0.53	-11.75	46.66	0	2
3	Cisco	2.409200e+11	20.47	5.132000e+10	20060000.0	2.74	1.19	29.59	28.37	94.97	4	4
4	Broadcom	1.102900e+11	33.95	2.131000e+10	2790000.0	8.19	0.91	7.86	4.99	74.77	2	4

4. **Ticker Assigment Process:** Since *investing.com* doesn't provide the corresponding ticker symbols for each company, we had to download a company list, which had all NASDAQ company names and tickers. Since each source saves each company name with different ways, but tickers with the same, we had to apply a ticker assignment process and also handle name exceptions [4]

	Name	Symbol
0	Apple	AAPL
1	Microsoft	MSFT
2	Cisco	CSCO
3	Universal Display	OLED
4	AudioCodes	AUDC
5	Intel	INTC
6	Taitron	TAIT
7	Qualcomm	QCOM
8	Xilinx	XLNX
9	Bruker	BRKR
10	Sapiens	SPNS
11	Ubiquiti	UBNT
12	Cypress	CY
13	Intuit	INTU
14	CDW Corp	CDW
15	Equinix	EQIX
16	Broadcom	AVGO
17	Elbit Systems	ESLT
18	Texas Instruments	TXN
19	Garmin	GRMN

[c.csv]

5. **Yahoo Finance.com**

(a) We only need to extract the adjusted closed prices for a 3 year period **start='20-06-2016', end='20-06-2019**. Based on just that data (and 3-month T-bill, and SP 500 index), we are able to calculate risk metrics, and do a fair technical analysis. We finally can handle the *stocks_yahoofinance* DataFrame.

[4]The process is explicitly described into the corresponding code cell

	AAPL	MSFT	CSCO	OLED	AUDC	INTC
Date						
2016-06-20	90.507019	47.147133	26.153543	69.143028	3.915042	29.689028
2016-06-21	91.277893	48.201756	26.126299	67.211601	3.895990	29.827463
2016-06-22	90.935295	48.013435	26.080891	66.853195	3.905516	29.799772
2016-06-23	91.458733	48.879723	26.534950	68.804527	3.962669	30.445789
2016-06-24	88.889130	46.921150	25.200026	65.100975	3.886464	29.116842

(b) **Daily & Annualized Returns:** Modeling in continuous time, the growth of the stocks with log differences and compute APR and YTD For all 1-year and 3-year, stimulated by *investing.com*'s **perfomance analysis**. However, for the scopes of this project we only used **YTD** criterion of our analysis

$$change_t = log(price_t) - log(price_{t-1})$$
$$apr_t = change_t * 252 * 100 - r_f$$

where r_f = risk free rate or alternatively annualized YTD of the 3 month T-bill [5]

$$\boxed{\textbf{[d.csv]}}$$

(c) Calculate apr for market benchmark selected $SP\ 500\ market_{apr}$

(d) **CAMP model(Linear Regression):** For each stock we should calculate the parameters of the model:

$$y_i = \alpha + \beta * x_i + \epsilon_i$$

where $x_i = stock_{apr}, y_i = market_{apr}$, or alternatively we try to modelize the adjusted closed prices of our stocks as a linear combination of the market, especially sine **SP500 describes decently the whole stock movement**

$$\beta = r\frac{\sigma_{stock_{apr}}}{\sigma_{marketapr}}$$

where

$$r = colleration(stock_{apr}, market_{apr})$$
$$\alpha = \overline{market_{apr}} - \beta * \overline{stock_{apr}}$$

Alpha is a measure of an investment's performance on a risk-adjusted basis. It takes the volatility (price risk) of a security or fund portfolio and compares its risk-adjusted performance to a benchmark index.

MAXIMIZE Alpha

Beta, also known as the beta coefficient, is a measure of the volatility, or systematic risk, of a security or a portfolio compared to the market as a whole. Beta is calculated using regression analysis and it represents the tendency of an investment's return to respond to movements in the market.

MINIMIZE Beta

(e) **Extra Risk Metrics**:

- **R squared:**

$$R^2 = r^2$$

R-squared is a statistical measure that represents the percentage of a fund portfolio or a security's movements that can be explained by movements in a benchmark index. We need good R-squared in order for the regression model parameters to be trustworthy

[5] We used $r_f = 0.05(5\%)$ after doing some research

MAXIMIZE R-squared

– **Sharpe ratio**:

$$sharpe = \frac{\overline{stock_{apr}}}{\sigma_{stock_{apr}}} * \sqrt{251}$$

[6]

The **Sharpe ratio** tells investors whether an investment's returns are due to wise investment decisions or the result of excess risk. This measurement is useful because while one portfolio or security may generate higher returns than its peers, it is only a good investment if those higher returns do not come with too much additional risk. The greater an investment's Sharpe ratio, the better its risk-adjusted performance.

MAXIMIZE Sharpe ratio

– **Trenor ratio:**

$$treynor = \frac{r_s - r_f}{\beta}$$

where r_s = stock's annualized 3-Year return
The **Treynor Ratio** is a portfolio performance measure that adjusts for systematic risk. In contrast to the Sharpe Ratio, which adjusts return with the standard deviation of the portfolio, the Treynor Ratio uses the Portfolio Beta, which is a measure of systematic risk.

MAXIMIZE Treynor ratio

– **F-test ratio**

$$f - test = \frac{\frac{R^2}{k-1}}{\frac{1-R^2}{n-k}}$$

where n = the # samples of daily data used (756), and $k =$ parameters of the model (2)

(f) **Dropping stocks:** After having calculated the previous risk metrics for all 91 stocks, it's time to give some investing horizon consensus insights. More specifically, we define the following risk-metrics constraints which stocks should meet in order to be risk-eligible

– $sharpe \geq 0$

– $treynor \geq 0$

– $alpha > 0$

– $f - test \geq 100$

After applying these constrainsts, and short the edited *stocks_yahoofinance_edited* DataFrame descending with regard to $f - test$ **we are left with 39 stocks** [7]

```
(39, 10)
```

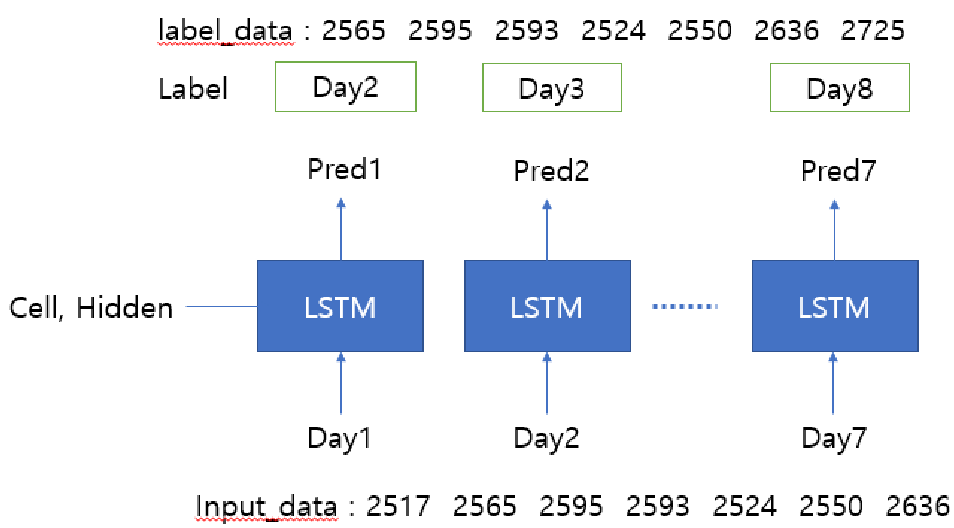
Symbol	alpha	beta	r-squared	share_ratio	treynor_ratio	f_test	1 Year	2 Year	3 Year	YTD
MSFT	19.814749	1.373946	64.113859	1.563936	0.267977	1347.089665	36.619081	42.099427	41.818563	35.910181
CSCO	11.454632	1.259557	57.955342	1.166749	0.199144	1039.331263	35.050945	38.497430	30.083349	34.366063
MPWR	3.799481	1.613388	47.494265	0.689570	0.125856	682.033608	-6.824284	16.652364	25.305480	12.435370
TXN	7.024910	1.333638	47.415005	0.847315	0.151439	679.869105	1.492692	22.091591	25.196454	20.975974
INTU	16.673324	1.223769	46.375700	1.287254	0.262849	652.078958	25.910058	39.146974	37.166676	36.021901
AAPL	11.614751	1.274979	44.646467	1.024879	0.195529	608.153341	8.577046	18.733969	29.929505	27.479941
INTC	0.243013	1.330235	40.711661	0.529017	0.086894	517.750922	-9.515439	19.814989	16.558963	1.811100
AMAT	2.311300	1.664746	39.746756	0.582717	0.111324	497.384900	-9.221590	1.043818	23.532582	33.332760
ADI	10.066974	1.257403	39.541214	0.909721	0.185387	493.130561	13.788578	21.115761	28.310638	32.544434

[e.csv]

We get our first insights (along with the info provided by *investing.com*) for our stocks: Microsoft, Cisco, Apple, Monolithic, MPWR, Texas Instruments etc... seem to be on top of risk-based fundamental analysis.

[6] Assuming 251 trading days in a year
[7] We also added annualized returns (1-Year, 3-Year, YTD)

6. **Technical analysis:** After having calculated our risk metrics, we should enhance DM'2 diversification.
   - **General about LSTM**
     * More specifically,unlike regression predictive modeling, time series also adds the complexity of a sequence dependence among the input variables.
     * A powerful type of neural network designed to handle sequence dependence is called recurrent neural networks. The **Long Short-Term Memory** network or LSTM network is a type of recurrent neural network used in deep learning because very large architectures can be successfully trained.
     *
   - The LSTM model:
     * LSTM networks can be stacked in Keras in the same way that other layer types can be stacked. One addition to the configuration that is required is that an LSTM layer prior to each subsequent LSTM layer must return the sequence. This can be done by setting the return_sequences parameter on the layer to True.
     * **Memory of the LSTM**: For example, given the current time (t) we want to predict the value at the next time in the sequence (t+1), we can use the current time (t), as well as the two prior times (t-1 and t-2) as input variables. **Here we set look_back = 4, so our LSTM we use 4 prior adjusted closed prices to predict the next one**
     * **Optimizer** We used Adam optimizer and learning rate of 0.01% (hyperparameter)
     * We used the 80% of the 756 adjusted closed prices data as a train set and we used the model to predict the rest 20%.
     * Mean Error We now have predictions for 152 days. [8]. However, for we calculate the mean error for the weeks' predictions (5 trading days) and months' predictions (22 trading days) not the absolute one, because the direction of the error is important in our occasion for a correct transformation into recommendation.



   - **Mean Error convertion into Recommendation**
     * Having calculated the array of weekly and monthly mean errors of each stocks, we now rescale all the results using the scale (-1,1) (**normalization**).
     * The final step is to translate the normalized mean error performance into recommendation (0-4), following the mapping below (equal ranges for each category)

$$\boxed{\begin{array}{c} result \leq -0.6 \longrightarrow 0 : StrongSell \\ -0.6 < result \leq -0.2 \longrightarrow 1 : Sell \\ -0.2 < result \leq 0.2 \longrightarrow 2 : Neutral \\ 0.2 < result \leq 0.6 \longrightarrow 3 : Buy \\ 0.6 < result \leq 1 \longrightarrow 4 : StrongBuy \end{array}}$$

$\boxed{\textbf{[f.csv]}}$

[8] We could have used some subsett of this train set, as a validation set to also tune hyperparameters not needed, or optimize others (like learning rate)

7. **Combining DM1 & DM2 results**

In this section we calculate score for each DM for each stock, the total score of the 2 DMs for each score (along with the average and the standard deviation score) and we exclude more stocks if they are not under a specific STD threshold s, reaching into a specific consensus

(a) After having completed and the conversion, we have in our hands 2 DMs **not normalized score**, the *stocks_investingcom* and the *stocks_yahoofinance_edited* DataFrames. Let's get a look in the latter one, since is finally created after combining appropriately the 2 DataFrames.

Symbol	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
MSFT	Microsoft	1.000000e+12	30.14	1.222100e+11	23690000.0	4.48	1.373946	35.910181	31.96	168.48	3.0	3.0
CSCO	Cisco	2.409200e+11	20.47	5.132000e+10	20060000.0	2.74	1.259557	34.366063	28.37	94.97	3.0	4.0
MPWR	Monolithic	5.570000e+09	53.67	5.945900e+08	295460.0	2.43	1.613388	12.435370	-9.41	87.19	4.0	4.0
TXN	Texas Instruments	1.046900e+11	20.16	1.559000e+10	4850000.0	5.51	1.333638	20.975974	-2.90	78.89	4.0	4.0
INTU	Intuit	6.746000e+10	41.34	6.780000e+09	1470000.0	6.25	1.223769	36.021901	21.67	142.00	2.0	2.0
AAPL	Apple	9.124400e+11	17.01	2.584900e+11	29290000.0	11.67	1.274979	27.479941	6.42	108.71	4.0	4.0
INTC	Intel	2.132400e+11	10.86	7.084000e+10	23770000.0	4.36	1.330235	1.811100	-11.75	46.66	4.0	4.0

[g.csv]

(b) **Vector Normalization** We build the customized function *normalization()*, which used to normalize each criterion values for each stock for both DM1, DM2. [9]. During this process, In order to be able to compare different kinds of criteria the first step is to make them dimensionless, i.e., eliminate the units of the criteria. In the normalized decision matrix, the normalized values of each performance x_{ij} is calculated as

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

(c) Calculate the total score for each DM. The final normalized results are:

Symbol	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly	DM1
MSFT	Microsoft	28.367598	4.451415	26.918748	4.165252	0.700526	1.025163	2.406621	17.322940	0.991374	2.119996	0.185695	88.655328
CSCO	Cisco	6.834322	3.023240	11.304068	3.527014	0.428446	0.991825	2.201296	15.377091	0.558825	2.119996	0.185695	46.551817
MPWR	Monolithic	0.158008	7.926591	0.130968	0.051949	0.379973	1.375219	0.752116	-5.100403	0.513046	0.529999	0.092848	6.810312
TXN	Texas Instruments	2.969804	2.977456	3.433952	0.852743	0.861584	1.016829	1.293698	-1.571856	0.464207	2.119996	0.185695	14.604107
INTU	Intuit	1.913678	6.105557	1.493406	0.258460	0.977296	0.900143	2.378352	11.745561	0.835560	2.119996	0.185695	28.913704

Symbol	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly	DM2
MSFT	Microsoft	28.367598	4.451415	26.918748	4.165252	0.700526	1.123230	2.457891	17.322940	0.991374	1.383797	0.131306	88.014077
CSCO	Cisco	6.834322	3.023240	11.304068	3.527014	0.428446	1.029714	2.352203	15.377091	0.558825	1.383797	0.175075	45.993794
MPWR	Monolithic	0.158008	7.926591	0.130968	0.051949	0.379973	1.318979	0.851145	-5.100403	0.513046	1.845062	0.175075	8.250392
TXN	Texas Instruments	2.969804	2.977456	3.433952	0.852743	0.861584	1.090277	1.435711	-1.571856	0.464207	1.845062	0.175075	14.534014
INTU	Intuit	1.913678	6.105557	1.493406	0.258460	0.977296	1.000457	2.465537	11.745561	0.835560	0.922531	0.087538	27.805580

(d) **Average, Total Score, Standard deviation**: Let's continue with the row-wise calculations. After creating a new *final* DataFrame which includes the score for each DM, we calculate the prementioned metrics.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

(e) **Dropping stocks:** We set the following threshold for standard deviation:

$$TH_{std} = 0.8$$

Standard deviation as the average distance of each score from the average of the 2 scores , indicates that the samples tend to be near to the average.

[9] The function has the argument *weights*, since it will also be used later on **Securities Selection** section

It is very important to mention that, the threshold defined above to excluded some stocks based on that is of utmost importance. More specifically, we prove that for stocks **below that threshold** we reached a "good" consensus, our whole previous data analysis of risk metrics and the recommendation predictions (after giving them appropriate high weights as criteria, since the rest were similar) were able to follow well the analysis of a giant source-analyst like *investing.com*

(f) After having dropped stocks with a big deviation, **we are left with 22 stocks**, for which we calculate their consensus, and finally the average whole percentage consensus. Below, we have the results sorted with regard to consensus:

```
(22, 6)
```

Symbol	DM1	DM2	Total Score	Average	St. dev	Standard Consensus
TXN	14.604107	14.534014	29.138121	14.569061	0.049563	93.804643
OTEX	19.417729	19.513887	38.931617	19.465808	0.067994	91.500772
AMAT	7.792784	7.663033	15.455817	7.727909	0.091748	88.531498
UBNT	39.425744	39.291003	78.716747	39.358374	0.095277	88.090401
AMSWA	13.476771	13.692423	27.169194	13.584597	0.152489	80.938835
TTEC	25.567020	25.329434	50.896454	25.448227	0.167999	79.000134
AVGO	20.341370	20.073766	40.415136	20.207568	0.189225	76.346897
AAPL	101.679142	101.370049	203.049190	101.524595	0.218562	72.679757
QCOM	33.416355	33.047022	66.463378	33.231689	0.261158	67.355254
XLNX	46.380164	45.966046	92.346210	46.173105	0.292826	63.396724
ESLT	29.808219	29.349819	59.158038	29.579019	0.324138	59.482776
TER	21.316604	20.817141	42.133745	21.066872	0.353173	55.853329
CSCO	46.551817	45.993794	92.545612	46.272806	0.394582	50.677288
SSNC	27.625701	28.238541	55.864242	27.932121	0.433343	45.832071
CDW	28.023398	27.391592	55.414990	27.707495	0.446755	44.155667
MSFT	88.655328	88.014077	176.669405	88.334702	0.453434	43.320799
MANT	20.523921	19.829817	40.353737	20.176869	0.490806	38.649287
GRMN	27.968396	27.191304	55.159700	27.579850	0.549487	31.314142
ADI	18.375722	17.594464	35.970186	17.985093	0.552432	30.945940
JKHY	12.181143	12.965321	25.146463	12.573232	0.554497	30.687822
BRKR	45.281044	44.439207	89.720252	44.860126	0.595269	25.591428
OLED	87.148405	86.250436	173.398841	86.699421	0.634960	20.630006

```
Final Average Consensus for DM1, DM2 is : 58.13%
```

[h.csv]

- Key Notes?
  - We worked into *colab.google()* which permits the use of GPU, something that was necessary to train the LSTMT model in less than 1 hour in data of 3 years. So, since *colab* works with sessions that terminates occasionaly we constructed a customized function *create_download_link()* to download the data of the intermediate steps in .csv form. The results are attached in the folder **end**$_{data}$
  - In the **Introduction** section we analyzed our low-risk horizon, however excluding stocks with high beta is not recommended, since we think we will get biased result, excluding stocks of high importance **eg.** Microsoft

- A reason about LSTM model was selected, is that not only tech stocks of NASDAQ follow well SP 500, but also they characterized by **trend**, which is easier to predict.
- The recommendation chart conversion from Strong Buy, Buy,... to 0,1,... , was also applied in the **investingcom** data.
- The **Group Decision** problem we resolved reaching a consensus, used normalized values of actual values for the criterion and **not estimations of importance of each criterion for each decision maker**. Therefore, we resolved a variation of the classical problem, especially when we consider as DM's 2 different analysis of actual data and not estimations of 2 physical persons
- It is worth mentioning that since our analysis limited to calculate 4/11 criteria, we decided to give greater weights on that criteria, so that we can have 2 diversified DM results, and apply a descent Group Decision Making
- We have added comments in our source code, for reasons of completeness and easier understanding of such a complex procedure followed!
- We decided to work with DataFrames and not with numpy arrays or python dictionaries, since indexing was much easier and is totally recommended for data handling. Especially, we did row and column wise calculations in one line of code.

2.2 *Securities Selection*

- <u>What can we do?</u>

Objectives

- The problem of selecting the right stocks to invest in is of immense interest for investors on both emerging and developed capital markets. Moreover, an investor should take into account all available data regarding stocks on the particular market. This includes fundamental and stock market indicators.
- In this section we will resolve the actual problem of this project, which is to reach a kernel of stock(s) which can be characterized as good alternatives for our portfolio. *More specifically, as a starting point could be considered the result from the whole previous analysis, the 22 stocks selected*
- **Using several MCDM methods often leads to divergent rankings**, a conclusion we will also reach.

Methods

a) **ELECTRE I with(out) veto**:
   - All ELECTRE methods appear to be similar in describing the concepts but differ in type of decision problem being solved. It has been proved that ELECTRE I is found to be best suited for selection problems,

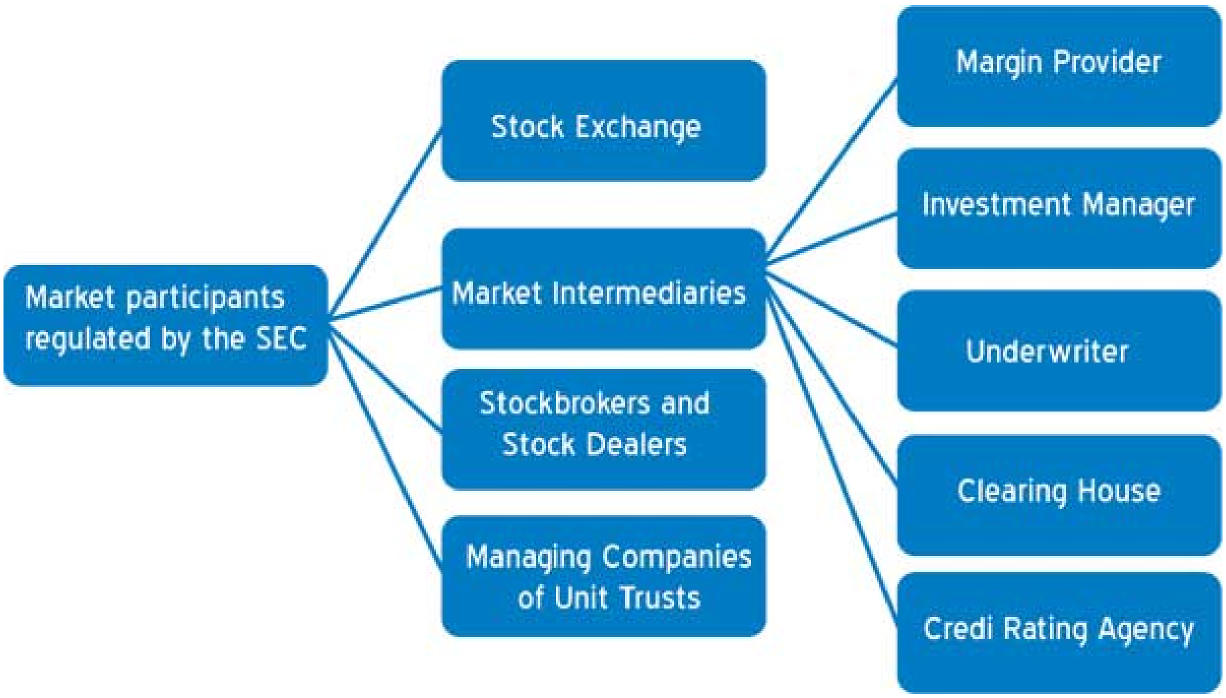
b) **TOPSIS**:
   - Technique of Order Preference Similarity to the Ideal Solution (TOPSIS) is a pretty straightforward MCDA method. As the name implies, the method is based on finding an ideal and an anti-ideal solution and comparing the distance of each one of the alternatives to those
   - The selected solutions should have the maximum (or top n maximums) distance of the worst ideal solution, in the geometric sense
   - Since, ideal and anti-ideal solutions follow monotonously decreasing function, their calculation is an easy task. Based on them we conclude in the ranking of the solutions, and we keep the top n.

c) **PROMETHEE**:
   - One of the creators of PROMETHEE, Professor Bertrand Mareschal, maintains a full list of references to his website that as of April 2017 numbered approximately 1,500 references, rendering the method to be quite popular. *Input data is similar to TOPSIS and VIKOR, but the modeler is optionally required to feed the algorithm with a couple of more variables, depending on his preference function choice*

Stakeholders' profile

Stakeholders in this type of problem are those persons, groups or institutions, i.e. actors, with a possible interest in the justice and security program.

1. **Stockbroker**: A corporate engaged in the business of buying or selling securities on behalf of investors. Yes. You have to open an account at Central Depositary System (CDS) through a stockbroker.
2. **Investment Manager** Since, we seek to resolve this problem, to a reach a kernel of 5-6 tech stocks in which we therefore invest, the role of an investment manager (if we choose to address one, since there is fee), is really important. Investment manager is a person who **for a fee or commission** engages in the business of managing a portfolio of listed securities on behalf of an investor or advises any person on the merits of investing, purchasing or selling listed securities.
3. **Clearing House**A clearing house acts as a go between for buyers and sellers by settling the buyers/seller trading accounts. Clearing house makes an equity market stable and efficient.



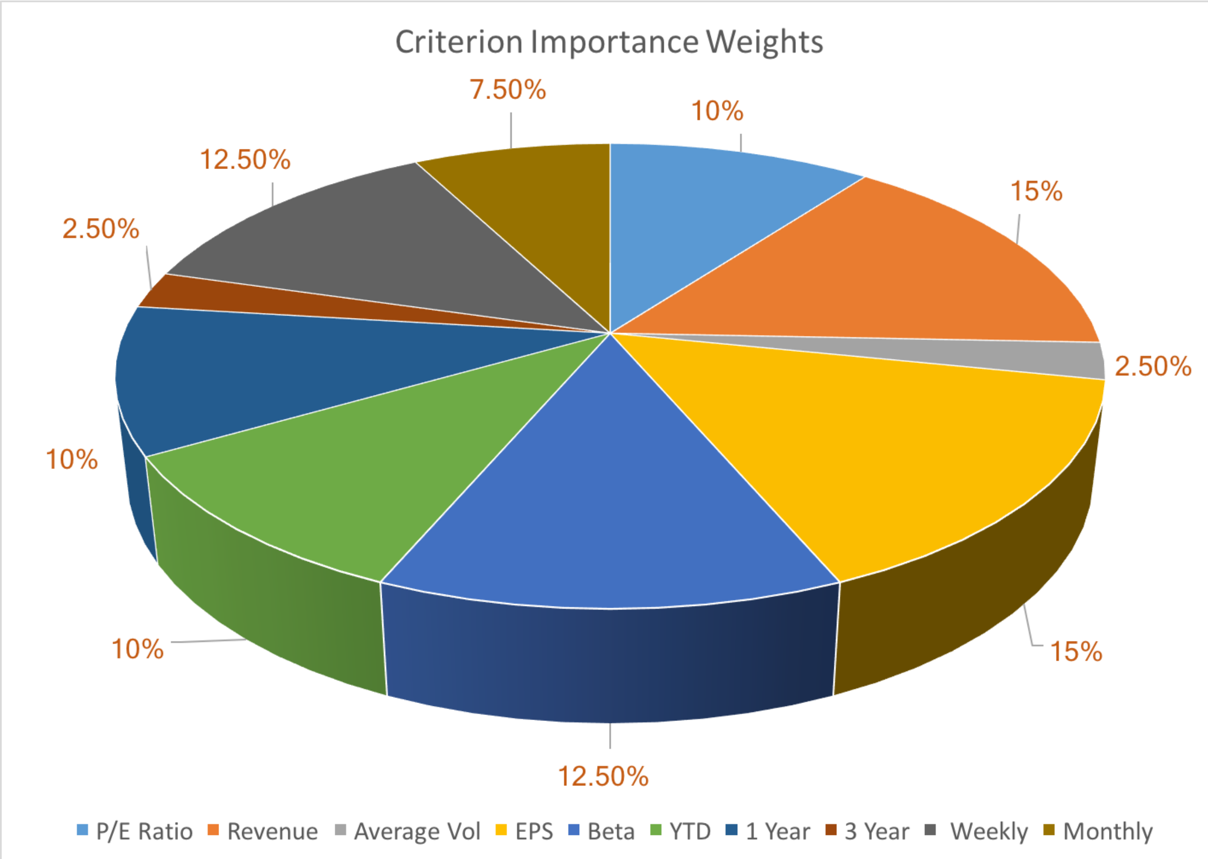
Basic Goal

Despite the following problem of technology stocks:

1. The Technology Industry is Tough to Understand
2. Planned Obsolescence Means a Loss of Competitive Advantages
3. Most Tech Companies are Too New and Unproven

our ultimate goal, is to invest in 5-6 stocks with established reputation and **trend**, especially for those who watch technology evolutions and is always informed. In that direction is important to mention that we seek stocks (with their respective weights of importance) with:

- **high** Market cap
- **low** P/E ratio
- **high** Revenue
- **Medium** Average Vol (3m).
- **high** EPS
- **low** beta
- **high** YTD
- **high** 1-Year return
- **high** 3-Year return
- **high** Weekly performance
- **high** Monthly performance



1st criterion : Market Capitalization

- **Definition**: Market capitalization refers to the total dollar market value of a company's outstanding shares. Commonly referred to as "market cap," it is calculated by multiplying a company's shares outstanding by the current market price of one share.
- Companies that have a market capitalization of between 300$million to$2 billion are generally classified as small-cap companies.
- The greatest advantage to adding large-cap stocks to an investment portfolio is the stability they can provide. Because large-cap companies are so large and have a well-established reputation with consumers, they are less likely to come across a business or economic circumstance that renders them insolvent or forces them to stop revenue-producing operations completely.
- **Goal:** (6B - 1T)

2nd criterion : P/E Ratio

- **Definition**: The price-to-earnings ratio (P/E ratio) is the ratio for valuing a company that measures its current share price relative to its per-share earnings (EPS). The price-to-earnings ratio is also sometimes known as the price multiple or the earnings multiple.
- Investors would incur less risk by investing in more-certain earnings instead of less-certain ones, so the company producing those sure earnings again commands a higher price today.
- The important is to consider what premium you are paying for a company's earnings today and determine if the expected growth warrants the premium. One way to value individual shares or the stockmarket as a whole is to compare share prices and earnings in a price-to-earnings ratio (P/E). A lower P/E ratio suggests better value.
- **Goal:** (5 - 35)

3rd criterion : Revenue

- **Definition**: Revenue is the total income earned by a company for selling its goods and services. Revenue is called the top line because it sits at the top of the income statement, which also refers to a company's gross sales. Revenue is the income generated before expenses are deducted.

- **Goal:** $>>$
- **Note:** The important is to include the revenue growth of a company and use it as criterion and so define a clear goal. The 22 NASDAQ tech stocks have high revenues. We won't apply a constraint. We only use the criterion since it is of high importance to extract a total score for a stock

4th criterion : EPS

- **Definition**: EPS is the bottom-line measure of a company's profitability and it's basically defined as net income divided by the number of outstanding shares.
- Because shareholders can't access the EPS attributed to their shares, the connection between EPS and a share's price can be difficult to define. This is particularly true for companies that pay no dividend. For example, it is common for **technology companies** to disclose in their initial public offering documents that the company does not pay a dividend and has no plans to do so in the future.
- For an investor who is primarily interested in a steady source of income, the EPS ratio can tell him/her the room a company has for increasing its existing dividend. Even if, EPS is very important and essential tool for investors, it should not be looked at in isolation. EPS of a company should always be considered in relation to other companies in order to make a more informed investment decision.
- **EPS is typically considered good when a corporation's profits outperform those of similar companies in the same sector**
- **Goal:** > 2.5
- **Note:** If we didn't deal with stocks of the same sector, the goal would be let undefined

5th criterion : Average Vol (3m)

- **Definition**: This is the daily average of the cumulative trading volume during the last three months.
- Thinly traded stocks tend to be extremely speculative and unpredictable. Because there is such a limited number of shares, a large purchase by a mutual fund or another big investor can cause a huge spike in the price.
- If the investor decides to sell, the share price will likely tank. Neither scenario is ideal for individual investors. That's why IBD considers a stock that trades fewer than 400,000 shares per day, based on a 50-day average, as thinly traded.
- **Goal:** > 500000

6th criterion : Beta

- **Definition**: A beta coefficient is a measure of the volatility, or systematic risk, of an individual stock in comparison to the unsystematic risk of the entire market. In statistical terms, beta represents the slope of the line through a regression of data points from an individual stock's returns against those of the market.
- Mega-cap tech stocks are being added to more and more passive index products that have very different investment objectives. Apple, Alphabet and Microsoft are currently held in a number of low-volatility and momentum offerings, according to a recent Wall Street Journal article.
- Based on the previous observation it is worth mentioning **we chose NASDAQ tech stocks to analyze because of their stability. Generally companies without a solid business are risky. But the most of these Large Cap companies are less riskier than other tech companies, eg. startups with amazing growth and huge volatility** [10]
- Although small-cap stocks are considered riskier investments than large-cap stocks, there are enough small-cap stocks offering excellent growth potential and high potential returns on equity to warrant their inclusion in the holdings of all but the most conservative investors. **We are not willing to take on big amoung of risk, so we focus on stable tech stocks.**
- textbfGoal: (0.6 - 1.4)

7th criterion : YTD, 1-Year, 3-Year

[10] Rapid adoption of AI, cloud, IoT, autonomous cars, wearables, virtual reality/augmented reality devices presents massive growth opportunity. Moreover, increasing video streaming has been driving user engagement that is, in turn, attracting advertising dollars. Additionally, growing demand for smart speakers and connected devices, which are powered by AI, machine learning and deep learning, is a key catalyst. Furthermore, the accelerated deployment of 5G technology is encouraging.

- **Definition**: Year to date (YTD) refers to the period beginning the first day of the current calendar year or fiscal year up to the current date. YTD information is useful for analyzing business trends or comparing performance data, and the acronym often modifies concepts such as investment returns, earnings and net pay.
- **Goal:** > 25
- **Note:** We deal with large cap tech stocks, so we expect them to outperform the market in a sttandard based to invest in

8th criterion : Weekly, Monthly recommendations

- **Definition**: In order to reach an opinion and communicate the value and volatility of a covered security, analysts research public financial statements, listen in on conference calls and talk to managers and the customers of a company, typically in an attempt to come up with findings for a research report.
- **Should an investor react accordingly to new analyst's recommendations and adjust a position based on the analyst's rating alone? Of course not. The research report and subsequent rating should be used to complement individual homework and strategy.**
- **Goal:** {3,4}
- **Note:** Since we would like to invest and not sell stocks, it is of high importance to invest in tech stocks that are considered as **'Strong Buy'** or at least **'Buy'**

Tech Stocks

We present the information of the selected 22 stocks (alternatives), in which we will apply the 3 methods.

Symbol	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
TXN	Texas Instruments	1.046900e+11	20.16	1.559000e+10	4850000.0	5.51	1.22	17.39	-2.90	78.89	4	4
OTEX	Open Text	1.110000e+10	40.51	2.880000e+09	499120.0	1.02	0.48	25.43	13.71	35.62	4	4
AMAT	Applied Materials	3.971000e+10	11.86	1.577000e+10	9230000.0	3.57	1.63	29.23	-12.91	79.05	4	3
UBNT	Ubiquiti	9.220000e+09	29.86	1.140000e+09	449380.0	4.44	1.34	30.95	51.18	225.04	2	4
AMSWA	American Software	4.432200e+08	73.81	1.117900e+08	66460.0	0.19	0.69	36.17	-6.13	42.16	4	4
TTEC	TTEC	2.070000e+09	41.87	1.530000e+09	84980.0	1.08	0.68	57.51	20.81	63.46	4	4
AVGO	Broadcom	1.102900e+11	33.95	2.131000e+10	2790000.0	8.19	0.91	7.86	4.99	74.77	2	4
AAPL	Apple	9.124400e+11	17.01	2.584900e+11	29290000.0	11.67	1.23	25.83	6.42	108.71	4	4
QCOM	Qualcomm	8.680000e+10	39.71	2.123000e+10	21120000.0	1.81	1.61	24.78	20.79	31.77	3	4
XLNX	Xilinx	2.829000e+10	32.81	3.060000e+09	4360000.0	3.41	1.22	30.29	61.67	138.65	2	4
ESLT	Elbit Systems	6.770000e+09	32.17	4.710000e+09	12680.0	4.85	0.83	38.38	30.99	70.87	4	4
TER	Teradyne	7.930000e+09	20.14	2.110000e+09	2160000.0	2.27	1.55	45.92	16.81	134.70	4	4
CSCO	Cisco	2.409200e+11	20.47	5.132000e+10	20060000.0	2.74	1.19	29.59	28.37	94.97	4	4
SSNC	SS&Cs	1.472000e+10	117.98	4.140000e+09	1360000.0	0.49	1.29	28.77	6.49	98.06	2	4
CDW	CDW Corp	1.534000e+10	23.78	1.659000e+10	684910.0	4.38	1.05	29.56	24.58	155.31	4	4
MSFT	Microsoft	1.000000e+12	30.14	1.222100e+11	23690000.0	4.48	1.23	32.35	31.96	168.48	4	4
MANT	ManTech	2.570000e+09	30.85	1.990000e+09	125000.0	2.08	0.94	23.35	18.13	77.52	4	4
GRMN	Garmin	1.532000e+10	21.80	3.400000e+09	1070000.0	3.71	0.98	27.62	31.53	90.05	4	4
ADI	Analog Devices	4.070000e+10	26.19	6.240000e+09	2780000.0	4.20	1.40	27.95	8.73	93.38	4	4
JKHY	Jack Henry&Associates	1.059000e+10	37.55	1.580000e+09	385560.0	3.66	0.94	8.20	4.39	63.66	2	3
BRKR	Bruker	7.410000e+09	40.46	1.930000e+09	811410.0	1.17	1.27	58.25	55.53	93.23	4	4
OLED	Universal Display	8.680000e+09	105.25	3.351800e+08	732430.0	1.75	1.51	96.59	109.15	164.87	4	4

Method 1 : ELECTRE I

1) Construct agreement matrix which will be used in both ELECTRE.

$$C(a,b) = \frac{1}{W} \sum_{g_j(a)>g_j(b)} w_j$$

$$W = \sum_{j=1}^{n} w_j$$

$$0 \leq C \leq 1$$

Symbol	TXN	OTEX	AMAT	UBNT	AMSWA	TTEC	AVGO	AAPL	QCOM	XLNX	ESLT	TER	CSCO	SSNC	CDW	MSFT	MANT	GRMN	ADI	JKHY	BRKR	OLED
Symbol																						
TXN	1	0.7	0.575	0.55	0.8	0.7	0.475	0.3	0.4	0.675	0.7	0.65	0.475	0.55	0.525	0.35	0.7	0.675	0.55	0.8	0.55	0.55
OTEX	0.5	1	0.4	0.5	0.65	0.4	0.5	0.4	0.425	0.3	0.35	0.475	0.3	0.45	0.3	0.3	0.6	0.3	0.4	0.7	0.475	0.375
AMAT	0.55	0.725	1	0.45	0.625	0.625	0.4	0.35	0.525	0.6	0.475	0.6	0.4	0.7	0.3	0.25	0.725	0.55	0.525	0.65	0.6	0.6
UBNT	0.525	0.575	0.55	1	0.675	0.525	0.55	0.525	0.45	0.6	0.375	0.475	0.675	0.7	0.675	0.325	0.625	0.675	0.55	0.725	0.4	0.425
AMSWA	0.4	0.55	0.5	0.4	1	0.425	0.4	0.4	0.425	0.4	0.325	0.3	0.4	0.3	0.4	0.4	0.4	0.4	0.4	0.4	0.3	0.2
TTEC	0.5	0.8	0.5	0.55	0.775	1	0.5	0.5	0.525	0.4	0.425	0.5	0.4	0.55	0.4	0.4	0.5	0.4	0.5	0.5	0.3	0.35
AVGO	0.6	0.575	0.6	0.65	0.675	0.575	1	0.175	0.425	0.625	0.675	0.525	0.325	0.55	0.525	0.325	0.525	0.525	0.525	0.675	0.425	0.425
AAPL	0.9	0.8	0.775	0.55	0.8	0.7	0.9	1	0.675	0.675	0.7	0.55	0.7	0.575	0.675	0.65	0.8	0.7	0.575	0.9	0.575	0.55
QCOM	0.675	0.65	0.475	0.625	0.65	0.55	0.65	0.4	1	0.625	0.5	0.6	0.325	0.775	0.5	0.3	0.7	0.5	0.6	0.825	0.55	0.55
XLNX	0.525	0.775	0.4	0.6	0.675	0.675	0.575	0.4	0.45	1	0.475	0.65	0.675	0.625	0.55	0.275	0.875	0.575	0.425	0.75	0.55	0.425
ESLT	0.5	0.85	0.65	0.7	0.875	0.775	0.4	0.5	0.575	0.6	1	0.7	0.65	0.7	0.65	0.55	0.825	0.7	0.65	0.725	0.5	0.5
TER	0.55	0.725	0.525	0.6	0.9	0.7	0.55	0.65	0.475	0.425	0.5	1	0.45	0.725	0.45	0.425	0.8	0.475	0.55	0.725	0.7	0.65
CSCO	0.725	0.9	0.725	0.4	0.8	0.8	0.75	0.5	0.75	0.4	0.55	0.75	1	0.75	0.725	0.2	0.9	0.65	0.625	0.75	0.575	0.55
SSNC	0.525	0.625	0.3	0.5	0.775	0.525	0.65	0.5	0.3	0.575	0.375	0.35	0.325	1	0.325	0.3	0.625	0.6	0.3	0.85	0.525	0.375
CDW	0.675	0.9	0.825	0.4	0.8	0.8	0.55	0.525	0.575	0.525	0.55	0.75	0.475	0.75	1	0.2	0.9	0.875	0.725	0.9	0.55	0.525
MSFT	0.85	0.9	0.875	0.75	0.8	0.8	0.75	0.675	0.775	0.8	0.65	0.775	1	0.775	1	1	0.9	1	0.875	0.9	0.575	0.575
MANT	0.5	0.6	0.4	0.45	0.8	0.7	0.55	0.4	0.375	0.2	0.375	0.4	0.3	0.45	0.3	0.3	1	0.3	0.4	0.7	0.5	0.5
GRMN	0.525	0.9	0.575	0.4	0.8	0.8	0.55	0.5	0.575	0.5	0.5	0.725	0.55	0.475	0.325	0.2	0.9	1	0.3	0.9	0.55	0.55
ADI	0.65	0.8	0.6	0.525	0.8	0.7	0.55	0.625	0.475	0.65	0.55	0.65	0.575	0.775	0.475	0.325	0.8	0.9	1	0.9	0.7	0.55
JKHY	0.2	0.3	0.425	0.4	0.6	0.5	0.45	0.1	0.175	0.375	0.275	0.275	0.25	0.275	0.1	0.1	0.425	0.1	0.1	1	0.175	0.325
BRKR	0.65	0.725	0.525	0.675	0.9	0.9	0.65	0.625	0.525	0.525	0.7	0.5	0.625	0.55	0.65	0.625	0.7	0.65	0.5	0.825	1	0.375
OLED	0.65	0.825	0.525	0.65	1	0.85	0.65	0.65	0.525	0.65	0.7	0.55	0.65	0.7	0.675	0.625	0.7	0.65	0.65	0.675	0.825	1

2a) Construct disagreement matrix for ELECTRE I without veto

$$D(a,b) = 0 \text{ if } g_j(a) \geq g_j(b), \forall j$$

$$D(a,b) = \frac{\delta}{max[g_j(b) - g_j(a)]}$$

$$\delta = max[g_j(c) - g_j(d)]$$

$$0 \leq D \leq 1$$

Symbol	TXN	OTEX	AMAT	UBNT	AMSWA	TTEC	AVGO	AAPL	QCOM	XLNX	ESLT	TER	CSCO	SSNC	CDW	MSFT	MANT	GRMN	ADI	JKHY	BRKR	OLED
Symbol																						
TXN	-6.00266e-13	5.35237e-12	-1.90084e-13	6.61793e-11	3.86671e-11	1.51267e-11	-9.10404e-13	1.16051e-12	4.55202e-12	1.95787e-11	-9.90439e-13	3.53157e-12	-6.30279e-13	8.28567e-11	-7.70341e-13	9.59425e-12	-8.8039e-13	-8.40372e-13	-4.20186e-13	2.39106e-12	1.5867e-11	7.01211e-11
OTEX	1.40062e-13	-6.00266e-13	5.50244e-13	1.09469e-10	1.83081e-11	7.08314e-12	2.17096e-12	5.65251e-12	5.30235e-13	2.30402e-11	-2.50111e-13	1.90885e-11	1.10049e-13	6.24977e-11	3.97076e-11	5.28834e-11	-1.40062e-13	-1.00044e-13	3.20142e-13	-1.40062e-13	7.82347e-12	5.04624e-11
AMAT	-1.01045e-12	1.36561e-11	-6.00266e-13	6.60193e-11	4.69708e-11	1.50167e-11	7.09314e-12	3.10137e-12	1.28557e-11	2.95931e-11	5.31235e-12	-6.80302e-13	-1.04046e-12	9.11604e-11	-1.18052e-12	9.43418e-12	3.99177e-12	-5.60248e-13	-6.70297e-13	1.06947e-11	2.34504e-11	7.84248e-11
UBNT	-7.20319e-13	-1.46065e-12	-3.10137e-13	-6.00266e-13	2.89628e-11	1.56069e-12	-1.03046e-12	2.23099e-12	-3.30146e-13	-7.20319e-13	-1.11049e-12	-3.90173e-13	-7.50333e-13	7.31524e-11	-8.90395e-13	-7.10315e-13	-1.00044e-12	-9.60426e-13	-5.40239e-13	-1.00044e-12	2.30102e-12	6.04168e-11
AMSWA	3.20142e-13	-8.10359e-13	3.40151e-13	1.02926e-10	-6.00266e-13	-6.1027e-13	3.00133e-12	6.48287e-12	3.20142e-13	2.28101e-11	-3.40151e-13	1.25456e-11	-1.00044e-13	2.91829e-11	3.31647e-11	4.63405e-11	-3.50155e-13	-3.10137e-13	1.10049e-13	-3.50155e-13	1.66674e-11	7.03112e-11
TTEC	-6.00266e-14	-8.00355e-13	3.50155e-13	8.16162e-11	1.69475e-11	-6.00266e-13	2.11094e-12	5.59248e-12	3.30146e-13	-6.00266e-14	-4.502e-13	2.7012e-13	-9.00399e-14	6.11371e-11	1.18553e-11	2.50311e-11	-3.40151e-13	-3.00133e-13	1.20053e-13	-3.40151e-13	-1.00044e-14	4.84015e-11
AVGO	-2.90129e-13	-1.03046e-12	1.20053e-13	7.03012e-11	2.4871e-11	2.46609e-11	-6.00266e-13	-2.80124e-13	1.00044e-13	1.16852e-11	5.52245e-12	1.30658e-11	-3.20142e-13	6.90606e-11	5.40239e-13	1.37161e-11	-5.70253e-13	-5.30235e-13	-1.10049e-13	-5.70253e-13	2.54013e-11	6.37583e-11
AAPL	-6.1027e-13	8.50377e-12	-2.00089e-13	3.63461e-11	4.18185e-11	9.86437e-12	1.94086e-12	-6.00266e-13	7.70341e-12	1.02545e-11	1.60071e-13	-2.80124e-13	-6.40284e-13	8.60081e-11	-7.80346e-13	-6.00266e-13	-8.90395e-13	-8.50377e-13	-4.30191e-13	5.54246e-12	8.45375e-12	7.32725e-11
QCOM	-9.90439e-13	-1.73077e-12	-5.80257e-13	1.1332e-10	1.91085e-11	7.73343e-12	1.38061e-12	4.86216e-12	-6.00266e-13	2.68919e-11	-1.38061e-12	2.29402e-11	-1.02045e-12	6.32981e-11	4.35593e-11	5.67351e-11	-1.27056e-12	-1.23055e-12	-8.10359e-13	-1.27056e-12	8.47376e-12	5.31235e-11
XLNX	-6.00266e-13	-1.34059e-12	-1.90084e-13	6.39283e-12	2.60115e-11	2.22098e-12	-2.20098e-13	3.26145e-12	-2.10093e-13	-6.00266e-13	-9.90439e-13	-2.7012e-13	-6.30279e-13	7.02011e-11	-7.70341e-13	-5.90262e-13	-8.8039e-13	-8.40372e-13	-4.20186e-13	-8.8039e-13	2.96131e-12	5.74655e-11
ESLT	-2.10093e-13	-9.50421e-13	2.00089e-13	7.42029e-11	2.66518e-11	-7.50333e-13	-5.20231e-13	1.82081e-12	1.8008e-13	-2.10093e-13	-6.00266e-13	1.20053e-13	-2.40106e-13	7.08414e-11	4.44197e-12	1.76178e-11	-4.90217e-13	-4.502e-13	-3.00133e-14	-4.90217e-13	-1.60071e-13	5.81058e-11
TER	-9.30412e-13	5.37238e-12	-5.20231e-13	1.03446e-11	3.86871e-11	6.73298e-12	9.20408e-13	4.40195e-12	4.57203e-12	-1.40062e-13	-1.32059e-12	-6.00266e-13	-9.60426e-13	8.28767e-11	-1.10049e-12	-9.20408e-13	-1.21054e-12	-1.17052e-12	-7.50333e-13	2.41107e-12	5.32236e-12	7.01411e-11
CSCO	-5.70253e-13	5.04223e-12	-1.60071e-13	5.00922e-11	3.8357e-11	6.40284e-12	4.502e-13	3.93174e-12	4.24188e-12	-5.70253e-13	-9.60426e-13	-2.40106e-13	-6.00266e-13	8.25466e-11	-7.40328e-13	-5.60248e-13	-8.50377e-13	-8.10359e-13	-3.90173e-13	2.08092e-12	4.99221e-12	6.98109e-11
SSNC	2.00089e-14	-1.41063e-12	-2.60115e-13	4.70008e-11	-1.20053e-12	3.74166e-12	2.7012e-12	6.18274e-12	-2.80124e-13	1.01845e-11	-6.40284e-13	-3.40151e-13	-7.0031e-13	-6.00266e-13	-8.40372e-13	-6.60293e-13	-9.50421e-13	-9.10404e-13	-4.90217e-13	-9.50421e-13	4.48199e-12	5.76856e-11
CDW	-4.30191e-13	1.73077e-12	-2.00089e-14	-3.10137e-13	3.50455e-11	3.09137e-12	-7.40328e-13	2.29102e-12	9.30412e-13	-4.30191e-13	-8.20364e-13	-1.00044e-13	-4.60204e-13	7.92351e-11	-6.00266e-13	-4.20186e-13	-7.10315e-13	-6.70297e-13	-2.50111e-13	-7.10315e-13	3.69164e-12	6.64995e-11
MSFT	-6.1027e-13	-1.3506e-12	-2.00089e-13	-4.90217e-13	2.86827e-11	1.60071e-13	-9.20408e-13	2.19097e-12	-2.20098e-13	-6.1027e-13	-1.00044e-12	-2.80124e-13	-6.40284e-13	7.28723e-11	-7.80346e-13	-6.00266e-13	-8.90395e-13	-8.50377e-13	-4.30191e-13	-8.90395e-13	9.00399e-13	6.01367e-11
MANT	-3.20142e-13	-1.06047e-12	9.00399e-14	6.75499e-11	2.79724e-11	9.16406e-12	1.11049e-12	4.59204e-12	7.0031e-14	-3.20142e-13	-7.10315e-13	1.00044e-14	-3.50155e-13	7.2162e-11	-4.90217e-13	1.09649e-11	-6.00266e-13	-5.60248e-13	-1.40062e-13	-6.00266e-13	9.90439e-12	5.94263e-11
GRMN	-3.6016e-13	3.71165e-12	5.00222e-14	5.50144e-11	3.70264e-11	5.07225e-12	-5.20231e-13	2.96131e-12	2.91129e-12	-3.6016e-13	-7.50333e-13	-3.00133e-14	-3.90173e-13	8.1216e-11	-5.30235e-13	-3.50155e-13	-6.40284e-13	-6.00266e-13	-1.8008e-13	7.50333e-13	5.6325e-12	6.84804e-11
ADI	-7.80346e-13	-6.80302e-13	-3.70164e-13	5.16829e-11	3.26345e-11	4.56202e-12	-1.01045e-12	2.4711e-12	-3.90173e-13	7.94352e-12	-1.17052e-12	-4.502e-13	-8.10359e-13	7.6824e-11	-9.50421e-13	-7.70341e-13	-1.06047e-12	-1.02045e-12	-6.00266e-13	-1.06047e-12	5.30235e-12	6.40884e-11
JKHY	-3.20142e-13	-1.06047e-12	9.00399e-14	8.14161e-11	2.12694e-11	2.43208e-11	-4.70208e-13	3.01133e-12	7.0031e-14	1.22854e-11	5.1823e-12	1.27256e-11	-3.50155e-13	6.5459e-11	1.16552e-11	2.4831e-11	-6.00266e-13	-5.60248e-13	-1.40062e-13	-6.00266e-13	2.50611e-11	6.34181e-11
BRKR	-6.50288e-13	-1.39062e-12	-2.40106e-13	5.1833e-11	1.83581e-11	-1.19053e-12	2.0209e-12	5.50244e-12	-2.60115e-13	-6.50288e-13	-1.04046e-12	-3.20142e-13	-6.80302e-13	6.25477e-11	-8.20364e-13	-6.40284e-13	-9.30412e-13	-8.90395e-13	-4.70208e-13	-9.30412e-13	-6.00266e-13	4.98121e-11
OLED	-8.90395e-13	-1.63072e-12	-4.80213e-13	-7.70341e-13	-1.42063e-12	-1.43063e-12	1.44064e-12	4.92218e-12	-5.00222e-13	-8.90395e-13	-1.28057e-12	-5.60248e-13	-9.20408e-13	-8.20364e-13	-1.06047e-12	-8.8039e-13	-1.17052e-12	-1.1305e-12	-7.10315e-13	-1.17052e-12	-8.40372e-13	-6.00266e-13

2b) Construct disagreement matrix for ELECTRE I without veto

$$D_k(a,b) = \begin{cases} 0 \text{ if } g_{jk}(b) - g_{jk}(a) < v_k \\ 1 \text{ if } g_{jk}(b) - g_{jk}(a) \geq v_k \end{cases}$$

The veto used where the following:

Market Cap	∞
P/E Ratio	15
Revenue	∞
Average Vol (3m.)	∞
EPS	5
Beta	0.6
YTD	25
1-Year Return	45
3-Year Return	80
Weekly performance	∞
Monthly performance	∞

Symbol	TXN	OTEX	AMAT	UBNT	AMSWA	TTEC	AVGO	AAPL	QCOM	XLNX	ESLT	TER	CSCO	SSNC	CDW	MSFT	MANT	GRMN	ADI	JKHY	BRKR	OLED
Symbol																						
TXN	0	1	0	1	1	1	0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1
OTEX	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	0	1	1
AMAT	0	1	0	1	1	1	1	1	1	1	1	0	0	1	0	1	1	0	0	1	1	1
UBNT	0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1
AMSWA	1	0	1	1	0	0	1	1	1	1	0	1	0	1	1	1	0	0	1	0	1	1
TTEC	0	0	1	1	1	0	1	1	1	0	0	1	0	1	1	1	0	0	1	0	0	1
AVGO	0	0	1	1	1	1	0	0	1	1	1	1	0	1	1	1	0	0	0	0	1	1
AAPL	0	1	0	1	1	1	1	0	1	1	1	0	0	1	0	0	0	0	0	1	1	1
QCOM	0	0	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	0	0	0	1	1
XLNX	0	0	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1
ESLT	0	0	1	1	1	0	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	1
TER	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0	1	1	1
CSCO	0	1	0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0	1	1	1
SSNC	1	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1
CDW	0	1	0	0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1
MSFT	0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1
MANT	0	0	1	1	1	1	1	1	1	0	0	1	0	1	0	1	0	0	0	0	1	1
GRMN	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	1	1	1
ADI	0	0	0	1	1	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	1
JKHY	0	0	1	1	1	1	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
BRKR	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1
OLED	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3a) Construct kernel for the ELECTRE I without veto based on the the following superiority relationship

$$\boxed{aSb \Longleftrightarrow C(a,b) \geq \hat{c} \text{ and } D(a,b) \leq \hat{d}}$$

We started with initial values

$$\hat{c} = 1.0, \hat{d} = 0.0$$

and in each iteration we were "relaxing" the parameters by 0.01 (decreasing the $\hat{c}$ and increasing the $\tilde{d}$). Initially the kernel includes all he 22 stocks. For the iterations in which, stock(s) are removed from the kernel, we save these $\hat{c}, \hat{d}$ that led to that removal, while we always check to break if the kernels has not more than 7 stocks.

```
Electre I withtout veto : We will invest in the following companies :
['Ubiquiti', 'Apple', 'Elbit Systems', 'Microsoft', 'Universal Display']
```

while:

Finally we present the results for these stocks selected by this method:

$\hat{c}$	$\hat{d}$	**Kernel**
1.0	0.0	['TXN', 'OTEX', 'AMAT', 'UBNT', 'TTEC', 'AVGO', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'MANT', 'ADI', 'JKHY', 'BRKR', 'OLED']
0.899	0.099	['AMAT', 'UBNT', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'ADI', 'BRKR', 'OLED']
0.869	0.129	['UBNT', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'BRKR', 'OLED']
0.819	0.18	['UBNT', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'OLED']
0.799	0.2	['UBNT', 'AAPL', 'QCOM', 'ESLT', 'TER', 'SSNC', 'MSFT', 'OLED']
0.769	0.23	['UBNT', 'AAPL', 'ESLT', 'MSFT', 'OLED']

Table 1: Relaxation process

Symbol	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
UBNT	Ubiquiti	9.220000e+09	29.86	1.140000e+09	449380.0	4.44	1.34	30.95	51.18	225.04	2	4
AAPL	Apple	9.124400e+11	17.01	2.584900e+11	29290000.0	11.67	1.23	25.83	6.42	108.71	4	4
ESLT	Elbit Systems	6.770000e+09	32.17	4.710000e+09	12680.0	4.85	0.83	38.38	30.99	70.87	4	4
MSFT	Microsoft	1.000000e+12	30.14	1.222100e+11	23690000.0	4.48	1.23	32.35	31.96	168.48	4	4
OLED	Universal Display	8.680000e+09	105.25	3.351800e+08	732430.0	1.75	1.51	96.59	109.15	164.87	4	4

3b) Construct kernel ELECTRE I with veto based on the the following superiority relationship

$$\boxed{aSb \iff C(a,b) \geq s \text{ and } D_k(a,b) = 0, \forall k}$$

We started with initial value

$$s = 1.0$$

and in each iteration we were "relaxing" the parameter by 0.01 Initially the kernel includes all he 22 stocks. For the iterations in which, stock(s) are removed from the kernel, we save these s that led to that removal, while we always check to break if the kernels has not more than 7 stocks.

```
Electre with veto : We will invest in the following companies :
['Apple', 'Elbit Systems', 'SS&Cs', 'Microsoft', 'Universal Display']
```

while:

Finally we present the results for these stocks selected by this method:

Symbol	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
AAPL	Apple	9.124400e+11	17.01	2.584900e+11	29290000.0	11.67	1.23	25.83	6.42	108.71	4	4
ESLT	Elbit Systems	6.770000e+09	32.17	4.710000e+09	12680.0	4.85	0.83	38.38	30.99	70.87	4	4
SSNC	SS&Cs	1.472000e+10	117.98	4.140000e+09	1360000.0	0.49	1.29	28.77	6.49	98.06	2	4
MSFT	Microsoft	1.000000e+12	30.14	1.222100e+11	23690000.0	4.48	1.23	32.35	31.96	168.48	4	4
OLED	Universal Display	8.680000e+09	105.25	3.351800e+08	732430.0	1.75	1.51	96.59	109.15	164.87	4	4

Method 2 : TOPSIS

TOPSIS method are used to derive the closeness coefficient and the outranking index of each stock, respectively. Based on the closeness coefficient, the outranking index, and selection threshold, we can easily obtain three type of the investment ratio in accordance with different investment preference of final decision-maker. It is a reasonable way in real decision environment

1. Use of the normalized stocks data, the *stocks_normalized* DataFrame.

s	Kernel
1.0	['TXN', 'OTEX', 'AMAT', 'UBNT', 'TTEC', 'AVGO', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'MANT', 'ADI', 'JKHY', 'BRKR', 'OLED']
0.899	['AMAT', 'UBNT', 'AVGO', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'ADI', 'BRKR', 'OLED']
0.869	['UBNT', 'AVGO', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'BRKR', 'OLED']
0.819	['UBNT', 'AVGO', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'OLED']
0.799	['UBNT', 'AVGO', 'AAPL', 'QCOM', 'ESLT', 'TER', 'SSNC', 'MSFT', 'OLED']
0.769	['UBNT', 'AVGO', 'AAPL', 'ESLT', 'SSNC', 'MSFT', 'OLED']
0.749	['AAPL', 'ESLT', 'SSNC', 'MSFT', 'OLED']

Table 2: Relaxation process

2. **Determination of the Ideal (Zenith) and Anti-ideal (Nadir) Solutions**

$$A^* = \{v_1^*, v_2^*, ..., v_n^*\} = \{(max_j v_{ij}|i \in I'), (min_j v_{ij}|i \in I''\}, \quad i = 1, 2, ..., m \; j = 1, ..., n$$

$$A^- = \{v_1^-, v_2^-, ..., v_n^-\} = \{(max_j v_{ij}|i \in I'), (min_j v_{ij}|i \in I''\}, \quad i = 1, 2, ..., m \; j = 1, ..., n$$

3. This step is about the calculation of the distances of each alternative from the ideal solution as

$$D_i^* = \sqrt{\sum_{j=1}^{n} (v_{ij} - v_j^*)^2}$$

Similarly, the distances from the anti-ideal solution are calculated as

$$D_i^- = \sqrt{\sum_{j=1}^{n} (v_{ij} - v_j^-)^2}$$

4. The relative closeness C_i^* is always between 0 and 1 and an alternative is best when it is closer to 1. It is calculated for each alternative and is defined as

$$C_i^* = \frac{D_i^-}{D_i^* - D_i^-}$$

Symbol	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly	D_plus	D_minus	C_closeness
TXN	0.754252	0.614305	2.124190	0.657164	2.170438	2.194766	0.976380	-0.694651	1.226219	3.159194	0.217571	43.686894	4.554020	0.094402
OTEX	0.079971	1.234400	0.392410	0.067630	0.401787	0.863515	1.427795	3.284022	0.553656	3.159194	0.217571	43.054234	6.719751	0.135005
AMAT	0.286096	0.361392	2.148715	1.250645	1.406255	2.932352	1.641150	-3.092394	1.228706	3.159194	0.163178	45.209548	4.088907	0.082942
UBNT	0.066427	0.909879	0.155329	0.060890	1.748956	2.410645	1.737721	12.259392	3.497887	1.579597	0.217571	38.990413	15.871662	0.289301
AMSWA	0.003193	2.249101	0.015232	0.009005	0.074843	1.241302	2.030803	-1.468348	0.655310	3.159194	0.217571	45.979810	3.375561	0.068393
TTEC	0.014914	1.275842	0.208468	0.011515	0.425422	1.223312	3.228961	4.984719	0.986384	3.159194	0.217571	42.163467	8.767989	0.172153
AVGO	0.794598	1.034507	2.903559	0.378039	3.226114	1.637080	0.441308	1.195279	1.162180	1.579597	0.217571	41.996721	6.239085	0.129346
AAPL	6.573785	0.518320	35.220131	3.968731	4.596918	2.212756	1.450253	1.537813	1.689723	3.159194	0.217571	25.198411	36.701146	0.592915
QCOM	0.625361	1.210023	2.892659	2.861714	0.712975	2.896372	1.391300	4.979929	0.493814	2.369396	0.217571	39.802336	9.423257	0.191430
XLNX	0.203819	0.999770	0.416935	0.590770	1.343230	2.194766	1.700665	14.772112	2.155092	1.579597	0.217571	37.908481	18.105980	0.323238
ESLT	0.048775	0.980268	0.641753	0.001718	1.910458	1.493161	2.154886	7.423184	1.101561	3.159194	0.217571	40.566877	10.996578	0.213263
TER	0.057133	0.613696	0.287495	0.292675	0.894173	2.788433	2.578228	4.026580	2.093696	3.159194	0.217571	42.506566	8.056419	0.159334
CSCO	1.735738	0.623752	6.992522	2.718086	1.079311	2.140797	1.661362	6.795603	1.476157	3.159194	0.217571	35.252597	12.825610	0.266766
SSNC	0.106052	3.595028	0.564089	0.184277	0.193015	2.320696	1.615322	1.554581	1.524186	1.579597	0.217571	43.716260	6.080806	0.122112
CDW	0.110519	0.724612	2.260443	0.092804	1.725321	1.888938	1.659678	5.887766	2.414045	3.159194	0.217571	39.937041	9.863132	0.198054
MSFT	7.204622	0.918411	16.651523	3.209943	1.764712	2.212756	1.816325	7.655533	2.618752	3.159194	0.217571	26.771686	21.640885	0.447010
MANT	0.018516	0.940046	0.271144	0.016937	0.819331	1.691050	1.311011	4.342766	1.204924	3.159194	0.217571	42.533080	7.789842	0.154797
GRMN	0.110375	0.664279	0.463261	0.144983	1.461402	1.763009	1.550754	7.552533	1.399683	3.159194	0.217571	40.716871	10.996370	0.212641
ADI	0.293228	0.798049	0.850221	0.376684	1.654418	2.518584	1.569283	2.091139	1.451443	3.159194	0.217571	43.084251	6.156573	0.125030
JKHY	0.076297	1.144205	0.215280	0.052243	1.441707	1.691050	0.460398	1.051558	0.989493	1.579597	0.163178	44.408392	4.542273	0.092793
BRKR	0.053386	1.232877	0.262969	0.109944	0.460873	2.284716	3.270509	13.301368	1.449111	3.159194	0.217571	38.534862	16.827787	0.303956
OLED	0.062536	3.207125	0.045669	0.099243	0.689341	2.716473	5.423149	26.145225	2.562640	3.159194	0.217571	36.325830	29.972628	0.452086

5. Ranking of the Preference Order Finally, the alternatives are ranked from best (higher relative closeness value) to worst. The best alternatives and the solution to the problem is on the top n of the list. We set as a threshold of relattive closeness:

$$TH_{totpsis} = 0.25$$

and we drop all the stocks are below that threshold leading to a kernel of 7 stocks.

```
We will invest in the following companies :
['Apple', 'Universal Display', 'Microsoft', 'Xilinx', 'Bruker', 'Ubiquiti', 'Cisco']
```

Finally we present the results for these stocks selected by this specific method:

Symbol	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
AAPL	Apple	9.124400e+11	17.01	2.584900e+11	29290000.0	11.67	1.23	25.83	6.42	108.71	4	4
OLED	Universal Display	8.680000e+09	105.25	3.351800e+08	732430.0	1.75	1.51	96.59	109.15	164.87	4	4
MSFT	Microsoft	1.000000e+12	30.14	1.222100e+11	23690000.0	4.48	1.23	32.35	31.96	168.48	4	4
XLNX	Xilinx	2.829000e+10	32.81	3.060000e+09	4360000.0	3.41	1.22	30.29	61.67	138.65	2	4
BRKR	Bruker	7.410000e+09	40.46	1.930000e+09	811410.0	1.17	1.27	58.25	55.53	93.23	4	4
UBNT	Ubiquiti	9.220000e+09	29.86	1.140000e+09	449380.0	4.44	1.34	30.95	51.18	225.04	2	4
CSCO	Cisco	2.409200e+11	20.47	5.132000e+10	20060000.0	2.74	1.19	29.59	28.37	94.97	4	4

Method 2 : PROMETHEE

- It has to be pointed out that MCDA techniques in general place the decision makers in the center of the process and different decision makers can model the problem in different ways, according to their preferences
- In PROMETHEE, a preference degree is an expression of how one action is preferred against another action. For small deviations among the evaluations of a pair of criteria, the decision maker can allocate a small preference; if the deviation can be considered negligible, then this can be modeled in PROMETHEE too
- Therefore the preference function, if the criterion is to be maximized, can be defined as

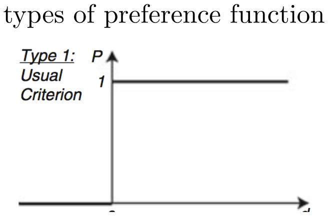
$$P_j(a,b) = F_j[d_j(a,b)], \forall a, b \in A$$

$$d_j(a,b) = g_j(a) - g_j(b)$$

- We will define the thresholds of absolute preference, indifference and the type of preference for each criterion:

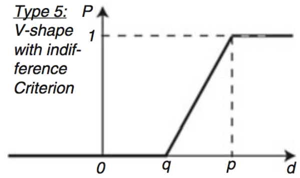
Criterion	p	q	type
Market Cap	10^9	$10^{1}1$	typeV
P/E Ratio	15	35	typeV
Revenue	10^8	$10^{1}0$	typeV
Average Vol (3m.)	10^7	10^8	typeV
EPS	1	6	typeV
Beta	0	0.6	typeV
YTD	5	25	typeV
1-Year Return	5	45	typeV
3-Year Return	10	75	typeV
Weekly performance	0	1	typeI
Monthly performance	0	1	typeI

where they 2 types of preference function are the following



$$P(d) = \begin{cases} 0 & d \leq 0 \\ 1 & d > 0 \end{cases}$$

–



$$P(d) = \begin{cases} 0 & d \leq q \\ \frac{d-q}{p-q} & q < d \leq p \\ 1 & d > p \end{cases}$$

p, q

– **Methodology**

1. Calculate of the deviations between the evaluations of the alternatives in each criterion. So we would create **13 new matrices with the differences** between the evaluations of the stocks on the specific criterion.
2. Use the the *crit* as index to pump the thresholds and the type from the *p_q_type* dictionary, for the specific criterion. Apply the corresponding preference function to these **13 differences matrices**, and calculate result for each pair of stocks. So from this step **13 pairwise comparison matrices** occur.

3. The fourth step of PROMETHEE involves the calculation of **the unicriterion net flows**. To obtain the value of the **positive outranking flow** for **eg. Criterion : P/E Ratio** , the decision maker has to sum the values of the respective line (the main diagonal element is naturally 0) and divide the result with the number of actions minus 1, since a stock is not compared with itself. For the value of the **negative outranking flow** of the same stock , the decision maker has to sum all the elements of the respective column.
4. Finally we calculate the global preference net flows, by adding the **positive and the negative outranking flow**, and we multiply the result by the corresponding criterion weight

eg. Criterion : P/E Ratio

P/E Ratio

Symbol	TXN	OTEX	AMAT	UBNT	AMSWA	TTEC	AVGO	AAPL	QCOM	XLNX	ESLT	TER	CSCO	SSNC	CDW	MSFT	MANT	GRMN	ADI	JKHY	BRKR	OLED	Positive Flow	Negative Flow	Net Flow
Symbol																									
TXN	0	0.2675	0	0	1	0.3355	0	0	0.2275	0	0	0	0	1	0	0	0	0	0	0.1195	0.265	1	0.200714	0.000000	-0.200714
OTEX	0	0	0	0	0.915	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.138810	0.103214	-0.035595
AMAT	0	0.6825	0	0.15	1	0.7505	0.3545	0	0.6425	0.2975	0.2655	0	0	1	0	0.164	0.1995	0	0	0.5345	0.68	1	0.367667	0.000000	-0.367667
UBNT	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.142857	0.007143	-0.135714
AMSWA	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0.822	0.086762	0.887357	0.800595
TTEC	0	0	0	0	0.847	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.135571	0.127500	-0.008071
AVGO	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.142857	0.021500	-0.121357
AAPL	0	0.425	0	0	1	0.493	0.097	0	0.385	0.04	0.008	0	0	1	0	0	0	0	0	0.277	0.4225	1	0.245119	0.000000	-0.245119
QCOM	0	0	0	0	0.955	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.140714	0.089881	-0.050833
XLNX	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.142857	0.016071	-0.126786
ESLT	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.142857	0.013024	-0.129833
TER	0	0.2685	0	0	1	0.3365	0	0	0.2285	0	0	0	0	1	0	0	0	0	0	0.1205	0.266	1	0.200952	0.000000	-0.200952
CSCO	0	0.252	0	0	1	0.32	0	0	0.212	0	0	0	0	1	0	0	0	0	0	0.104	0.2495	1	0.197024	0.000000	-0.197024
SSNC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.000000	0.952381	0.952381
CDW	0	0.0865	0	0	1	0.1545	0	0	0.0465	0	0	0	0	1	0	0	0	0	0	0	0.084	1	0.160548	0.000000	-0.160548
MSFT	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.142857	0.007810	-0.135048
MANT	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.142857	0.009500	-0.133357
GRMN	0	0.1855	0	0	1	0.2535	0	0	0.1455	0	0	0	0	1	0	0	0	0	0	0.0375	0.183	1	0.181190	0.000000	-0.181190
ADI	0	0	0	0	1	0.034	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.144476	0.000000	-0.144476
JKHY	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.142857	0.056810	-0.086048
BRKR	0	0	0	0	0.9175	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0.138929	0.102381	-0.036548
OLED	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.000000	0.943905	0.943905

5. **PROMETHEE** method, on contrary to **ELECTRE I**, produces a final ranking and not selection, but can be used, by choosing the top n stocks with the best net flow performance [11]

Final Net Flow Results

[11] It would be very interesting to use Visual PROMETHEE to produce the corresponding figure

Symbol	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly	Net Flow
AAPL	0.022769	-0.024512	0.150000	0.004164	0.146400	0.017063	-0.014424	-0.030600	0.002890	0.035714	0.007143	0.316607
OLED	-0.007784	0.094390	-0.078329	-0.000678	-0.034129	0.070139	0.100000	0.100000	0.018078	0.035714	0.007143	0.304545
MSFT	0.024850	-0.013505	0.135714	0.002830	0.017900	0.017063	-0.002617	0.018575	0.018662	0.035714	0.007143	0.262331
CSCO	0.020238	-0.019702	0.121429	0.001966	-0.014457	0.008730	-0.008367	0.011454	-0.001355	0.035714	0.007143	0.162793
CDW	-0.006407	-0.016055	0.073131	-0.000680	0.016086	-0.020734	-0.008417	0.003842	0.016444	0.035714	0.007143	0.100067
BRKR	-0.008010	-0.003655	-0.062826	-0.000673	-0.044543	0.025397	0.080781	0.065057	-0.001860	0.035714	0.007143	0.092525
TXN	0.013762	-0.020071	0.069343	-0.000460	0.040071	0.014980	-0.041829	-0.046689	-0.005364	0.035714	0.007143	0.066602
QCOM	0.009807	-0.005083	0.089711	0.002218	-0.033100	0.085317	-0.016545	-0.003736	-0.017071	-0.065476	0.007143	0.053186
TER	-0.007917	-0.020095	-0.061268	-0.000602	-0.024329	0.076687	0.047531	-0.011420	0.011171	0.035714	0.007143	0.052614
ADI	-0.000701	-0.014448	-0.019608	-0.000569	0.013000	0.050198	-0.011005	-0.026752	-0.001819	0.035714	0.007143	0.031153
AMAT	-0.000904	-0.036767	0.069978	-0.000228	0.002043	0.088095	-0.008967	-0.065969	-0.005325	0.035714	-0.071429	0.006242
ESLT	-0.008127	-0.012983	-0.035727	-0.000716	0.025157	-0.062996	0.017307	0.016612	-0.007175	0.035714	0.007143	-0.025790
UBNT	-0.007683	-0.013571	-0.070344	-0.000693	0.017157	0.038889	-0.005669	0.058992	0.024294	-0.101190	0.007143	-0.052676
AVGO	0.014678	-0.012136	0.089885	-0.000569	0.109400	-0.048710	-0.076917	-0.032842	-0.006345	-0.101190	0.007143	-0.057603
GRMN	-0.006411	-0.018119	-0.048670	-0.000660	0.004443	-0.035218	-0.011476	0.017705	-0.002734	0.035714	0.007143	-0.058284
XLNX	-0.003489	-0.012679	-0.051960	-0.000486	-0.000843	0.014980	-0.007079	0.072939	0.012325	-0.101190	0.007143	-0.070338
TTEC	-0.009067	-0.000807	-0.066585	-0.000712	-0.046086	-0.087897	0.080252	-0.003698	-0.008964	0.035714	0.007143	-0.100705
SSNC	-0.006526	0.095238	-0.041340	-0.000644	-0.057871	0.029464	-0.009733	-0.030483	-0.000449	-0.101190	0.007143	-0.116393
MANT	-0.008976	-0.013336	-0.062307	-0.000710	-0.028386	-0.043056	-0.020502	-0.008906	-0.005690	0.035714	0.007143	-0.149011
AMSWA	-0.009369	0.080060	-0.080441	-0.000713	-0.064300	-0.086409	0.008643	-0.052842	-0.014618	0.035714	0.007143	-0.177132
OTEX	-0.007314	-0.003560	-0.053634	-0.000690	-0.047200	-0.108929	-0.015186	-0.017468	-0.016182	0.035714	0.007143	-0.227305
JKHY	-0.007419	-0.008605	-0.066152	-0.000696	0.003586	-0.043056	-0.075783	-0.033770	-0.008913	-0.101190	-0.071429	-0.413427

Finally we present the results for these stocks selected by this specific method:

```
We will invest in the following companies :
['Apple', 'Universal Display', 'Microsoft', 'Cisco', 'CDW Corp', 'Bruker', 'Texas Instruments']
```

Symbol	Name	Market Cap	P/E Ratio	Revenue	Average Vol. (3m)	EPS	Beta	YTD	1 Year	3 Years	Weekly	Monthly
AAPL	Apple	9.124400e+11	17.01	2.584900e+11	29290000.0	11.67	1.23	25.83	6.42	108.71	4	4
OLED	Universal Display	8.680000e+09	105.25	3.351800e+08	732430.0	1.75	1.51	96.59	109.15	164.87	4	4
MSFT	Microsoft	1.000000e+12	30.14	1.222100e+11	23690000.0	4.48	1.23	32.35	31.96	168.48	4	4
CSCO	Cisco	2.409200e+11	20.47	5.132000e+10	20060000.0	2.74	1.19	29.59	28.37	94.97	4	4
CDW	CDW Corp	1.534000e+10	23.78	1.659000e+10	684910.0	4.38	1.05	29.56	24.58	155.31	4	4
BRKR	Bruker	7.410000e+09	40.46	1.930000e+09	811410.0	1.17	1.27	58.25	55.53	93.23	4	4
TXN	Texas Instruments	1.046900e+11	20.16	1.559000e+10	4850000.0	5.51	1.22	17.39	-2.90	78.89	4	4

- Key Notes?
  - We use the same weights for our criteria for each one of the methods used.
  - **ELECTRE I** : The application of ELECTRE I with veto method led to almost same results with the results occured when applying ELECTRE I without veto. They were differed in just 1 stock.

3 Conclusions

- The subset of stocks, which constitutes the intersection of the kernels which were produced by the 3 (actually 4) methods is:

  1. **Apple - APPL**
  2. **Microsoft - MSFT**
  3. **Universal Display - OLED**

  Choosing the kernel of which ever method is recommended for a further **asset allocation / goal programming** problem, and for the worried ones the intersection if the perfect solution occurred as convergence of the 3 methods.

- The interesting part is that all the methods conclude to **Universal Display** as a proper solution to be included in the final kernel. It is amazing, how its excellent annualized performance (**YTD, 1-Year, 3-Year**) compensates its worse performance in the rest of criteria in order to still consider it as a good solution.

- An important and challenging step in the solution of a decision making problem is the **elicitation of weights**. Choosing different weights could lead to different kernels.

4 Sources

- **MCDA**
  - https://www.wikiwand.com/en/Multiple-criteria_decision_analysis
  - https://www.researchgate.net/publication/276803686_Stock_selection_using_a_hybrid_MCDM_approach
- **Group Decision making**
  - https://www.rightpathinvestments.com/content/wp-content/uploads/2012/03/GroupDecisionMaking_2009_Vanguard
- **LSTM**
  - https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
  - https://zhuanlan.zhihu.com/p/31783805
- **RECOMENDATION CHART**
  - https://www.investopedia.com/financial-edge/0512/understanding-analyst-ratings.aspx
- **NASDAQ**
  - https://www.asx.com.au/documents/asx-news/ASXselectsNasdaqSentinel.pdf
  - https://marketrealist.com/2015/04/nasdaq-less-prone-y2k-type-crash/
- **Treyner Ratio**
  - https://corporatefinanceinstitute.com/resources/knowledge/finance/treynor-ratio/
- **Market Benchmark - S&P 500**
  - https://www.investopedia.com/ask/answers/041315/what-are-pros-and-cons-using-sp-500-benchmark.asp
  - https://www.investopedia.com/terms/b/benchmark.asp
- **STOCKS**
  - https://www.investopedia.com/ask/answers/advantages-and-disadvantages-buying-stocks-instead-of-bonds/
  - https://www.investopedia.com/articles/stocks/10/primer-on-the-tech-industry.asp
  - https://www.marketwatch.com/story/these-15-tech-stocks-are-backed-by-the-quarters-best-sales-figures-2019-05-09
  - https://money.stackexchange.com/questions/96421/why-are-tech-stocks-risky-investments
  - https://www.ruleoneinvesting.com/blog/how-to-invest/tech-stocks/
- **Beta**
  - https://www.investopedia.com/ask/answers/032615/how-do-risks-large-cap-stocks-differ-risks-small-cap-stocks.asp
  - https://www.nasdaq.com/article/4-compelling-low-beta-tech-stocks-in-the-market-right-now-cm884754
- **Average Volume**
  - https://www.investors.com/how-to-invest/investors-corner/how-much-volume-should-a-stock-have/
- **EPS**
  - https://finance.zacks.com/considered-good-eps-stock-market-2938.html
  - https://www.quora.com/Why-is-earnings-per-share-important
- **Marke Cap**

- https://www.schroders.com/en/us/institutional/insights/equities/small-cap-vs-large-cap-how-valuations-compare/
- https://www.investopedia.com/ask/answers/041015/what-are-common-advantages-investing-large-cap-stocks.asp

- **ELECTRE I**
  - https://file.scirp.org/pdf/CS_2016052714021050.pdf
- **TOPSIS & PROMETHEE**
  - https://books.google.gr/books?id=tXdvDwAAQBAJ&pg=PP6&lpg=PP6&v=onepage&q&f=false

5 Appendix

- For the sake of completeness we attach the code executed, especially for those who don't want to wait for the training of the LSTM and its predictions ($\sim$ 45 minutes)
- However, despite being in *.ipynb* form, we recommend its execution in the **https://colab.research.google.com**, since GPU is available for free.

final

June 28, 2019

[8]:
```
import numpy as np
import pandas as pd
from pandas_datareader import data as web
import fix_yahoo_finance
import random
from scipy import stats
from functools import reduce
from operator import mul
import matplotlib.pyplot as plt
from matplotlib.ticker import FormatStrFormatter
import math
from datetime import datetime
! pip install quandl
import quandl
from collections import defaultdict
import csv
from IPython.display import display

# to download csv preprocessed files later
from IPython.display import HTML
pointer = 0
def create_download_link(title = "Download CSV file", filename = "data.csv"):
    html = '<a href={filename}>{title}</a>'
    html = html.format(title=title,filename=filename)
    return HTML(html)
```

```
Requirement already satisfied: quandl in /usr/local/lib/python3.6/dist-packages
(3.4.8)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from quandl) (1.12.0)
Requirement already satisfied: numpy>=1.8 in /usr/local/lib/python3.6/dist-
packages (from quandl) (1.16.4)
Requirement already satisfied: requests>=2.7.0 in /usr/local/lib/python3.6/dist-
packages (from quandl) (2.21.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist-
packages (from quandl) (2.5.3)
Requirement already satisfied: pandas>=0.14 in /usr/local/lib/python3.6/dist-
```

```
packages (from quandl) (0.24.2)
Requirement already satisfied: pyOpenSSL in /usr/local/lib/python3.6/dist-
packages (from quandl) (19.0.0)
Requirement already satisfied: inflection>=0.3.1 in
/usr/local/lib/python3.6/dist-packages (from quandl) (0.3.1)
Requirement already satisfied: more-itertools<=5.0.0 in
/usr/local/lib/python3.6/dist-packages (from quandl) (5.0.0)
Requirement already satisfied: ndg-httpsclient in /usr/local/lib/python3.6/dist-
packages (from quandl) (0.5.1)
Requirement already satisfied: pyasn1 in /usr/local/lib/python3.6/dist-packages
(from quandl) (0.4.5)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.6/dist-packages (from requests>=2.7.0->quandl) (2019.3.9)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in
/usr/local/lib/python3.6/dist-packages (from requests>=2.7.0->quandl) (3.0.4)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests>=2.7.0->quandl) (1.24.3)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from requests>=2.7.0->quandl) (2.8)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-
packages (from pandas>=0.14->quandl) (2018.9)
Requirement already satisfied: cryptography>=2.3 in
/usr/local/lib/python3.6/dist-packages (from pyOpenSSL->quandl) (2.7)
Requirement already satisfied: cffi!=1.11.3,>=1.8 in
/usr/local/lib/python3.6/dist-packages (from
cryptography>=2.3->pyOpenSSL->quandl) (1.12.3)
Requirement already satisfied: asn1crypto>=0.21.0 in
/usr/local/lib/python3.6/dist-packages (from
cryptography>=2.3->pyOpenSSL->quandl) (0.24.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.6/dist-
packages (from cffi!=1.11.3,>=1.8->cryptography>=2.3->pyOpenSSL->quandl) (2.19)
```

[9]:
```
####################################################
#      A. data preprocessed by investing.com       #
####################################################

##############################
#   1. Check data's format   #
##############################
# > 91 stocks were selected related to technology, along with 8 criteria:
 ↪"Market Cap","P/E Ratio","Revenue","Average Vol.
 ↪(3m)","EPS","Beta","Dividend","Yield"
# > However the parsing wasn't accurate and led to the following .csv data file
 ↪which needs to be preprocessed

stocks = pd.read_csv("fundamental.csv", delimiter=',')
# to be able to see the whole names
```

```
pd.set_option('display.max_colwidth', -1)
# to check the 5 columns
stocks.head()
```

[9]:

```
  Name,"Market Cap","P/E Ratio","Revenue","Average Vol.
(3m)","EPS","Beta","Dividend","Yield"
0  Apple,"912.44B","17.01","258.49B","29.29M","11.67","1.23","3.08","1.55%"
1  Microsoft,"1.03","30.14","122.21B","23.69M","4.48","1.23","1.84","1.36%"
2  Intel,"213.24B","10.86","70.84B","23.77M","4.36","0.83","1.26","2.66%"
3  Cisco,"240.92B","20.47","51.32B","20.06M","2.74","1.19","1.40","2.50%"
4  Broadcom,"110.29B","33.95","21.31B","2.79M","8.19","0.91","10.60","3.81%"
```

[0]:

```
#########################################################################################
#   2. Preprocess csv files & Combine fundamental, technical and performance␣
 ↪analysis  #
#########################################################################################
# Step 1: For all 91 stocks extract the criteria results.
fundamental = open("fundamental.csv",mode= 'r')
fundamental = csv.reader(fundamental, delimiter=',')
fields_fundamenttal = next(fundamental)[0].split(',')

performance = open("performance.csv",mode= 'r')
performance = csv.reader(performance, delimiter=',')
fields_performance = next(performance)[0].split(',')

technical = open("technical.csv",mode= 'r')
technical = csv.reader(technical, delimiter=',')
technical_dic = {"Strong Sell": 0, "Sell": 1, "Neutral": 2, "Buy": 3, "Strong␣
 ↪Buy": 4}
fields_technical = next(technical)[0].split(',')


# Step 2: Choose the criteria necessary for our analysis
# > choose performance criteria: YTD, 1 Year, 3 Year
# > choosse technical criteria: Weekly, Monthly
fields = fields_fundamenttal + fields_performance[-3:] + fields_technical[-2:]
fields = [fields[0]] + [i[1:-1] for i in fields[1:]]


# Step 3: parse simutanously all the stocks, keep the criteria results␣
 ↪necessary,
# combine them, and create a new whole analysis stock csv with all the criteria
# combined

stocksCSV = open("stocks.csv", mode= 'w')
stocksCSV = csv.DictWriter(stocksCSV, fieldnames = fields)
stocksCSV.writeheader()
```

```
for f, p, t in zip(fundamental, performance, technical):
  #################### FUNDAMENTAL ####################
  f = f[0].split(",")
  stock_name = f[0]
  f = [i[1:-1] for i in f[1:]]

  # edit 'Market Cap'
  if f[0][-1] == "B":
    f[0] = float(f[0][:-1]) * 10**9
  elif f[0][-1] == "M":
    f[0] = float(f[0][:-1]) * 10**6
  elif f[0][-1] == "K":
    f[0] = float(f[0][:-1]) * 10**3
  else :
    f[0] = float(f[0][:-1]) * 10**12

  # edit 'Revenue'
  if f[2][-1] == "B":
    f[2] = float(f[2][:-1]) * 10**9
  elif f[2][-1] == "M":
    f[2] = float(f[2][:-1]) * 10**6
  elif row[2][-1] == "K":
    f[2] = float(f[2][:-1]) * 10**3
  else :
    f[2] = float(f[2][:-1])

  # edit 'Average Vol.'
  if f[3][-1] == "M":
    f[3] = float(f[3][:-1]) * 10**6
  elif f[3][-1] == "K":
    f[3] = float(f[3][:-1]) * 10**3
  else :
    f[3] = float(f[3][:-1])

  # edit 'Yield' : convert percentage to decia
  f[-1] = float(f[-1][:-1]) / 10

  # edit rest fields as floats
  f = list(map(float, f))


  #################### PERFORMANCE ####################
  p = p[0].split(",")
  p = [i[1:-1] for i in p[4:]]
  p = list(map(float, p))

  #################### TECHNICAL ####################
```

```
  t = t[0].split(",")
  t = [i[1:-1] for i in t[4:]]
  t = [technical_dic[i] for i in t]

  row_values = [stock_name] + f + p + t
  row_dic = {}

  for i in range(len(row_values)):
    row_dic[fields[i]] = row_values[i]

  stocksCSV.writerow(row_dic)
```

[11]:
```
################################################
#   3. Read stocks & criteria as a dataframe   #
################################################

stocks_investingcom = pd.read_csv("stocks.csv")

# Step 2.1 : drop unecessary criteria
del stocks_investingcom['Dividend']
del stocks_investingcom['Yield']
fields.remove('Dividend')
fields.remove('Yield')

# Step 2.2 : Handle exceptions :drop stocks for which no info are provided by␣
 ↪DM2 yahoo finance later
# EXCLUDE : Simulations Plus , Maxim
stocks_investingcom.index = range(91)
stocks_investingcom = stocks_investingcom.drop(stocks_investingcom.index[[34]])
stocks_investingcom = stocks_investingcom.drop(stocks_investingcom.index[[87]])
stocks_investingcom.index = range(89)



# to be able to see the whole names
pd.set_option('display.max_colwidth', -1)
display(stocks_investingcom.head())

# download stocks investing.com
stocks_investingcom.to_csv('a.csv')
create_download_link(filename = 'a.csv')
```

```
        Name     Market Cap  P/E Ratio  ...  3 Years  Weekly  Monthly
0  Apple      9.124400e+11  17.01      ...  108.71   4       4
1  Microsoft  1.000000e+12  30.14      ...  168.48   4       4
2  Intel      2.132400e+11  10.86      ...  46.66    0       2
```

```
3  Cisco      2.409200e+11  20.47      ...  94.97    4       4
4  Broadcom   1.102900e+11  33.95      ...  74.77    2       4

[5 rows x 12 columns]
```

[11]: <IPython.core.display.HTML object>

[12]:
```
########################
#   4. Normalization   #
########################
#############################################
#                        x_ij               #
#    r_ij = ---------------------------     #
#            sqrt(sum(x_ij) for all j)      #
#############################################
# In order to be able to compare different kinds of criteria the first step
# is to make them dimensionless
stocks_investingcom_normalized = stocks_investingcom.copy()
for criterion in fields[1:]:
  crit_values = list(stocks_investingcom_normalized[criterion])
  rms = np.sqrt(sum([i**2 for i in crit_values]))
  stocks_investingcom_normalized[criterion] =␣
 ↪stocks_investingcom_normalized[criterion] / rms

display(stocks_investingcom_normalized.head())
```

```
        Name  Market Cap  P/E Ratio  ...   3 Years    Weekly   Monthly
0  Apple        0.646330   0.024870  ...  0.123725  0.153393  0.139686
1  Microsoft    0.708354   0.044067  ...  0.191751  0.153393  0.139686
2  Intel        0.151049   0.015878  ...  0.053105  0.000000  0.069843
3  Cisco        0.170657   0.029929  ...  0.108087  0.153393  0.139686
4  Broadcom     0.078124   0.049638  ...  0.085097  0.076696  0.139686

[5 rows x 12 columns]
```

[13]:
```
#################################
#   5. Weigthed Normalization   #
#################################
# Step 1: We define the weights for each criterion
# Market Cap : 2.5%
# P/E Ratio: 2.5%
# Revenue: 20%
# Average Vol: 2.5%
# EPS: 20%
# Beta: 10%
# YTD: 10%
```

```
# 1 Year: 10%
# 3 Year: 2.5%
# Weekly: 7.5%
# Monthly: 12.5%
weights = [ 0.025, 0.025, 0.1, 0.025, 0.1, 0.2, 0.1, 0.1, 0.025, 0.2, 0.1]
for i in range(1,len(fields)-1):
  stocks_investingcom_normalized[fields[i]] =␣
 ↪stocks_investingcom_normalized[fields[i]] / weights[i]

display(stocks_investingcom_normalized.head())
```

```
        Name  Market Cap  P/E Ratio  ...   3 Years   Weekly   Monthly
0  Apple      25.853210   0.248700   ...  0.618626  1.533930  0.139686
1  Microsoft  28.334148   0.440671   ...  0.958753  1.533930  0.139686
2  Intel      6.041974    0.158782   ...  0.265524  0.000000  0.069843
3  Cisco      6.826263    0.299288   ...  0.540437  1.533930  0.139686
4  Broadcom   3.124973    0.496376   ...  0.425487  0.766965  0.139686

[5 rows x 12 columns]
```

[14]:
```
########################
#   6. Final Ranking   #
########################
# Final ranking for the stocks of the portfolio
# Total sum per row:
stocks_investingcom_normalized['Score'] = stocks_investingcom_normalized.sum(1)
stocks_investingcom_normalized = stocks_investingcom_normalized.
 ↪sort_values('Score', ascending=False)
display(stocks_investingcom_normalized.head())


# download normalized stocks investing.com
stocks_investingcom_normalized.to_csv('b.csv')
create_download_link(filename = 'b.csv')
```

```
                  Name  Market Cap  P/E Ratio  ...   Weekly   Monthly      Score
0   Apple              25.853210   0.248700   ...  1.53393  0.139686  72.180471
1   Microsoft          28.334148   0.440671   ...  1.53393  0.139686  58.994641
3   Cisco              6.826263    0.299288   ...  1.53393  0.139686  26.078603
72  Universal Display  0.245940    1.538838   ...  1.53393  0.139686  24.995191
80  AudioCodes         0.013101    0.494621   ...  1.53393  0.139686  22.661429

[5 rows x 13 columns]
```

[14]: <IPython.core.display.HTML object>

[15]:
```
########################################
#      B. data preprocessed by me     #
########################################
######################################################
#   1. Asssign ticker symbols  to the company names #
######################################################
# So far we have 2 corpora:
#      1) (Corpus s) Investing.com: Company names, Market Cap, P/E,...
#      2) (Corpus t) Yahoo Finance: Company names, ticker symbols resspectively
# For each company name in corpus s, we will find the most probable
# company name matching in corpus t, and we will keep each ticker symbol
# eg. (s) 'Apple' ---- best matching ---> (t) 'Apple Inc.'' ---> Keep 'AAPL'

# Step 1 : Read corpus t as dataframe
tickers = pd.read_csv("companylist.csv")
tickers = tickers[['Symbol','Name']]
stocks_tickers = pd.DataFrame(columns=['Name', 'Symbol'])


# Step 2 : Define replacement dictionaries used in the tokenization process␣
↪later
replacement_dic = {".": "", "&": " ", ",": " "}
replacement_bussiness_dic = {"corporation": "corp corpo","laboratories":
↪"labs","inc":" "}

# Step 3: Define functions used in the tokenization process
def tokenize(s):
  # Step 3.1 : Python is capital sensitive, so lower the strings to find the␣
↪ticker.
  s = s.lower()
  # Step 3.2 : replace common puncutation marks AND business abbreviations
  s = replace_all(s, replacement_dic)
  s = replace_all(s, replacement_bussiness_dic)
  # Step 3.3 : Tokenize company name by breaking down into words.
  s = s.split(" ")
  s = [i for i in s if i != '']
  return s

def replace_all(text, dic):
    for i, j in dic.items():
      text = text.replace(i, j)
    return text

def any_2(list1, list2):
  # Step 3.4 : Check if at least 2 stock tokensis part of the ticker companies␣
↪tokens.
  c = 0
```

```
  for i in list1:
    if i in list2:
      c+=1
  length = len(list1)
  if c > 1 or (length == 1 and c == 1) : return True
  else: return False


# Ticker Assigment process:
# (s) ---------------------------...--------------->
#       |        |
#     Apple  Microsoft
# 1. Create word tokens for the company names of s,t
#    eg. s : 'Kulicke&Soffa'  -> ['kulicke', 'soffa']
#        t : 'Kulicke and Soffa Industries, Inc.' -> ['kulicke', 'and',␣
↪'soffa', 'industries']
# 2. Create matching list between the s and t word tokens iff only at least 2 s␣
↪word tokens
#    exist in the t word tokens
# 3. For the non empty matching lists, find the respective ticker of the t␣
↪company name
#    and append (company name, ticker) in the new dataframe
# 4. Manually add the tickers for the excpeptions of the previous process
exceptions = []
stock_ptr = 0
for s in stocks_investingcom_normalized['Name']:
  matching = [t for t in tickers['Name'] if any_2(tokenize(s),tokenize(t))]
  if matching and s != "Bel Fuse A" and s != "Bel Fuse B":
    symbol = tickers.loc[tickers['Name'] == matching[0], 'Symbol'].item()
    stocks_tickers.loc[stock_ptr] = [s, symbol]
    stock_ptr+=1
  else:
    exceptions.append(s)

exceptions
stocks_tickers.loc[stock_ptr] = [exceptions[0], 'SSNC']
stocks_tickers.loc[stock_ptr + 1] = [exceptions[1], 'CTSH']
stocks_tickers.loc[stock_ptr + 2] = [exceptions[2], 'BELFA']
stocks_tickers.loc[stock_ptr + 3] = [exceptions[3], 'BELFB']

stocks_tickers.index = range(89)
print(stocks_tickers)


# download stock tickers
stocks_tickers.to_csv('c.csv')
create_download_link(filename = 'c.csv')
```

	Name	Symbol
0	Apple	AAPL
1	Microsoft	MSFT
2	Cisco	CSCO
3	Universal Display	OLED
4	AudioCodes	AUDC
5	Intel	INTC
6	Taitron	TAIT
7	Qualcomm	QCOM
8	Xilinx	XLNX
9	Bruker	BRKR
10	Sapiens	SPNS
11	Ubiquiti	UBNT
12	Cypress	CY
13	Intuit	INTU
14	CDW Corp	CDW
15	Equinix	EQIX
16	Broadcom	AVGO
17	Elbit Systems	ESLT
18	Texas Instruments	TXN
19	Garmin	GRMN
20	AstroNova	ALOT
21	Formula Systems ADR	FORTY
22	Teradyne	TER
23	Analog Devices	ADI
24	Cerner	CERN
25	TESSCO	TESS
26	TTEC	TTEC
27	Simulations Plus	SLP
28	Applied Materials	AMAT
29	Xperi	XPER
..	...	...
59	Amdocs	DOX
60	Citrix Systems	CTXS
61	Magic	MGIC
62	Gilat	GILT
63	NetApp	NTAP
64	NIC	EGOV
65	National Instruments	NATI
66	InterDigital	IDCC
67	MIND CTI	MNDO
68	Skyworks	SWKS
69	Asia Pacific Wire & Cable	APWC
70	Silicon Motion	SIMO
71	Luminex	LMNX
72	MKS Instruments	MKSI
73	Hollysys Automation Tech	HOLI
74	Littelfuse	LFUS

```
75  Activision Blizzard         ATVI
76  Western Digital             WDC
77  CDK Global Holdings LLC     CDK
78  Wayside                     WSTG
79  Monotype                    TYPE
80  Computer Programs&Systems   CPSI
81  Allied Motion               AMOT
82  LogMeIn                     LOGM
83  Ebix                        EBIX
84  Himax                       HIMX
85  SS&Cs                       SSNC
86  Cognizant A                 CTSH
87  Bel Fuse A                  BELFA
88  Bel Fuse B                  BELFB

[89 rows x 2 columns]
```

[15]: <IPython.core.display.HTML object>

[16]:
```
################################################
#   2. Read stocks & criteria as a dataframe   #
################################################
# Pull Adjusted closing prices with Pandas datareader
stocks_yahoofinance = pd.DataFrame()

for item in stocks_tickers['Symbol']:
  print(item)
  stocks_yahoofinance[item] = web.DataReader(item, data_source='yahoo',␣
 ↪start='20-06-2016', end='20-06-2019')['Adj Close']

stocks_yahoofinance.head()
```

```
AAPL
MSFT
CSCO
OLED
AUDC
INTC
TAIT
QCOM
XLNX
BRKR
SPNS
UBNT
CY
INTU
CDW
EQIX
```

AVGO
ESLT
TXN
GRMN
ALOT
FORTY
TER
ADI
CERN
TESS
TTEC
SLP
AMAT
XPER
MANT
CSGS
AVT
KLAC
OTEX
CONE
MTSC
NVDA
NXPI
JKHY
CGNX
PRGS
JCOM
KE
AMSWA
STX
CCMP
TACT
CMTL
MCHP
MLAB
RFIL
MPWR
SABR
FELE
LOGI
BLKB
KLIC
POWI
DOX
CTXS
MGIC
GILT
NTAP

```
EGOV
NATI
IDCC
MNDO
SWKS
APWC
SIMO
LMNX
MKSI
HOLI
LFUS
ATVI
WDC
CDK
WSTG
TYPE
CPSI
AMOT
LOGM
EBIX
HIMX
SSNC
CTSH
BELFA
BELFB
```

[16]:

```
                 AAPL       MSFT       CSCO  ...       CTSH      BELFA
BELFB
Date                                         ...
2016-06-20  90.507019  47.147133  26.153543  ...  59.739643  15.422829
17.929546
2016-06-21  91.277893  48.201756  26.126299  ...  60.032009  15.403572
17.958401
2016-06-22  90.935295  48.013435  26.080891  ...  60.148949  15.740524
18.112301
2016-06-23  91.458733  48.879723  26.534950  ...  61.104008  16.077478
18.602863
2016-06-24  88.889130  46.921150  25.200026  ...  56.533382  15.894562
17.996874

[5 rows x 89 columns]
```

[17]:

```
####################################################
#   3. Calculate daily & annualized/ytd returns   #
####################################################
from functools import reduce
from operator import mul
```

```
def daily(data):
  daily_returns = data.pct_change()
  daily_returns = daily_returns + 1
  daily_returns = daily_returns.fillna(0)
  return daily_returns


def annualized(data, tickers, daily_returns):
  d = {'Annualized Returns': ['1 Year', '2 Year', '3 Year', 'YTD']}
  annualized_returns = pd.DataFrame(d).set_index('Annualized Returns')
  length = data.shape[0]

  print()

  for item in tickers['Symbol']:
    ar = []
    # Year 1,2,3
    for i in range(251,length,251):
      ar.append((reduce(mul,daily_returns[item][(length-i):],1)**(251/i) -
 ↪1)*100)
    # YTD

    ytd_length = data[data.index >= '01-01-2019'].shape[0]
    ar.append( ( reduce(mul, daily_returns[item][(length - ytd_length):], 1) -
 ↪1)*100)
    annualized_returns[item] = pd.Series(ar, index = annualized_returns.index)

  return annualized_returns


#######################DAILY RETURNS#######################
# Simple daily returns by using 'pct_change()' funciton (Percentage change
 ↪between the current and a prior element.)
daily_returns = daily(stocks_yahoofinance)


#######################ANNUALIZED & YTD RETURNS#######################
# Annualized Return's Difference From Average Return
# Calculations of simple averages only work when numbers are independent of
 ↪each other. The annualized return is used because the amount of investment
 ↪lost or
# gained in a given year is interdependent with the amount from the other years
 ↪under consideration because of compounding. For example, if a mutual fund
 ↪manager loses half of
# her client's money, she has to make a 100% return to break even. Using the
 ↪more accurate annualized return also gives a clearer picture when comparing
 ↪various mutual funds or
```

```
# the return of stocks that have traded over different time periods.
# However, when we want to know the average of annual returns that are␣
↪compounded, the simple average is not accurate
# Note: Annualise daily returns ~> 250 trading days in a year

annualized_returns = annualized(stocks_yahoofinance, stocks_tickers,␣
↪daily_returns)

display(annualized_returns.head())


# download stock from yahoo finance
stocks_yahoofinance.to_csv('d.csv')
create_download_link(filename = 'd.csv')
```

```
                         AAPL       MSFT  ...      BELFA      BELFB
Annualized Returns                        ...
1 Year               8.577046  36.619081  ... -29.228781 -22.380219
2 Year              18.733969  42.099427  ... -17.043819 -15.692833
3 Year              29.929505  41.818563  ...  -2.923429  -1.994546
YTD                 27.479941  35.910181  ...   5.421110  -6.831252

[4 rows x 89 columns]
```

[17]: <IPython.core.display.HTML object>

[18]:
```
############################
#   5. Calculate metrics   #
############################
def apr(data, r_f):
  apr = (np.log(data) - np.log(data.shift(1)))*252*100
  apr -= r_f
  apr = apr.to_frame()
  apr.fillna(0,inplace=True)
  return apr

# We will work based on CAMP model:
########################################
#     R_s = R_f + beta * (R_m - R_f)   #
########################################
# R_s : Expected return of the securiy
# R_f : Risk-free rat
# R_m : Expected return of the market chosen
```

```
# Step 1 : Risk free rate based on [ Tbills ]
# This asset exists only in theory but often yields on low-risk instruments␣
↪like 3-month U.S.
# Treasury Bills can be viewed as  being virtually risk-free and thus their␣
↪yields can be used
# to approximate the risk-free rate. I get the data for these instruments below.
# KEY NOTE: we won't get the most recent Treasury Bill rate, but instead we␣
↪will use whole historic
# data, so that our calculations are more precise
tbill = pd.DataFrame()
tbill = web.DataReader('^IRX', data_source='yahoo', start='20-06-2016',end =␣
↪'20-06-2019')['Adj Close']

# Step 2 : Market retun based on [S&P500]
market = pd.DataFrame({'^GSPC' : web.DataReader('^GSPC', data_source='yahoo',␣
↪start='20-06-2016',end = '20-06-2019')['Adj Close']})
market_ticker = pd.DataFrame(columns = ['Symbol'])
market_ticker.loc[0] = ['^GSPC']


# Step 3 : 1) Calculate [ annual percentage rate ] of the market, so the amount␣
↪of interest an investment earns over the course of a year
#          2) Daily returns of the market
#          3) Annualized returns of the markett
market_apr = apr(market['^GSPC'], tbill)['^GSPC']
market_daily_returns = daily(market)
market_annualized_returns = annualized(market, market_ticker,␣
↪market_daily_returns)
# std, mean
market_std = market_apr.std()
market_mean = market_apr.mean()



# Step 4 : Define stocks_yahoofinance dataframe & calculate some metrics
stocks_yahoofinance_edited = pd.DataFrame(columns = ['alpha', 'beta',␣
↪'r-squared', 'share_ratio', 'treynor_ratio', 'f_test'])
stock_ptr = 0


for item in stocks_tickers['Symbol']:
  # Step 4.1 : Calculate [ annual percentage rate ] of each stock
  stock_apr = apr(stocks_yahoofinance[item], tbill)
```

```
# Step 4.2 : How much each stock is correlated with Market Benchmark (our
↪approximation of the market).
smcorr = stock_apr.corrwith(market_apr).item()


# Step 4.3 : Calculate alpha and beta
# std, mean
stock_std = stock_apr.std().item()
stock_mean = stock_apr.mean().item()

# beta, alpha
########################################
#                _Y         _       _  #
#    b = r_XY * ----- , a = Y - b * X  #
#                _X                    #
########################################

beta = smcorr * stock_std / market_std
alpha = stock_mean - beta * market_mean

# Step 4.4 :  Calculate Annualised sharpe ratio
sharpe_ratio = (stock_mean / stock_std) * math.sqrt(252)

# Step 4.5: R^2

↪################################################################################
      #                        _         _
↪      #
      #             1      (yi- y)    (xi- x)
↪     #
      #   R^2 =   (-----  (-------- x --------))**2 or R^2 = ((smcorr))**2
↪   #
      #            (n-1)     (yi)      (xi)
↪   #

↪################################################################################
r_squared = smcorr**2 * 100

# Step 4.6 : Treynor Measure
#################################
#                  R_s - R_f    #
#     treynor = -------------   #
#                   beta        #
#################################
```

```
    # > The Treynor ratio, also known as the reward-to-volatility ratio, is a␣
↪performance
    #   metric for determining how much excess return was generated for each unit␣
↪of
    #   risk taken on by a portfolio.
    # > R_s: We will use last 3 year annualized return, since we have an
    #   long term horizon concept and we seek for low volatility
    # > R_f : 5%
    treynor_ratio = ( annualized_returns[item][2]/100 - 0.05) / beta


    # Step 4.7 : F-test
    #############################
    #                R^2        #
    #               -----       #
    #                k-1        #
    #          F = -------      #
    #               1-R^2       #
    #               -----       #
    #                n-k        #
    #############################
    # > k = 2 parameters of the CAMP model (alpa, beta)
    # > n = the length of the samples

    f_test = r_squared/100 / ( (1 - r_squared/100) / (stocks_yahoofinance.
↪shape[0] - 2) )

    stocks_yahoofinance_edited.loc[stock_ptr] = [alpha, beta, r_squared ,␣
↪sharpe_ratio, treynor_ratio, f_test]
    stock_ptr+=1


# Step 5.1 : Droping stocks based on their bad sharpe ratio performance. The␣
↪higher the better
stocks_yahoofinance_edited.index = stocks_tickers['Symbol']
stocks_yahoofinance_edited = stocks_yahoofinance_edited.
↪sort_values(['share_ratio'], ascending=[False])
stocks_yahoofinance_edited =␣
↪stocks_yahoofinance_edited[stocks_yahoofinance_edited['share_ratio'] >=0]


# Step 5.2 : Droping stocks based on their bad treynor_ratio performance. The␣
↪higher the better
stocks_yahoofinance_edited = stocks_yahoofinance_edited.
↪sort_values(['treynor_ratio'], ascending=[False])
```

```
stocks_yahoofinance_edited =␣
↪stocks_yahoofinance_edited[stocks_yahoofinance_edited['treynor_ratio'] >=0]


# Step 5.3 : Droping stocks based on their alpha performance, since Alpha is␣
↪one of the five major risk management
# indicators for mutual funds, stocks and bonds, and in a sense tells investors␣
↪whether an asset has performed
# better or worse than its beta predicts.
stocks_yahoofinance_edited = stocks_yahoofinance_edited.sort_values(['alpha'],␣
↪ascending=[False])
stocks_yahoofinance_edited =␣
↪stocks_yahoofinance_edited[stocks_yahoofinance_edited['alpha'] > 0 ]

# Step 5.4 : Droping stocks based on their f_test performance. Based on that␣
↪metric we can
# conclude upon the importance of the CAMP model and how much useful is it. The␣
↪higher the better
# since we can guarantee about the accuracy of our results
stocks_yahoofinance_edited = stocks_yahoofinance_edited.sort_values(['f_test'],␣
↪ascending=[False])
stocks_yahoofinance_edited =␣
↪stocks_yahoofinance_edited[stocks_yahoofinance_edited['f_test'] >= 100]

print(stocks_yahoofinance_edited.shape)
stocks_yahoofinance_edited
```

```
(39, 6)
```

[18]:

	alpha	beta	r-squared	share_ratio	treynor_ratio	f_test
Symbol						
MSFT	19.814749	1.373946	64.113859	1.563936	0.267977	1347.089665
CSCO	11.454632	1.259557	57.955342	1.166749	0.199144	1039.331263
MPWR	3.799481	1.613388	47.494265	0.689570	0.125856	682.033608
TXN	7.024910	1.333638	47.415005	0.847315	0.151439	679.869105
INTU	16.673324	1.223769	46.375700	1.287254	0.262849	652.078958
AAPL	11.614751	1.274979	44.646467	1.024879	0.195529	608.153341
INTC	0.243013	1.330235	40.711661	0.529017	0.086894	517.750922
AMAT	2.311300	1.664746	39.746756	0.582717	0.111324	497.384900
ADI	10.066974	1.257403	39.541214	0.909721	0.185387	493.130561
CCMP	15.456637	1.435710	39.473736	1.045820	0.235152	491.740187
JKHY	7.930429	0.804229	39.264237	0.998277	0.184960	487.443206
CGNX	5.345490	1.682517	37.589646	0.653258	0.153955	454.132867
MCHP	1.946082	1.438331	36.832062	0.559250	0.108339	439.643519
MKSI	1.398742	1.671800	35.220898	0.522691	0.111517	409.955623
NTAP	15.336362	1.469337	34.752846	0.966016	0.226620	401.605956

```
TER      12.283742  1.489663  33.916776  0.853566     0.192971       386.985501
CTXS     2.907192   0.897934  33.372209  0.618283     0.100226       377.659904
CDW      19.887590  1.039808  33.127767  1.337944     0.331396       373.523286
NVDA     9.108891   1.978557  32.668734  0.673375     0.221990       365.836358
SSNC     9.829231   1.187472  32.453747  0.836368     0.187046       362.272141
XLNX     15.645715  1.416737  30.907754  0.937600     0.232729       337.294667
BRKR     9.881420   1.198599  29.367811  0.794186     0.181697       313.501958
AVGO     7.082129   1.270464  28.857721  0.674043     0.153020       305.847973
CY       11.004322  1.550022  28.738191  0.737184     0.196883       304.070242
OTEX     2.347583   0.877016  27.336902  0.536440     0.086151       283.665638
KLIC     4.447761   1.425235  25.923383  0.540231     0.126745       263.865063
LOGI     17.253369  1.230360  25.615457  0.972608     0.256966       259.651457
MANT     6.805041   1.064949  25.399499  0.664815     0.170833       256.717068
GRMN     15.259400  0.893680  24.951526  1.080261     0.281347       250.683991
NATI     1.677464   1.011987  23.682464  0.460144     0.106941       233.977386
STX      10.534566  1.508952  23.087789  0.656271     0.179580       226.338481
OLED     8.116416   1.916836  21.825213  0.536761     0.185823       210.505342
QCOM     0.204630   1.157574  21.191717  0.381460     0.083778       202.752230
PRGS     3.118255   1.099358  19.352631  0.456898     0.127618       180.934400
ESLT     11.270688  0.708287  18.618599  0.893567     0.245280       172.501616
UBNT     25.958607  1.230820  14.461676  0.942967     0.372817       127.476233
TTEC     4.945958   0.962309  14.347634  0.462276     0.157403       126.302590
EQIX     5.137604   0.622902  13.790434  0.544330     0.121112       120.612916
AMSWA    5.218630   0.918608  13.379307  0.462036     0.163558       116.461751
```

[19]:
```
##################################
#   6. Merge metrics & Returns   #
##################################
annualized_returns = annualized_returns.transpose()
annualized_returns.index.names = ['Symbol']
annualized_returns
stocks_yahoofinance_edited = pd.merge(stocks_yahoofinance_edited,␣
 ↪annualized_returns, how='inner', on = 'Symbol')



# download stock from yahoo finance & metrics calculated
stocks_yahoofinance_edited.to_csv('e.csv')
create_download_link(filename = 'e.csv')

print(stocks_yahoofinance_edited.shape)
stocks_yahoofinance_edited
```

```
(39, 10)
```

[19]:
```
             alpha      beta  r-squared  ...     2 Year     3 Year        YTD
Symbol                                   ...
MSFT     19.814749  1.373946  64.113859  ...  42.099427  41.818563  35.910181
```

```
CSCO     11.454632  1.259557  57.955342  ...  38.497430  30.083349  34.366063
MPWR      3.799481  1.613388  47.494265  ...  16.652364  25.305480  12.435370
TXN       7.024910  1.333638  47.415005  ...  22.091591  25.196454  20.975974
INTU     16.673324  1.223769  46.375700  ...  39.146974  37.166676  36.021901
AAPL     11.614751  1.274979  44.646467  ...  18.733969  29.929505  27.479941
INTC      0.243013  1.330235  40.711661  ...  19.814989  16.558963   1.811100
AMAT      2.311300  1.664746  39.746756  ...   1.043818  23.532582  33.332760
ADI      10.066974  1.257403  39.541214  ...  21.115761  28.310638  32.544434
CCMP     15.456637  1.435710  39.473736  ...  23.775371  38.761013  17.029779
JKHY      7.930429  0.804229  39.264237  ...  16.250315  19.875007  10.217149
CGNX      5.345490  1.682517  37.589646  ...   1.463716  30.903264  21.813713
MCHP      1.946082  1.438331  36.832062  ...   4.678138  20.582779  20.001686
MKSI      1.398742  1.671800  35.220898  ...   3.012902  23.643435  18.468841
NTAP     15.336362  1.469337  34.752846  ...  29.870620  38.298077   6.315836
TER      12.283742  1.489663  33.916776  ...  20.225752  33.746227  47.866843
CTXS      2.907192  0.897934  33.372209  ...  11.558450  13.999668  -2.317207
CDW      19.887590  1.039808  33.127767  ...  32.762841  39.458785  34.175385
NVDA      9.108891  1.978557  32.668734  ...  -1.376552  48.921897  15.681567
SSNC      9.829231  1.187472  32.453747  ...  24.237226  27.211185  30.434097
XLNX     15.645715  1.416737  30.907754  ...  34.166345  37.971514  35.627288
BRKR      9.881420  1.198599  29.367811  ...  31.263182  26.778148  63.711698
AVGO      7.082129  1.270464  28.857721  ...   9.476023  24.440682  10.721518
CY       11.004322  1.550022  28.738191  ...  33.608840  35.517235  76.609592
OTEX      2.347583  0.877016  27.336902  ...  15.615187  12.555602  28.586478
KLIC      4.447761  1.425235  25.923383  ...   5.636393  23.064133  11.063502
LOGI     17.253369  1.230360  25.615457  ...   5.252514  36.616110  26.598458
MANT      6.805041  1.064949  25.399499  ...  31.759313  23.192800  24.875809
GRMN     15.259400  0.893680  24.951526  ...  32.273455  30.143393  30.837133
NATI      1.677464  1.011987  23.682464  ...   3.394344  15.822270 -10.095290
STX      10.534566  1.508952  23.087789  ...  11.396847  32.097704  23.027980
OLED      8.116416  1.916836  21.825213  ...  25.778624  40.619199  98.907035
QCOM      0.204630  1.157574  21.191717  ...  18.207489  14.697894  30.483966
PRGS      3.118255  1.099358  19.352631  ...  22.002315  19.029790  20.409672
ESLT     11.270688  0.708287  18.618599  ...  14.750963  22.372827  40.836914
UBNT     25.958607  1.230820  14.461676  ...  66.064812  50.887023  36.729617
TTEC      4.945958  0.962309  14.347634  ...   8.120352  20.147038  59.994754
EQIX      5.137604  0.622902  13.790434  ...  10.227479  12.544081  45.772452
AMSWA     5.218630  0.918608  13.379307  ...  20.537518  20.024601  44.065993

[39 rows x 10 columns]
```

[20]:

```python
###########################################
#   7. LSTM & Stock movement prediciton  #
###########################################
# Step 1 : import necessary libraries
import numpy
import matplotlib.pyplot as plt
```

```
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Flatten
from keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error


# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
                a = dataset[i:(i+look_back), 0]
                dataX.append(a)
                dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)


def LSTM_model(batch_size, look_back):
  model = Sequential()
  model.add(  LSTM(4, batch_input_shape = (batch_size, look_back, 1), stateful␣
 ↪= True, return_sequences = True)  )
  model.add(  LSTM(4, batch_input_shape = (batch_size, look_back, 1), stateful␣
 ↪= True)  )
  model.add( Dense(1) )
  model.compile(Adam(lr=0.01), loss='mean_squared_error')
  return model

weekly = [0] * stocks_yahoofinance_edited.shape[0]
monthly = [0] * stocks_yahoofinance_edited.shape[0]
ptr = 0

# Step 2 :
for item in stocks_yahoofinance_edited.index:
  # Step 2.0 :  fix random seed for reproducibility
  numpy.random.seed(7)

  # Step 2.1 : load data (adjusted closed prices) & normalize
  data = stocks_yahoofinance[item].values
  data = data.astype('float32')
  data = data.reshape(-1,1)
  scaler = MinMaxScaler(feature_range = (0,1))
  data = scaler.fit_transform(data)
```

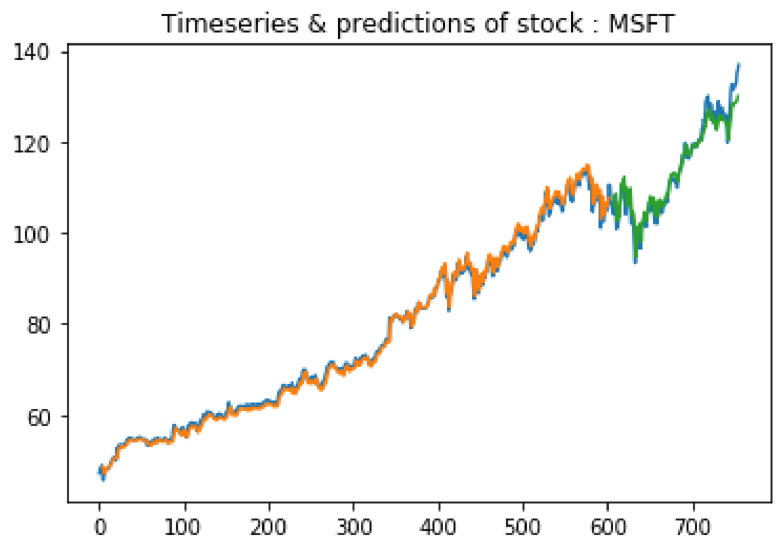
```
# Step 2.2 : split into train & test sets
train_size = int(len(data) * 0.8)
# we will analyse weekly & and montthly predictions to extract resulst
# for our technical analysis (22 & 5 trading days respectively)
test_size_monthly = 27
test_size_weekly = 9
train_data, test_data = data[0:train_size, :], data[train_size : len(data), :]

# Step 2.3 :
# reshape into X=t and Y=t+1
look_back = 4
trainX, trainY = create_dataset(train_data, look_back)
testX, testY = create_dataset(test_data, look_back)

# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))

# Step 2.4 : Create and fit the LSTM network
batch_size = 1
model = LSTM_model(batch_size, look_back)

# Step 2.5 : Train model
for i in range(10):
  model.fit(trainX, trainY, epochs = 1, batch_size = batch_size,verbose=0␣
↪,shuffle = True)
  model.reset_states()

# Step 2.6 : Make predictions
trainPredict = model.predict(trainX, batch_size=batch_size)
model.reset_states()
testPredict = model.predict(testX, batch_size=batch_size)

# Step 2.7 : invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])

testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# Step 2.8 : calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))

testMonthlyScore = math.sqrt(mean_squared_error(testY[0][-test_size_monthly :␣
↪], testPredict[-test_size_monthly : ,0]))
print('Monthly Test Score: %.2f RMSE' % (testMonthlyScore))
```

```
testWeeklyScore = math.sqrt(mean_squared_error(testY[0][-test_size_weekly :␣
↪], testPredict[-test_size_weekly:,0]))
print('Weekly Test Score: %.2f RMSE' % (testMonthlyScore))

MonthlyME = np.mean( testPredict[-test_size_monthly:,0] -␣
↪testY[0][-test_size_monthly:])
WeeklyME = np.mean( testPredict[-test_size_weekly:,0] -␣
↪testY[0][-test_size_weekly:])
print('Weekly Mean Error: %.2f ' % (MonthlyME))
print('Monly Mean Error: %.2f ' % (WeeklyME))

weekly[ptr] = WeeklyME
monthly[ptr] = MonthlyME

# Step 2.9 : Shift predictions for plotting
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(data)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(data)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(data)-1, :] =␣
↪testPredict

# Step 2.10 : ploting baseline & monthly, weekly predictions
plt.plot(scaler.inverse_transform(data))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.title("Timeseries & predictions of stock : " + item)
plt.show()

ptr+=1
```

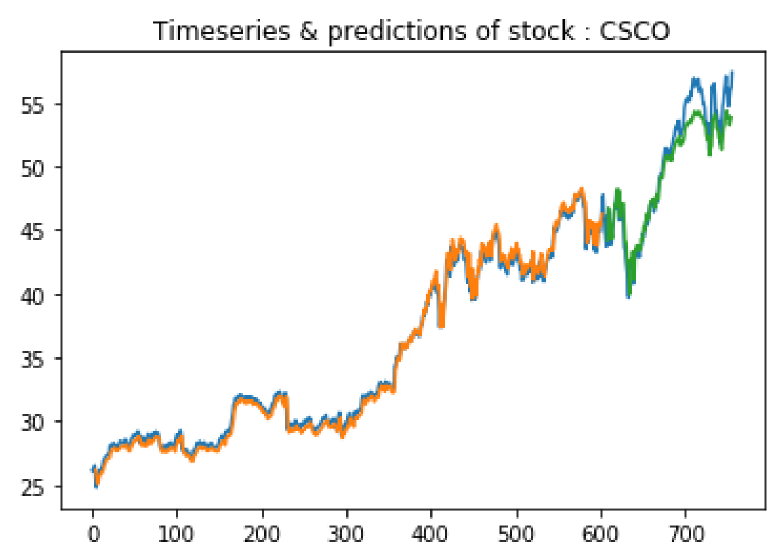
```
Using TensorFlow backend.
WARNING: Logging before flag parsing goes to stderr.
W0627 23:50:21.317102 139944323762048 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:74:
The name tf.get_default_graph is deprecated. Please use
tf.compat.v1.get_default_graph instead.

W0627 23:50:21.353706 139944323762048 deprecation_wrapper.py:119] From
```

```
trainPredictPlot = numpy.empty_like(data)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(data)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(data)-1, :] = testPredict

# Step 2.10 : ploting baseline & monthly, weekly predictions
plt.plot(scaler.inverse_transform(data))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.title("Timeseries & predictions of stock : " + item)
plt.show()

ptr+=1
```

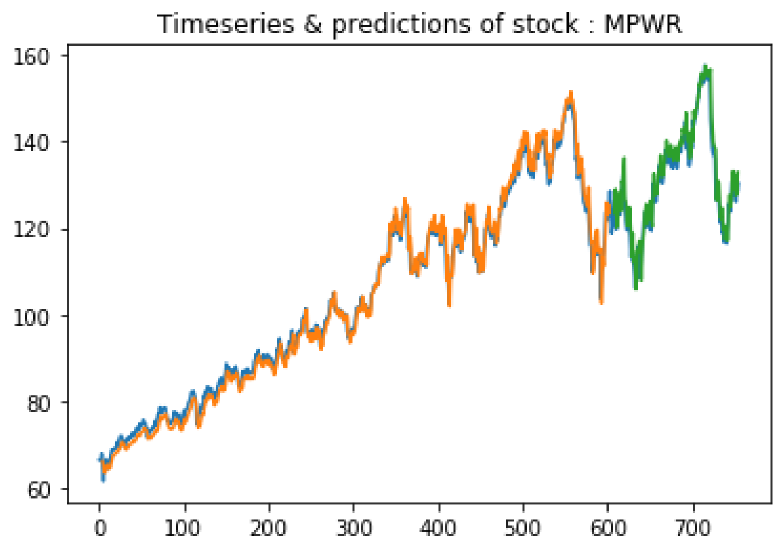
```
Train Score: 1.31 RMSE
Monthly Test Score: 3.41 RMSE
Weekly Test Score: 3.41 RMSE
Weekly Mean Error: -2.53
Monly Mean Error: -4.69
```



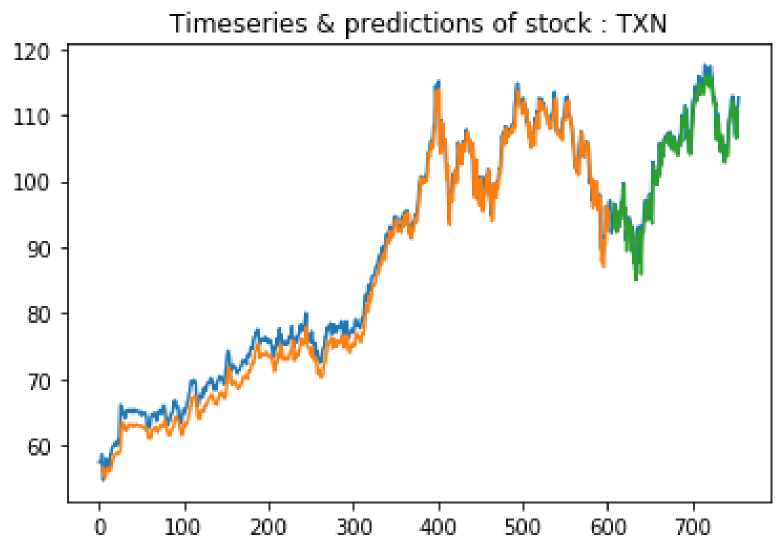

```
Train Score: 0.60 RMSE
Monthly Test Score: 2.04 RMSE
Weekly Test Score: 2.04 RMSE
Weekly Mean Error: -1.69
Monly Mean Error: -2.22
```



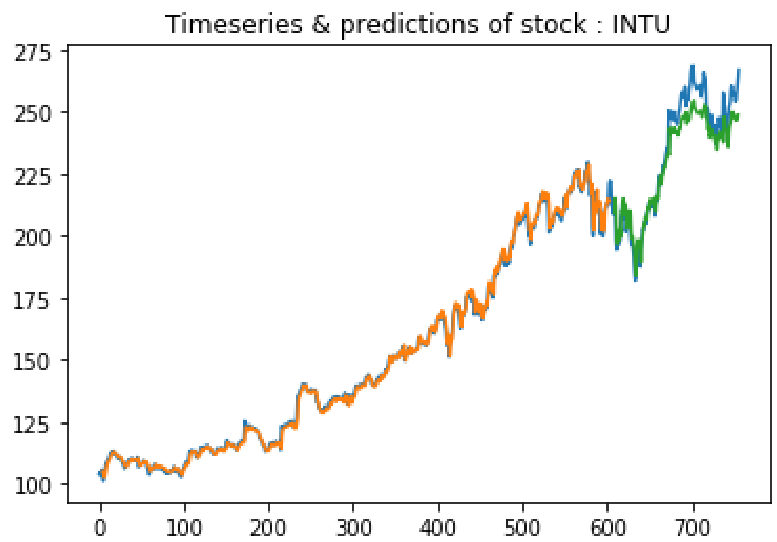
```
Train Score: 2.33 RMSE
Monthly Test Score: 3.83 RMSE
Weekly Test Score: 3.83 RMSE
Weekly Mean Error: 1.76
Monly Mean Error: 1.26
```



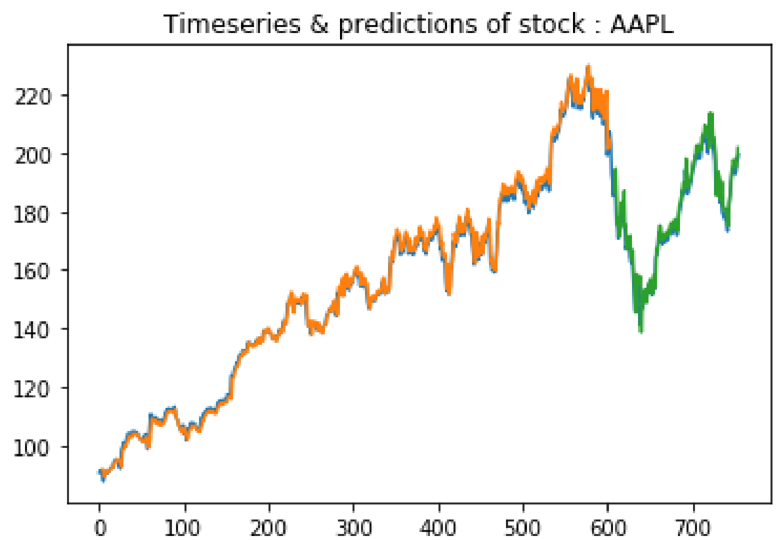

```
Train Score: 2.15 RMSE
Monthly Test Score: 2.08 RMSE
Weekly Test Score: 2.08 RMSE
Weekly Mean Error: -0.39
Monly Mean Error: -0.67
```



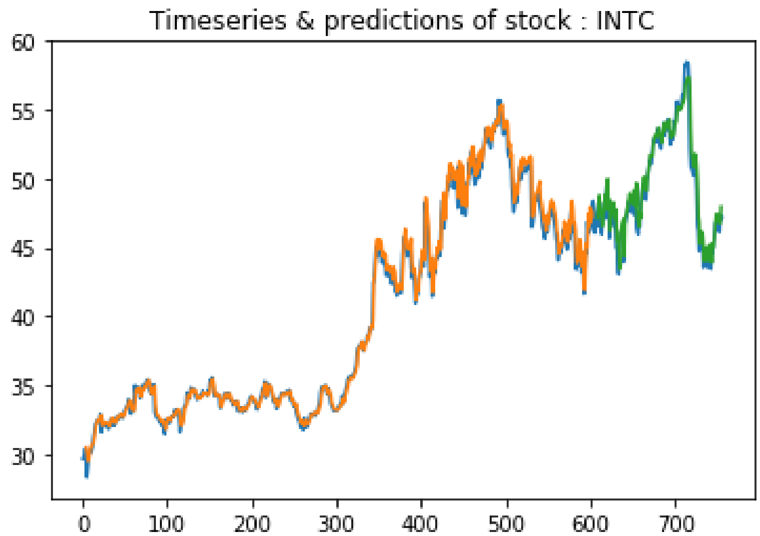

```
Train Score: 2.28 RMSE
Monthly Test Score: 8.65 RMSE
Weekly Test Score: 8.65 RMSE
Weekly Mean Error: -7.10
Monly Mean Error: -9.94
```



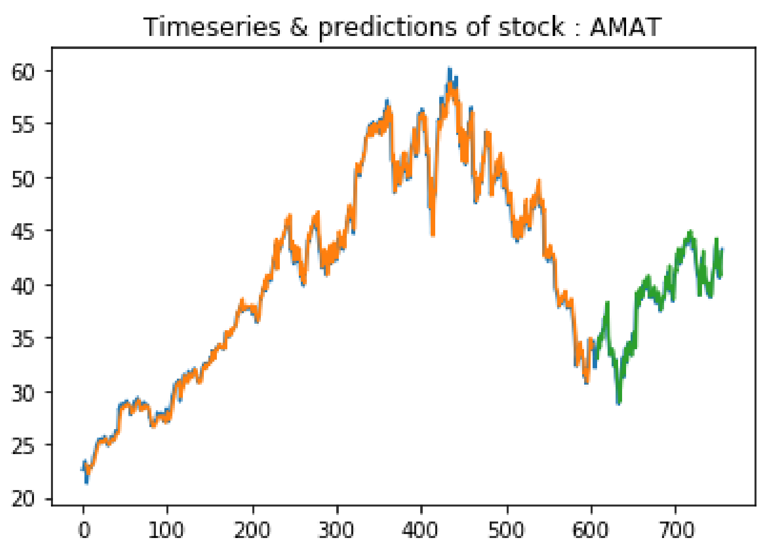
```
Train Score: 2.57 RMSE
Monthly Test Score: 4.41 RMSE
Weekly Test Score: 4.41 RMSE
Weekly Mean Error: 2.40
Monly Mean Error: 1.51
```



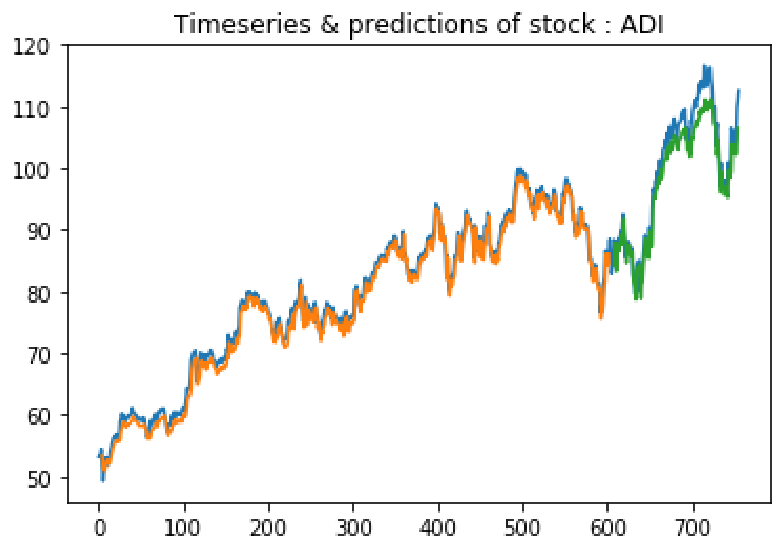

```
Train Score: 0.75 RMSE
Monthly Test Score: 0.90 RMSE
Weekly Test Score: 0.90 RMSE
Weekly Mean Error: 0.49
Monly Mean Error: 0.36
```



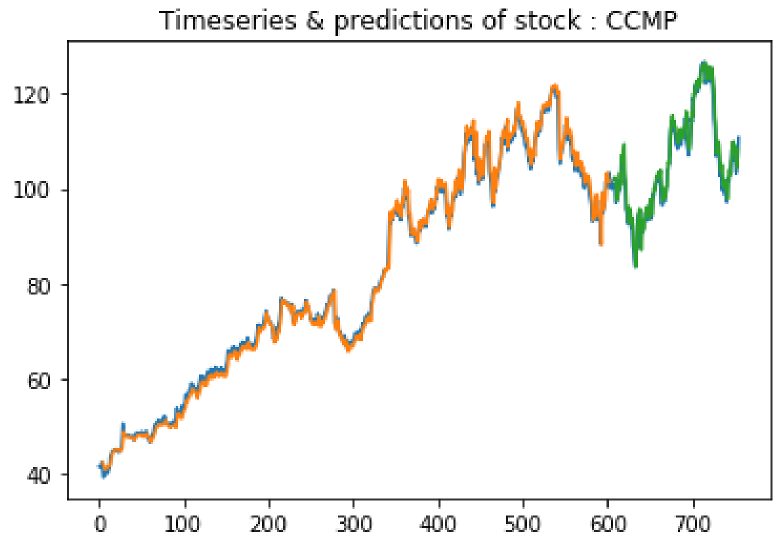

```
Train Score: 0.92 RMSE
Monthly Test Score: 1.16 RMSE
Weekly Test Score: 1.16 RMSE
Weekly Mean Error: 0.25
Monly Mean Error: 0.25
```



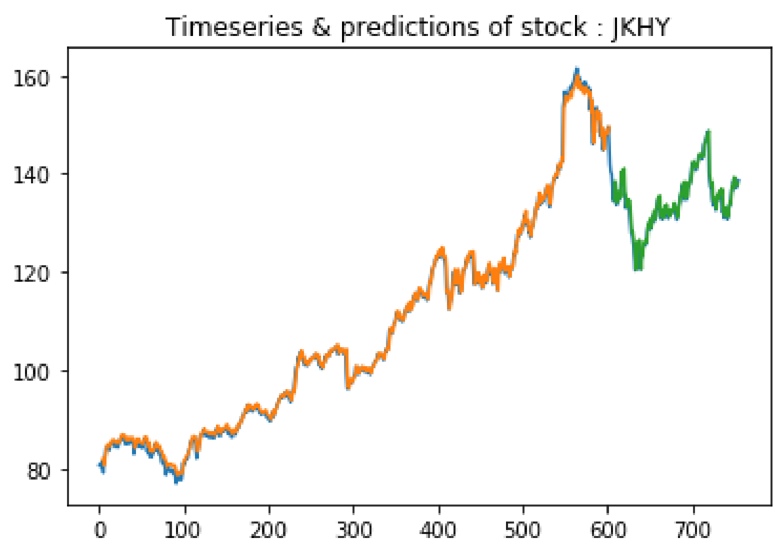
```
Train Score: 1.50 RMSE
Monthly Test Score: 3.05 RMSE
Weekly Test Score: 3.05 RMSE
Weekly Mean Error: -1.67
Monly Mean Error: -3.29
```



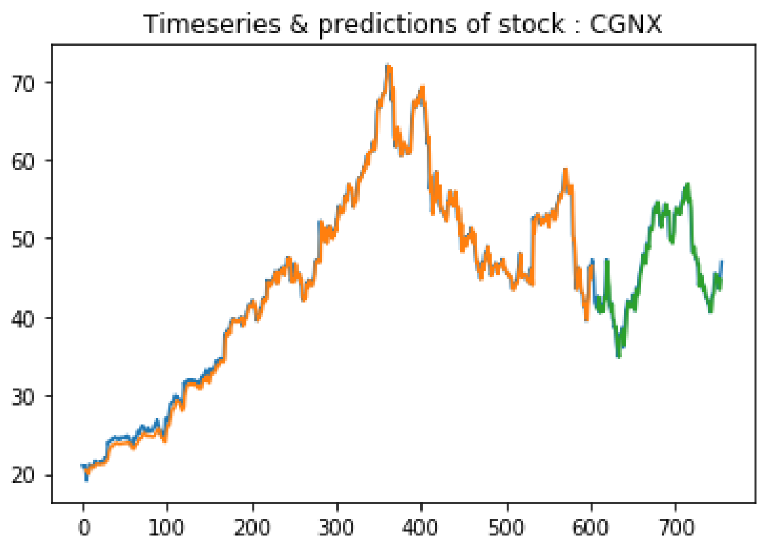

```
Train Score: 1.79 RMSE
Monthly Test Score: 2.68 RMSE
Weekly Test Score: 2.68 RMSE
Weekly Mean Error: 1.43
Monly Mean Error: 1.10
```



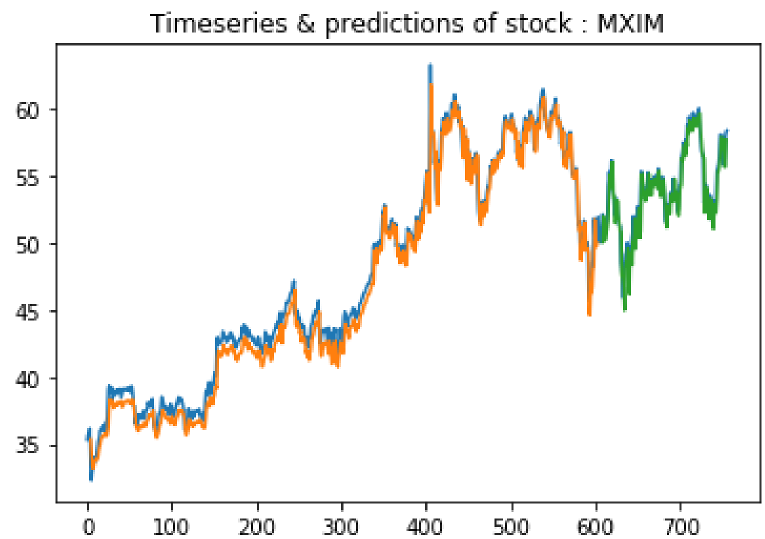

```
Train Score: 1.22 RMSE
Monthly Test Score: 1.37 RMSE
Weekly Test Score: 1.37 RMSE
Weekly Mean Error: 0.42
Monly Mean Error: 0.03
```



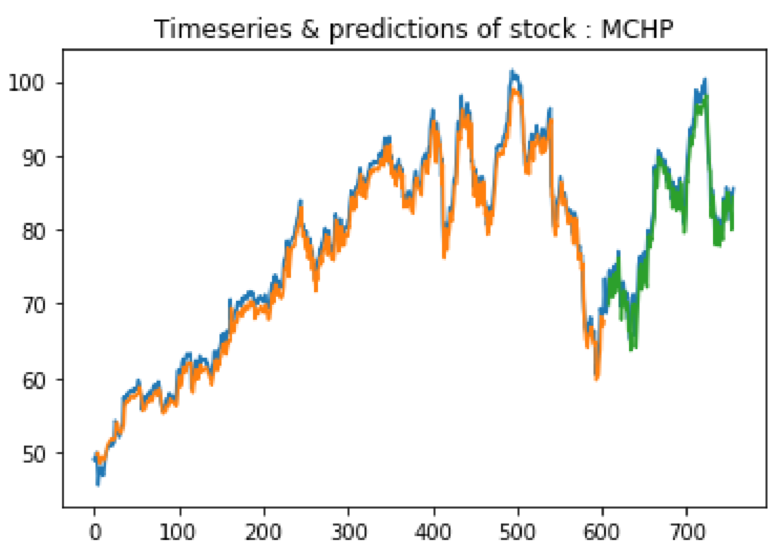
```
Train Score: 1.08 RMSE
Monthly Test Score: 0.97 RMSE
Weekly Test Score: 0.97 RMSE
Weekly Mean Error: -0.06
Monly Mean Error: -0.39
```



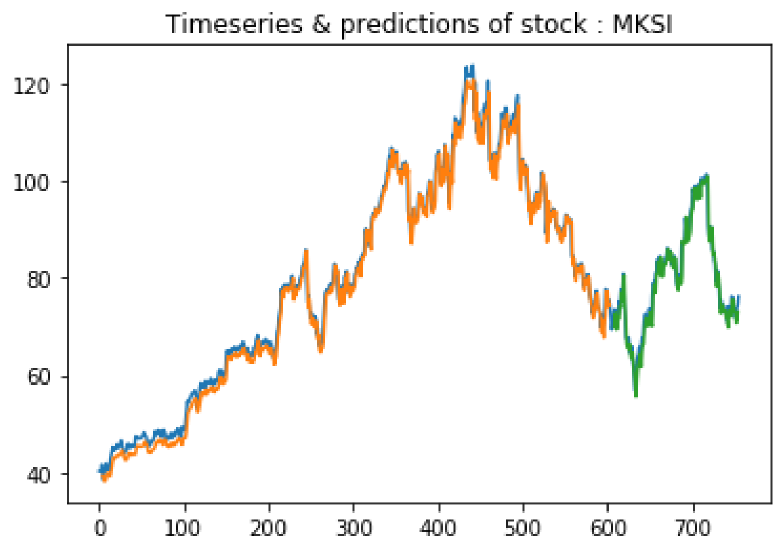

```
Train Score: 1.07 RMSE
Monthly Test Score: 1.20 RMSE
Weekly Test Score: 1.20 RMSE
Weekly Mean Error: -0.31
Monly Mean Error: -0.46
```



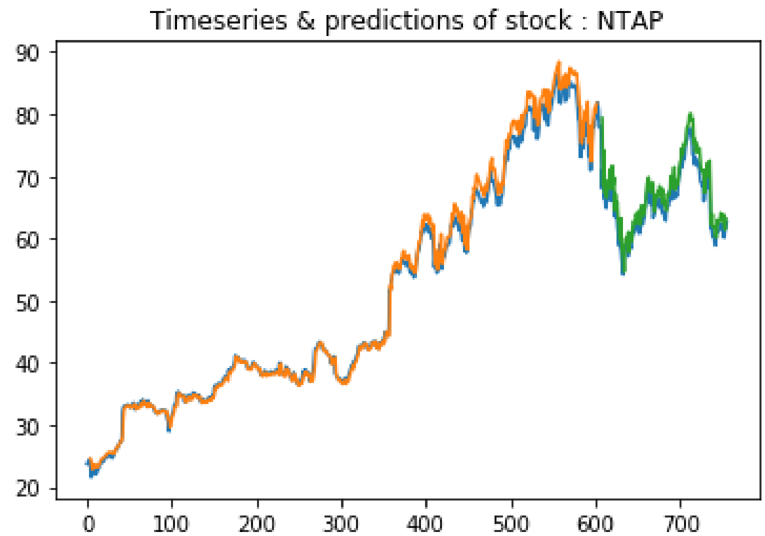

```
Train Score: 1.78 RMSE
Monthly Test Score: 2.32 RMSE
Weekly Test Score: 2.32 RMSE
Weekly Mean Error: -0.88
Monly Mean Error: -1.12
```



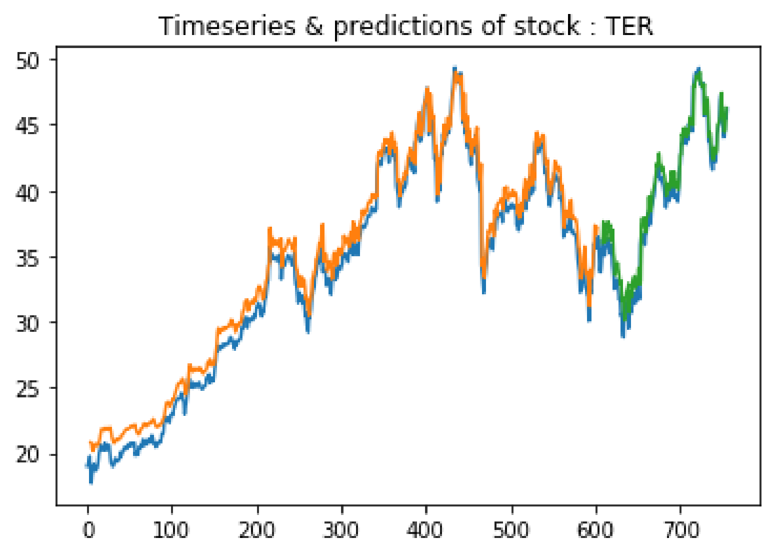
```
Train Score: 2.24 RMSE
Monthly Test Score: 1.84 RMSE
Weekly Test Score: 1.84 RMSE
Weekly Mean Error: -0.55
Monly Mean Error: -0.97
```



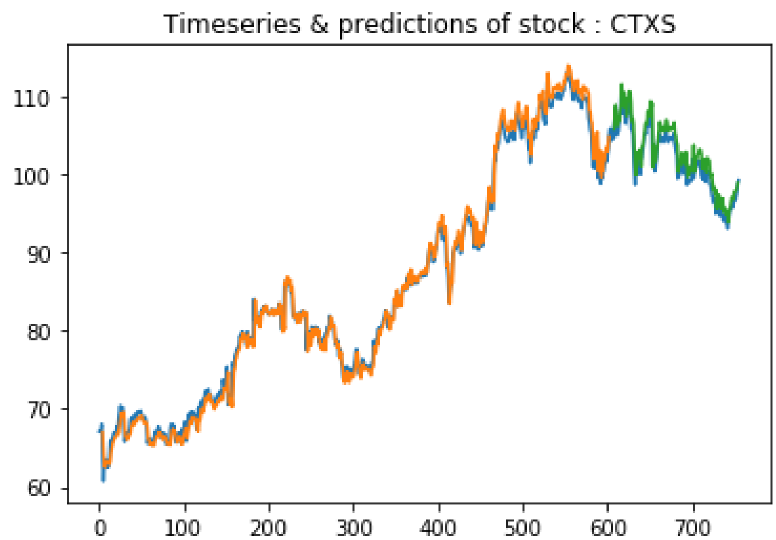

```
Train Score: 1.51 RMSE
Monthly Test Score: 2.55 RMSE
Weekly Test Score: 2.55 RMSE
Weekly Mean Error: 1.87
Monly Mean Error: 1.44
```



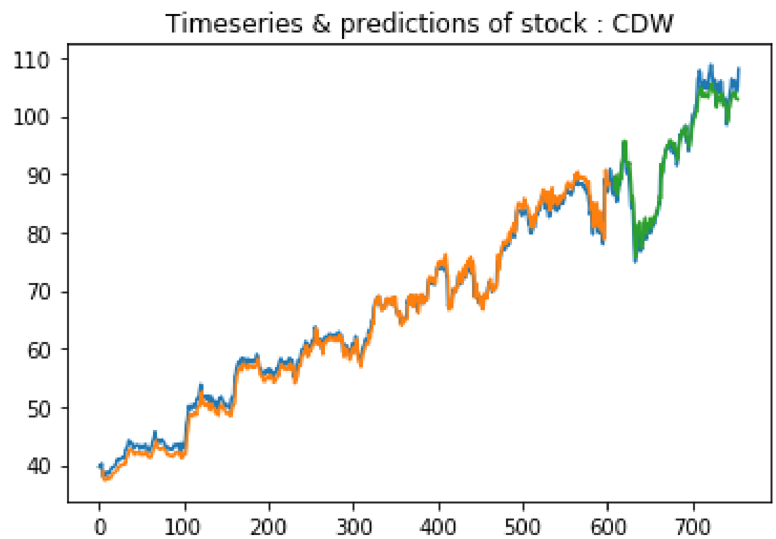

```
Train Score: 1.29 RMSE
Monthly Test Score: 1.10 RMSE
Weekly Test Score: 1.10 RMSE
Weekly Mean Error: 0.53
Monly Mean Error: 0.28
```



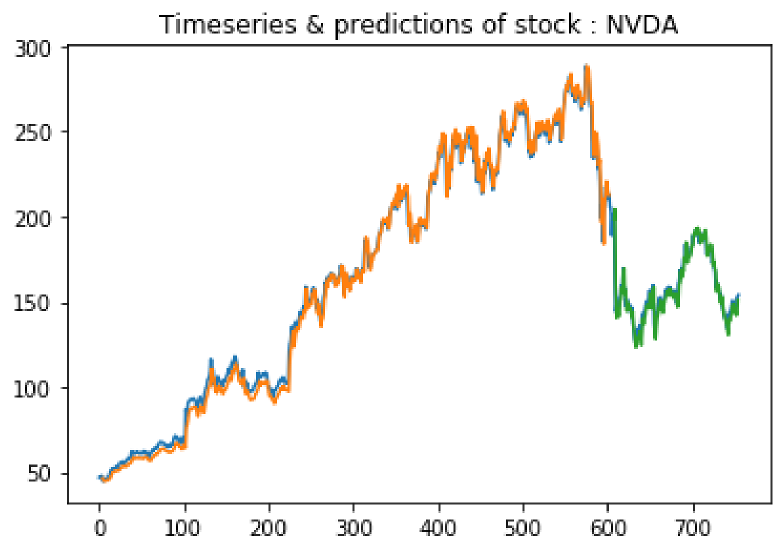
```
Train Score: 1.28 RMSE
Monthly Test Score: 1.25 RMSE
Weekly Test Score: 1.25 RMSE
Weekly Mean Error: 1.03
Monly Mean Error: 0.82
```



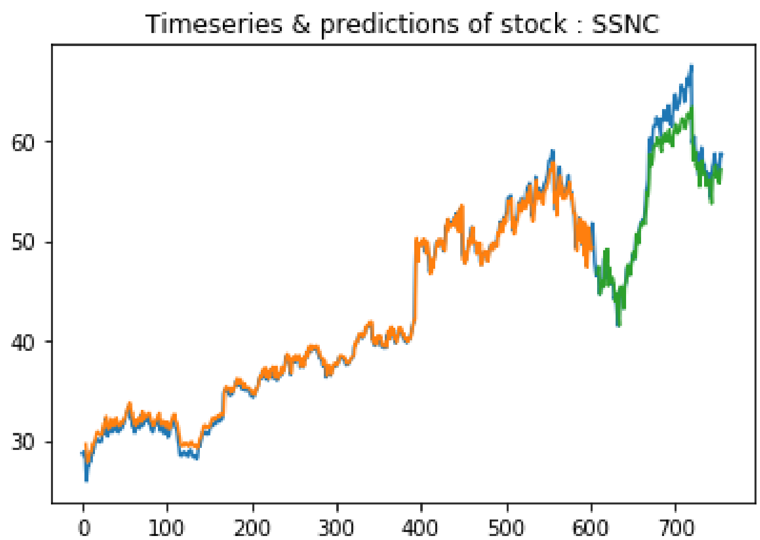

```
Train Score: 1.37 RMSE
Monthly Test Score: 2.02 RMSE
Weekly Test Score: 2.02 RMSE
Weekly Mean Error: -1.29
Monly Mean Error: -1.86
```



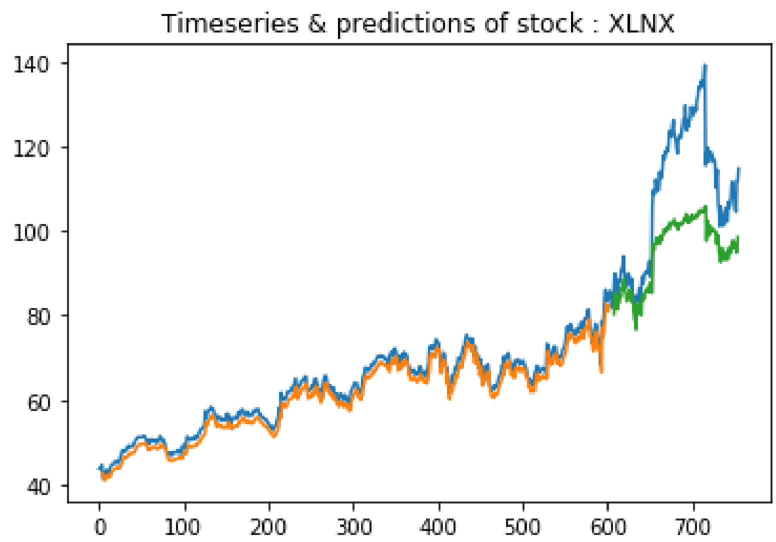

```
Train Score: 5.64 RMSE
Monthly Test Score: 4.68 RMSE
Weekly Test Score: 4.68 RMSE
Weekly Mean Error: -1.68
Monly Mean Error: -3.33
```



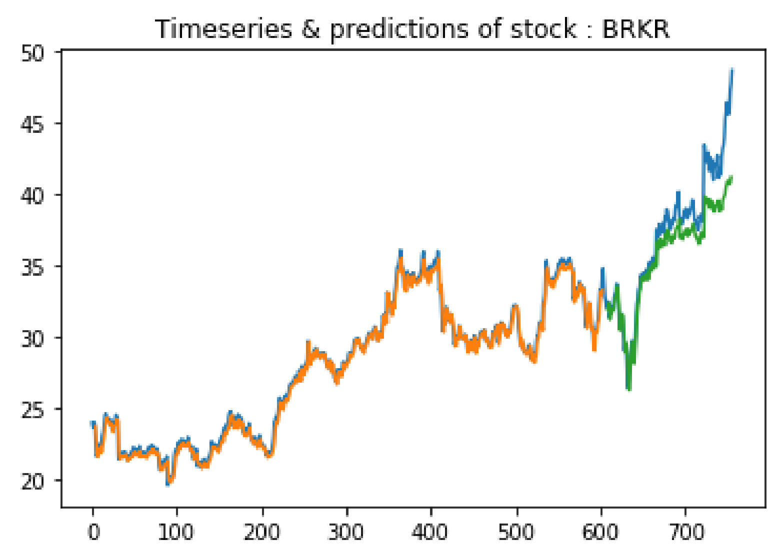
```
Train Score: 0.78 RMSE
Monthly Test Score: 1.28 RMSE
Weekly Test Score: 1.28 RMSE
Weekly Mean Error: -0.89
Monly Mean Error: -1.07
```



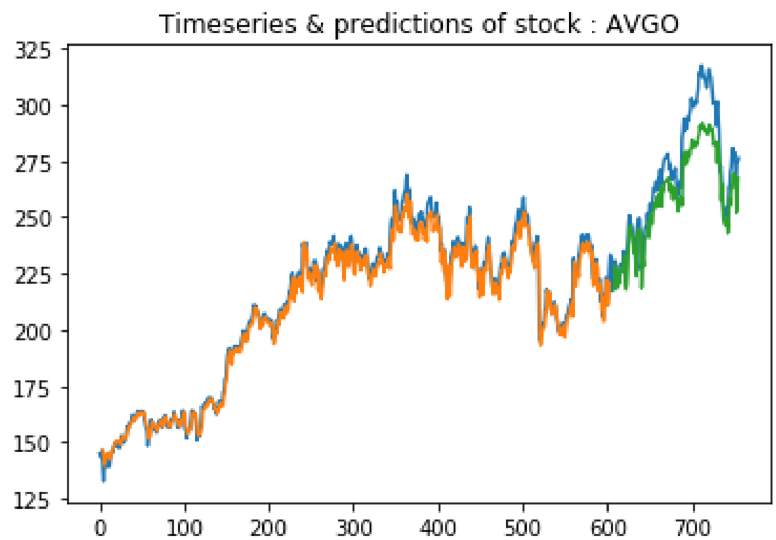

```
Train Score: 2.16 RMSE
Monthly Test Score: 11.23 RMSE
Weekly Test Score: 11.23 RMSE
Weekly Mean Error: -10.87
Monly Mean Error: -12.37
```



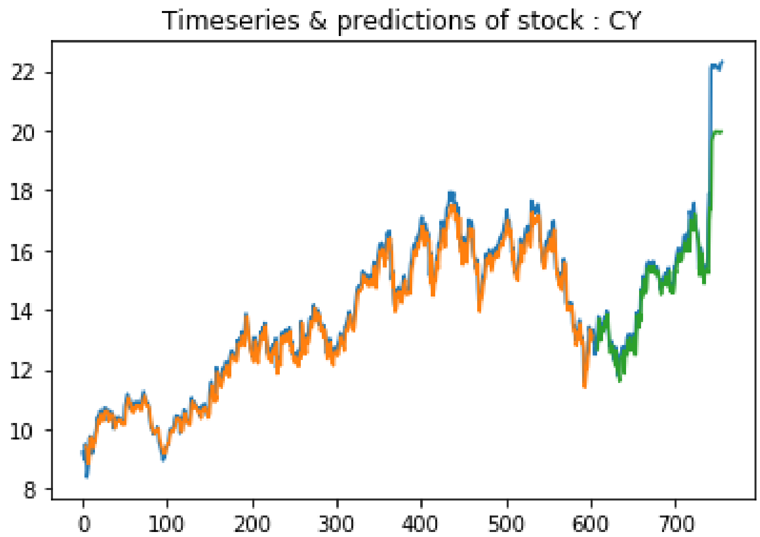

```
Train Score: 0.50 RMSE
Monthly Test Score: 3.92 RMSE
Weekly Test Score: 3.92 RMSE
Weekly Mean Error: -3.67
Monly Mean Error: -5.35
```



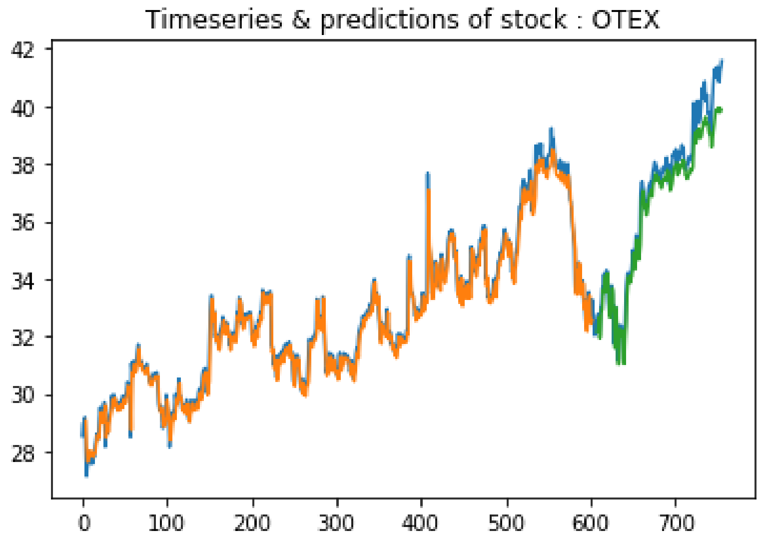
```
Train Score: 4.78 RMSE
Monthly Test Score: 11.33 RMSE
Weekly Test Score: 11.33 RMSE
Weekly Mean Error: -9.04
Monly Mean Error: -9.99
```



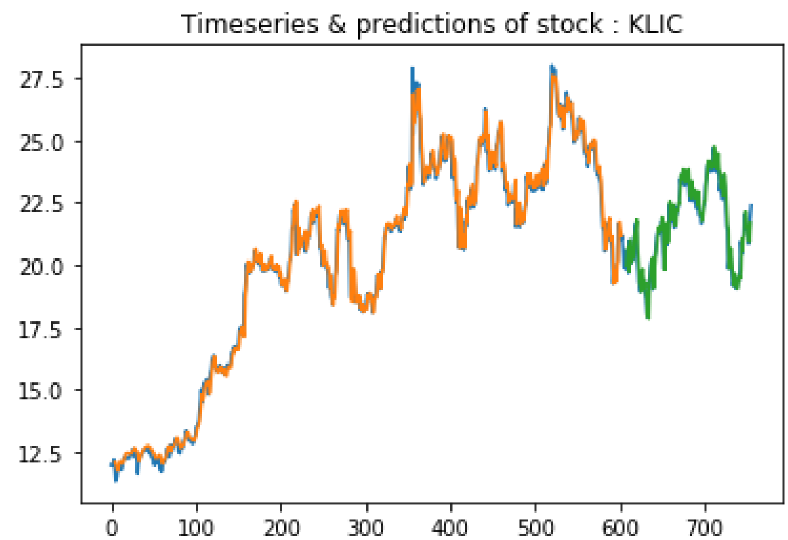

```
Train Score: 0.33 RMSE
Monthly Test Score: 1.82 RMSE
Weekly Test Score: 1.82 RMSE
Weekly Mean Error: -1.34
Monly Mean Error: -2.19
```



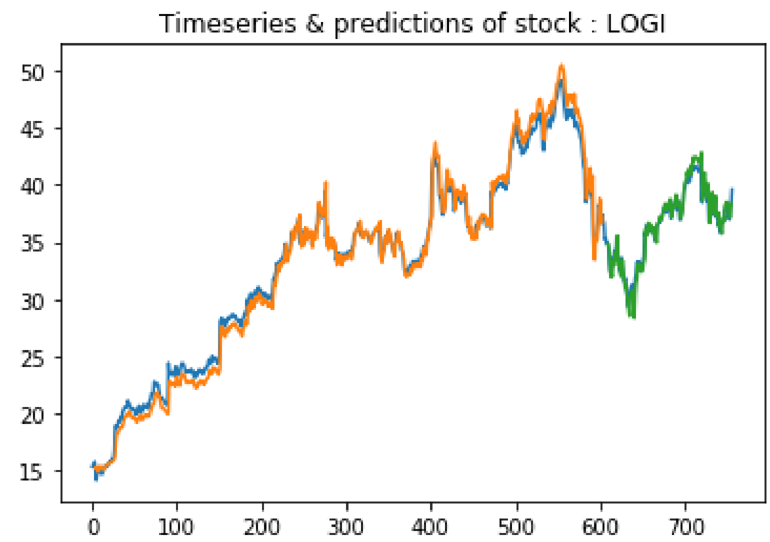

```
Train Score: 0.47 RMSE
Monthly Test Score: 1.15 RMSE
Weekly Test Score: 1.15 RMSE
Weekly Mean Error: -1.06
Monly Mean Error: -1.40
```



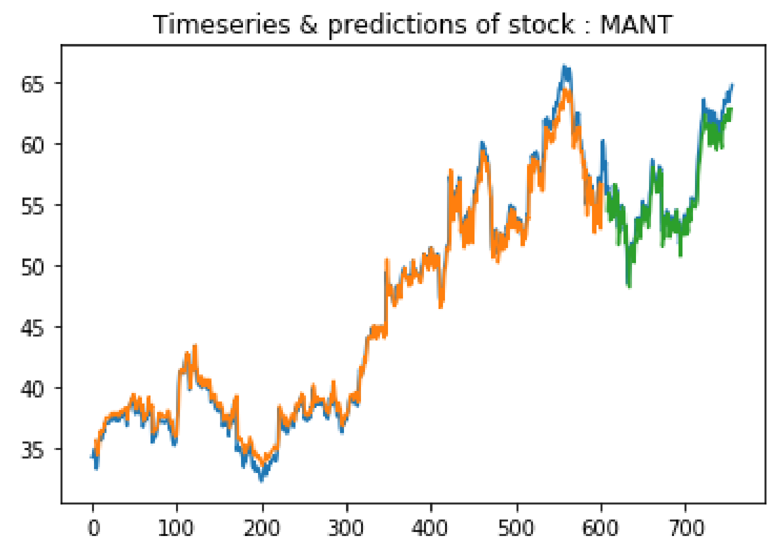
```
Train Score: 0.48 RMSE
Monthly Test Score: 0.55 RMSE
Weekly Test Score: 0.55 RMSE
Weekly Mean Error: 0.04
Monly Mean Error: 0.01
```



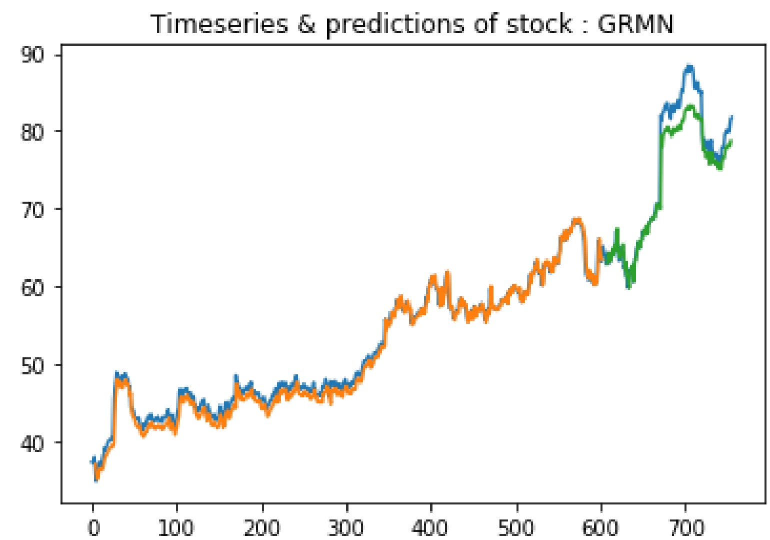

```
Train Score: 0.97 RMSE
Monthly Test Score: 0.87 RMSE
Weekly Test Score: 0.87 RMSE
Weekly Mean Error: 0.27
Monly Mean Error: 0.12
```



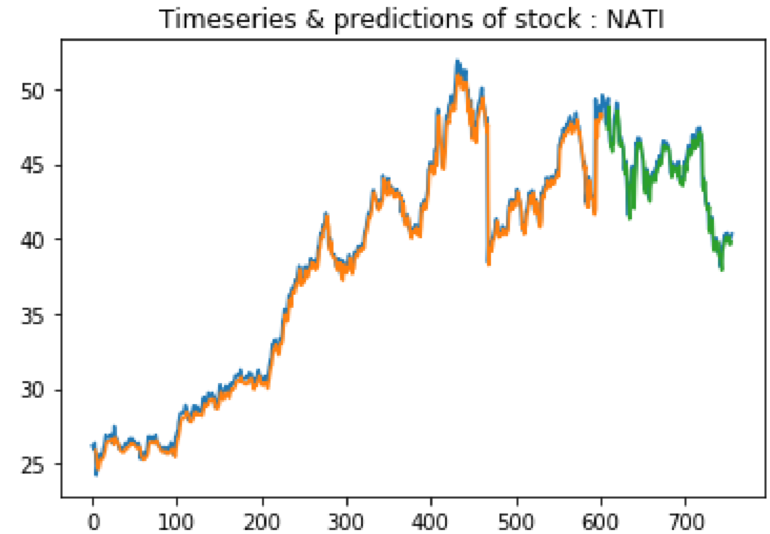

```
Train Score: 0.98 RMSE
Monthly Test Score: 1.49 RMSE
Weekly Test Score: 1.49 RMSE
Weekly Mean Error: -1.27
Monly Mean Error: -1.53
```



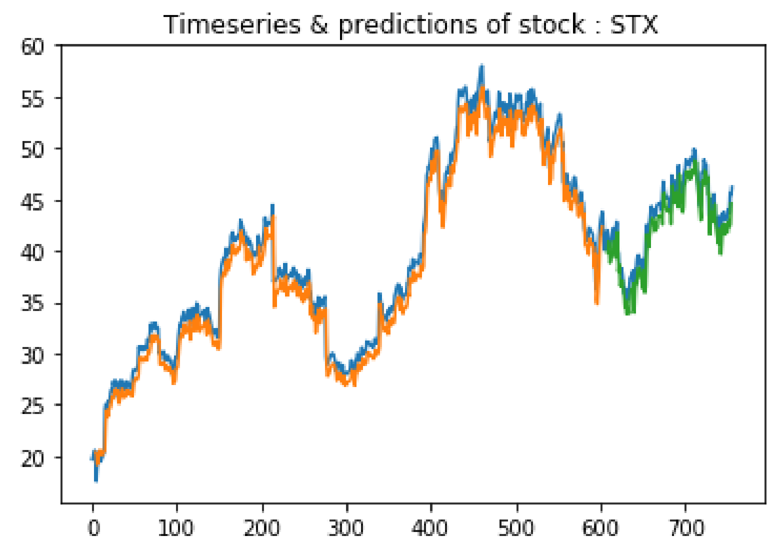
```
Train Score: 0.99 RMSE
Monthly Test Score: 1.66 RMSE
Weekly Test Score: 1.66 RMSE
Weekly Mean Error: -1.43
Monly Mean Error: -2.19
```



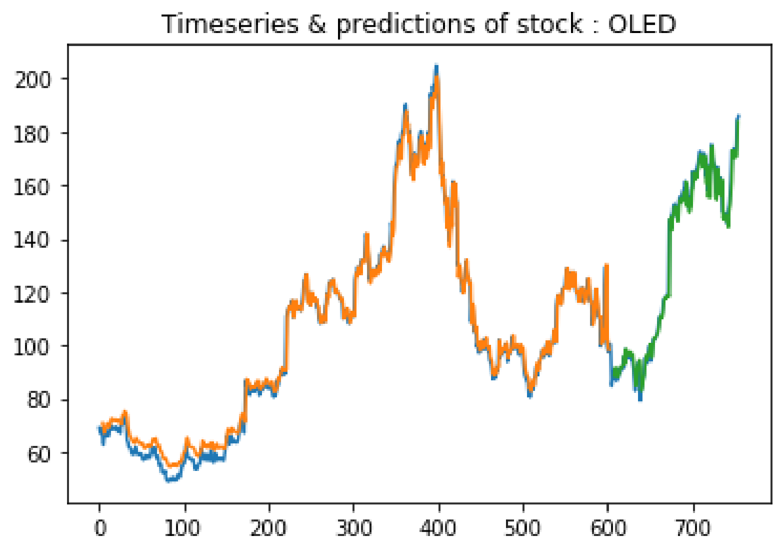

```
Train Score: 0.76 RMSE
Monthly Test Score: 0.63 RMSE
Weekly Test Score: 0.63 RMSE
Weekly Mean Error: -0.14
Monly Mean Error: -0.23
```



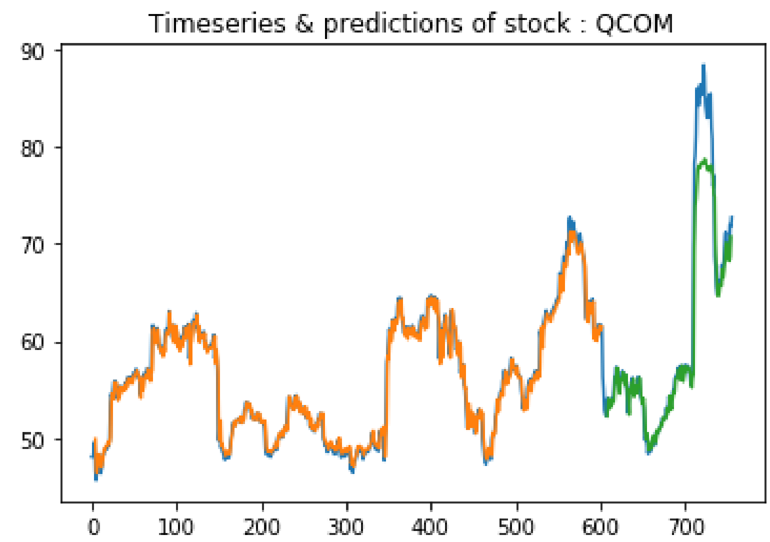

```
Train Score: 1.56 RMSE
Monthly Test Score: 1.63 RMSE
Weekly Test Score: 1.63 RMSE
Weekly Mean Error: -1.19
Monly Mean Error: -1.37
```



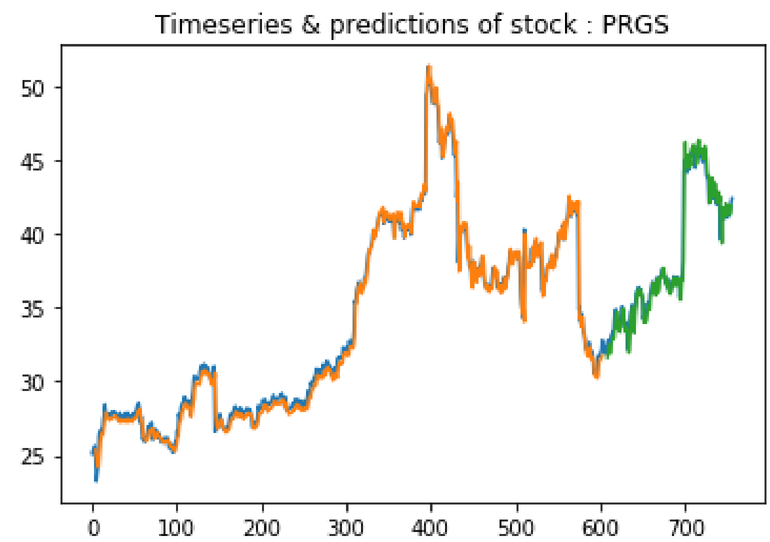
```
Train Score: 4.26 RMSE
Monthly Test Score: 5.07 RMSE
Weekly Test Score: 5.07 RMSE
Weekly Mean Error: -1.89
Monly Mean Error: -4.18
```



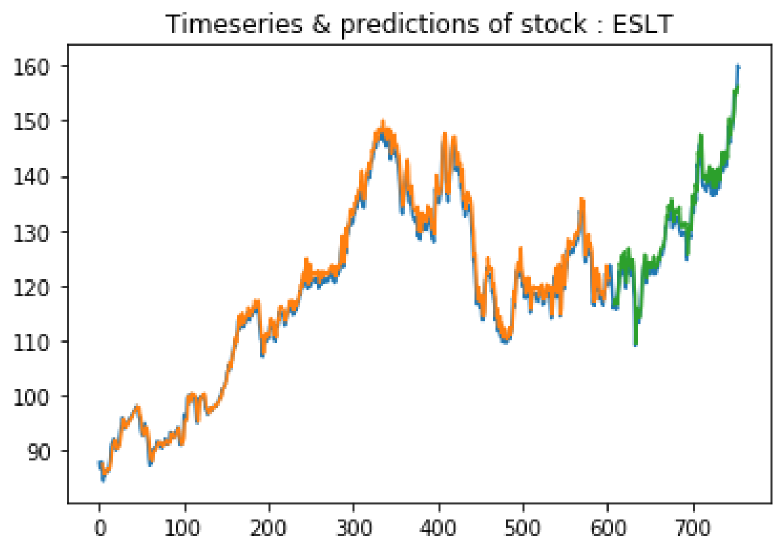

```
Train Score: 1.01 RMSE
Monthly Test Score: 3.09 RMSE
Weekly Test Score: 3.09 RMSE
Weekly Mean Error: -1.23
Monly Mean Error: -1.18
```



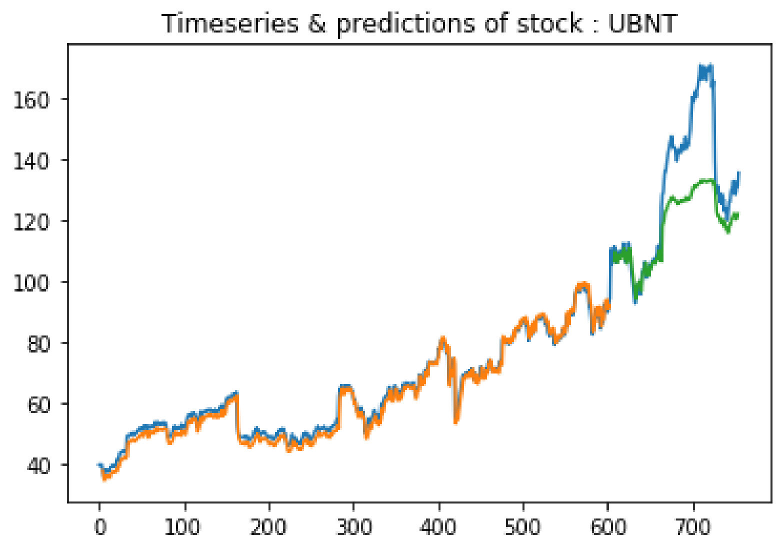

```
Train Score: 0.79 RMSE
Monthly Test Score: 0.88 RMSE
Weekly Test Score: 0.88 RMSE
Weekly Mean Error: 0.24
Monly Mean Error: 0.07
```



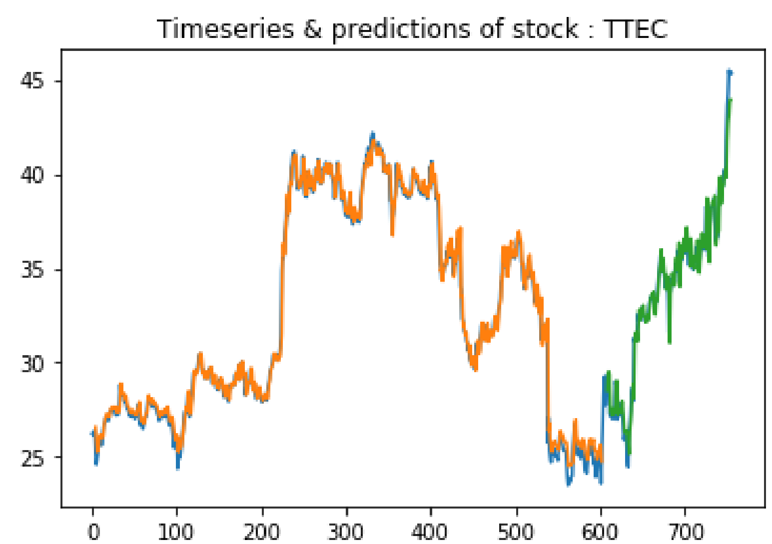
```
Train Score: 1.91 RMSE
Monthly Test Score: 2.04 RMSE
Weekly Test Score: 2.04 RMSE
Weekly Mean Error: 0.61
Monly Mean Error: -0.76
```



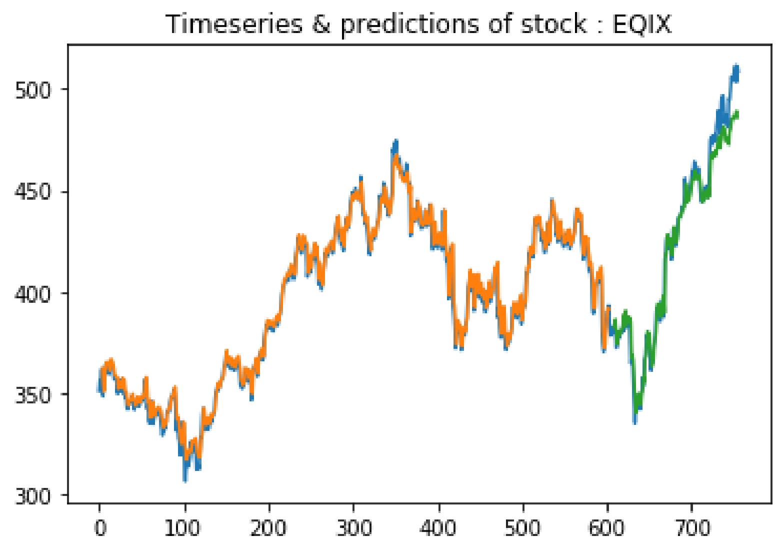

```
Train Score: 2.30 RMSE
Monthly Test Score: 7.92 RMSE
Weekly Test Score: 7.92 RMSE
Weekly Mean Error: -7.55
Monly Mean Error: -9.75
```



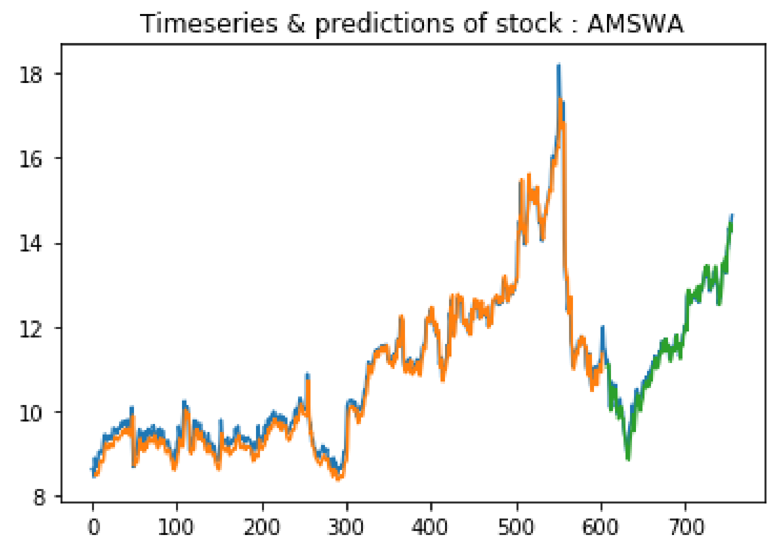

```
Train Score: 0.61 RMSE
Monthly Test Score: 1.23 RMSE
Weekly Test Score: 1.23 RMSE
Weekly Mean Error: -0.40
Monly Mean Error: -1.27
```



```
Train Score: 5.49 RMSE
Monthly Test Score: 16.30 RMSE
Weekly Test Score: 16.30 RMSE
Weekly Mean Error: -15.00
Monly Mean Error: -20.86
```




```
Train Score: 0.29 RMSE
Monthly Test Score: 0.21 RMSE
Weekly Test Score: 0.21 RMSE
Weekly Mean Error: 0.03
Monly Mean Error: -0.03
```



[333]:
```
#####################################################################
#   8. Convertion of technical predictions into recomendations   #
#####################################################################
# In this section, we use the arrays weekly, monthly that occured as the ME
# (mean error) of our predictions for last's week, month data, and after
# we normalize them between (-1,1) we classify the result as 'Strong Buy',␣
↪'Buy'
# ..., and more specifically as 0 (Strong Sell) to 4 (Strong Buy)

weekly = []
monthly = []

scaler = MinMaxScaler(feature_range = (-1,1))
weekly = np.array(weekly).reshape(-1,1)
monthly = np.array(monthly).reshape(-1,1)

weekly = scaler.fit_transform(weekly).reshape(1,-1)[0].tolist()
monthly = scaler.fit_transform(monthly).reshape(1,-1)[0].tolist()



print(np.mean(weekly))
print(np.mean(monthly))
```

```
    1457                                                self._handle, args,
 -> 1458                                                run_metadata_ptr)
    1459         if run_metadata:
    1460           proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)


KeyboardInterrupt:
```

[21]:
```
#################################################################
#   8. Convertion of technical predictions into recomendations  #
#################################################################
# In this section, we use the arrays weekly, monthly that occured as the ME
# (mean error) of our predictions for last's week, month data, and after
# we normalize them between (-1,1) we classify the result as 'Strong Buy',␣
 ↪'Buy'
# ..., and more specifically as 0 (Strong Sell) to 4 (Strong Buy)

weekly = [-2.53,-1.69, 1.76, -0.39, -7.10, 2.4, 0.49,0.25,
         -1.67, 1.43, 0.42, -0.06,-0.88, -0.55,
         1.87, 0.53,1.03,-1.29,-1.68,-0.89,-10.87,-3.67,
         -9.04,-1.34,-1.06,0.04,0.27,-1.27,-1.43,-0.14,
         -1.19,-1.89,-1.23,0.24,0.61,-7.55,-0.4,-15.0,
         0.03]
monthly = [-4.69,-2.22, 1.26, -0.67, -9.94, 1.51, 0.36, 0.25,
          -3.29, 1.1, 0.03, -0.39,-1.12, -0.97,
          1.44, 0.28,0.82,-1.86,-3.33,-1.07,-12.37,-5.35,
          -9.99, -2.19,-1.4,0.01,0.12,-1.53,-2.19,-0.23,
          -1.37,-4.18,-1.18,0.07,-0.76,-9.75,-1.27,-20.86,
          -0.03]


print(np.mean(weekly))
print(np.mean(monthly))

scaler = MinMaxScaler(feature_range = (-1,1))
weekly = np.array(weekly).reshape(-1,1)
monthly = np.array(monthly).reshape(-1,1)

weekly = scaler.fit_transform(weekly).reshape(1,-1)[0].tolist()
monthly = scaler.fit_transform(monthly).reshape(1,-1)[0].tolist()



for i in range(len(weekly)):
  # strong sell
  if weekly[i] <= -0.6:
```

```
    weekly[i] = 0
  # sell
  elif weekly[i] > -0.6 and weekly[i] <= -0.2:
    weekly[i] = 1
  # neutral
  elif weekly[i] > -0.2 and weekly[i] <= 0.2:
    weekly[i] = 2
  # buy
  elif weekly[i] > 0.2 and weekly[i] <= 0.6:
    weekly[i] = 3
  # strong buy
  else:
    weekly[i] = 4


for i in range(len(monthly)):
  # strong sell
  if monthly[i] <= -0.6:
    monthly[i] = 0
  # sell
  elif monthly[i] > -0.6 and monthly[i] <= -0.2:
    monthly[i] = 1
  # neutral
  elif monthly[i] > -0.2 and monthly[i] <= 0.2:
    monthly[i] = 2
  # buy
  elif monthly[i] > 0.2 and monthly[i] <= 0.6:
    monthly[i] = 3
  # strong buy
  else:
    monthly[i] = 4

stocks_yahoofinance_edited['Weekly'] = pd.Series(weekly,␣
 ↪index=stocks_yahoofinance_edited.index)
stocks_yahoofinance_edited['Monthly'] = pd.Series(monthly,␣
 ↪index=stocks_yahoofinance_edited.index)
stocks_yahoofinance_edited



# download stock from yahoo finance & metrics calculated by using LSTM
stocks_yahoofinance_edited.to_csv('f.csv')
create_download_link(filename = 'f.csv')
```

```
-1.6266666666666665
-2.4858974358974364
```

[21]: <IPython.core.display.HTML object>

[22]:
```
###########################################################################
#      C. DM1 (investing.com) & DM2 (preprocess yahoofinance) consensus   #
###########################################################################
# In this sectiton we calculate score for each DM for each stock, the total
# score of the 2 DM's for each score (along with average, standard deviation)
# and we exclude more stocks if they are not under a specific STD threshold s.
# HERE we prefer s = 0.18, so every stock with s > 0.18 is excluded
##################################################################################
#   1. DM2 results combined (our own Beta, YTD, weekly, monthly␣
↪recommendations) #
##################################################################################
# Step 1 : changstocks_investingcome index of DM1
# 1.1
tickers = [stocks_tickers.loc[stocks_tickers['Name'] == i, 'Symbol'].item() for␣
↪i in stocks_investingcom['Name']]
stocks_investingcom.index = tickers
stocks_investingcom.index.names = ['Symbol']
# 1.2
tickers_exclude = list(set(list(stocks_investingcom.index)) - ␣
↪set(list(stocks_yahoofinance_edited.index)) )
stocks_investingcom = stocks_investingcom.drop(tickers_exclude)
stocks_investingcom = stocks_investingcom.reindex(stocks_yahoofinance_edited.
↪index)

# Step 2 : define the new DM2 Dataframe
stocks_yahoofinance_edited2 = pd.DataFrame(columns = stocks_investingcom.
↪columns, index = stocks_yahoofinance_edited.index )
for item in stocks_yahoofinance_edited.index:

  # Step 2.1 : select the respective rows froms invesing_com, and preprocessed␣
↪yahoo finance
  # to update the values of beta, YTD, weekly, monthly based on ou finding from
  # the previous analysis

  stock_yahoofinance_row = stocks_yahoofinance_edited.loc[item].to_frame().
↪transpose()
  stock_investitngcom_row = stocks_investingcom.loc[item].to_frame().transpose()

  # Step 2.3 : Update the values and append the new row in the new DM2␣
↪datatfame
  # change beta
  stock_investitngcom_row['Beta'] = stock_yahoofinance_row['beta']
  # change YTD
  stock_investitngcom_row['YTD'] = stock_yahoofinance_row['YTD']
  # change weekly
```

```
  stock_investitngcom_row['Weekly'] = stock_yahoofinance_row['Weekly']
  # change monthly
  stock_investitngcom_row['Monthly'] = stock_yahoofinance_row['Monthly']


  stock_investitngcom_row = stock_investitngcom_row.
 ↪convert_objects(convert_numeric = True)
  stocks_yahoofinance_edited2.loc[item] = stock_investitngcom_row.squeeze()
  stocks_yahoofinance_edited2 = stocks_yahoofinance_edited2.
 ↪convert_objects(convert_numeric = True)

display(stocks_yahoofinance_edited2)


# download combined results fromn DM1, DM2
stocks_yahoofinance_edited2.to_csv('g.csv')
create_download_link(filename = 'g.csv')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:31: FutureWarning:
convert_objects is deprecated.  To re-infer data dtypes for object columns, use
DataFrame.infer_objects()
For all other conversions use the data-type specific converters pd.to_datetime,
pd.to_timedelta and pd.to_numeric.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:33: FutureWarning:
convert_objects is deprecated.  To re-infer data dtypes for object columns, use
DataFrame.infer_objects()
For all other conversions use the data-type specific converters pd.to_datetime,
pd.to_timedelta and pd.to_numeric.
```

```
                         Name    Market Cap  ...  Weekly  Monthly
Symbol                                       ...
MSFT    Microsoft              1.000000e+12  ...  3.0     3.0
CSCO    Cisco                  2.409200e+11  ...  3.0     4.0
MPWR    Monolithic             5.570000e+09  ...  4.0     4.0
TXN     Texas Instruments      1.046900e+11  ...  4.0     4.0
INTU    Intuit                 6.746000e+10  ...  2.0     2.0
AAPL    Apple                  9.124400e+11  ...  4.0     4.0
INTC    Intel                  2.132400e+11  ...  4.0     4.0
AMAT    Applied Materials      3.971000e+10  ...  4.0     4.0
ADI     Analog Devices         4.070000e+10  ...  3.0     3.0
CCMP    Cabot                  3.100000e+09  ...  4.0     4.0
JKHY    Jack Henry&Associates  1.059000e+10  ...  4.0     4.0
CGNX    Cognex                 7.760000e+09  ...  4.0     4.0
MCHP    Microchip              2.021000e+10  ...  4.0     4.0
MKSI    MKS Instruments        4.040000e+09  ...  4.0     4.0
NTAP    NetApp                 1.523000e+10  ...  4.0     4.0
TER     Teradyne               7.930000e+09  ...  4.0     4.0
```

```
CTXS    Citrix Systems         1.293000e+10  ...  4.0     4.0
CDW     CDW Corp               1.534000e+10  ...  3.0     4.0
NVDA    NVIDIA                 9.323000e+10  ...  3.0     3.0
SSNC    SS&Cs                  1.472000e+10  ...  4.0     4.0
XLNX    Xilinx                 2.829000e+10  ...  1.0     1.0
BRKR    Bruker                 7.410000e+09  ...  3.0     3.0
AVGO    Broadcom               1.102900e+11  ...  1.0     2.0
CY      Cypress                8.120000e+09  ...  3.0     4.0
OTEX    Open Text              1.110000e+10  ...  4.0     4.0
KLIC    Kulicke&Soffa          1.430000e+09  ...  4.0     4.0
LOGI    Logitech               6.350000e+09  ...  4.0     4.0
MANT    ManTech                2.570000e+09  ...  3.0     4.0
GRMN    Garmin                 1.532000e+10  ...  3.0     4.0
NATI    National Instruments   5.290000e+09  ...  4.0     4.0
STX     Seagate                1.259000e+10  ...  3.0     4.0
OLED    Universal Display      8.680000e+09  ...  3.0     3.0
QCOM    Qualcomm               8.680000e+10  ...  3.0     4.0
PRGS    Progress               1.840000e+09  ...  4.0     4.0
ESLT    Elbit Systems          6.770000e+09  ...  4.0     4.0
UBNT    Ubiquiti               9.220000e+09  ...  2.0     2.0
TTEC    TTEC                   2.070000e+09  ...  4.0     4.0
EQIX    Equinix                4.234000e+10  ...  0.0     0.0
AMSWA   American Software      4.432200e+08  ...  4.0     4.0

[39 rows x 12 columns]
```

[22]: <IPython.core.display.HTML object>

[23]:
```
########################
#   2. Normalization  #
########################

def normalization(dm, decision, weights):
  dm_normalized = dm.copy()
  fields = list(dm_normalized.columns.values)

  # Step 1: In order to be able to compare different kinds of criteria the␣
↪first step
  # is to make them dimensionless
  for criterion in fields[1:]:
    crit_values = list(dm_normalized[criterion])
    rms = np.sqrt(sum([i**2 for i in crit_values]))
    dm_normalized[criterion] = dm_normalized[criterion] / rms

  for i in range(1,len(fields)-1):
    dm_normalized[fields[i]] = dm_normalized[fields[i]] / weights[i]
```

```
    # Step 3: Final ranking for the stocks of the portfolio
    # Total sum per row:
    dm_normalized['DM' + str(decision)] = dm_normalized.sum(1)
    return dm_normalized

# Market Cap : 1 %
# P/E Ratio: 2.5%
# Revenue: 2.5%
# Average Vol: 1.5%
# EPS: 10%
# Beta: 25%
# YTD: 15%
# 1 Year: 6.5%
# 3 Year: 1%
# Weekly: 25%
# Monthly: 10%
weights = [ 0.01, 0.025, 0.025, 0.015, 0.1, 0.25, 0.15, 0.065, 0.01, 0.25, 0.1]

stocks_investingcom_normalized = normalization(stocks_investingcom,1, weights)
stocks_yahoofinance_normalized =
 ↪normalization(stocks_yahoofinance_edited2,2,weights)

display(stocks_investingcom_normalized.head())
display(stocks_yahoofinance_normalized.head())
```

```
                       Name  Market Cap  P/E Ratio  ...    Weekly   Monthly        DM1
Symbol                                               ...
MSFT    Microsoft            28.367598   4.451415   ...  2.119996  0.185695  88.655328
CSCO    Cisco                6.834322    3.023240   ...  2.119996  0.185695  46.551817
MPWR    Monolithic           0.158008    7.926591   ...  0.529999  0.092848  6.810312
TXN     Texas Instruments    2.969804    2.977456   ...  2.119996  0.185695  14.604107
INTU    Intuit               1.913678    6.105557   ...  2.119996  0.185695  28.913704

[5 rows x 13 columns]
```

```
                       Name  Market Cap  P/E Ratio  ...    Weekly   Monthly        DM2
Symbol                                               ...
MSFT    Microsoft            28.367598   4.451415   ...  1.383797  0.131306  88.014077
CSCO    Cisco                6.834322    3.023240   ...  1.383797  0.175075  45.993794
MPWR    Monolithic           0.158008    7.926591   ...  1.845062  0.175075  8.250392
TXN     Texas Instruments    2.969804    2.977456   ...  1.845062  0.175075  14.534014
INTU    Intuit               1.913678    6.105557   ...  0.922531  0.087538  27.805580

[5 rows x 13 columns]
```

[24]:
```
#####################################
#   3. Combine scores of DM1, DM2   #
#####################################
# Step 1 : Combine DM1, DM2 scores
final = pd.merge(stocks_investingcom_normalized,stocks_yahoofinance_normalized,␣
 ↪left_index=True, right_index=True )
final = final[['DM1','DM2']]
#display(final.head())

# Step 2.1 : Calculate total score
final['Total Score'] = final.sum(1)
#display(final.head())

# Step 2.2 : Rank stocks by total score (sorting)
final = final.sort_values('Total Score', ascending=False)
#display(final.head())


# Step 2.2 : Average of scores of DM1, DM2
final['Average'] = final[['DM1', 'DM2']].mean(1)
#display(final.head())


# Step 2.3 : Standard deviation of scores of DM1, DM2
final['St. dev'] = final[['DM1', 'DM2']].std(1)

#######################DROP STOCKS######################
TH_std = 0.8

# Step 3.1 : Droping stocks based on their bad total score
final = final[final['Total Score'] >= 0]

# Step 3.2 : Droping stocks based on the standard deviation threshold
final = final[final['St. dev'] <= TH_std]


# Step 4.1 : Calculate consensus & total % consensus
def consensus(data):
  return 100 - (100 * data['St. dev'] / TH_std)

# Step 4.2 : Droping stocks based on their bad consensus performance
TH_con = 15

final['Standard Consensus'] = final[['DM1', 'DM2', 'St. dev']].apply(consensus,␣
 ↪axis = 1)
final = final[final['Standard Consensus'] >= TH_con]
final = final.sort_values('Standard Consensus', ascending=False)
```

```
print(final.shape)
display(final)
print("Final Average Consensus for DM1, DM2 is : " + str(round(final['Standard␣
 ↪Consensus'].mean(0), 2)) + "%")

# download consensus table
final.to_csv('h.csv')
create_download_link(filename = 'h.csv')
```

```
(22, 6)

               DM1         DM2  ...   St. dev  Standard Consensus
Symbol                          ...
TXN      14.604107   14.534014  ...  0.049563  93.804643
OTEX     19.417729   19.513887  ...  0.067994  91.500772
AMAT     7.792784    7.663033   ...  0.091748  88.531498
UBNT     39.425744   39.291003  ...  0.095277  88.090401
AMSWA    13.476771   13.692423  ...  0.152489  80.938835
TTEC     25.567020   25.329434  ...  0.167999  79.000134
AVGO     20.341370   20.073766  ...  0.189225  76.346897
AAPL     101.679142  101.370049 ...  0.218562  72.679757
QCOM     33.416355   33.047022  ...  0.261158  67.355254
XLNX     46.380164   45.966046  ...  0.292826  63.396724
ESLT     29.808219   29.349819  ...  0.324138  59.482776
TER      21.316604   20.817141  ...  0.353173  55.853329
CSCO     46.551817   45.993794  ...  0.394582  50.677288
SSNC     27.625701   28.238541  ...  0.433343  45.832071
CDW      28.023398   27.391592  ...  0.446755  44.155667
MSFT     88.655328   88.014077  ...  0.453434  43.320799
MANT     20.523921   19.829817  ...  0.490806  38.649287
GRMN     27.968396   27.191304  ...  0.549487  31.314142
ADI      18.375722   17.594464  ...  0.552432  30.945940
JKHY     12.181143   12.965321  ...  0.554497  30.687822
BRKR     45.281044   44.439207  ...  0.595269  25.591428
OLED     87.148405   86.250436  ...  0.634960  20.630006

[22 rows x 6 columns]


Final Average Consensus for DM1, DM2 is : 58.13%
```

[24]:
```
<IPython.core.display.HTML object>
```

[25]:
```
###########################################################
#       D. Apply TOPSIS, ELECTRE with veto, PROMETHE    #
###########################################################
# > In this sectiton we apply all the above metthodologies to reach a single␣
 ↪(or a pool)
```

```
# of stocks
# > Depsite the use of our own data to conclude upon the DM2 values, for this␣
↪section
# we will proceed with actual data we already have from a trust worthy source
# investing.com
stock_tickers = list(final.index)

# Step 1 :  Keep the 22 stocks
tickers_exclude = list(set(list(stocks_investingcom.index)) -␣
↪set(stock_tickers))
stocks = stocks_investingcom.drop(tickers_exclude)
stocks = stocks.reindex(final.index)

# Step 2 :  Normalize again with different weights this time since our␣
↪perspective
# have changed. In the previous setction we give greater weights to our risk␣
↪metrics
# and YTD calculated by our own analysis so that we could have a fair enough DM2
# results.
# Market Cap : 2.5 %
# P/E Ratio: 10%
# Revenue: 15%
# Average Vol: 2.5%
# EPS: 15%
# Beta: 12.5%
# YTD: 10%
# 1 Year: 10%
# 3 Year: 2.5%
# Weekly: 12.5%
# Monthly: 7.5%
weights = [ 0.025, 0.1, 0.15, 0.025, 0.15, 0.125, 0.1, 0.1, 0.025, 0.125, 0.075]

stocks_normalized = normalization(stocks,1,weights)
del stocks_normalized['DM1']
del stocks_normalized['Name']
display(stocks)
```

```
                         Name    Market Cap  ... Weekly  Monthly
Symbol                                       ...
TXN     Texas Instruments      1.046900e+11  ... 4       4
OTEX    Open Text              1.110000e+10  ... 4       4
AMAT    Applied Materials      3.971000e+10  ... 4       3
UBNT    Ubiquiti               9.220000e+09  ... 2       4
AMSWA   American Software      4.432200e+08  ... 4       4
TTEC    TTEC                   2.070000e+09  ... 4       4
AVGO    Broadcom               1.102900e+11  ... 2       4
AAPL    Apple                  9.124400e+11  ... 4       4
```

```
QCOM    Qualcomm               8.680000e+10  ...  3       4
XLNX    Xilinx                 2.829000e+10  ...  2       4
ESLT    Elbit Systems          6.770000e+09  ...  4       4
TER     Teradyne               7.930000e+09  ...  4       4
CSCO    Cisco                  2.409200e+11  ...  4       4
SSNC    SS&Cs                  1.472000e+10  ...  2       4
CDW     CDW Corp               1.534000e+10  ...  4       4
MSFT    Microsoft              1.000000e+12  ...  4       4
MANT    ManTech                2.570000e+09  ...  4       4
GRMN    Garmin                 1.532000e+10  ...  4       4
ADI     Analog Devices         4.070000e+10  ...  4       4
JKHY    Jack Henry&Associates  1.059000e+10  ...  2       3
BRKR    Bruker                 7.410000e+09  ...  4       4
OLED    Universal Display      8.680000e+09  ...  4       4

[22 rows x 12 columns]
```

[26]:
```
##################
#   1. TOPSIS    #
##################
# 1. The basic principle of the TOPSIS method is that the chosen alternative
#    should have the shortest distance from the positive ideal solution (PIS)
#    and the farthest distance from the negative ideal solution (NIS). It is an
#    effective method to determine the total ranking order of decision␣
↪alternatives.
# 2. TOPSIS method are used to derive the closeness coefficient and the␣
↪outranking
#    index of each stock, respectively. Based on the closeness coefficient, the
#    outranking index, and selection threshold, we can easily obtain three type␣
↪of
#    the investment ratio in accordance with different investment preference of␣
↪final decision-maker.
#    It is a reasonable way in real decision environment

# Step 1 : Ideal and anti-ideal solutions
topsis_ideal = pd.DataFrame(index = ['Positive ideal solution','Negative ideal␣
↪solution'], columns = list(stocks_normalized.columns))
topsis_ideal.loc['Positive ideal solution'] = stocks_normalized.max()
topsis_ideal.loc['Negative ideal solution'] = stocks_normalized.min()

# Step 2 : Calculation of the Separation Measures D+, D-. This step is about␣
↪the calculation
# of the distances of each alternative from the ideal solution as:
############################################
#                 _______________       #
#                / --                   #
```

```
#               / \                2    #
#       D_i* =  | /  (v_ij - v_j*)      #
#              \| --                    #
#                                       #
#########################################
#########################################
#                ---------------        #
#               | --                    #
#          _    | \              _ 2    #
#       D_i  =  | /  (v_ij - v_j )      #
#              \| --                    #
#                                       #
#########################################
stocks_topsis = stocks_normalized.copy()

# Step 2.1 : Calculate D+, D-
def D_plus(data):
  return np.sqrt( sum((data - topsis_ideal.loc['Positive ideal solution'])**2) ␣
 ↪)

def D_minus(data):
  return np.sqrt( sum((data - topsis_ideal.loc['Negative ideal solution'])**2) ␣
 ↪)

D_plus = stocks_topsis.apply(D_plus, axis = 1)
D_minus = stocks_topsis.apply(D_minus, axis = 1)


stocks_topsis['D_plus'] = D_plus
stocks_topsis['D_minus'] = D_minus


# Step 3 : Calculation of the Relative Closeness to the Ideal Solution
# The relative closeness C_i* is always between 0 and 1 and an alternative is
# best when it is closer to 1. It is calculated for each alternative and is␣
 ↪defined as
#########################################
#                    _                  #
#                 D_i                   #
#                                       #
#       C_i* = ----------               #
#                        _              #
#              D_i* + D_i               #
#                                       #
#########################################
def C(data):
  return data['D_minus'] / (data['D_plus'] + data['D_minus'])
```

```
stocks_topsis['C_closeness'] =  stocks_topsis.apply(C, axis = 1)
display(stocks_topsis )

# Step 4 : Step 6. Ranking of the Preference Order Finally, the alternatives
# are ranked from best (higher relative closeness value) to worst.
# The best alternative and the solution to the problem is on the top of the␣
↪list.
TH_topsis = 0.25
stocks_topsis = stocks_topsis.sort_values('C_closeness', ascending=False)
stocks_topsis = stocks_topsis[stocks_topsis['C_closeness'] >= TH_topsis]

display(stocks_topsis)

# Step 5 :  Print company names we will invest in

companynames_topsis = [stocks_tickers.loc[stocks_tickers['Symbol'] == i,␣
↪'Name'].item() for i in list(stocks_topsis.index) ]
print("We will invest in the following companies :")
print(companynames_topsis)

display(stocks.loc[list(stocks_topsis.index)])
```

```
        Market Cap  P/E Ratio    Revenue  ...     D_plus    D_minus  C_closeness
Symbol                                    ...
TXN       0.754252   0.614305   2.124190  ...  43.686894   4.554020     0.094402
OTEX      0.079971   1.234400   0.392410  ...  43.054234   6.719751     0.135005
AMAT      0.286096   0.361392   2.148715  ...  45.209548   4.088907     0.082942
UBNT      0.066427   0.909879   0.155329  ...  38.990413  15.871662     0.289301
AMSWA     0.003193   2.249101   0.015232  ...  45.979810   3.375561     0.068393
TTEC      0.014914   1.275842   0.208468  ...  42.163467   8.767989     0.172153
AVGO      0.794598   1.034507   2.903559  ...  41.996721   6.239085     0.129346
AAPL      6.573785   0.518320  35.220131  ...  25.198411  36.701146     0.592915
QCOM      0.625361   1.210023   2.892659  ...  39.802336   9.423257     0.191430
XLNX      0.203819   0.999770   0.416935  ...  37.908481  18.105980     0.323238
ESLT      0.048775   0.980268   0.641753  ...  40.566877  10.996578     0.213263
TER       0.057133   0.613696   0.287495  ...  42.506566   8.056419     0.159334
CSCO      1.735738   0.623752   6.992522  ...  35.252597  12.825610     0.266766
SSNC      0.106052   3.595028   0.564089  ...  43.716260   6.080806     0.122112
CDW       0.110519   0.724612   2.260443  ...  39.937041   9.863132     0.198054
MSFT      7.204622   0.918411  16.651523  ...  26.771686  21.640885     0.447010
MANT      0.018516   0.940046   0.271144  ...  42.533080   7.789842     0.154797
GRMN      0.110375   0.664279   0.463261  ...  40.716871  10.996370     0.212641
ADI       0.293228   0.798049   0.850221  ...  43.084251   6.156573     0.125030
JKHY      0.076297   1.144205   0.215280  ...  44.408392   4.542273     0.092793
BRKR      0.053386   1.232877   0.262969  ...  38.534862  16.827787     0.303956
OLED      0.062536   3.207125   0.045669  ...  36.325830  29.972628     0.452086
```

```
[22 rows x 14 columns]


        Market Cap  P/E Ratio    Revenue  ...     D_plus    D_minus  C_closeness
Symbol                                     ...
AAPL      6.573785   0.518320  35.220131  ...  25.198411  36.701146     0.592915
OLED      0.062536   3.207125   0.045669  ...  36.325830  29.972628     0.452086
MSFT      7.204622   0.918411  16.651523  ...  26.771686  21.640885     0.447010
XLNX      0.203819   0.999770   0.416935  ...  37.908481  18.105980     0.323238
BRKR      0.053386   1.232877   0.262969  ...  38.534862  16.827787     0.303956
UBNT      0.066427   0.909879   0.155329  ...  38.990413  15.871662     0.289301
CSCO      1.735738   0.623752   6.992522  ...  35.252597  12.825610     0.266766

[7 rows x 14 columns]


We will invest in the following companies :
['Apple', 'Universal Display', 'Microsoft', 'Xilinx', 'Bruker', 'Ubiquiti',
'Cisco']

                     Name    Market Cap  P/E Ratio  ...  3 Years  Weekly  Monthly
Symbol                                              ...
AAPL                Apple  9.124400e+11      17.01  ...   108.71       4        4
OLED    Universal Display  8.680000e+09     105.25  ...   164.87       4        4
MSFT            Microsoft  1.000000e+12      30.14  ...   168.48       4        4
XLNX               Xilinx  2.829000e+10      32.81  ...   138.65       2        4
BRKR               Bruker  7.410000e+09      40.46  ...    93.23       4        4
UBNT             Ubiquiti  9.220000e+09      29.86  ...   225.04       2        4
CSCO                Cisco  2.409200e+11      20.47  ...    94.97       4        4

[7 rows x 12 columns]
```

[27]:
```python
########################################
#   2. ELECTRE I with & without veto  #
########################################
# Step 1 : Define the necessary DataFrames of agreement & disagreement
stocks_electreI = stocks.copy()
del stocks_electreI['Name']

# Step 2 : Define veto thresholds. The veto thresholds express the power␣
↪attributed to a given criteria to be against the assertion a outranks b, when
# the difference in the evaluation between g(b) and g(a) is greater than this␣
↪threshold.
veto = [10000000000000000000000000,␣
↪15,10000000000000000000000000,10000000000000000000000000,5,0.6,25,␣
↪45,80,10000000000000000000000000,10000000000000000000000000]
```

```
eledctrI_agree = pd.DataFrame(index = stocks_electreI.index, columns =␣
 ↪stocks_electreI.index)
eledctrI_disagree = eledctrI_agree.copy()
eledctrI_disagree_veto = eledctrI_agree.copy()

# Step 2 : Calculate agreement & disagreemnt table
# define delta for disagreement

c = stocks_electreI.max()
d = stocks_electreI.min()
delta = max(c - d)

for i in eledctrI_agree.columns:
  for j in eledctrI_agree.columns:
    a = stocks_electreI.loc[i]
    b = stocks_electreI.loc[j]
    # Step 2.1 : agreement cell calculation
    eledctrI_agree[j].loc[i] =  sum([ weights[i] for i in range(len((a-b))) if␣
 ↪(a-b)[i] >=0 ])

    # Step 2.2 : disagreement cell calculation
    # with veto
    dis = (b - a) - veto
    dis_result = 1 if any(i >= 0 for i in dis) else 0
    eledctrI_disagree_veto[j].loc[i] = dis_result
    # without veto
    eledctrI_disagree[j].loc[i] = max(dis) / delta

display(eledctrI_agree)
display(eledctrI_disagree)
display(eledctrI_disagree_veto)


#display(eledctrI_agree)
#display(eledctrI_disagree_veto)

# Step 3.1.1 : FIND kernel after applying ELECTRE withtout veto
# Start with c = 1.0, d = 0 as initial threshold values for
# agreement and disagreement. Relaxing those thresholds until we
# include at least 5 stocks in our kernel
c = 1.0
d = 0.0

kernel = list(eledctrI_agree.columns)
changes = {}
```

```
for k in range(1000000000):
  length = len(kernel)
  for i in eledctrI_agree.columns:
    for j in eledctrI_agree.columns:
      if i == j: continue
      if(eledctrI_agree[j].loc[i]  >= c and eledctrI_disagree[j].loc[i] <= d):
        if j in kernel:
          kernel.remove(j)
  a = kernel.copy()
  if(len(kernel) < length): changes.update({(c,d):a})
  if(len(kernel) < 6): break
  c -= 0.01
  d += 0.01

# Step 3.1.2 :  Print company names we will invest in
companynames_electreI = [stocks_tickers.loc[stocks_tickers['Symbol'] == i,␣
 ↪'Name'].item() for i in kernel ]
print("Electre I withtout veto : We will invest in the following companies :")
print(companynames_electreI)
print()
print(changes)
print()
display(stocks.loc[kernel])

# Step 3.2 : FIND kernel after applying ELECTRE withtout veto
kernel_veto = list(eledctrI_agree.columns)
changes_veto = {}
s = 1
for k in range(100000000):
  length = len(kernel_veto)
  for i in eledctrI_agree.columns:
    for j in eledctrI_agree.columns:
      if i == j: continue
      if(eledctrI_agree[j].loc[i]  >= s and eledctrI_disagree_veto[j].loc[i] ==␣
 ↪0):
        if j in kernel_veto:
          kernel_veto.remove(j)
  a = kernel_veto.copy()
  if(len(kernel_veto) < length): changes_veto.update({(s):a})
  if(len(kernel_veto) < 6): break
  s -= 0.01


# Step 3.3.2 :  Print company names we will invest in
companynames_electreI = [stocks_tickers.loc[stocks_tickers['Symbol'] == i,␣
 ↪'Name'].item() for i in kernel_veto ]
print("Electre with veto : We will invest in the following companies :")
```

```python
print(companynames_electreI)
print()
print(changes_veto)

display(stocks.loc[kernel_veto])
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:190:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)

Symbol    TXN   OTEX   AMAT   UBNT  AMSWA  ...   GRMN    ADI   JKHY   BRKR   OLED
Symbol                                     ...
TXN     1      0.7    0.575  0.55   0.8    ...  0.675  0.55   0.8    0.55   0.55
OTEX    0.5    1      0.4    0.5    0.65   ...  0.3    0.4    0.7    0.475  0.375
AMAT    0.55   0.725  1      0.45   0.625  ...  0.55   0.525  0.65   0.6    0.6
UBNT    0.525  0.575  0.55   1      0.675  ...  0.675  0.55   0.725  0.4    0.425
AMSWA   0.4    0.55   0.5    0.4    1      ...  0.4    0.4    0.4    0.3    0.2
TTEC    0.5    0.8    0.5    0.55   0.775  ...  0.4    0.5    0.5    0.3    0.35
AVGO    0.6    0.575  0.6    0.65   0.675  ...  0.525  0.525  0.675  0.425  0.425
AAPL    0.9    0.8    0.775  0.55   0.8    ...  0.7    0.575  0.9    0.575  0.55
QCOM    0.675  0.65   0.475  0.625  0.65   ...  0.5    0.6    0.825  0.55   0.55
XLNX    0.525  0.775  0.4    0.6    0.675  ...  0.575  0.425  0.75   0.55   0.425
ESLT    0.5    0.85   0.65   0.7    0.875  ...  0.7    0.65   0.725  0.5    0.5
TER     0.55   0.725  0.525  0.6    0.9    ...  0.475  0.55   0.725  0.7    0.65
CSCO    0.725  0.9    0.725  0.4    0.8    ...  0.65   0.625  0.75   0.575  0.55
SSNC    0.525  0.625  0.3    0.5    0.775  ...  0.6    0.3    0.85   0.525  0.375
CDW     0.675  0.9    0.825  0.4    0.8    ...  0.875  0.725  0.9    0.55   0.525
MSFT    0.85   0.9    0.875  0.75   0.8    ...  1      0.875  0.9    0.575  0.575
MANT    0.5    0.6    0.4    0.45   0.8    ...  0.3    0.4    0.7    0.5    0.5
GRMN    0.525  0.9    0.575  0.4    0.8    ...  1      0.3    0.9    0.55   0.55
ADI     0.65   0.8    0.6    0.525  0.8    ...  0.9    1      0.9    0.7    0.55
JKHY    0.2    0.3    0.425  0.4    0.6    ...  0.1    0.1    1      0.175  0.325
BRKR    0.65   0.725  0.525  0.675  0.9    ...  0.65   0.5    0.825  1      0.375
OLED    0.65   0.825  0.525  0.65   1      ...  0.65   0.65   0.675  0.825  1

[22 rows x 22 columns]


Symbol           TXN         OTEX  ...         BRKR         OLED
Symbol                             ...
TXN    -6.00266e-13  5.35237e-12  ...   1.5867e-11  7.01211e-11
OTEX    1.40062e-13 -6.00266e-13  ...  7.82347e-12  5.04624e-11
AMAT   -1.01045e-12  1.36561e-11  ...  2.34504e-11  7.84248e-11
UBNT   -7.20319e-13 -1.46065e-12  ...  2.30102e-12  6.04168e-11
```

```
AMSWA   3.20142e-13 -8.10359e-13  ...  1.66674e-11  7.03112e-11
TTEC   -6.00266e-14 -8.00355e-13  ... -1.00044e-14  4.84015e-11
AVGO   -2.90129e-13 -1.03046e-12  ...  2.54013e-11  6.37583e-11
AAPL   -6.1027e-13   8.50377e-12  ...  8.45375e-12  7.32725e-11
QCOM   -9.90439e-13 -1.73077e-12  ...  8.47376e-12  5.31235e-11
XLNX   -6.00266e-13 -1.34059e-12  ...  2.96131e-12  5.74655e-11
ESLT   -2.10093e-13 -9.50421e-13  ... -1.60071e-13  5.81058e-11
TER    -9.30412e-13  5.37238e-12  ...  5.32236e-12  7.01411e-11
CSCO   -5.70253e-13  5.04223e-12  ...  4.99221e-12  6.98109e-11
SSNC    2.00089e-14 -1.41063e-12  ...  4.48199e-12  5.76856e-11
CDW    -4.30191e-13  1.73077e-12  ...  3.69164e-12  6.64995e-11
MSFT   -6.1027e-13  -1.3506e-12   ...  9.00399e-13  6.01367e-11
MANT   -3.20142e-13 -1.06047e-12  ...  9.90439e-12  5.94263e-11
GRMN   -3.6016e-13   3.71165e-12  ...  5.6325e-12   6.84804e-11
ADI    -7.80346e-13 -6.80302e-13  ...  5.30235e-12  6.40884e-11
JKHY   -3.20142e-13 -1.06047e-12  ...  2.50611e-11  6.34181e-11
BRKR   -6.50288e-13 -1.39062e-12  ... -6.00266e-13  4.98121e-11
OLED   -8.90395e-13 -1.63072e-12  ... -8.40372e-13 -6.00266e-13

[22 rows x 22 columns]


Symbol  TXN  OTEX  AMAT  UBNT  AMSWA  TTEC  ...  MANT  GRMN  ADI  JKHY  BRKR  OLED
Symbol                                      ...
TXN     0    1     0     1     1      1     ...  0     0     0    1     1     1
OTEX    1    0     1     1     1      1     ...  0     0     1    0     1     1
AMAT    0    1     0     1     1      1     ...  1     0     0    1     1     1
UBNT    0    0     0     0     1      1     ...  0     0     0    0     1     1
AMSWA   1    0     1     1     0      0     ...  0     0     1    0     1     1
TTEC    0    0     1     1     1      0     ...  0     0     1    0     0     1
AVGO    0    0     1     1     1      1     ...  0     0     0    0     1     1
AAPL    0    1     0     1     1      1     ...  0     0     0    1     1     1
QCOM    0    0     0     1     1      1     ...  0     0     0    0     1     1
XLNX    0    0     0     1     1      1     ...  0     0     0    0     1     1
ESLT    0    0     1     1     1      0     ...  0     0     0    0     0     1
TER     0    1     0     1     1      1     ...  0     0     0    1     1     1
CSCO    0    1     0     1     1      1     ...  0     0     0    1     1     1
SSNC    1    0     0     1     0      1     ...  0     0     0    0     1     1
CDW     0    1     0     0     1      1     ...  0     0     0    0     1     1
MSFT    0    0     0     0     1      1     ...  0     0     0    0     1     1
MANT    0    0     1     1     1      1     ...  0     0     0    0     1     1
GRMN    0    1     1     1     1      1     ...  0     0     0    1     1     1
ADI     0    0     0     1     1      1     ...  0     0     0    0     1     1
JKHY    0    0     1     1     1      1     ...  0     0     0    0     1     1
BRKR    0    0     0     1     1      0     ...  0     0     0    0     0     1
OLED    0    0     0     0     0      0     ...  0     0     0    0     0     0

[22 rows x 22 columns]
```

```
Electre I withtout veto : We will invest in the following companies :
['Ubiquiti', 'Apple', 'Elbit Systems', 'Microsoft', 'Universal Display']

{(1.0, 0.0): ['TXN', 'OTEX', 'AMAT', 'UBNT', 'TTEC', 'AVGO', 'AAPL', 'QCOM',
'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'MANT', 'ADI', 'JKHY', 'BRKR', 'OLED'],
(0.8999999999999999, 0.09999999999999999): ['AMAT', 'UBNT', 'AAPL', 'QCOM',
'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'ADI', 'BRKR', 'OLED'],
(0.8699999999999999, 0.12999999999999998): ['UBNT', 'AAPL', 'QCOM', 'XLNX',
'ESLT', 'TER', 'SSNC', 'MSFT', 'BRKR', 'OLED'], (0.8199999999999998,
0.18000000000000002): ['UBNT', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC',
'MSFT', 'OLED'], (0.7999999999999998, 0.20000000000000004): ['UBNT', 'AAPL',
'QCOM', 'ESLT', 'TER', 'SSNC', 'MSFT', 'OLED'], (0.7699999999999998,
0.23000000000000007): ['UBNT', 'AAPL', 'ESLT', 'MSFT', 'OLED']}


                     Name    Market Cap  P/E Ratio  ...  3 Years  Weekly  Monthly
Symbol                                               ...
UBNT    Ubiquiti          9.220000e+09  29.86      ...  225.04   2       4
AAPL    Apple             9.124400e+11  17.01      ...  108.71   4       4
ESLT    Elbit Systems     6.770000e+09  32.17      ...  70.87    4       4
MSFT    Microsoft         1.000000e+12  30.14      ...  168.48   4       4
OLED    Universal Display 8.680000e+09  105.25     ...  164.87   4       4

[5 rows x 12 columns]


Electre with veto : We will invest in the following companies :
['Apple', 'Elbit Systems', 'SS&Cs', 'Microsoft', 'Universal Display']

{1: ['TXN', 'OTEX', 'AMAT', 'UBNT', 'TTEC', 'AVGO', 'AAPL', 'QCOM', 'XLNX',
'ESLT', 'TER', 'SSNC', 'MSFT', 'MANT', 'ADI', 'JKHY', 'BRKR', 'OLED'],
0.8999999999999999: ['AMAT', 'UBNT', 'AVGO', 'AAPL', 'QCOM', 'XLNX', 'ESLT',
'TER', 'SSNC', 'MSFT', 'ADI', 'BRKR', 'OLED'], 0.8699999999999999: ['UBNT',
'AVGO', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER', 'SSNC', 'MSFT', 'BRKR', 'OLED'],
0.8199999999999998: ['UBNT', 'AVGO', 'AAPL', 'QCOM', 'XLNX', 'ESLT', 'TER',
'SSNC', 'MSFT', 'OLED'], 0.7999999999999998: ['UBNT', 'AVGO', 'AAPL', 'QCOM',
'ESLT', 'TER', 'SSNC', 'MSFT', 'OLED'], 0.7699999999999998: ['UBNT', 'AVGO',
'AAPL', 'ESLT', 'SSNC', 'MSFT', 'OLED'], 0.7499999999999998: ['AAPL', 'ESLT',
'SSNC', 'MSFT', 'OLED']}

                     Name    Market Cap  P/E Ratio  ...  3 Years  Weekly  Monthly
Symbol                                               ...
AAPL    Apple             9.124400e+11  17.01      ...  108.71   4       4
ESLT    Elbit Systems     6.770000e+09  32.17      ...  70.87    4       4
SSNC    SS&Cs             1.472000e+10  117.98     ...  98.06    2       4
MSFT    Microsoft         1.000000e+12  30.14      ...  168.48   4       4
OLED    Universal Display 8.680000e+09  105.25     ...  164.87   4       4
```

```
[5 rows x 12 columns]
```

[38]:
```
######################
#   3. PROMETTHEE    #
######################
stocks_promethee = stocks.copy()
promethee_flows = stocks.copy()
del stocks_promethee['Name']
del promethee_flows['Name']
# threshold of absolute preference, threshold of indifference and type for each␣
↪criterion
p_q_type = {'Market Cap':(10**9, 10**11, 'typeV'), 'P/E Ratio': (15,␣
↪35,'typeV'), 'Revenue':(10**8, 10**10, 'typeV'), 'Average Vol. (3m)':(10**7,␣
↪10**8, 'typeV'),
            'EPS':(1,6, 'typeV'), 'Beta':(0,0.6, 'typeV'), 'YTD':(5, 25,␣
↪'typeV'), '1 Year':(5, 45, 'typeV'), '3 Years':(10, 75,'typeV'), 'Weekly':
↪(0,1,'typeI'), 'Monthly':(0,1,'typeI')}

def typeV(d, q, p):
  if(d <= q): return 0
  elif (d > p): return 1
  else: return (d - q)/ (p - q)

def typeI(d):
  return 1 if d > 0 else 0


for crit in stocks_promethee.columns:
  print(crit)
  promethee_preferences_crit = pd.DataFrame(index = stocks_promethee.index,␣
↪columns = stocks_promethee.index)
  for i in promethee_preferences_crit.columns:
    for j in promethee_preferences_crit.columns:
      # Step 1 : Calculate the differences between the evaluations of the␣
↪stocks on the specific criterion
      diff = stocks_promethee[crit].loc[j] - stocks_promethee[crit].loc[i]
      # Step 2 : Calculate pairwise comparison
      # Use as a criterion :  linear (type V) preference functions are best␣
↪suited for quantitative criteria (e.g. prices, costs, power, ...)

      q,p,type_i = p_q_type[crit]

      if(type_i == 'typeV'):
        promethee_preferences_crit[j].loc[i] = typeV(diff, q, p)
      else:
```

```
        promethee_preferences_crit[j].loc[i] = typeI(diff)


    # Step 3 : Positive, negative, and net flows for the investment criterion
    length = len(promethee_preferences_crit) - 1
    promethee_preferences_crit['Positive Flow'] = promethee_preferences_crit.
 ↪sum(1) / length
    promethee_preferences_crit['Negative Flow'] = promethee_preferences_crit.
 ↪sum(0) / length
    promethee_preferences_crit['Net Flow'] = promethee_preferences_crit['Negative␣
 ↪Flow'] - promethee_preferences_crit['Positive Flow']

    # Step 4 : Append flows for this criterion in the final dataframe
    promethee_flows[crit] =  promethee_preferences_crit['Net Flow']

display(promethee_flows)

# Step 5 : Calculate weighted global net flow
for i in promethee_flows.index:
    promethee_flows.loc[i] = promethee_flows.loc[i] * weights

promethee_flows['Net Flow'] = promethee_flows.sum(1)
promethee_flows = promethee_flows.sort_values('Net Flow', ascending = False)
display(promethee_flows)


# Step 6 :  Print company names we will invest in
kernel_promethee = list(promethee_flows.head(7).index)
companynames_promethee = [stocks_tickers.loc[stocks_tickers['Symbol'] == i,␣
 ↪'Name'].item() for i in kernel_promethee ]
print("We will invest in the following companies :")
print(companynames_promethee)

display(stocks.loc[kernel_promethee])
```

```
Market Cap

/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:190:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)

P/E Ratio
Revenue
```

Average Vol. (3m)
EPS
Beta
YTD
1 Year
3 Years
Weekly
Monthly

	Market Cap	P/E Ratio	Revenue	...	3 Years	Weekly	Monthly
Symbol				...			
TXN	0.550495	-0.200714	0.462290	...	-0.214542	0.285714	0.095238
OTEX	-0.292570	-0.035595	-0.357561	...	-0.647297	0.285714	0.095238
AMAT	-0.036154	-0.367667	0.466522	...	-0.213018	0.285714	-0.952381
UBNT	-0.307327	-0.135714	-0.468961	...	0.971766	-0.809524	0.095238
AMSWA	-0.374761	0.800595	-0.536277	...	-0.584725	0.285714	0.095238
TTEC	-0.362661	-0.008071	-0.443901	...	-0.358564	0.285714	0.095238
AVGO	0.587128	-0.121357	0.599230	...	-0.253780	-0.809524	0.095238
AAPL	0.910746	-0.245119	1.000000	...	0.115597	0.285714	0.095238
QCOM	0.392283	-0.050833	0.598076	...	-0.682828	-0.523810	0.095238
XLNX	-0.139554	-0.126786	-0.346402	...	0.492989	-0.809524	0.095238
ESLT	-0.325086	-0.129833	-0.238177	...	-0.286982	0.285714	0.095238
TER	-0.316683	-0.200952	-0.408451	...	0.446828	0.285714	0.095238
CSCO	0.809524	-0.197024	0.809524	...	-0.054190	0.285714	0.095238
SSNC	-0.261060	0.952381	-0.275599	...	-0.017971	-0.809524	0.095238
CDW	-0.256288	-0.160548	0.487542	...	0.657751	0.285714	0.095238
MSFT	0.994016	-0.135048	0.904762	...	0.746462	0.285714	0.095238
MANT	-0.359054	-0.133357	-0.415377	...	-0.227590	0.285714	0.095238
GRMN	-0.256442	-0.181190	-0.324468	...	-0.109348	0.285714	0.095238
ADI	-0.028058	-0.144476	-0.130721	...	-0.072755	0.285714	0.095238
JKHY	-0.296740	-0.086048	-0.441015	...	-0.356505	-0.809524	-0.952381
BRKR	-0.320391	-0.036548	-0.418841	...	-0.074403	0.285714	0.095238
OLED	-0.311363	0.943905	-0.522196	...	0.723106	0.285714	0.095238

[22 rows x 11 columns]

	Market Cap	P/E Ratio	Revenue	...	Weekly	Monthly	Net Flow
Symbol				...			
AAPL	0.022769	-0.024512	0.150000	...	0.035714	0.007143	0.316607
OLED	-0.007784	0.094390	-0.078329	...	0.035714	0.007143	0.304545
MSFT	0.024850	-0.013505	0.135714	...	0.035714	0.007143	0.262331
CSCO	0.020238	-0.019702	0.121429	...	0.035714	0.007143	0.162793
CDW	-0.006407	-0.016055	0.073131	...	0.035714	0.007143	0.100067
BRKR	-0.008010	-0.003655	-0.062826	...	0.035714	0.007143	0.092525
TXN	0.013762	-0.020071	0.069343	...	0.035714	0.007143	0.066602
QCOM	0.009807	-0.005083	0.089711	...	-0.065476	0.007143	0.053186
TER	-0.007917	-0.020095	-0.061268	...	0.035714	0.007143	0.052614

```
ADI    -0.000701   -0.014448  -0.019608  ...  0.035714  0.007143  0.031153
AMAT   -0.000904   -0.036767   0.069978  ...  0.035714 -0.071429  0.006242
ESLT   -0.008127   -0.012983  -0.035727  ...  0.035714  0.007143 -0.025790
UBNT   -0.007683   -0.013571  -0.070344  ... -0.101190  0.007143 -0.052676
AVGO    0.014678   -0.012136   0.089885  ... -0.101190  0.007143 -0.057603
GRMN   -0.006411   -0.018119  -0.048670  ...  0.035714  0.007143 -0.058284
XLNX   -0.003489   -0.012679  -0.051960  ... -0.101190  0.007143 -0.070338
TTEC   -0.009067   -0.000807  -0.066585  ...  0.035714  0.007143 -0.100705
SSNC   -0.006526    0.095238  -0.041340  ... -0.101190  0.007143 -0.116393
MANT   -0.008976   -0.013336  -0.062307  ...  0.035714  0.007143 -0.149011
AMSWA  -0.009369    0.080060  -0.080441  ...  0.035714  0.007143 -0.177132
OTEX   -0.007314   -0.003560  -0.053634  ...  0.035714  0.007143 -0.227305
JKHY   -0.007419   -0.008605  -0.066152  ... -0.101190 -0.071429 -0.413427

[22 rows x 12 columns]


We will invest in the following companies :
['Apple', 'Universal Display', 'Microsoft', 'Cisco', 'CDW Corp', 'Bruker',
'Texas Instruments']

                     Name    Market Cap  P/E Ratio  ...  3 Years  Weekly  Monthly
Symbol                                              ...
AAPL    Apple              9.124400e+11  17.01      ...  108.71   4       4
OLED    Universal Display  8.680000e+09  105.25     ...  164.87   4       4
MSFT    Microsoft          1.000000e+12  30.14      ...  168.48   4       4
CSCO    Cisco              2.409200e+11  20.47      ...  94.97    4       4
CDW     CDW Corp           1.534000e+10  23.78      ...  155.31   4       4
BRKR    Bruker             7.410000e+09  40.46      ...  93.23    4       4
TXN     Texas Instruments  1.046900e+11  20.16      ...  78.89    4       4

[7 rows x 12 columns]
```

[0]:

```
```