Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
2IMD15 Data engineering
Postbus 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Author

A. Einarsson	(1493604)
S.G. Jónsson	(1499203)
G.O. Palsson	(1479334)
B. Coremans	(0962143)

Responsible Lecturer
Odysseas Papapetrou

Date
June 9, 2020

Milestone 2

2IMD15 Group Project

A. Einarsson (1493604)
a.einarsson@student.tue.nl

G.O. Palsson (1479334)
g.o.palsson@student.tue.nl

S. G. Jónsson (1499203)
s.g.jonsson@student.tue.nl

B. Coremans (0962143)
b.coremans@student.tue.nl

Where innovation starts

Dataset

The stock sample dataset from 2018 was chosen because of its size and structure. This dataset contains information on more than 2000 stocks from various markets in the form of `.txt` files. It features the name of the stock, date and time of update, the first-, highest-, lowest-, closing price, sum of volume of all transactions, all within this update frequency.

The frequency of updates is inconsistent within and between stocks, some update every minute, some once a day and some somewhere in between. First the `.txt` files where merged, a `.csv` header added, and a column containing the name of each stock was added. This yielded one big `.txt` file, roughly 9 GB, containing all the stock data sorted by the name of the stock. Having one big data file made it easier to upload online and work on collaboratively. The file was split up into dataframes, one for each stock. The dataframes had varying sizes because of the inconsistency of update frequency between stocks. A decision was made to use one measurement a day for all the stocks. This means we throw away every line except for the last update for each day. This was done because a good proportion of the stocks only had fewer than 5 measures per day and instead of interpolating and making up thousands of minutes for them we simply treat the data as a daily update. The minute wise updates are not so drastic that this should affect the overall trend in the pairwise correlation. The missing days were interpolated (linearly) for all stocks and cut off some days at the beginning and end of the year. This was done because not all stocks started updating on January 2nd all through December 31st. This resulted in a dataset that started on the 4th of January and ended on the 27th of December. Those roughly 20 stocks that did not meet this requirement were thrown out. Stocks from the currency exchange market were also thrown out because they contained basically the same stocks multiple times. In the end around 1700 stocks were left and usable for comparisons.

Correlation and Aggregation Measures

In this assignment there are two types of correlation measures applied to the data, Pearson correlation and Total Correlation.

Total Correlation

Total Correlation is one of several generalizations of mutual information and is also known as the multivariate constraint or multi-information. It quantifies the redundancy or dependency among a set of n random variables.

For a given set of n random variables $\{X_1, X_2, ..., X_n\}$, the total correlation $C(X_1, X_2, .., X_n)$ is defined as the Kullback-Leibler divergence from the joint distribution $p(X_1, ..., X_n)$ to the independent distribution of $p(X_1)p(X_2)\cdots p(X_n)$,

$$C(X_1, X_2, .., X_n) \equiv D_{KL}[p_(X_1, .., X_n)||p(X_1)p(x_2)\cdots p(X_n)] \tag{1}$$

This divergence reduces to the simpler difference of entropies

$$C(X_1, X_2, ..., X_n) = \left[\sum_{i=1}^{n} H(X_i)\right] - H(X_1, X_2, ..., X_n) \tag{2}$$

where $H(X_i)$ is the information entropy of variable X_i, and $H(X_1, X_2, ..., X_n)$ is the joint entropy of the variable set $\{X_1, X_2, ..., X_n\}$. In terms of the discrete probability distributions on variables $\{X_1, X_2, .., X_n\}$, the total correlation is given by

$$C(X_1, X_2, .., X_n) = \sum_{x_1 \in \mathcal{X}} \sum_{x_2 \in \mathcal{X}_2} \cdots \sum_{x_n \in \mathcal{X}_n} p(x_1, x_2, .., x_n) \log \frac{p(x_1, x_2, ..., x_n)}{p(x_1)p(x_2)\cdots p(x_n)} \tag{3}$$

Notice that the Kullback-Leibler divergence is a measure of how on probability distribution is different from a second, reference probability distribution.

For discrete probability distributions P and Q defined on the same probability space, $\mathcal{X}$, the Kullback-Leibler divergence from Q to P is defined to be

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{4}$$

Pearson Correlation

The Pearson correlation is a statistic that measures linear correlation between two variables X and Y. It has a value between $+1$ and -1, where 1 is total positive linear correlation, 0 is no linear correlation, and -1 is total negative linear correlation.

Pearson's correlation coefficient when applied to sample is often represented by r_{xy}. Given paired data, in our case one stock and the average of two or more stocks:

$$\{(x_1, y_1), ..., (x_n, y_n)\}$$

where n is the number of stocks, r_{xy} is defined as:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{5}$$

In Milestone 1, we considered Mutual Information. Total Correlation reduces to that measure when $p = 2$. Total Correlation is a "distance" between two or more probability distributions whereas Pearson Correlation is a linear distance between two random variables.

Aggregation Measures

In this assignment it is necessary to define an aggregation function to use the Pearson Correlation. The purpose of the aggregation function is to reduce the number of input time series. For example, if the goal is to find high multiple correlations of size $p \geq 3$ then it is required to do a linear combination so Pearson's correlation can be applied because that measurement only uses two inputs.

The group decided to only use an aggregation function that measures the average between two or more time series. The function calculates the average for each day of the stock inputs. Max and min aggregation functions were also tested, where the max or min of two or more times series was collected. The resulting correlations had high variance and produced both high and low correlations. Average was chosen because it gave higher correlations more frequently than the other aggregation functions when using Pearson correlation.

The identity function was used as aggregation function for total correlation, where all the stocks in question were compared as they were because there was no limit of inputs.

System Architecture

Following are pseudocodes for functions used in the code. Below them is a detailed reflection on the architecture.

Here is the pseudocode for the two correlation measures used in this assignment

```
# Input: List of three or more stocks
# Output: Total Correlation
def total_correlation(X)
    independentDistribution = []
    for stocks in X:
        independentDistribution.append( ComputeIndependentDistribution(stocks))
    s = 1
    for stocks in X:
        s *= ComputeJointDistribution(stocks)
        * log(ComputeJointDistribution(stocks) /
        innerProduct(independentDistribution)
return s
```

```
#Input: List of two stocks
#Output: Pearson Correlation
def pearson_correlation(X, Y):
    mean_x = mean(X)
    mean_y = mean(Y)

    for x in X:
            diff_x.append(x-mean_x)

    for y in Y:
            diff_y.append(y-mean_y)

    for dx, dy in diff_x, diff_y:
            sum_diff_xy += dx * dy
            sum_sqr_diff_x += dx * dx
            sum_sqr_diff_y += dy * dy

return sum_diff_xy / sum_sqr_diff_x * sum_sqr_diff_y
```

To calculate correlation by reducing a list of stocks by key, the following function was used.

```
#Input: List of <key,value> pairs
#Output: List of correlations calculated on <key,value> pairs
def reduce_mapping_pearson(x):
    #let sprark worker calculate correlation with reduce by key
    return reduce_by_key(corr_func, x)
```

When calculating the Total Correlation a similar function was used, but instead of reduce, map was used because there are no keys to reduce. All stocks are sent in at the same time.

```
#Input: List of subsets
#Output: List of correlations calculated on the subsets using Total correlation
def reduce_mapping_total(x):
    #let sprark worker calculate correlation with map
    return map(corr_func, x)
```

To calculate the average of multiple vectors the following function was used. Trivially similar functions for max and min aggregation methods were used.

```
#Input: List of tuple (name, vector). For size of list l: 2 < l < p-1
#Output: Tuple including the combined names of the stocks and their average vector
def calculate_average(X):
    n = X.length #Number of tuples
    sum = array[0 for i --> X[0].length] #Array of zeros with same length as arrays in X
    names = []
    for i --> n:
        sum = sum + x[i].value
        names.append(x[i].name)
    average = sum / n
    combined_name = concat(names)
return (combined_name, average)
```

For all the datapoints the following function was used to compute all possible combinations of indexes of size $p - 1$ for Pearson correlation. A similar function was used to compute indexes of size p for total correlation. This was inspired by [1], considering the stocks as the number of bits in a vector. The basic idea is to set p bits out of this vector to 1, and letting all others to 0.

```
#Input: List A to be split into subsets. K, maximum size of sub arrays to be generated.
#Output: List of lists of all subsets containing unique unordered combination from A of size 1
#to K.
def get_subsets(A,K):
    collection_subsets = [[] for i --> K]
    N = len(A)

    # looking at subsets less or equal K
    mask = 0
    while mask < (1<<N):
        subset = []
        for n --> N:
            if ((mask>>n)&1) == 1:
                subset.append(A[n])
        if len(subset) > 0:
            #append subset to collection at the same place as others of
            #the same size
            collection_subsets[len(subset)-1].append(subset)

        # when K is zero
        if K == 0:
            break

        # next mask
        if bin(mask).count("1") < K:
            mask += 1
        else:
            mask = (mask|(mask-1))+1

    return collection_subsets
```

The Pearson Correlations were computed by applying the following pseudo code. The Total correlations were computed in a similar way with differences described below.

```
#Input: Stock data, p>=3, aggregation function, correlation function, partion size
#Output: Correlation for the 10 highest stocks

def compute_correlation(data, agg_func, corr_func, p, partion):
    subsets = get_subsets(data, p-1)

    pair_averages=[subsets[0]] #initialized containing singles

    for subset in subsets[1:]:
        subset = createSparkDataframe(subset)
        # Apply aggregation function to subset using spark
        temp = parallelize(subset).flatMap(agg_func).collect()
        pair_averages.append(temp)

    #Create unique key for every possible pair
    all_combinations = [[]...[]] 1 --> partition #list of "partition" number of empty lists

    t=0
    for i-->p/2: #Half the size of the total nr. of subsets
        for s-->p:
            for a--> pair_averages[i].length:
                for b--> pair_averages[s].length:
                    index = t % partition #Used to partition equally
                    #Put opening price array into global dictionary
                    global_dict[t] = [pair_averages[i][a].value, pair_averages[s][b].value]
                    # Append tuple with index and name of stock(s)
                    all_combinations[index].append((t,pair_averages[i][a].name))
                    all_combinations[index].append((t,pair_averages[s][b].name))
                    t++

    #Compute correlations
    res = parallelize(all_combinations) #Parallelize array
          .flatMap(reduce_mapping(x)) #Compute correlation, distributed using spark
          .filter(lambda line: abs(line[0][0]) >= 0.9) #Filter correlations >0.9 or <-0.9
          .sortBy(lambda line: -abs(line[0][0])) # Sort by highest correlation
          .take(10) #Return 10 stock correlations
    return res
```

First the number of stocks for comparison is selected and the stocks are selected at random from the dataset. This is done because the stocks are sorted by stock markets and running comparisons on the full 1700 stocks is too computationally heavy. Getting a subset of stocks from various stock markets is desired so a random approach was chosen. The stocks are collected on the form `<name, [time series]>`. After initializing `Spark` and defining the aggregation and correlation functions, a function named `compute_correlation()` is called, corresponding to `correlationFunction` from the milestone description constraints. It takes as input the stocks to be compared, aggregation function, correlation function, p (size of multiple correlation) and partition (how much work each Spark worker is given).

To make the desired combinations of stocks the function `get_subsets()` was used. The function is described above and is based on bitwise subset generation. The type of subsets to generate depends on input variable p and the correlation function. When the correlation function is chosen as total correlation the subsets to generate are all unique unordered combinations of the stocks of size p. For this we use a function similar to `get_subsets()` pseudocode above, with a few adjustments. When the Pearson correlation function is chosen as the correlation function all combinations of stocks of sizes 1 to $p-1$ were generated. This means if $p = 4$ and the stocks to consider are $[a, b, c, d, e]$ the subsets generated are:

```
#list containing all sizes of sublists
[
#list containing all subsets of size 1
[['a'], ['b'], ['c'], ['d'], ['e']],
#list containing all subsets of size 2
[['a', 'b'], ['a', 'c'], ['b', 'c'], ['a', 'd'],
['b', 'd'], ['c', 'd'], ['a', 'e'], ['b', 'e'],
['c', 'e'], ['d', 'e']],
#list containing all subsets of size 3 = p-1
[['a', 'b', 'c'],['a', 'b', 'd'], ['a', 'c', 'd'],
['b', 'c', 'd'], ['a', 'b', 'e'], ['a', 'c', 'e'],
['b', 'c', 'e'], ['a', 'd', 'e'], ['b', 'd', 'e'],
['c', 'd', 'e']]
]
```

After the subsets have been generated for Pearson the task is to match them up. The goal is to match up two subsets so their combined size is p. For $p = 4$ this means subsets of size 1 is paired with subsets of size 3 and subsets of size 2 is paired with other subsets of size 2. To avoid trivial comparisons subsets are not matched with other subsets if they share any stocks. This means for example `['a', 'b']` and `['a', 'c']` are not compared. Before the subsets are matched together they are aggregated using one of the aggregation functions. The work is distributed to spark workers using `flatMap` where the aggregation function is applied on each subset containing more than 1 stock. Using the average aggregation on subset of length 3 looks something like this `['a', 'b', 'c'] -> [(a+b+c)/3]`. Thus the result is a single "stock" containing information from all 3. The aggregation can be seen in the pseudocode above describing `compute_correlation()` on lines 9-13.

All desired combinations are then given a unique key for reduction. This looks something like `<key1, A>,<key2, B>` if the goal was to compare a and b. These key value pairs were then put into a global dictionary. This is done so we do not have to send all the vectors to the Spark workers, which caused Java heap memory problems. The keys and names of the stocks were placed in another variable called `all_combination`. The variable `all_combination` is split up into a number of partitions, to be thought of as a list of lists. Each partition contains a number of `<key1, name-A>, <key1, name-B>` pairs. This can be seen in the pseudocode above describing `compute_correlation()` on lines 18-29. This is done so instead of reducing over the keys, which initializes a new spark worker for each comparison, we can map over `all_combination`. When we use map we initialize a one spark worker for each partition in `all_combination`. The worker is then responsible for more comparisons but we do not have to pay as high of a initialization price. The map function is called using a function called something like `reduce_mapping_pearson()` depending on the correlation function to be used. This function is executed on a Spark worker and takes in keys and names of the stocks in the partition to be correlated. The values of the time series is collected from the global dictionary `global_dict`. The correlation is then calculated as seen in the pseudocode above describing correlation measures. Finally all correlations values too small are filtered out, under some value τ, and the rest are sorted. The 10 highest correlations are then returned along with the names of the stocks in question.

The process for Total Correlation is similar but with a few key differences. First, as stated above, all unique unordered combinations of the stocks of size p are generated. Using the identity function there is no need to worry about aggregation. The subsets are partitions into `all_combination` as they are above and saved to the global dictionary `global_dict`. The mapping is done on `all_combination`, using a function called `reduce_mapping_total()` as seen in the pseudocode above. The Spark worker executes this function and takes as input all the subsets to be compared in the partition of *all_combination*. The worker maps of over all the subset applying the total correlation function as seen in the pseudocode above. This yields the correlations, which are subsequently filtered out if they are under some value τ. The rest are sorted and top 10 are returned.

Distribution of Computation

As described in the previous chapter a bitwise subset generation was used to make all the subsets of stocks. This method generates unique unordered combination of the stocks of size p. This means if for example subset `[a, b, c]` is the same as `[b, c, a]` and only appears once. This is what we want, because the the order does not effect the result using the aggregation functions chosen. This eliminates some redundant comparisons. The generation of subsets is done in $O(2^n)$ time.

The aggregation calculations were distributed to Spark workers. Each array containing every subset of a specific size was parallelized and flatmap applied using the aggregation function. Thus each Spark worker got one subset to aggregate at a time and returned the combined time series. This is not a bottleneck and takes relatively short amount of time considering the amount of subset to aggregate. In layman's term for every test done this never took more than a minute.

When generating the keys and splitting the subsets into partitions no Spark feature was used but it still takes a trivial amount of time. Two subsets of combined length p were given the same key. To refresh, the subsets contain stocks. This means if $p = 5$, all combinations of subsets of length 1 and 5 were given the same keys, similarly subsets of length 2 and 3 were given the same keys. To avoid redundant computations the mirror subsets were not given keys, meaning subset of length 3 and 2 are the same as 2 and 3. To avoid even more redundant computations if the subsets to be compared shared the same stock they were not considered for correlation calculation because it could yield trivial correlation.

The final step is to calculate the correlations finally. The list containing the partitions was parallelized. Given number N of `<key1, name1>, <key1, name2>` different pairs to compare, they are split into M number of partitions. This means that each partition contains N/M number of pairs. During experimentation it was noted that at around $M = 1000 - 3000$ it reached its optimum, although after $M = 100$ the difference was small. It did not seem to matter much for M what the size of N was. The spark map function initializes M many workers instead of N if reduce by key would have been used on the pairs without partitioning. Each worker is responsible to calculate N/M comparisons and does so using either reduce for Pearson or map for Total correlation, as described above. These calculations on the workers are not done with spark but in native python. This part is the bottleneck of the process, not surprisingly, because here most of the calculations are carried out. That being said splitting the pairs into partitions does save time. According to experiments, the time saved is at least by a factor of 10 but up to 100 in some cases.

Theoretical Complexity and Experimental Performance

The following computers and servers were used for the computational part

- MacBook Pro from 2015. Processor is 2,2 GHz Intel Core i7, and RAM 16 GB 1600 MHz DDR3.
- MacBook Pro from 2019. Processor is 2,2 GHz Intel 8-Core i9, and RAM 32 GB 2400 MHz DDR4.
- Server specs: Amazon web service EMR cluster that contains three instances(1 master and 2 slaves). Each instance contains 4 virtual cores and 16GB memory.

The calculation of the correlation measures vary vastly in speed. Generating the subsets for both is relatively fast but the calculations of Total correlation is much slower then Pearson. Although there are fewer comparisons and the work is distributed in the same way, Total correlation is always slower to make those calculations. This is probably due to the fact how they are calculated on the workers but not how they are distributed. Therefore when comparing the two methods on the same set of stocks, Total correlation is the always bottleneck.

As stated in a previous chapter the complexity of subset generation is $O(2^n)$. For creating key, value pairs it is $O(n^2)$ where n is the number of subsets, because they are generated with two for loops. The final step, using the map function on the key, value pairs is done on N/M partitions. In the end all the comparisons have to be calculated, they are of magnitude $O(n^n)$ and thus take the longest time to finish.

The longest run locally on the Pearson correlation was on a subset of 600 stocks and took 27 minutes and 43 seconds. For this computation $p = 3$ was used and the total number of unique indexes and number of comparisons was 107.820.000. This resulted in 215.640.000 key-value pairs to be reduced. This computation was performed on the Macbook Pro and without filtering and sorting the data after computing the correlations. The filtering and sorting methods required more memory usage and therefor the computation of the 10 highest correlations had to be performed on a smaller subset.

The subset generation has different computational complexity for the two correlation measures. This is because Total correlation only generates subsets of size p but Pearson generates all subsets of size less than p. The number of subsets generated then reflects the number of comparisons made. In the figure below the two correlation measures are compared, by taking different sized subsets with a fixed $p = 3$.

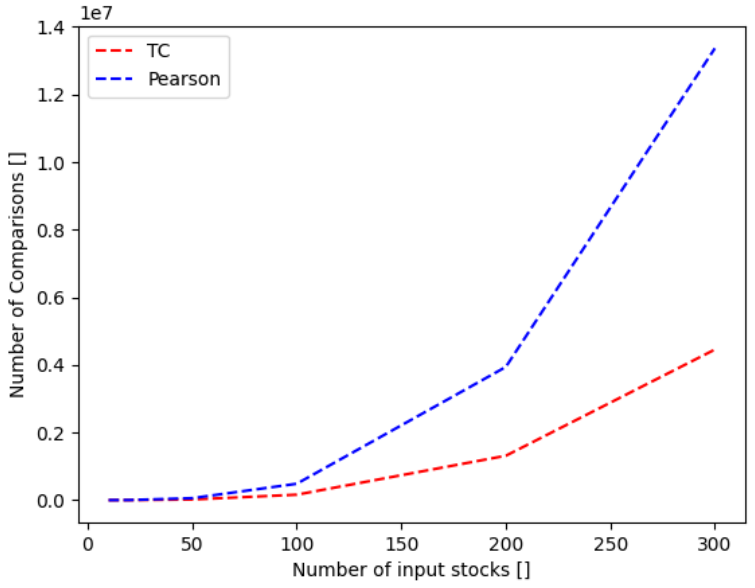


Figure 1: Comparison of calculations for TC and Pearson

In Figure 1 we see that the number of calculations for Pearson is higher than Total Correlation. And the gap between the two measures grows rapidly after taking more than 100 stocks from the data.

Table of Performance				
Pearson correlation				
p=3				
Number of stocks:	*20*	*50*	*100*	*200*
Macbook 2019	2.41	40.646	553.561	1728.146
Server	3.554	20.169	272.608	>1800
p=4				
Macbook 2019	54.218	1238.476	>1800	>1800
Server	60.101	1569	>1800	>1800
Total correlation				
p=3				
Number of stocks	*5*	*10*	*20*	*50*
Macbook 2019	10.412	26.004	169.699	>1800
Server	14.876	35.844	224.332	>1800
p=4				
Macbook 2019	6.948	37.584	948.947	>1800
Server	12.345	128.54	>1800	>1800

Table 1: Table describing the time in seconds needed to make comparisons for different machines.

Running the computations was faster when running on the Macbook Pro since the Macbook Pro allowed for more storage space. Garbage collection between each stage took a long time on the server. The following table shows the number of comparisons required for different correlation measure and p values.

Table of Comparisons				
Pearson correlation				
p=3				
Number of stocks:	*20*	*50*	*100*	*200*
Comparisons	3,420	58,800	485,100	3,940,200
p=4				
Comparisons	48,450	2,303,000	39,212,250	>70,000,000
Total correlation				
p=3				
Number of stocks	*5*	*10*	*20*	*50*
Comparisons	10	120	1,140	19,600
p=4				
Comparisons	5	210	4,845	230,300

Table 2: Table describing the number of comparisons needed for each computation

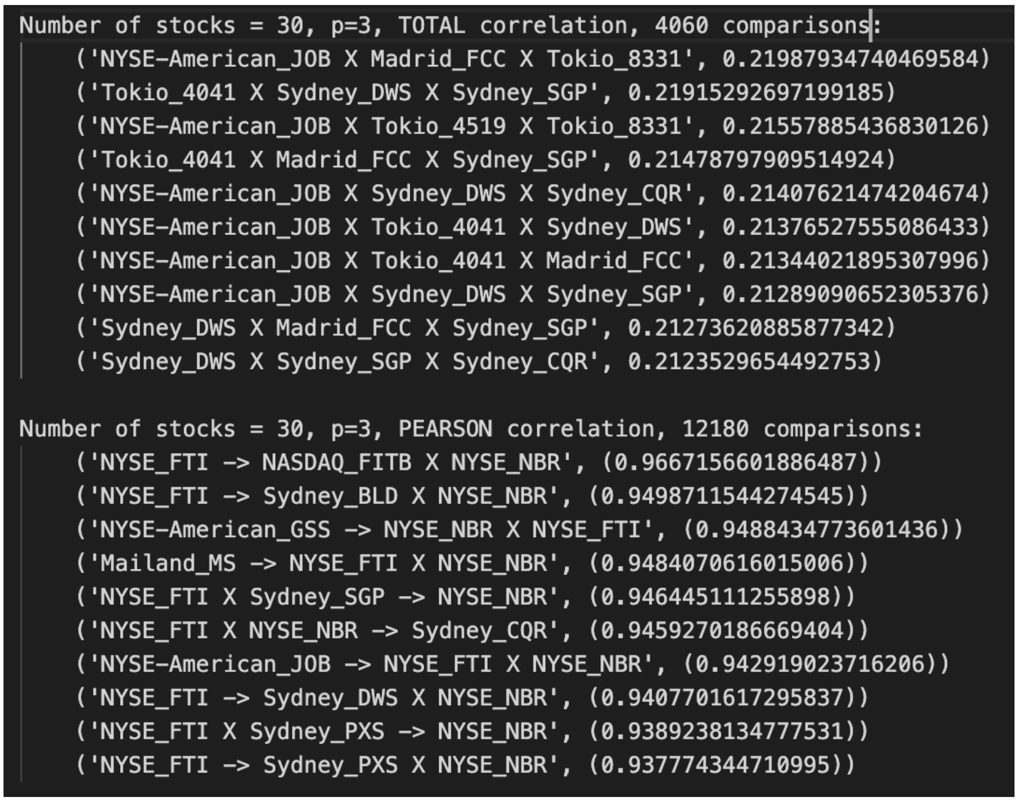
There were far more comparisons to be computed for Pearson correlation compared to Total Correlation however the calculations for Total Correlation were much slower.

There was limited time to optimize the performance of the Amazon Web Service EMR cluster since the usage of the school server was expected in the beginning. Further optimizing would have included finding the best partition on the RDD based on the cores available on each worker instance as well as allocating more driver memory for caching by using storage buckets.

Insights

It is interesting to compare the results from the Pearson correlation and Total Correlation. The Pearson Correlation computes the linear correlation between the vectors while the Total Correlation measure is more general and measures the redundancy or dependency among a set of n random variables. Pearson Correlation assumes for linearity and the absence of outliers. The assumption of linearity is perhaps not accurate when looking at stock data. And if there are outliers in the stock data, some Pearson Correlation measures are invalid. Therefore, it might seem more reasonable to emphasize more on the Total Correlations which does not have as many assumptions.

As stated in the previous chapter Total Correlation is the bottleneck of this analysis. The following results were obtained for the top 10 correlations of both methods for $p = 3$. τ= 0.93 was used for Pearson and τ= 0.212 for Total correlation. In the following figures $->$ means the companies are aggregated together and X means correlation function was applied between the stocks.

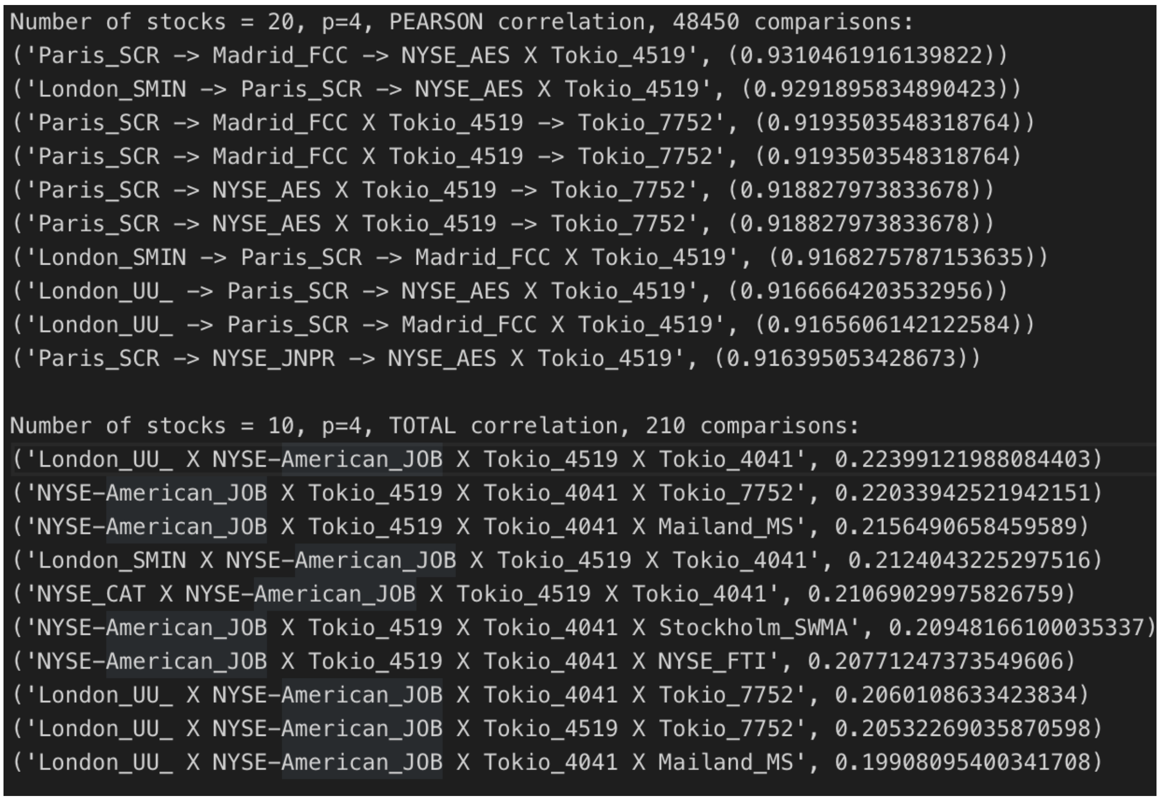


```
Number of stocks = 30, p=3, TOTAL correlation, 4060 comparisons:
    ('NYSE-American_JOB X Madrid_FCC X Tokio_8331', 0.21987934740469584)
    ('Tokio_4041 X Sydney_DWS X Sydney_SGP', 0.21915292697199185)
    ('NYSE-American_JOB X Tokio_4519 X Tokio_8331', 0.21557885436830126)
    ('Tokio_4041 X Madrid_FCC X Sydney_SGP', 0.21478797909514924)
    ('NYSE-American_JOB X Sydney_DWS X Sydney_CQR', 0.21407621474204674)
    ('NYSE-American_JOB X Tokio_4041 X Sydney_DWS', 0.21376527555086433)
    ('NYSE-American_JOB X Tokio_4041 X Madrid_FCC', 0.21344021895307996)
    ('NYSE-American_JOB X Sydney_DWS X Sydney_SGP', 0.21289090652305376)
    ('Sydney_DWS X Madrid_FCC X Sydney_SGP', 0.21273620885877342)
    ('Sydney_DWS X Sydney_SGP X Sydney_CQR', 0.2123529654492753)

Number of stocks = 30, p=3, PEARSON correlation, 12180 comparisons:
    ('NYSE_FTI -> NASDAQ_FITB X NYSE_NBR', (0.9667156601886487))
    ('NYSE_FTI -> Sydney_BLD X NYSE_NBR', (0.9498711544274545))
    ('NYSE-American_GSS -> NYSE_NBR X NYSE_FTI', (0.9488434773601436))
    ('Mailand_MS -> NYSE_FTI X NYSE_NBR', (0.9484070616015006))
    ('NYSE_FTI X Sydney_SGP -> NYSE_NBR', (0.946445111255898))
    ('NYSE_FTI X NYSE_NBR -> Sydney_CQR', (0.9459270186669404))
    ('NYSE-American_JOB -> NYSE_FTI X NYSE_NBR', (0.942919023716206))
    ('NYSE_FTI -> Sydney_DWS X NYSE_NBR', (0.9407701617295837))
    ('NYSE_FTI X Sydney_PXS -> NYSE_NBR', (0.9389238134777531))
    ('NYSE_FTI -> Sydney_PXS X NYSE_NBR', (0.937774344710995))
```

Figure 2: Result of comparisons where p=3

The following results were obtained for the top 10 correlations of both methods for p=4. τ= 0.916 was used for Pearson and τ= 0.199 for Total correlation.

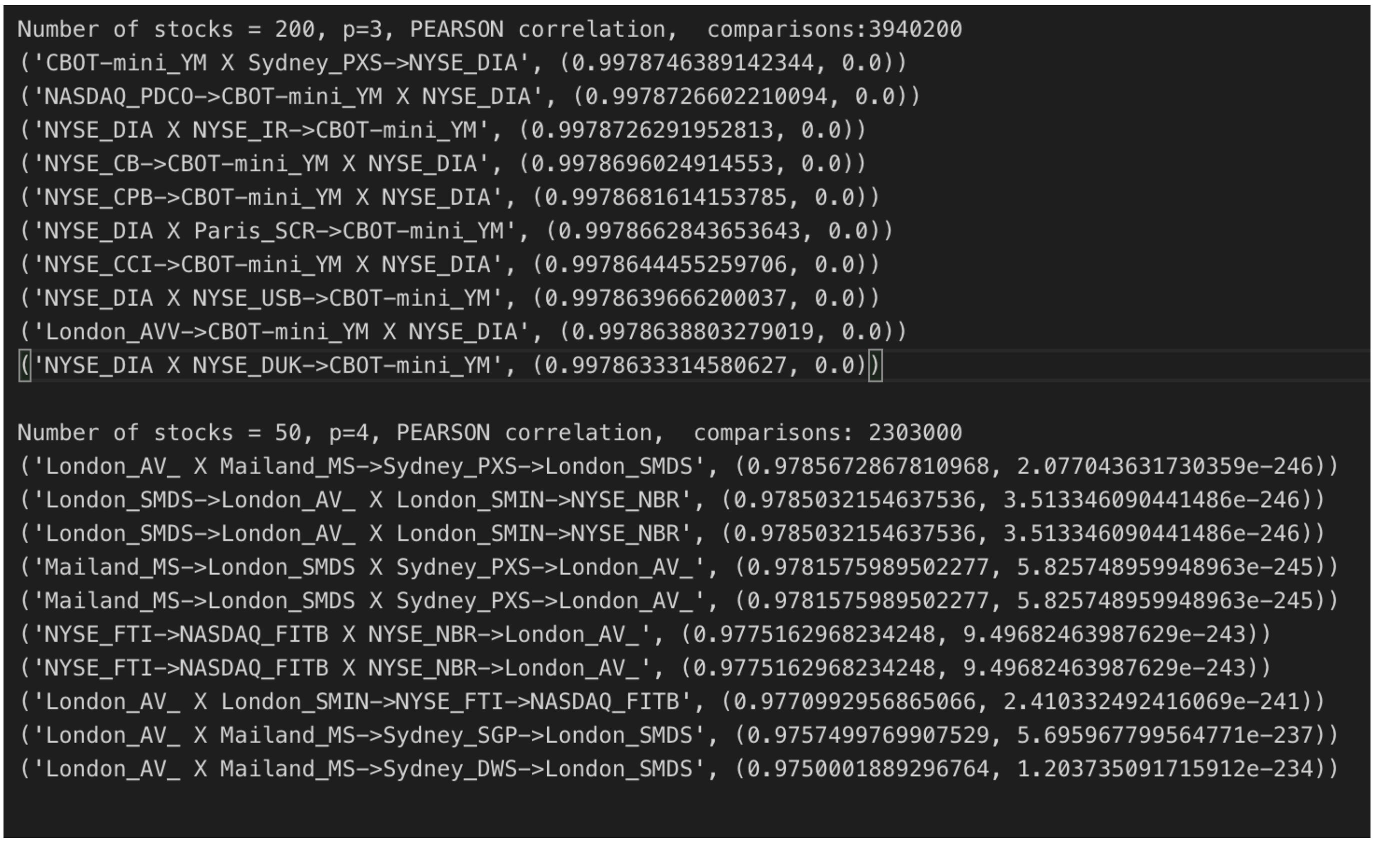


```
Number of stocks = 20, p=4, PEARSON correlation, 48450 comparisons:
('Paris_SCR -> Madrid_FCC -> NYSE_AES X Tokio_4519', (0.9310461916139822))
('London_SMIN -> Paris_SCR -> NYSE_AES X Tokio_4519', (0.9291895834890423))
('Paris_SCR -> Madrid_FCC X Tokio_4519 -> Tokio_7752', (0.9193503548318764))
('Paris_SCR -> Madrid_FCC X Tokio_4519 -> Tokio_7752', (0.9193503548318764)
('Paris_SCR -> NYSE_AES X Tokio_4519 -> Tokio_7752', (0.918827973833678))
('Paris_SCR -> NYSE_AES X Tokio_4519 -> Tokio_7752', (0.918827973833678))
('London_SMIN -> Paris_SCR -> Madrid_FCC X Tokio_4519', (0.9168275787153635))
('London_UU_ -> Paris_SCR -> NYSE_AES X Tokio_4519', (0.9166664203532956))
('London_UU_ -> Paris_SCR -> Madrid_FCC X Tokio_4519', (0.9165606142122584))
('Paris_SCR -> NYSE_JNPR -> NYSE_AES X Tokio_4519', (0.916395053428673))

Number of stocks = 10, p=4, TOTAL correlation, 210 comparisons:
('London_UU_ X NYSE-American_JOB X Tokio_4519 X Tokio_4041', 0.22399121988084403)
('NYSE-American_JOB X Tokio_4519 X Tokio_4041 X Tokio_7752', 0.22033942521942151)
('NYSE-American_JOB X Tokio_4519 X Tokio_4041 X Mailand_MS', 0.2156490658459589)
('London_SMIN X NYSE-American_JOB X Tokio_4519 X Tokio_4041', 0.2124043225297516)
('NYSE_CAT X NYSE-American_JOB X Tokio_4519 X Tokio_4041', 0.21069029975826759)
('NYSE-American_JOB X Tokio_4519 X Tokio_4041 X Stockholm_SWMA', 0.20948166100035337)
('NYSE-American_JOB X Tokio_4519 X Tokio_4041 X NYSE_FTI', 0.20771247373549606)
('London_UU_ X NYSE-American_JOB X Tokio_4041 X Tokio_7752', 0.2060108633423834)
('London_UU_ X NYSE-American_JOB X Tokio_4519 X Tokio_7752', 0.20532269035870598)
('London_UU_ X NYSE-American_JOB X Tokio_4041 X Mailand_MS', 0.19908095400341708)
```

Figure 3: Result of comparisons where p=4

For further inspection, the following are the results of Pearson correlation used on more stocks at once with p=3 and τ= 0.997 and p=4 and τ= 0.975.



```
Number of stocks = 200, p=3, PEARSON correlation,  comparisons:3940200
('CBOT-mini_YM X Sydney_PXS->NYSE_DIA', (0.9978746389142344, 0.0))
('NASDAQ_PDCO->CBOT-mini_YM X NYSE_DIA', (0.9978726602210094, 0.0))
('NYSE_DIA X NYSE_IR->CBOT-mini_YM', (0.997872629195 2813, 0.0))
('NYSE_CB->CBOT-mini_YM X NYSE_DIA', (0.997869602491 4553, 0.0))
('NYSE_CPB->CBOT-mini_YM X NYSE_DIA', (0.9978681614153785, 0.0))
('NYSE_DIA X Paris_SCR->CBOT-mini_YM', (0.9978662843653643, 0.0))
('NYSE_CCI->CBOT-mini_YM X NYSE_DIA', (0.9978644455259706, 0.0))
('NYSE_DIA X NYSE_USB->CBOT-mini_YM', (0.9978639666200037, 0.0))
('London_AVV->CBOT-mini_YM X NYSE_DIA', (0.9978638803279019, 0.0))
('NYSE_DIA X NYSE_DUK->CBOT-mini_YM', (0.9978633314580627, 0.0))

Number of stocks = 50, p=4, PEARSON correlation,  comparisons: 2303000
('London_AV_ X Mailand_MS->Sydney_PXS->London_SMDS', (0.9785672867810968, 2.077043631730359e-246))
('London_SMDS->London_AV_ X London_SMIN->NYSE_NBR', (0.9785032154637536, 3.513346090441486e-246))
('London_SMDS->London_AV_ X London_SMIN->NYSE_NBR', (0.9785032154637536, 3.513346090441486e-246))
('Mailand_MS->London_SMDS X Sydney_PXS->London_AV_', (0.9781575989502277, 5.825748959948963e-245))
('Mailand_MS->London_SMDS X Sydney_PXS->London_AV_', (0.9781575989502277, 5.825748959948963e-245))
('NYSE_FTI->NASDAQ_FITB X NYSE_NBR->London_AV_', (0.9775162968234248, 9.49682463987629e-243))
('NYSE_FTI->NASDAQ_FITB X NYSE_NBR->London_AV_', (0.9775162968234248, 9.49682463987629e-243))
('London_AV_ X London_SMIN->NYSE_FTI->NASDAQ_FITB', (0.9770992956865066, 2.410332492416069e-241))
('London_AV_ X Mailand_MS->Sydney_SGP->London_SMDS', (0.9757499769907529, 5.695967799564771e-237))
('London_AV_ X Mailand_MS->Sydney_DWS->London_SMDS', (0.9750001889296764, 1.203735091715912e-234))
```

Figure 4: Result of comparisons for Pearson of larger number of stocks

The top correlations in Figure 2-3 are different with each correlation measure. Therefore, it can be concluded that the choice of a correlation measure plays a vital role when calculating correlations between stocks. The Pearson Correlation has assumptions that are most likely violated in this stock data, such as absence of outliers and linearity assumptions.

Furthermore, it can be seen that some of the most correlated stocks are actually the same company on different markets, for example Fit Bit, as seen in Figure 2 for the top correlation of Pearson. In further analysis it might be useful to filter these same company correlations out. It might also be interesting to examine those companies that appear often in the top 10 and look into why that is. This is for example `Tokio_4041` and `NYSE_American_JOB` in Figure 3.

References

[1] Karl Fischer. 2015. Common Bitwise Techniques for Subset Iterations. Retrieved from `https://fishi.devtail.io/weblog/2015/05/18/common-bitwise-techniques-subset-iterations/`

[2] Wikipedia, Total Correlation. (n.d.) Retrieved from `https://en.wikipedia.org/wiki/Total_correlation`

[3] Wikipedia, Leibler Divergence. (n.d.) Retrieved from `https://en.wikipedia.org/wiki/Kullback%%80%93Leibler_divergence`