SAPIENZA UNIVERSITY - ROMA

Students: Alessio Borgi, Martina Doku, Giuseppina Iannotti

Reference email:

borgi.*****@studenti.uniroma1.it**

doku.*****@studenti.uniroma1.it**

iannotti.*****@studenti.uniroma1.it**

DELIVERY DATE: July 2022 the 12th

Applied Computer Science and Artificial Intelligence

COMPUTER PERFORMANCE ANALYSIS

Contents

Introduction 3

Tools 3

Data 3

PROCESSOR 3

- Clustering Idle Vs Working 4
- Implementation: 4
- Focus on Performances: 6
- Focus On Time Performances 9
  - Implementation: 9

MEMORY 11

- Page Faults Analysis 11
  - Implementation: 11
- Cache Faults Analysis 13
  - Implementation: 13
- Physical Disk Linear Regression 14
  - Implementation: 14

CACHE 15

- Cache Focus 15
  - Implementation: 15

FINAL COMPARISON 18

- Implementation: 18

Introduction

This project aims to come up with a detailed analysis of data regarding the most important computers' components and to obtain useful conclusions about the performance of different devices from them.

The Project is divided in ***4 parts***.
Parts one, two and three focus on three different components:

1. Processor
2. Memory
3. Physical Disk

For each of them the most relevant features are analyzed to retrieve information about performances and to compare them among the different computers.
The fourth and last part consists of general research of the most relevant features, the ones that influence the most the overall computer performance.

Tools

To achieve that result, we used **Performance Monitoring System,** an application present in every Windows Environment, through which we have collected some data from our own devices, during the execution of pre-established tasks.

Data

To carry on our analysis, we used data regarding these five principal components of a PC:

- Processor
- Battery
- Memory
- Cache
- Physical Disk

PROCESSOR

The first component we decided to analyze is the **processor**. It is commonly known as **CPU (Central Processing Unit)** and is the logical and physical unit or subsystem that oversees the functionalities of a computer. It is by far the most relevant component in terms of performance since it executes all the instructions that the device must execute.

The **CPU** is based on the following cycle:

- **Instruction Fetch:** The Processor fetches the instruction from the memory, present in the address (typically logical) specified by a "specific" register.
- **Decoding:** Once the word has been picked up, it is determined which operation must be performed and how to obtain the operands.
- **Execute:** The desired computation is performed. In the last step of the execution the PC is increased: typically, by one if the instruction was not a conditional jump, otherwise the increment depends on the instruction and on the outcome of this.

Clustering Idle Vs Working

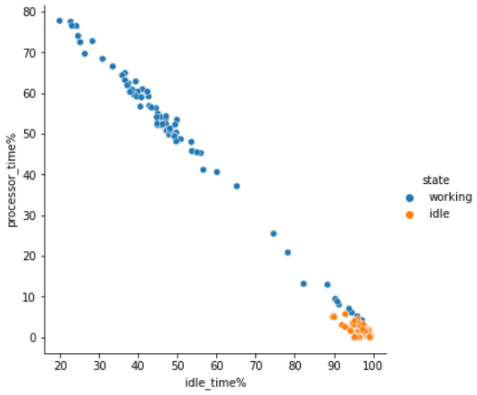
The first part of the examination of the processor consists of studying its features, applying some clustering techniques to ***classify*** in which state it is.

Our Processors can be in two main completely different states: **idle** and **working.** For this reason, we proceed to gather some data while the pc is completely resting and while is Training a ML Model. We then also save the correspondent "label" in the column of the Dataset "state", that has value:

- **Idle**: This state is achieved when our PCs are switched on, but no real tasks are performed. It is, indeed, in a sort of hibernation state, waiting for the user to assign it some tasks.
- **Working**: This state is achieved when the pc is currently being used. In our case, we are particularly referring to the case in which our computers are training on a ML model to strain the pc's Processor.

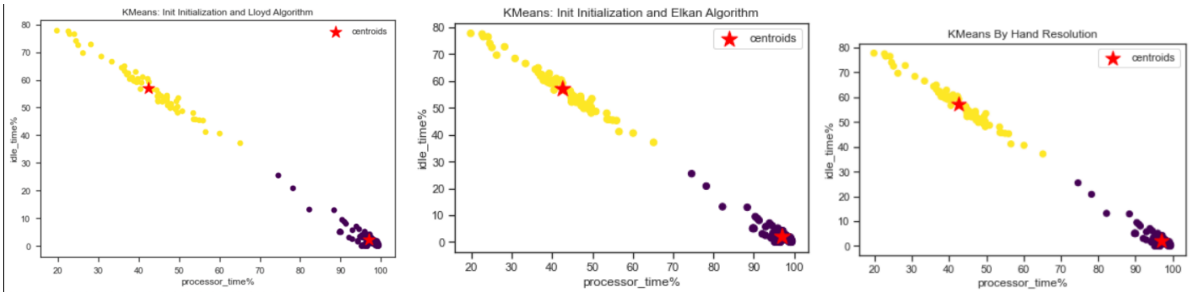
Implementation:

We decided to use, for the study and selection of the state, two **features** that well represent the amount of usage of the CPU, namely: the percentages of **idle time** and **processor time**.
A first step was to plot those data and differentiate them with colors that represent the state in which they were collected.



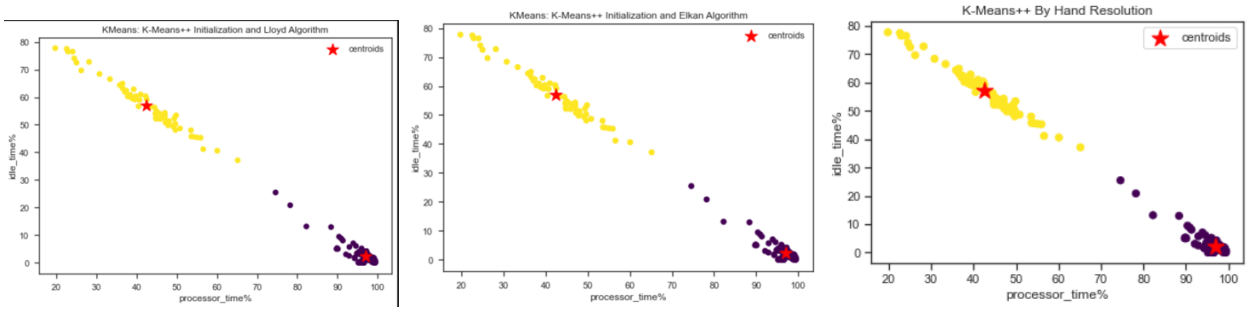
Then we proceeded to classify our data using different classification techniques. The first we tried is the k means algorithm. We used the '**Elbow'** method to decide what value of k fitted at best our case and we obtained a value of **k=2.** We then performed a sample classification run with different versions of k-means:

1. **K-Means Lloyd Algorithm**: Application of the standard K-Means Naïve version.
2. **K-Means Elkan Algorithm**: Application of the K-Means more-advanced version. Points usually stay in the same clusters after a few iterations. It reduces the distance computations by applying the **Inequality's Triangle Inequality.**
3. **K-Means By-Hand Algorithm**: Implementation of a By-Hand K-Means Version.



A further optimization was implemented using a heuristic for the initialization for the **K-means** algorithm that is known as **K-means ++** and classification is performed again with different versions of the algorithm:

1. **K-Means Lloyd Algorithm**
2. **K-Means Elkan Algorithm**
3. **K-Means By-Hand Algorithm**



We tried, however, to see another approach, to try to reach a better Classification, using the **Gaussian Mixture Model** clustering technique, that is an Expectation Maximization algorithm, again with both a random and optimized initialization.
It requires two parameters, the **Mean,** and the **Covariance** to describe the position and shape of each cluster. The model is based on the **Multivariate Gaussian Distribution** which is:

$$\sum \pi \mathrm{N}(x_i|\mu_k\Sigma_k)$$

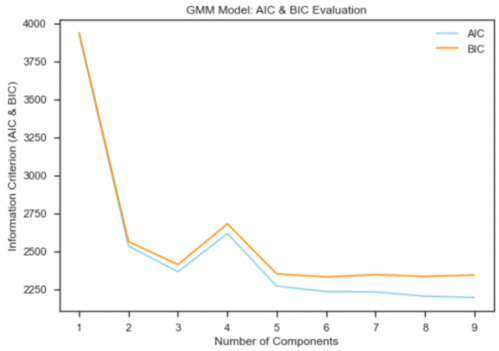
where for each cluster:

- **x**: Vector of a data point.

- **μ**: Vector of the Mean of each Cluster.

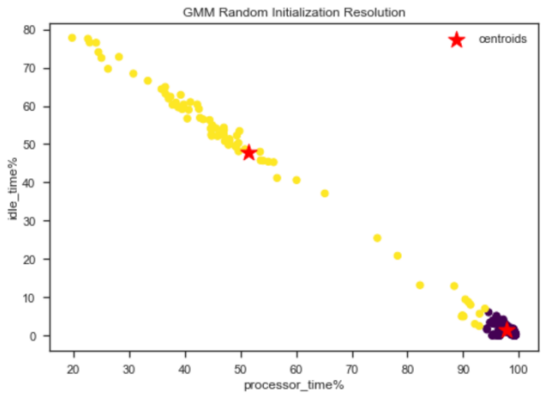
- **Σ**: Covariance Matrix between each Dimension.

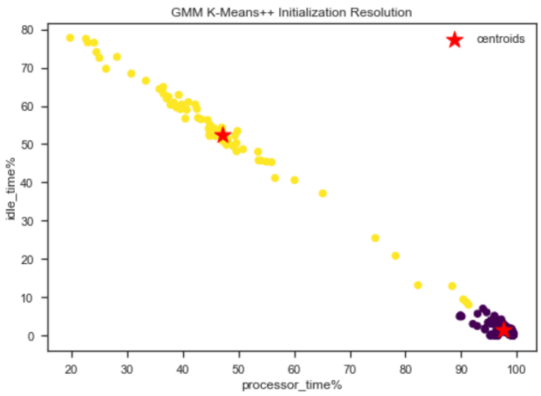
Before applying the GMMs Algorithms, we decided to look at them under an Information Criterion point of view, to find out what was the best number of components we wanted to choose.



After having selectioned **n_components = 2**, we have proceeded in applying the GMM Algorithms:

1. **GMM with Random Initialization**: Application of the GMM Algorithm with a random initialization.
2. **GMM with K-Means++ Initialization**: Application of the GMM Algorithm with the K-Means++ Initialization.





We did a final comparison between the used methods. First, we focus on differences, advantages, and disadvantages of the two methods:

K-means:

Advantages:

1. Simple to understand.
2. Very quick (all that is being computed is the distance between each point and cluster center).
3. Easy to implement

Disadvantages:

1. You must choose K manually.
2. Depends on initial centroid locations.
3. Potential to misrepresent centroid positions due to outliers.

GMM:

Advantages:

1. Can analyze more complex and mixed data w.r.t. K-Means.
2. Can handle outliers more easily.

Disadvantages:

1. More difficult to directly interpret results.
2. Does not directly assign data points to clusters.

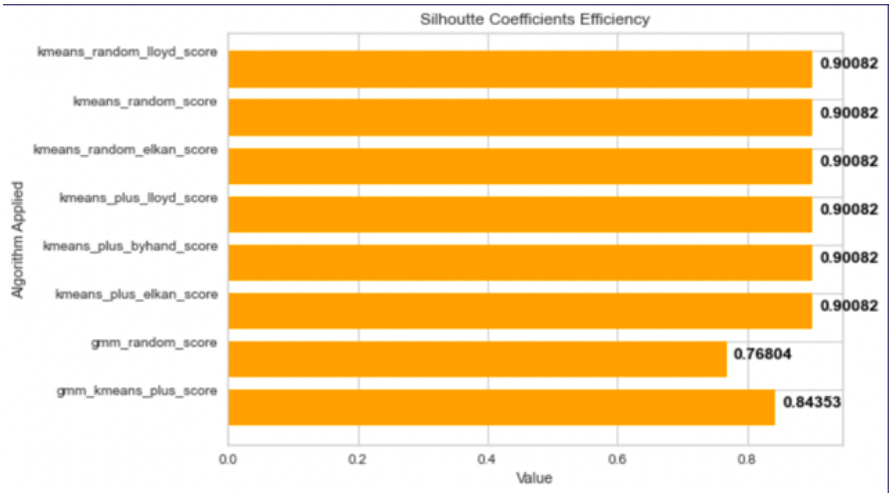
Focus on Performances:

We then decided to focus on Performances of our Algorithms we have used. As a first step, we decide to focus on the **Evaluation** of the **Quality of Clusters** created using Clustering Algorithms in terms of how well samples are clustered with other samples that are like each other. It is used to measure **how dense and well-separated the clusters are**. We therefore compute the **Silhouette Coefficient** where we have that:

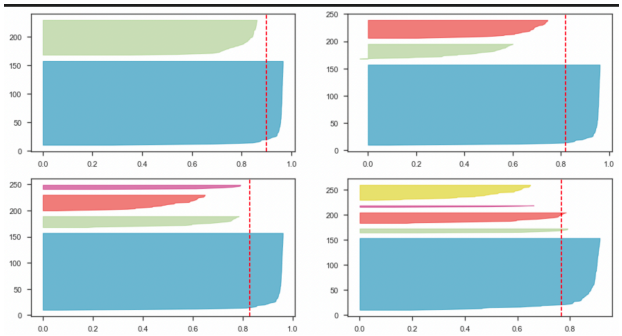
$$S = \frac{(b-a)}{max(a,b)}$$

1. **Mean Intra-Cluster Distance:** Mean distance between the observation and all other data points in the same cluster (denoted by ***a***).
2. **Mean Nearest-Cluster Distance:** Mean distance between the observation and all other data points of the next nearest cluster (denoted by ***b***).

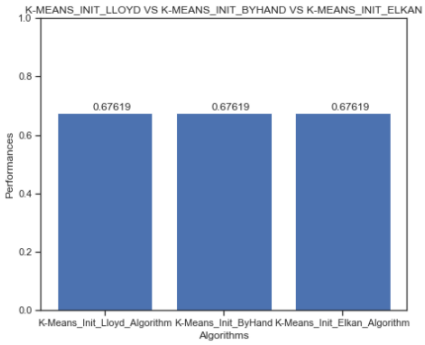
As a result, we have that the more the Score is towards 1, the more the Cluster is dense and well separated. If towards 0, we have overlapping clusters. As result, we have obtained that the **K-Means** implementations Clusters are **better** w.r.t the **GMM**.

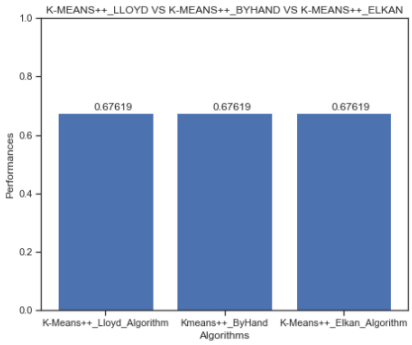


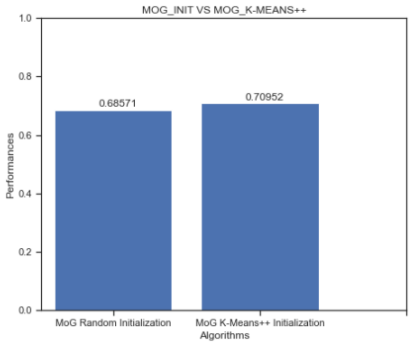
Silhouette Coefficient is also used as method for selecting the optimal value K (no. of cluster). We will use the **Silhouette Plot** to confirm the choice we have done using the **Pre-Processing** step with the Elbow Method on K-Means and AIC/BIC we have done. As a result, it turns out that the value ***3***, ***4*** and ***5*** looks to be **Suboptimal** for the given data due to: Presence of Clusters with below Average Silhouette Scores and Wide fluctuations in the size of the Silhouette Plots. The value **clusters = 2** *is* the **Optimal One**. Indeed, the Silhouette Score for each cluster is almost above the Average Silhouette Scores. Also, the fluctuation in size is similar, even if the Blue Cluster is a little bit big w.r.t. the green one.



As second Performance Measure, we decide to focus on the **Accuracy**, by implementing a By-Hand **Accuracy Score**. We can do this, because we have also the **Labels** since we have collected data by our own and we know what data are about the Idle State and what are about the Working State. The result of our study is that both all K-Means and K-Means++ Implementations returns the same score, whilst the GMM Implementations return a slightly higher Accuracy Score, with the **GMM Implementation** using **K-Means++** as **Initialization** reaching **~71%** of accuracy (w.r.t. the **~69%** of **GMM** with **Random Initialization** and **~68%** of **K-Means** and **K-Means++** Implementation).

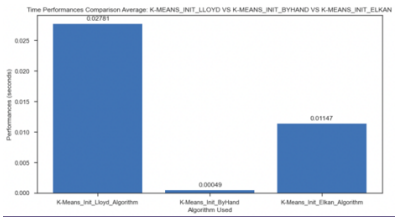


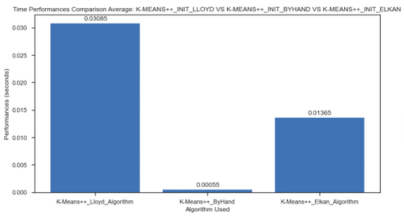


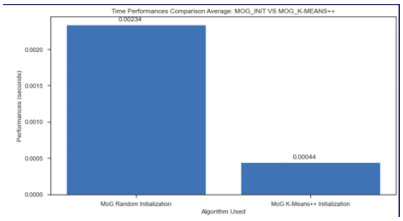


As third Performance Measure, we decide to consider the **Time Performance** time needed to perform the algorithms, and to merge all the information about each of them normalized to have a complete evaluation and to perform a good final choice. To find the fastest Algorithm, we first proceed in a sort of **Pruning Step**, in which we obtain **the Fastest Algorithm** for the three Categories: **K-Means**, **K-Means++** and **GMM**. It is remarkable to say that, due to the Randomicity that is present in all the implementations, we have that the results will not always be the same. However, they always follow the same Hierarchy (in terms of speed). We find out that for what concerns the **K-Means** we have that the fastest Implementation is the **By-Hand Implementation,** followed by the **K-Means** with **Elkan Algorithm**. We find out that for what concerns the **K-Means++** we have that the fastest Implementation is the **By-Hand Implementation,** followed by the **K-Means++** with **Elkan Algorithm**. We find out that for what concerns the **GMM** we have that the fastest Implementation is the **GMM with K-Means++ Initialization Implementation,** followed by the **GMM** with **Random Initialization**. It is remarkable to say that the reason for which the GMM K-Means++ Implementation is a straight line, is because over the 100 Tries, we give to him always the same seed, and therefore the initialization is always the same. We are now going to compare the best result obtained in the Pruning Step of before, to find out what is the fastest Implementation.

We find out that for what concerns the **Clustering Algorithms** we have that the fastest Implementation is the **GMM with K-Means++ Initialization Implementation,** followed by the **K-Means By-Hand Implementation** followed by the **K-Means++ By-Hand Implementation**.







We now proceed in choosing what is the **Best Clustering Algorithm** to choose. We will choose it to depend on:

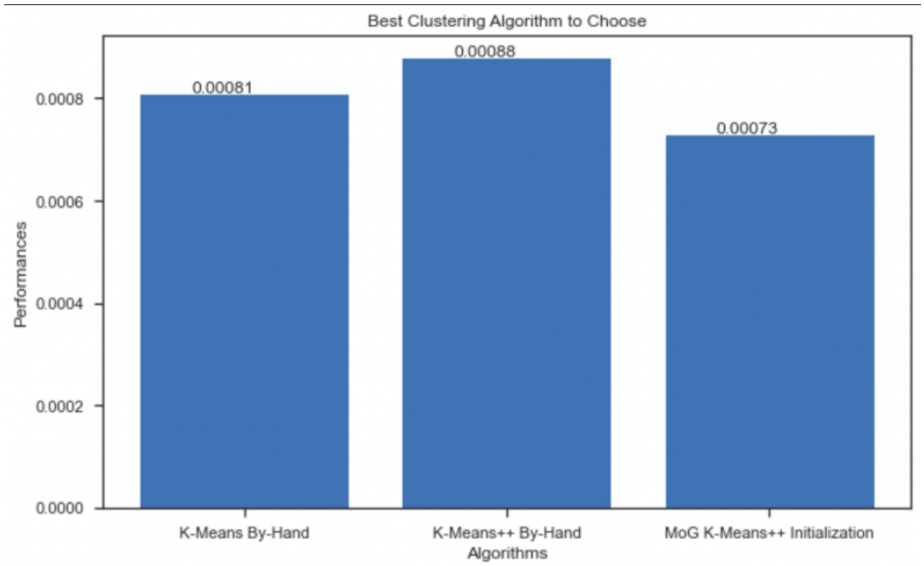
1. **Time Performances**
2. **Accuracy Performances**

However, we have a **Problem**. For what concerns the **Time Performances**, we have that the best Algorithm is the one that has the **Fastest Timing**, therefore the **Smallest Value**. However, for the **Accuracy Performances**, we have the opposite reasoning: in fact, we have that the best Algorithm is the one that has the **Highest Accuracy**, namely the **Biggest Value**.

The **Solution** is:

1. To find the best algorithm, we are going to first take, for the **Accuracy Performances**, the remaining part to arrive at 1, namely "**1 - Accuracy-Score**". In this way, the Algorithm that has the **Highest Accuracy Score**, will have the **Smallest Value** (that will be then, in the following step, added).
2. We can then proceed by **Scaling** the **Accuracy** at the same level of the **Time Performances**, namely, we standardize it, by dividing it by 1000.
3. Finally, **add** the Standardized Result to the Time Performances.
4. The best algorithm to be chosen would be the one to have the less **Time+Accuracy Value**.

From this last analysis, it turns out that, if we consider both the Accuracy and the Time Performances, the best Clustering Algorithm to use is the **GMM with K-Means++ Initialization**. Then, it follows the **K-Means By-Hand Implementation** and the **K-Means++ By-Hand Implementation**.

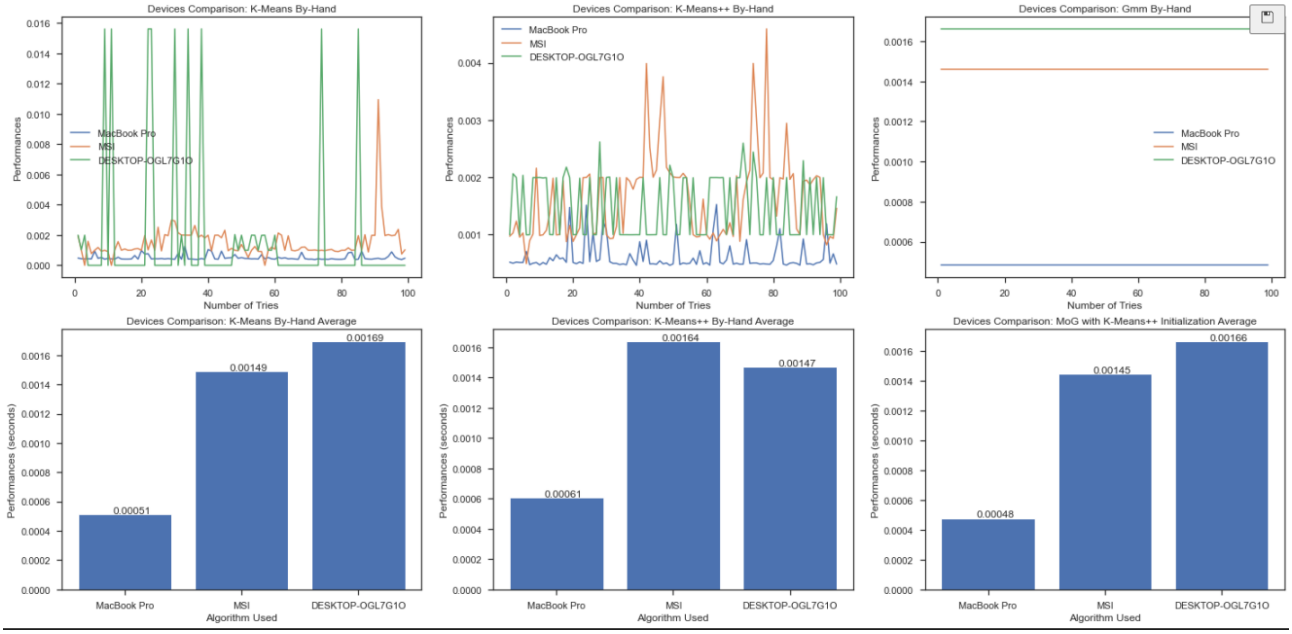


Focus On Time Performances

To continue our analysis on Processor, we now focus on Time Performances w.r.t. different algorithms implementations and **Different PCs Environment**. The time of execution of a task is a great indicator of how performant our compute is, and it's directly connected to the CPU. We then decided to track the time needed on several runs of the algorithms on our devices. We compared, using those data our device to find the one that was, on average faster.

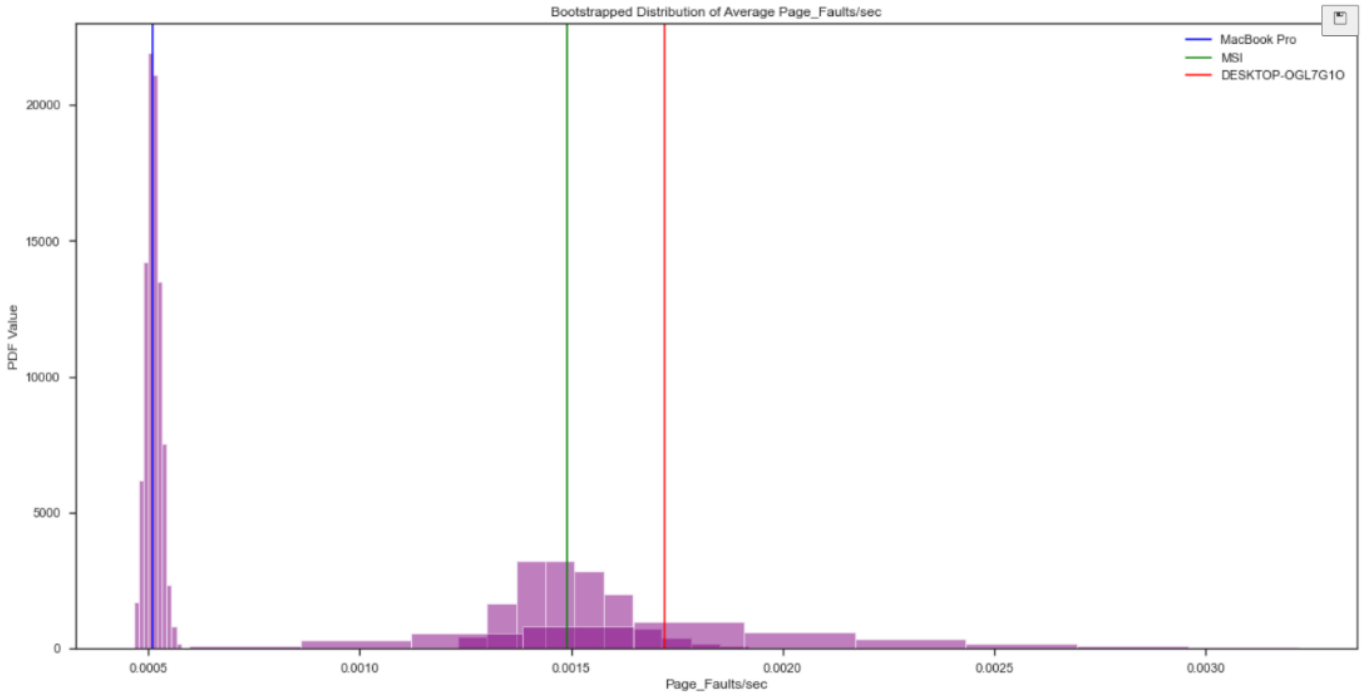
Implementation:

We need to see what the **Fastest** between our **Pcs** is. We will proceed by applying both the Empirical Behavior and the Non-Parametric Bootstrap, in such a way to find out the Mean throughout the Tries, and, for each of the three fastest algorithm (one for each category), find out the fastest PC.



As a result, we have found that the **Mac Book Pro** is way faster w.r.t. the other two pcs. Is faster with a ratio of **3:1** w.r.t. the **Second One** (that for the K-Means and GMM case is the **MSI**, whilst for the K-Means++ is the **OLG**. Instead, w.r.t. the **Third One**, the Mac Book Pro is faster with a ratio of **3.5: 1.**

Finally, we have implemented a **Non-Parametric Bootstrap**, with the aim of seeing whether the Mean we have computed in the previous steps **Empirically**, can also be obtained with this statistical analysis. We are going to apply it, for only the first case, namely the **K-Means By- Hand Implementation**, and not also for the other two, since we use it only as a sort of proof that what we are done is correct. What we observe is that when **B gets bigger** and bigger, we are going to get a **result** that **converges** to the **true empirical mean**.



MEMORY

Continuing our analysis, we proceed studying **Memory**. It is the storage area in which programs are kept when they are running and that contains the data needed by the running programs. The more programs your system is running, the more you'll need. The speed and performance of your system directly correlate to the **amount of memory** of a given device, for example If your system has too little memory, it can be slow and sluggish. But on the opposite end, you can install too much with little to no added benefit.

Page Faults Analysis

We focused our attention on **Page Faults**. For this Analysis, we focus our attention on **Page Faults**. At each logical memory reference, a page table lookup is performed. If the page is already present in the **Memory**, it is used, otherwise must be copied from the **Physical Disk**. This second situation, namely, the one for which the page is not in the Memory, is denoted as **Page Fault**.

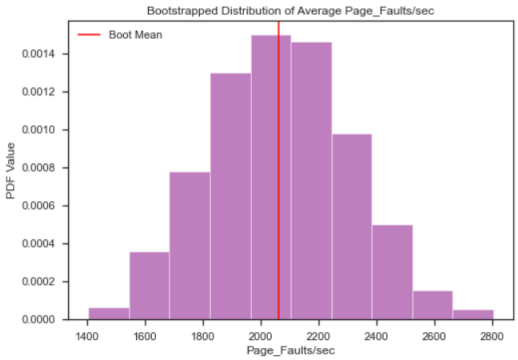
In order, then, to flush the Page in the Memory, we need to implement some solutions for keeping track of the **empty frames** in the Memory, or we need to perform some **Replacement Policy** if all Frames are occupied. Since this operation is highly costly in terms of time, we picked it among the many monitored features to evaluate the device performance.

Implementation:

Our first step was to upload data collected from memory during the execution of a python script that read and saved images from and to memory.
We decide to compute the average of the many 'page faults per seconds' value collected, to have an idea of how many page faults occurred on average during the execution of that program.
To do so, our first approach was to use a **non-Parametric Bootstrap**.
The non-parametric Bootstrap has three stages:

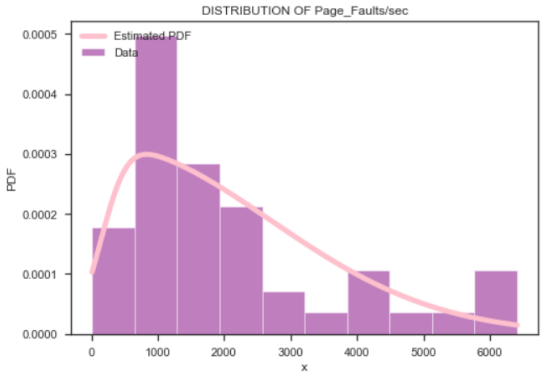
1. Estimate the population (or probability) distribution from the data set.
2. Simulate the sampling from the population distribution that led to the set of observations {xi}.
3. For each sampling, calculate the sample statistic of interest.

For the non-parametric Bootstrap, we simply use the frequency distribution of the n data values as our best guess of the population or probability) distribution.

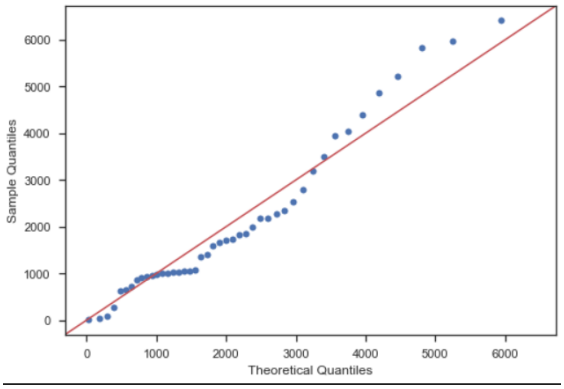


We decide to try a **parametric bootstrap** implementation as well. We decide to use a skew normal distribution to approximate our data. The procedure is the same as the non-parametric Bootstrap approach except for the distribution estimation stage:

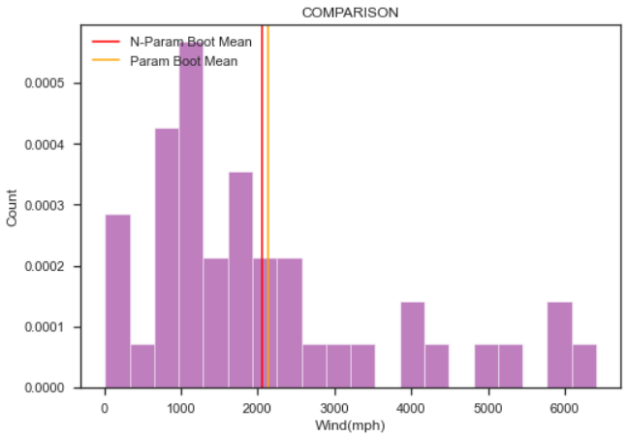
1. **Estimate the distribution from the data**: for the parametric Bootstrap, we select the distribution type we believe the data to come from and then find the MLE parameters for that distribution.
2. **Simulate the data collection**: just as with the non-parametric Bootstrap, we now replace each observation with a sample taken at random from the fitted population distribution
3. **Calculate the sample statistic**: we now run a large number of iterations, each one generating a new Bootstrap replicate, and for each Bootstrap replicate we calculate the sample estimate of the statistic in question



To evaluate how well the probability density function we decided to use, and the parameters obtained with the 'fit' are, we created a relative **qq plot,** to see how much the evaluation was good.
A Q-Q plot is a scatterplot created by plotting two sets of quantiles against one another. Quantiles, also known as percentiles, are points in your data below which a certain proportion of your data fall. Q-Q plots take your sample data, sort it in ascending order, and then plot them versus quantiles calculated from a theoretical distribution. If both sets of quantiles came from the same distribution, we should see the points forming a line that's roughly straight.



From our plot we can conclude that the pdf we chose approximates in a good way our Data.To conclude our computation of the general 'page faults per second' mean, we plotted both parametric and not parametric bootstrapped means to compare them.



After finding the general average on the different device we decide to test if the individual **average** of each of our devices had the '**same' average** and, if not, how different they were. To obtain this information we decided to perform a **one-way Anova test** on the means calculated on the three different devices. It uses the following null and alternative hypotheses:

- **H0 (NULL HYPOTHESIS):** $\mu 1 = \mu 2 = \mu 3 = \ldots = \mu k$ (all the population means are equal)

- **H1 (ALTERNATIVE HYPOTHESIS)**: at least one population mean is different from the rest

We were expecting a rejection of the null hypothesis, considering the several differences between our PCs. However, we could not reject it: the **p-value** is greater than 0.05.

We can then conclude that our computers don't show great differences in performance due to page faults has they have average page faults number that can't be proven to be different.

Cache Faults Analysis

Another aspect we focused on is the **Cache Fault Analysis**. In nowadays computers, there is/are a sort of Support Memory, way faster and smaller w.r.t. the Memory, that has the role to contain the most used Pages. (**80-20 % Rule**). Since the 80 percent of the time, we use the same 20 percent of data, we aim at using this Faster Memory when it is possible. We will have the same situation as with the Memory. If the frame we are requesting is not present in the Cache, we check whether it is present in the Memory, and if it is not, we flush it from the Physical Disk. When it is not present in the Cache, no matter whether it is in the Memory or in the Physical Memory, a **Cache Fault** is happening. Fetching memory is more time consuming, so the less cache faults a device has on average, the more it is performant.

The aim of this study was to check whether the average number of Cache faults is equal for each PC, to understand which one reaches a better performance.

Implementation:

To check if the average of the cache fault of the three computers are equal, we first import the different datasets, then perform an **Anova one way test**. It uses the following null and alternative hypotheses:

- H0 (NULL HYPOTHESIS): $\mu 1 = \mu 2 = \mu 3 = \ldots = \mu k$ (all the population means are equal)
- H1 (ALTERNATIVE HYPOTHESIS): at least one population mean is different from the rest

The Null Hypothesis has been **rejected** since the p-value is less than 0.05.

After having performed a global test, we are interested in the average number of Cache Faults for each pair of PCs. Since they appear to be different (not all three are equal) we want to know if at least one of them has an equal average.

A **two-sample T-test** allows us to conclude that only a pair of PCS "has the same behavior". In the other cases, the null hypothesis is always **rejected**.

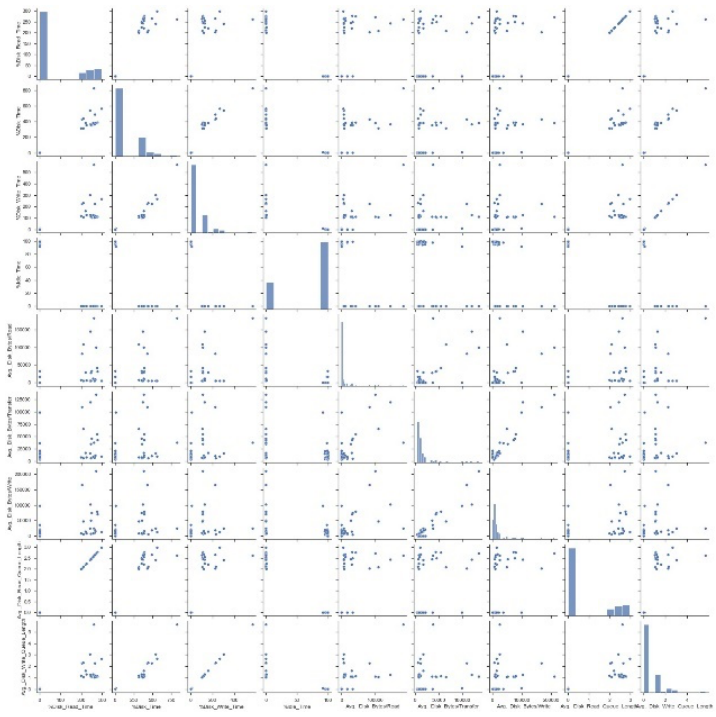
Physical Disk Linear Regression

It is the primary storage hardware/component within a computing device, and it's used to store, retrieve, and organize data. Hard disk drives and tape drives are common examples of a physical drive. A physical drive is primarily attached or installed in the chassis of a computer/laptop and is directly connected to the motherboard through any of the disk communication interfaces, buses or ports. Some physical drives are also external to the computer, such as an external hard disk drive or a USB pen drive. Most types of physical drives can be logically partitioned into one or more logical drives, which operate as a standard physical drive and independently of one another.

The objective in this part is to **analyze Physical Disk Information** and find a way to predict their values.

Implementation:

We start plotting a pair plot of all the features we tracked from our physical disk.



We focus our attention on the relation between the Average disk write queue length and the '**%Disk Time**' that seem to have a great correlation and model two important aspects of that component. The **'%Disk Time'** is the percentage of elapsed time that the selected disk drive is servicing read requests or write requests. The **Average Disk Write Queue Length** tracks the number of write requests that are queued and waiting for a disk during the sample interval, as well as requests in service. As a result, this might overstate activity. If more than two requests are continuously waiting on a single-disk system, the disk might be a bottleneck.

To check if our assumption is correct, we perform linear regression and check the R-squared value to see how well it fits our data.

OLS Regression Results

Dep. Variable:	y	R-squared:	0.887
Model:	OLS	Adj. R-squared:	0.886
Method:	Least Squares	F-statistic:	534.8
Date:	Sat, 09 Jul 2022	Prob (F-statistic):	6.16e-34
Time:	15:38:14	Log-Likelihood:	-393.44
No. Observations:	70	AIC:	790.9
Df Residuals:	68	BIC:	795.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	23.4424	9.133	2.567	0.012	5.218	41.667
x1	192.6872	8.332	23.125	0.000	176.060	209.314

Omnibus:	13.592	Durbin-Watson:	1.234
Prob(Omnibus):	0.001	Jarque-Bera (JB):	52.803
Skew:	-0.025	Prob(JB):	3.42e-12
Kurtosis:	7.255	Cond. No.	1.66

The R value we obtained is quite close to 1, which means that the linear regression it's a good approximation of our data. Finally, we obtained the values of the coefficients that would allow us to retrieve the value of Disk Average Queue Length from the Disk Time % and vice versa.

CACHE

The **Cache** is the last component we are going to analyze. It is a small amount of **faster**, more expensive memory used to improve the performance of recently or frequently accessed data. Cached data is stored temporarily in an accessible storage media that's local to the cache client and separate from the main storage. Cache decreases data access times, reduces latency, and improves input/output (I/O).

When a cache client attempts to access data, it first checks the cache. If the data is found there, that is referred to as a **cache hit**. The percent of attempts that result in a cache hit is called the cache hit rate or ratio.

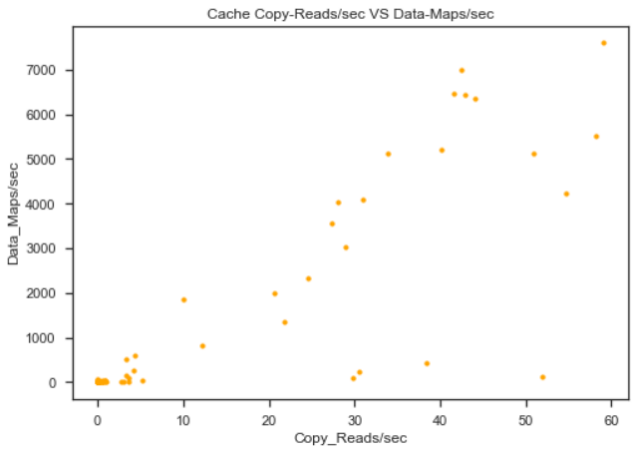
Cache Focus

There are many important aspects we could track about the cache, the one we studied in this case are:

- **Copy Reads/sec**: Measures the frequency of reads from cache pages that involve a memory copy of the data from the cache to the application's buffer.
- **Data Maps/Sec**: This is the frequency at which a file system such as maps a file page into the cache to read the page.

Implementation:

The first thing we did was plotting those two data to see their distribution.



As it can be observed from the graph the distribution of the data vaguely resembles a linear one. We decided to try again to perform linear regression to check whether our assumption is correct.

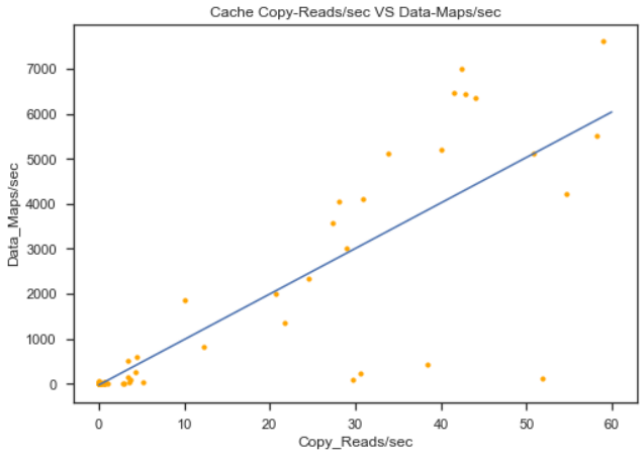
OLS Regression Results

Dep. Variable:	y	R-squared:	0.732
Model:	OLS	Adj. R-squared:	0.729
Method:	Least Squares	F-statistic:	205.4
Date:	Sat, 09 Jul 2022	Prob (F-statistic):	3.58e-23
Time:	15:38:14	Log-Likelihood:	-647.02
No. Observations:	77	AIC:	1298.
Df Residuals:	75	BIC:	1303.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-28.0270	147.651	-0.190	0.850	-322.163	266.109
x1	101.0734	7.053	14.331	0.000	87.023	115.124

Omnibus:	44.711	Durbin-Watson:	1.572
Prob(Omnibus):	0.000	Jarque-Bera (JB):	214.264
Skew:	-1.635	Prob(JB):	2.97e-47
Kurtosis:	10.489	Cond. No.	24.8

As we can see the R squared coefficient is not as high as it was in the previous analysis, even though our distribution seems linear. We try to plot the **regression line** and the residual to compare it to our data.



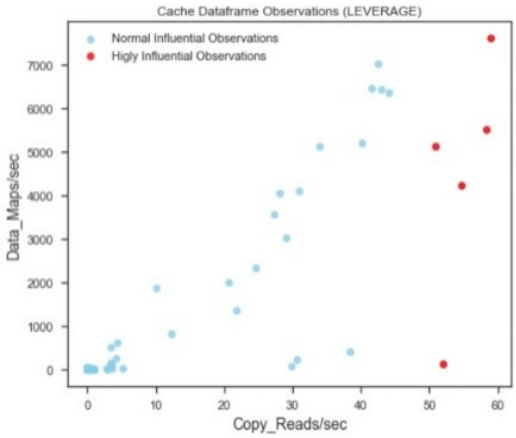
The main cause of that poor fit of the models are the few outliers that we can see in the lower part of the graph. To check what are the main outliers and then try to remove them to increase the R^2 value, we use the **leverage** and **cook's distance** metrics.

- **Leverage**: In statistics and in regression analysis, leverage is a measure of how far away the independent variable values of an observation are from those of the other observations. High-leverage points, if any, are outliers with respect to the independent variables.
- **Cook's distance**: Cook's Distance is an estimate of the influence of a data point. It takes into account both the leverage and residual of each observation. Cook's Distance is a summary of how much a regression model changes when the i-th observation is removed.

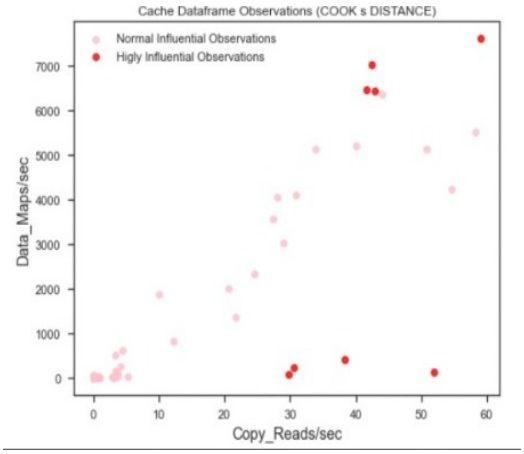
We printed the sorted list of both leverages and cook's distances of our data, and found which points corresponded to the highest values in th two lists.

Respectively we found:

-the **five** highest leverage points



-the eight highest cook's distance points



Since we found many points (13), that are classified as outliers, we decided to remove (at first) only the point that was present in both the 'high leverage group' and 'high cook's distance group'. We plotted again the distribution without that point and computed a new linear regression.

OLS Regression Results

Dep. Variable:	y	R-squared:	0.795
Model:	OLS	Adj. R-squared:	0.792
Method:	Least Squares	F-statistic:	282.6
Date:	Sat, 09 Jul 2022	Prob (F-statistic):	8.31e-27
Time:	15:38:15	Log-Likelihood:	-616.05
No. Observations:	75	AIC:	1236.
Df Residuals:	73	BIC:	1241.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-48.6198	122.799	-0.396	0.693	-293.357	196.118
x1	107.6781	6.406	16.810	0.000	94.912	120.445

Omnibus:	35.612	Durbin-Watson:	1.555
Prob(Omnibus):	0.000	Jarque-Bera (JB):	145.872
Skew:	-1.307	Prob(JB):	2.11e-32
Kurtosis:	9.313	Cond. No.	22.5

The R^2 increased greatly, but we want to try eliminating all the previously found outliers to see if the increase of the Rì2 in that case would be much higher (and thus losing all those points could be a good trade off).

OLS Regression Results

Dep. Variable:	y	R-squared:	0.951
Model:	OLS	Adj. R-squared:	0.950
Method:	Least Squares	F-statistic:	1242.
Date:	Sat, 09 Jul 2022	Prob (F-statistic):	1.26e-43
Time:	15:38:15	Log-Likelihood:	-474.40
No. Observations:	66	AIC:	952.8
Df Residuals:	64	BIC:	957.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-73.9305	44.821	-1.649	0.104	-163.471	15.610
x1	128.7785	3.655	35.235	0.000	121.477	136.080

Omnibus:	29.322	Durbin-Watson:	1.579
Prob(Omnibus):	0.000	Jarque-Bera (JB):	87.404
Skew:	-1.260	Prob(JB):	1.05e-19
Kurtosis:	8.043	Cond. No.	13.8

The R^2 increases again, but we decided that the data loss in eliminating all13 points wasn't worth it, so we used the previous dataset (only the greatest outlier deleted).

We reached the end of this 'Cache analysis' having found a good linear model to compute two of th most relevant features we can track of the Cache: Copy/reads sec and Data maps/sec.

FINAL COMPARISON

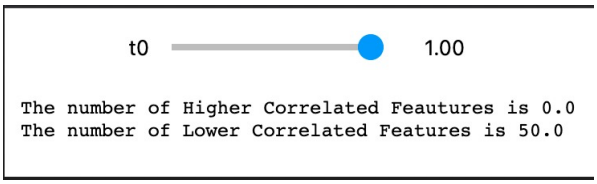
We decided to end our project with a **final comparison** between all the main features of the different component to find the ones that were more **correlated**, hence the features whose value have an influence on values of other components and that can be considered relevant.

Implementation:

We started by creating one **final dataset** comprehensive of the features coming from different components. Having done that, we computed, for each couple of features the correlation using **non-parametric bootstrap**. We inserted those correlations (the absolute value of the correlations) inside a matrix.

We decided to filter out the components that had the greatest correlation.

First, we created a widget to select the value of the threshold for the correlation we wanted to filter and the respective number of couples that satisfy that requirement.



Then we decided to retrieve the most influential features setting manually a threshold for the **correlation** of >**0.9** and we obtained the following:

```
[('Remaining_Capacity', 'Discharge_Rate'), ('Remaining_Capacity', 'Available_Bytes'), ('Discharge_Rate', 'Available_Bytes'), ('%Disk_Time', 'Avg._Disk_Write_Queue_Length')]
```

As a last step of our analysis we also retrieved the features that were less correlated that are those who have a **correlation** that is **<0.1**. We obtained the following:

```
[('%Processor_Time', 'Page_Faults/sec'), ('Interrupts/sec', '%Disk_Time'), ('Interrupts/sec', 'Avg._Disk_Write_Queue_Length'), ('Page_Faults/sec', 'Data_Flush_Pages/sec')]
```

Thank you for your Attention,

Best Regards

Alessio Borgi

Martina Doku

Giuseppina Iannotti