UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR DEGREE ON INFORMATICS ENGINEERING

DATA MINING

Q1 Course 2020-2021

Report of the deliverable

Project Development

GROUP 10

Authors:

Manel AGUILAR BARROSO

Daniel CANO CARRASCOSA

Oriol CATASÚS LLENA

Jesus MOLINA ROLDAN

Eduard ORTUÑO GARROTE

Adrià VENTURA HERCE

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Index

1. Motivation

1.1. Work motivation

The motivation for this study is given by our love for computer hardware, which as is well known we do computer engineering and understanding them is what moves us.

At the beginning we were looking at various datasets that we liked to be able to make the work more enjoyable, we found about video games and computer-related topics. The only one that met the demanded conditions was the one we found, this had data on the graphs made up to 2017.
This was very interesting because we could investigate one of the most important components of computers while doing a job for the university.

1.2. Description of the problem

Once we have the dataset we proceed to decide on which one we are going to focus on. We take a few and see that texture rate can be a good option to be response.

To understand the texture rate of a GPU, it helps first to know how it relates to the graphics card of a device and how they're both tied to the GPU clock speed. Also known as engine clock, GPU clock speed indicates how fast the cores of a graphics processing unit (GPU) are. The function of these cores is to render graphics; therefore, the higher the GPU clock speed, the faster the processing. Now, texture rate, also called texture fill rate, refers to the number of textured pixels that a graphics card can render on the screen every second. This means that it refers to the speed at which a graphics card can essentially "fill" the screen with pixels that may be related to the GPU power.

2. Data Source presentation

The data source chosen in this project was extracted from: https://www.kaggle.com/iliassekkaf/computerparts. This dataset gathers detailed information about 3406 different models of Graphic Processor Units (GPU), from 2000 until 2017.

The information, divided into 21 variables, covers the most important information to describe a GPU, ranging from general information like model name, release date or architecture to technical information like multiple memory and processor measurements (frequency, bandwidth, etc.). The dataset provides 9 numerical variables, 3 binary variables and 9 categorical variables, with a total null rate about 8.03%; therefore the data meets the requirements of the subject and is adequate to develop our goals.

3. Metadata

3.1. Meaning of each column

- Variable: name of the variable in question.
- Modalities: values that the variable can get. For example, if it is a boolean variable, it can take the value of true or false.
- Meaning: brief description of the variable.
- Type: if the variable is numerical, binary or qualitative.
- Measuring Unit: if it is a numeric variable, in which units it is measured.
- Missing Code: if the variable is missing, which value it has.
- Measuring Procedure: how is measured if it is a numeric variable.
- Range: the range of values it takes if it is a numeric variable.
- Role: a variable can be either explanatory or response. Is response if it is the objective of the study that is being carried out or is explanatory if the variable is one that explains changes in that variable.

3.2. Metadata Matrix

https://docs.google.com/spreadsheets/d/1Wy-xxOKpn6Atnwx2XWOc9no4cVE2f88hltUpdnENesg/edit?usp=sharing (access to the link with UPC mail).

3.3. Inclusion and exclusion criteria

We need to detect variables that we don't want since these variables don't give us any useful information. However, in our case we have only detected this in one variable, but we wanted to point out that case in order to point out that case since on a first look it could seem that we have deleted one variable without any judgment.

The variable that we are talking about it's called **Integrated** which is a variable of boolean type that tells us if our graphic card is integrated or not with the CPU. It could look like an useful variable, however when we take a look and compare it with the **Dedicated** variable we observe that both of them are the same since when Integrated is FALSE, Dedicated is TRUE and the same in REVERSE, therefore we decided to delete that variable because of the lack of useful information that it provided us.

We chose these variables because they meet the condition of having less than 15% missing and we consider that they could be useful variables for describing the GPU.

4. Univariate Descriptive

4.1. Explanation of the script

Now that we have our Metadata Matrix we are going to analyse each variable independently. However, in order to do that we will use a script that was provided to us for the teachers, so to know what this script does we are going to analyze it.

In the first section of our code we can see how we load the data matrix, which has been created by us before after a cleaning process described in the previous sections, and it checks what datatype is and the dimensions of the whole data matrix. Also it visualizes all the variable's names.

```
dd <- read.table("clean_table_numericas.csv",header=T, sep=",", dec=".")
class(dd) # Datatype
n<-dim(dd)[1] # Number of rows or number of GPUs
K<-dim(dd)[2] # Number of variables for each GPU
names(dd) #Variable's names
```

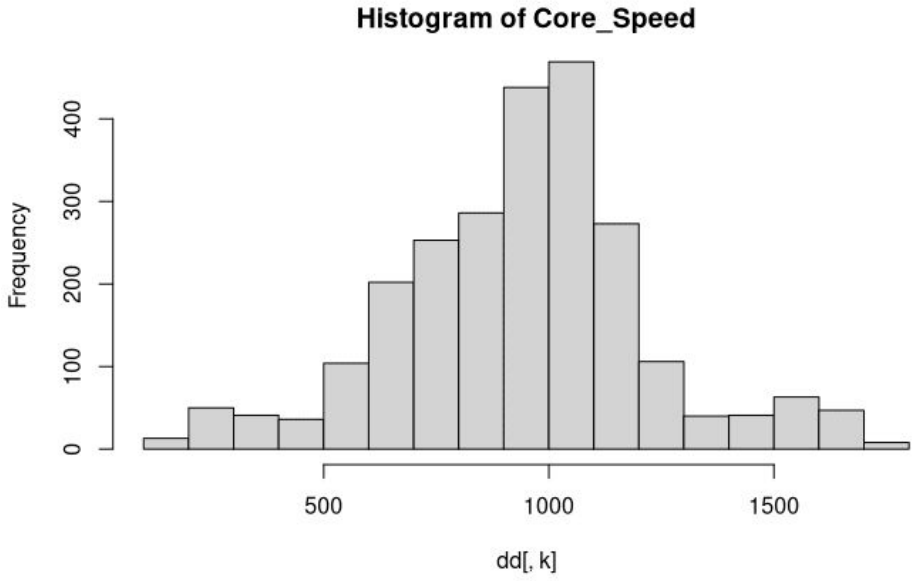
After visualizing all the properties of our data matrix it decides which are going to be the "useful" variables to visualize, in order to do that it executes the following script:

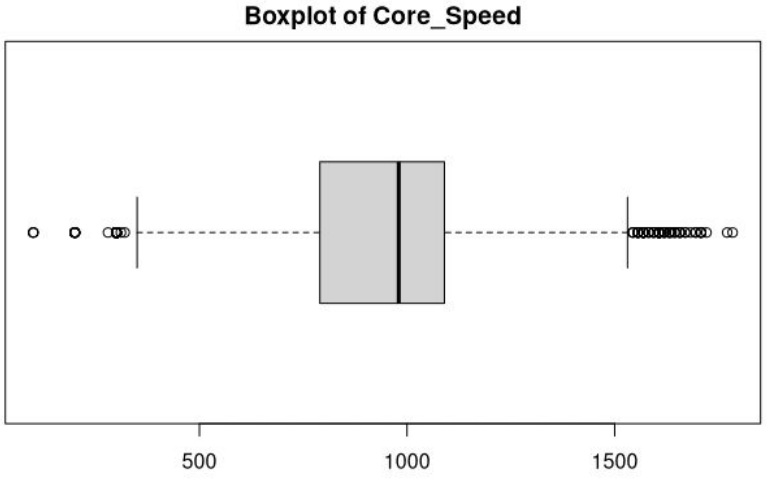
```
actives ← c(1:K)
```

Which generates a vector that goes from the element number 1 to K, for example (1,2,3,4,5) which will indicate the index of the selected columns to analyze, that could be useful if our case was that, but since it's not our case we will set actives as we seen before in order to select all the possible variables.

Finally once we have selected all the variables that we want to analyze we only need to generate all the plots for each variable, but we already have that done in the script provided by our teachers. Therefore, we only need to execute it to visualize the plots, nevertheless, we want to point out that the plots that we will see are the ones already implemented however we could add more if we wanted but for our case it is not necessary.

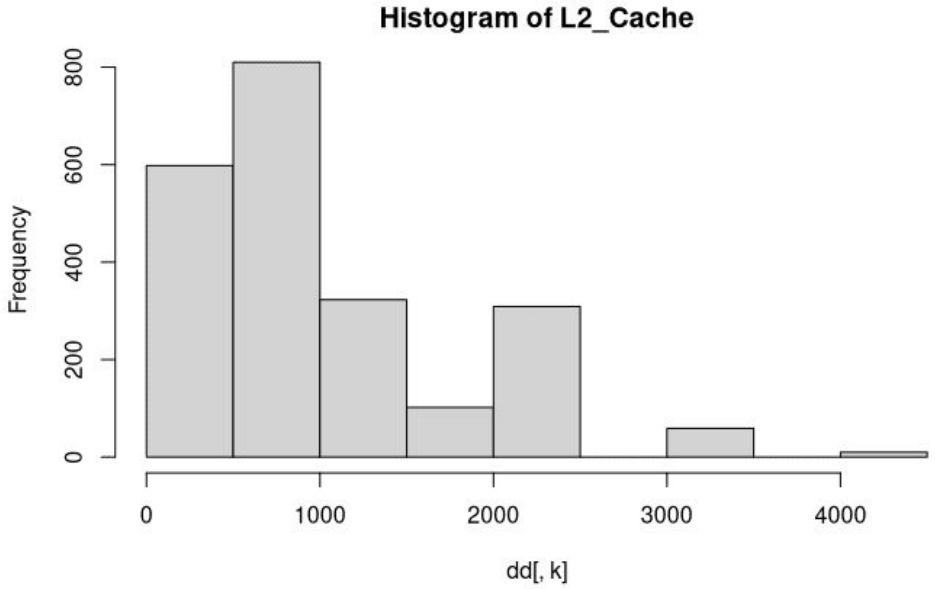
4.2. Results

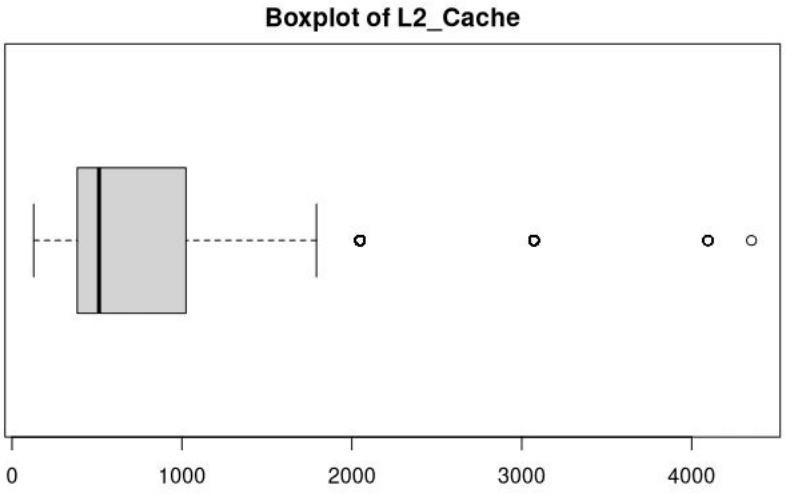




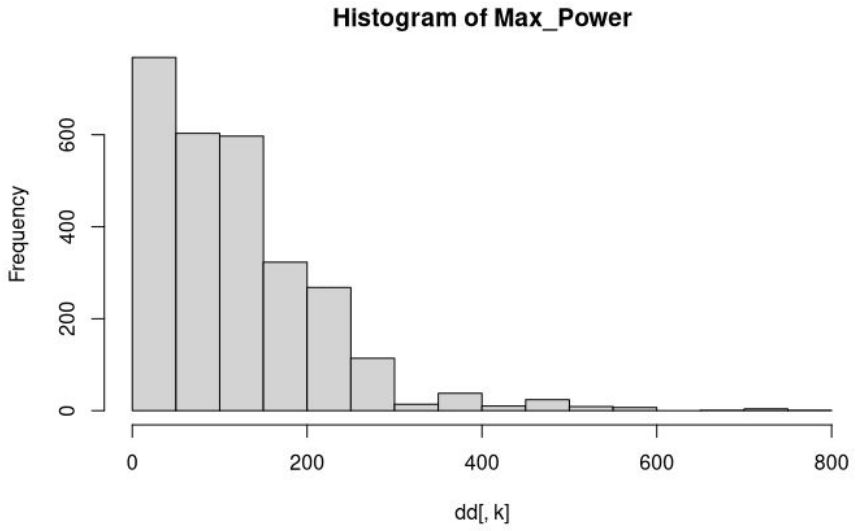
The histogram shows a normal distribution being around 1000 MHz the most frequent core speed with more than 800 GPUs. The less frequent core speeds are situated below 500 MHz and above approximately 1250 MHz. In the boxplot we can observe that there exists some

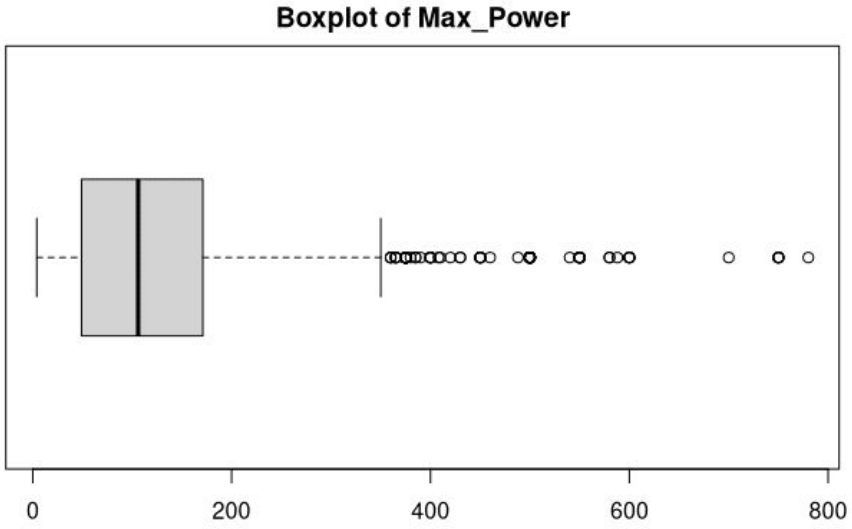
outliers, especially above 1500MHz of core speed, but the majority of values remains between 250MHz and 1500MHz.



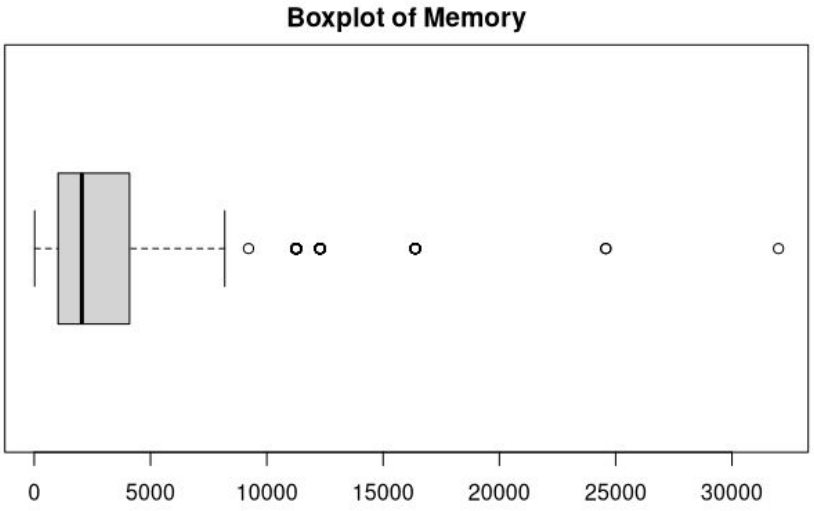


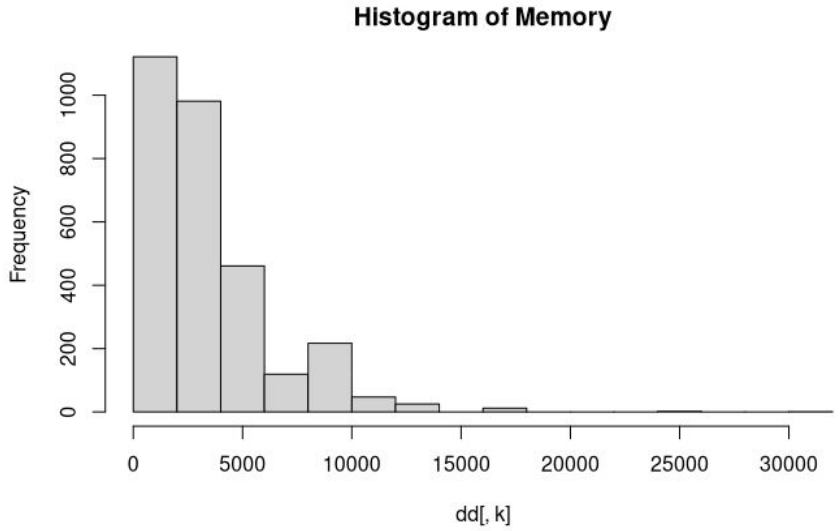
The histogram and the boxplot clearly displays that most GPUs have a L2 cache of between 500 KB and 1000 KB. The highest value without counting outliers is about 2000 KB, with 3 outliers scoring more than 3000 KB of L2 cache.



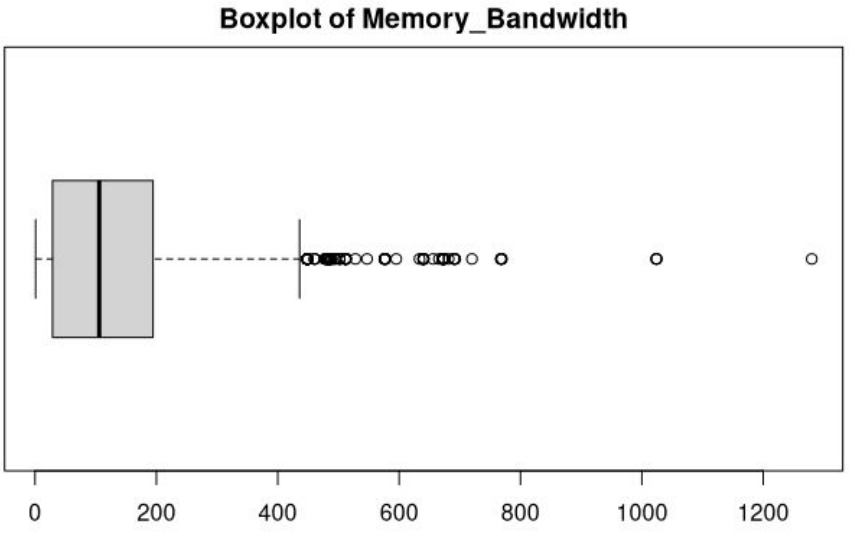


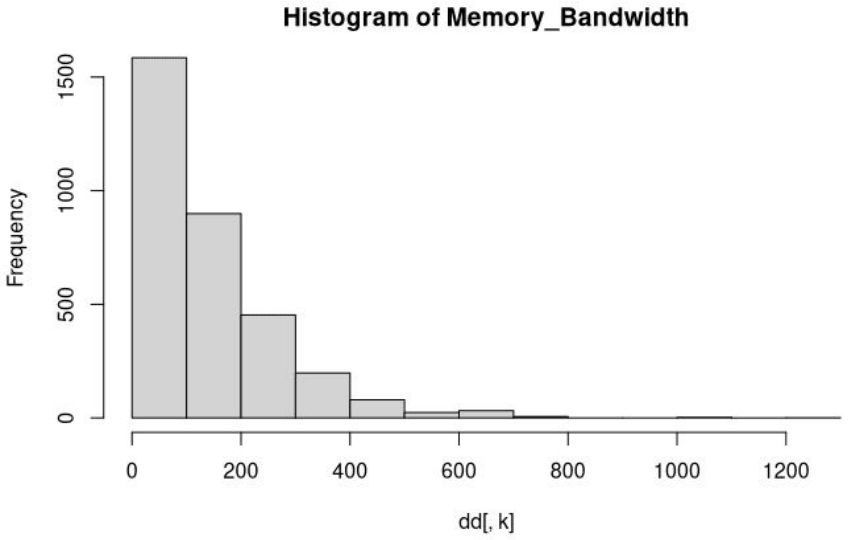
The histogram of Max Power shows that the most frequent value is between 0W and 50W. In fact, as most values are null or have unusually low values, it's impossible for any type of GPU to be powered with 4W or 5W. The boxplot graphic shows that 50% of values are between 50W and 200W, with the mean situated approximately at 100W. Nearly all values are found between 0W and 350W, with some outliers reaching 800W.



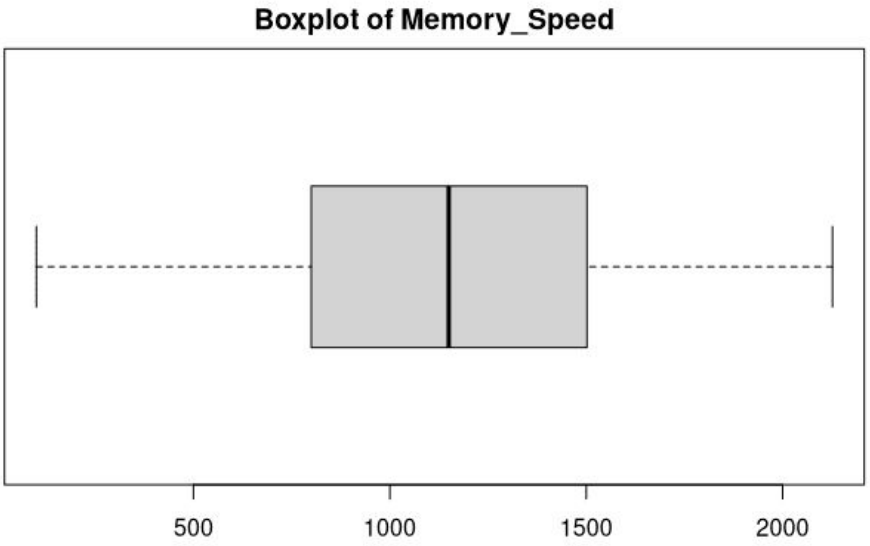


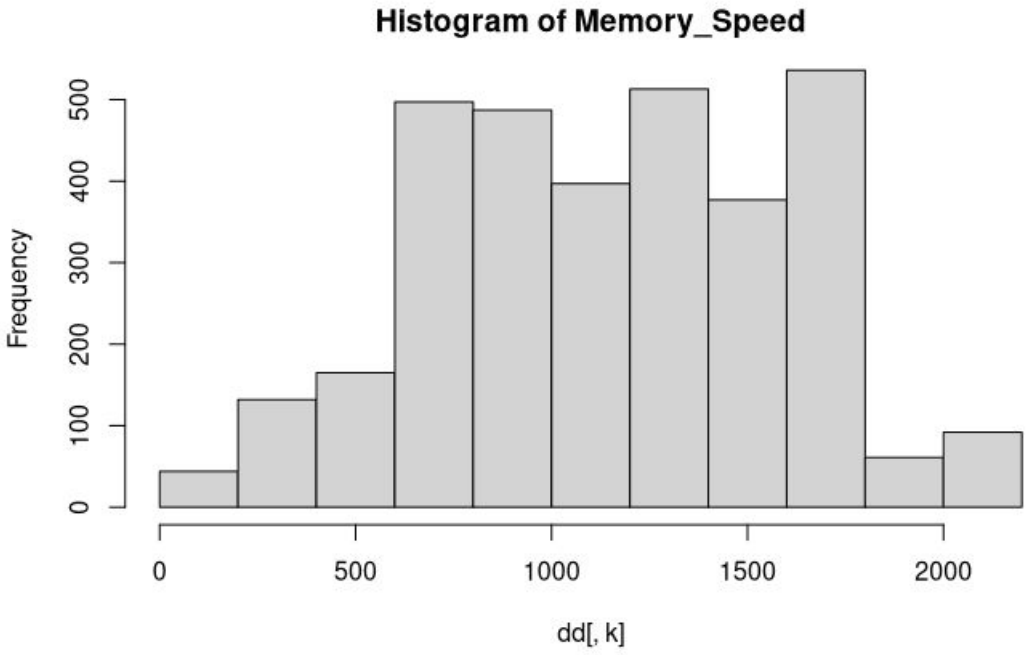
The histogram of Memory variable shows that the most frequent values are located between 0 and 5000. The boxplot graphic shows that 50% of values are placed between 1250 and 5000, with the mean situated approximately at 1500. Nearly all values are found between 0 and 7500, with 6 outliers ranging from 10000 until more than 30000.



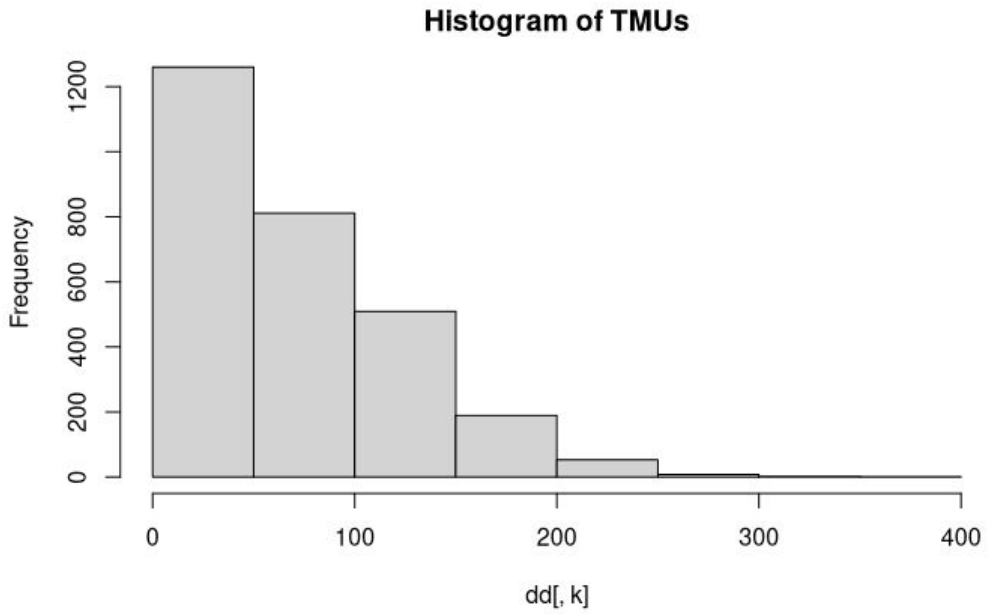


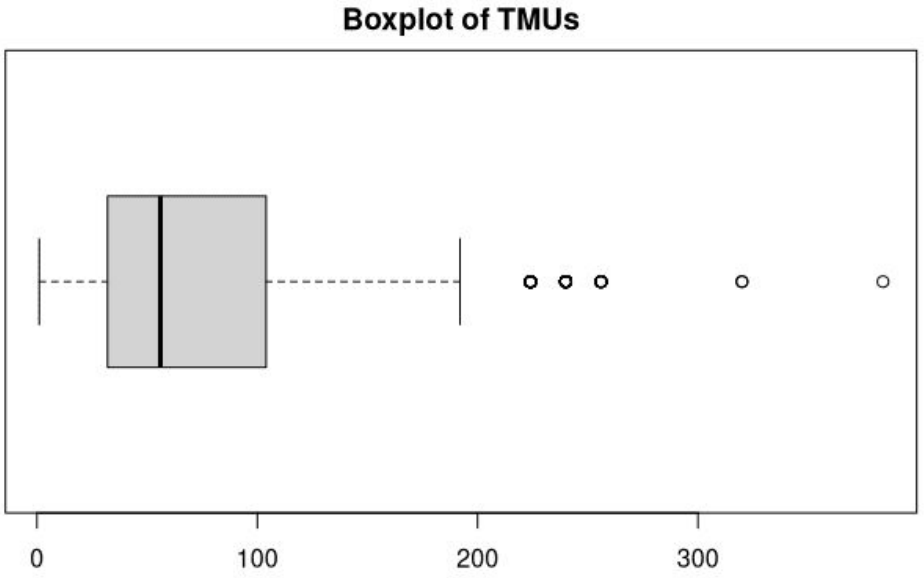
The histogram of Memory_Bandwidth shows that the vast majority of values are concentrated between (0,400]. Also, observing the boxplot, we can see that above 800 could be considered as outliers because it breaks the chart layout since there are several values between [400,800].



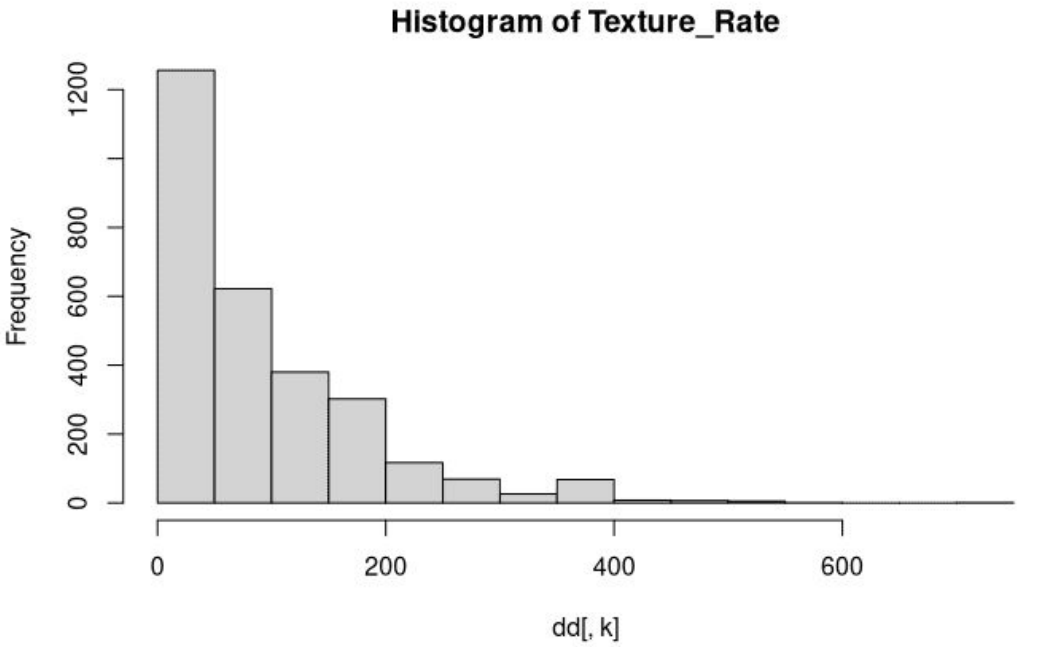


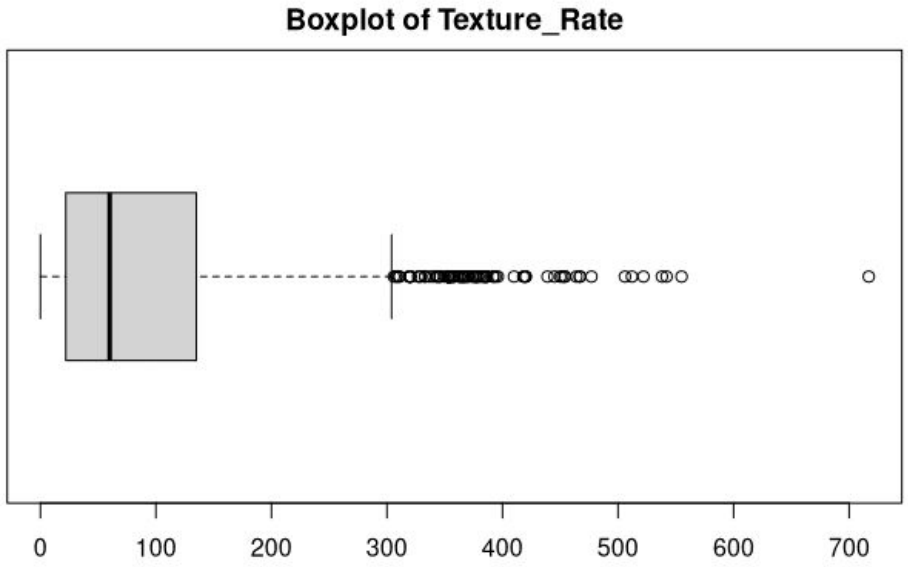
If we observe the histogram of Memory_Speed, we can guess that it follows (more or less) a normal distribution since the vast majority of values are at the middle of the histogram. Also, we can see that on the boxplot, it is almost symmetrical because of the normal distribution.



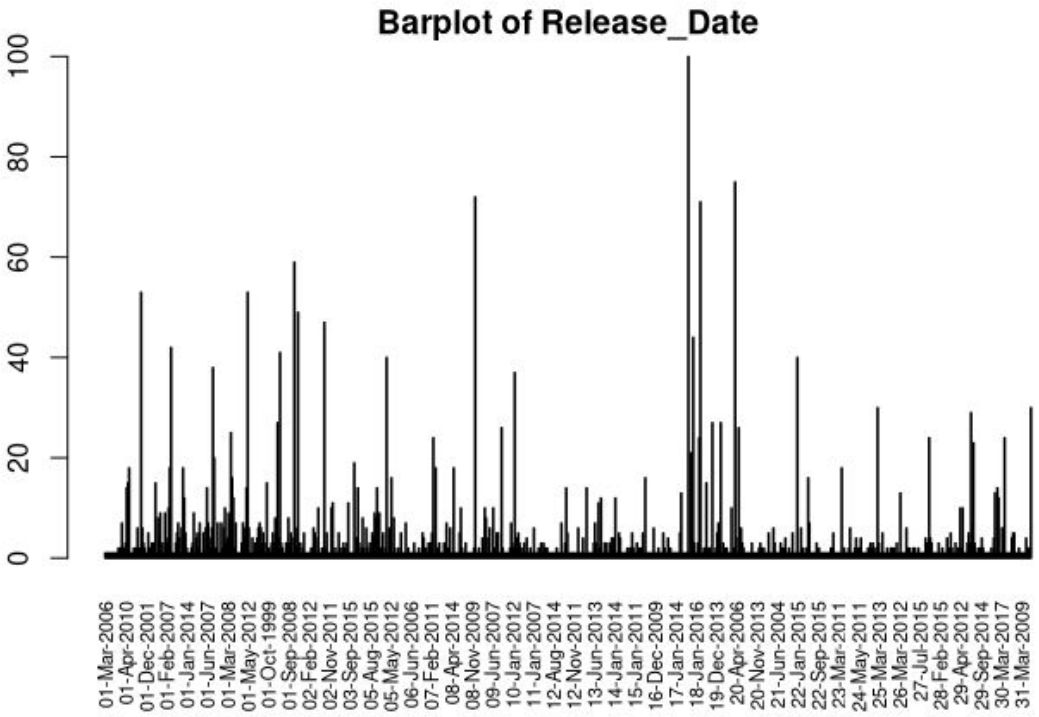


As can be seen from both graphics boxplot and histogram, the vast majority of values are between (0, 100] and is that so because the majority of the GPUs have a lower number of texture mapping units. So, obviously, the histogram represents a logarithmic decrease, so that the values that are above 200 are probably fairly recent GPUs. Despite this, as it breaks our distribution of the boxplot and also the histogram, we should consider the values above approximately 230 as outliers.

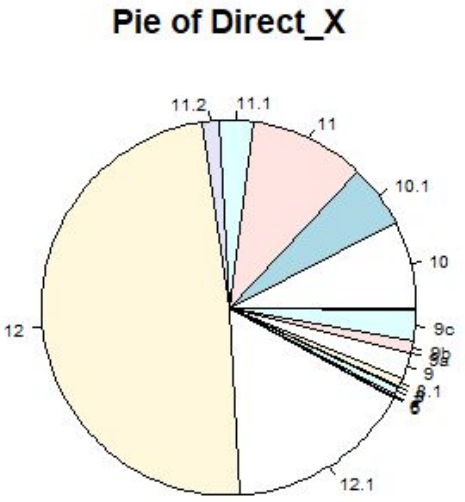


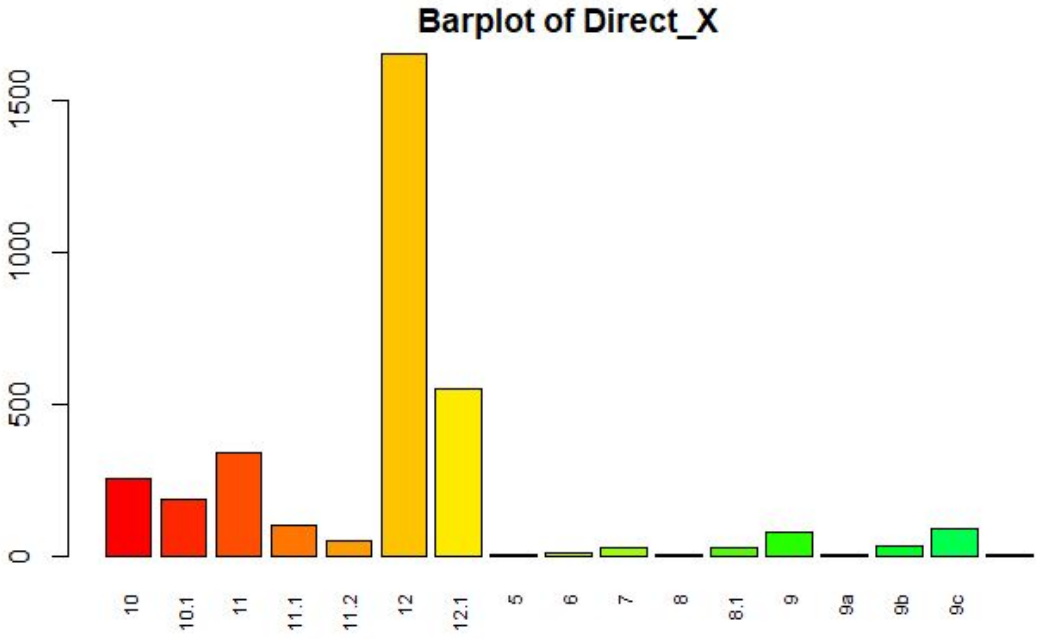


Firstly, if we see the histogram of Texture_Rate, above the value of 400, we cannot take any good information as the majority of values are between (0,400] which we can also see it on the boxplot. As we saw, there are some other outliers separated from each other from 400 and on.

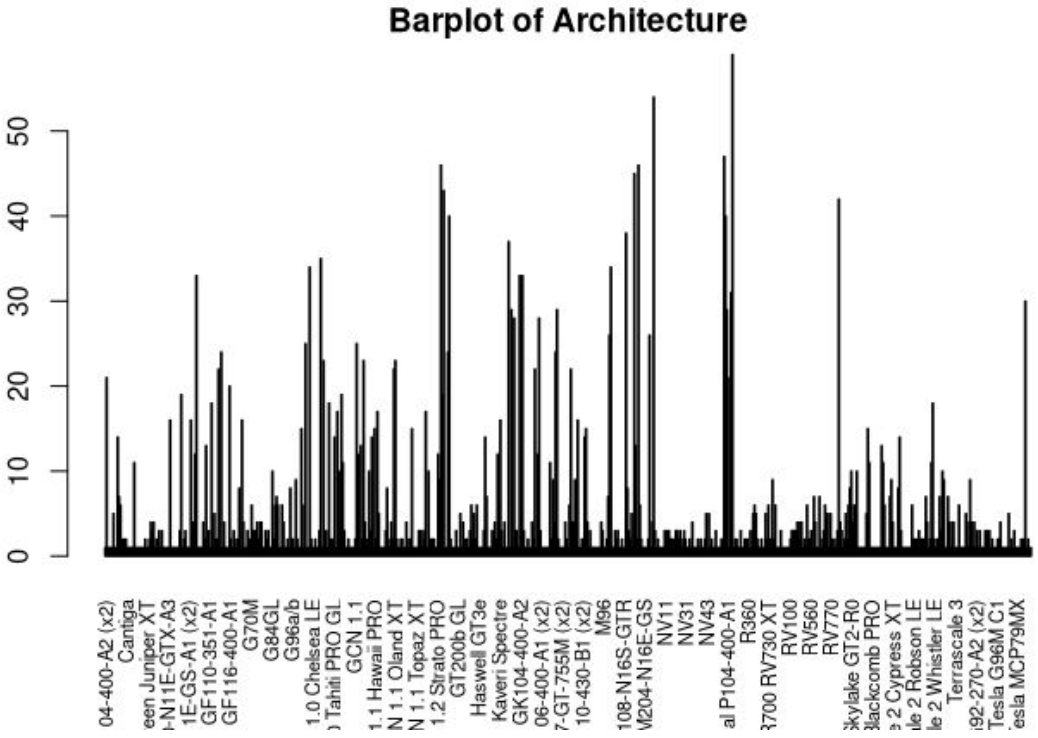


The barplot of Release_date tells us that a huge number of GPUs went on the market on 18th January, 2016. From the barplot we can only observe that all the graphics have been released more or less uniformly.

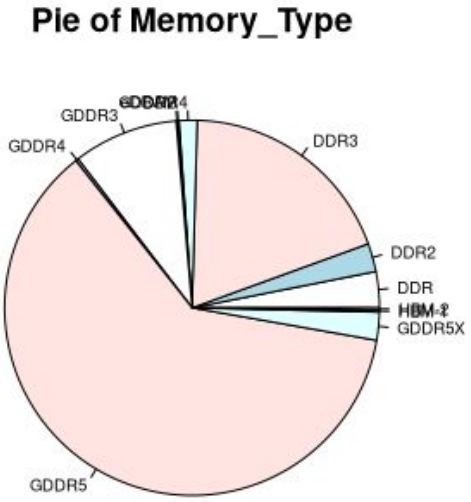


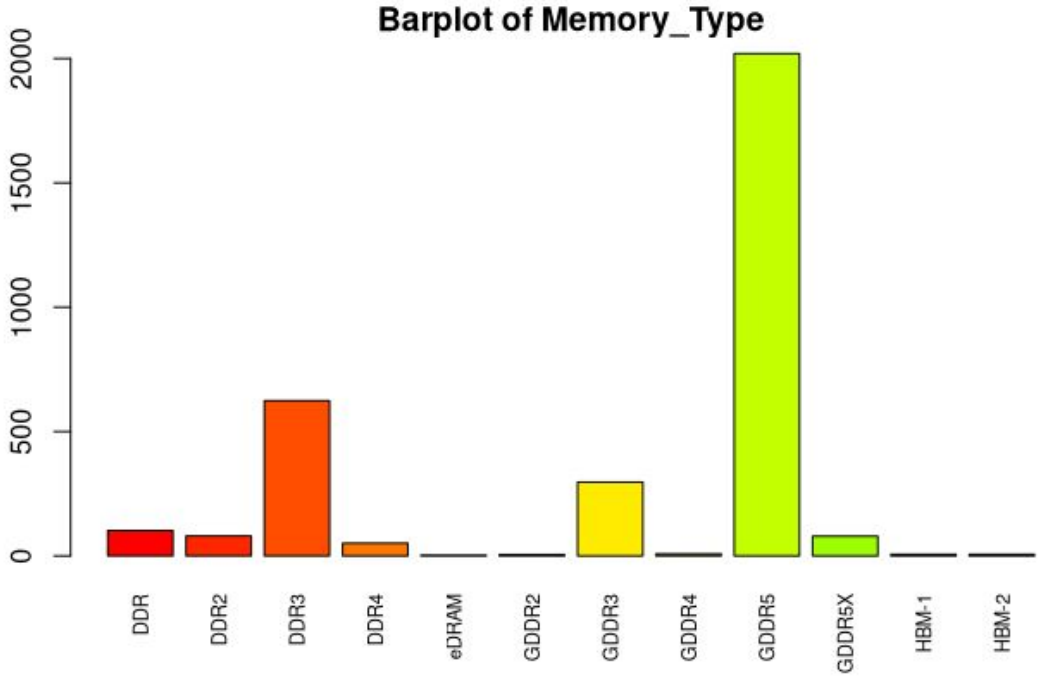


The pie of Direct_X shows us that the majority of the values are 12 and 12.1. Also we can see on the barplot this. The other values are almost uniformly distributed.

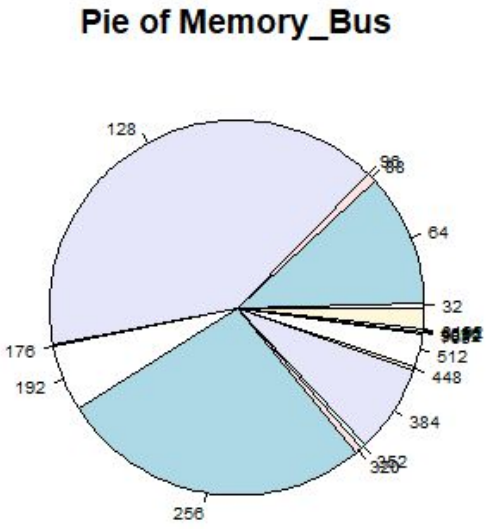


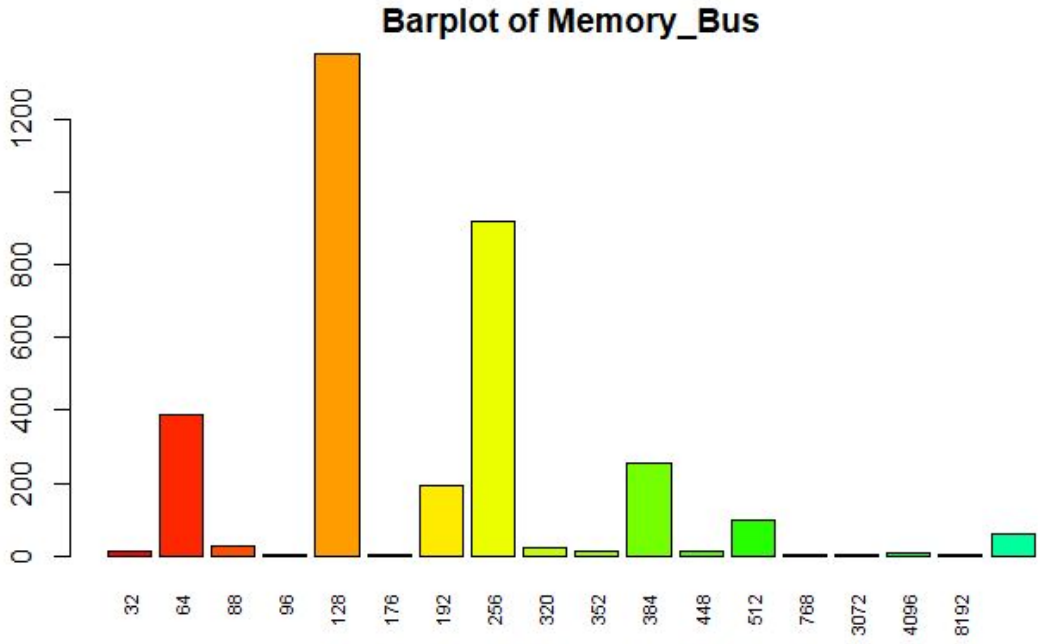
The barplot of Architecture shows that most of the values are in P104-400-A1. Also, there are other kinds of architectures that have a huge amount of values.



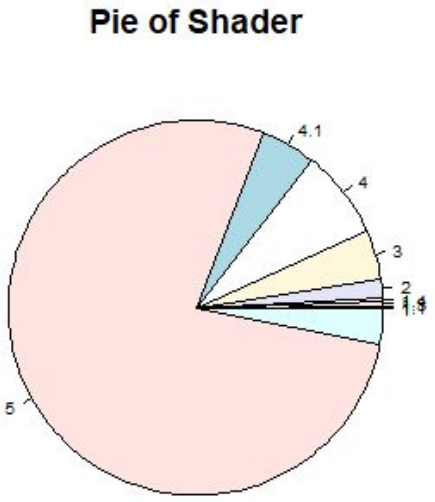


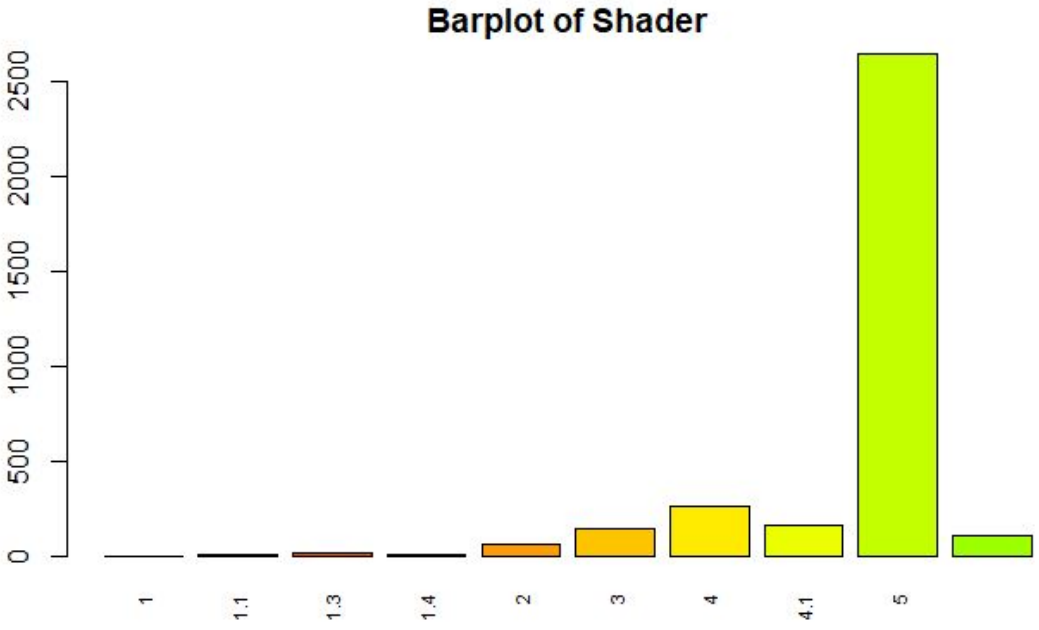
Observing the pie of Memory_type the vast majority of the values are in GDDR5 and, with a little less, DDR3. Also, seeing the barplot we can say that eDRAM, GDDR2, GDDR4, HBM-1 and HBM-2 do not have enough values to be useful.



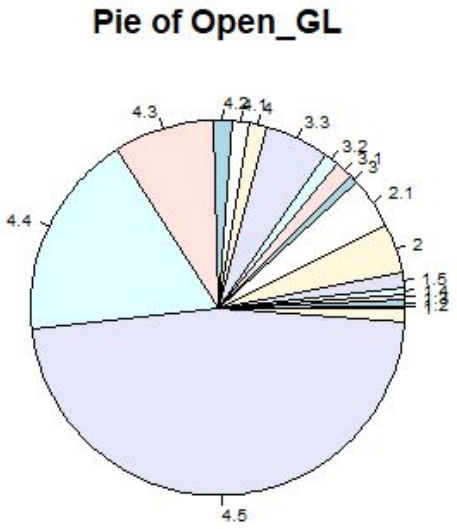


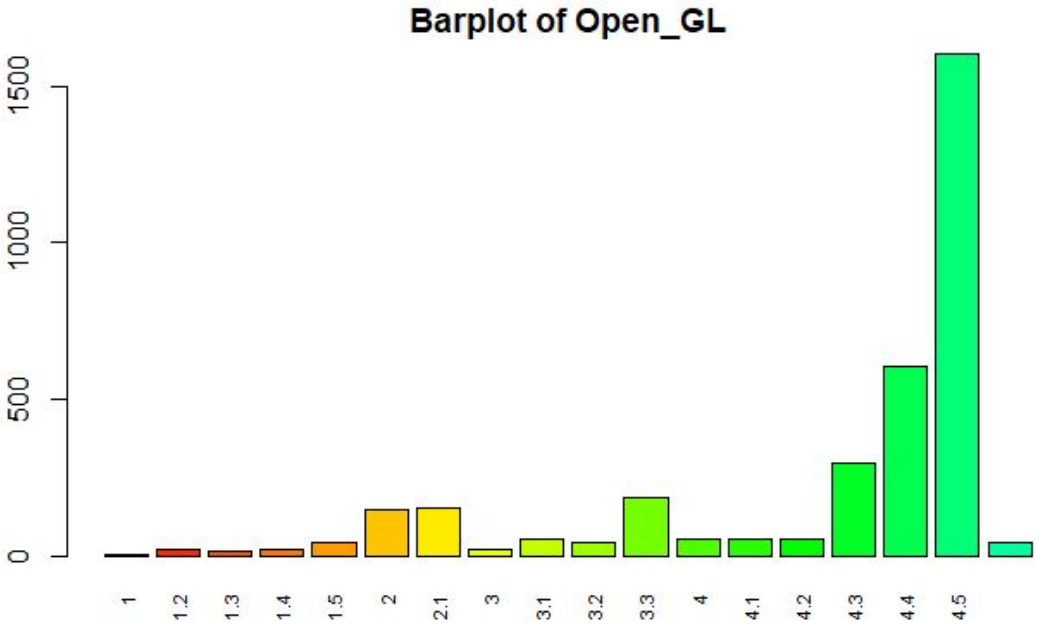
Observing the pie and barplot of Memory_bus we can see that the majority of the values are in “128”, “256” and “64”. There are other values that are almost uniformly distributed except for “192” and “384” that stick out.



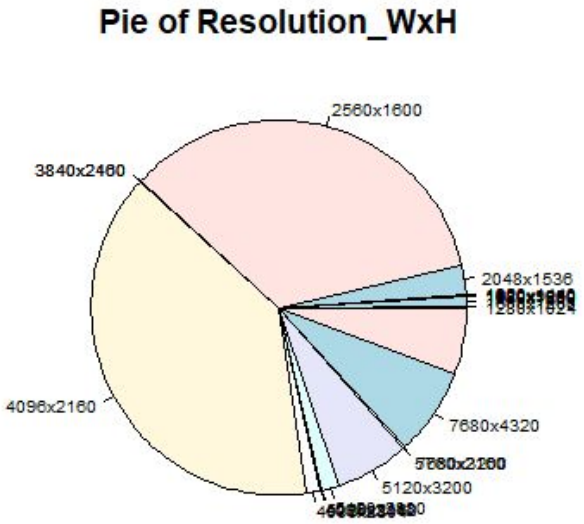


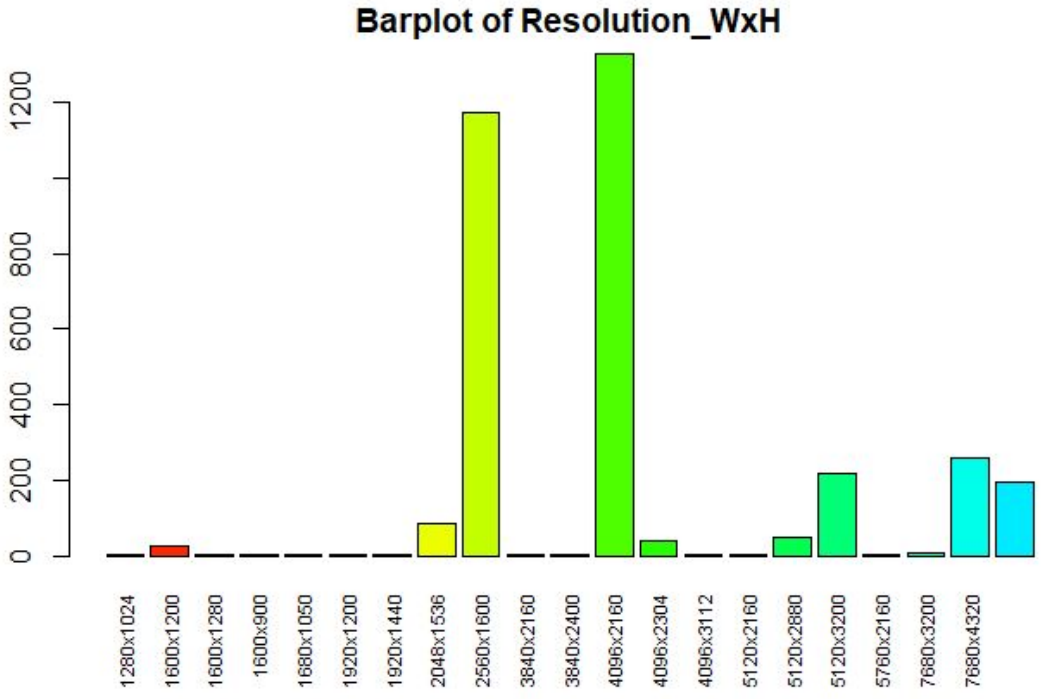
If we observe only the pie of Shader that is the graphic that tells us more information about the variable, we can see that approximately 75% of the values are in 5 and the other 25% are uniformly distributed among the other variables.



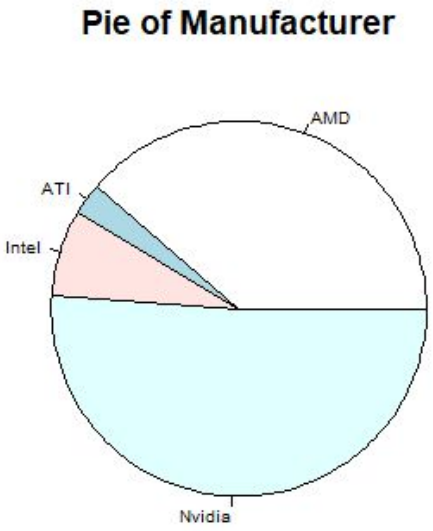


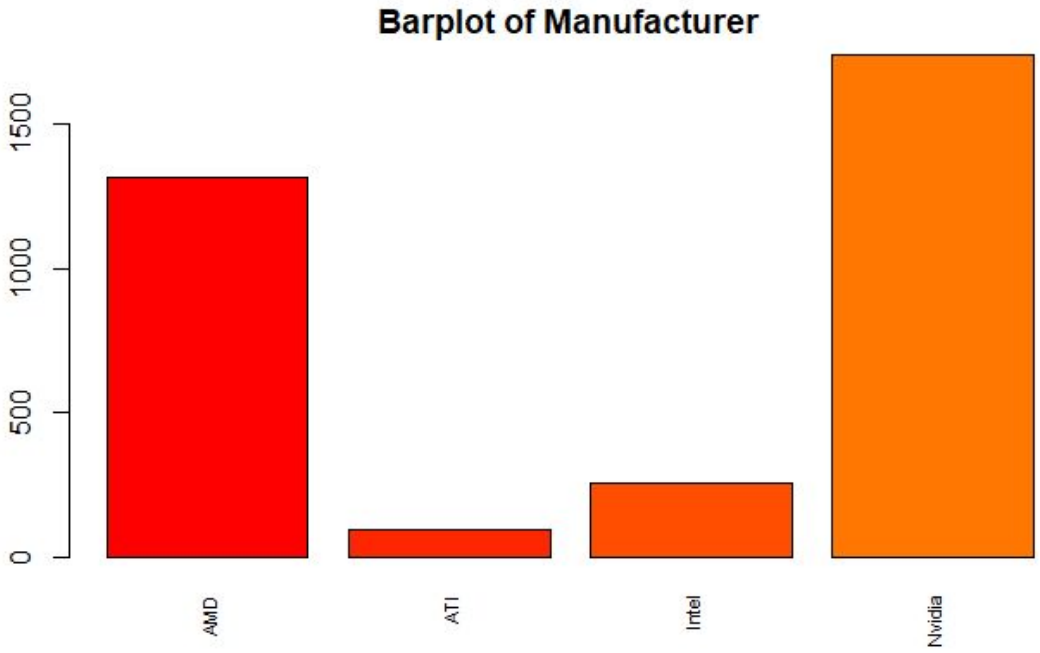
Observing the pie of Open_GL we can see that almost the 50% of the values are in the version 4.5 and 25% are in 4.4 and 4.3 and, for the other 25% of the variables are uniformly distributed.



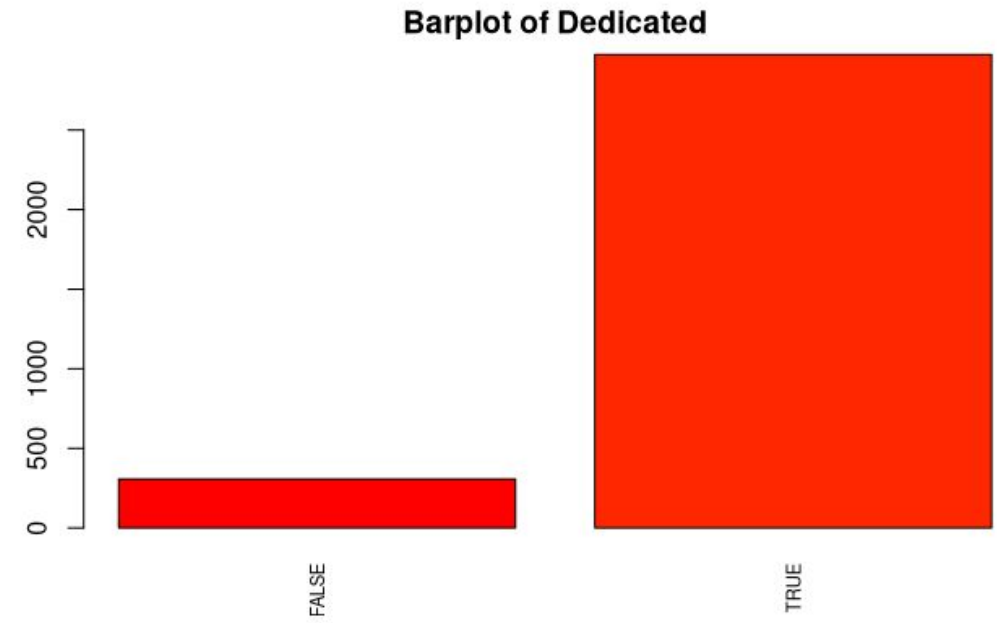


As we can see in the pie chart, 2560x1600 and 4096x2160 are the most common values in the Resolution_WxH column. It can be seen also in the partplot.

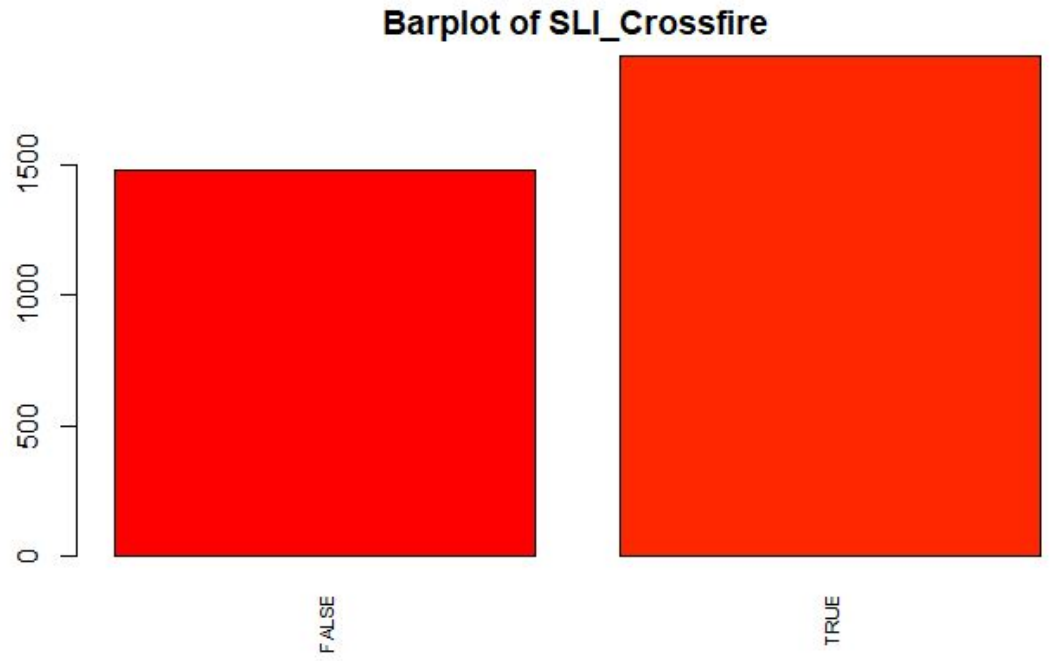




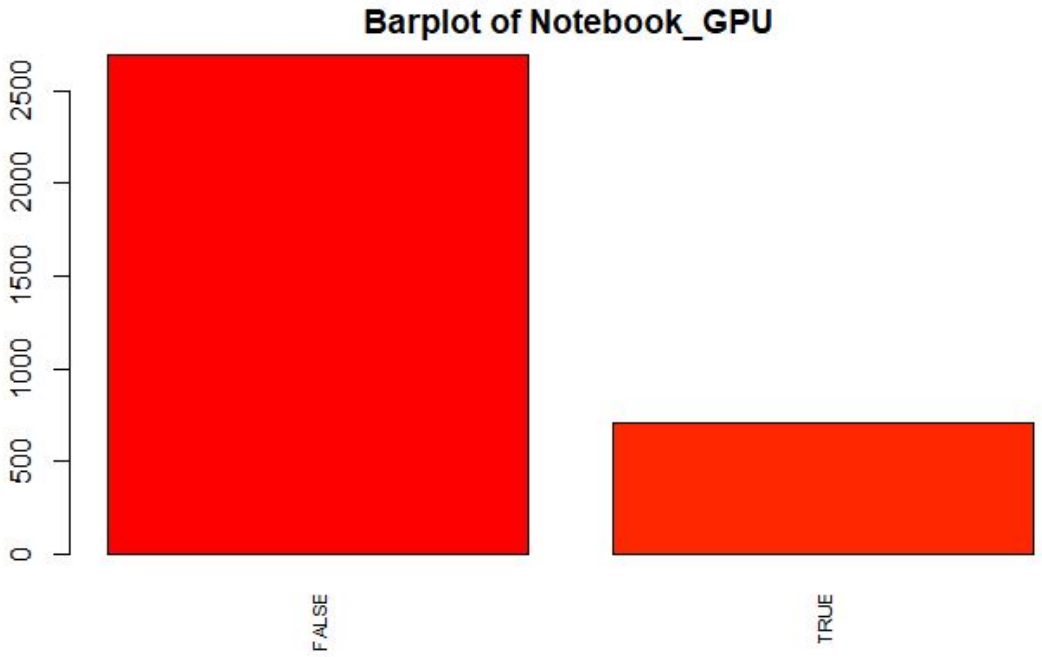
The pie of Manufacturer shows us that the majority of the values are AMD and Nvidia. Also we can see it on the barplot.



The barplot of Dedicated shows that most of the samples have the value Dedicated as True.



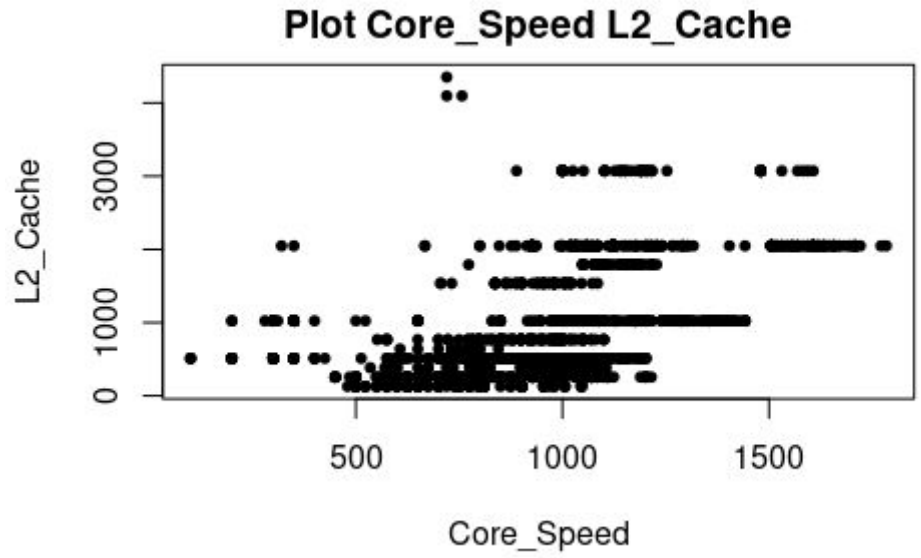
The barplot of SLI_Crossfire shows that the quantity of samples with SLI_Crossfire true is similar to the amount of population with SLI_Crossfire false.



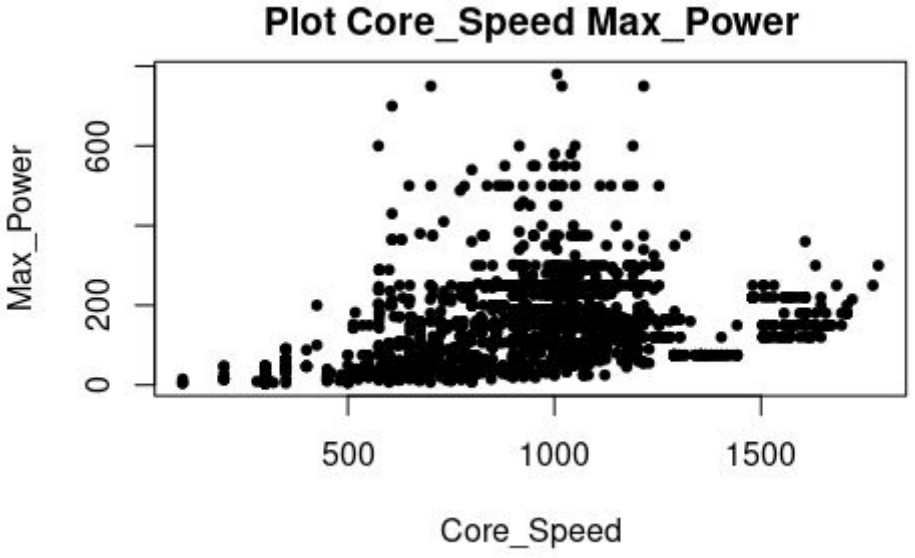
The barplot of Notebook_GPU shows that most of the samples have the value Notebook_GPU as True.

5. Bivariate Descriptive

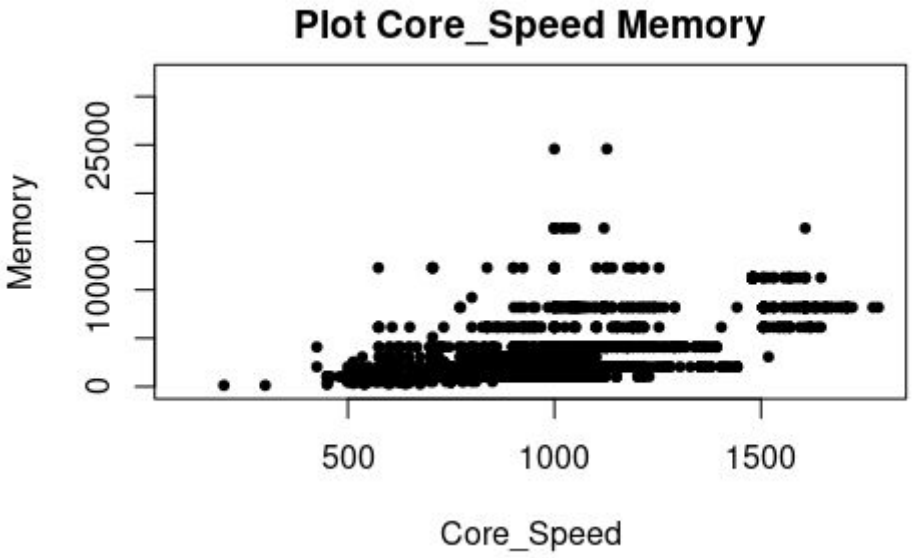
In this section we have posted all the bivariate descriptive statistics. For every pair of numeric variables, we created a plot. For every pair of categorical variables, we created a barplot.



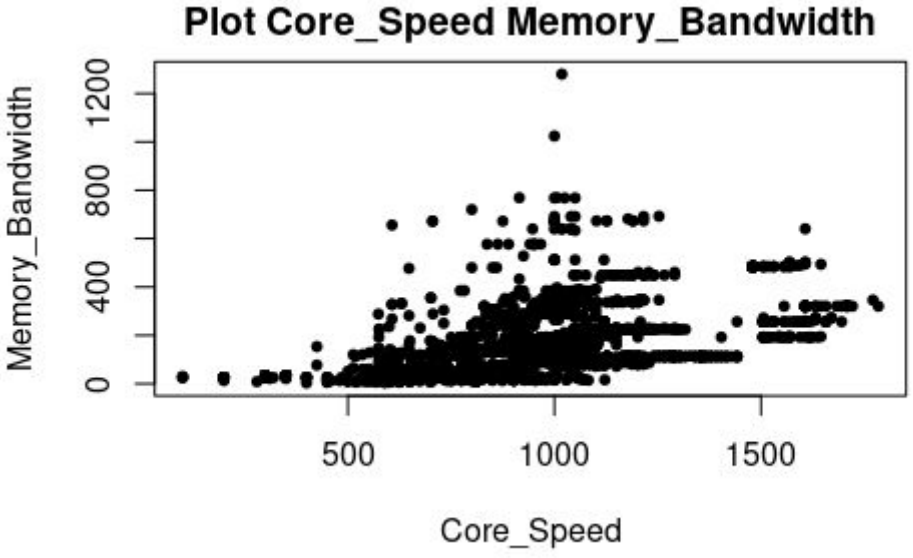
The plot above displays the relationship between the core speed of a GPU and the amount of L2 cache that the GPU has. Most GPUs are situated between 500 MHz and 1500 MHs and below 2000 MB of L2 cache. We can clearly observe that GPUs with lower core speed tend to have a L2 cache with less capacity.



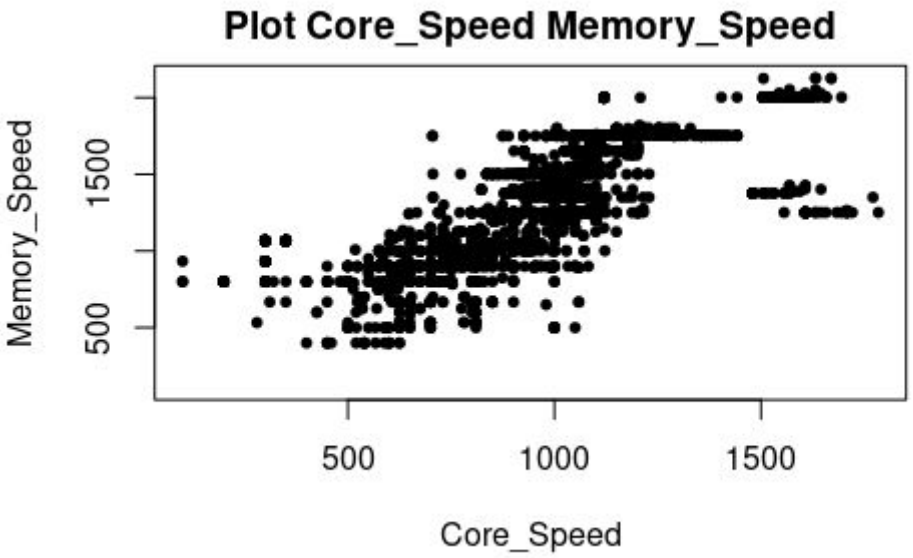
This plot shows the association between core speed and max power variables. In general, we can see that as core speed increases, max power increase as well, although there are some cases in which this tendency do not apply.



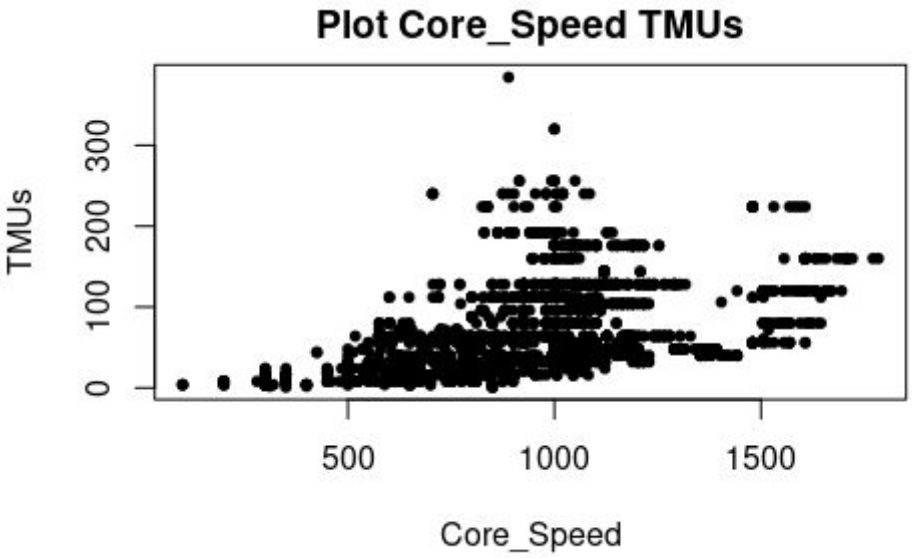
In this plot we can see the relationship between core speed and the amount of graphic memory. In most cases an increase in core speed does not transform into an increment on memory as can be seen with most GPUs between 500MHz and 1000MHz.



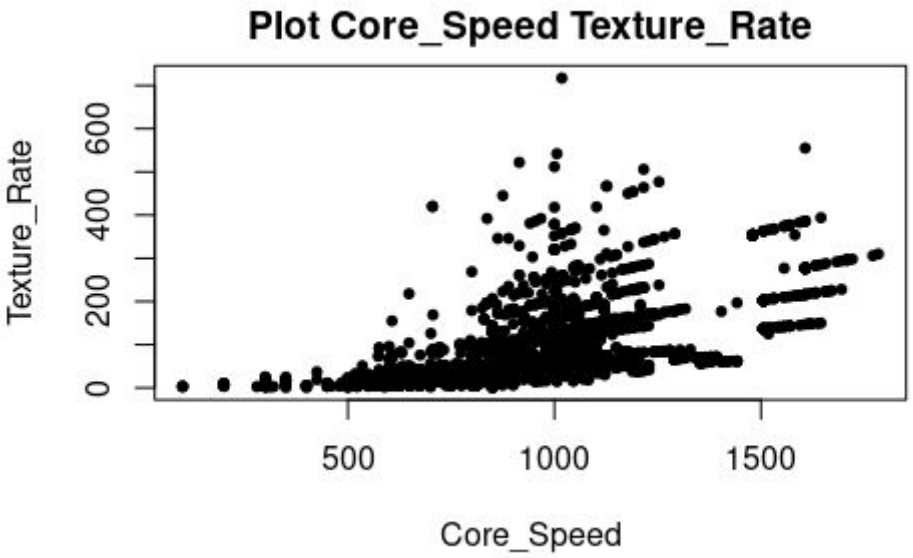
The plot above displays the relationship between core speed and memory bandwidth. We can observe increments in core speed tend to be followed by increments in memory bandwidth.



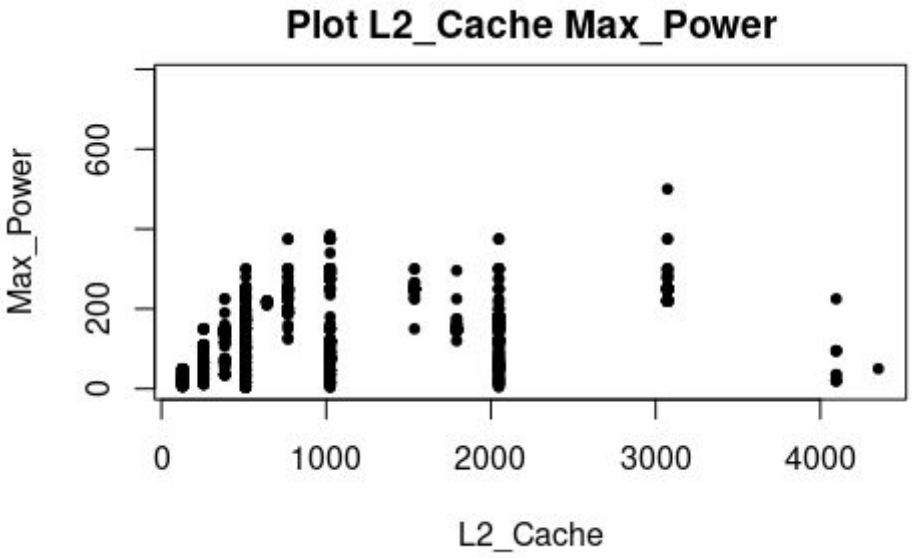
In this plot we can see the relationship between core speed and memory speed. There is a clear tendency where increments in core speed result in increments on memory speed, as a faster GPU tends to require a faster memory to be worth it.



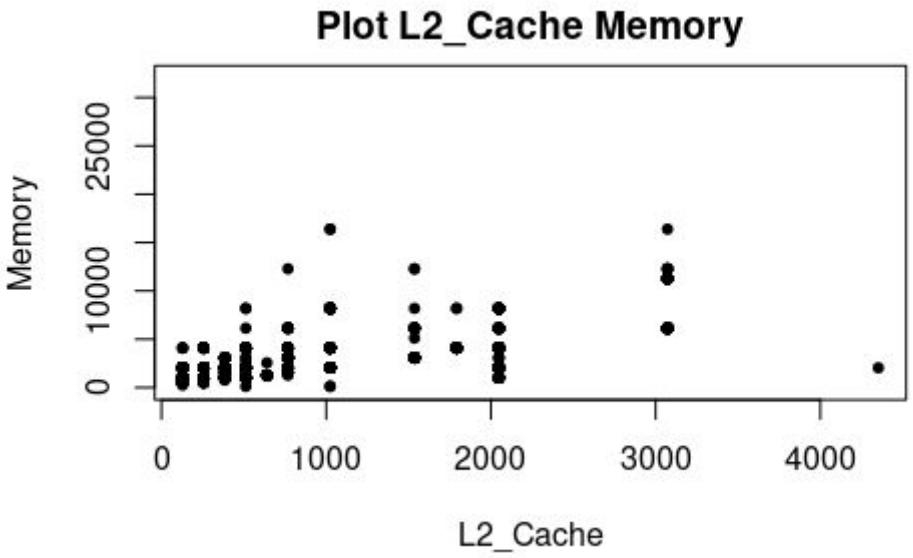
This plot displays the association between core speed and TMUs. In this case most GPUs are situated between 500 MHz and 1500 MHz and below 200 TMUs. We can observe that GPUs tend to have more TMUs when they have more core speed, as faster GPUs need a higher amount of TMUs to really exploit the performance leap.



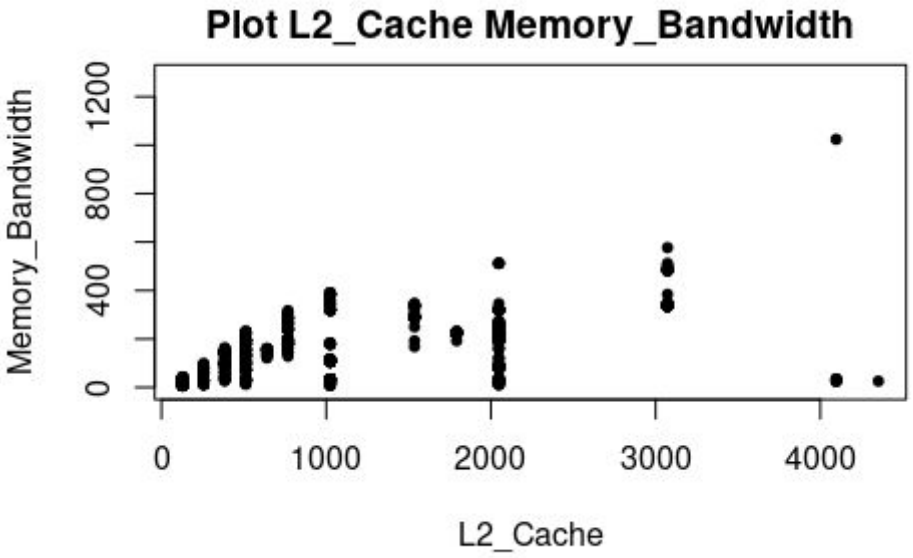
In this plot we can see the relationship between core speed and texture rate. In most of the cases we can observe a tendency where GPUs with more core speed tend to have more texture rate, as the increment in performance lets the GPU output a higher texture rate.



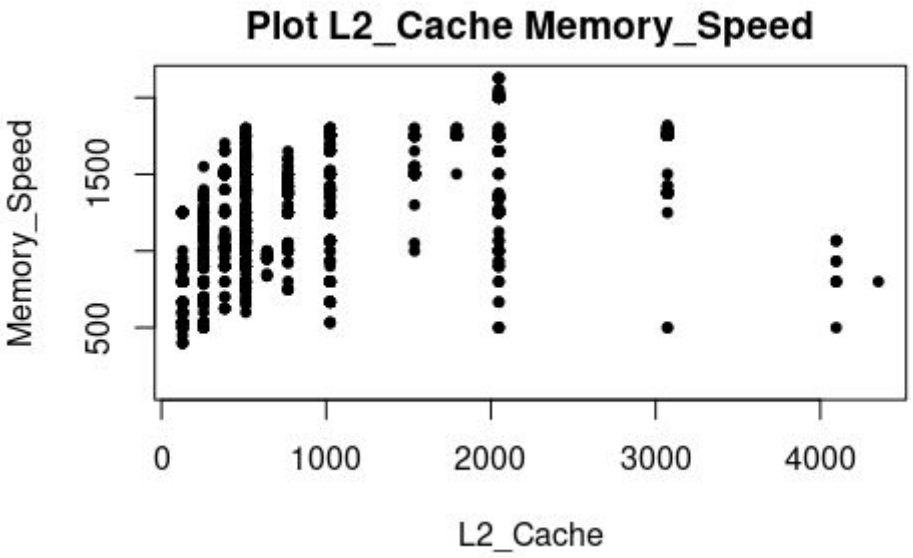
This plot shows the relationship between the amount of L2 cache of a GPU and its maximum power. In this case we can observe that a higher amount of L2 cache does not always imply more power consumption.



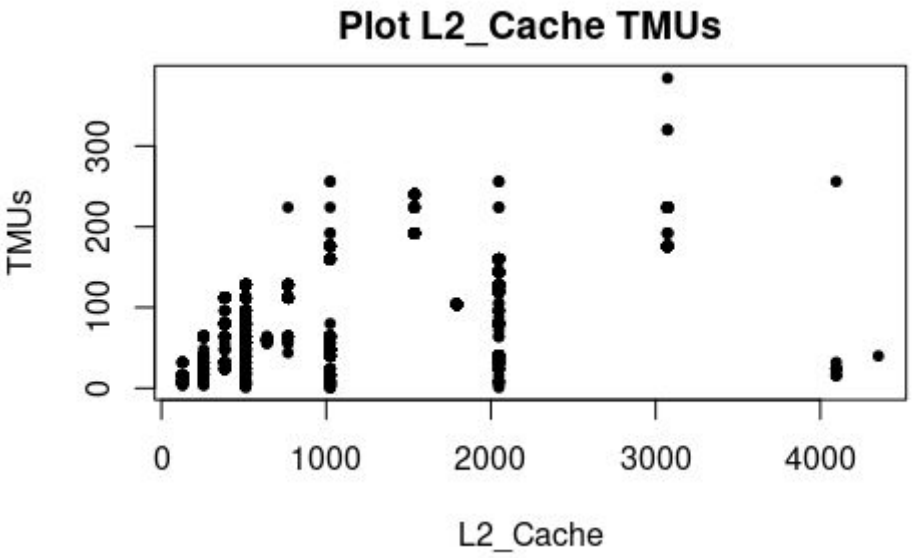
This plot displays the association between L2 cache and the amount of memory of the GPU. We can see a tendency in which GPUs with more L2 cache tend to have more memory available.



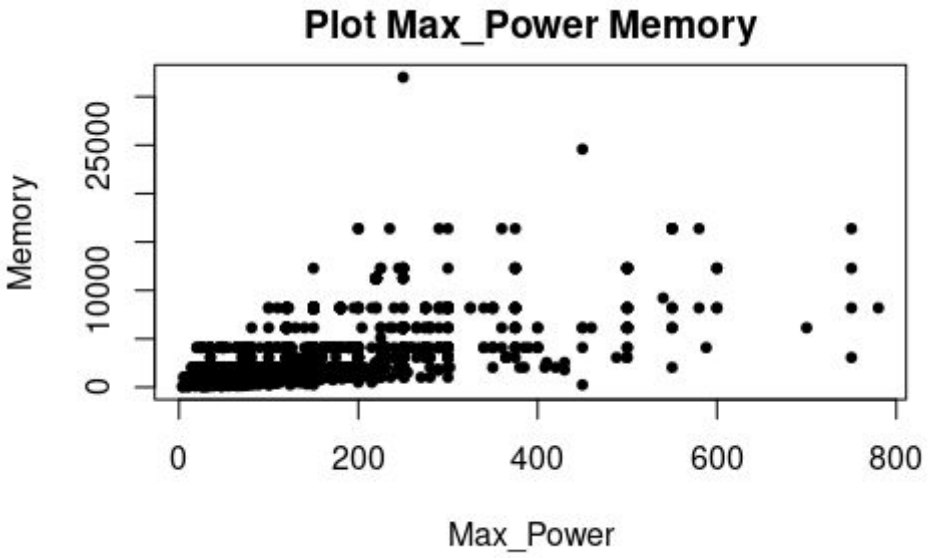
In this plot we can see the relation between L2 cache and memory bandwidth. There is a clear tendency in which a higher L2 cache tends to imply more memory bandwidth, probably because a higher bandwidth can help to achieve greater performance with more L2 cache.



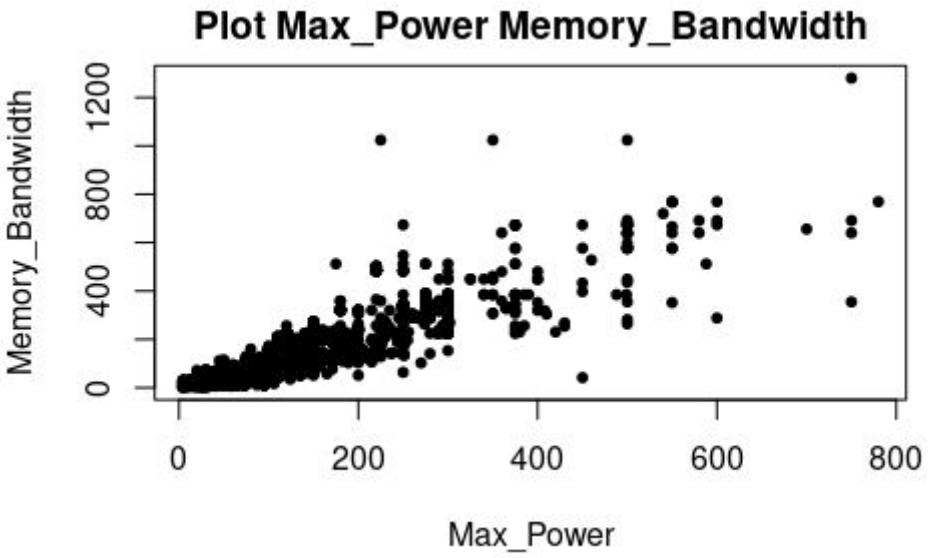
This plot shows the relationship between the amount of L2 cache and the speed of the GPU's memory. There does not seem to exist any relationship between both variables as higher values of L2 cache does not have higher values of memory speed, and the GPUs which have greatest memory speed have about 2000KB of L2 cache, which is approximately a middle value of L2 cache.



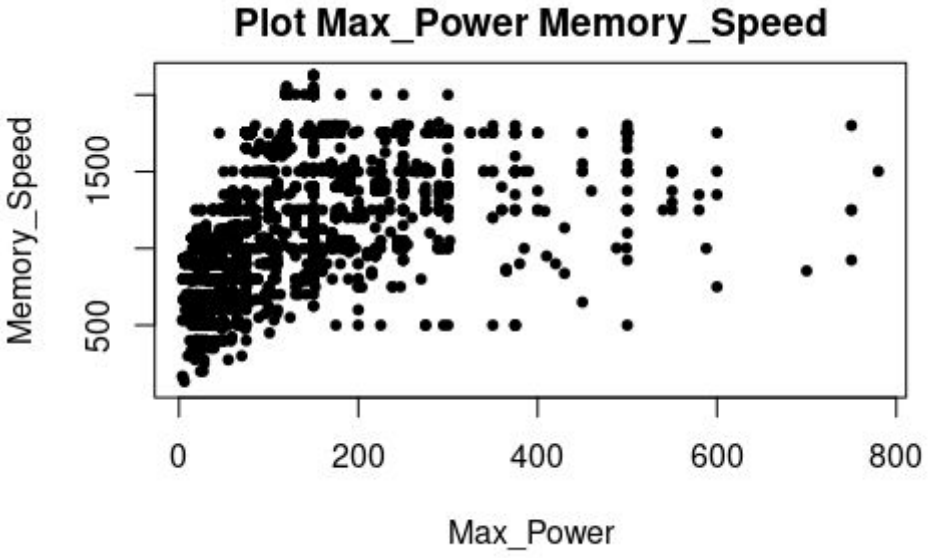
In this plot we can observe the association between the amount of L2 cache of a GPU and its TMUs. There seems to be a tendency in which GPUs with higher amounts of L2 cache have more TMUs.



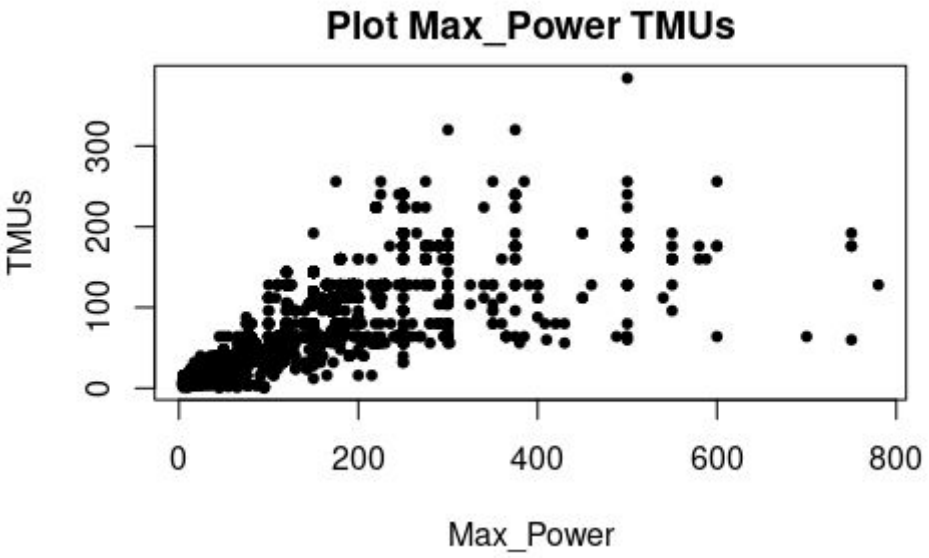
This plot displays the relationship between GPUs maximum power and its amount of memory. We can see that most GPUs are situated below 400W of max power consumption and below 10000 MB of memory. In addition, it do not seem to exist any association between both variables, as higher power consumption do not imply higher memory.



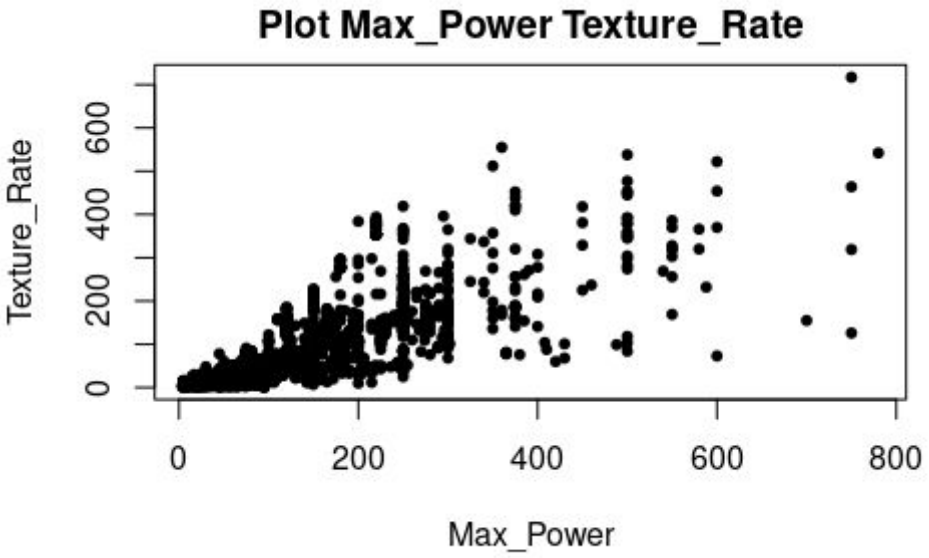
This plot displays the relationship between maximum power consumption and memory bandwidth. As higher bandwidth usually requires more power consumption, we can observe a clear tendency in which GPUs with higher max power tend to have a higher memory bandwidth.



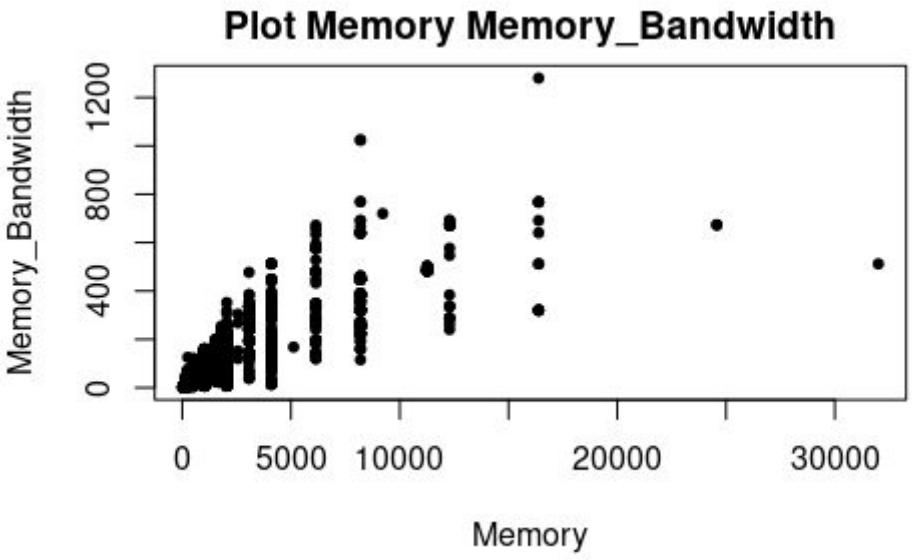
This plot displays the relationship between maximum power consumption of a GPU and its memory speed. We can see that most GPUs are situated below 200 W of maximum power and below 2000 of memory speed. It does not seem to exist any relationship between both variables as all range of max power have GPUs with higher and lower memory speed.



In this plot we can see the relationship between maximum power consumption and TMUs. It can be observed that GPUs with higher amounts of max power have a clear tendency of having more TMUs, as more TMUs usually require a major power source.

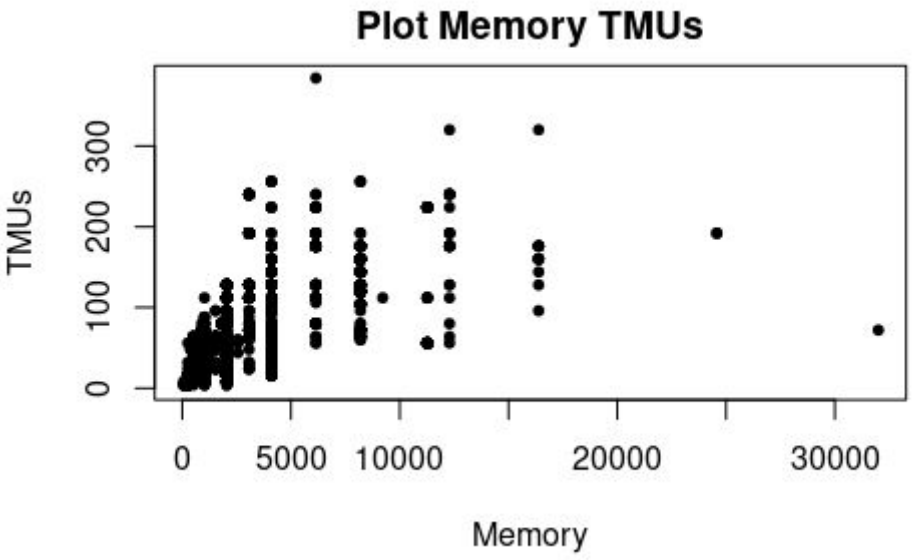


This plot displays the relationship between GPU power consumption and its texture rate. We can clearly see that GPUs with higher max power tend to have a higher texture rate as well.

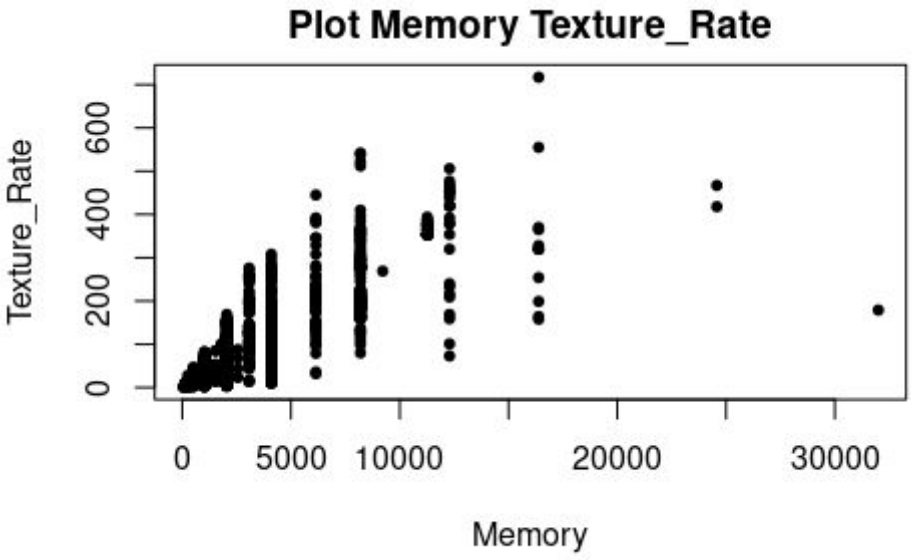


This plot displays the association between GPU's amounts of memory and memory bandwidth. It can clearly be seen that there is a huge dependency between both variables as higher memory tends to imply a higher memory bandwidth.

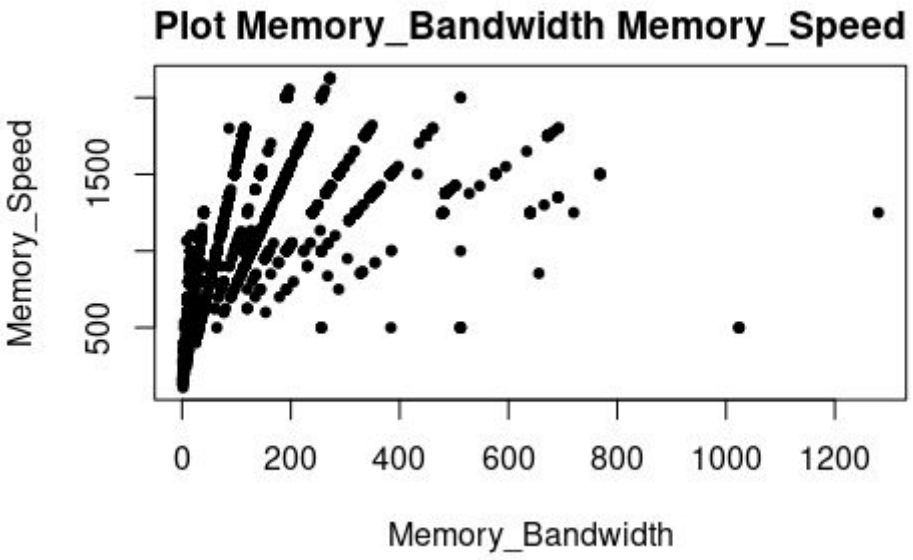
In this plot we can see the relationship between memory and memory speed. It can be observed that most of the GPUs are situated below 5000 MB of memory and below 1500 of memory speed. There is a tendency in which GPUs with more memory available usually have more memory speed, as higher memory speed is needed in order to exploit the increased memory.



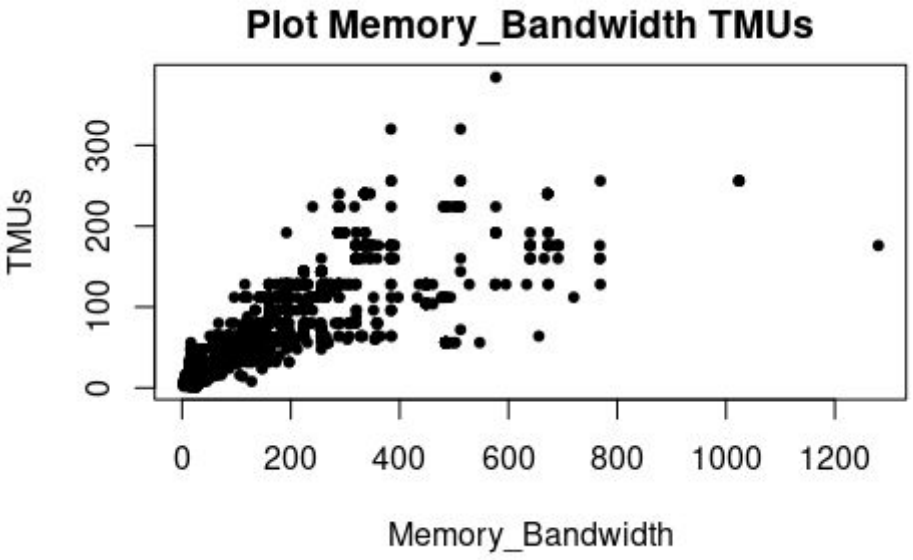
This plot displays the relationship between GPUs memory and its number of TMUs. We can see that most of the GPUs are situated below 5000 MB of memory and below 150 TMUs. There seems to be a tendency in which GPUs with more memory have a higher number of TMUs.



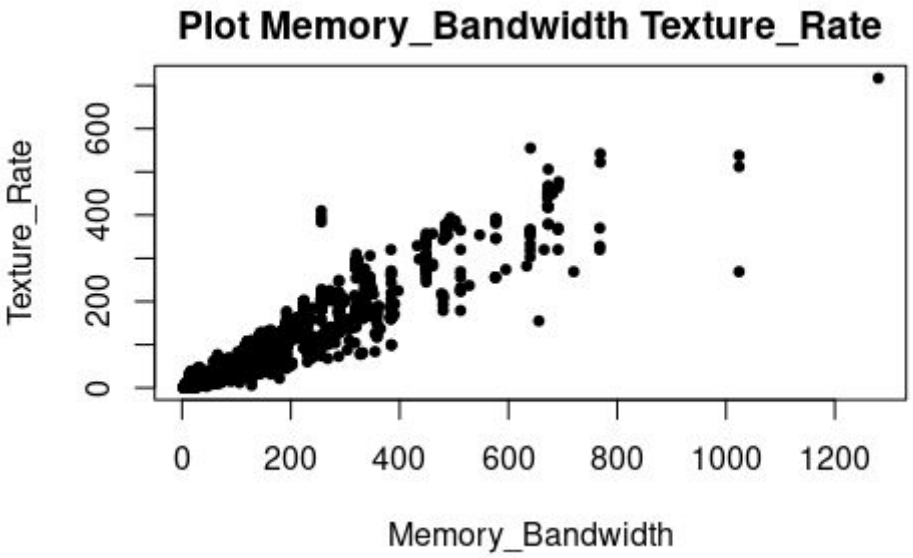
This plot shows the association between memory and texture rate. We can see that most GPUs are situated below 5000 MB of memory and below 200 of texture rate. There is a clear tendency in which GPU with more memory have more texture rate as more memory let the GPU perform better and produce a higher texture rate.



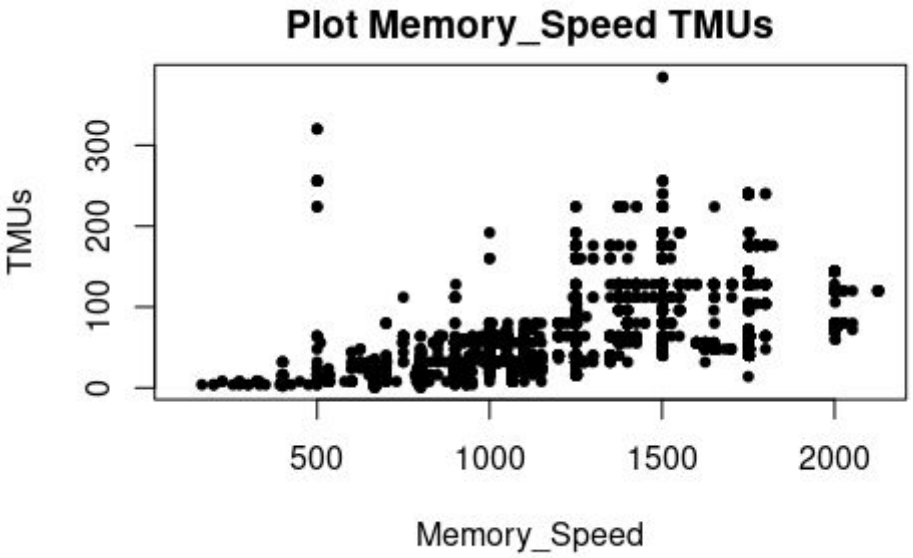
This plot displays the relationship between memory bandwidth and memory speed. Most of the GPUs are situated below 200 of memory bandwidth and below 1000 of memory speed. There does not seem to exist any tendency between both variables as maximum memory speed is located within GPU of approximately 200 of memory bandwidth.



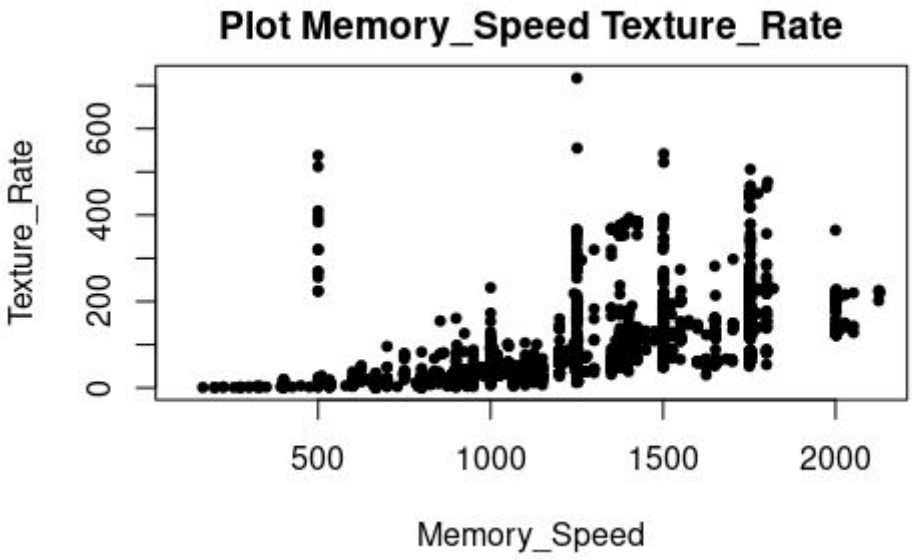
In this plot we can see the relationship between memory bandwidth and the number of TMUs. It is clear that most of the GPUs are located below 300 of memory bandwidth and below 150 TMUs. There is a clear tendency in which GPUs with more memory bandwidth tend to have more TMUs, as the increase in memory bandwidth would not be really effective if the GPU did not have enough TMUs available.



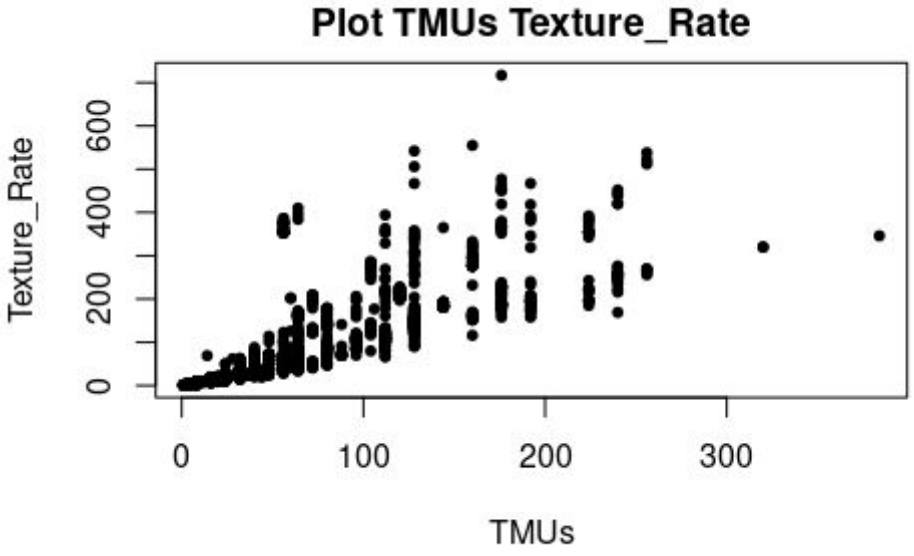
This plot displays the association between memory bandwidth and texture rate. Most of the GPUs are located below 400 of memory bandwidth and below 300 of texture rate. In this case there is a clear relationship in which GPUs with more memory tend to have more texture rate.



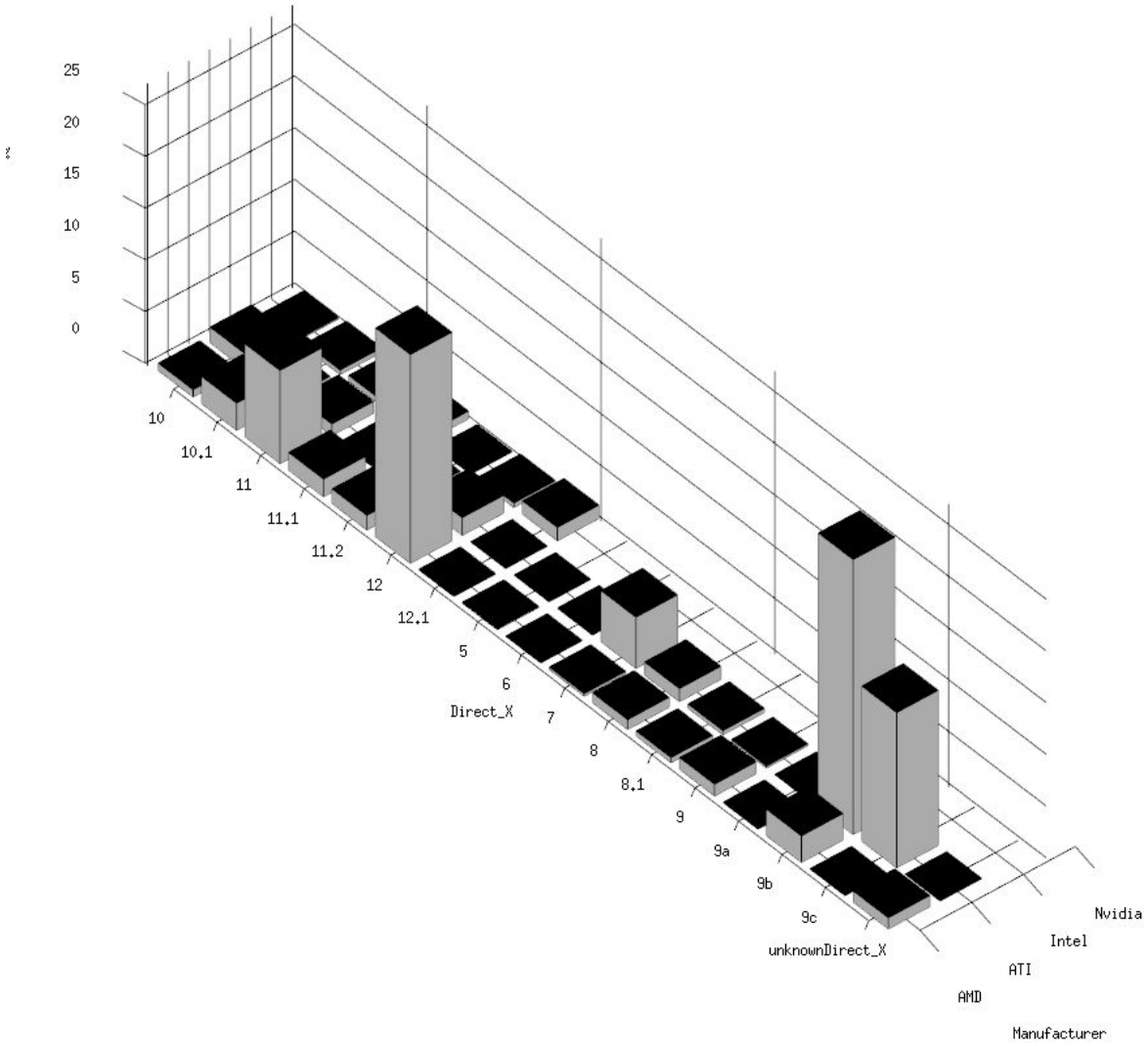
In this plot we can see the relationship between memory speed and TMUs. In most cases, GPUs with an increased memory speed tend to have an increased number of TMUs, as they need those TMUs to handle the increment in memory speed.



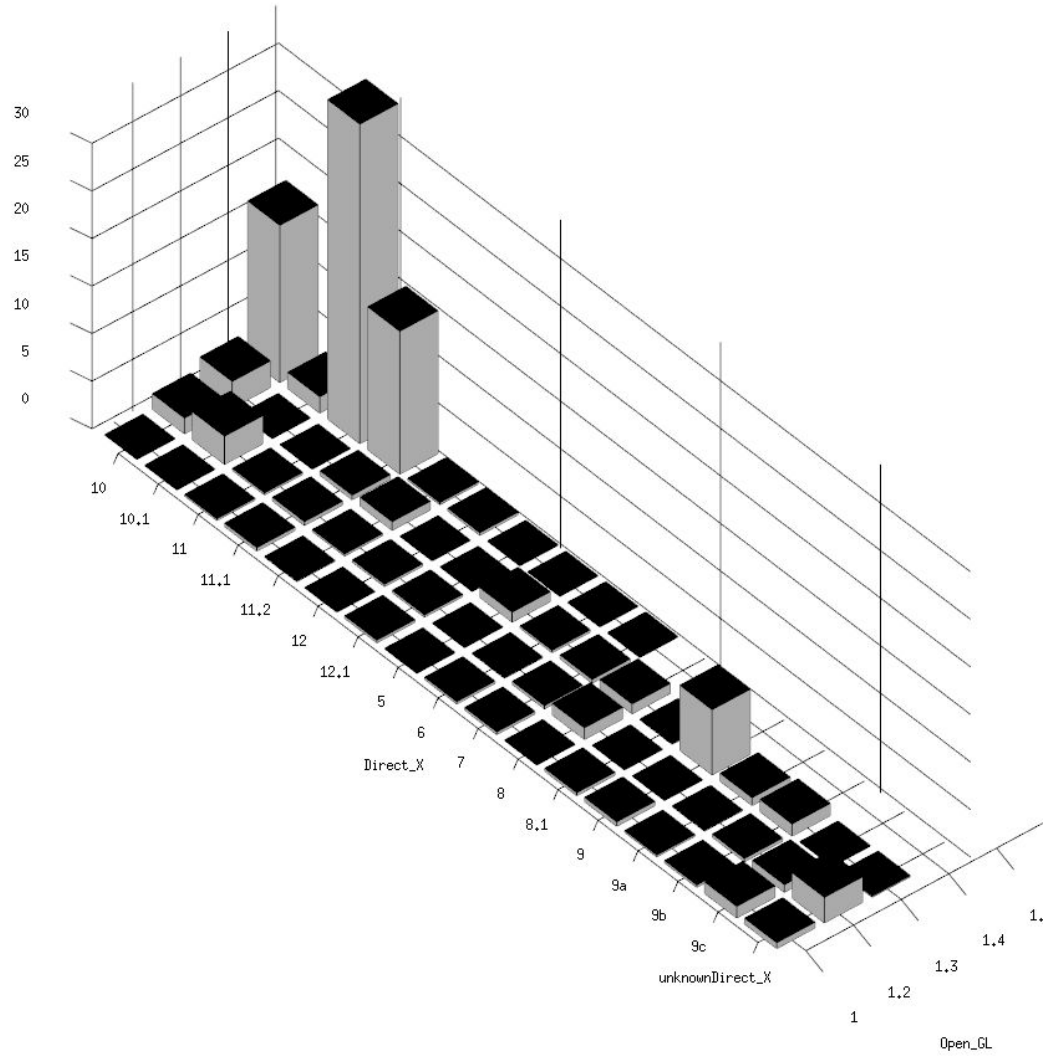
This plot shows the relationship between memory speed and texture rate. Most of the GPUs have less than 300 of texture rate. There is a clear tendency between both variables as GPUs with more memory speed tend to have more texture rate, because the increase in memory speed let the GPU perform better and therefore increase its texture rate.



This plot displays the association between the number of TMUs available in a GPU and its texture rate. Most of the GPUs are located below 150 TMUs and below 400 of texture rate. There exist a clear relationship between both variables in which GPUs with more TMUs tend to have a greater performance and increment its texture rate.

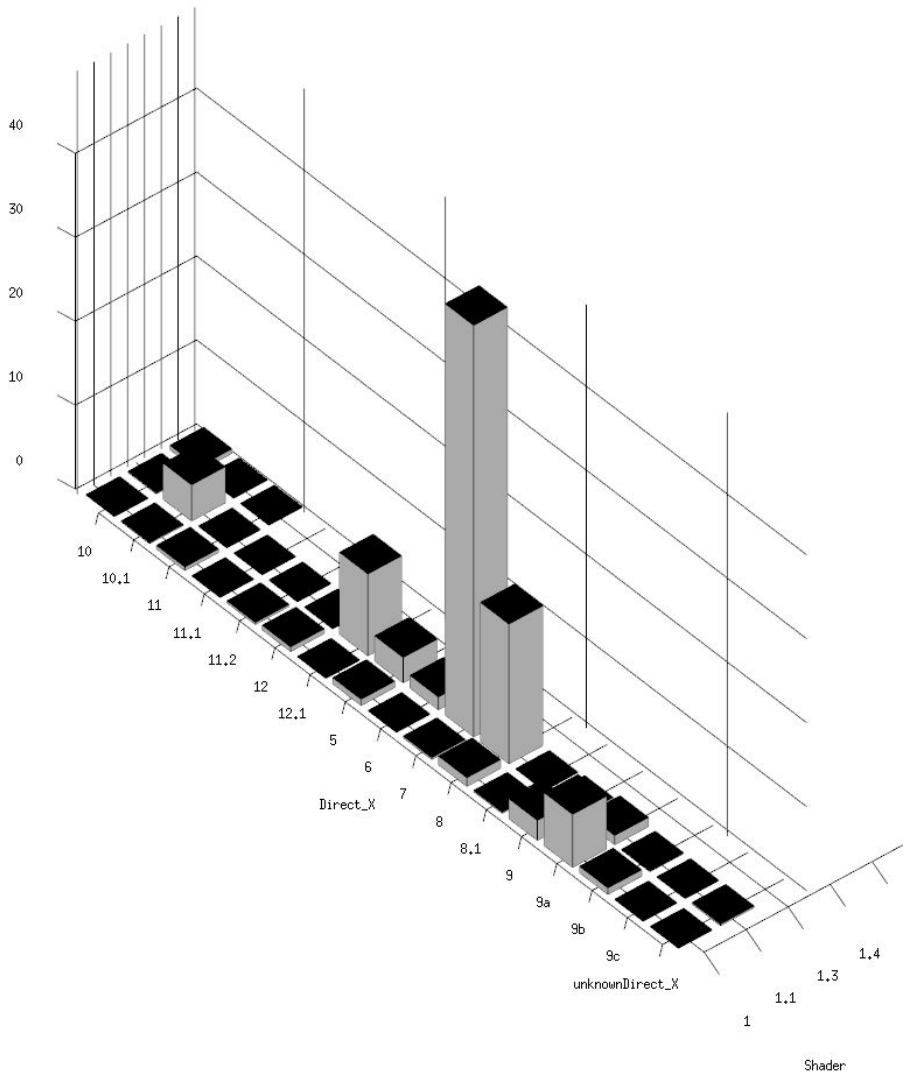


This barplot shows us the relationship between the variable Manufacturer and Direct_X. As we can see, in our dataset almost all of the GPUs from ATI have DirectX number 9b or 9c. Furthermore, almost all of the AMD GPUs have DIrectX number 12 or 11.

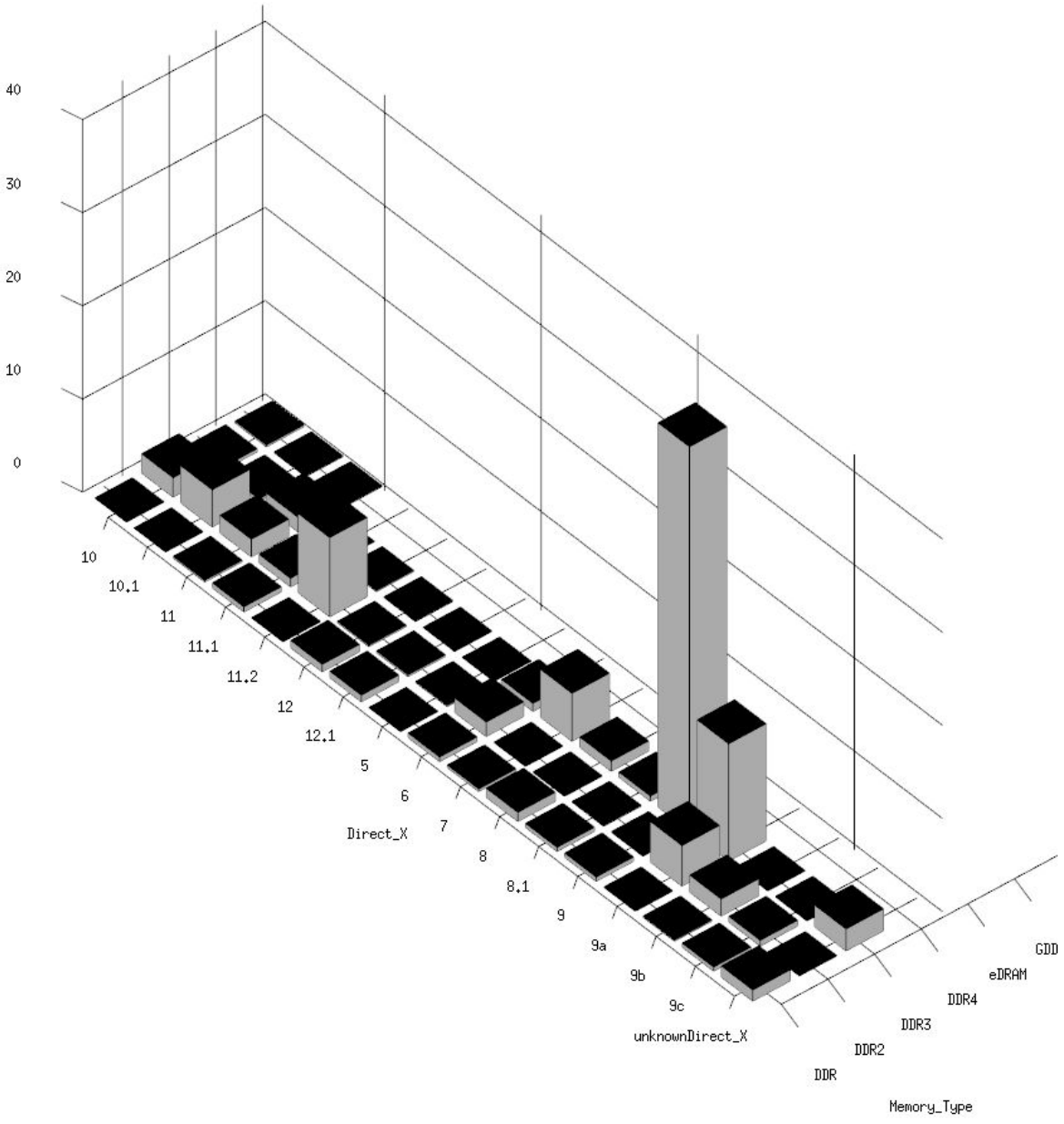


In our dataset we can find that most of the GPUs with OpenGL 1.3 have a Direct X version 9. As we can see, most of the GPUs with OpenGL 1.4 has a Direct X version 10.1. Also, we can find in our dataset a lot of GPUs with OpenGL 1.4 and DirectX 11.1, and GPUs with OpenGL 1.4 and Direct X 10.

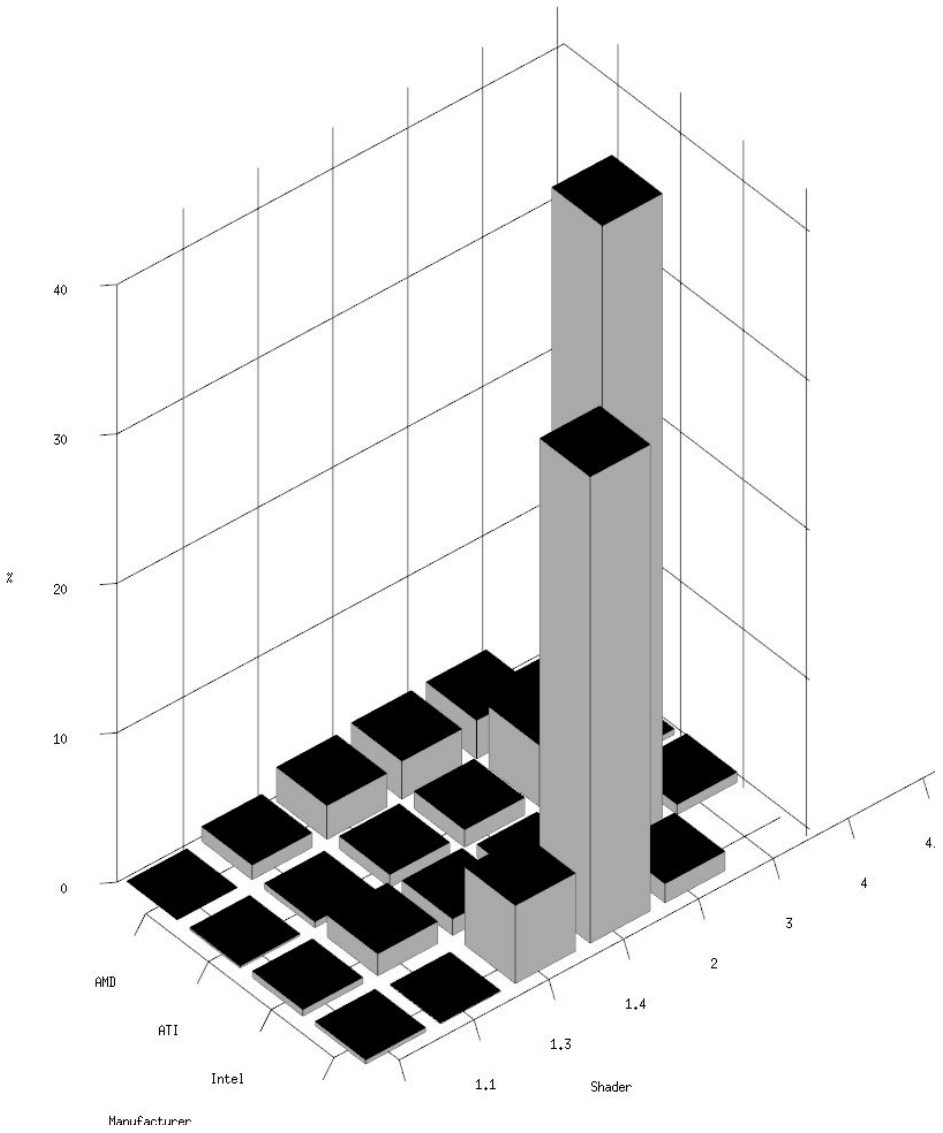
This barplot explains the relation between shader and Direct X. We can see that almost all of the Shader with version 1.1 are relationed with Direct X 7, 8 and 12.1. Respecting the Shader version 1, we can see that all GPUs have Direct X 9a.



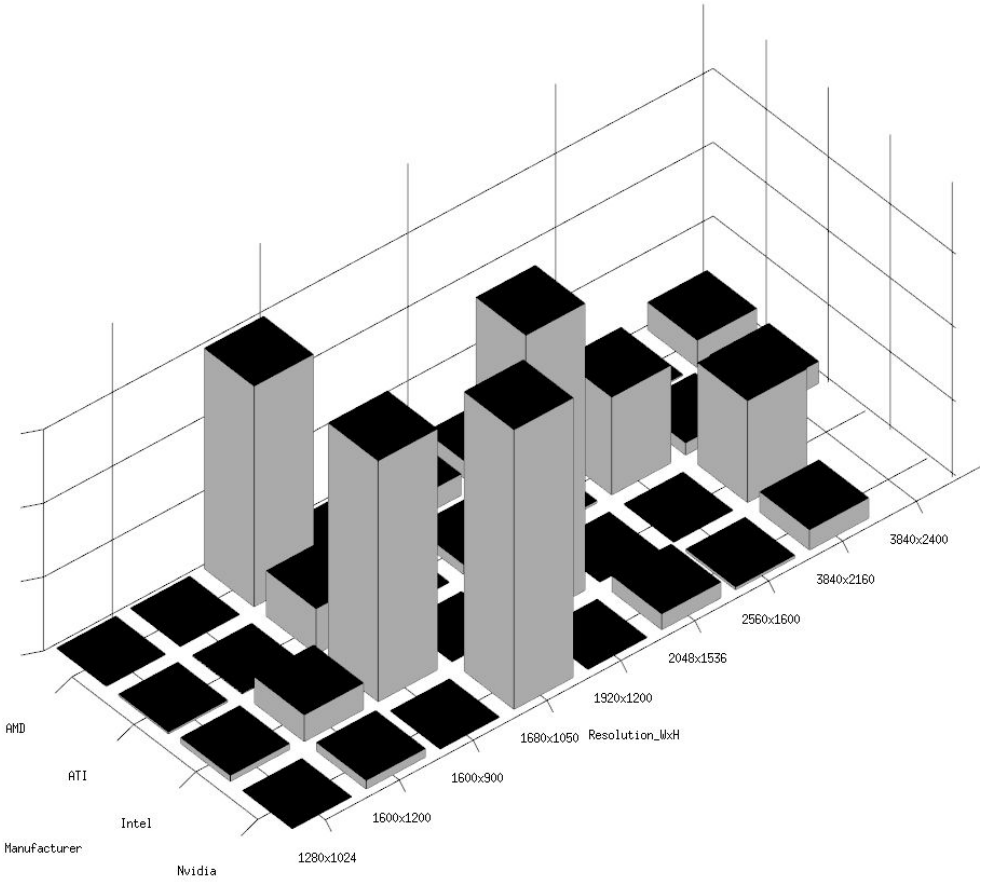
In our dataset GPUs with a Memory type DDR3 has Direct X 9 or Direct X 8. The GPUs with a Memory type DDR2 have Direct X 11.2 or Direct X 10.1. We can find a few GPUs with Memory type DDR, but all GPUs with DDR have an unknown version of Direct X.


40
30
20
10
0
10
10.1
11
11.1
11.2
12
12.1
5
6
7
8
8.1
9
9a
9b
9c
unknownDirect_X
Direct_X
DDR
DDR2
DDR3
DDR4
eDRAM
GDD
Memory_Type

As we can note the GPUs with Intel have a Shader version 2. Moreover, NVIDIA GPUs have a Shader version 1.4. In relation to AMD GPUs, highlight the Shader version 1.4, 2 and 3. Furthermore, ATI GPUs have a Shader version 2.

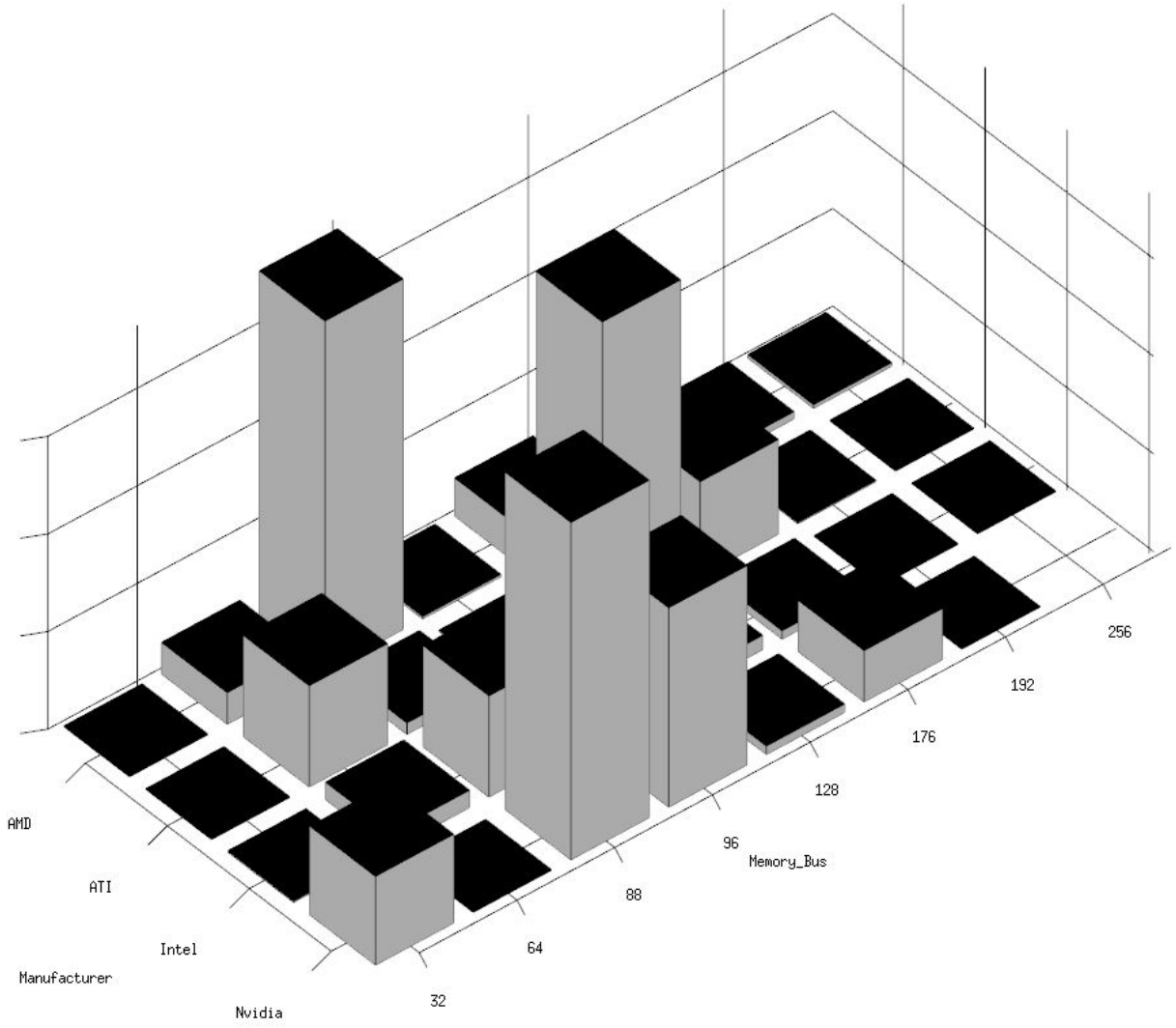


We can focus on the dependence between GPUs manufacturers and the resolution of the GPUs. We can say that almost all Nvidia GPUs from our dataset support a resolution of 1680x1050. Moreover, nearly all Intel GPUs from our dataset support 1600x900 resolution, the same as ATI GPUS. Part of our dataset is composed by ATI GPUs with 2560x1600 resolution.

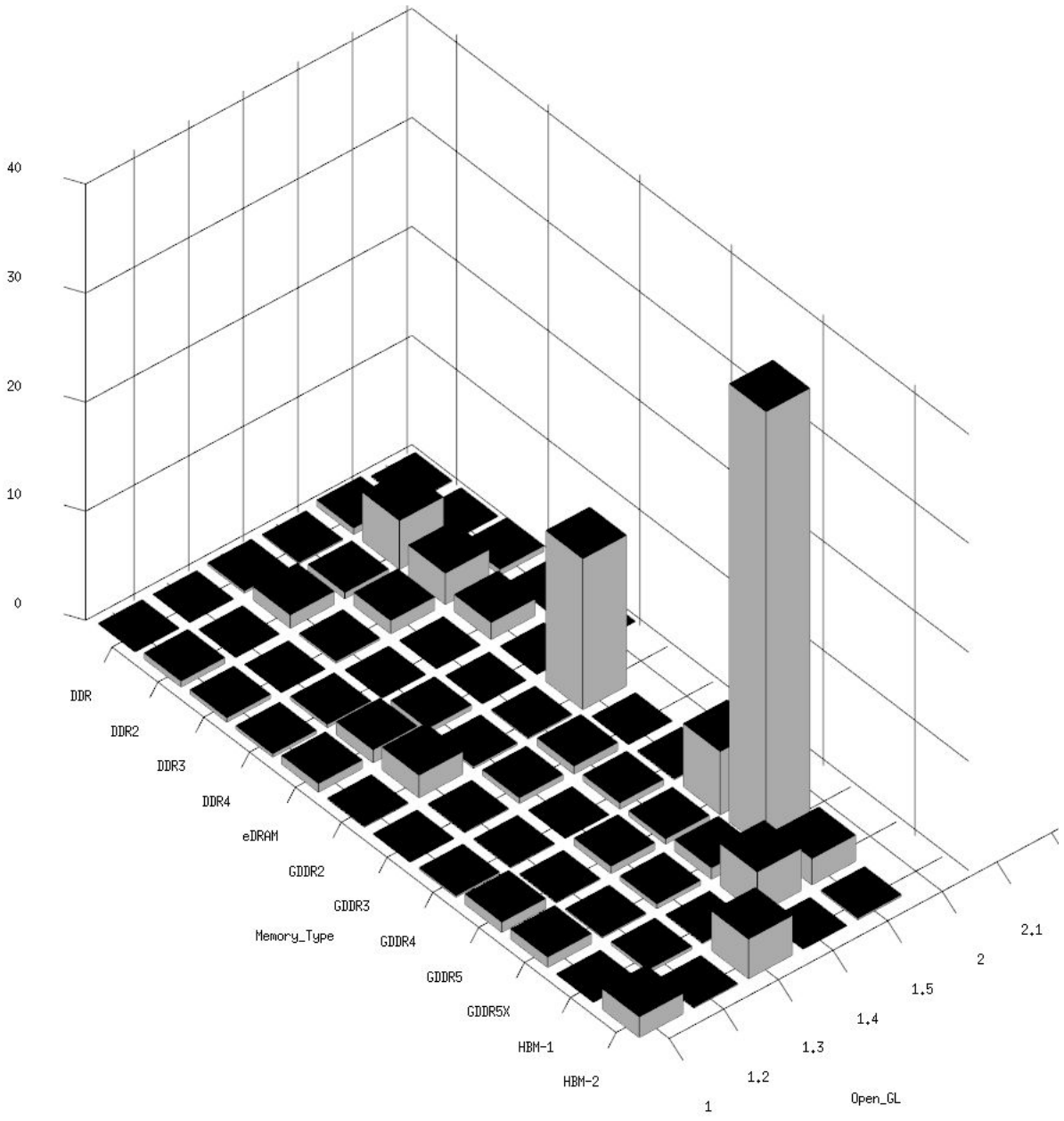


If we pay attention to the relation between GPUs manufacturers and memory bus, we can see that all Nvidia GPUs have a memory bus with 88 KB or 56 KB. In addition, our dataset is full of Intel GPUs

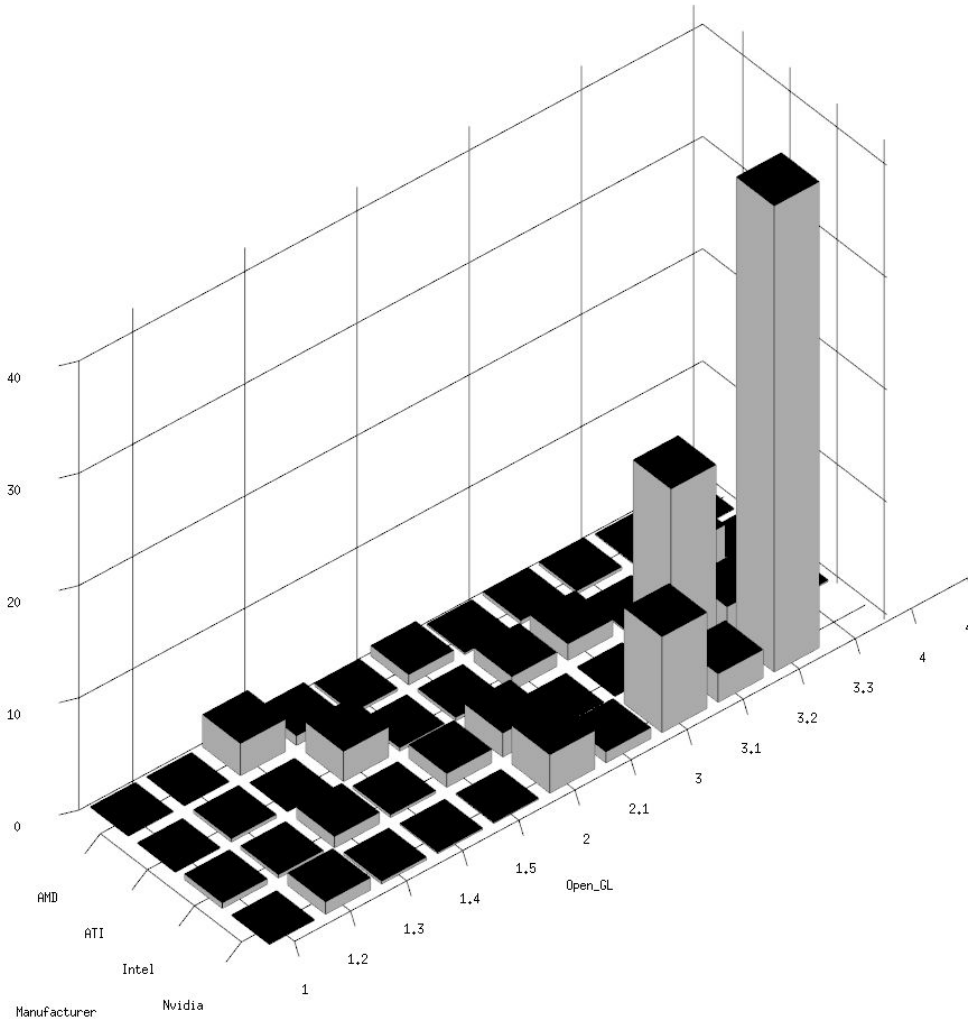
with a memory bus of 58KB. ATI GPUs have a memory bus of 128Kb and AMD GPUs have a memory bus of 88KB.



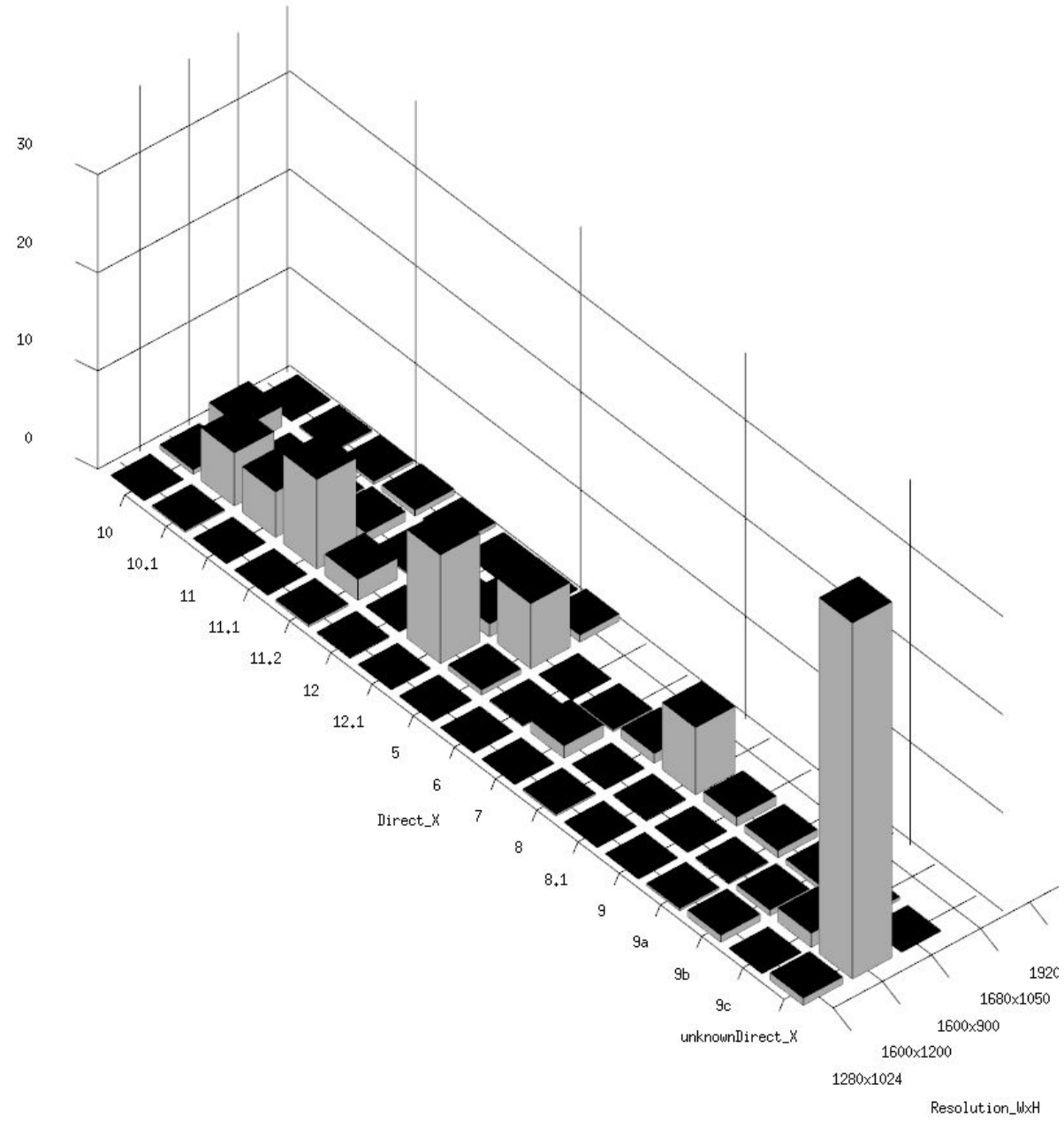
In our dataset we can find that all GPUs with OpenGL 1.5 have a memory of GDDR5X or a memory of GDDR2. Almost all of the GPUs with OpenGL 1.4 have a memory of HBM-1 and GPUs with OpenGL 1.3 have a memory of HBM-2.



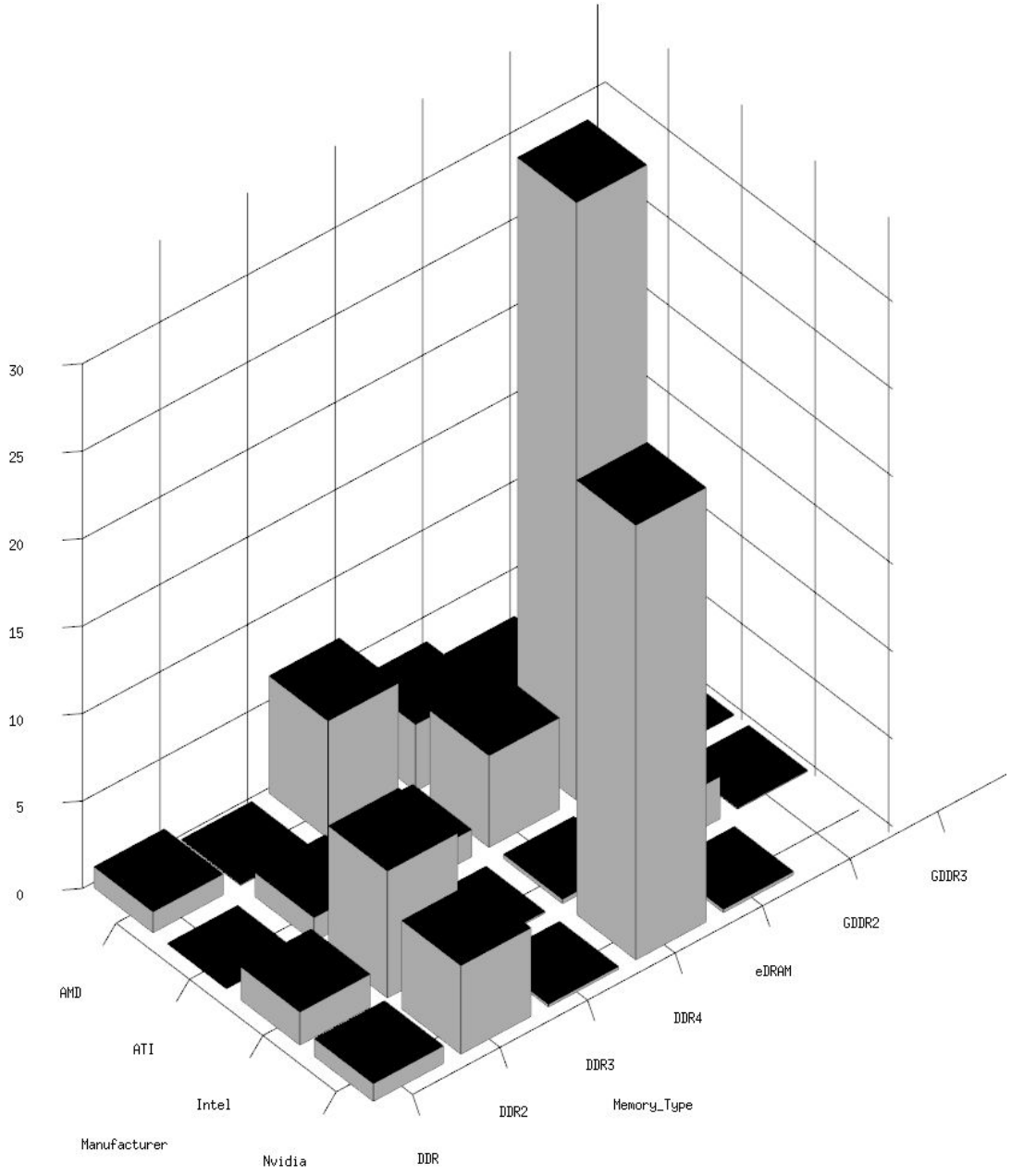
We can find an important relationship between the variable Manufacturer and Open_GL. Nvidia GPUs have OpenGL 3.2 and some of them have OpenGL 3. Intel GPUs have OpenGL 3.1. ATI and AMD GPUs support principally lower versions of OpenGL.



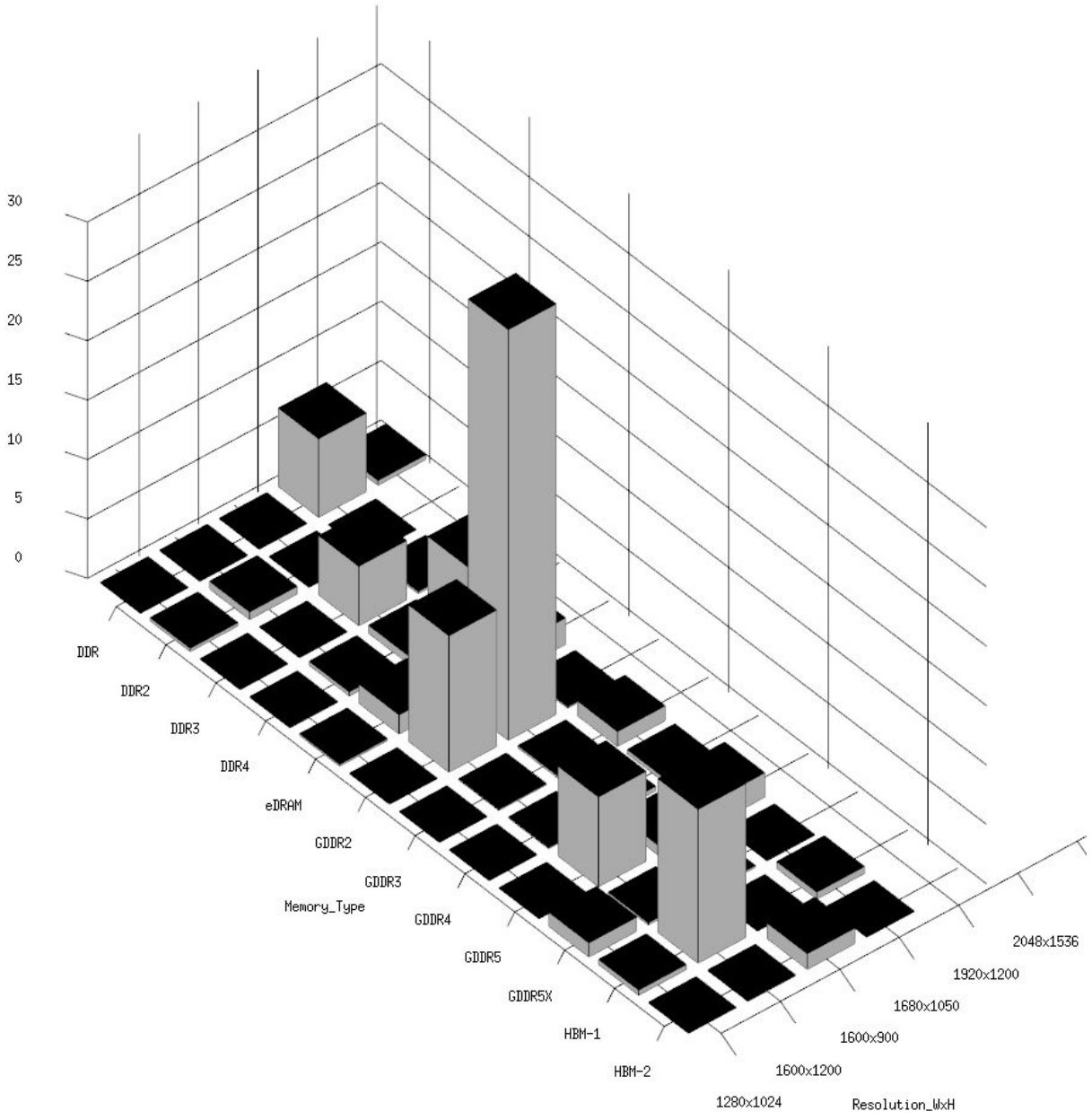
We can see that a big amount of GPUs with a resolution of 1600x1200 have an unknown version of Direct X as well as there are GPUs with a resolution of 1600x1200 whose Direct X is between 10.1 and 11.2. Most of the GPUs with Direct X 5 support a resolution of 1680x1050.



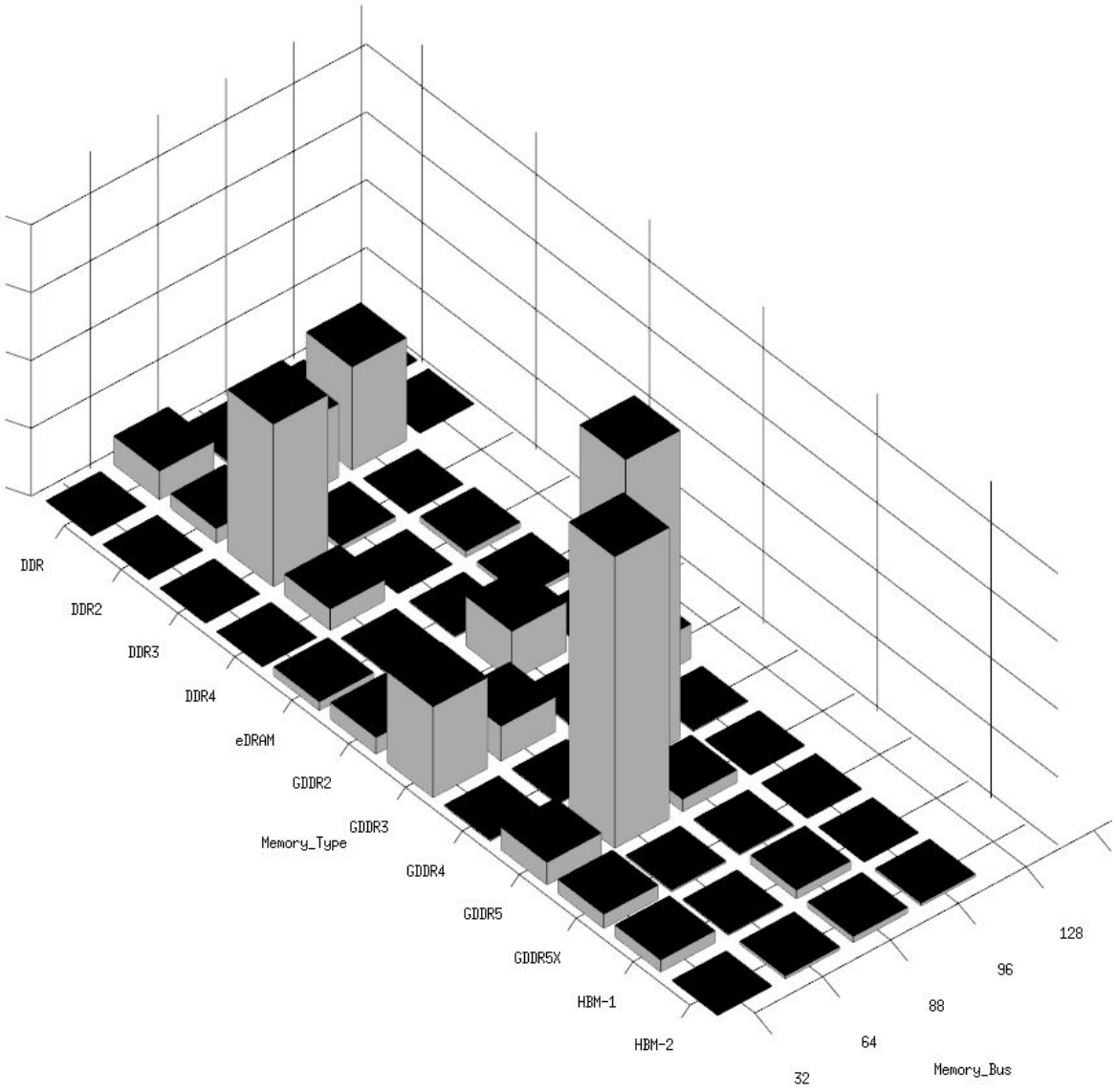
A high quantity of Nvidia GPUs have a DDR4 RAM, and others have a DDR2 RAM. Most of the instances from our dataset are ATI GPUs with GDDR2.Principally, AMD GPUs have a DDR3 RAM and Intel GPUs have a DDR2 RAM.



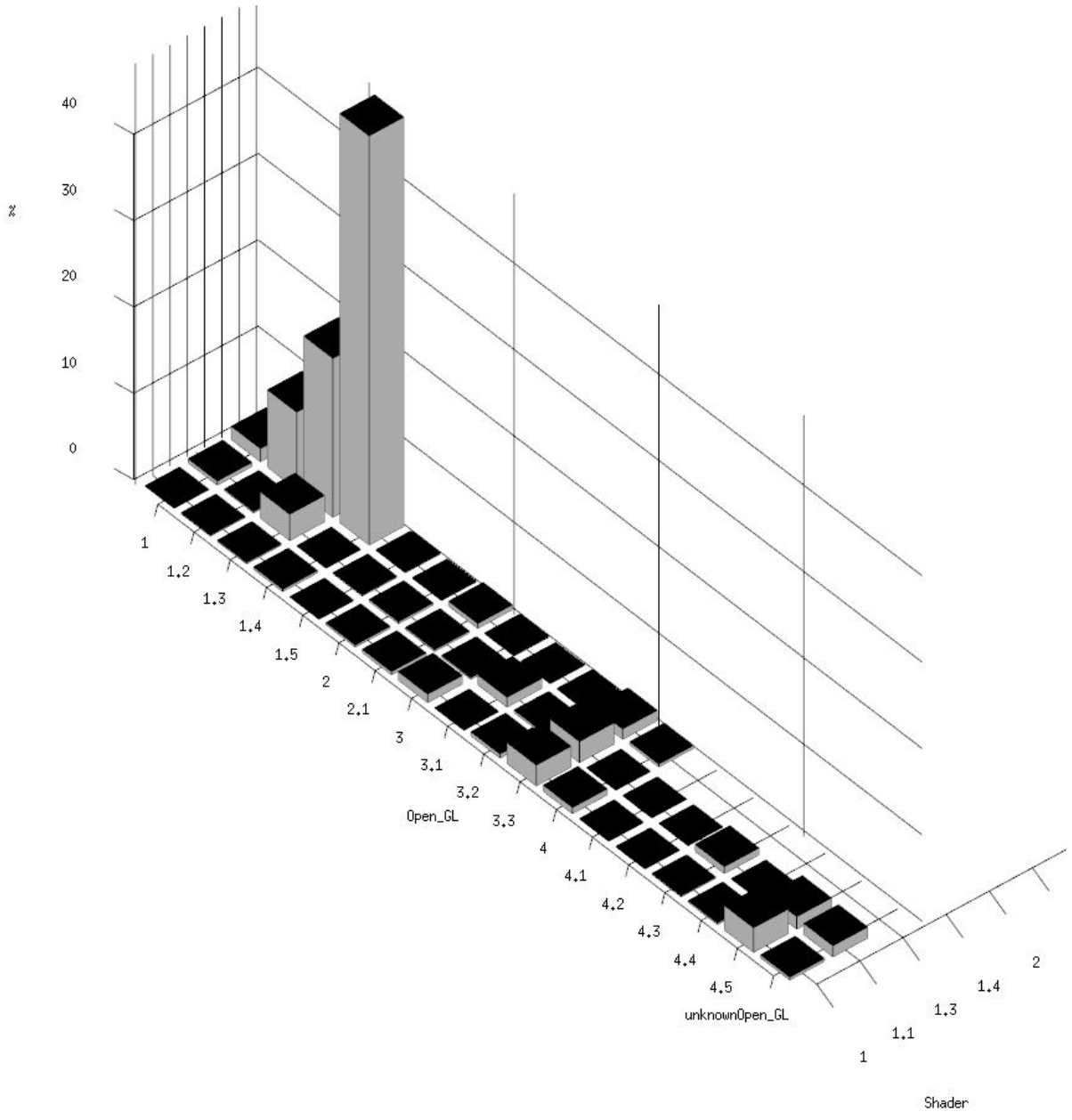
In our dataset we can find a lot of GPUs which support a resolution of 1680x1050 and a GDDR. In addition, we can find a lot of GPUs which support a resolution of 1600x1200. Moreover, an amount of GPUs, with resolution of 1920X1200, have a DDR memory. A big variety of RAMs can be found in the CPUs of resolution 1290x1024.



We can see a relation between the memory and the memory bus. In our dataset, GDDR5s have mainly a memory bus of 64 KB and GDDR3s have a memory bus of 32 KB. DDR2s have a memory bus of 88 KB or a memory bus of 96 KB. Moreover, DDR3s have a 64 KB memory bus.



This plot shows us the relation between OpenGL and the Shader Model. Lower versions of OpenGL have a Shader Model whose version is 1.3. Furthermore, OpenGL v4.5 has a Shader Model whose version is 1.0.



6. Data Preprocessing

6.1. Reading the original data matrix

In order to read the initial dataset, we will type in RStudio the following lines with the objective of read the csv file:

```
setwd("Path") #set the working directory
df <- read.delim("All_GPUs.csv", sep = “,”, header=T);
```

Once we have executed these two lines we already have our data matrix stored in the variable called df. Notice that when we read the csv file we indicate the separators of each value, in this case we have that all the values are separated by commas and also we have a header, which contains the name of the columns, therefore, we have to specify that characteristic setting header to TRUE.

6.2. Cutting the original data matrix

6.2.1. Treatment of Rows

Once we have read the dataset stored in our variable when we execute the following command:

```
dim(df)[1]
```

We can observe that the output tells us that it has 3406 rows, therefore, we will not delete any row from now on since we have been told that the maximum recommended number of instances were about 3000 due to some algorithms or future procedures which take a lot of time if we have many instances, consequently we won’t delete any instance unless we see that one or more rows are irrelevant for some reason or that maybe we need to delete rows when cleaning the data because of the missing values.

Nevertheless, there are some rows that need to be deleted in order to clean our dataset. That’s because if we look into our dataset we see that all the variables have at least one null value, which can be an empty string, or

incorrect information, everything related to this process of cleaning is explained in the following section. Therefore, we will face the fact that we won't be able to compute Knn or PCA since both of these algorithms don't support Nans. Because of this situation we decided to delete some rows that had Nan values, however, we make this decision taking into account one main factor, which is that we wanted to "clean" so these variables won't have any empty value but we had to delete the minimum necessary number of rows, so after executing the following script we decided which of them delete.

```
for (k in var_numericas) {
    l <- sum(is.na(df_selected[,k]))
    print(k)
    print(l)
}
```

```
[1] "Core_Speed"
[1] 936
[1] "L2_Cache"
[1] 1185
[1] "Max_Power"
[1] 626
[1] "Memory"
[1] 420
[1] "Memory_Bandwidth"
[1] 126
[1] "Memory_Speed"
[1] 105
[1] "TMUs"
[1] 539
[1] "Texture_Rate"
[1] 545
```

```
for (k in var_cualitativas) {
  l <- sum(is.na(df_selected[,k]))
  print(k)
  print(l)
}
```

```
[1] "Direct_X"
[1] 0
[1] "Architecture"
[1] 0
[1] "Manufacturer"
[1] 0
[1] "Memory_Type"
[1] 0
[1] "Open_GL"
[1] 0
[1] "Shader"
[1] 0
[1] "Name"
[1] 0
[1] "Resolution_WxH"
[1] 0
[1] "Release_Date"
[1] 0
[1] "Memory_Bus"
[1] 0
```

First of all, as we can see we only delete nulls in our numeric variables, since our boolean variables are "perfect" and in our categorical variables if we find a Nan we are going to substitute that value by "Unknown" ++ "Name of the variable". So as we can see the variables that have less nulls are Memory_Speed and Memory_Bandwith, which will be enough to recover all

our null values with the KNN algorithm. However, we want to point out that since the variable Memory_Bus was perfect all this time (it had not missing values), and it is considered as a categorical variable but its values are numerical we will also use this variable to compute the KNN algorithm. Finally, we want to point out that we will focus on the main reason of deleting these rows for these two variables in the KNN section since the reason is how the KNN algorithm works, therefore we decided to explain all the information related to this in that section.

6.2.2. Treatment of Columns

As we have a large number of variables (34 columns), which is too many for our objective, we are going to suppress some of them with the objective of making the future analysis of the dataset more friendly, however, we are always granting that the minimum number of variables for each type is respected. In order to choose which variables we are going to delete first of all we are going to look at which of them have more missing values (which we will see this part at point 5. Decisions taken for each step).
We decided to select the following variables which, from our point of view, were the most interesting ones between those variables with an understandable amount of missing values.

```
var_numericas ← c("Core_Speed", "L2_Cache", "Max_Power", "Memory", "Memory_Bandwidth", "Memory_Speed", "TMUs", "Texture_Rate", "Release_Date")

var_binarias ← c("Dedicated", "Notebook_GPU", "SLI_Crossfire");

var_cualitativas ← c("Direct_X", "Architecture", "Manufacturer", "Memory_Type", "Open_GL", "Shader", "Memory_Bus", "Name", "Resolution_WxH")
```

Doing that, we have created 3 vectors (var_numericas, var_binarias and var_cualitativas) with the names of the variables (columns) we want to keep.

Observe that some variables like Integrated have been eliminated, however we discuss why we have done that in the leftOver section.

Then, after selecting the columns that we considered more useful for our objective, we can create one more vector to add them all.

```
all ← c(var_cualitativas, var_numericas, var_binarias)
```

Finally, to create the dataset without the variables we have not selected, we will do:

```
df_selected ← df[all]
```

so that we store on df_selected the dataset with only the columns that are in "all" we created before.

Nevertheless, it won't be that easy to obtain all the data prepared for checking if it was missing, if it had outliers or be able to compare numerical data between them. That's because when we take a look inside the different variables we notice that despite the fact that some of them are numerical, like Memory_Speed, they were classified as categorical by Rstudio when doing the summary. That was because a lot of them had characters to indicate the unity of measurement, for example Mbytes/s or MHz. Therefore for all of these variables first we had to delete all these characters and after deleting these characters transform from the actual string into a number.

String		String		Number
"480 MHz"	→	"480"	→	480

In order to do that, we firstly removed the " MHz" substring from the string value as follows:

```
df[var_numericas]$Core_Speed←sub("MHz","",df[var_numericas]$Core_
Speed)
```

So that, almost all numerical variables had a substring with its units or a prefix/suffix. Then, we proceed to remove all of them the get a string with the numerical value:

```
df[var_numericas]$Max_Power←sub("Watts","",df[var_numericas]$Max_Power)

df[var_numericas]$L2_Cache←sub("KB","",df[var_numericas]$L2_Cache)

df[var_numericas]$Memory_Bandwidth←sub("GB/sec","",df[var_numericas]$Memory_Bandwidth)

df[var_numericas]$Memory←sub("MB", "", df[var_numericas]$Memory)

df[var_numericas]$Memory_Speed←sub("MHz","",df[var_numericas]$Memory_Speed)

df[var_numericas]$Texture_Rate←sub("GTexel/s","",df[var_numericas]$Texture_Rate)

df[var_cualitativas]$Memory_Bus←sub("Bit","",df[var_cualitativas]$Memory_Bus)

df[var_cualitativas]$Direct_X←sub("DX","",df[var_cualitativas]$Direct_X)
```

Once done, we have to convert the numeric string into a number since we do not want Rstudio to detect the variable as categorical.

```
df[var_numericas]$L2_Cache←as.numeric(df[var_numericas]$L2_Cache)

df[var_numericas]$Max_Power←as.numeric(df[var_numericas]$Max_Power)

df$Core_Speed ← as.numeric(as.character(df$Core_Speed))

df[var_numericas]$Memory_Bandwidth←as.numeric(df[var_numericas]$Memory_Bandwidth)

df[var_numericas]$Memory ← as.numeric(df[var_numericas]$Memory)

df[var_numericas]$Memory_Speed←as.numeric(df[var_numericas]$Memory_Speed)

df[var_numericas]$Texture_Rate←as.numeric(df[var_numericas]$Texture_Rate)
```

In the case of the categorical variables, we had fewer problems, such as the variable Direct_X having values “10” and “10.0” which its meaning was the same. In order to fix this problem, we did as follows:

```
df[var_cualitativas]$Direct_X←gsub("*\\.[0]+","",df[var_cualitativas]$Dir
ect_X)
```

Also, there were some variables marked as numerical by R instead of categoricals because the values that they contained were the compatible version, for example 3.2, therefore R understood this as a numeric variable instead of a categorical one. To fix this we use the function as.factor() which transform numerical values into categorical ones.

```
df[var_cualitativas]$Direct_X ← as.factor(df[var_cualitativas]$Direct_X)

df[var_cualitativas]$Open_GL ←as.factor(df[var_cualitativas]$Open_GL)

df[var_cualitativas]$Shader ← as.factor(dar_cualitativas]$Shader)
```

Finally, for Release_date we had to clean the string a bit:

```
df[var_cualitativas]$Release_Date←sub("\n","",df[var_cualitativas]$Relea
se_Date)

df[var_cualitativas]$Release_Date←gsub("","",df[var_cualitativas]$Relea
se_Date)
```

Also for the boolean variables instead of having the TRUE or FALSE value they were declared as “Yes” and “No” so we needed to fix this too with the script:

```
df[var_binarias] ←  df_selected[var_binarias] == “Yes”
```

6.3. Detection and treatment

6.3.1. Missing Data

The missing variables were not easy to solve, since the dataset represented NA with special characters like “\n-” which for some reason R had difficulties to detect them, and there were also some cells that were empty but

there was not a missing value declared. Therefore we needed to fill all these cells with missing values with the following script:

```
df_selected[df_selected==””] ← NA
```

Although, for the categorical variables, we did as follows:

```
df_selected[df_selected==””] ← UnknownNV
```

where NV is the name of the variable. For example, a missing value of the variable “Name” would be UnknownName.

6.3.2. Outliers and errors

As we saw on point 3.2, there were several outliers, but now, we will see which of them are indeed errors or whether they are not.

Firstly, observing the Boxplot of L2_Cache there are some points above the 4000 value. But, in fact, it is reasonable to think that this value is truly possible so that we will not proceed to delete it. Instead, if we observe the metadata matrix, we see that L2_Cache takes some (or one) values to 0 which is not possible, then, we proceed to remove them.

Secondly, observing the plots of Max_Power we see that there are some outliers above the 600 value, which, as before, is perfectly possible. What it is not, is that, when taking a look at the metadata matrix, we see that there are values below 4 which is not possible. Then, we proceed to remove the values below 4.

Thirdly, there are the other numerical plots which have several outliers such as Memory, Memory_Bandwidth, TMU and Texture_Rate. But, when analysing them, we can assure that they are not mistaken.

Finally, we do at the script as follows:

```
l ← which(df_selected$Max_Power <4)
df[l, "Max_Power"] ← NA

l←which(df_selected$L2_Cache == 0)
df[l, "L2_Cache"] ← NA
```

6.3.3. KNN

Now that we have cleaned all the Outliers, replaced all the values like “” by Nans, cleaned all the variables which were strings by numerical and so on. Therefore, we finally have a Dataset that has all the values in a correct format so we can compute the KNN algorithm.

First of all we are going to explain how the KNN algorithm works which will let us explain why we have deleted previously all those rows from our dataset. The main objective of the KNN algorithm is to fill the null values of numerical variables with numerical values, which would be reasonable, for example if we have two variables, for example the price of a house and the square meters that it has follows this equation price_of_house = square_meters*2, then if we find a null value in the price of the one house with the square meters the knn algorithm should be able to predict that the result is two times the square_meters.

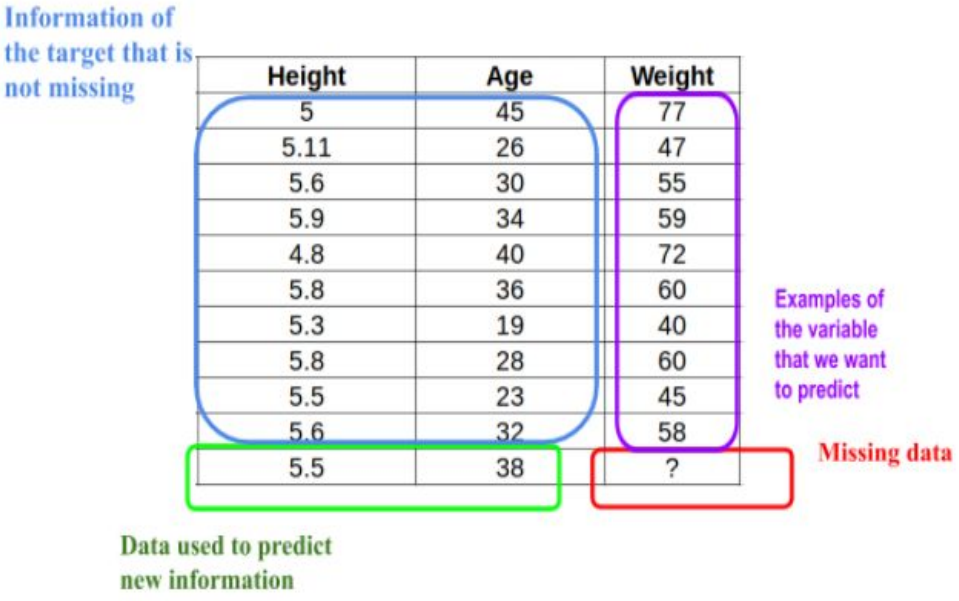
Therefore if we provide to the knn algorithm a list of information then it should respond to us with a response that corresponds to information that it considers correct. But how do we do this? How do we train our KNN algorithm in order to learn the relationship between one by one variable or two by one ? Well in order to do this we have to preparate our data but there is one restriction, that there can be missing values, which is the main reason that we deleted rows as we mentioned before. That’s because the knn algorithm works in the following way.

Imagine that from the following table we want to predict the weight when we have the Height and Age because we have a missing value, then we could split our table into three different tables.

- The first one would contain all the values of the variable that has missings and therefore we want to predict but only the part that is not missing.

- The second table would be the submatrix that contains all the other variables **that can not have missing values** and that corresponds to the examples of the variable that we want to predict but that are not nulls.

- The third table would be the submatrix that contains all the other variables **that can not have missing values** and that corresponds to the rows where the variable that we want to predict is missing.



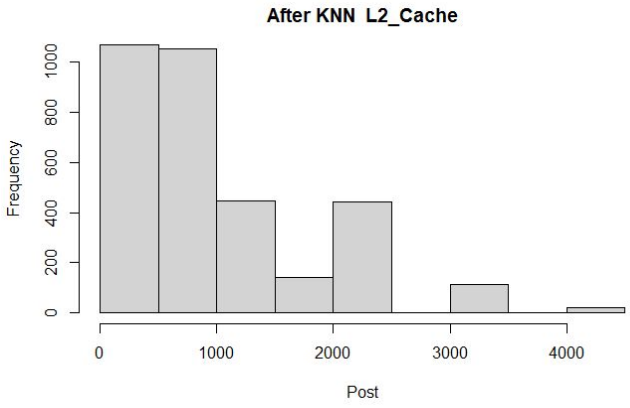
The objective here is to show to the knn algorithm the second table and the first one, so the algorithm can learn the relationship between that variables that have all the information that we have (in the example of the house would be the square meters) and the information that we want to predict (the price of the house), with the objective that when it learns that relationship it will be able to, given a set of values which have not information, give us what it thinks the value should be.

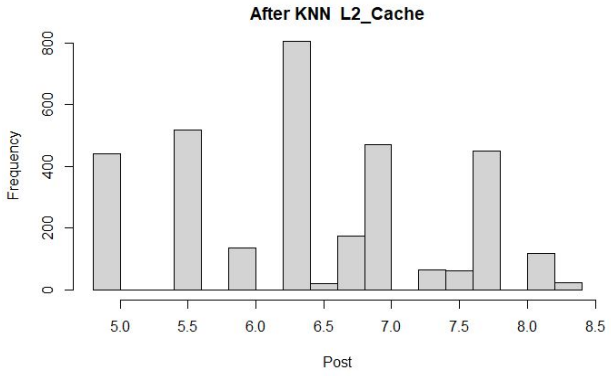
That's why we needed to clean some variables at the beginning of the preprocessing because it wouldn't be possible to predict the new variables or learn the relationship if we have nulls in the second or third table, since it would be like learning the relationship of Nothing with some information,

which doesn’t make sense. Of course other thing that we have done is for example if we have 4 columns that have null values and 2 that doesn’t, we pick up these 2 variables without null values, then we predict the value of the null elements for a third column, and repeat the process for the fourth column but this time we add that new column, then now we would have 3 variables that doesn't have nulls, and 3 that have some nulls, therefore as we proceed with this method our algorithm can learn better relationships and therefore make a better approximation. Finally we apply log for the exponential variables.

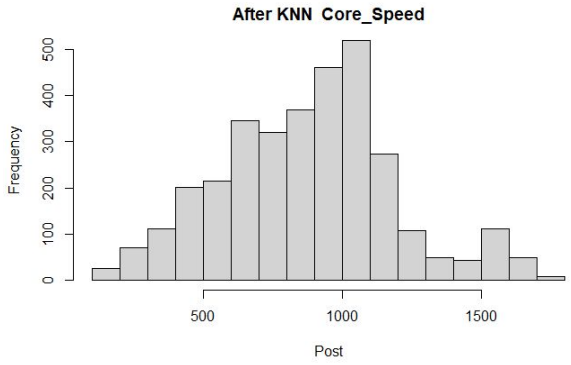
6.3.4. Results after preprocessing

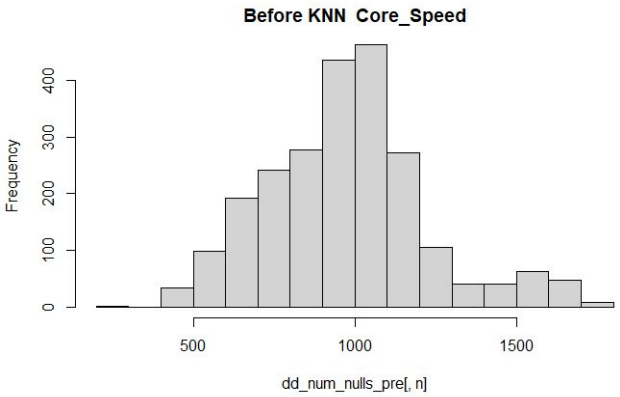
For each page we have the result before preprocessing and after preprocessing.



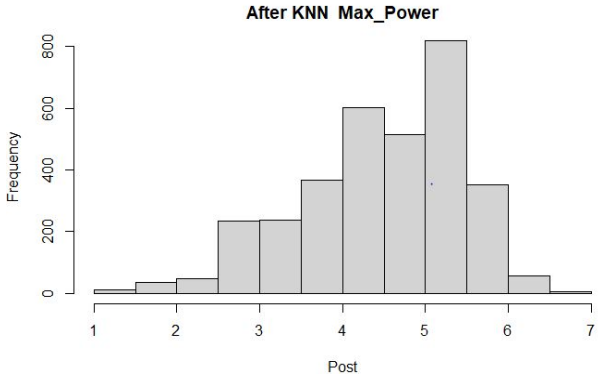


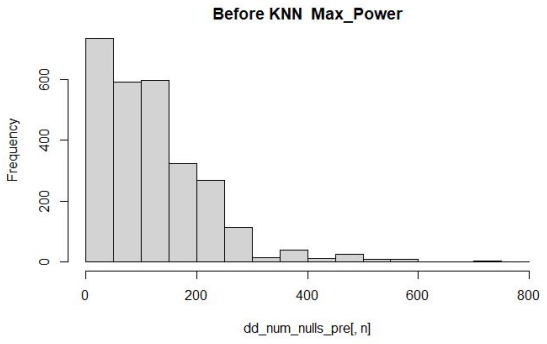
The first observation that can be taken is that, after applying KNN, there are more values around 1000-1500 thus the plot follows a descending distribution even more than before. Also, now, there are more values between 0-500 than before on 500-1000. Overall, the values changed to get an almost-descending plot.





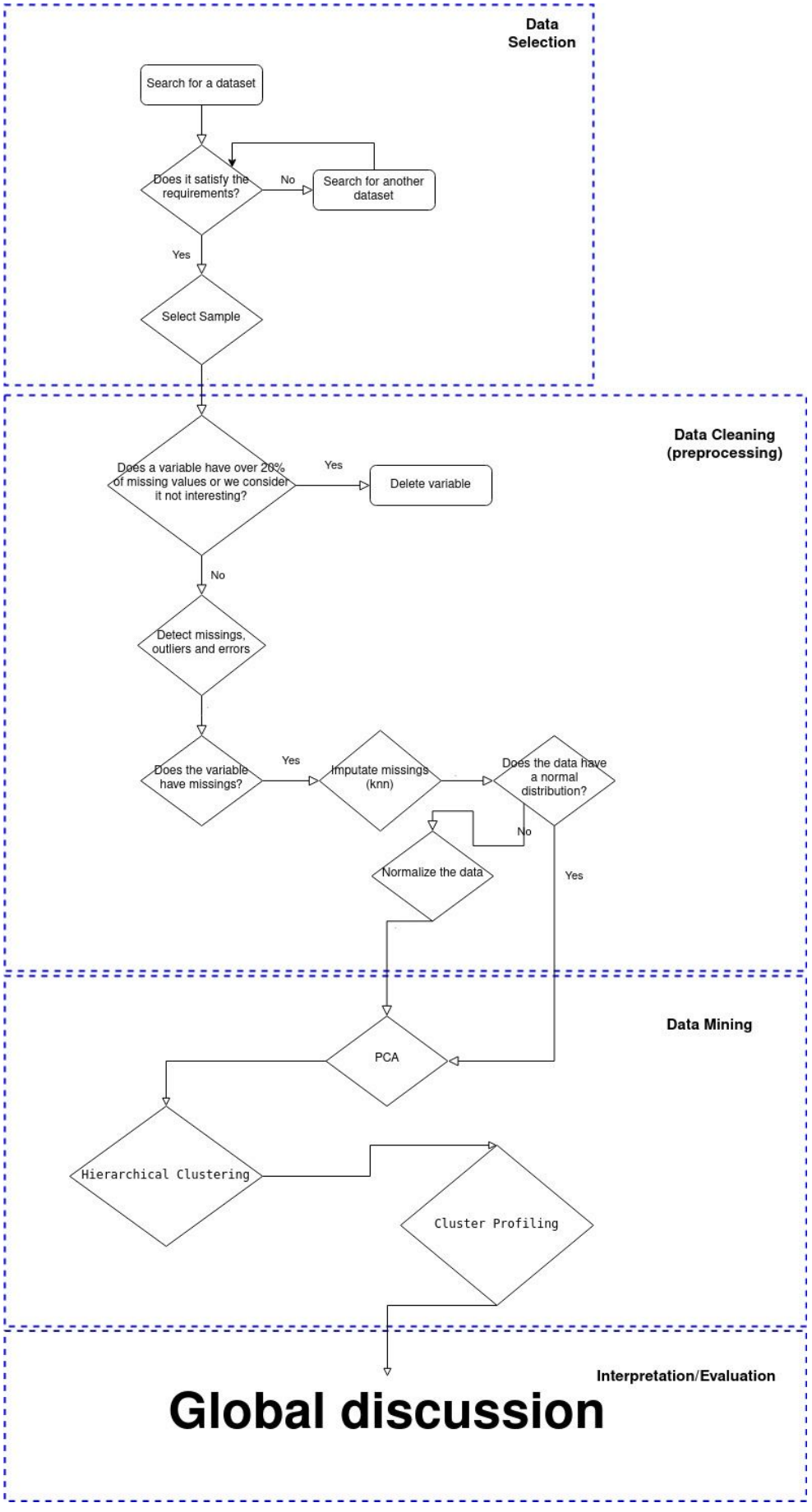
The first observation that can be taken is that, after applying KNN, there are more values around 1000-1500 thus the plot follows a descending distribution even more than before. Also, now, there are more values between 0-500 than before on 500-1000. Overall, the values changed to get an almost-descending plot.





The first observation that can be taken is that, after applying KNN, there are more values around 1000-1500 thus the plot follows a descending distribution even more than before. Also, now, there are more values between 0-500 than before on 500-1000. Overall, the values changed to get an almost-descending plot.

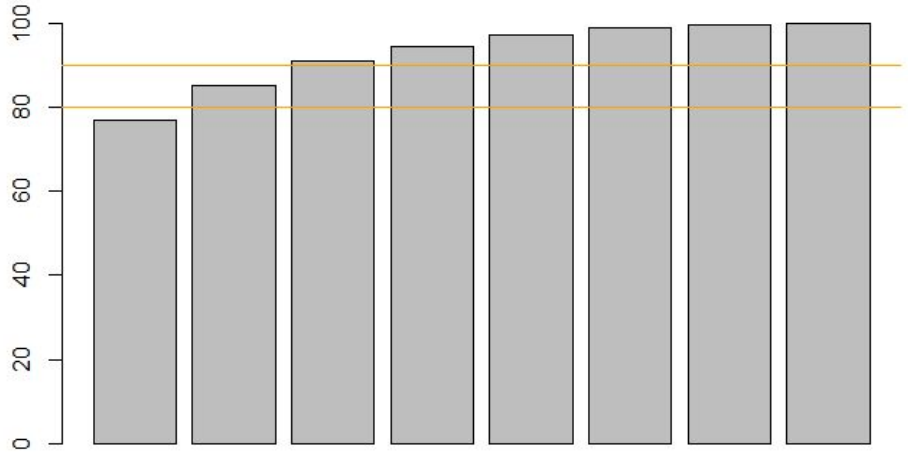
7. Data Mining process performed



8. PCA

8.1. Scree Plot

In this Section we are going to see the percentage of information about the accumulated variance among all the variables, or in other words the weight of each axis.

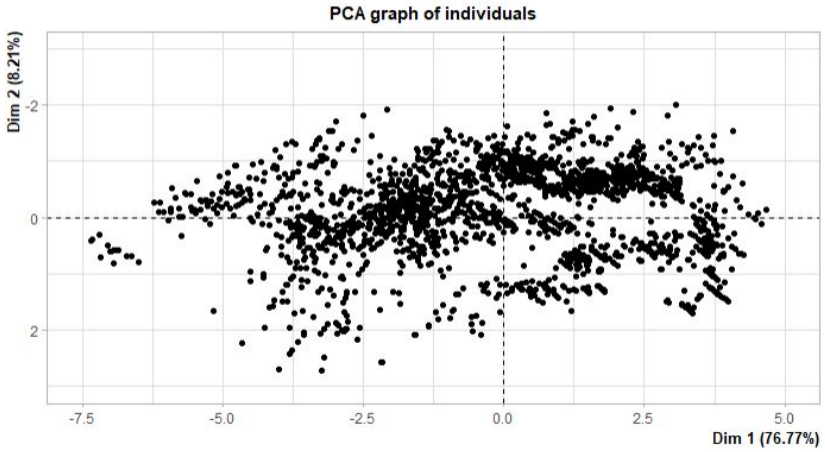


Here as we can see in our case with two components we are achieving a representation of 80 percent of the total variance, and with three components we achieve until 90 percent of the total variance representation, which is very good since we are able to visualize with a 3D plot the 90% the original relationship of our variables and also because since to his all the conclusions that we will able to extract from the analysis of the PCA will have a huge probability of being true, which allow us to have confidence in all the conclusions that we will made.

8.2. Factorial map visualization

8.2.1. Individual Projections

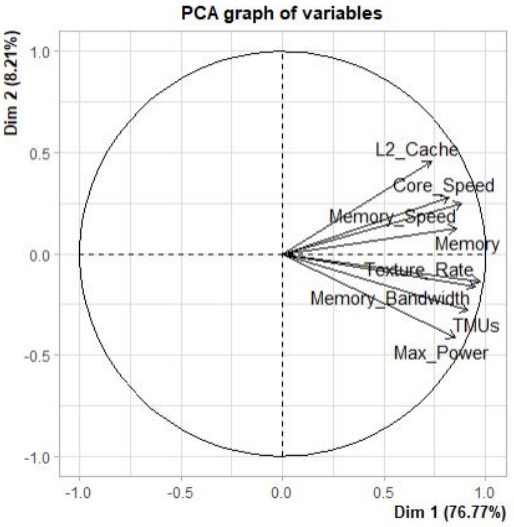
Now that we have already seen that the number of principal components that we need to visualize the relationship between the original variables with a 80% of fidelity respect the original distribution of the data, we are going to process to visualize the projection in this plot with 2 dimensions.



As we can see the first component comprends 76.77% while the second component only comprends 8%. Also, without any type of modification in this plot we can already see groups of points that are together and separated from others, which indicates that exist different groups, however we will discuss later in the following sections and we will see the characteristics that make each group unique.

8.2.2. Projections of Numerical and Qualitative variables

Now that we have visualized all the variables plotted in 2 dimensions, it's time to talk about how these variables are mapped into these 2 components, which can give us information about which variable has had the biggest importance when deciding all the Factorial planes. The first plot gives us that information. What the plot says is, if we interpret each numerical variable as a vector, how would look this vector in the new dimensions defined by the PCA algorithm. In order to analyze which of them has the biggest influence we have to focus on two characteristics:

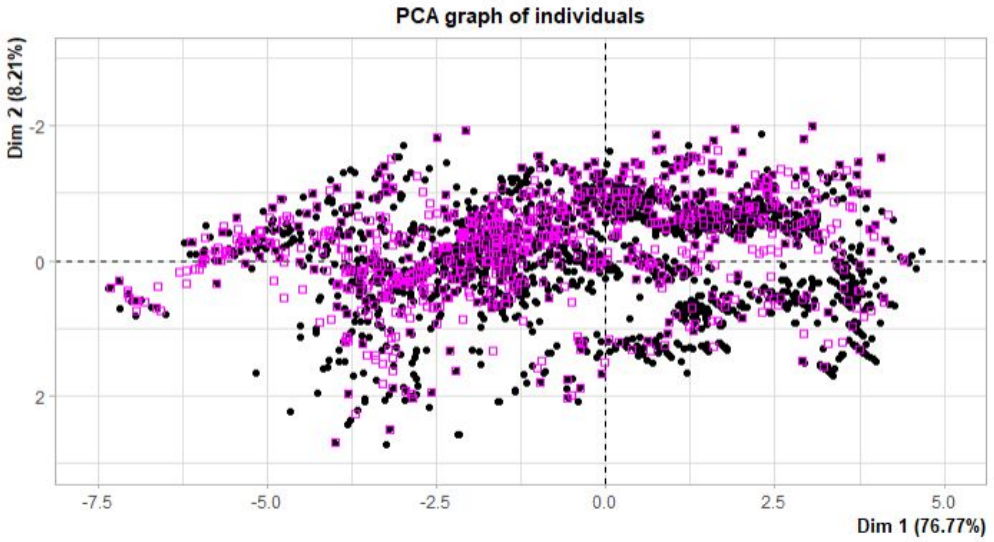


1) The first one would be to look for the variable that has most variation among the first factorial plane, which will be analyzed in the following sections, however we can already say that that variable is Memory
2) Which variable is more close to the first Component, since because of how this algorithm works, the variable which has the most impact will be the one closest to the first component, which is again the Memory variable.

   Therefore only looking at the plot we have been able to determine the numerical variable that has had the biggest importance when classifying all our data.

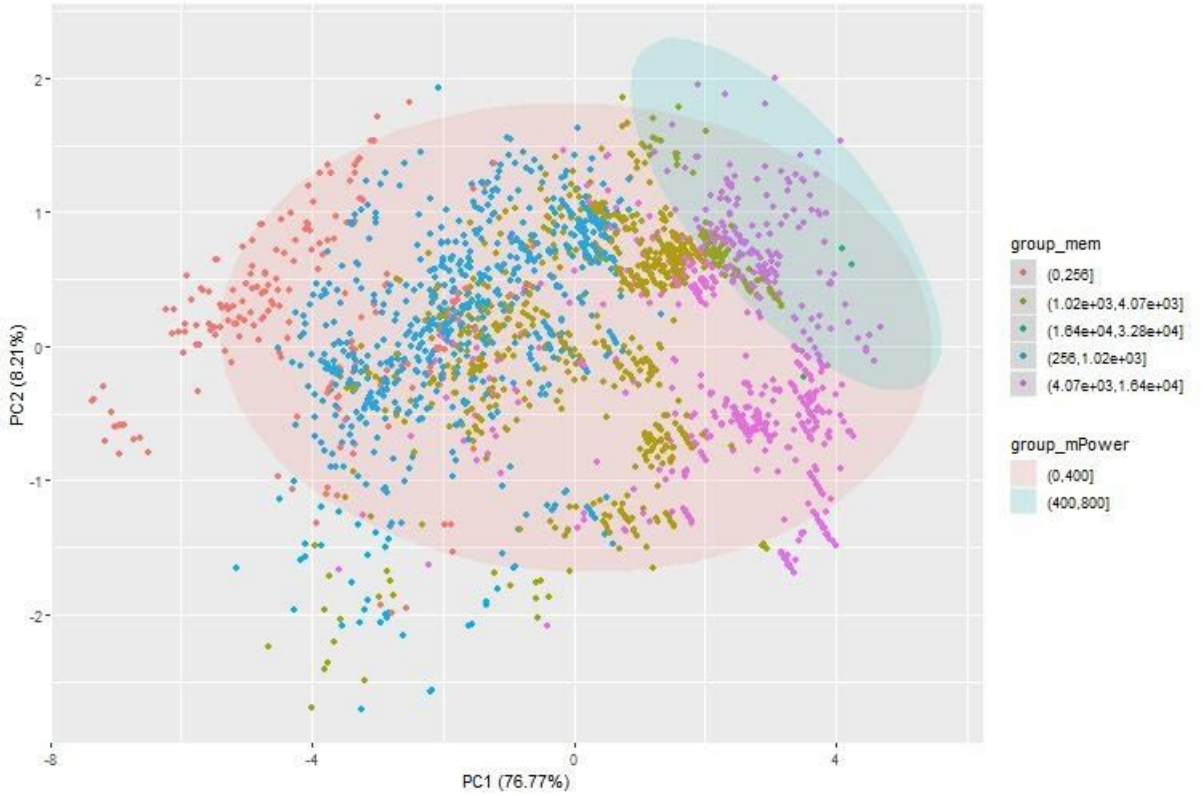
Nevertheless we still have a question that hasn't been answered yet, and that's what happened with the categorical variables that we had in our dataset, well we can see that in the following plot where all the purple points represents the centroids of these categorical variables, since they are not numeric they have to be treated in that way in order to make a correct analysis. As we can see the majority of them coincide with the numerical variables which is not extrange since we have to remember that we are representing the 80% of the total variance, so there are some differences between the numerical data and

categorical ones but due to our big capacity to represent the information this doesn't represent a problem for us.



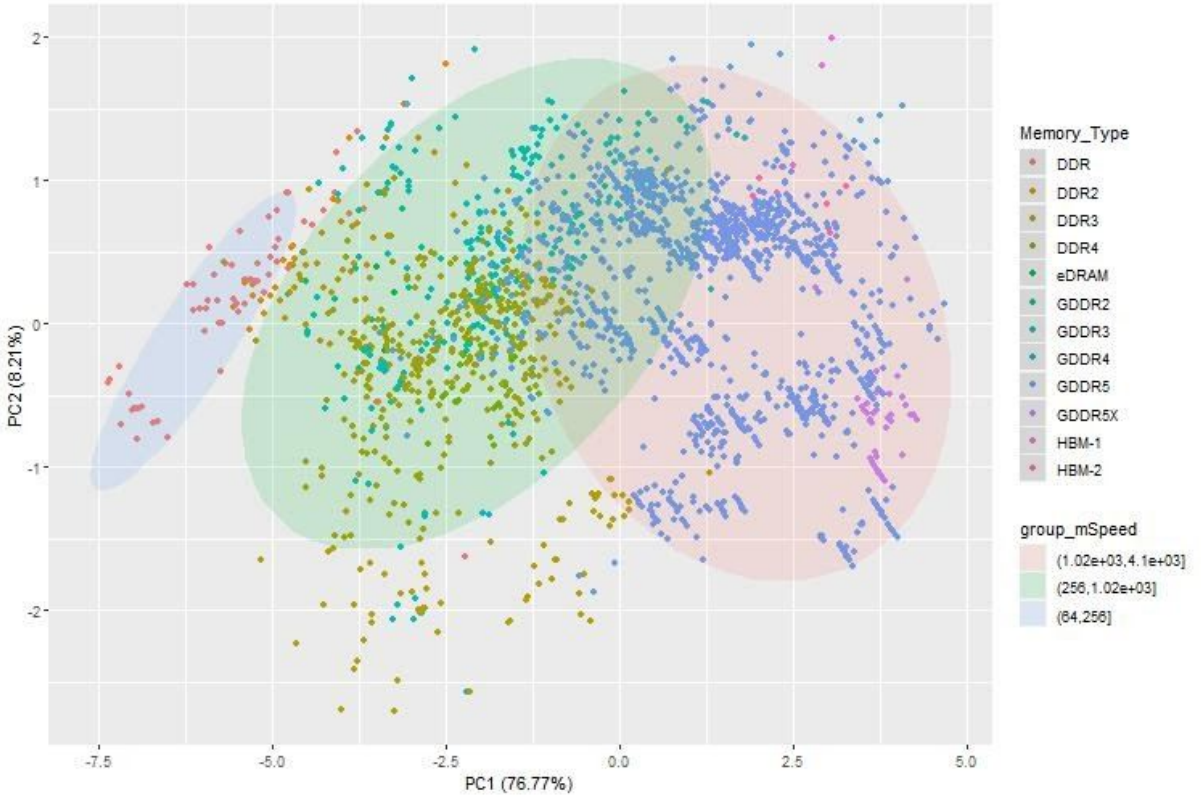
8.2.3. Relationship among variables

First, we are proceeding to explain which variables we have decided to study in order to associate them. We will act mainly on: Memory, Memory_Bandwidth, Max_Power, Memory_Type, Memory_Speed, Memory_Bus. In the graphs that we show you will be able to see that some variables have "group" in front of them, this is because we have converted the numerical variables to categories by grouping them in pre-established ranges. And then being able to get their ellipses and see in a much more visual way if they are related or follow an interesting pattern.



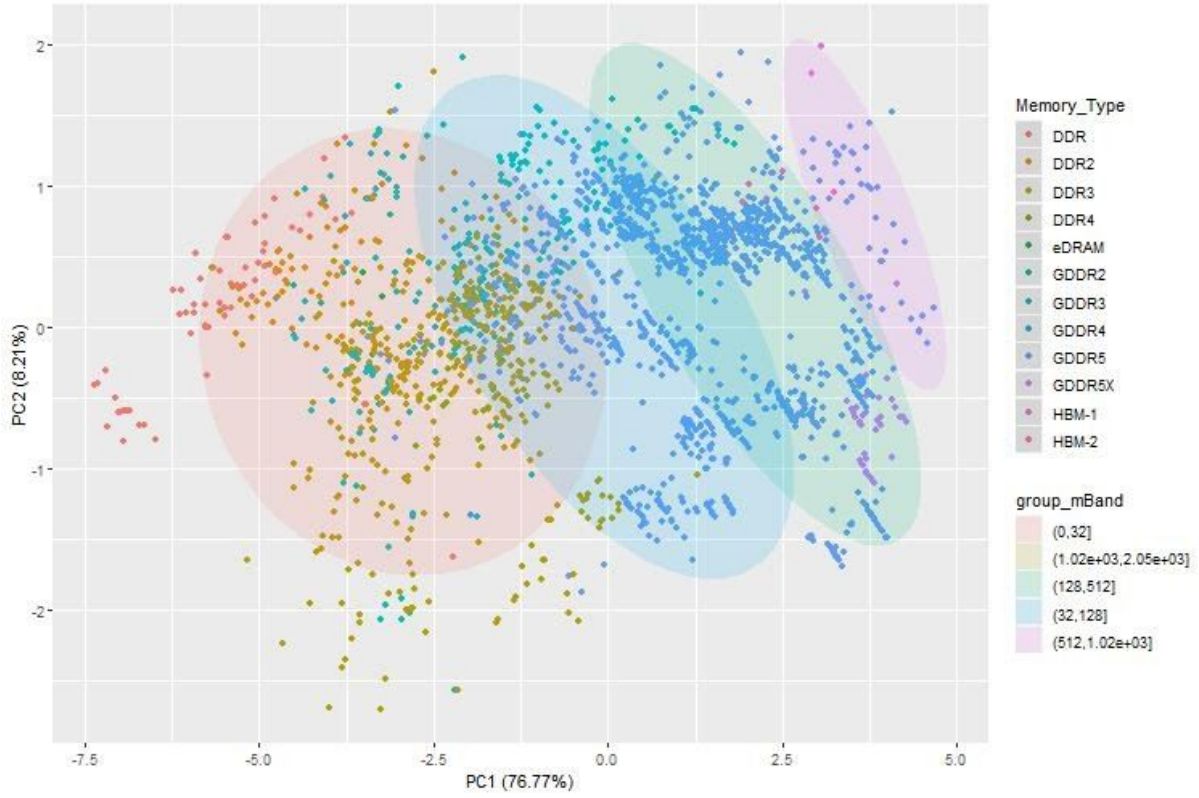
In this graph we analyze Memory and Max_Power. First of all, it is necessary to mention that group_mPower is divided into two large groups: 0 to 400 W and 400 W to 800 W. We decided to divide it this way because the ellipses resulting from managing it, for instance, by 100 to 100 did not offer much more info. And in the end they end up finding out something similar to two large groups. And then by group_mem we have simply divided it into powers of two until we gain the upper limit of this variable.

So, as we can see in the graph we have that GPUs with memory from 0 to 4096 MB have a consumption of 0 to 400 W. And that GPUs with more memory have a consumption of 400 W to 800 W. Although the ellipse of 400 to 800 W does not cover most of the memory points from 4096 to 16.384 MB we can clearly see that it is on the same PC1 axis and that the distance difference is not so outstanding or substantial.



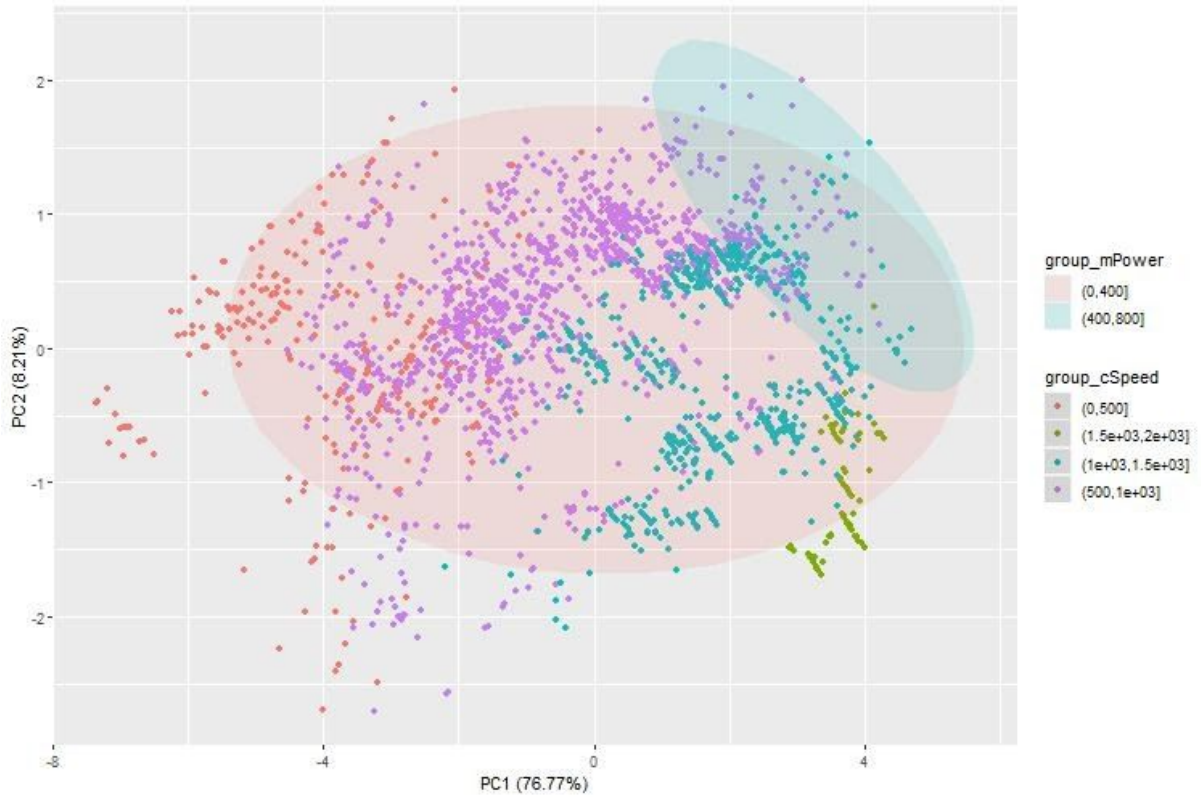
In this graph we will put Memory_Type and group_cSpeed under analysis. As we can see, group_mSpeed is the Memory Speed of the GPU, this has been divided into three different ranges that were the most significant. We can see a clear vertical division of the data.

Going from the left, we have a small group where we encounter the lowest speed of memory 64 to 256 MHz and in this ellipse is mostly the case of DDR memory, this being the oldest model. At speeds from 256 MHz to 1 GHz we would see DDR2 / 3/4 and GDDR2 / 3 memories, these are more modern memories than DDR and therefore quicker, so it makes sense that they are within this ellipse. Final 1 to 4 GHz we would see the GDDR4 / 5 / 5x and HBM-1/2 memories, these types of memories are the most modern memories within our dataset, seeing even a small sample of HBM being the fastest on the market.



Now we will see what type of relationship there is between Memory_Type and group_mBand. First I'm going to refresh what Memory_Bandwidth is: the rate at which data can be read from or stored into a semiconductor memory by a processor.

In this graph we can picture a clear segmentation of the data by the four ellipses. Something similar occurs in the other case of memory type, only that in this case we have that the highest transmission speed ellipses in bytes / second have the newest and fastest memory types inside. To clarify, we can realize that there is a color that does not appear and this is yellow, it is because it did not have enough points to create the ellipse, this is the fastest and is related to HBM.



Here we can consider something alike to what occurred in the first graph, where we were comparing the Max_Power and the Memory. We hold two large groups in which it can be seen that the graphics with more Core Speed have more consumption than the others. Even taking in the green points (those with the highest core speed) are not within the ellipse of 400 to 800 W, we can conclude that being on the same axis as this there will not be much difference between those that are within the ellipse of high consumption.

8.2.4. Conclusions

Once all the relationships that we considered provide interesting information have been analyzed, the only thing missing is to conclude the section.
Due to the fact that only with two components we already reached 80% of the total variance, we can say that the results obtained and discussed previously will not be far from reality.

Starting with the first graph, we can see that the Memory variable has a fairly clear distribution, thus providing information that the more memory the graph has, it offers a higher consumption, this is because it will probably be able to perform more operations and have more load of work than those with less.
The second graph we see the comparison between Memory_Type and Memory_Speed, it has helped to glimpse that both are strongly interrelated, because as we have seen Memory_Type is distributed quite well in the three speed groups of Memory_Speed.
In the third graph we find out the comparison between Memory_Type and Memory_Bandwidth, this has given us a lot of information because we did not expect them to be so related. The ellipses of Memory_Bandwidth divide vertically, going along with smaller groups of Memory_Type.
Finally, we have Core_Speed and Max_Power, in this case we are in a situation similar to that of the first graph. Where by not repeating ourselves, we end up concluding that the more Core_Speed, the more maximum consumption the graph has.

To point out, within the comparison of variables we have seen that both Memory_Type and Max_Power are two elements that help a lot to draw conclusions. That's because one indicates the type of memory, and doing a small search on Google how new the type of memory that we find in our data set is, it is easy to reach conclusions. Also on the other hand, the two ellipses of maximum power help to quickly identify and conclude that a clear pattern is seen that the newer or the higher transmission speed a component has, the more it consumes compared to the old ones.

9. Hierarchical Clustering

9.1. Description of data used

After looking at some different approaches, we decided to use all of our data, including all variables. This way we had more clusters, which gave us the opportunity to make a more detailed analysis and profiling of the clusters. But, when observing the resulting plots, we saw that the qualitative variables *name* and *Release_date* did not contribute anything since their values, in the case of *names*, are all different and, in the case of *release_date*, as we saw in section 2 in the *release_date* barplot, their values also take very different values.

We also did hierarchical clustering with the resulting output of the PCA and with just the numerical variables, but we concluded that it wasn't the right approach because we left out the equation a lot of valuable information.

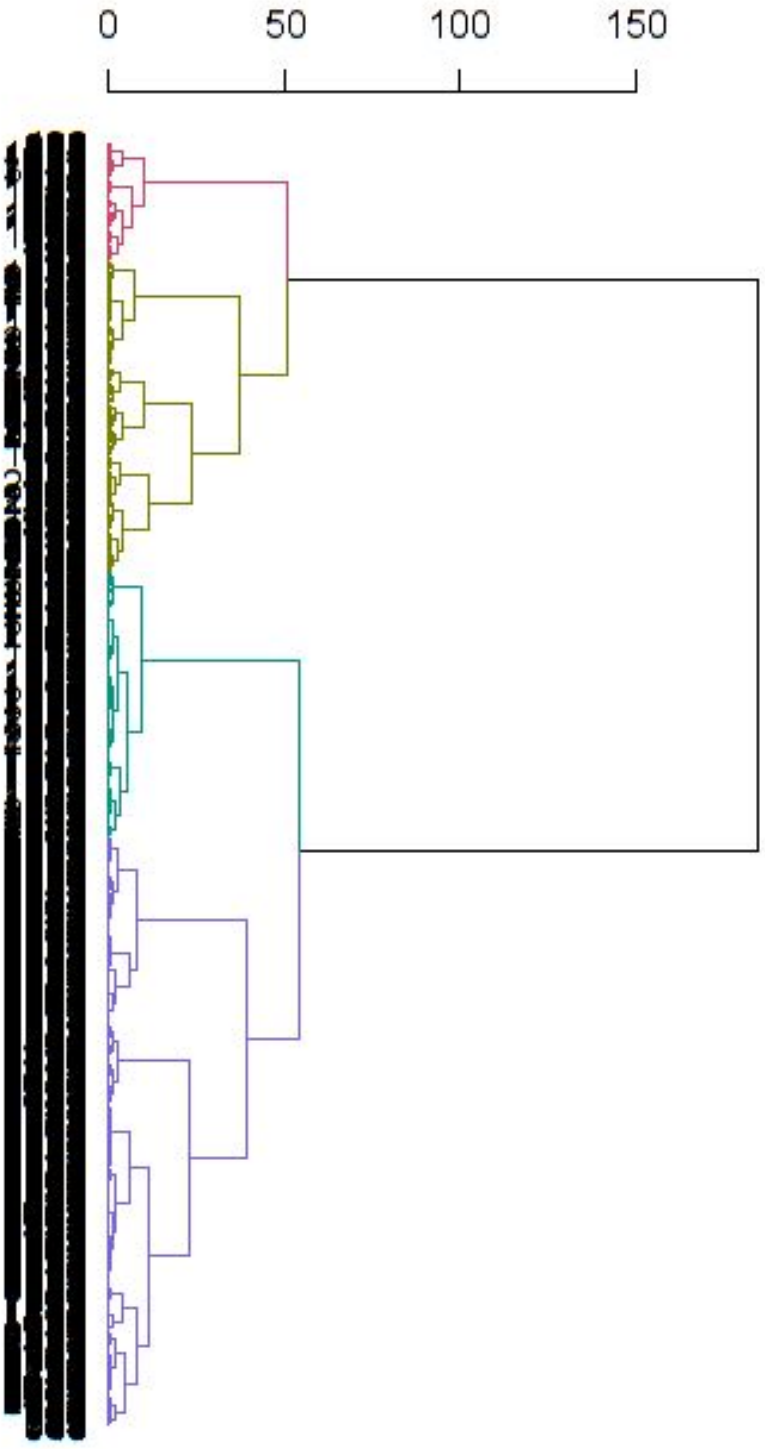
9.2. Methods and criterias used

We used agglomerative hierarchical clustering (also known as AGNES, Agglomerative Nesting) instead of the diviside strategy.

For the metrics, we used Gower dissimilarity coefficient to the square instead of using the euclidean distance, this way we could include all of our categorical and binary variables.

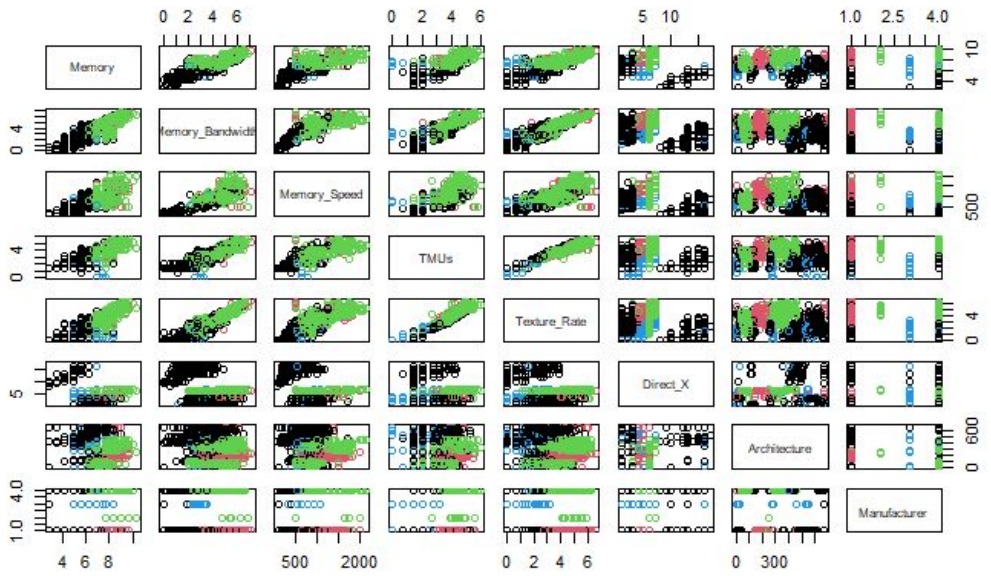
We used "Ward.D" for our aggregation criteria, since it was the criteria that had the most understandable dendrogram, the sizes of each cluster were very well distributed and the outcoming plots were clear enough.

9.3. Resulting Dendrogram

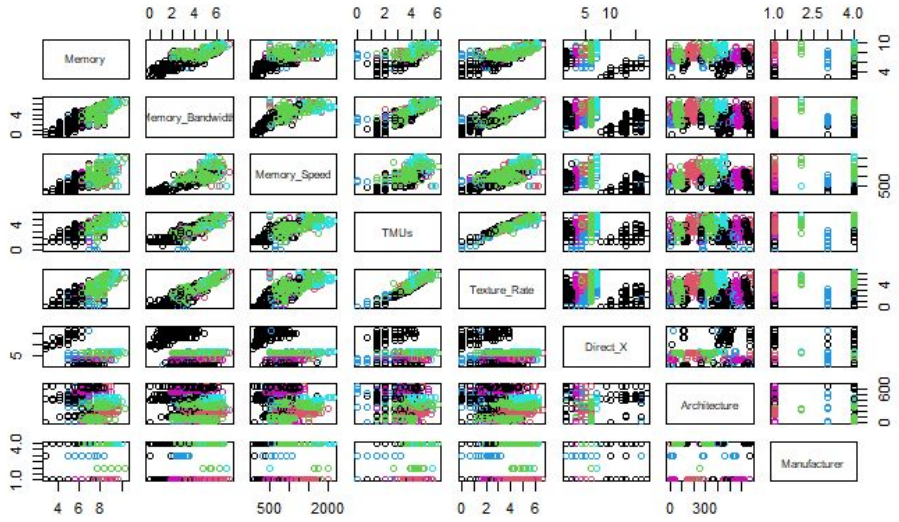


9.4. Final number of clusters and its size

Firstly, with the previous dendrogram, we had doubts as to whether to take 4 or 6 clusters. So, in order to decide if we had to take either 4 or 6 clusters, we plotted several variables so that the distribution of the values of each cluster could be seen clearer.



Plots with 4 clusters



Plots with 6 clusters

On each image, we have 8 columns (and rows) which gives us 8 different variables. The first 5 columns are numerical variables and the other 3 are categorical.

As can be seen from the second image, the cluster which is navy blue does not have such a great impact on the plots. Indeed, on an overall view, 4 colours can only be seen, predominantly green and black over numerical variables. In the case of the categorical variables, magenta barely can be seen as well as cyan.

So that, we finally took 4 clusters having each cluster the following size:

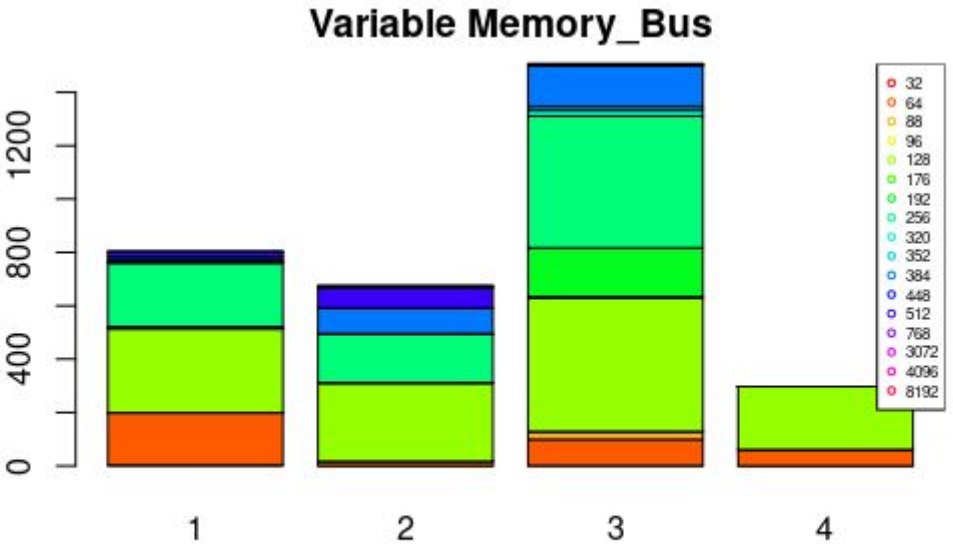
Class 1	Class 2	Class 3	Class 4
798	675	1507	301

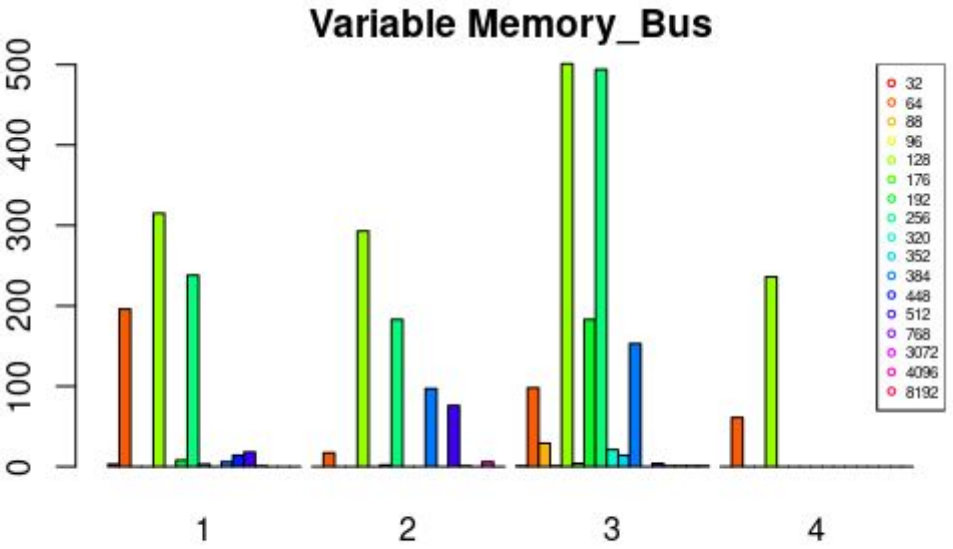
we can observe that class 1 and 2 are almost the same size although class 3 and 4 they distance themselves from the other classes. Indeed, if we calculate the mean, which would be 820.25, class 1 and 2 are pretty close to the mean but class 3 deviates from the mean 686.75 as well as class 4 which deviates 519.25.

That tells us that class 3 and 4 have similar deviations from the mean as well as class 1 and 2 have.

10. Profiling of clusters

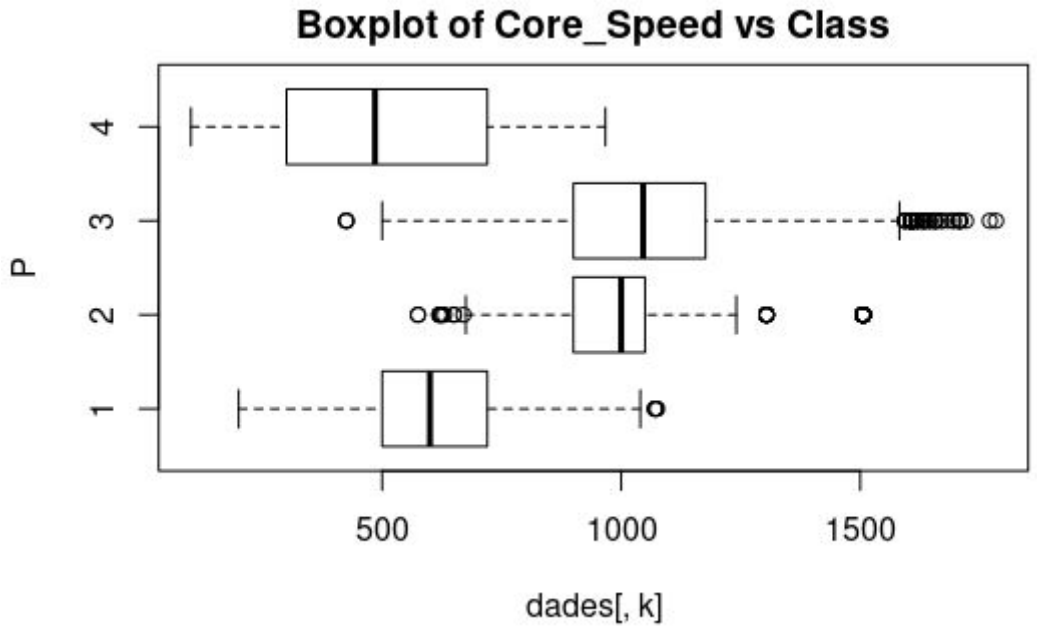
In this section we explain the profiling results obtained from the hierarchical clustering.

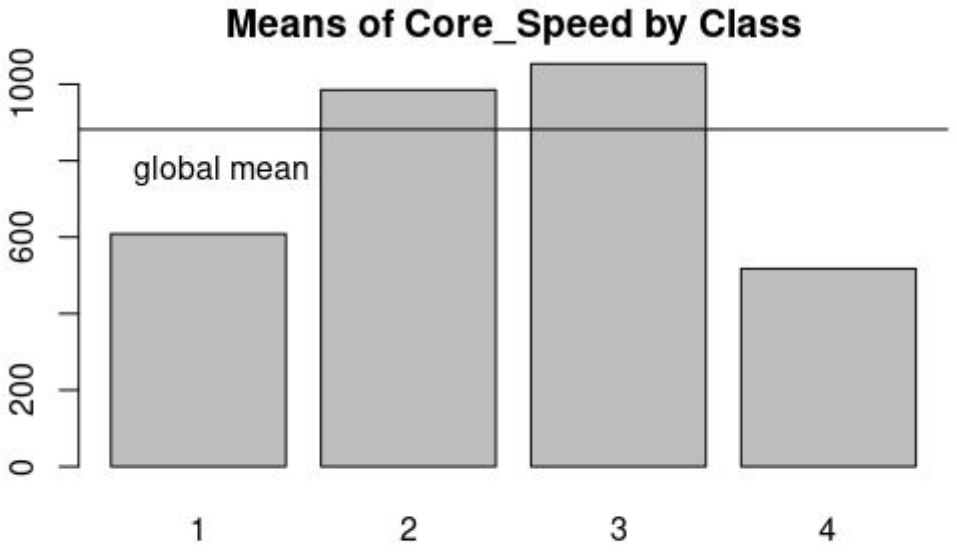




In the above pictures we can see the frequency of each memory bus in every class. As we can see, most of the individuals of 384 KB are in the group 3. Also, there is a high quantity of individuals of

64 KB respect the other groups. Group 3 is the group with the most memory buses, as we explained before.

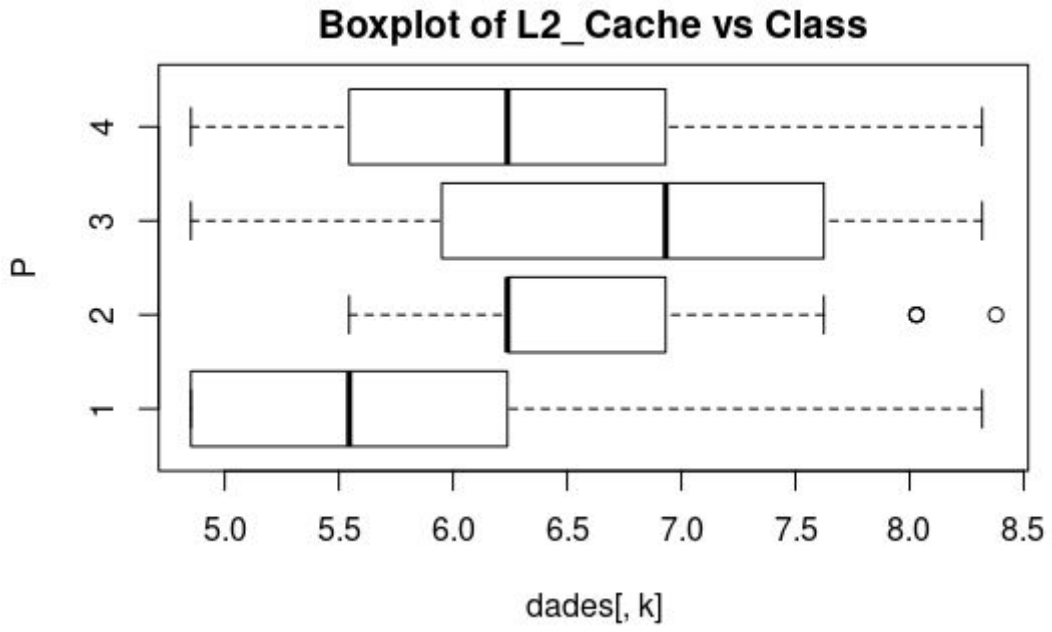


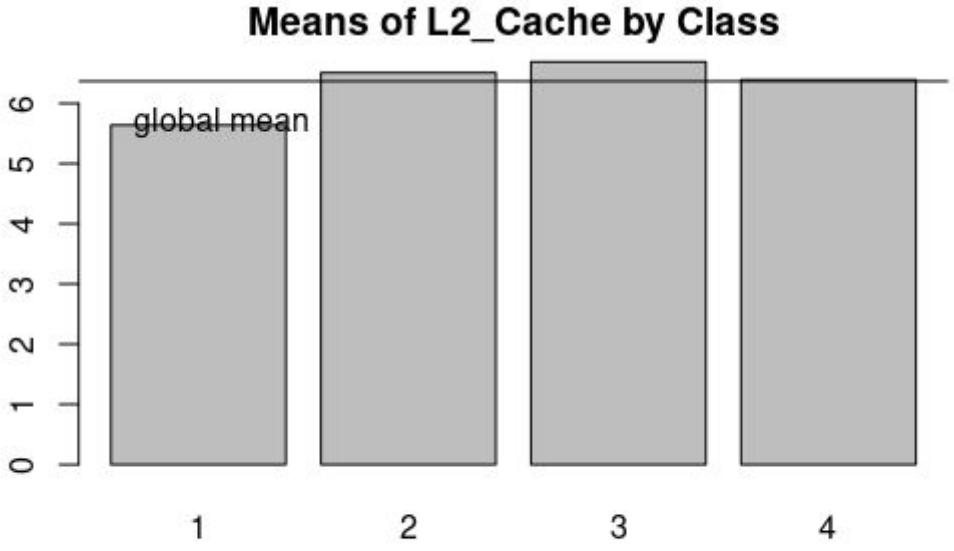


Respect the Core Speed, the global mean is situated in 750 Hz. In the group 1, the mean is located in 517.7 Hz, the 1-quantile (Q1) is in 300.0 Hz and the 3-quantile (Q3) is in 720 Hz.

In the group 2, the mean is located in 1046 Hz, the 1-quantile (Q1) is in 900 Hz and the 3-quantile (Q3) is in 1176 Hz. Group 3, the mean is located in 985.3 Hz, the 1-quantile (Q1) is in 900 Hz and

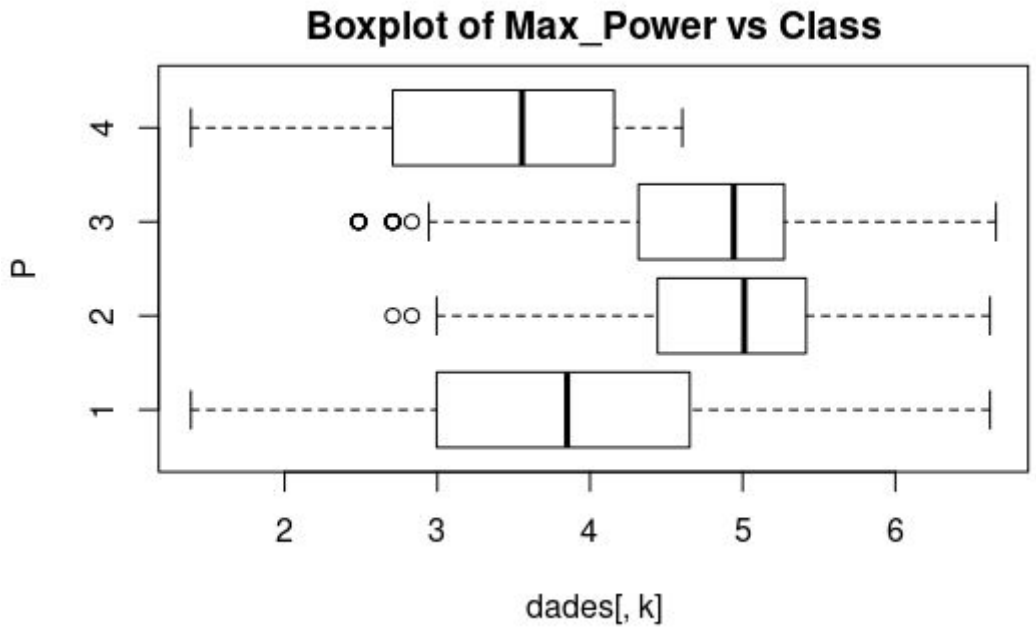
the 3-quantile (Q3) is in 1050.0 Hz. Group 4, the mean is located in 608.6 Hz, the 1-quantile (Q1) is in 500.0 Hz and the 3-quantile (Q3) is in 720.0 Hz.



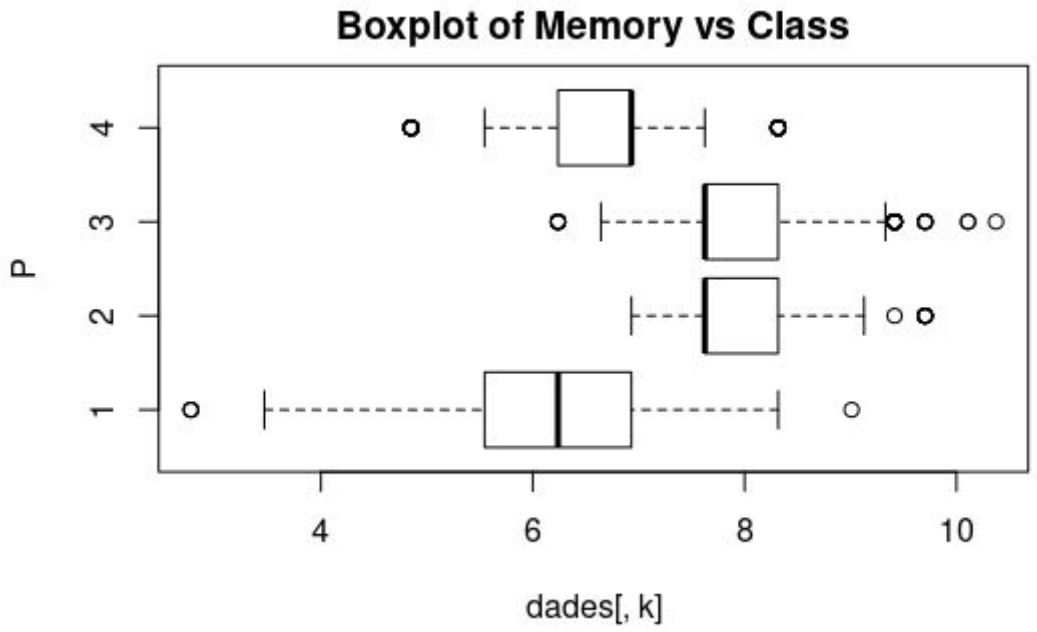


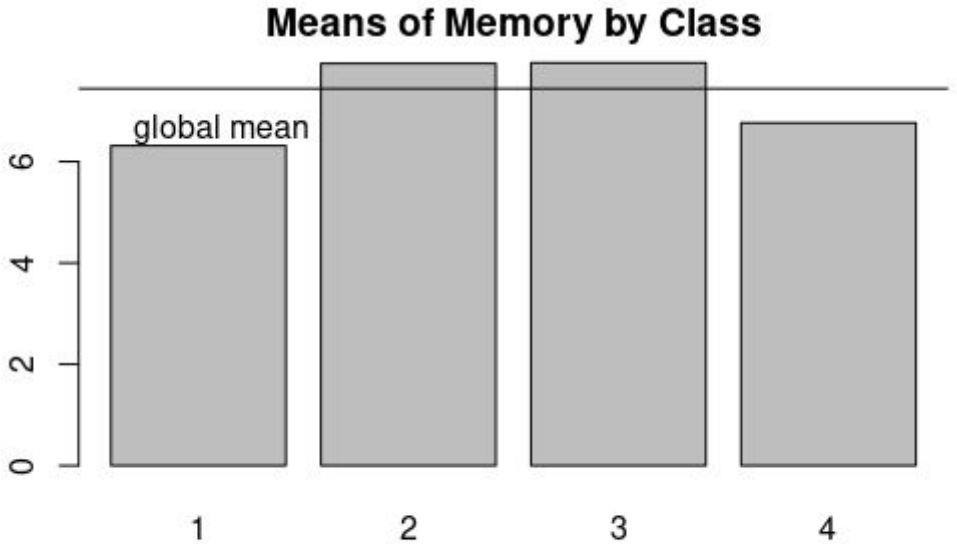
In L2_Cache, the global mean is 6.2 KB. The group 2 mean and group 3 mean is higher than the global mean. If we see the boxplot then we observe two clear outliers.

Seeing the two figures, we conclude that caches L2 ,with bigger size, are in group 3 and, with small caches, are in group 3.



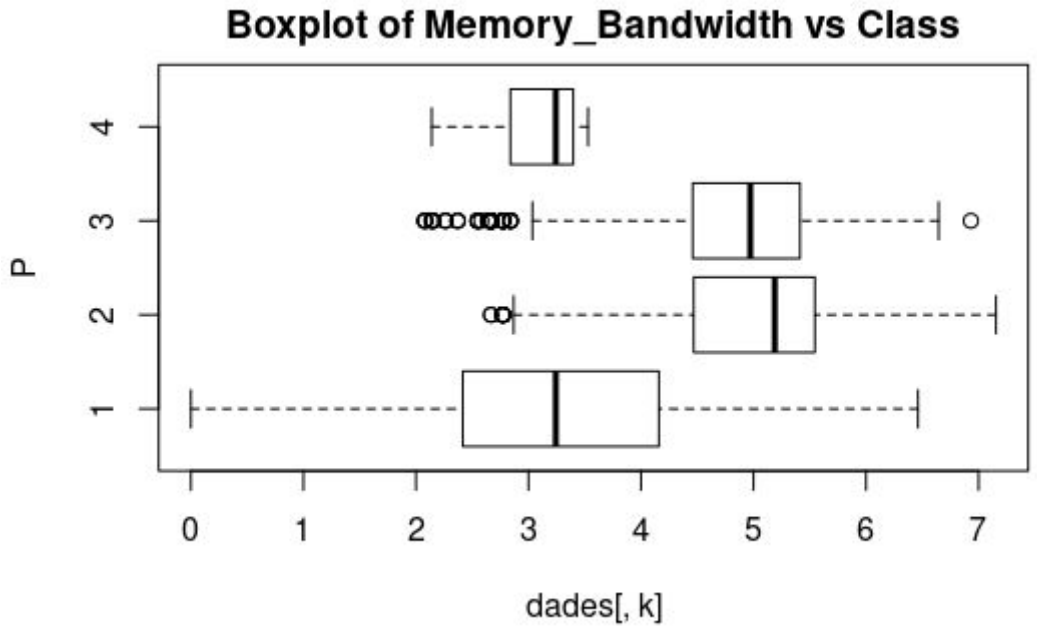
In relation to the Max Power, the group 2 and group 3 are very similar because both have a mean of 5 W approximately. The group 1 is characterized by low power GPUs.

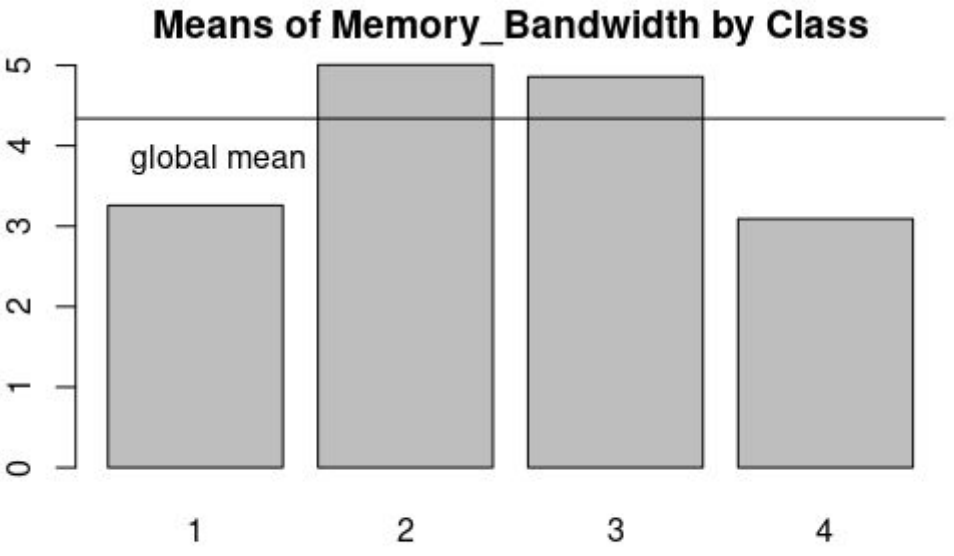




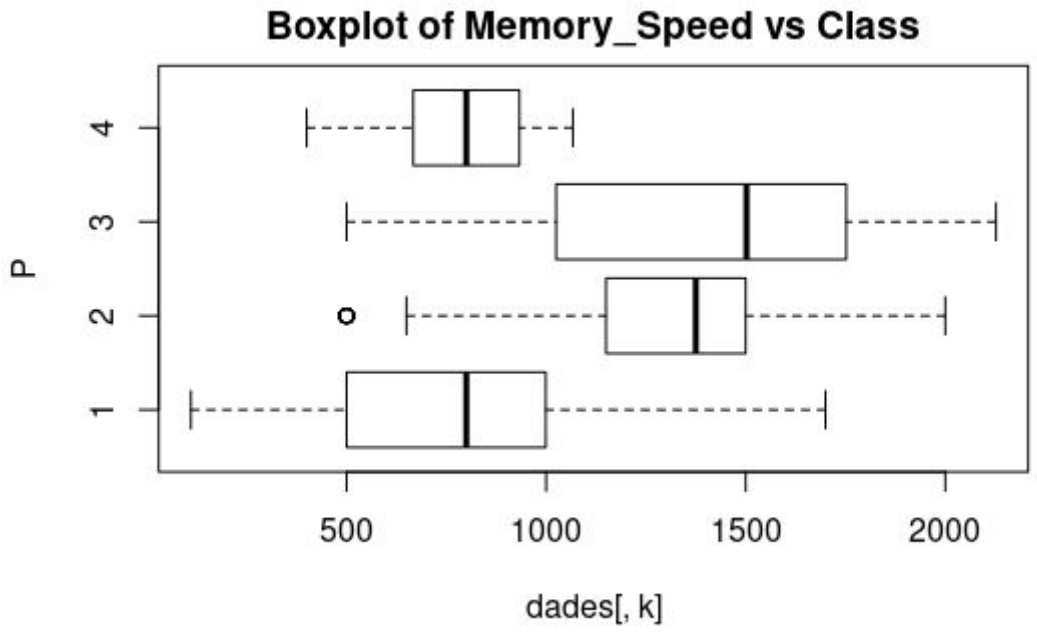
The best memory of graphics cards are in the group 2 and 3 because the mean is placed in 4 MB, the Q1 is 7.625 and the Q3 is 8.318. It seems that the memory of graphics cards in group 2 and group 3 have the same statistical values: mean, Q1 and Q3. It's peculiar the quantity of outliers in each group

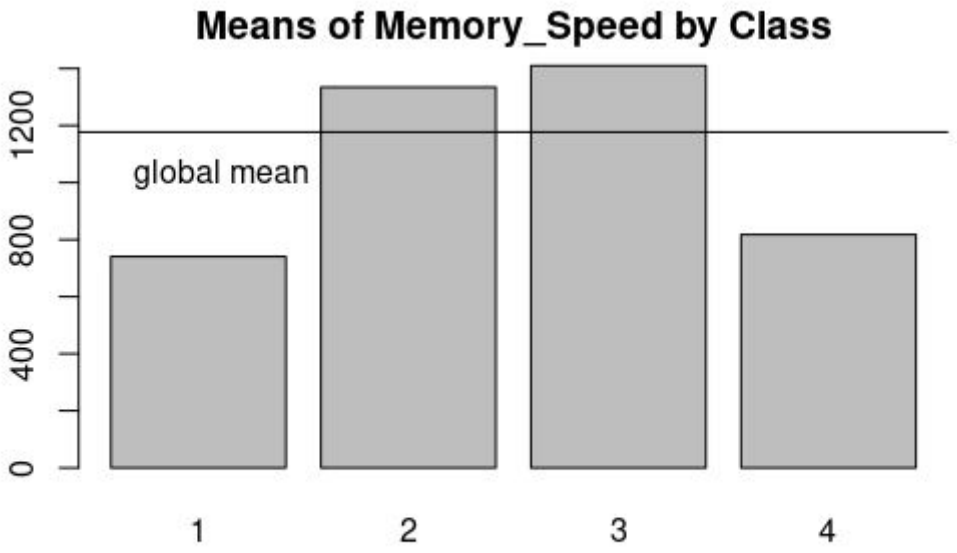
because most of them are above the Q3. As we can see, the means vary near the global mean, it means that the hierarchical clusterings has been performed correctly.



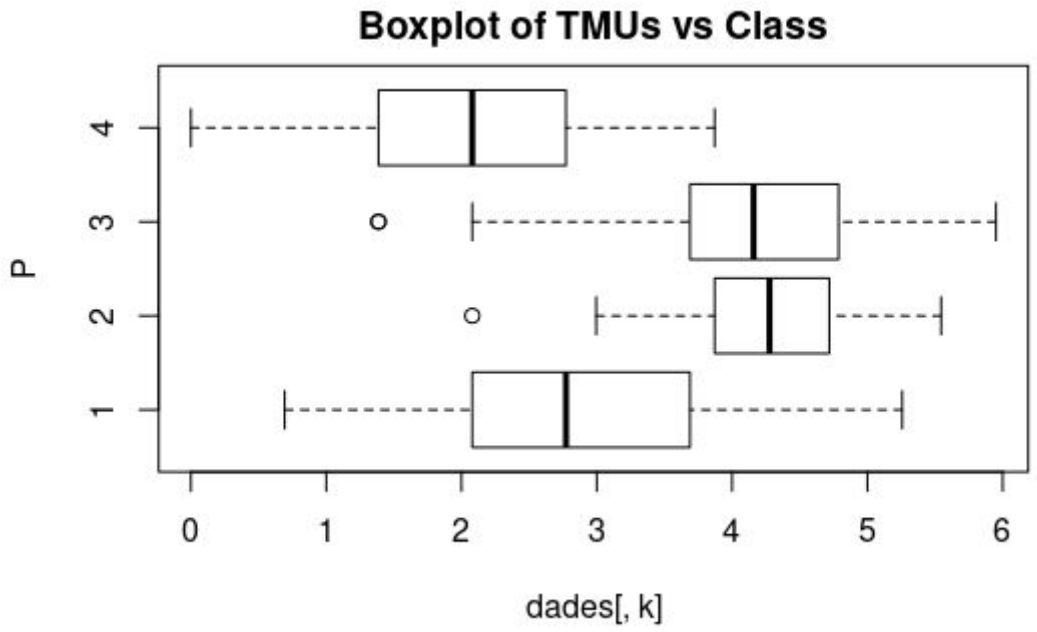


The memory bandwidth's global mean is 4.2 GB/s. The fastest memory in GPUs are in the group 2 and 3. However, we can see that group 2 is better than group 3. Also, it's interesting to say that the group 3 has a big amount of variation, in other words the standard deviation is high.

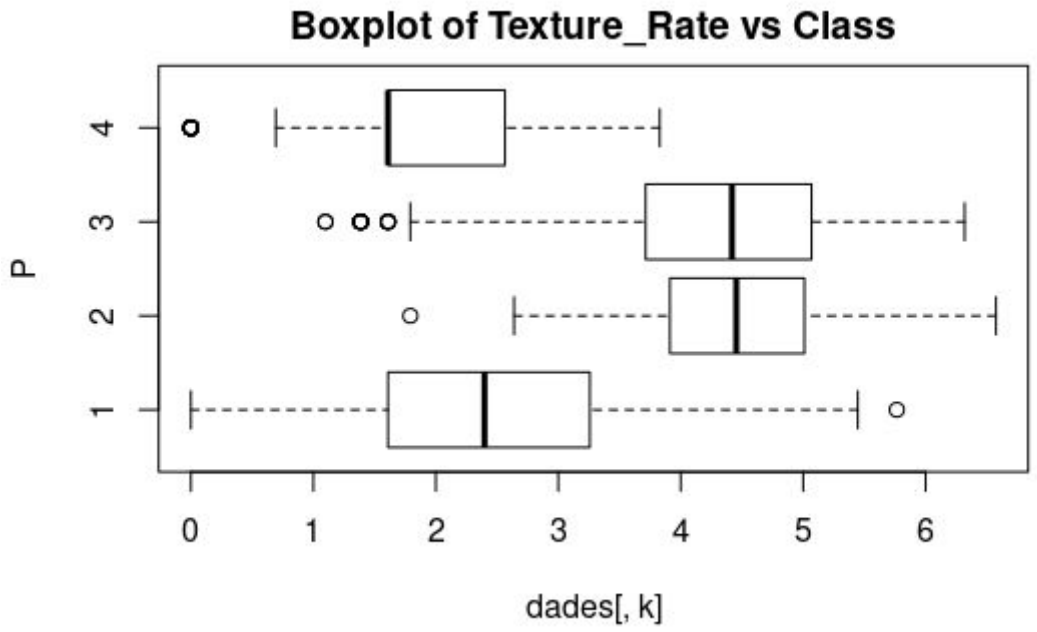

Boxplot of Memory_Speed vs Class
P
4
3
2
1
500
1000
1500
2000
dades[, k]

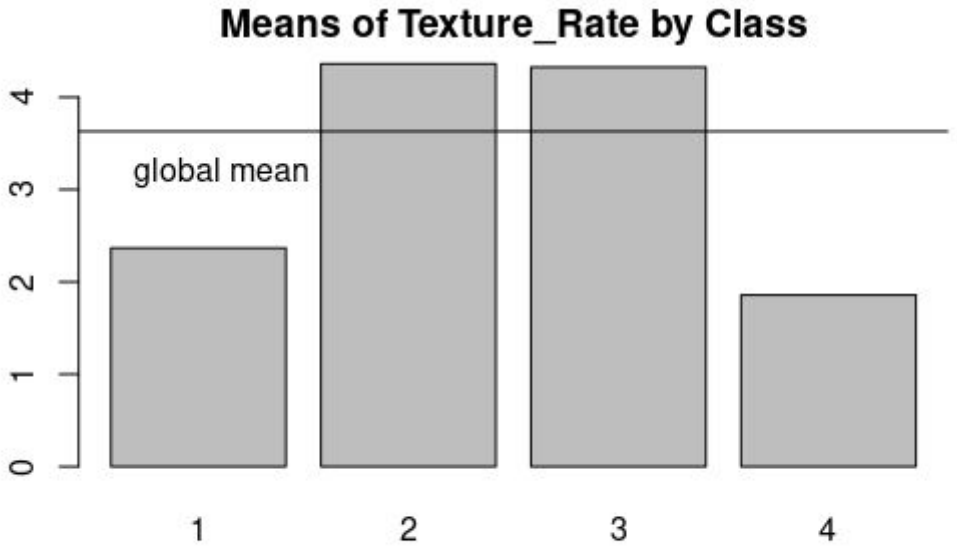


The fastest GPUs are in the group 2 and 3. The global mean is situated in 1200 MHz. In the group 1, the mean is located in 817.7 MHz, the 1-quantile (Q1) is in 667.0 MHz and the 3-quantile (Q3) is in 933.0 MHz. In the group 2, the mean is located in 1410 MHz, the 1-quantile (Q1) is in 1025 MHz and the 3-quantile (Q3) is in 1752 MHz. Group 3, the mean is located in 1334 MHz, the 1-quantile (Q1) is in 1150 MHz and the 3-quantile (Q3) is in 1500 MHz. Group 4, the mean is located in 740.8 MHz, the 1-quantile (Q1) is in 500.0 MHz and the 3-quantile (Q3) is in 999.0 MHz.



These pictures show us the descriptive statistics of each group in relation to the variable MTUs. The global mean is situated in 3.8 Units. As we conclude in the other variables, the group 2 and 3 are the best group. Group 2 and Group 3, with a mean of 1410 Units and a mean 1334 Units respectively, is better than group 1 and 4, whose means are 817.7 Units and 740.8 Units respectively.

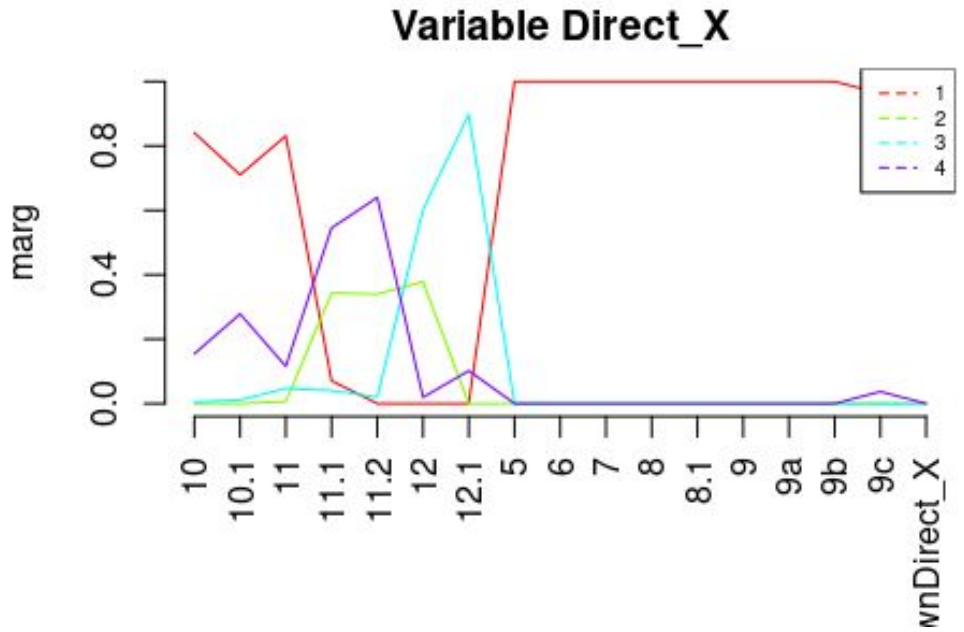




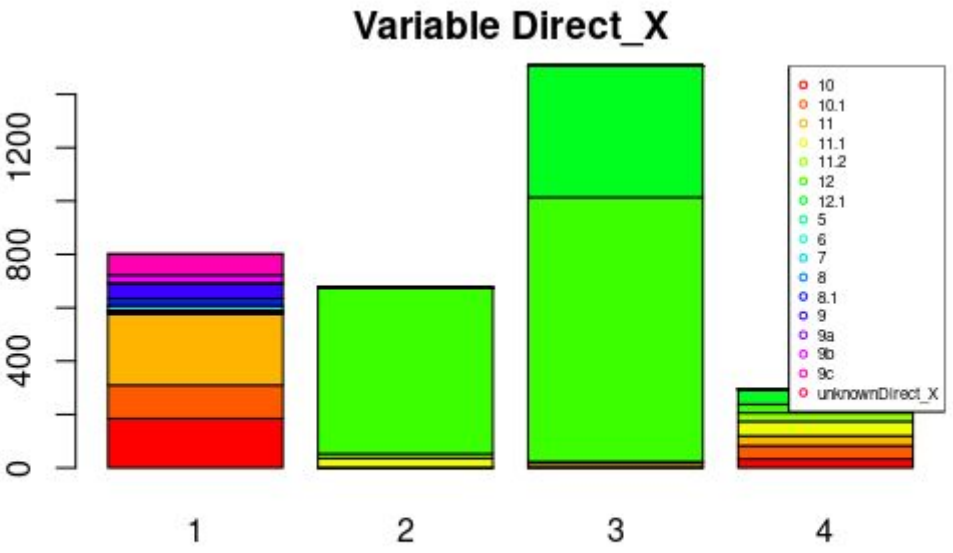
The texture rate's global mean is 3.8 GTexel/s. The fastest GPUs are in the group 2 and 3.

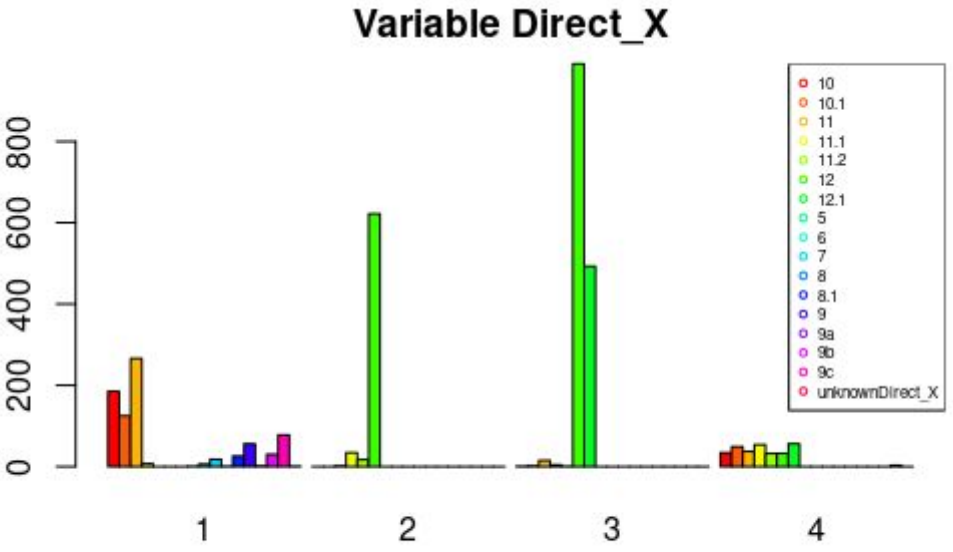
Group 2 has a mean of 4.162 GTexel/s and Group 3 has a mean of 4.225 GTexel/s.

The mean of group 1 is 2.043 GTexel/s, lower than the mean of the group 4, 2.8589 GTexel/s.

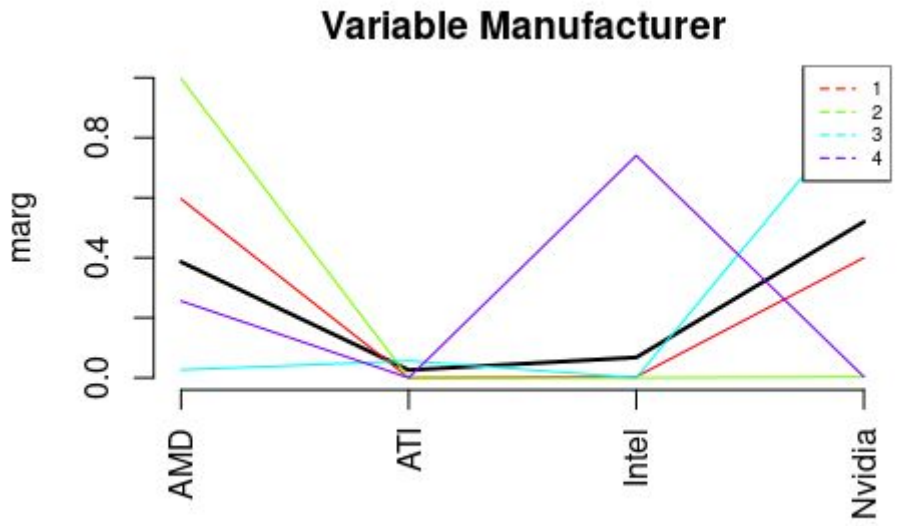


In this picture we can see in the horizontal axis the different versions of Direct X and in the legend the 4 classes. We can see that the lower versions of Direct X are in group 1. We can also see that the higher versions of Direct X are found in group 1 and group 4. The versions between 11 and 11.2 can be found in group 2 and 4. The version 12.1 can be found in group 3.

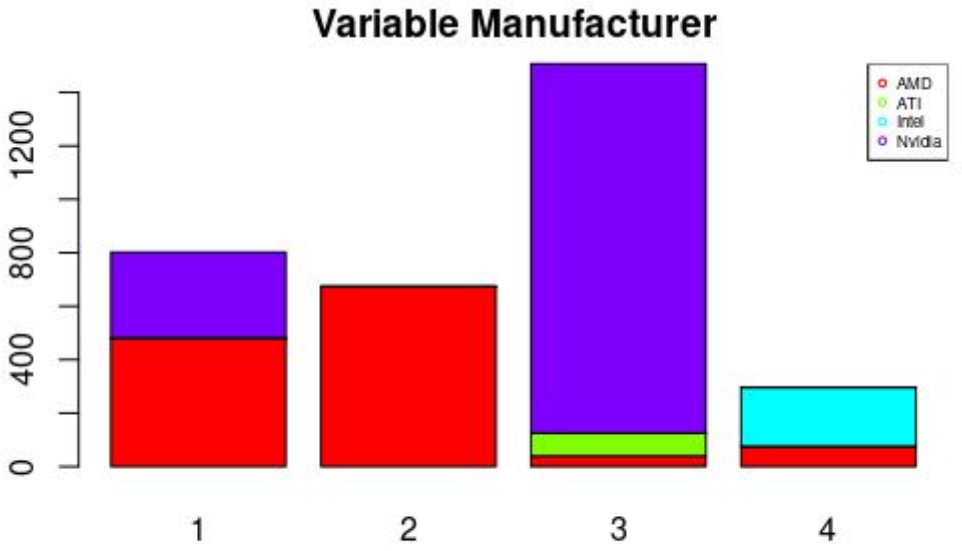




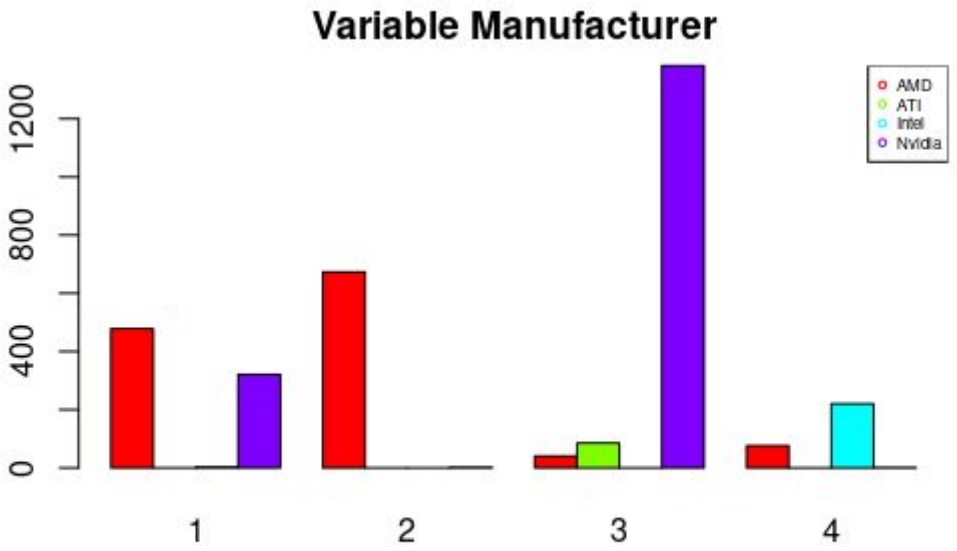
Reaffirming what has been explained, we see in the accumulative barplot and in the normal barplot that Direct X 11.1, with Direct X 11.2, populated the class 3. Most of the Direct X 10, 10.1 and 11 are in the group 1. We could say that group 1 are the lower versions and in group 2 and group 3 are the higher versions of Direct X.



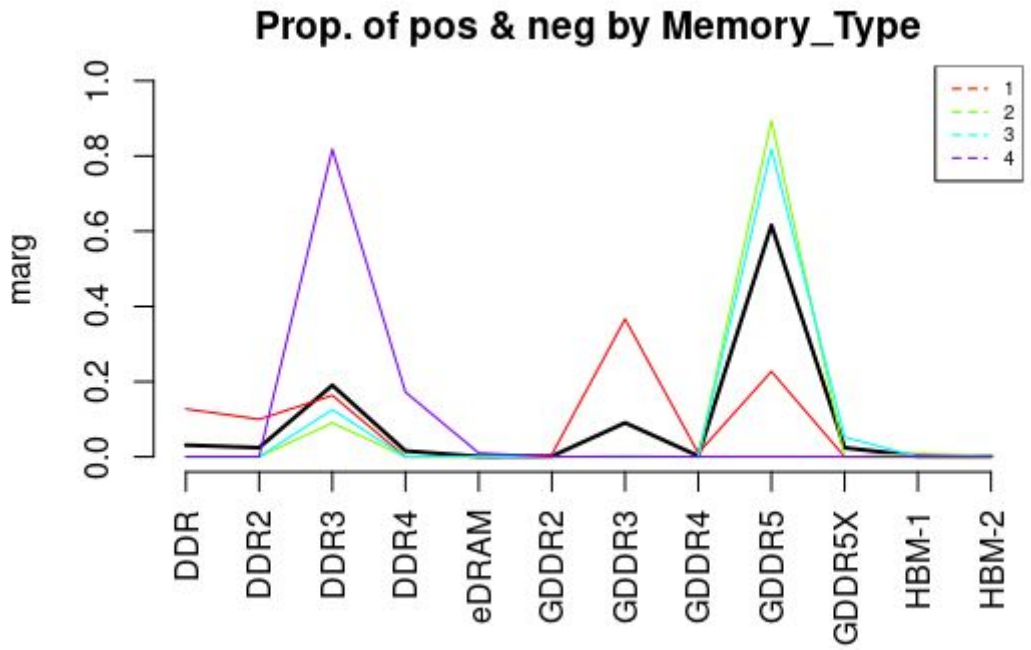
In relation to the companies of GPUs, we would say that a high percentage of the group 4 is constituted by Intel. All the members of group 2 are AMD GPUs. The group 1 is formed by Nvidia GPUs and AMD GPUs. Group 3 is formed mostly by Nvidia GPUs, however we can see some ATI GPUs and AMD GPUs.



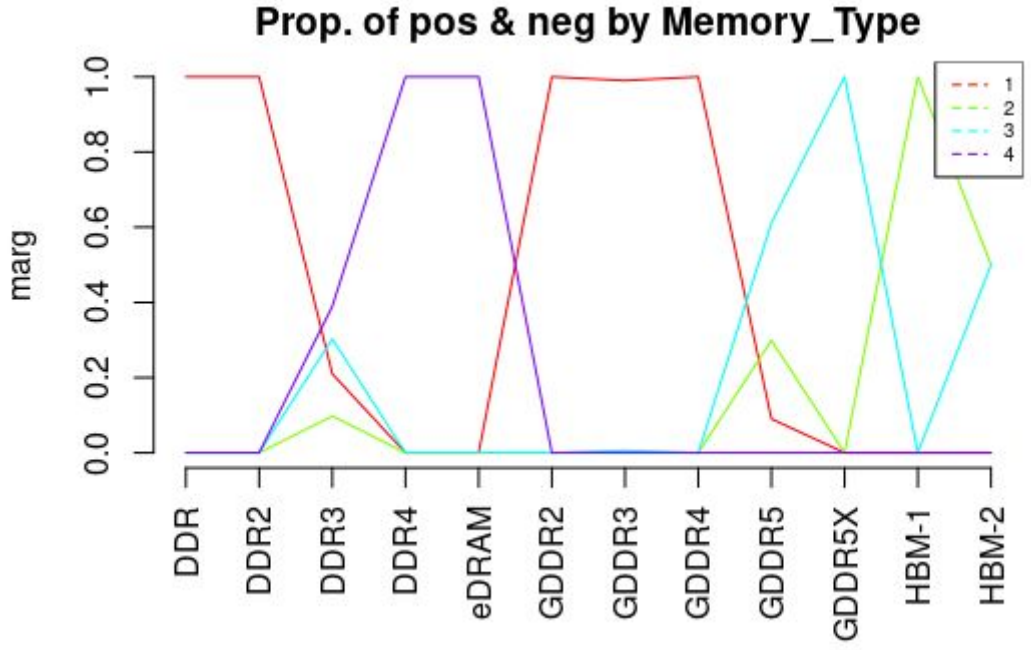
In this figure, like in the other ones, we can see the amount of members in each group. The most populated group is the number 3. The second group most populated is number 1. The groups with fewer members are group 2 and 4.

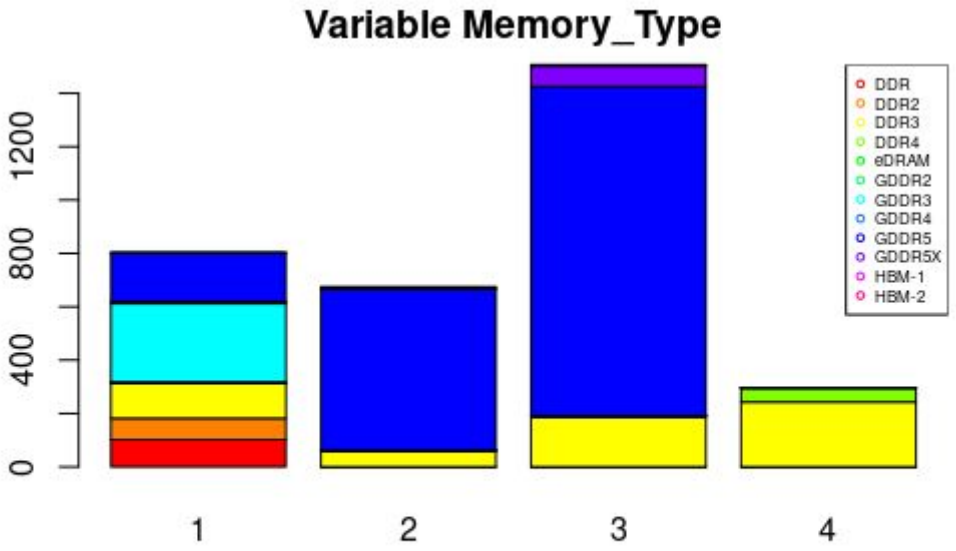


All we have explained about the line chart, can be seen in the barplots. Nvidia is found in group 3 and 1. ATI only is in group 3. Intel is in the lower group in relation to speed, group 4. AMD can be found in all of the groups, but it stands out in group 1 and group 2.

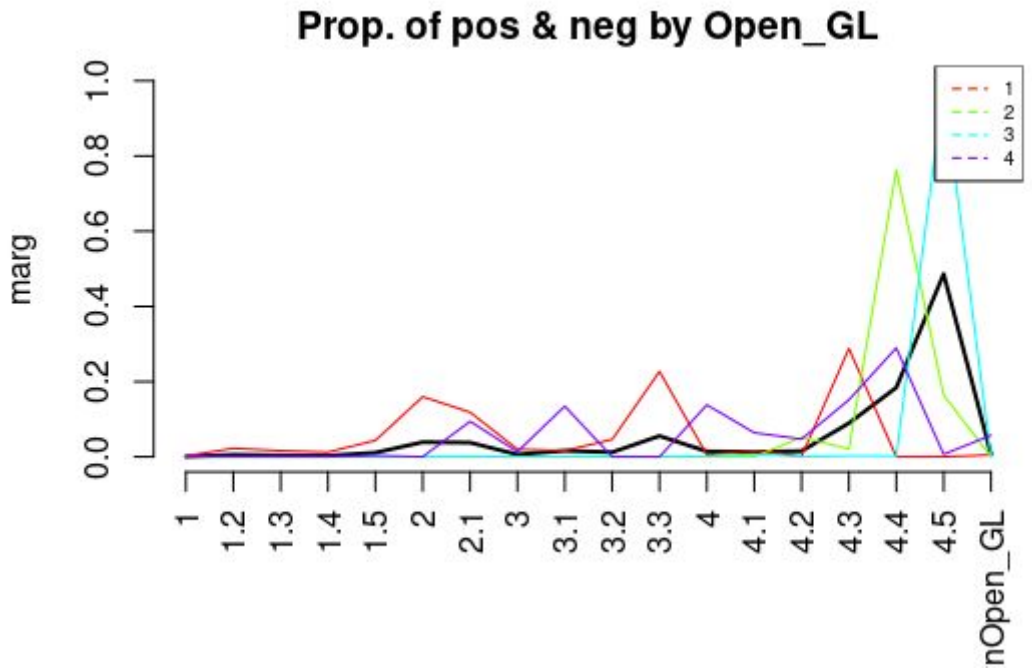


As we can observe in this plot, 81% of the group 4 is constituted by DDR3 and the other 19% is constituted by DDR4. In relation of group 3 and 2, high percentage of the members are GDDR5 and the rest of the members are DDR3. The group 1 is formed by GDDR3, GDDR5, DDR3 and DDR.

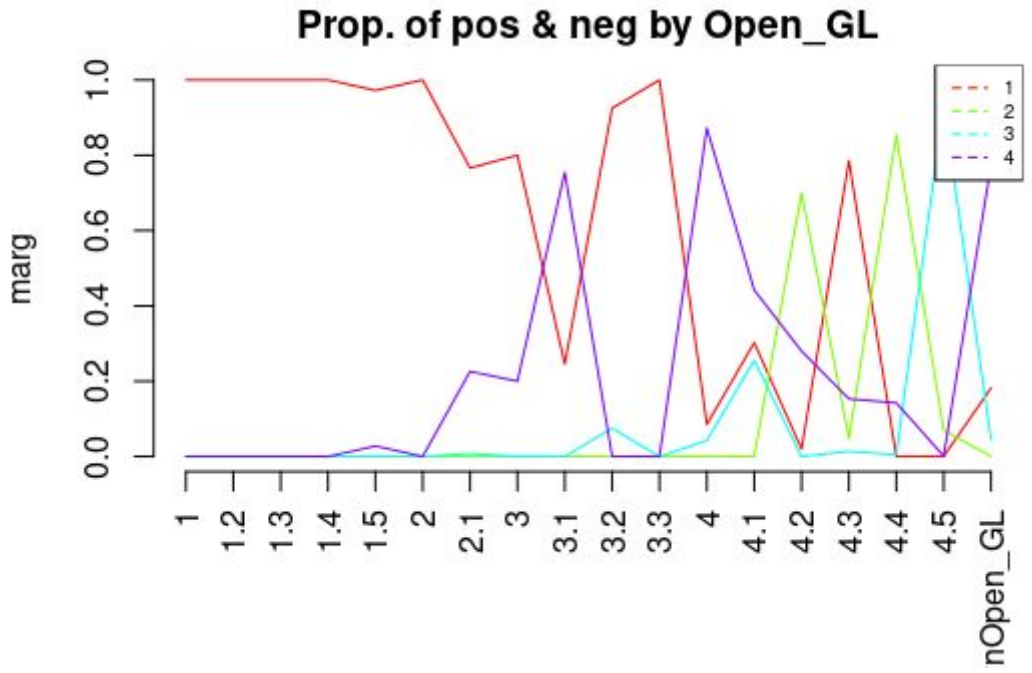




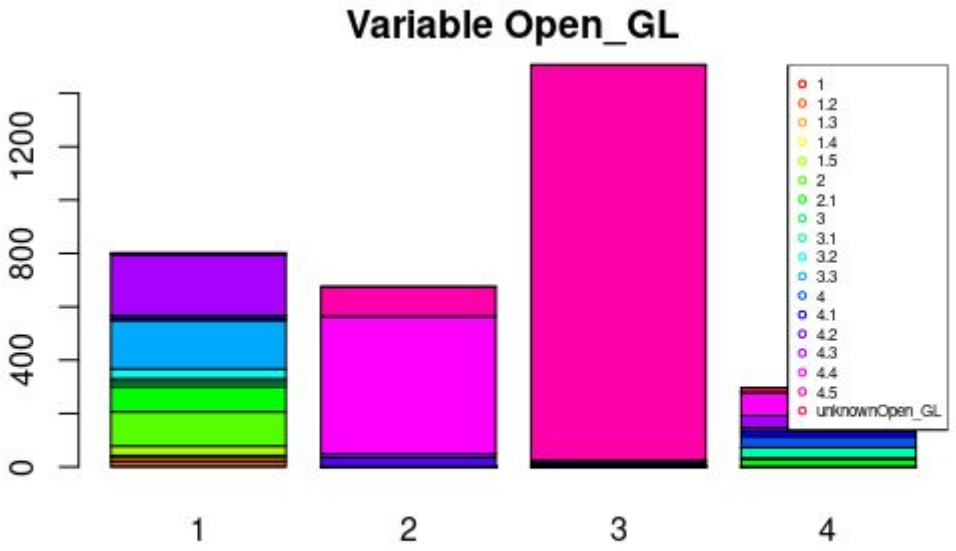
These plots show us the proportion of each Memory type in each class. We see that DDR, GDDR2, GDDR3 and GDDR4 are in the group 1. DDR4 and eDRAM are in the group 4. DDR3 and GDDR5 are in the group 2 and 3. GDDR5X is in the group 1 and HBM-1 is in the group 2.

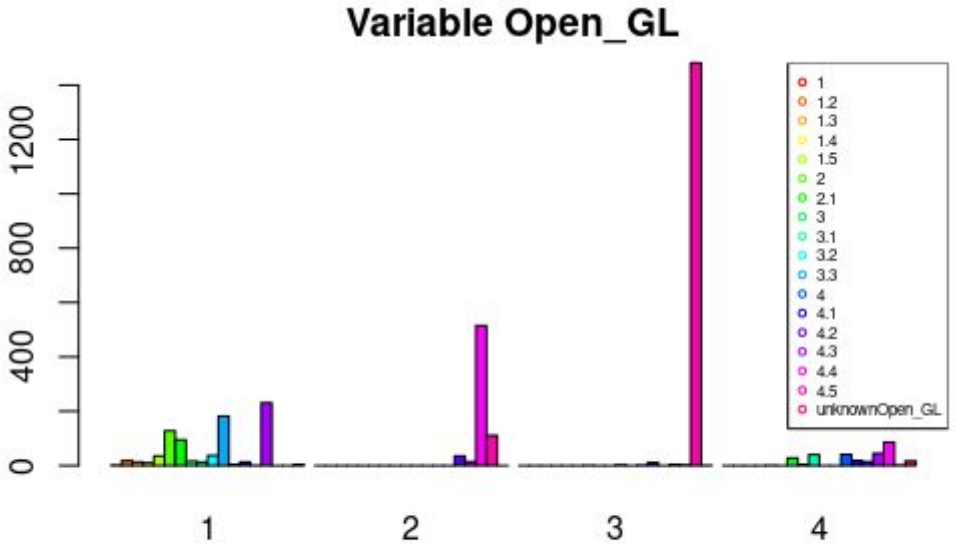


In this line plot we can recognize that groups 2 and 3 are formed by the high versions of OpenGL. Group 1 is formed principally by 3 versions of OpenGL: OpenGL 2, OpenGL 3.3 and OpenGL 4.3. Open GL 2.1, 3.1, 4 and 4.4 construct the group 1.

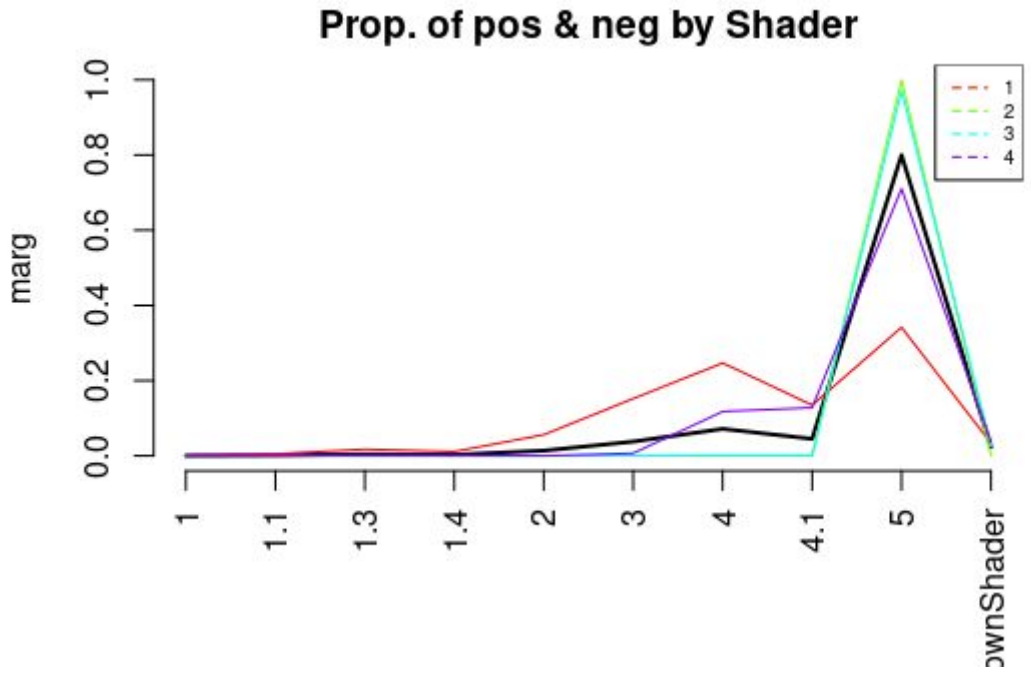


The low versions of OpenGL are in the class 1. OpenGL 3.2, 3.3 and 4.3 are also in group 4. The high versions of OpenGL are in the group 2 and 3.

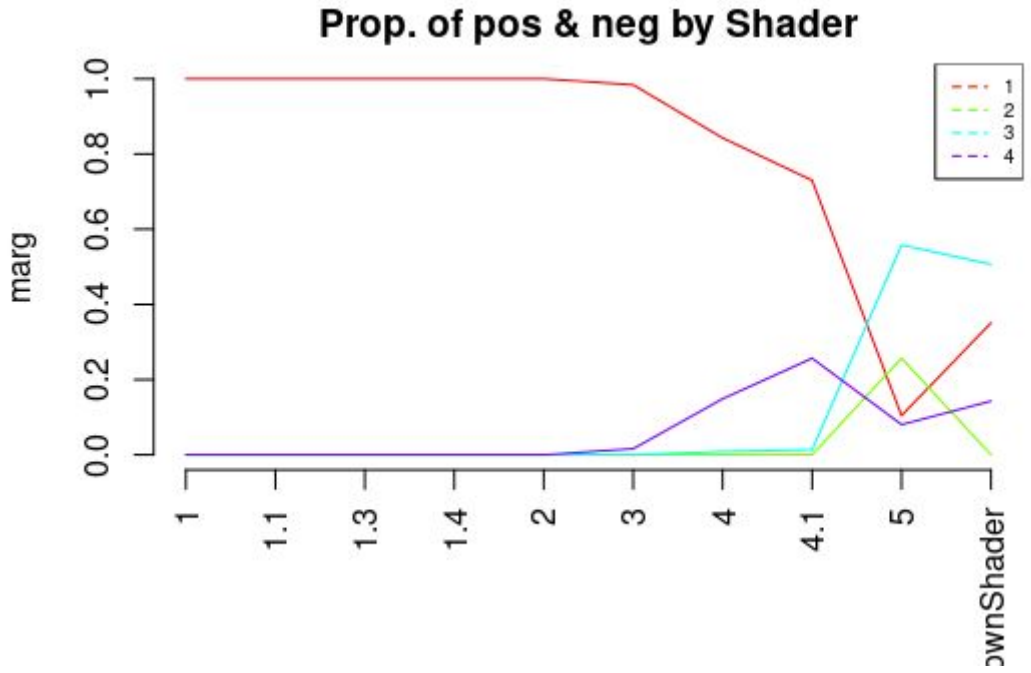




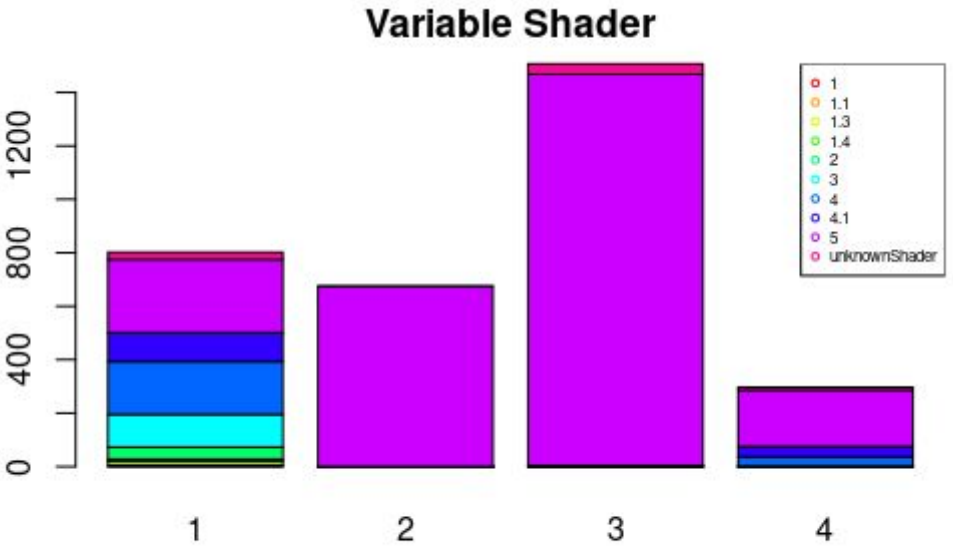
All we have seen in the previous linear plots, can be seen in the barplots. Groups 2 and 3 are formed by higher versions of OpenGL. Group 1 is formed principally by 3 versions of OpenGL: OpenGL 2, OpenGL 3.3 and OpenGL 4.3. Open GL 2.1, 3.1, 4 and 4.4 construct the group 1.

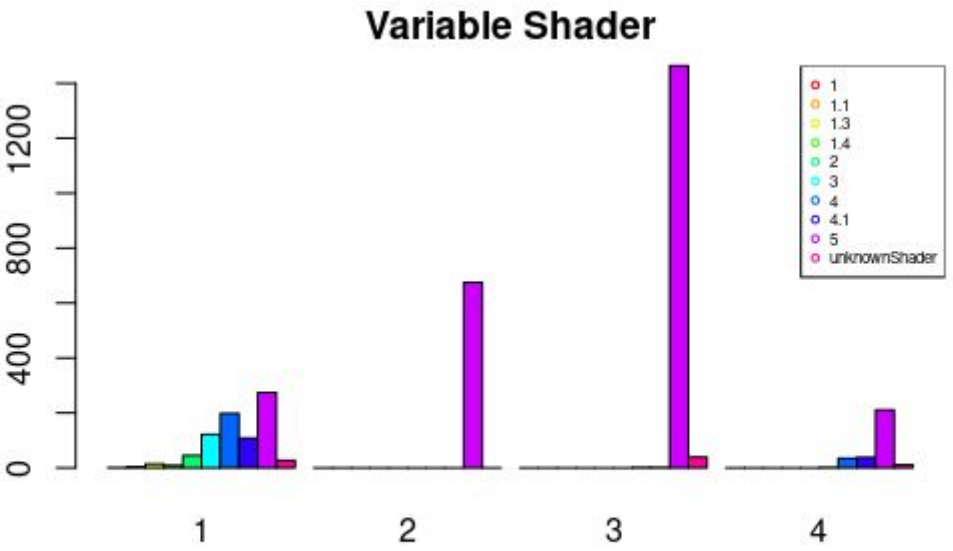


In this line plot we can see the contribution of each Shader Model in the classes. We see that more than the 90% of the group 2, 3, 4 is formed by Shader 5. All the shaders contribute to the generation of class 1.

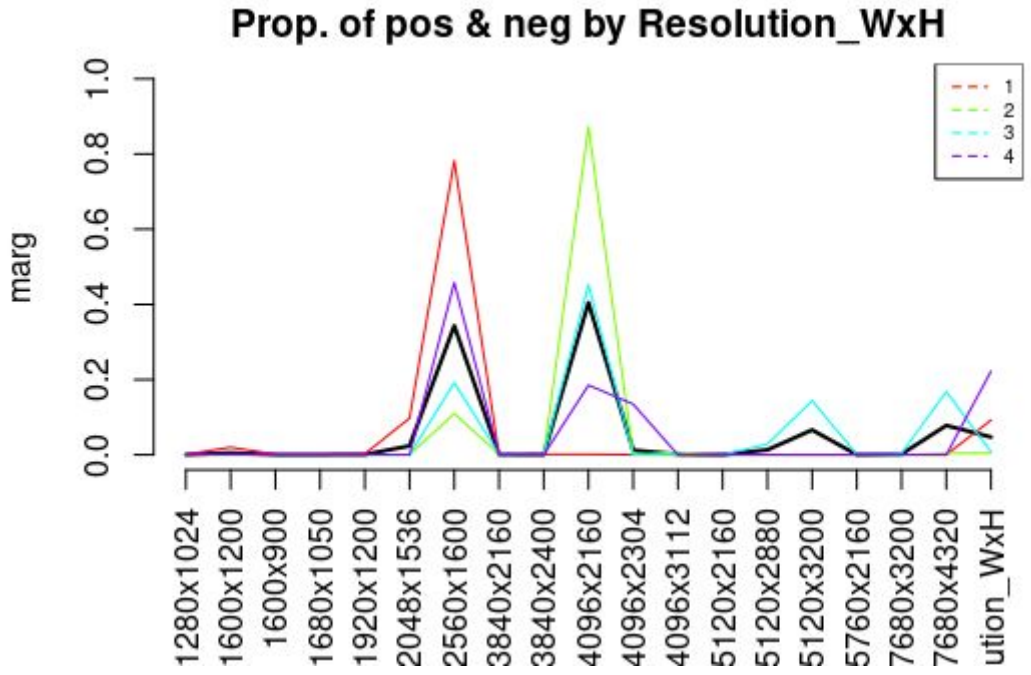


Versions lower than 2 are only in the group 1. Versions between 3 and 4.1 are mostly in class 1 and 4. The Shader model 5 and unknown are found in all the classes.

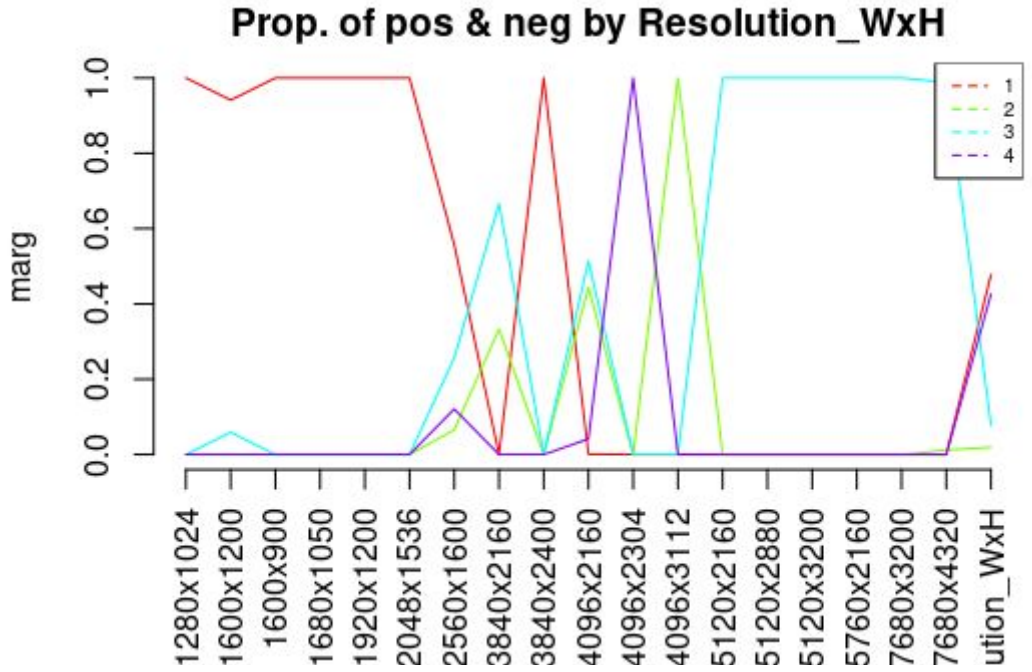

Variable Shader
0
400
800
1200
1
2
3
4
1
1.1
1.3
1.4
2
3
4
4.1
5
unknownShader



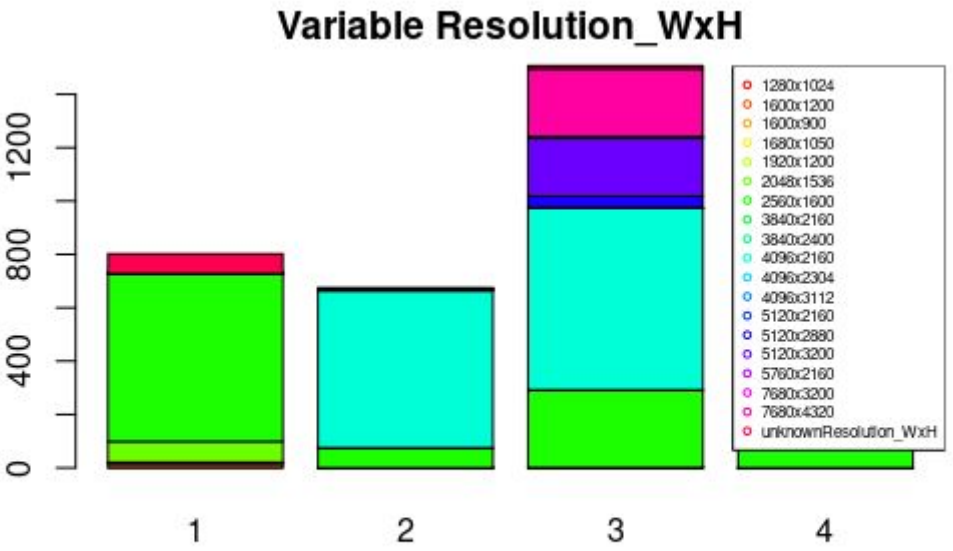
As we can see, there are a lot of GPUs with Shader model 5. In class 1 we can find all the versions, probably the worst versions because in class 1 we can find the lowest GPUs.

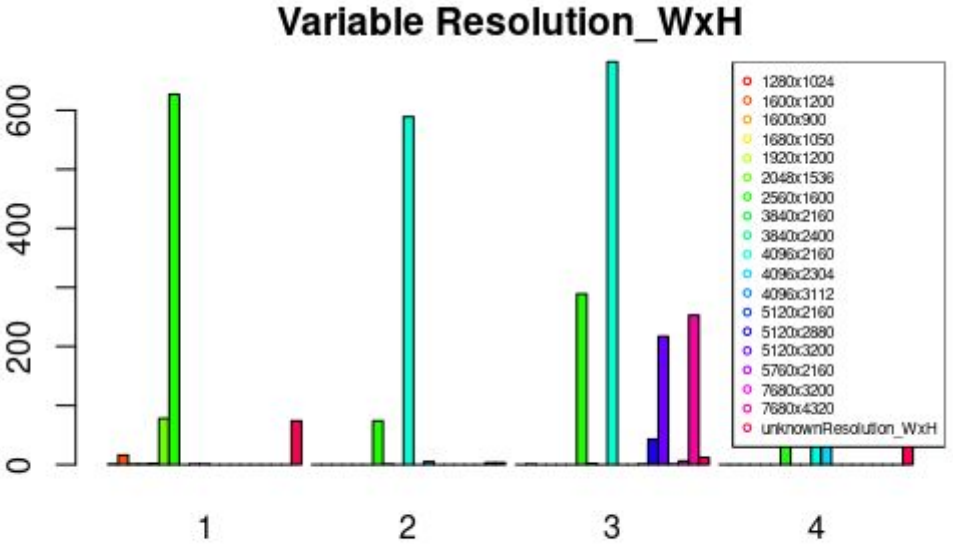


It's important to explain what resolution we can find in each group. We can see that the GPUs which support a resolution of 2560x1600 can be found in all the groups. Also, we can observe that GPUs which support a resolution of 4096x2160 contribute a lot in the classes 2, 3 and 4. The other resolutions can be found in all the classes, however they don't contribute a lot.

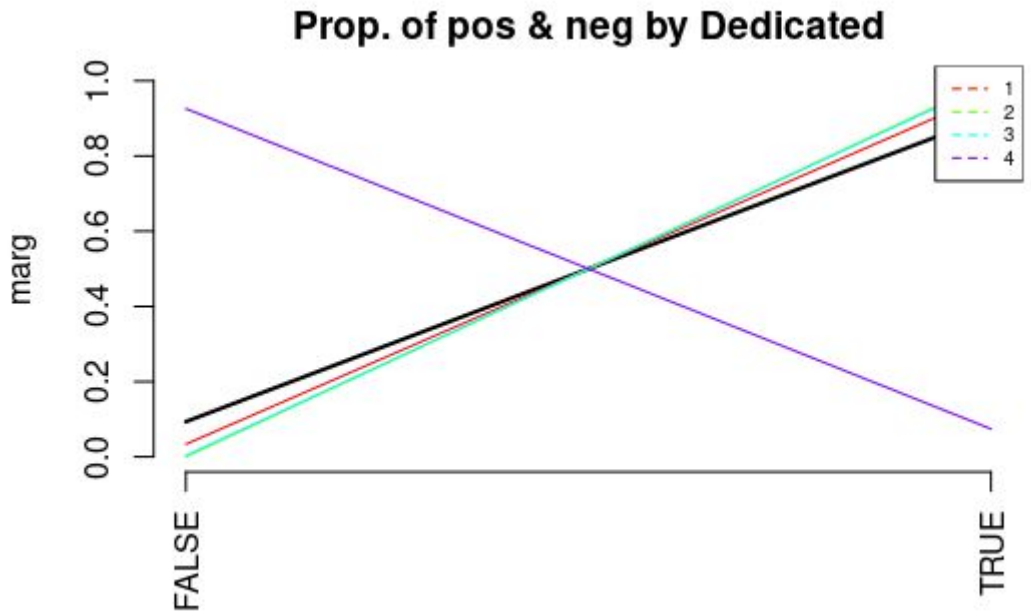


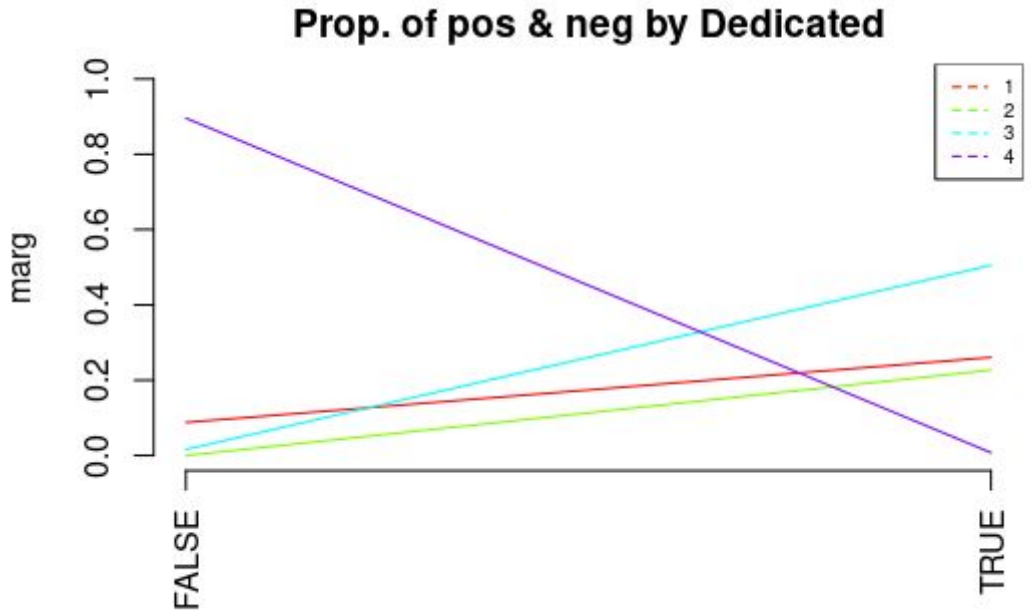
We can see that the lower resolutions are found in the class 1. The higher resolutions are found in the class 3. The resolution between 2048x1535 and 4096x2160 are found proportionally in all the classes.



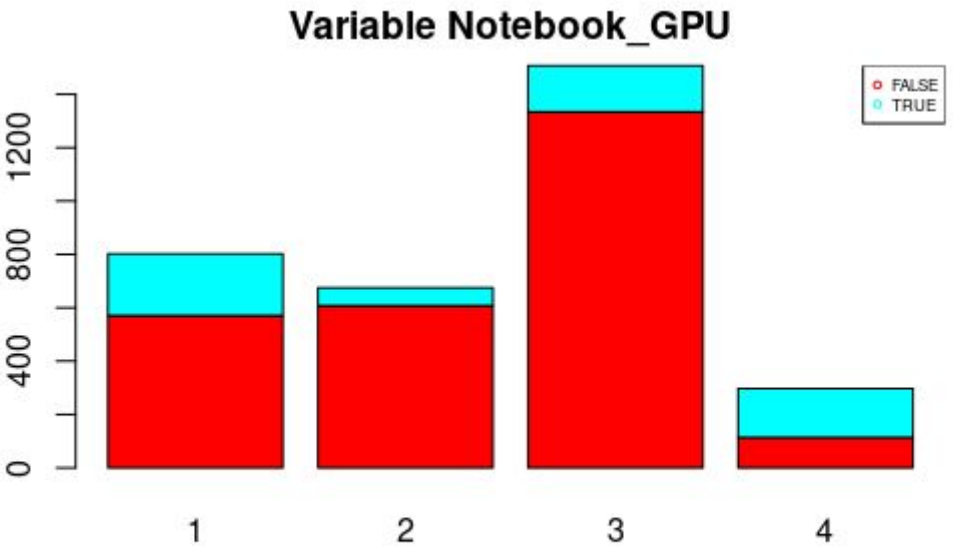


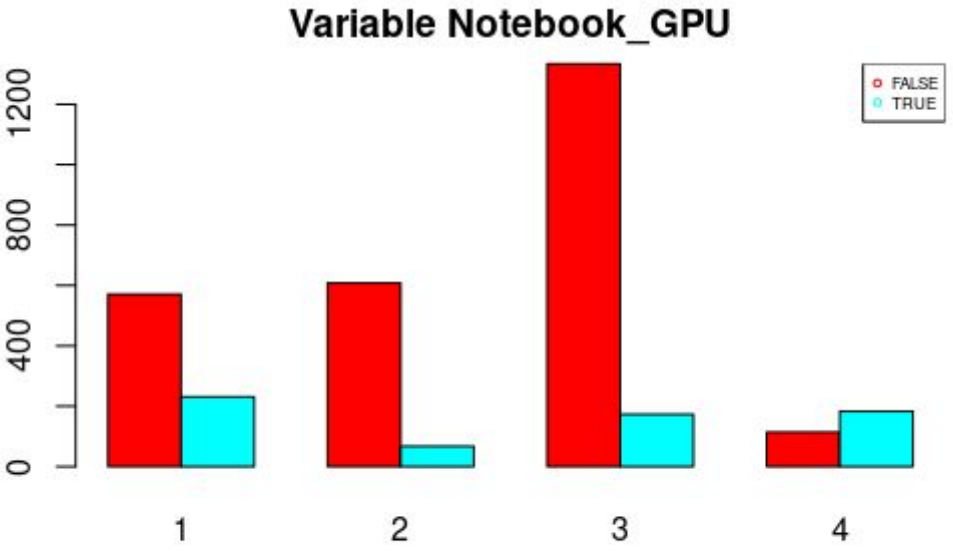
In addition, in the bar plots we can see the amount of resolutions in each class. In cluster 3 we can find a huge variety of resolutions, but specially highlight the resolution 4096x2160.



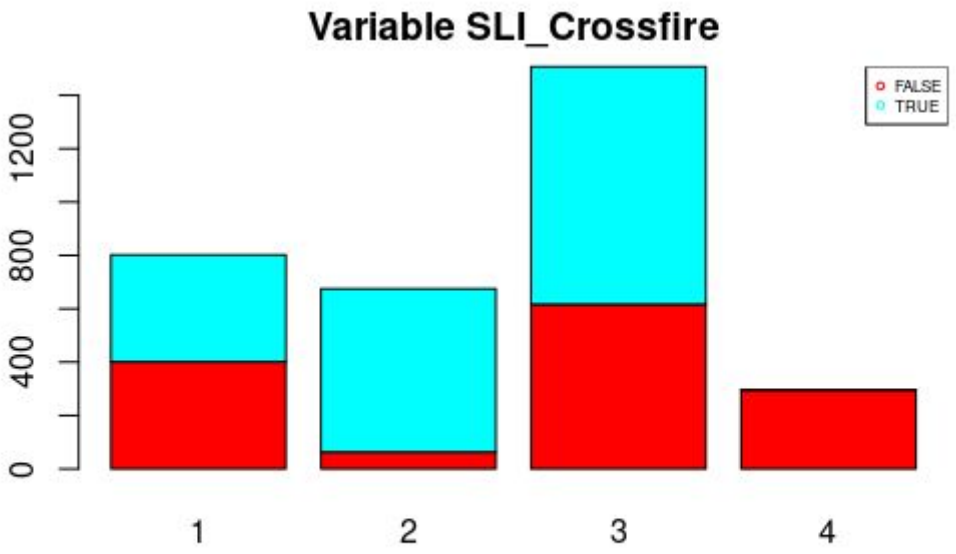


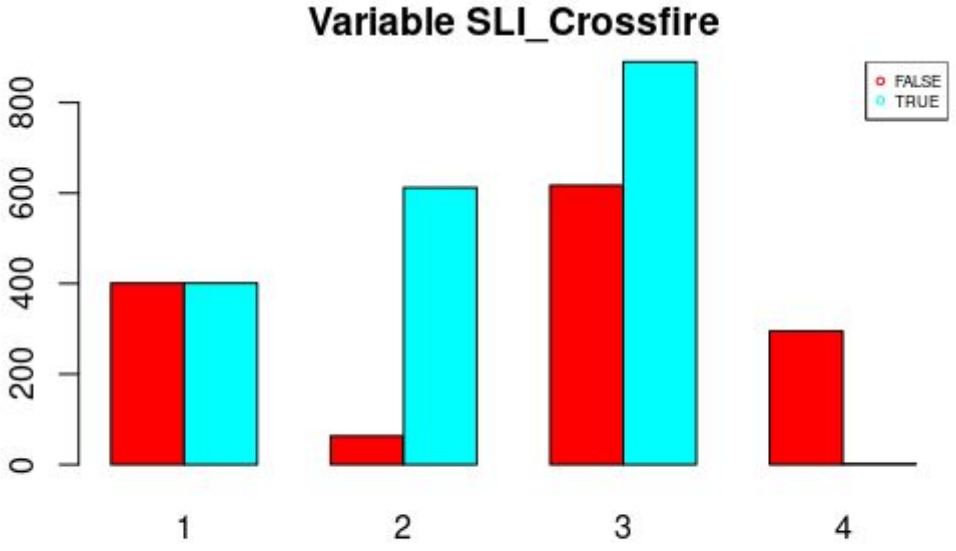
In regard to the variable Dedicated, we can see that hierarchical clustering has joined all the GPUs with no graphic card dedicated in class 4 and the GPUs with graphic cards in the other clusters.



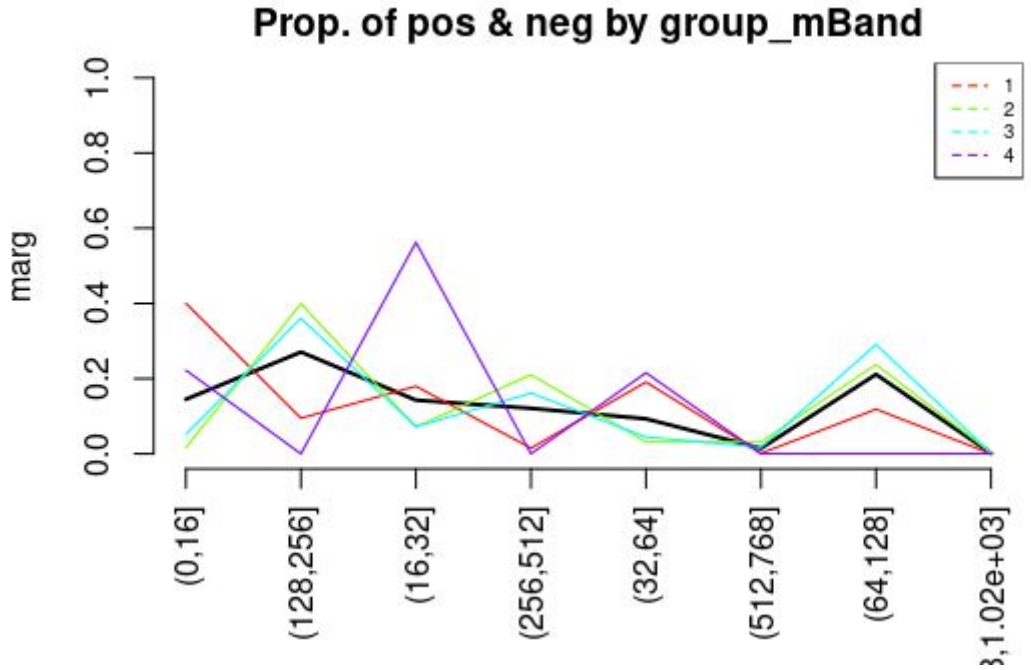


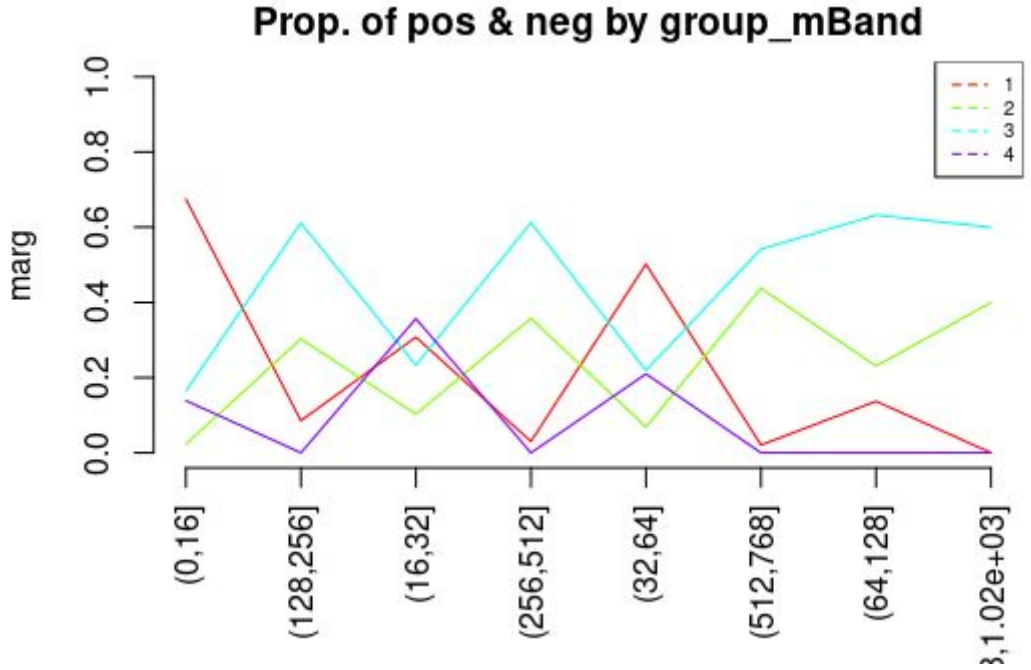
There is a relationship between the binary variable Notebook and the clusters. We can see that in cluster 4 there are more notebook GPUs than desktop GPUs. On the other hand, the clusters from 1 to 3 predominate the desktop GPUs.



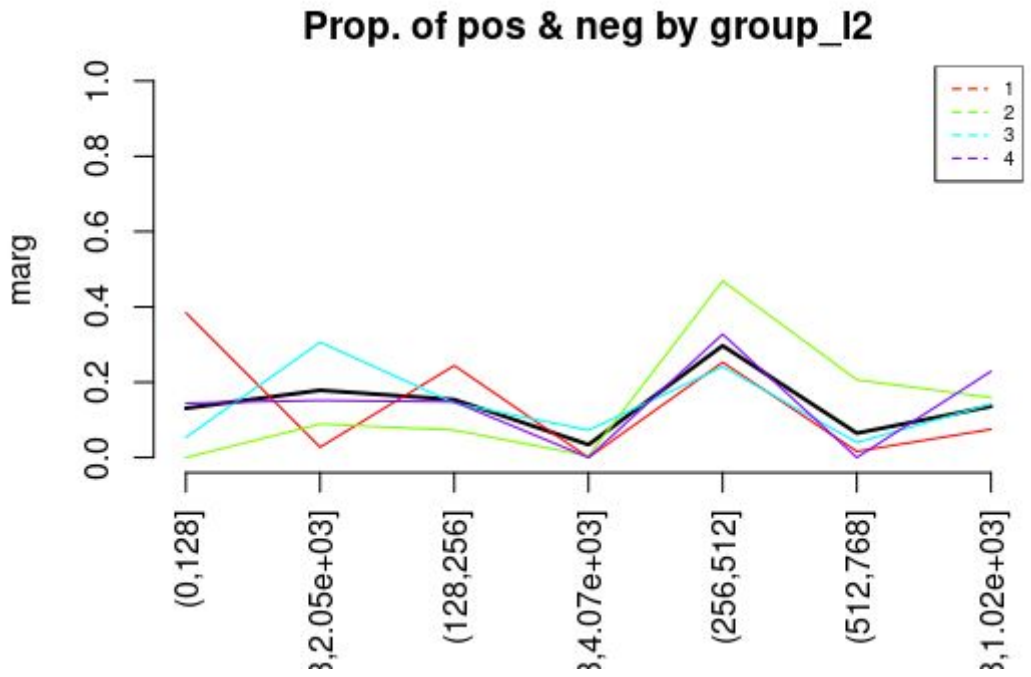


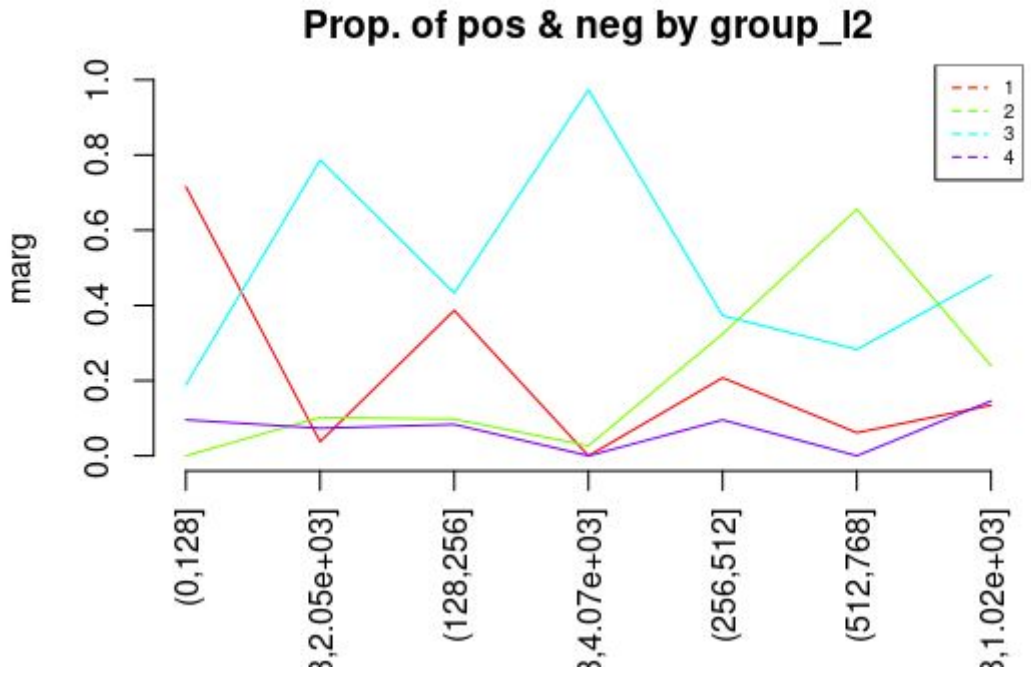
In cluster 1, the quantity of GPUs with Crossfire technology is equal to the quantity of GPUs with SLI. In cluster 2 and 3, there are more GPUs with SLI than GPUs with Crossfire technology. In cluster 4, highlight cluster with Crossfire Technology.


Prop. of pos & neg by group_mBand
marg
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
(0,16]
(128,256]
(16,32]
(256,512]
(32,64]
(512,768]
(64,128]

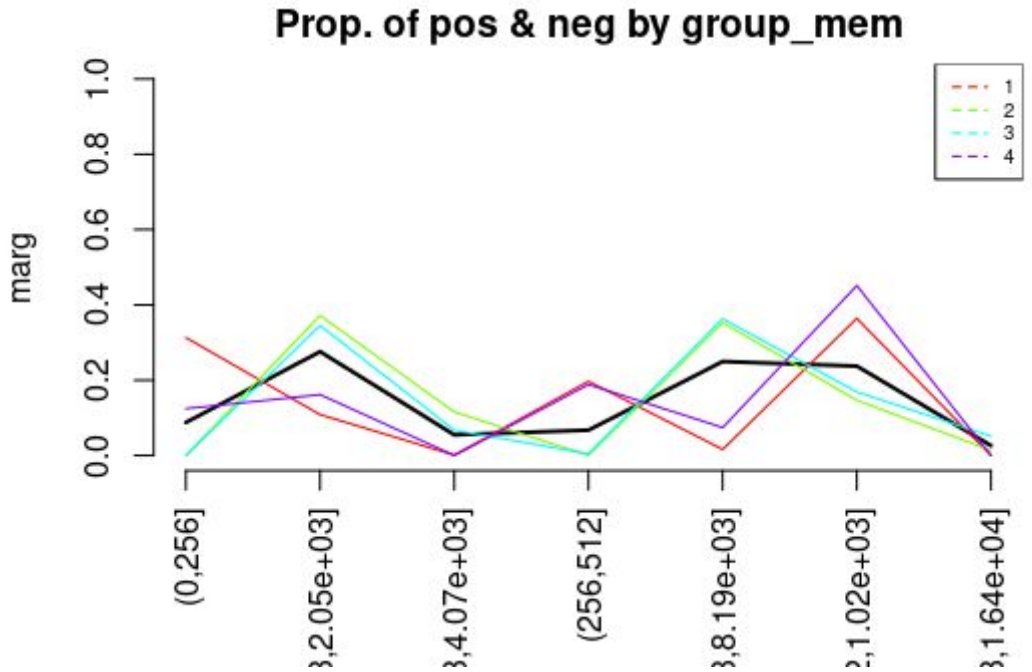

Prop. of pos & neg by group_mBand
marg
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
(0,16]
(128,256]
(16,32]
(256,512]
(32,64]
(512,768]
(64,128]

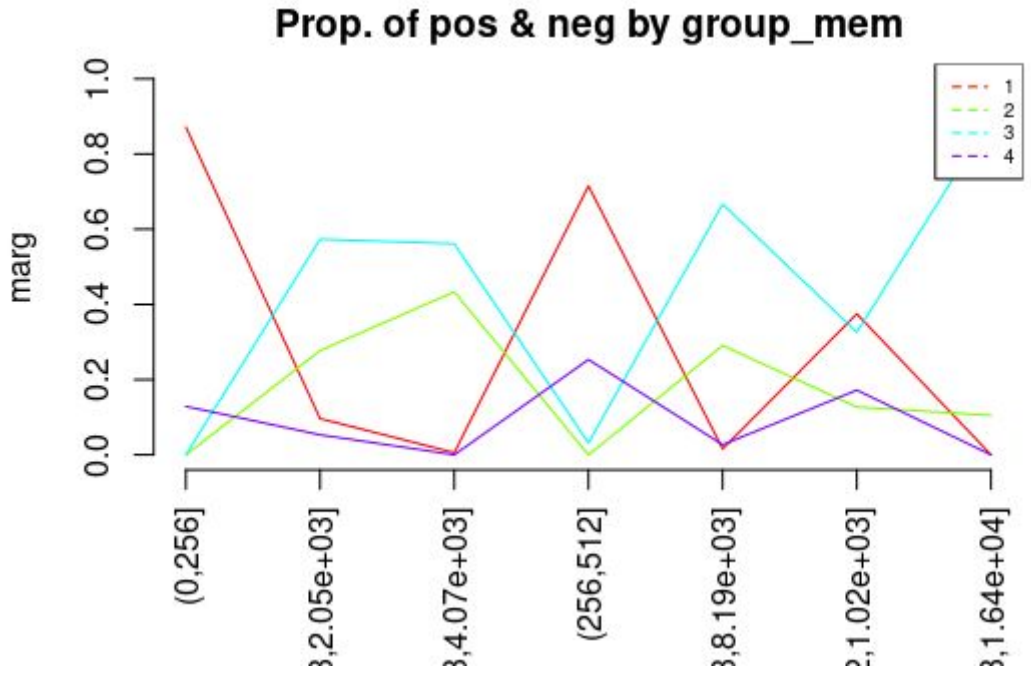
These plots show the discretization of variable Memory_Bandwidth. The GPUs with (16,23] GB/s describe basically the group 4- The GPUs with (128,256], (256,32] and (64,128] GB/s describe particularly the cluster 3 and 2. The GPUs with low bandwidth form the cluster 1.



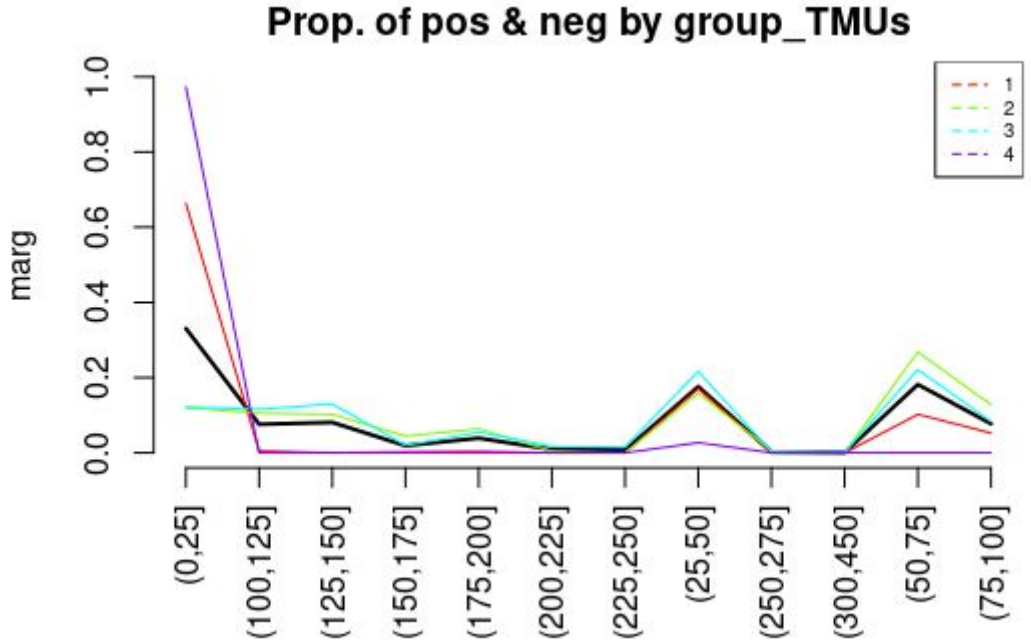


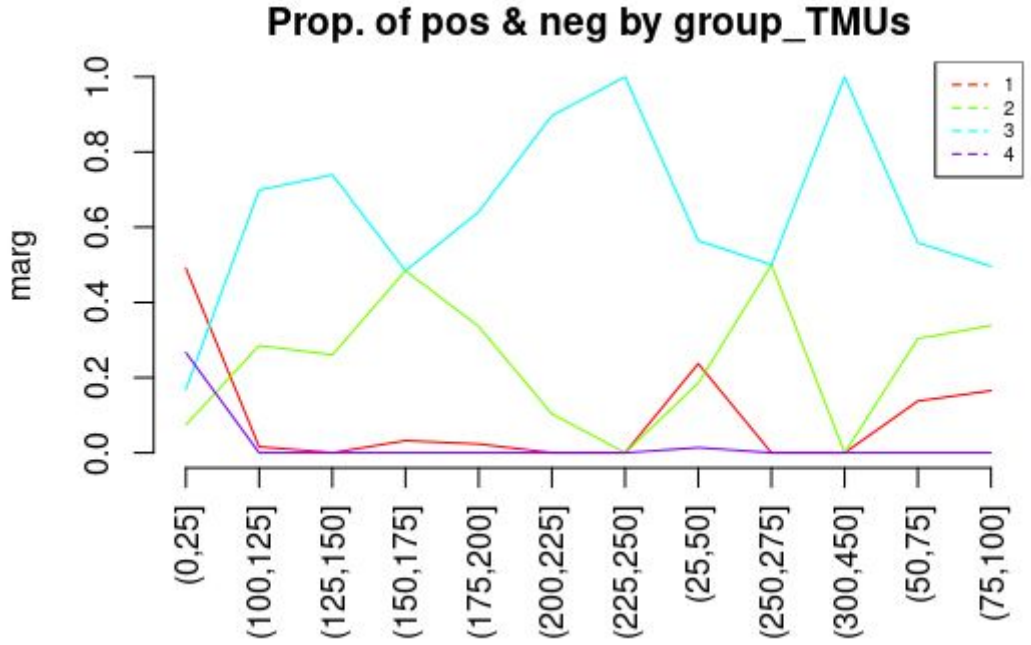
The GPUs, whose L2 Cache is between 256 and 512 KB, predominate in all the clusters. Moreover, 30% of cluster 3 is formed by GPUs with L2 Cache between 102 and 205 KB.





In the above pictures we can see the discretization of the variable Memory. (0,256],(245,512] and (512, 1020] are primarily in the cluster 1. Both cluster, 2 and 3, are formed by the CPUs with memory between 4070 and 8190 MB.





As we can see in the discretization of TMUs, 97% of the group 4 is formed by the attribute (0,25]. The 70% of the group 1 is formed by (0,25] and the rest is formed by (25,50] and (50,75]. The group 2 and 3 is formed by all the groups.

10.1. Selecting Relevant Variables

After analyzing each plot in the previous section we are ready to make some conclusions after visualizing the plots. And say which are the most important variables among all the clusters, that have decided the final group for each cluster.

These variables are **Core_Speed**, **Memory_Speed**, **Texture_Rate**, **TMUs**, **DirectX**, **Dedicated**, **Resolution WxH** and finally **Memory_Type**.

That's because as we have seen in the previous plots, first of all if we talk about the Core_Speed we can see thanks to the ANOVA test, where we can see a huge difference between the clusters 1 and 4, compared to the clusters 2 and 3, where it's clear that clusters 2 and 3 have a huge core_speed compared to the other clusters. This conclusion is supported by the Memory_Speed variable which is telling us again about that difference between these clusters, which seems to be the achieved performance (this happens again with Texture_Rate and TMUs variables). However this results indicates that our implementation of the Hierarchical Clustering it's fine since we have to remember that in our Metadata Matrix we indicated that

Core Clock Speed * TMUs = Texture Fill Rate

Therefore, it's a good signal that Hierarchical Clustering has not only separated into different groups depending on the performance achieved that also has perceived the relationship between these variables. Then when we have to talk about the variable Direct X and its importance we can see how its versions has been divided in order to reserve the newest ones only for the Cluster 2 and 3, meanwhile the cluster 4 has a mix of versions, but it's clear how in the Cluster 1 we have the older versions of this Driver, which support the previous versions made with Core_Speed and others.

Now when we have to analyze the Dedicated variable, we found a lot of useful information, since we can see how only the cluster 4 has no dedicated GPUs or the most of them are not Dedicated, which allows us to interpret the group 1 and 4 as the clusters that comprend the lower performance, despite the cluster 4 are only integrated graphic cards, which would mean that group 1 GPUs are very old or very bad.

After that when we talk about the Resolution WxH we can see the previous separation between clusters 1,4 and 2,3 but now we can see finally how it separates the group 2,3 the most powerful graphics into another category of power, since we can see how only the cluster 3 has the capacity to achieve the higher resolutions meanwhile the cluster 2 has a limit that can not surpass.

And finally if we see the Memory Type we will see that is the one that has the easiest distribution to see, the cluster 3 has mainly the GDDR3, GDDR4 and GDDR5 type meanwhile the cluster 2 has the HBM-1 type of memory which after searching we found that it is a type special of Memory with a lot of power, but due to its cost and that it's not always used it's not used mainly for enterprisers or for scientific search. That fact would explain why the Maximum Resolution is lower, since GPUs can be used, and are used, to scientific research the main field where they are used is for gaming, where we can find those high resolutions that we don't find in the scientific research, since it's not necessary.

10.2. Cluster's Interpretation

Cluster 1

The GPUs in this cluster have less L2 Cache and less Memory Speed than the other 3 clusters. Also, in this cluster, they have less Max Power, less Core Speed, less Memory, less Memory Bandwidth, less TMUs and less Texture Rate than clusters 2 and 3, but more than 4. The manufacturer in this cluster is divided between AMD (over 60%) and Nvidia (less than 40%). Also, all GPUs in this cluster are dedicated, and most of them are desktop's GPUs.

Cluster 2

It's the cluster with the second highest Core Speed, L2 Cache and Memory. The GPUs manufacturer in this cluster is AMD. The most common memory type in this cluster is GDDR5 (the rest is DDR3), and the most common OpenGL version is 4.4. In this cluster, the SLI technology is dominant over the Crossfire technology. Also, all GPUs in this cluster are dedicated, and most of them are desktop's GPUs.

Cluster 3

The GPUs in this cluster are the ones with the most different types of buses and they are the ones with more Core Speed and more Memory than the other 3 clusters. The manufacturer of the majority of these GPUs is Nvidia, but there are some that are from AMD and from ATI (all of the GPUs whose manufacturer is ATI are in this cluster). The predominant Memory Type in this cluster is GDDR5, and the rest are DDR3. The most common OpenGL version in this cluster is 4.4. Around 60% of the GPUs in this cluster have a SLI technology, the resting 40% have Crossfire technology. Also, all GPUs in this cluster are dedicated, and most of them are desktop's GPUs.

Cluster 4

The GPUs in this cluster have less MaxPower, less Memory Bandwidth, less Memory Speed, less TMUs and less Texture Rate than in the other clusters. The predominant manufacturer is Intel, but AMD also has a presence and the most common Memory Type is DDR3. Also, in this cluster (unlike in the rest), the Crossfire technology is dominant over the SLI technology, all GPUs are not dedicated, and over 60% of them are notebook's GPUs.

11. Global Discussion

Finally we are going to discuss our final conclusions after all the work that we have done and the difficulties that we have had to face in this project and some doubts that we have had to resolve.

On the one hand, as we said when we started this project we chose this dataset in order to analyze something that we liked, that in our case it has been the graphic cards, of course when we started we had to clean all our data, which give us some difficulties with some variables until the point that we had to use regular expressions, after that we had to implement the KNN algorithm, which was very difficult because of we didn't really understand how KNN works, but after some research we success in its implementation. After that we implemented the Bivariate analysis, which was easy in our case, however, due to the amount of different values for some categorical variables we couldn't generate some plots. And finally we made the PCA and hierarchical clustering which were easy to implement but difficult to analyze, for example in hierarchical clustering we had serious doubts about which method to use to measure the distance among variables and after that the number of clusters, since it was our first time doing something like that.

On the other hand, after implementing PCA and Hierarchical clustering we were able to make conclusions about our dataset, for example thanks to PCA we have been able to to see the relationship between Memory_Bus and the core_speed achieved among our GPUs or a group of special graphic cards that have a lot of power consumption but since they are all together we have been able to discover some of their characteristics like a big Memory_Bus and a high Core_Speed. Secondly, thanks to Clustering, we were able to visualize the different groups of GPUs depending on they performance, for example, in our Cluster 2 and 3 have the best performance, specifically in our Cluster 3 we had principally GPUs from Nvidia and cluster 2 was from AMD, on the other hand Cluster 1 and 3 were the worst, where Cluster 1 where a mix of AMD and Nvidia and in Cluster 4 we had graphics from Intel. However we found some differences between PCA and Hierarchical Clustering for example in PCA it doesn't divide the GPUs based on its Core Speed although Hierarchical Clustering does that since we can observe how we have seen a tendency to separate the different GPUs based on their performance, and in the PCA we can see that for the variable Memory_Bus, that is an indicator of Performance, is one of the most distributed variables in our plot, or in other words one of the variables that is easier to be classificated.

12. Working plan

The following Gantt diagram represents the final distribution of the different tasks along the time of the project. Respect to the project we didn't think that PCA analysis would have taken us more time than expected as well as Hierarchical Clustering and Bivariate analysis because of the amount of plots to be commented on.

The table below the Gantt diagram shows the final distribution of the different tasks among the members of the team, which some of us moved from one task to another in order to support other classmates.

	Sept.			Oct.				
	W3	W4	W5	W1	W2	W3	W4	W5
1. Definition and projects assignment.								
2. Project kick-off								
3. Project development								
3.1.Initial working plan								
3.2.Metadata file								
3.4.Univariate Descriptive								
3.5.Data Preprocessing								
3.6.Decisions taken for each step								
4. Report to be delivered								
4.1.Motivation								
4.2.Data Source presentation								
4.3.Formal description of Data								
4.4.Data Mining process performed								
4.5.Description of Preprocessing								
4.6.Statistical descriptive analysis								
4.7.PCA analysis								
4.8.Hierarchical Clustering								
4.9.Profiling of clusters								
4.10.Global discussion								
4.11.Working plan								
4.12.R Scripts								
5.PPT								

	Sept.			Oct.				
	W3	W4	W5	W1	W2	W3	W4	W5
1. Definition and projects assignment.								
2. Project kick-off								
3. Project development								
3.1.Initial working plan								
3.2.Metadata file								
3.4.Univariate Descriptive								
3.5.Data Preprocessing								
3.6.Decisions taken for each step								
4. Report to be delivered								
4.1.Motivation								
4.2.Data Source presentation								
4.3.Formal description of Data								
4.4.Data Mining process performed								
4.5.Description of Preprocessing								
4.6.Statistical descriptive analysis								
4.7.PCA analysis								
4.8.Hierarchical Clustering								
4.9.Profiling of clusters								
4.10.Global discussion								
4.11.Working plan								
4.12.R Scripts								
5.PPT								

Task	Manel Aguilar	Daniel Cano	Oriol Catasús	Jesús Molina	Eduard Ortuño	Adrià Ventura
1. Definition and projects assignment.	X	X	X	X	X	X
2. Project kick-off	X	X	X	X	X	X
3. Project development						
3.1.Initial working plan			X			X
3.2.Metadata file					X	X
3.4.Univariate Descriptive	X	X				
3.5.Data Preprocessing	X	X		X		
3.6.Decisions taken for each step	X	X	X	X	X	X
4. Report to be delivered						
4.1.Motivation	X					X
4.2.Data Source presentation			X			
4.3.Formal description of Data		X				X
4.4.Data Mining process performed				X	X	
4.5.Description of Preprocessing	X	X				
4.6.Statistical descriptive analysis			X	X		
4.7.PCA analysis		X			X	X
4.8.Hierarchical Clustering	X		X		X	
4.9.Profiling of clusters		X		X	X	
4.10.Global discussion	X	X	X	X	X	X
4.11.Working plan	X	X	X	X	X	X
4.12.R Scripts	X	X	X	X	X	X
5.PPT	X	X	X	X	X	X

Task	Manel Aguilar	Daniel Cano	Oriol Catasús	Jesús Molina	Eduard Ortuño	Adrià Ventura
1. Definition and projects assignment.	X	X	X	X	X	X
2. Project kick-off	X	X	X	X	X	X
3. Project development						
3.1.Initial working plan			X			X
3.2.Metadata file	X				X	X
3.4.Univariate Descriptive	X	X				
3.5.Data Preprocessing	X	X		X		
3.6.Decisions taken for each step	X	X	X	X	X	X
4. Report to be delivered						
4.1.Motivation	X					X
4.2.Data Source presentation			X			
4.3.Formal description of Data		X				X
4.4.Data Mining process performed				X	X	
4.5.Description of Preprocessing	X	X				
4.6.Statistical descriptive analysis			X	X		
4.7.PCA analysis		X				X
4.8.Hierarchical Clustering	X		X		X	
4.9.Profiling of clusters		X		X	X	
4.10.Global discussion	X	X	X	X	X	X
4.11.Working plan	X	X	X	X	X	X
4.12.R Scripts	X	X	X	X	X	X
5.PPT	X	X	X	X	X	X