VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF APPLIED SCIENCE

PROBABILITY AND STATISTICS (MT2013)

Group Assignment

”The impact of GPU’s characteristics on its Release_Price”

Instructor:	Dr. Phan Thi Huong
Group’s Members:	Tran Nguyen The Nhat – 2252556
	Nguyen Tien Hung – 2252280
	Phan Chi Vy – 2252938
	Ngo Bao Ngoc – 2252536
	Hoang Van Phi – 2252608
Class:	CC03 – Group: 9

Ho Chi Minh City, May 2024

Member list & Workload

No.	**Fullname**	**Student ID**	**Evaluation**	**Percentage of work**
1	Tran Nguyen The Nhat	2252556		20%
2	Nguyen Tien Hung	2252280		20%
3	Phan Chi Vy	2252938		20%
4	Ngo Bao Ngoc	2252536		20%
5	Hoang Van Phi	2252608		20%

Leader: Tran Nguyen The Nhat – **Email:** nhat.tran2402@hcmut.edu.vn – **Zalo:** 0785082515

Contents

1 Introduction

1.1 Report

This is a report for Team Project of Group 9, class CC03 of Probability and Statistics Course (MT2013) at Ho Chi Minh University of Technology. This project was implemented during Semester HK232, with the main point was to analyse the dataset **All_GPUs.csv**. For all technical calculations, comparisons and building models, our team used R - a programming language focused mainly in statistical computing and data visualization, and RStudio - an IDE for running codes.

1.2 Dataset

A graphic processing unit (GPU) is an electronic circuit that can perform mathematical calculations at high speed. Computing tasks like graphics rendering, machine learning (ML), and video editing require the application of similar mathematical operations on a large dataset. A GPU's design allows it to perform the same operation on multiple data values in parallel. This increases its processing efficiency for many compute-intensive tasks.

In this project, our team used dataset **All_GPUs.csv** which was retrieved from **Computer Parts (CPUs and GPUs) Dataset (Kaggle)** by author **Ilissek**. This dataset contains detailed specifications, release dates, and release prices of computer parts, it has the shape of **3406 observations** and **34 attributes**:

- `Architecture` - the name of the architecture that builds the GPU.
- `Best_Resolution` - resolution recommended by the manufacturer.
- `Boost_Clock` - the maximum clock speed that the graphics card can achieve under normal conditions before the GPU Boost is activated.
- `Core_Speed` - also known as engine clock, GPU clock speed indicates how fast the cores of a GPU are.
- `DVI_Connection` - the number of DVI (Digital Visual Interface) ports that the GPU has.
- `Dedicated` - indicates whether the GPU is separate from the CPU.
- `Direct_X` - a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms.
- `DisplayPort_Connection` - number of DP connection ports that the GPU has.
- `HDMI_Connection` - number of HDMI (High-Definition Multimedia Interface) connection ports that the GPU has.
- `Integrated` - indicates whether the GPU has integrated CPU or not.
- `L2_Cache` - like CPU, GPU also has L2 cache. The L2 cache feeds the L1 cache, which feeds the processor.
- `Manufacturer` - the name of the manufacturer of the GPU.
- `Max_Power` - the maximum amount of graphics board power that the system power supply should be able to provide to the GPU.
- `Memory` - GPU's own memory capacity.
- `Memory_Bandwidth` - the external bandwidth between a GPU and its associated system.
- `Memory_Bus` - the graphics processor is connected to the RAM (Random Access Memory) on the card via a memory bus.
- `Memory_Speed` - the memory clock is the speed of the VRAM (Video RAM) on the GPU.

- `Memory_Type` - type of graphics card memory.
- `Name` - the name of the GPU.
- `Notebook_GPU` - GPUs for laptops.
- `Open_GL` - OpenGL version supported by GPU.
- `Power_Connector` - type of port that connects the source to the GPU.
- `Process` - GPU manufacturing process.
- `ROPs` - a hardware component in modern GPUs and one of the final steps in the rendering process of modern graphics cards.
- `Release_Date` - the date the manufacturer opens the GPU for sale.
- `Release_Price` - starting price.
- `Resolution_WxH` - maximum resolution.
- `SLI_Crossfire` - ability to support SLI(Nvidia) or Crossfire(AMD).
- `Shader` - is a type of computer program originally used for shading in 3D scenes (the production of appropriate levels of light, darkness, and color in a rendered image).
- `TMUs` - number of texture mapping units.
- `Texture_Rate` - the texture filter rate of the GPU is representative of how many pixels (specifically 'texels') the GPU can render per second.
- `VGA_Connection` - number of VGA connection ports.

2 Background

2.1 Exploratory data analysis

Exploratory data analysis (EDA) plays a vital role in the data science workflow. It helps to unearth valuable insights from complex data sets, allowing data scientists to make data-driven decisions and develop more accurate models. With its focus on uncovering hidden patterns and exploring the data's underlying structure, EDA serves as an essential step in the analytical process for any data-driven project. By plotting the raw data, we can gain an understanding of the general behavior and distribution of the variables:

- `Histograms` are used to visualize the distribution of a numerical variable.
- `Box plots` are used to display the distributions of numerical data values, particularly when comparing them across several groups.
- `Scatter plots` are employed to comprehend the finest characteristics that may be applied to describe a link between two variables or to create the most distinct clusters.
- `Correlation Matrix`: The correlation coefficients between variables are displayed in a table called a correlation matrix. The association between two variables is displayed in each cell of the table.

2.2 K-Nearest Neighbor Algorithm

The K-Nearest Neighbor (K-NN) algorithm is a supervised machine learning method employed to tackle classification and regression problems. The idea in k-NN methods is to identify k samples in the dataset that are similar or close in the space. Then we use these k samples to estimate the value of the missing data points. Each sample's missing values are imputed using the mean value of the k-neighbors found in the dataset. The assumption behind using k-NN for missing values is that a point value can be approximated by the values of the points that are closest to it, based on other variables. The nearest, most similar, neighbours are found by minimising a distance function, usually the $Euclideandistance$.

$$E(a,b) = \sqrt{\sum_{i \in D} (x_{ai} - x_{bi})^2}$$

where:

- $E(a,b)$ is the distance between the two cases a and b.
- x_{ai} and x_{bi} are the values of attribute i in cases a and b respectively.
- D is the set of attributes with non-missing values in both cases.

To be easily understood, k-NN algorithm has the following basic steps:

1. Calculate distance.
2. Find closest neighbors.
3. Vote for labels.

2.3 Log-transformation

Logarithm transformation is a data transformation method in which it replaces each variable x with a log(x). The choice of the logarithm base is usually left up to the analyst and it would depend on the purposes of statistical modeling. When our original continuous data do not follow the bell curve, we can log transform this data to make it as "normal" as possible so that the statistical analysis results from this data become more valid. Therefore, we decided to convert the data to a logarithmic equation of the form: $f(x) = log(x)$ **where:** x is values in our dataset.

2.4 Inferential Knowledge Background

2.4.1 Multiple Linear Regression

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression is to model the linear relationship between the explanatory (independent) variables and response (dependent) variables. The formula of multiple linear regression:

$$y = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n + \epsilon$$

where:

- y is the predicted value of the dependent variable.
- β_0 is the y-intercept (value of y when all other parameters are set to 0).
- $\beta_1 X_1$ is the regression coefficient of the first independent variable.
- $\beta_n X_n$ is the regression coefficient of the last independent variable.
- e is the model error a.k.a. how much variation there is in our estimate of y.

When using linear regression, there are several assumptions that are typically made.

Assumption 1: Linearity, the relationship between the dependent variable and the independent variable is linear:
Assumption 2: Independence, the observations are independent of each other:

$$Cov(\varepsilon_i, \varepsilon_j) = 0, i \neq j$$

Assumption 3: Homoscedasticity, the variance of the errors is constant across all levels of the independent variable(s):

$$Var(\varepsilon_i) = \sigma^2, \forall i$$

Assumption 4: Normality, the errors are normally distributed:

$$\varepsilon \sim N(0, \sigma^2)$$

To find the best-fit line for each independent variable, multiple linear regression calculates three things:

- The regression coefficients that lead to the smallest overall model error.
- The t-statistic of the overall model.
- The associated p-value (how likely it is that the t-statistic would have occurred by chance if the null hypothesis of no relationship between the independent and dependent variables was true).
- It then calculates the t-statistic and p-value for each regression coefficient in the model.

2.4.2 Support Vector Machine

Support Vector Machine is a machine learning algorithm that uses supervised learning models to solve complex classification, regression, and outlier detection problems by performing optimal data transformations that determine boundaries between data points based on predefined classes, labels, or outputs. SVMs are widely adopted across disciplines such as healthcare, natural language processing, signal processing applications, and speech & image recognition fields. Technically, the primary objective of the SVM algorithm is to identify a hyperplane that distinguishably segregates the data points of different classes. The hyperplane is localized in such a manner that the largest margin separates the classes under consideration.

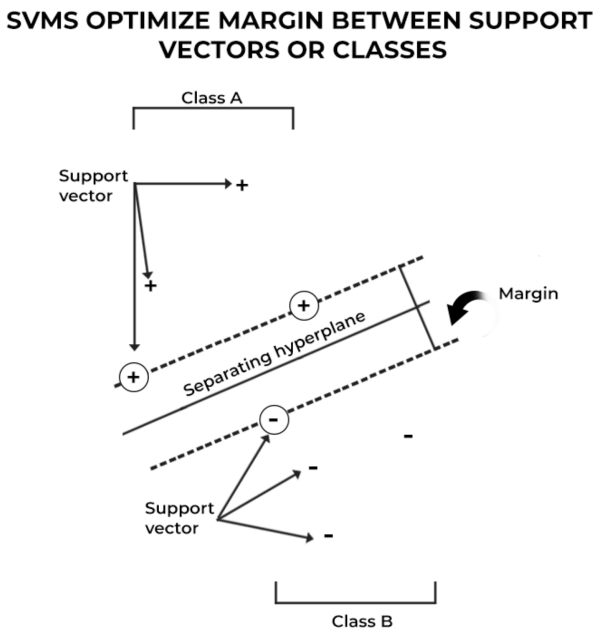


Figure 1: *SVMs Optimize Margin Between Support Vectors or Classes*

As seen in the above figure, the margin refers to the maximum width of the slice that runs parallel to the hyperplane without any internal support vectors. Such hyperplanes are easier to define for linearly separable problems; however, for real-life problems or scenarios, the SVM algorithm tries to maximize the margin between the support vectors, thereby giving rise to incorrect classifications for smaller sections of data points.
SVMs are potentially designed for binary classification problems. However, with the rise in computationally intensive multiclass problems, several binary classifiers are constructed and combined to formulate SVMs that can implement such multiclass classifications through binary means. SVM works for classification:

1. **Data Representation:** The input data is represented as feature vectors in an n-dimensional space, where each dimension corresponds to a feature.
2. **Finding the Hyperplane:** SVM aims to find the hyperplane that maximizes the margin between the classes. The hyperplane is a decision boundary that separates the data points into different classes. In a binary classification scenario, this hyperplane is a line in 2D space, a plane in 3D space, and a hyperplane in higher-dimensional spaces.
3. **Maximizing Margin:** The margin is the distance between the hyperplane and the nearest data points from each class, known as support vectors. SVM aims to maximize this margin, as it generalizes better to unseen data and improves the classifier's robustness.
4. **Handling Non-Linearity:** In cases where the data is not linearly separable, SVM can still find a separating hyperplane by transforming the input space into a higher-dimensional space using a technique called the kernel trick. This allows SVM to effectively deal with non-linear decision boundaries.
5. **Classification:** Once the hyperplane is determined, classification of new data points is done by evaluating which side of the hyperplane they fall on. If a point lies on one side of the hyperplane, it's classified as belonging to one class; if it lies on the other side, it's classified as belonging to the other class.
6. **Optimization:** SVM involves solving an optimization problem to find the optimal hyperplane. The objective is to minimize the classification error while maximizing the margin. This is typically done using techniques such as gradient descent or quadratic programming.

SVMs have several advantages, including their effectiveness in high-dimensional spaces, ability to handle non-linear decision boundaries, and resistance to overfitting. However, they can be sensitive to the choice of kernel and parameters, and they can be computationally expensive, especially with large datasets.

2.4.3 Random Forest Regression

Random Forest Regression is a supervised learning algorithm and bagging technique that uses an ensemble learning method for regression in machine learning. The trees in random forests run in parallel, meaning there is no interaction between these trees while building the trees.

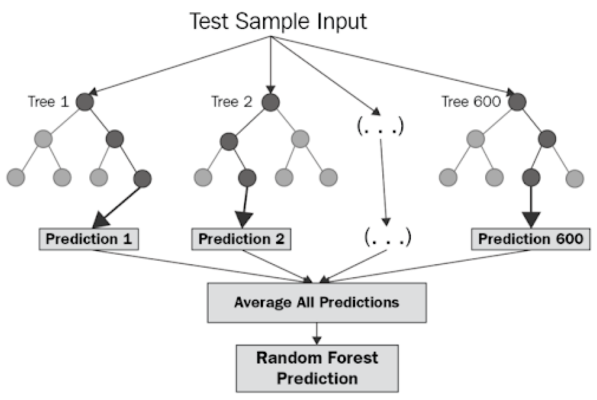


Figure 2: *Random Forest sample*

Random forest operates by constructing a multitude of decision trees at training time and outputting the class that's the mode of the classes (classification) or mean prediction (regression) of the individual trees. A random forest is a meta-estimator (i.e. it combines the result of multiple predictions), which aggregates many decision trees with some helpful modifications.

Advantages:

1. Random forest is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
2. It runs efficiently on large databases.
3. It can handle thousands of input variables without variable deletion.
4. It gives estimates of what variables are important in the classification.
5. It generates an internal unbiased estimate of the generalization error as the forest building progresses.
6. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

2.4.4 Analysis of Variance

Analysis of variance (ANOVA) is a statistical test used to evaluate the difference between the means of more than two groups. This statistical analysis tool separates the total variability within a data set into two components: random and systematic factors. A one-way ANOVA uses one independent variable. A two-way ANOVA uses two independent variables. Analysts use the ANOVA test to determine independent variables' influence on the dependent variable in a regression study.

Key Takeaways

1. Analysis of variance (ANOVA) is a statistical test used to evaluate the difference between the means of more than two groups.
2. A one-way ANOVA uses one independent variable. A two-way ANOVA uses two independent variables.
3. If no true variance exists between the groups, the ANOVA's F-ratio should equal close to 1.

Using ANOVA

1. An ANOVA test can be applied when data needs to be experimental. Analysis of variance is employed if there is no access to statistical software and ANOVA must be calculated by hand. It is simple to use and best suited for small samples. It is employed with subjects, test groups, and between and among groups.
2. ANOVA is similar to multiple two-sample t-tests. However, it results in fewer type I errors. ANOVA groups differences by comparing each group's means and includes spreading the variance into diverse sources. Analysts use a one-way ANOVA with collected data about one independent variable and one dependent variable. A two-way ANOVA uses two independent variables. The independent variable should have at least three different groups or categories. ANOVA determines if the dependent variable changes according to the level of the independent variable.
3. A researcher might test students from multiple colleges to see if students from one of the colleges consistently outperform students from the other schools. In a business application, an R&D researcher might test two different processes of creating a product to see if one is better than the other in terms of cost efficiency.

$$F = \frac{MST}{MSE}$$

where:

- F: ANOVA coefficient.
- MST: Mean sum of squares due to treatment.
- MSE: Mean sum of squares due to error.

One-Way vs. Two-Way ANOVA

1. A one-way ANOVA evaluates the impact of a sole factor on a sole response variable. It determines whether all the samples are the same. The one-way ANOVA is used to determine whether there are any statistically significant differences between the means of three or more independent groups.
2. A two-way ANOVA is an extension of the one-way ANOVA. With a one-way, you have one independent variable affecting a dependent variable. With a two-way ANOVA, there are two independents. For example, a two-way ANOVA allows a company to compare worker productivity based on two independent variables, such as salary and skill set. It is utilized to observe the interaction between the two factors and test the effect of two factors simultaneously.

2.4.5 Welch's Test

Welch's Test for Unequal Variances (also called Welch's t test, Welch's adjusted T or unequal variances t-test) is a modification of Student's t-test(more reliable when the two samples have unequal variances and/or unequal sample sizes) to see if two sample means are significantly different. The modification is to the degrees of freedom used in the test, which increases the test power for samples with unequal variance. There are two hypotheses in the Welch's test:

- Null hypothesis H_0: for the test is that the means are equal.
- Alternative hypothesis H_1: for the test is that means are not equal.

Below is the formula of Welch's test:

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{se_1^2 + se_2^2}}$$

where:

- $\overline{X_i}$ is the i^{th} sample mean.
- s_i is the i^{th} standard deviation.

3 Descriptive statistics

3.1 Data Preprocessing

3.1.1 Import and Selection

```r
pacman::p_load(
  dplyr,   #data manipulation
  ggplot2, #visualization
  ggfortify, #visualization
  ggcorrplot, #statistical modeling
  broom, #additional visualization functionalities
  rio # for export function
)
all_gpus <- read.csv("All_GPUs.csv")
data = all_gpus %>%
select("Manufacturer","Boost_Clock","Core_Speed","Max_Power","Memory",
"Memory_Bandwidth","Memory_Bus","Memory_Speed","Release_Price","Shader","TMUs")
```

- After review, our team chose to extract 11 out of 34 variables: **Manufacturer, Boost_Clock, Core_Speed, Max_Power, Memory, Memory_Bandwidth, Memory_Bus, Memory_Speed, Release_Price, Shader, TMUs.**
- This is the summary of the chosen data:

```
> summary(data)
 Manufacturer        Boost_Clock          Core_Speed          Max_Power        
 Length:3406        Length:3406        Length:3406        Length:3406       
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  

 Memory_Bandwidth    Memory_Bus         Memory_Speed       Release_Price      
 Length:3406        Length:3406        Length:3406        Length:3406       
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  

      TMUs            Memory               Shader     
 Min.   :  1.00    Length:3406         Min.   :1.000  
 1st Qu.: 24.00    Class :character    1st Qu.:5.000  
 Median : 56.00    Mode  :character    Median :5.000  
 Mean   : 69.38                        Mean   :4.706  
 3rd Qu.:104.00                        3rd Qu.:5.000  
 Max.   :384.00                        Max.   :5.000  
 NA's   :538                           NA's   :107    
```

Listing 1: Summary

3.1.2 Conversion

Since some of the attributes are **character**, we will convert it to number using function *as.numeric(...)*. After conversion, here is the result:

```
Manufacturer Boost_Clock Core_Speed Max_Power Memory Memory_Bandwidth Memory_Bus
1       Nvidia          NA        738       141   1024             64.0        256
2          AMD          NA         NA       215    512            106.0        512
3          AMD          NA         NA       200    512             51.2        256
4          AMD          NA         NA        NA    256             36.8        128
5          AMD          NA         NA        45    256             22.4        128
  Release_Price Shader TMUs Memory_Speed
1            NA      4   64         1000
2            NA      4   16          828
3            NA      4   16          800
4            NA      4    8         1150
5            NA      4    8          700
```

Listing 2: 5 lines of data after conversion

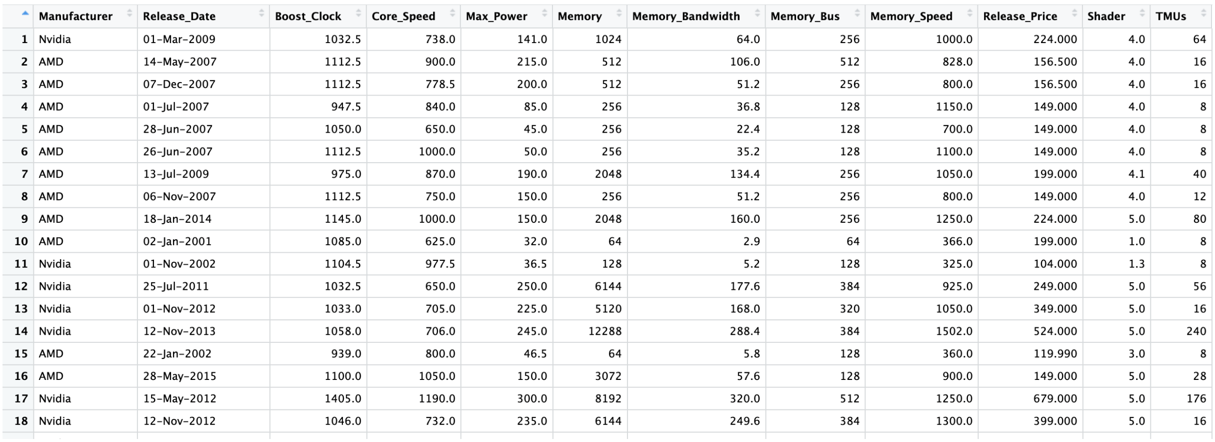
After choosing the appropriate attributes and converting them to reproducible types (numeric), we now have the subset of the original raw dataset. However, we should check missing values using function *colSums*(*is.na*(*data*)). Here is the result:

```
Manufacturer        Boost_Clock        Core_Speed         Max_Power            Memory
           0               1960               936               625               420
Memory_Bus      Memory_Speed    Release_Price         Shader             TMUs
        62               105             2850            107              538
Memory_Bandwidth
             121
```

Listing 3: Missing values

3.1.3 Applying k-NN to the dataset

In order to **deal with missing values**, our team used **K-Nearest Neighbor model** as introduced in the background to predict the missing values. When applying model to the dataset, we choose k (numbers of nearest neighbors used) equals to square root of numbers of observations (3406) **(kNN(data, k = sqrt(nrow(data)), imp_var = FALSE))**. Here is the view of dataset:



	Manufacturer	Release_Date	Boost_Clock	Core_Speed	Max_Power	Memory	Memory_Bandwidth	Memory_Bus	Memory_Speed	Release_Price	Shader	TMUs
1	Nvidia	01-Mar-2009	1032.5	738.0	141.0	1024	64.0	256	1000.0	224.000	4.0	64
2	AMD	14-May-2007	1112.5	900.0	215.0	512	106.0	512	828.0	156.500	4.0	16
3	AMD	07-Dec-2007	1112.5	778.5	200.0	512	51.2	256	800.0	156.500	4.0	16
4	AMD	01-Jul-2007	947.5	840.0	85.0	256	36.8	128	1150.0	149.000	4.0	8
5	AMD	28-Jun-2007	1050.0	650.0	45.0	256	22.4	128	700.0	149.000	4.0	8
6	AMD	26-Jun-2007	1112.5	1000.0	50.0	256	35.2	128	1100.0	149.000	4.0	8
7	AMD	13-Jul-2009	975.0	870.0	190.0	2048	134.4	256	1050.0	199.000	4.1	40
8	AMD	06-Nov-2007	1112.5	750.0	150.0	256	51.2	256	800.0	149.000	4.0	12
9	AMD	18-Jan-2014	1145.0	1000.0	150.0	2048	160.0	256	1250.0	224.000	5.0	80
10	AMD	02-Jan-2001	1085.0	625.0	32.0	64	2.9	64	366.0	199.000	1.0	8
11	Nvidia	01-Nov-2002	1104.5	977.5	36.5	128	5.2	128	325.0	104.000	1.3	8
12	Nvidia	25-Jul-2011	1032.5	650.0	250.0	6144	177.6	384	925.0	249.000	5.0	56
13	Nvidia	01-Nov-2012	1033.0	705.0	225.0	5120	168.0	320	1050.0	349.000	5.0	16
14	Nvidia	12-Nov-2013	1058.0	706.0	245.0	12288	288.4	384	1502.0	524.000	5.0	240
15	AMD	22-Jan-2002	939.0	800.0	46.5	64	5.8	128	360.0	119.990	3.0	8
16	AMD	28-May-2015	1100.0	1050.0	150.0	3072	57.6	128	900.0	149.000	5.0	28
17	Nvidia	15-May-2012	1405.0	1190.0	300.0	8192	320.0	512	1250.0	679.000	5.0	176
18	Nvidia	12-Nov-2012	1046.0	732.0	235.0	6144	249.6	384	1300.0	399.000	5.0	16

Figure 3: *View of dataset after k-NN*

For later use, we export the dataset to **"gpu_clean.csv"** file.

```
export(df,"gpu_clean.csv") #for later_use
```

The reason we choose k-NN is because this algorithm can compete with the most accurate model as it gives highly accurate predictions. The k-NN algorithm is a type of lazy learning, where the computation for the generation of the predictions is deferred until classification. Although this method increases the costs of computation compared to other algorithms, k-NN is still the better choice for applications where predictions are not requested frequently but where accuracy is important.

3.2 Descriptive statistic

3.2.1 Summary

After cleaning process, we certainly have a clear and clean dataset in the data frame with 0 missing value. Here is the summary of **df**:

```
Manufacturer         Release_Year        Boost_Clock      Core_Speed        Max_Power
Length:3406          Length:3406         Min.   : 400     Min.   : 100.0    Min.   :  1.0
Class :character     Class :character    1st Qu.:1033     1st Qu.: 700.0    1st Qu.: 45.0
Mode  :character     Mode  :character    Median :1085     Median : 910.0    Median : 95.0
                                         Mean   :1121     Mean   : 885.7    Mean   :119.5
                                         3rd Qu.:1176     3rd Qu.:1054.0    3rd Qu.:170.0
                                         Max.   :1936     Max.   :1784.0    Max.   :780.0
Memory_Bandwidth       Memory_Bus        Memory_Speed     Release_Price        Shader
Min.   :   1.0       Min.   :  32.0      Min.   : 100     Min.   :   23.0    Min.   :1.000
1st Qu.:  28.8       1st Qu.: 128.0      1st Qu.: 800     1st Qu.:  149.0    1st Qu.:5.000
```

```
Median : 102.4      Median : 128.0     Median :1125    Median :  199.0    Median :5.000
Mean   : 134.8      Mean   : 203.9     Mean   :1160    Mean   :  231.6    Mean   :4.687
3rd Qu.: 192.3      3rd Qu.: 256.0     3rd Qu.:1502    3rd Qu.:  240.0    3rd Qu.:5.000
Max.   :1280.0      Max.   :8192.0     Max.   :2127    Max.   :14999.0    Max.   :5.000
    Memory              TMUs
Min.   :   16    Min.   :  1.00
1st Qu.: 1024    1st Qu.: 16.00
Median : 2048    Median : 48.00
Mean   : 2630    Mean   : 60.26
3rd Qu.: 4096    3rd Qu.: 80.00
Max.   :32000    Max.   :384.00
```

Listing 4: Summary of df

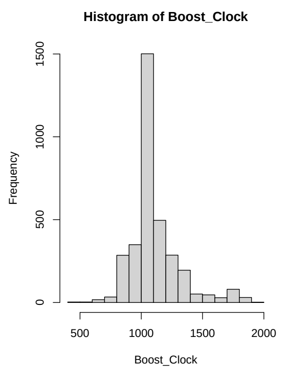
Because **Manufacturer** and **Release_Year** are categorical variables, we can not have an overview information in this variable, so we try to make these more specific:

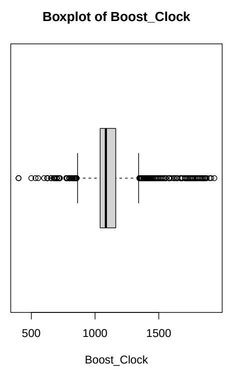
```
> my_table=(table(df$Manufacturer))
> cat("Number of Manufacturers :",nrow(my_table))

Number of Manufacturers : 4

> show(my_table)
   AMD     ATI   Intel  Nvidia
  1317      92     254    1743

> my_table=(table(df$Release_Year))
> cat("Total years :",nrow(my_table))
Total years : 20

> show(my_table)
1998 1999 2000 2001 2002 2003 2004 2005 2006 2007
   2    7    5   10   31   28   37   38   80  106
2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
138  114  257  334  526  476  375  407  279  156
```

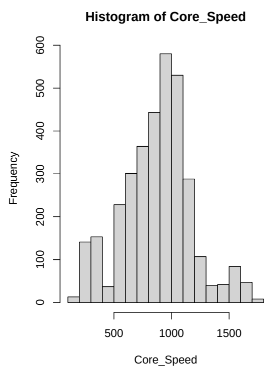
Listing 5: Summary of df

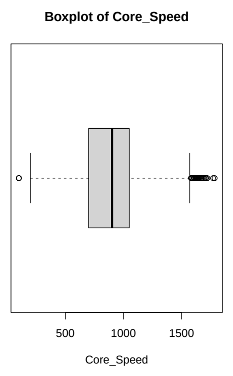
3.2.2 Plot

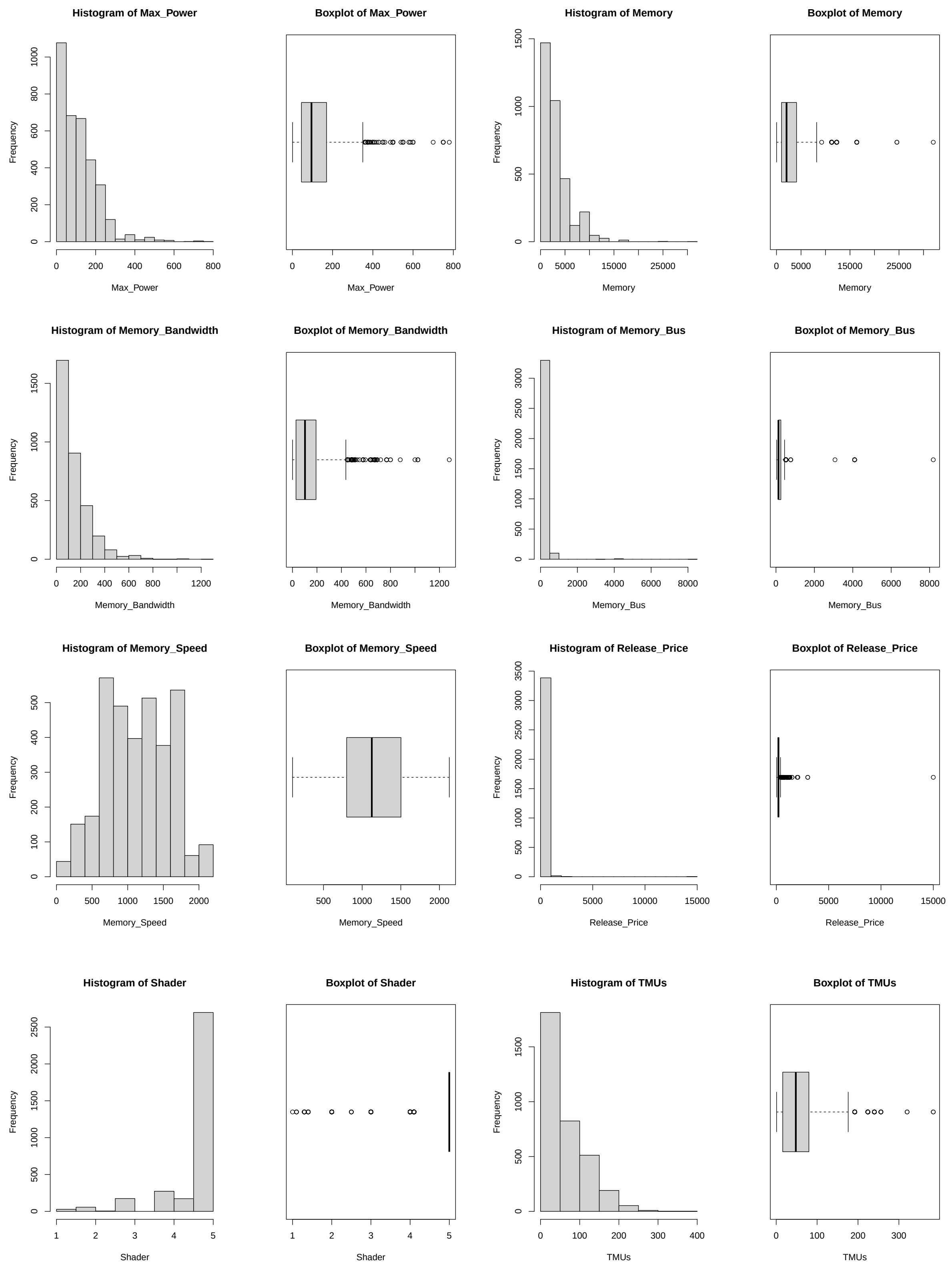
Next, we take a better look at the distribution of the 10 independent variables, utilizing the Histogram and Boxplot:











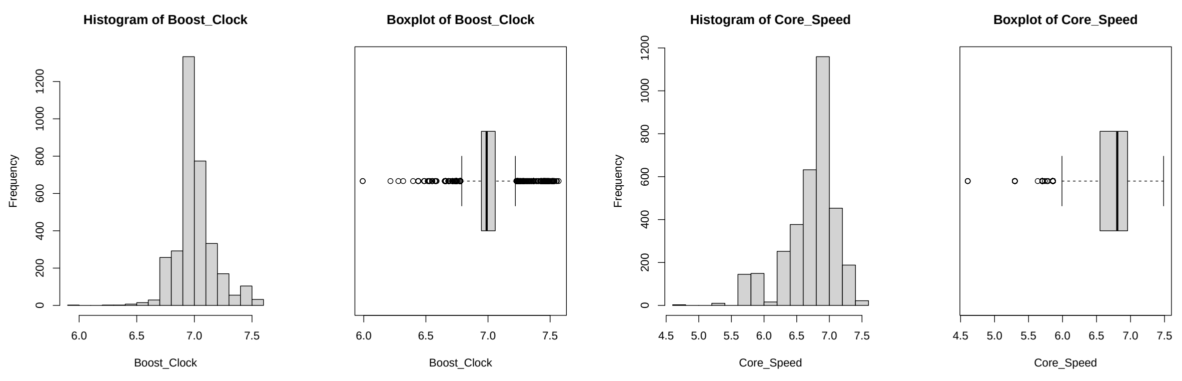
Explanation:

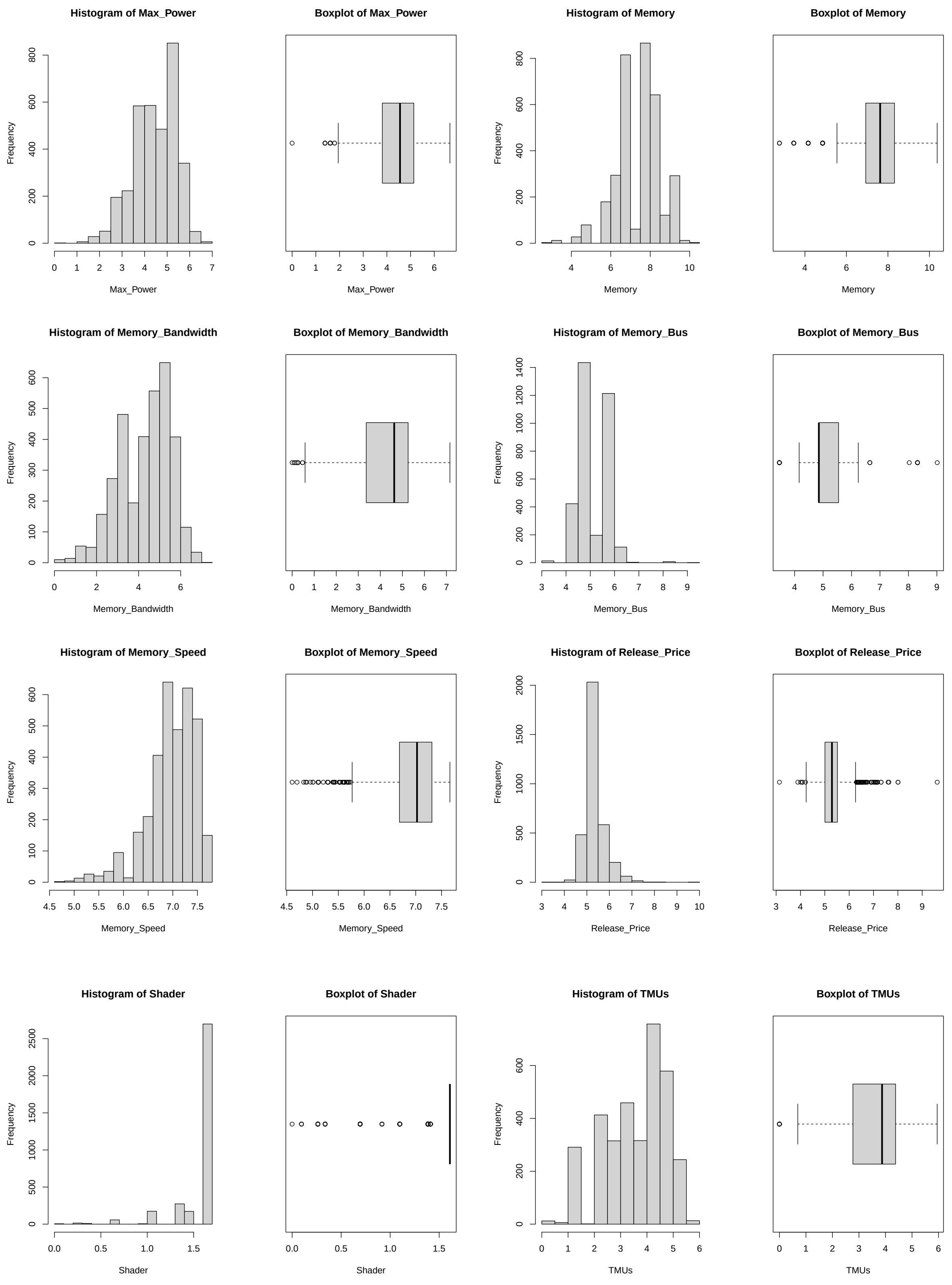
- `Boost Clock and Core Speed:`Both show distributions centered around 1000 MHz, with Boost Clock being slightly right-skewed and Core Speed more symmetric. Boxplots indicate medians around 1000 MHz with outliers on both ends, suggesting variations in clock speeds among different GPU models.

- `Max Power`:Right-skewed distribution with most values under 200 Watts. Boxplot shows a median below 200 Watts, with outliers indicating higher power requirements for some GPUs.
- `Memory`:Right-skewed distribution with most GPUs having up to 5000 MB. Boxplot median around 5000 MB, with outliers showing higher memory capacities.
- `Memory Bandwidth and Memory Bus`:Both attributes show right-skewed distributions, indicating most GPUs have lower bandwidth and bus widths. Boxplots reveal medians at lower ranges with outliers suggesting higher specifications in some models.
- `Memory Speed`:Multi-modal distribution, indicating common speeds at several points. Boxplot shows a median around 1000 units, with a symmetrical spread and outliers.
- `Release Price`:Right-skewed distribution, most GPUs priced under 5000 units. Boxplot median around 5000 units, with some GPUs being significantly more expensive.
- `Shader`:Highly skewed towards level 5, indicating a common standard or preference for this shader level in GPUs. Boxplot collapses at level 5, showing no variability among most GPUs and treating other levels as outliers.
- `TMUs`:Right-skewed distribution, with most GPUs having fewer than 100 TMUs. Boxplot shows a median around 50 TMUs, with an interquartile range from about 25 to 75 TMUs and several outliers indicating higher performance models.

As can be seen, the attribute we considered (Release_Price) is right-skewed, which is difficult for inferential statistics. So we will apply **log-transformation** to the dataset for a better representation.

```r
df['Max_Power'] <- log(df['Max_Power'])
df['Boost_Clock'] <- log(df['Boost_Clock'])
df['Core_Speed'] <- log(df['Core_Speed'])
df['Memory'] <- log(df['Memory'])
df['Memory_Bandwidth'] <- log(df['Memory_Bandwidth'])
df['Memory_Bus'] <- log(df['Memory_Bus'])
df['Memory_Speed'] <- log(df['Memory_Speed'])
df['Release_Price'] <- log(df['Release_Price'])
df['Shader'] <- log(df['Shader'])
df['TMUs'] <- log(df['TMUs'])
```





Finally, not only Release_Price but also most of other attributes are now normally distributed. Notice that the occurrences of $log(Release\_Price) \geq 8$ and $log(Release\_Price) \leq 3.5$ is rare, so it should be eliminated in the next analysis. Now, we use Correlation plot to give the overview of the relation between each pair of variable after log_transformation:

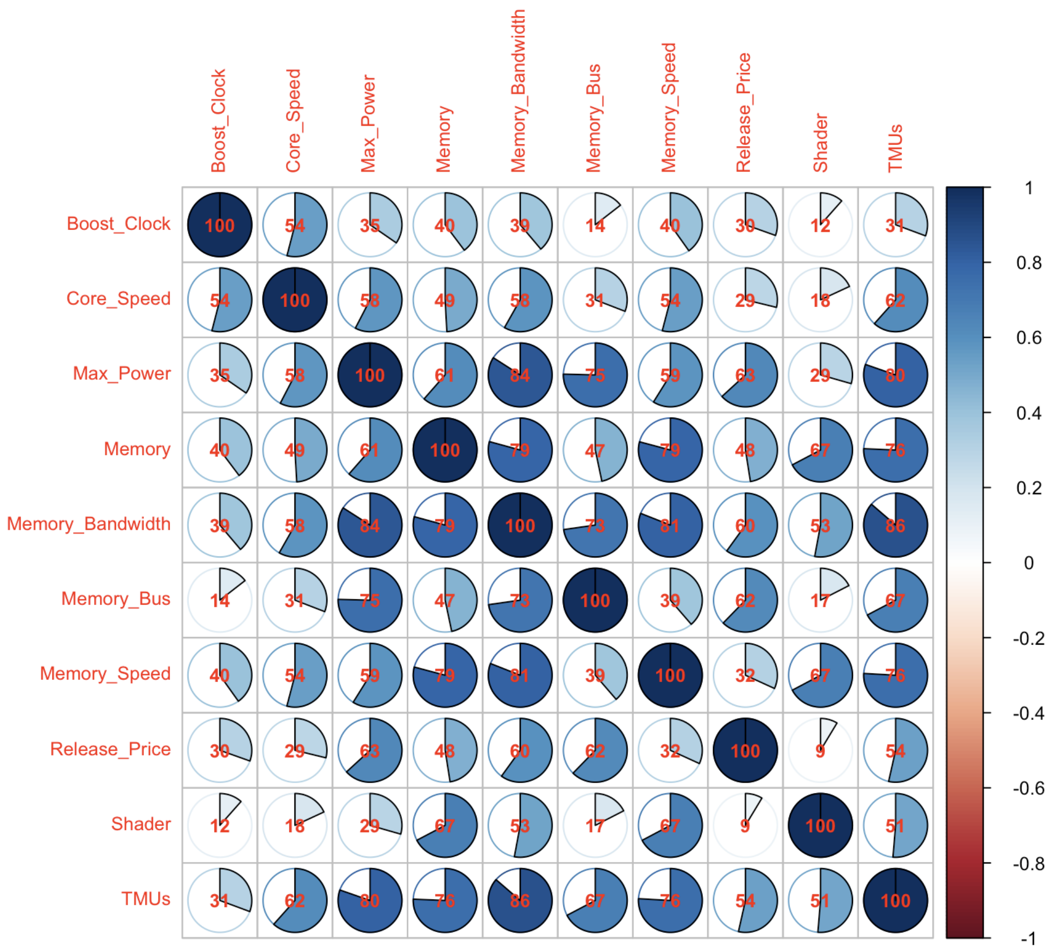


Figure 4: *Correlation Plot*

1. `Boost_Clock`:Strong positive correlation with Core_Speed, which is expected as both involve operational speeds of the GPU. With attributes like Shader and TMUs, indicating that the boost clock speed is not significantly influenced by the number of shaders or texture mapping units.
2. `Core_Speed`: Strongly correlated with Boost_Clock due to their related nature in defining GPU speeds. Shows weak correlation with Shader and TMUs, similar to Boost_Clock.
3. `Max_Power`:Shows a strong positive correlation with Memory, suggesting that GPUs with higher power consumption tend to have more memory. Weak negative correlation with Memory_Bus, indicating that higher power consumption does not necessarily mean a wider memory bus.
4. `Memory`: Strong positive correlations with Memory_Bandwidth and Memory_Speed, as more memory capacity typically supports higher bandwidth and speed. Relatively weak correlation with Shader, suggesting that the amount of memory is not directly related to the number of shaders.
5. `Memory_Bandwidth`:Highly correlated with Memory and Memory_Speed, which is logical since higher bandwidth is often accompanied by higher memory speeds. Weak correlation with Shader and TMUs, showing that memory bandwidth is not closely related to these attributes.
6. `Memory_Bus`: Shows a strong correlation with Memory_Speed, indicating that wider buses are often associated with faster memory speeds. Weak correlation with Max_Power, suggesting that the width of the memory bus does not increase with power consumption.
7. `Memory_Speed`: Strongly correlated with Memory_Bandwidth and Memory_Bus, as faster memory speeds require higher bandwidth and potentially wider buses. Weak correlation with Shader and TMUs, indicating that memory speed is largely independent of these attributes.

8. `Release_Price:`Shows some correlation with Memory and Memory_Bandwidth, suggesting that GPUs with more memory and higher bandwidth tend to be priced higher. Weak correlation with Shader, indicating that the number of shaders does not have a strong impact on the release price.

9. `Shader:` Shows some correlation with TMUs, as both are related to the rendering capabilities of GPUs. Very weak correlations with most other attributes like Boost_Clock, Core_Speed, and Memory, indicating that the number of shaders is quite independent of these features.

10. `TMUs:`Correlates with Shader, which is expected as both are involved in texture and rendering processes. Weak correlations with attributes like Boost_Clock and Core_Speed, showing that TMUs are not significantly influenced by the operational speeds of the GPU.

.

4 Inferential statistic

According to our team's discussion, the main analysis attribute of this report is **Release_Price** of the GPU and the effect of other attributes on this dependent variable. The reason why we choose this attribute is that price forecasting has played an increasingly important role in the ability of a company to remain competitive in the real world. Back to the project, we would try to utilize some Regression models to predict its Release_Price via other configurations (or determinants).

4.1 Data preparation

First, again we load the clean dataset and log-transform it:

```r
#inferential statistics
df <- read.csv("gpu_clean.csv")
df['Max_Power'] <- log(df['Max_Power'])
df['Boost_Clock'] <- log(df['Boost_Clock'])
df['Core_Speed'] <- log(df['Core_Speed'])
df['Memory'] <- log(df['Memory'])
df['Memory_Bandwidth'] <- log(df['Memory_Bandwidth'])
df['Memory_Bus'] <- log(df['Memory_Bus'])
df['Memory_Speed'] <- log(df['Memory_Speed'])
df['Release_Price'] <- log(df['Release_Price'])
df['Shader'] <- log(df['Shader'])
df['TMUs'] <- log(df['TMUs'])
```

According to 3.2 and our statement previously, the occurrences of $log(Release\_Price) \geq 8$ and $log(Release\_Price) \leq 3.5$ is rare is rare, so we will eliminate it from the dataset:

```r
df <- df[df$Release_Price < 8,]
df <- df[df$Release_Price > 3.5,]
```

Because we would make use of Regression models to capture the relationships, a Test Set and a Training Set must be present to perform cross-validation to test the fitness of the model. In machine learning, splitting the data into distinct sets helps prevent overfitting, which occurs when a model memorizes the training data too well and fails to generalize to new, unseen data. So that, we split the data into training and testing sets by 80% and 20% amount of data.

```r
#predict:
set.seed(50)
# use 80% of dataset as training and 20% as testing
sample <- sample(c(T, F), nrow(df), replace=T, prob=c(0.8,0.2))
```

4.2 Hypothesis Testing

4.2.1 One-way ANOVA

In this section, we want to know whether or not a significant difference in the average Release_Price among manufacturers. Then we are going to use function selection to filter the data based on the Release_Price of each manufacturer in df:

```r
aovRelease_Price <- select(df, Manufacturer, Release_Price)
```

ANOVA Test

In order to determine if there is any significant difference in Release_Price among these manufacturers, we enumerate the null hypothesis H_0 and alternative hypothesis H_1.

- H_0: $u1 = u2 = ... = u_i = 0$: There is a similarity in average Release_Price among 4 manufacturers: Nvidia, AMD, Intel and ATI.
- H_1: $u_i \neq 0$ for at least one of the manufacturer have a significant different in Release_Price compared to others.

```
> aov_one <- aov(Release_Price~Manufacturer, data=aovRelease_Price)
> aov_one
Call:
   aov(formula = Release_Price ~ Manufacturer, data = aovRelease_Price)

Terms:
                Manufacturer Residuals
Sum of Squares       19.2300  673.3388
Deg. of Freedom            3      3398

Residual standard error: 0.4451487
Estimated effects may be unbalanced
> summary(aov_one)
               Df Sum Sq Mean Sq F value Pr(>F)
Manufacturer    3   19.2   6.410   32.35 <2e-16 ***
Residuals    3398  673.3   0.198
---
Signif. codes:  0    ***    0.001    **    0.01    *    0.05    .    0.1         1
```

Listing 6: ANOVA

We can see that as the **p-value < 0.05**, we can reject the null hypothesis H_0 and accept the alternative hypothesis H_1.
To compute **one-way ANOVA test**, the data in df must be a fixed-impact model meet the follow requirements:

- All data are independent and collected randomly, since the data is collected from 4 different manufacturers, so this condition is **satisfied**.
- The population from those samples should be normally distributed.
- Homogeneity of variance: the variance among the groups should be approximately equal.

Check **condition 2**, we will use Q-Q plot to see if the data is normally distributed.

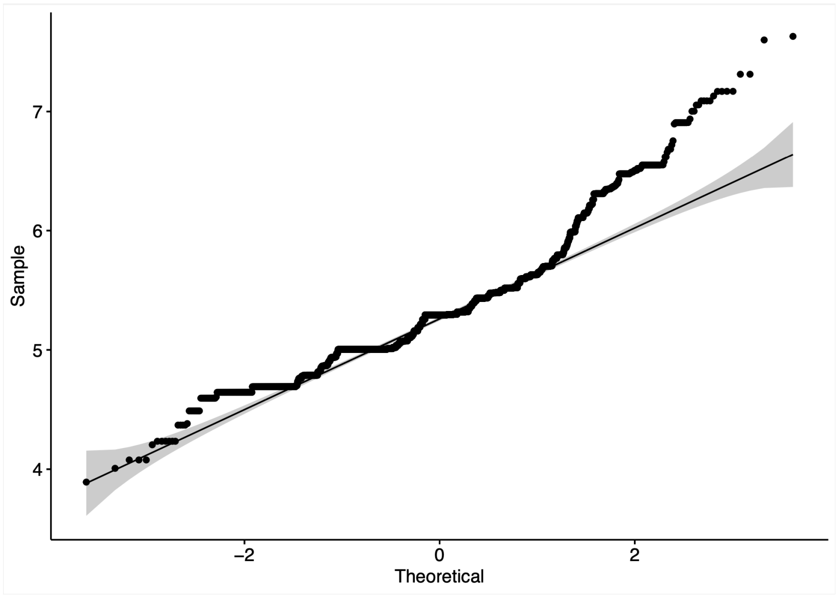


Figure 5: *Q-Q plot for ANOVA*

According to the plot provided, it can be seen that **Release_Price** is not normally distributed. However, this assumption can be rejected since the sample size is large and the **Central Limit Theorem** ensures that the distribution of disturbance term will approximate normality. Also, **the one-way ANOVA** is considered a robust test against the normality assumption. This means that it tolerates violations to its normality assumption rather well. We can conclude that the second condition is **not necessarily satisfied**.

Check **condition 3**, we will use **Levene-test** to evaluate if the variances of two populations are equal. Levene's test is an inferential statistic used to evaluate the equality of variances for a variable determined by two or more groups. The statistical hypotheses are:

- Null hypothesis H_0: All populations are equal.
- Alternative hypothesis H_1: At least two of them are different.

```
Levene's Test for Homogeneity of Variance (center = mean)
        Df F value    Pr(>F)
group    3  183.18 < 2.2e-16 ***
      3398
---
Signif. codes:  0   ***   0.001   **   0.01   *   0.05   .   0.1        1
```

Listing 7: Levene-test

From the output above, we can see that the p-value (< 2.2e-16) is much smaller compares to the alpha level 0.05. Now we reject the null hypothesis H_0 that all populations are equal, which also means there is a difference among manufacturers.

Welch's test

Since the variance test fail as its p-value is less than 0.05 and the qq-plot has a lot of point that deviate from the line, we will use Welch's test to check if the assumption is correct or not.

```r
oneway.test(Release_Price ~ Manufacturer, data = aovRelease_Price)
```

```
> oneway.test(Release_Price ~ Manufacturer, data = aovRelease_Price)

  One-way analysis of means (not assuming equal variances)

data:  Release_Price and Manufacturer
F = 43.219, num df = 3.00, denom df = 411.37, p-value < 2.2e-16
```

Listing 8: Levene-test

According to Welch's test result, we can officially claim that **there is a difference in average Release_Price among manufacturers.**

4.3 Multiple Linear Regression

In this section, we will conduct an investigation on the relationship between Release_Price with other features.

4.3.1 Fitting the multi-linear model

```r
#MLR
data = df %>%
  select("Boost_Clock","Core_Speed","Max_Power","Memory","Memory_Bandwidth"
  ,"Memory_Bus","Memory_Speed","Release_Price","Shader","TMUs")
train <- data[sample, ]
test <- data[!sample, ]

mdl_price_vs_all = lm(Release_Price~., data=train)
```

```
summary(mdl_price_vs_all)
```

The **summary()** gives an overview over some statistics in our model.

```
> summary(mdl_price_vs_all)

Call:
lm(formula = Release_Price ~ ., data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-1.11709 -0.18625 -0.03658  0.16537  1.46774

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       4.40185    0.36023  12.219  < 2e-16 ***
Boost_Clock       0.42684    0.04682   9.117  < 2e-16 ***
Core_Speed       -0.26378    0.02340 -11.272  < 2e-16 ***
Max_Power         0.09984    0.01471   6.785 1.45e-11 ***
Memory            0.13632    0.01170  11.654  < 2e-16 ***
Memory_Bandwidth  0.16804    0.01650  10.187  < 2e-16 ***
Memory_Bus        0.09097    0.02058   4.420 1.03e-05 ***
Memory_Speed     -0.32002    0.02988 -10.711  < 2e-16 ***
Shader           -0.60689    0.04799 -12.645  < 2e-16 ***
TMUs              0.05912    0.01235   4.788 1.79e-06 ***
---
Signif. codes:  0   ***   0.001   **   0.01   *   0.05   .   0.1       1

Residual standard error: 0.2973 on 2412 degrees of freedom
Multiple R-squared:  0.5622,  Adjusted R-squared:  0.5606
F-statistic: 344.2 on 9 and 2412 DF,  p-value: < 2.2e-16
```

Listing 9: Summary of linear model

It can be observed that no variable have **p-value** $\geq$ **0.05**, which means all predictors that we have chosen is involved in the building process.

4.3.2 Homoscedasticity checking

Now we must check for Multi-Linear model assumption:

1. Residual Errors have a Mean Value of Zero.
2. Residual Errors have Constant Variance.
3. The errors are normally distributed.

Testing for Residual Errors have a Mean Value of Zero: We can check the residuals by plotting their histograms and distributions.

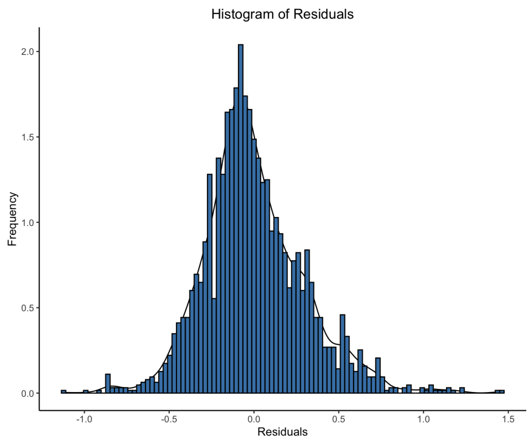


Figure 6: *Histogram of residuals*

From the above figure, we can see a “fair” normal distribution because residuals distribute mainly around value 0. Based on these residuals, we can say that our model meets the assumption.

Testing for Residual Errors have Constant Variance: We can check this assumption using the Scale-Location plot. In this plot we can see the fitted values vs the square root of the standardized residuals. Ideally, we would want to see the residual points equally spread around the red line, which would indicate constant variance.

```r
plot(mdl_price_vs_all, which = 3)
```

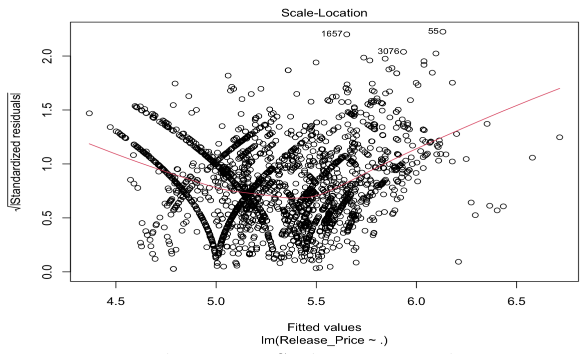


Figure 7: *Scale-Location plot*

In the above plot, the residuals scatter is not following any formal distribution and it do spread around the red line. So the assumption is approved.

Testing for normality of the the errors: To check this, we have to use Q-Q plot for normality consideration. The output we expect that the residuals will mostly scatter close to the straight line to get normality hypothesis accepted.

```r
plot(mdl_price_vs_all, which = 2)
```

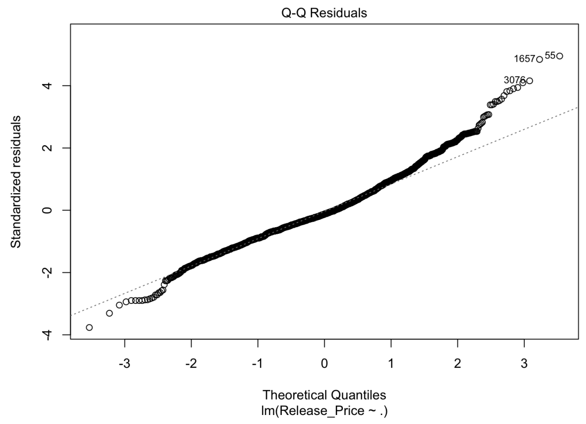


Figure 8: *Q-Q plot*

In the Q-Q plot, it can be seen that those points are lying near the line, while only a few points are not lying near the line. We can conclude that this model can be normally accurate.

4.3.3 Linear equation

```
> coef(mdl_price_vs_all)
     (Intercept)       Boost_Clock        Core_Speed         Max_Power            Memory
      4.40184988        0.42684428       -0.26377554        0.09983657        0.13631975
Memory_Bandwidth        Memory_Bus      Memory_Speed            Shader              TMUs
      0.16804059        0.09096643       -0.32001565       -0.60688925        0.05911582
```

Listing 10: Coeff() function

From listing 10, we can conclude that our linear equation will be(since we did log-transform at the beginning, so we should add exp and log to the equation):

```
Release_Price=exp(4.40184988+0.42684428*log(Boost_Clock)-0.26377554*log(Core_Speed)
+0.09983657*log(Max_Power)+0.13631975*log(Memory)+0.16804059*log(Memomy_Bandwidth)
+0.09096643*log(Memory_Bus)-0.32001565*log(Memory_Speed)-0.60688925*log(Shader)
+0.05911582*log(TMUs))
```

4.3.4 Model prediction

We will do the scatter plotting for the predicted value of test set compared with real values of the dataset using plot() function for more clear vision of this trend.

```
comtab.lr <- test['Release_Price']
comtab.lr['predicted_RP'] <- as.data.frame(predict(mdl_price_vs_all, newdata =
test))
# Plotting
# The majority of points lie near the line, so its ok.
ggplot(comtab.lr, aes(x = Release_Price, y = predicted_RP)) +
  geom_point(shape=1, color="blue") +
  geom_abline(mapping=aes(intercept= 0, slope = 1), color="darkblue") +
  labs(x = "log(Release_Price)", y = "Predicted")
```

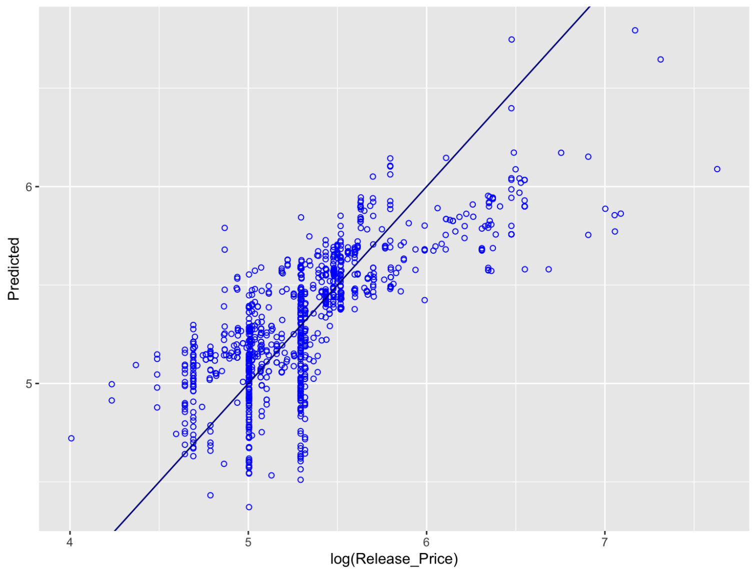


Figure 9: *Prediction plot*

The plotted line in graph is (d) : y = x. The more concentration on this line the more correct the model does. To be more specific, we will show 10 values with their predicted :

```
pred <- data.frame(predict(mdl_price_vs_all, newdata = test))
compare <- cbind(test$Release_Price,pred)
```

```
colnames(compare) <- c("test_set","prediction")
head(compare,10)
```

```
   test_set  prediction
1  5.433722    5.463404
4  5.003946    4.816500
17 6.344759    5.953211
23 4.938065    5.532841
24 4.938065    5.476956
38 4.938065    5.540844
42 5.102911    5.168405
46 7.089243    5.862903
47 6.107023    5.754647
51 6.520621    5.968979
```

Listing 11: Comparison of testing and predicting value

The top 10 above predicted values are not really close to the test_set and the accuracy of the whole test set is only 56.73%, which means this model is not **accurate**.

```
> SSE <- sum((test$Release_Price - pred)^2)
> SST <- sum((test$Release_Price - mean(test$Release_Price))^2)
> cat("The accuracy of the model on test set: " , round((1 - SSE / SST )* 100 ,2)
, "%" )
The accuracy of the model on test set:  56.73 %
```

5 Discussion and extension

5.1 Multiple Linear Regression

5.1.1 Advantages

The biggest advantage of linear regression models is linearity: It makes the estimation procedure simple and, most importantly, these linear equations have an easy to understand interpretation on a modular level. The mathematical equation of Linear Regression is also fairly easy to understand and interpret.

5.1.2 Disadvantages

Multiple linear regression, like any statistical method, has its limitations and disadvantages. Here are some of them:

1. `Assumption of Linearity:` Multiple linear regression assumes that the relationship between the independent variables and the dependent variable is linear. If this assumption is violated, the model's predictions may be inaccurate.

2. `Assumption of Independence:` Multiple linear regression assumes that the independent variables are independent of each other. If there is multicollinearity (high correlation) among the independent variables, it can lead to unreliable estimates of the regression coefficients.

3. `Interpretability:` As the number of independent variables increases, it becomes more challenging to interpret the coefficients of the model. Understanding the individual impact of each independent variable on the dependent variable becomes less straightforward.

4. `Sensitive to Outliers:` Multiple linear regression can be sensitive to outliers, which are data points that deviate significantly from the rest of the data. Outliers can have a disproportionate influence on the estimated regression coefficients and may lead to misleading results.

5. `Assumption of Homoscedasticity:` Multiple linear regression assumes that the residuals (the differences between the observed and predicted values) have constant variance across all levels of the independent variables. Violation of this assumption, known as heteroscedasticity, can lead to biased standard errors and confidence intervals.

6. `Non-linearity of the Relationship:` While multiple linear regression assumes a linear relationship between the independent variables and the dependent variable, this may not always be the case in reality. In such situations, the model may not accurately capture the true relationship between the variables.

Linear regression may not be the appropriate choice in certain situations, particularly when the assumptions of the model are violated or when the relationship between the variables is not linear. For instance, when dealing with data that exhibit a nonlinear relationship, such as exponential or polynomial patterns, linear regression may yield inaccurate predictions and unreliable parameter estimates. Because of that, we will try other models in the **extension** part.

5.2 Extension

5.2.1 SVM model

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. In the project, we will try to build SVM model.

```r
library(e1071)
svm_model = svm(Release_Price~. , data=train)
summary(svm_model)
svm_prediction_linear <- predict(svm_model)
pred_svm <- data.frame(predict(svm_model, newdata = test))
compare_2 <- cbind(test$Release_Price,pred_svm)
colnames(compare) <- c("test_set","prediction")
```

```
head(compare,10)
```

```
   test_set prediction
1  5.433722   5.386743
4  5.003946   5.180472
17 6.344759   5.701946
23 4.938065   5.114874
24 4.938065   5.100951
38 4.938065   5.190016
42 5.102911   5.133553
46 7.089243   6.189586
47 6.107023   6.129083
51 6.520621   6.472899
```

Listing 12: Comparison of testing and predicting value

```
> predictions_svm <- predict(svm_model, newdata = test)
> SSE_svm <- sum((predictions_svm - test$Release_Price)^2)
> SST_svm <- sum((test$Release_Price - mean(test$Release_Price))^2)
> r2_score <- 1 - SSE_svm / SST_svm
> print(paste("The accuracy of the model on test set: ",r2_score))
[1] "The accuracy of the model on test set:  0.83715919086462"
```

After modelling, when comparing test_set and prediction, we can see that the prediction values have a small residual due to actual value. Even more, the accuracy is **84%**, which means this model is more reliable than MLR.
Now we will check whether the residuals of the model is normally distributed or not using Q-Q plot.

```
comtab.svm <- test['Release_Price']
comtab.svm['predicted_RP'] <- as.data.frame(predict(svm_model, newdata = test))
# Plotting
residuals <- comtab.svm$Release_Price - comtab.svm$predicted_RP
qqnorm(residuals)
qqline(residuals)
```

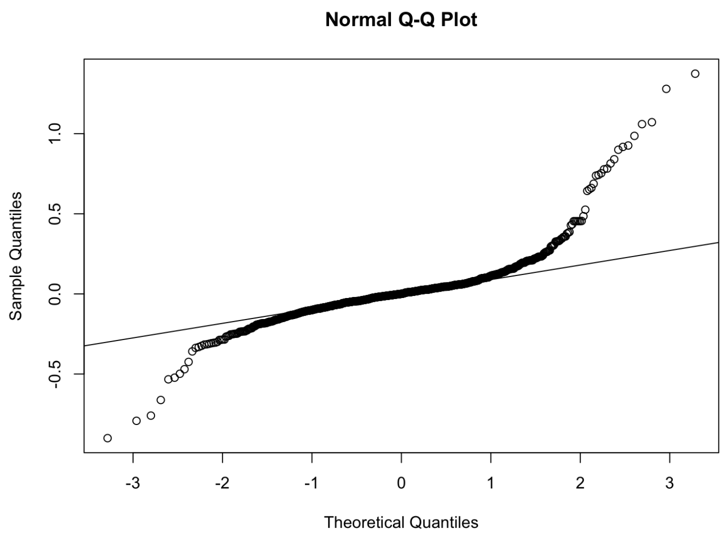


Figure 10: *SVM prediction plot*

Although all points are not fully lying near the line, most of them does. We conclude that this model is normally accurate.
We will do the scatter plotting for the predicted value of test set compared with real values of the test data set using plot() function:

```r
ggplot(comtab.svm, aes(x = Release_Price, y = predicted_RP)) +
  geom_point(shape=1, color="blue") +
  geom_abline(mapping=aes(intercept= 0, slope = 1), color="darkblue") +
  labs(x = "log(Release_Price)", y = "Predicted_RP")
```

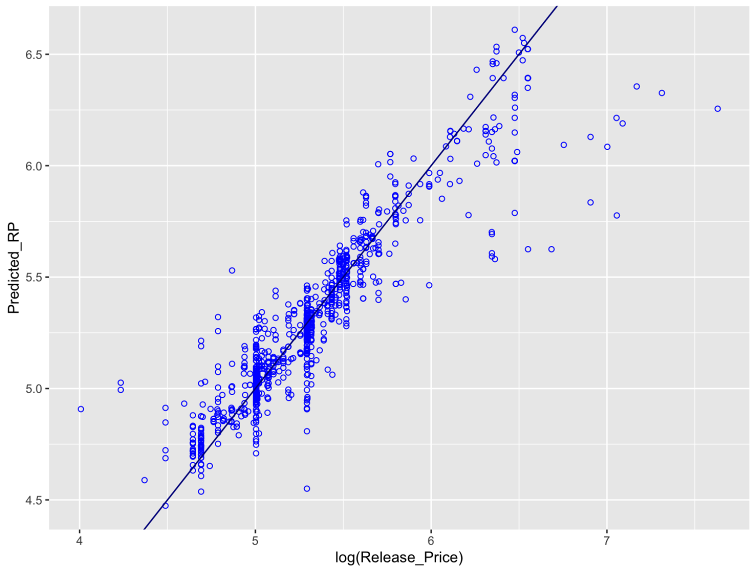


Figure 11: *SVM prediction plot*

As we can see, the closer the points to the line, the more accurate the prediction, and we clearly see, similarly to the previous model, there are more points close to the line than MLR, which is explained SVM is better than MLR.

5.2.2 Random Forest Regression

Random Forest Regression models are often used for predicting system performance metric. This algorithm can handle both continuous and categorical data. First, we will build the Random Forest regression model:

```r
library("randomForest")
model.rfr <- randomForest(formula = Release_Price ~ ., data = train, ntree = 500)
print(model.rfr)
```

```
Call:
 randomForest(formula = Release_Price ~ ., data = train, ntree = 500)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 0.02319768
                    % Var explained: 88.46
```

Listing 13: Summary of Random Forest model

In the output above, the “% Var explained” (percentage of variance explained) is a measure of the amount of variation in the target variable that is explained by the random forest regression model. Specifically, it represents the percentage of the total variance in the target variable that is accounted for by the model, and a higher percentage of variance explained is considered better, as it indicates that the model is able to explain a larger proportion of the variation in the target variable.
A % variance explained of **88.46%** is generally considered to be a good result for a random forest regression model, as it suggests that the model is able to explain a significant amount of the variation in the target variable.
Now, we will check the accuracy of this model using **r2_score**:

```
> comtab.rfr <- test['Release_Price']
> comtab.rfr['RP_predicted'] <- as.data.frame(predict(model.rfr, newdata = test),
row.names = NULL)
> # Evaluate model performance
> SSE <- sum((comtab.rfr$Release_Price - comtab.rfr$RP_predicted)^2)
> SST <- sum((comtab.rfr$Release_Price - mean(comtab.rfr$Release_Price))^2)
> cat("The accuracy of the model on test set: " , round((1 - SSE / SST )* 100 ,2)
, "%" )
The accuracy of the model on test set:  88.15 %
```

With the accuracy of 88.15%, it is clear that this random forest regression model is better than SVM model and Multi-Linear Regression in overall. However, it is required to check whether this model's residual is normally distributed. Once again, we will use the Q-Q plot.

```
residuals2 <- comtab.rfr$Release_Price - comtab.rfr$RP_predicted
qqnorm(residuals2)
qqline(residuals2)
```

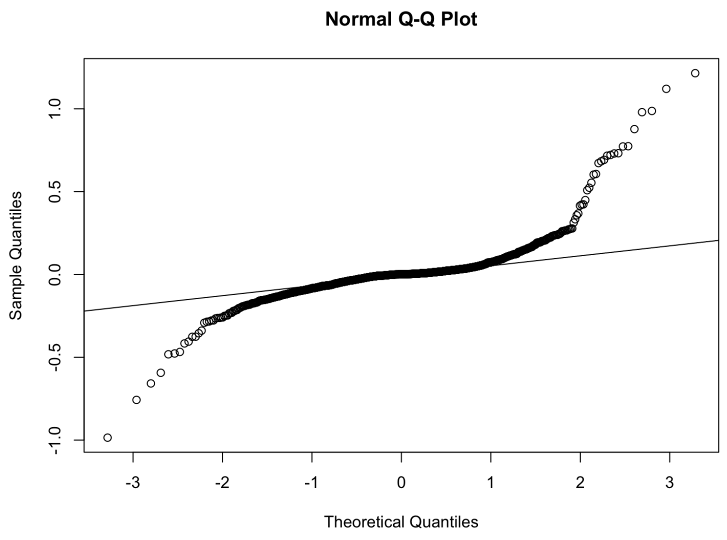


Figure 12: *Q-Q plot of random forest regression*

According to the above plot, we can conclude that this model is normally distributed while there are some outliers.
We will do the scatter plotting for the predicted value of test set compared with real values of the test data set using plot() function:

```
# Plot the predicted - actual
ggplot(comtab.rfr, aes(x = Release_Price, y = RP_predicted)) +
  geom_point(shape = 1, color = "blue") +
  geom_abline(mapping = aes(intercept = 0, slope = 1), color = "darkblue") +
  labs(x = "log(Release_Price)", y = "RP Predicted")
```

In comparison to other prediction plot, it is clear that the random forest prediction plot below has more points closer to the line than two other models, which means theoretically **Random Forest is the best model of all three**.

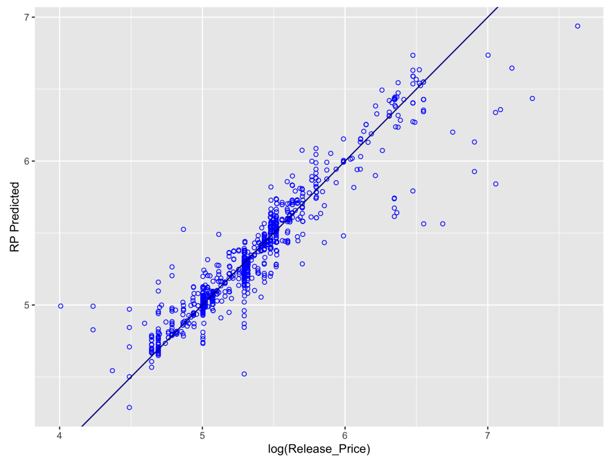


Figure 13: *Random Forest prediction plot*

5.2.3 Comparison

To determine between multi-linear regression, SVM regression and Random Forest Regression which one is more efficient exactly, we will consider the rate of accuracy of two models(calculated by subtracting SSE from SST, dividing the result by SST, and then multiplying by 100 to express the accuracy as a percentage).As can be seen above we have calculated and show the results of each model.

```
The accuracy of the model on test set: 56.73% #multi-linear
The accuracy of the model on test set: 83.72% #SVM
The accuracy of the model on test set: 88.15% #Random Forest
```

Since **Random Forest** has a better accuracy rate, we should check their RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) for a better evaluation.

```
model1 <- mdl_price_vs_all #multi-linear
model2 <- svm_model #svm
model3 <- model.rfr #random forest
predictions_m1 <- predict(model1, test)
predictions_m2 <- predict(model2, test)
predictions_m3 <- predict(model3, test)
# Calculate RMSE (Root Mean Squared Error)
rmse_m1 <- sqrt(mean((test$Release_Price - predictions_m1)^2))
rmse_m2 <- sqrt(mean((test$Release_Price - predictions_m2)^2))
rmse_m3 <- sqrt(mean((test$Release_Price - predictions_m3)^2))
# Calculate MAE (Mean Absolute Error)
mae_m1 <- mean(abs(test$Release_Price - predictions_m1))
mae_m2 <- mean(abs(test$Release_Price - predictions_m2))
mae_m3 <- mean(abs(test$Release_Price - predictions_m3))
```

```
> paste("RMSE for model1: ", rmse_m1)
[1] "RMSE for model1:  0.301143401362998"
> paste("MAE for model1: ", mae_m1)
[1] "MAE for model1:  0.229336454060829"
> paste("RMSE for model2: ", rmse_m2)
[1] "RMSE for model2:  0.18474145956431"
> paste("MAE for model2: ", mae_m2)
[1] "MAE for model2:  0.108300786582277"
> paste("RMSE for model3: ", rmse_m3)
[1] "RMSE for model3:  0.157580846686117"
```

```
> paste("MAE for model3: ", mae_m3)
[1] "MAE for model3:  0.0834563933389319"
```

Model3(Random Forest) significantly outperforms the other two models on all three metrics. It has the lowest RMSE and MAE, meaning the model has the smallest average errors in predicting GPU prices. In conclusion, Random Forest regression has a better performance in GPU release_price prediction than others.

6 Data and code availability

6.1 Dataset

DATASET

6.2 Code

CODE

7 Conclusion

Overall, through working on this project and writing this report, our team has achieved:

- Learn how to handle data and deal with data loss or corruption. At the same time, we also learned graphs related to statistics and how to plot them.
- Learn how to use R in RStudio, and apply it to solve the given data.
- Learn how to transition knowledge from classroom to practice as well as use appropriate testing method in research.
- Learn how to research new knowledge, applying them and implementing the problem solving process facing various obstacles on the way, for example using regression model to predict.
- Learn how to work as a group, to help and support each other to achieve specific goals.

In addition to that, due to the limited personal capacity and knowledge of each member, the report cannot avoid unpredictable errors. Therefore, our group is looking forward to receiving contributions and comments so that we can correct and learn from those mistakes.

8 References

References

[1] D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers*, 7th ed. Kendallville: Wiley, 2018.

[2] T. D. Nguyen and D. H. Nguyen, *Probability – Statistics and Data Analysis*. Ho Chi Minh City: VNUHCM Press, 2020.

[3] Zoumana Keita, *Multiple Linear Regression in R: Tutorial With Examples*, Nov, 2022. [Online]. Available: https://www.datacamp.com/tutorial/multiple-linear-regression-r-tutorial.

[4] *Random Forest Regression in Python Explained* [Online]https://builtin.com/data-science/random-forest-python.

[5] *Usage of KNN* [Online]https://www.ibm.com/docs/en/db2oc?topic=knn-usage.

[6] *Top Evaluation Metrics*[Online]https://www.freecodecamp.org/news/evaluation-metrics-for-regression-problems-machine-learning/

[7] *Log-Trans* [Online]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4120293/

[8] K. Yang, J. Tu and T. Chen, *Homoscedasticity: an overlooked critical assumption for linear regression*, 2019;32:e100148. doi: 10.1136/gpsych-2019-100148.

[9] P. Jonsson and C. Wohlin, “An evaluation of k-nearest neighbour imputation using Likert data,” *10th International Symposium on Software Metrics, 2004. Proceedings.*, 2004, pp. 108-118, doi: 10.1109/METRIC.2004.1357895.

[10] Geeksforgeeks, *Support Vector Machine (SVM) Algorithm*, [Online]. Available: https://www.geeksforgeeks.org/support-vector-machine-algorithm/.