VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PROBABILITY & STATISTICS (MT2013)

Assignment

Predicting CPU's price using linear regression models

Student: Nguyen Minh Khue - 2153488

HO CHI MINH CITY, April 2023

Contents

1 Theory

1.1 Multiple regression

Multiple regression is a statistical technique used to analyze the relationship between a dependent variable Y and two or more independent variables $X_1, X_2, ..., X_p$. The general form of a multiple regression model is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$$

where β_0 represents the intercept, $\beta_1, \beta_2, \ldots, \beta_p$ represent the regression coefficients for the independent variables $X_1, X_2, \ldots, X_p$, and ϵ represents the error term.
The goal of multiple regression analysis is to estimate the regression coefficients that best fit the data, such that the sum of squared residuals is minimized. The regression coefficients provide information about the strength and direction of the relationship between the dependent variable and each independent variable, while the error term represents the unexplained variation in the dependent variable.

1.2 Estimation of parameter: Ordinary Least Square

Ordinary least square (OLS) is a methodology to find the most suitable linear function between a dependent variable and one or more independent variables. Suppose we want to find a relationship between feature y and n other features $x_1, x_2, x_3, ..., x_n$. The linear function we want to figure out has the form:

$$\hat{y} = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

Therefore, the relationship between the real data y and other independent variables can be shown as follow:

$$y_i = \hat{y}_i + \varepsilon_i = \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_n x_{in} + \varepsilon_i$$

If there are m observations, m should be larger or equal to n, We take n linear function with the above pattern. We can represent the set of n linear functions in matrix notation.

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

We can write it shortly as

$$\vec{y} = A\vec{\beta} + \vec{E} \tag{1}$$

The final goal is to find a linear equation in which the total sum of square of error from each observation is the minimum. In other word, we want to find the min of $\left\|\vec{E}\right\|^2$.
The vector norm $\left\|\vec{E}\right\|^2$ can also be written like this:

$$\left\|\vec{E}\right\|^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - \beta_1 x_{i1} - \beta_2 x_{i2} - ... - \beta_n x_{in})^2$$

To find the min of the norm of vector E. Take partial derivative and solve the equation

$$\frac{\partial E}{\partial \beta_k} = 0$$

The general equation for a partial derivative of β_k is:

$$\begin{aligned}
&\sum_{i=1}^{n} -2x_{ik}(y_i - \beta_1 x_{i1} - \beta_2 x_{i2} - ... - \beta_n x_{in}) = 0 \\
\iff &\sum_{i=1}^{n} -x_{ik}y_i + \beta_1 \sum_{i=1}^{n} x_{ik}x_{i1} + ... + \beta_n \sum_{i=1}^{n} x_{ik}x_{in} = 0 \\
\iff &\langle \vec{x_k}, \vec{x_1} \rangle \beta_1 + \langle \vec{x_k}, \vec{x_2} \rangle \beta_2 + ... + \langle \vec{x_k}, \vec{x_n} \rangle \beta_n = \langle \vec{x_k}, \vec{y} \rangle
\end{aligned}$$

To find $\vec{\beta}$, we have to solve this matrix equation:

$$\begin{bmatrix} \langle \vec{x_1}, \vec{x_1} \rangle & \langle \vec{x_1}, \vec{x_2} \rangle & \dots & \langle \vec{x_1}, \vec{x_n} \rangle \\ \langle \vec{x_2}, \vec{x_1} \rangle & \langle \vec{x_2}, \vec{x_2} \rangle & \dots & \langle \vec{x_2}, \vec{x_n} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \vec{x_n}, \vec{x_1} \rangle & \langle \vec{x_n}, \vec{x_2} \rangle & \dots & \langle \vec{x_n}, \vec{x_n} \rangle \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} = \begin{bmatrix} \langle \vec{x_1}, \vec{y} \rangle \\ \langle \vec{x_2}, \vec{y} \rangle \\ \vdots \\ \langle \vec{x_n}, \vec{y} \rangle \end{bmatrix}$$

To make it shorter, using the symbol in equation (1). It leads to this final equation

$$A^T A \vec{\beta} = A^T \vec{y} \tag{2}$$

1.3 Estimating the variance of the error

Consider the linear function

$$y_i = \hat{y}_i + \varepsilon_i = \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_n x_{in} + \varepsilon_i$$

It can be observed that, the error ε varies around the $\hat{y}$. Therefore, it is important to calculate the variance of the error around the linear regression model. The logical estimator for $\hat{\sigma}^2$ is

$$\hat{\sigma}^2 = \frac{\sum\limits_{i=1}^{n} \varepsilon_i^2}{n - (k+1)} = \frac{\sum\limits_{i=1}^{n} (y_i - \hat{y}_i)^2}{n - (k+1)}$$

in which n is the number of observations, k is the number of independent variables. The $\hat{\sigma}^2$ is the unbiased estimator for the population variance σ^2.

1.4 Model Selection

1.4.1 Bayesian Model Averaging

Bayesian model average: A parameter estimate (or a prediction of new observations) obtained by averaging the estimates (or predictions) of the different models under consideration, each weighted by its model probability.
The BMA estimate is obtained through the equation:

$$p(t) = \sum_i p(tH_i)p(H_i)$$

with:

- **t** An unknown quantity
- H_i hypotheses or models Hi that describe the relationship between **t** and the data.

The BMA estimate is adjusted as new information becomes available. This information can be referred to collectively as "data," and the BMA estimate becomes

$$p(tdata) = \sum_i p(tH_i, data)p(H_i data)$$

Bayesian model averaging (BMA) provides a coherent and systematic mechanism for accounting for model uncertainty. It can be regarded as an direct application of Bayesian inference to the problem of model selection, combined estimation and prediction. BMA produces a straightforward model choice criterion and less risky predictions.

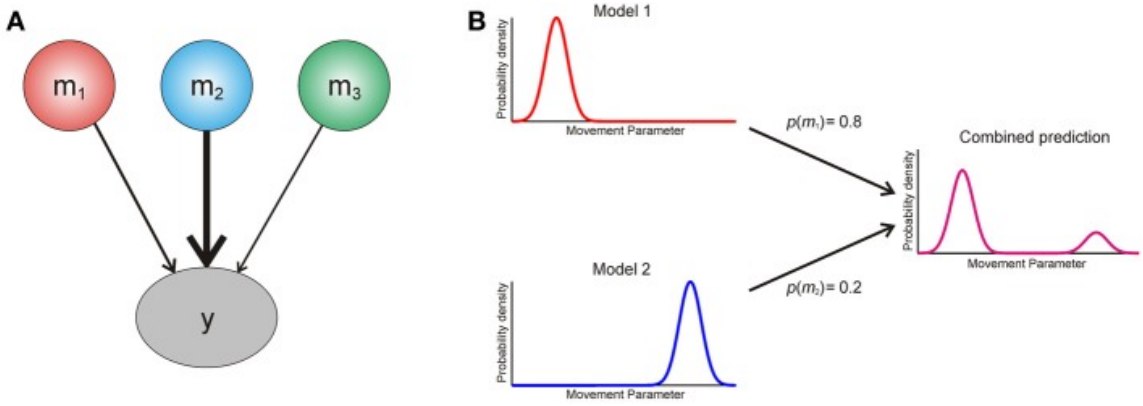

Figure 1: Graphical illustration of Bayesian model averaging, here m_i is a model and y is the value of interest

1.4.2 Multicollinearity

Multicollinearity occurs when independent variables in a regression model are correlated. This correlation is a problem because independent variables should be independent. If the degree of correlation between variables is high enough, it can cause problems when you fit the model and interpret the results.
Multicollinearity causes the following two basic types of problems:

- The coefficient estimates can swing wildly based on which other independent variables are in the model. The coefficients become very sensitive to small changes in the model.
- Multicollinearity reduces the precision of the estimated coefficients, which weakens the statistical power of your regression model. You might not be able to trust the p-values to identify independent variables that are statistically significant.

1.4.3 Overfitting

In linear regression overfitting occurs when the model is "too complex". This usually happens when there are a large number of parameters compared to the number of observations. Such a model will not generalise well to new data. That is, it will perform well on training data, but poorly on test data

1.4.4 Underfitting

Underfitting describes a model which does not capture the underlying relationship in the dataset on which it's trained. An example of underfitting would be a linear regression model which is trained on a dataset that exhibits a polynomial relationship between the input and output variables. Such a model will never be able to adequately capture this relationship (assuming polynomial data features are not used), so the model will underfit and will neither perform well on the training set nor generalize well to unseen data.

1.4.5 Variance inflation factor (VIF)

The variance inflation factor (VIF) will helps us identify correlation between independent variables and the strength of that correlation.

- VIF $= 1$: No correlation
- VIF from 1 to 5 : Moderate correlation
- VIF > 10 : High correlation

1.5 Hypothesis Tests In Multiple Linear Regression

Multiple linear regression models are often used in statistical analysis to explain the relationships between several predictor variables and a single response variable. Hypothesis tests in multiple linear regression can help determine if the relationships between the predictor variables and the response variable are statistically significant.

1.5.1 Null and Alternative Hypotheses

In multiple linear regression, the null hypothesis (H_0) states that the regression coefficients for all predictor variables are equal to zero, indicating that there is no significant relationship between the predictor variables and the response variable. The alternative hypothesis (H_1) states that at least one of the regression coefficients is not equal to zero, indicating that there is a significant relationship between at least one predictor variable and the response variable.

1.5.2 F-Test

The F-test is used to test the overall significance of the multiple linear regression model. It compares the fit of the full model to the fit of a reduced model that only includes the intercept term. The F-test statistic is calculated as:

$$F = \frac{(SSE_{reduced} - SSE_{full})/(p-1)}{SSE_{full}/(n-p)}$$

where SSE is the sum of squared errors, p is the number of predictor variables, and n is the sample size. The F-test statistic follows an F-distribution with $p-1$ degrees of freedom for the numerator and $n-p$ degrees of freedom for the denominator. If the F-test statistic is greater than the critical value from the F-distribution, we reject the null hypothesis and conclude that at least one of the regression coefficients is not equal to zero.

1.5.3 t-Tests

To determine which predictor variables are significant in the multiple linear regression model, we can perform t-tests on the individual regression coefficients. The t-test statistic for a regression coefficient β_j is calculated as:

$$t = \frac{\beta_j}{SE(\beta_j)}$$

where $SE(\beta_j)$ is the standard error of the regression coefficient. The t-test statistic follows a t-distribution with $n-p$ degrees of freedom. We reject the null hypothesis for a given predictor variable if the absolute value of the t-test statistic is greater than the critical value of the t-distribution at the desired significance level.

1.5.4 Conclusion

Hypothesis tests in multiple linear regression provide a way to determine if the relationships between predictor variables and the response variable are statistically significant. The F-test is used to test the overall significance of the model, while t-tests are used to determine the significance of individual predictor variables.

1.6 Assumptions about Linear Regression model

1.6.1 Theory

Assumptions are an essential part of the statistical modeling process. They are a set of conditions that must be met for the model to be valid and reliable. Violation of these assumptions can lead to biased estimates, incorrect conclusions, and invalid inferences. Therefore, it is important to check if the assumptions of the model are fulfilled before interpreting the results.

The four main assumptions of the linear regression model are:

1. Linearity: The relationship between the predictor variables and the outcome variable is linear.
2. Independence: The observations are independent of each other.
3. Normality: The residuals (errors) are normally distributed.
4. Homoscedasticity: The variance of the residuals is constant across all levels of the predictor variables.

1.6.2 Analyze

To check if the assumptions of the linear regression model are fulfilled, there are several diagnostic plots that can be used. These plots provide visual representations of the data and can help identify any patterns or deviations from the assumptions.

- Residuals vs Fitted plot: This plot shows the relationship between the residuals and the predicted values. It is used to check for linearity and homoscedasticity. Ideally, the residuals should be randomly scattered around zero, with no pattern or trend.

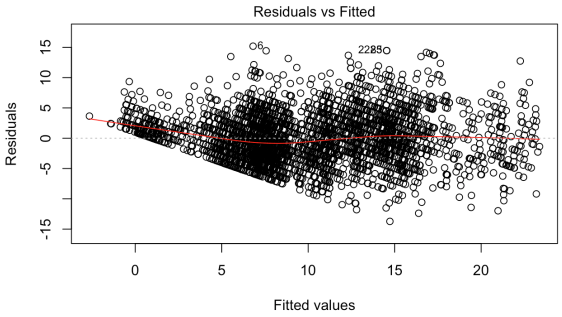

Figure 2: Residuals vs Fitted plot

- Normal Quantile - Quantile plot: This plot shows the distribution of the residuals compared to a normal distribution. It is used to check for normality. Ideally, the points should fall on a straight line, indicating that the residuals are normally distributed.

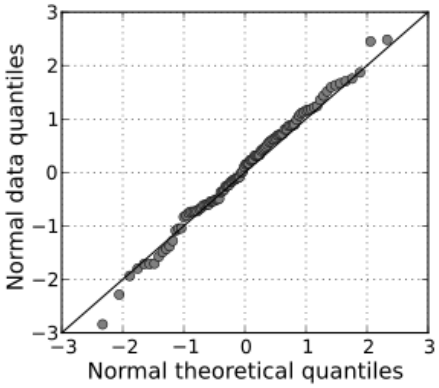

Figure 3: QQ plot

- Scale - Location plot A scale-location plot is a type of plot that displays the fitted values of a regression model along the x-axis and the the square root of the standardized residuals along the y-axis. This particular type of plot is used to ensure constancy in variance of error terms

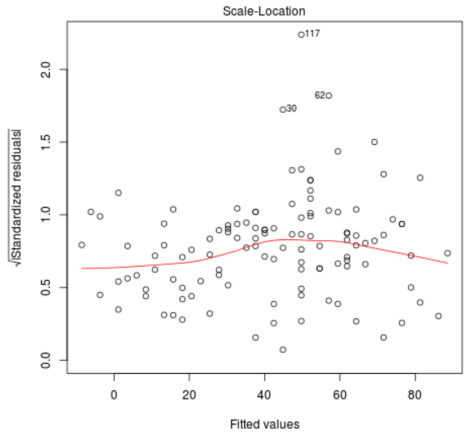

Figure 4: Scale - Location plot

- Residuals - Leverage plot A residuals vs. leverage plot is a type of diagnostic plot that allows us to identify influential observations in a regression model. Each observation from the dataset is shown as a single point within the plot. The x-axis shows the leverage of each point and the y-axis shows the standardized residual of each point

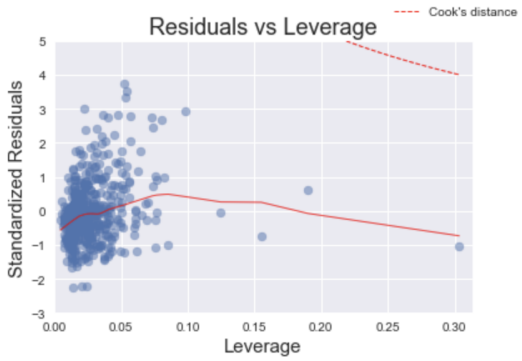

Figure 5: Residuals - Leverage plot

1.7 Interpret graphs

1.7.1 Box plot

Box plot, sometimes called box-and-whisker plot, is a type of graphical display than can simultaneously describes several important features of a data set, such as center, spread, departure from symmetry, and identification of unusual observations or outliers. Box plot has the shape of a rectangle with two lines sketched from two side of the rectangle. As can be seen below

Figure 6: Box plot

The plot can be aligned either vertically or horizontally. The rectangle encloses the inter-quartile range with the left (or lower) is at the first quartile and the right (or higher) is at the third quartile. Inside the box, there is the line represented for the median of the data set. Two lines extend from two side of the box are the whiskers. The lower whisker start from the smallest data point calculated by taking the first quartile subtracts 1.5 inter-quartile range. The higher whisker starts from the third quartile to the point which is calculated by taking the addition of the third quartile and 1.5 inter-quartile range. A point beyond a whisker, but less than three interquartile ranges from the box edge, is called an outlier. A point more than three interquartile ranges from the box edge is called an extreme outlier.

1.7.2 Histogram

A histogram is a graphical representation of a dataset's distribution. It helps estimate the probability distribution of a continuous variable. Histograms show the shape of the data's distribution, its central tendency, and its variability or dispersion. They allow us to see the frequency distribution of a set of continuous data, which is essential in understanding the underlying pattern of the data. Additionally, histograms can identify any outliers or gaps in the data.

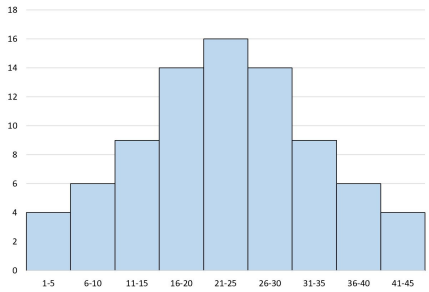

Figure 7: Box plot

1.7.3 Scatter plot

A scatter plot is a type of graph used to display the relationship between two variables. It is commonly used in data analysis and statistics to visually examine the correlation or association between two variables.

By providing a clear and concise representation of data, scatter plots allow for easy interpretation of complex datasets, making them an invaluable tool in the field of data analysis.

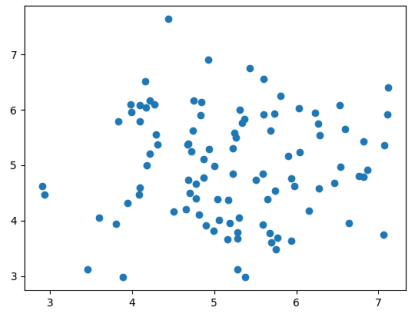

Figure 8: Box plot

1.7.4 Correlation plot

A correlation plot is a graphical representation of the relationship between two variables, typically using a scatter plot or a heatmap. It is a commonly used tool in exploratory data analysis to visually identify the strength and direction of the relationship between two variables. Correlation plots are often used in the context of linear regression to assess the relationship between the independent variables and the dependent variable.

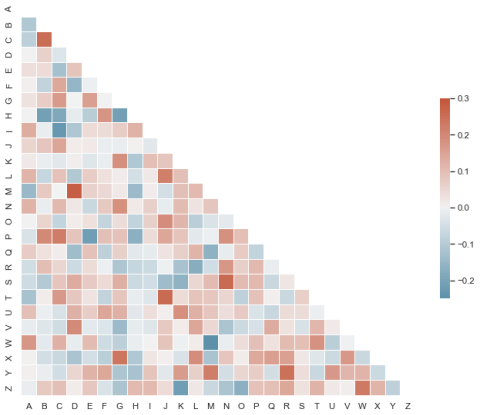

Figure 9: Box plot

2 Main Activity

2.1 Introduction

The dataset contains information about a set of CPUs (Central Unit Processors) sold in the market in the past until now and their performance in computational benchmarks relative to each other to predict their price using linear regression models.
The original data is provided at:
https://www.kaggle.com/datasets/alanjo/cpu-benchmarks

2.2 Data Description

Explanation of the variables in the dataset:

- **passMark**: A benchmark software that's used to put the processor under heavy stress by running complicated algorithms to utilize all the cores and threads of the CPU, and by doing so it shows how fast the CPU can calculate relative to each other through a system of scores. For example, if CPU 1 has a cpuMark score of 1000 and CPU 2 has a score of 2000, it means that CPU 2 can finish the multi-threaded calculation in half the amount of time of CPU 1.
- **cpuName**: The model name of the CPU
- **price**: Last seen market price, in USD ($)
- **cpuMark**: Multi thread CPU Mark (Multi threaded tasks that can be referenced with this metric in real computer usage is video editing apps like Adobe Premiere, DaVinci Resolve, code compilation like Chromium, streaming, 3D modelling like autoCAD and AI / Machine Learning applications which will reflect the CPU's ability to do intensive tasks for users with specialized working fields)
- **cpuValue**: Multi thread CPU Mark / Price, reflects the value the user is getting for their money for intensive multi threaded tasks to be used in their specialized jobs
- **threadMark**: Single thread CPU Mark (Single threaded tasks that can be referenced with this metric in real computer usage is photo editing apps like Adobe Photoshop, office suit apps like Microsoft 365, web browsing etc. for the average computer user like students, office workers, teachers).
- **threadValue**: Single thread CPU Mark / Price, reflects the value the user is getting for their money for single threaded tasks in their daily usage
- **TDP**: thermal design power, in W, show how much power the CPU uses to run
- **powerPerf**: Power Performance = CPU Mark / TDP, reflects how efficient the CPU is at using the power given to it
- **cores**: Number of logical cores of a CPU
- **testDate**: date of first benchmark submission

2.3 Import data

Firstly, let's import our data with **readxl** library, use the command **read_csv** to read the data:

```
install.packages("readxl")
library(readxl)
cpuInfo <- read.csv("/Users/khuenguyen/Desktop/CPU_benchmark_v4.csv")
View(cpuInfo)
```

First glance at the inputted data:

	cpuName	price	cpuMark	cpuValue	threadMark	threadValue	TDP	powerPerf	cores	testDate	socket	category
1	AMD Ryzen Threadripper PRO 5995WX	NA	108822	NA	3330	NA	280	388.65	64	2022	sWRX8	Desktop
2	AMD EPYC 7763	7299.99	88338	12.10	2635	0.36	280	315.49	64	2021	SP3	Server
3	AMD EPYC 7J13	NA	86006	NA	2387	NA	NA		64	2021	unknown	Server
4	AMD EPYC 7713	7060.00	85861	12.16	2727	0.39	225	381.6	64	2021	SP3	Server
5	AMD Ryzen Threadripper PRO 3995WX	6807.98	83971	12.33	2626	0.39	280	299.9	64	2020	sWRX8	Desktop
6	AMD Ryzen Threadripper 3990X	8399.69	81568	9.71	2569	0.31	280	291.31	64	2020	sTRX4	Desktop
7	AMD Ryzen Threadripper PRO 5975WX	NA	80842	NA	3340	NA	280	288.72	32	2022	sWRX8	Desktop
8	AMD EPYC 7B13	NA	77460	NA	2564	NA	NA		60	2021	unknown	Server
9	AMD EPYC 7643	5424.99	76455	14.09	2695	0.50	225	339.8	48	2021	SP3	Server
10	AMD EPYC 7702	4000.00	71646	17.91	2097	0.52	200	358.23	64	2020	SP3	Server
11	AMD EPYC 7662	6817.00	71576	10.50	2054	0.30	225	318.12	64	2021	SP3	Server
12	AMD EPYC 7742	5045.99	68749	13.62	2145	0.43	225	305.55	64	2019	SP3	Server
13	AMD EPYC 75F3	NA	68505	NA	2775	NA	280	244.66	32	2021	SP3	Server
14	AMD Ryzen Threadripper PRO 5965WX	NA	68405	NA	3346	NA	280	244.3	24	2022	sWRX8	Desktop
15	AMD EPYC 7642	3684.00	67748	18.39	2155	0.59	225	301.1	48	2021	SP3	Server
16	AMD EPYC 7543P	2999.00	67144	22.39	2742	0.91	225	298.42	32	2021	SP3	Server
17	AMD EPYC 7R32	NA	64727	NA	1922	NA	NA		48	2020	unknown	Server
18	AMD Ryzen Threadripper 3970X	2997.99	63835	21.29	2694	0.90	280	227.98	32	2019	sTRX4	Desktop
19	AMD Ryzen Threadripper PRO 3975WX	4499.99	63495	14.11	2666	0.59	280	226.77	32	2020	sWRX8	Desktop

Figure 10: Input data

2.4 Process data (Data Cleaning)

As you can see, there are total 12 columns in this data frame, but for this report, we will neglect the **cores** and **socket** attributes as they don't contribute much to our research. Also we don't really need to care about the name of the cpu, and just the important attributes are enough, so cpuName is also neglected. We overwrite the current data frame with the new one containing only the columns of interest:

```
cpuInfo <- cpuInfo[,c("price", "cpuMark", "threadMark", "TDP", "powerPerf","category")]
```

	price	cpuMark	threadMark	threadValue	TDP	powerPerf	category
1	NA	108822	3330	NA	280	388.65	Desktop
2	7299.99	88338	2635	0.36	280	315.49	Server
3	NA	86006	2387	NA	NA		Server
4	7060.00	85861	2727	0.39	225	381.6	Server
5	6807.98	83971	2626	0.39	280	299.9	Desktop
6	8399.69	81568	2569	0.31	280	291.31	Desktop
7	NA	80842	3340	NA	280	288.72	Desktop
8	NA	77460	2564	NA	NA		Server
9	5424.99	76455	2695	0.50	225	339.8	Server
10	4000.00	71646	2097	0.52	200	358.23	Server
11	6817.00	71576	2054	0.30	225	318.12	Server
12	5045.99	68749	2145	0.43	225	305.55	Server
13	NA	68505	2775	NA	280	244.66	Server
14	NA	68405	3346	NA	280	244.3	Desktop
15	3684.00	67748	2155	0.59	225	301.1	Server
16	2999.00	67144	2742	0.91	225	298.42	Server
17	NA	64727	1922	NA	NA		Server
18	2997.99	63835	2694	0.90	280	227.98	Desktop
19	4499.99	63495	2666	0.59	280	226.77	Desktop

Figure 11: New data frame after removing 3 columns

We need to check whether our data has any NA (Not Available) values before diving into analyzing the data. We use the is.na command to check if there is any missing values (Not Applicable).

```
colSums(is.na(cpuInfo))
```

```
> colSums(is.na(cpuInfo))
      price     cpuMark  threadMark threadValue         TDP   powerPerf    category
       1858           0           0        1858         685           0           0
>
```

Figure 12: Checking missing values

Our data is containing a lot of missing numbers from **price**, **threadValue**, **TDP** Now let's see if there is any duplicated rows (or, by other words, duplicated tuples):

```
> library(magrittr)
> duplicated(cpuInfo) %>% sum()
[1] 1
>
```

Figure 13: Checking duplicated values

Luckily, there is only 1 duplicated value in our data set. So far, there has been a few thousands missing values and 1 duplicated value, we shall first omit the NAs, then remove the 1 duplicated row:

```
> cpuInfo <- cpuInfo[!duplicated(cpuInfo), ]
> colSums(is.na(cpuInfo))
      price     cpuMark  threadMark threadValue         TDP   powerPerf    category
          0           0           0           0           0           0           0
> duplicated(cpuInfo) %>% sum()
[1] 0
```

Figure 14: Data after cleaning

So we have basically removed all the irrelevant and incomplete tuples.

2.5 Data Analyze and Visualization

2.5.1 Descriptive statistics

By using this command

```
summary(cpuInfo)
```

The program will produce a summary of the considering data set. Here is the final result

```
   cpuName              price            cpuMark         cpuValue        threadMark
 Length:1900        Min.   :   3.99   Min.   :  159   Min.   :  0.22   Min.   : 224
 Class :character   1st Qu.:  68.03   1st Qu.: 1702   1st Qu.: 12.74   1st Qu.:1096
 Mode  :character   Median : 168.81   Median : 4126   Median : 25.98   Median :1598
                    Mean   : 452.55   Mean   : 8266   Mean   : 35.87   Mean   :1691
                    3rd Qu.: 398.95   3rd Qu.:10143   3rd Qu.: 46.42   3rd Qu.:2208
                    Max.   :8978.00   Max.   :88338   Max.   :345.33   Max.   :4317
 threadValue         TDP           powerPerf          cores          testDate
 Min.   :  0.13   Min.   :  4.00   Min.   :  2.75   Min.   : 1.000   Min.   :2007
 1st Qu.:  4.45   1st Qu.: 40.00   1st Qu.:  28.76   1st Qu.: 2.000   1st Qu.:2011
 Median :  9.73   Median : 65.00   Median :  71.14   Median : 4.000   Median :2014
 Mean   : 15.10   Mean   : 76.28   Mean   : 113.72   Mean   : 5.679   Mean   :2014
 3rd Qu.: 18.96   3rd Qu.: 95.00   3rd Qu.: 155.76   3rd Qu.: 6.000   3rd Qu.:2018
 Max.   :267.82   Max.   :280.00   Max.   : 999.97   Max.   :64.000   Max.   :2022
    socket           category
 Length:1900        Length:1900
 Class :character   Class :character
 Mode  :character   Mode  :character
```

Figure 15: Summary of Data

2.5.2 Box plot

Box plot is a powerful plot that can give the reader a huge amount of information about the data in just one plot. It concentrates on the the range of the value and some critical point of the set of data. In this data set, we will figure out the best features for each CPU platform by taking 8 aspects: *price*, *cpuMark*, *cpuValue*, *threadMark*, *threadValue*, *TDP*, *powerPerf*, and *core* into consideration.

```
# Boxplot for price
pricePlot = ggplot(cpuInfo, aes(price, category, colour=category))+
  geom_boxplot(show.legend = FALSE)+
  labs(x="Platform",y="Price",title="Price for each platform")

# Boxplot for cpuMark
cpuMarkPlot = cpuInfo %>%
  ggplot(aes(cpuMark, category, colour=category))+
  geom_boxplot(show.legend = FALSE)+
```

```
  labs(x="Platform",y="Score",title="Multi Thread Benchmark for each platform")

# Boxplot for cpuValue
cpuValuePlot = ggplot(cpuInfo, aes(cpuValue, category, colour=category))+
  geom_boxplot(show.legend = FALSE)+
  labs(x="Platform",y="Score/Price",title="Multi Thread score per price for each platform")

# Boxplot for thread Mark
threadMarkPlot = ggplot(cpuInfo, aes(threadMark, category, colour=category))+
  geom_boxplot(show.legend = FALSE)+
  labs(x="Platform",y="Score",title="Single Thread benchmark for each platform")

# Boxplot for thread Value
threadValPlot = ggplot(cpuInfo, aes(threadValue, category, colour=category))+
  geom_boxplot(show.legend = FALSE)+
  labs(x="Platform",y="Score/Price",title="Single Thread score per price for each
    platform")
# Boxplot for TDP value
tdpPlot = ggplot(cpuInfo, aes(TDP, category, colour=category))+
  geom_boxplot(show.legend = FALSE)+
  labs(x="Platform",y="Watt",title="Electricity Usage for each platform")
# Boxplot for power performance
powerPerfPlot = ggplot(cpuInfo, aes(powerPerf, category, colour=category))+
  geom_boxplot(show.legend = FALSE)+
  labs(x="Platform",y="Score/Watt",title="CPU strength over electricity usage for each
    platform")

ggarrange(cpuMarkPlot, cpuValuePlot, threadMarkPlot, threadValPlot, ncol=2, nrow=2)
ggarrange(pricePlot, tdpPlot, powerPerfPlot)
```

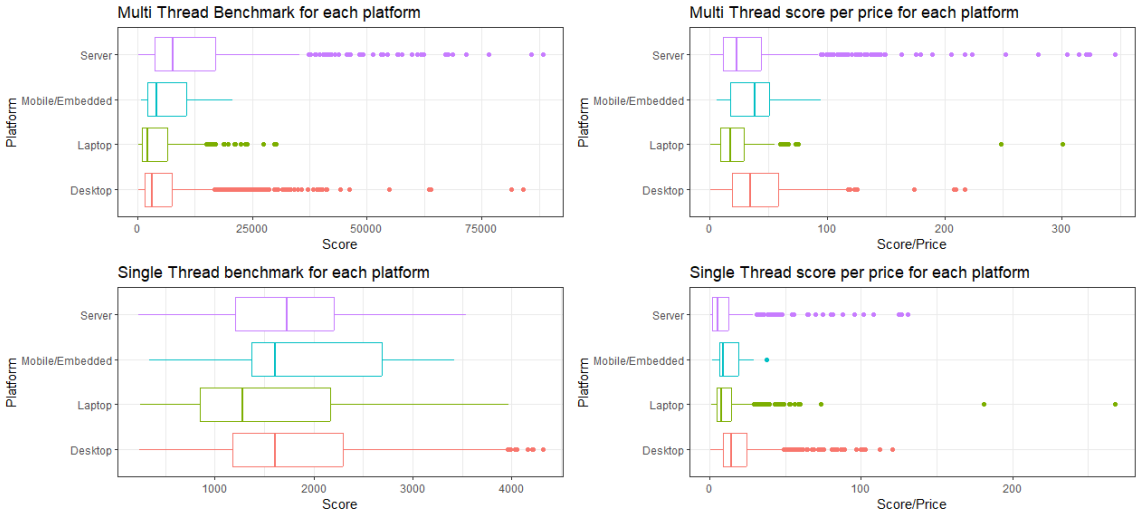

Figure 16: box plot for score and score per price

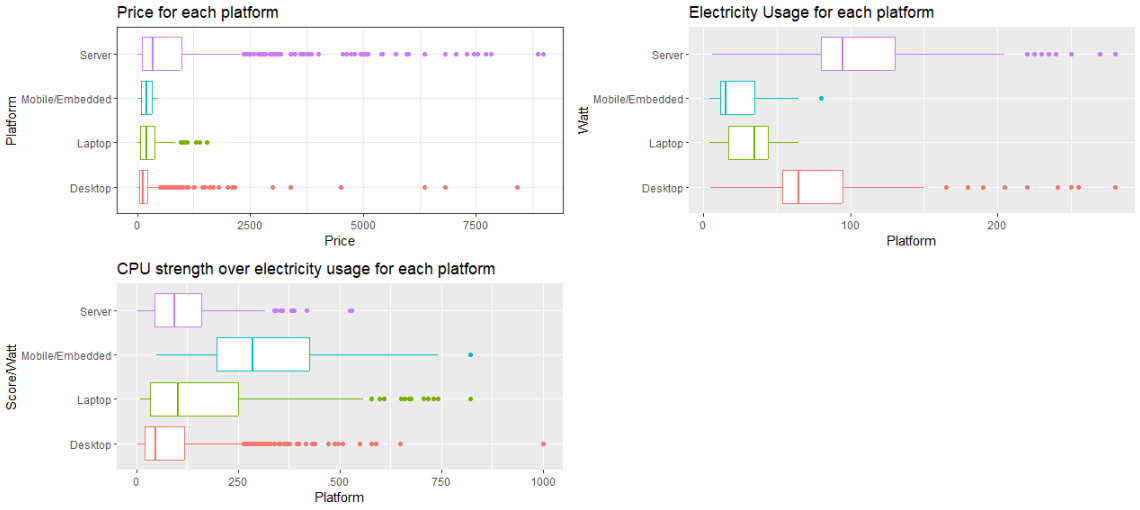

Figure 17: box plot for the other features

From these figures, it can be seen that Server's CPU shows its strength when consider the hardware investment and CPU power on average. Therefore, it leads to the fact Server's CPU is the most expensive and electrical-consuming CPU platform. On the other hand, although desktop and laptop CPU cannot compete with Server platform, they are more suitable for normal user because they are cheaper and have better strength per price. Beside, for device that uses battery, electricity usage is a critical factor. Base on the plot, we can see that Laptop and Mobile/Embedded CPU consume the least electricity and they are utilized for performance with power shortage.

2.5.3 Histogram

The histogram is to display the frequency distribution of CPU prices, it becomes possible to quickly identify which value ranges are the most common and which are the least common among our interested variables. We will plot 5 histograms to view the overall distribution trend of these 5 variables: *price*, *TDP*, *powerPerf*, *cpuMark*, *threadMark*

```
hist(cpuInfo$price, breaks = 20, col = 2, xlab = "Price", xlim = c(0,8000), main =
    "Histogram of CPU price", labels = T)
hist(cpuInfo$cpuMark, breaks = 20, col = 3, xlab = "CPU Mark", xlim = c(0,80000), main =
    "Histogram of CPU Mark", labels = T)
hist(cpuInfo$threadMark, breaks = 20, col = 4, xlab = "Thread Mark", xlim = c(0,4000),
    main = "Histogram of Thread Mark", labels = T)
hist(cpuInfo$TDP, breaks = 20, col = 5, xlab = "TDP",xlim = c(0,250), main = "Histogram of
    TDP", labels = T)
hist(cpuInfo$powerPerf, breaks = 20, col = 6, xlab = "Power Performance",xlim = c(0,1000),
    main = "Histogram of Power Performance", labels = T)
```

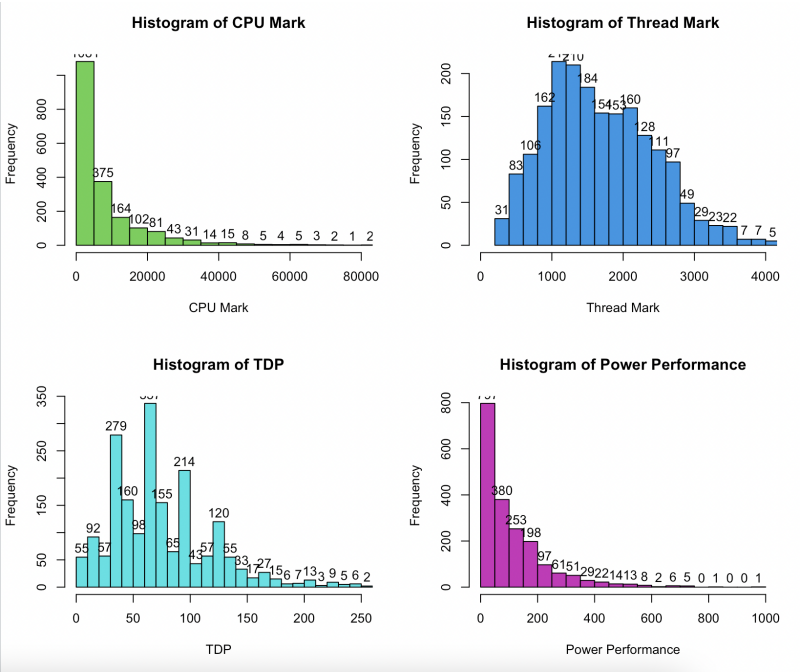

Figure 18: histogram of TDP, powerPerf, cpuMark, threadMark

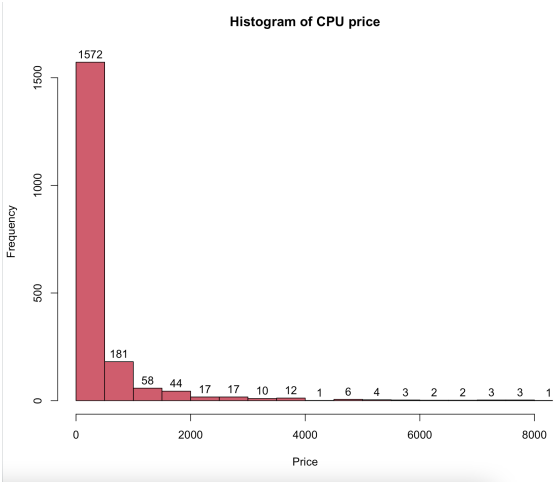

Figure 19: price histogram

- **Price**: The majority of CPU prices fall within the range of 0 to 400, with 1534 instances recorded. This suggests that this price range is the most common for CPUs.
- **CPU Mark**: The CPU Mark values are primarily distributed within the range of 0 to 4000, with a peak occurrence of 1043 instances. This indicates that CPUs with a performance score in this range are the most commonly found.
- **Thread mark**: The histogram is used to display the frequency distribution of Thread Mark scores. Which is particularly useful when evaluating the performance of different CPUs and determining which ones offer the best Thread Mark performance for a given application.
- **Power performance**: The power performance histogram shows that the majority of values fall within the range of 0 and 50, with a peak of 759 values. This suggests that the system is optimized for low power consumption, which can be beneficial for energy-efficient computing applications.
- **TDP**: The histogram is displays the frequency distribution of TDP values, in order to compare and identify the CPUs that consume the least amount of power while still delivering optimal performance. By analyzing the TDP values of different CPUs, individuals can select the most energy-efficient option for their needs, ultimately saving money on energy costs while still achieving high levels of performance.

2.5.4 Scatter plot

Plot 4 scatter plot to show the relationship between *price* and *cpuMark*, *threadMark*, *TDP*, *powerPerf*

```
attach(mtcars)
par(mfrow=c(2,2))
plot(cpuInfo$powerPerf,cpuInfo$price, main="price vs power performance",
```

```
     xlab="powerPerf (point) ", ylab="price (USD)", pch=19)
plot(cpuInfo$cpuMark,cpuInfo$price, main="price vs cpu mark",
     xlab="cpuMark (point) ", ylab="price (USD)", pch=19)
plot(cpuInfo$threadMark,cpuInfo$price, main="price vs thread mark",
     xlab="threadMark (point) ", ylab="price (USD)", pch=19)
plot(cpuInfo$TDP,cpuInfo$price, main="price vs TDP",
     xlab="TDP (W) ", ylab="price (USD)", pch=19)
```

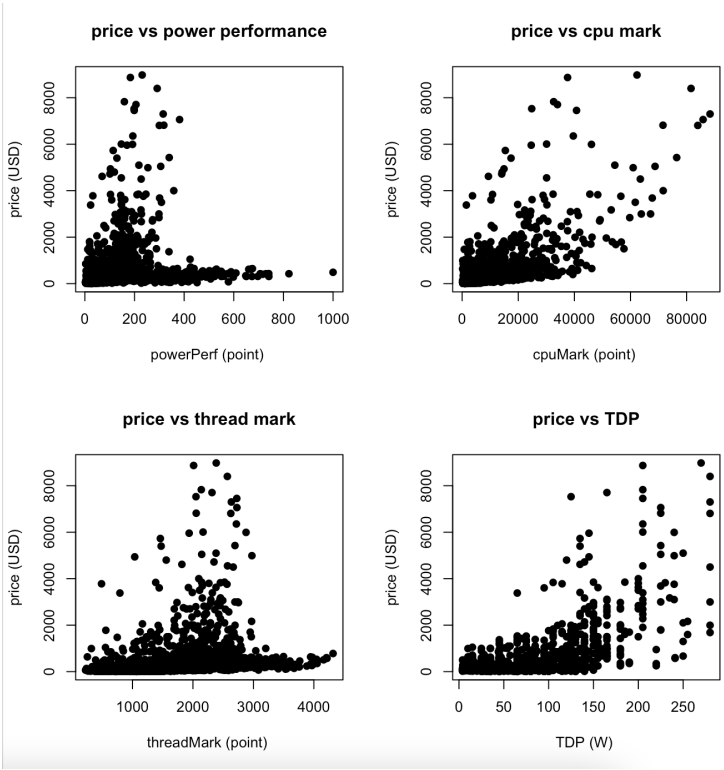

Figure 20: Scatter plot of price according to every specifications

- **price vs threadMark**: Most of the data points are evenly distributed along the x-axis at price below 1000. There are a few outliers that are located at the upper middle of the scatter plot.
- **price vs cpuMark**: Most of the data points are clustered in the bottom left corner of the scatter plot. The rest are randomly distributed everywhere.
- **price vs powerPerf**: Distribution of this plot is similar to that of price&threadMark with most of data point evenly scattered long x-axis at price below 1000 USD while the rest are. With some outliers at small value of powerPerf, indicating that there exist some extremely expensive CPUs that require a lot of energy and are not power efficient.

- **price vs TDP**: Almost all of the CPUs that operates at low TDP have a price below 2000 USD.

Conclusion: When considering all category in 1 graph, it is hard to observe any trend between price and specifications. Because different categories have different criteria for charging price. If we want to construct a linear regression model to predict price according to other specs, we will have to separate the categories.

2.6 Predicting desktop's CPU price using multiple linear regression

2.6.1 Separate category

Based on the Box plots and scatter plots given in previous sections, we can see clearly that: for each category, the distribution of every parameters differ significantly. Furthermore, there is no specific way to determine the exact relationship between the CPU's feature and its category (i.e., how to transform "category" into a meaningful numerical value). Therefore, if we want to build a model for predicting the price of a particular CPU, we have to divide our initial data set based on different categories, and determine which category we want to build a regression model on. Let's try with one of the most popular categories among our students: **laptop**.
The goal of this activity is to build a linear regression model to predict the price of a laptop's CPU based on its specifications. The dependent variable is price, and the independent variables are all the specifications of a CPU (thread mark, CPU mark, power performance, TDP...). Firstly, lets create a new data frame desktop by separating every CPU with category of "Laptop", we can also remove the category column of laptop since we already knew the category:

```
laptop <- (cpuInfo[cpuInfo$category == 'Laptop',])
laptop = subset(laptop, select = c(price, cpuMark, threadMark, powerPerf, TDP, cpuValue,
    threadValue))
View(laptop)
```

Untitled1* × laptop ×

Filter

	price	cpuMark	threadMark	powerPerf	TDP	cpuValue	threadValue
111	635.00	30213	3946	671.40	45	47.58	6.21
118	617.00	29714	3974	660.32	45	48.16	6.44
136	457.00	27391	3766	608.69	45	59.94	8.24
189	583.00	23629	3312	525.09	45	40.53	5.68
193	311.00	23530	3678	522.90	45	75.66	11.83
220	556.00	22570	3298	501.56	45	40.59	5.93
246	546.00	21385	3202	475.22	45	39.17	5.86
248	395.00	21363	3134	474.74	45	54.08	7.93
253	395.00	21212	3219	471.38	45	53.70	8.15
287	311.00	19820	3628	440.44	45	63.73	11.66
309	457.00	19052	3604	423.37	45	41.69	7.89
386	250.00	16458	3148	365.74	45	65.83	12.59
390	395.00	16403	3185	364.51	45	41.53	8.06
391	330.00	16332	2772	466.63	35	49.49	8.40
394	583.00	16271	2869	361.57	45	27.91	4.92
398	250.00	16073	3061	357.17	45	64.29	12.24
407	556.00	15679	2810	348.42	45	28.20	5.05
416	450.00	15426	2778	342.79	45	34.28	6.17

Figure 21: A glance at the laptop data frame

2.6.2 Dealing with outliers

Now that we got all of the CPUs coming from **laptop**, let's use box plots to see to overall description of our desktop data frame and spot any outliers:

```
par(mfrow=c(2,4))
boxplot(laptop$cpuMark)
boxplot(laptop$threadMark)
boxplot(laptop$threadValue)
boxplot(laptop$cpuValue)
boxplot(laptop$TDP)
boxplot(laptop$powerPerf)
boxplot(laptop$price)
```

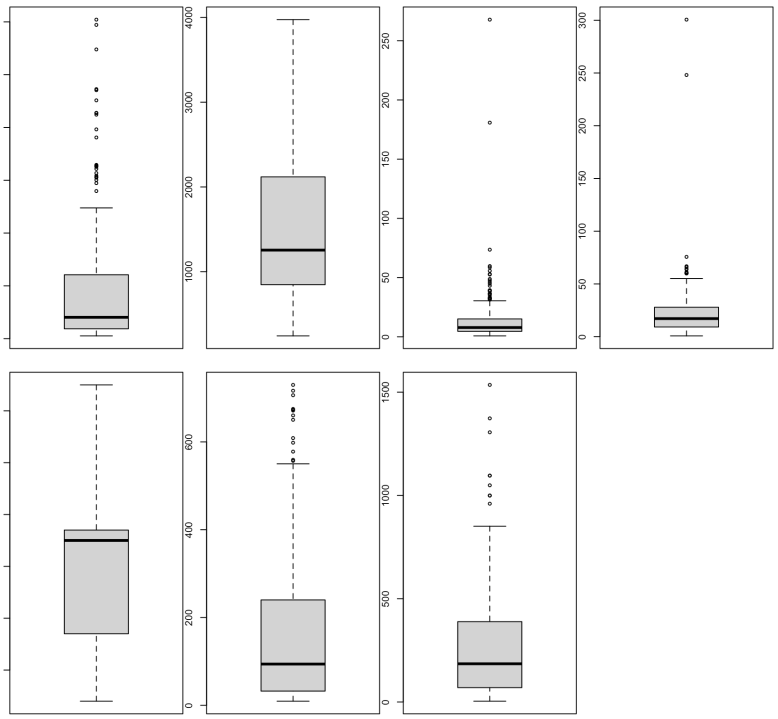

Figure 22: Box plot to show outliers from laptop

As you can see, there seems to be many outliers in every specification, especially in price, CPU mark, thread value and power performance. Because outliers are observations that appear inconsistent with the remainder of the data set which can affect the accuracy of our model, we need to eliminate all the outliers from the data set. This is how we remove the outliers: firstly, we find first (Q1) and third (Q3) quartiles. Then, we find inter quartile range (IQR) by IQR() function. In addition, we calculate Q1 – 1.5*IQR to find lower limit and Q3 + 1.5*IQR to find upper limit for outliers. Then, we use subset() function to remove outliers:

```
quartiles <- quantile(laptop$price, probs=c(.25, .75), na.rm = FALSE)
IQR <- IQR(laptop$price)
Lower <- quartiles[1] - 1.5*IQR
Upper <- quartiles[2] + 1.5*IQR
laptop <- subset(laptop, laptop$price > Lower & laptop$price < Upper)
```

Applying the same algorithm to remove outliers from other specifications, we obtain a more reasonable data frame with less extreme observations:

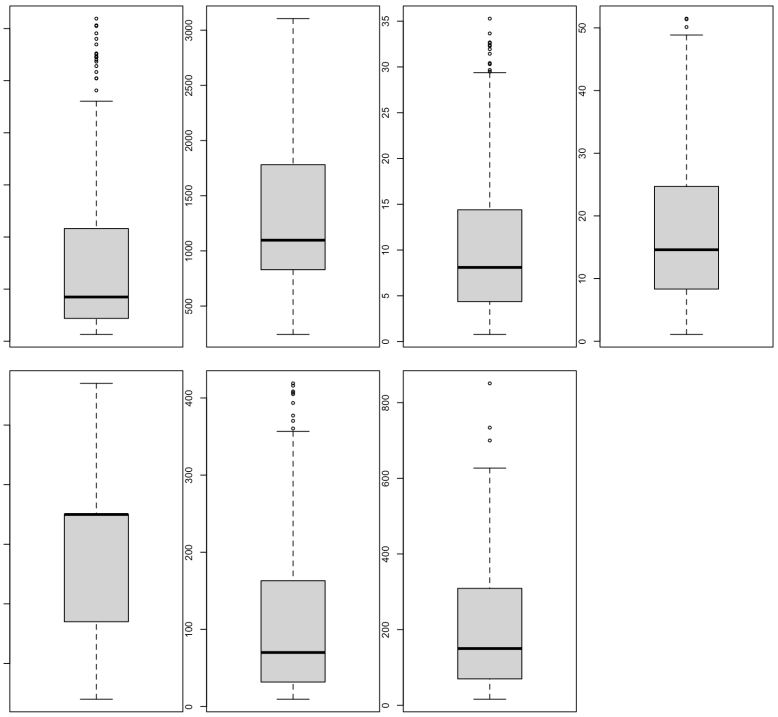

Figure 23: Data after removing all of the outliers

2.6.3 Data transformation

Now that we have obtained a complete data frame, to understand more about the data, we may need to observe the relationship between the variables. Therefore, we construct a pair plot by using the ggpairs() command of ggplot2 library. Further- more, to reduce the complexity of combining geometric objects with transformed data, we combine the ggplot2 library with its extended GGally package and use the command ggpair() to plot:

```
library(ggplot2)
library(GGally)
ggpairs(laptop, columns = 1:7)
```

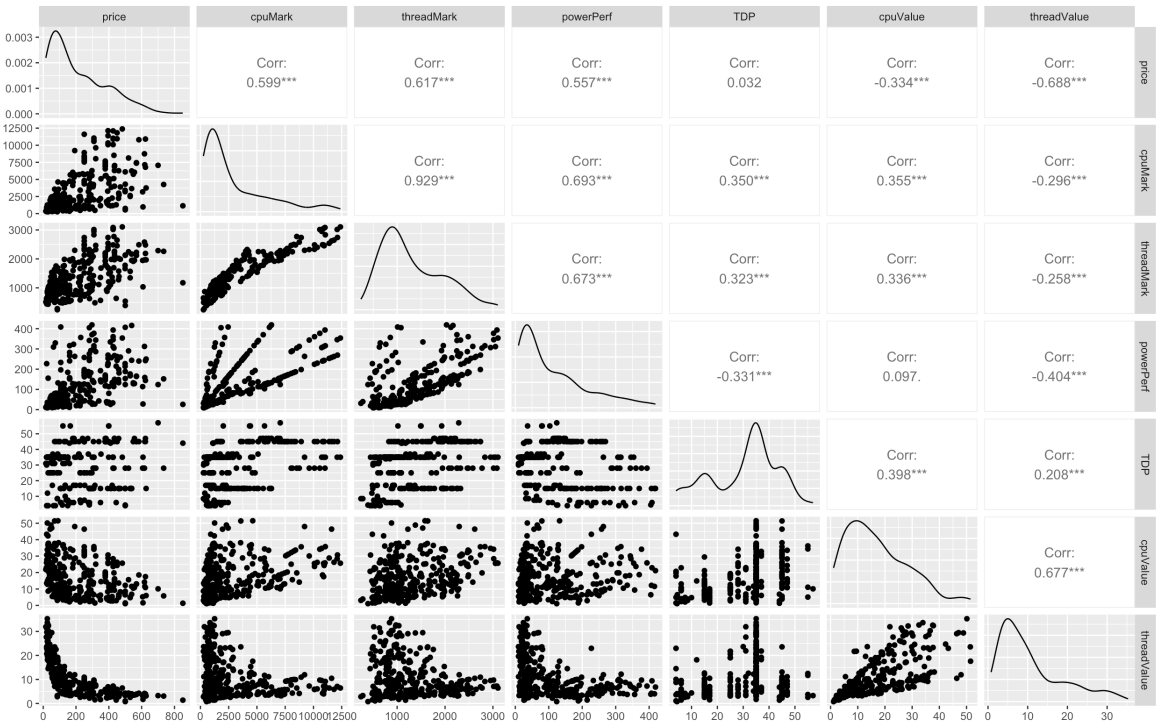

Figure 24: Pair plot

As given by the pair plot, the relationship between a CPU price and its specifications are unclear, rather than a linear relationship, we are seeing more of an exponential relationship in many cases. Furthermore, we can also see that most of the distribution specifications such as price, CPU mark, power performance, CPU value, ,thread value and thread mark are right – skewed. We know that in order to achieve the most accurate linear regression model, it is best if our data comes from a normal distribution. Therefore, to make the right – skewed distribution become more normal, as well as improve linearity between the variable, we apply a logarithmic function to those unevenly distributed specifications. Let's take price for example, we want to construct a new data frame using log(price+1) (the +1 is to make sure we don't get infinite value). Now let's transform price, CPU mark, power performance, cpu value, thread value and thread mark to its log base:

```
newData_2 <- laptop
newData_2[, c("price","cpuMark","powerPerf","threadValue","cpuValue")] <- log(newData_2[,
    c("price","cpuMark","powerPerf","threadValue","cpuValue")]+1)
ggpairs(newData_2, columns = 1:7)
```

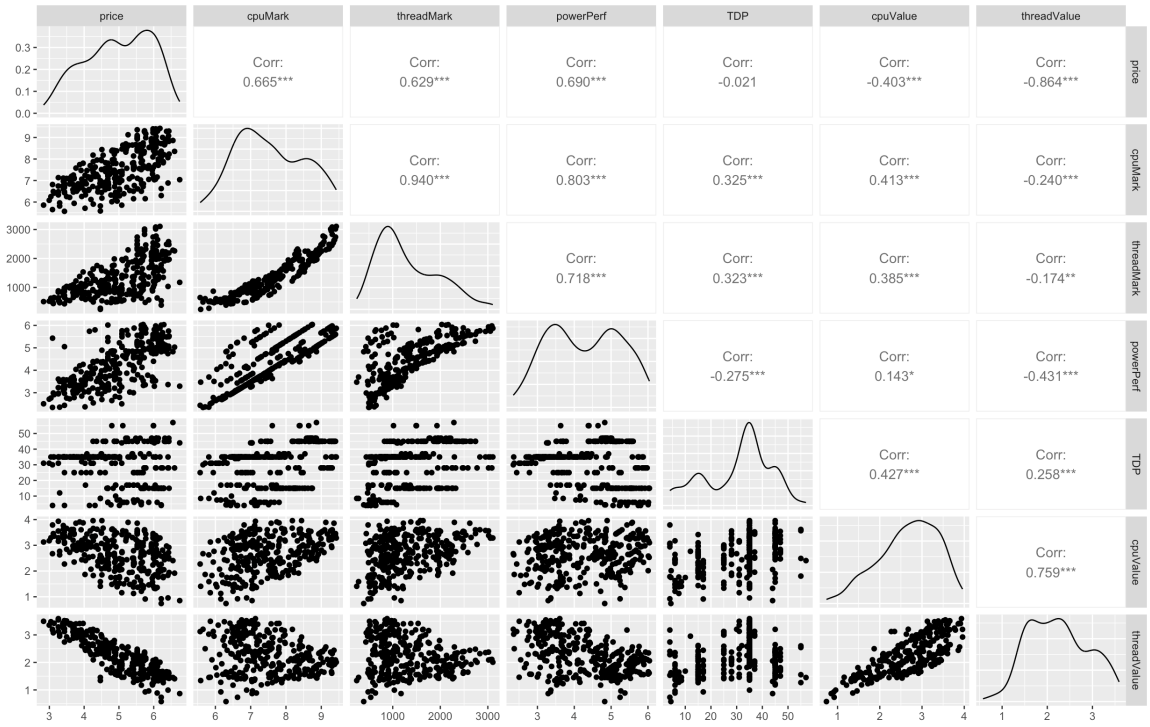

Figure 25: Pair plot after log transformation

As we can see, after log transformation, all of our variables are normally distributed, and also the linearity between price and other specs have significantly improved. Now we can check the correlation between each variable using the correlation heat map. We use functions from the library corrplot and RColorBrewer:

```
library(corrplot)
library(RColorBrewer)
par(mfrow=c(1,1))
CR <- cor(newData_2)
corrplot(CR, type="lowe", order="hclust",col=brewer.pal(n=3, name="RdYlBu"))
```

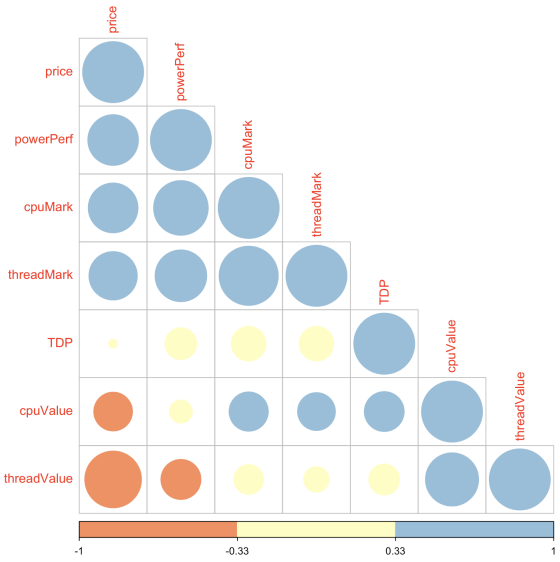

Figure 26: Correlation plot

The figure shows that there is *log(price+1)* has a strong positive correlation with *log(powerPerf+1)*, *log(cpuMark+1)*, *threadMark* and a negative correlation with *log(threadValue+1)* and *log(cpuValue+1)* - this tells us that the more expensive the CPU get, the less performance it actually increases (you may have to pay too much to gain little increase in performance).
Also, the correlation plot shows little to no relationship between price and TDP (manyfacturers tend to add different TDP of CPUs to increase diversity of CPUs at the same price tag so there is no relationship between those 2 value). And if we were to construct a linear regression model to predict *log(price+1)* using TDP, it will recommend us to remove the independent variable:

```
linearModule <-lm(price~TDP,data=train)
summary(linearModule)
```

```
lm(formula = price ~ TDP, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-2.15330 -0.71723  0.03774  0.74458  1.78483

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.043164   0.143991  35.024   <2e-16 ***
TDP         -0.001827   0.004397  -0.416    0.678
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.9244 on 271 degrees of freedom
Multiple R-squared:  0.0006369, Adjusted R-squared:  -0.003051
F-statistic: 0.1727 on 1 and 271 DF,  p-value: 0.678

>
```

Figure 27: linear model using TDP as predictor

As you can see, the p-value for coefficient of TDP being 0 is more than 67% which tells us that there should be no linear relationship between predictor and dependent variable (67% is far more than the significant level in almost all of the cases).

2.6.4 Split train data set & Model selection

Before going straight into finding the model, we will split our data set into two data sets, a training set train and testing set test. The training set will count for 90% of the original data, and testing set will count for 10% rest.

```
sample <- sample(c(TRUE, FALSE), nrow(newData_2), replace=TRUE, prob=c(0.9,0.1))
train <- newData_2[sample , ]
test <- newData_2[!sample ,]
```

Finally, we will use the Bayesian Model Averaging for linear regression models. We will use the function bicreg() to fit many linear regression models for our independent variables and then output the summary. Based on the summary we will select the best multiple linear regression model. But there is a problem relating to semantic feature of the model: because we already knew by the description that CPU value and Thread value are the 2 attributes that is calculated using price, and it would be pointless to try and predict the price of a CPU if we already got its CPU value and (or) Thread value. Therefore, we will build our model using only *cpuMark*, *threadMark*, *powerPerf*, *TDP*.
We will find our model from the training set *train*. Our linear regression model will include:

- Dependent variable: log(price+1)
- Independent variables: log(cpuMark+1), log(powerPerf+1), threadMark, TDP.

```
install.packages("BMA")
library(BMA)
search_for_models = bicreg(subset(train, select = c(cpuMark, threadMark, powerPerf,TDP)),
    train$price, strict=FALSE, OR=20)
summary(search_for_models)
```

```
Call:
bicreg(x = subset(train, select = c(cpuMark, threadMark, powerPerf,     TDP)), y = train$price, strict = FALSE, OR = 20)


  6  models were selected
 Best  5  models (cumulative posterior probability =  0.9573 ):

           p!=0    EV         SD         model 1     model 2     model 3
Intercept  100.0   2.0742433  0.7832509   2.525e+00   1.011e+00   2.411e-01
cpuMark     27.0   0.0940534  0.1928858         .     2.927e-01   7.067e-01
threadMark  73.2   0.0002839  0.0001928   3.970e-04         .           .
powerPerf   94.0   0.4271232  0.1341567   4.493e-01   4.094e-01         .
TDP         14.8  -0.0004725  0.0057151         .           .     -1.959e-02

nVar                                        2           2           2
r2                                          0.485       0.479       0.476
BIC                                        -1.697e+02  -1.670e+02  -1.650e+02
post prob                                   0.641       0.166       0.060
           model 4     model 5
Intercept   2.336e+00   2.145e+00
cpuMark           .     7.624e-02
threadMark  3.338e-04   3.124e-04
powerPerf   4.910e-01   4.300e-01
TDP         3.072e-03         .

nVar            3           3
r2          0.485       0.485
BIC        -1.645e+02  -1.644e+02
post prob   0.046       0.045
>
```

Figure 28: Results given by BMA analyze

From summary of model selection, we see that the program chooses model 1 to be the best model that can describe the relationship between the data points, which chooses to include only log(powerPerf+1) and threadMark, this is reasonable considering the overall trend that people look for when buying their laptop: they want it to be power efficient and have good thread power. "post prob" indicates that the probability of model 1 being a true model that best represent the relationship between the dependent variable and those of the independent (with the probability of 64.1% being the right model after considering data input).
Let's analyze the correlation between these variable using pair plot:

```
ggpairs(subset(train, select = c(price, powerPerf, TDP)), columns = 1:3)
```

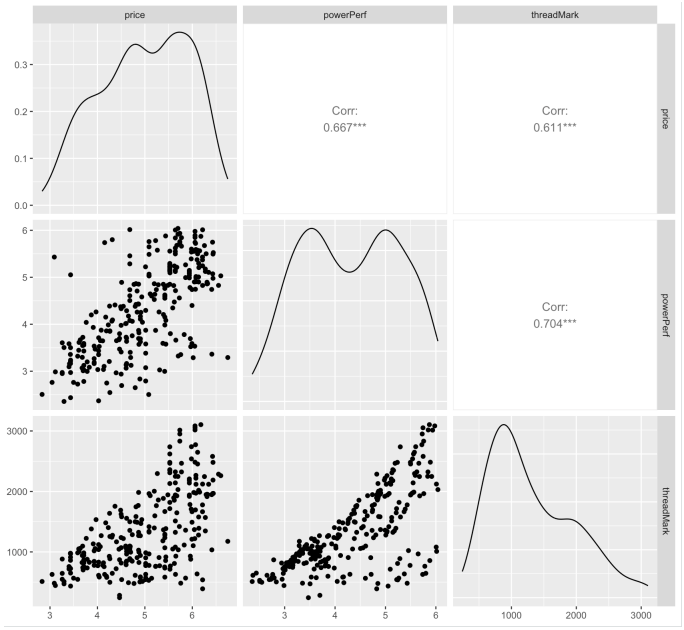

Figure 29: Relationship between the interested variables

As we can see, there is a approximate linear relationship between **price** and **log(powerPerf+1)**, **threadMark**. As power efficiency and thread mark increases, price of laptop also seems to increase proportionally according to that.
Now we build the model above using the *lm()* function, choose the data frame train, independent variable *log(price+1)* and dependent variable *threadMark* and *log(powerPerf+1)*:

```
linearModule <-lm(price~TDP+powerPerf,data=train)
summary(linearModule)
```

```
Call:
lm(formula = price ~ powerPerf + threadMark, data = train)

Residuals:
      Min       1Q   Median       3Q      Max
-2.06530 -0.46519 -0.08943  0.42688  2.27705

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.525e+00  1.954e-01  12.922  < 2e-16 ***
powerPerf   4.493e-01  5.888e-02   7.630 4.03e-13 ***
threadMark  3.970e-04  8.705e-05   4.561 7.74e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6651 on 270 degrees of freedom
Multiple R-squared:  0.4846,    Adjusted R-squared:  0.4808
F-statistic: 126.9 on 2 and 270 DF,  p-value: < 2.2e-16

>
```

Figure 30: Linear regression summary

2.6.5 Estimating parameters

Since we are working with the regression models, our output should also be a line that most fit the points, our model equation can be expressed as follow:

$$log(price + 1) = 2.525 + 0.4493 * log(powerPerf + 1) + 0.000397 * threadMark \tag{3}$$

Calculate sum of squares

```
       SSE <- (linearModule$residuals ^ 2) %>% sum()
SSR <- ((linearModule$fitted.values - mean(train$price)) ^ 2) %>% sum()
SST <- ((train$price - mean(train$price)) ^ 2) %>% sum()
ss <- as.data.frame(c(SSE, SSR, SST), row.names = c("SSE", "SSR", "SST"))
names(ss) <- c("Value")
ss
```

```
> SSE <- (linearModule$residuals ^ 2) %>% sum()
> SSR <- ((linearModule$fitted.values - mean(train$price)) ^ 2) %>% sum()
> SST <- ((train$price - mean(train$price)) ^ 2) %>% sum()
> ss <- as.data.frame(c(SSE, SSR, SST), row.names = c("SSE", "SSR", "SST"))
> names(ss) <- c("Value")
> ss
       Value
SSE 119.4341
SSR 112.3078
SST 231.7419
```

Figure 31: SSE, SSR and SST value

Calculate adjusted R squared

$$R^2_{adj} = 1 - \frac{SSE/(n-p)}{SST/(n-1)} = 1 - \frac{119.4341/(273-3)}{231.7419/(273-1)} = 0.4808$$

with n being the number of total observations, p is number of regression parameters.
Adjusted R squared avoid over-fitting problem in R squared value. Basically R^2 will always increase whenever a new variable is added into our model, but R^2_{adj} adds precision and **reliability** by considering the impact of additional independent variables that tend to skew the results of R-squared measurements. It can also be seen that this R^2_{adj} value is equal to that in the summary of linear regression model (Figure 38).

2.6.6 Test hypothesis for regression model

Testing existence of linear regression model - significant level: 0.01%

- Hypothesis H_0 : $\beta_i = 0$ $(\forall i = 0, 1)$ (the linear regression model representing the dependent variable and all the predictor variables does not exist).
- Hypothesis H_1 : $\beta_i \neq 0$ $(\exists i = 0, 1)$ (the linear regression model exists).

The probability given by F-statistic is extremely below significant level $\alpha = 0.01$, thus we can reject the null hypothesis H_0. In conclusion, there **exists** a linear regression model that represent the dependent variable *log(price+1)* and the independent variables *log(powerPerf+1)* and *threadMark*.

Testing regression coefficient - significant level: 0.05%
Now we conduct a test on each of the regression coefficient β_0, β_1 at signiificant level: 0.05.

- Hypothesis H_0 : $\beta_i = 0$ (there is no relationship between response and the predictor variables).
- Hypothesis H_1 : $\beta_i \neq 0$ (there exists a relationship between variable i and dependent variable).

The probability of t-value $Pr(> |t|)$ for all the predictors are significantly less than $\alpha = 0.05$. In conclusion, the coefficient of all predictors are **meaningful** to our model.

2.6.7 R^2 and adjusted R^2

The R^2 is 48.46% and the R^2_{adj} is 48.08%, both explain the variability of dependent variable responsive to the independent variables. It implies in 100% the variability of log(price+1), 48.08% is caused by these factors *log(powerPerf+1)*, and *threadMark*

2.6.8 Multicorrelation

Next we will check multicorrelation condition using the vif() function:

```
library(car)
vif(linearModule)
```

```
> vif(linearModule)
 powerPerf threadMark
  1.983066   1.983066
>
```

Figure 32: VIF test

As you can see, all of the values are significantly less than 10 so we do not have to worry about multi-correlation in this model.

2.6.9 Check if assumptions of the model are fulfilled

The next step is to check if the model's assumptions are completely fulfilled so that we can determine if the model is appropriate to the task as well as the credibility of the model.
Recall the assumptions of the regression model: $Y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + + \beta_i X_i + \epsilon_i$ (i = 1,2,...,n) are:

- Linearity of data: the relationship between predictor variable X (independent variables) and dependent variable Y is assumed to be linear.
- The errors terms are assumed to be:

- Normally distributed.
- The variance is constant. $\epsilon_i \sim \mathrm{N}(0,\sigma^2)$
- The errors ϵ_1, ϵ_2, . . . , ϵ_n are independent of each other.

Linearity of the data

The linearity assumption can be checked by inspecting the Residuals vs Fitted plot. Ideally, the residual plot will show no fitted pattern. That is, the red line should be approximately horizontal at zero. The presence of a pattern may indicate a problem with some aspect of the linear model.

```
par(mfrow=c(1,1))
plot(linearModule, col = "steel blue", which = 1)
```

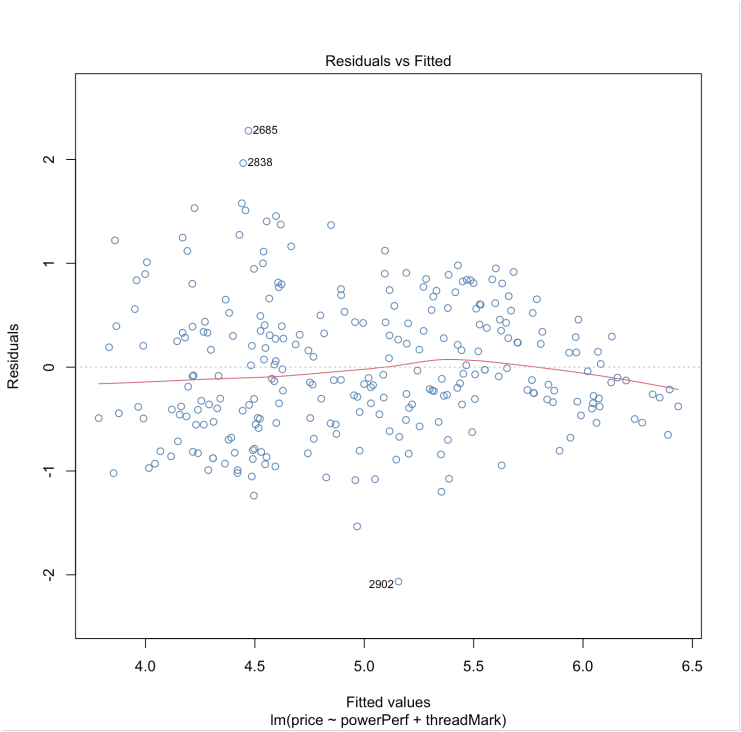

Figure 33: Residuals vs Fitted

In our example, there is no pattern in the residual plot. This suggests that we can assume linear relationship between the predictors and the outcome variables.

Normality of residuals

The QQ plot of residuals can be used to visually check the normality assumption. The normal probability plot of residuals should approximately follow a straight line.

```
plot(linearModule, col = "steel blue", which = 2)
```

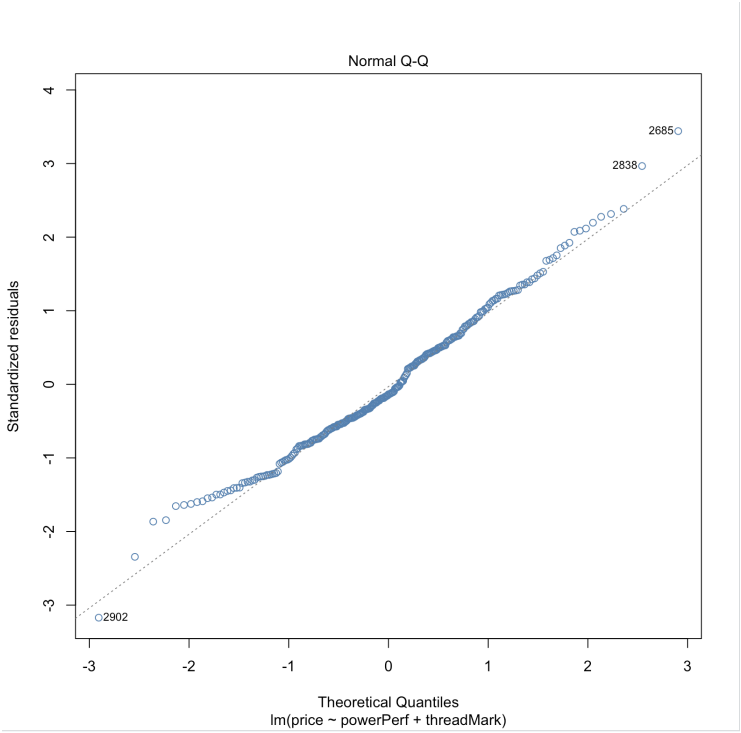

Figure 34: Normal Q-Q plot

In our example, all the points fall approximately along this reference line, so we can assume normality.

Homogeneity of variance

This assumption can be checked by examining the scale-location plot, also known as the spread-location plot. This plot shows if residuals are spread equally along the ranges of predictors. It's good if you see a horizontal line with equally spread points.

```
plot(linearModule, col = "steel blue", which = 3)
```

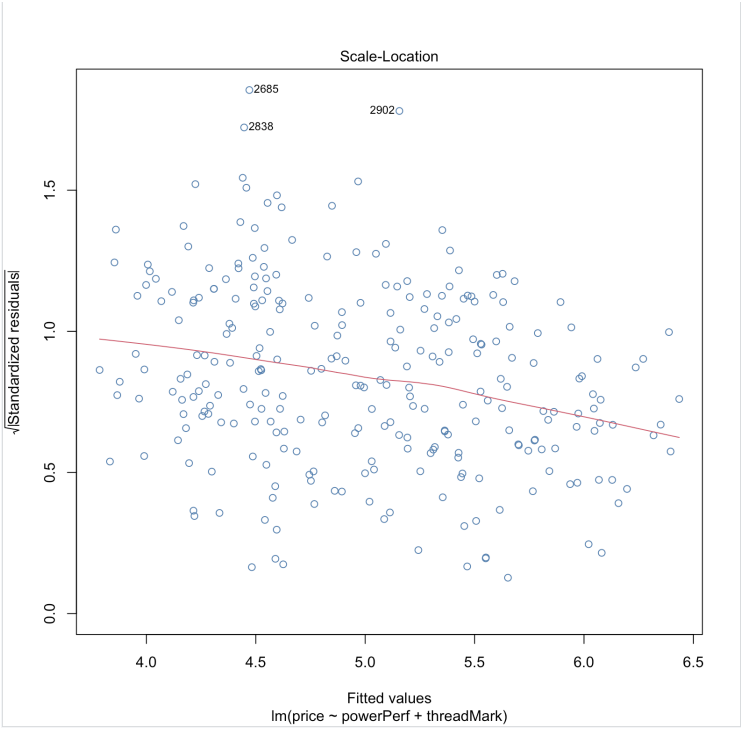

Figure 35: Scale-Location

It can be seen that the variability (variances) of the residual points decreases with the value of the fitted outcome variable, suggesting non-constant variances in the residuals errors (or heteroscedasticity). However, the decrease we are observing is relatively small, so we can accept this slight heteroscedasticity problem.

Outliers and high leverage points

Outliers and high leverage points can be identified by inspecting the Residuals vs Leverage plot:

```
plot(linearModule, col = "steel blue", which = 5)
```

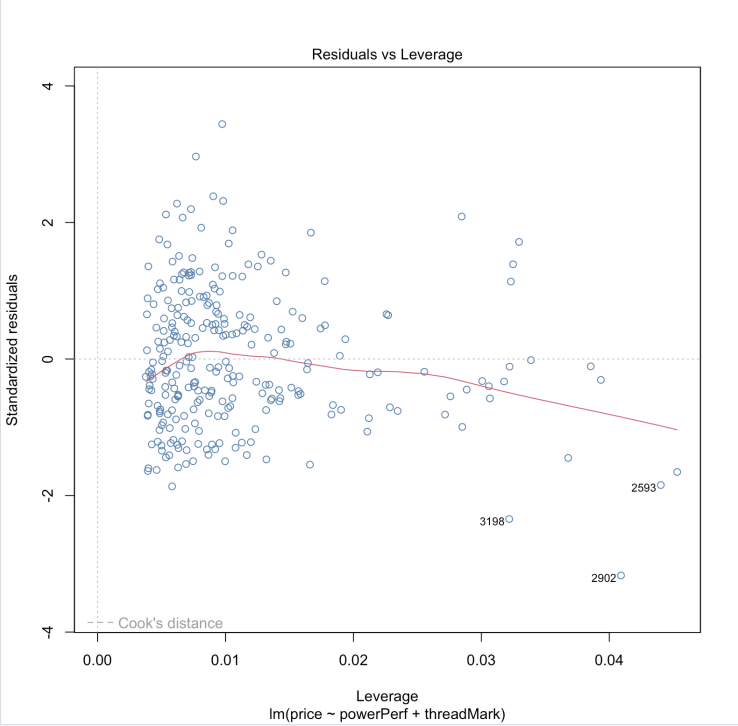

Figure 36: Residuals vs Leverage

The plot above highlights the top 3 most extreme points (#2593, #3198 and #2902), with a standardized residuals below -2. However, there is no outliers that exceed 2 standard deviations, what is good.
Additionally, there is no high leverage point in the data. That is, all data points, have a leverage statistic below $2(p+1)/n = 6/273 \approx 0.02$.

2.6.10 Prediction

In this section, we will use the **predict()** function to test the data set test we have splitted before.

```
predicts <- predict(linearModule, newdata = test)
actuals <- test$price
evaluate <- data.frame(actuals ,predicts)
summary(evaluate)
```

```
    actuals         predicts
 Min.   :2.995   Min.   :3.775
 1st Qu.:3.995   1st Qu.:4.419
 Median :4.905   Median :4.684
 Mean   :4.882   Mean   :4.885
 3rd Qu.:5.833   3rd Qu.:5.405
 Max.   :6.370   Max.   :6.096
```

Figure 37: Prediction of general characteristic of test data

As you can see, our model predict approximately close to the description of original data. Next, we use the code below to calculate the sum of square error between the actual and fitted value:

```
SSE <- (m$residuals ^ 2) %>% sum()
SSE
```

```
> m <- lm(actuals~predicts,data=evaluate)
> SSE <- (m$residuals ^ 2) %>% sum()
> SSE
[1] 14.13521
>
```

Figure 38: SSE value

Use that value to calculate mean squared error, we have:

$$MSE = \frac{SSE}{n} = 0.34$$

The smaller and closer to 0 our MSE value is, the better our model is at predicting the actual value. In our case, a MSE value of 0.34 indicates that our model is **relatively good** at predicting the actual price.
Finally let's try to construct a linear regression model using **actuals** as dependent variable and **predicts** as predictors to see how much variation in our actual values can the predicted values explain:

```
m <- lm(actuals~predicts,data=evaluate)
summary(m)
```

```
Call:
lm(formula = actuals ~ predicts, data = evaluate)

Residuals:
    Min      1Q  Median      3Q     Max
-0.9354 -0.4167 -0.0029  0.3484  1.5463

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.6808     0.7028  -2.391   0.0216 *
predicts      1.3434     0.1426   9.418 1.06e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5945 on 40 degrees of freedom
Multiple R-squared:  0.6892,    Adjusted R-squared:  0.6814
F-statistic:  88.7 on 1 and 40 DF,  p-value: 1.056e-11
```

Figure 39: Model accuracy

Conclusion

Based on the given result of function summary the model m have an R^2 value of 68.92% , this indicates that almost 69% of the actual price variation can be explained by the predictions of linear Module. It can be seen that the model we are having is **statistically significant**.