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

# PROBABILITY AND STATISTICS (MT2013) - SEMESTER 231

**Project Topic**

# *Computer Parts (CPUs and GPUs)*

Lecturer: Nguyen Tien Dung

Class: CC02 Group: 6

Students: Nong Thuc Khue - 2252385
Vu Quynh Huong - 2252286
Le Phong Hao - 2252182
Huynh Van Anh Hoang - 2252229
Truong Minh Khang - 2252314

HO CHI MINH CITY, DECEMBER 2023

# Contribution

| No. | Name | Student ID | Task | Completion |
|---|---|---|---|---|
| 1 | Nong Thuc Khue | 2252385 | Code writer | 100% |
| 2 | Vu Quynh Huong | 2252286 | Code writer | 100% |
| 3 | Le Phong Hao | 2252182 | Report management | 100% |
| 4 | Huynh Van Anh Hoang | 2252229 | Report management | 100% |
| 5 | Truong Minh Khang | 2252314 | Report management | 100% |

# List of Figures

# Contents

# 1 Theoretical foundation

## 1.1 Linear regression

### 1.1.1 Two types of linear regression

**Single Linear Regression:** If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

The expected value of $Y$ at each level of $x$ is a random variable $\mathbb{E}(Y|x) = \alpha + \beta x$. We assume that each observation, $Y$, can be described by the model $Y = \alpha + \beta x + \epsilon$. That is, $Y_i = \alpha + \beta x_i + \epsilon_i, i = 1, 2, 3, ..., n$.

To achieve the best−fit regression line, it is essential to make predictions the value that the difference between the predicted value and the true value $Y$ is the *minimum*. So, we have to update the value of $\alpha$ and $\beta$, to reach the best values of both ones.

**Multiple Linear Regression:** If more than one independent variable is used to predict the value of a numerically dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

### 1.1.2 Linear regression model

Linear regression is a powerful tool for understanding and predicting the behavior of a variable, however, it needs to meet a few conditions in order to be accurate and dependable solutions.

- **Linearity:** The independent and dependent variables have a linear relationship with each other. This implies that changes in the dependent variable follow those in the independent variable(s) in a linear fashion.
- **Independence:** The observations in the dataset are independent of each other. This means that the value of the dependent variable for one observation does not depend on the value of the dependent variable for another observation.
- **Homoscedasticity:** Across all levels of the independent variable(s), the variance of the errors is constant. This indicates that the amount of independent variable(s) does not have an impact on the variance of the errors.
- **Normality:** The errors in the model are normally distributed.
- **No multi-collinearity:** There is no high correlation between the independent variables. This indicates that there is little or no correlation between the independent variables.

### 1.1.3 Cost function

The cost function or the loss function is nothing more than the error or difference between the predicted $\widehat{Y}$ and the true value $Y$. It is the Mean Squared Error (MSE) between the predicted value and the true value. The joint function ($J$) can be written as:

$$\text{Cost function}(J) = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)^2$$

**Residuals:** The distance between the actual value and predicted values is called *residual*. If the observed points are far from the regression line, then the residual will be high, and so the cost function will be high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

#### 1.1.4 Gradient descent

Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.

A regression model uses gradient descent to update the coefficients of the line by reducing the cost function. It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

#### 1.1.5 Model performance

The goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of the various models is called optimization. It can be achieved by R−squared method:

- R−squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of $0 - 100\%$.
- The high value of R−square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a coefficient of determination, or coefficient of multiple determination for multiple regression.
- It can be calculated from the below formula:

$$R\!-\!squared = \frac{Explained\ variation}{Total\ variation}$$

### 1.2 Stepwise regression

#### 1.2.1 Definition of stepwise regression

Stepwise regression is a method of fitting a regression model by iteratively adding or removing variables. It is used to build a model that is accurate and parsimonious, which means that it has the smallest number of variables that can explain the data. There are two main types of stepwise regression:

- **Forward Selection:** the algorithm starts with an empty model and iteratively adds variables to the model until no further improvement is made.
- **Backward Elimination:** the algorithm starts with a model that includes all variables and iteratively removes variables until no further improvement is made.

The advantage of stepwise regression is that it can automatically select the most important variables for the model and build a parsimonious model. The disadvantage is that it may not always select the best model and can be sensitive to the order in which the variables are added or removed.

#### 1.2.2 Stepwise regression and Linear regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. In other words, it is a method for predicting a response (or dependent variable) based on one or more predictor variables.

Stepwise regression is a method for building a regression model by adding or removing predictors in a step-by-step fashion. The goal of stepwise regression is to identify the subset of predictors that provides the best predictive performance for the response variable. This is done by starting with an empty model and iteratively adding or removing predictors based on the strength of their relationship with the response variable.

In summary, linear regression is a method for modeling the relationship between a response and one or more predictor variables, while stepwise regression is a method for building a regression model by iteratively adding or removing predictors.

## 1.3 Analysis of variance (ANOVA)

### 1.3.1 Definition of ANOVA

Analysis of variance (ANOVA) is an analysis tool used in statistics that divides an observed aggregate variability found inside a data set into two parts: systematic factors and random factors. Systematic factors have a statistical influence on the given data set, while random factors do not.

Analysts use the ANOVA test to determine the influence that independent variables have on the dependent variable in a regression study.

The t− and z−test methods developed in the 20th century were used for statistical analysis until 1918, when Ronald Fisher created the analysis of variance method.

ANOVA is also called Fisher analysis of variance and is the extension of the t− and z−tests.

What are t−test and z−test?

Both serve as hypothesis tests that evaluate whether there is a notable difference between the means of two distinct groups or populations.

The t−test is used when the population variance is unknown, or the sample size is small ($n < 30$). At the same time, the z−test is applied when the population variance is known, and the sample size is large ($n > 30$).

T−test employs the Student's t−distribution, while z−test uses the standard normal distribution. As the sample size increases, t−distribution will converge into the standard normal distribution.

The t−test can be understood as a statistical test which is used to compare and analyze whether the means of the two population is different from one another or not when the standard deviation is not known. As against, z−test is a parametric test, which is applied when the standard deviation is known, to determine whether the means of the two datasets differ from each other.

### 1.3.2 What does the Analysis of Variance (ANOVA) reveal?

The ANOVA test is the initial step in analyzing the factors that affect a given data set. Once the test is finished, an analyst performs additional testing on the methodical factors that measurably contribute to the data set's inconsistency. The analyst utilizes the results of the ANOVA test in an f−test to generate additional data that align with the proposed regression models.

The ANOVA test allows a comparison of more than two groups at the same time to determine whether a relationship exists between them. The result of the ANOVA formula, the F statistic (also called the F−ratio), allows for the analysis of multiple groups of data to determine the variability between samples and within samples.

$$F = \frac{MSTr}{MSE} = \frac{\dfrac{SSTr}{I-1}}{\dfrac{SSE}{N-1}}$$

Where,

- $MSTr = \dfrac{SST}{I-1}$ : mean square for treatment.

- $MSE = \dfrac{SSE}{N-1}$ : mean square for error.

If there is no real difference between the tested groups, which is called the null hypothesis, the result of the ANOVA's F−ratio statistic will be close to 1. The distribution of all possible values of the F statistic is the F−distribution. This is actually a group of distribution functions, with two characteristic numbers called the numerator degrees of freedom and the denominator degrees of freedom.

### 1.3.3 One-way ANOVA versus Two-way ANOVA

There are two main types of ANOVA: one-way (or unidirectional) and two-way. There also variations of ANOVA. For example, MANOVA (multivariate ANOVA) differs from ANOVA as the former tests for multiple dependent variables simultaneously while the latter assesses only one dependent variable at a time.

One-way or two-way refers to the number of independent variables in your analysis of variance test. 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.

One-way analysis of variance (ANOVA) is used to determine whether there are statistically significant differences between the means of three or more independent (unrelated) groups.

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. It is utilized to observe the interaction between the two factors and tests the effect of two factors at the same time.

Furthermore, ANOVA relies on assumptions. ANOVA tests assume that the data are normally distributed and that the levels of variance in each group are roughly equal. Finally, we assume that all observations are made independently. If these assumptions are not accurate, ANOVA may not be useful for comparing groups.

## 1.4 Support vector machine

### 1.4.1 Definition of support vector machine

Support Vector Machine (SVM) is a powerful machine learning algorithm used for linear or nonlinear classification, regression, and even outlier detection tasks. SVMs can be used for a variety of tasks, such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection, and anomaly detection. SVMs are adaptable and efficient in a variety of applications because they can manage high-dimensional data and nonlinear relationships.

SVM algorithms are very effective as we try to find the maximum separating hyperplane between the different classes available in the target feature.

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Although we also say regression problems, it is best suited for classification. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries to make the margin between the closest points of different classes as maximum as possible. The dimension of the hyperplane depends on the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

### 1.4.2 Types of support vector machine

Based on the nature of the decision boundary, Support Vector Machines (SVM) can be divided into two main parts:

- **Linear SVM:** Linear SVMs use a linear decision boundary to separate the data points of different classes. When the data can be precisely linearly separated, linear SVMs are very suitable. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their respective classes. A hyperplane that maximizes the margin between the classes is the decision boundary.

- **Non-Linear SVM:** Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line (in the case of 2D). By using kernel functions, nonlinear SVMs can handle nonlinearly separable data. The original input data is transformed by these kernel functions into a higher-dimensional feature space, where the data points can be linearly separated. A linear SVM is used to locate a non-linear decision boundary in this modified space.

# 2 Data processing

## 2.1 Data importing

### 2.1.1 Import file

```r
#read data
CPU_data<-read.csv("~/Desktop/Assignment/work/Intel_CPUs.csv")
head(CPU_data, 3)
```

```
                   Product_Collection Vertical_Segment Processor_Number
1 7th Generation Intel Core i7 Processors           Mobile          i7-7Y75
2 8th Generation Intel Core i5 Processors           Mobile         i5-8250U
3 8th Generation Intel Core i7 Processors           Mobile         i7-8550U
    Status Launch_Date Lithography Recommended_Customer_Price nb_of_Cores
1 Launched      Q3'16       14 nm                    $393.00           2
2 Launched      Q3'17       14 nm                    $297.00           4
3 Launched      Q3'17       14 nm                    $409.00           4
  nb_of_Threads Processor_Base_Frequency Max_Turbo_Frequency        Cache
1             4                 1.30 GHz            3.60 GHz 4 MB SmartCache
2             8                 1.60 GHz            3.40 GHz 6 MB SmartCache
3             8                 1.80 GHz            4.00 GHz 8 MB SmartCache
   Bus_Speed   TDP Embedded_Options_Available Conflict_Free Max_Memory_Size
1 4 GT/s OPI 4.5 W                        No           Yes           16 GB
2 4 GT/s OPI  15 W                        No           Yes           32 GB
3 4 GT/s OPI  15 W                        No           Yes           32 GB
             Memory_Types Max_nb_of_Memory_Channels Max_Memory_Bandwidth
1 LPDDR3-1866, DDR3L-1600                         2            29.8 GB/s
2  DDR4-2400, LPDDR3-2133                         2            34.1 GB/s
3  DDR4-2400, LPDDR3-2133                         2            34.1 GB/s
  ECC_Memory_Supported Processor_Graphics_ Graphics_Base_Frequency
1                   No                  NA                 300 MHz
2                   No                  NA                 300 MHz
3                   No                  NA                 300 MHz
  Graphics_Max_Dynamic_Frequency Graphics_Video_Max_Memory Graphics_Output
1                       1.05 GHz                     16 GB eDP/DP/HDMI/DVI
2                       1.10 GHz                     32 GB eDP/DP/HDMI/DVI
3                       1.15 GHz                     32 GB eDP/DP/HDMI/DVI
  Support_4k Max_Resolution_HDMI Max_Resolution_DP
1         NA      4096x2304@24Hz    3840x2160@60Hz
2         NA      4096x2304@24Hz    4096x2304@60Hz
3         NA      4096x2304@24Hz    4096x2304@60Hz
  Max_Resolution_eDP_Integrated_Flat_Panel DirectX_Support OpenGL_Support
1                           3840x2160@60Hz              12             NA
2                           4096x2304@60Hz              12             NA
3                           4096x2304@60Hz              12             NA
  PCI_Express_Revision PCI_Express_Configurations_ Max_nb_of_PCI_Express_Lanes
1                    3  1x4, 2x2, 1x2+2x1 and 4x1                          10
2                    3  1x4, 2x2, 1x2+2x1 and 4x1                          12
3                    3  1x4, 2x2, 1x2+2x1 and 4x1                          12
      T Intel_Hyper_Threading_Technology_ Intel_Virtualization_Technology_VTx_
1 100 C                               Yes                                  Yes
2 100 C                               Yes                                  Yes
3 100 C                               Yes                                  Yes
  Intel_64_ Instruction_Set Instruction_Set_Extensions Idle_States
1       Yes          64-bit          SSE4.1/4.2, AVX 2.0         Yes
2       Yes          64-bit          SSE4.1/4.2, AVX 2.0         Yes
3       Yes          64-bit          SSE4.1/4.2, AVX 2.0         Yes
```

```
  Thermal_Monitoring_Technologies Secure_Key Execute_Disable_Bit
1                             Yes        Yes                 Yes
2                             Yes        Yes                 Yes
3                             Yes        Yes                 Yes
```

This command to import a data file of .csv form from the path into the program.

### 2.1.2 Import libraries

```r
# Import libraries
library(dplyr)
library(stringr)
library(GGally)
library(corrplot)
library(caTools)
library(MASS)
library(car)
library(e1071)
library(nortest)
```

### 2.1.3 Explanation of the dataset

With the technological revolution, computers vary in purpose, leading to variation in their component, including variation of CPUs in statistical information. The dataset contains CPUs' vital information. All of them have a relationship with each other.

## 2.2 Data cleaning

### 2.2.1 Removing unused data

Checking how many missing values in each variable.

```r
# Checking missing values
apply(is.na(CPU_data), 2, sum)
```

```
           Product_Collection              Vertical_Segment
                            0                             0
             Processor_Number                        Status
                            0                             0
                  Launch_Date                   Lithography
                            0                             0
  Recommended_Customer_Price                   nb_of_Cores
                            0                             0
                nb_of_Threads      Processor_Base_Frequency
                          856                             0
           Max_Turbo_Frequency                         Cache
                            0                             0
                    Bus_Speed                           TDP
                            0                             0
   Embedded_Options_Available                 Conflict_Free
                            0                             0
              Max_Memory_Size                  Memory_Types
                            0                             0
    Max_nb_of_Memory_Channels          Max_Memory_Bandwidth
```

```
                              869                                   0
               ECC_Memory_Supported              Processor_Graphics_
                                0                                2283
          Graphics_Base_Frequency     Graphics_Max_Dynamic_Frequency
                                0                                   0
        Graphics_Video_Max_Memory                    Graphics_Output
                                0                                   0
                       Support_4k                 Max_Resolution_HDMI
                             2283                                   0
                Max_Resolution_DP Max_Resolution_eDP_Integrated_Flat_Panel
                                0                                   0
                  DirectX_Support                      OpenGL_Support
                                0                                2283
              PCI_Express_Revision         PCI_Express_Configurations_
                                0                                   0
         Max_nb_of_PCI_Express_Lanes                                  T
                             1104                                   0
     Intel_Hyper_Threading_Technology_    Intel_Virtualization_Technology_VTx_
                                0                                   0
                         Intel_64_                     Instruction_Set
                                0                                   0
          Instruction_Set_Extensions                         Idle_States
                                0                                   0
       Thermal_Monitoring_Technologies                         Secure_Key
                                0                                   0
                 Execute_Disable_Bit
                                0
```

Values of **Processor_Graphics_**, **Support_4k**, and **OpenGL_Support** are missing in all rows; therefore, we remove columns of these variables.

Moreover, we will not use **Product_Collection, Vertical_Segment, Processor_Number, Status, Launch_Date, Instruction_Set_Extensions, PCI_Express_Configurations_, Max_Resolution_eDP_Integrated_Flat_Panel, Max_Resolution_DP, Graphics_Output, Max_Resolution_HDMI, Memory_Types, Bus_Speed, Cache, DirectX_Support, PCI_Express_Revision, Conflict_Free** to analyze, therefore, we also remove these variables.

```r
# Remove unused variables
delete = c("OpenGL_Support", "Support_4k", "Processor_Graphics_", "Product_Collection", "Vertical_Segment", "Processor_Number", "Status", "Launch_Date", "Instruction_Set_Extensions","PCI_Express_Configurations_","Max_Resolution_eDP_Integrated_Flat_Panel","Max_Resolution_DP","Graphics_Output","Max_Resolution_HDMI","Memory_Types","Bus_Speed","Cache","DirectX_Support", "PCI_Express_Revision", "Conflict_Free")

CPU_data <- CPU_data[, !names(CPU_data) %in% delete, drop = FALSE]
```

### 2.2.2 Filling missing values

We convert space and escape sequence to $NA$ value.

```r
# Convert " " and "\n- " to NA
CPU_data[(CPU_data == "") | (CPU_data == "\n- ")] <- NA
```

We remove unit of some numerical variables and transform them to numeric form.

```
# Remove unit of numerical variables and transform to numeric form
CPU_data$Lithography <- as.numeric(sub("nm", "", CPU_data$Lithography))
CPU_data$Max_Turbo_Frequency <- as.numeric(sub("GHz", "", CPU_data$Max_Turbo_Frequency))
CPU_data$TDP <- as.numeric(sub("W", "", CPU_data$TDP))
CPU_data$Max_Memory_Size <- as.numeric(sub("GB", "", CPU_data$Max_Memory_Size))
CPU_data$Max_Memory_Bandwidth <- as.numeric(sub("GB/s", "", CPU_data$Max_Memory_Bandwidth))
CPU_data$Graphics_Base_Frequency <- as.numeric(sub("MHz", "", CPU_data$Graphics_Base_Frequency))
CPU_data$Graphics_Video_Max_Memory <- as.numeric(sub("GB", "", CPU_data$Graphics_Video_Max_Memory))
CPU_data$T <- as.numeric(sub("°C", "", CPU_data$T))
```

There are some numerical variables that need to be converted unit before removing units and transform to numeric form.

```
# Convert units for Graphics_Max_Dynamic_Frequency
subset_GHz <- CPU_data[grepl("GHz", CPU_data$Graphics_Max_Dynamic_Frequency, ignore.case = TRUE) , ]
CPU_data <- CPU_data[!grepl("GHz", CPU_data$Graphics_Max_Dynamic_Frequency, ignore.case = TRUE), ]

subset_GHz$Graphics_Max_Dynamic_Frequency <- gsub("GHz", "",subset_GHz$Graphics_Max_Dynamic_Frequency,fixed = TRUE)
subset_GHz$Graphics_Max_Dynamic_Frequency <- as.numeric(subset_GHz$Graphics_Max_Dynamic_Frequency)
subset_GHz$Graphics_Max_Dynamic_Frequency <- subset_GHz$Graphics_Max_Dynamic_Frequency*(1000)

CPU_data$Graphics_Max_Dynamic_Frequency <- gsub("MHz", "",CPU_data$Graphics_Max_Dynamic_Frequency,fixed = TRUE)
CPU_data$Graphics_Max_Dynamic_Frequency <- as.numeric(CPU_data$Graphics_Max_Dynamic_Frequency)

CPU_data <- bind_rows(CPU_data,subset_GHz)

# Convert units for Processor_Base_Frequency
subset_GHz <- CPU_data[grepl("GHz", CPU_data$Processor_Base_Frequency, ignore.case = TRUE) , ]
CPU_data <- CPU_data[!grepl("GHz", CPU_data$Processor_Base_Frequency, ignore.case = TRUE), ]

subset_GHz$Processor_Base_Frequency <- gsub("GHz", "",subset_GHz$Processor_Base_Frequency,fixed = TRUE)

subset_GHz$Processor_Base_Frequency <- as.numeric(subset_GHz$Processor_
```

```
    Base_Frequency)
subset_GHz$Processor_Base_Frequency <- subset_GHz$Processor_Base_
    Frequency*(1000)

CPU_data$Processor_Base_Frequency <- gsub("MHz", "",CPU_data$Processor_
    Base_Frequency,fixed = TRUE)
CPU_data$Processor_Base_Frequency <- as.numeric(CPU_data$Processor_Base
    _Frequency)

CPU_data <- bind_rows(CPU_data,subset_GHz)
```

We fill missing values of numerical variables by their *mean* values.

```
# Filling missing values for numerical variables
CPU_data$Lithography[is.na(CPU_data$Lithography)] = mean(CPU_data$
    Lithography, na.rm=T)
CPU_data$nb_of_Cores[is.na(CPU_data$nb_of_Cores)] = mean(CPU_data$nb_of
    _Cores, na.rm=T)
CPU_data$nb_of_Threads[is.na(CPU_data$nb_of_Threads)] = mean(CPU_data$
    nb_of_Threads, na.rm=T)
CPU_data$Max_Turbo_Frequency[is.na(CPU_data$Max_Turbo_Frequency)] =
    mean(CPU_data$Max_Turbo_Frequency, na.rm=T)
CPU_data$TDP[is.na(CPU_data$TDP)] = mean(CPU_data$TDP, na.rm=T)
CPU_data$Max_Memory_Size[is.na(CPU_data$Max_Memory_Size)] = mean(CPU_
    data$Max_Memory_Size, na.rm=T)
CPU_data$Max_nb_of_Memory_Channels[is.na(CPU_data$Max_nb_of_Memory_
    Channels)] = mean(CPU_data$Max_nb_of_Memory_Channels, na.rm=T)
CPU_data$Max_Memory_Bandwidth[is.na(CPU_data$Max_Memory_Bandwidth)] =
    mean(CPU_data$Max_Memory_Bandwidth, na.rm=T)
CPU_data$Graphics_Base_Frequency[is.na(CPU_data$Graphics_Base_Frequency
    )] = mean(CPU_data$Graphics_Base_Frequency, na.rm=T)
CPU_data$Graphics_Video_Max_Memory[is.na(CPU_data$Graphics_Video_Max_
    Memory)] = mean(CPU_data$Graphics_Video_Max_Memory, na.rm=T)
CPU_data$Max_nb_of_PCI_Express_Lanes[is.na(CPU_data$Max_nb_of_PCI_
    Express_Lanes)] = mean(CPU_data$Max_nb_of_PCI_Express_Lanes, na.rm=T
    )
CPU_data$T[is.na(CPU_data$T)] = mean(CPU_data$T, na.rm=T)
CPU_data$Graphics_Max_Dynamic_Frequency[is.na(CPU_data$Graphics_Max_
    Dynamic_Frequency)] = mean(CPU_data$Graphics_Max_Dynamic_Frequency,
    na.rm=T)
CPU_data$Processor_Base_Frequency[is.na(CPU_data$Processor_Base_
    Frequency)] = mean(CPU_data$Processor_Base_Frequency, na.rm=T)

CPU_data$Recommended_Customer_Price <- gsub("$", "", CPU_data$
    Recommended_Customer_Price, fixed = TRUE)
CPU_data$Recommended_Customer_Price <- as.numeric(CPU_data$Recommended_
    Customer_Price)
```

We fill missing values of categorical variables by their *mode*.

```r
# Filling missing values for categorical variables
fillmode <- function(column) {
  mode_value <- names(sort(table(column), decreasing = TRUE))[1]
  column[is.na(column)] <- mode_value
  return(column)
}

fill_col <- c("Embedded_Options_Available","ECC_Memory_Supported","Intel_Hyper_Threading_Technology_","Intel_Virtualization_Technology_VTx_","Intel_64_","Instruction_Set","Idle_States","Thermal_Monitoring_Technologies","Secure_Key","Execute_Disable_Bit")

CPU_data[fill_col] <- lapply(CPU_data[fill_col], fillmode)
```

We transform categorical variables to numeric form by using categorical signs “1” and“0”. This step is conducted for training model.

```r
# Transform categorical variables to numerical variables
CPU_data$Embedded_Options_Available <- ifelse(CPU_data$Embedded_Options_Available == "Yes", 1, 0)
CPU_data$ECC_Memory_Supported <- ifelse(CPU_data$ECC_Memory_Supported == "Yes", 1, 0)
CPU_data$Intel_Hyper_Threading_Technology_ <- ifelse(CPU_data$Intel_Hyper_Threading_Technology_ == "Yes", 1, 0)
CPU_data$Intel_Virtualization_Technology_VTx_ <- ifelse(CPU_data$Intel_Virtualization_Technology_VTx_ == "Yes", 1, 0)
CPU_data$Intel_64_ <- ifelse(CPU_data$Intel_64_ == "Yes", 1, 0)
CPU_data$Idle_States <- ifelse(CPU_data$Idle_States == "Yes", 1, 0)
CPU_data$Thermal_Monitoring_Technologies <- ifelse(CPU_data$Thermal_Monitoring_Technologies == "Yes", 1, 0)
CPU_data$Secure_Key <- ifelse(CPU_data$Secure_Key == "Yes", 1, 0)
CPU_data$Execute_Disable_Bit <- ifelse(CPU_data$Execute_Disable_Bit == "Yes", 1, 0)

CPU_data$Instruction_Set <- gsub("32-bit", "0",CPU_data$Instruction_Set,fixed = TRUE)
CPU_data$Instruction_Set <- gsub("64-bit", "1",CPU_data$Instruction_Set,fixed = TRUE)
CPU_data$Instruction_Set <- gsub("Itanium 1", "2",CPU_data$Instruction_Set,fixed = TRUE)
CPU_data$Instruction_Set <- as.numeric(CPU_data$Instruction_Set)
```

After all, we check the missing values again.

```r
# Checking missing values
apply(is.na(CPU_data), 2, sum)
```

```
                 Lithography          Recommended_Customer_Price
                           0                                1262
                 nb_of_Cores            Processor_Base_Frequency
```

```
                              0                                     0
              Max_Turbo_Frequency                                  TDP
                              0                                     0
       Embedded_Options_Available                      Max_Memory_Size
                              0                                     0
             Max_Memory_Bandwidth                 ECC_Memory_Supported
                              0                                     0
          Graphics_Base_Frequency       Graphics_Max_Dynamic_Frequency
                              0                                     0
        Graphics_Video_Max_Memory                                    T
                              0                                     0
 Intel_Hyper_Threading_Technology_ Intel_Virtualization_Technology_VTx_
                              0                                     0
                        Intel_64_                      Instruction_Set
                              0                                     0
                      Idle_States       Thermal_Monitoring_Technologies
                              0                                     0
                       Secure_Key                   Execute_Disable_Bit
                              0                                     0
```

Since our objective of this report is to analyze and predict the recommended customer price of CPU, **Recommended_Customer_Price** is the only variable that has missing values after the pre-processing step.

## 2.3 Data visualization

### 2.3.1 Descriptive statistics for each variable

We divide CPU_data into 2 separate data CPU_learn and CPU_train, which CPU_train is the set that does not include missing values, even missing values in the Recommended_Customer_Price variable.

```r
# Divide CPU_data into CPU_learn and CPU_train
CPU_learn <- subset(CPU_data, is.na(Recommended_Customer_Price))
CPU_train <- subset(CPU_data, !is.na(Recommended_Customer_Price))
summary(CPU_train)
```

```
  Lithography      Recommended_Customer_Price  nb_of_Cores
 Min.   : 14.00   Min.   :  2.54             Min.   : 1.000
 1st Qu.: 14.00   1st Qu.:107.00             1st Qu.: 2.000
 Median : 22.00   Median :239.50             Median : 2.000
 Mean   : 26.57   Mean   :268.37             Mean   : 3.148
 3rd Qu.: 32.00   3rd Qu.:378.00             3rd Qu.: 4.000
 Max.   :130.00   Max.   :999.00             Max.   :16.000
 Processor_Base_Frequency Max_Turbo_Frequency      TDP
 Min.   :  32             Min.   :1.300        Min.   :  0.025
 1st Qu.:1800             1st Qu.:3.198        1st Qu.: 17.000
 Median :2300             Median :3.198        Median : 35.000
 Mean   :2304             Mean   :3.181        Mean   : 44.061
 3rd Qu.:2800             3rd Qu.:3.200        3rd Qu.: 60.242
 Max.   :4100             Max.   :4.500        Max.   :140.000
 Embedded_Options_Available Max_Memory_Size  Max_Memory_Bandwidth
 Min.   :0.0000             Min.   :  1.00   Min.   : 1.60
 1st Qu.:0.0000             1st Qu.: 16.00   1st Qu.:25.60
 Median :0.0000             Median : 32.00   Median :29.80
 Mean   :0.3235             Mean   : 78.03   Mean   :29.91
 3rd Qu.:1.0000             3rd Qu.:128.00   3rd Qu.:35.08
 Max.   :1.0000             Max.   :768.00   Max.   :85.30
```

```
ECC_Memory_Supported Graphics_Base_Frequency Graphics_Max_Dynamic_Frequency
Min.   :0.0000       Min.   :100.0           Min.   : 500
1st Qu.:0.0000       1st Qu.:327.5           1st Qu.:1000
Median :0.0000       Median :420.1           Median :1032
Mean   :0.3914       Mean   :405.9           Mean   :1031
3rd Qu.:1.0000       3rd Qu.:420.1           3rd Qu.:1050
Max.   :1.0000       Max.   :900.0           Max.   :1350
Graphics_Video_Max_Memory       T           Intel_Hyper_Threading_Technology_
Min.   : 1.00             Min.   : 53.90   Min.   :0.000
1st Qu.:22.81             1st Qu.: 73.30   1st Qu.:0.000
Median :22.81             Median : 83.25   Median :1.000
Mean   :22.54             Mean   : 86.18   Mean   :0.621
3rd Qu.:22.81             3rd Qu.:100.00   3rd Qu.:1.000
Max.   :64.00             Max.   :110.00   Max.   :1.000
Intel_Virtualization_Technology_VTx_   Intel_64_       Instruction_Set
Min.   :0.000                         Min.   :0.0000   Min.   :0.0000
1st Qu.:1.000                         1st Qu.:1.0000   1st Qu.:1.0000
Median :1.000                         Median :1.0000   Median :1.0000
Mean   :0.942                         Mean   :0.9691   Mean   :0.9679
3rd Qu.:1.000                         3rd Qu.:1.0000   3rd Qu.:1.0000
Max.   :1.000                         Max.   :1.0000   Max.   :2.0000
 Idle_States      Thermal_Monitoring_Technologies   Secure_Key
Min.   :0.0000   Min.   :0.0000                   Min.   :0.0000
1st Qu.:1.0000   1st Qu.:1.0000                   1st Qu.:1.0000
Median :1.0000   Median :1.0000                   Median :1.0000
Mean   :0.9704   Mean   :0.9259                   Mean   :0.9506
3rd Qu.:1.0000   3rd Qu.:1.0000                   3rd Qu.:1.0000
Max.   :1.0000   Max.   :1.0000                   Max.   :1.0000
Execute_Disable_Bit
Min.   :0.0000
1st Qu.:1.0000
Median :1.0000
Mean   :0.9975
3rd Qu.:1.0000
Max.   :1.0000
```

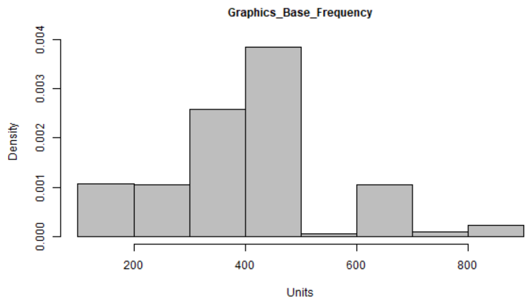
### 2.3.2 Plotting graph

#### 2.3.2.a Histogram

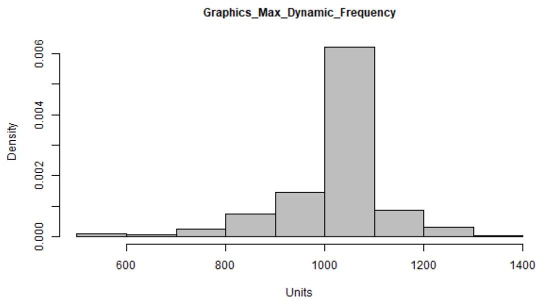
```r
variables<-colnames(CPU_train)
# Histogram
not_his = c("Embedded_Options_Available", "ECC_Memory_Supported", "Intel_Hyper_Threading_Technology_", "Intel_Virtualization_Technology_VTx_", "Intel_64_", "Idle_States", "Thermal_Monitoring_Technologies", "Secure_Key", "Execute_Disable_Bit", "Instruction_Set")
his_variables <- setdiff(variables, not_his)

for(var in his_variables){
  temp <-gsub(",","", CPU_train[var]);
  temp <-as.numeric(unlist(CPU_train[var]));
  hist(temp,
    main=var,
    col = "gray",
```

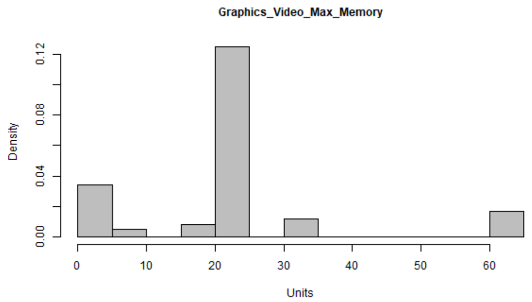
```
    xlab="Units",
    freq = FALSE,
    cex.main = 1
  );
}
```



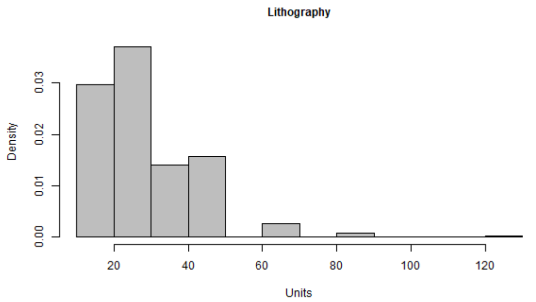

Concentrate around a point but not symmetric



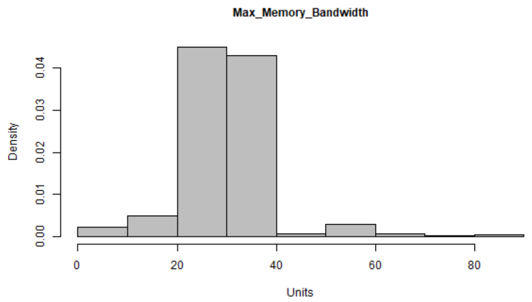

Symmetrically distributed



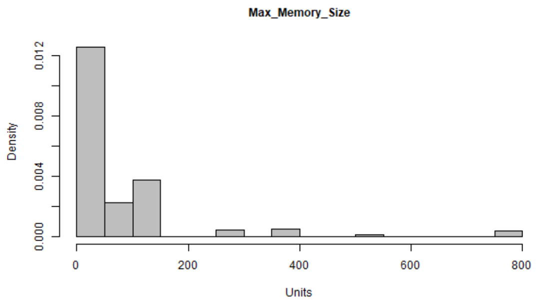

Exist an outlier



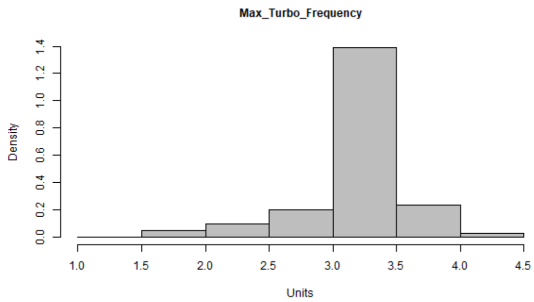

Right-skewed distribution



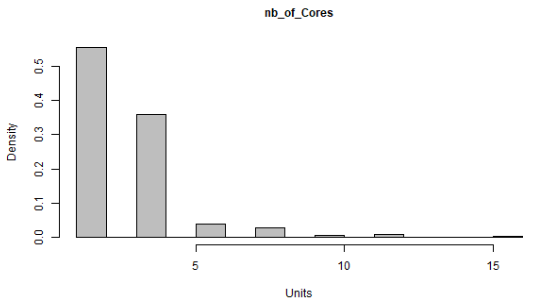

Concentrate around a point



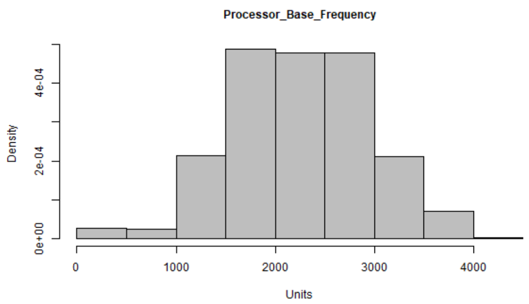

Right-skewed distribution



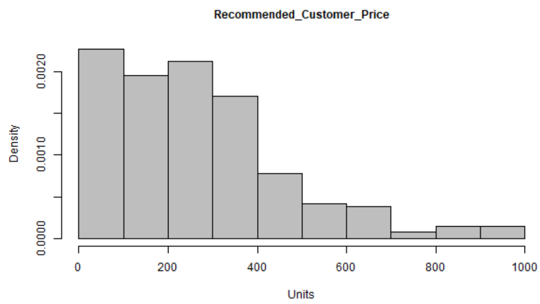

Left-skewed distribution



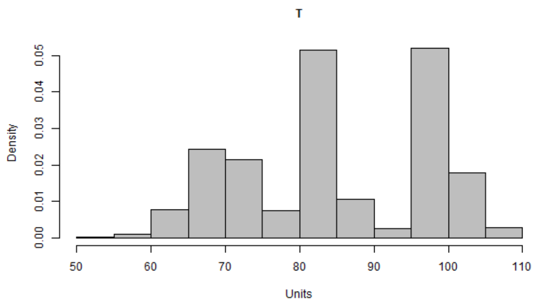

Distribution not good



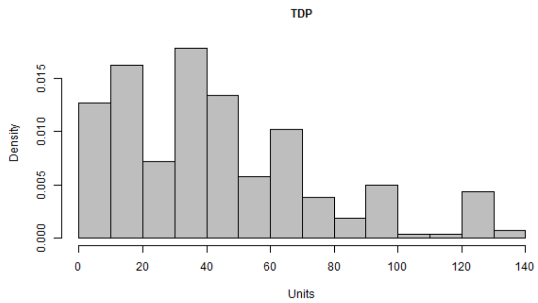

Symmetric distribution




Right-skewed distribution




Bimodal distribution



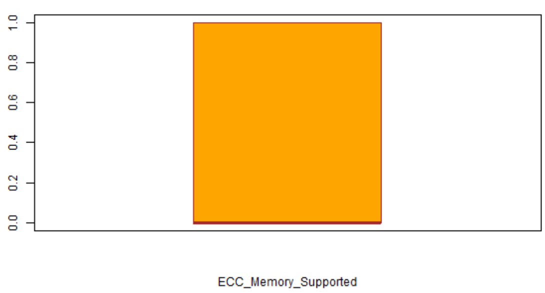

Right-skewed distribution

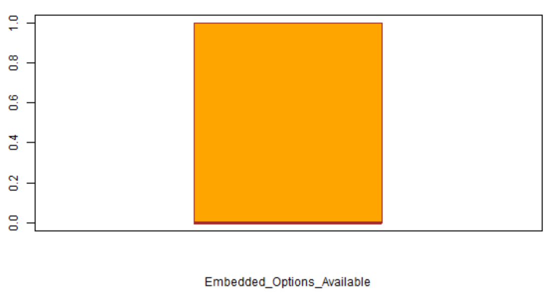
Figure 1: Histograms of data features

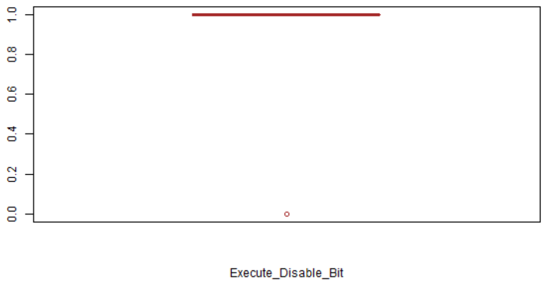
The data features have their own unique distribution, which contribute to the model later on.

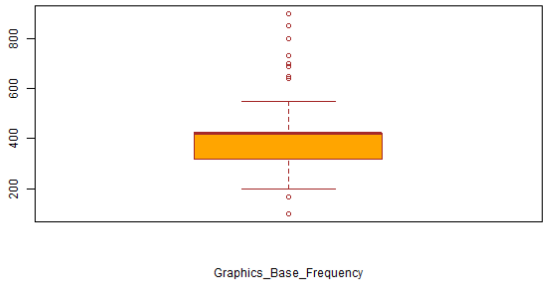
#### 2.3.2.b Box plot

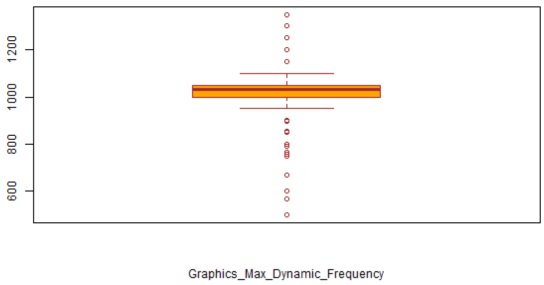
```
for (i in variables){
  boxplot(CPU_train[i],
    col="orange",
    xlab = i,
    cex.lab = 1,
    title.cex = 1,
    border="brown")
}
```

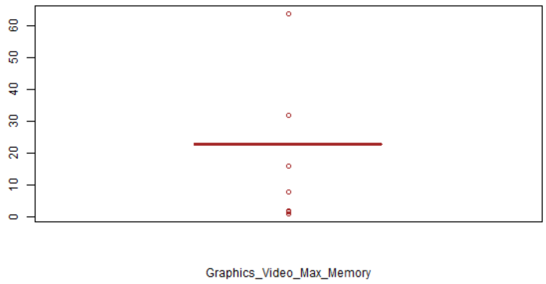


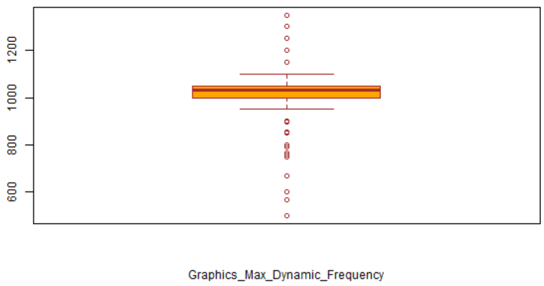



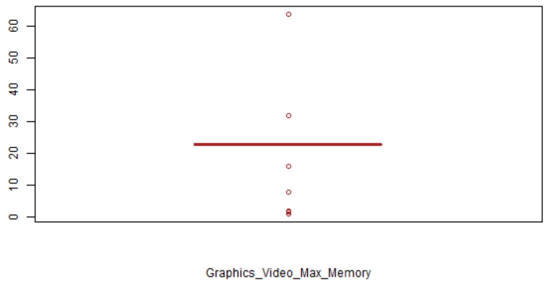

Execute_Disable_Bit


Graphics_Base_Frequency

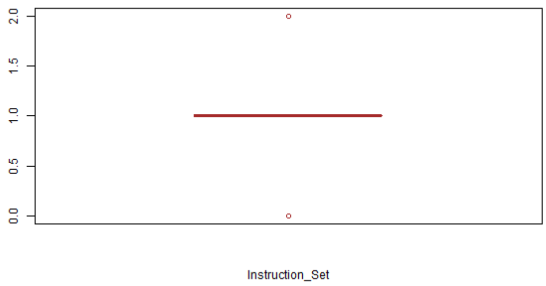

Graphics_Max_Dynamic_Frequency

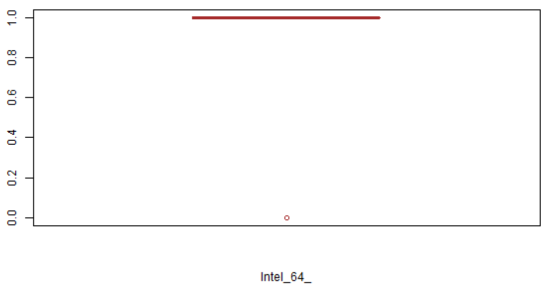

Graphics_Video_Max_Memory

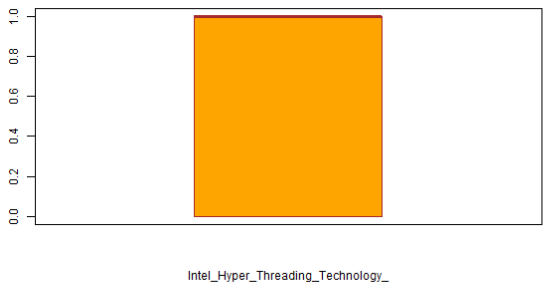

Graphics_Max_Dynamic_Frequency

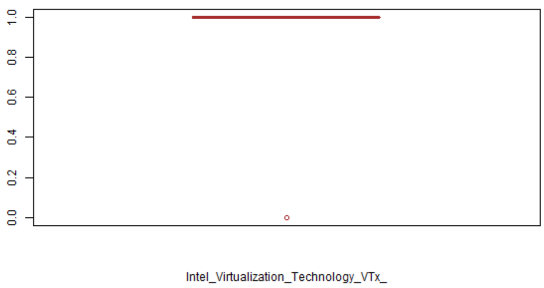

Graphics_Video_Max_Memory

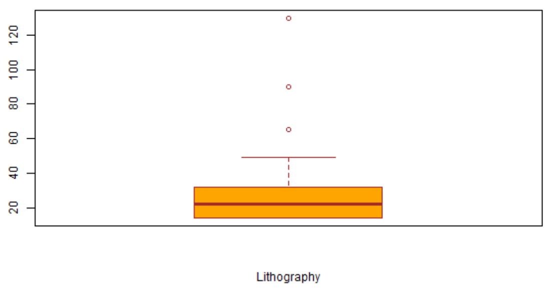

Idle_States


Instruction_Set


Intel_64_


Intel_Hyper_Threading_Technology_


Intel_Virtualization_Technology_VTx_


Lithography

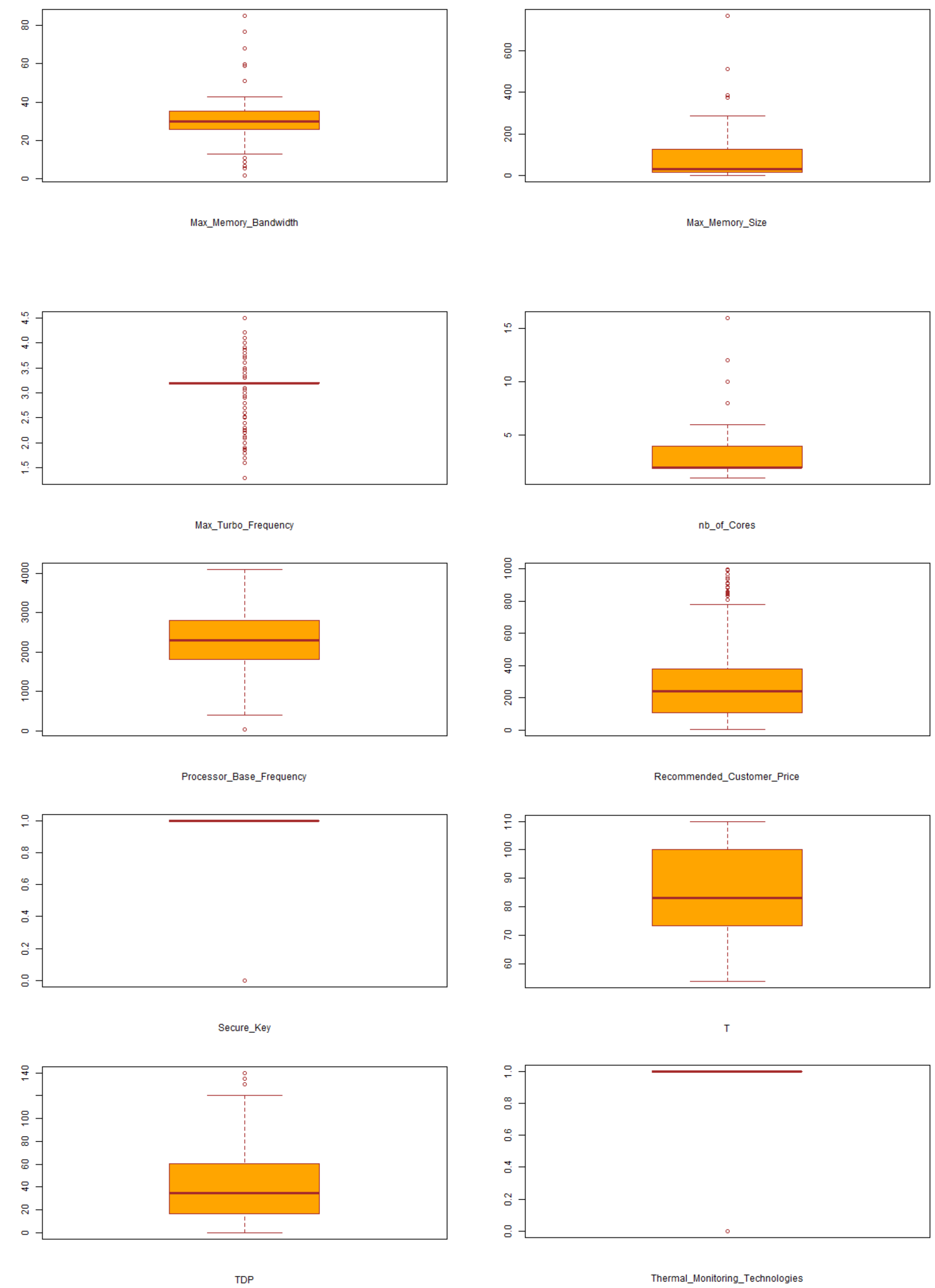



Figure 2: Scatter plots of data features

Some of the features are highly distributed and some other are highly concentrated.

#### 2.3.2.c Correlation matrix

```
# Correlation
# Calculate the correlation matrix
select <- c("Processor_Base_Frequency", "Lithography","nb_of_Cores","Max_Turbo_Frequency","Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order = "hclust", tl.col = "black", tl.srt = 45)
```

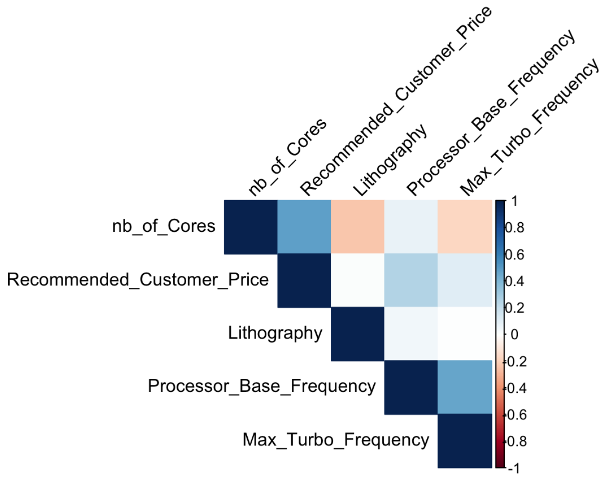



Figure 3: Correlation matrix (1)

```
# Calculate the correlation matrix
select <- c("Max_Memory_Bandwidth",  "TDP", "Embedded_Options_Available" ,"Max_Memory_Size" ,"Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order = "hclust", tl.col = "black", tl.srt = 45)
```

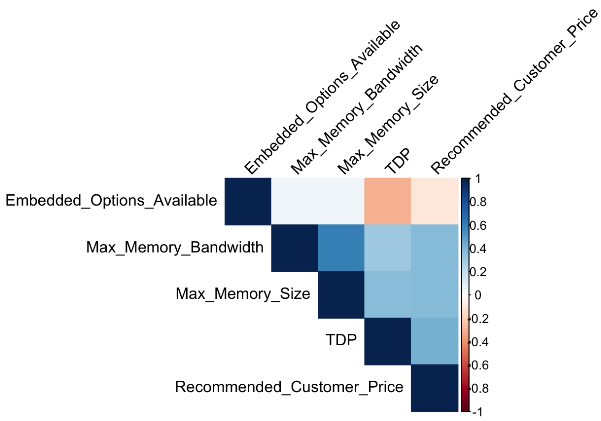



Figure 4: Correlation matrix (2)

```
# Calculate the correlation matrix
```

```
select <- c("ECC_Memory_Supported",   "Graphics_Base_Frequency",  "Graphics_Max_Dynamic_Frequency" ,"Graphics_Video_Max_Memory" ,"Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])

# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order = "hclust", tl.col = "black", tl.srt = 45)
```

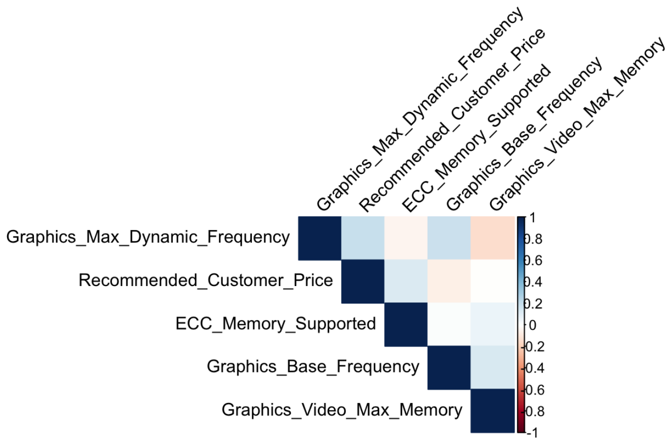



Figure 5: Correlation matrix (3)

```
# Calculate the correlation matrix
select <- c("T",   "Intel_Hyper_Threading_Technology_",  "Intel_Virtualization_Technology_VTx_" ,"Intel_64_","Instruction_Set"  ,"Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order = "hclust", tl.col = "black", tl.srt = 45)
```

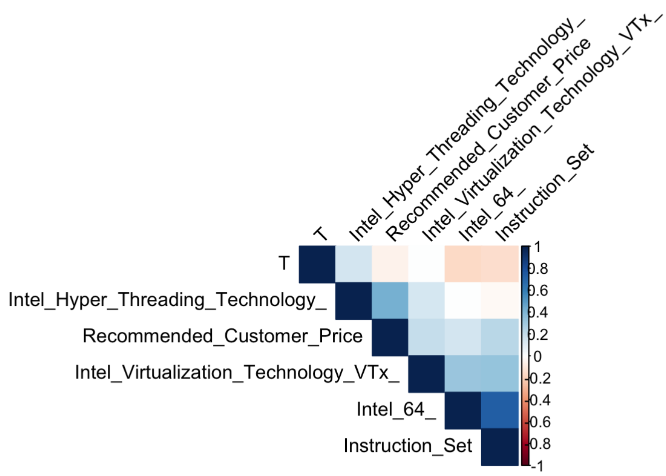



Figure 6: Correlation matrix (4)

```r
# Calculate the correlation matrix
select <- c("Idle_States",  "Thermal_Monitoring_Technologies",  "
    Secure_Key" , "Execute_Disable_Bit"  ,"Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order =
    "hclust", tl.col = "black", tl.srt = 45)
```

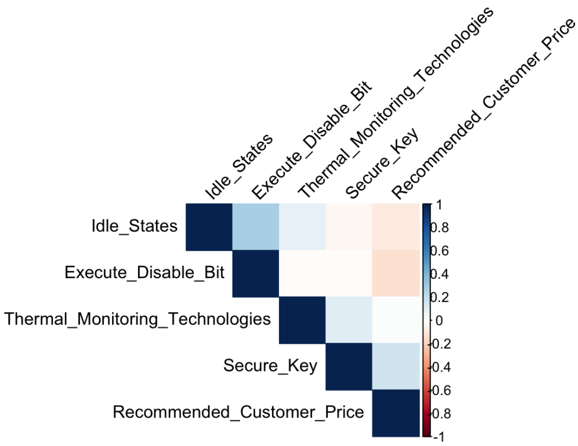



Figure 7: Correlation matrix (5)

In the correlation matrices, we respectively delete Lithography, Graphics Base Frequency, Graphics Video Max Memory, T, Idle states, Thermal monitoring technology due to low corelation.

## 2.4 Models building

### 2.4.1 Multivariate Linear Regression (MLR)

We divide CPU_train into 2 separate subsets, which are the train set and the test set. This is good practice for training prediction model.

```r
# Divide CPU_train into train_df and test_df
set.seed(42)
# Use 70% of dataset as training set and 30% as test set
split <- sample.split(CPU_train, SplitRatio = 0.70)
train_df <- subset(CPU_train, split == TRUE)
test_df <- subset(CPU_train, split == FALSE)
# Fitting model
model1 <- lm(Recommended_Customer_Price~., data = train_df)
summary(model1)
```

```
## Call:
## lm(formula = Recommended_Customer_Price ~ ., data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -331.63  -62.02   -2.56   49.79  689.49
##
## Coefficients:
```

```
##                                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)                            -9.696e+02  2.311e+02  -4.195 3.19e-05 ***
## Lithography                             2.607e+00  6.131e-01   4.253 2.49e-05 ***
## nb_of_Cores                             3.952e+01  4.101e+00   9.636  < 2e-16 ***
## Processor_Base_Frequency               -8.799e-03  1.362e-02  -0.646 0.518484
## Max_Turbo_Frequency                     3.212e+01  1.783e+01   1.802 0.072143 .
## TDP                                     1.542e+00  3.500e-01   4.405 1.28e-05 ***
## Embedded_Options_Available             -1.626e+01  1.497e+01  -1.087 0.277737
## Max_Memory_Size                         2.732e-01  7.160e-02   3.816 0.000151 ***
## Max_Memory_Bandwidth                    1.999e+00  7.553e-01   2.647 0.008371 **
## ECC_Memory_Supported                    4.462e+00  1.367e+01   0.326 0.744315
## Graphics_Base_Frequency                -7.534e-02  4.498e-02  -1.675 0.094523 .
## Graphics_Max_Dynamic_Frequency          1.845e-01  6.260e-02   2.947 0.003351 **
## Graphics_Video_Max_Memory              -7.518e-01  3.912e-01  -1.922 0.055146 .
## T                                       3.721e+00  5.665e-01   6.569 1.21e-10 ***
## Intel_Hyper_Threading_Technology_       1.331e+02  1.217e+01  10.937  < 2e-16 ***
## Intel_Virtualization_Technology_VTx_    1.004e+02  3.127e+01   3.210 0.001406 **
## Intel_64_                              -1.778e+02  8.299e+01  -2.143 0.032589 *
## Instruction_Set                         2.623e+02  7.430e+01   3.530 0.000452 ***
## Idle_States                            -5.362e+01  3.335e+01  -1.608 0.108501
## Thermal_Monitoring_Technologies         1.815e+01  2.669e+01   0.680 0.496678
## Secure_Key                              1.313e+01  3.024e+01   0.434 0.664268
## Execute_Disable_Bit                     1.141e+02  2.020e+02   0.565 0.572461
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 124.5 on 530 degrees of freedom
## Multiple R-squared:  0.6135, Adjusted R-squared:  0.5982
## F-statistic: 40.07 on 21 and 530 DF,  p-value: < 2.2e-16
```

We are going to use stepwise regression to minimize the number of predictors.

```r
# Stepwise
model2 <- stepAIC(model1, direction = "both")
summary(model2)
```

```
## Call:
## lm(formula = Recommended_Customer_Price ~ Lithography + nb_of_Cores +
##     Max_Turbo_Frequency + TDP + Max_Memory_Size + Max_Memory_Bandwidth +
##     Graphics_Base_Frequency + Graphics_Max_Dynamic_Frequency +
##     Graphics_Video_Max_Memory + T + Intel_Hyper_Threading_Technology_ +
##     Intel_Virtualization_Technology_VTx_ + Intel_64_ + Instruction_Set +
##     Idle_States, data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -328.09  -62.43   -1.53   55.45  691.04
##
## Coefficients:
##                                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)                            -853.83393  113.93898  -7.494 2.78e-13 ***
## Lithography                               2.56372    0.55937   4.583 5.70e-06 ***
## nb_of_Cores                              39.76142    3.89737  10.202  < 2e-16 ***
## Max_Turbo_Frequency                      32.48697   16.57622   1.960 0.050531 .
## TDP                                       1.44426    0.28265   5.110 4.49e-07 ***
## Max_Memory_Size                           0.27880    0.06897   4.043 6.06e-05 ***
## Max_Memory_Bandwidth                      2.04415    0.71365   2.864 0.004342 **
## Graphics_Base_Frequency                  -0.08689    0.04274  -2.033 0.042551 *
## Graphics_Max_Dynamic_Frequency            0.17887    0.05798   3.085 0.002141 **
```

```
## Graphics_Video_Max_Memory              -0.84533    0.37893  -2.231 0.026103 *
## T                                       3.65881    0.54728   6.685 5.79e-11 ***
## Intel_Hyper_Threading_Technology_     133.41953   11.70402  11.399  < 2e-16 ***
## Intel_Virtualization_Technology_VTx_  108.74411   28.33553   3.838 0.000139 ***
## Intel_64_                            -145.95678   53.21008  -2.743 0.006291 **
## Instruction_Set                       238.27994   49.66475   4.798 2.08e-06 ***
## Idle_States                           -45.87275   32.49853  -1.412 0.158668
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 124.1 on 536 degrees of freedom
## Multiple R-squared:  0.6118, Adjusted R-squared:  0.6009
## F-statistic:  56.3 on 15 and 536 DF,  p-value: < 2.2e-16
```

### 2.4.2 Analysis of Variance (ANOVA)

```
multiAnova <- aov(Recommended_Customer_Price ~ Lithography + nb_of_Cores + Max_Turbo_Frequency + TDP + Max_Memory_Size + Max_Memory_Bandwidth + Graphics_Base_Frequency +  Graphics_Max_Dynamic_Frequency + Graphics_Video_Max_Memory + T + Intel_Hyper_Threading_Technology_ + Intel_Virtualization_Technology_VTx_ + Intel_64_ + Instruction_Set + Idle_States, data = train_df)
summary(multiAnova)
```

```
##                                       Df  Sum Sq Mean Sq F value   Pr(>F)
## Lithography                            1    1231    1231   0.080 0.777447
## nb_of_Cores                            1 5560077 5560077 361.297  < 2e-16 ***
## Max_Turbo_Frequency                    1  894154  894154  58.103 1.14e-13 ***
## TDP                                    1 1020681 1020681  66.324 2.70e-15 ***
## Max_Memory_Size                        1  679416  679416  44.149 7.49e-11 ***
## Max_Memory_Bandwidth                   1  127876  127876   8.309 0.004102 **
## Graphics_Base_Frequency                1  295168  295168  19.180 1.43e-05 ***
## Graphics_Max_Dynamic_Frequency         1  173493  173493  11.274 0.000842 ***
## Graphics_Video_Max_Memory              1  105013  105013   6.824 0.009247 **
## T                                      1 1288664 1288664  83.738  < 2e-16 ***
## Intel_Hyper_Threading_Technology_      1 1950353 1950353 126.735  < 2e-16 ***
## Intel_Virtualization_Technology_VTx_   1  407478  407478  26.478 3.75e-07 ***
## Intel_64_                              1   49410   49410   3.211 0.073722 .
## Instruction_Set                        1  413499  413499  26.869 3.09e-07 ***
## Idle_States                            1   30662   30662   1.992 0.158668
## Residuals                            536 8248616   15389
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
av_residual= rstandard(aov(Recommended_Customer_Price~.,
             data =CPU_train))
shapiro.test(av_residual)
```

```
##  Shapiro-Wilk normality test
##
## data:  av_residual
## W = 0.94051, p-value < 2.2e-16
```

This dataset does not have normal distribution since p-value $< 0.05$, it means MLR is not efficient anough. Therefore, we apply the support vector machine method.

### 2.4.3 Support vector machine (SVM)

```
model3 <- svm(Recommended_Customer_Price ~ Lithography + nb_of_Cores +
    Max_Turbo_Frequency + TDP + Max_Memory_Size + Max_Memory_Bandwidth +
     Graphics_Base_Frequency +  Graphics_Max_Dynamic_Frequency +
    Graphics_Video_Max_Memory + T + Intel_Hyper_Threading_Technology_ +
    Intel_Virtualization_Technology_VTx_ + Intel_64_ + Instruction_Set +
     Idle_States, data = train_df)
```

## 2.5 Model comparison

```
anova(model1, model2)
```

```
## Analysis of Variance Table
##
## Model 1: Recommended_Customer_Price ~ Lithography + nb_of_Cores +
    Processor_Base_Frequency +
##     Max_Turbo_Frequency + TDP + Embedded_Options_Available +
##     Max_Memory_Size + Max_Memory_Bandwidth + ECC_Memory_Supported +
##     Graphics_Base_Frequency + Graphics_Max_Dynamic_Frequency +
##     Graphics_Video_Max_Memory + T + Intel_Hyper_Threading_Technology_ +
##     Intel_Virtualization_Technology_VTx_ + Intel_64_ + Instruction_Set +
##     Idle_States + Thermal_Monitoring_Technologies + Secure_Key +
##     Execute_Disable_Bit
## Model 2: Recommended_Customer_Price ~ Lithography + nb_of_Cores + Max_Turbo_Frequency
     +
##     TDP + Max_Memory_Size + Max_Memory_Bandwidth + Graphics_Base_Frequency +
##     Graphics_Max_Dynamic_Frequency + Graphics_Video_Max_Memory +
##     T + Intel_Hyper_Threading_Technology_ + Intel_Virtualization_Technology_VTx_ +
##     Intel_64_ + Instruction_Set + Idle_States
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1    530 8210950
## 2    536 8248616 -6    -37665 0.4052 0.8757
```

With p-value greater than 0.05, model1 and model2 have equivalent efficiency. However, R-squared of model2 is greater than model1 0.6009 > 0.5982, model2 is a little better than model1. Therefore, we will compare accuracy between model2 and model3.

```
pred_test=predict(model1,newdata=test_df)
SSE <- sum((test_df$Recommended_Customer_Price - pred_test) ^ 2) #Sum
    of Squares Error
SST <- sum((test_df$Recommended_Customer_Price - mean(test_df$
    Recommended_Customer_Price)) ^ 2) #Sum of Squares Total
cat("The accuracy of the model on test dataset: ",round((1 - SSE/SST)*
    100,2),"%")

pred_test=predict(model2,newdata=test_df)
SSE <- sum((test_df$Recommended_Customer_Price - pred_test) ^ 2) #Sum
    of Squares Error
SST <- sum((test_df$Recommended_Customer_Price - mean(test_df$
    Recommended_Customer_Price)) ^ 2) #Sum of Squares Total
cat("The accuracy of the model on test dataset: ",round((1 - SSE/SST)*
    100,2),"%")
```

```
## The accuracy of the model on test dataset:  60.32 %
## The accuracy of the model on test dataset:  62.28 %
```

Accuracy of model3 is 62.28%, which is more than accuracy of model2 (60.32%), thus, model3 is the final model.

# 3 Conclusion

In this report, we imported the 'Intel_CPUs.csv' file for examination and performed data cleaning and data visualizations. Subsequently, we analyzed the processed dataset of 2283 CPU observations with 22 variables, including one target variable, "Recommended_Customer_Price", which contained missing values. To address these missing values, we employed linear models, evaluating the performance of various models and selecting the best fit model as our final method.

Our findings demonstrate the effectiveness of linear models in imputing missing values for CPU prices. The selected model successfully predicted missing values with high accuracy, utilizing the information provided by the remaining 21 variables. The analysis of the dataset also revealed several insights into the factors influencing CPU prices. These findings highlight the importance of these hardware specifications in determining the overall value of a CPU. This approach allowed us to utilize the complete dataset for further analysis and modeling efforts.

Despite our best efforts, due to the lack of knowledge and techniques and the limitations of the dataset, there might be some inaccuracies in this report. We would greatly appreciate your valuable feedback to help us improve our analysis and make it more robust in the future.

# 4 R script

```
## Import data
# Import file
CPU_data = read.csv("~/Desktop/XSTK Assignment/work/Intel_CPUs.csv")
head(CPU_data, 3)
# Import libraries
library(dplyr)
library(stringr)
library(GGally)
library(corrplot)
library(caTools)
library(MASS)
library(car)
library(e1071)
library(nortest)
## Data cleaning
# Checking missing values
apply(is.na(CPU_data), 2, sum)
# Remove unused variables
delete = c("OpenGL_Support", "Support_4k", "Processor_Graphics_", "Product_Collection", "Vertical_Segment", "Processor_Number", "Status", "Launch_Date","nb_of_Threads", "Instruction_Set_Extensions","PCI_Express_Configurations_","Max_Resolution_eDP_Integrated_Flat_Panel", "Max_Resolution_DP","Graphics_Output","Max_Resolution_HDMI","Memory_Types","Bus_Speed","Cache","DirectX_Support","Max_nb_of_PCI_Express_Lanes", "Max_nb_of_Memory_Channels", "PCI_Express_Revision", "Conflict_Free")
CPU_data <- CPU_data[, !names(CPU_data) %in% delete, drop = FALSE]
# Convert " " and "\n- " to NA
CPU_data[(CPU_data == "") | (CPU_data == "\n- ")] <- NA
# Remove unit of numerical variables and transform to numeric form
CPU_data$Lithography <- as.numeric(sub("nm", "", CPU_data$Lithography))
CPU_data$Max_Turbo_Frequency <- as.numeric(sub("GHz", "", CPU_data$Max_Turbo_Frequency))
CPU_data$TDP <- as.numeric(sub("W", "", CPU_data$TDP))
CPU_data$Max_Memory_Size <- as.numeric(sub("GB", "", CPU_data$Max_Memory_Size))
CPU_data$Max_Memory_Bandwidth <- as.numeric(sub("GB/s", "", CPU_data$Max_Memory_Bandwidth))
CPU_data$Graphics_Base_Frequency <- as.numeric(sub("MHz", "", CPU_data$Graphics_Base_Frequency))
CPU_data$Graphics_Video_Max_Memory <- as.numeric(sub("GB", "", CPU_data$Graphics_Video_Max_Memory))
CPU_data$T <- as.numeric(sub(" C ", "", CPU_data$T))
# Convert units for Graphics_Max_Dynamic_Frequency
subset_GHz <- CPU_data[grepl("GHz", CPU_data$Graphics_Max_Dynamic_
```

```
Frequency, ignore.case = TRUE) , ]
CPU_data <- CPU_data[!grepl("GHz", CPU_data$Graphics_Max_Dynamic_Frequency, ignore.case = TRUE), ]
subset_GHz$Graphics_Max_Dynamic_Frequency <- gsub("GHz", "",subset_GHz$Graphics_Max_Dynamic_Frequency,fixed = TRUE)
subset_GHz$Graphics_Max_Dynamic_Frequency <- as.numeric(subset_GHz$Graphics_Max_Dynamic_Frequency)
subset_GHz$Graphics_Max_Dynamic_Frequency <- subset_GHz$Graphics_Max_Dynamic_Frequency*(1000)
CPU_data$Graphics_Max_Dynamic_Frequency <- gsub("MHz", "",CPU_data$Graphics_Max_Dynamic_Frequency,fixed = TRUE)
CPU_data$Graphics_Max_Dynamic_Frequency <- as.numeric(CPU_data$Graphics_Max_Dynamic_Frequency)
CPU_data <- bind_rows(CPU_data,subset_GHz)
# Convert units for Processor_Base_Frequency
subset_GHz <- CPU_data[grepl("GHz", CPU_data$Processor_Base_Frequency, ignore.case = TRUE) , ]
CPU_data <- CPU_data[!grepl("GHz", CPU_data$Processor_Base_Frequency, ignore.case = TRUE), ]
subset_GHz$Processor_Base_Frequency <- gsub("GHz", "",subset_GHz$Processor_Base_Frequency,fixed = TRUE)
subset_GHz$Processor_Base_Frequency <- as.numeric(subset_GHz$Processor_Base_Frequency)
subset_GHz$Processor_Base_Frequency <- subset_GHz$Processor_Base_Frequency*(1000)
CPU_data$Processor_Base_Frequency <- gsub("MHz", "",CPU_data$Processor_Base_Frequency,fixed = TRUE)
CPU_data$Processor_Base_Frequency <- as.numeric(CPU_data$Processor_Base_Frequency)
CPU_data <- bind_rows(CPU_data,subset_GHz)
# Filling missing values for numerical variables
CPU_data$Lithography[is.na(CPU_data$Lithography)] = mean(CPU_data$Lithography, na.rm=T)
CPU_data$nb_of_Cores[is.na(CPU_data$nb_of_Cores)] = mean(CPU_data$nb_of_Cores, na.rm=T)
CPU_data$Max_Turbo_Frequency[is.na(CPU_data$Max_Turbo_Frequency)] = mean(CPU_data$Max_Turbo_Frequency, na.rm=T)
CPU_data$TDP[is.na(CPU_data$TDP)] = mean(CPU_data$TDP, na.rm=T)
CPU_data$Max_Memory_Size[is.na(CPU_data$Max_Memory_Size)] = mean(CPU_data$Max_Memory_Size, na.rm=T)
CPU_data$Max_Memory_Bandwidth[is.na(CPU_data$Max_Memory_Bandwidth)] = mean(CPU_data$Max_Memory_Bandwidth, na.rm=T)
CPU_data$Graphics_Base_Frequency[is.na(CPU_data$Graphics_Base_Frequency)] = mean(CPU_data$Graphics_Base_Frequency, na.rm=T)
CPU_data$Graphics_Video_Max_Memory[is.na(CPU_data$Graphics_Video_Max_Memory)] = mean(CPU_data$Graphics_Video_Max_Memory, na.rm=T)
CPU_data$T[is.na(CPU_data$T)] = mean(CPU_data$T, na.rm=T)
```

```r
CPU_data$Graphics_Max_Dynamic_Frequency[is.na(CPU_data$Graphics_Max_Dynamic_Frequency)] = mean(CPU_data$Graphics_Max_Dynamic_Frequency, na.rm=T)
CPU_data$Processor_Base_Frequency[is.na(CPU_data$Processor_Base_Frequency)] = mean(CPU_data$Processor_Base_Frequency, na.rm=T)
CPU_data$Recommended_Customer_Price <- gsub("$", "", CPU_data$Recommended_Customer_Price, fixed = TRUE)
CPU_data$Recommended_Customer_Price <- as.numeric(CPU_data$Recommended_Customer_Price)
# Filling missing values for categorical variables
fillmode <- function(column) {
  mode_value <- names(sort(table(column), decreasing = TRUE))[1]
  column[is.na(column)] <- mode_value
  return(column)
}
fill_col <- c("Embedded_Options_Available","ECC_Memory_Supported","Intel_Hyper_Threading_Technology_","Intel_Virtualization_Technology_VTx_","Intel_64_","Instruction_Set","Idle_States","Thermal_Monitoring_Technologies","Secure_Key","Execute_Disable_Bit")
CPU_data[fill_col] <- lapply(CPU_data[fill_col], fillmode)
# Transform categorical variables to numerical variables
CPU_data$Embedded_Options_Available <- ifelse(CPU_data$Embedded_Options_Available == "Yes", 1, 0)
CPU_data$ECC_Memory_Supported <- ifelse(CPU_data$ECC_Memory_Supported == "Yes", 1, 0)
CPU_data$Intel_Hyper_Threading_Technology_ <- ifelse(CPU_data$Intel_Hyper_Threading_Technology_ == "Yes", 1, 0)
CPU_data$Intel_Virtualization_Technology_VTx_ <- ifelse(CPU_data$Intel_Virtualization_Technology_VTx_ == "Yes", 1, 0)
CPU_data$Intel_64_ <- ifelse(CPU_data$Intel_64_ == "Yes", 1, 0)
CPU_data$Idle_States <- ifelse(CPU_data$Idle_States == "Yes", 1, 0)
CPU_data$Thermal_Monitoring_Technologies <- ifelse(CPU_data$Thermal_Monitoring_Technologies == "Yes", 1, 0)
CPU_data$Secure_Key <- ifelse(CPU_data$Secure_Key == "Yes", 1, 0)
CPU_data$Execute_Disable_Bit <- ifelse(CPU_data$Execute_Disable_Bit == "Yes", 1, 0)
CPU_data$Instruction_Set <- gsub("32-bit", "0",CPU_data$Instruction_Set,fixed = TRUE)
CPU_data$Instruction_Set <- gsub("64-bit", "1",CPU_data$Instruction_Set,fixed = TRUE)
CPU_data$Instruction_Set <- gsub("Itanium 1", "2",CPU_data$Instruction_Set,fixed = TRUE)
CPU_data$Instruction_Set <- as.numeric(CPU_data$Instruction_Set)
# Checking missing values
apply(is.na(CPU_data), 2, sum)

## Data visualization
```

```r
# Divide data set
CPU_learn <- subset(CPU_data, is.na(Recommended_Customer_Price))
CPU_train <- subset(CPU_data, !is.na(Recommended_Customer_Price))
summary(CPU_train)
# Variable for plotting
variables<-colnames(CPU_train)
# Histogram
not_his = c("Embedded_Options_Available", "ECC_Memory_Supported", "
    Intel_Hyper_Threading_Technology_", "Intel_Virtualization_Technology
    _VTx_", "Intel_64_", "Idle_States", "Thermal_Monitoring_Technologies
    ", "Secure_Key", "Execute_Disable_Bit", "Instruction_Set")
his_variables <- setdiff(variables, not_his)
for(var in his_variables){
  temp <-gsub(",","", CPU_train[var]);
  temp <-as.numeric(unlist(CPU_train[var]));
  hist(temp,
       main=var,
       col = "gray",
       xlab="Units",
       freq = FALSE,
       cex.main = 1
  );
}
# Box plot
for (i in variables){
  boxplot(CPU_train[i],
          col="orange",
          xlab = i,
          cex.lab = 1,
          title.cex = 1,
          border="brown")
}
# Correlation
# Calculate the correlation matrix
select <- c("Processor_Base_Frequency", "Lithography","nb_of_Cores","
    Max_Turbo_Frequency","Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order =
    "hclust", tl.col = "black", tl.srt = 45)
# Calculate the correlation matrix
select <- c("Max_Memory_Bandwidth",  "TDP", "Embedded_Options_Available
    " ,"Max_Memory_Size" ,"Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order =
    "hclust", tl.col = "black", tl.srt = 45)
```

```
# Calculate the correlation matrix
select <- c("ECC_Memory_Supported",   "Graphics_Base_Frequency",  "
    Graphics_Max_Dynamic_Frequency" ,"Graphics_Video_Max_Memory" ,"
    Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order =
    "hclust", tl.col = "black", tl.srt = 45)
# Calculate the correlation matrix
select <- c("T",   "Intel_Hyper_Threading_Technology_",  "Intel_
    Virtualization_Technology_VTx_" ,"Intel_64_","Instruction_Set"  ,"
    Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order =
    "hclust", tl.col = "black", tl.srt = 45)
# Calculate the correlation matrix
select <- c("Idle_States",   "Thermal_Monitoring_Technologies",  "
    Secure_Key" , "Execute_Disable_Bit"  ,"Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order =
    "hclust", tl.col = "black", tl.srt = 45)

## Models building
# Divide CPU_train into train_df and test_df
set.seed(42)
# Use 70% of dataset as training set and 30% as test set
split <- sample.split(CPU_train, SplitRatio = 0.70)
train_df <- subset(CPU_train, split == TRUE)
test_df <- subset(CPU_train, split == FALSE)
# Fitting model
model1 <- lm(Recommended_Customer_Price~., data =train_df)
summary(model1)
# Stepwise regression
model2 <- stepAIC(model1, direction = "both")
summary(model2)
# Multi-factor ANOVA
multiAnova <- aov(Recommended_Customer_Price ~ Lithography + nb_of_
    Cores + Max_Turbo_Frequency + TDP + Max_Memory_Size + Max_Memory_
    Bandwidth + Graphics_Base_Frequency +  Graphics_Max_Dynamic_
    Frequency + Graphics_Video_Max_Memory + T + Intel_Hyper_Threading_
    Technology_ + Intel_Virtualization_Technology_VTx_ + Intel_64_ +
    Instruction_Set + Idle_States, data = train_df)
summary(multiAnova)
# Apply support vector machine
model3 <- svm(Recommended_Customer_Price ~ Lithography + nb_of_Cores +
```

```
    Max_Turbo_Frequency + TDP + Max_Memory_Size + Max_Memory_Bandwidth +
     Graphics_Base_Frequency +  Graphics_Max_Dynamic_Frequency +
    Graphics_Video_Max_Memory + T + Intel_Hyper_Threading_Technology_ +
    Intel_Virtualization_Technology_VTx_ + Intel_64_ + Instruction_Set +
     Idle_States, data = train_df)

## Models comparison
anova(model1, model2)
pred_test = predict(model2, newdata = test_df)
SSE <- sum((test_df$Recommended_Customer_Price - pred_test) ^ 2) #Sum
    of Squares Error
SST <- sum((test_df$Recommended_Customer_Price - mean(test_df$
    Recommended_Customer_Price)) ^ 2) #Sum of Squares Total
cat("The accuracy of the model on test dataset: ",round((1 - SSE/SST)*
    100,2),"%")
pred_test = predict(model3, newdata = test_df)
SSE <- sum((test_df$Recommended_Customer_Price - pred_test) ^ 2) #Sum
    of Squares Error
SST <- sum((test_df$Recommended_Customer_Price - mean(test_df$
    Recommended_Customer_Price)) ^ 2) #Sum of Squares Total
cat("The accuracy of the model on test dataset: ",round((1 - SSE/SST)*
    100,2),"%")
```

# References

[1] Douglas C. Montgomery, George C. Runger, *Applied Statistics and Probability for Engineers*, Wiley, Hoboken, NJ, 2019.

[2] Prof. Nabendu Pal (2023), *Probability and Statistics lectures' notes.*

[3] GeeksforGeeks, *Support Vector Machine (SVM) algorithm.* Available at: https://www.geeksforgeeks.org/support-vector-machine-algorithm/