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 242

Assignment

Computer Parts (CPUs and GPUs)

Advisor: Nguyễn Tiến Dũng
Class: CC02
Students: Phạm Lê Hữu Hiệp - 2252223
Huỳnh Nga - 2252508
Lê Thanh Lâm - 2252421
Lê Nguyễn Minh Duẫn - 2252108

HO CHI MINH CITY, APRIL 2024

Contents

1 Theoretical foundation

1.1 Linear regression

Linear regression is a statistical model which estimates the linear relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables).

1.1.1 Two types of linear regression

Simple linear regression: aims to find a linear relationship to describe the correlation between an independent variable x and possibly dependent variable Y. .

The expected value of Y at each level of x is a random variable.

$$\hat{Y} = E(Y|x) = \alpha + \beta x$$

.

We assume that each observation, Y, can be described by the model.

$$Y = \alpha + \beta x + \epsilon$$

.

That is. Each pair of observations satisfies the relationship.

$$Y_i = \alpha_i + \beta_i x + \epsilon_i (i = 1, 2, 3, ..., n)$$

.

where $\epsilon_i = Y_i - \hat{Y}_i$ is called the residual describing the error in the fit of the model to the i observation Y_i.

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 α and β, to reach the best values of both ones.

Multiple linear regression: refers to a statistical technique that uses two or more independent variables to predict the outcome of a dependent variable.

Multiple Linear Regression Formula:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + ... + \beta_k x_{i,k} + \epsilon_i$$

where:

- Y_i is the dependent or predicted variable.
- β_0 is the y-intercept.
- $\beta_1 \beta_2 ... \beta_k$ are the regression coefficients.
- ϵ is the model's random error (residual) term.

1.1.2 Linear regression model

Linear regression is a powerful tool for understanding and predicting the behavior of a variable. However, there are few assumptions to check the data in order to be accurate and dependable solutions.

6 Assumptions of linear regression include:

- **Linearity:** he relationship between the dependent and independent variables is linear.
- **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:** The variance of the errors is constant across all levels of the independent variables.
- **Normality:** The errors follow a normal distribution.
- **No multicollinearity:** The independent variables are not highly correlated with each other.
- **No endogeneity:** There is no relationship between the errors and the independent variables.

1.1.3 Cost function

The cost function measures the difference between the predicted values $\hat{Y}$ of the model and the actual target values Y. It is the Mean Squared Error (MSE) between the predicted value and the true value. The formula of the cost function:

$$Costfunction(J) = \frac{1}{n} + \sum_{i=1}^{n}(\hat{y_i} - y_i)^2$$

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:

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

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 method used to model the relationship between a dependent variable and one or more independent variables. The aim is to establish a linear relationship that can be used for prediction or inference. In linear regression, the relationship between the independent variables and the dependent variable is modeled as a linear equation. Linear regression assumes that there is a linear relationship between the variables, the residuals (errors) are normally distributed, and there is no multicollinearity among the independent variables.

Stepwise regression is a variable selection technique used to determine the most relevant subset of independent variables for predicting the dependent variable. It aims to improve the predictive accuracy of the model and/or to identify the most important predictors. Stepwise regression involves iteratively adding or removing independent variables from the model based on a specified criterion. Stepwise regression relies on assumptions similar to those of linear regression, including linearity between variables, normality of residuals, and absence of multicollinearity.

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 aggre gate 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 factors that affect a given data set. Once the test is finished, further analysis is carried out on the systematic factors that significantly influence the variability within the dataset. Utilizing the results from the ANOVA test, analysts employ an f-test to gather supplementary data that corresponds to the suggested regression models.

The ANOVA test facilitates simultaneous comparison of two groups to ascertain the presence of any relationship among them. The outcome of the ANOVA formula, known as the F statistic or F-ratio, enables the examination of variability both between and within different sets of data.

If no real difference exists 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.

$$F = \frac{MST}{MSE} = \frac{\frac{SST}{I-1}}{\frac{SSE}{N-1}}$$

F: ANOVA Coefficient

$MST = \dfrac{SST}{I-1}$ mean square for treatment

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

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. The one-way ANOVA is used to determine whether there are any 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. For example, a two-way ANOVA allows a company to compare worker productivity based on two independent variables, such as salary and skill set. It is utilized to observe the interaction between the two factors and 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 Decision Trees and Random Forests

Decision Trees are the building blocks of Random Forests, and the ensemble approach of Random Forests leverages the strengths of multiple trees to improve predictive performance and robustness. Random Forests are widely used in various domains due to their versatility and effectiveness in handling complex datasets.

1.4.1 Decision Trees

A Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. It's essentially a flowchart-like tree structure where each internal node represents a "decision" based on the value of a feature attribute, each branch represents the outcome of the decision, and each leaf node represents the final decision or target value.

- **Tree Construction:** Decision Trees are constructed recursively from a training dataset by selecting the best attribute (feature) at each node to split the data into subsets. This process continues until all instances belong to the same target class or have similar target values, or a predefined stopping criterion is met (e.g., maximum depth, minimum number of samples).
- **Splitting Criteria:** The algorithm uses various measures (e.g., Gini impurity, information gain, entropy) to determine the best attribute to split the data at each node. The goal is to minimize impurity or maximize information gain, leading to more homogeneous subsets.
- **Pruning:** Decision Trees can be prone to overfitting, especially when the tree grows deep. Pruning techniques are used to simplify the tree by removing nodes that provide little predictive power or do not improve performance on validation data.

1.4.2 Random Forests

Random Forest is an ensemble learning method that operates by constructing a multitude of Decision Trees during training and outputting the mode (classification) or mean (regression) prediction of the individual trees.

- **Bootstrap Sampling:** Random Forest employs bootstrapping, a resampling technique, to create multiple training datasets by randomly sampling with replacement from the original dataset. Each sample is used to grow a separate Decision Tree.
- **Feature Randomness:** At each node of the Decision Tree, instead of considering all features to determine the best split, Random Forest selects a random subset of features. This introduces randomness and decorrelation among the trees, reducing the risk of overfitting and improving generalization.
- **Voting or Averaging:** For classification tasks, the final prediction is determined by a majority vote among the individual trees. For regression tasks, the final prediction is the average of the predictions made by all trees.
- **Advantages:** Random Forests are robust against overfitting, handle high-dimensional data well, and are less sensitive to noisy data compared to individual Decision Trees. They often yield higher accuracy and better generalization performance.

2 Data processing

2.1 Data importing

2.1.1 Import file

The "CPU_data" dataset provided is stored in CSV (Comma-Separated Values) format, which is a plain text file that stores data by delimiting data entries with commas. CSV files are often used when data needs to be compatible with many different programs. CSVs can be opened in text editors, spreadsheet programs like Excel, or other specialized applications.

In order to import the data from CSV file into R, we use the read.csv() command and name it **"CPU_data"**:

```
CPU_data <- read.csv("D:/Intel_CPUs.csv")
```

```
# Viewing the data structure
str(CPU_data)
```

```
'data.frame':        2283 obs. of  45 variables:
 $ Product_Collection                    : chr  "7th Generation IntelÂ® Coreâ\u0084¢ i7
 ↪  Processors" "8th Generation IntelÂ® Coreâ\u0084¢ i5 Processors" "8th Generation IntelÂ®
 ↪  Coreâ\u0084¢ i7 Processors" "IntelÂ® Coreâ\u0084¢ X-series Processors" ...
 $ Vertical_Segment                      : chr  "Mobile" "Mobile" "Mobile" "Desktop" ...
 $ Processor_Number                      : chr  "i7-7Y75" "i5-8250U" "i7-8550U" "i7-3820" ...
 $ Status                                : chr  "Launched" "Launched" "Launched" "End of Life"
 ↪  ...
 $ Launch_Date                           : chr  "Q3'16" "Q3'17" "Q3'17" "Q1'12" ...
 $ Lithography                           : chr  "14 nm" "14 nm" "14 nm" "32 nm" ...
 $ Recommended_Customer_Price            : chr  "$393.00 " "$297.00 " "$409.00 " "$305.00 "
 ↪  ...
 $ nb_of_Cores                           : int  2 4 4 4 2 2 2 2 2 1 ...
 $ nb_of_Threads                         : int  4 8 8 8 4 2 2 2 2 NA ...
 $ Processor_Base_Frequency              : chr  "1.30 GHz" "1.60 GHz" "1.80 GHz" "3.60 GHz"
 ↪  ...
 $ Max_Turbo_Frequency                   : chr  "3.60 GHz" "3.40 GHz" "4.00 GHz" "3.80 GHz"
 ↪  ...
```

```
$ Cache                                    : chr  "4 MB SmartCache" "6 MB SmartCache" "8 MB
↪  SmartCache" "10 MB SmartCache" ...
$ Bus_Speed                                : chr  "4 GT/s OPI" "4 GT/s OPI" "4 GT/s OPI" "5 GT/s
↪  DMI2" ...
$ TDP                                      : chr  "4.5 W" "15 W" "15 W" "130 W" ...
$ Embedded_Options_Available               : chr  "No" "No" "No" "No" ...
$ Conflict_Free                            : chr  "Yes" "Yes" "Yes" "" ...
$ Max_Memory_Size                          : chr  "16 GB" "32 GB" "32 GB" "64.23 GB" ...
$ Memory_Types                             : chr  "LPDDR3-1866, DDR3L-1600" "DDR4-2400,
↪  LPDDR3-2133" "DDR4-2400, LPDDR3-2133" "DDR3 1066/1333/1600" ...
$ Max_nb_of_Memory_Channels                : int  2 2 2 4 2 2 1 2 2 NA ...
$ Max_Memory_Bandwidth                     : chr  "29.8 GB/s" "34.1 GB/s" "34.1 GB/s" "51.2
↪  GB/s" ...
$ ECC_Memory_Supported                     : chr  "No" "No" "No" "No" ...
$ Processor_Graphics_                      : logi  NA NA NA NA NA NA ...
$ Graphics_Base_Frequency                  : chr  "300 MHz" "300 MHz" "300 MHz" "" ...
$ Graphics_Max_Dynamic_Frequency           : chr  "1.05 GHz" "1.10 GHz" "1.15 GHz" "" ...
$ Graphics_Video_Max_Memory                : chr  "16 GB" "32 GB" "32 GB" "" ...
$ Graphics_Output                          : chr  "eDP/DP/HDMI/DVI" "eDP/DP/HDMI/DVI"
↪  "eDP/DP/HDMI/DVI" "" ...
$ Support_4k                               : logi  NA NA NA NA NA NA ...
$ Max_Resolution_HDMI                      : chr  "4096x2304@24Hz" "4096x2304@24Hz"
↪  "4096x2304@24Hz" "" ...
$ Max_Resolution_DP                        : chr  "3840x2160@60Hz" "4096x2304@60Hz"
↪  "4096x2304@60Hz" "" ...
$ Max_Resolution_eDP_Integrated_Flat_Panel: chr  "3840x2160@60Hz" "4096x2304@60Hz"
↪  "4096x2304@60Hz" "" ...
$ DirectX_Support                          : chr  "12" "12" "12" "" ...
$ OpenGL_Support                           : logi  NA NA NA NA NA NA ...
$ PCI_Express_Revision                     : chr  "3" "3" "3" "2" ...
$ PCI_Express_Configurations_              : chr  "1x4, 2x2, 1x2+2x1 and 4x1" "1x4, 2x2, 1x2+2x1
↪  and 4x1" "1x4, 2x2, 1x2+2x1 and 4x1" "" ...
$ Max_nb_of_PCI_Express_Lanes              : int  10 12 12 40 10 12 4 4 NA NA ...
$ T                                        : chr  "100Â°C" "100Â°C" "100Â°C" "66.8Â°C" ...
$ Intel_Hyper_Threading_Technology_        : chr  "Yes" "Yes" "Yes" "Yes" ...
$ Intel_Virtualization_Technology_VTx_     : chr  "Yes" "Yes" "Yes" "Yes" ...
$ Intel_64_                                : chr  "Yes" "Yes" "Yes" "Yes" ...
$ Instruction_Set                          : chr  "64-bit" "64-bit" "64-bit" "64-bit" ...
$ Instruction_Set_Extensions               : chr  "SSE4.1/4.2, AVX 2.0" "SSE4.1/4.2, AVX 2.0"
↪  "SSE4.1/4.2, AVX 2.0" "SSE4.2, AVX, AES" ...
$ Idle_States                              : chr  "Yes" "Yes" "Yes" "Yes" ...
$ Thermal_Monitoring_Technologies          : chr  "Yes" "Yes" "Yes" "Yes" ...
$ Secure_Key                               : chr  "Yes" "Yes" "Yes" "" ...
$ Execute_Disable_Bit                      : chr  "Yes" "Yes" "Yes" "Yes" ...
```

2.1.2 Import libraries

```
install.packages("dplyr", type = "source")
install.packages("stringr", type = "source")
install.packages("GGally", type = "source")
install.packages("corrplot", type = "source")
install.packages("caTools", type = "source")
install.packages("MASS", type = "source")
install.packages("car", type = "source")
install.packages("e1071", type = "source")
install.packages("nortest", type = "source")
install.packages("lifecycle")
```

```
# Import libraries
library (dplyr)
library (stringr)
library (GGally)
library (corrplot)
library (caTools)
library (MASS) # for using Stepwise model
library (car)
library (e1071) # for using ANOVA
library (nortest)
library (ggplot2)
```

2.1.3 Explanation of the dataset

This dataset provides comprehensive information about various Central Processing Units (CPUs) utilized in modern computing devices. Each entry details essential specifications and features of different CPU models, encompassing their generation, segment, and specific processor numbers. As computing technology continues to advance, CPUs play a pivotal role in catering to diverse computing needs, resulting in a wide array of CPU configurations. This dataset encapsulates key statistical insights into these CPUs, offering valuable data for analysis and comparison.

2.2 Data cleaning

2.2.1 Removing unused data

Initially, we check the missing values by using is.na(CPU_data) which returns a logical matrix of the same size as your dataset data, where each element is TRUE if the corresponding element in data is NA and FALSE otherwise.

Additionally, we utilize the any() function to determine whether any element in the logical matrix is TRUE. If at least one of the elements is TRUE, any() returns TRUE; otherwise, it returns false.

```
any(is.na(CPU_data))

# Check which columns have missing values
columns_with_missing <- colSums(is.na(CPU_data)) > 0

# Display column names with missing values
names(columns_with_missing)[columns_with_missing]
```

Therefore, there is the fact that the dataset has missing values since the output is TRUE:

```
[1] TRUE
```

The output list of column with missing values:

```
[1] "nb_of_Threads"              "Max_nb_of_Memory_Channels"   "Processor_Graphics_"
[4] "Support_4k"                 "OpenGL_Support"              "Max_nb_of_PCI_Express_Lanes"
```

After that, calculate the number of missing values in each column.

```
# Calculating the number of missing values in each column
# '2' specifies the column
# 'sum' indicates the using function
h <- apply(is.na(CPU_data),2,sum)
print(h)
```

Result:

```
                  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
```

Showing the columns that are missing in all rows:

```
# Trim whitespace from column names
colnames(CPU_data) <- trimws(colnames(CPU_data))
```

```
# Calculate the sum of missing values for each column
missing_counts <- apply(is.na(CPU_data), 2, sum)

# Identify columns with missing values in all rows
columns_with_missing_all <- names(missing_counts)[missing_counts == nrow(CPU_data)]

# Display column names with missing values in all rows
cat(columns_with_missing_all, sep="\n")
```

Result:

```
Processor_Graphics_
Support_4k
OpenGL_Support
```

As we can see, values of **Processor_Graphics, Support_4k** and **OpenGL_Support** are missing in all rows; therefore, we remove columns of these variables.

Moreover, the columns **Product_Collection, Vertical_Segment, Processor_Number, Status, Launch_Date, Cache, Bus_Speed, Conflict_Free, Memory_Types, Processor_Graphics_ , Graphics_Output, Support_4k, Max_Resolution_HDMI, Max_Resolution_DP, Max_Resolution_eDP_Integrated_Flat_Panel, DirectX_Support, OpenGL_Support, PCI_Express_Revision, PCI_Express_Configurations_, Instruction_Set_Extensions** to analyze, so we remove those columns.

```
# Deleting the columns that are not necessary in analyzing the dataset (including the three
# above fields)
# Define a vector of column names to remove

columns_to_remove <- c("Product_Collection", "Vertical_Segment"
                       ,"Processor_Number","Status","Launch_Date","Cache"
                       ,"Bus_Speed","Conflict_Free","Memory_Types"
                       ,"Processor_Graphics_","Graphics_Output"
                       ,"Support_4k","Max_Resolution_HDMI"
                       ,"Max_Resolution_DP","Max_Resolution_eDP_Integrated_Flat_Panel"
                       ,"DirectX_Support","OpenGL_Support"
                       ,"PCI_Express_Revision","PCI_Express_Configurations_"
                       ,"Instruction_Set_Extensions")

# Remove specified columns from the dataset
CPU_data <- CPU_data[, !(names(CPU_data) %in% columns_to_remove)]
# After deleting
head(CPU_data)
```

After deleting, we got result below:

```
  Lithography Recommended_Customer_Price nb_of_Cores nb_of_Threads Processor_Base_Frequency
  ↪  Max_Turbo_Frequency
1       14 nm                   $393.00            2             4                 1.30 GHz
↪  3.60 GHz
2       14 nm                   $297.00            4             8                 1.60 GHz
↪  3.40 GHz
3       14 nm                   $409.00            4             8                 1.80 GHz
↪  4.00 GHz
4       32 nm                   $305.00            4             8                 3.60 GHz
↪  3.80 GHz
5       14 nm                   $281.00            2             4                 1.20 GHz
↪  3.30 GHz
6       14 nm                   $107.00            2             2                 1.50 GHz
```

```
   TDP Embedded_Options_Available Max_Memory_Size Max_nb_of_Memory_Channels
   ↪  Max_Memory_Bandwidth
1 4.5 W                         No           16 GB                         2            29.8
↪  GB/s
2  15 W                         No           32 GB                         2            34.1
↪  GB/s
3  15 W                         No           32 GB                         2            34.1
↪  GB/s
4 130 W                         No        64.23 GB                         4            51.2
↪  GB/s
5 4.5 W                         No           16 GB                         2            29.8
↪  GB/s
6  15 W                         No           16 GB                         2            25.6
↪  GB/s
  ECC_Memory_Supported Graphics_Base_Frequency Graphics_Max_Dynamic_Frequency
  ↪  Graphics_Video_Max_Memory
1                   No                 300 MHz                       1.05 GHz
↪  16 GB
2                   No                 300 MHz                       1.10 GHz
↪  32 GB
3                   No                 300 MHz                       1.15 GHz
↪  32 GB
4                   No
5                   No                 300 MHz                        950 MHz
↪  16 GB
6                                      100 MHz                        800 MHz
  Max_nb_of_PCI_Express_Lanes       T Intel_Hyper_Threading_Technology_
  ↪  Intel_Virtualization_Technology_VTx_
1                          10  100Â°C                                Yes
↪  Yes
2                          12  100Â°C                                Yes
↪  Yes
3                          12  100Â°C                                Yes
↪  Yes
4                          40 66.8Â°C                                Yes
↪  Yes
5                          10  100Â°C                                Yes
↪  Yes
6                          12  105Â°C                                 No
↪  Yes
  Intel_64_ Instruction_Set Idle_States Thermal_Monitoring_Technologies Secure_Key
  ↪  Execute_Disable_Bit
1       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
2       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
3       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
4       Yes          64-bit         Yes                             Yes
↪  Yes
5       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
6       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
```

2.2.2 Filling missing values

We convert space and escape sequence to NA value:

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

Result after converted:

```
  Lithography Recommended_Customer_Price nb_of_Cores nb_of_Threads Processor_Base_Frequency
  ↪  Max_Turbo_Frequency
1       14 nm                   $393.00            2             4                 1.30 GHz
↪  3.60 GHz
2       14 nm                   $297.00            4             8                 1.60 GHz
↪  3.40 GHz
3       14 nm                   $409.00            4             8                 1.80 GHz
↪  4.00 GHz
4       32 nm                   $305.00            4             8                 3.60 GHz
↪  3.80 GHz
5       14 nm                   $281.00            2             4                 1.20 GHz
↪  3.30 GHz
6       14 nm                   $107.00            2             2                 1.50 GHz
↪  <NA>
    TDP Embedded_Options_Available Max_Memory_Size Max_nb_of_Memory_Channels
    ↪  Max_Memory_Bandwidth
1 4.5 W                         No           16 GB                         2            29.8
↪  GB/s
2  15 W                         No           32 GB                         2            34.1
↪  GB/s
3  15 W                         No           32 GB                         2            34.1
↪  GB/s
4 130 W                         No        64.23 GB                         4            51.2
↪  GB/s
5 4.5 W                         No           16 GB                         2            29.8
↪  GB/s
6  15 W                         No           16 GB                         2            25.6
↪  GB/s
  ECC_Memory_Supported Graphics_Base_Frequency Graphics_Max_Dynamic_Frequency
  ↪  Graphics_Video_Max_Memory
1                   No                300 MHz                       1.05 GHz
↪  16 GB
2                   No                300 MHz                       1.10 GHz
↪  32 GB
3                   No                300 MHz                       1.15 GHz
↪  32 GB
4                   No                   <NA>                          <NA>
↪  <NA>
5                   No                300 MHz                        950 MHz
↪  16 GB
6                 <NA>                100 MHz                        800 MHz
↪  <NA>
  Max_nb_of_PCI_Express_Lanes       T Intel_Hyper_Threading_Technology_
  ↪  Intel_Virtualization_Technology_VTx_
1                          10  100Â°C                                Yes
↪  Yes
2                          12  100Â°C                                Yes
↪  Yes
3                          12  100Â°C                                Yes
↪  Yes
4                          40 66.8Â°C                                Yes
↪  Yes
5                          10  100Â°C                                Yes
↪  Yes
6                          12  105Â°C                                 No
↪  Yes
```

```
  Intel_64_ Instruction_Set Idle_States Thermal_Monitoring_Technologies Secure_Key
  ↪  Execute_Disable_Bit
1       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
2       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
3       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
4       Yes          64-bit         Yes                             Yes       <NA>
↪  Yes
5       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
6       Yes          64-bit         Yes                             Yes        Yes
↪  Yes
```

Following that, we remove unit of some numerical variables and convert them into 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))


# Check for NAs after conversion
any(is.na(CPU_data))
head(CPU_data)
```

Result:

```
[1] TRUE
```

```
 Lithography Recommended_Customer_Price nb_of_Cores nb_of_Threads Processor_Base_Frequency
 ↪  Max_Turbo_Frequency
1          14                  $393.00            2             4                 1.30 GHz
↪  3.6
2          14                  $297.00            4             8                 1.60 GHz
↪  3.4
3          14                  $409.00            4             8                 1.80 GHz
↪  4.0
4          32                  $305.00            4             8                 3.60 GHz
↪  3.8
5          14                  $281.00            2             4                 1.20 GHz
↪  3.3
6          14                  $107.00            2             2                 1.50 GHz
↪  NA
```

```
    TDP Embedded_Options_Available Max_Memory_Size Max_nb_of_Memory_Channels
    ↪  Max_Memory_Bandwidth
1   4.5                         No           16.00                        2
↪  29.8
2  15.0                         No           32.00                        2
↪  34.1
3  15.0                         No           32.00                        2
↪  34.1
4 130.0                         No           64.23                        4
↪  51.2
5   4.5                         No           16.00                        2
↪  29.8
6  15.0                         No           16.00                        2
↪  25.6
  ECC_Memory_Supported Graphics_Base_Frequency Graphics_Max_Dynamic_Frequency
  ↪  Graphics_Video_Max_Memory
1                   No                    300                       1.05 GHz
↪  16
2                   No                    300                       1.10 GHz
↪  32
3                   No                    300                       1.15 GHz
↪  32
4                   No                     NA                           <NA>
↪  NA
5                   No                    300                        950 MHz
↪  16
6                 <NA>                    100                        800 MHz
↪  NA
  Max_nb_of_PCI_Express_Lanes  T Intel_Hyper_Threading_Technology_
  ↪  Intel_Virtualization_Technology_VTx_ Intel_64_
1                          10 NA                                 Yes
↪  Yes       Yes
2                          12 NA                                 Yes
↪  Yes       Yes
3                          12 NA                                 Yes
↪  Yes       Yes
4                          40 NA                                 Yes
↪  Yes       Yes
5                          10 NA                                 Yes
↪  Yes       Yes
6                          12 NA                                  No
↪  Yes       Yes
  Instruction_Set Idle_States Thermal_Monitoring_Technologies Secure_Key Execute_Disable_Bit
1          64-bit         Yes                             Yes        Yes                 Yes
2          64-bit         Yes                             Yes        Yes                 Yes
3          64-bit         Yes                             Yes        Yes                 Yes
4          64-bit         Yes                             Yes       <NA>                 Yes
5          64-bit         Yes                             Yes        Yes                 Yes
6          64-bit         Yes                             Yes        Yes                 Yes
```

After printing the result, we can see that there are some numerical variables which are in **Graphics_Max_Dynamic_Frequency** and **Processor_Base_Frequency** that need to be converted unit before removing units and converting to numeric form.

```
 # Graphics_Max_Dynamic_Frequency
# Subset Data by GHz Frequency
subset_GHz <- CPU_data[grepl("GHz", CPU_data$Graphics_Max_Dynamic_Frequency, ignore.case =
↪  TRUE), ]

# Remove GHz Frequency Rows
```

```r
CPU_data <- CPU_data[!grepl("GHz", CPU_data$Graphics_Max_Dynamic_Frequency, ignore.case = TRUE),
↪  ]

# Remove "GHz" from Subset_GHz Column
subset_GHz$Graphics_Max_Dynamic_Frequency <- gsub("GHz", "",
↪  subset_GHz$Graphics_Max_Dynamic_Frequency, fixed = TRUE)

# Convert Subset_GHz Column to Numeric
subset_GHz$Graphics_Max_Dynamic_Frequency <-
↪  as.numeric(subset_GHz$Graphics_Max_Dynamic_Frequency)

# Convert Subset_GHz Frequency to MHz
subset_GHz$Graphics_Max_Dynamic_Frequency <- subset_GHz$Graphics_Max_Dynamic_Frequency * (1000)

# Remove "MHz" from CPU_data Column
CPU_data$Graphics_Max_Dynamic_Frequency <- gsub("MHz", "",
↪  CPU_data$Graphics_Max_Dynamic_Frequency, fixed = TRUE)

# Convert CPU_data Column to Numeric
CPU_data$Graphics_Max_Dynamic_Frequency <- as.numeric(CPU_data$Graphics_Max_Dynamic_Frequency)

# Merge Data
CPU_data <- bind_rows(CPU_data, subset_GHz)

# Processor_Base_Frequency

# Create a subset of data containing only rows where Processor_Base_Frequency column contains
↪  "GHz"
subset_GHz <- CPU_data[grepl("GHz", CPU_data$Processor_Base_Frequency, ignore.case = TRUE), ]

# Remove rows from the original dataframe where Processor_Base_Frequency contains "GHz"
CPU_data <- CPU_data[!grepl("GHz", CPU_data$Processor_Base_Frequency, ignore.case = TRUE), ]

# Remove "GHz" from the Processor_Base_Frequency column in the subset dataframe
subset_GHz$Processor_Base_Frequency <- gsub("GHz", "", subset_GHz$Processor_Base_Frequency,
↪  fixed = TRUE)

# Convert Processor_Base_Frequency column in the subset dataframe to numeric
subset_GHz$Processor_Base_Frequency <- as.numeric(subset_GHz$Processor_Base_Frequency)

# Convert Processor_Base_Frequency from GHz to MHz by multiplying by 1000
subset_GHz$Processor_Base_Frequency <- subset_GHz$Processor_Base_Frequency * (1000)

# Remove " MHz " from the Processor_Base_Frequency column in the original dataframe
CPU_data$Processor_Base_Frequency <- gsub("MHz", "", CPU_data$Processor_Base_Frequency, fixed =
↪  TRUE)

# Convert Processor_Base_Frequency column in the original dataframe to numeric
CPU_data$Processor_Base_Frequency <- as.numeric(CPU_data$Processor_Base_Frequency)

# Merge the subset dataframe with the original dataframe
CPU_data <- bind_rows(CPU_data, subset_GHz)
```

Filling missing values for numerical columns by their means.

```r
 # Filling missing values for specified columns
columns_to_fill_mean <- c("Lithography", "nb_of_Cores", "nb_of_Threads", "Max_Turbo_Frequency",
                          "TDP", "Max_Memory_Size", "Max_nb_of_Memory_Channels",
                          "Max_Memory_Bandwidth", "Graphics_Base_Frequency",
                          "Graphics_Video_Max_Memory", "Max_nb_of_PCI_Express_Lanes",
                          "T", "Graphics_Max_Dynamic_Frequency", "Processor_Base_Frequency")
```

```r
# Loop through each column
for (col in columns_to_fill_mean) {
  # Calculate the mean of the column, excluding missing values
  mean_value <- mean(CPU_data[[col]], na.rm = TRUE)

  # Replace missing values with the mean
  CPU_data[[col]][is.na(CPU_data[[col]])] <- mean_value
}

# Convert Recommended_Customer_Price column to numeric
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)
```

```r
 # Check the column names in CPU_data
colnames(CPU_data)
```

Result:

```
 [1] "Lithography"                          "Recommended_Customer_Price"
 [3] "nb_of_Cores"                          "nb_of_Threads"
 [5] "Processor_Base_Frequency"             "Max_Turbo_Frequency"
 [7] "TDP"                                  "Embedded_Options_Available"
 [9] "Max_Memory_Size"                      "Max_nb_of_Memory_Channels"
[11] "Max_Memory_Bandwidth"                 "ECC_Memory_Supported"
[13] "Graphics_Base_Frequency"              "Graphics_Max_Dynamic_Frequency"
[15] "Graphics_Video_Max_Memory"            "Max_nb_of_PCI_Express_Lanes"
[17] "T"                                    "Intel_Hyper_Threading_Technology_"
[19] "Intel_Virtualization_Technology_VTx_" "Intel_64_"
[21] "Instruction_Set"                      "Idle_States"
[23] "Thermal_Monitoring_Technologies"      "Secure_Key"
[25] "Execute_Disable_Bit"
```

Filling missing values for categorical columns by their modes.

```r
 # Define a function to fill missing values with mode
fillmode <- function(column) {
  # Calculate mode value
  mode_value <- names(sort(table(column), decreasing = TRUE))[1]

  # Replace missing values with mode value
  column[is.na(column)] <- mode_value

  # Return the modified column
  return(column)
}

# Specify the columns to fill missing values for
columns_to_fill_2 <- 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")

# Apply the fillmode function to specified columns
CPU_data[columns_to_fill_2] <- lapply(CPU_data[columns_to_fill_2], fillmode)
```

Re-checking missing values and print the result horizontally:

```
# Re-check
any(is.na(CPU_data))

missing_values <- apply(is.na(CPU_data), 2, sum)

# Print the result horizontally
print(missing_values)
```

Result:

```
[1] TRUE
```

```
                          Lithography           Recommended_Customer_Price
                                    0                                 1473
                          nb_of_Cores                       nb_of_Threads
                                    0                                    0
            Processor_Base_Frequency                  Max_Turbo_Frequency
                                    0                                    0
                                  TDP            Embedded_Options_Available
                                    0                                    0
                    Max_Memory_Size             Max_nb_of_Memory_Channels
                                    0                                    0
                Max_Memory_Bandwidth                  ECC_Memory_Supported
                                    0                                    0
              Graphics_Base_Frequency       Graphics_Max_Dynamic_Frequency
                                    0                                    0
           Graphics_Video_Max_Memory          Max_nb_of_PCI_Express_Lanes
                                    0                                    0
                                    T     Intel_Hyper_Threading_Technology_
                                    0                                    0
Intel_Virtualization_Technology_VTx_                              Intel_64_
                                    0                                    0
                       Instruction_Set                         Idle_States
                                    0                                    0
     Thermal_Monitoring_Technologies                          Secure_Key
                                    0                                    0
                 Execute_Disable_Bit
                                    0
```

After that, we transform categorical variables to numerical variables:

```
# Transform categorical variables to numerical variables

# Convert "Embedded_Options_Available" to numerical (1 for "Yes", 0 for "No")
CPU_data$Embedded_Options_Available <- ifelse(CPU_data$Embedded_Options_Available == "Yes", 1,
↪  0)

# Convert "ECC_Memory_Supported" to numerical (1 for "Yes", 0 for "No")
CPU_data$ECC_Memory_Supported <- ifelse(CPU_data$ECC_Memory_Supported == "Yes", 1, 0)

# Convert "Intel_Hyper_Threading_Technology_" to numerical (1 for "Yes", 0 for "No")
CPU_data$Intel_Hyper_Threading_Technology_ <- ifelse(CPU_data$Intel_Hyper_Threading_Technology
↪  == "Yes", 1, 0)

# Convert "Intel_Virtualization_Technology_VTx_" to numerical (1 for "Yes", 0 for "No")
CPU_data$Intel_Virtualization_Technology_VTx_ <-
↪  ifelse(CPU_data$Intel_Virtualization_Technology_VTx == "Yes", 1, 0)
```

```r
# Convert "Intel_64_" to numerical (1 for "Yes", 0 for "No")
CPU_data$Intel_64_ <- ifelse(CPU_data$Intel_64_ == "Yes", 1, 0)

# Convert "Idle_States" to numerical (1 for "Yes", 0 for "No")
CPU_data$Idle_States <- ifelse(CPU_data$Idle_States == "Yes", 1, 0)

# Convert "Thermal_Monitoring_Technologies" to numerical (1 for "Yes", 0 for "No")
CPU_data$Thermal_Monitoring_Technologies <- ifelse(CPU_data$Thermal_Monitoring_Technologies ==
↪  "Yes", 1, 0)

# Convert "Secure_Key" to numerical (1 for "Yes", 0 for "No")
CPU_data$Secure_Key <- ifelse(CPU_data$Secure_Key == "Yes", 1, 0)

# Convert "Execute_Disable_Bit" to numerical (1 for "Yes", 0 for "No")
CPU_data$Execute_Disable_Bit <- ifelse(CPU_data$Execute_Disable_Bit == "Yes", 1, 0)

# Convert "Instruction_Set" to numerical (0 for "32-bit", 1 for "64-bit", 2 for "Itanium")
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 processing, we re-check and print the result horizontally.

```r
# Re-check
any(is.na(CPU_data))

missing_values <- apply(is.na(CPU_data), 2, sum)

# Print the result horizontally
print(missing_values)
```

Result:

```
[1] TRUE
```

```
                        Lithography          Recommended_Customer_Price
                                  0                                1473
                       nb_of_Cores                       nb_of_Threads
                                  0                                   0
          Processor_Base_Frequency                  Max_Turbo_Frequency
                                  0                                   0
                                TDP            Embedded_Options_Available
                                  0                                   0
                    Max_Memory_Size             Max_nb_of_Memory_Channels
                                  0                                   0
                Max_Memory_Bandwidth                  ECC_Memory_Supported
                                  0                                   0
             Graphics_Base_Frequency        Graphics_Max_Dynamic_Frequency
                                  0                                   0
          Graphics_Video_Max_Memory           Max_nb_of_PCI_Express_Lanes
                                  0                                   0
                                  T      Intel_Hyper_Threading_Technology_
                                  0                                   0
Intel_Virtualization_Technology_VTx_                            Intel_64_
                                  0                                   0
                     Instruction_Set                          Idle_States
```

```
                                0                                 0
Thermal_Monitoring_Technologies                        Secure_Key
                                0                                 0
            Execute_Disable_Bit
                                0
```

The only variable with missing data after the pre-processing stage is Recommended Customer Price, since the goal of this report is to study and forecast the recommended customer price of CPU.

2.3 Data visualization

2.3.1 Descriptive statistics for each variable

We partition the CPU data into two subsets: CPU train and CPU test. The latter is the set that is devoid of all missing values, including those in the Recommended Customer Price variable.

```r
# Create CPU_train dataset without missing values in Recommended_Customer_Price
CPU_train <- CPU_data[complete.cases(CPU_data$Recommended_Customer_Price), ]

# Create CPU_test dataset with missing values in Recommended_Customer_Price
CPU_test <- CPU_data[!complete.cases(CPU_data$Recommended_Customer_Price), ]

# Summary
summary(CPU_train)
```

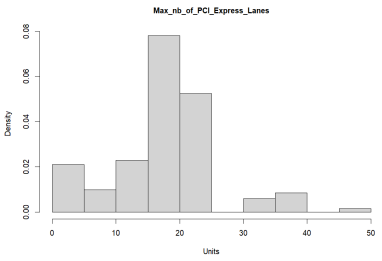
```
Lithography     Recommended_Customer_Price  nb_of_Cores     nb_of_Threads
 Min.   : 14.00   Min.   :  2.54             Min.   : 1.000   Min.   : 1.00
 1st Qu.: 14.00   1st Qu.:107.00             1st Qu.: 2.000   1st Qu.: 4.00
 Median : 22.00   Median :239.50             Median : 2.000   Median : 4.00
 Mean   : 26.57   Mean   :268.37             Mean   : 3.148   Mean   : 5.77
 3rd Qu.: 32.00   3rd Qu.:378.00             3rd Qu.: 4.000   3rd Qu.: 8.00
 Max.   :130.00   Max.   :999.00             Max.   :16.000   Max.   :24.00
 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_nb_of_Memory_Channels
 Min.   :0.0000             Min.   :  1.00   Min.   :1.000
 1st Qu.:0.0000             1st Qu.: 16.00   1st Qu.:2.000
 Median :0.0000             Median : 32.00   Median :2.000
 Mean   :0.3235             Mean   : 78.03   Mean   :2.167
 3rd Qu.:1.0000             3rd Qu.:128.00   3rd Qu.:2.000
 Max.   :1.0000             Max.   :768.00   Max.   :6.000
 Max_Memory_Bandwidth ECC_Memory_Supported Graphics_Base_Frequency
 Min.   : 1.60        Min.   :0.0000       Min.   :100.0
 1st Qu.:25.60        1st Qu.:0.0000       1st Qu.:327.5
 Median :29.80        Median :0.0000       Median :420.1
 Mean   :29.91        Mean   :0.3914       Mean   :405.9
 3rd Qu.:35.08        3rd Qu.:1.0000       3rd Qu.:420.1
 Max.   :85.30        Max.   :1.0000       Max.   :900.0
 Graphics_Max_Dynamic_Frequency Graphics_Video_Max_Memory
 Min.   : 500                   Min.   : 1.00
 1st Qu.:1000                   1st Qu.:22.81
```

```
Median :1032                  Median :22.81
Mean   :1031                  Mean   :22.54
3rd Qu.:1050                  3rd Qu.:22.81
Max.   :1350                  Max.   :64.00
Max_nb_of_PCI_Express_Lanes       T          Intel_Hyper_Threading_Technology_
Min.   : 1.00               Min.   : 64.00   Min.   :0.000
1st Qu.:12.00               1st Qu.: 83.22   1st Qu.:0.000
Median :16.00               Median : 83.22   Median :1.000
Mean   :16.79               Mean   : 83.24   Mean   :0.621
3rd Qu.:20.40               3rd Qu.: 83.22   3rd Qu.:1.000
Max.   :48.00               Max.   :105.00   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
```

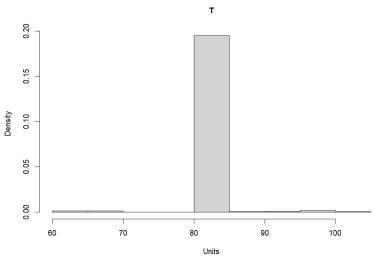
2.3.2 Plotting graph

2.3.2.a Histogram

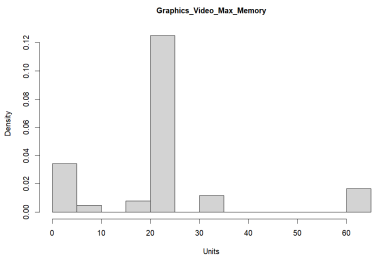
```
his = c("Lithography", "Recommended_Customer_Price", "nb_of_Cores", "nb_of_Threads",
"Processor_Base_Frequency", "Max_Turbo_Frequency", "TDP", "Max_Memory_Size",
"Max_nb_of_Memory_Channels", "Max_Memory_Bandwidth", "Graphics_Base_Frequency",
"Graphics_Max_Dynamic_Frequency", "Graphics_Video_Max_Memory", "Max_nb_of_PCI_Express_Lanes", "T")

for (var in his) {
  hist(CPU_train[[var]], main = var, xlab =" Units ", freq = FALSE, cex.main = 1)
}
```



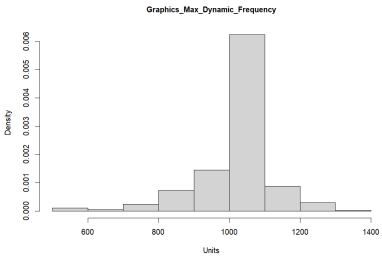
Left-skewded Distribution



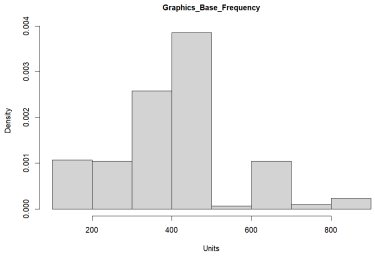
Gather around a point



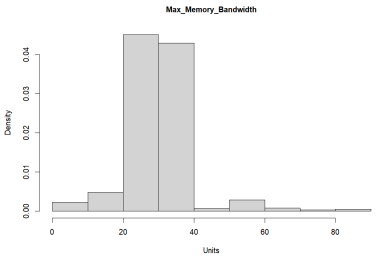
Exist outliers



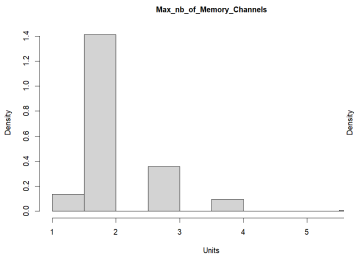
Symmetric distribution



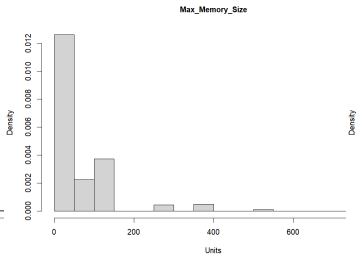
Gather around a point, not symmetric



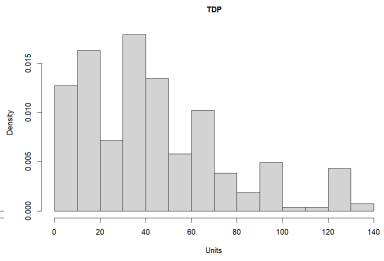
Exist outliers



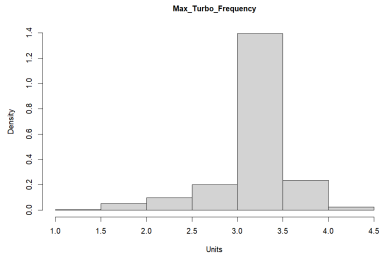
Right-skewded Distribution



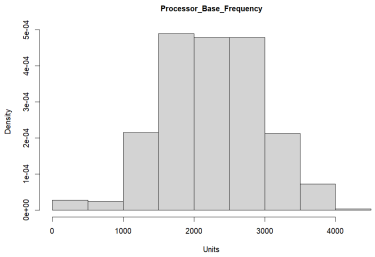
Right-skewded Distribution



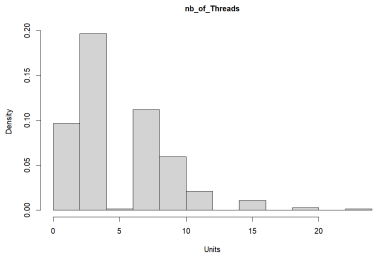
Right-skewded Distribution



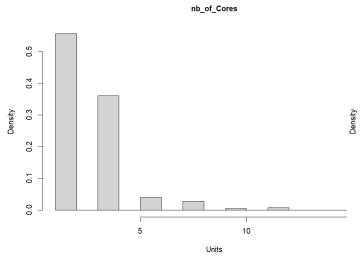
Left-skewded Distribution



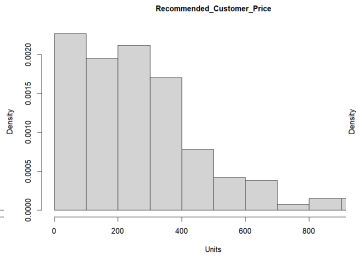
Symmetric distribution



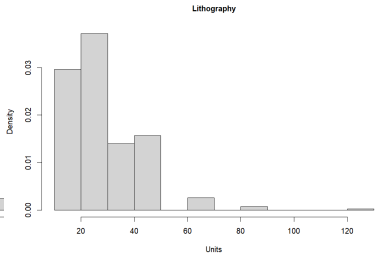
Right-skewded Distribution



Distribution is not good



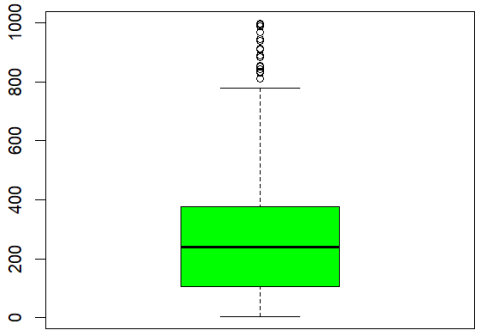
Right-skewded Distribution



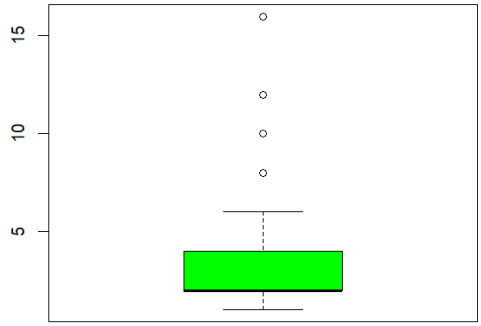
Right-skewded Distribution

2.3.2.b Boxplot

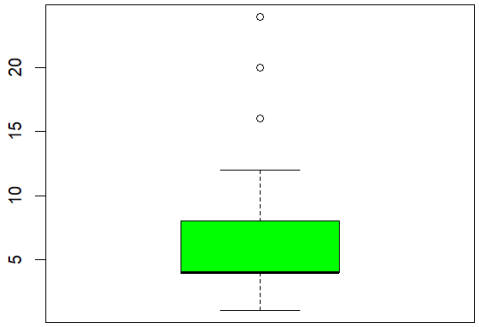
```
box <- colnames(CPU_train)
for (var in box) {
  boxplot(CPU_train[[var]],
  col="green",
  xlab = var,
  cex.lab = 1,
  title.cex = 1,
  border = "brown")
}
```



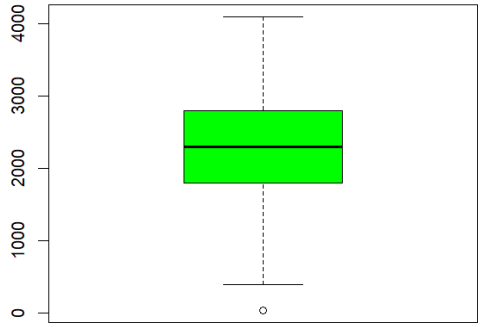
Recommended_Customer_Price



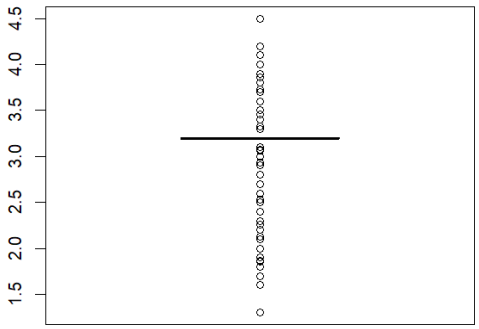
nb_of_Cores



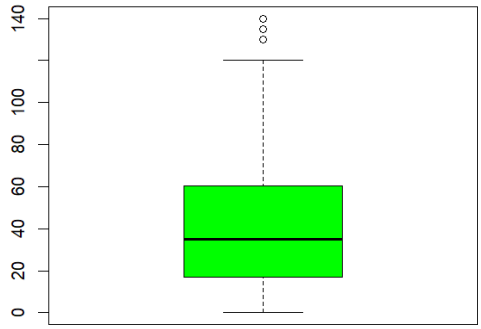
nb_of_Threads



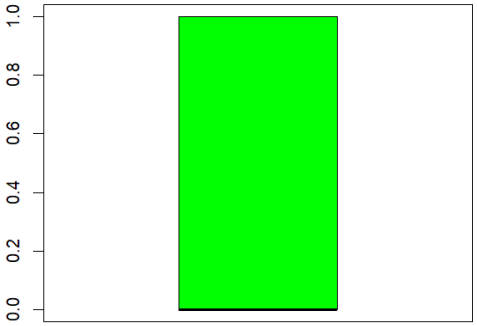
Processor_Base_Frequency



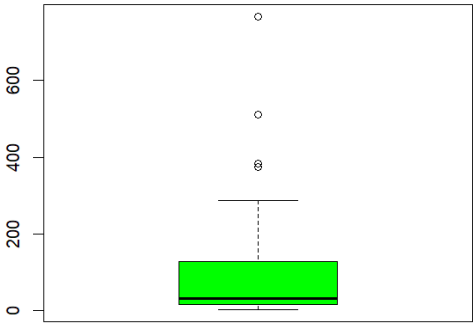
Max_Turbo_Frequency



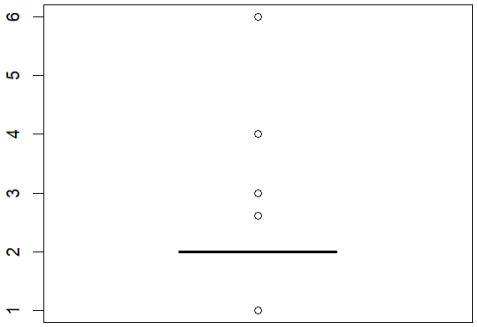
TDP



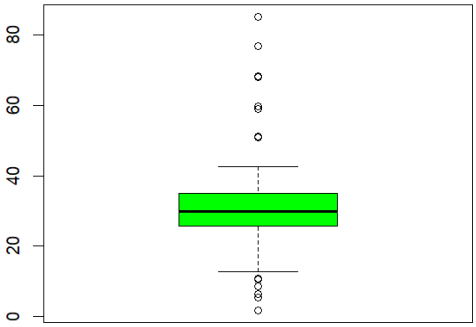
Embedded_Options_Available



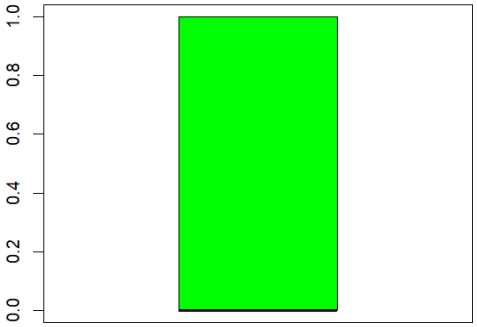
Max_Memory_Size



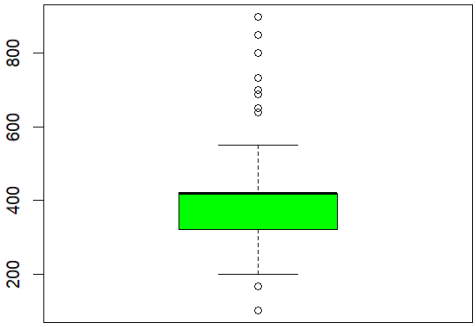
Max_nb_of_Memory_Channels



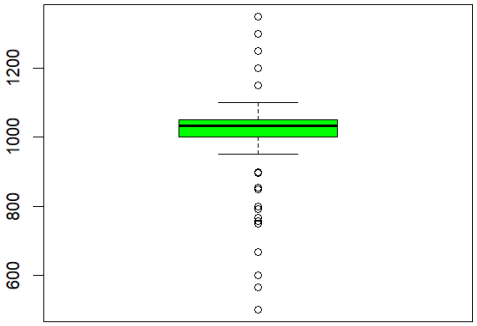
Max_Memory_Bandwidth



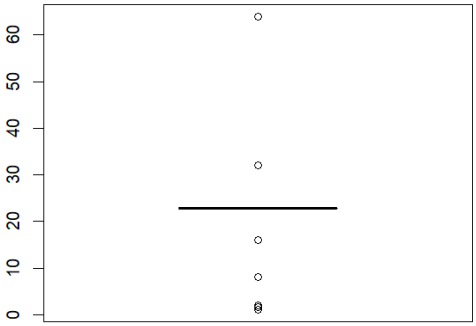
ECC_Memory_Supported



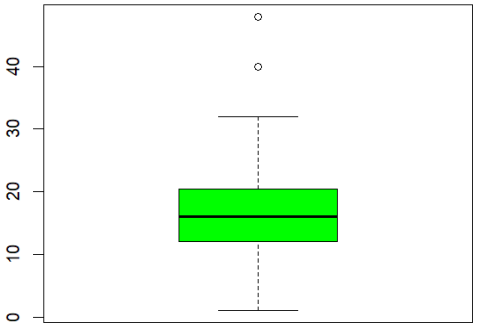
Graphics_Base_Frequency



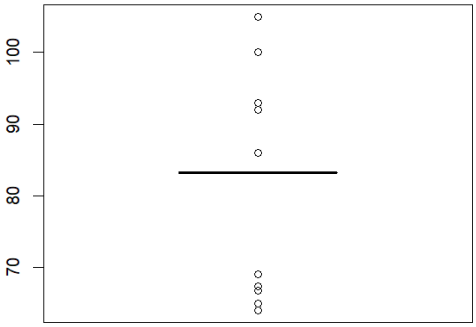
Graphics_Max_Dynamic_Frequency



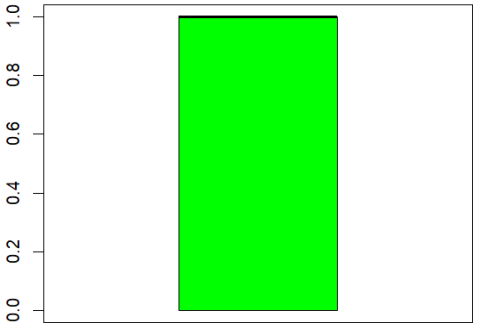
Graphics_Video_Max_Memory



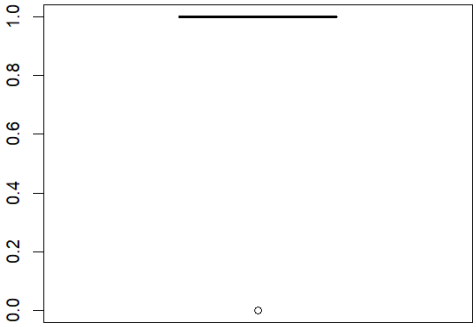
Max_nb_of_PCI_Express_Lanes



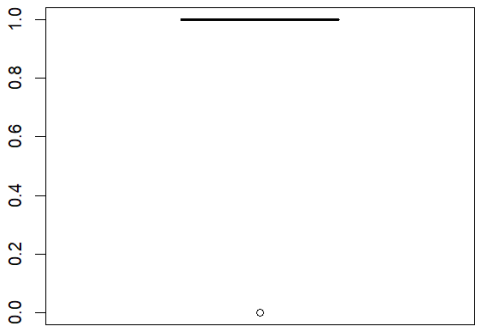
T



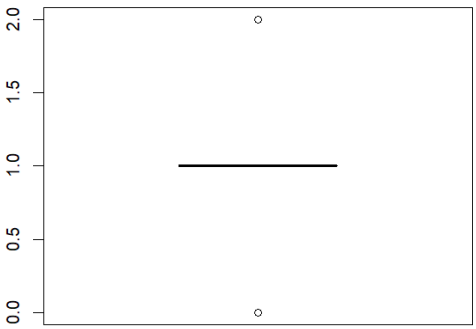
Intel_Hyper_Threading_Technology_



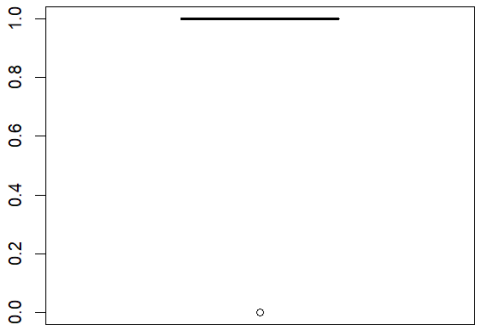
Intel_Virtualization_Technology_VTx_



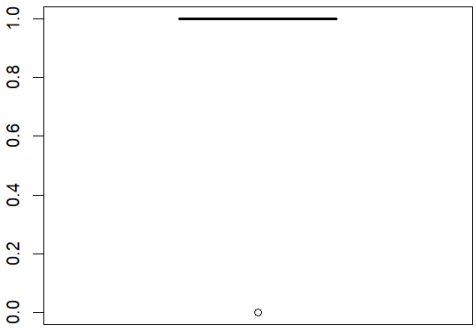
Intel_64_



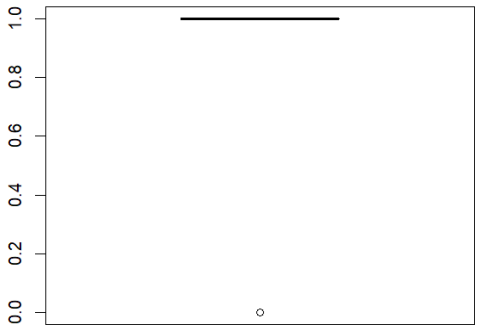
Instruction_Set



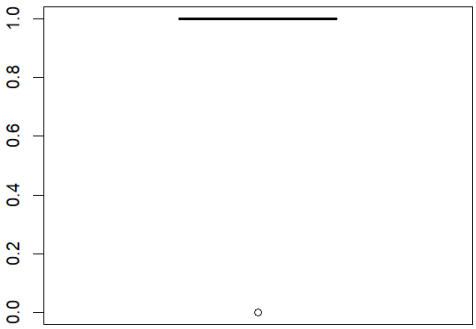
Idle_States



Thermal_Monitoring_Technologies



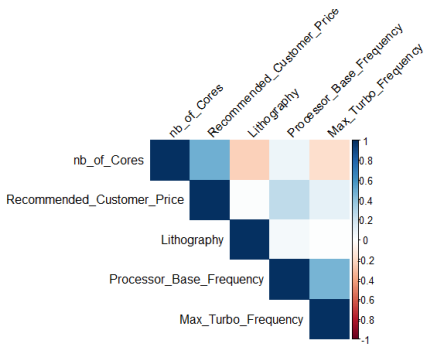
Secure_Key



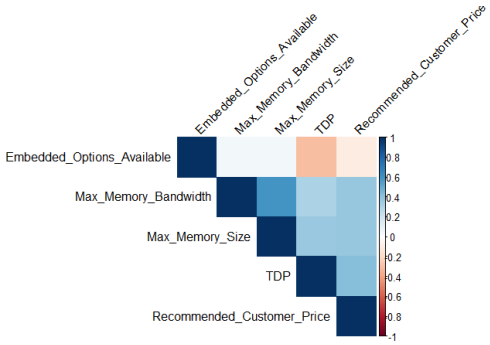
Execute_Disable_Bit

2.3.2.c Correlation matrix

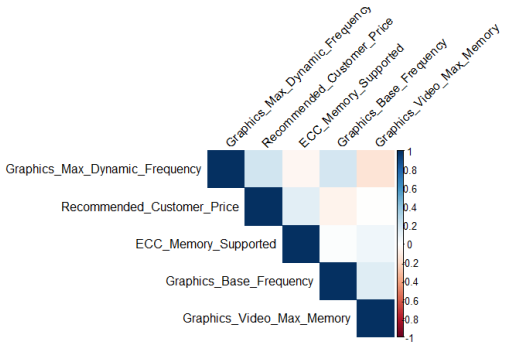
```
# Calculate the correlation matrix
select<-c("Processor_Base_Frequency","Lithography","nb_of_Cores",
"Max_Turbo_Frequency","Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order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



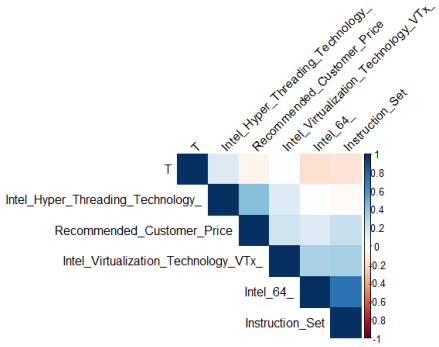

```
select <- c("Max_Memory_Bandwidth", "TDP", "Embedded_Options_Available" ,"Max_Memory_Size"
↪  ,"Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot (correlation_matrix, method = "color", type = "upper", order = "hclust", tl.col =
↪  "black",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



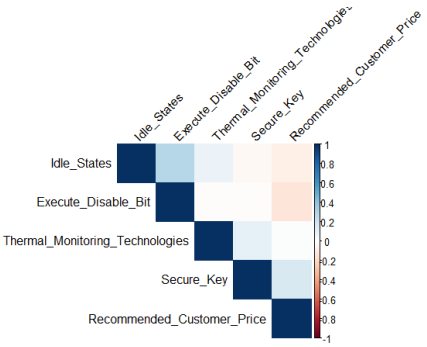
```r
# Calculate the correlation matrix
select <- c("ECC_Memory_Supported", "Graphics_Base_Frequency", "Graphics_Max_Dynamic_Frequency"
↪  ,"Graphics_Video_Max_Memory" ,"Recommended_Customer_Price")
correlation_matrix <- cor(CPU_train[select])
# Plot the correlation matrix
corrplot(correlation_matrix,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




```r
# 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)
```



```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)
```



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.

```r
# Divide CPU_train into train_df and test_df
# Set the seed for reproducibility
set.seed (42)
# Use 70% of dataset as training set and 30% as test set (randomly)
split <- sample.split(CPU_train , SplitRatio = 0.70)
train_df <- subset(CPU_train, split == TRUE)
test_df <- subset(CPU_train, split == FALSE)
```

For simplicity, we firstly try out some linear models. We use the train set for fitting MLR model:

```r
# MLR
# Fitting MLR model
model_mlr <- lm(Recommended_Customer_Price ~ ., data = train_df)
summary(model_mlr)
```

Result:

```
Call:
lm(formula = Recommended_Customer_Price ~ ., data = train_df)

Residuals:
    Min      1Q  Median      3Q     Max
-337.20  -74.27   -3.34   60.31  650.60

Coefficients:
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                     -435.30912  277.87448  -1.567 0.117816
```

```
Lithography                              1.86791    0.68267   2.736 0.006426 **
nb_of_Cores                             11.59640    6.78604   1.709 0.088065 .
nb_of_Threads                           21.89689    3.90409   5.609 3.29e-08 ***
Processor_Base_Frequency                -0.01873    0.01374  -1.363 0.173450
Max_Turbo_Frequency                     52.55787   17.92416   2.932 0.003512 **
TDP                                      0.63478    0.38478   1.650 0.099596 .
Embedded_Options_Available               7.81178   15.16417   0.515 0.606666
Max_Memory_Size                          0.22419    0.07634   2.937 0.003464 **
Max_nb_of_Memory_Channels               49.77334   18.01912   2.762 0.005941 **
Max_Memory_Bandwidth                    -0.04232    0.84912  -0.050 0.960265
ECC_Memory_Supported                     7.92313   14.40344   0.550 0.582494
Graphics_Base_Frequency                 -0.13440    0.04478  -3.001 0.002816 **
Graphics_Max_Dynamic_Frequency           0.18071    0.06282   2.876 0.004186 **
Graphics_Video_Max_Memory                0.17679    0.40898   0.432 0.665714
Max_nb_of_PCI_Express_Lanes             -4.28271    1.16439  -3.678 0.000259 ***
T                                       -0.37576    2.33211  -0.161 0.872056
Intel_Hyper_Threading_Technology_      142.97575   14.16106  10.096  < 2e-16 ***
Intel_Virtualization_Technology_VTx_   123.00845   30.53342   4.029 6.44e-05 ***
Intel_64_                              -23.53003   74.51332  -0.316 0.752292
Instruction_Set                        158.92154   66.14908   2.402 0.016629 *
Idle_States                            -63.52603   36.37895  -1.746 0.081354 .
Thermal_Monitoring_Technologies          4.27608   27.21798   0.157 0.875222
Secure_Key                             -25.51079   29.03697  -0.879 0.380039
Execute_Disable_Bit                    -78.13094  170.59729  -0.458 0.647152
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 126.2 on 527 degrees of freedom
Multiple R-squared:  0.6438,        Adjusted R-squared:  0.6276
F-statistic: 39.69 on 24 and 527 DF,  p-value: < 2.2e-16
```

Then, we use stepwise regression (SW) to minimize the number of predictors.

```r
# Stepwise regression
# Fitting Stepwise Regression model
model_sw <- stepAIC(lm(Recommended_Customer_Price ~ ., data = train_df), direction = "both")
summary(model_sw)
```

Result:

```
Call:
lm(formula = Recommended_Customer_Price ~ Lithography + nb_of_Cores +
    nb_of_Threads + Max_Turbo_Frequency + Max_Memory_Size + Max_nb_of_Memory_Channels +
    Graphics_Base_Frequency + Graphics_Max_Dynamic_Frequency +
    Max_nb_of_PCI_Express_Lanes + Intel_Hyper_Threading_Technology_ +
    Intel_Virtualization_Technology_VTx_ + Instruction_Set +
    Idle_States + Execute_Disable_Bit, data = train_df)

Residuals:
    Min      1Q  Median      3Q     Max
-329.95  -75.14   -3.72   58.33  643.28

Coefficients:
                                      Estimate Std. Error t value Pr(>|t|)
(Intercept)                         -414.42469  133.30254  -3.109 0.001977 **
Lithography                            2.30075    0.54367   4.232 2.73e-05 ***
nb_of_Cores                           17.48242    5.89614   2.965 0.003161 **
nb_of_Threads                         19.84871    3.56152   5.573 3.97e-08 ***
Max_Turbo_Frequency                   52.47405   15.05196   3.486 0.000530 ***
```

```
Max_Memory_Size                          0.24087    0.07180   3.355 0.000850 ***
Max_nb_of_Memory_Channels               52.48119   16.32026   3.216 0.001380 **
Graphics_Base_Frequency                 -0.11780    0.04132  -2.851 0.004524 **
Graphics_Max_Dynamic_Frequency           0.15180    0.05743   2.643 0.008457 **
Max_nb_of_PCI_Express_Lanes             -3.71062    1.03930  -3.570 0.000389 ***
Intel_Hyper_Threading_Technology_      144.46819   13.42184  10.764  < 2e-16 ***
Intel_Virtualization_Technology_VTx_   118.12884   27.84317   4.243 2.60e-05 ***
Instruction_Set                        125.96946   31.01991   4.061 5.62e-05 ***
Idle_States                            -57.26815   35.22061  -1.626 0.104540
Execute_Disable_Bit                   -161.98204  103.07627  -1.571 0.116661
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 125.7 on 537 degrees of freedom
Multiple R-squared:  0.6401,        Adjusted R-squared:  0.6307
F-statistic: 68.22 on 14 and 537 DF,  p-value: < 2.2e-16
```

2.4.2 Analysis of Variance (ANOVA)

```
# 2-way 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   76858   76858   4.431  0.03575 *
nb_of_Cores                          1 6437494 6437494 371.161  < 2e-16 ***
Max_Turbo_Frequency                  1  993666  993666  57.291 1.65e-13 ***
TDP                                  1  728297  728297  41.991 2.08e-10 ***
Max_Memory_Size                      1  531646  531646  30.653 4.84e-08 ***
Max_Memory_Bandwidth                 1  206694  206694  11.917  0.00060 ***
Graphics_Base_Frequency              1  456508  456508  26.321 4.05e-07 ***
Graphics_Max_Dynamic_Frequency       1  173062  173062   9.978  0.00167 **
Graphics_Video_Max_Memory            1   13349   13349   0.770  0.38071
T                                    1   56776   56776   3.273  0.07097 .
Intel_Hyper_Threading_Technology_    1 3590745 3590745 207.029  < 2e-16 ***
Intel_Virtualization_Technology_VTx_ 1  487859  487859  28.128 1.66e-07 ***
Intel_64_                            1   26198   26198   1.510  0.21961
Instruction_Set                      1  440420  440420  25.393 6.40e-07 ***
Idle_States                          1   62295   62295   3.592  0.05861 .
Residuals                          536 9296495   17344
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Checking whether the distribution is normal by using Shapiro-Wilk normality test:

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

```
        Shapiro-Wilk normality test

data:  av_residual
W = 0.9371, p-value < 2.2e-16
```

This dataset does not have normal distribution since $p - value < 0.05$, it means that there is nonlinear relationships between variables and MLR is not efficient enough. Furthermore, there are also outliers which will affect our predicted models. To handle this case, we should try a further method by using Decision Trees model.

2.4.3 Decision Tree (DT)

```r
# Decision trees
library(rpart)
model_dt <- rpart(Recommended_Customer_Price ~ ., data = train_df)
summary(model_dt)
```

```
Call:
rpart(formula = Recommended_Customer_Price ~ ., data = train_df)
  n= 552

           CP nsplit rel error    xerror       xstd
1  0.32476374      0 1.0000000 1.0047709 0.08147895
2  0.10482911      1 0.6752363 0.6782048 0.05784612
3  0.06479055      2 0.5704072 0.5748831 0.05286371
4  0.05392032      3 0.5056166 0.5247536 0.05352305
5  0.03684444      4 0.4516963 0.4913881 0.05514798
6  0.02927598      5 0.4148518 0.4649467 0.05219677
7  0.02429200      6 0.3855759 0.4288584 0.04840337
8  0.01445573      7 0.3612839 0.4187725 0.04714921
9  0.01373355      8 0.3468281 0.4178921 0.04830719
10 0.01180496      9 0.3330946 0.4002299 0.04706030
11 0.01000000     10 0.3212896 0.3983008 0.04713512


Node number 1: 552 observations,    complexity param=0.3247637
  mean=267.8583, MSE=42714.42
  left son=2 (323 obs) right son=3 (229 obs)
  Primary splits:
      nb_of_Threads                     < 6        to the left,  improve=0.3247637, (0 missing)
      nb_of_Cores                       < 3        to the left,  improve=0.2275234, (0 missing)
      Max_nb_of_Memory_Channels         < 2.807638 to the left,  improve=0.2159864, (0 missing)
      Intel_Hyper_Threading_Technology_ < 0.5      to the left,  improve=0.1898362, (0 missing)
      TDP                               < 37.5     to the left,  improve=0.1692425, (0 missing)
  Surrogate splits:
      nb_of_Cores                < 3        to the left,  agree=0.815, adj=0.555, (0 split)
      Max_nb_of_Memory_Channels  < 2.307638 to the left,  agree=0.781, adj=0.472, (0 split)
      Max_Memory_Size            < 64.115   to the left,  agree=0.779, adj=0.467, (0 split)
      TDP                        < 39       to the left,  agree=0.772, adj=0.450, (0 split)
      Max_nb_of_PCI_Express_Lanes < 18       to the left,  agree=0.726, adj=0.341, (0 split)

Node number 2: 323 observations,    complexity param=0.06479055
  mean=168.6867, MSE=14173.22
  left son=4 (112 obs) right son=5 (211 obs)
  Primary splits:
      nb_of_Threads                     < 3        to the left,  improve=0.3336986, (0 missing)
```

```
      Intel_Hyper_Threading_Technology_ < 0.5      to the left,  improve=0.1965322, (0 missing)
      Max_Memory_Size                   < 8.395    to the left,  improve=0.1717669, (0 missing)
      Graphics_Base_Frequency           < 305.5    to the right, improve=0.1605371, (0 missing)
      Max_Turbo_Frequency               < 3.199223 to the left,  improve=0.1556590, (0 missing)
  Surrogate splits:
      Intel_Hyper_Threading_Technology_ < 0.5      to the left,  agree=0.814, adj=0.464, (0
      ↪  split)
      nb_of_Cores                       < 1.5      to the left,  agree=0.728, adj=0.214, (0
      ↪  split)
      Max_nb_of_Memory_Channels         < 1.5      to the left,  agree=0.700, adj=0.134, (0
      ↪  split)
      TDP                               < 4.25     to the left,  agree=0.690, adj=0.107, (0
      ↪  split)
      Intel_64_                         < 0.5      to the left,  agree=0.690, adj=0.107, (0
      ↪  split)
......
Node number 44: 99 observations
  mean=191.9495, MSE=4969.947

Node number 45: 7 observations
  mean=398.2857, MSE=56989.35
```

From analyzing this output, there are some points that we must consider:

- The difference between "rel error" (training error) and "xerror" (cross-validated error) starts at around 0.0019 and gradually decreases to around 0.0006 as the number of splits increases, indicating that the model's performance on unseen data is not as good as its performance on the training data.
- Some terminal nodes have relatively high MSE values, suggesting that the model might not generalize well to new data.
  - Node 7: MSE=42019.66
  - Node 15: MSE=50561.72
  - Node 30: MSE=46564.53
  - Node 31: MSE=26256.64
  - Node 45: MSE=56989.3
- The model has many splits (11), which might be an indication of overfitting, especially if the dataset is not large enough to support such complexity.

It's a high chance that there's overfitting in the Decision Trees model. Therefore we should proceed with a random forest model to avoid overfitting.

2.4.4 Random Forest (RF)

```r
install.packages("randomForest",type="source")
library(randomForest)
model_rf <- randomForest(Recommended_Customer_Price ~ ., data = train_df)
summary(model_rf)
```

2.4.5 Model Comparison

With p-value greater than 0.05, model_mlr and model_sw have equivalent efficiency. However, R-squared of model_sw is greater than model_mlr 0.6307 ¿ 0.6276, model_sw is a little better than model_mlr. Additionally, we consider the entire 4 models.

```
# Ensure all models are trained on the same dataset
common_training_data <- train_df

# Fit MLR model
model_mlr <- lm(Recommended_Customer_Price ~ ., data = common_training_data)

# Fit Stepwise model (assuming you already have the stepwise model)
model_sw <- step(lm(Recommended_Customer_Price ~ ., data = common_training_data))

# Fit Decision Tree model
model_dt <- rpart(Recommended_Customer_Price ~ ., data = common_training_data)

# Fit Random Forest model
model_rf <- randomForest(Recommended_Customer_Price ~ ., data = common_training_data)

# Predictions on test data
pred_mlr <- predict(model_mlr, newdata = test_df)
pred_sw <- predict(model_sw, newdata = test_df)
pred_dt <- predict(model_dt, newdata = test_df)
pred_rf <- predict(model_rf, newdata = test_df)

# Calculate MSE for each model
mse_mlr <- mean((test_df$Recommended_Customer_Price - pred_mlr)^2)
mse_sw <- mean((test_df$Recommended_Customer_Price - pred_sw)^2)
mse_dt <- mean((test_df$Recommended_Customer_Price - pred_dt)^2)
mse_rf <- mean((test_df$Recommended_Customer_Price - pred_rf)^2)

# Print MSE for each model
cat("MLR MSE:", mse_mlr, "\n")
cat("Stepwise MSE:", mse_sw, "\n")
cat("Decision Tree MSE:", mse_dt, "\n")
cat("Random Forest MSE:", mse_rf, "\n")
```

```
MLR MSE: 15263.96
Stepwise MSE: 15512.17
Decision Tree MSE: 18115.53
Random Forest MSE: 10963.35
```

```
# Create a vector of model names
models <- c("MLR", "SW", "DT", "RF")

# Create a vector of MSE values for each model
mse_values <- c(mse_mlr, mse_sw, mse_dt, mse_rf)

# Set the ylim to include the maximum MSE value plus a buffer for readability
ylim_max <- max(mse_values) * 1.3  # Add a buffer of 10% to the maximum value
barplot(mse_values, names.arg = models, col = "skyblue", main = "Mean Squared Error", ylab =
↪  "MSE", ylim = c(0, ylim_max))
```

```
# Add data labels
text(x = 1:length(models), y = mse_values + 1000, labels = round(mse_values, 2), col = "black",
↪  pos = 3, cex = 1.2)
```

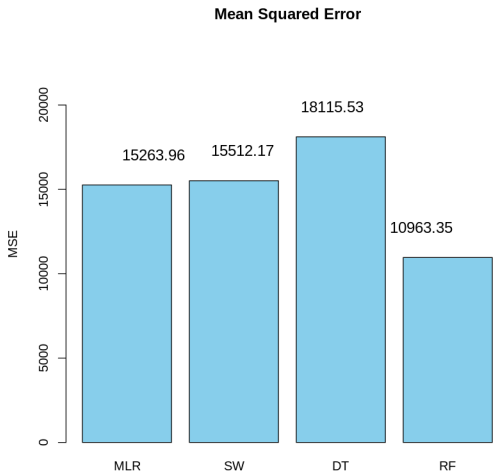


Figure 3: Mean Squared Error

Calculating accuracy for 4 models:

```
# Calculate SSE and SST for model_mlr
SSE_model_mlr <- sum((test_df$Recommended_Customer_Price - pred_mlr)^2)
SST_model_mlr <- sum((test_df$Recommended_Customer_Price -
↪  mean(test_df$Recommended_Customer_Price))^2)

# Calculate accuracy for model_mlr
accuracy_model_mlr <- round((1 - SSE_model_mlr / SST_model_mlr) * 100, 2)

# Print accuracy for model_mlr
cat("The accuracy of model_mlr: ", accuracy_model_mlr, "%\n")


# Calculate SSE and SST for model_sw
SSE_model_sw <- sum((test_df$Recommended_Customer_Price - pred_sw)^2)
SST_model_sw <- sum((test_df$Recommended_Customer_Price -
↪  mean(test_df$Recommended_Customer_Price))^2)

# Calculate accuracy for model_sw
accuracy_model_sw <- round((1 - SSE_model_sw / SST_model_sw) * 100, 2)

# Print accuracy for model_sw
cat("The accuracy of model_sw: ", accuracy_model_sw, "%\n")


# Calculate SSE and SST for model_dt
SSE_model_dt <- sum((test_df$Recommended_Customer_Price - pred_dt)^2)
SST_model_dt <- sum((test_df$Recommended_Customer_Price -
↪  mean(test_df$Recommended_Customer_Price))^2)
```

```r
# Calculate accuracy for model_dt
accuracy_model_dt <- round((1 - SSE_model_dt / SST_model_dt) * 100, 2)

# Print accuracy for model_dt
cat("The accuracy of model_dt: ", accuracy_model_dt, "%\n")


# Calculate SSE and SST for model_rf
SSE_model_rf <- sum((test_df$Recommended_Customer_Price - pred_rf)^2)
SST_model_rf <- sum((test_df$Recommended_Customer_Price -
↪  mean(test_df$Recommended_Customer_Price))^2)

# Calculate accuracy for model_rf
accuracy_model_rf <- round((1 - SSE_model_rf / SST_model_rf) * 100, 2)

# Print accuracy for model_rf
cat("The accuracy of model_rf: ", accuracy_model_rf, "%\n")
```

```
The accuracy of model_mlr:  57.97 %
The accuracy of model_sw:  57.29 %
The accuracy of model_dt:  50.12 %
The accuracy of model_rf:  69.81 %
```

Visualizing the accuracy by plotting them:

```r
# Create a vector of model names
models <- c("MLR", "Stepwise", "DT", "RF")

# Create a vector of accuracies for each model
accuracies <- c(accuracy_model_mlr, accuracy_model_sw, accuracy_model_dt, accuracy_model_rf)

# Create a bar plot
barplot(accuracies, names.arg = models, col = "skyblue", main = "Accuracies", ylab = "Accuracy
↪  (%)", ylim = c(0, 100))

# Add data labels
text(x = 1:length(models), y = accuracies + 2, labels = paste(accuracies, "%"), col = "black",
↪  pos = 3, cex = 1.2)
```

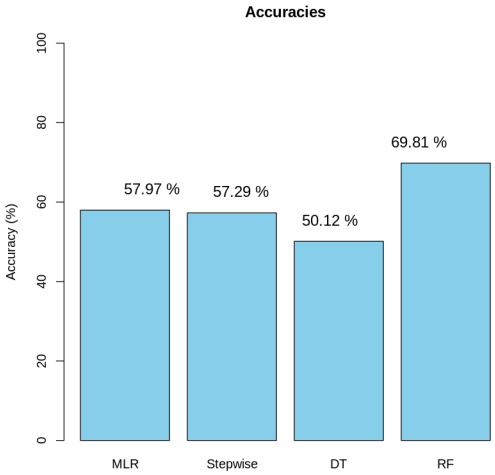


Figure 4: Accuracy

Accuracy of model_rf is 69.81%, which is the highest accuracy of the remaining models. Therefore, model_rf is the final model.

3 Conclusion

The "Intel CPUs.csv" file was imported for analysis in this report, and data cleansing and visualization were done. The processed dataset of 2283 CPU observations with 22 variables—among which one target variable, "Recommended Customer Price," had missing values—was then examined. We used linear models to handle these missing variables; we compared the performance of the models and ultimately chose the model that best fit the data.

Our results show that linear models are a useful tool for impute missing CPU price values. The chosen model made use of the data supplied by the other 21 variables to successfully and accurately predict missing values. The dataset analysis also provided some new information on the variables affecting CPU costs. These results demonstrate how crucial these hardware specifications are in establishing a CPU's total worth. We were able to use the entire dataset for additional analysis and modeling thanks to this method.

Despite our best efforts, there may be some flaws in this report due to the constraints of the dataset, lack of knowledge, and methodology. Your insightful comments would be greatly appreciated as it would enable us to refine and strengthen our analysis moving forward.

4 R script

```r
CPU_data<-read.csv("./sample_data/Intel_CPUs.csv", fileEncoding = 'latin1')
head(CPU_data,4)

install.packages("dplyr", type = "source")
install.packages("stringr", type = "source")
install.packages("GGally", type = "source")
install.packages("corrplot", type = "source")
install.packages("caTools", type = "source")
install.packages("MASS", type = "source")
install.packages("car", type = "source")
install.packages("e1071", type = "source")
install.packages("nortest", type = "source")

# Import libraries
install.packages("lifecycle")

library (dplyr)
library (stringr)
library (GGally)
library (corrplot)
library (caTools)
library (MASS) # for using Stepwise model
library (car)
library (e1071) # for using ANOVA
library (nortest)
library (ggplot2)

# Viewing the data structure
str(CPU_data)

# Get the summary
summary(CPU_data)

# Checking the missing values
# is.na(CPU_data) returns a logical matrix of the same size as your dataset data, where each
↪  element is TRUE if the corresponding element in data is NA and FALSE otherwise.
# any() function checks if any element in the logical matrix is TRUE. If at least one element is
↪  TRUE, any() returns TRUE; otherwise, it returns FALSE.
# therefore, there is the fact that the dataset has missing values since the output is TRUE.
any(is.na(CPU_data))


# Check which columns have missing values
columns_with_missing <- colSums(is.na(CPU_data)) > 0

# Display column names with missing values
names(columns_with_missing)[columns_with_missing]

# Calculating the number of missing values in each column
# '2' specifies the column
# 'sum' indicates the using function
h <- apply(is.na(CPU_data),2,sum)
print(h)

# Trim whitespace from column names
colnames(CPU_data) <- trimws(colnames(CPU_data))

# Calculate the sum of missing values for each column
```

```r
missing_counts <- apply(is.na(CPU_data), 2, sum)

# Identify columns with missing values in all rows
columns_with_missing_all <- names(missing_counts)[missing_counts == nrow(CPU_data)]

# Display column names with missing values in all rows
cat(columns_with_missing_all, sep="\n")

# Deleting the columns that are not necessary in analyzing the dataset (including the three above
↪  fields)
# Define a vector of column names to remove

columns_to_remove <- c("Product_Collection", "Vertical_Segment"
                       ,"Processor_Number","Status","Launch_Date","Cache"
                       ,"Bus_Speed","Conflict_Free","Memory_Types"
                       ,"Processor_Graphics_","Graphics_Output"
                       ,"Support_4k","Max_Resolution_HDMI"
                       ,"Max_Resolution_DP","Max_Resolution_eDP_Integrated_Flat_Panel"
                       ,"DirectX_Support","OpenGL_Support"
                       ,"PCI_Express_Revision","PCI_Express_Configurations_"
                       ,"Instruction_Set_Extensions")

# Remove specified columns from the dataset
CPU_data <- CPU_data[, !(names(CPU_data) %in% columns_to_remove)]
# After deleting
head(CPU_data)

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

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))



# Check for NAs after conversion
any(is.na(CPU_data))

head(CPU_data)

# Graphics_Max_Dynamic_Frequency
# Subset Data by GHz Frequency
subset_GHz <- CPU_data[grepl("GHz", CPU_data$Graphics_Max_Dynamic_Frequency, ignore.case =
↪  TRUE), ]
```

```r
# Remove GHz Frequency Rows
CPU_data <- CPU_data[!grepl("GHz", CPU_data$Graphics_Max_Dynamic_Frequency, ignore.case = TRUE),
↪  ]

# Remove "GHz" from Subset_GHz Column
subset_GHz$Graphics_Max_Dynamic_Frequency <- gsub("GHz", "",
↪  subset_GHz$Graphics_Max_Dynamic_Frequency, fixed = TRUE)

# Convert Subset_GHz Column to Numeric
subset_GHz$Graphics_Max_Dynamic_Frequency <-
↪  as.numeric(subset_GHz$Graphics_Max_Dynamic_Frequency)

# Convert Subset_GHz Frequency to MHz
subset_GHz$Graphics_Max_Dynamic_Frequency <- subset_GHz$Graphics_Max_Dynamic_Frequency * (1000)

# Remove "MHz" from CPU_data Column
CPU_data$Graphics_Max_Dynamic_Frequency <- gsub("MHz", "",
↪  CPU_data$Graphics_Max_Dynamic_Frequency, fixed = TRUE)

# Convert CPU_data Column to Numeric
CPU_data$Graphics_Max_Dynamic_Frequency <- as.numeric(CPU_data$Graphics_Max_Dynamic_Frequency)

# Merge Data
CPU_data <- bind_rows(CPU_data, subset_GHz)

# Processor_Base_Frequency

# Create a subset of data containing only rows where Processor_Base_Frequency column contains
↪  "GHz"
subset_GHz <- CPU_data[grepl("GHz", CPU_data$Processor_Base_Frequency, ignore.case = TRUE), ]

# Remove rows from the original dataframe where Processor_Base_Frequency contains "GHz"
CPU_data <- CPU_data[!grepl("GHz", CPU_data$Processor_Base_Frequency, ignore.case = TRUE), ]

# Remove "GHz" from the Processor_Base_Frequency column in the subset dataframe
subset_GHz$Processor_Base_Frequency <- gsub("GHz", "", subset_GHz$Processor_Base_Frequency,
↪  fixed = TRUE)

# Convert Processor_Base_Frequency column in the subset dataframe to numeric
subset_GHz$Processor_Base_Frequency <- as.numeric(subset_GHz$Processor_Base_Frequency)

# Convert Processor_Base_Frequency from GHz to MHz by multiplying by 1000
subset_GHz$Processor_Base_Frequency <- subset_GHz$Processor_Base_Frequency * (1000)

# Remove " MHz " from the Processor_Base_Frequency column in the original dataframe
CPU_data$Processor_Base_Frequency <- gsub("MHz", "", CPU_data$Processor_Base_Frequency, fixed =
↪  TRUE)

# Convert Processor_Base_Frequency column in the original dataframe to numeric
CPU_data$Processor_Base_Frequency <- as.numeric(CPU_data$Processor_Base_Frequency)

# Merge the subset dataframe with the original dataframe
CPU_data <- bind_rows(CPU_data, subset_GHz)

# Filling missing values for specified columns
columns_to_fill_mean <- c("Lithography", "nb_of_Cores", "nb_of_Threads", "Max_Turbo_Frequency",
                    "TDP", "Max_Memory_Size", "Max_nb_of_Memory_Channels",
                    ↪  "Max_Memory_Bandwidth",
                    "Graphics_Base_Frequency", "Graphics_Video_Max_Memory",
                    ↪  "Max_nb_of_PCI_Express_Lanes",
                    "T", "Graphics_Max_Dynamic_Frequency", "Processor_Base_Frequency")
```

```r
# Loop through each column
for (col in columns_to_fill_mean) {
  # Calculate the mean of the column, excluding missing values
  mean_value <- mean(CPU_data[[col]], na.rm = TRUE)

  # Replace missing values with the mean
  CPU_data[[col]][is.na(CPU_data[[col]])] <- mean_value
}

# Convert Recommended_Customer_Price column to numeric
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)

# Define a function to fill missing values with mode
fillmode <- function(column) {
  # Calculate mode value
  mode_value <- names(sort(table(column), decreasing = TRUE))[1]

  # Replace missing values with mode value
  column[is.na(column)] <- mode_value

  # Return the modified column
  return(column)
}

# Specify the columns to fill missing values for
columns_to_fill_2 <- 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")

# Apply the fillmode function to specified columns
CPU_data[columns_to_fill_2] <- lapply(CPU_data[columns_to_fill_2], fillmode)

# Re-check
any(is.na(CPU_data))

missing_values <- apply(is.na(CPU_data), 2, sum)

# Print the result horizontally
print(missing_values)

# Transform categorical variables to numerical variables

# Convert "Embedded_Options_Available" to numerical (1 for "Yes", 0 for "No")
CPU_data$Embedded_Options_Available <- ifelse(CPU_data$Embedded_Options_Available == "Yes", 1,
↪  0)

# Convert "ECC_Memory_Supported" to numerical (1 for "Yes", 0 for "No")
CPU_data$ECC_Memory_Supported <- ifelse(CPU_data$ECC_Memory_Supported == "Yes", 1, 0)

# Convert "Intel_Hyper_Threading_Technology_" to numerical (1 for "Yes", 0 for "No")
CPU_data$Intel_Hyper_Threading_Technology_ <- ifelse(CPU_data$Intel_Hyper_Threading_Technology
↪  == "Yes", 1, 0)

# Convert "Intel_Virtualization_Technology_VTx_" to numerical (1 for "Yes", 0 for "No")
CPU_data$Intel_Virtualization_Technology_VTx_ <-
↪  ifelse(CPU_data$Intel_Virtualization_Technology_VTx == "Yes", 1, 0)
```

```r
# Convert "Intel_64_" to numerical (1 for "Yes", 0 for "No")
CPU_data$Intel_64_ <- ifelse(CPU_data$Intel_64_ == "Yes", 1, 0)

# Convert "Idle_States" to numerical (1 for "Yes", 0 for "No")
CPU_data$Idle_States <- ifelse(CPU_data$Idle_States == "Yes", 1, 0)

# Convert "Thermal_Monitoring_Technologies" to numerical (1 for "Yes", 0 for "No")
CPU_data$Thermal_Monitoring_Technologies <- ifelse(CPU_data$Thermal_Monitoring_Technologies ==
↪  "Yes", 1, 0)

# Convert "Secure_Key" to numerical (1 for "Yes", 0 for "No")
CPU_data$Secure_Key <- ifelse(CPU_data$Secure_Key == "Yes", 1, 0)

# Convert "Execute_Disable_Bit" to numerical (1 for "Yes", 0 for "No")
CPU_data$Execute_Disable_Bit <- ifelse(CPU_data$Execute_Disable_Bit == "Yes", 1, 0)

# Convert "Instruction_Set" to numerical (0 for "32-bit", 1 for "64-bit", 2 for "Itanium")
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)

# Re-check
any(is.na(CPU_data))


missing_values <- apply(is.na(CPU_data), 2, sum)

# Print the result horizontally
print(missing_values)

# Create CPU_train dataset without missing values in Recommended_Customer_Price
CPU_train <- CPU_data[complete.cases(CPU_data$Recommended_Customer_Price), ]

# Create CPU_test dataset with missing values in Recommended_Customer_Price
CPU_test <- CPU_data[!complete.cases(CPU_data$Recommended_Customer_Price), ]

# Summary
summary(CPU_train)

his = c("Lithography", "Recommended_Customer_Price", "nb_of_Cores", "nb_of_Threads",
↪  "Processor_Base_Frequency",
        "Max_Turbo_Frequency", "TDP", "Max_Memory_Size", "Max_nb_of_Memory_Channels",
        ↪  "Max_Memory_Bandwidth",
        "Graphics_Base_Frequency", "Graphics_Max_Dynamic_Frequency",
        ↪  "Graphics_Video_Max_Memory",
        "Max_nb_of_PCI_Express_Lanes", "T")

for (var in his) {
  hist(CPU_train[[var]], main = var, xlab =" Units ", freq = FALSE, cex.main = 1)
}

box <- colnames(CPU_train)
for (var in box) {
  boxplot(CPU_train[[var]],
  col="green",
  xlab = var,
  cex.lab = 1,
  title.cex = 1,
  border = "brown")
```

```
}

# 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)

# Divide CPU_train into train_df and test_df
# Set the seed for reproducibility
set.seed (42)
# Use 70% of dataset as training set and 30% as test set (randomly)
split <- sample.split(CPU_train , SplitRatio = 0.70)
train_df <- subset(CPU_train, split == TRUE)
test_df <- subset(CPU_train, split == FALSE)

# MLR
# Fitting MLR model
model_mlr <- lm(Recommended_Customer_Price ~ ., data = train_df)
summary(model_mlr)

# Stepwise regression
# Fitting Stepwise Regression model
model_sw <- stepAIC(model_mlr, direction = "both")
summary(model_sw)

# 2-way ANOVA
```

```r
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)

# Check whether the distribution is normal
av_residual = rstandard(aov(Recommended_Customer_Price ~.,data =CPU_train))
shapiro.test(av_residual)

# Decision trees
library(rpart)
model_dt <- rpart(Recommended_Customer_Price ~ ., data = train_df)
summary(model_dt)

install.packages("randomForest",type="source")
library(randomForest)
model_rf <- randomForest(Recommended_Customer_Price ~ ., data = train_df)
summary(model_rf)

anova(model_mlr,model_sw)

# Ensure all models are trained on the same dataset
common_training_data <- train_df

# Fit MLR model
model_mlr <- lm(Recommended_Customer_Price ~ ., data = common_training_data)

# Fit Stepwise model (assuming you already have the stepwise model)
model_sw <- step(lm(Recommended_Customer_Price ~ ., data = common_training_data))

# Fit Decision Tree model
model_dt <- rpart(Recommended_Customer_Price ~ ., data = common_training_data)

# Fit Random Forest model
model_rf <- randomForest(Recommended_Customer_Price ~ ., data = common_training_data)

# Predictions on test data
pred_mlr <- predict(model_mlr, newdata = test_df)
pred_sw <- predict(model_sw, newdata = test_df)
pred_dt <- predict(model_dt, newdata = test_df)
pred_rf <- predict(model_rf, newdata = test_df)

# Calculate MSE for each model
mse_mlr <- mean((test_df$Recommended_Customer_Price - pred_mlr)^2)
mse_sw <- mean((test_df$Recommended_Customer_Price - pred_sw)^2)
mse_dt <- mean((test_df$Recommended_Customer_Price - pred_dt)^2)
mse_rf <- mean((test_df$Recommended_Customer_Price - pred_rf)^2)

# Print MSE for each model
cat("MLR MSE:", mse_mlr, "\n")
cat("Stepwise MSE:", mse_sw, "\n")
cat("Decision Tree MSE:", mse_dt, "\n")
cat("Random Forest MSE:", mse_rf, "\n")

# Create a vector of model names
models <- c("MLR", "SW", "DT", "RF")
```

```r
# Create a vector of MSE values for each model
mse_values <- c(mse_mlr, mse_sw, mse_dt, mse_rf)

# Set the ylim to include the maximum MSE value plus a buffer for readability
ylim_max <- max(mse_values) * 1.3  # Add a buffer of 10% to the maximum value
barplot(mse_values, names.arg = models, col = "skyblue", main = "Mean Squared Error", ylab =
↪  "MSE", ylim = c(0, ylim_max))

# Add data labels
text(x = 1:length(models), y = mse_values + 1000, labels = round(mse_values, 2), col = "black",
↪  pos = 3, cex = 1.2)

# Calculate SSE and SST for model_mlr
SSE_model_mlr <- sum((test_df$Recommended_Customer_Price - pred_mlr)^2)
SST_model_mlr <- sum((test_df$Recommended_Customer_Price -
↪  mean(test_df$Recommended_Customer_Price))^2)

# Calculate accuracy for model_mlr
accuracy_model_mlr <- round((1 - SSE_model_mlr / SST_model_mlr) * 100, 2)

# Print accuracy for model_mlr
cat("The accuracy of model_mlr: ", accuracy_model_mlr, "%\n")


# Calculate SSE and SST for model_sw
SSE_model_sw <- sum((test_df$Recommended_Customer_Price - pred_sw)^2)
SST_model_sw <- sum((test_df$Recommended_Customer_Price -
↪  mean(test_df$Recommended_Customer_Price))^2)

# Calculate accuracy for model_sw
accuracy_model_sw <- round((1 - SSE_model_sw / SST_model_sw) * 100, 2)

# Print accuracy for model_sw
cat("The accuracy of model_sw: ", accuracy_model_sw, "%\n")


# Calculate SSE and SST for model_dt
SSE_model_dt <- sum((test_df$Recommended_Customer_Price - pred_dt)^2)
SST_model_dt <- sum((test_df$Recommended_Customer_Price -
↪  mean(test_df$Recommended_Customer_Price))^2)

# Calculate accuracy for model_dt
accuracy_model_dt <- round((1 - SSE_model_dt / SST_model_dt) * 100, 2)

# Print accuracy for model_dt
cat("The accuracy of model_dt: ", accuracy_model_dt, "%\n")


# Calculate SSE and SST for model_rf
SSE_model_rf <- sum((test_df$Recommended_Customer_Price - pred_rf)^2)
SST_model_rf <- sum((test_df$Recommended_Customer_Price -
↪  mean(test_df$Recommended_Customer_Price))^2)

# Calculate accuracy for model_rf
accuracy_model_rf <- round((1 - SSE_model_rf / SST_model_rf) * 100, 2)

# Print accuracy for model_rf
cat("The accuracy of model_rf: ", accuracy_model_rf, "%\n")

# Create a vector of model names
models <- c("MLR", "Stepwise", "DT", "RF")
```

```r
# Create a vector of accuracies for each model
accuracies <- c(accuracy_model_mlr, accuracy_model_sw, accuracy_model_dt, accuracy_model_rf)

# Create a bar plot
barplot(accuracies, names.arg = models, col = "skyblue", main = "Accuracies", ylab = "Accuracy
↪  (%)", ylim = c(0, 100))

# Add data labels
text(x = 1:length(models), y = accuracies + 2, labels = paste(accuracies, "%"), col = "black",
↪  pos = 3, cex = 1.2)
```

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] Deepanshu Bhalla, *DECISION TREE IN R : STEP BY STEP GUIDE*. Available at: https://www.listendata.com/2015/04/decision-tree-in-r.html

[4] GeeksforGeeks (2023, May 23), *Stepwise regression in Python*. Available at: https://www.geeksforgeeks.org/stepwise-regression-in-python/

[5] Mahabeer, S. (2023, September 4), *Analysis of Variance (ANOVA)*. Available at: https://www.linkedin.com/pulse/analysis-variance-anova-shrikesh-mahabeer#:~:text=Analysis%20of%20variance%20(ANOVA)%20is%20an%20analysis%20tool%20used%20in,the%20random%20factors%20do%20not.