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

PROJECT REPORT

COURSE: PROBABILITY AND STATISTICS

Group 11 - Faculty of Computer Science and Engineering

Semester 233
Supervisor: TS.NGUYỄN TIẾN DŨNG

No.	Name	Student ID	Class
1.	Trần Anh Tuấn	2353276	DTQ1
2.	Vũ Tiến Vinh	2353334	DTQ1
3.	Mai Nam Phương	2252656	DTQ1
4.	Lâm Minh Tùng Quân	2352991	DTQ1
5.	Hoàng Đình Hiếu	2310945	DTQ1

HO CHI MINH CITY, AUGUST 2024

TABLE OF CONTENTS

1 Data Overview

This dataset contains detailed specifications, release dates, and release prices for Intel CPUs. The dataset consists of a single CSV file named `intel_cpus.csv` that includes various data attributes such as graphic base or max dynamic frequency, maximum temperature, power consumption, thread count, release date, recommended price, virtualization support,number of cores or threads of that cpu has and more.

2 Basic knowledge

2.1 Sample theoretical basis:

- **Some concepts:**
  - Statistical population (Population): is a set of elements belonging to the research object, which need to be observed, collected and analyzed according to one or some characteristics. The elements that make up the statistical population are called population units.
  - A sample (Sample) is a number of units selected from a population by some sampling method. The characteristics of the sample are used to generalize the characteristics of the population as a whole.
  - Statistical characteristics (research indicators) are important properties directly related to the research and survey content that require data collection on the population; They are divided into two types: attribute characteristics and quantitative characteristics.
  - Primary data is data collected directly from the research subject at the request of the researcher.
  - Secondary data is data from available sources, often already synthesized and processed; using secondary data helps researchers save time, effort and costs compared to collecting primary data; it should be noted that this data sometimes does not meet the more detailed requirements of the research.
- **Characteristics of random samples:**
  The function $g(X_1, \ldots, X_n)$ with $(X_1, \ldots, X_n)$ being a random sample is called a sample function or a statistic. Since the sample $(X_1, \ldots, X_n)$ is a random vector, $g(X_1, \ldots, X_n)$ is a random variable. There are two important groups of sample statistics that characterize population random variables:
  - The characteristic numbers give us an idea of the central position of the sample, that is, the tendency of the data in the sample to cluster around certain numbers. For example, the sample mean, the sample median, the sample mode, ...
  - Numbers that characterize the dispersion of data: mean deviation, standard deviation, and sample variance.
    * Sample mean: Consider a random sample $(X_1, \ldots, X_n)$ of BNN X, the statistic $\bar{X} = \frac{1}{n}(X_1 + X_2 + \ldots + X_n) = \frac{1}{n}\sum_{i=1}^{n} X_i$ is called the sample mean. For a specific sample $(x_1, \ldots, x_n)$ then: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the value that the sample mean receives corresponding to the given sample. If the data is given in a table, then $\bar{X} = \frac{1}{n}\sum_{i=1}^{k} x_i n_i$
    * Sample variance: Similar to the sample mean, the sample variance is defined as the expectation of the squared deviations of the sample components with respect to the sample mean and is denoted by:
      $S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$
      $\hat{S}^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$ with the statistic $\hat{S}^2$ called the incorrect sample standard deviation and the statistic S^2 called the corrected sample standard deviation.
    * Sample proportion (qualitative sample): $F = \frac{M}{n}$ and $f \equiv \hat{p} \equiv \frac{m}{n}$
    * Coefficient of variation (CV): Coefficient of variation measures the relative variability of a data sample, and is used when one wants to compare the variability of samples that do not have the same unit of measurement.
      CV (of sample) = $\frac{s}{x} \times 100\%$
    * Median: Suppose the sample of size n is arranged in ascending order according to the value under investigation: $x_1 \leq x_2 \leq \ldots \leq x_{n-1} \leq x_n$ If $n = 2k + 1$, the sample median is the value x_{k+1}. If $n = 2k$, the sample median is the value $\frac{x_k + x_{k+1}}{2}$.
    * Quartiles: The median value divides the ordered data sample into two sets of equal number of elements. The median of the smaller data set is Q1 (called the lower quartile) and the median of the larger data set is Q3 (called the upper quartile). Q2 is taken as the median value.
    * Outlier points: also known as outliers, exceptional points, outliers... These are sample elements whose values lie outside the range (Q1 − 1.5IQR; Q3 + 1.5IQR).

2.2 Statistical hypothesis testing:

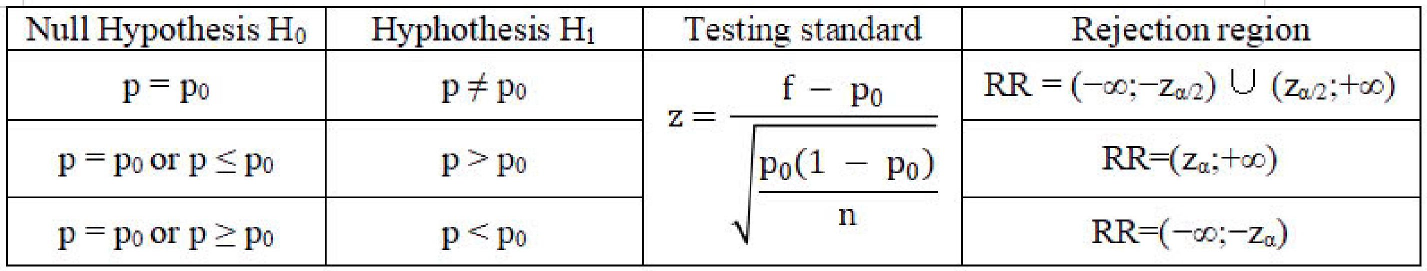
- Principle of small probability: if an event has a very small probability of occurring, it can be assumed that it will not occur when performing an experiment related to that event.
- Method of contradiction: If a correct hypothesis leads to an absurdity, we reject it (accept the alternative hypothesis).

*** General principles of statistical hypothesis testing:**

- Statistical Hypothesis Testing Criterion: From the original random variable X of the population, draw a random sample $X_1, \ldots, X_n$ and select a statistic $T = T(X_1, \ldots, X_n)$ which may depend on parameters specified in the null hypothesis H_0. If the null hypothesis H_0 is true, then the distribution law of T must be completely specified. Such a statistic is called a test criterion.
- Test Rule: If we succeed in dividing the domain of the test statistic T into two parts, R_α and $\bar{R_\alpha}$, where R_α is the rejection region and $\bar{R_\alpha}$ is the acceptance region of H_0.
- Type I and Type II errors With the above testing rule, the following two types of errors can be made: (i) Type I error: rejecting a true hypothesis. We see that the probability of making a type I error is exactly equal to the significance level α. Type I errors arise due to too small sample size, due to sampling method...
(ii) Type II Error: Accepting a false hypothesis. The probability of a Type II error is denoted by β and is defined as: $P(T \notin R_\alpha \mid H_1) = \beta$
- Statistical hypothesis testing procedure: Through the content presented above, we can build a statistical hypothesis testing procedure including the following steps:
(i) State the null hypothesis H_0 and the alternative hypothesis H_1.
(ii) From the total study population, randomly sample size n.
(iii) Choose the test criterion T and determine the probability distribution law of T under the condition that the hypothesis H_0 is true.
(iv) Based on the probability distribution of T, find the rejection region R_α such that: $P(T \in R_\alpha \mid H_1) = \alpha$

*** Tests using one sample:**

1. One-sample proportion test.



Null Hypothesis H_0	Hyphothesis H_1	Testing standard	Rejection region
$p = p_0$	$p \neq p_0$	$z = \dfrac{f - p_0}{\sqrt{\dfrac{p_0(1 - p_0)}{n}}}$	$RR = (-\infty; -z_{\alpha/2}) \cup (z_{\alpha/2}; +\infty)$
$p = p_0$ or $p \leq p_0$	$p > p_0$		$RR = (z_\alpha; +\infty)$
$p = p_0$ or $p \geq p_0$	$p < p_0$		$RR = (-\infty; -z_\alpha)$

2. One-sample mean test.

Type	Null Hypothesis H_0	Hyphothesis H_1	Hyphothesis H_1	Hyphothesis H_1
Normally distributed with known variance σ^2	$\mu = \mu_o$	$\mu \neq \mu_o$	$z = \dfrac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$	$RR = (-\infty;-z_{\alpha/2}) \cup (z_{\alpha/2};+\infty)$
		$\mu > \mu_o$		$RR=(z_\alpha;+\infty)$
		$\mu < \mu_o$		$RR=(-\infty;-z_\alpha)$
Normally distributed with unknown variance σ^2, n < 30	$\mu = \mu_o$	$\mu \neq \mu_o$	$t = \dfrac{\bar{x} - \mu_0}{s/\sqrt{n}}$	$RR = (-\infty;-t_{\alpha/2;n-1}) \cup (t_{\alpha/2;n-1};+\infty)$
		$\mu > \mu_o$		$RR=(t_{\alpha;n-1};+\infty)$
		$\mu < \mu_o$		$RR=(-\infty;-t_{\alpha;n-1})$
Arbitrary distribution with unknown variance σ^2, n $\geq$ 30	$\mu = \mu_o$	$\mu \neq \mu_o$	$z = \dfrac{\bar{x} - \mu_0}{s/\sqrt{n}}$	$RR = (-\infty;-z_{\alpha/2}) \cup (z_{\alpha/2};+\infty)$
		$\mu > \mu_o$		$RR=(z_\alpha;+\infty)$
		$\mu < \mu_o$		$RR=(-\infty;-z_\alpha)$

*** Tests using two samples:**

1. Two-sample proportion test.

Null Hypothesis H_0	Hyphothesis H_1	Testing standard	Rejection region
$p_1 = p_2$	$p_1 \neq p_2$	$z = \dfrac{f_1 - f_2}{\sqrt{\dfrac{\bar{f}(1 - \bar{f})}{\bar{n}}}}$ $\bar{f} = \dfrac{m_1 + m_2}{n_1 + n_2}$ $\bar{n} = \dfrac{n_1 n_2}{n_1 + n_2}$	$RR = (-\infty;-z_{\alpha/2}) \cup (z_{\alpha/2};+\infty)$
$p_1 = p_2$	$p_1 > p_2$		$RR=(z_\alpha;+\infty)$
$p_1 = p_2$	$p1 < p_2$		$RR=(-\infty;-z_\alpha)$

2. Two-sample mean test.

Type	Null Hypothesis H_0	Hyphothesis H_1	Hyphothesis H_1	Hyphothesis H_1
Normally distributed with known variances σ_1^2, σ_2^2 (z-test)	$\mu_1 = \mu_2$	$\mu_1 \neq \mu_2$	$z = \dfrac{\bar{x}_1 - \bar{x}_2}{\sqrt{\dfrac{\sigma^2{}_1}{n_1} + \dfrac{\sigma^2{}_2}{n_2}}}$	$RR = (-\infty; -z_{\alpha/2}) \cup (z_{\alpha/2}; +\infty)$
		$\mu_1 > \mu_2$		$RR = (z_\alpha; +\infty)$
		$\mu_1 < \mu_2$		$RR = (-\infty; -z_\alpha)$
Normally distributed with unknown variances σ_1^2, σ_2^2, $\sigma_1^2 = \sigma_2^2$, $n_1, n_2 \leq 30$ (t-test)	$\mu_1 = \mu_2$	$\mu_1 \neq \mu_2$	$t = \dfrac{\bar{x}_1 - \bar{x}_2}{\sqrt{\dfrac{s^2}{n_1} + \dfrac{s^2}{n_2}}}$	$RR = (-\infty; -t_{\alpha/2; n_1+n_2-2}) \cup (t_{\alpha/2; n_1+n_2-2}; +\infty)$
		$\mu_1 > \mu_2$		$RR = (t_{\alpha/2; n_1+n_2-2}; +\infty)$
		$\mu_1 < \mu_2$	$s^2 = \dfrac{(n_1 - 1)s^2{}_1 + (n_2 - 1)s^2{}_2}{n_1 + n_2 - 2}$	$RR = (-\infty; -t_{\alpha/2; n_1+n_2-2})$
Normally distributed with unknown variances σ_1^2, σ_2^2, $\sigma_1^2 \neq \sigma_2^2$, $n_1, n_2 \leq 30$ (t-test)	$\mu_1 = \mu_2$	$\mu_1 \neq \mu_2$	$t = \dfrac{\bar{x}_1 - \bar{x}_2}{\sqrt{\dfrac{s^2{}_1}{n_1} + \dfrac{s^2{}_2}{n_2}}}$	$RR = (-\infty; -t_{\alpha/2;v} \cup (t_{\alpha/2;v}; +\infty)$
		$\mu_1 > \mu_2$	$v = \dfrac{(\dfrac{s^2{}_1}{n_1} + \dfrac{s^2{}_2}{n_2})^2}{\dfrac{(\dfrac{s^2{}_1}{n_1})^2}{n_1 - 1} + \dfrac{(\dfrac{s^2{}_2}{n_2})^2}{n_2 - 1}}$	$RR = (t_{\alpha/2;v}; +\infty)$
		$\mu_1 < \mu_2$		$RR = (-\infty; -t_{\alpha/2;v})$
Arbitrary distribution with unknown variances σ_1^2, σ_2^2 and $n_1, n_2 \geq 30$ (z-test)	$\mu_1 = \mu_2$	$\mu_1 \neq \mu_2$	$z = \dfrac{\bar{x}_1 - \bar{x}_2}{\sqrt{\dfrac{s^2{}_1}{n_1} + \dfrac{s^2{}_2}{n_2}}}$	$RR = (-\infty; -z_{\alpha/2}) \cup (z_{\alpha/2}; +\infty)$
		$\mu_1 > \mu_2$		$RR = (z_\alpha; +\infty)$
		$\mu_1 < \mu_2$		$RR = (-\infty; -z_\alpha)$

2.3 Simple Linear Regression Problem:

Regression analysis is the study of the relationship between one variable (called the dependent variable) and one or more other variables (called the independent variables), with the goal of estimating the average (overall) value of the dependent variable based on the values of the independent variables, using a known sample.

* **Definition:** The regression function of Y with respect to X is the conditional expectation of Y given X, i.e., $E(Y|X)$. The simple linear regression equation is of the form: $f_Y(X) = E(Y|X) = \beta_0 + \beta_1 X$.

* **Simple Linear Regression Model:** The assumptions of the simple linear regression model are as follows: We have parameters β_0, β_1, and σ^2 such that for each value x of the independent variable, the dependent variable Y is related to x according to the equation $Y = \beta_0 + \beta_1 x + \epsilon$, where ϵ is a random error term with a normal distribution $N(0, \sigma^2)$.

* **Correlation coefficient and sample linear regression equation:**

(a) Sample correlation coefficient:

$$r = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{\sqrt{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}\sqrt{n\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}}$$

(b) Sample linear regression equation:

$$\bar{y_x} = A + Bx$$

with

$$B = \frac{n\sum_{i=1}^{k} n_i x_i y_i - \sum_{i=1}^{k} n_i x_i \sum_{i=1}^{k} n_i y_i}{n\sum_{i=1}^{k} n_i x_i^2 - \left(\sum_{i=1}^{k} n_i x_i\right)^2}$$

and

$$A = \frac{\sum_{i=1}^{n} y_i - B \cdot \sum_{i=1}^{n} x_i}{n}$$

2.4 ANOVA Problem (Analysis of Variance):

Analysis of Variance (ANOVA) is the comparison of the means of several groups (populations) based on the means of the observed samples from these groups and through testing the hypothesis of the conclusion about the equality of these population means. In research, analysis of variance is used as a tool to examine the influence of a causal factor (qualitative) on a result factor (quantitative).

- **One-factor ANOVA:** One-factor ANOVA is the analysis of the influence of a causal factor (qualitative variable) on a result factor (quantitative variable) under study. In the case of k populations with normal distribution and equal variance: Suppose that we want to compare the mean of k populations (for example k = 3) based on independent random samples of n1, n2, n3, ..., nk observed from k populations. It is necessary to remember the following three assumptions about the populations in which ANOVA is conducted:

  - The populations are normally distributed.
  - Population variances are equal.
  - Sampled observations are independent.

  If the population means are denoted as $\mu_1, \mu_2, \mu_3, \ldots, \mu_k$, then when the above assumptions are met, the one-way ANOVA model is described in terms of hypothesis testing as follows: $H_0 : \mu_1 = \mu_2 = \mu_3 = \ldots = \mu_k$. The hypothesis assumes that the means of the k populations are equal (in terms of the research relationship, this hypothesis assumes that the factor has no effect on the issue being studied). And the alternative hypothesis is: H_1 : There is at least one pair $\mu_i \neq \mu_j$, $i \neq j$. To test this, we propose the following three hypotheses:

  - Each sample follows a normal distribution $N(\mu, \sigma^2)$.
  - The population variances are equal.
  - We take k independent samples from k populations. Each sample is observed n_j times.

  ANOVA analysis steps:

  1. Calculate the mean values: Mean of each column and the overall mean.
  2. Calculate the sum of squares:
     - Sum of squares due to columns: SSG $= \sum_{i=1}^{k} n_i(\bar{x_i} - \bar{x})^2$
     - Total sum of squares: SST $= \sum_{i=1}^{k}\sum_{j=1}^{n_i}(x_{ij} - \bar{x})^2$
     - Sum of squares within columns: SSW = SST − SSG
  3. Calculate the variances:
     - Mean square due to columns: MSG $= \frac{\text{SSG}}{k-1}$
     - Mean square within columns: MSW $= \frac{\text{SSW}}{n-k}$
  4. Test statistic: $F = \frac{\text{MSG}}{\text{MSW}}$
  5. Reject the null hypothesis H_0 if $F > F_{k-1,n-k,\alpha}$

- **Two-factor variance analysis:** Two-factor variance analysis aims to simultaneously consider two causal factors (in the form of qualitative data) affecting the outcome factor (in the form of quantitative data) being studied. For example: Study the effect of fuel type and drying oven type on the rate of drying grade 1 fabric. Two-factor variance analysis helps us add causal factors to the analysis to make the research results more valuable.
  Suppose we study the effect of 2 qualitative causal factors on a certain quantitative outcome factor. We take a non-repetitive sample, then the sample units of the first causal factor are arranged into K groups (columns), the sample units of the second causal factor are arranged into H blocks (rows). Thus, we have a table combining 2 causal factors including K columns and H rows and (K x H) data cells. The total number of observed samples is n = (K x H).

  To test, we make the following two assumptions:

  - Each sample follows a normal distribution $N(\mu, \phi^2)$.
  - We take K independent samples from K populations and H independent samples from H populations. Each sample is observed once, without repetition.

  Steps to follow:

  1. Calculate the mean values.
  2. Compute the total sum of squares.
  3. Determine the variances.
  4. Perform the hypothesis test.

3 Data preprocessing

Key variables in the dataset:

- **Product_Collection**: Refers to the family or series that the CPU belongs to, such as Intel Core, AMD Ryzen, etc. Different collections target different markets or performance levels.
- **Vertical_Segment**: The specific market segment the CPU is designed for, such as desktop, mobile, server, or embedded systems.
- **Processor_Number** : A specific model number that identifies the CPU within a product line, like Intel i7-10700K or AMD Ryzen 5 3600.
- **Status**: The current market status of the CPU, indicating whether it is currently being produced, discontinued, or retired.
- **Launch_Date**: The date when the CPU was officially released to the market.
- **Lithography**: Refers to the manufacturing process technology, often measured in nanometers (nm), such as 7nm or 14nm. Smaller lithography typically indicates a more advanced, power-efficient, and potentially faster CPU.
- **Recommended_Customer_Price** : The price suggested by the manufacturer for customers to purchase the CPU.
- **nb_of_Cores**: The number of cores in the CPU. A core is an individual processing unit within the CPU, capable of executing its own instructions.
- **nb_of_Threads**: The number of threads the CPU can handle simultaneously. Threads represent virtual cores, helping in multitasking.
- **Processor_Base_Frequency**: The base operating speed of the CPU, usually measured in GHz. It indicates the clock speed when the CPU is running under normal conditions.
- **Max_Turbo_Frequency**: The maximum speed the CPU can achieve temporarily under load, thanks to technologies like Intel Turbo Boost.
- **Cache**: A small amount of high-speed memory located within the CPU used to store frequently accessed data and instructions. Cache sizes are usually measured in MB.
- **Bus_Speed**: The speed at which data is transferred between the CPU and the memory or other components. It's often measured in MHz.

- **TDP (Thermal Design Power)**: The amount of heat the CPU is expected to dissipate under maximum load, usually measured in watts. It's an indicator of the power consumption and cooling requirements.
- **Embedded_Options_Available**: The speed at which data is transferred between the CPU and the memory or other components. It's often measured in MHz.
- **Conflict_Free**: Indicates whether the CPU is made without conflict minerals, which are sourced from areas associated with human rights abuses.
- **Max_Memory_Size**: The maximum amount of memory (RAM) the CPU can support.
- **Memory_Types**: The types of RAM that are compatible with the CPU, such as DDR4 or DDR5.
- **Max_nb_of_Memory_Channels**: The maximum number of memory channels the CPU can utilize. More channels typically mean better memory bandwidth.
- **Max_Memory_Bandwidth**: The maximum rate at which data can be read from or stored into memory by the CPU, measured in GB/s.
- **ECC_Memory_Supported**: Indicates whether the CPU supports Error-Correcting Code (ECC) memory, which is used to detect and correct data corruption.
- **Processor_Graphics**: Refers to the integrated graphics processing unit (GPU) included within the CPU. Not all CPUs have integrated graphics.
- **Graphics_Base_Frequency**: The base frequency of the integrated graphics, indicating the speed at which the GPU operates under normal conditions.
- **Graphics_Max_Dynamic_Frequency**: The maximum frequency that the integrated graphics can reach temporarily under load.
- **Graphics_Video_Max_Memory**: The maximum amount of video memory that the integrated graphics can use.
- **Graphics_Output**: The types of video outputs supported by the integrated graphics, like HDMI, DisplayPort, etc.
- **Support_4k**: Indicates whether the CPU's integrated graphics can support 4K resolution displays.
- **Max_Resolution_HDMI**: The maximum resolution supported by the integrated graphics through an HDMI connection.
- **Max_Resolution_DP (DisplayPort)**: The maximum resolution supported by the integrated graphics through a DisplayPort connection.
- **Max_Resolution_eDP_Integrated_Flat_Panel**: The maximum resolution supported by the integrated graphics for an embedded DisplayPort (eDP) connected to an integrated flat panel.
- **DirectX_Support** : The version of Microsoft's DirectX API supported by the integrated graphics, which is important for gaming and multimedia applications.
- **OpenGL_Support**: The version of the OpenGL API supported by the integrated graphics, used in rendering 2D and 3D vector graphics.
- **PCI_Express_Revision**: The version of the PCI Express (PCIe) standard supported by the CPU, which affects the data transfer speed between the CPU and other components like GPUs.
- **PCI_Express_Configurations**: The possible configurations of PCIe lanes that the CPU can support (e.g., x16, x8, x4). This influences the performance and flexibility of the system.
- **Max_nb_of_PCI_Express_Lanes**: The maximum number of PCIe lanes available on the CPU. More lanes allow for more or faster connections to GPUs, SSDs, and other peripherals.
- **T**: This could be a specific variable or setting in the context of your data. In general, it might refer to a temperature setting or thermal threshold in CPUs, but it's ambiguous without additional context.

- **Intel_Hyper_Threading_Technology**: Intel's technology that allows a single physical CPU core to act like two virtual cores (threads), improving multitasking performance.
- **Intel_Virtualization_Technology_VTx**: Intel's technology that allows a CPU to run multiple operating systems or virtual machines on a single physical machine, enhancing virtualization performance.
- **Intel_64**: Refers to Intel's 64-bit architecture, which allows the CPU to handle more data at once compared to a 32-bit architecture.
- **Instruction_Set**: The set of instructions that the CPU can execute, like x86 or ARM. This defines how software interacts with the CPU.
- **Instruction_Set_Extensions**: Additional instruction sets supported by the CPU that extend its capabilities, like SSE, AVX, or MMX, used to accelerate specific types of computations.
- **Idle_States**:Power-saving states that the CPU can enter when it's not fully active, reducing energy consumption.
- **Thermal_Monitoring_Technologies**: Features that monitor and manage the CPU's temperature to prevent overheating and ensure stable operation.
- **Secure_Key**: A security feature that supports encryption by generating cryptographic keys directly on the CPU.
- **Execute_Disable_Bit**: A hardware-based security feature that helps prevent malicious code from being executed in certain memory areas, enhancing protection against some types of attacks.

4 Descriptive statistics

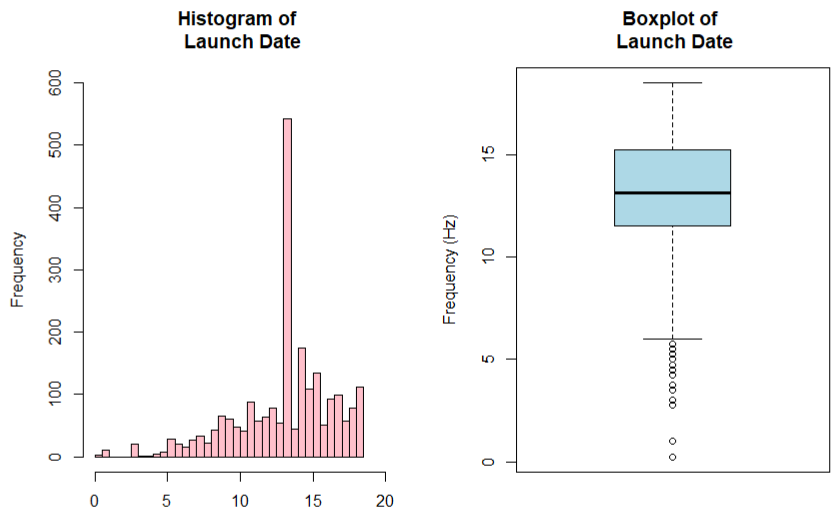
This section is going to plot histogram and boxplot of every significant field of the dataset

4.1 Launch date

- **Code:**

```
    #Diagram of Launch date
    hist(cpu_df$Launch_Date, breaks = 50, xlim = c(0,20), ylim = c(0,600),
        xlab = "", ylab = "Frequency",
        main = "Histogram of Launch_Date")
    boxplot(cpu_df$Launch_Date, main = "Boxplot of Launch_Date")
```

- **Result:**



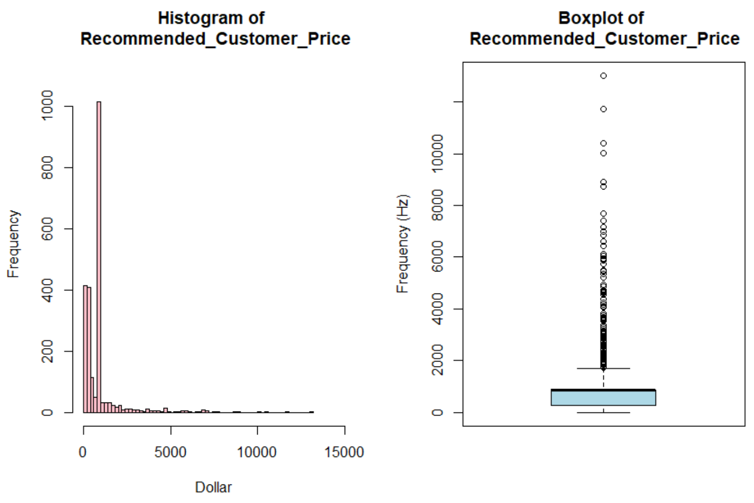
- **Comments:**
  - **Histogram of** `Launch_Date`**:**
    - **Shape of the Distribution:** The histogram displays the distribution of CPU launch dates. You might observe peaks indicating periods with high numbers of CPU releases, reflecting technological advancements or market demand spikes. For instance, if there are peaks around `Q1'10` and `Q2'20`, it suggests that these quarters saw more releases.
    - **Spread of the Data:** The width of the histogram shows the range of launch dates. If the bars are spread across many years, it suggests that the dataset includes CPUs released over a long time span, from early years to recent.
    - **Frequency of Launch Dates:** The y-axis represents the number of CPUs released in each quarter. Taller bars indicate more frequent releases in those periods, while shorter bars represent less common launch times.
  - **Box Plot of** `Launch_Date`**:**
    - **Median Launch Date:** The thick line inside the box represents the median launch date. For example, if the median is around `Q2'15`, it indicates that 50% of the CPUs were released before or during this quarter, and 50% were released after.
    - **Interquartile Range (IQR):** The length of the box shows the interquartile range, encompassing the middle 50% of the data. A longer box suggests greater variability in the launch dates of CPUs, while a shorter box indicates a more concentrated release period.
    - **Outliers:** Outliers are plotted as individual points outside the whiskers. These could be CPUs released in very early or very recent quarters compared to the majority of the dataset. For instance, a CPU released in `Q1'00` or `Q4'21` might be an outlier.
    - **Symmetry:** The position of the median within the box and the length of the whiskers provide insights into the distribution's symmetry. If the median is closer to one end, it suggests a skew in the data, with more CPUs released in either the earlier or later years.

4.2 Recommend customer price

- **Code:**

```r
    #Diagram of Recommended customer price
    hist(cpu_df$Recommended_Customer_Price, breaks = 50, xlab = "Dollar", ylab = "
       Frequency",
        xlim = c(0, 15000), ylim = c(0, 1100),
        main = "Histogramr of \n Recommended_Customer_Price")
    boxplot(cpu_df$Recommended_Customer_Price, main = "Boxplot of \n Recommended_
       Customer_Price")
```

- **Result:**



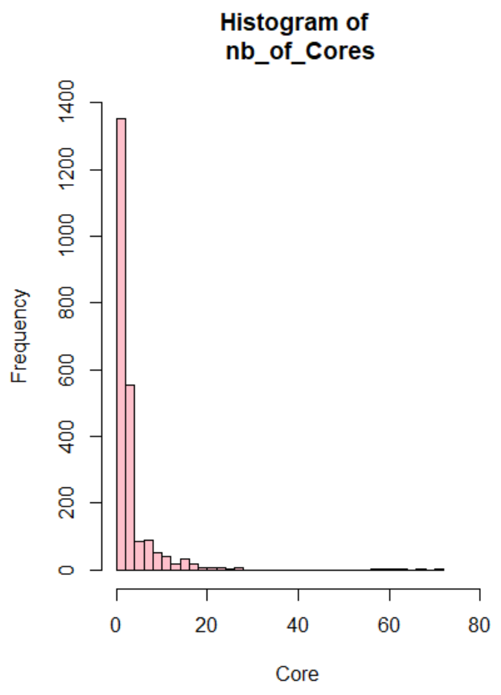
- **Comments:**
  - **Histogram of** `Recommended_Customer_Price`**:**
    - **Shape of the Distribution:** The histogram shows how prices are distributed. A peak in lower price ranges might suggest that most CPUs are budget-friendly, with fewer high-end options.
    - **Spread of the Data:** The width of the histogram indicates the range of prices. A broad spread suggests a wide range of CPU prices, from low to high.
    - **Frequency of Price Ranges:** Taller bars represent price ranges with more CPUs, while shorter bars show less common price ranges. For example, if most CPUs are priced between \$200 and \$500, this range will have higher bars.
  - **Box Plot of** `Recommended_Customer_Price`**:**
    - **Median Price:** The median, indicated by the thick line in the box, represents the middle price point. If the median is around \$300, it means that half of the CPUs are priced below this amount, and half are above.
    - **Interquartile Range (IQR):** The box length shows the range where the central 50% of prices fall. A larger box indicates more variability in CPU prices, while a smaller box indicates less variability.
    - **Outliers:** Outliers are individual points outside the whiskers and could be exceptionally high or low prices compared to the majority of CPUs.
    - **Symmetry:** The median's position within the box and the whisker lengths reveal whether the distribution is skewed. If the median is closer to one side, it suggests a skew towards higher or lower prices.

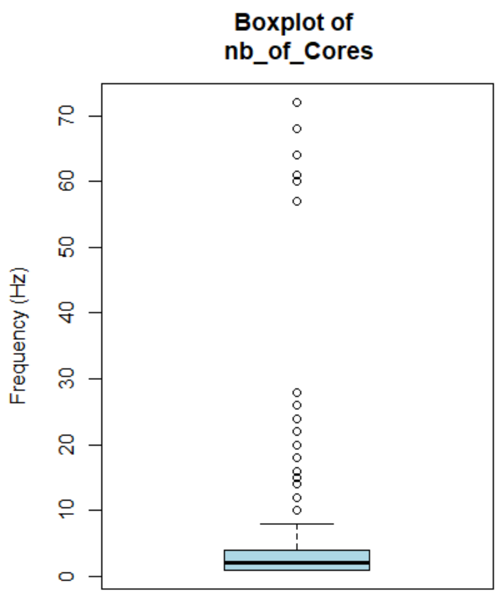
4.3 Number of cores

- **Code:**

```r
#Diagram of nb of cores
hist(cpu_df$nb_of_Cores, breaks = 50, xlab = "Core", ylab = "Frequency",
     xlim = c(0, 80), ylim = c(0, 1400), main = "Histogram of nb_of_Cores")
boxplot(cpu_df$nb_of_Cores, main = "Boxplot of nb_of_Cores")
```

- **Result:**





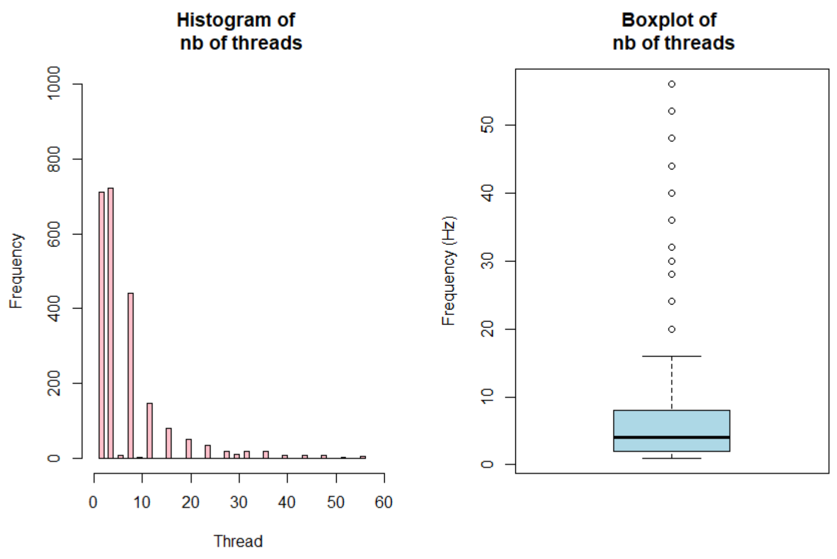
- **Comments:**
  - **Histogram of `nb_of_Cores`:**
    - **Shape of the Distribution:** The histogram displays the distribution of CPU core counts. Peaks might be observed at common core counts like `4`, `6`, or `8`, reflecting the typical configurations used in CPUs.
    - **Spread of the Data:** The width shows the range of core counts. A wider distribution suggests that CPUs in the dataset include both low and high core counts.
    - **Frequency of Core Counts:** Taller bars on the y-axis represent more frequent core counts, with shorter bars for less common counts. For instance, `4 cores` might be common, while `16 cores` are less frequent.
  - **Box Plot of `nb_of_Cores`:**
    - **Median Core Count:** The median core count is indicated by the thick line in the box. For example, if the median is `6`, it suggests that half of the CPUs have fewer than `6 cores`, and half have more.
    - **Interquartile Range (IQR):** The box length shows the variability of core counts. A longer box suggests a wider range of core counts among CPUs, while a shorter box indicates a narrower range.
    - **Outliers:** Outliers are core counts significantly different from the majority. For instance, `32 cores` might be an outlier if most CPUs have fewer cores.
    - **Symmetry:** The position of the median and whisker lengths reveal the data's symmetry. If the median is closer to one end, it indicates a skew in core counts.

4.4 Number of threads

- **Code:**

```r
#Diagram of nb of threads
hist(cpu_df$nb_of_Threads, breaks = 50, xlab = "Thread", ylab = "Frequency",
     xlim = c(0, 60), ylim = c(0, 1000), main = "Histogram of nb_of_Threads")
boxplot(cpu_df$nb_of_Threads, main = "Boxplot of nb_of_Threads")
```

- **Result:**



- **Comments:**
  - **Histogram of `nb_of_Threads`:**
    - **Shape of the Distribution:** The histogram shows the distribution of CPU threads. Peaks at common thread counts such as `8`, `12`, or `16` reflect typical configurations.

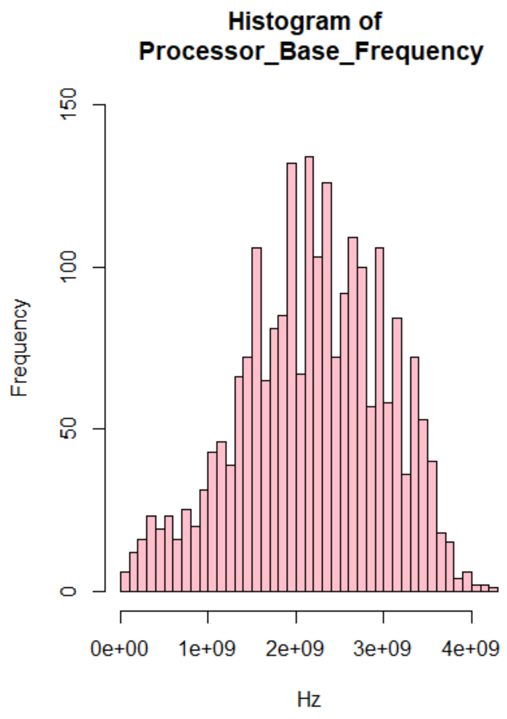
  - ∗ **Spread of the Data:** The width indicates the range of thread counts. A broad range suggests CPUs with a variety of thread counts are included.
    - ∗ **Frequency of Thread Counts:** Taller bars represent more frequent thread counts, while shorter bars indicate less common counts. For example, `8 threads` might be common, whereas `32 threads` are less frequent.
  - **Box Plot of `nb_of_Threads`:**
    - ∗ **Median Thread Count:** The median thread count is shown by the thick line in the box. If the median is `12`, it indicates that half of the CPUs have fewer than `12 threads`, and half have more.
    - ∗ **Interquartile Range (IQR):** The box length represents the middle 50% of thread counts. A longer box indicates greater variability, while a shorter box indicates less variability.
    - ∗ **Outliers:** Outliers are thread counts significantly different from the majority. For instance, '64 threads' might be an outlier if most CPUs have fewer threads.
    - ∗ **Symmetry:** The median's position and whisker lengths reveal symmetry in the distribution. A median closer to one end suggests skewness in thread counts.

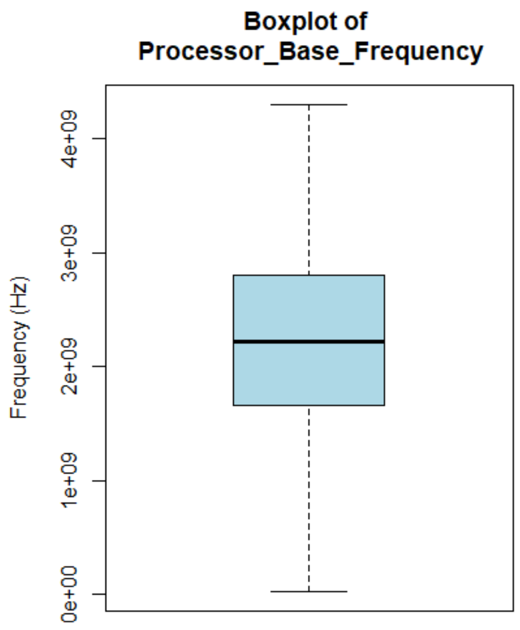
4.5 Processor base frequency

- **Code:**

```
# Create a histogram of Processor_Base_Frequency
hist(cpu_df$Processor_Base_Frequency,
    breaks = 50, xlim = c(0, 4.300e+09), ylim = c(0, 150),
    xlab = "Hz", ylab = "Frequency",
    main = "Histogram of \n Processor_Base_Frequency",
    col = "pink", border = "black")
# Create a box plot of Processor_Base_Frequency
boxplot(cpu_df$Processor_Base_Frequency,
    main = "Boxplot of \n Processor_Base_Frequency",
    ylab = "Frequency (Hz)", col = "lightblue", border = "black")
```

- **Result:**





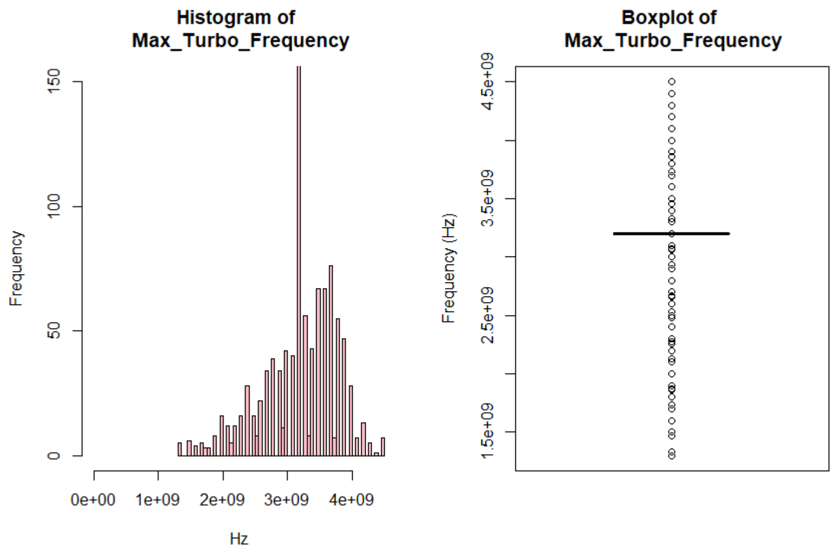
- **Comments:**
  - **Histogram of** `Processor_Base_Frequency`**:**
    - **Shape of the Distribution:** The histogram shows the distribution of base frequencies. Peaks at specific frequencies like `2.0 GHz` or `3.5 GHz` reflect common base speeds used in CPUs.
    - **Spread of the Data:** The width of the histogram indicates the range of frequencies. A broad range suggests diverse base frequencies across CPUs.
    - **Frequency of Base Frequencies:** Taller bars indicate more CPUs with those frequencies. For instance, `2.5 GHz` might be common, while higher frequencies like '5.0 GHz' are less frequent.
  - **Box Plot of** `Processor_Base_Frequency`**:**
    - **Median Base Frequency:** The median base frequency is shown by the thick line in the box. If the median is `3.0 GHz`, it indicates that half of the CPUs have base frequencies below `3.0 GHz`, and half are above.
    - **Interquartile Range (IQR):** The length of the box represents the variability in base frequencies. A longer box suggests a wider range of base frequencies, while a shorter box indicates less variability.
    - **Outliers:** Outliers are base frequencies significantly different from the majority. For example, frequencies above `4.5 GHz` might be outliers if most CPUs are below this threshold.
    - **Symmetry:** The position of the median and whisker lengths reveal if the data is skewed. A median closer to one end indicates a skew in base frequencies.

4.6 Max turbo frequency

- **Code:**

```
# Create a histogram of Max_Turbo_Frequency
hist(cpu_df$Max_Turbo_Frequency,
    breaks = 50, xlim = c(0, 4.500e+09), ylim = c(0, 150),
    xlab = "Hz", ylab = "Frequency",
    main = "Histogram of \n Max_Turbo_Frequency",
    col = "pink", border = "black")
# Create a box plot of Max_Turbo_Frequency
boxplot(cpu_df$Max_Turbo_Frequency,
    main = "Boxplot of \n Max_Turbo_Frequency",
    ylab = "Frequency (Hz)", col = "lightblue", border = "black")
```

- **Result:**



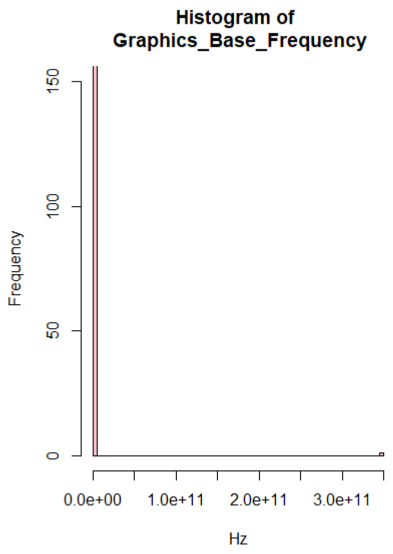
- **Comments:**
  - **Histogram of `Max_Turbo_Frequency`:**
    - **Shape of the Distribution:** The histogram shows how turbo frequencies are distributed. Peaks at common turbo speeds like `4.0 GHz` or `5.0 GHz` indicate typical boost capabilities.
    - **Spread of the Data:** The width of the histogram indicates the range of turbo frequencies. A broad range suggests diverse maximum turbo frequencies among CPUs.
    - **Frequency of Turbo Frequencies:** Taller bars represent more common turbo frequencies, while shorter bars show less frequent speeds. For instance, `4.5 GHz` might be common, while `5.5 GHz` is less frequent.
  - **Box Plot of `Max_Turbo_Frequency`:**
    - **Median Turbo Frequency:** The median turbo frequency is represented by the thick line in the box. If the median is `4.5 GHz`, it suggests that half of the CPUs have turbo frequencies below `4.5 GHz`, and half are higher.
    - **Interquartile Range (IQR):** The length of the box shows the variability of turbo frequencies. A longer box indicates a wide range of turbo frequencies, while a shorter box suggests less variability.
    - **Outliers:** Outliers are turbo frequencies significantly different from the majority. For example, frequencies above `5.0 GHz` might be outliers if most CPUs have lower boosts.
    - **Symmetry:** The position of the median and whisker lengths reveal if the data is skewed. A median closer to one end indicates skewness in turbo frequencies.

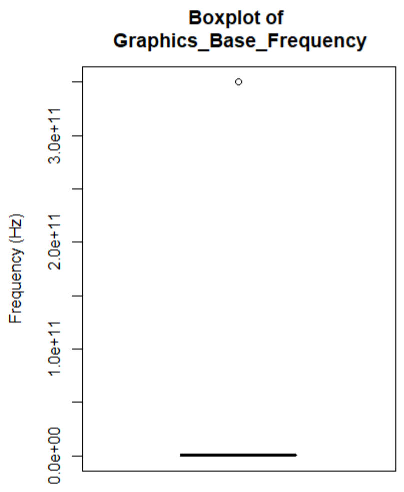
4.7 Graphic base frequency

- **Code:**

```r
# Create a histogram of Graphics_Base_Frequency
hist(cpu_df$Graphics_Base_Frequency,
    breaks = 50, xlim = c(0, 20000000000),  ylim = c(0, 3000),
    xlab = "Hz", ylab = "Frequency",
    main = "Histogram of \n Graphics_Base_Frequency",
    col = "pink", border = "black")
# Create a box plot of Graphics_Base_Frequency
boxplot(cpu_df$Graphics_Base_Frequency,
    main = "Boxplot of \n Graphics_Base_Frequency",
    ylab = "Frequency (Hz)", col = "lightblue", border = "black")
```

- **Result:**





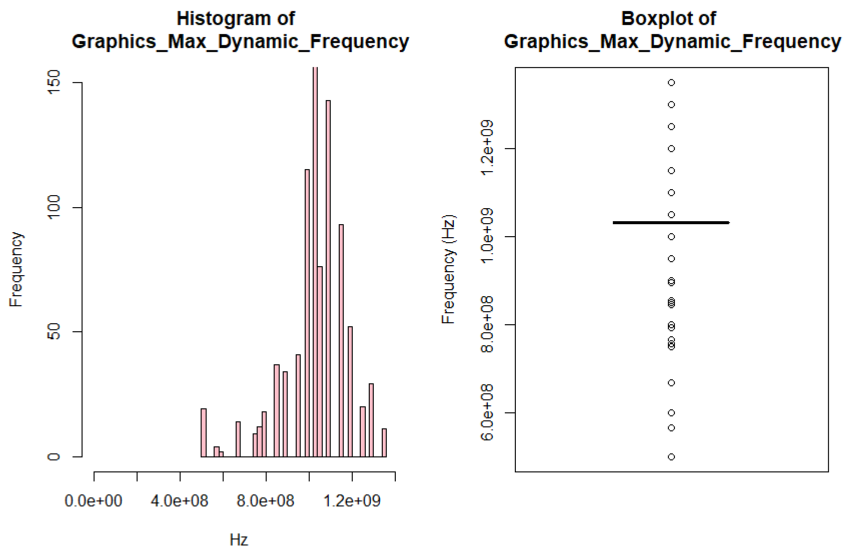
- **Comments:**
  - **Histogram of `Graphics_Base_Frequency`:**
    - **Shape of the Distribution:** The histogram shows the distribution of base frequencies for integrated graphics. Peaks at common frequencies like `300 MHz` or `500 MHz` indicate typical graphics base speeds
    - **Spread of the Data:** The width indicates the range of base frequencies. A broad range suggests a variety of base frequencies for graphics.
    - **Frequency of Graphics Base Frequencies:** Taller bars represent more common base frequencies, while shorter bars indicate less frequent frequencies. For example, `400 MHz` might be common, while `600 MHz` is less frequent.
  - **Box Plot of `Graphics_Base_Frequency`:**
    - **Median Graphics Base Frequency:** The median base frequency for graphics is shown by the thick line in the box. If the median is `400 MHz`, it suggests that half of the CPUs have graphics base frequencies below `400 MHz`, and half have more.
    - **Interquartile Range (IQR):** The length of the box represents the variability in graphics base frequencies. A longer box indicates a wider range of frequencies, while a shorter box suggests less variability.
    - **Outliers:** Outliers are graphics base frequencies significantly different from the majority. For instance, frequencies above `600 MHz` might be outliers if most GPUs have lower base frequencies.
    - **Symmetry:** The position of the median and whisker lengths reveal if the distribution is skewed. A median closer to one end indicates skewness in graphics base frequencies.

4.8 Graphic max dynamic frequency

- **Code:**

```r
# Create a histogram of Graphics_Max_Dynamic_Frequency
hist(cpu_df$Graphics_Max_Dynamic_Frequency,
    breaks = 50, xlim = c(0, 1.350e+09), ylim = c(0, 150),
    xlab = "Hz", ylab = "Frequency",
    main = "Histogram of \n Graphics_Max_Dynamic_Frequency",
    col = "pink", border = "black")
# Create a box plot of Graphics_Max_Dynamic_Frequency
boxplot(cpu_df$Graphics_Max_Dynamic_Frequency,
    main = "Boxplot of \n Graphics_Max_Dynamic_Frequency",
    ylab = "Frequency (Hz)", col = "lightblue", border = "black")
```

- **Result:**



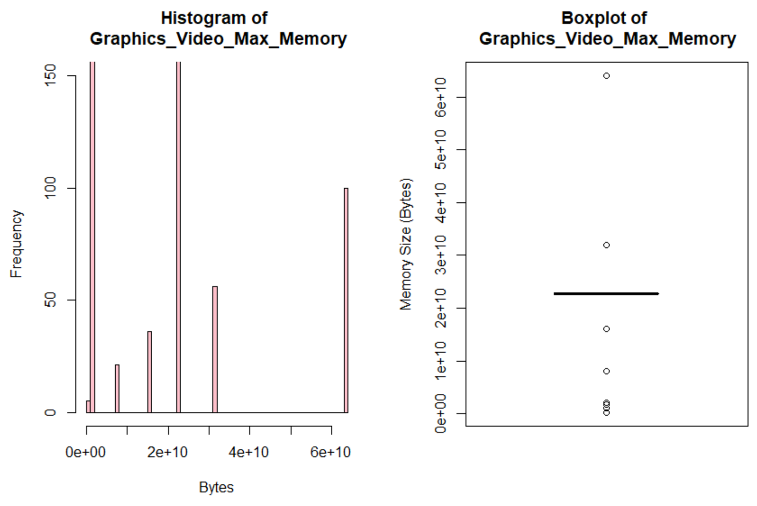
- **Comments:**
  - **Histogram of `Graphics_Max_Dynamic_Frequency`:**
    - **Shape of the Distribution:** The histogram shows the distribution of maximum dynamic frequencies for integrated graphics. Peaks at frequencies like `1.0 GHz` or `1.5 GHz` indicate common maximum speeds.
    - **Spread of the Data:** The width of the histogram indicates the range of maximum dynamic frequencies. A broad range suggests a variety of maximum speeds.
    - **Frequency of Graphics Base Frequencies:** Taller bars represent more common frequencies, while shorter bars indicate less frequent ones. For example, `1.2 GHz` might be common, while `1.8 GHz` is less frequent.
  - **Box Plot of `Graphics_Max_Dynamic_Frequency`:**
    - **Median Graphics Max Dynamic Frequency:** The median maximum dynamic frequency is shown by the thick line inside the box. If the median is `1.4 GHz`, it suggests that half of the CPUs have frequencies below `1.4 GHz`, and half have more.
    - **Interquartile Range (IQR):** The length of the box represents the variability in maximum dynamic frequencies. A longer box indicates a wider range of frequencies, while a shorter box suggests less variability.
    - **Outliers:** Outliers are frequencies significantly different from the majority. For instance, values above `1.6 GHz` might be outliers if most GPUs have lower frequencies.
    - **Symmetry:** The position of the median and whisker lengths reveal if the distribution is skewed. A median closer to one end indicates skewness in maximum dynamic frequencies.

4.9 Graphics video max memory

- **Code:**

```r
# Create a histogram of Graphics_Video_Max_Memory
hist(cpu_df$Graphics_Video_Max_Memory,
    breaks = 50, xlim = c(0, 6.40e+10), ylim = c(0, 150),
    xlab = "Bytes", ylab = "Frequency",
    main = "Histogram of \n Graphics_Video_Max_Memory",
    col = "pink", border = "black")
# Create a box plot of Graphics_Video_Max_Memory
boxplot(cpu_df$Graphics_Video_Max_Memory,
    main = "Boxplot of \n Graphics_Video_Max_Memory",
    ylab = "Memory Size (Bytes)", col = "lightblue", border = "black")
```

- **Result:**



- **Comments:**

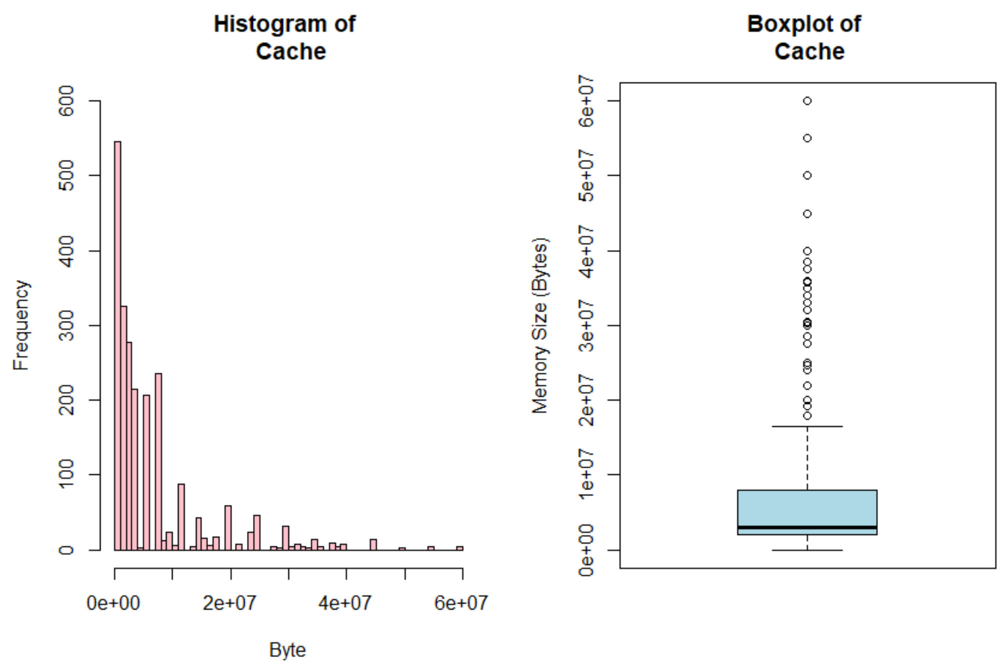
- **Histogram of `Graphics_Video_Max_Memory`:**
  - **Shape of the Distribution:** The histogram shows the distribution of maximum video memory for integrated graphics. Peaks at values like `1 GB` or `2 GB` indicate common maximum memory sizes.
  - **Spread of the Data:** The width indicates the range of video memory sizes. A broad range suggests that GPUs support various amounts of video memory.
  - **Frequency of Graphics Video Max Memory:** Taller bars represent more common memory sizes, while shorter bars indicate less frequent sizes. For example, 2 GB might be common, while `4 GB` is less frequent.
- **Box Plot of `Graphics_Video_Max_Memory`:**
  - **Median Graphics Video Max Memory:** The median maximum video memory is shown by the thick line inside the box. If the median is `2 GB`, it suggests that half of the GPUs have memory sizes below `2 GB`, and half have more.
  - **Interquartile Range (IQR):** The length of the box represents the variability in video memory sizes. A longer box indicates a wider range of supported memory sizes, while a shorter box suggests less variability.
  - **Outliers:** Outliers are memory sizes significantly different from the majority. For instance, `4 GB` or `8 GB` might be outliers if most GPUs have less memory.
  - **Symmetry:** The position of the median and the length of the whiskers reveal if the distribution is skewed. A median closer to one end indicates skewness in video memory sizes.

4.10 Cache

- **Code:**

```r
#Diagram of Cache
hist(cpu_df$Cache, breaks = 50, xlab = "Byte", ylab = "Frequency", ylim = c(0, 600),
     main = "Histogram of Cache")
boxplot(cpu_df$Cache, main = "Boxplot of Cache")
```

- **Result:**



- **Comments:**
  - **Histogram of `Cache`:**

- **Shape of the Distribution:** The histogram illustrates the distribution of cache sizes. Peaks at sizes like `8 MB` or `12 MB` show common cache sizes among CPUs.
- **Spread of the Data:** The width indicates the range of cache sizes. A broad range suggests that CPUs with various cache sizes are included.
- **Frequency of Cache Sizes:** Taller bars on the y-axis represent more common cache sizes. For instance, `16 MB` might be common, while larger sizes like `32 MB` are less frequent.

– **Box Plot of `Cache`:**

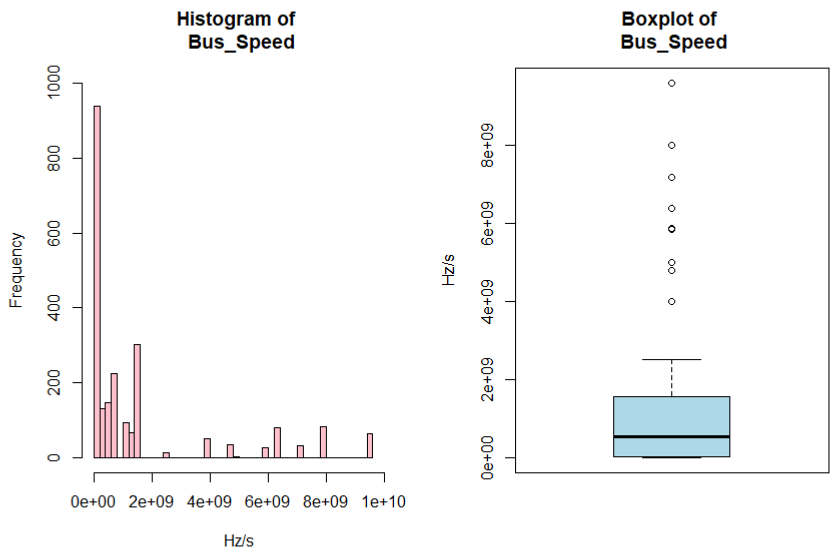
- **Median Cache Size:** The median cache size is shown by the thick line inside the box. If the median is `12 MB`, it indicates that half of the CPUs have cache sizes below 12 MB, and half have more.
- **Interquartile Range (IQR):** The box length shows the variability in cache sizes. A longer box indicates a wider range of cache sizes, while a shorter box suggests less variability.
- **Outliers:** Outliers are cache sizes significantly different from the majority. For instance, `64 MB` might be an outlier if most CPUs have less cache.
- **Symmetry:** The position of the median and the length of the whiskers reveal if the distribution is skewed. A median closer to one end indicates a skew in cache sizes.

4.11 BUS speed

- **Code:**

```
#Diagram of Bus speed
hist(cpu_df$Bus_Speed, breaks = 50, xlim = c(0, 10000000000), ylim = c(0, 1000),
     xlab = "Hz/s", ylab = "Frequency", main = "Histogram of Bus_Speed")
boxplot(cpu_df$Bus_Speed, main = "Boxplot of Bus_Speed")
```

- **Result:**



- **Comments:**

– **Histogram of `Bus_Speed`:**

- **Shape of the Distribution:** The histogram shows the distribution of bus speeds. Peaks at common speeds like `1.6 GHz` or `2.0 GHz` indicate typical speeds used in CPUs.
- **Spread of the Data:** The width of the histogram indicates the range of bus speeds. A wide range suggests diverse bus speeds across CPUs.
- **Frequency of Bus Speed:** Taller bars represent more common bus speeds, while shorter bars indicate less frequent speeds. For example, `1.8 GHz` might be common, while `2.5 GHz` is less frequent.

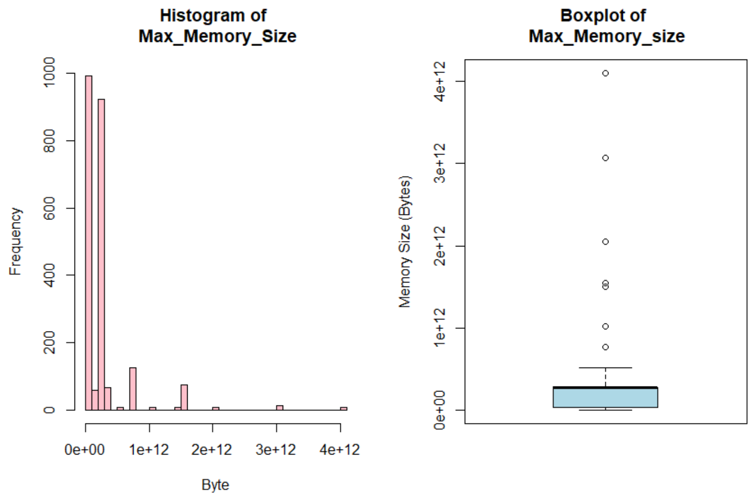
- **Box Plot of `Bus_Speed`:**
  - **Median Bus Speed:** The median bus speed is shown by the thick line in the box. If the median is `1.8 GHz`, it suggests that half of the CPUs have bus speeds below `1.8 GHz`, and half are higher.
  - **Interquartile Range (IQR):** The length of the box represents the variability in bus speeds. A longer box indicates a wider range of bus speeds, while a shorter box indicates less variability.
  - **Outliers:** Outliers are bus speeds significantly different from the majority. For instance, speeds above `2.2 GHz` might be outliers if most CPUs are below this threshold.
  - **Symmetry:** The position of the median and whisker lengths reveal if the data is skewed. A median closer to one end indicates skewness in bus speeds.

4.12 Max memory size

- **Code:**

```
#Diagram of Max memory size
hist(cpu_df$Max_Memory_Size, breaks = 50, ylim = c (0, 1000), xlab = "Byte",
    ylab = "Frequency", main = "Histogram of \n Max_Memory_Size")
boxplot(cpu_df$Max_Memory_Size, main = "Boxplot of \n Max_Memory_Size")
```

- **Result:**



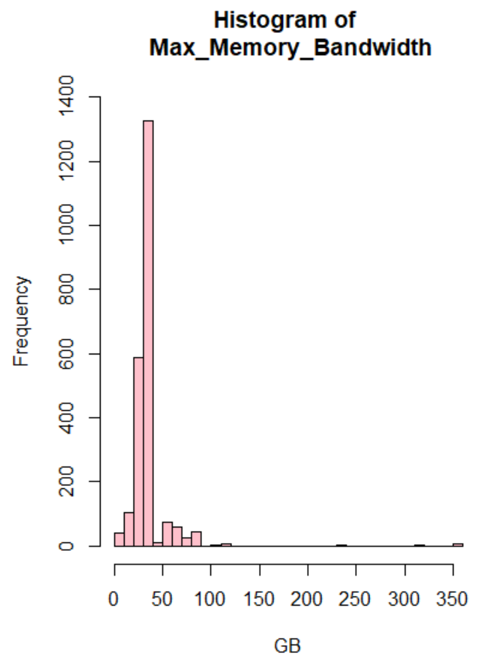
- **Comments:**
  - **Histogram of `Max_Memory_Size`:**
    - **Shape of the Distribution:** The histogram shows the distribution of maximum memory sizes supported by CPUs. Peaks at values like `64 GB` or `128 GB` indicate common maximum memory capacities.
    - **Spread of the Data:** The width of the histogram indicates the range of supported memory sizes. A broad range suggests CPUs supporting a wide variety of maximum memory sizes.
    - **Frequency of Memory Sizes:** Taller bars represent more common memory sizes, while shorter bars indicate less frequent sizes. For example, 128 GB might be common, while 256 GB is less frequent.
  - **Box Plot of `Max_Memory_Size`:**
    - **Median Memory Size:** The median maximum memory size is shown by the thick line inside the box. If the median is `128 GB`, it suggests that half of the CPUs support up to `128 GB` of memory, and half support more.
    - **Interquartile Range (IQR):** The length of the box represents the variability in memory sizes. A longer box indicates a wider range of maximum memory sizes, while a shorter box suggests less variability.
    - **Outliers:** Outliers are maximum memory sizes significantly different from the majority. For instance, `512 GB` might be an outlier if most CPUs support less memory.
    - **Symmetry:** The position of the median and whisker lengths reveal if the distribution is skewed. A median closer to one end indicates skewness in memory sizes.

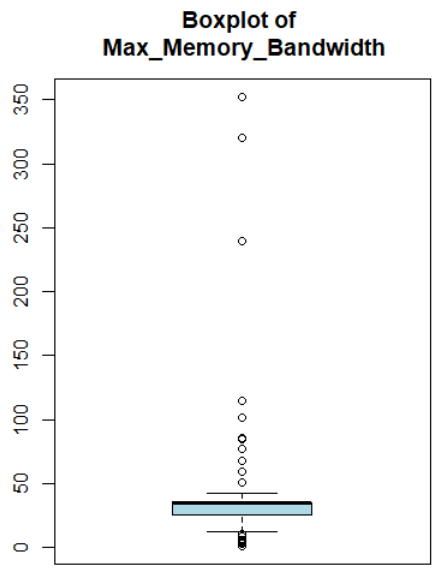
4.13 Max memory bandwidth

- **Code:**

```
#Diagram of Max memory bandwidth
hist(cpu_df$Max_Memory_Bandwidth, breaks = 50,
     ylim = c (0, 1400), xlab = "GB", ylab = "Frequency",
     main = "Histogram of \n Max_Memory_Bandwidth")
boxplot(cpu_df$Max_Memory_Bandwidth, main = "Boxplot of \n Max_Memory_Bandwidth")
```

- **Result:**





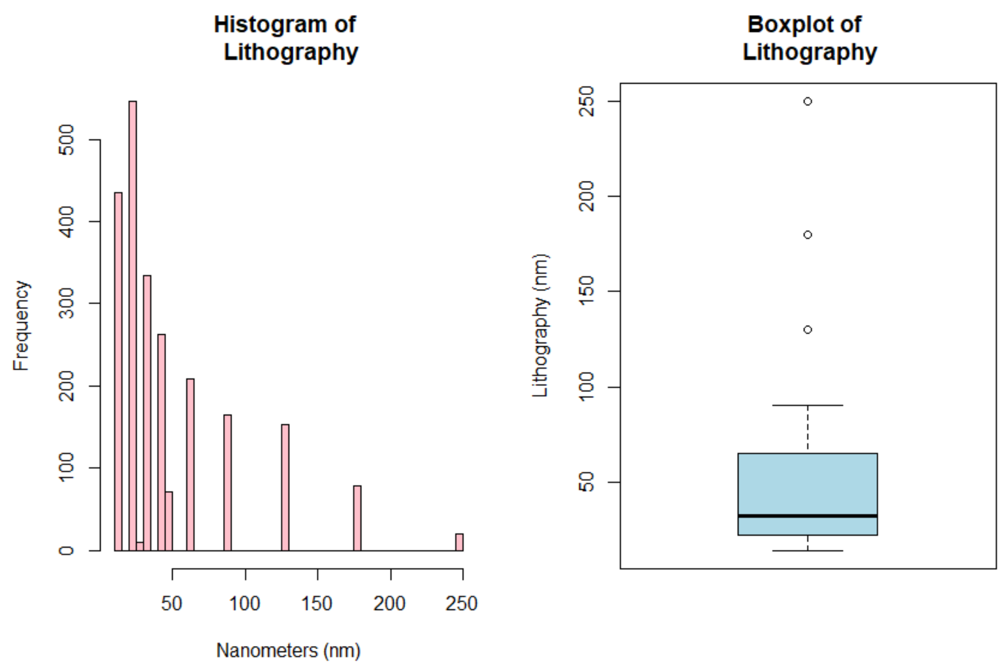
- **Comments:**
  - **Histogram of** `Max_Memory_Bandwidth`**:**
    - **Shape of the Distribution:** The histogram shows the distribution of maximum memory bandwidth values. Peaks at values like `25 GB/s` or `40 GB/s` indicate common bandwidth levels.
    - **Spread of the Data:** The width indicates the range of memory bandwidth values. A broad range suggests CPUs supporting a variety of bandwidth levels.
    - **Frequency of Memory Bandwidth:** Taller bars represent more common bandwidth levels, while shorter bars indicate less frequent levels. For example, `30 GB/s` might be common, while `50 GB/s` is less frequent.
  - **Box Plot of** `Max_Memory_Bandwidth`**:**
    - **Median Memory Bandwidth:** The median memory bandwidth is shown by the thick line in the box. If the median is `30 GB/s`, it suggests that half of the CPUs have bandwidth below `30 GB/s`, and half have more.
    - **Interquartile Range (IQR):** The length of the box represents the variability in memory bandwidth. A longer box indicates a wider range of supported bandwidths, while a shorter box suggests less variability.
    - **Outliers:** Outliers are memory bandwidth values significantly different from the majority. For instance, values above `50 GB/s` might be outliers if most CPUs have lower bandwidth.
    - **Symmetry:** The position of the median and the length of the whiskers reveal if the distribution is skewed. A median closer to one end indicates skewness in memory bandwidth.

4.14 Lithography

- **Code:**

```
# Create a histogram of Lithography
hist(cpu_df$Lithography,
    breaks = 50, xlab = "Nanometers (nm)", ylab = "Frequency",
    main = "Histogram of \n Lithography", col = "pink", border = "black")
# Create a box plot of Lithography
boxplot(cpu_df$Lithography,
    main = "Boxplot of \n Lithography", ylab = "Lithography (nm)", col = "lightblue"
        , border = "black")
```

- **Result:**



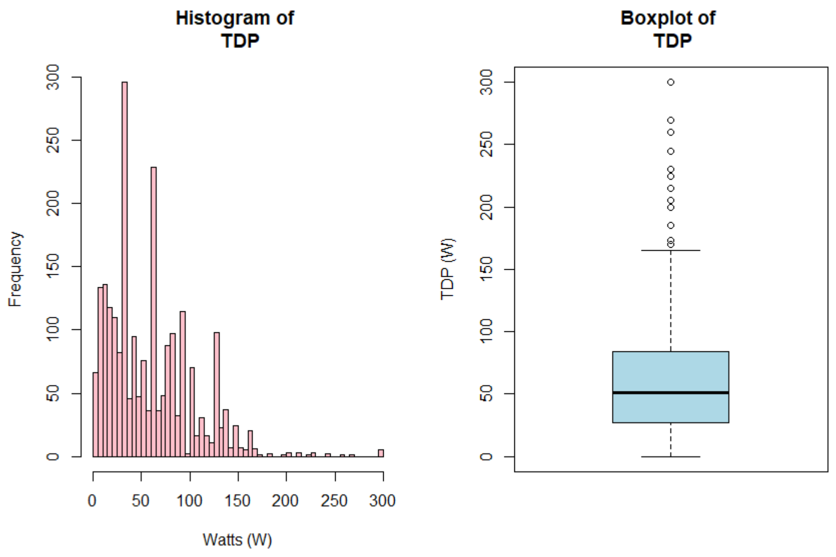
- **Comments:**
  - **Histogram of** `Lithography`**:**
    - **Shape of the Distribution:** The histogram illustrates how lithography sizes are distributed. Peaks at specific sizes, such as `14nm` or `10nm`, indicate common manufacturing processes used in many CPUs.
    - **Spread of the Data:** The histogram's width shows the range of lithography sizes. If there are several distinct peaks, it suggests the use of various lithography processes over time.
    - **Frequency of Memory Bandwidth:** Taller bars on the y-axis represent more common lithography sizes, while shorter bars indicate less frequent sizes. For example, if `14nm` and `7nm` are common, they will appear as high bars.
  - **Box Plot of** `Lithography`**:**
    - **Median Lithography Size:** The median, represented by the thick line inside the box, shows the central tendency of lithography sizes. For instance, if the median is `14nm`, this size is typical for many CPUs.
    - **Interquartile Range (IQR):** The length of the box indicates the middle 50% of lithography sizes. A shorter box suggests a smaller range of common sizes, while a longer box indicates greater variability.
    - **Outliers:** Outliers are points outside the whiskers and represent lithography sizes significantly different from the rest. For instance, very large or very small sizes compared to the median may appear as outliers.
    - **Symmetry:** The position of the median and the whisker lengths reveal if the distribution is symmetric. A median closer to one end of the box suggests a skew, with more CPUs using either smaller or larger lithography sizes.

4.15 Thermal design power (TDP)

- **Code:**

```r
# Create a histogram of TDP
hist(cpu_df$TDP,
    breaks = 50, xlab = "Watts (W)",  ylab = "Frequency", main = "Histogram of \n
       TDP",
    col = "pink", border = "black")
# Create a box plot of TDP
boxplot(cpu_df$TDP,
    main = "Boxplot of \n TDP", ylab = "TDP (W)", col = "lightblue", border = "black
       ")
```

- **Result:**



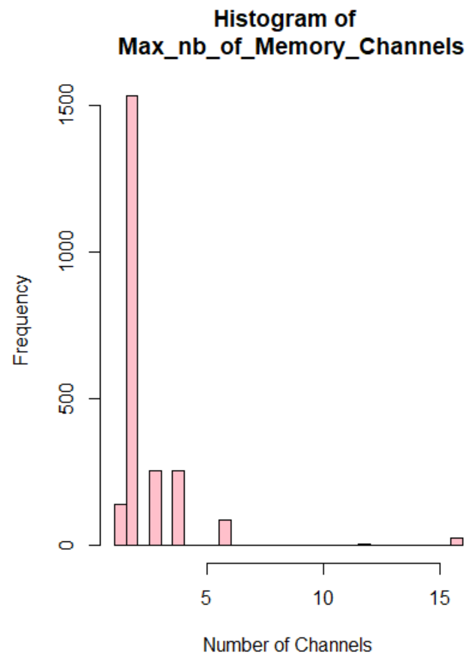
- **Comments:**
  - **Histogram of** `TDP`**:**
    - **Shape of the Distribution:** The histogram shows the distribution of Thermal Design Power (TDP) values. Peaks at values like `65W` or `95W` indicate common power consumption levels.
    - **Spread of the Data:** The width of the histogram indicates the range of TDP values. A broad range suggests diverse power requirements among CPUs.
    - **Frequency of TDP:** Taller bars represent more common TDP values, while shorter bars indicate less frequent power levels. For example, `95W` might be common, while `125W` is less frequent.
  - **Box Plot of** `TDP`**:**
    - **Median TDP:** The median TDP is shown by the thick line in the box. If the median is `95W`, it suggests that half of the CPUs have TDP values below `95W`, and half are higher.
    - **Interquartile Range (IQR):** The length of the box represents the variability in TDP values. A longer box indicates a wider range of power requirements, while a shorter box suggests less variability.
    - **Outliers:** Outliers are TDP values significantly different from the majority. For instance, values above `125W` might be outliers if most CPUs have lower power requirements.
    - **Symmetry:** The position of the median and the length of the whiskers reveal if the data is skewed. A median closer to one end indicates skewness in TDP values.

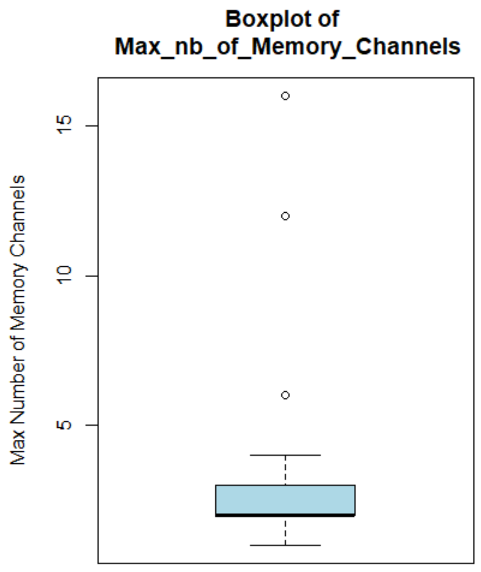
4.16 Max number of Memory channels

- **Code:**

```r
# Create a histogram of Max_nb_of_Memory_Channels
hist(cpu_df$Max_nb_of_Memory_Channels,
    breaks = 50, xlab = "Number of Channels", ylab = "Frequency",
    main = "Histogram of \n Max_nb_of_Memory_Channels",
    col = "pink", border = "black")
# Create a box plot of Max_nb_of_Memory_Channels
boxplot(cpu_df$Max_nb_of_Memory_Channels,
    main = "Boxplot of \n Max_nb_of_Memory_Channels",
    ylab = "Max Number of Memory Channels", col = "lightblue", border = "black")
```

- **Result:**





- **Comments:**
  - **Histogram of** `Max_nb_of_Memory_Channels`**:**
    - **Shape of the Distribution:** The histogram shows the distribution of the number of memory channels supported by CPUs. Peaks at common values like 2 or 4 indicate typical channel configurations.
    - **Spread of the Data:** The width indicates the range of supported memory channels. A broad range suggests CPUs with varying channel support.
    - **Frequency of Memory Bandwidth:** Taller bars represent more common memory channel counts, while shorter bars indicate less frequent configurations. For example, `2 channels` might be common, while `8 channels` is less frequent.
  - **Box Plot of** `Max_nb_of_Memory_Channels`**:**
    - **Median Number of Memory Channels:** The median number of memory channels is shown by the thick line inside the box. If the median is 4, it suggests that half of the CPUs support up to `4 channels`, and half support more.
    - **Interquartile Range (IQR):** The length of the box represents the variability in channel counts. A longer box indicates a wider range of supported channels, while a shorter box suggests less variability.

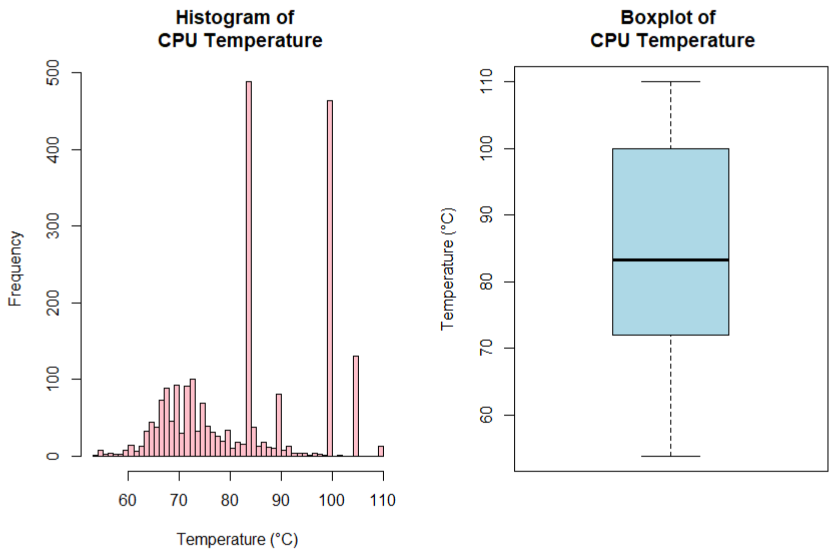
* **Outliers:** Outliers are channel counts significantly different from the majority. For instance, `8 channels` might be an outlier if most CPUs support fewer channels.
* **Symmetry:** The position of the median and the whisker lengths reveal if the distribution is skewed. A median closer to one end indicates skewness in channel counts.

4.17 CPU temperature (T)

- **Code:**

```r
# Create a histogram of Temperature (T)
hist(cpu_df$T,
    breaks = 50, xlab = "Temperature (degree of Celsius)", ylab = "Frequency",
    main = "Histogram of \n CPU Temperature", col = "pink", border = "black")
# Create a box plot of Temperature (T)
boxplot(cpu_df$T,
    main = "Boxplot of \n CPU Temperature", ylab = "Temperature (degree of Celsius)"
        ,
    col = "lightblue", border = "black")
```

- **Result:**



- **Comments:**
  - **Histogram of `T`:**
    * **Shape of the Distribution:** The histogram displays the distribution of operating temperatures. Peaks at temperatures like `70°C` or `80°C` indicate common operating ranges.
    * **Spread of the Data:** The width of the histogram shows the range of temperatures. A broad range suggests CPUs operate at various temperature levels.
    * **Frequency of Temperature:** Taller bars represent more common temperature ranges, while shorter bars indicate less frequent temperatures. For example, `75°C` might be common, while `90°C` is less frequent.
  - **Box Plot of `T`:**
    * **Median Temperature:** The median temperature is shown by the thick line in the box. If the median is `75°C`, it suggests that half of the CPUs operate at temperatures below `75°C`, and half at higher temperatures.
    * **Interquartile Range (IQR):** The length of the box represents the variability in temperatures. A longer box indicates a wider range of operating temperatures, while a shorter box suggests less variability.

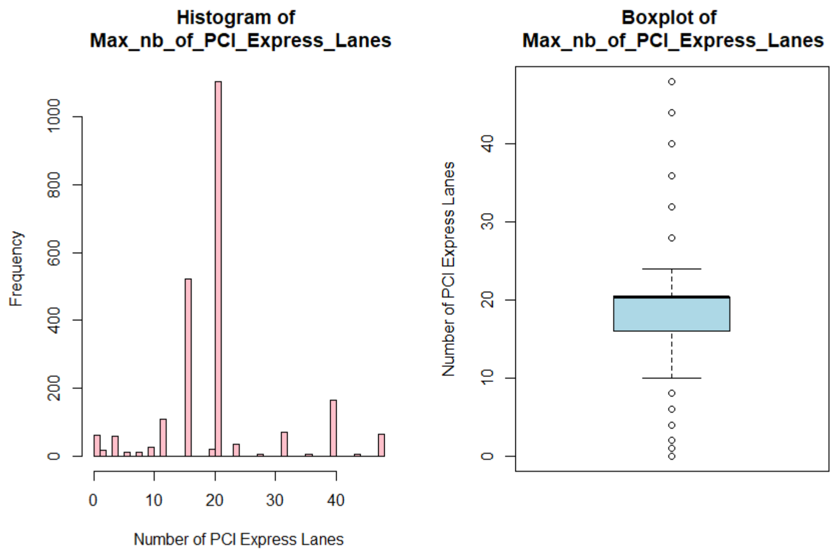
  * **Outliers:** Outliers are temperatures significantly different from the majority. For instance, temperatures above `85°C` might be outliers if most CPUs operate at lower temperatures.
  * **Symmetry:** The position of the median and the whisker lengths reveal if the distribution is skewed. A median closer to one end indicates skewness in operating temperatures.

4.18 Max number of PCI Express lanes

- **Code:**

```r
# Create a histogram of Max_nb_of_PCI_Express_Lanes
hist(cpu_df$Max_nb_of_PCI_Express_Lanes,
    breaks = 50, xlab = "Number of PCI Express Lanes", ylab = "Frequency",
    main = "Histogram of \n Max_nb_of_PCI_Express_Lanes",
    col = "pink", border = "black")
# Create a box plot of Max_nb_of_PCI_Express_Lanes
boxplot(cpu_df$Max_nb_of_PCI_Express_Lanes,
    main = "Boxplot of \n Max_nb_of_PCI_Express_Lanes",
    ylab = "Number of PCI Express Lanes", col = "lightblue", border = "black")
```

- **Result:**



- **Comments:**
  - **Histogram of `Max_nb_of_PCI_Express_Lanes`:**
    * **Shape of the Distribution:** The histogram shows the distribution of PCI Express lanes supported by CPUs. Peaks at values like `16` or `24` indicate common configurations.
    * **Spread of the Data:** The width of the histogram indicates the range of PCI Express lanes. A broad range suggests diverse support for PCI Express lanes.
    * **Frequency of PCI Express Lanes:** Taller bars represent more common configurations, while shorter bars indicate less frequent ones. For example, `16 lanes` might be common, while `32 lanes` are less frequent.
  - **Box Plot of `Max_nb_of_PCI_Express_Lanes`:**
    * **Median PCI Express Lanes:** The median number of PCI Express lanes is shown by the thick line in the box. If the median is `24`, it suggests that half of the CPUs support up to `24 lanes`, and half support more.
    * **Interquartile Range (IQR):** The length of the box represents the variability in PCI Express lane counts. A longer box indicates a wider range of supported lanes, while a shorter box suggests less variability.

* **Outliers:** Outliers are counts significantly different from the majority. For instance, `40 lanes` might be an outlier if most CPUs support fewer lanes.
* **Symmetry:** The position of the median and the whisker lengths reveal if the distribution is skewed. A median closer to one end indicates skewness in PCI Express lane counts.

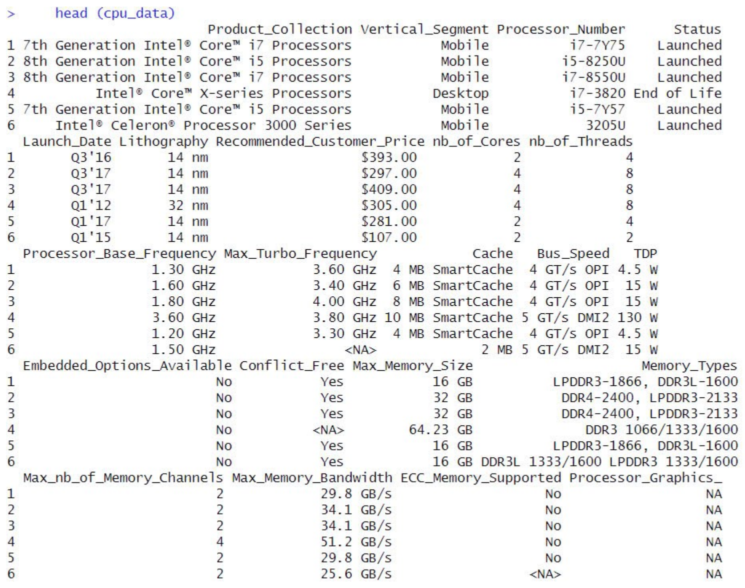
5 Inferential statistics

5.1 Read data

- Instructions: Use the `read.table()` command to read data from a file and store the data in the variable `test`.
- Code:

```
cpu_data = read.csv("G:\\Downloads\\Intel_CPUs.csv" , na.strings = c("", "N/A"))
missing_data <- apply(is.na(cpu_data), 2, sum)
print(missing_data)
data_feature <- c('Product_Collection',
                  'Launch_Date',
                  'Lithography',
                  'Recommended_Customer_Price',
                  'nb_of_Cores', 'nb_of_Threads',
                  'Processor_Base_Frequency',
                  'Max_Turbo_Frequency',
                  'Cache', 'Bus_Speed','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','Intel_Hyper_Threading_Technology_')
cpu_df <- cpu_data[, data_feature, drop = FALSE]
head (cpu_data)
```

- Result:



```
>    head (cpu_data)
                          Product_Collection Vertical_Segment Processor_Number      Status
1 7th Generation Intel® Core™ i7 Processors           Mobile          i7-7Y75    Launched
2 8th Generation Intel® Core™ i5 Processors           Mobile          i5-8250U    Launched
3 8th Generation Intel® Core™ i7 Processors           Mobile          i7-8550U    Launched
4         Intel® Core™ X-series Processors          Desktop          i7-3820 End of Life
5 7th Generation Intel® Core™ i5 Processors           Mobile          i5-7Y57    Launched
6     Intel® Celeron® Processor 3000 Series           Mobile            3205U    Launched
  Launch_Date Lithography Recommended_Customer_Price nb_of_Cores nb_of_Threads
1      Q3'16       14 nm                    $393.00           2             4
2      Q3'17       14 nm                    $297.00           4             8
3      Q3'17       14 nm                    $409.00           4             8
4      Q1'12       32 nm                    $305.00           4             8
5      Q1'17       14 nm                    $281.00           2             4
6      Q1'15       14 nm                    $107.00           2             2
  Processor_Base_Frequency Max_Turbo_Frequency             Cache   Bus_Speed   TDP
1                 1.30 GHz            3.60 GHz  4 MB SmartCache  4 GT/s OPI 4.5 W
2                 1.60 GHz            3.40 GHz  6 MB SmartCache  4 GT/s OPI  15 W
3                 1.80 GHz            4.00 GHz  8 MB SmartCache  4 GT/s OPI  15 W
4                 3.60 GHz            3.80 GHz 10 MB SmartCache 5 GT/s DMI2 130 W
5                 1.20 GHz            3.30 GHz  4 MB SmartCache  4 GT/s OPI 4.5 W
6                 1.50 GHz                <NA>             2 MB 5 GT/s DMI2  15 W
  Embedded_Options_Available Conflict_Free Max_Memory_Size                 Memory_Types
1                         No           Yes           16 GB       LPDDR3-1866, DDR3L-1600
2                         No           Yes           32 GB        DDR4-2400, LPDDR3-2133
3                         No           Yes           32 GB        DDR4-2400, LPDDR3-2133
4                         No          <NA>        64.23 GB           DDR3 1066/1333/1600
5                         No           Yes           16 GB       LPDDR3-1866, DDR3L-1600
6                         No           Yes           16 GB DDR3L 1333/1600 LPDDR3 1333/1600
  Max_nb_of_Memory_Channels Max_Memory_Bandwidth ECC_Memory_Supported Processor_Graphics_
1                         2            29.8 GB/s                   No                  NA
2                         2            34.1 GB/s                   No                  NA
3                         2            34.1 GB/s                   No                  NA
4                         4            51.2 GB/s                   No                  NA
5                         2            29.8 GB/s                   No                  NA
6                         2            25.6 GB/s                 <NA>                  NA
```

Because the `test` variable is large, we will use the `head()` function to print the first 6 lines of the `test` variable as an example.

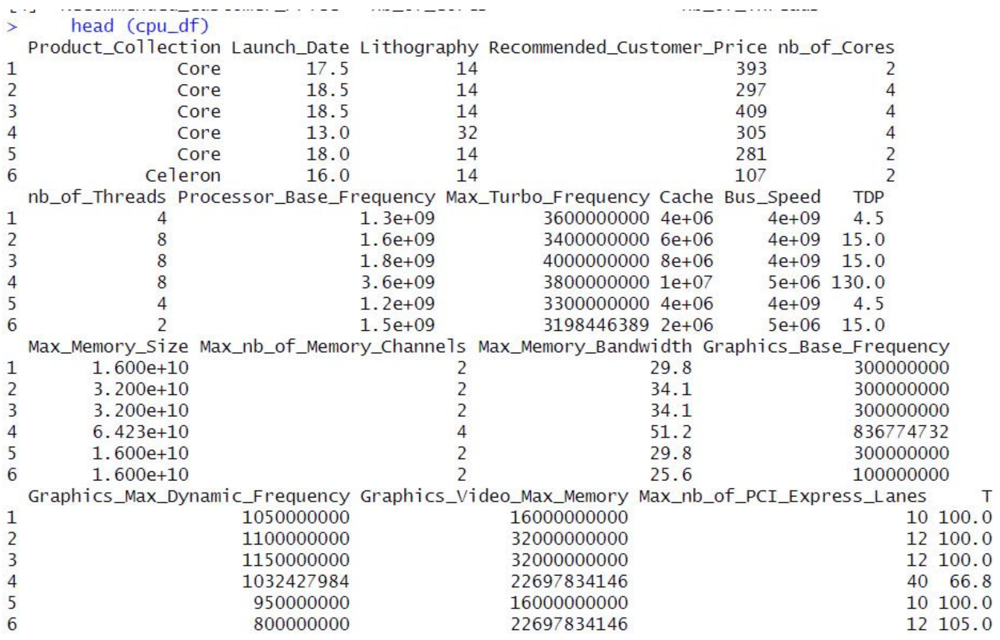
5.2 Data cleaning

Among the variables being examined, some contain many missing values (`NA` - Not Available). We use the following command to check and output a table that shows the percentage of missing values for each variable.

- Code:

```r
launch_date_replacements <- c('Q1\'00', 'Q2\'00', 'Q3\'00', 'Q4\'00', 'Q1\'01', 'Q2\'01', 'Q3\'01', 'Q4\'01', 'Q1\'02', 'Q2\'02', 'Q3\'02', 'Q4\'02', 'Q1\'03', 'Q2\'03', 'Q3\'03', 'Q4\'03', 'Q1\'04', 'Q2\'04', 'Q3\'04', 'Q4\'04', 'Q1\'05', 'Q2\'05', 'Q3\'05', 'Q4\'05', 'Q1\'06', 'Q2\'06', 'Q3\'06', 'Q4\'06', 'Q1\'07', 'Q2\'07', 'Q3\'07', 'Q4\'07', 'Q1\'08', 'Q2\'08', 'Q3\'08', 'Q4\'08', 'Q1\'09', 'Q2\'09', 'Q3\'09', 'Q4\'09', 'Q1\'10', 'Q2\'10', 'Q3\'10', 'Q4\'10', 'Q1\'11', 'Q2\'11', 'Q3\'11', 'Q4\'11', 'Q1\'12', 'Q2\'12', 'Q3\'12', 'Q4\'12', 'Q1\'13', 'Q2\'13', 'Q3\'13', 'Q4\'13', 'Q1\'14', 'Q2\'14', 'Q3\'14', 'Q4\'14', 'Q1\'15', 'Q2\'15', 'Q3\'15', 'Q4\'15', 'Q1\'16', 'Q2\'16', 'Q3\'16', 'Q4\'16', 'Q1\'17', 'Q2\'17', 'Q3\'17', 'Q4\'17', 'Q1\'18', 'Q2\'18', 'Q3\'18', 'Q4\'18', 'Q1 \'15', '04\'16', 'Q1\'99', 'Q2\'99')
replacement_values <- c(seq(1, 19, 0.25), 15.75, 17.0, 0.0, 0.25)
# Perform replacement
cpu_df$Launch_Date <- ifelse(cpu_df$Launch_Date %in% launch_date_replacements,
                             replacement_values[match(cpu_df$Launch_Date, launch_date_replacements)],
                             cpu_df$Launch_Date)
cpu_df <- cpu_df %>%
    mutate(Product_Collection = gsub('.*Core.*', 'Core', Product_Collection),
        Product_Collection = gsub('.*X-series.*', 'X-series', Product_Collection),
        Product_Collection = gsub('.*Celeron.*', 'Celeron', Product_Collection),
        Product_Collection = gsub('.*Pentium.*', 'Pentium', Product_Collection),
        Product_Collection = gsub('.*Quark.*', 'Quark', Product_Collection),
        Product_Collection = gsub('.*Core. [mM].*', 'm', Product_Collection),
        Product_Collection = gsub('.*Atom.*', 'Atom', Product_Collection),
        Product_Collection = gsub('.*Itanium.*', 'Itanium', Product_Collection),
        Product_Collection = gsub('.*Xeon.*', 'Xeon', Product_Collection))
PriceProcessor <- function(x) {
    x <- gsub(",", "", x)
    matches <- regmatches(x, gregexpr("\\$([0-9]+(\\.[0-9]+)?)", x))
    if (any(grepl("-", x))) {
    values <- as.numeric(gsub("\\$", "", unlist(matches)))
    if (length(values) >= 2) {
        ans <- mean(values, na.rm = TRUE)
    } else {
        ans <- NA
    }
    } else if (length(matches[[1]]) > 0) {
    ans <- as.numeric(sub("\\$", "", matches[[1]][1]))
    } else {
    ans <- NA
    }
    return(ans)
}
cpu_df$Recommended_Customer_Price <- sapply(cpu_df$Recommended_Customer_Price, PriceProcessor)
cpu_df$Recommended_Customer_Price <- as.numeric(cpu_df$Recommended_Customer_Price)
    # Function to convert frequency to Hz
    convert_to_hz <- function(frequency) {
    # Extract the numeric part and unit
    freq_number <- as.numeric(str_extract(frequency, "[\\d.]+"))
    freq_unit <- str_extract(frequency, "[a-zA-Z]+")
    # Convert to Hz
    if (grepl("GHz", freq_unit, ignore.case = TRUE)) {
        return(freq_number * 1e9) # Convert GHz to Hz
    } else if (grepl("MHz", freq_unit, ignore.case = TRUE)) {
        return(freq_number * 1e6) # Convert MHz to Hz
    } else if (grepl("Hz", freq_unit, ignore.case = TRUE)) {
        return(freq_number) # Already in Hz
    } else {
        return(NA) # Handle any unexpected cases
    }
}
# Function to convert memory size to Bytes
convert_to_bytes <- function(size) {
    # Extract the numeric part and unit
    size_number <- as.numeric(str_extract(size, "[\\d.]+"))
    size_unit <- str_extract(size, "[a-zA-Z]+")
    # Convert to Bytes
    if (grepl("GB", size_unit, ignore.case = TRUE)) {
        return(size_number * 1e9) # Convert GB to Bytes
    } else if (grepl("MB", size_unit, ignore.case = TRUE)) {
        return(size_number * 1e6) # Convert MB to Bytes
    } else if (grepl("KB", size_unit, ignore.case = TRUE)) {
        return(size_number * 1e3) # Convert KB to Bytes
    } else if (grepl("B", size_unit, ignore.case = TRUE)) {
        return(size_number) # Already in Bytes
    } else {
        return(NA) # Handle any unexpected cases
    }
}
# Function to remove "C" and convert to numeric
 remove_degree_celsius <- function(column_data) {
    # Remove "C" and convert to numeric
    numeric_data <- as.numeric(gsub("C", "", column_data))
    return(numeric_data)
}
# Function to remove "W" and convert to numeric
remove_watt <- function(column_data) {
    # Remove "W" and convert to numeric
    numeric_data <- as.numeric(gsub("W", "", column_data))
    return(numeric_data)
}
# Function to remove "nm" and convert to numeric
remove_nm <- function(column_data) {
    # Remove "nm" and convert to numeric
    numeric_data <- as.numeric(gsub("nm", "", column_data))
    return(numeric_data)
}
# Function to fill missing values in a column using the mean method
fill_column_mean <- function(column_data) {
    # Calculate the mean of the non-missing values
    mean_value <- mean(column_data, na.rm = TRUE)
    # Fill missing values with the mean
    column_data[is.na(column_data)] <- mean_value
    return(column_data)
}
# Function to fill missing values in a column using updown method
fill_column_updown <- function(column_data) {
    # Convert the column data to a data frame
    temp_df <- data.frame(column_data = column_data)
    # Fill missing values up and then down
    temp_df_filled <- temp_df %>% fill(column_data, .direction = "updown")
    # Return the filled column
    return(temp_df_filled$column_data)
}
# Function to fill missing values using the mean for Byte
fill_missing_with_mean_byte <- function(sizes) {
    # Convert all sizes to Bytes
    sizes_bytes <- sapply(sizes, convert_to_bytes)
    # Calculate the mean, excluding NA values
    mean_size <- mean(sizes_bytes, na.rm = TRUE)
    # Replace NA values with the mean
    sizes_filled <- ifelse(is.na(sizes_bytes), mean_size, sizes_bytes)
    return(sizes_filled)
}
# Function to fill missing values using the mean
fill_missing_with_mean <- function(frequencies) {
    # Convert all frequencies to Hz
    frequencies_hz <- sapply(frequencies, convert_to_hz)
    # Calculate the mean, excluding NA values
    mean_frequency <- mean(frequencies_hz, na.rm = TRUE)
    # Replace NA values with the mean
    frequencies_filled <- ifelse(is.na(frequencies_hz), mean_frequency, frequencies_hz)
    return(frequencies_filled)
}
# Function to convert a number to scientific notation
convert_to_scientific <- function(number) {
    # Convert the number to scientific notation
    scientific_notation <- format(number, scientific = TRUE)
    return(scientific_notation)
}
# Apply the function to the Processor_Base_Frequency column
cpu_df$Processor_Base_Frequency <- fill_missing_with_mean(cpu_df$Processor_Base_Frequency)
# Apply the function to the Graphics_Base_Frequency column
cpu_df$Graphics_Base_Frequency <- fill_missing_with_mean(cpu_df$Graphics_Base_Frequency)
# Apply the function to the Max_Turbo_Frequency column
cpu_df$Max_Turbo_Frequency <- fill_missing_with_mean(cpu_df$Max_Turbo_Frequency)
# Apply the function to the Graphics_Max_Dynamic_Frequency column
cpu_df$Graphics_Max_Dynamic_Frequency <- fill_missing_with_mean(cpu_df$Graphics_Max_Dynamic_
    Frequency)
# Apply the function to the Max_Memory_Size column
cpu_df$Graphics_Video_Max_Memory <- fill_missing_with_mean_byte(cpu_df$Graphics_Video_Max_
    Memory)
# Apply the function to the Max_nb_of_PCI_Express_Lanes
cpu_df$Max_nb_of_PCI_Express_Lanes <- fill_column_mean(cpu_df$Max_nb_of_PCI_Express_Lanes)
# Remove "C" and convert to numeric
cpu_df$T <- remove_degree_celsius(cpu_df$T)
# Fill missing values in the 'T' column using mean method
cpu_df$T <- fill_column_mean(cpu_df$T)
# Apply the function to the Max_nb_of_Memory_Channels
cpu_df$Max_nb_of_Memory_Channels <- fill_column_updown(cpu_df$Max_nb_of_Memory_Channels)
# Apply the function to the nb_of_Threads
cpu_df$nb_of_Threads <- fill_column_updown(cpu_df$nb_of_Threads)
# Remove "W" and convert to numeric
cpu_df$TDP <- remove_watt(cpu_df$TDP)
# Fill missing values in the 'TDP' column using mean method
cpu_df$TDP <- fill_column_mean(cpu_df$TDP)
# Remove "nm" and convert to numeric
cpu_df$Lithography <- remove_nm(cpu_df$Lithography)
# Fill missing values in the 'Lithography' column using mean method
cpu_df$Lithography <- fill_column_mean(cpu_df$Lithography)
get_numbers <- function(word) {
  if (is.character(word)) {
    return(as.numeric(str_extract(word, "[\\d]*[.]?[\\d]+")))
} else {
    return(word)
  }
}
CacheMapper = function(x) {
  if (is.numeric(x)){
    return(x)
  } else if (grepl("K", x)){
    fac <- 1000
  } else if (grepl("M", x)){
    fac <- 1000000
  } else if (grepl("G", x)){
    fac <- 1000000000
  } else if (grepl("T", x)){
    fac <- 1000000000000
  } else {
    fac <- 1
  }
  return(fac*get_numbers(x))
}
yes_is_1 = function(x){
  if(x == "Yes"){
    return(1);
  }else{
    return(0);
  }
}
cpu_df$Cache <- sapply(cpu_df$Cache, CacheMapper)
cpu_df$Bus_Speed <- sapply(cpu_df$Bus_Speed, CacheMapper)
cpu_df$Launch_Date <- as.numeric(cpu_df$Launch_Date)
cpu_df$Max_Memory_Size <- sapply(cpu_df$Max_Memory_Size, CacheMapper)
cpu_df$Max_Memory_Bandwidth <- gsub("[^0-9.]", "", cpu_df$Max_Memory_Bandwidth)
cpu_df$Max_Memory_Bandwidth <- as.numeric(cpu_df$Max_Memory_Bandwidth)
cpu_df$Product_Collection <- as.factor(cpu_df$Product_Collection)
for (col in 2: ncol(cpu_df)) {
  if (class(cpu_df[[col]]) != "character") {
    cpu_df[[col]] <- ifelse(is.na(cpu_df[[col]]), mean(cpu_df[[col]], na.rm = TRUE), cpu_df[[
        col]])
  }
}
```

- Result:



```
>    head (cpu_df)
  Product_Collection Launch_Date Lithography Recommended_Customer_Price nb_of_Cores
1               Core        17.5          14                        393           2
2               Core        18.5          14                        297           4
3               Core        18.5          14                        409           4
4               Core        13.0          32                        305           4
5               Core        18.0          14                        281           2
6            Celeron        16.0          14                        107           2
  nb_of_Threads Processor_Base_Frequency Max_Turbo_Frequency Cache Bus_Speed   TDP
1             4                  1.3e+09          3600000000 4e+06     4e+09   4.5
2             8                  1.6e+09          3400000000 6e+06     4e+09  15.0
3             8                  1.8e+09          4000000000 8e+06     4e+09  15.0
4             8                  3.6e+09          3800000000 1e+07     5e+06 130.0
5             4                  1.2e+09          3300000000 4e+06     4e+09   4.5
6             2                  1.5e+09          3198446389 2e+06     5e+06  15.0
  Max_Memory_Size Max_nb_of_Memory_Channels Max_Memory_Bandwidth Graphics_Base_Frequency
1       1.600e+10                         2                 29.8               300000000
2       3.200e+10                         2                 34.1               300000000
3       3.200e+10                         2                 34.1               300000000
4       6.423e+10                         4                 51.2               836774732
5       1.600e+10                         2                 29.8               300000000
6       1.600e+10                         2                 25.6               100000000
  Graphics_Max_Dynamic_Frequency Graphics_Video_Max_Memory Max_nb_of_PCI_Express_Lanes     T
1                     1050000000               16000000000                          10 100.0
2                     1100000000               32000000000                          12 100.0
3                     1150000000               32000000000                          12 100.0
4                     1032427984               22697834146                          40  66.8
5                      950000000               16000000000                          10 100.0
6                      800000000               22697834146                          12 105.0
```

From the results, we can see that there are no missing values in the dataset, so we can proceed with data processing.

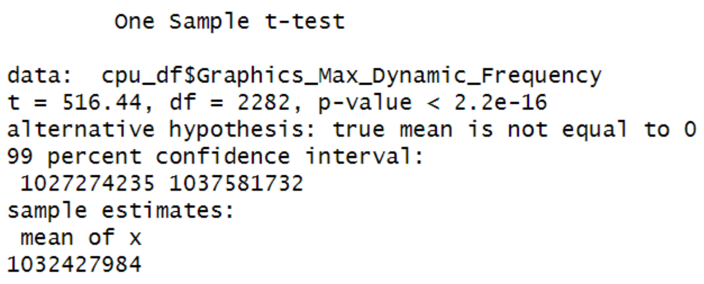
5.3 Find CI (Confidence Interval)

Use the t-test by using the `t.test()` command on the file `cpu_df.csv` with a significance level of 0.99. The result is printed with the title 'One-Sample t-Test' showing a calculated t-value of 516.44, degrees of freedom of 2282, and the confidence interval ranging from 1027274235 to 1037581732. Additionally, the mean value of this sample is 1032427984. Finding this confidence interval helps us verify the accuracy of the sample, as the mean value falls within the calculated confidence interval.

- Code:

```
t.test(cpu_df $Graphics_Max_Dynamic_Frequency, conf.level = 0.99)
```

- Result:



```
        One Sample t-test

data:  cpu_df$Graphics_Max_Dynamic_Frequency
t = 516.44, df = 2282, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 1027274235 1037581732
sample estimates:
 mean of x
1032427984
```

5.4 One-sample testing

Example: A customer believes that the average recommended price is not less than 800$.
Let μ be the average recommended price.

Hypotheses:

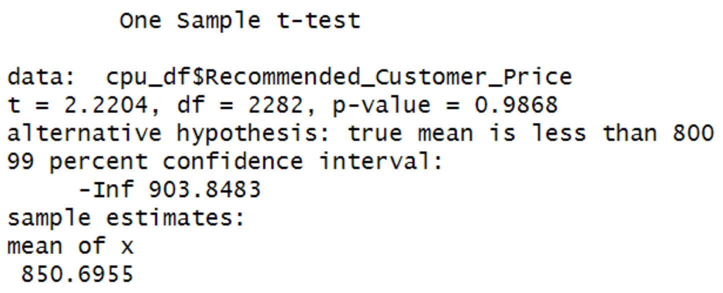
- H_0: $\mu \geq 800$$
- H_1: $\mu < 800$$

Use the `t.test()` command on the file `Intel_CPU.csv` with `alternative="less"` to perform a one-sided test (left-sided) with a significance level of 0.99. The degrees of freedom (df) are 2282. Note that since the p-value $\approx 0.9871 > 0.05$, we reject H_1 and accept H_0, meaning that we can say the average recommended price is not less than 800$.

- Code:

```r
#a cpus has a recomended price is not less than 800$ with confidence level = 99%
t.test(cpu_df $ Recommended_Customer_Price , alternative =  "less" , mu = 800, conf.level =
    0.99)
```

- Result:



```
                One Sample t-test

data:  cpu_df$Recommended_Customer_Price
t = 2.2204, df = 2282, p-value = 0.9868
alternative hypothesis: true mean is less than 800
99 percent confidence interval:
     -Inf 903.8483
sample estimates:
mean of x
 850.6955
```

5.5 Two-sample testing

Example: The Welch Two Sample t-test results indicate that CPUs with Intel Hyper-Threading Technology (Yes) have a significantly higher average Recommended_Customer_Price than CPUs without this technology (No). The mean difference is statistically significant, with a confidence interval suggesting that, on average, CPUs with hyper-threading technology are priced between approximately \$376 and \$538 higher than those without it.

Hypotheses:

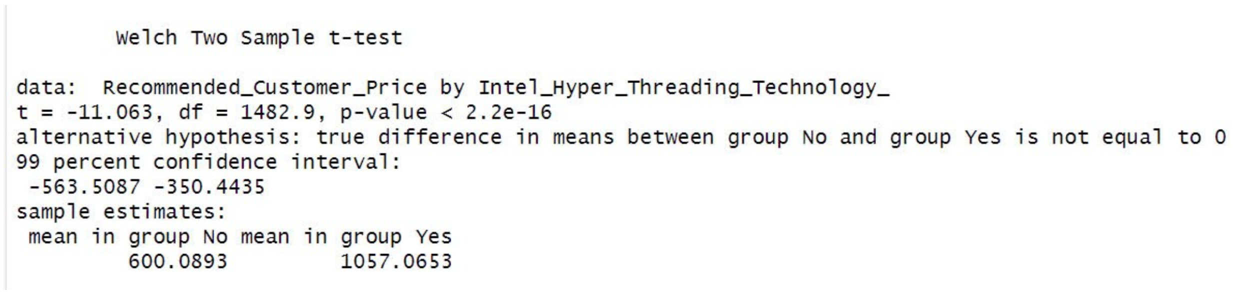
- H_0: $\mu_1 = \mu_2$
- H_1: $\mu_1 \neq \mu_2$

Use the `t.test()` command on the file `cpu_df.csv` to perform a two-sided test with a significance level of 0.99. The degrees of freedom (df) are 2282. Note that since the p-value $< 2.2 \times 10^{-16} < 0.05$, we reject H_0 and accept H_1, meaning that there is a statistically significant difference in the versions of recommended price between the versions that have the Intel hyper threading technology and those do not.

- Code:

```r
#There is no significant difference in the recommended price between CPUs equipped with Intel
     Hyper-Threading Technology and those without it.
t.test(Recommended_Customer_Price ~ Intel_Hyper_Threading_Technology_, data = cpu_df, conf.
    level = 0.99)
```

- Result:



```

        Welch Two Sample t-test

data:  Recommended_Customer_Price by Intel_Hyper_Threading_Technology_
t = -11.063, df = 1482.9, p-value < 2.2e-16
alternative hypothesis: true difference in means between group No and group Yes is not equal to 0
99 percent confidence interval:
 -563.5087 -350.4435
sample estimates:
 mean in group No mean in group Yes
         600.0893         1057.0653
```

5.6 One-way Analysis Of Variance (One-way ANOVA)

The first condition is that the samples must be independent. In this case, we can see that the samples taken from the file are independent.

The second condition is that the dependent variable must be continuous. We observe that in the `cpu_df.csv` file, after processing the data, the variables are continuous, so this condition is fully met.

The third condition is that the groups should be drawn from a normal or nearly normal distribution. To check this, we need to use the Shapiro-Wilk test, which requires the `nortest` library.
The hypotheses for this test are:

- H_0: The versions of all of the variables that we are considering excepts ones in the skip column(Production_Collection and Intel_Hyper_Threading_Technology_) follow a normal distribution.
- H_1: The versions of all of the variables that we are considering excepts ones in the skip column(Production_Collection and Intel_Hyper_Threading_Technology_) do not follow a normal distribution.
- Code:

```
skip_columns <- c('Product_Collection','Intel_Hyper_Threading_Technology_')

    for (col in names(cpu_df)) {
      if (!(col %in% skip_columns)) {
        test_result <- shapiro.test(cpu_df[[col]])
        cat("Shapiro-Wilk test for", col, "\n")
        print(test_result)
        cat("\n")
      }
    }
```

Observation: Since all p-values are mostly $= 2.2 \times 10^{-16}$ (the Processor_Base_Frequency's value is 1.7×10^{-12})($\ll 0.05$), we have sufficient grounds to reject H_0 and accept H_1, meaning that all the variables which are tested by Shapiro_Wilk, do not follow normal distributions. The fourth condition is that the groups must have equal variances. Use the `car` library and the `leveneTest()` command to check this with the following hypotheses:

- H_0: The variances of all of the variables like we did in the Shapiro_Wilk test as among the collections are the same.
- H_1: The variances of all of the variables like we did in the Shapiro_Wilk test as among the collections are different.
- Code:

```
skip_columns <- c('Product_Collection','Intel_Hyper_Threading_Technology_')
    for (col in names(cpu_df)) {
      if (!(col %in% skip_columns)) {
        test_result <- leveneTest(cpu_df[[col]] ~ as.factor(cpu_df $Product_Collection ))
        cat("leveneTest test for", col, "\n")
        print(test_result)
        cat("\n")
      }
    }
```

- Result:

Observation: Since most of the p-values $\approx 2.2 \times 10^{-16}$ ($\ll 0.05$) and most of them are really small compare to 0.05 we have sufficient grounds to reject H_0 and accept H_1, meaning that the most of variances of the variables versions across different manufacturers are different.
However, the p-value of Graphics_Base_Frequency impressively equals to 0.9781 which mean its variance are the same among the collections, H_0 is accepted, but as the conclusions above, even Graphics_Base_Frequency are not normally distributed.

Conclusion: All the variables are not met for ANOVA analysis because all of them failed the Shapiro_Wilk tests which all of them are not normally distributed and with the Levene test all of them among the collections, 17/18 variables failed, leave only Graphics_Base_Frequency succeeded but failed to meet the ANOVA analysis in the end.

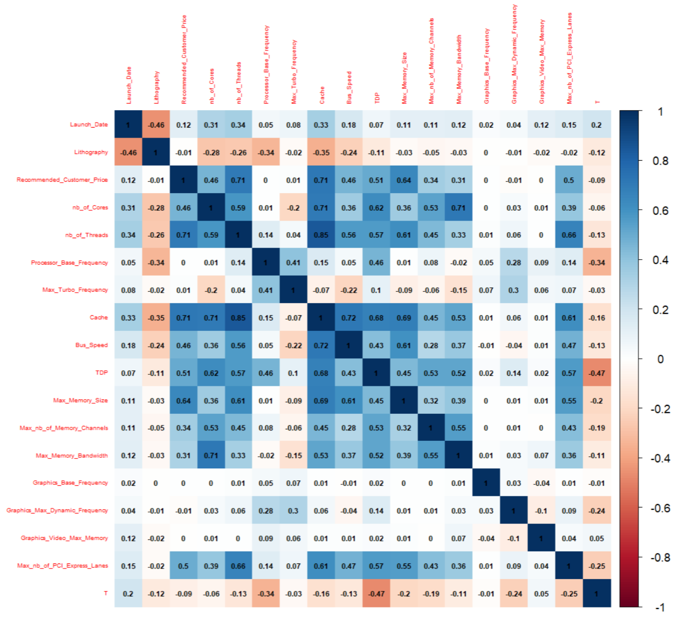
5.7 Linear regression

First, we draw a matrix where each cell represents the correlation between two variables.
Using `cor()` to calculate a matrix which its result is a symmetric matrix where each element represents the correlation coefficient between two variables and `corrplot()` to visualize the correlation matrix as a color-coded plot.

- Code:

```
    cor(subset(cpu_df, select = -c(Product_Collection,Intel_Hyper_Threading_Technology_)))
    corrplot(cor(subset(cpu_df, select = -c(Product_Collection,Intel_Hyper_Threading_
       Technology_))) ,
           number.cex = 0.5, tl.cex = 0.4,
           method = "color",
           addCoef.col = "black",
           type = "full")
```

- Result:



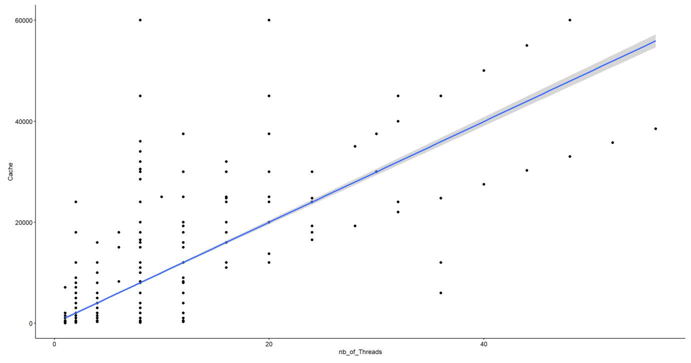
`Observation`: As the correlation number between Cache and nb_of_Threads is 0.85 and it was considered a strong connection between two variables.

The overall regression equation with one independent variable is as follows: $Y = \beta_0 + \beta_1 X + \epsilon$.
R code for the variable nb_of_Threads: Plot the regression graph using the `ggscatter()` command from the `ggpubr` library.

- Code:

```
ggscatter(data = cpu_df , y = "Cache" , x =  "nb_of_Threads") + geom_smooth(method = "lm" ,
    se = TRUE)
```

- Result:



Start building the regression model using the `lm()` command and print the results using the `summary()` command.

- Code:

```
modtest_1_factor <- lm(cpu_df $Cache ~ cpu_df $nb_of_Threads, data = cpu_df)
    summary(modtest_1_factor)
```

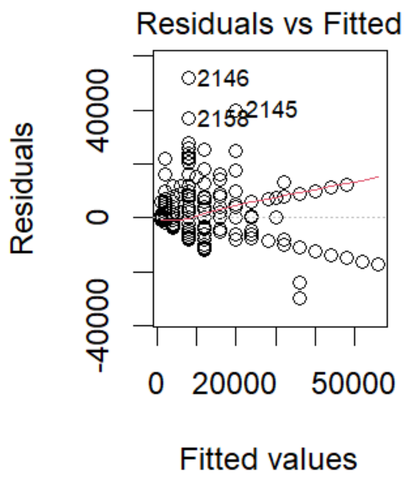
- Result:

```
Residuals:
   Min     1Q Median     3Q    Max 
-29965  -1795    -51    949  51964 

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)             56.40     138.58   0.407    0.684    
cpu_df$nb_of_Threads   997.47      13.15  75.867   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4925 on 2281 degrees of freedom
Multiple R-squared:  0.7162,	Adjusted R-squared:  0.7161 
F-statistic:  5756 on 1 and 2281 DF,  p-value: < 2.2e-16
```

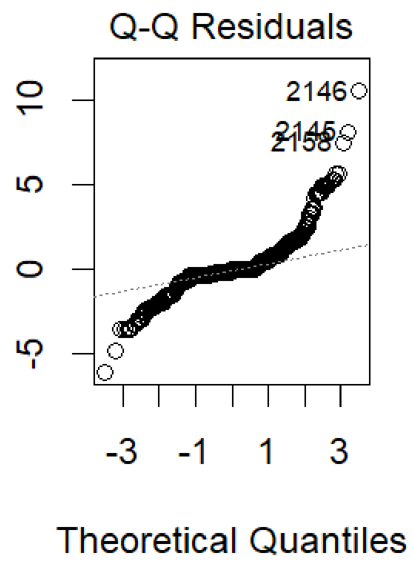
We obtain: The adjusted R^2 is 0.7162, $\beta_0 = 56.40$, $\beta_1 = 997.47$.
Thus, the linear regression equation based on the variable Cache is $Y = 56.40 + 997.47 \times$ nb_of_Threads.
Use the `plot()` command to check the normal distribution of the residuals.

- Code:

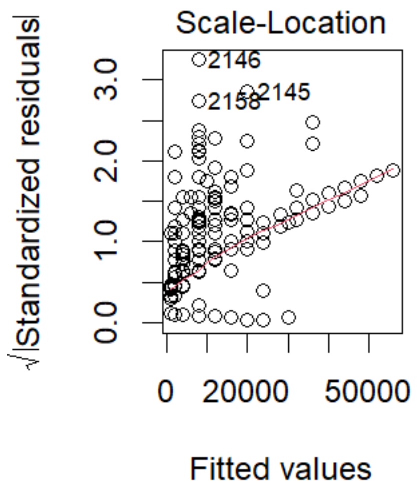
```
plot(modtest_1_factor)
```



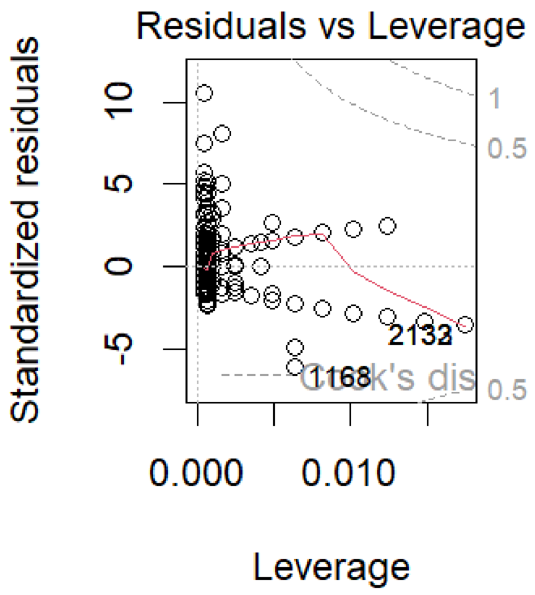
- The Residuals vs. Fitted plot provides insight into the linearity of the data. The red line, which represents the local regression of the residuals, smooths the scatter plot to reveal underlying trends. If this line approximates a straight path, it indicates that the data likely adheres to a linear relationship. In this instance, the red line remains largely straight, suggesting that the assumption of linearity is reasonably well satisfied.



- Observation: The Q-Q plot is used to assess whether the residuals follow a normal distribution. If the points align closely with the reference line, it suggests that the residuals are normally distributed. In this case, the majority of the residuals cluster along the straight line, with only minor deviations at the tails. These slight deviations are not substantial enough to undermine the assumption, so we can conclude that the normality of the residuals is reasonably upheld.



- Observation: The Scale-Location plot is instrumental in assessing the homogeneity of variance, or constant variance, of the residuals. This plot, which displays the square root of the standardized residuals against the fitted values, ideally shows a horizontal red line with residuals evenly scattered around it. However, in this case, the plot indicates that the assumption of constant variance is not satisfied, likely due to the presence of variance instability or unequal variance. The red line deviates from a horizontal path, and the residuals are unevenly distributed around it, suggesting variability in the spread of residuals across different levels of the fitted values.



- Observation: The Residuals vs. Leverage plot is used to identify influential observations within the dataset, which could potentially impact the analysis significantly. These influential points are often outliers that can disproportionately affect the results. In the plot, if we see a dashed red line representing Cook's distance, and no points exceed this line, it indicates that there are no influential observations present. In this instance, the plot shows that Cook's distance line is only present at the corners, and no points surpass this threshold, suggesting that there are no observations with high influence.

6 Code and References:

Information: `https://www.kaggle.com/datasets/iliassekkaf/computerparts`
Code: `https://github.com/khafu19768/btl_xxtk_24_R_source_Intel_cpus`