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

Course name h

Report type h

Report title h

Advisor: Advisor h

HO CHI MINH CITY, APRIL 2023

Contents

1 Abstract

2 Background

3 Data description

3.1 Overview of the dataset

Write something about the dataset, because no one will understand what a chip is if we don't elaborate them.

Some parts from Importing data section could be brought to here.

3.2 Importing data

For the first part of our project, we need to select a suitable dataset for us to analyze, as we are computer science students, we have decided to select a CPU data set, the data of which can be found here:
The data give us information about 2283 CPU and 45 of their feature which include:

- Product_Collection: tell us which type of series the core belongs to.
- Vertical_Segment: show what kind of system the CPU was designed for (embedded, mobile, desktop, or sever).
- Processor_Number : process ID.
- Status: show the status of the CPU (announce, launched, end of life, end of support).
- Launch_Date: The date the product was first introduced.
- Lithography: refers to the semiconductor technology used to manufacture an integrated circuit, and is reported in nanometers (nm), indicative of the size of features built on the semiconductor.
- Recommended_Customer_Price: recommended customer price.
- nb_of_Cores: total number of cores in a proccessor.
- nb_of_Threads: total number of thread in a processor.
- Processor_Base_Frequency: Describes the rate at which the processor's transistors open and close.
- Max_Turbo_Frequency: The maximum single core frequency at which the processor is capable of operating using Intel® Turbo Boost Technology.
- Cache: CPU Cache is an area of fast memory located on the processor.
- Bus_Speed: refers to how much data can move across the bus simultaneously.
- TDP(thermal design power): Represents the average power, in watts, the processor dissipates when operating at Base Frequency with all cores.
- Embedded_Options_Available: is it allow to be embedded system

- Conflict_Free: Defined by the U.S. Securities and Exchange Commission rules to mean products that do not contain conflict minerals (tin, tantalum, tungsten).
- Max_Memory_Size: The maximum memory capacity supported by the processor.
- Memory_Types: Single Channel, Dual Channel, Triple Channel, and Flex Mode.The maximum memory capacity supported by the processor.
- Max_nb_of_Memory_Channels: The number of memory channels refers to the bandwidth operation for real world application.
- Max_Memory_Bandwidth: The maximum rate at which data can be read from or stored into a semiconductor memory by the processor (in GB/s).
- ECC_Memory_Supported: ECC memory is a type of system memory that can detect and correct common kinds of internal data corruption.
- Processor_Graphics: integrated graphics processing unit (GPU) that is built into some of Intel's processors.
- Graphics_Base_Frequency: The rated/guaranteed graphics render clock frequency in MHz.
- Graphics_Max_Dynamic_Frequency: The maximum opportunistic graphics render clock frequency (in MHz) that can be supported using Intel HD Graphics with Frequency feature.
- Graphics_Video_Max_Memory: The maximum amount of memory accessible to processor graphics. Processor graphics operates on the same physical memory as the CPU (subject to OS, driver, and other system limitations).
- Graphics_Output: Graphics Output defines the interfaces available to communicate with display devices.
- Support_4k: indicates the product's support of 4K
- Max_Resolution_HDMI: the maximum resolution supported by the processor via the HDMI interface (24bits per pixel & 60Hz). System or device display resolution is dependent on multiple system design factors; actual resolution may be lower on your system.
- Max_Resolution_DP: The maximum resolution supported by the processor via the DP interface (24bits per pixel & 60Hz). System or device display resolution is dependent on multiple system design factors.
- Max_Resolution_eDP_Integrated_Flat_Panel
- DirectX_Support: Indicates support for a specific version of DirectX, a Microsoft collection of APIs for handling multimedia compute tasks.
- OpenGL_Support: Indicates support for OpenGL, a cross-language, multi-platform API for rendering 2D and 3D vector graphics.
- PCI_Express_Revision: The PCIe version supported by the processor.

- PCI_Express_Configurations_: The available PCIe lane configurations that can be used to link the PCH PCIe lanes to PCIe devices.
- T : The maximum temperature allowed on the chip.
- Max_nb_of_PCI_Express_Lanes: maximum number of PCI Express Lanes that are supported.
- Intel_Hyper_Threading_Technology_: Delivers two processing threads per physical core. Highly threaded applications can get more work done in parallel, completing tasks sooner.
- Intel_Virtualization_Technology_VTx_: Allows one hardware platform to function as multiple "virtual" platforms. It offers improved manageability by limiting downtime and maintaining productivity by isolating computing activities into separate partitions.
- Intel_64_: Delivers 64-bit computing on server, workstation, desktop and mobile platforms when combined with supporting software. Intel 64 architecture improves performance by allowing systems to address more than 4 GB of both virtual and physical memory.
- Instruction_Set: Which instrution set the CPU use.
- Instruction_Set_Extensions : Instruction set extension
- Idle_States: Used to save power when the processor is idle.
- Thermal_Monitoring_Technologies: Protects the processor package and the system from thermal failure through several thermal management features.
- Secure_Key: The CPU is supported with secure key or not.
- Execute_Disable_Bit: Hardware-based security feature that can reduce exposure to viruses and malicious code attacks.

For coding the data, our team use a wide range of package which include

- `rio`.
- `ggplot2`.
- `zoo`

Two primary packages used in this process are:

- `rio` : for intuitive I/O code. With this package, import and export dataset is easier and safer. It could also handle multiple file formats, so that we do not have to change the command each time we change the file format.
- `zoo` : for year-quarter format. In our data, the `Launch date` is in non-standard format, and difficult to be operated on. This package helps to transform into standard year-quarter format, and provides useful operations, such as plotting and taking difference on these formats.

```
pacman::p_load(
  rio,      # for imports & exports
  ggplot2, # for plots
  zoo       # for year-quarter formats
  )
```

for importing in the code, we use the function `import()` from rio to import our data set.

```
# Import data

data <- import("./cpu-raw.csv") # rio::import

```

MAY NEN NOI GI DO THEM, VI CAI NAY TAO CHUA VIET XONG!!!!
Viet tung do this.
You should describe how the features are chosen, why it is chosen, and stuff...
NK merge this preprocessing section with this section.

3.3 data preprocessing

After examining the code, as we only want to focus on the performance and trend of the cpu designing, as there are many unesscessary such as Product_Collection and Processor_Number which only tell us about series of cpu they are from, some give only information about cpu hardware support, and some feature which also has a lot of NA values which make it hard to anylyze the data.So our team has decided to cut back some feature and use only a couple of them, that inlcude:

Need: Vertical_Segment, Status, Launch_Date, Lithography, Recommended_Customer_Price, nb_of_Cores, nb_of_Threads, Processor_Base_Frequency, TDP, Max_Memory_Bandwidth, T.

The reason for the choice of these feature is that they are all give us a lot of information about the performance and effiecency. They are also often the most important factors in measuring the performance of each cpu.

```
#Data feature selection
data <- data[, c("Vertical_Segment","Status","Launch_Date",
                 "Lithography","Recommended_Customer_Price",
                 "nb_of_Cores","nb_of_Threads","Processor_Base_Frequency",
                 "TDP","Max_Memory_Bandwidth","T")]

```

Changing labels

The original labels are very long and descriptive, we might not want that such level of details during coding. Therefore, the labels are suppressed into small, compact abbreviations. The mapping of each label is summarized in the below table:

`Vertical segment -> market`	`Status -> status`	`Launch date -> ldate`
`Lithography -> litho`	`Recommended price -> rprice`	`Num. cores -> ncore`
`Base frequency -> bfreq`	`TDP -> tdp`	`Memory bandwidth -> memband`
`T -> temp`		

```
# Rename labels
names(data) <- c("market", "status", "ldate", "litho",
                 "rprice", "ncore", "nthread", "bfreq", "tdp",
                 "memband", "temp") # Rename columns, easier to deal with

```

3.4 Data cleaning

After choosing the approriate features, we now have the subset of the original raw dataset. However, since the values vary in types (such as string, non-standard year-quarter format and numeric-string), we might want transform them into reproducible types, so that the analysis later on is easier, homogeneous and accurate.

Note that this cleaning process **does not** remove the `NA` values, unless necessary. The reason is that, in one instance, there might be important values that should not be eliminated, under a specific scope of study, so we do not treat instances with `NA` as an invalid datum. In later sections, when we focus on a specific pattern of the data, only by then that the data will have a specialized `NA` cleaning, and we do not, by chance, loose any important instance.

This process took `/rcode/cpu-short.csv` from the importing procedure above as an input, and produce `/rcode/cpu-clean.csv` as an output.

ldate

`market` and `status` are left unchanged, since the values are straightforward. See Data clarification for further analysis.

The remaining features (columns) are processed as followed:

```
data[,"ldate"] <- (
  as.yearqtr(data[,"ldate"], format = "Q%q'%y")
)
```

Our goal is to transform raw, non-standard string representation of year-quarter into `zoo`'s standard representation. The function `as.yearqtr` takes a column and a format string as parameters. The format string is represented as: `"Q%q'%y"`, in which two flags `"%q"`, `"%y"` stands for quarter and year, respectively. The format string hints the function to know the positions of quarter and year in our raw string.

litho

```
data[,"litho"] <- as.numeric(
  gsub(" nm", "", data[,"litho"])
)
```

Our goal is to cut out `"nm"`, since every entry is recorded in nanometers anyway. `gsub` helps to replace a pattern of a string with another string. In this case, we replace any occurence of `"nm"` with an empty string, and then cast the remaining value to numeric type. Notice that the pattern could be regular expression as well, this will be used intensively in the following cleaning process.

rprice

```
data[,"rprice"] <- gsub(
  "(^\\$(\\d)+.(\\d)+ - )",
  "",
  data[,"rprice"]
  )
#preporcess prices - stage 2 : change all "N/A" to N/A
data$rprice <- ifelse(data$rprice == "N/A", NA, data$rprice)
# Preprocess prices - stage 3 : Cut out dollar sign '$ '
data$rprice <- as.numeric(gsub('\\$|,', '', data$rprice))
```

HEY VIET TUNG WRITE THIS FOR ME THANK YOU! but the pattern is a regular expression (regex). Please refer to Appendix for further information.

bfreq

```
data[,"bfreq"] <- as.numeric(
  gsub("( GHz)|( MHz)", "", data[,"bfreq"])
)
data<- data[!is.na(data$bfreq), ]    # Truncate NAs because subscripting with NA
                                     # is not allowed.
data$bfreq[data$bfreq > 10] <- data$bfreq[data$bfreq > 10]*0.001
```

Our goal is to cut out "GHz" and "MHz" from the string, and convert all "MHz" values into "GHz". Heuristically, we observed that any value greater than 10 must be MHz, so we can every value like that will be multiply by 0.001 to get the according MHz value. gsub is used like as above. This time, that regex is used to match any substring that is GHz OR MHz. Then, we multiply with the 0.001 any number greater than 10, and store it back to the dataframe. Notice that the NA values must be truncated first, as it is necessary to ensure dataframe subscripting to work.

tdp

```
data[,"tdp"] <- as.numeric(
  gsub(" W", "", data[,"tdp"])
)
```

Our goal is to cut out "W" from the string. The approach is similiar to the above.

memband

```
data[,"memband"] <- as.numeric(
  gsub(" GB/s", "", data[,"memband"])
)
```

Our goal is to cut out "GB/s" from the string. The approach is similiar to the above.

temp

```
data[,"temp"] <- (gsub("[^0-9.\\-]+", ",", data[,"temp"]))  #
for (i in seq_along(data[["temp"]])) {                      # For each elements in the
same entry
  temp_values <- strsplit(data[i, "temp"], ",")             # Split into a list of words
  temp_values <- unlist(lapply(temp_values, as.numeric))    # Transform them into
  equivalent numerics
  max_value <- max(temp_values, na.rm = TRUE)               # Find max
  if (is.infinite(max_value)) {                             # Is it an invalid numeric?
    max_value <- NA
  }
  data[i, "temp"] <- max_value                              # Store the maximum value
}
```

Our goal is to only match the numeric values, then, take the maximum among those, since we are only interested in the maximum temperature. This feature's data are very varying in forms and the pattern we want is difficult if only a simple regular expression is used to match. Our approach is can be described as follows:

- First, we attempt to match every decimal numbers possible, including the irrelevant number. The rest are replaced with commas `","`.

  The result of this process will create a string of numbers separated by commas. By doing this, the numbers are well isolated for our purpose.

- Second, we split these numbers and form a vector of them. This can be done through `strsplit` function. Notice that our numbers are still in string format.

- Third, we cast all these strings to numeric and push them into a vector of values using `unlist` and `lapply`

- Fourth, we find the maximum among all these values. Invalid numbers will automatically become $-\infty$, and will be further treated as `NA`.

Notice that, we must loop through each row of the list to accomplish the above procedure.

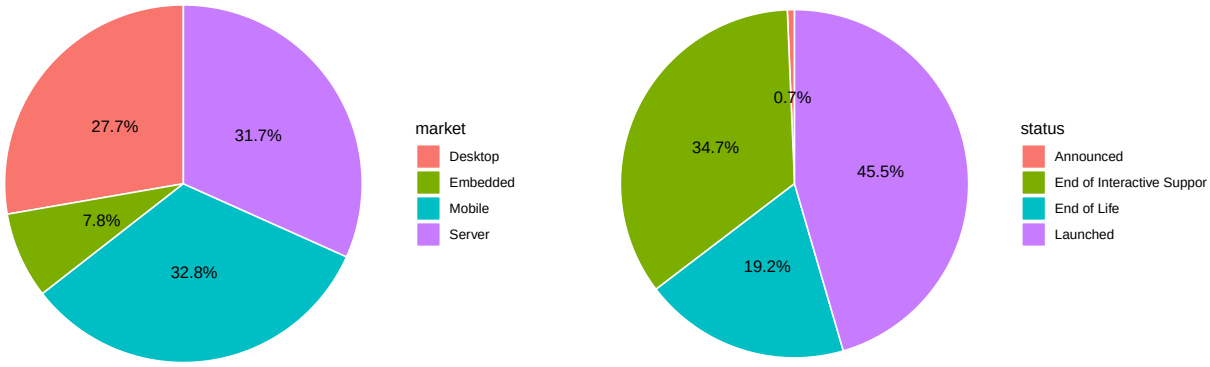
Finally, the program produces `cpu-clean.csv` as a cleaned data, ready for further exploitation in the later sections. This section uses a lot of regular expression to match the desired strings, please refer to Appendix for more details.

4 Data clarification

4.1 Overview of features

In this section, we provide a big picture of the our data via visualizations.

4.1.1 Categorical features



(a) *Pie of market share* **(b)** *Pie of support status*

4.1.2 Continuous features

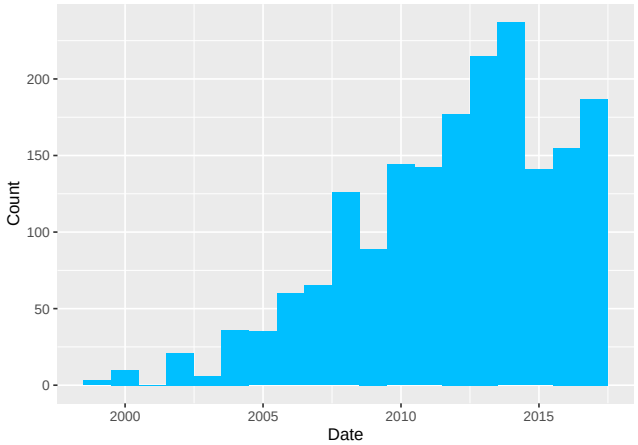
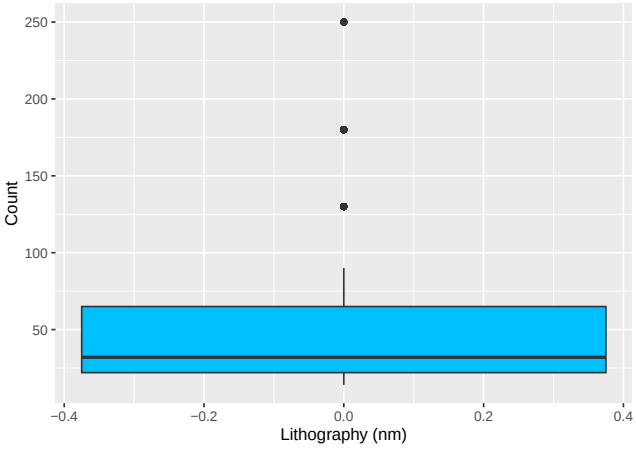


Figure 2: *Histogram of launch date*

Note that we skipped plotting three features: Recommended Price, Number of Cores, Number of Threads and Number of Maximum Memory Bandwidth since they are market-dependent and do not represent well for



Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
14.00	22.00	32.00	49.17	65.00	250.00	71

Figure 3: *Box plot and Summary of Lithography*

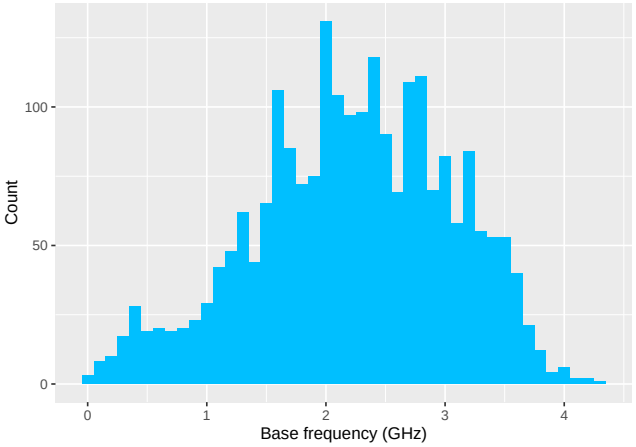
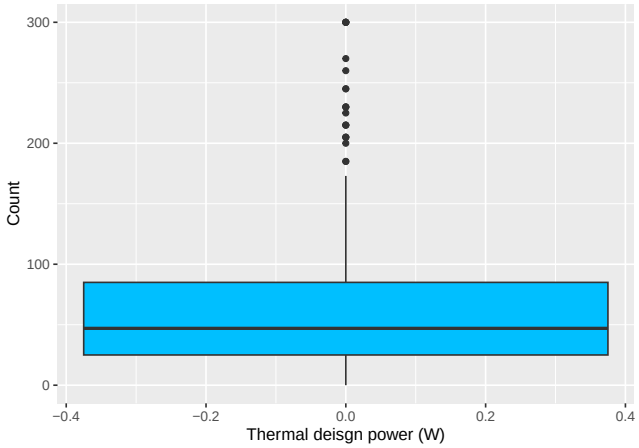


Figure 4: *Histogram of Base Frequency*

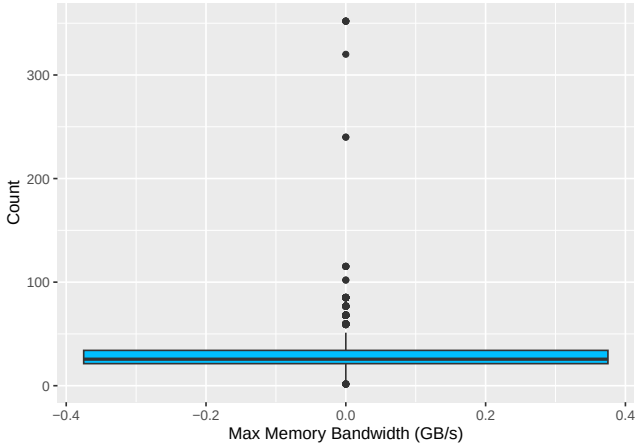
the overall trends.

4.2 Relationship between features




```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.025  25.000  47.000  60.242  85.000 300.000      49
```

Figure 5: *Histogram of Thermal Design Power (TDP)*




```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.60   21.30   25.60   35.51   34.10  352.00    1136
```

Figure 6: *Histogram of Maximum Memory Bandwidth*

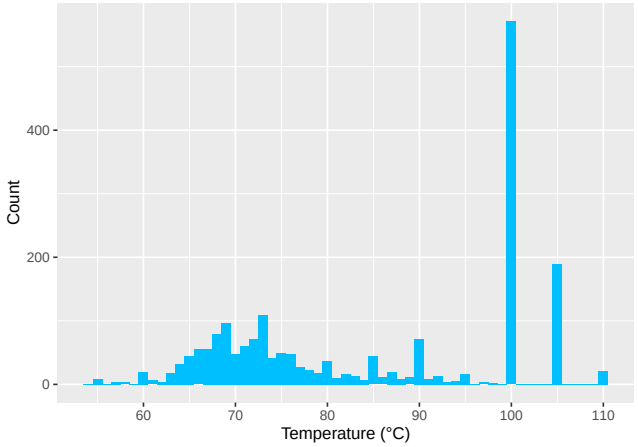


Figure 7: *Histogram of Temperature*

5 Data analysis

5.1 Analysis of Variance (ANOVA)

5.2 Regression

6 Conclusion

7 Appendix

Regular expression appendix