VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF APPLIED SCIENCE

PROBABILITY AND STATISTICS (MT2013)

Project topic:

IDENTIFICATION AND ANALYSIS OF FACTORS AFFECTING GPU'S PRICES & EVALUATION OF PRICES OF UNKNOWN GPUS

Group 03 - Class CC08 - Semester 232

Advisor: Dr. Nguyen Tien Dung

Students: Nguyen Chi Thuan - 2252787
Le Huu Tri - 2252841
Le Hoai Thien An - 2252006
Nguyen Trinh Tien Dat - 2252147
Nguyen Tuan Phong - 2252612

HO CHI MINH CITY, April 2024

MEMBER LIST & DUTY TABLE

No.	Full name	Student ID	Tasks	Contribution
1	Nguyen Chi Thuan	2252787	Summerize & edit files Compose section 8 & 9	100%
2	Le Huu Tri	2252841	Compose section 1 & 2	100%
3	Nguyen Tuan Phong	2252612	Compose section 4, 6 & 7	100%
4	Le Hoai Thien An	2252006	Compose section 5	100%
5	Nguyen Trinh Tien Dat	2252147	Compose R-Code & section 3	100%

TABLE OF CONTENTS

1 THEORETICAL FOUNDATION

1.1. Regression Model

1.1.1. Linear Regression Model

Multiple linear regression (MLR), also known as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression is to model the linear relationship between the explanatory (independent) variables and response (dependent) variables. The formula of multiple linear regression: $yi = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon$

where:

- y_i is the dependent variable.
- x_{ip} is the explanatory variable.
- β, is the y-intercept (constant term).
- β_p is the slope coefficient for each explanatory variable.
- ϵ is the model's error term (also known as the residuals).

When using linear regression, several assumptions are typically made.

- Assumption 1: **Linearity**, the relationship between the dependent variable and the independent variable is linear.
- Assumption 2: **Independence**, the observations are independent of each other

$Cov(\varepsilon_i, \varepsilon_j) = 0, i \neq j$

where $Cov(\varepsilon_i, \varepsilon_j)$ is the covariance between the errors for observations i and j.

- Assumption 3: **Homoscedasticity**, the variance of the errors is constant across all levels of the independent variable(s)

$Var(\varepsilon i) = \sigma^2 , \forall i$

where $Var(\varepsilon_i)$ is the variance of the error for observations i and σ^2 is a constant.

- Assumption 4: **Normality**, the errors are normally distributed

$\varepsilon \sim N(0, \sigma^2)$

where ε is the error term and $N(0, \sigma^2)$ denotes a normal distribution with mean 0 and variance σ^2.

1.1.2. Ordinary least squares

In statistics, **ordinary least squares (OLS)** is a type of linear least squares method for choosing the unknown parameters in a linear regression model (with fixed level-one effects of a linear function of a set of explanatory variables) by the principle of least squares: minimizing the sum of the squares of the differences between the observed dependent variable (values of the variable being observed) in the input dataset and the output of the (linear) function of the independent variable.

Simple linear regression considers a single regressor variable or predictor variable x and a dependent or response variable Y.

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

$E(Y|x) = \beta_0 + \beta_1 x$

where the intercept β_0 and the slope β_1 are unknown regression coefficients. We assume that each observation, Y, can be described by the model.

$Y = \beta_0 + \beta_1 x + \epsilon$

The fitted or estimated regression line is therefore

$\hat{Y}_i = \beta_0 + \beta_1 x_i$

Note that each pair of observations satisfies the relationship

$Y_i = \beta_0 + \beta_1 x_i + e_i, \qquad i = 1, \ldots, n$

where $e_i = Y_i - \hat{Y}_i$ is called the residual, which describes the error in the fit of the model to the i-th observation yi.

The least-squares estimates of the intercept and slope in the simple linear regression model are

$\beta_1 = \frac{Sxy}{Sxx}$

and $\beta_0 = \overline{y} - \beta_1 \overline{x}$,

where

$Sxx = \sum_{i=1}^{n}(xi - \overline{x})^2$

$Sxy = \sum_{i=1}^{n}(xi - \overline{x})(yi - \overline{y})$

$Syy = \sum_{i=1}^{n}(yi - \overline{y})^2$

The sum of squares of the residuals

$$SSE = \sum(y_i - \hat{y}_i)^2.$$

The total sum of squares of the response variable

$$SST = \sum(y_i - \overline{y})^2.$$

The sum of squares for regression

$$SSR = \sum(\hat{y}_i - \overline{y})^2.$$

Fundamental identity

$$SST = SSE + SSR$$

Coefficient of determination

$$r^2 = 1 - \frac{SSE}{SST}.$$

2 DATA INTRODUCTION

2.1. Dataset description

A **graphics processing unit** (**GPU**) is a specialized electronic circuit initially designed to accelerate computer graphics and image processing (either on a video card or embedded on motherboards, mobile phones, personal computers, workstations, and game consoles). After their initial design, GPUs were found to be useful for non-graphic calculations involving embarrassingly parallel problems due to their parallel structure. Other non-graphical uses include the training of neural networks and cryptocurrency mining.

The dataset is retrieved from Computer Parts (CPUs and GPUs) Dataset (Kaggle) by author Ilissek. In this project, we will analyze a dataset containing detailed technical specifications, release dates, and launch prices of over 3000 Graphics Processing Units (GPUs).

The input data is stored in a CSV file: gpus.csv for GPUs. The data table includes columns representing necessary information of the data, and some of the features will include:Boost_Clock, Core_Speed, Memory, Memory_Bandwidth, Memory_Bus, Memory_Speed, Release_Price and many others. Then, we have made a decision about creating a new dataset consisting of the required data.

2.2. Variables description

Variable	Data type	Unit	Description
Boost_Clock	Continuous	MHz	The processing speed of the graphics card after boost clock.
Core_Speed	Continuous	MHz	The processing speed of the graphics card.
Memory	Continuous	MB	The memory of the graphics card.
Memory_Bandwidth	Continuous	GB/sec	The speed at which a processor can read from or write to semiconductor memory. Memory bandwidth is typically expressed in bytes per second, although this may vary for systems with

			data sizes that are not multiples of the 8-bit bytes commonly used.
Memory_Bus	Continuous	Bit	A type of computer bus, typically in the form of a set of wires or conductors connecting electrical components, allowing the transmission of data and addresses from the main memory to the central processing unit (CPU) or memory controller.
Memory_Speed	Continuous	MHz	The memory speed of the GPU is measured in MHz, simply understood as the speed at which the GPU can access data stored in RAM.
Texture_Rate	Continuous	GTexel/s	The number of pixels the GPU can generate per second.
Release_Price	Continuous	$	The selling price of the GPU.

3 DATA PRE-PROCEEDING

3.1. Data reading (Importing and naming data)

The first line of this piece of code is importing file name ”gpus.csv” into data frame (noted as **data** in R). Then we choose the appropriate data that we want to analyze:

```
data<-read.csv("gpus.csv")
data <- data[, c("Boost_Clock","Core_Speed","Memory","Memory_Bandwidth",
                 "Memory_Bus","Memory_Speed","Release_Price","Texture_Rate")]
names(data)<-c("boost_clock", "core_speed",
               "mem", "mem_bandwidth", "mem_bus", "mem_speed", "release_price", "texture_rate")
```

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.

	boost_clock	core_speed	mem	mem_bandwidth	mem_bus	mem_speed	release_price	texture_rate
1		738 MHz	1024 MB	64GB/sec	256 Bit	1000 MHz		47 GTexel/s
2		-	512 MB	106GB/sec	512 Bit	828 MHz		12 GTexel/s
3		-	512 MB	51.2GB/sec	256 Bit	800 MHz		10 GTexel/s
4		-	256 MB	36.8GB/sec	128 Bit	1150 MHz		7 GTexel/s
5		-	256 MB	22.4GB/sec	128 Bit	700 MHz		6 GTexel/s
6		-	256 MB	35.2GB/sec	128 Bit	1100 MHz		6 GTexel/s
7		870 MHz	2048 MB	134.4GB/sec	256 Bit	1050 MHz		35 GTexel/s
8		-	256 MB	51.2GB/sec	256 Bit	800 MHz		7 GTexel/s
9		-	2048 MB	160GB/sec	256 Bit	1250 MHz		62 GTexel/s
10		-	64 MB	2.9GB/sec	64 Bit	366 MHz		
11		-	128 MB	5.2GB/sec	128 Bit	325 MHz		2 GTexel/s
12		650 MHz	6144 MB	177.6GB/sec	384 Bit	925 MHz		36 GTexel/s
13		705 MHz	5120 MB	168GB/sec	320 Bit	1050 MHz		
14		706 MHz	12288 MB	288.4GB/sec	384 Bit	1502 MHz		169 GTexel/s
15		-	64 MB	5.8GB/sec	128 Bit	360 MHz		
16	1100 MHz	1050 MHz	3072 MB	57.6GB/sec	128 Bit	900 MHz		62 GTexel/s
17		-	8192 MB	320GB/sec	512 Bit	1250 MHz		
18		732 MHz	6144 MB	249.6GB/sec	384 Bit	1300 MHz		
19	1000 MHz	300 MHz		34.1GB/sec	128 Bit	1067 MHz		8 GTexel/s

Figure 1: The initial data frame

```
> summary(data)
 boost_clock         core_speed            mem            mem_bandwidth        mem_bus            mem_speed
 Length:3406        Length:3406        Length:3406        Length:3406        Length:3406        Length:3406
 Class :character   Class :character   Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character
 release_price      texture_rate
 Length:3406        Length:3406
 Class :character   Class :character
 Mode  :character   Mode  :character
```

Figure 2: Summarize data before cleaning

3.2. Data cleaning

After choosing the approriate attributes, we now have the subset of the original raw dataset. However, since the values vary in types (such as string 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 different scopes of study, we can not treat instances with NA as an invalid datum for all scopes. In later sections, when we focus on a specific pattern of the data, only by then that the data will have a tailored NA cleaning, and we will not, by chance, loose any important instance.

a) Boost_Clock, Core_Speed, Memory, Memory_Bandwidth, Memory_Bus, Memory_Speed, Texture_Rate

```
data[,"boost_clock"] <- as.numeric(gsub("( MHz)", "", data[,"boost_clock"]))
#data <- data[!is.na(data$boost_clock), ]

data[,"core_speed"] <- as.numeric(gsub("( MHz)", "", data[,"core_speed"]))
#data <- data[!is.na(data$core_speed), ]

data[,"mem"] <- as.numeric(gsub("( MB)", "", data[,"mem"]))
#data <- data[!is.na(data$mem), ]

data[,"mem_bandwidth"] <- as.numeric(gsub("(GB/sec)", "", data[,"mem_bandwidth"]))
#data <- data[!is.na(data$mem_bandwidth), ]

data[,"mem_bus"] <- as.numeric(gsub("( Bit)", "", data[,"mem_bus"]))
#data <- data[!is.na(data$mem_bus), ]

data[,"mem_speed"] <- as.numeric(gsub("( MHz)", "", data[,"mem_speed"]))
#data <- data[!is.na(data$mem_speed), ]

data[,"texture_rate"] <- as.numeric(gsub("( GTexel/s)", "", data[,"texture_rate"]))
#data <- data[!is.na(data$texture_rate), ]
```

Our goal is to cut out the " MHz" in Boost_Clock column, since every entry is recorded in mega-hertz. Notice that the pattern are regular expressions, and would be used intensively during this cleaning process.

We do the same for the remain data.

b) Release_Price

```
data[,"release_price"] <- gsub("(^\\$(\\d)+.(\\d)+ - )", "", data[,"release_price"])
data$release_price <- ifelse(data$release_price == "N/A", NA, data$release_price)
data$release_price <- as.numeric(gsub('\\$|,', '', data$release_price))
```

We want to cut out uneccesary characters and only keep the price. After that, we eliminate $ symbol from the string, as well as cast the string to numeric type.

	boost_clock	core_speed	mem	mem_bandwidth	mem_bus	mem_speed	release_price	texture_rate
1	NA	738	1024	64.0	256	1000	NA	47
2	NA	NA	512	106.0	512	828	NA	12
3	NA	NA	512	51.2	256	800	NA	10
4	NA	NA	256	36.8	128	1150	NA	7
5	NA	NA	256	22.4	128	700	NA	6
6	NA	NA	256	35.2	128	1100	NA	6
7	NA	870	2048	134.4	256	1050	NA	35
8	NA	NA	256	51.2	256	800	NA	7
9	NA	NA	2048	160.0	256	1250	NA	62
10	NA	NA	64	2.9	64	366	NA	NA
11	NA	NA	128	5.2	128	325	NA	2
12	NA	650	6144	177.6	384	925	NA	36
13	NA	705	5120	168.0	320	1050	NA	NA
14	NA	706	12288	288.4	384	1502	NA	169
15	NA	NA	64	5.8	128	360	NA	NA
16	1100	1050	3072	57.6	128	900	NA	62
17	NA	NA	8192	320.0	512	1250	NA	NA
18	NA	732	6144	249.6	384	1300	NA	NA
19	1000	300	NA	34.1	128	1067	NA	8
20	NA	NA	256	23.4	256	730	NA	3
21	NA	575	6144	144.0	384	750	NA	NA
22	NA	575	6144	144.0	384	750	NA	NA

Figure 3: Data after cleaning

```
> summary(data)
  boost_clock      core_speed          mem         mem_bandwidth       mem_bus          mem_speed     release_price
 Min.   : 400   Min.   : 100.0   Min.   :   16   Min.   :   1.0   Min.   :  32.0   Min.   : 100   Min.   :   23.0
 1st Qu.:1050   1st Qu.: 790.0   1st Qu.: 1024   1st Qu.:  28.8   1st Qu.: 128.0   1st Qu.: 800   1st Qu.:  159.8
 Median :1176   Median : 980.0   Median : 2048   Median : 105.8   Median : 128.0   Median :1150   Median :  240.0
 Mean   :1206   Mean   : 946.9   Mean   : 2873   Mean   : 137.4   Mean   : 205.4   Mean   :1176   Mean   :  371.6
 3rd Qu.:1317   3rd Qu.:1090.0   3rd Qu.: 4096   3rd Qu.: 194.5   3rd Qu.: 256.0   3rd Qu.:1502   3rd Qu.:  421.5
 Max.   :1936   Max.   :1784.0   Max.   :32000   Max.   :1280.0   Max.   :8192.0   Max.   :2127   Max.   :14999.0
 NA's   :1960   NA's   :936      NA's   :420     NA's   :125      NA's   :62       NA's   :105    NA's   :2850
  texture_rate
 Min.   :  0.00
 1st Qu.: 22.00
 Median : 60.00
 Mean   : 90.27
 3rd Qu.:135.00
 Max.   :717.00
 NA's   :544
```

Figure 4: Summarize data after cleaning

It can be seen that there are 1960 over 3406 NA's in boost_clock (more than 50%) so we decide to remove that variable in order not to affect out analysis. For other variable, we decide to used the mean value of the known value to replace the NA value:

```
data <- data[, -which(names(data) == "boost_clock")] #delete boost_clock col
data$core_speed[is.na(data$core_speed)] <- mean(data$core_speed, trim = 0, na.rm = TRUE) #change NA to mean value
data$mem[is.na(data$mem)] <- mean(data$mem, trim = 0, na.rm = TRUE)
data$mem_bandwidth[is.na(data$mem_bandwidth)] <- mean(data$mem_bandwidth, trim = 0, na.rm = TRUE)
data$mem_bus[is.na(data$mem_bus)] <- mean(data$mem_bus, trim = 0, na.rm = TRUE)
data$mem_speed[is.na(data$mem_speed)] <- mean(data$mem_speed, trim = 0, na.rm = TRUE)
data$texture_rate[is.na(data$texture_rate)] <- mean(data$texture_rate, trim = 0, na.rm = TRUE)
```

Figure 5: Data after checking missing value

After that, we split out data into 2 frames base on the release_price. We choose the row which the price is known for the first data frame (noted **data**) and the other data frame includes NA price (noted **data_NA**).

	core_speed	mem	mem_bandwidth	mem_bus	mem_speed	release_price	texture_rate
1	738.0000	1024.000	64.0000	256.0000	1000.000	NA	47.00000
2	946.8939	512.000	106.0000	512.0000	828.000	NA	12.00000
3	946.8939	512.000	51.2000	256.0000	800.000	NA	10.00000
4	946.8939	256.000	36.8000	128.0000	1150.000	NA	7.00000
5	946.8939	256.000	22.4000	128.0000	700.000	NA	6.00000
6	946.8939	256.000	35.2000	128.0000	1100.000	NA	6.00000
7	870.0000	2048.000	134.4000	256.0000	1050.000	NA	35.00000
8	946.8939	256.000	51.2000	256.0000	800.000	NA	7.00000
9	946.8939	2048.000	160.0000	256.0000	1250.000	NA	62.00000
10	946.8939	64.000	2.9000	64.0000	366.000	NA	90.27463
11	946.8939	128.000	5.2000	128.0000	325.000	NA	2.00000
12	650.0000	6144.000	177.6000	384.0000	925.000	NA	36.00000
13	705.0000	5120.000	168.0000	320.0000	1050.000	NA	90.27463
14	706.0000	12288.000	288.4000	384.0000	1502.000	NA	169.00000
15	946.8939	64.000	5.8000	128.0000	360.000	NA	90.27463
16	1050.0000	3072.000	57.6000	128.0000	900.000	NA	62.00000
17	946.8939	8192.000	320.0000	512.0000	1250.000	NA	90.27463
18	732.0000	6144.000	249.6000	384.0000	1300.000	NA	90.27463

```r
data_NA <- data[is.na(data$release_price), ]
View(data_NA)

data <- data[!is.na(data$release_price), ]
```

	core_speed	mem	mem_bandwidth	mem_bus	mem_speed	release_price	texture_rate
43	946.8939	2872.769	480.0000	384.0000	1250	1199.00	343
46	946.8939	12288.000	547.2000	96.0000	1425	1199.00	354
48	705.0000	12288.000	672.0000	384.0000	1750	2999.00	420
50	705.0000	12288.000	672.0000	384.0000	1750	2999.00	420
52	1140.0000	12288.000	336.6000	384.0000	1753	1099.00	240
53	1127.0000	24576.000	673.2000	384.0000	1753	2059.00	467
55	1000.0000	24576.000	673.2000	384.0000	1753	1998.00	418
56	1000.0000	12288.000	336.6000	384.0000	1753	999.00	209
57	1127.0000	12288.000	336.6000	384.0000	1753	1029.99	233
60	1000.0000	12288.000	384.0000	3072.0000	500	999.00	320
62	1000.0000	16384.000	512.0000	4096.0000	500	1299.00	320
66	889.0000	6144.000	336.0000	384.0000	1750	999.00	235
69	837.0000	6144.000	288.4000	384.0000	1502	999.00	196
171	960.0000	1024.000	134.4000	256.0000	1050	249.00	38
173	850.0000	1024.000	124.8000	256.0000	975	249.00	34
184	575.0000	512.000	57.6000	256.0000	900	130.00	18
187	750.0000	512.000	51.2000	128.0000	800	109.00	24
203	750.0000	512.000	25.6000	128.0000	800	67.00	24
934	946.8939	8192.000	256.0000	512.0000	500	499.00	384
935	946.8939	8192.000	256.0000	512.0000	500	599.00	410

Showing 1 to 20 of 556 entries, 7 total columns

	core_speed	mem	mem_bandwidth	mem_bus	mem_speed	release_price	texture_rate
1	738.0000	1024.000	64.0000	256.0000	1000.000	NA	47.00000
2	946.8939	512.000	106.0000	512.0000	828.000	NA	12.00000
3	946.8939	512.000	51.2000	256.0000	800.000	NA	10.00000
4	946.8939	256.000	36.8000	128.0000	1150.000	NA	7.00000
5	946.8939	256.000	22.4000	128.0000	700.000	NA	6.00000
6	946.8939	256.000	35.2000	128.0000	1100.000	NA	6.00000
7	870.0000	2048.000	134.4000	256.0000	1050.000	NA	35.00000
8	946.8939	256.000	51.2000	256.0000	800.000	NA	7.00000
9	946.8939	2048.000	160.0000	256.0000	1250.000	NA	62.00000
10	946.8939	64.000	2.9000	64.0000	366.000	NA	90.27463
11	946.8939	128.000	5.2000	128.0000	325.000	NA	2.00000
12	650.0000	6144.000	177.6000	384.0000	925.000	NA	36.00000
13	705.0000	5120.000	168.0000	320.0000	1050.000	NA	90.27463
14	706.0000	12288.000	288.4000	384.0000	1502.000	NA	169.00000
15	946.8939	64.000	5.8000	128.0000	360.000	NA	90.27463
16	1050.0000	3072.000	57.6000	128.0000	900.000	NA	62.00000
17	946.8939	8192.000	320.0000	512.0000	1250.000	NA	90.27463
18	732.0000	6144.000	249.6000	384.0000	1300.000	NA	90.27463
19	300.0000	2872.769	34.1000	128.0000	1067.000	NA	8.00000
20	946.8939	256.000	23.4000	256.0000	730.000	NA	3.00000

Showing 1 to 20 of 2,850 entries, 7 total columns

Figure 6: Data frame (556 entries) and data_NA frame (2850 entries)

```r
#data summary by xtabs
xtabs(~core_speed,data=data)
xtabs(~mem,data=data)
xtabs(~mem_bandwidth,data=data)
xtabs(~mem_bus,data=data)
xtabs(~mem_speed,data=data)
xtabs(~texture_rate,data=data)
xtabs(~release_price,data=data)
```

```
> summary(data)
   core_speed          mem          mem_bandwidth        mem_bus          mem_speed      release_price       texture_rate
 Min.   : 550.0   Min.   :  512   Min.   :  12.8   Min.   :  64   Min.   : 500   Min.   :   23.0   Min.   :  5.0
 1st Qu.: 946.9   1st Qu.: 2048   1st Qu.: 112.1   1st Qu.: 128   1st Qu.:1375   1st Qu.:  159.8   1st Qu.: 68.0
 Median :1024.5   Median : 4096   Median : 192.3   Median : 256   Median :1750   Median :  240.0   Median :128.0
 Mean   :1080.0   Mean   : 4777   Mean   : 217.3   Mean   : 292   Mean   :1574   Mean   :  371.6   Mean   :141.9
 3rd Qu.:1175.0   3rd Qu.: 8192   3rd Qu.: 256.0   3rd Qu.: 256   3rd Qu.:1753   3rd Qu.:  421.5   3rd Qu.:193.2
 Max.   :1784.0   Max.   :32000   Max.   :1024.0   Max.   :8192   Max.   :2127   Max.   :14999.0   Max.   :555.0
```

Figure 7: Data summary after that

```
> xtabs(~core_speed,data=data)
core_speed
             550              574              575              576              600              602              607 
               1                1                1                3                4                1                2 
             625              633              648              650              675              700              701 
               3                1                2                3                3                2                1 
             705              720              725              730              732              738              750 
               2                2                2                3                1                2                3 
             772              778              783              797              800              810              822 
               1                1                1                2                7                7                1 
             827              837              850              855              860              863              870 
               1                1                6                1                2                3                1 
             875              876              880              889              900              902              910 
               1                1                2                1                5                4                1 
             915              918              925              926              928 946.893927125506              947 
               5                1                6               17                1               79                1 
             950              960              965              967              970              975              980 
               4                1                1                1                5                1                8 
             985              990              993             1000             1006             1010             1018 
               4                2                2               36                1                2                1 
            1020             1024             1025             1030             1032             1033             1040 
               3                3                2                2                2                1                1 
            1046             1050             1051             1058             1060             1076             1088 
               2               13                1                1                2                1                1 
            1090             1100             1102             1106             1120             1121             1126 
              12                4                4                1               42                1                5 
            1127             1140             1143             1150             1152             1165             1175 
               6                5                1                1                1                3               29 
            1178             1180             1190             1200             1203             1208             1216 
```

Figure 7.1: Summary of core speed

```
> xtabs(~mem,data=data)
mem
             512              768              896             1024             1280             1536             1792 
              11                1                3               45                3                3                1 
            2048 2872.76892163429             3072             4096             6144             8192            11264 
             117                8               32              142               43              119                8 
           12288            16384            24576            32000 
              10                7                2                1 
```

Figure 7.2: Summary of memory

```
> xtabs(~mem_bandwidth,data=data)
mem_bandwidth
            12.8             14.4               16             25.6             28.5             28.8               29 
               1                4                1                3                1                6                1 
            38.4               40             40.1             41.6             51.2             54.4             57.6 
               2                2                1                2                3                1                5 
            57.7               64             70.4               72             73.6               80             86.4 
               1                1                3                7                3                3                3 
            89.9               96             98.4            102.7              104            105.8            108.8 
               1                5                1                1                4                8                1 
           111.9              112            112.1            112.2            112.3            115.2            124.8 
               2               52               24               11                1                3                2 
             127              128            128.1            128.3            128.6            130.6            133.9 
               1                2                1                3                1                1                1 
           134.4 137.350807680585            140.8            141.7            144.2              152            153.6 
               3                6                1                1                4                1                4 
             159            163.4              176            177.4            179.2            182.4            185.6 
               1                1                8                1               11                8                3 
           188.8              192            192.2            192.3            192.4            194.6            196.8 
               1                4               37                6                1                2                2 
             197              208            211.2            211.6            223.8              224            224.3 
               2                1               12                1                1               35                1 
           224.4            230.4              240              256            256.3            259.5            262.7 
              25                1                4               58               18                1                1 
             264              272            272.3            281.6              288            288.4              320 
               1                2                3                1                3                4                4 
           320.3            323.3              336            336.6            340.6            340.8              352 
              13                1                2               18                1                1                1 
           354.8            358.4              384            390.4            448.8              480            484.4 
```

Figure 7.3: Summary of memory bandwidth

```
> xtabs(~mem_bus,data=data)
mem_bus
              64               88               96              128              176              192 205.356459330144
              10                5                1              186                1               53                5
             256              320              352              384              448              512             3072
             209                2                2               46                4               23                1
            4096             8192
               7                1
```

Figure 7.4: Summary of memory bus

```
> xtabs(~mem_speed,data=data)
mem_speed
 500  650  702  750  800  802  837  850  891  900  902  905  924  950  975  999 1000 1001 1002 1005 1020 1025 1050 1100
  14    2    1    1    9    1    1    2    1   18    1    1    2    1    2    3    3    3    4    1    1    3    3    5
1107 1125 1134 1150 1200 1242 1250 1251 1252 1263 1350 1375 1376 1400 1425 1426 1450 1475 1500 1502 1525 1625 1650 1653
   1    7    1    3    4    1   15   14    1    1    3   10    6   13    9    2    3    1   24   15    2    5   12    9
1702 1750 1752 1753 1755 1774 1775 1800 2000 2002 2027 2052 2125 2127
   1   95   25   63    1    1    1    2   57   54    3    3    2    3
```

Figure 7.5: Summary of memory speed

```
> xtabs(~texture_rate,data=data)
texture_rate
  5   6  11  13  14  15  16  17  18  19  20  21  24  25  26  27  29  32  34  35  36  37  38  40  41  42  43  44  45  47
  2   1   1   5   3   2   3   3   2   4   2   1   2   5   2   1   3   2   6   1   1   1   4   1   1   1   2   2   2   2
 48  49  50  51  52  53  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  77  78  80
  5   2   2   2   4   2   1   1   2  10   8   1   3   3   1   2   4   2  12   9   3   4   2   3   1   1   3   3   3   4
 82  83  84  86  87  89  90  91  92  94  96  99 101 103 104 105 108 109 110 111 112 113 114 118 121 123 124 125 126 127
 20  10  10   2   2   1   1   1   1   1   1   1   1   1   3   1   1   1  10   1   2   2   1   3   1  12   3   4   2   5
128 129 130 132 133 134 135 137 139 140 141 142 143 145 148 151 152 154 155 156 157 159 160 161 162 163 164 169 170 172
  6   1   1   3   1   3   1   8   2   2   2   4   4   1   3   1   1   2   1   7   1   4   3   4   2   2   3   1   2   1
173 175 176 178 179 182 184 185 186 188 189 191 192 193 194 195 196 197 199 200 201 202 203 205 206 207 208 209 211 213
  2   1   1   3   1  24   2   6   3   3   7   1   2   8   3   2   2   8   4   2   3  12   2   3   2   3   1   6   1   2
214 216 217 220 223 224 225 227 228 230 233 235 240 245 249 256 261 269 277 283 284 294 296 298 310 311 320 327 328 333
  1   4   1   3   1   2   3   2   1   1   1   1   1   1   1   3   1   2  10   1   1   1   2   1   1   1   4   1   1   1
337 343 346 354 358 365 370 377 379 384 396 410 418 420 467 512 538 555
  1   1   1   7   1   1   1   2   2   2   1   1   1   2   1   1   1   1
```

Figure 7.6: Summary of texture rate

```
> xtabs(~release_price,data=data)
release_price
     23      49      55      59      67      69      79      80      89      99   99.99     109  109.99     110     114
      1       1       1       3       1       6       4       1       7       9       1       9       6       1       1
    115     119  119.99     120     127     129  129.99     130     132     135     139  139.99     140     145     149
      1       1       5      18       1       2       7      10       1       3       1       2       5       1      12
 149.99     150     154     159     160     165     169     170     174     179     180     184     185     199  199.99
      2       9       2       4       6       1       5       1       2       7       1       1      10      22       3
    200     203     206     209  209.99     210     214     215     218     219  219.99     220     221     224     229
     28       1       1       3       3       3       1       1       1       2       2       8       2       1       9
 229.99     230     235  235.02     239  239.99     240  243.44     245     249  249.99     250     255  259.99     260
      3       2       1       1       4       1      11       1       3      10       6      15       1       2       2
    265     269  269.99     270     278     279  279.99     289  289.99     292     298     299  299.99     300  300.64
      1       2       1       3       1       5       1       1       4       1       3       9       3       1       1
    318  319.49     320     329  329.99     330     339  339.99     349  349.99     358     359  359.99     369  369.99
      2       1       2       7       4       1       1       1       6       3       1       1       1       2       1
    370     379     380     398     399     400     419     429     430     438     439     449  449.99     450     458
      1       4       1       3       6       1       3       2       1       2       1       7       1       4       1
    480     485     499     500     549  549.99     559     560  569.99  579.99     585     590     599     609     619
      1       2       6       1       3       4       1       1       3       3       2       2       4       1       1
    649  649.99     658  659.98     669     679     690     699  699.99     719     749     779     799  799.99     829
     14       1       2       1       3       3       1      17       3       1       2       1       1       2       1
    858     988     999  999.99 1029.99    1099 1099.98    1158 1159.98    1199    1249    1298    1299    1499    1998
      1       1       8       1       1       1       1       1       1       4       1       2       2       2       1
   2059    2999   14999
      1       2       1
```

Figure 7.7: Summary of release price

4 DESCRIPTIVE STATISTICS

4.1. GPU prices in the market through a histogram table

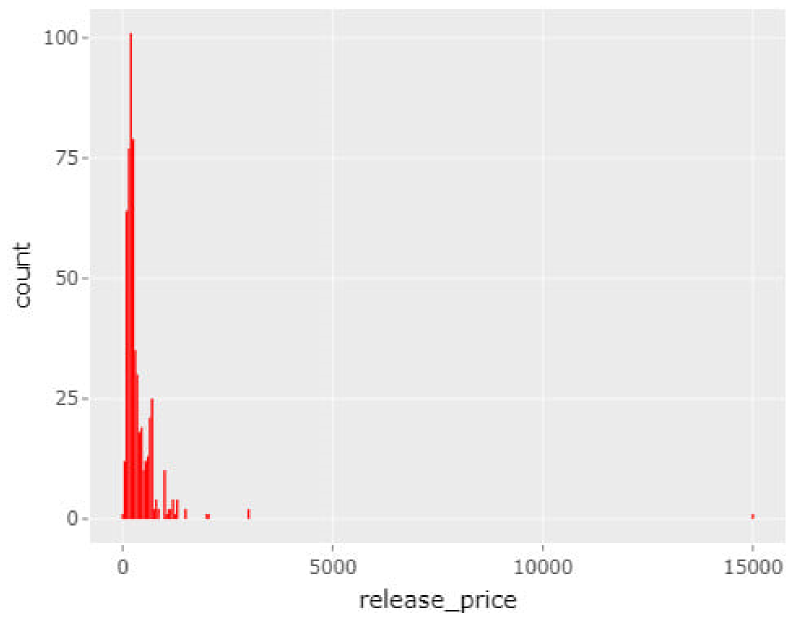

Figure 8: Histogram table for GPU release price

Remark: From this histogram chart, we can see there is only one GPU that approximately has the cost about 15,000$. Other than that, most of the release price is about 0$ to 1000$, and the mode is in the range from 40$ to 100$.

4.2. Scatter plots for the relationship between factors influencing GPU prices

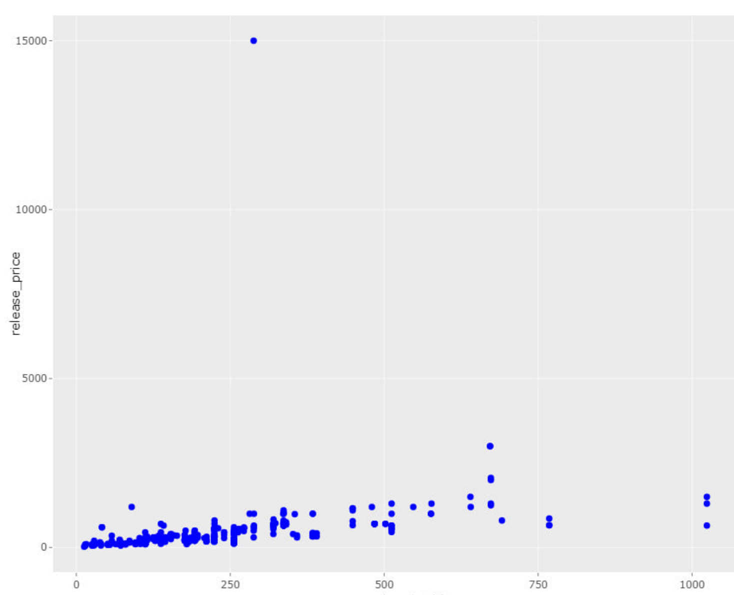

Figure 9: Scatter plot for relationship between memory bandwidth and release price

Remark: As the memory bandwidth is in the range from 0 to 1200 bytes/sec, mostly memory bandwidth in the market will have the value from 10 bytes/sec to 300 bytes/sec, and the value is in the range from 10$ to 1000$. Moreover, this graph can use linear regression to predict the cost if we are given the value of memory bandwidth.

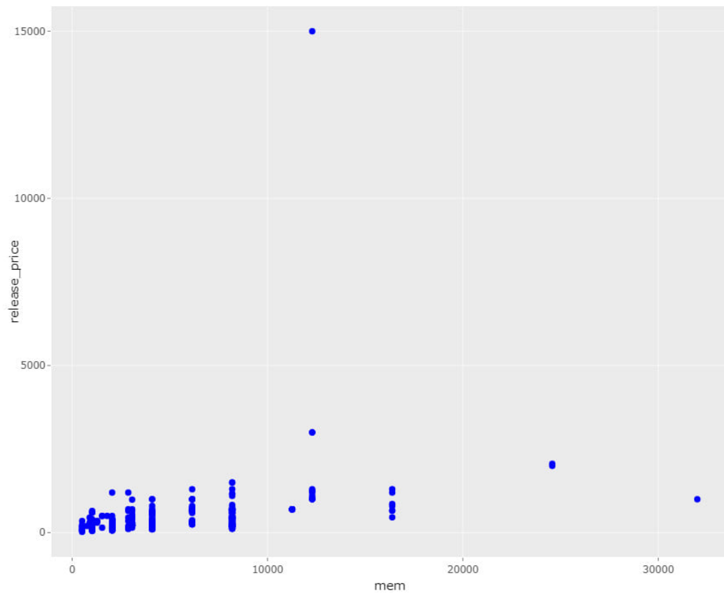

Figure 10: Scatter plot for relationship between memory and release price

Remark: From the scatter plot above, we can see that in the market there are only available some specific sizes of memory. Moreover, most of the price of GPUs is in the range from 10$ to 1000$, too.

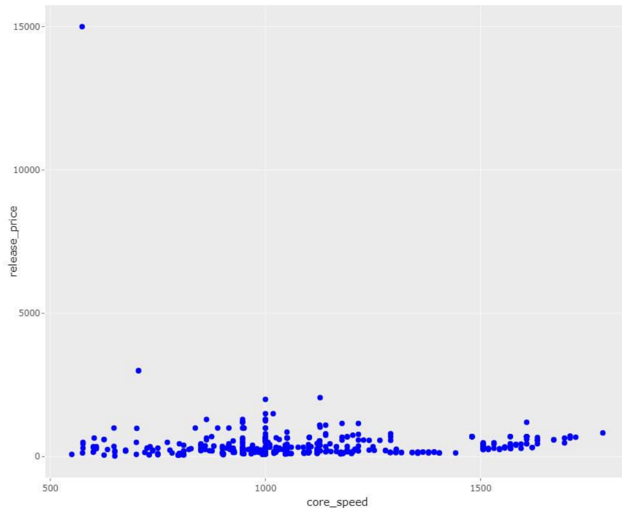

Figure 11: Scatter plot for relationship between core speed and release price

Remark: We can see that the relationship between core speed and release price is unable to be clearly demonstrated, as the dots are randomly distributed.

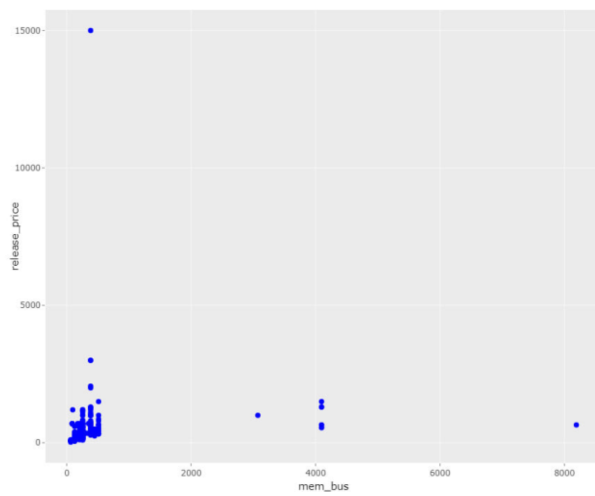

Figure 12: Scatter plot for relationship between memory bus and release price

Remark: Same with memory, memory bus is available in the market with only some specific sizes. From the table above, the size of memory bus is from 16 Bit to 512 Bit or 4096 Bit; and the cost mostly would be lower than 1000$.

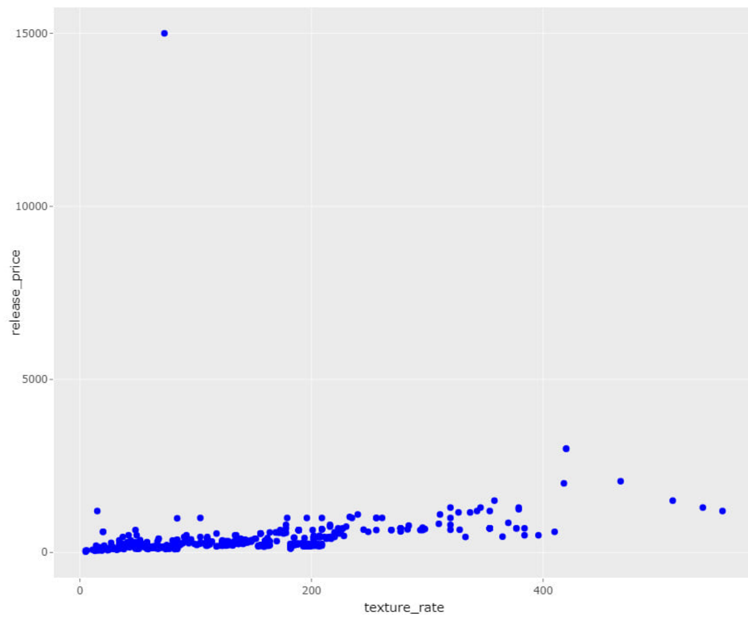

Figure 13: Scatter plot for relationship between texture rate and release price

Remark: As the texture rate is in the range from 0 to 600 GTexel/sec, mostly texture rate in the market will have the value from 10 GTexel/sec to 400 GTexel/sec. Moreover, this graph can use linear regression to predict the cost if we are given the value of texture rate.

5 INFERENTIAL STATISTICS

5.1. Introduction

Our task is to build a model for the factors influencing the variable Release_Price, where the variable we are interested in ("Release_Price") is the dependent variable and the remaining variables are considered independent variables.

$$\text{Release\_Price} = \boldsymbol{b_0} + \boldsymbol{b_1} \times \text{Core\_Speed} + \boldsymbol{b_2} \times \text{Memory} + \boldsymbol{b_3} \times \text{Memory\_Bandwidth} + \boldsymbol{b_4} \times \text{Memory\_Bus} + \boldsymbol{b_5} \times \text{Memory\_Speed} + \boldsymbol{b_6} \times \text{Texture\_Rate} + \boldsymbol{\varepsilon}$$

Before constructing the model, we begin by splitting the dataset into 2 new datasets according to the variable Release_Price, as following:

The first set which contains 556 observations with known Release_Price is used to find the formula of the variable of interest Release_Price based on related factors.

The second which contains the 2850 other observations with known Release_Price is used to predict the Release_Price.

5.2. Processing the data

After splitting the dataset, our task is to build a suitable model to describe the factors affecting our targeted variable Release_Price.

We statistically analyze values with the quantity of values, as following:

```
release_price
     23      49      55      59      67      69      79      80      89      99   99.99     109  109.99     110     114
      1       1       1       3       1       6       4       1       7       9       1       9       6       1       1
    115     119  119.99     120     127     129  129.99     130     132     135     139  139.99     140     145     149
      1       1       5      18       1       2       7      10       1       3       1       2       5       1      12
 149.99     150     154     159     160     165     169     170     174     179     180     184     185     199  199.99
      2       9       2       4       6       1       5       1       2       7       1       1      10      22       3
    200     203     206     209  209.99     210     214     215     218     219  219.99     220     221     224     229
     28       1       1       3       3       3       1       1       1       2       2       8       2       1       9
 229.99     230     235  235.02     239  239.99     240  243.44     245     249  249.99     250     255  259.99     260
      3       2       1       1       4       1      11       1       3      10       6      15       1       2       2
    265     269  269.99     270     278     279  279.99     289  289.99     292     298     299  299.99     300  300.64
      1       2       1       3       1       5       1       1       4       1       3       9       3       1       1
    318  319.49     320     329  329.99     330     339  339.99     349  349.99     358     359  359.99     369  369.99
      2       1       2       7       4       1       1       1       6       3       1       1       1       2       1
    370     379     380     398     399     400     419     429     430     438     439     449  449.99     450     458
      1       4       1       3       6       1       3       2       1       2       1       7       1       4       1
    480     485     499     500     549  549.99     559     560  569.99  579.99     585     590     599     609     619
      1       2       6       1       3       4       1       1       3       3       2       2       4       1       1
    649  649.99     658  659.98     669     679     690     699  699.99     719     749     779     799  799.99     829
     14       1       2       1       3       3       1      17       3       1       2       1       1       2       1
    858     988     999  999.99 1029.99    1099 1099.98    1158 1159.98    1199    1249    1298    1299    1499    1998
      1       1       8       1       1       1       1       1       1       4       1       2       2       2       1
   2059    2999   14999
      1       2       1
```

Figure 14

After that, we proceed to filter out outlier values by using STD method. (*R-Code below*)

```
mean_data <- mean(data$release_price)
std_data <- sd(data$release_price)
limit = 3*std_data
lower = mean_data - limit
upper = mean_data + limit
data = data[data$release_price < upper & data$release_price > lower, ]
View(data)
```

Then we only have the dataset of 553 observations (2999 and 14999 are been eliminated)

	core_speed	mem	mem_bandwidth	mem_bus	mem_speed	release_price	texture_rate
43	946.8939	2872.769	480.0000	384.0000	1250	1199.00	343
46	946.8939	12288.000	547.2000	96.0000	1425	1199.00	354
52	1140.0000	12288.000	336.6000	384.0000	1753	1099.00	240
53	1127.0000	24576.000	673.2000	384.0000	1753	2059.00	467
55	1000.0000	24576.000	673.2000	384.0000	1753	1998.00	418
56	1000.0000	12288.000	336.6000	384.0000	1753	999.00	209
57	1127.0000	12288.000	336.6000	384.0000	1753	1029.99	233
60	1000.0000	12288.000	384.0000	3072.0000	500	999.00	320
62	1000.0000	16384.000	512.0000	4096.0000	500	1299.00	320
66	889.0000	6144.000	336.0000	384.0000	1750	999.00	235
69	837.0000	6144.000	288.4000	384.0000	1502	999.00	196
171	960.0000	1024.000	134.4000	256.0000	1050	249.00	38
173	850.0000	1024.000	124.8000	256.0000	975	249.00	34
184	575.0000	512.000	57.6000	256.0000	900	130.00	18

Figure 15

5.3. Splitting the dataset

Before constructing the model, we begin by splitting the first dataset containing a total of 553 observation values into two new datasets as follows:

- Set A (Training set) contains 70% of the data, corresponding to 387 observation values.
- Set B (Test set) contains 30% of the data, corresponding to 166 observation values.

Set A is used to find the formula, while set B is used to test that formula.

5.4. The initial model

There are 6 factors affecting our targeted variable Release_Price, including Core_Speed, Memory, Memory_Bandwidth, Memory_Bus, Memory_Speed, Texture_Rate. (*R-Code below*)

```
Call:
lm(formula = release_price ~ core_speed + mem + mem_bandwidth +
    mem_bus + mem_speed + texture_rate, data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-476.23  -97.52   -3.77   52.03  911.21

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)   287.506651  48.531148   5.924 7.04e-09 ***
core_speed      0.050274   0.044528   1.129  0.25959
mem             0.010543   0.004056   2.599  0.00971 **
mem_bandwidth   1.392975   0.166490   8.367 1.14e-15 ***
mem_bus        -0.112577   0.019889  -5.660 2.98e-08 ***
mem_speed      -0.245761   0.026959  -9.116  < 2e-16 ***
texture_rate    0.480190   0.227044   2.115  0.03508 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 160.7 on 380 degrees of freedom
Multiple R-squared:  0.6853,	Adjusted R-squared:  0.6803
F-statistic: 137.9 on 6 and 380 DF,  p-value: < 2.2e-16
```

Figure 16

At a glance, we see that the Pr (>|t|) value of variable Core_Speed is too large, which means the impact of Core_Speed on Release_Price is not significant (*Figure below shows it*). Hence, we decide to eliminate that variable.

```
> summary(reg)$coefficient
                 Estimate Std. Error  t value     Pr(>|t|)
(Intercept)  287.50665126 48.53114768  5.924168 7.036754e-09
core_speed     0.05027433  0.04452775  1.129056 2.595864e-01
mem            0.01054307  0.00405631  2.599178 9.708659e-03
mem_bandwidth  1.39297486  0.16649036  8.366700 1.143006e-15
mem_bus       -0.11257671  0.01988938 -5.660142 2.983931e-08
mem_speed     -0.24576054  0.02695940 -9.115950 4.574299e-18
texture_rate   0.48018967  0.22704370  2.114966 3.508329e-02
>
```

Figure 17

5.5. The improved model

After removing the variable Core_Speed, we have the new model:

```
Call:
lm(formula = release_price ~ mem + mem_bandwidth + mem_bus +
    mem_speed + texture_rate, data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-473.31  -98.54    0.04   51.52  921.34

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)   319.263653  39.563498   8.070 9.30e-15 ***
mem             0.011067   0.004031   2.745  0.00633 **
mem_bandwidth   1.330157   0.156975   8.474 5.26e-16 ***
mem_bus        -0.108309   0.019534  -5.545 5.51e-08 ***
mem_speed      -0.234933   0.025205  -9.321  < 2e-16 ***
texture_rate    0.586546   0.206655   2.838  0.00478 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 160.7 on 381 degrees of freedom
Multiple R-squared:  0.6842,    Adjusted R-squared:  0.6801
F-statistic: 165.1 on 5 and 381 DF,  p-value: < 2.2e-16
```

Figure 18

According to the fact that p-value is not really noticeable (2.2x10-16) and the insignificant difference between the adjusted R-squared of the improved model and the initial model, we can conclude that the data is accurate and this improved model is suitable one.

5.6. Conducting the formula

According to the improved model above, we form a linear regression model, in which the independent variables affect the dependent variable Release_Price. The equation for this model:

Release_Price = 319.263653 + 0.011067 × ***Memory*** + 1.330157 × ***Memory_Bandwidth*** – 0.108309 × ***Memory_Bus*** – 0.234933 × ***Memory_Speed*** + 0.586546 × ***Texture_Rate***

The R-code below shows how we predict the unknown value.

```
install.packages("Metrics")
library(Metrics)
data_1 = data.frame(train)
y = data_1$release_price
x = predict(reg)
plot(x,y,xlim = c(0,1500),ylim = c(0,1500),xlab = 'Predicted value',ylab =
       'Actual value',main = 'Relationship between predicted and actual value')

View(test)
actual <- test$release_price
predictions <- predict(reg, newdata = test)
actual <- test$release_price
rmse <- sqrt(mean((predictions - actual)^2))
rmse

data_range <- range(test)
data_range

data_range <- range(test$release_price)
data_range
```

Figure 19

5.7. Testing the formula

In the previous session, we have conducted an equation illustrating the dependence of Release_Price on other factors:

Release_Price = 319.263653 + 0.011067 × ***Memory*** + 1.330157 × ***Memory_Bandwidth*** – 0.108309 × ***Memory_Bus*** – 0.234933 × ***Memory_Speed*** + 0.586546 × ***Texture_Rate***

In this session, to test the aforementioned equation, we apply the equation to the set B (test set) and compare the predicted values (using equation) to the actual values (given data). Figure 20 is used to describe that comparison.

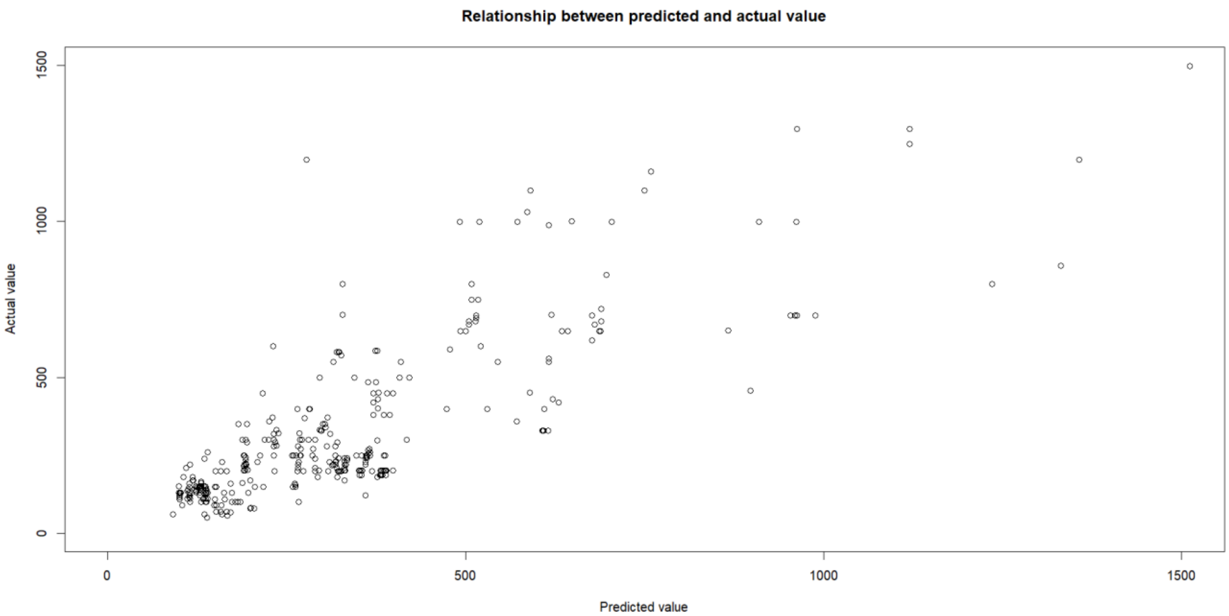

Figure 20

Remark: The horizontal ordinates represent predicted values, while the vertical ordinates represent the actual values. We can observe that these points are on or near the line y=x, that means the predicted values (using conducted formula) approximately equal to the actual values. In conclusion, the we can use the formula to predict the unknown Release_Price of second dataset.

5.8. Predicting

The given chart below shows the prediction for unknown Release_Price values:

	core_speed	mem	mem_bandwidth	mem_bus	mem_speed	release_price	texture_rate
1	738.0000	1024.000	64.0000	256.0000	1000.000	180.6339	47.00000
2	946.8939	512.000	106.0000	512.0000	828.000	222.9864	12.00000
3	946.8939	512.000	51.2000	256.0000	800.000	183.2260	10.00000
4	946.8939	256.000	36.8000	128.0000	1150.000	91.1159	7.00000
5	946.8939	256.000	22.4000	128.0000	700.000	177.0949	6.00000
6	946.8939	256.000	35.2000	128.0000	1100.000	100.1478	6.00000
7	870.0000	2048.000	134.4000	256.0000	1050.000	266.8243	35.00000
8	946.8939	256.000	51.2000	256.0000	800.000	178.6332	7.00000
9	946.8939	2048.000	160.0000	256.0000	1250.000	269.7265	62.00000
10	946.8939	64.000	2.9000	64.0000	366.000	283.8624	90.27463
11	946.8939	128.000	5.2000	128.0000	325.000	238.5534	2.00000
12	650.0000	6144.000	177.6000	384.0000	925.000	385.7072	36.00000
13	705.0000	5120.000	168.0000	320.0000	1050.000	371.0048	90.27463
14	706.0000	12288.000	288.4000	384.0000	1502.000	543.5385	169.00000
15	946.8939	64.000	5.8000	128.0000	360.000	282.1976	90.27463
16	1050.0000	3072.000	57.6000	128.0000	900.000	240.9411	62.00000
17	946.8939	8192.000	320.0000	512.0000	1250.000	539.4045	90.27463
18	732.0000	6144.000	249.6000	384.0000	1300.000	425.2132	90.27463

***Figure** 21*

5.9. Conclusion

After employing multiple linear regression method, the team identified significant factors influencing the retail price of the product, and we have derived a model equation to predict GPU prices based solely on measuring 6 indices (Core_Speed, Memory, Memory_Speed, Memory_Bandwidth, Memory_Bus, Texture_Rate). This model will assist both buyers and sellers in assessing the suitability of the retail cost for this product. Therefore, the model has somewhat attracted potential customers to the product and simultaneously supported the manufacturer in pricing the product.

Overall, the retail price of GPUs after using the model is relatively consistent with the original prices.

Furthermore, it is easy to observe that if the values of variables "Memory", "Memory_Bandwidth", “Memory_Bus” and "Texture_Rate" increase, the value of "Release_Price" will also increase. This implies that the retail prices of GPUs are directly proportional to the mentioned variables, while the remaining variables will be inversely proportional to the GPU retail price.

6 DISCUSSION AND EXPANSION

ADVANTAGE:

- Multiple linear regression allows us to analyze the effects of multiple independent variables on a dependent variable. This helps us to determine the relative importance of each independent variable in predicting the dependent variable
- By using 6 key factors impacting the dependent variable “Release_Price”, we were able to predict selling price of GPUs

LIMITATIONS:

- It is necessary to have 6 required variables for the model to work optimally
- To make the model work efficiently, it is necessary to remove outliers, which are observation values significantly different from other observation values in the dataset, as they may influence relationship between predictor variable and dependent variable and lead to inaccurate predictions

7 CONCLUSION

Our team identified significant factors influencing the selling price of the product, and we obtained a model equation to predict GPU prices by measuring 6 variables by using multiple linear regression method. The model helped potential customers consider this product and simultaneously supported the manufacturer in pricing the product.

Overall, the retail price of GPUs after using the model is relatively appropriate compared to the original price. Additionally, if the values of the variables "Memory," "Memory_Bandwidth," and "Texture_Rate" increase, the value of "Release_Price" will also increase. This means that the selling price of GPUs will be directly proportional to those variables, while the remaining variables will be inversely proportional to the GPU’s selling price.

8 DATA & CODE SOURCE

8.1. Data source

https://www.kaggle.com/datasets/iliassekkaf/computerparts?resource=downlo

8.2. Code source

https://drive.google.com/drive/folders/1oDNnHuHQRdTt1lpWjedP4onzRmtp9Dr8?usp=share_link

REFERENCES

1. *Card màn hình (VGA) và các thông số quan trọng thường gặp,* truy cập từ https://gearvn.com/pages/card-man-hinh-va-cac-thong-so-quan-trong-thuong-gap

2. Douglas C.Montgomery & George C.Runger, *APPLIED STATISTICS AND PROBABILITY FOR ENGINEERS 6TH EDITION.*

3. *GRAPHICS PROCESSING UNIT,* truy cập từ https://en.wikipedia.org/wiki/Graphics_processing_unit

4. *Ilissek, COMPUTERS PARTS(CPUs and GPUs),* truy cập từ https://www.kaggle.com/datasets/iliassekkaf/computerparts/data

5. Nguyễn Tiến Dũng (eds.), Nguyễn Đình Huy, *Xác suất – Thống kê & Phân tích số liệu*, *2019.*

6. *ORDINARY LEAST SQUARES*, truy cập từ https://en.wikipedia.org/wiki/Ordinary_least_squares