ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC ỨNG DỤNG

XÁC SUẤT VÀ THỐNG KÊ (MT 2013)

Báo cáo Bài tập lớn - Học kỳ 20241

PHÂN TÍCH DỮ LIỆU GPU

GIẢNG VIÊN HƯỚNG DẪN: NGUYỄN KIỀU DUNG

NHÓM MT09

STT	Họ tên	MSSV	Lớp
1	**Tạ Tiến Tài**	2313013	Lớp L14
2	Đoàn Công Vinh	2313906	Lớp L17
3	Phạm Công Võ	2313946	Lớp L17
4	Nguyễn Lê Thảo Ly	2312010	Lớp L14
5	Đinh Trác Đức Anh	2310068	Lớp L14
6	Nguyễn Thị Quỳnh Tâm	2313037	Lớp L17
7	Bành Huỳnh Minh Huy	2311118	Lớp L04

Thành phố Hồ Chí Minh, 11/2024

DANH SÁCH THÀNH VIÊN

HỌ TÊN	MSSV	NHIỆM VỤ	ĐIỂM CHIA
TẠ TIẾN TÀI	2313013	Soạn thảo báo cáo	0
ĐOÀN CÔNG VINH	2313906	Thống kê mô tả	0
PHẠM CÔNG VÕ	2313946	Kiểm định hai mẫu	0
NGUYỄN LÊ THẢO LY	2312010	Kiểm định một mẫu	0
ĐINH TRÁC ĐỨC ANH	2310068	Hồi quy tuyến tính	0
NGUYỄN THỊ QUỲNH TÂM	2313037	Tổng quan dữ liệu	0
BÀNH HUỲNH MINH HUY	2311118	Bài toán Anova	0

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

TỔNG QUAN DỮ LIỆU

1.1 Tổng quan về tập dữ liệu

GPU (Bộ xử lý đồ họa) là một chip hoặc mạch điện tử có khả năng xử lý đồ họa và hình ảnh với tốc độ cao. Nó rất quan trọng trong công nghệ máy tính hiện nay và được ứng dụng rộng rãi trong cả máy tính chuyên nghiệp và cá nhân. GPU hoạt động theo phương pháp xử lý song song, cho phép thực hiện cùng một phép toán trên nhiều giá trị dữ liệu đồng thời, điều này rất hữu ích trong các ứng dụng đồ họa, hoạt hình và video. Mặc dù nổi bật trong ngành trò chơi, GPU hiện nay còn đóng vai trò quan trọng trong các lĩnh vực như máy học, trí tuệ nhân tạo và mô phỏng. Để nghiên cứu chi tiết các đặc điểm của GPU, nhóm nghiên cứu sử dụng tập dữ liệu chứa 3,406 mẫu với 34 biến số đặc trưng.

Thuộc tính	**Thông tin**
Nguồn dữ liệu	`All_GPUs.csv`
Đối tượng	Tập hợp thuộc tính của GPU
Số lượng quan sát	3406
Số lượng biến	34

Bảng 1.1.1: Tổng quan về tập dữ liệu

1.2 Ý nghĩa từng biến trong tập dữ liệu

STT	Tên biến	Kiểu biến	Đơn vị	Mô tả
1	Architecture	Phân loại		Kiến trúc phần cứng GPU.
2	Best Resolution	Phân loại		Độ phân giải tốt nhất của GPU.
3	Boost Clock	Liên tục	MHz	Xung nhịp ép xung tối đa.
4	Core Speed	Liên tục	MHz	Tốc độ xử lý của GPU.
5	DVI Connection	Phân loại		Số cổng DVI kết nối video.
6	Dedicated	Phân loại		Bộ nhớ đồ họa riêng của GPU.
7	Direct X	Phân loại		Bộ API xử lý đa phương tiện.
8	DisplayPort Connection	Phân loại		Số cổng DisplayPort kết nối video.
9	HDMI Connection	Phân loại		Số cổng HDMI của GPU.
10	Integrated	Phân loại		GPU tích hợp trong CPU.
11	L2 Cache	Liên tục	KB	Bộ đệm giữa GPU và bộ nhớ chính.
12	Manufacturer	Phân loại		Công ty sản xuất GPU.
13	MaxPower	Liên tục	W	Công suất tối đa GPU.
14	Memory	Liên tục	MB	Dung lượng bộ nhớ GPU.
15	Memory Bandwidth	Liên tục	GB/s	Tốc độ truyền dữ liệu bộ nhớ.
16	Memory Bus	Liên tục	Bit	Độ rộng kênh truyền bộ nhớ.
17	Memory Speed	Liên tục	MHz	Tốc độ đọc/ghi bộ nhớ.
18	Memory Type	Phân loại		Loại bộ nhớ của GPU.
19	Name	Phân loại		Tên của GPU.
20	Notebook GPU	Phân loại		GPU dành cho laptop.
21	OpenGL	Liên tục		Thư viện đồ họa 2D/3D.
22	PSU	Liên tục	W, A	Nguồn điện GPU yêu cầu.

Bảng 1.2.2: Danh sách thuộc tính GPU (1-22)

STT	Tên biến	Kiểu biến	Đơn vị	Mô tả
23	Pixel Rate	Liên tục	GPixel/s	Số pixel hiển thị mỗi giây.
24	Power Connector	Phân loại	Pin	Cổng cấp điện bổ sung cho GPU.
25	Process	Liên tục	nm	Công nghệ sản xuất GPU.
26	ROPs	Phân loại		Đơn vị xử lý và xuất pixel.
27	Release Date	Liên tục		Ngày phát hành GPU.
28	Release Price	Liên tục		Giá phát hành của GPU.
29	Resolution WxH	Phân loại		Độ phân giải màn hình (pixel).
30	SLI Crossfire	Phân loại		Phương pháp đa GPU.
31	Shader	Phân loại		Xử lý màu sắc và hiệu ứng.
32	TMUs	Liên tục		Đơn vị xử lý kết cấu ảnh.
33	Texture Rate	Liên tục	Gtexel/s	Số pixel có kết cấu mỗi giây.
34	VGA Connection	Phân loại		Số cổng kết nối VGA.

Bảng 1.2.3: Danh sách thuộc tính GPU (23-34)

Nhìn chung, tập dữ liệu này cung cấp một nguồn tài nguyên phong phú, hữu ích cho việc phân tích hiệu năng, so sánh và đánh giá các GPU khác nhau dựa trên nhiều tiêu chí kỹ thuật. Điều này có thể hỗ trợ trong việc tối ưu hóa phần cứng, lựa chọn GPU cho từng loại tác vụ, và nghiên cứu về xu hướng phát triển của công nghệ đồ họa.

Chương 2

TỔNG QUAN KIẾN THỨC

2.1 Hồi quy tuyến tính đa biến (Multivariate Linear Regression)

2.1.1 Công thức hồi quy tuyến tính đa biến

Hồi quy tuyến tính đa biến mô hình hóa mối quan hệ giữa một biến phụ thuộc (y) và nhiều biến độc lập ($x_1, x_2, \ldots, x_k$):

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \epsilon$$

Trong đó:

- y: Biến phụ thuộc (đầu ra).
- $x_1, x_2, \ldots, x_k$: Các biến độc lập (đầu vào).
- $\beta_0, \beta_1, \ldots, \beta_k$: Hệ số hồi quy.
- ϵ: Sai số ngẫu nhiên.

2.1.2 Điều kiện để mô hình dự đoán tốt

1. **Tính tuyến tính:** Quan hệ giữa y và x_i phải tuyến tính.

2. **Độc lập của sai số:** Sai số (ϵ) phải độc lập (kiểm tra bằng Durbin-Watson test).
3. **Sai số có phân phối chuẩn:** Kiểm tra bằng Shapiro-Wilk hoặc QQ-Plot.
4. **Phương sai đồng nhất:** Phương sai của sai số không thay đổi.
5. **Không đa cộng tuyến:** Kiểm tra bằng hệ số VIF (Variance Inflation Factor).

2.1.3 Phương pháp đánh giá mô hình

- **P-value của hệ số hồi quy (β_i):** Kiểm định ý nghĩa từng biến độc lập:

$$H_0 : \beta_i = 0 \quad \text{và} \quad H_1 : \beta_i \neq 0$$

  Nếu P-value < 0.05, biến x_i có ý nghĩa thống kê.

- **Hệ số R^2:** Cho biết phần trăm phương sai của y được giải thích bởi các biến x_i:

$$R^2 \in [0, 1]$$

- **Kiểm định tổng thể (F-test):**

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_k = 0$$

  Nếu P-value < 0.05, mô hình tổng thể có ý nghĩa.

- **Đồ thị chẩn đoán:** Residual Plot và QQ-Plot giúp kiểm tra các giả định mô hình.

2.2 Ước lượng 1 mẫu (One-Sample Estimation)

2.2.1 Lý thuyết ước lượng 1 mẫu

Ước lượng một mẫu suy luận tham số tổng thể (μ, p, σ^2) dựa trên dữ liệu mẫu:

- **Ước lượng điểm:** Giá trị duy nhất, ví dụ trung bình mẫu ($\bar{x}$).

- **Ước lượng khoảng:** Khoảng tin cậy (Confidence Interval - CI):

$$CI = \text{Ước lượng điểm} \pm \text{Biên sai số}$$

2.2.2 Kiểm định phân phối chuẩn

- **QQ-Plot:** So sánh phân vị dữ liệu thực tế với phân vị chuẩn lý thuyết.
- **Shapiro-Wilk Test:**

$$H_0 : \text{Dữ liệu phân phối chuẩn}, \quad H_1 : \text{Dữ liệu không phân phối chuẩn}.$$

Nếu P-value < 0.05, bác bỏ H_0.

2.2.3 Các trường hợp và hàm ước lượng khoảng tin cậy

1. **Phân phối chuẩn, biết phương sai tổng thể:**

$$CI = \bar{x} \pm z\frac{\sigma}{\sqrt{n}}$$

2. **Phân phối chuẩn, không biết phương sai tổng thể:**

$$CI = \bar{x} \pm t\frac{s}{\sqrt{n}}$$

3. **Dữ liệu không phân phối chuẩn:** Dùng các phương pháp phi tham số như bootstrap.

2.3 Kiểm định 2 mẫu (Two-Sample Testing)

2.3.1 Lý thuyết kiểm định 2 mẫu

Kiểm định 2 mẫu so sánh tham số tổng thể giữa hai nhóm:

- So sánh trung bình (μ_1 và μ_2).
- So sánh tỷ lệ (p_1 và p_2).
- So sánh phương sai (σ_1^2 và σ_2^2).

2.3.2 Các trường hợp bài toán

1. **Hai mẫu độc lập:** Hai nhóm không liên quan (ví dụ: Nam và Nữ). Sử dụng t-test hoặc Z-test.
2. **Hai mẫu ghép cặp:** So sánh dữ liệu trước-sau của một nhóm. Sử dụng paired t-test.

2.3.3 Hàm ước lượng khoảng tin cậy

- **Trung bình hai mẫu độc lập:**

$$CI = (\bar{x}_1 - \bar{x}_2) \pm t\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

- **Trung bình hai mẫu ghép cặp:**

$$CI = \bar{d} \pm t\frac{s_d}{\sqrt{n}}$$

  Trong đó d là hiệu số giữa các cặp.

- **Tỷ lệ hai mẫu:**

$$CI = (\hat{p}_1 - \hat{p}_2) \pm z\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$$

2.4 Phân tích phương sai (ANOVA - Analysis of Variance)

2.4.1 Lý thuyết phân tích phương sai

ANOVA kiểm tra sự khác biệt trung bình giữa nhiều nhóm ($k \geq 3$):

- **Giả thuyết:**

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_k, \quad H_1 : \text{Có ít nhất một nhóm khác biệt.}$$

- **F-statistic:**

$$F = \frac{\text{Phương sai giữa các nhóm}}{\text{Phương sai trong nhóm}}$$

2.4.2 Các loại ANOVA

1. **One-Way ANOVA:** Xét một yếu tố (factor).
2. **Two-Way ANOVA:** Xét đồng thời hai yếu tố.
3. **Repeated Measures ANOVA:** Đo lặp trên cùng đối tượng.

2.4.3 Đánh giá kết quả

- Nếu P-value < 0.05, bác bỏ H_0, các nhóm có sự khác biệt.
- Sử dụng Post-hoc Tests (Tukey, Bonferroni) để so sánh từng cặp nhóm.

Chương 3

TIỀN XỬ LÝ SỐ LIỆU

3.1 Đọc Dữ liệu Từ File CSV và Khám Phá Sơ Bộ

```
# Doc du lieu tu file csv "All_GPUs.csv"
GPU_data <- read.csv("All_GPUs.csv")
head(GPU_data)
```

Ở bước này ta thực hiện hai thao tác chính thực hiện hai thao tác chính:

- **Đọc dữ liệu từ file**, lệnh `read.csv("All_GPUs.csv")` được sử dụng để đọc dữ liệu từ file `All_GPUs.csv` và lưu trữ dữ liệu vào biến `GPU_data`. Hàm `read.csv` là một hàm phổ biến trong R để tải dữ liệu từ file CSV, tự động chuyển đổi dữ liệu thành một cấu trúc `data.frame`, rất thuận tiện cho các thao tác xử lý và phân tích sau này. Mặc định, hàm `read.csv` sẽ coi dòng đầu tiên của file CSV là tiêu đề của các cột và mỗi dòng tiếp theo là một bản ghi dữ liệu. Để thực hiện được lệnh này, file `All_GPUs.csv` cần phải có đường dẫn đúng hoặc phải nằm trong thư mục làm việc hiện tại của R để tránh lỗi "không tìm thấy file".

- **Kiểm tra sơ bộ cấu trúc bộ dữ liệu**, lệnh `head(GPU_data)` hiển thị 6 dòng đầu tiên của `GPU_data`. Kết quả từ hàm `head()` cho phép người dùng nhanh chóng xem cấu trúc của dữ liệu: tên các cột, loại dữ liệu (như số hoặc chuỗi ký tự), và tính chính xác của dữ liệu đầu vào. Điều này giúp xác định xem dữ liệu có được nhập

đúng cách không, như tiêu đề của các cột có hiển thị đúng không hoặc có bản ghi nào bị lỗi trong quá trình nhập dữ liệu không.

Mục tiêu của đoạn code này là đảm bảo dữ liệu từ file CSV đã được nhập vào đúng cách và sẵn sàng để xử lý ở các bước tiếp theo.

	Architecture	Best_Resolution	Boost_Clock	Core_Speed	DVI_Connection	Dedicated	Direct_X	DisplayPort_Connection
1	Tesla G92b	<NA>	<NA>	738 MHz	2	Yes	DX 10.0	<NA>
2	R600 XT	1366 x 768	<NA>	<NA>	2	Yes	DX 10	<NA>
3	R600 PRO	1366 x 768	<NA>	<NA>	2	Yes	DX 10	<NA>
4	RV630	1024 x 768	<NA>	<NA>	2	Yes	DX 10	<NA>
5	RV630	1024 x 768	<NA>	<NA>	2	Yes	DX 10	<NA>
6	RV630	1024 x 768	<NA>	<NA>	2	Yes	DX 10	<NA>

	HDMI_Connection	Integrated	L2_Cache	Manufacturer	Max_Power	Memory	Memory_Bandwidth	Memory_Bus	Memory_Speed
1	0	No	0KB	Nvidia	141 Watts	1024 MB	64GB/sec	256 Bit	1000 MHz
2	0	No	0KB	AMD	215 Watts	512 MB	106GB/sec	512 Bit	828 MHz
3	0	No	0KB	AMD	200 Watts	512 MB	51.2GB/sec	256 Bit	800 MHz
4	0	No	0KB	AMD	<NA>	256 MB	36.8GB/sec	128 Bit	1150 MHz
5	0	No	0KB	AMD	45 Watts	256 MB	22.4GB/sec	128 Bit	700 MHz
6	0	No	0KB	AMD	50 Watts	256 MB	35.2GB/sec	128 Bit	1100 MHz

	Memory_Type	Name	Notebook_GPU	Open_GL	PSU	Pixel_Rate	Power_Connector
1	GDDR3	GeForce GTS 150	No	3.3	450 Watt & 38 Amps	12 GPixel/s	None
2	GDDR3	Radeon HD 2900 XT 512MB	No	3.1	550 Watt & 35 Amps	12 GPixel/s	None
3	GDDR3	Radeon HD 2900 Pro	No	3.1	550 Watt & 35 Amps	10 GPixel/s	None
4	GDDR4	Radeon HD 2600 XT Diamond Edition	No	3.3	<NA>	3 GPixel/s	None
5	GDDR3	Radeon HD 2600 XT	No	3.1	400 Watt & 25 Amps	3 GPixel/s	None
6	GDDR4	Radeon HD 2600 XT 256MB GDDR4	No	3.3	400 Watt & 26 Amps	3 GPixel/s	None

	Process	ROPs	Release_Date	Release_Price	Resolution_WxH	SLI_Crossfire	Shader	TMUs	Texture_Rate	VGA_Connection
1	55nm	16	\n01-Mar-2009	<NA>	2560x1600	Yes	4	64	47 GTexel/s	0
2	80nm	16	\n14-May-2007	<NA>	2560x1600	Yes	4	16	12 GTexel/s	0
3	80nm	16	\n07-Dec-2007	<NA>	2560x1600	Yes	4	16	10 GTexel/s	0
4	65nm	4	\n01-Jul-2007	<NA>	2560x1600	Yes	4	8	7 GTexel/s	0
5	65nm	4	\n28-Jun-2007	<NA>	2560x1600	Yes	4	8	6 GTexel/s	0
6	65nm	4	\n26-Jun-2007	<NA>	2560x1600	Yes	4	8	6 GTexel/s	0

Hình 3.1: Hiển thị 6 dòng đầu tiên của bộ dữ liệu

3.2 Chuẩn hóa dữ liệu khuyết

Đoạn code này thực hiện các thao tác xử lý dữ liệu để thay thế những giá trị không hợp lệ hoặc thiếu thông tin trong bộ dữ liệu `gpu_data` bằng giá trị `NA` trong R. Mục tiêu của các bước này là làm sạch dữ liệu, chuẩn hóa các giá trị khuyết và tạo điều kiện thuận lợi cho các bước phân tích tiếp theo.

```
# Thay the tat ca cac o trong trong dataset bang NA
gpu_data[gpu_data == ""] <- NA
# Thay the cac gia tri co dinh dang "-\n" bang NA
gpu_data[] <- lapply(gpu_data, function(x) gsub("^\\n- $", NA, x))
# Thay the cac chuoi "na" bang NA
gpu_data[gpu_data == "na"] <- NA
```

Đoạn mã này thực hiện các bước xử lý dữ liệu trong `gpu_data` để thay thế các giá trị không hợp lệ hoặc thiếu thông tin bằng `NA`:

- **Thay thế ô trống:** Các ô có giá trị chuỗi rỗng ("") được thay thế bằng NA. Bằng cách thay thế tất cả các ô trống bằng NA, chúng ta chuẩn hóa dữ liệu và đảm bảo rằng các ô này sẽ được xử lý như các giá trị khuyết (missing values) thay vì giữ lại chuỗi trống, điều này giúp cho quá trình phân tích dữ liệu chính xác hơn.

- **Thay thế chuỗi "\n-":** Để thực hiện việc này, nhóm nghiên cứu sử dụng hàm gsub("^ \\ n- $ ", NA, x) để tìm kiếm và thay thế tất cả các giá trị có định dạng chính xác là chuỗi "\n-" trong các cột của gpu_data. Lệnh^ \\ n- $ là một biểu thức chính quy (regex) dùng để khớp với chuỗi có dạng "\n-", bao gồm một ký tự xuống dòng (\n) và dấu gạch ngang (-), với không gian sau dấu gạch ngang. Nếu một giá trị trong cột nào đó khớp với chuỗi này, nó sẽ bị thay thế bằng giá trị NA. Việc này loại bỏ những giá trị không có ý nghĩa và không thể sử dụng trong phân tích dữ liệu.

- **Thay thế chuỗi "na":** Trong một số bộ dữ liệu, giá trị "na" có thể được sử dụng để đại diện cho dữ liệu khuyết hoặc không xác định. Nhằm đảm bảo tính đồng nhất trong dữ liệu, vì R sử dụng NA để biểu thị các giá trị bị thiếu hoặc không có giá trị. Các chuỗi "na" được thay thế bằng NA.

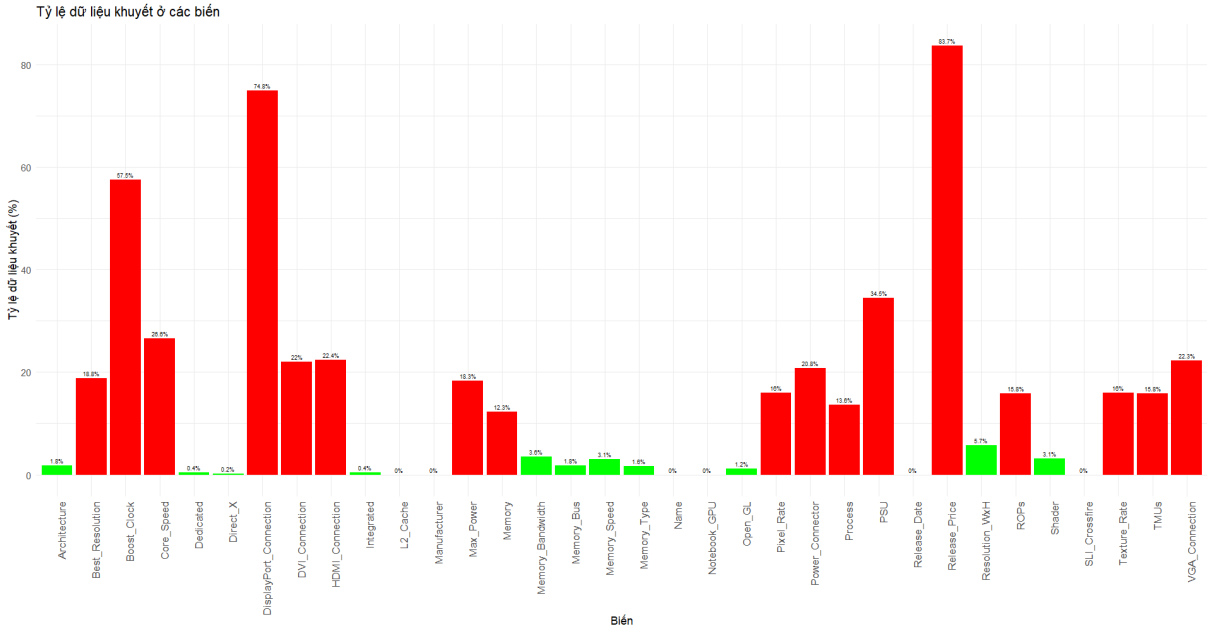
Để trực quan hơn tỉ lệ dữ liệu khuyết trong bộ dữ liệu, chúng ta sẽ sử dụng hàm ggplot của thư viện ggplot2 để vẽ biểu đồ thể hiện tỉ lệ dữ liệu khuyết trong bộ dữ liệu. Trong đó,

- **Cột xanh:** là những cột có tỉ lệ dữ liệu khuyết dưới 10% và sẽ được giữ lại sau bước lọc dữ liệu khuyết.

- **Cột đỏ:** là những cột có tỉ lệ khuyết trên 10% và sẽ được loại bỏ sau bước lọc dữ liệu khuyết.

```
# Goi thu vien ggplot2 de su dung ham ggplot
library(ggplot2)
# Tao bieu do cot cho du lieu khuyet o cac bien
ggplot(na_summary, aes(x = column, y = na_percentage,
    fill = na_percentage > 10)) +
    geom_bar(stat = "identity") +
    geom_text(aes(label = paste0(round(na_percentage, 1), "%")),
              vjust = -0.5, size = 2) +
    labs(title = "Ty le du lieu khuyet o cac bien",
       x = "Bien",
       y = "Ty le du lieu khuyet (%)") +
    scale_fill_manual(values = c("TRUE" = "red", "FALSE" = "green")) +
    theme_minimal() +
    theme(axis.text.x = element_text(size = 10, angle = 90, hjust = 1),
        legend.position = "none")
```

Sau khi chạy đoạn code trên ta thu được biểu đồ sau



Hình 3.2: Tỷ lệ dữ liệu khuyết ở từng biến của tập dữ liệu

3.3 Lọc bỏ dữ liệu khuyết

```
# Loc cac cot co ti le khuyet tren 10%
selected_columns <- na_summary$column[na_summary$na_percentage < 10]
# Giu lai cac cot thoa man dieu kien trong dataframe ban dau
new_gpu_data <- gpu_data[, selected_columns]
# Xoa cac quan sat chua du lieu khuyet
new_gpu_data<-na.omit(new_gpu_data)
# Lua chon cac bien can xoa don vi
columns_to_clean <- c("l2_cache","memory_bandwidth","memory_bus",
"memory_speed")
remove_units <- function(column) {
# Su dung sub đe lay phan so o đau chuoi truoc bat ky ky tu
nao khac
cleaned_column <- sub("([0-9.]+).*", "\\1", column)
# Chuyen đoi ket qua ve kieu numeric
cleaned_column <- as.numeric(cleaned_column)
return(cleaned_column)
}
```

Ở bước này ta lọc các cột trong dữ liệu gốc có tỷ lệ dữ liệu khuyết dưới 10%, sau đó giữ lại những cột này trong một dataframe mới. Tiếp theo đó, xác định các cột cần làm sạch dữ liệu (xóa đơn vị), và áp dụng hàm để loại bỏ phần đơn vị trong các giá trị của các cột này, chỉ giữ lại giá trị số để chuẩn bị cho các phân tích tiếp theo. Cụ thể như sau:

- **Lọc các cột có tỷ lệ dữ liệu khuyết dưới 10%:** Đoạn code đầu tiên lọc các cột trong bảng tóm tắt `na_summary` có tỷ lệ dữ liệu khuyết nhỏ hơn 10%. Điều này được thực hiện bằng cách sử dụng biểu thức điều kiện `na_summary$na_percentage < 10` để chọn các cột có tỷ lệ khuyết thấp hơn mức ngưỡng này. Kết quả của thao tác này được lưu vào biến `selected_columns`, chứa tên của các cột thỏa mãn điều kiện.

- **Giữ lại các cột thỏa mãn điều kiện trong dataframe ban đầu:** Dòng tiếp theo thực hiện việc lọc lại dữ liệu trong `gpu_data` để chỉ giữ lại các cột có tỷ lệ dữ liệu khuyết dưới 10%. Câu lệnh `gpu_data[, selected_columns]` sẽ chọn tất cả các dòng của các cột có tên trong `selected_columns`, tạo ra một dataframe mới

`new_gpu_data`. Điều này giúp giảm thiểu sự phức tạp của dữ liệu, chỉ giữ lại các thông tin quan trọng và đáng tin cậy.

- **Xóa các quan sát chứa dữ liệu khuyết (commented out):** Hàm `na.omit()` sẽ loại bỏ bất kỳ hàng nào chứa dữ liệu thiếu (NA) trong `new_gpu_data`, giúp dữ liệu sau khi xử lý sạch hơn và không còn chứa giá trị khuyết.

- **Lựa chọn các biến cần xoá đơn vị:** Biến `columns_to_clean` được sử dụng để chứa danh sách các tên cột cần phải xử lý. Các biến `L2_cache`, `memory_bandwidth`, `memory_bus`, và `memory_speed` được liệt kê trong danh sách này, cho thấy rằng chúng cần phải được làm sạch dữ liệu (tức là loại bỏ các đơn vị đi kèm với các giá trị trong các cột này).

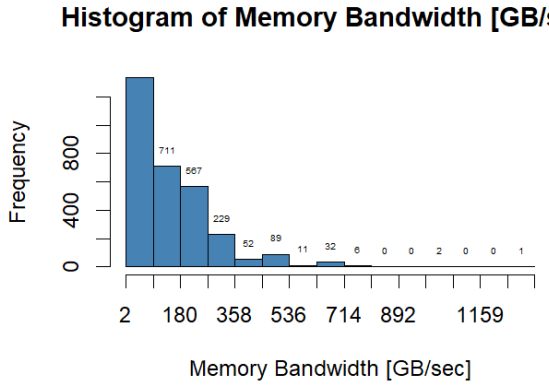
- **Chức năng `remove_units` để loại bỏ đơn vị:** Hàm `remove_units` được định nghĩa để làm sạch dữ liệu trong các cột có đơn vị đính kèm. Hàm này sử dụng hàm `sub()` trong R để lấy phần số ở đầu chuỗi trước bất kỳ ký tự nào khác (tức là loại bỏ các ký tự không phải số). Cụ thể, biểu thức chính quy `"([0-9.]+).*"` tìm tất cả các ký tự số và dấu chấm, và giữ lại phần này. Sau đó, kết quả sẽ được chuyển đổi về kiểu dữ liệu số thực (`numeric`) để có thể sử dụng trong các phép toán sau này. Cuối cùng, hàm trả về kết quả đã được làm sạch.

Chương 4

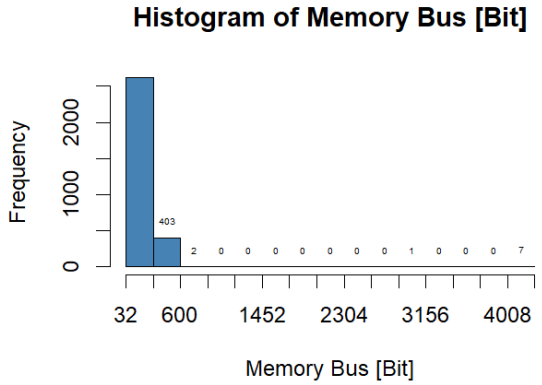
THỐNG KÊ MÔ TẢ

4.1 Biểu đồ tần số

Biểu đồ tần số giúp chúng ta thấy được sự thay đổi và biến động của các thông số cấu hình GPU. Sau khi đã làm sạch dữ liệu, chúng ta sẽ sử dụng hàm `hist()` cùng với các tham số phù hợp để vẽ biểu đồ tần số cho một số thông số định lượng.



Hình 4.1: Biểu đồ phân phối tần số mẫu Memory_Bandwidth

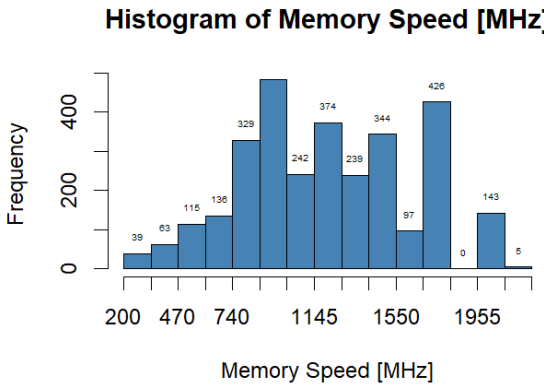


Hình 4.2: Biểu đồ phân phối tần số mẫu Memory_Bus

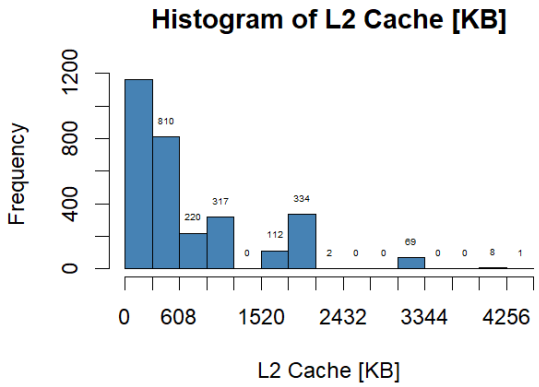
- **Memory Bandwidth:** Băng thông GPU trong tập dữ liệu có phân phối lệch trái, chủ yếu nằm trong khoảng từ 2 đến 269 GB/giây. Mặc dù các giá trị băng thông lớn hiện nay vẫn còn hiếm, nhưng với sự cạnh tranh trong ngành, băng thông GPU dự kiến sẽ tiếp tục được cải thiện để đáp ứng nhu cầu đồ họa ngày càng cao của khách hàng.

- **Memory Bus:** Phần lớn các GPU có bus bộ nhớ rộng từ 64 đến 512 bit và phân

phối lệch trái. Tuy nhiên, các GPU ngày càng được trang bị bus bộ nhớ rộng hơn để tăng cường băng thông và khả năng xử lý.



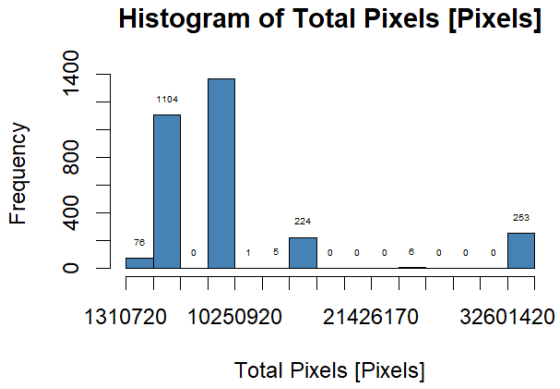
Hình 4.3: Biểu đồ phân phối tần số mẫu Memory_speed



Hình 4.4: Biểu đồ phân phối tần số mẫu L2_Cache

- **Memory Speed:** Tốc độ bộ nhớ của GPU có dạng hình răng cưa, thường nằm trong khoảng từ 740 MHz đến 1550 MHz, với tần suất từ 200-500 sản phẩm mỗi khoảng giá trị. Sự phân bố này phản ánh việc các nhà sản xuất cung cấp các sản phẩm có tốc độ khác nhau để đáp ứng nhu cầu phong phú của người dùng.

- **L2 Cache:** Dung lượng L2 Cache của hầu hết GPU hiện nay dao động từ 0 KB đến 1000 KB. Các GPU có dung lượng L2 Cache trên 1000 KB thường thuộc về các dòng sản phẩm cao cấp và chuyên dụng.



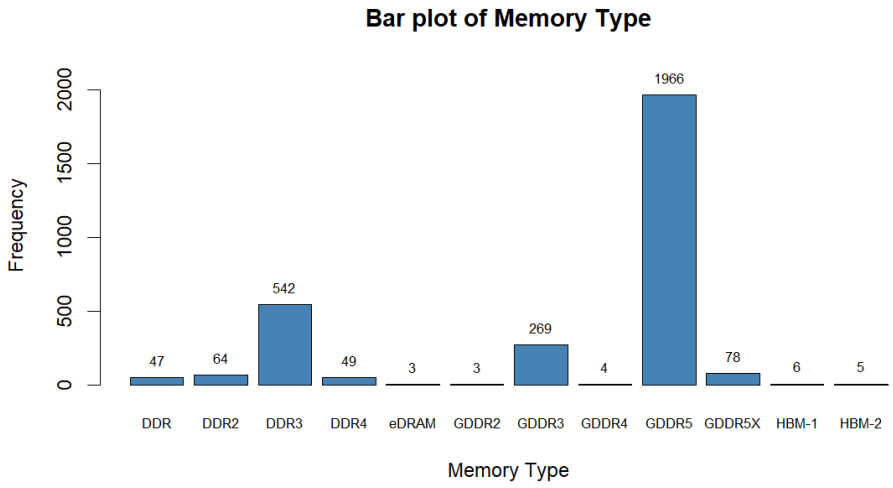
Hình 4.5: Biểu đồ phân phối tần số mẫu Total_Pixels

- **Total Pixels:** Đây là tích của chiều dài và chiều rộng của biến ResolutionWxH. Biểu đồ cho thấy độ phân giải của các GPU trong tập dữ liệu tập trung chủ yếu

ở hai mức: 2560x1600 pixels và 4096x2160 pixels. Độ phân giải 2560x1600 pixels thường phù hợp với máy tính cá nhân không yêu cầu hình ảnh quá cao, trong khi độ phân giải 4096x2160 pixels thường dành cho tivi hoặc thiết bị phục vụ điện ảnh, đáp ứng nhu cầu hình ảnh sắc nét của các chuyên gia trong lĩnh vực điện ảnh và trò chơi điện tử.

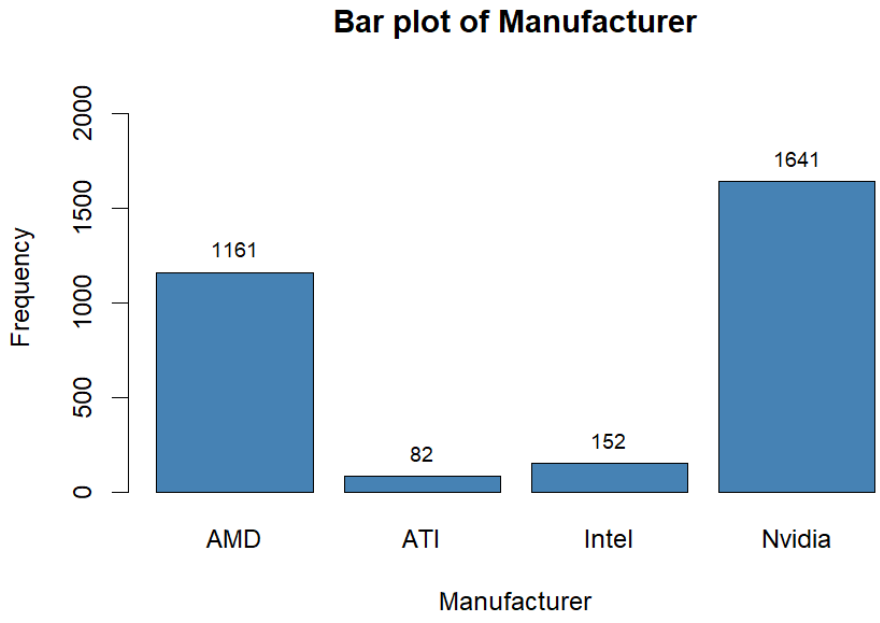
4.2 Biểu đồ cột

Nhóm nghiên cứu đã chọn biểu đồ cột để thể hiện tần suất và sự phân bố của các biến phân loại như Manufacturer và Memory Type trong GPU. Loại biểu đồ này giúp cung cấp cái nhìn trực quan về các thành phần khác nhau của các nhà sản xuất và loại bộ nhớ của GPU trong tập dữ liệu.



Hình 4.6: Biểu đồ cột biến Memory_Type

Nhận xét: Phần lớn các GPU trong tập dữ liệu sử dụng bộ nhớ GDDR5, mang lại hiệu suất cao và tốc độ truyền tải dữ liệu nhanh. Các loại bộ nhớ khác cùng dòng như GDDR3 và DDR3 cũng xuất hiện phổ biến trên thị trường, nhưng số lượng GPU sử dụng mỗi loại bộ nhớ này vẫn chưa đạt tới một nửa so với GDDR5. Các loại bộ nhớ khác như eDRAM và HBM cũng có mặt nhưng vẫn chiếm thị phần nhỏ.



Hình 4.7: Biểu đồ cột biến Manufacturer

Nhận xét: Dựa vào biểu đồ cột biểu diễn biến Manufacturer, ta có thể thấy Nvidia và AMD là hai nhà sản xuất thống trị thị trường GPU, trong đó Nvidia dẫn đầu và AMD theo sau. Ngược lại, Intel và ATI có số lượng GPU sản xuất ít hơn đáng kể so với Nvidia và AMD.

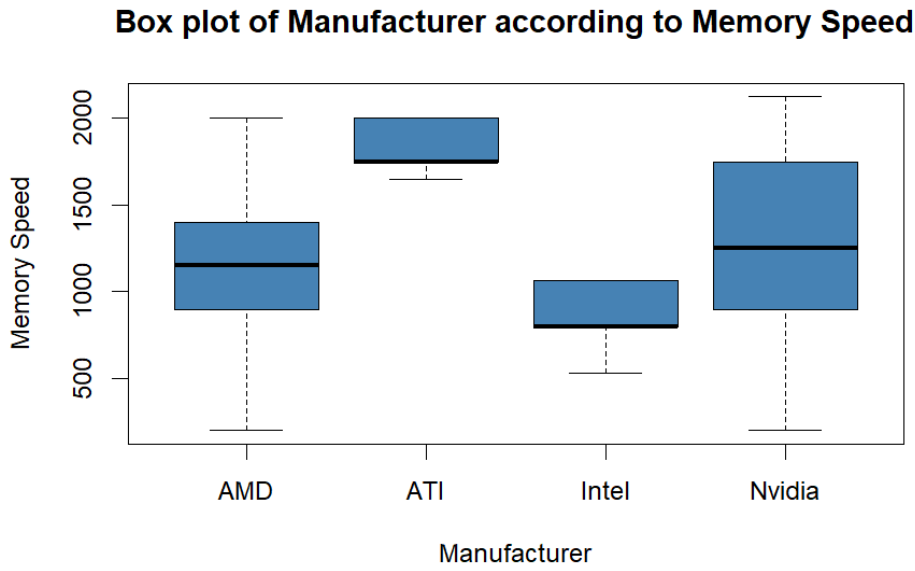
4.3 Biểu đồ hộp

Dùng các hàm `tapply()`, `mean()`, `sd()`, `min()`, `max()`,... để thống kê các giá trị thống kê mô tả (kỳ vọng, phương sai, độ lệch chuẩn, GTNN, GTLN, các điểm tứ phân vị) cho biến định lượng Memory_Speed theo biến Manufacturer và biến Shader.

Biểu đồ hộp của biến Memory Speed theo biến Manufacturer

	Mean	Var	Sd	Min	Max	Q1	Q2	Q3
AMD	1131.2145	126397.34	355.5240	200	2000	900	1150	1400
ATI	1828.0488	18339.36	135.4229	1650	2000	1750	1750	2000
Intel	880.8026	22897.67	151.3198	533	1067	800	800	1067
Nvidia	1267.3985	197563.96	444.4817	200	2127	900	1253	1750

Hình 4.8: Các giá trị đặc trưng của biến Memory_Speed theo biến Manufacturer



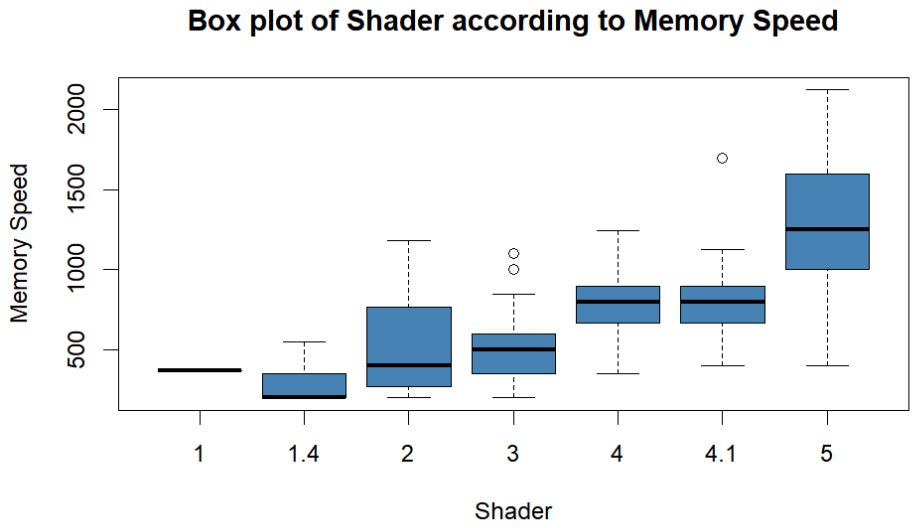
Hình 4.9: Biểu đồ hộp biến Memory_Speed theo biến Manufacturer

Nhận xét: Về tốc độ bộ nhớ (Memory Speed), AMD và Nvidia có tốc độ bộ nhớ phân bố rộng, phù hợp với nhiều phân khúc từ trung cấp đến cao cấp. ATI có tốc độ bộ nhớ trung bình cao nhất và ít phân tán. Trong khi đó, Intel cũng có mức độ phân tán thấp nhưng tốc độ bộ nhớ trung bình lại thấp hơn đáng kể, tập trung vào GPU có yêu cầu thấp về tốc độ bộ nhớ.

Biểu đồ hộp của biến Memory Speed theo biến Shader

	Mean	Var	Sd	Min	Max	Q1	Q2	Q3
1	366.0000	*NA*	*NA*	366	366	366.00	366	366
1.4	292.8571	25357.14	159.2393	200	550	200.00	200	350
2	525.7179	104384.05	323.0852	200	1180	270.00	400	765
3	508.3537	38036.03	195.0283	200	1100	350.00	500	600
4	768.0052	45149.45	212.4840	350	1242	667.00	800	900
4.1	791.1270	32541.10	180.3915	400	1700	675.25	800	900
5	1300.2103	142964.01	378.1058	400	2127	1000.00	1253	1600

Hình 4.10: Các giá trị đặc trưng của biến Memory_Speed theo biến Shader

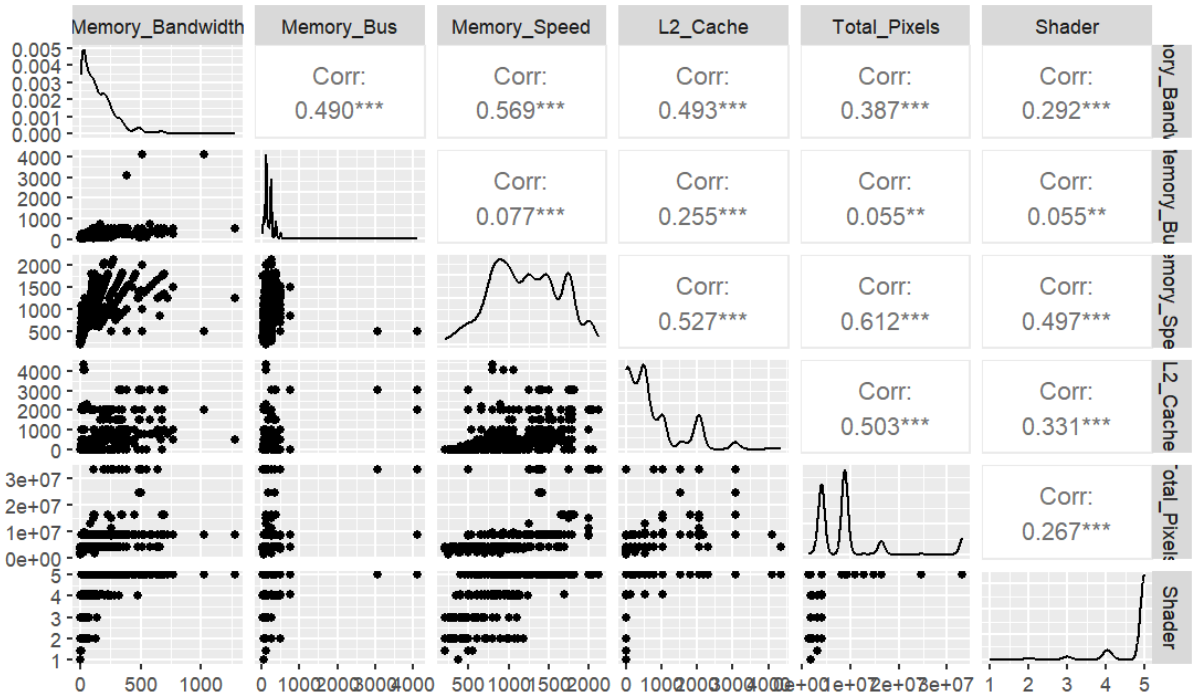


Hình 4.11: Biểu đồ hộp biến Memory_Speed theo biến Shader

Nhận xét: Biểu đồ cho thấy mối quan hệ rõ ràng giữa Shader và Memory Speed. Các giá trị trung vị (median) của Memory Speed đều tăng dần qua các mức Shader. Khi shader bằng 2 và 5, phạm vi giá trị Memory Speed khá rộng, cho thấy mức độ biến động lớn hơn. Tại giá trị 5 của biến shader, Memory Speed đạt đến giá trị cao nhất, với một phạm vi lớn nhất cả về hộp lẫn đường râu. Shader cao hơn thường tương ứng với Memory Speed lớn hơn, mặc dù có sự biến động tùy thuộc vào mức Shader cụ thể.

4.4 Biểu đồ tán xạ

Ta đi tìm mối liên hệ giữa các biến Memory_Bandwidth, Memory_Bus, Memory_Speed, L2_Cache, Total_Pixels, Shader có trong bộ dữ liệu bằng ma trận tương quan giữa các dữ liệu này.



Hình 4.12: Biểu đồ tán xạ mô tả mối quan hệ giữa các biến

Nhận xét:

- Memory_Speed và Total_Pixels (Corr = 0.612): Mối quan hệ này cho thấy tốc độ bộ nhớ (Memory_Speed) có xu hướng tăng khi tổng số điểm ảnh (Total_Pixels) tăng, phản ánh nhu cầu tốc độ xử lý cao khi độ phân giải cao. Hai biến này có quan hệ tuyến tính không quá mạnh.

- Memory_Speed và Memory_Bandwidth (Corr = 0.569): Mối quan hệ này thể hiện rằng tốc độ bộ nhớ càng cao thì băng thông bộ nhớ (Memory_Bandwidth) càng được cải thiện. Điều này phù hợp với nguyên tắc rằng tăng tốc độ bộ nhớ giúp dữ liệu được truyền tải nhanh hơn.

- Memory_Speed và L2_Cache (Corr = 0.527): Mối liên kết giữa hai biến này cho thấy các GPU có tốc độ bộ nhớ cao thường đi kèm với bộ nhớ đệm L2 lớn. Dù vậy, mối tương quan chỉ ở mức trung bình và vẫn có sự phân tán đáng kể trong dữ liệu.

- Memory_Speed và Shader (Corr = 0.497): Quan hệ này phản ánh rằng khi tốc độ bộ nhớ (Memory_Speed) tăng, giá trị Shader cũng tăng theo. Điều này có thể được

lý giải bởi Shader, vốn thực hiện các tác vụ xử lý đồ họa như ánh sáng, bóng, và kết cấu, sẽ hoạt động hiệu quả hơn khi tốc độ truyền tải dữ liệu cao.

Chương 5

THỐNG KÊ SUY DIỄN

5.1 Hồi quy tuyến tính đa biến

5.1.1 Chọn lọc dữ liệu

Chọn lọc dữ liệu là một bước quan trọng trong quá trình xây dựng mô hình hồi quy tuyến tính đa biến, giúp cải thiện hiệu quả và độ chính xác của mô hình. Trước khi bắt đầu việc xây dựng mô hình, việc lựa chọn và chuẩn bị dữ liệu hợp lý có thể quyết định mức độ thành công của mô hình hồi quy.

```
data < - read.csv("New_GPU_Data.csv") #Doc du lieu da duoc tien xu ly
colnames(data) #In ra cac cot trong bo du lieu
```

Ta thu được kết quả sau:

```
> colnames(data)
 [1] "Architecture" "Dedicated" "Direct_X" "Integrated" "L2_Cache" "
   Manufacturer" "Memory_Bandwidth" "Memory_Bus"
 [9] "Memory_Speed" "Memory_Type" "Name" "Notebook_GPU" "Open_GL" "
   Release_Date" "Resolution_WxH" "SLI_Crossfire"
[17] "Shader"
```

Nhóm nghiên cứu đặt mục tiêu là sử dụng mô hình hồi quy tuyến tính đa biến để phân tích các yếu tố có thể ảnh hưởng đến tốc độ bộ nhớ (Memory_Speed). Do đó, nhóm đã chọn ra một số đặc trưng có thể ảnh hưởng lên tốc độ xử lý của bộ nhớ. Cụ thể là các đặc trưng như

Dedicated, Integrated, L2_Cache, Manufacturer, Memory_Bandwidth, Memory_Bus, Memory_Type, Notebook_GPU, Open_GL, Resolution_WxH, SLI_Crossfire.

```
 # Chon loc du lieu
data_variables<-data[,c("Dedicated", "Integrated", "Manufacturer" ,"L2_
    Cache", "Memory_Bandwidth", "Memory_Bus",  "Open_GL", "Shader", "Memory_
    Type", "Notebook_GPU", "SLI_Crossfire", "Resolution_WxH",  "Memory_Speed
    ")
head(data_variables) #In ra 6 dong dau cua bo du lieu da chon loc
```

	Dedicated	Integrated	Manufacturer	L2_Cache	Memory_Bandwidth	Memory_Bus	Open_GL	Shader	Memory_Type
1	Yes	No	Nvidia	0	64.0	256	3.3	4	GDDR3
2	Yes	No	AMD	0	106.0	512	3.1	4	GDDR3
3	Yes	No	AMD	0	51.2	256	3.1	4	GDDR3
4	Yes	No	AMD	0	36.8	128	3.3	4	GDDR4
5	Yes	No	AMD	0	22.4	128	3.1	4	GDDR3
6	Yes	No	AMD	0	35.2	128	3.3	4	GDDR4

	Notebook_GPU	SLI_Crossfire	Resolution_WxH	Memory_Speed
1	No	Yes	2560x1600	1000
2	No	Yes	2560x1600	828
3	No	Yes	2560x1600	800
4	No	Yes	2560x1600	1150
5	No	Yes	2560x1600	700
6	No	Yes	2560x1600	1100

Hình 5.1: Dữ liệu trước khi xử lý

5.1.2 Mã hóa các biến phân loại

Để mã hóa các biến phân loại, ta sẽ sử dụng phương pháp **Label Encoding**, đây là phương pháp chuyển đổi các giá trị phân loại thành các số nguyên duy nhất. Mỗi giá trị phân loại sẽ được gán một số nguyên, giúp các mô hình học máy có thể xử lý dữ liệu này một cách trực tiếp.

Giả sử ta có biến phân loại "Màu sắc" với các giá trị: {Đỏ, Xanh, Vàng}. Sau khi áp dụng Label Encoding, dữ liệu sẽ được chuyển thành: {0, 1, 2} tương ứng với {Đỏ, Xanh, Vàng}. Nhóm nghiên cứu đã áp dụng phương pháp này cho các biến như Dedicated, Integrated, Manufacturer, Memory_Type, Notebook_GPU, SLI_Crossfire nhằm mục đích chuyển các đặc trưng phân loại kiểu chuỗi thành kiểu số, thuận lợi cho việc xác định hệ số hồi quy của mô hình.

Ví dụ về một đoạn mã nguồn nhằm hiện thực phương pháp Label Encoding cho đặc trưng Dedicated:

```
unique(data_variables$Dedicated) #Liet ke cac mau phan loai trong Dedicated
# Gan nhan cho "Yes" = 1 va "No" = 0
data_variables$Dedicated = factor(data_variables$Dedicated,
                                  levels = c("Yes", "No"),
                                  label = c(1, 0))
# Chuyen cac nhan ve kieu so
data_variables$Dedicated <- as.numeric(as.character(data_variables$
    Dedicated))
```

5.1.3 Xử lý đặc trưng không ở dạng chuẩn:

Đối với đặc trưng Resolution_WxH trong bộ dữ liệu đã được chọn lọc, các thành phần của đặc trưng không thuộc dạng phân loại kiểu chuỗi mà nó có kiểu chuỗi đặc biệt có cấu trúc "WxH" được dịch ra là "WidthxHeight".

Ví dụ: Như một thành phần của đặc trưng là "2560x1600" thì có ý nghĩa rằng độ phân giải của GPU này có chiều rộng là 2560 pixel và 1600 pixel.

Đoạn mã R nhằm hiện thực việc chia tách đặc trưng Resolution_WxH:

```
library(dplyr) #Cai dat cac thu vien can thiet
library(tidyr)
#Chia cot Resolution\_WxH thanh Resolution_Width va Resolution_Height
data_variables<- data_variables %>% separate(Resolution_WxH, c('Resolution_
    Width', 'Resolution_Height'), sep = "x")
#Bien doi cac cot ve dang so
data_variables$Resolution_Width <- as.numeric(data_variables$Resolution_
    Width)
data_variables$Resolution_Height <- as.numeric(data_variables$Resolution_
    Height)
```

5.1.4 Kết quả sau khi thực hiện tiền xử lý

Nhờ những phương pháp xử lý dữ liệu đã đề cập ở phía trên, bộ dữ liệu sau khi xử lý đã có thể sử dụng cho mô hình hồi quy tuyến tính đa biến.

```
head(data\_variables) #In 6 dong dau cua bo du lieu sau khi xu ly
```

	Dedicated	Integrated	Manufacturer	L2_Cache	Memory_Bandwidth	Memory_Bus	Open_GL	Shader	Memory_Type
1	1	0	0	0	64.0	256	3.3	4.0	0
2	1	0	1	0	106.0	512	3.1	4.0	0
4	1	0	1	0	36.8	128	3.3	4.0	1
5	1	0	1	0	22.4	128	3.1	4.0	0
6	1	0	1	0	35.2	128	3.3	4.0	1
7	1	0	1	0	134.4	256	3.3	4.1	2

	Notebook_GPU	SLI_Crossfire	Resolution_Width	Resolution_Height	Memory_Speed
1	0	1	2560	1600	1000
2	0	1	2560	1600	828
4	0	1	2560	1600	1150
5	0	1	2560	1600	700
6	0	1	2560	1600	1100
7	0	1	2560	1600	1050

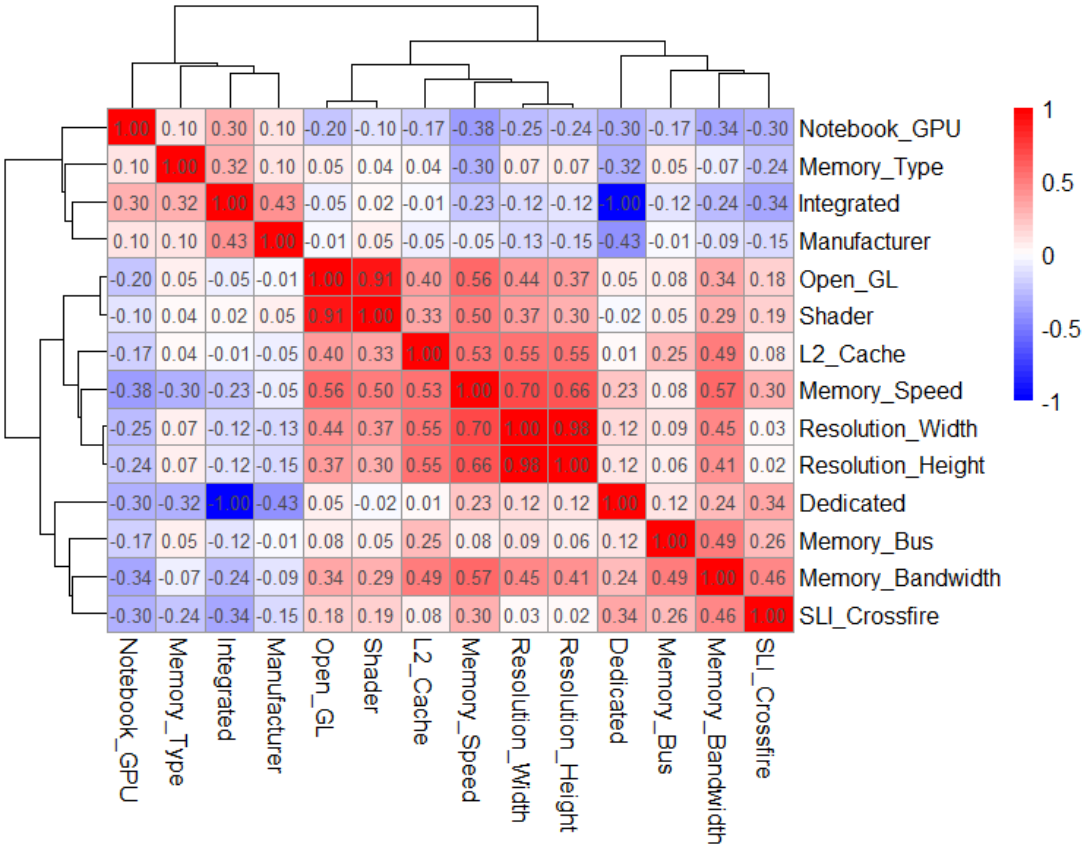
Hình 5.2: Dữ liệu sau khi xử lý

5.1.5 Kiểm tra hiện tượng đa cộng tuyến giữa các đặc trưng mẫu

Hiện tượng đa cộng tuyến (multicollinearity) xảy ra trong mô hình hồi quy khi các biến độc lập có tương quan cao với nhau. Điều này gây ra khó khăn trong việc xác định ảnh hưởng riêng biệt của từng biến độc lập đến biến phụ thuộc.

Do đó, để loại bớt những đặc trưng có mối liên hệ với nhau, nhóm nghiên cứu đề xuất sử dụng ma trận tương quan và biểu diễn bằng heatmap nhằm kiểm tra có tồn tại hiện tượng đa cộng tuyến giữa các đặc trưng của mô hình. Nhờ có sự kiểm tra và loại bỏ các đặc trưng không phù hợp, mô hình có thể được đảm bảo rằng các đặc trưng của nó là độc lập với nhau.

```
#Tinh toan ma tran tuong quan
cor_matrix <- cor(data_variables)
#Thu vien bieu dien heatmap
library(pheatmap)
#Ve heatmap
pheatmap(cor_matrix,
         display_numbers = TRUE,
         color = colorRampPalette(c("blue", "white", "red"))(50),
         main = "Heatmap of Correlation Matrix",
         fontsize = 12)
```



Hình 5.3: Biểu đồ nhiệt biểu diễn tương quan giữa các đặc trưng của mô hình

Kết quả sau khi vẽ biểu đồ nhiệt:

Biểu đồ nhiệt trên đã biểu diễn tốt các mức độ tương quan giữa các đặc trưng của mô hình. Dựa vào màu sắc và những con số trên từng ô của biểu đồ nhiệt. Tuy nhiên, nhóm nghiên cứu tập trung loại bỏ những đặc trưng có mức độ tương quan rất cao (từ 0.8 trở nên và -0.8 trở xuống) vì các đặc trưng này gần như có quan hệ tuyến tính với nhau như:

- Open_GL và Shader với mức tương quan 0.91.
- Resolution_Width và Resolution_Height với mức tương quan 0.98.
- Interated và Dedicated với mức tương quan -1.00.

Đối với những đặc trưng trên có mối tương quan rất cao chỉ nên lựa chọn 1 trong 2 đặc trưng để huấn luyện với mô hình.

5.1.6 Phân chia dữ liệu

Với tập dữ liệu gồm 3036 mẫu, đây là một con số đủ lớn giúp nhóm nghiên cứu có thể sử dụng để chia dữ liệu thành các tập huấn luyện và kiểm tra với 80% mẫu dùng làm tập kiểm thử và 20% mẫu làm tập kiểm tra.

```
#Caret de su dung chuc nang createDataPartition()
library(caret)
#Tim vi tri de chia tap du lieu dua theo bien muc tieu Memory_Speed
splitIndex <- createDataPartition(data_variables$Memory_Speed, p = 0.8,
    list = FALSE)
#Tach tap huan luyen
train_data <- data_variables[splitIndex, ]
#Tach tap kiem tra
test_data <- data_variables[-splitIndex, ]
```

Kết quả:

Tập dữ liệu	**Số lượng quan sát và biến**
Tập huấn luyện (train_data)	2431 quan sát, 14 biến
Tập kiểm thử (test_data)	605 quan sát, 14 biến

Bảng 5.1.1: Số lượng quan sát và biến trong các tập dữ liệu

5.1.7 Xây dựng mô hình

Một trong những công cụ quan trọng trong R là hàm lm(), được sử dụng để xây dựng mô hình hồi quy tuyến tính. Cách sử dụng cơ bản của hàm `lm()` trong R như sau:

```
model <- lm(y ~ x1 + x2 + x3, data = dataset)
```

Trong đó:

- `y` là biến phụ thuộc.
- `x1`, `x2`, `x3` là các biến độc lập.
- `dataset` là tập dữ liệu chứa các biến.

Nhằm đáp ứng cho mục tiêu là sử dụng mô hình hồi quy tuyến tính đa biến để phân tích các yếu tố có thể ảnh hưởng đến tốc độ bộ nhớ (Memory_Speed) của GPU. Nhóm nghiên cứu đề xuất xây dựng 2 mô hình hồi quy tuyến tính đa biến là **model1** và **model2**:

- **model1**: là mô hình sử dụng toàn bộ các đặc trưng của bộ dữ liệu dùng để xây dựng mô hình (tức là bộ dữ liệu chưa loại bỏ các đặc trưng bị đa cộng tuyến cũng như những biến chưa được kiểm định hệ số hồi quy).
- **model2**: là mô hình sau khi đã kiểm định thống kê các hệ số hồi quy của **model1**, loại bỏ các đặc trưng bị đa cộng tuyến và các đặc trưng không cần thiết sau khi kiểm định hệ số hồi quy. Đây cũng là mô hình được sử dụng để đưa ra dự đoán cho mẫu kiểm thử.

Việc xây dựng hai mô hình hồi quy tuyến tính không chỉ giúp phân tích mức độ ảnh hưởng của các đặc trưng lên tốc độ bộ nhớ của GPU mà còn nhằm so sánh hiệu suất giữa chúng. Điều này chứng minh rằng việc sử dụng các kỹ thuật kiểm định thống kê để kiểm định hệ số hồi quy nhằm loại bỏ các biến không cần thiết có thể giữ nguyên hiệu suất mô hình, đồng thời giúp giảm độ phức tạp tính toán, cải thiện khả năng tổng quát hóa và tránh hiện tượng đa cộng tuyến, từ đó tăng độ tin cậy và khả năng diễn giải của mô hình.

Đoạn mã nhằm xây dựng **model1**:

```
# Xay dung model1 voi day du cac dac trung
model1 <- lm(Memory_Speed~Dedicated + Integrated + Manufacturer + L2_Cache
    +
          Memory_Bandwidth + Memory_Bus + Open_GL + Shader + Memory_Type +
          Notebook_GPU + SLI_Crossfire + Resolution_Width + Resolution_
    Height,
          train_data)
```

Đoạn mã nhằm xây dựng **model2**:

```
# Xay dung model2 khi da loai bo cac dac trung sau khi kiem dinh gia thuyet
    thong ke
model2 <- lm(Memory_Speed~ Manufacturer + L2_Cache + Memory_Bandwidth +
             Memory_Bus + Open_GL + Memory_Type + Notebook_GPU + SLI_
   Crossfire +
             Resolution_Width, train_data)
```

5.1.8 Tổng quan về model1

```
summary(model1)
```

Sau khi thực thi đoạn mã trên, R cho chúng ta biết tổng quan về mô hình **model1** đã được xây dựng.

```
    Residuals:
        Min       1Q   Median       3Q      Max
    -953.04 -129.34    11.49   142.61   934.85

    Coefficients: (1 not defined because of singularities)
                        Estimate Std. Error t value Pr(>|t|)
    (Intercept)         7.412e+01  5.067e+01   1.463    0.144
    Dedicated           1.300e+01  2.033e+01   0.639    0.523
    Integrated                 NA         NA      NA       NA
    Manufacturer        5.821e+01  6.514e+00   8.937  < 2e-16 ***
    L2_Cache            5.188e-02  7.385e-03   7.026 2.76e-12 ***
    Memory_Bandwidth    6.163e-01  4.645e-02  13.268  < 2e-16 ***
    Memory_Bus         -2.288e-01  2.254e-02 -10.149  < 2e-16 ***
    Open_GL             1.274e+02  1.590e+01   8.014 1.70e-15 ***
    Shader              2.499e+01  1.892e+01   1.321    0.187
    Memory_Type        -8.107e+01  2.845e+00 -28.500  < 2e-16 ***
    Notebook_GPU       -1.066e+02  1.197e+01  -8.910  < 2e-16 ***
    SLI_Crossfire       7.978e+01  1.067e+01   7.475 1.07e-13 ***
    Resolution_Width    1.243e-01  1.512e-02   8.216 3.38e-16 ***
    Resolution_Height   2.658e-02  2.748e-02   0.967    0.333
    ---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 202.1 on 2418 degrees of freedom
Multiple R-squared:  0.7717,  Adjusted R-squared:  0.7706
F-statistic: 681.2 on 12 and 2418 DF,  p-value: < 2.2e-16
```

Giải thích một số giá trị quan trọng

- **Residuals**: Residuals là phần sai số giữa giá trị dự đoán của mô hình và giá trị thực tế.
- **Coefficients** :Phần này liệt kê các hệ số hồi quy của mô hình cho từng biến độc lập.
- **Residual Standard Error:** Sai số chuẩn của residuals (202.1) biểu thị mức độ sai lệch của giá trị dự đoán so với giá trị thực tế.
- **Multiple R-squared:** (0.7717): Đo lường tỷ lệ phương sai của biến phụ thuộc được giải thích bởi các biến độc lập. Giá trị R^2 cao cho thấy mô hình phù hợp tốt.
- **Adjusted R-squared:** (0.7706): Là phiên bản điều chỉnh của R^2, giảm thiểu việc phóng đại độ phù hợp khi thêm biến không cần thiết.
- **F-statistic và p-value:** Giá trị *F-statistic* (681.2) kiểm tra giả thuyết rằng tất cả các hệ số hồi quy (trừ *intercept*) bằng 0. Giá trị p rất nhỏ ($< 2.2 \times 10^{-16}$) cho thấy ít nhất một biến độc lập có ảnh hưởng đáng kể đến biến phụ thuộc.

Kiểm định mô hình model1

Nhằm phân tích mức độ ảnh hưởng của các đặc trưng tới biến mục tiêu, ta đặt ra giả thiết kiểm định và đối giả thiết kiểm định cho mô hình hồi quy tuyến tính đa biến:

- $H_0 : \beta_0 = \beta_1 = \beta_2 = ... = \beta_k = 0$ (Mô hình không có ý nghĩa).
- $H_1 : \exists \beta_i \neq 0, \text{với } i \in [1, k]$ (Mô hình có ý nghĩa/Biến mục tiêu phụ thuộc ít nhất vào 1 đặc trưng)

Do đã đề cập ở phần trên, giá trị p-value của toàn bộ mô hình rất nhỏ ($< 2.2 \times 10^{-6}$) so với mức ý nghĩa là 0.05 cho thấy ít nhất một biến độc lập có ảnh hưởng đáng kể đến biến phụ thuộc. Từ đó nhóm nghiên cứu đủ điều kiện để bác bỏ giả thiết H_0. Kết luận rằng mô hình có ý nghĩa.

Kiểm định bằng p-value với mức ý nghĩa 5% cho từng đặc trưng trong mô hình nhằm loại bỏ những đặc trưng không cần thiết trong model1

Tuy nhiên, mô hình vẫn còn một số hạn chế do chưa có phương pháp kết luận mức độ ảnh hưởng của một đặc trưng lên biến mục tiêu là có ảnh hưởng hay không ảnh hưởng. Cũng như chưa giải quyết vấn đề đa cộng tuyến xảy ra khiến cho mô hình không xác định được hệ số hồi quy cho một hoặc nhiều đặc trưng. Vì thế nhóm nghiên cứu cần đi kiểm định hệ số hồi quy cho từng đặc trưng.

Ta có giả thiết kiểm định thống kê như sau:

- $\beta_j = 0$, với $j \in [1, k]$ (Đặc trưng j không có ảnh hưởng tới biến mục tiêu)
- $\beta_j \neq 0$, với $j \in [1, k]$ (Đặc trưng j có ảnh hưởng tới biến mục tiêu)

Nhờ có phương pháp kiểm định đã đề ra, kết hợp với giá trị p-value mà ngôn ngữ R đã tính toán sẵn cho từng đặc trưng ở cột Pr(>|t|) như trên. Nhóm nghiên cứu có thể đưa ra kết luận rằng nếu giá trị p-value cho hệ số hồi quy của một biến độc lập lớn hơn mức ý nghĩa 0.05 thì ta không có đủ bằng chứng để có thể bác bỏ giả thiết H_0. Từ phương pháp trên, một số đặc trưng không có ảnh hưởng tới biến mục tiêu bao gồm:

- **Dedicated** với p-value = 0.523.
- **Shader** với p-value = 0.0.187.
- **Resoolution_Height** với p-value = 0.333.

Từ đó nhóm nghiên cứu đã xây dựng **model2** là mô hình đã loại bỏ các biến ở trên, đồng thời cũng loại bỏ các đặc trưng **Integrated** đã đề cập ở phần kiểm tra hiện tượng đa cộng tuyến. Lưu ý rằng, loại bỏ đặc trưng **Shader** và **Resolution_Height** cũng đã giải quyết vấn đề đa cộng tuyến với đặc trưng **Open_GL**.

5.1.9 Tổng quan về model2

```
summary(model2)
```

Sau khi thực thi đoạn mã trên, dưới đây chính là thông tin về **model2**

```
Residuals:
    Min      1Q  Median      3Q     Max
-946.93 -130.58   11.07  140.47  957.35

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)       1.407e+02  2.851e+01   4.934 8.59e-07 ***
Manufacturer      5.681e+01  5.919e+00   9.599  < 2e-16 ***
L2_Cache          5.234e-02  7.139e-03   7.332 3.08e-13 ***
Memory_Bandwidth  6.105e-01  4.578e-02  13.336  < 2e-16 ***
Memory_Bus       -2.299e-01  2.247e-02 -10.233  < 2e-16 ***
Open_GL           1.439e+02  6.821e+00  21.091  < 2e-16 ***
Memory_Type      -8.144e+01  2.745e+00 -29.668  < 2e-16 ***
Notebook_GPU     -1.051e+02  1.159e+01  -9.065  < 2e-16 ***
SLI_Crossfire     8.312e+01  1.042e+01   7.980 2.24e-15 ***
Resolution_Width  1.387e-01  3.835e-03  36.156  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 202.1 on 2421 degrees of freedom
Multiple R-squared:  0.7715,  Adjusted R-squared:  0.7706
F-statistic: 908.1 on 9 and 2421 DF,  p-value: < 2.2e-16
```

Nhờ có sự chọn lọc các đặc trưng nhờ phương pháp kiểm định thống kê mà **model1** và **model2** có mức độ phù hợp khá tương đồng, với R^2 lần lượt là 0.7715 và 0.7706, và sai số chuẩn của residuals gần như bằng nhau (202.1 ở Model1 và 202.1 ở Model2).

Kiểm định model2

Với cặp giả thiết thống kê:

- $H_0 : \beta_0 = \beta_1 = \beta_2 = ... = \beta_k = 0$ (Mô hình không có ý nghĩa).

- $H_1 : \exists \beta_i \neq 0$, với $i \in [1, k]$ (Mô hình có ý nghĩa/Biến mục tiêu phụ thuộc ít nhất vào 1 đặc trưng)

Kết luận: Với giá trị p-value của model2 rất nhỏ ($p < 2.2 \times 10^{-6}$). Nhóm nghiên cứu bác bỏ giả thiết H_0. Kết luận mô hình thứ 2 có ý nghĩa.

Kiểm định hệ số hồi quy cho các đặc trưng của model2

Ta có giả thiết kiểm định thống kê cho một đặc trưng như sau:

- $\beta_j = 0$, với $j \in [1, k]$ (Đặc trưng j không có ảnh hưởng tới biến mục tiêu)
- $\beta_j \neq 0$, với $j \in [1, k]$ (Đặc trưng j có ảnh hưởng tới biến mục tiêu)

Sau khi đã loại bỏ các đặc trưng không phù hợp của model1. Các đặc trưng còn lại được sử dụng trong model2 có giá trị p-value rất nhỏ và không có đặc trưng nào có giá trị p-value lớn hơn mức ý nghĩa 5% cho nên nhóm nghiên cứu bác bỏ giả thiết H_0 và kết luận tất cả các đặc trưng trong model2 đều có ảnh hưởng tới Memory_Speed.

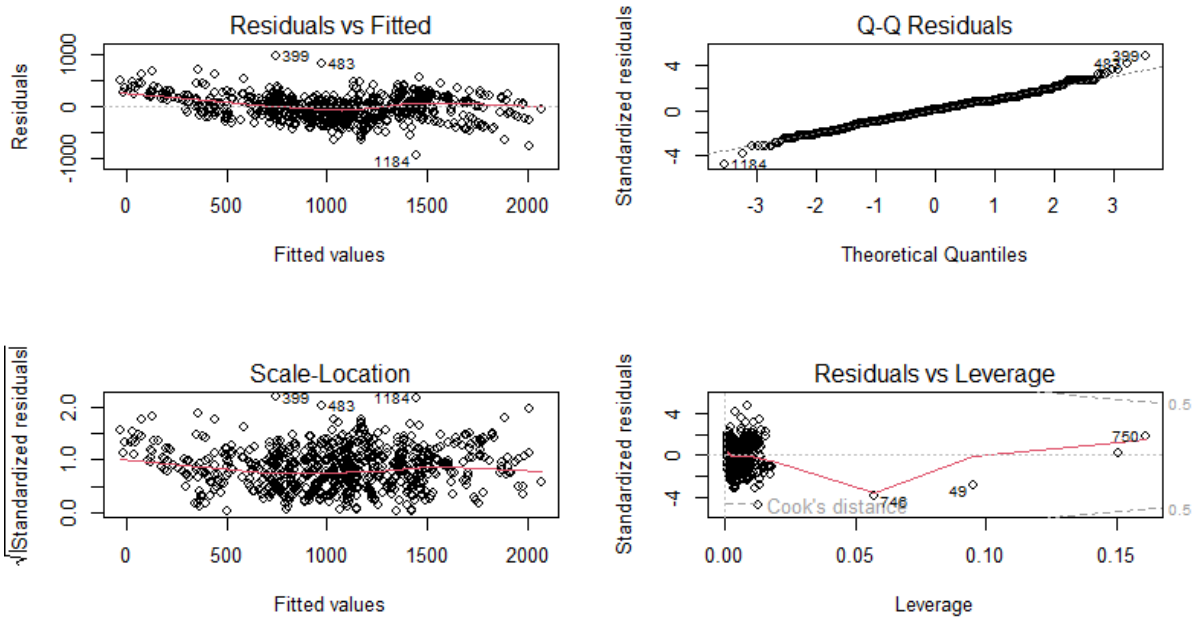
Kiểm tra các giả định của model2

Trước khi sử dụng model2 vào mục đích dự đoán tốc độ của GPU, trước tiên phải cần kiểm tra các giả định về phần dư của mô hình. Nhóm nghiên cứu đã đặt ra những giả định cho model2 như sau:

- Phân phối phần dư (Residuals): Kiểm tra xem phần dư có phân phối chuẩn hay không.
- Tính đồng nhất phương sai (Homoscedasticity): Kiểm tra xem phương sai của phần dư có đồng đều hay không.
- Tính tuyến tính (Linearity): Kiểm tra xem mối quan hệ giữa biến phụ thuộc và các biến độc lập có tuyến tính hay không.
- Không có tự tương quan (No Autocorrelation): Kiểm tra xem các phần dư có tương quan với nhau hay không.

Ta vẽ đồ thị phần dư bằng lệnh này sẽ tạo ra 4 đồ thị để kiểm tra mô hình:

```
par(mfrow = c(2, 2)) #Chia bo cuc thanh 2x2
plot(model2)
```



Hình 5.4: Các đồ thị biểu diễn tổng quát phần dư của mô hình

Nhận xét:

- Đối với đồ thị Residual vs Fitted: Ta có thể thấy dữ liệu phân đố ngẫu nhiên khá đồng đều bao quanh đường zero. Ta có thể kết luận phương sai của phần dư đồng đều.
- Đối với đồ thị Normal Q-Q của phần dư. Ta nhận thấy các điểm dữ liệu tập hợp rất gần đường thẳng lý tưởng nên có thể kết luận phần dư có phân phối chuẩn. Tuy nhiên vẫn cần một phương pháp khác đáng tin cậy hơn để đưa ra kết luận.
- Đối với đồ thị Scale-Location: các điểm dữ liệu khá sát nhau và phân bố dàn trải xung quanh đường y = 1. Ta có thể kết luận phương sai của phần dư tương đối đồng nhất.
- Đối với đồ thị Residual vs Leverage: Các điểm đồ thị tập trung sát nhau và lệch về phía trái của đồ thị và chỉ có 4 điểm dữ liệu ngoại lai nằm lệch về phía bên phải.

Kết luận rằng mô hình vẫn có tồn tại ngoại lai trong các đặc trưng của nó nhưng không gây ra quá nhiều ảnh hưởng bởi số lượng ngoại lai rất ít.

Để kiểm tra hiện tượng đa cộng tuyến (Multicollinearity) giữa các biến độc lập: ta thực hiện tính hệ số VIF (Variance Inflation Factor) để kiểm tra đa cộng tuyến giữa các biến độc lập. Nếu giá trị VIF của một biến vượt quá 5 hoặc 10, có thể có đa cộng tuyến giữa biến đó và các biến khác.

```
library(car) #Cai dat thu vien can thiet de su dung VIF
vif(model2)
```

Kết quả:

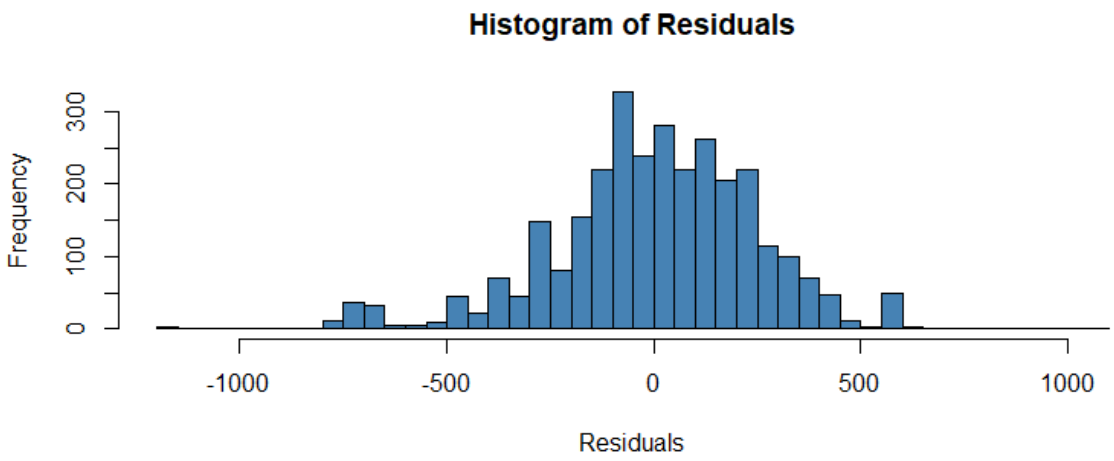
```
    Manufacturer          L2_Cache Memory_Bandwidth        Memory_Bus
        1.046878          1.666720          2.258313          1.418015
         Open_GL       Memory_Type     Notebook_GPU    SLI_Crossfire
        1.355964          1.093381          1.210501          1.504988
Resolution_Width
        1.815687
```

Trong tất cả những đặc trưng nhóm nghiên cứu đã chọn lọc cho model2 đã không xảy ra hiện tượng đa cộng tuyến do không có giá trị VIF nào vượt quá 5 và 10.

Ngoài ra ta có thể vẽ biểu đồ histogram để kiểm tra phân phối của phần dư:

```
par(mfrow = c(1, 1))
hist(residuals(model), breaks = 50, main = "Histogram of Residuals",
     xlab = "Residuals", col = "steelblue")
```

Kết quả:



Hình 5.5: Đồ thị histogram cho phần dư

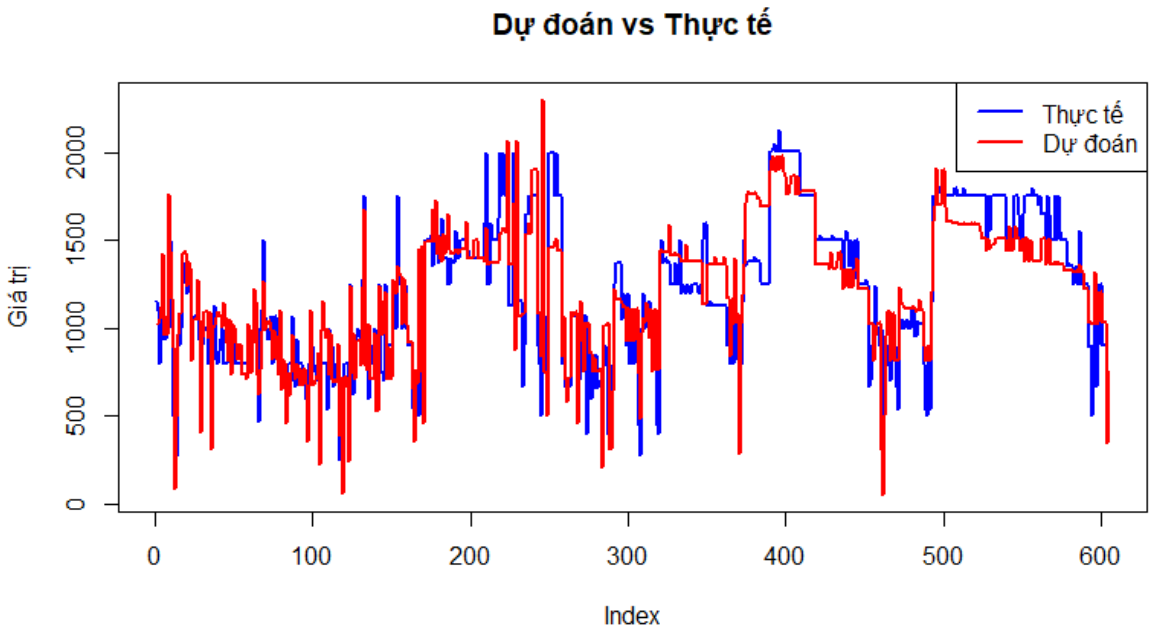
Từ đồ thị, phần dư của mô hình có phân phối khá chuẩn, dữ liệu tập trung có dạng hình chuông úp ngược khá tương đồng với đồ thị phân phối Gauss.

5.1.10 Kết quả dự đoán của mô hình hồi quy tuyến tính

Dưới đây là đoạn mã nhằm hiện thực chức năng dự đoán của mô hình.

```
#Tao cot Memory_Speed_Prediction moi trong tap test
test_data$Memory_Speed_Prediction <- predict(model2, test_data)
#Gia tri thuc te
actual <- test_data$Memory_Speed
#Gia tri du doan
predicted <- test_data$Memory_Speed_Prediction
#Ve do thi tuong quan giua du doan va thuc te
par(mfrow = c(1, 1))
plot(actual, type = "l", col = "blue", lwd = 2,
    xlab = "Index", ylab = "Gia tri",
     main = "Du doan vs Thuc te", ylim = range(c(actual, predicted)))
lines(predicted, col = "red", lwd = 2)
legend("topright", legend = c("Thuc te", "Du doan"),
       col = c("blue", "red"), lty = 1, lwd = 2)
```

Dưới đây là kết quả dự đoán của model 2 trên 605 quan sát của tập kiểm thử:



Hình 5.6: Đồ thị histogram cho phần dư

Mô hình hồi quy tuyến tính đa biến model2 với hệ số $R^2 = 0.77$ cho thấy mô hình giải thích được 77% sự biến thiên của biến phụ thuộc dựa trên các biến độc lập trong dữ liệu. Điều này chứng tỏ rằng mô hình đã học được mối quan hệ đáng kể giữa các biến, phản ánh sự phù hợp khá cao. Tuy nhiên, vẫn còn 23% phương sai chưa được giải thích, có thể do các yếu tố khác chưa được đưa vào mô hình hoặc do sai số ngẫu nhiên.

Ngoài ra ta có thể đánh giá mô hình thông qua giá trị RMSE (Root Mean Squared Error):

```
print(rmse(actual, predicted))
```

Kết quả:

```
> print(rmse(actual, predicted))
[1] 207.0712
```

Tổng kết:

Mô hình hồi quy có mức độ dự đoán tương đối tốt với $R^2 = 0.77$, tức là mô hình có thể giải thích được phần lớn sự biến động của Memory_Speed.

Tuy nhiên, RMSE = 207.0712 cho thấy vẫn còn sai số trong dự đoán. Mức độ này có thể chấp nhận được hoặc không, tùy thuộc vào đặc điểm của dữ liệu thực tế.

5.2 Ước lượng một mẫu

5.2.1 Giới thiệu bài toán

Bài toán này nhằm giúp người học hiểu rõ các khái niệm cơ bản trong lý thuyết ước lượng và biết cách áp dụng chúng để xây dựng, giải quyết một bài toán thực tế. Cụ thể, yêu cầu đặt ra là ước lượng khoảng tin cậy hai phía cho giá trị trung bình tổng thể của biến Memory_Speed trong bộ dữ liệu đã qua tiền xử lý, với mức độ tin cậy là 95%. Vì mẫu được chọn có kích thước lớn và phương sai tổng thể chưa được biết, chúng ta sẽ sử dụng các công cụ và câu lệnh trong ngôn ngữ R, cùng với những kiến thức về xác suất và thống kê, để tính toán và đưa ra khoảng tin cậy cho trung bình tổng thể của mẫu.

5.2.2 Kiểm tra phân phối chuẩn

Trước khi tiến hành kiểm định thống kê, một bước quan trọng là xác định xem dữ liệu có tuân theo phân phối chuẩn hay không. Trong bài toán này, chúng ta sẽ kiểm tra tính chuẩn của biến Memory_Speed bằng cách sử dụng biểu đồ QQ plot và kiểm định Shapiro-Wilk. Kiểm định Shapiro-Wilk cho phép xác định tính chuẩn của dữ liệu thông qua việc kiểm tra xem dữ liệu có khác biệt đáng kể nào so với phân phối chuẩn không. Đồng thời, biểu đồ QQ plot sẽ cung cấp trực quan về mức độ phù hợp của dữ liệu so với phân phối chuẩn. Bước kiểm tra này giúp đảm bảo rằng các giả định về phân phối của dữ liệu là hợp lý, từ đó tăng độ tin cậy cho các kết quả thống kê trong phân tích.

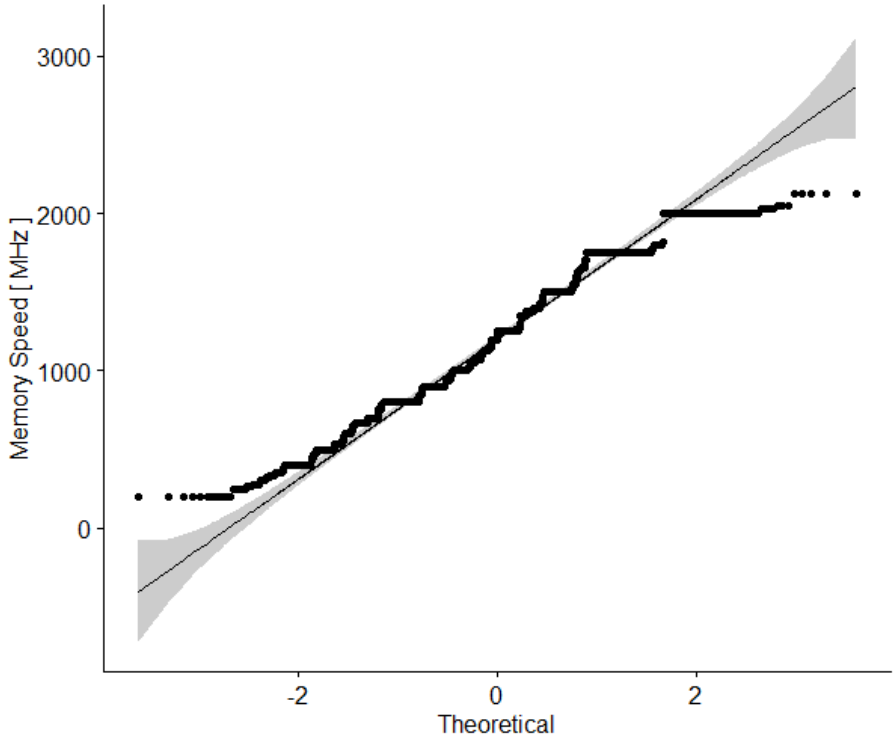
Kiểm tra phân phối chuẩn bằng biểu đồ Q-Q

Trước tiên, chúng ta sẽ vẽ biểu đồ Q-Q để kiểm tra xem mẫu có tuân theo phân phối chuẩn hay không. Ở đây, ta dùng hàm ggqqplot() để vẽ Q-Q plot, hàm này yêu cầu ta cài đặt các gói ggpubr và ggplot2. Dưới đây là đoạn mã R để thực hiện việc này:

```
# Ve bieu do Q-Q plot
ggqqplot (GPU_data$Memory_Speed, ylab = "Memory Speed [ MHz ]")
```

Kết quả của đoạn code cho ra hình ảnh sau

- Trục hoành (Theoretical): Đại diện cho các phân vị của phân phối chuẩn lý thuyết.



Hình 5.7: Biểu đồ Q-Q plot cho mẫu Memory_Speed

- Trục tung (Memory Speed [MHz]): Đại diện cho các phân vị của dữ liệu mẫu (Memory_Speed).

Từ biểu đồ trên ta có nhận xét sau: các điểm dữ liệu đa phần nằm gần đường chéo cho thấy sự tương đồng giữa dữ liệu thực tế và phân phối chuẩn, đặc biệt là ở phần trung tâm. Điều này chứng tỏ rằng trung tâm của dữ liệu gần với giả định phân phối chuẩn. Tuy nhiên, tại hai đầu của biểu đồ, các điểm dữ liệu lệch đáng kể khỏi đường chéo. Sự sai lệch này cho thấy phần đuôi dữ liệu không hoàn toàn tuân theo phân phối chuẩn

Kiểm tra phân phối chuẩn bằng kiểm định Shapiro - Wilk

- **Giả thuyết null (H_0):** Dữ liệu tuân theo phân phối chuẩn.
- **Giả thuyết đối (H_1):** Dữ liệu không tuân theo phân phối chuẩn.

Khi thực hiện kiểm định này, kết quả sẽ trả ra hai giá trị W và p-value, ở đây ta sẽ tập trung vào phân tích giá trị p-value để xác định kết quả của kiểm định. Theo đó,

- Nếu p-value $\geq$ mức ý nghĩa α (thường lấy giá trị 0.05), chưa có đủ bằng chứng để

bác bỏ giả thuyết H_0. Điều này có nghĩa là dữ liệu có thể được coi là có phân phối chuẩn.

- Nếu giá trị p < mức ý nghĩa α, ta bác bỏ giả thuyết H_0, thừa nhận giả thuyết H_1. Do vậy, ta có thể kết luận dữ liệu không tuân theo phân phối chuẩn.

Để thực hiện kiểm định Shapiro - Wilk, ta nhập đoạn mã sau vào R:

```
# Kiem dinh phan phoi chuan bang kiem dinh Shapiro-Wilk
shapiro.test(GPU_data$Memory_Speed)
```

Áp dụng kiểm định Shapiro - Wilk cho mẫu Memory_Speed cho ra kết quả sau:

```
        Shapiro-Wilk normality test

data:  GPU_data$Memory_Speed
W = 0.97597, p-value < 2.2e-16
```

Hình 5.8: Kết quả kiểm tra Shapiro-Wilk cho mẫu Memory_Speed

Kết quả của hàm kiểm định `shapiro.test()` cho ta thấy giá trị của p-value rất nhỏ, gần như bằng 0 (p-value < 2.2 e-16). Từ đó, ta đủ cơ sở để khẳng định rằng dữ liệu này **không tuân theo phân phối chuẩn**.

⇒ **Kết luận:** Cả Q-Q plot và hàm kiểm định `shapiro.test()` đều cho thấy rằng dữ liệu của mẫu **Memory_Speed** không tuân theo phân phối chuẩn.

5.2.3 Tiến hành ước lượng

Trong thống kê, phương pháp ước lượng khoảng tin cậy cho trung bình tổng thể của một mẫu phụ thuộc vào các yếu tố như phân phối tổng thể và tính chất của phương sai. Trong R, chúng ta sử dụng các hàm kiểm định và ước lượng khoảng tin cậy khác nhau tùy thuộc vào từng trường hợp đã được đề cập trong phần lý thuyết:

- Tổng thể tuân theo phân phối chuẩn, đã biết phương sai, sử dụng hàm, `z.test()`.
- Tổng thể tuân theo phân phối chuẩn, không biết phương sai, sử dụng hàm `t.test()`.

- Tổng thể không tuân theo phân phối chuẩn nhưng cỡ mẫu lớn thì theo *Định lý giới hạn trung tâm*, phân phối của trung bình mẫu tiến gần về phân phối chuẩn khi kích cỡ mẫu tăng lên , do vậy, ta có thể dùng hàm `z.test()` để khảo sát. Đồng thời, khi cỡ mẫu lớn, phân phối chuẩn và phân phối Student xấp xỉ nhau, cho phép ta dùng `t.test()` để khảo sát mẫu.

Ta đã kết luận rằng mẫu **Memory_Speed** có kích thước lớn, tổng thể của mẫu này không tuân theo phân phối chuẩn, chưa biết phương sai tổng thể. Do đó, ta dùng hàm `t.test()` để tìm khoảng tin cậy cho trung bình tổng thể của mẫu này.

```
# Ap dung t.test() cho mau voi do tin cay (conf.level) la 0.95
t.test(GPU_data$Memory_Speed, conf.level = 0.95)
```

5.2.4 Nhận xét và kết luận

Sau khi sử dụng kiểm định `t.test()` nhóm đã thu được kết quả sau:

```
        One Sample t-test

data:  GPU_data$Memory_Speed
t = 158.8, df = 3035, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 1196.154 1226.062
sample estimates:
mean of x
 1211.108
```

Hình 5.9: Kết quả chạy hàm `t.test()`

Vì đây là bài toán tìm khoảng ước lượng cho trung bình tổng thể nên ta chỉ cần quan tâm đến dòng `95 percent confidence interval` trở đi. Kết quả này cho ta biết rằng với độ tin cậy 95% , khoảng ước lượng hai phía cho trung bình tổng thể của **Memory_Speed** là

$$(1196.154; 1226.062)$$

Dựa trên kết quả của bài toán, nhóm đã tính toán và xác định được khoảng tin cậy hai phía cho trung bình tổng thể của **Memory_Speed** với độ tin cậy 95%. Khoảng tin

cậy này là (1196.154; 1226.062), cho thấy rằng với xác suất 95%, trung bình tổng thể của **Memory_Speed** nằm trong khoảng từ **1196.154** đến **1226.062**.

Quá trình thực hiện đã cho thấy sự thành công trong việc áp dụng lý thuyết ước lượng vào thực tế thông qua việc sử dụng hàm `t.test()` trong R để tính toán khoảng ước lượng với độ tin cậy 95%, trong trường hợp không biết phương sai tổng thể. Kết quả này minh họa hiệu quả của phương pháp ước lượng thống kê, giúp đưa ra các kết luận về tổng thể từ dữ liệu mẫu, từ đó hỗ trợ cho các phân tích và quyết định tiếp theo.

Tuy nhiên, do tỷ lệ khuyết thiếu trong mẫu dữ liệu khá lớn, các giá trị khuyết thiếu đã bị loại bỏ, nên không thể loại trừ khả năng kết quả thu được có thể có sự sai khác đôi chút so với kết quả chính xác.

5.3 Kiểm định hai mẫu

5.3.1 Giới thiệu bài toán

Tốc độ xử lý bộ nhớ là một trong những yếu tố quan trọng nhất quyết định hiệu suất của các hệ thống máy tính hiện đại, đặc biệt là trong các ứng dụng yêu cầu sức mạnh tính toán lớn như đồ họa, học máy, và mô phỏng khoa học. Sự cạnh tranh về hiệu suất giữa các nhà sản xuất GPU hàng đầu như Nvidia và AMD đã làm nổi bật tầm quan trọng của yếu tố này, khi người dùng và các chuyên gia không ngừng tìm kiếm câu trả lời cho câu hỏi: liệu sự khác biệt giữa các thương hiệu có thực sự mang lại ưu thế rõ rệt về tốc độ xử lý bộ nhớ?

Trong thế giới công nghệ không ngừng đổi mới, mỗi cải tiến về tốc độ xử lý bộ nhớ đều có thể mang lại những thay đổi đáng kể, từ cải thiện trải nghiệm người dùng đến tối ưu hóa hiệu quả vận hành của hệ thống. Nvidia và AMD, hai gã khổng lồ trong lĩnh vực sản xuất GPU, đều đã xây dựng những sản phẩm nổi bật với những đặc trưng riêng về hiệu suất và độ tin cậy. Mặc dù cả hai thương hiệu đều được đánh giá cao, vẫn tồn tại những ý kiến trái chiều về việc liệu GPU của Nvidia có tốc độ xử lý bộ nhớ vượt trội hơn so với AMD hay không, và ngược lại.

Vì thế, việc tiến hành nghiên cứu so sánh tốc độ xử lý bộ nhớ trung bình giữa các

GPU của Nvidia và AMD trở nên đặc biệt có ý nghĩa. Bằng cách kiểm định với mức ý nghĩa 5%, nghiên cứu này không chỉ nhằm làm rõ sự khác biệt có thể tồn tại giữa hai nhà sản xuất mà còn góp phần cung cấp những thông tin giá trị cho người dùng khi chọn lựa sản phẩm.

Câu hỏi nghiên cứu đặt ra là: Liệu tốc độ xử lý bộ nhớ (Memory Speed) trung bình của các GPU Nvidia có vượt trội so với các sản phẩm từ AMD hay không, khi phân tích dựa trên dữ liệu thực nghiệm và kiểm định thống kê với mức ý nghĩa 5%.

5.3.2 Tiến hành phân tích

Trước khi tiến hành so sánh, chúng ta sẽ sử dụng hàm subset() để phân loại dữ liệu thành hai nhóm (GPU của Nvidia và GPU của AMD) và kiểm tra tính chuẩn của phân phối dữ liệu bằng các lệnh `qqnorm()`, `qqline()`, và `shapiro.test()`. Kết quả từ những phép thử này sẽ cho phép chúng ta xác định phương pháp kiểm định phù hợp, đảm bảo tính khách quan và độ chính xác của kết luận cuối cùng.
Ta chia dữ liệu biến Memory Speed thành 2 nhóm như sau:

- Nhóm 1 (X_1): Các GPU của nhà sản xuất Nvidia.
- Nhóm 2 (X_2): Các GPU của nhà sản xuất AMD.

```
Group_1_Data <- subset(New_GPU_Data, Manufacturer == "Nvidia")
Group_2_Data <- subset(New_GPU_Data, Manufacturer == "AMD")
```

Thiết lập giả thuyết kiểm định

- Giả thuyết H_0: Tốc độ xử lý bộ nhớ (Memory_Speed) trung bình của GPU do "Nvidia" sản xuất không cao hơn tốc độ xử lý bộ nhớ trung bình GPU do "AMD" sản xuất ($\mu_{\text{Nvidia}} \leq \mu_{\text{AMD}}$).
- Giả thuyết H_1: : Tốc độ xử lý bộ nhớ (Memory_Speed) trung bình của GPU do "Nvidia" sản xuất cao hơn tốc độ xử lý bộ nhớ trung bình của GPU do "AMD" sản xuất ($\mu_{\text{Nvidia}} > \mu_{\text{AMD}}$).

Kiểm tra kiện phân phối chuẩn của biến Memory_Speed thuộc Nhóm 1 (Nvidia)

Để xác định liệu biến Memory_Speed thuộc Nhóm 1(Nvidia) có phân phối chuẩn hay không, ta sử dụng phương pháp kiểm định giả thuyết.

- Giả thuyết H_0: Biến Memory_Speed có phân phối chuẩn.
- Giả thuyết H_1: Biến Memory_Speed không có phân phối chuẩn.

Sử dụng hàm `shapiro.test` để tính giá trị p-value. Dựa trên giá trị này, ta có thể đưa ra quyết định về giả thuyết H_0 cần tìm.

- Nếu p-value $\geq 5\%$: Chưa có đủ bằng chứng để bác bỏ giả thuyết H_0. Điều này có nghĩa là dữ liệu có thể được coi là có phân phối chuẩn.
- Nếu p_value $< 5\%$: Bác bỏ giả thuyết H_0, cho thấy dữ liệu không tuân theo phân phối chuẩn.

```
shapiro.test(Group_1_Data$Memory_Speed)
```

```
#  Shapiro-Wilk normality test
# data:  Group_1_Data$Memory_Speed
# W = 0.95611, p-value < 2.2e-16
```

Kết quả của hàm kiểm định `shapiro.test()` cho p-value rất nhỏ (p-value $< 2.2 \times 10^{-16}$), nghĩa là thấp hơn đáng kể so với ngưỡng ý nghĩa 5%. P-value nhỏ như vậy cho phép chúng ta bác bỏ giả thuyết H_0, tức là giả thuyết cho rằng dữ liệu tuân theo phân phối chuẩn. Từ đó ta đưa ra kết luận dữ liệu `Memory_Speed` trong nhóm "Nvidia" không tuân theo phân phối chuẩn.

Kiểm tra kiện phân phối chuẩn của biến Memory_Speed thuộc Nhóm 2 (AMD)

Để xác định liệu biến Memory_Speed thuộc Nhóm 2 (AMD) có phân phối chuẩn hay không, ta sử dụng phương pháp kiểm định giả thuyết.

- Giả thuyết H_0: Biến Memory_Speed có phân phối chuẩn.

- Giả thuyết H_1: Biến Memory_Speed không có phân phối chuẩn.

Sử dụng hàm `shapiro.test` để tính giá trị p-value. Dựa trên giá trị này, ta có thể đưa ra quyết định về giả thuyết H_0 cần tìm.

- Nếu p-value $\geq 5\%$: Chưa có đủ bằng chứng để bác bỏ giả thuyết H_0. Điều này có nghĩa là dữ liệu có thể được coi là có phân phối chuẩn.
- Nếu p_value $< 5\%$: Bác bỏ giả thuyết H_0, cho thấy dữ liệu không tuân theo phân phối chuẩn.

```
shapiro.test(Group_2_Data$Memory_Speed)
```

```
#  Shapiro-Wilk normality test
# data:  Group_2_Data$Memory_Speed
# W = 0.98884, p-value = 9.925e-08
```

Kết quả của hàm kiểm định shpiro.test() với p-value $= 9.925 * 10^{-8}$ nhỏ hơn rất nhiều so với ngưỡng ý nghĩa 5% (0.05). Điều này dẫn đến việc bác bỏ giả thuyết H0, tức là giả thuyết cho rằng Memory_Speed của nhóm "AMD" không tuân theo phân phối chuẩn. Qua những kiểm định phía trên ta rút ra được kết luận rằng cả hai nhóm dữ liệu Memory_Speed của Nvidia và AMD đều cho thấy không tuân theo phân phối chuẩn, do dó bài toán thuộc loại bài toán kiểm định với 2 mẫu độc lập; X1, X2 có phân phối bất kỳ và cỡ mẫu lớn.

Tính toán các đặc trưng mẫu

Từ dữ liệu, ta tính toán các đặc trưng mẫu cho mỗi nhóm như sau:

```
# Tinh cac dac trung mau cua bien Memory\_Speed thuoc nhom 1
n1 <- length(Group_1_Data$Memory_Speed)
X1_tb <- mean(Group_1_Data$Memory_Speed)
s1 <- sd(Group_1_Data$Memory_Speed)

# Tinh các dac trung mau cua bien Memory\_Speed thuoc nhom 2
n2 <- length(Group_2_Data$Memory_Speed)
X2_tb <- mean(Group_2_Data$Memory_Speed)
```

```
s2 <- sd(Group_2_Data$Memory_Speed)


# In khung du lieu
data.frame(n1, X1_tb, s1, n2, X2_tb, s2)
```

Ta thu được kết quả sau:

```
#     n1    X1_tb       s1   n2    X2_tb      s2
# 1 1641 1267.399 444.4817 1161 1131.214 355.524
```

Tính giá trị thống kê kiểm định thực nghiệm (Z_o)

```
# Tim gia tri thong ke kiem đinh Zo thuc nghiem
z0 <- (X1_tb - X2_tb)/sqrt(s1^2/n1 + s2^2/n2)
```

Ta thu được kết quả sau:

```
# [1] 8.994157
```

Kiểm định một phía ($X_1 > X_2$): Đối với kiểm định một phía và mức ý nghĩa $\alpha = 0.05$, giá trị tới hạn là

```
# Tim gia tri toi han Z_a
qnorm(p= .05, lower.tail = FALSE ) #Z(a) voi muc y nghia 5%
```

Ta thu được kết quả sau:

```
# [1] 1.644854
```

Từ đó, ta suy ra miền bác bỏ là $RR = (1.644854; +\infty)$ .Vì $Z_o = 8.994157$ nằm trong miền bác bỏ, ta đủ cơ sở để bác bỏ giả thuyết H_0. Do đó, ta có thể chấp nhận giả thiết H_1. Kết luận rằng, tốc độ xử lý bộ nhớ trung bình của Nvidia cao hơn AMD.

5.3.3 Phân tích mở rông - xét ngược lại điều kiện ban đầu

Giả sử tổng thể X_1 và X_2 là hai mẫu độc lập và có phân phối chuẩn. Khi đó, chúng ta sẽ thực hiện kiểm định phương sai hai mẫu để so sánh sự khác biệt về phương sai giữa hai nhóm.

Đặt giả thuyết:

- Giả thuyết H_0: Phương sai của nhóm 1 bằng phương sai của nhóm 2, tức là $\sigma_1^2 = \sigma_2^2$.
- Giả thuyết H_1: Phương sai của nhóm 1 lớn hơn phương sai của nhóm 2, tức là $\sigma_1^2 > \sigma_2^2$.

Lưu ý: Dấu bất đẳng thức trong giả thuyết thay đổi tùy thuộc vào các giá trị mẫu. Trong trường hợp này, vì $S_1 > S_2$ nên chúng ta sẽ kiểm định xem phương sai của nhóm 1 có lớn hơn phương sai của nhóm 2 hay không.

Bước 1: Thực hiện kiểm định phương sai hai mẫu

Ta sử dụng kiểm định F để so sánh phương sai của hai nhóm, với giả thuyết một phía (alternative = "greater")

```
# Kiem dinh phuong sai hai mau
var.test(Group_1_Data$Memory_Speed, Group_2_Data$Memory_Speed,
alternative = "greater")
```

Ta thu được kết quả sau:

```
#  F test to compare two variances
#
# data:  Group_1_Data$Memory_Speed and Group_2_Data$Memory_Speed
# F = 1.563, num df = 1640, denom df = 1160, p-value = 2.22e-16
# alternative hypothesis: true ratio of variances is greater than 1
# 95 percent confidence interval:
#  1.428981      Inf
# sample estimates:
# ratio of variances
#           1.563039
```

Kết luận: Ta nhận thấy, $p - value = 2.22 * 10^{-16}$ nhỏ hơn mức ý nghĩa 0.05, nên có thể ta bác bỏ giả thuyết H_0 và chấp nhận giả thuyết H1. Do đó, có đủ bằng chứng để kết luận rằng phương sai của nhóm 1 lớn hơn phương sai của nhóm 2, tức là $\sigma_1^2 > \sigma_2^2$. Điều này cho thấy có sự khác biệt về độ biến thiên giữa hai nhóm ($\sigma_1^2 \neq \sigma_1^2$).

Bước 2: Thực hiện phép kiểm định

Ở bước này, ta sử dụng hàm `t.test()` để thực hiện kiểm định. Chúng ta có thể dùng hàm này để so sánh tốc độ xử lý bộ nhớ giữa hai nhóm dữ liệu. Lệnh kiểm định như sau:

```
# Thuc hien kiem định 2 mau bang ham t.test()
t.test(Group_1_Data$Memory_Speed,Group_2_Data$Memory_Speed,
alternative = "greater")
```

Ta thu được kết quả như sau:

```
#  Welch Two Sample t-test
#
# data:  Group_1_Data$Memory_Speed and Group_2_Data$Memory_Speed
# t = 8.9942, df = 2758.3, p-value < 2.2e-16
# alternative hypothesis: true difference in means is greater than 0
# 95 percent confidence interval:
#  111.2703      Inf
# sample estimates:
# mean of x mean of y
#  1267.399  1131.214
```

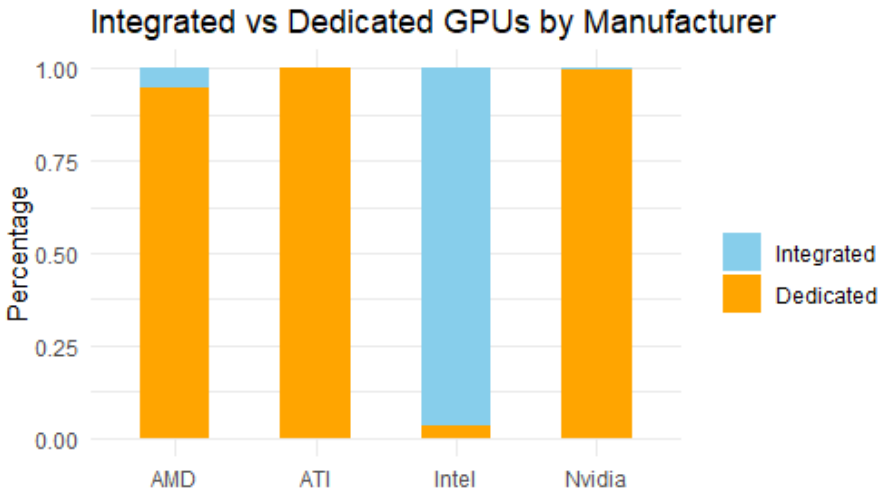
Vì $p - value < 2.22 * 10^{-16}$ nhỏ hơn mức ý nghĩa 0.05 rất nhiều.Với kết quả này, ta có đủ cơ sở để bác bỏ giả thuyết H_0 và chấp nhận giả thuyết H_1, tức là tốc độ xử lý bộ nhớ trung bình (Memory_Speed) của Nvidia cao hơn đáng kể so với AMD.

5.4 Phân tích phương sai Anova

5.4.1 Giới thiệu bài toán

Khi so sánh các GPU, thông số Memory Bandwidth (băng thông bộ nhớ) là một yếu tố quan trọng vì nó ảnh hưởng trực tiếp đến hiệu năng của chúng. Memory Bandwidth được xác định bởi lượng dữ liệu tối đa có thể truyền giữa bộ xử lý đồ họa (GPU) và bộ nhớ đồ họa (VRAM) trong một giây. Thông số này quan trọng vì nó đảm bảo GPU có thể xử lý dữ liệu ở tốc độ cao và số lượng lớn, đặc biệt với các ứng dụng hay tác vụ yêu cầu đồ họa cao và tốc độ xử lý nhanh.

Vấn đề đặt ra là thông số Memory Bandwidth liệu có khác nhau giữa các nhà sản xuất GPU hay không. Nếu có, thì sản phẩm của nhà sản xuất nào có băng thông bộ nhớ cao hơn. Lưu ý rằng việc so sánh thông số này của GPU chỉ nhằm đưa ra quan sát tổng quan, không nên dùng để đánh giá hiệu năng tổng thể của các sản phẩm giữa các nhà sản xuất. Bởi vì GPU của mỗi nhà sản xuất có mục đích sử dụng, thiết kế khác nhau, lấy ví dụ là GPU tích hợp trên CPU (Integrated) so với GPU rời (Dedicated). GPU tích hợp tiết kiệm năng lượng hơn, giá thành rẻ hơn được dùng để xử lý các tác vụ nhẹ nhàng so với GPU rời vốn dĩ được chuyên dùng để xử lý đồ họa.



Hình 5.10: Tỉ lệ GPU rời so với GPU tích hợp của các nhà sản xuất

Như biểu đồ trên đã thể hiện, Intel là nhà sản xuất chip, do đó GPU của hãng hầu như là dạng được tích hợp vào CPU. Trong khi đó Nvidia nổi tiếng là nhà sản xuất card đồ họa rời có hiệu năng tốt trên thị trường. Nhận xét là tương tự với AMD và ATI.

5.4.2 Thực hiện phân tích

Trước khi tiến hành phân tích, cần phải tách dữ liệu cho từng nhà sản xuất từ bộ dữ liệu chung.

```
AMDdata <- subset(New_GPU_Data, Manufacturer=="AMD")
Nvidiadata <- subset(New_GPU_Data, Manufacturer=="Nvidia")
Inteldata <- subset(New_GPU_Data, Manufacturer=="Intel")
ATIdata <- subset(New_GPU_Data, Manufacturer=="ATI")
```

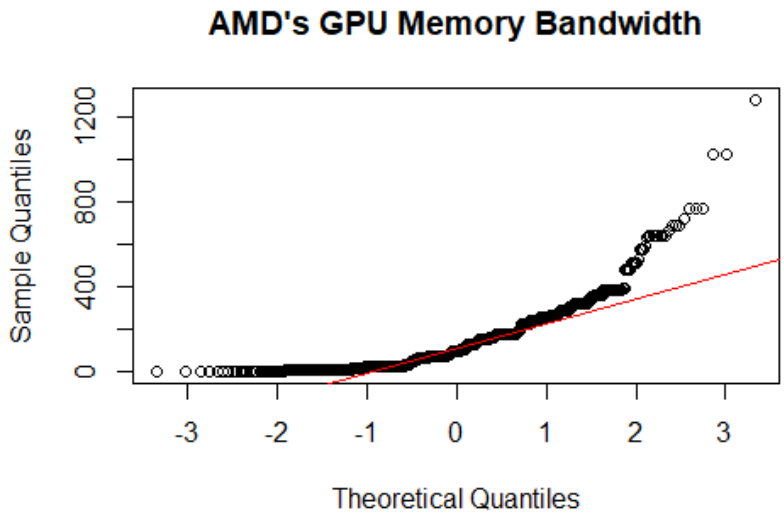
Kiểm tra các giả định

- Các tổng thể có phân phối chuẩn.
- Phương sai của các tổng thể là bằng nhau.
- Các quan sát được lấy mẫu độc lập (điều kiện này được giả định là đúng).

Thứ nhất, kiểm tra phân phối chuẩn. Đối với dữ liệu của nhà sản xuất AMD, ta sử dụng lệnh sau để vẽ đồ thị

```
# Ve do thi
qqnorm(AMDdata$Memory_Bandwidth, main = "AMD's GPU Memory Bandwidth")
# Ve duong phan phoi chuan
qqline(AMDdata$Memory_Bandwidth, col = "red")
```

Ta thu được kết quả sau:



Hình 5.11: Đồ thị kiểm tra phân phối chuẩn của dữ liệu nhà AMD

Qua quan sát, có thể dễ dàng nhận định rằng các quan trắc nằm lệch khỏi đường phân phối chuẩn. Do đó, dữ liệu băng thông bộ nhớ của nhà AMD không tuân theo phân phối chuẩn.

Mặt khác, để kiểm tra một bộ dữ liệu có phân phối chuẩn hay không ta có thể dùng tiêu chuẩn kiểm định Shapiro-Wilk. Với các giả thuyết thống kê là:

- H_0: Dữ liệu có phân phối chuẩn
- H_1: Dữ liệu không tuân theo phân phối chuẩn

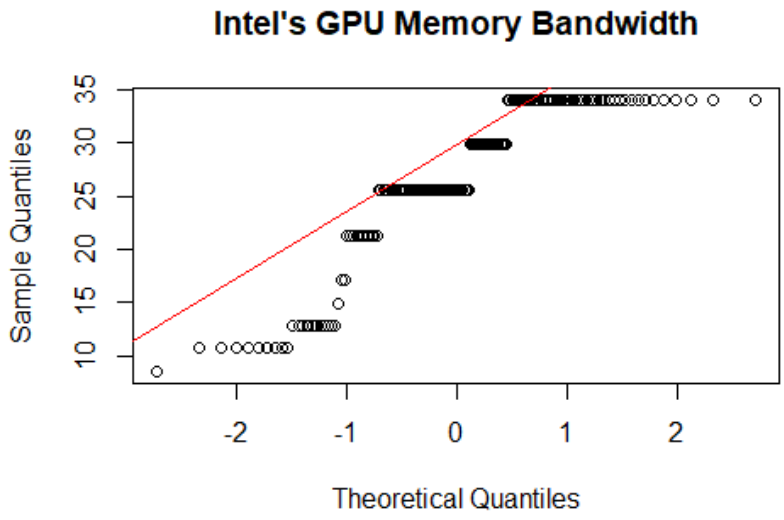
```
shapiro.test(AMDdata$Memory_Bandwidth)
```

Ta thu được kết quả sau:

```
# Shapiro-Wilk normality test
#
# data:  AMD_data$Memory_Bandwidth
# W = 0.80665, p-value < 2.2e-16
```

Thực hiện tương tự với 3 nhà sản xuất còn lại ta thu được các kết quả như sau:

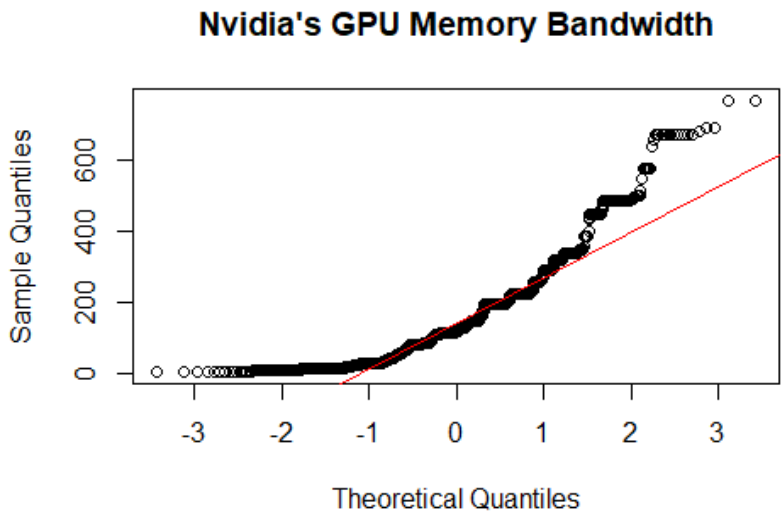
Đối với Intel:



Hình 5.12: Đồ thị kiểm tra phân phối chuẩn của dữ liệu nhà Intel

```
# Shapiro-Wilk normality test
#
# data:  Inteldata$Memory_Bandwidth
# W = 0.84555, p-value = 2.364e-11
```

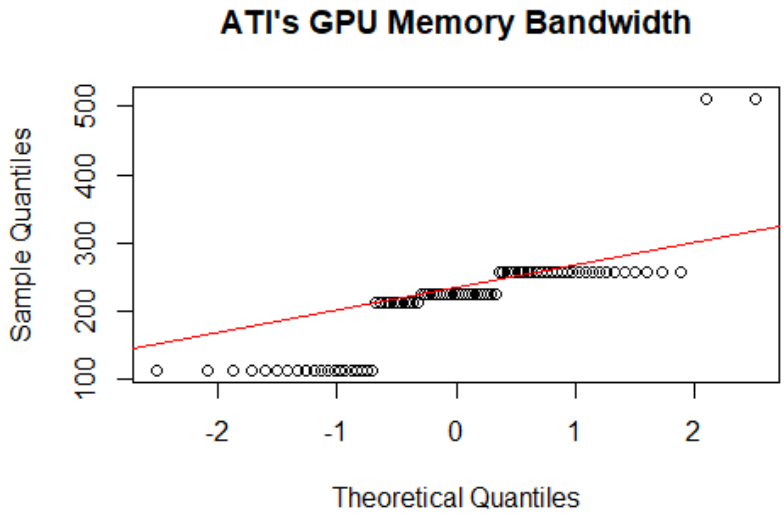
Đối với Nvidia:



Hình 5.13: Đồ thị kiểm tra phân phối chuẩn của dữ liệu nhà Nvidia

```
# Shapiro-Wilk normality test
#
# data:  Nvidiadata$Memory_Bandwidth
# W = 0.87221, p-value < 2.2e-16
```

Đối với ATI:



Hình 5.14: Đồ thị kiểm tra phân phối chuẩn của dữ liệu nhà ATI

```
# Shapiro-Wilk normality test
#
```

```
# data:  ATIdata$Memory_Bandwidth
# W = 0.7408, p-value = 1.003e-10
```

Kết luận: Từ những biểu đồ phía trên, ta có thể kết luận rằng dữ liệu của cả 3 nhà sản xuất còn lại đều không tuân theo phân phối chuẩn.

Thứ hai, kiểm tra phương sai các tổng thể .Để kiểm tra phương sai của các tổng thể nhóm sử dụng phương pháp kiểm định Levene. Cặp giả thuyết kiểm định gồm:

- H_0: Phương sai của các tổng thể bằng nhau.
- H_1: Tồn tại một cặp có phương sai khác nhau.

```
leveneTest(Memory_Bandwidth ~ as.factor(Manufacturer), New_GPU_Data)
```

Ta thu được kết quả sau:

```
##  Levene's Test for Homogeneity of Variance (center = median)
##          Df F value    Pr(>F)
##  group    3  48.537 < 2.2e-16 ***
##        3032
```

Kết quả trả về giá trị $p - value = 2.2 \times 10^{-16} < 0.05$ (mức ý nghĩa 5%), do đó bác bỏ H_0, chấp nhận H_1. Vậy tồn tại một cặp tổng thể có phương sai không bằng nhau.

Mô hình Anova

Mô hình Anova so sánh trung bình thông số Memory Bandwidth giữa các nhà sản xuất GPU. Mô hình này chỉ xem xét sự bằng nhau giữa trung bình các tổng thể do đó cặp giả thuyết kiểm định là:

- H_0: trung bình thông số Memory Bandwidth giữa các nhà sản xuất bằng nhau.
- H_1: có ít nhất 2 nhà sản xuất có trung bình thông số này chênh lệch nhau.

```
      # Gan cho anovaStat
anovaStat<-aov(Memory_Bandwidth ~ Manufacturer, data = New_GPU_Data)
summary(anovaStat) # In ket qua ham tinh Anova
```

Ta thu được kết quả sau:

```
#                Df    Sum Sq Mean Sq F value Pr(>F)
# Manufacturer    3  2770350  923450   52.95 <2e-16 ***
# Residuals    3032 52878166   17440
```

Kết quả trả về giá trị $p-value = 2.2 \times 10^{-16} < 0.05$ (mức ý nghĩa 5%), do đó đủ điều kiện bác bỏ H_0, chấp nhận H_1. Vậy tồn tại 2 nhà sản xuất có trung bình thông số Memory Bandwidth khác nhau.

Chương 6

THẢO LUẬN MỞ RỘNG

6.1 Lý thuyết mở rộng về kiểm định TukeyHSD

6.1.1 Mục tiêu bài toán

Khi thực hiện phân tích ANOVA, nếu bác bỏ giả thuyết H_0 (không có sự khác biệt giữa trung bình các nhóm) và chấp nhận H_1, điều này cho thấy rằng có ít nhất một cặp trung bình tổng thể khác biệt. Tuy nhiên, ANOVA không chỉ ra cụ thể những nhóm nào khác nhau hoặc mức độ khác biệt ra sao.

Để giải quyết vấn đề này, cần tiến hành các kiểm định hậu nghiệm (*post-hoc tests*) để so sánh chi tiết từng cặp trung bình. Phương pháp phổ biến nhất là kiểm định TukeyHSD (*Tukey's Honest Significant Difference*).

6.1.2 Phương pháp TukeyHSD

Phương pháp TukeyHSD được thiết kế để so sánh trung bình của tất cả các cặp nhóm, với mục tiêu:

- Xác định khoảng tin cậy (*Confidence Interval*, CI) cho sự chênh lệch giữa hai trung bình tổng thể ($\mu_1 - \mu_2$).
- Đánh giá ý nghĩa thống kê của sự khác biệt giữa các cặp nhóm.

Nguyên tắc hoạt động

- **Khoảng tin cậy (CI) của từng cặp nhóm:**
  - Nếu CI chứa toàn các giá trị dương, điều này cho thấy $\mu_1 > \mu_2$.
  - Nếu CI chứa toàn các giá trị âm, điều này cho thấy $\mu_1 < \mu_2$.
  - Nếu CI chứa số 0, không thể kết luận có sự khác biệt giữa μ_1 và μ_2.
- **Gia đình kiểm định (*Family-Wise Confidence Level*):**
  - Phương pháp TukeyHSD điều chỉnh mức ý nghĩa để kiểm soát tỷ lệ sai lầm loại I (*false positives*) khi thực hiện nhiều phép kiểm định cùng lúc.
  - Ví dụ: Nếu thực hiện nhiều kiểm định t độc lập, xác suất mắc lỗi loại I tăng lên, nhưng TukeyHSD duy trì mức ý nghĩa cố định cho toàn bộ các so sánh.

6.1.3 Cách thực hiện kiểm định TukeyHSD

Trong R, kiểm định TukeyHSD được thực hiện bằng cách áp dụng hàm `TukeyHSD` lên kết quả của phân tích ANOVA. Sau đó, có thể trực quan hóa khoảng tin cậy của từng cặp bằng đồ thị để dễ dàng nhận biết sự khác biệt.

6.2 Thực hiện kiểm định TukeyHSD

Sau khi thực hiện phân tích ANOVA và bác bỏ giả thuyết H_0, ta cần kiểm tra cụ thể cặp nhóm nào có sự khác biệt đáng kể. Để thực hiện, kiểm định hậu nghiệm TukeyHSD được sử dụng. Phương pháp này so sánh từng cặp trung bình tổng thể và xác định khoảng tin cậy (CI) cho sự khác biệt giữa chúng.

Kiểm định TukeyHSD không chỉ cung cấp mức ý nghĩa thống kê (p-value) cho từng cặp, mà còn cho phép đánh giá khoảng tin cậy của chênh lệch giữa trung bình tổng thể $(\mu_1 - \mu_2)$:

- Nếu khoảng tin cậy chứa toàn số dương, kết luận rằng $\mu_1 > \mu_2$.
- Nếu khoảng tin cậy chứa toàn số âm, kết luận rằng $\mu_1 < \mu_2$.

- Nếu khoảng tin cậy chứa số 0, không có bằng chứng để khẳng định sự khác biệt giữa μ_1 và μ_2.

Kết quả kiểm định TukeyHSD trên dữ liệu hiệu suất GPU được thực hiện bằng R như sau:

```
TukeyHSD(anovaStat)
```

Kết quả đầu ra:

```
##  Tukey multiple comparisons of means
##  95% family-wise confidence level
##
##  Fit: aov(formula = Memory_Bandwidth~Manufacturer, data = New_GPU_Data)
##
##  $Manufacturer
##                   diff          lwr         upr     p adj
##  ATI-AMD         74.06996   35.282043  112.85787 0.0000058
##  Intel-AMD     -112.38396 -141.664449  -83.10346 0.0000000
##  Nvidia-AMD      17.53716    4.519064   30.55526 0.0030427
##  Intel-ATI     -186.45392 -232.965712 -139.94212 0.0000000
##  Nvidia-ATI     -56.53280  -94.944655  -18.12094 0.0009079
##  Nvidia-Intel   129.92112  101.140644  158.70160 0.0000000
```

Từ kết quả trên:

- **diff**: Sự khác biệt giữa trung bình của từng cặp nhóm.
- **lwr**: Giá trị dưới của khoảng tin cậy.
- **upr**: Giá trị trên của khoảng tin cậy.
- **p adj**: Giá trị p-value đã điều chỉnh để kiểm soát lỗi loại I.

Ví dụ:

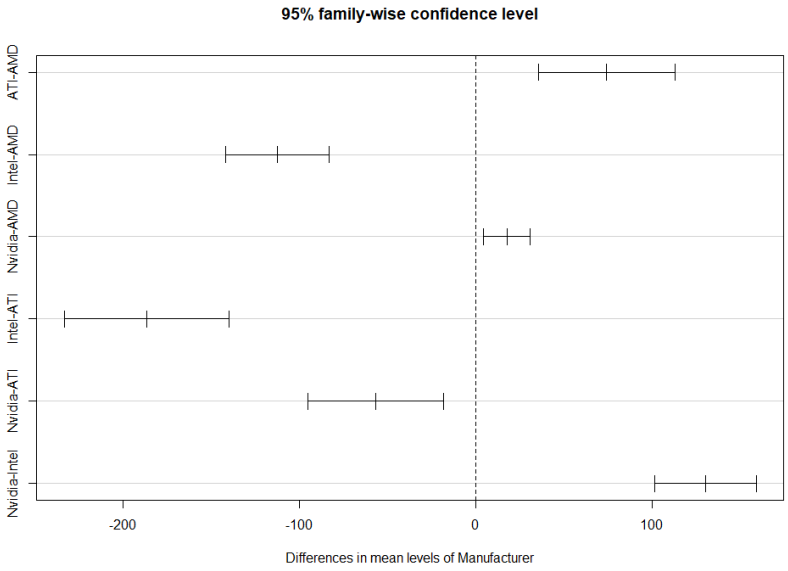
- **Cặp Intel-AMD:** Khoảng tin cậy $[-141.66, -83.10]$ chứa toàn số âm, chứng tỏ $\mu_{\text{Intel}} < \mu_{\text{AMD}}$.

- **Cặp Nvidia-Intel:** Khoảng tin cậy $[101.14, 158.70]$ chứa toàn số dương, chứng tỏ $\mu_{\text{Nvidia}} > \mu_{\text{Intel}}$.

Để trực quan hơn, có thể vẽ đồ thị khoảng tin cậy bằng lệnh sau:

```
plot(TukeyHSD((anovaStat)))
```

Kết quả đồ thị:



Hình 6.1: Khoảng tin cậy của từng cặp nhóm

Từ biểu đồ, có thể nhận thấy rằng:

- GPU rời của Nvidia có thông số Memory Bandwidth cao hơn AMD, chứng tỏ hiệu năng tổng thể tốt hơn. Điều này được xác nhận bởi $|\mu_{\text{Nvidia}} - \mu_{\text{AMD}}| > 0$.
- Hiệu suất GPU rời (Nvidia) cao hơn GPU tích hợp (Intel) với giá trị chênh lệch trung bình tương đối lớn ($|\mu_{\text{Nvidia}} - \mu_{\text{Intel}}| > 100$).

Kiểm định TukeyHSD cung cấp cách tiếp cận toàn diện và hiệu quả để đánh giá sự khác biệt giữa các cặp trung bình, giúp đưa ra kết luận cụ thể hơn sau phân tích ANOVA. Điều này rất hữu ích trong các bài toán thực tiễn như so sánh hiệu suất sản phẩm hoặc phân tích nhóm đối tượng.

Chương 7

NGUỒN CODE VÀ NGUỒN DỮ LIỆU

7.1 Nguồn code R

Tham khảo đường link dưới đây để tới file code của nhóm:

https://drive.google.com/file/d/1aYMOQfr1l0Ml64CDKGa1mP2bnl5UXUiQ/view?usp=drive_link

7.2 Nguồn dữ liệu

Tham khảo đường link dưới đây để tới file dữ liệu của nhóm:

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

Tài liệu tham khảo

[1] TechPowerUp. *GPU Specs Database*. 2024. Đường dẫn: `https://www.techpowerup.com/gpu-specs/`.

[2] The R Project for Statistical Computing. *Documentation*. 2024. Đường dẫn: `https://www.r-project.org/other-docs.html`.

[3] Seaborn. *Visualizing distributions of data*. 2024. Đường dẫn: `https://seaborn.pydata.org/tutorial/distributions.html`.

[4] Ilias Sekkaf. *Computer Parts (CPUs and GPUs)*. 2024. Đường dẫn: `https://www.kaggle.com/datasets/iliassekkaf/computerparts?resource=downlo`.

[5] Nguyễn Kiều Dung. *Bài giảng Xác suất và Thống kê*. Trường Đại học Bách khoa ĐHQG - HCM. 2024.