ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Bài Tập Lớn

Đề tài: "Dự đoán sự phát triển của các thông số kỹ thuật và hiệu suất của máy tính trong tương lai"

Môn học: Xác Suất Thống Kê (MT2013)

Nhóm: 17

Lớp: L11

Giáo viên hướng dẫn:

TS. Nguyễn Bá Thi

Ngày 8 tháng 5 năm 2024

Mục lục

Giới thiệu thành viên và phân công

Bảng dưới đây biểu diễn thông tin về thành viên, phân công và đánh giá trong bài tập lớn lần này của chúng em.

Họ và tên	Mã số sinh viên	Công việc	Đánh giá phần trăm
Lâm Vũ	2213994	Tổng hợp thông tin, viết báo cáo	100%
Phạm Gia Nguyên	2212319	Thống kê suy diễn	100%
Vương Nhật Minh	2212094	Thống kê tả	100%
Lê Nguyễn Gia Hưng	2211363	Tiền xử lý số liệu	100%
Võ Phương Minh Nhật	2212413	Thống kê tả	100%

1 Tổng quan dữ liệu

Bài tập lớn này trình bày phân tích về thông số kỹ thuật ảnh hưởng đến hiệu suất **bộ xử lý đồ họa - Graphics Processing Units (GPUs)** như thế nào theo thời gian, tập trung vào hiệu suất tính toán chung. Do có giới hạn về thời gian nghiên cứu và báo báo cáo, nhóm chúng em đã quyết định chọn ra các biến chính và quan trọng thực sự ảnh hưởng đến sự ảnh hưởng đến hiệu năng của GPUs và phân tích chúng. Nhóm chúng em tìm hiểu được các yếu tố ảnh hưởng đến GPUs qua các bài báo cáo và website để trong phần Tài liệu tham khảo. Vậy các biến chính trong bộ dữ liệu này bao gồm:

Core Speed là tốc độ đồng hồ của các lõi xử lý của GPU. Một corespeed cao thường dẫn đến hiệu suất tính toán nhanh hơn, cho phép GPU thực hiện nhiều hoạt động mỗi giây hơn và xử lý các nhiệm vụ đồ họa phức tạp một cách hiệu quả hơn.

Memory Bộ nhớ GPU, còn được gọi là VRAM (Bộ nhớ Truy cập Ngẫu nhiên Video), lưu trữ dữ liệu và chỉ thị cần thiết để hiển thị đồ họa. Dung lượng bộ nhớ cao cho phép GPU xử lý các texture và mô hình lớn hơn và chi tiết hơn, dẫn đến hiệu suất mượt mà hơn, đặc biệt là ở độ phân giải cao.

Memory Bus Độ rộng bus bộ nhớ xác định tốc độ mà dữ liệu có thể được chuyển đổi giữa GPU và bộ nhớ của nó. Bus bộ nhớ rộng hơn cho phép lưu lượng dữ liệu lớn hơn, tăng cường hiệu suất tổng thể, đặc biệt là trong các tác vụ yêu cầu nhiều bộ nhớ như chơi game và làm phim.

Memory Speed Tốc độ bộ nhớ, đo bằng megahertz (MHz) hoặc gigahertz (GHz), cho biết tốc độ mà bộ nhớ của GPU có thể truy cập và trích xuất dữ liệu. Bộ nhớ nhanh hơn dẫn đến thời gian tải ngắn hơn, tốc độ khung hình mượt mà hơn và đáp ứng tốt hơn, đặc biệt là trong các ứng dụng yêu cầu đồ họa.

Memory Type Loại bộ nhớ được sử dụng trong một GPU, như GDDR6, GDDR5 hoặc HBM (Bộ nhớ Băng thông Cao), ảnh hưởng đến các đặc tính hiệu suất của nó. Các loại bộ nhớ mới thường cung cấp băng thông cao hơn và tiêu thụ điện năng thấp hơn, dẫn đến hiệu suất tổng thể và hiệu quả cải thiện.

Process Công nghệ quy trình đề cập đến quy trình sản xuất được sử dụng để sản xuất vi chip silic của GPU. Các quy trình nhỏ hơn, như 7nm hoặc 5nm, cho phép sản xuất các GPU hiệu quả và mạnh mẽ hơn với mật độ transistor cao hơn, tiêu thụ điện năng thấp hơn và tính năng nhiệt tốt hơn.

Pixel Rate biểu thị số lượng pixel mà GPU có thể hiển thị mỗi giây. Một tốc độ pixel cao dẫn đến đồ họa mượt mà và chi tiết hơn, đặc biệt là trong các trò chơi nhanh hoặc ứng dụng có hiệu ứng hình ảnh phức tạp.

Texture Rate chỉ ra tốc độ mà GPU có thể xử lý các hoạt động ánh xạ texture. GPU có texture_rate cao hơn có thể áp dụng các texture lên bề mặt nhanh chóng và hiệu quả hơn, dẫn đến chất lượng hình ảnh và hiện thực cải thiện.

Resolution_wxh biểu thị độ phân giải hiển thị tối đa được hỗ trợ bởi GPU, được chỉ định bằng chiều rộng (w) và chiều cao (h) trong pixel. Độ phân giải cao đòi hỏi nhiều sức mạnh tính toán hơn để hiển thị đồ họa, do đó GPU hỗ trợ độ phân giải cao có thể cung cấp độ rõ nét và độ chi tiết tốt hơn.

Manufacture Nhà sản xuất của một GPU ảnh hưởng đến các yếu tố như chất lượng thiết kế, độ tin cậy, hỗ trợ driver và uy tín tổng thể. Các nhà sản xuất khác nhau có thể cung cấp GPU với các cấp độ hiệu suất, tính năng và giá cả khác nhau, ảnh hưởng đến sự lựa chọn của người dùng dựa trên yêu cầu cụ thể của họ.

Trong bài tập lớn này, nhóm chúng em chọn R/R studio để làm công cụ và môi trường để phân tích dữ liệu.

2 Kiến thức nền

2.1 Phân tích và khám phá dữ liệu

Phân tích và khám phá dữ liệu (EDA) là phương pháp phân tích dữ liệu bằng cách sử dụng các biểu đồ và thống kê để khám phá mối quan hệ giữa các biến, nhằm phát hiện mô hình, xu hướng và giá trị ngoại lệ trong dữ liệu. EDA đóng vai trò quan trọng trong việc xác định các đặc điểm quan trọng cho các mô hình dự đoán. Bằng đồ thị biểu thị từ dữ liệu gốc, chúng ta có thể thể hiện được:

- **Biểu đồ tần suất (Histograms)** là một dạng biểu đồ cột cho thấy sự thay đổi, biến động của biến số
- **Biểu đồ phân tán (Scatter plots)** là một loại biểu đồ sử dụng tọa độ Descartes để hiển thị giá trị và mối quan hệ giữa hai biến định lượng cho một tập dữ liệu. Dữ liệu được hiển thị dưới dạng tập hợp các điểm, mỗi điểm có giá trị của một biến xác định vị trí trên trục hoành và giá trị của biến khác xác định vị trí trên trục tung.

2.2 Hồi quy tuyến tính bội

Hồi quy tuyến tính bội là phần mở rộng của hồi quy tuyến tính đơn. Nó được sử dụng khi chúng ta muốn dự đoán giá trị của một biến phản hồi dựa trên giá trị của hai hoặc nhiều biến giải thích. Biến chúng ta muốn dự đoán gọi là biến phản hồi (hoặc biến phụ thuộc). Các biến mà chúng ta sử dụng để dự đoán giá trị của biến phản hồi được gọi là các biến giải thích (hoặc biến độc lập)
Công thức tổng quát của hồi quy đa biến (MLR):

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon$$

Trong đó

Y**:** biến phụ thuộc

x_i**:** biến độc lập thứ i

B_0**:** giá trị của Y khi tất cả x_i đều bằng 0

B_i**:** hệ số hồi quy của mỗi x_i. Chỉ số này cho biết về mức thay đổi của Y gây ra bởi x_i tương ứng.

ϵ**:** sai số ngẫu nhiên của mô hình, chỉ số này càng lớn kiến cho khả năng dự đoán của hồi quy trở nên kém chính xác hơn hoặc sai lệch nhiều hơn so với thực tế.

Trong khi xây dựng hồi quy tuyến tính bội, cần kiểm tra các giả thiết sau:

1. Hàm hồi quy là tuyến tính theo các tham số.
2. Các sai số là sai số cố định ("cố định" ở đây có nghĩa là không có gì sai sót ngẫu nhiên trong đo lường).
3. Các sai số độc lập với nhau.
4. Các sai số tuân theo luật phân phối chuẩn.
5. Các sai số có giá trị trung bình (mean) là 0.
6. Các sai số có phương sai bằng nhau.

2.3 Độ phù hợp của mô hình

Ký hiệu:

$TSS = \sum(y_i - \bar{y})^2$**:** Total Sum of Squares

$ESS = \sum(\hat{y}_i - \bar{y})^2$**:** Explained Sum of Squares

$RSS = \sum(y_i - \hat{y})^2$**:** Explained Sum of Squares

Ta có công thức: $TSS = ESS + RSS$

Ý nghĩa các thành phần:

TSS**:** Tổng bình phương của tất cả các sai lệch giữa các giá trị quan sát y_i và giá trị trung bình $\bar{y}$.

ESS**:** Tổng bình phương của tất cả các giá trị sai lệch giữa các giá trị của biến phụ thuộc $\hat{y}$ nhận được từ hàm hồi quy mẫu và giá trị trung bình $\bar{y}$. Dùng để đo độ chính xác của hàm hồi quy.

RSS**:** Tổng bình phương của tất cả các giá trị sai lệch giữa các giá trị quan sát y_i và các giá trị của biến phụ thuộc $\hat{y}$ nhận được từ hàm hồi quy.

Định nghĩa: Hệ số xác định của hàm hồi quy, ta ký hiệu là R^2, được xác định theo công thức

$$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$$

Do TSS, ESS, RSS đều không âm, nên từ biểu thức trên có thể thấy $0 \leq R^2 \leq 1$. Giá trị R đo sự phù hợp của mô hình (hàm hồi quy) với số liệu mẫu. Ta kỳ vọng rằng nếu mô hình có độ phù hợp cao với số liệu mẫu thì nó cũng phù hợp trong tổng thể

Ý nghĩa: Với mô hình hồi quy có k biến, R^2 có ý nghĩa như sau:

R^2**:** tỉ lệ thay đổi của biến phụ thuộc được giải thích bởi các biến độc lập trong mô hình. Thể hiện mức độ tương quan tuyến tính giữa biến phụ thuộc với các biến độc lập. Cụ thể, với $0 \leq R^2 \leq 1$, ta có hai trường hợp đặc biệt:

- Nếu $R^2 = 1$, nghĩa là 100% sự thay đổi của biến phụ thuộc được giải thích bởi các biến độc lập trong mô hình.
- Nếu $R = 0$, nghĩa là các biến độc lập không giải thích được một chút nào đối với sự thay đổi của biến phụ thuộc.

2.4 Lý thuyết về ANOVA (Phân tích phương sai)

Mục tiêu của phân tích phương sai (Analysis of Variance - ANOVA) là so sánh trung bình của nhiều nhóm (tổng thể) dựa trên các trị trung bình của các mẫu quan sát từ các nhóm này và thông qua kiểm định giả thuyết của kết luận về sự bằng nhau của các trung bình tổng thể này.

Ta có các mô hình phân tích phương sai: phân tích phương sai một yếu tố và hai yếu tố. Cụm từ yếu tố ở đây ám chỉ số lượng yếu tố nguyên nhân ảnh hưởng đến yếu tố kết quả đang nghiên cứu.

Phân tích phương sai một yếu tố (One way ANOVA) là phân tích ảnh hưởng của một yếu tố nguyên nhân (dạng biến định tính) ảnh hưởng đến một yếu tố kết quả (dạng biến định lượng) đang nghiên cứu.

Trường hợp k tổng thể có phân phối chuẩn và phương sai bằng nhau. Giả sử rằng chúng ta muốn so sánh trung bình của k tổng thể dựa trên những mẫu ngẫu nhiên độc lập gồm $n_1, n_2, n_3, ..., n_k$ quan sát từ k tổng thể. Cần ghi nhớ ba giả định sau đây về các nhóm tổng thế được tiến hành phân tích ANOVA.

- Các tổng thể này có phân phối chuẩn
- Các phương sai có tổng thể bằng nhau
- Các quan sát được lấy mẫu là độc lập nhau

Nếu trung bình của các tổng thể được kí hiệu là $\mu_1, \mu_2, \ldots, \mu_k$ thì khi các giả định trên được đáp ứng, mô hình phân tích phương sai một yếu tố ảnh hưởng được mô tả dưới dạng kiểm định giả thuyết như sau:

$$H_0 : \mu_1 = \mu_1 = \cdots = \mu_k$$

Giả thuyết H_0 cho rằng trung bình của k tổng thể đều bằng nhau (về mặt nghiên cứu liên hệ thì giả thuyết này cho rằng yếu tố nguyên nhân không có tác động gì đến vấn đề ta đang nghiên cứu). Và giả thuyết đối là:

$$H_1 : \text{Tồn tại ít nhất một cặp trung bình tổng thể khác nhau.}$$

Các bước thực hiện:

Bước 1: Tính các trung bình mẫu của các nhóm (xem như đại diện của các tổng thể)

Trước hết ta xem cách tính các trung bình mẫu từ những quan sát của k mẫu ngẫu nhiên độc lập (kí hiệu $\bar{x_1}, \bar{x_2}, \ldots, \bar{x_k}$) và trung bình chung của k mẫu quan sát (kí hiệu ($\bar{x_1}$) từ trường hợp tổng

quát như sau:

Tổng thể			
1	2	...	3
x_{11}	x_{21}	...	x_{31}
x_{12}	x_{22}	...	x_{32}
...	...	...	...
x_{1n_1}	x_{2n_2}	...	x_{3n_2}

Bảng 1: Bảng số liệu tổng quát hiện thực phân tích phương sai

Tính trung bình mẫu của từng nhóm $\bar{x_1}, \bar{x_2}, \ldots, \bar{x_k}$ theo công thức

$$\bar{x_i} = \frac{\sum_{j=1}^{n_i} x_{ij}}{n_i} (i = 1, 2, \ldots, k)$$

Tính trung bình chung của k mẫu (trung bình chung của toàn bộ mẫu khảo sát)

$$\bar{x} = \frac{\sum_{i=1}^{k} n_i . \bar{x_i}}{\sum_{i=1}^{k} n_i}$$

Bước 2: Tính các tổng các chênh lệch bình phương (Tổng bình phương)

Tính tổng các chênh lệch bình phương trong nội bộ nhóm SSW và tổng các chênh lệch bình phương giữa các nhóm SSG

+ Tổng các chênh lệch bình phương trong nội bộ nhóm (SSW) được tính bằng cách cộng các chênh lệch bình phương giữa các giá trị quan sát với trung bình mẫu của từng nhóm, rồi sau đó lại tính tổng cộng kết quả tất cả các nhóm lại. SSW phản ánh phần biến thiên của yếu tố kết quả do ảnh hưởng của các yếu tố khác, chứ không phải do yếu tố nguyên nhân đang nghiên cứu (là yếu tố dùng để phân biệt các tổng thể/ nhóm đang so sánh)

+ Tổng các chênh lệch bình phương của từng nhóm dược tính theo công thức:
  Nhóm 1: $SS1 = \sum_{j=1}^{n_i} (x_{1j} - \bar{x_1})^2$
  Nhóm 2: $SS2 = \sum_{j=1}^{n_i} (x_{2j} - \bar{x_2})^2$
  Tương tự như vậy ta tính cho đến nhóm thứ k được SS . Vậy tổng các chênh lệch bình phương trong nội bộ các nhóm được tính như sau:

$$SSW = SS_1 + SS_2 + \cdots + SS_k$$

+ Tổng các chênh lệch bình phương giữa các nhóm (SSG) được tính bằng cách cộng các chênh lệch được lấy bình phương giữa các trung bình mẫu của từng nhóm với trung bình chung của k nhóm (các chênh lệch này đều được nhận thêm với số quan sát tương ứng của từng nhóm). SSG phản ánh phần biến thiên của yếu tố kết quả do ảnh hưởng của yếu tố nguyên nhân đang nghiên cứu.

$$SSG = \sum_{i=1}^{k} n_i(\bar{x}_i - \bar{x})^2$$

+ Tổng các chênh lệch bình phương toàn bộ SST được tính bằng cách cộng tổng các chênh lệch đã lấy bình phương giữa từng giá trị quan sát của toàn bộ mẫu nghiên cứu (x_{ij}) với trung bình chung toàn bộ ($\bar{x}$). SST phản ánh biến thiên của yếu tố kết quả do ảnh hưởng của tất cả các nguyên nhân.

$$SST = \sum_{i=1}^{k}\sum_{j=1}^{k}(x_{ij} - \bar{x})^2$$

Có thể dễ dàng chứng minh là tổng các chênh lệch bình phương toàn bộ bằng tổng cộng tổng các chênh lệch bình phương trong nội bộ các nhóm và tổng các chênh lệch bình phương giữa các nhóm.

$$SST = SSW + SSG$$

Như vậy công thức trên cho thấy, SST là toàn bộ biến thiên của yếu tố kết quả đã được phân tích thành 2 phần: phần biến thiên do yếu tố đang nghiên cứu tạo ra (SSG) và phần biến thiên còn lại do các yếu tố khác không nghiên cứu ở đây tạo ra (SSW). Nếu phần biến thiên do yếu tố nguyên nhân đang xét tạo ra càng "đáng kể" so với phần biến thiên do các yếu tố khác không xét tạo ra, thì chúng ta càng có cơ sở để bác bỏ Ho và kết luận là yếu tố nguyên nhân đang nghiên cứu ảnh hưởng có ý nghĩa đến yếu tố kết quả.

Bước 3: Tính các phương sai (là trung bình của các chênh lệch bình phương). Các phương sai được tính bằng cách lấy các tổng các chênh lệch bình phương chia cho bậc tự do tương úng.
Tính phương sai trong nội bộ nhóm (MSW) bằng cách lấy tổng các chênh lệch bình phương trong nội bộ các nhóm (SSW) chia cho bậc tự do tương ứng là n-k (n là số quan sát, k là số nhóm so sánh). MSW là ước lượng phần biến thiên của yếu tố kết quả do các yếu tố khác gây ra (hay giải thích được)

$$MSW = \frac{SSW}{n-k}$$

Tính phương sai giữa các nhóm (MSG) bằng cách lấy tổng các chênh lệch bình phương giữa các nhóm chia cho bậc tự do tương ứng là k 1. MSG là ước lượng phần biến thiên của yếu tố kết quả do yếu tố nguyên nhân đang nghiên cứu gây ra (hay giải thích được).

$$MSG = \frac{SSG}{k-1}$$

Bước 4: Kiểm định giả thuyết

Giả thuyết về sự bằng nhau của k trung bình tổng thể được quyết định dựa trên tỉ số của hai phương sai: phương sai giữa các nhóm (MSG) và phương sai trong nội bộ nhóm (MSW), Tỉ số này được gọi là tỉ số F vì nó tuân theo qui luật Fisher-Snedecor với bậc tự do là $k-1$ ở tử số và $n-k$ ở mẫu số

$$F = \frac{MSG}{MSW}$$

Ta bác bỏ giả thuyết H_0 cho rằng trị trung bình của k tổng thể bằng nhau khi

$$F > F_{(k-1;n-k);\alpha}$$

$F > F_{(k-1;n-k);\alpha}$ là giá trị giới hạn tra từ bảng tra số 8 với bậc tự do tra theo cột số $k-1$ và hàng $n-k$.

Nguồn biến thiên	Tổng bình phương	Bậc tự do	Phương sai	Tỉ lệ F
Giữa các nhóm	SSG	$k-1$	$MSG = \frac{SSG}{k-1}$	$F = \frac{MSG}{MSW}$
Trong nội bộ nhóm	SSW	$n-k$	$MSW = \frac{SSW}{n-k}$	-
Toàn bộ	SST	$n-1$	-	-

Bảng 2: Bàng kết quả tổng quát ANOVA từ chương trình excel hay SPSS

Kiểm tra các giả định của phân tích phương sai Chúng ta có thể kiểm tra nhanh các giả định này bằng đồ thị. Histogram là phương pháp tốt nhất để kiểm tra giả định về phân phối bình thường của dữ liệu nhưng nó đòi hỏi một số lượng quan sát khá lớn.

Một phương pháp kiểm định tham số chắc chắn hơn cho giả định phương sai bằng nhau là kiểm định Leneve về phương sai của các tổng thể. Kiểm định này xuất phát từ giả thuyết sau.

$H_0 = \sigma_1^2, \sigma_2^2, \ldots, \sigma_k^2$

H_1**:** Có ít nhất 1 cặp phương sai khác nhau

Để quyết định chấp nhận hay bác bỏ H_0 ta tính toán giá trị kiểm định F theo công thức

$$F_{max} = \frac{S_{max}^2}{S_{min}^2}$$

Trong đó:
S_{max}^2 là phương sai lớn nhất trong các nhóm nghiên cứu và S_{min}^2 là phương sai nhỏ nhất trong các nhóm nghiên cứu.
Giá trị F tính được được đem so sánh với giá trị F(k;df);a tra được từ bảng phân phối Hartley F_{max}. Trong đó k là số nhóm so sánh, bậc tự do df tính theo công thức $df = (\bar{x} - 1)$
Trong tình huống các nhóm n_i; khác nhau thì ở $\bar{x} = \frac{\sum_{i=1}^{k} n_i}{k}$ (chú ý là nếu kết quả tính $\bar{x}$ là số thập phân thì ta lấy phần nguyên).
Quy tắc quyết định:

$F > F_{(k-1;n-k);\alpha}$ thì ta bác bỏ H_0 cho rằng phương sai bằng nhau và ngược lại.
Nếu chúng ta không chắc chắn về các giả định hoặc nếu kết quả kiểm định cho thấy các giả định hoặc nếu kết quả kiểm định cho thấy các giả định không được thỏa mãn thì một phương pháp kiểm định thay thế cho ANOVA là phương pháp kiểm định phi tham số Krusksl-Wallis sẽ được áp dụng. Tuy nhiên trong ví dụ này ở đây, ta có thể xem các giả định để tiến hành phân tích phương sai đã được thỏa mãn.

Phân tích sâu ANOVA Mục đích của phân tích phương sai là kiểm định giả thuyết H_0 rằng trung bình của các tổng thể bằng nhau. Sau khi phân tích và kết luận, có hai trường hợp xảy ra là chấp nhận giả thuyết H_0 hoặc bác bỏ giả thuyết H_0. Nếu chấp nhận giả thuyết H_0 thì phân tích kết thúc. Nếu bác bỏ giả thuyết H_0, kết luận trung bình của các tổng thể không bằng nhau. Vì vậy, vấn để tiếp theo là phân tích sâu hơn để xác định nhóm (tổng thể) nào khác nhóm nào, nhóm nào có trung bình lớn hơn hay nhỏ hơn.

Có nhiều phương pháp để tiếp tục phân tích sâu ANOVA khi bác bỏ giả thuyết H_0. Trong chương này chỉ để cập đến 1 phương pháp thông dụng đó là phương pháp Tukey, phương pháp này còn

được gọi là kiểm định HSD (Honestly Significant Differences). Nội dung của phương pháp này là so sánh từng cặp các trung bình nhóm ở mức ý nghĩa α nào đó cho tất cả các cặp kiểm định có thể để phát hiện ra những nhóm khác nhau. Nếu có k nhóm nghiên cứu, và chúng ta so sánh tất cả các cặp nhóm thì số lương cặp cần phải so sánh là tổ hợp chập 2 của k nhóm.

$$C_k^n = \frac{k!}{2!(k-2)!} = \frac{k(k-1)}{2}$$

Giá trị giới hạn Turkey được tính theo công thức:

$$T = q_{\alpha;n-k}\sqrt{\frac{MSW}{n_i}}$$

Trong đó:

+ $q_{\alpha;n-k}$ là giá trị tra được thông qua bảng phân phối kiểm định Turkey ở mức ý nghĩa α, với bậc tự do k và $n-k$, với n là tổng số mẫu quan sát ($n = \sum n_i$).

+ MSW là phương sai trong nội bộ nhóm

+ n_i là số quan sát trong 1 nhóm (tổng thể), trong trường hợp mỗi nhóm có số quan sát n_i khác nhau, sử dụng giá trị n_i nhỏ nhất.

Tiêu chuẩn quyết định là bác bỏ giả thuyết H_0 khi độ lệch giữa các cặp trung bình mẫu lớn hơn hoặc bằng T giới hạn. Bên cạnh việc kiểm định để phát hiện ra những nhóm khác biệt, chúng ta có thể tìm khoảng ước lượng cho chênh lệch giữa các nhóm có khác biệt có ý nghĩa thống kê. Ước lượng khoảng về chênh lệch giữa hai trung bình nhóm có khác biệt tính theo công thức:

$$\sigma_1 - \sigma_2 = (\bar{x_1} - \bar{x_2}) \pm (t_{n-k;\frac{\alpha}{2}}\sqrt{\frac{2MSW}{n_i}})$$

Trong đó t là giá trị tra từ bảng phân phối Student t với $(n-k)$ bậc tự do.

3 Tiền xử lí số liệu

Đọc dữ liệu ở tệp "gpus.cvs" bằng câu lệnh `read.csv()`. Sau khi đọc dữ liệu ta bắt đầu trích xuất những thuộc tính đặc trưng của mỗi tệp đã trình bài ở phần giới thiệu và lưu thành tệp mới `newGPU`. Từ đoạn này về sau ta mọi thao tác đều được thao tác trên tệp mới. Đọc và lấy ra các cột

dữ liệu cần quan tâm (chuyển các ô dữ liệu bị khuyết hoặc không có dữ liệu thành NA để tiện thao tác sau này):

```
GPU=read.csv("/kaggle/input/computerparts/All_GPUs.csv",header=TRUE,na.strings=c("","\n- ","\n","\nUnknown Release Date "))
GPU_new=GPU[,c("Memory","Resolution_WxH",,"Manufacturer",,"Core_Speed",     "Memory_Bus","Memory_Speed","Memory_Type","Process","Pixel_Rate",,"Texture_Rate")]
```

Tiếp theo ta tạo 1 bản tóm tắt thống kê của dataframe của `GPU_new`

```
summary(GPU_new)
```

```
print( summary(GPU_new) )
```

```
   Memory          Resolution_WxH     Manufacturer        Core_Speed       
 Length:3406        Length:3406        Length:3406        Length:3406       
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
  Memory_Bus        Memory_Speed       Memory_Type          Process          
 Length:3406        Length:3406        Length:3406        Length:3406       
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
  Pixel_Rate         Texture_Rate                                            
 Length:3406        Length:3406                                             
 Class :character   Class :character                                        
 Mode  :character   Mode  :character
```

Tất cả các biến đều có định dạng là ký tự và có số lượng là 3406.

Kiểm tra các biến thiếu Ta tính tổng số giá trị thiếu (NA) trong mỗi cột của dataframe `GPU_new`

```
print( apply(is.na(GPU_new),2,sum) )
```

```
print( apply(is.na(GPU_new),2,sum) )
```

```
      Memory Resolution_WxH   Manufacturer     Core_Speed      Memory_Bus 
         420            195              0            936             62 
Memory_Speed    Memory_Type        Process   Pixel_Rate   Texture_Rate 
         105             56            463            544            544
```

Trong đó, biến `Memory_Speed`, `Memory_Bus`, `Memory_Type` có tỷ lệ giá trị khuyết đều bé hơn 3%.

Chuyển đổi dữ liệu Ta định nghĩa một hàm để tách các ký tự với nhau bằng khoảng trắng và chuyển về dạng số thực:

```
helper <- function(x){
    if(is.na(x))
        {return (NA)}
    else
        {return (as.double( strsplit(x," ")[[1]][[1]] ))
    }
}
```

- Chuyển đổi dữ liệu biến `Core_Speed`

  Chuyển đổi từ character về double:

```
GPU_new$Core_Speed <- sapply(GPU_new$Core_Speed, helper)
```

- Chuyển đổi dữ liệu biến `Memory`

  Chuyển đổi từ character về double:

```
GPU_new$Memory = sapply(GPU_new$Memory, helper)
```

- Chuyển đổi dữ liệu biến `Memory_Bus`

  Chuyển đổi từ character về double:

```
GPU_new$Memory_Bus = sapply(GPU_new$Memory_Bus, helper)
```

- Chuyển đổi dữ liệu biến `Memory_Speed`

  Chuyển đổi từ character về double:

```
GPU_new$Memory_Speed = sapply(GPU_new$Memory_Speed, helper)
```

- Chuyển đổi dữ liệu biến `Pixel_Rate`

  Chuyển đổi từ character về double:

```
GPU_new$Pixel_Rate = sapply(GPU_new$Pixel_Rate, helper)
```

- Chuyển đổi dữ liệu biến `Process`

  Chuyển đổi từ character về double:

```
GPU_new$Process = as.double( gsub("[^0-9]","",GPU_new$Process) )
```

- Chuyển đổi dữ liệu biến `Texture_Rate`

  Chuyển đổi từ character về double:

```
GPU_new$Texture_Rate = sapply(GPU_new$Texture_Rate, helper)
```

Xử lí các giá trị khuyết

- Xử lý biến `Memory_Type` Ta thay thế các missing value bằng mode

```
GPU_new$Memory_Type[is.na(GPU_new$Memory_Type)] = "GDDR5"
GPU_new$Memory_Type = gsub("[^A-Za-z]+.*","",GPU_new$Memory_Type)
```

- Xử lý biến `Resolution_WxH`

```
GPU_new$Resolution_WxH[is.na(GPU_new$Resolution_WxH)] = "4096x2160"
```

  Vì giá trị "4096x2160" là mode (giá trị xuất hiện nhiều nhất, chiếm 39%) nên thay thế các giá trị khuyết bằng giá trị này. Sau đó, chúng ta thực hiện gom nhóm: 4096x2160 (39%), 2560x1600 (34%), Other (26%).

```
GPU_new$Resolution_WxH <- ifelse(GPU_new$Resolution_WxH == "4096x2160", 1,
                      ifelse(GPU_new$Resolution_WxH == "2560x1600", 2, 3))
```

  Và thực hiện chuyển đổi thành factor vì đây là một biến phân nhóm (categorical variable)

```
GPU_new$Resolution_WxH = factor(GPU_new$Resolution_WxH)
```

- Xử lý biến `Memory_Bus` Ta thay thế các missing value bằng mode

```
GPU_new$Memory_Bus[is.na(GPU_new$Memory_Bus)] = 128
```

- Xử lý biến `Memory_Speed`

```
GPU_new <- GPU_new[complete.cases(GPU_new$Memory_Speed), ]
```

  Chuyển đổi biến về số thực, vì tỷ lệ giá trị khuyết thấp hơn 3% so với mẫu nên loại bỏ.

- Xử lý biến `Core_Speed`

```
GPU_new$Core_Speed[is.na(GPU_new$Core_Speed)]=median(GPU_new$Core_Speed, na
    .rm=T)
```

  Chúng ta chuyển đổi biến về dạng số thực và thay thế các giá trị khuyết bằng trung vị.

- Xử lý biến `Memory`

```
GPU_new$Memory[is.na(GPU_new$Memory)] = median(GPU_new$Memory, na.rm=T)
```

Thực hiện chuyển đổi biến về số thực và thay thế giá trị khuyết bằng trung vị.

- Xử lý biến `Process`

```
GPU_new$Process[is.na(GPU_new$Process)] = median(GPU_new$Process, na.rm=T)
```

Thay thế các giá trị khuyết bằng trung vị

- Xử lý biến `Pixel_Rate`

```
GPU_new$Pixel_Rate[is.na(GPU_new$Pixel_Rate)] = median(GPU_new$Pixel_Rate,
    na.rm=T)
```

Thay thế các giá trị khuyết bằng trung vị

- Xử lý biến `Texture_Rate`

```
GPU_new$Texture_Rate[is.na(GPU_new$Texture_Rate)]=median(GPU_new$Texture_
    Rate, na.rm=T)
```

Thay thế các giá trị khuyết bằng trung vị

Loại bỏ các giá trị trùng lặp Ta tiến hành các bước sau

```
str(GPU_new)
```

```
# Loại bỏ các giá trị trùng lặp
str(GPU_new)
```

```
'data.frame':   3285 obs. of  10 variables:
 $ Memory        : num  1024 512 512 256 256 ...
 $ Resolution_WxH: Factor w/ 3 levels "1","2","3": 2 2 2 2 2 2 2 2 1 3 ...
 $ Manufacturer  : Factor w/ 4 levels "AMD","ATI","Intel",..: 4 1 1 1 1 1 1 1 1 1 ...
 $ Core_Speed    : num  738 980 980 980 980 980 870 980 980 980 ...
 $ Memory_Bus    : num  256 512 256 128 128 128 256 256 256 64 ...
 $ Memory_Speed  : num  1000 828 800 1150 700 1100 1050 800 1250 366 ...
 $ Memory_Type   : Factor w/ 4 levels "DDR","eDRAM",..: 3 3 3 3 3 3 3 3 3 1 ...
 $ Process       : num  55 80 80 65 65 65 55 80 28 28 ...
 $ Pixel_Rate    : num  12 12 10 3 3 3 14 7 25 26 ...
 $ Texture_Rate  : num  47 12 10 7 6 6 35 7 62 60 ...
```

Trước khi loại bỏ ta có 3301 hàng với 10 cột (10 biến)

```
GPU_new <- GPU_new %>% dplyr::distinct()
str(GPU_new)
```

```
GPU_new <- GPU_new %>% dplyr::distinct()
str(GPU_new)
```

```
'data.frame':   2210 obs. of  10 variables:
 $ Memory        : num  1024 512 512 256 256 ...
 $ Resolution_WxH: Factor w/ 3 levels "1","2","3": 2 2 2 2 2 2 2 2 1 3 ...
 $ Manufacturer  : Factor w/ 4 levels "AMD","ATI","Intel",..: 4 1 1 1 1 1 1 1 1 1 ...
 $ Core_Speed    : num  738 980 980 980 980 980 870 980 980 980 ...
 $ Memory_Bus    : num  256 512 256 128 128 128 256 256 256 64 ...
 $ Memory_Speed  : num  1000 828 800 1150 700 1100 1050 800 1250 366 ...
 $ Memory_Type   : Factor w/ 4 levels "DDR","eDRAM",..: 3 3 3 3 3 3 3 3 3 1 ...
 $ Process       : num  55 80 80 65 65 65 55 80 28 28 ...
 $ Pixel_Rate    : num  12 12 10 3 3 3 14 7 25 26 ...
 $ Texture_Rate  : num  47 12 10 7 6 6 35 7 62 60 ...
```

Sau khi loại bỏ, ta còn 2222 hàng

Chuyển sang dạng log Ta thực hiện đoạn code sau để Biến đổi thành dataframe mới tên là `GPU_new_log`.

```
GPU_new_log <- GPU_new
GPU_new_log[,c("Memory","Resolution_WxH","Core_Speed","Memory_Bus","Memory_Speed
    ","Memory_Type","Process","Pixel_Rate","Texture_Rate","Manufacturer")] <- log
    (GPU_new_log[,c("Memory","Core_Speed","Memory_Bus","Memory_Speed","Process","
    Pixel_Rate","Texture_Rate")])
```

Giải thích lý do chuyển sang dạng log

- Mối quan hệ phi tuyến tính: Mối quan hệ giữa biến phụ thuộc và biến độc lập không tuyến tính với nhau, tức là biến độc lập ảnh hưởng đối với biến phụ thuộc không theo cách tuyến tính, mô hình hồi quy thông thường có thể không đáp ứng được. Khi mô hình hồi quy cần mô hình hóa một quan hệ phi tuyến, sử dụng logarit có thể giúp biểu diễn mối quan hệ theo cách linh hoạt và dễ dàng diễn giải.

- Biến độc lập có biên độ lớn: Khi biến độc lập có biên độ lớn và tác động không đồng nhất trên biến phụ thuộc, có thể xảy ra hiện tượng hiệu ứng không chính xác hoặc không đồng đều. Bằng cách áp dụng logarit, chúng ta giảm điều lo âu làm cho mô hình trở nên ổn định, đơn giản trong việc diễn giải và tiếp cận.

- Biến phụ thuộc không tuân theo phân phối chuẩn: Nếu biến phụ thuộc không tuân theo phân phối chuẩn, điều này có thể ảnh hưởng đến giả định của mô hình hồi quy. Áp dụng logarit có thể giúp làm cho phân phối của biến phụ thuộc gần với phân phối chuẩn hơn, từ đó cải thiện chất lượng của mô hình.

Quá trình áp dụng mô hình hồi quy với việc sử dụng logarit đòi hỏi sự cẩn trọng và chặt chẽ. Việc đưa ra quyết định này nên dựa trên quá trình kiểm tra chi tiết và sự hiểu biết sâu sắc về bản chất của dữ liệu. Điều này là chìa khóa quan trọng để đảm bảo rằng mô hình hồi quy không chỉ phản ánh chính xác mối quan hệ giữa các biến mà còn đáp ứng đúng nhu cầu và yêu cầu của vấn đề nghiên cứu.

Kiểm tra các thông số âm ta có kết quả sau

```
for(var in numerical) {
    result <- any(GPU_new[[var]] < 0)
    print(paste(var, ":", result))
}
```

```
[1] "Memory : FALSE"
[1] "Core_Speed : FALSE"
[1] "Memory_Bus : FALSE"
[1] "Memory_Speed : FALSE"
[1] "Process : FALSE"
[1] "Pixel_Rate : FALSE"
[1] "Texture_Rate : FALSE"
```

+ Code + Markdown

```
for(var in numerical) {
    result <- any(GPU_new_log[[var]] < 0)
    print(paste(var, ":", result))
}
```

```
[1] "Memory : FALSE"
[1] "Core_Speed : FALSE"
[1] "Memory_Bus : FALSE"
[1] "Memory_Speed : FALSE"
[1] "Process : FALSE"
[1] "Pixel_Rate : FALSE"
[1] "Texture_Rate : FALSE"
```

Tổng kết Ta tạo bảng tóm tắt các cột dữ liệu và tính tổng số giá trị thiếu (NA) trong mỗi cột của dataframe `GPU_new_log` sau khi xử lý missing value.

```
summary(GPU_new_log)
```

```
    Memory       Resolution_WxH Manufacturer    Core_Speed      Memory_Bus
Min.   : 2.773   1:1023         AMD   : 895   Min.   :4.605   Min.   :3.466
1st Qu.: 6.931   2: 827         ATI   :  55   1st Qu.:6.712   1st Qu.:4.852
Median : 7.625   3: 360         Intel :  73   Median :6.888   Median :4.852
Mean   : 7.457                  Nvidia:1187   Mean   :6.810   Mean   :5.147
3rd Qu.: 8.318                                3rd Qu.:6.937   3rd Qu.:5.545
Max.   :10.373                                Max.   :7.487   Max.   :9.011
 Memory_Speed   Memory_Type     Process        Pixel_Rate     Texture_Rate
Min.   :4.605   DDR  : 597   Min.   :2.639   Min.   :0.000   Min.   :0.000
1st Qu.:6.685   eDRAM:   3   1st Qu.:3.332   1st Qu.:2.303   1st Qu.:3.178
Median :6.976   GDDR :1599   Median :3.332   Median :3.258   Median :4.094
Mean   :6.939   HBM  :  11   Mean   :3.417   Mean   :2.973   Mean   :3.879
3rd Qu.:7.315                3rd Qu.:3.689   3rd Qu.:3.638   3rd Qu.:4.771
Max.   :7.662                Max.   :5.011   Max.   :5.561   Max.   :6.575
```

+ Code + Markdown

```
print( apply(is.na(GPU_new_log),2,sum) )
```

```
      Memory Resolution_WxH   Manufacturer     Core_Speed     Memory_Bus
           0              0              0              0              0
Memory_Speed    Memory_Type        Process     Pixel_Rate   Texture_Rate
           0              0              0              0              0
```

Đến đây xem như đã hoàn thành việc tiền xử lý số liệu bởi bộ dữ liệu đã được loại bỏ những chỗ bị khuyết cũng như đã được xử lý sơ bộ trước khi tiến hành thống kê. Sau đây, ta sẽ in ra 10 dòng đầu tiên của dataframe `GPU_new_log`. Điều này giúp ta có cái nhìn tổng quan về dữ liệu bằng cách hiển thị một phần của dataframe.

```
1 head(GPU_new_log, 10)
```

A data.frame: 10 × 10

	Memory	Resolution_WxH	Manufacturer	Core_Speed	Memory_Bus	Memory_Speed	Memory_Type	Process	Pixel_Rate	Texture_Rate
	<dbl>	<fct>	<fct>	<dbl>	<dbl>	<dbl>	<fct>	<dbl>	<dbl>	<dbl>
1	6.931472	2	Nvidia	6.603944	5.545177	6.907755	GDDR	4.007333	2.484907	3.850148
2	6.238325	2	AMD	6.887553	6.238325	6.719013	GDDR	4.382027	2.484907	2.484907
3	6.238325	2	AMD	6.887553	5.545177	6.684612	GDDR	4.382027	2.302585	2.302585
4	5.545177	2	AMD	6.887553	4.852030	7.047517	GDDR	4.174387	1.098612	1.945910
5	5.545177	2	AMD	6.887553	4.852030	6.551080	GDDR	4.174387	1.098612	1.791759
6	5.545177	2	AMD	6.887553	4.852030	7.003065	GDDR	4.174387	1.098612	1.791759
7	7.624619	2	AMD	6.768493	5.545177	6.956545	GDDR	4.007333	2.639057	3.555348
8	5.545177	2	AMD	6.887553	5.545177	6.684612	GDDR	4.382027	1.945910	1.945910
9	7.624619	1	AMD	6.887553	5.545177	7.130899	GDDR	3.332205	3.218876	4.127134
10	4.158883	3	AMD	6.887553	4.158883	5.902633	DDR	3.332205	3.258097	4.094345

+ Code + Markdown

4 Thống kê tả

4.1 Thông tin tổng quát về các biến của bộ dữ liệu

Ta lập bảng tính các giá trị thống kê mô tả (trung bình, độ lệch chuẩn, min, max, trung vị) cho các biến định lượng trong `GPU_new`

```
mean<-apply(GPU_new[,numerical],2,mean)
sd<-apply(GPU_new[,numerical],2,sd)
min<-apply(GPU_new[,numerical],2,min)
max<-apply(GPU_new[,numerical],2,max)
median<-apply(GPU_new[,numerical],2,median)
data.frame(mean,sd,min,max,median)
```

	mean	sd	min	max	median
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Memory	2732.81810	2754.29012	16	32000	2048
Core_Speed	938.34661	223.79016	100	1784	980
Memory_Bus	213.68326	293.25158	32	8192	128
Memory_Speed	1136.04434	441.52563	100	2127	1071
Process	32.99367	15.76054	14	150	28
Pixel_Rate	32.46606	34.09740	1	260	26
Texture_Rate	84.55475	87.29824	1	717	60

Bảng các giá trị thống kê mô tả cho GPU mới

Tương tự, ta cũng lập bảng tính các giá trị thống kê mô tả cho các biến định lượng trong `GPU_new_log`

```r
mean<-apply(GPU_new_log[,numerical],2,mean)
sd<-apply(GPU_new_log[,numerical],2,sd)
min<-apply(GPU_new_log[,numerical],2,min)
max<-apply(GPU_new_log[,numerical],2,max)
median<-apply(GPU_new_log[,numerical],2,median)
data.frame(mean,sd,min,max,median)
```

	mean	sd	min	max	median
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Memory	2732.81810	2754.29012	16	32000	2048
Core_Speed	938.34661	223.79016	100	1784	980
Memory_Bus	213.68326	293.25158	32	8192	128
Memory_Speed	1136.04434	441.52563	100	2127	1071
Process	32.99367	15.76054	14	150	28
Pixel_Rate	32.46606	34.09740	1	260	26
Texture_Rate	84.55475	87.29824	1	717	60

Bảng các giá trị thống kê mô tả cho `GPU_new_log`

Ta thấy nếu biểu diễn dưới dạng bảng như này thì khó có thể hình dung dữ liệu như thế nào hay có liên hệ với nhau ra sao. Do đó ta sẽ biểu diễn các dữ liệu này dưới dạng các biểu đồ để dễ quan sát hơn.

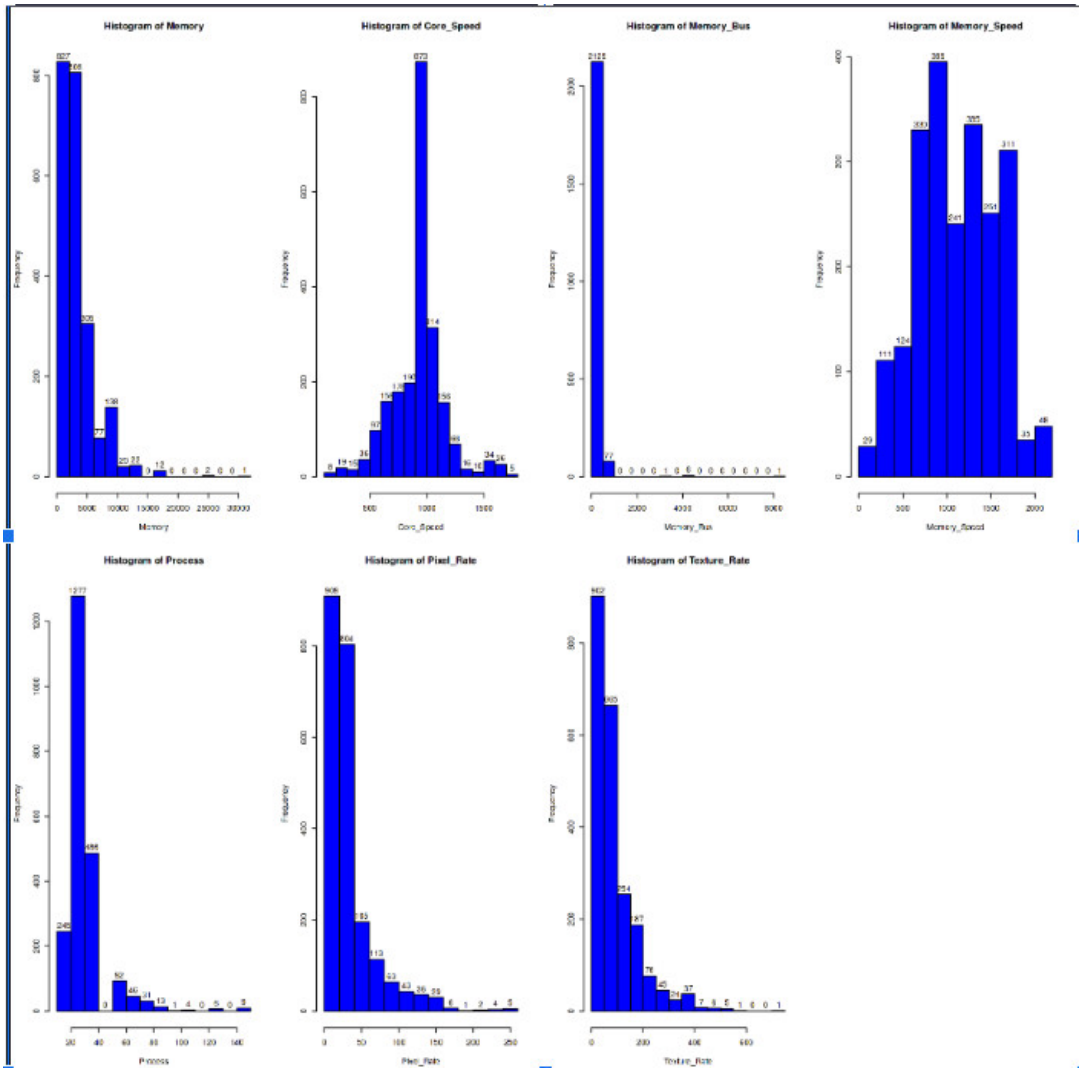
4.2 Biểu đồ - đồ thị

Biểu đồ tần suất (histogram) thường dùng để thể hiện phân phối của biến định lượng. Ta vẽ biểu đồ tần suất cho các dữ liệu `GPU_new` và `GPU_new_log`.

Ta tiến hành vẽ biểu đồ tần suất cho dữ liệu `GPU_new`.

```r
# make the layout in the format of 2 rows and 4 columns
par(mfrow=c(2,4))
# Draw histogram for each numerical variable of GPU_new
for (i in 1:length(numerical)) {
  hist_data <- GPU_new[[numerical[i]]]
  hist(hist_data,
```

```
        xlab = names(GPU_new)[which(names(GPU_new) == numerical[i])],
        main = paste("Histogram of", names(GPU_new)[which(names(GPU_new) ==
    numerical[i])]),
        labels = TRUE,
        col = "blue")
}
```

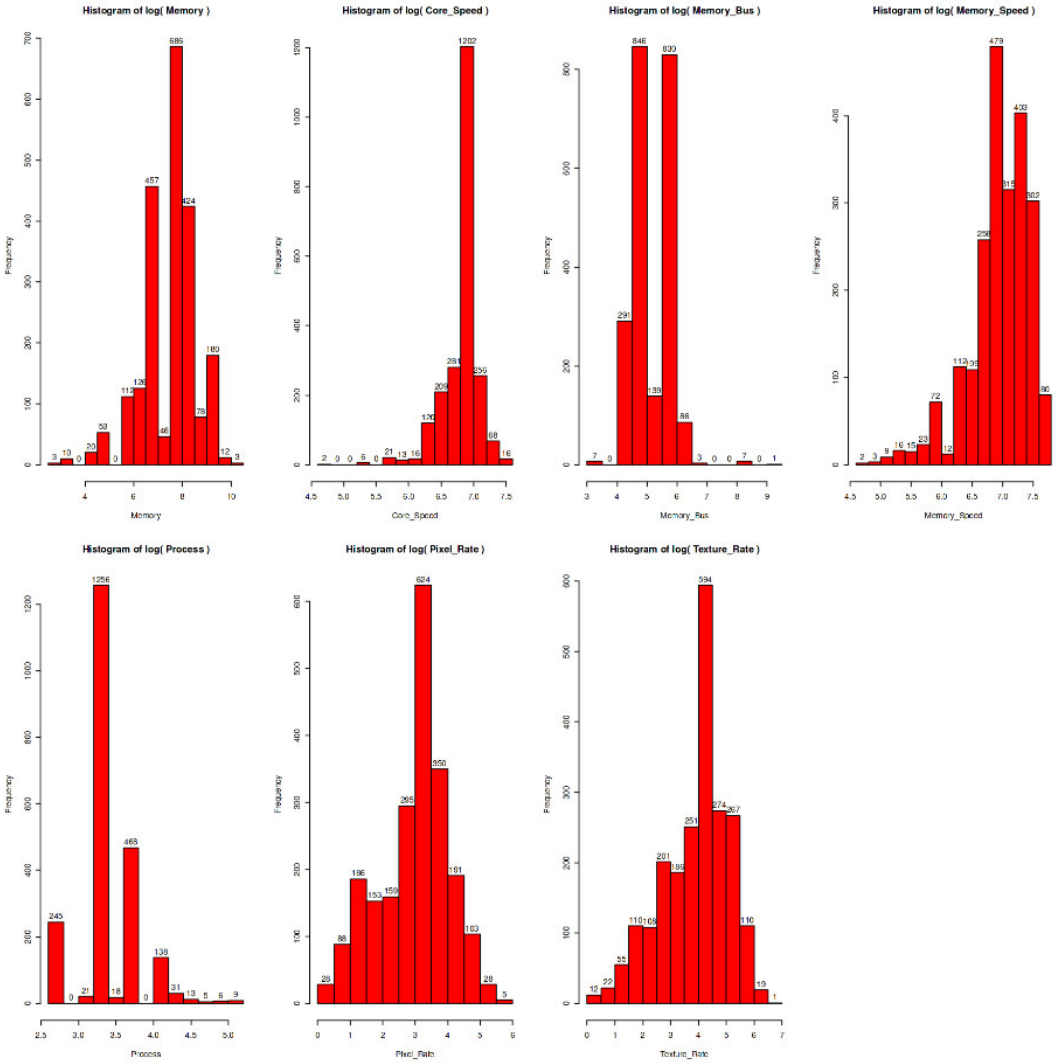


Biểu đồ tần suất của các biến trong `GPU_new`

Nhận xét: Ta thấy các biến `Memory`, `Process`, `Pixel_Rate` và `Textur_Rate` có phân phối lệch phải. Biến `Core_Speed` có phân phối gần giống phân phối chuẩn. Trong khi đó biến `Memory_Speed` có hầu hết các cột cao gần bằng nhau ngoại trừ các cột ở biên, còn biến `Memory_Bus` gần như chỉ có một cột, các cột còn lại không đáng kể.

Ta tiến hành vẽ biểu đồ tần suất cho dữ liệu `GPU_new_log`.

```r
# make the layout in the format of 2 rows and 4 columns
par(mfrow=c(2, 4))
# Draw histogram for each numerical variable of GPU_new_log
for (i in 1:length(numerical)) {
  hist_data <- GPU_new_log[[numerical[i]]]
  hist(hist_data,
       xlab = names(GPU_new_log)[which(names(GPU_new_log) == numerical[i])],
       main = paste("Histogram of log(", names(GPU_new_log)[which(names(GPU_new_log) == numerical[i])], ")"),
       labels = TRUE,
       col = "red")
}
```



Biểu đồ tần suất của các biến trong `GPU_new_log`

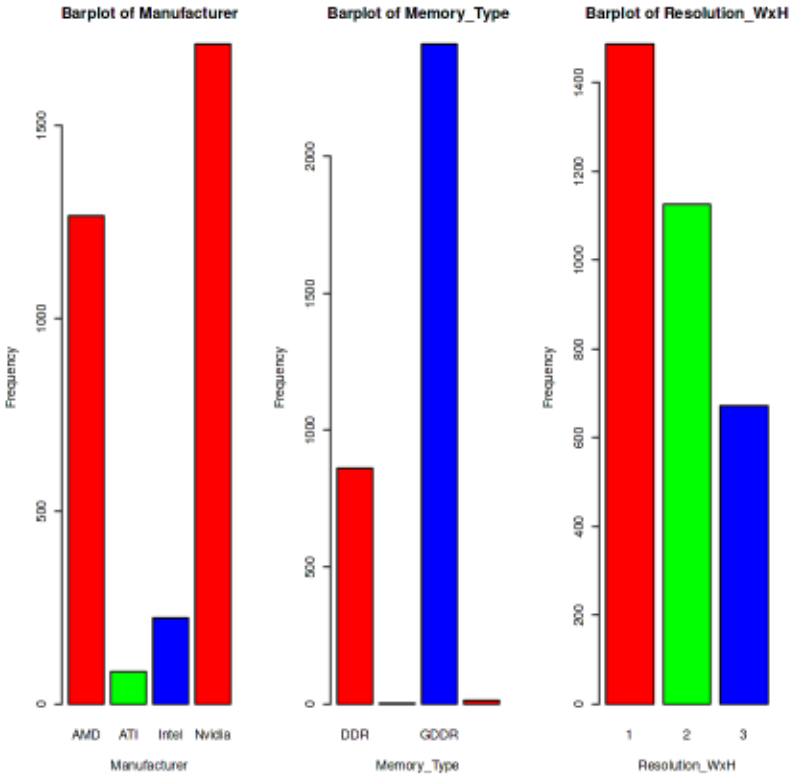
Nhận xét: Ta thấy sau khi chuyển đổi các biến x sang dạng $log(x)$ thì phân phối các biến trở

nên gần giống với phân phối chuẩn hơn, ví dụ như biến `Memory_Bus` hay `Memory_Speed`. Các biến `Memory`, `Process`, `Pixel_Rate` và `Texture_Rate` cũng không còn phân phối lệch như trước.

Biểu đồ cột (Barplot) thường dùng để thể hiện phân phối giữa các biến định tính.

```
par(mfrow=c(1,3))
barplot(table(GPU_new$Manufacturer), xlab="Manufacturer", ylab="Frequency", main
   ="Barplot of Manufacturer", col=c("red","green","blue"))
barplot(table(GPU_new$Memory_Type), xlab="Memory_Type", ylab="Frequency", main="
   Barplot of Memory_Type", col=c("red","green","blue"))
barplot(table(GPU_new$Resolution_WxH), xlab="Resolution_WxH", ylab="Frequency",
   main="Barplot of Resolution_WxH", col=c("red","green","blue"))
```

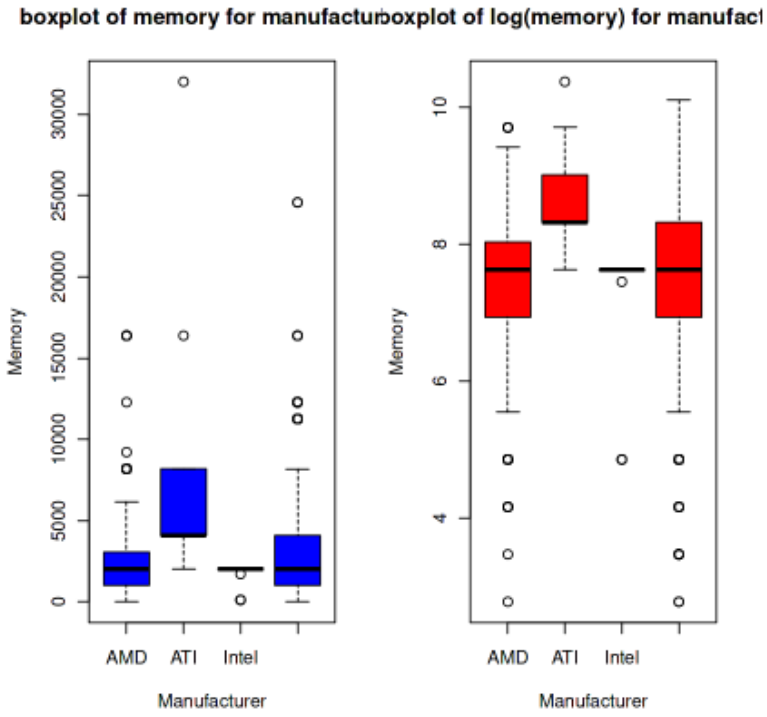
Biểu đồ cột thể hiện tần suất của các biến `Manufacturer`, `Memory_Type` và `Resolution_WxH` trong `GPU_new`



Nhận xét: Ta thấy hầu hết GPU đều được sản xuất bởi AMD và Nvidia, 2 nhà sản xuất còn lại (ATI và Intel) không đáng kể. Chỉ có 2 kiểu bộ nhớ là DDR và GDDR, và đa số thuộc về GDDR. Về chỉ số `Resolution_WxH` thì ta thấy không có sự chênh lệch đáng kể giữa 3 cột như 2 biến kia.

Biểu đồ boxplot là biểu đồ diễn tả 5 vị trí phân bố của dữ liệu: giá trị nhỏ nhất (min), tứ phân vị thứ nhất (Q1), trung vị (median), tứ phân vị thứ 3 (Q3) và giá trị lớn nhất (max). Ta vẽ biểu đồ boxplot thể hiện phân phối của biến `Memory` và log(`Memory`) theo từng phân loại của các biến rời rạc.

```
par(mfrow=c(1,2))
boxplot(Memory~Manufacturer, data=GPU_new, main="boxplot of memory for
    manufacturer", col="blue")
boxplot(Memory~Manufacturer, data=GPU_new_log, main="boxplot of log(memory) for
    manufacturer", col="red")
```

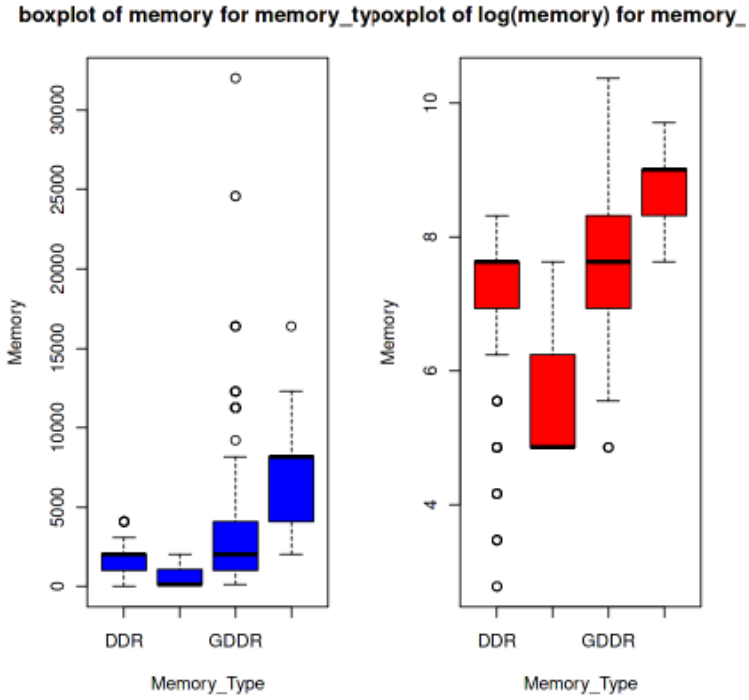


Biểu đồ boxplot thể hiện phân phối của biến `Memory` và log(`Memory`) theo từng phân loại của biến `Manufacturer`

Nhận xét: Ta thấy với nhà sản xuất là AMD và Nvidia thì dung lượng bộ nhớ có phân phối hơi lệch phải, còn với ATI thì lệch trái. Nhưng sau khi lấy log(`Memory`) thì cả 4 phân phối đều trở thành phân phối chuẩn.

```
par(mfrow=c(1,2))
boxplot(Memory~Memory_Type, data=GPU_new, main="boxplot of memory for memory_
    type", col="blue")
```

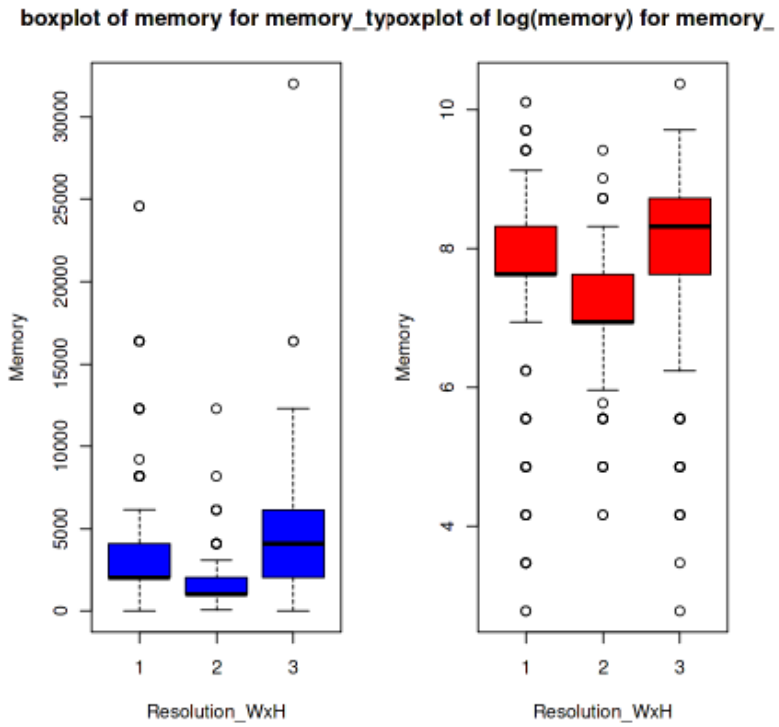
```
boxplot(Memory~Memory_Type, data=GPU_new_log, main="boxplot of log(memory) for
    memory_type", col="red")
```



Biểu đồ boxplot thể hiện phân phối của biến `Memory` và log(`Memory`) theo từng phân loại của biến `Memory_Type`

Nhận xét: Với kiểu bộ nhớ GDDR thì dữ liệu hơi lệch phải, còn eDRAM thì hoàn toàn lệch phải. Sau khi lấy log thì phân phối của GDDR đã trở nên phân phối chuẩn, nhưng với eDRAM thì vẫn còn bị lệch.

```
par(mfrow=c(1,2))
boxplot(Memory~Memory_Type, data=GPU_new, main="boxplot of memory for memory_
    type", col="blue")
boxplot(Memory~Memory_Type, data=GPU_new_log, main="boxplot of log(memory) for
    memory_type", col="red")
```



Biểu đồ boxplot thể hiện phân phối của biến `Memory` và log(`Memory`) theo từng phân loại của biến `Resolution_WxH`

Nhận xét: Các phân phối ở 2 hình trên đều gần với phân phối chuẩn kể cả trước và sau khi dùng hàm $log(x)$.

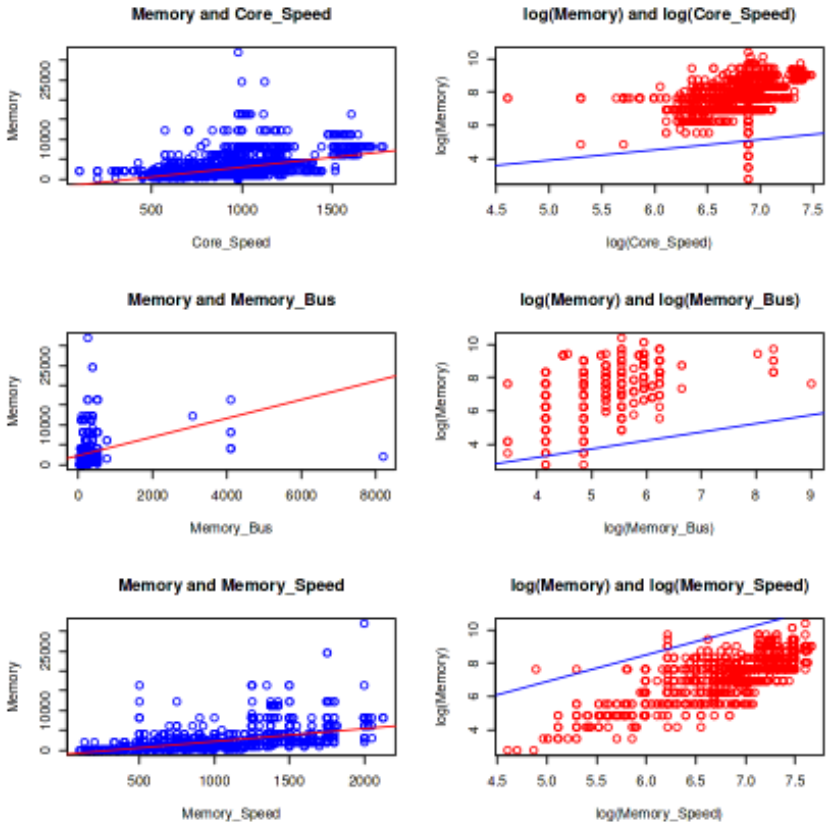
Biểu đồ phân tán là biểu đồ thường được dùng để thể hiện mối liên hệ giữa 2 biến định lượng. Ta vẽ biểu đồ phân tán giữa biến Memory với một số biến định lượng khác.

```
par(mfrow=c(1,2))
plot(GPU_new[, "Core_Speed"], GPU_new[, "Memory"], xlab = "Core_Speed", ylab = "
    Memory", main = "Memory and Core_Speed", col = "blue")
fit_new <- lm(Memory ~ Core_Speed, data = GPU_new)
abline(fit_new, col = "red")
plot(GPU_new_log[, "Core_Speed"], GPU_new_log[, "Memory"], xlab = "log(Core_
    Speed)", ylab = "log(Memory)", main = "log(Memory) and log(Core_Speed)", col
    = "red")
fit_new_log <- lm(log(Memory) ~ log(Core_Speed), data = GPU_new_log)
abline(fit_new_log, col = "blue")

```

```r
plot(GPU_new[, "Memory_Bus"], GPU_new[, "Memory"], xlab = "Memory_Bus", ylab = "Memory", main = "Memory and Memory_Bus", col = "blue")
fit_new <- lm(Memory ~ Memory_Bus, data = GPU_new)
abline(fit_new, col = "red")
plot(GPU_new_log[, "Memory_Bus"], GPU_new_log[, "Memory"], xlab = "log(Memory_Bus)", ylab = "log(Memory)", main = "log(Memory) and log(Memory_Bus)", col = "red")
fit_new_log <- lm(log(Memory) ~ log(Memory_Bus), data = GPU_new_log)
abline(fit_new_log, col = "blue")

plot(GPU_new[, "Memory_Speed"], GPU_new[, "Memory"], xlab = "Memory_Speed", ylab = "Memory", main = "Memory and Memory_Speed", col = "blue")
fit_new <- lm(Memory ~ Memory_Speed, data = GPU_new)
abline(fit_new, col = "red")
plot(GPU_new_log[, "Memory_Speed"], GPU_new_log[, "Memory"], xlab = "log(Memory_Speed)", ylab = "log(Memory)", main = "log(Memory) and log(Memory_Speed)", col = "red")
fit_new_log <- lm(log(Memory) ~ log(Memory_Speed), data = GPU_new_log)
abline(fit_new_log, col = "blue")
```



Biểu đồ phân tán giữa biến `Memory` với các biến `Core_Speed`, `Memory_Bus` và `Memory_Speed` ở dữ liệu `GPU_new` và `GPU_new_log`

Nhận xét: Ta thấy biến `Memory` có quan hệ tuyến tính tăng với biến `Core_Speed`. Điều này còn thể hiện rõ hơn giữa biến `Memory` với `Memory_Speed`. Và gần như không có mối liên hệ nào giữa `Memory` và `Memory_Bus`.

5 Thống kê suy diễn

Ở đây, nhóm nghiên cứu chọn biến `memory` là biến phụ thuộc và tiến hành xây dựng mô hình hồi quy để nghiên cứu sự ảnh hưởng của các nhân tố khác đến bộ nhớ của một GPU. Các biến độc lập bao gồm:

-Biến liên tục (numeric variable): `memory`, `Core_Speed`, `Memory_Bus`, `Memory_Speed`, `Process`, `Pixel_Rate`, `Texture_Rate`.

-Biến phân loại (categorical variable): `Resolution_WxH`, `Memory_Type`, `Manufacturer`.

5.1 Đánh giá mối quan hệ giữa các biến

Ở đây chúng ta sử dụng hệ số tương quan, là một chỉ số thống kê đo lường mối liên hệ tương quan giữa hai biến số, như `Memory` và `Core_Speed`. Hệ số tương quan có giá trị từ -1 đến 1. Hệ số tương quan bằng 0 (hay gần 0) có nghĩa là hai biến số không có liên hệ gì với nhau; ngược lại nếu hệ số bằng -1 hay 1 có nghĩa là hai biến số có một mối liên hệ tuyệt đối. Nếu giá trị của hệ số tương quan là âm, có nghĩa là khi x tăng cao thì y giảm và ngược lại, khi x giảm thì y tăng. Nếu hệ số tương quan là dương, nghĩa là khi x tăng thì y cũng tăng và x giảm thì y cũng giảm.

Hệ số tương quan Pearson:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Trong đó $\bar{x}$ và $\bar{y}$ là giá trị trung bình của biến số x và y.

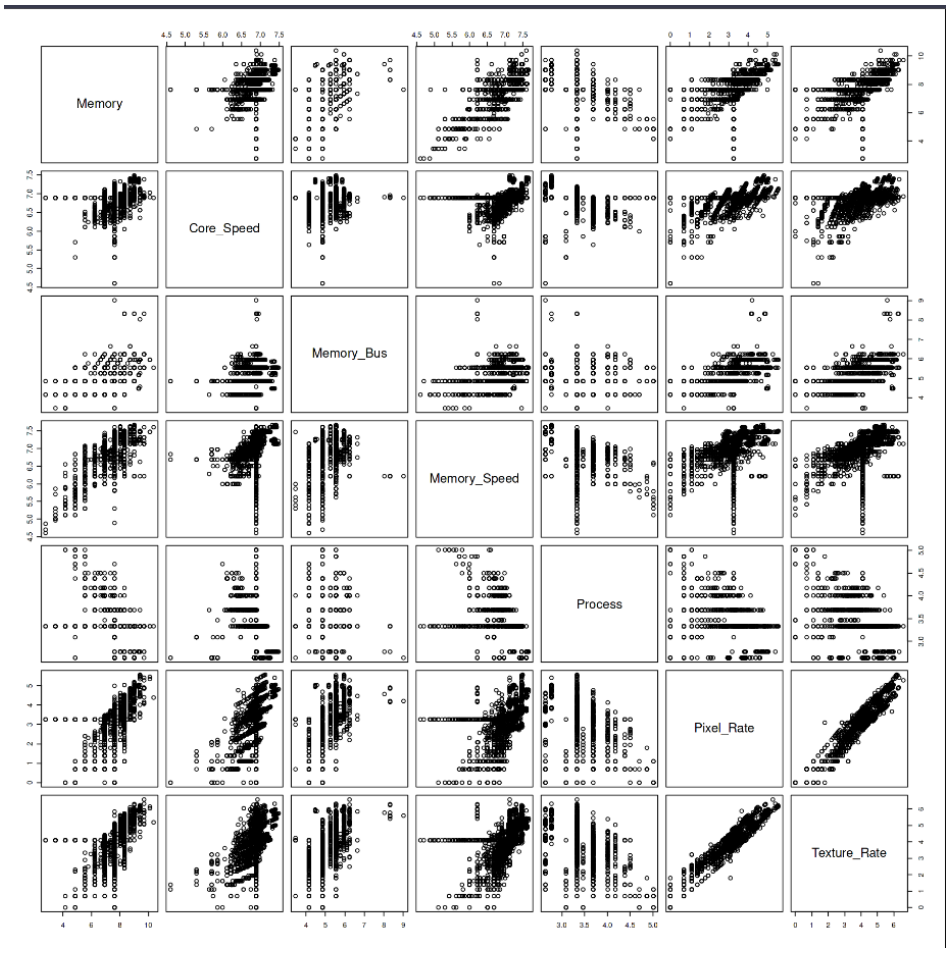
Chúng ta có thể kiểm định giả thiết hệ số tương quan bằng 0 (hai biến không có liên hệ) bằng hàm cor.test().

	Memory	Pearson_Correlation	P_Value
cor	Memory	1.0000000	0.000000e+00
cor1	Core_Speed	0.2281184	1.759540e-27
cor2	Memory_Bus	0.4135984	4.568408e-92
cor3	Memory_Speed	0.7347999	0.000000e+00
cor4	Process	-0.4569130	1.976354e-114
cor5	Pixel_Rate	0.4648248	7.060714e-119
cor6	Texture_Rate	0.5017833	2.838068e-141

Hệ số tương quan giữa các biến

Như vậy, biến phụ thuộc `Memory` có quan hệ thống kê với những biến `Memory_Speed`, `Process`, `Pixel_Rate`, `Texture_Rate` (p-value < 0.05).

```
1 pairs(GPU_new_log[numerical])
```



Đồ thị biểu diễn mối tương quan giữa các biến

Biểu đồ này cho ta biết mối liên hệ giữa các biến với nhau.

5.2 Xây dựng mô hình hồi quy tuyến tính

Đầu tiên, ta thực hiện chia tập dữ liệu thành tập train và test với tỉ lệ 2:1 và xây dựng mô hình hồi quy tuyến tính, lưu kết quả vào biến model

```r
index <- createDataPartition(GPU_new_log$Memory, p = 2/3, list = FALSE)
train <- GPU_new_log[index,]
test <- GPU_new_log[index,]
model <- lm(Memory ~ ., data = train)
print( summary(model) )
```

```r
print( summary(model) )
```

```
Call:
lm(formula = Memory ~ ., data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-2.45646 -0.40165 -0.03426  0.35772  2.95189

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       -3.06735    0.70670  -4.340 1.52e-05 ***
Resolution_WxH2   -0.09148    0.04453  -2.054 0.040112 *
Resolution_WxH3   -0.21111    0.05304  -3.980 7.22e-05 ***
ManufacturerATI   -0.01074    0.11990  -0.090 0.928637
ManufacturerIntel  0.37197    0.11424   3.256 0.001155 **
ManufacturerNvidia 0.01185    0.03594   0.330 0.741687
Core_Speed        -0.19095    0.08177  -2.335 0.019673 *
Memory_Bus         0.26383    0.04400   5.996 2.55e-09 ***
Memory_Speed       1.66351    0.04966  33.498  < 2e-16 ***
Memory_TypeeDRAM  -1.95458    0.37273  -5.244 1.80e-07 ***
Memory_TypeGDDR   -0.57650    0.05609 -10.279  < 2e-16 ***
Memory_TypeHBM     1.31887    0.28274   4.665 3.38e-06 ***
Process           -0.28207    0.07258  -3.886 0.000106 ***
Pixel_Rate         0.18964    0.05535   3.426 0.000629 ***
Texture_Rate      -0.05544    0.05177  -1.071 0.284317
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.619 on 1460 degrees of freedom
Multiple R-squared:  0.6603,    Adjusted R-squared:  0.657
F-statistic: 202.7 on 14 and 1460 DF,  p-value: < 2.2e-16
```

Kết quả tổng kết từ mô hình

Dựa vào kết quả (cột Estimate) ta có được hệ số Beta của mô hình và từ đó có được công thức của

biến log(`Memory`) là:

$$\begin{aligned}\log(\text{Memory}) = &-3.0674 - 0.0915 \times \log(\text{Resolution\_WxH}^2) \\ &- 0.2111 \times \log(\text{Resolution\_WxH}^3) \\ &- 0.0107 \times \log(\text{ManufacturerATI}) \\ &+ 0.3720 \times \log(\text{ManufacturerIntel}) \\ &+ 0.0119 \times \log(\text{ManufacturerNvidia}) \\ &- 0.1910 \times \log(\text{Core\_Speed}) \\ &+ 0.2638 \times \log(\text{Memory\_Bus}) \\ &+ 1.6635 \times \log(\text{Memory\_Speed}) \\ &- 1.9546 \times \log(\text{Memory\_TypeeDRAM}) \\ &- 0.5765 \times \log(\text{Memory\_TypeGDDR}) \\ &+ 1.3189 \times \log(\text{Memory\_TypeHBM}) \\ &- 0.2821 \times \log(\text{Process}) \\ &+ 0.1896 \times \log(\text{Pixel\_Rate}) \\ &- 0.0554 \times \log(\text{Texture\_Rate}) \\ &+ 1.3189 \times \log(\text{Memory\_TypeHBM}) \\ &- 0.2821 \times \log(\text{Process}) \\ &+ 0.1896 \times \log(\text{Pixel\_Rate}) \\ &- 0.0554 \times \log(\text{Texture\_Rate})\end{aligned}$$

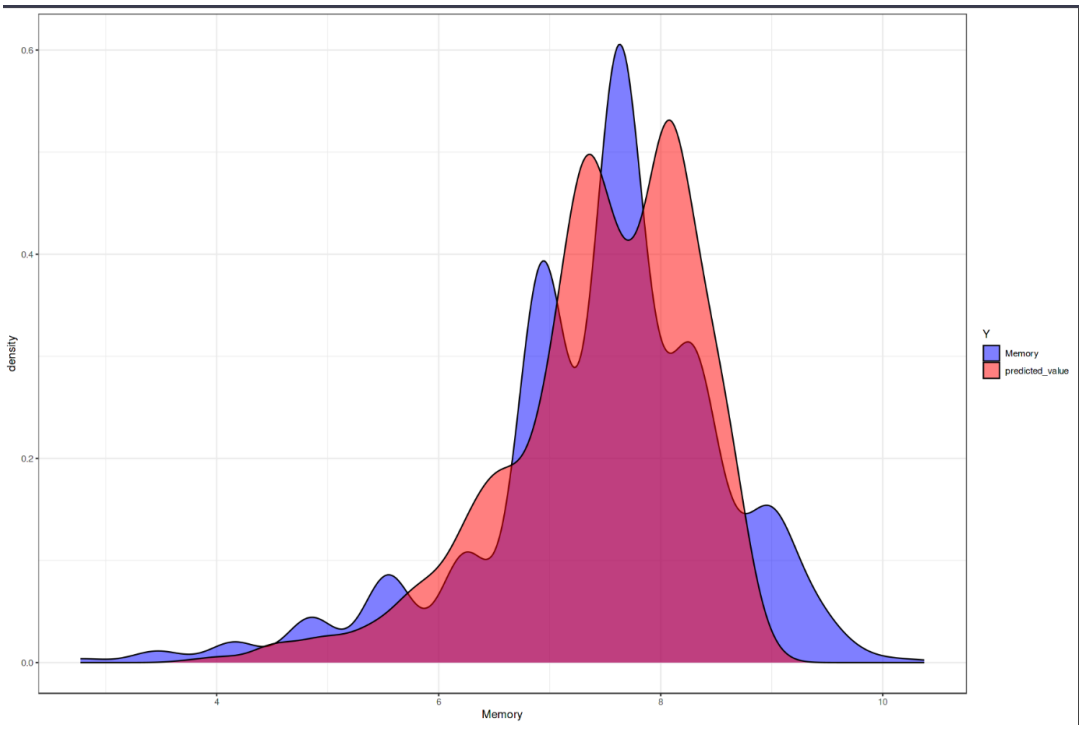
Nhận xét: Sai số tiêu chuẩn (residuals standard error) là 0.619 Hệ số xác định là 0.6603 và hệ số xác định điều chỉnh là 0.657. Nghĩa là khoảng 66% sự biến thiên của `Memory` được giải thích bởi các biến độc lập. Đối với mức ý nghĩa (ta chọn là 5%) thì các biến `ManufacturerATI`, `ManufacturerNvidia`, `Texture_Rate` không có ý nghĩa trong mô hình.

Trong đó, ta có thông tin về phần dư của mô hình hồi quy (sai số của giá trị quan sát và giá trị dự đoán từ mô hình):

```
Residuals:
     Min       1Q   Median       3Q      Max
-2.45646 -0.40165 -0.03426  0.35772  2.95189
```

Phần dư của mô hình

Chúng ta biết rằng trung bình phần dư càng gần 0 càng tốt, đại biểu ý nghĩa là giá trị dự đoán từ mô hình càng gần giá trị quan sát được. Ở đây, trung vị là -0.03426 rất gần 0, số tứ phân vị thứ nhất và thứ ba cũng khá cân đối xung quanh số trung vị. Cột thứ tư là giá trị kiểm định t (t = Estimate/ Std. Error) cho các biến và cột thứ năm là trị số p-value dùng cho kiểm định giả thiết. Để kiểm tra mức độ chính xác của mô hình, chúng ta thử nghiệm trên tập test:



Biểu đồ biểu diễn giá trị quan sát được và giá trị tiên đoán

Trong đó, màu xanh là giá trị `Memory` quan sát được, còn màu đỏ là giá trị dự đoán được tính toán từ mô hình. Có thể thấy mô hình dự đoán khác tốt, những phần trùng nhau chiếm diện tích lớn. Chúng ta có thể kiểm chứng lại với các thông số MAE (Mean Absolute Error trung bình của sai biệt tuyệt đối) và MSE (Mean Squared Error trung bình bình phương sai số), các giá trị càng nhỏ thì mô hình càng chính xác, công thức cụ thể:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
mae_value <- mae(predicted_value, test$Memory)
mse_value <- mse(predicted_value, test$Memory)
# In ket qua
cat("MAE (trung binh cua sai so tuyet doi): ", mae_value, "\n")
cat("MSE (trung binh binh phuong sai so): ", mse_value, "\n")
```

```
MAE (trung bình của sai biệt tuyệt đối):  0.4985635
MSE (trung bình bình phương sai số):  0.4292266
```

Tính giá trị MAE và MSE

5.3 Phân tích sự khác biệt giữa các nhóm phân loại

Để đánh giá `Memory` trung bình giữa các nhóm dựa trên các biến phân loại (`Manufacturer`, `Memory_Type`, `Resolution_WxH`), ta phân tích phương sai giữa các nhóm bằng lệnh aov().

```
model_ANOVA <- aov(Memory ~ Manufacturer + Resolution_WxH + Memory_Type, data=
    GPU_new_log)
summary(model_ANOVA)
```

```
model_ANOVA <- aov(Memory ~ Manufacturer + Resolution_WxH + Memory_Type, data= GPU_new_log)
summary(model_ANOVA)
```

```
               Df Sum Sq Mean Sq F value  Pr(>F)
Manufacturer    3   60.7   20.23   23.84 3.5e-15 ***
Resolution_WxH  2  310.6  155.32  183.03 < 2e-16 ***
Memory_Type     3  269.0   89.68  105.68 < 2e-16 ***
Residuals    2201 1867.7    0.85
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Phân tích phương sai dựa trên các biến phân loại

Ta có giả thuyết H_0:

- `Memory` trung bình ở các Manufacture khác nhau là bằng nhau.

- `Memory` trung bình ở các `Resolution_WxH` khác nhau là bằng nhau.

- `Memory` trung bình ở các `Memory_Type` khác nhau là bằng nhau.

Quan sát kết quả nhận được, ta thấy trị số p-value của tất cả các biến đều bé hơn mức ý nghĩa (ở đây chọn là 5%), cho nên đủ cơ sở để bác bỏ giả thuyết H0. Đồng nghĩa với việc `Memory` trung bình khác nhau giữa các biến phân loại.

Để thấy được sự khác biệt giữa từng cặp giá trị trong mỗi nhóm, ta thực hiện so sánh bội sau ANOVA bằng lệnh TukeyHSD

```
TukeyHSD(model_ANOVA)
```

```
TukeyHSD(model_ANOVA)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Memory ~ Manufacturer + Resolution_WxH + Memory_Type, data = GPU_new_log)

$Manufacturer
                    diff         lwr        upr     p adj
ATI-AMD       1.07740319  0.74838587  1.4064205 0.0000000
Intel-AMD     0.12972483 -0.15855520  0.4180049 0.6541290
Nvidia-AMD    0.08815360 -0.01669274  0.1930000 0.1343890
Intel-ATI    -0.94767836 -1.37055338 -0.5248033 0.0000001
Nvidia-ATI   -0.98924958 -1.31591557 -0.6625836 0.0000000
Nvidia-Intel -0.04157122 -0.32716471  0.2440223 0.9821277

$Resolution_WxH
          diff        lwr         upr     p adj
2-1 -0.8011879 -0.9022156 -0.70016021 0.0000000
3-1 -0.1580034 -0.2903970 -0.02560971 0.0143089
3-2  0.6431845  0.5067682  0.77960091 0.0000000

$Memory_Type
                 diff        lwr       upr     p adj
eDRAM-DDR  -0.8526308 -2.2234420 0.5181805 0.3792829
GDDR-DDR    0.6975355  0.5839417 0.8111293 0.0000000
HBM-DDR     1.5833434  0.8627038 2.3039829 0.0000001
GDDR-eDRAM  1.5501662  0.1815042 2.9188282 0.0190050
HBM-eDRAM   2.4359741  0.8933616 3.9785867 0.0002963
HBM-GDDR    0.8858079  0.1692650 1.6023508 0.0081751
```

Phân tích hậu định bằng TukeyHSD

Đối với biến `Manufacturer`, ta thấy Nvidia-Intel có p-value bằng 0.9821, Intel-AMD là 0.6541 và Nvidia-Intel là 0.9821 là lớn hơn 5% cho nên không đủ bằng chứng để bác bỏ H0. Có nghĩa là `Memory` trung bình giữa Nvidia và Intel, giữa Intel và AMD và giữa Nvidia và Intel là bằng nhau Tương tự, ở `Memory_Type` thì có eDRAM và DDR là có `Memory` trung bình bằng nhau với p-value bằng 0.3793 lớn hơn 5Những nhóm còn lại có p-value bé hơn 5% nên ta đủ cơ sở để khẳng định `Memory` trung bình giữa các nhóm là khác nhau và sự sai biệt này có ý nghĩa thống kê.

6 Thảo luận và mở rộng

Trong bài tập lớn, đối với các biến định lượng chứa tỉ lệ dữ liệu khuyết nhỏ hơn 3%, nhóm sẽ sử dụng phương pháp xóa các quan sát chứa dữ liệu khuyết hoặc thay thế thành trung vị của những dữ liệu đã có với tỉ lệ lớn hơn 3%. Đối với biến định tính, nhóm sẽ thay thế các dữ liệu khuyết bằng trung vị. Dưới đây là một số ưu điểm và nhược điểm của phương pháp này.

Ưu điểm:

- Bảo toàn thông tin quan trọng: Khi tỷ lệ NA lớn, việc thay thế có thể giúp bảo toàn thông tin quan trọng trong dữ liệu thay vì loại bỏ hoàn toàn các quan sát có chứa NA. Điều này có thể quan trọng nếu có sự không đồng đều trong việc xuất hiện NA trong các biến. Giảm mất mát dữ liệu: Bằng cách kết hợp thay thế và xóa, có thể giảm mất mát dữ liệu so với việc loại bỏ tất cả các quan sát hoặc thay thế toàn bộ biến.
- Áp dụng linh hoạt: Chiến lược này có thể được điều chỉnh để phù hợp với đặc điểm cụ thể của dữ liệu và yêu cầu của mô hình.

Nhược điểm:

- Phức tạp hóa quá trình xử lý: Khi kết hợp cả thay thế và xóa, quá trình xử lý dữ liệu có thể trở nên phức tạp hơn so với việc chỉ sử dụng một phương pháp.
- Đòi hỏi sự quan sát chặt chẽ: Việc quyết định nên thay thế hay xóa dựa trên tỷ lệ NA có thể đòi hỏi sự quan sát chặt chẽ và kiểm soát cẩn thận để đảm bảo rằng quyết định được đưa ra một cách hợp lý.
- Không áp dụng cho tất cả các trường hợp: Chiến lược này có thể không phù hợp cho tất cả các tình huống. Đôi khi, có những biện pháp xử lý dữ liệu khác như sử dụng mô hình dự đoán để điền giá trị NA có thể mang lại kết quả tốt hơn.

7 Nguồn dữ liệu và nguồn code

Nguồn dữ liệu từ Kaggle: link ở đây.

Nguồn code của nhóm: link ở đây

Tài liệu

[1] Nguyễn Đình Huy (chủ biên) và Nguyễn Bá Thi. *Giáo trình Xác suất và Thống kê*. Nhà Xuất Bản Đại Học Quốc Gia Hồ Chí Minh, 2018.