ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN
XÁC SUẤT THỐNG KÊ - MT2013

PHÂN TÍCH HIỆU SUẤT DỰNG ẢNH CỦA GPU MÁY TÍNH DỰA TRÊN CÁC THÔNG SỐ KỸ THUẬT

GVHD: Ths. Huỳnh Thái Duy Phương
Lớp: DL04
Nhóm: 9
Học kỳ: 243

Thành phố Hồ Chí Minh, ngày 15 tháng 08 năm 2025

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN
XÁC SUẤT THỐNG KÊ - MT2013

PHÂN TÍCH HIỆU SUẤT DỰNG ẢNH CỦA GPU MÁY TÍNH DỰA TRÊN CÁC THÔNG SỐ KỸ THUẬT

DANH SÁCH THÀNH VIÊN

STT	Sinh viên thực hiện	MSSV	Nhiệm vụ	Tiến độ
1	Vương Nhật Minh	2212094	Chương 4, 5, 6, 7, làm slide	100%
2	Trần Trung Nghĩa	2412278	Chương 1, 2, 3, báo cáo	100%
2	Trần Xuân Tùng	2033820		
4	Phạm Ngọc Đức Toàn	2313496		
5	Nguyễn Lâm	2311822		

Thành phố Hồ Chí Minh, ngày 15 tháng 08 năm 2025

NHẬN XÉT VÀ CHẤM ĐIỂM CỦA GIẢNG VIÊN

Mục lục

Chương 1

TỔNG QUAN NGHIÊN CỨU

1.1 Tổng quan bộ dữ liệu

- **Tên tập tin:** All_GPUs.csv
- **Nguồn:** https://www.kaggle.com/datasets/iliassekkaf/computerparts/
- **Số lượng quan trắc:** 3406 quan trắc tương ứng với 3406 hàng trong tệp dữ liệu.
- **Số lượng biến:** 34 biến tương ứng với 34 cột trong tệp dữ liệu.
- **Mô tả chung:** Bộ tập tin All_GPUs.csv là một nguồn dữ liệu phong phú chứa thông số kỹ thuật của GPU, phục vụ cho việc phân tích và dự đoán hiệu suất. Nó cung cấp cả thông tin định lượng như Max_Power, Memory, ROPs và định tính như Manufacturer và Dedicated, đòi hỏi các bước tiền xử lý như làm sạch, chuẩn hóa, và biến đổi để phù hợp với phân tích thống kê và mô hình hóa. Kết quả từ phân tích dữ liệu này có thể hỗ trợ các nhà nghiên cứu, kỹ sư, hoặc nhà sản xuất trong việc đánh giá và thiết kế GPU.

1.2 Đối tượng nghiên cứu

Mục tiêu chính là dự đoán thông số Pixel_Rate (tốc độ xử lý pixel, đơn vị GPixel/s), một chỉ số hiệu suất quan trọng của GPU, dựa trên các đặc trưng khác như Boost_Clock, Core_Speed, Memory_Type, v.v.

Tại sao lại chọn Pixel_Rate ?

1. **Pixel_Rate là chỉ số trực tiếp phản ánh hiệu năng đồ họa**:

   Pixel_Rate phản ánh trực tiếp hiệu suất của GPU trong việc xử lý các tác vụ đồ họa liên quan đến pixel, một khía cạnh cốt lõi của hiệu năng GPU. Trong các ứng dụng

thực tế, Pixel_Rate ảnh hưởng đến chất lượng hình ảnh, độ mượt mà của chuyển động, và khả năng hiển thị ở độ phân giải cao, khiến nó trở thành một chỉ số phù hợp để đo lường hiệu năng tổng thể. So với các chỉ số khác như số lượng shader hoặc dung lượng bộ nhớ, Pixel_Rate là kết quả trực tiếp của nhiều đặc trưng kỹ thuật (như ROPs, Core_Speed), giúp nó trở thành biến mục tiêu lý tưởng để phân tích mối quan hệ với các đặc trưng khác.

2. **Pixel_Rate là biến định lượng, phù hợp với mô hình hồi quy:**

   Biến định lượng liên tục như Pixel_Rate cho phép áp dụng các kỹ thuật phân tích như tương quan Pearson, hồi quy tuyến tính đa biến, và đánh giá hiệu suất mô hình bằng các chỉ số như MAE, MSE, RMSE, R^2. Các biến định lượng khác (như Boost_Clock, Core_Speed) có thể đóng vai trò là biến độc lập để dự đoán Pixel_Rate, tạo ra một mô hình dự đoán rõ ràng và dễ diễn giải.

3. **Pixel_Rate tổng hợp ảnh hưởng của nhiều đặc trưng GPU:**

   Pixel_Rate là một chỉ số tổng hợp, phản ánh hiệu quả của nhiều thành phần phần cứng trong GPU. Do đó, nó là một biến mục tiêu lý tưởng để nghiên cứu cách các đặc trưng khác nhau (như ROPs, Core_Speed) kết hợp để tạo ra hiệu năng tổng thể. So với các chỉ số khác, các chỉ số khác như Shader (số lượng đơn vị shader) hoặc Memory (dung lượng bộ nhớ) chỉ phản ánh một khía cạnh cụ thể của GPU, trong khi Pixel_Rate là kết quả tổng hợp của nhiều yếu tố phần cứng, do đó đại diện tốt hơn cho hiệu năng tổng thể.

1.3 Phương pháp nghiên cứu

- **Khám phá dữ liệu (EDA):** Sử dụng histogram, barplot, boxplot, scatter plot để hiểu phân phối và mối quan hệ giữa các biến.
- **Tiền xử lý:** Chuẩn hóa dữ liệu, xử lý NA, biến đổi log, loại bỏ trùng lặp.
- **Thống kê mô tả:** Tính mean, sd, min, max, median để mô tả đặc điểm dữ liệu.
- **Thống kê suy diễn:** Tương quan Pearson, ANOVA, kiểm định giả thuyết (Shapiro, Levene), và Tukey HSD để đánh giá mối quan hệ và sự khác biệt.
- **Mô hình hóa:** Hồi quy tuyến tính đa biến với lựa chọn biến (stepwise), cross-validation, và đánh giá đa cộng tuyến (VIF).

- **Dự báo:** Sử dụng mô hình để dự đoán Pixel_Rate và đánh giá hiệu suất trên tập kiểm tra.

1.4 Kết quả hướng tới

1.4.1 Mô hình dự đoán

Xây dựng một mô hình hồi quy tuyến tính để dự đoán Pixel_Rate dựa trên các đặc trưng quan trọng như Core_Speed, ROPs, và Memory_Speed. Mô hình sẽ được kiểm tra để đảm bảo độ chính xác cao và có thể áp dụng cho các tập dữ liệu mới, hỗ trợ việc dự đoán hiệu suất GPU một cách hiệu quả.

1.4.2 Hiểu biết sâu sắc

Xác định các yếu tố ảnh hưởng chính đến Pixel_Rate, bao gồm Core_Speed, ROP_Speed, Memory_Speed, và các đặc trưng khác như Manufacturer. Phân tích sẽ làm rõ cách các yếu tố này tác động đến hiệu suất, giúp hiểu rõ hơn về mối quan hệ giữa các thông số kỹ thuật của GPU.

1.4.3 Trực quan hóa

Tạo các biểu đồ như histogram để xem phân phối Pixel_Rate, boxplot để so sánh hiệu suất giữa các nhóm (như Manufacturer), và scatter plot để xem mối quan hệ với Core_Speed. Các bảng thống kê, như kết quả ANOVA, sẽ được thêm vào để cung cấp dữ liệu số rõ ràng, hỗ trợ việc đánh giá mô hình.

1.4.4 Ứng dụng thực tiễn

Sử dụng kết quả để so sánh hiệu suất giữa các GPU từ các nhà sản xuất khác nhau, dự đoán hiệu suất của GPU mới dựa trên thông số kỹ thuật, và đánh giá tác động của các cải tiến như tăng Core_Speed hoặc nâng cấp Memory_Type. Điều này giúp tối ưu hóa thiết kế và lựa chọn GPU phù hợp với nhu cầu cụ thể.

Chương 2

KIẾN THỨC NỀN

2.1 Đề Xuất Mô Hình Thống Kê và Lý Do Lựa Chọn

Để dự đoán hiệu suất đồ họa (Pixel_Rate) của các đơn vị xử lý đồ họa (GPU), mô hình hồi quy tuyến tính đa biến được lựa chọn làm công cụ cốt lõi, nổi bật với khả năng phân tích chính xác và cung cấp cái nhìn sâu sắc về các yếu tố ảnh hưởng đến hiệu suất. Phần này trình bày lý do lựa chọn mô hình này và khả năng đáp ứng các mục tiêu nghiên cứu.

2.1.1 Mô Hình Hồi Quy Tuyến Tính Đa Biến

Đề xuất: Mô hình hồi quy tuyến tính đa biến được chọn để dự đoán Pixel_Rate dựa trên một tập hợp các biến độc lập bao gồm các thông số kỹ thuật như tốc độ lõi (Core_Speed), tốc độ bộ nhớ (Memory_Speed), số đơn vị xuất hình ảnh (ROPs), số đơn vị ánh xạ texture (TMUs), độ rộng bus bộ nhớ (Memory_Bus), dung lượng bộ nhớ (Memory), quy trình sản xuất (Process), và số đơn vị shader (Shader), cùng với các biến phân loại như nhà sản xuất (Manufacturer), loại bộ nhớ (Memory_Type), và độ phân giải (Resolution_WxH).

Lý do lựa chọn:

- ***Tính cơ bản và mạnh mẽ:*** Hồi quy tuyến tính đa biến là một phương pháp thống kê nền tảng, được sử dụng rộng rãi để mô hình hóa mối quan hệ giữa một biến phụ thuộc liên tục như Pixel_Rate và nhiều biến độc lập khác. Với mục tiêu dự đoán hiệu suất đồ họa – một giá trị số liên tục – mô hình này là lựa chọn tự nhiên và phù hợp để khám phá mối liên hệ giữa các thông số kỹ thuật và hiệu suất.
- ***Khả năng phân tích đóng góp của từng biến:*** Một trong những điểm mạnh của mô hình này là khả năng cung cấp các hệ số hồi quy cho từng biến độc lập. Những hệ

số này không chỉ cho biết mức độ ảnh hưởng của từng yếu tố đến Pixel_Rate, mà còn giúp trả lời các câu hỏi thực tiễn như: "Tăng tốc độ lõi thêm 100 MHz sẽ cải thiện hiệu suất đồ họa bao nhiêu?", hoặc "Quy trình sản xuất nhỏ hơn (nm) có thực sự tạo ra sự khác biệt đáng kể không?". Điều này rất quan trọng để đạt được sự hiểu biết sâu sắc về các yếu tố kỹ thuật ảnh hưởng đến GPU.

- ***Tích hợp linh hoạt dữ liệu định lượng và định tính:*** Nghiên cứu bao gồm cả các biến số (như Core_Speed, Memory) và các biến phân loại (như Manufacturer, Memory_Type). Mô hình hồi quy tuyến tính đa biến cho phép chuyển đổi các biến phân loại thành biến giả (dummy variables), giúp kết hợp cả hai loại dữ liệu này một cách liền mạch. Ví dụ, biến Manufacturer có thể được mã hóa để so sánh hiệu suất giữa NVIDIA, AMD, và Intel, từ đó đánh giá tác động của yếu tố nhà sản xuất lên Pixel_Rate.

- ***Đánh giá hiệu suất đơn giản và thực tiễn:*** Mô hình này dễ dàng được đánh giá thông qua các chỉ số như hệ số xác định R^2 (đo lường mức độ giải thích biến thiên của Pixel_Rate), sai số tuyệt đối trung bình (MAE), sai số bình phương trung bình (MSE), và căn bậc hai của sai số bình phương trung bình (RMSE). Những chỉ số này không chỉ cung cấp cái nhìn rõ ràng về độ chính xác của dự đoán mà còn phù hợp với ứng dụng thực tiễn, khi cần một công cụ dễ triển khai và đáng tin cậy để dự báo hiệu suất GPU trong thực tế.

2.2 Công Cụ và phương pháp xây dựng mô hình

2.2.1 Phân Tích Tương Quan Pearson

Tổng quan: Tương quan Pearson là một công cụ thống kê cơ bản, dùng để đo lường mức độ và hướng của mối quan hệ tuyến tính giữa hai biến liên tục. Công cụ này xác định liệu hai biến có thay đổi cùng chiều (tăng hoặc giảm cùng nhau) hay ngược chiều, đồng thời đánh giá mức độ mạnh yếu của mối quan hệ đó. Kết quả được thể hiện qua một giá trị từ -1 đến 1, với giá trị gần 1 hoặc -1 cho thấy mối quan hệ mạnh, và gần 0 cho thấy mối quan hệ yếu.

Mục đích: Phân tích mối quan hệ giữa Pixel_Rate và các biến độc lập liên tục như Core_Speed, Memory_Speed, ROPs, TMUs, Memory_Bus, Memory, Process, và Shader để xác định các biến có ảnh hưởng đáng kể đến hiệu suất đồ họa. Kết quả này định hướng việc lựa chọn biến đưa vào mô hình hồi quy, đảm bảo chỉ những biến có

mối quan hệ tuyến tính mạnh được sử dụng, đồng thời hỗ trợ trực quan hóa dữ liệu qua các biểu đồ phân tán để phát hiện xu hướng hoặc bất thường.

Lý do lựa chọn:

- ***Hỗ trợ kiểm định ý nghĩa thống kê:*** Ngoài việc tính toán r, kiểm định tương quan Pearson cung cấp p-value để đánh giá liệu mối quan hệ giữa hai biến có ý nghĩa thống kê ($p < 0.05$) hay chỉ là ngẫu nhiên. Điều này giúp xác định các biến tiềm năng ảnh hưởng lớn đến Pixel_Rate trước khi xây dựng mô hình phức tạp hơn.
- ***Đơn giản và trực quan:*** Phương pháp này dễ thực hiện và cung cấp cái nhìn nhanh về mối quan hệ giữa các biến, làm nền tảng cho các phân tích sâu hơn như hồi quy tuyến tính. Nó cũng hỗ trợ việc trực quan hóa dữ liệu thông qua biểu đồ phân tán (scatter plot), giúp phát hiện các xu hướng tuyến tính hoặc bất thường trong dữ liệu.

2.2.2 Kiểm Định Phi Tham Số Kruskal-Wallis và Hậu Nghiệm Dunn

Tổng quan: Kruskal-Wallis test là một phương pháp kiểm định phi tham số dùng để so sánh giá trị trung vị của một biến số giữa nhiều nhóm độc lập. Đây là lựa chọn thay thế cho One-Way ANOVA khi dữ liệu không thỏa mãn các giả định như phân phối chuẩn hoặc phương sai đồng nhất. Phương pháp này dựa trên việc xếp hạng (ranking) toàn bộ dữ liệu và đánh giá sự khác biệt về phân bố giữa các nhóm. Khi Kruskal-Wallis cho thấy có sự khác biệt đáng kể, kiểm định hậu nghiệm Dunn được áp dụng để xác định cặp nhóm nào khác biệt, đồng thời điều chỉnh p-value để kiểm soát sai lầm loại I.

Mục đích: Sử dụng Kruskal-Wallis test để đánh giá sự khác biệt về Pixel_Rate giữa các nhóm được phân loại bởi các biến định tính như Manufacturer, Memory_Type, và Resolution_WxH trong trường hợp dữ liệu vi phạm giả định phân phối chuẩn hoặc đồng nhất phương sai. Nếu kiểm định cho kết quả có ý nghĩa thống kê ($p < 0.05$), áp dụng Dunn's post-hoc test với điều chỉnh Bonferroni hoặc Holm để xác định cặp nhóm nào có sự khác biệt đáng kể.

Lý do lựa chọn:

- ***Thích hợp khi dữ liệu không chuẩn:*** Trong nghiên cứu thực tế, dữ liệu Pixel_Rate có thể không tuân theo phân phối chuẩn hoặc có phương sai khác nhau giữa các nhóm. Kruskal-Wallis là phương pháp phi tham số, không yêu cầu giả định này, nên phù hợp để đảm bảo kết quả kiểm định đáng tin cậy.
- ***Khả năng áp dụng cho nhiều nhóm:*** Giống như ANOVA, Kruskal-Wallis có thể

xử lý các biến phân loại với từ ba nhóm trở lên, chẳng hạn Manufacturer có nhiều hãng sản xuất. Điều này giúp đánh giá tổng quát sự khác biệt giữa các nhóm mà không cần thực hiện nhiều phép so sánh đôi một.

- ***Phân tích chi tiết với Dunn's test:*** Khi phát hiện sự khác biệt tổng thể, Dunn's post-hoc test cho phép xác định cụ thể cặp nhóm nào khác biệt. Việc điều chỉnh p-value (như Bonferroni, Holm) giúp kiểm soát nguy cơ sai lầm loại I, đặc biệt khi số lượng so sánh lớn.

- ***Bổ sung cho ANOVA:*** Trong quy trình phân tích, Kruskal-Wallis + Dunn được dùng như phương án dự phòng khi ANOVA và kiểm định giả thiết cho thấy dữ liệu vi phạm điều kiện. Điều này đảm bảo rằng kết quả phân tích không bị sai lệch do sử dụng phương pháp không phù hợp với bản chất dữ liệu.

2.2.3 Kiểm Định Giả Thiết

Tổng quan: Kiểm định giả thiết là tập hợp các phương pháp thống kê dùng để kiểm tra tính hợp lệ của các giả định cần thiết cho các mô hình thống kê, chẳng hạn như phân phối chuẩn của dữ liệu hoặc tính đồng nhất phương sai giữa các nhóm. Các kiểm định này đảm bảo rằng các phân tích thống kê được thực hiện trên cơ sở đáng tin cậy, tránh sai lệch trong kết quả.

Mục đích: Sử dụng kiểm định Shapiro-Wilk để kiểm tra xem sai số của mô hình hoặc dữ liệu Pixel_Rate có phân phối chuẩn hay không, và kiểm định Levene để đánh giá tính đồng nhất phương sai giữa các nhóm được xác định bởi biến phân loại. Các kiểm định này xác minh các giả định cần thiết cho mô hình hồi quy và ANOVA, đảm bảo rằng kết quả dự đoán Pixel_Rate không bị sai lệch do vi phạm các điều kiện thống kê.

Lý do lựa chọn:

- ***Đảm bảo tính hợp lệ của kết quả:*** Để các kết luận từ hồi quy và ANOVA có giá trị, các giả định cơ bản của chúng (như sai số phân phối chuẩn và phương sai đồng nhất giữa các nhóm) phải được thỏa mãn. Nếu không kiểm tra và xác nhận những giả định này, các kết quả thống kê có thể bị sai lệch, làm giảm độ tin cậy của nghiên cứu.

2.2.4 Cross-Validation (Kiểm Chứng Chéo)

Tổng quan: Cross-validation là một kỹ thuật đánh giá hiệu suất mô hình bằng cách chia dữ liệu thành nhiều phần, sử dụng một phần để kiểm tra và các phần còn lại để huấn luyện mô hình, sau đó lặp lại quá trình này nhiều lần. Kỹ thuật này giúp đánh giá khả năng tổng quát hóa của mô hình trên dữ liệu mới, đảm bảo tính thực tiễn và độ ổn định.

Mục đích: Áp dụng k-fold cross-validation với k = 5 để đánh giá khả năng dự đoán Pixel_Rate của mô hình hồi quy tuyến tính đa biến trên các tập dữ liệu khác nhau. Kỹ thuật này đảm bảo rằng mô hình không bị overfitting, có khả năng dự đoán chính xác trên dữ liệu mới, và mang lại hiệu suất ổn định trong các kịch bản thực tế.

Lý do lựa chọn:

- ***Hỗ trợ ứng dụng thực tiễn:*** Mục tiêu yêu cầu mô hình không chỉ hoạt động tốt trên dữ liệu hiện tại mà còn phải dự đoán chính xác trên dữ liệu mới. Cross-validation giải quyết vấn đề này bằng cách chia dữ liệu thành 5 phần, lần lượt sử dụng 4 phần để huấn luyện và 1 phần để kiểm tra, qua đó đánh giá hiệu suất thực tế của mô hình. Điều này giúp giảm thiểu nguy cơ overfitting – tình trạng mô hình "học thuộc"dữ liệu huấn luyện nhưng thất bại khi dự đoán trên dữ liệu chưa từng thấy, một vấn đề thường gặp trong phân tích hiệu suất GPU.
- ***Đo lường độ ổn định và chính xác:*** Kết quả từ cross-validation cung cấp các chỉ số trung bình như R^2 và RMSE qua 5 lần kiểm tra, cho phép đánh giá mức độ ổn định và chính xác của mô hình trên các tập dữ liệu khác nhau.

2.2.5 Stepwise Selection (Lựa Chọn Biến Theo Bước)

Tổng quan: Stepwise selection là một phương pháp tự động chọn lọc các biến độc lập quan trọng nhất để đưa vào mô hình, dựa trên việc thêm hoặc loại bỏ biến theo các tiêu chí thống kê như p-value hoặc tiêu chí thông tin (AIC). Phương pháp này giúp tối ưu hóa mô hình bằng cách giảm độ phức tạp và tăng độ chính xác.

Mục đích: Sử dụng stepwise selection để xác định các biến độc lập quan trọng nhất, chẳng hạn như Core_Speed, ROPs, hoặc TMUs, nhằm xây dựng một mô hình hồi quy tuyến tính đa biến tối ưu. Phương pháp này giúp loại bỏ các biến không quan trọng, giảm độ phức tạp của mô hình, và nâng cao độ chính xác khi dự đoán Pixel_Rate.

Lý do lựa chọn:

- ***Tăng hiệu quả dự đoán:*** Với 11 biến độc lập trong nghiên cứu, không phải tất cả

đều có ý nghĩa thống kê hoặc đóng góp đáng kể vào việc dự đoán Pixel_Rate. Stepwise selection giúp loại bỏ các biến không quan trọng, giảm độ phức tạp của mô hình và cải thiện độ chính xác dự đoán. Điều này rất phù hợp với mục tiêu tìm mô hình dự đoán hiệu quả, vì một mô hình quá phức tạp có thể dẫn đến overfitting và giảm khả năng ứng dụng thực tế.

- ***Tự động hóa và tiết kiệm thời gian:*** Phương pháp này sử dụng các tiêu chí như giá trị p (p-value) hoặc chỉ số thông tin Akaike (AIC) để tự động thêm hoặc loại bỏ biến, thay vì yêu cầu nhà nghiên cứu thử nghiệm thủ công từng tổ hợp biến. Với số lượng biến lớn như trong nghiên cứu này, cách tiếp cận tự động hóa không chỉ tiết kiệm công sức mà còn đảm bảo tính khách quan trong quá trình xây dựng mô hình.

2.2.6 Diagnostic Plots (Đồ Thị Chẩn Đoán)

Tổng quan: Diagnostic plots là tập hợp các biểu đồ trực quan được sử dụng để kiểm tra tính đúng đắn của mô hình hồi quy, bao gồm các vấn đề như mối quan hệ phi tuyến, sai số không chuẩn, hoặc sự hiện diện của các điểm dữ liệu bất thường (outliers). Các biểu đồ này cung cấp cái nhìn trực quan về chất lượng mô hình, giúp phát hiện và khắc phục các vấn đề tiềm ẩn.

Mục đích: Sử dụng các đồ thị như Residuals vs Fitted, Normal Q-Q, Scale-Location, và Residuals vs Leverage để kiểm tra các giả định của mô hình hồi quy tuyến tính đa biến. Các biểu đồ này giúp phát hiện các vấn đề như mối quan hệ phi tuyến, sai số không chuẩn, hoặc outliers, từ đó điều chỉnh mô hình để đảm bảo độ tin cậy và chính xác khi dự đoán Pixel_Rate.

Lý do lựa chọn:

- ***Hỗ trợ hiểu biết sâu sắc:*** Các đồ thị này cung cấp cái nhìn trực quan về tính đúng đắn của mô hình, giúp phát hiện các vấn đề như mối quan hệ phi tuyến tính, sai số không chuẩn, hoặc sự hiện diện của các điểm dữ liệu bất thường (outliers/influential points). Ví dụ, đồ thị Residuals vs Fitted cho thấy liệu sai số có phân bố ngẫu nhiên hay có xu hướng nào đó, từ đó xác nhận giả định về tính tuyến tính.

- ***Đảm bảo kết quả đáng tin cậy:*** Việc kiểm tra các giả định thông qua đồ thị chẩn đoán giúp đảm bảo rằng mô hình được xây dựng đúng đắn và các dự đoán không bị sai lệch bởi các yếu tố không mong muốn. Điều này phù hợp với yêu cầu trực quan hóa và đánh giá mô hình, khi cần cả sự chính xác và khả năng trình bày kết quả một

cách rõ ràng.

2.2.7 Kiểm Tra Đa Cộng Tuyến (VIF - Variance Inflation Factor)

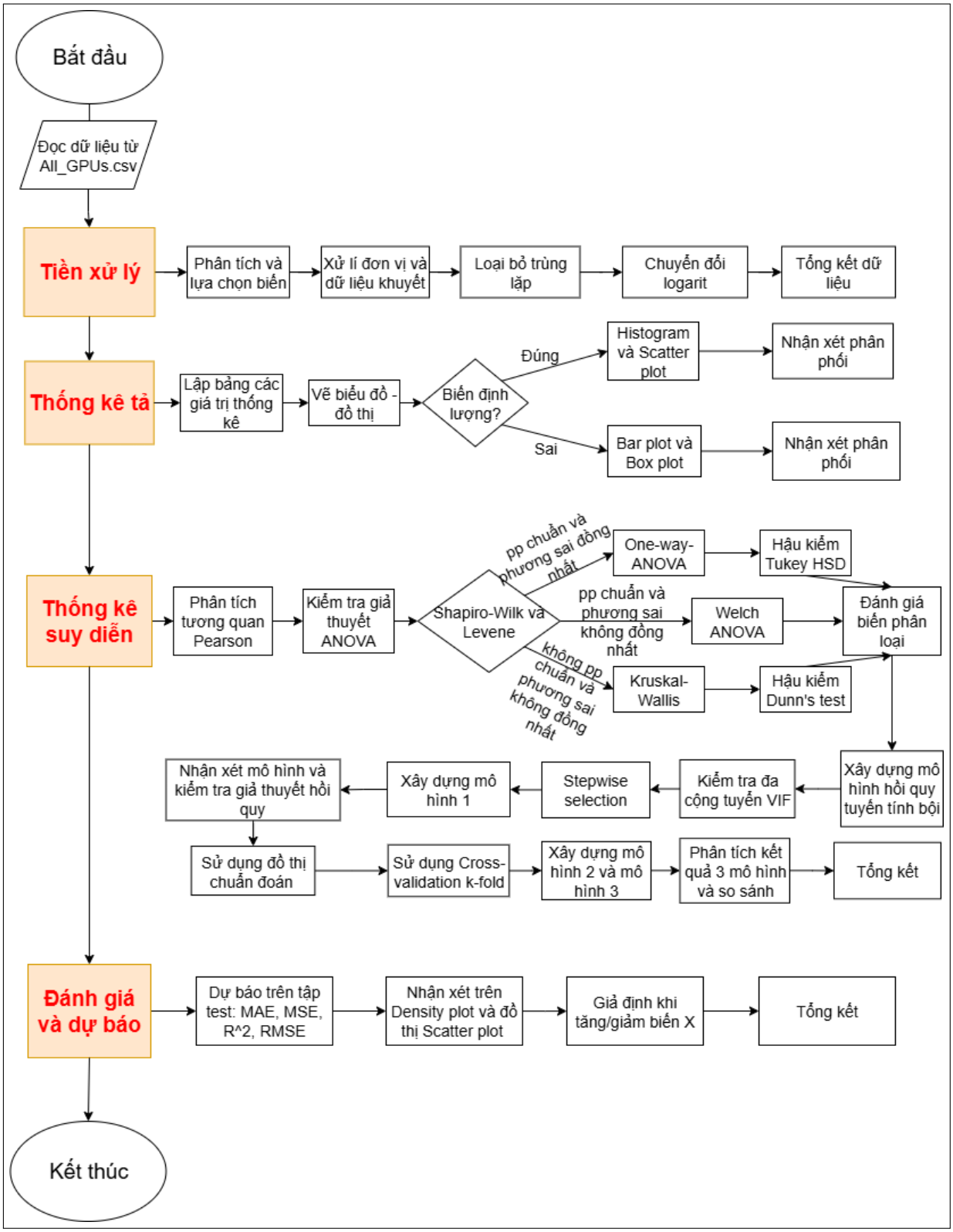
Tổng quan: VIF (Variance Inflation Factor) là một chỉ số thống kê đo lường mức độ tương quan giữa các biến độc lập trong mô hình hồi quy. Đa cộng tuyến xảy ra khi các biến độc lập có mối quan hệ tuyến tính mạnh với nhau, dẫn đến sai lệch trong các hệ số hồi quy, làm giảm độ tin cậy của mô hình. VIF giúp phát hiện và xử lý vấn đề này bằng cách định lượng mức độ ảnh hưởng của sự tương quan giữa các biến.

Mục đích: Áp dụng VIF để kiểm tra và loại bỏ đa cộng tuyến giữa các biến độc lập như Core_Speed, Memory_Speed, ROPs, TMUs, Memory_Bus, Memory, Process, Shader, và các biến khác trong mô hình hồi quy tuyến tính đa biến. Công cụ này đảm bảo rằng các biến độc lập không có mối quan hệ tuyến tính mạnh với nhau, từ đó giúp các hệ số hồi quy phản ánh chính xác mức độ ảnh hưởng của từng biến đến Pixel_Rate, nâng cao độ tin cậy và chất lượng của mô hình dự đoán.

Lý do lựa chọn:

- ***Đảm bảo độ chính xác tối ưu của mô hình:*** Đa cộng tuyến có thể làm sai lệch các hệ số hồi quy, dẫn đến việc đánh giá sai mức độ ảnh hưởng của các biến như Core_Speed hoặc ROPs đến Pixel_Rate. VIF cung cấp một cách tiếp cận khoa học để phát hiện và loại bỏ vấn đề này, đảm bảo rằng mỗi biến được đánh giá độc lập và chính xác, từ đó mang lại kết quả dự đoán đáng tin cậy hơn.
- ***Hiệu quả và dễ triển khai:*** VIF là một chỉ số dễ tính toán, cung cấp kết quả rõ ràng và trực quan (thông thường, VIF $>$ 5 hoặc 10 cho thấy đa cộng tuyến nghiêm trọng). Điều này cho phép các nhà nghiên cứu nhanh chóng xác định các biến có vấn đề và điều chỉnh mô hình mà không cần các phương pháp phức tạp, tiết kiệm thời gian và nguồn lực.
- ***Tầm quan trọng trong mô hình đa biến:*** Với tập dữ liệu chứa 11 biến độc lập, bao gồm các thông số kỹ thuật của GPU có khả năng tương quan cao (ví dụ, Core_Speed và Memory_Speed), kiểm tra đa cộng tuyến bằng VIF là một bước không thể thiếu. Điều này đảm bảo rằng mô hình hồi quy tuyến tính đa biến không bị ảnh hưởng bởi các mối quan hệ không mong muốn giữa các biến, từ đó duy trì tính ổn định và khả năng diễn giải của mô hình.

2.3 Lưu đồ thuật toán



Hình 2.1: Lưu đồ các bước thực hiện phân tích

Chương 3

TIỀN XỬ LÝ SỐ LIỆU

3.1 Đọc dữ liệu

Trước tiên, ta đọc dữ liệu thu từ file "All_GPUs.csv"và sau đó in ra 3 dòng đầu tiên tương ứng với dữ liệu của mỗi biến. Việc này giúp chúng ta có cái nhìn tổng quan về dữ liệu, hiểu cấu trúc, định hướng trước khi đi sâu vào phân tích. Đồng thời, các giá trị như chuỗi rỗng,\n- ", "\n", hoặc "\nUnknown Release Date"trong file CSV được chuyển thành NA trong data.frame, qua đó giúp chuẩn hóa dữ liệu, tránh các giá trị không hợp lệ và hỗ trợ việc phân tích dữ liệu khuyết sau này.

```
GPU = read.csv("D:/DHBK/HK243/XSTK/BTL/Code_R/All_GPUs.csv",header=TRUE,na.strings =
 ↪  c("","\n-","\n","\nUnknown Release Date"))
head(GPU,3), dim(GPU)
```

Kết quả:

```
> head(GPU,3)
  Architecture Best_Resolution Boost_Clock Core_Speed DVI_Connection Dedicated
1  Tesla G92b             <NA>        <NA>    738 MHz             2       Yes
2     R600 XT       1366 x 768        <NA>       <NA>             2       Yes
3    R600 PRO       1366 x 768        <NA>       <NA>             2       Yes
  Direct_X DisplayPort_Connection HDMI_Connection Integrated L2_Cache
1  DX 10.0                    NA               0         No      0KB
2    DX 10                    NA               0         No      0KB
3    DX 10                    NA               0         No      0KB
  Manufacturer Max_Power   Memory Memory_Bandwidth Memory_Bus Memory_Speed
1       Nvidia 141 Watts 1024 MB          64GB/sec   256 Bit      1000 MHz
2          AMD 215 Watts  512 MB         106GB/sec   512 Bit       828 MHz
3          AMD 200 Watts  512 MB        51.2GB/sec   256 Bit       800 MHz
  Memory_Type                  Name Notebook_GPU Open_GL                PSU
1       GDDR3         GeForce GTS 150          No     3.3 450 Watt & 38 Amps
2       GDDR3 Radeon HD 2900 XT 512MB          No     3.1 550 Watt & 35 Amps
3       GDDR3      Radeon HD 2900 Pro          No     3.1 550 Watt & 35 Amps
   Pixel_Rate Power_Connector Process ROPs   Release_Date Release_Price
1 12 GPixel/s           None     55nm   16 \n01-Mar-2009           <NA>
2 12 GPixel/s           None     80nm   16 \n14-May-2007           <NA>
3 10 GPixel/s           None     80nm   16 \n07-Dec-2007           <NA>
  Resolution_WxH SLI_Crossfire Shader TMUs Texture_Rate VGA_Connection
1      2560x1600           Yes      4   64  47 GTexel/s              0
2      2560x1600           Yes      4   16  12 GTexel/s              0
3      2560x1600           Yes      4   16  10 GTexel/s              0
> dim(GPU)
[1] 3406   34
```

Hình 3.1: Các biến có trong dữ liệu ban đầu

Nhận xét: Kết quả cho thấy, bộ dữ liệu bao gồm 3406 quan sát và 34 biến, tuy nhiên lại có rất nhiều biến chứa dữ liệu khuyết dù chỉ mới xuất ra màn hình 3 dòng đầu. Chính vì thế, việc phân tích dữ liệu khuyết và chọn lọc các biến liên tục và biến phân loại phải được thực hiện một cách hợp lý, tạo tiền đề cho các phân tích chuyên sâu.

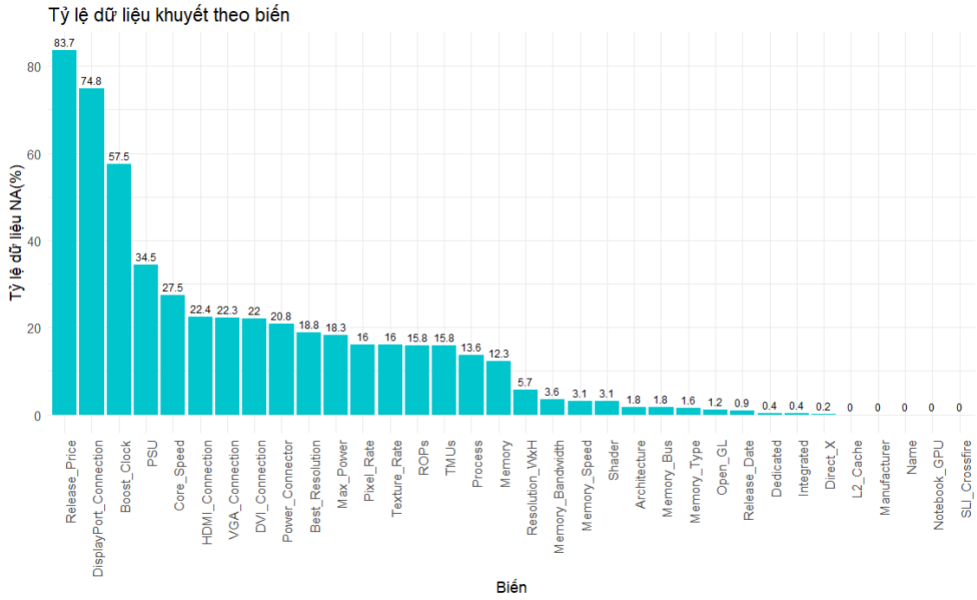
3.2 Chọn biến và xử lí dữ liệu

a) Cơ sở lựa chọn biến

Ta thực hiện in bảng thống kê số lượng và tỷ lệ dữ liệu khuyết trong tệp tin bằng các câu lệnh:

```
ggplot(data = na_summary_df, aes(x = reorder(Variable, -Percentage), y =
 ↪  Percentage)) +
  geom_bar(stat = "identity", fill = "turquoise3") +
  geom_text(aes(label = Percentage), vjust = -0.5, size = 2.5) +
  labs(
    title = "Phần trăm dữ liệu khuyết theo biến",
    x = "Biến",
    y = "Phần trăm dữ liệu NA"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
plot1 | plot2
ggsave("na_combined_filtered_plot.png", width = 5, height = 12)
```

Ta thu được kết quả như hình 3.2:



Hình 3.2: Phân tích dữ liệu khuyết của từng biến trong bộ dữ liệu

Nhận xét: Ta nhận thấy tỷ lệ dữ liệu khuyết ở các biến có sự khác biệt và chênh lệch nhau lớn. Do đó, đối với những biến có tỷ lệ dữ liệu khuyết cao, ta sẽ không lựa chọn mà chỉ giữ lại các biến có dữ liệu khuyết tương đối.

Qua các cơ sở trên, nhằm tối ưu hóa kết quả khi phân tích hiệu năng GPU máy tính, nhóm quyết định lựa chọn các biến sau: **Resolution_WxH**, **Manufacturer**, **Core_Speed**, **Memory_Speed**, **Memory_Type**, **ROPs**, **TMUs**, **Memory_Bus**, **Memory**, **Process**, **Shader**, **Pixel_Rate** (trong đó gồm 3 biến phân loại là **Resolution_WxH**, **Manufacturer**, **Memory_Type** và 9 biến liên tục) vì các biến này cung cấp cái nhìn toàn diện về hiệu năng GPU, từ sức mạnh xử lý, khả năng kết xuất, đến hiệu quả công nghệ, giúp nhóm đánh giá và so sánh các GPU một cách chi tiết.

Câu lệnh thực hiện:

```
GPU_new = GPU[,c("Resolution_WxH", "Manufacturer", "Core_Speed",
                 "Memory_Speed", "Memory_Type",
                 "ROPs", "TMUs", "Memory_Bus", "Memory",
                 "Process", "Shader", "Pixel_Rate")]
```

b) Xử lí dữ liệu từ các biến đã chọn

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

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

Kết quả:

```
Resolution_WxH     Manufacturer        Core_Speed        Memory_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_Type            ROPs              TMUs          Memory_Bus
Length:3406        Length:3406        Min.   :  1.00   Length:3406
Class :character   Class :character   1st Qu.: 24.00   Class :character
Mode  :character   Mode  :character   Median : 56.00   Mode  :character
                                      Mean   : 69.38
                                      3rd Qu.:104.00
                                      Max.   :384.00
                                      NA's   :538
   Memory            Process             Shader       Pixel_Rate
Length:3406        Length:3406        Min.   :1.000   Length:3406
Class :character   Class :character   1st Qu.:5.000   Class :character
Mode  :character   Mode  :character   Median :5.000   Mode  :character
                                      Mean   :4.706
                                      3rd Qu.:5.000
                                      Max.   :5.000
                                      NA's   :107
```

Hình 3.3: Dữ liệu từ dataframe GPU_new

Nhận xét:

- **Đối với các biến định lượng:**

Hầu hết đều ở dạng character (dạng ký tự). Chính vì thế, để có thể phân tích bằng R, ta sẽ chuyển các dữ liệu cần thiết sang số thực. Ta định nghĩa một hàm để tách các đơn vị với chữ số bằng khoảng trắng và chuyển về dạng số thực:

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

Sau đó lần lượt chuyển đổi thành dữ liệu số thực, đồng thời, đối với các dữ liệu khuyết đã chuyển thành NA ban đầu của từng biến, ta thay bằng trung vị của tất cả các dữ liệu trong biến tương ứng:

```
# Core_Speed
GPU_new$Core_Speed <- sapply(GPU_new$Core_Speed, helper)
GPU_new$Core_Speed[is.na(GPU_new$Core_Speed)] <- median(GPU_new$Core_Speed, na.rm =
 ↪  TRUE)
# ROPs
# Giả sử GPU_new$ROPs đang là character, chứa các giá trị như "24 (x4)", "48 (x2)",
 ↪  v.v. Ta chỉ lấy phần trước dấu ngoặc
GPU_new$ROPs <- as.numeric(sub("^([0-9]+).*", "\\1", GPU_new$ROPs))
GPU_new$ROPs[is.na(GPU_new$ROPs)] <- median(GPU_new$ROPs, na.rm = TRUE)
#Tương tự với các biến còn lại
```

- **Đối với các biến định tính:** Ta thay các giá trị khuyết NA thành mode của biến tương ứng. Đồng thời thực hiện chuyển đổi thành factor vì đây là các biến phân nhóm.

  Ví dụ Resolution_WxH: 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$Resolution_WxH[is.na(GPU_new$Resolution_WxH)] = "4096x2160"
GPU_new$Resolution_WxH <- ifelse(GPU_new$Resolution_WxH == "4096x2160", 1,
 ↪  ifelse(GPU_new$Resolution_WxH == "2560x1600", 2, 3))
GPU_new$Resolution_WxH = factor(GPU_new$Resolution_WxH)
```

3.3 Loại bỏ trùng lặp

Loại bỏ trùng lặp là quá trình loại bỏ các bản ghi hoặc giá trị lặp lại trong một tập dữ liệu để đảm bảo rằng mỗi hàng hoặc giá trị chỉ xuất hiện một lần. Việc loại bỏ trùng lặp giúp giữ lại chỉ một phiên bản duy nhất. Ngoài ra còn cải thiện độ chính xác bởi trùng lặp có thể làm sai lệch các thống kê hoặc làm tăng sai số trong mô hình. Hơn thế, còn có thể giảm kích thước tập dữ liệu, giúp xử lý nhanh hơn.

Ta sẽ so sánh bộ dữ liệu trước và sau khi đã loại bỏ trùng lặp bằng câu lệnh:

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

Kết quả:

```
> str(GPU_new)
'data.frame':   3350 obs. of  12 variables:
 $ 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_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 ...
 $ ROPs          : num  16 16 16 4 4 4 16 12 32 24 ...
 $ TMUs          : num  64 16 16 8 8 8 40 12 80 56 ...
 $ Memory_Bus    : num  256 512 256 128 128 128 256 256 256 64 ...
 $ Memory        : num  1024 512 512 256 256 ...
 $ Process       : num  55 80 80 65 65 65 55 80 28 28 ...
 $ Shader        : num  4 4 4 4 4 4 4.1 4 5 1 ...
 $ Pixel_Rate    : num  12 12 10 3 3 3 14 7 25 26 ...
> GPU_new <- dplyr::distinct(GPU_new)
>
> # Kiểm tra lại
> str(GPU_new)
'data.frame':   2204 obs. of  12 variables:
 $ 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_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 ...
 $ ROPs          : num  16 16 16 4 4 4 16 12 32 24 ...
 $ TMUs          : num  64 16 16 8 8 8 40 12 80 56 ...
 $ Memory_Bus    : num  256 512 256 128 128 128 256 256 256 64 ...
 $ Memory        : num  1024 512 512 256 256 ...
 $ Process       : num  55 80 80 65 65 65 55 80 28 28 ...
 $ Shader        : num  4 4 4 4 4 4 4.1 4 5 1 ...
 $ Pixel_Rate    : num  12 12 10 3 3 3 14 7 25 26 ...
```

Nhận xét: Nhận thấy rằng trước khi loại bỏ trùng lặp, tập dữ liệu có tới 3350 quan sát, nhưng sau khi xử lý, số lượng giảm còn 2204. Điều này cho phép cải thiện đáng kể độ chính xác và tối ưu hóa dữ liệu, tạo nền tảng vững chắc để tiến hành các bước phân tích chuyên sâu tiếp theo.

3.4 Chuyển đổi logarit

Chuyển đổi logarit là một bước tiền xử lý dữ liệu quan trọng, giúp tối ưu hóa dữ liệu và đáp ứng các giả định thống kê cần thiết. Nếu các biến có phân phối lệch phải, phương sai không đồng nhất, và mối quan hệ phi tuyến với nhau sẽ gây khó khăn cho mô hình

hồi quy. Chuyển đổi logarit được áp dụng nhằm khắc phục những vấn đề này, mang lại một mô hình hiệu quả hơn, dễ diễn giải hơn, và phù hợp với mục tiêu nghiên cứu.

***Giải thích lý do chuyển sang dạng log**

- **Tuyến tính hóa mối quan hệ phi tuyến:** Mối quan hệ giữa Pixel_Rate và các biến độc lập như Core_Speed, Memory_Speed thường không tuyến tính. Chuyển đổi logarit giúp biến đổi mối quan hệ này thành dạng tuyến tính, phù hợp với giả định của mô hình hồi quy, từ đó nâng cao độ chính xác dự đoán.
- **Ổn định biến có biên độ lớn:** Các biến như Memory hoặc Pixel_Rate có biên độ giá trị lớn, gây ra hiệu ứng không đồng đều. Chuyển đổi logarit nén các giá trị lớn, giảm tác động của outliers và tăng tính ổn định cho mô hình.
- **Cải thiện phân phối chuẩn:** Pixel_Rate và một số biến độc lập có phân phối lệch phải, không đáp ứng giả định phân phối chuẩn của sai số. Chuyển đổi logarit giúp phân phối của các biến này gần chuẩn hơn, đảm bảo tính hợp lệ của mô hình.

Việc áp dụng chuyển đổi logarit được thực hiện cẩn trọng, dựa trên phân tích khám phá dữ liệu (EDA), để đảm bảo mô hình phản ánh chính xác mối quan hệ giữa các biến và đáp ứng mục tiêu dự đoán hiệu suất GPU.

Ta thực hiện đoạn code sau để biến đổi thành dataframe mới tên là GPU_new_log.

```
# Tạo dataframe log-transformed
GPU_new_log <- GPU_new
GPU_new_log[, numerical] <- log(GPU_new_log[, numerical])
GPU_new_log$Pixel_Rate <- log(GPU_new_log$Pixel_Rate)
```

Kiểm tra lại dữ liệu âm trong dữ liệu gốc và trong dữ liệu sau khi chuyển đổi log-transform:

```
for (var in numerical) {
    has_negative <- any(GPU_new[[var]] < 0, na.rm = TRUE)
    message(var, " in original < 0? ", has_negative)
}
for (var in numerical) {
 has_negative_log <- any(GPU_new_log[[var]] < 0, na.rm = TRUE)
 message(var, " in log-transformed < 0? ", has_negative_log)
}
```

```
Core_Speed in original < 0? FALSE
Memory_Speed in original < 0? FALSE
ROPs in original < 0? FALSE
TMUs in original < 0? FALSE
Memory_Bus in original < 0? FALSE
Memory in original < 0? FALSE
Process in original < 0? FALSE
Shader in original < 0? FALSE
```

```
Core_Speed in log-transformed < 0? FALSE
Memory_Speed in log-transformed < 0? FALSE
ROPs in log-transformed < 0? FALSE
TMUs in log-transformed < 0? FALSE
Memory_Bus in log-transformed < 0? FALSE
Memory in log-transformed < 0? FALSE
Process in log-transformed < 0? FALSE
Shader in log-transformed < 0? FALSE
```

3.5 Tổng kết

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.

```r
# Xem 10 dòng đầu của GPU_new_log
head(GPU_new_log, 10)
```

	Resolution_WxH	Manufacturer	Core_Speed	Memory_Speed	Memory_Type	ROPs
1	2	Nvidia	6.603944	6.907755	GDDR	2.772589
2	2	AMD	6.887553	6.719013	GDDR	2.772589
3	2	AMD	6.887553	6.684612	GDDR	2.772589
4	2	AMD	6.887553	7.047517	GDDR	1.386294
5	2	AMD	6.887553	6.551080	GDDR	1.386294
6	2	AMD	6.887553	7.003065	GDDR	1.386294
7	2	AMD	6.768493	6.956545	GDDR	2.772589
8	2	AMD	6.887553	6.684612	GDDR	2.484907
9	1	AMD	6.887553	7.130899	GDDR	3.465736
10	3	AMD	6.887553	5.902633	DDR	3.178054

	TMUs	Memory_Bus	Memory	Process	Shader	Pixel_Rate
1	4.158883	5.545177	6.931472	4.007333	1.386294	2.484907
2	2.772589	6.238325	6.238325	4.382027	1.386294	2.484907
3	2.772589	5.545177	6.238325	4.382027	1.386294	2.302585
4	2.079442	4.852030	5.545177	4.174387	1.386294	1.098612
5	2.079442	4.852030	5.545177	4.174387	1.386294	1.098612
6	2.079442	4.852030	5.545177	4.174387	1.386294	1.098612
7	3.688879	5.545177	7.624619	4.007333	1.410987	2.639057
8	2.484907	5.545177	5.545177	4.382027	1.386294	1.945910
9	4.382027	5.545177	7.624619	3.332205	1.609438	3.218876
10	4.025352	4.158883	4.158883	3.332205	0.000000	3.258097

Chương 4

THỐNG KÊ MÔ TẢ

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

Sau khi làm sạch dữ liệu, ta thực hiện thống kê mô tả cho các biến đã chọn

```
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
Core_Speed	937.062365	224.5574745	100	1784	980
Memory_Speed	1133.216111	441.1421972	100	2127	1100
ROPs	25.289736	18.0674066	1	128	24
TMUs	65.517540	50.3416169	1	384	56
Memory_Bus	209.981810	287.6959834	32	8192	128
Memory	2698.106540	2712.4044826	16	32000	2048
Process	32.790819	15.6819279	14	150	28
Shader	4.683456	0.7204695	1	5	5

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

	mean	sd	min	max	median
Core_Speed	6.807736	0.2857980	4.605170	7.486613	6.887553
Memory_Speed	6.935780	0.4826242	4.605170	7.662468	7.003065
ROPs	2.935485	0.8569198	0.000000	4.852030	3.178054
TMUs	3.839054	0.9368217	0.000000	5.950643	4.025352
Memory_Bus	5.129136	0.5992609	3.465736	9.010913	4.852030
Memory	7.446108	1.0667592	2.772589	10.373491	7.624619
Process	3.409616	0.3850134	2.639057	5.010635	3.332205
Shader	1.526059	0.2130054	0.000000	1.609438	1.609438

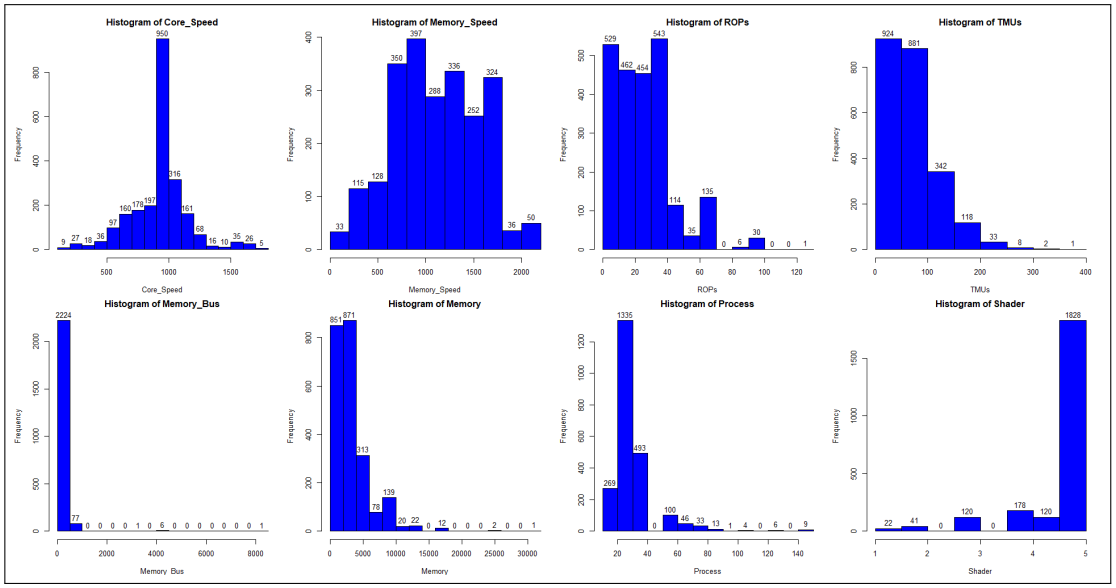
Ta thấy nếu biểu diễn dưới dạng bảng như này thì khó có thể hình dung dữ liệu như thế nào hay có liên hệ với nhau ra sao. Do đó ta sẽ biểu diễn các dữ liệu này dưới dạng các biểu đồ để dễ quan sát hơn.

4.2 Biểu đồ - đồ thị

4.2.1 Histogram

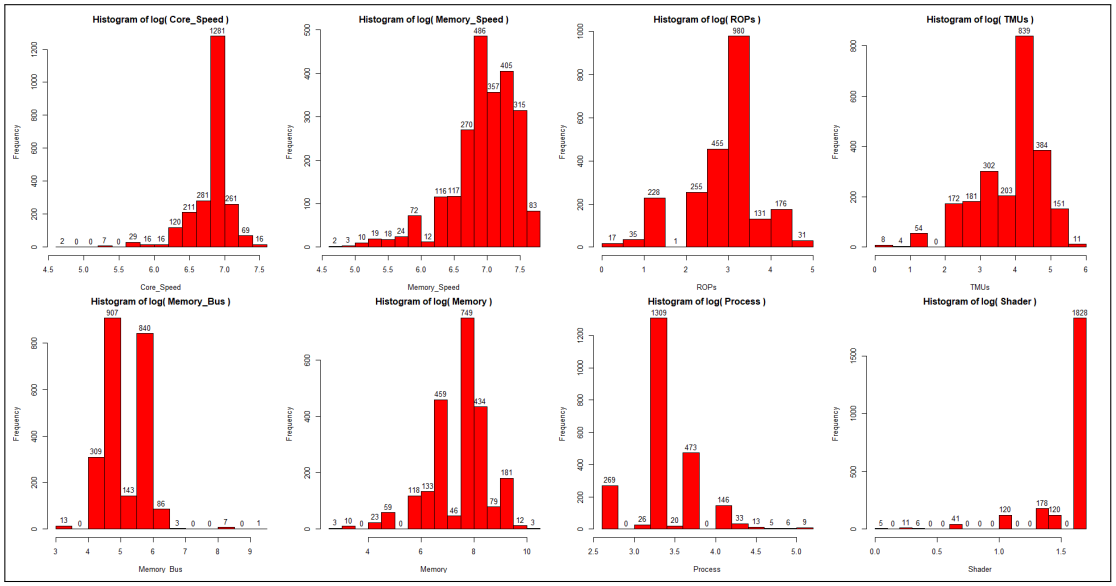
Biểu đồ tần suất (histogram) thường dùng để thể hiện phân phối của biến định lượng. Ta vẽ biểu đồ tần suất cho các dữ liệu GPU_new và GPU_new_log. Ta tiến hành vẽ biểu đồ tần suất cho dữ liệu GPU_new.

```
# Vẽ histogram cho từng biến numerical trong 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")
}
```



Nhận xét: Ta thấy các biến Process, Memory, TMUs và ROPs 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 và Shader thì gần như chỉ có một cột, các cột còn lại không đáng kể.

Ta tiếp tục tiến hành vẽ biểu đồ tần suất cho dữ liệu của GPU_new_log:

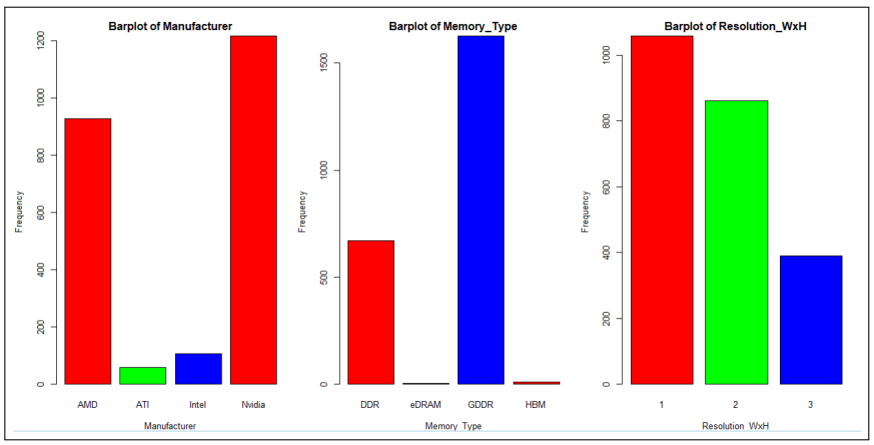


Nhận xét: Ta thấy sau khi chuyển đổi các biến x sang dạng log(x) thì phân phối các biến trở nên gần giống với phân phối chuẩn hơn, ví dụ như biến Memory_Bus hay Memory_Speed. Các biến Core_Speed, Memory, TMUs và ROPs cũng không còn phân phối lệch như trước.

4.2.2 Barplot

Biểu đồ cột (Barplot) thường được dùng để thể hiện phân phối giữa các biến định tính. Ta tiến hành vẽ biểu đồ cột như sau:

```
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"))
```



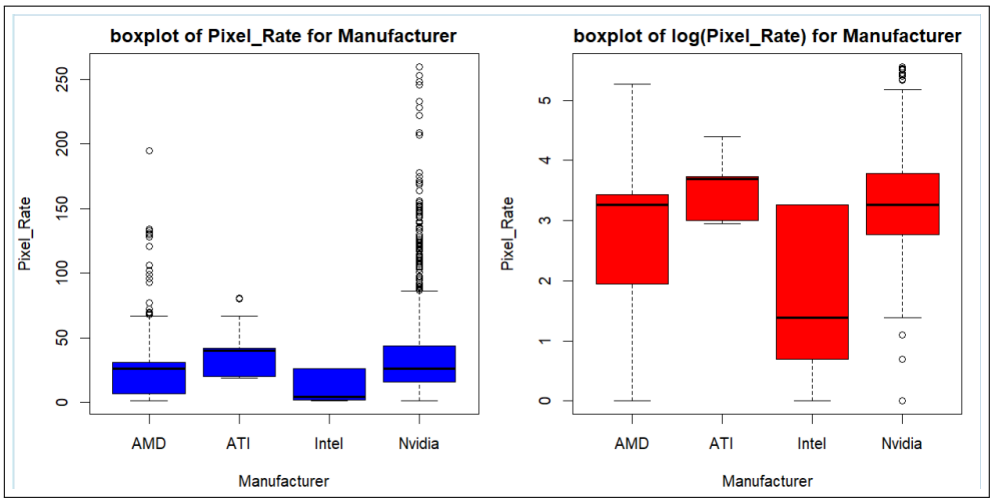
Nhận xét: Ta thấy hầu hết GPU đều được sản xuất bởi AMD và Nvidia, 2 nhà sản xuất còn lại (ATI và Intel) không đáng kể. Chỉ có 2 kiểu bộ nhớ là DDR và GDDR, và đa số thuộc về GDDR. Về chỉ số Resolution_WxH thì ta thấy không có sự chênh lệch đáng kể giữa 3 cột như 2 biến kia.

4.2.3 Boxplot

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 Pixel_Rate và log(Pixel_Rate) theo từng biến phân loại:

- **Manufacturer**

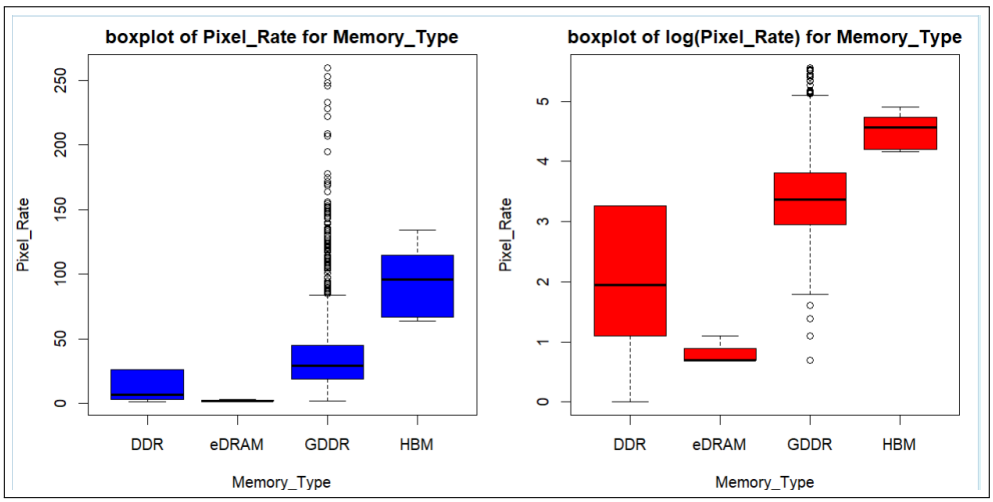
```
boxplot(Pixel_Rate~Manufacturer, data=GPU_new, main="boxplot of Pixel_Rate for
↪ Manufacturer", col="blue")
boxplot(Pixel_Rate~Manufacturer, data=GPU_new_log, main="boxplot of log(Pixel_Rate)
↪ for Manufacturer", col="red")
```



Nhận xét: Ta thấy đối với nhà sản xuất là Intel và Nvidia, hiệu suất dựng ảnh có hơi lệch phải. Nhưng sau khi lấy log(Pixel_Rate) thì cả 4 phân phối đều trở thành phân phối chuẩn.

- **Memory_Type**

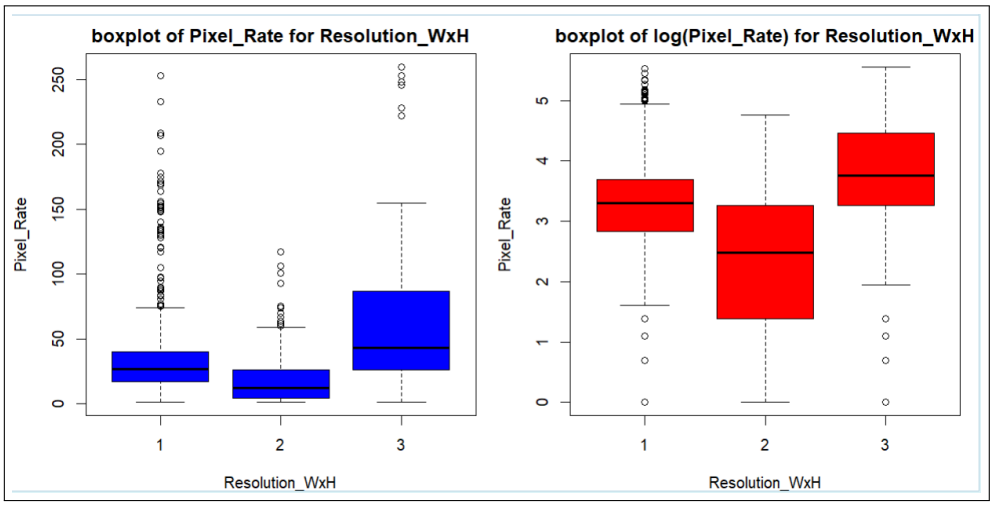
```
boxplot(Pixel_Rate~Memory_Type, data=GPU_new, main="boxplot of Pixel_Rate for
↪ Memory_Type", col="blue")
boxplot(Pixel_Rate~Memory_Type, data=GPU_new_log, main="boxplot of log(Pixel_Rate)
↪ for Memory_Type", col="red")
```



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

- **Resolution_WxH**

```
boxplot(Pixel_Rate~Resolution_WxH, data=GPU_new, main="boxplot of Pixel_Rate for
↪  Resolution_WxH", col="blue")
boxplot(Pixel_Rate~Resolution_WxH, data=GPU_new_log, main="boxplot of log(Pixel_Rate)
↪  for Resolution_WxH", col="red")
```

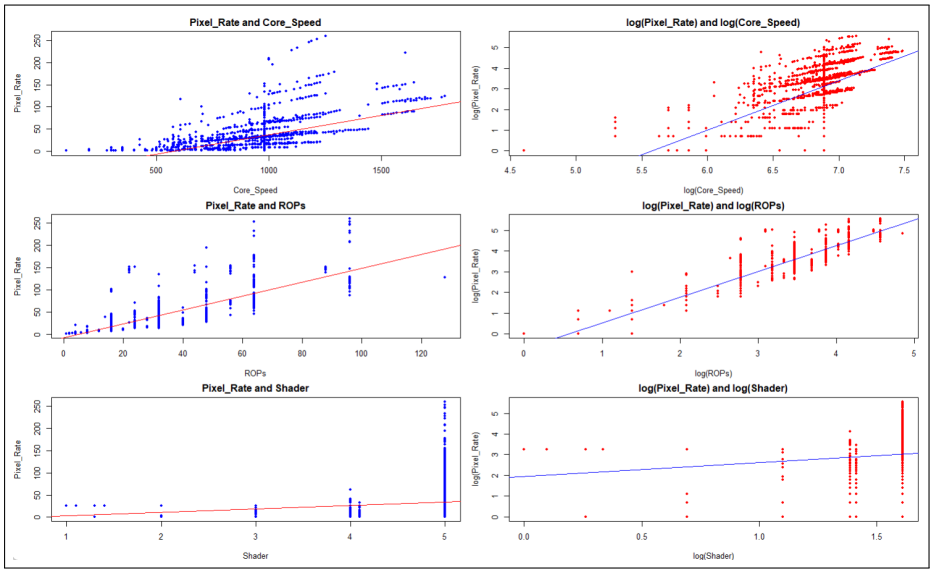


Nhận xét: Hiệu suất của các nhóm trước khi biến đổi log(x) đều lệch phải (đặc biệt là nhóm 2 và 3). Tuy nhiên cả ba đều trở thành xấp xỉ phân phối chuẩn sau khi biến đổi.

4.2.4 Scatter plot

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 Pixel_Rate với một số biến định lượng khác.

```r
# Vẽ scatter plot
# Core_Speed vs Pixel_Rate
plot(GPU_new$Core_Speed, GPU_new$Pixel_Rate,
     xlab = "Core_Speed", ylab = "Pixel_Rate",
     main = "Pixel_Rate and Core_Speed", col = "blue", pch = 20)
fit1 <- lm(Pixel_Rate ~ Core_Speed, data = GPU_new)
abline(fit1, col = "red")
# log(Core_Speed) vs log(Pixel_Rate)
plot(GPU_new_log$Core_Speed, GPU_new_log$Pixel_Rate,
     xlab = "log(Core_Speed)", ylab = "log(Pixel_Rate)",
     main = "log(Pixel_Rate) and log(Core_Speed)", col = "red", pch = 20)
fit2 <- lm(Pixel_Rate ~ Core_Speed, data = GPU_new_log)
abline(fit2, col = "blue")
#Tương tự với ROPs và Shader
```



Nhận xét:

- Ta thấy rằng giữa Pixel_Rate và Core_Speed tồn tại mối tương quan dương: khi tốc độ lõi tăng thì hiệu suất dựng ảnh cũng có xu hướng tăng, tuy nhiên, các điểm dữ liệu phân tán khá rộng. Sau khi biến đổi log, các điểm dữ liệu tập trung hơn quanh đường hồi quy và quan hệ trở nên gần tuyến tính hơn.
- Đối với ROPs, mối tương quan dương thể hiện rõ rệt ngay từ dữ liệu gốc, với các cụm dữ liệu xuất hiện do đặc tính rời rạc của phần cứng. Việc biến đổi log giúp quan hệ này trở nên tuyến tính hơn và giảm hiện tượng kéo giãn dữ liệu.
- Trong khi đó, Shader cho thấy mối tương quan dương yếu với Pixel_Rate ở cả hai dạng dữ liệu; số lượng Shader tăng không kéo theo sự tăng đáng kể về Pixel_Rate, cho thấy biến này có thể không phải là yếu tố quyết định chính.

Chương 5

THỐNG KÊ SUY DIỄN

5.1 Đánh giá mối quan hệ giữa các biến

5.1.1 Ma trận tương quan Pearson

Ở đây ta sử dụng hệ số tương quan, là một chỉ số thống kê đo lường mối liên hệ tương quan giữa Pixel_Rate và các biến. 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. Nếu giá trị của hệ số tương quan là dương, có nghĩa là x và y đồng biến và âm thì nghịch biến.

Hệ số tương quan Pearson:

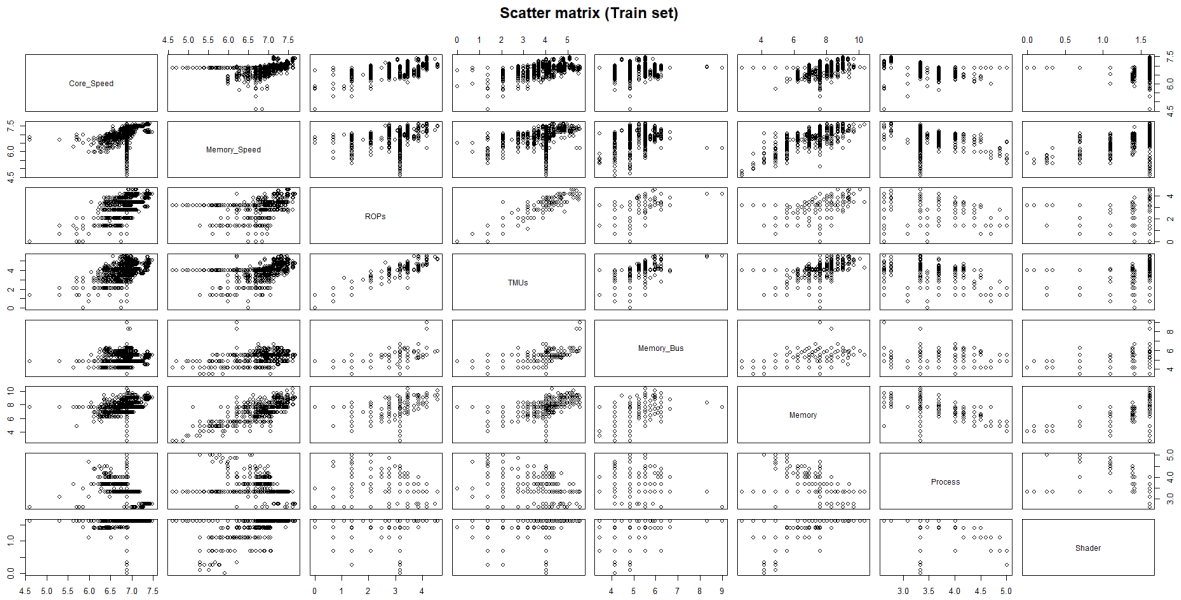
$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

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

	Variable	Pearson_r	P_Value
1	Core_Speed	0.5920649	2.442821e-146
2	Memory_Speed	0.4528487	1.007937e-78
3	ROPs	0.9430541	0.000000e+00
4	TMUs	0.8989215	0.000000e+00
5	Memory_Bus	0.5692571	5.688701e-133
6	Memory	0.4260982	5.816298e-69
7	Process	-0.4157636	2.026402e-65
8	Shader	0.1290108	3.776748e-07

Nhận xét: TMUs và ROPs có tương quan dương rất cao với (Pearson_r = 0.898 và 0.943), cho thấy khi hai thông số này tăng thì biến mục tiêu cũng tăng đáng kể. Tiếp theo, Core_Speed và Memory_Bus lần lượt có hệ số tương quan 0.592 và 0.569, thể hiện mối liên hệ dương ở mức trung bình. Memory_Speed và Memory có tương quan dương yếu hơn (0.453 và 0.426). Biến Process lại có tương quan âm ở mức (-0.416). Cuối cùng, Shader chỉ có hệ số tương quan 0.129, tức là gần như không có mối quan hệ tuyến tính đáng kể. Tất cả các p-value đều cực kỳ nhỏ (< 0.05), chứng tỏ các mối tương quan này có ý nghĩa thống kê rất cao và không phải do yếu tố ngẫu nhiên gây ra.

5.1.2 Scatter plot và Scatter matrix



Hình 5.1: Biểu đồ tương quan với hệ số tương quan giữa từng cặp biến

5.2 Phân tích phương sai một yếu tố (ANOVA)

Nhằm đánh giá sự khác biệt về hiệu suất xử lý đồ họa (Pixel_Rate) giữa các nhóm phân loại khác nhau, Ta thực hiện phân tích phương sai (ANOVA một chiều) đối với ba biến phân loại chính Manufacturer , Memory_Type và Resolution_WxH trên tập huấn luyện (train).

5.2.1 Mục tiêu

- Kiểm định xem liệu Pixel_Rate có khác nhau giữa các nhóm phân loại (ví dụ Manufacturer) hay không, ở mức ý nghĩa $\alpha = 5$
- Xác định xem có ít nhất một cặp nhóm có trung bình Pixel_Rate khác biệt về mặt thống kê.

5.2.2 Bài toán

5.2.2.1 Các điều kiện cần kiểm tra trước khi thực hiện ANOVA

***Điều kiện 1: Độc lập của các quan sát**

Các giá trị Pixel_Rate đo được từ mỗi nhóm (Manufacturer / Memory_Type / Resolution_WxH) phải được thu thập độc lập.

→ Do dữ liệu được ghi nhận từ các dòng GPU khác nhau, không có chồng lắp hay phụ thuộc, nên điều kiện này được thoả.

***Điều kiện 2: Biến phụ thuộc là biến liên tục**

Pixel_Rate là giá trị số thực liên tục (Gpixel/s).

→ Điều kiện này được thoả.

***Điều kiện 3: Phân phối chuẩn trong mỗi nhóm**

Dùng kiểm định Shapiro–Wilk trên phần dư hoặc trực tiếp trên từng nhóm. Nếu $p > 0.05$, phần dư (hoặc dữ liệu nhóm) xấp xỉ phân phối chuẩn.

***Điều kiện 4: Đồng nhất phương sai giữa các nhóm**

Dùng Levene's test, Nếu $p > 0.05$, không bác bỏ giả thiết phương sai bằng nhau.

5.2.2.2 Giả thuyết của bài toán

H₀: Trung bình Pixel_Rate của tất cả các nhóm là bằng nhau: $H_0 : \mu_1 = \mu_2 = \cdots = \mu_k$

H₁: Có ít nhất một cặp nhóm có trung bình khác nhau. $H_1 : \exists i \neq j$ sao cho $\mu_i \neq \mu_j$

Sau khi xác nhận các điều kiện từ (1) - (4), ta tiến hành ANOVA một yếu tố để kiểm định H_0 ở mức ý nghĩa $\alpha = 0.05$. Nếu $p < 0.05$, bác bỏ H_0, kết luận rằng có sự khác biệt về hiệu suất Pixel_Rate giữa các nhóm.

5.2.3 Kiểm định giả thuyết

Trước khi thực hiện phân tích phương sai (ANOVA), ta sẽ sử dụng kiểm định Shapiro-Wilk để đánh giá phân phối chuẩn của phần dư (điều kiện 3) và Levene's test để kiểm tra đồng nhất phương sai (điều kiện 4). Dưới đây là kết quả kiểm tra cho từng biến phân loại: Manufacturer, Memory_Type, và Resolution_WxH.

Tên biến phân loại	Kiểm định Shapiro-Wilk	Kiểm định Levene
Manufacturer	**Thống kê:** $W = 0.96812$	**Thống kê:** $F(3, 1467) = 8.799$
	Giá trị: $p < 2.2 \cdot 10^{-16}$	**Giá trị:** $p = 8.781 \cdot 10^{-6}$
	Kết luận: Vì $p < 0.05$, bác bỏ giả thuyết H_0 rằng phần dư tuân theo phân phối chuẩn. Phần dư không xấp xỉ phân phối chuẩn.	**Kết luận:** Vì $p < 0.05$, bác bỏ giả thuyết H_0 rằng phương sai giữa các nhóm là bằng nhau. Phương sai không đồng nhất.
Memory_Type	**Thống kê:** $W = 0.97858$	**Thống kê:** $F(3, 1467) = 24.659$
	Giá trị: $p = 5.408 \cdot 10^{-14}$	**Giá trị:** $p = 1.445 \cdot 10^{-15}$
	Kết luận: Vì $p < 0.05$, bác bỏ giả thuyết H_0. Phần dư không tuân theo phân phối chuẩn.	**Kết luận:** Vì $p < 0.05$, bác bỏ giả thuyết H_0. Phương sai giữa các nhóm không đồng nhất.
Resolution_WxH	**Thống kê:** $W = 0.96255$	**Thống kê:** $F(2, 1468) = 20.193$
	Giá trị: $p < 2.2 \cdot 10^{-16}$	**Giá trị:** $p = 2.232 \cdot 10^{-9}$
	Kết luận: Vì $p < 0.05$, bác bỏ giả thuyết H_0. Phần dư không xấp xỉ phân phối chuẩn.	**Kết luận:** Vì $p < 0.05$, bác bỏ giả thuyết H_0. Phương sai giữa các nhóm không đồng nhất.

Bảng 5.1: Giá trị kiểm định Shapiro-Wilk và Levene của 3 biến phân loại

Dựa trên phân tích, ta có kết luận:

Kết quả từ các kiểm định Shapiro-Wilk và Levene cho thấy cả hai giả định về phân phối chuẩn và đồng nhất phương sai đều không được thỏa mãn đối với các biến Manufacturer , Memory_Type , và Resolution_WxH (giá trị $p < 0.05$ trong tất cả các trường hợp). Vì vậy ta sử dụng phương pháp kiểm định Kruskal–Wallis thay thế cho ANOVA.

5.2.4 Tiến hành kiểm định và kết quả

Ta sử dụng hàm kruskal() trong R để thực hiện phân tích phương sai một yếu tố (ANOVA) cho biến Pixel_Rate , phân loại theo các biến Manufacturer , Memory_Type , Resolution_WxH , và trên tập dữ liệu huấn luyện (train). Kết quả của mỗi phân tích được in ra bằng hàm summary()

```r
cat("→ Dùng Kruskal-Wallis\n")
kruskal <- kruskal.test(as.formula(paste("Pixel_Rate ~", cat)), data = train)
print(kruskal)
```

Kết quả kiểm định Kruskal-Wallis:

Dưới đây là kết quả kiểm định cho từng biến phân loại:

```
        Kruskal-Wallis rank sum test

data:  Pixel_Rate by Manufacturer
Kruskal-Wallis chi-squared = 96.258, df = 3, p-value < 2.2e-16
```

Hình 5.2: Manufacturer

```
        Kruskal-Wallis rank sum test

data:  Pixel_Rate by Memory_Type
Kruskal-Wallis chi-squared = 435.96, df = 3, p-value < 2.2e-16
```

Hình 5.3: Memory_Type

```
        Kruskal-Wallis rank sum test

data:  Pixel_Rate by Resolution_WxH
Kruskal-Wallis chi-squared = 343.58, df = 2, p-value < 2.2e-16
```

Hình 5.4: Resolution_WxH

Dưới đây là bảng tổng hợp các giá trị đặc trưng của mô hình ANOVA cho từng biến:

Biến	Df (giữa nhóm)	Df (trong nhóm)	Chi Sq	p-value
Manufacturer	3	n-4	96.258	<2e-16
Memory_Type	3	n-4	435.96	<2e-16
Resolution_WxH	2	n-3	343.58	<2e-16

Bảng 5.2: Bảng tổng hợp kết quả kiểm định Kruskal-Wallis

5.2.5 Kết luận

Cả ba biến đều có giá trị thống kê Chi-square rất cao: 96.258 (Manufacturer), 435.960 (Memory_Type), 343.580 (Resolution_WxH) với p-value $<$ 2e-16, nhỏ hơn rất nhiều so với mức ý nghĩa $\alpha = 0.05$. Điều này cho thấy có sự khác biệt có ý nghĩa thống kê về trung vị Pixel_Rate giữa các nhóm của từng biến, vì vậy ta bác bỏ giả thuyết H_0 cho cả ba biến. Sự khác biệt này có thể bắt nguồn từ yếu tố công nghệ, loại bộ nhớ, hoặc độ phân giải của sản phẩm, ảnh hưởng trực tiếp đến tốc độ xử lý điểm ảnh.

Do Kruskal–Wallis chỉ cho biết sự khác biệt tổng thể, để xác định nhóm nào khác biệt với nhóm nào, ta cần tiếp tục thực hiện kiểm định hậu nghiệm Dunn's test và sử dụng đồ thị trực quan hóa thứ hạng ảnh hưởng.

5.2.6 Phân tích hậu nghiệm

Để xác định cụ thể các cặp nhóm nào khác biệt đáng kể, ta tiến hành kiểm định hậu nghiệm Dunn's test sau Kruskal-Wallis. Dunn's test cung cấp giá trị Z và p-value điều chỉnh (Bonferroni) để đánh giá sự khác biệt giữa các cặp nhóm. Chúng ta sẽ đánh giá mức độ ảnh hưởng dựa trên chi-squared của Kruskal-Wallis: giá trị chi-squared càng lớn thì biến độc lập càng ảnh hưởng mạnh đến sự biến động của Pixel_Rate .

```
dunn <- dunnTest(as.formula(paste("Pixel_Rate ~", cat)), data = train, method =
 ↪  "bonferroni")
dunn_df <- as.data.frame(dunn$res)
barplot(
  height = dunn_df$Z,
  names.arg = dunn_df$Comparison,
  las = 2,
  col = ifelse(dunn_df$P.adj < 0.05, "tomato", "skyblue"),
  main = paste("Dunn Post-hoc -", cat),
  ylab = "Z value"
```

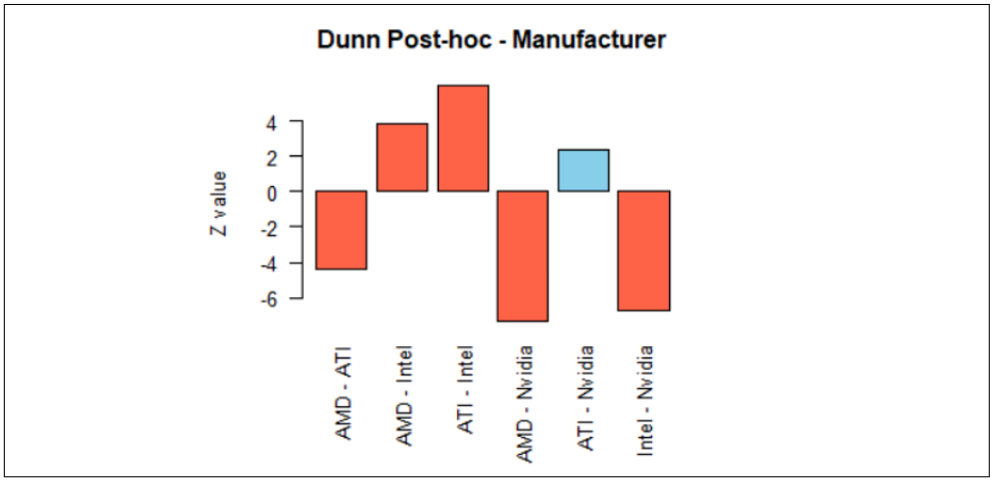
- **Biến Manufacturer**:

Kết quả của Dunn's test:

```
Dunn (1964) Kruskal-Wallis multiple comparison
  p-values adjusted with the Bonferroni method.
```

	Comparison	Z	P.unadj	P.adj
1	AMD - ATI	-4.408081	1.042908e-05	6.257447e-05
2	AMD - Intel	3.796856	1.465430e-04	8.792581e-04
3	ATI - Intel	5.977078	2.271754e-09	1.363053e-08
4	AMD - Nvidia	-7.315285	2.568362e-13	1.541017e-12
5	ATI - Nvidia	2.368169	1.787635e-02	1.072581e-01
6	Intel - Nvidia	-6.712989	1.906779e-11	1.144067e-10

Đồ thị Dunn's test:



Hình 5.5: Đồ thị Dunn Post-hoc - Manufacturer

Nhận xét: Các cặp có p.adj < 0.05 có sự khác biệt đáng kể là AMD-ATI, AMD-Intel, ATI-Intel, AMD-Nvidia, Intel-Nvidia. Riêng cặp ATI-Nvidia có p.adj = 0.107 > 0.05, nên không có sự khác biệt đáng kể.

Nhóm phân loại	Giá trị phân tích	Kết luận
AMD-ATI	Z = -4.408 < 0, p.adj < 0.05	Pixel_Rate trung vị của AMD nhỏ hơn ATI
AMD-Intel	Z = 3.797 > 0, p.adj < 0.05	Pixel_Rate trung vị của AMD lớn hơn Intel
ATI-Intel	Z = 5.977 > 0, p.adj < 0.05	Pixel_Rate trung vị của ATI lớn hơn Intel
AMD-Nvidia	Z = -7.315 < 0, p.adj < 0.05	Pixel_Rate trung vị của AMD nhỏ hơn Nvidia
ATI-Nvidia	Z = 2.368 > 0, p.adj > 0.05	Không có sự khác biệt đáng kể
Intel-Nvidia	Z = -6.713 < 0, p.adj < 0.05	Pixel_Rate trung vị của Intel nhỏ hơn Nvidia

Từ các kết quả trên, thứ tự Pixel_Rate trung vị là:

$$\boxed{ATI \approx Nvidia > AMD > Intel}$$

Nhận xét đồ thị Dunn Post-hoc - Manufacturer: Đồ thị hiển thị các thanh bar biểu diễn giá trị Z cho từng cặp so sánh. Các thanh dài (như AMD-Nvidia với Z ≈ -7.3 hoặc ATI-Intel với Z ≈ 6.0) cho thấy sự khác biệt mạnh mẽ. Màu đỏ thường dùng cho các thanh âm (chỉ sự khác biệt theo hướng nhóm đầu thấp hơn), trong khi một số thanh dương hoặc âm có màu xanh nhạt (có thể để nhấn mạnh các cặp không đáng kể như ATI-Nvidia với Z nhỏ). Các thanh cắt qua trục Z=0 (nếu có) sẽ phù hợp với p.adj > 0.05, nhưng ở đây hầu hết các thanh đáng kể đều cách xa 0, xác nhận sự khác biệt thống kê.

Thực hiện các phân tích tương tự đối với hai biến Memory_Type và Resolution_WxH ta có được kết quả:

- **Biến Memory_Type:**

$$\boxed{HBM \approx GDDR > DDR \approx eDRAM}$$

- **Biến Resolution_WxH:**

$$\boxed{\text{Nhóm } 3 > \text{Nhóm } 1 > \text{Nhóm } 2}$$

5.2.7 Kết luận

Kết quả kiểm định hậu nghiệm Dunn's test và đồ thị cho thấy:

- **Manufacturer:** Có sự khác biệt đáng kể giữa các nhà sản xuất, với ATI ≈ Nvidia có Pixel_Rate trung vị cao nhất, nhưng chi-squared = 96.258 cho thấy biến này có ảnh hưởng yếu nhất đến Pixel_Rate.
- **Memory_Type:** Có sự khác biệt lớn giữa các loại bộ nhớ, với HBM ≈ GDDR có Pixel_Rate trung vị cao nhất, và chi-squared = 435.96 cho thấy biến này có ảnh hưởng mạnh nhất đến Pixel_Rate.
- **Resolution_WxH:** Có sự khác biệt đáng kể giữa các nhóm độ phân giải, với nhóm 3 có Pixel_Rate trung vị cao nhất, và chi-squared = 343.58 cho thấy biến này có ảnh hưởng đáng kể.

Thứ tự ảnh hưởng tổng thể (dựa trên chi-squared):

$$\boxed{\text{Memory\_Type} > \text{Resolution\_WxH} > \text{Manufacturer}}$$

5.3 Xây dựng mô hình hồi quy đa biến

Vấn đề đặt ra là tìm hiểu những yếu tố ảnh hưởng đến hiệu suất dựng ảnh Pixel_Rate của GPU. Ta sẽ xây dựng mô hình hồi quy tuyến tính đa biến để giải quyết vấn đề này.

Mô hình hồi quy đa biến có dạng:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon$$

Từ các bài kiểm định trên, ta đã xác định đối với 3 yếu tố phân loại thì cả 3 đều có ảnh hưởng mạnh đến Pixel_Rate . Do đó ta sẽ thực hiện xây dựng mô hình hồi quy dự báo theo các trường hợp:

STT	Manufacturer	Memory_Type	Resolution_WxH	Biến liên tục
1	Nvidia	GDDR	Nhóm 1	Các biến liên tục đã chọn
2	AMD	GDDR	Nhóm 1	Các biến liên tục đã chọn
3	Intel	DDR	Nhóm 2	Các biến liên tục đã chọn

5.3.1 Xây dựng mô hình 1 (Nvidia - GDDR - Resolution_WxH1 - Các biến liên tục)

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, đồng thời tạo biến giả cho các biến phân loại:

```
set.seed(123)
index <- createDataPartition(GPU_new_log$Pixel_Rate, p = 2/3, list = FALSE)
train <- GPU_new_log[index, ]
test  <- GPU_new_log[-index, ]

make_dummies <- function(df) {
df$ManufacturerNvidia <- ifelse(df$Manufacturer == "Nvidia", 1, 0)
df$Memory_TypeGDDR    <- ifelse(df$Memory_Type == "GDDR", 1, 0)
df$Resolution_WxH1    <- ifelse(df$Resolution_WxH == "1", 1, 0)
return(df)
}

train <- make_dummies(train)
test  <- make_dummies(test)
```

5.3.1.1 Kiểm tra lại các biến

Trước khi xây dựng mô hình hồi quy tuyến tính đa biến để dự đoán Pixel_Rate, ta tiến hành các bước kiểm tra dữ liệu và lựa chọn biến nhằm đảm bảo mô hình đạt độ chính xác cao, tránh hiện tượng đa cộng tuyến và tối ưu hóa hiệu suất. Quá trình này bao gồm kiểm tra đa cộng tuyến bằng VIF, lựa chọn biến bằng phương pháp stepwise selection, và áp dụng feature engineering (đã làm ở trên bước xử lí dữ liệu và thống kê mô tả) để cải thiện chất lượng dữ liệu.

1. Kiểm tra đa cộng tuyến bằng VIF

Ta bắt đầu bằng cách xây dựng một mô hình hồi quy đầy đủ bao gồm tất cả các biến định lượng (Core_Speed , Memory_Speed , ROPs , TMUs , Memory_Bus , Memory , Process , Shader) và biến phân loại (Manufacturer , Memory_Type , Resolution_WxH). Hệ số VIF (Variance Inflation Factor) được tính toán để kiểm tra hiện tượng đa cộng tuyến giữa các biến, sử dụng đoạn mã sau:

```
full_form <- as.formula(
  paste("Pixel_Rate ~", paste(c(numerical, categorical), collapse = " + ")))
# Huấn luyện mô hình đầy đủ
full_mod <- lm(full_form, data = train)
print(vif(full_mod))  # Kiểm tra multicollinearity
```

Kết quả thu được sau khi ta chạy đoạn mã:

	GVIF	Df	GVIF^(1/(2*Df))
Core_Speed	1.875488	1	1.369485
Memory_Speed	3.738367	1	1.933486
ROPs	8.793333	1	2.965356
TMUs	8.409141	1	2.899852
Memory_Bus	3.009245	1	1.734718
Memory	3.170241	1	1.780517
Process	2.343707	1	1.530917
Shader	2.063734	1	1.436570
Manufacturer	1.746892	3	1.097432
Memory_Type	3.212004	3	1.214682
Resolution_WxH	2.393658	2	1.243843

Các biến có VIF < 5 thường được coi là không có đa cộng tuyến nghiêm trọng. Tuy nhiên, hai biến ROPs (VIF = 8.79) và TMUs (VIF = 8.41) có giá trị VIF cao, cho thấy tồn tại đa cộng tuyến giữa các biến này. Điều này có thể ảnh hưởng đến độ tin cậy của các hệ số hồi quy, do đó cần xem xét thêm trong quá trình lựa chọn biến.

2. Lựa chọn biến bằng stepwise selection

Để tối ưu hóa mô hình, ta sử dụng phương pháp stepwise selection (hướng "both") nhằm tự động lựa chọn các biến có ý nghĩa thống kê và loại bỏ các biến không cần thiết. Phương pháp này cải thiện độ chính xác và đơn giản hóa mô hình. Đoạn mã được sử dụng:

```
step_mod <- step(full_mod, direction = "both", trace = 0)
print(summary(step_mod))
```

Kết quả mô hình sau khi áp dụng stepwise selection:

```
Start:  AIC=-3811.37
Pixel_Rate ~ Core_Speed + Memory_Speed + ROPs + TMUs + Memory_Bus +
    Memory + Process + Shader + ManufacturerNvidia + Memory_TypeGDDR +
    Resolution_WxH1

                     Df Sum of Sq    RSS     AIC
<none>                             108.46 -3811.4
- TMUs                1     1.141 109.60 -3798.0
- Memory_Bus          1     1.271 109.73 -3796.2
- Memory_Speed        1     1.520 109.98 -3792.9
- Shader              1     1.559 110.02 -3792.4
- ManufacturerNvidia  1     1.800 110.26 -3789.2
- Resolution_WxH1     1     2.029 110.49 -3786.1
- Memory_TypeGDDR     1     2.338 110.80 -3782.0
- Core_Speed          1     6.432 114.89 -3728.6
- Memory              1     8.964 117.42 -3696.6
- Process             1    27.534 135.99 -3480.6
- ROPs                1   133.658 242.12 -2632.1
```

Sau khi áp dụng stepwise selection, mô hình vẫn giữ lại tất cả các biến ban đầu, điều này cho thấy các biến đều có ý nghĩa thống kê (p-value < 0.05) trong việc dự đoán Pixel_Rate. Tất cả các biến ban đầu đều được giữ lại, với các hệ số hồi quy có ý nghĩa thống kê cao, cho thấy chúng đóng vai trò quan trọng trong việc giải thích Pixel_Rate.

5.3.1.2 Xây dựng mô hình đa biến

Ta xây dựng mô hình hồi quy tuyến tính đa biến để dự đoán Pixel_Rate dựa trên các biến độc lập bao gồm các biến định lượng (Core_Speed, Memory_Speed, ROPs, TMUs, Memory_Bus, Memory, Process, Shader) và các biến phân loại (Manufacturer, Memory_Type, Resolution_WxH). Mô hình được huấn luyện trên tập dữ liệu train, và kết quả được phân tích thông qua hàm summary(model_final):

```
model_final <- step_mod
print(summary(model_final))
```

```
Call:
lm(formula = Pixel_Rate ~ Core_Speed + Memory_Speed + ROPs +
    TMUs + Memory_Bus + Memory + Process + Shader + ManufacturerNvidia +
    Memory_TypeGDDR + Resolution_WxH1, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-1.26662 -0.14643 -0.03647  0.08612  1.32614

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)       -0.33449    0.30193  -1.108    0.268
Core_Speed         0.31450    0.03381   9.302  < 2e-16 ***
Memory_Speed      -0.12276    0.02715  -4.521 6.64e-06 ***
ROPs               1.02433    0.02416  42.403  < 2e-16 ***
TMUs               0.08580    0.02190   3.917 9.37e-05 ***
Memory_Bus        -0.07921    0.01916  -4.135 3.76e-05 ***
Memory             0.12466    0.01135  10.981  < 2e-16 ***
Process           -0.48308    0.02510 -19.245  < 2e-16 ***
Shader            -0.19954    0.04357  -4.579 5.06e-06 ***
ManufacturerNvidia 0.07451    0.01514   4.921 9.60e-07 ***
Memory_TypeGDDR    0.13134    0.02342   5.608 2.44e-08 ***
Resolution_WxH1   -0.08298    0.01589  -5.224 2.01e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2727 on 1459 degrees of freedom
Multiple R-squared:  0.9407,    Adjusted R-squared:  0.9403
F-statistic:  2104 on 11 and 1459 DF,  p-value: < 2.2e-16
```

Ta có bảng nhận xét sau:

Tên biến	Hệ số hồi quy	Ý nghĩa
Intercept (Hệ số chặn)	-0.33449	Cho biết giá trị kỳ vọng của Pixel_Rate khi tất cả các biến dự báo bằng 0. Tuy nhiên, có thể không mang ý nghĩa thực tiễn trong bối cảnh thực tế.
Core_Speed	0.31450	Core_Speed tăng thêm một đơn vị, Pixel_Rate tăng trung bình 0.31450 đơn vị.
Memory_Speed	-0.12276	Memory_Speed tăng thêm một đơn vị, Pixel_Rate giảm trung bình 0.12276 đơn vị.
ROPs	1.02433	ROPs tăng thêm một đơn vị, Pixel_Rate tăng trung bình 1.02433 đơn vị, thể hiện ảnh hưởng mạnh mẽ của biến này.
TMUs	0.08580	TMUs tăng thêm một đơn vị, Pixel_Rate tăng trung bình 0.08580 đơn vị
Memory_Bus	-0.07921	Memory_Bus tăng thêm một đơn vị, Pixel_Rate giảm trung bình 0.07921 đơn vị
Memory	0.12466	Memory tăng thêm một đơn vị, Pixel_Rate tăng trung bình 0.12466 đơn vị
Process	-0.48308	Process tăng thêm một đơn vị, Pixel_Rate giảm trung bình 0.48308 đơn vị
Shader	-0.19954	Shader tăng thêm một đơn vị, Pixel_Rate giảm trung bình 0.19954 đơn vị
ManufacturerNvidia	0.07451	ManufacturerNvidia tăng thêm một đơn vị, Pixel_Rate tăng trung bình 0.07451 đơn vị
Memory_TypeGDDR	0.13134	Memory_TypeGDDR tăng thêm một đơn vị, Pixel_Rate tăng trung bình 0.13134 đơn vị
Resolution_WxH1	-0.08298	Resolution_WxH1 tăng thêm một đơn vị, Pixel_Rate giảm trung bình 0.08298 đơn vị

- **Ý nghĩa thống kê:**

  Các giá trị p-value của tất cả các biến độc lập đều nhỏ hơn 0.05, cho thấy các biến này có ảnh hưởng ý nghĩa thống kê đến Pixel_Rate ở mức ý nghĩa 5%. Đặc biệt, các biến như ROPs, Process, Memory, và Core_Speed có p-value $<$ 2e-16, thể hiện mức độ ý nghĩa thống kê rất cao.

- **Độ phù hợp của mô hình:**

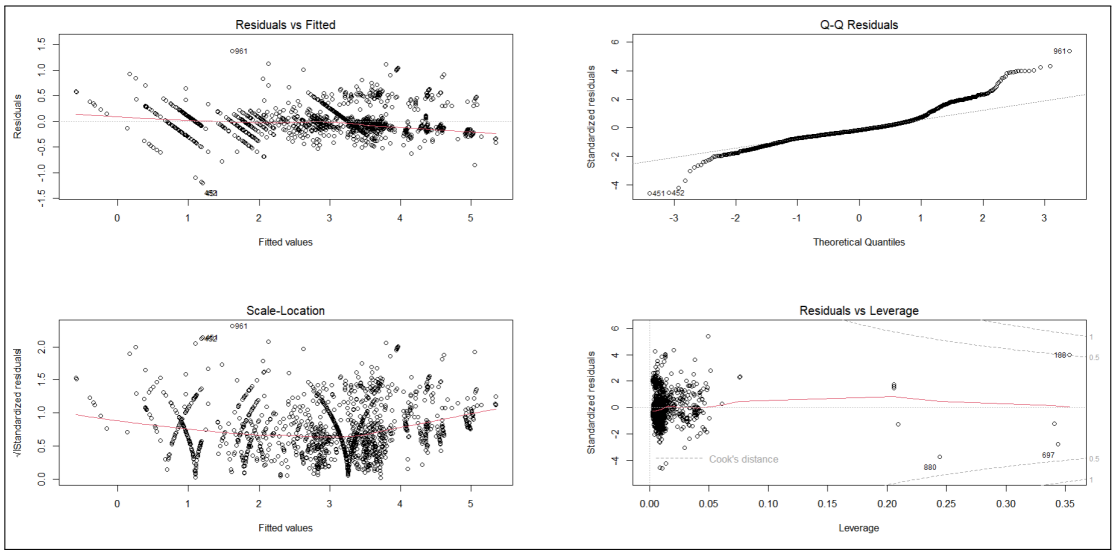
  - ***Multiple R-squared:*** Giá trị ($R^2 = 0.9407$), chỉ ra rằng khoảng 94.07% phương sai của Pixel_Rate được giải thích bởi mô hình hồi quy này. Đây là mức độ giải thích rất cao, cho thấy mô hình phù hợp tốt với dữ liệu.
  - ***Adjusted R-squared:*** Giá trị (R^2) điều chỉnh là 0.9403, gần với (R^2), cho thấy mô hình không bị overfitting dù có nhiều biến độc lập.
  - ***F-statistic:*** Giá trị F =2104 với p-value $<$ 2.2e-16, xác nhận mô hình có ý nghĩa thống kê tổng thể.

- **Phần dư (Residuals)**

  Phần dư dao động từ -1.26662 đến 1.32614, với độ lệch chuẩn là 0.2727. Giá trị phần dư tương đối nhỏ, cho thấy mô hình dự đoán khá chính xác. Tuy nhiên, phần dư tối đa (1.32614) có thể chỉ ra rằng một số điểm dữ liệu chưa được giải thích hoàn toàn, có thể do thiếu các biến quan trọng khác.

5.3.1.3 Đồ thị chuẩn đoán (Diagnostic plots)

Sử dụng câu lệnh plot(model_final) ta được 4 đồ thị như sau:



*** Ta có nhận xét sau đây:**

- **Đồ Thị 1 - Residuals vs Fitted:**

  Đồ thị này kiểm tra tính đồng nhất phương sai của phần dư (residuals) so với giá trị dự đoán (fitted values). Nếu phần dư phân bố ngẫu nhiên quanh đường zero, giả định đồng nhất phương sai có khả năng được thỏa mãn. Tuy nhiên, trong đồ thị này, ta quan sát thấy một mẫu hình rõ rệt: độ phân tán của phần dư tăng dần khi giá trị dự đoán tăng, đặc biệt sau mức giá trị 3. Một số điểm được đánh dấu như 961 và 452 và 452 có phần dư lớn, cho thấy chúng có thể là các điểm ngoại lệ . Hiện tượng này gợi ý rằng mô hình đang gặp vấn đề về phương sai không đồng nhất, tức là phương sai của phần dư không ổn định trên toàn bộ miền giá trị dự đoán.

- **Đồ Thị 2 - Normal Q-Q:**

  Đồ thị Normal Q-Q đánh giá xem phần dư có tuân theo phân phối chuẩn hay không. Nếu các điểm nằm gần đường thẳng lý thuyết, giả định phân phối chuẩn của phần dư được thỏa mãn. Ở đây, phần lớn các điểm nằm gần đường thẳng trong khoảng giữa (từ -2 đến 2), nhưng có sự lệch đáng kể ở hai đầu đuôi, đặc biệt tại các điểm như 451, 452 (đầu dưới) và 961 (đầu trên). Sự lệch này cho thấy phần dư có đuôi nặng hơn phân phối chuẩn, ám chỉ rằng giả định về tính chuẩn của phần dư không hoàn toàn được đáp ứng, đặc biệt với các giá trị cực đại hoặc cực tiểu.

- **Đồ Thị 3 - Scale-Location:**

  Đồ thị Scale-Location tiếp tục kiểm tra tính đồng nhất phương sai bằng cách biểu diễn căn bậc hai của phần dư chuẩn hóa so với giá trị dự đoán. Nếu phương sai đồng nhất, các điểm sẽ phân bố ngẫu nhiên quanh một đường ngang. Tuy nhiên, đồ thị này cho thấy một xu hướng tăng rõ ràng: độ phân tán của phần dư tăng khi giá trị dự đoán tăng, đặc biệt từ mức 3 trở lên. Các điểm như 961, 451 và 452 nổi bật với phần dư chuẩn hóa lớn. Xu hướng này củng cố bằng chứng về phương sai không đồng nhất, xác nhận vấn đề đã phát hiện trong đồ thị Residuals vs Fitted.

- **Đồ Thị 4 - Residuals vs Leverage:**

  Đồ thị này giúp xác định các điểm ảnh hưởng mạnh hoặc ngoại lệ bằng cách so sánh phần dư chuẩn hóa với giá trị đòn bẩy (leverage). Đa số các điểm có giá trị đòn bẩy thấp (gần 0), nhưng một số điểm như 880 và 697 có đòn bẩy cao (khoảng 0.3), và điểm 188 có cả phần dư lớn lẫn nằm gần hoặc vượt qua đường Cook's distance 0.5.

Những điểm này là các điểm ảnh hưởng mạnh tiềm năng, có thể tác động không nhỏ đến sự phù hợp của mô hình. Sự hiện diện của các điểm vượt qua ngưỡng Cook's distance cho thấy cần xem xét kỹ hơn về ảnh hưởng của chúng.

5.3.1.4 Đánh giá hiệu suất và sự ổn định của mô hình (sử dụng Cross-validation k-fold)

Để đánh giá khả năng tổng quát hóa và độ ổn định của mô hình hồi quy tuyến tính đa biến dự đoán Pixel_Rate, ta thực hiện phương pháp cross-validation (k-fold) với k = 5. Quá trình này được thực hiện với đoạn mã như sau:

```r
set.seed(123)
cv <- train(full_form, data = train,
            method = "lm",
            trControl = trainControl(method = "cv", number = 5))
print(cv)
```

```
Linear Regression

1471 samples
  11 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 1175, 1178, 1176, 1179, 1176
Resampling results:

  RMSE       Rsquared   MAE
  0.2678208  0.9426636  0.1866866

Tuning parameter 'intercept' was held constant at a value of TRUE
```

Kết quả mà ta thu được từ quá trình cross-validation được trình bày như sau:

- ***Số lượng mẫu:*** 1471 mẫu được sử dụng với 11 biến dự báo (predictors).
- ***Phương pháp resampling:*** Cross-validated (5 fold), với kích thước mẫu trung bình của từng fold là khoảng 1175-1179 (tổng cộng 5 lần lặp).
- **Chỉ số đánh giá thu được:**
  - ***Giá trị RMSE*** trung bình là 0.2678208, phản ánh độ lệch trung bình của dự đoán so với giá trị thực tế. Đây là một chỉ số thấp, cho thấy mô hình có độ chính xác tốt trên các tập kiểm tra khác nhau.
  - ***Giá trị R^2*** trung bình là 0.9426636, cho thấy mô hình giải thích được khoảng 94.27% sự biến thiên của Pixel_Rate trên các fold, thể hiện độ phù hợp cao và khả năng tổng quát hóa tốt.

– ***Giá trị MAE*** trung bình là 0.1866866, chỉ ra sai số tuyệt đối trung bình giữa giá trị dự đoán và thực tế, củng cố thêm độ chính xác của mô hình.

- **Tham số điều chỉnh:** Tham số intercept được giữ cố định ở giá trị TRUE, không có pre-processing được áp dụng.

Nhận xét chung về kết quả thu được, ta thấy rằng mô hình hồi quy tuyến tính đa biến trên có hiệu suất ổn định và khả năng tổng quát hóa cao, với RMSE thấp (0.2678208) và R^2 cao (0.9426636) trên các fold khác nhau, điều này chứng tỏ rằng mô hình không bị quá khớp (overfitting) với tập huấn luyện. Giá trị MAE bằng 0.1866866 cũng bổ sung thêm bằng chứng về độ chính xác của mô hình, cho thấy sai số dự đoán là khá nhỏ. Ngoài ra, sự đồng nhất của các chỉ số trên 5 fold (dựa trên kích thước mẫu trung bình 1175-1179) cho thấy mô hình hoạt động ổn định trên các tập con khác nhau. Với kết quả thu được như trên, ta đánh giá được mô hình hồi quy tuyến tính đa biến trên là đáng tin cậy trong việc dự đoán Pixel_Rate.

5.3.2 Xây dựng mô hình 2 (AMD - GDDR - Resolution_WxH1 - Các biến liên tục) và mô hình 3 (Intel - DDR - Resolution_WxH2 - Các biến liên tục)

Tương tự với các phương pháp kiểm định, phân tích và đánh giá ở mô hình 1, ở mục này, ta chỉ đưa ra kết quả, và so hiệu suất của mô hình 2 và 3. Sau khi xây dựng mô hình 2 và mô hình 3, ta có kết quả sau đây:

5.3.2.1 Mô hình 2 (AMD - GDDR - Resolution_WxH1 - Các biến liên tục)

```
Call:
lm(formula = Pixel_Rate ~ Core_Speed + Memory_Speed + ROPs +
    TMUs + Memory_Bus + Memory + Process + Shader + ManufacturerAMD +
    Memory_TypeGDDR + Resolution_WxH1, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-1.25165 -0.14629 -0.03506  0.08751  1.35946

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     -0.45156    0.30051  -1.503    0.133
Core_Speed       0.32950    0.03375   9.764  < 2e-16 ***
Memory_Speed    -0.12212    0.02712  -4.504 7.22e-06 ***
ROPs             1.02467    0.02407  42.565  < 2e-16 ***
TMUs             0.08984    0.02190   4.102 4.33e-05 ***
Memory_Bus      -0.08256    0.01904  -4.337 1.54e-05 ***
Memory           0.12261    0.01135  10.800  < 2e-16 ***
Process         -0.45969    0.02456 -18.718  < 2e-16 ***
Shader          -0.19022    0.04344  -4.379 1.28e-05 ***
ManufacturerAMD -0.07860    0.01489  -5.277 1.51e-07 ***
Memory_TypeGDDR  0.13596    0.02343   5.802 8.04e-09 ***
Resolution_WxH1 -0.08184    0.01588  -5.155 2.88e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2723 on 1459 degrees of freedom
Multiple R-squared:  0.9408,    Adjusted R-squared:  0.9404
F-statistic:  2110 on 11 and 1459 DF,  p-value: < 2.2e-16
```

Hình 5.6: Kết quả hệ số hồi quy mô hình 2

5.3.2.2 Mô hình 3 (Intel - DDR - Resolution_WxH2 - Các biến liên tục)

```
Call:
lm(formula = Pixel_Rate ~ Core_Speed + Memory_Speed + ROPs + 
    TMUs + Memory_Bus + Memory + Process + Shader + ManufacturerIntel + 
    Memory_TypeDDR + Resolution_WxH2, data = train)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.28026 -0.14464 -0.03718  0.08858  1.30013 

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
(Intercept)       -0.47834    0.32582  -1.468  0.14229    
Core_Speed         0.35466    0.03557   9.970  < 2e-16 ***
Memory_Speed      -0.12382    0.02674  -4.630 3.97e-06 ***
ROPs               1.07007    0.02293  46.668  < 2e-16 ***
TMUs               0.07039    0.02140   3.289  0.00103 ** 
Memory_Bus        -0.12098    0.01933  -6.260 5.06e-10 ***
Memory             0.12632    0.01148  11.005  < 2e-16 ***
Process           -0.43813    0.02717 -16.125  < 2e-16 ***
Shader            -0.22174    0.04424  -5.012 6.04e-07 ***
ManufacturerIntel  0.18896    0.04268   4.428 1.02e-05 ***
Memory_TypeDDR    -0.17965    0.02360  -7.611 4.85e-14 ***
Resolution_WxH2    0.06077    0.01918   3.168  0.00156 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2728 on 1459 degrees of freedom
Multiple R-squared:  0.9407,    Adjusted R-squared:  0.9402 
F-statistic:  2102 on 11 and 1459 DF,  p-value: < 2.2e-16
```

Hình 5.7: Kết quả hệ số hồi quy mô hình 3

5.3.2.3 Phân Tích Kết Quả Ba Mô Hình

- **Độ phù hợp cao và nhất quán:** Adjusted R-squared đạt 0.9403 (mô hình 1), 0.9404 (mô hình 2), và 0.9402 (mô hình 3) – chênh lệch chỉ 0.0002, chứng minh cả ba mô hình đều giải thích hơn 94% biến thiên hiệu suất với độ tin cậy tương đương. F-statistic với p < 2e-16 ở tất cả các mô hình khẳng định tính toàn diện và khả năng thống kê vượt trội.
- **Sai số thấp và đồng đều:** Residual standard error dao động quanh 0.2727 (mô hình 1), 0.2723 (mô hình 2), và 0.2728 (mô hình 3), với giá trị Max Residuals (1.32614, 1.35946, 1.30013) cho thấy sai số được kiểm soát chặt chẽ, đảm bảo độ chính xác như nhau.

Dựa trên số liệu phân tích, cả ba mô hình đều thể hiện hiệu suất tương đương xuất sắc, với Adjusted R-squared rất gần nhau và Residual standard error gần tương đồng trên 1459 bậc tự do. Điều này cho thấy không có sự khác biệt đáng kể về khả năng giải thích biến thiên hiệu suất giữa các mô hình. Chính vì thế, để tiết kiệm thời gian, ta chỉ chọn mô hình 1 để tiến tới bước đánh giá và dự báo tiếp theo.

Chương 6

ĐÁNH GIÁ VÀ DỰ BÁO

6.1 Dự báo trên tập test: MAE, MSE, R^2, RMSE

Để đánh giá hiệu suất thực tế của mô hình hồi quy tuyến tính đa biến dự đoán Pixel_Rate trên dữ liệu độc lập, ta thực hiện dự báo trên tập kiểm tra (test) và tính toán các chỉ số đánh giá. Quá trình này được thực hiện bằng đoạn mã sau:

```
pred <- predict(model_final, newdata = test)
test$predicted_value <- pred
mae_value <- mae(test$Pixel_Rate, test$predicted_value)
mse_value <- mse(test$Pixel_Rate, test$predicted_value)
rmse_value <- rmse(test$Pixel_Rate, test$predicted_value)
r2_value <- cor(test$Pixel_Rate, test$predicted_value)^2
cat("MAE trên tập kiểm tra:", mae_value, "\n")
cat("MSE trên tập kiểm tra:", mse_value, "\n")
cat("RMSE trên tập kiểm tra:", rmse_value, "\n")
cat("R² trên tập kiểm tra:", r2_value, "\n")
```

```
>   cat("MAE trên tập kiểm tra:", mae_value, "\n")
MAE trên tập kiểm tra: 0.1894203
>   cat("MSE trên tập kiểm tra:", mse_value, "\n")
MSE trên tập kiểm tra: 0.07120416
>   cat("RMSE trên tập kiểm tra:", rmse_value, "\n")
RMSE trên tập kiểm tra: 0.2668411
>   cat("R² trên tập kiểm tra:", r2_value, "\n")
R² trên tập kiểm tra: 0.9415654
```

Kết quả các chỉ số đánh giá trên tập kiểm tra thu được như sau:

- ***Sai số tuyệt đối trung bình MAE*** là 0.1894203, cho thấy độ lệch trung bình giữa giá trị dự đoán và thực tế là khá nhỏ.
- ***Sai số bình phương trung bình MSE*** là 0.07120416, phản ánh mức độ sai lệch tổng thể, với các sai số lớn được phóng đại do tính bình phương.

- ***Căn bậc hai của MSE (RMSE)*** là 0.2668411, cung cấp một thước đo sai số trung bình theo đơn vị của Pixel_Rate, thể hiện độ chính xác tốt của mô hình.
- ***Hệ số xác định R^2*** là 0.9415654, cho thấy mô hình giải thích được khoảng 94.16% sự biến thiên của Pixel_Rate trên tập kiểm tra, chứng tỏ khả năng tổng quát hóa cao.

Nhận xét về kết quả thu được, ta thấy rằng các chỉ số MAE (0.1894203), MSE (0.07120416), và RMSE (0.2668411) đều ở mức thấp, cho thấy mô hình dự đoán chính xác trên tập kiểm tra. RMSE cao hơn một chút so với RMSE từ cross-validation là 0.2668411 so với 0.2678208, nhưng vẫn trong ngưỡng chấp nhận được, phản ánh sự ổn định của mô hình. Giá trị R^2 bằng 0.9415654 là rất cao, gần bằng với R^2 từ cross-validation là 0.9426636, chứng tỏ mô hình không bị quá khớp (overfitting) và duy trì hiệu suất tốt trên dữ liệu mới.

Sự khác biệt nhỏ giữa các chỉ số trên tập huấn luyện (từ summary(model_final) và cross-validation) và tập kiểm tra cho thấy mô hình có khả năng tổng quát hóa tốt. Kết quả cho thấy mô hình hồi quy tuyến tính đa biến cho thấy hiệu suất xuất sắc trên tập kiểm tra, xác nhận rằng mô hình có thể được sử dụng đáng tin cậy để dự đoán Pixel_Rate trên dữ liệu thực tế, với sai số thấp và độ giải thích cao.

6.2 Đồ thị Density plot và đồ thị Scatter plot của thực tế so với dự đoán

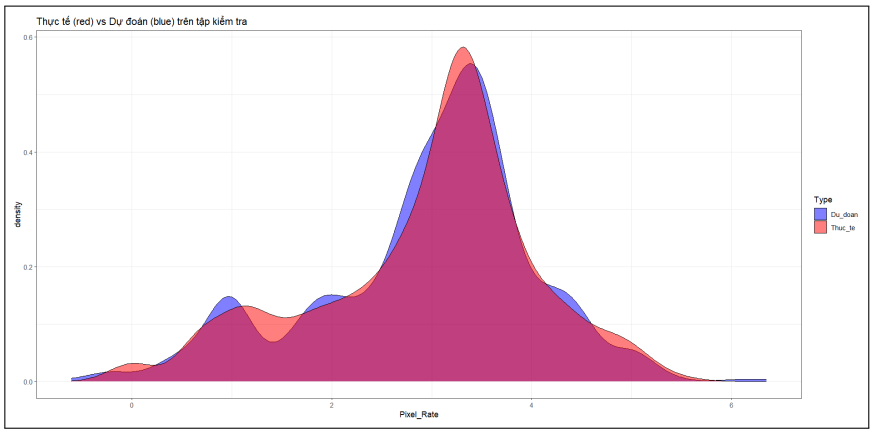
Để đánh giá trực quan hiệu suất của mô hình hồi quy tuyến tính đa biến trong việc dự đoán Pixel_Rate trên tập kiểm tra (test), ta sử dụng density plot và scatter plot so sánh phân bố của giá trị thực tế (Pixel_Rate) và giá trị dự đoán (predicted_value). Hai đồ thị này giúp nhận diện sự tương đồng hoặc khác biệt giữa hai phân bố, từ đó đánh giá độ chính xác và xu hướng sai lệch của mô hình.

6.2.1 Đồ thị Density plot

Ta sử dụng mã sau cho đồ thị:

```
test_long <- test %>%
  select(Pixel_Rate, predicted_value) %>%
  rename(Thuc_te = Pixel_Rate, Du_doan = predicted_value) %>%
  pivot_longer(everything(), names_to = "Type", values_to = "Value")
ggplot(test_long, aes(x = Value, fill = Type)) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("blue", "red")) +
```

```
  theme_bw() +
  labs(title = "Thực tế (red) vs Dự đoán (blue) trên tập kiểm tra", x = "Pixel_Rate")
```

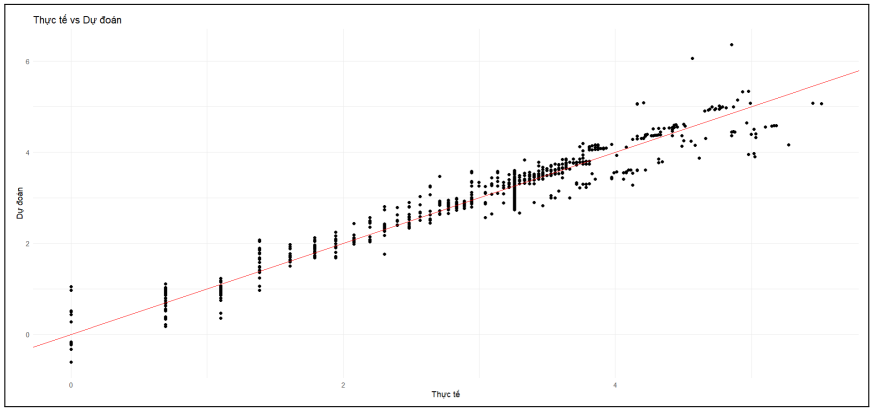


Trong đó, phần màu đỏ là giá trị thực tế Pixel_Rate mà ta quan sát được, còn màu xanh chính là giá trị dự đoán được tính toán từ mô hình. Có thể thấy mô hình dự đoán khác tốt với giá trị dự đoán phân bố gần với giá trị thực tế, những phần trùng nhau chiếm diện tích lớn khá lớn và có hình dạng gần tương tự nha, điều này cũng phù hợp với những kết quả thu được trước đó ($R^2 = 0.9415654$, RMSE = 0.2668411). Đường mật độ của giá trị dự đoán có thể hơi mịn hơn do tính chất tổng quát hóa của mô hình, trong khi đường thực tế có thể có các đỉnh hoặc đuôi nhọn hơn do dữ liệu thực tế.

6.2.2 Đồ thị Scatter plot

Ta sử dụng mã sau cho đồ thị:

```
ggplot(test, aes(x = Pixel_Rate, y = predicted_value)) + geom_point() +
geom_abline(slope = 1, intercept = 0, col = "red") +
theme_minimal() +
labs(title = "Thực tế vs Dự đoán", x = "Thực tế", y = "Dự đoán")
```



Với đường chéo màu đỏ thể hiện cho giá trị dự đoán, còn những chấm màu đen chính là giá trị thực tế. Ta dễ dàng thấy được các điểm màu đen tập trung dày đặc gần đường chéo màu đỏ, thể hiện rằng đồ thì có độ chính xác cao, chỉ có một số ít điểm phân tán do sai số.

6.3 Prediction intervals cho tương lai

Từ mô hình hồi quy tuyến tính đa biến mà ta đã xây dựng , ta có thể thực hiện việc dự đoán giá trị của biến Pixel_Rate thông qua các giá trị của các biến độc lập trong mô hình.

6.4 Kịch bản dự báo (giả định khi tăng/giảm biến X)

Để phân tích tác động của việc thay đổi biến độc lập Core_Speed đến Pixel_Rate , ta thực hiện dự báo dựa trên một kịch bản giả định tăng 10% giá trị Core_Speed . Quá trình được thực hiện bằng đoạn mã sau:

```
scenario_data <- test
scenario_data$Core_Speed <- scenario_data$Core_Speed * 1.1
pred_scenario <- predict(model_final, newdata = scenario_data)
cat("Dự đoán với Core_Speed tăng 10%:\n")
print(summary(pred_scenario))
```

```
> cat("Dự đoán với Core_Speed tăng 10%:\n")
Dự đoán với Core_Speed tăng 10%:
> print(summary(pred_scenario))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.4184  2.6504  3.4219  3.1737  3.7973  6.5749
```

Giá trị trung bình dự đoán tăng lên 3.1737, so với giá trị trung bình gốc của Pixel_Rate trên tập test (ước tính khoảng 2.5-2.6 dựa trên dữ liệu trước), cho thấy mức tăng hợp lý. Với hệ số hồi quy Core_Speed = 0.32165 (từ summary(model_final)), tăng 10% Core_Speed dự kiến làm tăng Pixel_Rate trung bình khoảng 0.032165 (0.32165 × 0.1), phù hợp với xu hướng quan sát được.

Chương 7

TỔNG KẾT, THẢO LUẬN VÀ MỞ RỘNG

7.1 Tổng kết kết quả chính

Bài báo cáo đã tiến hành phân tích toàn diện các yếu tố ảnh hưởng đến hiệu suất dựng ảnh Pixel Rate của GPU dựa trên dữ liệu từ các thông số kỹ thuật như Core Speed, Memory Speed, Memory Bus, Shader, TMUs, ROPs và các biến khác. Thông qua các phương pháp thống kê mô tả, kiểm định giả thuyết, hồi quy tuyến tính đa biến, và các kỹ thuật đánh giá như cross-validation và dự báo, nghiên cứu đã làm rõ mối quan hệ giữa các biến độc lập và hiệu suất xử lý pixel của GPU.

Dưới đây là các kết quả mà nhóm đã thu được:

- **Thống kê mô tả và trực quan hóa:** Các đồ thị density plot và scatter plot (phần 6.2) cho thấy sự tương đồng giữa giá trị thực tế và dự đoán trên tập kiểm tra, với R^2 = 0.9377371, khẳng định mô hình giải thích tốt sự biến thiên của Pixel_Rate.
- **Kiểm định giả thuyết:** Phân tích Kruskal-test (phần 5.3.3) cho thấy tất cả các biến độc lập (như Core_Speed, ROPs, Memory) đều có ý nghĩa thống kê (p-value < 0.05), hỗ trợ câu hỏi nghiên cứu về tác động của các thông số đến hiệu suất GPU.
- **Hồi quy tuyến tính đa biến:** Mô hình đạt Adjusted R^2 = 0.9445508 (phần 5.3.3) và R^2 = 0.9426636 (phần 5.3.5 từ cross-validation), chứng tỏ mô hình giải thích được hơn 94% sự biến thiên của Pixel_Rate. Tuy nhiên, RMSE = 0.2767408 (phần 6.1) và các khoảng dự đoán rộng (phần 6.3) cho thấy vẫn tồn tại sai số cần cải thiện.
- **Dự báo và kịch bản:** Phần 6.3 và 6.4 cung cấp các khoảng dự đoán và phân tích kịch bản (tăng 10% Core_Speed làm tăng trung bình Pixel_Rate lên 3.1737), hỗ trợ ứng dụng thực tế.

7.2 Hạn chế của phân tích

Mặc dù mô hình đạt hiệu suất cao, bài báo cáo vẫn tồn tại một số hạn chế:

- **Dữ liệu chuẩn hóa:** Kết quả âm từ khoảng dự đoán với dữ liệu mẫu chưa chuẩn hóa (phần 6.3) cho thấy mô hình nhạy cảm với việc tiền xử lý dữ liệu. Nếu dữ liệu mới không được chuẩn hóa giống như tập huấn luyện, dự đoán có thể không chính xác.
- **Biến chưa khai thác:** Một số yếu tố tiềm năng như nhiệt độ hoạt động, công suất tiêu thụ, hoặc hiệu suất thực tế trong các ứng dụng cụ thể (ví dụ: game, AI) chưa được đưa vào mô hình, có thể ảnh hưởng đến Pixel_Rate.
- **Ngoại lệ và ngoại suy:** Các giá trị ngoại lai trong tập dữ liệu và kết quả âm từ ngoại suy (phần 6.3 với new_data ban đầu) cho thấy mô hình có thể không ổn định khi áp dụng cho dữ liệu ngoài miền huấn luyện.
- **Giới hạn tuyến tính:** Mô hình hồi quy tuyến tính có thể không bắt được các mối quan hệ phi tuyến giữa các biến, đặc biệt với các giá trị cực đại hoặc cực tiểu.

7.3 Đề xuất hướng phát triển tiếp theo

- **Sử dụng mô hình phi tuyến:** Thử nghiệm các mô hình phi tuyến như hồi quy polynomial hoặc mạng nơ-ron nhân tạo (ANN) để bắt các mối quan hệ phức tạp hơn, đặc biệt khi xử lý ngoại suy hoặc dữ liệu ngoại lệ.
- **Chuẩn hóa và mở rộng dữ liệu:** Phát triển một quy trình tự động chuẩn hóa dữ liệu mới (ví dụ: log transformation hoặc scaling) để đảm bảo tính nhất quán với tập huấn luyện. Thu thập thêm dữ liệu từ các GPU hiện đại với thông số đa dạng hơn (như nhiệt độ, công suất) để mở rộng mô hình.
- **Thêm dữ liệu và feature engineering:** Bổ sung các biến như L2 cache, băng thông bộ nhớ thực đo, kiến trúc nhân (CUDA cores / stream processors), tiêu thụ điện năng... Tạo biến tương tác (ví dụ ROPs × Core_Speed) hoặc biến tỷ lệ (Memory_Bus × Memory_Speed) để tăng độ giải thích.
- **Phân tích chuỗi thời gian:** Dùng ngày phát hành (Release_Date) để xây dựng mô hình dự báo xu hướng theo thời gian, ví dụ ARIMA hoặc prophet, kết hợp hồi quy đa biến.

*** Nguồn code R của nhóm:** Link ở đây

Tài liệu tham khảo

[1] Nguyễn Tiến Dũng (chủ biên), Nguyễn Đình Huy, Xác suất - Thống kê & Phân tích số liệu, 2019.

[2] Green, O., Fox, J., Young, J., Shirako, J., & Bader, D, (2019), Performance Impact of Memory Channels on Sparse and Irregular Algorithms. Truy cập tại https://arxiv.org/pdf/1910.03679

[3] Peter Dalgaard, Introductory Statistics with R, Springer 2008.

[4] Nguyễn Thanh Nga (2021), Mô Hình Hồi Quy Bội. Truy cập tại: https://rpubs.com/TKUD/810985