VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY

Probability and Statistics (MT2013)

Assignment Report

Evaluate the impact of GPU's characteristics on its Memory Bandwidth

---

Instructor: Prf. Nguyễn Tiến Dũng, *HCMUT*

Student(s): Lê Minh Trung - 2252854, *(Group CCO1, Leader)*
Nguyễn Anh Quân - 2252678, *(Group CCO1)*
Nguyễn Đình Khôi Nguyên - 2053278, *(Group CCO1)*
Nguyễn Phúc Nhân - 2252563, *(Group CCO1)*

HO CHI MINH CITY, DECEMBER 2023

Contents

1 Member list & Workload

Name	Student ID	Contribution	Percentage
Lê Minh Trung	2252854	Data Analyzing & Coding	30%
Nguyễn Anh Quân	2252678	Project Writing & Coding	30%
Nguyễn Đình Khôi Nguyên	2053278	Background	20%
Nguyễn Phúc Nhân	2252563	Background	20%
~~Lê Minh Hưng~~	~~2252277~~		0%

2 Data introduction

2.1 Data context

This dataset contains detailed specifications, release dates, and release prices of computer parts.

We choosing this topic to know more about how each characteristics of GPU affect on the performance of the product.

2.2 Data description

The dataset contains two CSV files: *gpus.csv* for Graphics Processing Units (GPUs), and *cpus.csv* for Central Processing Units (CPUs). Each table has its own list of unique entries, but the list of features includes: clock speeds, maximum temperatures, display resolutions, power draws, number of threads, release dates, release prices, die size, virtualization support, and many other similar fields. For more specific column-level metadata refer to the Column Metadata.

The data given here belongs mainly to Intel, Game-Debate, and the companies involved in producing the part.

In our report, we just use *gpus.csv* since we focus on evaluate the characteristics of GPU on its **Memory Bandwidth**.

2.3 Variable description

1. **Architecture**: The underlying design or structure of the GPU.
2. **Best_Resolution**: The maximum display resolution supported.
3. **Boost_Clock**: The maximum clock speed that the GPU can reach under load.
4. **Core_Speed**: The speed at which the GPU's core operates.
5. **DVI_Connection**: Availability of a Digital Visual Interface connection for monitors.
6. **Dedicated**: Denotes whether the GPU is dedicated, meaning it is a separate unit and not integrated into the CPU.
7. **Direct_X**: The version of DirectX API supported.
8. **DisplayPort_Connection**: Indicates whether the GPU has a DisplayPort connection.
9. **HDMI_Connection**: Availability of an HDMI connection for monitors.
10. **Intergrated**: Whether the GPU is integrated into the CPU or motherboard.

11. **L2_Cache**: Size of the GPU's level 2 cache memory.
12. **Manufacturer**: The company or entity that produces the GPU.
13. **Max_Power**: Maximum power consumption of the GPU.
14. **Memory**: Amount of dedicated graphics memory.
15. **Memory_Bandwidth**: The maximum rate at which data can be read from or stored into the graphics memory.
16. **Memory_Bus**: Width of the memory bus.
17. **Memory_Speed**: The speed at which the GPU's memory operates.
18. **Memory_Type**: Type of memory used in the GPU (e.g., GDDR5, GDDR6).
19. **Name**: The model or name of the GPU.
20. **Notebook_GPU**: Whether the GPU is designed for notebook/laptop use.
21. **Open_GL**: The version of OpenGL supported.
22. **PSU**: Minimum Power Supply Unit required for the GPU.
23. **Pixel_Rate**: The maximum number of pixels the GPU can render to the screen per second.
24. **Power_Connector**: Specifies the type of power connector required to power the GPU (e.g., 6-pin, 8-pin).
25. **Process**: The manufacturing process node used to produce the GPU.
26. **ROPs**: Render Output Units - the number of operations the GPU can perform per clock cycle.
27. **Release_Date**: The date when the GPU was released.
28. **Release_Price**: The price at which the GPU was initially released.
29. **Resolution_WxH**: Supported resolutions in width x height format.
30. **SLI_Crossfire**: Support for NVIDIA SLI or AMD Crossfire multi-GPU configurations.
31. **Shader**: The version of shader model supported.
32. **TMUs**: Texture Mapping Units - the part of the GPU responsible for applying textures to polygons.
33. **Texture_Rate**: The rate at which the GPU can apply textures to polygons.
34. **VGA_Connection**: Availability of a VGA connection for monitors.

2.4 Memory_Bandwidth

Memory Bandwidth (GPU) – Memory bandwidth is one of the most frequently showcased stats for any new GPU, often rating in hundreds of gigabytes per second of throughput potential. Memory Bandwidth is the theoretical maximum amount of data that the bus can handle at any given time, playing a determining role in how quickly a GPU can access and utilize its framebuffer. Memory bandwidth can be best explained by the formula used to calculate it:

$$Memory\_Bandwidth = \frac{Memory\ bus\ width}{8 * Memory\ clock * 2 * 2} \tag{1}$$

The memory bus width is our Memory Interface, which is a given in specs listings. We're dividing by 8 to convert the bus width to bytes (for easier reading by humans). We then multiply by the memory clock (also a given – use GPU-Z to see this), then multiply the product by 2 (for DDR) and then by 2 again (for GDDR5). This gives us our memory bandwidth rating. Reason for choosing this topic: In our report, we want to know about the impact of each characteristics of GPUs on Memory Bandwidth stats. For example, when we increase memory bus width, it will also increase memory bandwidth as we can see in the previous formula. Or core speed can indirectly affect memory bandwidth by influencing the frequency of memory accesses and cache effectiveness, optimizing memory subsystem components such as memory speed, bus width, and memory controller efficiency is also crucial for maximizing overall system performance.

3 Background

3.1 kNN (k Nearest Neighbour)

The K-Nearest Neighbors (KNN) algorithm is a popular machine learning technique used for classification and regression tasks. It relies on the idea that similar data points tend to have similar labels or values.

During the training phase, the KNN algorithm stores the entire training dataset as a reference. When making predictions, it calculates the distance between the input data point and all the training examples, using a chosen distance metric such as Euclidean distance.

Next, the algorithm identifies the K nearest neighbors to the input data point based on their distances. In the case of classification, the algorithm assigns the most common class label among the K neighbors as the predicted label for the input data point. For regression, it calculates the average or weighted average of the target values of the K neighbors to predict the value for the input data point.

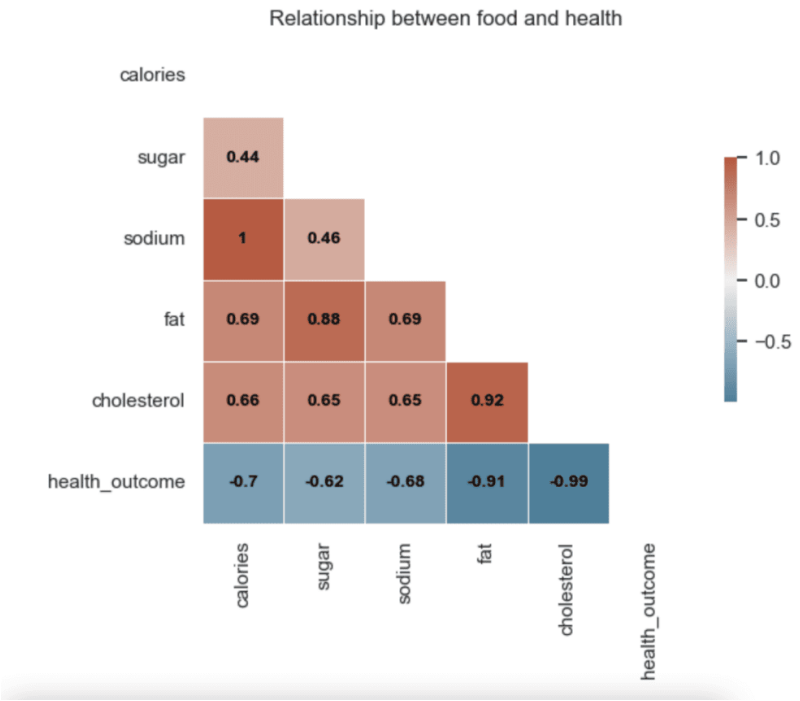
The KNN algorithm is straightforward and easy to understand, making it a popular choice in various domains. However, its performance can be affected by

the choice of K and the distance metric, so careful parameter tuning is necessary for optimal results.

KNN Algorithm can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry.

3.2 Correlation heatmap

A correlation heat map is a graphical representation of correlation data that shows the correlation coefficients between sets of variables. Each cell in the heat map shows the correlation between two variables and is often color-coded to reflect the correlation's magnitude.



Example for correlation heatmap

3.3 QQ plot

The QQ plot, or quantile-quantile plot, is a graphical tool to help us assess if a set of data plausibly came from some theoretical distribution such as a normal or exponential. For example, if we run a statistical analysis that assumes our residuals are normally distributed, we can use a normal QQ plot to check that assumption. It's just a visual check, not an air-tight proof, so it is somewhat subjective. But it allows us to see at-a-glance if our assumption is plausible, and if not, how the

assumption is violated and what data points contribute to the violation.

A QQ plot is a scatter plot created by plotting two sets of quantiles against one another. If both sets of quantiles came from the same distribution, we should see the points forming a line that's roughly straight. Here's an example of a normal QQ plot when both sets of quantiles truly come from normal distributions.

3.4 R square

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable in a regression model. In other words, r-squared shows how well the data fit the regression model (the goodness of fit).

Whereas correlation explains the strength of the relationship between an independent and a dependent variable, R-squared explains the extent to which the variance of one variable explains the variance of the second variable.

This is a basic formula for R-Square:

$$R^2 = 1 - \frac{UnexplainedVariation}{TotalVariation}$$

3.5 P-value

In statistics, a p-value is a number that indicates how likely you are to obtain a value that is at least equal to or more than the actual observation if the null hypothesis is correct.

The p-value serves as an alternative to rejection points to provide the smallest level of significance at which the null hypothesis

3.6 One-way ANOVA

3.6.1 Definition

A one-way ANOVA ("analysis of variance") compares the means of three or more independent groups to determine if there is a statistically significant difference between the corresponding population means.

3.6.2 One-way ANOVA: Assumptions

For the results of a one-way ANOVA to be valid, the following assumptions should be met:

- Normality: Each sample was drawn from a normally distributed population.
- Equal Variances: The variances of the populations that the samples come from are equal.

- Independence: The observations in each group are independent of each other and the observations within groups were obtained by a random sample.

3.6.3 One-way ANOVA: The Process

A one-way ANOVA uses the following null and alternative hypotheses:

- H_0(null hypothesis): $\mu_1 = \mu_2 = \mu_3 = ... = \mu_k$ (all the population means are equal)
- H_1 (alternative hypothesis): at least one population mean is different from the rest
- SSB: regression sum of squares between groups $= \sum_{i=1}^{k} n_i(\bar{x}_i - \bar{x})^2$
- SSW: error sum of squares within groups SSW $= \Sigma SS_i$ with $SS_i = \sum_{i=1}^{k}(x_{ij} - \bar{x}_i)$
- SST: total sum of squares ($SST = SSR + SSW$)
- df_r: regression degrees of freedom ($df_r = k - 1$)
- df_e: error degrees of freedom ($df_e = N - k$)
- df_t: total degrees of freedom ($df_t = N - 1$)
- MSB: regression mean square ($MSB = \frac{SSB}{df_r}$)
- MSW: error mean square ($MSW = \frac{SSW}{df_e}$)
- Then the F statistic itself is computed as $F = \frac{MSB}{MSW}$
- p: The p-value that corresponds to $F_{dfr,dfe}$

If the p-value is less than your chosen significance level (e.g. 0.05), then you can reject the null hypothesis and conclude that at least one of the population means is different from the other

3.7 Shapiro-Wilk Test

The Shapiro-Wilk's test or Shapiro test is a normality test in frequentist statistics. The null hypothesis of Shapiro's test H_0 is that the population is distributed normally.

Suppose a sample, say $x_1, x_2, ..., x_n$, has come from a normally distributed population. Then according to the Shapiro-Wilk's tests, it calculates the statistic called

W and here is its formula:

$$W = \frac{(\sum_{i=1}^{n} a_i x_i)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2}$$

Where a is called Shapiro-Wilk Test coefficient, the numerator is the estimate of the slope of observed data due to QQ-plot (an estimate of σ^2) and the denominator is the estimate of σ^2 If H_0 true, W should be equal to 1.

3.8 Levene's Test

Levene's test is used to check that variances are equal for all samples when your data comes from a non normal distribution. You can use Levene's test to check the assumption of equal variances before running a test of One-way ANOVA.

The null hypothesis for Levene's is that the variances are equal across all samples. In more formal terms, that's written as:
$H_0 : \sigma_1^2 = \sigma_2^2 = ... = \sigma_k^2$ The alternate hypothesis (the one you're testing), is that the variances are not equal for at least one pair: $H_1 : \sigma_1^2 \neq \sigma_2^2$ for example The test statistic is a little ugly and involves a few summations:

$$W = \frac{N-k}{k-1} \frac{\sum_{i=1}^{k} N_i (Z_i - Z)^2}{\sum_{i=1}^{k} \sum_{j=1}^{N_i} (Z_{ij} - Z_i)^2} \tag{3}$$

$Z_{i,j}$ can take on three meanings, depending on if you use the mean, median, or trimmed mean of any subgroup. In case ZZ_{ij} is the mean value of the i^{th} group this formula turn into the F-test we use in One-Way Anova: $W = \frac{MSB}{MSE}$

3.9 Kruskal-Wallis H Test

3.9.1 Definition

The Kruskal–Wallis test is a statistical test used to compare two or more groups for a continuous or discrete variable. It is a non-parametric test, meaning that it assumes no particular distribution of your data and is analogous to the one-way analysis of variance (ANOVA). The Kruskal Wallis test is sometimes referred to as the one-way ANOVA on ranks or the Kruskal Wallis one-way ANOVA.

The hypotheses of the Kruskal–Wallis test are as follows:

- The null hypothesis (H_0) is that the population medians are equal.
- The alternative hypothesis (H_1) is that the population medians are not equal, or that the population median differs from the population median of one of the other groups.

3.9.2 Assumptions

Assumptions for the Kruskal–Wallis test are detailed below:

- Data are assumed to be non-Normal or take a skewed distribution. One-way ANOVA should be used when data follow a Normal distribution.
- The variable of interest should have two or more independent groups. The test is most commonly used in the analysis of three or more groups – for analyzing two groups the Mann-Whitney U test should be used instead.
- The data are assumed to take a similar distribution across the groups.
- The data should be randomly selected independent samples, in that the groups should have no relationship to each other.
- Each group sample should have at least 5 observations for a sufficient sample size.

These assumptions are similar to the Mann–Whitney U test, as the Kruskal–Wallis test is essentially an extension of that test with more than two independent samples. Similar to the Mann-Whitney U Test, the Kruskal–Wallis test is based on ranking the data and calculating a test statistic.

3.9.3 When to use the Kruskal–Wallis test

The Kruskal Wallis test and other non-parametric (or distribution-free) tests are useful to test hypotheses when the assumption for normality of the data does not hold. They make no assumptions about the shape of data distributions and this makes them particularly useful when a dataset is small. It is important to note that when conducting non-parametric statistical tests, they tend to give a more conservative result (a larger p-value) than their parametric counterparts. The Kruskal Wallis test should be used when the variable of interest is continuous (taking any number within a range eg. age, height, blood pressure) or discrete (taking on a certain value that can be counted eg. shoe size, number of hospital visits, number of people in a household).

3.10 Dunn's Test

If the results of a Kruskal-Wallis test are statistically significant, then it's appropriate to conduct Dunn's Test to determine exactly which groups are different. Dunn's Test performs pairwise comparisons between each independent group and tells you which groups are statistically significantly different at some level of α.

The formula to calculate the z-test statistic for the difference between two groups is:

$$z_i = \frac{y_i}{\sigma_i} \tag{4}$$

where i is one of the 1 to m comparisons, $y_i = W_A \breve{} W_B$ (where W_A is the average of the sum of the ranks for the i^{th} group) and σ_i is calculated as:

$$\sigma_i = \sqrt{(\frac{n(n+1)}{12}) - (\frac{\Sigma(f^3 - f)}{12(n-1)})(\frac{1}{n_i} + \frac{1}{n_j})} \tag{5}$$

where n is the total number of observations across all groups, and f is the number of observations.

The Dunn Test is commonly used in fields such as medical research, biology, and social sciences, where researchers frequently need to compare differences between experimental and control groups. It is particularly useful when the data does not follow a normal distribution or when sample sizes are unequal among groups.

3.11 Mean Absolute Error

Mean absolute error (MAE) is a popular metric because, as with Root mean squared error (RMSE), see next subsection, the error value units match the predicted target value units. Unlike RMSE, the changes in MAE are linear and therefore intuitive. MSE and RMSE penalize larger errors more, inflating or increasing the mean error value due to the square of the error value. In MAE, different errors are not weighted more or less, but the scores increase linearly with the increase in errors. The MAE score is measured as the average of the absolute error values. The Absolute is a mathematical function that makes a number positive. Therefore, the difference between an expected value and a predicted value can be positive or negative and will necessarily be positive when calculating the MAE.

The MAE value can be calculated as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \tilde{y}_i|^2 \tag{6}$$

where: n is the number of observations in dataset.
y_i is the true value.
$\tilde{y}_i$ is the predicted value.

The MAE is a linear score, meaning all individual differences contribute equally to the mean. It provides an estimate of the size of the inaccuracy, but not its direction (e.g., over or under-prediction).

3.12 Pearson

Pearson coefficient also called correlation coefficient is used to determine whether the relationship between variables is linear or not

$$r_{xy} = \frac{\overline{xy} - \bar{x}\bar{y}}{\widehat{S_x}\widehat{S_y}} = \frac{S_{xy}}{\sqrt{S_{xx}S^{yy}}} \tag{7}$$

Note: $-1 \leq r_{xy} \leq 1$
Conclusion:
$|r_{xy}| \leq 0.3$: There is no linear relationship or the linear relationship is very weak
$0.3 < |r_{xy}| \leq 0.5$ has weak linear relationship
$0.5 < |r_{xy}| \leq 0.8$ has normal linear relationship
$0.8 < |r_{xy}|$: x, y has strong linear relationship
Furthermore, $|r_{xy}| < 0$: decreasing function, $|r_{xy}| > 0$: increasing function

3.13 Multiple linear regression

Multiple linear regression refers to a statistical technique that is used to predict the outcome of a variable based on the value of two or more variables. It is sometimes known simply as multiple regression, and it is an extension of linear regression. The variable that we want to predict is known as the dependent variable, while the variables we use to predict the value of the dependent variable are known as independent or explanatory variables. Multiple Linear Regression Formula:

$$y_i = \beta_0 + \beta_1 x_i 1 + \beta_2 x_i 2 + ... + \beta_p x_i p + \varepsilon \tag{8}$$

where:

- y_i is the dependent or predicted variable
- β_0 is the y-intercept
- β_1 and β_2 are the regression coefficients representing the change in y relative to a one-unit change in x_{i1} and x_{i2}, respectively.
- β_p is the slope coefficient for each independent variable
- ε is the model's random error (residual) term.

3.13.1 Assumptions

Multiple linear regression is based on the following assumptions:

1. A linear relationship between the dependent and independent variables: There is a linear relationship between the dependent variable and each of the independent variables.
2. The independent variables are not highly correlated with each other: The data should not show multicollinearity, which occurs when the independent variables (explanatory variables) are highly correlated. When independent variables show multicollinearity, there will be problems figuring out the specific variable that contributes to the variance in the dependent variable.
3. The variance of the residuals is constant: Multiple linear regression assumes that the amount of error in the residuals is similar at each point of the linear model.

4. Independence of observation: The model assumes that the observations should be independent of one another. Simply put, the model assumes that the values of residuals are independent.

5. Multivariate normality: Multivariate normality occurs when residuals are normally distributed.

3.14 Random Forest

Random Forest is a widely used machine learning algorithm for classification, belonging to the ensemble learning group. It involves the creation of multiple decision trees during the training process and outputs the mode of the classes (for classification) or the mean/median of the predictions (for regression) from these trees. The essence of Random Forest is combining multiple simpler models to create a stronger overall model. Below are the key components of Random Forest:

3.14.1 Bootstrap Aggregating (Bagging)

Random Forest utilizes the Bagging technique to improve prediction accuracy. In Bagging, multiple random subsets are created from the original training dataset through sampling with replacement. Each subset is then used to train an independent decision tree. Ultimately, the predictions from each tree are aggregated to produce the final prediction.

3.14.2 Decision Trees

Decision trees are the fundamental components of Random Forest. Each tree is built on a dataset sampled from the original data. A distinctive feature of Random Forest compared to traditional decision tree methods is that at each node of the tree, the attribute selection for splitting the data is based on a subset of features rather than all features. This helps reduce overfitting and increases the diversity of the trees in the model.

3.14.3 How to Implement Random Forest

- **Model Building**

  Choose Parameters: Decide on the number of trees to include in the forest and other hyperparameters such as the maximum depth of the trees, the minimum number of samples required to split an internal node, and the number of features to consider when looking for the best split. These parameters can significantly affect the performance and speed of your Random Forest.

  + **Training the Model**: Use the training dataset to build each decision tree. Random subsets of the data and features are selected to train each

tree independently. This randomness helps in creating a diverse set of trees and reduces overfitting.

- **Model Evaluation**
  + **Cross-validation**: Use k-fold cross-validation to assess model robustness. This involves dividing the data into k subsets and iteratively training the model on k-1 subsets while using the remaining subset for testing.
  + **Performance Metrics**: Evaluate the model using appropriate metrics. For classification tasks, metrics like accuracy, precision, recall, and F1-score are common. For regression tasks, mean squared error (MSE), mean absolute error (MAE), and R-squared are used.
- Model Tuning
  + **Hyperparameter Tuning**: Use techniques like grid search or random search to find the optimal parameters for your Random Forest model. This can improve model accuracy and efficiency.
  + **Feature Importance Evaluation**: Analyze the importance scores assigned to each feature in the Random Forest. This can provide insights into which features are most influential in predicting the outcome, allowing for possible feature engineering or refinement.
- **Prediction and Interpretation**
  + **Making Predictions**: Once the model is trained and validated, use it to make predictions on new data.
  + **Interpret Results**: Although individual tree interpretations are straightforward, interpreting a Random Forest model as a whole can be challenging due to its ensemble nature. However, examining feature importance and the general trends in predictions can provide valuable insights.

4 Data preprocessing

4.1 Data reading

We will read the data in the "*All_GPUs.csv*" file and summarize them to know the attributes of the data.

```
data <- read.csv("/content/All_GPUs.csv")
summary(data)
```

4.2 Data cleaning

First, we will convert invalid data (missing lacking units, incorrect format, etc.) into *NA* for easier handling. For example, in the **Boost_Clock** column, values that do not contain "MHz" will be converted to *NA*, or in the **Memory_Bandwidth** column, values that do not contain "GB" will be converted to *NA* too. Then, we will find the number of missing values in each column, calculate the percentages and finally, print them.

```
data$Best_Resolution <- ifelse(!grepl("x", data$Best_Resolution), NA,
    data$Best_Resolution)
data$Boost_Clock <- ifelse(!grepl("MHz", data$Boost_Clock), NA, data$
    Boost_Clock)
data$Core_Speed <- ifelse(!grepl("MHz", data$Core_Speed), NA, data$Core_
    Speed)
data$Dedicated <- ifelse(data$Dedicated == "Yes", 1, ifelse(data$
    Dedicated == "No", 0, NA))
data$Direct_X <- ifelse(!grepl("DX", data$Direct_X), NA, data$Direct_X)
data$Integrated <- ifelse(data$Integrated == "Yes", 1, ifelse(data$
    Integrated == "No", 0, NA))
data$L2_Cache <- ifelse(!grepl("KB", data$L2_Cache), NA, data$L2_Cache)
data$Max_Power <- ifelse(!grepl("Watt", data$Max_Power), NA, data$Max_
    Power)
data$Memory <- ifelse(!grepl("MB", data$Memory), NA, data$Memory)
data$Memory_Bandwidth <- ifelse(!grepl("GB", data$Memory_Bandwidth), NA,
    data$Memory_Bandwidth)
data$Memory_Bus <- ifelse(!grepl("Bit", data$Memory_Bus), NA, data$Memory
    _Bus)
data$Memory_Speed <- ifelse(!grepl("MHz", data$Memory_Speed), NA, data$
    Memory_Speed)
data$Memory_Type <- ifelse(!grepl("DDR", data$Memory_Type), NA, data$
    Memory_Type)
data$Notebook_GPU <- ifelse(data$Notebook_GPU == "Yes", 1, ifelse(data$
    Notebook_GPU == "No", 0, NA))
data$PSU <- ifelse(!grepl("Watt", data$PSU), NA, data$PSU)
data$Pixel_Rate <- ifelse(!grepl("Pixel", data$Pixel_Rate), NA, data$
    Pixel_Rate)
data$Process <- ifelse(!grepl("nm", data$Process), NA, data$Process)
data$ROPs[data$ROPs == ""] <- NA
data$Shader[data$Shader == ""] <- NA
data$TMUs[data$TMUs == ""] <- NA
```

```r
data$VGA_Connection[data$VGA_Connection == ""] <- NA
data$Release_Date <- ifelse(!grepl("-", data$Release_Date), NA, data$Release_Date)
data$Release_Price <- ifelse(!grepl(".", data$Release_Price), NA, data$Release_Price)
data$Resolution_WxH <- ifelse(!grepl("x", data$Resolution_WxH), NA, data$Resolution_WxH)
data$SLI_Crossfire <- ifelse(data$SLI_Crossfire == "Yes", 1, ifelse(data$SLI_Crossfire == "No", 0, NA))
data$Texture_Rate <- ifelse(!grepl("Texel", data$Texture_Rate), NA, data$Texture_Rate)
missing_values <- colSums(is.na(data))
percentage_missing <- (missing_values / nrow(data)) * 100
cat("Columns with missing values and their respective percentages:\n")
for (column in names(data)) {
  if (missing_values[column] > 0) {
    cat("Column:", column, " - Missing Percentage:", percentage_missing[column], "%\n")
  }
}
```

Percentages of missing values:

```
Columns with missing values and their respective percentages:
Column: Best_Resolution  - Missing Percentage: 18.84909 %
Column: Boost_Clock  - Missing Percentage: 57.54551 %
Column: Core_Speed  - Missing Percentage: 27.48092 %
Column: DVI_Connection  - Missing Percentage: 22.01996 %
Column: Dedicated  - Missing Percentage: 0.4110393 %
Column: Direct_X  - Missing Percentage: 0.1761597 %
Column: DisplayPort_Connection  - Missing Percentage: 74.83852 %
Column: HDMI_Connection  - Missing Percentage: 22.40164 %
Column: Integrated  - Missing Percentage: 0.4110393 %
Column: Max_Power  - Missing Percentage: 18.34997 %
Column: Memory  - Missing Percentage: 12.33118 %
Column: Memory_Bandwidth  - Missing Percentage: 3.669994 %
Column: Memory_Bus  - Missing Percentage: 1.820317 %
Column: Memory_Speed  - Missing Percentage: 3.082795 %
Column: Memory_Type  - Missing Percentage: 2.084557 %
Column: Open_GL  - Missing Percentage: 1.174398 %
Column: PSU  - Missing Percentage: 34.49794 %
Column: Pixel_Rate  - Missing Percentage: 15.97181 %
Column: Process  - Missing Percentage: 13.59366 %
Column: ROPs  - Missing Percentage: 15.79565 %
Column: Release_Date  - Missing Percentage: 0.8807986 %
Column: Release_Price  - Missing Percentage: 83.67587 %
Column: Resolution_WxH  - Missing Percentage: 5.725191 %
Column: Shader  - Missing Percentage: 3.141515 %
Column: TMUs  - Missing Percentage: 15.79565 %
Column: Texture_Rate  - Missing Percentage: 15.97181 %
Column: VGA_Connection  - Missing Percentage: 22.25484 %
```

4.2.1 Process missing data

Now, we remove the irrelevant columns as we believe that they don't support our topic: **Architecture, Manufacturer, Name, Power_Connector**

Based on the results, we will also remove the columns that have lost too much data (more than 50%): **Boost_Clock, DisplayPort_Connection, Release_Price**.

Next, we remove the rows that contain missing value of the columns that have

the percentages of missing values less than 10%: **Dedicated, Direct_X, Integrated, Memory_Bandwidth, Memory_Bus, Memory_Speed, Memory_Type, Open_GL, Release_Date, Shader, Resolution_WxH**, and this will not affect the outcome.

After that, we move the modified columns into a new file: "*modified_dataset.csv*"

```
data_subset <- data[, !(names(data) %in% c("Architecture", "Manufacturer
    ", "Name", "Power_Connector", "Boost_Clock", "DisplayPort_Connection"
    , "Release_Price"))]
columns_to_check <- c("Dedicated", "Direct_X", "Integrated", "Memory_
    Bandwidth", "Memory_Bus", "Memory_Speed", "Memory_Type", "Open_GL", "
    Release_Date", "Shader", "Resolution_WxH")
data_subset <- data_subset[complete.cases(data_subset[, columns_to_check
    ]), ]
write.csv(data_subset, file = "modified_dataset.csv", row.names = FALSE)
```

With the columns which have the percentages of missing values between 10% and 50%, we must fill the missing values in order to use them. In this project, we use the **k-Nearest Neighbors algorithms** to fill the missing values.

And because, we use kNN algorithms, which is a supervised machine learning method and has some randomness so when we run the code multiple times, the results will slightly vary, but overall it does not affect the outcome.

We choose $k = 55$ because when we try to use the number between 0 and 100, 55 seems to be the most suitable number for this dataset, it is approximately the square root of 3000.

After that, we export the imputed dataset to a new file: "*imputed_modified_dataset.csv*"

```
#Taking square root of 3000 which is about 55
imputed_modified_dataset <- kNN(modified_dataset, k = 55)
imputed_modified_dataset <- imputed_modified_dataset[, c("Best_Resolution
    ", "Core_Speed", "DVI_Connection", "Dedicated", "Direct_X", "HDMI_
    Connection", "Integrated", "L2_Cache", "Max_Power", "Memory", "Memory_
    Bandwidth", "Memory_Bus", "Memory_Speed", "Memory_Type", "Notebook_GPU
    ", "Open_GL", "PSU", "Pixel_Rate", "Process", "ROPs", "Release_Date",
    "Resolution_WxH", "SLI_Crossfire", "Shader", "TMUs", "Texture_Rate", "
    VGA_Connection")]
write.csv(imputed_modified_dataset, file = "imputed_modified_dataset.csv"
    , row.names = FALSE)
```

Now, we would like to transform data from raw form to an analyzable format. For example, in the **Max_Power** column, we would like to have "141" instead of "141 Watts". After transform the data, we write them back to a CSV file, called "*cleaned_dataset.csv*"

```
cleaned_dataset <- data %>%
mutate(
Best_Resolution = sapply(Best_Resolution, multiply_resolution),
Core_Speed = sapply(Core_Speed, extract_number),
L2_Cache = sapply(L2_Cache, extract_number),
Max_Power = sapply(Max_Power, extract_number),
```

```
Memory = sapply(Memory, extract_number),
Memory_Bandwidth = sapply(Memory_Bandwidth, extract_number),
Memory_Bus = sapply(Memory_Bus, extract_number),
Memory_Speed = sapply(Memory_Speed, extract_number),
Pixel_Rate = sapply(Pixel_Rate, extract_number),
Process = sapply(Process, extract_number),
Resolution_WxH = sapply(Resolution_WxH, multiply_resolution),
Texture_Rate = sapply(Texture_Rate, extract_number),
ROPs = sapply(ROPs, handle_rops)
)
write_csv(cleaned_dataset, "cleaned_dataset.csv")
```

5 Descriptive statistics

5.1 Correlation map

We will draw a correlation heat map based on the cleaned dataset in the previous part to find the relevant features of GPU that seems to affect the memory bandwidth value.

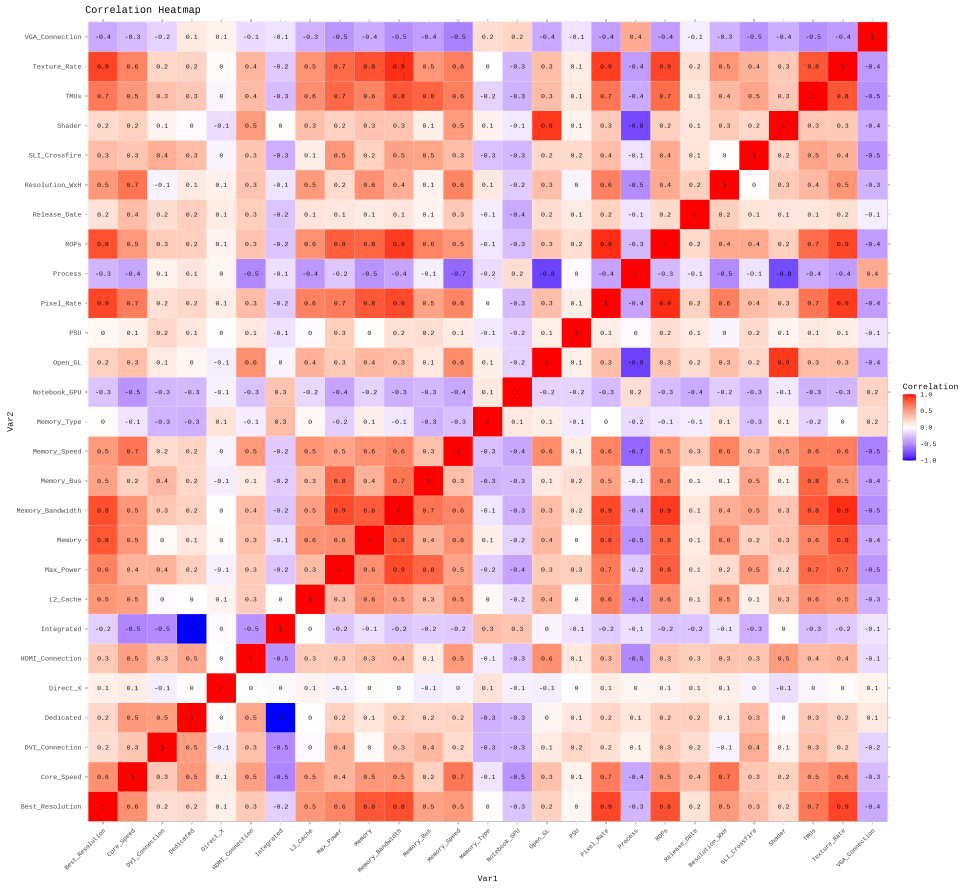
Firstly, we need to encode the data in the columns. Label encoding is a method used in Machine Learning to convert categorical data into numerical form. For each unique value in the categorical data, label encoding assigns a unique integer. Then, based on these integer, we will find the correlations of other factors.

In order to determine the correlation between columns in the dataset, we utilized the Pearson correlation coefficient. This statistical measure assesses the linear relationship between two continuous variables.

```
# Define a function for label encoding
label_encode <- function(column) {
  levels <- unique(column)
  labels <- as.integer(factor(column, levels = levels))
  return(labels)
}
# Apply label encoding to non-numeric columns
encoded_dataset <- cleaned_dataset %>%
  mutate_if(~ !is.numeric(.), label_encode)
# Compute correlation matrix
correlation_matrix <- cor(encoded_dataset, use = "pairwise.complete.obs")
# Create a dataframe for correlation values
correlation_df <- as.data.frame(as.table(correlation_matrix))
names(correlation_df) <- c("Var1", "Var2", "Correlation")
# Create a heatmap
heatmap <- ggplot(data = correlation_df, aes(Var1, Var2, fill =
    Correlation, label = round(Correlation, 1))) +
  geom_tile(color = "white") +
  geom_text(color = "black", size = 3) +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1,1), space = "Lab",
                       name="Correlation") +
```

```
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, size = 8, hjust
      = 1)) +
  coord_fixed() +
  labs(title = "Correlation Heatmap")
# Save the heatmap with larger dimensions
ggsave("correlation_heatmap.png", heatmap, width = 20, height = 16, dpi =
    300)
```

Correlation heat map:



The sample size plays a crucial role in the accuracy of correlation estimates. In this project, we have a large sample (3000 observations), then the sample correlation coefficient remains approximately unbiased, but may not be efficient. So we will choose the columns that have the correlations with **Memory_Bandwidth** more than 0.5.

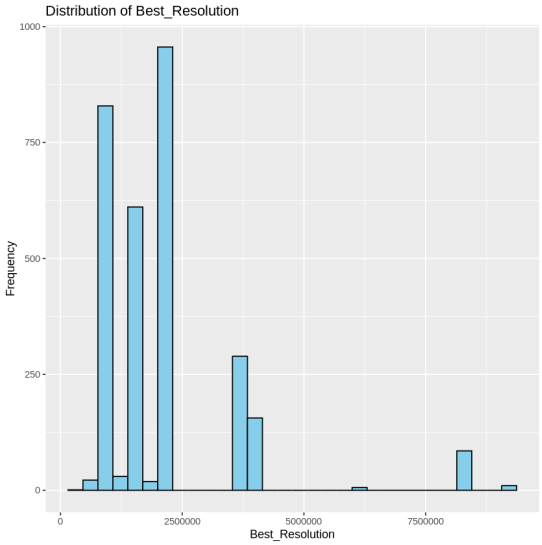
From the correlation heat map, we can choose the relevant features of GPU that seems to affect the memory bandwidth value which are the values: **Best_Resolution,**

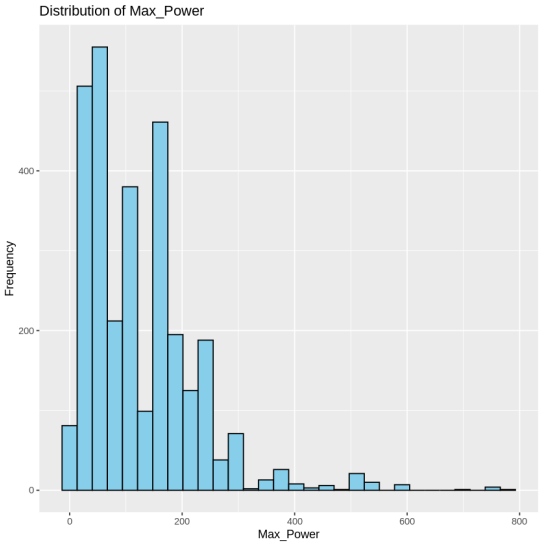
Max_Power, Memory, Memory_Bus, Memory_Speed, Pixel_Rate, ROPs, TMUs, Texture_Rate

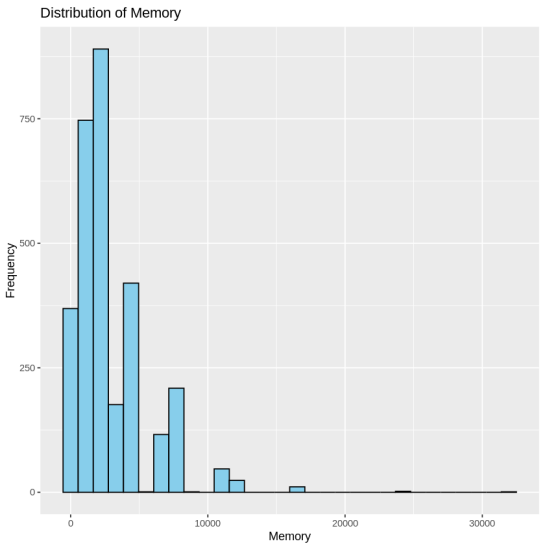
5.2 Plot

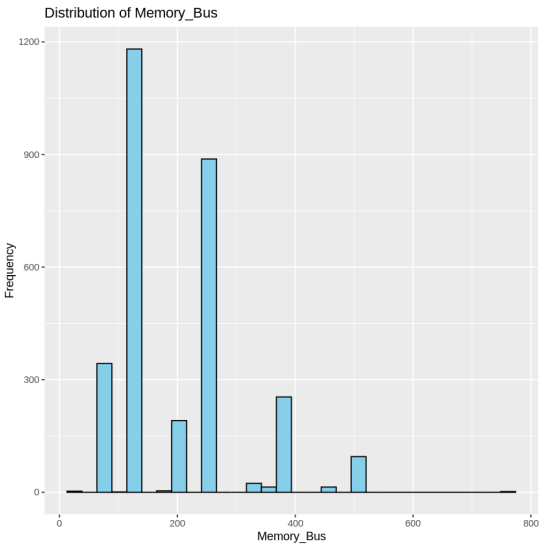
We will draw the distribution plot for 9 columns above and also the **Memory_Bandwidth** column.

```r
# Function to draw distribution plot
draw_distribution <- function(cleaned_dataset, bins = 30) {
  columns_to_plot <- c("Best_Resolution", "Max_Power", "Memory", "Memory_Bus", "Memory_Speed", "Pixel_Rate", "ROPs", "TMUs", "Texture_Rate", "Memory_Bandwidth"
  )
  for (col in columns_to_plot) {
    if (is.numeric(cleaned_dataset[[col]])) {
      plot <- ggplot(cleaned_dataset, aes(x = !!rlang::sym(col))) +
        geom_histogram(bins = bins, fill = "skyblue", color = "black") +
        labs(title = paste("Distribution of", col),
             x = col, y = "Frequency")
    } else {
      plot <- ggplot(cleaned_dataset, aes(x = !!rlang::sym(col), fill = !!rlang::sym(col))) +
        geom_bar() +
        labs(title = paste("Distribution of", col),
             x = col, y = "Frequency")
    }
    print(plot)
  }
}
# Call the function to draw distribution plots for specific columns
draw_distribution(cleaned_dataset, bins = 30)
```

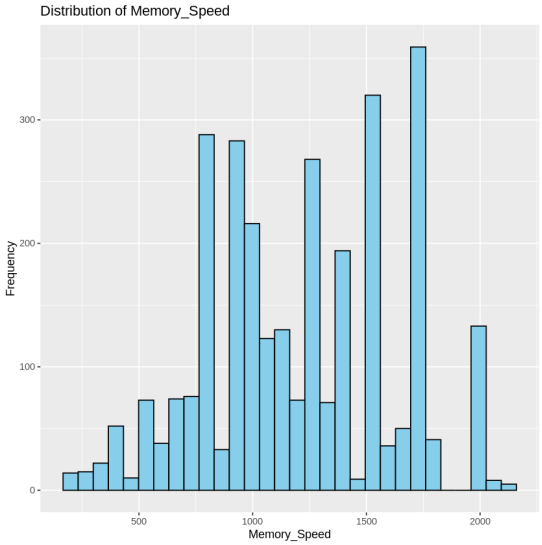

Distribution of Best_Resolution
Frequency
Best_Resolution
0
250
500
750
1000
0
2500000
5000000
7500000

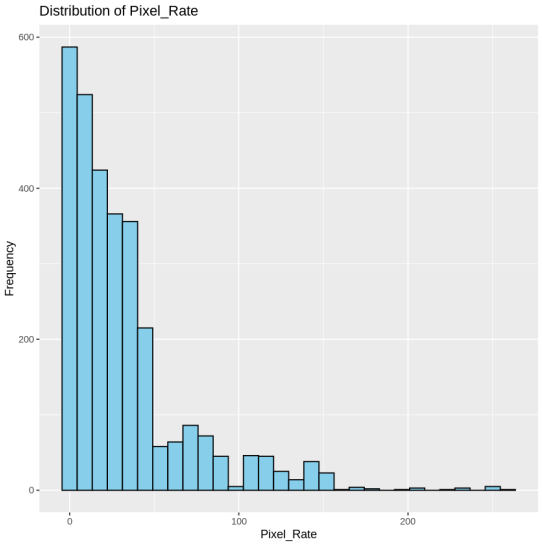

Distribution of Max_Power
Frequency
Max_Power
0
200
400
0
200
400
600
800

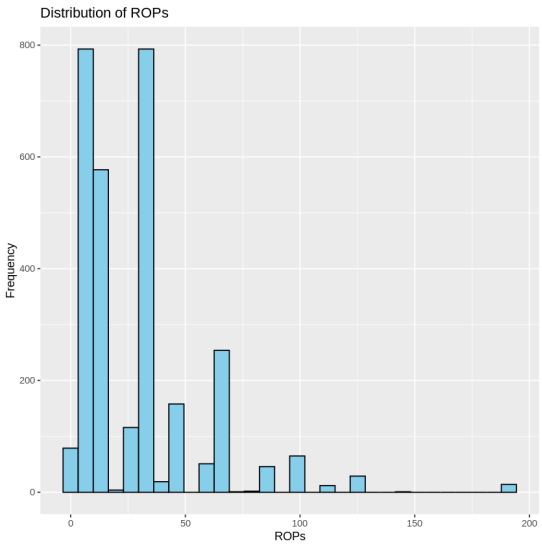

Distribution of Memory
Frequency
Memory
0
250
500
750
0
10000
20000
30000

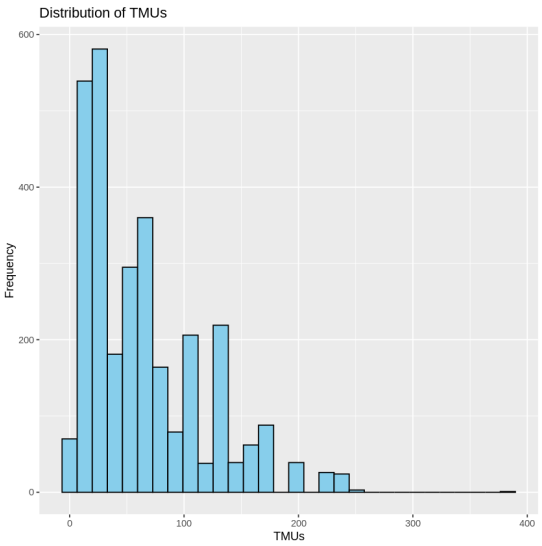

Distribution of Memory_Bus
Frequency
Memory_Bus
0
300
600
900
1200
0
200
400
600
800

.

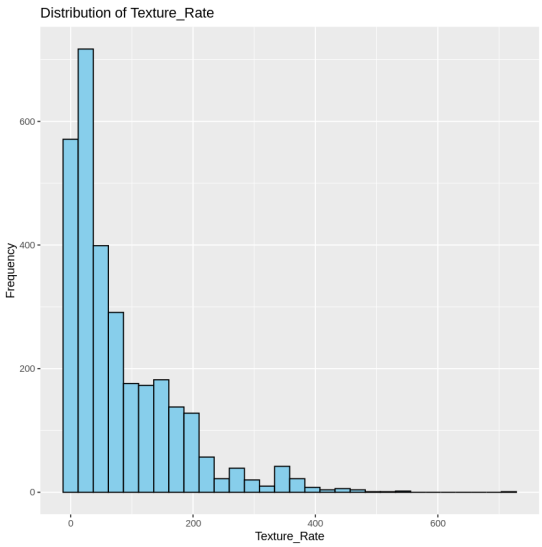

Distribution of Memory_Speed
Frequency
Memory_Speed
0
100
200
300
500
1000
1500
2000

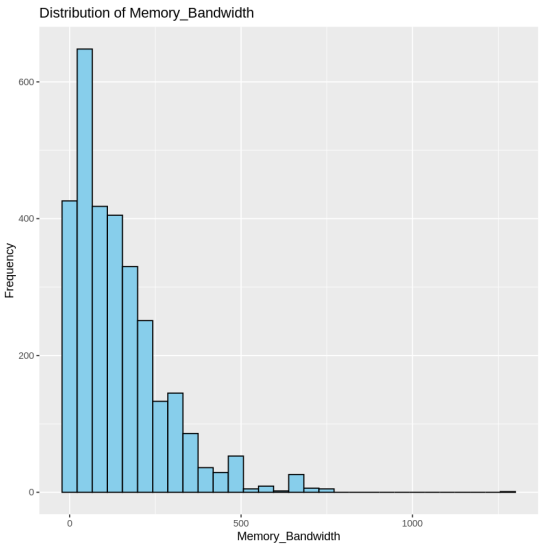

Distribution of Pixel_Rate
Frequency
Pixel_Rate
0
200
400
600
0
100
200


Distribution of ROPs
Frequency
ROPs
0
200
400
600
800
0
50
100
150
200


Distribution of TMUs
Frequency
TMUs
0
200
400
600
0
100
200
300
400

.



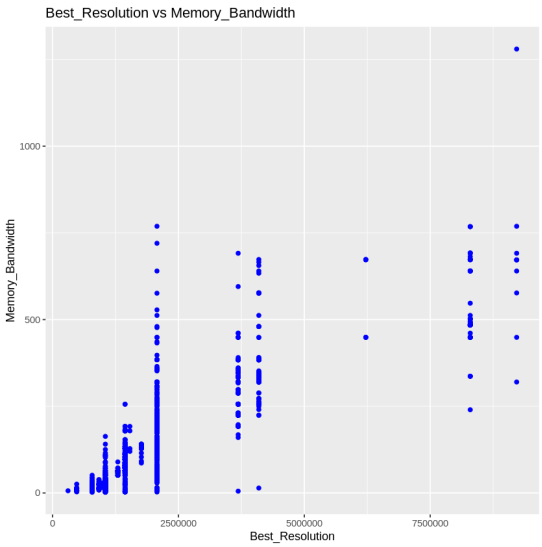


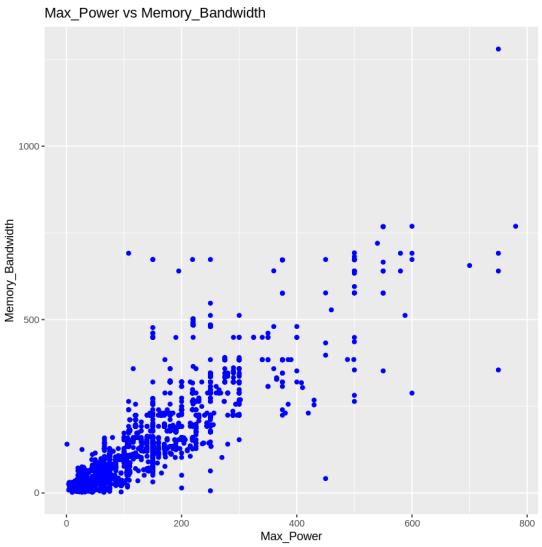
The histogram plots mostly display left-skewed distributions (except for the **Memory_Speed**, with many values concentrated on the left side of the plot and some higher values extending towards the right side.

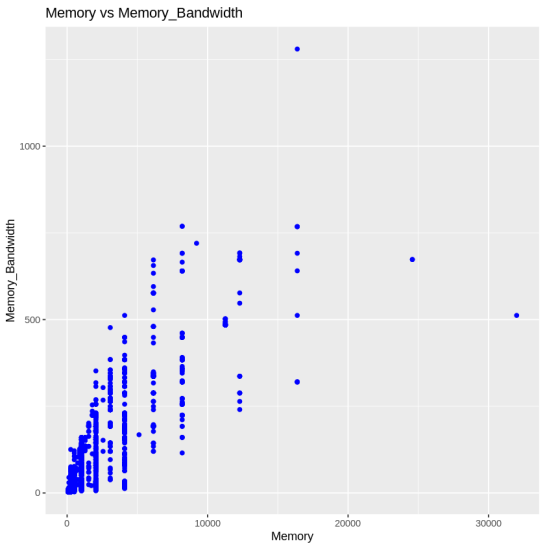
There are some outliers on the right side of the distribution, representing values with very high values that exceed the usual limits.

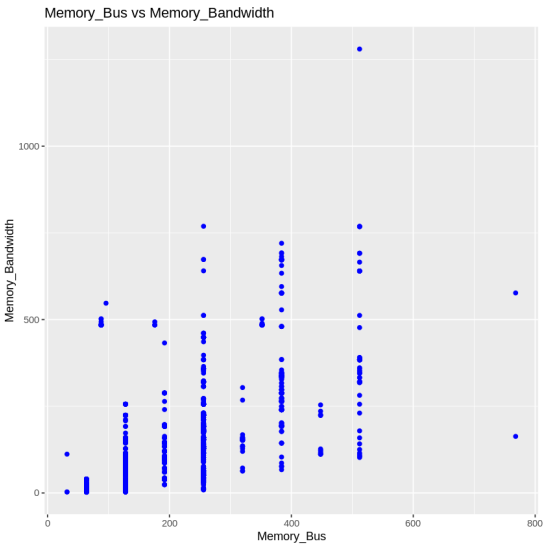
The distribution is unimodal (except for the **Memory_Speed**), indicating concentration of data in a specific range of memory bandwidth.

Next, we draw the scatter plot to display the relationship between 9 columns and **Memory_Bandwidth** column.


Best_Resolution vs Memory_Bandwidth
Memory_Bandwidth
1000
500
0
0
2500000
5000000
7500000
Best_Resolution

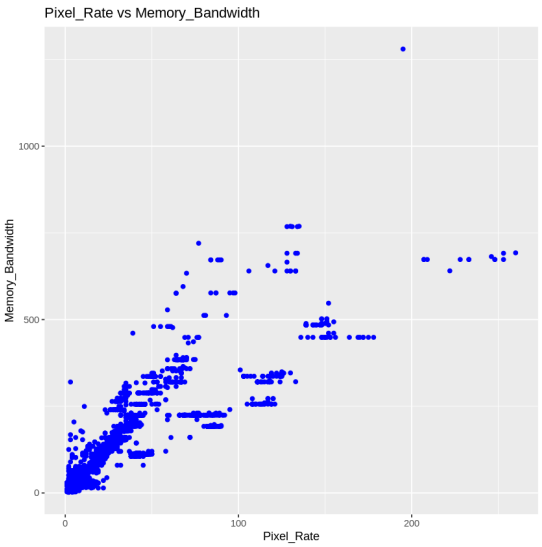

Max_Power vs Memory_Bandwidth
Memory_Bandwidth
1000
500
0
0
200
400
600
800
Max_Power

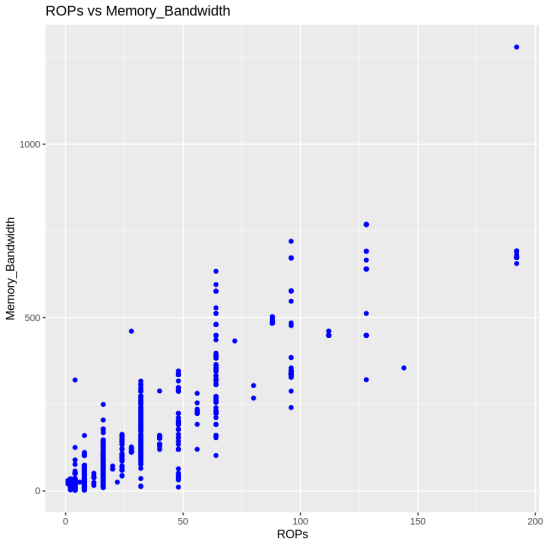

Memory vs Memory_Bandwidth
Memory_Bandwidth
1000
500
0
0
10000
20000
30000
Memory

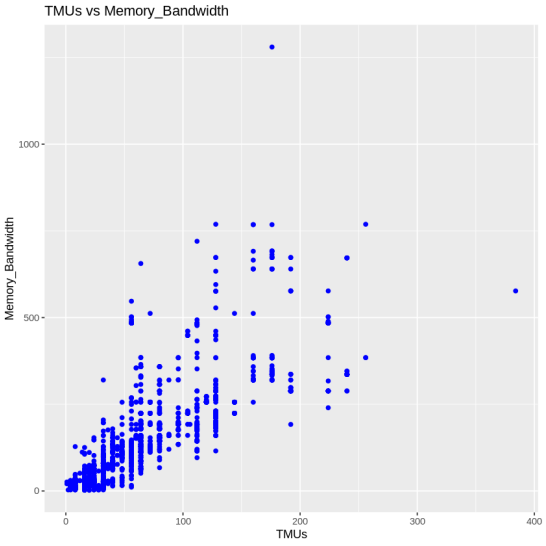

Memory_Bus vs Memory_Bandwidth
Memory_Bandwidth
1000
500
0
0
200
400
600
800
Memory_Bus

.

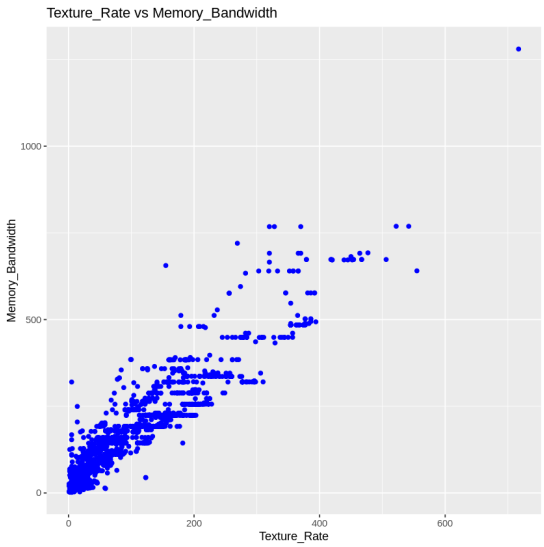

Memory_Speed vs Memory_Bandwidth
Memory_Bandwidth
Memory_Speed
0
500
1000
1500
2000


Pixel_Rate vs Memory_Bandwidth
Memory_Bandwidth
Pixel_Rate
0
100
200
500
1000


ROPs vs Memory_Bandwidth
Memory_Bandwidth
ROPs
0
50
100
150
200
500
1000


TMUs vs Memory_Bandwidth
Memory_Bandwidth
TMUs
0
100
200
300
400
500
1000

.



The scatter plot shows a positive correlation between **Memory_Bandwidth** and 9 features, as these features increase, **Memory_Bandwidth** tend to increase as well. The relationships appears to be moderately strong, indicating that there is a noticeable trend between them. However, there are also some variations.

Many relationships appear to be linear, with some outliers. These outliers may represent exceptional cases that deviate from the general trend.

6 Inferential statistics

First, we will remove rows where the **Memory_Bandwidth** values are greater than 500. This is because these rows represent an extremely small portion of the distribution, which could potentially introduce erroneous effects or distortions to our analysis. This approach helps to minimize the influence of outliers and ensures that our analysis is based on a more reliable dataset.

```
data_inferential <- data_inferential[data_inferential$Memory_Bandwidth <=
    500, ]
```

6.1 ANOVA

We need to convert data in **Memory_Bus** from numeric or character type to a factor. This is necessary as ANOVA requires categorical variables.

Now we performs an ANOVA to examine whether there are significant differences in **Memory_Bandwidth** among groups defined by **Memory_Bus** width.

```
data_inferential$Memory_Bus <- as.factor(data_inferential$Memory_Bus)
```

```
anova_result_memory_bus <- aov(Memory_Bandwidth ~ Memory_Bus, data = data
    _inferential)
```

After that, we have the result:

```
[1] "ANOVA Summary for Memory_Bus"
             Df   Sum Sq Mean Sq F value Pr(>F)
Memory_Bus    12 25056819 2088068   483.8 <2e-16 ***
Residuals   2943 12701049    4316
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

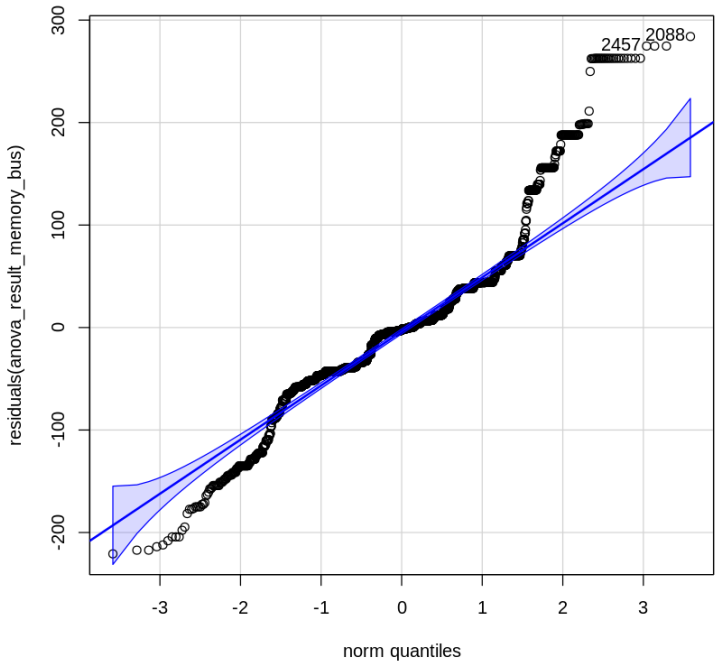
6.1.1 Shapiro - Wilk Test and Levene's Test

Now, we will check the assumptions necessary for the validity of ANOVA results. First, we generate a QQ plot for the residuals of the ANOVA model. Then we sequentially make a Shapiro-Wilk test for normality and a Levene's test for equality of variances. The Shapiro-Wilk test will specifically checks the normality of the residuals, while the Levene's test will check for homogeneity of variances across groups defined by **Memory_Bus**

```
qqPlot(residuals(anova_result_memory_bus))
shapiro_test_memory_bus <- shapiro.test(residuals(anova_result_memory_bus
    ))
levene_test <- leveneTest(Memory_Bandwidth ~ Memory_Bus, data = data_
    inferential)
```

After that, we have the result:

```
[1] "Shapiro-Wilk Test for Normality - Memory_Bus"

        Shapiro-Wilk normality test

data:  residuals(anova_result_memory_bus)
W = 0.91755, p-value < 2.2e-16

[1] "Levene's Test for Equality of Variances"
Levene's Test for Homogeneity of Variance (center = median)
       Df F value    Pr(>F)
group   12  38.627 < 2.2e-16 ***
      2943
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



Based on the results, the F-value is 483.8 and $Pr(> F)$ is nearly $0(< 2e-16)$, indicating that the difference between the considered groups of **Memory_Bus** is statistically significant. We can conclude that there is a significant difference in **Memory_Bandwidth** among the **Memory_Bus** groups.

The result of the Shapiro-Wilk test shows that the p-value is very small ($< 2.2e-16$), indicating that the residuals do not follow a normal distribution.

The result of Levene's Test for equality of variances is also very small ($<$ 2.2e-16), suggesting that there is no homogeneity of variances among the groups.

6.1.2 Kruskal-Wallis

Since the normality and homogeneity of variances are violated in one-way ANOVA, we should not reject the null hypothesis yet (can be type I error) and use other non-parametric test to further prove our point and reject the null hypothesis. In this project, we will use Kruskal-Wallis test.

```
kruskal_result_memory_bus <- kruskal.test(Memory_Bandwidth ~ Memory_Bus,
    data = data_inferential)
```

```
[1] "Kruskal-Wallis Test Summary for Memory_Bus"

        Kruskal-Wallis rank sum test

data:  Memory_Bandwidth by Memory_Bus
Kruskal-Wallis chi-squared = 1972.4, df = 12, p-value < 2.2e-16
```

Based on the result of the Kruskal-Wallis test, the very small p-value ($<$ 2.2e-16) indicates a significant difference in memory bandwidth between groups defined by the variable **Memory_Bus**. This suggests that the variable **Memory_Bus** significantly influences **Memory_Bandwidth**, leading us to reject the null hypothesis (H_0).

6.1.3 Dunn's Test

However, the Kruskal-Wallis test only indicates whether there is some difference among at least two groups, without specifying which pairs of groups differ. To gain more insight into this, we performed the Dunn's test. The Dunn's test is used to conduct pairwise comparisons between groups after detecting a general difference using the Kruskal-Wallis test.

```
dunn_result_memory_bus <- dunn.test(data_inferential$Memory_Bandwidth, g
    = data_inferential$Memory_Bus, method = "bonferroni")
print("Dunn Test Summary for Memory_Bus")
print(dunn_result_memory_bus)
```

```
  Kruskal-Wallis rank sum test

data: x and group
Kruskal-Wallis chi-squared = 1972.4326, df = 12, p-value = 0
```

Comparison of x by group
(Bonferroni)

Col Mean / Row Mean	128	176	192	256	32	320
176	-4.483221 0.0003*					
192	-11.38356 0.0000*	2.687220 0.2810				
256	-25.64622 0.0000*	2.203143 1.0000	-3.176850 0.0581			
32	1.040199 1.0000	3.727239 0.0075*	2.559210 0.4092	3.013253 0.1008		
320	-4.455407 0.0003*	2.456669 0.5469	-0.142463 1.0000	1.076388 1.0000	-2.482104 0.5094	
352	-7.729832 0.0000*	0.004651 1.0000	-4.552743 0.0002*	-3.789604 0.0059*	-4.405968 0.0004*	-3.745028 0.0070*
384	-24.84477 0.0000*	0.875373 1.0000	-9.320266 0.0000*	-8.886281 0.0000*	-4.138520 0.0014*	-4.123216 0.0015*
448	-3.706342 0.0082*	2.203008 1.0000	-0.392288 1.0000	0.537993 1.0000	-2.511340 0.4691	-0.231239 1.0000
512	-16.18291 0.0000*	0.666932 1.0000	-7.523368 0.0000*	-6.412880 0.0000*	-4.256160 0.0008*	-4.212206 0.0010*
64	14.10067 0.0000*	6.241785 0.0000*	19.43175 0.0000*	30.99189 0.0000*	0.506777 1.0000	8.565221 0.0000*

The chi-squared statistic for the Dunn test is 1972.433, indicating a significant difference among the groups.

Several pairwise comparisons have p-values less than 0.05, indicating significant differences between those groups. Adjusted p-values are also provided to account for multiple comparisons. If any adjusted p-values are less than 0.05, it suggests significant differences after adjusting for multiple testing.

Based on the table x in the Kruskal-Wallis test results, we can see that there are several pairwise comparisons between groups with p-values less than 0.05, indicating significant differences between those groups. Suppose we want to compare group 128 with group 192. In the table, we see that the p-value for this comparison is 0.0003. This p-value is smaller than the significance threshold of 0.05, indicating a significant difference between these two groups.

However, when adjusting for multiple comparisons, the p-values (P.adjusted) may increase. If the adjusted p-value remains below 0.05, then the difference is still considered significant after correcting for multiple comparisons. For example,

suppose the adjusted p-value for the comparison between group 128 and group 192 is 0.001, still below the threshold of 0.05. This implies that there is a significant difference between group 128 and group 192 even after adjusting for multiple comparisons.

In summary, the Dunn test reveals significant differences between certain pairs of groups defined by the variable **Memory_Bus**, further supporting the rejection of the null hypothesis obtained from the Kruskal-Wallis test.

6.2 Multiple Linear Regression

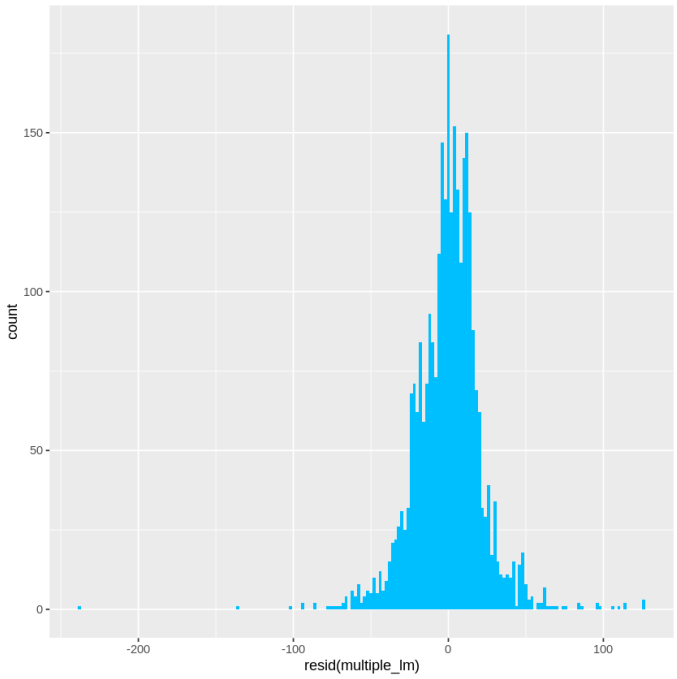
Furthermore, we will use other model (Multiple Linear Regression) for this topic. Firstly, we try to performs multiple linear regression using **Memory_Bandwidth** as dependent variable and the 9 columns as independent variables.

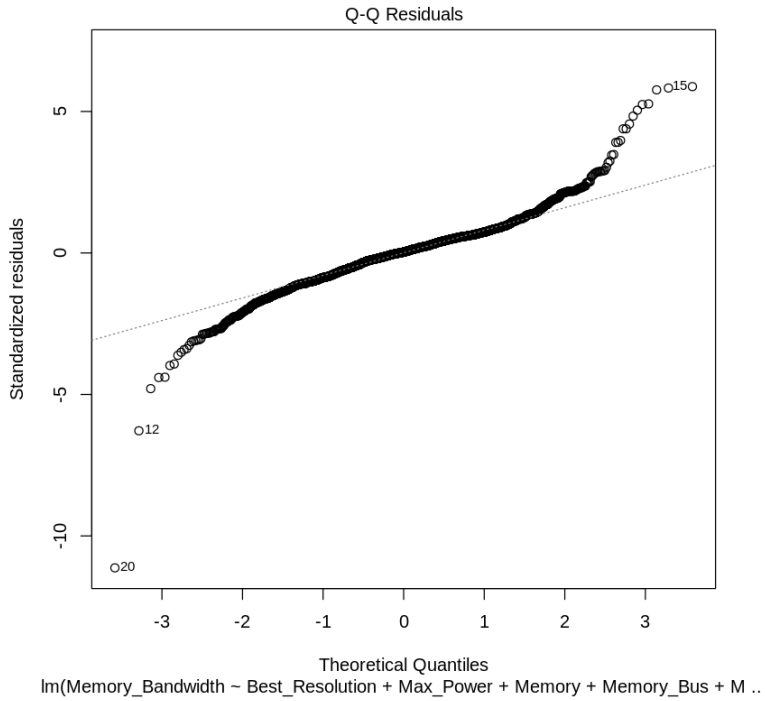
```
data_inferential <- read.csv("data_inferential.csv")
multiple_lm <- lm(Memory_Bandwidth ~ Best_Resolution + Max_Power + Memory
    + Memory_Bus + Memory_Speed + Pixel_Rate + ROPs + TMUs + Texture_Rate
   , data = data_inferential)
summary(multiple_lm)
```

```
Call:
lm(formula = Memory_Bandwidth ~ Best_Resolution + Max_Power +
    Memory + Memory_Bus + Memory_Speed + Pixel_Rate + ROPs +
    TMUs + Texture_Rate, data = data_inferential)

Residuals:
     Min       1Q   Median       3Q      Max
-238.455  -11.579    0.664   11.825  126.418

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)     -5.672e+01  1.870e+00 -30.332  < 2e-16 ***
Best_Resolution  5.611e-06  7.515e-07   7.467 1.07e-13 ***
Max_Power        1.894e-01  1.008e-02  18.796  < 2e-16 ***
Memory           1.517e-03  3.255e-04   4.659 3.31e-06 ***
Memory_Bus       2.715e-01  8.094e-03  33.546  < 2e-16 ***
Memory_Speed     3.192e-02  1.504e-03  21.223  < 2e-16 ***
Pixel_Rate      -1.009e+00  5.730e-02 -17.611  < 2e-16 ***
ROPs             1.581e+00  7.223e-02  21.884  < 2e-16 ***
TMUs            -5.529e-01  2.271e-02 -24.344  < 2e-16 ***
Texture_Rate     1.057e+00  2.066e-02  51.162  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Now we will plot a histogram of the residuals from the regression model. The histogram helps check the distribution of residuals, part of diagnosing the model to see if linear regression assumptions are met. We also create a Q-Q plot (for checking the normal distribution of residuals).

```
ggplot(multiple_lm, aes(x = resid(multiple_lm))) +
geom_histogram(binwidth = 2, fill = "deepskyblue")
plot(multiple_lm, which = 2)
```





With the Q-Q plot, we can see that the distribution of residuals is not normal, so we will try to solve this problem. Firstly, we decrease the number of columns that have strong collerations with each other based on our correlation heatmap.

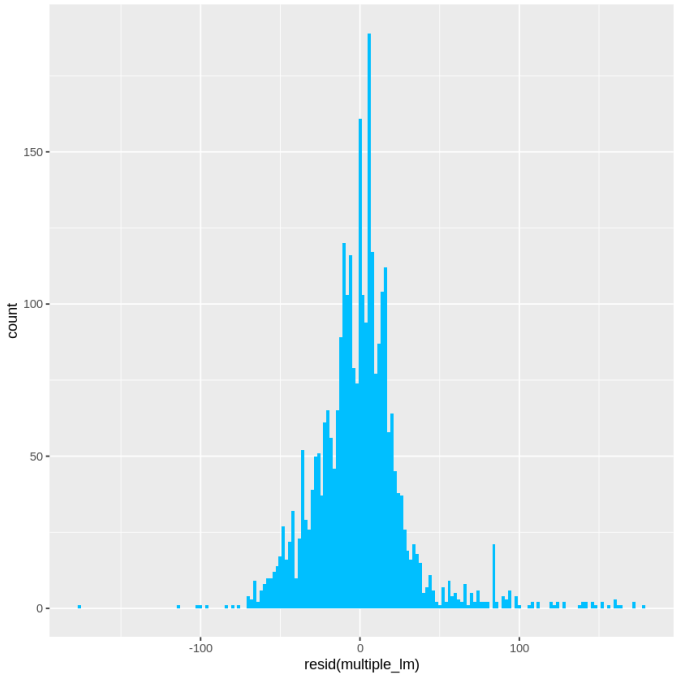
After several trials, we now use only **Memory_Bus**, **Memory_Speed**, **Texture_Rate** as independent variables.

```
data_inferential <- read.csv("data_inferential.csv")
multiple_lm <- lm(Memory_Bandwidth ~ Memory_Bus + Memory_Speed + Texture_
    Rate, data = data_inferential)
summary(multiple_lm)
```

```
Call:
lm(formula = Memory_Bandwidth ~ Memory_Bus + Memory_Speed + Texture_Rate,
    data = data_inferential)

Residuals:
     Min       1Q   Median       3Q      Max
-175.465  -15.452    0.166   12.922  177.462

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -32.099094   2.035681 -15.768   <2e-16 ***
Memory_Bus     0.316368   0.006055  52.247   <2e-16 ***
Memory_Speed   0.015835   0.001802   8.786   <2e-16 ***
Texture_Rate   1.080210   0.010567 102.220   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 29.76 on 2952 degrees of freedom
Multiple R-squared:  0.9307,    Adjusted R-squared:  0.9307
F-statistic: 1.322e+04 on 3 and 2952 DF,  p-value: < 2.2e-16
```

We will plot a histogram of the residuals from the regression model to check the distribution of residuals in order to see if linear regression assumptions are met.

```
ggplot(multiple_lm, aes(x = resid(multiple_lm))) +
geom_histogram(binwidth = 2, fill = "deepskyblue")
```



Now we will try to find data outlier and reevaluate the model after removing those outliers. Based on the histogram plot, we remove residuals greater than 50 or less than -50 then rebuild the regression model with the cleaned dataset.

```r
data_inferential <- read.csv("data_inferential.csv")
multiple_lm <- lm(Memory_Bandwidth ~ Memory_Bus + Memory_Speed + Texture_Rate, data = data_inferential)
residuals_values <- residuals(multiple_lm)
outlier_indices <- which(residuals_values > 50 | residuals_values < -50)
data_cleaned <- data_inferential[-outlier_indices, ]
multiple_lm_cleaned <- lm(Memory_Bandwidth ~ Memory_Bus + Memory_Speed + Texture_Rate, data = data_cleaned)
summary(multiple_lm_cleaned)
num_removed <- length(outlier_indices)
num_remaining <- nrow(data_cleaned)
cat("Number of outliers removed:", num_removed, "\n")
cat("Remaining number of observations after removing outliers:", num_remaining, "\n")
```

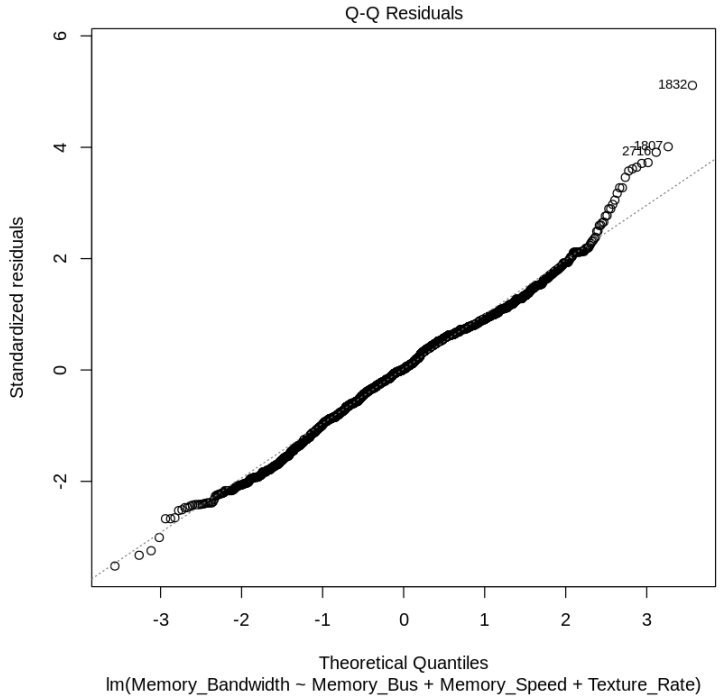
```
Call:
lm(formula = Memory_Bandwidth ~ Memory_Bus + Memory_Speed + Texture_Rate,
    data = data_cleaned)

Residuals:
    Min      1Q  Median      3Q     Max
-64.738 -11.809   0.407  12.575  93.471

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -48.653420   1.419153  -34.28   <2e-16 ***
Memory_Bus     0.357705   0.004456   80.28   <2e-16 ***
Memory_Speed   0.030411   0.001264   24.06   <2e-16 ***
Texture_Rate   0.922263   0.009063  101.76   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18.46 on 2729 degrees of freedom
Multiple R-squared:  0.9658,    Adjusted R-squared:  0.9657
F-statistic: 2.566e+04 on 3 and 2729 DF,  p-value: < 2.2e-16
Number of outliers removed: 223
Remaining number of observations after removing outliers: 2733
```

We can see that there are 223 outliers are removed, and 2733 remaining, so so the data is large enough for our model. After outlier removal, residual standard error has decreased from 29.76 to 18.46, indicating that the model after outlier removal fits the data better. R-squared: Increased from 0.9307 to 0.9658, indicating that the new model explains 96.58% of the variation in **Memory_Bandwidth**,

significantly higher than the initial model.

We now create a Q-Q plot for checking the normal distribution of residuals.

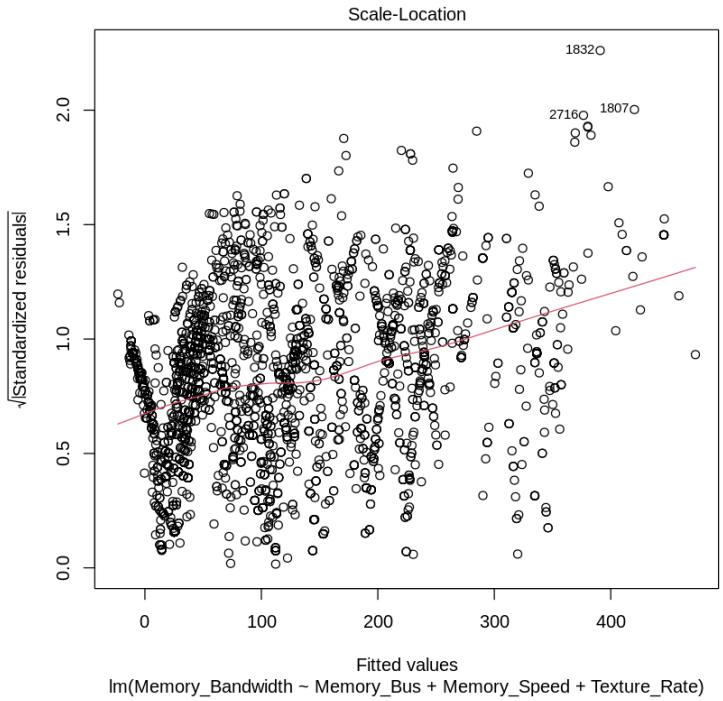
```
plot(multiple_lm_cleaned, which = 2)
```



Now we can consider residuals has normal distribution despite some outliners.

Next, we will make a Scale-Location plot for checking the homoscedasticity of residuals' variance.

```
plot(multiple_lm_cleaned, which = 3)
```



On the plot, we can see data points scattered around the red line, there are no prominent patterns or trends with varying coefficients of dispersion. Tt suggests that the variance of the residuals is constant, supporting the homoscedasticity assumption.

We can write the multiple linear regression equation as follows:

$$Y = 48.653420 + 0.357705 * X_1 + 0.030411 * X_2 + 0.922263 * X_3 \tag{9}$$

where

- $Y =$ **Memory_Bandwidth**
- $X_1 =$ **Memory_Bus**
- $X_2 =$ **Memory_Speed**
- $X_3 =$ **Texture_Rate**

6.3 Random Forest

Now we will use other model (Random Forest) in order to compare with other models and also understand more about the problem.

```
1 data_inferential <- read.csv("data_inferential.csv")
```

```
model.rfr <- randomForest(formula = Memory_Bandwidth ~ Best_Resolution +
    Max_Power + Memory + Memory_Bus + Memory_Speed + Pixel_Rate + ROPs +
    TMUs + Texture_Rate, data = data_inferential, ntree = 100)
print(model.rfr)
```

```
Call:
 randomForest(formula = Memory_Bandwidth ~ Best_Resolution + Max_Power +
               Type of random forest: regression
                     Number of trees: 100
No. of variables tried at each split: 3

          Mean of squared residuals: 152.9333
                    % Var explained: 98.8
```

Based on the result, we can see that the model explains around 98.8% of the variance in the data. This indicates the model's ability to explain a large portion of the variation in the target **Memory_Bandwidth**

Now we want to evaluate the model performance. We have to compare the real values of **Memory_Bandwidth** and the predicted value to know if the model was good or not by calculating the Accuracy and the Mean Absolute Error (MAE).

```
comtab.rfr <- data_inferential['Memory_Bandwidth']
comtab.rfr['Memory_Bandwidth_predicted'] <- as.data.frame(predict(model.
    rfr, newdata = data_inferential), row.names = NULL)
  accuracy <- sum(1-abs(comtab.rfr$Memory_Bandwidth_predicted- comtab.rfr
      $Memory_Bandwidth) / comtab.rfr$Memory_Bandwidth) / nrow(comtab.rfr)
  MAE <- sum(abs(comtab.rfr$Memory_Bandwidth_predicted- comtab.rfr$Memory
      _Bandwidth)) / nrow(comtab.rfr)
  print(paste("Accuracy:", accuracy))
  print(paste("MAE:", MAE))
```

```
[1] "Accuracy: 0.967484260325928"
[1] "MAE: 1.94952519484957"
```

Overall, the model seems to have performed well, with a high level of accuracy (about 96.75%) and a relatively low MAE(about 1.95) , indicating that it is effective in predicting Memory_Bandwidth based on the input features.

Furthermore, we will calculate R-squared on testing data to confirm the performance of the model.

```
r2_test <- cor(comtab.rfr$Memory_Bandwidth, comtab.rfr$Memory_Bandwidth_
    predicted)^2
print(r2_test)
```

```
[1] 0.9977128
```

The value of R^2 is approximately 0.9977128. This implies that the model can explain about 99.77% of the variance in the dependent variable **Memory_Bandwidth** based on the predicted values **Memory_Bandwidth_predicted**. This indicates that the Random Forest model we have built has a high ability to explain much of the variation in **Memory_Bandwidth** within our data.

7 Conclusion

In general, we have conducted several test models and received many positive outcomes that provide strong proof to reject the null hypothesis that there is no significant relationship between these categories, therefore confirming our prediction that the GPU memory bandwidth strongly depends its characteristics.

With the topic **Evaluate the impact of GPU's characteristics on its Memory Bandwidth** uses the R programming language to process statistical data and realize the model linear regression model, our team had a more intuitive view of how to extract data, process and analyze raw data, turning them into valuable data sources long-term, or better yet, being able to generalize the general situation and make predictions about the data set.

8 Code

Code link: https://colab.research.google.com/

References

[1] Brian Beers, "P-Value: What It Is, How to Calculate It, and Why It Matters" https://www.investopedia.com/terms/p/p-value.asp

[2] Clay Ford, "Understanding QQ Plots" https://library.virginia.edu/data/articles/understanding-q-q-plots

[3] IBM "What is random forest?" https://www.ibm.com/topics/random-forest

[4] Jason Fernando, "R-Squared: Definition, Calculation Formula, Uses, and Limitations" https://www.investopedia.com/terms/r/r-squared.as

[5] LINEAR MODELS IN STATISTICS – Second Edition, Alvin C. Rencher and G. Bruce Schaalje

[6] Rebecca Bevans, "Multiple Linear Regression" https://www.scribbr.com/statistics/multiple-linear-regression/

[7] Rosane Rech, "ONE-WAY ANOVA STEP-BY-STEP" https://statdoe.com/one-way-anova-r-tutorial/

[8] Statistics Kingdom, "Shapiro Wilk Test" https://www.statskingdom.com/doc_shapiro_wilk.html

[9] Zach Bobbitt, "Dunn's Test for Multiple Comparisons" https://www.statology.org/dunns-test/