A black background with a black square

Description automatically generated with medium confidence

Bachelor of Computer Science (Hons)

Bachelor of Software Engineering (Hons)

**Introduction to Data Science**

**XBDS2014/N**

Prepared by Ts. Dr. Law Foong Li

Semester May 2023

A close up of text on a white background

Description automatically generated

A black background with a black square

Description automatically generated with medium confidenceA close up of text on a white background

Description automatically generated

SCCM/CSForm/Rev. Ver 2.5 Date: 28/3/2023

FOUNDATION DIPLOMA  DEGREE  MASTER

*Please tick* ✓ *or click if using MS WORD*

**Marks Breakdown**

Signature: **Total**

**For office use only** – Lecturer comments (if applicable)

*I/We declare that:*

* *This assignment is my/our own original work, except where I/we have appropriately cited the original source.*
* *This assignment or parts of it has not previously been submitted for assessment in this or any other subject.*
* *I/We allow the assessor of this assignment to test any work submitted by me/us, using text comparison software for plagiarism.****(For more information, Please read the Academic Integrity Guidelines)***

Assignment Coversheet **Please complete all details required clearly.** For softcopy submissions,please ensure this cover sheet is included at the start of your document or in the file folder.

**Assignment & Course Details:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject Code:** XBDS2014/N | | **Subject Name** *(e.g. Fundamentals of Computing*)**:** Introduction to Data Science | |
| **Course** *(e.g. Bachelor in Computing)* : Bachelor of Software Engineering  Bachelor of Computer Science | | | |
| **Lecturer Name:** Ts. Dr. Law Foong Li | | | |
| **Assessment Due Date:** *(dd/mm/yy)* | 9 August 2023 | **Assessment Title:** | Group Assignment |

**Name :  
Student ID:  
Email :  
Mobile No: Signature:  
Date:**

**Name :  
Student ID:  
Email :  
Mobile No: Signature:  
Date:**

**Name :  
Student ID:  
Email :  
Mobile No: Signature:  
Date:**

**Name : Hui Keng Fai  
Student ID: 0133848  
Email : 0133848@student.uow.edu.my  
Mobile No: 017 622 7175 Signature: kf  
Date: 05/08/2023**

A black and white text

Description automatically generated

**Name : Ng Hang Ee  
Student ID: 0130836  
Email : 0130836@student.uow.edu.my  
Mobile No: 0127022725 Signature:**  **Date: 07/08/2023**

**Name :Lim Yan Bin  
Student ID:0131120  
Email :0131120@student.uow.edu.my  
Mobile No:011-36878262 Signature:yb  
Date:9/8/2023**

Table of Contents

[1.0 Introduction 5](#_Toc142481461)

[1.1 Describe problem to be solved 5](#_Toc142481462)

[1.2 Discuss the limitations of existing solutions to the problem 7](#_Toc142481463)

[1.3 Explains your solution or approach 7](#_Toc142481464)

[1.4 Related studies 8](#_Toc142481465)

[1.5 Hypothesis and research questions 9](#_Toc142481466)

[2.0 Methodology 10](#_Toc142481467)

[2.1 Describes tools used with justification 10](#_Toc142481468)

[2.2 Describes techniques used with justification 12](#_Toc142481469)

[2.3 Describes algorithms used with justification 13](#_Toc142481470)

[2.3.1 Support Vector Machine (SVM) 13](#_Toc142481471)

[2.3.2 Polynomial Regression 13](#_Toc142481472)

[2.3.3 Gradient boosting 14](#_Toc142481473)

[2.3.4 Lasso Regression 14](#_Toc142481474)

[3.0 Implementation 16](#_Toc142481475)

[3.1 Data cleaning and data preprocessing 16](#_Toc142481476)

[3.2 Gradient Boosting model 17](#_Toc142481477)

[3.3 Lasso Regression model 18](#_Toc142481478)

[3.4 Polynomial regression model 19](#_Toc142481479)

[3.5 Support vector machine model 20](#_Toc142481480)

[4.0 Result 21](#_Toc142481481)

[5.0 Discussion 26](#_Toc142481482)

[5.1 Briefly describe the board research area and narrow it down to your particular focus. 26](#_Toc142481483)

[5.2 Comparison between our findings and previous studies 29](#_Toc142481484)

[6.0 Conclusion 32](#_Toc142481485)

[6.1 Summary of the hypothesis and purpose of the study 32](#_Toc142481486)

[6.2 Discuss potential future enhancements 33](#_Toc142481487)

[7.0 References 34](#_Toc142481488)

[8.0 Appendix 35](#_Toc142481489)

[8.1 Data cleaning and data preprocessing 35](#_Toc142481490)

[8.2 Gradient Boosting model 38](#_Toc142481491)

[8.3 Lasso Regression model 45](#_Toc142481492)

[8.4 Polynomial regression model 50](#_Toc142481493)

[8.5 Support vector machine model 55](#_Toc142481494)

# 1.0 Introduction

This study aims to address the problems faced by a newly founded custom laptop store in Malaysia. They are relatively new to the customized laptop industry and could benefit from more numerical insights to help attract customers and maximize profitability. As a newcomer in the customized laptop industry, the company recognizes the potential of leveraging data-driven insights to improve customer engagement and increase profitability.

## 1.1 Describe problem to be solved

Several problems have come to the forefront, one of which is inconsistent pricing decisions. The company struggles with setting optimal prices on laptops, sometimes leading to underpricing or overpricing models. This inability to find the right pricing sweet spots leads to fluctuating profit margins and impacts overall profitability. Next, another problem faced is unclear product positioning. The lack of clear differentiation between laptop models in terms of pricing and respective features is an issue as customers may find it challenging to understand the differences and value propositions among various laptop models offered by the company. Another issue arising from inconsistent pricing decisions is the company struggles with setting optimal prices on laptops, sometimes leading to underpricing or overpricing models. This inconsistent pricing leads to fluctuating profit margins and impacts overall profitability. Related to that is also procurement and inventory management problems. Misaligned pricing contributes to overstocking or stockouts of certain laptop models.

Several other studies in the area of laptop price prediction for the purpose of business analytics and pricing optimization have explored similar data-driven methodologies to enhance companies’ decision making and profitability. A study by Kolla (2016) achieved 81% precision in laptop price prediction using multiple linear regression. This study particularly highlighted features like processor speed and brand in predicting laptop prices while also discovering that Random Forest Regression(RFR) and Gradient Boosting Regression(GBR) outperformed other algorithms. Conversely, they found that features like battery life, weight and graphics card had a more modest influence on the prediction of prices. Next, a recent study by Siburian (2022) used XGBoost, a gradient boosting algorithm for supervised learning, also a decision tree algorithm to help users predict laptop prices in order to prevent buying laptops at overpriced rates. Finally, a study by (Jean & Himpens, 2019) compared predictive power of using traditional hedonic regressions with machine learning predictive models to predict prices of laptops and suggested that random forests and Lasso predictors could be a promising way to predict prices and highlighted the need for further research.

## 1.2 Discuss the limitations of existing solutions to the problem

Dataset suitability and quality: The most obvious would be the dataset itself, how accurate is it? Is it applicable to the Malaysian market? Our dataset is from India.

Rapidly changing tech landscape: The technological landscape is also changing rapidly, with new tech and advancements being introduced regularly. The regression model’s prediction ability is limited by how fast new tech is introduced. These datasets quickly become outdated and hence have a short cutoff date.

Nuanced factors besides these numerical features : Regression models can provide insights into the relationships between laptop specifications, we can see everything in numbers, but the psychology of purchasing isn’t entirely number dependent. Many other factors are involved too, brand reputation, aesthetics, ease of use like battery life, laptop size, etc.

## 1.3 Explains your solution or approach

In our study, to address the previously discussed problems, our study aims to perform laptop price prediction using several machine learning techniques, namely Support Vector Machine (SVM), polynomial regression, gradient boosting, and lasso regression algorithms. The predictive model will encompass a comprehensive set of laptop specifications, using features such as laptop type, Dedicated Graphic Memory Capacity, SSD, RAM (in GB), Expandable Memory, touchscreen, screen Size (in inch), screen\_resolution, Storage space, CPU\_ranking, Gpu\_benchmark, ram\_type tokenized, and Refresh Rate.

## 1.4 Related studies

Compared to the study by Kolla (2016), the features used in our study is similar, we did not include the following features: IPS Display and Touch Screen. The algorithms by that study is different from our study, they used the C4.5 algorithm which is a decision tree classifier, Random Forest Regression (RFR), Gradient Boosting Regression(GBR), Support Vector Regression (SVR) which is used for categorical target variables, K-Nearest Neighbours Regression (KNN), and Artificial Neural Networks (ANN).

Compared to the study by Siburian (2022), after feature selection they only used screen resolution, CPU, Ram, Memory and Weight(kg). However, through their feature engineering phase, they created more features from the existing datasets, for instance, the CPU column was divided into 2 new columns, which were CPU Brand and CPU Clock. Screen resolution became resolution, screen type and touch screen. This was to help uncover hidden patterns in their dataset and improve the predictive power of machine learning. They only used a 80/20% training split while our study used both 80/20% and 70/30%.

The combination of these features within the predictive machine learning algorithm will generate well-informed price forecasts. Accurate price prediction can be a strategic tool to aid in demand forecasting and inventory optimization, encouraging efficient procurement and a well-balanced stock inventory. Furthermore, accurate price prediction enables the company to easily establish different tiers based on the predicted pricing, and help establish a clear product hierarchy, which in turn results in clear value propositions of each laptop and help customers make more informed decisions with confidence.

Within the confines of our solution, several limitations were identified, such as the suitability and applicability of the dataset itself when used in the Malaysian market. Given that our dataset originates from Indonesia, sufficient scrutiny should be placed to assess the dataset’s cross-border suitability and representativeness to ensure precise price predictions in Malaysia. Next, another constraint is the rapid evolution of technology. The models’ predictive efficacy may be affected by the fast emergence of new technological advancements in the laptop industry. Finally, while numerical data of laptop specifications are invaluable and insightful, we should consider that the act of purchasing laptops is also influenced by multifaceted considerations such as brand reputation, aesthetic appeal, ease of use and even the physical size of the laptop.

## 1.5 Hypothesis and research questions

Hypothesis 1: There is a positive correlation between laptop specifications (such as RAM, storage capacity, and display resolution) and performance benchmark rankings.

Research Question 1: What is the relationship between specific laptop specifications, such as RAM, storage capacity, display resolution and performance benchmark rankings?

Hypothesis 2: Laptops with higher performance rankings will have higher prices in the market.

Research Question 2: Is there a significant price difference between laptops with different performance benchmark rankings?

# 2.0 Methodology

## 2.1 Describes tools used with justification

Jupyter Notebook is an open source application that allows any computer science expert to write and execute code in a more dynamic and iterative manner. The application is supported by the Anaconda platform, which is a prepackaged distribution of python which contains a full set of Python modules and libraries to conduct all sorts of data science related tasks. My team chose to use Jupyter Notebook as our models training environment for several reasons.

Firstly, Jupyter Notebook supports the inclusion of markdown cells, allowing my team to document and explain each part of the code, which is particularly helpful when the tasks of model training are distributed to each team member. It saves us from wasting time and resources for using other applications to explain the use of the code.

Secondly, Jupyter Notebook offers the flexibility to run code in cells, making it easier to test and debug specific sections of code without having to rerun the entire program. While training a model, we need to perform steps like data cleaning, feature scaling, feature selection, data splitting etc. Each of the steps can be done in a separate cell to prevent messy structure if all the codes are mixed. This feature is especially valuable when working with large datasets or complex models.

Additionally, Jupyter Notebook supports powerful programming languages, including Python, which is widely used in the field of data analysis and machine learning. This versatility allows me to leverage the rich ecosystem of Python libraries and tools, such as NumPy and Pandas, which are essential for handling and analyzing data, as well as building regression models.

Considering these advantages, Jupyter Notebook proved to be a suitable and efficient tool for implementing the regression model using Python for analyzing the dataset to predict the price and rank.

## 2.2 Describes techniques used with justification

Regression is a suitable technique that my team chose for predicting the price and rank of computers because it allows us to examine the relationship between features (such as CPU, GPU, RAM, etc.). This is essential as we are interested in estimating precise numerical values rather than classifying into discrete categories/values (classification technique).

Regression is a widely used technique in the field of machine learning for predicting continuous outcomes. The techniques offer powerful functions in handling different types of regression problems. No matter if it's a simple linear regression or more advanced algorithms like decision trees, random forests, or gradient boosting. Which also allows us to determine which features matter most, can be ignored, and how they influence each other in order to generate the outcomes.

By employing regression, we can leverage its strengths in exploring the relationships between independent variables on a dependent variable, with the assistance of various regression algorithms.

## 2.3 Describes algorithms used with justification

We are using 4 algorithms for this project they are: Support Vector Machine (SVM), Polynomial Regression, Gradient boosting, and Lasso Regression.

### 2.3.1 Support Vector Machine (SVM)

SVM is a popular and widely used classification algorithm in supervised learning, although it can be used in regression as well. In SVM there is hyperplane (a boundary that splits the data), support vectors (data point that is nearest to the hyperplane) and margin (space between the support vectors from the classes and in between there will be the hyperplane). To place the hyperplane correctly the margin has to be as large as possible (maximize margin) and the position of the support vectors are the key to decide where the hyperplane places as well.

Above are linear SVM and there is another type of SVM called non-linear SVM where the classes can’t be separated linearly. In real-world, not every time we have data points can be separated linearly, so we use some advanced techniques like kernel tricks to do the classification for us.

### 2.3.2 Polynomial Regression

"Polynomial regression is a type of regression analysis used to model the relationship between the independent variable (or predictor) and the dependent variable (or target) when the relationship is not linear" (Hastie et al., 2009). It allows for capturing more complex and curved relationships by introducing higher-order polynomial terms. In polynomial regression, the model equation includes polynomial terms of the independent variable.

It's important to note that using very high-degree polynomials can lead to overfitting, where the model fits the training data too closely and fails to generalize well to new data.

### 2.3.3 Gradient boosting

"Gradient Boosting is a powerful ensemble learning technique that combines the predictions of multiple weak learners (typically decision trees) to create a strong predictive model" (Brownlee, 2021). It's often chosen for its high predictive accuracy, as it corrects errors made by previous models iteratively, reducing both bias and variance and leading to better generalization on new data (Géron, 2019). It's particularly effective in capturing complex non-linear relationships between features and the target variable, making it suitable for problems with intricate patterns. Gradient Boosting provides a measure of feature importance, which aids in identifying influential features and understanding the underlying data. It's robust to outliers and noise, handles missing data well, and supports both regression and classification tasks. Libraries like XGBoost, LightGBM, and CatBoost offer efficient implementations that make Gradient Boosting easy to use. However, it has longer trained times compared to simpler algorithms and requires careful tuning to prevent overfitting. Overall, Gradient Boosting, with its state-of-the-art performance and flexibility, is a popular choice for improving model accuracy and handling complex relationships in machine learning tasks.

### 2.3.4 Lasso Regression

"Lasso regression, also known as L1 regularization, is a type of linear regression that can help mitigate overfitting and perform feature selection" (Hastie et al., 2009). Lasso regression introduces a penalty term that encourages the coefficients of less important features to become exactly zero, effectively removing them from the model (Friedman et al., 2010). This property makes Lasso regression useful when dealing with datasets containing many irrelevant or redundant features.

In cases where you suspect that only a subset of features is relevant to the target variable, Lasso regression can automatically identify and select those features, leading to a simpler and more interpretable model (Brownlee, 2020). Lasso regression is particularly effective when dealing with high-dimensional data, as it helps prevent multicollinearity and improves the model's generalization performance.

# 3.0 Implementation

Refer to the appendix for the codes

## 3.1 Data cleaning and data preprocessing

1. Import library
2. Import dataset
3. Display dataset details
4. Missing data handling
5. Duplicate data handling
6. Deriving new attribute, convert original variable into meaningful variable
7. Convert records of variable to proper values that would improve model accuracy
8. Features selection, select necessary features and drop non-related features from the dataset
9. Exploratoty data analysis, discover the relationship between dependent variable and other features using heatmap
10. Export cleaned dataset to device and the dataset is prepared to use for model training

## 3.2 Gradient Boosting model

1. Import library
2. Import cleaned dataset
3. Select Independent and dependent variables
4. Data Spliting (70/30) -70% training set and 30% for testing set
5. Data Spliting (80/20) -80% training set and 20% for testing set
6. Look for Iteration (Epochs) that give minimum error using a loop with 300 iterations (70/30)
7. Model training (70/30)
8. Evaluation of model accuracy (70/30)
9. Plot predicted result (70/30)
10. Look for Iteration (Epochs) that give minimum error using a loop with 300 iterations (80/20)
11. Model training (80/20)
12. Evaluation of model accuracy (80/20)
13. Plot predicted result (80/20)

## 3.3 Lasso Regression model

1. Import library
2. Import cleaned dataset
3. Select Independent and dependent variables
4. Data Spliting (70/30) -70% training set and 30% for testing set
5. Data Spliting (80/20) -80% training set and 20% for testing set
6. Model training (70/30)
7. Evaluation of model accuracy (70/30)
8. Plot predicted result (70/30)
9. Model training (80/20)
10. Evaluation of model accuracy (80/20)
11. Plot predicted result (80/20)

## 3.4 Polynomial regression model

1. Import library
2. Import cleaned dataset
3. Select Independent and dependent variables
4. Data Spliting (70/30) -70% training set and 30% for testing set
5. Data Spliting (80/20) -80% training set and 20% for testing set
6. Model training (70/30)
7. Evaluation of model accuracy (70/30)
8. Plot predicted result (70/30)
9. Model training (80/20)
10. Evaluation of model accuracy (80/20)
11. Plot predicted result (80/20)

## 3.5 Support vector machine model

1. Import library
2. Import cleaned dataset
3. Select Independent and dependent variables
4. Data Spliting (70/30) -70% training set and 30% for testing set
5. Data Spliting (80/20) -80% training set and 20% for testing set
6. Model training (70/30)
7. Evaluation of model accuracy (70/30)
8. Plot predicted result (70/30)
9. Model training (80/20)
10. Evaluation of model accuracy (80/20)
11. Plot predicted result (80/20)

# 4.0 Result

1. Accuracy results for models with 70% for training set, 30% for testing set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | MSE | RMSE | MAE | MAPE |
| Gradient Boosting | 1141003.78 | 1068.18 | 515.64 | 0.12 |
| Lasso regression | 1479445.62 | 1216.32 | 660.25 | 0.16 |
| Polynomial regression | 1603742.94 | 1266.39 | 661.2 | 0.17 |
| Support Vector Machine | 1376708.51 | 1173.33 | 540.55 | 0.12 |

Table 1.0

1. Accuracy results for models with 80% for training set, 20% for testing set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | MSE | RMSE | MAE | MAPE |
| Gradient Boosting | 1436138.45 | 1198.39 | 519.51 | 0.12 |
| Lasso regression | 1777285.45 | 1333.15 | 699.58 | 0.18 |
| Polynomial regression | 1675790.61 | 1294.52 | 648.59 | 0.16 |
| Support Vector Machine | 1773492.31 | 1331.73 | 559.49 | 0.12 |

Table 2.0

The Evaluation Metrics used are:

mse = Mean Squared Error

rmse = Root Mean Squared Error

mae = Mean Absolute Error

mape = Mean Absolute Percentage Error

1. Predict and actual value’s graph for Gradient Boosting model

A graph with lines and dots

Description automatically generated

Figure 1.0 (70% for training set, 30% for testing set)

A graph with lines and dots

Description automatically generated

Figure 2.0 (80% for training set, 20% for testing set)

1. Predict and actual value’s graph for Lasso regression model

A graph with lines and numbers

Description automatically generated

Figure 3.0 (70% for training set, 30% for testing set)

A graph with lines and dots

Description automatically generated

Figure 4.0 (80% for training set, 20% for testing set)

1. Predict and actual value’s graph for Polynomial regression model

A graph with lines and dots

Description automatically generated

Figure 5.0 (70% for training set, 30% for testing set)

A graph with lines and dots

Description automatically generated

Figure 6.0 (80% for training set, 20% for testing set)

1. Predict and actual value’s graph for Support vector machine model

A graph with lines and dots

Description automatically generated

Figure 7.0 (70% for training set, 30% for testing set)

A graph with lines and dots

Description automatically generated

Figure 8.0 (80% for training set, 20% for testing set)

# 5.0 Discussion

## 5.1 Briefly describe the board research area and narrow it down to your particular focus.

Based on the MSE, RMSE, MAE, and the MAPE in Table 1.0 and Table 2.0 under 4.0 Result section, it can be noticed that the Gradient Boosting model exhibits superior performance compared to other three regression models in predicting laptop prices. Upon data splitting into train an test sets, it is evident that the 70% for training set, 30% for testing set yeilds better model’s performance in terms of accuracy compared to the 80% for training set, 20% for testing set.

We also can noticed that the Lasso Regression model exhibits lowest performance compared to other three regression models in predicting laptop prices with 80% for training set, 20% for testing set.

Other than that, the accuracy of model will be affected by features use for model training and the random state while splitting data.

During feature selection, we planned to drop an important feature which is the brand of laptop. Back to the 1.0 about the problem we trying to solve using supervised learning regression model, a newly founded custom laptop store in Malaysia are relatively new to the customized laptop industry and going to manufacture their own laptop’s brand. According to Asus and MSI Malaysia’s official website, the price difference between two brands with same hardware specifications is RM 3000, the reason behind this big difference is the variance in the materials utilize in the construction of the laptops.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Brand | CPU | RAM | Screen Size | GPU | Storage | Refresh Rate | Price |
| Asus | Intel® Core™ i9-13980HX | DDR5 64GB | 18-inch | RTX4090 16GB GDDR6 | 2TB NVMe PCIe Gen4x4 SSD | 240Hz | RM19,999 |
| MSI | Intel® Core™ i9-13980HX | DDR5 64GB | 18-inch | RTX4090 16GB GDDR6 | 2TB NVMe PCIe Gen4x4 SSD | 240Hz | RM22,999 |

Table 3.0

To predict an approximate price range for new laptop based on specific hardware specifications, it is necessary to drop the brand of laptop as a feature. However, this action may lead to a significant drop in the accuracy of model predictions. By contrast, clients would be able to get a broader price range for the new laptop, taking into account the materials utilize in the construction. When the client seeks higher accuracy and a narrower prediction price range close to the current market price, gradient boosting model will be the best option. On the other hand, when they prioritize less accuracy and are willing to accept a larger prediction price range, lasso regression is recommended in this case.

During model training, we used multiple method for each algorithm’s models to optimise the performance:

For gradient boosting, a loop with maximum 300 epochs with learning rate 0.1 which will find the epoch that give minimum lost function value. The epoch getting from the loop will be used for training gradient boosting model.

For Lasso regression, regularization parameter named as alpha was used to control the balance between fitting the data effectively, which influences the model’s complexity and helps improve accuracy.

For polynomial regression, the model is start with a linear regression model, then the degree of the model will be adjusted to make it as a polynomial regression model. From the observation when the degree equals to 2, it gave us the best model based on data fitting and performance.

For the support vector machine model, it is trained using the radial basis function (RBF) kernel. The RBF is chosen as it allows the SVM to deal with complex and non-linear relationships between features in the dataset. The training process involves 500000 iterations and the epsilon value is set to 0.1. The parameters chosen gave us the best model in terms of accurate predictions.

## 5.2 Comparison between our findings and previous studies

**The Kolla (2016) Study:**

Similar to their study, our Gradient Boosting Regression (GBR) was also among their top performing algorithms when it came to predicting laptop prices. Specifically, their GBR achieved an MSE of 349.25 and an MAE of 14.57 while we scored an MSE of 1141003.78 and an MAE of 515.64 in our GBR.

Similar to their study, we found that GBR was one of our top-performing algorithms for laptop price prediction. However, there is a notable distinction in respective outcomes, while their GBR model achieved an MSE of 349.25 and an MAE of 14.57, ours yielded an MSE of 1141003.78 and an MAE of 515.64 in our GBR. This major difference may be explained by our deliberate feature omission in our study. As mentioned, we excluded the ‘laptop brand’ feature from our independent variables, which may have contributed to the observed difference. This omission is aligned with our study's unique problem statement and emphasis of our study, which revolves around predicting prices of laptops for a newly introduced customized laptop brand. Thus, in our specific problem context, brand is less significant as a feature, which explains the notable difference in model performance.

Next, after Kolla (2016) analyzed their features to identify which ones had most significant impact on price prediction, they found that in their study, the results highlighted RAM, processor speed, storage capacity, screen size, and brand as pivotal attributes for predicting laptop prices. Conversely, features like battery life, weight, graphics card held a relatively smaller role in accurate price prediction. This is quite different from our study as Dedicated graphic memory capacity, RAM, screen resolution and CPU ranking were the more robust predictors, with RAM being notably the most significant. Notably, unlike their study, our study found that expandable memory and screen size were relatively less significant features in price prediction. The contrasting outcomes found may be indicative of very different consumer preferences and requirements in these distinct regions, as their dataset was from India while ours was from Indonesia.

**The Siburian (2022) Study:**

The study conducted by Siburian (2022) had a different problem to solve when predicting laptop prices, their focus centered on mitigating the problem of buying laptops at overly expensive rates. However, their machine learning models were essentially doing the same thing, predicting laptop prices using laptop specifications. When comparing our study’s methodology to theirs, both use regression algorithms, and also follow a similar methodology sequence involving data acquisition, data cleaning, feature engineering, exploratory data analysis (EDA), and model building. However, while their study employs only 3 algorithms: Random Forest Regressor, Gradient Boosting Regressor, and XGBoost Regressor, our study used 4 algorithmsL Support Vector Machine (SVM), Polynomial Regression, Gradient Boosting, and Lasso Regression. The only algorithm in common is Gradient Boosting Regression (GBR).

Next, our study employed a dataset partitioning strategy of using both 70-30 splits and 80-20 splits for training and testing sets, with the 70-30 split resulting in more accurate prediction power. In their study, they only used the 70-30 split. Their top performing model was from the XGBoost model with an R2 score of 92.77% indicating high accuracy in the prediction model while ours was Gradient Boosting model. The features selected for training their models closely resembled ours, except they had operating systems in their features, while they didn't include refresh rate, a feature we included. Despite these nuanced differences, both studies came to the same conclusion, which is that RAM emerged as a significant feature influencing price prediction in laptops.

# 6.0 Conclusion

## 6.1 Summary of the hypothesis and purpose of the study

For the hypothesis 1 we fail to reject hypothesis 1 and reject the alternate hypothesis. There is a positive correlation between laptop specifications such as RAM, storage capacity, and display resolution and performance benchmark rankings.

As for the Hypothesis 2 we fail to reject it as well and we proved that the laptops with higher performance rankings will have higher prices in the market. The purpose of this study is to predict the price of a laptop, usually we need it when a company want to sell off, they laptop to the second-hand market and we provide this price prediction service for them.

We manage to find out the relationship between the specs of the computer and we know with what kind of specs can have what selling price. On the other hand, we proved that there is a significant price difference between laptops with different performance benchmark rankings.

## 6.2 Discuss potential future enhancements

1. Data cleaning and data preprocessing

Feature selection: Getting various methods for feature selection to identify most relevant features that would impact the model regarding the prediction.

Feature engineering: Experiment with constructing and deriving new features or transforming existing ones to improve data quality for model training.

Data Splitting: Decide more data splitting ratio into train and test set with suitable random state value.

1. Gradient Boosting model

Parameter tuning: Explore new parameters combination that would optimise the model performance. The known parameters for current model are epochs value and learning rate.

1. Lasso regression model

Regularization parameter tuning: Try with different value for L1 regularization parameter known as alpha to find the best regularization apply to the model during model training which prevent overfitting of dataset.

1. Polynomial regression model

Degree selection: Try all others degree for the hypothesis to build a detail and complex model which provide accurate prediction for non-linear problems.

Feature scaling: Polynomial regression is a model sensitive to the scale of independent variables, feature scaling could apply to standardise and normalise the the scale of independent variables.

1. Support Vector Machine model

Kernel selection: Try with others kernel algorithms to drive the dataset into others high dimensional space and analyse the impact of each kernel on the model.

Parameters tuning: Explore the impact of the C-parameter and epsilon on the model’s performance which can use cross-validation test to get the optimum.

# 7.0 References

1. Jean, Z. D., & Himpens, S. (2019). Webscraping Laptop Prices To Estimate Hedonic Models And Extensions To Other Predictive Methods.
2. Kolla, V. R. K. (2016). Forecasting Laptop Prices: A Comparative Study Of Machine Learning Algorithms For Predictive Modeling. International Journal of Information Technology & Management Information System (IJITMIS), 7(3), 8–14.
3. Siburian, A. D. (2022). Laptop Price Prediction with Machine Learning Using Regression Algorithm. JUSIKOM Journal of Information Systems and Computer Science, 6(1), 87–91
4. Brownlee, J. (2021). *What Is Gradient Boosting?* [Online] Available at: <https://machinelearningmastery.com/gradient-boosting/>
5. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.).* Available at: O'Reilly Media.
6. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.
7. Brownlee, J. (2021). *Polynomial Regression for Machine Learning.* [Online] Available at: <https://machinelearningmastery.com/polynomial-features-transforms-for-machine-learning/>
8. Friedman, J., Hastie, T., Tibshirani, R., & Simon, N. (2010). *Regularization Paths for Generalized Linear Models via Coordinate Descent.* Available at: Journal of Statistical Software, 33(1), 1-22.
9. Brownlee, J. (2020). *Feature Selection with the LASSO: How to choose the right alpha?* [Online] Available at: <https://machinelearningmastery.com/feature-selection-with-the-lasso/>

# 8.0 Appendix

## 8.1 Data cleaning and data preprocessing

1. Import library

|  |
| --- |
| import pandas as pd  import numpy as np  import seaborn as sns  import matplotlib.pyplot as plt |

1. Import dataset

|  |
| --- |
| data = pd.read\_csv("C://Users/User/ Laptop ranked dataset.csv") |

1. Display dataset details

|  |
| --- |
| data.head()  data.shape  data.dtypes |

1. Missing data handling

|  |
| --- |
| nan\_col = data.columns[data.isnull().any()]  for i in nan\_col:  print(i, data[i].isnull().sum()) |

|  |
| --- |
| data.drop(columns=['user rating'], axis = 1, inplace = True) |

1. Duplicate data handling

|  |
| --- |
| data.duplicated().sum() |

1. Deriving new attribute

|  |
| --- |
| data['Price(RM)'] = (data['Price (in Indian Rupees)'] \* 0.055).round(2)  data.drop(columns=['Price (in Indian Rupees)'], inplace=True) |

1. Convert records to proper values

|  |
| --- |
| data['Refresh Rate'].value\_counts()  def convert\_refresh\_rate(int):  if int == 60:  return 1  else:  return 2    data['Refresh\_Rate'] = data['Refresh Rate'].apply(convert\_refresh\_rate)  data.drop(columns=['Refresh Rate'], inplace=True)  sns.barplot(x=data['Refresh\_Rate'],y=data['Price(RM)'])  plt.xticks(rotation='vertical')  plt.show() |

1. Features selection

|  |
| --- |
| print(data.dtypes.to\_string(max\_rows=None))  data.drop(columns=['index', 'link', 'name', 'Processor name', 'gpu name ','Processor Brand', 'RAM Type','Operating System', 'company', 'battery\_backup', 'Weight (in kg)'], inplace=True)  data.head() |

1. Exploratoty data analysis

|  |
| --- |
| correlation\_matrix = data.corr()  sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')  plt.title('Correlation Heatmap')  plt.show()  data.corr()['Price(RM)'] |

|  |
| --- |
| data.drop(columns=['gpu\_processor tokenized'], inplace=True) |

1. Export cleaned dataset

|  |
| --- |
| data.to\_csv('cleaned\_dataset.csv', index=False) |

## 8.2 Gradient Boosting model

1. Import library

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split, cross\_val\_score  from sklearn.ensemble import GradientBoostingRegressor  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, mean\_absolute\_percentage\_error |

1. Import cleaned dataset

|  |
| --- |
| data = pd.read\_csv("C://Users/User/ Cleaned\_dataset.csv") |

1. Independent and dependent variables

|  |
| --- |
| x = data.drop(columns=['Price(RM)'])  y = data['Price(RM)'] |

1. Data Spliting (70/30)

|  |
| --- |
| X\_train1, X\_test1, Y\_train1, Y\_test1 = train\_test\_split(x, y, test\_size=0.3, random\_state = 3) |

1. Data Spliting (80/20)

|  |
| --- |
| X\_train2, X\_test2, Y\_train2, Y\_test2 = train\_test\_split(x, y, test\_size=0.2, random\_state = 3) |

1. Iteration (Epochs) that give minimum error (70/30)

|  |
| --- |
| mse\_values = []  n\_estimators = 300  min\_mse\_iteration = 0  min\_mse = float('inf')  model1 = GradientBoostingRegressor(learning\_rate = 0.1)  for i in range(n\_estimators):  model1.fit(X\_train1, Y\_train1)  model\_pred\_test1 = model1.predict(X\_test1)    mse = mean\_squared\_error(Y\_test1, model\_pred\_test1)  mse\_values.append(mse)    if mse < min\_mse:  min\_mse\_iteration = i  min\_mse = mse  plt.plot(range(1, n\_estimators + 1), mse\_values, marker='o')  plt.xlabel('Number of Iterations (Trees)')  plt.ylabel('Mean Squared Error (MSE)')  plt.title('MSE on Each Iteration')  plt.show()  print("Minimum MSE:", min\_mse)  print("Iteration:", min\_mse\_iteration) |

1. Model training (70/30)

|  |
| --- |
| model = GradientBoostingRegressor(learning\_rate=0.1)  for i in range(min\_mse\_iteration - 1):  model.fit(X\_train1, Y\_train1)  model\_pred\_test = model.predict(X\_test1) |

1. Evaluation (70/30)

|  |
| --- |
| mse = mean\_squared\_error(Y\_test1, model\_pred\_test)  rmse = (np.sqrt(mean\_squared\_error(Y\_test1, model\_pred\_test)))  mae = mean\_absolute\_error(Y\_test1, model\_pred\_test)  def fetch\_mpe(y\_true, y\_pred):  mpe = np.mean((y\_true - y\_pred) / y\_true) \* 100  return mpe  mpe = fetch\_mpe(Y\_test1, model\_pred\_test)  scoring\_results = [  {'Gradient Boosting Regression': 'MSE', 'Value': mse},  {'Gradient Boosting Regression': 'RMSE', 'Value': rmse},  {'Gradient Boosting Regression': 'MAE', 'Value': mae},  {'Gradient Boosting Regression': 'MAPE', 'Value': mape},  {'Gradient Boosting Regression': 'MPE', 'Value': mpe},  ]  scoring\_df = pd.DataFrame(scoring\_results)  scoring\_df |

1. Plot predicted result (70/30)

|  |
| --- |
| difference = Y\_test1 - model\_pred\_test  difference = difference.abs()  pred\_df = pd.DataFrame({'Predicted': model\_pred\_test, 'Original' : Y\_test1, 'Difference' : difference})  pred\_df  num\_random\_points = 15  pred\_df = pred\_df.sample(n=num\_random\_points, random\_state=4)  plt.figure(figsize=(10, 6))  plt.plot(pred\_df.index, pred\_df['Predicted'], label='Predicted', marker='o')  plt.plot(pred\_df.index, pred\_df['Original'], label='Original', marker='x')  plt.xlabel('Independent')  plt.ylabel('Price(RM)')  plt.title('Original vs. Predicted')  plt.legend()  plt.grid(True)  plt.show() |

1. Iteration (Epochs) that give minimum error (80/20)

|  |
| --- |
| mse\_values = []  n\_estimators = 300  min\_mse\_iteration1 = -1  min\_mse1 = float('inf')  model2 = GradientBoostingRegressor(learning\_rate = 0.1)  for i in range(n\_estimators):  model2.fit(X\_train2, Y\_train2)  model\_pred\_test2 = model2.predict(X\_test2)    mse = mean\_squared\_error(Y\_test2, model\_pred\_test2)  mse\_values.append(mse)    if mse < min\_mse1:  min\_mse\_iteration1 = i  min\_mse1 = mse  plt.plot(range(1, n\_estimators + 1), mse\_values, marker='o')  plt.xlabel('Number of Iterations (Trees)')  plt.ylabel('Mean Squared Error (MSE)')  plt.title('MSE on Each Iteration')  plt.show()  print("Minimum MSE:", min\_mse1)  print("Iteration:", min\_mse\_iteration1) |

1. Model training (80/20)

|  |
| --- |
| model = GradientBoostingRegressor(learning\_rate=0.1)  for i in range(min\_mse\_iteration1 - 1):  model.fit(X\_train2, Y\_train2)  model\_pred\_test = model.predict(X\_test2) |

1. Evaluation (80/20)

|  |
| --- |
| mse = mean\_squared\_error(Y\_test2, model\_pred\_test)  rmse = (np.sqrt(mean\_squared\_error(Y\_test2, model\_pred\_test)))  mae = mean\_absolute\_error(Y\_test2, model\_pred\_test)  mape = mean\_absolute\_percentage\_error(Y\_test2, model\_pred\_test)  def fetch\_mpe(y\_true, y\_pred):  mpe = np.mean((y\_true - y\_pred) / y\_true) \* 100  return mpe  mpe = fetch\_mpe(Y\_test2, model\_pred\_test) |

1. Plot predicted result (80/20)

|  |
| --- |
| difference = Y\_test2 - model\_pred\_test  difference = difference.abs()  pred\_df = pd.DataFrame({'Predicted': model\_pred\_test, 'Original' : Y\_test2, 'Difference' : difference})  pred\_df  num\_random\_points = 15  pred\_df = pred\_df.sample(n=num\_random\_points, random\_state=4)  plt.figure(figsize=(10, 6))  plt.plot(pred\_df.index, pred\_df['Predicted'], label='Predicted', marker='o')  plt.plot(pred\_df.index, pred\_df['Original'], label='Original', marker='x')  plt.xlabel('Independent')  plt.ylabel('Price(RM)')  plt.title('Original vs. Predicted')  plt.legend()  plt.grid(True)  plt.show() |

## 8.3 Lasso Regression model

1. Import library

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split, cross\_val\_score  from sklearn.linear\_model import Lasso  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, mean\_absolute\_percentage\_error |

1. Import cleaned dataset

|  |
| --- |
| data = pd.read\_csv("C://Users/User/ Cleaned\_dataset.csv") |

1. Independent and dependent variables

|  |
| --- |
| x = data.drop(columns=['Price(RM)'])  y = data['Price(RM)'] |

1. Data Spliting (70/30)

|  |
| --- |
| X\_train1, X\_test1, Y\_train1, Y\_test1 = train\_test\_split(x, y, test\_size=0.3, random\_state = 3) |

1. Data Spliting (80/20)

|  |
| --- |
| X\_train2, X\_test2, Y\_train2, Y\_test2 = train\_test\_split(x, y, test\_size=0.2, random\_state = 3) |

1. Model training (70/30)

|  |
| --- |
| a = 1  model = Lasso(alpha=a)  model.fit(X\_train1, Y\_train1)  model\_pred\_test = model.predict(X\_test1) |

1. Evaluation (70/30)

|  |
| --- |
| mse = mean\_squared\_error(Y\_test1, model\_pred\_test)  rmse = (np.sqrt(mean\_squared\_error(Y\_test1, model\_pred\_test)))  mae = mean\_absolute\_error(Y\_test1, model\_pred\_test)  def fetch\_mpe(y\_true, y\_pred):  mpe = np.mean((y\_true - y\_pred) / y\_true) \* 100  return mpe  mpe = fetch\_mpe(Y\_test1, model\_pred\_test)  scoring\_results = [  {'Gradient Boosting Regression': 'MSE', 'Value': mse},  {'Gradient Boosting Regression': 'RMSE', 'Value': rmse},  {'Gradient Boosting Regression': 'MAE', 'Value': mae},  {'Gradient Boosting Regression': 'MAPE', 'Value': mape},  {'Gradient Boosting Regression': 'MPE', 'Value': mpe},  ]  scoring\_df = pd.DataFrame(scoring\_results)  scoring\_df |

1. Plot predicted result (70/30)

|  |
| --- |
| difference = Y\_test1 - model\_pred\_test  difference = difference.abs()  pred\_df = pd.DataFrame({'Predicted': model\_pred\_test, 'Original' : Y\_test1, 'Difference' : difference})  pred\_df  num\_random\_points = 15  pred\_df = pred\_df.sample(n=num\_random\_points, random\_state=4)  plt.figure(figsize=(10, 6))  plt.plot(pred\_df.index, pred\_df['Predicted'], label='Predicted', marker='o')  plt.plot(pred\_df.index, pred\_df['Original'], label='Original', marker='x')  plt.xlabel('Independent')  plt.ylabel('Price(RM)')  plt.title('Original vs. Predicted')  plt.legend()  plt.grid(True)  plt.show() |

1. Model training (80/20)

|  |
| --- |
| a=1  model = Lasso(alpha=a)  model.fit(X\_train2, Y\_train2)  model\_pred\_test = model.predict(X\_test2) |

1. Evaluation (80/20)

|  |
| --- |
| mse = mean\_squared\_error(Y\_test2, model\_pred\_test)  rmse = (np.sqrt(mean\_squared\_error(Y\_test2, model\_pred\_test)))  mae = mean\_absolute\_error(Y\_test2, model\_pred\_test)  mape = mean\_absolute\_percentage\_error(Y\_test2, model\_pred\_test)  def fetch\_mpe(y\_true, y\_pred):  mpe = np.mean((y\_true - y\_pred) / y\_true) \* 100  return mpe  mpe = fetch\_mpe(Y\_test2, model\_pred\_test) |

1. Plot predicted result (80/20)

|  |
| --- |
| difference = Y\_test2 - model\_pred\_test  difference = difference.abs()  pred\_df = pd.DataFrame({'Predicted': model\_pred\_test, 'Original' : Y\_test2, 'Difference' : difference})  pred\_df  num\_random\_points = 15  pred\_df = pred\_df.sample(n=num\_random\_points, random\_state=4)  plt.figure(figsize=(10, 6))  plt.plot(pred\_df.index, pred\_df['Predicted'], label='Predicted', marker='o')  plt.plot(pred\_df.index, pred\_df['Original'], label='Original', marker='x')  plt.xlabel('Independent')  plt.ylabel('Price(RM)')  plt.title('Original vs. Predicted')  plt.legend()  plt.grid(True)  plt.show() |

## 8.4 Polynomial regression model

1. Import library

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split, cross\_val\_score  from sklearn.linear\_model import LinearRegression  from sklearn.preprocessing import PolynomialFeatures  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, mean\_absolute\_percentage\_error |

1. Import cleaned dataset

|  |
| --- |
| data = pd.read\_csv("C://Users/User/ Cleaned\_dataset.csv" |

1. Independent and dependent variables

|  |
| --- |
| x = data.drop(columns=['Price(RM)'])  y = data['Price(RM)'] |

1. Data Spliting (70/30)

|  |
| --- |
| X\_train1, X\_test1, Y\_train1, Y\_test1 = train\_test\_split(x, y, test\_size=0.3, random\_state = 3) |

1. Data Spliting (80/20)

|  |
| --- |
| X\_train2, X\_test2, Y\_train2, Y\_test2 = train\_test\_split(x, y, test\_size=0.2, random\_state = 3) |

1. Model training (70/30)

|  |
| --- |
| degree = 2  poly = PolynomialFeatures(degree=degree)  X\_poly = poly.fit\_transform(X\_train1)  model = LinearRegression()  model.fit(X\_poly, Y\_train1)  X\_new\_poly = poly.transform(X\_test1)  model\_pred\_test = model.predict(X\_new\_poly) |

1. Evaluation (70/30)

|  |
| --- |
| mse = mean\_squared\_error(Y\_test1, model\_pred\_test)  rmse = (np.sqrt(mean\_squared\_error(Y\_test1, model\_pred\_test)))  mae = mean\_absolute\_error(Y\_test1, model\_pred\_test)  def fetch\_mpe(y\_true, y\_pred):  mpe = np.mean((y\_true - y\_pred) / y\_true) \* 100  return mpe  mpe = fetch\_mpe(Y\_test1, model\_pred\_test)  scoring\_results = [  {'Gradient Boosting Regression': 'MSE', 'Value': mse},  {'Gradient Boosting Regression': 'RMSE', 'Value': rmse},  {'Gradient Boosting Regression': 'MAE', 'Value': mae},  {'Gradient Boosting Regression': 'MAPE', 'Value': mape},  {'Gradient Boosting Regression': 'MPE', 'Value': mpe},  ]  scoring\_df = pd.DataFrame(scoring\_results)  scoring\_df |

1. Plot predicted result (70/30)

|  |
| --- |
| difference = Y\_test1 - model\_pred\_test  difference = difference.abs()  pred\_df = pd.DataFrame({'Predicted': model\_pred\_test, 'Original' : Y\_test1, 'Difference' : difference})  pred\_df  num\_random\_points = 15  pred\_df = pred\_df.sample(n=num\_random\_points, random\_state=4)  plt.figure(figsize=(10, 6))  plt.plot(pred\_df.index, pred\_df['Predicted'], label='Predicted', marker='o')  plt.plot(pred\_df.index, pred\_df['Original'], label='Original', marker='x')  plt.xlabel('Independent')  plt.ylabel('Price(RM)')  plt.title('Original vs. Predicted')  plt.legend()  plt.grid(True)  plt.show() |

1. Model training (80/20)

|  |
| --- |
| degree=2  poly = PolynomialFeatures(degree=degree)  X\_poly = poly.fit\_transform(X\_train2)  model = LinearRegression()  model.fit(X\_poly, Y\_train2)  X\_new\_poly = poly.transform(X\_test2)  model\_pred\_test = model.predict(X\_new\_poly) |

1. Evaluation (80/20)

|  |
| --- |
| mse = mean\_squared\_error(Y\_test2, model\_pred\_test)  rmse = (np.sqrt(mean\_squared\_error(Y\_test2, model\_pred\_test)))  mae = mean\_absolute\_error(Y\_test2, model\_pred\_test)  mape = mean\_absolute\_percentage\_error(Y\_test2, model\_pred\_test)  def fetch\_mpe(y\_true, y\_pred):  mpe = np.mean((y\_true - y\_pred) / y\_true) \* 100  return mpe  mpe = fetch\_mpe(Y\_test2, model\_pred\_test) |

1. Plot predicted result (80/20)

|  |
| --- |
| difference = Y\_test2 - model\_pred\_test  difference = difference.abs()  pred\_df = pd.DataFrame({'Predicted': model\_pred\_test, 'Original' : Y\_test2, 'Difference' : difference})  pred\_df  num\_random\_points = 15  pred\_df = pred\_df.sample(n=num\_random\_points, random\_state=4)  plt.figure(figsize=(10, 6))  plt.plot(pred\_df.index, pred\_df['Predicted'], label='Predicted', marker='o')  plt.plot(pred\_df.index, pred\_df['Original'], label='Original', marker='x')  plt.xlabel('Independent')  plt.ylabel('Price(RM)')  plt.title('Original vs. Predicted')  plt.legend()  plt.grid(True)  plt.show() |

## 8.5 Support vector machine model

1. Import library

|  |
| --- |
| import pandas as pd  import numpy as np  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split, cross\_val\_score  from sklearn.svm import SVR  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, mean\_absolute\_percentage\_error |

1. Import cleaned dataset

|  |
| --- |
| data = pd.read\_csv("C://Users/User/ Cleaned\_dataset.csv") |

1. Independent and dependent variables

|  |
| --- |
| x = data.drop(columns=['Price(RM)'])  y = data['Price(RM)'] |

1. Data Spliting (70/30)

|  |
| --- |
| X\_train1, X\_test1, Y\_train1, Y\_test1 = train\_test\_split(x, y, test\_size=0.3, random\_state = 3) |

1. Data Spliting (80/20)

|  |
| --- |
| X\_train2, X\_test2, Y\_train2, Y\_test2 = train\_test\_split(x, y, test\_size=0.2, random\_state = 3) |

1. Model training (70/30)

|  |
| --- |
| iteration = 5000000  model = SVR(kernel='rbf', C=iteration, epsilon=0.1)  model.fit(X\_train1,Y\_train1)  model\_pred\_test = model.predict(X\_test1) |

1. Evaluation (70/30)

|  |
| --- |
| mse = mean\_squared\_error(Y\_test1, model\_pred\_test)  rmse = (np.sqrt(mean\_squared\_error(Y\_test1, model\_pred\_test)))  mae = mean\_absolute\_error(Y\_test1, model\_pred\_test)  def fetch\_mpe(y\_true, y\_pred):  mpe = np.mean((y\_true - y\_pred) / y\_true) \* 100  return mpe  mpe = fetch\_mpe(Y\_test1, model\_pred\_test)  scoring\_results = [  {'Gradient Boosting Regression': 'MSE', 'Value': mse},  {'Gradient Boosting Regression': 'RMSE', 'Value': rmse},  {'Gradient Boosting Regression': 'MAE', 'Value': mae},  {'Gradient Boosting Regression': 'MAPE', 'Value': mape},  {'Gradient Boosting Regression': 'MPE', 'Value': mpe},  ]  scoring\_df = pd.DataFrame(scoring\_results)  scoring\_df |

1. Plot predicted result (70/30)

|  |
| --- |
| difference = Y\_test1 - model\_pred\_test  difference = difference.abs()  pred\_df = pd.DataFrame({'Predicted': model\_pred\_test, 'Original' : Y\_test1, 'Difference' : difference})  pred\_df  num\_random\_points = 15  pred\_df = pred\_df.sample(n=num\_random\_points, random\_state=4)  plt.figure(figsize=(10, 6))  plt.plot(pred\_df.index, pred\_df['Predicted'], label='Predicted', marker='o')  plt.plot(pred\_df.index, pred\_df['Original'], label='Original', marker='x')  plt.xlabel('Independent')  plt.ylabel('Price(RM)')  plt.title('Original vs. Predicted')  plt.legend()  plt.grid(True)  plt.show() |

1. Model training (80/20)

|  |
| --- |
| iteration = 5000000  model = SVR(kernel='rbf', C=iteration, epsilon=0.1)  model.fit(X\_train2,Y\_train2)  model\_pred\_test = model.predict(X\_test2) |

1. Evaluation (80/20)

|  |
| --- |
| mse = mean\_squared\_error(Y\_test2, model\_pred\_test)  rmse = (np.sqrt(mean\_squared\_error(Y\_test2, model\_pred\_test)))  mae = mean\_absolute\_error(Y\_test2, model\_pred\_test)  mape = mean\_absolute\_percentage\_error(Y\_test2, model\_pred\_test)  def fetch\_mpe(y\_true, y\_pred):  mpe = np.mean((y\_true - y\_pred) / y\_true) \* 100  return mpe  mpe = fetch\_mpe(Y\_test2, model\_pred\_test) |

1. Plot predicted result (80/20)

|  |
| --- |
| difference = Y\_test2 - model\_pred\_test  difference = difference.abs()  pred\_df = pd.DataFrame({'Predicted': model\_pred\_test, 'Original' : Y\_test2, 'Difference' : difference})  pred\_df  num\_random\_points = 15  pred\_df = pred\_df.sample(n=num\_random\_points, random\_state=4)  plt.figure(figsize=(10, 6))  plt.plot(pred\_df.index, pred\_df['Predicted'], label='Predicted', marker='o')  plt.plot(pred\_df.index, pred\_df['Original'], label='Original', marker='x')  plt.xlabel('Independent')  plt.ylabel('Price(RM)')  plt.title('Original vs. Predicted')  plt.legend()  plt.grid(True)  plt.show() |