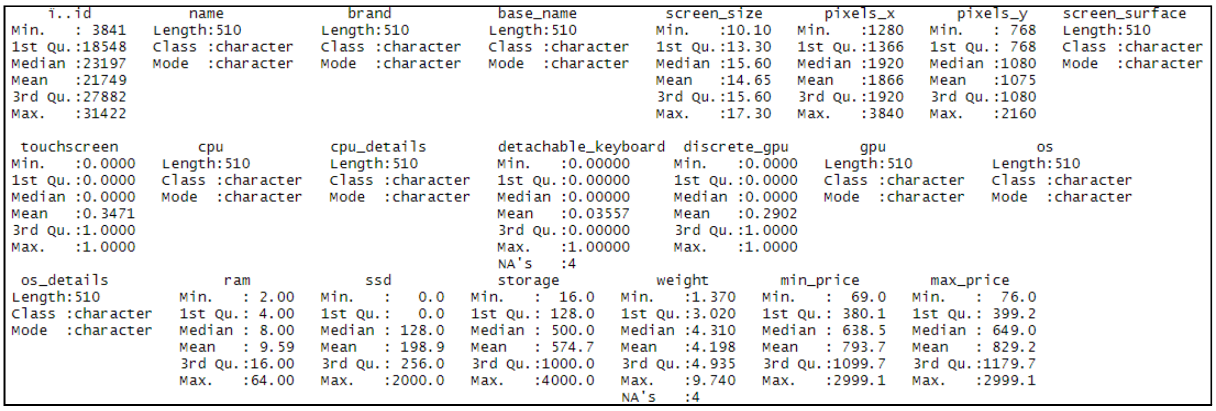
## Introduction

The provided data described various features of laptops (such as brand, CPU and GPU details) and their minimum and maximum price over three weeks time. The given data consisted of two parts: a training dataset with 510 observations and a test dataset with 222 observations. Both of them contained 20 features each, however, only the training set included target variables minimum and maximum price. The test dataset was provided to rank the trained model in the competition leaderboard based on half of the test data. Later the second (hidden) leaderboard was revealed with rankings based on the second half of the data. Thus the main aim of this assignment was to build a predictive model which could predict the minimum and maximum price and perform similarly in both leaderboards. R software was used for this task.

## Data Exploration

The first step was to explore the characteristics of the training data using both visual and numerical analysis. We focused on the informativeness and completeness of data and relationships between variables. Based on this the subsequent data processing and modeling approach was developed.

Summary statistics of the training dataset (Figure 1) indicated that it contained a number of categorical variables. Table 1 further showed that most of them comprised a substantial number of categories rendering it difficult to use methods like linear regression due to loss of degrees of freedom and hence reduced precision. Also, it was decided that variables *name*, *base_name*, *cpu*, *cpu_details*, *gpu* and *os_details* were too granular and often included information which was often already included in other variables, e.g. the most important piece of information from *os_details* is whether the OS is Windows, Mac or other, which is already included in the variable *os*. On the other hand, some information from the *name, gpu* and *cpu_details* variables was extracted to create new variables (see later in the Data Preprocessing section).



```
     i..id             name              brand            base_name          screen_size       pixels_x        pixels_y      screen_surface
 Min.    : 3841   Length:510         Length:510         Length:510         Min.   :10.10   Min.   :1280   Min.   : 768   Length:510
 1st Qu.:18548   Class :character   Class :character   Class :character   1st Qu.:13.30   1st Qu.:1366   1st Qu.: 768   Class :character
 Median :23197   Mode  :character   Mode  :character   Mode  :character   Median :15.60   Median :1920   Median :1080   Mode  :character
 Mean   :21749                                                            Mean   :14.65   Mean   :1866   Mean   :1075
 3rd Qu.:27882                                                            3rd Qu.:15.60   3rd Qu.:1920   3rd Qu.:1080
 Max.   :31422                                                            Max.   :17.30   Max.   :3840   Max.   :2160

  touchscreen         cpu            cpu_details        detachable_keyboard  discrete_gpu        gpu                 os
 Min.   :0.0000   Length:510         Length:510         Min.   :0.00000     Min.   :0.0000   Length:510         Length:510
 1st Qu.:0.0000   Class :character   Class :character   1st Qu.:0.00000     1st Qu.:0.0000   Class :character   Class :character
 Median :0.0000   Mode  :character   Mode  :character   Median :0.00000     Median :0.0000   Mode  :character   Mode  :character
 Mean   :0.3471                                         Mean   :0.03557     Mean   :0.2902
 3rd Qu.:1.0000                                         3rd Qu.:0.00000     3rd Qu.:1.0000
 Max.   :1.0000                                         Max.   :1.00000     Max.   :1.0000
                                                        NA's   :4
  os_details             ram             ssd             storage            weight         min_price         max_price
 Length:510         Min.   : 2.00   Min.   :   0.0   Min.   :  16.0   Min.   :1.370   Min.   :  69.0   Min.   :  76.0
 Class :character   1st Qu.: 4.00   1st Qu.:   0.0   1st Qu.: 128.0   1st Qu.:3.020   1st Qu.: 380.1   1st Qu.: 399.2
 Mode  :character   Median : 8.00   Median : 128.0   Median : 500.0   Median :4.310   Median : 638.5   Median : 649.0
                    Mean   : 9.59   Mean   : 198.9   Mean   : 574.7   Mean   :4.198   Mean   : 793.7   Mean   : 829.2
                    3rd Qu.:16.00   3rd Qu.: 256.0   3rd Qu.:1000.0   3rd Qu.:4.935   3rd Qu.:1099.7   3rd Qu.:1179.7
                    Max.   :64.00   Max.   :2000.0   Max.   :4000.0   Max.   :9.740   Max.   :2999.1   Max.   :2999.1
                                                                      NA's   :4
```

***Figure 1.*** *Summary Statistics of the Given Training Dataset*

***Table 1.*** *Number of Categories for Each Categorical Variable*

| Categorical variable | name | brand | base_ name | screen_ surface | cpu | cpu_ details | gpu | os | os_ details |
|---|---|---|---|---|---|---|---|---|---|
| **No. of categories** | 510 | 18 | 385 | 2 | 23 | 133 | 74 | 5 | 21 |

After analysing categorical variables, we looked at the numerical ones. Figure 2 shows the correlogram of all numerical variables in the training dataset. As expected, maximum price and minimum price are highly positively correlated among themselves and both of them are highly / moderately positively correlated with *ram*, *discrete_gpu* and *storage*. Therefore the latter features might be very important in predicting the prices. On the other hand, we noticed that a few explanatory variables are also highly correlated among themselves which might introduce multicollinearity. This will be checked after the data preprocessing stage, since indeed in the case of multicollinearity some variables might have to be removed. However, since *pixels_x* and *pixels_y* have a correlation of 0.98 (most laptops have a fixed 16:9 aspect ratio), we already decided to use only *pixels_y* for the modelling.

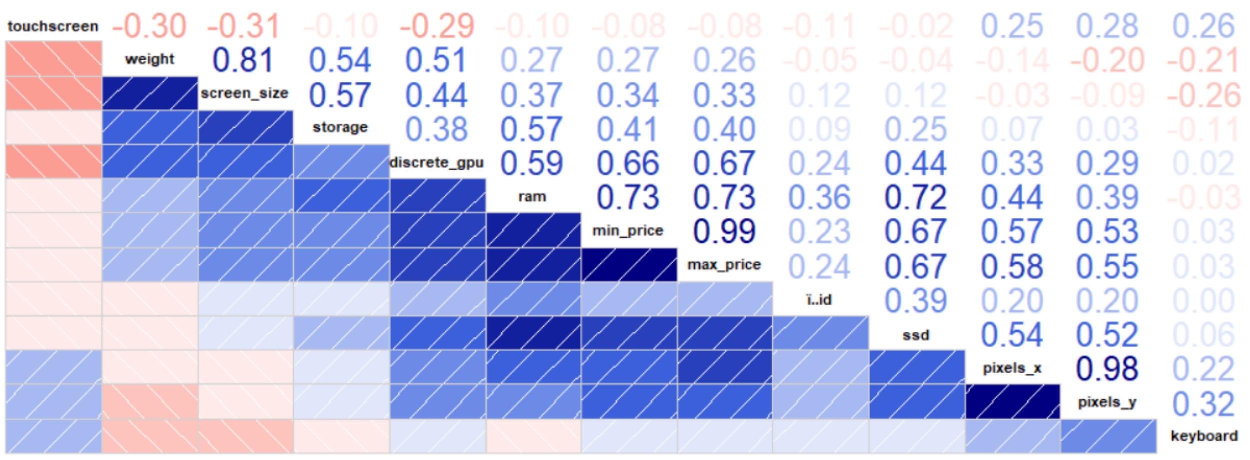


***Figure 2.*** *Correlogram of Numerical Variables*

Missingness analysis revealed that 0.3% of the training dataset values were missing, which, in particular, affected columns *screen_surface*, *cpu_details*, *weight, detachable_keyboard*, *os_details*, *os* and *gpu* (Figure 3, left panel). However, analysis by instance (Figure 3, right panel) showed that the missing values were distributed such that no laptop had more than four missing values. Since the number of variables is relatively large in this case, the missing value imputation could be done without significant information loss and increase in bias (e.g. towards median if median imputation was used). Note also that variables like *cpu_details* and *os_details* were later removed due to their granularity as explained above, thus reducing the number of missing values. Additional missing value diagnostics will be performed after feature extraction.

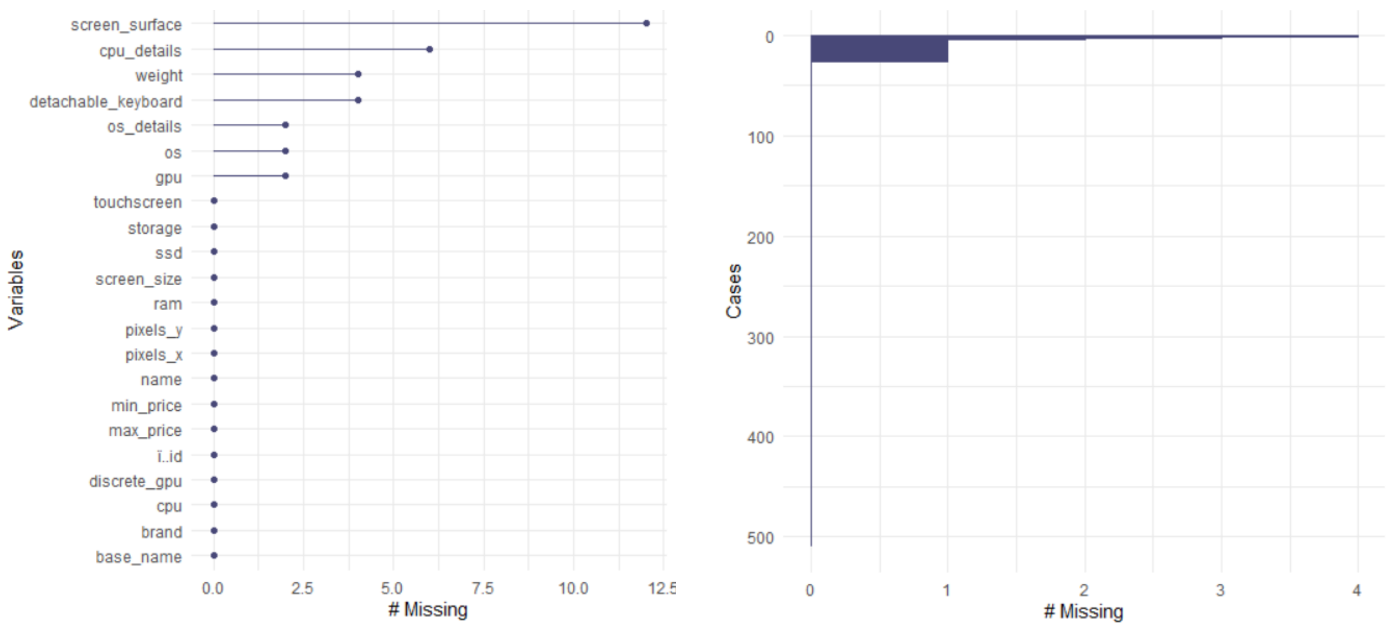


***Figure 3.*** *Missigness by Variable (Left Panel) and by Instance (Right Panel)*

## Data Preprocessing

After data exploration we proceeded with data preprocessing which aimed to improve the quality of the data. The main stages of this step included data cleaning, feature extraction and missing value imputation. All steps were first performed on the training dataset and then applied to the test set to avoid data leakage.

### Data Cleaning

Some additional observations made from the data exploration included the need to change the first column name to *id* (of course, this variable would not be used in the modeling anyway), set all missing values and empty cells to NA and set the *screen_surface* variable to lowercase (otherwise it would include four categories “matte”, “Matte”, “glossy” and “Glossy” instead of two).

### Feature Extraction

Regarding the feature extraction, since the categorical features *cpu_details* and *name* included some information that might have helped in predicting the target variables, this information was extracted to create new variables. In particular, the *cpu_details* variable was used to extract features *ghz*, *thread* (multi-threading / hyper-threading / no-threading) and *cores* (2, 4, 6 or 8 cores based on keywords dual, quad, hexa and octa). Meanwhile, the *name* variable contained additional specific laptop features, such as Bluetooth or HDMI, thus intermediate binary variables *bluetooth*, *webcam*, *usb*, *wifi*, *dvd*, *hdmi*, *fingerprint* and *gold* (colour of the laptop) were created and then combined into an additional variable *perks*, a weighted sum of the eight newly created variables. Only the latter variable was used in the following analysis.

Based on the personal experience it was also hypothesized that a brand like Apple would have relatively overpriced laptops for their specifications, thus a variable *brand_premium* was created to reflect whether the laptop brand was Apple or not. Additionally, it was hypothesized that the models would give higher accuracy when the difference between maximum and minimum price, and the minimum price itself were used as outcome variables (instead of the given maximum and minimum prices). Therefore, additional variable *price_range* (absolute value of max_price - min_price) was created as an outcome variable.

It is important to note that the newly added *ghz* and *cores* variables had eight and seven missing values, respectively. However, after extracting necessary information from certain variables and deleting them (e.g. *cpu_details*), we observed no overall increase in the percentage of missing values in the dataset and all instances still had at most four missing values.

Finally, after extensive research on laptops and their components, it was found that, considering the recent developments in computationally intensive modelling techniques such as neural networks and GPU usage in tasks like cryptocurrency mining, the Graphical Processing Unit (GPU) is arguably the most expensive part of a laptop making up ~50% of the entire laptop price for high-end laptops. Therefore, it was decided that adding GPU parameters as potential explanatory variables was extremely important for our analysis. However, since the information about GPU in the received training dataset was very limited (containing only GPU names) an external dataset containing details about GPUs was used[1]. A total of 1244 GPUs released between years 2012 and 2020 was extracted and the data was preprocessed by deleting whitespaces, changing brand name capitalization and modifying graphic card names to correspond to the GPU name format in the assignment dataset. Afterwards the original laptop price dataset was joined with the newly extracted and processed GPU details dataset using GPU name as a joining value. As a result, the following features relating to the GPUs were obtained: memory in terms of megabytes in case of integrated GPU (*gpu_memory*), GPU clock (*gpu_clock*), memory clock (*gpu_memory_clock*), shaders (*gpu_shaders*), TMUs (*gpu_tmu*) and ROPs (*gpu_rop*). In case of the value "Memory Shared" in variables *gpu_memory* and *gpu_memory_clock*, we changed it to 0, since shared memory refers to no dedicated memory but rather sharing RAM with CPU and other parts.

After joining the original train dataset with the new GPU details dataset we observed that some GPUs in the original data did not get matches from the new data. In particular, there were 17 instances in total which now had missing values in all of the newly created columns. Thus in addition to the missing values described above we also had to deal with these new missing values.

[1] https://www.techpowerup.com/gpu-specs/?mobile=No&sort=name

## Missing Data Handling

To deal with missing values we tried several combinations of kNN, median, mode and other imputation methods and selected the best one based on how they improved model fit. The following procedure provided the most accurate results.

To fill in missing values for the newly created variables we did the following:

1. We extracted the GPU brand name and if it did not contain one of the three major brands (Intel, Nvidia or AMD), we classified it as "Other".
2. Given the brand (other than "Other"), we searched for the most popular GPU (*gpu*) of that brand in the original training dataset and imputed missing GPU details (*gpu_memory*, *gpu_clock*, etc.) from the most popular GPU.
3. For the brand "Other", we searched for the most popular discrete GPU in the original training dataset (which in our case was AMD Radeon R4) and imputed missing GPU details (*gpu_memory*, *gpu_clock*, etc.) from this GPU. The reasoning behind selecting only discrete GPUs was that instances with no matches from the new dataset were hypothesized to have above average GPUs which is not the case for non-discrete GPUs such as Intel.

For the rest of the variables kNN imputation with 10 nearest neighbours proved to be the best technique for missing data handling.

## Multicollinearity and Feature Selection

The last step in our data preprocessing was multicollinearity checking and specification of the final feature space. As indicated in Figure 2, some of the explanatory variables were highly or moderately correlated which might cause problems with multicollinearity. For Parallel Random Forests, which we used as our final modeling technique (see Section on Modeling), correlated features could hinder final interpretability and cause misinterpretation, thus it was important to eliminate any multicollinearity between explanatory variables.

As a result, we performed VIF (Variable Inflation Factor) diagnostics on numerical variables. Note that VIF lower than 10 indicates no multicollinearity. The results in Table 2 indicate that variable *gpu_memory_clock* had the highest VIF (41.623) followed by *discrete_gpu* (VIF = 27.228), *gpu_clock* (VIF = 12.718) and *gpu_memory* (VIF = 10.760). After removing the *gpu_memory_clock* variable we recalculated the statistic to see how it changed for other variables (Table 3). The *discrete_gpu* still had a high VIF value of 14.316, however the inflation factor for other variables decreased significantly with the only other variable having VIF above 10 being *gpu_clock* (VIF = 10.168). Thus also we removed the *discrete_gpu* variable. After recalculating VIF again all variables were within satisfactory bounds and no further variable removal was required. As an additional check, the mean VIF was calculated (without the removed variables) and checked to be not much larger than 1, indicating no multicollinearity.

*Table 2. VIF Statistic for Numerical Variables*

| Numerical variable | screen_ size | pixels_y | touch screen | detachable_ keyboard | discrete_ gpu | ram | ssd | storage | weight |
|---|---|---|---|---|---|---|---|---|---|
| **VIF** | 4.085 | 2.162 | 1.635 | 1.302 | 27.228 | 4.233 | 2.767 | 2.199 | 4.690 |

| perks | gpu_ clock | gpu_memory_ clock | ghz | gpu_ memory | gpu_ shaders | gpu_tmu | gpu_rop | cores | brand_ premium |
|---|---|---|---|---|---|---|---|---|---|
| 1.339 | 12.718 | 41.623 | 1.869 | 10.760 | 1.753 | 1.719 | 1.574 | 2.953 | 1.285 |

*Table 3. VIF Statistic for Numerical Variables without gpu_memory_clock*

| Numerical variable | screen_ size | pixels_y | touch screen | detachable_ keyboard | discrete_ gpu | ram | ssd | storage | weight |
|---|---|---|---|---|---|---|---|---|---|
| **VIF** | 4.078 | 2.164 | 1.635 | 1.300 | 14.316 | 4.217 | 2.766 | 2.187 | 4.690 |

| perks | gpu_ clock | ghz | gpu_ memory | gpu_ shaders | gpu_tmu | gpu_rop | cores | brand_ premium |
|---|---|---|---|---|---|---|---|---|
| 1.334 | 10.168 | 1.856 | 8.771 | 1.675 | 1.690 | 1.573 | 2.950 | 1.275 |

The final feature space consisted of the following 21 variables: *brand*, *screen_size*, *pixels_y*, *screen_surface*, *touchscreen*, *detachable_keyboard*, *os*, *ram*, *ssd*, *storage*, *weight*, *gpu_clock*, *ghz*, *thread*, *gpu_memory*, *gpu_shaders*, *gpu_tmu*, *gpu_rop*, *cores*, *brand_premium* and *perks*. As mentioned before, other variables were removed due to having too many categories, containing information that is already included in other variables or multicollinearity.

As a final note, the entire data preprocessing was also performed on the test dataset. However, missing value imputation was done using the training dataset to avoid any data leakage.

## Modeling

Due to the substantial number of categorical explanatory variables, we decided to fit the data using Extreme Gradient Boosting, Stochastic Gradient Boosting and Parallel Random Forest, and compare their performance using 5-fold cross-validation. The latter was used to tune model parameters, test the model's ability to predict unseen data, generalize and indicate problems of overfitting and selection bias [1]. Since cross-validation automatically splits data to training and validation sets, we decided to use the whole given training dataset for model tuning and compare model accuracies using leaderboard. Mean absolute error (MAE) was selected as the accuracy metric, since it was used to rank models in the leaderboard.

At first, the default method of repeated cross-validation with 5 folds and 3 repeats was used (*trainControl* function with parameters *method* = 'repeatedcv', *number* = 5 and *repeats* = 3). The MAEs for Extreme Gradient Boosting, Stochastic Gradient Boosting and Parallel Random Forest using different outcome variables (minimum and maximum price versus minimum price and price range) and different combinations of missing value imputation methods (only kNN versus only imputation using median / mode / our method for GPU variables versus a combination of both) are shown in Table 4. In all four cases Parallel Random Forest performed the best in terms of test set accuracy and the best combination of outcome variables was minimum price and the newly created price range, while the best missing value imputation method was a combination of kNN and our popular GPU method for GPU details. Thus further parameter tuning was carried out using Parallel Random Forest with the mentioned specifications on outcome variables and imputation methods.

***Table 4.*** *Comparison of Test Set MAE for Extreme Gradient Boosting, Parallel Random Forest and Stochastic Gradient Boosting*

| | **Model** | | |
|---|---|---|---|
| | **Extreme Gradient Boosting** | **Parallel Random Forest** | **Stochastic Gradient Boosting** |
| **min_price + max_price + combo of imputation methods** | 351.874 | 342.905 | 357.110 |
| **min_price + price_range + combo of imputation methods** | 339.900 | 337.846 | 349.958 |
| **min_price + price_range + kNN imputation** | 345.228 | 343.189 | 353.945 |
| **min_price + price_range + no kNN imputation** | 353.668 | 345.113 | 356.416 |

The next step was random forest hyperparameter (number of decision trees and number of random features considered by the tree at each node split) tuning. Even though random forests in general do not face the risk of overfitting and thus can use any number of trees, we observed that in our case the number of trees still affected model error. Thus we decided that instead of specifying this parameter blindly it is better to select it according to model MAE. Considering that the *caret* package in R for cross-validation does not allow to tune the number of trees for Random Forests automatically, we tuned this parameter manually. Using the results shown in Figure 3, 300 was selected as the best value for this parameter.

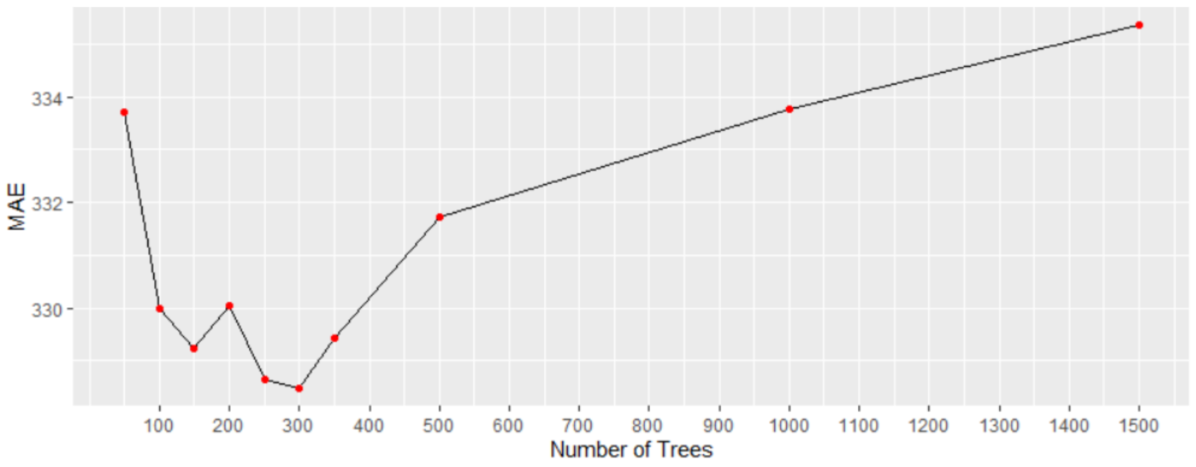


***Figure 3.*** *Mean Absolute Error for Different Number of Trees*

The number of random features considered by the tree at each node split (let us call it $M$ for convenience) was tuned using cross-validation (*caret* package) with the number of trees specified as 300. However, the default cross-validation only considers 3 random candidates for $M$, therefore we also tried grid search and random search to check more candidate values for $M$. Using the first one it was possible to specify the grid of values for $M$ to be considered (we specified range from 1 to 25), while the second one tried a specified number (in this case, 25) of random candidates for $M$. The results for all three methods can be seen in Table 5. As expected, the random search performed the worst, since it blindly tried random values and in this case did not result in a lucky hit. Grid search performed a bit worse than default cross-validation, however, it was possible that the default search performed better only on the given test dataset (public leaderboard) and that grid search would generalize better on unseen data, that is, on the hidden leaderboard. Thus we decided to use the $M$ parameter value specified by the cross-validation using grid search, $M = 2$.

***Table 5.*** *Mean Absolute Error on Test Set using Different Cross-Validation Methods*

| | **Default Search** | **Grid Search** | **Random Search** |
|---|---|---|---|
| **MAE** | 328.488 | 334.649 | 342.932 |

## Final Model: Interpretation

The final model that was selected for the leaderboard was Parallel Random Forest with 300 trees and the feature subset considered at each split of size 2. Since Random Forests in general are a black-box technique, *Boruta* package was used to determine feature importance in the final model. It does so by creating the so-called "shadow attributes" through random shuffle of the original attributes and ranks the latter by comparing them to the former using Z-scores. The technique is iterative and very robust.

The most influential features for the price range and minimum price can be seen in Figures 4 and 5, respectively. It is interesting to note that minimum price prediction was influenced by almost all of the features except for one, while price range prediction was influenced only by 13 out of 21 features. For both outcome variables *detachable_keyboard* was unimportant, while *gpu_clock* and *storage* were equally important. On the other hand, minimum price was mostly influenced by *ram*, *pixels_y*, *ssd* and *thread*, while range was mostly influenced by *gpu_memory*, *cores*, *brand_premium* and *weight*. It seems that the minimum price is predicted by variables which are necessary to satisfy the needs of an average computer user. RAM, SSD and CPU with threading are most important for smooth operation of a laptop, while *pixels_y* show that users also consider high image quality as an important requirement for their laptop. On the other hand, price range is predicted by variables which are mostly associated with high-end laptops. GPU memory and CPU core counts are very important for resource intensive tasks (such as video games), while low weight and known brands usually also result in a higher price. We also utilized *varImp* function which produced similar results.

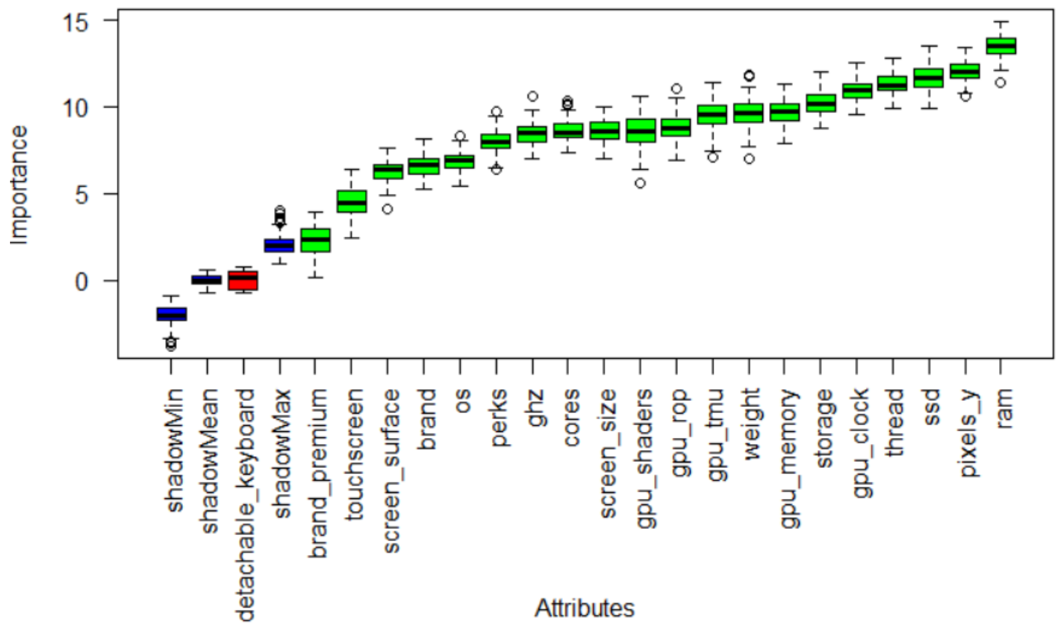


***Figure 4.*** *Feature Importance in Predicting Minimum Price using Final Model (Green = Important, Red = Unimportant)*

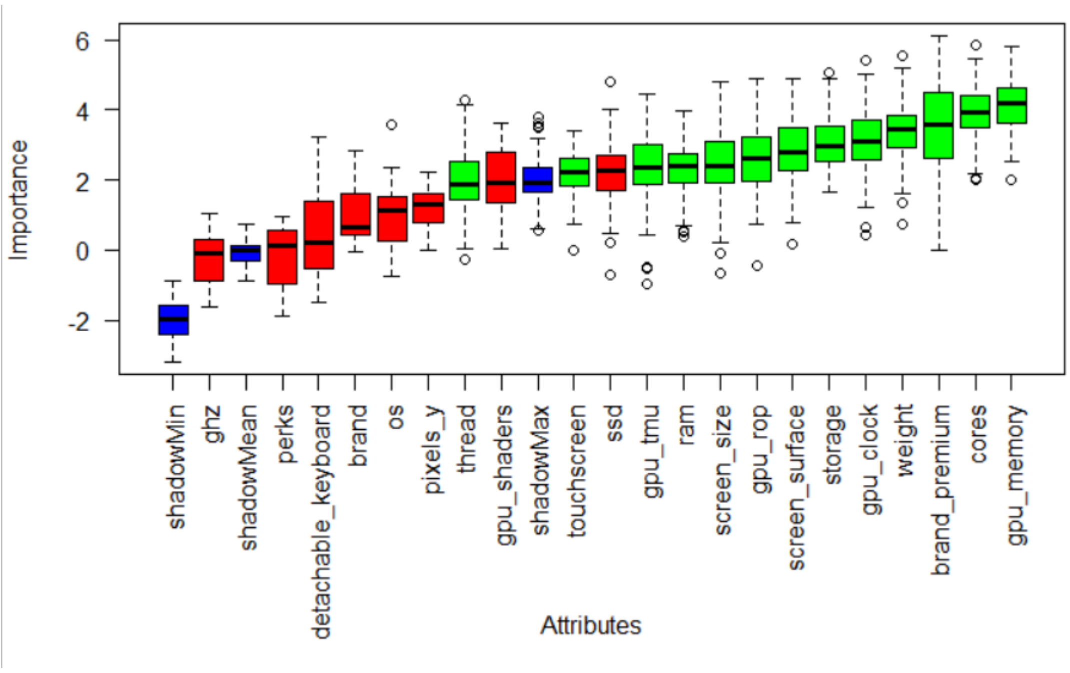


***Figure 5.*** *Feature Importance in Predicting Price Range using Final Model (Green = Important, Red = Unimportant)*

## Final Model: Evaluation

The results after the hidden leaderboard was revealed (Table 6) showed that the MAE score on the hidden part of the test set was higher by almost 25% than the public score, indicating overfitting on the first part of the test set.

***Table 6.*** *Leaderboard Results*

| | **Public Score** | **Final Score** | **Number of Tries** |
|---|---|---|---|
| **MAE** | 334.649 | 415.897 | 53 |

One of the possible reasons for overfitting was the metric used for model evaluation (MAE). It penalizes residuals linearly, which means that a single large residual is not worse than a couple of small ones. RMSE (root mean squared error), on the other hand, gives higher weight to residuals which are large, making them even more undesirable. This implies that using RMSE instead of MAE could lead to a model which is less likely to make huge mispredictions. In order to test this hypothesis, we compared our final model evaluated using MAE metric and the same model evaluated using RMSE. The model predictions were made on the full test dataset with labels, provided after the final leaderboard reveal. However, the model with RMSE did not have a significant improvement over the MAE model (error of

373.68 vs 374.86, respectively). Moreover, the presence of large residuals was not reduced. Therefore, we concluded that a different choice of evaluation metric would not have made a difference in prediction accuracy.

## References

[1] Cawley, Gavin C.; Talbot, Nicola L. C. (2010). "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation" (PDF). 11. Journal of Machine Learning Research: 2079–2107.