Project Stage 2

Group number: *02*

Activity Code: *08*

Topic and research question

This data analysis focuses on further exploration and analysis over laptop prices on the online retail market. The primary objective is to understand how the different components and characteristics of the laptop may influence the price, based on that build a model that can allow the prediction of laptop prices based only on these attributes.

Researcher Question

Which are the factors and attributes that determine the prices of laptops in online markets? And how can we predict the prices based only on the attributes?

Problem Statement

Laptops are a highly involved purchasing decision, users usually inform themselves about the different components and prices of the laptop before buying one. On top of that, different types of users will prefer different types of components and prices.

For example, a laptop for studying purposes is smaller and light weight with long durability on the battery while a gamer user might prefer a last generation laptop, with better graphic card, big screen and high resolution. With a lot of competencies in the industry it is important for users and retailers to understand what drives a laptop's price too so they can set competitive prices on the retail and take informed decisions on the user side.

Significance and stakeholders

Retailers need an accurate pricing model for each product that allows them to stay competitive within the industry and maximize their profits. Understanding the attributes that affect laptop prices, retailers can refine their pricing strategies. This may also be helpful to define categories of product with different price modelling based on the correlation between difference attributes and laptops differences.

Actionable Insights and Decision-making

Developing a reliable predictive model will lead to a proper forecasting of the laptop prices based on attributes like CPU, RAM, Storage type, etc. With this Retail will use the insights to set competitive prices in the industry. The consumer will be provided with a clear understanding of how components influence price, this will lead to an informed purchasing decision.

Dataset

The cleaned dataset of "Laptop Prices Online Cleaned" present 6492 rows and 26 columns. Some of the columns (attributes) are integer or float for further numerical analysis including; Average Rating, Total Rating, Retail Price, Display Size, RAM Size (GB), SSD Storage (GB), CPU Speed (GHz) and Year of Release, 9 in total. Other 6 correspond to Boolean attributes mostly referencing upgrades or additions of laptop features, for instance; microphone, webcam, etc. Finally, the rest of attributes are string that extend the detail of laptop features; Manufacturer, Color, Country of manufacture, etc.

The main challenges we can observe on this dataset are 3:

- **Missing data:** Handling the missing data is important before running the model. Could either be dropping the NaN values or imputing missing data replacing it with mean or median. We can observe that **47.8%** of our dataset contains NaN values where some of the attributes have high volume of NaN values such as: Country of Manufacture **97%** NaNs, Average and Total rating **95.5%** NaNs, Year of Release **88%** NaNs.
- **Class Imbalance**: This may lead to affect the performance of the model when some attributes have more listing than others, meaning that they are overly frequent values which could skew predictions. For example, attributes

like “Product Condition” can lead to an imbalance because the most frequent attribute is “New” representing **81%** of the cases. Another example is CPU brand where the most frequent attribute is “Intel” with **93%** of the cases. Finally, “Storage type” with **78%** of the cases being “SSD”

- **Categorical Features**: Almost half of the attributes presented on our dataset are categorical values. Since this attribute represents groups or labels, we need to transform them into numerical inputs so that the machine learning algorithm can interpret the values. For instance, Manufacturer, Product Condition, Country of Manufacture, Storage type and Color are some of the attributes that need to be encoded.

Impact on the model

The challenges presented on the dataset might highly impact the model accuracy and the interpretability. For instance, imputed data with methods like filing with median() or mode().iloc[0]) for the most frequent values can obscure the importance of the relationship between features and the target variable (‘Retail Price’). The opposite, not imputing data and dropping most of the null values may lead to fail in generalize the model due not enough information.

The class imbalance may impact the model performance and predictions. For example, if less common attribute values are highly likely to impact the target attribute, due to its low frequency the model might fail to recognize them.

Lastly, categorical features are the majority on the dataset, this might lead to a complex model to analyze. Since in most of the cases the categorical features need to be transform into numerical or use methods like one-hot encoding to help the model process to interpret these values, this might lead to either a potential overfitting or increase the complexity in terms of computational work needed, reducing model interpretability and accuracy.

Setup

Modelling agreements

Since the attribute to analyze is ‘Retail Price’ and our goal is to predict such attribute based on the dataset information with continues values we choose to use metrics; **RMSE** (Root Mean Squared Error) which measure the errors of the differences between the actual and predicted values. RMSE is appropriate since it indicates a prediction of errors magnitude since we want to minimize the deviation from the true prices of laptop on our model.

We will also be using **R-squared** to explain the proportion of variance in ‘Retail Price’. Since it is ideal to measure how well our model fits the data in a range of 0 to 1. If the model is closer to one that means that the retail price depends on the big magnitude of the features on the dataset.

Data division

The data split into **80%** training set and **20%** test set. This approach ensures enough data for training and a reliable estimate of model performance using the test set. This split usually provides a good balance between enough data to train and to test to achieve meaningful results and is also used as a common proportion in data science.

Predictive model

We will be using XGBoost(extreme Gradient Boosting) which is derived from the Grading boost algorithm, family of the ensemble model. The ensemble models are basically models like; random forest, extra trees, gradient boosting, etc. that are generated based on many individual models. Gradient Boosting will gradually minimize the loss function while weights are optimized. How is this done? While building the weak learners the algorithm compares the predictions with the actual values, the difference will be the error rate of the model, used to calculate the gradient. The gradient will be used in the next round to reduce the error rate. XGBoost (eXtreme Gradient Boosting) can be used for both classification and regression problems. This one combines multiple weak learners (small decision trees) into a strong one (W. Zhang, 2023).

Strengths and limitations

XGBoost is faster than gradient boosting because it uses **parallelization** on the tree's construction. XGBoost is suitable for the research question because the relationship between Retail Price and the features is likely non-linear and complex (due to its diversity). Since the dataset contains both numerical and categorical variables, XGBoost has the quality of handle both types, which makes it a great fit for the context of the dataset. Additionally, XGBoost can manage the overfitting that may be caused by the noise generated during the preprocessing or before and missing values, making the data preparation (pre-processing) less time consuming.

Some on the limitations of this model is the complexity of the many hyperparameters that contains (learning rate, number of trees, tree depth, etc) it is required to find the right combination of this parameters to get a better model which might add complexity to its development. Also, while decision trees are easy to interpret, the boosting process makes it harder to understand how its decision during the process is made. Finally, a limitation is the large memory that this model requires (W. Zhang, 2023).

Model algorithm

The principles of this method are building trees one by one and then using the residuals of the previous tree are used to train the subsequent model. During the training the model added values of previously trained trees to achieve the best outcome (W. Zhang, 2023).

Mathematical equations

The learning objective of this model it contains two: Loss Functions and regularization term (Wade, 2020):

$$obj(\theta) = l(\theta) + \Omega(\theta) \qquad (1)$$

Where $l(\theta)$ is the loss function, which is basically the mean squared error (MSE) and $\Omega(\theta)$ is the regularization function which prevents over-fitting and reduces model complexity. As per the MSE, the $l(\theta)$

$l(\theta) = \sum_{i=1}^{n}(y_i - \overline{y_l})^2$ (2) where y_i is target value and $\overline{y_l}$ is predicted value for row i^{th}. Since we know this is an iterative process of generating trees, the final tree is the previous tree model plus its newly generated tree model and can be expressed as:

$\overline{y}_i^{(t)} = \overline{y}_i^{(t-1)} + f_t(x_i)$ (3) where t is the total number of base tree models.

Replacing (3) into (2) and (2) into (1) we will end up with this equation:

$$obj^{(t)} = \sum_{i=1}^{n}\left(y_i - \left(\overline{y}_i^{(t-1)} + f_t(x_i)\right)\right)^2 + \Omega(f_t)$$

We can approximate with Taylor polynomial to obtain (Exchange, 2019):

$$obj^{(t)} \approx \sum_{i=1}^{n} g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 + \Omega(f_t)$$

Where and g_i are h_i the first and second derivatives of the loss function. This means that most of this expression can be computed only **once** and reused as constant for all the rest of the trees on the model, we will only need to compute $f_t(x_i)$ or cost function and $\Omega(f_t)$. For the regularization term function or $\Omega(f_t)$, we can express it like this:

$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$, where γ and λ are penalty constants to reduce overfitting.

T is the number of leaves, w is the corresponding weight of the leaf.

After substitutions and derivations, we will end up with the result of the XGBoost functions that used to check how well the model fits the data:

$$obj^{(t)} = \frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

Hyperparameters

One of the limitations (or advantages) of XGBoost is the great amount of hyperparameters that can be customized (XGBoost, 2024). Here some of them:

- **N_estimator:** Is the number of trees in the ensemble, can be from 1 to infinite and increasing this may improve the scores in large data. Default=100.
- **Learning_rate:** Shrinks the tree weight each round of boosting, can be a value between 0 to infinite. Decreasing this value prevents overfitting. Default =0.3.
- **Max_depth:** The depth of the tree can be from 0 to infinite. Decreasing this value prevents overfitting. Default = 6.

The list keeps going, but these are the most important ones. We will be using *objective='reg:squarederror'* that is our MSE for the regression model problem in the XGBRegressor.

Step-by-Step pseudocode

1. *For tree in a range t=1 to **N_estimator***
2. *Residual = Real value – Predicted value*
3. *Fit the decision tree to the **Residual***
4. *Update model with the new tree weighted by the **Learning_rate***
5. *Stop when max **N_estimator** is reached, or errors converge*
6. *Return final model*

Model development

For the model development we start preparing our data through **preprocessing** and **feature engineering** techniques such as handling missing values by *.dropna()* function. Split attributes into features (x) and target variables (y). Features attributes are the ones selected to explain our target variable 'Retail Price'. To extract them we just use *.drop('Retail Price')* function leaving all the features on (x). Although, since our model can handle categorical values and numerical values at the same time, transform all our categorical columns that are tagged as 'object' datatype, due to its implementation, to 'category' data type through the *.astype('category')* method. Then we need to enable the *enable_categorical=True* to allow category data type on our XGBoost model.

On the other hand, for the numeric features, we will need to escalate them to avoid features disproportions that might affect the model. To do this we use *StandarScale()* function to ensure that all numeric features have a similar range. This step is crucial to ensure standardized scale. On our model we apply: *.fit_transform()* to the data we are going to train (x_train), *.fit()* ensures that the model learns the parameters based on mean and standard deviation and *.transform()* apply the learned parameters to the data for scaling. **We only apply fit to the training set** to keep the same scale for the testing data, that's why we only apply *.transform()* method on the data we going to test (x_test), ensuring a coherent scale of measurement based on the trained data. Will be as follows (we are using *Scaler = StandardScaler())*

- X_train_scalated = scaler.**fit_transform**(x_train # for the numeric values to train)
- X_test_scalated = scaler**.transform**(x_test # for the numeric values to test)

Finally, we combine both scalated numeric and catgorical features to form the final dataset for our model.

Next step is to apply the right **functions** and **hyperparameters** to the model. The hyperparameters chosen are based on the nature of the dataset while trying to prevent overfitting. **Objective**:**'reg:squarederror'** since the problem case is a regression prediction a continuous value 'Retail Price', **N_estimator**: **'300'** corresponds to the boosting round (Trees), improve the accuracy but lead to overfitting, 300 is a balance between both. **Learning_rate: '0.1'** controls how much each tree contributes to overall model, using a small rate ensure the update is slow and controlled to reduce overfitting risk. **Max_depth: '6'** this limits the depth of each decision tree, deeper trees capture more complex patterns but lead to overfitting, 6 is a balance value between complexity and overfitting. As mentioned above, we also apply **'enable_categorical=True'** to allow categorical data types to our model. Once our XGBoost model hyperparameters are assigned we use the training data to train using *.fit()* function as *xgb_model.fit(X_train, y_train)*. The model will start executing based on the training set learning the relationship between the features and target variable trying to minimize the loss (MSE) adjusting the internal parameters mentioned above.

Model evaluation and optimization

The metrics we used to evaluate our model are Root Mean Squared Error (RMSE) and R-Squared. Through our XGBoost we observe a *RMSE* on the test data of **209.2**, this measure the error magnitude, the 'Retail Price' mean is **255.5**, then the RSME is **82%** of the mean, representing a high -moderate error, but its standard deviation is **219.1**, since the RMSE is lower than the standard deviation of the Target variable is a good result.

For the training RMSE we have a value of **137.45**, which is much lower than the test RMSE, this might indicate an overfitting degree where the model performs better on the trained data than the unseen data.

We also have a **R-squared** of **0.74** on the test set, this one explains how much the variance in 'Retail Price' is explained by the features of the model and **0.26** of the variance remain unexplained. While the r-squared of the training set is **0.88**, higher than the test set. Again, this might be explained by overfitting. Once we observe the predicted values, we can see that the actual model is performing relatively good, but there is still room for improvement. The first 10 predicted values compared with the real values there are cases where: The actual price is **$1030**, the predicted value is **$1029**, but there are other cases where the predicted value is as far as **$219** when the actual value is **$721**, indicating struggle in cases for the prediction to be accurate.

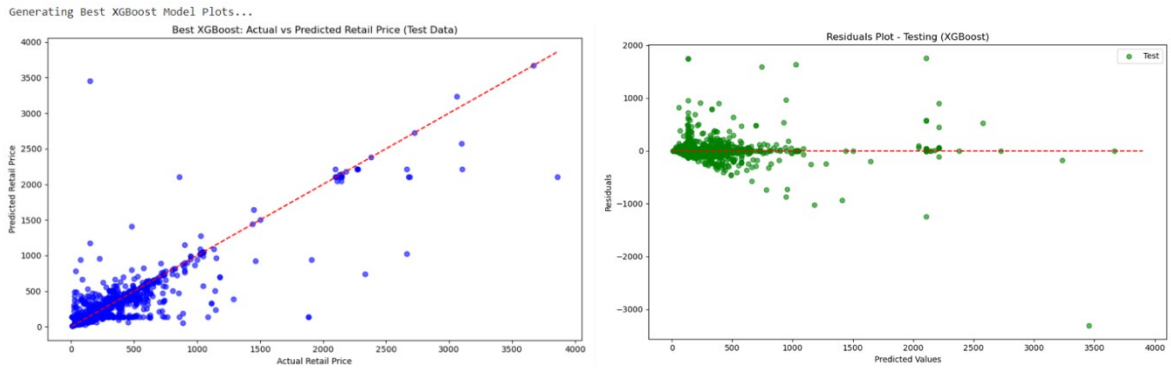

Figure 1: Actual data vs test data (Left) & Residual plot on test data (Right)

In Figure 1 we can observe that prices within the range $0-$1000 are concentrated close to the predictor line, once the values start growing to higher prices we can observe more dispersion, that might be due to less information or less features values that can explain this cluster. The second plot will show us the errors made by the model. We can observe a similar behavior, where errors are agglomerated closer to 0 at the beginning (with lower prices) but once it starts growing there are more errors/further from 0 line.

An extension of the analysis is an **R-squared cross-validation**. Among the 10-fold cross validation, the scores run from the ranges of **0.47** being the lowest and **0.94** being the highest. Most of the values are above **0.7** indicating consistency of the model between training and validation sets. On top of that, with the Feature importance analysis we can obverse the attribute with more impact on the model, being the 'Graphics card' being the most important for the model impact.

Model optimization

For the model optimization we will customize the model hyperparameter to find the best combination. To achieve this, we will use **Randomized Search CV**, since it is faster and more efficient than **Grid Search CV** when dealing with large numbers of hyperparameters. Even when it does not guarantee to find the optimal solution as Grid Search (since this one explores every possible solution) it is a good approximation. We will be sorting **learning_rate** in {0.05, 0.1, 0.2 ,0.3 and 0.4} values, **max_depth** in {3, 4, 5, 6, 7} and **n_estimator** in {75, 100, 200, 300, 400}. We will be testing over 50 random combinations among these hyperparameters customization which goal will be to reduce the MSE as much as possible for the best solution. Also, the test will be split into 5 parts, where 4 will be used for training and 1 for testing.

Observing the result, the possible best solution resulted is for the hyperparameters is: {'n_estimators': 300, 'max_depth': 3, 'learning_rate': 0.2}. We can tell that there was a slight improvement on the tested RMSE, from **209.2** originally to **203.1** on the Randomized Search. The same behavior happens on the r-squared with an improvement from **0.74** to **0.75**. While the Randomized Search CV helped to improve our model customization, there was no significant impact on the results, this might be due to the original model is good enough based on the dataset features. It is also possible that the dataset has some intrinsic limitations (like noise or major complexity that cannot be captured by the model).

Predictive Model

1.1. Model description:

Random Forest is an algorithm trademarked by Leo Breiman and Adele Cutler which combines multiple decision trees to reach a single result. This reduces one of the most common issues about using decision trees in machine learning, which is its tendency to overfit and for biases. When multiple decision trees are combined to form an ensemble like random forest, we gain improvements in accuracy and generalization (IBM, 2024). The random forest can be seen as an extension of bagging as it makes use of bagging and feature randomness in the creation of uncorrelated forest of decision trees.

Strengths:

- Random Forest can model complex relationships between target variable and features even if they are nonlinear.
- Random forests can provide important data which helps us in understanding the underlying patterns.
- The model also tends to have higher accuracy and is flexible and robust with the ability to handle both numerical and categorical data.

Weaknesses:

- There is less interpretability since no single figure can explain the predictions. However, we get valuable information such as feature importance to help with analysis.
- Random forest can be computationally expensive when working with large datasets since it requires a lot of memory.

1.2. Model algorithm:

Random Forest operates on the principles of **ensemble learning** and **bagging**, where multiple decision trees are trained on random subsets of the data and features. It then aggregates the predictions from all the trees to provide a final output.

Key Principles:

1. **Bootstrap Aggregating (Bagging)**: Each tree is trained on a randomly sampled subset of the training data with replacement.
2. **Random feature selection**: A random subset of features is considered at each node for the best split, preventing individual trees building correlation.
3. **Voting/Averaging**: After multiple decision trees are build the final output is the mean of all predictions.

Algorithm Execution: The Random Forest algorithm can be described as follows:

1. **Initialize**: Set the number of decision trees (n_estimators).
2. **Bootstrap sample**: For each tree, create a bootstrap sample from the training data.
3. **Tree creation**:
   - For each node, randomly select a subset of features.
   - Find the best split based on a criterion (e.g., mean squared error).
   - Repeat until the stopping condition is met (max_depth or min_samples_leaf).
4. **Prediction**: Once all trees are built, predict the target variable by averaging the predictions of all trees for regression.
5. **Feature importance**: Calculate feature importance based on the average reduction in impurity across all trees for each feature.

Recent Advancements:

There have been some recent improvements to Random Forest including **ExtraTrees**, which increases randomness in splitting criteria, leading to faster training times and often improved accuracy and **Quantile Random Forest**, which provides more quantile estimates for regression insight into prediction uncertainty by offering quantile estimates for regression —relevant for uncertain datasets like pricing.

1.3 Model Development

Data preprocessing:

- **Removal of Irrelevant and High Null Columns:** Columns having null value % and less relevant in pricing.
- **Handling missing values:** For categorical columns the missing values were filled with mode while median was used to fill missing values for numerical columns.
- **Datatype Conversion:** Boolean column was converted to integer for better compatibility with random forest and easier interpretation and consistency.
- **Feature Engineering:** 2 new categorical features and 3 new numerical features were introduced CPU_RAM_Interaction, CPU_SSD_Interaction these two interactions combine cpu with ram size and ssd respectively creating combinations that help the model learn specific combinations that influence the target variable.

  SSD_x_RAM, SSD_x_CPU, CPU_x_RAM are made by using multiplicative interaction aiming to capture any nonlinear interactions the individual features might miss.
- **Encoding Strategy:** one-hot encoding of categorical variables and alignment of training and testing datasets was done to ensure consistency in features being represented and manage cardinality.

Base Model Training: The Random Forest Regressor was then initialized with fixed random state of 42 for reproducibility. The model was trained on the encoded training data, and initial predictions were made on both training and testing sets. Evaluation metrics were computed to assess performance, revealing an RMSE of approximately 143.3471and an R^2 of 0.8705 on the test set and RMSE: 182.7278 and R^2: 0.8033 the training set. Showing that the model has a very strong fit on both the training and testing data. The difference in R^2 between the training and testing data is small showing that the model can generalize successfully but has slight overfitting.

Model Hypertuning: Grid search was chosen to do the model tuning with three-fold cross validation offering a balance of performance and generalization on unseen data, along with the following **Hyperparameters:**

- n_estimators ([100, 200, 300, 400]): Number of trees in the forest. As the number of trees increases performance generally trends up aswell, this grid allows them model to find a balance between performance and computational cost.
- max_depth ([10, 20, None]): Maximum depth of each decision tree. This grid allows the model to test different depths for the best result. The shallower the depth the better the generalisation.
- min_samples_split ([2, 5, 10]): Minimum number of samples required to split an internal node. This prevents the model form splitting leaves with very few samples. The higher the sample split the better the generalisation
- min_samples_leaf ([1, 2, 4]): Minimum number of samples required to be at a leaf node. This prevents the model form creating leaves with few samples. Higher numbers can help reduce the complexity of the model and overfitting.
- max_features (['sqrt', 'log2', None]): this Parameter controls the number of features when looking for the best split sqrt is the default and helps maintain the model performance while reducing overfitting.
- bootstrap: Whether to use bootstrapped samples when building trees. It allows the trees to sample with replacement if toggled on.

These hyperparameters were chosen to help find the right balance between overfitting and underfitting and performance costs. By controlling the model complexity generalization, and computation time.

2. Model Evaluation and Optimization

2.1 Model Optimization

GridSearchCV was used to fine-tune the model's hyperparameters. Grid Search was chosen over Random Search since it exhaustively searches the grid for the best hyperparameters systematically exploring every possible combination.

The parameter grid explored included varying numbers of estimators, maximum depths, minimum samples for splitting and leaf nodes, and bootstrap options. The grid search process systematically evaluated combinations of these parameters through cross-validation, identifying the configuration that minimized the **negative mean squared error**. This optimization enhanced the model's performance by balancing bias and variance, leading to more accurate and generalizable predictions. The selection of optimal hyperparameters was validated through improved evaluation metrics on the test set.

The following hyper parameter grid was used *n_estimators'*: [100, 200, 300, 400], *'max_depth':* [10, 20, None], *'min_samples_split'*: [2, 5, 10], *'min_samples_leaf'*: [1, 2, 4], *'max_features'*: ['sqrt', 'log2', None], *'bootstrap'*: [True, False] }

After running the grid search, the following hyperparameters were found to provide the best performance:**n_estimators**: 300, **max_depth**: none ,**min_samples_split**: 2, **min_samples_leaf**: 1, **bootstrap**: false, **'max_features'**: 'log2'

These settings were selected because they resulted in the lowest RMSE and the highest R-squared value during cross-validation. The n_estimators value of 300 was chosen because it balanced predictive performance and computation time, while the max_depth=none setting allowed for the trees to grow enough capture the complexity of the data without overfitting, as regularized by other parameters.

2.2 Model Evaluation

The model's performance was evaluated using RMSE, R^2, and Mean Absolute Error (MAE). On the test data, the optimized Random Forest model achieved an RMSE of 136.5396, an R^2 of 0.0.8111, and an MAE of 50.2959 on the training dataset and RMSE of 179.0652, R^2 of 0.8111 and MAE of 84.9505 on the testing data set suggests that the model explains approximately **81%** of the **variance** in the target variable and has a reasonable prediction accuracy. The residual plots indicated a random distribution of errors, affirming the model's reliability. However, the difference between training and testing RMSE values indicates that the model performs better on training data compared to unseen data which is a sign of overfitting as seen in figure 2.

Scatter Plot Analysis

The training plot in figure 2 closely follows a diagonal line indicating better fit than the testing data with more pronounced outliers. Especially for higher values. This indicates a difficulty faced by the model in evaluating laptops with higher prices as their costs are not entirely specs dependent and higher models are less represented in the dataset.

The Tighter clustering for the training data suggests that the model has learned some noise that is translating to prediction errors in the test data. The model still captures the overall trend in the test data except for extreme values mostly greater than **800**.

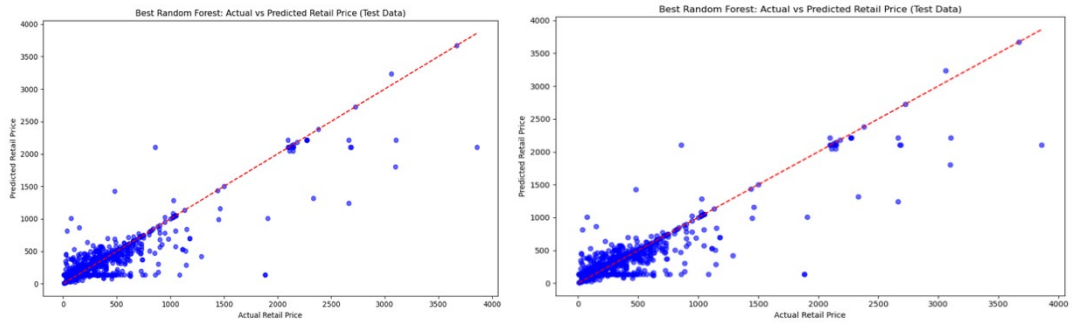

Figure 2 Training vs prediction Scatterplots for Training Data (Left) for Test Data(Right)

Residual Plot Analysis:

The residuals are mostly contained near the red dotted line/zero line indicating that the errors are randomly distributed with higher residuals at lower prices. The Spread also increases more and more and is less clustered as the value increases. As was seen in figure 2, figure 3 also shows that the model struggles at higher values to predict accurately depicting **heteroscedasticity**. The model seems to be performing great until the **500** mark after

which the spread of outliers grew particularly points with residuals between -200 and 150 can be seen. Investigating laptops with error data can help improve the model. The model fits reasonably well as most of the residuals are close to zero.

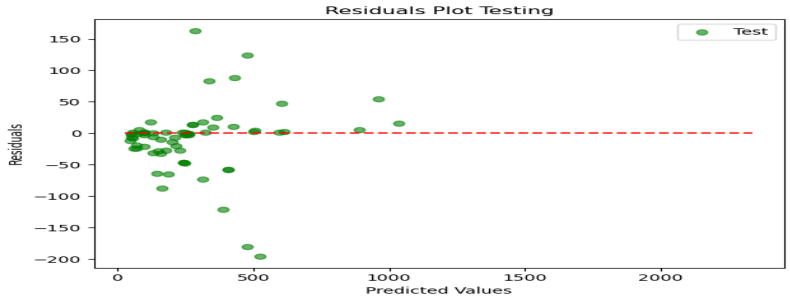

Figure 3 Residual plot Random Forest

Feature Importance :

Analyzing the feature importance and partial dependance of the random forest model as seen in figure 4 shows **Graphics Cards** as the most important feature from the partial dependence graph we can clearly in figure 4 that price increases depending on the type of graphics card. The top 3 graphics card models seem to be similar cards from intel these had significant contribution in predicting the retail price.

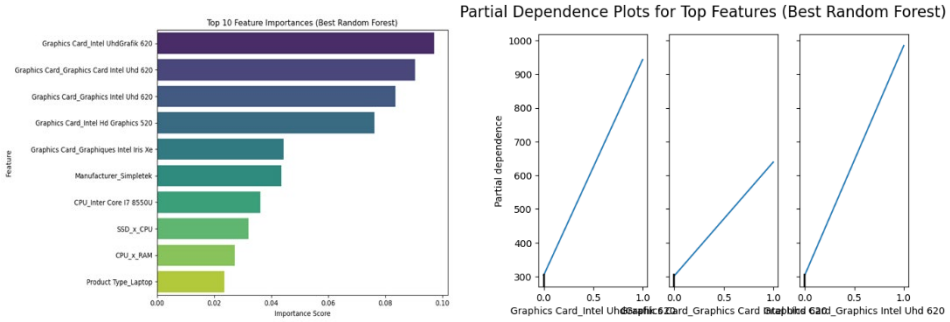

Figure 4: Feature importance plot with top 10 features(left) and Partial Dependance of the top 3 feature(right)

Future Improvements/Optimization:

- Attributes showing similar partial dependency and similar feature importance like the intel graphics card can be clubbed together as they are likely very similar models.
- Hyperparameter tuning with more efficient search methods, such as Bayesian optimization.
- Optimizing the data by analyzing real world data and improving the dataset by getting correct model names and specs of machines from company websites.

Discussion

XGboost Strenghts & Limitations

XGBoost model is highly effective in handling both datatypes; categorical and numerical features, making it suitable for this dataset. The characteristic of handling overfitting makes it useful dealing with non-linear relationships with data, likely the dataset in this case. It demonstrates being a good model that suits this type of data set, with a consistent high r-squared of 0.75 and RMSE of 203.

Despite its good predictive analysis, the conclusion is a slight overfitting over the model as demonstrated in the differences presented between the training set and testing set results, RMSE 137 for training and 209 for test before the optimization. It is also a complex model that requires a high computational cost and is likely to be less interpretative than Random Forest hyperparameters, such as, n_estimators, max_depth and learning_rate.

Random Forest Strengths & Limitations

Random forest offered a good performance and not-complex interpretation. It provided a high accuracy in terms of RMSE being 136.5 on the test set and r-squared of 0.81, where 81% of the variance in Retail Price is likely to be explained by the features in the model. The nature of Random Forest allowed us to handle complex scenarios with non-linear relationships being more effective than other less complex models.

Even when this model had the best performance among the 2 presented, still a small amount of overfitting was presented and shown on the differences between the training set and the testing set. Like XGBoost Random Forest also is relatively slow compared to simpler models and is less insightful

Quantitative and Qualitative comparison:

In terms of RMSE the Random Forest achieved a better value of 136.5 while XGboost achieved 203.1, being only a 66% of the error compared one with the other we need to consider the standard deviation for our Target Variable is 219, which is a closer value to XGBoost rather than Random Forest. While on the r-squared measurement we can observe a better accuracy on Random Forest the difference in not big being only 0.81 for the first and 0.75 for XGBoost. Both models explain relatively most of the variance presented in the Target Variables ‘Retail Price’.

Through the hyperparameter tuning we achieved the best performance model for both due to optimization methods like Random Search CV and Grid Search.

Through feature importance analysis it is easy to interpret which feature impact on the laptop the most, in this case were the ‘Graphic Card’ feature attributes that is more important feature in predicting laptop prices in both models.

Conclusion

While both models were effective, Random Forest model has the best balance of accuracy and generalization, making it the preferred model in answering our research question the models highlighted graphics card, manufacturer and cpu as key features in predicting the price of the laptops. Due to the low number of higher priced laptops and large number of null values the models struggled at accurately predicting prices of expensive laptops.

Random Forest model provided a clear insight into which attributes influence Laptop Prices the most. Through this analysis we discovered that key factors such as CPU speed, RAM size, SSD Storage, Year of Release, Graphic Card and GPU are drivers of laptop prices. All attributes combined can offer retailers and consumers a better understanding of how individual components and features affect the final retail price. Moreover, the high r-squared indicated that these attributes strongly influence the laptop price and can rely on these attributes for future prediction of prices. Even though laptops with a high market price are most likely to being harder to predict, since the Retail Price set is mostly a set of prices within the range of 100-500 dollars, it is a model to rely on when having a low or medium ticket price for laptops. With this Retailers will optimize their prices strategies and consumers will have a deeper understanding of how different attributes affect laptop prices.

References

Exchange, S. (2019, march 21). *Cross Validated*.

Retrieved from Stack Exchange: https://stats.stackexchange.com/questions/202858/xgboost-loss-function-approximation-with-taylor-expansion

W. Zhang, X. G. (2023). Comprehensive review of machine learning in geotechnical reliability analysis: algorithms, applications and further challenges. *Applied Soft Computing, 136*.

Retrieved from https://doi.org/10.1016/j.asoc.2023.110066

Wade, C. (2020). *Hands-On Gradient Boosting with XGBoost and scikit-learn.* Birmingham, UK: Packt Publishing Ltd. Retrieved from https://learning.oreilly.com/library/view/hands-on-gradient-boosting/9781839218354/

XGBoost. (2024, 10 15). *XGBoost.*

Retrieved from https://xgboost.readthedocs.io/en/latest/index.html

IBM. (2024, 10 15). What is random forest?

Retrieved from IBM: https://www.ibm.com/topics/random-forest