Predicting Laptop Prices

Emma Berman
Brown University
Github

1. Introduction

The Problem

Buying a new laptop has become an increasingly large project, requiring hours of research. Depending on a person's career and goals, they have specific needs for their laptop's features. A great laptop should last several years and function at a capacity that allows a person to complete their work efficiently. However, one of the most important factors when choosing a laptop to purchase is the price. A laptop can be quite expensive, so a consumer would want to be able to estimate how much the investment would cost ahead of time. The machine learning pipeline seeks to predict laptop prices based on its attributes.

The Dataset

A Kaggle user, Talha Barkaat Ahmad, uploaded the "Laptop Prices Dataset - October 2023", using laptops sold on Amazon. [3] The data is a regression dataset with 'price' as the target variable. The dataset contains 4441 rows (after dropping the five rows that were missing the price value) and 14 columns, including the price. The original dataset included all values as strings, so the numeric values were converted into continuous features. Of the 13 attributes, 9 of them are categorical: brand, model, color, central processing unit (cpu), operating system (OS), special features, graphics, graphics coprocessor, and cpu speed. The remaining 4 features are continuous: screen size, harddisk, random-access memory (ram), and rating.

2. Exploratory Data Analysis (EDA)

Feature Analysis

After converting some of the continuous features from strings into floats, exploratory data analysis was performed. Each row in the dataset represents a different laptop for sale on the

Amazon website. In order to better understand the dataset, I generated several figures to look at the relationship between different features and the target variable.

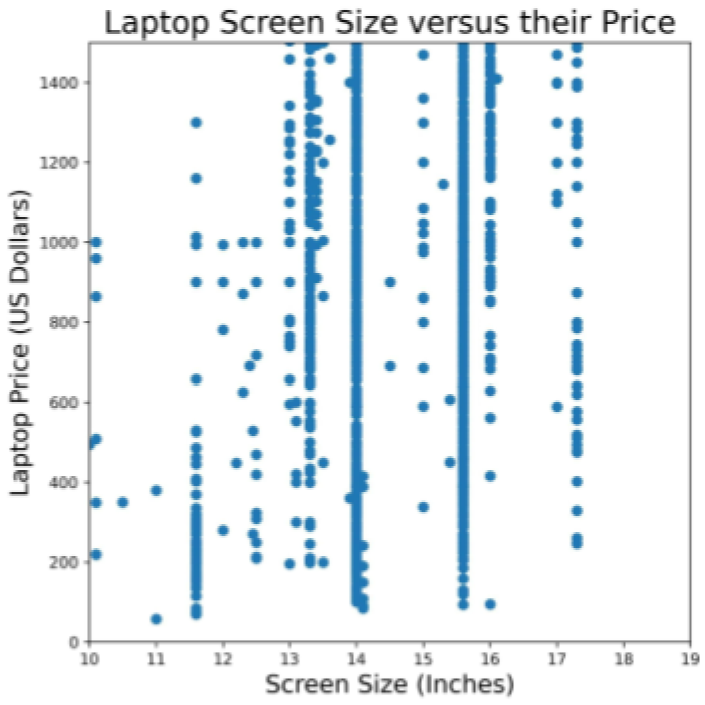

Figure 1: Zoomed-in scatter plot of laptop screen size versus the target variable, price.

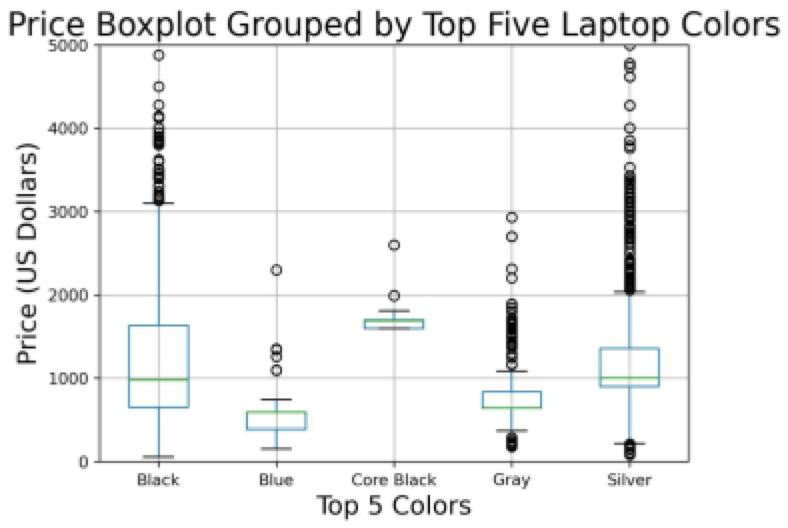

Figure 2: Boxplot of the top five laptop colors and their prices

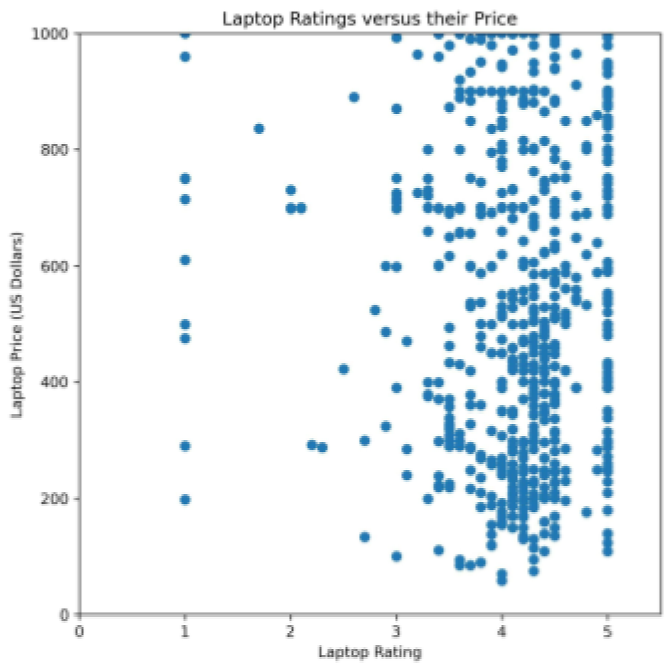

Figure 3: Zoomed-in scatter plot of each laptop's Amazon rating and its price

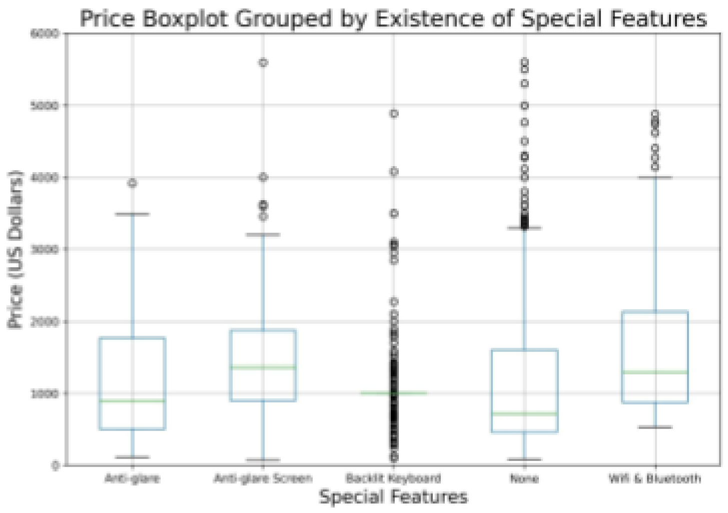

Figure 4: Boxplot looking at the affect the existence of special features has on median price of the laptop

The various features explored gave some insight into the relationships between each feature and the target variable. For example, it appears that the screen sizes have a direct relationship with price, as the highest price for smaller laptops is much lower than the maximum price for laptops with screen sizes of at least 13 inches. The top five color options and the Amazon ratings seemingly have no correlation with the price, except that laptops in the color "core black" have a higher median price than other colors, but this correlation does not translate to black laptops. Having no special features resulted in a slightly lower median laptop price.

The Target Variable

After looking at some of the other features, it was most important to broaden my understanding of the target feature, price. I used .describe to look more closely at the range of laptops prices appearing in the data. The average price was $1,189.33 with prices ranging from $57.99 to $11,261.24. The distribution revealed a skew to the right, as most laptops were priced under $1000.

count	4441
mean	1,189.33
std	826.02
min	57.99
25%	589.99
50%	999.99
75%	1,612.99
max	11,261.24

Figure 5: Summary of statistics on the laptop prices, each rounded to the nearest cent

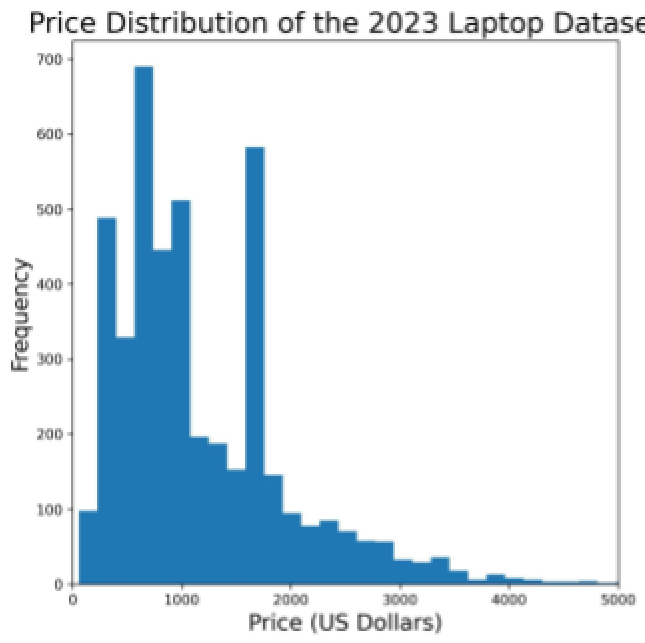

Figure 6: Histogram of the price distribution

Missing Values

The most challenging aspect with regard to the laptop data was the large magnitude of missing data. Every row and column had at least one missing value, except for the brand column. Therefore, the pipeline had to handle these values prior to initiating the predictive algorithms.

model	0.262
screen_size	0.007
color	0.130
harddisk	0.130
cpu	0.022
ram	0.014
OS	0.006
special_features	**0.538**
graphics	0.015
graphics_coprocessor	0.421
cpu_speed	**0.658**
rating	**0.511**

Figure 7: Fraction of missing values in each feature, highlighting those with over half missing, each rounded to the thousandth place

3. Methods

Splitting Strategy

For each algorithm, the dataset was initially split using a basic train_test_split with 20% of the data in the testing set, and the remaining 80% as other. For the Lasso, Ridge, Elastic Net, and Random Forest methods the other set was split using a KFold with four splits for cross validation. However, for XGBoost, I used another basic split, so that the entire dataset was 60-20-20 in the training, validation, and test sets, respectively. The data has no group structure, so a basic split is sufficient, except for when cross validation is necessary. The splitting each time used the same random states to help reduce the uncertainty of the pipeline.

Preprocessing

The dataset contains no ordinal features, so I only had to preprocess the categorical and continuous features. For the categorical features, I first used a SimpleImputer to address the missing values by creating a new category in each feature titled 'missing', then the OneHotEncoder standardized the values. For the continuous variables, the linear methods and Random Forest, an IterativeImputer using Linear Regression to impute the missing values was used and then a StandardScaler standardized the values for the machine learning pipeline. After preprocessing, the dataset had 1539 columns due to the wide range of categorical variables.

<u>Evaluation Metric</u>

Since this is a regression problem, I chose the root mean-squared error as the evaluation metric because I suspected the model would not have a high accuracy at predicting the exact price. Therefore, I prioritized how much the prediction differed from the true price to evaluate the algorithms.

<u>ML Pipeline</u>

I attempted five different algorithms and tuned various hyperparameters as shown in Figure 8.

Algorithm	**Hyperparameters**
Lasso	'model__alpha': [1, 10, 100]
Ridge	'model__alpha': [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2]
Elastic Net	'model__alpha': [1, 10, 100], 'model__l1_ratio': [0.1, 0.5, 0.9], 'model__max_iter': [1000, 2000, 5000]
Random Forest	'model__n_estimators': [30, 50], 'model__max_depth': [1, 5, 10]
XGBoost	"n_estimators": [5000], "max_depth": [1, 5, 10]

Figure 8: A chart describing the algorithms trained and which of their hyperparameters were tuned

Each algorithm used GridSearchCV to tune the hyperparameters and runs through 10 different random states. XGBoost used 50 early stopping rounds in order to ensure the algorithm does not run for too long. After each algorithm is run, I averaged each of the 10 scores to get the test score, helping to eliminate some of the randomness of the Random Forest method. All other methods used were deterministic, but the random states help to diminish the uncertainty in the average test score.

4. Results

Test Scores

Algorithm	Average Test Score	Standard Deviation
Baseline	**817.674**	**37.403**
Lasso	440.442	41.574
Ridge	406.914	46.255
Elastic Net	516.248	40.263
Random Forest	390.201	18.565
XGBoost	344.175	15.414

Figure 9: Average test root mean-squared error and standard deviations for each algorithm

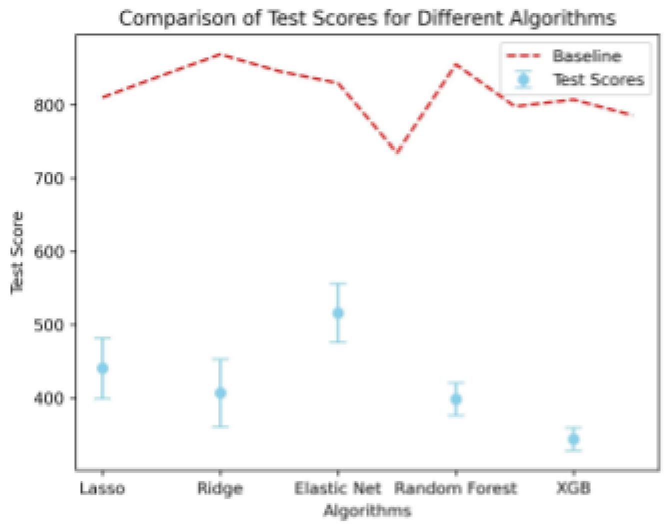

Figure 10: Mean test score visualization for each algorithm in comparison to the baseline scores

Model Performance

Looking at each algorithm's performance on the test set, as presented in Figure 10, XGBoost performed the best, as it has the lowest average root mean squared error with an RMSE of 344.175. Due to the large number of missing values present in the data, XGBoost performed the best, as it had the most effective mechanism for handling missing values. Even with XGBoost performing best, all of the algorithms performed much better (by scoring a significantly lower

RMSE) than the baseline RMSE, which predicted the mean as the true price, resulting in a root mean squared error of 817.674.

Global Feature Importance

With XGBoost performing the best, I looked at all five of its metrics for global feature importance, as well as SHAP values.

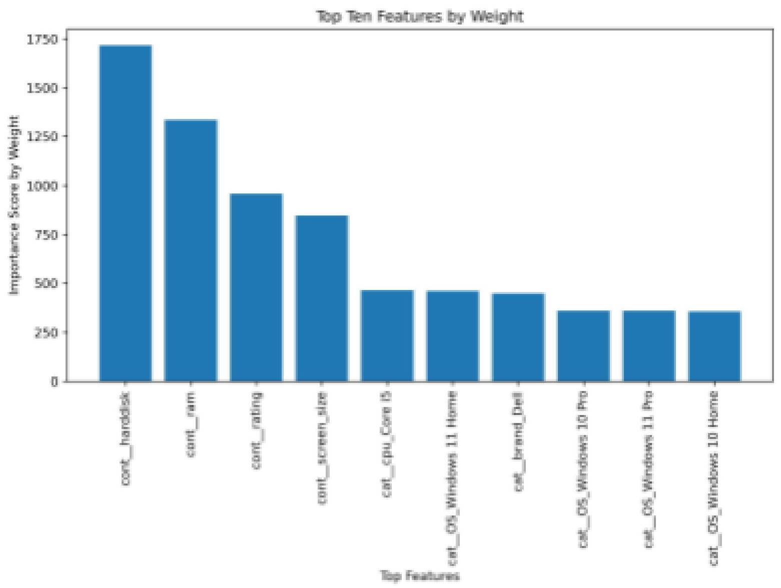

Figure 11: XGBoost global feature importance by weight

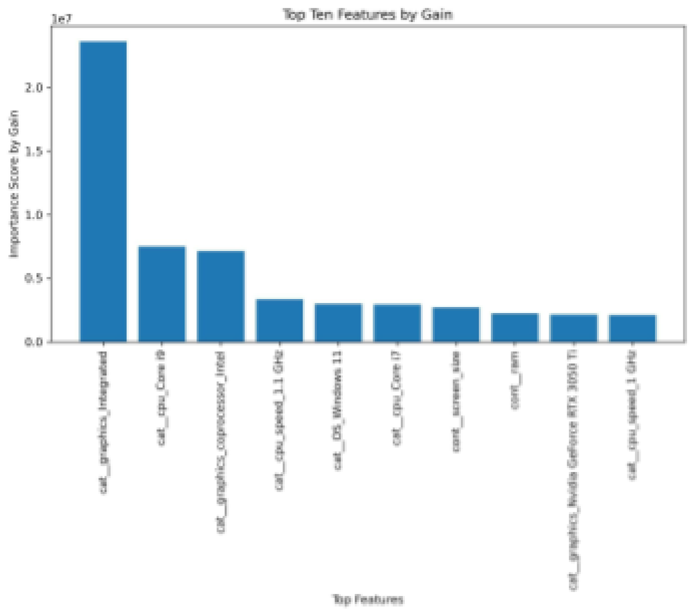

Figure 12: XGboost global feature importance by gain

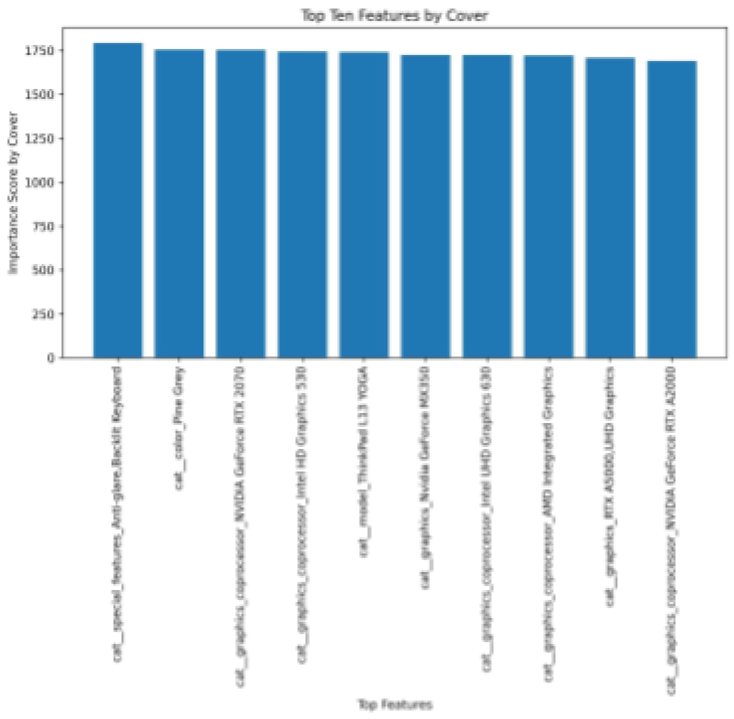

Figure 13: XGboost global feature importance by cover

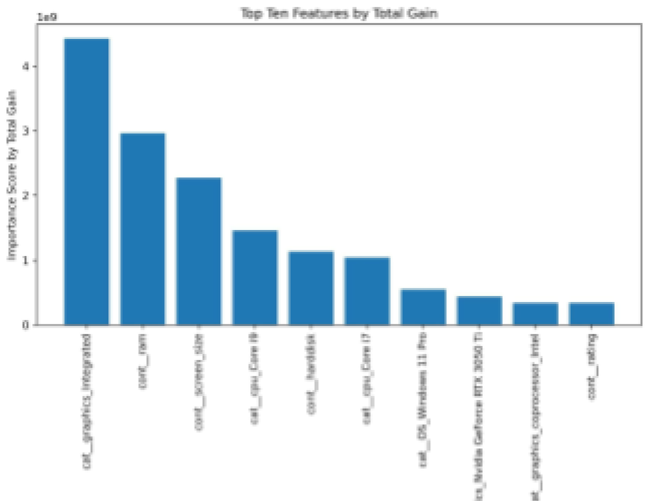

Figure 14: XGboost global feature importance by total gain

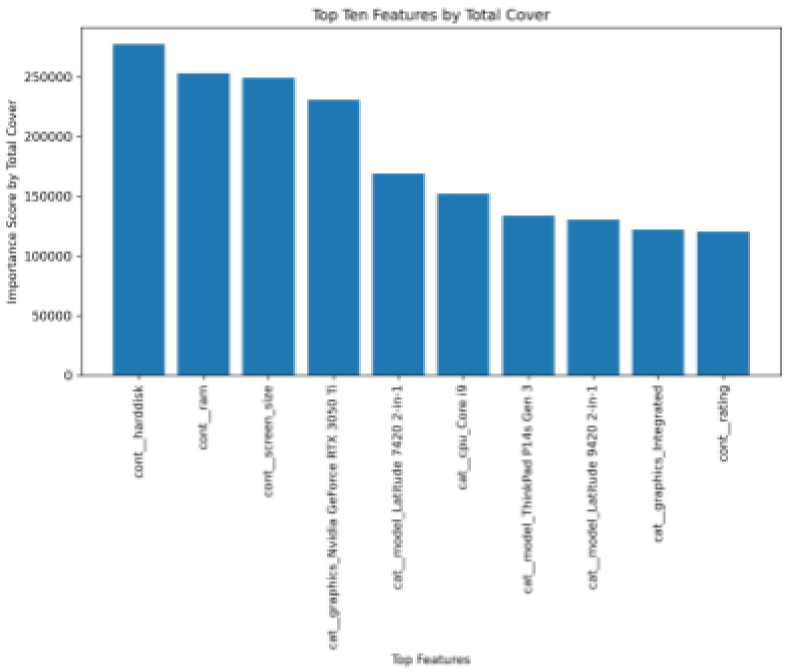

Figure 15: XGboost global feature importance by total cover

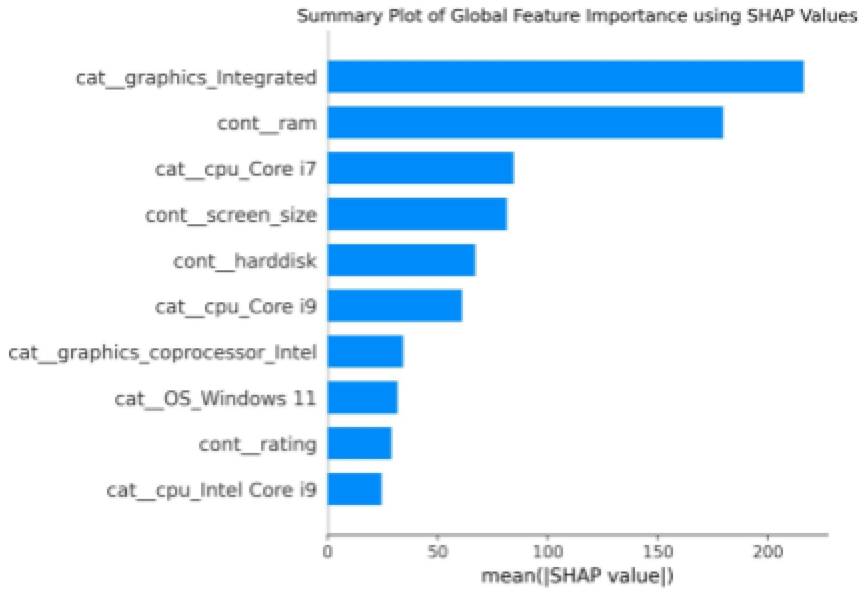

Figure 16: Global feature importance using mean absolute SHAP value

<u>Local Feature Importance</u>

Using indices 0 and 100, I calculated local feature importance, as indicated in Figures 17 and 18.

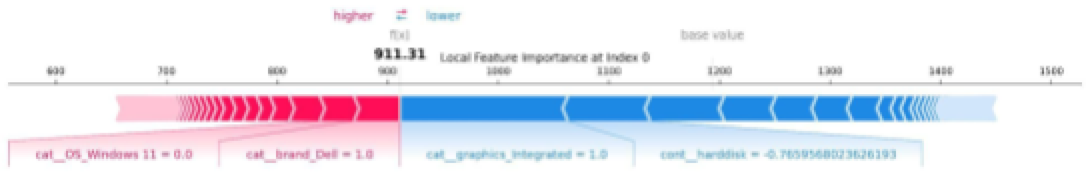

Figure 17: Local feature importance at index 0, using SHAP value

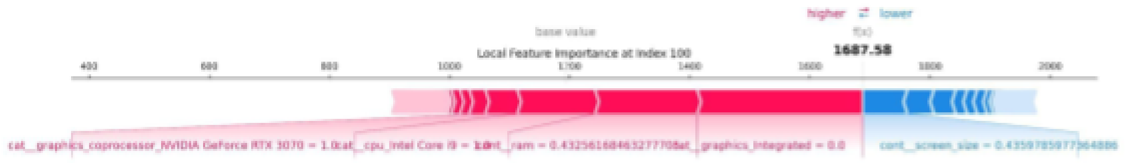

Figure 18: Local feature importance at index 100, using SHAP value

<u>Feature Importances</u>

Globally, harddisk, ram, and screen size appear as important features several times, likely because harddisk and ram greatly impact a computer's performance, and laptops often are available in different sizes, where larger versions are more expensive. Locally, integrated graphics are the greatest cause for the prediction to be below the base value ($1200) at index 0, but above it at index 100. Cpu speed, OS, and special features do not appear often, making them the least important features in the model's predictive power. Although rating appears a few times, it was surprising that customer satisfaction did not have a greater impact on the price.

5. Outlook

<u>Analysis</u>

The XGBoost model performed significantly better than the baseline, but it still had an average root mean squared error over 300, meaning its predictive power can continue to improve. Further feature engineering could have helped since the dataset had over 1500 columns after preprocessing. In the future, combining different categories within features or combining features together can improve the algorithm and interpretability. Combining multiple models' predictions together is another technique that can increase the model's performance. Continuing to add more laptops to the dataset can give the pipeline more information to train each algorithm. Market data specific to the technology industry, such as when technology prices rise and fall would be an intriguing and helpful feature to address some of the weaknesses in the current approach.

Overall, with more time, further research on this dataset has the potential for improved predictive power and interpretability.

6. References

[1]"What is a machine learning pipeline?," *What is a Machine Learning Pipeline?* [Online]. Available: https://valohai.com/machine-learning-pipeline/.

[2]"Is there any benefit to using cross validation from the XGBoost library over Sklearn when tuning hyperparameters?," *Data Science Stack Exchange*, 01-Feb-1969. [Online]. Available: https://datascience.stackexchange.com/questions/117367/is-there-any-benefit-to-using-cross-validation-from-the-xgboost-library-over-skl.

[3]A. Elshahat, "Amazon laptop prices analysis," *Kaggle*, 02-Dec-2023. [Online]. Available: https://www.kaggle.com/code/ahmedelshahat97/amazon-laptop-prices-analysis.

[4]T. B. Ahmad, "Laptop prices dataset - October 2023," *Kaggle*, 09-Oct-2023. [Online]. Available: https://www.kaggle.com/datasets/talhabarkaatahmad/laptop-prices-dataset-october-2023/code.

7. Github Repository

https://github.com/emlberman/project-emlberman.git