Universitat Politècnica de Catalunya

Facultad de Informática de Barcelona

PROJECT REPORT ON LAPTOP PRICE PREDICTION

MACHINE LEARNING

Spring 2024

Authors:
Arijit Samal
arijit.samal@estudiantat.upc.edu
Hieu Nguyen Minh
hieu.nguyen.minh@estudiantat.upc.edu

Supervisors:
Prof. MARTA ARIAS VICENTE

Contents

1 Overview **2**

2 Data exploration and Preprocessing **2**

3 Exploratory Data Analysis (EDA) **4**

4 Modeling Methodology **6**

5 Results and Discussion **8**

6 Conclusion **9**

1 Overview

We decided to tackle a regression problem focused on predicting laptop prices. We will use various laptop features to estimate prices for this task. We selected a dataset from Kaggle that provides relevant information for this purpose[1].

This dataset has 1303 instances with 12 features. The dataset dataframe is shown in Figure 1.

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	71378.6832
1	1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	47895.5232
2	2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg	30636.0000
3	3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg	135195.3360
4	4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg	96095.8080
...	...	...	...	...	...	...	...	...	...	...	...	...
1298	1298	Lenovo	2 in 1 Convertible	14.0	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i7 6500U 2.5GHz	4GB	128GB SSD	Intel HD Graphics 520	Windows 10	1.8kg	33992.6400
1299	1299	Lenovo	2 in 1 Convertible	13.3	IPS Panel Quad HD+ / Touchscreen 3200x1800	Intel Core i7 6500U 2.5GHz	16GB	512GB SSD	Intel HD Graphics 520	Windows 10	1.3kg	79866.7200
1300	1300	Lenovo	Notebook	14.0	1366x768	Intel Celeron Dual Core N3050 1.6GHz	2GB	64GB Flash Storage	Intel HD Graphics	Windows 10	1.5kg	12201.1200
1301	1301	HP	Notebook	15.6	1366x768	Intel Core i7 6500U 2.5GHz	6GB	1TB HDD	AMD Radeon R5 M330	Windows 10	2.19kg	40705.9200
1302	1302	Asus	Notebook	15.6	1366x768	Intel Celeron Dual Core N3050 1.6GHz	4GB	500GB HDD	Intel HD Graphics	Windows 10	2.2kg	19660.3200

Figure 1: Dataframe of the dataset.

2 Data exploration and Preprocessing

After loading the dataset, we checked for any duplicate instances or null values, ensuring data integrity. There were no duplicate or null values in the dataset. Upon inspection, we identified a redundant column labeled "Unnamed: 0," originally serving as the index column, which we promptly removed to streamline our dataset because it did not affect the laptop prices in any way other than storing the index of the rows.

Subsequently, we used a series of preprocessing steps to prepare the dataset for our primary objective: predicting laptop prices. These modifications, outlined below, aimed to enhance feature relevance and compatibility with regression models:

1. We extracted various text information embedded in the screen resolution column and made several features out of it. We Created four new features, "Touchscreen" (boolean), "IPS" (boolean), "PPI" (float), and "Retina" (boolean), from the feature "ScreenResolution", then dropped that column.

2. The "Touchscreen" column indicates whether the laptop features a touchscreen interface, while "IPS" denotes the presence of an IPS screen. Similarly, the "Retina" feature identifies laptops with retina displays, often associated with Apple MacBooks, which tend to sell at higher prices, as shown in EDA.

3. Similarly, we also extracted the resolution height and width from the column. Utilizing information on resolution height, width, and screen size (in inches) from the "Inches" column, we calculated the "PPI" (pixels per inch) feature using the formula:

$$\text{PPI} = \frac{\sqrt{\text{resolution\_width}^2 + \text{resolution\_height}^2}}{\text{inches}}$$

4. Create two new features, "Cpu_name" (categorical string) and "Cpu_clock_speed" (float), from the feature "Cpu", then drop that column. We also observed that there is only one laptop with a Samsung CPU, which does not help the overall analysis, so we removed it.

[1] https://www.kaggle.com/datasets/ayush12nagar/laptop-data-price-predition/data

5. Cpu_name column was obtained by exploring the different CPU types available in the dataset and analyzing their prices' distribution. This gave us a nice idea of which groups to create and keep in the cpu_name column. We finally created 8 categories of CPU and created the feature: Intel Core i7, Intel Core i5, Intel Core i3, Intel Core M, AMD Ryzen, Intel Xeon processors, other Intel processors, and other AMD processors.

6. Extracting the CPU clock speed from the "Cpu" column yielded the "Cpu_clock_speed" feature, which exhibited a positive correlation with laptop prices.

7. Remove the "GB" string in the column "Ram" column and set the value type to integer.

8. Create four new features, "HDD" (integer), "SSD" (integer), "Hybrid" (integer), and "Flash_storage" (integer) from the feature "Memory", then drop that column.

9. All these columns were created to represent memory capacities in GBs for each storage type present in a laptop.

10. Create a new feature "Gpu_brand" from the feature "Gpu", then drop that column.

11. This feature contained which brand of GPU the laptop had extracted from the GPU feature, which is made into categories of Intel, AMD, or Nvidia.

12. The "OpSys" column underwent categorization, retaining "Windows" and "Mac" OS while grouping other OS types (Linux, Android, Chrome OS, No OS) into a single category. Subsequently, we dropped the "OpSys" column.

13. Finally, in the "Weight" feature, we removed the "kg" string and converted values to float type.

After completing all the preprocessing steps, we obtain a new data frame with 1302 instances and 18 features, as shown in Figure 2.

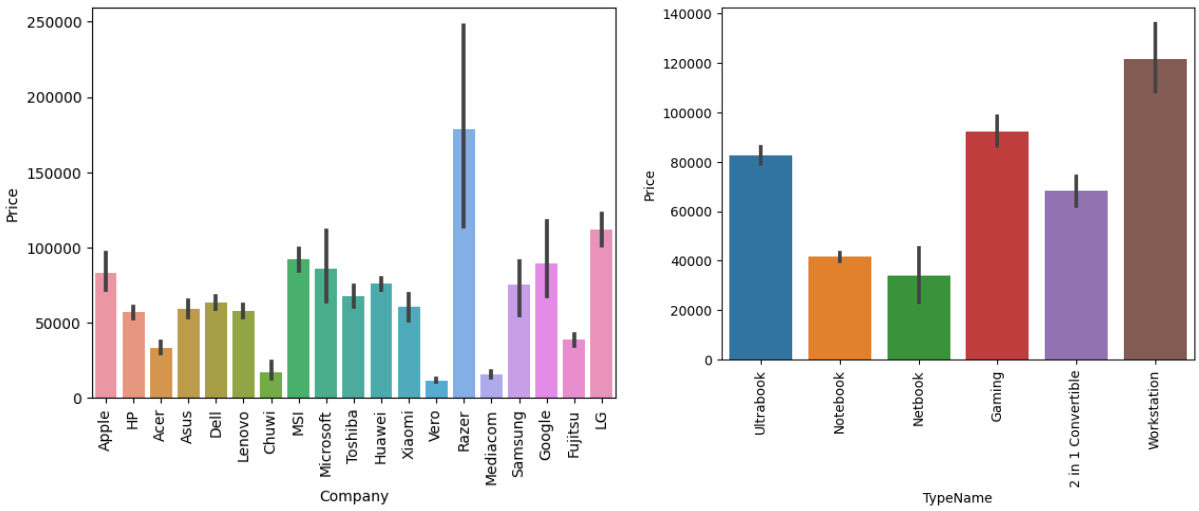
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1302 entries, 0 to 1302
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Company          1302 non-null   object
 1   TypeName         1302 non-null   object
 2   Inches           1302 non-null   float64
 3   Ram              1302 non-null   int32
 4   Weight           1302 non-null   float32
 5   Price            1302 non-null   float64
 6   Touchscreen      1302 non-null   int64
 7   IPS              1302 non-null   int64
 8   PPI              1302 non-null   int32
 9   Retina           1302 non-null   int64
 10  Cpu_name         1302 non-null   object
 11  Cpu_clock_speed  1302 non-null   float32
 12  HDD              1302 non-null   int64
 13  SSD              1302 non-null   int64
 14  Hybrid           1302 non-null   int64
 15  Flash_storage    1302 non-null   int64
 16  Gpu_brand        1302 non-null   object
 17  OS               1302 non-null   object
dtypes: float32(2), float64(2), int32(2), int64(7), object(5)
```

Figure 2: Dataframe info after preprocessing.

3 Exploratory Data Analysis (EDA)

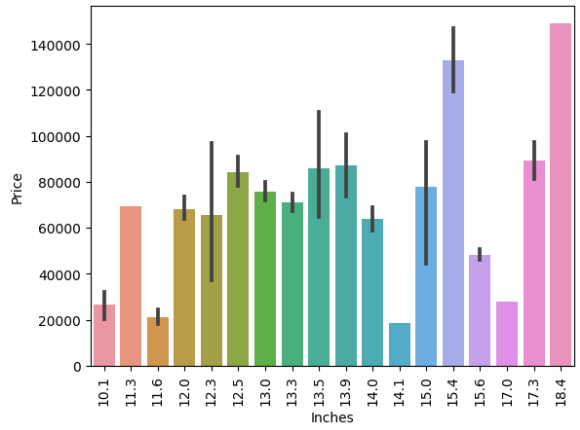
We performed exploratory data analysis in parallel to the data preprocessing to get insights from the new and old features. All the figures here provide specific insights into the dataset. All the figures and reasoning are mentioned below:

1. Laptop prices among companies do not vary much, except "Chuwi", "Vero" and "Mediacom" where the price is very low, and "Razer" where the price is very high.
2. High-end laptop types (Ultrabook, Gaming, 2in1 Convertible, Workstation) have higher prices than basic ones (Notebook, Netbook).
3. Laptops with larger screens tend to have higher prices than ones with smaller screens.
4. Laptops with higher RAM capacity have higher prices than ones with smaller RAM capacity.
5. Heavier laptops tend to have higher prices.
6. Apart from some special cases, high-resolution laptops (higher PPI) tend to have higher prices than low-resolution ones.
7. Intel-CPU laptops tend to have higher prices than AMD-and-other-CPU laptops.
8. Laptops with faster CPUs (with more clock speed) have higher prices than those with slower CPUs.
9. Laptops with higher SSD/HDD/Hybrid/Flash_storage capacity tend to have higher prices than ones with lower disk capacity.
10. NVIDIA-GPU laptops have higher prices than Intel and AMD ones.
11. MacOS laptops have higher prices than ones with Windows and other operating systems.
12. Laptops with touch screens, IPS screens, and Retina screens have higher prices than ones without these screen types.

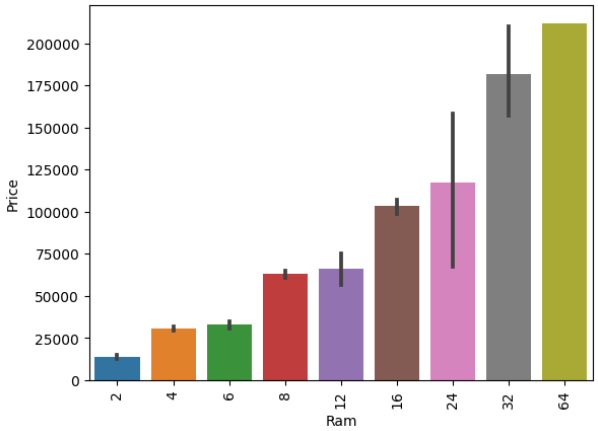


(a) Company-Price. (b) TypeName-Price.

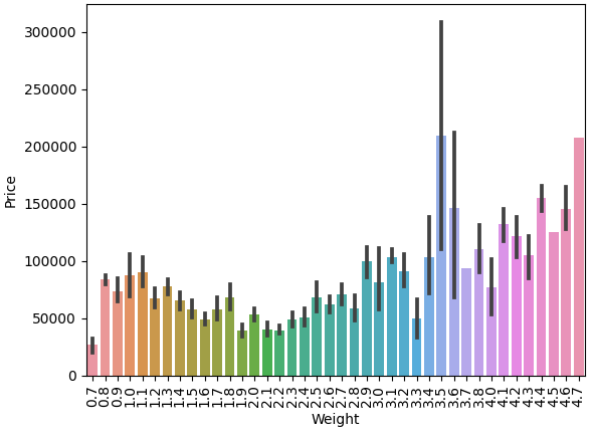
Figure 3: Feature-Price relations.



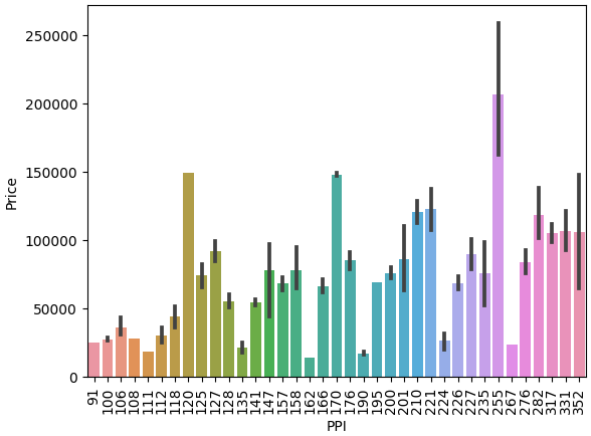
(a) Inches-Price.



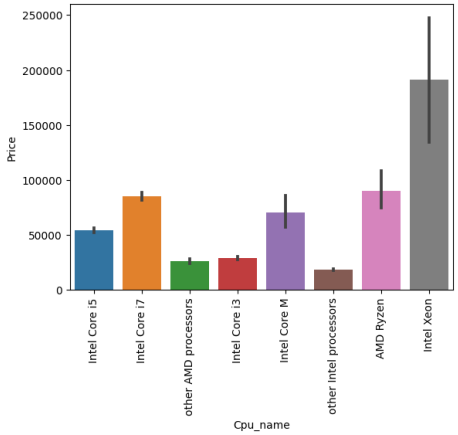
(b) Ram-Price.



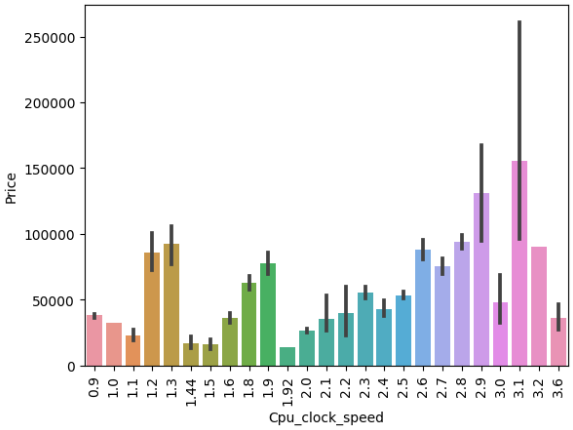
(c) Weight-Price.



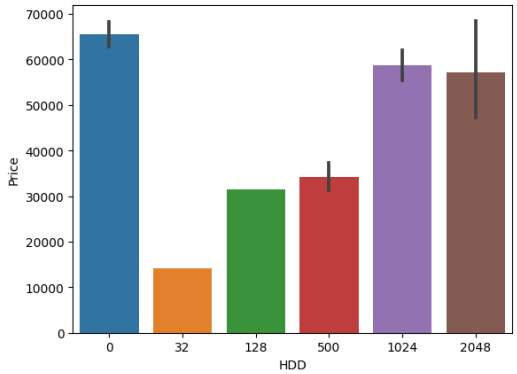
(d) PPI-Price.



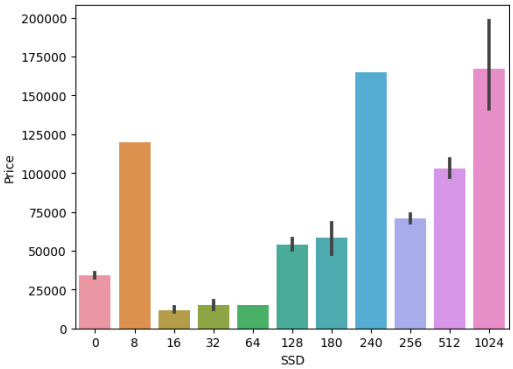
(e) Cpu_name-Price.



(f) Cpu_clock_speed-Price.

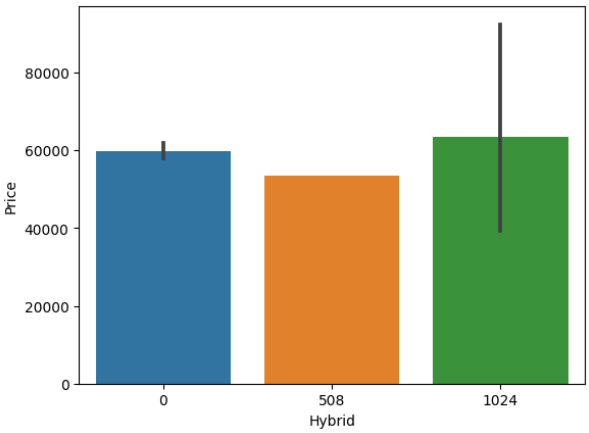


(g) HDD-Price.

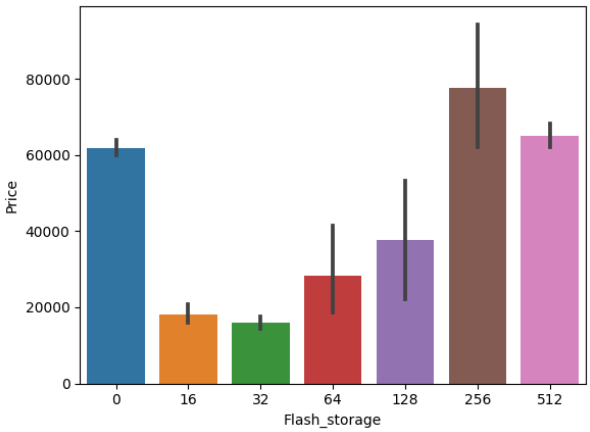


(h) SSD-Price.

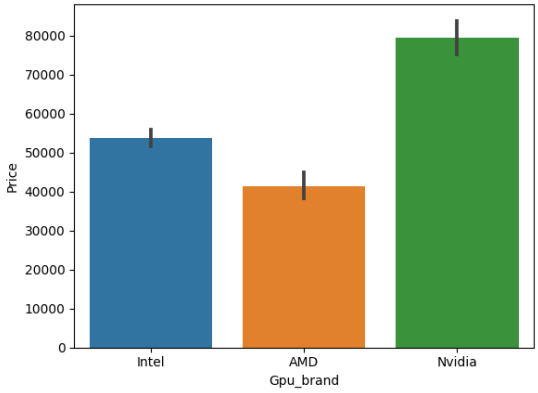
Figure 4: Feature-Price relations.



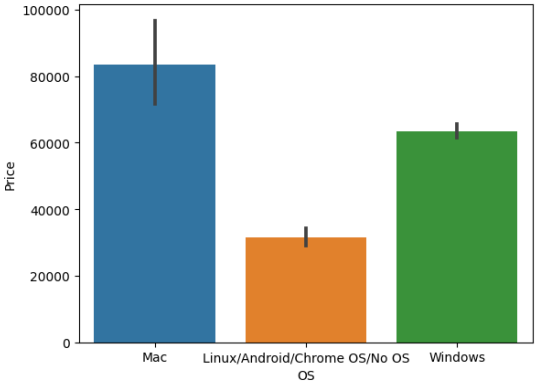
(a) Hybrid-Price



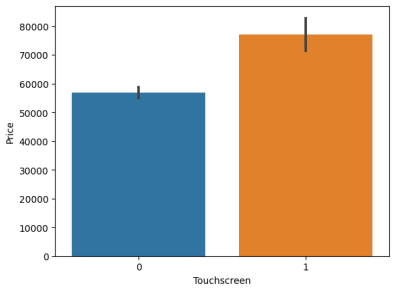
(b) Flash_storage-Price.



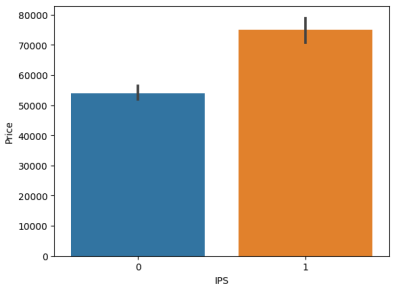
(c) Gpu_brand-Price.



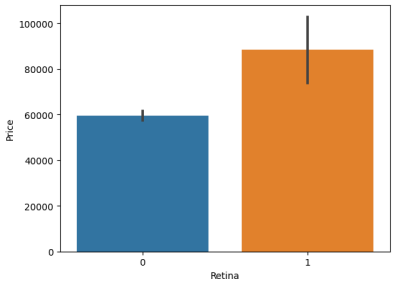
(d) OS-Price.



(e) Touchscreen-Price.



(f) IPS-Price.



(g) Retina-Price.

Figure 5: Feature-Price relations

4 Modeling Methodology

To gain insights into the relationship between our newly engineered features and the target variable – the price of laptops – we conducted a correlation analysis, visualized through a correlation matrix (refer to Figure 6). The matrix revealed that most features exhibit some level of correlation with the laptop's price, indicating their potential contribution to price determination.

Subsequently, leveraging machine learning regression models, we estimated laptop prices based on our enhanced dataset. Initially, we structured input data and corresponding labels from the data frame. We adopted one-hot encoding for categorical features such as Company, TypeName, Cpu_name, Gpu_brand, O, HDD, SSD, Hybrid, and Flash_storage using the pandas dummies function. This transformation converted categorical attributes into numerical counterparts (Bi-

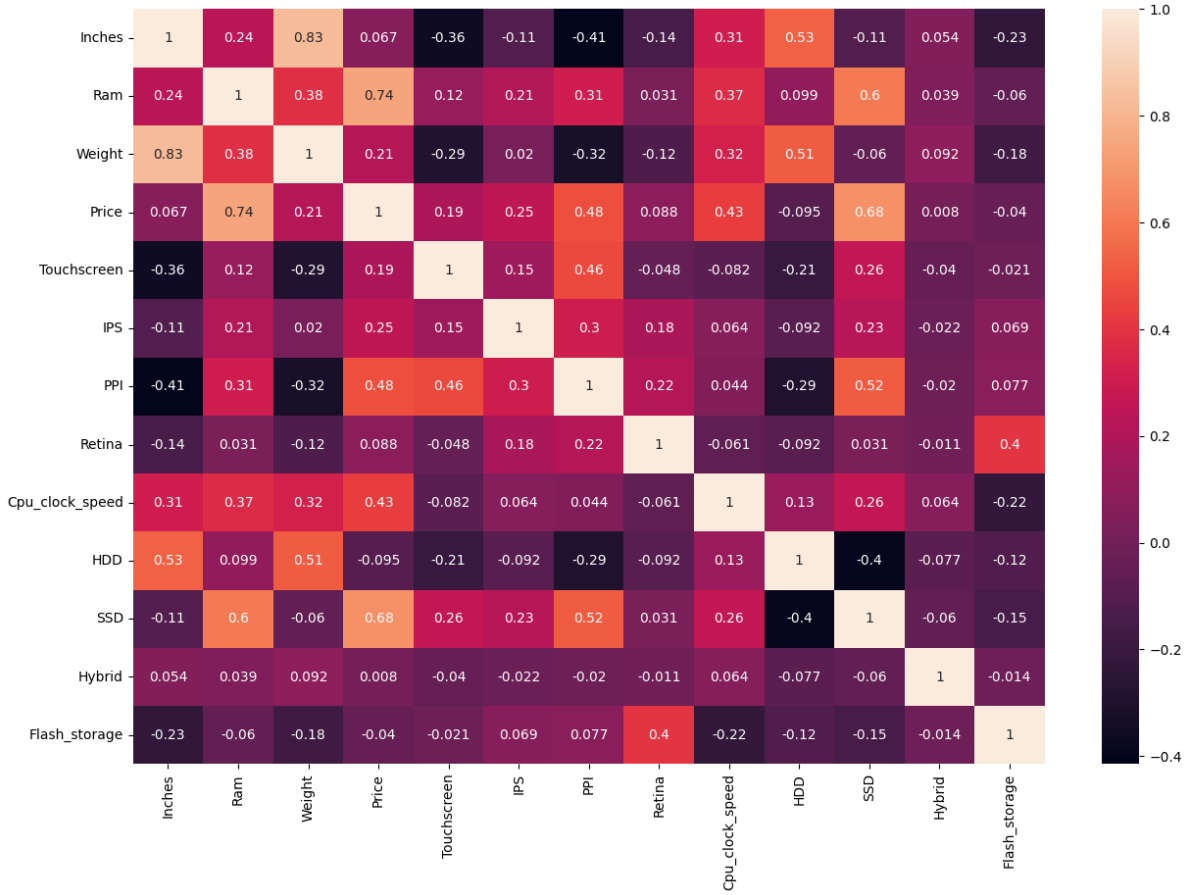


Figure 6: Correlation Matrix

nary feature), a prerequisite for training certain machine learning models and enhancing overall model performance.

We applied a logarithmic transformation to the price values to stabilize variance and promote a more normal distribution, which is essential for improved model performance. However, post-prediction, we reverted this transformation to restore predictions to their original scale, ensuring interpretability and practicality.

This preprocessing yielded a training matrix (X), with a shape of $(1302, 82)$, along with a label column matrix, which is the target (y) comprising 1302 entries. Following this, we partitioned the data into an 80% training set and a 20% test set to facilitate model training and evaluation.

For our regression pipeline, we leveraged the scikit-learn library to construct a robust framework to estimate laptop prices. Our pipeline incorporated ten distinct regression models, each offering unique methodologies for price estimation:

1. Support Vector Regression (SVR)
2. Linear Regression
3. Ridge Regression
4. Lasso Regression
5. ElasticNet Regrression

6. K Neighbors Regressor
7. Decision Tree Regressor
8. Gradient Boosting Regressor
9. Random Forest Regressor
10. XGB Regressor

We introduced two pivotal parameters within our pipeline: **exhaustive and tuning**. When tuning remained deactivated (set to false), we executed the models in their raw form, utilizing default hyperparameters. This initial run provided baseline results to evaluate the effectiveness of subsequent tuning efforts in improving the predictions.

Activating tuning initiated a phase of Hyperparameter Tuning across all models, which could follow either exhaustive or non-exhaustive search methodologies. Enabling the exhaustive parameter triggered the GridSearchCV algorithm, where a predefined Hyperparameter Grid was meticulously explored. This exhaustive search involved training and evaluating the model across every conceivable combination of hyperparameters.

Conversely, when exhaustive search wasn't warranted, we employed the RandomizedSearchCV algorithm for hyperparameter tuning. This method, serving as an alternative to exhaustive search, sampled a fixed number of hyperparameter combinations from predefined ranges, reducing the time needed to fit the models and finding the best model.

During the training phase of both hyperparameter tuning methods, we adopted 5-fold cross-validation. Here, the training dataset was partitioned into five equally sized folds. For every hyperparameter combination, the model underwent training on four folds while being validated on the remaining fold. This process is iterated five times, with each fold serving once as the validation set. The hyperparameter combination yielding the best average performance across these folds was deemed optimal.

To evaluate model performance, we computed several metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), R^2 score, and Best Score – the best average score achieved by the optimal hyperparameter over all cross-validation folds. Our model selection criterion prioritized the model with the least MAE and MSE, coupled with the highest R^2 score and best score. Notably, we adopted the R^2 score as the primary scoring method for hyperparameter tuning, applied across both grid search and randomized search with 5-fold cross-validation.

5 Results and Discussion

The results of all models and methods are presented in Table **??**. From the table, we have some observations:

1. Without the hyperparameter tuning, the Support vector regression, Lasso regression, and ElasticNet regression models do not perform well. This is because these models have important parameters like kernel, regularization parameter, or mixing parameter, which define the ratio of L1 and L2 regularization in Elasticnet and must be appropriately set. Other models' performances are similar, including the XGBRegressor, which outperforms the others.

2. With RandomSearch hyperparameter tuning, the issues of SVR, Lasso, and ElasticNet regression have been resolved as those models achieve good performance. Other models'

performances do not vary much, but there is a decrease in the performance of DecisionTreeRegressor. This can be explained by the fact that this hyperparameter tuning method cannot find the optimal solution in the parameter space.

3. With GridSearch parameter tuning, all models obtain better performance than their previous methods. The best one is still XGBRegressor. the XGBRegressor performance increases in grid search as it finds the best hyperparameter when we exhaustively search through the entire space of hyperparameters.

MODELS	No Hyperparameter Tuning				With RandomSearchCV				With GridSearchCV			
	R^2	MSE	MAE	Best Score	R^2	MSE	MAE	Best Score	R^2	MSE	MAE	Best Score
SVR	0.3945	0.2333	0.3769	0.3661	0.8349	0.0636	0.1992	0.8458	0.8349	0.0637	0.1984	0.8462
LinearRegression	0.8332	0.0643	0.1987	0.8456	0.8333	0.0642	0.1987	0.8456	0.8332	0.0643	0.1986	0.8457
Ridge	0.8363	0.0631	0.1978	0.2161	0.8364	0.0631	0.1977	0.8502	0.8363	0.0631	0.1976	0.8478
Lasso	0.263	0.2839	0.4164	0.8501	0.8369	0.0628	0.1977	0.8478	0.8369	0.0629	0.1976	0.8502
ElasticNet	0.2645	0.2834	0.4165	0.2168	0.8379	0.0625	0.1979	0.8478	0.8379	0.0625	0.1979	0.8478
KNeighborsRegressor	0.8189	0.0697	0.2045	0.8075	0.8167	0.0706	0.2152	0.7934	0.8267	0.0668	0.1989	0.8086
DecisionTreeRegressor	0.7735	0.0873	0.1931	0.7735	0.7486	0.0969	0.2471	0.7839	0.6722	0.3554	0.2821	0.7356
GradientBoostingRegressor	0.8734	0.0488	0.1739	0.8715	0.8818	0.0466	**0.1455**	0.8927	0.8899	0.0424	0.1535	0.895
RandomForestRegressor	0.8707	0.0497	0.1606	0.8754	0.8583	0.0545	0.185	0.8652	0.8531	0.0566	0.1796	0.8666
XGBRegressor	**0.889**	**0.0428**	**0.1591**	**0.8848**	**0.8862**	**0.0435**	0.1583	**0.8881**	**0.9014**	**0.038**	**0.1439**	**0.8992**

Table 1: Comparison of different regression models based on MSE, R^2 score, MAE on the test set and best scores over all the cross-validation folds

Using the metrics outlined in Table 1, it is evident that XGB regression consistently performs better than all other regression algorithms across various metrics. Given its superior performance, we opt to leverage the best hyperparameters obtained during the grid search phase. The hyperparameters that exhibited the best scores across all metrics are employed to fit the XGB regression model to the test data and the whole dataset, facilitating the estimation of laptop prices.

To ensure compatibility with the original scale, we reverse the logarithmic transformations previously applied to the target column during model training using exponentiation on the estimated prices, thereby restoring the predicted prices. Subsequently, the predictions are exported to a CSV file for further analysis.

Additionally, recognizing the value of preserving the trained model for future use, we save the best-performing model in both pickle and joblib file formats. This enables convenient and easy future inference by simply loading the pre-trained model.

6 Conclusion

In this project, we successfully estimated laptop prices using a variety of machine learning regression models, achieving a notable R^2 score of 0.9014 on the test set. Our methodology encompassed thorough data exploration and preprocessing to construct a robust training dataset. We experimented with ten distinct regression models, applying hyperparameter tuning techniques to enhance performance. Among these, GridSearchCV was particularly effective in identifying the optimal hyperparameters. Remarkably, the XGBRegressor model consistently outperformed all other models across various experimental settings.