Automobile Price Prediction

Authors: Pol Monroig
Josep Maria Olivé

Subject: Aprenentatge Automàtic

Delivery date: January 11th, 2021

Table of contents

1 Introduction

The main objective of this project is building different machine learning approaches to perform a regression task and predict the price of a car according to a set of characteristics available, such as its size, engine type, brand, miles per gallon, etc. The dataset used is available at UCI's Machine Learning Repository [9] and consists of 26 attributes (10 categorical and 16 numerical) and 205 instances. This is an extremely small dataset and it is very unrealistic because in a real environment you could easily find out more than 205 different car types, thus we are only using this dataset for research and educational purposes.

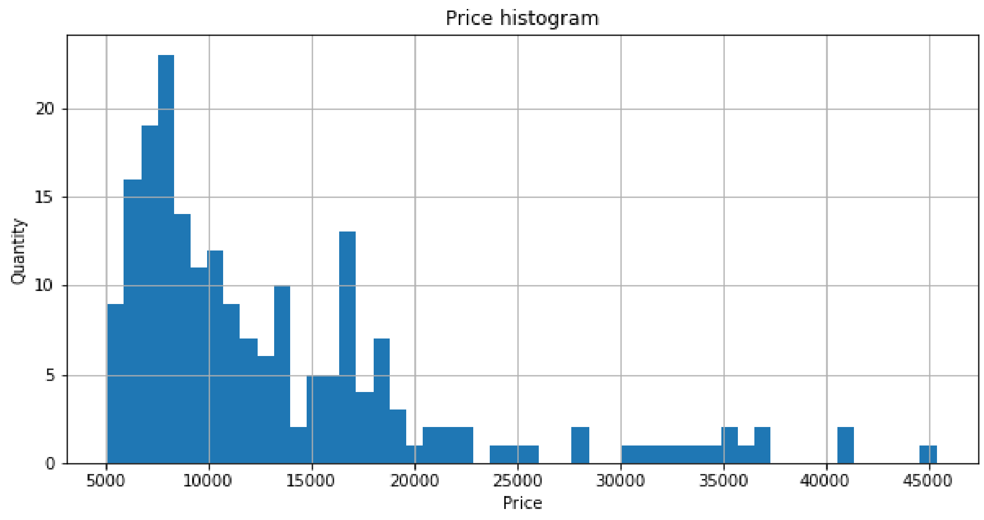
We wanted to find out a model that is as simple as possible and that gives the best results, that is why we compared the performance and generalization of 5 different types of machine learning models. Three were grouped as linear/quadratic methods: linear regression, k-nearest-neighbors, and linear SVM. And two were non-linear methods: MLP and Random Forest. For each model, we made sure to find out the best hyperparameters. In the end, MLP turned out to be the best method with an R2 score near 0.75.

2 Previous Work

Previously to all the work done, we had to select a dataset to perform our predictions. We explored many different Machine Learning repositories such as UCI, University of Edinburgh [9], Kaggle, and others, looking for a not so popular but still good dataset. Finally, we came across the automobile dataset, which meets our requirements. This dataset is very small but it is a well-known dataset in the machine learning community, as it is used in educational environments so their multiple sources of previous work, it isn't a specially challenging dataset.

3 Exploratory Data Analysis

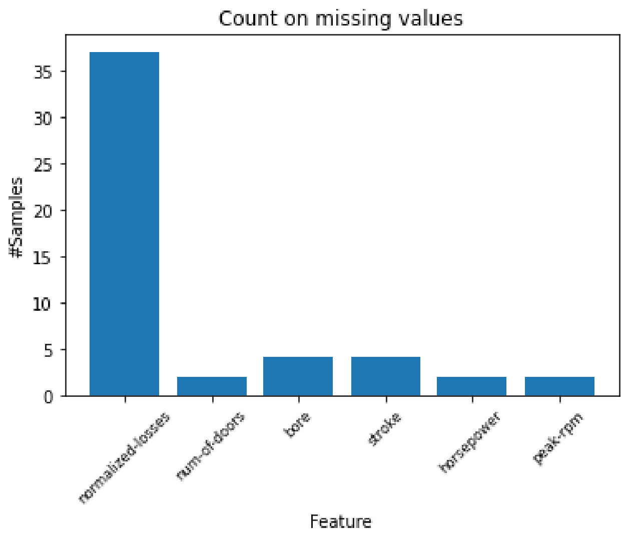
First of all, we took a glance at the distribution of our target variable, the price of each car. At first look, the distribution of the price seems to be a lot like Chi-Squared. Most values are below 20000.



3.1 Preprocessing

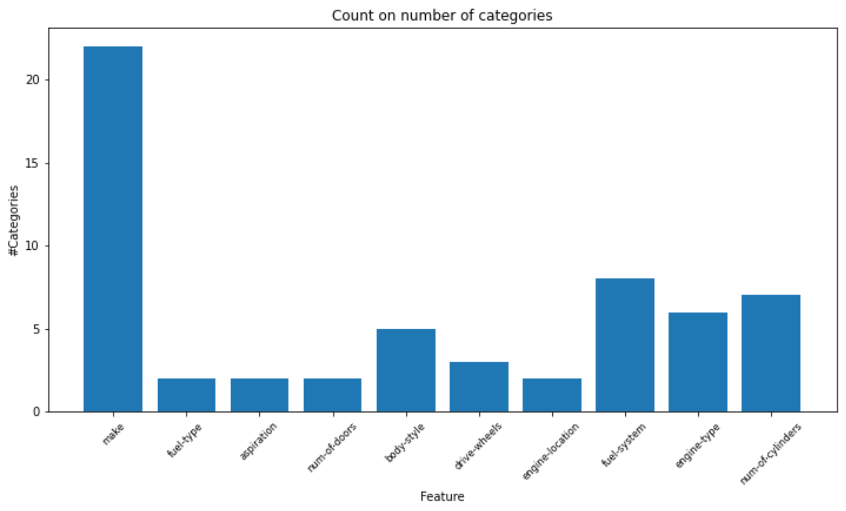
3.1.1 Missing Values

When preprocessing our dataset, we had to fill in all the missing values. We used SimpleImputer [3] with a most frequent strategy, which fills missing values with the mean in numerical attributes and with the most frequent value in categoricals. The strategy we chose wasn't really transcendental on the model itself. We can justify this hypothesis by the fact that there were very few missing values, with the exception of the normalized-losses, a variable that is the relative average loss payment per insured vehicle year. Another approach we could have done was to remove the entries with these missing values but as we already said, imputing very few values were missing.



3.1.2 OrdinalEncoding

For categorical features, it is mandatory to use a conversion method to make them numerical. We added OrdinalEncoder from SKLearn [3] preprocessing tools to automate this requirement. We used an OrdinalEncoder because each categorical feature has an exclusive category, so we needed to encode each feature with a different number, contrary to what a OneHotEncoder gives us. We also preferred this type of encoding because it returns us a dense matrix.
In the following plot, we can see the number of categories per feature, most categories have a similar count (near 5 categories) with the exception of the *make* feature, which has more than 20 unique categories.



3.1.3 Normalization

Once all features are represented with numbers, we have to normalize them between numbers from 0 to 1 for a better fitting in later steps. For this matter, we used MinMaxScaling from SKLearn preprocessing, which once again, automates this task for us.

3.1.4 Feature extraction

In the end, we didn't add any of the extracted features we thought about. Some of the possibilities we tried were the following ones: horsepower/mpg, car volume, avg mpg, price/horsepower, horsepower/engine size, horsepower/number of cylinders, etc.
These were all good candidates that we thought would apport new information to our research, but unfortunately, we were wrong. We carried many different analyses on the weight of these new features. First, the correlation matrix clearly showed that the new data was completely correlated with the original features we used to extract these new attributes, thus, no new information was added to the dataset. In the second place, and we will get deeper on this matter lately, we used PCA [1] to reduce redundant information. When running PCA analysis on our new dataset that included those extracted features, all of them were instantly removed as their explained variance ratio was near zero.

3.2 Dimensionality Reduction

Another step in the preparation pipeline includes a dimensionality reduction. Each feature in the dataset represents a specific characteristic of a car, each component in a car is related(e.g., engine size, car size,wheel-size, etc.), so we can only assume that most features will be related. Based on this premise we developed a study on the relationship between each pair of variables. In the following figures, we only reflect numerical features because it wouldn't make sense with categorical ones.

3.2.1 Dimensionality Reduction Applied

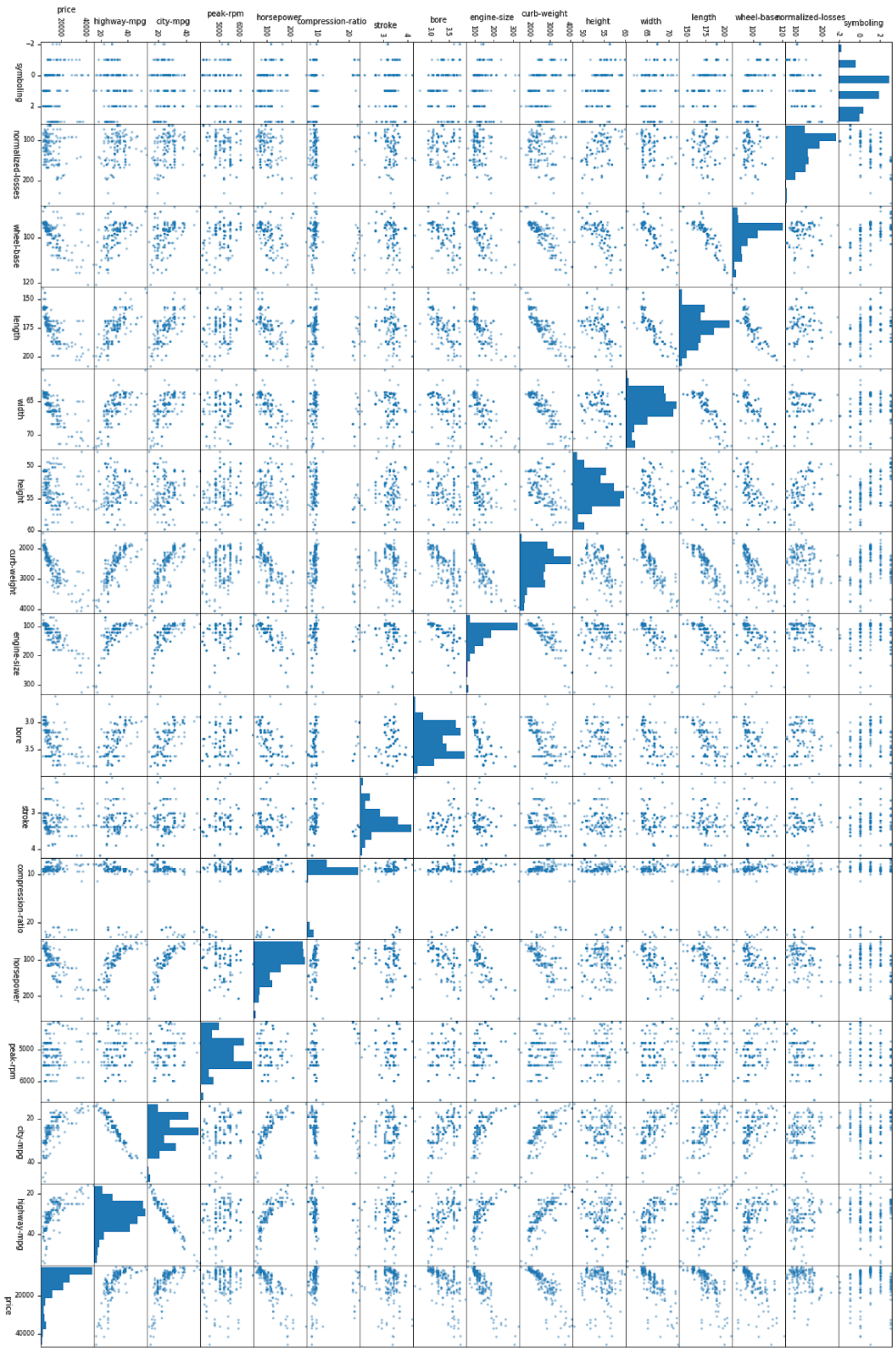
As we will later see, a lot of variables tend to have similarities in terms of relationships and meaning, thus we found out how to minimize some variables. As a first approach we tried reducing variables manually, but this reduction was a little arbitrary since we couldn't decide which variables were more important, based on this "importance" description, we found out a method called Forward/Backward feature selection [2], which selected features based on their effect on the evaluation error, this yielded to great results and dimensionality reduction (< 5 variables) but it has a stochastic nature and we didn't like that a lot since we had a small dataset and it could yield to unexpected results on a real environment. Ultimately we applied a Principal Component Analysis with a number of components of 0.8 (keep 0.8 of the explained variance) as it provided us with the best results and less overhead.

3.2.2 Pairwise Study

In this study, we created plots that reflect the relationship between the values of each feature [1]. We generated a matrix of plots where the diagonal represents the distribution of the feature in a histogram and other elements represent the Feature vs Feature relationship. We can see that most features tend to have a normal distribution shape which assures that even though we have few samples, the data is extremely representative.

We can also see that features such as *Symboling* and the *Compression-Ratio* do not have any type of linear relationship with any variable, although on the *Compression-Ratio* this might be caused by the shape of its unbalanced distribution. We can also see that both *mpg* (miles per gallon) variables have a linear relationship and that makes sense, since the more miles you do, the more gallons you use, independently on whether you are on the highway or the city; so we can clearly eliminate one of the two. Subsequently, we can also see that all variables related to the size of the car tend to have a positive relationship between them, this relationship is not exactly linear so we might need to test which variable gives us more information.

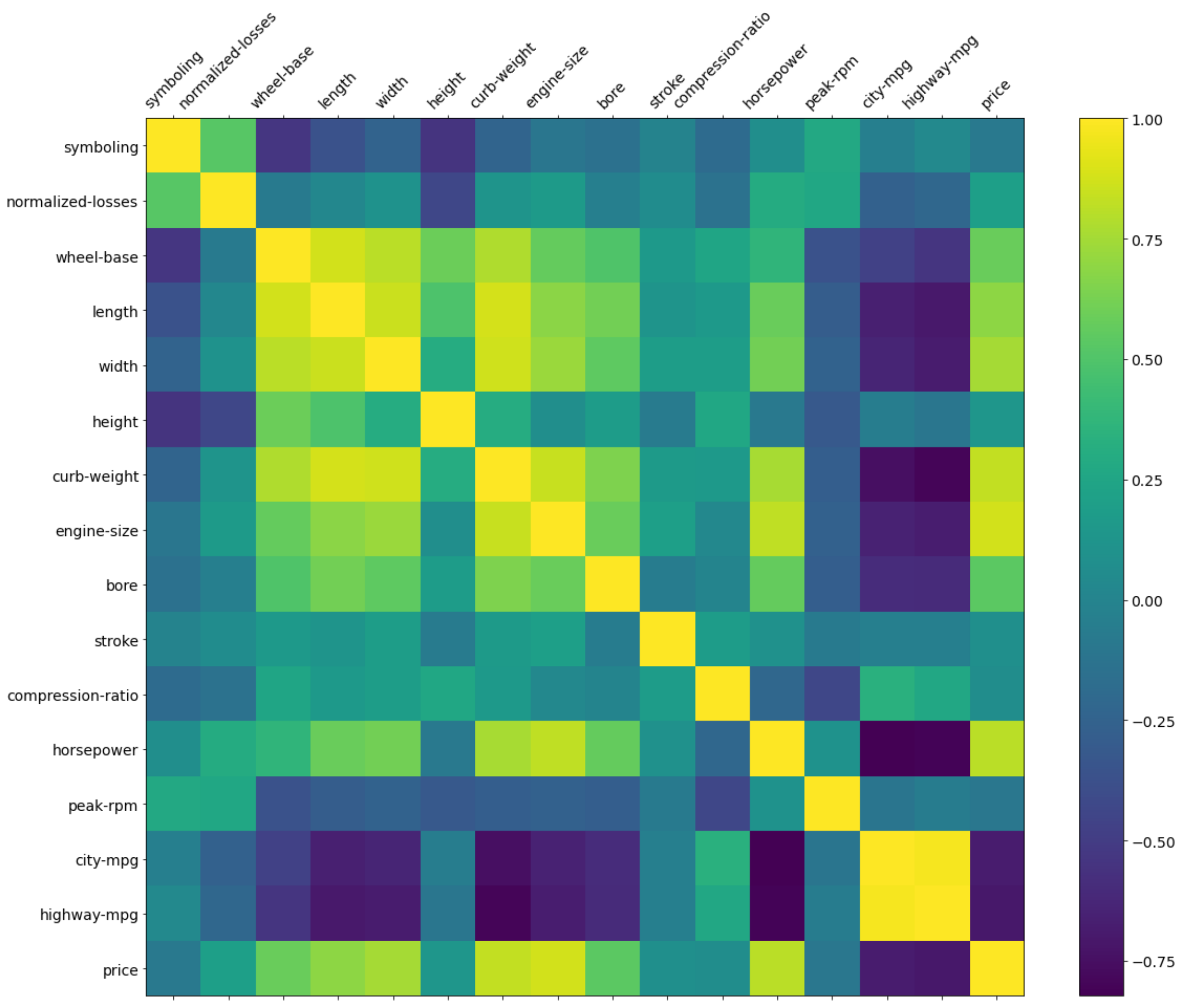
Finally, we can also see that the prediction feature (*price*) seems to have a relationship between most variables, this is a good sign since this relationship shows us that it is easier to model. This relationship is useful for the model but it also causes a problem, since all variables seem helpful, it might be more difficult to decide which variables are important and which are not. We can only assume that similar variables must be discarded to prevent redundancy.



price
highway-mpg
city-mpg
peak-rpm
horsepower
compression-ratio
stroke
bore
engine-size
curb-weight
height
width
length
wheel-base
normalized-losses
symboling

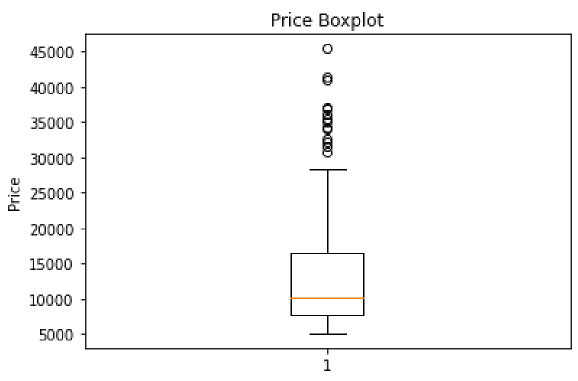
3.2.3 Correlation Study

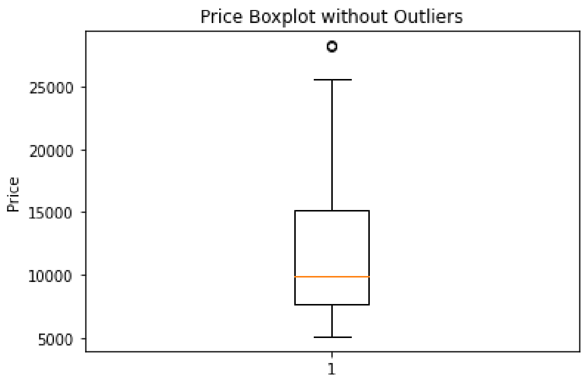
In this study, we are focused on finding the same: relationship between variables. But instead of looking at the relationships ourselves, we are going to rely on Pearson's Linear Correlation Coefficient [10], which gives how correlated two variables are. Making use of the two studies is necessary since even though we might find a correlation between two variables, this correlation might not be linear, in other words, the coefficient might be deceiving us. In the following correlation matrix, we can see that similar results. For instance, the *mpg* variables clearly have an incredibly high positive correlation. We can also see a relationship with variables related to size, but the correlation value is not too high. A thing that can surprise us is that variables like *length, width,* and *horse-power* have a very high negative correlation with the *mpg* variables, something that was not as clear in the latter study. This correlation is justified by the fact that the bigger the car is, the more gallons it consumes. Another justification for removing one of the two *mpg* variables is that their relationship between other variables is the same. Similar to what we saw before, the price variable has a linear relationship between the car characteristics, but this relationship is not that clearly reflected in the correlation matrix, in fact, the variables that seem to have the most correlation are the *mpgs*.



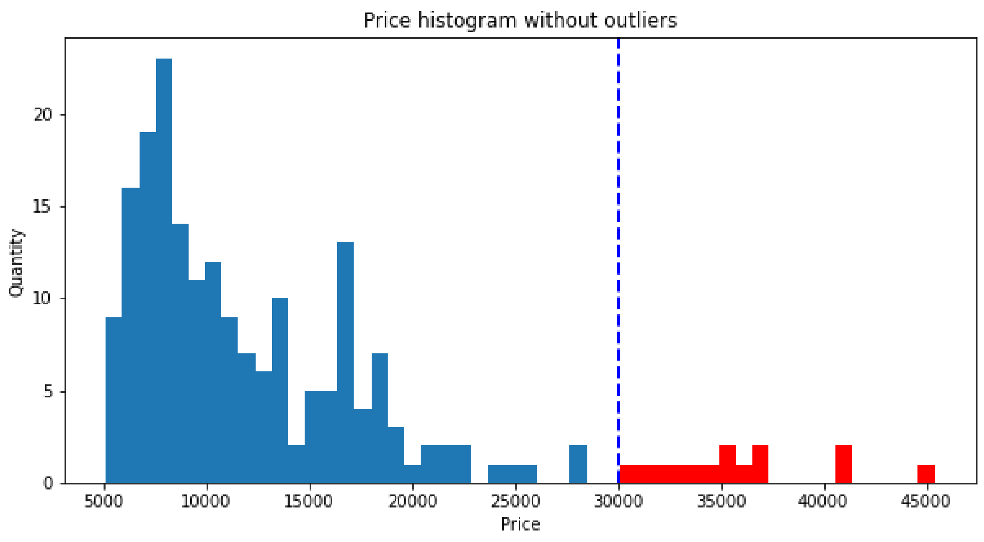
3.3 Outlier Detection

Having a small dataset is dangerous since a few outliers can have a profound effect on the model, thus, this type of study was of great importance to us. Based on the initial distribution we can observe that most price data is between 5000 and 20000 units (possibly US dollars), we hypothesis that we could remove cars that contained a price outside of this range, but before applying the results we developed a Boxplot [8] to get more insight. This boxplot showed us that our range was off by 10000 units and that outliers correspond to prices bigger than 30000. The following two boxplots show the difference between the data with outliers and without, choosing a 30000 threshold was really the way to go.





Removing outliers would lead us to remove all the prices marked in red in the histogram. After thorough thinking, we decided to keep outliers, since they are representative of the dataset, and removing them made the size of the dataset much smaller and did not provide any type of improvement. We thought that removing outliers would be unrealistic since in a real-world scenario we would encounter all types of prices and removing them would prevent the model from learning such important information. Finally, removing these outliers would have caused us some problems while testing the cross-validation, that is because these outliers are real, they are not oddities and they should have been included in the test set, and excluded from the train set.



4 Training & Results

4.1 Resampling

For the resampling, we didn't have a lot of data and we used k-Fold cross-validation to reduce the bias as much as possible. We used 5 folds, which ensured we split the data with at least 20% of the test size. To train our dataset we first applied 5-Fold cross-validation to find out the best parameters, to find them, we used a GridSearch to search all the combinations. Once we had the best parameters, we then trained the models again using k-Fold cross-validation to get the final results.

4.2 Hyperparameters Tuning

In the following sections, we will explore the different parameters used and modified for each different machine learning method. When GridSearch is used, we manually ran a binary search to find the best combination of hyperparameters given all the possible values.

4.2.1 Linear Regression

We didn't use any custom parameter for Linear Regression but used the technique AS-IS.

4.2.2 K-Nearest-Neighbours

For K-Nearest-Neighbours we only used one hyperparameter, the number of neighbors, k. The optimum value for the number of neighbors turned out to be 34.

4.2.3 Linear SVM

For this technique, we first set the Dual parameter to False, as it is recommended when using a dataset with a bigger number of samples than features, next we used GridSearch to tune the hyperparameters C and Epsilon. The best value for C in our case turned out to be 800, and 100 for epsilon.

4.2.4 MLP

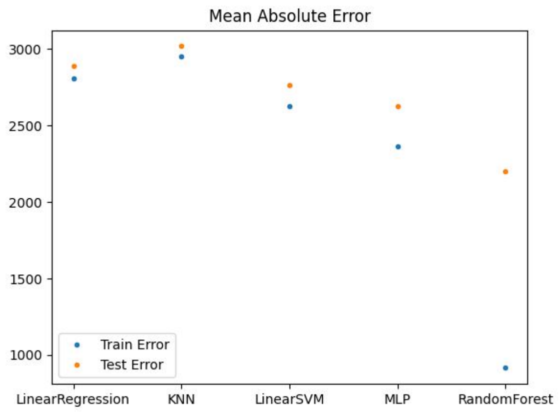
For Multi-Layer-Perceptron, we ran GridSearch with different possible values over the following hyperparameters: the size of the hidden layer, the optimizer, alpha, initial value for learning rate, the number of iterations, and using or not early stopping.
We chose adam over sgd as an optimizer, because sgd was throwing infinite values when training and crashing our scripts. For the rest of the hyperparameters, we got the best results with 0.001 for an alpha, [5000] for the size of the hidden layer, 0.1 for the initial value for the learning rate, and 100 as the maximum number of iterations.
We didn't modify the learning rate itself apart from its initial value, because the adam optimizer already configures this hyperparameter for each weight individually.

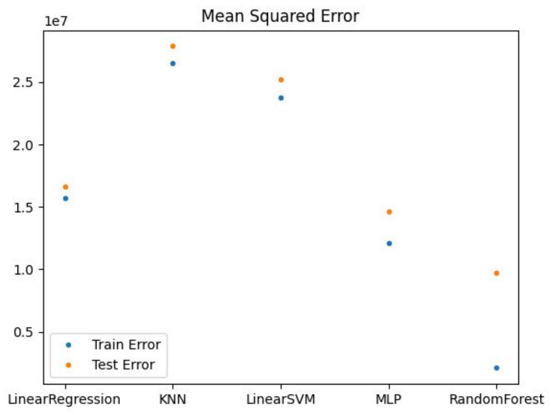
4.2.5 Random Forest

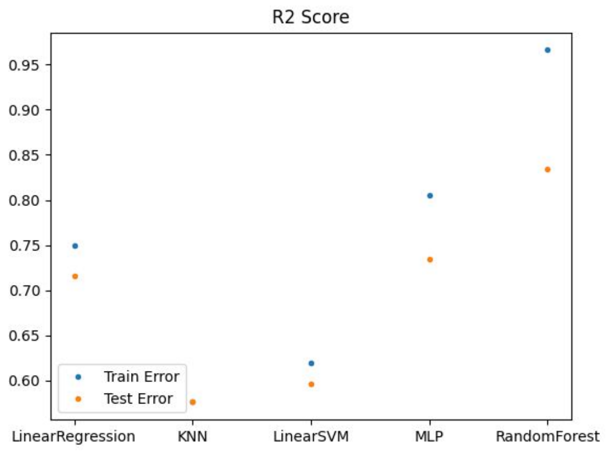
When using RF, the optimal values change with each execution. The hyperparameters we explore different values for are: the maximum depth of the trees in the forest, the number of estimators and the minimum number of samples in a node to become a leaf.
On a concrete execution, we observed the best results with 500 as the maximum depth, 10 for the number of estimators, and 1 for the minimum number of samples in a leaf.

4.3 Final Model

For the final model, we made two different comparisons, we plotted the mean squared error and the mean absolute error of the train and test set, both errors are the average of all the different folds, using the best hyper-parameters. We can see that most models perform really well, but only one is the winner. Even though random forest has better test results, we recommend using the Multilayer Perceptron, because it has less variance which would be better in a real-world example, another advantage of this model is that we could create an online model in a real-world environment. Another advantage of using the MLP is that in the future we could tune the hyperparameters even more to get even better results.







5 Conclusion

Given the shortage of samples on our dataset, we truly think that our results are quite consistent. Thus, we can affirm that we are confident in our success in this project. Even though we still have some doubts about extracted features and if there was a way that scapes our knowledge to handle them and extract meaningful new data from there.

As a possible extension for this project, we would like to explore more different values for MLP hyperparameters, as we really think that we could improve our result even further if we had more computational power and time available.

6 References

1. Géron, A. (2020). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. Beijing: O'Reilly.

2. Alpaydin, E. (2020). *Introduction to machine learning*. Cambridge, MA: The MIT Press.

3. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011, Retrieved January 07, 2021, [online] from: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

4. Pandas User Guide. (n.d.). Retrieved January 07, 2021, [online] from: https://pandas.pydata.org/docs/user_guide/index.html

5. NumPy User Guide., (n.d.). Retrieved January 07, 2021, [online] from: https://numpy.org/doc/stable/contents.html

6. Matplotlib User Guide., November 12, 2020, Retrieved January 07, 2021, [online] from: https://matplotlib.org/3.3.3/contents.html

7. Albon, C. (2018). *Machine learning with Python cookbook: Practical solutions from preprocessing to deep learning*. Sebastopol, CA: O'Reilly Media.

8. Bruce, P. C., Bruce, A. G., & Gedeck, P. (2020). *Practical statistics for data scientists: 50+ essential concepts using R and Python*. Beijing: O'Reilly.

9. UCI Machine Learning Repository: Automobile Data Set. (n.d.). Retrieved January 07, 2021, [online] from: https://archive.ics.uci.edu/ml/datasets/automobile

10. Pearson correlation coefficient. (2020, December 31). Retrieved January 07, 2021, [online] from: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient