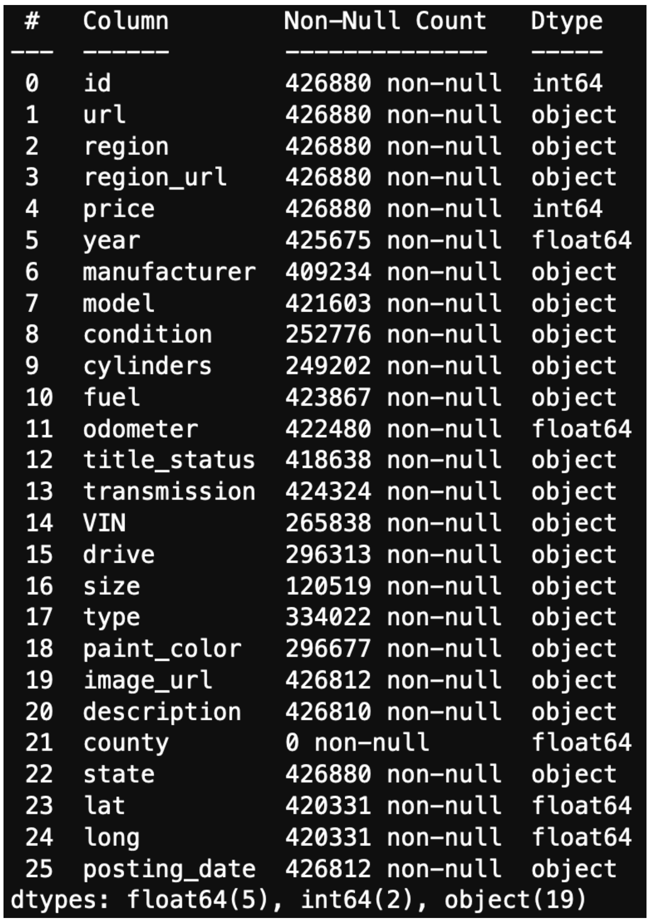
Advanced Data Mining Project

Emirhan Aykan

Car price prediction using the Craigslist dataset

Problem

The problem that is solved with this project is prediction of car prices using the Craigslist dataset. Craigslist is the world's largest collection of used vehicles for sale. But the data as itself is not sufficient to create a prediction model. Thus, the main problem to tackle was to discriminate the necessary data, and use the important features in the model with feature selection methods.



```
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            426880 non-null  int64
 1   url           426880 non-null  object
 2   region        426880 non-null  object
 3   region_url    426880 non-null  object
 4   price         426880 non-null  int64
 5   year          425675 non-null  float64
 6   manufacturer  409234 non-null  object
 7   model         421603 non-null  object
 8   condition     252776 non-null  object
 9   cylinders     249202 non-null  object
 10  fuel          423867 non-null  object
 11  odometer      422480 non-null  float64
 12  title_status  418638 non-null  object
 13  transmission  424324 non-null  object
 14  VIN           265838 non-null  object
 15  drive         296313 non-null  object
 16  size          120519 non-null  object
 17  type          334022 non-null  object
 18  paint_color   296677 non-null  object
 19  image_url     426812 non-null  object
 20  description   426810 non-null  object
 21  county        0 non-null       float64
 22  state         426880 non-null  object
 23  lat           420331 non-null  float64
 24  long          420331 non-null  float64
 25  posting_date  426812 non-null  object
dtypes: float64(5), int64(2), object(19)
```

The figure gives us information about the data. It requires a lot of preprocessing before starting on the model. Such as a low number of numeric columns (even with the ones that are not going to be included in the input columns), very high number of null values, existence of columns that uniquely identify a certain car as features, and columns that are not important.
After the processing of the data, the next step was to use feature selection methods to determine which one of the columns shall be used for the model.

Data: https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data

Description of the method

Firstly, during preprocessing, I have used different methods for some of the different columns that will be explained in the details of the used methods. At the beginning, I approached it more strictly regarding the elimination of the data, but later on I tried to use as much of it as possible.

I approached the problem with the trial and error method, and thus there was a lot of testing for the selection of suitable columns. I have used a correlation map (Seaborn library) for feature selection. I have used the spearman correlation method because the data was not linear. But even after determining the correlation between the price (output column) and the rest of the columns, I didn’t take it into consideration at the beginning, during the testing.

I have used the RandomForestRegressor for the testing of all stances of the preprocessed data. I was using the ‘score’ method from the scikit-learn library at the beginning, and at some point with the implementation of my own neural network I was also using the mean squared error loss function in order to compare the results more accurately. In addition, in the early stages of the testing, I have used hold-out validation with RandomForestRegressor with R squared metric. Later on, I have used cross-validation to evaluate the performance of the deep learning model I’ve created. And finally, while using filter-based feature selection method, I have used K-fold cross-validation evaluation. It is important to note that on each step I was using the MSE loss function, and also the R-squared metric while the RandomForestRegressor involved in the evaluation.

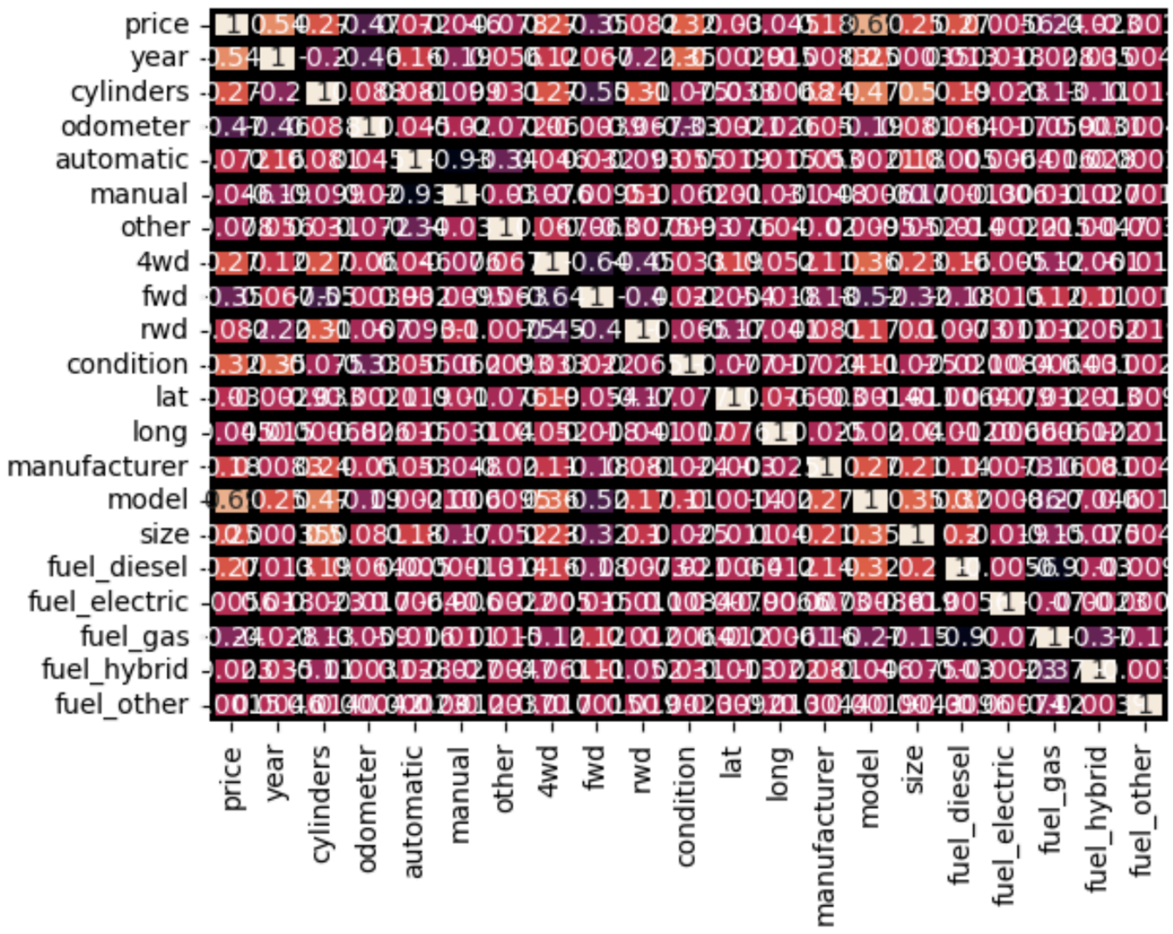
Finally, I used hyperparameter tuning for the comparison of the results I got from both from the model I have created and RandomForestRegressor.

Details

In this section I will try to explain the methods I have used step by step in detail.

For starters, I will explain the data preprocessing. I have mentioned that I used different methods for different columns. For instance, I used hot encoding for the columns that didn’t have a large number of unique values, such as transmission, fuel and drive. Then, I have used mapping for the size and the condition columns. The reason I have used mapping for these columns is that the increasing number in the newly created columns match the different values in the original column. For example, if the number 0 presents the smallest size of a car, the number N, N being the biggest number of the different values in the column, would present the biggest size of it. And finally for the manufacturer and the model column, I took the mean value of the column price for each manufacturer and model, then normalize it. This might not be very accurate for the correlation map, but it was much more efficient than adding the model column as 2000 new columns with hot encoding.

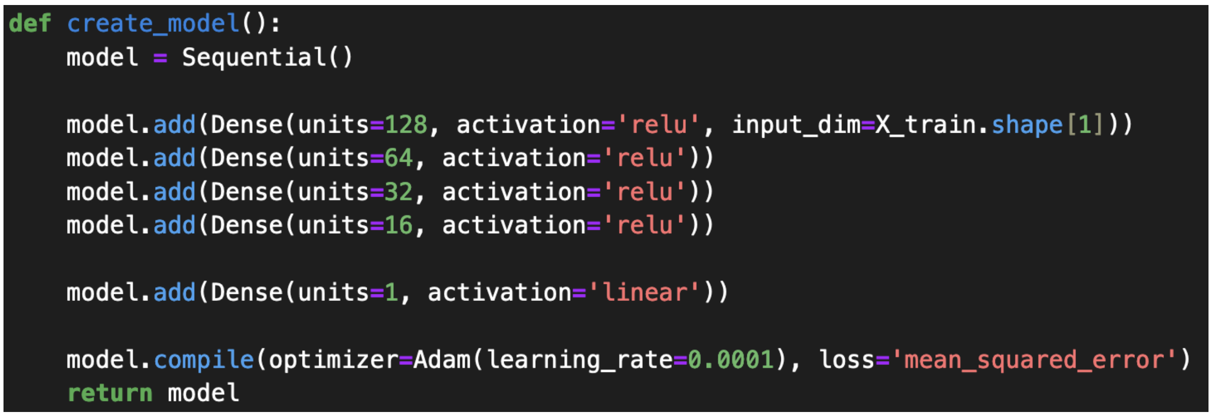
Final version of the correlation heatmap:



For the testing of the data, I didn’t do it at the end of all the data preprocessing. I was performing them simultaneously after including, excluding or altering the data, and comparing the results using the ‘score’ method from the scikit-learn library. This was kind of a less efficient feature selection method but a very effective one. The changes made in the input data could clearly make the difference in the evaluation of RandomForestRegressor model. I was examining the results from the score method after each step, and finally after I was done with all the data preprocessing I used RandomizedSearchCV for hyperparameter tuning for RandomForestRegressot in order to compare the results I got previously.

After this step, I started working on my own neural network. I have started by using 3 Dense layers and mean squared error loss function, then moved on with manual and automatic hyperparameter tuning. I could get the best results with 5 Dense layers due to the complexity of the model and because of the model being a regression model. Determination of the optimizer, learning rate, batch size and the number of epochs mainly based on hyperparameter tuning as I mentioned.

The final version of the model:

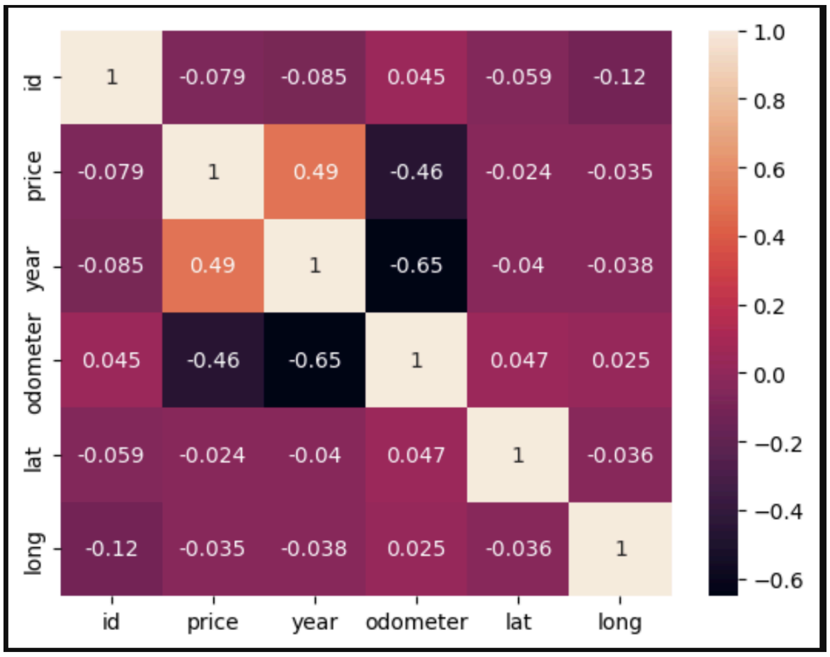


```
def create_model():
    model = Sequential()

    model.add(Dense(units=128, activation='relu', input_dim=X_train.shape[1]))
    model.add(Dense(units=64, activation='relu'))
    model.add(Dense(units=32, activation='relu'))
    model.add(Dense(units=16, activation='relu'))

    model.add(Dense(units=1, activation='linear'))

    model.compile(optimizer=Adam(learning_rate=0.0001), loss='mean_squared_error')
    return model
```

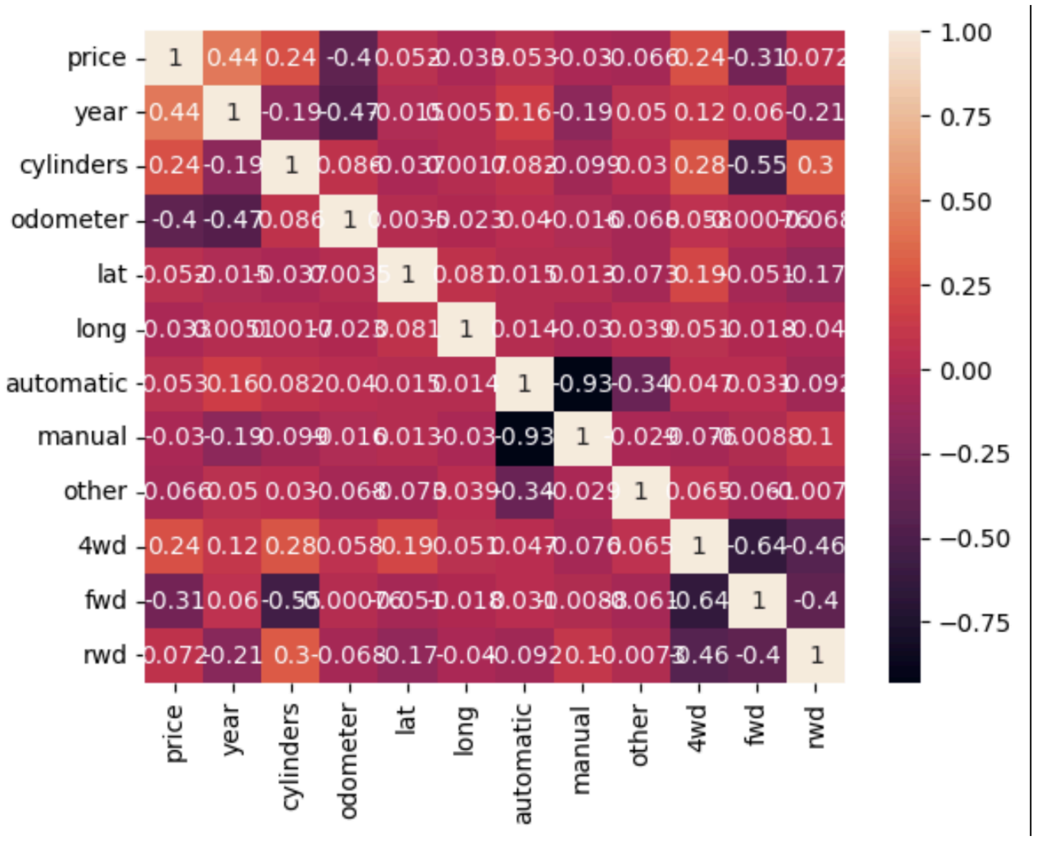
And finally, I did more tests with RandomForestRegressor in order to be more precise with the feature selection. I have determined certain levels to filter out some of the columns starting with big differences such as 10%, 20% up to 70%. These percentages represent the correlation, if the percentage selected is 50%, the model only considers the columns that had over 0.5 correlation with the column 'price'. But the results I was getting were worse than the initial one. Considering this, I decreased the levels to much smaller percentages such as 1%, 2%, 3% up to 10%, of course with the consideration of the number of columns that were being filtered out. In this scenario, I got the best results with 3%, again I used the score method to get the results. Even though the best result I got was not much better than the initial result, I wanted to move on with it but I wanted to consider the probability of getting lucky with the train test split method, so I tried using cross validation prediction in order to be sure and finally it was determined that the best percentage to be used was only 1%, which was excluding 2 columns from the data.

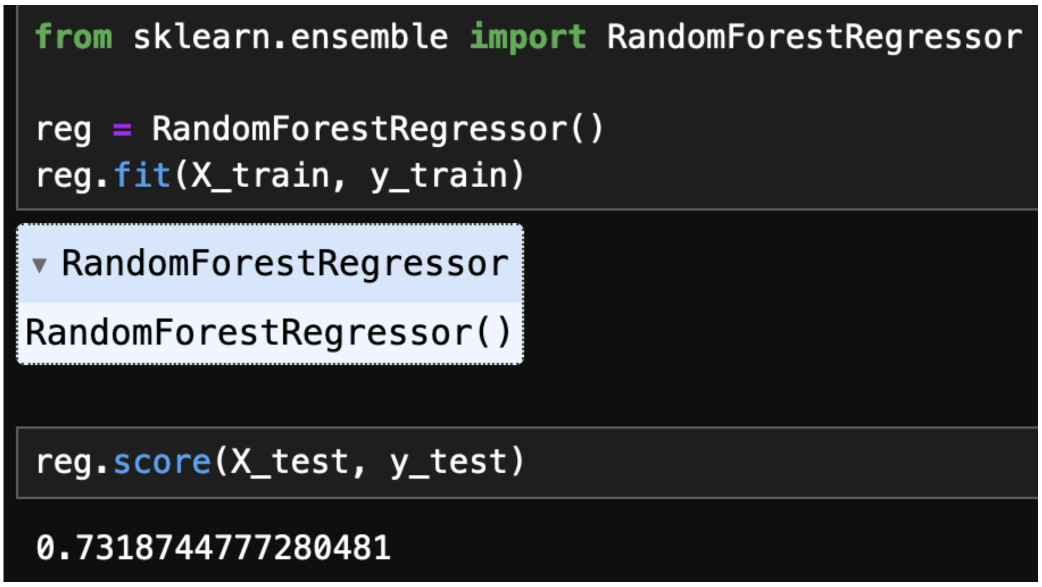
Results

The initial version of the correlation map only including numeric columns:



And after step by step, I was testing the new data on RandomForestRegressor:



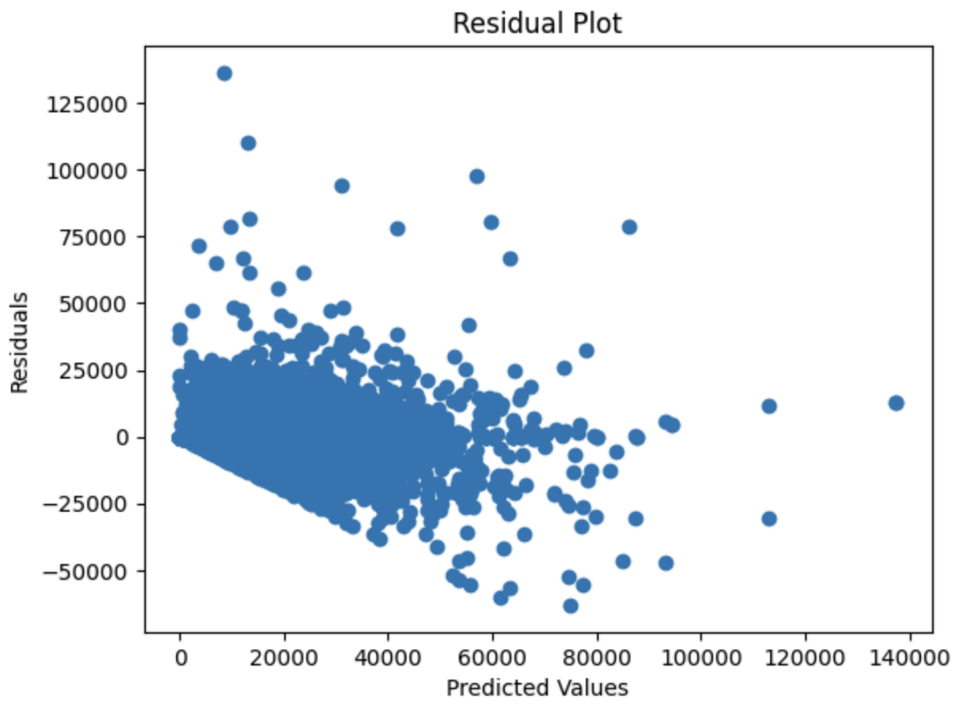


```
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor()
reg.fit(X_train, y_train)
```

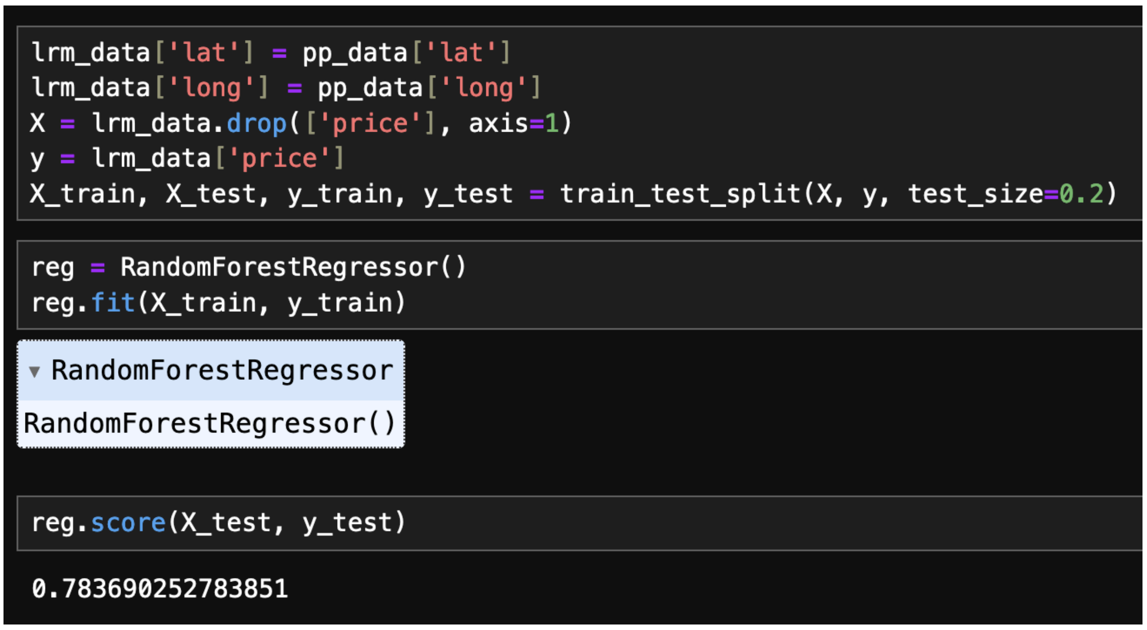
```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
reg.score(X_test, y_test)
```

```
0.7318744777280481
```



The initial results were much worse than the one I got in this one because there were some extreme outliers. Some cars had millions and millions of kilometers on their odometers, or a regular car would cost 0 or hundred, even billions of dollars. And it was possible to gather these first results after excluding those outliers.



```
lrm_data['lat'] = pp_data['lat']
lrm_data['long'] = pp_data['long']
X = lrm_data.drop(['price'], axis=1)
y = lrm_data['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
reg = RandomForestRegressor()
reg.fit(X_train, y_train)
```

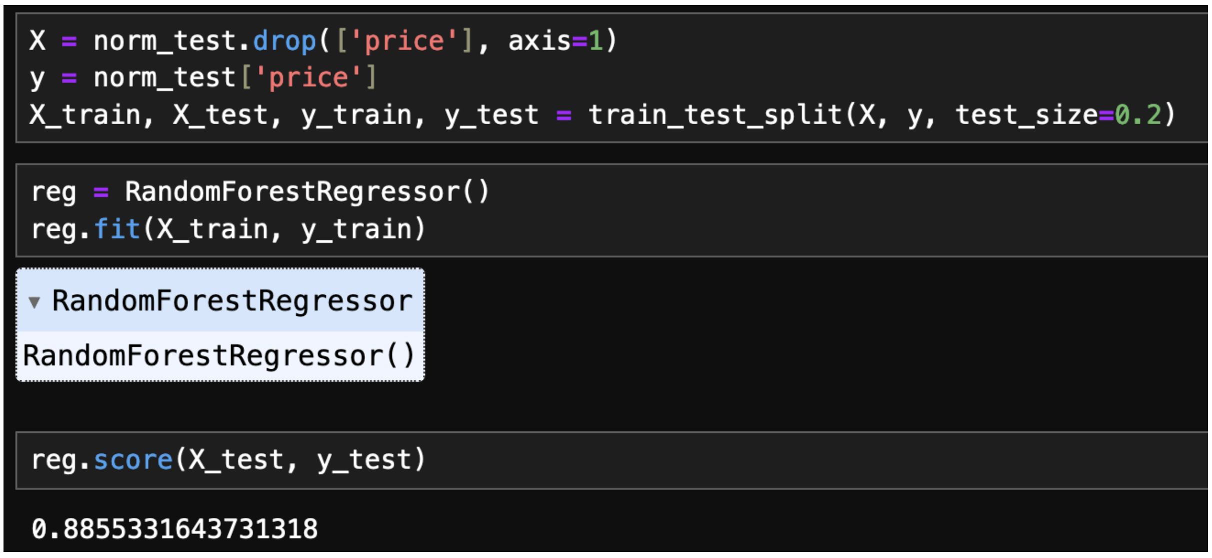
```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
reg.score(X_test, y_test)
```

```
0.783690252783851
```

Another result that I have received with a 0.05 score is by including new data that I thought would be useless initially, and including some by mapping.

The best result I could get by using the final version of the data, that has the correlation heatmap mentioned in the 'details' section, had the result:



```
X = norm_test.drop(['price'], axis=1)
y = norm_test['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
reg = RandomForestRegressor()
reg.fit(X_train, y_train)
```

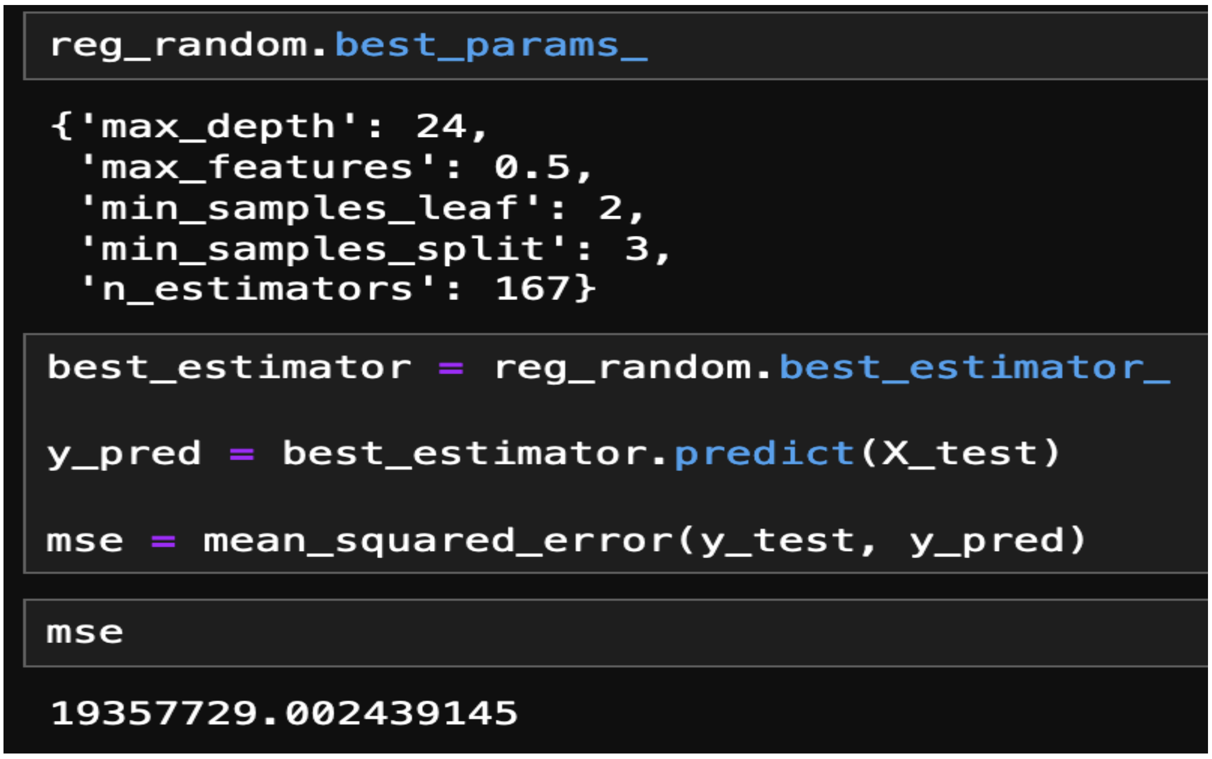
```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
reg.score(X_test, y_test)
```

```
0.8855331643731318
```

Which is 0.15 higher than the initial version.

In the next figure the best parameters for RandomForestRegressor was determined using RandomizedSearchCV for hyperparameter tuning:



```
reg_random.best_params_
```

```
{'max_depth': 24,
 'max_features': 0.5,
 'min_samples_leaf': 2,
 'min_samples_split': 3,
 'n_estimators': 167}
```

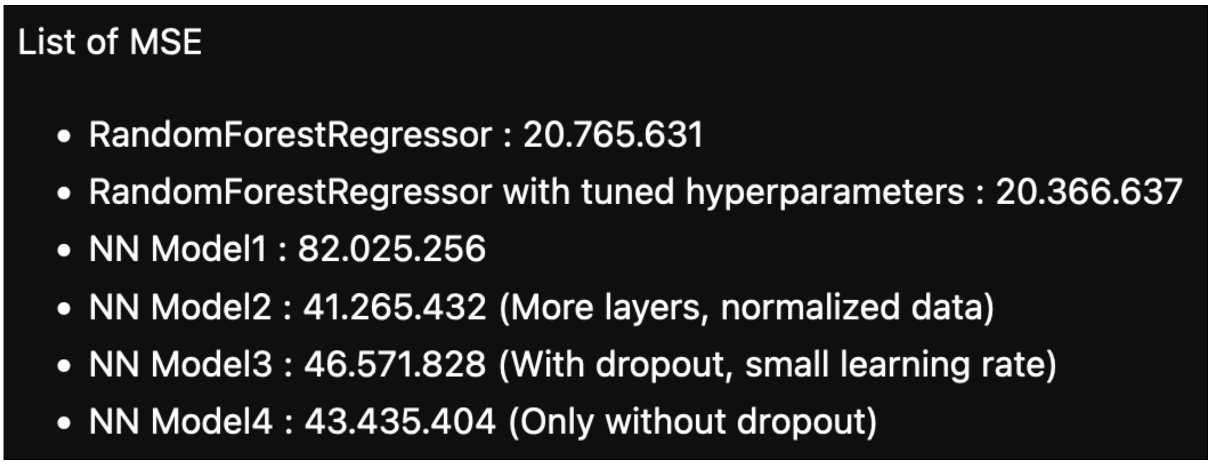
```
best_estimator = reg_random.best_estimator_

y_pred = best_estimator.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
```

```
mse
```

```
19357729.002439145
```

And for this result we can see the first usage of mean squared error for the first the before using the model I have created for comparison.

The following list is the list of mean squared error values I got from manual/automatic hyperparameter tuning:

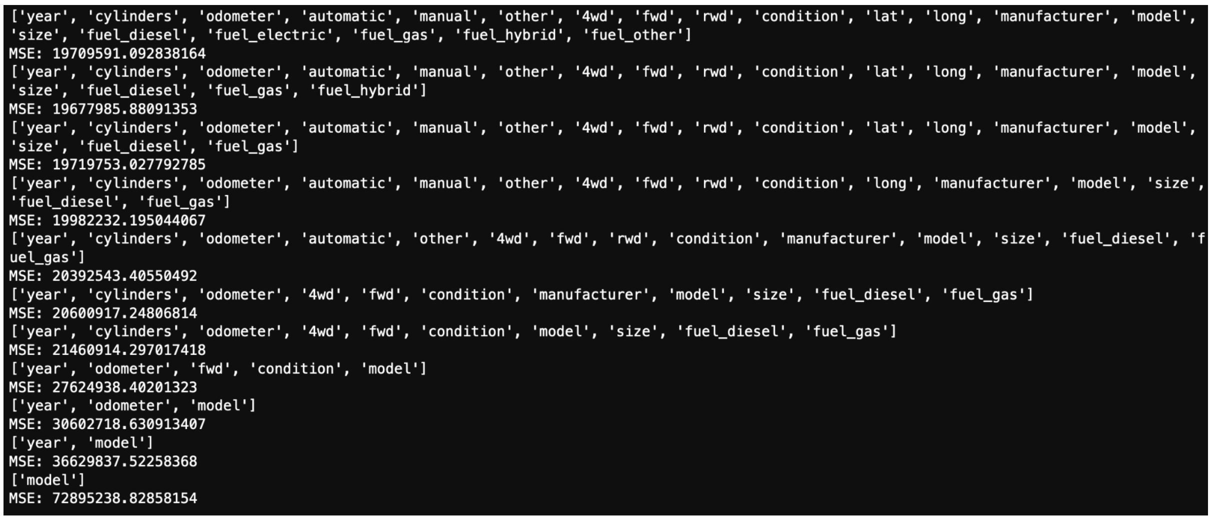


List of MSE

- RandomForestRegressor : 20.765.631
- RandomForestRegressor with tuned hyperparameters : 20.366.637
- NN Model1 : 82.025.256
- NN Model2 : 41.265.432 (More layers, normalized data)
- NN Model3 : 46.571.828 (With dropout, small learning rate)
- NN Model4 : 43.435.404 (Only without dropout)

The values here are not the best or worse values but the average of what I have gathered by many tests.

<u>And finally, the following figure shows the mean squared error values I have gathered with many different filterings of the input data using correlation:</u>



```
['year', 'cylinders', 'odometer', 'automatic', 'manual', 'other', '4wd', 'fwd', 'rwd', 'condition', 'lat', 'long', 'manufacturer', 'model',
'size', 'fuel_diesel', 'fuel_electric', 'fuel_gas', 'fuel_hybrid', 'fuel_other']
MSE: 19709591.092838164
['year', 'cylinders', 'odometer', 'automatic', 'manual', 'other', '4wd', 'fwd', 'rwd', 'condition', 'lat', 'long', 'manufacturer', 'model',
'size', 'fuel_diesel', 'fuel_gas', 'fuel_hybrid']
MSE: 19677985.88091353
['year', 'cylinders', 'odometer', 'automatic', 'manual', 'other', '4wd', 'fwd', 'rwd', 'condition', 'lat', 'long', 'manufacturer', 'model',
'size', 'fuel_diesel', 'fuel_gas']
MSE: 19719753.027792785
['year', 'cylinders', 'odometer', 'automatic', 'manual', 'other', '4wd', 'fwd', 'rwd', 'condition', 'long', 'manufacturer', 'model', 'size',
'fuel_diesel', 'fuel_gas']
MSE: 19982232.195044067
['year', 'cylinders', 'odometer', 'automatic', 'other', '4wd', 'fwd', 'rwd', 'condition', 'manufacturer', 'model', 'size', 'fuel_diesel', 'f
uel_gas']
MSE: 20392543.40550492
['year', 'cylinders', 'odometer', '4wd', 'fwd', 'condition', 'manufacturer', 'model', 'size', 'fuel_diesel', 'fuel_gas']
MSE: 20600917.24806814
['year', 'cylinders', 'odometer', '4wd', 'fwd', 'condition', 'model', 'size', 'fuel_diesel', 'fuel_gas']
MSE: 21460914.297017418
['year', 'odometer', 'fwd', 'condition', 'model']
MSE: 27624938.40201323
['year', 'odometer', 'model']
MSE: 30602718.630913407
['year', 'model']
MSE: 36629837.52258368
['model']
MSE: 72895238.82858154
```

Discussion

Regarding what I have presented in the 'Results' section, I think it makes the most sense to consider the last figure with input data filtering. But also, it is very crucial to consider the results that I have gathered before, and after hyperparameter tuning for both the neural network and RandomForestRegressor model. During the trial and error process I have gathered many other results that didn't make it into the report since the loss function was quite big, over 100 million up to 2 billion.

The most challenging part of the projects was creating the neural network and optimizing it. Not only was it taking so much time to run, the results were not always useful. It required a long time of research determining not only the best parameters to be used, but even the values that could be tested in order to find the best parameters, even with hyperparameter tuning using randomized search. And of course, even though it was not as nearly as hard as the neural network, I had similar problems with RandomForestRegressor.

But I can say without a doubt that I have learned many new things working on this project. Engaging in this project has been truly enjoyable, offering both intellectual stimulation and practical insights. It has been a rewarding experience that significantly enhanced my understanding of applied machine learning and data science.