# METCS521 Final Project Submission

# Predict Used Car Price with Random Forest Model


# Author: Xi Shen, Shuhan Li, Peiju Yu


## Introduction:

Our project is about the price forecasting of used cars. We called data from the used car market, where we could see various aspects of the car, such as manufacturer name, model name, color, milages, transmission, and so on. After removing some unnecessary columns, three histogram graphs were drawn. Then we digitized the various categories in the dataset to get a more accurate correlation. using the random forest model, we divided the data in the dataset into 20% of the test set and 80% of the training set, got our prediction results, and obtained the important feature variables by integration. In addition, in order to cope with different market conditions, three other prediction models were made, all of which can get machine-predicted used car prices that can compare well with the market situation.

## Downloading:

Our script is Final Project-Used Car Price. ipynb and the users need to run the dataset named cars.csv, bringing it into the environment. The dataset includes a lot of cars' information, and the ipynb file will give our steps.

## Dataset Describe:

After downloading the data, we can see the different variables in the data through the head() function, a total of 30 variables, and through the describe function can be derived from the different variables count, mean, std, min, and max values. Next, through

the info() function we analyzed the data type has some variables are objects, and some variables are int64 and bool.

| | manufacturer_name | model_name | transmission | color | odometer_value | year_produced | en |
|---|---|---|---|---|---|---|---|
| **0** | Subaru | Outback | automatic | silver | 190000 | 2010 | |
| **1** | Subaru | Outback | automatic | blue | 290000 | 2002 | |
| **2** | Subaru | Forester | automatic | red | 402000 | 2001 | |
| **3** | Subaru | Impreza | mechanical | blue | 10000 | 1999 | |
| **4** | Subaru | Legacy | automatic | black | 280000 | 2001 | |

5 rows × 30 columns

| | odometer_value | year_produced | engine_capacity | price_usd | number_of_photos | |
|---|---|---|---|---|---|---|
| **count** | 38531.000000 | 38531.000000 | 38521.000000 | 38531.000000 | 38531.000000 | 3 |
| **mean** | 248864.638447 | 2002.943734 | 2.055161 | 6639.971021 | 9.649062 | |
| **std** | 136072.376530 | 8.065731 | 0.671178 | 6428.152018 | 6.093217 | |
| **min** | 0.000000 | 1942.000000 | 0.200000 | 1.000000 | 1.000000 | |
| **25%** | 158000.000000 | 1998.000000 | 1.600000 | 2100.000000 | 5.000000 | |
| **50%** | 250000.000000 | 2003.000000 | 2.000000 | 4800.000000 | 8.000000 | |
| **75%** | 325000.000000 | 2009.000000 | 2.300000 | 8990.000000 | 12.000000 | |
| **max** | 1000000.000000 | 2019.000000 | 8.000000 | 50000.000000 | 86.000000 | |

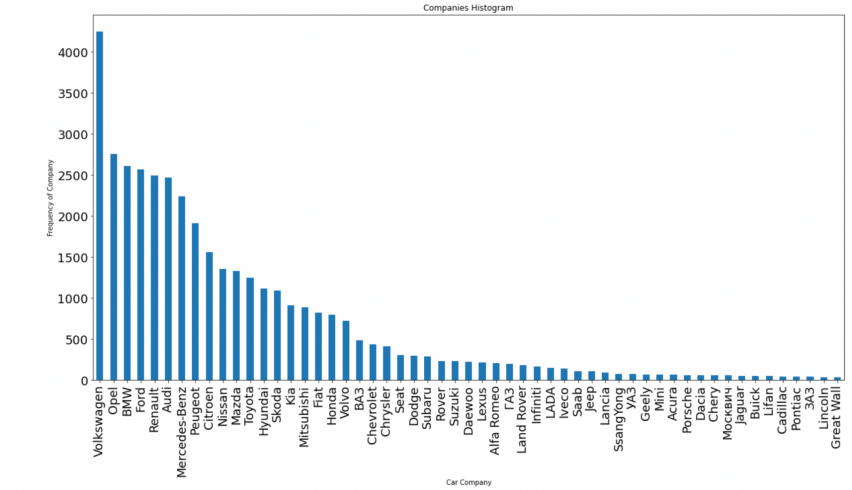
We find some missing values in the data, and irrelevant columns, such as “feature_0”, “feature_1” ... we drop these values and get a new data frame.

```
manufacturer_name      0
model_name             0
transmission           0
color                  0
odometer_value         0
year_produced          0
engine_fuel            0
engine_has_gas         0
engine_type            0
engine_capacity       10
body_type              0
has_warranty           0
state                  0
drivetrain             0
price_usd              0
is_exchangeable        0
location_region        0
number_of_photos       0
up_counter             0
feature_0              0
feature_1              0
feature_2              0
feature_3              0
feature_4              0
feature_5              0
feature_6              0
feature_7              0
feature_8              0
feature_9              0
```

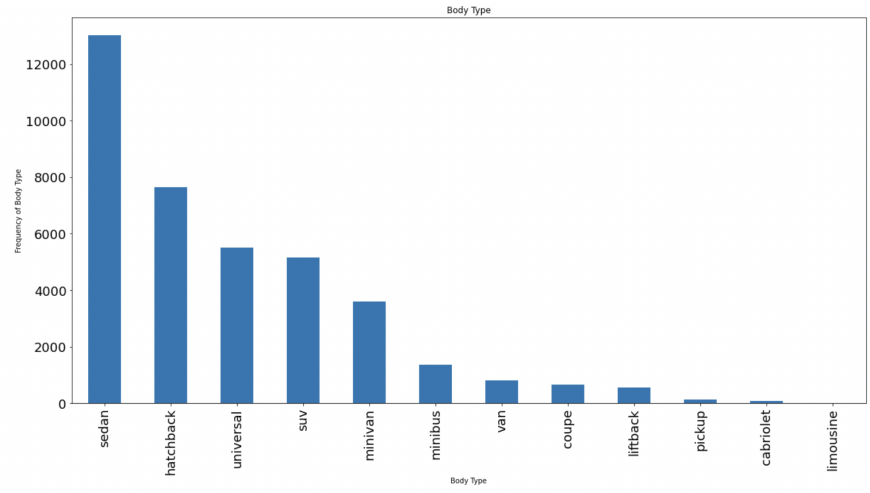
| | manufacturer_name | model_name | transmission | color | odometer_value | year_produced | en |
|---|---|---|---|---|---|---|---|
| **0** | Subaru | Outback | automatic | silver | 190000 | 2010 | |
| **1** | Subaru | Outback | automatic | blue | 290000 | 2002 | |
| **2** | Subaru | Forester | automatic | red | 402000 | 2001 | |
| **3** | Subaru | Impreza | mechanical | blue | 10000 | 1999 | |
| **4** | Subaru | Legacy | automatic | black | 280000 | 2001 | |
| **5** | Subaru | Outback | automatic | silver | 132449 | 2011 | |
| **6** | Subaru | Forester | automatic | black | 318280 | 1998 | |
| **7** | Subaru | Legacy | automatic | silver | 350000 | 2004 | |
| **8** | Subaru | Outback | automatic | grey | 179000 | 2010 | |
| **9** | Subaru | Forester | automatic | silver | 571317 | 1999 | |

## Data Analysis:

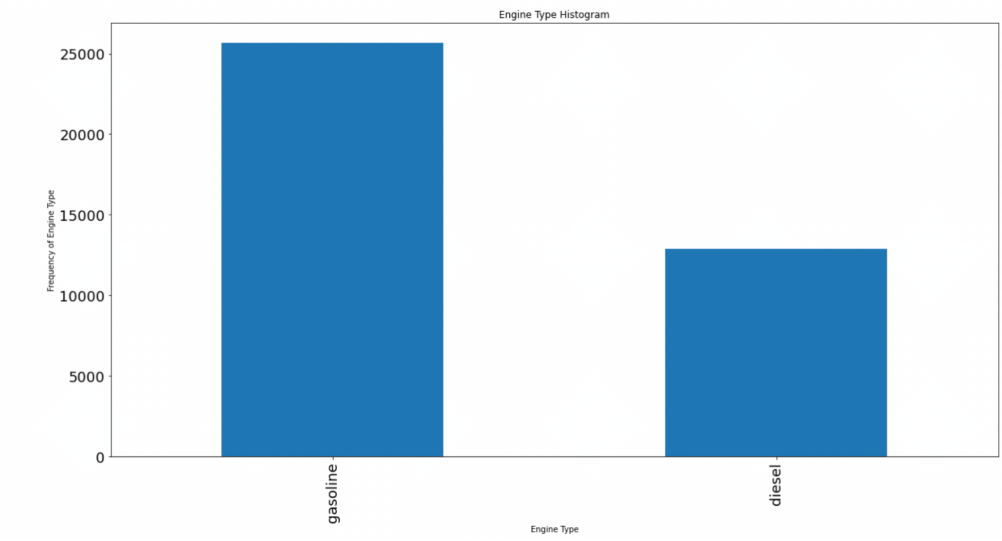
First, we used a histogram to describe the relationship between car companies and the frequency of companies. This bar chart shows the relationship between the frequency of cars sold by each car brand. The average sales volume of used German cars is higher than that of Japanese cars and American cars, and the most popular brand among them is Volkswagen.



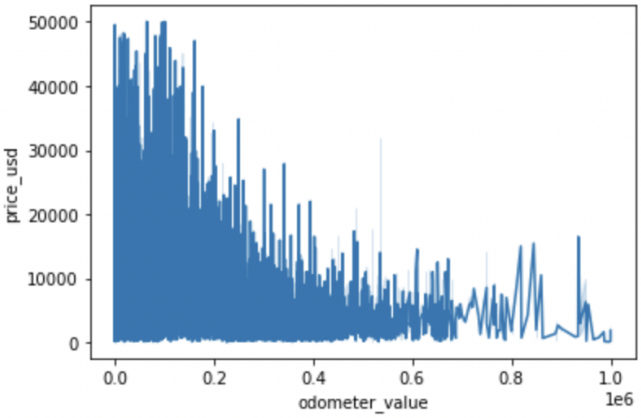

The second histogram is about the relationship between car type and sales. In daily life, the vast majority of consumers will choose small cars as their daily travel tools. Because they are generally cheap. As can be seen from the second bar chart, the sales of sedans have always been at the top of the list in the field of used car sales.

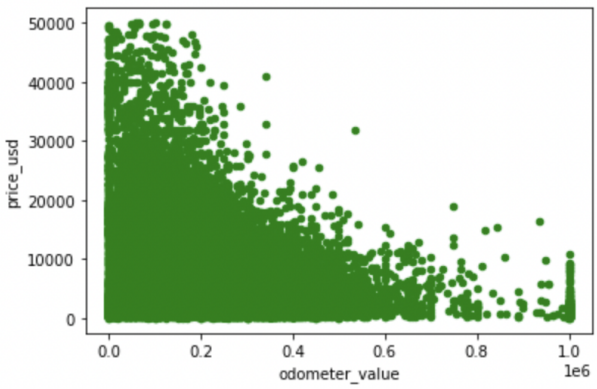


The last bar chart is related to the classification of vehicle energy types. In the traditional used car sales range, used cars are mainly divided into gasoline cars and diesel cars. In terms of used car sales, gasoline vehicles are still the main force in the used car sales market. Since diesel vehicles are mainly for commercial transportation or off-road, it can be seen that the sales of diesel vehicles are not as good as those of gasoline vehicles.

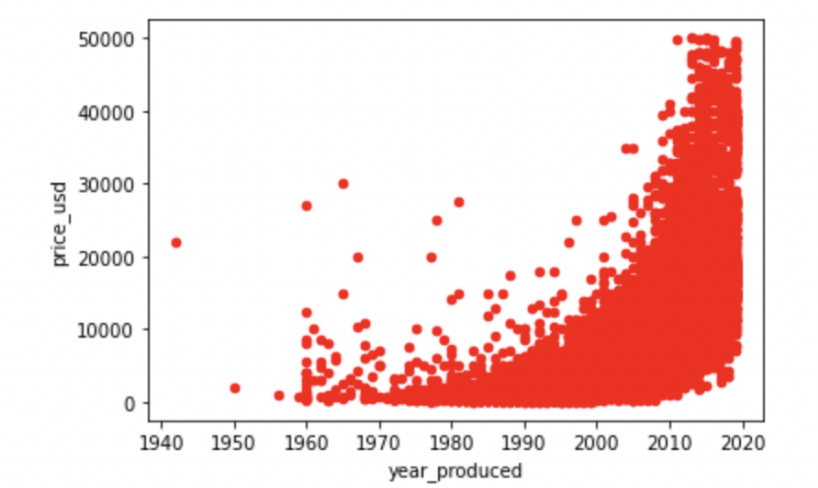



Second, we refined the relationship between used car mileage and sales price using a line plot and a dot plot. Since the mileage of a car is a factor that directly affects the selling price. And a line plot and dot plot can intuitively show the factors that the lower the mileage, the higher the price.



Finally, we used a dot plot to show the relationship between the year the car was made and the sales price. In the dot plot, the year the used car is sold is closer to the current time period, and the more expensive it will be.




**Random Forest Model:**

Random forest is actually a special kind of bagging method, which uses decision trees as a model in bagging. First, the bootstrap method is used to generate m training sets, then, for each training set, a decision tree is constructed, and when the nodes find features for splitting, instead of finding the one that maximizes the index (e.g., information gain) for all features, a portion of the features are randomly selected, and the optimal solution is found among the selected features and applied to the nodes for splitting. The random forest approach is actually equivalent to sampling both samples and features because of the idea of bagging, i.e., integration (if the training data is viewed as a matrix, as is common in practice, then it is a process where both rows and columns are sampled), so overfitting can be avoided.

It has many advantages: it performs well on datasets, it has a great advantage over other algorithms on many current datasets, it can handle very high dimensional (many features) data and does not need to do feature selection, and it can give which features are more important after

training, it uses an unbiased estimation of the generalization error when creating a random forest, it is fast in training, it can detect interactions between features during training, it is easy to make a parallelization method.

Therefore, the first step is to transform the data from category variables into numerical variables, which is called data standardization.

```
# View Correlations
df.corr()
```

| | manufacturer_name | model_name | transmission | color | odometer_val |
|---|---|---|---|---|---|
| manufacturer_name | 1.000000 | 0.216418 | 0.146214 | 0.050061 | -0.0480 |
| model_name | 0.216418 | 1.000000 | -0.047009 | 0.033049 | -0.0242 |
| transmission | 0.146214 | -0.047009 | 1.000000 | 0.099128 | 0.2296 |





| | manufacturer_name | model_name | transmission | color | odometer_val |
|---|---|---|---|---|---|
| color | 0.050061 | 0.033049 | 0.099128 | 1.000000 | 0.0242 |
| odometer_value | -0.048007 | -0.024228 | 0.229648 | 0.024233 | 1.0000 |
| year_produced | 0.000777 | 0.163829 | -0.385235 | -0.040747 | -0.4884 |
| engine_fuel | -0.028508 | -0.120546 | -0.144661 | -0.043006 | -0.2604 |
| engine_has_gas | -0.027704 | 0.011594 | -0.018669 | 0.004910 | 0.0577 |
| engine_type | -0.034166 | -0.120931 | -0.134511 | -0.042474 | -0.2463 |
| engine_capacity | -0.198033 | 0.073432 | -0.428596 | -0.099253 | 0.1057 |
| body_type | -0.053337 | 0.022452 | -0.148584 | -0.035392 | 0.0410 |
| has_warranty | 0.089543 | 0.061610 | -0.065499 | -0.010412 | -0.1895 |
| state | -0.040831 | -0.022800 | 0.005943 | 0.004483 | 0.0818 |
| drivetrain | -0.056542 | -0.201231 | 0.229088 | 0.060723 | 0.1967 |
| price_usd | -0.028763 | 0.157436 | -0.476201 | -0.067657 | -0.4209 |
| is_exchangeable | -0.034278 | -0.000562 | -0.027045 | -0.008914 | 0.0423 |
| location_region | 0.008158 | 0.025443 | -0.084227 | -0.012109 | -0.0927 |
| number_of_photos | -0.025495 | 0.039485 | -0.196863 | -0.045639 | -0.1435 |
| up_counter | -0.012103 | 0.012340 | -0.044324 | -0.001217 | -0.0209 |
| duration_listed | -0.012176 | 0.006139 | -0.031937 | 0.002253 | -0.0005 |

Then, we can run the code “df. columns”, showing us how many columns there are and what their name is now.

```
df.columns
```

```
Index(['manufacturer_name', 'model_name', 'transmission', 'color',
       'odometer_value', 'year_produced', 'engine_fuel', 'engine_has_gas',
       'engine_type', 'engine_capacity', 'body_type', 'has_warranty', 'state',
       'drivetrain', 'price_usd', 'is_exchangeable', 'location_region',
       'number_of_photos', 'up_counter', 'duration_listed'],
      dtype='object')
```

Just as we had hoped, the data in the data frame is normalized so that it does not give us excessive data bias, and the columns we need in the data frame are still there, while the columns we don't need are removed in the previous process.

Now, our goal is to build the model. We need to set up a training set and a test set in the model. For the accuracy of the prediction, we set up 80% of the training set and therefore only 20% of the test set, but we believe we will get a good result.

For the setting of independent variables, we selected "transmission", "odometer value", "year produced", "drive train", and "numbers of photos" as the factors affecting the price of used cars, and the dependent variable is direct "price usd". After completing the model using the training set, we brought the test set into the model and named it "rf2".

Of course, there is the very important step of obtaining the degree of influence of the independent variable on the dependent variable. Therefore, we need to use the concept of "feature importance", and here are the results we obtained.

```
important_features = pd.Series(data = rf.feature_importances_,index = X_train
important_features.sort_values(ascending = False, inplace = True)
important_features
```

```
year_produced       0.624997
drivetrain          0.142217
odometer_value      0.133110
number_of_photos    0.070785
transmission        0.028891
dtype: float64
```

We can see that the year of manufacture is the most influential factor on the price of a used car, reaching 0.62, while the mileage, which we thought was very important, i.e. "odometer value", is the third most influential factor on the price, at 0.13.

**Predict the Price**

After we get the predictive price model, we enter the data and get predicted prices. We input the transmission is automatic, the odometer value is 10,000, the year produced is 2020, the drivetrain is all, and the number of photos is 15. When we finish the value input, we get the predicted price is 39081.94.

```
predict_price()
```

```
please input the transmission(mechanical or automatic)automatic
please input the odometer value10000
please input the year produced2020
please input the drivetrainall
please input the number of photos15
39081.9419111445
```

## AdaBoost Regressor：

Adaboost was proposed by Freund and Schapire in 1997 to maintain a vector of distribution weights W over the entire training set, generate a classification hypothesis (base learner) y(x) by a weak classification algorithm using the training set given weights, and then calculate the error rate, and use the obtained error rate to update the vector of distribution weights w, assigning larger weights to the misclassified samples, The misclassified samples are assigned larger weights and the correctly classified samples are assigned smaller weights. After each update, the same weak classification algorithm is used to generate new classification hypotheses, and the sequence of these classification hypotheses constitutes a multi-classifier.

These multi-classifiers are combined in a weighted way to obtain the final decision result. The previous learner changes the weight w and then goes through the next learner, and finally, all the learners together form the final learner. If a sample is misclassified in the previous learner, the weight corresponding to it is aggravated, and accordingly, the weight of the correctly classified sample is reduced.

```
# Build other models to predict prices
ab = AdaBoostRegressor()
ab.fit(X_train, Y_train)
Y_hat = ab.predict(X_test)
r2 = metrics.r2_score(Y_test, Y_hat)
r2
```

```
/Users/victoriali/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/vali
dation.py:63: DataConversionWarning: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  return f(*args, **kwargs)
0.6447888547084841
```

## Linear Regression:

The success achieved by random forest may mask its weaknesses. It is known that it may produce low-quality predictions when the samples fall in the low-explored region during training. This is typical in many real-world use cases where the data is not stationary and is beyond the range explored at the time of fitting. In this sense, linear models may be as good as tree-based models when modeling non-stationary systems, thanks to their ability to learn trend behavior. Why not combine them? Why not create a model that learns both linear and more complex relationships?

First, the linear model we chose fits the original data to approximate the target. Second (and finally), the random forest is trained on the same set of data to reproduce the residuals from

the previous step. The final prediction is the sum of the linear and forest predictions.

```
# Build other models to predict prices
lin = LinearRegression()
lin.fit(X_train, Y_train)
```



```
Y_hat = lin.predict(X_test)
r3 = metrics.r2_score(Y_test, Y_hat)
r3
```

```
0.5658747899694421
```

## GradientBoosting Regressor:

Unlike Adaboost, Gradient Boosting chooses the direction of gradient descent during the iteration to ensure the best final result. The loss function is used to describe the "reliability" of the model, assuming that the model is not overfitted, and the larger the loss function, the higher the error rate of the model. If our model is able to keep the loss function decreasing, it means that our model is constantly improving, and the best way to do this is to let the loss function decrease in the direction of its gradient.

```
# Build other models to predict prices
gb = GradientBoostingRegressor()
gb.fit(X_train, Y_train)
Y_hat = gb.predict(X_test)
r4 = metrics.r2_score(Y_test, Y_hat)
r4
```

```
/Users/victoriali/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/vali
dation.py:63: DataConversionWarning: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  return f(*args, **kwargs)
0.7848278065226124
```

**Conclusion:**

All car sales are done under the same model. Since the overall model is based on the conclusions of online sales data, all the results can directly represent the purchasing needs of people in real life. The final result will be close to real life and meet the purchasing needs of the people.

According to the used car market data predictions, we can learn that the lower the mileage the higher the price of used cars, and the more recent the year the higher the price. Volkswagen has the best sales volume and is the most popular in the used car market. People are more likely to choose cars and more likely to choose gasoline cars for their choice of models. With the price prediction model, we can predict the price of used cars by different variables.