

CS412 Machine Learning– TERM PROJECT

Report

Project Title: Car Price Prediction

Group Members: , Orkun Erdem, Selçuk Yılmaz, Çağan Veziroğlu, Asif Md Imtiaz

S

Supervisor(s): Berrin Yanikoglu

Date: 1.9.2022



1. PROJECT SUMMARY

As a group, we aim to predicting the price using the given features. To make this prediction, we need to choose the right regression. And we must change with Features and the model to develop the model we have chosen.

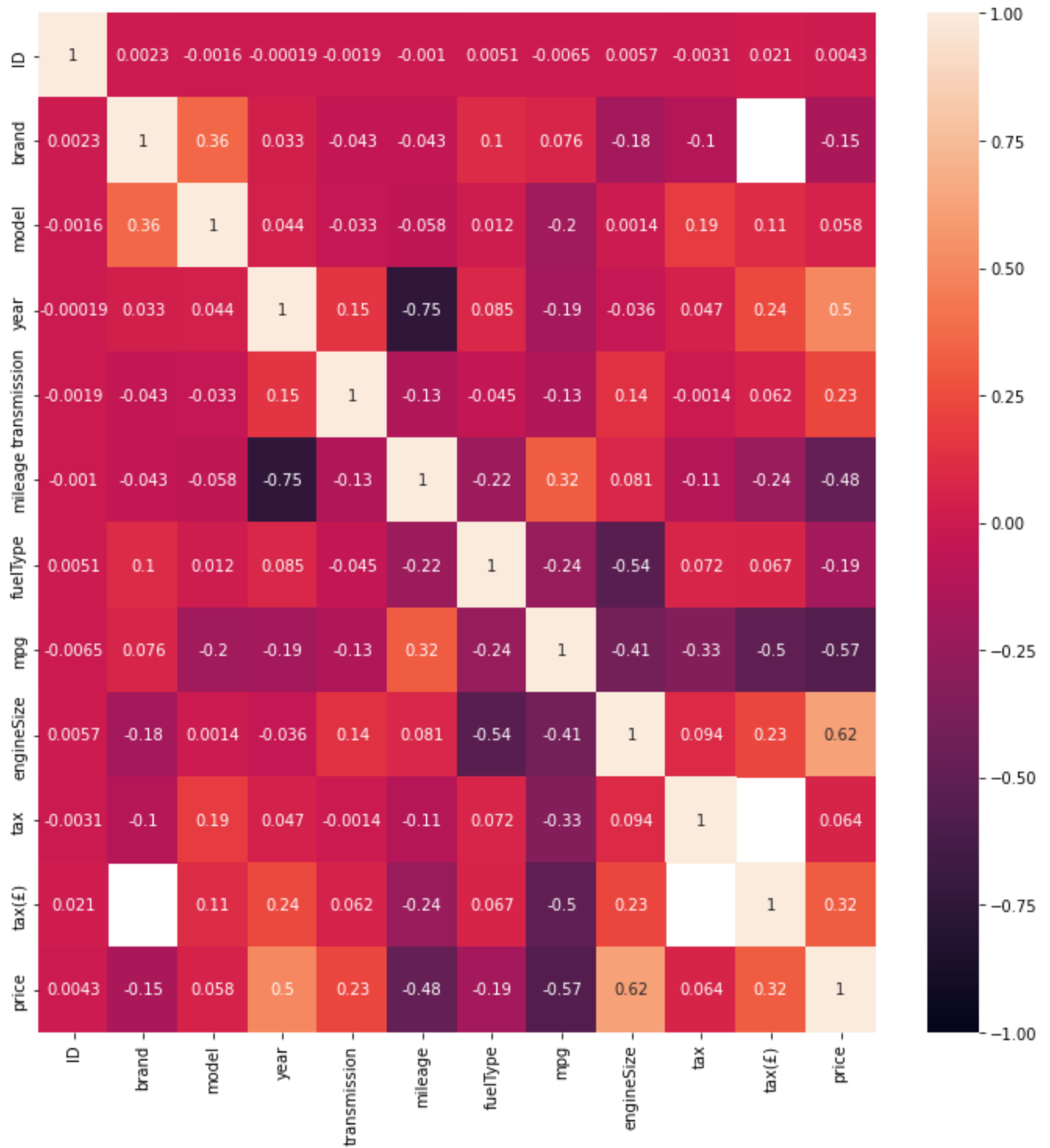
The features provided are as follows

Features:

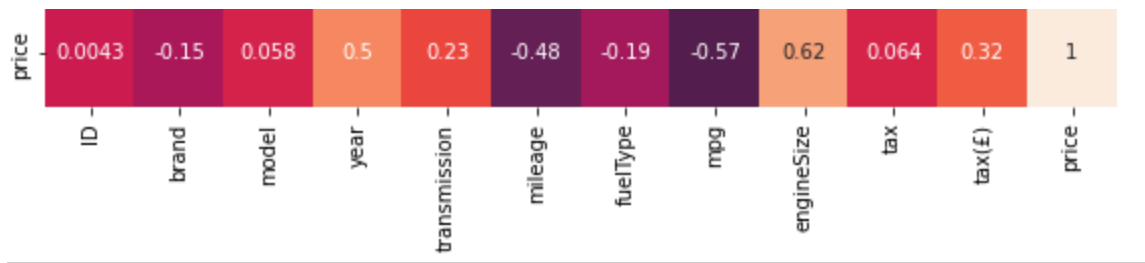
- ID (Unique id number for a given sample)
- brand
- model
- year (year of production)
- transmission
- mileage (how many miles has this car been used)
- fuel Type
- mpg (fuel consumption; miles per gallon)
- engine Size
- tax

2. DATA ANALYSIS

We drew some graphs to understand the data. The correlation heatmap was the first of these.



The important point here is that what we noticed with Price, some events are inverse proportions, and some are direct proportions.



3. DATA PREPARATION

We changed the numbers from the inversely proportional ones because we wanted to make the right proportion and reduced the multiplier value.

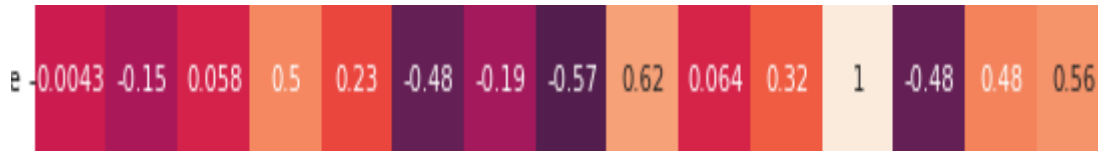
BEFORE

```
ID          59999.0
year        2020.0
mileage     323003.0
mpg         470.8
engineSize   6.6
tax         580.0
tax(£)      555.0
price      159999.0
dtype: float64
```

AFTER

```
ID          59999.0
year        2020.0
mileage     323003.0
mpg         470.8
engineSize   6.6
tax         580.0
tax(£)      555.0
price      159999.0
age          51.0
mil         399998.0
fuel        498.9
dtype: float64
```

By this way we scaled Datas. Logically, changing the inverse or direct proportion does not affect anything, but the numbers are scaled.



After that we fill none collums but we use three different type that None for object, Zero for numbers and Mode for features which about mode of cars.

```
cols_for_none = ('model', 'brand', 'fuelType')
for c in cols_for_none:
    X_train[c] = X_train[c].fillna("None")
    test[c] = test[c].fillna("None")

cols_for_zero = ('age', 'mileage', 'mpg', 'tax', 'tax(£)', 'mil', 'fuel')
for c in cols_for_zero:
    X_train[c] = X_train[c].fillna(0.0)
    test[c] = test[c].fillna(0.0)

cols_for_mode = ('engineSize', 'transmission')
for c in cols_for_mode:
    X_train[c] = X_train[c].fillna(X_train[c].mode())
    test[c] = test[c].fillna(test[c].mode())
```

We want to maximize accuracy so we fill mode nan with mode of feautres.

```
# ID brand model year transmission mileage fuelType mpg engineSize tax tax(£) price age mil fuel
# 0 ford Fiesta 2019.0 Manual 18.0 Petrol 60.1 2.0 145.0 145.0 9995.0 2.0 399982.0 439.9
X_train['engineSize'] = X_train['engineSize'].fillna(2.0)
X_train['transmission'] = X_train['transmission'].fillna('Manual')

test['engineSize'] = test['engineSize'].fillna(2.0)
test['transmission'] = test['transmission'].fillna('Manual')
```

All data fulfilled

```
Data columns (total 14 columns):
```

#	Column	Non-Null	Count	Dtype
0	ID	60000	non-null	int64
1	brand	60000	non-null	object
2	model	60000	non-null	object
3	year	59941	non-null	float64
4	transmission	60000	non-null	object
5	mileage	60000	non-null	float64
6	fuelType	60000	non-null	object
7	mpg	60000	non-null	float64
8	engineSize	60000	non-null	float64
9	tax	60000	non-null	float64
10	tax(£)	60000	non-null	float64
11	age	60000	non-null	float64
12	mil	60000	non-null	float64
13	fuel	60000	non-null	float64

```
Data columns (total 14 columns):
```

#	Column	Non-Null	Count	Dtype
0	ID	25555	non-null	int64
1	brand	25555	non-null	int64
2	model	25555	non-null	int64
3	year	25526	non-null	float64
4	transmission	25555	non-null	int64
5	mileage	25555	non-null	int64
6	fuelType	25555	non-null	int64
7	mpg	25555	non-null	int64
8	engineSize	25555	non-null	int64
9	tax	25555	non-null	int64
10	tax(£)	25555	non-null	float64
11	age	25555	non-null	int64
12	mil	25555	non-null	int64
13	fuel	25555	non-null	int64

Now we need to drop useless and repeated datas.

	Features	Scores
6	engineSize	38891.649594
5	mpg	27439.850078
8	age	20410.437209
3	mileage	18026.146259
9	mil	9679.414263
7	tax	7031.827084
10	fuel	4320.785326
2	transmission	3313.263383
4	fuelType	2310.403806
0	brand	1183.517122
1	model	199.832937

We decided that mpg is better however you can see those scores of ages is better than year. Also, mileage is better. So, we will drop “mil”, “year” and “fuel”. Also, ID is useless features and there was too much null at tax(euro) so we drop to.

Last datas which we use for regression is here you can observe that we did label encode and change all of them to int64. Data is ready for regression.

```
# Column Non-Null Count Dtype
---  ---
0 brand 60000 non-null int64
1 model 60000 non-null int64
2 transmission 60000 non-null int64
3 mileage 60000 non-null int64
4 fuelType 60000 non-null int64
5 mpg 60000 non-null int64
6 engineSize 60000 non-null int64
7 tax 60000 non-null int64
8 age 60000 non-null int64
dtypes: int64(9)
memory usage: 4.1 MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25555 entries, 0 to 25554
Data columns (total 9 columns):
# Column Non-Null Count Dtype
---  ---
0 brand 25555 non-null int64
1 model 25555 non-null int64
2 transmission 25555 non-null int64
3 mileage 25555 non-null int64
4 fuelType 25555 non-null int64
5 mpg 25555 non-null int64
6 engineSize 25555 non-null int64
7 tax 25555 non-null int64
8 age 25555 non-null int64
```

4. DECIDE REGRESSION MODEL

We tried different models on the train data to choose the one that gives the best results. Then we would develop the model with which we got the best results.

We use 4 different regression model

Random Forest Regressor Scores

Scores: [2030.31566156 2366.54993635 2164.8431763 2168.24836695 2376.24727474]

Mean: 2221.240883180341

Standard Deviation: 132.35033111125378

Gradient Boosting Regressor Scores

Scores: [3084.22683719 3318.81931998 3177.96369999 3237.18535525 3241.15762598]

Mean: 3211.8705676790496

Standard Deviation: 77.94767922181897

Linear Regression Scores

Scores: [4878.94871586 5310.71212066 5218.92228387 5105.09291254 5114.15859536]

Mean: 5125.566925658908

Standard Deviation: 144.4741103167083

xGB Scores

Scores: [3121.50417391 3342.18346855 3194.92036574 3196.00020125 3224.90334463]

Mean: 3215.9023108168826

Standard Deviation: 71.79873765357716

All of them had very high error mean but random forest could come back with a nice model. For this reason, we chose the random forest model.

5. REGRESSION BY RFM

```
import requests, io
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators = 50, random_state = 5)

model.fit(X_train, y_train)

y = model.predict(X_train)
rmse = float(format(np.sqrt(mean_squared_error(y_train, y)), '.3f'))
print("\nRMSE: ", rmse)
```

RMSE: 831.297

We did experiments on train data. During these trials, we tried to find the lowest ERROR. For this, we changed the estimators and random state values. We had the best results in the high estimator low random state. There were Runs that we got lower than in the image, but now it takes too long, but changes very little. It was overfit after a certain point.

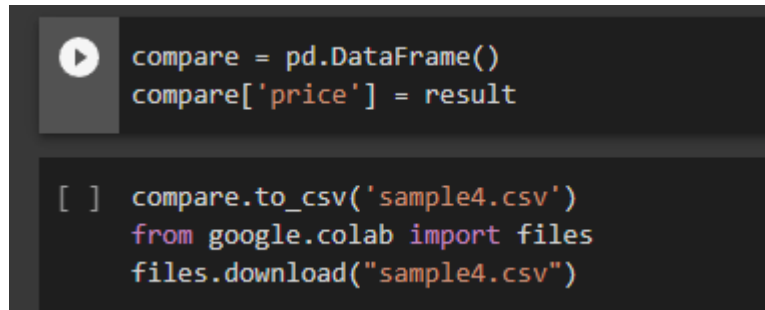
So we decided this was the best combination.

$n_estimators = 50, random_state = 5$

```
model = RandomForestRegressor(n_estimators = 50, random_state = 5)
model.fit(X_train, y_train)
result = model.predict(test)
```

6. SUBMISSION

To submit, we put it in the result dataframe and downloaded it. Then we uploaded it to kaggle. Our best result is 0.82688.



```
compare = pd.DataFrame()
compare['price'] = result

[ ] compare.to_csv('sample4.csv')
    from google.colab import files
    files.download("sample4.csv")
```

We share our colab you can enter link and observe our code.

COLAB:

https://colab.research.google.com/drive/1Zx1wq419SbCGA_ShopuaQ42tQCAOPBk3?usp=sharing