Definition of the prices of the cars

Content

Project Description

Car sales company would like to develop an app for the attraction of new clients, in which clients could get the market price of the car based on it's parameters. For that purpose it's required to train a model for the prediction of the car price based on the data provided from company. It's required to train several model, compare it by quality, training time, prediction time and select the best model.

Important parameters of the model for client is:

- quality of prediction;
- prediction time;
- training time.

Data preparation

In [1]:
```
import numpy as np
import pandas as pd
```

```
import lightgbm as lgb
from sklearn.metrics import mean_squared_error as mse
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestRegressor
```

Data import and overview

In [2]:
```
DF_cars = pd.read_csv('autos.csv')
```

In [3]:
```
DF_cars.head()
```

Out[3]:

	DateCrawled	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Kilometer	RegistrationMonth	FuelType	Brand	NotRepaired	Date
0	2016-03-24 11:52:17	480	NaN	1993	manual	0	golf	150000	0	petrol	volkswagen	NaN	201
1	2016-03-24 10:58:45	18300	coupe	2011	manual	190	NaN	125000	5	gasoline	audi	yes	201
2	2016-03-14 12:52:21	9800	suv	2004	auto	163	grand	125000	8	gasoline	jeep	NaN	201
3	2016-03-17 16:54:04	1500	small	2001	manual	75	golf	150000	6	petrol	volkswagen	no	201
4	2016-03-31 17:25:20	3600	small	2008	manual	69	fabia	90000	7	gasoline	skoda	no	201

In [4]:
```
DF_cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354369 entries, 0 to 354368
Data columns (total 16 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   DateCrawled        354369 non-null  object
 1   Price              354369 non-null  int64
 2   VehicleType        316879 non-null  object
 3   RegistrationYear   354369 non-null  int64
 4   Gearbox            334536 non-null  object
 5   Power              354369 non-null  int64
 6   Model              334664 non-null  object
 7   Kilometer          354369 non-null  int64
 8   RegistrationMonth  354369 non-null  int64
 9   FuelType           321474 non-null  object
 10  Brand              354369 non-null  object
 11  NotRepaired        283215 non-null  object
 12  DateCreated        354369 non-null  object
 13  NumberOfPictures   354369 non-null  int64
 14  PostalCode         354369 non-null  int64
 15  LastSeen           354369 non-null  object
dtypes: int64(7), object(9)
memory usage: 43.3+ MB
```

In [5]:
```
DF_cars.describe()
```

Out[5]:

	Price	RegistrationYear	Power	Kilometer	RegistrationMonth	NumberOfPictures	PostalCode
count	354369.000000	354369.000000	354369.000000	354369.000000	354369.000000	354369.0	354369.000000
mean	4416.656776	2004.234448	110.094337	128211.172535	5.714645	0.0	50508.689087
std	4514.158514	90.227958	189.850405	37905.341530	3.726421	0.0	25783.096248
min	0.000000	1000.000000	0.000000	5000.000000	0.000000	0.0	1067.000000
25%	1050.000000	1999.000000	69.000000	125000.000000	3.000000	0.0	30165.000000
50%	2700.000000	2003.000000	105.000000	150000.000000	6.000000	0.0	49413.000000
75%	6400.000000	2008.000000	143.000000	150000.000000	9.000000	0.0	71083.000000
max	20000.000000	9999.000000	20000.000000	150000.000000	12.000000	0.0	99998.000000

Conclusion

- Dataset has 354369 rows and 15 columns;
- Target column - 'price'
- Numeric features are - 'registration year','power','kilometer','registration month','postal code'
- Categorical features are - 'VehicleType','Gearbox','Model','FuelType','Brand','NotRepaired','DateCreated'
- Data set has useless columns and null values, its required to perform data preparation prior to model training

Data preparation

In [6]:
```
# deletion of useless columns
DF_cars = DF_cars.drop(columns = ['DateCrawled','LastSeen'])
```

In [7]:
```
DF_cars.head()
```

Out[7]:

	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Kilometer	RegistrationMonth	FuelType	Brand	NotRepaired	DateCreated	Numl
0	480	NaN	1993	manual	0	golf	150000	0	petrol	volkswagen	NaN	2016-03-24 00:00:00	
1	18300	coupe	2011	manual	190	NaN	125000	5	gasoline	audi	yes	2016-03-24 00:00:00	
2	9800	suv	2004	auto	163	grand	125000	8	gasoline	jeep	NaN	2016-03-14 00:00:00	
3	1500	small	2001	manual	75	golf	150000	6	petrol	volkswagen	no	2016-03-17 00:00:00	
4	3600	small	2008	manual	69	fabia	90000	7	gasoline	skoda	no	2016-03-31 00:00:00	

In [8]:
```
# datatime value reformating
DF_cars['DateCreated'] = pd.to_datetime(DF_cars['DateCreated'], format='%Y-%m-%d')
```

In [9]:
```
DF_cars[DF_cars['RegistrationYear']> 2022]['RegistrationYear'].count()
```

Out[9]:
```
105
```

In [10]:
```
# deletion of incorrect data (year > 2022)
DF_cars =DF_cars.drop(index = DF_cars[DF_cars['RegistrationYear']> 2022].index).reset_index().drop(columns = 'index')
```

In [11]:
```
DF_cars
```

Out[11]:

	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Kilometer	RegistrationMonth	FuelType	Brand	NotRepaired	DateCr
0	480	NaN	1993	manual	0	golf	150000	0	petrol	volkswagen	NaN	2016-
1	18300	coupe	2011	manual	190	NaN	125000	5	gasoline	audi	yes	2016-
2	9800	suv	2004	auto	163	grand	125000	8	gasoline	jeep	NaN	2016-
3	1500	small	2001	manual	75	golf	150000	6	petrol	volkswagen	no	2016-
4	3600	small	2008	manual	69	fabia	90000	7	gasoline	skoda	no	2016-
...	...	...	...	...	...	...	...	...	...	...	...	
354259	0	NaN	2005	manual	0	colt	150000	7	petrol	mitsubishi	yes	2016-
354260	2200	NaN	2005	NaN	0	NaN	20000	1	NaN	sonstige_autos	NaN	2016-
354261	1199	convertible	2000	auto	101	fortwo	125000	3	petrol	smart	no	2016-
354262	9200	bus	1996	manual	102	transporter	150000	3	gasoline	volkswagen	no	2016-
354263	3400	wagon	2002	manual	100	golf	150000	6	gasoline	volkswagen	NaN	2016-

354264 rows × 14 columns

In [12]:
```
DF_cars[DF_cars['RegistrationYear']< 1900]['RegistrationYear'].count()
```

Out[12]:
66

In [13]:
```
# deletion of incorrect records - too old vehicles
DF_cars =DF_cars.drop(index = DF_cars[DF_cars['RegistrationYear']< 1900].index).reset_index().drop(columns = 'index')
```

In [14]:
```
DF_cars.describe()
```

Out[14]:

	Price	RegistrationYear	Power	Kilometer	RegistrationMonth	NumberOfPictures	PostalCode
count	354198.000000	354198.000000	354198.000000	354198.000000	354198.000000	354198.0	354198.000000
mean	4417.651314	2003.084789	110.078242	128267.607383	5.716819	0.0	50511.793813
std	4514.081022	7.536418	189.536766	37823.538557	3.725539	0.0	25783.464340
min	0.000000	1910.000000	0.000000	5000.000000	0.000000	0.0	1067.000000
25%	1050.000000	1999.000000	69.000000	125000.000000	3.000000	0.0	30165.000000
50%	2700.000000	2003.000000	105.000000	150000.000000	6.000000	0.0	49413.000000
75%	6400.000000	2008.000000	143.000000	150000.000000	9.000000	0.0	71083.000000
max	20000.000000	2019.000000	20000.000000	150000.000000	12.000000	0.0	99998.000000

In [15]:
```
list_of_nan = []
for i in DF_cars.columns:
    print(i,DF_cars[i].isna().sum(),int(round(DF_cars[i].isna().sum()/DF_cars[i].count()*100,0)),'%')
    if DF_cars[i].isna().sum() > 0:
        list_of_nan.append(i)

print('\n',list_of_nan)
```

```
Price 0 0 %
VehicleType 37319 12 %
RegistrationYear 0 0 %
Gearbox 19695 6 %
Power 0 0 %
Model 19630 6 %
Kilometer 0 0 %
RegistrationMonth 0 0 %
FuelType 32767 10 %
Brand 0 0 %
NotRepaired 71007 25 %
DateCreated 0 0 %
NumberOfPictures 0 0 %
PostalCode 0 0 %

 ['VehicleType', 'Gearbox', 'Model', 'FuelType', 'NotRepaired']
```

In [16]:
```
DF_cars.isna().mean() * 100
```

Out[16]:
```
Price                0.000000
VehicleType         10.536197
RegistrationYear     0.000000
Gearbox              5.560449
Power                0.000000
Model                5.542098
Kilometer            0.000000
RegistrationMonth    0.000000
FuelType             9.251040
Brand                0.000000
NotRepaired         20.047262
DateCreated          0.000000
NumberOfPictures     0.000000
PostalCode           0.000000
dtype: float64
```

In [17]:
```
# nulls elimination in numeric columns
DF_cars['VehicleType'] = DF_cars.groupby(['Brand', 'Model'])['VehicleType'].apply(lambda x: x.fillna(x.mode()[0] if not x.mode()
```

In [18]:
```
# nulls elimination in categorical columns
categorial_columns = ['VehicleType','Gearbox','Model','FuelType','Brand','NotRepaired','DateCreated']
categorial_features = DF_cars[categorial_columns]
categorial_features = categorial_features.fillna('unknown')
```

Categorical features encoding

In [19]:
```
encoder = OrdinalEncoder()
encoder.fit(categorial_features)
features_ordinal = encoder.transform(categorial_features)
features_ordinal = pd.DataFrame(features_ordinal, columns = categorial_features.columns)
```

In [20]:
```
features_ordinal
```

Out[20]:

	VehicleType	Gearbox	Model	FuelType	Brand	NotRepaired	DateCreated
0	4.0	1.0	116.0	6.0	38.0	1.0	94.0
1	7.0	1.0	228.0	2.0	1.0	2.0	94.0
2	6.0	0.0	117.0	2.0	14.0	1.0	84.0
3	5.0	1.0	116.0	6.0	38.0	0.0	87.0
4	5.0	1.0	101.0	2.0	31.0	0.0	101.0
...	...	...	...	...	...	...	...
354193	4.0	1.0	78.0	6.0	22.0	2.0	91.0
354194	7.0	2.0	228.0	7.0	33.0	1.0	84.0
354195	1.0	0.0	106.0	6.0	32.0	0.0	75.0
354196	0.0	1.0	224.0	2.0	38.0	0.0	89.0
354197	8.0	1.0	116.0	2.0	38.0	1.0	90.0

354198 rows × 7 columns

In [21]:
```
df_prepared = DF_cars.copy()
df_prepared[categorial_columns] = features_ordinal
```

In [22]:
```
df_prepared
```

Out[22]:

	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Kilometer	RegistrationMonth	FuelType	Brand	NotRepaired	DateCreated	Nun
0	480	4.0	1993	1.0	0	116.0	150000	0	6.0	38.0	1.0	94.0	
1	18300	7.0	2011	1.0	190	228.0	125000	5	2.0	1.0	2.0	94.0	
2	9800	6.0	2004	0.0	163	117.0	125000	8	2.0	14.0	1.0	84.0	
3	1500	5.0	2001	1.0	75	116.0	150000	6	6.0	38.0	0.0	87.0	
4	3600	5.0	2008	1.0	69	101.0	90000	7	2.0	31.0	0.0	101.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
354193	0	4.0	2005	1.0	0	78.0	150000	7	6.0	22.0	2.0	91.0	
354194	2200	7.0	2005	2.0	0	228.0	20000	1	7.0	33.0	1.0	84.0	
354195	1199	1.0	2000	0.0	101	106.0	125000	3	6.0	32.0	0.0	75.0	
354196	9200	0.0	1996	1.0	102	224.0	150000	3	2.0	38.0	0.0	89.0	
354197	3400	8.0	2002	1.0	100	116.0	150000	6	2.0	38.0	1.0	90.0	

354198 rows × 14 columns

Data split on train, valid and test samples

In [23]:
```
target = df_prepared['Price']
features = df_prepared.drop(columns='Price')
```

In [24]:
```
features_train, features_valid,target_train,target_valid = train_test_split(features,target, test_size=0.4,random_state=12345)
```

In [25]:
```
features_valid, features_test,target_valid,target_test = train_test_split(features_valid,target_valid, test_size=0.5,random_stat
```

Models training

Gradient boost model traning

In [26]:
```
param = {
    'task': 'train',
    'boosting': 'gbdt',
    'objective': 'regression',
    'num_leaves': 10,
    'verbose': -1,
    'metric' : 'rmse'}
```

In [27]:
```
train_dataset = lgb.Dataset(features_train, target_train, feature_name=features_train.columns.tolist())
test_dataset = lgb.Dataset(features_valid, target_valid, feature_name=features_valid.columns.tolist())
```

In [28]:
```
%%time
num_round = 25
bst = lgb.train(param, train_dataset, num_round,valid_sets= (test_dataset))
```

```
[1]     valid_0's rmse: 4244.71
[2]     valid_0's rmse: 4006.9
[3]     valid_0's rmse: 3808.11
[4]     valid_0's rmse: 3621.71
[5]     valid_0's rmse: 3462.04
[6]     valid_0's rmse: 3327.68
[7]     valid_0's rmse: 3208.24
[8]     valid_0's rmse: 3092.5
[9]     valid_0's rmse: 2992.35
[10]    valid_0's rmse: 2910.96
[11]    valid_0's rmse: 2833.7
[12]    valid_0's rmse: 2766.88
[13]    valid_0's rmse: 2710.2
[14]    valid_0's rmse: 2660.87
[15]    valid_0's rmse: 2610.16
[16]    valid_0's rmse: 2568.03
[17]    valid_0's rmse: 2527.5
[18]    valid_0's rmse: 2493.92
[19]    valid_0's rmse: 2462.27
[20]    valid_0's rmse: 2431.8
[21]    valid_0's rmse: 2405.78
[22]    valid_0's rmse: 2383.43
[23]    valid_0's rmse: 2362.7
[24]    valid_0's rmse: 2343.73
[25]    valid_0's rmse: 2326.53
CPU times: total: 2.81 s
Wall time: 469 ms
```

In [29]:
```
%%time
bst_pred = bst.predict(features_valid)
```

```
CPU times: total: 250 ms
Wall time: 45.9 ms
```

In [30]:
```
boost_rms = sqrt(mean_squared_error(target_valid, bst_pred))
round(boost_rms,0)
```

Out[30]:
```
2327.0
```

Linear regression model training

In [31]:
```
%%time
LR_model = LinearRegression()
LR_model.fit(features_train,target_train)
```

```
CPU times: total: 625 ms
Wall time: 181 ms
```

Out[31]:
```
LinearRegression()
```

In [32]:
```
%%time
LR_prediction = LR_model.predict(features_valid)
```

```
CPU times: total: 15.6 ms
Wall time: 11 ms
```

In [33]:
```
LR_rms = mean_squared_error(target_valid, LR_prediction,squared = False)
round(LR_rms,0)
```

Out[33]:
```
3466.0
```

Random forest model training

In [34]:
```
%%time
model_RF = RandomForestRegressor(n_estimators=100, max_depth=10,max_features=10, random_state=12345)
model_RF.fit(features_train,target_train)
```

```
CPU times: total: 41 s
Wall time: 41 s
```

Out[34]:
```
RandomForestRegressor(max_depth=10, max_features=10, random_state=12345)
```

In [35]:
```
RF_predictions = model_RF.predict(features_valid)
```

In [36]:
```
RF_rms = mean_squared_error(target_valid, RF_predictions,squared = False)
round(RF_rms,0)
```

Out[36]:
```
1995.0
```

Models comparison

In [37]:
```
%%time
LR_prediction_test = LR_model.predict(features_test)
```

```
CPU times: total: 15.6 ms
Wall time: 7.98 ms
```

In [38]:
```
LR_rms_test = mean_squared_error(target_test, LR_prediction_test,squared = False)
round(LR_rms_test,0)
```

Out[38]:
```
3465.0
```

Linear regression model scores:

- RMSE: 3465
- Training time: 181 milliseconds
- Prediction time: 7,98 milliseconds

*scores could be different after restart of code

In [39]:
```
%%time
bst_pred_test = bst.predict(features_test)
```

```
CPU times: total: 281 ms
Wall time: 38.9 ms
```

In [40]:
```
boost_rms = sqrt(mean_squared_error(target_test, bst_pred_test))
round(boost_rms,0)
```

Out[40]:
```
2321.0
```

Gradient boosting model scores:

- RMSE: 2321
- Training time: 45,9 seconds
- Prediction time: 38,9 milliseconds

*scores could be different after restart of code

In [41]:
```
%%time
RF_predictions = model_RF.predict(features_test)
```

```
CPU times: total: 1.23 s
Wall time: 737 ms
```

In [42]:
```
RF_rms_test = mean_squared_error(target_test, RF_predictions,squared = False)
round(RF_rms_test,0)
```

Out[42]:
```
1994.0
```

Random forest model scores:

- RMSE: 1994
- Training time: 41 seconds
- Prediction time: 737 milliseconds

*scores could be different after restart of code

General Conclusion

Based on the analysis the preferable model is random forest model, however the training time is not the best.

If it necessary to reduce the training time, it's possible to do using hyperparameters search, however RMSE score could be worse.

Scores of gradient boosting model also could be increased, but it will also increase the trining time.