Модель для определения рыночной стоимости автомобиля

Сервис по продаже автомобилей с пробегом «Не бит, не крашен» разрабатывает приложение для привлечения новых клиентов. В нём можно быстро узнать рыночную стоимость своего автомобиля. В вашем распоряжении исторические данные: технические характеристики, комплектации и цены автомобилей. Вам нужно построить модель для определения стоимости.

Заказчику важны:

- качество предсказания;
- скорость предсказания;
- время обучения.

Признаки:

DateCrawled — дата скачивания анкеты из базы

VehicleType — тип автомобильного кузова

RegistrationYear — год регистрации автомобиля

Gearbox — тип коробки передач

Power — мощность (л. с.)

Model — модель автомобиля

Kilometer — пробег (км)

RegistrationMonth — месяц регистрации автомобиля

FuelType — тип топлива

Brand — марка автомобиля

NotRepaired — была машина в ремонте или нет

DateCreated — дата создания анкеты

NumberOfPictures — количество фотографий автомобиля

PostalCode — почтовый индекс владельца анкеты (пользователя)

LastSeen — дата последней активности пользователя

Целевой признак:

Price — цена (евро)

1. Подготовка данных

In [1]:

```
import pandas as pd
import numpy as np
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split #импортируем функцию train_test_sp
lit из библиотеки sklearn

import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LinearRegression
from catboost import CatBoostRegressor
from sklearn.model_selection import (cross_val_score, train_test_split,
                                     GridSearchCV, RandomizedSearchCV)
from sklearn.metrics import r2_score

from lightgbm.sklearn import LGBMRegressor


from sklearn.metrics import mean_squared_error
import seaborn as sns
from sklearn.preprocessing import StandardScaler

from pylab import *
import matplotlib
import matplotlib.pyplot as plt
```

In [2]:

```
#прочитаем файл с исходными данными df

df = pd.read_csv("/datasets/autos.csv")
```

In [3]:

```
df.describe()
```

Out[3]:

	Price	RegistrationYear	Power	Kilometer	RegistrationMonth	Nu
count	354369.000000	354369.000000	354369.000000	354369.000000	354369.000000	
mean	4416.656776	2004.234448	110.094337	128211.172535	5.714645	
std	4514.158514	90.227958	189.850405	37905.341530	3.726421	
min	0.000000	1000.000000	0.000000	5000.000000	0.000000	
25%	1050.000000	1999.000000	69.000000	125000.000000	3.000000	
50%	2700.000000	2003.000000	105.000000	150000.000000	6.000000	
75%	6400.000000	2008.000000	143.000000	150000.000000	9.000000	
max	20000.000000	9999.000000	20000.000000	150000.000000	12.000000	

In [4]:

```
# удалим ненужные столбцы. в столбце с фото везде нули
df = df.drop(['DateCrawled', 'DateCreated', 'LastSeen', 'NumberOfPictures', 'PostalCod
e'], axis = 1)
```

In [5]:

```
df
```

Out[5]:

	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Kilometer	Registı
0	480	NaN	1993	manual	0	golf	150000	
1	18300	coupe	2011	manual	190	NaN	125000	
2	9800	suv	2004	auto	163	grand	125000	
3	1500	small	2001	manual	75	golf	150000	
4	3600	small	2008	manual	69	fabia	90000	
...	...	...	...	...	...	...	...	
354364	0	NaN	2005	manual	0	colt	150000	
354365	2200	NaN	2005	NaN	0	NaN	20000	
354366	1199	convertible	2000	auto	101	fortwo	125000	
354367	9200	bus	1996	manual	102	transporter	150000	
354368	3400	wagon	2002	manual	100	golf	150000	

354369 rows × 11 columns

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354369 entries, 0 to 354368
Data columns (total 11 columns):
Price                354369 non-null int64
VehicleType          316879 non-null object
RegistrationYear     354369 non-null int64
Gearbox              334536 non-null object
Power                354369 non-null int64
Model                334664 non-null object
Kilometer            354369 non-null int64
RegistrationMonth    354369 non-null int64
FuelType             321474 non-null object
Brand                354369 non-null object
NotRepaired          283215 non-null object
dtypes: int64(5), object(6)
memory usage: 29.7+ MB
```

In [7]:

```
#удалим дубликаты

df = df.drop_duplicates().reset_index(drop = True)
```

In [8]:

```
#проверим отсутствие дубликатов

df.duplicated().sum()
```

Out[8]:

```
0
```

In [9]:

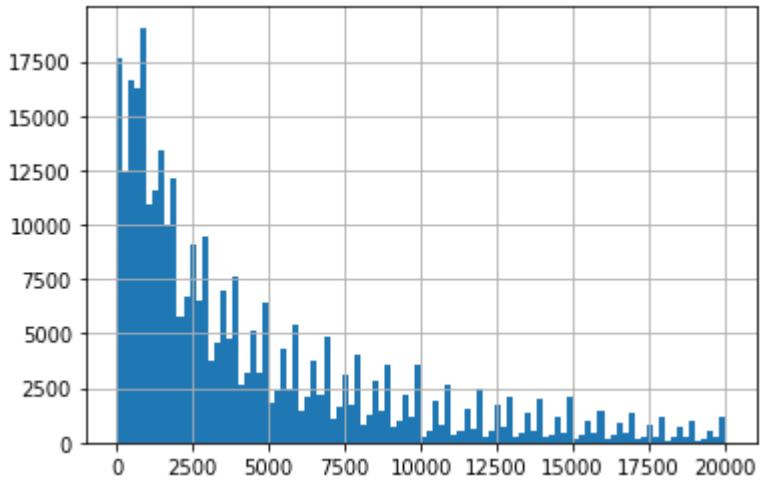
```
#красиво назовем столбцы

df.columns = ['price', 'vehicle_type', 'registration_year', 'gearbox', 'power', 'model'
, 'kilometer', 'registration_month', 'fuel_type', 'brand', 'not_repaired']
```

In [10]:

```
#построим гистограммы по некоторым столбцам
#изучим цену

df['price'].hist(bins=100, range=(0,20020))
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6f675e0090>
```



In [11]:

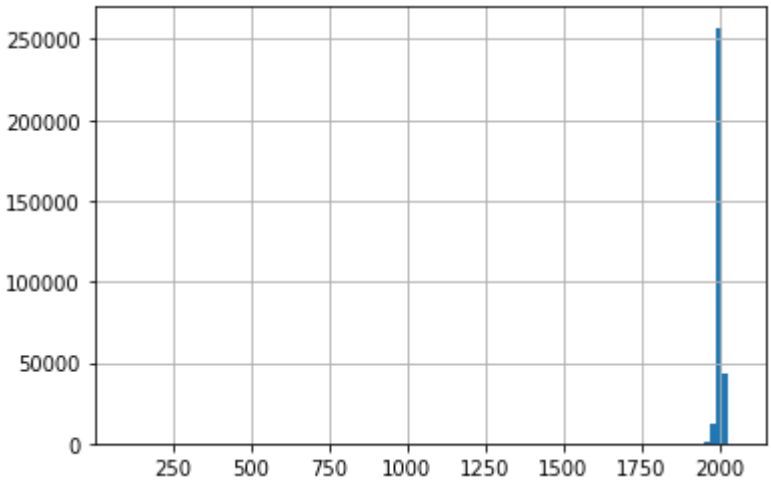
```
#отфильтруем таблицу только для авто дороже 100 евро, чтобы избавиться от авто без цены

df = df.query('price > 100')
```

In [12]:

```
df['registration_year'].hist(bins = 100, range = (100,2050))
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6f6937ecd0>
```



In [13]:

```
df['registration_year'].describe()
```

Out[13]:

```
count    314059.000000
mean       2003.973604
std          71.785991
min        1000.000000
25%        1999.000000
50%        2003.000000
75%        2008.000000
max        9999.000000
Name: registration_year, dtype: float64
```

In [14]:

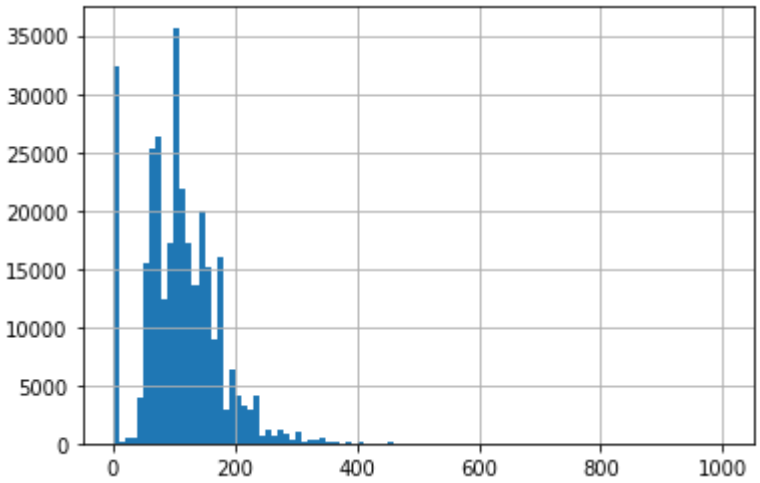
```
df = df.query('1900<registration_year <= 2020')
```

In [15]:

```
df['power'].hist(bins = 100, range = (0,1000))
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6f67e63810>
```



In [16]:

```
#отфильтруем таблицу только для авто с мощностью больше 50, чтобы избавиться от авто бе
з указания лс

df = df.query('power > 50')
```

In [17]:

```
#выгрузим названия столбцов

columns = df.columns.tolist()
```

In [18]:

```
columns
```

Out[18]:

```
['price',
 'vehicle_type',
 'registration_year',
 'gearbox',
 'power',
 'model',
 'kilometer',
 'registration_month',
 'fuel_type',
 'brand',
 'not_repaired']
```

In [19]:

```
#посмотрим какое количество пустых строк в столбцах таблицы

for i in columns:
    print("Количество пустых строк в столбце", i , df[i].isna().sum(), '{:.2%}'.format(
df[i].isna().sum() / df.shape[0]))
    print("")
```

```
Количество пустых строк в столбце price 0 0.00%

Количество пустых строк в столбце vehicle_type 19187 7.04%

Количество пустых строк в столбце registration_year 0 0.00%

Количество пустых строк в столбце gearbox 5163 1.89%

Количество пустых строк в столбце power 0 0.00%

Количество пустых строк в столбце model 11033 4.05%

Количество пустых строк в столбце kilometer 0 0.00%

Количество пустых строк в столбце registration_month 0 0.00%

Количество пустых строк в столбце fuel_type 17780 6.52%

Количество пустых строк в столбце brand 0 0.00%

Количество пустых строк в столбце not_repaired 41779 15.33%
```

In [20]:

```
#заполним столбцы с нан на other или удалим
df[['vehicle_type','gearbox', 'model', 'fuel_type','not_repaired']]=df[['vehicle_type',
'gearbox', 'model', 'fuel_type','not_repaired']].fillna('other')
```

In [21]:

```
#для всех категорийных признаков поменяем тип

categ = ['vehicle_type', 'gearbox', 'model', 'registration_month', 'fuel_type', 'brand'
, 'not_repaired']

for i in categ:
    df[i]=df[i].astype('category')
```

In [22]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 272541 entries, 1 to 326825
Data columns (total 11 columns):
price                 272541 non-null int64
vehicle_type          272541 non-null category
registration_year     272541 non-null int64
gearbox               272541 non-null category
power                 272541 non-null int64
model                 272541 non-null category
kilometer             272541 non-null int64
registration_month    272541 non-null category
fuel_type             272541 non-null category
brand                 272541 non-null category
not_repaired          272541 non-null category
dtypes: category(7), int64(4)
memory usage: 12.5 MB
```

In [23]:

```
#Преобразовать категориальные признаки в численные поможет техника прямого кодирования,
или отображения (англ. One-Hot Encoding, OHE).
df_ohe=pd.get_dummies(df, drop_first=True)
df_ohe
```

Out[23]:

	price	registration_year	power	kilometer	vehicle_type_convertible	vehicle_type_cou
1	18300	2011	190	125000	0	
2	9800	2004	163	125000	0	
3	1500	2001	75	150000	0	
4	3600	2008	69	90000	0	
5	650	1995	102	150000	0	
...	...	...	...	...	...	
326819	5250	2016	150	150000	0	
326820	3200	2004	225	150000	0	
326823	1199	2000	101	125000	1	
326824	9200	1996	102	150000	0	
326825	3400	2002	100	150000	0	

272541 rows × 319 columns

In [24]:

```
#определим признаки и целевой признак
features_ohe=df_ohe.drop(['price'], axis=1)
target_ohe=df_ohe['price']
```

In [25]:

```
# разделим выборки, на тестовую и обучающую выборку признаков и целев признака
features_train_ohe, features_test_ohe, target_train_ohe, target_test_ohe = train_test_s
plit(features_ohe, target_ohe, test_size=0.25, random_state=12345)
```

In [26]:

```
#Произведем масштабирование
numeric = ['registration_year', 'power', 'kilometer']

scaler_ohe = StandardScaler()
scaler_ohe.fit(features_train_ohe[numeric])
features_train_ohe[numeric] = scaler_ohe.transform(features_train_ohe[numeric])
features_test_ohe[numeric] = scaler_ohe.transform(features_test_ohe[numeric])
```

In [27]:

```
#Для более эффективной работы всех остальных моделей будем кодировать категориальные пр
изнаки с помощью порядкового кодирования

from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
df[categ] = pd.DataFrame(encoder.fit_transform(df[categ]),
                         columns=df[categ].columns)
```

In [28]:

```
#определим признаки и целевой признак
features_or=df.drop(['price'], axis=1)
target_or=df['price']
```

In [29]:

```
# разделим выборки, на тестовую и обучающую выборку признаков и целев признака
features_train_or, features_test_or, target_train_or, target_test_or = train_test_split
(features_or, target_or, test_size=0.25, random_state=12345)
```

In [30]:

```
#Произведем масштабирование

scaler_or = StandardScaler()
scaler_or.fit(features_train_or[numeric])
features_train_or[numeric] = scaler_or.transform(features_train_or[numeric])
features_test_or[numeric] = scaler_or.transform(features_test_or[numeric])
```

Вывод:

Данные выгружены, прошли предобработку. Добалили необходимые библиотеки, изучили общую информацию по датафрейму, провели масштабирование числовых значений, разделили на выборки, присвоили значения признакам.

OHE используется только для линейных моделей (включая линейную регрессию). Для более эффективной работы всех остальных моделей необхоимо кодировать категориальные признаки с помощью порядкового кодирования. С помощью порядкового кодирования необходимо кодировать не только категориальные признаки, а абсолютно все, включая численные. Кодируем только нужные нам признаки, иначе мы меняем распределение численных признаков. По поводу масштабирования: обучаем сначала Scaler на данных, закодированных каким-либо одним образом, и трансформируем все данные, закодированные этим образом. И так дважды, не смешиваем.

2. Обучение моделей

In [31]:

```
#обучим модель линейной регрессии

model_LR = LinearRegression().fit(features_train_ohe,target_train_ohe)
```

In [32]:

```
model = CatBoostRegressor()

grid = {'learning_rate': [0.03, 0.1],
        'depth': [4, 10],
        'l2_leaf_reg': [1, 5]}

grid_search_result = model.grid_search(grid,
                                       X=features_train_or,
                                       y=target_train_or,
                                       plot=True)
```

```
0:      loss: 2120.1984029      best: 2120.1984029 (0)  total: 1m 43s   re
maining: 12m 5s
1:      loss: 2082.1630250      best: 2082.1630250 (1)  total: 3m 24s   re
maining: 10m 14s
2:      loss: 2121.1479056      best: 2082.1630250 (1)  total: 5m 6s    re
maining: 8m 30s
3:      loss: 2082.7970437      best: 2082.1630250 (1)  total: 6m 47s   re
maining: 6m 47s
4:      loss: 2080.7246583      best: 2080.7246583 (4)  total: 10m 35s  re
maining: 6m 21s
5:      loss: 2076.7399498      best: 2076.7399498 (5)  total: 14m 24s  re
maining: 4m 48s
6:      loss: 2083.2294253      best: 2076.7399498 (5)  total: 18m 13s  re
maining: 2m 36s
7:      loss: 2077.9334032      best: 2076.7399498 (5)  total: 22m 2s   re
maining: 0us
Estimating final quality...
```

In [33]:

```
model_CB = CatBoostRegressor(loss_function="RMSE", depth=10, learning_rate=0.1, l2_leaf
_reg=1)
model_CB.fit(features_train_or, target_train_or,  verbose=False)
```

Out[33]:

```
<catboost.core.CatBoostRegressor at 0x7f6f6ac47d10>
```

In [34]:

```
from sklearn.metrics import SCORERS
#print(SCORERS.keys())
```

In [35]:

```
hyper_space = {'n_estimators': [1000,  2500],
               'max_depth':  [4, 9],
               'learning_rate': [0.1, 0.3]}
               #'num_leaves': [15,  63, 127]
               #'subsample': [0.6, 0.7, 0.8, 1.0],
               #'colsample_bytree': [0.6, 0.7, 0.8, 1.0]}

est = LGBMRegressor(boosting_type='gbdt', n_jobs=-1, random_state=12345)

gs = GridSearchCV(est, hyper_space, scoring='neg_mean_squared_error', cv=2, verbose=1)
gs_results = gs.fit(features_train_or, target_train_or)
print("BEST PARAMETERS: " + str(gs_results.best_params_))
```

```
Fitting 2 folds for each of 8 candidates, totalling 16 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed: 18.9min finished

BEST PARAMETERS: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 10
00}
```

In [36]:

```
model_LGBM = LGBMRegressor(boosting_type = 'gbdt', n_jobs = -1, depth = 9, learning_rat
e = 0.1, n_estimators = 2500)
model_LGBM.fit(features_train_or, target_train_or)
```

Out[36]:

```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.
0,
              depth=9, importance_type='split', learning_rate=0.1, max_dep
th=-1,
              min_child_samples=20, min_child_weight=0.001, min_split_gain
=0.0,
              n_estimators=2500, n_jobs=-1, num_leaves=31, objective=None,
              random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=Tru
e,
              subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

3. Анализ моделей

In [37]:

```
%%time

predict_LR = model_LR.predict(features_test_ohe)
rmse_LR = mean_squared_error(target_test_ohe, predict_LR)**0.5
rmse_LR
```

```
CPU times: user 101 ms, sys: 70 ms, total: 171 ms
Wall time: 175 ms
```

Out[37]:

```
2773.3745383999603
```

In [38]:

```
%%time

predict_CB = model_CB.predict(features_test_or)
rmse_CB = mean_squared_error(target_test_or, predict_CB)**0.5
rmse_CB
```

```
CPU times: user 589 ms, sys: 20.3 ms, total: 609 ms
Wall time: 608 ms
```

Out[38]:

```
2084.6486073286314
```

In [39]:

```
%%time

predict_LGBM = model_LGBM.predict(features_test_or)
rmse_LGBM = mean_squared_error(target_test_or, predict_LGBM)**0.5
rmse_LGBM
```

```
CPU times: user 18.9 s, sys: 0 ns, total: 18.9 s
Wall time: 18.8 s
```

Out[39]:

```
2104.159994035155
```

Вывод:

Быстрее всего работает линейная регрессия, затем по скорости работы идет catboost и LGBM самая медленная. Качество модели было оценено по показателю RMSE. Лучше всего себя показала модель LGBM без сильного отрува от catboost. И безнадежно отстала модель линейной регрессии. Я бы выбрала CatBoost.