Определение стоимости автомобилей

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)

Подготовка данных

B [3]:
```
import pandas as pd
import numpy as np

import urllib
import os

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder

from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error as MSE
RMSE = lambda x, y: MSE(x, y) ** .5

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
```

B [2]:
```
name, url = 'datasets/autos.csv', 'https://code.s3.yandex.net/datasets/autos.csv'
os.makedirs(name.split('/')[0], exist_ok=True)
if not os.path.exists(name):
    _ = urllib.request.urlretrieve(url, name)
```

B [4]:
```
data = pd.read_csv(name)
data.head(3)
```

Out[4]:

	DateCrawled	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Kilometer	Registrat
0	2016-03-24 11:52:17	480	NaN	1993	manual	0	golf	150000	
1	2016-03-24 10:58:45	18300	coupe	2011	manual	190	NaN	125000	
2	2016-03-14 12:52:21	9800	suv	2004	auto	163	grand	125000	

B [4]:
```
data.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
```

Имеют место пропуски в данных, выделим признаки с пропусками

B [5]:
```
data.isna().sum().apply(lambda x: x if x > 0 else None).dropna() / data.shape[0]
```

Out[5]:
```
VehicleType    10.579368
Gearbox         5.596709
Model           5.560588
FuelType        9.282697
NotRepaired    20.079070
dtype: float64
```

B [6]:
```
def mode_upd(series):
    out = series.mode()
    try:
        return out[0]
    except:
        return np.nan
```

B [7]:
```
data.loc[data.Model.isna(), 'Model'] =\
    data.loc[data.Model.isna(), 'Brand'].map(
        data.groupby('Brand').agg(lambda x: mode_upd(x)).Model)
```

B [8]:
```
for col in ['VehicleType', 'Gearbox', 'FuelType', 'NotRepaired']:
    data.loc[data[col].isna(), col] =\
        data.loc[data[col].isna(), 'Model'].map(
            data.groupby('Model').agg(lambda x: mode_upd(x))[col])
```

B [9]:
```
data.isna().sum().apply(lambda x: x if x > 0 else None).dropna() / data.shape[0]
```

Out[9]:
```
VehicleType    0.293762
Gearbox        0.311257
Model          0.952115
FuelType       0.311540
NotRepaired    0.373904
dtype: float64
```

Пропуски в признаках менее 1% можем удалить

B [10]:
```
data = data.dropna()
```

B [11]:
```
%%time
columns_time = ['DateCrawled', 'DateCreated', 'LastSeen']
for col in columns_time:
    data.loc[:, col] = pd.to_datetime(data.loc[:, col])
    base_time = data.loc[:, col].min()
    data.loc[:, col] = data.loc[:, col].apply(lambda x: (x - base_time).days)
```

```
Wall time: 37.3 s
```

B [12]:
```
data[columns_time].describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
DateCrawled	350993.0	15.521418	9.094440	0.0	7.0	16.0	24.0	33.0
DateCreated	350993.0	741.798307	9.387058	0.0	734.0	742.0	750.0	759.0
LastSeen	350993.0	23.887966	9.175613	0.0	17.0	29.0	31.0	33.0

B [13]:
```
data.loc[:, 'DateCrawled'] = data.DateCrawled.astype(np.uint8)
data.loc[:, 'DateCreated'] = data.DateCreated.astype(np.uint16)
data.loc[:, 'LastSeen'] = data.LastSeen.astype(np.uint8)
```

Обработка аномалий

B [14]:
```
data.describe().T
```

Out[14]:

	count	mean	std	min	25%	50%	75%	
DateCrawled	350993.0	15.521418	9.094440	0.0	7.0	16.0	24.0	
Price	350993.0	4410.813102	4503.352517	0.0	1099.0	2700.0	6399.0	2
RegistrationYear	350993.0	2004.067209	79.123213	1000.0	1999.0	2003.0	2008.0	
Power	350993.0	110.268213	189.080808	0.0	69.0	105.0	142.0	2
Kilometer	350993.0	128574.672429	37470.142690	5000.0	125000.0	150000.0	150000.0	15
RegistrationMonth	350993.0	5.728274	3.721769	0.0	3.0	6.0	9.0	
DateCreated	350993.0	741.798307	9.387058	0.0	734.0	742.0	750.0	
NumberOfPictures	350993.0	0.000000	0.000000	0.0	0.0	0.0	0.0	
PostalCode	350993.0	50538.698604	25763.657663	1067.0	30169.0	49429.0	71088.0	9
LastSeen	350993.0	23.887966	9.175613	0.0	17.0	29.0	31.0	

B [15]:
```
data = data[~(data.Price == 0)]
```

B [16]:
```
data = data.drop(['NumberOfPictures', 'PostalCode'], axis=1)
```

B [17]:
```
data[data.RegistrationYear > 2008].groupby('RegistrationYear')['RegistrationYear'
```

Out[17]:
```
RegistrationYear
5000    15
5555     2
5600     1
5900     1
5911     2
6000     3
6500     1
7000     3
7100     1
7800     1
8000     2
8200     1
8500     1
9000     1
9999    16
Name: RegistrationYear, dtype: int64
```

B [18]:
```
data = data[~(data.RegistrationYear > 2019)]
```

B [19]:
```
data = data.query('30 <= Power <= 1000')
```

B [20]:
```
columns_obj = [col for col in data.columns if data[col].dtype == np.object]
columns_obj
```

Out[20]:
```
['VehicleType', 'Gearbox', 'Model', 'FuelType', 'Brand', 'NotRepaired']
```

Применять к `Model` и `Brand` `OHE` неразумено, так как появится очень много столбцов, попробуем на них использовать `Ordinal Encoding`

B [21]:
```
columns_obj = list(set(columns_obj) - set(['Model', 'Brand']))
for col in columns_obj:
    data = pd.concat([data, pd.get_dummies(data[col], prefix=col+'_dummy_', dummy
        drop([col], axis=1)
```

B [22]:
```
oe = OrdinalEncoder()
oe.fit(data[['Model', 'Brand']])
data.loc[:, ['Model', 'Brand']] = oe.transform(data[['Model', 'Brand']])
```

Вывод

Заполнили модами пропуски в данных, после чистки удалили пропуски (процентное соотношение которых было меньше 1%). Преобразовали признаки, содержащие временные значения в количество дней от объекта с минимальным значением. Для признаков с небольшим количеством дискретных значений использовали `OHE` . Удалили `NumberOfPictures` , так как признак не содержит никакой предсказательной информации.

Обучение моделей

`features - X`
`target - y`

B [26]:
```
X_train, X_test, y_train, y_test =\
    train_test_split(data.drop(['Price'], axis=1), data.Price, random_state=1, te
```

B [27]:
```
get_percentage = lambda x: int(x.shape[0] / data.shape[0] * 100)
list(map(get_percentage, [X_train, X_test, y_train, y_test]))
```

Out[27]:
```
[74, 25, 74, 25]
```

B [29]:
```
features_cols_num = [col for col in X_train.columns\
                     if X_train[col].dtype != np.object and\
                        '_dummy' not in X_train[col].name]
features_cols_num
```

Out[29]:
```
['DateCrawled',
 'RegistrationYear',
 'Power',
 'Model',
 'Kilometer',
 'RegistrationMonth',
 'Brand',
 'DateCreated',
 'LastSeen']
```

B [31]:
```
scaler = StandardScaler()
scaler.fit(X_train[features_cols_num])
```

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

B [32]:
```
%%time
X_train.loc[:, features_cols_num] = scaler.transform(X_train[features_cols_num])
X_test.loc[:, features_cols_num] = scaler.transform(X_test[features_cols_num])
```

```
Wall time: 1.06 s
```

B [33]:
```
models = {}
```

B [34]:
```
%%time
model_lin = LinearRegression()
model_lin.fit(X_train, y_train)
models['linear'] = {}
```

```
Wall time: 764 ms
```

B [35]:
```
%%time
params = {
    'max_depth': [10, 25, 50, 75, 100]
}

model_tree = DecisionTreeRegressor(random_state=1)
gs_dt = GridSearchCV(estimator=model_tree, param_grid=params,\
                        scoring=make_scorer(RMSE), n_jobs=-1, cv=5)

gs_dt = gs_dt.fit(X_train, y_train)
print(gs_dt.best_params_, gs_dt.best_score_)
model_tree = gs_dt.best_estimator_
models['decision_tree'] = {}
```

```
{'max_depth': 75} 2198.1198683570033
Wall time: 41.3 s
```

B [36]:
```
%%time
params = {
    'max_depth': [10, 50],
    'n_estimators': [10, 50]
}

model_forest = RandomForestRegressor(random_state=1)
gs_rf = GridSearchCV(estimator=model_forest, param_grid=params,\
                        scoring=make_scorer(RMSE), n_jobs=-1, cv=3)

gs_rf = gs_rf.fit(X_train, y_train)
print(gs_rf.best_params_, gs_rf.best_score_)
model_forest = gs_rf.best_estimator_
models['random_forest'] = {}
```

```
{'max_depth': 10, 'n_estimators': 10} 1902.3550705300192
Wall time: 3min 49s
```

B [37]:
```
%%time
params = {
    'n_estimators': [50, 100, 150]
}

model_lgbm = LGBMRegressor()
gs_lgbm = GridSearchCV(estimator=model_lgbm, param_grid=params,\
                        scoring=make_scorer(RMSE), n_jobs=-1, cv=5)

gs_lgbm = gs_lgbm.fit(X_train, y_train)
print(gs_lgbm.best_params_, gs_lgbm.best_score_)
model_lgbm = gs_lgbm.best_estimator_
models['LGBM'] = {}
```

```
{'n_estimators': 50} 1776.2438545917455
Wall time: 22.8 s
```

B [38]:
```
%%time
model_cat = CatBoostRegressor(verbose=False)
model_cat.fit(X_train, y_train)
models['cat_boost_model'] = {}
```

```
Wall time: 43.3 s
```

B [48]:
```
models['linear']['fit_time'] = '700 ms'
models['decision_tree']['fit_time'] = '40 s'
models['random_forest']['fit_time'] = '3min 40s'
models['LGBM']['fit_time'] = '20 s'
models['cat_boost_model']['fit_time'] = '40 s'
```

Анализ моделей

B [40]:
```
%%time
pred = model_lin.predict(X_test)
models['linear']['RMSE'] = int(RMSE(y_test, pred))
```

```
Wall time: 98.8 ms
```

B [41]:
```
%%time
pred = model_tree.predict(X_test)
models['decision_tree']['RMSE'] = int(RMSE(y_test, pred))
```

```
Wall time: 207 ms
```

B [42]:
```
%%time
pred = model_forest.predict(X_test)
models['random_forest']['RMSE'] = int(RMSE(y_test, pred))
```

```
Wall time: 259 ms
```

B [43]:
```
%%time
pred = model_lgbm.predict(X_test)
models['LGBM']['RMSE'] = int(RMSE(y_test, pred))
```

```
Wall time: 256 ms
```

B [44]:
```
%%time
pred = model_cat.predict(X_test)
models['cat_boost_model']['RMSE'] = int(RMSE(y_test, pred))
```

```
Wall time: 110 ms
```

B [49]:
```
models['linear']['predict_time'] = '100 ms'
models['decision_tree']['predict_time'] = '200 ms'
models['random_forest']['predict_time'] = '260 ms'
models['LGBM']['predict_time'] = '260 ms'
models['cat_boost_model']['predict_time'] = '110 ms'
```

B [50]:
```
pd.DataFrame(models).T
```

Out[50]:

	fit_time	RMSE	predict_time
linear	700 ms	3010	100 ms
decision_tree	40 s	2153	200 ms
random_forest	3min 40s	1878	260 ms
LGBM	20 s	1762	260 ms
cat_boost_model	40 s	1555	110 ms

Итоговый вывод

Исходя из таблице выше и учитывая то, что заказчику важны:

- качество предсказания
    1. catboost

    2. LGBM
    3. randomforest
- скорость предсказания;
    1. Linreg
    2. catboost
    3. dectree
- время обучения.
    1. Linreg
    2. LGBM
    3. catboost

Лучший выбор из представленных моделей - `CatBoostRegressor` .