Определение стоимости автомобилей (учебный проект)

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 нашем распоряжении исторические данные: технические характеристики, комплектации и цены автомобилей. Необходимо построить модель для определения стоимости.

Заказчику важны:

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

Для решении задачи заказчик предоставил датасет:

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)

План работы

- Изучение и подготовка данных
- Обучение моделей
- Анализ моделей
- Тестирование лучшей модели

Подготовка данных

In [89]:

```
import pandas as pd
import numpy as np
import random
import seaborn as sea
import time

from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, GridSearc

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor
```

In [90]:

```
data_original = pd.read_csv('/datasets/autos.csv')
data = data_original
```

In [91]:

```
data.columns = ['date_crawled', 'price','vehicle_type','registration_year', 'gea
```

In [92]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354369 entries, 0 to 354368
Data columns (total 16 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   date_crawled       354369 non-null  object
 1   price              354369 non-null  int64 
 2   vehicle_type       316879 non-null  object
 3   registration_year  354369 non-null  int64 
 4   gearbox            334536 non-null  object
 5   power              354369 non-null  int64 
 6   model              334664 non-null  object
 7   km                 354369 non-null  int64 
 8   reg_month          354369 non-null  int64 
 9   fuel_type          321474 non-null  object
 10  brand              354369 non-null  object
 11  not_repaired       283215 non-null  object
 12  date_created       354369 non-null  object
 13  pictures           354369 non-null  int64 
 14  postal_code        354369 non-null  int64 
 15  last_seen          354369 non-null  object
dtypes: int64(7), object(9)
memory usage: 43.3+ MB
```

price

У 10772 цена 0, чаще встречается цена от 500 до сих 6 тысяч
Строки, где нет цены, удалю, так как это целевой признак
Есть цены 1 евро и др <10 евро. Или они ошибочны, или единица измерения - тысяча евро? Хотелось бы эту информацию уточнить у заказчика

In [93]:

```
data = data.drop(index = data[data.price==0].index, axis=0)
data.price.isna().sum()
```

Out[93]:

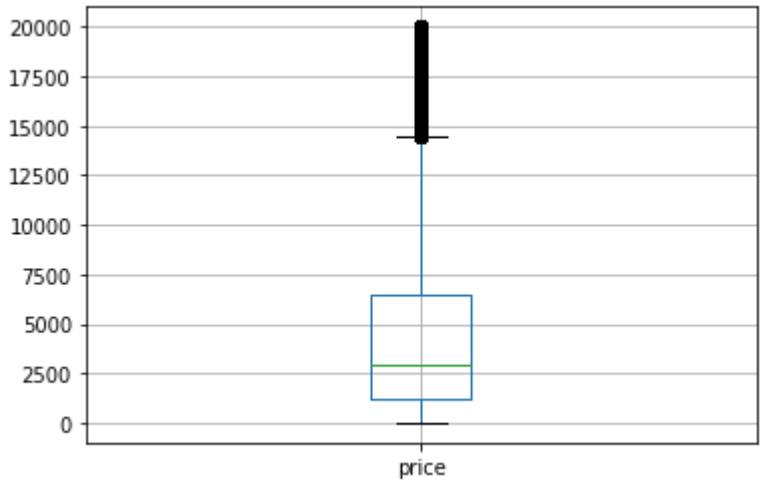
0

In [94]:

```
data[['price']].boxplot()
```

Out[94]:

<AxesSubplot:>



vehicle_type

Тип кузова: нет данных - 37490, other - 3288. Я предполагала, что для ряда моделей можно взять данные из аналогичных моделей с заполненным типом кузова, но у одной и той же модели может быть разный тип кузова и этот вариант не подошел. Поэтому выделим их отдельной группой

Выделяю пустые значения в отедльную группу "нет данных"

In [95]:

```
data.loc[data.vehicle_type.isna(),'vehicle_type']='no_data'
```

Отделю тестовую выборку

In [96]:

```
data.loc[data['price']<=1200, 'price_class'] = 0
data.loc[(data['price']>1200)&(data['price']<=2890), 'price_class'] = 1
data.loc[(data['price']>2890)&(data['price']<=6500), 'price_class'] = 2
data.loc[(data['price']>6500)&(data['price']<=20000), 'price_class'] = 3

train, test = train_test_split(
    data,
    test_size=0.25,
    random_state=12345,
    stratify=data['price_class']
)
print(train.shape, test.shape)

```

```
(257697, 17) (85900, 17)
```

registration_year

Есть неадекватные года - больше даты скачивания и меньше 1768. [статья на вики (https://translated.turbopages.org/proxy_u/en-ru.ru.a2922144-63446787-7d62ef71-74722d776562/https/en.wikipedia.org/wiki/Vehicle_registration_plates_of_the_Soviet_Union#1931_to_1946)](). Думаю, началом учета могу условно считать 1931 год. Стандартизация учета произошла в СССР в 1975 году и в России в 1993 году, но как я пониманию и ранее учет был. При этом не исключаю, что в этом поле указывают год производства по техпаспорту. Автомобили начали производить в 1768 году. По этому вопросу я бы уточнила информацию у заказчика. Также даннные можно проверить по дате выпуска модели. так у форда мондео стоит год регистрации 1000, хотя модель произведена в 1993. Всего таких 166 строк. 82 строкb в периоде с 1768 по 1933. Их хочу проверить, действительно ли продаются древние модели

1. Моделям с некорректным годом регистрации установлю год рандомно из диапазона по годам выпуска модели (меньше 1768 и больше текущего). Предполагаю, что возможно база выгружалась в 2016-м году, но возможно обновлялась после выгрузки? Так как не могу это проверить, использую текущий год.
2. Проверив вручную дату выхода моделей, у которых в данных стоит год до 1933 сделала вывод, что все эти модели вышли позже, а значит данные ошибочны. Заполню их также рандомными значениями по годам выпуска
3. После выполнения первых двух пунктов осталась 41 строка с некорректным годом. Часть из них - sonstige_autos (прочие бренды в переводе с немецкого) и еще несколько строк без указания модели. Эти строки удалю, так как данные по наименованию модели определяющие для формирования цены.

In [97]:

```
1 data[(data.registration_year<=1933)|(data.registration_year>2022)][['brand','mod
```

Out[97]:

brand	model	
volkswagen	golf	17
opel	corsa	7
ford	other	6
bmw	3er	6
	other	5
opel	other	4
volkswagen	polo	4
opel	zafira	4
volkswagen	kaefer	4
fiat	other	4
mercedes_benz	other	3
peugeot	other	3
ford	mondeo	3
alfa_romeo	156	2
mercedes_benz	a_klasse	2
porsche	911	2
ford	ka	2

In [98]:

```
#данные выхода моделей брала из википедии
years = [['volkswagen', 'golf', 1974, 2022],
         ['alfa_romeo', '156', 1997, 2007],
         ['opel', 'corsa', 1982, 2022],
         ['bmw','3er',1990,1998],
         ['opel','zafira',1999,2022],
         ['volkswagen','kaefer',1946,2003],
         ['volkswagen','polo',1975, 2022],
         ['mercedes_benz','a_klasse', 1997, 2022],
         ['bmw','5er', 2003, 2010],
         ['volkswagen','passat', 1973, 2022],
         ['subaru','impreza', 1992, 2022],
         ['renault','twingo', 1993, 2022],
         ['renault','laguna', 1994, 2015],
         ['renault','clio', 1990, 2022],
         ['porsche','911', 1963, 2022],
         ['skoda','octavia', 1996, 2022],
         ['bmw','1er', 2004, 2022],
         #['opel','kadett', 1937-1940, 1962-1993]
         ['mitsubishi','colt', 1962, 2012],
         ['mercedes_benz','clk', 1996,2009],
         ['mazda','6_reihe', 2002, 2022],
         ['ford','fiesta',1976, 2022],
         ['ford','escort', 1968, 2003],
         ['fiat','panda', 1980, 2022],
         ['citroen','c4',2004, 2022],
         ['bmw','x_reihe',1999, 2022],
         ['volkswagen','transporter', 1949, 2022],
         ['ford','ka', 1996, 2016],
         ['ford', 'mondeo', 1992, 2022],
         ['citroen','c3',2002, 2022],
         ['volkswagen','caddy',1980, 2022],
         ['volkswagen','beetle',1998, 2010],
         ['renault','espace',1984, 2022],
         ['audi','a2',1999, 2005],
         ['opel','agila',2000, 2015],
         ['opel','calibra',1989, 1997],
         ['fiat','punto',1993, 2018]]

def registration(data):
    #отдельно для опеля, так как был перерыв в годах выпуска
    data.loc[(data.registration_year<=1933)|(data.registration_year>2022)
        &(data.brand=='opel')
        &(data.model=='kadett'),
            'registration_year']=random.choice(list(range(1937, 1940+1))+list(ra

    for one in years:
        data.loc[(data.registration_year<=1933)|(data.registration_year>2022)
                 &(data.brand==one[0])
                 &(data.model==one[1]),
                 'registration_year']=random.randint(one[2],one[3]+1)
    data = data.drop(index = data[(data.registration_year<=1933)|(data.registrat
    print(data[(data.registration_year<=1933)|(data.registration_year>2022)])
    return data

test = registration(test)
train = registration(train)

```

```
59
Empty DataFrame
Columns: [date_crawled, price, vehicle_type, registration_year, gearbo
x, power, model, km, reg_month, fuel_type, brand, not_repaired, date_c
reated, pictures, postal_code, last_seen, price_class]
Index: []
Empty DataFrame
Columns: [date_crawled, price, vehicle_type, registration_year, gearbo
x, power, model, km, reg_month, fuel_type, brand, not_repaired, date_c
reated, pictures, postal_code, last_seen, price_class]
Index: []
```

gearbox

Автомат или ручная. 19833 не заполнены. Модели, выпущенные до 1940 года - МКПП. ссылка на вики (https://ru.wikipedia.org/wiki/Автоматическая_коробка_передач#История)
Для моделей до 1940 года заполняю manual. Остальные пропуски объединяю в отдельную группу

In [99]:

```
def gearbox(data):
    data.loc[data.registration_year<=1940, 'gearbox'] = 'manual'
    data.loc[data.gearbox.isna(),'gearbox'] = 'no_data'
    return data
train = gearbox(train)
test = gearbox(test)
```

power

40225 нулей. Самая слабая машина на рынке имеет мощность 5 и 17 лс, однако в данных есть 83 машины с мощностью меньше 5. Я погуглила нескольно моделей - данные некорректны. Данные меньше 5 удаляю.
Все пустые данные заполню -999
Для моделей с мощностью более 1600 установлю медианное значение мощности.

In [100]:

```
def power(data):
    data.power = data.power.drop(index = data[(data.power<5)&(data.power>0)].inc
    data.loc[data.power.isna(),'power'] = -999
    data.loc[data.power>1600,'power'] = data.power.median()
    return data
test = power(test)
train = power(train)


```

Данные по самым мощным автомобилям по годам выпуска взяты из таблицы хронологии (https://translated.turbopages.org/proxy_u/en-ru.ru.05545da8-63465441-a8a9547e-74722d776562/https/en.wikipedia.org/wiki/Highest_horsepower_engine). Для тех моделей, где мощность больше, чем выпускаемая в этот год самая мощная модель, устанавливаю максимально возможное для этого года значение

In [101]:

```
horses = [[1894,1.5],
[1897,8],
[1899,23],
[1901,35],
[1902,45],
[1903,60],
[1907,75],
[1908,91],
[1910,121],
[1912,200],
[1928,269],
[1932,324],
[1935,406],
[1958,405],
[1963,431],
[1965,492],
[1991,560],
[1992,627],
[1995,680],
[2004,817],
[2005,1001],
[2009,1305],
[2014,1360],
[2015,1521],
[2021,1600]
]
horses = pd.DataFrame(horses, columns = ['year','power']).sort_values('power',as


```

Данные для моделей с незаполненной мощностью заполню медианой по бренду

In [102]:

```
def horses_f(data):
    for one in range(len(horses)):
        data.loc[(data.registration_year == horses.loc[one,'year'])&
                 (data.power == horses.loc[one,'power']),'power'] = horses['powe

    for one in data.brand.unique():
        data.loc[(data.power==0)&(data.brand ==one), 'power'] = data[data.brand=
    return data
train = horses_f(train)
test = horses_f(test)
```

Проверяю данные >500 лс и сверяю их с данными с сайта drom.ru. Если значение не входит в диапазон модели, то заполняю рандомным числом из диапазона.

In [103]:

```
horses2 = [['volkswagen', 'golf', 50, 310],
['bmw', '3er', 75, 387],
['opel', 'astra', 60, 200],
['bmw', 'm_reihe', 195, 560],
['mercedes_benz', 'e_klasse', 72, 612],
['volkswagen', 'passat', 68, 300],
['volkswagen', 'polo', 40, 220],
['opel', 'corsa', 58, 210],
['porsche', 'cayenne', 240, 570],
['renault', 'twingo', 55, 133],
['opel', 'vectra', 57, 280],
['mercedes_benz', 's_klasse', 112, 630],
['mercedes_benz', 'a_klasse', 60, 421],
['renault', 'scenic', 64, 163],
['chevrolet', 'matiz', 51, 82],
['fiat', '500',69,135],
['audi', 'a4',75,286],
['audi', 'a6',90,350],
['nissan', 'micra',50,110],
['jaguar', 's_type',200,395],
['volkswagen', 'transporter',84,204],
['bmw', '5er', 86, 530],
['ford', 'mondeo',88,240],
['ford', 'ka',50, 136],
['ford', 'fiesta',40, 200],
['peugeot', '3_reihe',45,225],
['renault', 'espace',88,241],
['audi', 'a3',110,250],
['opel', 'zafira',82,240],
['mercedes_benz', 'm_klasse', 150, 558],
['volvo', 'v40', 102, 249],
['ford', 'galaxy',90,240],
['citroen', 'c4', 88, 180],
['fiat', 'punto',60,135],
['ford', 'focus', 75, 350],
['volkswagen', 'lupo',50, 125],
['honda', 'civic',45,205],
['volkswagen', 'touran', 90, 190],
['seat', 'arosa', 50, 100],
['lancia', 'ypsilon', 54, 95],
['audi', 'a8', 150, 571],
['mercedes_benz', 'c_klasse', 75, 517],
['citroen', 'c2', 60, 122],
['skoda', 'fabia', 50, 180],
['seat', 'leon', 85, 265],
['volkswagen', 'sharan', 90, 220],
['toyota', 'corolla', 55, 192],
['skoda', 'octavia', 68, 230],
['smart', 'forfour', 60, 177],
['volkswagen', 'kaefer', 25, 50],
['smart', 'fortwo', 41, 109],
['toyota', 'rav', 116, 269],
['toyota', 'yaris', 65, 272],
['audi', '80', 54, 172],
['renault', 'laguna', 83, 241],
['bmw', '1er', 115, 340],
['citroen', 'berlingo', 71, 130],
['dacia', 'sandero', 65, 101],
['daewoo', 'nubira', 106, 136],
```

```
['ford', 'mustang', 85, 760],
['kia', 'rio', 75, 138],
['mazda', '6_reihe', 120, 274],
['mercedes_benz', 'cl', 279, 517],
['mercedes_benz', 'clk', 136, 582],
['mitsubishi', 'colt', 55, 163],
['mitsubishi', 'outlander', 98, 230],
['nissan', 'x_trail', 114, 280],
['peugeot', '2_reihe', 100, 170],
['renault', 'kangoo',55, 115],
['mini', 'cooper', 98, 130]]

```

In [104]:

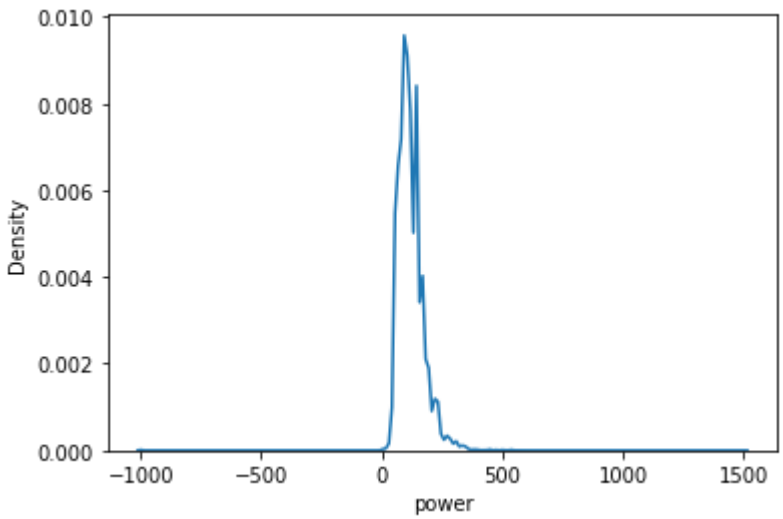
```
def horses_f2(data):
    for one in horses2:
        data.loc[(data.brand==one[0])
                 &(data.model==one[1])
                 &((data.power>one[3])|(data.power<one[2]))
                 ,'power'] = random.randint(one[2],one[3]+1)
    return data
train = horses_f2(train)
test = horses_f2(test)
```

In [105]:

```
sea.kdeplot(train['power'])
```

Out[105]:

```
<AxesSubplot:xlabel='power', ylabel='Density'>
```

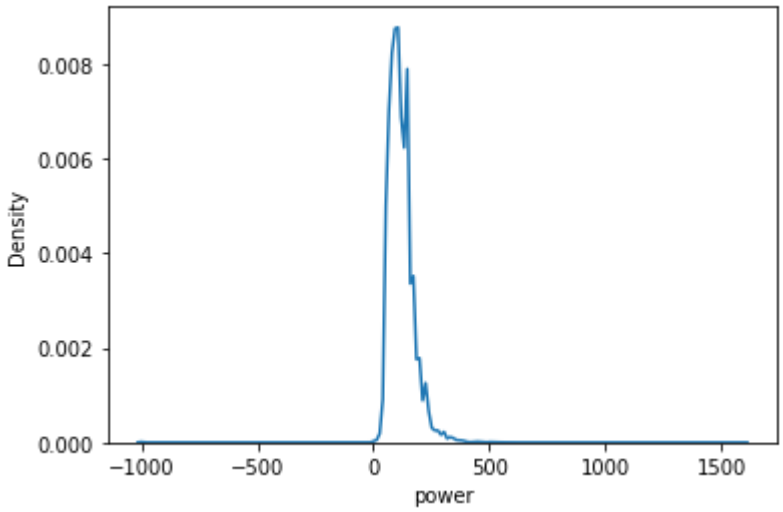


In [106]:

```
1 sea.kdeplot(test['power'])
```

Out[106]:

<AxesSubplot:xlabel='power', ylabel='Density'>



✓ **Успех:** Молодец 👍

model

Больше всего golf, other - 24421, пустые значения 19705. Им можно присвоить other. Некоторые модели есть у нескольких брендов (1_reihe). rangerover и range_rover - похожие названия модели, но выпускаются разными брендами rover и landrover

In [107]:

```
1 def model(data):
2     data.loc[data.model.isna(), 'model'] = 'other'
3     data.model.isna().sum()
4     return data
5 train = model(train)
6 test = model(test)
```

fuel_type

petrol (англ), gasoline (амер)- синонимы, lpg - пропан-бутан, cng - метан. hybrid - гибриддые, electric - электрические, other - 204, нет данных - 32895.
Petrol и gasoline объединю. Пустые заполню как other

In [108]:

```
def fuel_type(data):
    data.loc[data.fuel_type == 'petrol','fuel_type'] = 'gasoline'
    data.loc[data.fuel_type.isna(),'fuel_type'] = 'other'
    return data
train = fuel_type(train)
test = fuel_type(test)
```

not_repaired

not_repaired - 71154 нет данных, yes/no. Нет данных - сделаю отдельной группой

In [109]:

```
def not_repaired(data):
    data.loc[data.not_repaired.isna(),'not_repaired'] = 'no_data'
    data[data.not_repaired.isna()]
    return data
train = not_repaired(train)
test = not_repaired(test)
```

last_seen

Довольно много строк, где год регистрации автомобиля больше, чем год последнего просмотра. Возможно их стоит удалить

In [110]:

```
def last_seen(data):
    data = data.drop(data[data.registration_year>2022].index, axis = 0)
    data.registration_year = pd.to_datetime(data.registration_year, format='%Y')
    data = data.drop(data[data.registration_year>data.last_seen].index, axis = 0
    data.registration_year = data.registration_year.astype('int')

    return data
train = last_seen(train)
test = last_seen(test)
```

In [111]:

```
train.power = train.power.astype('int')
test.power = test.power.astype('int')
```

Не требуют обработки

km - категориальный признак. Пустых значений нет. Больше всего с пробегом 150 тыс.

brand - пропусков и других проблем нет **postal_code** - пропусков нет, важен, так как по сути это регион продажи. По индексу видно, что это даже разные страны

Не нужны

date_crawled Дата скачивания анкеты из базы - если обнаружится какая-то ошибка, то будем изучать,

для модели не нужен. Анкеты скачаны в апреле и марте 2016 года, есть точное время скачивания

reg_month - не понятно, зачем нужен этот признаки. Нулей - 37352. Пропусков нет. Больше всего регистрировались в марте, июне и апреле. Меньше всего в феврале, августе и январе.

date_created - всего 1 дата за 2014 год, 18 дат за 2015, остальные - за 2016 (89 уникальных дат 2016-го года). Пустых нет.

pictures - 354369 с количеством 0, пропусков нет, столбец можно удалить

last_seen - пропусков нет, все пользователи были активны в 2016

In [112]:

```
def drop(data):
    data = data.drop(['date_crawled','reg_month','date_created','pictures','last
    data = data.drop_duplicates()
    return data
train = drop(train)
test = drop(test)
```

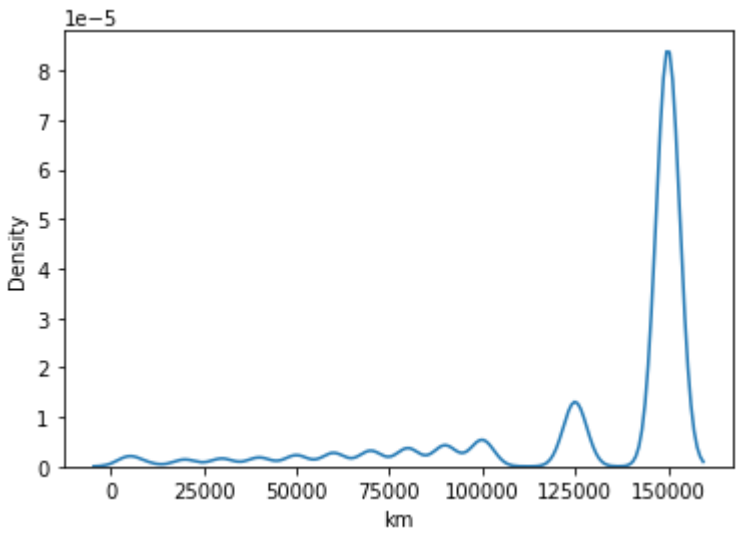
In [113]:

```
sea.kdeplot(train['km'])

```

Out[113]:

```
<AxesSubplot:xlabel='km', ylabel='Density'>
```



In [114]:

```
train[train['power']>1500]
```

Out[114]:

	price	vehicle_type	registration_year	gearbox	power	model	km	fuel_type	bra
145154	1200	sedan	946684800000000000	manual	1503	other	150000	gasoline	br

In [115]:

```
test[test['power']>1500]
```

Out[115]:

	price	vehicle_type	registration_year	gearbox	power	model	km	fuel_type	bra
146238	1500	sedan	820454400000000000	manual	1596	other	150000	gasoline	br

Кодирование признаков

In [116]:

```
target_train = train['price']
features_train = train.drop(['price','price_class'],axis=1)
target_test = test['price']
features_test = test.drop(['price','price_class'],axis=1)

print(target_train.shape, features_train.shape, target_test.shape, features_test
```

```
(234310,) (234310, 10) (80779,) (80779, 10)
```

In [117]:

```
encoder = OrdinalEncoder(handle_unknown = 88888)
encoder.fit(features_train)
features_train_ordinal = pd.DataFrame(encoder.transform(features_train), columns
features_test_ordinal = pd.DataFrame(encoder.transform(features_test), columns =
features_test_ordinal
```

Out[117]:

	vehicle_type	registration_year	gearbox	power	model	km	fuel_type	brand	not_repaired	postal_code
0	6.0	73.0	1.0	101.0	173.0	6.0	2.0	38.0	0.0	3162.0
1	6.0	63.0	1.0	86.0	33.0	12.0	2.0	20.0	0.0	796.0
2	6.0	69.0	1.0	56.0	83.0	9.0	2.0	24.0	2.0	3799.0
3	5.0	71.0	0.0	146.0	180.0	12.0	2.0	23.0	0.0	6118.0
4	5.0	61.0	1.0	86.0	59.0	12.0	2.0	20.0	0.0	7745.0
...	...	...	...	...	...	...	...	...	...	...
80774	4.0	72.0	1.0	64.0	166.0	5.0	2.0	23.0	0.0	4505.0
80775	2.0	69.0	1.0	260.0	248.0	11.0	2.0	2.0	0.0	4841.0
80776	6.0	68.0	1.0	61.0	245.0	11.0	2.0	36.0	0.0	3211.0
80777	8.0	66.0	1.0	105.0	166.0	12.0	2.0	25.0	0.0	5710.0

In [118]:

```
features_train_ohe = pd.get_dummies(features_train,drop_first = True)
features_test_ohe = pd.get_dummies(features_test,drop_first = True)
#for one in features_train_ohe.columns[~features_train_ohe.columns.isin(features
#    features_test_ohe[one] = 0
for one in features_train_ohe.columns:
    if one not in features_test_ohe.columns:
        features_test_ohe[one] = 0
for one in features_test_ohe.columns:
    if one not in features_train_ohe.columns:
        features_train_ohe[one] = 0
features_train_ohe.shape,features_test_ohe.shape

```

Out[118]:

```
((234310, 309), (80779, 309))
```

Корреляция признаков

In [119]:

```
features_train.corr()
```

Out[119]:

	registration_year	power	km	postal_code
registration_year	1.000000	0.108428	-0.214832	0.035511
power	0.108428	1.000000	0.115379	0.057420
km	-0.214832	0.115379	1.000000	-0.013770
postal_code	0.035511	0.057420	-0.013770	1.000000

Ярко выраженной линейной зависимости между признаками нет

Обучение моделей

Константная модель

In [120]:

```
def rmse(target, prediction):
    rmse = mean_squared_error(target, prediction, squared = False)
    return rmse

```

In [121]:

```
prediction = [data.price.median()]*len(data)
rmse(data.price, prediction)

```

Out[121]:

```
4812.3097981057535
```

Линейная регрессия

In [122]:

```
features_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 234310 entries, 275374 to 39122
Data columns (total 10 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   vehicle_type       234310 non-null  object
 1   registration_year  234310 non-null  int64 
 2   gearbox            234310 non-null  object
 3   power              234310 non-null  int64 
 4   model              234310 non-null  object
 5   km                 234310 non-null  int64 
 6   fuel_type          234310 non-null  object
 7   brand              234310 non-null  object
 8   not_repaired       234310 non-null  object
 9   postal_code        234310 non-null  int64 
dtypes: int64(4), object(6)
memory usage: 19.7+ MB
```

In [123]:

```
train_rmse = [0,0,0]
test_rmse = [0,0,0]
fit_time = [0,0,0]
p_time = [0,0,0]

pd.options.mode.chained_assignment = None
scaler = StandardScaler()
scaler.fit(features_train_ohe[['power','km','postal_code','registration_year']])
features_scaled_train = features_train_ohe
features_scaled_test = features_test_ohe
features_scaled_train[['power','km','postal_code','registration_year']] = scaler
features_scaled_test[['power','km','postal_code','registration_year']] = scaler.
```

In [124]:

```
model_l = LinearRegression()
scor = make_scorer(rmse, greater_is_better=False)
scores = cross_val_score(model_l, features_scaled_train, target_train, cv=5, sco
train_rmse[2]= scores.mean()
print(train_rmse[2])

start_time = time.time()
model_l.fit(features_scaled_train, target_train)
fit_time[2] = (time.time() - start_time)
print("Время обучения %s с" % (fit_time[2]))
start_time = time.time()
predict_l = model_l.predict(features_scaled_train)
p_time[2] = time.time() - start_time
print("Время предсказания %s с" % p_time[2])
```

```
-2755.2868212521785
Время обучения 21.788912057876587 с
Время предсказания 0.40083742141723633 с
```

LightGBM

In [125]:

```

trees = [10, 50, 100]
best_scores = 999999
parametrs = {'n_estimators': range (10, 40, 10),
             'max_depth': range (10,40,10),
             'num_leaves': range (10,40,10)}
model_lgbm = LGBMRegressor()
grid = GridSearchCV(model_lgbm, parametrs, cv=5, scoring = 'neg_root_mean_square

start_time = time.time()
grid.fit(features_train_ordinal, target_train)
fit_time[0] = (time.time() - start_time)

train_rmse[0] =  grid.best_score_
print(grid.best_estimator_, train_rmse[0])
print("Время обучения %s с" % (fit_time[0]))

start_time = time.time()
predict_lgbm = grid.predict(features_train_ordinal)
p_time[0] = time.time() - start_time
print("Время предсказания %s с" % p_time[0])

```

```
LGBMRegressor(max_depth=10, n_estimators=30, num_leaves=30) -1975.0848
555962239
Время обучения 782.7031135559082 с
Время предсказания 0.5144298076629639 с
```

RandomForest

In [126]:

```
trees = [10, 50, 100]
best_scores = 999999
parametrs = {'n_estimators': range(10, 40,20),
             'max_depth': range(10,40,10),
             'min_samples_split': [0.1,0.2,0.3],
             'min_samples_leaf': range(10,40,10)}
model_rf = RandomForestRegressor()
grid_rf = GridSearchCV(model_rf, parametrs, cv=5, scoring = 'neg_root_mean_squar
start_time = time.time()
grid_rf.fit(features_train_ordinal, target_train)
fit_time[1]=(time.time() - start_time)
train_rmse[1] = grid_rf.best_score_
print(grid_rf.best_estimator_, train_rmse[1])
print("Время обучения %s с" % (fit_time[1]))
start_time = time.time()
predict_rf= grid_rf.predict(features_train_ordinal)
p_time[1] = time.time() - start_time
print("Время предсказания %s с" % p_time[1])

```

```
RandomForestRegressor(max_depth=30, min_samples_leaf=20, min_samples_s
plit=0.1,
                      n_estimators=10) -2944.008969945214
Время обучения 760.2367017269135 с
Время предсказания 0.08147072792053223 с
```

Анализ моделей

In [127]:

```
work_time = pd.DataFrame(columns = ['name','time_predict','time_fit', 'train_rms
work_time.time_fit = fit_time
work_time.time_predict = p_time
work_time.name = ['LGBM','RandomForest','LinearRegression']
work_time.train_rmse = np.array(train_rmse).astype('int')
work_time
```

Out[127]:

	name	time_predict	time_fit	train_rmse
0	LGBM	0.514430	782.703114	-1975
1	RandomForest	0.081471	760.236702	-2944
2	LinearRegression	0.400837	21.788912	-2755

Лучший результат rmse - у градиентного бустинга. Быстрее работает линейная регрессия.

Тестирование лучшей модели

In [131]:

```
start_time = time.time()
best_prediction = grid.predict(features_test_ordinal)
fit_time_best = (time.time() - start_time)
best_rmse = rmse(best_prediction,target_test)
print('RMSE лучшей модели', best_rmse)
print("Время обучения %s с" % work_time.time_fit[0])
print("Время предсказания %s с" % fit_time_best)

```

```
RMSE лучшей модели 1988.5825310269688
Время обучения 782.7031135559082 с
Время предсказания 0.19648075103759766 с
```

Рекомендую заказчику использовать модель LightGBM, так как она показывает лучшее качество предсказаний и единственная выполняет поставленную задачу по метрике качества. При этом она показывает худший результат по времени. В случае неудовлетворенности результатом предложу продожлить подбор гиперпараметров для RandomForest.

Вывод

In [ ]:

```

# Не забывай про финальный вывод)
```

В рамках проекта проделана работа по предобработке данных, обучению 3 моделей для предсказания рыночной цены автомобиля. Модель LightGBM работает дольше других моделей, но показывает лучшее качество в текущей задаче регрессии и является рекомендованной к применению.