Отчет по выполненной работе “Предсказание стоимости б/у автомобиля”

Выполнила А.В. Болтнева

DS_18+

Содержание

Введение

Целью проекта является получить предсказание стоимости автомобиля, которое наиболее соответствует реальной стоимости. Оцениваться предсказания будет метрикой MAPE. Чем ниже значение этой метрики, тем точнее предсказывает модель.

Рекомендательная система оценки стоимости автомобиля основывается на выявлении зависимостей и закономерностей имеющихся в характеристиках автомобиля, продавца, даты продажи и т.д.

Нам предстоит решить задачу регрессии.

Этапы нашего исследования включают в себя:

- загрузка и ознакомление с данными;
- анализ и предварительная обработка данных, включающая в себя анализ пропусков, выявление дубликатов, приведение к необходимому типу данных.
- анализ имеющихся признаков и разработка синтетических признаков;
- подбор гиперпараметров и обучение модели;
- итоговая оценка качества предсказания лучшей модели;
- анализ важности ее признаков.

Библиотеки, которые использовались в ходе выполнения работы:

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import datetime
import catboost
import optuna
```

+ Code + Markdown

[3]:
```
from sklearn.model_selection import train_test_split
from catboost import CatBoostRegressor, Pool, cv
from sklearn.model_selection import GridSearchCV
from scipy.stats import uniform
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
```

В работе использовался градиентный бустинг модель CatboostRegression библиотеки Catboost.

Описание исходных данных

В рамках данной работы нам были предоставлены два набора данных, один назывался train, который мы использовали для обучения и валидации, а второй test, используемый исключительно для оценки результирующей модели.

Оба набора данных включали в себя одинаковые признаки, за тем исключением, что в test отсутствовал целевой признак selling price, вероятно, чтобы исключить подгонку результатов.

- train data включающий в себя 440236 наблюдений;
- test содержит 110058 наблюдений.

Описание признаков

Первоначальный датасет содержал следующие признаки:

- **year** - год производства
- **make** - производитель
- **model** - модель
- **trim** - модификация
- **body** - тип кузова
- **transmission** - тип КПП
- **vin** - идентификатор (вин)
- **state** - штат регистрации
- **condition** - состояние по шкале (1-5)
- **odometer** - пробег в милях
- **color** - цвет кузова
- **interior** - цвет интерьера
- **seller** - продавец
- **sellingprice** - стоимость продажи
- **saledate** - дата продажи.

Тип данных

Признаки в данных представлены категориальными (object) и числовыми (int64, float64) признаками.

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	sellingprice	saledate
0	int64	object	object	object	object	object	object	object	float64	float64	object	object	object	int64	object

Предобработка и исследовательский анализ

Обе выборки обрабатывала идентично, поэтому что написано про train актуально для test.

Первым шагом я удалила неинформативный признак VIN.

Предобработку данных я начала с приведения данных к нужному формату:

Признак saledate разбила на синтетические признаки:

- sale_year
- sale_month.
- удалила дублирующий признак saledate.

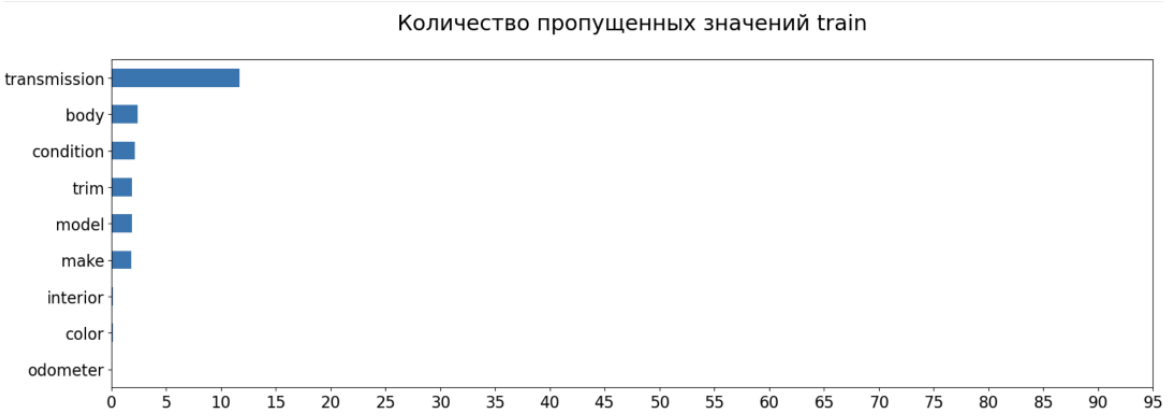
saledate
Wed Jan 14 2015 04:30:00 GMT-0800 (PST)
Fri Feb 27 2015 01:00:00 GMT-0800 (PST)
Tue Feb 24 2015 01:30:00 GMT-0800 (PST)
Fri Mar 06 2015 02:00:00 GMT-0800 (PST)
Wed Jun 03 2015 03:30:00 GMT-0700 (PDT)

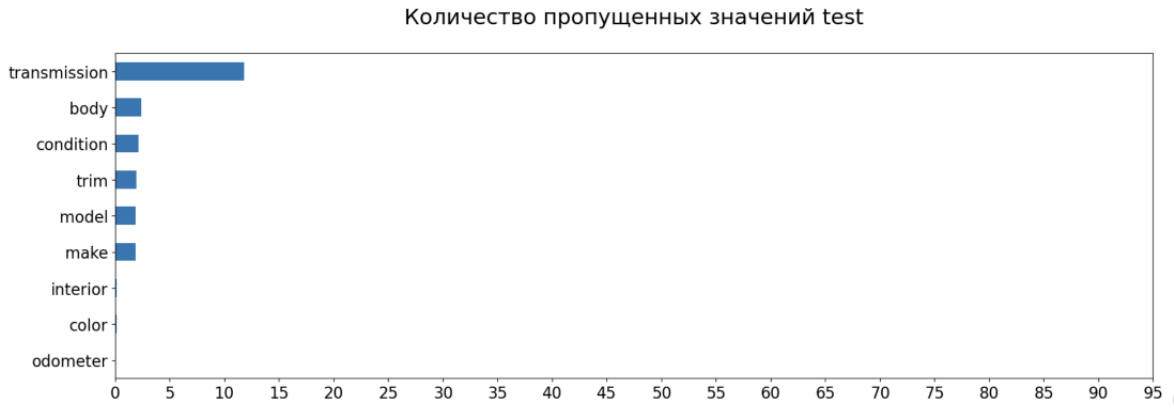
Признак saledate был представлены в формате object, я заменила дату на формат datetime. Это было необходимо сделать для того, чтобы можно было разделить дату на год и месяц.

Пропущенные значения

В исходных данных имелось значительное число пропущенных значений (от 15%).

Пропуски имелись как в категориальных, так и в числовых признаках. В обоих датасетах пропущенные значения имелись в одних и тех же признаках и сопоставимы по количеству.





Несмотря на то, что выбранная мной модель может работать с пропущенными значениями, я приняла решение предварительно заполнить пропущенные значения чтобы сократить время вычисления модели.

Основная часть пропусков содержалась в категориальных признаках, там я заполнила все пропуски заглушкой "no_info".

Пропуски в числовых признаках содержались в основном только в condition (9405), а в odometer (69).

Исходя из предположения, что между признаками odometer и condition имеется прямая связь я заполнила пропуски в condition средним значением для пробега:

- до 30 тыс
- 30-40
- 40-50
- 50-60
- 60-80
- 80-100
- 100-150.

После 150 тыс я сочла нецелесообразным заполнять по такому же принципу, так как состояние автомобилей с таким большим пробегом уже не является определяющим фактором. Тем более, что это в милях, а значит в километрах, это огромный пробег для любого автомобиля.

	0.02	0.05	0.10	0.50	0.65	0.75	0.85	0.93	0.99
odometer	5856.32	10475.0	15616.0	52098.0	77689.9	99272.0	125266.0	157501.38	226987.68

После каждого заполнения проверяла распределение выборки.

Пропуски в condition (после 150 тыс) и odometer заполнила нереальным для данного набора данных числом -999, чтобы модель учла факт пропусков.

Аномалии и редкие значения

Имеются аномалии и редкие значения в признаках

odometer - пробег в милях

year - год производства

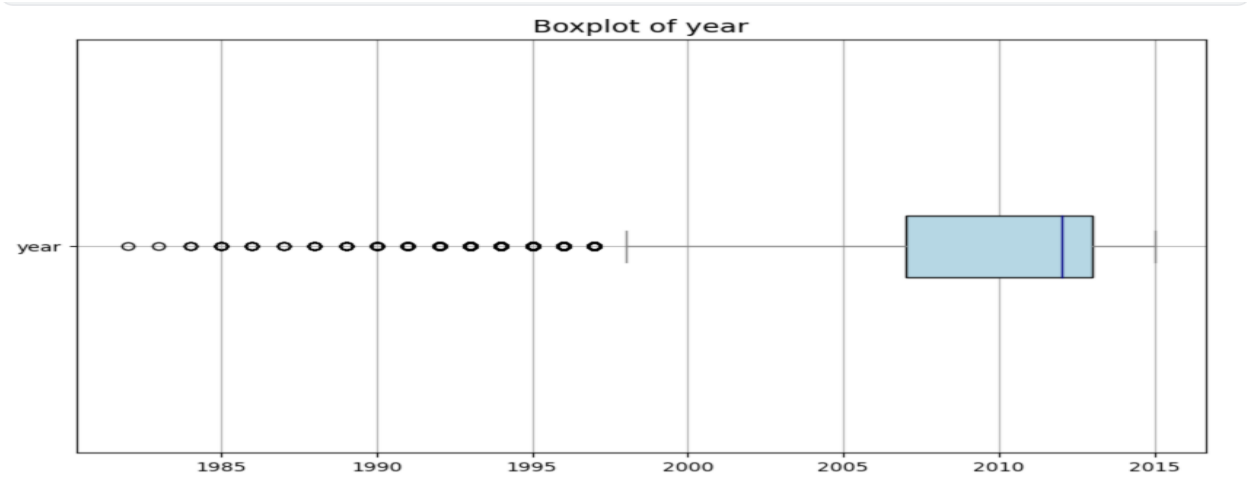
selling_price стоимость автомобиля на момент продажи

Year

Мы видим, что только менее двух процентов автомобилей выпущены ранее 2000 г.

	0.02	0.05	0.10	0.50	0.65	0.75	0.85	0.90	0.99
year	2000.0	2002.0	2004.0	2012.0	2012.0	2013.0	2014.0	2014.0	2015.0

На графике ниже представлено распределение объектов по году выпуска. А дата продажи представлена только 2014, 2015 годами.

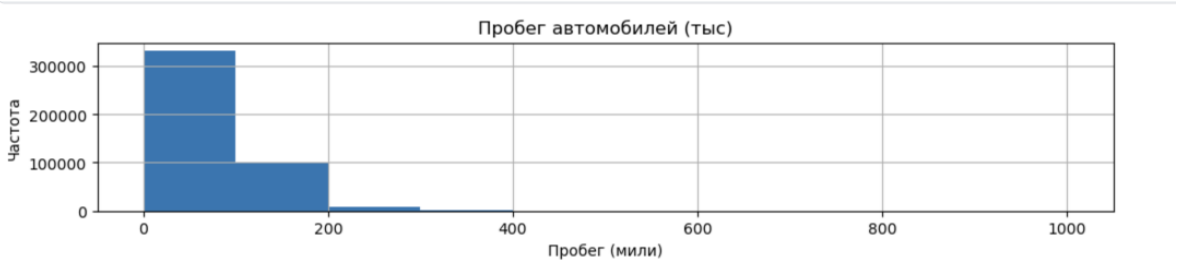


Учитывая, что по условиям задачи нам нельзя удалять объекты, то на данном этапе я просто зафиксировала что они имеются.

Odometer

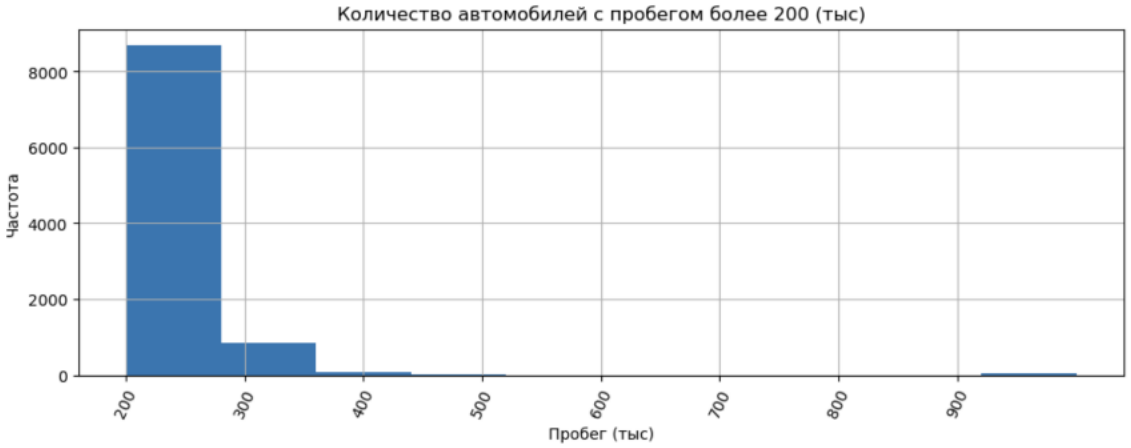
Значения в этом признаке для удобства я разделила на 1000.

Ниже представлен график распределения количества автомобилей в зависимости от пробега

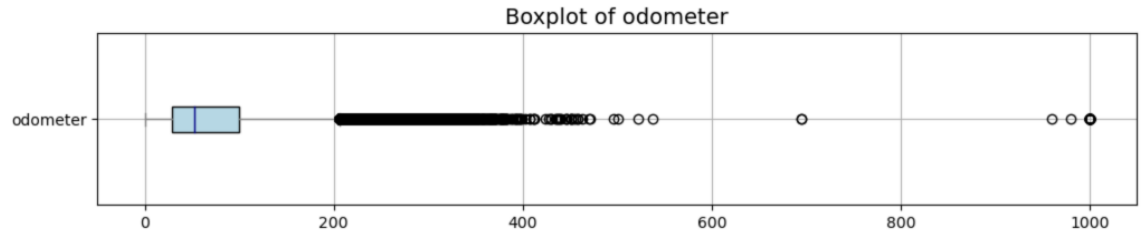


На этом графике мы увидели, что пробег основной массы автомобилей лежит в диапазоне до 200 тыс миль.

Далее анализируем распределение автомобилей с пробегом после 200 тыс миль:



на боксплоте мы видим, что после 200 тыс уже редкие значения, и значение 1 млн вообще представлялся нереалистичным, учитывая что единица измерения мили.



	0.0200	0.0500	0.5000	0.6500	0.7500	0.8500	0.9000	0.9825	0.9900
odometer	5.85632	10.475	52.098	77.6899	99.272	125.266	143.186	207.879475	226.98768

На боксплоте так же подтверждается, что после 200 тыс объекты уже можно считать редкими и аномальными.

Целевой признак sellingprice.

Единица измерения нам так же доподлинно не известна, но вероятнее всего это доллары.

В этом признаке также имеются наблюдения похожие на аномалии и/или редкие значения, например объекты дешевле 700.

Ниже представлены средние значения признаков year , condition, odomerer для объектов, которые дешевле 700:

	0
year	2000.298489
condition	1.871080
odometer	165823.042642

При анализе этого признака для меня важнее был не факт низкой или высокой стоимости, а непротиворечие между признаком sellingprice и остальными признаками, например год выпуска, марка пробег., отсутствие отрицательных значений и т.д.

Категориальные признаки

В категориальных признаках обнаружились следующие проблемы:

- разный регистр;
- пробелы;
- разные знаки внутри слов;
- цифры написаны слитно со словами.

Обработка категориальных признаков включала в себя:

- заменила пробелы на underscore _

- заменила дефис на underscore _
- добавила underscore между словом и цифрой
- привела к строчному регистру.

Это позволило мне избавиться от неявных дубликатов:

train До/После

make 86	make 63
model 825	model 827
trim 1497	trim 1832
body 78	body 45
transmission 3	transmission 3
state 38	state 38
color 21	color 21
interior 18	interior 18
seller 7623	seller 13018

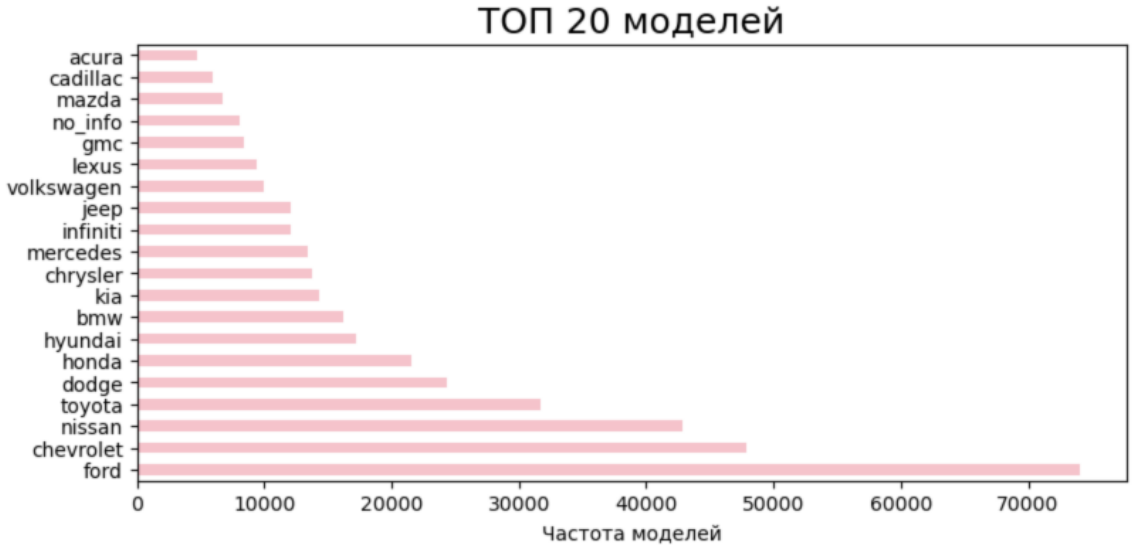
test До/После

make 86	make 59
model 825	model 734
trim 1497	trim 1446
body 78	body 42
transmission 3	transmission 3
state 38	state 38
color 21	color 21
interior 18	interior 18
seller 7623	seller 7619

p. s. не везде добавился знак _ между словом и цифрой, там где только одна буква и цифра функция не сработала. Это было в моделях, если значение состоит из одной буквы и цифры например s150. Я исправила вручную только там где увидела дубликаты. Это обязательно нужно доработать если использовать модель с интерфейсом пользовательским.

Далее я удалила неявные дубликаты в признаке make:

- train 63/55
- test 59/54.
- удалила неявные дубликаты в признаке model самых популярных марок автомобилей.



Резюме:

После предобработки и анализа мои данные имели следующие характеристики:

1. признаки:
- year
- make
- model
- trim
- body
- transmission
- state
- condition
- odometer
- color
- interior
- seller
- sellingprice
- sale_year
- sale_month

2. пропуски заполнены

3. выявлены аномалии

4. удалены неявные дубликаты.

Обоснование выбора модели и метрики

Прежде чем приступить к описанию выбранной модели резюмируем что мы имеем:

- Задача регрессии;
- Данные неоднородные;
- Признаки представлены категориальными и числовыми значениями;
- Редкие значения и аномалии.

ДЛя решения этой задачи можно было также использовать модель случайного леса.

Выбор алгоритма и модели

Исходя из вышесказанного для решения задачи я выбрала модель CatBoost градиентного бустинга.

Во первых CatBoost имеет встроенный алгоритм кодирования категориальных признаков и не нуждается в предварительном перекодировании категориальных признаков.

Во-вторых он может сам заполнить пропуски.

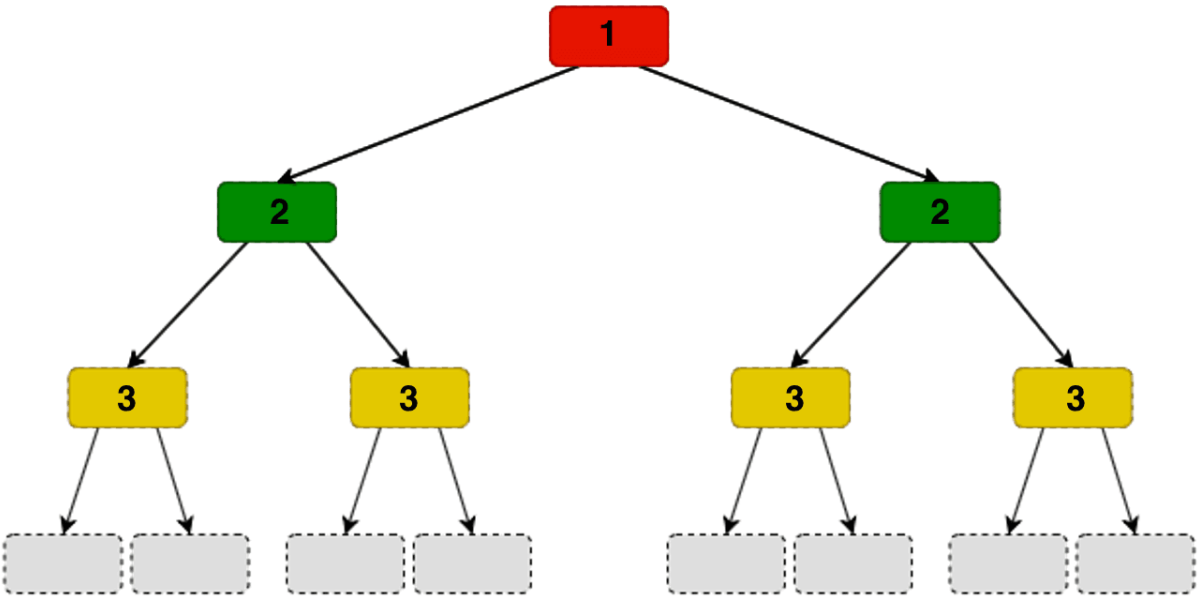
В третьих этот алгоритм и конкретная модель отлично себя зарекомендовали для работы с категориальными неоднородными признаками.

Такой бустинг способен эффективно находить нелинейные зависимости в данных различной природы. Этим свойством обладают все алгоритмы, использующие деревья решений, однако именно GBDT обычно выигрывает в подавляющем большинстве задач. Бустинг, использующий деревья решений в качестве базовых алгоритмов, называется градиентным бустингом над решающими деревьями, Gradient Boosting on Decision Trees, GBDT.

Он отлично работает на выборках с «табличными», неоднородными данными. Примером таких данных может служить описание автомобилей в наших данных: год выпуска, марка, модель, пробег, цвет, состояние, характеристика кузова и т.д. [https://academy.yandex.ru/handbook/ml/article/gradientnyj-busting]

CatBoost строит деревья по принципу: «Все вершины одного уровня имеют одинаковый предикат». Одинаковые сплиты во всех вершинах одного уровня

позволяют избавиться от ветвлений (конструкций if-else) в коде инференса модели с помощью битовых операций и получить более эффективный код, который в разы ускоряет применение модели, в особенности в случае применения на батчах. Кроме этого, такое ограничение на форму дерева выступает в качестве сильной регуляризации, что делает модель более устойчивой к переобучению. Основным критерием остановки, как и в случае XGBoost, является ограничение на глубину дерева. Однако, в отличие от XGBoost, в CatBoost всегда создаются полные бинарные деревья. Такие деревья обычно являются более устойчивыми к переобучению.несмотря на то, что в некоторые поддеревья может не попасть ни одного объекта из обучающей выборки.



Метрика качества

В условиях данной работы у нас уже определена метрика для оценки предсказаний, это MAPE - mean absolute percentage error. Поэтому эта метрика останется в качестве пользовательской метрики.

$$MAE(y^{true}, y^{pred}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)|$$

В качестве метрики для обучения модели я выбрала mean squared error, потому что эта метрика квадратично штрафует за большие ошибки, а значит хорошо подходит для работы с редкими значениями и выбросами.

$$MSE(y^{true}, y^{pred}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$

Обучение модели и подбор гиперпараметров

Train выборку я разделила на обучающую (train_data) и валидацинную (validation).

```
#Разделим выборки
train_data, validation = train_test_split(train, random_state=42)
print(train_data.shape[0])
print(validation.shape[0])
```

```
330177
110059
```

+ Code + Markdown

```
#выделим обучающие признаки и целевой признак
X = ['year', 'make', 'model', 'trim', 'body', 'transmission', 'state','condition', 'odometer', 'color',
     'interior', 'seller', 'sale_year','sale_month']
y = ['sellingprice']
```

Далее я создала объекты Pool (библиотеки catboost).

Его преимущество заключается в том, что вместо того, чтобы передавать обучающие признаки и целевой признак обучающей, валидационной и тестовой мы передаем объект Pool. Объекты библиотеки catboost могут принимать данные объекты в качестве параметров.

test_pool содержит всю train выборку, чтобы лучше обучить модель перед предсказанием на тестовой выборке. Параметры выборок имеют следующие характеристики:

```
train_pool = Pool(data=train_data[X],
                  label=train_data[y],
                  cat_features=cat_features)
print(train_pool.shape[0])
```

```
330177
```

```
validation_pool = Pool(data=validation[X],
                       label=validation[y],
                   cat_features=cat_features)
print(validation_pool.shape[0])
```

```
110059
```

```
test_pool = Pool(data=train[X],
                   label=train[y],
                   cat_features=cat_features)
print(test_pool.shape[0])
```

```
440236
```

Подбор гиперпараметров

Учитывая, что модель я уже выбрала, мне необходимо лишь подобрать для нее оптимальные гиперпараметры, которые позволят мне получить предсказания необходимой точности.

Далее я опишу параметры, которые я использовала в своей модели.

- loss_function

метрика, используемая при обучении модели. В моей работе использована метрика RMSE. Метрика MSE квадратично штрафует за большие ошибки в предсказаниях и модель должна хорошо подстраиваться под выбросы, а в наших данных они имеются.

Именно эту метрику модель будет использовать в качестве обучения и будет стремиться ее уменьшить.

- eval_metric

эта метрика используется для обнаружения переобучения и выбора лучшей модели.

- iterations

максимальное количество деревьев, которые модель может строить для обучения

- learning_rate

Это один их важнейших параметров алгоритма, выбор которого сильно влияет на работу модели.

- random_seed аналогично random_state. Используется для воспроизводимости
- l2_leaf_reg. Параметр, отвечающий за штраф модели в случае неправильных предсказаний. Значение по умолчанию = 3.
- depth глубина дерева

- verbose я использовала значение = 100. Выводит каждые 100 итераций
- early_stopping_rounds детектор переобучения. Модель перестает обучаться после того как достигнуто оптимальное значение метрики.

Инициировала модель и передала ей метрики, записанные в словаре:

```
params = {
    'cat_features': cat_features,
    'eval_metric': 'MAPE',
    'loss_function': 'RMSE',
    'learning_rate': 0.15,
    'random_seed': 42,
    'verbose': 100
}
```

+ Code + Markdown

```
#инициализируем модель
model = CatBoostRegressor(**params)
```

Подбор гиперпараметров.

Далее я подбирала гиперпараметры с помощью двух способов:

- RandomGridSearch (https://catboost.ai/en/docs/concepts/python-reference_catboost_randomized_search)
- Optuna.

Randomized_search

Для подбора гиперпараметров модели я использовала встроенный метод библиотеки catboost, который называется RandomGridSearch.

Параметры, которые я буду подбирать:

iterations, Количество итераций во время обучения модели. По умолчанию этот параметр равен 1000.

Увеличение числа итераций может способствовать более глубокому обучению модели и нахождению неявных паттернов, но также увеличение этого параметра может приводить к переобучению модели и увеличивает время вычисления.

learning_rate шаг на который модель увеличивает свои встроенные алгоритмы, то есть как быстро сокращает нивелирует ошибку предыдущего алгоритма. Слишком высокое число так же ведет к переобучению и может пропустить оптимальные параметры, так как слишком быстро увеличивает значения. Меньшее значение этого параметра увеличивает точность модели.

depth, (глубина деревьев) указывает насколько сложный алгоритм модель может строить в рамках обучения. Увеличение этого параметра также может влиять на то что модель лучше находит неявные и скрытые паттерны, но увеличивается время вычисления. А уменьшение этого параметра ведет к упрощению модели, т.е. точность будет ниже. Рекомендуемые разработчиком от 6 до 10

l2_leaf_reg контролирует силу регуляционного коэфициента. Увеличивая это значение можно регулировать переобучение, наказывая большими штрафами за большую ошибку, иными словами чем больше ошибка тем больше штраф. По умолчанию 3

random_state фиксированный параметр, для воспроизводимости.

```
# параметры для перебора
param_grid = {
    'iterations':[3000, 4000],
    'learning_rate': [0.01, 0.10],
    'depth': [4, 6, 15],
    'l2_leaf_reg': [1, 3, 5, 7, 9]
```

Преимуществом этого метода является то, что после поиска гиперпараметров, модель можно сразу использовать, нет необходимости ее обучать на тренировочных данных.

Подбор параметров не занимает много времени.

К недостаткам я отнесла качество получаемых результатов

- при использовании Optuna я получила метрику 14.0 тогда как randomized_search только 16.4.

```
randomized_search_result = model.randomized_search(param_grid,
                                                   train_pool)
```

Промежуточным результатом выполнения этой ячейки является такой вывод (напоминаю, что у меня параметр verbos=100 регулировал, что выводиться должен каждая 100 итерация).

```
3700:   learn: 0.2973234        test: 0.2302565 best: 0.2302556 (3699)  total: 12m 53s  remaining: 1m 2s
Stopped by overfitting detector  (15 iterations wait)

bestTest = 0.2299843634
bestIteration = 3722

0:      loss: 0.2299844 best: 0.2299844 (0)     total: 13m 1s   remaining: 1h 57m 17s
0:      learn: 0.8768508        test: 0.8618739 best: 0.8618739 (0)     total: 269ms    remaining: 17m 57s
100:    learn: 0.3595880        test: 0.2617273 best: 0.2617273 (100)   total: 21.5s    remaining: 13m 48s
200:    learn: 0.3203563        test: 0.2474506 best: 0.2472794 (197)   total: 42.5s    remaining: 13m 23s
300:    learn: 0.3001574        test: 0.2360068 best: 0.2359964 (298)   total: 1m 3s    remaining: 13m 3s
400:    learn: 0.2898373        test: 0.2283126 best: 0.2282254 (398)   total: 1m 25s   remaining: 12m 48s
500:    learn: 0.2812555        test: 0.2231970 best: 0.2231970 (500)   total: 1m 47s   remaining: 12m 32s
600:    learn: 0.2725876        test: 0.2180530 best: 0.2180460 (599)   total: 2m 9s    remaining: 12m 13s
700:    learn: 0.2675495        test: 0.2145097 best: 0.2144589 (698)   total: 2m 31s   remaining: 11m 53s
800:    learn: 0.2610469        test: 0.2105909 best: 0.2105909 (800)   total: 2m 53s   remaining: 11m 31s
900:    learn: 0.2588340        test: 0.2079239 best: 0.2078650 (897)   total: 3m 14s   remaining: 11m 8s
1000:   learn: 0.2558113        test: 0.2060282 best: 0.2060282 (1000)  total: 3m 36s   remaining: 10m 47s
1100:   learn: 0.2525459        test: 0.2039553 best: 0.2039553 (1100)  total: 3m 58s   remaining: 10m 28s
1200:   learn: 0.2481601        test: 0.2018050 best: 0.2017988 (1195)  total: 4m 21s   remaining: 10m 8s
1300:   learn: 0.2454016        test: 0.2000842 best: 0.2000669 (1298)  total: 4m 42s   remaining: 9m 45s
1400:   learn: 0.2412907        test: 0.1991344 best: 0.1991304 (1399)  total: 5m 3s    remaining: 9m 23s
1500:   learn: 0.2396626        test: 0.1978684 best: 0.1978576 (1498)  total: 5m 25s   remaining: 9m 2s
1600:   learn: 0.2377615        test: 0.1965874 best: 0.1965874 (1600)  total: 5m 47s   remaining: 8m 40s
1700:   learn: 0.2354243        test: 0.1951352 best: 0.1951352 (1700)  total: 6m 9s    remaining: 8m 19s
1800:   learn: 0.2339449        test: 0.1939662 best: 0.1939662 (1800)  total: 6m 31s   remaining: 7m 58s
1900:   learn: 0.2319411        test: 0.1928053 best: 0.1928053 (1900)  total: 6m 53s   remaining: 7m 37s
2000:   learn: 0.2301869        test: 0.1914461 best: 0.1914461 (2000)  total: 7m 15s   remaining: 7m 15s
2100:   learn: 0.2287729        test: 0.1904036 best: 0.1904036 (2100)  total: 7m 38s   remaining: 6m 54s
```

самая большая проблема, которую мне не удалось решить, это скорость работы этих методов.

начало работы

```
2023-06-02 18:09:23.872900
```

конец работы

```
2023-06-02 22:48:09.499553
```

подбор трех гиперпараметров на моей выборке занял почти 5 часов.

```
# параметры для перебора
param_grid = {
    'learning_rate': [0.01, 0.10],
    'depth': [6, 15],
    'l2_leaf_reg': [1, 3, 5, 7, 9]
}
```

Я намеренно убрала параметр iteration из параметров для перебора, так как это существенно затягивало выполнение, хотя для получения наилучших параметров этот параметр должен быть выше, чем по умолчанию.

В результате выполнения я получила следующие гиперпараметры:

depth = 15
l2_leaf_reg=7
learning_rate=0.1.

best_score

MAPE=0.09145215595530626

RMSE = 955.8361138003424

на валидационной выборке:

```
Root Mean Squared Error (RMSE): 2216.334
Mean Absolute Percentage Error (MAPE): 0.152%
```

На тестовой выборке это дало мне результат MAPE= 16.7.

Судя по всему тактика была учиться быстро, сильно штрафуя за большие ошибки. Так как скорость обучения довольно высокая, l2 reg по умолчанию = 3 в нашей модели он равен 6 при небольшой глубине деревьев.

После того, как я заменила выбросы в признаке odometer значением .98 квантиля равным 250 тыс, то значение метрики MAPE на тестовой выборке ухудшилось на 0.3.

Далее я пробовала уменьшить количество уникальных категорий в признаке color и оставила только основные цвета (black, white, silver,gray, blue,red), а остальные добавила в категорию other - MAPE выросла до 17.

Убрала неявные дубликаты в make и model - значения MAPE упали до 16.

Таким образом, благодаря подбору гиперпараметров методом randomized_search мы достигли метрики MAPE 16.0, но это занимает на нашей выборке от 4 часов.

OPTUNA

Оптуной я подбирала те же гиперпараметры, что и методом randomized_search.

Функция для OPTUNA выглядела таким образом:

```
def objective(trial):
    # определяем гиперпараметры для подбора
    params = {
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.15),
        'depth': trial.suggest_int('depth', 5, 10),
        'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 0.1, 10),
        'random_state': 42 }
     # создаем модель
    model = catboost.CatBoostRegressor(**params)
    model.fit(train_pool, verbose = False)
    # предсказываем на валидационной выборку
    predictions = model.predict(validation_pool)
    # выводим метрику для обучения (eval)
    rmse = mean_squared_error(validation[y], predictions, squared=False)
    return rmse
# создаем объект study
study = optuna.create_study(direction='minimize')
# оптимизация
study.optimize(objective, n_trials=100)
study.optimize(objective, n_trials=100, timeout=60*60*2)
# выводим лучшие гиперпараметры
best_params = study.best_params
print (best_params)
```

и я получила следующие параметры:

```
best_params = {'depth': 9,
               'iterations': 3000,
               'learning_rate': 0.07555198107934782,
               'l2_leaf_reg': 8.206677907347204,
               'cat_features': cat_features,
    'eval_metric': 'MAPE',
    'loss_function': 'RMSE',
    'random_seed': 42,
    'verbose': False,
    'early_stopping_rounds' : 15}
```

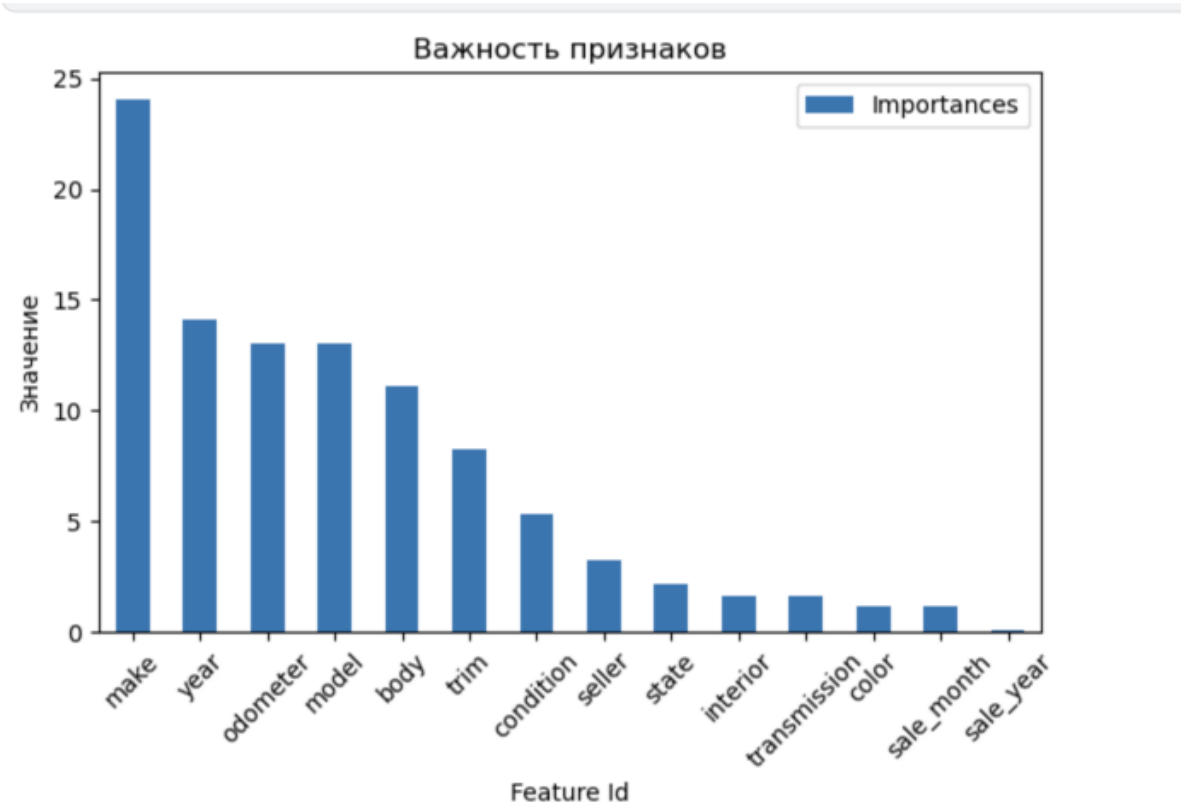
Результатом работы модели с этими гиперпараметрами стала метрика MAPE = 14.0.

Это был лучший результат, в моей работе.

Но в работе оптуны я так же не смогла добиться быстрой работы, работа оптуны занимала более 12 часов, для 25 попыток.

Анализ важности признаков

Важность признаков по результату обучения модели, полученная атрибутом get_feature_importance(prettified=True) библиотеки catboost



Дальнейшего анализа признаков я не выполняла, а эти признаки использовала для принятия решения о том, какой из признаков необходимо обработать более внимательно.

Удаление неявных дупликатов в признаках make, body, model в совокупности помогли уменьшить метрику примерно на 0.7 (с 16.7 до 16.0).

Выводы и рекомендации

Результатом работы модели стала метрика MAPE=14.0

Данные были предобработаны в следующем объеме:

- заполнены пропуски синтетическими данными (no_info, среднее значение для параметра, число -999)
- добавлены синтетические признаки (год и месяц продажи)
- были использованы разные техники подбора гиперпараметров
- проанализирована важность признаков.

Рекомендации по дальнейшей работе

1. В ходе дальнейшей работы необходимо более подробно проанализировать признаки и их важность, разработать синтетические признаки.
2. Проанализировать пропуски и выявить более качественный алгоритм их заполнения, возможно используя специальные библиотеки.
3. Использовать другие модели и метрики, возможно использовать библиотеки для подбора и сравнения моделей.
4. Проанализировать выбросы и аномалии, разработать алгоритм их устранения, улучшающий метрику модели.

Необходимо детальнее изучить целевой признак, а особенно, уделить внимание постобработке, так как модель предсказывала отрицательные значения, которые я не заметила.