Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Отчет по лабораторной работе №3

«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

по дисциплине «Технологии машинного обучения»

Выполнил:
студент группы ИУ5Ц-84Б
Папин А.В.
подпись, дата

Проверил:
к.т.н., доц., Ю.Е. Гапанюк
подпись, дата

2024 г.

СОДЕРЖАНИЕ ОТЧЕТА

1. Цель лабораторной работы:

Изучение способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

2. Описание задание

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
5. Произведите подбор гиперпараметра K с использованием GridSearchCV и RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Используйте не менее двух стратегий кросс-валидации.
6. Сравните метрики качества исходной и оптимальной моделей.

3. Основные характеристики датасета

Название датасета: Used Cars Dataset (Датасет поддержанных (б\у) автомобилей)

Ссылка: https://www.kaggle.com/datasets/andreinovikov/used-cars-dataset

О датасетах

Этот набор данных содержит данные о 762 091 подержанном автомобиле, собранном из cars.com . Данные были собраны в апреле 2023 года.

Датасет состоит из 20 столбцов и 762 091 строк, где каждая строка представляет:

manufacturer - название производителя автомобиля

model - название модели автомобиля

year - год, когда был выпущен автомобиль

mileage - миль, пройденных автомобилем с момента выпуска

engine - автомобильный двигатель

transmission - тип трансмиссии автомобиля

drivetrain - тип трансмиссии автомобиля

fuel_type - тип топлива, которое потребляет автомобиль

mpg - количество миль, которое автомобиль может проехать, используя один галлон топлива (мили на галлон)

exterior_color - цвет кузова автомобиля

interior_color - цвет салона автомобиля

accidents_or_damage - попадал ли автомобиль в АВАРИИ

one_owner - принадлежал ли автомобиль одному лицу

personal_use_only - использовался ли автомобиль только в личных целях

seller_name - имя продавца

seller_rating - рейтинг продавца

driver_rating - рейтинг автомобиля, данный водителями

driver_reviews_num - количество отзывов об автомобилях, оставленных водителями

price_drop - снижение цены по сравнению с начальной ценой

price - цена автомобиля

Выбор признаков для машинного обучения

Для машинного обучения выберем целевой признак - стоимость автомобиля. Сопоставим с остальными признаками, а именно, характеристики и конфигурации автомобиля выявляем примерную стоимость автомобиля.

4. Листинг

4.1.Изучение данных

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 762091 entries, 0 to 762090
Data columns (total 20 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   manufacturer         762091 non-null  object
 1   model                762091 non-null  object
 2   year                 762091 non-null  int64
 3   mileage              761585 non-null  float64
 4   engine               747041 non-null  object
 5   transmission         752187 non-null  object
 6   drivetrain           740529 non-null  object
 7   fuel_type            739164 non-null  object
 8   mpg                  620020 non-null  object
 9   exterior_color       753232 non-null  object
 10  interior_color       705116 non-null  object
 11  accidents_or_damage  737879 non-null  float64
 12  one_owner            730608 non-null  float64
 13  personal_use_only    737239 non-null  float64
 14  seller_name          753498 non-null  object
 15  seller_rating        548118 non-null  float64
 16  driver_rating        730459 non-null  float64
 17  driver_reviews_num   762091 non-null  float64
 18  price_drop           410112 non-null  float64
 19  price                762091 non-null  float64
dtypes: float64(9), int64(1), object(10)
memory usage: 116.3+ MB
```

Здесь можно заметить, что в датасете содержатся единиц 762091 строк. А также имеют 3 различные типы: object, int64 и float64. В целях экономии памяти можно преобразовать в другие типы.

```
display(df.head())

display(df.tail())
```

	manufacturer	model	year	mileage	engine	transmission	drivetrain	fuel_type	mpg	exterior_color	interior_color	accidents_or_damage	one_owner	persona
0	Acura	ILX Hybrid 1.5L	2013	92945.0	1.5L I-4 i-VTEC variable valve control, engine...	Automatic	Front-wheel Drive	Gasoline	39-38	Black	Parchment	0.0	0.0	
1	Acura	ILX Hybrid 1.5L	2013	47645.0	1.5L I4 8V MPFI SOHC Hybrid	Automatic CVT	Front-wheel Drive	Hybrid	39-38	Gray	Ebony	1.0	1.0	
2	Acura	ILX Hybrid 1.5L	2013	53422.0	1.5L I4 8V MPFI SOHC Hybrid	Automatic CVT	Front-wheel Drive	Hybrid	39-38	Bellanova White Pearl	Ebony	0.0	1.0	
3	Acura	ILX Hybrid 1.5L	2013	117598.0	1.5L I4 8V MPFI SOHC Hybrid	Automatic CVT	Front-wheel Drive	Hybrid	39-38	Polished Metal Metallic	NaN	0.0	1.0	
4	Acura	ILX Hybrid 1.5L	2013	114865.0	1.5L I4 8V MPFI SOHC Hybrid	Automatic CVT	Front-wheel Drive	Hybrid	39-38	NaN	Ebony	1.0	0.0	

4.2.Преобразование данных

```
# Проверим объем занимаемой памяти в Мбайтах до преобразования
print(f'Объем датасета до преобразования: {df.memory_usage(deep=True).sum() / 1024 / 1024:.3f} Мбайт')
```

```
Объем датасета до преобразования: 561.306 Мбайт
```

```
original_memory = df.memory_usage(deep=True).sum()
```

```
# Автоматизируем
def change_type_variable(dateframe, show_print_report=False):
    for name_column in dateframe:
        if(dateframe[name_column].dtype  == 'int64'):
            dateframe[name_column] = dateframe[name_column].astype('int32')
            if(show_print_report):
                print(f'Успешно, преобразовали в другой тип INT32 колонки: {name_column}')
        if(dateframe[name_column].dtype  == 'float64'):
            dateframe[name_column] = dateframe[name_column].astype('float32')
            if(show_print_report):
                print(f'Успешно, преобразовали в другой тип FLOAT32 колонки: {name_column}')
        if(name_column in ['accidents_or_damage', 'one_owner', 'personal_use_only']):
            dateframe[name_column] = dateframe[name_column].astype(bool)
    if not(show_print_report):
        print('Успешно, преобразованы в другой тип')
```

```
# Преобразуем их
change_type_variable(df)
```

```
Успешно, преобразованы в другой тип
```

```
# Проверим объем занимаемой памяти в Мбайтах до преобразования
print(f'Объем датасета после преобразования: {df.memory_usage(deep=True).sum() / 1024 / 1024:.3f} Мбайт')
```

```
Объем датасета после преобразования: 525.694 Мбайт
```

```
optimized_memory = df.memory_usage(deep=True).sum()
```

```
# Узнаем, сколько сэкономили памяти
savings_percentage = (original_memory - optimized_memory) / original_memory * 100
print(f"Сэкономлено {savings_percentage:.2f}% памяти")
```

```
Сэкономлено 6.34% памяти
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 762091 entries, 0 to 762090
Data columns (total 20 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   manufacturer         762091 non-null  object
 1   model                762091 non-null  object
 2   year                 762091 non-null  int32
 3   mileage              761585 non-null  float32
 4   engine               747041 non-null  object
 5   transmission         752187 non-null  object
 6   drivetrain           740529 non-null  object
 7   fuel_type            739164 non-null  object
 8   mpg                  620020 non-null  object
 9   exterior_color       753232 non-null  object
 10  interior_color       705116 non-null  object
 11  accidents_or_damage  762091 non-null  bool
 12  one_owner            762091 non-null  bool
 13  personal_use_only    762091 non-null  bool
 14  seller_name          753498 non-null  object
 15  seller_rating        548118 non-null  float32
 16  driver_rating        730459 non-null  float32
 17  driver_reviews_num   762091 non-null  float32
 18  price_drop           410112 non-null  float32
 19  price                762091 non-null  float32
dtypes: bool(3), float32(6), int32(1), object(10)
memory usage: 80.7+ MB
```

Рассмотрим описательную статистику

4.3.Описательная статистика

```
df.describe()
```

	year	mileage	seller_rating	driver_rating	driver_reviews_num	price_drop	price
count	762091.000000	7.615850e+05	548118.000000	730459.000000	762091.000000	410112.000000	7.620910e+05
mean	2017.791398	5.578170e+04	4.158568	4.623524	89.550911	1007.467163	3.648899e+04
std	5.110532	4.355788e+04	0.805741	0.276902	115.082268	1375.122192	1.984183e+06
min	1915.000000	0.000000e+00	1.000000	1.000000	0.000000	100.000000	1.000000e+00
25%	2016.000000	2.328700e+04	3.800000	4.500000	14.000000	380.000000	1.958300e+04
50%	2019.000000	4.559600e+04	4.500000	4.700000	51.000000	642.000000	2.798900e+04
75%	2021.000000	7.836500e+04	4.700000	4.800000	119.000000	1007.000000	3.948800e+04
max	2024.000000	1.119067e+06	5.000000	5.000000	1025.000000	170995.000000	1.000000e+09

Здесь стоит обратить внимание на следующие колонки:

Год. Мы чаще всего можем увидеть автомобиль, которая продается около 2017 года (среднее значение преобладает остальных). В объявлениях можем увидеть автомобиль с 1915 года.

Пройденный миль. В объявлениях чаще всего выставляют автомобиль с 5.58^4 милях. Нелья не отрицать, что в продажах выставляют автомобиль, которая ни разу не проехала. Существует автомобиль, которая проехала 1,11^6 миль.

Наличие авария автомобили. Статистика говорит, что в объявлениях редко указывают, что автомобиль попадает в аварию. Мы можем сталкиваться с автомобилей, у которой была авария, с вероятностью около 22%.

Одно лицо у автомобилей. Эта колонка говорит о том, что у этой автомобилей было только одно лицо - водитель. Если да, то одно лицо, в противном случае несколько лиц было у этой автомобили. Статистика говорит, что в среднем мы сталкиваемся с автомобилей, у которой было несколько лиц.

Пользование в личных целях. Статистика говорит, что чаще всего пользуются автомобилей в личных целях, около 65%.

Рейтинг продавца. Продавец в среднем чаще всего выставляют автомобиль с рейтингом 4.15, а самой минимальной - 1.00.

Рейтинг водителя. Водитель в среднем чаще всего выставляют автомобиль с рейтингом 4.62, а самой минимальной - 1.00.

Количество отзывов об автомобилях, оставленных водителями. Водитель в среднем чаще всего выставляют автомобиль с рейтингом 4.62, а самой минимальной - 1.00.

Снижение цены по сравнению с начальной ценой. В среднем мы можем увидеть в объявлениях, что продают автомобилей с 1007 долларов, а самой максимальной - 170995 долларов, минимальной - 100 долларов.

Цена автомобиля. В среднем мы можем увидеть в объявлениях, что продают автомобилей с 3.64^4 долларов, а самой максимальной - 10 00 000 000 долларов, минимальной - 1 долларов. Интересно узнать, какие же автомобили же.

В датасете содержатся широкий диапазон промежутков года автомобилей, начиная с 1915 по 2024 года. Нельзя не отрицать, что в объявлениях выставляют продажи раритетных автомобилей, что было обусловлено высокой стоимостью. Также в объявлениях вытавляют автомобилей с большими пробегами, которая нуждается в технических ремонтах, не говоря уж о несколько лиц у этой автомобилей. Скорее всего в объвлениях выставляют служебные автомобили: фургоны, пикапы, т.к. процент пользования в личных целях невысок (около 65%). Самое удивительное, что продавцы оставили отзыв автомобиля ниже по сравнению с отзывом водителей. Поэтому отсюда следует причина - сильное понижение цены по сравнению с начальной стоимостью автомобилей.

4.4.Предобработка данных

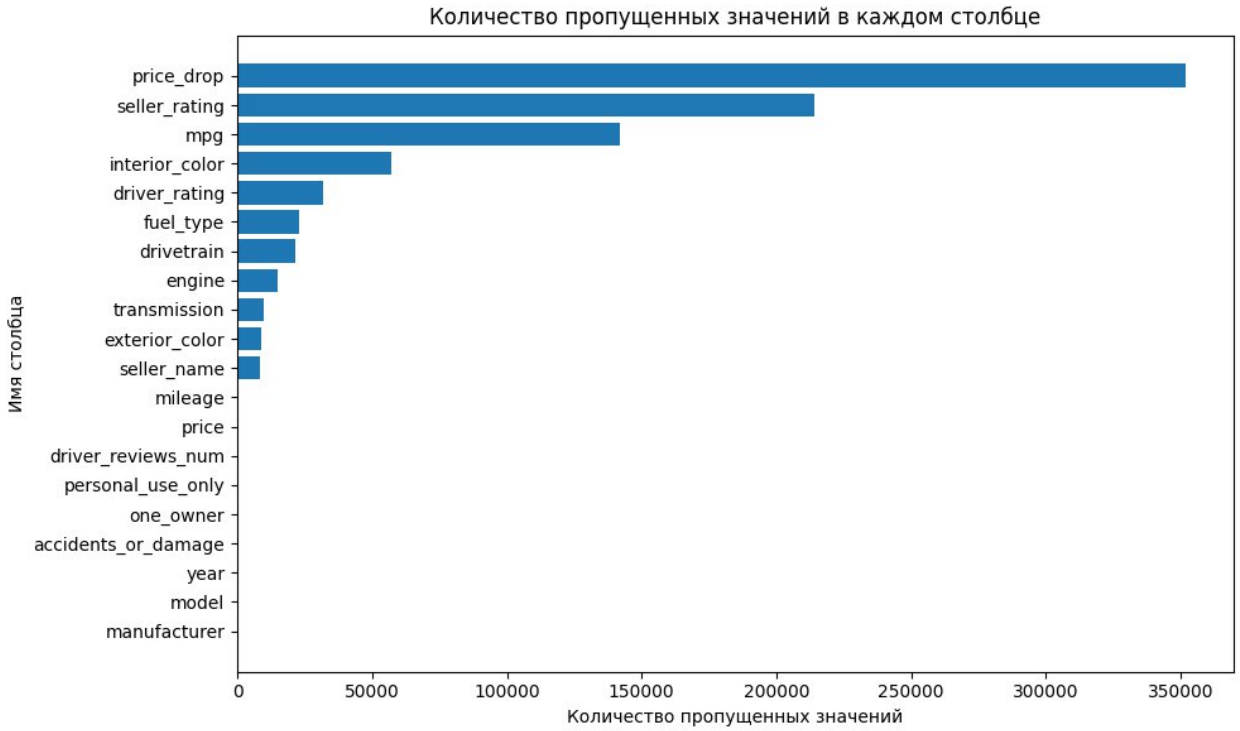
4.4.1. Пропущенные значения

```
# Создаем список с именами столбцов и количеством пропущенных значений
columns = df.columns
missing_counts = [df[column].isnull().sum() for column in columns]

# Сортируем столбцы в порядке убывания количества пропущенных значений
sorted_columns, sorted_missing_counts = zip(*sorted(zip(columns, missing_counts), key=lambda x: x[1], reverse=False))

# Создаем горизонтальную столбчатую диаграмму
plt.figure(figsize=(10, 6))
# Используем barh для горизонтальных столбцов
plt.barh(sorted_columns, sorted_missing_counts)
plt.xlabel('Количество пропущенных значений')
plt.ylabel('Имя столбца')
plt.title('Количество пропущенных значений в каждом столбце')
plt.tight_layout()

# Отображаем график
plt.show()
```



Как видим, что присутствуют огромные пропуски в столбцах: сниженная стоимость, рейтинг продаца и кол-во миль. Заполним пропуски медианными значениями только для численных типов, а остальных - устраним.

Медианными значениями заполняем, потому что они менее чувствительны к выбросам.

```
columns_isnull = [col for col, count in zip(sorted_columns, sorted_missing_counts) if count > 0]
print(f'Названий столбцов, у которых пропуски:')
for col in columns_isnull:
    print('\t' + col)
```

```
Названий столбцов, у которых пропуски:
        mileage
        seller_name
        exterior_color
        transmission
        engine
        drivetrain
        fuel_type
        driver_rating
        interior_color
        mpg
        seller_rating
        price_drop
```

```
# Создадим функцию, который будет автоматически выводит кол-во пропусков,
# находит медианное значение, а также заполнит его
def fill_isnull_median(df, columns_isnull):
    print(f'Кол-во пропусков {df[columns_isnull].isnull().sum()}')
    print(f'Тип колонки: {df[columns_isnull].dtype}')
    if(df[columns_isnull].dtype == 'float32'):
        print(f'Медианное значение {df[columns_isnull].median()}')
        # Заполняем пропуски медианным значением
        df[columns_isnull] = df[columns_isnull].fillna(df[columns_isnull].median())
        print('Заполнен пропуск')
    else:
        df = df[~df[columns_isnull].isnull()]
        print('Устранен пропуск')

    return df
```

```
for col in columns_isnull:
    print(f'Колонка: {col}')
    df = fill_isnull_median(df, col)
    print()
```

```
Колонка: mileage
Кол-во пропусков 506
Тип колонки: float32
Медианное значение 45596.0
Заполнен пропуск

Колонка: seller_name
Кол-во пропусков 8593
Тип колонки: object
Устранен пропуск

Колонка: exterior_color
Кол-во пропусков 8859
Тип колонки: object
Устранен пропуск

Колонка: transmission
Кол-во пропусков 9585
```

```
# Проверим еще раз
for col in columns_isnull:
    print(f'Колонка: {col}; \t Кол-во пропусков: {df[col].isnull().sum()}')
```

```
Колонка: mileage;        Кол-во пропусков: 0
Колонка: seller_name;    Кол-во пропусков: 0
Колонка: exterior_color;         Кол-во пропусков: 0
Колонка: transmission;   Кол-во пропусков: 0
Колонка: engine;         Кол-во пропусков: 0
Колонка: drivetrain;     Кол-во пропусков: 0
Колонка: fuel_type;      Кол-во пропусков: 0
Колонка: driver_rating;          Кол-во пропусков: 0
Колонка: interior_color;         Кол-во пропусков: 0
Колонка: mpg;    Кол-во пропусков: 0
Колонка: seller_rating;          Кол-во пропусков: 0
Колонка: price_drop;     Кол-во пропусков: 0
```

4.4.2. Дубликаты

```
# Кол-во дублирующие значения
df.duplicated().sum()
```

```
6212
```

Как видим, что присутствуют очень много дубликатов. Устраним их.

```
# Избавимся от них
df.drop_duplicates(inplace=True)
```

```
# Кол-во дублирующие значения
df.duplicated().sum()
```

```
0
```

4.4.3. Удаление выбросов

Перед удалением нужно заново рассмотреть описательную статистку, чтобы выявить наличие выбросов и устранить их.

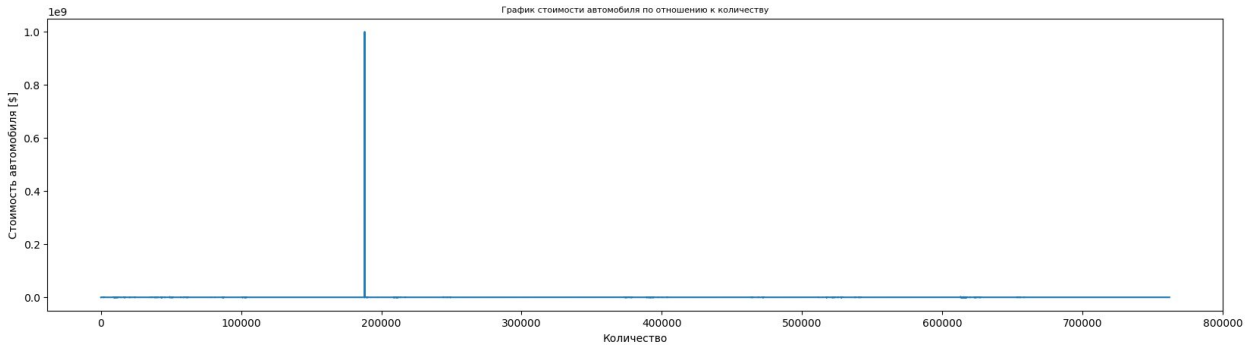
Описательная статистка

```
df.describe()
```

	year	mileage	seller_rating	driver_rating	driver_reviews_num	price_drop	price
count	563883.000000	5.638830e+05	563883.000000	563883.000000	563883.000000	563883.000000	5.638830e+05
mean	2017.855002	5.727803e+04	4.253538	4.642676	100.833824	791.907776	3.183164e+04
std	3.938123	4.128133e+04	0.702850	0.239818	119.788643	917.711304	1.331822e+06
min	1928.000000	0.000000e+00	1.000000	1.000000	0.000000	100.000000	1.000000e+00
25%	2016.000000	2.642700e+04	4.100000	4.600000	21.000000	500.000000	1.899800e+04
50%	2019.000000	4.787200e+04	4.500000	4.700000	62.000000	600.000000	2.648300e+04
75%	2020.000000	7.931400e+04	4.700000	4.800000	137.000000	750.000000	3.599800e+04
max	2023.000000	1.119067e+06	5.000000	5.000000	1025.000000	79909.000000	1.000000e+09

По описательной статистке видно, что есть выброс в стоимости автомобиля. Проверим на графике.

```
plt.figure(figsize=(20, 5))
plt.plot(df['price']);
plt.title('График стоимости автомобиля по отношению к количеству', fontsize=8);
plt.xlabel('Количество');
plt.ylabel('Стоимость автомобиля [$]');
```



Как и видим, устраним их.

```
df = df[df['price'] < df['price'].max()]
```

```
plt.figure(figsize=(20, 5));
plt.plot(df['price']);
plt.title('График стоимости автомобиля по отношению к количеству', fontsize=8);
plt.xlabel('Количество');
plt.ylabel('Стоимость автомобиля [$]');
```

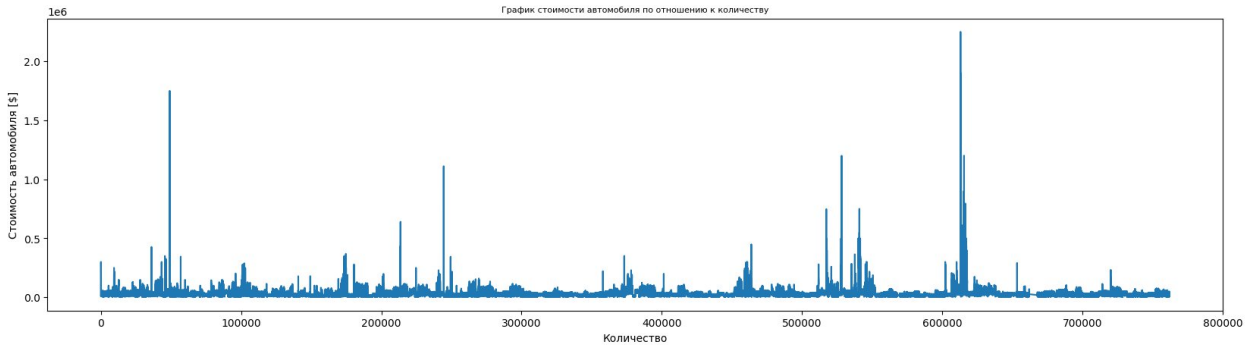


График получился более менее адекватным.

4.4.4. Преобразование в численный тип

У нас есть колонка не численного типа, которого нужно преобразовать в другой тип.

```
df['mpg'].head()
```

```
0    39-38
1    39-38
2    39-38
5    39-38
6    39-38
Name: mpg, dtype: object
```

```
df['mpg'].value_counts()
```

```
mpg
19-26      14733
18-25      14514
20-27      12373
17-25      11790
16-23      10518
           ...
29-42          1
46-41          1
49-45          1
48-47.5        1
19-27.5        1
Name: count, Length: 841, dtype: int64
```

```
# Разделение диапазонов mpg на две колонки
df[['mpg_city', 'mpg_highway']] = df['mpg'].str.split('-', expand=True)

# Преобразование колонок в числовой формат
# Флаг 'errors='coerce'' позволит обработать случаи,
# когда значения не могут быть преобразованы в числа, и они будут заменены на NaN, если такие случаи есть.
df['mpg_city'] = pd.to_numeric(df['mpg_city'], errors='coerce')
df['mpg_highway'] = pd.to_numeric(df['mpg_highway'], errors='coerce')

# Удаление исходного столбца 'mpg', если нужно
df.drop('mpg', axis=1, inplace=True)
```

4.4.5. Преобразование цветов автомобилей по ключевому названию

Бесспорно, что существуют много разных типов цветов автомобилей. Можно через TF-IDF пропустить, чтобы составить ключевые цвета со списков разных цветов автомобилей со всего датасета.

```
df['exterior_color'].value_counts()
```

```
exterior_color
Black                            41359
White                            30529
Gray                             20116
Silver                           16745
Summit White                     11503
                                 ...
Red Flame                            1
Rescue Green Metallic Clearcoat      1
True Blue Pearl Coat                 1
Honolulu Blue                        1
Electric Silver                      1
Name: count, Length: 6183, dtype: int64
```

```
df['interior_color'].value_counts()
```

```
interior_color
Black             229548
Gray               40341
Jet Black          37543
Ebony              33895
Charcoal           21526
                   ...
Mocha  leather         1
Gray - Gray            1
MOCHA                  1
Beige  leather         1
Blondlrettech          1
Name: count, Length: 3627, dtype: int64
```

Как и видим, что много разных цветов и причем с разными регистрами. Выделим конкретные уникальные цвета.

```
# Подберем список слов цветов для замены
selected_colors = ['black', 'white', 'gray', 'silver', 'red', 'green',
                   'yellow', 'blue', 'orange', 'purple', 'brown', 'pink', 'beige', 'cyan', 'olive', 'maroon', 'navy', 'teal']

# Заменяет значения в столбце 'exterior_color' на основе ключевых слов.
def replace_keywords(df, col_name, selected_colors):

    # Приведение к нижнему регистру
    df[col_name] = df[col_name].str.lower()

    # Замена значений на основе частичного совпадения
    for color in selected_colors:
        mask = df[col_name].notna() & df[col_name].str.contains(color, case=False, na=False)
        df.loc[mask, col_name] = color

    return df
```

```
%%time
# Замена значений на основе ключевых слов
df = replace_keywords(df, 'exterior_color', selected_colors)
df = replace_keywords(df, 'interior_color', selected_colors)
```

```
CPU times: user 15.9 s, sys: 38.8 ms, total: 15.9 s
Wall time: 15.9 s
```

Посмотрим, сколько уникальных значений получили в итоге.

```
df['exterior_color'].value_counts()
```

```
exterior_color
white              124259
black              123311
silver              70733
gray                59562
blue                50767
                     ...
moonlight pearl         1
nice and clean          1
chard                   1
royal ruby              1
osmium                  1
Name: count, Length: 1875, dtype: int64
```

```
df['interior_color'].value_counts()
```

```
interior_color
black                              313221
gray                                55923
ebony                               33897
beige                               22235
charcoal                            21536
                                     ...
vintage tan/ebony/ebony/vintag          1
brandy/ dark atmosphere                 1
cocoa w/shale accents                   1
vin tan/eb/eb/vin tan/eb                1
blondlrettech                           1
Name: count, Length: 1350, dtype: int64
```

Уникальных цветов довольно много, можно отсеивать их для машинное обучение.

4.4.6. Добавление новых фич для машинного обучения

4.4.6.1. Разница в продажах

Рассмотрим разницу в продажах, следуя по формуле:

$$X_{\text{price difference}} = Y_{\text{price}} - Y_{\text{price drop}}$$

```
df['price_difference'] = df['price'] - df['price_drop']
```

Рассмотрим кол-во отрицательных значений

```
print('Количество:', df[df['price_difference'] < 0]['price_difference'].count())
```

```
Количество: 20
```

Можно сказать, что "сниженная стоимость" продавец по какой то причине стал увеличить стоимость автомобиля, рассмотрим ниже в сортировке по возрастанию разницы стоимости.

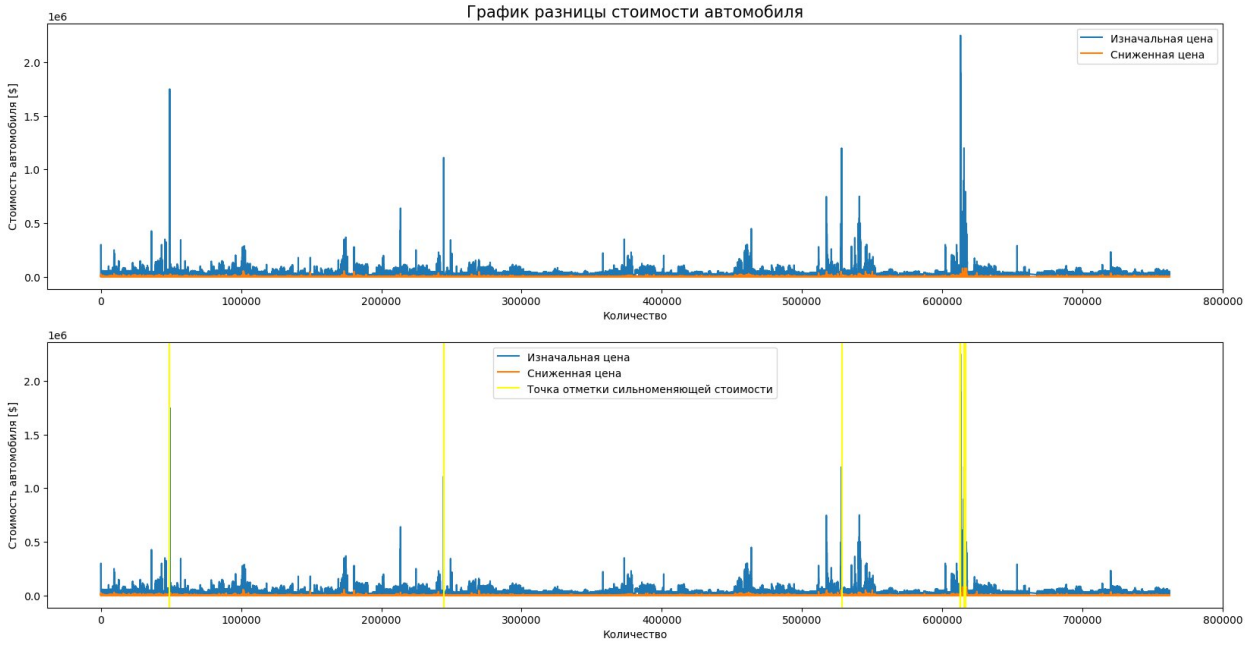
```
df[df['price_difference'] < 0][['model', 'year', 'price_difference', 'price', 'price_drop']] \
    .sort_values(by='price_difference')
```

	model	year	price_difference	price	price_drop
566508	Maxima SE	2001	-2056.0	500.0	2556.0
384190	Liberty Sport	2007	-1700.0	1600.0	3300.0
425094	Sorento LX	2011	-993.0	3500.0	4493.0
660696	Legacy 2.5i Limited	2019	-599.0	1.0	600.0
5658	TLX V6 A-Spec	2018	-599.0	1.0	600.0
13850	A4 2.0T Premium	2014	-599.0	1.0	600.0
555602	Mirage ES	2017	-510.0	3995.0	4505.0
440689	Sedona LX	2008	-500.0	3500.0	4000.0
584638	Versa SV	2021	-341.0	259.0	600.0
195371	EcoSport Titanium	2019	-311.0	289.0	600.0
66049	Enclave Premium	2017	-301.0	299.0	600.0
736374	Jetta S	2019	-281.0	319.0	600.0
162360	Pacifica Touring L	2020	-201.0	399.0	600.0
165200	Grand Caravan GT	2019	-201.0	399.0	600.0
680496	C-HR LE	2019	-201.0	399.0	600.0
553053	Outlander Sport SE	2021	-201.0	399.0	600.0
131167	Avalanche 1500	2004	-195.0	4900.0	5095.0
549313	S-Class S500	2001	-12.0	4988.0	5000.0
587396	NV200 S	2016	-10.0	19990.0	20000.0
70844	LeSabre Custom	2003	-1.0	1999.0	2000.0

```
fig, axs = plt.subplots(2, 1, figsize=(20, 10));

axs[0].plot(df['price']);
axs[0].plot(df['price_drop']);
axs[0].set_title('График разницы стоимости автомобиля', fontsize=15);
axs[0].set_xlabel('Количество');
axs[0].set_ylabel('Стоимость автомобиля [$]');
axs[0].legend(['Изначальная цена', 'Сниженная цена'])

axs[1].plot(df['price']);
axs[1].plot(df['price_drop']);
for max_value in df['price'].sort_values(ascending=False).head(15).index:
    axs[1].axvline(x=max_value, color='yellow');
axs[1].set_xlabel('Количество');
axs[1].set_ylabel('Стоимость автомобиля [$]');
axs[1].legend(['Изначальная цена', 'Сниженная цена', 'Точка отметки сильноменяющей стоимости']);
```



По этой графике можно заметить (желтая вертикальная линия), что в некоторых местах изначальная стоимость автомобиля была высокой, а потом сильно опустила вниз.

4.4.6.2. Разница в рейтингах, оставленных водителями и продавцами

Рассмотрим разницу в рейтингах, следуя по формуле:

$$X_{\text{rating difference}} = Y_{\text{driver rating}} - Y_{\text{seller rating}}$$

```
df['rating_difference'] = df['driver_rating'] - df['seller_rating']
```

```
print('Количество отрицательных значений:', df[df['rating_difference'] < 0]['rating_difference'].count())
```

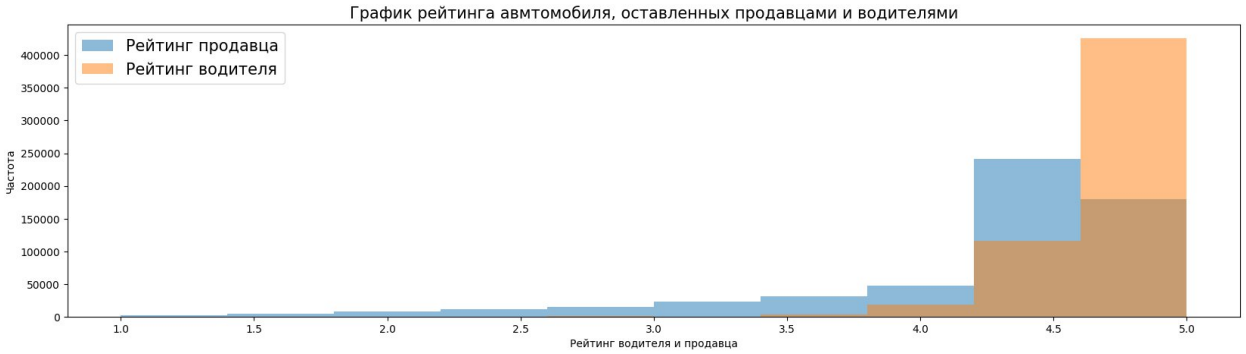
```
Количество отрицательных значений: 132807
```

Рассмотрим только ТОП-10 данных с отрицательными значениями

```
df[df['rating_difference'] < 0][['model', 'year', 'rating_difference']] \
    .sort_values(by='rating_difference').head(10)
```

	model	year	rating_difference
40663	M760 i xDrive	2021	-3.9
40635	M760 i xDrive	2021	-3.8
281777	Savana 3500 LT	2017	-3.8
279313	Savana 2500 Work Van	2020	-3.5
281804	Savana 3500 LS	2017	-3.5
281829	Savana 3500 LT	2017	-3.5
281830	Savana 3500 Work Van	2017	-3.5
40685	M760 i xDrive	2021	-3.5
40660	M760 i xDrive	2021	-3.5
279855	Savana 2500 Work Van	2020	-3.5

Как и здесь видим, что в какой-то причине рейтинг получилось сильно разным. Давайте рассмотрим график.



По графике видим, что видны существенные разницы рейтинг между водителями и продавцами. Однако, стоит обратить внимание, что водитель чаще всего оставляют высокий рейтинг нежели продавца.

4.5.Отсев до определенного кол-во уникальных значений

Для кодирования признаков ОНЕ или ОН будет черевато, если оставить много уникальных названия, потому что это приведет к созданию много новых признаков. Отсеиваем до небольших количеств, то есть сделаем так, чтобы создали максимум небольших новых закодированных признаков.

```
MAX_CONST_VALUE = 1000
```

```
# Сделаем копию датасета
df_copy = df.copy()
```

```
df_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 563882 entries, 0 to 762090
Data columns (total 23 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   manufacturer         563882 non-null  object
 1   model                563882 non-null  object
 2   year                 563882 non-null  int32
 3   mileage              563882 non-null  float32
 4   engine               563882 non-null  object
 5   transmission         563882 non-null  object
 6   drivetrain           563882 non-null  object
 7   fuel_type            563882 non-null  object
 8   exterior_color       563882 non-null  object
 9   interior_color       563882 non-null  object
 10  accidents_or_damage  563882 non-null  bool
 11  one_owner            563882 non-null  bool
 12  personal_use_only    563882 non-null  bool
 13  seller_name          563882 non-null  object
 14  seller_rating        563882 non-null  float32
 15  driver_rating        563882 non-null  float32
 16  driver_reviews_num   563882 non-null  float32
 17  price_drop           563882 non-null  float32
 18  price                563882 non-null  float32
 19  mpg_city             563882 non-null  int64
 20  mpg_highway          561469 non-null  float64
 21  price_difference     563882 non-null  float32
 22  rating_difference    563882 non-null  float32
dtypes: bool(3), float32(8), float64(1), int32(1), int64(1), object(9)
memory usage: 72.6+ MB
```

2.6.1. По цвету автомобилей

2.6.1.1. Цвет кузова - экстерьера

```
top_exterior_color = df_copy['exterior_color'].value_counts()[df_copy['exterior_color'].value_counts() >= MAX_CONST_VALUE].index
df_copy = df_copy[df_copy['exterior_color'].isin(top_exterior_color)]
```

2.6.1.2. Цвет салона - интерьера

```
top_interior_color = df_copy['interior_color'].value_counts()[df_copy['interior_color'].value_counts() >= MAX_CONST_VALUE].index
df_copy = df_copy[df_copy['interior_color'].isin(top_exterior_color)]
```

2.6.2. По типу трансмиссии

```
top_transmissions = df_copy['transmission'].value_counts()[df_copy['transmission'].value_counts() >= MAX_CONST_VALUE].index
df_copy = df_copy[df_copy['transmission'].isin(top_transmissions)]
```

2.6.3. По цепями привода (передний, задний)

```
top_drivetrain = df_copy['drivetrain'].value_counts()[df_copy['drivetrain'].value_counts() >= MAX_CONST_VALUE].index
df_copy = df_copy[df_copy['drivetrain'].isin(top_drivetrain)]
```

2.6.4. По типу топлива

```
top_fuel_type = df_copy['fuel_type'].value_counts()[df_copy['fuel_type'].value_counts() >= MAX_CONST_VALUE].index
df_copy = df_copy[df_copy['fuel_type'].isin(top_fuel_type)]
```

```
# Функция, которая разделяет численные и категориальные признаки
def divide_features(df):
    numerical_features = df.select_dtypes(include=['number']).columns
    categorical_features = df.select_dtypes(exclude=['number']).columns

    return numerical_features, categorical_features
```

```
numerical_features, categorical_features = divide_features(df_copy)
```

```
print("Численные признаки:", numerical_features.to_list())
```

```
Численные признаки: ['year', 'mileage', 'seller_rating', 'driver_rating', 'driver_reviews_num', 'price_drop', 'price', 'mpg_city', 'mpg_hig
hway', 'price_difference', 'rating_difference']
```

```
print("Нечисленные признаки:", categorical_features.to_list())
```

```
Нечисленные признаки: ['manufacturer', 'model', 'engine', 'transmission', 'drivetrain', 'fuel_type', 'exterior_color', 'interior_color', 'a
ccidents_or_damage', 'one_owner', 'personal_use_only', 'seller_name']
```

```
df_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 348668 entries, 13 to 762090
Data columns (total 23 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   manufacturer         348668 non-null  object 
 1   model                348668 non-null  object 
 2   year                 348668 non-null  int32  
 3   mileage              348668 non-null  float32
 4   engine               348668 non-null  object 
 5   transmission         348668 non-null  object 
 6   drivetrain           348668 non-null  object 
 7   fuel_type            348668 non-null  object 
 8   exterior_color       348668 non-null  object 
 9   interior_color       348668 non-null  object 
 10  accidents_or_damage  348668 non-null  bool   
 11  one_owner            348668 non-null  bool   
 12  personal_use_only    348668 non-null  bool   
 13  seller_name          348668 non-null  object 
 14  seller_rating        348668 non-null  float32
 15  driver_rating        348668 non-null  float32
 16  driver_reviews_num   348668 non-null  float32
 17  price_drop           348668 non-null  float32
 18  price                348668 non-null  float32
 19  mpg_city             348668 non-null  int64  
 20  mpg_highway          347354 non-null  float64
 21  price_difference     348668 non-null  float32
 22  rating_difference    348668 non-null  float32
dtypes: bool(3), float32(8), float64(1), int32(1), int64(1), object(9)
memory usage: 44.9+ MB
```

```
# Датасет получилось слишком объемным, отсеиваем 15%? чтобы было быстрее обучить модель
df_copy_fraction = df_copy.sample(frac=0.15, random_state=12345)
```

```
df_copy_fraction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 52300 entries, 569900 to 10010
Data columns (total 23 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   manufacturer         52300 non-null  object 
 1   model                52300 non-null  object 
 2   year                 52300 non-null  int32  
 3   mileage              52300 non-null  float32
 4   engine               52300 non-null  object 
 5   transmission         52300 non-null  object 
 6   drivetrain           52300 non-null  object 
 7   fuel_type            52300 non-null  object 
 8   exterior_color       52300 non-null  object 
 9   interior_color       52300 non-null  object 
 10  accidents_or_damage  52300 non-null  bool   
 11  one_owner            52300 non-null  bool   
 12  personal_use_only    52300 non-null  bool   
 13  seller_name          52300 non-null  object 
 14  seller_rating        52300 non-null  float32
 15  driver_rating        52300 non-null  float32
 16  driver_reviews_num   52300 non-null  float32
 17  price_drop           52300 non-null  float32
 18  price                52300 non-null  float32
 19  mpg_city             52300 non-null  int64  
 20  mpg_highway          52117 non-null  float64
 21  price_difference     52300 non-null  float32
 22  rating_difference    52300 non-null  float32
dtypes: bool(3), float32(8), float64(1), int32(1), int64(1), object(9)
memory usage: 6.7+ MB
```

4.6.Машинное обучение

[66]:
```
# Здесь будем сохранить результаты машинного обучения
results = pd.DataFrame()

# А это будет счетчтиком для нумерация моеделй
count_model = 0
```

4.6.1. Деление на обучающей и тестовой выборки

```
# Убираем лишние колонки для обучения
df_copy_fraction = df_copy_fraction.drop(['model', 'engine', 'seller_name'], axis=1)
```

```
# Получаем признак и цель
features = df_copy_fraction.drop('price', axis=1)
target = df_copy_fraction['price']
```

```
# Разделим обучающую, валидационную и тестовую выборку, потому что 60% обучащие выборки это приводит к ОЧЕНЬ ДОЛГОМУ ОБУЧЕНИЮ
# - 60% обучающей выборки (features_train, target_train)
# - 40% тестовой выборки (features_test, target_test)

# Разделяем данные на обучающую и остальные (валидационную и тестовую) выборки/
features_train, features_test, target_train, target_test = train_test_split(features, target, test_size=0.4, random_state=12345)
```

4.7. Кодирование признаков - прямое кодирование (One-Hot Encoding)

Закодируем признаки: manufacturer, transmission, drivetrain, fuel_type, exterior_color, interior_color, accidents_or_damage,one_owner, personal_use_only

```
# Выбираем категориальные признаки
categorical_features = ['manufacturer', 'transmission', 'drivetrain', 'fuel_type',
                        'exterior_color', 'interior_color', 'accidents_or_damage',
                        'one_owner', 'personal_use_only']
```

```
# Кодируем
encoder_ohe = OneHotEncoder(drop='first', handle_unknown='ignore', sparse_output=False)
```

```
# Обучаем энкодер на заданных категориальных признаках тренировочной выборки
encoder_ohe.fit(features_train[categorical_features])
```

```
OneHotEncoder
OneHotEncoder(drop='first', handle_unknown='ignore', sparse_output=False)
```

```
# Добавляем закодированные признаки в X_train_ohe
# Encoder_ohe.get_feature_names_out() позволяет получить названия колонок
features_train[encoder_ohe.get_feature_names_out()] = encoder_ohe.transform(features_train[categorical_features])
```

```
# Энкодером, который обучен на ТРЕНИРОВОЧНОЙ ВЫБОРКЕ, кодируем тестовую
features_test[encoder_ohe.get_feature_names_out()] = encoder_ohe.transform(features_test[categorical_features])
```

```
features_test[encoder_ohe.get_feature_names_out()] = encoder_ohe.transform(features_test[categorical_features])
```

```
# удаляем незакодированные категориальные признаки (изначальные колонки)
features_train = features_train.drop(categorical_features, axis=1)

features_test = features_test.drop(categorical_features, axis=1)
```

```
display(features_train.head())
display(features_test.head())
```

	year	mileage	seller_rating	driver_rating	driver_reviews_num	price_drop	mpg_city	mpg_highway	price_difference	rating_difference	...	interior_color_gra
645185	2015	75235.0	4.9	4.5	255.0	701.0	24	32.0	19797.0	-0.4	...	1
140604	2020	60831.0	3.3	4.8	55.0	500.0	29	36.0	18000.0	1.5	...	0
306119	2022	25148.0	4.7	5.0	4.0	600.0	19	26.0	36677.0	0.3	...	0
623627	2021	20756.0	4.5	4.8	99.0	2000.0	19	24.0	48998.0	0.3	...	0
194413	2020	7558.0	4.5	4.7	65.0	600.0	23	29.0	25998.0	0.2	...	0

5 rows × 95 columns

	year	mileage	seller_rating	driver_rating	driver_reviews_num	price_drop	mpg_city	mpg_highway	price_difference	rating_difference	...	interior_color_gra
397702	2016	68033.0	4.5	4.5	271.0	1000.0	21	28.0	17500.0	0.0	...	0
251856	2017	98966.0	4.5	4.8	353.0	3000.0	21	32.0	12995.0	0.3	...	0
48151	2019	37301.0	4.5	4.9	17.0	600.0	21	29.0	42288.0	0.4	...	0
499589	2019	42410.0	3.6	4.9	16.0	1087.0	26	34.0	25774.0	1.3	...	0
705714	2020	21169.0	4.6	4.7	237.0	1685.0	31	40.0	20842.0	0.1	...	1

5 rows × 95 columns

```
# Устраняем случайные и возможные пропуски для избежания ошибок
features_train = features_train.dropna()
target_train = target_train[features_train.index]

features_test = features_test.dropna()
target_test = target_test[features_test.index]
```

4.8. Обучение модели

4.8.1. KNeighborsRegressor

```
# Задаем значения гиперпараметров
parameters = {
    # Пример значений, можно добавить свои
    'kneighborsregressor__n_neighbors': [1, 2],
    # 'weights': ['uniform', 'distance'],
    'kneighborsregressor__weights': ['uniform'],
    # для параметра метрики (1 - манхэттенское расстояние, 2 - евклидово расстояние)
    # 'kneighborsregressor__p': [1]
}

# Инициализируем модель (включая масштабирование) и GridSearchCV
pipeline_scale = make_pipeline(StandardScaler(), KNeighborsRegressor())
model = GridSearchCV(pipeline_scale, param_grid=parameters, cv=5, scoring='neg_mean_absolute_error')
display(model)
```

```
GridSearchCV
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('standardscaler', StandardScaler()),
                                       ('kneighborsregressor',
                                        KNeighborsRegressor())]),
             param_grid={'kneighborsregressor__n_neighbors': [1, 2],
                         'kneighborsregressor__weights': ['uniform']},
             scoring='neg_mean_absolute_error')
estimator: Pipeline
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('kneighborsregressor', KNeighborsRegressor())])
StandardScaler
StandardScaler()
KNeighborsRegressor
KNeighborsRegressor()
```

```
%%notify -m "KNeighborsRegressor OHE"
%%time

# Обучим модель на обучающей выборке
model.fit(features_train, target_train)
time = model.refit_time_
params = model.best_params_

# Узнаем MAE обучающей выборки
result_MAE_t = -model.best_score_
print('MAE TRAIN:', result_MAE_t)
print('TIME TRAIN [s]:', round(time, 2))
```

```
MAE TRAIN: 5337.94921875
TIME TRAIN [s]: 0.07
CPU times: user 18.6 s, sys: 17.1 ms, total: 18.6 s
Wall time: 16.5 s
<IPython.core.display.Javascript object>
```

Проверка на тестовой выборки

```
%%time
start_time = timeit.default_timer()
# Получим предсказания на тестовой выборки
predictions = model.predict(features_test)

elapsed = round(timeit.default_timer() - start_time, 3)

# Узнаем MAE
result_MAE_v = mean_absolute_error(target_test, predictions)
print('MAE TEST:', result_MAE_v)
print('Предсказание:', predictions.mean())
```

```
MAE TEST: 5118.9834
Предсказание: 29884.068
CPU times: user 6.84 s, sys: 5.54 ms, total: 6.85 s
Wall time: 6.85 s
```

```
# Зафиксируем результаты
results[count_model] = pd.Series({
    'NAME': pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__,
    'MAE TRAIN': result_MAE_t,
    'MAE TEST': result_MAE_v,
    'PREDICTIONS': predictions.mean(),
    'TIME TRAINING [s]': model.refit_time_,
    'TIME PREDICTION [s]': elapsed,
    'PARAMETRS': model.best_params_
})

display(results[count_model])
count_model+=1
```

```
NAME                                                KNeighborsRegressor
MAE TRAIN                                                   5337.949219
MAE TEST                                                    5118.983398
PREDICTIONS                                                29884.068359
TIME TRAINING [s]                                              0.066248
TIME PREDICTION [s]                                               6.847
PARAMETRS             {'kneighborsregressor__n_neighbors': 2, 'kneig...
Name: 0, dtype: object
```

4.9.Итог

4.9.1. Анализ моделей

```
%%notify -m "Total result"
results = pd.DataFrame(results).T
```

```
<IPython.core.display.Javascript object>
```

```
results
```

	NAME	MAE TRAIN	MAE TEST	PREDICTIONS	TIME TRAINING [s]	TIME PREDICTION [s]	PARAMETRS
0	KNeighborsRegressor	5337.949219	5118.983398	29884.068359	0.066248	6.847	{'kneighborsregressor__n_neighbors': 2, 'kneig...