Классификация текстов на реальных данных.

Данный ноутбук берёт за основу мою лабораторную работу по дисциплине NLP.
Сделаны лишь незначительные корректировки.

Сбор данных

Данные для классификации текстов были взяты с маркетплейса https://www.wildberries.ru/. Для этого был разработан парсер, в котором используются библиотеки `requests` и `pydantic`. Данные на странице сайта подгружаются динамически, используя запросы к API микросервисов(например, к серверам с отзывами, к каталогам товаров и т.п.). С помощью функций библиотеки `requests` делаются запросы к серверам wildberries, после чего ответ парсится по заданным моделям данных и их атрибутами, указанными в этих моделях, наследованных от базового класса библиотеки `pydantic`.

В файле `models.py` хранятся модели данных, которые подтягиваются их ответа серверов в формате json, в файле `parser.py` реализован класс парсера.

Принцип работы парсера следующий:

- Инициализируется класс парсера с ссылкой на какой-либо товар с торговой площадки
- Из ссылки при помощи библиотеки регулярных выражений re парсится id товара
- Делается запрос к северу wildberries, с параметром, полученным на предыдущем шаге, и из запроса извлекается id продавца, товары которого будут потом парситься в цикле
- Создаётся файл csv с колонками `название`, `бренд`, `цена`, `скидка в %`, `цена со скидкой`, `кол-во фото в карточке`, `плюсы`, `минусы`, `наличие фото в отзыве`, `полезно`, `неполезно`, `текст отзыва`, `рейтинг`
- До тех пор, пока сервер возвращает не пустой json-ответ, делаем запрос к нему, получая список товаров продавца постранично, с полями `название`, `бренд`, `цена`, `скидка в %`, `цена со скидкой`, `кол-во фото в карточке`
- Получая список товаров, делаем запрос к серверу с обратной связью - отзывами к одному товару из списка, обрабатываем полученные данные и заносим их в csv файл

В качестве ссылок на товары я случайным образом выбрал товары с площадки из разных категорий разных продавцов, чтобы сделать выборку репрезентативной.

Импортируем заранее все нужные библиотеки

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

import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from wordcloud import WordCloud, STOPWORDS
from pymorphy3 import MorphAnalyzer

from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import classification_report, accuracy_score


%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ibasl\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Очистка данных и описательная статистика

Прочитаем данные, полученные в результате парсинга. Переименуем исходные названия, чтобы не было путаницы во время кодирования текста.

In [2]:
```
df = pd.read_csv("../../data/dataset.csv")
df.rename(columns={
    "название": "name",
    "бренд": "brand",
    "цена": "price",
    "скидка в %": "sale",
    "цена со скидкой": "price_with_sale",
    "кол-во фото в карточке": "pics",
    "плюсы": "pros",
    "минусы": "cons",
    "наличие фото в отзыве": "hasPhoto",
    "полезно": "useful",
    "неполезно": "unuseful",
    "текст отзыва": "text",
    'рейтинг': 'target',},
    inplace=True)
df.head()
```

```
C:\Users\ibasl\AppData\Local\Temp\ipykernel_13912\805906506.py:1: DtypeWarning: Col
umns (8) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("../../data/dataset.csv")
```

Out[2]:

	name	brand	price	sale	price_with_sale	pics	pros	cons	hasPhoto	useful	unu
0	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	204.0	5	NaN	NaN	False	0	
1	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	204.0	5	NaN	NaN	False	0	
2	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	204.0	5	NaN	NaN	False	0	
3	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	204.0	5	NaN	NaN	False	0	
4	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	204.0	5	NaN	NaN	False	0	

Удалим дубликаты в датасете и посмотрим на размер датасета

In [3]:
```
df.drop_duplicates(inplace=True)
df.shape
```

Out[3]:
```
(381859, 13)
```

Посмотрим основную информацию о датасете

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 381859 entries, 0 to 461663
Data columns (total 13 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   name             381859 non-null  object 
 1   brand            381857 non-null  object 
 2   price            381859 non-null  float64
 3   sale             381859 non-null  int64  
 4   price_with_sale  381859 non-null  float64
 5   pics             381859 non-null  int64  
 6   pros             0 non-null       float64
 7   cons             0 non-null       float64
 8   hasPhoto         381033 non-null  object 
 9   useful           381859 non-null  int64  
 10  unuseful         381859 non-null  int64  
 11  text             380199 non-null  object 
 12  target           381859 non-null  int64  
dtypes: float64(4), int64(5), object(4)
memory usage: 40.8+ MB
```

Заменим тип колонки `hasPhoto` на булевый тип

In [5]:
```
df['hasPhoto'] = df['hasPhoto'].astype(bool)
```

Посмотрим описательную статистику числовых признаков

In [6]:
```
df.describe()
```

Out[6]:

	price	sale	price_with_sale	pics	pros	cons	useful	
count	381859.000000	381859.000000	381859.000000	381859.000000	0.0	0.0	381859.000000	38
mean	8620.201093	58.793353	3335.807565	6.952530	NaN	NaN	1.337533	
std	17832.819749	20.181132	6753.828420	4.154662	NaN	NaN	4.079684	
min	99.000000	0.000000	85.000000	1.000000	NaN	NaN	0.000000	
25%	1500.000000	59.000000	465.000000	4.000000	NaN	NaN	0.000000	
50%	3200.000000	66.000000	1288.000000	6.000000	NaN	NaN	0.000000	
75%	7030.000000	70.000000	3146.000000	9.000000	NaN	NaN	1.000000	
max	629990.000000	93.000000	239396.000000	26.000000	NaN	NaN	332.000000	

Посмотрим описательную статистику нечисловых признаков

In [7]:
```
df.describe(include=['O'])
```

Out[7]:

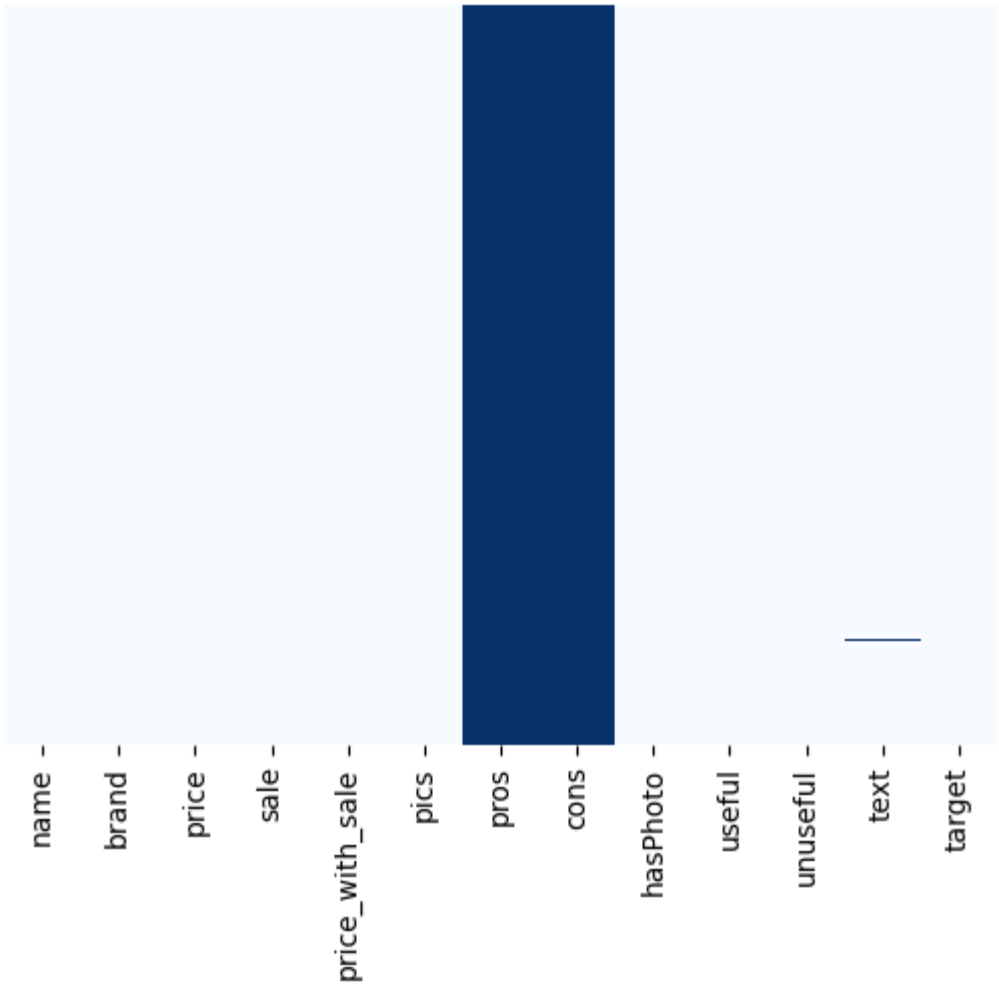
	name	brand	text
count	381859	381857	380199
unique	2012	102	201389
top	Ювелирные серьги женские из серебра 925	SOKOLOV	все отлично
freq	9322	178255	716

Посмотрим количество пропусков и визуализируем их

In [8]:
```
df.isna().sum()
```

Out[8]:
```
name                  0
brand                 2
price                 0
sale                  0
price_with_sale       0
pics                  0
pros             381859
cons             381859
hasPhoto              0
useful                0
unuseful              0
text               1660
target                0
dtype: int64
```

In [9]:
```
sns.heatmap(df.isna(), yticklabels=False, cbar=False, cmap='Blues');
```



Видно, что колонки плюсы и минусы содержат везде пустые значения, удалим их из датасета. Помимо этого удалим остальные встречающиеся пропуски, так как данных у нас очень много - 381 тысяча значений всего и 200 тысяч уникальных текстов.

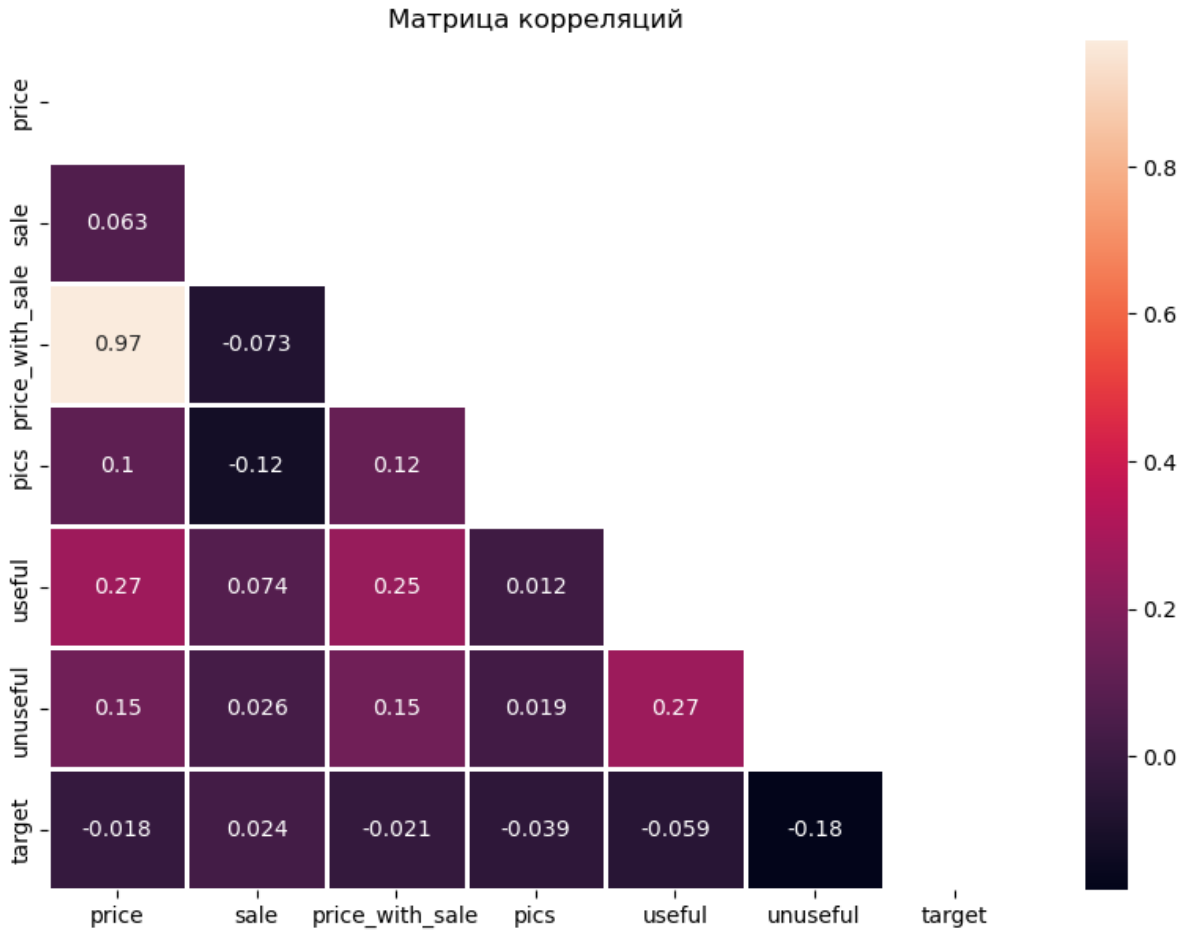
In [10]:
```
df.drop(columns=['pros', 'cons'], inplace=True)
df.dropna(inplace=True)
df.drop_duplicates(subset=['text'], inplace=True)
df.shape
```

Out[10]:
```
(201388, 11)
```

Посмотрим на корреляцию признаков. Моё предположение - столбцы с обычной ценой и ценой со скидкой коррелируются. Это логично, так как цену со скидкой можно получить линейными преобразованиями столбцов цена и скидка в % .

In [11]:
```
numeric_cols = df.select_dtypes(include=[np.number]).columns
df_ = df[numeric_cols]
corr = df_.corr()
mask = np.zeros_like(corr, dtype=np.bool_)
mask[np.triu_indices_from(mask)] = True

plt.figure(figsize = (10,7))
plt.title('Матрица корреляций')
sns.heatmap(corr,
            mask=mask,
            annot=True,
            fmt='.2g',
            linewidths=2);
```

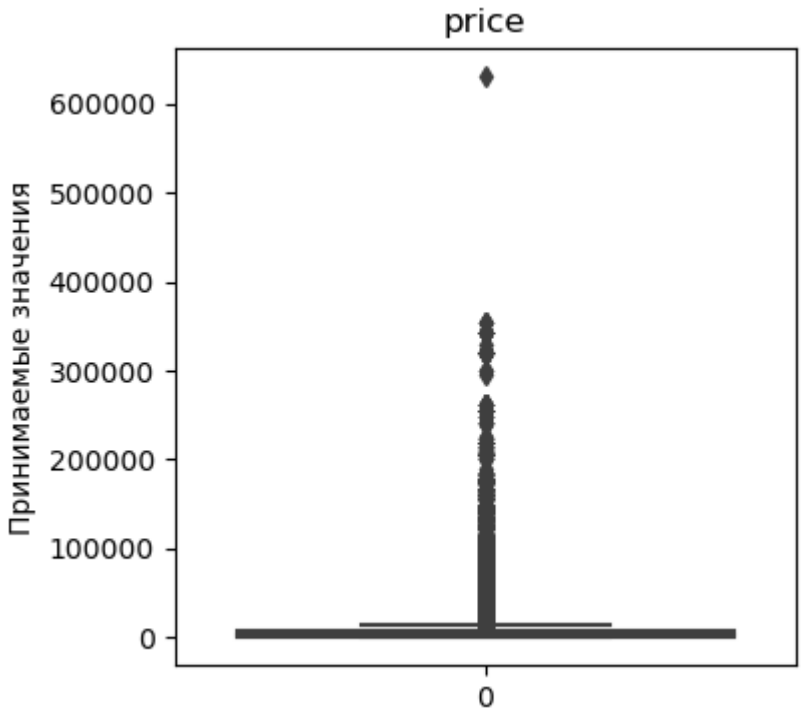


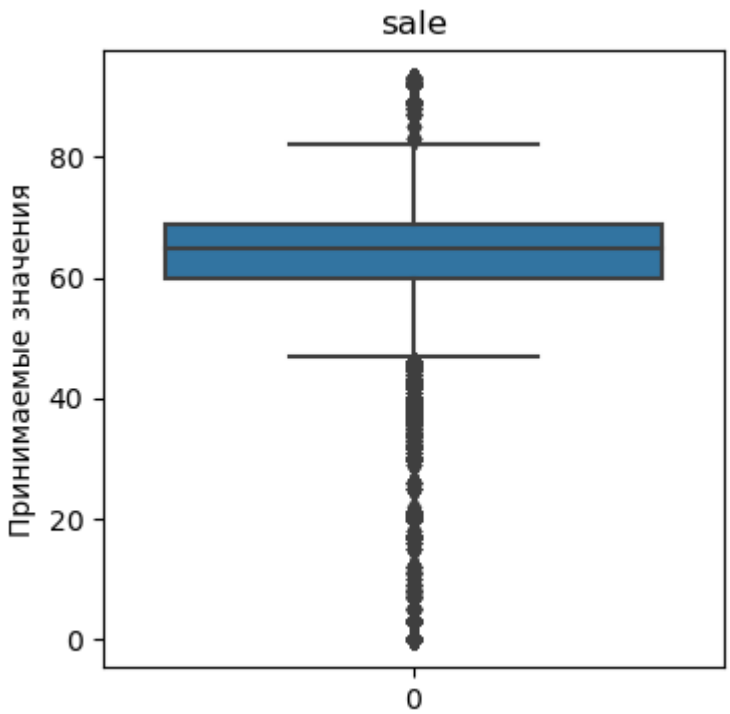
Действительно - цена со скидкой и цена коррелируют со значеним 0.97. Удалим столбец цена со скидкой .

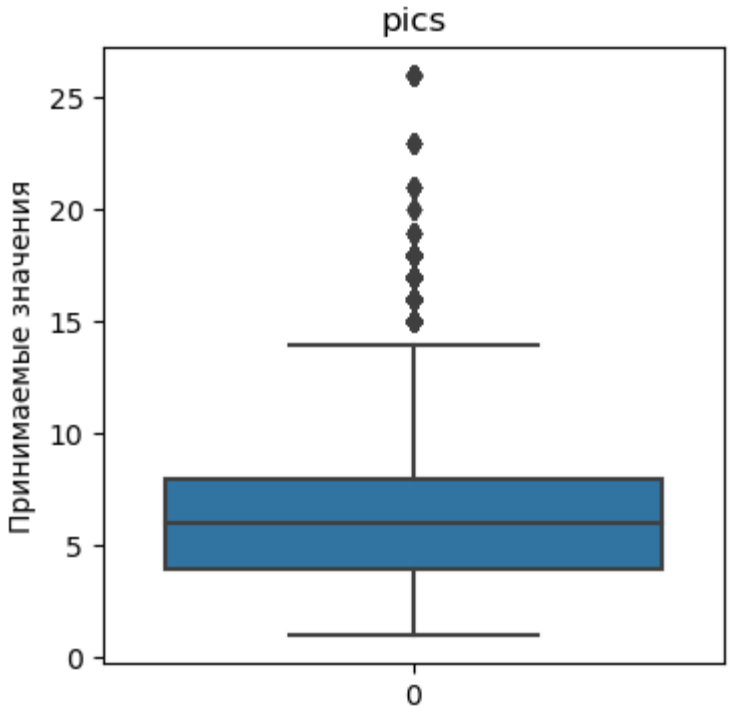
In [12]:
```
df.drop(columns=['price_with_sale'], inplace=True)
```

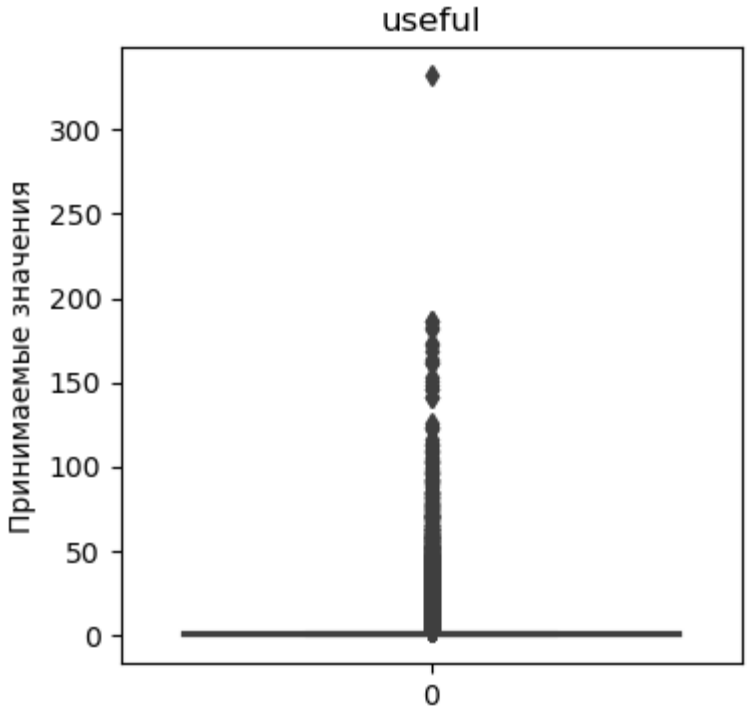
Посмотрим на выбросы в числовых данных при помощи графиков "ящик с усами"

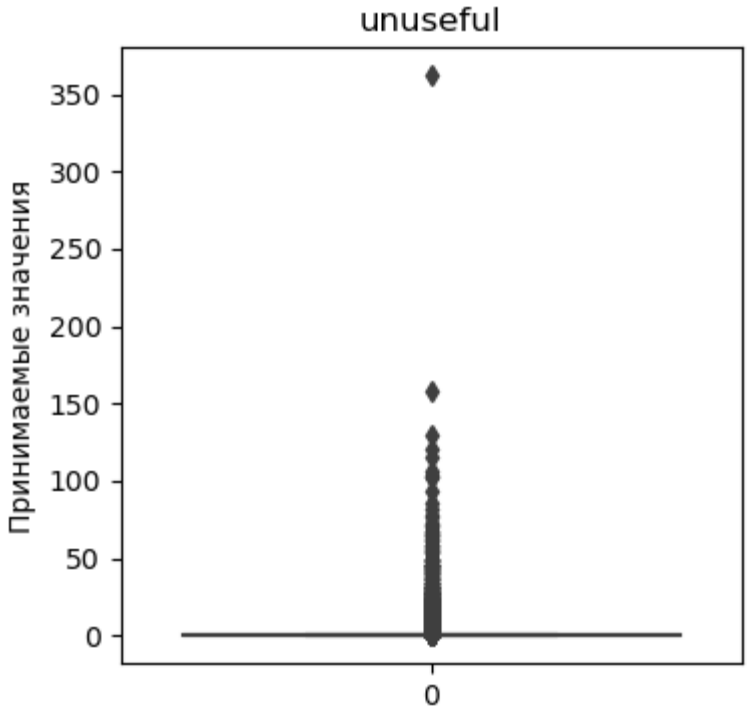
In [13]:
```
numeric_columns = df.select_dtypes(include=np.number).columns.tolist()
for col in numeric_columns:
    plt.figure(figsize=(4,4))
    sns.boxplot(data=df[col])
    plt.title(f'{col}')
    plt.ylabel('Принимаемые значения');
```

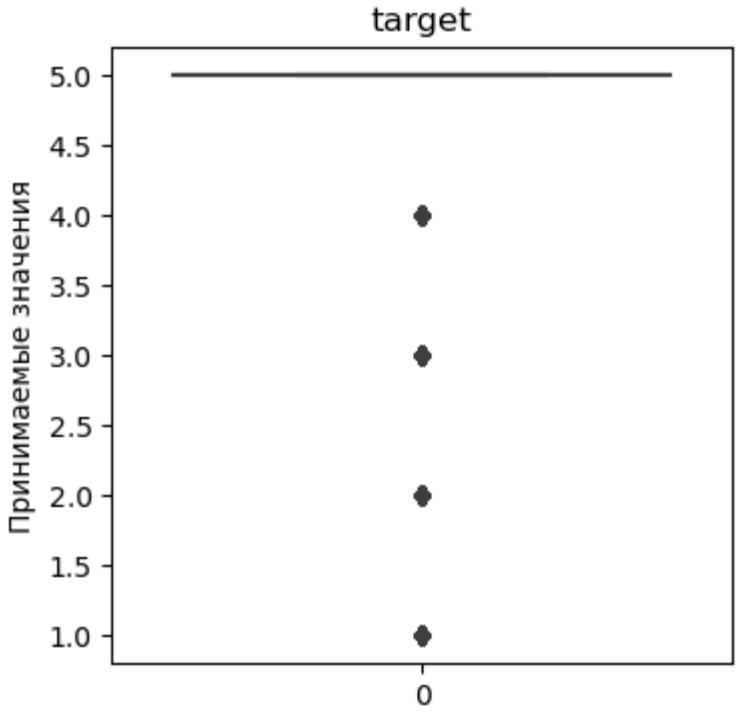





pics
Принимаемые значения
25
20
15
10
5
0
0


useful
Принимаемые значения
300
250
200
150
100
50
0
0





Имеет смысл удалить немного выбросов в колонках полезно и неполезно , а также удалить большую часть выбросов в столбце цена , опираясь на размах значений.

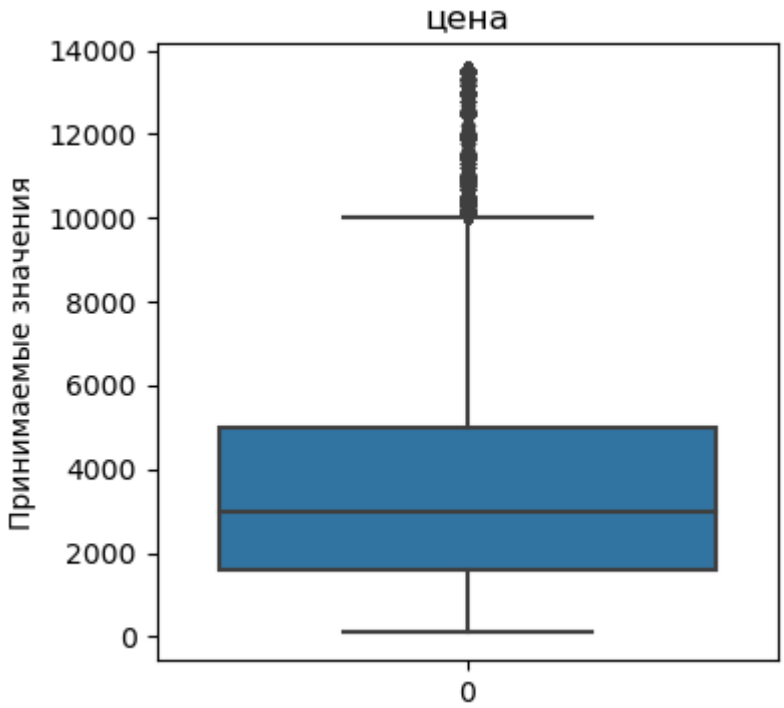
In [14]:
```
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df[(df['price'] >= lower_bound) & (df['price'] <= upper_bound)].shape
```

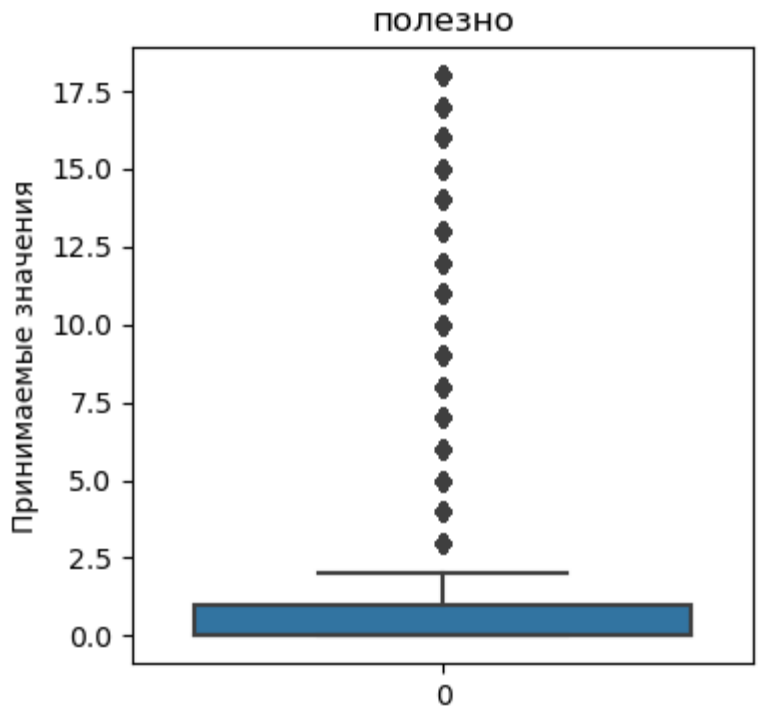
Out[14]:
```
(174487, 10)
```

In [15]:
```
plt.figure(figsize=(4,4))
sns.boxplot(data=df[(df['price'] >= lower_bound) & (df['price'] <= upper_bound)]['p
plt.title(f'цена')
plt.ylabel('Принимаемые значения');
```



Удалим примерно 12% выбросов по столбцу цена . Данных много, это некритично.

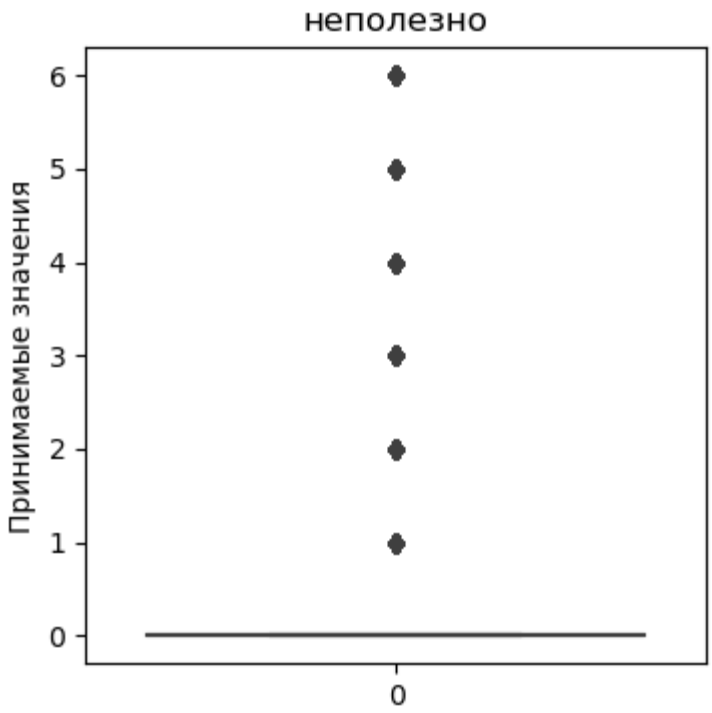
In [16]:
```
plt.figure(figsize=(4,4))
sns.boxplot(data=df['useful'][df['useful'] < df['useful'].quantile(0.99)])
plt.title(f'полезно')
plt.ylabel('Принимаемые значения');
```



In [17]:
```
df['useful'].value_counts()
```

Out[17]:
```
useful
0      112014
1       40038
2       17313
3        8874
4        5513
        ...
65          1
108         1
110         1
114         1
90          1
Name: count, Length: 127, dtype: int64
```

In [18]:
```
plt.figure(figsize=(4,4))
sns.boxplot(data=df['unuseful'][df['unuseful'] < df['unuseful'].quantile(0.99)])
plt.title(f'неполезно')
plt.ylabel('Принимаемые значения');
```



In [19]:
```
df['unuseful'].value_counts()
```

Out[19]:
```
unuseful
0      160961
1       24610
2        7409
3        3169
4        1610
        ...
103         1
63          1
82          1
47          1
72          1
Name: count, Length: 82, dtype: int64
```

Удалим по 1% в столбцах неполезно , полезно .

In [20]:
```
df = df[df['price'] < df['price'].quantile(0.85)]
df = df[df['unuseful'] < df['unuseful'].quantile(0.99)]
df = df[df['useful'] < df['useful'].quantile(0.99)]
df.shape
```

Out[20]:
```
(166779, 10)
```

Посмотрим на основную информацию очищенного датасета

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 166779 entries, 0 to 461608
Data columns (total 10 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   name      166779 non-null  object
 1   brand     166779 non-null  object
 2   price     166779 non-null  float64
 3   sale      166779 non-null  int64
 4   pics      166779 non-null  int64
 5   hasPhoto  166779 non-null  bool
 6   useful    166779 non-null  int64
 7   unuseful  166779 non-null  int64
 8   text      166779 non-null  object
 9   target    166779 non-null  int64
dtypes: bool(1), float64(1), int64(5), object(3)
memory usage: 12.9+ MB
```

Анализ предметной области и визуализация данных

In [22]:
```
df.head()
```

Out[22]:

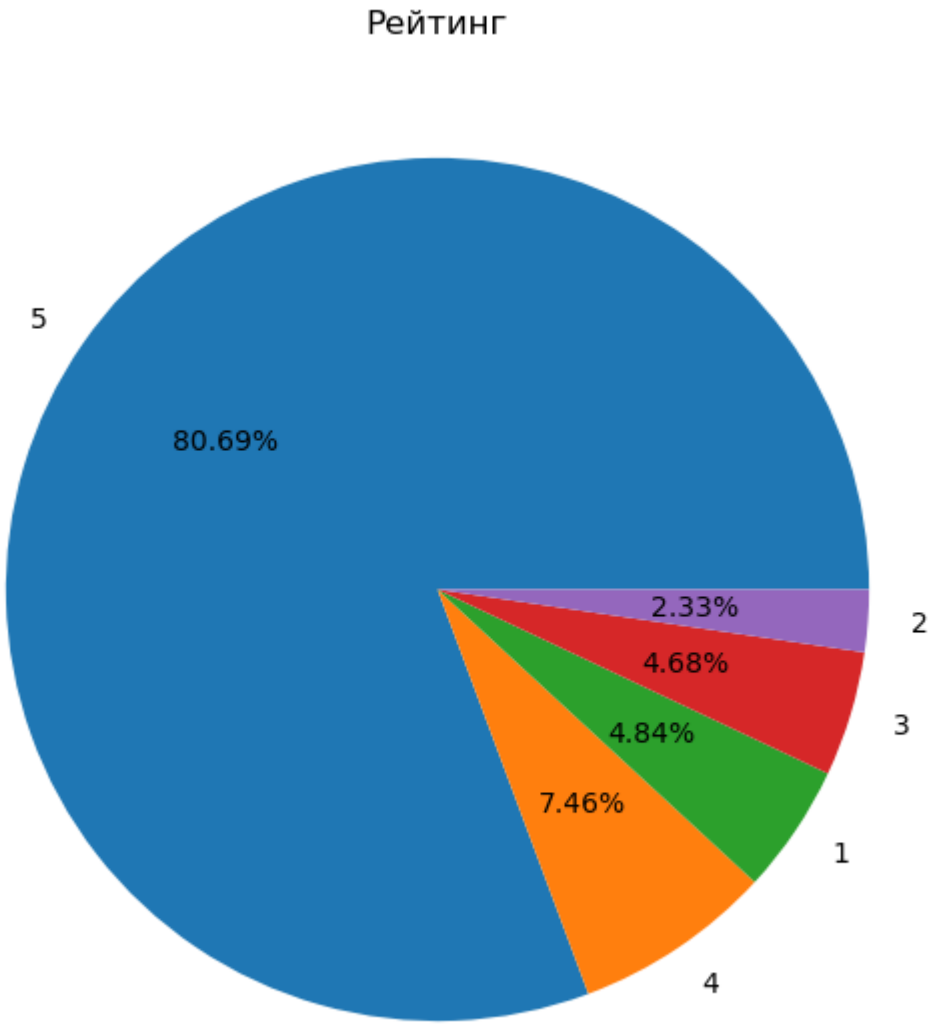
	name	brand	price	sale	pics	hasPhoto	useful	unuseful	text	target
0	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	5	False	0	1	отличные кусачки заточены хорошо со своей зада...	5
1	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	5	False	0	1	спасибо за качественный товар буду рекомендова...	5
2	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	5	False	0	0	щипчики хорошо стригут ногти все отлично	5
3	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	5	False	0	0	хороший набор пришло все целое	5
4	Клиппер маникюрный, кусачки для ногтей	Zebo Professional	499.0	59	5	False	0	0	получили кусочки пришли быстро и хорошо упаков...	5

Целевая метка - колонка рейтинг

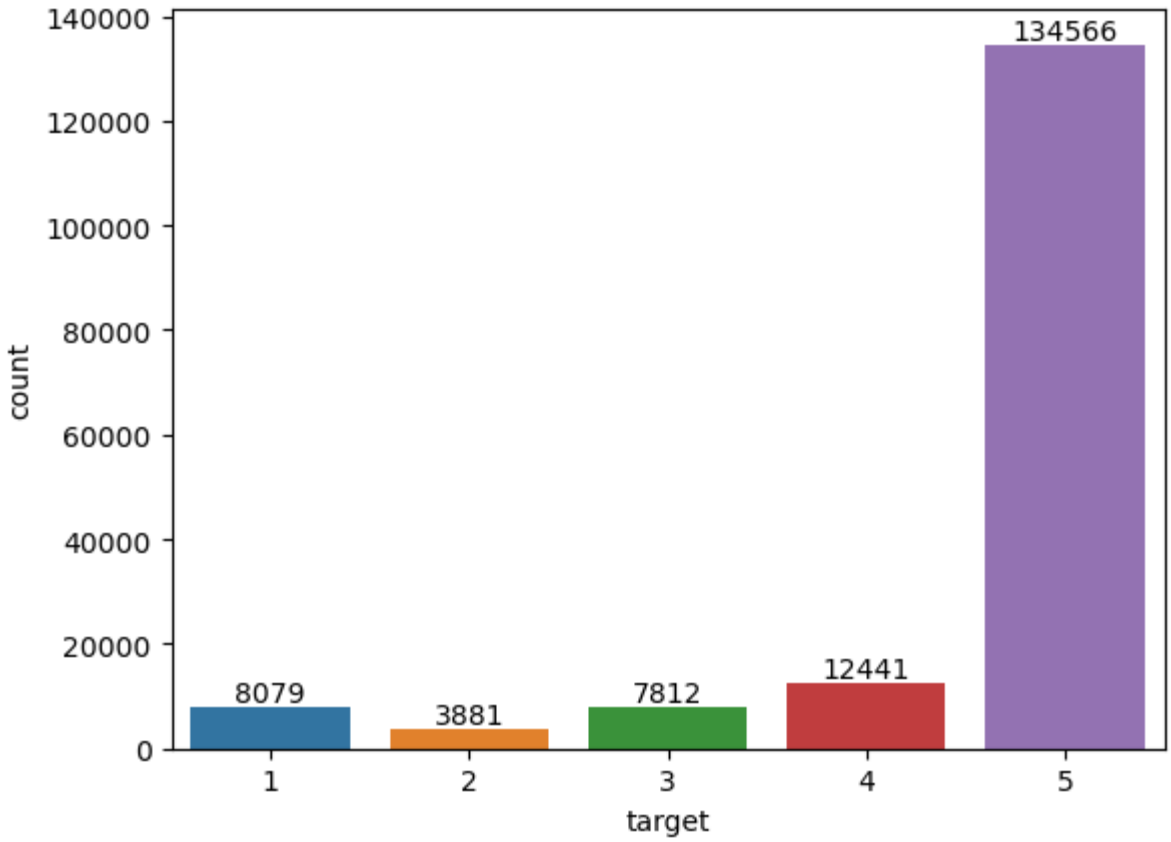
In [23]:
```
df['target'].value_counts()
```

Out[23]:
```
target
5    134566
4     12441
1      8079
3      7812
2      3881
Name: count, dtype: int64
```

In [24]:
```
df["target"].value_counts().plot(
                                    kind='pie',
                                    title='Рейтинг',
                                    figsize=(7, 7),
                                    autopct='%.2f%%')
plt.ylabel('');
```



In [25]:
```
ax = sns.countplot(x='target', data=df)
ax.bar_label(ax.containers[0]);
```

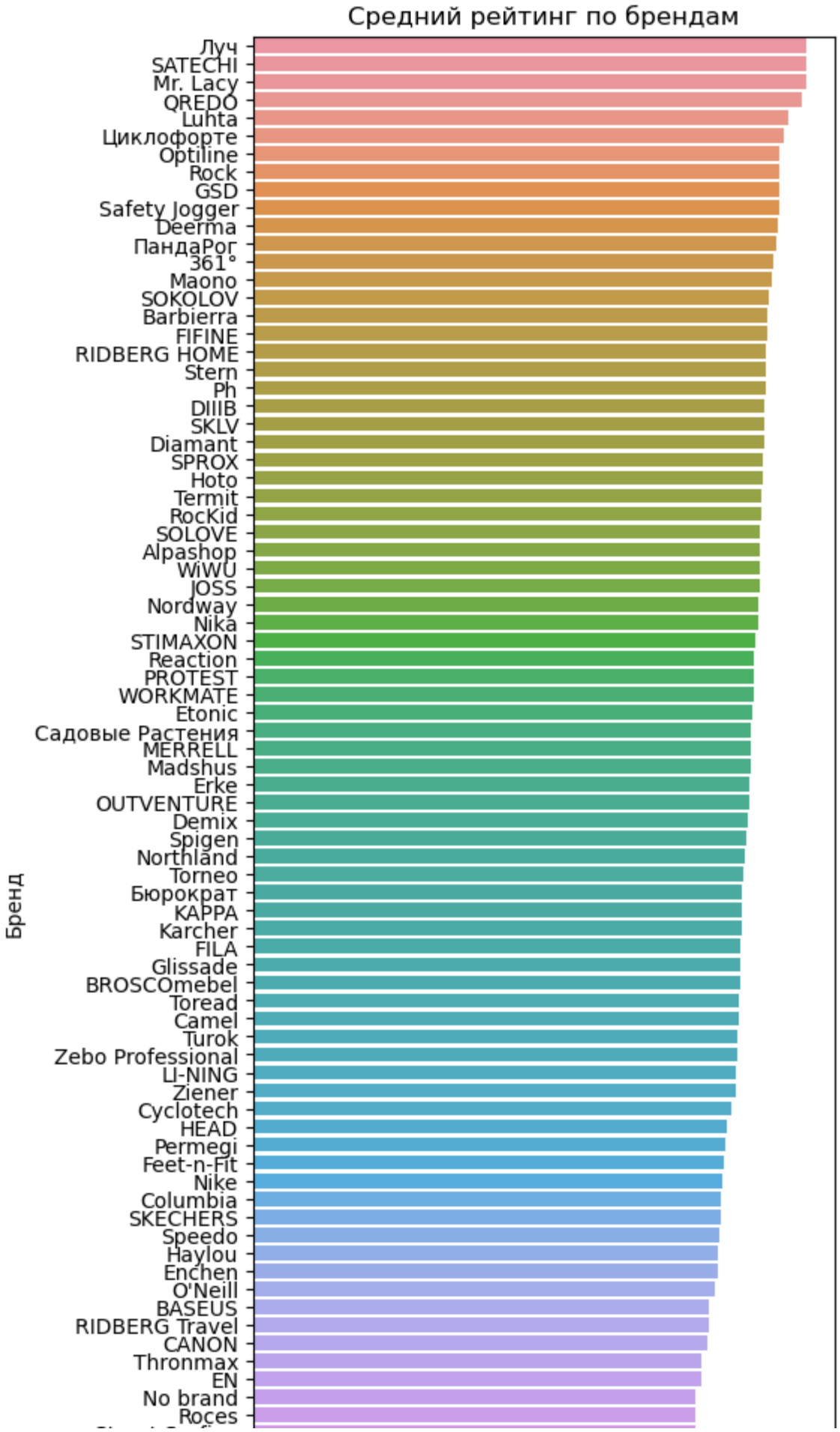


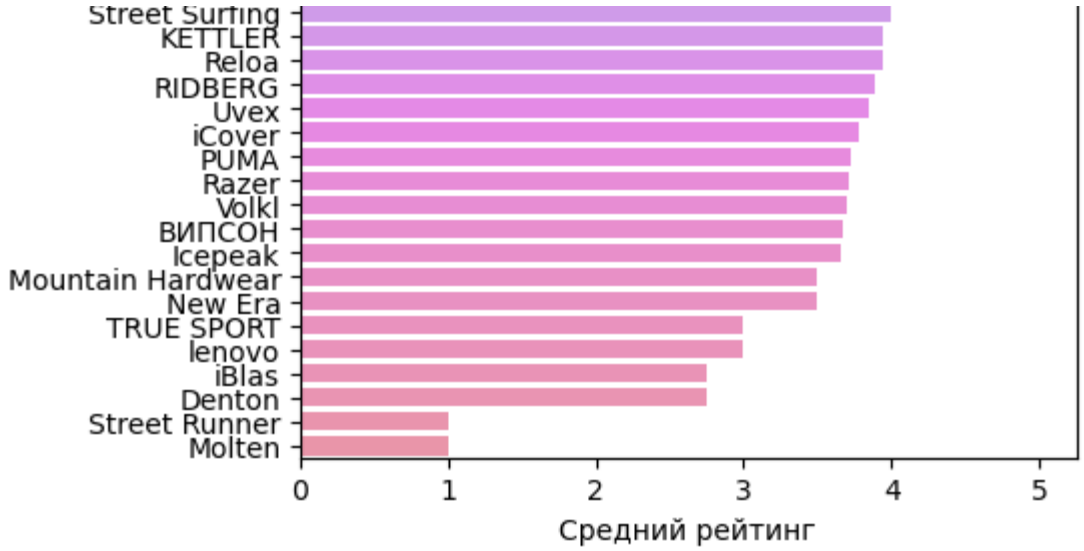
Классы несбалансированы - слишком много "5" и слишком мало остальных оценок, их в 4 раза меньше. Так как данных много, сделаем undersampling. Каждого класса будет по 3800 - этого хватит, чтобы наша модель имела хорошие обобщающие свойства.

Кроме этого можно сделать вывод, что в 4 из 5 случаях покупатель удовлетворён приобретённым товаром, что является хорошим показателем.

In [26]:
```
avg_rate_by_brand = pd.DataFrame(df.groupby('brand')['target'].mean())\
                                    .reset_index()\
                                    .sort_values(by='target', ascending=False)

plt.figure(figsize=(5, 15))
sns.barplot(x='target', y='brand', data=avg_rate_by_brand, orient='h')
plt.xlabel('Средний рейтинг')
plt.ylabel('Бренд')
plt.title('Средний рейтинг по брендам')
plt.show()
```

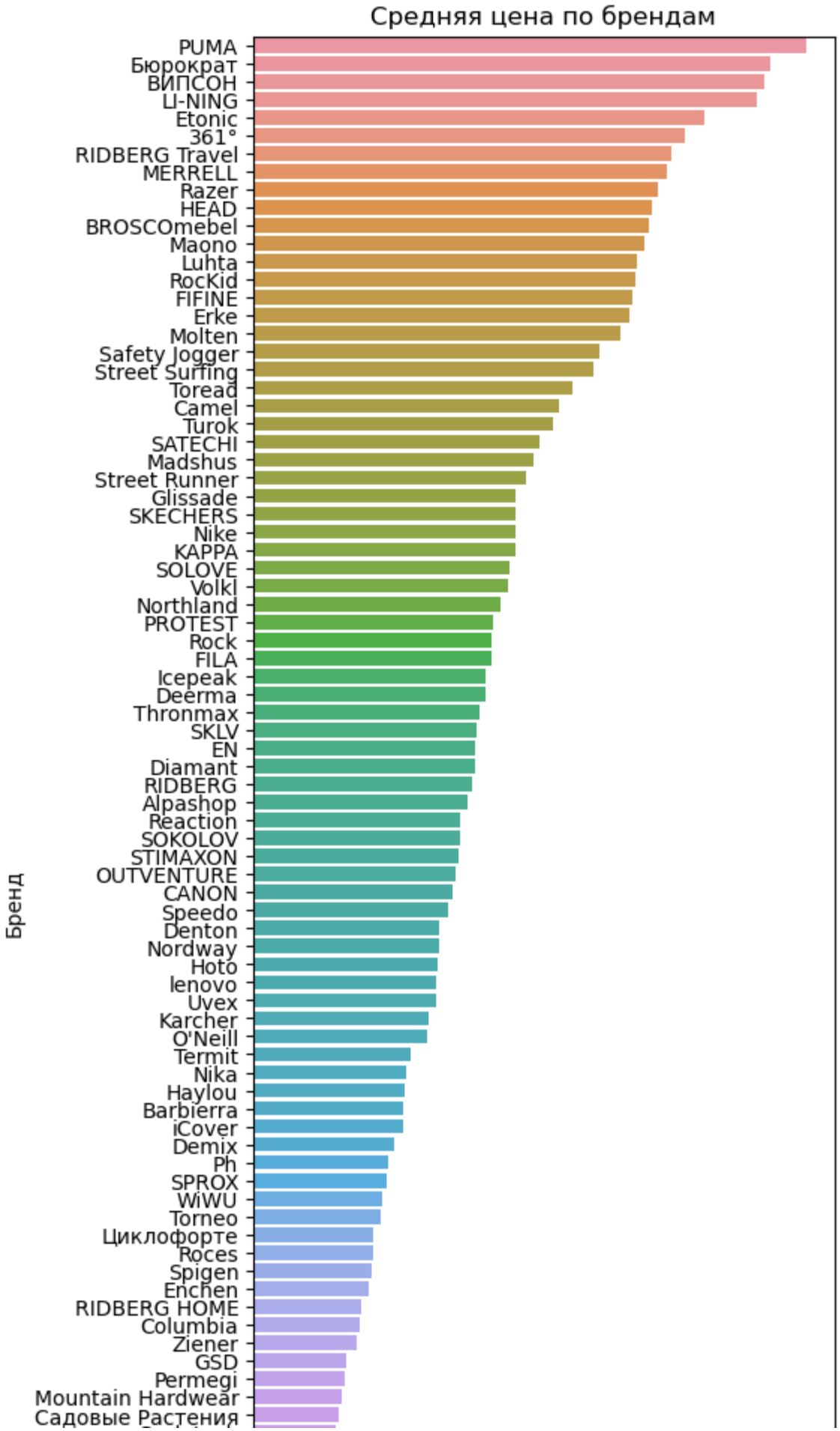

Средний рейтинг по брендам
Бренд
Луч
SATECHI
Mr. Lacy
QREDO
Luhta
Циклофорте
Optiline
Rock
GSD
Safety Jogger
Deerma
ПандаРог
361°
Maono
SOKOLOV
Barbierra
FIFINE
RIDBERG HOME
Stern
Ph
DIIIB
SKLV
Diamant
SPROX
Hoto
Termit
RocKid
SOLOVE
Alpashop
WiWU
JOSS
Nordway
Nika
STIMAXON
Reaction
PROTEST
WORKMATE
Etonic
Садовые Растения
MERRELL
Madshus
Erke
OUTVENTURE
Demix
Spigen
Northland
Torneo
Бюрократ
KAPPA
Karcher
FILA
Glissade
BROSCOmebel
Toread
Camel
Turok
Zebo Professional
LI-NING
Ziener
Cyclotech
HEAD
Permegi
Feet-n-Fit
Nike
Columbia
SKECHERS
Speedo
Haylou
Enchen
O'Neill
BASEUS
RIDBERG Travel
CANON
Thronmax
EN
No brand
Roces

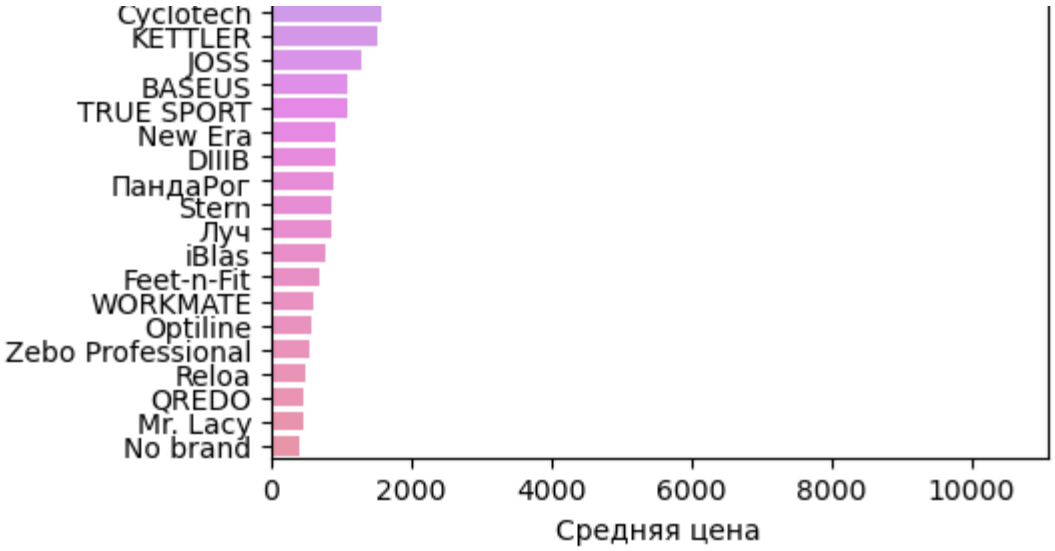


Большинство брендов держат марку - средняя оценка 4 и выше, но есть и недобросовестные, которые продают товары плохого качества, у которых оценка 3 и ниже, но таких производителей всего 6.

In [27]:
```
avg_price_by_brand = pd.DataFrame(df.groupby('brand')['price'].mean())\
                                    .reset_index()\
                                    .sort_values(by='price', ascending=False)

plt.figure(figsize=(5, 15))
sns.barplot(x='price', y='brand', data=avg_price_by_brand, orient='h')
plt.xlabel('Средняя цена')
plt.ylabel('Бренд')
plt.title('Средняя цена по брендам')
plt.show()
```


Средняя цена по брендам
Бренд
PUMA
Бюрократ
ВИПСОН
LI-NING
Etonic
361°
RIDBERG Travel
MERRELL
Razer
HEAD
BROSCOmebel
Maono
Luhta
RocKid
FIFINE
Erke
Molten
Safety Jogger
Street Surfing
Toread
Camel
Turok
SATECHI
Madshus
Street Runner
Glissade
SKECHERS
Nike
KAPPA
SOLOVE
Volkl
Northland
PROTEST
Rock
FILA
Icepeak
Deerma
Thronmax
SKLV
EN
Diamant
RIDBERG
Alpashop
Reaction
SOKOLOV
STIMAXON
OUTVENTURE
CANON
Speedo
Denton
Nordway
Hoto
lenovo
Uvex
Karcher
O'Neill
Termit
Nika
Haylou
Barbierra
iCover
Demix
Ph
SPROX
WiWU
Torneo
Циклофорте
Roces
Spigen
Enchen
RIDBERG HOME
Columbia
Ziener
GSD
Permegi
Mountain Hardwear
Садовые Растения

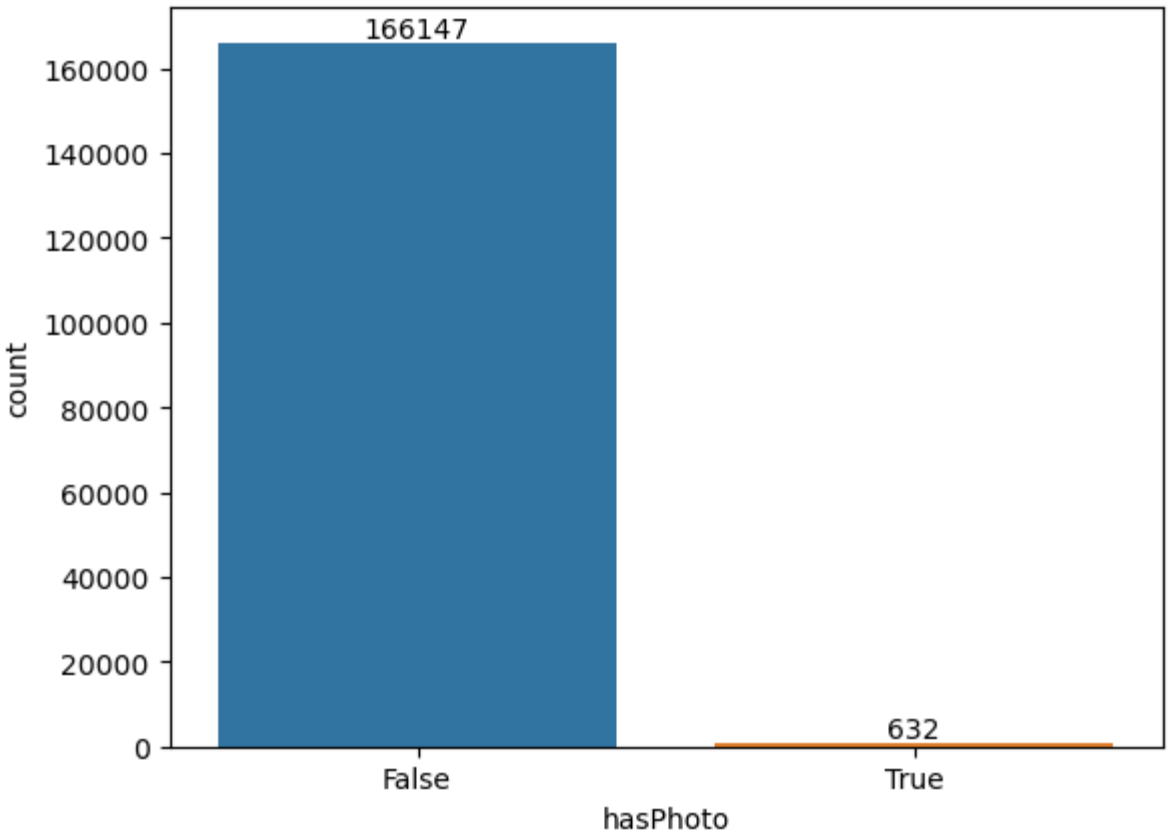


Большая часть товаров различных брендов продаётся по средней цене до ~5000 рублей.

Так же можно отметить, что замыкающая тройка брендов по среднему рейтингу имеет цену на товары в 4-6 тысяч рублей, т.е. за относительно немаленькие деньги покупатели получают ужасный товар.

Столбец наличие фото в отзыве

In [28]:
```
ax = sns.countplot(x='hasPhoto', data=df)
ax.bar_label(ax.containers[0]);
```

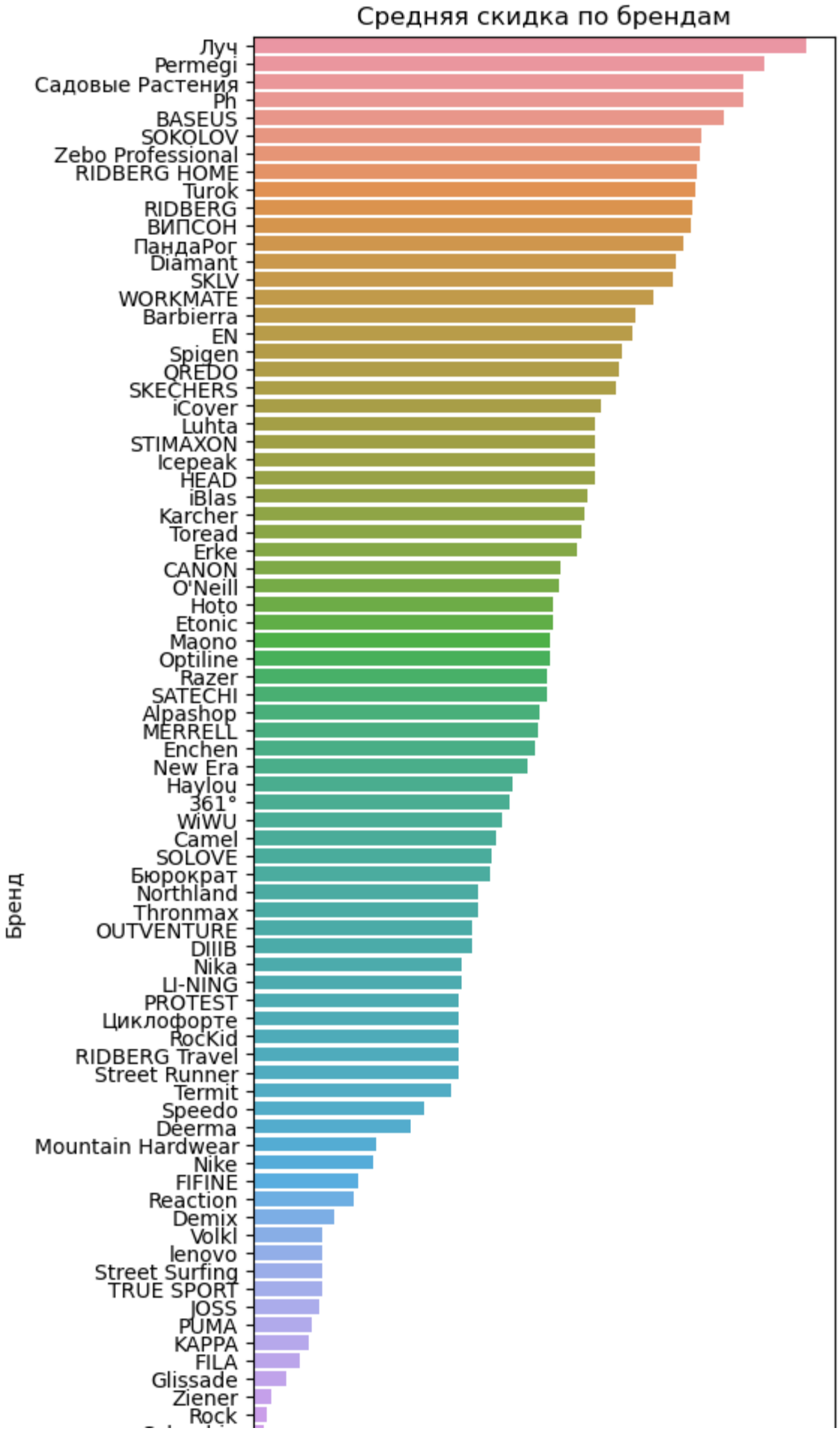


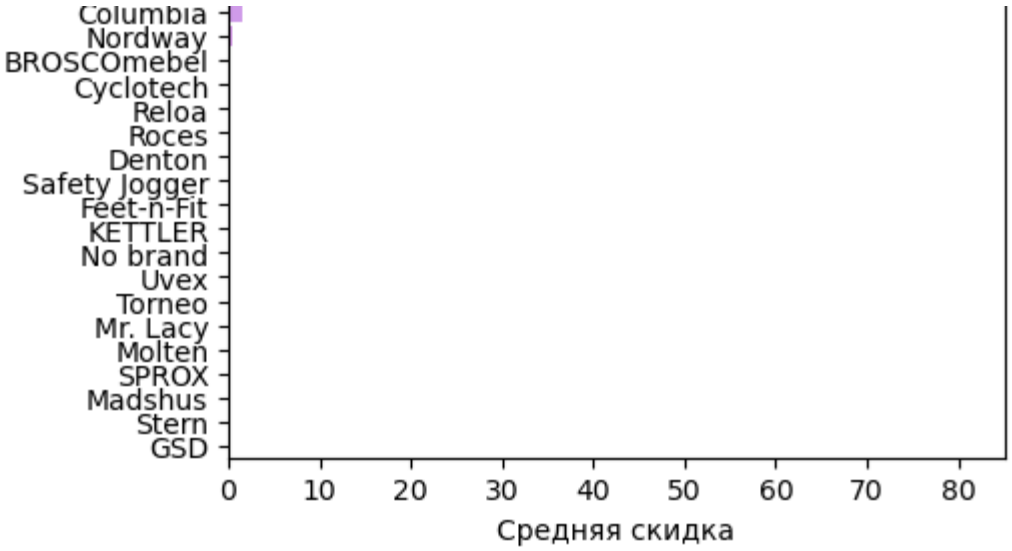
Вывод: подавляющее большинство пользователей предпочитает оставлять отзывы без фотографий.

Столбец скидка

In [29]:
```
avg_sale_by_brand = pd.DataFrame(df.groupby('brand')['sale'].mean())\
                                    .reset_index()\
                                    .sort_values(by='sale', ascending=False)

plt.figure(figsize=(5, 15))
sns.barplot(x='sale', y='brand', data=avg_sale_by_brand, orient='h')
plt.xlabel('Средняя скидка')
plt.ylabel('Бренд')
plt.title('Средняя скидка по брендам')
plt.show()
```

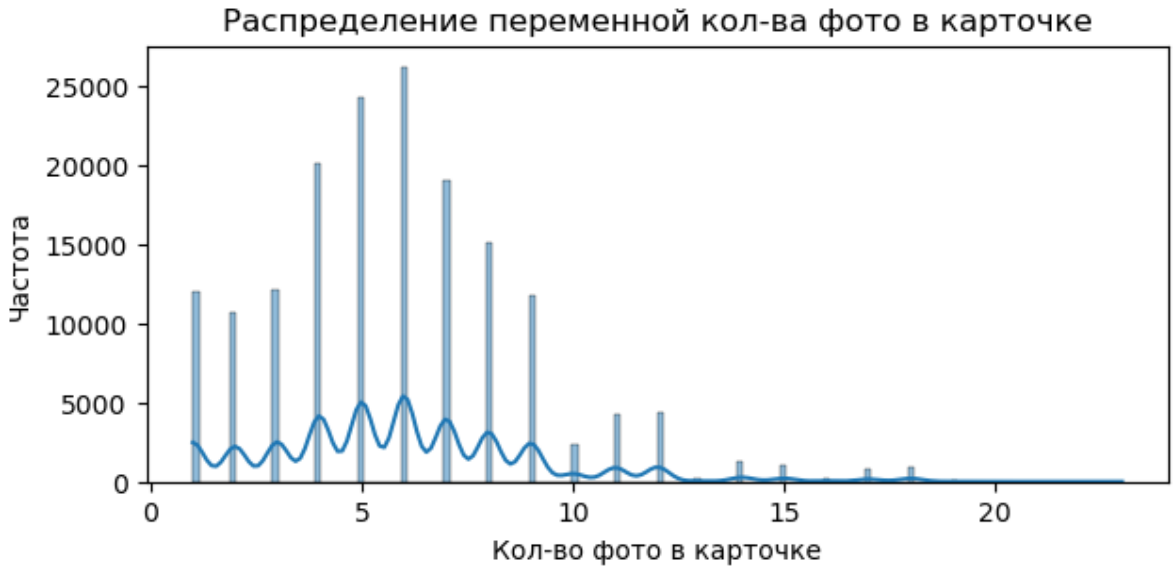

Средняя скидка по брендам
Бренд
Луч
Permegi
Садовые Растения
Ph
BASEUS
SOKOLOV
Zebo Professional
RIDBERG HOME
Turok
RIDBERG
ВИПСОН
ПандаРог
Diamant
SKLV
WORKMATE
Barbierra
EN
Spigen
OREDO
SKECHERS
iCover
Luhta
STIMAXON
Icepeak
HEAD
iBlas
Karcher
Toread
Erke
CANON
O'Neill
Hoto
Etonic
Maono
Optiline
Razer
SATECHI
Alpashop
MERRELL
Enchen
New Era
Haylou
361°
WiWU
Camel
SOLOVE
Бюрократ
Northland
Thronmax
OUTVENTURE
DIIIB
Nika
LI-NING
PROTEST
Циклофорте
RocKid
RIDBERG Travel
Street Runner
Termit
Speedo
Deerma
Mountain Hardwear
Nike
FIFINE
Reaction
Demix
Volkl
lenovo
Street Surfing
TRUE SPORT
JOSS
PUMA
KAPPA
FILA
Glissade
Ziener
Rock



Примерно половина брендов имеет скидку более 40%, что несомненно является накруткой, так как это делается для продвижения карточек товаров наверх списка и создания эффекта приобретения выгоды для покупателя. Приблизительно 20% брендов не имеют скидку.

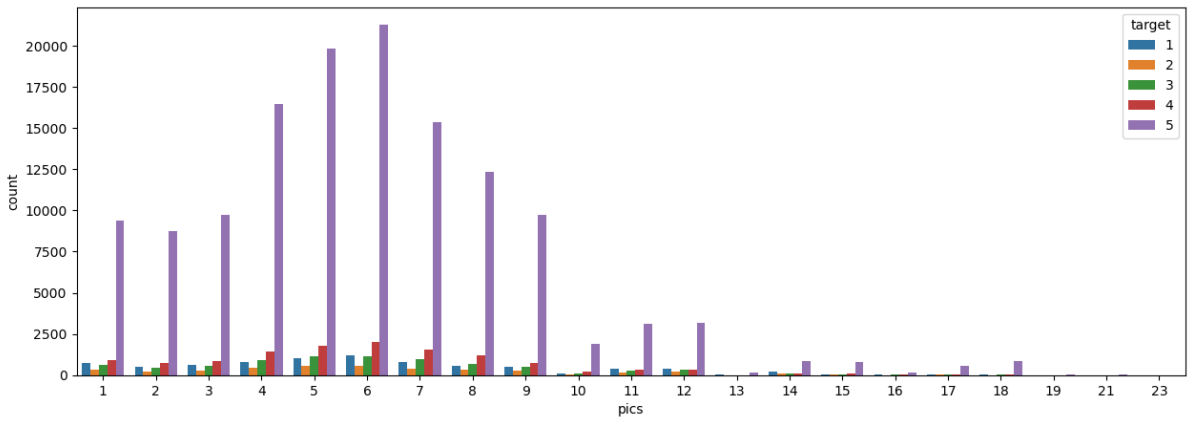
Столбец кол-во фото в карточке

In [30]:
```
plt.figure(figsize=(7,3))
sns.histplot(x=df['pics'], kde=True)
plt.title('Распределение переменной кол-ва фото в карточке')
plt.xlabel('Кол-во фото в карточке')
plt.ylabel('Частота');
```



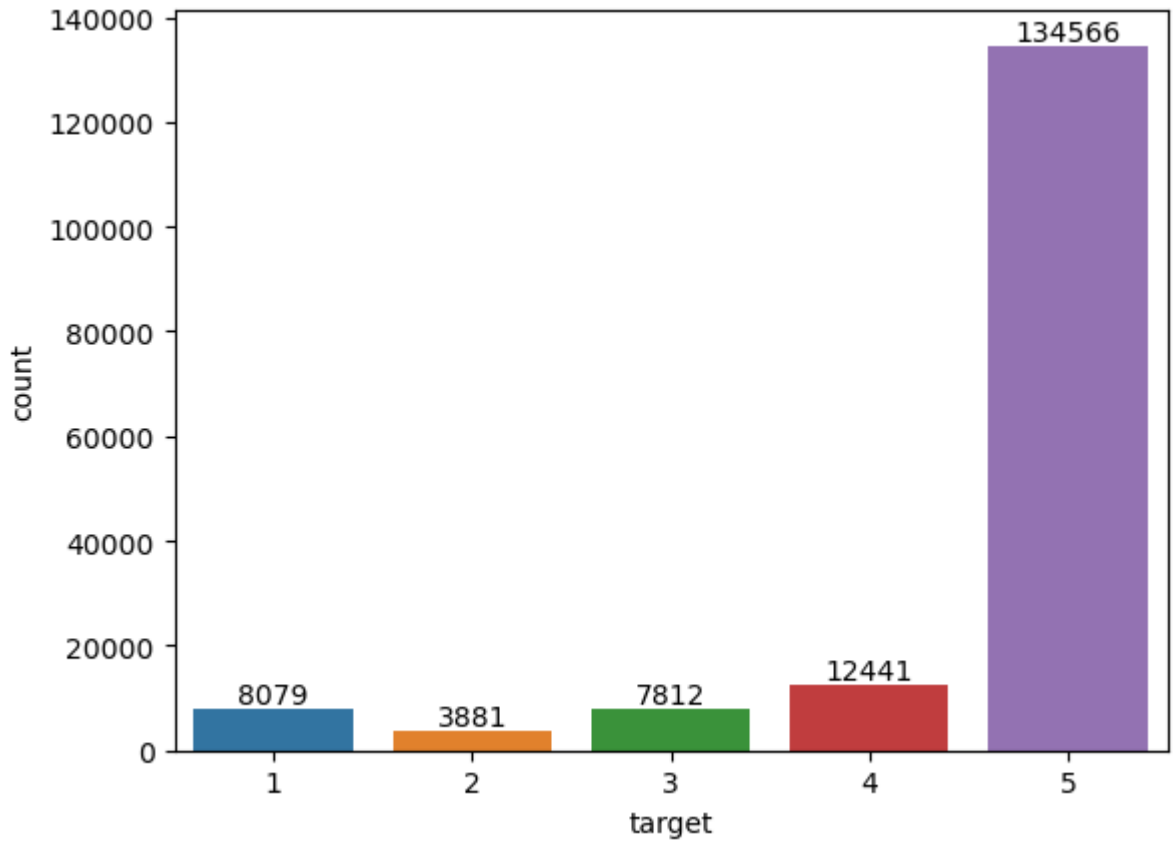
Чаще всего продавцы делают от 1 до 9 фото в карточке. Посмотрим на рейтинг товара в зависимости от количетсва фото в карточке товара

In [31]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(x='pics', data=df, hue='target');
```



Особой зависимости нет - где больше товаров по количеству фотографий, там больше отзывов в целом.

In [32]:
```
ax = sns.countplot(x='target', data=df)
ax.bar_label(ax.containers[0]);
```



Как я говорил ранее, классы несбалансированы, и в силу того, что данных много, сделаем undersampoling, возьмём по 3800 отзывов каждого рейтинга и будем учиться на них

Обработка текста

Облако слов

In [33]:
```
# Получение текстовой строки из списка слов
def str_corpus(corpus):
    str_corpus = ''
    for i in corpus:
        str_corpus += ' ' + i
    str_corpus = str_corpus.strip()
    return str_corpus


# Получение списка всех слов в корпусе
def get_corpus(data):
    corpus = []
    for phrase in data:
        for word in phrase.split():
            corpus.append(word)
    return corpus


# Получение облака слов
def get_wordCloud(corpus):
    wordCloud = WordCloud(background_color='white',
                          stopwords=STOPWORDS,
                          width=3000,
                          height=2500,
                          max_words=200,
                          random_state=42
                          ).generate(str_corpus(corpus))
    return wordCloud


corpus = get_corpus(df['text'].values)
procWordCloud = get_wordCloud(corpus)

fig = plt.figure(figsize=(20, 8))
plt.subplot(1, 2, 1)
plt.imshow(procWordCloud)
plt.axis('off')
plt.subplot(1, 2, 1);
```

Уберём стоп-слова и посмотрим на облаков слов. Пунктуация и эмодзи убраны на этапе парсинга

In [34]:
```
russian_stopwords = stopwords.words("russian")

df['text'] = df['text'].map(lambda x: x.split(' '))
df['text'] = df['text'].map(lambda x: [token for token in x if token not in russian
                                                                and token != " "]
df['text'] = df['text'].map(lambda x: ' '.join(x))
```

In [35]:
```
corpus = get_corpus(df['text'].values)
procWordCloud = get_wordCloud(corpus)

fig = plt.figure(figsize=(20, 8))
plt.subplot(1, 2, 1)
plt.imshow(procWordCloud)
plt.axis('off')
plt.subplot(1, 2, 1);
```

Много слов положительных, оно и понятно - 80% нашего датасета с отличной оценкой.

Сделаем ребалансировку и вновь посмотрим на облако слов.

In [36]:
```
X = df.drop('target', axis=1)
y = df['target']

rus = RandomUnderSampler(sampling_strategy='auto', random_state=42)

X_resampled, y_resampled = rus.fit_resample(X, y)
```

In [37]:
```
y_resampled.value_counts()
```

Out[37]:
```
target
1    3881
2    3881
3    3881
4    3881
5    3881
Name: count, dtype: int64
```

In [38]:
```
X_resampled.reset_index(drop=True, inplace=True)
y_resampled.reset_index(drop=True, inplace=True)

df = pd.concat([X_resampled, y_resampled], axis=1)
```

Теперь у нас есть 5 сбалансированных класов. Посмотрим на облако слов

In [39]:
```
corpus = get_corpus(df['text'].values)
procWordCloud = get_wordCloud(corpus)

fig = plt.figure(figsize=(20, 8))
plt.subplot(1, 2, 1)
plt.imshow(procWordCloud)
plt.axis('off')
plt.subplot(1, 2, 1);
```

В силу того, что больше половины датасета составляют отзывы о продукции "Sokolov", то встречается немало слов, связанных с тематикой ювелирных изделий(например, серебро, серьги/серёжки, кольцо и т.п.)

Обработка числовых и категориальных признаков

Стандартизируем числовые признаки, закодируем категориальные признаки, векторизируем текст

In [40]:
```
numeric_columns
```

Out[40]:
```
['price', 'sale', 'pics', 'useful', 'unuseful', 'target']
```

In [41]:
```
scaler = StandardScaler()
df[numeric_columns[:-1]] = scaler.fit_transform(df[numeric_columns[:-1]])
```

In [42]:
```
df
```

Out[42]:

	name	brand	price	sale	pics	hasPhoto	useful	unuseful	
0	Шлепанцы Mono	FILA	-0.144880	-2.472030	-0.618441	False	-0.030635	0.600217	a op
1	Кольцо из серебра	SOKOLOV	0.178815	0.401142	-0.925706	False	0.518779	-0.542338	
2	Ювелирный каучуковый шнурок с замком из серебр...	SOKOLOV	0.015126	0.575274	-0.925706	False	1.617607	0.600217	ра
3	Ювелирные серьги кольца серебро 925	SOKOLOV	-0.230407	0.314076	-0.618441	False	-0.580049	-0.542338	
4	Женские серьги пусеты гвоздики из золота 585 п...	SOKOLOV	2.715995	0.357609	0.610619	False	-0.580049	1.742772	по
...	...	...	...	...	...	...	...	...	
19400	Астильба Арендса Mix	Садовые Растения	-0.962916	0.749406	-1.540236	False	-0.030635	-0.542338	
19401	Кольцо из серебра с фианитом	SOKOLOV	-0.312252	0.314076	-0.311176	False	-0.580049	-0.542338	
19402	Женское кольцо на помолвку из серебра 925	SOKOLOV	0.015126	0.314076	0.917884	False	-0.580049	-0.542338	a
19403	Ювелирная подвеска кулон на шею серебро 925	SOKOLOV	-0.475941	0.314076	0.610619	False	2.167021	0.600217	пе

	name	brand	price	sale	pics	hasPhoto	useful	unuseful
19404	Шлепанцы	FILA	-0.144880	-2.472030	-0.618441	False	-0.580049	-0.542338

19405 rows × 10 columns

Закодируем категориальные признаки

In [43]:
```
df['brand'].value_counts()
```

Out[43]:
```
brand
SOKOLOV              9423
Zebo Professional    2060
Садовые Растения     1413
Demix                 911
Permegi               650
                     ...
Mountain Hardwear       1
HEAD                    1
Denton                  1
Street Surfing          1
QREDO                   1
Name: count, Length: 86, dtype: int64
```

In [44]:
```
df['name'].value_counts()
```

Out[44]:
```
name
Ювелирные серьги женские из серебра 925            711
Кольцо из серебра с фианитами                      667
Цепочка на шею из серебра 925                      423
Ювелирные серьги пусеты-гвоздики из серебра 925    394
Ювелирная цепочка на шею серебро 925               391
                                                  ...
Куртка софтшелл женская                              1
Ролики JUNIOR GIRL                                   1
Перчатки сноубордические KAILA                       1
Виноград девичий Yellow Wall                         1
Куртка утепленная для девочек                        1
Name: count, Length: 988, dtype: int64
```

Закодируем бренд OneHotEncoder'ом, а столбец с названием удалим.

In [45]:
```
brand = pd.get_dummies(df["brand"], drop_first=True, dtype=int)
df.drop(columns=["name", "brand"], inplace=True)
df = pd.concat([df, brand], axis=1)
df.head()
```

Out[45]:

	price	sale	pics	hasPhoto	useful	unuseful	text	target	Alpashop
0	-0.144880	-2.472030	-0.618441	False	-0.030635	0.600217	пришли абсолютно схожи оригиналом фото китайск...	1	0
1	0.178815	0.401142	-0.925706	False	0.518779	-0.542338	кольцо пришло согнутое думаю продавцу нужно пе...	1	0
2	0.015126	0.575274	-0.925706	False	1.617607	0.600217	шнурок вместо сильно разочаровал ваш товар хот...	1	0
3	-0.230407	0.314076	-0.618441	False	-0.580049	-0.542338	серьги еле надела это кошмар	1	0
4	2.715995	0.357609	0.610619	False	-0.580049	1.742772	понравилась застежка	1	0

5 rows × 93 columns

Закодируем наличие фото в отзыве OneHotEncoder'ом

In [46]:
```
has_photo = pd.get_dummies(df["hasPhoto"], drop_first=True, dtype=int)
df.drop(columns=["hasPhoto"], inplace=True)
df = pd.concat([df, has_photo], axis=1)
df.head()
```

Out[46]:

	price	sale	pics	useful	unuseful	text	target	Alpashop	BASEUS
0	-0.144880	-2.472030	-0.618441	-0.030635	0.600217	пришли абсолютно схожи оригиналом фото китайск...	1	0	0
1	0.178815	0.401142	-0.925706	0.518779	-0.542338	кольцо пришло согнутое думаю продавцу нужно пе...	1	0	0
2	0.015126	0.575274	-0.925706	1.617607	0.600217	шнурок вместо сильно разочаровал ваш товар хот...	1	0	0
3	-0.230407	0.314076	-0.618441	-0.580049	-0.542338	серьги еле надела это кошмар	1	0	0
4	2.715995	0.357609	0.610619	-0.580049	1.742772	понравилась застежка	1	0	0

5 rows × 93 columns

Обработка текста

In [47]:

```
morph = MorphAnalyzer()
def lemmatize(doc):
    tokens = []
    for token in doc.split():
        if token:
            token = token.strip()
            token = morph.normal_forms(token)[0]
            tokens.append(token)
    if len(tokens) > 0:
        return " ".join(tokens)
    return None

df["text"] = df["text"].apply(lemmatize)
df.head()
```

Out[47]:

	price	sale	pics	useful	unuseful	text	target	Alpashop	BASEUS
0	-0.144880	-2.472030	-0.618441	-0.030635	0.600217	прислать абсолютно схожий оригинал фото китайс...	1	0	0
1	0.178815	0.401142	-0.925706	0.518779	-0.542338	кольцо прийти согнутый думать продавец нужно п...	1	0	0
2	0.015126	0.575274	-0.925706	1.617607	0.600217	шнурок вместо сильно разочаровать ваш товар хо...	1	0	0
3	-0.230407	0.314076	-0.618441	-0.580049	-0.542338	серьга еле надеть это кошмар	1	0	0
4	2.715995	0.357609	0.610619	-0.580049	1.742772	понравиться застёжка	1	0	0

5 rows × 93 columns

In [48]:
```
df["text"].isna().sum()
```

Out[48]: 6

In [49]:
```
df.dropna(inplace=True)
```

In [50]:
```
df.head()
```

Out[50]:

	price	sale	pics	useful	unuseful	text	target	Alpashop	BASEUS
0	-0.144880	-2.472030	-0.618441	-0.030635	0.600217	прислать абсолютно схожий оригинал фото китайс...	1	0	0
1	0.178815	0.401142	-0.925706	0.518779	-0.542338	кольцо прийти согнутый думать продавец нужно п...	1	0	0
2	0.015126	0.575274	-0.925706	1.617607	0.600217	шнурок вместо сильно разочаровать ваш товар хо...	1	0	0
3	-0.230407	0.314076	-0.618441	-0.580049	-0.542338	серьга еле надеть это кошмар	1	0	0
4	2.715995	0.357609	0.610619	-0.580049	1.742772	понравиться застёжка	1	0	0

5 rows × 93 columns

In [51]:
```
corpus = df["text"].to_list()

vectorizer = TfidfVectorizer()

X_tf_idf = vectorizer.fit_transform(corpus)

X_tf_idf
```

Out[51]:
```
<19399x12312 sparse matrix of type '<class 'numpy.float64'>'
        with 189726 stored elements in Compressed Sparse Row format>
```

In [52]:
```
tfidf_df = pd.DataFrame(X_tf_idf.toarray(), columns=vectorizer.get_feature_names_ou
```

In [53]:
```
df.reset_index(drop=True, inplace=True)
tfidf_df.reset_index(drop=True, inplace=True)

df = pd.concat([df, tfidf_df], axis=1)
```

In [54]:
```
df.drop(columns=["text"], inplace=True)
```

In [55]:
```
df.columns = df.columns.astype(str)
```

In [56]:
```
X = df.drop(columns=["target"])
y = df["target"]
```

In [57]:
```
y.shape, X.shape, df.shape
```

Out[57]:
```
((19399,), (19399, 12403), (19399, 12404))
```

Обучение моделей

In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

KNN

In [59]:
```
knnclf = KNeighborsClassifier()
```

Train

In [60]:
```
%%time
knnclf.fit(X_train, y_train)
```

```
CPU times: total: 10.8 s
Wall time: 12.5 s
```

Out[60]:
▾ KNeighborsClassifier

KNeighborsClassifier()

Test

In [61]:
```
%%time
y_pred = knnclf.predict(X_test)
print(f'accuracy {accuracy_score(y_pred, y_test):.2}')
print(classification_report(y_test, y_pred))
```

```
accuracy 0.33
              precision    recall  f1-score   support

           1       0.34      0.46      0.39       767
           2       0.26      0.27      0.26       773
           3       0.24      0.19      0.21       792
           4       0.30      0.25      0.27       785
           5       0.48      0.47      0.47       763

    accuracy                           0.33      3880
   macro avg       0.32      0.33      0.32      3880
weighted avg       0.32      0.33      0.32      3880

CPU times: total: 1min 44s
Wall time: 11.5 s
```

Random Forest

In [62]:
```
rfc = RandomForestClassifier()
```

Train

In [63]:
```
%%time
rfc.fit(X_train, y_train)
```

```
CPU times: total: 1min 16s
Wall time: 1min 22s
```

Out[63]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

Test

In [64]:
```
%%time
y_pred = rfc.predict(X_test)
print(f'accuracy {accuracy_score(y_pred, y_test):.2}')
print(classification_report(y_test, y_pred))
```

```
accuracy 0.44
              precision    recall  f1-score   support

           1       0.44      0.62      0.52       767
           2       0.29      0.25      0.27       773
           3       0.31      0.20      0.24       792
           4       0.42      0.37      0.39       785
           5       0.63      0.77      0.70       763

    accuracy                           0.44      3880
   macro avg       0.42      0.44      0.42      3880
weighted avg       0.42      0.44      0.42      3880

CPU times: total: 562 ms
Wall time: 836 ms
```

AdaBoost

In [65]:
```
abc = AdaBoostClassifier()
```

Train

In [66]:
```
%%time
abc.fit(X_train, y_train)
```

```
CPU times: total: 1min 6s
Wall time: 1min 13s
```

Out[66]:
```
▾ AdaBoostClassifier

AdaBoostClassifier()
```

Test

In [67]:
```
%%time
y_pred = abc.predict(X_test)
print(f'accuracy {accuracy_score(y_pred, y_test):.2}')
print(classification_report(y_test, y_pred))
```

```
accuracy 0.4
              precision    recall  f1-score   support

           1       0.43      0.55      0.49       767
           2       0.28      0.15      0.19       773
           3       0.28      0.30      0.29       792
           4       0.35      0.38      0.36       785
           5       0.62      0.64      0.63       763

    accuracy                           0.40      3880
   macro avg       0.39      0.40      0.39      3880
weighted avg       0.39      0.40      0.39      3880

CPU times: total: 5.12 s
Wall time: 6.15 s
```

MLP

In [68]:
```
mlpc = MLPClassifier()
```

Train

In [69]:
```
%%time
mlpc.fit(X_train, y_train)
```

```
CPU times: total: 3h 59min 49s
Wall time: 50min 41s
C:\Users\ibasl\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Out[69]:
```
▾ MLPClassifier

MLPClassifier()
```

Test

In [70]:
```
%%time
y_pred = mlpc.predict(X_test)
print(f'accuracy {accuracy_score(y_pred, y_test):.2}')
print(classification_report(y_test, y_pred))
```

```
accuracy 0.42
              precision    recall  f1-score   support

           1       0.47      0.49      0.48       767
           2       0.32      0.32      0.32       773
           3       0.27      0.27      0.27       792
           4       0.37      0.35      0.36       785
           5       0.67      0.70      0.68       763

    accuracy                           0.42      3880
   macro avg       0.42      0.42      0.42      3880
weighted avg       0.42      0.42      0.42      3880

CPU times: total: 1.16 s
Wall time: 428 ms
```

In [71]:
```
column0 = ['KNN', 'Random Forest', 'AdaBoost','MLP']
column1 = ["8.49 s", '1min 23s', '1 min 10s', '36 min 18s']
column2 = ["658 ms", '1min 22s', '1min 5s', '36min 17s']
column3 = ['7.82 s', '605 ms', '5.12 s', '406 ms']
column4 = [0.33, 0.45, 0.4, 0.42]


data = {'Model': column0,
        'Run time': column1,
        'Traing time': column2,
        'Test time': column3,
        'Accuracy': column4}
table = pd.DataFrame(data)
table.set_index('Model', inplace=True)
table
```

Out[71]:

Model	Run time	Traing time	Test time	Accuracy
KNN	8.49 s	658 ms	7.82 s	0.33
Random Forest	1min 23s	1min 22s	605 ms	0.45
AdaBoost	1 min 10s	1min 5s	5.12 s	0.40
MLP	36 min 18s	36min 17s	406 ms	0.42

Вывод

Общее качество моделей среднее - каждая из них очень близка к угадыванию рейтинга, за исключением отзывов с оценкой «5». Эти тексты модели случайного леса и многослойного перцептрона различает гораздо лучше остальных 0.63 и 0.7 f1-меры соответственно. На мой взгляд это связано с четырьмя факторами - комментарии недостаточно длинные, чтобы модель могла извлечь нужный посыл; нужно использовать word embeddings вместо tf-idf векторизации; нужно использовать более сложные алгоритмы, например, глубокое обучение с rnn или с attention слоями; кроме этого отзывы покупателей предвзяты - каждый сам для себя определяет из-за чего снижать оценку.

Среднее качество потому, что даже современнные BERT-модели не дают точность более чем 0.6 согласно статьи

Table 2: SST-5 Results

Model	Training Time (per epoch)	Best Test Acc.
$\text{BERT}_{\text{BASE}}$	5:38	0.549
$\text{BERT}_{\text{LARGE}}$	12:38	0.562
$\text{ALBERT}_{\text{BASE}}$	3:16	0.490
$\text{DistilBERT}_{\text{BASE}}$	2:54	0.532
$\text{RoBERTa}_{\text{LARGE}}$	N/A	0.602

Table 3: Experiment results for classification task on SST-5 root nodes

In [ ]: