
Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Домашняя работа
по дисциплине
«Методы машинного обучения»

Выполнил:
студентка группы ИУ5-21М
Попова И.А.

Москва — 2020 г.

Домашнее задание

по дисциплине «Методы машинного обучения»

Домашнее задание по дисциплине направлено на решение комплексной задачи машинного обучения.

Домашнее задание включает выполнение следующих шагов:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1-2 гиперпараметров. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

В качестве набора данных используется датасет, содержащий данные о 1000 фотокамерах на основе 13 свойств.

Выбранный набор данных состоит из одного файла camera_dataset.csv, содержащего все данные датасета. Данный файл содержит следующие колонки(13 свойств каждой камеры):

- Model - модель
- Release date - дата выпуска
- Max resolution - максимальное разрешение
- Low resolution - низкое разрешение
- Effective pixels - эффективные пиксели (пиксели, которые захватывают данные изображения)
- Zoom wide (W) - широкий фокусный диапазон
- Zoom tele (T) - телефото зум
- Normal focus range - нормальный диапазон фокусировки
- Macro focus range - макро диапазон фокусировки
- Storage included - объем хранилища
- Weight (inc. batteries) - вес (включая батареи)
- Dimensions - габаритные размеры
- Price - цена

Original dataset: <https://www.kaggle.com/crawford/1000-cameras-dataset/>
[\(https://www.kaggle.com/crawford/1000-cameras-dataset/\)](https://www.kaggle.com/crawford/1000-cameras-dataset/)

In [1]:

```
!pip install gmdhpy
```

```
Collecting gmdhpy
  Downloading https://files.pythonhosted.org/packages/66/c2/660fd6f1f595f1
  858fe8edf010c398d1fb220bb20d9b2b35c0bbb5224130/GmdhPy-0.1.1a0-py2.py3-none
  -any.whl
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from gmdhpy) (1.12.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from gmdhpy) (1.18.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (from gmdhpy) (0.22.2.post1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->gmdhpy) (0.14.1)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->gmdhpy) (1.4.1)
Installing collected packages: gmdhpy
Successfully installed gmdhpy-0.1.1a0
```

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from gmdhpy import gmdh
%matplotlib inline
sns.set(style="ticks")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
```

```
    import pandas.util.testing as tm
```

In [3]:

```
# Обучающая выборка
data = pd.read_csv('camera_dataset.csv', sep=",")
data.head()
```

Out[3]:

	Model	Release date	Max resolution	Low resolution	Effective pixels	Zoom wide (W)	Zoom tele (T)	Normal focus range	Macro focus range	Storage included
0	Agfa ePhoto 1280	1997	1024.0	640.0	0.0	38.0	114.0	70.0	40.0	4.0
1	Agfa ePhoto 1680	1998	1280.0	640.0	1.0	38.0	114.0	50.0	0.0	4.0
2	Agfa ePhoto CL18	2000	640.0	0.0	0.0	45.0	45.0	0.0	0.0	2.0
3	Agfa ePhoto CL30	1999	1152.0	640.0	0.0	35.0	35.0	0.0	0.0	4.0
4	Agfa ePhoto CL30 Clik!	1999	1152.0	640.0	0.0	43.0	43.0	50.0	0.0	40.0

In [4]:

```
data.shape
```

Out[4]:

(1038, 13)

In [5]:

```
data.dtypes
```

Out[5]:

```
Model          object
Release date   int64
Max resolution float64
Low resolution float64
Effective pixels float64
Zoom wide (W)  float64
Zoom tele (T)  float64
Normal focus range float64
Macro focus range float64
Storage included float64
Weight (inc. batteries) float64
Dimensions     float64
Price          float64
dtype: object
```

In [6]:

```
data.isnull().sum()
```

Out[6]:

```
Model          0
Release date   0
Max resolution 0
Low resolution 0
Effective pixels 0
Zoom wide (W)  0
Zoom tele (T)  0
Normal focus range 0
Macro focus range 1
Storage included 2
Weight (inc. batteries) 2
Dimensions     2
Price          0
dtype: int64
```

In [0]:

```
data = data.fillna(0)
```

In [8]:

```
data.isnull().sum()
```

Out[8]:

```
Model          0
Release date   0
Max resolution 0
Low resolution 0
Effective pixels 0
Zoom wide (W) 0
Zoom tele (T)  0
Normal focus range 0
Macro focus range 0
Storage included 0
Weight (inc. batteries) 0
Dimensions      0
Price           0
dtype: int64
```

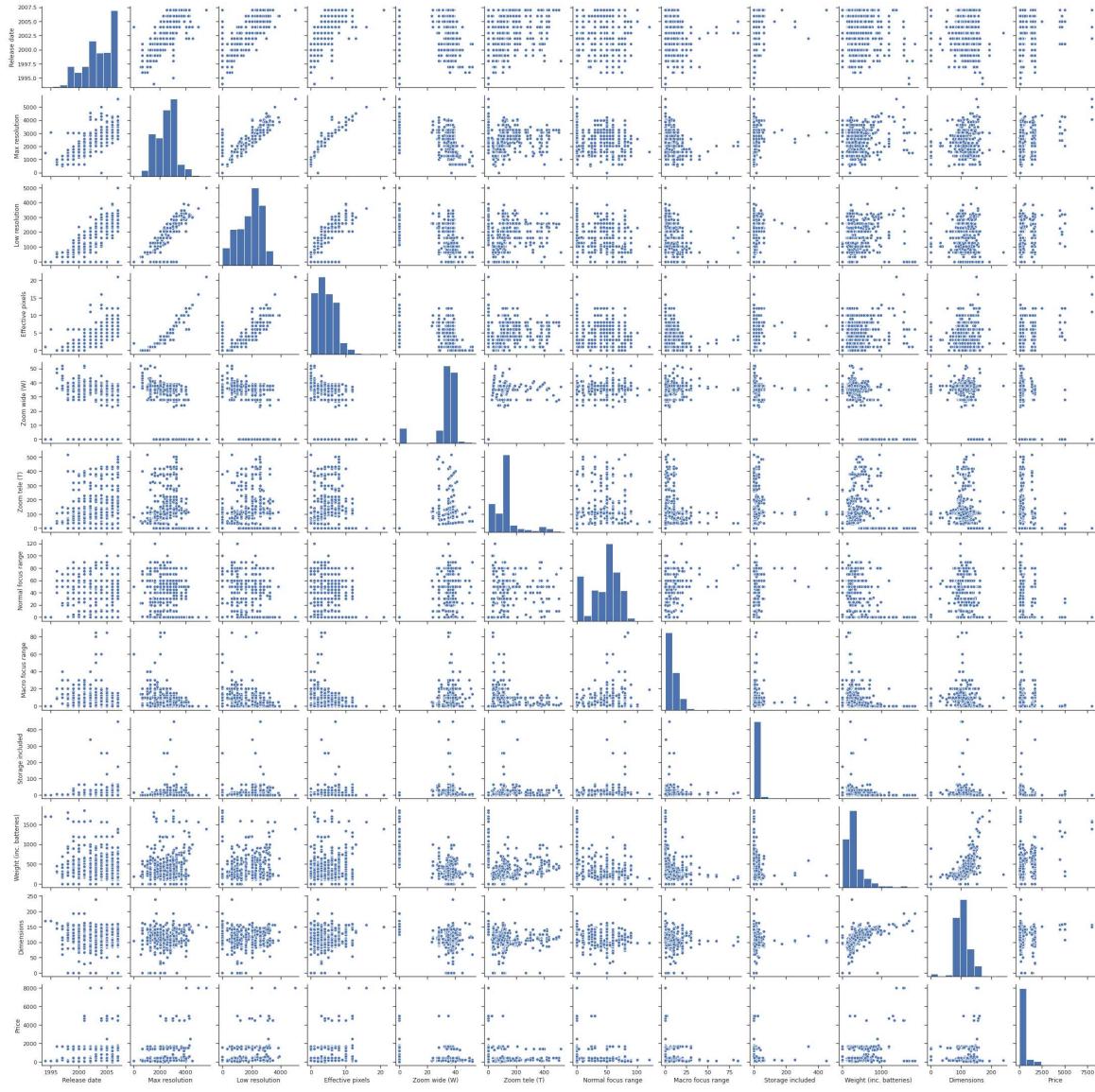
Вывод. Представленный набор данных не содержит пропусков ни в обучающей, ни в тестовой выборках. Построим некоторые графики для понимания структуры данных.

In [9]:

```
# Парные диаграммы
sns.pairplot(data)
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x7f6df9b52e80>
```

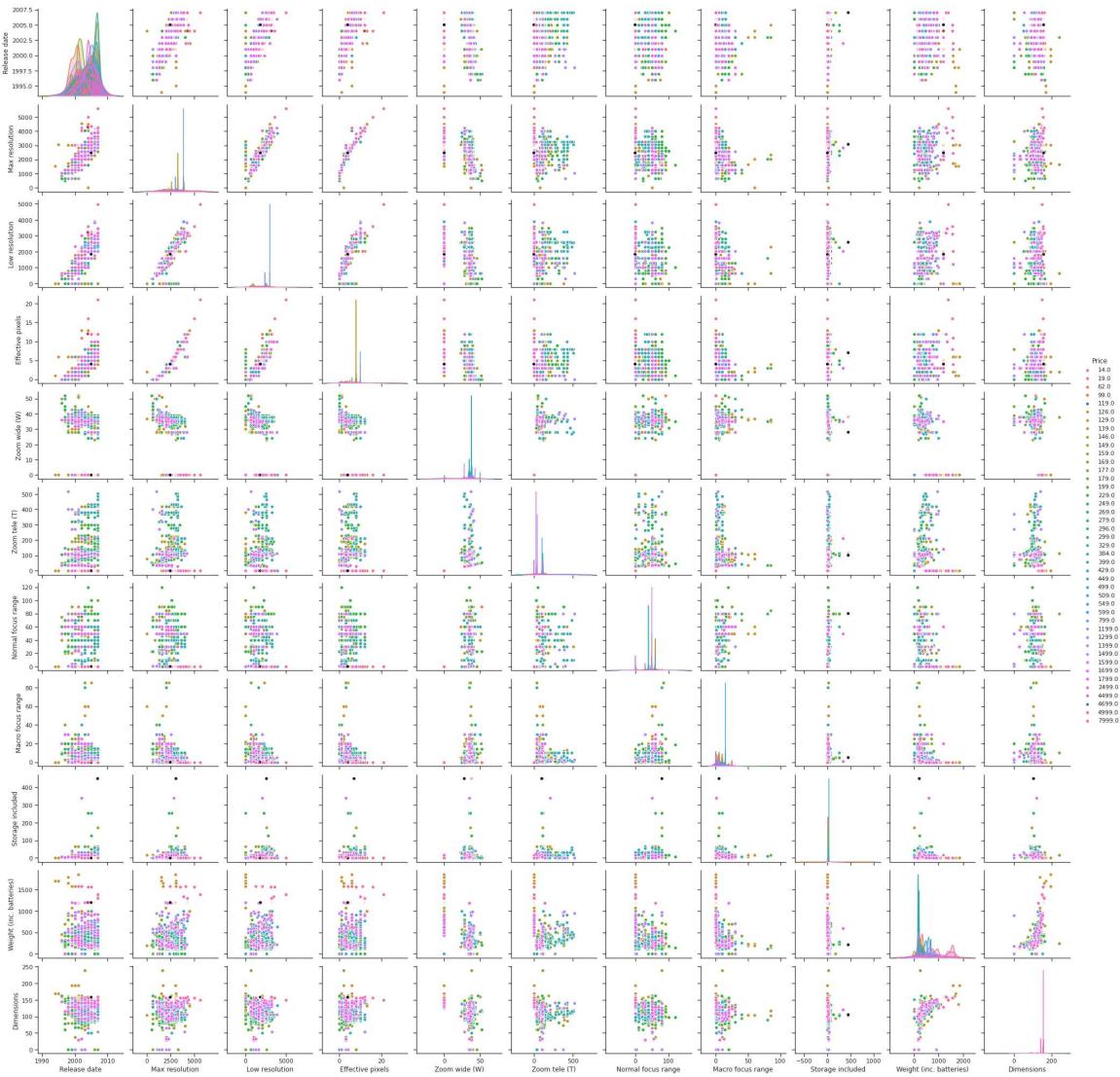


In [10]:

```
sns.pairplot(data, hue="Price")
```

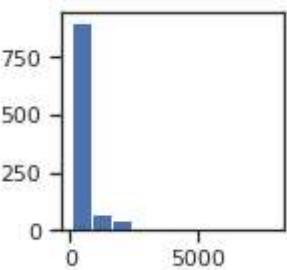

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x7f6df44bb5f8>
```



In [11]:

```
# Оценим дисбаланс классов для Price
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['Price'])
plt.show()
```



In [12]:

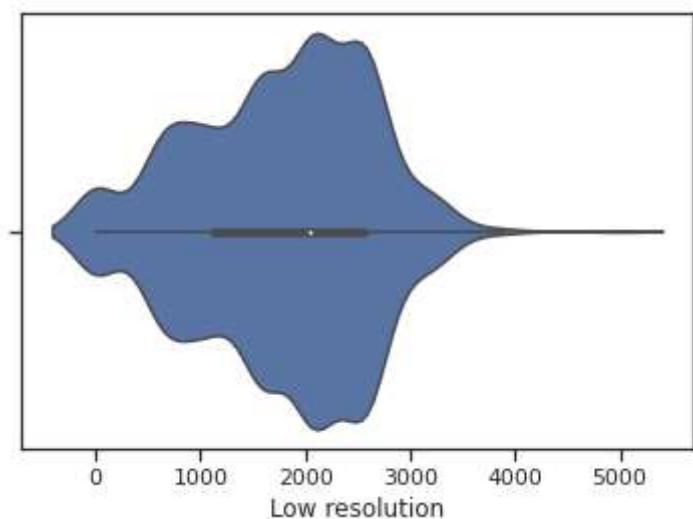
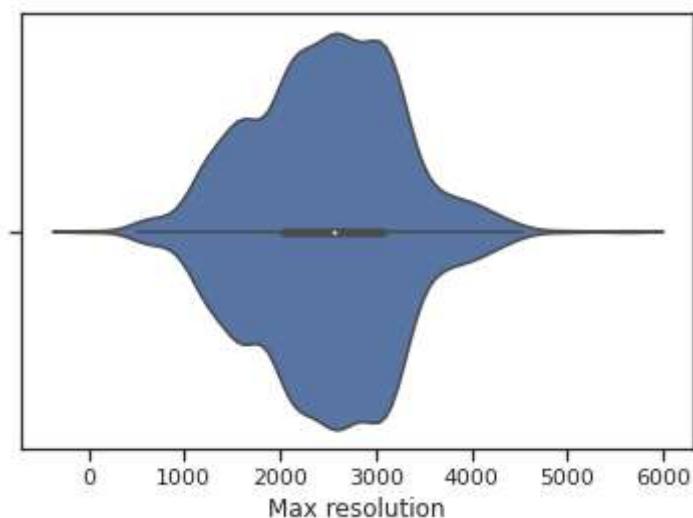
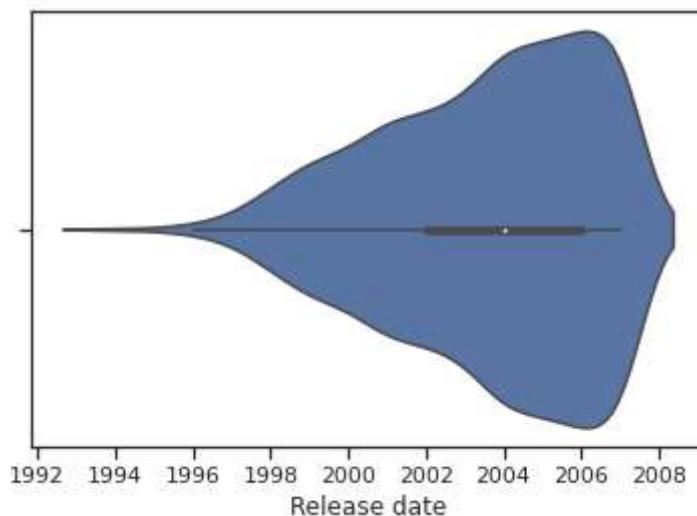
```
data.columns[1:]
```

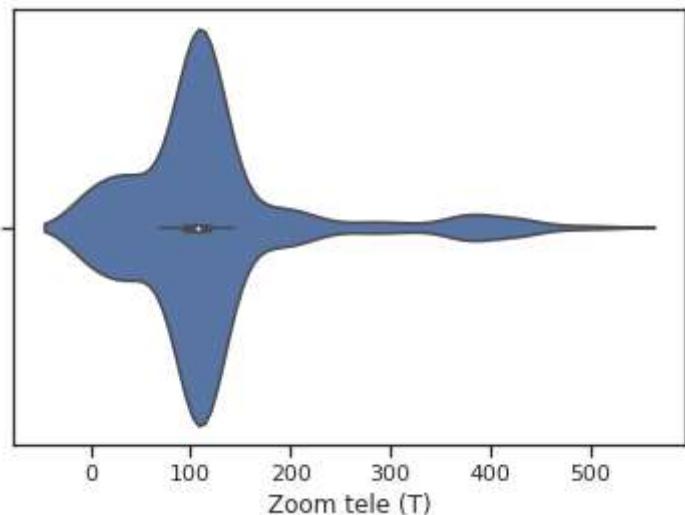
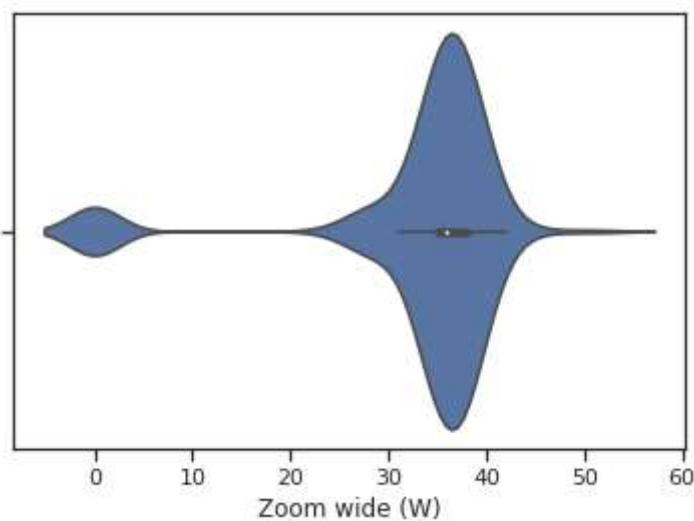
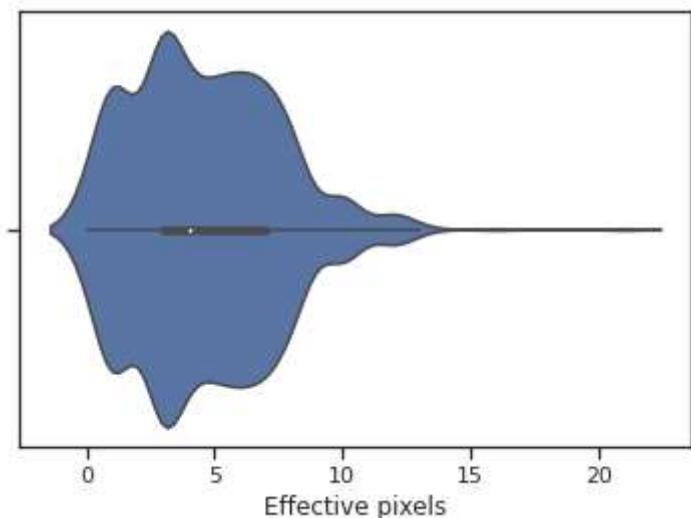
Out[12]:

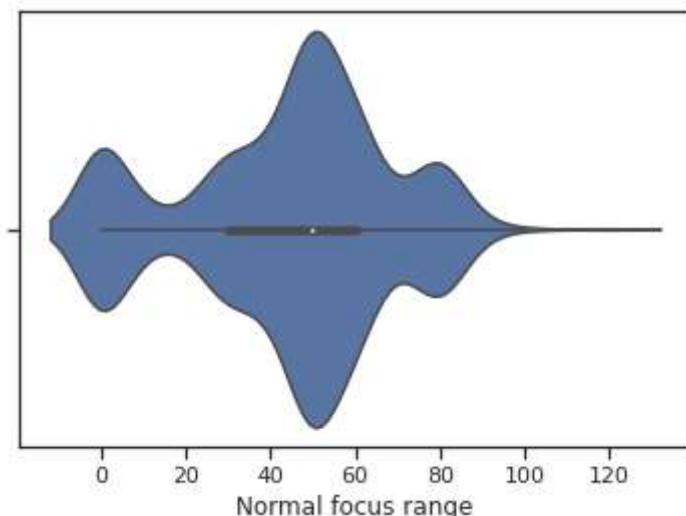
```
Index(['Release date', 'Max resolution', 'Low resolution', 'Effective pixels',  
       'Zoom wide (W)', 'Zoom tele (T)', 'Normal focus range',  
       'Macro focus range', 'Storage included', 'Weight (inc. batteries)',  
       'Dimensions', 'Price'],  
      dtype='object')
```

In [13]:

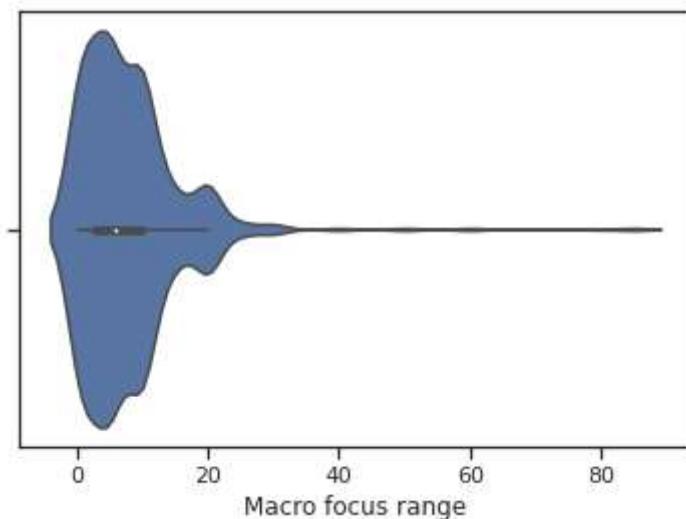
```
# Скрипичные диаграммы для числовых колонок
for col in data.columns[1:]:
    sns.violinplot(x=data[col])
    plt.show()
```



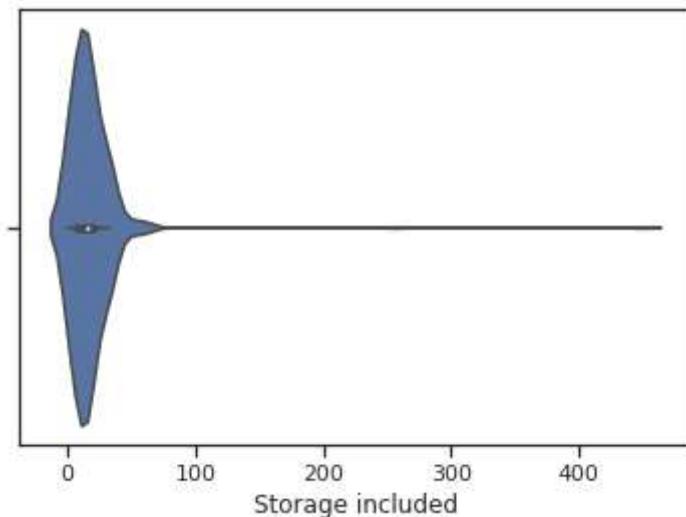




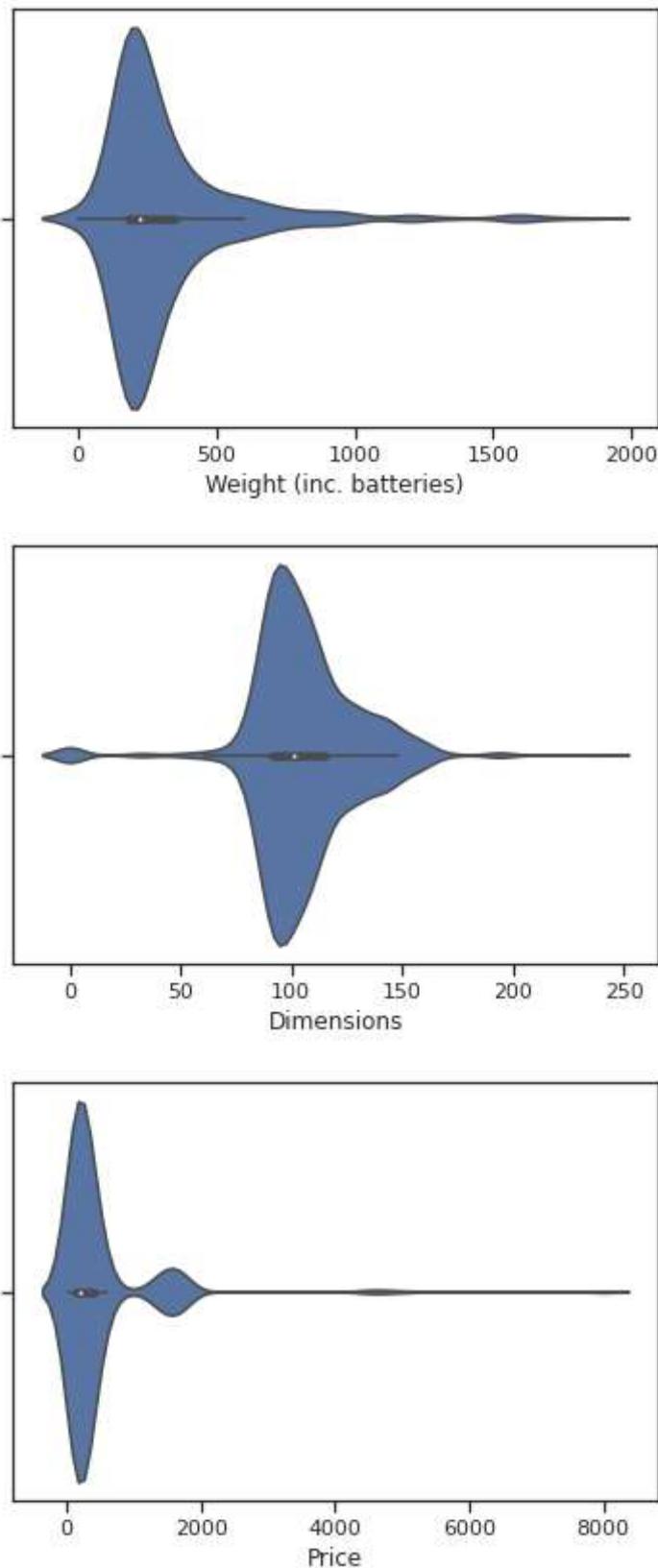
Normal focus range



Macro focus range



Storage included



3) Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

In [14]:

data.dtypes

Out[14]:

Model	object
Release date	int64
Max resolution	float64
Low resolution	float64
Effective pixels	float64
Zoom wide (W)	float64
Zoom tele (T)	float64
Normal focus range	float64
Macro focus range	float64
Storage included	float64
Weight (inc. batteries)	float64
Dimensions	float64
Price	float64
dtype:	object

In [0]:

```
from sklearn.preprocessing import LabelEncoder

# кодирование категориальных признаков числовыми
le = LabelEncoder()
data['Model'] = le.fit_transform(data['Model'])
```

In [16]:

data.head()

Out[16]:

	Model	Release date	Max resolution	Low resolution	Effective pixels	Zoom wide (W)	Zoom tele (T)	Normal focus range	Macro focus range	Storage included
	0	1	2	3	4	5	6	7	8	9
0	0	1997	1024.0	640.0	0.0	38.0	114.0	70.0	40.0	4.0
1	1	1998	1280.0	640.0	1.0	38.0	114.0	50.0	0.0	4.0
2	2	2000	640.0	0.0	0.0	45.0	45.0	0.0	0.0	2.0
3	3	1999	1152.0	640.0	0.0	35.0	35.0	0.0	0.0	4.0
4	4	1999	1152.0	640.0	0.0	43.0	43.0	50.0	0.0	40.0

In [0]:

```
scale_cols = data.columns
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

In [0]:

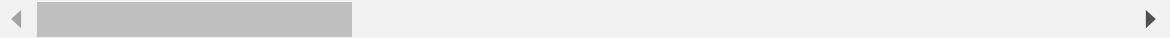
```
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

In [19]:

```
data.head()
```

Out[19]:

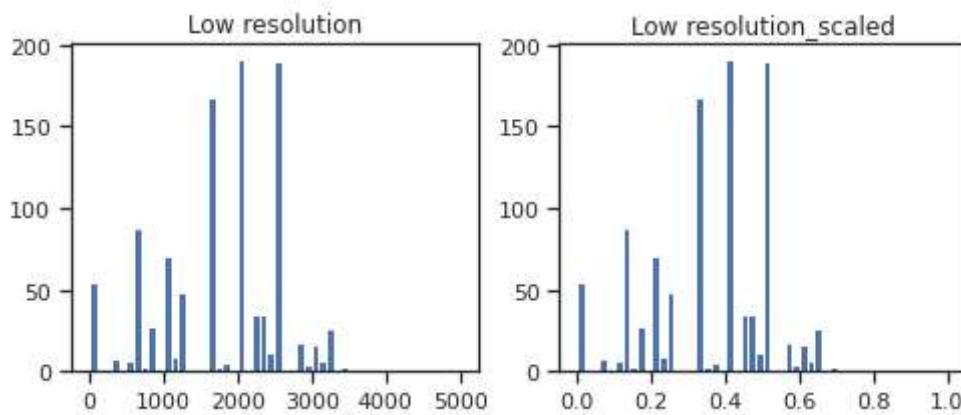
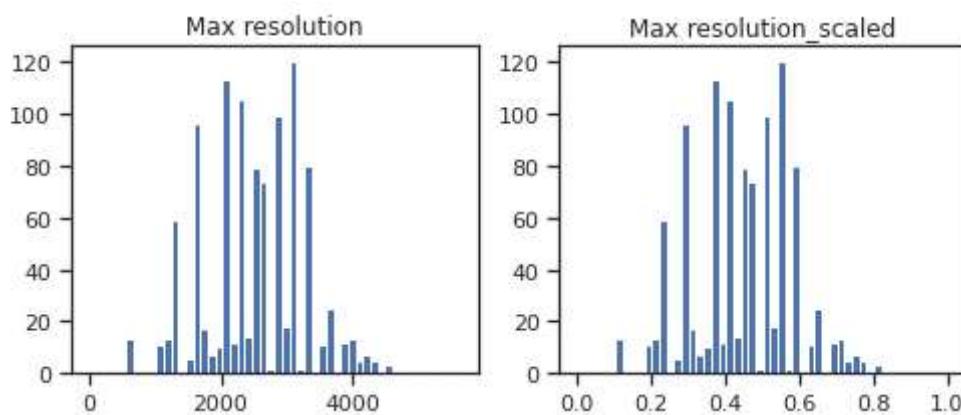
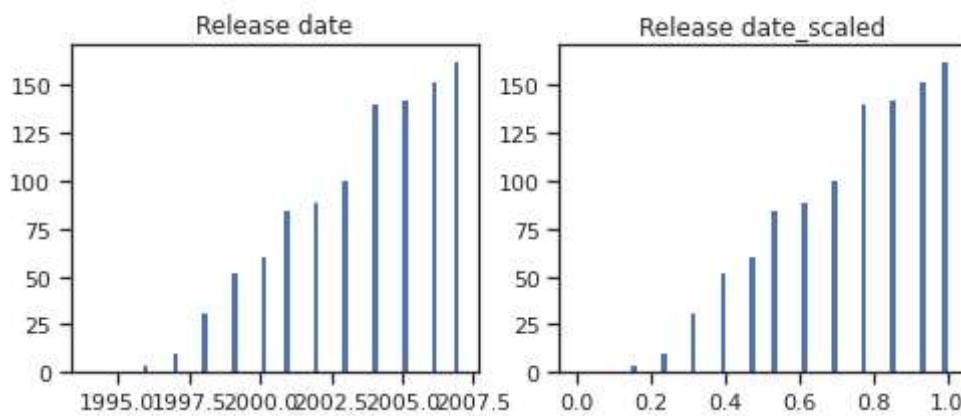
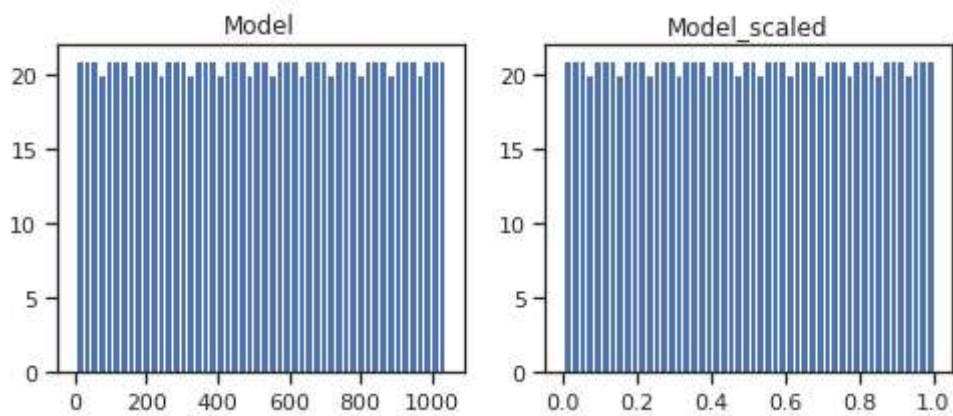
Model	Release date	Max resolution	Low resolution	Effective pixels	Zoom wide (W)	Zoom tele (T)	Normal focus range	Macro focus range	Storage included
0	0	1997	1024.0	640.0	0.0	38.0	114.0	70.0	40.0
1	1	1998	1280.0	640.0	1.0	38.0	114.0	50.0	0.0
2	2	2000	640.0	0.0	0.0	45.0	45.0	0.0	0.0
3	3	1999	1152.0	640.0	0.0	35.0	35.0	0.0	0.0
4	4	1999	1152.0	640.0	0.0	43.0	43.0	50.0	0.0

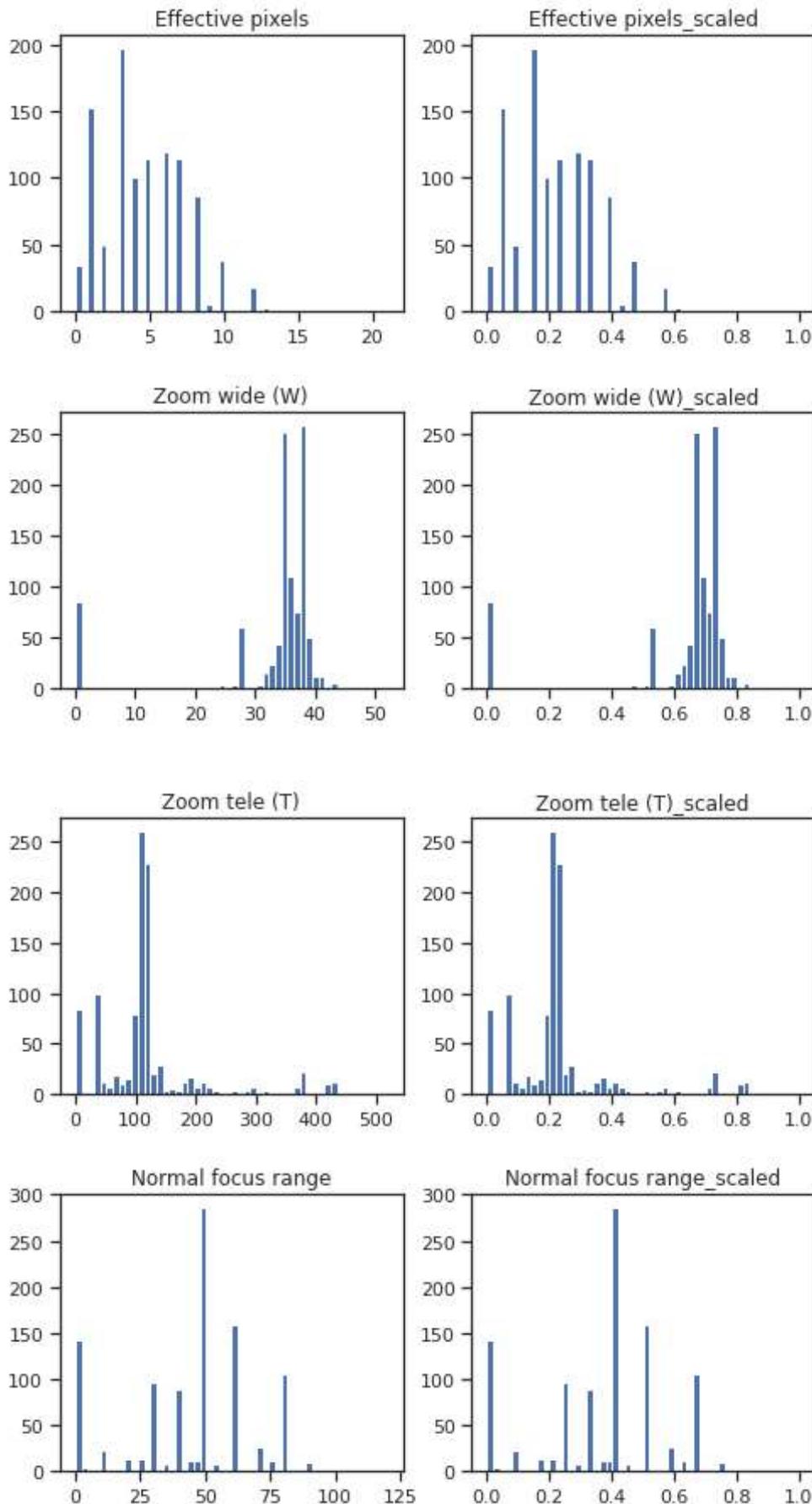


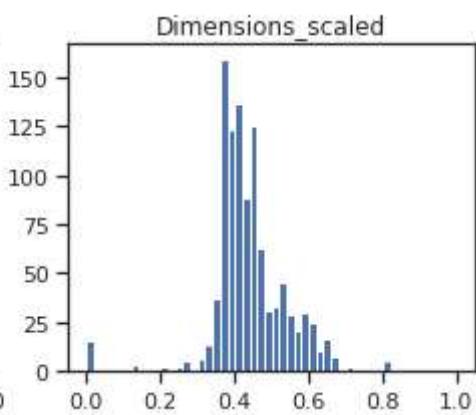
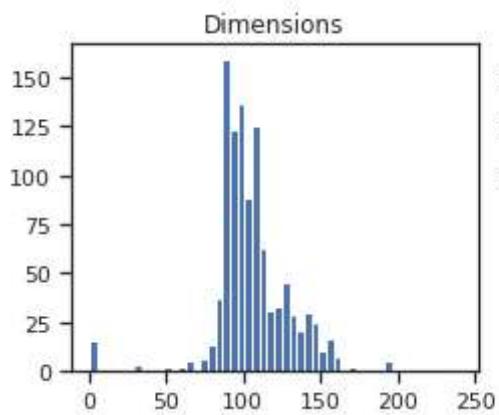
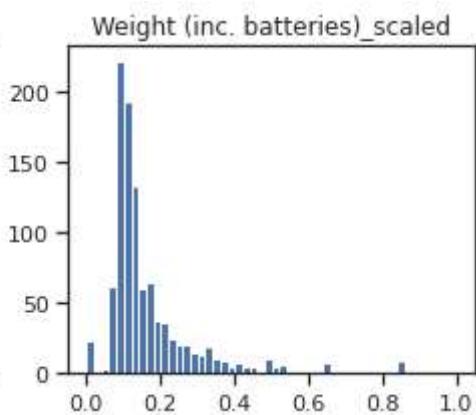
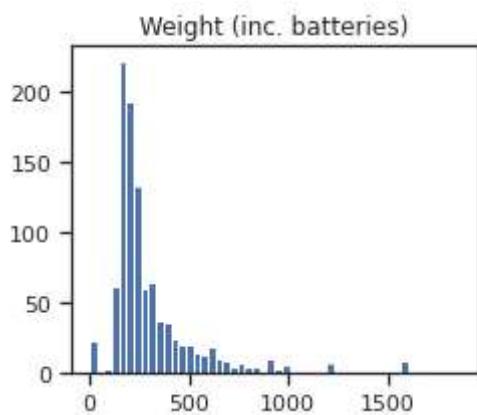
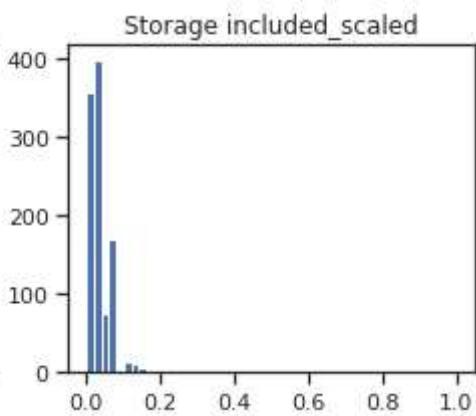
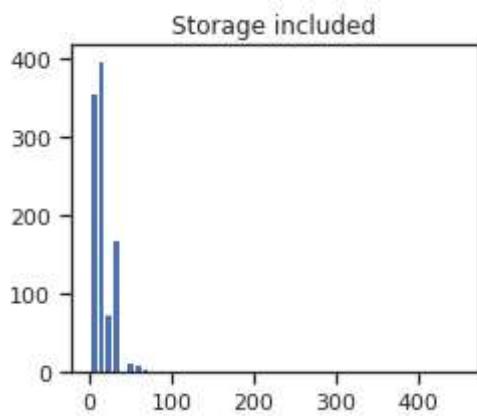
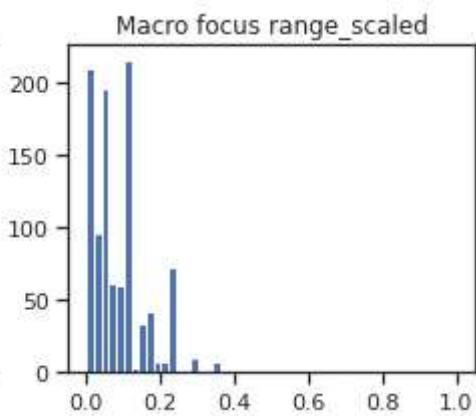
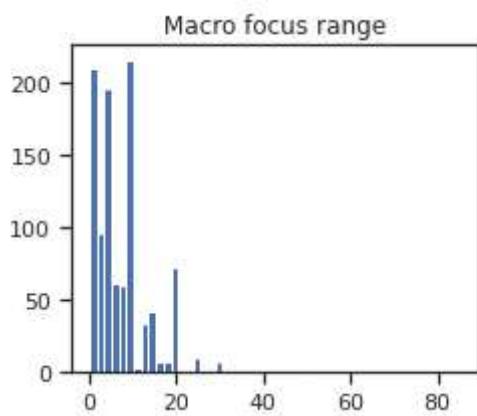
In [20]:

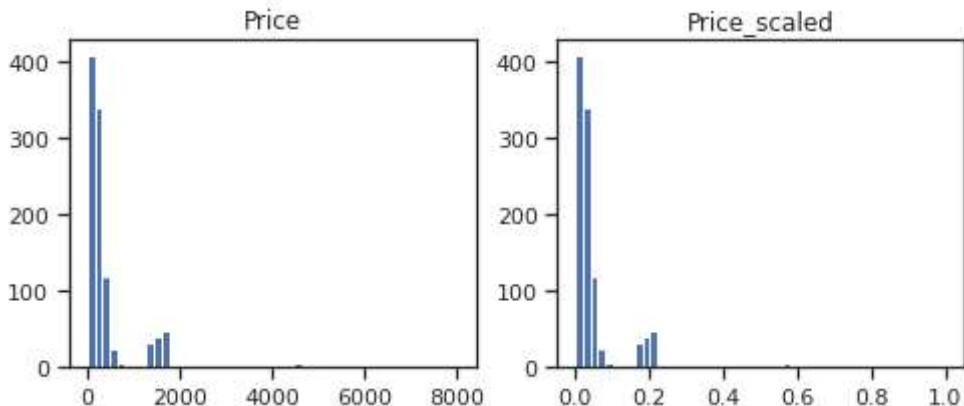
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data[col], 50)
    ax[1].hist(data[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```









4) Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

In [21]:

```
corr_cols_1 = scale_cols
corr_cols_2 = [x+'_scaled' for x in scale_cols]

print(corr_cols_1)
print(corr_cols_2)
```

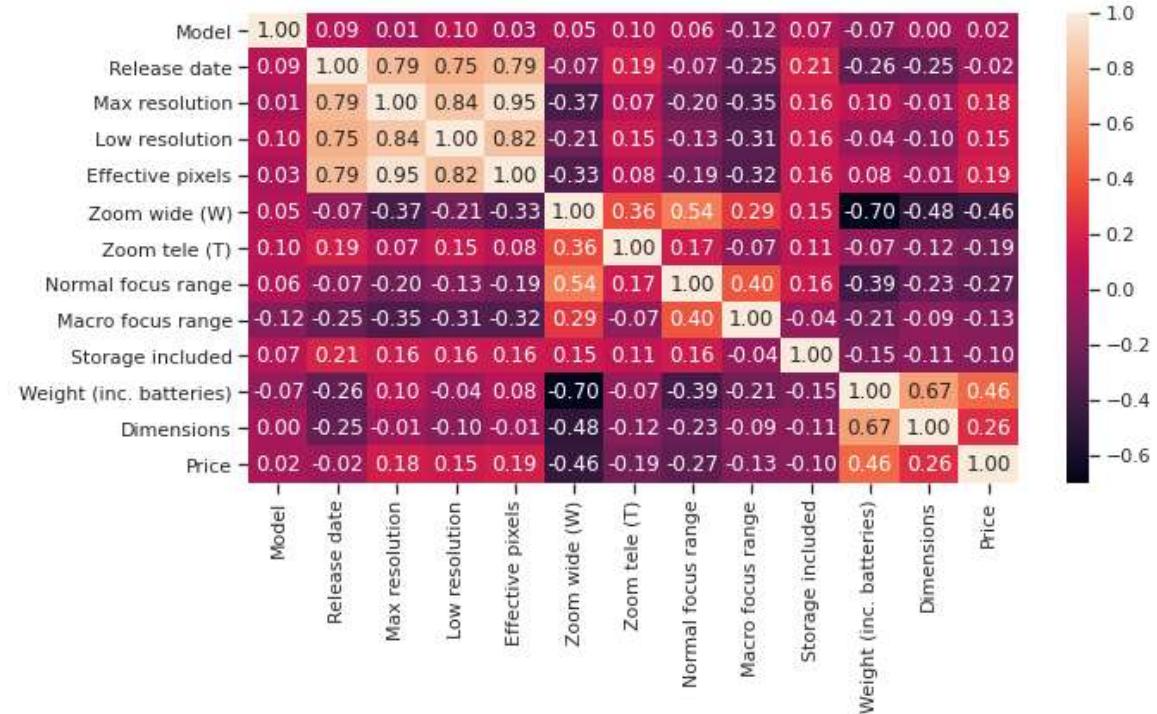
Index(['Model', 'Release date', 'Max resolution', 'Low resolution', 'Effective pixels', 'Zoom wide (W)', 'Zoom tele (T)', 'Normal focus range', 'Macro focus range', 'Storage included', 'Weight (inc. batteries)', 'Dimensions', 'Price'],
 dtype='object')
['Model_scaled', 'Release date_scaled', 'Max resolution_scaled', 'Low resolution_scaled', 'Effective pixels_scaled', 'Zoom wide (W)_scaled', 'Zoom tele (T)_scaled', 'Normal focus range_scaled', 'Macro focus range_scaled', 'Storage included_scaled', 'Weight (inc. batteries)_scaled', 'Dimensions_scaled', 'Price_scaled']

In [22]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

Out[22]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f6deae4f780>

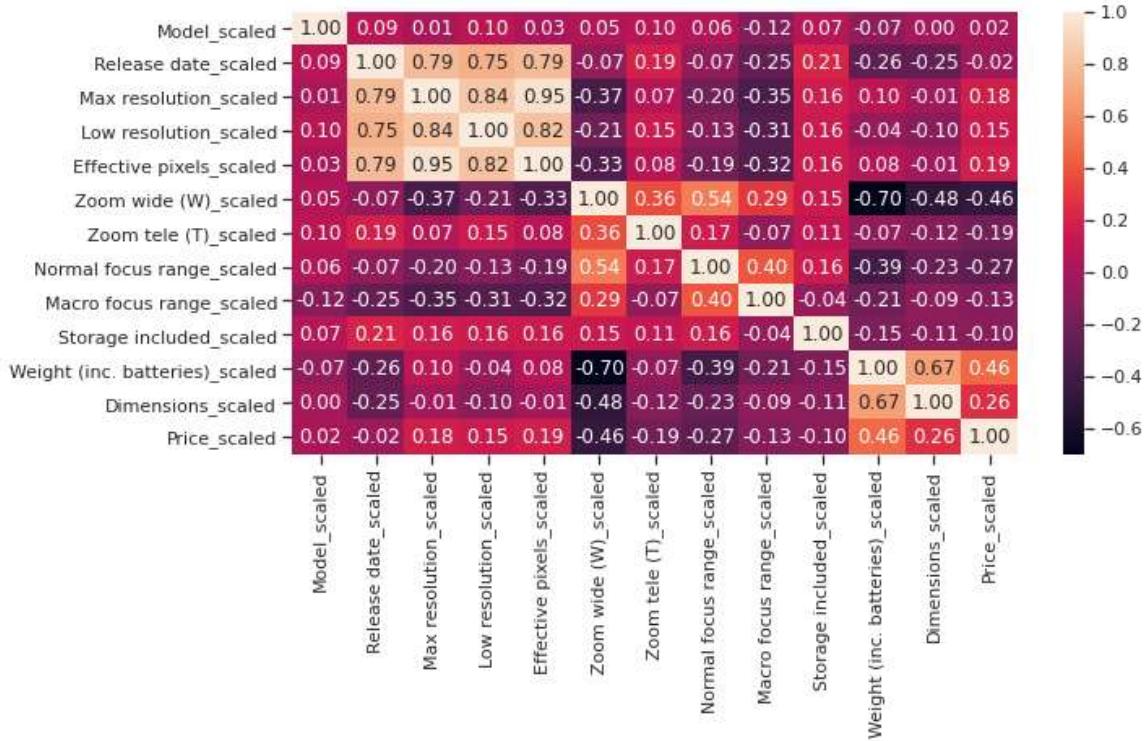


In [23]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
```

Out[23]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f6deac147f0>



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак Price наиболее коррелирует с признаком Weight (inc. batteries) (0.46). Этот признак обязательно следует оставить в модели регрессии.
- Также целевой признак Price имеет отрицательный коэффициент корреляции с признаком Zoom wide.(-0.46). Оставим этот признак в модели.
- С остальными признаками признак Price слабо коррелирован.
- Ряд признаков достаточно сильно коррелируют друг с другом:
 1. Признак Release date достаточно сильно коррелирует с рядом признаков: Max resolution(0.79), Low resolution (0.75), Effective pixels(0.79).
 2. Признак Max resolution достаточно сильно коррелирует с рядом признаков: Release date (0.79), Low resolution (0.84), Effective pixels(0.95).
 3. Признак Low resolution достаточно сильно коррелирует с рядом признаков: Release date (0.75), Max resolution (0.84), Effective pixels(0.82).
 4. Признак Effective pixels достаточно сильно коррелирует с рядом признаков: Release date (0.79), Max resolution (0.95), Low resolution(0.82).
 5. Признак Weight отрицательно коррелирован с признаком Zoom wide(-0.7).
 6. Признак Weight коррелирован с признаком Dimensions(0.67).

Из этих признаков можно оставить в модели следующие признаки: Max resolution, Effective pixels, Weight и Zoom wide так как они более, чем остальные коррелируют с целевым признаком.

- На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

5) Выбор метрик для последующей оценки качества моделей.

In [0]:

```

class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
             'alg': pd.Series([], dtype='str'),
             'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index,
inplace = True)
        # Добавление нового значения
        temp = [{ 'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Выход графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barrh(pos, array_metric,
                           align='center',
                           height=0.5,
                           tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()

```

6) Выбор наиболее подходящих моделей для решения задачи классификации или регрессии.

Для задачи регрессии будем использовать следующие модели:

- Линейная регрессия
- Метод ближайших соседей
- Машина опорных векторов
- Решающее дерево
- Случайный лес
- Градиентный бустинг

7) Формирование обучающей и тестовой выборок на основе исходного набора данных.

In [0]:

```
# Перейдем к разделению выборки на обучающую и тестовую.  
X = data.drop('Price',axis = 1).values  
y = data['Price'].values  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)
```

In [27]:

```
print(X_train.shape)  
print(X_test.shape)  
print(y_train.shape)  
print(y_test.shape)
```

```
(934, 25)  
(104, 25)  
(934,)  
(104,)
```

8) Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

In [0]:

```
# Модели  
regr_models = {'LR': LinearRegression(),  
               'KNN_5': KNeighborsRegressor(n_neighbors=5),  
               'SVR': SVR(),  
               'Tree': DecisionTreeRegressor(),  
               'RF': RandomForestRegressor(),  
               'GB': GradientBoostingRegressor() }
```

In [0]:

```
# Сохранение метрик  
regrMetricLogger = MetricLogger()
```

In [0]:

```
def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(X_train, y_train)
    Y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, Y_pred)
    mse = mean_squared_error(y_test, Y_pred)
    r2 = r2_score(y_test, Y_pred)

    regrMetricLogger.add('MAE', model_name, mae)
    regrMetricLogger.add('MSE', model_name, mse)
    regrMetricLogger.add('R2', model_name, r2)

    print('*****')
    print(model)
    print()
    print('MAE={}, MSE={}, R2={}'.format(
        round(mae, 3), round(mse, 3), round(r2, 3)))
    print('*****')
```

In [31]:

```
for model_name, model in regr_models.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
*****
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

MAE=0.0, MSE=0.0, R2=1.0
*****
*****
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')

MAE=221.048, MSE=146379.841, R2=0.591
*****
*****
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
     kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

MAE=269.3, MSE=401427.07, R2=-0.122
*****
*****
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')

MAE=0.0, MSE=0.0, R2=1.0
*****
*****
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)

MAE=3.443, MSE=527.949, R2=0.999
*****
*****
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                           init=None, learning_rate=0.1, loss='ls', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0, warm_start=False)

MAE=0.11, MSE=0.099, R2=1.0
*****
```

9) Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.

In [32]:

```
n_range = np.array(range(1,700,100))
tuned_parameters = [{n_neighbors: n_range}]
tuned_parameters
```

Out[32]:

```
[{n_neighbors: array([ 1, 101, 201, 301, 401, 501, 601])}]
```

In [33]:

```
%time
regr_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5, scoring='neg_mean_squared_error')
regr_gs.fit(X_train, y_train)
```

CPU times: user 544 ms, sys: 228 ms, total: 772 ms
Wall time: 501 ms

In [34]:

```
regr_gs.best_estimator_
```

Out[34]:

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [35]:

```
# Лучшее значение параметров
regr_gs.best_params_
```

Out[35]:

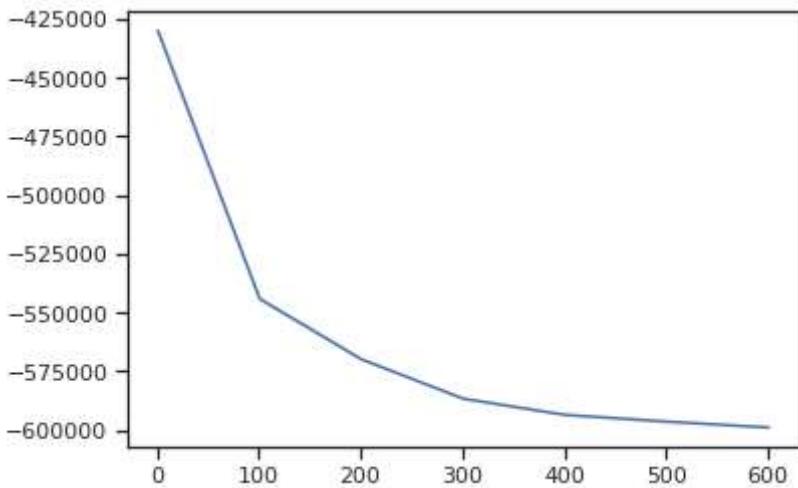
```
{n_neighbors: 1}
```

In [36]:

```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, regr_gs.cv_results_['mean_test_score'])
```

Out[36]:

[<matplotlib.lines.Line2D at 0x7f6dea3fadd8>]



10) Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

In [0]:

```
regr_models_grid = {'KNN_801':regr_gs.best_estimator_}
```

In [38]:

```
for model_name, model in regr_models_grid.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
*****
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

```
MAE=164.163, MSE=218150.567, R2=0.39
```

```
*****
```

11) Формирование выводов о качестве построенных моделей на основе выбранных метрик.

Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпараметров на качество моделей и т.д.

In [39]:

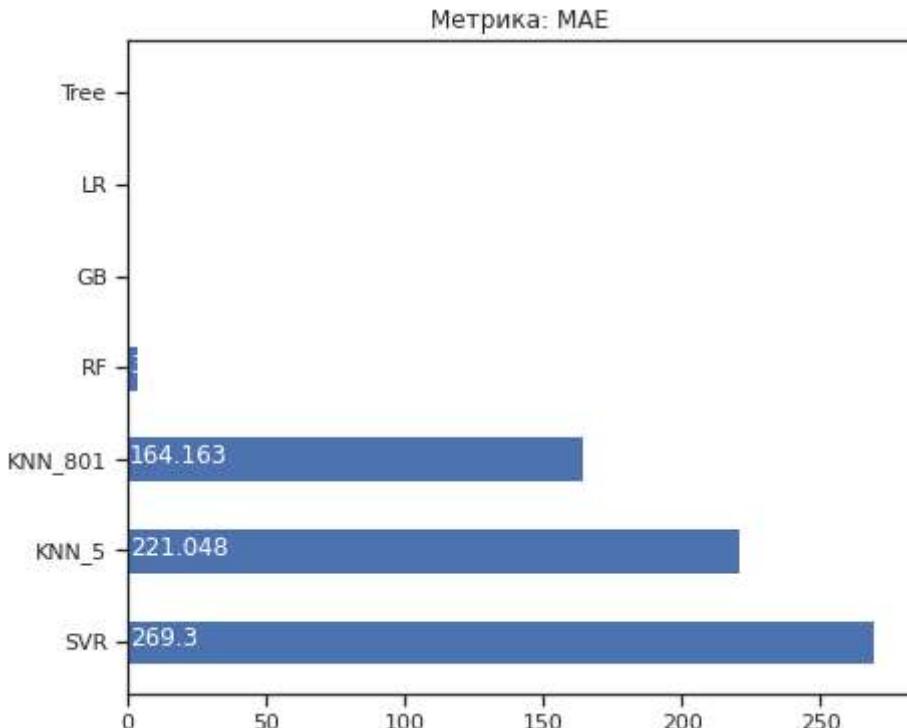
```
# Метрики качества модели
regr_metrics = regrMetricLogger.df['metric'].unique()
regr_metrics
```

Out[39]:

```
array(['MAE', 'MSE', 'R2'], dtype=object)
```

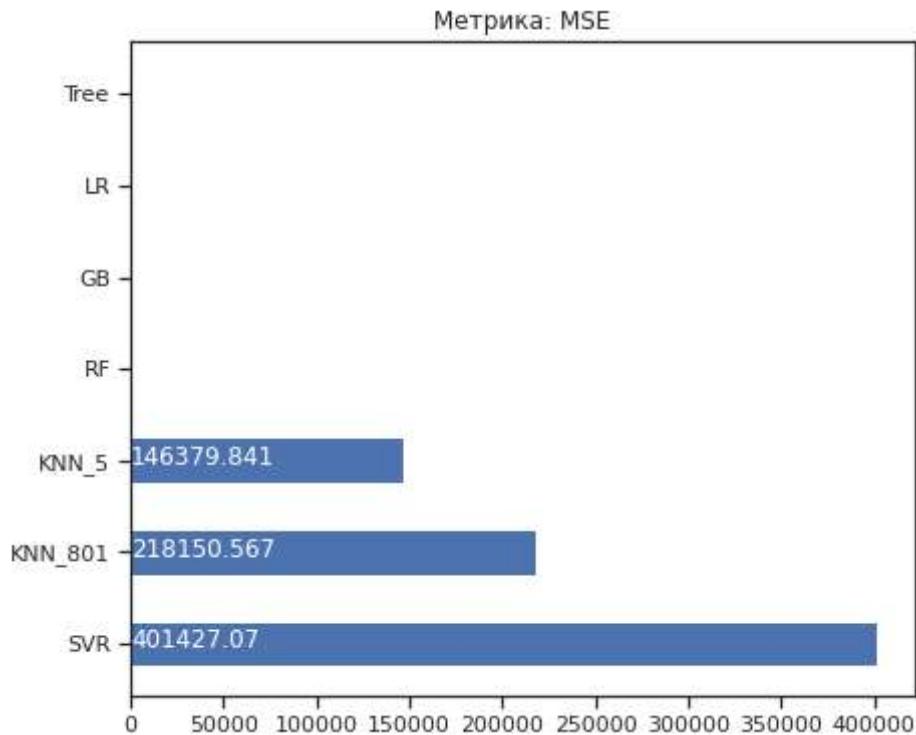
In [40]:

```
regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False, figsize=(7, 6))
```



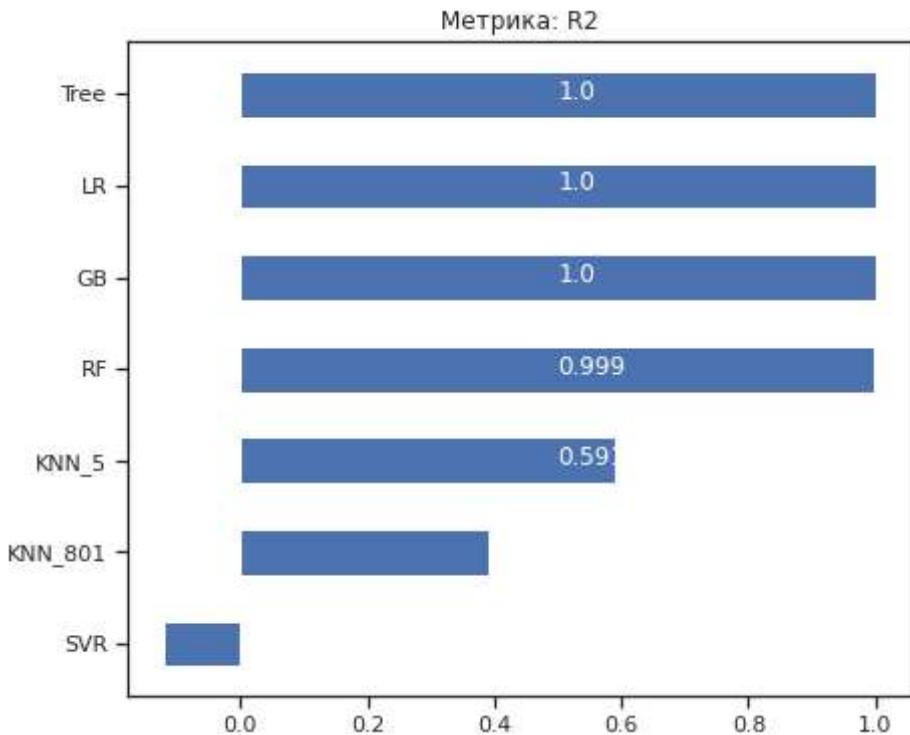
In [41]:

```
regrMetricLogger.plot('Метрика: ' + 'MSE', 'MSE', ascending=False, figsize=(7, 6))
```



In [42]:

```
regrMetricLogger.plot('Метрика: ' + 'R2', 'R2', ascending=True, figsize=(7, 6))
```



Вывод: лучшими оказались модели на основе линейной регрессии(LR), градиентного бустинга(GB) и решающего дерева(Tree). Также неплохо будет работать модель случайного леса(RF).