

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

По домашнему заданию
по курсу «Методы машинного обучения»

ИСПОЛНИТЕЛЬ: Красавцев К.Ю.
группа ИУ5-21М

подпись

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю.Е.

подпись

"__" _____ 2020 г.

Москва - 2020

Задание

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.

Решение

Решение

```
In [1]: 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,
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 sklearn.utils import shuffle
# !pip install gmdhpy
from gmdhpy import gmdh
%matplotlib inline
sns.set(style="ticks")
```

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных построение модели машинного обучения для решения или задачи регрессии.

В качестве набора данных возьмем набор с данными о песнях и их характеристиках.

Набор содержит такие колонки как:

- song_name - название песни
- song_popularity - индекс популярности песни
- song_duration_ms - длительность в мс
- acousticness - индекс акустики
- danceability - индекс танцевальности
- energy - индекс энергичности
- instrumentalness - индекс инструментальности
- key - ключ
- liveness - индекс живости
- loudness - индекс громкости
- audio_mode - режим аудио
- speechiness - индекс разговорности
- tempo - темп
- time_signature - временная метка
- audio_valence

Поставим задачу предсказания популярности песни по данным характеристикам. Построим модель машинного обучения для данного набора и решим задачу регрессии.

2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

```
In [2]: data = pd.read_csv('song_data.csv', sep=',')
data.head()

Out[2]:
   song_name  song_popularity  song_duration_ms  acousticness  danceability  energy  instrumentalness  key  liveness  loudness  audio_mode  speechiness
0  Boulevard of Broken Dreams          73        262333    0.005520      0.496    0.682       0.000029     8    0.0589    -4.095        1     0.029
1  In The End                      66        216933    0.010300      0.542    0.853       0.000000     3    0.1080    -6.407        0     0.049
2  Seven Nation Army                  76        231733    0.008170      0.737    0.463       0.447000     0    0.2550    -7.828        1     0.079
3  By The Way                     74        216933    0.026400      0.451    0.970       0.003550     0    0.1020    -4.938        1     0.107
4  How You Remind Me                  56        223826    0.000954      0.447    0.766       0.000000    10    0.1130    -5.065        1     0.031
.. . . . .
```

```
In [3]: data.shape

Out[3]: (18835, 15)

In [4]: data.columns

Out[4]: Index(['song_name', 'song_popularity', 'song_duration_ms', 'acousticness',
   'danceability', 'energy', 'instrumentalness', 'key', 'liveness',
   'loudness', 'audio_mode', 'speechiness', 'tempo', 'time_signature',
   'audio_valence'],
  dtype='object')

In [5]: data.isnull().sum()

Out[5]:
song_name      0
song_popularity 0
song_duration_ms 0
acousticness 0
danceability 0
energy 0
instrumentalness 0
key 0
liveness 0
loudness 0
audio_mode 0
speechiness 0
tempo 0
time_signature 0
audio_valence 0
dtype: int64

In [6]: data.dtypes

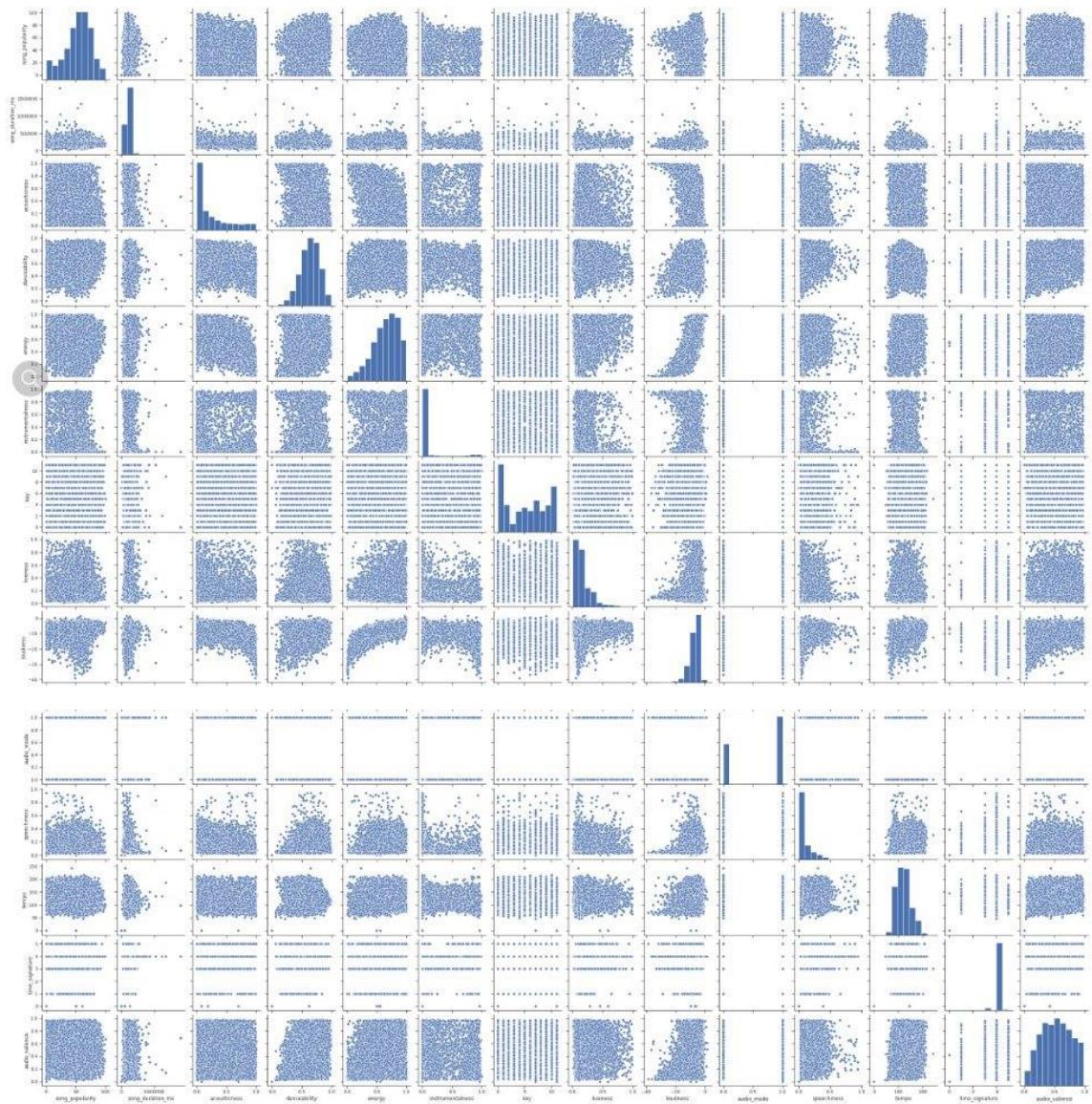
Out[6]:
song_name      object
song_popularity  int64
song_duration_ms  int64
acousticness  float64
danceability  float64
energy  float64
instrumentalness  float64
key  int64
liveness  float64
loudness  float64
audio_mode  int64
speechiness  float64
tempo  float64
time_signature  int64
audio_valence  float64
dtype: object

In [7]: data.describe()

Out[7]:
   song_popularity  song_duration_ms  acousticness  danceability  energy  instrumentalness  key  liveness  loudness  audio_r
count  18835.000000  1.883500e+04  18835.000000  18835.000000  18835.000000  18835.000000  18835.000000  18835.000000  18835.000000
mean   52.991877  2.182116e+05  0.258539  0.633348  0.644995  0.078008  5.289196  0.179650  -7.447435  0.62
std    21.905654  5.988754e+04  0.288719  0.156723  0.214101  0.221591  3.614595  0.143984  3.827831  0.48
min    0.000000  1.200000e+04  0.000001  0.000000  0.001070  0.000000  0.000000  0.010900  -38.768000  0.00
25%   40.000000  1.843395e+05  0.024100  0.533000  0.510000  0.000000  2.000000  0.092900  -9.044000  0.00
50%   56.000000  2.113060e+05  0.132000  0.645000  0.674000  0.000011  5.000000  0.122000  -6.555000  1.00
75%   69.000000  2.428440e+05  0.424000  0.748000  0.815000  0.002570  8.000000  0.221000  -4.908000  1.00
max   100.000000  1.799346e+06  0.996000  0.987000  0.999000  0.997000  11.000000  0.986000  1.585000  1.00
```

```
In [8]: sns.pairplot(data)
```

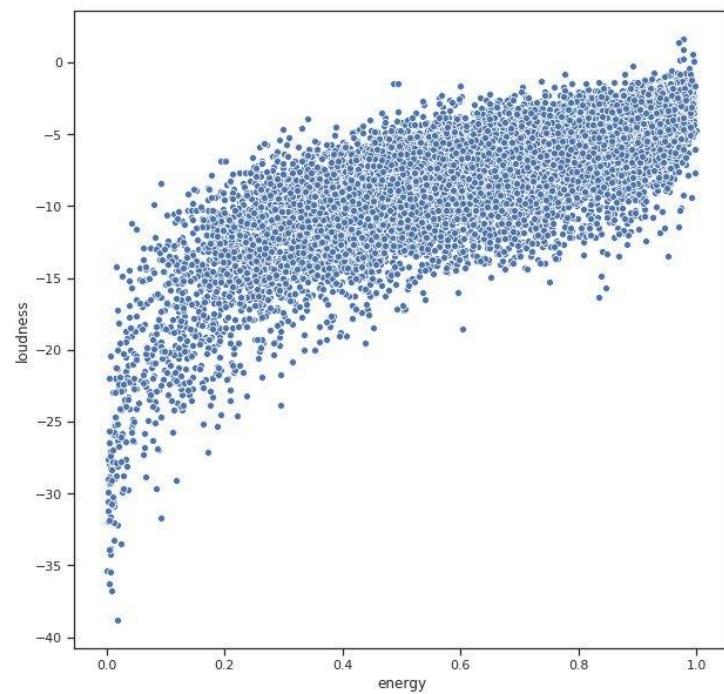
```
Out[8]: <seaborn.axisgrid.PairGrid at 0x7fe16448ae20>
```



Видим, что наиболее заметна корреляция таких характеристик как громкость и энергичность. В остальных случаях зависимости не такие очевидные.

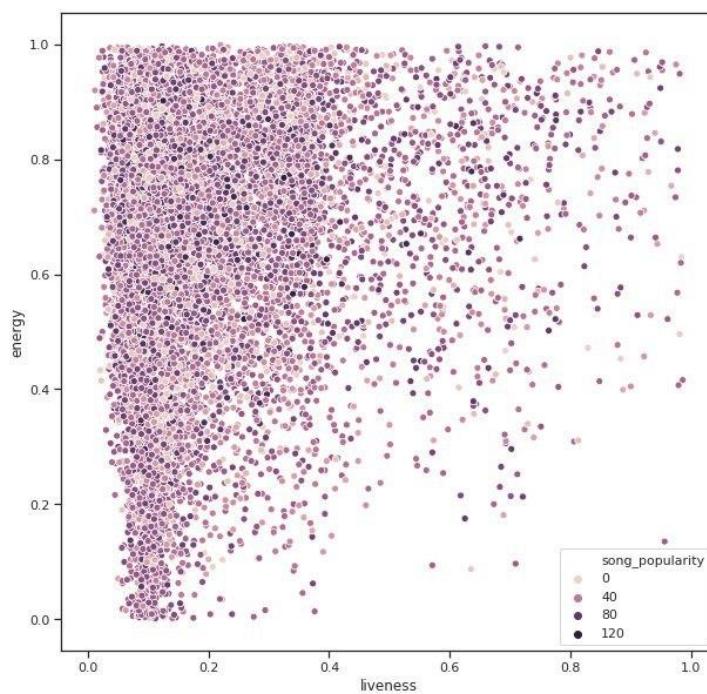
```
In [9]: fig, ax = plt.subplots(figsize=(10,10))
sns.scatterplot(ax=ax, x='energy', y='loudness', data=data)

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe15f4e94f0>
```



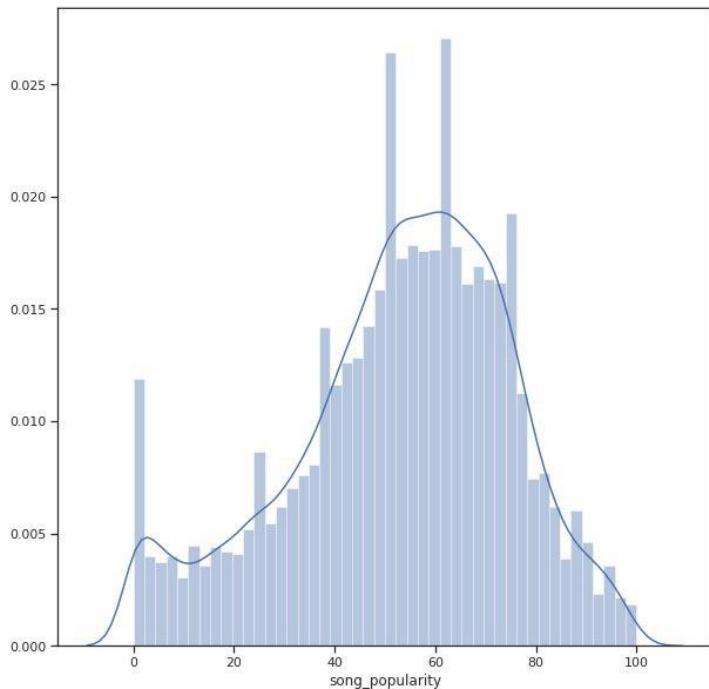
```
In [10]: fig, ax = plt.subplots(figsize=(10,10))
sns.scatterplot(ax=ax, x='liveness', y='energy', data=data, hue='song_popularity')

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe15fd666d0>
```



```
In [11]: fig, ax = plt.subplots(figsize=(10,10))
sns.distplot(data['song_popularity'])

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe15e896160>
```

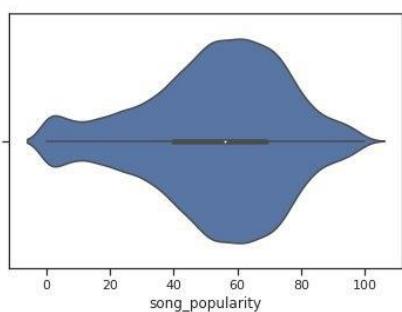


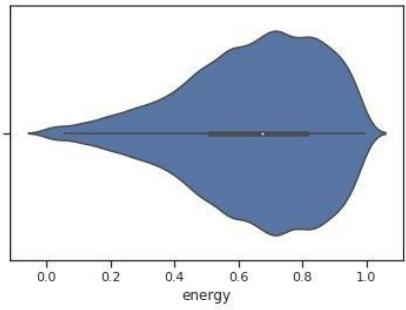
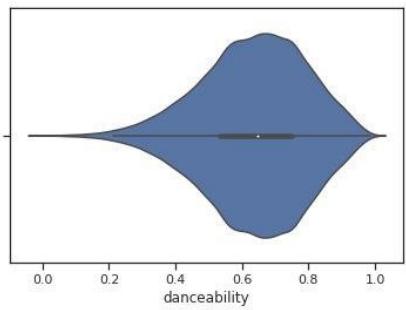
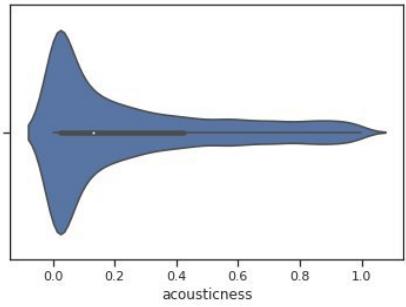
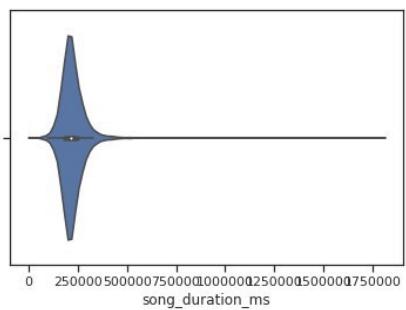
```
In [12]: data.columns

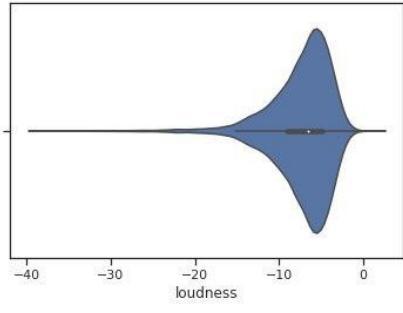
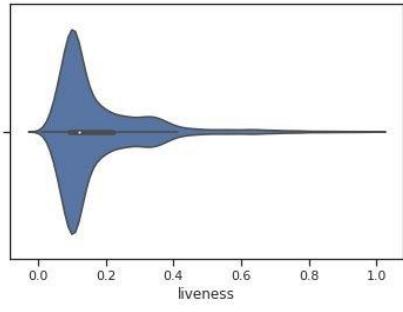
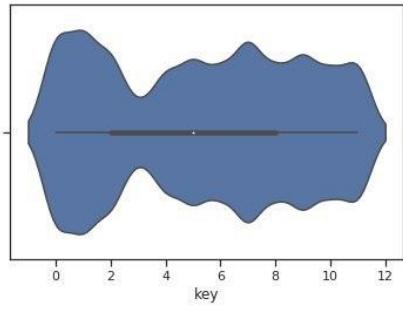
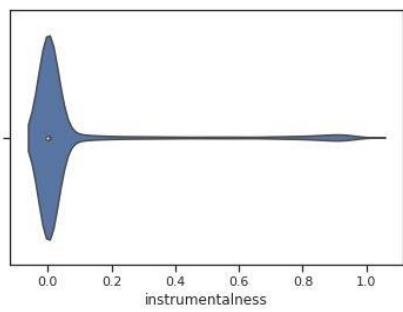
Out[12]: Index(['song_name', 'song_popularity', 'song_duration_ms', 'acousticness',
       'danceability', 'energy', 'instrumentalness', 'key', 'liveness',
       'loudness', 'audio_mode', 'speechiness', 'tempo', 'time_signature',
       'audio_valence'],
      dtype='object')
```

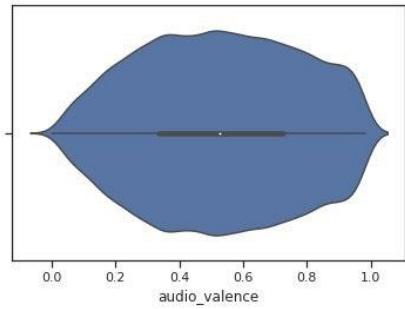
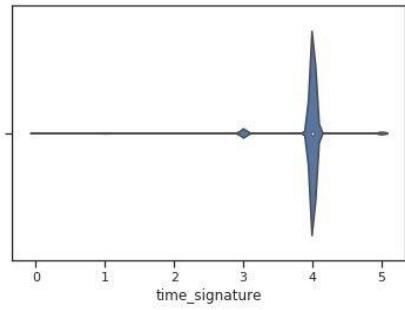
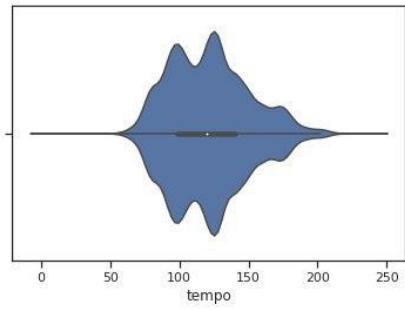
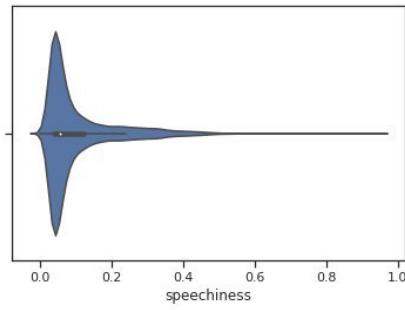
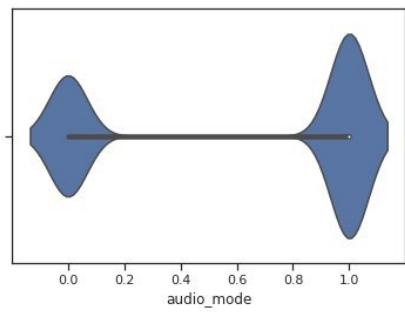
```
In [13]: # Скрипичные диаграммы для числовых колонок
for col in [ 'song_popularity', 'song_duration_ms', 'acousticness',
             'danceability', 'energy', 'instrumentalness', 'key', 'liveness',
             'loudness', 'audio_mode', 'speechiness', 'tempo', 'time_signature',
             'audio_valence']:

    sns.violinplot(x=data[col])
    plt.show()
```









Анализ и заполнение пропусков в данных.

Поскольку в данном наборе пустых значений нет, пропустим данный пункт.

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

Кодирование категориальных признаков числовыми

```
In [14]: from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder()  
data['song_name'] = le.fit_transform(data['song_name'])  
data.dtypes
```

```
Out[14]: song_name      int64  
song_popularity    int64  
song_duration_ms   int64  
acousticness       float64  
danceability        float64  
energy              float64  
instrumentalness   float64  
key                 int64  
liveness            float64  
loudness            float64  
audio_mode          int64  
speechiness         float64  
tempo               float64  
time_signature      int64  
audio_valence       float64  
dtype: object
```

```
In [15]: data.head()
```

```
Out[15]:
```

	song_name	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	audio_mode	speechiness
0	1561	73	262333	0.005520	0.496	0.682	0.000029	8	0.0589	-4.095	1	0.029
1	5541	66	216933	0.010300	0.542	0.853	0.000000	3	0.1080	-6.407	0	0.049
2	9638	76	231733	0.008170	0.737	0.463	0.447000	0	0.2550	-7.828	1	0.079
3	1760	74	216933	0.026400	0.451	0.970	0.003550	0	0.1020	-4.938	1	0.107
4	4988	56	223826	0.000954	0.447	0.766	0.000000	10	0.1130	-5.065	1	0.031

Масштабирование данных.

```
In [16]: scale_cols = ['song_popularity', 'song_duration_ms', 'acousticness',  
'danceability', 'energy', 'instrumentalness', 'key', 'liveness',  
'loudness', 'audio_mode', 'speechiness', 'tempo', 'time_signature',  
'audio_valence']
```

```
In [17]: data.columns
```

```
Out[17]: Index(['song_name', 'song_popularity', 'song_duration_ms', 'acousticness',  
'danceability', 'energy', 'instrumentalness', 'key', 'liveness',  
'loudness', 'audio_mode', 'speechiness', 'tempo', 'time_signature',  
'audio_valence'],  
dtype='object')
```

```
In [18]: scl = MinMaxScaler()  
scl_data = scl.fit_transform(data[scale_cols])
```

```
In [19]: # 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] = scl_data[:,i]
```

```
In [20]: data.head()
```

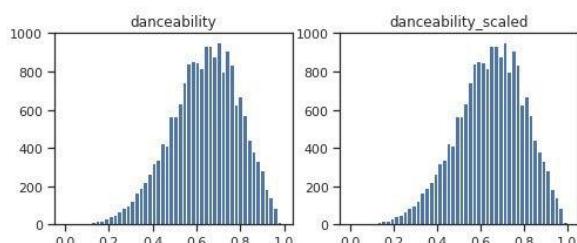
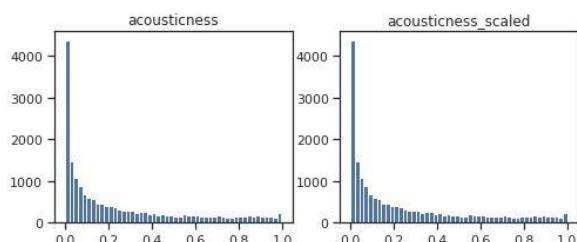
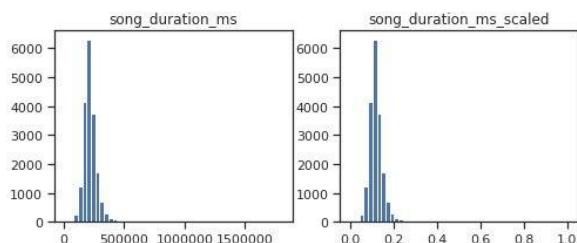
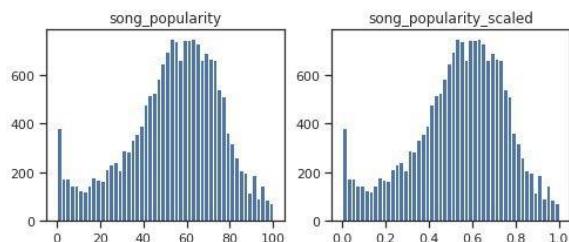
```
Out[20]:
```

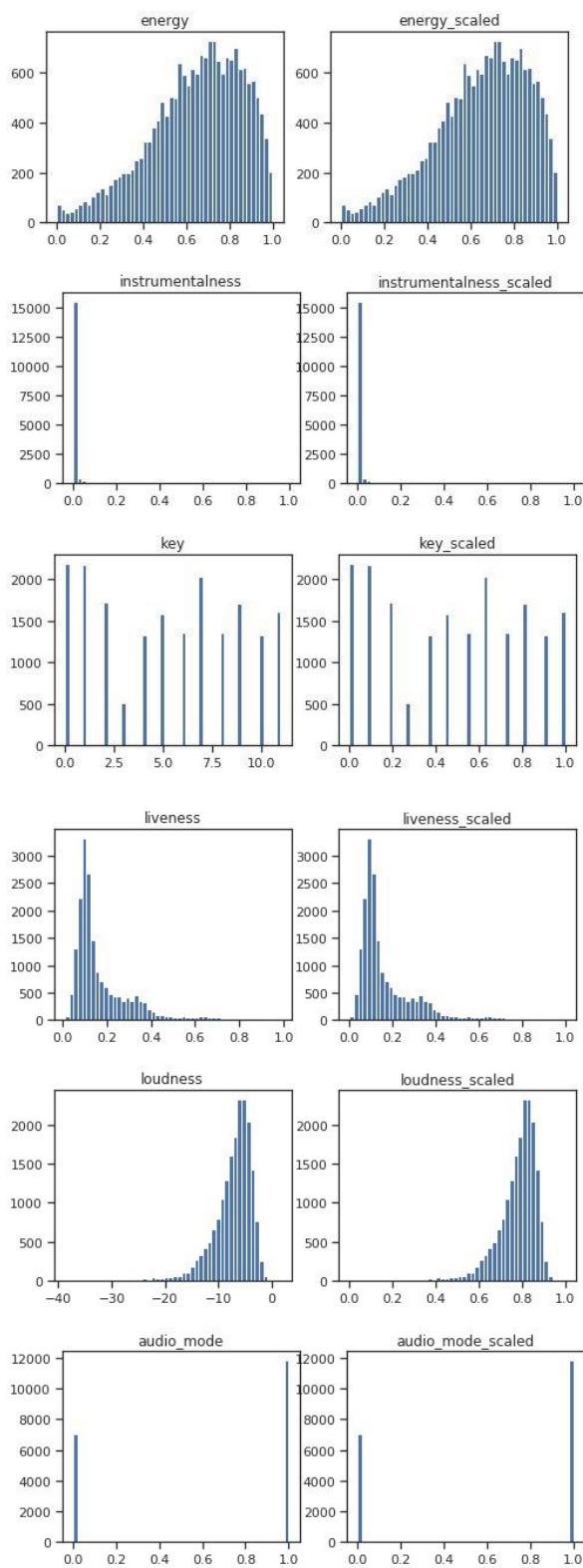
	song_name	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	... energy_scaled	instr
0	1561	73	262333	0.005520	0.496	0.682	0.000029	8	0.0589	-4.095	...	0.682342
1	5541	66	216933	0.010300	0.542	0.853	0.000000	3	0.1080	-6.407	...	0.853697
2	9638	76	231733	0.008170	0.737	0.463	0.447000	0	0.2550	-7.828	...	0.462888
3	1760	74	216933	0.026400	0.451	0.970	0.003550	0	0.1020	-4.938	...	0.970940
4	4988	56	223826	0.000954	0.447	0.766	0.000000	10	0.1130	-5.065	...	0.766517

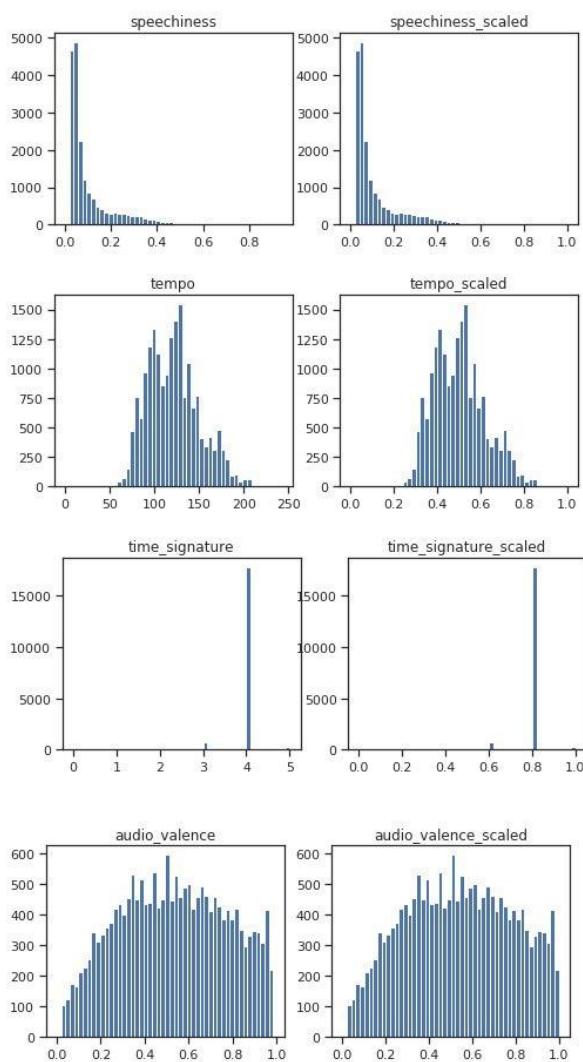
5 rows × 29 columns

```
In [21]: # 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 [22]: # Воспользуемся наличием тестовых выборок,
# включив их в корреляционную матрицу
corr_cols_1 = scale_cols
corr_cols_1
```

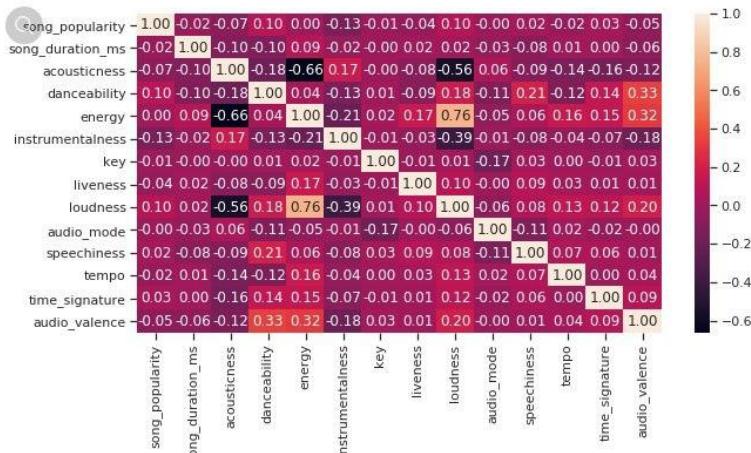
```
Out[22]: ['song_popularity',
'song_duration_ms',
'acousticness',
'danceability',
'energy',
'instrumentalness',
'key',
'liveness',
'loudness',
'audio_mode',
'speechiness',
'tempo',
'time_signature',
'audio_valence']
```

```
In [23]: scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix
corr_cols_2
```

```
Out[23]: ['song_popularity_scaled',
'song_duration_ms_scaled',
'acousticness_scaled',
'danceability_scaled',
'energy_scaled',
'instrumentalness_scaled',
'key_scaled',
'liveness_scaled',
'loudness_scaled',
'audio_mode_scaled',
'speechiness_scaled',
'tempo_scaled',
'time_signature_scaled',
'audio_valence_scaled']
```

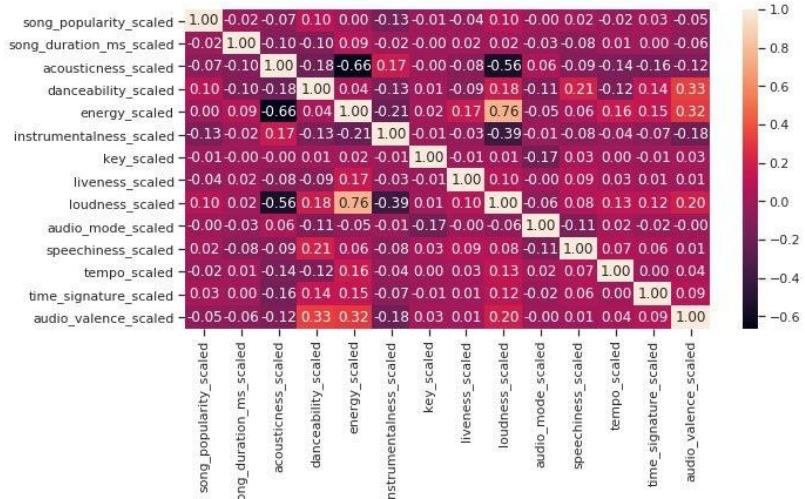
```
In [24]: fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt=".2f")
```

```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe15f3f81c0>
```



```
In [25]: fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt=".2f")
```

```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe15d8c520>
```



- Видим, что популярность песни не сильно коррелирует с данными характеристиками. Наибольшее влияние на популярность оказывают такие признаки как танцевальность трека и громкость.
- Наибольшую корреляцию видим между громкостью и энергичностью трека, как и во 2 пункте.

5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.

Возьмем метрики MAE, Median Absolute Error и R².

- MAE (Mean Absolute Error) — это среднее абсолютное значение ошибки(среднее модуля ошибки). Данная метрика удобна, так как показывает среднюю ошибку, но при этом не так чувствительна к выбросам, как, например, MSE.
- Медиана абсолютного отклонения(Median Absolute Error) - это альтернатива стандартного отклонения, но она менее чувствительна к воздействию промахов, чем среднее отклонение.
- Коэффициент детерминации, или R² покажет насколько модель соответствует или не соответствует данным.

```
In [26]: 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)):
        """
        Вывод 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.barh(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 регрессии.

- Возьмем модели случайный лес и дерево решений, поскольку в проведенных экспериментах в лабораторных работах случайный лес показал себя наилучшим образом. Результаты, которые удалось получить при помощи данной модели были сопоставимы с результатами самых сильных среди протестированных ансамблевых моделей. Дерево решений так же дает хорошие результаты по сравнению с, например, линейными моделями.
- В качестве ансамблевой модели возьмем лучшую модель, полученную при выполнении 6 лабораторной работы: 'TREE+RF=>LR', то есть на первом уровне у нас будут две модели: дерево и случайный лес, а на втором уровне - линейная регрессия.

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

```
In [27]: data1 = shuffle(data)
data1
```

Out[27]:

	song_name	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	... energy_scaled	i
4223	3044	43	250600	0.64300	0.352	0.339	0.003800	4	0.3580	-14.335	...	0.338631
536	2953	59	354640	0.22900	0.360	0.527	0.000792	9	0.0536	-9.883	...	0.527021
5722	2579	29	213320	0.00104	0.690	0.871	0.486000	6	0.2870	-7.445	...	0.871734
2436	2916	72	265120	0.84100	0.397	0.172	0.000068	6	0.1080	-15.698	...	0.171285
12683	5901	52	234706	0.05010	0.761	0.577	0.034400	1	0.1020	-6.770	...	0.577125
...
22	7843	13	255066	0.01370	0.518	0.538	0.000398	0	0.1410	-5.818	...	0.538044
9795	7290	67	484146	0.23400	0.574	0.512	0.000000	5	0.0946	-6.664	...	0.511990
5299	3661	57	242960	0.02380	0.660	0.523	0.000000	4	0.1420	-12.068	...	0.523013
215	6357	70	239400	0.13500	0.820	0.937	0.000128	5	0.3500	-4.810	...	0.937871
6376	2589	73	230693	0.38200	0.539	0.884	0.001660	9	0.7600	-6.530	...	0.884761

18835 rows × 29 columns

```
In [28]: len(data1)
Out[28]: 18835

In [29]: # На основе масштабированных данных выделим
# обучающую и тестовую выборки
train_data_all = data1[:13000]
test_data_all = data1[13001:]
train_data_all.shape, test_data_all.shape

Out[29]: ((13000, 29), (5834, 29))

In [30]: data.columns
Out[30]: Index(['song_name', 'song_popularity', 'song_duration_ms', 'acousticness',
       'danceability', 'energy', 'instrumentalness', 'key', 'liveness',
       'loudness', 'audio_mode', 'speechiness', 'tempo', 'time_signature',
       'audio_valence', 'song_popularity_scaled', 'song_duration_ms_scaled',
       'acousticness_scaled', 'danceability_scaled', 'energy_scaled',
       'instrumentalness_scaled', 'key_scaled', 'liveness_scaled',
       'loudness_scaled', 'audio_mode_scaled', 'speechiness_scaled',
       'tempo_scaled', 'time_signature_scaled', 'audio_valence_scaled'],
      dtype='object')

In [31]: # Признаки для задачи регрессии (опустим название)
task_regr_cols = ['song_duration_ms', 'acousticness',
                  'danceability', 'energy', 'instrumentalness', 'key', 'liveness',
                  'loudness', 'audio_mode', 'speechiness', 'tempo', 'time_signature',
                  'audio_valence', 'song_duration_ms_scaled',
                  'acousticness_scaled', 'danceability_scaled', 'energy_scaled',
                  'instrumentalness_scaled', 'key_scaled', 'liveness_scaled',
                  'loudness_scaled', 'audio_mode_scaled', 'speechiness_scaled',
                  'tempo_scaled', 'time_signature_scaled', 'audio_valence_scaled']

In [32]: # Выборки для задачи регрессии
regr_X_train = train_data_all[task_regr_cols]
regr_X_test = test_data_all[task_regr_cols]
regr_Y_train = train_data_all['song_popularity']
regr_Y_test = test_data_all['song_popularity']
regr_X_train.shape, regr_X_test.shape, regr_Y_train.shape, regr_Y_test.shape

Out[32]: ((13000, 26), (5834, 26), (13000,), (5834,))
```

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 [33]: # Модели
regr_models = {'Tree':DecisionTreeRegressor(max_depth=10),
               'RF':RandomForestRegressor(max_depth=10, n_estimators=30),
               }

In [34]: # Сохранение метрик
regrMetricLogger = MetricLogger()

In [35]: def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(regr_X_train, regr_Y_train)
    Y_pred = model.predict(regr_X_test)

    mae = mean_absolute_error(regr_Y_test, Y_pred)
    medae = median_absolute_error(regr_Y_test, Y_pred)
    r2 = r2_score(regr_Y_test, Y_pred)

    regrMetricLogger.add('MAE', model_name, mae)
    regrMetricLogger.add('MedAE', model_name, medae)
    regrMetricLogger.add('R2', model_name, r2)

    print('*'*50)
    print(model)
    print()
    print('MAE={}, MedAE={}, R2={}'.format(
        round(mae, 3), round(medae, 3), round(r2, 3)))
    print('*'*50)

In [36]: for model_name, model in regr_models.items():
    regr_train_model(model_name, model, regrMetricLogger)

*****
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=10,
                      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=16.322, MedAE=13.096, R2=0.063
*****
```

```
*****
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                     max_depth=10, 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=30, n_jobs=None, oob_score=False,
                     random_state=None, verbose=0, warm_start=False)

MAE=15.167, MedAE=13.106, R2=0.23
*****
```

Ансамблевая модель

```
In [37]: from heamy.estimator import Regressor
from heamy.pipeline import ModelsPipeline
from heamy.dataset import Dataset
# набор данных
dataset = Dataset(regr_X_train, regr_Y_train, regr_X_test)
# Возьмем лучшую модель: 'TREE+RF=>LR'
# модели первого уровня
model_tree = Regressor(dataset=dataset, estimator=DecisionTreeRegressor, parameters={'max_depth':10},name='tree')
model_lr = Regressor(dataset=dataset, estimator=LinearRegression, name='lr')
model_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor, parameters={'max_depth':10},name='rf')

# Первый уровень - две модели: дерево и случайный лес
# Второй уровень: линейная регрессия
pipeline = ModelsPipeline(model_tree, model_rf)
stack_ds = pipeline.stack(k=10, seed=1)
# модель второго уровня
stacker = Regressor(dataset=stack_ds, estimator=LinearRegression)
results = stacker.validate(k=10,scorer=mean_absolute_error)
print()

results = stacker.validate(k=10,scorer=median_absolute_error)
```

Metric: mean_absolute_error
Folds accuracy: [14.96437162340574, 15.261276472362008, 15.135652469642725, 14.533209557049261, 15.46177532011503, 14.986949918590824, 15.7205488743907, 15.0906352439273, 15.363554494975867, 15.112883031676219]
Mean accuracy: 15.163085700613568
Standard Deviation: 0.3042345899647139
Variance: 0.0925586857309976

Metric: median_absolute_error
Folds accuracy: [12.463363958441889, 12.722672504491005, 13.10741165786525, 12.384636592544865, 12.93080468687274, 12.635947082982504, 13.29587213688609, 12.835735765047367, 12.859997308686943, 12.665372444902232]
Mean accuracy: 12.790181413872089
Standard Deviation: 0.2642502052451234
Variance: 0.06982817097208982

9. Подбор гиперпараметров для выбранных моделей.

Случайный лес

```
In [38]: RandomForestRegressor().get_params()
```

```
Out[38]: {'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}
```

```
In [39]: n_range = np.array(range(0,50,5))
tuned_parameters = [{'max_depth': n_range}]
tuned_parameters
```

```
Out[39]: [{‘max_depth’: array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45])}]
```

```
In [40]: %%time
rf_gs = GridSearchCV(RandomForestRegressor(), tuned_parameters, cv=5, scoring=‘neg_mean_squared_error’)
rf_gs.fit(regr_X_train, regr_Y_train)
```

```
CPU times: user 10min 12s, sys: 586 ms, total: 10min 12s
Wall time: 10min 13s
```

```
Out[40]: GridSearchCV(cv=5, error_score=nan,
                     estimator=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),
                     iid='deprecated', n_jobs=None,
                     param_grid=[{'max_depth': array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45])}],
                     pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                     scoring='neg_mean_squared_error', verbose=0)
```

```
In [41]: # Лучшая модель
rf_gs.best_estimator_
```

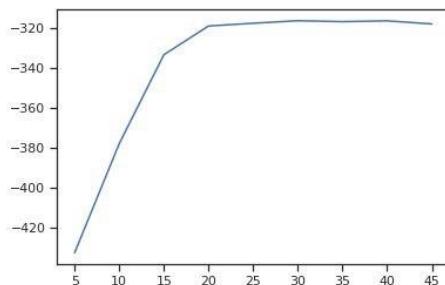
```
Out[41]: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                               max_depth=30, max_features='auto', max_leaf_nodes=None,
                               max_samples=None, min_impurity_decrease=0.0,
                               min_impurity_split=None, 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)
```

```
In [42]: # Лучшее значение параметров
rf_gs.best_params_
```

```
Out[42]: {'max_depth': 30}
```

```
In [43]: # Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, rf_gs.cv_results_['mean_test_score'])
```

```
Out[43]: [
```



Дерево

```
In [44]: DecisionTreeRegressor().get_params()
```

```
Out[44]: {'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'}
```

```
In [45]: n_range = np.array(range(0,50,5))
```

```
tuned_parameters = [{'max_depth': n_range}]
tuned_parameters
```

```
Out[45]: [{'max_depth': array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45])}]
```

```
In [46]: %%time
dt_gs = GridSearchCV(DecisionTreeRegressor(), tuned_parameters, cv=5, scoring='neg_mean_squared_error')
dt_gs.fit(regr_X_train, regr_Y_train)
```

```
CPU times: user 11.1 s, sys: 8 µs, total: 11.1 s
Wall time: 11.1 s
```

```
Out[46]: GridSearchCV(cv=5, error_score=nan,
                     estimator=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'),
                     iid='deprecated', n_jobs=None,
                     param_grid=[{'max_depth': array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45])}],
                     pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                     scoring='neg_mean_squared_error', verbose=0)
```

```
In [47]: # Лучшая модель
dt_gs.best_estimator_
```

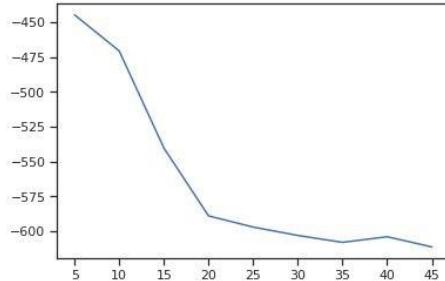
```
Out[47]: DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=5,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort='deprecated',
                                random_state=None, splitter='best')
```

```
In [48]: # Лучшее значение параметров
dt_gs.best_params_
```

```
Out[48]: {'max_depth': 5}
```

```
In [49]: # Изменение качества на тестовой выборке
plt.plot(n_range, dt_gs.cv_results_['mean_test_score'])
```

```
Out[49]: [<matplotlib.lines.Line2D at 0x7fe15c62cdf0>]
```



Ансамблевая модель

Поскольку параметры для случайного леса и дерева уже подобрали, то воспользуемся ими, а так же попробуем подобрать еще 2 параметра для данных моделей.

Decision tree

```
In [50]: n_range = [0, 0.5, 1, 1.5, 2, 2.5, 3]
tuned_parameters = [{'min_impurity_split': n_range}]
tuned_parameters
```

```
Out[50]: [{'min_impurity_split': [0, 0.5, 1, 1.5, 2, 2.5, 3]}]
```

```
In [51]: %%time
ens_dt_gs = GridSearchCV(DecisionTreeRegressor(), tuned_parameters, cv=5, scoring='neg_mean_squared_error')
ens_dt_gs.fit(regr_X_train, regr_Y_train)
```

```
CPU times: user 9.54 s, sys: 6.67 ms, total: 9.55 s
Wall time: 9.54 s

Out[51]: GridSearchCV(cv=5, error_score='nan',
                     estimator=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'),
                     iid='deprecated', n_jobs=None,
                     param_grid=[{'min_impurity_split': [0, 0.5, 1, 1.5, 2, 2.5, 3]}],
                     pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                     scoring='neg_mean_squared_error', verbose=0)
```

```
In [52]: # Лучшая модель
ens_dt_gs.best_estimator_
```

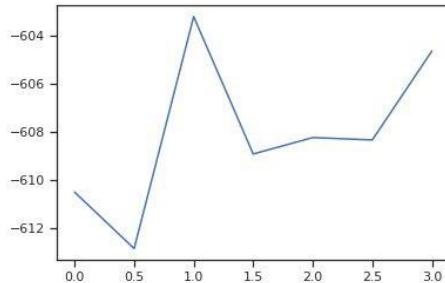
```
Out[52]: DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                               max_features=None, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=1,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, presort='deprecated',
                               random_state=None, splitter='best')
```

```
In [53]: # Лучшее значение параметров
ens_dt_gs.best_params_
```

```
Out[53]: {'min_impurity_split': 1}
```

```
In [54]: # Изменение качества на тестовой выборке
plt.plot(n_range, ens_dt_gs.cv_results_['mean_test_score'])
```

```
Out[54]: [<matplotlib.lines.Line2D at 0x7fe15c7034f0>]
```



Random Forest

```
In [55]: n_range = [1, 5, 10, 20, 30, 40, 50, 60]
tuned_parameters = [{'n_estimators': n_range}]
tuned_parameters
```

```
Out[55]: [{"n_estimators": [1, 5, 10, 20, 30, 40, 50, 60]}]
```

```
In [56]: %%time
ens_rf_gs = GridSearchCV(RandomForestRegressor(), tuned_parameters, cv=5, scoring='neg_mean_squared_error')
ens_rf_gs.fit(regr_X_train, regr_Y_train)
```

CPU times: user 2min 55s, sys: 343 ms, total: 2min 55s
Wall time: 2min 55s

```
Out[56]: GridSearchCV(cv=5, error_score=nan,
estimator=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),
iid='deprecated', n_jobs=None,
param_grid=[{'n_estimators': [1, 5, 10, 20, 30, 40, 50, 60]}],
pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
scoring='neg_mean_squared_error', verbose=0)
```

```
In [57]: # Лучшая модель
ens_rf_gs.best_estimator_
```

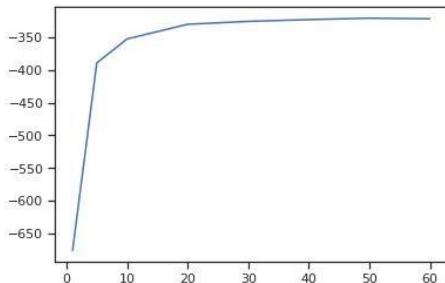
```
Out[57]: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
max_depth=None, max_features='auto', max_leaf_nodes=None,
max_samples=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=50, n_jobs=None, oob_score=False,
random_state=None, verbose=0, warm_start=False)
```

```
In [58]: # Лучшее значение параметров
ens_rf_gs.best_params_
```

```
Out[58]: {'n_estimators': 50}
```

```
In [59]: # Изменение качества на тестовой выборке
plt.plot(n_range, ens_rf_gs.cv_results_['mean_test_score'])
```

```
Out[59]: [<matplotlib.lines.Line2D at 0x7fe15c2eb1f0>]
```



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

```
In [60]: regr_models_grid = {'Tree':dt_gs.best_estimator_,
                           'RF': rf_gs.best_estimator_
                           }

In [61]: for model_name, model in regr_models_grid.items():
          regr_train_model(model_name, model, regrMetricLogger)

*****DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=5,
                           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=16.608, MedAE=13.954, R2=0.083
*****
*****RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                           max_depth=30, 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=12.323, MedAE=8.754, R2=0.387
*****
```

Удалось немного улучшить модель дерева решений и достаточно неплохо улучшить модель случайный лес

Ансамблевый метод

```
In [62]: # # # Возьмем лучшую модель: 'TREE+RF=>LR'
# # # модели первого уровня
model_tree = Regressor(dataset=dataset,
                        estimator=DecisionTreeRegressor,
                        parameters={'min_impurity_split':1.5,
                                    'max_depth':5},name='tree')
model_lr = Regressor(dataset=dataset,
                      estimator=LinearRegression,
                      name='lr')
model_rf = Regressor(dataset=dataset,
                      estimator=RandomForestRegressor,
                      parameters={'n_estimators': 60,
                                    'max_depth': 40},name='rf')

# Первый уровень - две модели: дерево и случайный лес
# Второй уровень: линейная регрессия
pipeline = ModelsPipeline(model_tree, model_rf)
stack_ds = pipeline.stack(k=10, seed=1)
# модель второго уровня
stacker = Regressor(dataset=stack_ds, estimator=LinearRegression)
```

```
In [63]: results = stacker.validate(k=10,scorer=mean_absolute_error)
print()

results = stacker.validate(k=10,scorer=median_absolute_error)

Metric: mean_absolute_error
Folds accuracy: [13.19989096900419, 13.000756582764984, 12.711228562935805, 12.478494444021358, 13.2324763373497,
12.618548762104865, 13.50245225264392, 12.95749150267499, 13.258452378334951, 12.972151058171917]
Mean accuracy: 12.993204097790292
Standard Deviation: 0.30270118673169716
Variance: 0.0916280084487778

Metric: median_absolute_error
Folds accuracy: [9.042351857825604, 9.726315742476164, 9.268826187749738, 9.220001930270072, 9.579315229641502, 9.
11200034777163, 9.714339248482386, 9.679652687298194, 9.45495211334001, 9.65332300197894]
Mean accuracy: 9.445107834683423
Standard Deviation: 0.24959120217344136
Variance: 0.062295768202383674
```

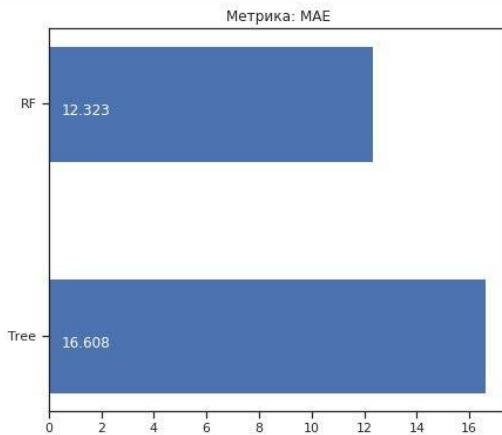
Удалось неплохо улучшить модель.

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

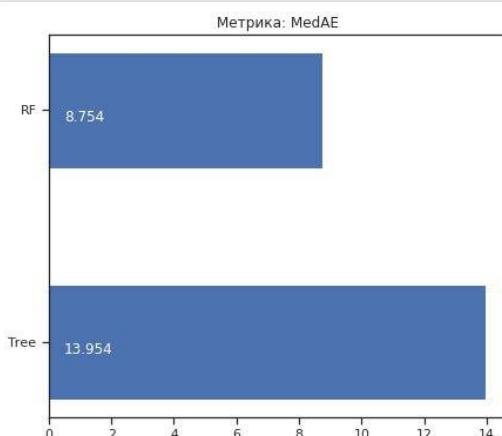
```
In [64]: # Метрики качества модели  
regr_metrics = regrMetricLogger.df['metric'].unique()  
regr_metrics
```

```
Out[64]: array(['MAE', 'MedAE', 'R2'], dtype=object)
```

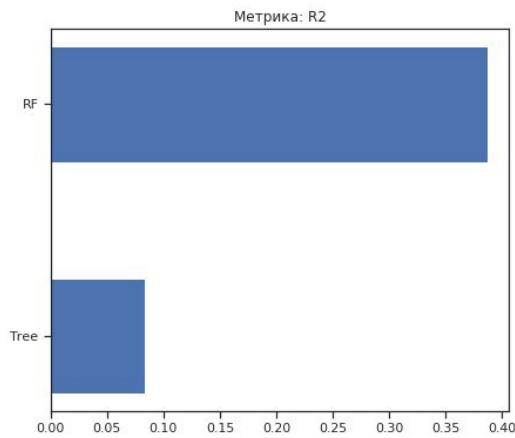
```
In [65]: regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE',  
                             ascending=False, figsize=(7, 6))
```



```
In [66]: regrMetricLogger.plot('Метрика: ' + 'MedAE', 'MedAE',  
                             ascending=False, figsize=(7, 6))
```



```
In [67]: regrMetricLogger.plot('Метрика: ' + 'R2', 'R2',
                             ascending=True, figsize=(7, 6))
```



Вывод:

Лучше всего показала себя модель случайный лес, на втором месте - ансамблевая модель, на третьем - дерево решений. Однако в другой задаче в лабораторной работе лучше показала себя ансамблевая модель, так что в дальнейшем можно использовать обе эти модели и проверять, какая будет работать лучше для конкретной задачи.