НИРС

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.

В качестве набора данных будет использоваться набор данных со статистикой всех популярных песен в стриминговом сервисе `Spotify`:
https://www.kaggle.com/datasets/asaniczka/top-spotify-songs-in-73-countries-daily-updated
Датасет состоит из одного файла:

- universal_top_spotify_songs.csv В файле есть следующие колонки:
- spotify_id - ID песни в `Spotify` (в анализе использоваться не будет)
- name - название песни
- artists - имена музыкантов-авторов песни, разделены через запятую
- snapshot_date - дата загрузки информации о песне в датасет (служебное поле, использоваться в анализе не будет)
- daily_rank - служебное поле которое не используется в работе
- daily_movement - то же
- weekly_movement - то же
- country - страна, в которой песня обрела наибольшую популярность
- popularity - популярность песни в `Spotify` (метрика от 0 до 100)
- is_explicit - есть ли в песне нецензурное содержание
- duration_ms - длительность песни в миллисекундах
- album_name - название альбома, на котором была выпущена песня
- album_release_date - дата релиза альбома с песней
- danceability - метрика танцевальности песни от 0 до 1
- energy - метрика энергичности песни от 0 до 1
- key - тональность песни (закодированный категориальный признак)
- loudness - средняя громкость песни в децибелах
- mode - мажрная или минорная тональность у песни
- speechiness - метрика количества произнесённых в песне слов от 0 до 1
- acousticness - метрика акустического качества песни от 0 до 1
- instrumentalness - метрика количества инструментала в песне от 0 до 1
- liveness - метрика присутствия аудитории в момент записи песни от 0 до 1
- valence - метрика позитивности песни от 0 до 1
- tempo - темп песни в ударах в минуту

- time_signature - тактовый размер песни (3/4 или 4/4) В данной работе будет решаться задача **регрессии**. В качестве целевого признака будем использовать поле `popularity`.

Импорт библиотек

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
from sklearn.model_selection import GridSearchCV
```

Загрузка данных

```
df = pd.read_csv('universal_top_spotify_songs.csv')
```

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

```
df.head()
```

```
                spotify_id                                      name  \
0  7fzHQizxTqy8wTXwlrgPQQ                    MILLION DOLLAR BABY
1  6AI3ezQ4o3HUoP6Dhudph3                            Not Like Us
2  2qSkIjg1o9h3YT9RAgYN75                               Espresso
3  7221xIgOnuakPdLqT0F3nP  I Had Some Help (Feat. Morgan Wallen)
4  2GxrNKugF82CnoRFbQfzPf              i like the way you kiss me

                      artists  daily_rank  daily_movement
weekly_movement  \
0               Tommy Richman           1               0
1
1             Kendrick Lamar            2               0
-1
2           Sabrina Carpenter           3               0
0
3  Post Malone, Morgan Wallen           4               0
46
```

```
4                     Artemas           5                0
-1

  country snapshot_date  popularity  is_explicit  ...  key loudness
mode  \
0     NaN    2024-05-17          96        False  ...    1   -5.106
0
1     NaN    2024-05-17          96         True  ...    1   -7.001
1
2     NaN    2024-05-17          99         True  ...    0   -5.478
1
3     NaN    2024-05-17          92         True  ...    7   -4.860
1
4     NaN    2024-05-17         100        False  ...   11   -4.263
1

   speechiness  acousticness  instrumentalness  liveness  valence
tempo  \
0       0.0436      0.098200          0.000215    0.0680    0.927
138.003
1       0.0776      0.010700          0.000000    0.1410    0.214
101.061
2       0.0285      0.107000          0.000065    0.1850    0.690
103.969
3       0.0264      0.007570          0.000000    0.2450    0.731
127.986
4       0.0447      0.000938          0.010600    0.0826    0.747
151.647

   time_signature
0               4
1               4
2               4
3               4
4               4

[5 rows x 25 columns]

df.shape

(763403, 25)

df.columns

Index(['spotify_id', 'name', 'artists', 'daily_rank',
'daily_movement',
       'weekly_movement', 'country', 'snapshot_date', 'popularity',
       'is_explicit', 'duration_ms', 'album_name',
'album_release_date',
       'danceability', 'energy', 'key', 'loudness', 'mode',
'speechiness',
```

```
       'acousticness', 'instrumentalness', 'liveness', 'valence',
'tempo',
       'time_signature'],
      dtype='object')
```

```
df.dtypes
```

```
spotify_id             object
name                   object
artists                object
daily_rank              int64
daily_movement          int64
weekly_movement         int64
country                object
snapshot_date          object
popularity              int64
is_explicit              bool
duration_ms             int64
album_name             object
album_release_date     object
danceability          float64
energy                float64
key                     int64
loudness              float64
mode                    int64
speechiness           float64
acousticness          float64
instrumentalness      float64
liveness              float64
valence               float64
tempo                 float64
time_signature          int64
dtype: object
```

Удаление дупликатов

```
df = df.drop_duplicates()
```

Проверка наличия нулевых значений

```
df.isnull().sum()
```

```
spotify_id                0
name                     27
artists                  27
daily_rank                0
daily_movement            0
weekly_movement           0
country               10307
```

```
snapshot_date              0
popularity                 0
is_explicit                0
duration_ms                0
album_name               261
album_release_date       261
danceability               0
energy                     0
key                        0
loudness                   0
mode                       0
speechiness                0
acousticness               0
instrumentalness           0
liveness                   0
valence                    0
tempo                      0
time_signature             0
dtype: int64
```

Согласно описанию датасета, поле `country` равняется `Null`, если песня попала в плейлист `Global Top 50`. Поэтому, для использования датасета, заменим пустые значения в колонке `country` на строку `"Global"`

```
df['country'] = df['country'].fillna("Global")
df.isnull().sum()
```

```
spotify_id               0
name                    27
artists                 27
daily_rank               0
daily_movement           0
weekly_movement          0
country                  0
snapshot_date            0
popularity               0
is_explicit              0
duration_ms              0
album_name             261
album_release_date     261
danceability             0
energy                   0
key                      0
loudness                 0
mode                     0
speechiness              0
acousticness             0
instrumentalness         0
liveness                 0
```

```
valence                 0
tempo                   0
time_signature          0
dtype: int64
```

Остальные строки с пропусками удалим.
Это мотивированно тем, что пропуски имеются в не числовых колонках, которые заменять на самое часто встречающееся значение было бы некорректно.

```
df = df.dropna()
df.isnull().sum()
```

```
spotify_id              0
name                    0
artists                 0
daily_rank              0
daily_movement          0
weekly_movement         0
country                 0
snapshot_date           0
popularity              0
is_explicit             0
duration_ms             0
album_name              0
album_release_date      0
danceability            0
energy                  0
key                     0
loudness                0
mode                    0
speechiness             0
acousticness            0
instrumentalness        0
liveness                0
valence                 0
tempo                   0
time_signature          0
dtype: int64
```

Так как датасет очень большой (более 700000 строк), сократим его до 5000 строк:

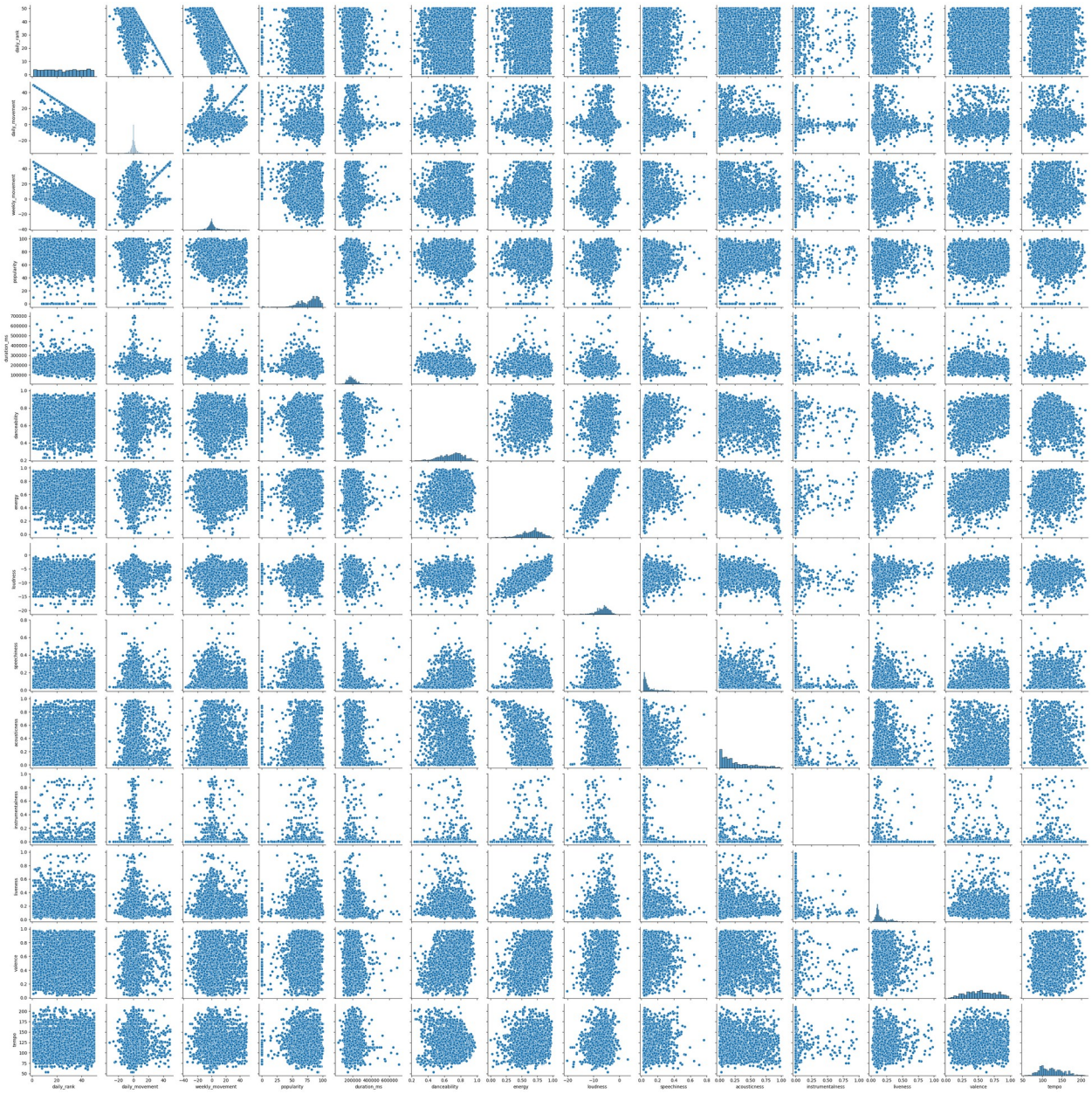
```
df = df.sample(n=5000, random_state=73)
```

Итог: удалили лишние колонки из датасета, очистили его от дупликатов и пустых значений.

Построение графиков для понимания структуры данных

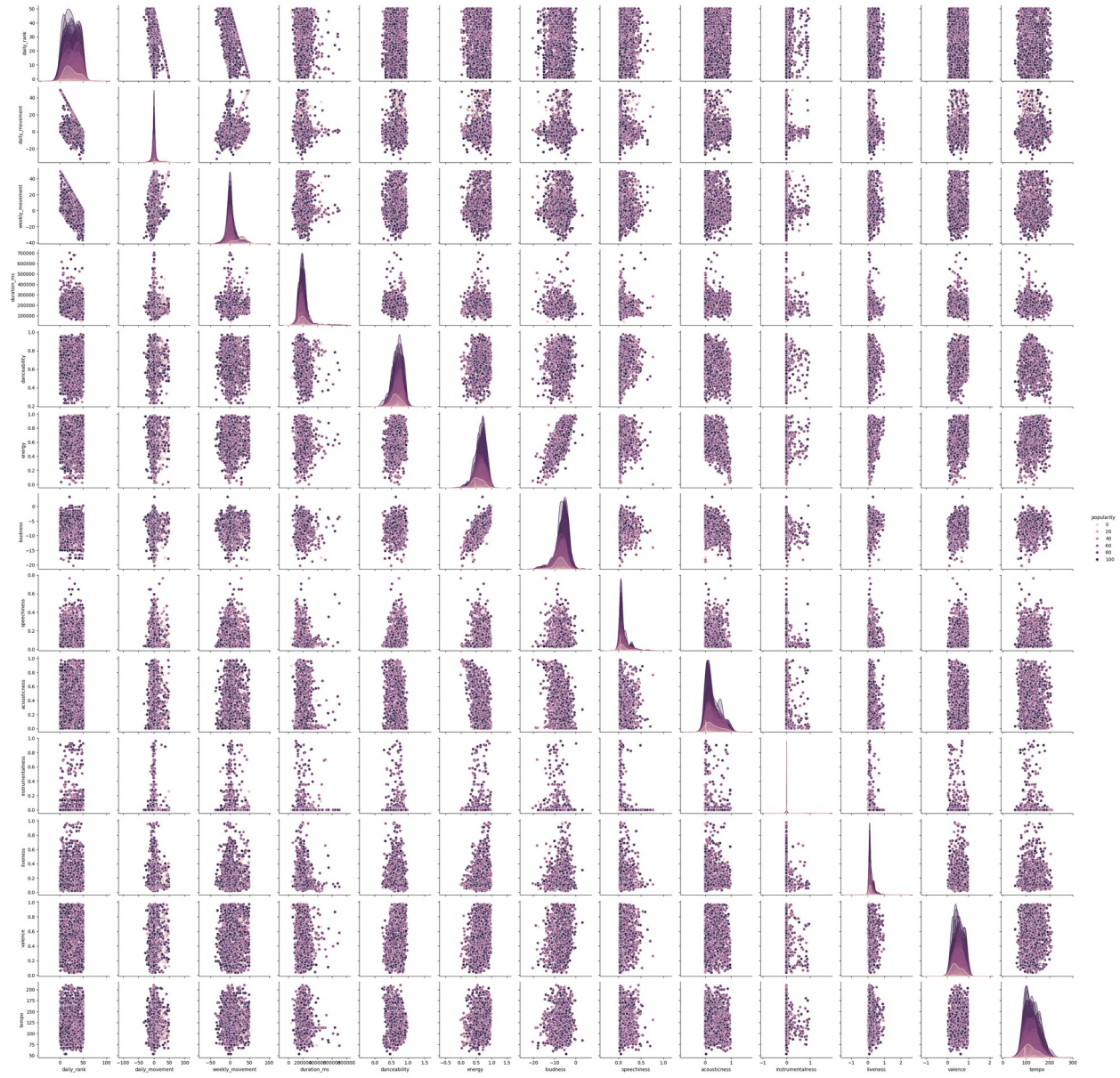
```
cat_cols = ['is_explicit', 'key', 'mode', 'time_signature'] # категориальные колонки, закодированные в самом датасете

sns.pairplot(df.drop(columns=cat_cols))
```

```
<seaborn.axisgrid.PairGrid at 0x7a3c76cfb500>
```



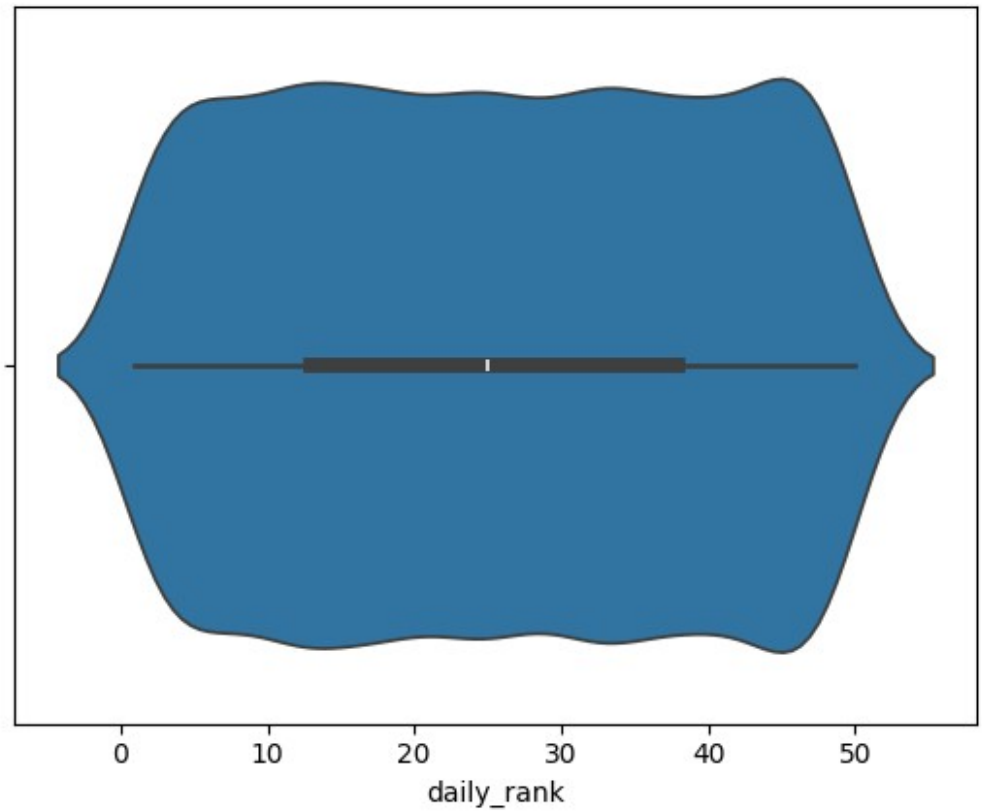

```
sns.pairplot(df.drop(columns=cat_cols), hue='popularity')
```

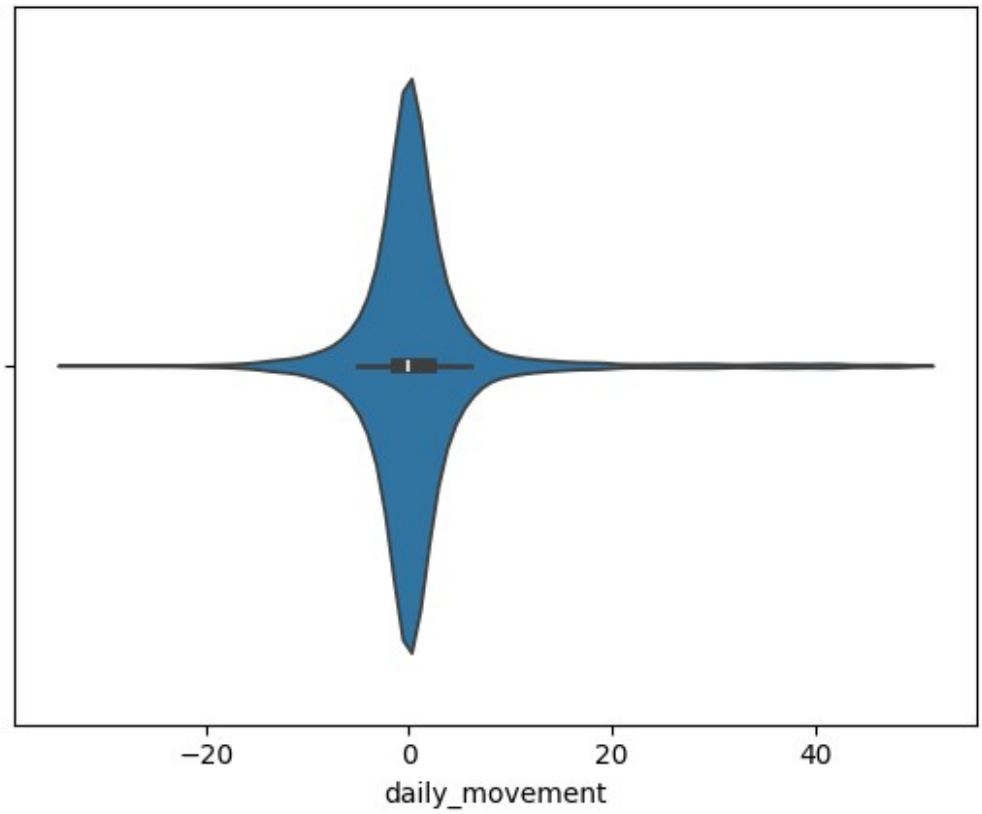
```
<seaborn.axisgrid.PairGrid at 0x7a3c2dd4e510>
```

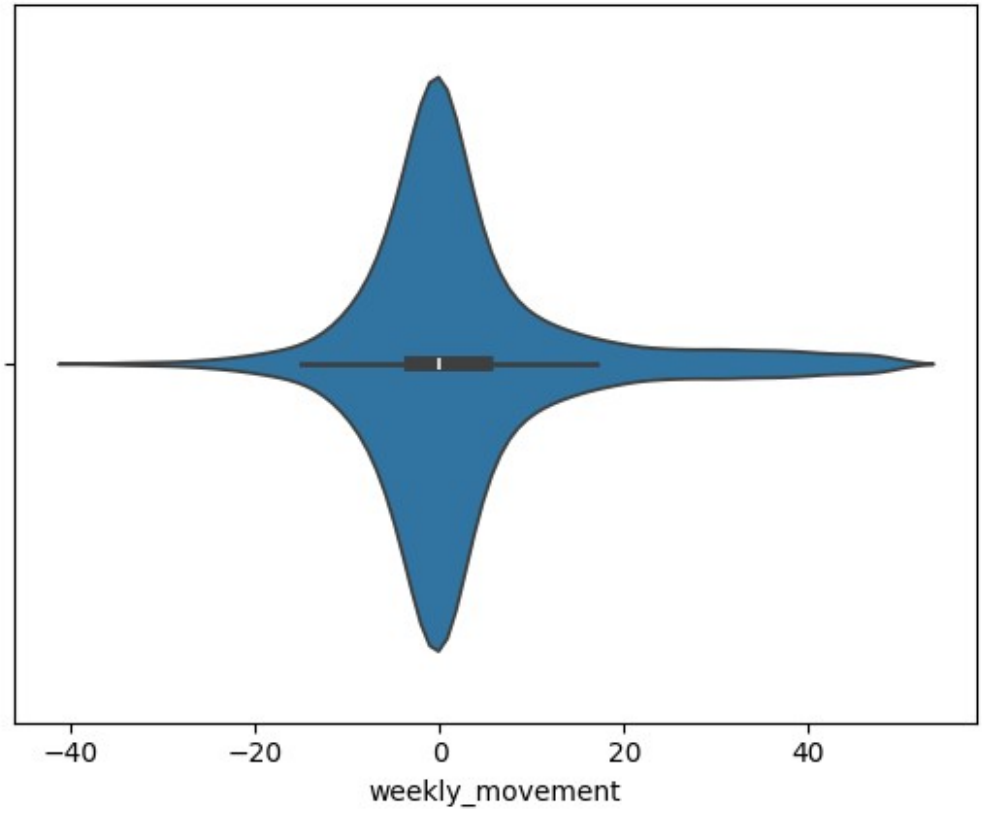


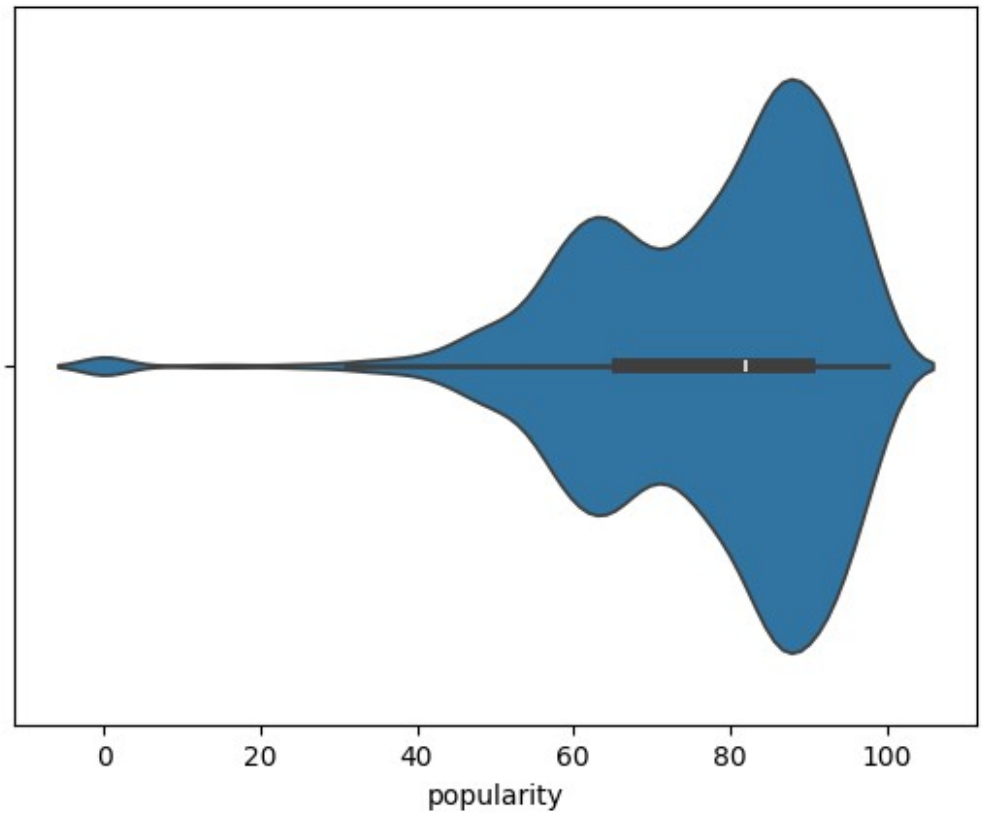

```
df.columns

Index(['spotify_id', 'name', 'artists', 'daily_rank',
'daily_movement',
       'weekly_movement', 'country', 'snapshot_date', 'popularity',
       'is_explicit', 'duration_ms', 'album_name',
'album_release_date',
       'danceability', 'energy', 'key', 'loudness', 'mode',
'speechiness',
       'acousticness', 'instrumentalness', 'liveness', 'valence',
'tempo',
```

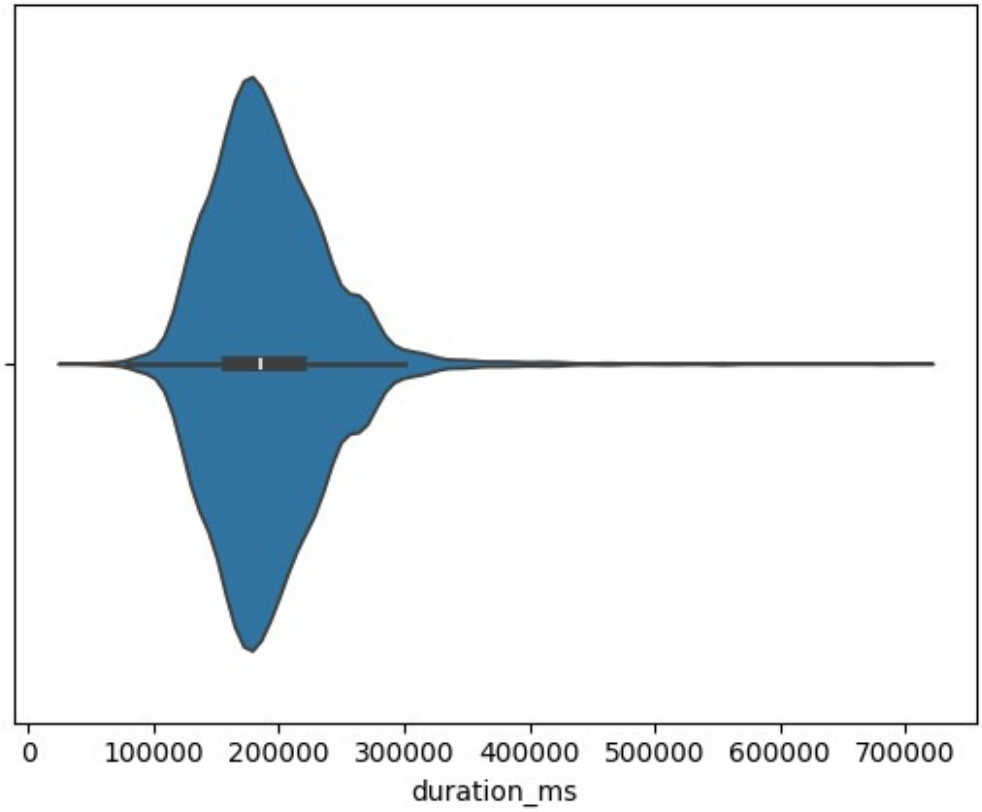
```
       'time_signature'],
      dtype='object')

for col in df.columns:
    if str(df[col].dtype) not in ['int64', 'float64']:
        continue
    sns.violinplot(x=df[col])
    plt.show()
```

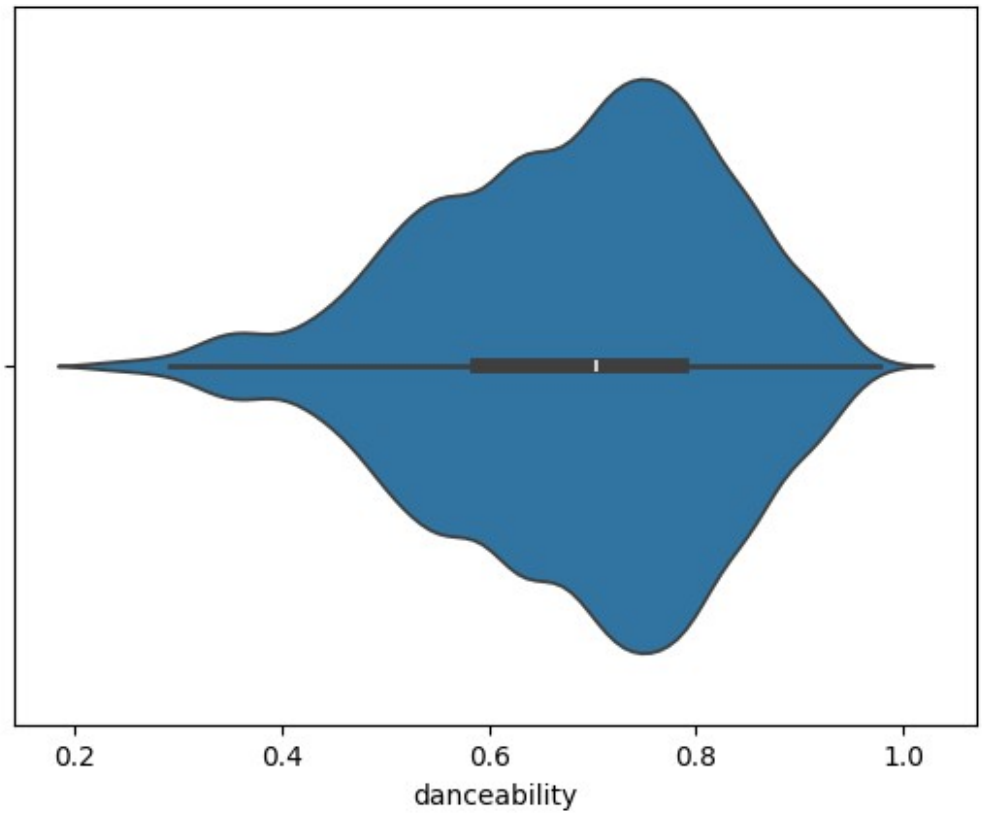


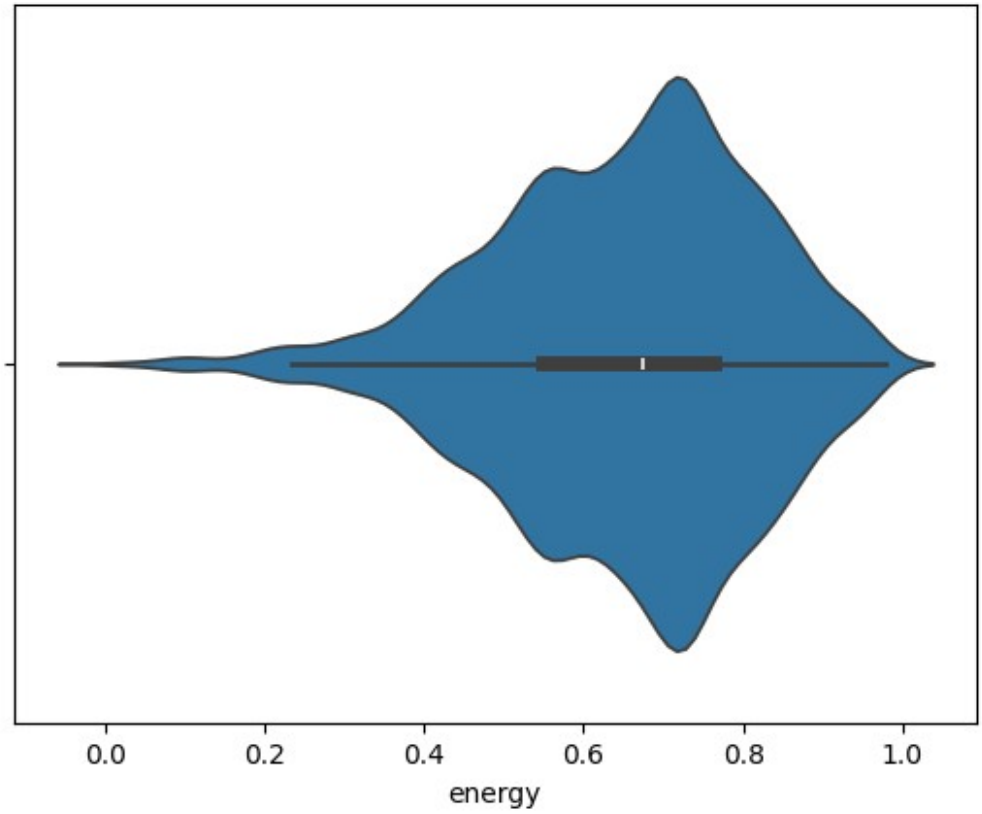

−20
0
20
40
daily_movement

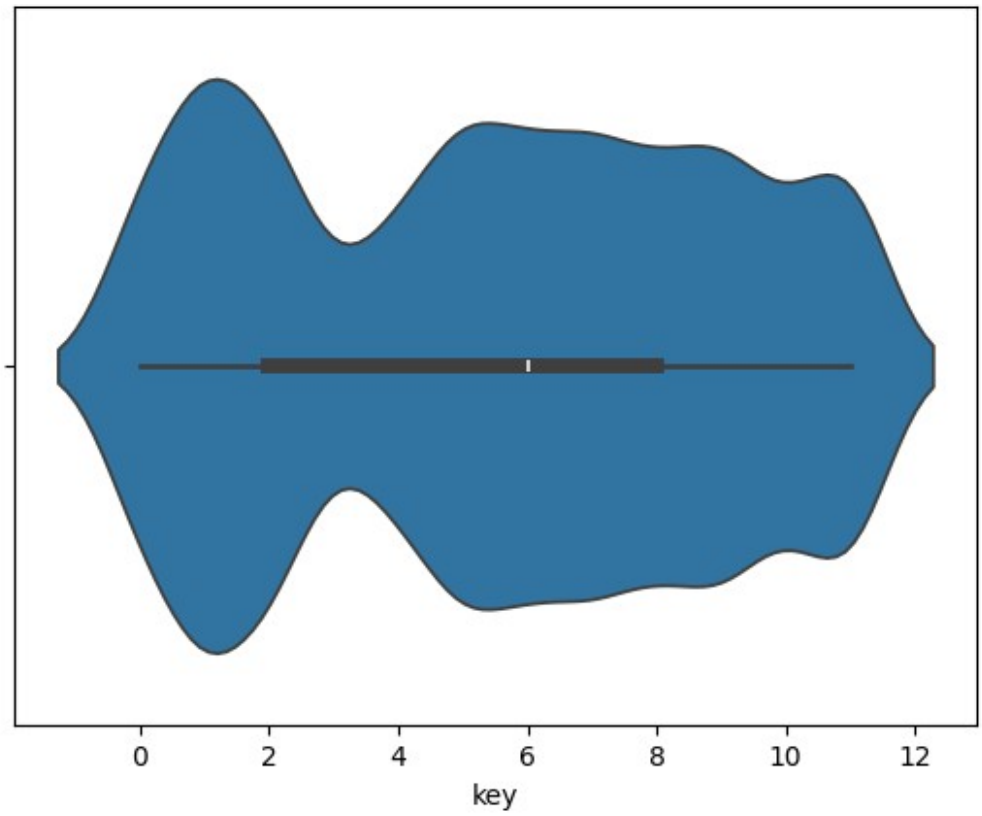

−40
−20
0
20
40
weekly_movement

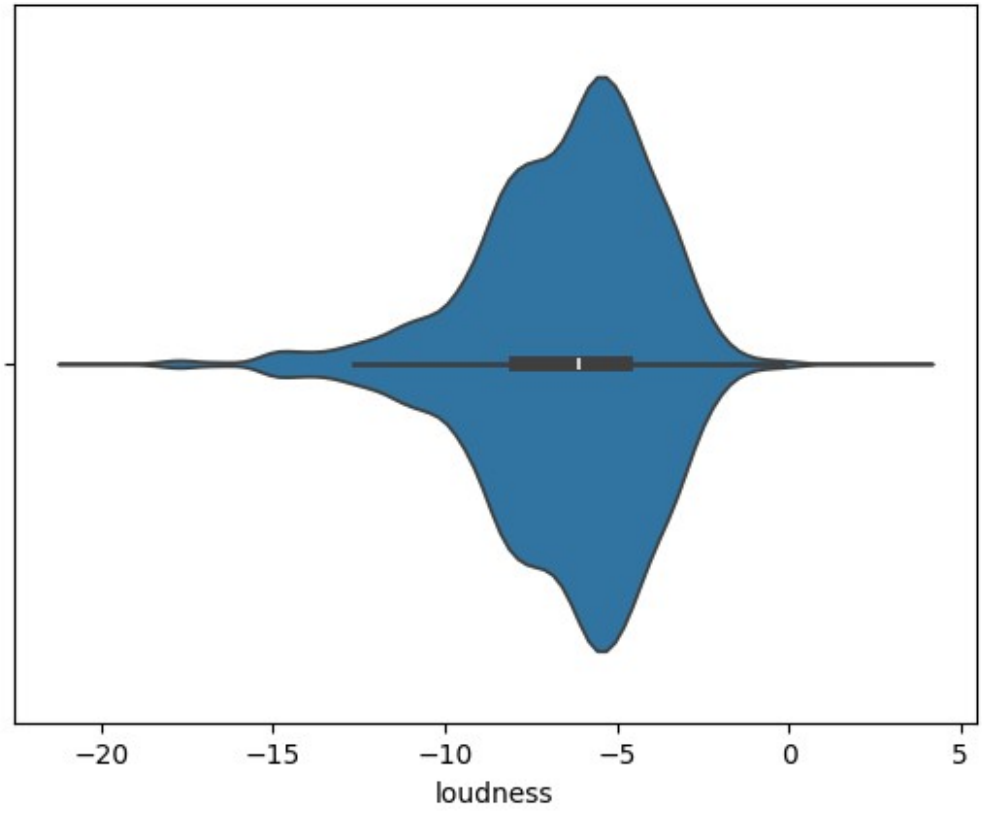

0
20
40
60
80
100
popularity

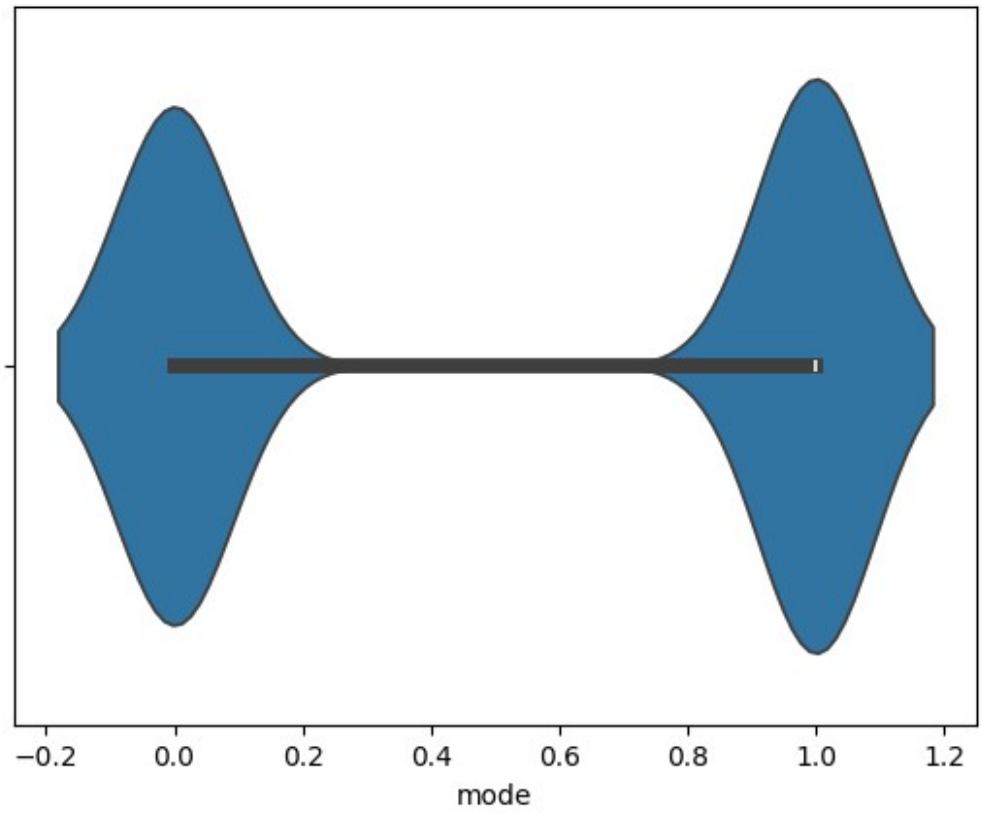

0
100000
200000
300000
400000
500000
600000
700000
duration_ms

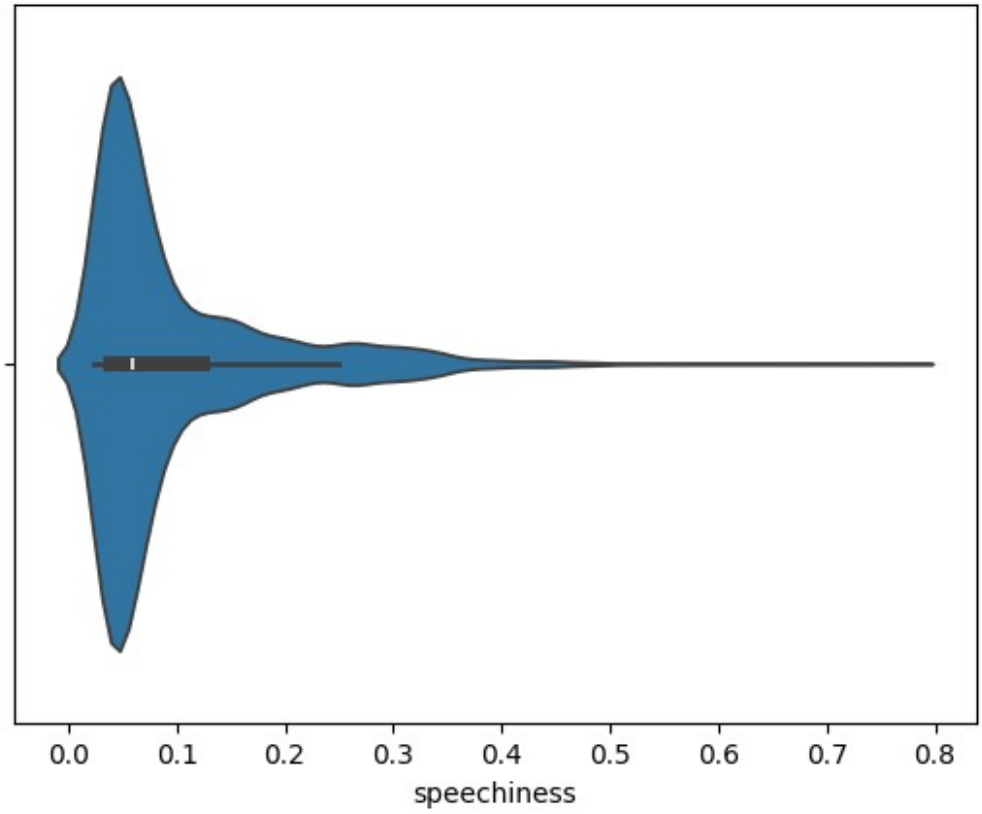

0.2
0.4
0.6
0.8
1.0
danceability

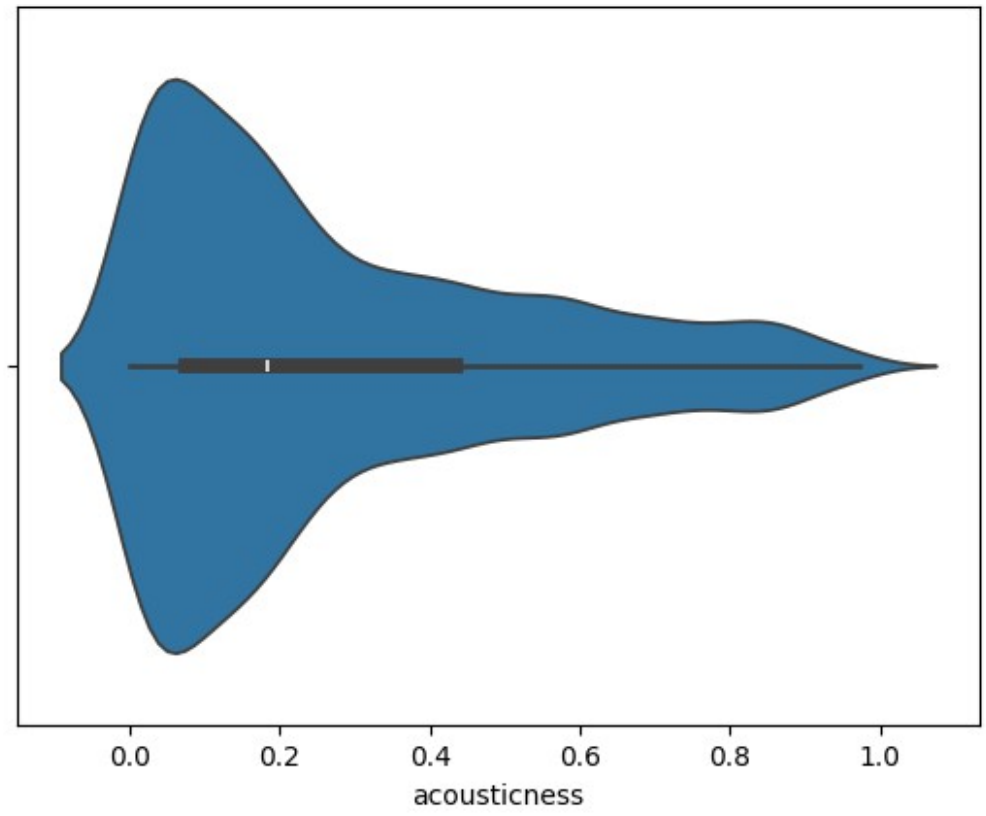

0.0
0.2
0.4
0.6
0.8
1.0
energy

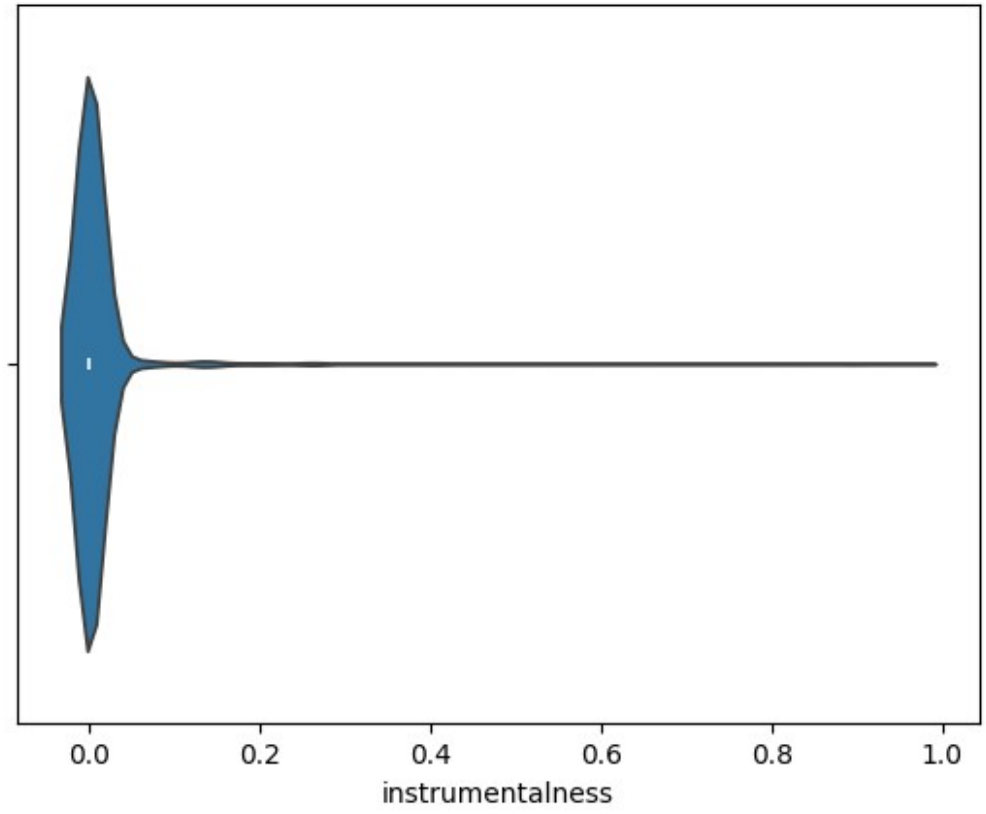

0
2
4
6
8
10
12
key

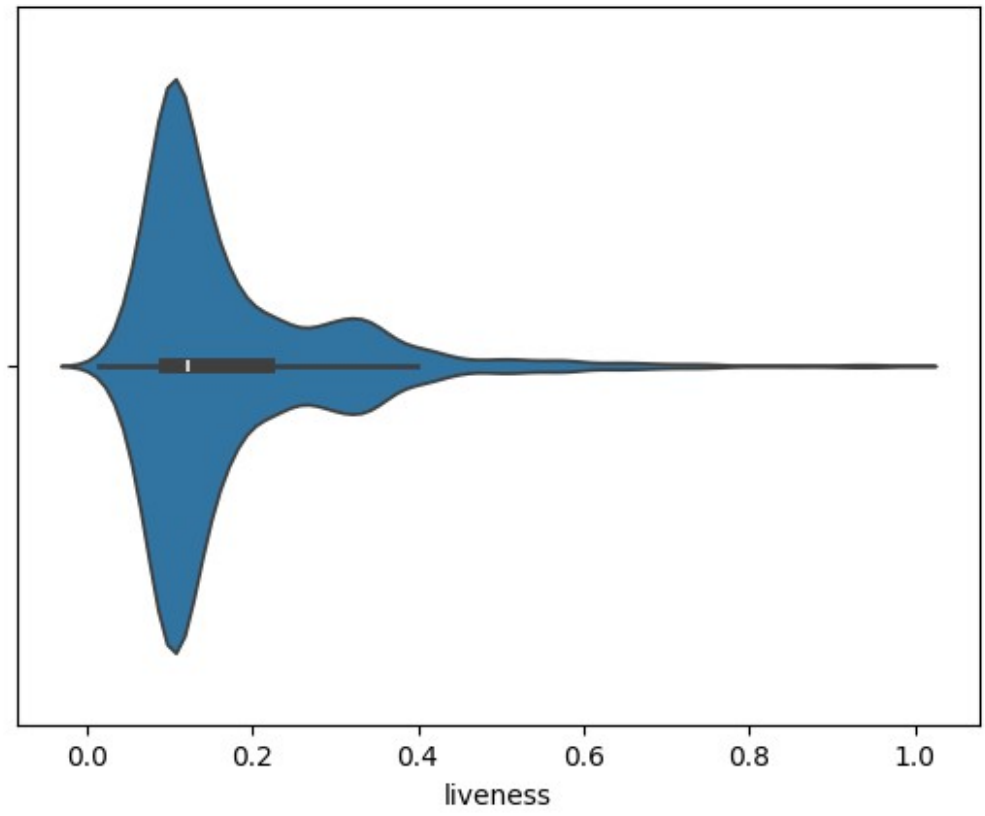

−20
−15
−10
−5
0
5
loudness

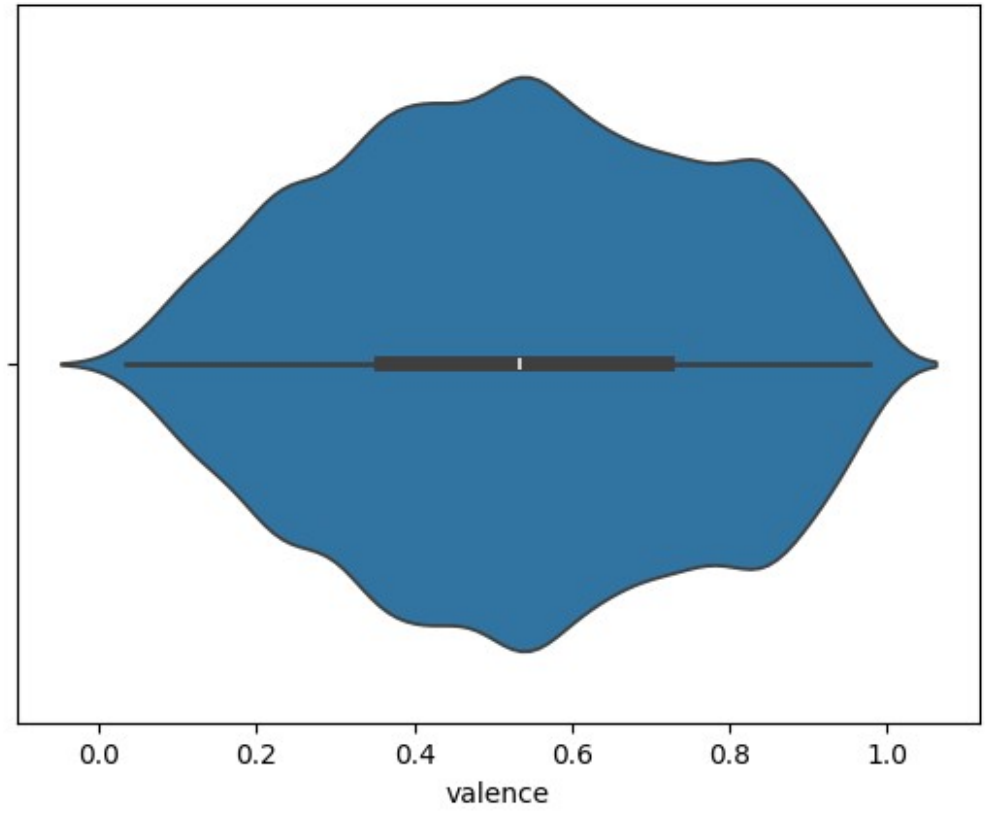

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
mode

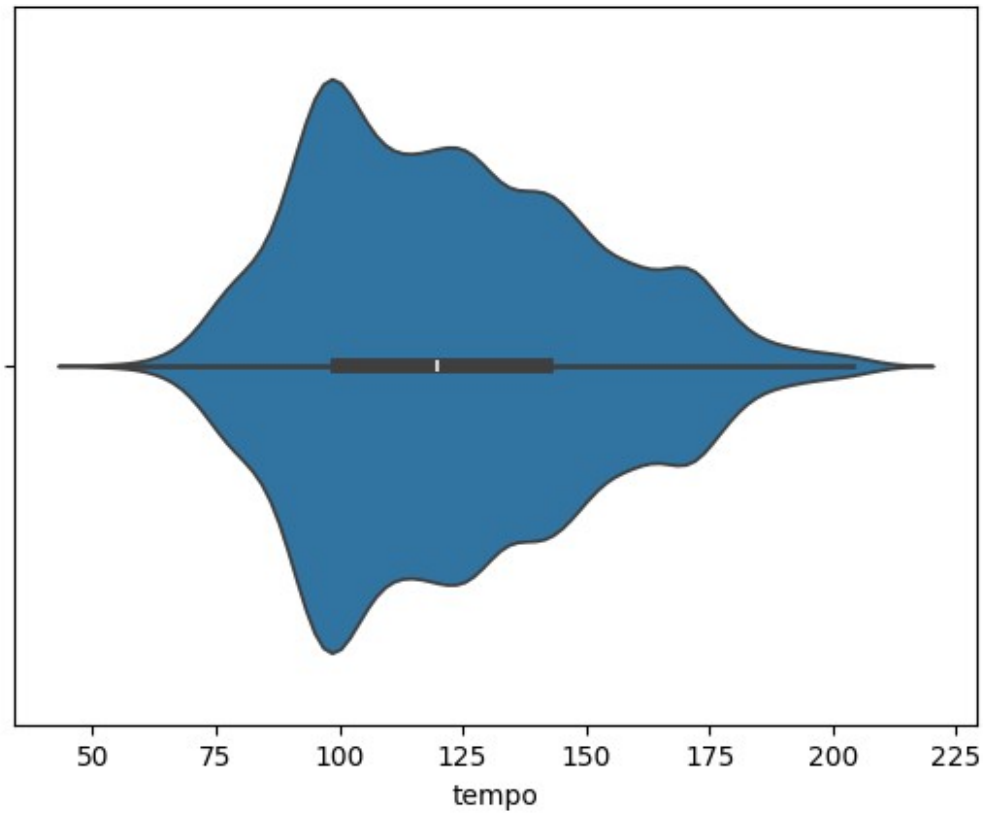

0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
speechiness

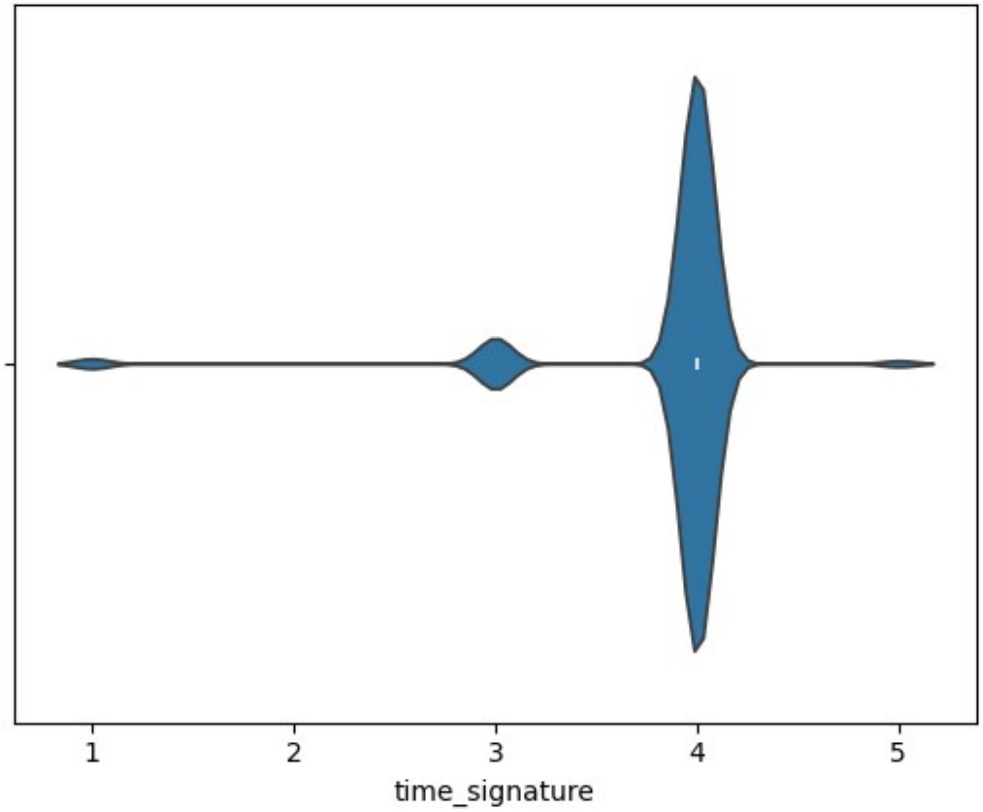

0.0
0.2
0.4
0.6
0.8
1.0
acousticness


0.0
0.2
0.4
0.6
0.8
1.0
instrumentalness


0.0
0.2
0.4
0.6
0.8
1.0
liveness


0.0
0.2
0.4
0.6
0.8
1.0
valence


50
75
100
125
150
175
200
225
tempo


1
2
3
4
5
time_signature

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

```
df.dtypes
```

```
spotify_id              object
name                    object
artists                 object
daily_rank               int64
daily_movement           int64
weekly_movement          int64
country                 object
snapshot_date           object
popularity               int64
is_explicit               bool
duration_ms              int64
album_name              object
album_release_date      object
danceability           float64
energy                 float64
key                      int64
loudness               float64
mode                     int64
speechiness            float64
acousticness           float64
instrumentalness       float64
liveness               float64
valence                float64
tempo                  float64
time_signature           int64
dtype: object
```

Удалим из датасета неиспользуемые служебные колонки (`daily_rank`, `daily_movement`, ...).

```
df = df.drop(columns=['daily_rank', 'daily_movement',
'weekly_movement', 'snapshot_date', 'spotify_id'])
```

Так же в анализе не будут использоваться такие поля, как `album_name`, `artists` и `name` - так как эти признаки служат для идентификации отдельных песен, но не представляют ценности именно для анализа данных. Поле `album_release_date` тоже не будет использоваться, так как мы не рассматриваем датасет как временной ряд.

```
df = df.drop(columns=['album_name', 'artists', 'name',
'album_release_date'])
```

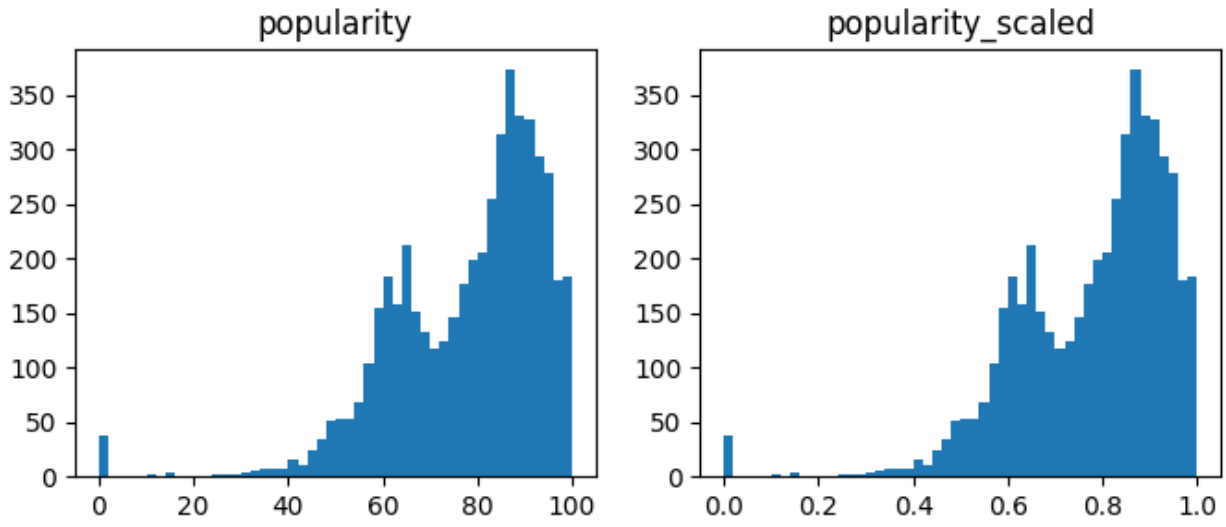
В датасете есть несколько категориальных признаков, но в большинстве своём они уже закодированны.
Исключением является только колонка `country`, исправим это.

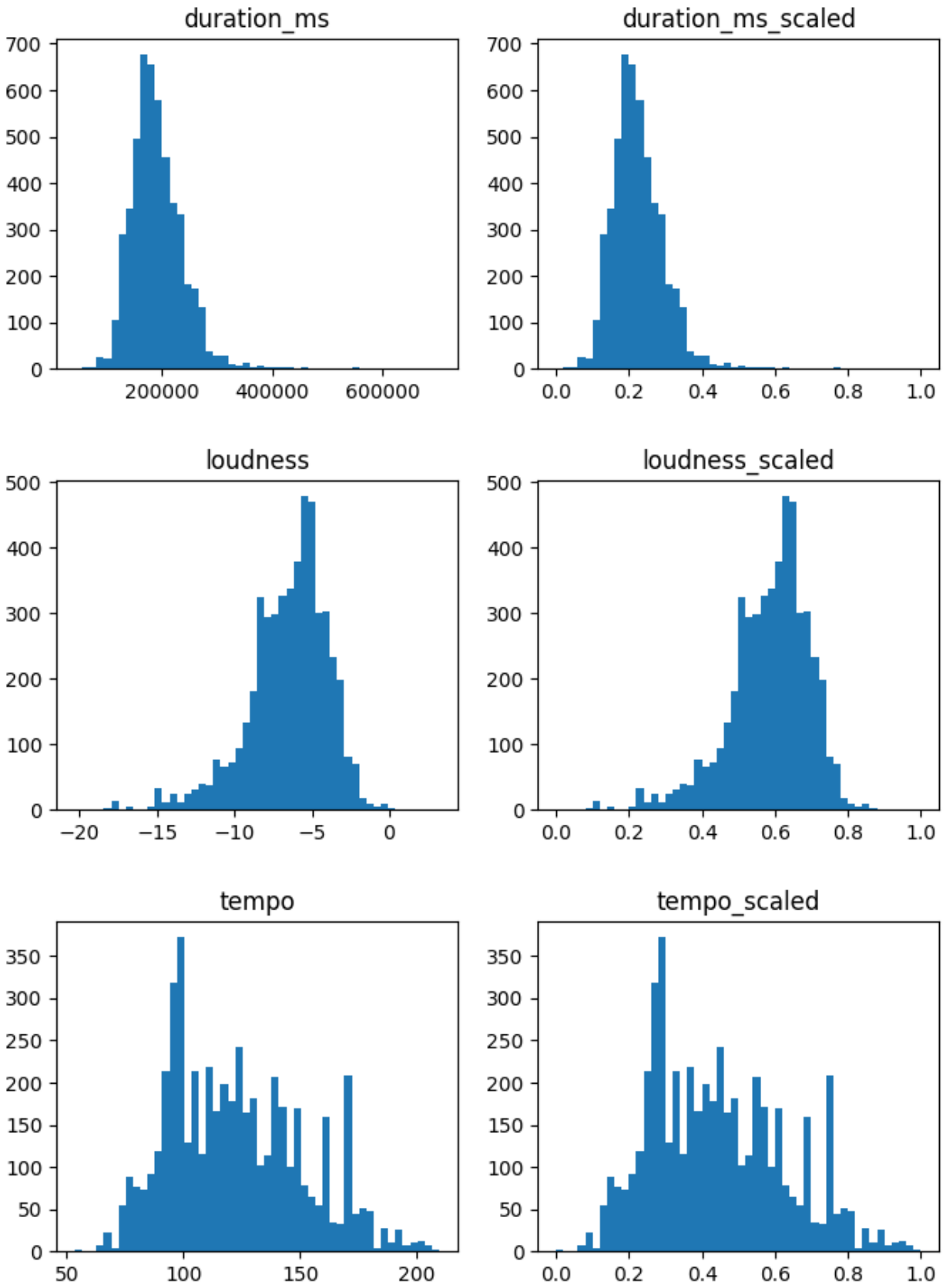
```
country_label_encoder = LabelEncoder()

df['country'] = country_label_encoder.fit_transform(df['country'])

df.head()
```

```
        country  popularity  is_explicit  duration_ms  danceability
energy  \
594957       20          61        False       216816         0.579
0.900
506186       44          83        False       168086         0.743
0.634
154673       57          89         True       197333         0.720
0.880
172136       27          70         True       171782         0.756
0.672
417910       59          94         True       230453         0.679
0.587

        key  loudness  mode  speechiness  acousticness
instrumentalness  \
594957    0    -2.534     0       0.1580        0.4350
0.000000
506186    9    -5.358     1       0.0334        0.0632
0.000000
154673    9    -2.834     1       0.1010        0.0562
0.060000
172136    0    -6.743     1       0.0522        0.4640
0.000003
417910    7    -7.015     1       0.2760        0.1410
0.000006

        liveness  valence    tempo  time_signature
594957     0.365    0.741  186.088               4
506186     0.198    0.263  129.888               3
154673     0.153    0.463  180.011               4
172136     0.108    0.739  114.935               4
417910     0.137    0.486  186.003               4
```

Проведём масштабирование данных

```
df_scaler = MinMaxScaler()

cols_to_scale = ['popularity', 'duration_ms', 'loudness', 'tempo']

for col in cols_to_scale:
    if str(df[col].dtype) not in ['int64', 'float64']:
        continue

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

    df[[col]] = df_scaler.fit_transform(df[[col]])

    ax[1].hist(df[col], 50)
    ax[1].title.set_text(col + '_scaled')
    plt.show()
```




duration_ms
duration_ms_scaled
loudness
loudness_scaled
tempo
tempo_scaled

Как видим, после масштабирования данных распределение значений не изменилось.

```
df.head()
```

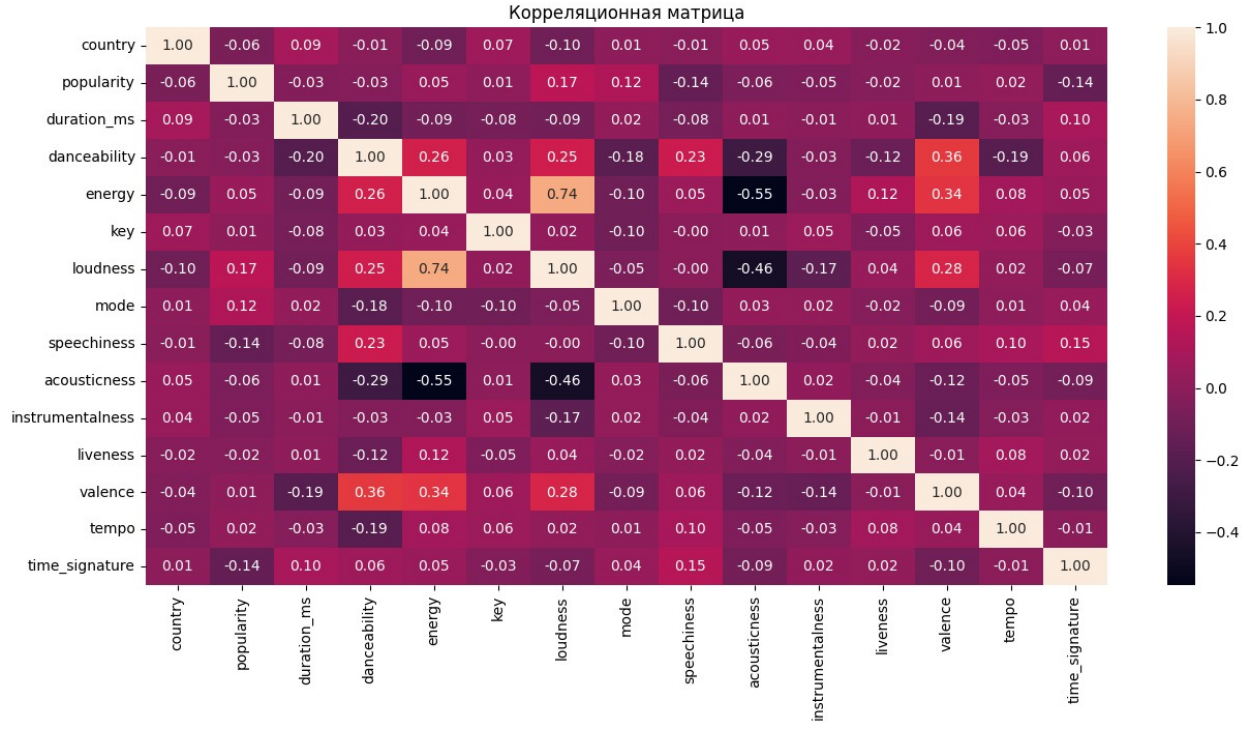
```
        country  popularity  is_explicit  duration_ms  danceability
energy  \
594957       20        0.61        False     0.264224         0.579
0.900
506186       44        0.83        False     0.190502         0.743
0.634
154673       57        0.89         True     0.234749         0.720
0.880
172136       27        0.70         True     0.196093         0.756
0.672
417910       59        0.94         True     0.284856         0.679
0.587

        key  loudness  mode  speechiness  acousticness
instrumentalness  \
594957    0  0.755013     0       0.1580        0.4350
0.000000
506186    9  0.635047     1       0.0334        0.0632
0.000000
154673    9  0.742268     1       0.1010        0.0562
0.060000
172136    0  0.576211     1       0.0522        0.4640
0.000003
417910    7  0.564656     1       0.2760        0.1410
0.000006

        liveness  valence     tempo  time_signature
594957     0.365    0.741  0.848369               4
506186     0.198    0.263  0.489107               3
154673     0.153    0.463  0.809521               4
172136     0.108    0.739  0.393519               4
417910     0.137    0.486  0.847825               4
```

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

```
corr_cols = list(df.columns)
corr_cols.remove('is_explicit')
corr_cols
```

```
['country',
 'popularity',
```

```
 'duration_ms',
 'danceability',
 'energy',
 'key',
 'loudness',
 'mode',
 'speechiness',
 'acousticness',
 'instrumentalness',
 'liveness',
 'valence',
 'tempo',
 'time_signature']

fig, ax = plt.subplots(figsize=(15,7))
sns.heatmap(df[corr_cols].corr(), annot=   1032 if
using_copy_on_write() and astype_is_view(values.dtype,
arr.dtype):True, fmt='.2f')
ax.set_title('Корреляционная матрица')
plt.show()
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Целевой признак `Popularity` слабо напрямую коррелирует с остальными признаками. Максимальные показатели корреляции - `0.16` с показателем `loudness` и `-0.15` с показателем `speechiness`.

- Признаки `loudness` И `energy` имеют очень большую корреляцию (0.74), поэтому уберём признак `energy`, так как признак `loudness` более объективный. Таким образом так же решим проблему высокой отрицательной корреляции призаков `energy` и `acousticness`
- Среди остальных признаков нет слишком сильной корреляции друг с другом, коэффициенты корреляции не превосходят по модулю `0.5` На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения

```
df = df.drop(columns='energy')
```

Выбор метрик для последующей оценки качества моделей

В качестве метрик для решения задачи регрессии будем использовать:

Mean absolute error - средняя абсолютная ошибка

$$MAE(y, \hat{y}) = \frac{1}{N} \Sigma_{i=1}^{N} |y_i - \hat{y}_i)$$

где:

y - истинное значение целевого признака

$\hat{y}$ - предсказанное значение целевого признака

N - размер тестовой выборки

Чем ближе значение к нулю, тем лучше качество регрессии.

Основная проблема метрики состоит в том, что она не нормирована.

Вычисляется с помощью функции mean_absolute_error.

Mean squared error - средняя квадратичная ошибка

$$MSE(y, \hat{y}) = \frac{1}{N} \Sigma_{i=1}^{N} (y_i - \hat{y}_i) 2$$

где:

y - истинное значение целевого признака

$\hat{y}$ - предсказанное значение целевого признака

N - размер тестовой выборки

Вычисляется с помощью функции mean_squared_error.

Метрика R2 или коэффициент детерминации

$$R^2(y,\hat{y}) = 1 - \frac{\Sigma_{i=1}^{N}\left(y_i - \widehat{y_i^2}\right)}{\Sigma_{i=1}^{N}\left(y_i - \overline{y_i^2}\right)}$$

где:

y - истинное значение целевого признака

$\hat{y}$ - предсказанное значение целевого признака

N - размер тестовой выборки

$$\overline{y_i} = \frac{1}{N} \cdot \Sigma_{i=1}^{N} y_i$$

Вычисляется с помощью функции r2_score.

```
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 = pd.DataFrame([{'metric':metric, 'alg':alg,
'value':value}])
        self.df = pd.concat([self.df, 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))
        metric_max = np.max(array_metric)
        metric_min = np.min(array_metric)
        metric_min_max_diff = metric_max - metric_min
        array_metric_normalized = np.copy(array_metric)
        for i in range(len(array_metric)):
            array_metric_normalized[i] = array_metric[i] * 1000.0
        rects = ax1.barh(pos, array_metric_normalized,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)

        for bar,value in zip(rects, array_metric):
            plt.text(bar.get_width()/2, bar.get_y() + bar.get_height()
/ 2, f'{value:.5f}',
                     va='center', ha='left')
            # plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
metric_logger = MetricLogger()
```

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

```
models = {
    'Linear Regression   ': LinearRegression(),
    'K Nearest Neighbors ': KNeighborsRegressor(),
    'Support Vector      ': SVR(),
    'Decision Tree       ': DecisionTreeRegressor(),
    'Random Forest       ': RandomForestRegressor(),
    'Gradient Boosting   ': GradientBoostingRegressor()
}
```

Формирование обучающей и тестовой выборок на основе исходного набора данных

```
from sklearn.model_selection import train_test_split

x_df = df.drop(columns='popularity')
y_df = df['popularity']

x_train, x_test, y_train, y_test = train_test_split(x_df, y_df,
test_size=0.25, random_state=73)

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(3750, 14) (3750,)
(1250, 14) (1250,)
```

Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров

```
def train_model(model_name, model, metric_logger):
    model.fit(x_train, y_train)
    y_predict = model.predict(x_test)

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

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

    print('{} \t MAE={}, MSE={}, R2={}'.format(
        model_name, round(mae, 3), round(mse, 3), round(r2, 3)))

for model_name, model in models.items():
    train_model(model_name, model, metric_logger)
```

```
Linear Regression      MAE=0.118, MSE=0.024, R2=0.101
K Nearest Neighbors    MAE=0.099, MSE=0.022, R2=0.193
Support Vector         MAE=0.121, MSE=0.026, R2=0.059
Decision Tree          MAE=0.082, MSE=0.024, R2=0.128
Random Forest          MAE=0.065, MSE=0.013, R2=0.534
Gradient Boosting      MAE=0.097, MSE=0.018, R2=0.338
```

Подбор гиперпараметров для выбранных моделей

```
tuned_params_knn = [{'n_neighbors': np.array(range(1,2100,100))}]
tuned_params_svr = [{'C': np.array([1.0, 2.0, 3.0, 4.0, 5.0])}]
```

```
tuned_params_dt = [{'max_depth': np.array(range(1, 110, 10))}]
tuned_params_rf = [{'n_estimators': np.array(range(1, 500, 100))}]
tuned_params_gb = [{'n_estimators': np.array(range(1, 1100, 100))}]
```

Сделаем визуализацию для метода ближайших соседей:

```
%%time

regr_gs = GridSearchCV(KNeighborsRegressor(), tuned_params_knn, cv=5,
scoring='neg_mean_squared_error')
regr_gs.fit(x_train, y_train)
```

```
CPU times: user 22.6 s, sys: 46 ms, total: 22.6 s
Wall time: 8.96 s
```

```
GridSearchCV(cv=5, estimator=KNeighborsRegressor(),
             param_grid=[{'n_neighbors': array([   1,  101,  201,
301,  401,  501,  601,  701,  801,  901, 1001,
       1101, 1201, 1301, 1401, 1501, 1601, 1701, 1801, 1901, 2001])}],
             scoring='neg_mean_squared_error')
```

```
# Лучшая модель
regr_gs.best_estimator_
```

```
KNeighborsRegressor(n_neighbors=101)
```

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

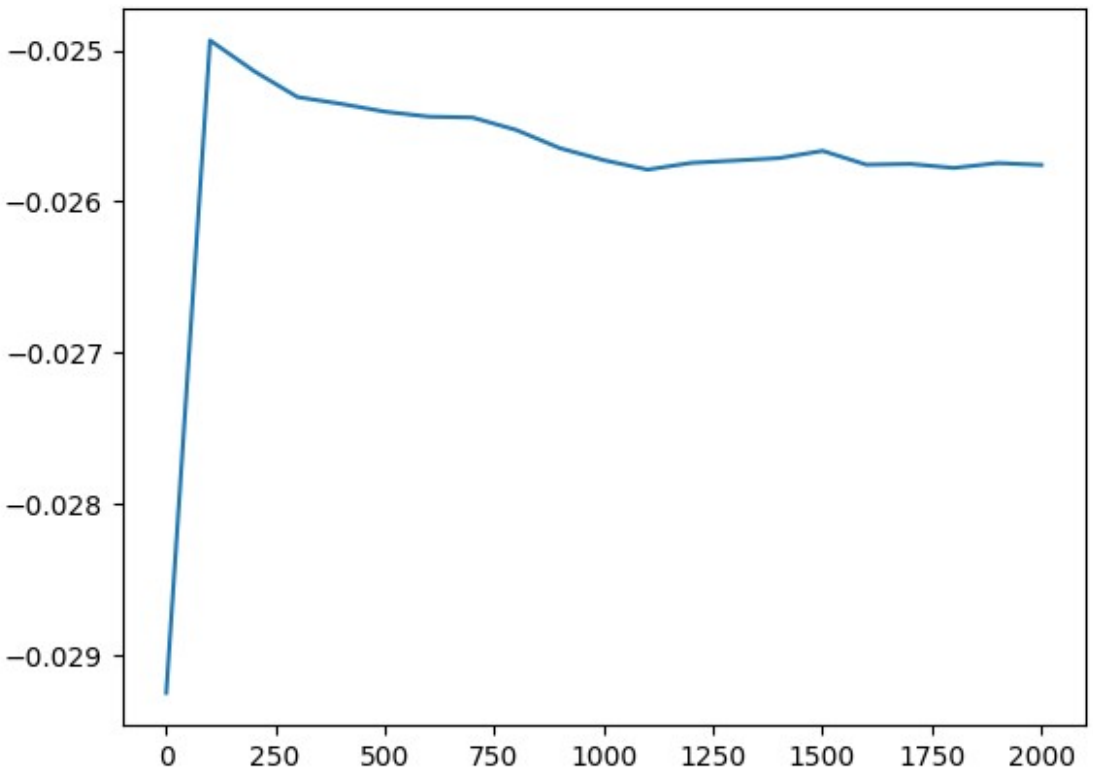
```
{'n_neighbors': 101}
```

```
regr_gs_best_params_txt = str(regr_gs.best_params_['n_neighbors'])
regr_gs_best_params_txt
```

```
'101'
```

```
plt.plot(np.array(range(1,2100,100)),
regr_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7d57ad8258e0>]
```



Аналогично для остальных моделей:

```
%%time

svr_gs = GridSearchCV(SVR(), tuned_params_svr, cv=5,
scoring='neg_mean_squared_error', verbose=1)
svr_gs.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
CPU times: user 6.56 s, sys: 1.69 ms, total: 6.56 s
Wall time: 6.67 s
```

```
GridSearchCV(cv=5, estimator=SVR(),
             param_grid=[{'C': array([1., 2., 3., 4., 5.])}],
             scoring='neg_mean_squared_error', verbose=1)
```

```
svr_gs.best_estimator_
```

```
SVR(C=5.0)
```

```
%%time

dt_gs = GridSearchCV(DecisionTreeRegressor(), tuned_params_dt, cv=5,
scoring='neg_mean_squared_error', verbose=1)
dt_gs.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
CPU times: user 1.64 s, sys: 933 µs, total: 1.64 s
Wall time: 1.66 s

GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid=[{'max_depth': array([  1,  11,  21,  31,  41,
51,  61,  71,  81,  91, 101])}],
             scoring='neg_mean_squared_error', verbose=1)

dt_gs.best_estimator_

DecisionTreeRegressor(max_depth=21)

%%time

rf_gs = GridSearchCV(RandomForestRegressor(), tuned_params_rf, cv=5,
scoring='neg_mean_squared_error', verbose=2)
rf_gs.fit(x_train, y_train)

Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV] END .....................................n_estimators=1; total
time=   0.0s
[CV] END .....................................n_estimators=1; total
time=   0.0s
[CV] END .....................................n_estimators=1; total
time=   0.0s
[CV] END .....................................n_estimators=1; total
time=   0.0s
[CV] END .....................................n_estimators=1; total
time=   0.0s
[CV] END ....................................n_estimators=101; total
time=   2.1s
[CV] END ....................................n_estimators=101; total
time=   2.2s
[CV] END ....................................n_estimators=101; total
time=   2.1s
[CV] END ....................................n_estimators=101; total
time=   2.1s
[CV] END ....................................n_estimators=101; total
time=   2.1s
[CV] END ....................................n_estimators=201; total
time=   4.2s
[CV] END ....................................n_estimators=201; total
time=   4.2s
[CV] END ....................................n_estimators=201; total
time=   4.1s
[CV] END ....................................n_estimators=201; total
time=   4.1s
[CV] END ....................................n_estimators=201; total
time=   4.1s
```

```
[CV] END ..................................n_estimators=301; total
time=   6.1s
[CV] END ..................................n_estimators=301; total
time=   6.2s
[CV] END ..................................n_estimators=301; total
time=   6.2s
[CV] END ..................................n_estimators=301; total
time=   6.1s
[CV] END ..................................n_estimators=301; total
time=   6.1s
[CV] END ..................................n_estimators=401; total
time=   8.1s
[CV] END ..................................n_estimators=401; total
time=   8.3s
[CV] END ..................................n_estimators=401; total
time=   8.2s
[CV] END ..................................n_estimators=401; total
time=   8.2s
[CV] END ..................................n_estimators=401; total
time=   8.2s
CPU times: user 1min 49s, sys: 414 ms, total: 1min 49s
Wall time: 1min 50s

GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid=[{'n_estimators': array([  1, 101, 201, 301,
401])}],
             scoring='neg_mean_squared_error', verbose=2)

rf_gs.best_estimator_

RandomForestRegressor(n_estimators=301)

%%time

gb_gs = GridSearchCV(GradientBoostingRegressor(), tuned_params_gb,
cv=5, scoring='neg_mean_squared_error', verbose=2)
gb_gs.fit(x_train, y_train)

Fitting 5 folds for each of 11 candidates, totalling 55 fits
[CV] END ..................................n_estimators=1; total
time=   0.0s
[CV] END ..................................n_estimators=1; total
time=   0.0s
[CV] END ..................................n_estimators=1; total
time=   0.0s
[CV] END ..................................n_estimators=1; total
time=   0.0s
[CV] END ..................................n_estimators=1; total
time=   0.0s
[CV] END ..................................n_estimators=101; total
```

```
time=   0.7s
[CV] END ..................................n_estimators=101; total
time=   0.7s
[CV] END ..................................n_estimators=101; total
time=   0.7s
[CV] END ..................................n_estimators=101; total
time=   0.7s
[CV] END ..................................n_estimators=101; total
time=   0.7s
[CV] END ..................................n_estimators=201; total
time=   1.3s
[CV] END ..................................n_estimators=201; total
time=   1.3s
[CV] END ..................................n_estimators=201; total
time=   1.3s
[CV] END ..................................n_estimators=201; total
time=   1.3s
[CV] END ..................................n_estimators=201; total
time=   1.3s
[CV] END ..................................n_estimators=301; total
time=   2.0s
[CV] END ..................................n_estimators=301; total
time=   2.0s
[CV] END ..................................n_estimators=301; total
time=   2.0s
[CV] END ..................................n_estimators=301; total
time=   1.9s
[CV] END ..................................n_estimators=301; total
time=   1.9s
[CV] END ..................................n_estimators=401; total
time=   2.5s
[CV] END ..................................n_estimators=401; total
time=   2.5s
[CV] END ..................................n_estimators=401; total
time=   2.5s
[CV] END ..................................n_estimators=401; total
time=   2.5s
[CV] END ..................................n_estimators=401; total
time=   2.6s
[CV] END ..................................n_estimators=501; total
time=   3.2s
[CV] END ..................................n_estimators=501; total
time=   3.3s
[CV] END ..................................n_estimators=501; total
time=   3.2s
[CV] END ..................................n_estimators=501; total
time=   3.2s
[CV] END ..................................n_estimators=501; total
time=   3.3s
```

```
[CV] END ..................................n_estimators=601; total
time=   3.9s
[CV] END ..................................n_estimators=601; total
time=   3.9s
[CV] END ..................................n_estimators=601; total
time=   3.9s
[CV] END ..................................n_estimators=601; total
time=   3.9s
[CV] END ..................................n_estimators=601; total
time=   3.8s
[CV] END ..................................n_estimators=701; total
time=   4.4s
[CV] END ..................................n_estimators=701; total
time=   4.4s
[CV] END ..................................n_estimators=701; total
time=   4.5s
[CV] END ..................................n_estimators=701; total
time=   4.4s
[CV] END ..................................n_estimators=701; total
time=   4.5s
[CV] END ..................................n_estimators=801; total
time=   5.1s
[CV] END ..................................n_estimators=801; total
time=   5.1s
[CV] END ..................................n_estimators=801; total
time=   5.1s
[CV] END ..................................n_estimators=801; total
time=   5.2s
[CV] END ..................................n_estimators=801; total
time=   5.1s
[CV] END ..................................n_estimators=901; total
time=   5.6s
[CV] END ..................................n_estimators=901; total
time=   5.7s
[CV] END ..................................n_estimators=901; total
time=   5.7s
[CV] END ..................................n_estimators=901; total
time=   5.8s
[CV] END ..................................n_estimators=901; total
time=   5.7s
[CV] END .................................n_estimators=1001; total
time=   6.3s
[CV] END .................................n_estimators=1001; total
time=   6.2s
[CV] END .................................n_estimators=1001; total
time=   6.2s
[CV] END .................................n_estimators=1001; total
time=   6.2s
[CV] END .................................n_estimators=1001; total
time=   6.3s
```

```
CPU times: user 3min 1s, sys: 92.6 ms, total: 3min 2s
Wall time: 3min 2s

GridSearchCV(cv=5, estimator=GradientBoostingRegressor(),
             param_grid=[{'n_estimators': array([   1,  101,  201,
301,  401,  501,  601,  701,  801,  901, 1001])}],
             scoring='neg_mean_squared_error', verbose=2)

gb_gs.best_estimator_

GradientBoostingRegressor(n_estimators=1001)
```

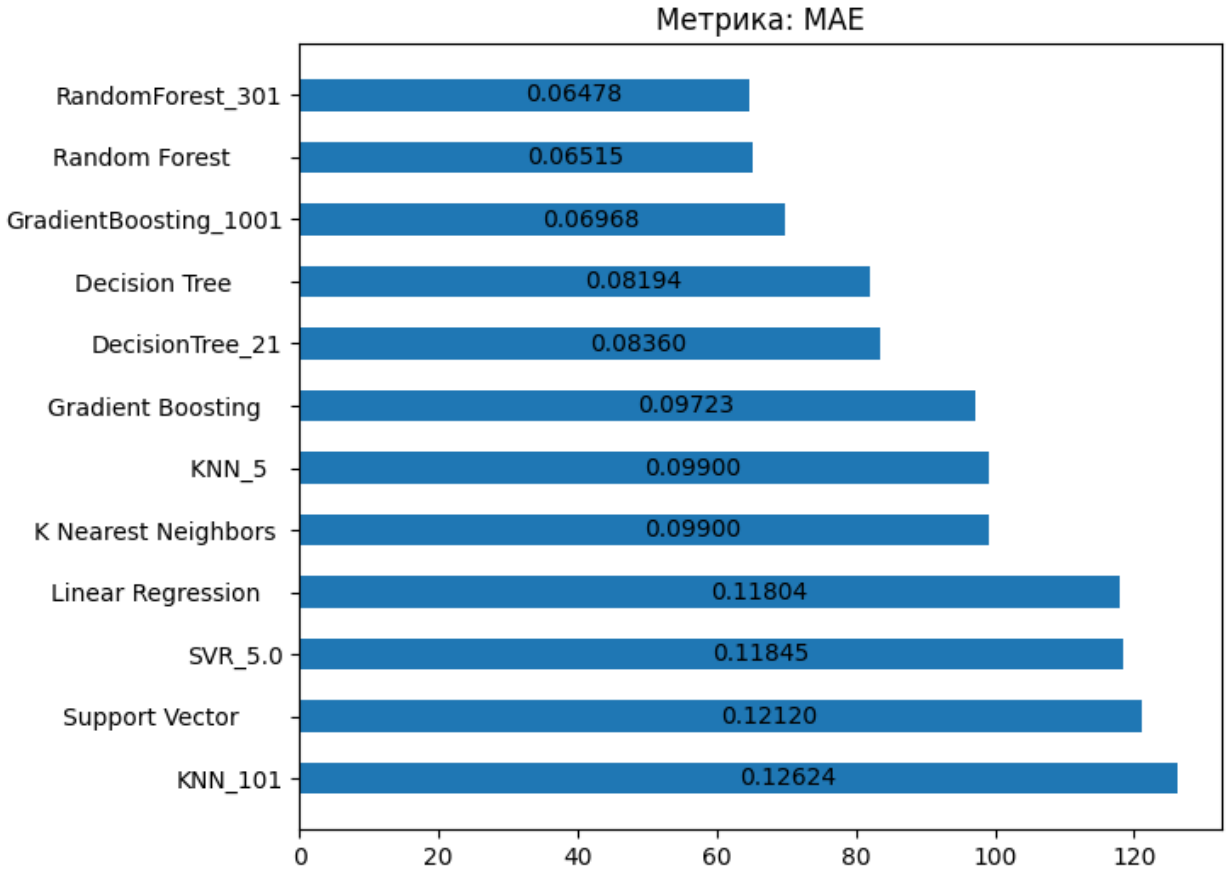
Теперь обучим и добавим в `metric_logger` модели с оптимальными гиперпараметрами:

```
regr_models_grid = {'KNN_5' + ' ' * (len('KNN_' +
regr_gs_best_params_txt) -
len('KNN_5')) :KNeighborsRegressor(n_neighbors=5),

str('KNN_'+regr_gs_best_params_txt):regr_gs.best_estimator_,

str('SVR_'+str(svr_gs.best_params_['C'])):svr_gs.best_estimator_,

str('DecisionTree_'+str(dt_gs.best_params_['max_depth'])):dt_gs.best_e
stimator_,

str('RandomForest_'+str(rf_gs.best_params_['n_estimators'])):rf_gs.bes
t_estimator_,

str('GradientBoosting_'+str(gb_gs.best_params_['n_estimators'])):gb_gs
.best_estimator_}

for model_name, model in regr_models_grid.items():
    train_model(model_name, model, metric_logger)

KNN_5       MAE=0.099, MSE=0.022, R2=0.193
KNN_101     MAE=0.126, MSE=0.026, R2=0.029
SVR_5.0     MAE=0.118, MSE=0.025, R2=0.087
DecisionTree_21  MAE=0.084, MSE=0.024, R2=0.124
RandomForest_301       MAE=0.065, MSE=0.012, R2=0.544
GradientBoosting_1001       MAE=0.07, MSE=0.013, R2=0.509
```

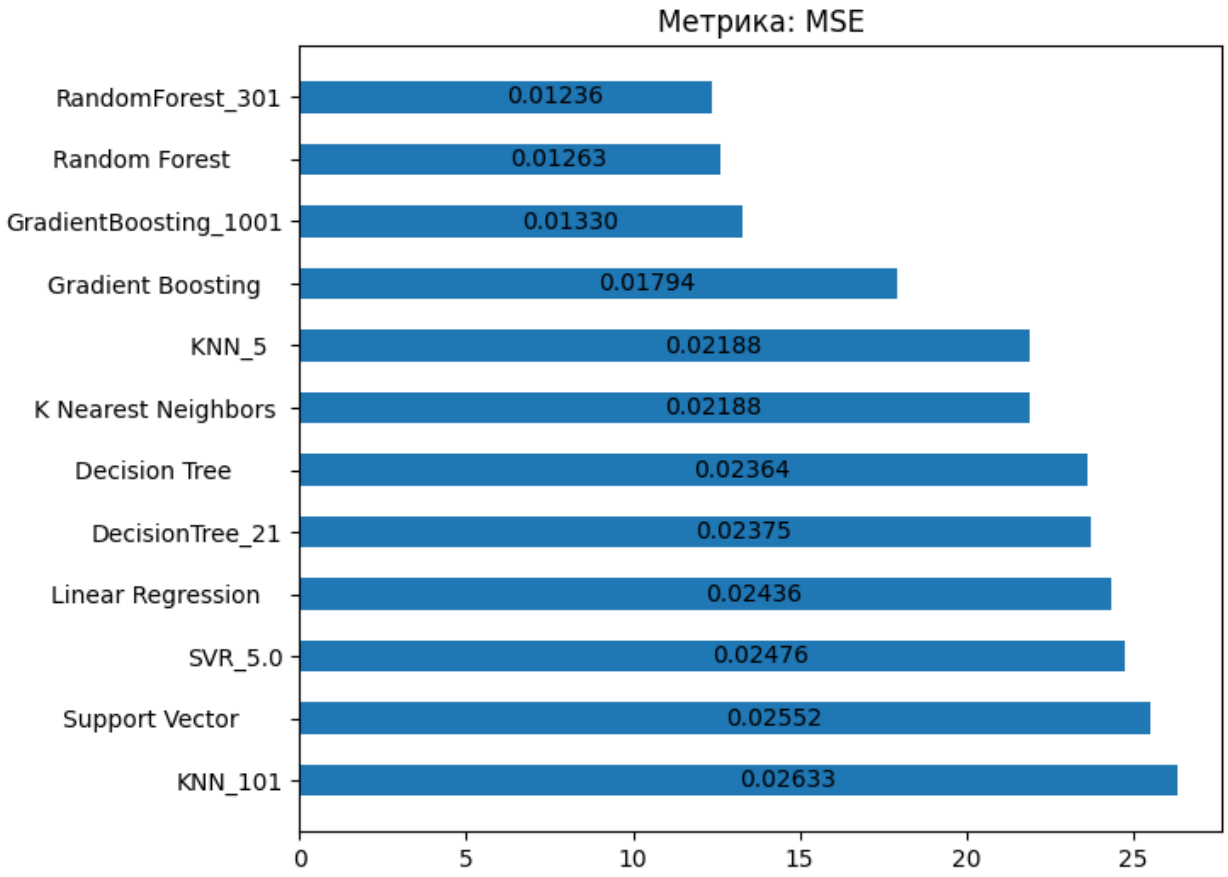
Формирование выводов о качестве построенных моделей на основе выбранных метрик.

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

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

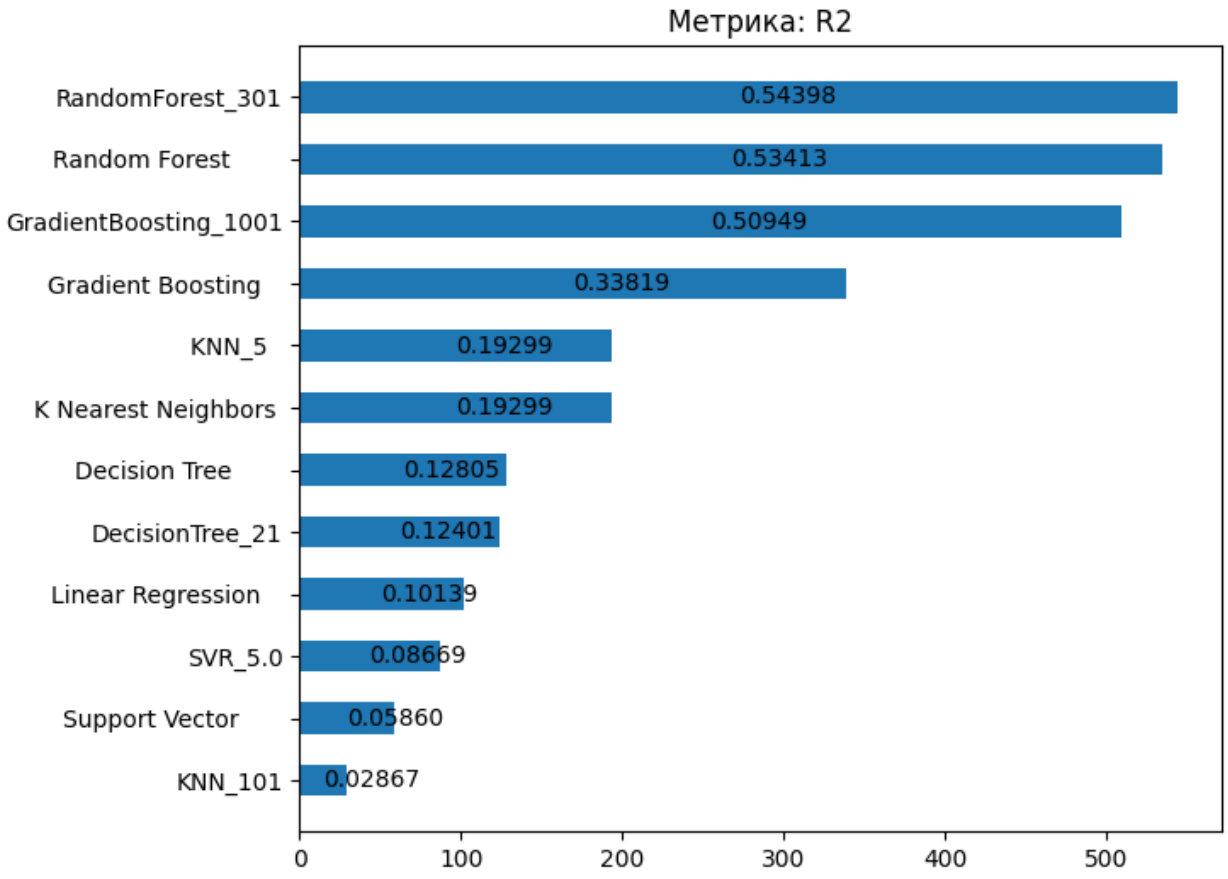
```
metric_logger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False,
figsize=(7, 6))
```




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




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



Вывод: лучшими оказались модели на основе случайного леса и градиентного бустинга. При отдельных запусках вместо градиентного бустинга оказывается лучшей модель решающего дерева.