Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Лабораторная работа № 3

по дисциплине «Методы машинного обучения»

Обработка признаков, часть 2

ИСПОЛНИТЕЛЬ:

студент ИУ5-23М

Бондаренко И. Г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю. Е.

___ "________" 2024 г.

Москва, 2024

Задание лабораторной работы

- Выбрать один или несколько наборов данных (датасетов) для решения следующих задач. Каждая задача может быть решена на отдельном датасете, или несколько задач могут быть решены на одном датасете. Просьба не использовать датасет, на котором данная задача решалась в лекции.
- Для выбранного датасета (датасетов) на основе материалов лекций решить следующие задачи:
  - масштабирование признаков (не менее чем тремя способами);
  - обработку выбросов для числовых признаков (по одному способу для удаления выбросов и для замены выбросов);
  - обработку по крайней мере одного нестандартного признака (который не является числовым или категориальным);
  - отбор признаков:
    - один метод из группы методов фильтрации (filter methods);
    - один метод из группы методов обертывания (wrapper methods);
    - один метод из группы методов вложений (embedded methods).

Выполнение работы

Импорт библиотек

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MaxAbsScaler
import scipy.stats as stats
```

Подключение Google Диска для работы с Google Colab

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Чтение данных

```
data = pd.read_csv('/content/drive/MyDrive/MMO/PopularSpotifySongs.csv', encoding='unicode_escape')
```

```
data.head()
```

	track_name	artist(s)_name	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_charts	
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	14	553	147	1
1	LALA	Myke Towers	1	2023	3	23	1474	48	1
2	vampire	Olivia Rodrigo	1	2023	6	30	1397	113	1
3	Cruel Summer	Taylor Swift	1	2019	8	23	7858	100	8
4	WHERE SHE GOES	Bad Bunny	1	2023	5	18	3133	50	3

5 rows × 24 columns

```
data.shape
```

```
(953, 24)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 953 entries, 0 to 952
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   track_name            953 non-null    object
 1   artist(s)_name        953 non-null    object
 2   artist_count          953 non-null    int64 
 3   released_year         953 non-null    int64 
 4   released_month        953 non-null    int64 
 5   released_day          953 non-null    int64 
 6   in_spotify_playlists  953 non-null    int64 
 7   in_spotify_charts     953 non-null    int64 
 8   streams               953 non-null    object
 9   in_apple_playlists    953 non-null    int64 
 10  in_apple_charts       953 non-null    int64 
 11  in_deezer_playlists   953 non-null    object
 12  in_deezer_charts      953 non-null    int64 
 13  in_shazam_charts      903 non-null    object
 14  bpm                   953 non-null    int64 
 15  key                   858 non-null    object
 16  mode                  953 non-null    object
 17  danceability_%        953 non-null    int64 
 18  valence_%             953 non-null    int64 
 19  energy_%              953 non-null    int64 
 20  acousticness_%        953 non-null    int64 
 21  instrumentalness_%    953 non-null    int64 
 22  liveness_%            953 non-null    int64 
 23  speechiness_%         953 non-null    int64 
dtypes: int64(17), object(7)
memory usage: 178.8+ KB
```

Первичная обработка данных

Оставим в исходной выборке лишь некоторые признаки:

```
data.drop(['streams', 'artist(s)_name', 'in_shazam_charts', 'in_spotify_playlists', 'in_spotify_charts', 'in_apple_playlists', 'in_apple
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 953 entries, 0 to 952
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   track_name          953 non-null    object
 1   artist_count        953 non-null    int64 
 2   released_year       953 non-null    int64 
 3   released_month      953 non-null    int64 
 4   released_day        953 non-null    int64 
 5   bpm                 953 non-null    int64 
 6   key                 858 non-null    object
 7   mode                953 non-null    object
 8   danceability_%      953 non-null    int64 
 9   valence_%           953 non-null    int64 
 10  energy_%            953 non-null    int64 
 11  acousticness_%      953 non-null    int64 
 12  instrumentalness_%  953 non-null    int64 
 13  liveness_%          953 non-null    int64 
 14  speechiness_%       953 non-null    int64 
dtypes: int64(12), object(3)
memory usage: 111.8+ KB
```

Удалим пропуски:

```
for column in data.columns:
  if (data[column].isnull().sum() != 0):
    print(column,':',data[column].isnull().sum())
```

```
key : 95
```

```
data.drop(data[data['key'].isnull()].index, inplace=True)
```

```
for column in data.columns:
  if (data[column].isnull().sum() != 0):
    print(column,':',data[column].isnull().sum())
```

Приведем бинарные свойства к int64:

Закодируем признаки:

```
data["mode"]=data["mode"].apply(lambda x: x == 'Major').astype('int64')
```

```
data.head()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	valence_%	energy_%	acoust
0	Seven (feat. Latto) (Explicit Ver.)	2	2023	7	14	125	B	1	80	89	83	
1	LALA	1	2023	3	23	92	C#	1	71	61	74	
2	vampire	1	2023	6	30	138	F	1	51	32	53	

LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
```

```
letype = LabelEncoder()
learrtype = letype.fit_transform(data["key"])
data["key"] = learrtype
data = data.astype({"key":"int64"})
```

```
leeng = LabelEncoder()
learren = leeng.fit_transform(data["track_name"])
data["track_name"] = learren
data = data.astype({"track_name":"int64"})
```

CountEncoder

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 3.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.0.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoc
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (202
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (20
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoder
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encoder
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

```
from category_encoders.count import CountEncoder as ce_CountEncoder
```

```
ce_CountEncoder1 = ce_CountEncoder()
data["track_name"] = ce_CountEncoder1.fit_transform(data['track_name'])
```

```
Warning: No categorical columns found. Calling 'transform' will only return input data.
```

```
ce_CountEncoder2 = ce_CountEncoder()
data["artist_count"] = ce_CountEncoder2.fit_transform(data['artist_count'])
```

```
Warning: No categorical columns found. Calling 'transform' will only return input data.
```

FrequencyEncoder

```
ce_CountEncoder3 = ce_CountEncoder(normalize=True)
data["bpm"] = ce_CountEncoder3.fit_transform(data['bpm'])
```

```
Warning: No categorical columns found. Calling 'transform' will only return input data.
```

```
data.head()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	valence_%	energy_%	acoust
0	618	2	2023	7	14	125	2	1	80	89	83	
1	357	1	2023	3	23	92	3	1	71	61	74	
2	845	1	2023	6	30	138	7	1	51	32	53	
3	153	1	2019	8	23	170	0	1	55	58	72	

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 858 entries, 0 to 952
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   track_name          858 non-null    int64
 1   artist_count        858 non-null    int64
 2   released_year       858 non-null    int64
 3   released_month      858 non-null    int64
 4   released_day        858 non-null    int64
 5   bpm                 858 non-null    int64
 6   key                 858 non-null    int64
 7   mode                858 non-null    int64
 8   danceability_%      858 non-null    int64
 9   valence_%           858 non-null    int64
 10  energy_%            858 non-null    int64
 11  acousticness_%      858 non-null    int64
 12  instrumentalness_%  858 non-null    int64
 13  liveness_%          858 non-null    int64
 14  speechiness_%       858 non-null    int64
dtypes: int64(15)
memory usage: 107.2 KB
```

Разделение выборки

```
data.describe()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	val
count	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.
mean	425.247086	1.551282	2018.241259	6.025641	13.724942	122.827506	5.152681	0.553613	67.256410	51.
std	246.031744	0.864335	11.107781	3.569192	9.292416	28.183522	3.230362	0.497407	14.652712	23.
min	0.000000	1.000000	1930.000000	1.000000	1.000000	65.000000	0.000000	0.000000	23.000000	4.
25%	212.250000	1.000000	2020.000000	3.000000	5.000000	100.000000	3.000000	0.000000	57.000000	32.
50%	425.500000	1.000000	2022.000000	5.000000	13.000000	121.000000	5.000000	1.000000	70.000000	51.
75%	637.750000	2.000000	2022.000000	9.000000	22.000000	141.750000	8.000000	1.000000	78.000000	70.

В качестве целевого признака возьмем признак `price`.

```
# DataFrame не содержащий целевой признак
Y = data['valence_%']
X_ALL = data.drop('valence_%', axis=1)
```

```
X_ALL
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	energy_%	acousticness_%
0	618	2	2023	7	14	125	2	1	80	83	31
1	357	1	2023	3	23	92	3	1	71	74	7
2	845	1	2023	6	30	138	7	1	51	53	17
3	153	1	2019	8	23	170	0	1	55	72	11
4	781	1	2023	5	18	144	0	0	65	80	14
...	...	...	...	...	...	...	...	...	...	...	...
948	470	1	2022	11	3	144	0	1	60	39	57
949	86	1	2022	10	21	166	8	1	42	24	83
950	11	2	2022	11	3	92	3	1	80	67	4
951	213	3	2022	10	20	97	3	1	82	77	8
952	39	1	2022	11	4	90	6	0	61	67	15

```
Y
```

```
0      89
1      61
2      32
3      58
4      23
       ..
948    24
949     7
950    81
951    67
952    32
Name: valence_%, Length: 858, dtype: int64
```

```
# Функция для восстановления датафрейма
# на основе масштабированных данных
def arr_to_df(arr_scaled):
    res = pd.DataFrame(arr_scaled, columns=X_ALL.columns)
    return res
```

```
# Разделим выборку на обучающую и тестовую
X_train, X_test, y_train, y_test = train_test_split(X_ALL, data['valence_%'],
                                                    test_size=0.2,
                                                    random_state=1)
# Преобразуем массивы в DataFrame
X_train_df = arr_to_df(X_train)
X_test_df = arr_to_df(X_test)

X_train_df.shape, X_test_df.shape
```

```
((686, 14), (172, 14))
```

Масштабирование признаков

Масштабирование на основе Z-оценки

```
x_col_list = ['bpm', 'artist_count', 'key']
```

```
# Обучаем StandardScaler на всей выборке и масштабируем
cs11 = StandardScaler()
data_cs11_scaled_temp = cs11.fit_transform(X_ALL)
# формируем DataFrame на основе массива
data_cs11_scaled =  arr_to_df(data_cs11_scaled_temp)
data_cs11_scaled
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	energy_%
0	0.783904	0.519451	0.428665	0.273151	0.029618	0.077129	-0.976522	0.897951	0.870216	1.163146
1	-0.277553	-0.638182	0.428665	-0.848205	0.998714	-1.094451	-0.666779	0.897951	0.255637	0.601993
2	1.707088	-0.638182	0.428665	-0.007188	1.752456	0.538660	0.572195	0.897951	-1.110094	-0.707364
3	-1.107198	-0.638182	0.068347	0.553490	0.998714	1.674738	-1.596009	0.897951	-0.836948	0.477292
4	1.446807	-0.638182	0.428665	-0.287527	0.460327	0.751675	-1.596009	-1.113647	-0.154082	0.976095
...	...	...	...	...	...	...	...	...	...	...
853	0.182005	-0.638182	0.338585	1.394506	-1.154834	0.751675	-1.596009	0.897951	-0.495515	-1.580268
854	-1.379680	-0.638182	0.338585	1.114167	0.783359	1.532728	0.881938	0.897951	-1.724673	-2.515523
855	-1.684696	0.519451	0.338585	1.394506	-1.154834	-1.094451	-0.666779	0.897951	0.870216	0.165541
856	-0.863185	1.677084	0.338585	1.114167	0.675682	-0.916939	-0.666779	0.897951	1.006789	0.789044
857	-1.570823	-0.638182	0.338585	1.394506	-1.047157	-1.165456	0.262452	-1.113647	-0.427229	0.165541

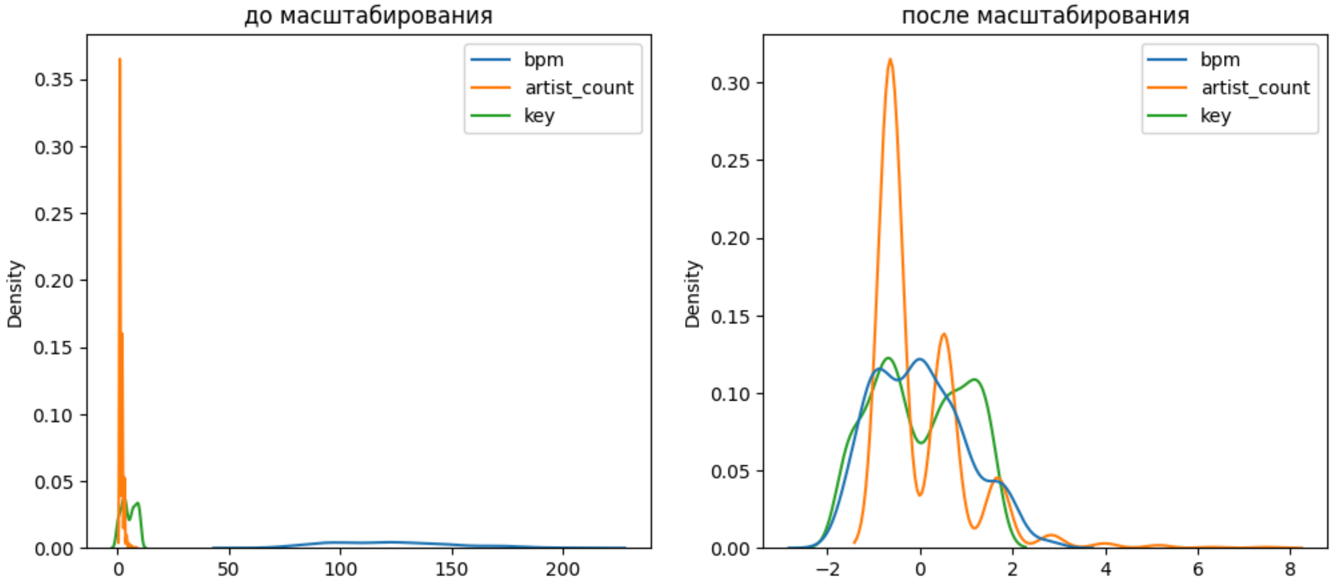
```
data_cs11_scaled.describe()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceabil
count	8.580000e+02	8.580000e+02	8.580000e+02	8.580000e+02	8.580000e+02	8.580000e+02	8.580000e+02	8.580000e+02	8.58000
mean	-4.347727e-17	-4.347727e-17	7.155116e-15	2.484415e-17	-6.625107e-17	-1.407835e-16	-3.519588e-17	1.086932e-16	-4.2028
std	1.000583e+00	1.000583e+00	1.000583e+00	1.000583e+00	1.000583e+00	1.000583e+00	1.000583e+00	1.000583e+00	1.00058
min	-1.729432e+00	-6.381825e-01	-7.948728e+00	-1.408883e+00	-1.370189e+00	-2.053016e+00	-1.596009e+00	-1.113647e+00	-3.02211
25%	-8.662350e-01	-6.381825e-01	1.584265e-01	-8.482048e-01	-9.394791e-01	-8.104317e-01	-6.667786e-01	-1.113647e+00	-7.0037
50%	1.028572e-03	-6.381825e-01	3.385855e-01	-2.875270e-01	-7.805985e-02	-6.488088e-02	-4.729182e-02	8.979509e-01	1.8735
75%	8.642253e-01	5.194509e-01	3.385855e-01	8.338284e-01	8.910368e-01	6.717943e-01	8.819384e-01	8.979509e-01	7.3364

Построим плотность распределения:

```
def draw_kde(col_list, df1, df2, label1, label2):
    fig, (ax1, ax2) = plt.subplots(
        ncols=2, figsize=(12, 5))
    # первый график
    ax1.set_title(label1)
    sns.kdeplot(data=df1[col_list], ax=ax1)
    # второй график
    ax2.set_title(label2)
    sns.kdeplot(data=df2[col_list], ax=ax2)
    plt.show()
```

```
draw_kde(x_col_list, data, data_cs11_scaled, 'до масштабирования', 'после масштабирования')
```



Обучаем StandardScaler на обучающей выборке и масштабируем обучающую и тестовую выборки:

```
cs12 = StandardScaler()
cs12.fit(X_train)
data_cs12_scaled_train_temp = cs12.transform(X_train)
data_cs12_scaled_test_temp = cs12.transform(X_test)
# формируем DataFrame на основе массива
data_cs12_scaled_train = arr_to_df(data_cs12_scaled_train_temp)
data_cs12_scaled_test = arr_to_df(data_cs12_scaled_test_temp)
```

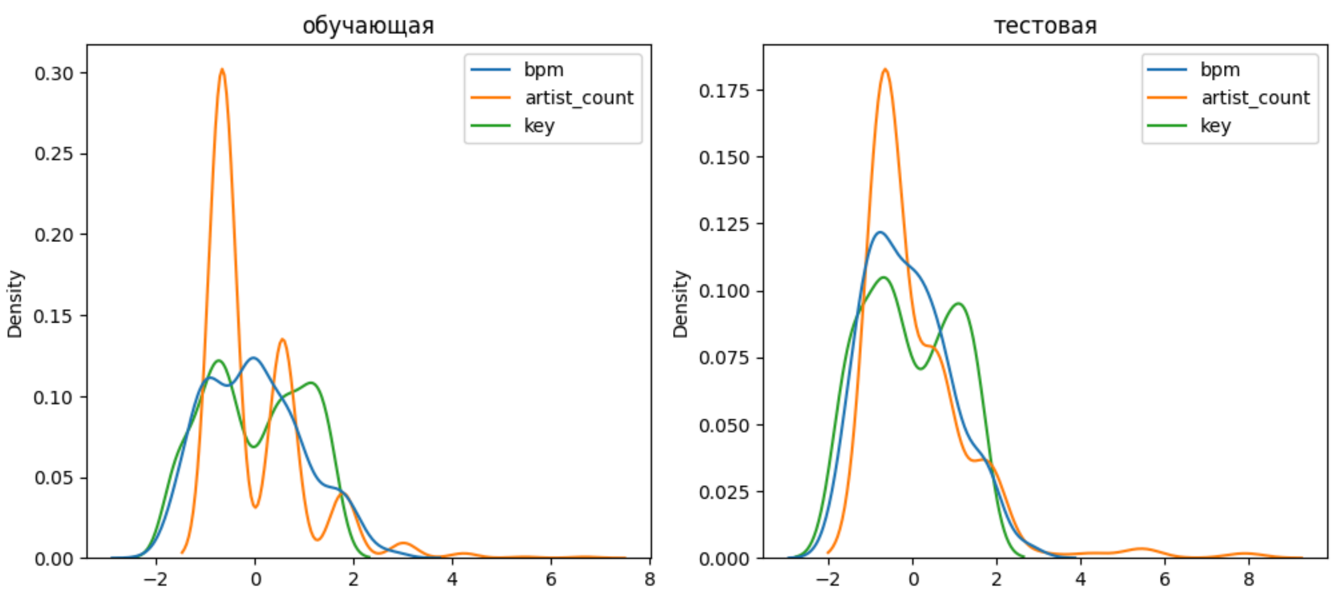
```
data_cs12_scaled_train.describe()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceabil
count	6.860000e+02	6.860000e+02	6.860000e+02	6.860000e+02	6.860000e+02	6.860000e+02	6.860000e+02	6.860000e+02	6.86000
mean	-9.775141e-17	-5.955715e-17	4.987264e-15	1.553665e-17	6.214659e-17	6.991492e-17	-1.204090e-16	-5.178883e-17	1.99387
std	1.000730e+00	1.000730e+00	1.000730e+00	1.000730e+00	1.000730e+00	1.000730e+00	1.000730e+00	1.000730e+00	1.00073
min	-1.724847e+00	-6.542721e-01	-7.921048e+00	-1.374955e+00	-1.342454e+00	-2.072496e+00	-1.629213e+00	-1.098084e+00	-3.07366
25%	-8.630952e-01	-6.542721e-01	1.575417e-01	-8.189757e-01	-9.133193e-01	-8.289468e-01	-6.884385e-01	-1.098084e+00	-6.97704
50%	2.697955e-03	-6.542721e-01	3.370659e-01	-2.629962e-01	-1.086912e-01	-8.281699e-02	-6.125548e-02	9.106774e-01	1.86776
75%	8.806142e-01	5.686997e-01	3.370659e-01	8.489630e-01	9.105043e-01	6.633128e-01	8.795190e-01	9.106774e-01	7.41744

```
data_cs12_scaled_test.describe()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	en
count	172.000000	172.000000	172.000000	172.000000	172.000000	172.000000	172.000000	172.000000	172.000000	172.
mean	-0.031956	0.099420	-0.001630	0.110351	0.113361	-0.089221	-0.066725	0.069801	-0.017707	0.
std	0.970286	1.257091	0.985165	0.955154	0.979230	1.003615	1.061938	0.993876	1.079923	0.
min	-1.720806	-0.654272	-5.497471	-1.374955	-1.342454	-1.859317	-1.629213	-1.098084	-3.004292	-2.
25%	-0.901485	-0.654272	0.157542	-0.540986	-0.725573	-0.900007	-1.002030	-1.098084	-0.784419	-0.
50%	-0.017507	-0.654272	0.337066	0.014994	0.052234	-0.189407	-0.374847	0.910677	0.048034	0.
75%	0.753321	0.568700	0.337066	0.918460	0.910504	0.592253	0.879519	0.910677	0.880487	0.

```
draw_kde(x_col_list, data_cs12_scaled_train, data_cs12_scaled_test, 'обучающая', 'тестовая')
```



Масштабирование Mean Normalization

```
class MeanNormalisation:

    def fit(self, param_df):
        self.means = X_train.mean(axis=0)
        maxs = X_train.max(axis=0)
        mins = X_train.min(axis=0)
        self.ranges = maxs - mins

    def transform(self, param_df):
        param_df_scaled = (param_df - self.means) / self.ranges
        return param_df_scaled

    def fit_transform(self, param_df):
        self.fit(param_df)
        return self.transform(param_df)
```

```
sc21 = MeanNormalisation()
data_cs21_scaled = sc21.fit_transform(X_ALL)
data_cs21_scaled.describe()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	en
count	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.
mean	-0.001865	0.002716	-0.000039	0.007234	0.007061	-0.003570	-0.004265	0.006966	-0.000701	0.
std	0.289449	0.144056	0.119439	0.324472	0.309747	0.199883	0.323036	0.497407	0.200722	0.
min	-0.502156	-0.089164	-0.948870	-0.449642	-0.417104	-0.413694	-0.519534	-0.546647	-0.606953	-0.
25%	-0.252450	-0.089164	0.018872	-0.267824	-0.283771	-0.165467	-0.219534	-0.546647	-0.141200	-0.
50%	-0.001567	-0.089164	0.040377	-0.086006	-0.017104	-0.016531	-0.019534	0.453353	0.036882	0.
75%	0.248138	0.077502	0.040377	0.277631	0.282896	0.130632	0.280466	0.453353	0.146472	0.

```
cs22 = MeanNormalisation()
cs22.fit(X_train)
data_cs22_scaled_train = cs22.transform(X_train)
data_cs22_scaled_test = cs22.transform(X_test)
```

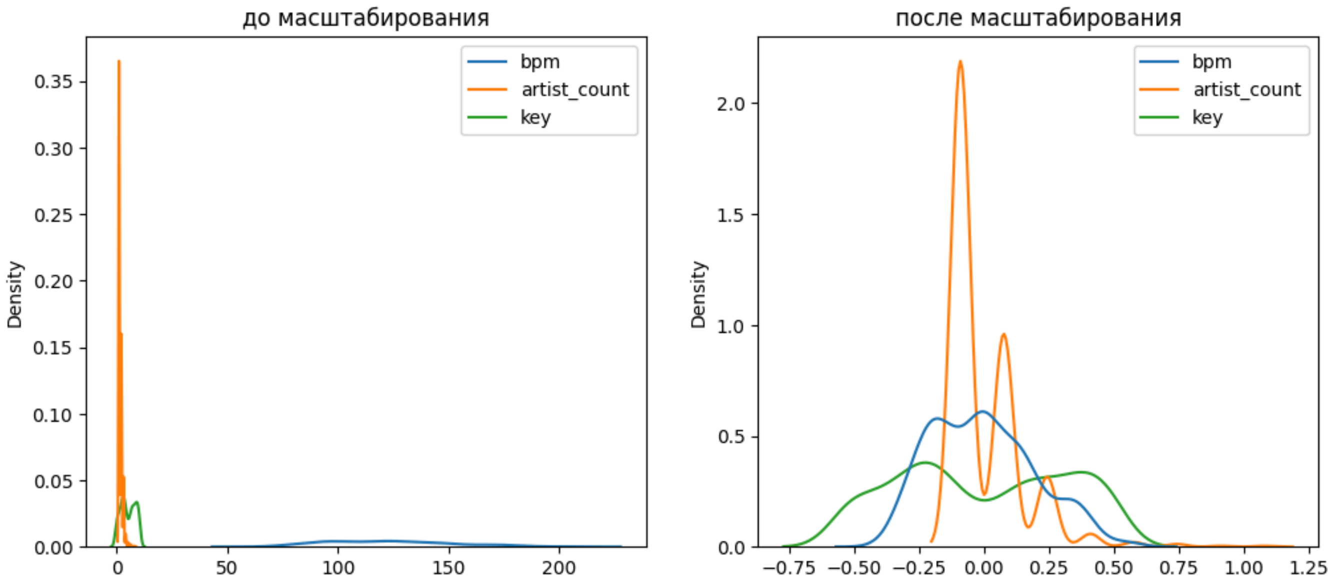
```
data_cs22_scaled_train.describe()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability
count	6.860000e+02	6.860000e+02	6.860000e+02	686.000000	6.860000e+02	6.860000e+02	6.860000e+02	6.860000e+02	6.860000e+
mean	-2.605625e-17	-1.035777e-17	5.988083e-16	0.000000	1.553665e-17	1.618401e-17	-4.110738e-17	-1.294721e-17	4.143106e
std	2.913430e-01	1.363795e-01	1.198784e-01	0.327262	3.109293e-01	1.997573e-01	3.191189e-01	4.981825e-01	1.976130e
min	-5.021557e-01	-8.916424e-02	-9.488699e-01	-0.449642	-4.171040e-01	-4.136944e-01	-5.195335e-01	-5.466472e-01	-6.069532e
25%	-2.512734e-01	-8.916424e-02	1.887206e-02	-0.267824	-2.837707e-01	-1.654674e-01	-2.195335e-01	-5.466472e-01	-1.377751e
						-1.653123e-	-1.953353e-		

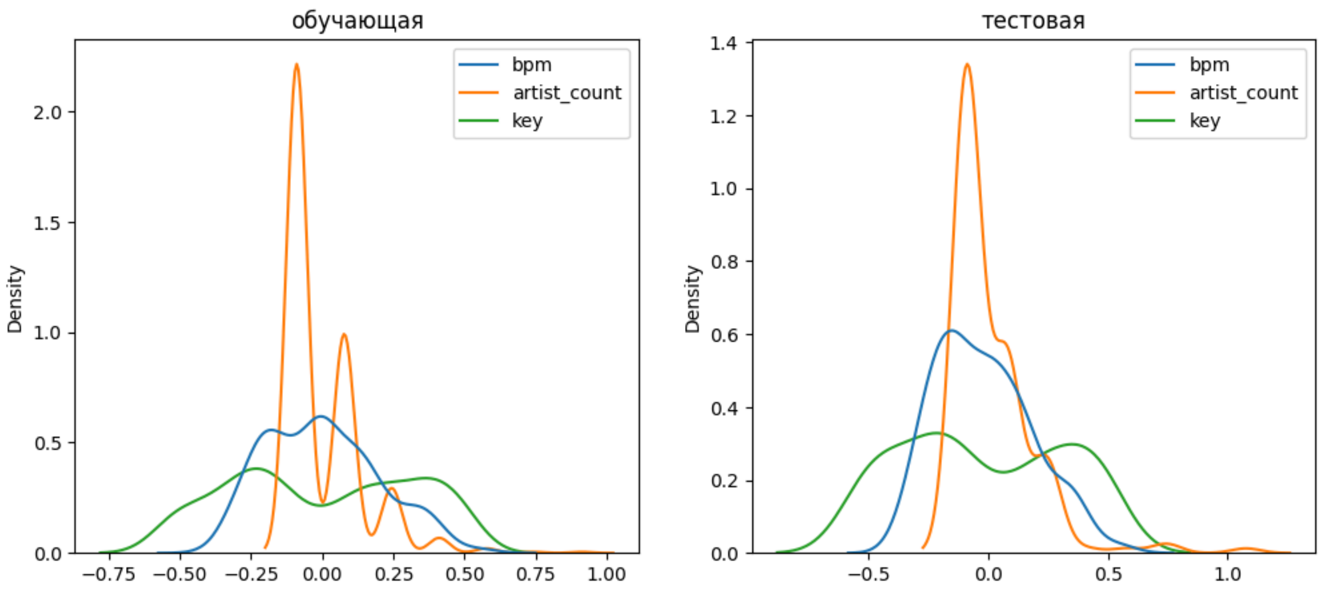
```
data_cs22_scaled_test.describe()
```

	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	en
count	172.000000	172.000000	172.000000	172.000000	172.000000	172.000000	172.000000	172.000000	172.000000	172.
mean	-0.009303	0.013549	-0.000195	0.036087	0.035222	-0.017809	-0.021278	0.034748	-0.003497	0.
std	0.282480	0.171316	0.118014	0.312357	0.304249	0.200333	0.338637	0.494771	0.213251	0.
min	-0.500979	-0.089164	-0.658547	-0.449642	-0.417104	-0.371141	-0.519534	-0.546647	-0.593255	-0.
25%	-0.262450	-0.089164	0.018872	-0.176915	-0.225437	-0.179652	-0.319534	-0.546647	-0.154898	-0.
50%	-0.005097	-0.089164	0.040377	0.004903	0.016229	-0.037808	-0.119534	0.453353	0.009485	0.
75%	0.219315	0.077502	0.040377	0.300358	0.282896	0.118221	0.280466	0.453353	0.173869	0.

```
draw_kde(x_col_list, data, data_cs21_scaled, 'до масштабирования', 'после масштабирования')
```




```
draw_kde(x_col_list, data_cs22_scaled_train, data_cs22_scaled_test, 'обучающая', 'тестовая')
```



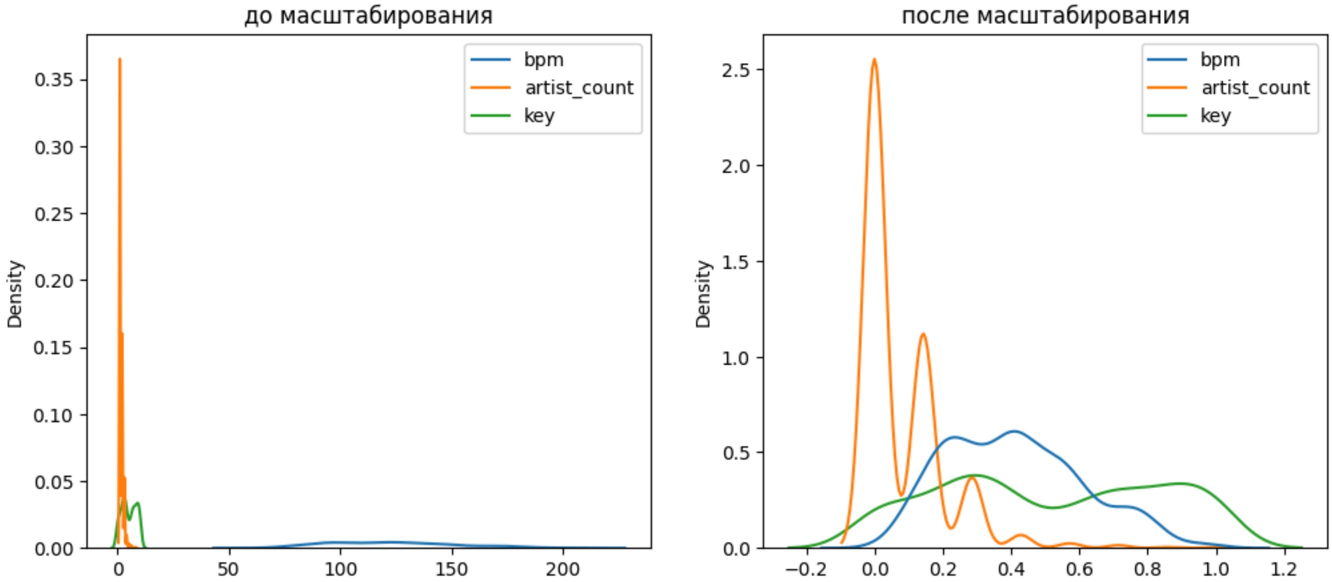
MinMax масштабирование

```
# Обучаем StandardScaler на всей выборке и масштабируем
cs31 = MinMaxScaler()
data_cs31_scaled_temp = cs31.fit_transform(X_ALL)
# формируем DataFrame на основе массива
data_cs31_scaled = arr_to_df(data_cs31_scaled_temp)
data_cs31_scaled.describe()
```

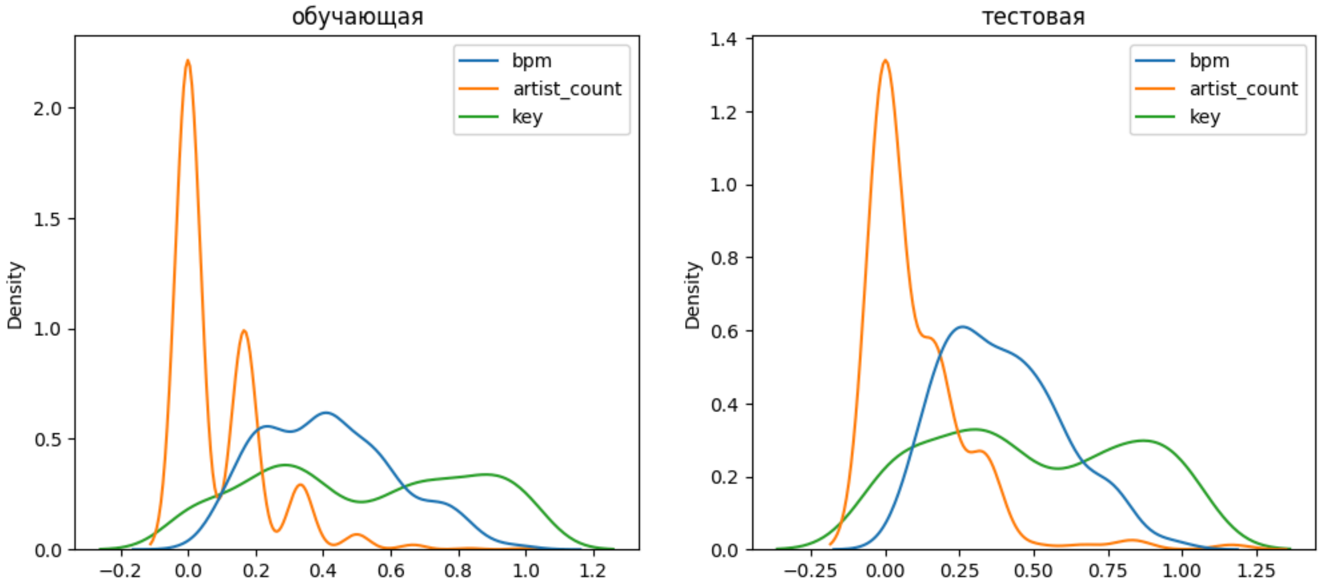
	track_name	artist_count	released_year	released_month	released_day	bpm	key	mode	danceability_%	en
count	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.
mean	0.499703	0.078755	0.948831	0.456876	0.424165	0.410124	0.515268	0.553613	0.606252	0.
std	0.289109	0.123476	0.119439	0.324472	0.309747	0.199883	0.323036	0.497407	0.200722	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.249412	0.000000	0.967742	0.181818	0.133333	0.248227	0.300000	0.000000	0.465753	0.
50%	0.500000	0.000000	0.989247	0.363636	0.400000	0.397163	0.500000	1.000000	0.643836	0.
75%	0.749412	0.142857	0.989247	0.727273	0.700000	0.544326	0.800000	1.000000	0.753425	0.

```
cs32 = MinMaxScaler()
cs32.fit(X_train)
data_cs32_scaled_train_temp = cs32.transform(X_train)
data_cs32_scaled_test_temp = cs32.transform(X_test)
# формируем DataFrame на основе массива
data_cs32_scaled_train = arr_to_df(data_cs32_scaled_train_temp)
data_cs32_scaled_test = arr_to_df(data_cs32_scaled_test_temp)
```

```
draw_kde(x_col_list, data, data_cs31_scaled, 'до масштабирования', 'после масштабирования')
```




```
draw_kde(x_col_list, data_cs32_scaled_train, data_cs32_scaled_test, 'обучающая', 'тестовая')
```



Обработка выбросов

```
def diagnostic_plots(df, variable, title):
    fig, ax = plt.subplots(figsize=(10,7))
    # гистограмма
    plt.subplot(2, 2, 1)
    df[variable].hist(bins=30)
    ## Q-Q plot
    plt.subplot(2, 2, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    # ящик с усами
    plt.subplot(2, 2, 3)
    sns.violinplot(x=df[variable])
    # ящик с усами
    plt.subplot(2, 2, 4)
    sns.boxplot(x=df[variable])
    fig.suptitle(title)
    plt.show()
```

```
from enum import Enum
class OutlierBoundaryType(Enum):
    SIGMA = 1
    QUANTILE = 2
    IRQ = 3
```

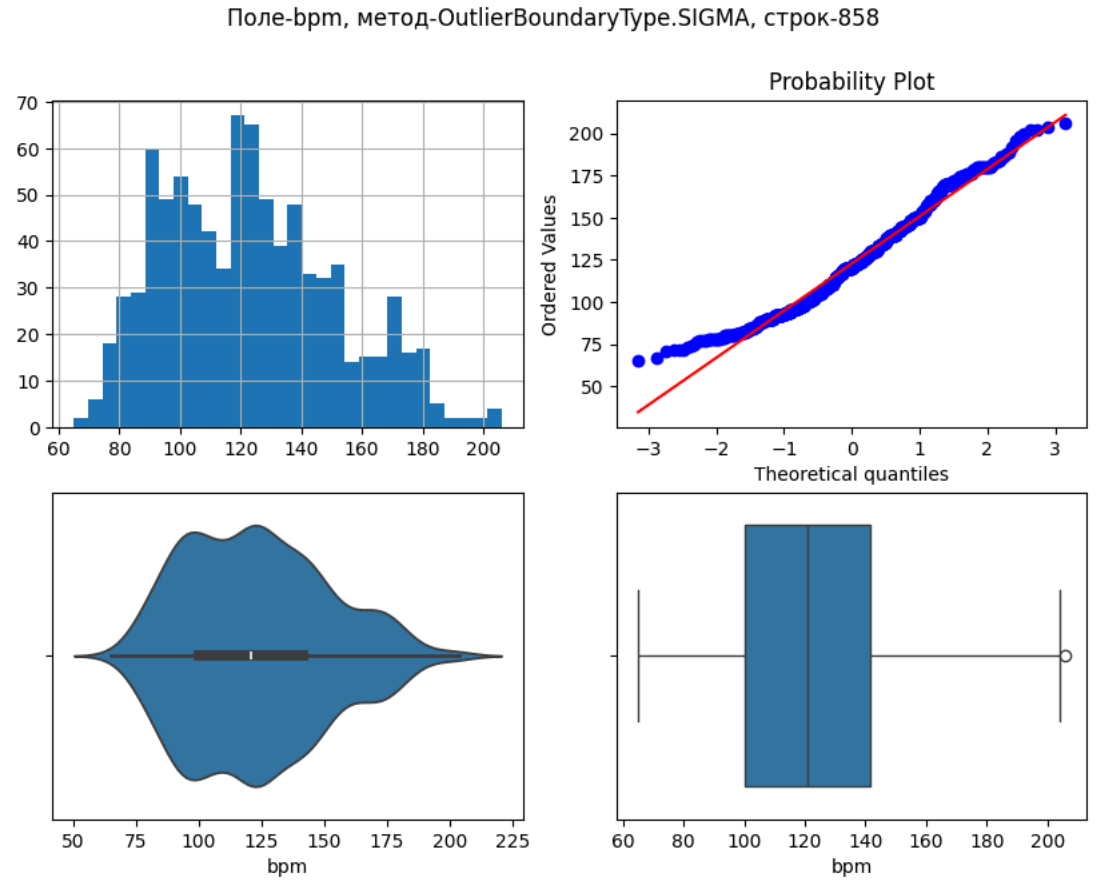
```python
# Функция вычисления верхней и нижней границы выбросов
def get_outlier_boundaries(df, col, outlier_boundary_type: OutlierBoundaryType):
    if outlier_boundary_type == OutlierBoundaryType.SIGMA:
        K1 = 3
        lower_boundary = df[col].mean() - (K1 * df[col].std())
        upper_boundary = df[col].mean() + (K1 * df[col].std())

    elif outlier_boundary_type == OutlierBoundaryType.QUANTILE:
        lower_boundary = df[col].quantile(0.05)
        upper_boundary = df[col].quantile(0.95)

    elif outlier_boundary_type == OutlierBoundaryType.IRQ:
        K2 = 1.5
        IQR = df[col].quantile(0.75) - df[col].quantile(0.25)
        lower_boundary = df[col].quantile(0.25) - (K2 * IQR)
        upper_boundary = df[col].quantile(0.75) + (K2 * IQR)

    else:
        raise NameError('Unknown Outlier Boundary Type')

    return lower_boundary, upper_boundary
```

Удаление выбросов

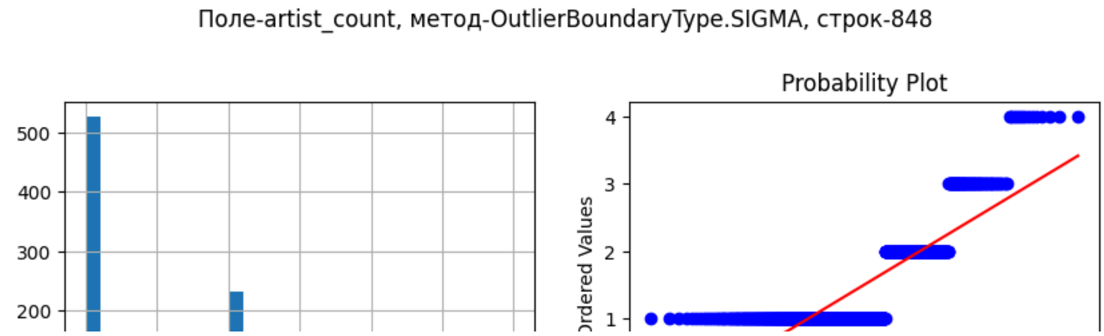
Воспользуемся методом OutlierBoundaryType.SIGMA:

```python
for col in x_col_list:
        # Вычисление верхней и нижней границы
        lower_boundary, upper_boundary = get_outlier_boundaries(data, col, OutlierBoundaryType.SIGMA)
        # Флаги для удаления выбросов
        outliers_temp = np.where(data[col] > upper_boundary, True,
                                 np.where(data[col] < lower_boundary, True, False))
        # Удаление данных на основе флага
        data_trimmed = data.loc[~(outliers_temp), ]
        title = 'Поле-{}, метод-{}, строк-{}'.format(col, OutlierBoundaryType.SIGMA, data_trimmed.shape[0])
        diagnostic_plots(data_trimmed, col, title)
```

```
<ipython-input-60-766c933c159f>:4: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and w
  plt.subplot(2, 2, 1)
```




```
<ipython-input-60-766c933c159f>:4: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and w
  plt.subplot(2, 2, 1)
```



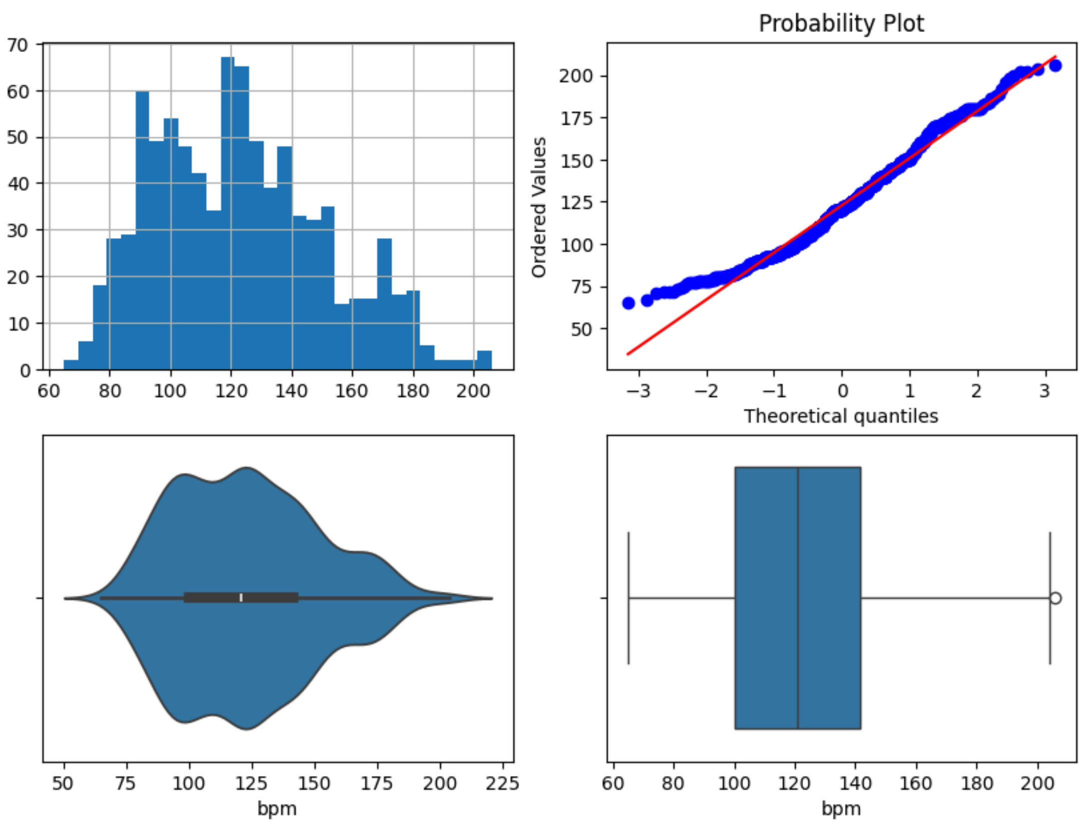
Замена выбросов

Проведём замену выбросов с помощью метода OutlierBoundaryType.SIGMA:

```
for col in x_col_list:
        # Вычисление верхней и нижней границы
        lower_boundary, upper_boundary = get_outlier_boundaries(data, col, OutlierBoundaryType.SIGMA)
        # Изменение данных
        data[col] = np.where(data[col] > upper_boundary, upper_boundary,
                                 np.where(data[col] < lower_boundary, lower_boundary, data[col]))
        title = 'Поле-{}, метод-{}'.format(col, OutlierBoundaryType.SIGMA)
        diagnostic_plots(data, col, title)
```

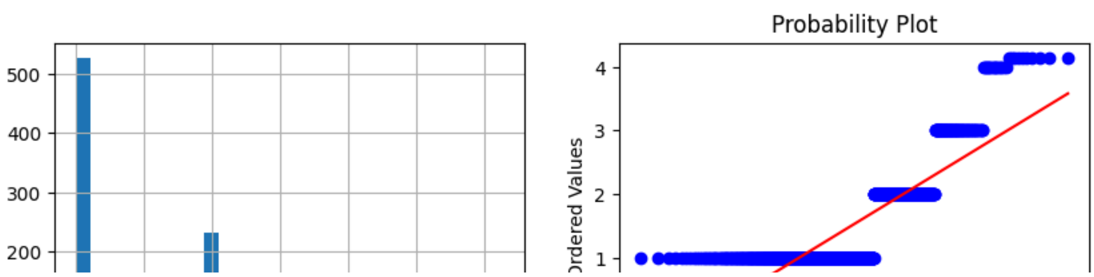
```
<ipython-input-60-766c933c159f>:4: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and w
  plt.subplot(2, 2, 1)
```

Поле-bpm, метод-OutlierBoundaryType.SIGMA




```
<ipython-input-60-766c933c159f>:4: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and w
  plt.subplot(2, 2, 1)
```

Поле-artist_count, метод-OutlierBoundaryType.SIGMA



Отбор признаков

Метод фильтрации (filter)

Воспользуемся методом "Удаление константных и псевдоконстантных (почти константных) признаков".

Известно, что в данном датасете `artist_count` и `mode` - константные признаки.

```
data['artist_count'].unique()
```

```
array([2.       , 1.       , 3.       , 4.1442877, 4.       ])
```

```
data['mode'].unique()
```

```
array([1, 0])
```

С помощью VarianceThreshold попробуем обнаружить больше таких признаков:

```
from sklearn.feature_selection import VarianceThreshold
```

```
selector = VarianceThreshold(threshold=0.15)
selector.fit(data)
# Значения дисперсий для каждого признака
selector.variances_
```

```
array([6.04610695e+04, 6.02616343e-01, 1.23238997e+02, 1.27242843e+01,
       8.62483523e+01, 7.93385164e+02, 1.04230756e+01, 2.47125640e-01,
       2.14451736e+02, 5.56064823e+02, 2.57230633e+02, 6.58790086e+02,
       7.32892793e+01, 1.83702943e+02, 1.01652399e+02])
```

Удалим константные и псевдоконстантные признаки:

```
selector.transform(data)
```

```
array([[6.180e+02, 2.000e+00, 2.023e+03, ..., 0.000e+00, 8.000e+00,
        4.000e+00],
       [3.570e+02, 1.000e+00, 2.023e+03, ..., 0.000e+00, 1.000e+01,
        4.000e+00],
       [8.450e+02, 1.000e+00, 2.023e+03, ..., 0.000e+00, 3.100e+01,
        6.000e+00],
       ...,
       [1.100e+01, 2.000e+00, 2.022e+03, ..., 0.000e+00, 8.000e+00,
        6.000e+00],
       [2.130e+02, 3.000e+00, 2.022e+03, ..., 0.000e+00, 1.200e+01,
        5.000e+00],
       [3.900e+01, 1.000e+00, 2.022e+03, ..., 0.000e+00, 1.100e+01,
        5.000e+00]])
```

Метод обертывания (wrapper)