МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

ОТЧЕТ

Лабораторная работа № 3
по дисциплине «Разработка нейросетевых систем»

Тема: «Обработка признаков, часть 2»

ИСПОЛНИТЕЛЬ: ___Кравцов А.Н.___
ФИО
группа ИУ5-24М _____
подпись
"26" апреля 2024 г.

ПРЕПОДАВАТЕЛЬ: ____Гапанюк Ю.Е.____
ФИО
_____
подпись
"__"_____2024 г.

Москва – 2024

Задание

1. 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.
2. Для выбранного датасета (датасетов) на основе материалов лекций решить следующие задачи: i. масштабирование признаков (не менее чем тремя способами); ii. обработку выбросов для числовых признаков (по одному способу для удаления выбросов и для замены выбросов); iii. обработку по крайней мере одного нестандартного признака (который не является числовым или категориальным); iv. 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).

Загрузка датасета

```
data = pd.read_csv("winequalityN.csv")
data
```

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	white	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	white	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	red	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
6493	red	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	NaN	11.2	6
6494	red	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
6495	red	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
6496	red	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

6497 rows × 13 columns

Устранение пропусков в данных

```
hdata = hdata_loaded
list(zip(hdata.columns, [i for i in hdata.dtypes]))
```

```
[('show_id', dtype('O')),
 ('type', dtype('O')),
 ('title', dtype('O')),
 ('director', dtype('O')),
 ('cast', dtype('O')),
 ('country', dtype('O')),
 ('date_added', dtype('O')),
 ('release_year', dtype('int64')),
 ('rating', dtype('O')),
 ('duration', dtype('O')),
 ('listed_in', dtype('O')),
 ('description', dtype('O'))]
```

```
# cols with missing values
hcols_with_na = [c for c in hdata.columns if hdata[c].isnull().sum() > 0]
hcols_with_na
```

```
['director', 'cast', 'country', 'date_added', 'rating', 'duration']
```

```
# Колонки с пропусками
hcols_with_na = [c for c in data.columns if data[c].isnull().sum() > 0]
hcols_with_na
# Количество пропусков
[(c, data[c].isnull().sum()) for c in hcols_with_na]
```

4]

```
[('fixed acidity', 10),
 ('volatile acidity', 8),
 ('citric acid', 3),
 ('residual sugar', 2),
 ('chlorides', 2),
 ('pH', 9),
 ('sulphates', 4)]
```

```
# Доля (процент) пропусков
[(c, data[c].isnull().mean()) for c in hcols_with_na]
```

5]

```
[('fixed acidity', 0.0015391719255040787),
 ('volatile acidity', 0.001231337540403263),
 ('citric acid', 0.00046175157765122367),
 ('residual sugar', 0.00030783438510081576),
 ('chlorides', 0.00030783438510081576),
 ('pH', 0.0013852547329536709),
 ('sulphates', 0.0006156687702016315)]
```

```
data = data.dropna(axis=0, how='any')
data
```

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	0	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	0	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	0	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	0	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	0	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6491	1	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	6
6492	1	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
6494	1	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
6495	1	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
6496	1	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

6463 rows × 13 columns

```python
data.reset_index(inplace=True)
# Колонки с пропусками
hcols_with_na = [c for c in data.columns if data[c].isnull().sum() > 0]
hcols_with_na
# Количество пропусков
[(c, data[c].isnull().sum()) for c in hcols_with_na]
# Доля (процент) пропусков
[(c, data[c].isnull().mean()) for c in hcols_with_na]
```

```
[]
```

```python
data
```

	index	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	0	0	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	1	0	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	2	0	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	3	0	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	4	0	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6458	6491	1	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	6
6459	6492	1	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
6460	6494	1	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
6461	6495	1	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
6462	6496	1	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

6463 rows × 14 columns

Масштабирование признаков

```python
# Нужно ли масштабирование
data.describe()
```

Python

	index	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000
mean	3255.125793	7.217755	0.339589	0.318758	5.443958	0.056056	30.516865	115.694492	0.994698	3.218332	0.531150	10.492825	5.818505
std	1869.906597	1.297913	0.164639	0.145252	4.756852	0.035076	17.758815	56.526736	0.003001	0.160650	0.148913	1.193128	0.873286
min	0.000000	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	1639.500000	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992330	3.110000	0.430000	9.500000	5.000000
50%	3257.000000	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.000000
75%	4872.500000	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.997000	3.320000	0.600000	11.300000	6.000000
max	6496.000000	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.000000

```python
# DataFrame не содержащий целевой признак
X_ALL = data.drop('quality', axis=1)
```

Python

```python
# 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
```

Python

```python
# Разделим выборку на обучающую и тестовую
X_train, X_test, y_train, y_test = train_test_split(X_ALL, data['quality'],
                                                    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
```

Python

```
((5170, 13), (1293, 13))
```

Масштабирование данных на основе Z-оценки

```
# 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
```

	index	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	-1.740931	-0.571931	-0.167786	-0.422710	0.283959	3.207420	-0.315228	0.815609	0.960779	2.099926	-1.359160	-0.544987	-1.418922
1	-1.740396	-0.571931	-0.707155	-0.240479	0.146257	-0.808151	-0.201180	-0.930138	0.288479	-0.232465	0.508399	-0.276354	-0.832184
2	-1.739861	-0.571931	0.679794	-0.361966	0.559363	0.306117	-0.172668	-0.029107	-0.330745	0.134053	0.259391	-0.612146	-0.329265
3	-1.739326	-0.571931	-0.013681	-0.665684	0.008554	0.642500	0.055427	0.928238	1.243853	0.300653	-0.176373	-0.880779	-0.496905
4	-1.738791	-0.571931	-0.013681	-0.665684	0.008554	0.642500	0.055427	0.928238	1.243853	0.300653	-0.176373	-0.880779	-0.496905
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6458	1.730634	1.748464	-0.321892	1.703315	-1.643871	-0.745079	0.340545	-0.141736	-1.374580	0.603864	1.255423	1.939871	-0.832184
6459	1.731169	1.748464	-0.784208	1.581828	-1.643871	-0.724055	0.967807	0.083522	-1.268427	0.067414	1.442179	0.328071	0.006014
6460	1.732239	1.748464	-0.707155	1.035136	-1.299616	-0.660984	0.568640	-0.085421	-1.339195	0.347301	1.255423	1.469762	0.425113
6461	1.732774	1.748464	-1.015366	1.855174	-1.368467	-0.724055	0.540129	0.083522	-1.268427	0.257337	2.189203	1.201129	-0.245446
6462	1.733308	1.748464	-0.938313	-0.179735	1.041320	-0.387673	0.312034	-0.704880	-1.303811	0.264001	1.068667	0.865337	0.425113

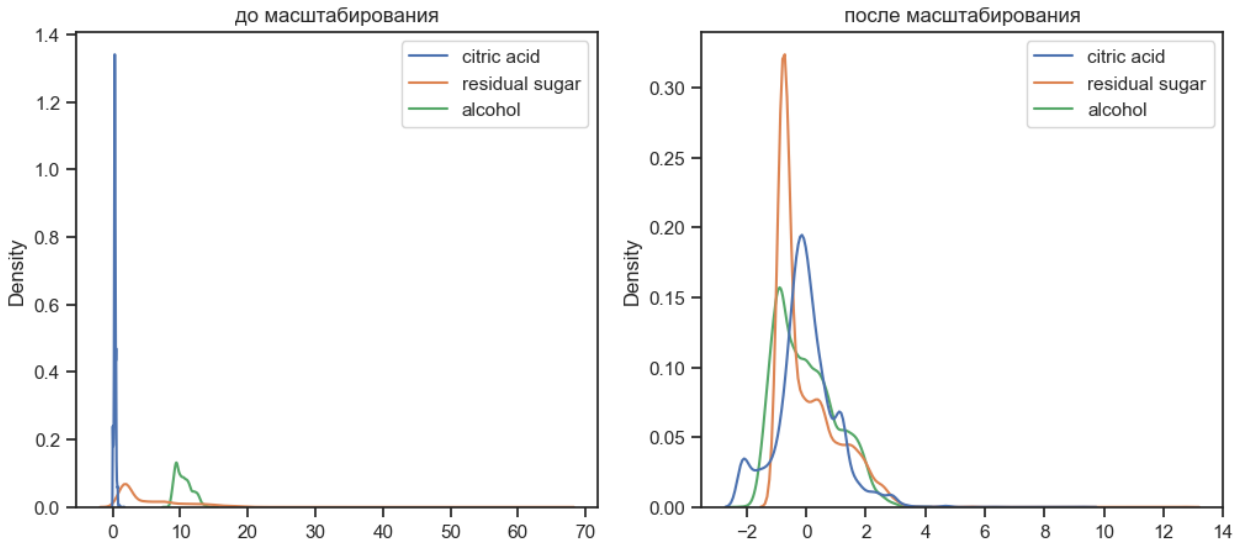
6463 rows × 13 columns

```
data_cs11_scaled.describe()
```

	index	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulph
count	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e+03	6.463000e
mean	1.407233e-16	-3.518083e-17	-1.231329e-16	-2.110850e-16	-8.795206e-17	1.934945e-16	-7.036165e-17	-7.036165e-17	1.055425e-16	5.219955e-14	9.850631e-16	1.407233
std	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e+00	1.000077e
min	-1.740931e+00	-5.719307e-01	-2.633473e+00	-1.576837e+00	-2.194680e+00	-1.018391e+00	-1.341655e+00	-1.662225e+00	-1.940727e+00	-2.528205e+00	-3.102215e+00	-2.089628e
25%	-8.640810e-01	-5.719307e-01	-6.301024e-01	-6.656838e-01	-4.734030e-01	-7.661033e-01	-5.148107e-01	-7.611944e-01	-6.845873e-01	-7.889073e-01	-6.743884e-01	-6.793038
50%	1.002377e-03	-5.719307e-01	-1.677861e-01	-3.012225e-01	-6.029661e-02	-5.138161e-01	-2.582038e-01	-8.542138e-02	4.078931e-02	6.408161e-02	-5.186862e-02	-1.420373
75%	8.650161e-01	-5.719307e-01	3.715830e-01	3.669567e-01	4.905119e-01	5.584045e-01	2.550099e-01	5.903516e-01	7.130896e-01	7.671311e-01	6.329031e-01	4.623876
max	1.733308e+00	1.748464e+00	6.689906e+00	7.534697e+00	9.234597e+00	1.268921e+01	1.582249e+01	1.455633e+01	5.737650e+00	1.475482e+01	4.928289e+00	9.864552e

```
# Построение плотности 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(['citric acid', 'residual sugar', 'alcohol'], data, data_cs11_scaled, 'до масштабирования', 'после масштабирования')
```



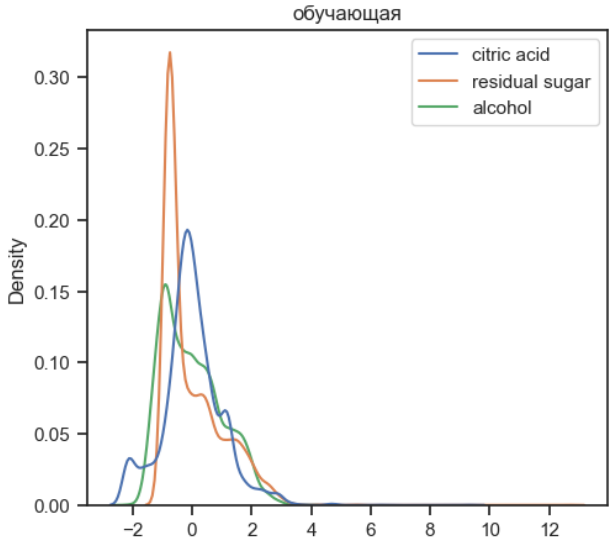
```python
# Обучаем StandardScaler на обучающей выборке
# и масштабируем обучающую и тестовую выборки
cs12 = StandardScaler()
cs12.fit(X_train)
data_cs12_scaled_train_temp = cs12.transform(X_train)
data_cs12_scaled_test_temp = cs12.transform(X_test)
# формируем DataFrame на основе массива
data_cs12_scaled_train = arr_to_df(data_cs12_scaled_train_temp)
data_cs12_scaled_test = arr_to_df(data_cs12_scaled_test_temp)
```

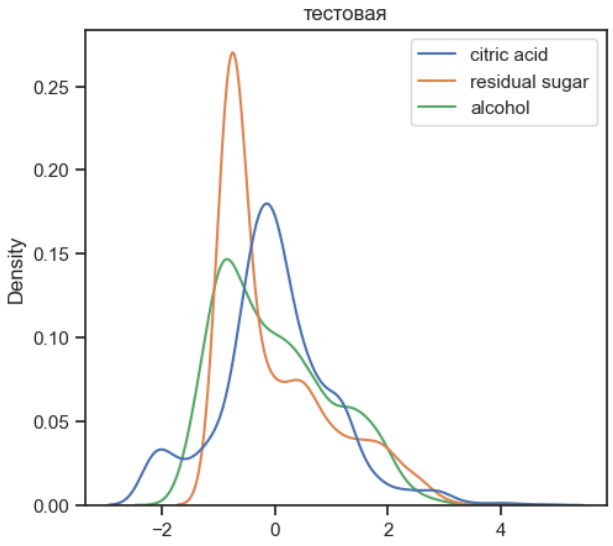
[17] Pyth

```python
data_cs12_scaled_train.describe()
```

[18] Pyth

	index	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sul
count	5.170000e+03	5.170000e+03	5.170000e+03	5.170000e+03	5.170000e+03	5.170000e+03	5.170000e+03	5.170000e+03	5.170000e+03	5.170000e+03	5.170000e+03	5.17000
mean	-9.757937e-17	4.260508e-17	4.641892e-16	-3.037330e-16	4.473533e-16	9.345630e-17	-8.108708e-17	1.003281e-16	3.229740e-17	9.510553e-16	-3.856447e-15	4.0268
std	1.000097e+00	1.000097e+00	1.000097e+00	1.000097e+00	1.000097e+00	1.000097e+00	1.000097e+00	1.000097e+00	1.000097e+00	1.000097e+00	1.000097e+00	1.00009
min	-1.729700e+00	-5.709475e-01	-2.622586e+00	-1.577543e+00	-2.204527e+00	-1.016795e+00	-1.322909e+00	-1.667163e+00	-1.952567e+00	-2.523179e+00	-3.099985e+00	-2.08709
25%	-8.711414e-01	-5.709475e-01	-6.234025e-01	-6.625140e-01	-4.749745e-01	-7.654524e-01	-5.060611e-01	-7.626958e-01	-6.737337e-01	-7.810652e-01	-6.762115e-01	-6.7846
50%	4.990670e-03	-5.709475e-01	-1.620525e-01	-2.965024e-01	-5.988198e-02	-5.141098e-01	-2.525565e-01	-8.434549e-02	3.672900e-02	6.339461e-02	-5.473114e-02	-1.4184
75%	8.619519e-01	-5.709475e-01	3.761891e-01	3.745188e-01	4.935747e-01	5.540961e-01	2.262853e-01	5.940048e-01	7.116686e-01	7.615700e-01	6.288973e-01	4.6184
max	1.729564e+00	1.751475e+00	6.681305e+00	7.572747e+00	9.279700e+00	1.263948e+01	1.563373e+01	1.461324e+01	5.755954e+00	1.472175e+01	4.917112e+00	9.85267





Масштабирование "Mean Normalisation"

```python
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)
```

[21] Python

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

[22] Python

	index	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	6463.00000	6463.000000	6463.000000	6463.00000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000
unique	6463.00000	2.000000	106.000000	187.00000	89.000000	315.000000	214.000000	135.000000	276.000000	996.000000	108.000000	111.000000	111.000000
top	-0.50002	-0.245841	-0.033947	-0.03907	-0.011238	-0.052984	-0.019878	-0.005181	-0.011364	0.048211	-0.045586	-0.017498	-0.143654
freq	1.00000	4870.000000	352.000000	285.00000	337.000000	235.000000	203.000000	180.000000	71.000000	69.000000	200.000000	271.000000	366.000000

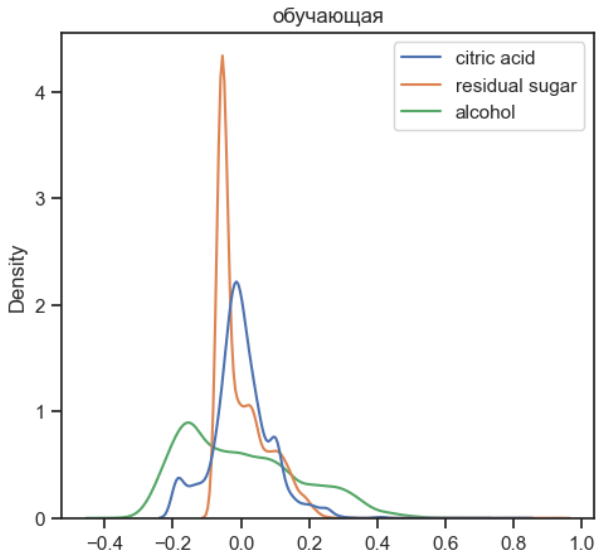
```python
cs22 = MeanNormalisation()
cs22.fit(X_train)
data_cs22_scaled_train = cs22.transform(X_train)
data_cs22_scaled_test = cs22.transform(X_test)
```

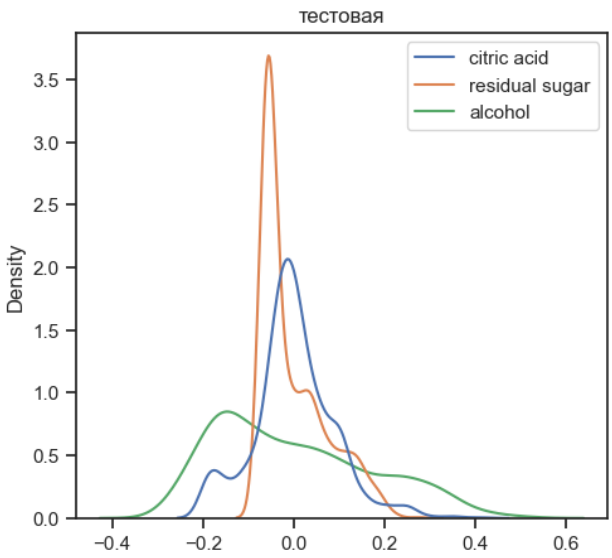
Python

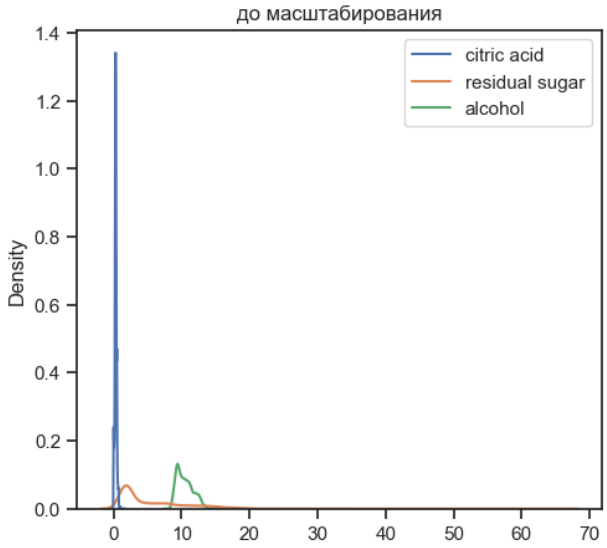
```python
data_cs22_scaled_train.describe()
```

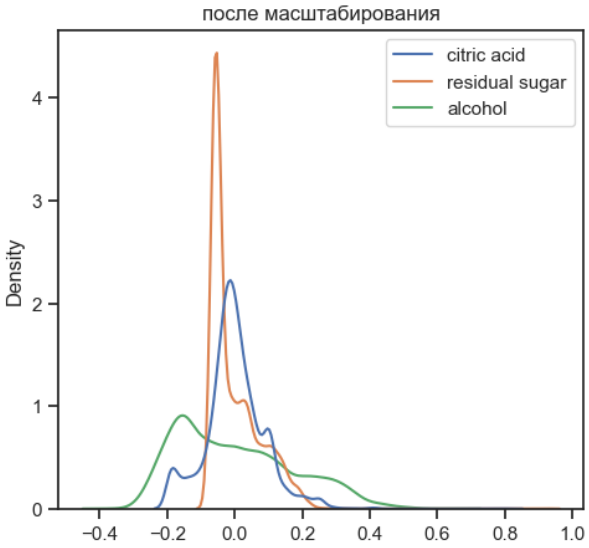
Python

	index	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	5170.000000	5170.000000	5170.000000	5170.000000	5170.000000	5170.000000	5170.000000	5170.000000	5170.000000	5170.000000	5170.00000	5170.000000	5170.000000
unique	5170.000000	2.000000	105.000000	182.000000	87.000000	301.000000	201.000000	126.000000	269.000000	944.000000	108.00000	108.000000	105.000000
top	-0.223696	-0.245841	-0.033947	-0.065737	-0.011238	-0.052984	-0.019878	-0.005181	-0.011364	0.055922	-0.06109	-0.017498	-0.143654
freq	1.000000	3899.000000	293.000000	220.000000	273.000000	190.000000	163.000000	151.000000	55.000000	60.000000	159.00000	221.000000	293.000000





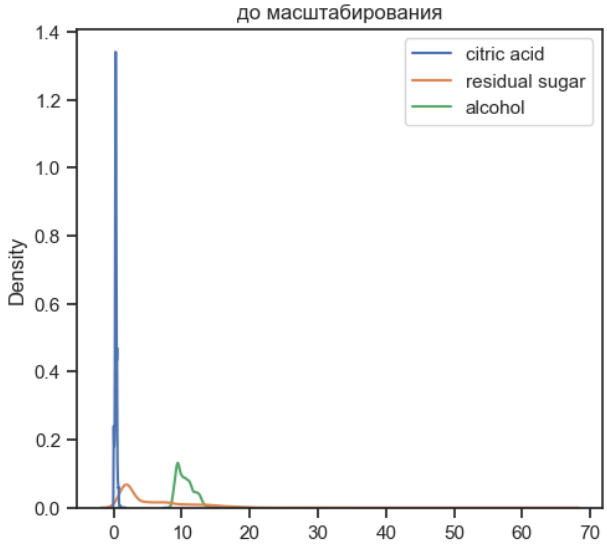


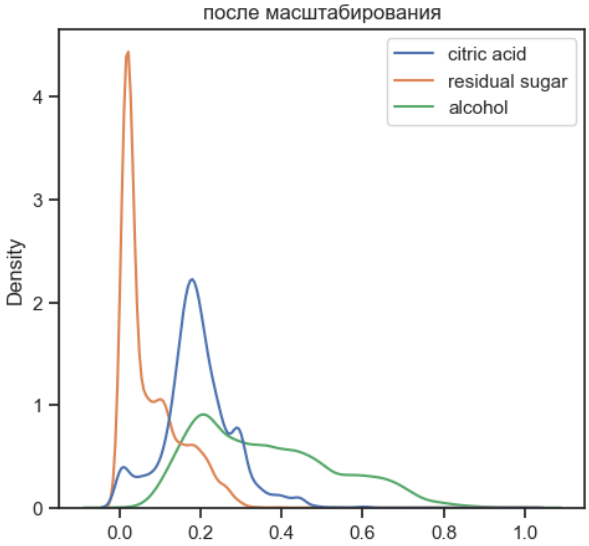


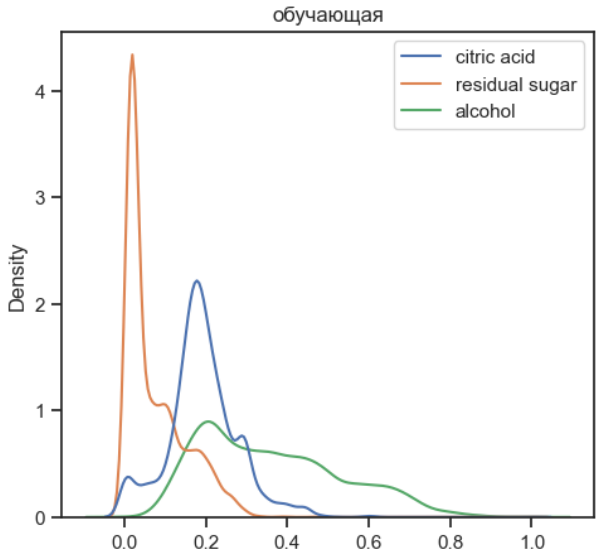
MinMax-масштабирование

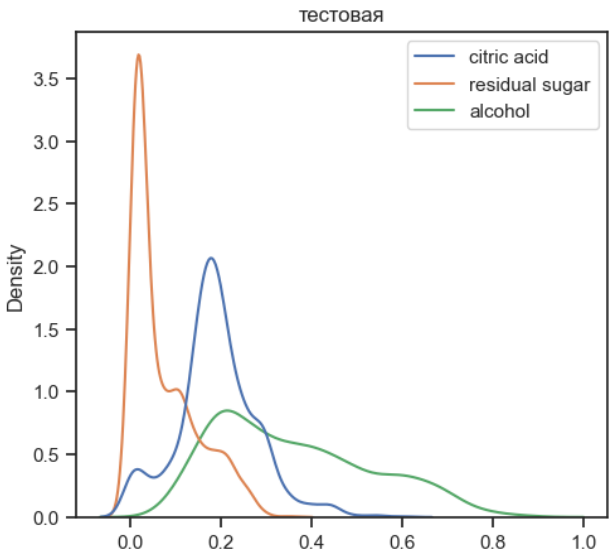
```python
cs32 = MinMaxScaler()
cs32.fit(X_train)
data_cs32_scaled_train_temp = cs32.transform(X_train)
data_cs32_scaled_test_temp = cs32.transform(X_test)
# формируем DataFrame на основе массива
data_cs32_scaled_train = arr_to_df(data_cs32_scaled_train_temp)
data_cs32_scaled_test = arr_to_df(data_cs32_scaled_test_temp)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

Python









Обработка выбросов для числовых признаков

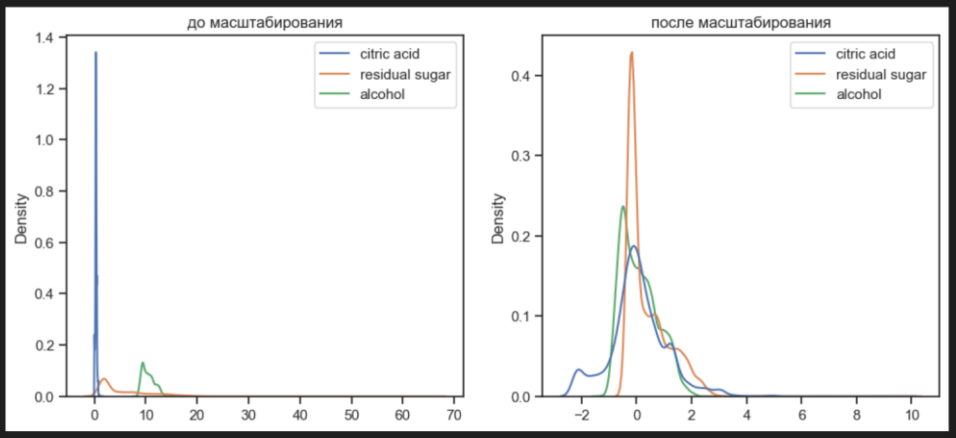
Масштабирование по медиане

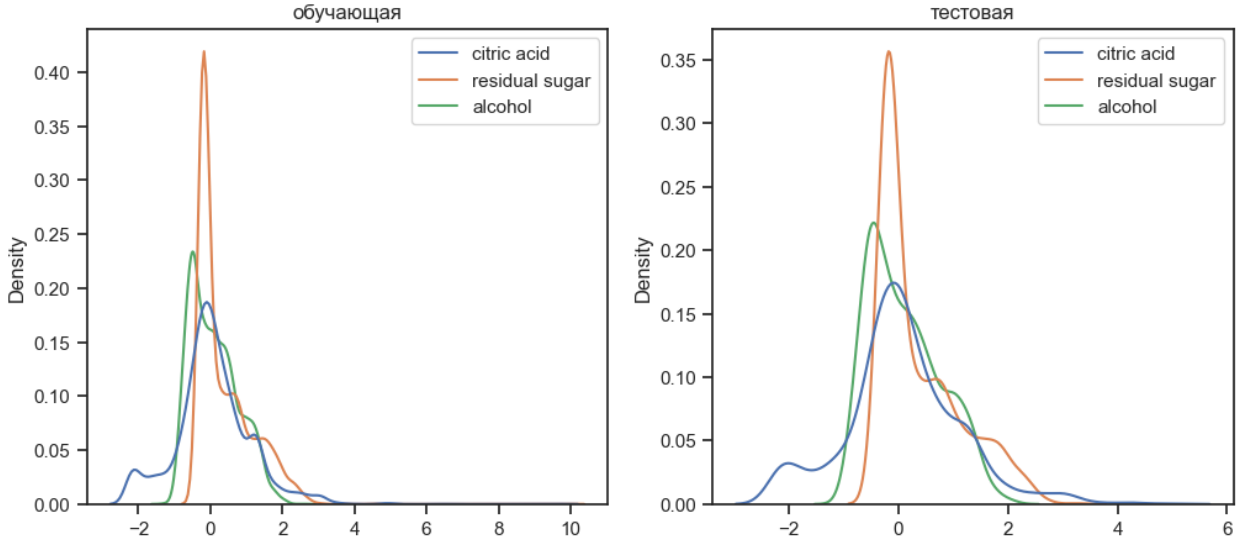
```
cs41 = RobustScaler()
data_cs41_scaled_temp = cs41.fit_transform(X_ALL)
# формируем DataFrame на основе массива
data_cs41_scaled = arr_to_df(data_cs41_scaled_temp)
data_cs41_scaled.describe()
```

	index	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000
mean	-0.000580	0.246480	0.167504	0.291701	0.062554	0.387930	0.335408	0.063203	-0.029184	-0.041183	0.039676	0.124410	0.107125
std	0.578381	0.430995	0.998395	0.968466	1.037516	0.755056	1.299105	0.739951	0.715528	0.642707	0.765000	0.875961	0.662849
min	-1.007423	0.000000	-2.461538	-1.235294	-2.214286	-0.380952	-1.407407	-1.166667	-1.417722	-1.665953	-2.333333	-1.705882	-1.277778
25%	-0.500309	0.000000	-0.461538	-0.352941	-0.428571	-0.190476	-0.333333	-0.500000	-0.518987	-0.548180	-0.476190	-0.470588	-0.444444
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.499691	0.000000	0.538462	0.647059	0.571429	0.809524	0.666667	0.500000	0.481013	0.451820	0.523810	0.529412	0.555556
max	1.001856	1.000000	6.846154	7.588235	9.642857	9.968254	20.888889	10.833333	4.075949	9.441113	3.809524	8.764706	2.555556

```
cs42 = RobustScaler()
cs42.fit(X_train)
data_cs42_scaled_train_temp = cs42.transform(X_train)
data_cs42_scaled_test_temp = cs42.transform(X_test)
# формируем DataFrame на основе массива
data_cs42_scaled_train = arr_to_df(data_cs42_scaled_train_temp)
data_cs42_scaled_test = arr_to_df(data_cs42_scaled_test_temp)
```

```
draw_kde(['citric acid', 'residual sugar', 'alcohol'], data, data_cs41_scaled, 'до масштабирования', 'после масштабирования')
```






```
x_col_list = ['citric acid', 'residual sugar', 'alcohol', 'quality']
x_col_lost = ['type','fixed acidity','volatile acidity','chlorides','free sulfur dioxide','total sulfur dioxide','density',
              'pH','sulphates']
x_col_all = ['type','fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dioxide',
             'total sulfur dioxide','density','pH','sulphates','alcohol','quality']
data2 = data.drop(x_col_lost, axis = 1)
data2
```

	index	citric acid	residual sugar	alcohol	quality
0	0	0.36	20.7	8.8	6
1	1	0.34	1.6	9.5	6
2	2	0.40	6.9	10.1	6
3	3	0.32	8.5	9.9	6
4	4	0.32	8.5	9.9	6
...	...	...	...	...	...
6458	6491	0.08	1.9	9.5	6
6459	6492	0.08	2.0	10.5	5
6460	6494	0.13	2.3	11.0	6
6461	6495	0.12	2.0	10.2	5
6462	6496	0.47	3.6	11.0	6

```
# Колонки с пропусками
hcols_with_na = [c for c in data2.columns if data2[c].isnull().sum() > 0]
hcols_with_na
# Количество пропусков
[(c, data2[c].isnull().sum()) for c in hcols_with_na]
```

```
[]
```

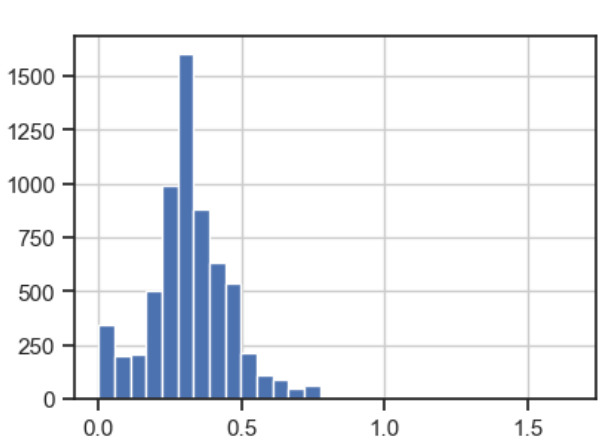
```
# Доля (процент) пропусков
[(c, data2[c].isnull().mean()) for c in hcols_with_na]
```

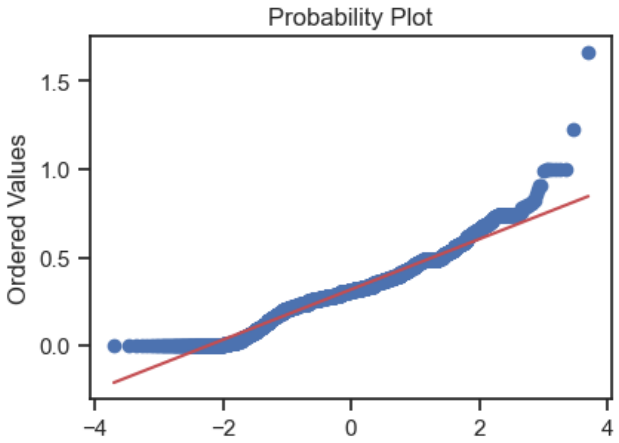
```
[]
```

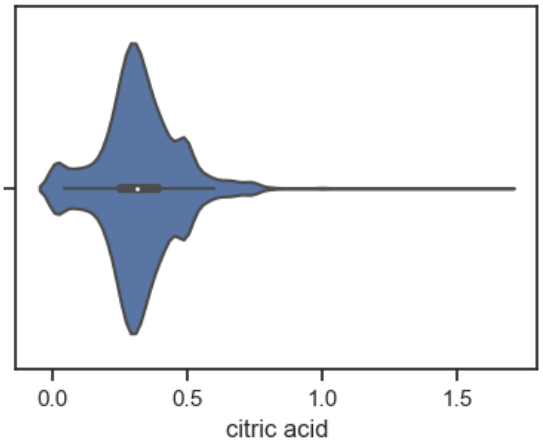
```
data2 = data2.dropna(axis=0, how='any')
data2
```

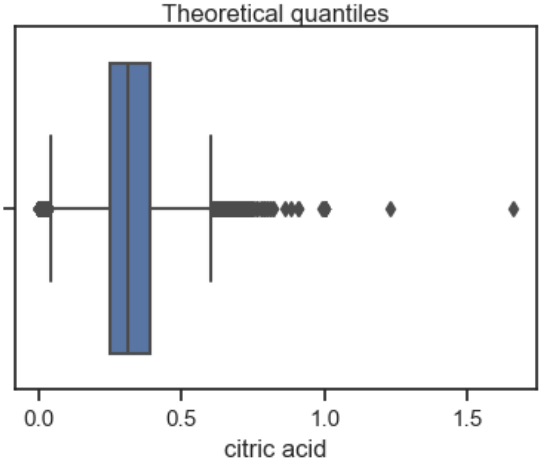
	index	citric acid	residual sugar	alcohol	quality
0	0	0.36	20.7	8.8	6
1	1	0.34	1.6	9.5	6
2	2	0.40	6.9	10.1	6
3	3	0.32	8.5	9.9	6
4	4	0.32	8.5	9.9	6
...	...	...	...	...	...
6458	6491	0.08	1.9	9.5	6
6459	6492	0.08	2.0	10.5	5

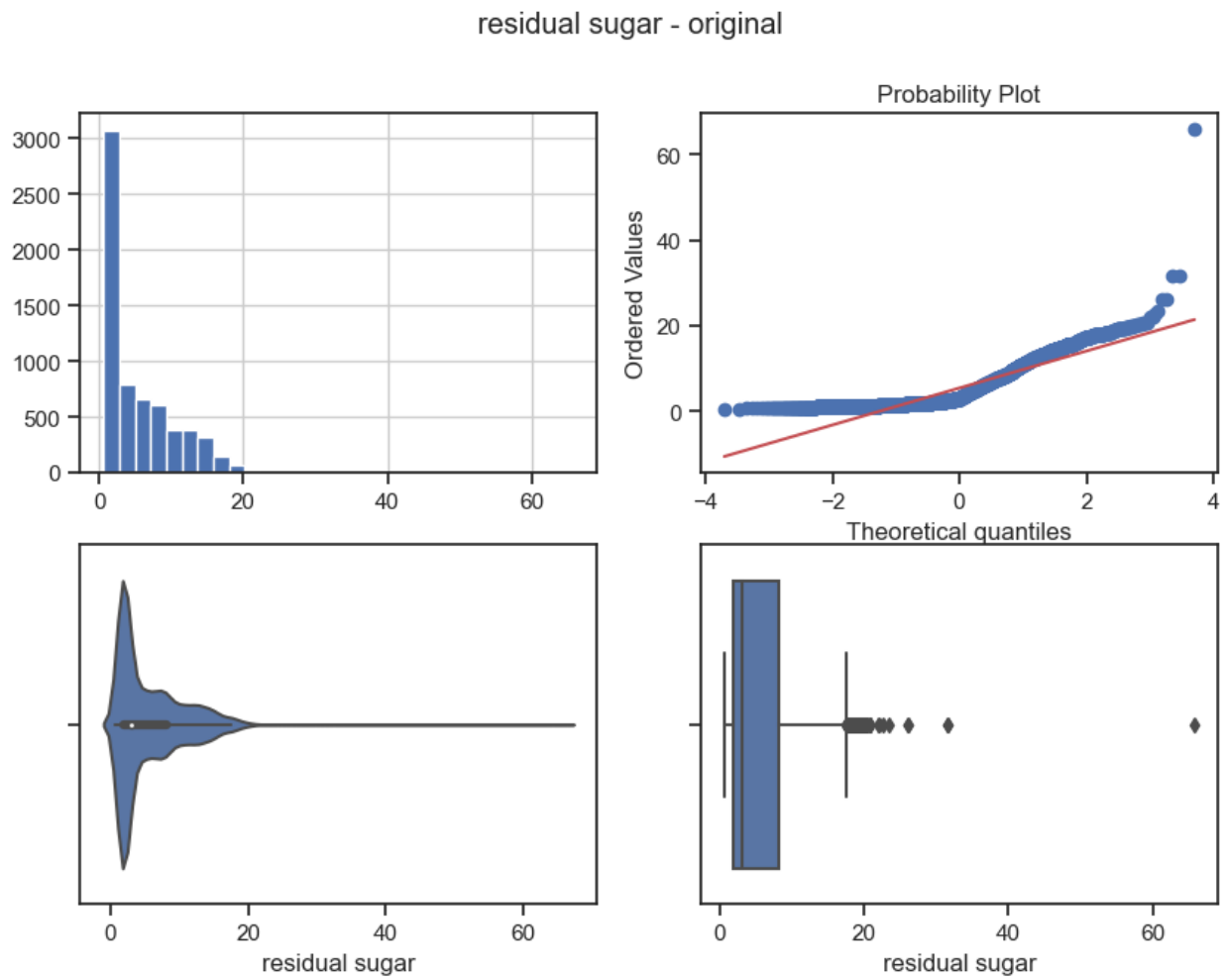
citric acid - original



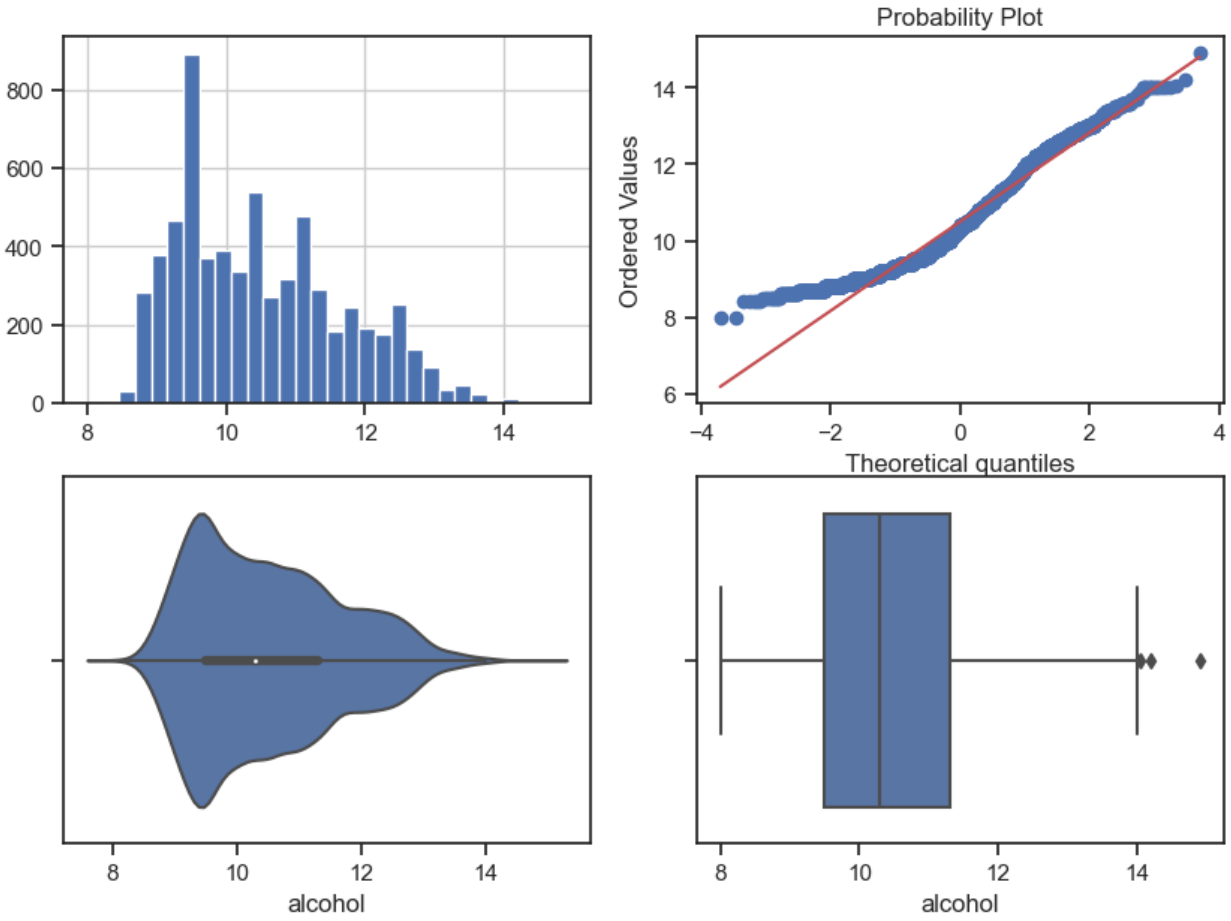







residual sugar - original
Probability Plot
Ordered Values
Theoretical quantiles
residual sugar
residual sugar

alcohol - original

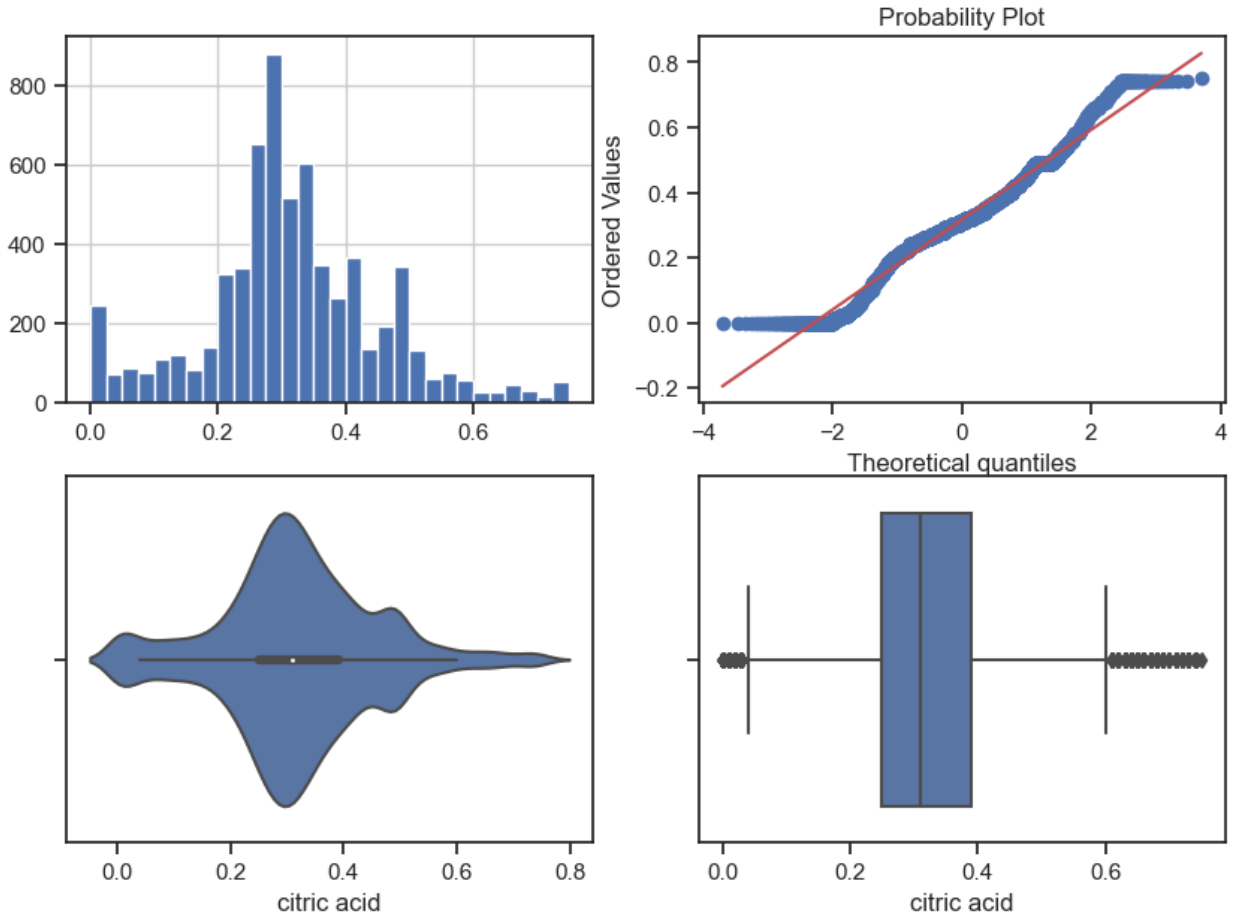


```
# Тип вычисления верхней и нижней границы выбросов
from enum import Enum
class OutlierBoundaryType(Enum):
    SIGMA = 1
    QUANTILE = 2
    IRQ = 3
```

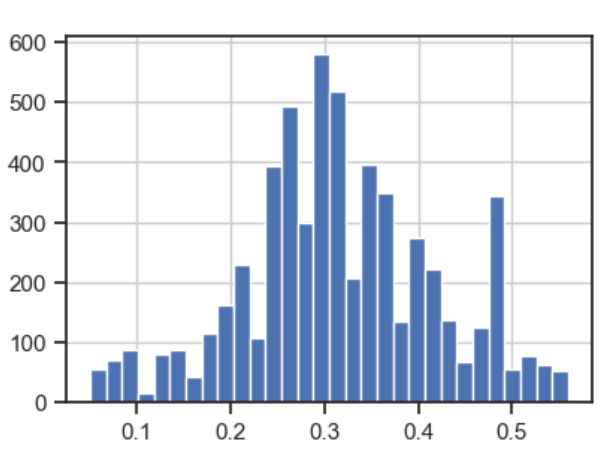
```
# Функция вычисления верхней и нижней границы выбросов
def get_outlier_boundaries(df, col, outlier_boundary_type: OutlierBoundaryType):
    if outlier_boundary_type == OutlierBoundaryType.SIGMA:
        K1 = 3
        lower_boundary = df[col].mean() - (K1 * df[col].std())
        upper_boundary =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

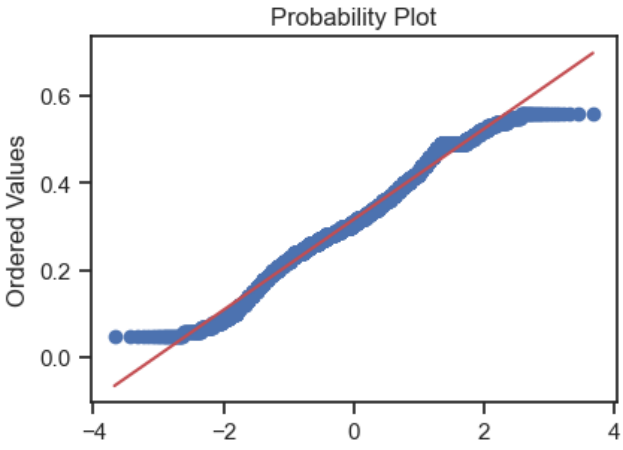
```
for col in x_col_list:
    for obt in OutlierBoundaryType:
        # Вычисление верхней и нижней границы
        lower_boundary, upper_boundary = get_outlier_boundaries(data2, col, obt)
        # Флаги для удаления выбросов
        outliers_temp = np.where(data2[col] > upper_boundary, True,
                                 np.where(data2[col] < lower_boundary, True, False))
        # Удаление данных на основе флага
        data_trimmed = data2.loc[~(outliers_temp), ]
        title = 'Поле-{}, метод-{}, строк-{}'.format(col, obt, data_trimmed.shape[0])
        diagnostic_plots(data_trimmed, col, title)
```

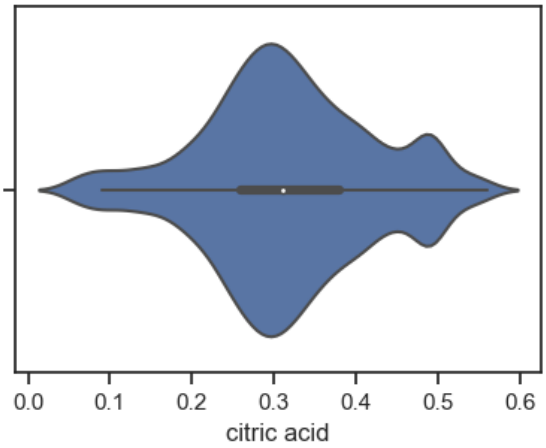
Поле-citric acid, метод-OutlierBoundaryType.SIGMA, строк-6435

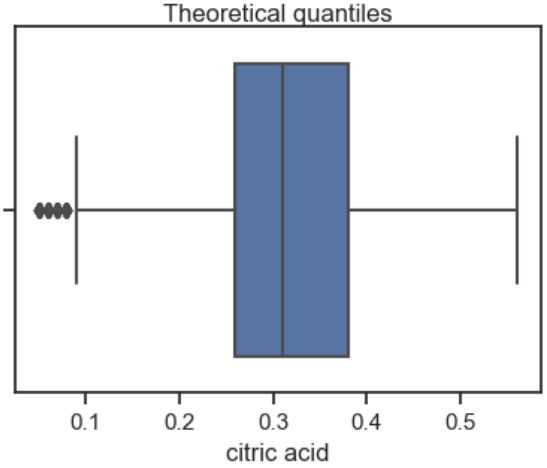


Поле-citric acid, метод-OutlierBoundaryType.QUANTILE, строк-5835

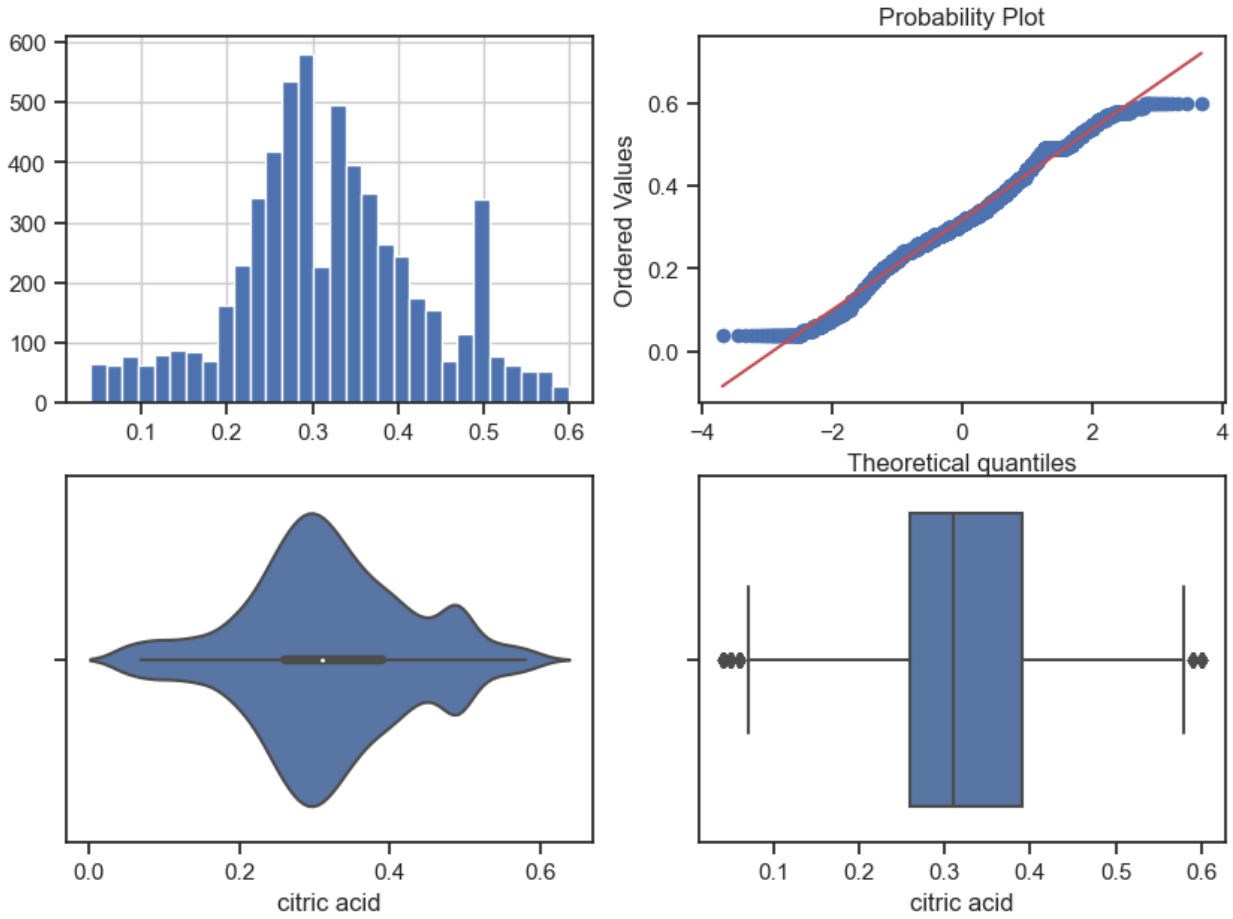




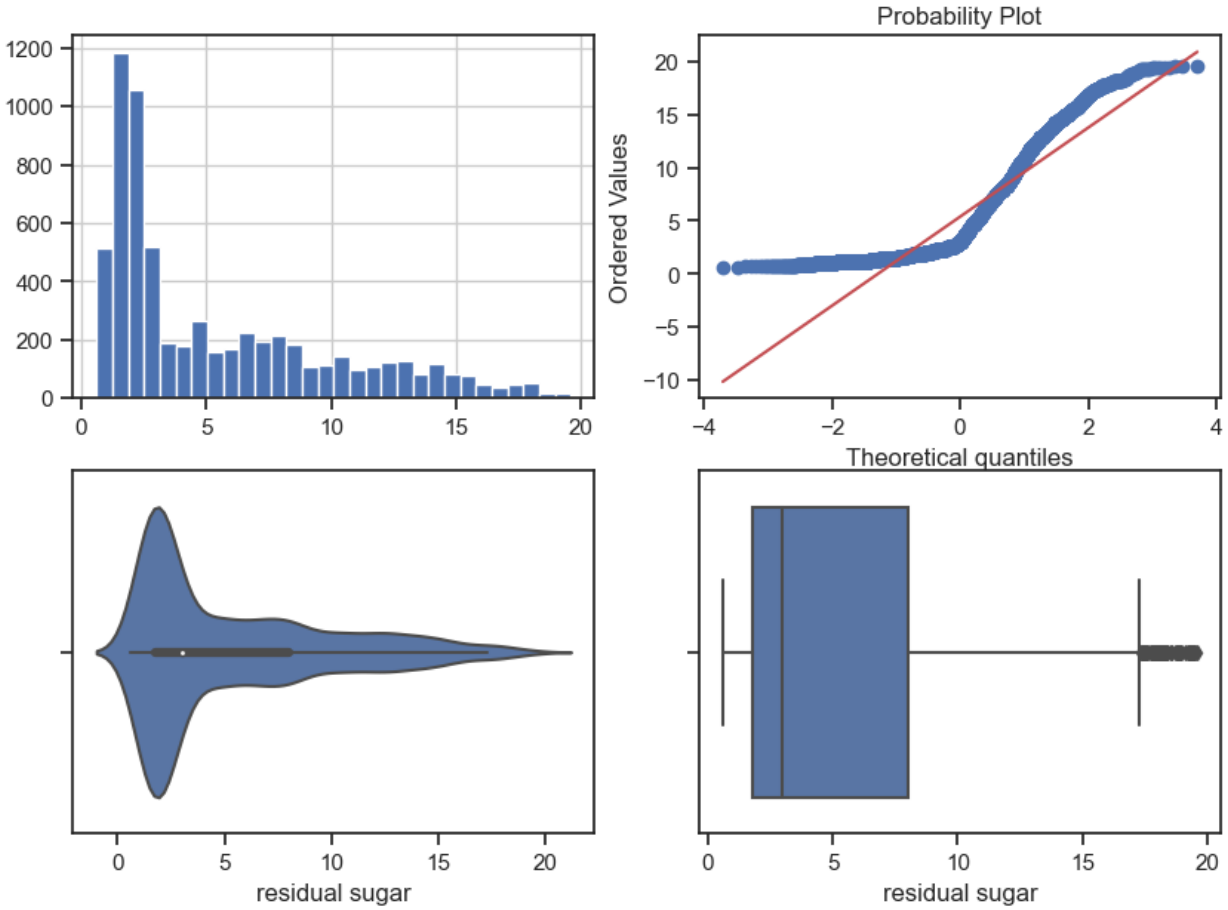




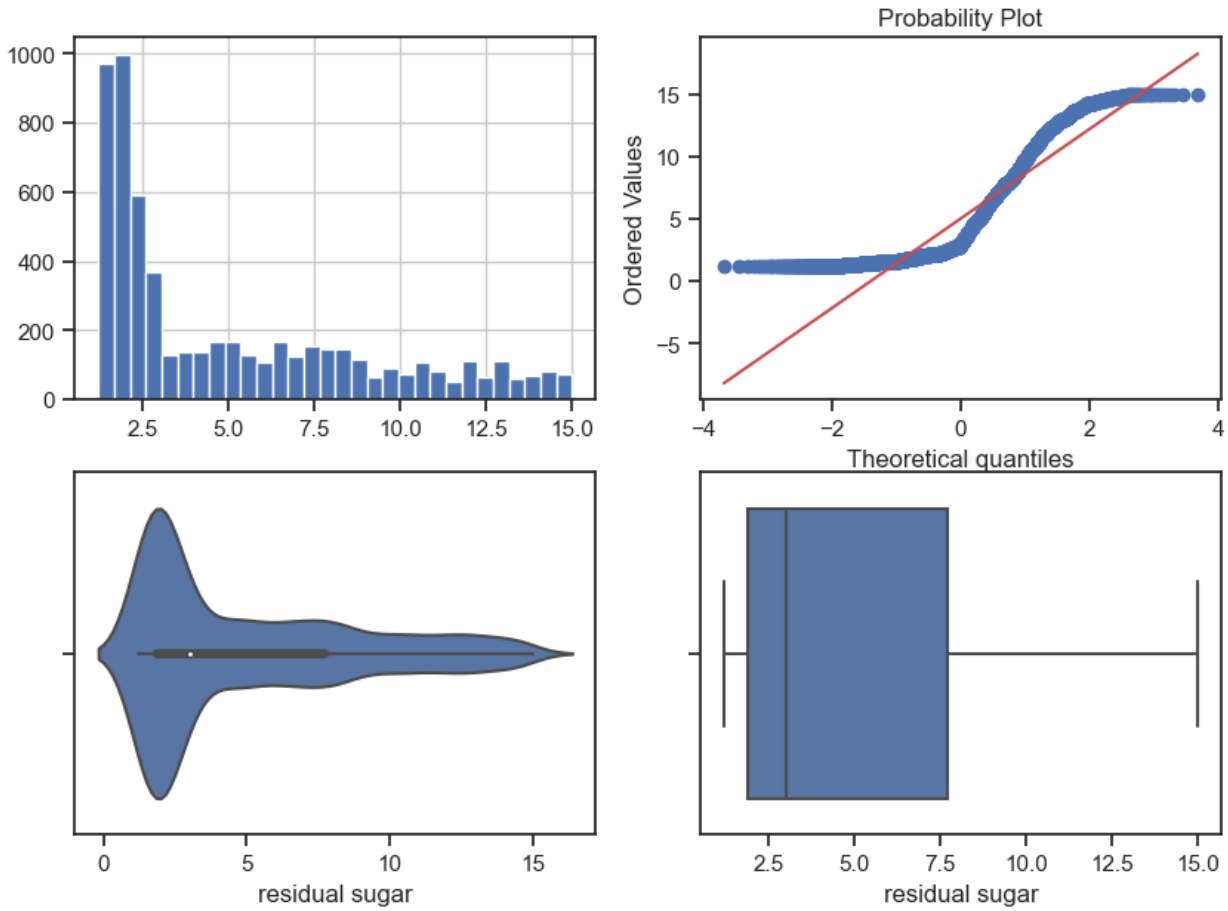
Поле-citric acid, метод-OutlierBoundaryType.IRQ, строк-5956



Поле-residual sugar, метод-OutlierBoundaryType.SIGMA, строк-6437



Поле-residual sugar, метод-OutlierBoundaryType.QUANTILE, строк-5821



Обработка по крайней мере одного нестандартного признака

```
from datetime import date
import matplotlib.dates as mpl_dates
timedata = pd.read_csv("DailyDelhiClimateTest.csv")
timedata
```

19]

	date	meantemp	humidity	wind_speed	meanpressure

```
x_col_list_n = ['date']
```

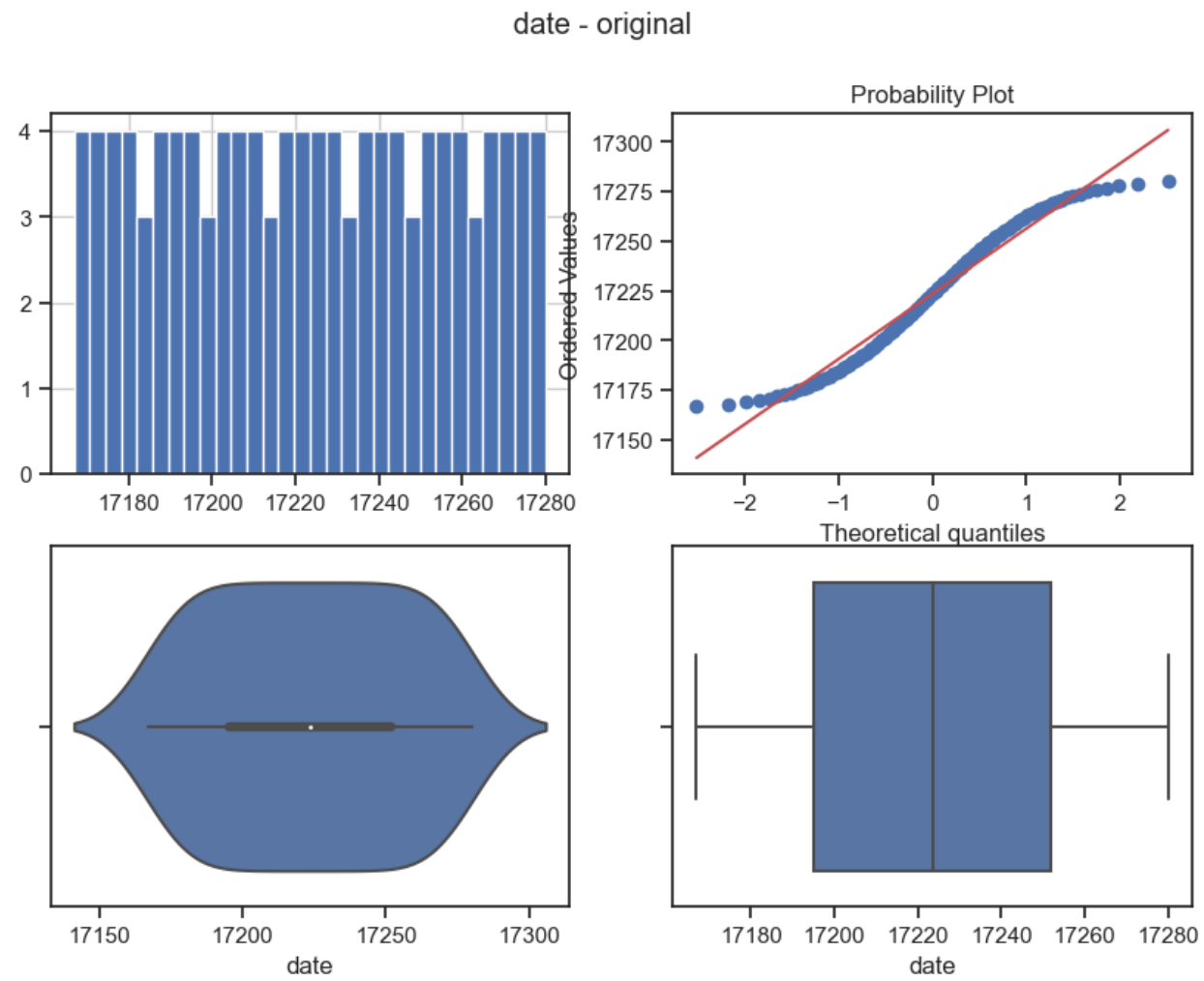
54]

```
timedata.reset_index(inplace=True)
timedata['date']=timedata['date'].apply(mpl_dates.date2num)
timedata = timedata.astype(float)
```

55]

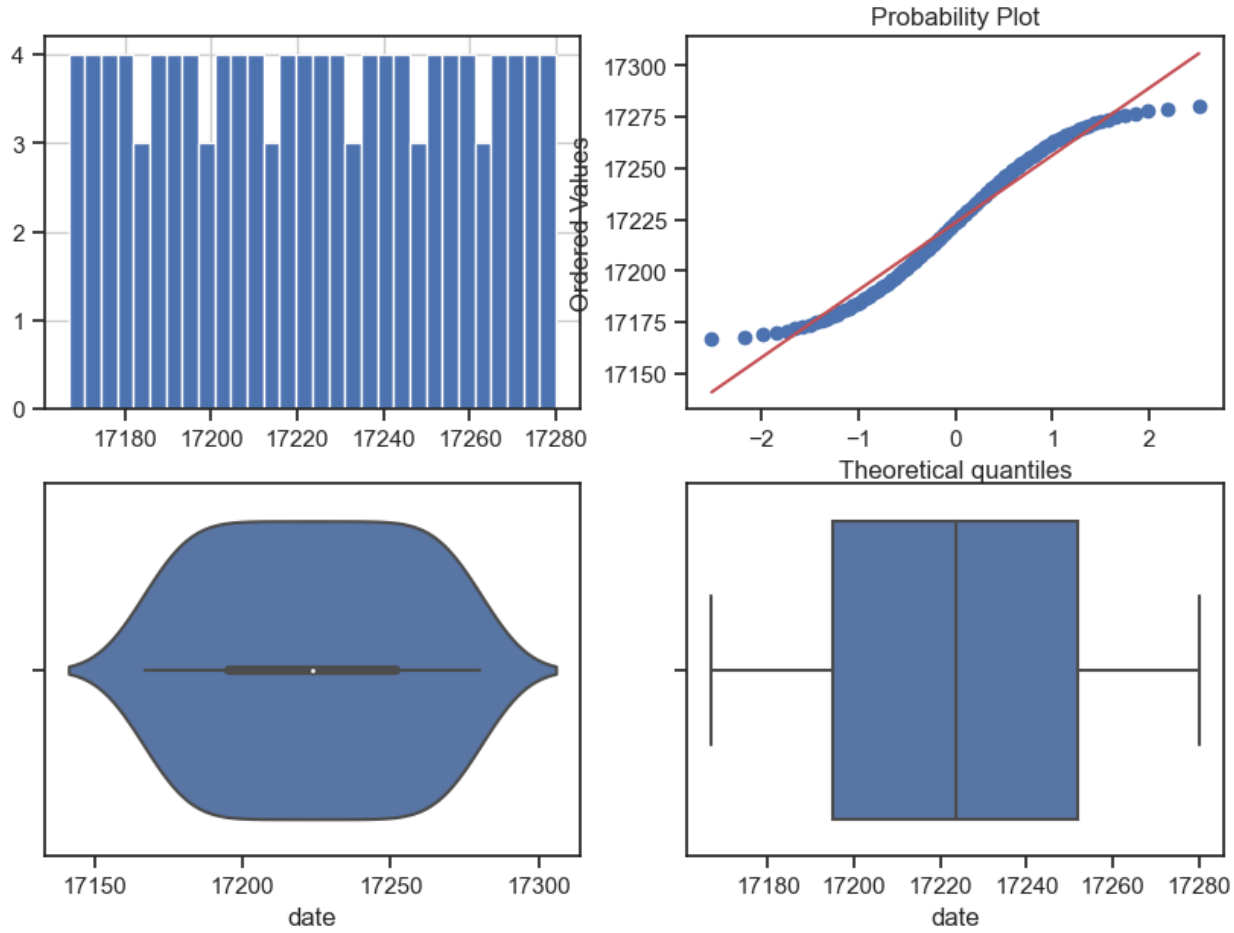
```
diagnostic_plots(timedata, 'date', 'date - original')
```

56]




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

Поле-date, метод-OutlierBoundaryType.SIGMA, строк-114



Отбор признаков

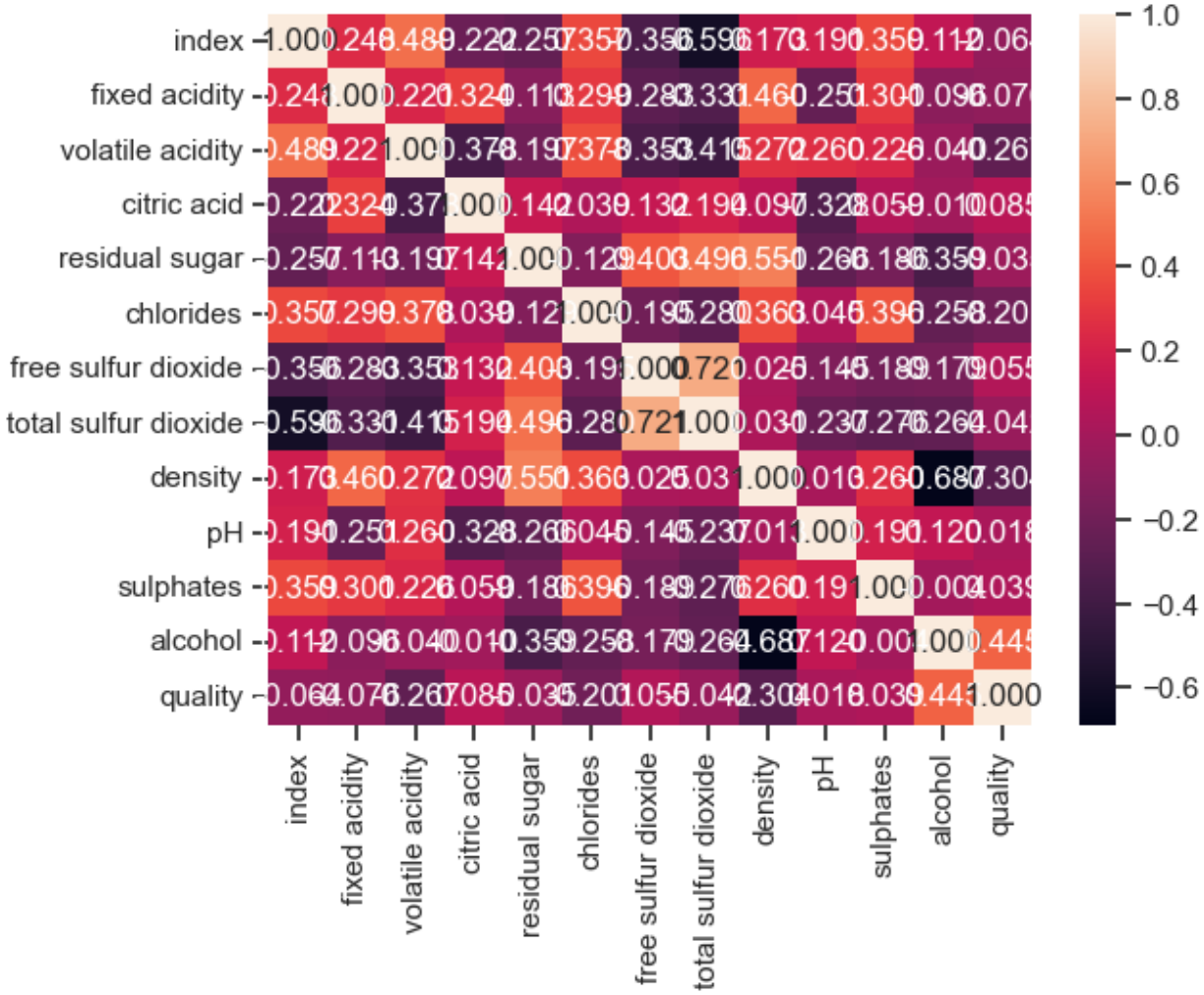
Методы фильтрации (filter methods)

```
selector_1211 = VarianceThreshold(threshold=0.15)
selector_1211.fit(data)
# Значения дисперсий для каждого признака
selector_1211.variances_
```

```
array([3.49600967e+06, 1.85727591e-01, 1.68431767e+00, 2.71018852e-02,
       2.10949595e-02, 2.26241369e+01, 1.23012312e-03, 3.15326731e+02,
       3.19477751e+03, 9.00726891e-06, 2.58043973e-02, 2.21717714e-02,
       1.42333473e+00, 7.62510717e-01])
```

```
# Константный и псевдоконстантный признаки удалены
selector_1211.transform(data)
```

```
array([[0, 0, 7.0, ..., 170.0, 8.8, 6],
       [1, 0, 6.3, ..., 132.0, 9.5, 6],
       [2, 0, 8.1, ..., 97.0, 10.1, 6],
       ...,
       [6494, 1, 6.3, ..., 40.0, 11.0, 6],
       [6495, 1, 5.9, ..., 44.0, 10.2, 5],
       [6496, 1, 6.0, ..., 42.0, 11.0, 6]], dtype=object)
```


index
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Корреляций 0.8 и выше нет

```
make_corr_df(data, 0.5)
```

[65]

	f1	f2	corr
0	total sulfur dioxide	free sulfur dioxide	0.721476
1	free sulfur dioxide	total sulfur dioxide	0.721476
2	alcohol	density	0.687432
3	density	alcohol	0.687432
4	total sulfur dioxide	index	0.596102
5	index	total sulfur dioxide	0.596102
6	density	residual sugar	0.551494
7	residual sugar	density	0.551494

```python
# Обнаружение групп коррелирующих признаков
def corr_groups(cr):
    grouped_feature_list = []
    correlated_groups = []

    for feature in cr['f1'].unique():
        if feature not in grouped_feature_list:
            # находим коррелирующие признаки
            correlated_block = cr[cr['f1'] == feature]
            cur_dups = list(correlated_block['f2'].unique()) + [feature]
            grouped_feature_list = grouped_feature_list + cur_dups
            correlated_groups.append(cur_dups)
    return correlated_groups
```

```python
# Группы коррелирующих признаков
corr_groups(make_corr_df(data, 0.5))
```

```
[['free sulfur dioxide', 'index', 'total sulfur dioxide'],
 ['density', 'alcohol'],
 ['density', 'residual sugar']]
```

Методы обертывания (wrapper methods)

```
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS

knn = KNeighborsClassifier(n_neighbors=3)
```

+ Code + Markdown

```
efs1 = EFS(knn,
           min_features=2,
           max_features=4,
           scoring='accuracy',
           print_progress=True,
           cv=5)

x_col_all_not = ['type','fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dioxide',
           'total sulfur dioxide','density','pH','sulphates','alcohol','quality']

efs1 = efs1.fit(X_test, y_test, custom_feature_names=x_col_all)

print('Best accuracy score: %.2f' % efs1.best_score_)
print('Best subset (indices):', efs1.best_idx_)
print('Best subset (corresponding names):', efs1.best_feature_names_)
```

Методы вложений (embedded methods)

```
# Используем L1-регуляризацию
e_lr1 = LogisticRegression(C=1000, solver='liblinear', penalty='l1', max_iter=500, random_state=1)
e_lr1.fit(X_test, y_test)
# Коэффициенты регрессии
e_lr1.coef_
```

```
array([[ 2.32506550e-03, -8.16523994e+00, -1.22362024e+00,
```

```
# Все 12 признаков являются "хорошими"
sel_e_lr1 = SelectFromModel(e_lr1)
sel_e_lr1.fit(X_test, y_test)
sel_e_lr1.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True])
```

```
e_lr2 = LinearSVC(C=0.01, penalty="l1", max_iter=2000, dual=False)
e_lr2.fit(X_test, y_test)
# Коэффициенты регрессии
e_lr2.coef_
```

```
    # Признаки 1,3,4,6,9,11 исключены
    sel_e_lr2 = SelectFromModel(e_lr2)
    sel_e_lr2.fit(X_test, y_test)
    sel_e_lr2.get_support()
```

```
array([ True, False,  True, False, False,  True, False,  True,  Tr
       False,  True, False,  True])
```

```
    # Используем L1-регуляризацию
    e_ls1 = Lasso(random_state=1)
    e_ls1.fit(X_test, y_test)
    # Коэффициенты регрессии
    list(zip(x_col_all, e_ls1.coef_))
```

```
[('type', -4.05281522319424e-05),
 ('fixed acidity', -0.0),
 ('volatile acidity', -0.0),
 ('citric acid', -0.0),
 ('residual sugar', 0.0),
 ('chlorides', 0.0),
 ('free sulfur dioxide', -0.0),
 ('total sulfur dioxide', 0.002751714205935953),
 ('density', -0.0017997923625971046),
 ('pH', -0.0),
 ('sulphates', -0.0),
 ('alcohol', 0.0),
 ('quality', 0.0)]
```

```
    sel_e_ls1 = SelectFromModel(e_ls1)
    sel_e_ls1.fit(X_test, y_test)
    list(zip(x_col_all, sel_e_ls1.get_support()))
```

6]

```
[('type', True),
 ('fixed acidity', False),
 ('volatile acidity', False),
 ('citric acid', False),
 ('residual sugar', False),
 ('chlorides', False),
 ('free sulfur dioxide', False),
 ('total sulfur dioxide', True),
 ('density', True),
 ('pH', False),
 ('sulphates', False),
 ('alcohol', False),
 ('quality', False)]
```