```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
import scipy.stats as stats
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.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from google.colab import drive
```

```
data = pd.read_csv("/content/house_sales.csv")
```

```
data.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl

5 rows × 81 columns

```
data = data.drop('Id', 1)
data.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: In a fut
an IPython kernel.
```

Сохранено

```
# Удаление колонок с высоким процентом пропусков (более 25%)
data.dropna(axis=1, thresh=1095)
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilit
0	60	RL	65.0	8450	Pave	Reg	Lvl	All
1	20	RL	80.0	9600	Pave	Reg	Lvl	All
2	60	RL	68.0	11250	Pave	IR1	Lvl	All
3	70	RL	60.0	9550	Pave	IR1	Lvl	All
4	60	RL	84.0	14260	Pave	IR1	Lvl	All
...	...	...	...	...	...	...	...	
1455	60	RL	62.0	7917	Pave	Reg	Lvl	All
1456	20	RL	85.0	13175	Pave	Reg	Lvl	All
1457	70	RL	66.0	9042	Pave	Reg	Lvl	All
1458	20	RL	68.0	9717	Pave	Reg	Lvl	All
1459	20	RL	75.0	9937	Pave	Reg	Lvl	All

1460 rows × 75 columns

```
# Заполним пропуски средними значениями
def impute_na(df, variable, value):
    df[variable].fillna(value, inplace=True)
impute_na(data, 'LotFrontage', data['LotFrontage'].mean())


data.describe()
```

Сохранено

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Y
[illegible]	[illegible]	[illegible]0	1460.000000	1460.000000	1460.000000	1460.000000	
mean	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	
std	42.300571	22.024023	9981.264932	1.382997	1.112799	30.202904	
min	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	
25%	20.000000	60.000000	7553.500000	5.000000	5.000000	1954.000000	

```
def obj_col(column):
    return column[1] == 'object'

col_names = []
for col in list(filter(obj_col, list(zip(list(data.columns), list(data.dtypes))))):
  col_names.append(col[0])
col_names.append('SalePrice')

X_ALL = data.drop(col_names, axis=1)
```

```
# 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['SalePrice'],
                                                    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
```

```
((1168, 36), (292, 36))
```

StandardScaler

```
# 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
```

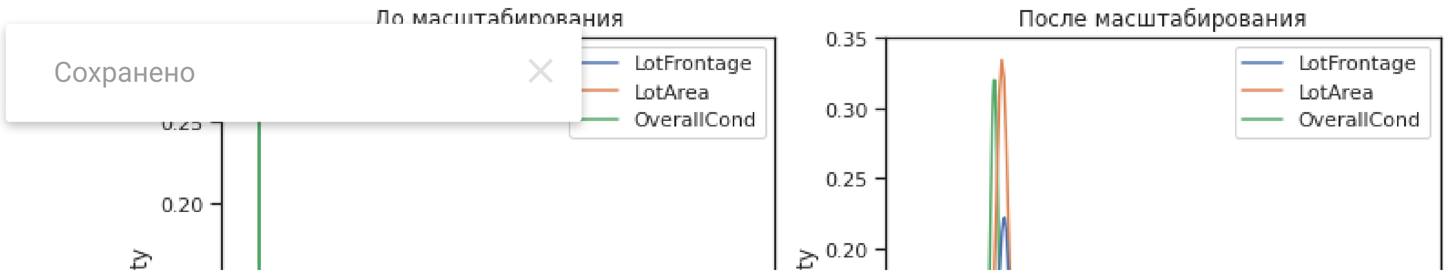
Сохранено

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemo
[illegible]	[illegible]	[illegible]	-0.207142	0.651479	-0.517200	1.050994	0.87
1	-0.872563	0.451936	-0.091886	-0.071836	2.179628	0.156734	-0.42
2	0.073375	-0.093110	0.073480	0.651479	-0.517200	0.984752	0.83
3	0.309859	-0.456474	-0.096897	0.651479	-0.517200	-1.863632	-0.72
4	0.073375	0.633618	0.375148	1.374795	-0.517200	0.951632	0.73
...	...	...	...	...	...	...	...
1455	0.073375	-0.365633	-0.260560	-0.071836	-0.517200	0.918511	0.73
1456	-0.872563	0.679039	0.266407	-0.071836	0.381743	0.222975	0.15
1457	0.309859	-0.183951	-0.147810	0.651479	3.078570	-1.002492	1.02
1458	-0.872563	-0.093110	-0.080160	-0.795151	0.381743	-0.704406	0.53
1459	-0.872563	0.224833	-0.058112	-0.795151	0.381743	-0.207594	-0.96

1460 rows × 36 columns

```
# 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(['LotFrontage', 'LotArea', 'OverallCond'], data, data_cs11_scaled, 'До масштабирован
```



Масштабирование "Mean Normalisation"

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

```
((1168, 36), (292, 36))
```

```
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()
```

Сохранено

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Yea
[illegible]	[illegible]	[illegible]0	1460.000000	1460.000000	1460.000000	1460.000000	14
mean	0.000962	-0.000452	-0.000119	-0.003900	-0.003058	-0.003544	
std	0.248827	0.075425	0.046653	0.153666	0.158971	0.218862	
min	-0.216081	-0.168431	-0.043200	-0.570491	-0.656678	-0.722876	
25%	-0.216081	-0.034869	-0.013970	-0.126046	-0.085250	-0.128673	
50%	-0.039610	-0.000452	-0.004973	-0.014935	-0.085250	0.009008	
75%	0.078037	0.030199	0.004951	0.096176	0.057608	0.204661	
max	0.783919	0.831569	0.956800	0.429509	0.486179	0.277124	

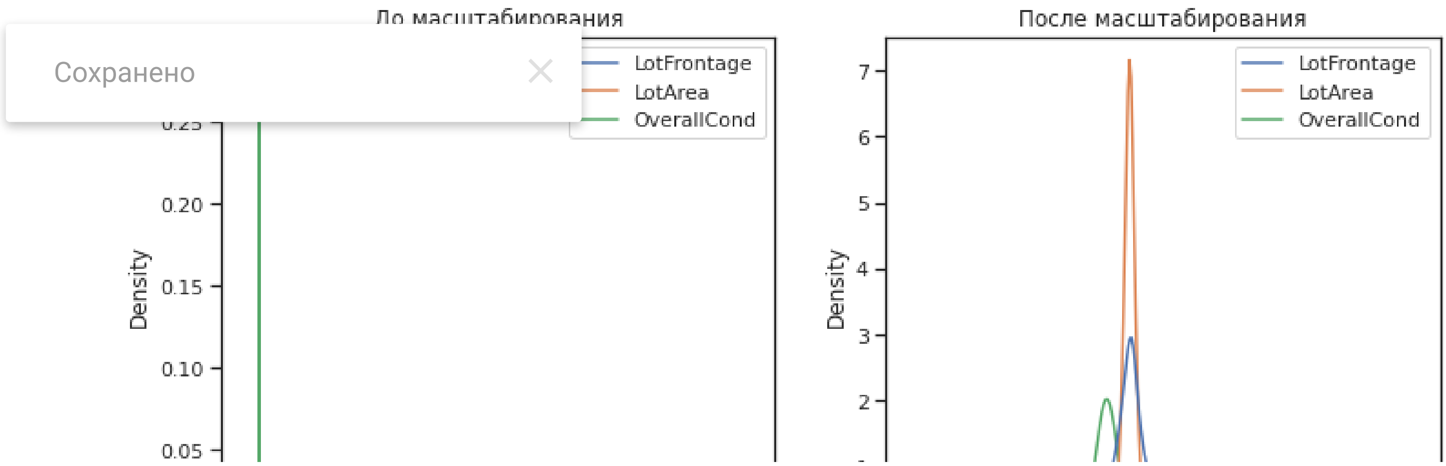
```
cs22 = MeanNormalisation()
cs22.fit(X_train)
data_cs22_scaled_train = cs22.transform(X_train)
data_cs22_scaled_test = cs22.transform(X_test)
```

```
data_cs22_scaled_train.describe()
```

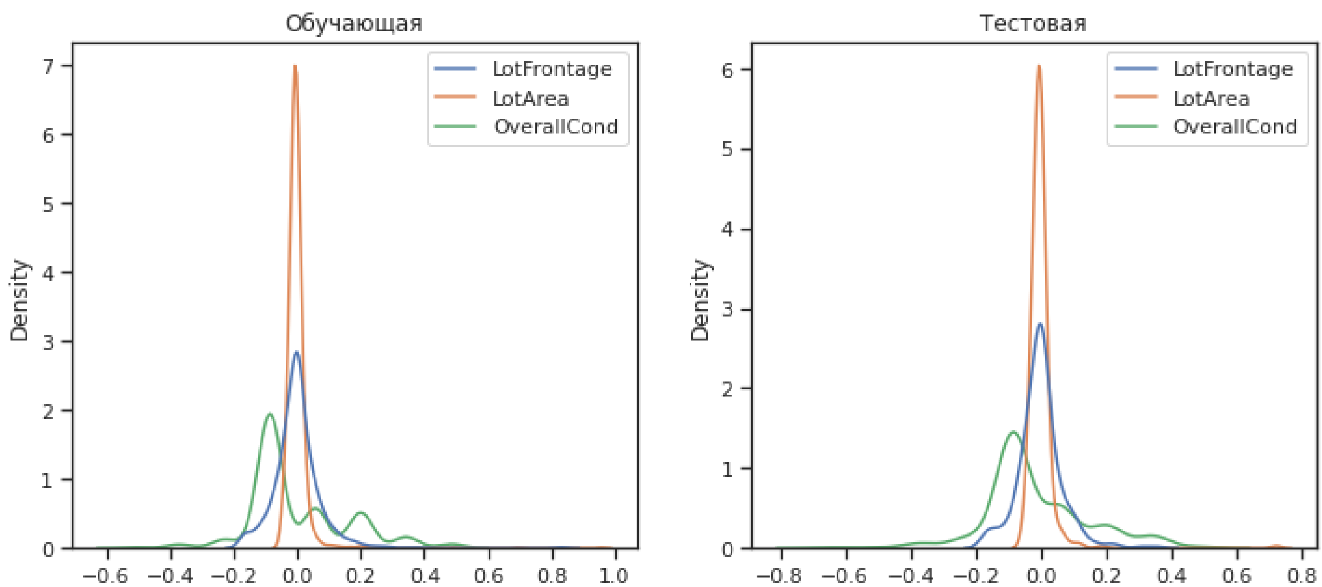
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt
count	1.168000e+03	1.168000e+03	1.168000e+03	1.168000e+03	1.168000e+03	1.168000e+03
mean	-2.932396e-17	6.185596e-17	-2.008002e-18	2.690010e-17	2.934772e-17	7.174151e-16
std	2.475340e-01	7.707084e-02	4.616115e-02	1.522067e-01	1.587482e-01	2.195064e-01
min	-2.160808e-01	-1.684311e-01	-4.319969e-02	-5.704909e-01	-5.138209e-01	-7.228757e-01
25%	-2.160808e-01	-3.486947e-02	-1.422028e-02	-1.260464e-01	-8.524951e-02	-1.286728e-01
50%	-3.961019e-02	-4.518024e-04	-4.865072e-03	-1.493531e-02	-8.524951e-02	1.625472e-02
75%	7.803687e-02	3.019903e-02	5.045185e-03	9.617580e-02	5.760763e-02	2.119069e-01
max	7.839192e-01	8.315689e-01	9.568003e-01	4.295091e-01	4.861791e-01	2.771243e-01

8 rows × 36 columns

```
draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data, data_cs21_scaled, 'До масштабирован
```




```
draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data_cs22_scaled_train, data_cs22_scaled_
```



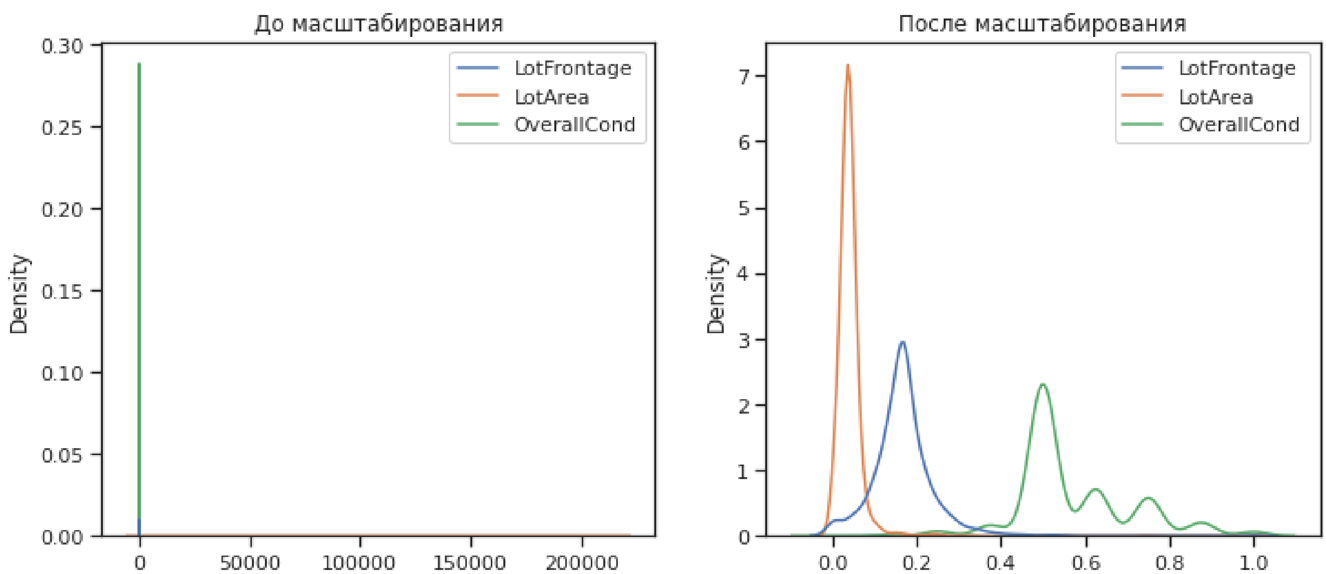
MinMax-масштабирование

```
# 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()
```

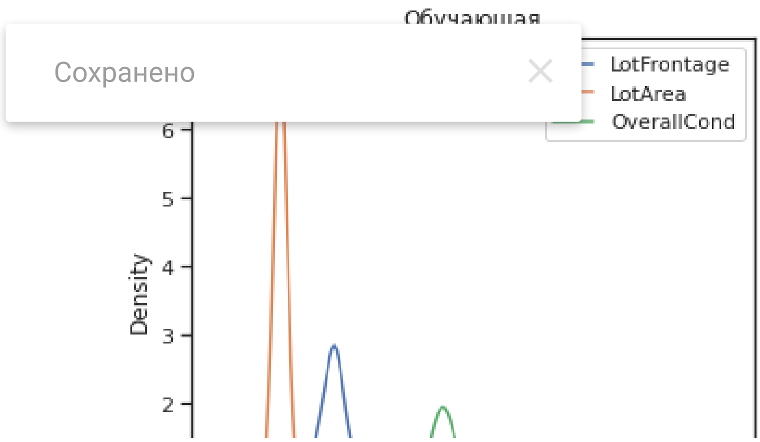
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Yea
		0	1460.000000	1460.000000	1460.000000	1460.000000	14
mean	0.217043	0.167979	0.043080	0.566591	0.571918	0.719332	
std	0.248827	0.075425	0.046653	0.153666	0.139100	0.218862	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.133562	0.029229	0.444444	0.500000	0.594203	
50%	0.176471	0.167979	0.038227	0.555556	0.500000	0.731884	
75%	0.294118	0.198630	0.048150	0.666667	0.625000	0.927536	

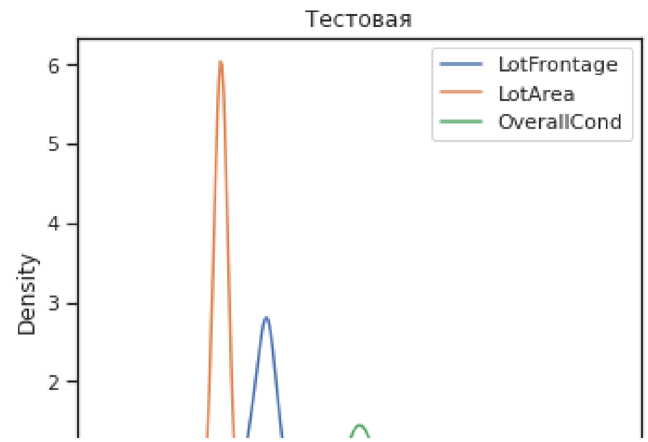
```
cs32 = MinMaxScaler()
cs32.fit(X_train)
data_cs32_scaled_train_temp = cs32.transform(X_train)
data_cs32_scaled_test_temp = cs32.transform(X_test)
# формируем DataFrame на основе массива
data_cs32_scaled_train = arr_to_df(data_cs32_scaled_train_temp)
data_cs32_scaled_test = arr_to_df(data_cs32_scaled_test_temp)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

```
draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data, data_cs31_scaled, 'До масштабирован
```




```
draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data_cs32_scaled_train, data_cs32_scaled_
```





Обработка выбросов для числовых признаков

```
data2 = pd.read_csv("/content/Car_sales.csv")
```

```
data2.head()
```

	Manufacturer	Model	Sales_in_thousands	__year_resale_value	Vehicle_type	Price_i
0	Acura	Integra	16.919	16.360	Passenger	
1	Acura	TL	39.384	19.875	Passenger	
2	Acura	CL	14.114	18.225	Passenger	
3	Acura	RL	8.588	29.725	Passenger	
4	Audi	A4	20.397	22.255	Passenger	

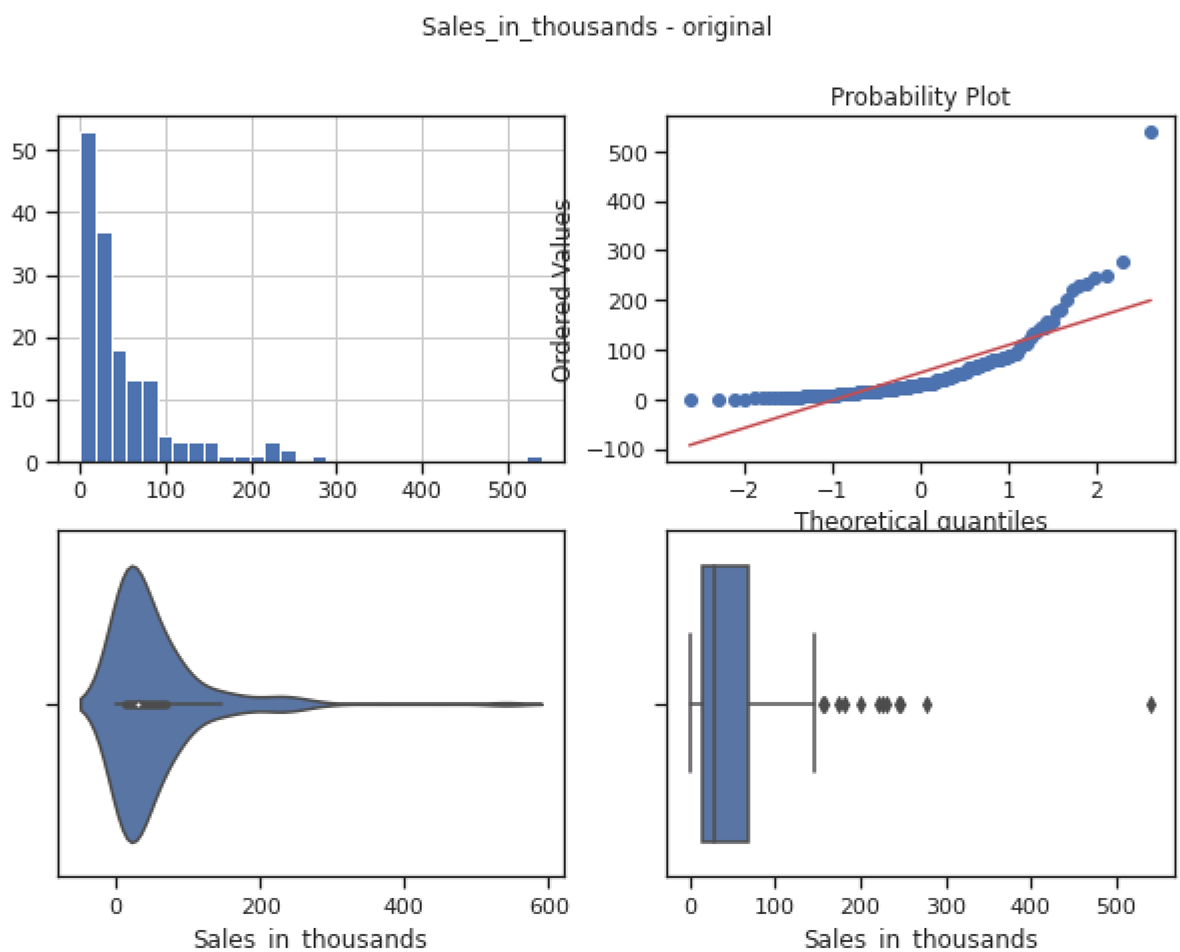
```
data2.describe()
```

Sales_in_thousands	__year_resale_value	Price_in_thousands	Engine_size	Horsep

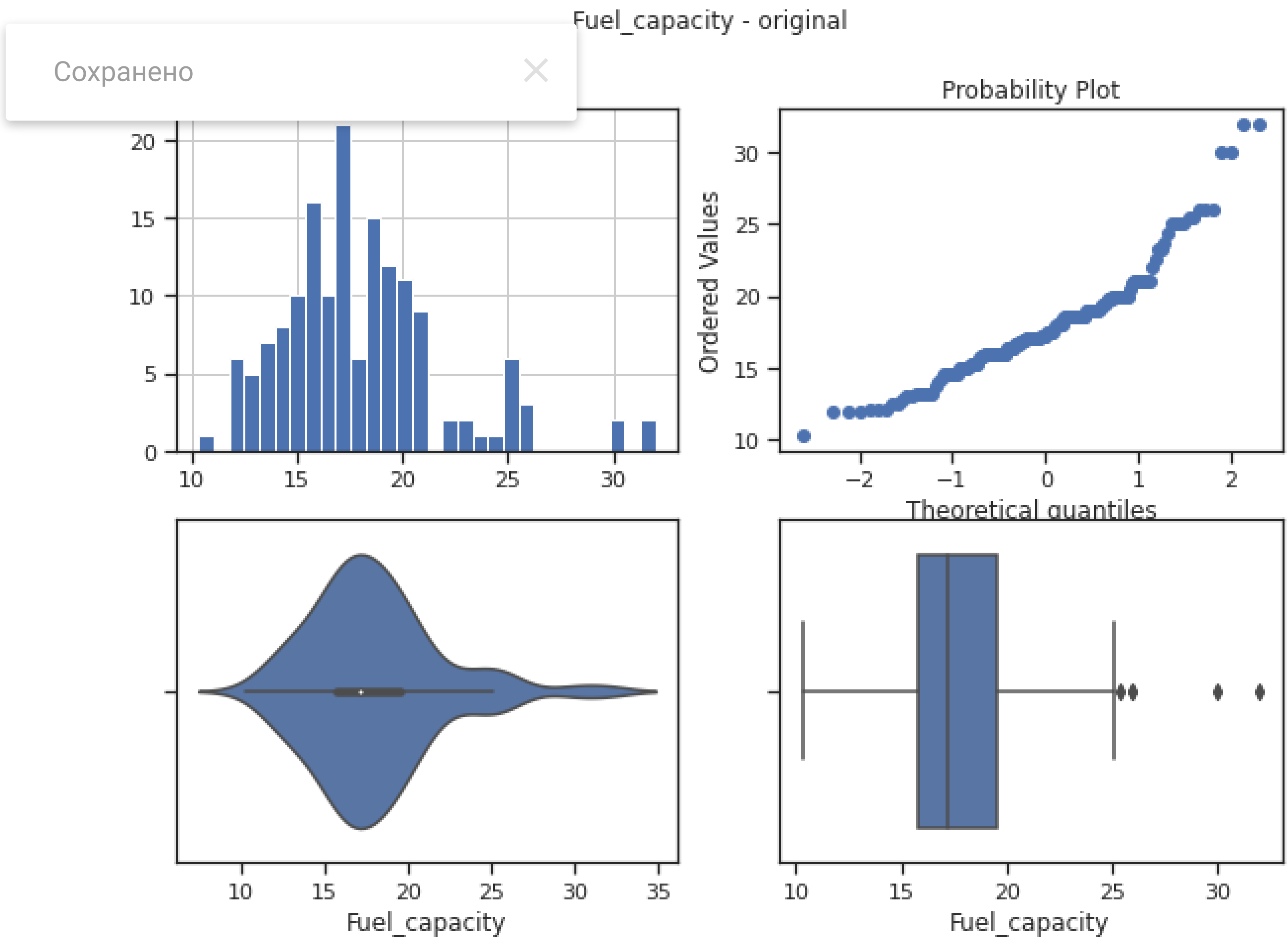
Сохранено

```
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()
```

```
diagnostic_plots(data2, 'Sales_in_thousands', 'Sales_in_thousands - original')
```




```
diagnostic_plots(data2, 'Fuel_capacity', 'Fuel_capacity - 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):
    lower_boundary = df[col].quantile(0.05)
    upper_boundary = df[col].quantile(0.95)
    return lower_boundary, upper_boundary
```

Удаление выбросов (number_of_reviews)

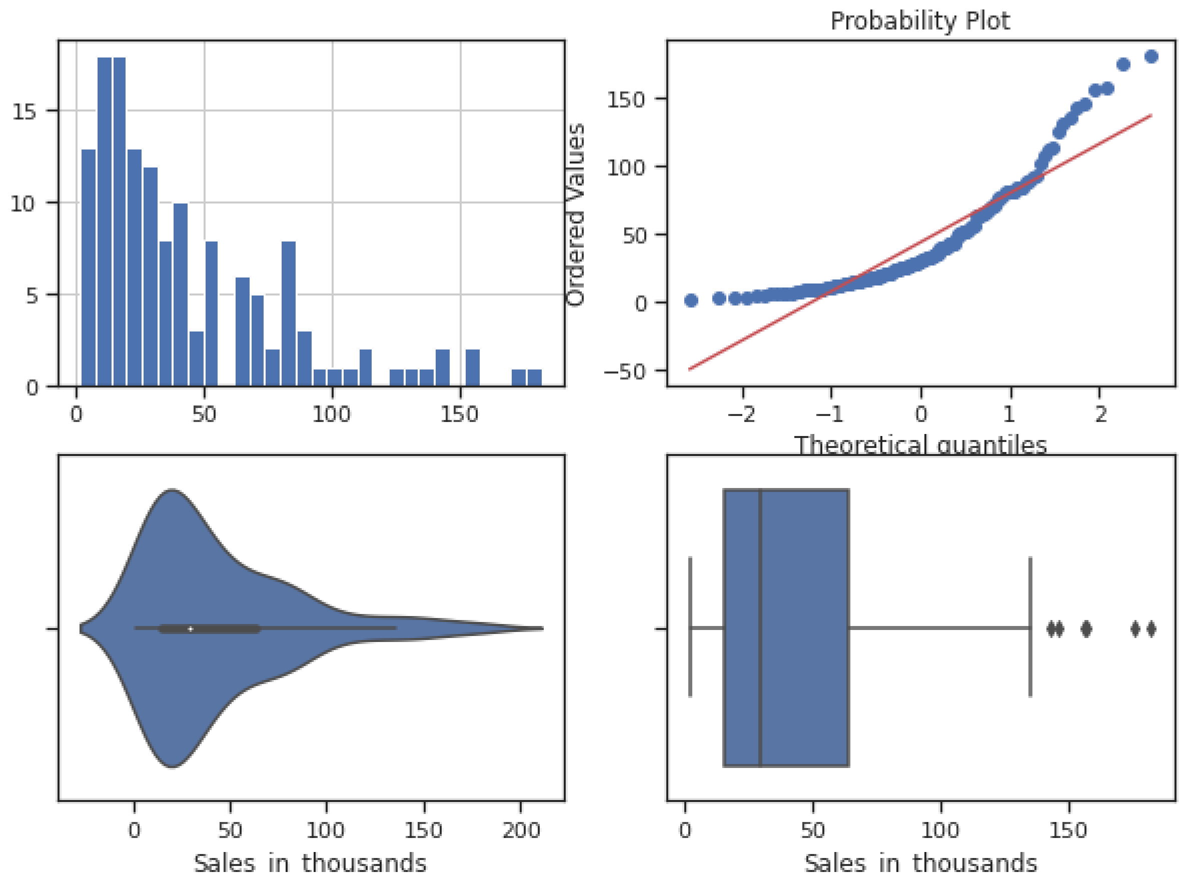
```
# Вычисление верхней и нижней границы
lower_boundary, upper_boundary = get_outlier_boundaries(data2, "Sales_in_thousands")
# Флаги для удаления выбросов
outliers_temp = np.where(data2["Sales_in_thousands"] > upper_boundary, True,
                         np.where(data2["Sales_in_thousands"] < lower_boundary, True, False))
# Удаление данных на основе флага
```

```
data_trimmed = data2.loc[~(outliers_temp), ]
```

Сохранено

```
}'.format("Sales_in_thousands", "QUANTILE", data_trimmed.s
es_in_thousands", title)
```

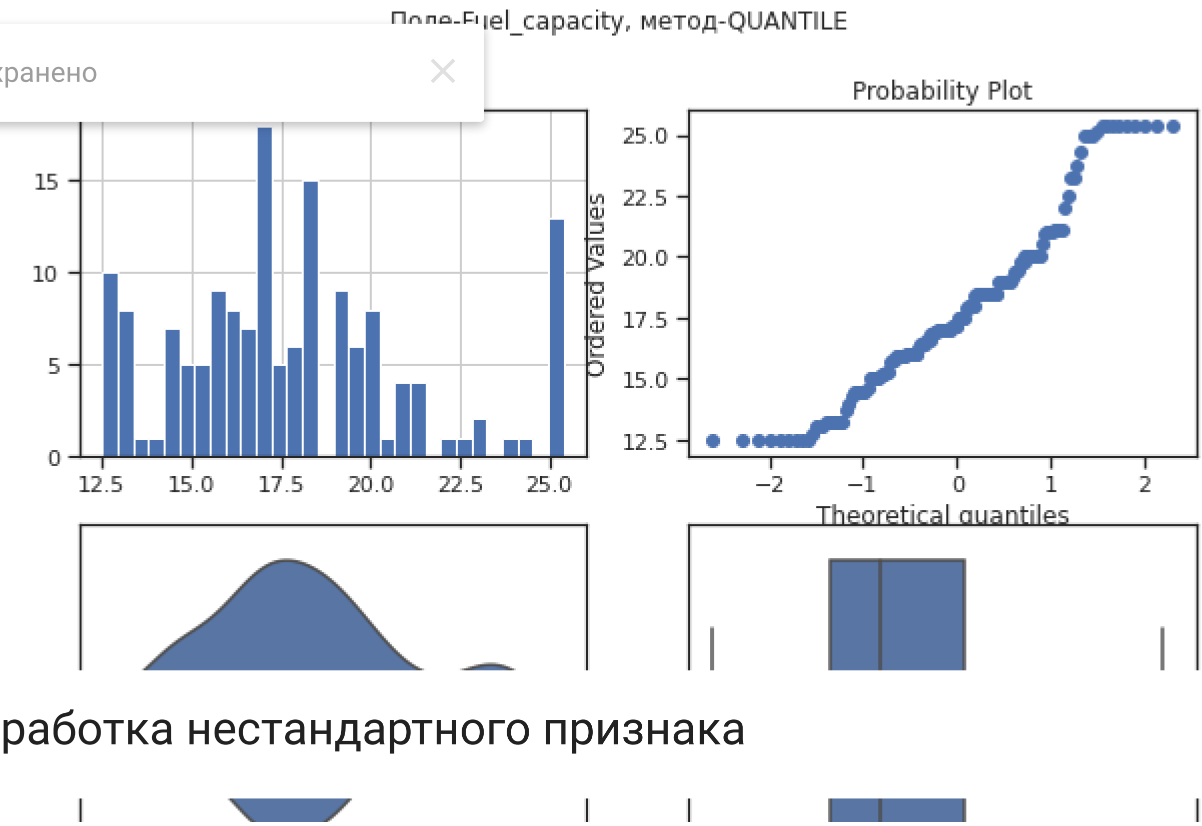
Поле-Sales_in_thousands, метод-QUANTILE, строк-141



Замена выбросов

```
# Вычисление верхней и нижней границы
lower_boundary, upper_boundary = get_outlier_boundaries(data2, "Fuel_capacity")
# Изменение данных
data2["Fuel_capacity"] = np.where(data2["Fuel_capacity"] > upper_boundary, upper_boundary,
                     np.where(data2["Fuel_capacity"] < lower_boundary, lower_boundary, data2[
title = 'Поле-{}, метод-{}'.format("Fuel_capacity", "QUANTILE")
diagnostic_plots(data2, "Fuel_capacity", title)
```

Сохранено



Обработка нестандартного признака

```
data2.dtypes
```

```
Manufacturer             object
Model                    object
Sales_in_thousands      float64
__year_resale_value     float64
Vehicle_type             object
Price_in_thousands      float64
Engine_size             float64
Horsepower              float64
Wheelbase               float64
Width                   float64
Length                  float64
Curb_weight             float64
Fuel_capacity           float64
Fuel_efficiency         float64
Latest_Launch            object
Power_perf_factor       float64
dtype: object
```

```
# Сконвертируем дату и время в нужный формат
data2["Latest_Launch_Date"] = data2.apply(lambda x: pd.to_datetime(x["Latest_Launch"], format
```

```
data2.head(5)
```

Сохранено

	Manufacturer	Model	Sales_in_thousands	__year_resale_value	Vehicle_type	Price_i
0			16.919	16.360	Passenger	
1	Acura	TL	39.384	19.875	Passenger	
2	Acura	CL	14.114	18.225	Passenger	
3	Acura	RL	8.588	29.725	Passenger	
4	Audi	A4	20.397	22.255	Passenger	

```
data2.dtypes
```

```
Manufacturer                     object
Model                            object
Sales_in_thousands              float64
__year_resale_value             float64
Vehicle_type                     object
Price_in_thousands              float64
Engine_size                     float64
Horsepower                      float64
Wheelbase                       float64
Width                           float64
Length                          float64
Curb_weight                     float64
Fuel_capacity                   float64
Fuel_efficiency                 float64
Latest_Launch                    object
Power_perf_factor               float64
Latest_Launch_Date       datetime64[ns]
dtype: object
```

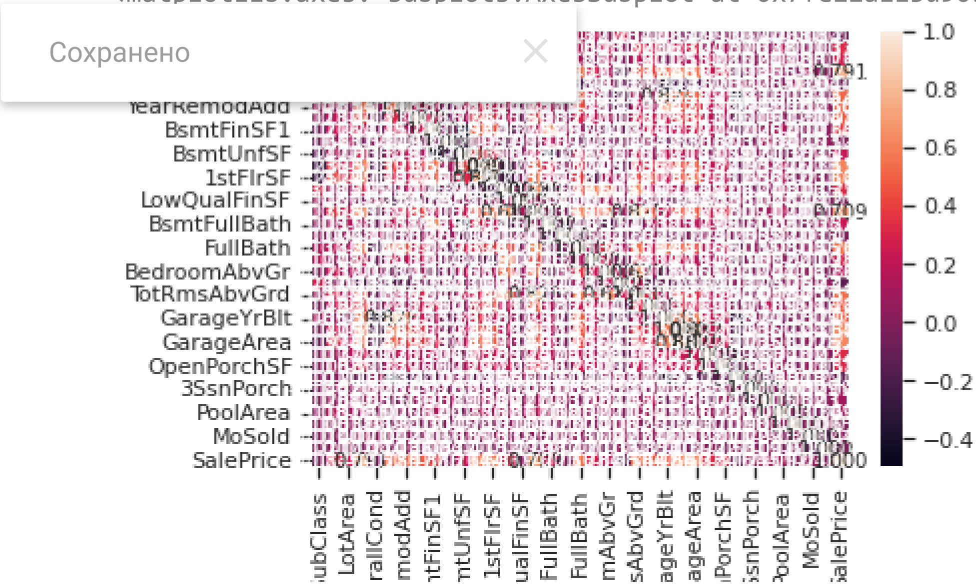
```
# День
data2['Latest_Launch_Day'] = data2['Latest_Launch_Date'].dt.day
# Месяц
data2['Latest_Launch_Month'] = data2['Latest_Launch_Date'].dt.month
# Год
data2['Latest_Launch_Year'] = data2['Latest_Launch_Date'].dt.year
```

Отбор признаков

Метод фильтрации (Корреляция признаков)

```
sns.heatmap(data.corr(), annot=True, fmt='.3f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe22a113d90>
```




```
# Формирование DataFrame с сильными корреляциями
def make_corr_df(df):
    cr = data.corr()
    cr = cr.abs().unstack()
    cr = cr.sort_values(ascending=False)
    cr = cr[cr >= 0.3]
    cr = cr[cr < 1]
    cr = pd.DataFrame(cr).reset_index()
    cr.columns = ['f1', 'f2', 'corr']
    return cr


# 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


# Группы коррелирующих признаков
corr_groups(make_corr_df(data))
```

```
[['GarageArea',
  'SalePrice',
  'OverallQual',
  'GarageYrBlt',
  'YearBuilt',
```

```
        'FullBath',
```



```
        'YearRemodAdd',
        'MasVnrArea',
        'TotRmsAbvGrd',
        'Fireplaces',
        'GarageCars'],
       ['GrLivArea',
        'TotRmsAbvGrd',
        'HalfBath',
        'BedroomAbvGr',
        'FullBath',
        'SalePrice',
        'MSSubClass',
        '2ndFlrSF'],
       ['BsmtFullBath',
        'TotalBsmtSF',
        'BsmtUnfSF',
        '1stFlrSF',
        'SalePrice',
        'BsmtFinSF1'],
       ['1stFlrSF',
        'GrLivArea',
        'TotalBsmtSF',
        'MSSubClass',
        'SalePrice',
        'GarageArea',
        'TotRmsAbvGrd',
        'LotArea',
        'LotFrontage'],
       ['YearBuilt', 'EnclosedPorch'],
       ['YearBuilt', 'GarageYrBlt', 'OverallCond'],
       ['GrLivArea', 'SalePrice', 'OverallQual', 'OpenPorchSF'],
       ['SalePrice', 'WoodDeckSF']]
```

Метод из группы методов вложений

```
data3 = pd.read_csv("/content/WineQT.csv", sep=",")
```

```
X3_ALL = data3.drop(['quality'], axis=1)
```

```
# Разделим выборку на обучающую и тестовую
X3_train, X3_test, y3_train, y3_test = train_test_split(X3_ALL, data3['quality'],
                                                     test_size=0.2,
                                                     random_state=1)
```

```
# Используем L1-регуляризацию
e_lr1 = LogisticRegression(C=1000, solver='liblinear', penalty='l1', max_iter=500, random_sta
```

```
e_lr1.fit(X3_train, y3_train)
```



```
array([[ 8.12685010e-01,  1.13666762e+01,  7.82623669e+00,
         2.73003859e-01,  2.20854445e+00, -8.14499398e-02,
        -6.07359291e-02, -9.71364320e+00,  1.05928330e+01,
        -3.02935401e+00, -3.49793957e+00,  4.48070237e-03],
       [-1.70947991e-02,  3.42135554e+00, -1.21007833e-01,
         8.32452278e-02,  3.20689559e+00,  1.03669460e-02,
        -1.25693925e-02, -5.18479271e+00,  2.46658035e+00,
         9.88462824e-01, -2.04766665e-01, -4.73535890e-04],
       [-1.50633685e-01,  1.93721323e+00,  1.12321685e+00,
         1.01141678e-02,  1.55206374e+00, -1.74615115e-02,
         1.48826890e-02,  5.10001726e+00, -2.81228295e-02,
        -2.62509731e+00, -9.26899115e-01,  5.26799951e-05],
       [ 1.90322225e-01, -1.79843954e+00, -2.04300613e+00,
        -4.72955643e-02,  2.58455381e+00,  1.21352411e-02,
        -7.83754176e-03, -2.99949432e+00,  9.79232831e-01,
         8.78802257e-01,  2.38635326e-01,  1.63131072e-04],
       [-2.89452663e-02, -3.07001091e+00,  1.47490514e+00,
         7.64831115e-02, -1.76133253e+01,  2.58137752e-02,
        -2.04458316e-02, -3.51585085e+00, -1.28269840e+00,
         2.73049298e+00,  8.81957513e-01, -5.47347256e-04],
       [-5.95096357e-01,  3.04283371e+00,  3.41733495e+00,
        -1.83182731e-01, -3.51167880e+01, -2.83696795e-02,
        -2.51328328e-02,  7.93053290e+00, -9.85694602e+00,
         3.86988223e+00,  1.26366792e+00,  6.15531404e-04]])
```

```
# Все признаки являются "хорошими"
from sklearn.feature_selection import SelectFromModel
sel_e_lr1 = SelectFromModel(e_lr1)
sel_e_lr1.fit(X3_train, y3_train)
sel_e_lr1.get_support()
```

```
array([ 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(X3_train, y3_train)
# Коэффициенты регрессии
e_lr2.coef_
```

```
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        -4.11591571e-03,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00, -8.74404725e-02,  2.16193629e-05],
       [-3.25616265e-02,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        -1.53901281e-03,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00, -5.09619360e-02, -7.57493371e-05],
       [ 5.37467586e-03,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00, -1.01440612e-02,
```

Сохранено

```
             9.75223363e-03,  0.00000000e+00,  2.67873814e-01,
                             325110e-01,  6.66630972e-05],
                             000000e+00,  0.00000000e+00,
                             000000e+00,  8.03363371e-03,
            -6.31234283e-03,  0.00000000e+00,  0.00000000e+00,
             0.00000000e+00,  0.00000000e+00,  1.50657492e-05],
           [-3.14859556e-03,  0.00000000e+00,  0.00000000e+00,
             0.00000000e+00,  0.00000000e+00,  3.10821666e-03,
            -4.09581683e-03,  0.00000000e+00, -2.53546831e-01,
             0.00000000e+00,  3.23765903e-02, -8.18789111e-05],
           [-3.59123136e-02,  0.00000000e+00,  0.00000000e+00,
             0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
            -3.69285024e-03,  0.00000000e+00,  0.00000000e+00,
             0.00000000e+00, -4.93611919e-02, -5.75423620e-05]])


# Признаки с флагом False д.б. исключены
sel_e_lr2 = SelectFromModel(e_lr2)
sel_e_lr2.fit(X3_train, y3_train)
sel_e_lr2.get_support()

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