



**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

Курс «Методы машинного обучения»

Отчет по лабораторной работе №3:
«Обработка признаков (часть 2)»

Выполнила:
студентка группы ИУ5-24М
Мащенко Е. И.

Проверил:
Балашов А.М.

Цель работы

Изучение продвинутых способов предварительной обработки данных для дальнейшего формирования моделей.

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.
2. 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

Лабораторная работа №3

```
B [1]: In [1]: 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
```

```
B [2]: In [2]: data = pd.read_csv("house_sales.csv")
```

```
B [3]: In [3]: data.head()
```

Out[3]:

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

5 rows × 81 columns

```
B [4]: In [4]: data = data.drop('Id', 1)
data.head()
```

Out[4]:

		MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	NaN
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	...	0	NaN	NaN	NaN	NaN
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	NaN
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	...	0	NaN	NaN	NaN	NaN
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	...	0	NaN	NaN	NaN	NaN

5 rows × 80 columns

```
B [5]: In [5]: # Удаление колонок с высоким процентом пропусков (более 25%)
data.dropna(axis=1, thresh=1095)
```

Out[5]:

		MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	EnclosedPorch	3SsnPorch	ScreenPorch
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	0	0
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	...	0	0	0
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	...	0	0	0
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	...	272	0	0
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	...	0	0	0
...
1455	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub	Inside	Gtl	0	0	0
1456	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub	Inside	Gtl	0	0	0
1457	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub	Inside	Gtl	0	0	0
1458	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub	Inside	Gtl	112	0	0
1459	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub	Inside	Gtl	0	0	0

1460 rows × 75 columns

```
B [6]: In [6]: # Заполним пропуски средними значениями
def impute_na(df, variable, value):
    df[variable].fillna(value, inplace=True)
impute_na(data, 'LotFrontage', data['LotFrontage'].mean())
```

```
B [7]: In [7]: data.describe()
```

Out[7]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	Wood
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000	...	1460.000000
mean	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46.549315	...	94.000000
std	42.300571	22.024023	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	161.319273	...	12.000000
min	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	...	0.000000
25%	20.000000	60.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000	...	0.000000
50%	50.000000	70.049958	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000	...	0.000000
75%	70.000000	79.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	0.000000	...	16.000000
max	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000	...	857.000000

8 rows × 37 columns

```
B [8]: 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')
```

```
B [9]: X_ALL = data.drop(col_names, axis=1)
```

```
B [10]: # 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
```

```
B [11]: # 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
```

```
Out[11]: ((1168, 36), (292, 36))
```

StandardScaler

```
B [12]: # 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)
```

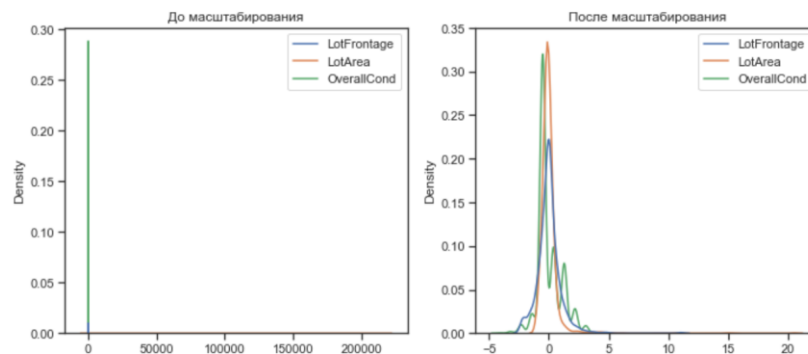
```
Out[12]:
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	GarageArea	V
0	0.073375	-0.229372	-0.207142	0.651479	-0.517200	1.050994	0.878668	0.510015	0.575425	-0.288653	...	0.351000	
1	-0.872563	0.451936	-0.091886	-0.071836	2.179628	0.156734	-0.429577	-0.572835	1.171992	-0.288653	...	-0.060731	
2	0.073375	-0.093110	0.073480	0.651479	-0.517200	0.984752	0.830215	0.322174	0.092907	-0.288653	...	0.631726	
3	0.309859	-0.456474	-0.096897	0.651479	-0.517200	-1.863632	-0.720298	-0.572835	-0.499274	-0.288653	...	0.790804	
4	0.073375	0.633618	0.375148	1.374795	-0.517200	0.951632	0.733308	1.360826	0.463568	-0.288653	...	1.698485	
...	
1455	0.073375	-0.365633	-0.260560	-0.071836	-0.517200	0.918511	0.733308	-0.572835	-0.973018	-0.288653	...	-0.060731	
1456	-0.872563	0.679039	0.266407	-0.071836	0.381743	0.222975	0.151865	0.084610	0.759659	0.722112	...	0.126420	
1457	0.309859	-0.183951	-0.147810	0.651479	3.078570	-1.002492	1.024029	-0.572835	-0.369871	-0.288653	...	-1.033914	
1458	-0.872563	-0.093110	-0.080160	-0.795151	0.381743	-0.704406	0.539493	-0.572835	-0.865548	6.092188	...	-1.090059	
1459	-0.872563	0.224833	-0.058112	-0.795151	0.381743	-0.207594	-0.962566	-0.572835	0.847389	1.509640	...	-0.921624	

1460 rows × 36 columns

```
B [13]: # 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()
```

```
B [14]: draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data, data_cs11_scaled, 'До масштабирования', 'После масштабирования')
```



Масштабирование "Mean Normalisation"

```
B [15]: # 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
```

Out[15]: ((1168, 36), (292, 36))

```
B [16]: 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)
```

```
B [17]: sc21 = MeanNormalisation()
data_cs21_scaled = sc21.fit_transform(X_ALL)
data_cs21_scaled.describe()
```

Out[17]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	Garage
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000	...	1460.00
mean	0.000962	-0.000452	-0.000119	-0.003900	-0.003058	-0.003544	-0.008644	-0.000898	-0.001612	0.001276	...	-0.00
std	0.248827	0.075425	0.046653	0.153666	0.158971	0.218862	0.344090	0.113166	0.080811	0.109443	...	0.15
min	-0.216081	-0.168431	-0.043200	-0.570491	-0.656678	-0.722876	-0.589740	-0.065702	-0.080216	-0.030304	...	-0.33
25%	-0.216081	-0.034869	-0.013970	-0.126046	-0.085250	-0.128673	-0.306407	-0.065702	-0.080216	-0.030304	...	-0.09
50%	-0.039610	-0.000452	-0.004973	-0.014935	-0.085250	0.009008	0.143593	-0.065702	-0.012267	-0.030304	...	0.00
75%	0.078037	0.030199	0.004951	0.096176	0.057608	0.204661	0.310260	0.038048	0.045980	-0.030304	...	0.07
max	0.783919	0.831569	0.956800	0.429509	0.486179	0.277124	0.410260	0.934298	0.919784	0.969696	...	0.66

8 rows x 36 columns

```
B [18]: cs22 = MeanNormalisation()
cs22.fit(X_train)
data_cs22_scaled_train = cs22.transform(X_train)
data_cs22_scaled_test = cs22.transform(X_test)
```

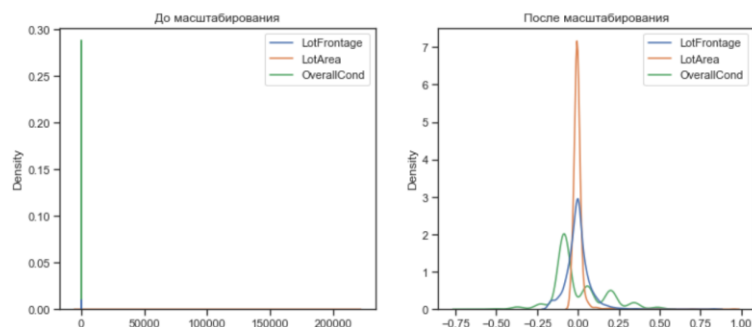
```
B [19]: data_cs22_scaled_train.describe()
```

Out[19]:

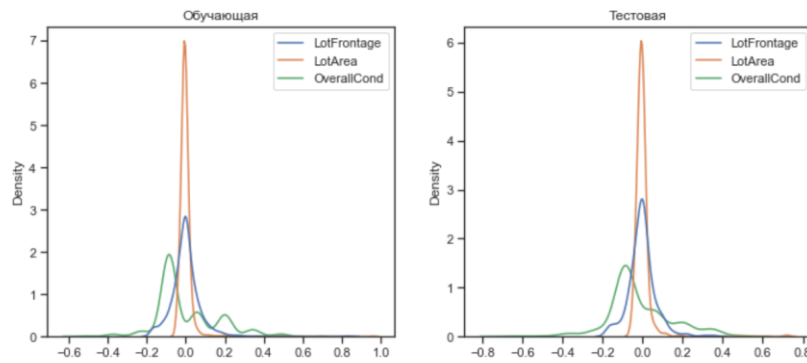
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	..
count	1.168000e+03	1.168000e+03	1.168000e+03	1.168000e+03	1.168000e+03	1.168000e+03	1.168000e+03	1.160000e+03	1.168000e+03	1.168000e+03	..
mean	-1.672939e-17	1.392531e-17	-1.140640e-18	2.718526e-17	9.125121e-18	7.224054e-16	-1.502508e-15	-2.584140e-18	5.322987e-18	-2.471387e-18	..
std	2.475340e-01	7.707084e-02	4.616115e-02	1.522067e-01	1.587482e-01	2.195064e-01	3.431316e-01	1.112988e-01	8.212989e-02	1.098439e-01	..
min	-2.160808e-01	-1.684311e-01	-4.319969e-02	-5.704909e-01	-5.138209e-01	-7.228757e-01	-5.897403e-01	-6.570151e-02	-8.021550e-02	-3.030380e-02	..
25%	-2.160808e-01	-3.486947e-02	-1.422028e-02	-1.260464e-01	-8.524951e-02	-1.286728e-01	-2.897403e-01	-6.570151e-02	-8.021550e-02	-3.030380e-02	..
50%	-3.961019e-02	-4.518024e-04	-4.865072e-03	-1.493531e-02	-8.524951e-02	1.625472e-02	1.435930e-01	-6.570151e-02	-9.609550e-03	-3.030380e-02	..
75%	7.803687e-02	3.019903e-02	5.045185e-03	9.617580e-02	5.760763e-02	2.119069e-01	3.102597e-01	4.070474e-02	4.890392e-02	-3.030380e-02	..
max	7.839192e-01	8.315689e-01	9.568003e-01	4.295091e-01	4.861791e-01	2.771243e-01	4.102597e-01	9.342985e-01	9.197845e-01	9.696962e-01	..

8 rows x 36 columns

```
B [20]: draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data, data_cs21_scaled, 'До масштабирования', 'После масштабирования')
```



B [21]: `draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data_cs22_scaled_train, data_cs22_scaled_test, 'Обучающая', 'Тестовая')`



MinMax-масштабирование

B [22]: `# 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()`

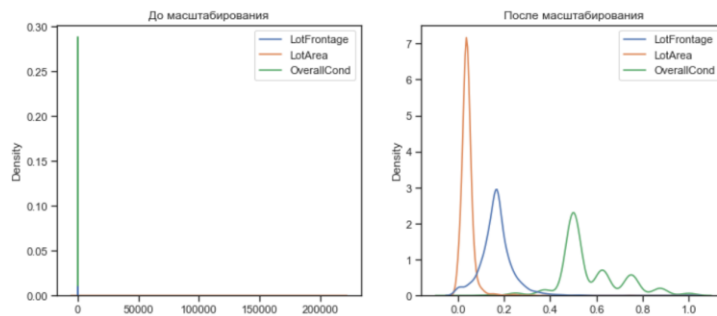
Out[22]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	Garage
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000	...	1460.00
mean	0.217043	0.167979	0.043080	0.566591	0.571918	0.719332	0.581096	0.064803	0.078604	0.031580	...	0.33
std	0.248827	0.075425	0.046653	0.153666	0.139100	0.218862	0.344090	0.113166	0.080811	0.109443	...	0.15
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.00
25%	0.000000	0.133562	0.029229	0.444444	0.500000	0.594203	0.283333	0.000000	0.000000	0.000000	...	0.23
50%	0.176471	0.167979	0.038227	0.555556	0.500000	0.731884	0.733333	0.000000	0.067948	0.000000	...	0.33
75%	0.294118	0.198630	0.048150	0.666667	0.625000	0.927536	0.900000	0.103750	0.126196	0.000000	...	0.40
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.00

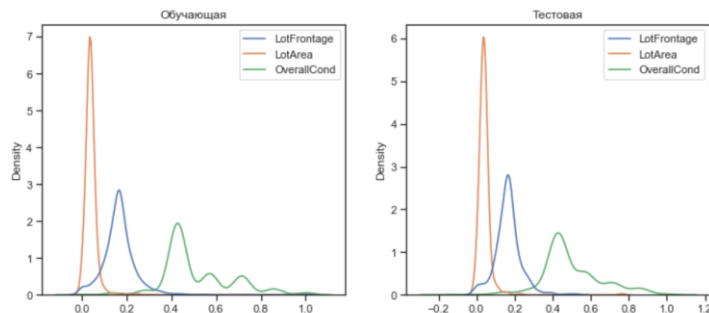
8 rows × 36 columns

B [23]: `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)`

B [24]: `draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data, data_cs31_scaled, 'До масштабирования', 'После масштабирования')`



B [25]: `draw_kde(['LotFrontage', 'LotArea', 'OverallCond'], data_cs32_scaled_train, data_cs32_scaled_test, 'Обучающая', 'Тестовая')`



Обработка выбросов для числовых признаков

B [26]: `data2 = pd.read_csv("Car_sales.csv")`

B [27]: `data2.head()`

```
Out[27]:
```

	Manufacturer	Model	Sales_in_thousands	__year_resale_value	Vehicle_type	Price_in_thousands	Engine_size	Horsepower	Wheelbase	Width	Length
0	Acura	Integra	16.919	16.360	Passenger	21.50	1.8	140.0	101.2	67.3	172.4
1	Acura	TL	39.384	19.875	Passenger	28.40	3.2	225.0	108.1	70.3	192.9
2	Acura	CL	14.114	18.225	Passenger	NaN	3.2	225.0	106.9	70.6	192.0
3	Acura	RL	8.588	29.725	Passenger	42.00	3.5	210.0	114.6	71.4	196.6
4	Audi	A4	20.397	22.255	Passenger	23.99	1.8	150.0	102.6	68.2	178.0

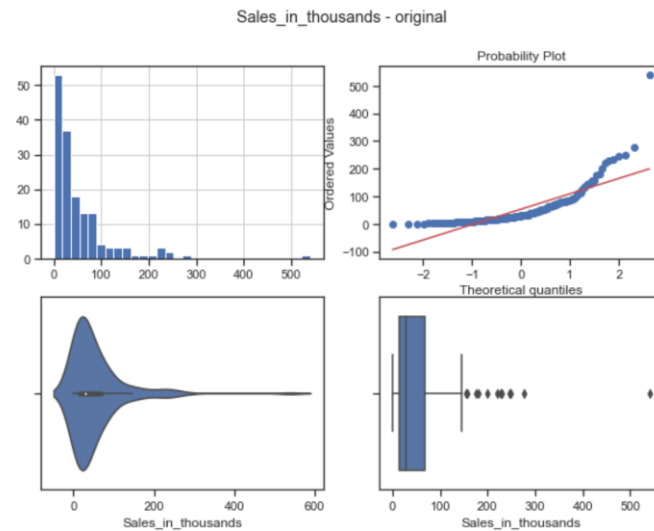
B [28]: `data2.describe()`

```
Out[28]:
```

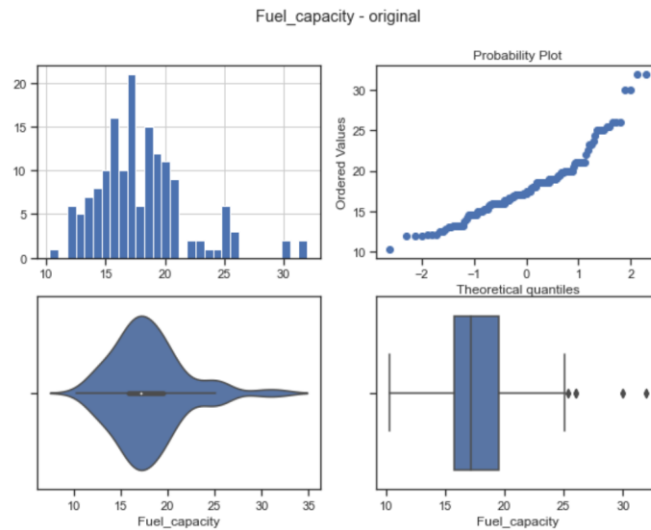
	Sales_in_thousands	__year_resale_value	Price_in_thousands	Engine_size	Horsepower	Wheelbase	Width	Length	Curb_weight	Fuel_capi
count	157.000000	121.000000	155.000000	156.000000	156.000000	156.000000	156.000000	156.000000	155.000000	156.00
mean	52.998076	18.072975	27.390755	3.060897	185.948718	107.487179	71.150000	187.343590	3.378026	17.95
std	68.029422	11.453384	14.351653	1.044653	56.700321	7.641303	3.451872	13.431754	0.630502	3.88
min	0.110000	5.160000	9.235000	1.000000	55.000000	92.600000	62.600000	149.400000	1.895000	10.30
25%	14.114000	11.260000	18.017500	2.300000	149.500000	103.000000	68.400000	177.575000	2.971000	15.80
50%	29.450000	14.180000	22.799000	3.000000	177.500000	107.000000	70.550000	187.900000	3.342000	17.20
75%	67.956000	19.875000	31.947500	3.575000	215.000000	112.200000	73.425000	196.125000	3.799500	19.57
max	540.561000	67.550000	85.500000	8.000000	450.000000	138.700000	79.900000	224.500000	5.572000	32.00

```
B [29]: 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)
# violin plot
plt.subplot(2, 2, 3)
sns.violinplot(x=df[variable])
# boxplot
plt.subplot(2, 2, 4)
sns.boxplot(x=df[variable])
fig.suptitle(title)
plt.show()
```

B [30]: `diagnostic_plots(data2, 'Sales_in_thousands', 'Sales_in_thousands - original')`



```
B [31]: diagnostic_plots(data2, 'Fuel_capacity', 'Fuel_capacity - original')
```

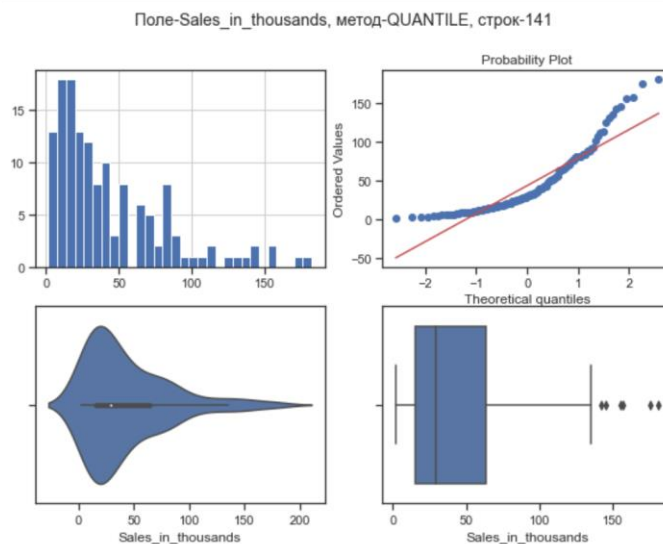


```
B [32]: # Тип вычисления верхней и нижней границы выбросов
from enum import Enum
class OutlierBoundaryType(Enum):
    SIGMA = 1
    QUANTILE = 2
    IRQ = 3
```

```
B [33]: # 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)

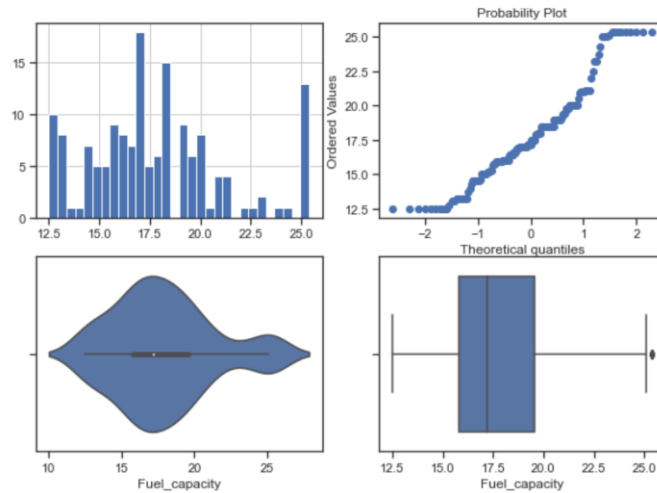
```
B [34]: # Вычисление верхней и нижней границы
lower_boundary, upper_boundary = get_outlier_boundaries(data2, "Sales_in_thousands")
# Флаги для удаления выбросов
outliers_temp = np.where(data2["Sales_in_thousands"] > upper_boundary, True,
                          np.where(data2["Sales_in_thousands"] < lower_boundary, True, False))
# Удаление данных на основе флагов
data_trimmed = data2.loc[~(outliers_temp), ]
title = 'Поле-{}, метод-{}, строка-{}'.format("Sales_in_thousands", "QUANTILE", data_trimmed.shape[0])
diagnostic_plots(data_trimmed, "Sales_in_thousands", title)
```



Замена выбросов

```
B [35]: # 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["Fuel_capacity"]))
title = 'Поле-{}, метод-{}'.format("Fuel_capacity", "QUANTILE")
diagnostic_plots(data2, "Fuel_capacity", title)
```

Поле-Fuel_capacity, метод-QUANTILE



Обработка нестандартного признака

```
B [36]: data2.dtypes
```

```
Out[36]: 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
```

```
B [37]: # Сконвертируем дату и время в нужный формат
data2["Latest_Launch_Date"] = data2.apply(lambda x: pd.to_datetime(x["Latest_Launch"], format='%m/%d/%Y'), axis=1)
```

```
B [38]: data2.head(5)
```

```
Out[38]:
```

	Manufacturer	Model	Sales_in_thousands	__year_resale_value	Vehicle_type	Price_in_thousands	Engine_size	Horsepower	Wheelbase	Width	Length
0	Acura	Integra	16.919	16.360	Passenger	21.50	1.8	140.0	101.2	67.3	172.4
1	Acura	TL	39.384	19.875	Passenger	28.40	3.2	225.0	108.1	70.3	192.9
2	Acura	CL	14.114	18.225	Passenger	NaN	3.2	225.0	106.9	70.6	192.0
3	Acura	RL	8.588	29.725	Passenger	42.00	3.5	210.0	114.6	71.4	196.6
4	Audi	A4	20.397	22.255	Passenger	23.99	1.8	150.0	102.6	68.2	178.0

```
B [39]: data2.dtypes
```

```
Out[39]: 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
```

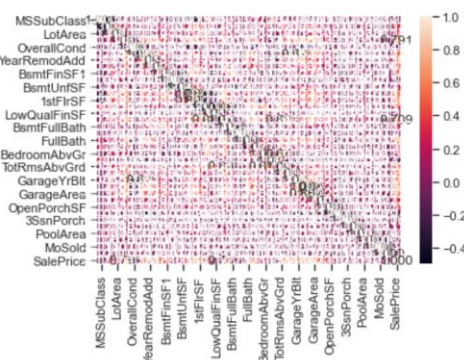
```
B [40]: # День
data2['Latest_Launch_Day'] = data2['Latest_Launch_Date'].dt.day
# Месяц
data2['Latest_Launch_Month'] = data2['Latest_Launch_Date'].dt.month
# Год
data2['Latest_Launch_Year'] = data2['Latest_Launch_Date'].dt.year
```

Отбор признаков

Метод фильтрации (Корреляция признаков)

```
B [41]: sns.heatmap(data.corr(), annot=True, fmt='.3f')
```

```
Out[41]: <AxesSubplot:>
```



```
B [42]: # 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
```

```

B [43]: ► # 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

```

```

B [44]: ► # Группы коррелирующих признаков
corr_groups(make_corr_df(data))

```

```

Out[44]: [['GarageArea',
'SalePrice',
'OverallQual',
'GarageYrBlt',
'YearBuilt',
'FullBath',
'GrLivArea',
'1stFlrSF',
'TotalBsmntSF',
'YearRemodAdd',
'MasVnrArea',
'TotRmsAbvGrd',
'Fireplaces',
'GarageCars'],
['GrLivArea',
'TotRmsAbvGrd',
'HalfBath',
'BedroomAbvGr',
'FullBath',
'SalePrice',
'MSSubClass',
'2ndFlrSF'],
['BsmntFullBath',
'TotalBsmntSF',
'BsmntUnfSF',
'1stFlrSF',
'SalePrice',
'BsmntFinSF1'],
['1stFlrSF',
'GrLivArea',
'TotalBsmntSF',
'MSSubClass',
'SalePrice',
'GarageArea',
'TotRmsAbvGrd',
'LotArea',

```

Метод из группы методов вложений

```
B [45]: data3 = pd.read_csv("WineQT.csv", sep=",")
```

```
B [46]: X3_ALL = data3.drop(['quality'], axis=1)
```

```
B [47]: # 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)
```

```
B [48]: # Используем L1-регуляризацию
e_lr1 = LogisticRegression(C=1000, solver='liblinear', penalty='l1', max_iter=500, random_state=1)
e_lr1.fit(X3_train, y3_train)
# Коэффициенты регрессии
e_lr1.coef_
```

```
Out[48]: 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]])
```

```
B [49]: # 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()
```

```
Out[49]: array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
  True,  True,  True])
```

```
B [50]: 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_
```

```
Out[50]: array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 -4.11584442e-03,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00, -8.74407788e-02,  2.16207334e-05],
 [-3.25628793e-02,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 -1.53902483e-03,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00, -5.09606676e-02, -7.57523845e-05],
 [ 5.38756260e-03,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00, -1.01417935e-02,
  9.74912863e-03,  0.00000000e+00,  2.68698244e-01,
  0.00000000e+00, -1.39093107e-01,  6.67223898e-05],
 [-3.23266765e-03,  0.00000000e+00,  0.00000000e+00,
 -3.14247006e-03,  0.00000000e+00,  8.03254900e-03,
 -6.31217465e-03,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  1.50706409e-05],
 [-3.14854297e-03,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  3.10839571e-03,
 -4.09621571e-03,  0.00000000e+00, -2.53447375e-01],
 [ 0.00000000e+00,  3.23459050e-02, -8.18767831e-05],
 [-3.58424690e-02,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 -3.69131986e-03,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00, -4.94273181e-02, -5.74232107e-05]])
```

```
B [51]: # 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()
```

```
Out[51]: array([ True, False, False,  True, False,  True,  True, False,  True,
  False,  True,  True])
```