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 «Методы машинного обучения в автоматизированных системах обработки информации и управления»

ИСПОЛНИТЕЛЬ: ___Стрихар П.А.__
ФИО

группа ИУ5-25М ____________________
подпись

"23" апреля 2024 г.

ПРЕПОДАВАТЕЛЬ: ____Гапанюк Ю.Е.____
ФИО

____
подпись

"__"_________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

Текстовое описание датасета

В качестве данных для анализа используется датасет `LifeExpectancy.csv`.

Для анализа в ЛР используются не все признаки.

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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Чтение данных

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

```
data.head()
```

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure
0	Afghanistan	2015	Developing	65.0	263	62	0.01	71.279624
1	Afghanistan	2014	Developing	59.9	271	64	0.01	73.523582
2	Afghanistan	2013	Developing	59.9	268	66	0.01	73.219243
3	Afghanistan	2012	Developing	59.5	272	69	0.01	78.184215
4	Afghanistan	2011	Developing	59.2	275	71	0.01	7.097109

```
data.shape
```

```
(2928, 22)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2928 entries, 0 to 2927
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Country                          2928 non-null   object 
 1   Year                             2928 non-null   int64  
 2   Status                           2928 non-null   object 
 3   Life expectancy                  2928 non-null   float64
 4   Adult Mortality                  2928 non-null   int64  
 5   infant deaths                    2928 non-null   int64  
 6   Alcohol                          2735 non-null   float64
 7   percentage expenditure           2928 non-null   float64
 8   Hepatitis B                      2375 non-null   float64
 9   Measles                          2928 non-null   int64  
 10   BMI                             2896 non-null   float64
 11  under-five deaths                2928 non-null   int64  
 12  Polio                            2909 non-null   float64
 13  Total expenditure                2702 non-null   float64
 14  Diphtheria                       2909 non-null   float64
 15   HIV/AIDS                        2928 non-null   float64
 16  GDP                              2485 non-null   float64
 17  Population                       2284 non-null   float64
 18   thinness  1-19 years            2896 non-null   float64
 19   thinness 5-9 years              2896 non-null   float64
 20  Income composition of resources  2768 non-null   float64
 21  Schooling                        2768 non-null   float64
dtypes: float64(15), int64(5), object(2)
memory usage: 503.4+ KB
```

```
data.columns
```

```
Index(['Country', 'Year', 'Status', 'Life expectancy', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
       'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
       ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object')
```

Первичная обработка данных

Оставим в исходной выборке лишь некоторые признаки:

```
data.drop(['Year', ' BMI ', 'Measles ', 'under-five deaths ', 'Diphtheria ', ' HIV/AIDS', ' thinness  1-19 years', ' thinness 5-9 years',
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2928 entries, 0 to 2927
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Country                 2928 non-null   object 
 1   Status                  2928 non-null   object 
 2   Life expectancy         2928 non-null   float64
 3   Adult Mortality         2928 non-null   int64  
 4   infant deaths           2928 non-null   int64  
 5   Alcohol                 2735 non-null   float64
 6   percentage expenditure  2928 non-null   float64
```

```
 7   Hepatitis B             2375 non-null   float64
 8   Polio                   2909 non-null   float64
 9   Total expenditure       2702 non-null   float64
 10  GDP                     2485 non-null   float64
 11  Population              2284 non-null   float64
dtypes: float64(8), int64(2), object(2)
memory usage: 274.6+ KB
```

Удалим пропуски:

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

```
Alcohol : 193
Hepatitis B : 553
Polio : 19
Total expenditure : 226
GDP : 443
Population : 644
```

```
data = data.dropna()
```

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

Приведем бинарные свойства к int64:

```
data['Status'] = data['Status'].apply(lambda x: x == 'Developed').astype('int64')
```

Закодируем признаки:

LabelEncoder

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

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

CountEncoder

```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in /usr/local/lib/python3.10/dist-packages (2.6.3)
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) (2023
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
```

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

```
data['Population'] = data['Population'].apply(lambda x: f'{x}chel')
```

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

FrequencyEncoder

```
data['Life expectancy'] = data['Life expectancy'].apply(lambda x: f'{x}years')
```

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

```
float_values = ['Country', 'Life expectancy', 'Adult Mortality', 'Alcohol', 'percentage expenditure', 'Hepatitis B', 'Polio', 'Total expe
for col in float_values:
    data[col] = data[col] * 1000
    data[col] = data[col].astype('int64')
    print(col, data[col].unique())
```

```
Total expenditure [ 8160  8180  8130  8520  7870  9200  9420  8330  6730  7430  8700  8790
  8820  7760  7800  8200  6000  5880  5660  5590  5710  5340  5790  5870
  6100  5860  6120  6380  6270  6300  6260  7210  7120  6140  5290  5120
  5360  4200  3820  3360  3240  3540  3310  4260  3300  3380  3390  4370
  3840  4790  4990  5200  5890  6550  7630  6660  6490  6680  6850  6840
  8220  8310  4480  4550  3710  4560  3800  4310  4580  5250  5500  5560
  5400  5940  6250  9360  9500  8780  8530  8490  8450  8570  8320  8390
 11210 11140 11170  1940 11190  1600  1400  1350  1530  1560  1480  1270
  1120  6400  5540  5370  5100  5330  5850  6170  7860  7920  6560  4470
  2820  2880  3160  3600  2910  2850  2800  2680  2620  2510  5690  6700
  4920  5550  6900  5950  6440  6340  6890  6590  6470  6620  6130  1590
  1570  1540  1420  1170  1390  9600  9250  9170  9240  9320  9300  8460
  8290  8119  5450  5610  5810  5900  4760  4400  4450  4390  4530  4380
  4500  3980  4590  4860  4950  4460  4750  4730  4630  4270  3570  3830
  3700  5170  6580  5270  5280  4410  4900  7750  5910  6910  9570  9460
  9940  9710  9580  9640  8580  8370  8280  8500  9400  5410  5840  5640
  6390  4710  4930  5620  4650  5730  4640  8480  8260  8900  8270  8650
  8240  8360  7700  6940  7130  7190  7300  8440  7930  7110  6880  7240
  6780  6610  6410  6670  7900  7410  7230  4960  7170  6770  6630  7540
  8300  8210  6960  1300 11490  9840  7100  4290  4610  4600  4830  4240
  3960  4300  5110  5600  5000  5680  5930  6240  6360  3750  4100  4340
  5700  5180  4690  1450  1670  1780  1820 11200  9830  9750  9560  9540
  3620  3730  3900  3580  3420  3000  3170  2950  2920  7790  7530  7000
  6970  7390  6870  6350  6180  5390  5260  5300  4890  5800  4320  4520
  4660  4720  4820  7200  6930  6640  6760  6110  5820  5920  5670  5960
  6750  6510  5130  4570  3650  9310  9470  9730  9660  9690  9100  7820
  7740  8230  7250  7830  8250  7440  7370  7460  7400  6500  6280  6370
  6420  6830  5780  5770  9140  8950  8710  8840  7720  4120  4220  4280
  5530  9160  7290  6480  5580  6220  6460  4620  3860  6950  6810  6210
  6320  7610  8170  3690  3290  2970  3140  3500  5830  6600  5160  4880
  5190  6860  4490  4160  4230  3680  3740  3610  3350  3480  3280  3870
 11540 11560 11440 11330 11280  1430  1220  9890  9770  3440  3130  3120
  3410  3430  2540  2860  2760  7420  9380  1500  1190  8990  8600  8720
 11300 11160  1990  1930 11250 11400  1180  1340  1520  1370  1620  1150
  1100  3560  4810  4850  4510  8800  9260  9180  9760  9410  9340  8610
  8470  7600  6200  6330  6790  6650  5490  3850  3210  5460  6570  6800
  7320  6450  4670  5970  7560  8100  9150  9780  8400  7890  7810  7280
  4330  4250  2930  2900  2710  2740  2830  2810  3100  2790  2370  2530
  2270  2230  1980  3320  3930  4130  7780  8150  8760  7730  7360  7330
  7380  7350  7490  9220  9280  9270  8890  8109  7910  5210  4870  4700
  4800  7450  8000  8420  8350  8880  9680  9900  9650  4360  4420  3190
  3950  3470  5720  5570  5230  3970  4170  4540  1210 12240 12230 13660
  9980  9000  6150  6290  6990  8830  8910  1900  1860 11700 11790  1870
  9800  8850 11230 11870 14390 11830  7180  7340  7680  7950  7670  7480
  3400  4150  3460  4980 11380  1960 12600 11670  1700  3890  3990 13730
  9520  6740  9950  8930  3770  3630  2870  3790  3230  4210  4970  3780
  5520  5350  4680  5990  6430  6920  5480  5220  5310  4440  4180  6980
  6230  5380  5430  4910  2280  2160  2220  1920  2500  1680  1830  1970
 11100  8430  6820  3670  4000  3660  4110  2610  2700  3200  2940  3260
  2560  7500  6310  4780  4940  7690  9810  1490  1330  5470  3940  3910
  3250  6190  9550  9740  1440  9620  9670  7710  7660  1200  7220  5440
  5240  7840 11120  9720  3370  3590  2640 11900 11590 11240 11980  1320
 13130  1290  5420  7470  7570  7770  9390  9480  7990  3760  7970  3180
  8189 11930 11970 11800  3720  5980  3810  3920  3490  3550  3510   760
   920   740  6520  7260  7150  6540  5630  5320  2120  1880  3340  7580
  9860  8680  8740  8550  8630 11150  5650  3520  6690  6530  6160]
GDP [584259 612696 631744 ...  57348 548587 547358]
Population [1000 2000]
```

```
data.head()
```

	Country	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	P
0	0	0	6	263000	62	10	71279	65000	(
1	0	0	3	271000	64	10	73523	62000	5
2	0	0	3	268000	66	10	73219	64000	6
3	0	0	1	272000	69	10	78184	67000	6

Далее: Посмотреть рекомендованные графики

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1659 entries, 0 to 2927
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Country                 1659 non-null   int64
 1   Status                  1659 non-null   int64
 2   Life expectancy         1659 non-null   int64
 3   Adult Mortality         1659 non-null   int64
 4   infant deaths           1659 non-null   int64
 5   Alcohol                 1659 non-null   int64
 6   percentage expenditure  1659 non-null   int64
 7   Hepatitis B             1659 non-null   int64
 8   Polio                   1659 non-null   int64
 9   Total expenditure       1659 non-null   int64
 10  GDP                     1659 non-null   int64
 11  Population              1659 non-null   int64
dtypes: int64(12)
memory usage: 168.5 KB
```

Разделение выборки

```
data.describe()
```

	Country	Status	Life expectancy	Adult Mortality	infant deaths	Alcoho
count	1659.000000	1659.000000	1659.000000	1659.000000	1659.000000	1659.00000
mean	66623.869801	0.145871	4.657625	168539.481615	32.734780	4513.93128
std	39160.608445	0.353083	3.599988	125106.473746	120.504952	4025.21261
min	0.000000	0.000000	0.000000	1000.000000	0.000000	10.00000
25%	33000.000000	0.000000	2.000000	77000.000000	1.000000	810.00000
50%	67000.000000	0.000000	4.000000	149000.000000	3.000000	3750.00000
75%	100000.000000	0.000000	6.000000	228500.000000	23.000000	7325.00000

В качестве целевого признака возьмем признак `price`.

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

```
X_ALL
```

	Country	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Polio	exp
0	0	0	263000	62	10	71279	65000	6000	
1	0	0	271000	64	10	73523	62000	58000	
2	0	0	268000	66	10	73219	64000	62000	
3	0	0	272000	69	10	78184	67000	67000	
4	0	0	275000	71	10	7097	68000	68000	
...	...	...	...	...	...	...	...	...	
2923	133000	0	723000	27	4360	0	68000	67000	
2924	133000	0	715000	26	4059	0	7000	7000	
2925	133000	0	73000	25	4430	0	73000	73000	
2926	133000	0	686000	25	1720	0	76000	76000	
2927	133000	0	665000	24	1680	0	79000	78000	

Далее: Посмотреть рекомендованные графики

```
Y
```

```
0        6
1        3
```

```
2       3
3       1
4       4
       ..
2923    0
2924    1
2925    1
2926    1
2927    2
Name: Life expectancy, Length: 1659, 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['Life expectancy'],
                                                    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
```

```
((1327, 11), (332, 11))
```

Масштабирование признаков

Масштабирование на основе Z-оценки

```
x_col_list = ['Country', 'Adult Mortality', 'Population']
```

```
# 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
```

	Country	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	I
0	-1.701811	-0.413259	0.755269	0.242928	-1.119267	-0.355957	-0.552607	-3.4
1	-1.701811	-0.413259	0.819233	0.259530	-1.119267	-0.354678	-0.669652	-1.1
2	-1.701811	-0.413259	0.795247	0.276132	-1.119267	-0.354851	-0.591622	-0.9
3	-1.701811	-0.413259	0.827229	0.301035	-1.119267	-0.352020	-0.474577	-0.7
4	-1.701811	-0.413259	0.851216	0.317636	-1.119267	-0.392550	-0.435562	-0.6
...	...	...	...	...	...	...	...	
1654	1.695483	-0.413259	4.433245	-0.047604	-0.038253	-0.396597	-0.435562	-0.7
1655	1.695483	-0.413259	4.369281	-0.055905	-0.113055	-0.396597	-2.815476	-3.4
1656	1.695483	-0.413259	-0.763896	-0.064206	-0.020858	-0.396597	-0.240487	-0.4
1657	1.695483	-0.413259	4.137408	-0.064206	-0.694317	-0.396597	-0.123442	-0.3
1658	1.695483	-0.413259	3.969501	-0.072507	-0.704257	-0.396597	-0.006397	-0.2

Далее: Посмотреть рекомендованные графики

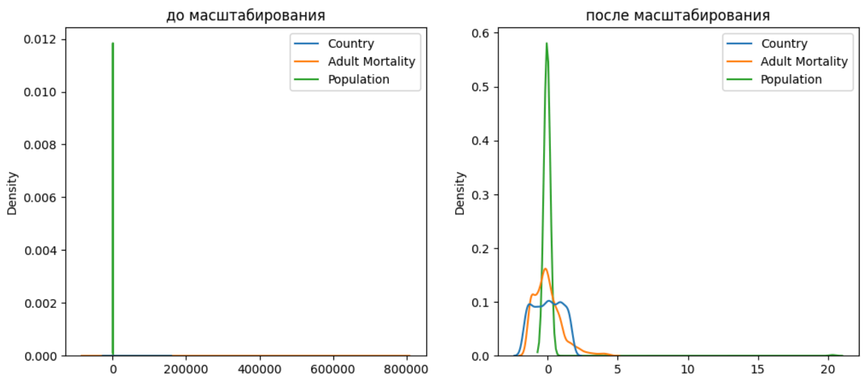
```
data_cs11_scaled.describe()
```

	Country	Status	Adult Mortality	infant deaths	Alcohol	percenta expenditu
count	1659.000000	1.659000e+03	1.659000e+03	1.659000e+03	1.659000e+03	1659.0000
mean	0.000000	1.713183e-17	-1.199228e-16	2.998071e-17	-1.713183e-17	0.0000
std	1.000302	1.000302e+00	1.000302e+00	1.000302e+00	1.000302e+00	1.0003
min	-1.701811	-4.132594e-01	-1.339579e+00	-2.717287e-01	-1.119267e+00	-0.3965
25%	-0.858873	-4.132594e-01	-7.319132e-01	-2.634278e-01	-9.204602e-01	-0.3750
		-4.132594e-		-2.468259e-		

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)
```

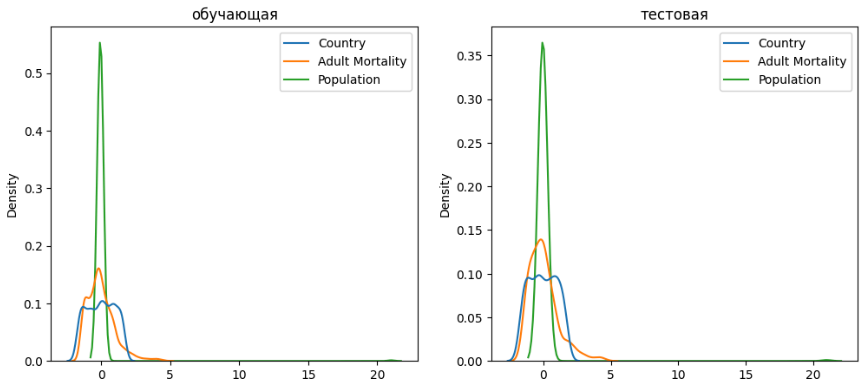
```
data_cs12_scaled_train.describe()
```

	Country	Status	Adult Mortality	infant deaths	Alcohol	percen expendi
count	1.327000e+03	1.327000e+03	1.327000e+03	1.327000e+03	1.327000e+03	1.327000
mean	-3.480428e-17	-2.944977e-17	1.204763e-16	-1.874077e-17	-5.220642e-17	-4.0158
std	1.000377e+00	1.000377e+00	1.000377e+00	1.000377e+00	1.000377e+00	1.000377
min	-1.706789e+00	-4.100412e-01	-1.359299e+00	-2.693859e-01	-1.106328e+00	-3.9723
25%	-8.631901e-01	-4.100412e-01	-7.410782e-01	-2.607726e-01	-9.327080e-01	-3.7631
		-4.100412e-		-2.435462e-		-3.1527

```
data_cs12_scaled_test.describe()
```

	Country	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatit
count	332.000000	332.000000	332.000000	332.000000	332.000000	332.000000	332.000(
mean	-0.018206	0.027579	0.017732	0.062794	0.053856	-0.001419	-0.046
std	1.005294	1.028775	1.085543	1.176481	0.990634	1.006000	0.997
min	-1.706789	-0.410041	-1.359299	-0.269386	-1.106328	-0.397181	-3.018(
25%	-0.895145	-0.410041	-0.743112	-0.260773	-0.839077	-0.372494	-0.288
50%	-0.019592	-0.410041	-0.147261	-0.243546	-0.100570	-0.311235	0.374
75%	0.855961	-0.410041	0.456725	-0.054055	0.766292	-0.115907	0.647(

```
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()
```

	Country	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	I
count	1659.000000	1659.000000	1659.000000	1659.000000	1659.000000	1659.000000	16
mean	-0.001072	0.001937	0.000604	0.000912	0.002433	-0.000026	
std	0.294441	0.353083	0.173278	0.075316	0.225376	0.092528	
min	-0.502003	-0.143934	-0.231445	-0.019547	-0.249747	-0.036712	
25%	-0.253883	-0.143934	-0.126182	-0.018922	-0.204954	-0.034723	
50%	0.001756	-0.143934	-0.026459	-0.017672	-0.040340	-0.029059	
75%	0.249877	-0.143934	0.083652	-0.005172	0.159828	-0.009886	

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

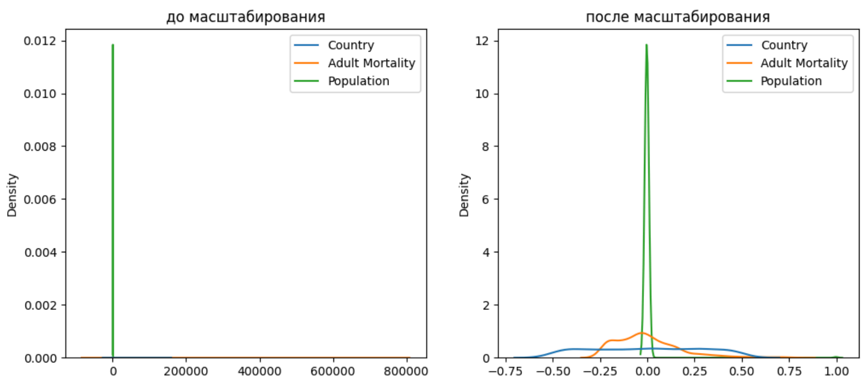
```
data_cs22_scaled_train.describe()
```

	Country	Status	Adult Mortality	infant deaths	Alcohol	percenta expenditu
count	1.327000e+03	1.327000e+03	1.327000e+03	1.327000e+03	1.327000e+03	1.327000e+
mean	-1.740214e-17	-2.175267e-17	2.225466e-17	-1.673283e-18	-1.472489e-17	-4.01587
std	2.942322e-01	3.511548e-01	1.703320e-01	7.259011e-02	2.258289e-01	9.245215e-
min	-5.020029e-01	-1.439337e-01	-2.314449e-01	-1.954738e-02	-2.497468e-01	-3.67116
25%	-2.538826e-01	-1.439337e-01	-1.261818e-01	-1.892238e-02	-2.105531e-01	-3.47783
		-1.439337e-	-2.645878e-	-1.767238e-	-4.593940e-	-2.91373

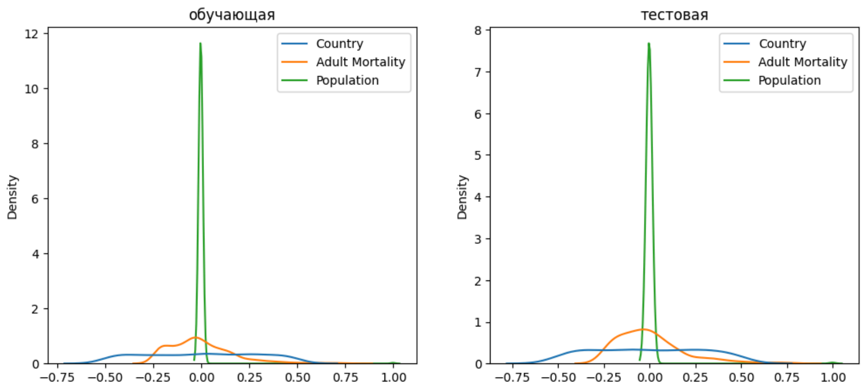
```
data_cs22_scaled_test.describe()
```

	Country	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepati
count	332.000000	332.000000	332.000000	332.000000	332.000000	332.000000	332.000
mean	-0.005355	0.009681	0.003019	0.004557	0.012158	-0.000131	-0.012
std	0.295678	0.361123	0.184833	0.085369	0.223630	0.092972	0.263
min	-0.502003	-0.143934	-0.231445	-0.019547	-0.249747	-0.036706	-0.797
25%	-0.263281	-0.143934	-0.126528	-0.018922	-0.189416	-0.034425	-0.076
50%	-0.005762	-0.143934	-0.025074	-0.017672	-0.022703	-0.028763	0.098
75%	0.251756	-0.143934	0.077766	-0.003922	0.172986	-0.010712	0.171

```
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')
```



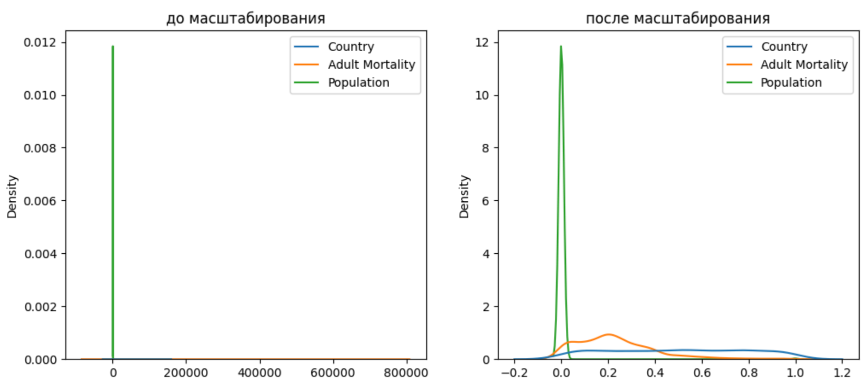
MinMax масштабирование

```
# 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()
```

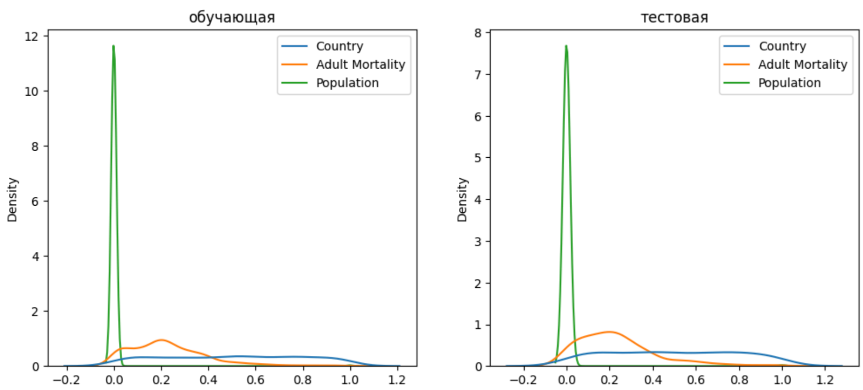
	Country	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	I
count	1659.000000	1659.000000	1659.000000	1659.000000	1659.000000	1659.000000	16
mean	0.500931	0.145871	0.232049	0.020459	0.252180	0.036685	
std	0.294441	0.353083	0.173278	0.075316	0.225376	0.092528	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.248120	0.000000	0.105263	0.000625	0.044793	0.001989	
50%	0.503759	0.000000	0.204986	0.001875	0.209406	0.007653	
75%	0.751880	0.000000	0.315097	0.014375	0.409574	0.026825	

```
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()
```

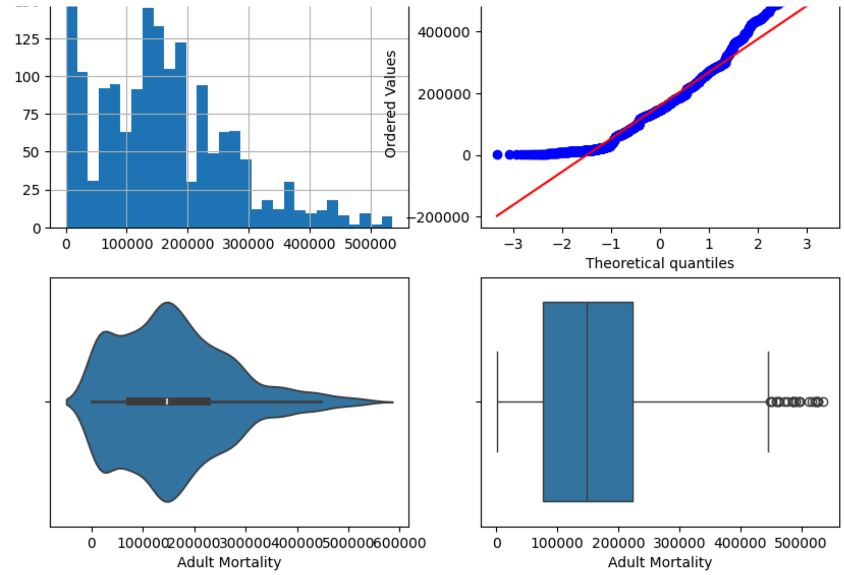
```
from enum import Enum
class OutlierBoundaryType(Enum):
    SIGMA = 1
    QUANTILE = 2
    IRQ = 3
```

```
# 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

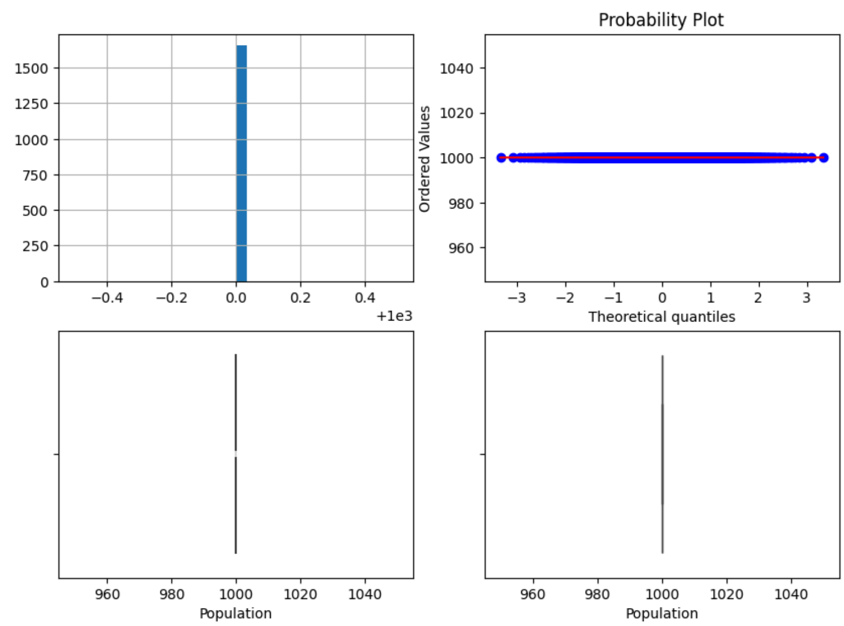
Воспользуемся методом OutlierBoundaryType.SIGMA:

```
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-271-766c933c159f>:4: MatplotlibDeprecationWarning: Auto-removal of c
  plt.subplot(2, 2, 1)
```

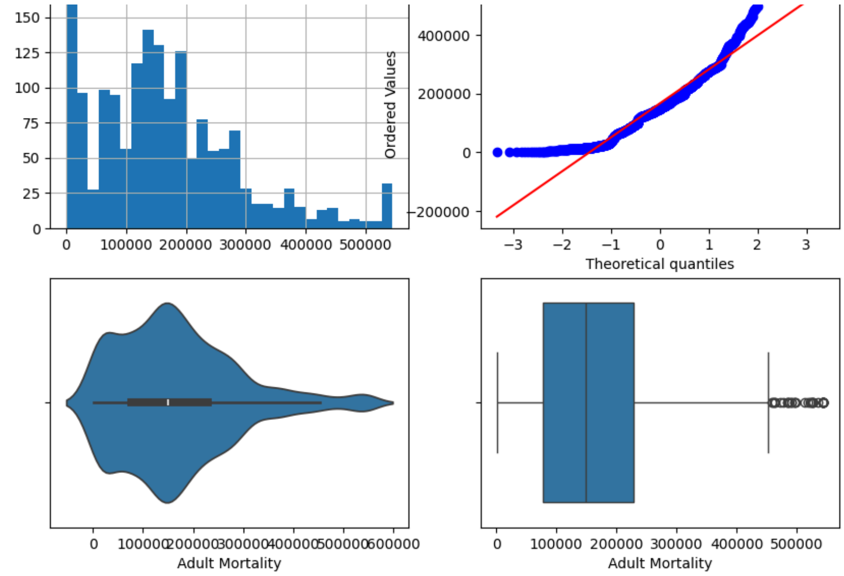
Поле-Population, метод-OutlierBoundaryType.SIGMA, строк-1655



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)
```




```
&lt;ipython-input-271-766c933c159f&gt;:4: MatplotlibDeprecationWarning: Auto-removal of c
  plt.subplot(2, 2, 1)
```

Поле-Population, метод-OutlierBoundaryType.SIGMA