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)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

Отчёт к лабораторным работам по курсу

«Методы машинного обучения»

Лабораторная работа №3 «Обработка признаков (часть 2)»

Выполнил:

студент(ка) группы ИУ5И-21М Лю Бэйбэй

подпись, дата

Проверил:

к.т.н., доц., Виноградовой М.В.

подпись, дата

Москва, 2022 г.

1. описание задания

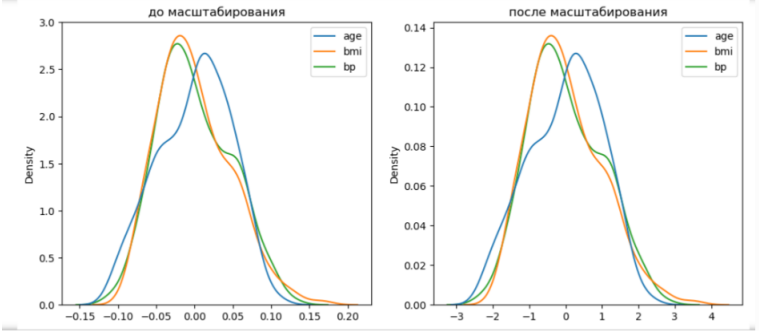
1. Выбрать один или несколько наборов данных (датасетов) для решения следующих задач. Каждая задача может быть решена на отдельном датасете, или несколько задач могут быть решены на одном датасете. Просьба не использовать датасет, на котором данная задача решалась в лекции.
2. Для выбранного датасета (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).

2. Текст программы и экранные формы с примерами выполнения программы.

масштабирование признаков.
Масштабирование данных на основе Z-оценки

```
# 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(['age', 'bmi', 'bp'], data, data_cs11_scaled, 'до масштабирования', 'посл
```

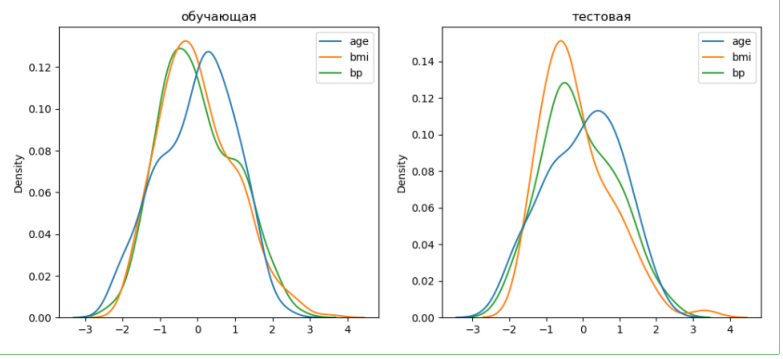


```
# Обучаем StandardScaler на обучающей выборке
# и масштабируем обучающую и тестовую выборки
cs12 = StandardScaler()
cs12.fit(X_train)
data_cs12_scaled_train_temp = cs12.transform(X_train)
data_cs12_scaled_test_temp = cs12.transform(X_test)
# формируем DataFrame на основе массива
data_cs12_scaled_train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

	age	sex	bmi	bp	s1	s2	s3	
count	3.530000e+02	3.530000e+02	3.530000e+02	3.530000e+02	3.530000e+02	3.530000e+02	3.530000e+02	3.530000e+
mean	6.250902e-17	5.912803e-17	-2.940676e-17	2.327380e-17	-5.032172e-18	7.233748e-18	1.320945e-17	-4.088640e-
std	1.001419e+00	1.001419e+00	1.001419e+00	1.001419e+00	1.001419e+00	1.001419e+00	1.001419e+00	1.001419e+
min	-2.262150e+00	-9.262261e-01	-1.915758e+00	-2.376419e+00	-2.685044e+00	-2.473851e+00	-2.166235e+00	-1.631226e+
25%	-7.275311e-01	-9.262261e-01	-7.265375e-01	-7.148018e-01	-7.160882e-01	-6.376931e-01	-7.511056e-01	-8.554566e-
50%	1.165093e-01	-9.262261e-01	-1.431463e-01	-1.368479e-01	-1.080284e-01	-1.254316e-01	-1.221591e-01	-7.968673e-
75%	7.303569e-01	1.079650e+00	6.646260e-01	7.300830e-01	5.579418e-01	6.196760e-01	5.854058e-01	6.960831e+
max	2.341707e+00	1.079650e+00	3.514267e+00	2.752922e+00	3.192868e+00	4.205506e+00	3.808757e+00	3.868982e+

```
ления для обучающей и тестовой выборки немного отличают
, 'bmi', 'bp'], data_cs12_scaled_train, data_cs12_scaled_test, 'обучающая', 'тестовая')
```



Масштабирование "Mean Normalisation"

```
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()
```

	age	sex	bmi	bp	s1	s2	s3	
count	442.000000	442.000000	442.000000	442.000000	442.000000	442.000000	442.000000	442
mean	0.000609	0.006569	-0.006701	-0.003481	-0.002909	-0.002648	0.003087	-0
std	0.218484	0.499561	0.182567	0.194806	0.170483	0.151460	0.170187	0
min	-0.491360	-0.461756	-0.352808	-0.463299	-0.456802	-0.370373	-0.362550	-0
25%	-0.170526	-0.461756	-0.137932	-0.153440	-0.125522	-0.099207	-0.122419	-0
50%	0.025307	-0.461756	-0.034626	-0.026679	-0.018379	-0.014795	-0.020445	-0
75%	0.175307	0.538244	0.113101	0.142335	0.098616	0.092277	0.107844	0
max	0.508640	0.538244	0.647192	0.536701	0.548124	0.629627	0.650608	0

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

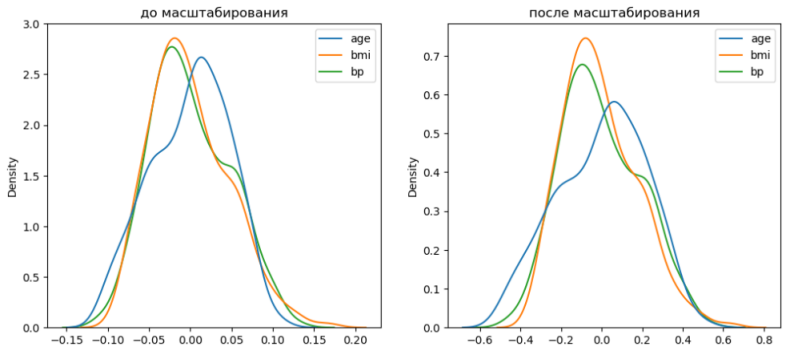
```
data_cs22_scaled_train.describe()
```

	age	sex	bmi	bp	s1	s2	
count	3.530000e+02	3.530000e+02	3.530000e+02	3.530000e+02	3.530000e+02	3.530000e+02	3.
mean	1.529309e-17	1.125949e-16	1.533240e-18	-1.258043e-18	-4.324523e-19	7.076492e-19	
std	2.175175e-01	4.992429e-01	1.844226e-01	1.952335e-01	1.703699e-01	1.499275e-01	1.
min	-4.913598e-01	-4.617564e-01	-3.528083e-01	-4.632992e-01	-4.568024e-01	-3.703726e-01	
25%	-1.580264e-01	-4.617564e-01	-1.338000e-01	-1.393555e-01	-1.218270e-01	-9.547222e-02	
50%	2.530689e-02	-4.617564e-01	-2.636200e-02	-2.667943e-02	-1.837871e-02	-1.877899e-02	
75%	1.586402e-01	5.382436e-01	1.223983e-01	1.423347e-01	9.492178e-02	9.277479e-02	9.
max	5.086402e-01	5.382436e-01	6.471917e-01	5.367008e-01	5.431976e-01	6.296274e-01	6.

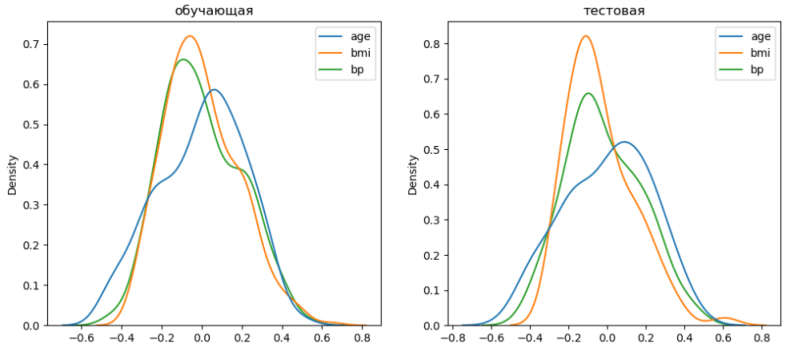
```
data_cs22_scaled_test.describe()
```

	age	sex	bmi	bp	s1	s2	s3	s4
count	89.000000	89.000000	89.000000	89.000000	89.000000	89.000000	89.000000	89.000000
mean	0.003022	0.032626	-0.033280	-0.017290	-0.014449	-0.013150	0.015333	-0.022745
std	0.223507	0.502801	0.173467	0.193576	0.171407	0.157823	0.180543	0.181658
min	-0.474693	-0.461756	-0.286693	-0.406961	-0.363206	-0.361408	-0.270445	-0.296575
25%	-0.174693	-0.461756	-0.154461	-0.153440	-0.136605	-0.143281	-0.112550	-0.155532
50%	0.025307	-0.461756	-0.080081	-0.054848	-0.013453	-0.003839	-0.020445	-0.014488
75%	0.191974	0.538244	0.068679	0.114166	0.104774	0.090783	0.124292	0.126556
max	0.441974	0.538244	0.610002	0.438109	0.548124	0.493173	0.650608	0.408643

```
draw_kde(['age', 'bmi', 'bp'], data, data_cs21_scaled, 'до масштабирова
```




```
draw_kde(['age', 'bmi', 'bp'], data_cs22_scaled_train, data_cs22_scaled_test, 'об
```



MinMax-масштабирование

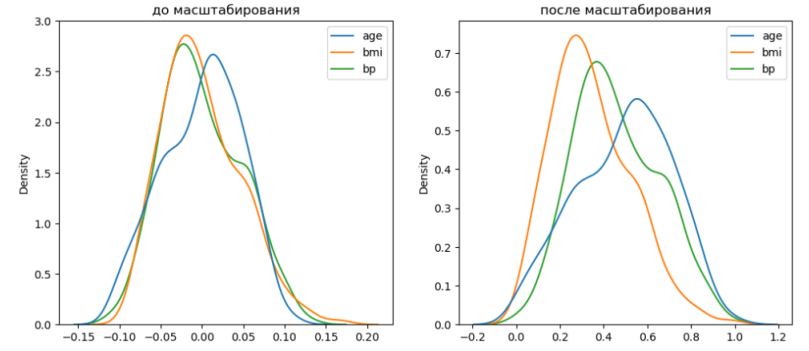
```
# Обучаем StandardScaler на всей выборке и масштабируе
cs31 = MinMaxScaler()
data_cs31_scaled_temp = cs31.fit_transform(X_ALL)
# формируем DataFrame на основе массива
data_cs31_scaled = arr_to_df(data_cs31_scaled_temp)
data_cs31_scaled.describe()
```

	age	sex	bmi	bp	s1	s2	s3	
count	442.000000	442.000000	442.000000	442.000000	442.000000	442.000000	442.000000	442
mean	0.491968	0.468326	0.346107	0.459818	0.451668	0.367725	0.360889	0
std	0.218484	0.499561	0.182567	0.194806	0.169647	0.151460	0.167977	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.320833	0.000000	0.214876	0.309859	0.329657	0.271165	0.237013	0
50%	0.516667	0.000000	0.318182	0.436620	0.436275	0.355578	0.337662	0
75%	0.666667	1.000000	0.465909	0.605634	0.552696	0.462649	0.464286	0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1

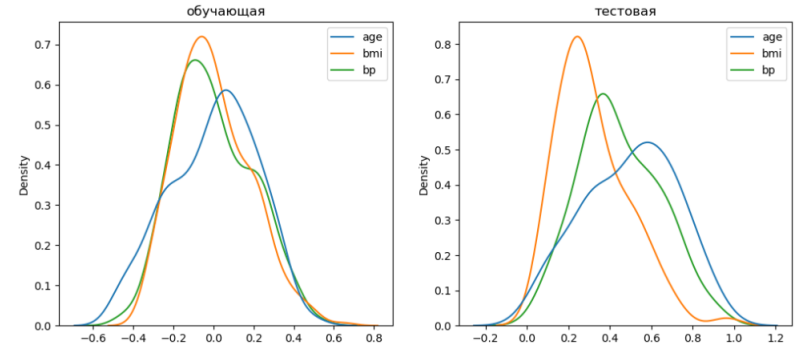
```
cs32 = MinMaxScaler()
cs32.fit(X_train)
data_cs32_scaled_train_temp = cs32.transform(X_train)
data_cs32_scaled_test_temp = cs32.transform(X_test)
# формируем DataFrame на основе массива
data_cs32_scaled_train = arr_to_df(data_cs32_scaled_train_temp)
```

data_cs32_scaled_test = arr_to_df(data_cs32_scaled_test_temp)

```
draw_kde(['age', 'bmi', 'bp'], data, data_cs31_scaled, 'до масштабирова
```

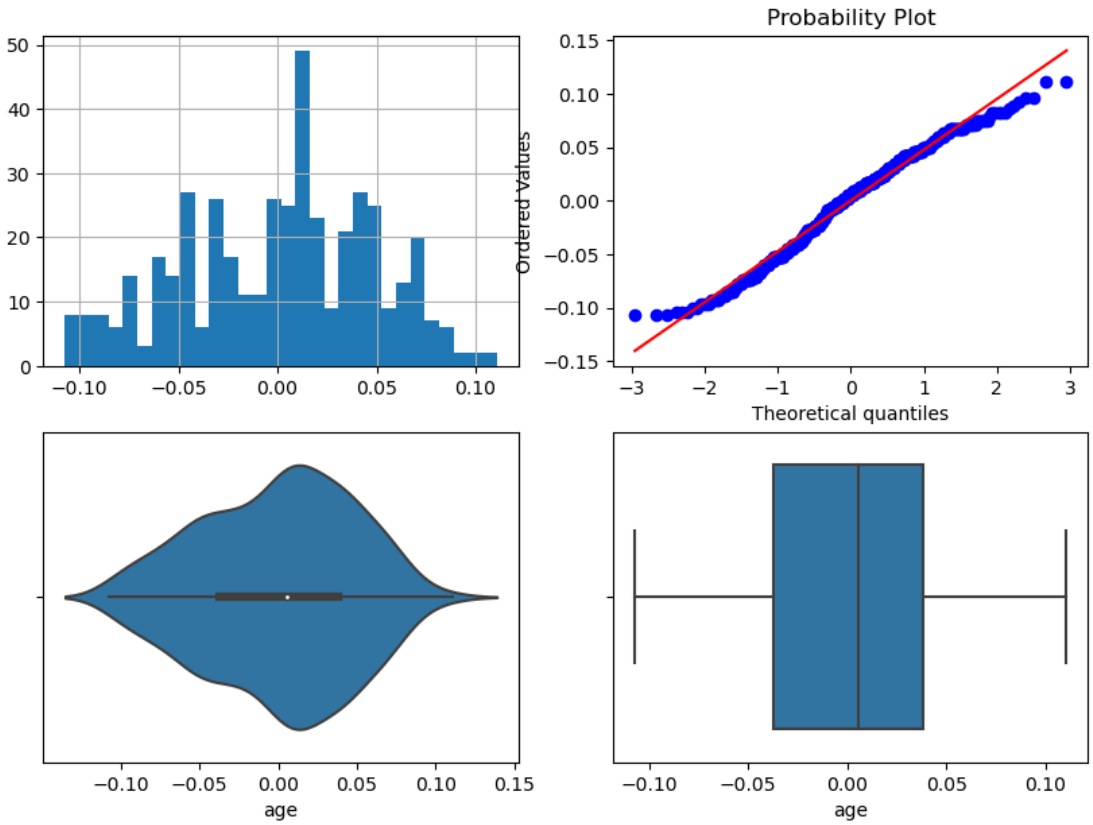



```
draw_kde(['age', 'bmi', 'bp'], data_cs22_scaled_train, data_cs32_scaled_test, 'обу
```

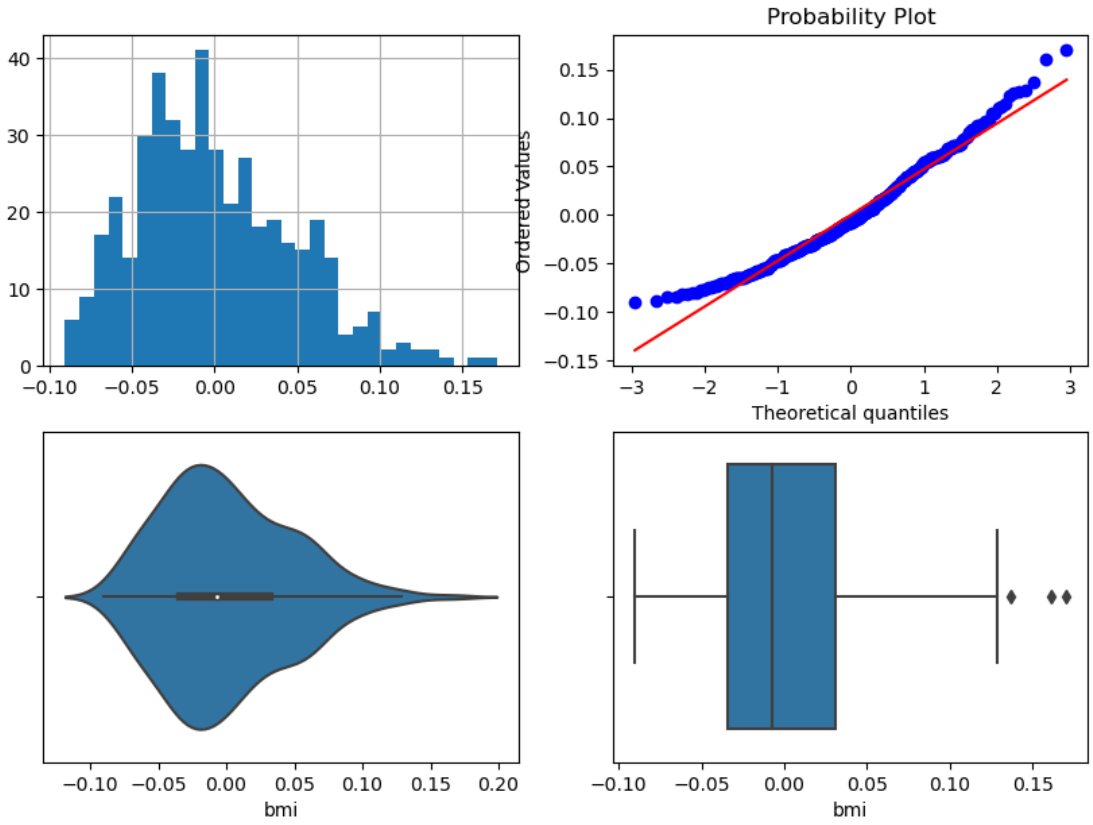


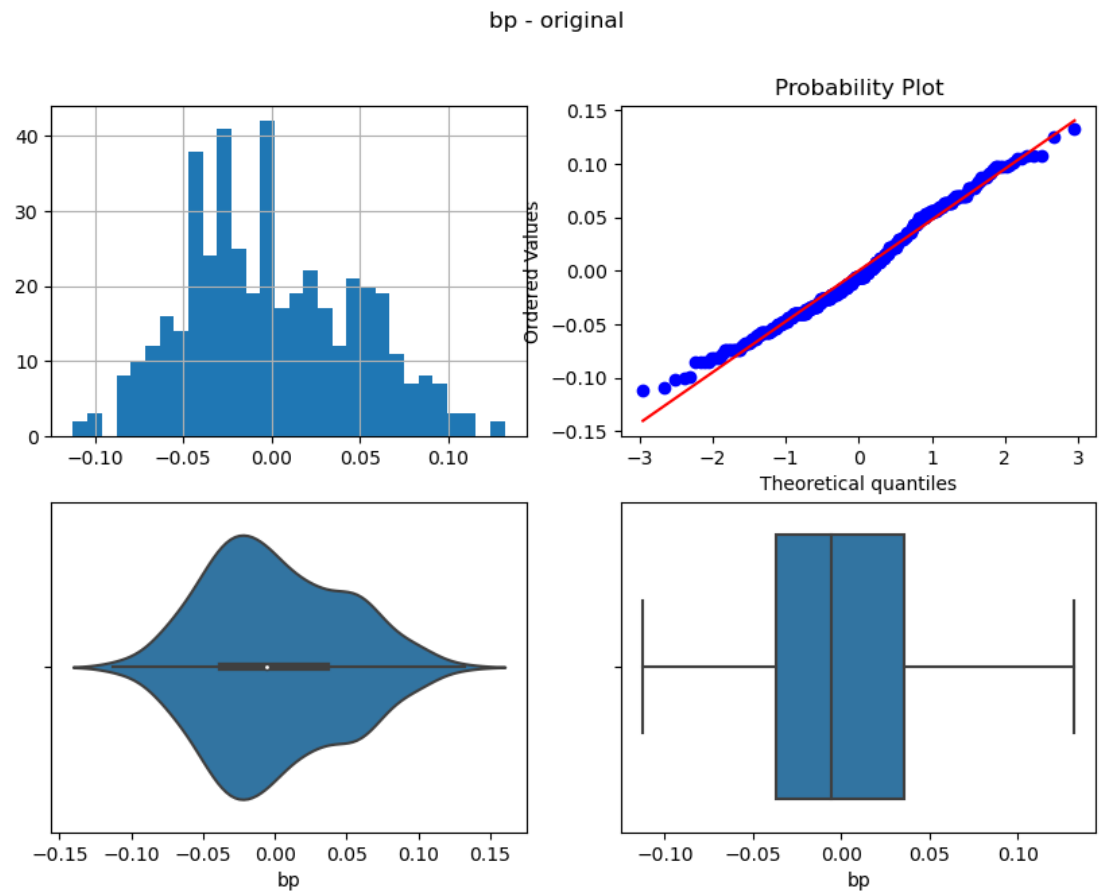
обработку выбросов для числовых признаков (по одному способу для удаления выбросов и для замены выбросов);
Необработанные данные:

age - original



bmi - original

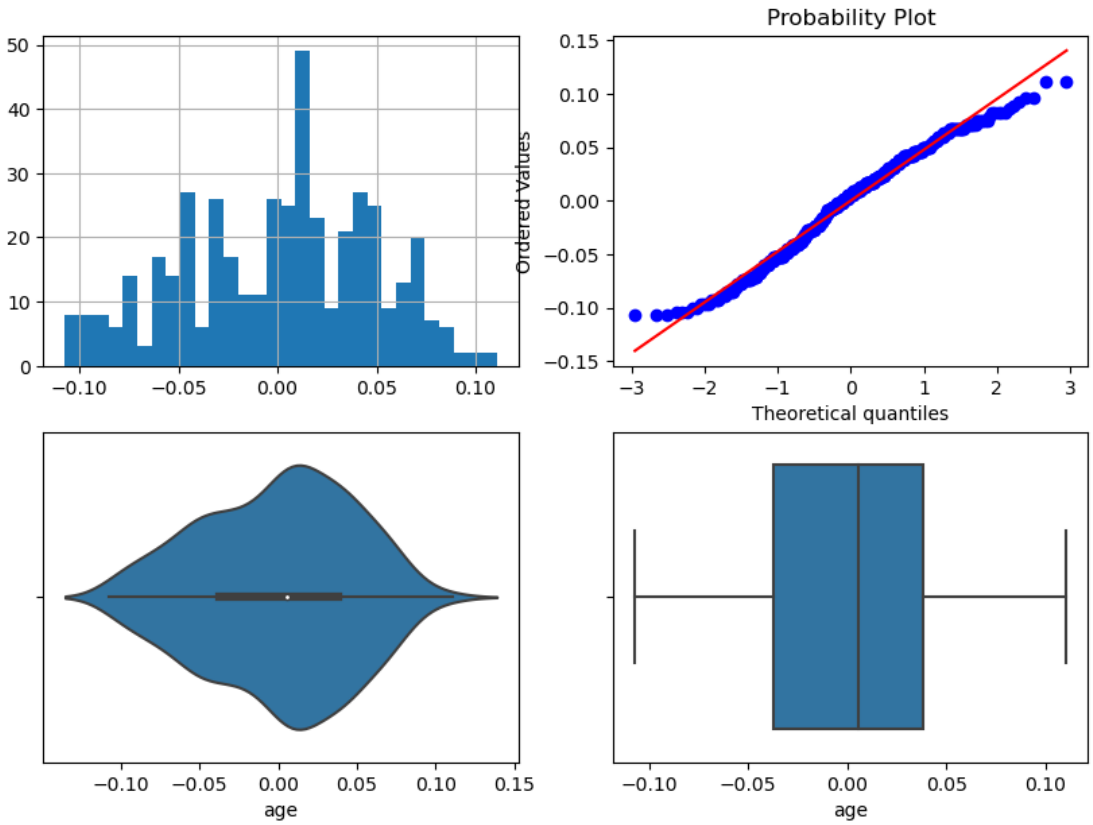




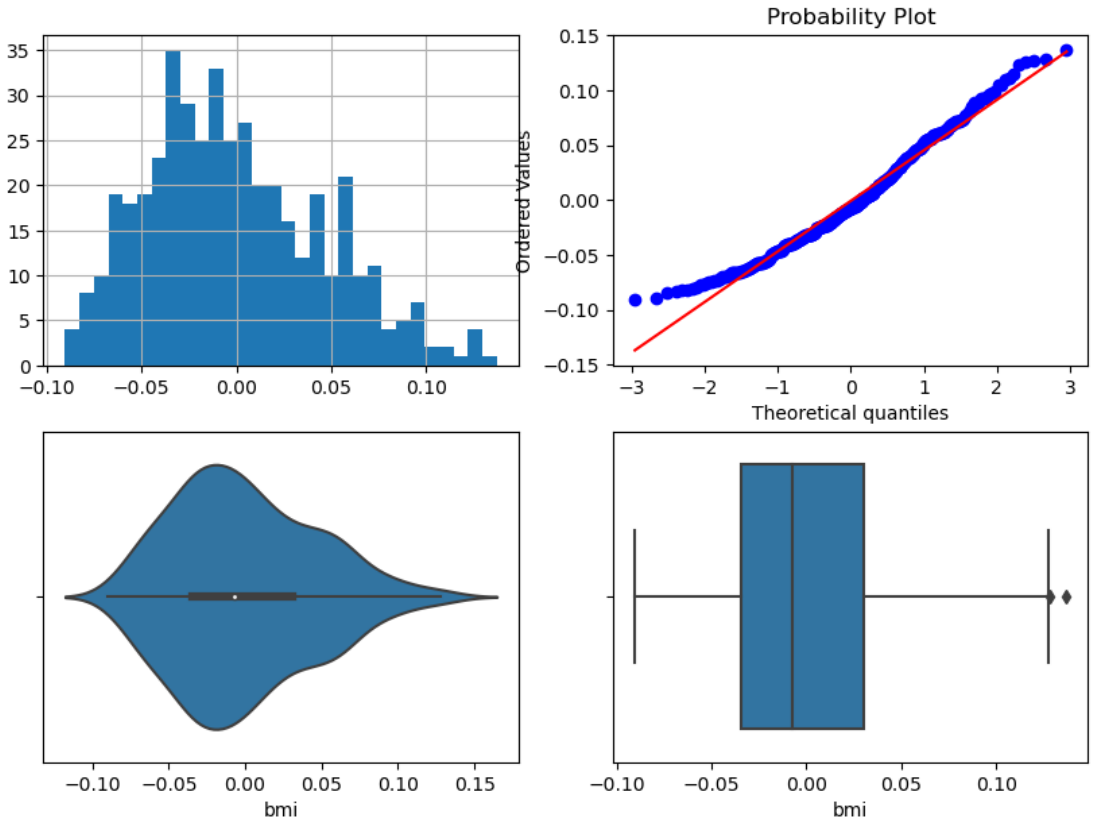
Удаление выбросов

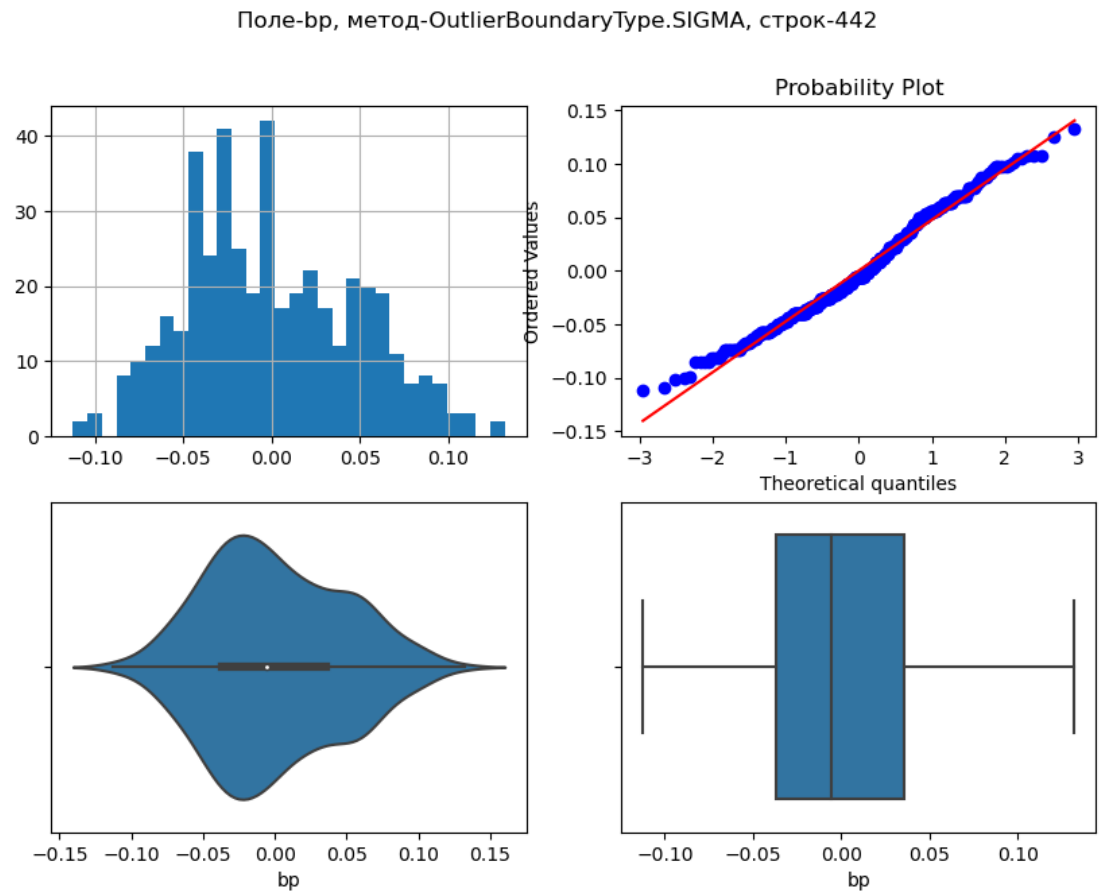
```
for col in x_col_list:
    for obt in OutlierBoundaryType:
        # Вычисление верхней и нижней границы
        lower_boundary, upper_boundary = get_outlier_boundaries(data, col, obt)
        # Флаги для удаления выбросов
        outliers_temp = np.where(data[col] > upper_boundary, True,
                                 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, obt,
data_trimmed.shape[0])
        diagnostic_plots(data_trimmed, col, title)
```

Поле-age, метод-OutlierBoundaryType.SIGMA, строк-442



Поле-bmi, метод-OutlierBoundaryType.SIGMA, строк-440

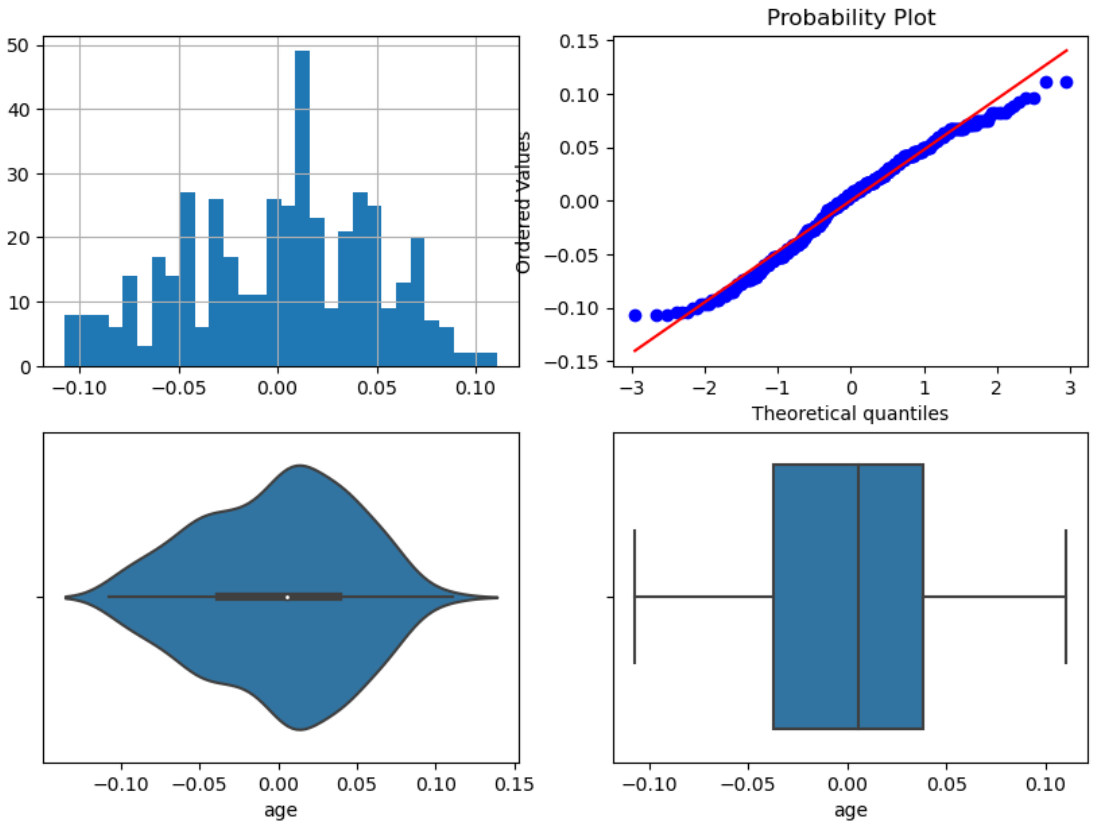




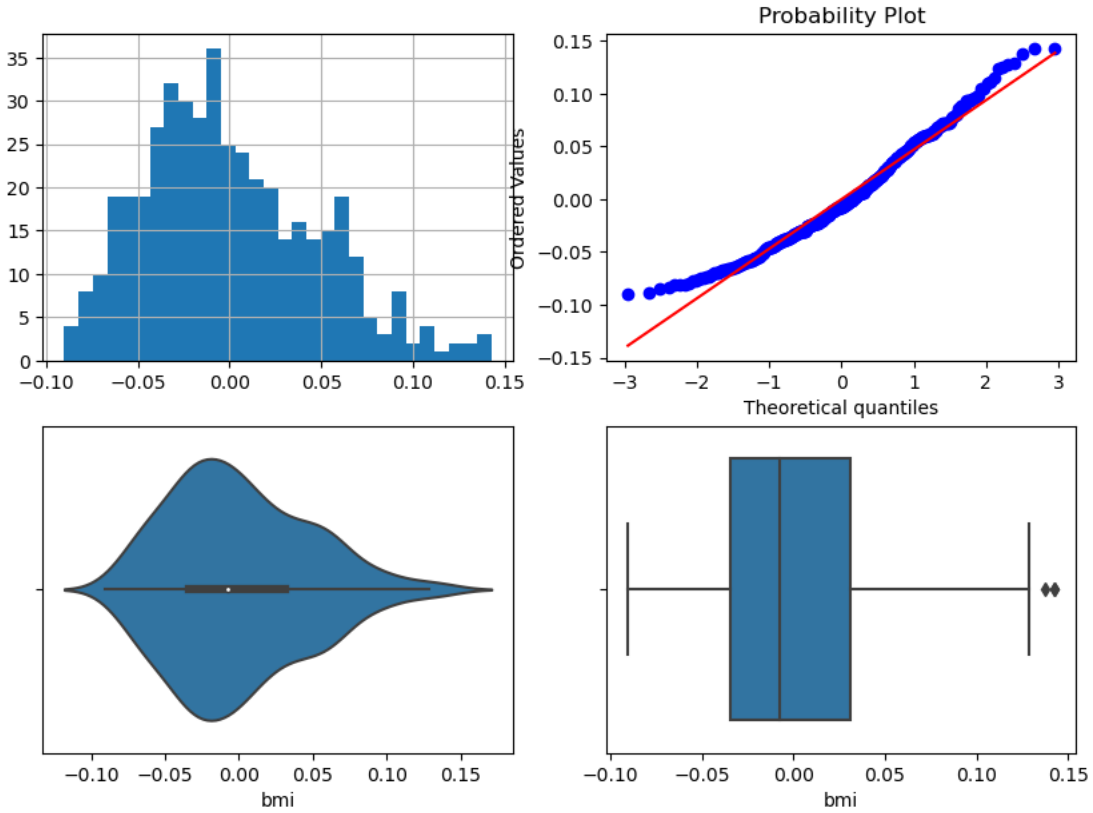
замены выбросов:

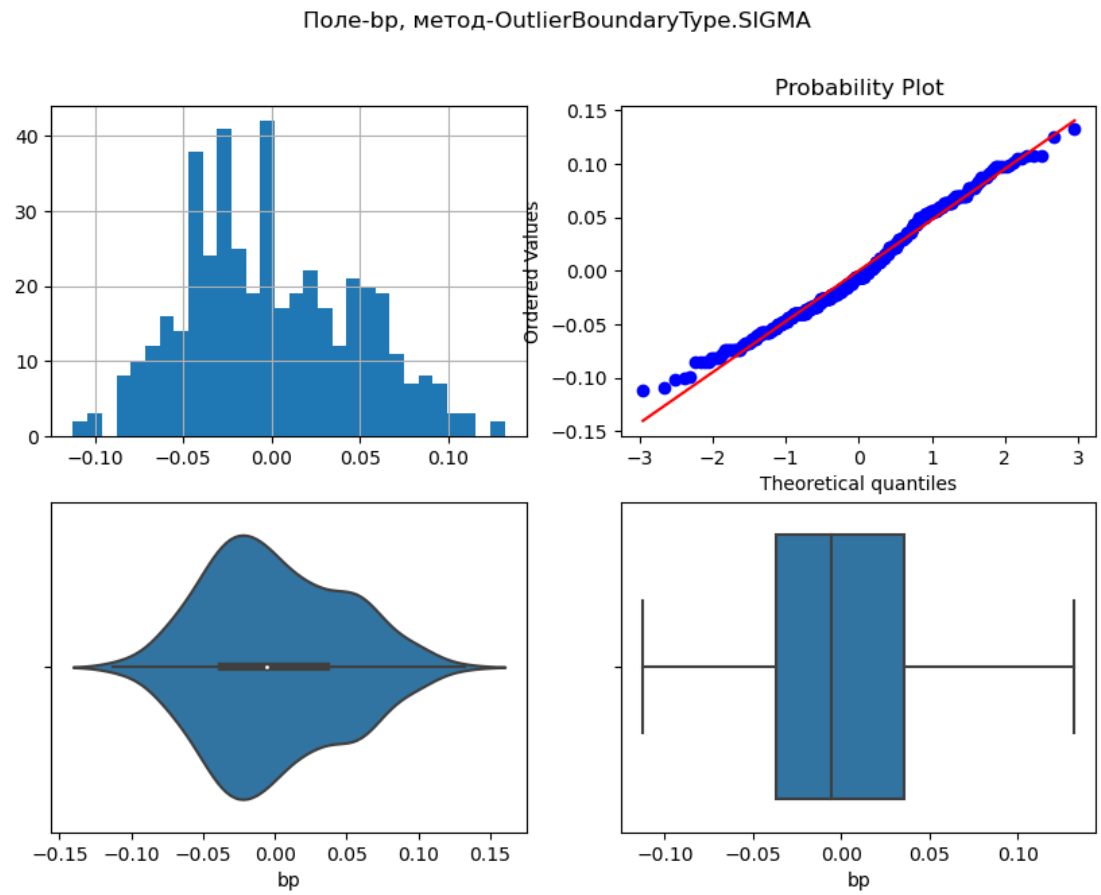
```
for col in x_col_list:
    for obt in OutlierBoundaryType:
        # Вычисление верхней и нижней границы
        lower_boundary, upper_boundary = get_outlier_boundaries(data, col, obt)
        # Изменение данных
        data[col] = np.where(data[col] > upper_boundary, upper_boundary,
           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

Поле-age, метод-OutlierBoundaryType.SIGMA



Поле-bmi, метод-OutlierBoundaryType.SIGMA





обработку по крайней мере одного нестандартного признака (который не является числовым или категориальным);

```
data3.head()
```

	Time	0	1	2	3	4	5	6	7	8	...
0	2008-07-19 11:55:00	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	...
1	2008-07-19 12:32:00	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	...
2	2008-07-19 13:17:00	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.4436	...
3	2008-07-19 14:43:00	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.4882	...
4	2008-07-19 15:22:00	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5031	...

5 rows × 592 columns

```
data3.dtypes
```

```
Time          object
0            float64
1            float64
2            float64
3            float64
              ...
586          float64
587          float64
588          float64
589          float64
Pass/Fail      int64
Length: 592, dtype: object
```

```
data3['dt'] = data3.apply(lambda x: pd.to_datetime(x['Time'], format='%Y/%m/%d %H:%M:%S'), axis=1)
# День
data3['day'] = data3['dt'].dt.day
# Месяц
data3['month'] = data3['dt'].dt.month
# Год
data3['year'] = data3['dt'].dt.year
# Часы
data3['hour'] = data3['dt'].dt.hour
#Минуты
data3['minute'] = data3['dt'].dt.minute
#Секунды
data3['second'] = data3['dt'].dt.second
#День недели
data3['dayofweek'] = data3['dt'].dt.dayofweek
```

```
data3.head()
```

	Time	0	1	2	3	4	5	6	7	8	...
0	2008-07-19 11:55:00	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	...
1	2008-07-19 12:32:00	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	...
2	2008-07-19 13:17:00	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.4436	...
3	2008-07-19 14:43:00	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.4882	...
4	2008-07-19 15:22:00	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5031	...

```
# Создадим масштабируемые признаки для дальней
dt_features = ['year', 'day', 'month', 'hour', 'minute', 'second', 'dayofweek']
dt_features_scaled = []
for f in dt_features:
    f_new = str(f + '_scaled')
    dt_features_scaled.append(f_new)
    data3[f_new] = MinMaxScaler().fit_transform(data3[[f]])
dt_features_scaled
```

```
['year_scaled',
 'day_scaled',
 'month_scaled',
 'hour_scaled',
 'minute_scaled',
 'second_scaled',
 'dayofweek_scaled']
```

```
data3.head()
```

	Time	0	1	2	3	4	5	6	7	8	...
0	2008-07-19 11:55:00	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	...
1	2008-07-19 12:32:00	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	...
2	2008-07-19 13:17:00	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.4436	...
3	2008-07-19 14:43:00	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.4882	...
4	2008-07-19 15:22:00	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5031	...

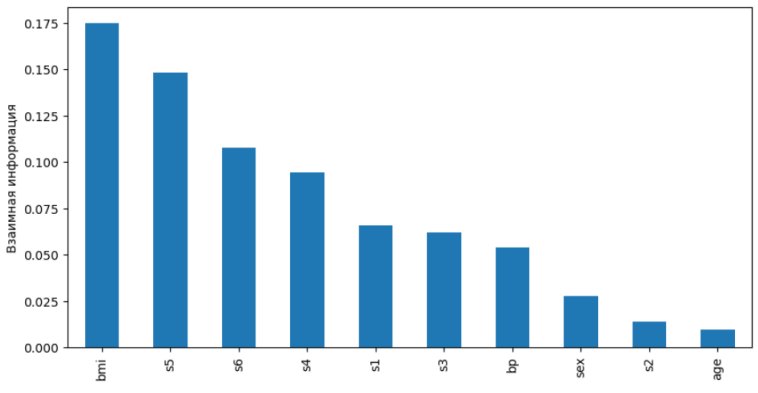
5 rows × 607 columns

отбор признаков:

```
wine = load_wine()
wine_X = wine.data
wine_y = wine.target
wine_feature_names = wine['feature_names']
wine_x_df = pd.DataFrame(data=wine['data'], columns=wine['feature_names'])
diabetes = load_diabetes()
diabetes_X = diabetes.data
diabetes_y = diabetes.target
diabetes_feature_names = diabetes['feature_names']
diabetes_x_df = pd.DataFrame(data=diabetes['data'], columns=diabetes['feature_names'])
```

```
mi = mutual_info_regression(diabetes_X, diabetes_y)
mi = pd.Series(mi)
mi.index = diabetes_feature_names
mi.sort_values(ascending=False).plot.bar(figsize=(10,5))
plt.ylabel('Взаимная информация')
```

```
Text(0, 0.5, 'Взаимная информация')
```



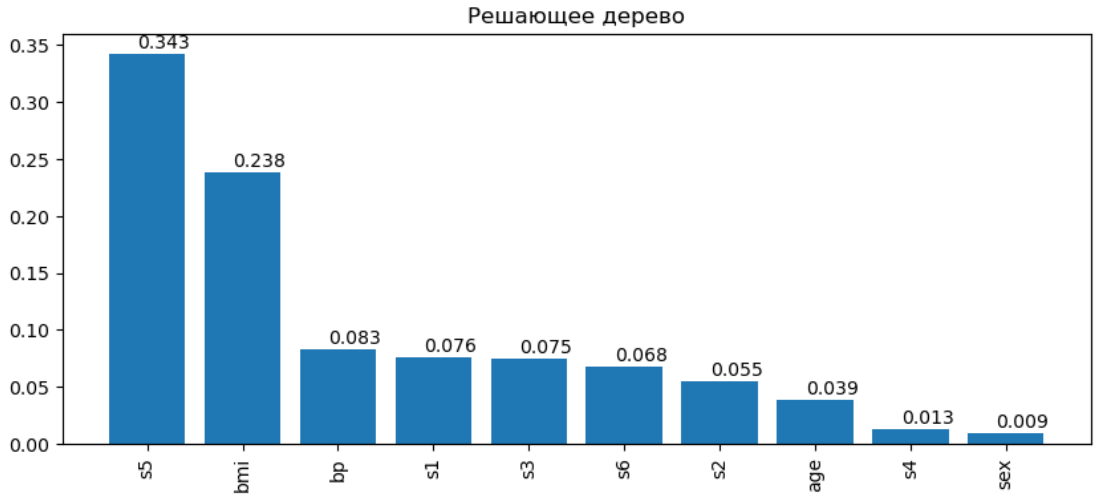

```
sel_mi = SelectKBest(mutual_info_regression, k=5).fit(diabetes_X, diabetes_y)
list(zip(diabetes_feature_names, sel_mi.get_support()))
```

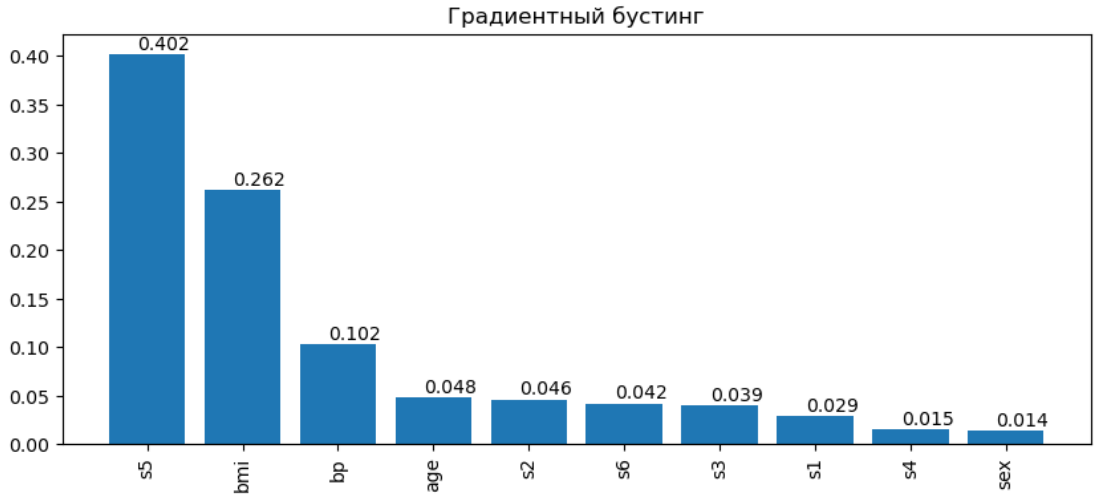
```
[('age', False),
 ('sex', False),
 ('bmi', True),
 ('bp', False),
 ('s1', True),
 ('s2', False),
 ('s3', False),
 ('s4', True),
 ('s5', True),
 ('s6', True)]
```

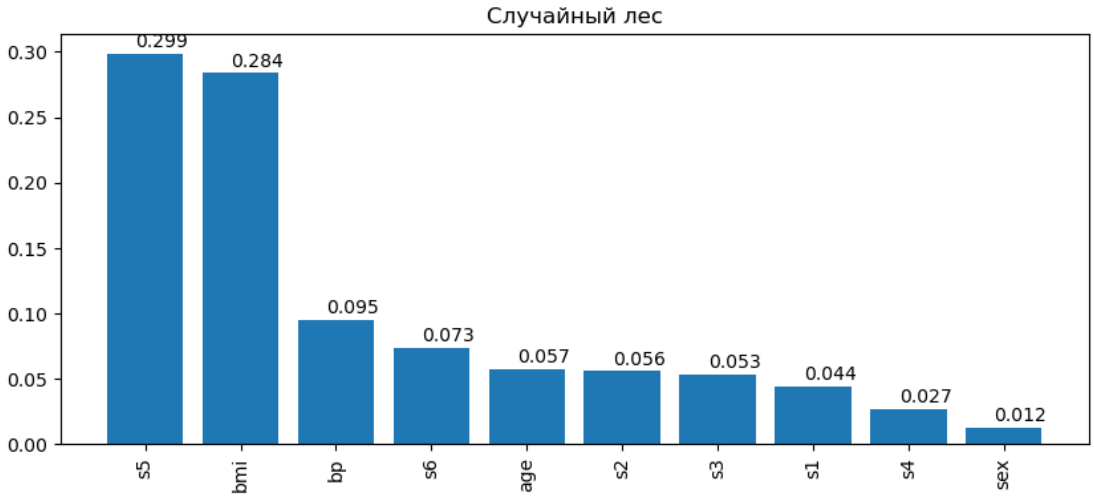
```
dtr1 = DecisionTreeRegressor()
rfr1 = RandomForestRegressor()
gbr1 = GradientBoostingRegressor()
dtr1.fit(diabetes_X, diabetes_y)
rfr1.fit(diabetes_X, diabetes_y)
gbr1.fit(diabetes_X, diabetes_y)

# Важность признаков
dtr1.feature_importances_, sum(dtr1.feature_importances_)
```

```
from operator import itemgetter

def draw_feature_importances(tree_model, X_dataset, title, figsize=(7,4)):
    """
    Вывод важности признаков в виде графика
    """
    # Сортировка значений важности признаков по убыванию
    list_to_sort = list(zip(X_dataset.columns.values,
tree_model.feature_importances_))
    sorted_list = sorted(list_to_sort, key=itemgetter(1), reverse = True)
    # Названия признаков
    labels = [x for x,_ in sorted_list]
    # Важности признаков
    data = [x for _,x in sorted_list]
```

```
# Вывод графика
fig, ax = plt.subplots(figsize=figsize)
ax.set_title(title)
ind = np.arange(len(labels))
plt.bar(ind, data)
plt.xticks(ind, labels, rotation='vertical')
# Вывод значений
for a,b in zip(ind, data):
        plt.text(a-0.1, b+0.005, str(round(b,3)))
plt.show()
return labels, data
```






Случайный лес
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.299
0.284
0.095
0.073
0.057
0.056
0.053
0.044
0.027
0.012
s5
bmi
bp
s6
age
s2
s3
s1
s4
sex