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, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)

ОТЧЕТ

Лабораторная работа №3

«Обработка признаков»

по курсу «Методы машинного обучения»

ИСПОЛНИТЕЛЬ: Савченко Г. А.

ФИО

группа ИУ5-21М

____

подпись

"__"_________2023 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю.Е.

ФИО

____

подпись

"__"_________2023 г.

Москва - 2023

Масштабирование признаков

Масштабирование - это изменение диапазона измерения признака с целью улучшения качества построения модели.

Почему необходимо масштабировать признаки?

Многие алгоритмы машинного обучения устроены таким образом, что признаки с меньшей амплитудой оказываются "оштрафованы" по сравнению с признаками с большей амплитудой, и оказывают меньшее влияние на процесс построения модели.

Методы машинного обучения (как с учителем, так и без учителя), **ЗАВИСЯЩИЕ** от масштабирования признаков:

- Метод ближайших соседей
- Линейная регрессия
- Логистическая регрессия
- Метод опорных векторов (SVM)
- Нейронные сети
- Некоторые алгоритмы кластеризации (K-means)
- Анализ главных компонент (Principal Component Analysis, PCA)

Методы машинного обучения, **НЕ ЗАВИСЯЩИЕ** от масштабирования признаков:

- Деревья решений и другие алгоритмы на их основе:
  - Случайный лес
  - Градиентный бустинг В алгоритме построения дерева решения **не строится единое метрическое пространство по всем признакам**. Строится набор ветвлений по отдельным признакам, масштаб признаков не имеет значения.

Признаки нужно масштабировать до или после деления на обучающую и тестовую выборку?

Предположим, что мы разделили данные на обучающую и тестовую выборки, и взяли данные для масштабирования только из обучающей выборки. В этом случае наличие выбросов в тестовой выборке может нарушить схему масштабирования. Традиционным является подход, при котором данные делятся на обучающую и тестовую выборки ДО масштабирования. Параметры масштабирования (например, среднее значение, дисперсия) берутся только из обучающей выборки и затем применяются к тестовой выборке. Если выбросы в тестовой выборке мешают реализации этого подхода, то данные делятся на обучающую и тестовую выборки ПОСЛЕ масштабирования.

Обработка признаков

In [1]:

```
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
```

```
/root/miniconda3/lib/python3.9/site-packages/scipy/__init__.py:146: UserWarning: A N
umPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected ve
rsion 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:

```
data = pd.read_csv('data/graduation_rate.csv', sep=",")
```

In [3]:

```
data.head()
```

Out[3]:

	ACT composite score	SAT total score	parental level of education	parental income	high school gpa	college gpa	years to graduate
0	22	1625	high school	40999	3.0	3.1	7
1	29	2090	associate's degree	75817	4.0	3.4	5
2	30	2188	bachelor's degree	82888	4.0	3.9	3
3	33	2151	associate's degree	93518	4.0	3.7	5
4	29	2050	associate's degree	79153	4.0	3.4	6

In [4]:

```
data.describe()
```

Out[4]:

	ACT composite score	SAT total score	parental income	high school gpa	college gpa	years to graduate
count	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000
mean	28.607000	1999.906000	67377.85200	3.707400	3.376500	4.982000
std	2.774211	145.078361	18827.33105	0.287381	0.237179	1.414099
min	20.000000	1598.000000	18906.00000	2.800000	2.600000	3.000000
25%	27.000000	1898.000000	54269.75000	3.500000	3.200000	4.000000
50%	28.500000	2000.000000	67842.50000	3.800000	3.400000	5.000000
75%	31.000000	2099.000000	80465.50000	4.000000	3.500000	6.000000
max	36.000000	2385.000000	124470.00000	4.000000	4.000000	10.000000

In [5]:

```
data.columns
```

Out[5]:

```
Index(['ACT composite score', 'SAT total score', 'parental level of education',
       'parental income', 'high school gpa', 'college gpa',
       'years to graduate'],
      dtype='object')
```

Масштабирование

In [6]:

```
# 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
```

In [7]:

```
data1 = data.drop('parental level of education', axis=1)
X_ALL = data1.drop('SAT total score', axis=1)
```

In [8]:

```
# Разделим выборку на обучающую и тестовую
X_train, X_test, y_train, y_test = train_test_split(X_ALL, data['SAT total score'],
                                                    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[8]:

```
((800, 5), (200, 5))
```

Масштабирование данных на основе Z-оценки

In [9]:

```
# 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
```

Out[9]:

	ACT composite score	parental income	high school gpa	college gpa	years to graduate
0	-2.382770	-1.401795	-2.462772	-1.166368	1.427771
1	0.141733	0.448463	1.018670	0.099131	0.012735
2	0.502376	0.824222	1.018670	2.208296	-1.402301
3	1.584306	1.389110	1.018670	1.364630	0.012735
4	0.141733	0.625741	1.018670	0.099131	0.720253
...	...	...	...	...	...
995	1.223662	-1.951324	1.018670	-1.588201	3.550325
996	-0.579554	-0.421665	-0.373907	-0.744535	-0.694783
997	1.223662	1.436777	1.018670	1.786463	-0.694783
998	1.223662	-1.895261	0.670526	-0.744535	2.135289
999	1.944949	1.405796	1.018670	1.786463	0.720253

1000 rows × 5 columns

In [10]:

```
data_cs11_scaled.describe()
```

Out[10]:

	ACT composite score	parental income	high school gpa	college gpa	years to graduate
count	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03
mean	2.273737e-16	6.750156e-17	-9.485746e-16	-1.989520e-16	-1.207923e-16

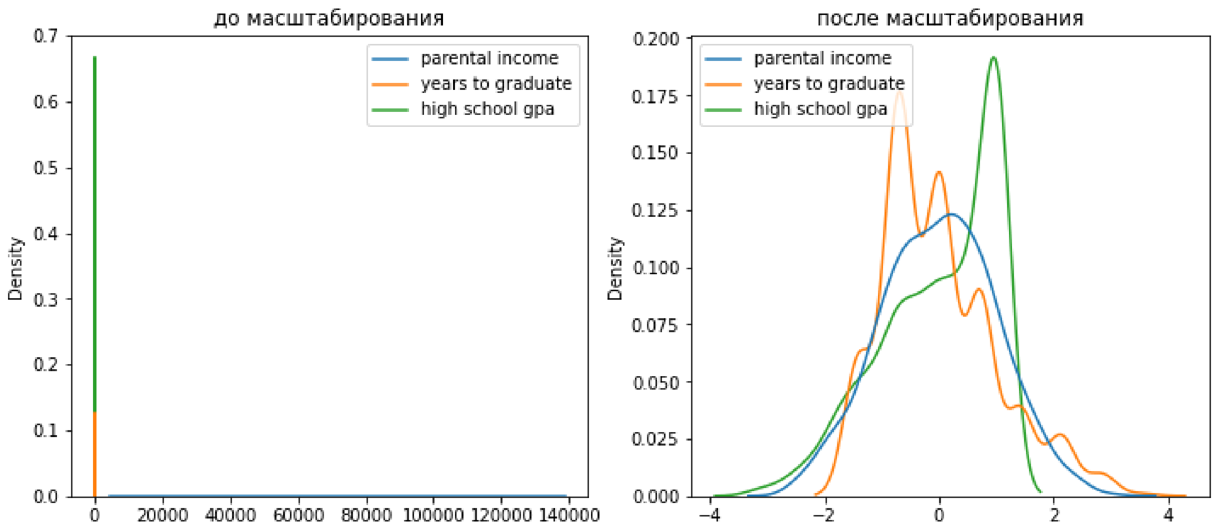
	ACT composite score	parental income	high school gpa	college gpa	years to graduate
std	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00
min	-3.104056e+00	-2.575835e+00	-3.159061e+00	-3.275533e+00	-1.402301e+00
25%	-5.795536e-01	-6.965757e-01	-7.220512e-01	-7.445352e-01	-6.947826e-01
50%	-3.858882e-02	2.469179e-02	3.223816e-01	9.913075e-02	1.273532e-02
75%	8.630192e-01	6.954887e-01	1.018670e+00	5.209637e-01	7.202533e-01
max	2.666235e+00	3.033925e+00	1.018670e+00	2.630129e+00	3.550325e+00

In [11]:

```
# 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()
```

In [12]:

```
draw_kde(['parental income', 'years to graduate', 'high school gpa'], data, data_cs1
```



In [13]:

```
# Обучаем StandardScaler на обучающей выборке и масштабируем обучающую и тестовую вы
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)
```

In [14]:

```
data_cs12_scaled_train.describe()
```

Out[14]:

	ACT composite score	parental income	high school gpa	college gpa	years to graduate
count	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02

	ACT composite score	parental income	high school gpa	college gpa	years to graduate
mean	-5.440093e-17	3.663736e-17	1.056932e-15	3.774758e-16	-3.241851e-16
std	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00
min	-3.034003e+00	-2.564127e+00	-3.113250e+00	-3.295399e+00	-1.436199e+00
25%	-5.726938e-01	-7.077868e-01	-7.091960e-01	-7.486408e-01	-6.992144e-01
50%	1.305373e-01	4.865866e-02	3.211130e-01	1.002786e-01	3.777047e-02
75%	8.337683e-01	6.926155e-01	1.007986e+00	5.247383e-01	7.747553e-01
max	2.591846e+00	3.025304e+00	1.007986e+00	2.647037e+00	2.985710e+00

In [15]:

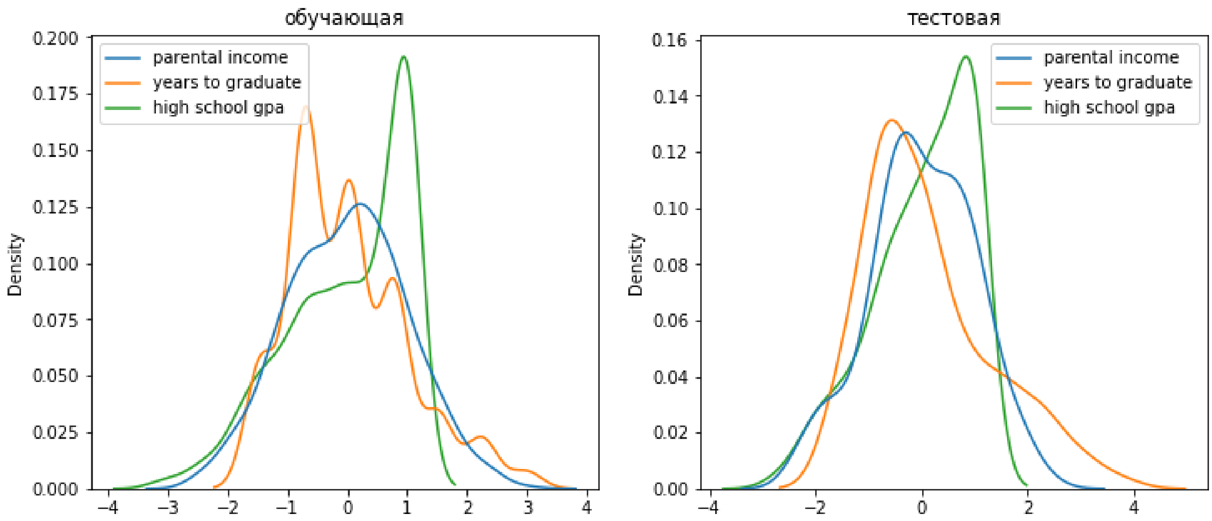
```
data_cs12_scaled_test.describe()
```

Out[15]:

	ACT composite score	parental income	high school gpa	college gpa	years to graduate
count	200.000000	200.000000	200.000000	200.000000	200.000000
mean	-0.038238	0.011870	0.015455	0.002653	0.122524
std	0.869149	0.984152	0.932654	1.033342	1.191741
min	-2.330772	-2.363400	-2.769814	-2.870939	-1.436199
25%	-0.572694	-0.555283	-0.709196	-0.748641	-0.699214
50%	-0.221078	-0.014747	0.321113	0.100279	0.037770
75%	0.482153	0.719910	1.007986	0.949198	0.774755
max	2.240231	2.410840	1.007986	2.647037	3.722695

In [16]:

```
# распределения для обучающей и тестовой выборки немного отличаются
draw_kde(['parental income', 'years to graduate', 'high school gpa'], data_cs12_scal
```



Масштабирование "Mean Normalisation"

In [17]:

```
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)
```

In [18]:

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

Out[18]:

	ACT composite score	parental income	high school gpa	college gpa	years to graduate
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	-0.001359	0.000425	0.000750	0.000089	0.005542
std	0.173388	0.178350	0.239484	0.169414	0.235683
min	-0.539297	-0.458746	-0.755417	-0.554554	-0.324792
25%	-0.101797	-0.123747	-0.172083	-0.125982	-0.158125
50%	-0.008047	0.004826	0.077917	0.016875	0.008542
75%	0.148203	0.124403	0.244583	0.088304	0.175208
max	0.460703	0.541254	0.244583	0.445446	0.841875

In [19]:

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

In [20]:

```
data_cs22_scaled_train.describe()
```

Out[20]:

	ACT composite score	parental income	high school gpa	college gpa	years to graduate
count	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02
mean	-8.881784e-18	6.383782e-18	2.614575e-16	6.078471e-17	-7.438494e-17
std	1.778622e-01	1.790210e-01	2.427974e-01	1.683864e-01	2.262881e-01
min	-5.392969e-01	-4.587455e-01	-7.554167e-01	-5.545536e-01	-3.247917e-01
25%	-1.017969e-01	-1.266295e-01	-1.720833e-01	-1.259821e-01	-1.581250e-01
50%	2.320312e-02	8.705477e-03	7.791667e-02	1.687500e-02	8.541667e-03
75%	1.482031e-01	1.239152e-01	2.445833e-01	8.830357e-02	1.752083e-01
max	4.607031e-01	5.412545e-01	2.445833e-01	4.454464e-01	6.752083e-01

In [21]:

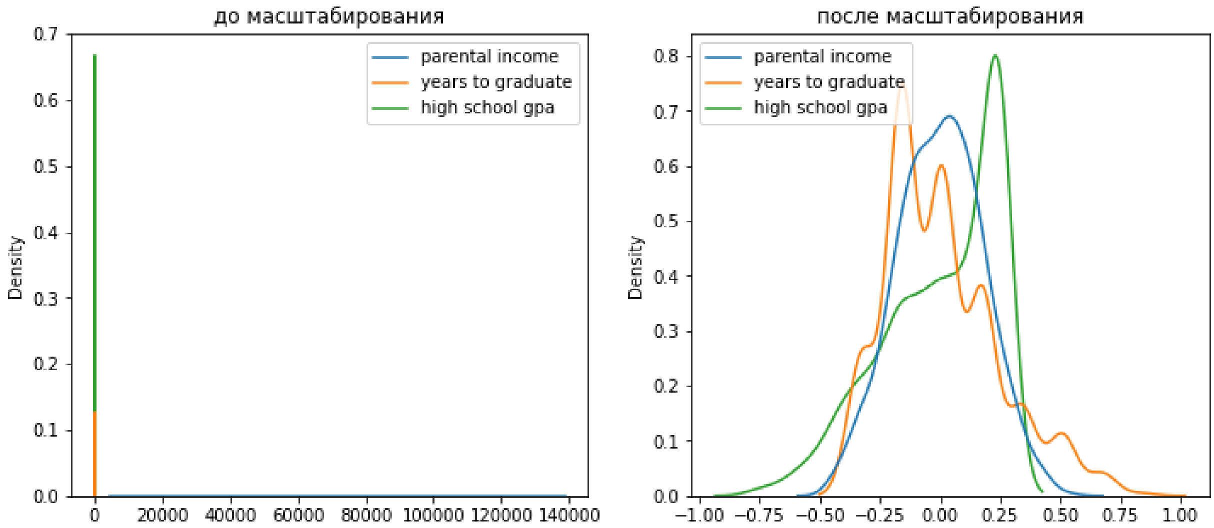
```
data_cs22_scaled_test.describe()
```

Out[21]:

	ACT composite score	parental income	high school gpa	college gpa	years to graduate
count	200.000000	200.000000	200.000000	200.000000	200.000000
mean	-0.006797	0.002124	0.003750	0.000446	0.027708
std	0.154492	0.176074	0.226304	0.173892	0.269508
min	-0.414297	-0.422834	-0.672083	-0.483125	-0.324792
25%	-0.101797	-0.099345	-0.172083	-0.125982	-0.158125
50%	-0.039297	-0.002638	0.077917	0.016875	0.008542
75%	0.085703	0.128799	0.244583	0.159732	0.175208
max	0.398203	0.431321	0.244583	0.445446	0.841875

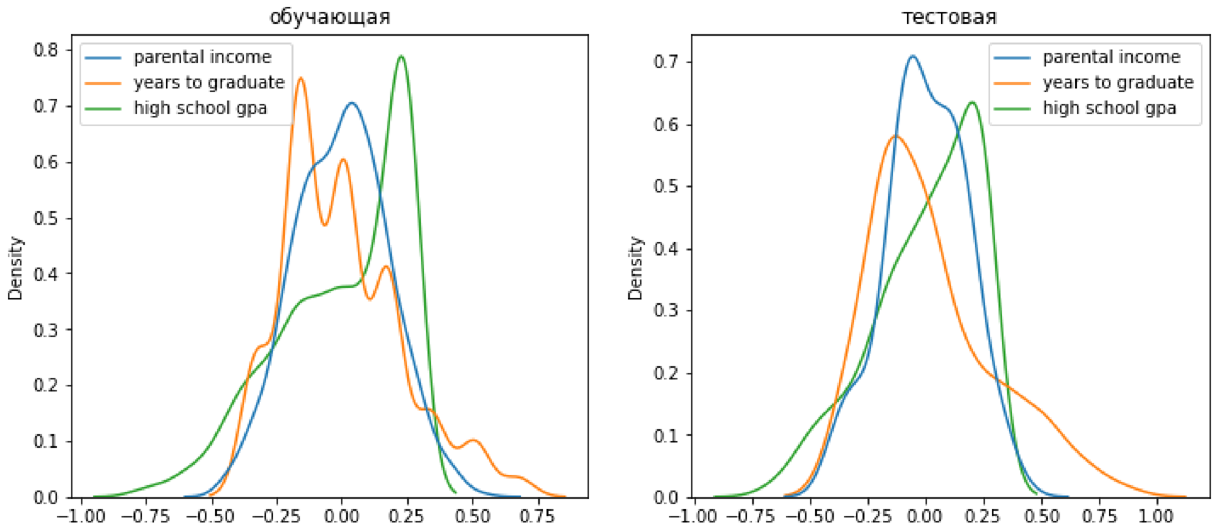
In [22]:

```
draw_kde(['parental income', 'years to graduate', 'high school gpa'], data, data_cs2
```



In [23]:

```
draw_kde(['parental income', 'years to graduate', 'high school gpa'], data_cs22_scal
```



MinMax-масштабирование

In [24]:

```
# 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[24]:

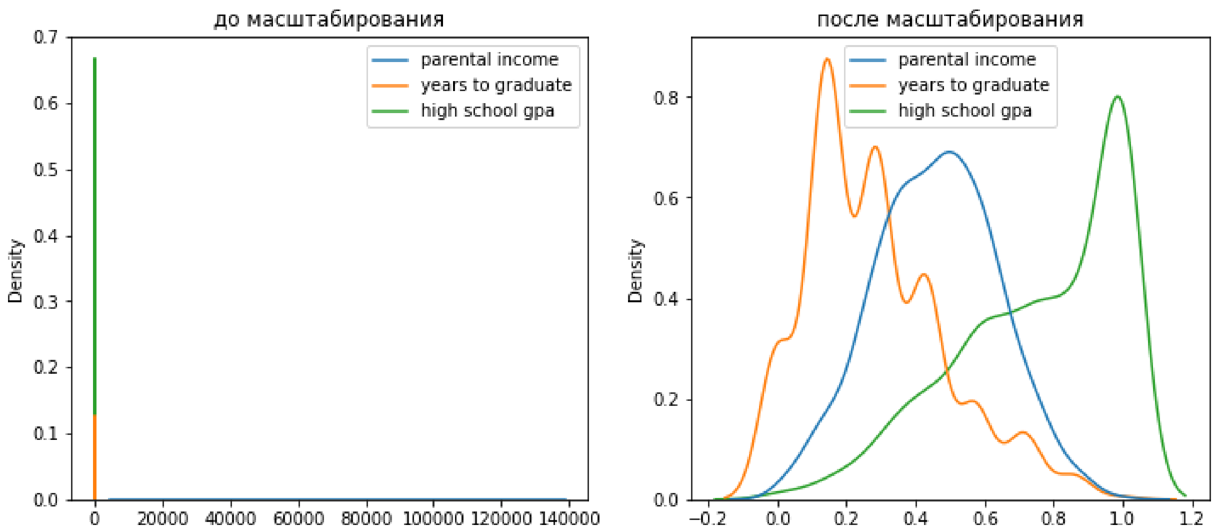
	ACT composite score	parental income	high school gpa	college gpa	years to graduate
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.537937	0.459170	0.756167	0.554643	0.283143
std	0.173388	0.178350	0.239484	0.169414	0.202014
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.437500	0.334998	0.583333	0.428571	0.142857
50%	0.531250	0.463572	0.833333	0.571429	0.285714
75%	0.687500	0.583149	1.000000	0.642857	0.428571
max	1.000000	1.000000	1.000000	1.000000	1.000000

In [25]:

```
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)
```

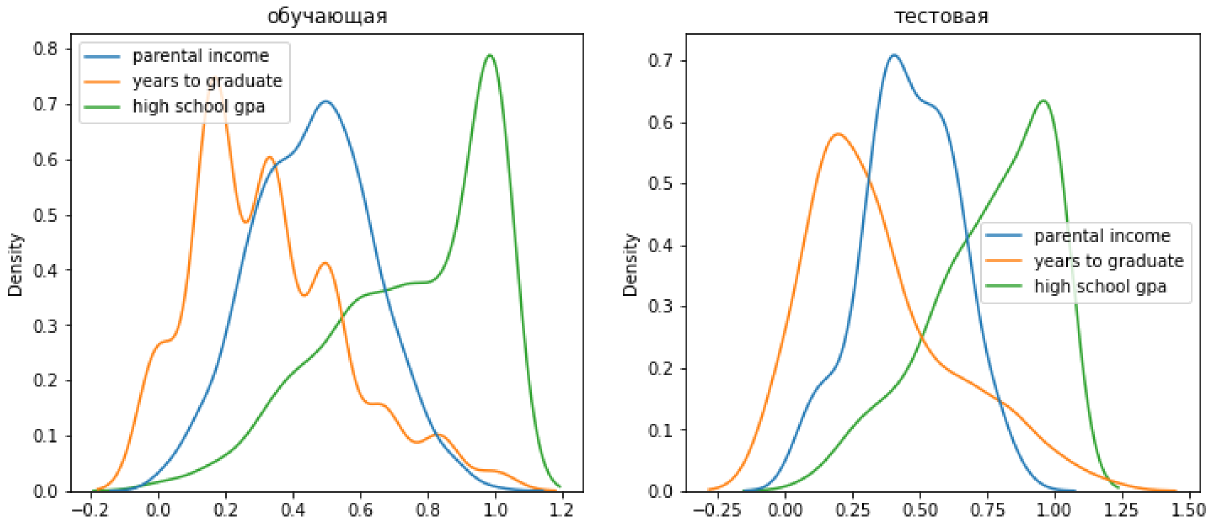
In [26]:

```
draw_kde(['parental income', 'years to graduate', 'high school gpa'], data, data_cs3
```



In [27]:

```
draw_kde(['parental income', 'years to graduate', 'high school gpa'], data_cs32_scal
```



Обработка выбросов

Удаление выбросов

In [28]:
```
data.shape
```

Out[28]:
```
(1000, 7)
```

In [29]:
```
x_col_list = ['parental income']
```

In [30]:
```
import scipy.stats as stats
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()
```

In [31]:
```
# Тип вычисления верхней и нижней границы выбросов
from enum import Enum
class OutlierBoundaryType(Enum):
    SIGMA = 1
    QUANTILE = 2
    IRQ = 3
```

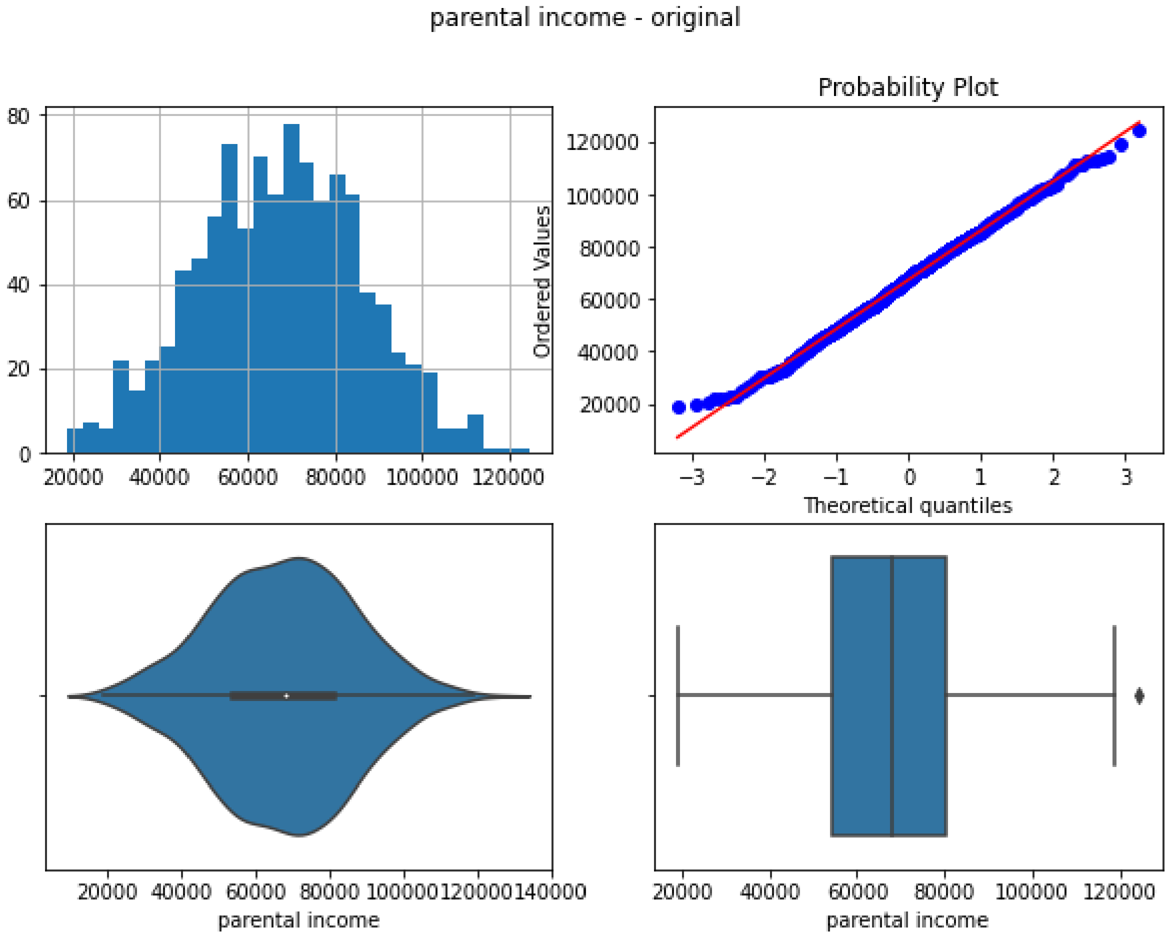
In [32]:
```
# Функция вычисления верхней и нижней границы выбросов
def get_outlier_boundaries(df, col, outlier_boundary_type: OutlierBoundaryType):
    if outlier_boundary_type == OutlierBoundaryType.SIGMA:
```

```
        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
```

In [33]:

```
diagnostic_plots(data, 'parental income', 'parental income - original')
```

```
/tmp/ipykernel_776/3064582745.py:5: MatplotlibDeprecationWarning: Auto-removal of ov
erlapping axes is deprecated since 3.6 and will be removed two minor releases later;
explicitly call ax.remove() as needed.
  plt.subplot(2, 2, 1)
```

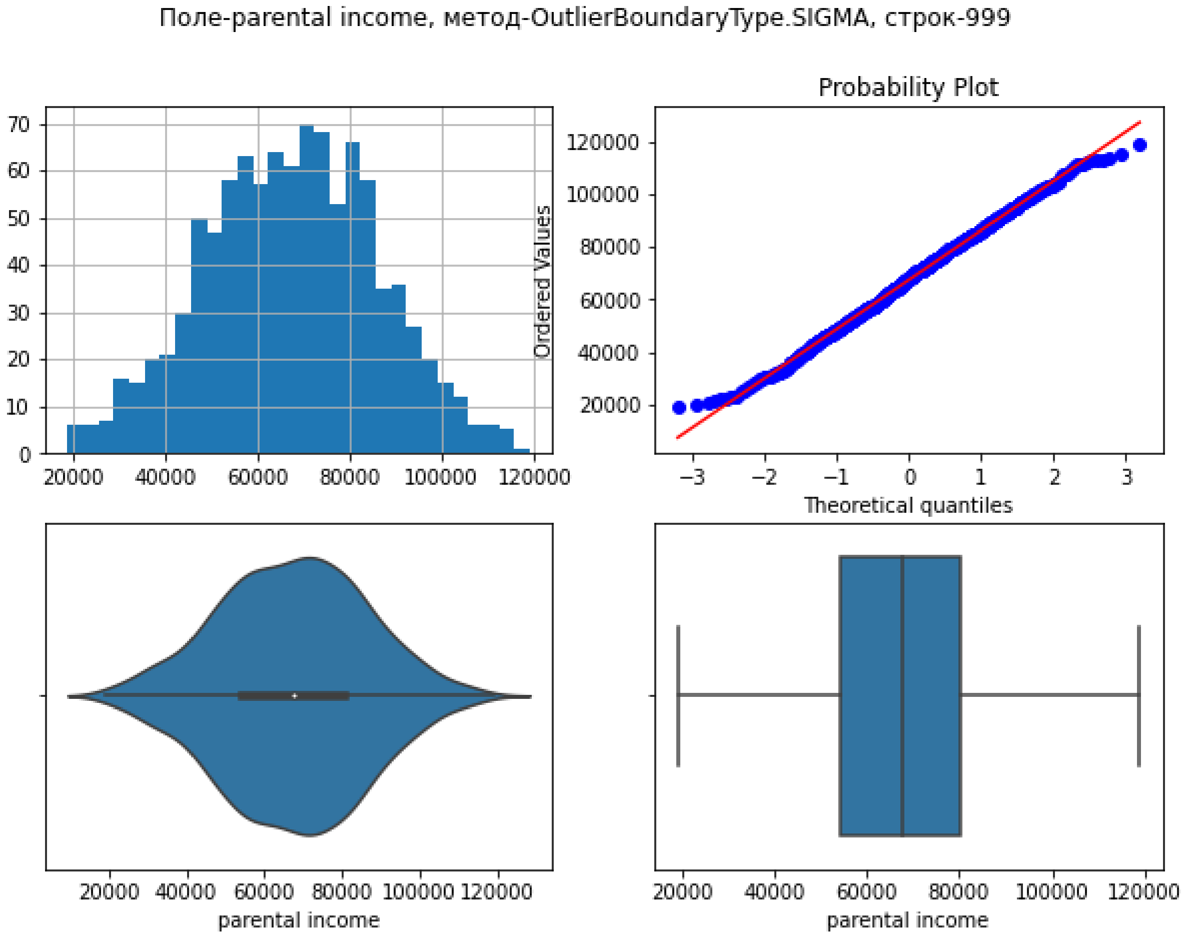


In [34]:

```
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,
```

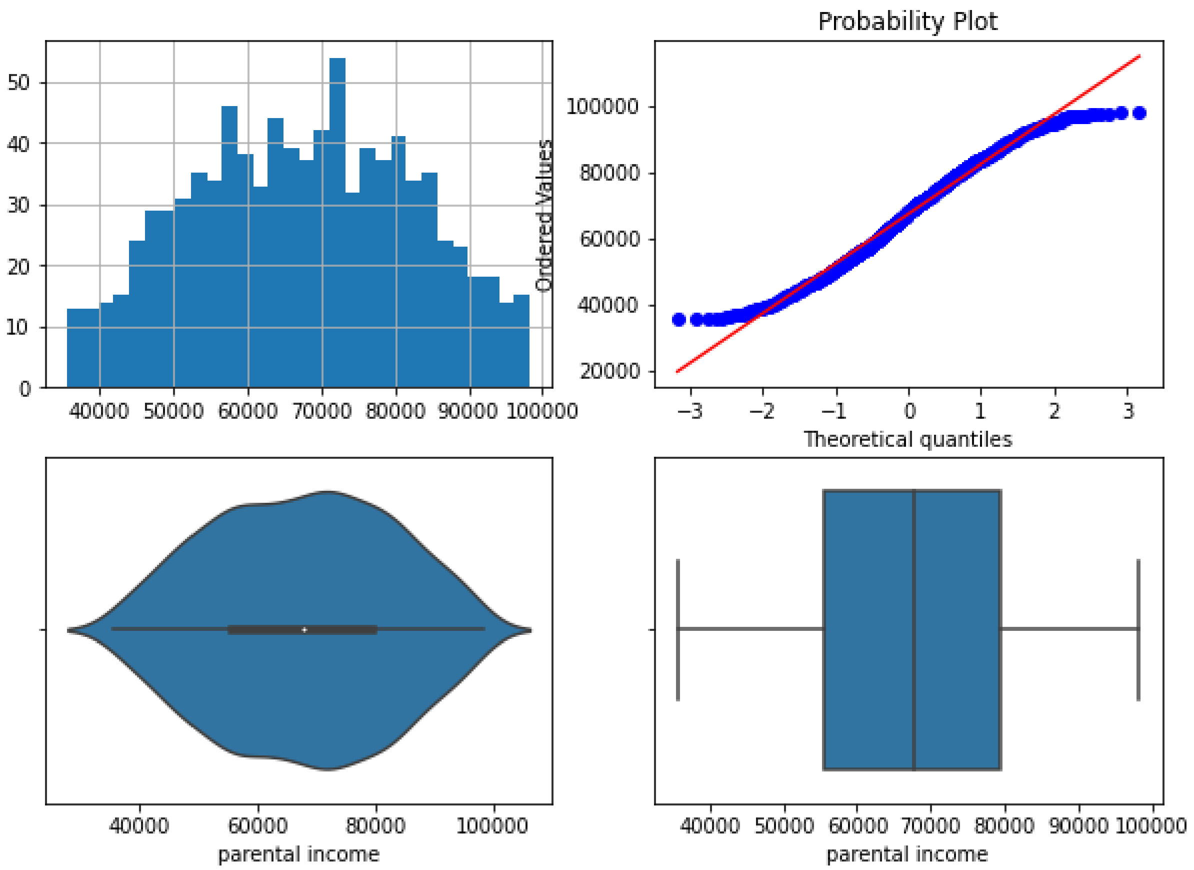
```
                        np.where(data[col] < lower_boundary, True, False))
        # Удаление данных на основе флага
        data_trimmed = data.loc[~(outliers_temp), ]
        title = 'Поле-{}, метод-{}, строк-{}'.format(col, obt, data_trimmed.shape[0]
        diagnostic_plots(data_trimmed, col, title)
```

```
/tmp/ipykernel_776/3064582745.py:5: MatplotlibDeprecationWarning: Auto-removal of ov
erlapping axes is deprecated since 3.6 and will be removed two minor releases later;
explicitly call ax.remove() as needed.
  plt.subplot(2, 2, 1)
```



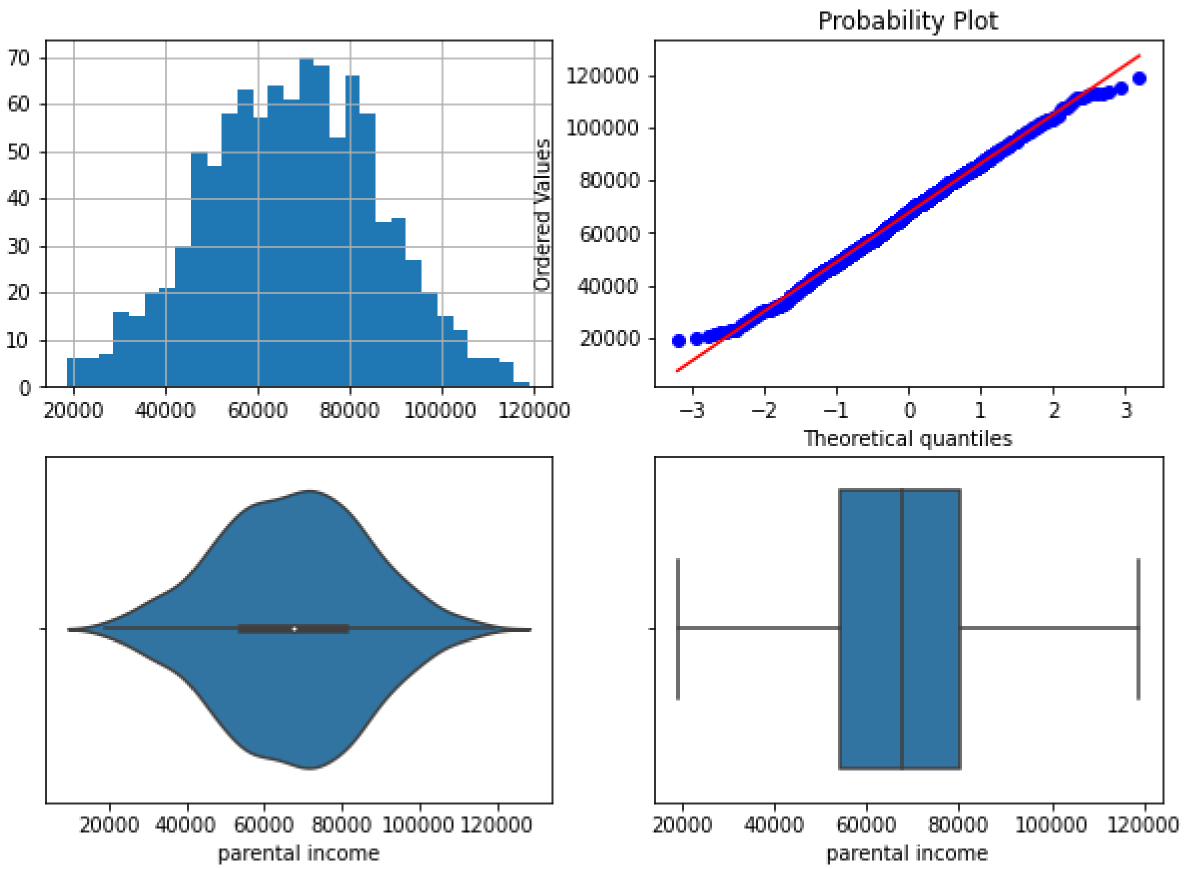

```
/tmp/ipykernel_776/3064582745.py:5: MatplotlibDeprecationWarning: Auto-removal of ov
erlapping axes is deprecated since 3.6 and will be removed two minor releases later;
explicitly call ax.remove() as needed.
  plt.subplot(2, 2, 1)
```

Поле-parental income, метод-OutlierBoundaryType.QUANTILE, строк-900




```
/tmp/ipykernel_776/3064582745.py:5: MatplotlibDeprecationWarning: Auto-removal of ov
erlapping axes is deprecated since 3.6 and will be removed two minor releases later;
explicitly call ax.remove() as needed.
  plt.subplot(2, 2, 1)
```

Поле-parental income, метод-OutlierBoundaryType.IRQ, строк-999



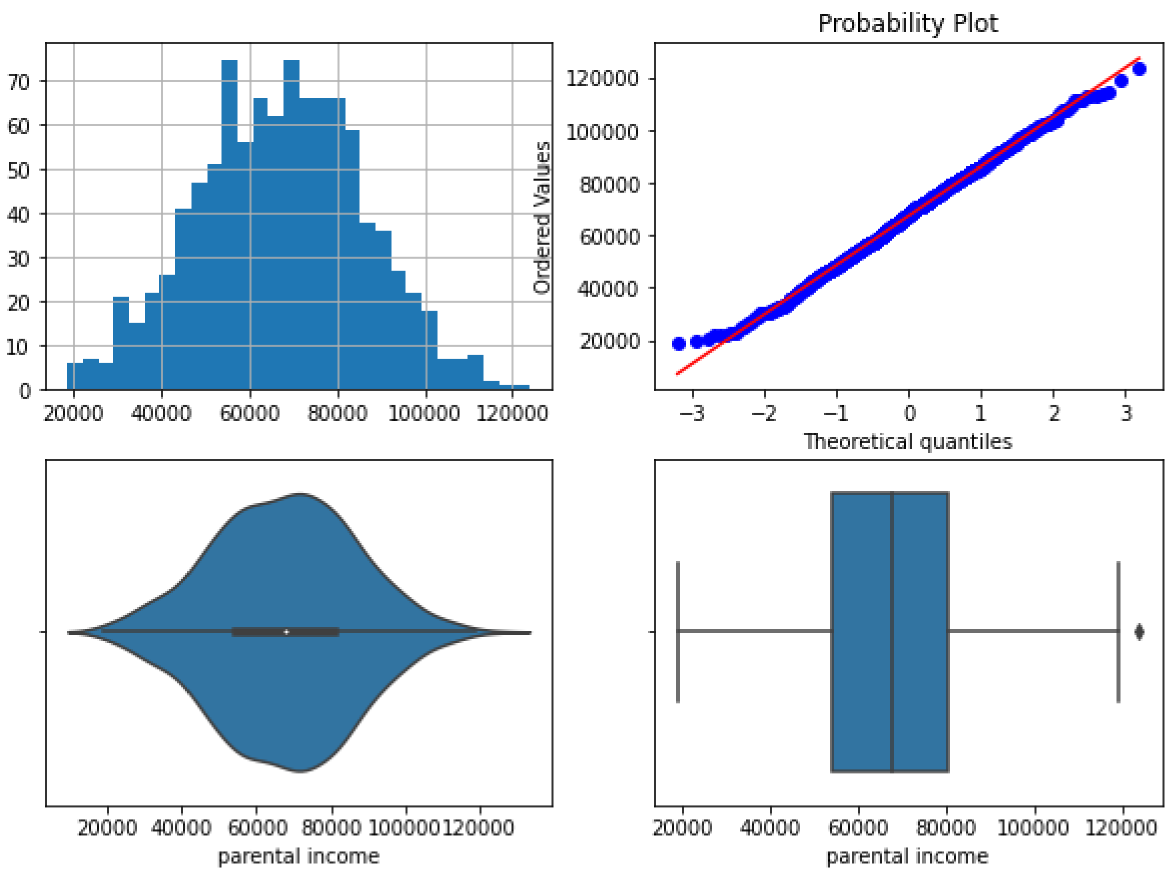
Замена выбросов

In [35]:

```
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
        title = 'Поле-{}, метод-{}'.format(col, obt)
        diagnostic_plots(data, col, title)
```

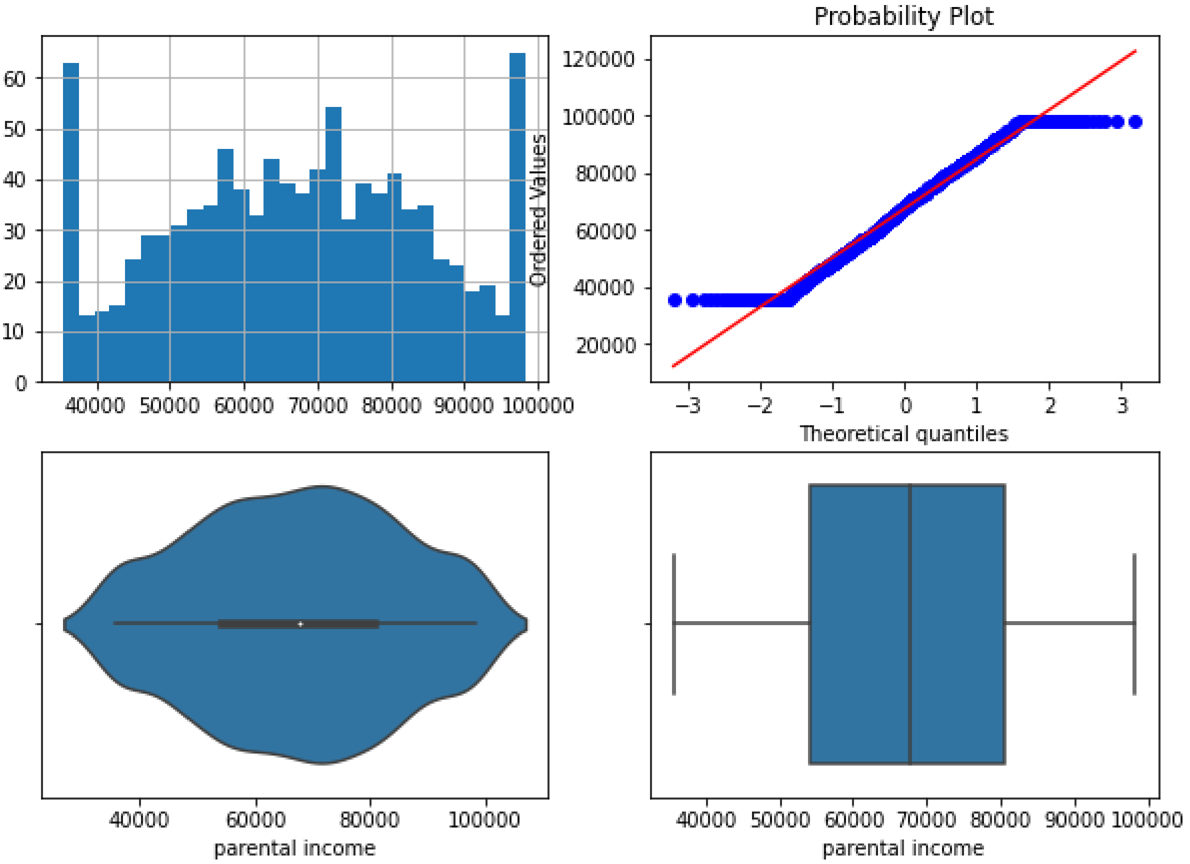
```
/tmp/ipykernel_776/3064582745.py:5: MatplotlibDeprecationWarning: Auto-removal of ov
erlapping axes is deprecated since 3.6 and will be removed two minor releases later;
explicitly call ax.remove() as needed.
  plt.subplot(2, 2, 1)
```

Поле-parental income, метод-OutlierBoundaryType.SIGMA



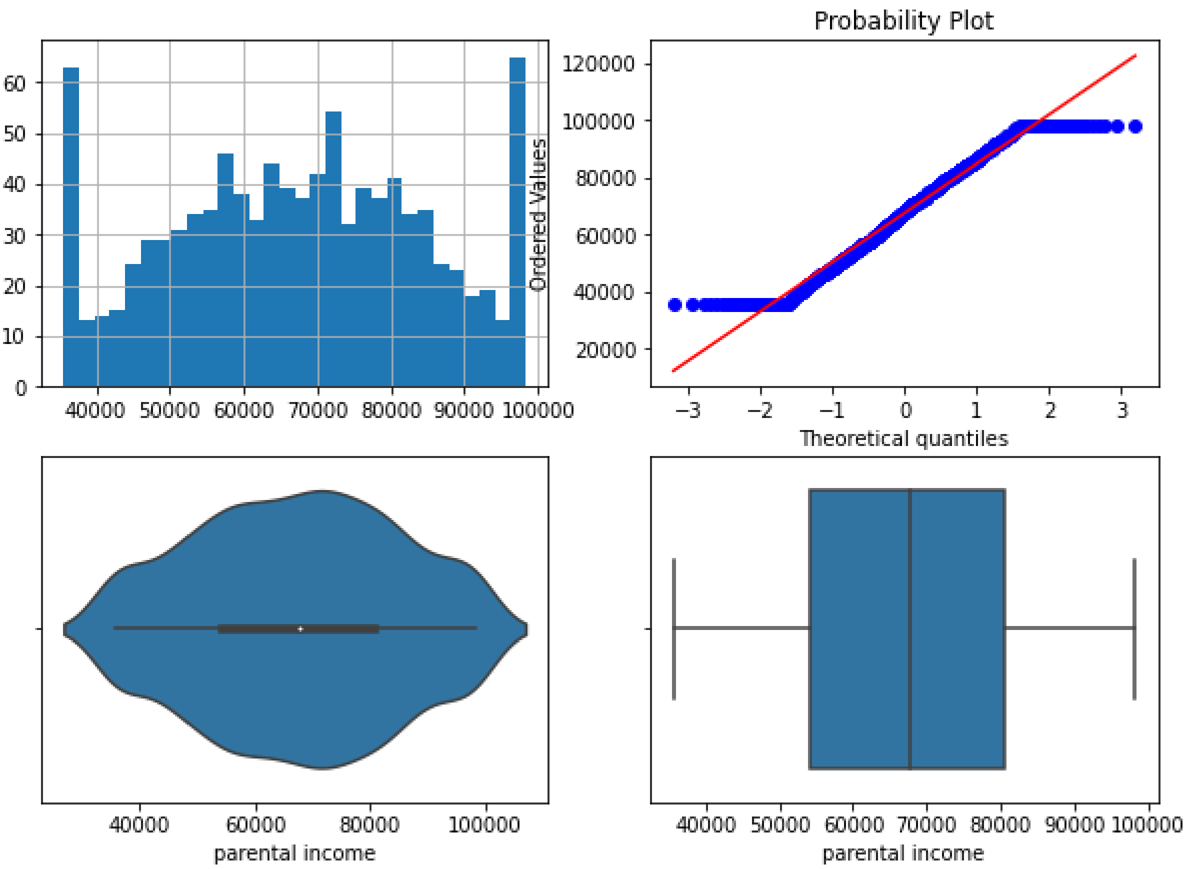

```
/tmp/ipykernel_776/3064582745.py:5: MatplotlibDeprecationWarning: Auto-removal of ov
erlapping axes is deprecated since 3.6 and will be removed two minor releases later;
explicitly call ax.remove() as needed.
  plt.subplot(2, 2, 1)
```

Поле-parental income, метод-OutlierBoundaryType.QUANTILE




```
/tmp/ipykernel_776/3064582745.py:5: MatplotlibDeprecationWarning: Auto-removal of ov
erlapping axes is deprecated since 3.6 and will be removed two minor releases later;
explicitly call ax.remove() as needed.
  plt.subplot(2, 2, 1)
```

Поле-parental income, метод-OutlierBoundaryType.IRQ



Обработка нестандартного признака

In [36]:
```
data = pd.read_csv('data/bike-hour.csv', sep=",")
```

In [37]:
```
data
```

Out[37]:

	instant	dteday	season	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp
0	1	01-01-2011	1	1	0	0	6	0	1	0.24	0.2879
1	2	01-01-2011	1	1	1	0	6	0	1	0.22	0.2727
2	3	01-01-2011	1	1	2	0	6	0	1	0.22	0.2727
3	4	01-01-2011	1	1	3	0	6	0	1	0.24	0.2879
4	5	01-01-2011	1	1	4	0	6	0	1	0.24	0.2879
...	...	...	...	...	...	...	...	...	...	...	...
8640	8641	31-12-2011	1	12	19	0	6	0	1	0.42	0.4242
8641	8642	31-12-2011	1	12	20	0	6	0	1	0.42	0.4242
8642	8643	31-12-2011	1	12	21	0	6	0	1	0.40	0.4091
8643	8644	31-12-2011	1	12	22	0	6	0	1	0.38	0.3939
8644	8645	31-12-2011	1	12	23	0	6	0	1	0.36	0.3788

8645 rows × 15 columns

In [38]:
```
data.dtypes
```

Out[38]:
```
instant         int64
dteday         object
season          int64
mnth            int64
hr              int64
holiday         int64
weekday         int64
workingday      int64
weathersit      int64
temp          float64
atemp         float64
hum           float64
windspeed     float64
casual          int64
```

```
cnt              int64
dtype: object
```

In [39]:

```python
data = data.drop('season', 1)
data = data.drop('mnth', 1)
data = data.drop('holiday', 1)
data = data.drop('weekday', 1)
data = data.drop('workingday', 1)
data.shape
```

```
/tmp/ipykernel_776/2854027586.py:1: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  data = data.drop('season', 1)
/tmp/ipykernel_776/2854027586.py:2: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  data = data.drop('mnth', 1)
/tmp/ipykernel_776/2854027586.py:3: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  data = data.drop('holiday', 1)
/tmp/ipykernel_776/2854027586.py:4: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  data = data.drop('weekday', 1)
/tmp/ipykernel_776/2854027586.py:5: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  data = data.drop('workingday', 1)
```

Out[39]:

```
(8645, 10)
```

In [40]:

```python
# Сконвертируем дату и время в нужный формат
data['dt'] = data.apply(lambda x: pd.to_datetime(x['dteday'], format='%d-%m-%Y'), ax
```

In [41]:

```python
data.head()
```

Out[41]:

	instant	dteday	hr	weathersit	temp	atemp	hum	windspeed	casual	cnt	dt
0	1	01-01-2011	0	1	0.24	0.2879	0.81	0.0	3	16	2011-01-01
1	2	01-01-2011	1	1	0.22	0.2727	0.80	0.0	8	40	2011-01-01
2	3	01-01-2011	2	1	0.22	0.2727	0.80	0.0	5	32	2011-01-01
3	4	01-01-2011	3	1	0.24	0.2879	0.75	0.0	3	13	2011-01-01
4	5	01-01-2011	4	1	0.24	0.2879	0.75	0.0	0	1	2011-01-01

In [42]:

```python
data.dtypes
```

Out[42]:

```
instant                int64
dteday                object
hr                     int64
weathersit             int64
temp                 float64
atemp                float64
hum                  float64
windspeed            float64
casual                 int64
cnt                    int64
dt            datetime64[ns]
dtype: object
```

In [43]:
```
# День
data['day'] = data['dt'].dt.day
# Месяц
data['month'] = data['dt'].dt.month
# Год
data['year'] = data['dt'].dt.year
#Неделя года
data['week'] = data['dt'].dt.isocalendar().week
#Квартал
data['quarter'] = data['dt'].dt.quarter
#День недели
data['dayofweek'] = data['dt'].dt.dayofweek
#Выходной день
data['day_name'] = data['dt'].dt.day_name()
data['is_holiday'] = data.apply(lambda x: 1 if x['dt'].dayofweek in [5,6] else 0, ax
```

In [44]:
```
data.head()
```

Out[44]:

	instant	dteday	hr	weathersit	temp	atemp	hum	windspeed	casual	cnt	dt	day	month
0	1	01-01-2011	0	1	0.24	0.2879	0.81	0.0	3	16	2011-01-01	1	1
1	2	01-01-2011	1	1	0.22	0.2727	0.80	0.0	8	40	2011-01-01	1	1
2	3	01-01-2011	2	1	0.22	0.2727	0.80	0.0	5	32	2011-01-01	1	1
3	4	01-01-2011	3	1	0.24	0.2879	0.75	0.0	3	13	2011-01-01	1	1
4	5	01-01-2011	4	1	0.24	0.2879	0.75	0.0	0	1	2011-01-01	1	1

In [45]:
```
import datetime
```

In [46]:
```
# Разница между датами
data['now'] = datetime.datetime.today()
data['diff'] = data['now'] - data['dt']
data.dtypes
```

Out[46]:
```
instant                int64
dteday                object
hr                     int64
weathersit             int64
temp                 float64
atemp                float64
hum                  float64
windspeed            float64
casual                 int64
cnt                    int64
dt            datetime64[ns]
day                    int64
month                  int64
year                   int64
week                  UInt32
quarter                int64
```

```
dayofweek                 int64
day_name                 object
is_holiday                int64
now              datetime64[ns]
diff            timedelta64[ns]
dtype: object
```

In [47]:
```
data.head()
```

Out[47]:

	instant	dteday	hr	weathersit	temp	atemp	hum	windspeed	casual	cnt	...	day	month	y(
0	1	01-01-2011	0	1	0.24	0.2879	0.81	0.0	3	16	...	1	1	2(
1	2	01-01-2011	1	1	0.22	0.2727	0.80	0.0	8	40	...	1	1	2(
2	3	01-01-2011	2	1	0.22	0.2727	0.80	0.0	5	32	...	1	1	2(
3	4	01-01-2011	3	1	0.24	0.2879	0.75	0.0	3	13	...	1	1	2(
4	5	01-01-2011	4	1	0.24	0.2879	0.75	0.0	0	1	...	1	1	2(

5 rows × 21 columns

Отбор признаков

Отбор признаков из группы методом фильтрации (корреляция признаков)

In [48]:
```
data = pd.read_csv('data/graduation_rate.csv', sep=",")
```

In [49]:
```
data.columns
```

Out[49]:
```
Index(['ACT composite score', 'SAT total score', 'parental level of education',
       'parental income', 'high school gpa', 'college gpa',
       'years to graduate'],
      dtype='object')
```

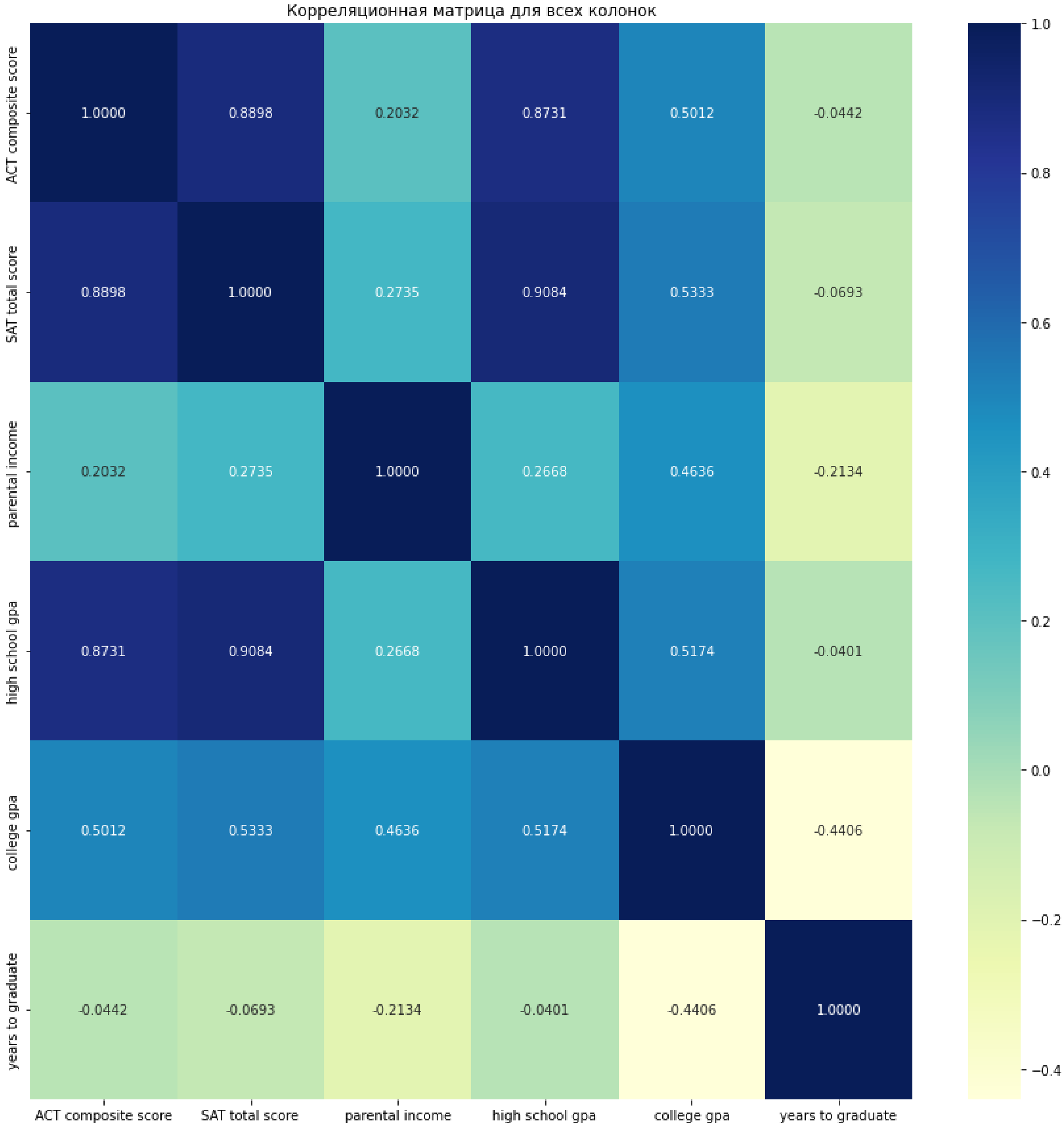
In [50]:
```
data.head()
```

Out[50]:

	ACT composite score	SAT total score	parental level of education	parental income	high school gpa	college gpa	years to graduate
0	22	1625	high school	40999	3.0	3.1	7
1	29	2090	associate's degree	75817	4.0	3.4	5
2	30	2188	bachelor's degree	82888	4.0	3.9	3
3	33	2151	associate's degree	93518	4.0	3.7	5
4	29	2050	associate's degree	79153	4.0	3.4	6

In [51]:
```
col_ch=['ACT composite score', 'SAT total score', 'parental income', 'high school gp
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(data[col_ch].corr(), annot=True, fmt='.4f', cmap="YlGnBu")
ax.set_title('Корреляционная матрица для всех колонок')
```

Out[51]:
```
Text(0.5, 1.0, 'Корреляционная матрица для всех колонок')
```



In [52]:
```
# 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 >= 0.45]
    cr = cr[cr < 1]
    cr = pd.DataFrame(cr).reset_index()
    cr.columns = ['f1', 'f2', 'corr']
    return cr
```

In [53]:
```
make_corr_df(data)
```

Out[53]:

	f1	f2	corr
0	high school gpa	SAT total score	0.908418
1	SAT total score	high school gpa	0.908418
2	SAT total score	ACT composite score	0.889816
3	ACT composite score	SAT total score	0.889816
4	ACT composite score	high school gpa	0.873126
5	high school gpa	ACT composite score	0.873126
6	SAT total score	college gpa	0.533280
7	college gpa	SAT total score	0.533280
8	high school gpa	college gpa	0.517441
9	college gpa	high school gpa	0.517441
10	ACT composite score	college gpa	0.501218
11	college gpa	ACT composite score	0.501218
12	parental income	college gpa	0.463646
13	college gpa	parental income	0.463646

In [54]:

```
# 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
```

In [55]:

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

Out[55]:

```
[['SAT total score', 'ACT composite score', 'college gpa', 'high school gpa'],
 ['college gpa', 'parental income']]
```

Отбор признаков из группы методом обертывания (алгоритм полного перебора)

In [56]:

```
import joblib
import sys
sys.modules['sklearn.externals.joblib'] = joblib
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
```

In [ ]:

```
!pip install mlxtend
```

In [58]:

```
import warnings
warnings.simplefilter("ignore", UserWarning)
```

In [59]:

```
data = pd.read_csv('data/bike-hour.csv', sep=",")
```

In [60]:

```
col_ch=['season', 'mnth', 'hr', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
       'casual']
```

In [61]:

```
iris_X = data[col_ch]
iris_y = data['cnt']
iris_feature_names = col_ch
```

In [62]:

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

efs1 = efs1.fit(iris_X, iris_y)

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

```
Features: 781/781
Best accuracy score: 0.03
Best subset (indices): (2, 4, 8, 9)
Best subset (corresponding names): ('hr', 'weekday', 'atemp', 'hum')
```

Отбор признаков из группы методов вложения (логистическая регрессия)

In [63]:

```
from sklearn.linear_model import LogisticRegression
# Используем L1-регуляризацию
e_lr1 = LogisticRegression(C=1000, solver='liblinear', penalty='l1', max_iter=500, r
e_lr1.fit(iris_X, iris_y)
# Коэффициенты регрессии
e_lr1.coef_
```

Out[63]:

```
array([[-1.79803706e-01, -7.48908923e-02, -1.70834448e-01, ...,
         5.41522385e-01,  1.94308626e+00, -1.65263659e+00],
       [-3.08821362e-01,  3.75976134e-03, -1.68993067e-01, ...,
         1.26470983e-01, -4.68818557e-01, -1.11936699e+00],
       [-9.77261731e-02,  6.76173561e-03, -1.66651060e-01, ...,
        -1.25060669e-01, -7.10751734e-01, -8.62844296e-01],
       ...,
       [-4.87437825e-01,  2.59064130e+00,  1.37191532e-01, ...,
         5.34606024e+00,  3.62448873e+00,  3.92293263e-02],
       [-3.63688746e+01,  3.57562898e+00,  1.77341113e+00, ...,
```

```
        -1.53237722e+02, -9.47353916e+01,  4.10587087e-01],
       [-2.11368802e+00,  1.10130679e+00, -1.87572443e+00, ...,
        -6.04493952e+01, -7.86759346e+01,  2.20013824e-01]])
```

In [65]:

```python
from sklearn.feature_selection import SelectFromModel
sel_e_lr1 = SelectFromModel(e_lr1)
sel_e_lr1.fit(iris_X, iris_y)
sel_e_lr1.get_support()
```

Out[65]:

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

In [66]:

```python
list(zip(col_ch, sel_e_lr1.get_support()))
```

Out[66]:

```
[('season', True),
 ('mnth', True),
 ('hr', True),
 ('holiday', True),
 ('weekday', True),
 ('workingday', True),
 ('weathersit', True),
 ('temp', True),
 ('atemp', True),
 ('hum', True),
 ('windspeed', True),
 ('casual', True)]
```

In [ ]: