Лабораторная работа №4

Предобработка неискажённых данных

Цель: Ознакомиться с методами предобработки данных из библиотеки Scikit Learn

Импорт библиотек

In [40]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
import copy
```

Задание А. Загрузка данных

1. Загрузить датасет из прилагаемого файла (Данные представлены в виде csv таблицы). Можно скачать отсюда https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data.

In [3]:
```
df = pd.read_csv('data/heart_failure_clinical_records_dataset.csv')
df.head()
```

Out[3]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	s
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	

Загрузили датасет, вывели первые записи для проверки корректности импорта

2. Создать Python скрипт. Загрузить датасет в датафрейм, и исключить бинарные признаки и признак времени. Вывести датафрейм на консоль (простой print(df)).

Вместо скрипта python работу я выполняю в jupyter notebook. Датасет загружен. осталось очистить датасет.

In [4]:
```
df = df.loc[:, ["age", "creatinine_phosphokinase", "ejection_fraction", "platelets", "serum_creatinine", "serum_sodium"]]
df.head()
```

Out[4]:

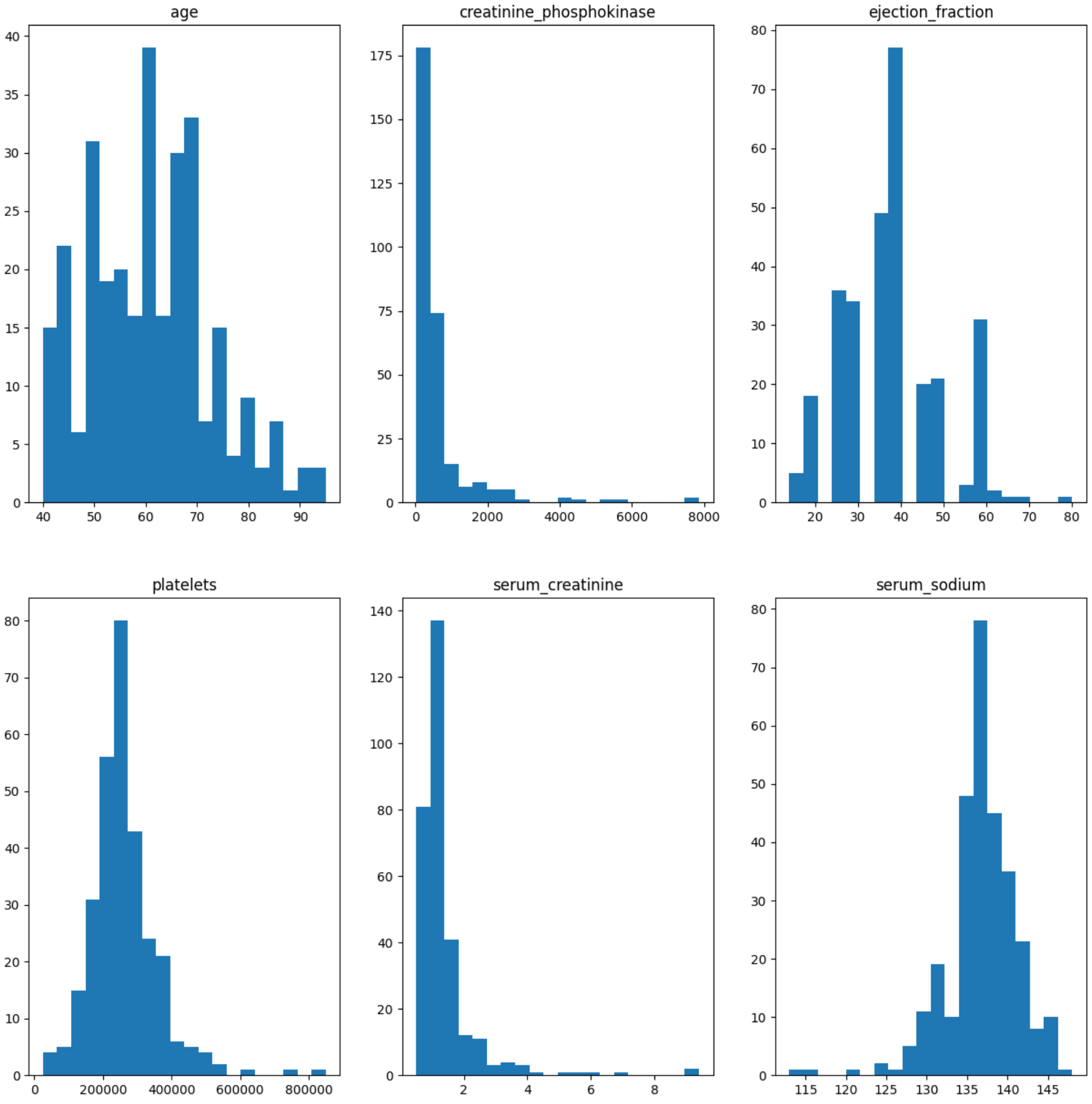
	age	creatinine_phosphokinase	ejection_fraction	platelets	serum_creatinine	serum_sodium
0	75.0	582	20	265000.00	1.9	130
1	55.0	7861	38	263358.03	1.1	136
2	65.0	146	20	162000.00	1.3	129
3	50.0	111	20	210000.00	1.9	137
4	65.0	160	20	327000.00	2.7	116

Видно, что очищены бинарные признаки: anaemia, diabetes, high_blood_pressure, smoking, DEATH_EVENT.
Видно, что очищен временной признак time

3. Построить гистограммы признаков

In [8]:
```
n_bins = 20

fig, axs = plt.subplots(2,3, figsize=(15, 15))

axs[0, 0].hist(df['age'].values, bins = n_bins)
axs[0, 0].set_title('age')

axs[0, 1].hist(df['creatinine_phosphokinase'].values, bins = n_bins)
axs[0, 1].set_title('creatinine_phosphokinase')

axs[0, 2].hist(df['ejection_fraction'].values, bins = n_bins)
axs[0, 2].set_title('ejection_fraction')

axs[1, 0].hist(df['platelets'].values, bins = n_bins)
axs[1, 0].set_title('platelets')

axs[1, 1].hist(df['serum_creatinine'].values, bins = n_bins)
axs[1, 1].set_title('serum_creatinine')

axs[1, 2].hist(df['serum_sodium'].values, bins = n_bins)
axs[1, 2].set_title('serum_sodium')
```

```
plt.show()
```



4. На основании гистограмм определите диапазоны значений для каждого из признаков, а также возле какого значения лежит наибольшее количество наблюдений.

1. Age: диапазон значений от 40 до 95, наибольшее количество наблюдений лежит около 60.
2. creatinine_phosphokinase: диапазон значений от 0 до 8000, наибольшее количество наблюдений лежит около 500.
3. ejection_fraction: диапазон значений от 5 до 80, наибольшее количество наблюдений лежит около 37.
4. platelets: диапазон значений от 0 до 900 000, наибольшее количество наблюдений лежит около 250 000.
5. serum_creatinine: диапазон значений от 0 до 10, наибольшее количество наблюдений лежит около 1.
6. serum_sodium: диапазон значений от 0 до 150, наибольшее количество наблюдений лежит около 137.

5. Так как библиотека Sklearn работает с NumPy массива, то преобразуйте датафрейм к двумерному массиву NumPy, где строка соответствует наблюдению, а столбец признаку

In [13]:
```
data = df.to_numpy(dtype='float')
print(type(data))
```

```
<class 'numpy.ndarray'>
```

Теперь у нас есть данные в виде двухмерного массива numpy

Задание B. Стандартизация данных

1. Подключите модуль Sklearn. Настройте стандартизацию на основе первых 150 наблюдений используя StandardScaler

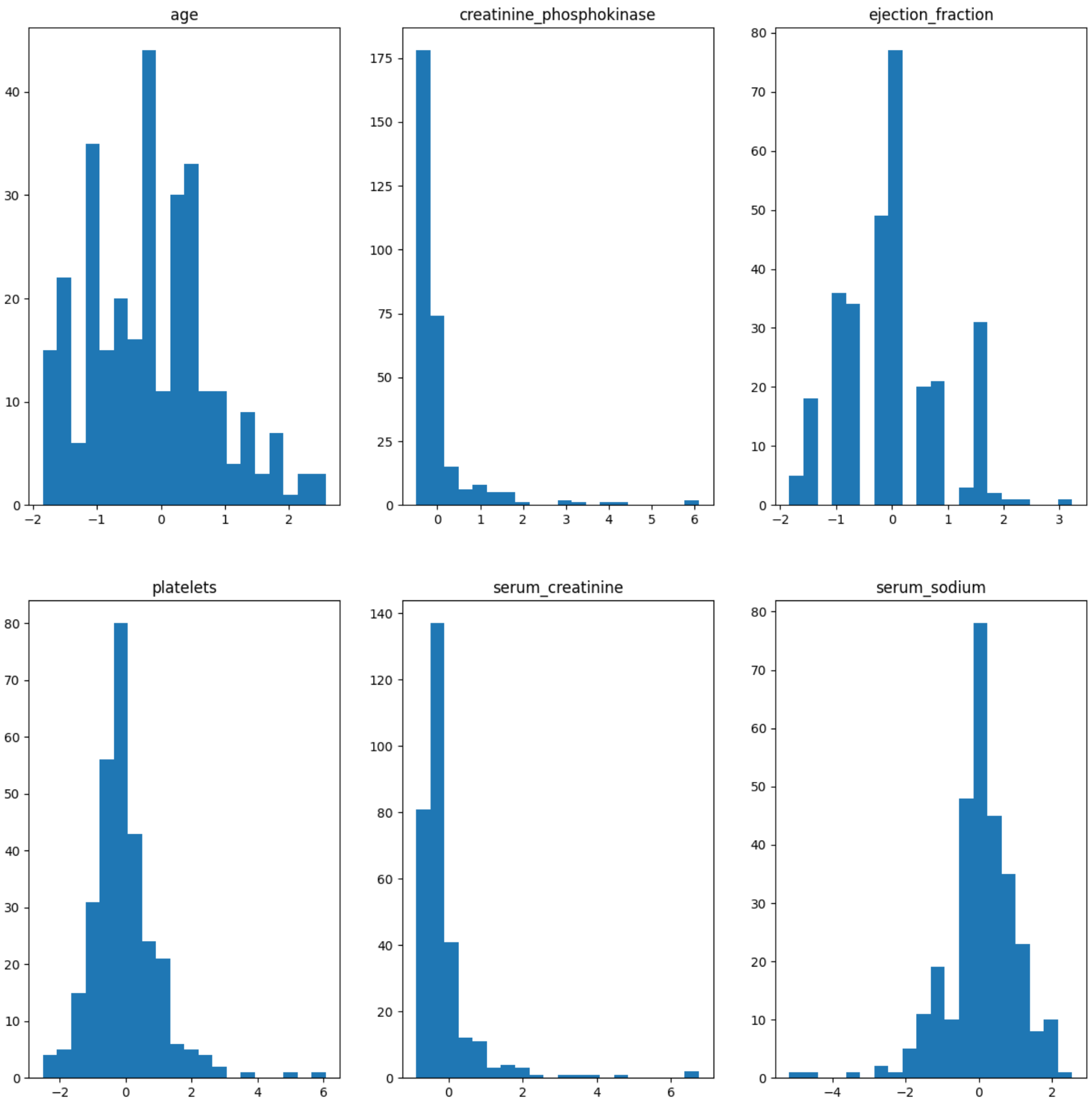
In [14]:
```
scaler = preprocessing.StandardScaler().fit(data[:150,:])
```

2. Стандартизуйте все данные

In [15]:
```
data_scaled = scaler.transform(data)
```

3. Постройте гистограммы стандартизированных данных

In [30]:
```
def build_hist(data):
    fig, axs = plt.subplots(2,3, figsize=(15, 15))
    axs[0, 0].hist(data[:,0], bins = n_bins)
    axs[0, 0].set_title('age')
    axs[0, 1].hist(data[:,1], bins = n_bins)
    axs[0, 1].set_title('creatinine_phosphokinase')
    axs[0, 2].hist(data[:,2], bins = n_bins)
    axs[0, 2].set_title('ejection_fraction')
    axs[1, 0].hist(data[:,3], bins = n_bins)
    axs[1, 0].set_title('platelets')
    axs[1, 1].hist(data[:,4], bins = n_bins)
    axs[1, 1].set_title('serum_creatinine')
    axs[1, 2].hist(data[:,5], bins = n_bins)
    axs[1, 2].set_title('serum_sodium')
    plt.show()

build_hist(data_scaled)
```



4. Сравните данные до и после стандартизации. Опишите, что изменилось и почему.

Теперь большая часть значений сконцентрированы около нуля по каждому признаку. Также заметно уменьшилась дисперсия по каждому признаку.

5. Рассчитайте мат. ожидание и СКО до и после стандартизации. На основании этих значений выведите для каждого признака формулы по которым они стандартизировались.

In [22]:
```
print("До стандартизации")
print("Математическое ожидание по столбцам:", *[round(i, 2) for i in np.mean(data, axis=0)])
print("Дисперсия по столбцам:", *[round(i, 2) for i in np.var(data, axis=0)])
print()
```

```
print("После стандартизации")
print("Математическое ожидание по столбцам:", *[round(i, 2) for i in np.mean(data_scaled, axis=0)])
print("Дисперсия по столбцам:", *[round(i, 2) for i in np.var(data_scaled, axis=0)])
```

```
До стандартизации
Математическое ожидание по столбцам: 60.83 581.84 38.08 263358.03 1.39 136.63
Дисперсия по столбцам: 141.01 938309.88 139.6 9533676546.27 1.07 19.4

После стандартизации
Математическое ожидание по столбцам: -0.17 -0.02 0.01 -0.04 -0.11 0.04
Дисперсия по столбцам: 0.91 0.66 0.82 1.03 0.78 0.94
```

Формула стандартизации:

$$X_{si} = \frac{X_i - \overline{X}}{\sigma_X}$$

Где:

X_{si} - стандартизированное значение i-го элемента из X

$\overline{X}$ - среднее арифметическое X

σ_X - среднеквадратическое отклонение X

Для каждого признака здесь будут среднее значение и среднеквадратическое отклонение по этому признаку соответственно.

6. Сравните значений из формул с полями mean_ и var_ объекта scaler

In [24]:
```
print("scaler.mean_: ", *[round(i, 2) for i in scaler.mean_])
print("scaler.var_: ", *[round(i, 2) for i in scaler.var_])
```

```
scaler.mean_:  62.95 607.15 37.95 266746.75 1.52 136.45
scaler.var_:  155.0 1415488.82 170.02 9252860499.08 1.36 20.61
```

Эти значения достаточно близки к средним значениям и среднеквадратическим отклонениям по столбцам, однако немного отличаются.

Вероятно, отличие возникло из-за того, что данные характеристики вычислены по первым 150 записям, а не по всем записям.

7. Проведите настройку стандартизации на всех данных и сравните с результатами настройки на основании 150 наблюдений

In [25]:
```
all_scaler = preprocessing.StandardScaler().fit(data)
data_all_scaled = all_scaler.transform(data)

print("После стандартизации на 150 строках")
print("Математическое ожидание по столбцам:", *[round(i, 2) for i in np.mean(data_scaled, axis=0)])
print("Дисперсия по столбцам:", *[round(i, 2) for i in np.var(data_scaled, axis=0)])
print()

print("После стандартизации на всех строках")
print("Математическое ожидание по столбцам:", *[round(i, 2) for i in np.mean(data_all_scaled, axis=0)])
print("Дисперсия по столбцам:", *[round(i, 2) for i in np.var(data_all_scaled, axis=0)])
```

```
После стандартизации на 150 строках
Математическое ожидание по столбцам: -0.17 -0.02 0.01 -0.04 -0.11 0.04
Дисперсия по столбцам: 0.91 0.66 0.82 1.03 0.78 0.94

После стандартизации на всех строках
Математическое ожидание по столбцам: 0.0 0.0 -0.0 0.0 0.0 -0.0
Дисперсия по столбцам: 1.0 1.0 1.0 1.0 1.0 1.0
```

После стандартизации на всех строках математическое ожидание ушло в точный ноль, а дисперсия в точную 1.

In [29]:
```
print("Исходные данные")
print("mean: ", *[round(i, 2) for i in np.mean(data, axis=0)])
print("var: ", *[round(i, 2) for i in np.var(data, axis=0)])
print()
print("Стандартизация на 150 строках")
print("scaler.mean_: ", *[round(i, 2) for i in scaler.mean_])
print("scaler.var_: ", *[round(i, 2) for i in scaler.var_])
print()
print("Стандартизация на всех строках")
print("scaler.mean_: ", *[round(i, 2) for i in all_scaler.mean_])
print("scaler.var_: ", *[round(i, 2) for i in all_scaler.var_])
```

```
Исходные данные
mean:  60.83 581.84 38.08 263358.03 1.39 136.63
var:  141.01 938309.88 139.6 9533676546.27 1.07 19.4

Стандартизация на 150 строках
scaler.mean_:  62.95 607.15 37.95 266746.75 1.52 136.45
scaler.var_:  155.0 1415488.82 170.02 9252860499.08 1.36 20.61

Стандартизация на всех строках
scaler.mean_:  60.83 581.84 38.08 263358.03 1.39 136.63
scaler.var_:  141.01 938309.88 139.6 9533676546.27 1.07 19.4
```

Видно, что параметры mean_ и var_ объекта scaler определяются как среднее значение и среднеквадратическое отклонение исходных данных по столбцам соответственно.

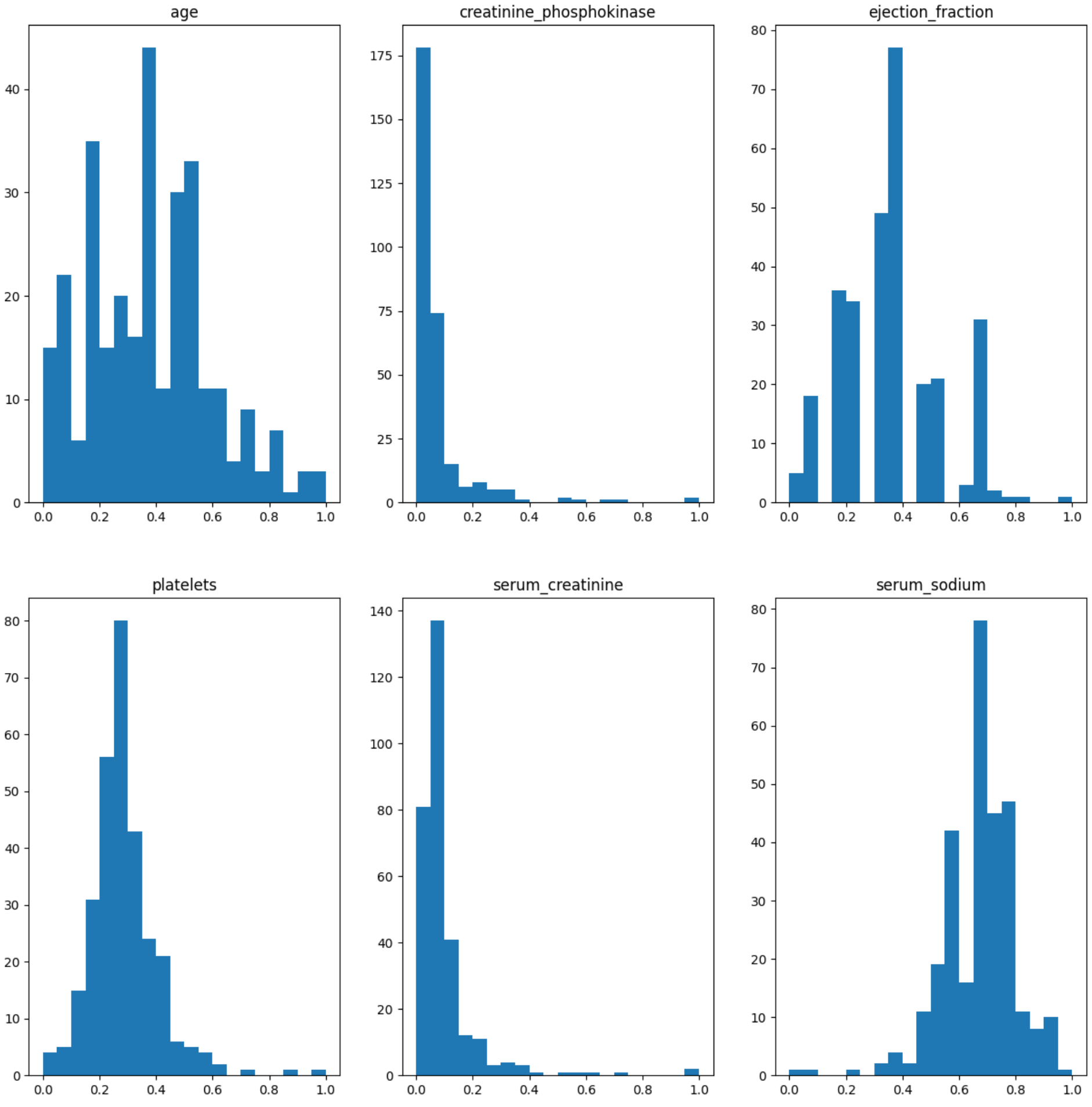
Задание C. Приведение к диапазону

1. Приведите данные к диапазону [0,1], используя MinMaxScaler

In [32]:
```
min_max_scaler = preprocessing.MinMaxScaler().fit(data)
data_min_max_scaled = min_max_scaler.transform(data)
```

2. Постройте гистограммы и сравните с исходными данными

In [33]:
```
build_hist(data_min_max_scaled)
```



Видно, что данные спроецированы (масштабированы) на отрезок от 0 до 1.

3. Через параметры MinMaxScaler определите минимальное и максимальное значение в данных для каждого признака

In [37]:
```
print("min_max_scaler.data_min_: ", *[round(i, 2) for i in min_max_scaler.data_min_])
print("min_max_scaler.data_min_: ", *[round(i, 2) for i in min_max_scaler.data_max_])
```

```
min_max_scaler.data_min_:  40.0 23.0 14.0 25100.0 0.5 113.0
min_max_scaler.data_min_:  95.0 7861.0 80.0 850000.0 9.4 148.0
```

Минимальные и максимальные значения примерно совпадают с моими оценками минимальных и максимальных оценок по гистограммам в задании А.

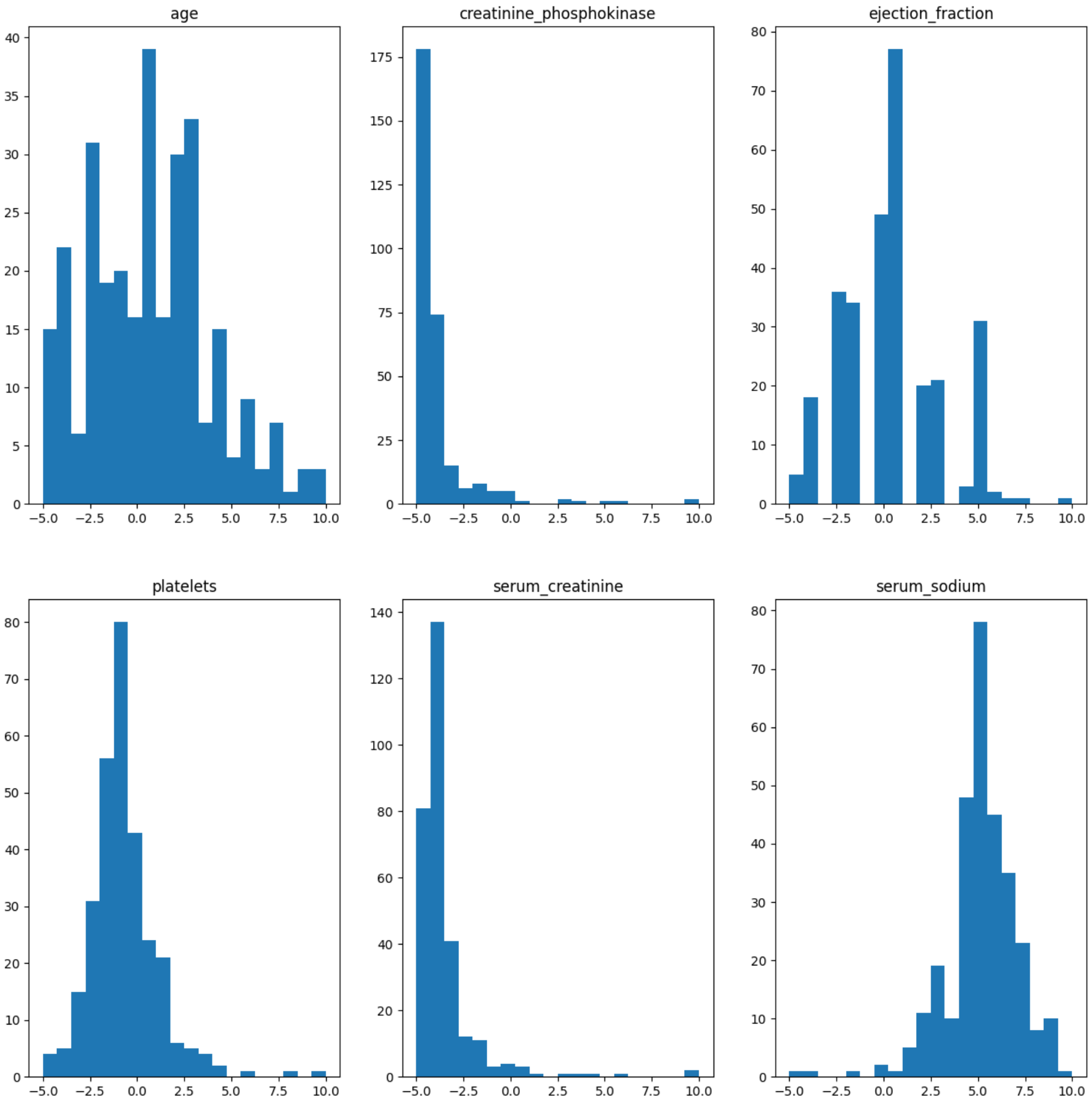
4. Напишите функцию, которая приводит все данные к диапазону [-5 10]. Примените её и постройте гистограммы полученных данных.

Чтобы спроецировать данные на отрезок [a, b], воспользуемся формулой

$$X_{ni} = \frac{X_i - min(X)}{max(X) - min(X)} * (b - a) + a$$

In [55]:
```
def minimax(data, a=0, b=1):
    new_data = copy.deepcopy(data)
    n = new_data.shape[0]
```

```
    m = new_data.shape[1]
    for i in range(m):
        minn = np.min(new_data[:, i])
        maxx = np.max(new_data[:, i])
        new_data[:, i] = (new_data[:, i] - np.full(n, minn))/(maxx - minn)*(b-a) + np.full(n, a)
    return new_data

build_hist(minimax(data, -5, 10))
```



Функцию можно реализовать разными способами, например используя два вложенных цикла. Но такой подход заметно медленнее векторных операций numpy т.к. в векторных операциях numpy используется распараллеливание вычислений. Поэтому я использовал векторные операции.

Как видно на гистограммах, данные спроецировались на нужный отрезок [-5, 10]

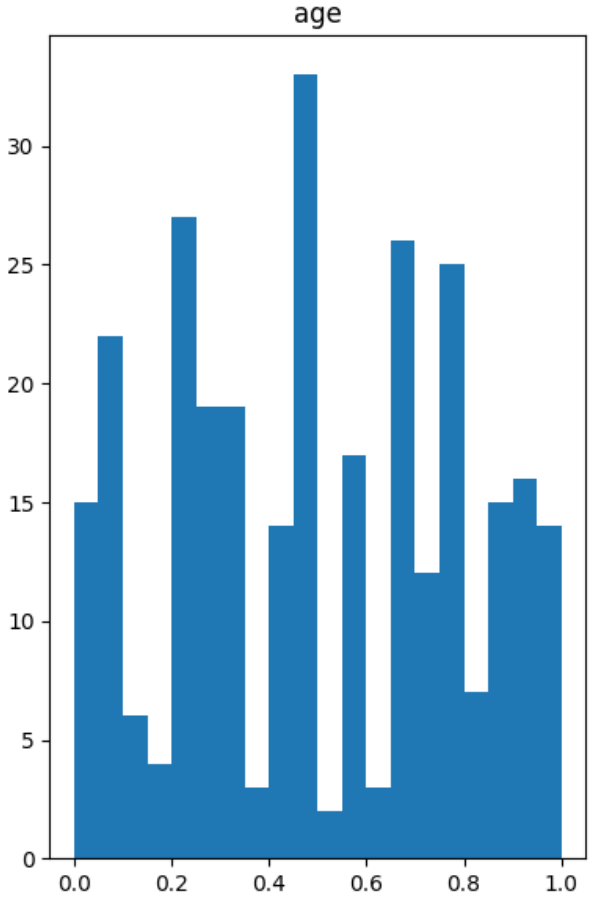
Задание D. Нелинейные преобразования

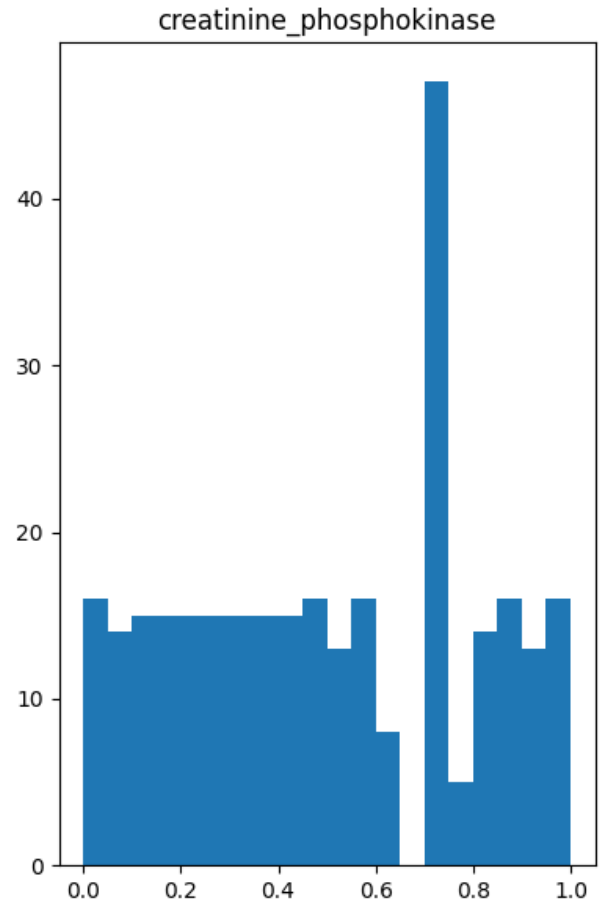
1. Приведите данные к равномерному распределению используя QuantileTransformer

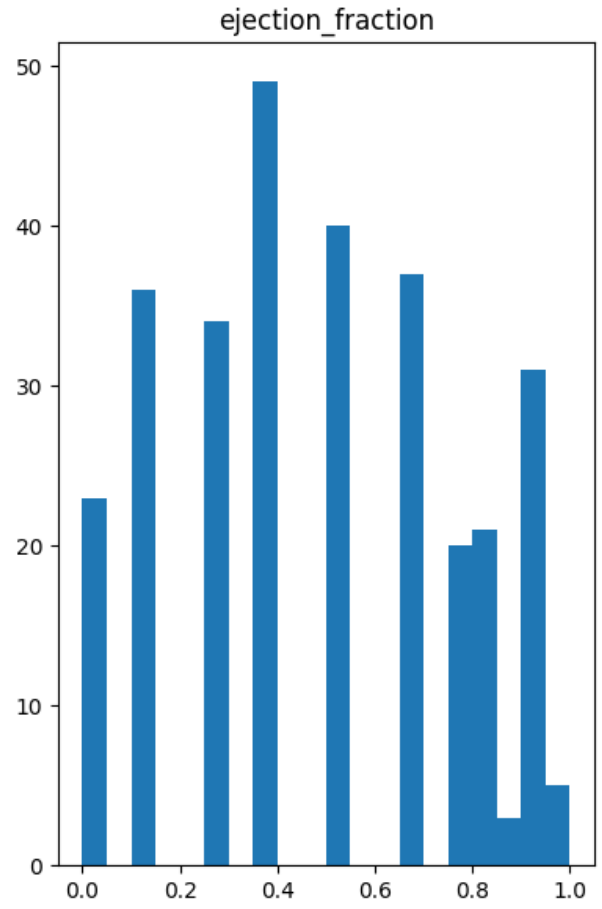
In [58]:
```
quantile_transformer = preprocessing.QuantileTransformer(n_quantiles = 100, random_state=0).fit(data)
data_quantile_scaled = quantile_transformer.transform(data)
```

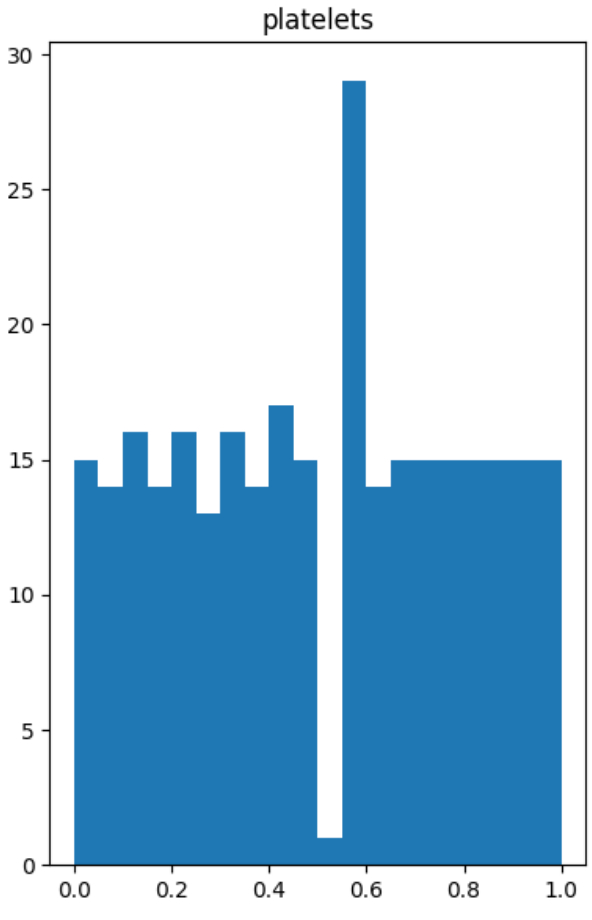
2. Постройте гистограммы и сравните с исходными данными

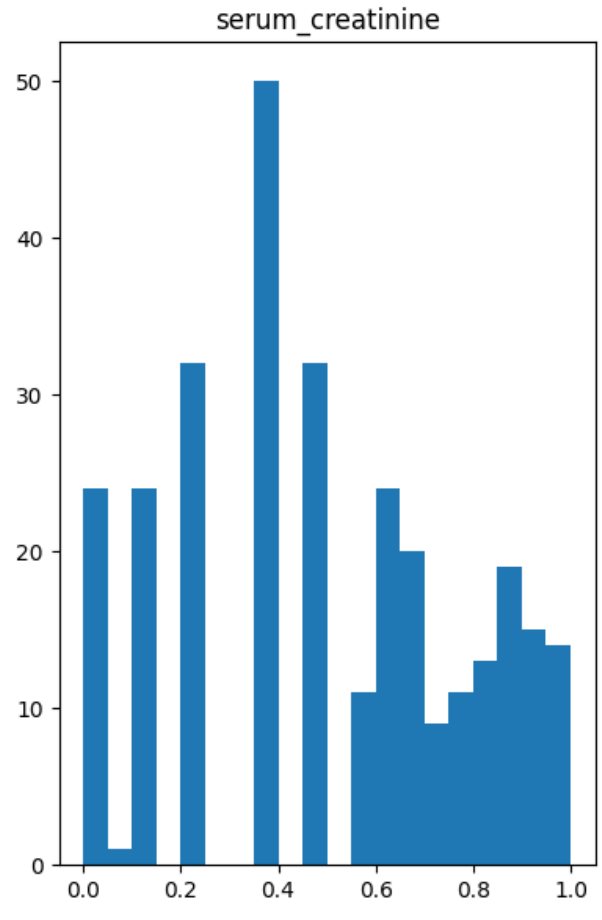
In [59]:
```
build_hist(data_quantile_scaled)
```

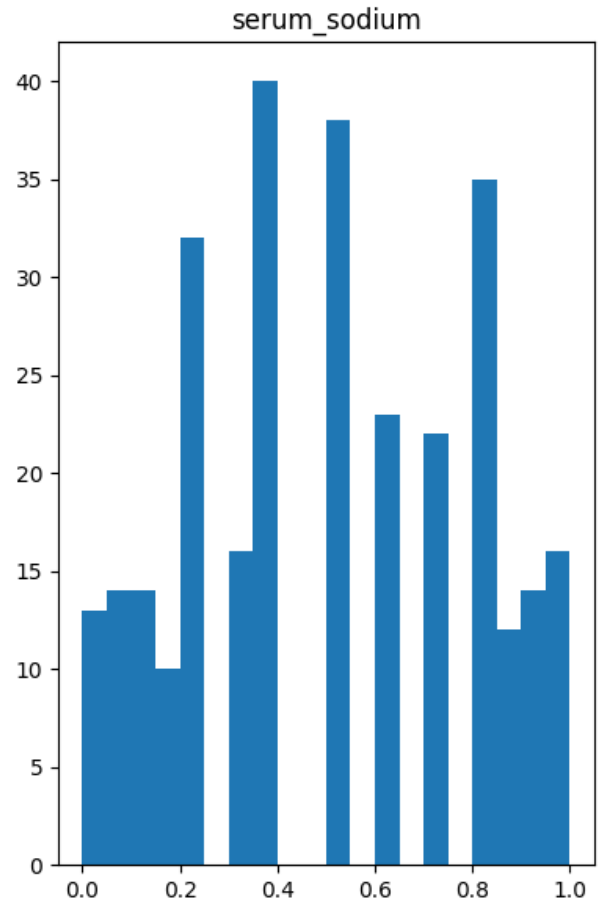












Распределения сильно поменялись, а принцип, по которому это произошло, неочевиден, исходя из гистограмм. Заметно, что теперь все значения принадлежат отрезку [0, 1]

3. Определите, как и на что влияет значение параметра n_quantiles

Согласно документации https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.QuantileTransformer.html

n_quantiles - Количество квантилей для вычисления. Оно соответствует количеству ориентиров, используемых для дискретизации кумулятивной функции распределения. Если n_quantiles больше размера выборки, n_quantiles устанавливается равным размера выборки, поскольку большее количество квантилей не дает лучшего приближения оценки кумулятивной функции распределения.

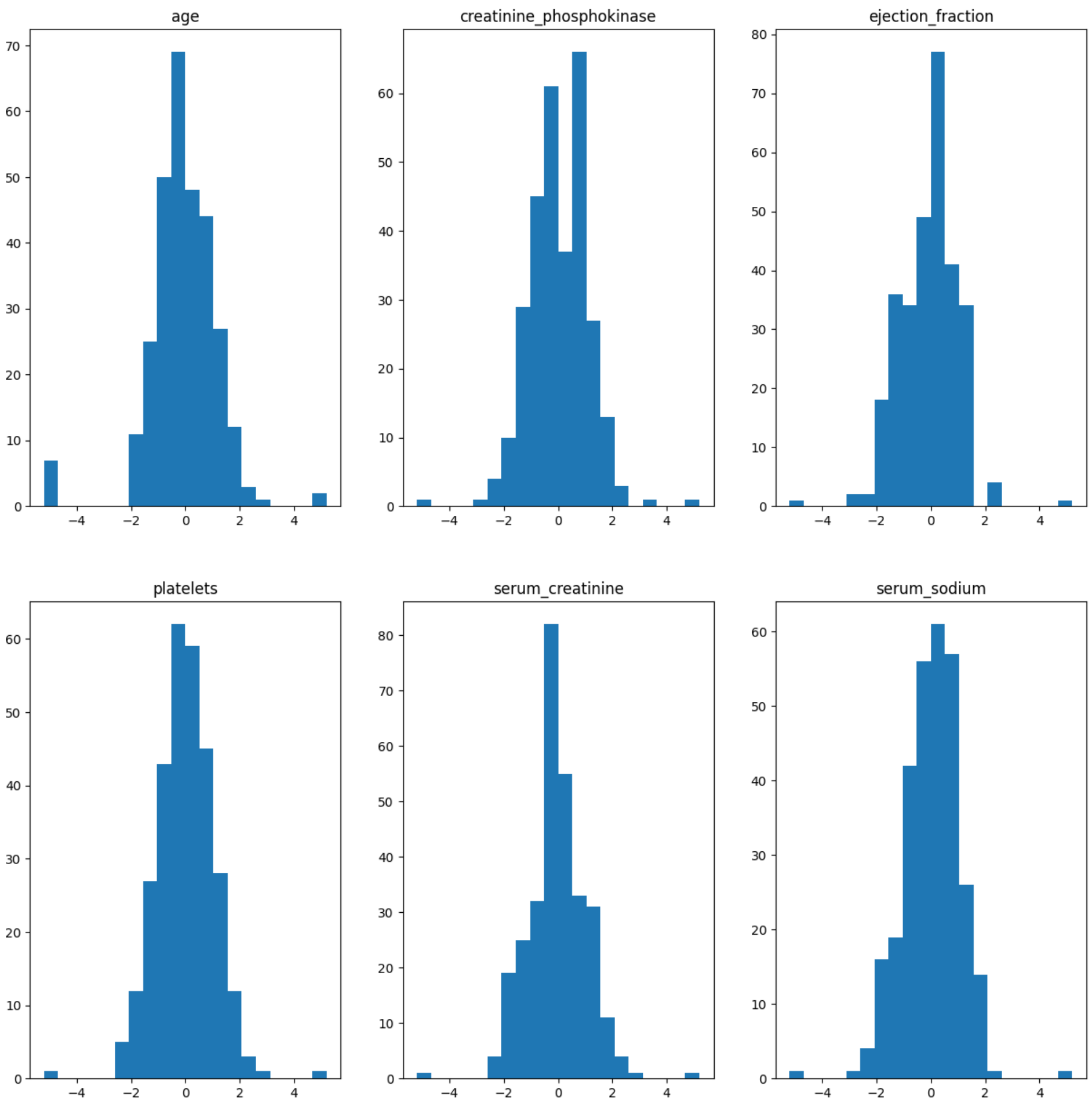
Иными словами чем ближе значение n_quantiles к размеру выборки, тем ближе полученное распределение будет ближе к равномерному. При n_quantiles == 2 полученное распределение почти не будет отличатся от исходного, за исключением проекции на отрезок [0, 1].

4. Приведите данные к нормальному распределению передав в QuantileTransformer параметр output_distribution='normal'

In [67]:
```
quantile_transformer = preprocessing.QuantileTransformer(n_quantiles = 100, random_state=0, output_distribution='normal').fit(data)
data_quantile_scaled = quantile_transformer.transform(data)
```

5. Постройте гистограммы и сравните с исходными данными

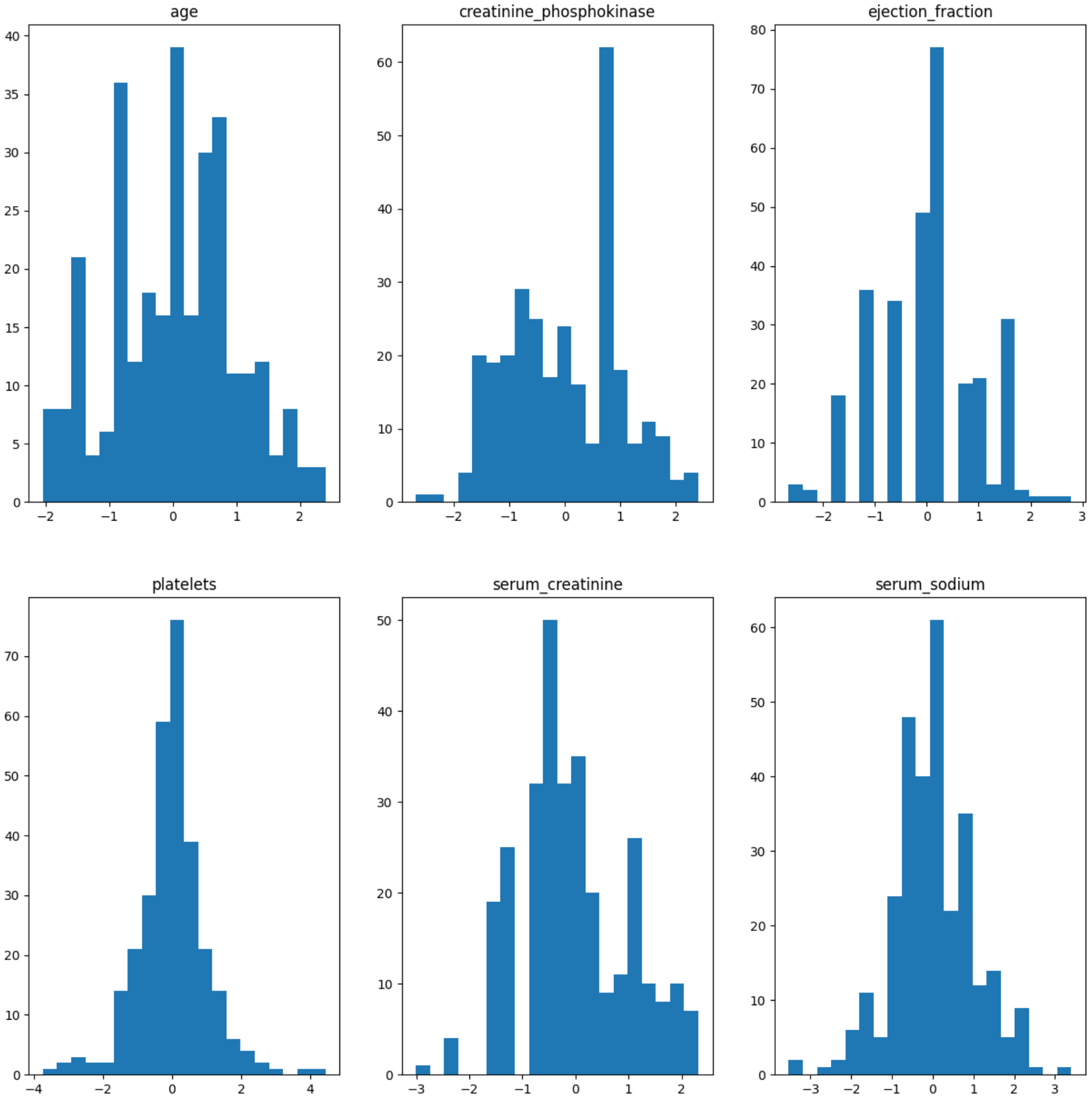
In [66]:
```
build_hist(data_quantile_scaled)
```



Теперь распределение каждого признака близко к нормальному распределение с мат. ожиданием равным нулю.

6. Самостоятельно приведите данные к нормальному распределению используя PowerTransformer. Приведите скрипт операции и гистограммы полученных данных.

In [68]:
```
transformed_data = preprocessing.PowerTransformer().fit_transform(data)
build_hist(transformed_data)
```



Теперь данные тоже приведены к нормальному распределению, но, на мой взгляд, в предыдущий раз распределения получились более близки к нормальным.

Задание E. Дискретизация признаков

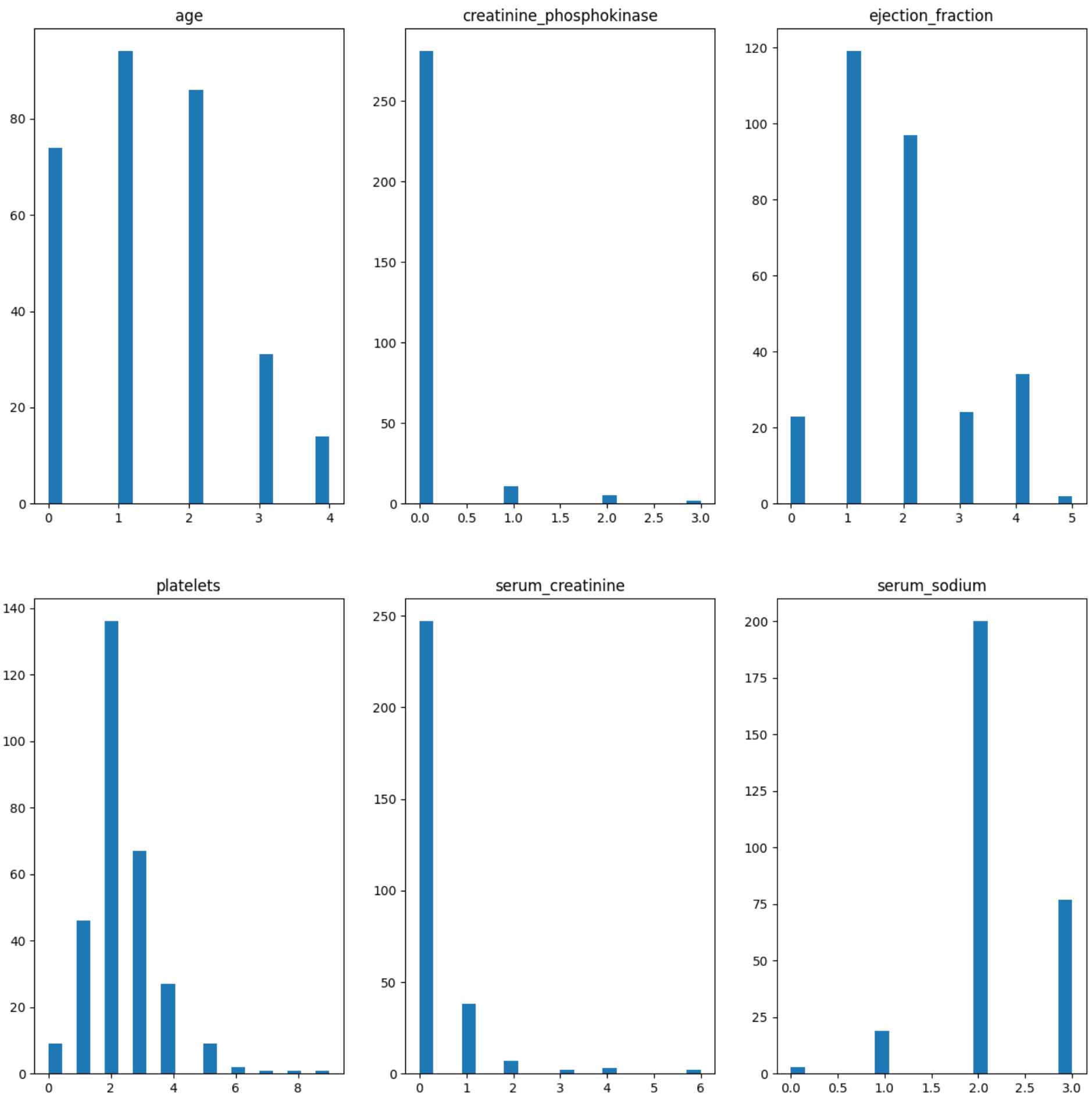
1. Проведите дискретизацию признаков, используя KBinsDiscretizer, на следующее количество диапазонов:
   age - 5
   creatinine_phosphokinase - 4
   ejection_fraction - 6
   platelets - 10
   serum_creatinine - 7
   serum_sodium - 4

In [83]:
```
bins = [5, 4, 6, 10, 7, 4]
new_data = []
for i in range(data.shape[1]):
    discretizer = preprocessing.KBinsDiscretizer(n_bins=bins[i], encode='ordinal', strategy='uniform')
    feature = discretizer.fit_transform(data[:, i].reshape(-1, 1))
    new_data.append(feature)
new_data = np.hstack(new_data)
```

По-отдельности преодразовали каждый признак используя KBinsDiscretizer

2. Постройте гистограммы. Объясните полученные результаты

In [84]:
```
build_hist(new_data)
```



Тут признаки дискретизировались по бинам. Если у нас есть признак X и мы хотим его разбить на b интервалов (бины), то:

$$k = \frac{max(X) - min(X)}{b}$$

$$X_b i = X_i // k$$

Где k - длинна бина.

In [89]:
```
print("Диапазоны признаков\n")
for i in range(data.shape[1]):
    discretizer = preprocessing.KBinsDiscretizer(n_bins=bins[i], encode='ordinal', strategy='uniform')
    feature = discretizer.fit_transform(data[:, i].reshape(-1, 1))
    print(f"Признак {df.columns[i]}:", *discretizer.bin_edges_)
```

```
Диапазоны признаков

Признак age: [40. 51. 62. 73. 84. 95.]
Признак creatinine_phosphokinase: [  23.  1982.5 3942.  5901.5 7861. ]
Признак ejection_fraction: [14. 25. 36. 47. 58. 69. 80.]
Признак platelets: [ 25100. 107590. 190080. 272570. 355060. 437550. 520040. 602530. 685020.
 767510. 850000.]
Признак serum_creatinine: [0.5        1.77142857 3.04285714 4.31428571 5.58571429 6.85714286
 8.12857143 9.4       ]
Признак serum_sodium: [113.   121.75 130.5  139.25 148.  ]
```

Дискретизация признаков полезна, если мы хотим работать с числовым признаков, как с категориальным.

Вывод

Я ознакомился с методами предобработки данных из библиотеки Scikit Learn, изучил математические формулы, которые за ними стоят, документацию этих методов, а также реализовал некоторые из методов предобработки с помощью библиотеки numpy.