

Продолжение кода по композитным материалам

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import plotly.express as px
import tensorflow as tf
import sklearn

from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error, mean_absolute_error
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer, LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

from tensorflow import keras as keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, Activation
from pandas import read_excel, DataFrame, Series
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from tensorflow.keras.models import Sequential
from numpy.random import seed
from scipy import stats
import warnings
warnings.filterwarnings("ignore")

In [2]: df = pd.read_excel(r"C:\Users\Avona\Desktop\Моя ВКР\Itog\itog.xlsx")
```

Проведем предобработку данных

```
In [4]: #Вспомним, что пропуски в данных отсутствуют. Значит, сразу приступаем к работе с выбросами  
# Шаг 1 Удаление выбросов. Посчитаем, сколько значений у нас в каждом столбце выбивающихся из распределения.  
df.isna().sum()  
#Т.к.значений не очень много, их вполне можно исключить.
```

```
Out[4]: Unnamed: 0          0  
Соотношение матрица-наполнитель      0  
Плотность, кг/м3                      0  
модуль упругости, ГПа                  0  
Количество отвердителя, м.%           0  
Содержание эпоксидных групп,%_2       0  
Температура вспышки, С_2              0  
Поверхностная плотность, г/м2         0
```

```
In [5]: #Для удаления выбросов существует 2 основных метода - метод 3-х сигм и межквартильных расстояний. Сравним эти 2 метода.
```

```
metod_3s = 0  
metod_iq = 0
```

```
count_iq = [] # Список, куда записывается количество выбросов по каждой колонке датафрейма методом.
count_3s = [] # Список, куда записывается количество выбросов по каждой колонке датафрейма.
for column in df:
    d = df.loc[:, [column]]
    # методом 3-х сигм
    zscore = (df[column] - df[column].mean()) / df[column].std()
    d['3s'] = zscore.abs() > 3
    metod_3s += d['3s'].sum()
    count_3s.append(d['3s'].sum())
    print(column, '3s', ':', d['3s'].sum())

    # методом межквартильных расстояний
    q1 = np.quantile(df[column], 0.25)
    q3 = np.quantile(df[column], 0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    d['iqr'] = (df[column] < lower) | (df[column] > upper)
```

```
d['iq'] = (df['column'] <= lower) | (df['column'] >= upper)
metod_iq += d['iq'].sum()
count_iq.append(d['iq'].sum())
print(column, ': ', d['iq'].sum())
print('Метод 3-х сигм, выбросов:', metod_3s)
print('Метод межквартильных расстояний, выбросов:', metod_iq)
```

Unnamed: 0 3s : 0
Unnamed: 0 : 0
Соотношение матрица-наполнитель 3s : 0
Соотношение матрица-наполнитель : 0
Плотность, кг/м3 3s : 0
Плотность, кг/м3 : 0
модуль упругости, ГПа 3s : 0
модуль упругости, ГПа : 0
Количество отвердителя, м.% 3s : 0
Количество отвердителя, м.% : 0
Содержание эпоксидных групп,%_2 3s : 0
Содержание эпоксидных групп,%_2 : 0
Температура вспышки, С_2 3s : 0
Температура вспышки, С_2 : 0
Поверхностная плотность, г/м2 3s : 0
Поверхностная плотность, г/м2 : 0
Модуль упругости при растяжении, ГПа 3s : 0
Модуль упругости при растяжении, ГПа : 0
Прочность при растяжении, МПа 3s : 0
Прочность при растяжении, МПа : 0
Потребление смолы, г/м2 3s : 0
Потребление смолы, г/м2 : 0
Угол нашивки 3s : 0
Угол нашивки : 0
Шаг нашивки 3s : 0
Шаг нашивки : 0
Плотность нашивки 3s : 0
Плотность нашивки : 0
Метод 3-х сигм, выбросов: 0
Метод межквартильных расстояний, выбросов: 0

In [6]: # С целью предотвращения удаления особенностей признака или допущения ошибки, посчитаем распределение выбросов по каждому столбцу.

```
In [7]: m = df.copy()
for i in df.columns:
    m[i] = abs((df[i] - df[i].mean()) / df[i].std())
    print(f"sum(m[{i}] > 3) выбросов в признаке {i}")
print(f'Всего sum(sum(m.values > 3))) выброса')
```

0 выбросов в признаке Unnamed: 0
0 выбросов в признаке Соотношение матрица-наполнитель
0 выбросов в признаке Плотность, кг/м3
0 выбросов в признаке модуль упругости, ГПа
0 выбросов в признаке Количество отвердителя, м.%
0 выбросов в признаке Содержание эпоксидных групп,%_2
0 выбросов в признаке Температура вспышки, С_2
0 выбросов в признаке Поверхностная плотность, г/м2
0 выбросов в признаке Модуль упругости при растяжении, ГПа
0 выбросов в признаке Прочность при растяжении, МПа
0 выбросов в признаке Потребление смолы, г/м2
0 выбросов в признаке Угол нашивки
0 выбросов в признаке Шаг нашивки

0 выбросов в признаке Плотность нашивки
Всего 0 выброса

```
In [8]: # Результаты показывают, что выбросы распределены по разным признакам. Какого-то скопления выбросов в одном из признаков не обнаружено
```

```
In [9]: #Создадим переменную со списком всех параметров, в которых есть выбросы
df.columns
column_list_drop = ["Соотношение матрица-наполнитель",
                    "Плотность, кг/м3",
                    "модуль упругости, ГПа",
                    "Количество отвердителя, м.%",
                    "Содержание эпоксидных групп,%_2",
                    "Температура вспышки, С_2",
                    "Поверхностная плотность, г/м2",
                    "Модуль упругости при растяжении, ГПа",
                    "Прочность при растяжении, МПа"]
```

```
In [10]: # Исключим выбросы, очистим данные от выбросов методом межквартильного расстояния (далее 1,5 межквартильных размахов)
# Выбор сделан в пользу этого метода, потому что хотим добиться в данной работе полностью избавления от выбросов, и метод межквартильного расстояния это делает
for i in column_list_drop:
    q75, q25 = np.percentile(df.loc[:, i], [75, 25])
    df = df[(df[i] >= q25) & (df[i] <= q75)]
```

```
intr_qr = q75 - q25
max = q75 + (1.5 * intr_qr)
min = q25 - (1.5 * intr_qr)
df.loc[df[i] < min, i] = np.nan
```

```
In [11]: # Так же для удаления выбросов можно использовать вот эту формулу, но мне привычней вариант выше:  
df.loc[df[i] < min, i] = np.nan  
df.loc[df[i] > max, i] = np.nan
```

```
#df = df[~((df<(min))|(df>(max))).any(axis=1)]  
#df
```

```
m_3s = pd.DataFrame(index=df.index)
for column in df:
    zscore = (df[column] - df[column].mean()) / df[column].std()
    m_3s[column] = (zscore.abs() > 3)
df = df[m_3s.sum(axis=1)==0]
df.shape
```

(922, 14)

```
In [13]: #Посмотрим на сумму выбросов по каждому из столбцов
df.isnull().sum()
#Несколько строк с выбросами можно их удалить.
```

```
Out[13]: Unnamed: 0          0
Соотношение матрица-наполнитель      0
Плотность, кг/м3                      0
модуль упругости, ГПа                  0
Количество отвердителя, м.-%           0
Содержание эпоксидных групп, %_2       0
Температура вспышки, С_2               0
Поверхностная плотность, г/м2          0
Модуль упругости при растяжении, ГПа 0
Прочность при растяжении, МПа         0
Потребление смолы, г/м2               0
Угол нашивки                           0
Шаг нашивки                            0
Плотность нашивки                      0
dtype: int64
```

```
In [14]: #Удаляем строки с выбросами
df = df.dropna(axis=0)
```

```
In [15]: #И еще раз посмотрим на сумму выбросов по каждому из столбцов, чтобы убедиться, что все работают
df.isnull().sum()
```

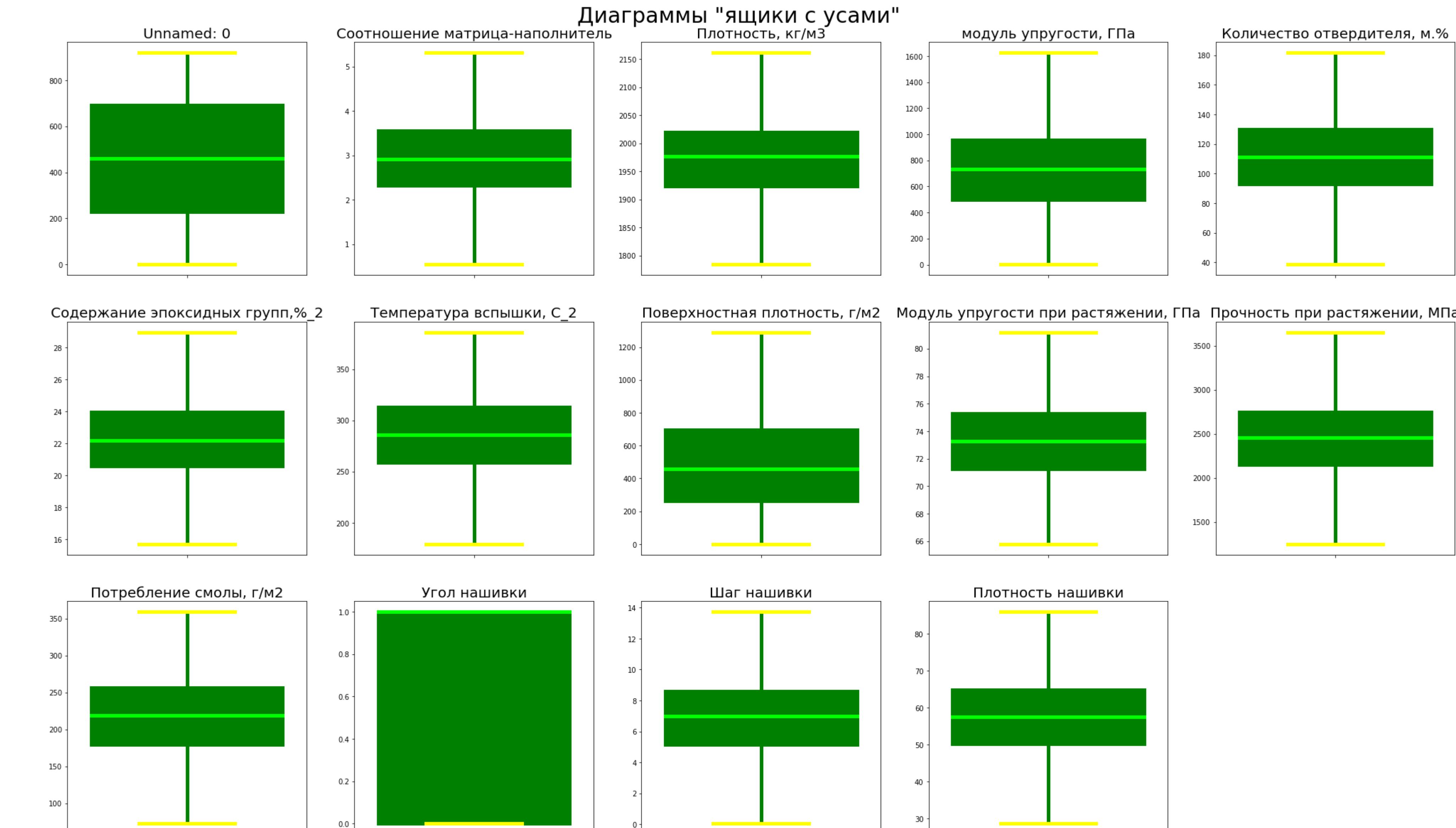
```
Out[15]: Unnamed: 0          0
Соотношение матрица-наполнитель      0
Плотность, кг/м3                      0
модуль упругости, ГПа                  0
Количество отвердителя, м.-%           0
Содержание эпоксидных групп, %_2       0
Температура вспышки, С_2               0
Поверхностная плотность, г/м2          0
Модуль упругости при растяжении, ГПа 0
Прочность при растяжении, МПа         0
Потребление смолы, г/м2               0
Угол нашивки                           0
Шаг нашивки                            0
Плотность нашивки                      0
dtype: int64
```

```
In [16]: #Посмотрим информацию о "чистом" датасете после удаления пропусков. Видим, что строк стало меньше
df.info()
#Несколько строк с выбросом в датасете осталось 936 строк и 13 колонок
```

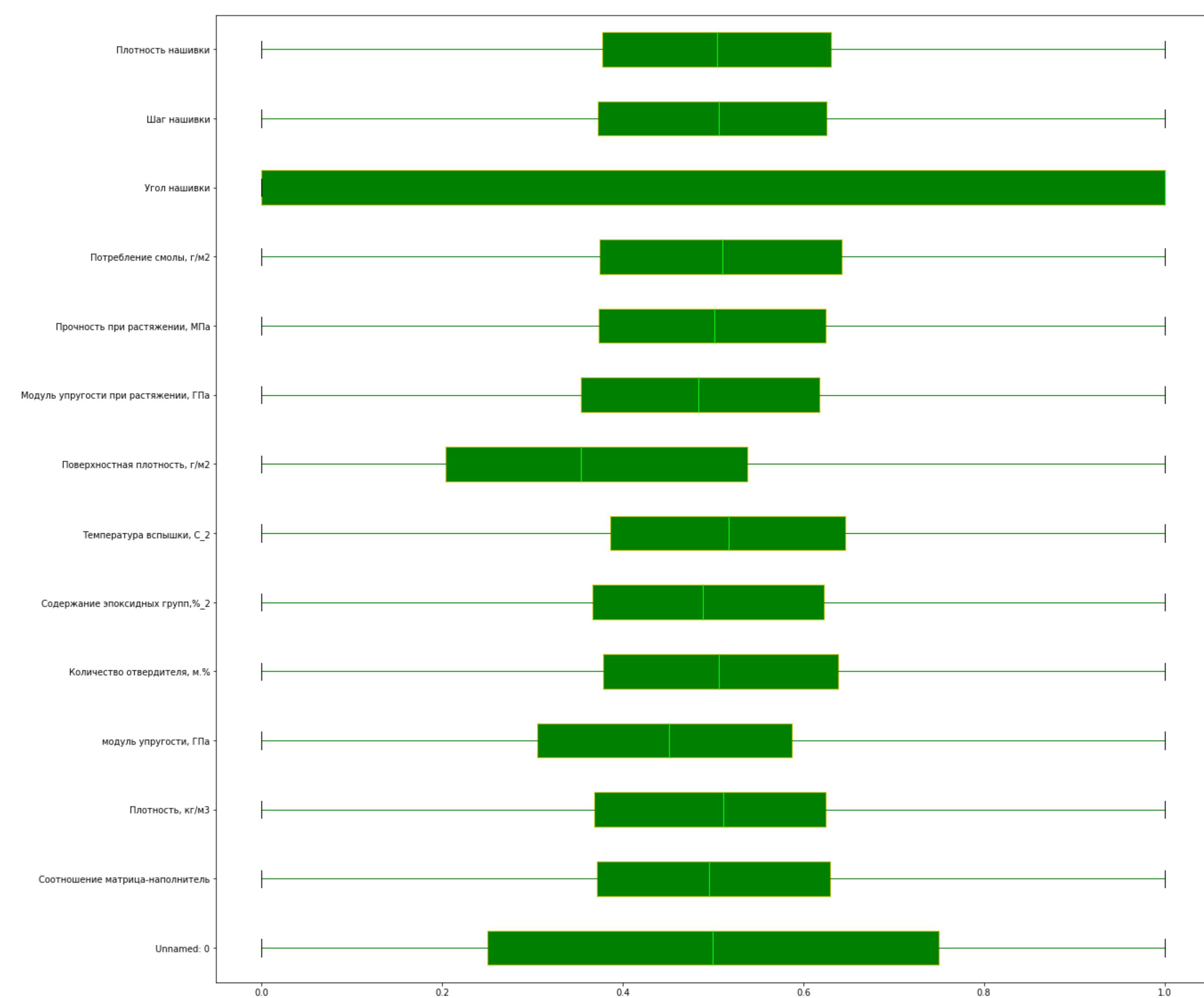
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 936 entries, 0 to 921
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
 ---  -- 
 0   Unnamed: 0        936 non-null   float64
 1   Соотношение матрица-наполнитель  936 non-null   float64
 2   Плотность, кг/м3                 936 non-null   float64
 3   модуль упругости, ГПа           936 non-null   float64
 4   Количество отвердителя, м.-%    936 non-null   float64
 5   Содержание эпоксидных групп, %_2 936 non-null   float64
 6   Температура вспышки, С_2        936 non-null   float64
 7   Поверхностная плотность, г/м2    936 non-null   float64
 8   Модуль упругости при растяжении, ГПа 936 non-null   float64
 9   Прочность при растяжении, МПа    936 non-null   float64
 10  Потребление смолы, г/м2        936 non-null   float64
 11  Угол нашивки                  936 non-null   int64  
 12  Шаг нашивки                   936 non-null   float64
 13  Плотность нашивки             936 non-null   float64
dtypes: float64(12), int64(2)
memory usage: 188.8 KB
```

```
In [17]: # Яшки с усами (Вероятность)
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # инициализация plot counter
```

```
plt.figure(figsize = (35,35))
plt.suptitle('Диаграммы "яшки с усами"', y = 0.9 , fontsize = 30)
for col in df.columns:
    plt.subplot(a, b, c)
    plt.figure(figsize=(7,5))
    sns.boxplot(data = df, x = df[col], fliersize = 15, linewidth = 5, boxprops = dict(facecolor = 'y', color = 'g'), medianprops = dict(color = "lime"), whiskerprops = dict(color="yellow"), capprops = dict(color = "y", markeredgecolor = "lime"))
    plt.xlabel(None)
    plt.ylabel(None)
    plt.title(df[col].name, loc='center')
    plt.show()
    c += 1
```



```
In [18]: #Построим "яшки с усами" и наблюдаем все еще наличие выбросов
scaler = MinMaxScaler()
scaler.fit(df)
plt.figure(figsize = (20, 20))
#Построим на "яшки с усами", чтобы наглядно увидеть, что выбросов нет, но они есть
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels = df.columns, patch_artist = True, meanline = True, vert = False, boxprops = dict(facecolor = 'y', color = 'y'), medianprops = dict(color = "lime"), whiskerprops = dict(color = "black"), capprops = dict(color = "y", markeredgecolor = "maroon"))
plt.show()
```



In [19]: # На графике выше мы видим выбросы в некоторых скобках. Они всё еще есть, поэтому подберем удаление выбросов

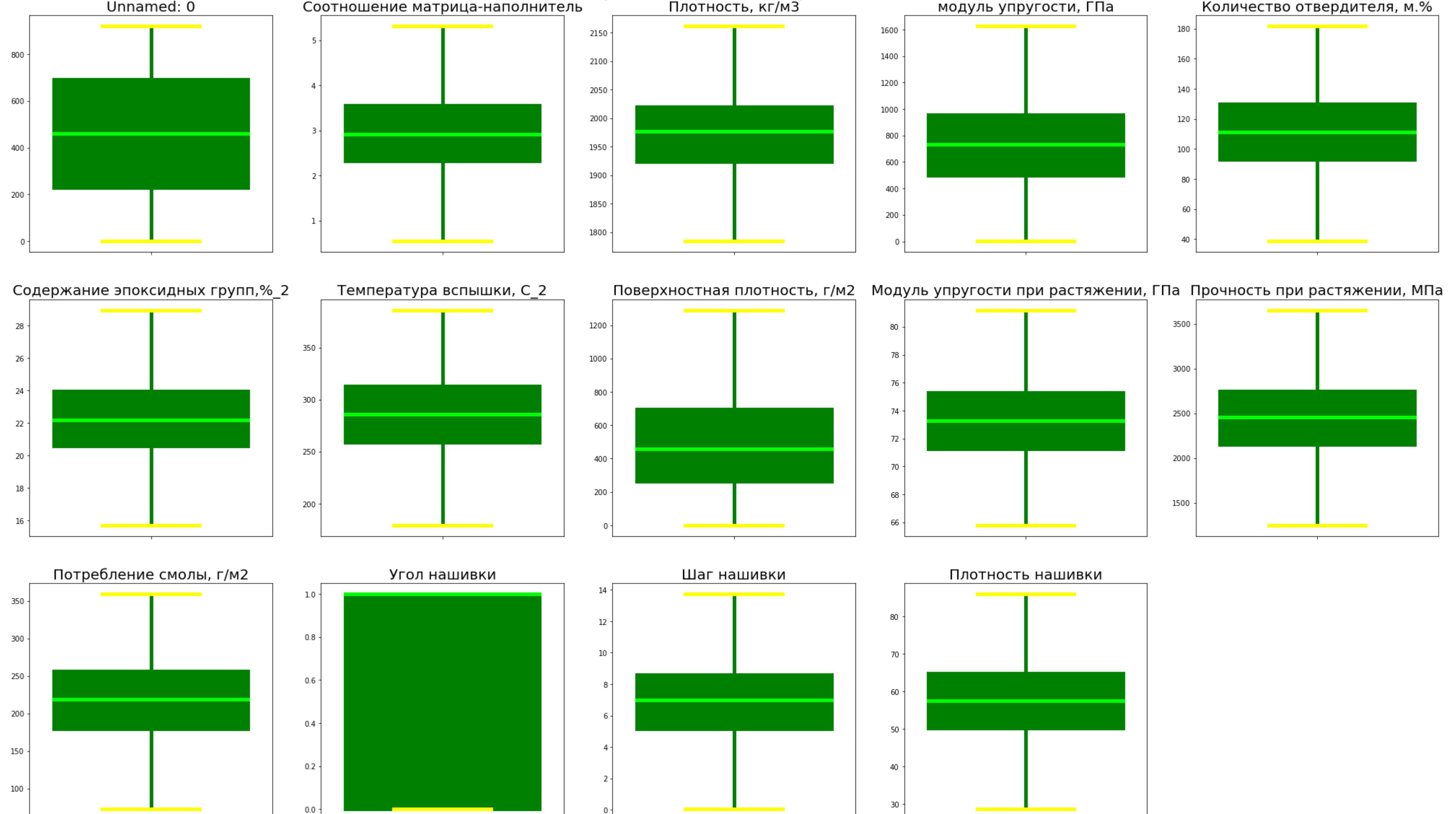
```
for i in column_list_drop:
    q25, q75 = np.percentile(df.loc[:, i], [25, 75])
    intr_qr = q75 - q25
    max = q75 + (1.5 * intr_qr)
    min = q25 - (1.5 * intr_qr)
    df.loc[df[i] < min] = np.nan
    df.loc[df[i] > max] = np.nan
```

In [20]: # Ящики с усами (Вероят. Варианты)

```
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # инициализация plot counter

plt.figure(figsize=(35,35))
plt.suptitle('Диаграммы "ящики с усами"', y = 0.9,
            fontsize = 30)
for col in df.columns:
    plt.subplot(a, b, c)
    plt.figure(figsize=(7,5))
    plt.boxplot(data = df, flierprops = dict(facecolor = 'y', color = 'g'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = "g"), capprops = dict(color="yellow"), flierprops = dict(color = "y", markeredgecolor = "lime"))
    plt.xlabel(None)
    plt.title(col, size = 20)
    plt.show()
    c += 1
```

Диаграммы "ящики с усами"



In [21]: df.skew()# ассиметрия по всем колонкам. Никак не использовал в дальнейшем.

```
Out[21]:
Unnamed: 0      0.000000
Соотношение матрица-наполнитель  0.049923
Поверхностная плотность, Г/м2     0.001483
Модуль упругости, ГПа             0.070099
Количество отвердителя, м.%       -0.108432
Содержание эпоксидных групп, %_2  0.045384
Температура вспышки, С_2          0.082646
Потребление смолы, Г/м2           0.230770
Модуль упругости при растяжении, ГПа  0.119982
Прочность при растяжении, МПа     0.043496
Потребление смолы, Г/м2           -0.014619
Угол нашивки                      0.043465
Шаг нашивки                        0.043469
Плотность нашивки                  -0.035787
dtype: float64
```

In [22]: df.kurt()# эксцесс по всем колонкам. Никак не использовал в дальнейшем

```

Out[22]: Unnamed: 0           -1.388988
Соотношение матрица-наполнитель   -0.314964
Плотность, кг/м3                      -0.218122
Модуль упругости, ГПа                  -0.370833
Количество отвердителя, м.%            -0.247936
Содержание эпоксидных групп,%_2      -0.315152
Температура вспышки, С_2              -0.380182
Поверхностная плотность, г/м2          -0.544512
Модуль упругости при растяжении, ГПа -0.328738
Прочность при растяжении, МПа        -0.423721
Потребление смолы, г/м2                -0.423731
Угол наклона                           -0.402459
Шаг нашивки                            -0.111678
Плотность нашивки                     -0.209895
dtype: float64

In [23]: #Проверим сумму выбросов по каждому из столбцов
df.isnull().sum()

Out[23]: Unnamed: 0           0
Соотношение матрица-наполнитель   0
Плотность, кг/м3                      0
Модуль упругости, ГПа                  0
Количество отвердителя, м.%            0
Содержание эпоксидных групп,%_2      0
Температура вспышки, С_2              0
Поверхностная плотность, г/м2          0
Модуль упругости при растяжении, ГПа -0
Прочность при растяжении, МПа        0
Потребление смолы, г/м2                0
Угол наклона                           0
Шаг нашивки                            0
Плотность нашивки                     0
dtype: int64

In [24]: #И снова удалаем строки, которые содержат выбросы
df = df.dropna(axis=0)

In [25]: #Третий раз проверим сумму выбросов по каждому столбцу
df.isnull().sum()

Out[25]: Unnamed: 0           0
Соотношение матрица-наполнитель   0
Плотность, кг/м3                      0
Модуль упругости, ГПа                  0
Количество отвердителя, м.%            0
Содержание эпоксидных групп,%_2      0
Температура вспышки, С_2              0
Поверхностная плотность, г/м2          0
Модуль упругости при растяжении, ГПа -0
Прочность при растяжении, МПа        0
Потребление смолы, г/м2                0
Угол наклона                           0
Шаг нашивки                            0
Плотность нашивки                     0
dtype: int64

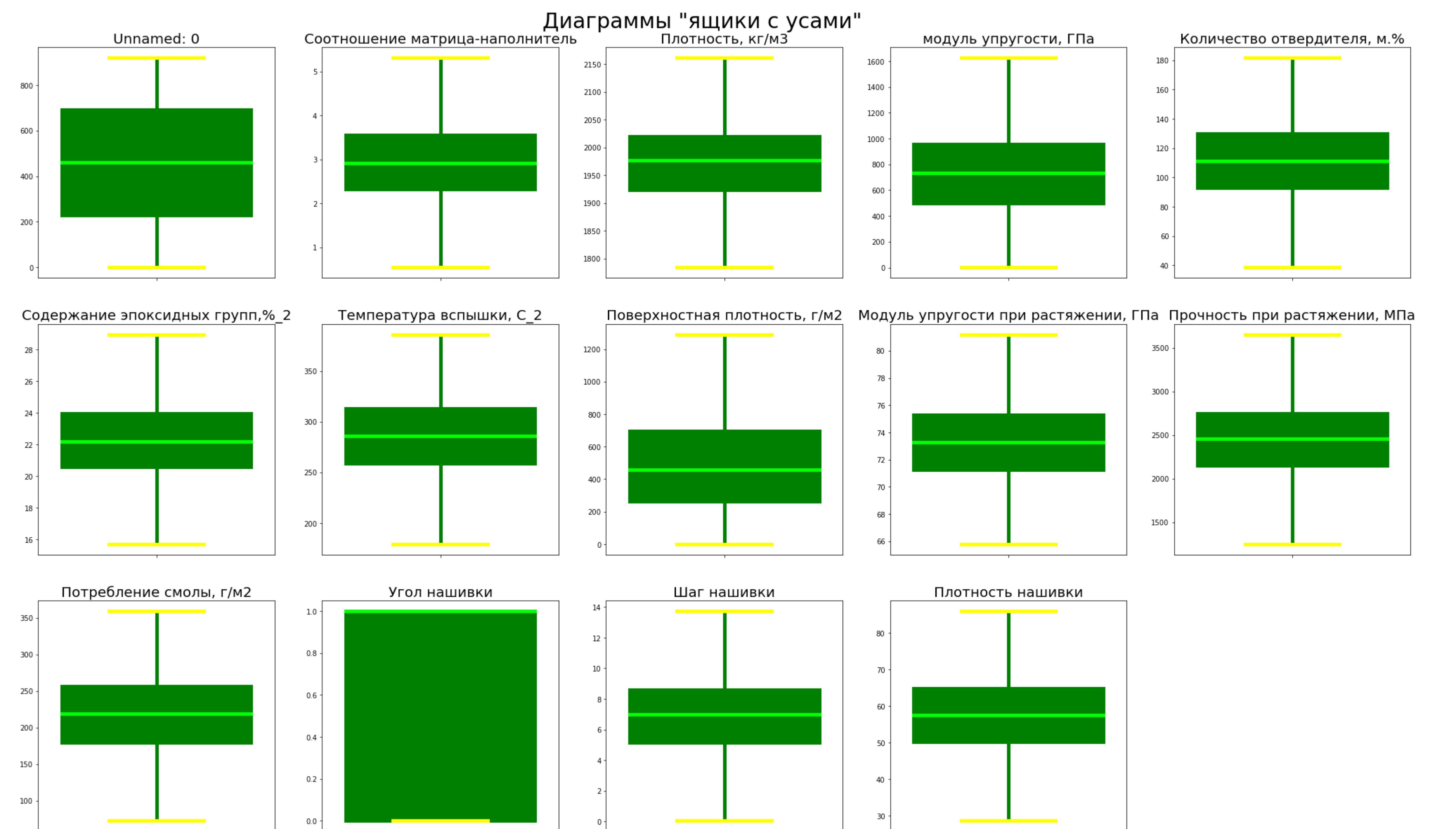
In [26]: #Просмотрим информацию о нашем датасете после еще одного удаления пропусков. Видим, что строк стало еще меньше
df.info()

class 'pandas.core.frame.DataFrame'>
Int64Index: 922 entries, 0 to 921
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   Unnamed: 0       922 non-null    int64  
 1   Соотношение матрица-наполнитель  922 non-null    float64 
 2   Плотность, кг/м3                  922 non-null    float64 
 3   Модуль упругости, ГПа             922 non-null    float64 
 4   Количество отвердителя, м.%       922 non-null    float64 
 5   Содержание эпоксидных групп,%_2  922 non-null    float64 
 6   Температура вспышки, С_2          922 non-null    float64 
 7   Поверхностная плотность, г/м2     922 non-null    float64 
 8   Модуль упругости при растяжении, ГПа - float64 
 9   Прочность при растяжении, МПа    922 non-null    float64 
 10  Потребление смолы, г/м2           922 non-null    float64 
 11  Угол наклона                     922 non-null    float64 
 12  Шаг нашивки                      922 non-null    float64 
 13  Плотность нашивки                922 non-null    float64 
dtypes: float64(12), int64(2)
memory usage: 108.8 KB

In [27]: #В первый раз построим на "лишки с усами"
scaler = MinMaxScaler()
df = pd.DataFrame(scaler.fit_transform(df))
sns.set(style="whitegrid")
plt.figure(figsize = (20, 20))
#Ребордукция "лишек"
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels = df.columns, patch_artist = True, meanline = True, vert = False, boxprops = dict(facecolor = 'g', color = 'y'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = "g"), capprops = dict(color = "black"), flierprops = dict(color = "y", markeredgecolor = "maroon"))
plt.show()



```



In [29]: # И снова видим, что выбросы остались в некоторых исходных данных. Это вообще когда-нибудь закончится???

```
In [30]: # Подготовка процедуры с выбросами еще раз. Надеюсь последний
for i in column_list_drop:
    q75, q25 = np.percentile(df.loc[:,i],[75, 25])
    max = q75 + (1.5*intc_qr)
    min = q25 - (1.5*intc_qr)
    df.loc[df[i] < min,i] = np.nan
    df.loc[df[i] > max,i] = np.nan
```

In [31]: #Еще раз проверим сумму выбросов в каждом столбце
df.isnull().sum()

```
Out[31]:
Unnamed: 0          0
Соотношение матрица-наполнитель      0
Плотность, кг/м³          0
модуль упругости, ГПа          0
Количество отвердителя, м.%, 0
Содержание эпоксидных групп, %_2      0
Температура вспышки, С_2          0
Поверхностная плотность, г/м²          0
Модуль упругости при растяжении, ГПа      0
Прочность при растяжении, МПа          0
Потребление смолы, г/м²          0
угол нашивки          0
шаг нашивки          0
плотность нашивки          0
dtype: int64
```

In [32]: #Еще раз удаляем строки с выбросами
df = df.dropna(axis=0)

```
In [33]: #Избавляемся от строк с выбросами
df.isnull().sum()

Out[33]:
Unnamed: 0          0
Соотношение матрица-наполнитель      0
Плотность, кг/м³          0
модуль упругости, ГПа          0
Количество отвердителя, м.%, 0
Содержание эпоксидных групп, %_2      0
Температура вспышки, С_2          0
Поверхностная плотность, г/м²          0
Модуль упругости при растяжении, ГПа      0
Прочность при растяжении, МПа          0
Потребление смолы, г/м²          0
угол нашивки          0
шаг нашивки          0
плотность нашивки          0
dtype: int64
```

In [34]: #Проверяем на чистый датасет
df.info()

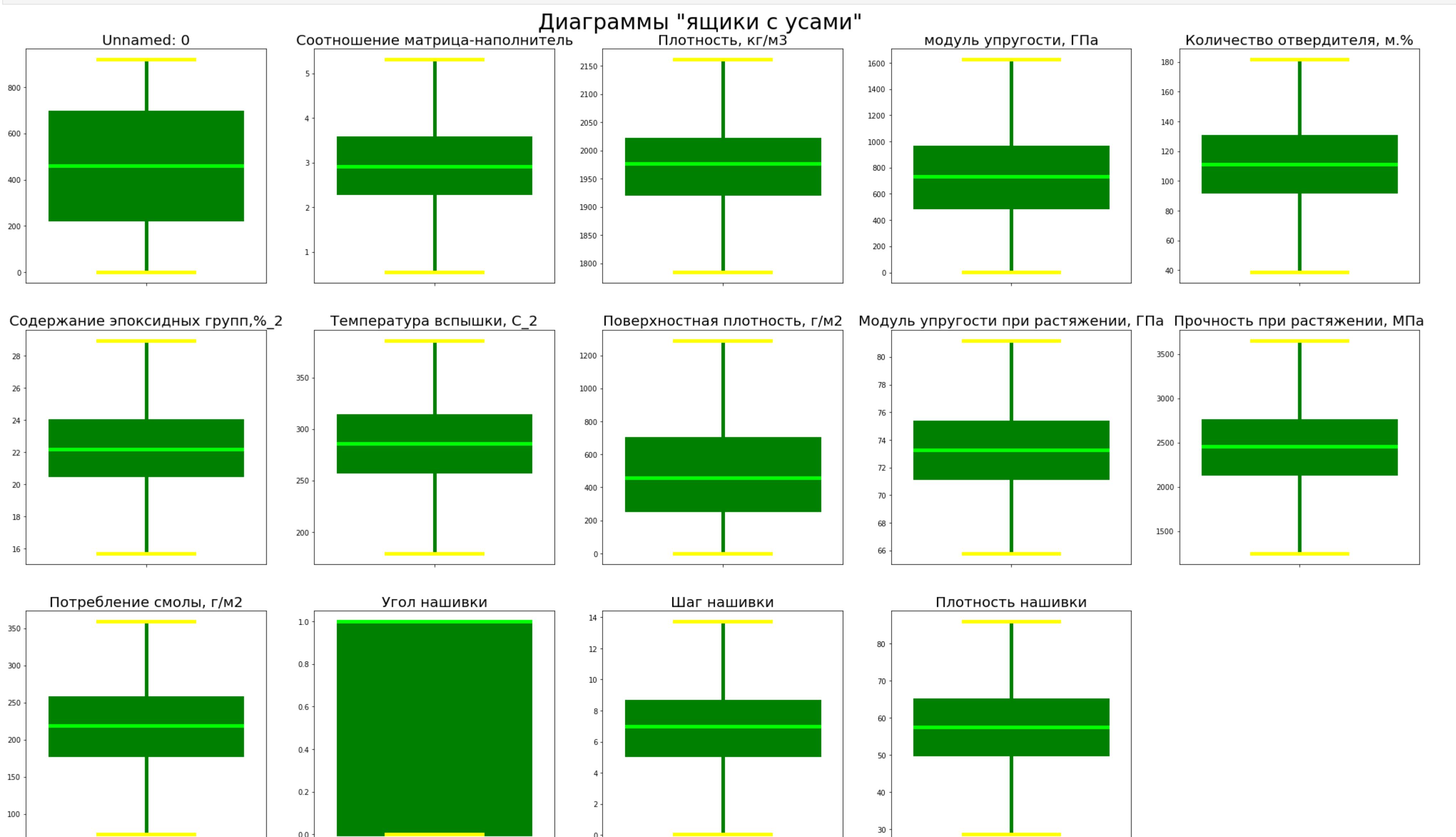
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 922 entries, 0 to 921
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   Unnamed: 0        922 non-null    int64  
 1   Соотношение матрица-наполнитель  922 non-null    float64 
 2   Плотность, кг/м³          922 non-null    float64 
 3   модуль упругости, ГПа          922 non-null    float64 
 4   Количество отвердителя, м.%, 922 non-null    float64 
 5   Содержание эпоксидных групп, %_2 922 non-null    float64 
 6   Температура вспышки, С_2          922 non-null    float64 
 7   Поверхностная плотность, г/м²          922 non-null    float64 
 8   Модуль упругости при растяжении, ГПа 922 non-null    float64 
 9   Прочность при растяжении, МПа          922 non-null    float64 
 10  Потребление смолы, г/м²          922 non-null    float64 
 11  Угол нашивки          922 non-null    int64  
 12  шаг нашивки          922 non-null    float64 
 13  плотность нашивки          922 non-null    float64 
dtypes: float64(12), int64(2)
memory usage: 108.8 kB
```

```
In [35]: # "наши и узкие"(boxсплайны) (первый вариант)
scaler = MinMaxScaler()
scaler.fit(df)
plt.figure(figsize = (20, 20))
plt.suptitle('Диаграммы "ящики с усами"', y = 0.9,
            fontsize = 30)
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels = df.columns, patch_artist = True, meanline = True, vert = False, boxprops = dict(facecolor = 'g', color = 'y'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = "g"), capprops = dict(color = "black"), flierprops = dict(color = "y", markeredgecolor = "maroon"))
plt.show()
```

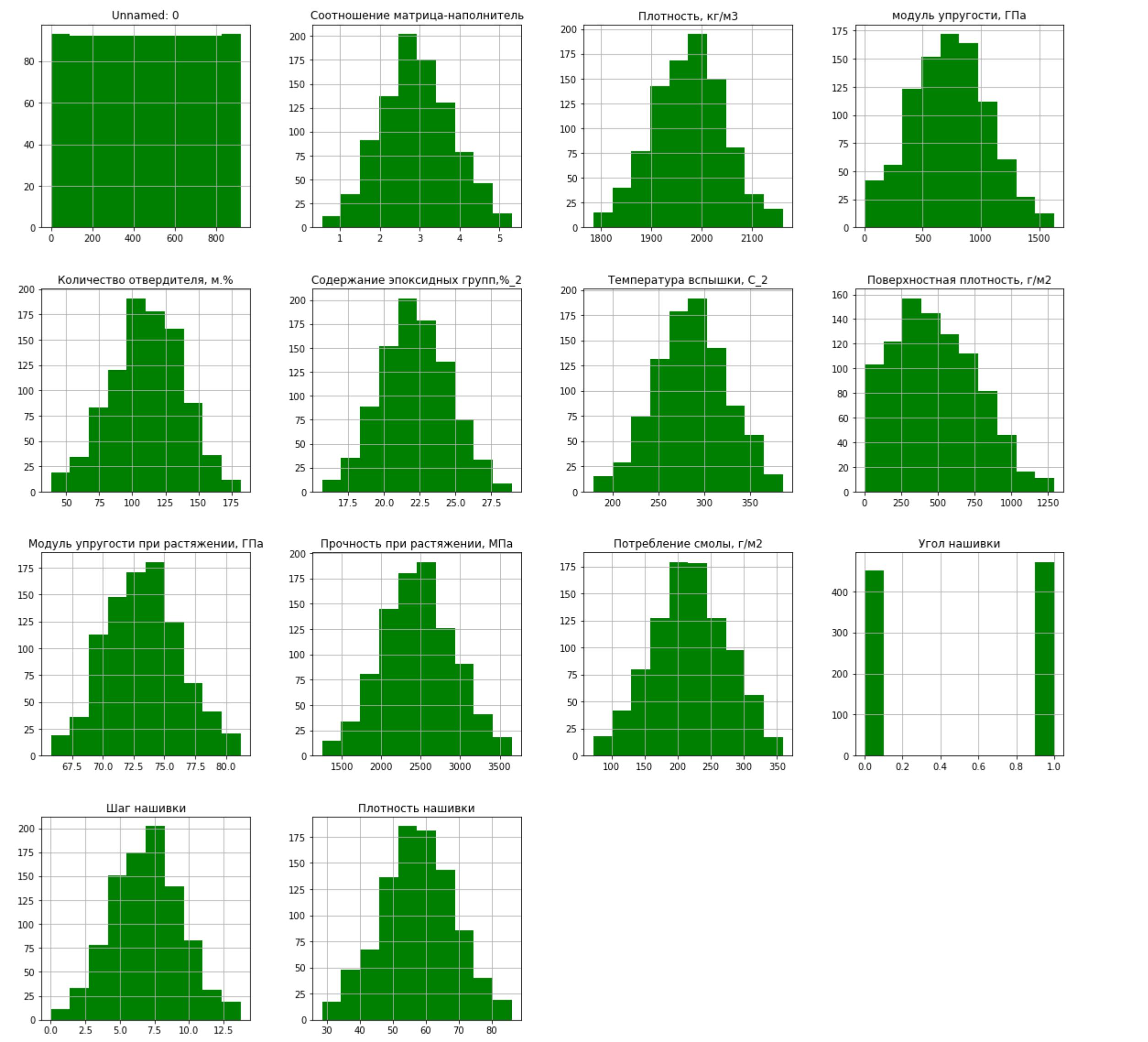
Диаграммы "ящики с усами"



```
In [36]: # Ящики с усами (Ворон Борис)
# a = 5 # количество строк
# b = 5 # количество столбцов
# c = 1 # инициализация plot counter
plt.figure(figsize = (35,35))
plt.suptitle("Диаграммы \"ящики с усами\"", y = 0.9 , fontsize = 30)
for col in df.columns:
    plt.figure(figsize=(7,5))
    sns.boxplot(data = df, y = df[col], fliersize = 15, linewidth = 5, boxprops = dict(facecolor = 'y', color = 'g'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = "g"), capprops = dict(color = "yellow"), flierprops = dict(color = "y", markeredgecolor = "lime"))
    plt.ylabel(None)
    plt.title(col, size = 20)
    plt.show()
c += 1
```



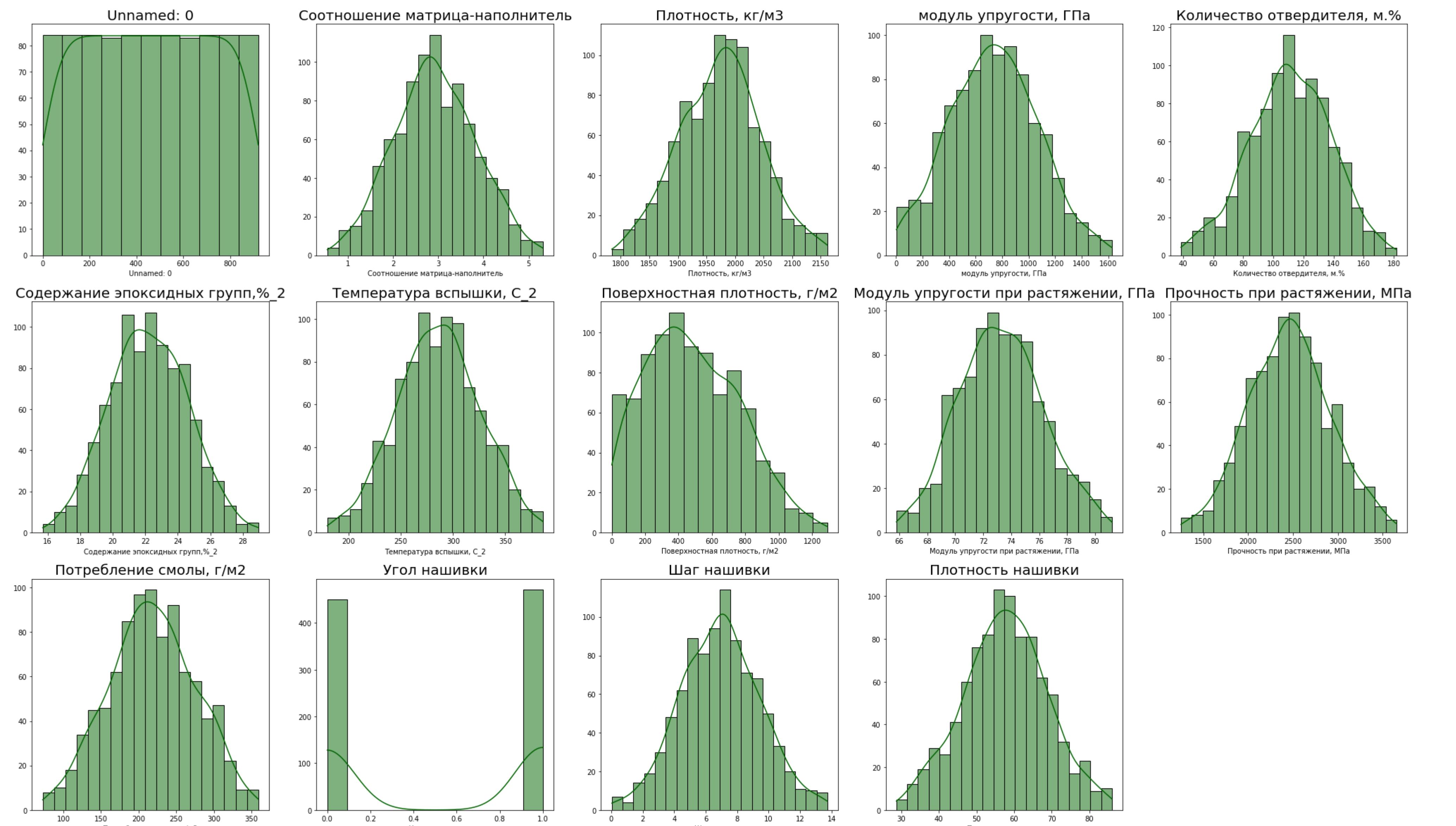
```
In [37]: # Построим гистограммы распределения каждой из переменных без нормализации
df.hist(figsize = (20,20), color = "g")
plt.show()
```



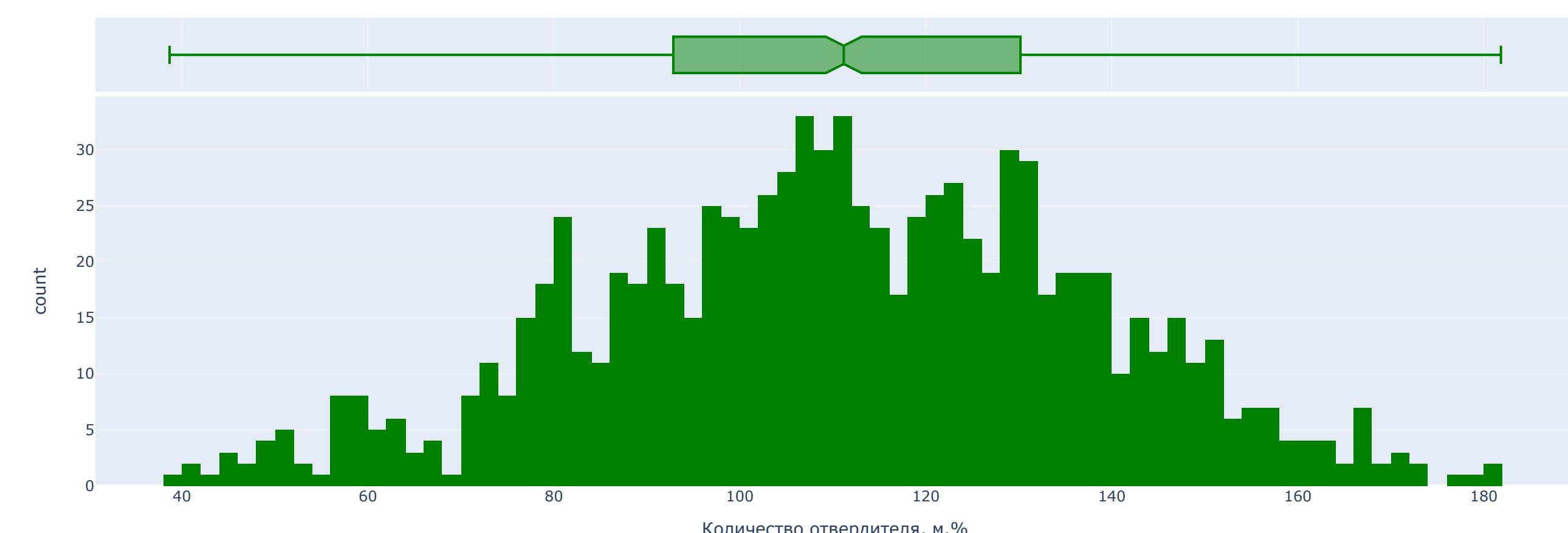
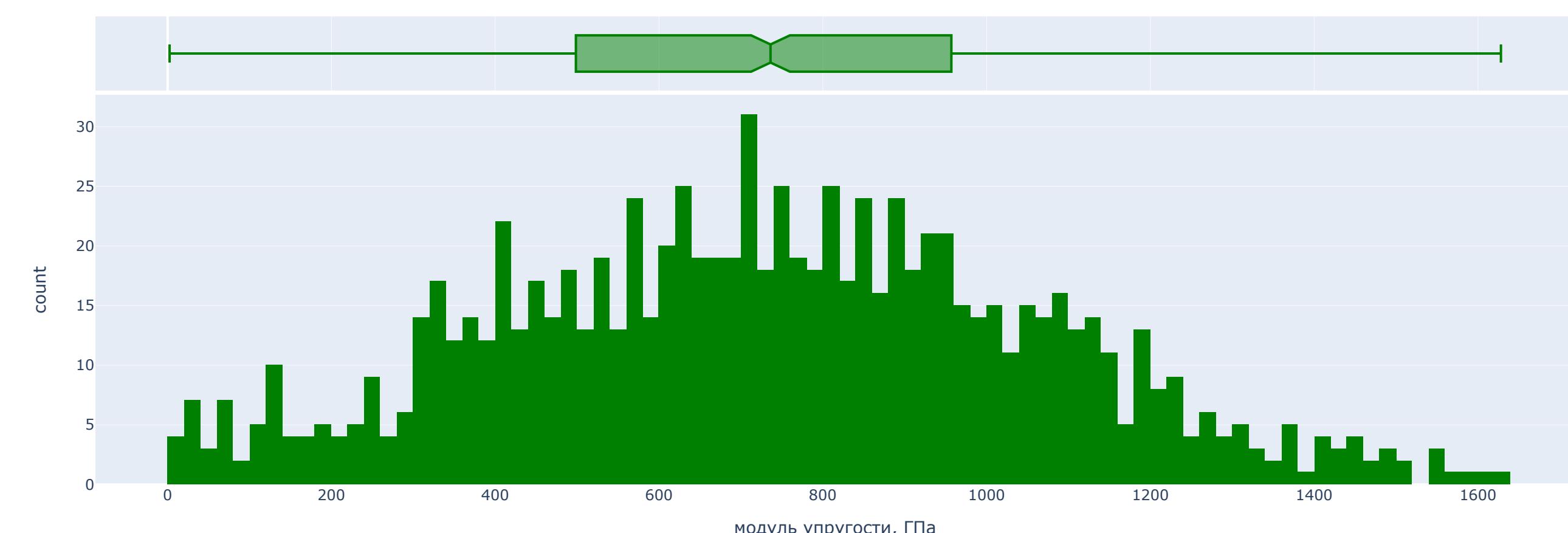
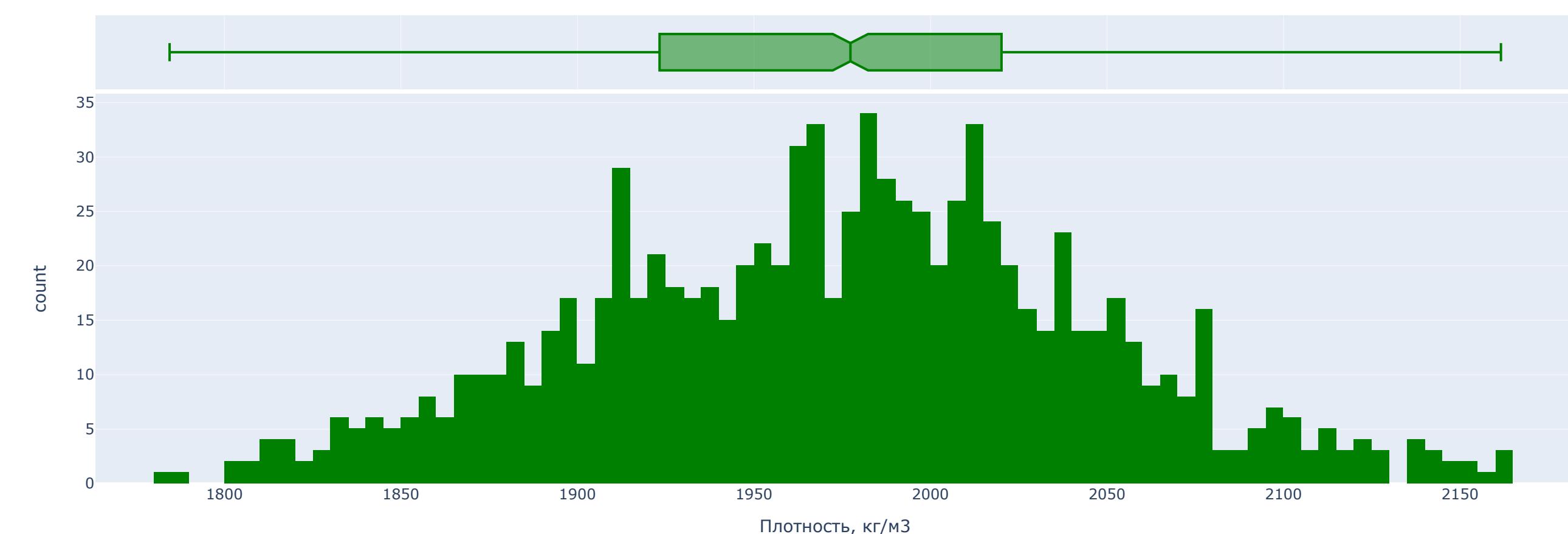
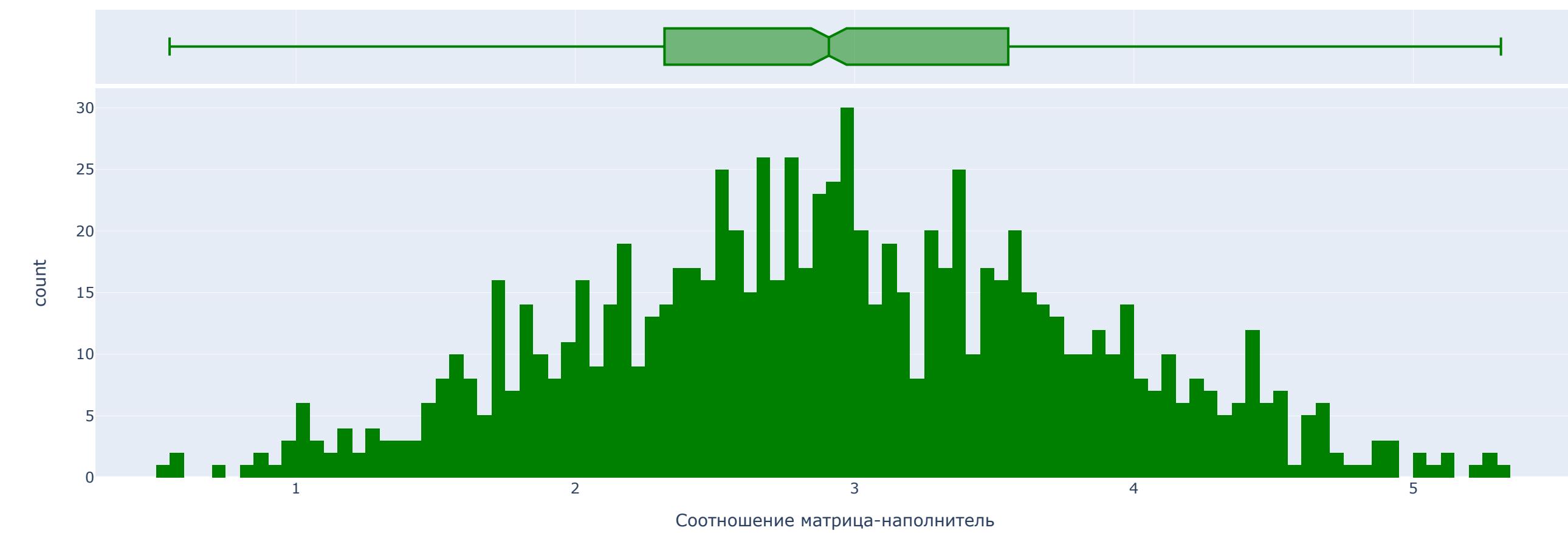
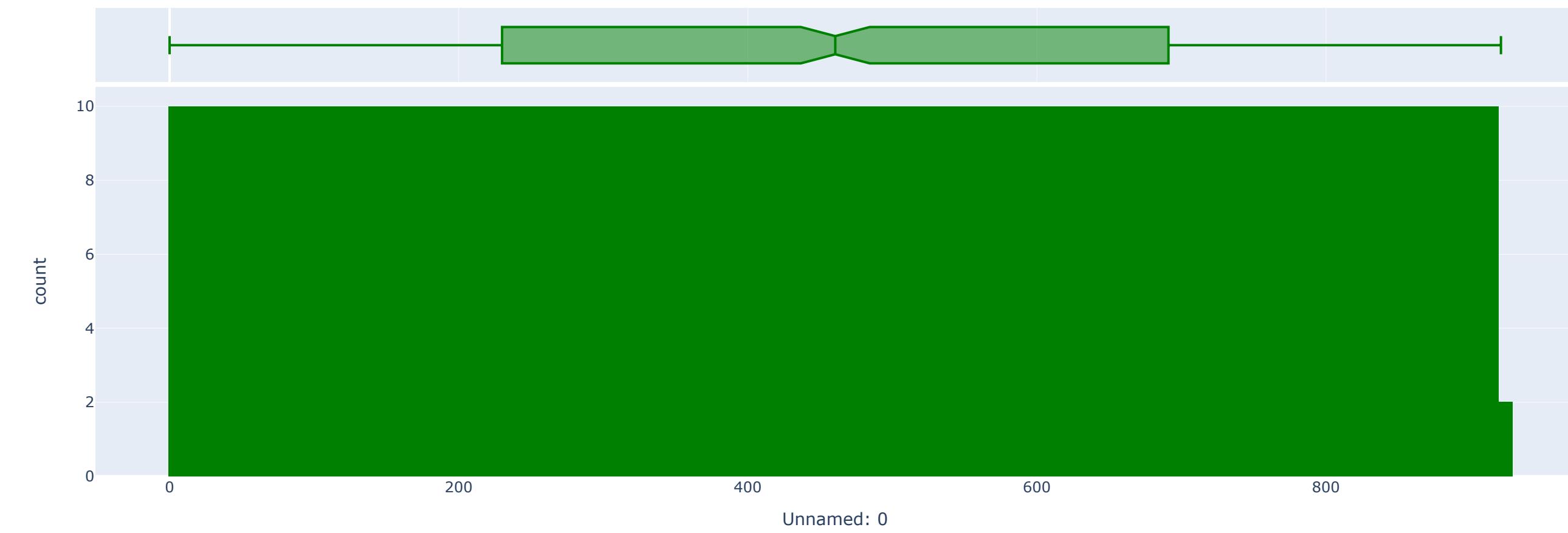
```
In [38]: # Гистограммы распределений (второй вариант)
a = # количество строк
b = # количество столбцов
c = # количество свободных ячеек
sns.set()
plt.figure(figsize=(35,35))
plt.suptitle('Гистограммы переменных', fontsize = 30)
for i in range(0,a):
    for j in range(0,b):
        if i == j:
            plt.subplot(a,b,j)
            sns.histplot(data = df[col], kde = True, color = "darkgreen")
            plt.title(col, size = 20)
            plt.show()
        else:
            plt.subplot(a,b,j)
            sns.histplot(data = df[col], kde = False, color = "darkgreen")
            plt.title(col, size = 20)
            plt.show()
c += 1
```

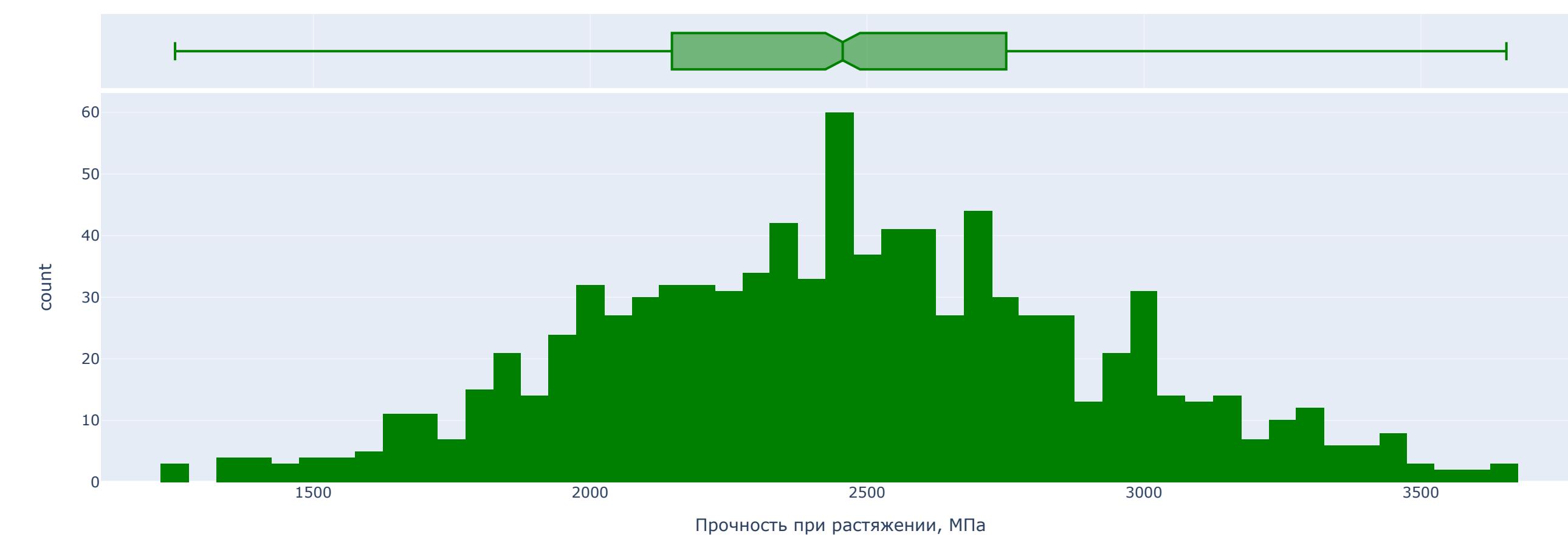
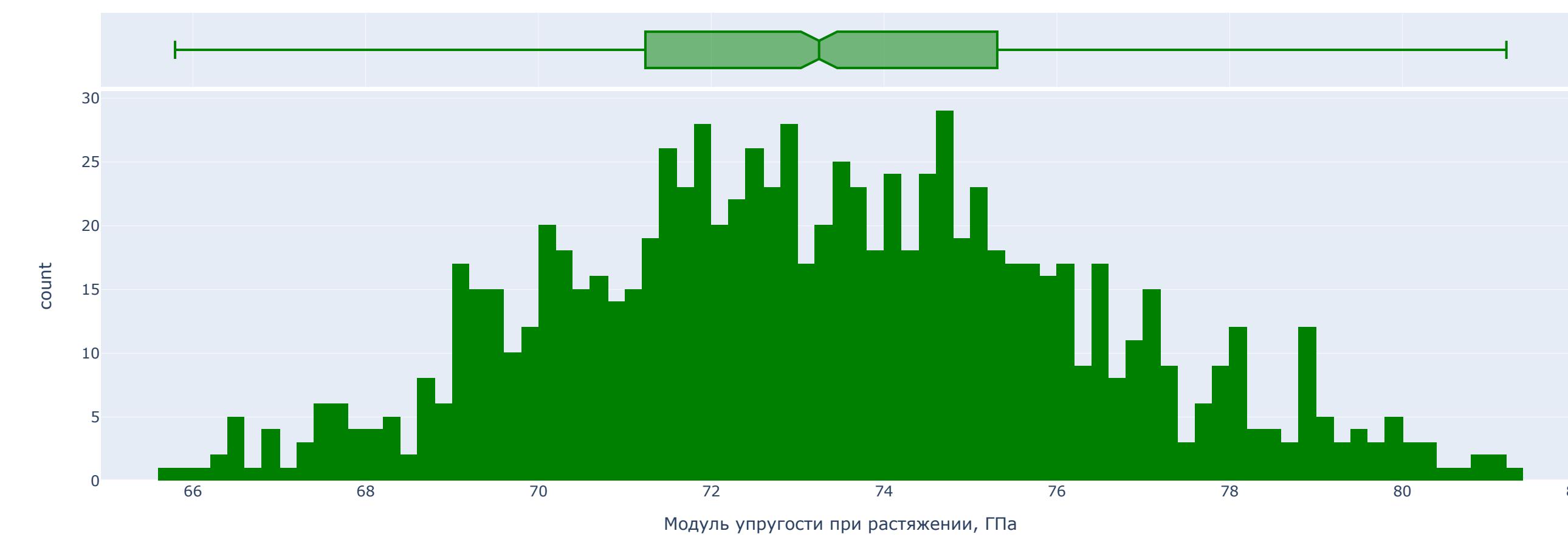
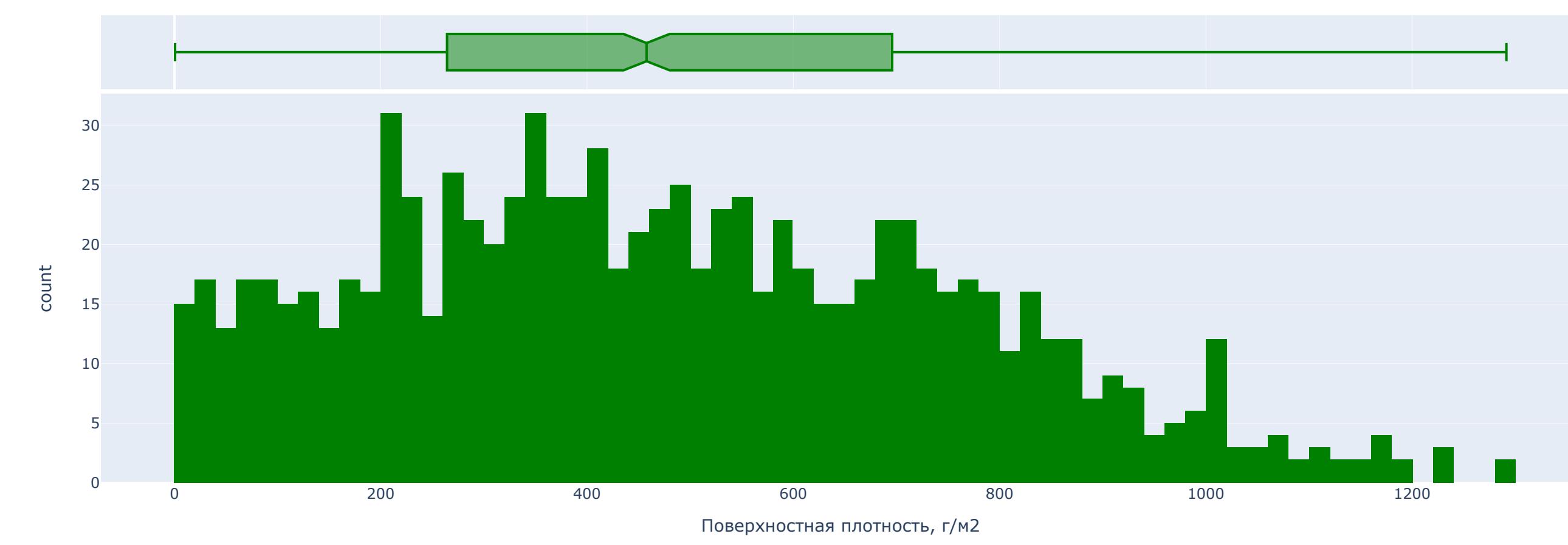
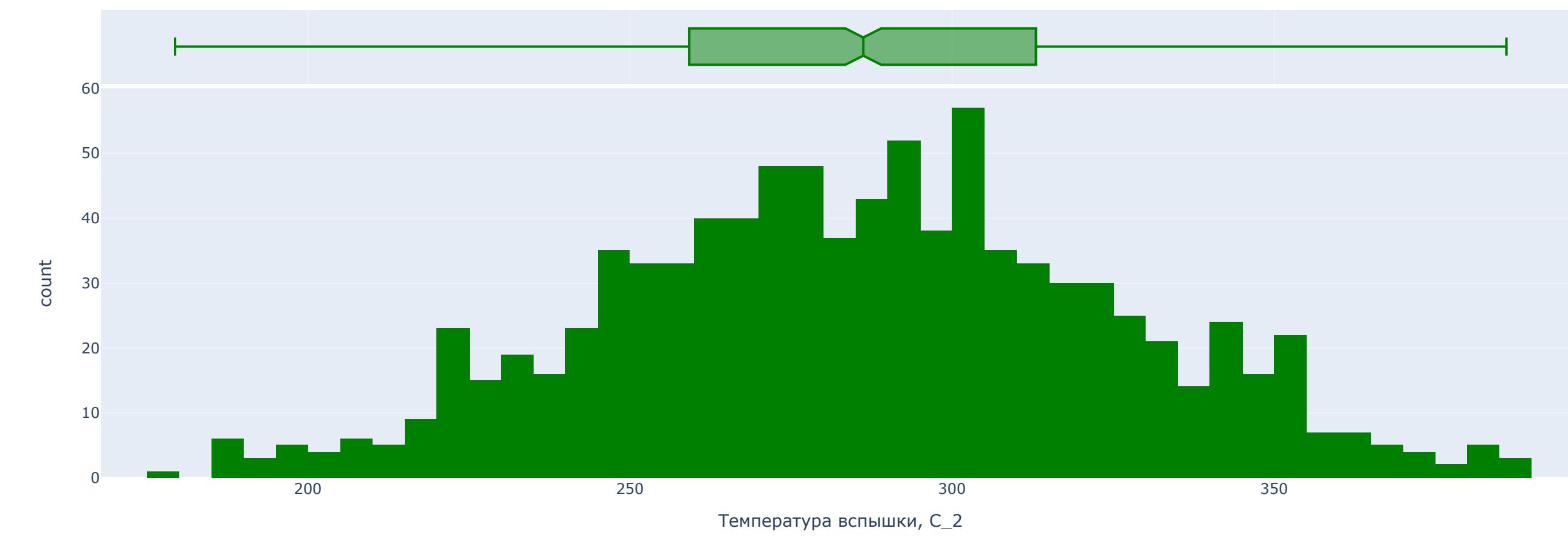
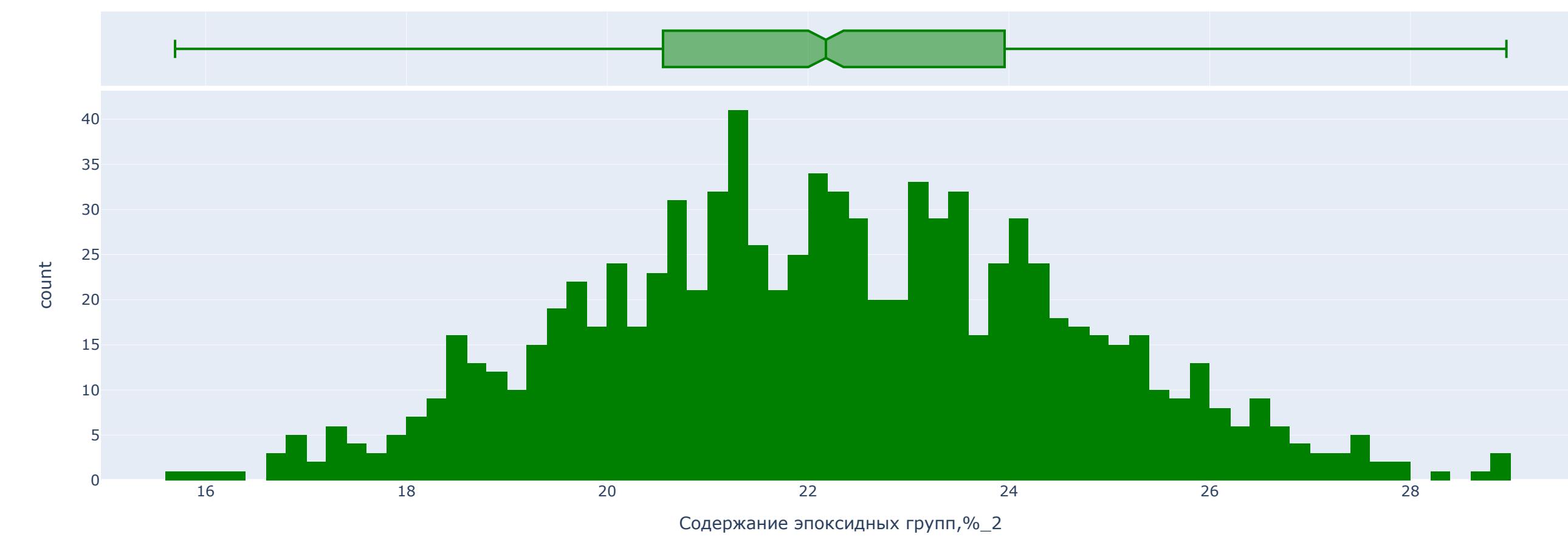
Данные стремятся к нормальному распределению практически всегда, кроме угла нашивки, имеющим только 2 значения, с которым мы уже поработали ранее.

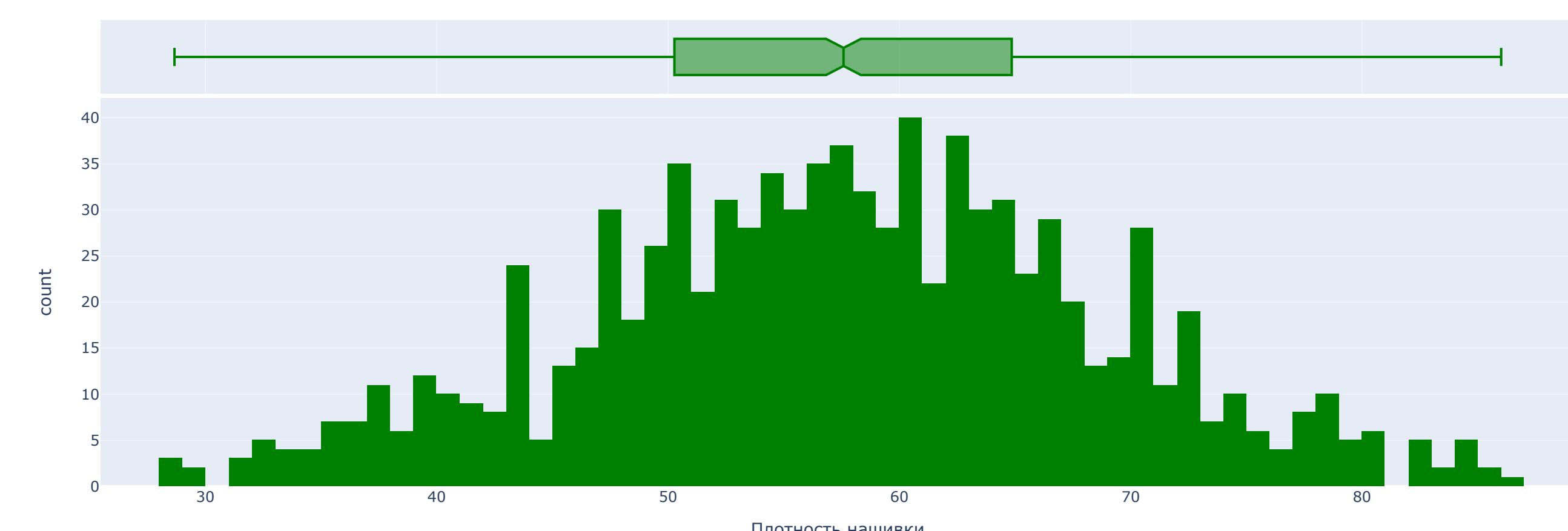
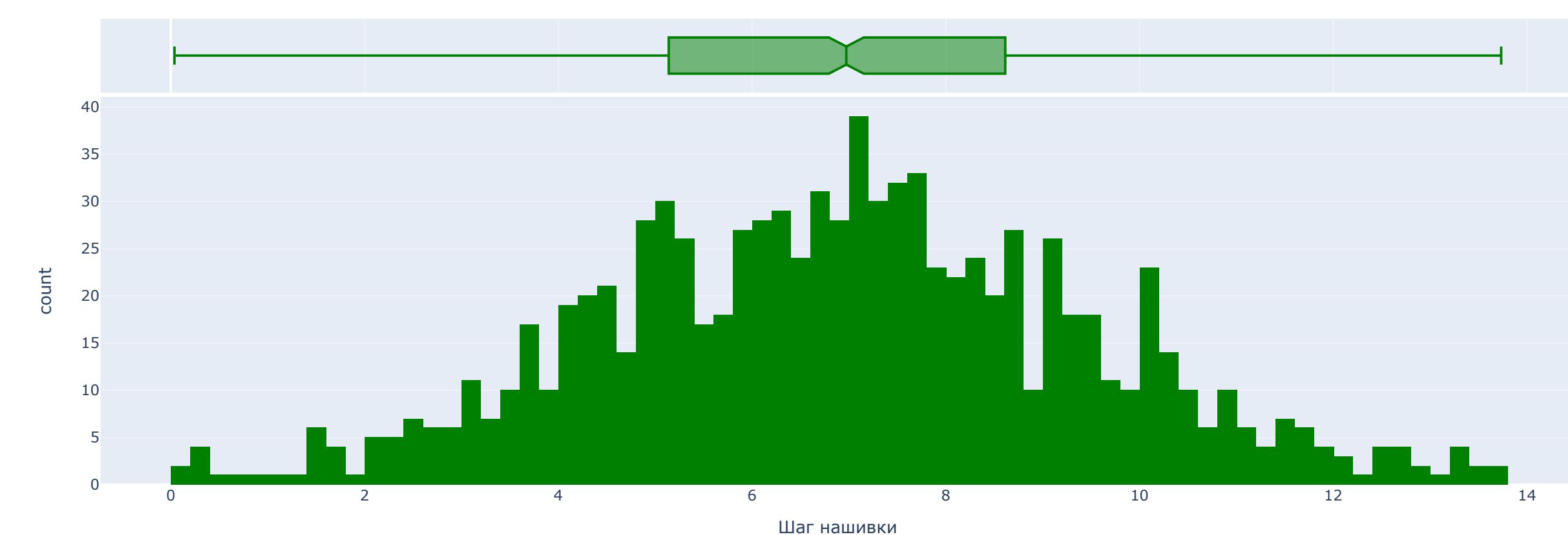
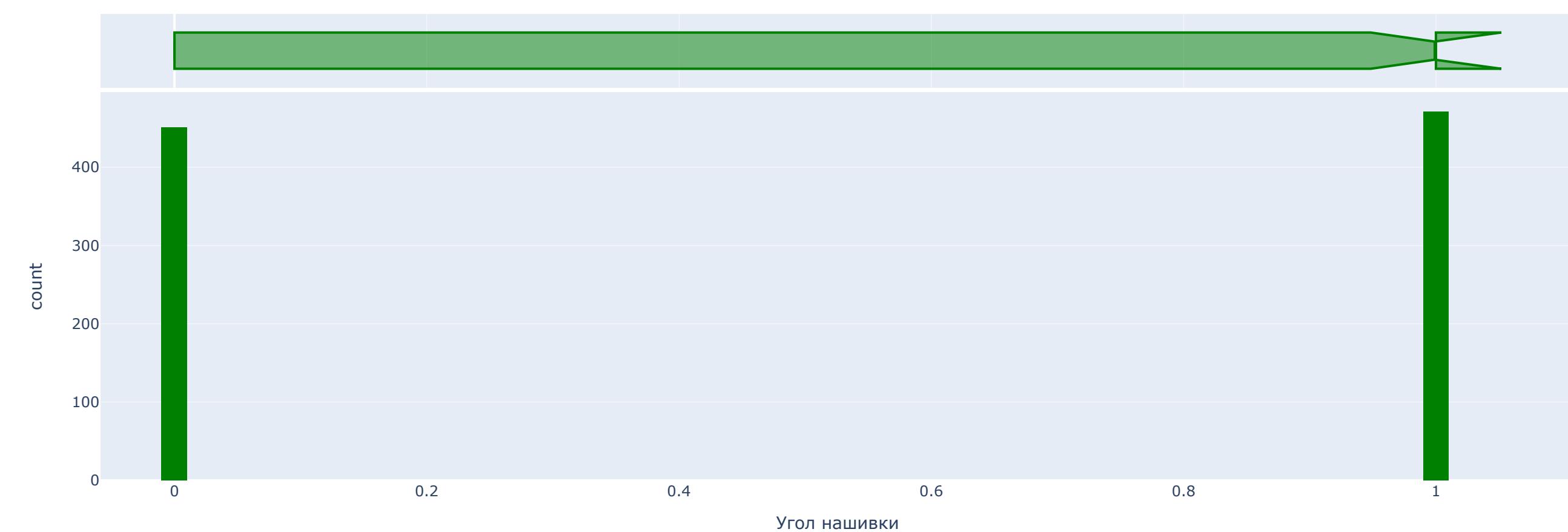
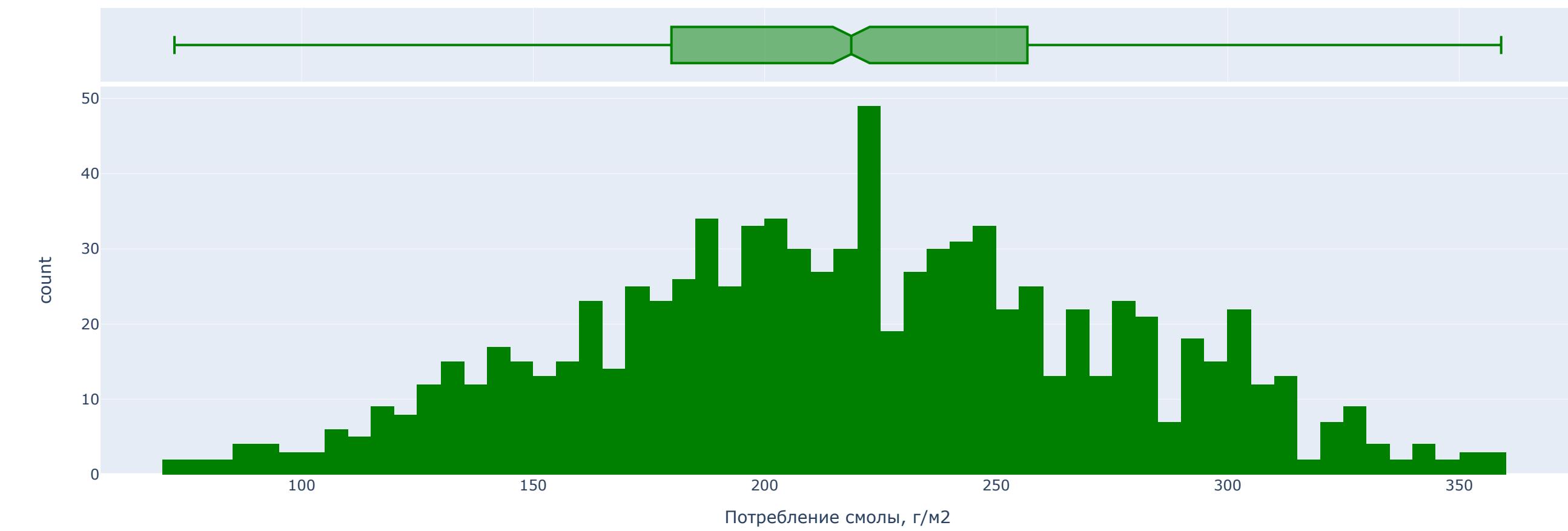
Гистограммы переменных



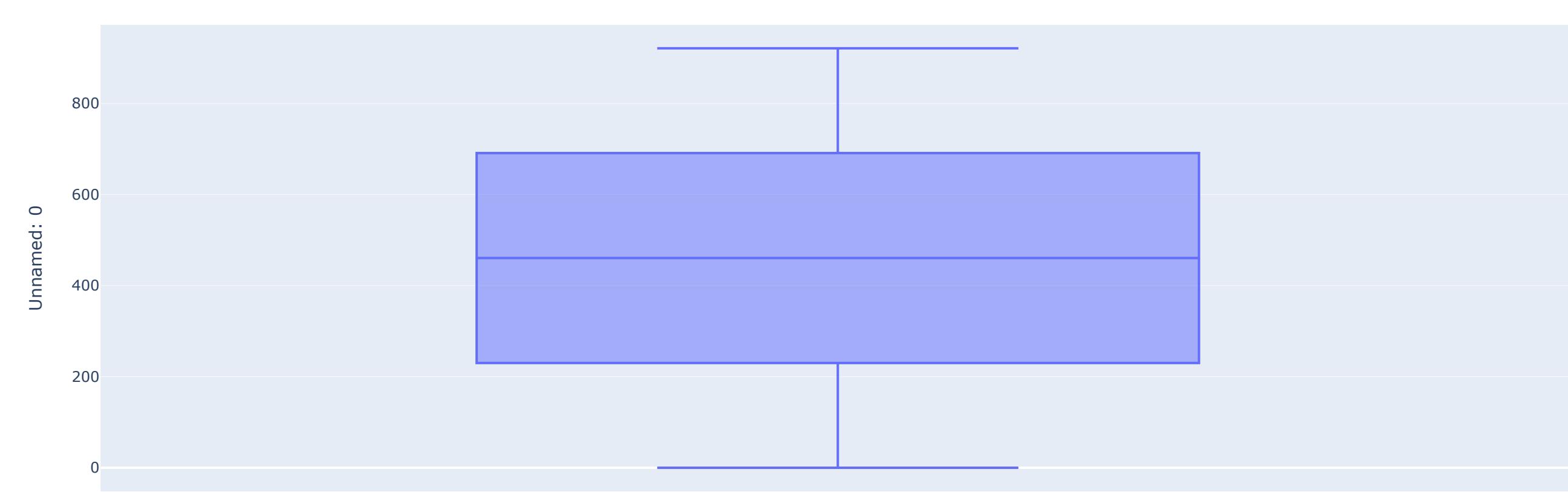
```
In [39]: # гистограмма распределения и боксплоты (третий вариант)
for column in df.columns:
    fig = px.histogram(df, x=column, color_discrete_sequence=['green'], nbins=100, marginal="box")
    fig.show()
```







```
In [ ]: for column in df.columns:
    fig = px.box(df, y=column)
    fig.show()
```



```
In [ ]: ## Данные с условием на одном рисунке
plt.figure(figsize=(15,10))
ax = sns.boxplot(data=df)
ax.set_xticklabels(ax.get_xticklabels(), rotation=30);
print("")


##нашикка кода выше не работает в колабе, поэтому выбросы проверим еще другими способами
for column in df.columns:
    print(column)

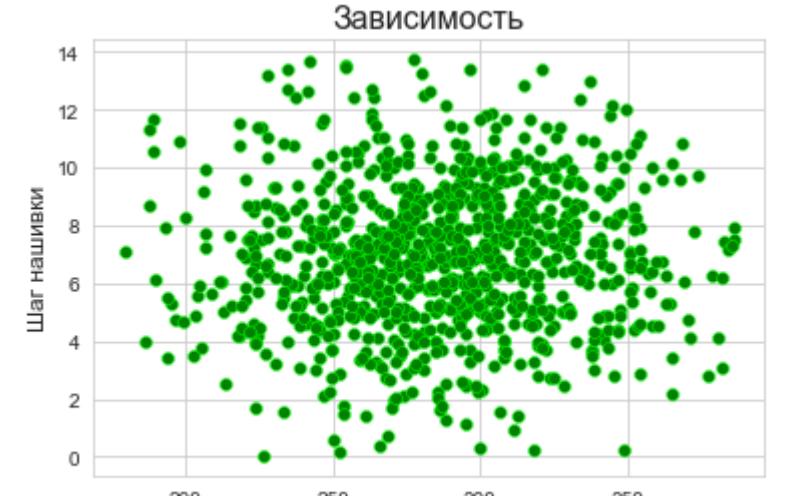
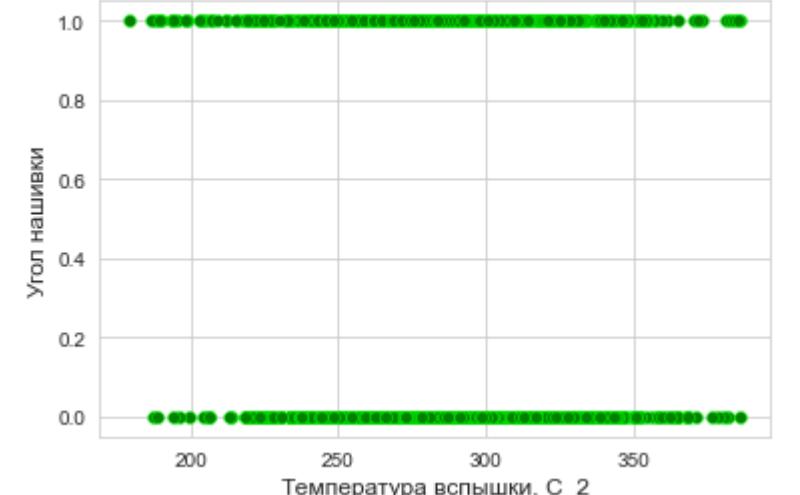
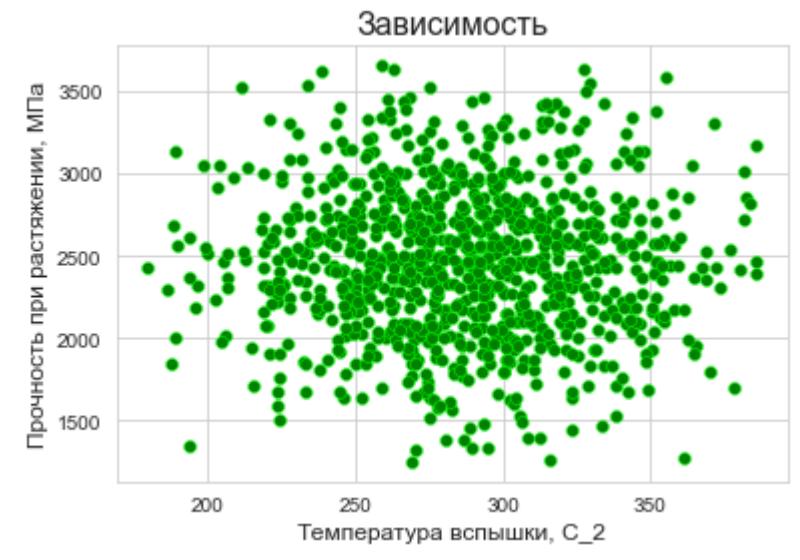
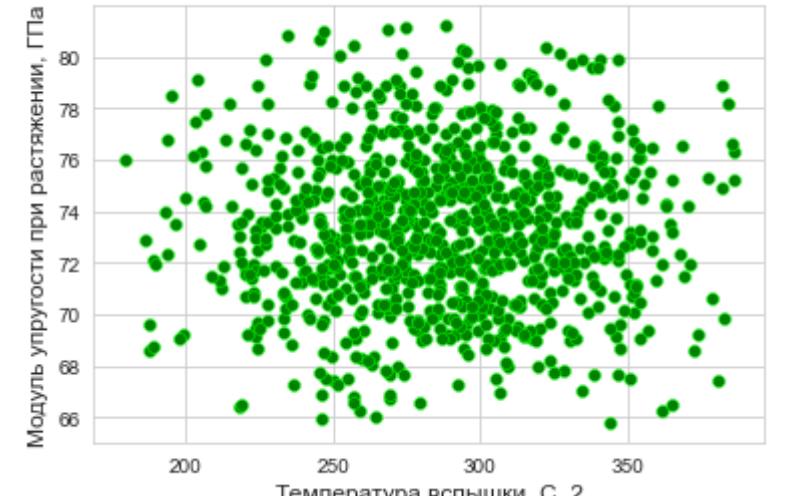
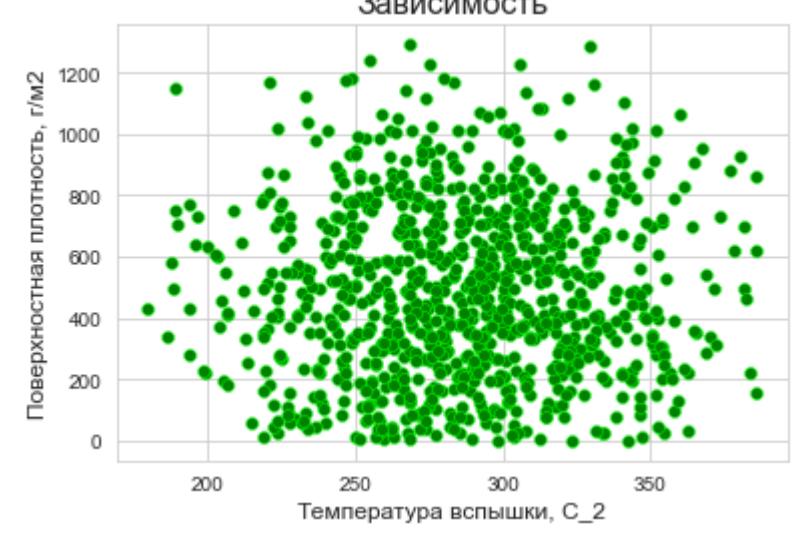
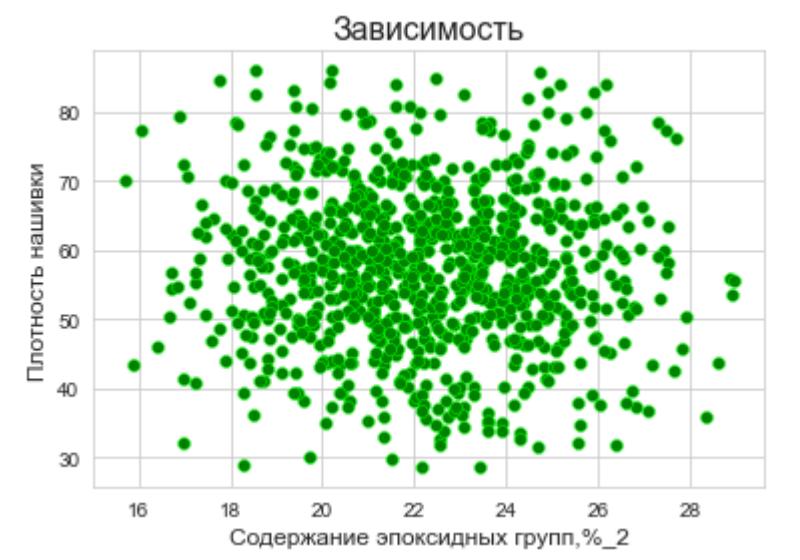
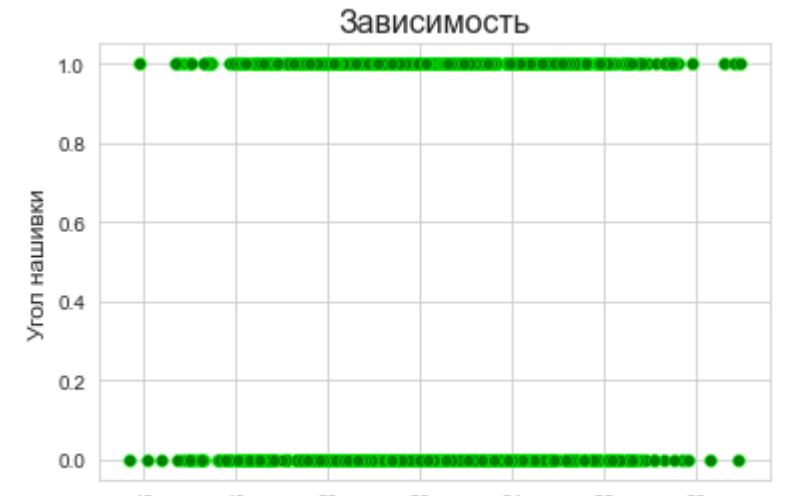
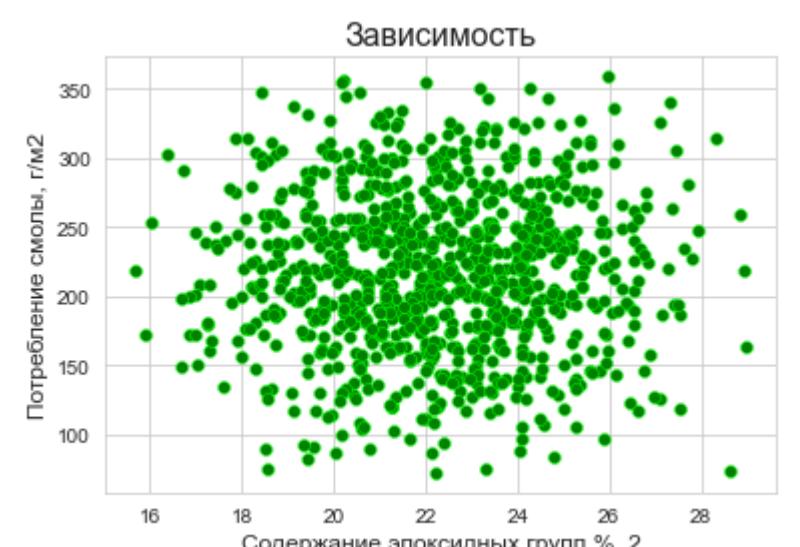
##нашикка кода выше не работает в колабе, поэтому выбросы проверим еще другими способами
gls = df[column_name]
sns.set_style("whitegrid")
sns.kdeplot(data = gls, shade = True, palette = "colorblind", color = "g")
plt.show()

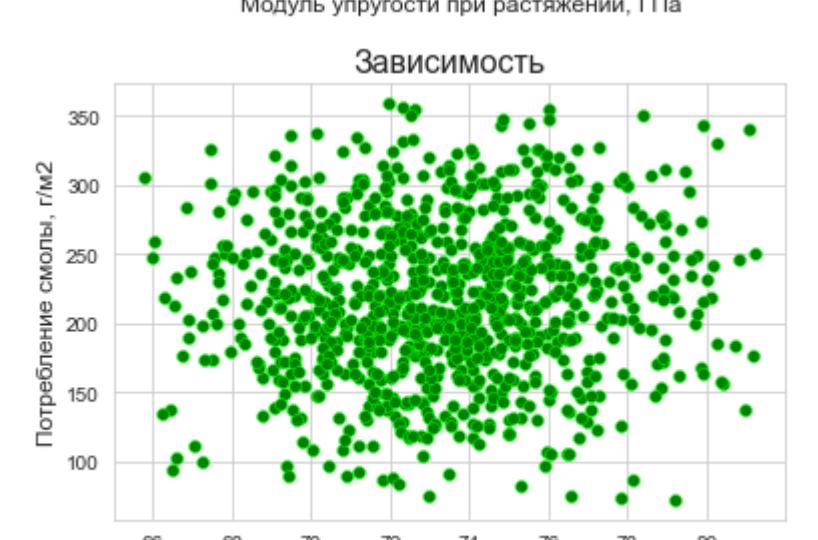
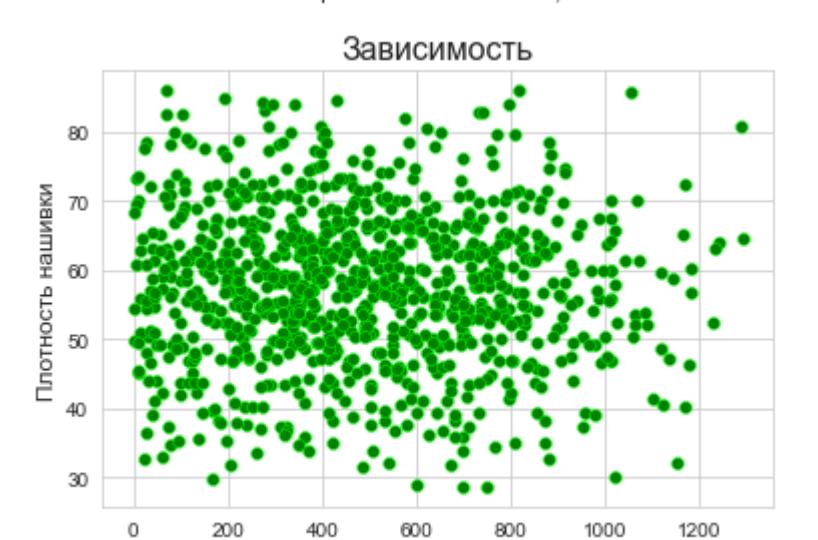
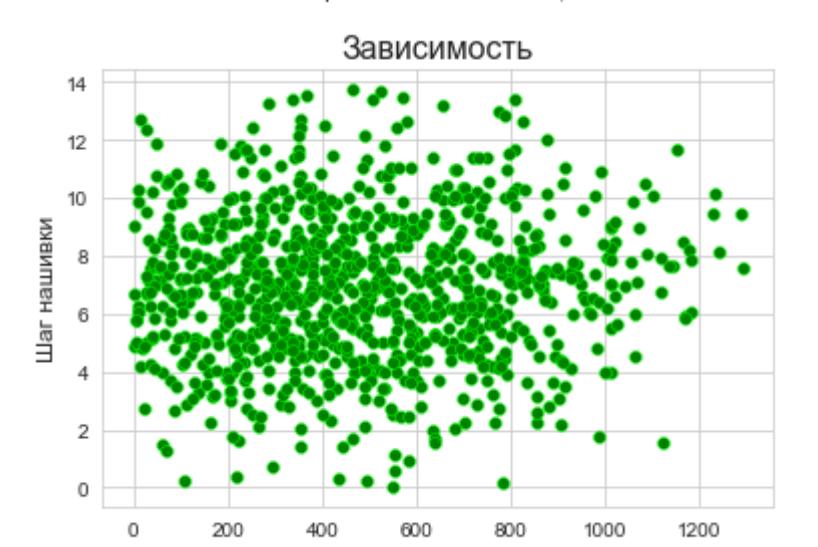
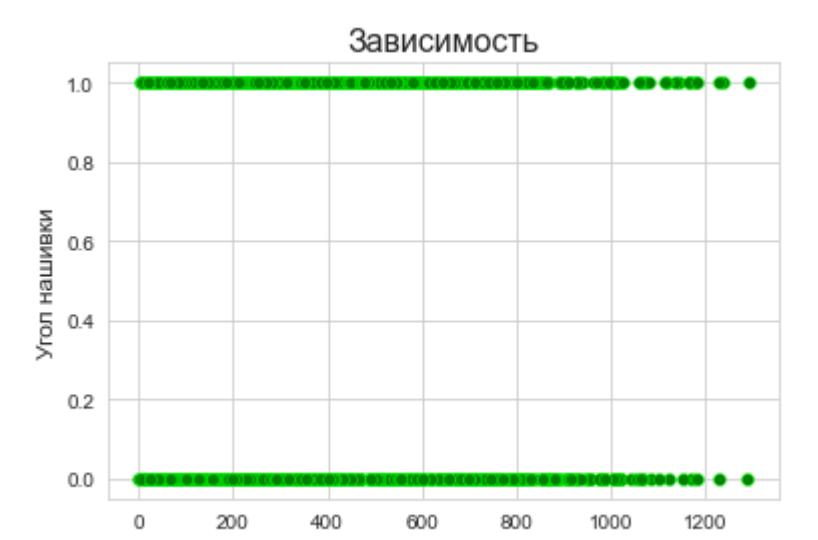
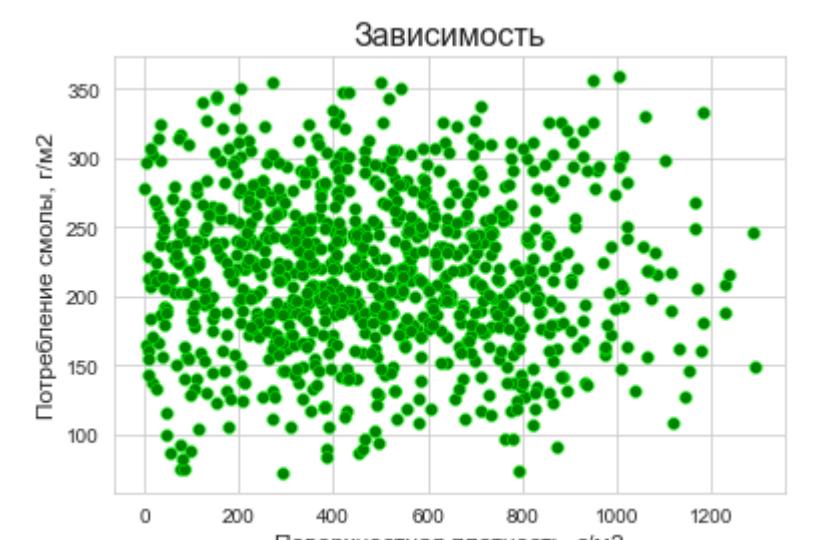
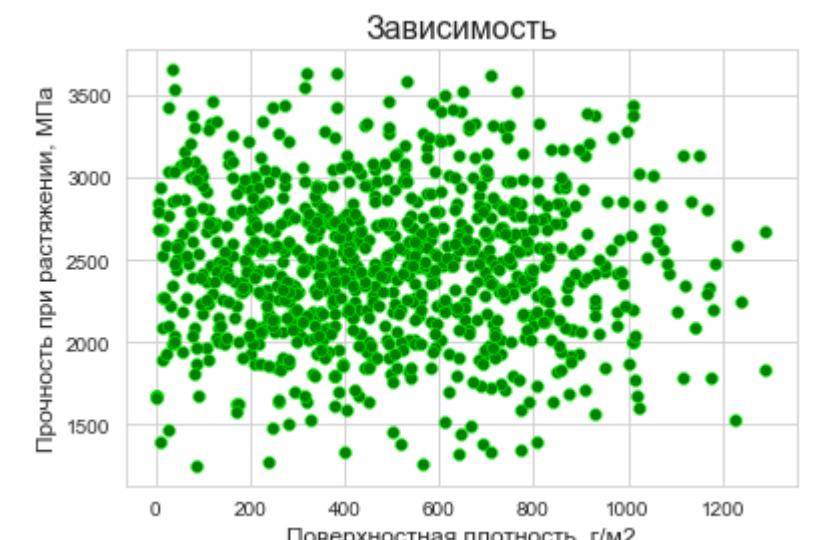
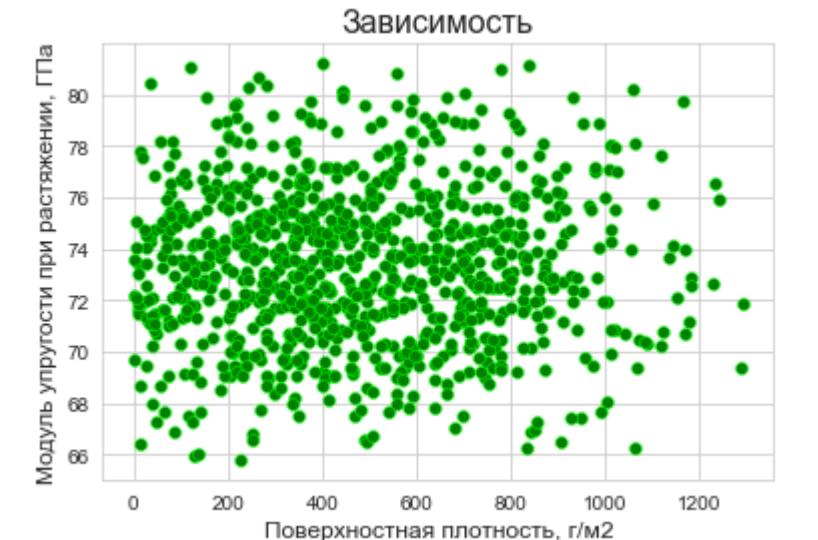
##нашикка кода выше не работает в колабе, поэтому выбросы проверим еще другими способами
sns.boxplot(x = gls, color = "g");
plt.show()

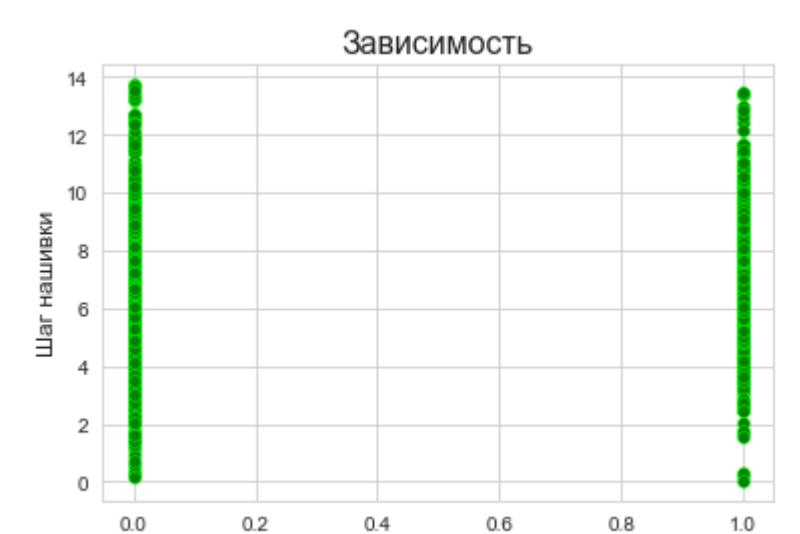
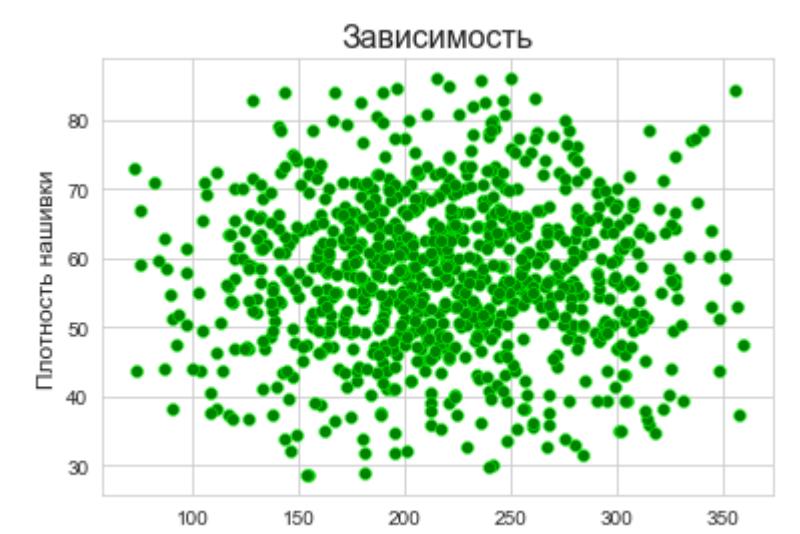
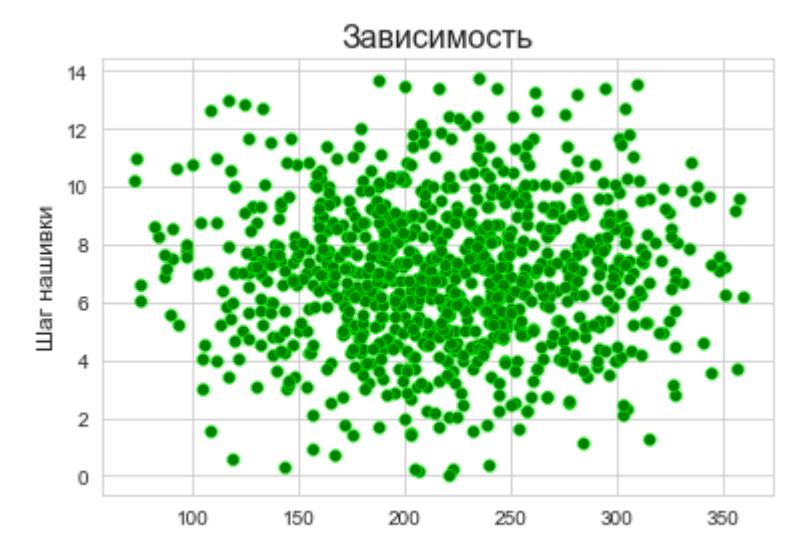
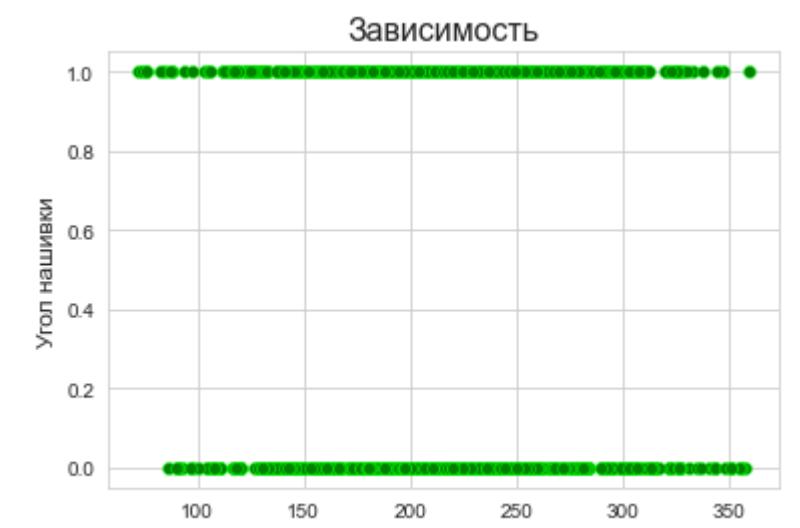
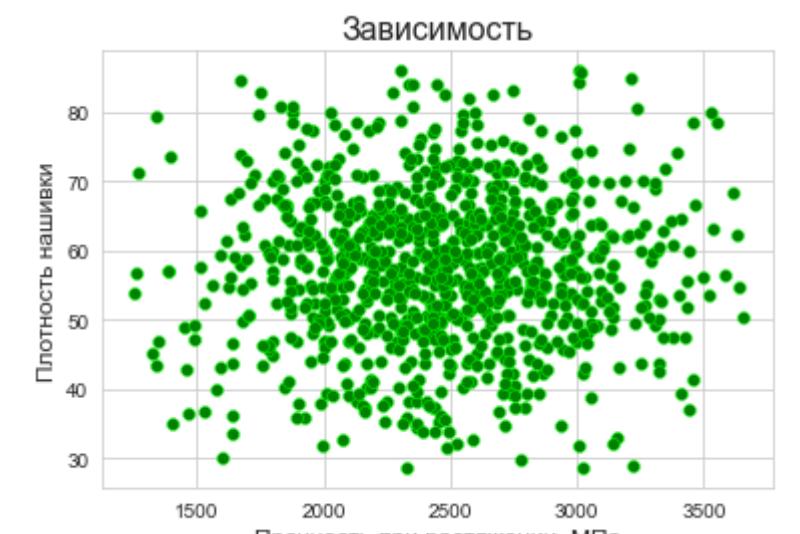
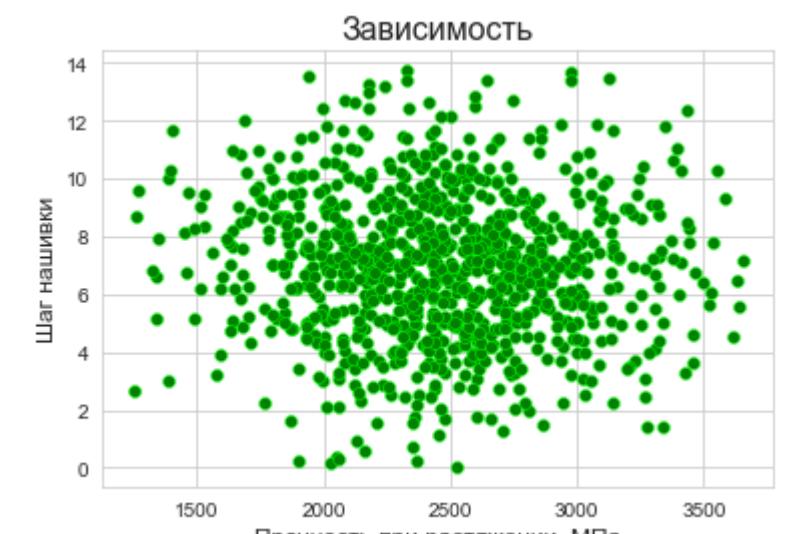
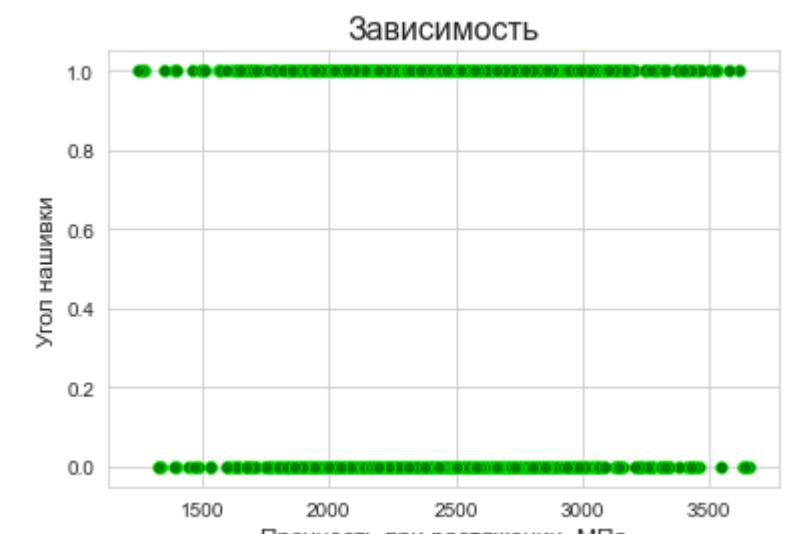
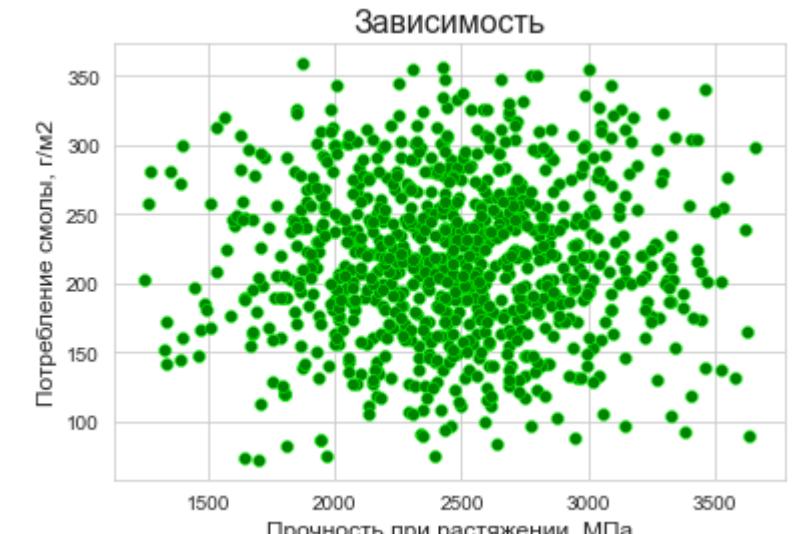
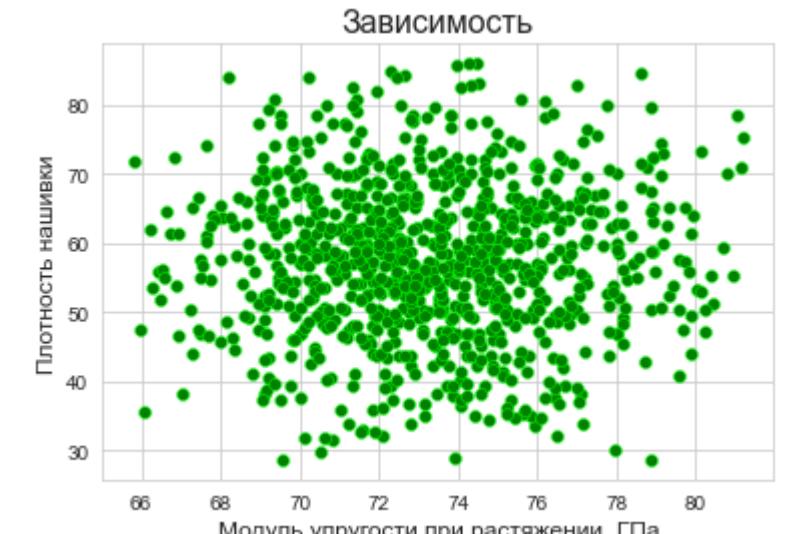
##нашикка кода выше не работает в колабе, поэтому выбросы проверим еще другими способами
print("Минимальное значение: ", end = " ")
print("Максимальное значение: ", end = " ")
```

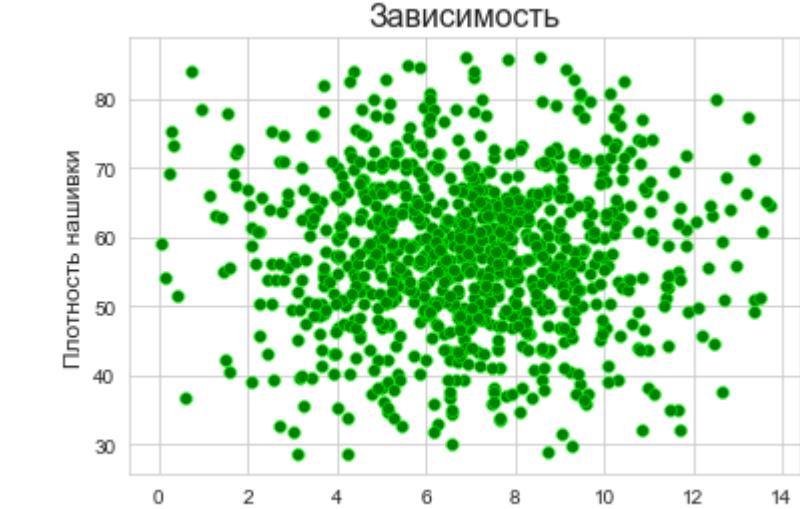
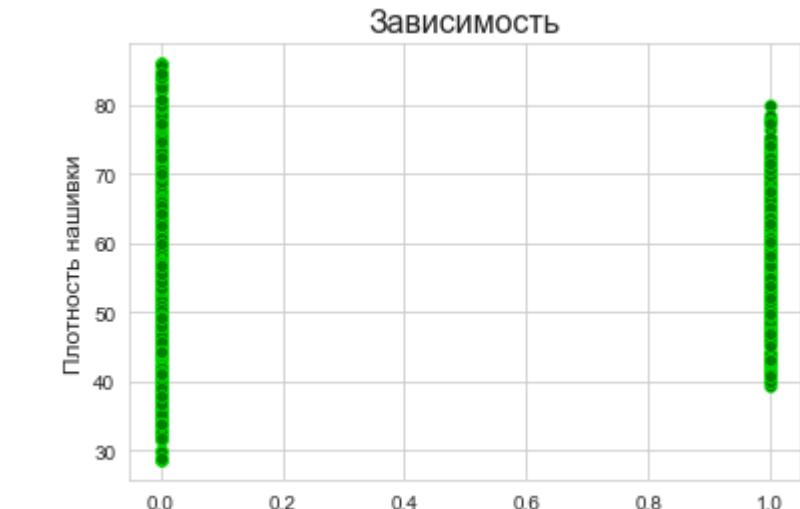
```
print(np.min(gis))
print("Минимальное значение: ", end = " ")
print(np.max(gis))
print("Максимальное значение: ", end = " ")
print("Среднее значение: ", end = " ")
print(np.mean(gis))
print("Медианное значение: ", end = " ")
print(np.median(gis))
print("\n\n")
```







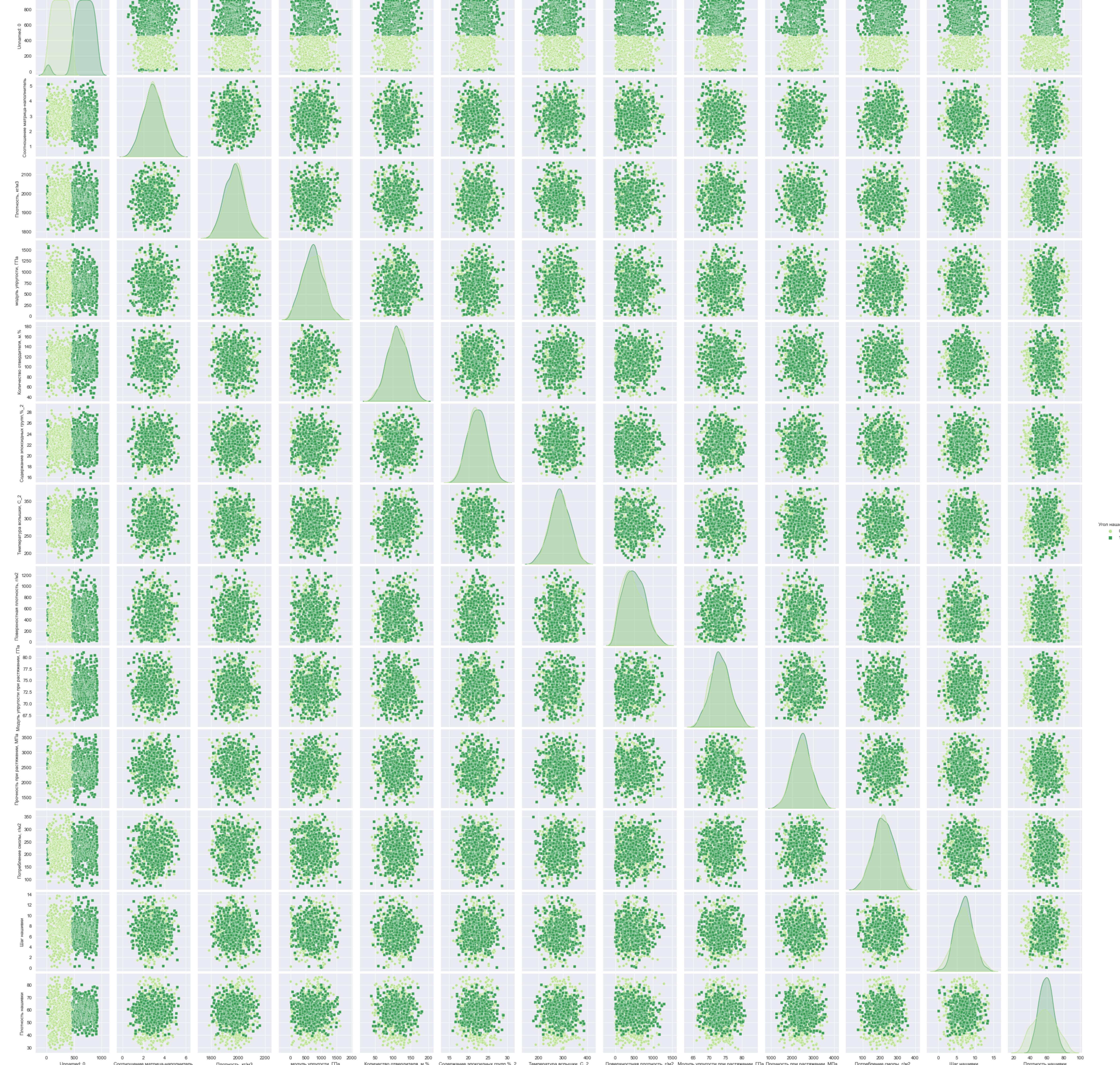




```
In [44]: sns.set_style('darkgrid')
sns.pairplot(df, hue = 'Угол нашивки', markers = ["o", "s"], diag_kind = 'auto', palette = 'YlGn')

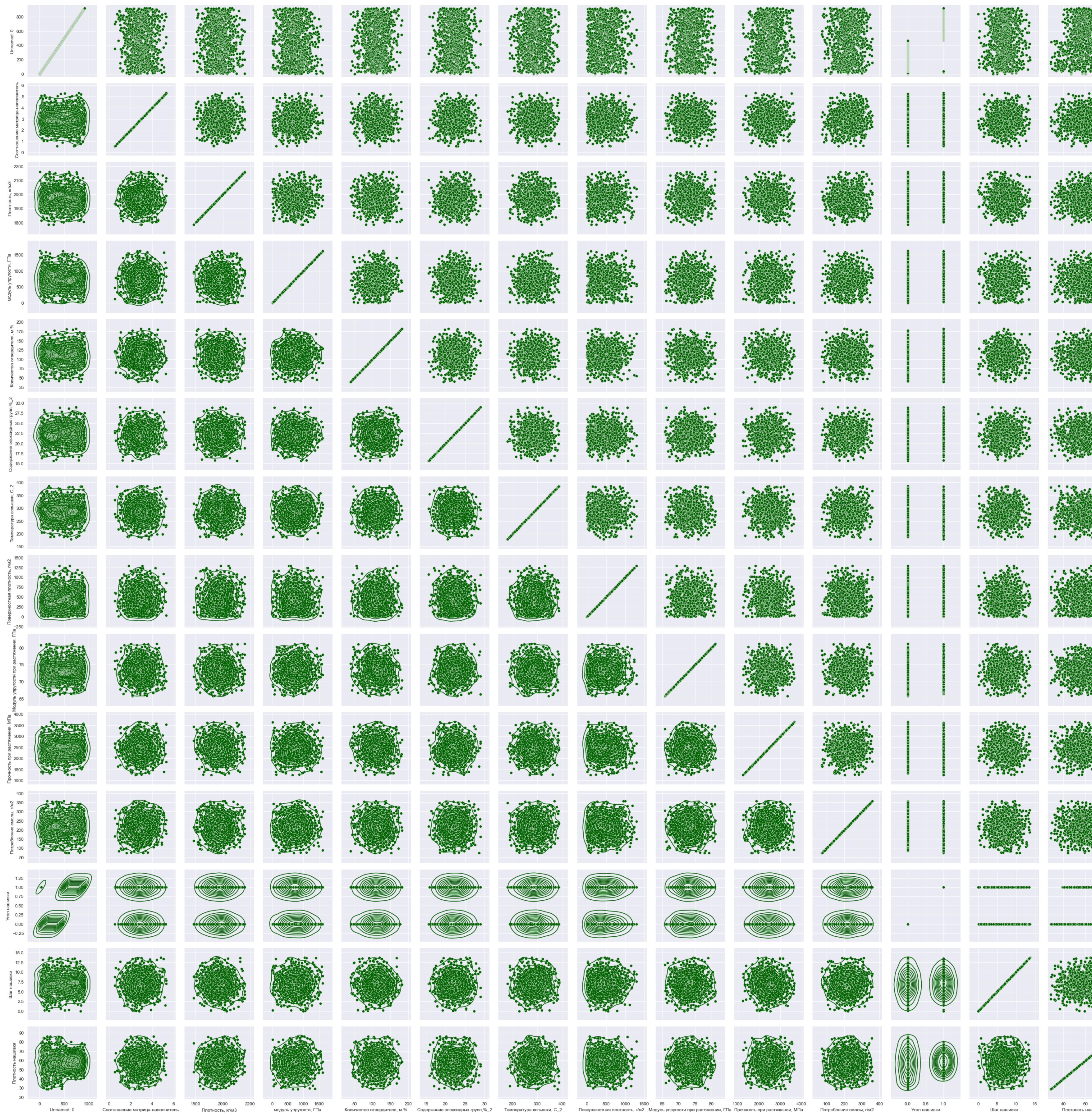
Out[44]: <seaborn.axisgrid.PairGrid at 0x1fbdb4246040>
```

```
Out[44]:
```



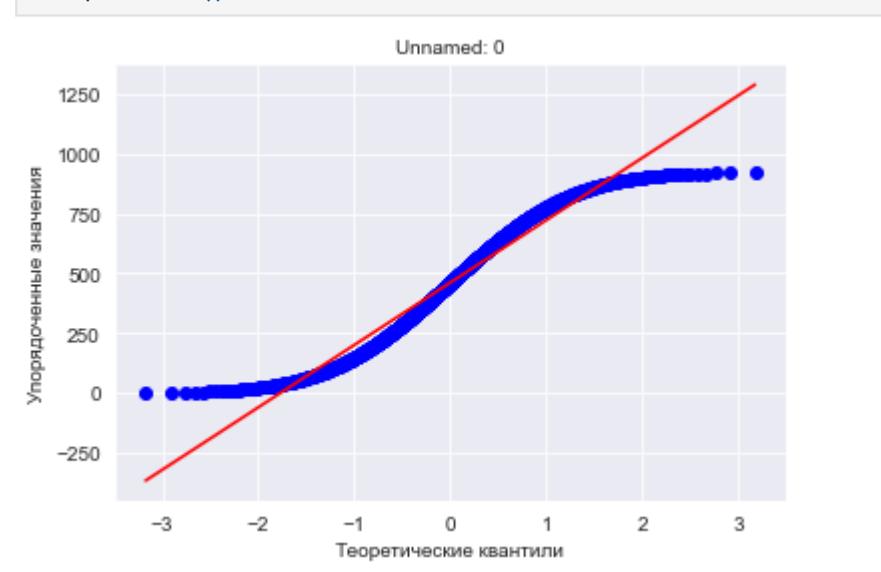
```
In [45]: # Попарные графики рассеяния точек - скаттерплоты (второй вариант)
g = sns.PairGrid(df[df.columns])
g.map(sns.scatterplot, color = 'darkgreen')
g.map_upper(sns.scatterplot, color = 'darkgreen')
g.map_lower(sns.kdeplot, color = 'darkgreen')
plt.show
```

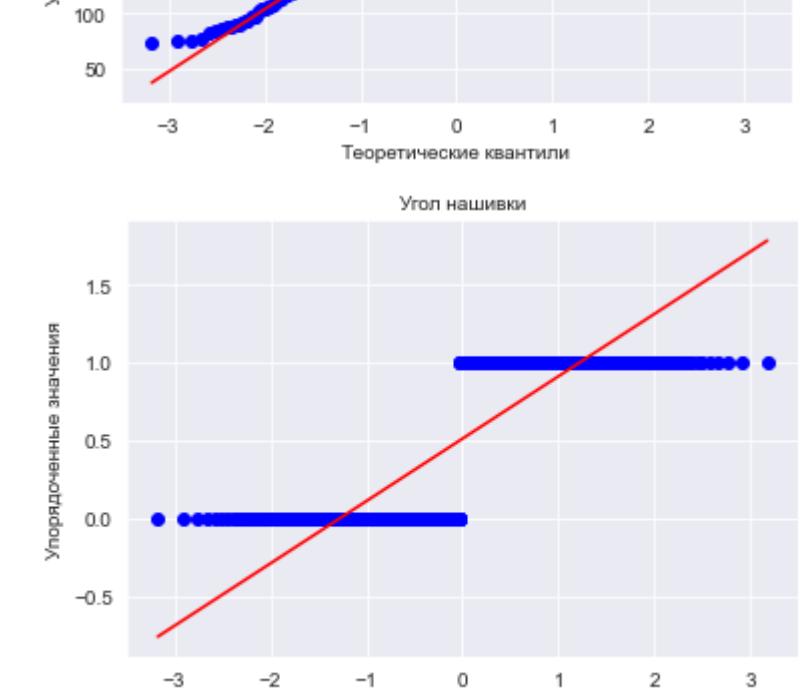
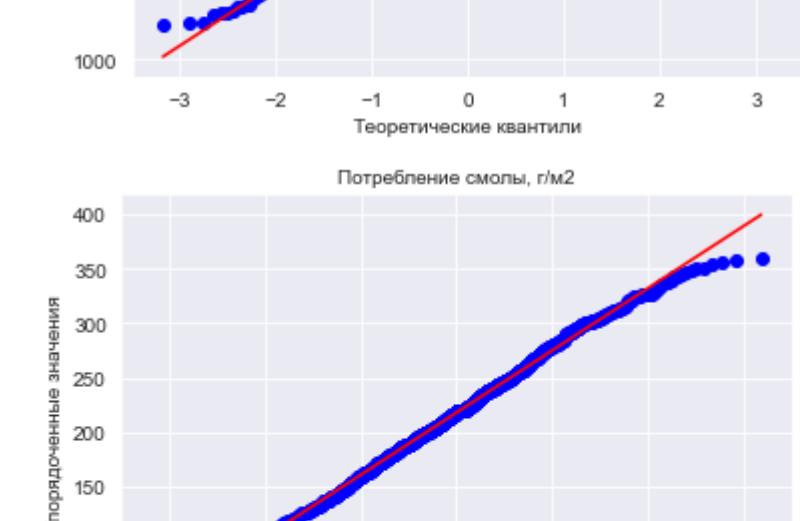
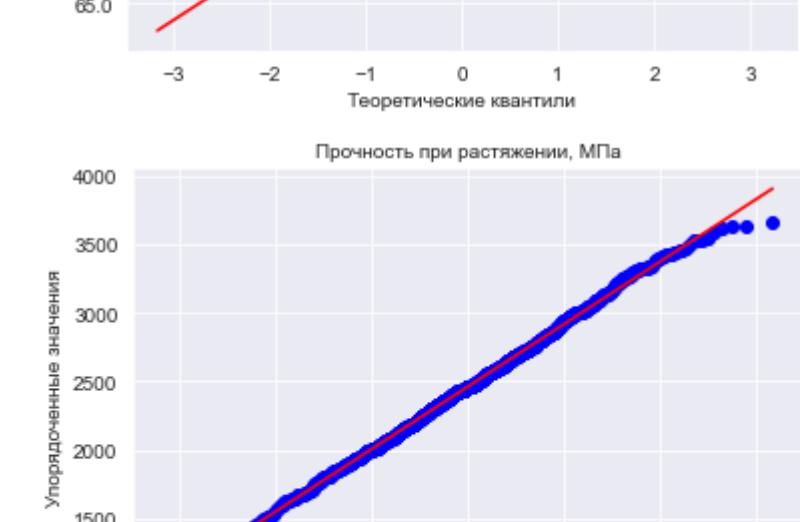
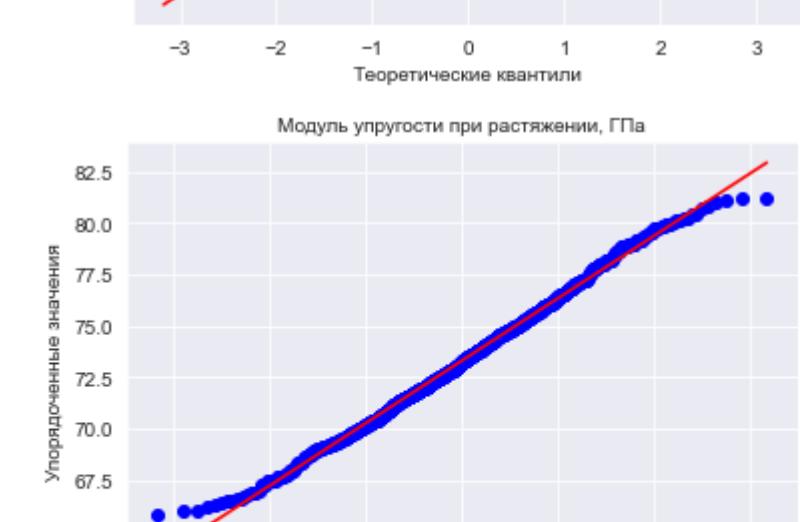
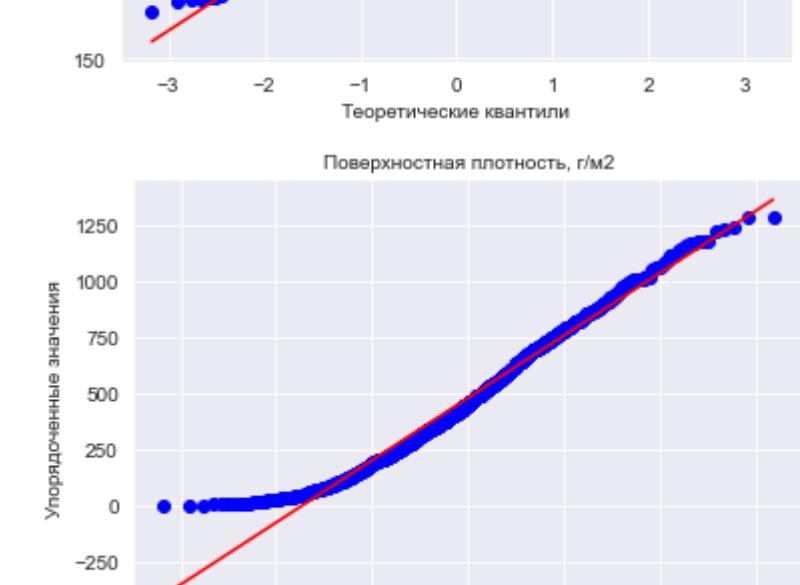
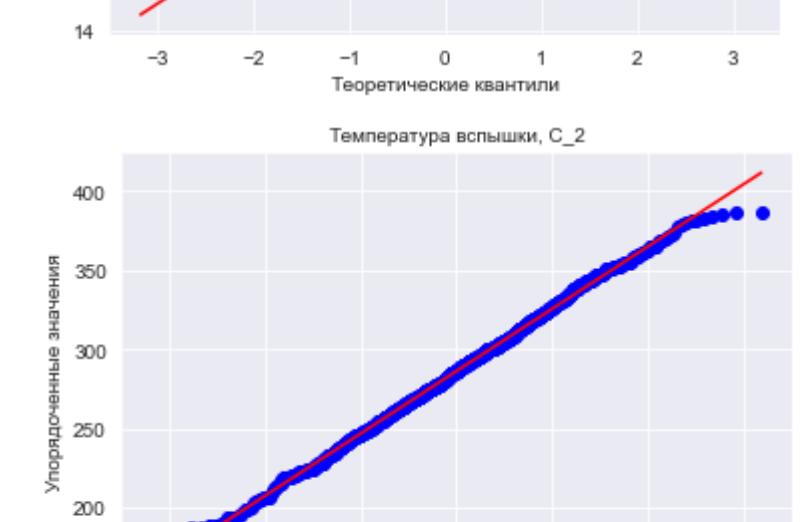
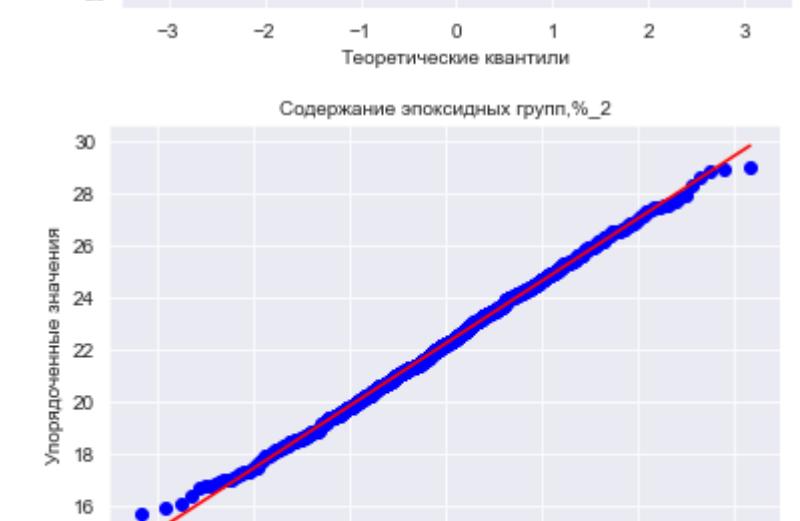
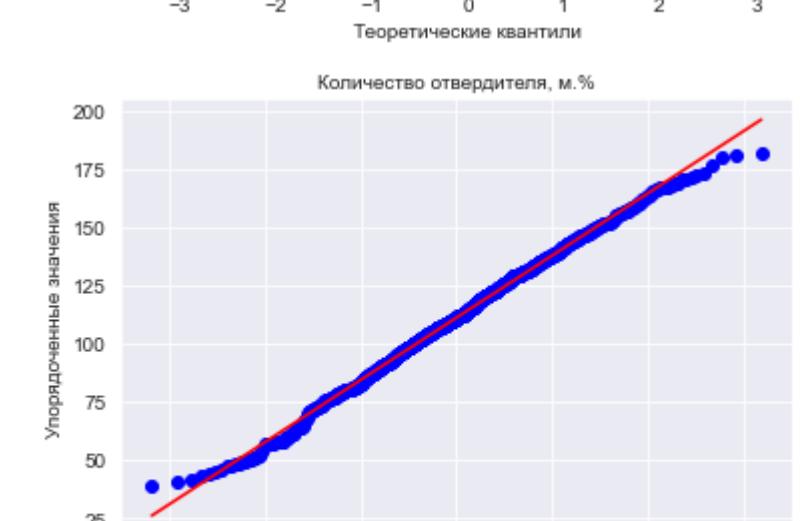
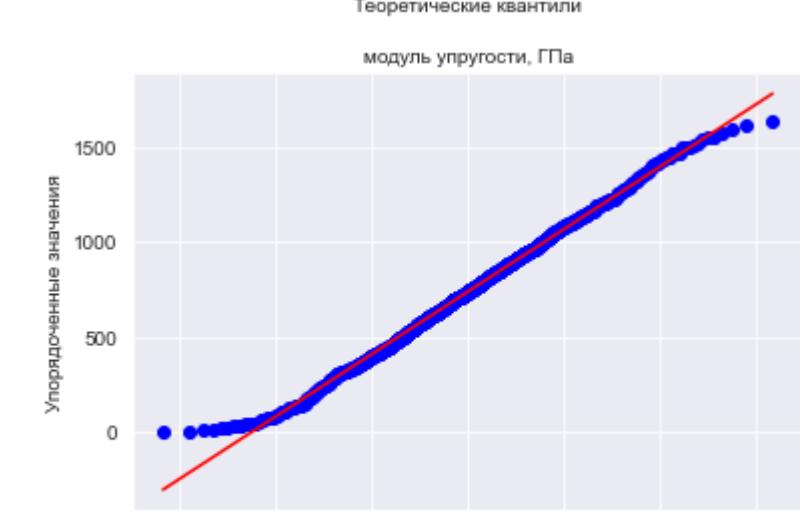
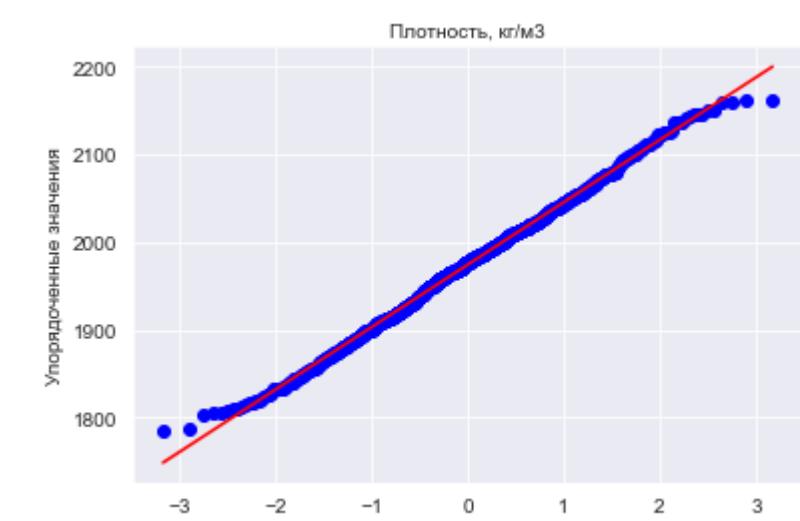
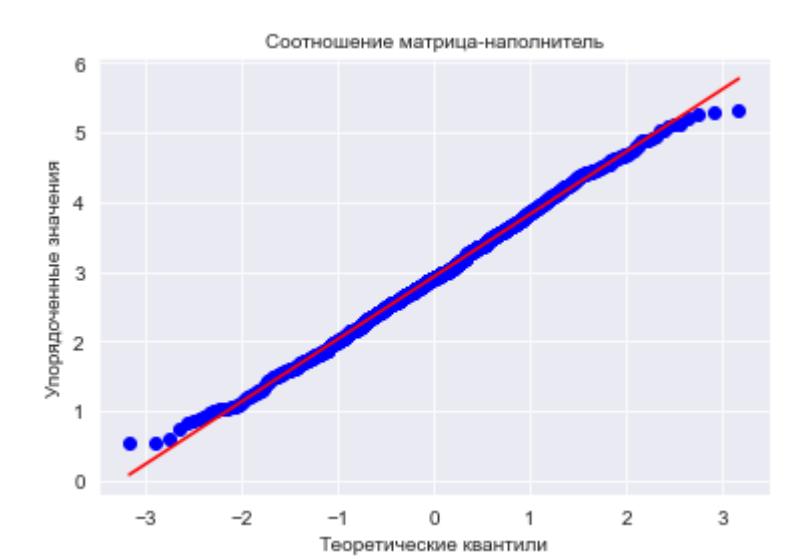
```
Out[45]: <function matplotlib.pyplot.show(close=None, block=None)>
```

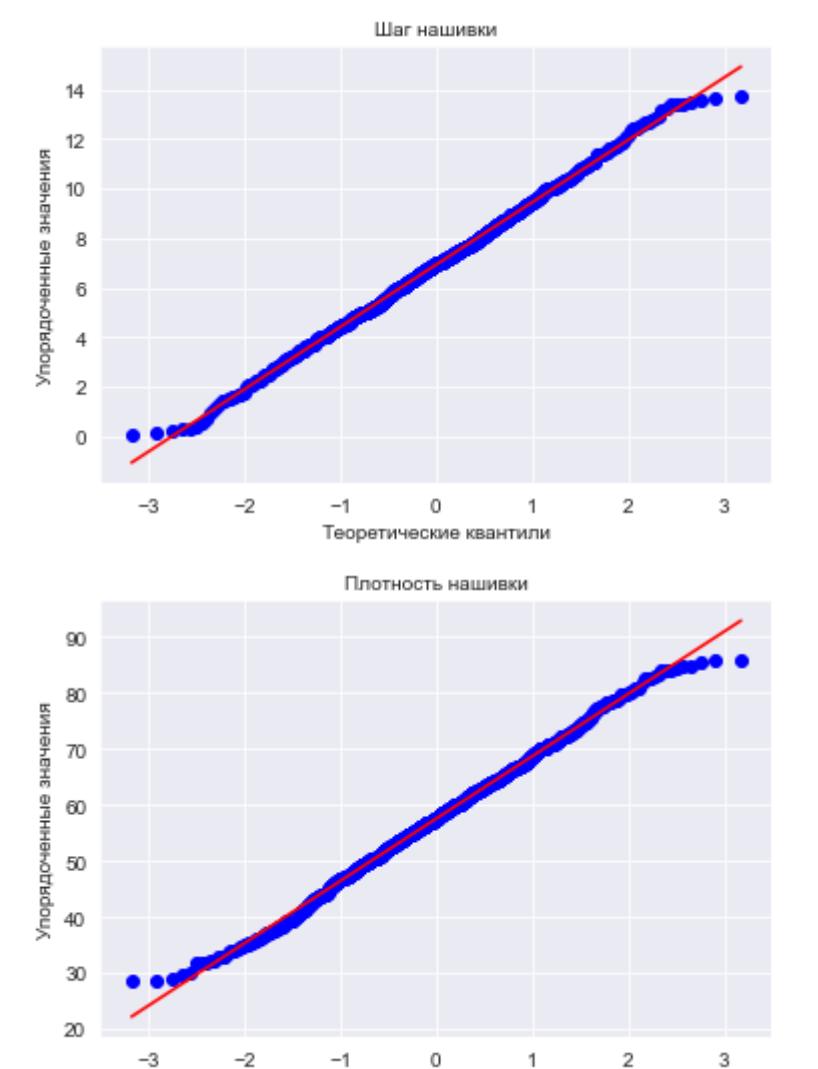


```
In [46]:
```

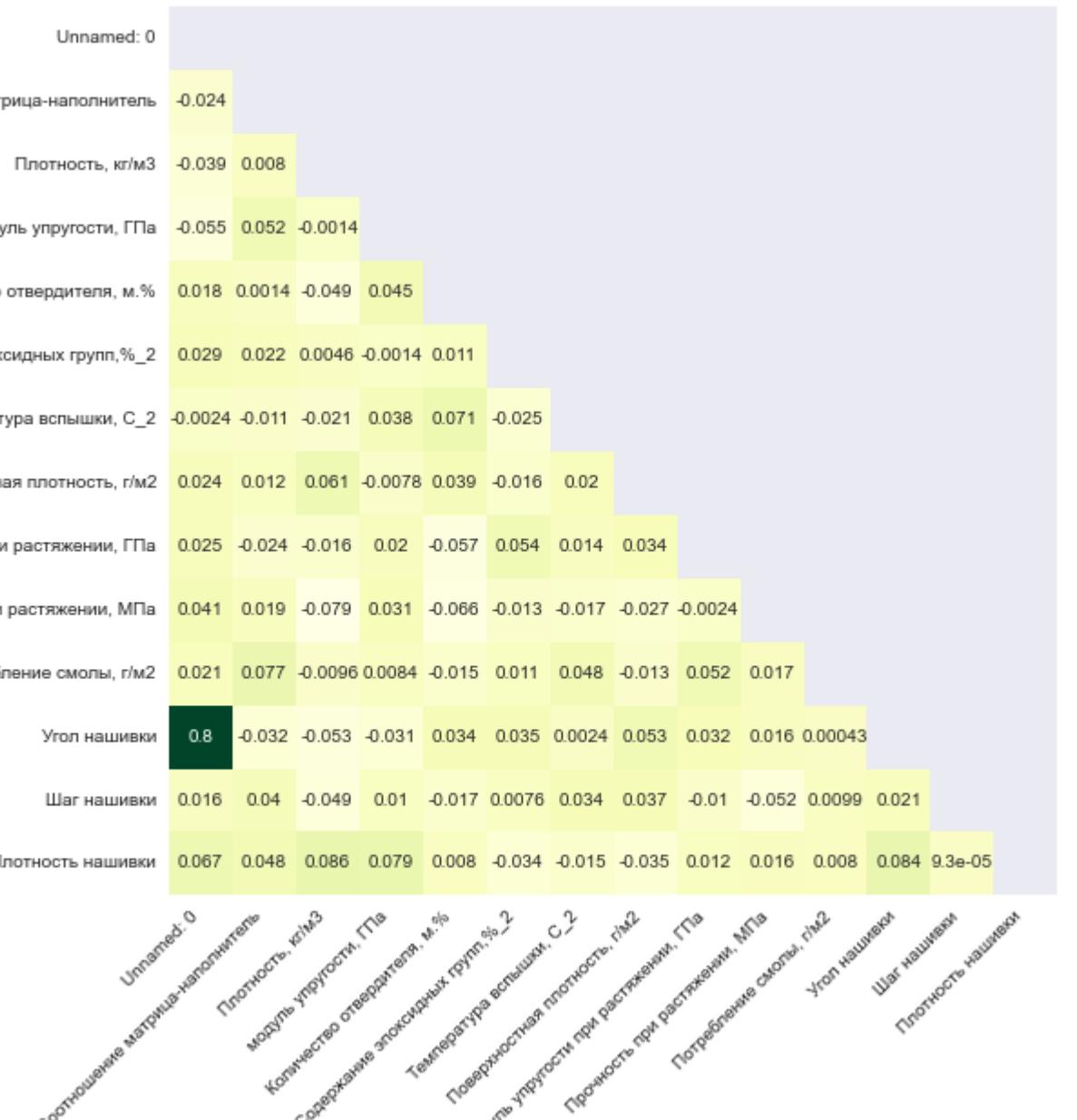
```
for i in df.columns:  
    plt.figure(figsize = (6, 4))  
    res = stats.probplot(df[i], plot = plt)  
    plt.title(i, fontsize = 10)  
    plt.xlabel("Теоретические значения", fontsize = 10)  
    plt.ylabel("Упорядоченные значения", fontsize = 10)  
    plt.show()
```







```
In [47]: #Визуализация корреляционной матрицы с помощью тепловой карты
mask = np.triu(df.corr())
# Создаем полотно для отображения большого графика
f, ax = plt.subplots(figsize=(11, 9))
# Используя данные корреляции, и создаем тепловую панель
sns.heatmap(df.corr(), mask=mask, annot=True, square=True, cmap = 'YlGn')
plt.xticks(rotation = 45, ha = 'right')
plt.show()
```



```
In [48]: # Уфф, наконец-то, убеждаюсь, что байросб не осталась. В итоге осталось всего 922 строки
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 922 entries, 0 to 921
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   Угол нашивки    922 non-null    int64  
 1   Соотношение матрица-наполнитель 922 non-null    float64
 2   Плотность, кг/м³      922 non-null    float64
 3   модуль упругости, ГПа     922 non-null    float64
 4   Количество отвердителя, м.% 922 non-null    float64
 5   Содержание эпоксидных групп, % 922 non-null    float64
 6   Температура вспышки, С, 2  922 non-null    float64
 7   Поверхностная плотность, г/м² 922 non-null    float64
 8   Модуль упругости при растяжении, ГПа 922 non-null    float64
 9   Прочность при растяжении, МПа 922 non-null    float64
 10  Потребление смолы, г/м² 922 non-null    float64
 11  Угол нашивки    922 non-null    int64  
 12  Шаг нашивки     922 non-null    float64
 13  Плотность нашивки 922 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 140.3 KB
```

```
In [49]: #Создаю пустую, без байросб датасет, чтобы в excel проверить дополнительно
df.to_csv('Itop_Itop.csv', encoding = 'cp1251')
df.to_excel('Itop_Itop.xlsx')
```

```
In [50]: #Продолжим повторный разбивочный анализ уже без байросб
```

```
In [51]: #Выделим корреляции между параметрами
df.corr()
```

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м³	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %	Температура вспышки, С, 2	Поверхностная плотность, г/м²	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м²	Угол нашивки	Шаг нашивки	Плотность нашивки		
0	1.000000	-0.023675	-0.039305	-0.054851	0.018401	0.029299	-0.002380	0.023803	0.025344	0.041157	0.020676	0.798601	0.015623	0.067374		
1	Соотношение матрица-наполнитель	-0.023675	1.000000	0.007996	0.051643	0.001353	0.021982	-0.010565	0.019190	-0.024316	0.019141	0.076857	-0.032144	0.039924	0.047835	
2	Плотность, кг/м³	-0.039305	0.007996	1.000000	-0.001416	-0.048938	0.004568	-0.021256	0.061496	-0.015597	-0.079188	-0.009609	-0.052993	-0.048648	0.086460	
3	модуль упругости, ГПа	-0.054851	0.051643	-0.001416	1.000000	0.044550	-0.001442	0.037622	-0.007805	0.020063	0.031041	0.008368	-0.031490	0.010238	0.078810	
4	Количество отвердителя, м.%	0.018401	0.001353	0.044550	1.000000	0.011429	0.011429	0.038762	-0.057026	-0.065711	0.034103	-0.017394	0.007981			
5	Содержание эпоксидных групп, %	0.029299	0.021982	0.004568	-0.001442	0.011429	1.000000	-0.025315	0.053387	-0.013099	0.010808	0.034520	0.007571	-0.034481		
6	Температура вспышки, С, 2	-0.023800	-0.010565	-0.021256	0.037622	0.070623	-0.025315	1.000000	0.020307	0.014168	-0.017263	0.046142	0.002371	0.034395	-0.015014	
7	Поверхностная плотность, г/м²	0.023803	0.011910	0.061496	-0.007805	0.038762	-0.015844	0.020307	1.000000	0.033526	-0.027320	-0.012606	0.053180	0.036931	-0.034989	
8	Модуль упругости при растяжении, ГПа	0.025344	-0.024316	-0.015597	0.020063	0.053387	0.014168	0.033526	0.003393	1.000000	-0.002393	0.051676	0.031910	-0.010193	0.012488	
9	Прочность при растяжении, МПа	0.041157	0.019141	-0.079188	0.031041	-0.065711	-0.013099	-0.017263	-0.027320	-0.002393	1.000000	0.016753	0.016144	-0.051580	0.016311	
10	Потребление смолы, г/м²	0.020676	0.076857	-0.009609	0.008368	-0.048938	0.010808	0.048142	-0.012606	0.016753	0.009000	0.004333	0.009932	0.008012		
11	Угол нашивки	0.798601	-0.032144	-0.052993	-0.031490	0.034103	0.034520	0.053180	0.031910	0.016144	0.000433	1.000000	0.021464	0.083564		
12	Шаг нашивки	0.015623	0.039924	-0.048648	0.010238	-0.017394	0.007571	0.034395	-0.010193	-0.051580	0.009932	0.021464	1.000000	0.000093		
13	Плотность нашивки	0.067374	0.047835	0.086460	0.078810	0.007981	-0.034481	-0.015014	0.012488	0.016311	0.008012	0.083564	0.000093	1.000000		

```
In [52]: #Посмотрим на средние и медианные значения датасета после выбросов
mean_and_sd = df.describe()
mean_and_sd.loc['mean', '50%']
```

У меня есть, что после удаления выбросов среднее и медианное значение осталось в пределах предыдущих значений

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м³	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %	Температура вспышки, С, 2	Поверхностная плотность, г/м²	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м²	Угол нашивки	Шаг нашивки	Плотность нашивки
mean	460.5	2.927964	1974.118744	736.119982	111.136066	22.200570	286.181128	482.42070	73.303464	2461.491315	218.048059	0.510846	6.931939	57.562887
50%	460.5	2.907832	1977.321002	736.178435	111.162090	22.177681	286.220763	457.732246	73.247594	2455.744662	218.697660	1.000000	6.972862	57.584225

Приготовим переменные и проверим данные перед нормализацией данных

```
In [53]: df_norm = df.copy()
```

```
In [54]: df_norm.info()
```

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м³	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %	Температура вспышки, С, 2	Поверхностная плотность, г/м²	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м²	Угол нашивки	Шаг нашивки	Плотность нашивки
mean	460.5	2.927964	1974.118744	736.119982	111.136066	22.200570	286.181128	482.42070	73.303464	2461.491315	218.048059	0.510846	6.931939	57.562887
50%	460.5	2.907832	1977.321002	736.178435	111.162090	22.177681	286.220763	457.732246	73.247594	2455.744662	218.697660	1.000000	6.972862	57.584225

Предобработка данных.

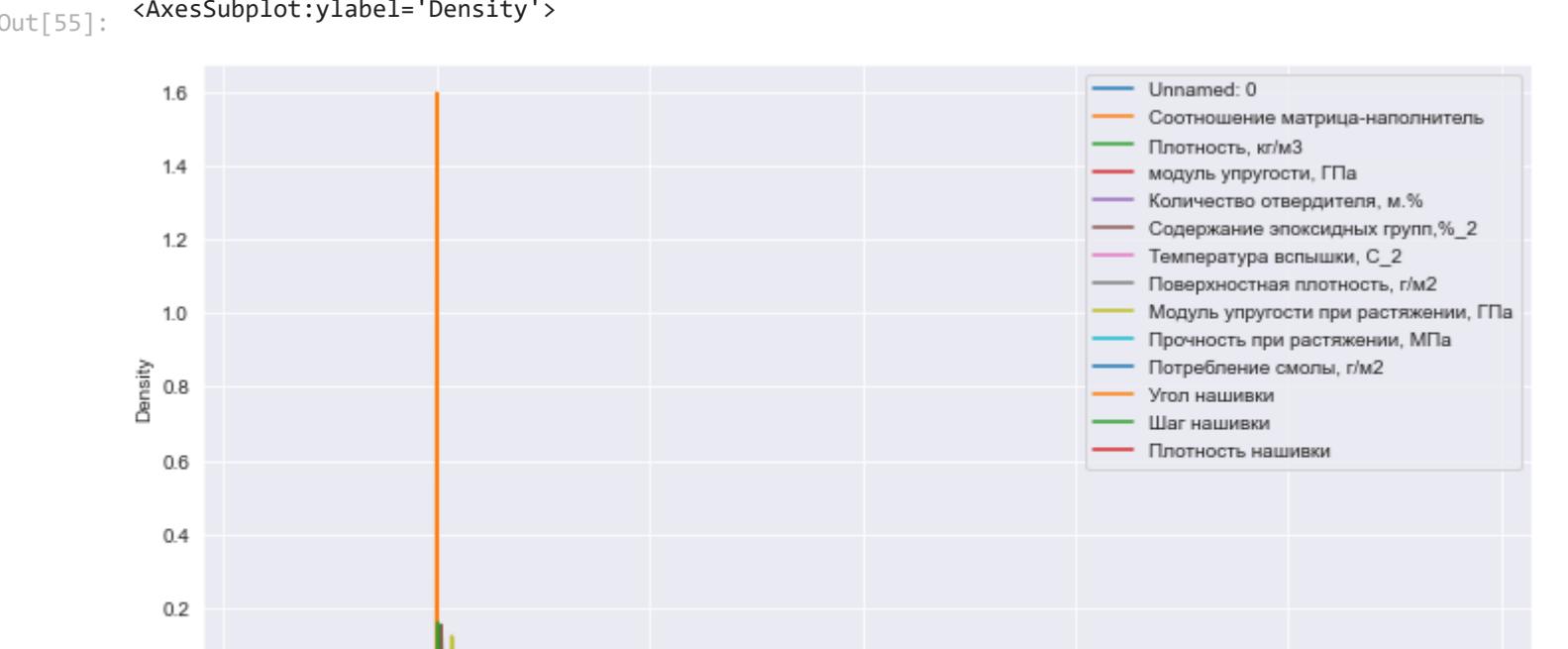
Нормализуем данные

У нас в основном количественные признаки, поэтому можно применить нормализацию (приведение в диапазон от 0 до 1) или стандартизацию (приведение к матожиданию 0, стандартному отклонению 1). Т.к. это в том числе учебная работа, то используем и нормализацию, и стандартизацию.

Этап предобработки данных нужен нам и для введенных данных в будущем приложении, которое явится результатом нашей работы.

In [55]: # На данный момент в наших датафрейме более 922 строк. Они все без бирбосов, пропущенных значений, все имеют int или float.

fig, ax = plt.subplots(figsize = (12, 6))
df_norm.plot(kind = 'kde', ax = ax)
Оценка плотности ядра показывает, что наши данные находятся в разных диапазонах. А в связи с тем, что диапазоны очень разные, данные нужно нормализовать. Можем приступать к нормализации данных.

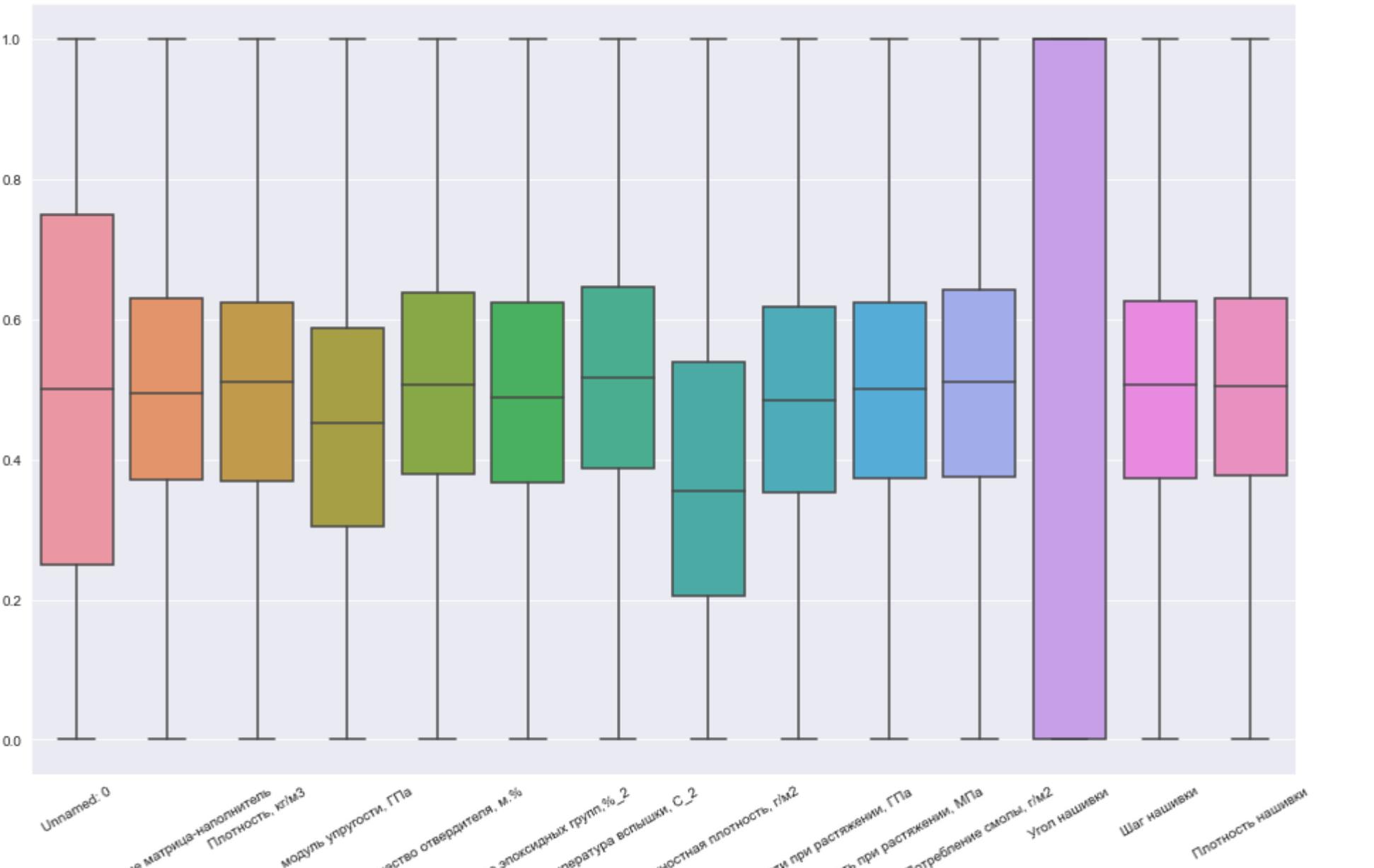


In [56]: #Нормализуем данные с помощью MinMaxScaler()
scaler = preprocessing.MinMaxScaler()
col = df.columns
result = scaler.fit_transform(df)
df_minmax_n = pd.DataFrame(result, columns = col)
df_minmax_n.describe()

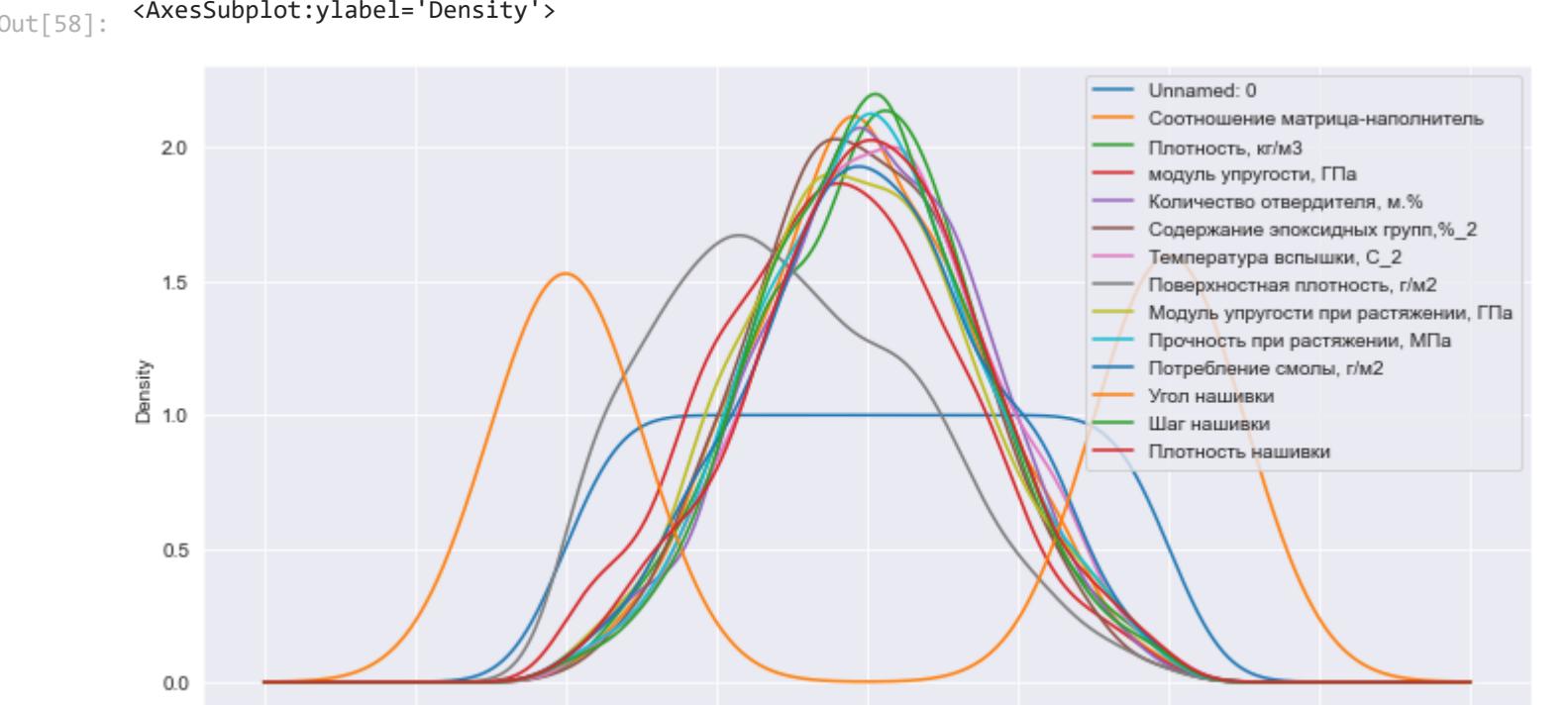
Out[56]:

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Шаг нашивки	Плотность нашивки	
count	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000		
mean	0.499412	0.502904	0.451341	0.506200	0.490578	0.516739	0.373295	0.487343	0.503776	0.507876	0.510846	0.503426	0.503938	
std	0.289145	0.187858	0.188395	0.201534	0.186876	0.180548	0.190721	0.217269	0.196366	0.188668	0.199418	0.501514	0.183587	0.193933
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.250000	0.371909	0.368184	0.305188	0.378514	0.366571	0.386228	0.204335	0.353512	0.373447	0.374647	0.000000	0.372844	0.376869
50%	0.500000	0.495189	0.511396	0.451377	0.506382	0.488852	0.516931	0.354161	0.483718	0.501481	0.510143	1.000000	0.506414	0.504310
75%	0.750000	0.629774	0.624719	0.587193	0.638735	0.623046	0.646553	0.538397	0.617568	0.624299	0.642511	1.000000	0.626112	0.630842
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [57]: plt.figure(figsize = (16, 10))
ax = sns.boxplot(data = df_minmax_n)
ax.set_xticklabels(ax.get_xticklabels(), rotation=30);



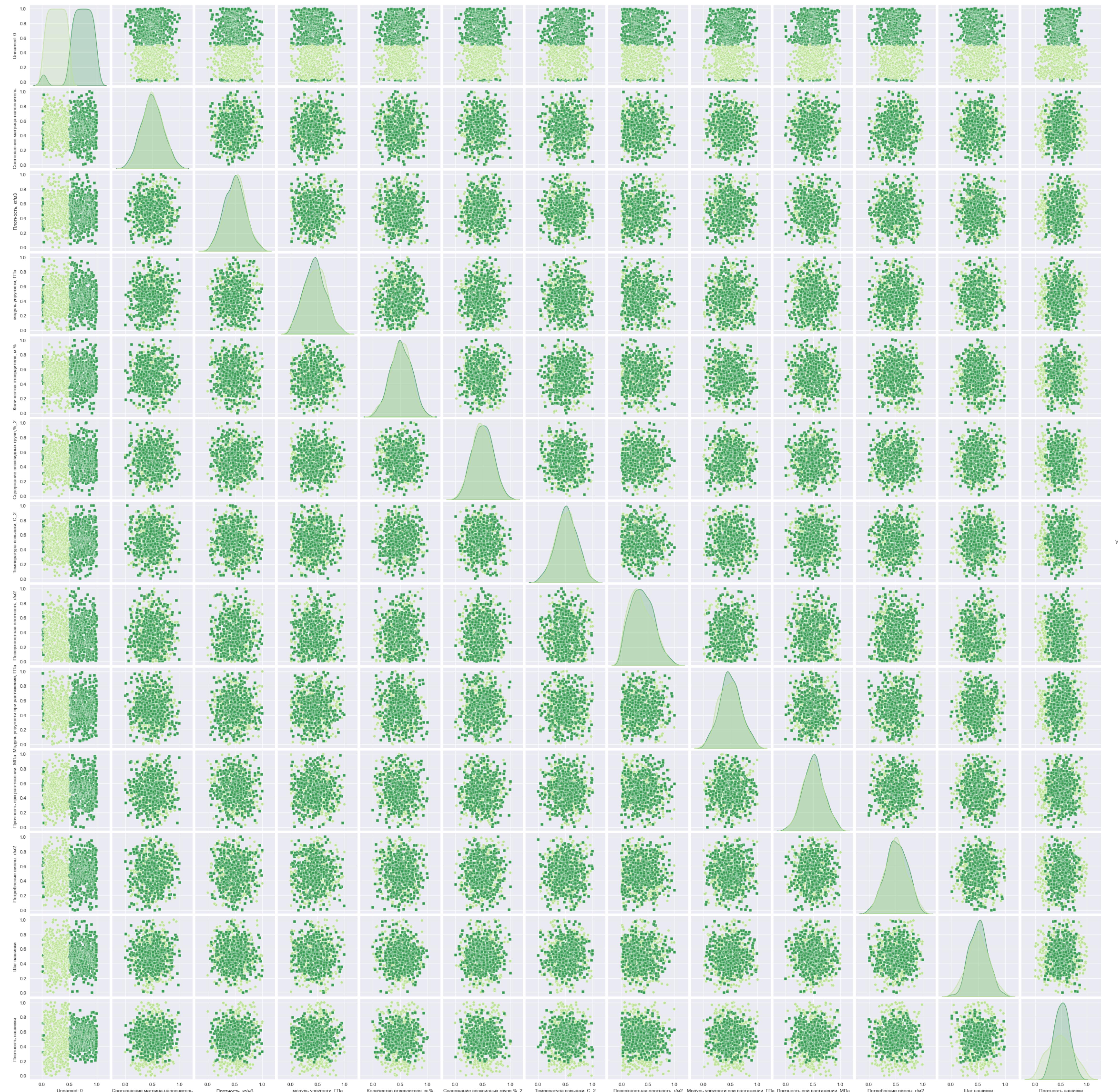
In [58]: fig, ax = plt.subplots(figsize = (12, 6))
df_minmax_n.plot(kind = 'kde', ax = ax)



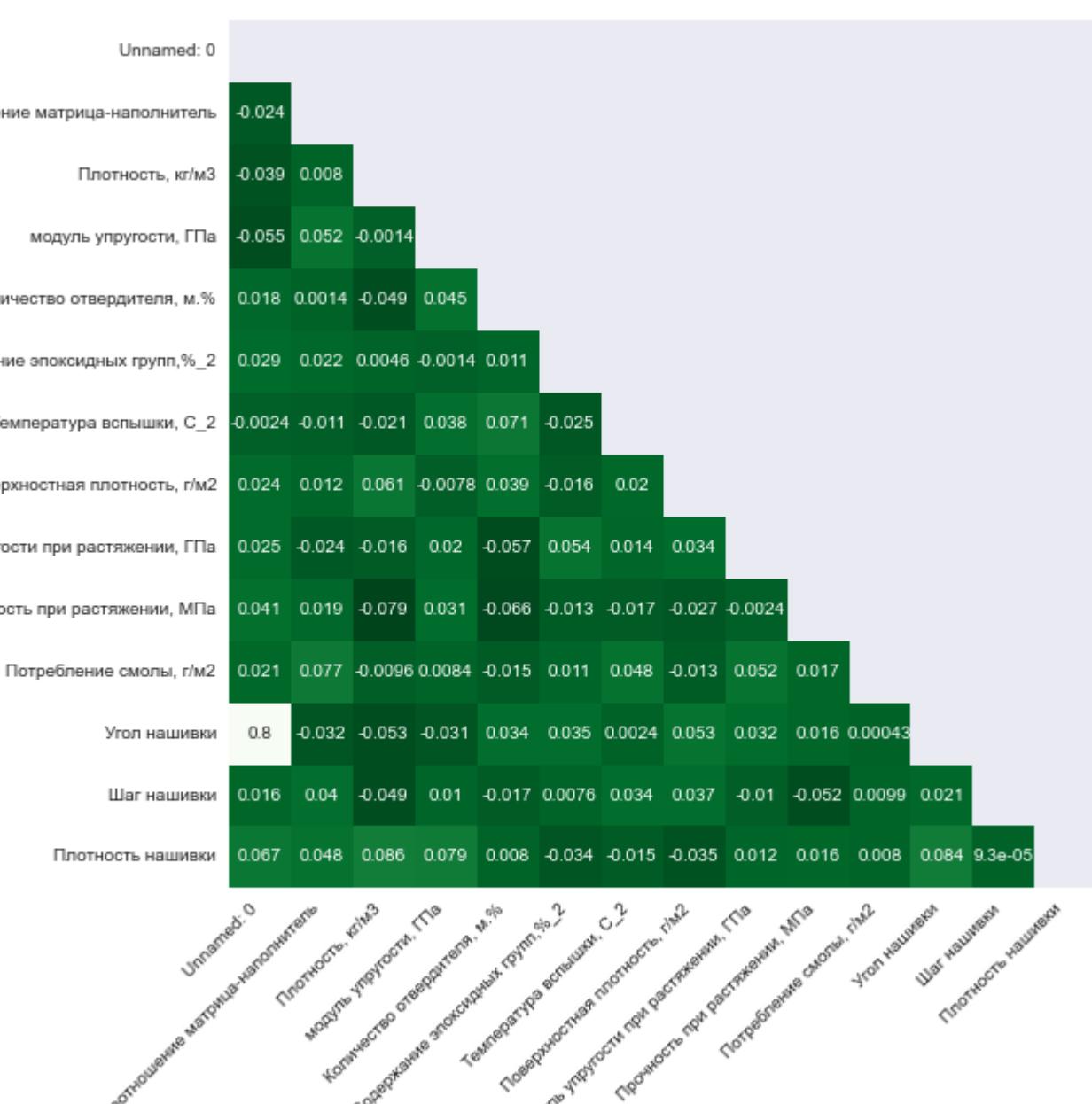
In [59]: sns.pairplot(df_minmax_n, hue = 'Шаг нашивки', markers = ["o", "s"], diag_kind = 'auto', palette = 'viridis')

Out[59]:

<seaborn.axisgrid.PairGrid at 0x1fbdb133fb50>



```
In [60]: mask = np.triu(df_minmax_n.corr())
f, ax = plt.subplots(figsize=(11, 9))
sns.heatmap(df_minmax_n.corr(), mask=mask, annot=True, square=True, cmap='Greens_r')
plt.xticks(rotation=45, ha='right')
plt.show()
```



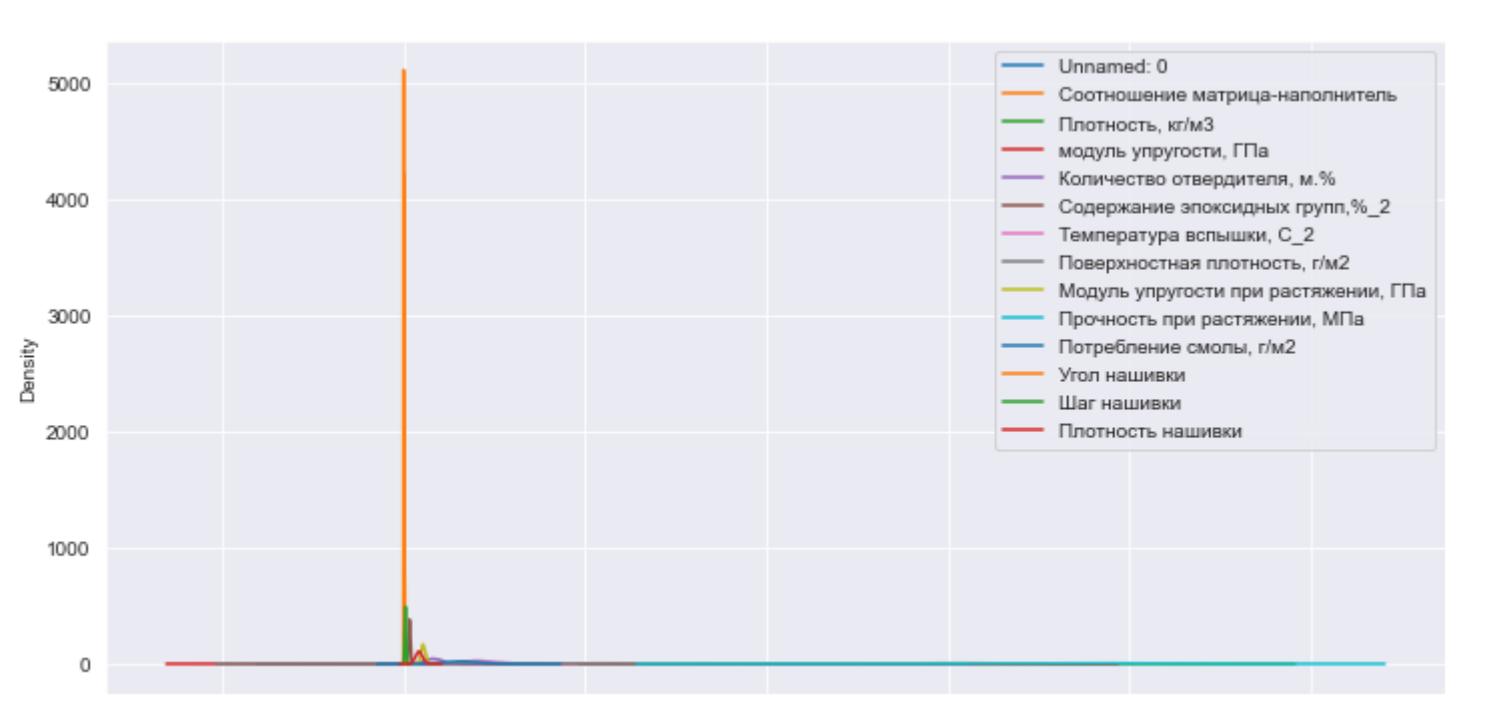
In [61]: #Нормализуем данные с помощью Normalizer()

```
normalizer = Normalizer()
res = normalizer.fit_transform(df)
df_norm_n = pd.DataFrame(res, columns = df.columns)
df_norm_n
```

```
Out[61]: Unnamed: 0 Соотношение матрица-наполнитель Плотность, кг/м3 модуль упругости, ГПа Количество отвердителя, м.% Содержание эпоксидных групп,%_2 Температура вспышки, С_2 Поверхностная плотность, г/м2 Модуль упругости при растяжении, ГПа Прочность при растяжении, МПа Потребление смолы, г/м2 Угол нашивки Шаг нашивки Плотность нашивки
0 0.000000 0.000499 0.545436 0.198490 0.013434 0.006381 0.076473 0.056424 0.018808 0.805064 0.059111 0.000000 0.001075 0.016121
1 0.000268 0.000499 0.545011 0.198335 0.034634 0.005705 0.080543 0.056380 0.018793 0.805435 0.059065 0.000000 0.001342 0.016168
2 0.000537 0.000744 0.544829 0.202097 0.030022 0.005976 0.076388 0.056362 0.018787 0.805167 0.059046 0.000000 0.001342 0.015298
3 0.000809 0.000746 0.539270 0.201687 0.030161 0.006004 0.076742 0.056623 0.018874 0.808906 0.059320 0.000000 0.001348 0.016178
4 0.001089 0.000699 0.519919 0.219672 0.030449 0.006062 0.077475 0.057164 0.019055 0.816626 0.059886 0.000000 0.001361 0.019055
... ...
917 0.272016 0.000674 0.579061 0.270787 0.025905 0.005969 0.096340 0.062056 0.021681 0.708159 0.037082 0.000297 0.002692 0.015948
918 0.276207 0.001036 0.616828 0.133811 0.043923 0.005897 0.076488 0.105506 0.021940 0.710192 0.035422 0.000301 0.003179 0.016172
919 0.257998 0.000921 0.553720 0.117022 0.031031 0.006726 0.069742 0.207786 0.020981 0.747579 0.066425 0.000281 0.001168 0.018986
920 0.283708 0.001143 0.637357 0.228655 0.043604 0.005935 0.080545 0.197815 0.022833 0.638873 0.060789 0.000308 0.001947 0.017966
921 0.250777 0.001037 0.514736 0.113630 0.035175 0.007481 0.081946 0.206598 0.020234 0.777743 0.050329 0.000272 0.001655 0.021084
```

922 rows × 14 columns

```
In [62]: fig, ax = plt.subplots(figsize = (12, 6))
df_norm_n.plot(kind = "kde", ax = ax)
AxesSubplot:ylabel='Density'
```



```
In [63]: #Создадим с данными до нормализации
df_head(10)
```

```
Out[63]: Unnamed: 0 Соотношение матрица-наполнитель Плотность, кг/м3 модуль упругости, ГПа Количество отвердителя, м.% Содержание эпоксидных групп,%_2 Температура вспышки, С_2 Поверхностная плотность, г/м2 Модуль упругости при растяжении, ГПа Прочность при растяжении, МПа Потребление смолы, г/м2 Угол нашивки Шаг нашивки Плотность нашивки
0 0 1.857143 2030.0 738.736842 50.00 23.750000 284.615385 210.0 70.0 3000.0 220.0 0 4.0 60.0
1 1 1.857143 2030.0 738.736842 129.00 21250000 300.000000 210.0 70.0 3000.0 220.0 0 5.0 47.0
2 2 2.771331 2030.0 753.000000 111.86 22.267857 284.615385 210.0 70.0 3000.0 220.0 0 5.0 57.0
3 3 2.767918 2000.0 748.000000 111.86 22.267857 284.615385 210.0 70.0 3000.0 220.0 0 5.0 60.0
4 4 2.569620 1910.0 807.000000 111.86 22.267857 284.615385 210.0 70.0 3000.0 220.0 0 5.0 70.0
5 5 2.561475 1900.0 535.000000 111.86 22.267857 284.615385 380.0 75.0 1800.0 120.0 0 7.0 47.0
6 6 3.557018 1930.0 889.000000 129.00 21250000 300.000000 380.0 75.0 1800.0 120.0 0 7.0 57.0
7 7 3.532338 2100.0 1421.000000 129.00 21250000 300.000000 1010.0 78.0 2000.0 300.0 0 7.0 60.0
8 8 2.919678 2160.0 933.000000 129.00 21250000 300.000000 1010.0 78.0 2000.0 300.0 0 7.0 70.0
9 9 2.877358 1990.0 1628.000000 129.00 21250000 300.000000 1010.0 78.0 2000.0 300.0 0 9.0 47.0
```

```
In [64]: # Проверяем перевод данных из нормализованных в исходные
col = df_minmax_n.columns
result_reverse = scaler.inverse_transform(df_minmax_n)
initial_data = pd.DataFrame(result_reverse, columns = col)
initial_data.head(10)
```

```
Out[64]: Unnamed: 0 Соотношение матрица-наполнитель Плотность, кг/м3 модуль упругости, ГПа Количество отвердителя, м.% Содержание эпоксидных групп,%_2 Температура вспышки, С_2 Поверхностная плотность, г/м2 Модуль упругости при растяжении, ГПа Прочность при растяжении, МПа Потребление смолы, г/м2 Угол нашивки Шаг нашивки Плотность нашивки
0 0.0 1.857143 2030.0 738.736842 50.00 23.750000 284.615385 210.0 70.0 3000.0 220.0 0 4.0 60.0
1 1.0 1.857143 2030.0 738.736842 129.00 21250000 300.000000 210.0 70.0 3000.0 220.0 0 5.0 47.0
2 2.0 2.771331 2030.0 753.000000 111.86 22.267857 284.615385 210.0 70.0 3000.0 220.0 0 5.0 57.0
3 3.0 2.767918 2000.0 748.000000 111.86 22.267857 284.615385 210.0 70.0 3000.0 220.0 0 5.0 60.0
4 4.0 2.569620 1910.0 807.000000 111.86 22.267857 284.615385 210.0 70.0 3000.0 220.0 0 5.0 70.0
5 5.0 2.561475 1900.0 535.000000 111.86 22.267857 284.615385 380.0 75.0 1800.0 120.0 0 7.0 47.0
6 6.0 3.557018 1930.0 889.000000 129.00 21250000 300.000000 380.0 75.0 1800.0 120.0 0 7.0 57.0
7 7.0 3.532338 2100.0 1421.000000 129.00 21250000 300.000000 1010.0 78.0 2000.0 300.0 0 7.0 60.0
8 8.0 2.919678 2160.0 933.000000 129.00 21250000 300.000000 1010.0 78.0 2000.0 300.0 0 7.0 70.0
9 9.0 2.877358 1990.0 1628.000000 129.00 21250000 300.000000 1010.0 78.0 2000.0 300.0 0 9.0 47.0
```

```
In [65]: #Рассмотрим несколько вариантов корреляции между параметрами после нормализации (первый вариант)
df_norm_n[df_norm_n.columns].corr()
```

```
Out[65]: Unnamed: 0 Соотношение матрица-наполнитель Плотность, кг/м3 модуль упругости, ГПа Количество отвердителя, м.% Содержание эпоксидных групп,%_2 Температура вспышки, С_2 Поверхностная плотность, г/м2 Модуль упругости при растяжении, ГПа Прочность при растяжении, МПа Потребление смолы, г/м2 Угол нашивки Шаг нашивки Плотность нашивки
   Угол нашивки 1.000000 -0.024313 0.028357 -0.085713 0.030100 0.051446 0.021113 -0.011376 0.047641 -0.191178 0.026305 0.97244 0.035460 0.089432
   Соотношение матрица-наполнитель 0.000000 -0.024313 1.000000 0.280171 0.035787 0.134893 0.219623 0.177884 0.025246 0.261010 -0.249325 0.157188 0.042057 0.135479 0.171938
   Плотность, кг/м3 0.028357 0.280171 1.000000 -0.036469 0.384277 0.650936 0.557692 0.069777 0.8171372 -0.828491 0.308873 -0.011069 0.285142 0.429998
   модуль упругости, ГПа 0.035787 -0.036469 1.000000 0.051490 -0.024971 0.070553 -0.023131 0.021747 -0.365297 -0.007789 -0.059148 0.004350 0.046465
   Количество отвердителя, м.% 0.030100 0.134893 0.384277 1.000000 0.298724 0.305717 0.068327 0.370752 -0.375533 0.138065 0.027012 0.117672 0.195089
   Содержание эпоксидных групп,%_2 0.051446 0.219623 0.650936 -0.024971 0.298724 1.000000 0.396485 0.027019 0.652763 -0.544787 0.238866 0.032763 0.230545 0.266271
   Температура вспышки, С_2 0.021113 0.177884 0.557692 0.007553 0.305717 0.396485 1.000000 0.039367 0.556485 -0.502367 0.241478 0.049424 0.211308 0.248782
   Поверхностная плотность, г/м2 -0.01376 0.025246 0.069777 -0.023131 0.068327 0.027019 0.039367 1.000000 0.057128 -0.309697 0.006518 0.029093 0.066656 -0.010737
   Модуль упругости при растяжении, ГПа 0.047641 0.261010 0.8171372 0.007553 -0.023131 0.037622 0.020685 0.057128 -0.735849 0.130000 0.324273 0.020469 0.295766 0.400691
   Прочность при растяжении, МПа -0.191178 0.026305 0.035460 0.089432 0.035460 0.042057 0.135479 0.171938 0.006518 0.324273 0.020469 0.295766 0.400691
   Потребление смолы, г/м2 0.035460 0.135479 0.285142 0.429998 0.285142 0.044350 0.068327 0.032763 0.238866 0.035460 0.042057 0.135479 0.171938 0.429998
   Угол нашивки 0.072718 0.148993 0.211308 0.248782 0.248782 0.049424 0.020469 0.031134 0.085510 0.020469 0.042057 0.135479 0.171938 0.429998
   Шаг нашивки 0.010169 0.017394 0.066656 0.027012 0.027012 0.049424 0.020469 0.031134 0.085510 0.020469 0.042057 0.135479 0.171938 0.429998
   Плотность нашивки 0.089432 0.171938 0.429998 0.046465 0.046465 0.195089 0.026271 0.027012 0.039126 0.040091 0.042057 0.135479 0.171938 0.429998
```

```
In [66]: #Рассмотрим второй вариант корреляции между параметрами после нормализации (второй вариант)
df_minmax_n[df_minmax_n.columns].corr()
```

```
Out[66]: Unnamed: 0 Соотношение матрица-наполнитель Плотность, кг/м3 модуль упругости, ГПа Количество отвердителя, м.% Содержание эпоксидных групп,%_2 Температура вспышки, С_2 Поверхностная плотность, г/м2 Модуль упругости при растяжении, ГПа Прочность при растяжении, МПа Потребление смолы, г/м2 Угол нашивки Шаг нашивки Плотность нашивки
   Угол нашивки 1.000000 -0.023675 0.007996 0.051643 0.001353 0.021982 -0.010565 0.019190 -0.024316 0.019141 0.076857 -0.032144 0.039924 0.047835
   Соотношение матрица-наполнитель -0.023675 1.000000 0.007996 0.051643 0.001353 0.021982 -0.010565 0.019190 -0.024316 0.019141 0.076857 -0.032144 0.039924 0.047835
   Плотность, кг/м3 -0.039305 0.007996 1.000000 -0.001416 0.048938 0.004568 -0.021256 0.061496 -0.015597 -0.079188 -0.009609 -0.052993 -0.048648 0.088460
   модуль упругости, ГПа 0.054851 0.051643 -0.001416 1.000000 0.044550 -0.001442 0.037622 -0.007805 0.020663 0.031041 0.008368 -0.031490 0.010238 0.078810
   Количество отвердителя, м.% 0.018401 0.001353 -0.048938 0.044550 1.000000 0.011429 0.070623 0.038762 -0.057106 -0.014827 0.034103 -0.017394 0.007981
   Содержание эпоксидных групп,%_2 0.029299 0.021982 0.004568 -0.001442 0.011429 1.000000 -0.025315 0.053887 -0.013099 0.016808 0.034520 0.007571 -0.034481
   Температура вспышки, С_2 -0.002380 0.010565 -0.021256 0.037622 -0.025315 0.000000 1.000000 0.020307 0.014168 0.048142 0.002371 0.034395 -0.015014
   Поверхностная плотность, г/м2 0.023803 0.011910 0.061496 -0.007805 0.038762 -0.015844 0.0020307 1.000000 0.033526 -0.027320 -0.012606 0.053180 0.036931 -0.034989
   Модуль упругости при растяжении, ГПа 0.025344 -0.024316 -0.015597 0.020663 -0.057026 0.053887 0.014168 0.033526 1.000000 -0.
```

Out[67]:

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки	Шаг нашивки	Плотность нашивки
0	0.000000	0.274768	0.651097	0.452951	0.079153	0.607455	0.509164	0.162230	0.272962	0.727777	0.514688	0.0	0.289334	0.546433
1	0.001086	0.274768	0.651097	0.452951	0.0630983	0.418887	0.583596	0.162230	0.272962	0.727777	0.514688	0.0	0.362355	0.319758
2	0.002172	0.466552	0.651097	0.461725	0.511257	0.495653	0.509164	0.162230	0.272962	0.727777	0.514688	0.0	0.362355	0.494123
3	0.003257	0.465836	0.571539	0.458649	0.511257	0.495653	0.509164	0.162230	0.272962	0.727777	0.514688	0.0	0.362355	0.546433
4	0.004343	0.424236	0.332665	0.494944	0.511257	0.495653	0.509164	0.162230	0.272962	0.727777	0.514688	0.0	0.362355	0.720799
...
917	0.95657	0.361662	0.444480	0.560064	0.337550	0.333908	0.703458	0.161609	0.473553	0.472912	0.183151	1.0	0.660014	0.320103
918	0.996743	0.607674	0.704573	0.272088	0.749605	0.294428	0.362087	0.271207	0.462512	0.461722	0.157752	1.0	0.768759	0.437468
919	0.97928	0.573391	0.498274	0.254927	0.501991	0.623085	0.334063	0.572959	0.580201	0.587558	0.572648	1.0	0.301102	0.679468
920	0.988914	0.662497	0.748688	0.454635	0.717585	0.267818	0.466417	0.496511	0.535317	0.341643	0.434855	1.0	0.458245	0.516112
921	1.000000	0.684036	0.280923	0.255222	0.632264	0.888354	0.588206	0.587373	0.552644	0.668015	0.426577	1.0	0.441137	0.850430

922 rows × 14 columns

In [68]: df_norm_n

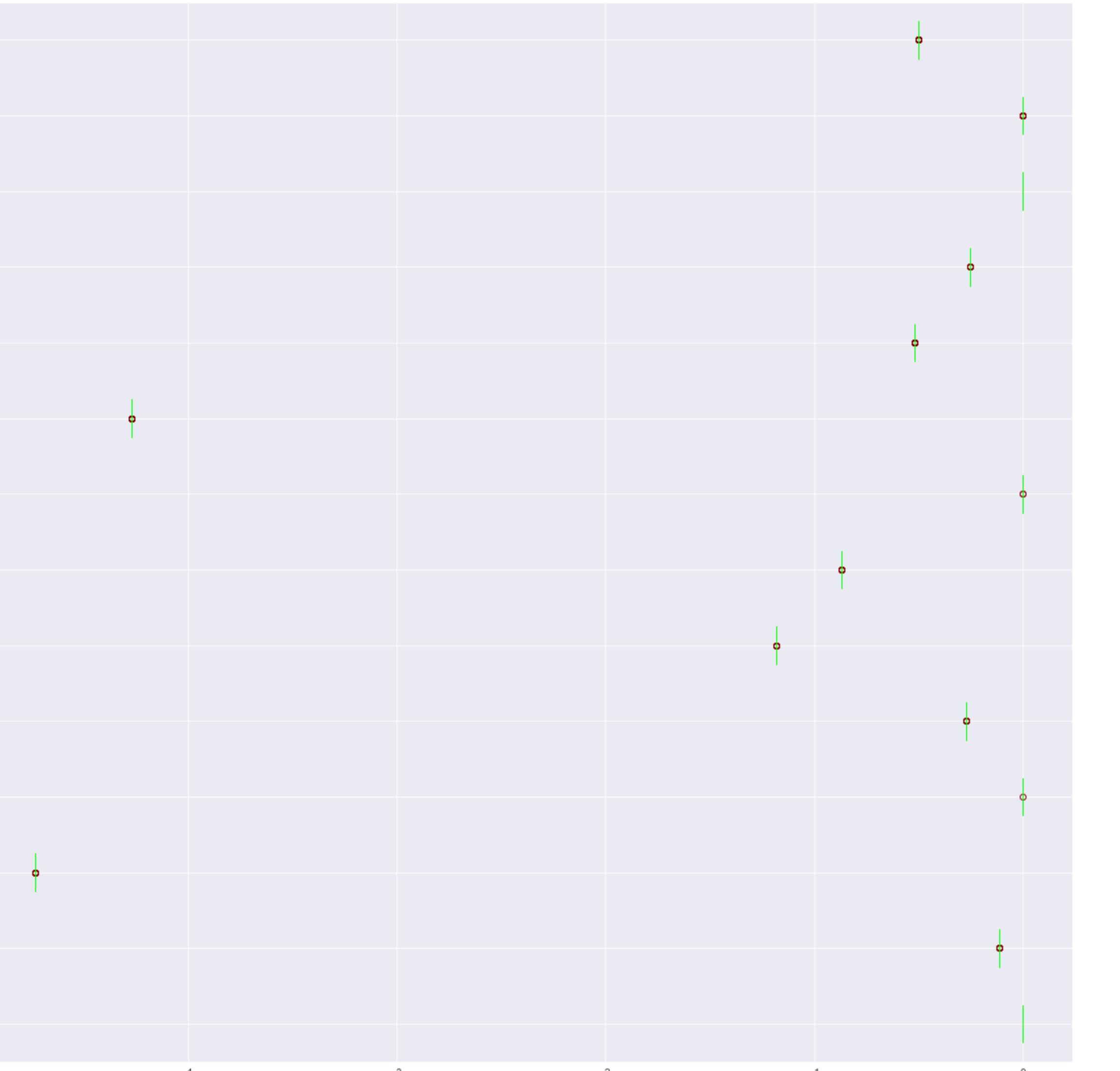
Out[68]:

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки	Шаг нашивки	Плотность нашивки
0	0.000000	0.000499	0.545436	0.198490	0.013434	0.006381	0.076473	0.056424	0.018808	0.808064	0.059111	0.000000	0.001075	0.016121
1	0.000268	0.000499	0.545011	0.198335	0.034634	0.005705	0.080543	0.056380	0.018793	0.805435	0.059065	0.000000	0.001342	0.012618
2	0.000537	0.000744	0.544829	0.202097	0.30022	0.005976	0.076388	0.056362	0.018787	0.805167	0.059046	0.000000	0.001342	0.015298
3	0.000809	0.000746	0.539270	0.201687	0.030161	0.006004	0.076742	0.056623	0.018874	0.808096	0.059320	0.000000	0.001348	0.016178
4	0.001089	0.000699	0.519919	0.219672	0.303049	0.006062	0.077475	0.057164	0.019055	0.816626	0.059886	0.000000	0.001361	0.019055
...
917	0.272016	0.000674	0.579061	0.270787	0.253605	0.005969	0.096340	0.062056	0.021681	0.70159	0.037082	0.000297	0.002692	0.013948
918	0.276207	0.001036	0.616828	0.133811	0.043923	0.005897	0.076488	0.105506	0.021940	0.710192	0.035422	0.000301	0.003179	0.016172
919	0.25798	0.000921	0.553720	0.117022	0.031031	0.006726	0.069742	0.207786	0.020981	0.747579	0.066425	0.000281	0.001168	0.018986
920	0.283708	0.001143	0.637357	0.228655	0.043604	0.005935	0.085045	0.197815	0.022833	0.638873	0.060789	0.000308	0.001947	0.017966
921	0.25077	0.001037	0.514736	0.113630	0.035175	0.007481	0.081946	0.206598	0.020234	0.777743	0.053029	0.000272	0.001655	0.021084

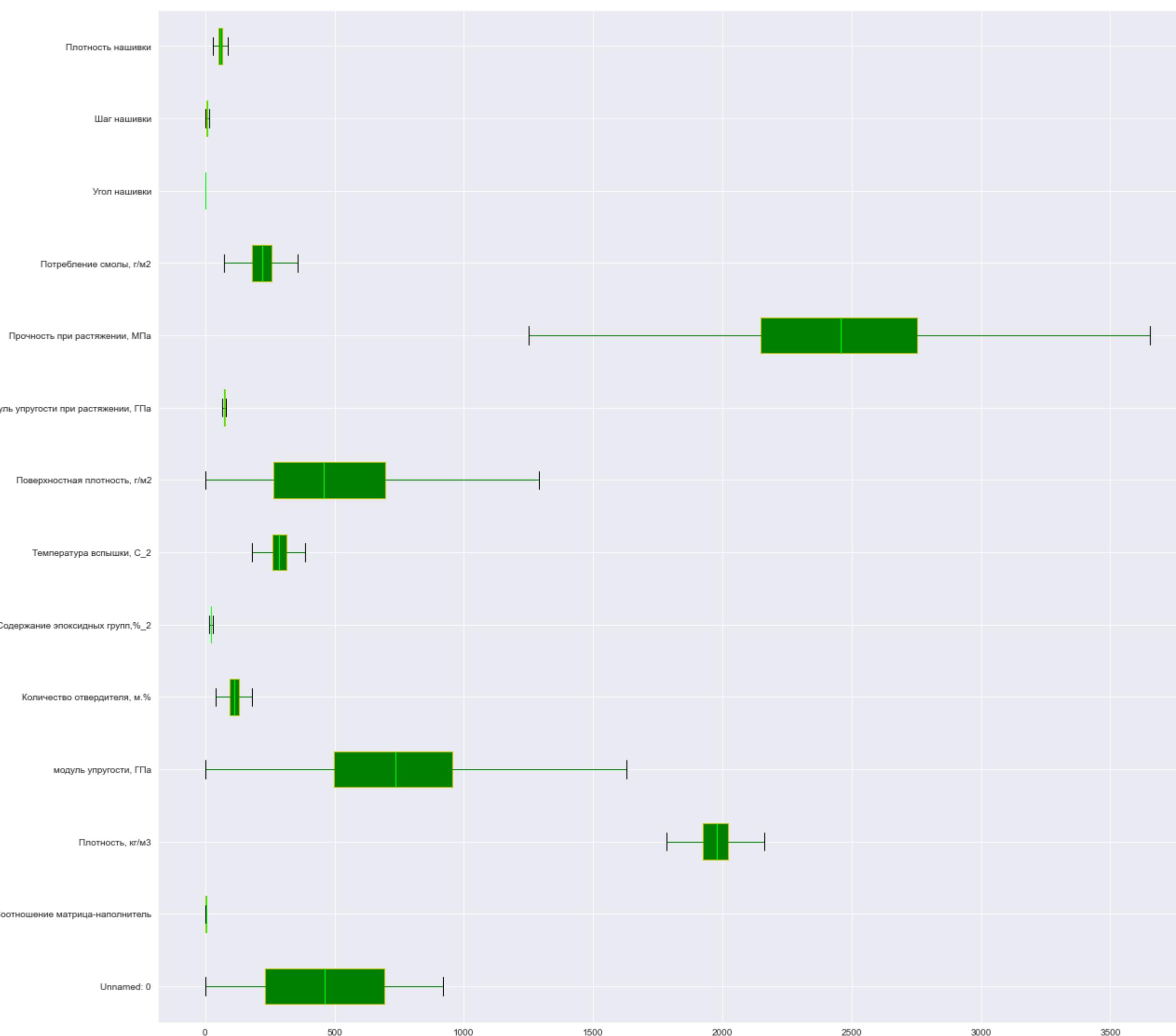
922 rows × 14 columns

In [69]: #Помимо лежачих с усами
scaler = MinMaxScaler()
scaler.fit(df)

plt.figure(figsize=(20, 20))
#Будут лежачими
plt.boxplot(pd.DataFrame(scaler.transform(df_norm_n)), labels = df_norm_n.columns, patch_artist = True, meanline = True, vert = False, boxprops = dict(facecolor = 'g', color = 'y'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = "g"), capprops = dict(color="black"), flierprops = dict(color = "y", markeredgecolor = "maroon"))
plt.show()



In [70]: #Помимо лежачими с усами
scaler = MinMaxScaler()
scaler.fit(df_minmax_n)
plt.figure(figsize = (20, 20))
#Будут лежачими
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels = df_minmax_n.columns, patch_artist = True, meanline = True, vert = False, boxprops = dict(facecolor = 'g', color = 'y'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = "g"), capprops = dict(color="black"), flierprops = dict(color = "y", markeredgecolor = "maroon"))
plt.show()



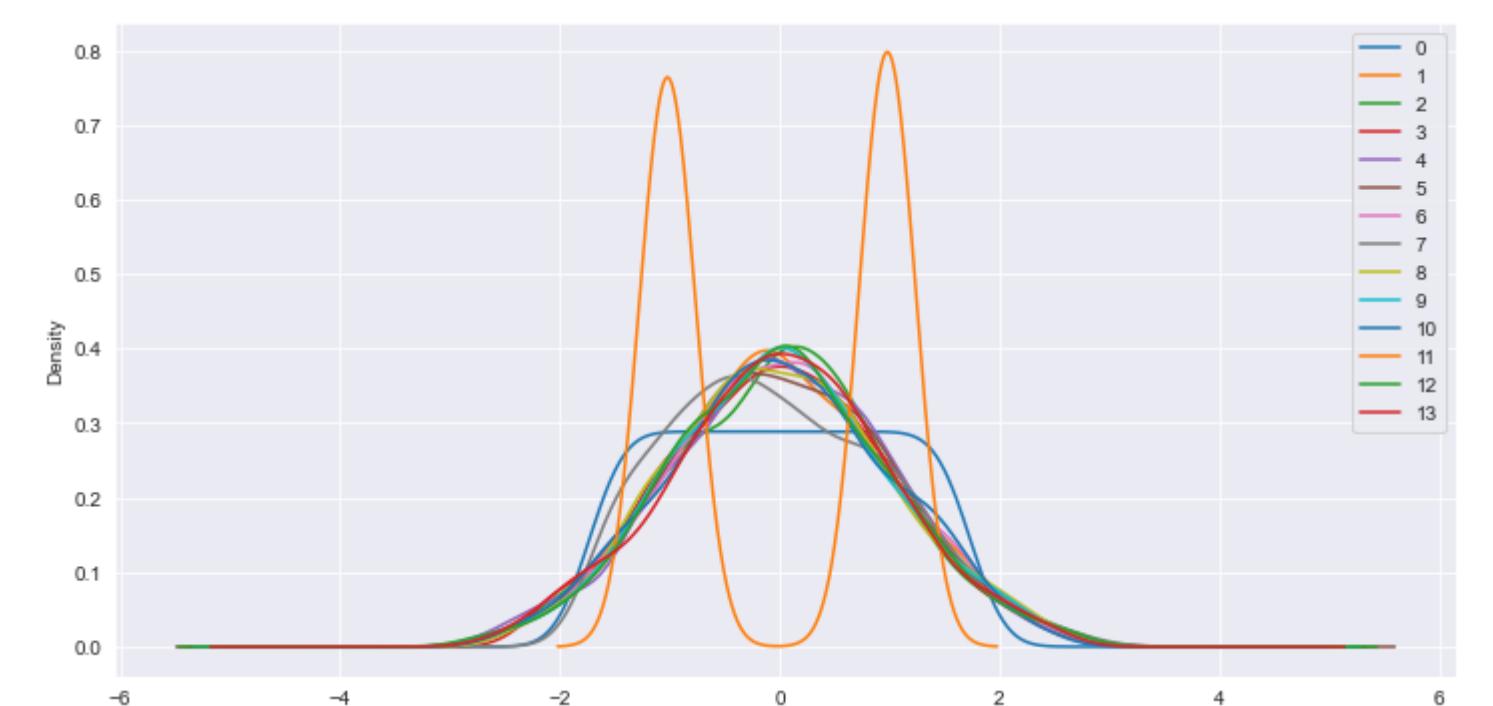
In [71]: # Визуализация графиков показывает, что нормализация при помощи "Normalizer" дает нам большое количество фибростр

Стандартизируем данные

```
In [72]: X1 = df_maxmax_n.copy()
X2 = df_norm_n.copy()

In [73]: df_std_X1 = preprocessing.StandardScaler().fit(X1)
df_standart_1 = pd.DataFrame(df_std_X1)

In [74]: fig, ax = plt.subplots(figsize = (12, 6))
df_standart_1.plot(kind = 'kde', ax = ax)
<AxesSubplot: xlabel='Density'>
```



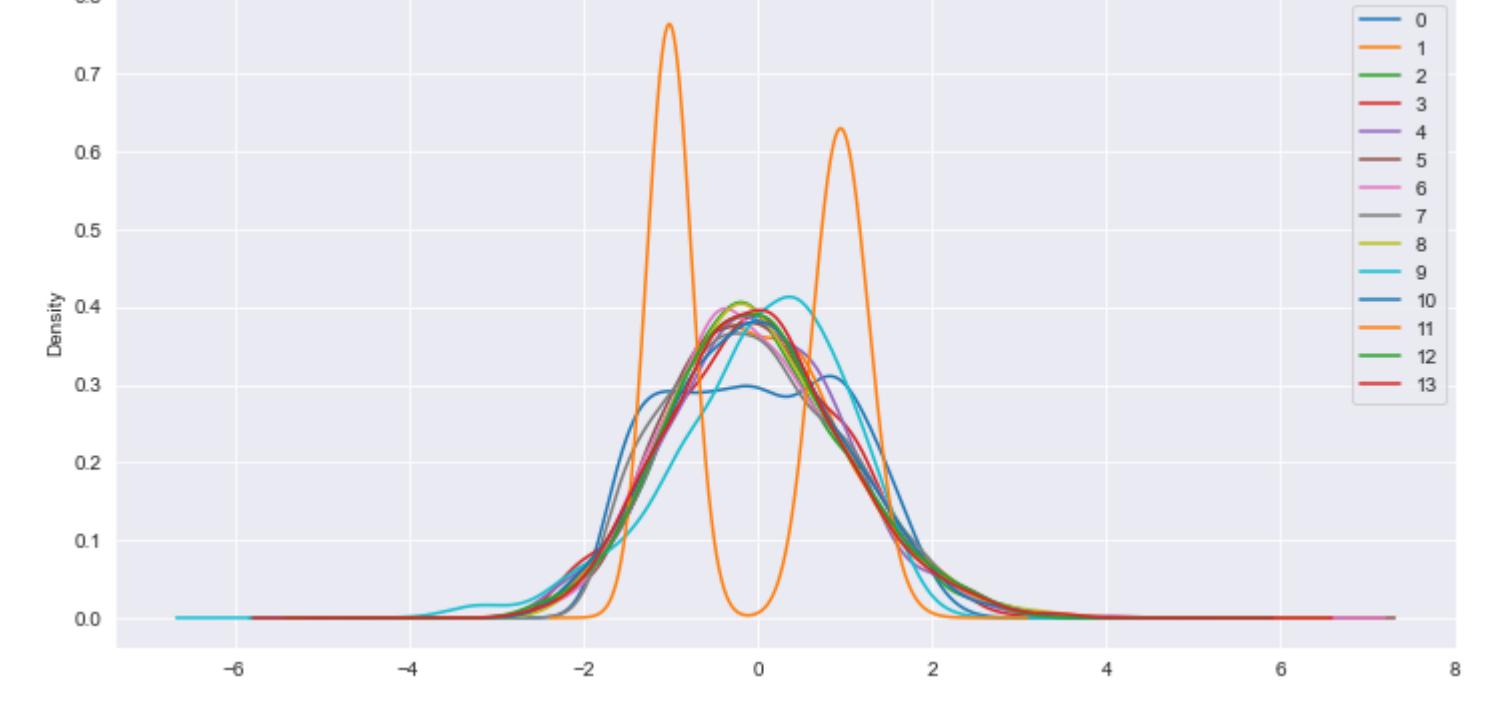
```
In [75]: df_standart_1
Out[75]:
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	-1.730173	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
1	-1.726416	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
2	-1.722659	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
3	-1.718902	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
4	-1.715145	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
...
917	1.715145	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
918	1.718902	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
919	1.722659	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
920	1.726416	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449
921	1.730173	-0.733662	-0.310284	0.539767	-0.902956	-0.868217	0.979545	-0.974830	-0.070267	-0.163679	-0.162944	0.978538	0.853400	-0.948449

922 rows x 14 columns

```
In [76]: df_std_X2 = preprocessing.StandardScaler().fit(X2)
df_standart_2 = df_std_X2.transform(X2)
df_standart_2 = pd.DataFrame(df_standart_2)

In [77]: fig, ax = plt.subplots(figsize = (12, 6))
df_standart_2.plot(kind = 'kde', ax = ax)
<AxesSubplot: xlabel='Density'>
```



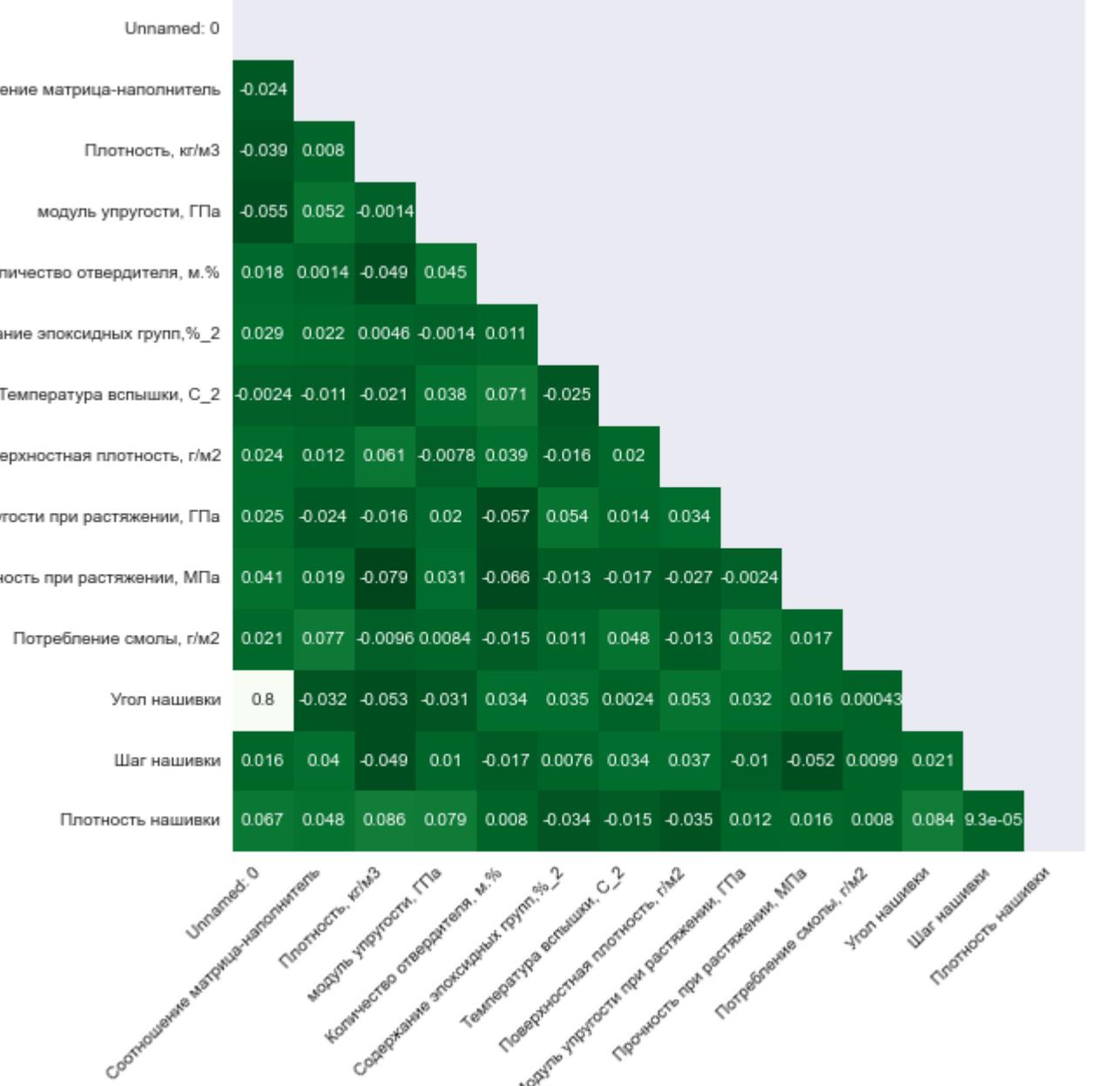
In [78]: df_standart_2

```
Out[78]:
```

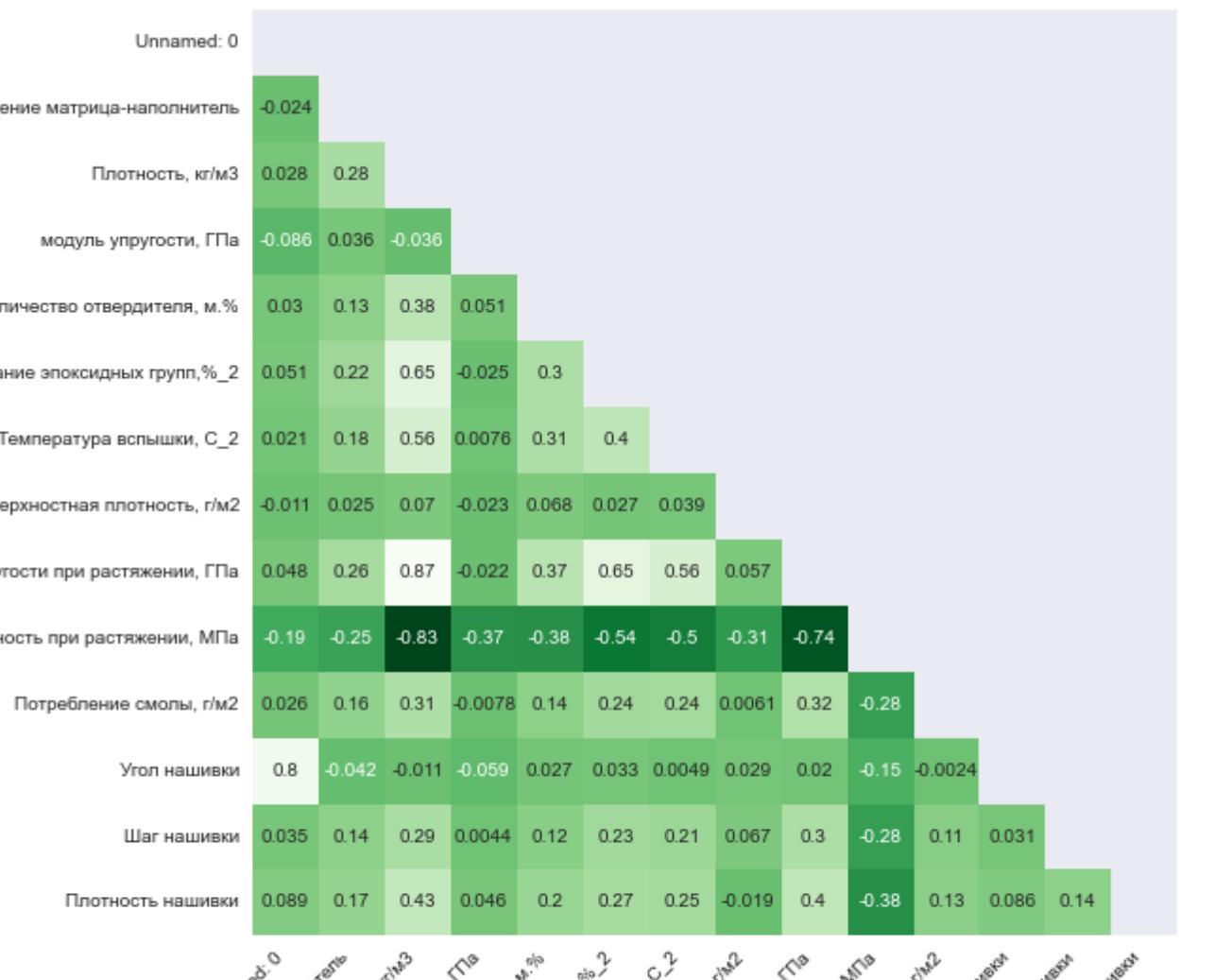
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	-1.724444	-1.329972	-0.700777	-0.199459	-2.242397	-0.250952	-0.699386	-1.045715	-1.276054	1.205017	-0.326616	-1.011172	-1.253970	-0.282841
1	-1.721052	-1.331354	-0.707561	-0.201094	0.158850	-0.931277	-0.333386	-1.046426	-1.282131	1.196004	-0.329148	-1.011172	-0.917959	-1.223240
2	-1.717662	-0.460974	-0.710455	-0.161372	-0.362737	-0.658186	-0.615158	-1.046473	-1.284723	1.192159	-0.330237	-1.011172	-0.918520	-0.503822
3	-1.714223	-0.451979	-0.799074	-0.165697	-0.346938	-0.630247	-0.591104	-1.043314	-1.248952	1.245744	-0.315178	-1.011172	-0.910696	-0.267581
4	-1.710886	-0.618305	-1.107575	0.02405	-0.314317	-0.572557	-0.541437	-1.036791	-1.173988	1.356585	-0.284104	-1.011172	-0.894541	0.504676
...
917	1.712483	-0.709564	-0.164740	0.563907	-0.840584	-0.665383	0.737759	-0.977744	0.086138	-0.198020	-1.535884	0.973430	0.776909	-0.866358
918	1.765433	0.571121	0.437335	-0.882390	1.212475	-0.738006	-0.608349	-0.453314	0.021061	-0.168883	-1.626759	1.001810	1.378791	-0.269097
919	1.535370	0.168035	-0.248403	-0.056959	-0.24403	-0.105788	0.781166	-0.376314	0.366898	0.074761	0.867000	-1.365786	0.486316	0.212339
920	1.860216	0.954882	0.764596	0.119051	1.176352	-0.699560	-0.028154	0.660817	0.330817	-1.190931	-0.234515	1.051954	-0.159078	0.212339
921	1.444119	0.579408	-1.190191	-1.095472	0.221196	0.856447	-0.238278	0.766821	-0.6865730	0.799168	-0.660428	0.810526	-0.525227	1.04630

922 rows x 14 columns

```
In [79]: mask = np.triu(X1.corr())
f, ax = plt.subplots(figsize=(11, 9))
sns.heatmap(X1.corr(), mask=mask, annot=True, square=True, cmap='Greens_r')
plt.xticks(rotation=45, ha='right')
plt.show()
```



```
In [80]: mask = np.triu(X2.corr())
f, ax = plt.subplots(figsize=(11, 9))
sns.heatmap(X2.corr(), mask=mask, annot=True, square=True, cmap='Greens_r')
plt.xticks(rotation=45, ha='right')
plt.show()
```



```
In [81]: df_norm_n.describe()
```

```
Out[81]:
```

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки	Шаг нашивки	Плотность нашивки
count	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	
mean	0.136481	0.000874	0.589395	0.217380	0.033223	0.006630	0.085460	0.143065	0.021899	0.721977	0.065062	0.000151	0.002074	0.017175
std	0.079188	0.000282	0.062762	0.094759	0.008830	0.000994	0.014756	0.082898	0.002416	0.069619	0.018230	0.000150	0.000797	0.003727
min	0.000000	0.000162	0.443526	0.000703	0.011338	0.004102	0.049258	0.000230	0.016004	0.463108	0.021501	0.000000	0.000011	0.007152
25%	0.067487	0.000676	0.544650	0.149169	0.027253	0.005917	0.075299	0.079085	0.020174	0.680318	0.051933	0.000000	0.001527	0.014627
50%	0.134841	0.000860	0.584696	0.217972	0.030305	0.006579	0.084087	0.138037	0.021709	0.730982	0.064760	0.000239	0.002040	0.017102
75%	0.203894	0.001053	0.630216	0.285099	0.039153	0.007259	0.095598	0.204477	0.023481	0.773008	0.077307	0.000294	0.002580	0.019520
max	0.321947	0.001803	0.802854	0.525070	0.062859	0.010626	0.144122	0.414125	0.030444	0.875356	0.122368	0.000409	0.004517	0.030187

```
In [82]: df_minmax_n.describe()
```

```
Out[82]:
```

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки	Шаг нашивки	Плотность нашивки
count	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000	922.000000
mean	0.500000	0.499412	0.502904	0.451341	0.506200	0.490578	0.516739	0.487343	0.503776	0.510846	0.503426	0.503938	0.503938	0.503938
std	0.289145	0.187858	0.186835	0.201534	0.186806	0.180548	0.190721	0.217269	0.196366	0.199418	0.501054	0.183567	0.193933	0.193933
min	0.000000	0.371909	0.368184	0.305188	0.378514	0.366571	0.386228	0.204335	0.353512	0.373447	0.000000	0.000000	0.000000	0.000000
25%	0.500000	0.495189	0.511396	0.451377	0.506382	0.488852	0.516931	0.354161	0.483718	0.501043	1.000000	0.506414	0.504310	0.504310
50%	0.750000	0.629774	0.624719	0.587193	0.638735	0.623046	0.646553	0.538397	0.617568	0.642511	1.000000	0.626112	0.630842	0.630842
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Разработка и обучение нескольких моделей для прогноза прочности при растяжении

***даные в нашем итоговом датасете в основном непрерывные и на них приходит сразу решение данной задачи с использованием регрессионных моделей.
Но наличие графиков рассеивания точек и тепловых карт не дают нам реальной взаимосвязи и возможности прямого прогнозирования, поэтому будем использовать и категориальные подходы к прогнозированию (например, попробуем метод ближайших соседей)

или обучение со скрытыми слоями, чтобы выявить дополнительные взаимосвязи.

Задача ВКР на этом этапе звучит так: "При построении моделей провести поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10."

Сформируем выборку и посмотрим, что получилось

```
In [84]: # Проверяем прочность при растяжении
```

После всех подготовительных работ переходим к процессу создания, обучения моделей. Мы будем использовать в Python библиотеку Scikit-Learn. В качестве базового уровня предсажем медианное значение цели на обучающем наборе для всех примеров в тестовом наборе. В качестве метрики возьмем среднюю абсолютную ошибку (mae) в прогнозах. Для обучения используем 70 % данных, а для тестиования — 30 %.

```
In [85]: #разделяем на тестовую, тренировочную выборки, выделяем предикторы и целевые переменные
x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(
    df_norm_n.loc[:, df_norm_n.columns != 'Прочность при растяжении', 'Mpa'],
    df['Прочность при растяжении', 'Mpa'],
    test_size = 0.3,
    random_state = 42)
```

```
In [86]: #проверка правильности разбиения
df_norm_n.shape[8] - x_train_1.shape[8]
```

```
0
```

```
Out[86]:
```

x_train_1.head()

Out[87]:

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м ³	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %	Температура вспышки, С	2	Поверхностная плотность, г/м ²	Модуль упругости при растяжении, ГПа	Потребление смолы, г/м ²	Угол нашивки	Шаг нашивки	Плотность нашивки
481	0.1384462	0.000523	0.583394	0.130876	0.020795	0.006194	0.0762	0.184291	0.020024	0.072334	0.000288	0.002263	0.015413	
650	0.196184	0.000422	0.605304	0.227991	0.026722	0.007203	0.074110	0.079766	0.024554	0.055735	0.000302	0.001324	0.017954	
483	0.14097	0.000574	0.579585	0.168815	0.042968	0.007003	0.095586	0.131480	0.020985	0.041073	0.000292	0.002227	0.018740	
355	0.095834	0.000907	0.543384	0.304721	0.024357	0.006580	0.070163	0.159182	0.018842	0.049917	0.000000	0.001766	0.012482	
850	0.264153	0.000553	0.574561	0.254615	0.032952	0.007252	0.086400	0.092201	0.022685	0.083270	0.000311	0.002667	0.016569	

In [88]:

```
y_train_1
```

Out[88]:

Прочность при растяжении, МПа

481	2641.571967
650	2404.689821
483	2619.854215
355	2793.878901
850	298.985700
...	...
106	1994.674603
270	2419.732206
860	2758.414767
435	2347.35204
102	1529.604423

645 rows × 1 columns

In [89]:

```
y_test_1
```

Out[89]:

Прочность при растяжении, МПа

319	2167.533030
377	2705.619718
538	2952.839631
296	2305.241225
531	1399.10555
...	...
420	3305.286922
133	2085.866383
490	2461.609016
558	2616.114331
363	2478.484767

277 rows × 1 columns

In [90]:

```
y_train_1.shape
```

Out[90]:

```
(645, 1)
```

In [91]:

```
#функция для сравнения результатом предсказаний с моделью, выдающей среднее значение по тестовой выборке
def mean_mean(y_test_1):
    return np.mean(y_test_1) for _ in range(len(y_test_1))
y_1_pred_mean = mean_mean(y_test_1)
```

In [92]:

```
#Проверка различных моделей при стандартных параметрах
# Метод опорных векторов - 1
```

In [93]:

```
svr = make_pipeline(StandardScaler(), SVR(kernel = 'rbf', C = 500.0, epsilon = 1.0))
#подключаем модуль
svr.fit(x_train_1, np.ravel(y_train_1))
#выводим коэффициент детерминации
y_pred_svr = svr.predict(x_test_1)
mse_svr = mean_squared_error(y_pred_svr, y_test_1)
mse_svr_elast = mean_squared_error(y_test_1,y_pred_svr)
print("Support Vector Regression Results Train:")
print("Test score: {:.2f} (svr.score(x_train_1, y_train_1)) # Скор для тренировочной выборки")
print("Support Vector Regression Results Test:")
print("SVR_MAE: {:.2f} (mean_absolute_error(y_test_1, y_pred_svr))")
print("SVR_MPE: {:.2f} (format(mean_absolute_percentage_error(y_test_1, y_pred_svr)))")
print("SVR_RMSE: {:.2f} (format(np.sqrt(mse_svr_elast)))")
print("SVR_RMSPE: {:.2f} (format(np.sqrt(mse_svr_elast)))")
print("SVR_MSE: {:.2f} (format(sv.scorer(x_test_1, y_test_1))) # Скор для тестовой выборки")
Support Vector Regression Results Train:
Test score: 0.89
Support Vector Regression Results:
SVR_MAE: 76
SVR_MPE: 0.83
SVR_RMSE: 11099.84
SVR_RMSPE: 105.36
Test score: 0.95

In [94]:


```
#Подключаем модуль, выдающий среднее значение
mse_lin_elast = mean_squared_error(y_test_1, y_1_pred_mean)
print("MAE for mean target: ", mean_absolute_error(y_test_1, y_1_pred_mean))
print("MSE for mean target: ", mse_lin_elast)
print("RMSE for mean target: ", np.sqrt(mse_lin_elast))
MAE for mean target: 368.968081863398946
MSE for mean target: 214046.375545478
RMSE for mean target: 462.6514627536157
```



In [95]:



```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Support Vector Regression")
plt.plot(y_pred_svr, label = "Прогноз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "green")
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);
```



Tестовые и прогнозные значения Support Vector Regression



Прогноз



Тест



Количество наблюдений



Прочность при растяжении, МПа



In [96]:



```
Метод случайного леса - Random Forest Regressor - 2
```



In [97]:



```
rfr = RandomForestRegressor(n_estimators=15,max_depth=7, random_state=33)
rfr.fit(x_train_1, np.ravel(y_train_1))
y_pred_rfr = rfr.predict(x_test_1)
mae_rfr = mean_absolute_error(y_pred_rfr, y_test_1)
mse_rfr_elast = mean_squared_error(y_test_1,y_pred_rfr)
print("Support Vector Regression Results Train:")
print("Test score: {:.2f} (rfr.score(x_train_1, y_train_1)) # Скор для тренировочной выборки")
print("Random Forest Regressor Results:")
print("RF_MAE: {:.2f} (round(mean_absolute_error(y_test_1, y_pred_rfr)))")
print("RF_MPE: {:.2f} (format(mean_absolute_percentage_error(y_test_1, y_pred_rfr)))")
print("RF_RMSE: {:.2f} (format(np.sqrt(mse_rfr_elast)))")
print("RF_RMSPE: {:.2f} (format(rfr.score(x_test_1, y_test_1))) # Скор для тестовой выборки")
Random Forest Regressor Results Train:
Test score: 0.98
Random Forest Regressor Results:
RF_MAE: 0.03
RF_MPE: 0.03
RF_RMSE: 9651.67
RF_RMSPE: 98.24
Test score: 0.95
```



In [98]:



```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Random Forest Regressor")
plt.plot(y_pred_rfr, label = "Прогноз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);
```



Прогноз



Тест

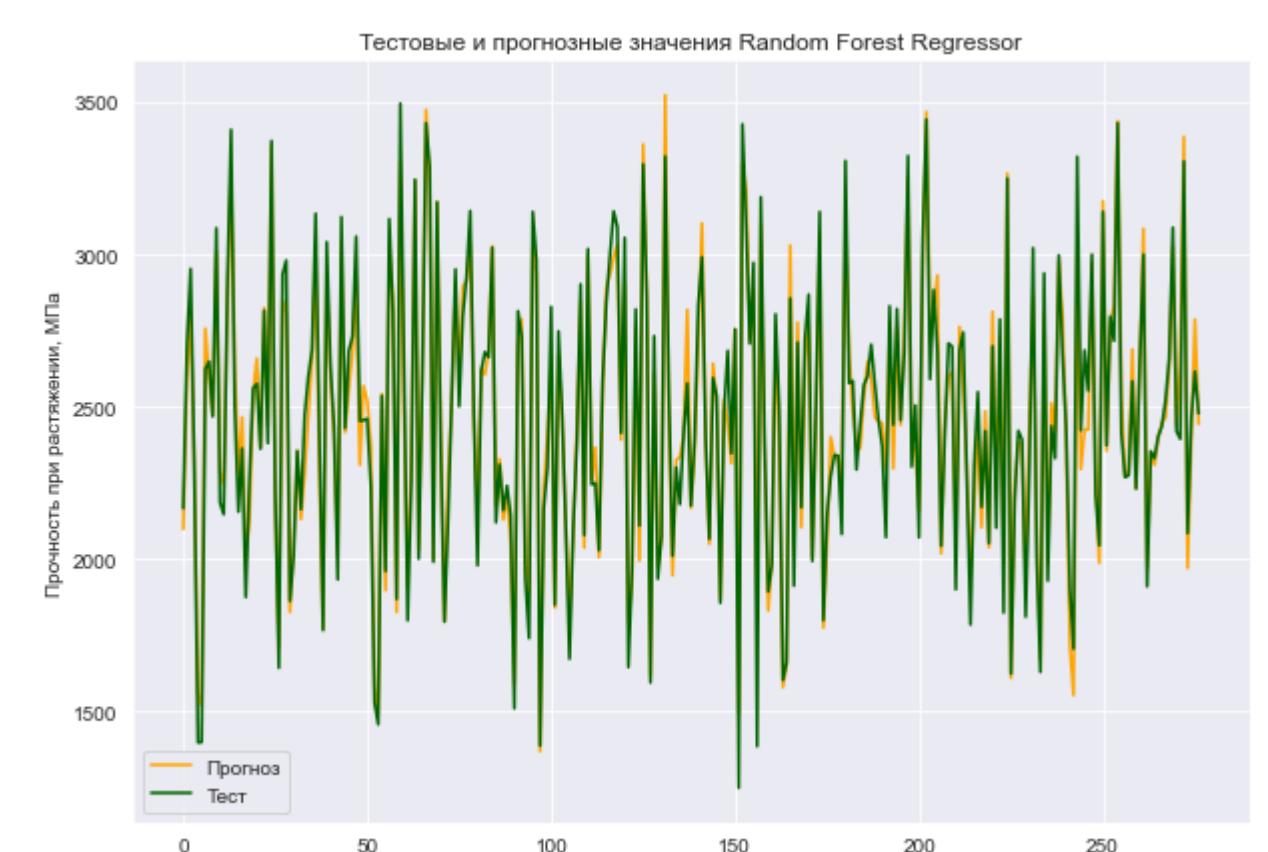


Количество наблюдений



Прочность при растяжении, МПа


```



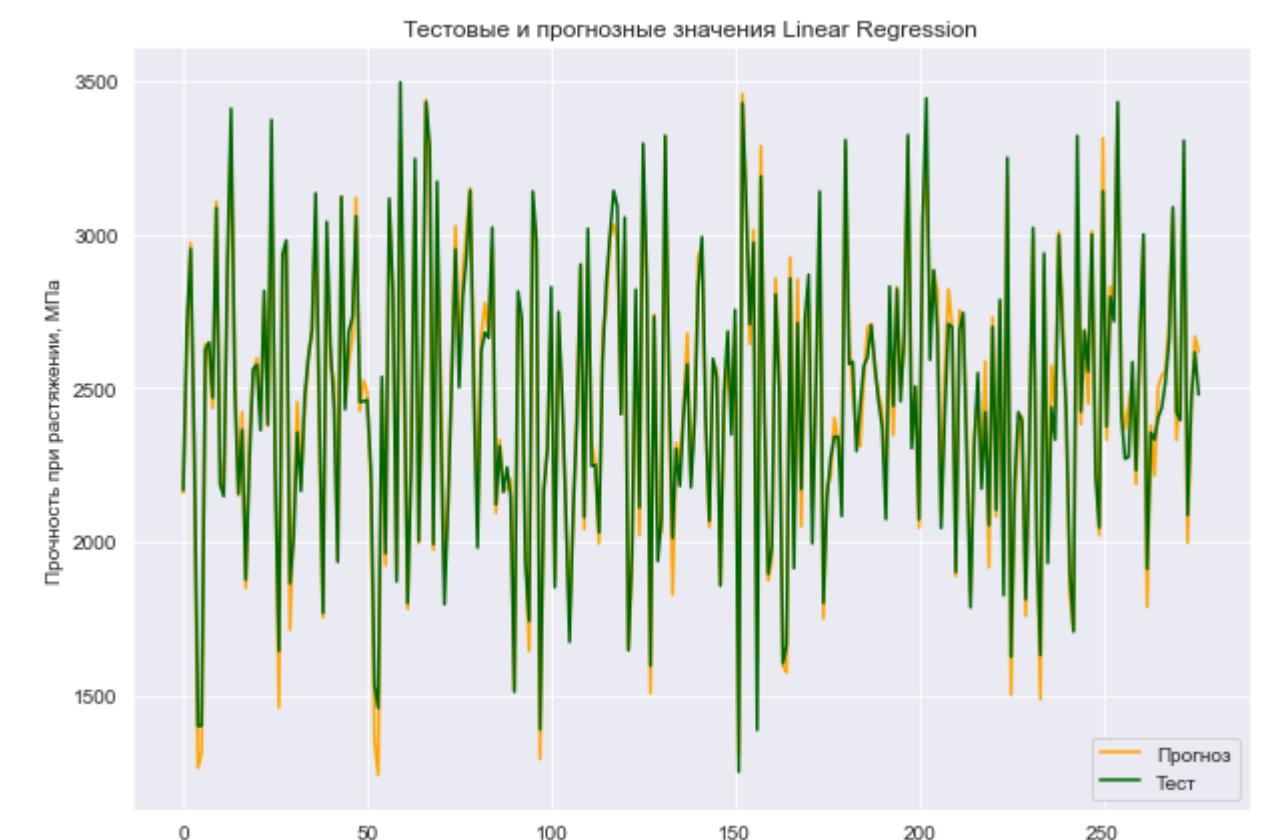
In [99]: #Метод линейной регрессии - Linear Regression - 3

```
In [100]: #построение модели и визуализация линейной регрессии
lr = LinearRegression()
lr.fit(x_train_1, y_train_1)
y_pred_lr = lr.predict(x_test_1)
mae_lr = mean_absolute_error(y_pred_lr, y_test_1)
mse_lr_elast = mean_squared_error(y_test_1, y_pred_lr)
print("Linear Regression Results Train:") # Скор для тренировочной выборки
print("Test score: {:.2f}".format(lr.score(x_train_1, y_train_1)))
print("Linear Regression Results:")
print("lr_MAE: ", round(mae_lr))
print("lr_MAPE: {:.2f}%".format((mae_lr * 100) / np.mean(y_test_1, axis=0)))
print("lr_RMSE: {:.2f} ".format(np.sqrt(mse_lr_elast)))
print("Test score: {:.2f} ".format(lr.score(x_test_1, y_test_1))) # Скор для тестовой выборки
```

```
Linear Regression Results Train:
Test score: 0.97
Linear Regression Results:
lr_MAE: 0
lr_MAPE: 0.02
lr_RMSE: 6382.76
lr_RMSE: 79.89
Test score: 0.97
```

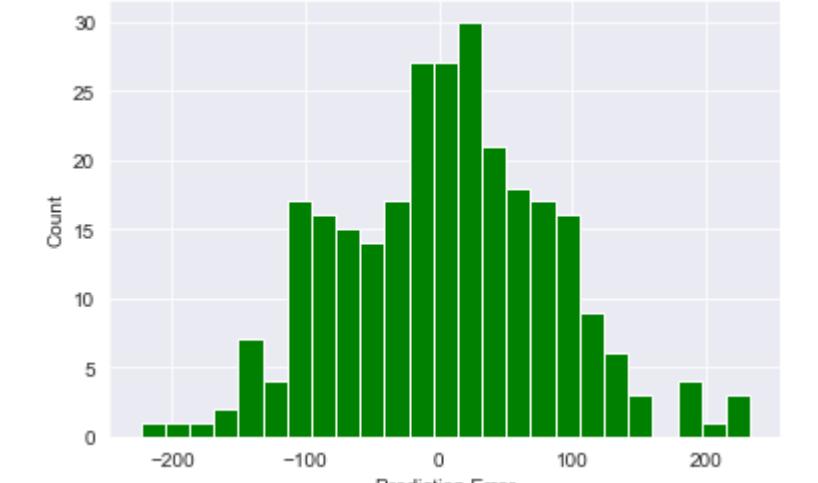
```
In [101]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Linear Regression")
plt.plot(y_test_1, color = "orange")
plt.plot(y_pred_lr, label = "тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Проницаемость при растяжении, МПа")
plt.legend()
plt.grid(True);

#Линейная регрессия с задачей справилась в 97 % случаев.
```



In [102]: #Визуализация гистограммы распределения ошибки

```
In [103]: error = y_test_1 - y_pred_lr
plt.hist(error, bins = 25, color = "g")
plt.xlabel("Prediction Error")
_ = plt.ylabel("Count")
```



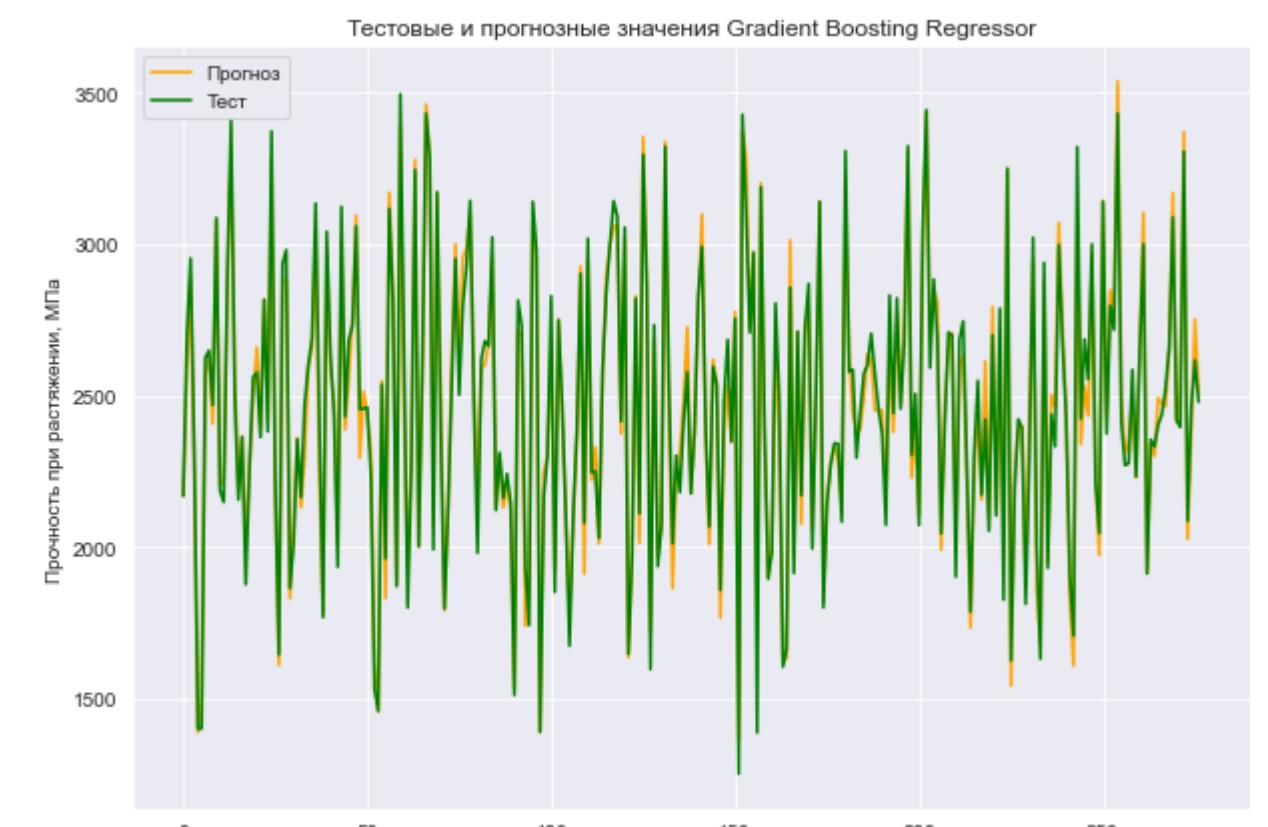
In [104]: #Метод градиентного бустинга - Gradient Boosting Regressor - 4

```
In [105]: gbr = make_pipeline(StandardScaler(), GradientBoostingRegressor())
gbr.fit(x_train_1, np.ravel(y_train_1))
y_pred_gbr = gbr.predict(x_test_1)
mae_gbr = mean_absolute_error(y_pred_gbr, y_test_1)
mse_gbr_elast = mean_squared_error(y_test_1, y_pred_gbr)
print("Gradient Boosting Regressor Results Train:")
print("Test score: {:.2f} ".format(gbr.score(x_train_1, y_train_1))) # Скор для тренировочной выборки
print("Gradient Boosting Regressor Results:")
print("gbr_MAE: {:.2f} ".format(mae_gbr))
print("gbr_MAPE: {:.2f}%".format((mae_gbr * 100) / np.mean(y_test_1, axis=0)))
print("gbr_RMSE: {:.2f} ".format(np.sqrt(mse_gbr_elast)))
print("gbr_RMSE: {:.2f} ".format(np.sqrt(mse_gbr_elast)))
print("Test score: {:.2f} ".format(gbr.score(x_test_1, y_test_1))) # Скор для тестовой выборки
```

```
Gradient Boosting Regressor Results Train:
Test score: 0.99
Gradient Boosting Regressor Results:
gbr_MAE: 61
gbr_MAPE: 0.03
gbr_RMSE: 6261.35
gbr_RMSE: 79.13
Test score: 0.97
```

```
In [106]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Gradient Boosting Regressor")
plt.plot(y_test_1, color = "orange")
plt.plot(y_pred_gbr, label = "тест", color = "green")
plt.xlabel("Количество наблюдений")
plt.ylabel("Проницаемость при растяжении, МПа")
plt.legend()
plt.grid(True);

#Градиентный бустинг с задачей справился в 97 % случаев.
```



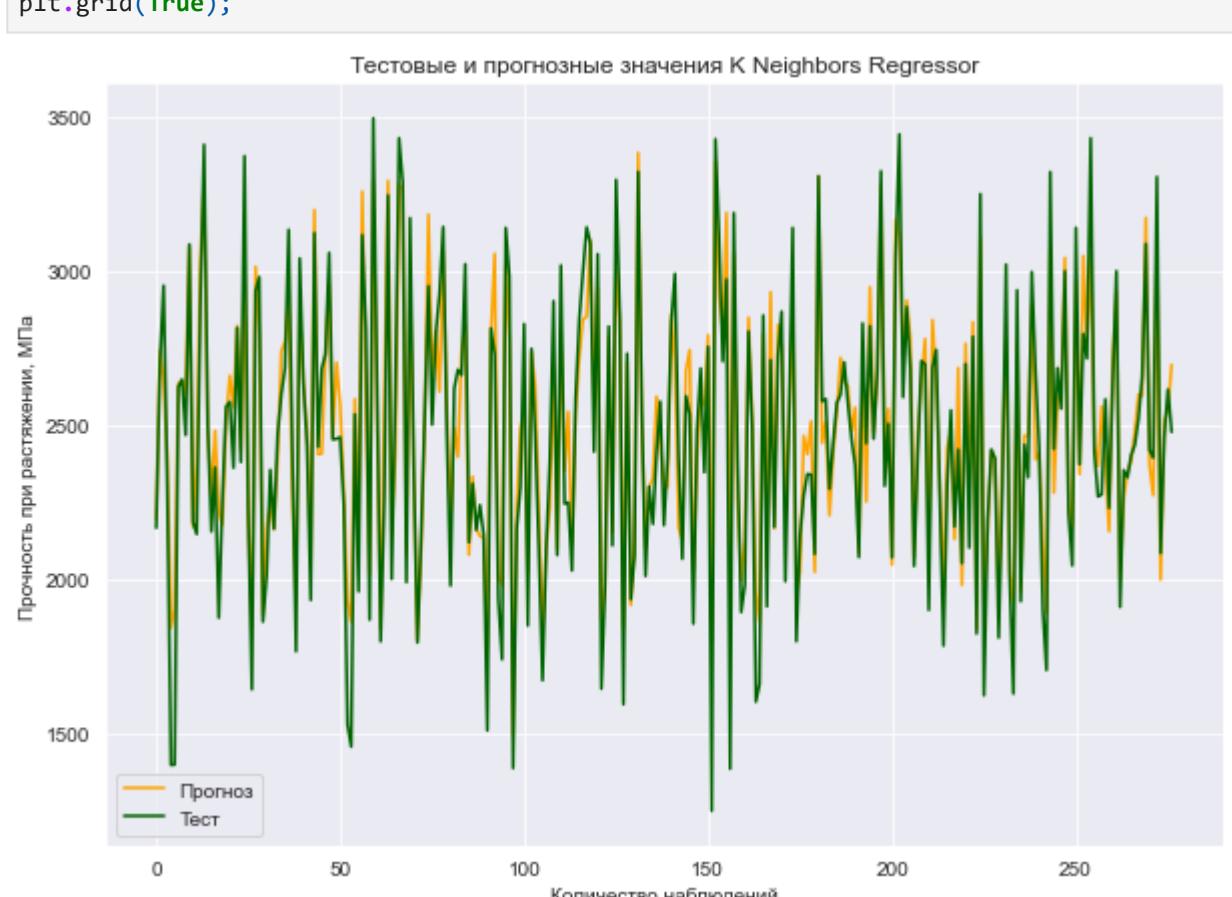
In [107]: # Метод K ближайших соседей - K Neighbors Regressor - 5

```
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train_1, y_train_1)
y_pred_knn = knn.predict(x_test_1)
mae_knn = mean_absolute_error(y_pred_knn, y_test_1)
mse_knn_elast = mean_squared_error(y_test_1, y_pred_knn)
print("K Neighbors Regressor Results Train:")
print("Test score: {:.2f} ".format(knn.score(x_train_1, y_train_1))) # Скор для тренировочной выборки
print("K Neighbors Regressor Results:")
print("KNN_MAE: ", round(mean_absolute_error(y_test_1, y_pred_knn)))
```

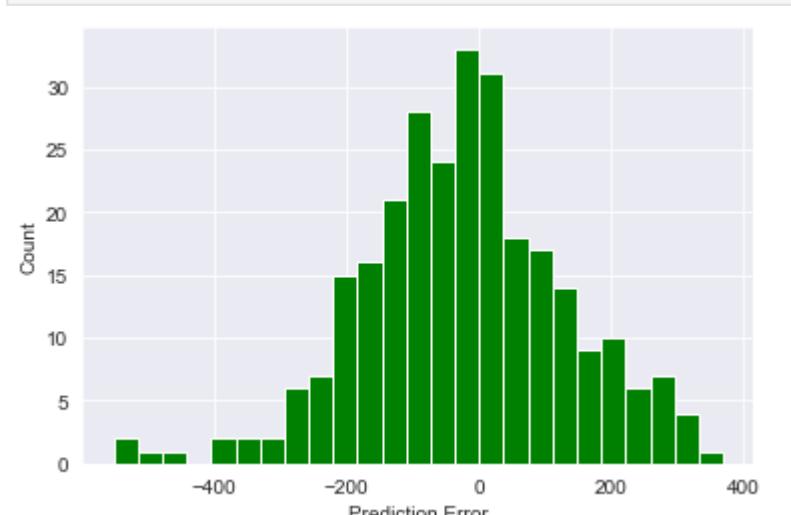
```
print("KNN_MAPE: {:.2f}".format(mean_absolute_percentage_error(y_test_1, y_pred_knn)))
print("KNN_MSE: {:.2f}".format(msse_knn_elast))
print("KNN_RMSE: {:.2f}".format(np.sqrt(msse_knn_elast)))# Скор для тестовой выборки
print("Test score: {:.2f}".format(knn.score(x_test_1, y_test_1)))
```

```
K Neighbors Regressor Results Train:
Test score: 0.92
K Neighbors Regressor Results:
KNN_MAE: 0.05
KNN_MSE: 25243.37
KNN_RMSE: 156.88
Test score: 0.88
```

```
In [108]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения K Neighbors Regressor")
plt.plot(y_pred_knn, label = "Пропоз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);
```



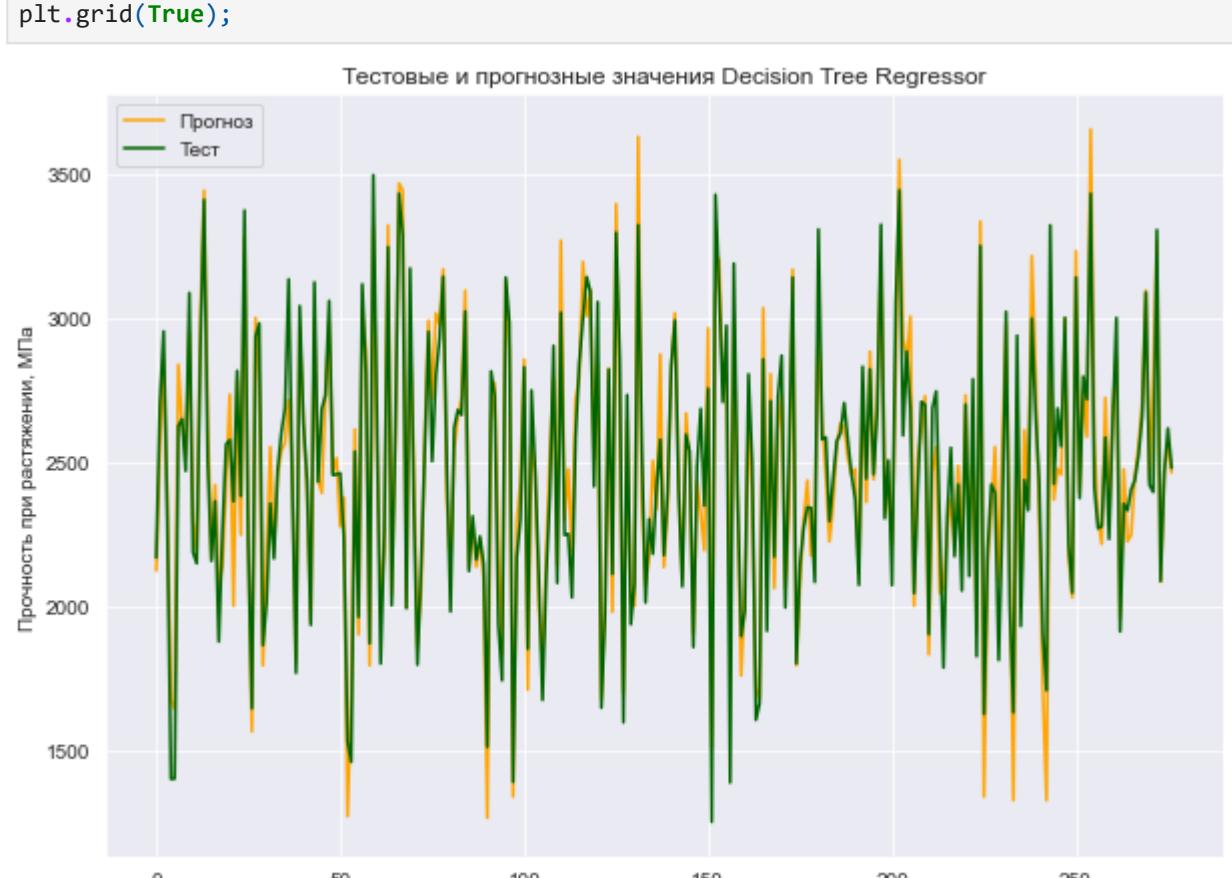
```
In [109]: #Визуализация гистограммы распределения ошибки
error = y_test_1 - y_pred_knn
plt.hist(error, bins = 25, color = "g")
plt.xlabel("Prediction Error")
plt.ylabel("Count")
```



```
In [110]: #Деревья решений - Decision Tree Regression - 6
dtr = DecisionTreeRegressor()
dtr.fit(x_train_1, y_train_1.values)
y_pred_dtr = dtr.predict(x_test_1)
mae_dtr = mean_absolute_error(y_pred_dtr, y_test_1)
mse_dtr_elast = mean_squared_error(y_test_1, y_pred_dtr)
print("Decision Tree Regressor Results Train:")
print("Test score: {:.2f}".format(knn.score(x_train_1, y_train_1)))# Скор для тренировочной выборки
print("Decision Tree Regressor Results:")
print("DTR_MAE: {:.2f}".format(mae_dtr))
print("DTR_MSE: {:.2f}".format(mse_dtr_elast))
print("DTR_RMSE: {:.2f}".format(np.sqrt(mse_dtr_elast)))
print("DTR_MAPE: {:.2f}".format(mean_absolute_percentage_error(y_test_1, y_pred_dtr)))
print("Test score: {:.2f}".format(dtr.score(x_test_1, y_test_1)))# Скор для тестовой выборки
```

```
Decision Tree Regressor Results Train:
Test score: 0.92
Decision Tree Regressor Results:
DTR_MAE: 115
DTR_MSE: 21371.25
DTR_RMSE: 146.19
DTR_MAPE: 0.00
Test score: 0.98
```

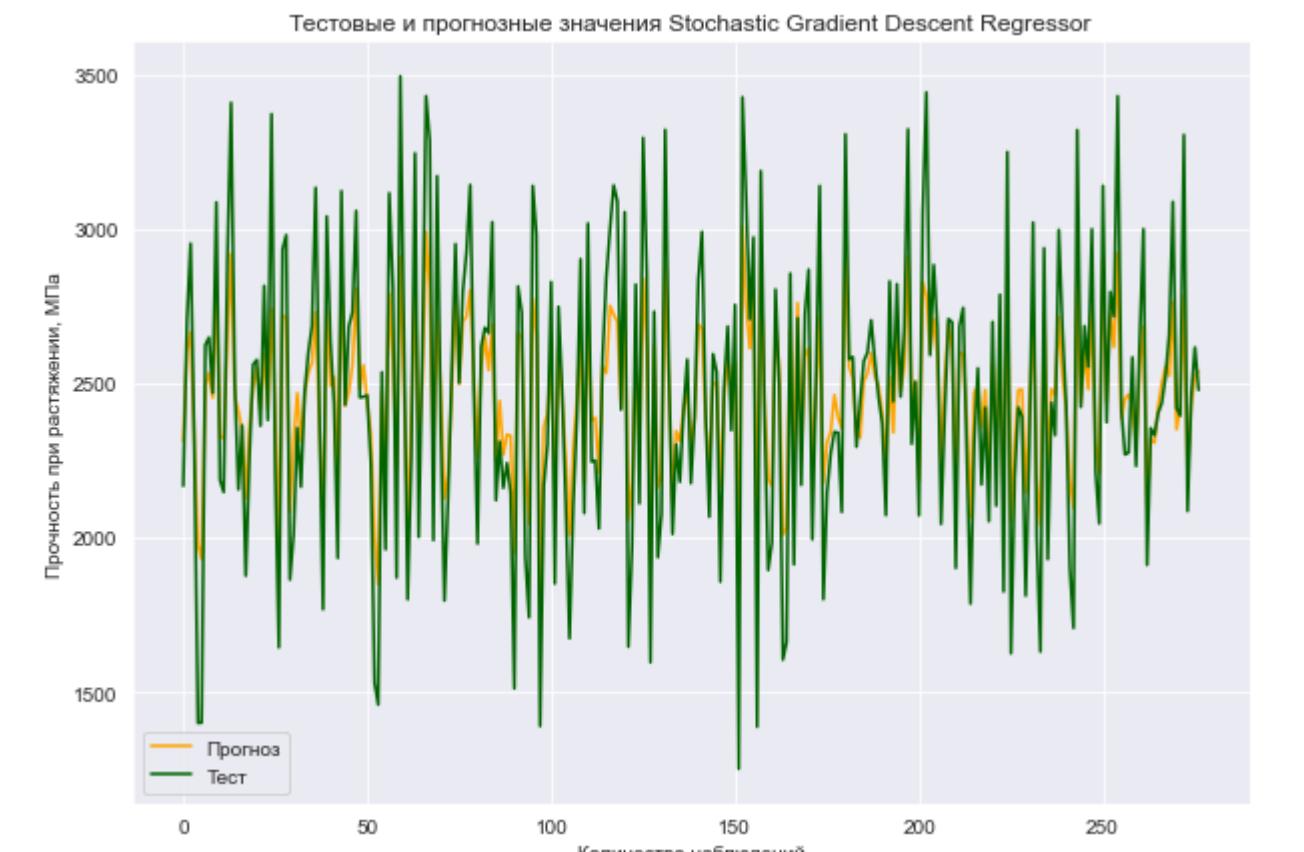
```
In [111]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Decision Tree Regression")
plt.plot(y_pred_dtr, label = "Пропоз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);
```



```
In [112]: # Стохастический градиентный спуск (SGD) - Stochastic Gradient Descent Regressor - 7
sgd = SGDRegressor()
sgd.fit(x_train_1, y_train_1)
y_pred_sgd = sgd.predict(x_test_1)
mae_sgd = mean_absolute_error(y_pred_sgd, y_test_1)
mse_sgd_elast = mean_squared_error(y_test_1, y_pred_sgd)
print("Stochastic Gradient Descent Regressor Results Train:")
print("Test score: {:.2f}".format(sgd.score(x_train_1, y_train_1)))# Скор для тренировочной выборки
print("Stochastic Gradient Descent Regressor Results:")
print("SGD_MAE: {:.2f}".format(mae_sgd))
print("SGD_MSE: {:.2f}".format(mse_sgd_elast))
print("SGD_RMSE: {:.2f}".format(np.sqrt(mse_sgd_elast)))
print("SGD_MAPE: {:.2f}".format(mean_absolute_percentage_error(y_test_1, y_pred_sgd)))
print("Test score: {:.2f}".format(sgd.score(x_test_1, y_test_1)))# Скор для тестовой выборки
```

```
Stochastic Gradient Descent Regressor Results Train:
Test score: 0.72
Stochastic Gradient Descent Regressor Results:
SGD_MAE: 189
SGD_MSE: 56271.38
SGD_RMSE: 237.22
SGD_MAPE: 0.08
Test score: 0.74
```

```
In [113]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Stochastic Gradient Descent Regressor")
plt.plot(y_pred_sgd, label = "Пропоз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);
```



```
In [114]: # Многослойный перцептрон - Multi-layer Perceptron regressor - 8
mlp = MLPRegressor(max_iter = 1, max_iter = 500)
mlp.fit(x_train_1, y_train_1)
y_pred_mlp = mlp.predict(x_test_1)
mae_mlp = mean_absolute_error(y_pred_mlp, y_test_1)
mse_mlp_elast = mean_squared_error(y_pred_mlp, y_test_1)
print("Multi-layer Perceptron regressor Results Train:")
print("Test score: {:.2f}".format(mlp.score(x_train_1, y_train_1)))# Скор для тренировочной выборки
print("SGD_MAE: ", round(mean_absolute_error(y_test_1, y_pred_mlp)))
print("SGD_MAPE: {:.2f}%".format(mean_absolute_percentage_error(y_test_1, y_pred_mlp)))
print("SGD_MSE: {:.2f} ".format(mse_mlp_elast))
print("SGD_RMSE: {:.2f} ".format(np.sqrt(mse_mlp_elast)))
print("SGD_RMSLE: {:.2f} ".format(np.sqrt(mlp.score(x_test_1, y_test_1))))# Скор для тестовой выборки
print("Test score: {:.2f} ".format(mlp.score(x_test_1, y_test_1)))
print("Test score: 12.46")

Multi-layer Perceptron regressor Results Train:
Test score: 13.11
Multi-layer Perceptron regressor Results:
SGD_MAE: 0.64
SGD_MAPE: 0.64
SGD_MSE: 2880353.81
SGD_RMSE: 1697.16
Test score: 12.46
```

```
In [115]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Multi-layer Perceptron regressor")
plt.plot(y_pred_mlp, label = "Прогноз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);


```

```
In [116]: # Лассо регрессия - the Lasso - 9
clf = linear_model.Lasso(alpha=0.1)
clf.fit(x_train_1, y_train_1)
y_pred_clf = clf.predict(x_test_1)
mae_clf = mean_absolute_error(y_pred_clf, y_test_1)
mse_clf_elast = mean_squared_error(y_test_1,y_pred_clf)
print("Lasso regression Results Train:")
print("Test score: {:.2f}".format(clf.score(x_train_1, y_train_1)))# Скор для тренировочной выборки
print("Lasso regression Results:")
print("SGD_MAE: ", round(mean_absolute_error(y_test_1, y_pred_clf)))
print("SGD_MAPE: {:.2f}%".format(mean_absolute_percentage_error(y_test_1, y_pred_clf)))
print("SGD_MSE: {:.2f} ".format(mse_clf_elast))
print("SGD_RMSE: {:.2f} ".format(np.sqrt(mse_clf_elast)))
print("SGD_RMSLE: {:.2f} ".format(np.sqrt(clf.score(x_test_1, y_test_1))))# Скор для тестовой выборки
print("Test score: {:.2f} ".format(clf.score(x_test_1, y_test_1)))
print("Test score: 0.96")

Lasso regression Results Train:
Test score: 0.95
Lasso regression Results:
SGD_MAE: 70
SGD_MAPE: 93
SGD_MSE: 1993.22
SGD_RMSE: 89.40
Test score: 0.96
```

```
In [117]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Lasso regressor")
plt.plot(y_pred_clf, label = "Прогноз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);


```

```
In [118]: # группировка наших моделей по метрике MAE
mae_df = {'Perceptron': ['Support Vector', 'RandomForest', 'Linear Regression', 'GradientBoosting', 'KNeighbors', 'DecisionTree', 'SGD', 'MLP', 'Lasso'], 'MAE': [mae_svr, mae_rfr, mae_lr, mae_gbr, mae_knr, mae_dtr, mae_sdg, mae_mlp, mae_clf]}

mae_df = pd.DataFrame(mae_df)

In [119]: mae_df
```

Перецессор	MAE
0 Support Vector	76.111030
1 RandomForest	76.531400
2 Linear Regression	63.456419
3 GradientBoosting	61.110269
4 KNeighbors	122.059974
5 DecisionTree	113.159004
6 SGD	189.206480
7 MLP	1619.254573
8 Lasso	70.176750

```
In [120]: # поиск оптимальных гиперпараметров.
# В машинном обучении гиперпараметры называются параметрами алгоритмов, значения которых устанавливаются перед запуском процесса обучения.
# В этом смысле они отличаются от обычных параметров, вычисляемых в процессе обучения. Гиперпараметры используются для управления процессом обучения.
# Одним из способов настройки гиперпараметров состоит в том, чтобы засадить компьютер пробовать все возможные комбинации значений параметров.
# Для этого используются модуль GridSearchCV из библиотеки Scikit Learn. Попытайся найти наилучшую комбинацию гиперпараметров для построения классификатора для нашего набора данных.
# Использование GridSearchCV:
# Примечание: Этот метод требует очень много времени. Если диапазон или число гиперпараметров велики, то время может исчисляться миллиами, и на завершение потребуется довольно много времени.

In [121]: # Проблема поиска по сетке гиперпараметров с перекрестной проверкой, количество блоков равно 10 (cv = 10), для
# модели случайного леса - Random Forest Regressor - 2
parameters = { 'n_estimators': [200, 300],
```

```

'max_depth': [9, 15],
'max_features': ['auto'],
'criterion': ['mse'])
grid = GridSearchCV(estimator = rfr, param_grid = parameters, cv = 10)
grid.fit(x_train_1, y_train_1)

Out[121]: GridSearchCV(cv=10,
                     estimator=RandomForestRegressor(max_depth=7, n_estimators=15,
                                                     max_features='auto',
                                                     min_samples_leaf=2),
                     param_grid=[{'criterion': 'mse', 'max_depth': 9, 'max_features': 'auto', 'n_estimators': 300}])

In [122]: grid.best_params_
Out[122]: {'criterion': 'mse',
           'max_depth': 15,
           'max_features': 'auto',
           'n_estimators': 300}

In [123]: # подбираем оптимальные параметры для оптимальной модели
gs1 = GridSearchCV(gs1, param_grid, cv=10)
gs1.fit(x_train_1, y_train_1)
print("R2-score RFR для прочности при растижении, MAE: %s" % (gs1.score(x_test_1, y_test_1).round(3)))
RandomForestRegressor(criterion='mse', max_depth=15, n_estimators=300, random_state=33)
R2-score RFR для прочности при растижении, MAE: 0.962

In [124]: #подбираем оптимальные гиперпараметры в нову модель случайного леса
rfr_grid = RandomForestRegressor(n_estimators = 200, criterion = "mse", max_depth = 15, max_features = 'auto')
#обучаем модель
rfr_grid.fit(x_train_1, y_train_1)
predictions_rfr_grid = rfr_grid.predict(x_test_1)
#оцениваем точность на тестовом наборе
mae_rfr_grid = mean_absolute_error(predictions_rfr_grid, y_test_1)
mae_rfr_grid

Out[124]: 72.08635818902486

In [125]: new_row_in_mae_df = {'Perceptron': 'RandomForest_GridSearchCV', 'MAE': mae_rfr_grid}
mae_df = mae_df.append(new_row_in_mae_df, ignore_index=True)

In [126]: # поиск гиперпараметров не дал улучшений для уже имеющейся модели RandomForestRegressor
# Вероятно, нужно указывать больше барийшай параметров при работе с GridSearchCV
mae_df

Out[126]: Perceptron MAE
0 Support Vector 76.111030
1 RandomForest 76.531400
2 Linear Regression 63.456419
3 GradientBoosting 61.110269
4 KNeighbors 122.095974
5 DecisionTree 113.159004
6 SGD 189.206680
7 MLP 1619.254573
8 Lasso 70.176750
9 RandomForest_GridSearchCV 72.086358
9 KNeighbors_GridSearchCV 72.086358

In [127]: # Проделем поиск по семику гиперпараметров с перекрестной проверкой, количеством блоков равно 10 (cv = 10), для
# Метода К ближайших соседей - K Neighbors Regressor - 5
knn = KNeighborsRegressor()
knn_params = {'n_neighbors': range(1, 301),
              'weights': ['uniform', 'distance'],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']}
#Запускаем обучение модели. В начале оценки модели будем использовать коэффициент детерминации (R^2)
# Если R2<0, это значит, что разработанная модель дает прогноз даже хуже, чем простое усреднение.
gs = GridSearchCV(knn, knn_params, cv = 10, verbose = 1, n_jobs = -1, scoring = 'r2')
gs.fit(x_train_1, y_train_1)
knn = gs.best_estimator_
gs.best_params_

Fitting 10 folds for each of 1200 candidates, totaling 12000 fits
{'algorithm': 'brute', 'n_neighbors': 5, 'weights': 'distance'}

Out[127]: {'algorithm': 'brute', 'n_neighbors': 5, 'weights': 'distance'}

In [128]: # подбираем оптимальные гиперпараметры для оптимальной модели
print(gs.best_estimator_)
gs1 = gs.best_estimator_
print("R2-score KNN для прочности при растижении, MAE: %s" % (gs1.score(x_test_1, y_test_1).round(3)))
KNeighborsRegressor(algorithm='brute', weights='distance')
R2-score KNN для прочности при растижении, MAE: 0.888

In [129]: #подбираем оптимальные гиперпараметры в нову модель метода к ближайших соседей
knn_grid = KNeighborsRegressor(algorithm = 'brute', n_neighbors = 7, weights = 'distance')
#обучаем модель
knn_grid.fit(x_train_1, y_train_1)
predictions_knn_grid = knn_grid.predict(x_test_1)
#оцениваем точность на тестовом наборе
mae_knn_grid = mean_absolute_error(predictions_knn_grid, y_test_1)
mae_knn_grid

Out[129]: 110.7337738808438

In [130]: new_row_in_mae_df = {'Perceptron': 'KNeighbors_GridSearchCV', 'MAE': mae_knn_grid}
mae_df = mae_df.append(new_row_in_mae_df, ignore_index=True)
mae_df

Out[130]: Perceptron MAE
0 Support Vector 76.111030
1 RandomForest 76.531400
2 Linear Regression 63.456419
3 GradientBoosting 61.110269
4 KNeighbors 122.095974
5 DecisionTree 113.159004
6 SGD 189.206680
7 MLP 1619.254573
8 Lasso 70.176750
9 RandomForest_GridSearchCV 72.086358
10 KNeighbors_GridSearchCV 110.733774
10 KNeighbors_GridSearchCV 110.733774

In [131]: # Проделем поиск по семику гиперпараметров с перекрестной проверкой, количеством блоков равно 10 (cv = 10), для
# Метода Деревьев Решений - Decision Tree Regressor - 6
criterion = ['squared_error', 'friedman_mse', 'absolute_error', 'poisson']
splitter = ['best', 'random']
min_samples_split = range(2, 150, 200)
min_samples_leaf = range(2, 250, 200)
max_features = ['auto', 'sqrt', 'log2']
parameters = {'criterion': criterion,
              'splitter': splitter,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf,
              'max_features': max_features}
#Запускаем обучение модели. В начале оценки модели будем использовать коэффициент детерминации (R^2)
# Если R2<0, это значит, что разработанная модель дает прогноз даже хуже, чем простое усреднение.
gs4 = GridSearchCV(dtr, parameters, cv = 10, verbose = 1, n_jobs = -1, scoring = 'r2')
gs4.fit(x_train_1, y_train_1)
dtr_3 = gs4.best_estimator_
gs4.best_params_

Fitting 10 folds for each of 1088 candidates, totaling 10880 fits
{'algorithm': 'brute', 'n_neighbors': 5, 'weights': 'distance'}

Out[131]: {'algorithm': 'brute', 'n_neighbors': 5, 'weights': 'distance'}

In [132]: # подбираем оптимальные гиперпараметры для оптимальной модели
print(gs4.best_estimator_)
gs1 = gs4.best_estimator_
print("R2-score DTR для прочности при растижении, MAE: %s" % (gs1.score(x_test_1, y_test_1).round(3)))
DecisionTreeRegressor(criterion='poisson', max_depth=5, max_features='auto',
                      min_samples_leaf=10, min_samples_split=200)
R2-score DTR для прочности при растижении, MAE: 0.821

In [133]: #подбираем оптимальные гиперпараметры в нову модель метода дерева решений
dtr_grid = DecisionTreeRegressor(criterion = 'poisson', max_depth = 7, max_features = 'auto',
                                  min_samples_leaf = 100, min_samples_split = 250)
#обучаем модель
dtr_grid.fit(x_train_1, y_train_1)
predictions_dtr_grid = dtr_grid.predict(x_test_1)
#оцениваем точность на тестовом наборе
mae_dtr_grid = mean_absolute_error(predictions_dtr_grid, y_test_1)
mae_dtr_grid

Out[133]: 169.60058525087166

In [134]: new_row_in_mae_df = {'Perceptron': 'DecisionTree_GridSearchCV', 'MAE': mae_dtr_grid}
mae_df = mae_df.append(new_row_in_mae_df, ignore_index=True)
mae_df

```

	Regressor	MAE
0	Support Vector	76.111030
1	RandomForest	76.531400
2	Linear Regression	63.456419
3	GradientBoosting	61.110269
4	KNeighbors	122.095974
5	DecisionTree	113.159004
6	SGD	189.206680
7	MLP	1619.254573
8	Lasso	70.176750
9	RandomForest_GridSearchCV	72.086358
10	KNeighbors_GridSearchCV	110.733774
11	DecisionTree_GridSearchCV	169.600585

```
In [135]: pipe = Pipeline([('preprocessing', StandardScaler()), ('regressor', SVR())])
param_grid = [
    {'regressor': [SVR()], 'preprocessing': [StandardScaler(), MinMaxScaler(), None],
     'regressor__C': [0.001, 0.01, 0.1, 1, 10, 100],
     'regressor__epsilon': [0.001, 0.01, 0.1, 1, 10, 100]},
    {'regressor': [RandomForestRegressor(n_estimators = 100)],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [LinearRegression()],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [GradientBoostingRegressor()],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [KNeighborsRegressor()],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [DecisionTreeRegressor()],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [MLPRegressor(random_state = 1, max_iter = 500)],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [linear_model.Lasso(alpha = 0.1)],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]}]
grid = GridSearchCV(pipe, param_grid, cv = 10)
grid.fit(X_train_1, np.ravel(y_train))
print("Наилучшее значение правильности перекрестной проверки: {:.2f}.".format(grid.best_params_))
print("Наилучшее значение правильности перекрестной проверки: {:.2f}.".format(grid.best_score_))
print("Правильность на тестовом наборе: {:.2f}.".format(grid.score(X_test_1, y_test_3)))
print("Правильность на тестовом наборе: {:.2f}.".format(grid.score(X_test_1, y_test_3)))
```

Наилучшие параметры:
{'preprocessing': None, 'regressor': GradientBoostingRegressor()}

Наилучшее значение правильности перекрестной проверки: 0.97

Правильность на тестовом наборе: 0.97

```
In [136]: print("Наилучшая модель:\n".format(grid.best_estimator_))
```

Наилучшая модель:

Pipeline(steps=[('preprocessing', None),
 ('regressor', GradientBoostingRegressor())])