


```

df_nup = pd.read_excel(r'C:\Users\Avona\Desktop\VKor BKP\Datasets\x_nup.xlsx')
df_nup.shape
Out[6]: (1040, 4)

In [7]: # удаляем первый неинформативный столбец
df_nup.drop(['Unnamed: 0'], axis=1, inplace=True)
# Проверим, что первые 5 строк первого датасета идентичны, то есть нужный первый столбец успешно удален
df_nup.head()
Out[7]: Угол нашивки, град Шаг нашивки Плотность нашивки
0 0.0 4.0 57.0
1 0.0 4.0 60.0
2 0.0 4.0 70.0
3 0.0 5.0 47.0
4 0.0 5.0 57.0

In [8]: # Проверим размерность второго файла
df_nup.shape
Out[8]: (1040, 3)

Объединянем по индексу, тип объединения INNER, смотрим итоговый датасет

In [9]: # Понимаем, что эти два датасета имеют разные объемы строк.
# Но наша задача собрать исходные данные файлы в один, единий набор данных.
# По условию задачи объединим их по типу INNER.
df = df_bp.merge(df_nup, left_index = True, right_index = True, how = 'inner')
df.head().T
Out[9]:
0 1 2 3 4
Соотношение матрица-наполнитель 1.857143 1.857143 1.857143 1.857143 2.771331
Плотность, кг/м3 2030.000000 2030.000000 2030.000000 2030.000000 2030.000000
модуль упругости, ГПа 738.736842 738.736842 738.736842 738.736842 753.000000
Количество отвердителя, м.% 30.000000 50.000000 49.900000 129.000000 111.860000
Содержание эпоксидных групп, %_2 22.267857 23.750000 33.000000 21.250000 22.267857
Температура вспышки, С_2 100.000000 284.615385 284.615385 300.000000 284.615385
Поверхностная плотность, г/м2 210.000000 210.000000 210.000000 210.000000 210.000000
Модуль упругости при растяжении, ГПа 70.000000 70.000000 70.000000 70.000000 70.000000
Прочность при растяжении, МПа 3000.000000 3000.000000 3000.000000 3000.000000 3000.000000
Потребление смолы, г/м2 220.000000 220.000000 220.000000 220.000000 220.000000
Угол нашивки, град 0.000000 0.000000 0.000000 0.000000 0.000000
Шаг нашивки 4.000000 4.000000 4.000000 5.000000 5.000000
Плотность нашивки 57.000000 60.000000 70.000000 47.000000 57.000000

Проверяем размеры данных

In [10]: #Посмотрим количество колонок и строк
df.shape
# Итоговый датасет имеет 13 столбцов и 1023 строки, 17 строк из таблицы X_bp было отброшено, т.е. часть данных удалена на начальном этапе исследования.
Out[10]: (1023, 13)

Познакомимся с датасетом ближе, проведем разведочный анализ.

Знакомство с данными

In [11]: # Посмотрим на начальные и конечные строки нашего датасета на данном этапе работы
df
Out[11]: 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, град Шаг нашивки Плотность нашивки
0 1.857143 2030.000000 738.736842 30.000000 22.267857 100.000000 210.000000 70.000000 3000.000000 220.000000 0.0 4.000000 57.000000
1 1.857143 2030.000000 738.736842 50.000000 23.750000 284.615385 210.000000 70.000000 3000.000000 220.000000 0.0 4.000000 60.000000
2 1.857143 2030.000000 738.736842 49.900000 33.000000 284.615385 210.000000 70.000000 3000.000000 220.000000 0.0 4.000000 70.000000
3 1.857143 2030.000000 738.736842 129.000000 21.250000 300.000000 210.000000 70.000000 3000.000000 220.000000 0.0 5.000000 47.000000
4 2.771331 2030.000000 753.000000 111.860000 22.267857 284.615385 210.000000 70.000000 3000.000000 220.000000 0.0 5.000000 57.000000
...
1018 2.271346 1952.087902 912.855545 86.992183 20.123249 324.774576 209.198700 73.090961 2387.292495 125.07669 90.0 9.076380 47.019770
1019 3.444022 2059.089171 444.732634 145.981978 19.599769 254.215401 350.666830 72.920827 2360.392784 117.730999 90.0 10.565614 53.750790
1020 3.280604 1972.372865 416.836524 110.533477 23.957502 248.423047 740.142791 74.734344 2662.906940 236.606764 90.0 4.161154 67.629884
1021 3.705351 2066.799773 741.475517 141.397963 192.46945 275.779840 641.468152 74.042708 2071.715856 197.126067 90.0 6.313201 58.261074
1022 3.808020 1890.413468 417.316232 129.183416 27.474763 300.952708 758.747882 74.309704 2856.328932 194.754342 90.0 6.078902 77.434468
1023 rows x 13 columns

In [12]: #Просмотрим информацию о датасете, проверим тип данных в каждом столбце (типы признаков)
df.info()
# Все переменные содержат значения float64, качественные характеристики отсутствуют. Пропусков не имеется. Ни одна из записей не является NaN, очистка не требуется. Объединенный файл имеет всего 1023 строки.

class 'pandas.core.frame.DataFrame'
Int64Index: 1023 entries, 0 to 1022
Data columns (total 13 columns):
 # ... 
Non-Null Count Dtype
-- 
0 Соотношение матрица-наполнитель 1023 non-null float64
1 Плотность, кг/м3 1023 non-null float64
2 модуль упругости, ГПа 1023 non-null float64
3 Количество отвердителя, м.% 1023 non-null float64
4 Содержание эпоксидных групп, %_2 1023 non-null float64
5 Температура вспышки, С_2 1023 non-null float64
6 Поверхностная плотность, г/м2 1023 non-null float64
7 Модуль упругости при растяжении, ГПа 1023 non-null float64
8 Прочность при растяжении, МПа 1023 non-null float64
9 Потребление смолы, г/м2 1023 non-null float64
10 Угол нашивки, град 1023 non-null float64
11 Шаг нашивки 1023 non-null float64
12 Плотность нашивки 1023 non-null float64
dtypes: float64(13)
memory usage: 111.9 KB

In [13]: #Посчитаем уникальные значения с помощью функции nunique
df.nunique()
# Видим в основном общее число уникальных значений в каждом столбце, но в столбце "Угол нашивки" всего 2 значения. Поработаем с ним.

Out[13]: Соотношение матрица-наполнитель 1014
Плотность, кг/м3 1013
модуль упругости, ГПа 1020
Количество отвердителя, м.% 1085
Содержание эпоксидных групп, %_2 1084
Температура вспышки, С_2 1083
Поверхностная плотность, г/м2 1084
Модуль упругости при растяжении, ГПа 1084
Прочность при растяжении, МПа 1083
Потребление смолы, г/м2 1083
Угол нашивки, град 2
Шаг нашивки 989
Плотность нашивки 988
dtype: int64

In [14]: # Поработаем со столбцом "Угол нашивки"
Out[14]: # Проверим кол-во элементов, где Угол нашивки равен 0 градусов
df['Угол нашивки, град'].nunique()
# Видим как кол-во уникальных значений в колонке Угол нашивки равно 2, можем привести данные в этой колонке к значениям 0 и 1
Out[15]: 2

In [16]: #Проверим кол-во элементов, где Угол нашивки, град = 0
df['Угол нашивки, град'][df['Угол нашивки, град'] == 0].count()
520

In [17]: #Прибедем столбец "Угол нашивки" к значениям 0 и 1
df.replace({'Угол нашивки, град': {0.0 : 0, 90.0 : 1}})
df['Угол нашивки, град'] = df['Угол нашивки, град'].astype(int)
Out[16]: 

In [18]: #Изменим столбец
df = df.rename(columns={'Угол нашивки, град': 'Угол нашивки'})
df

```



```
mean_and_50 = df.describe()  
mean_and_50.loc[["mean", "50%"]]  
#в целом мы видим близкое друг к другу значение
```

	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
mean	2.930366	1975.734888	739.923233	110.570769	22.244390	285.882151	482.731833	73.328571	2466.928843	218.423144	0.491691	6.89922	57.153929
50%	2.906878	1977.621657	739.664328	110.56840	22.230744	285.896012	451.864365	73.268805	2459.524526	219.198882	0.000000	6.916144	57.341920

In [32]: # среднее значение

In [33]: df.mean()

```
Соотношение матрица-наполнитель 2.930366  
Плотность, кг/м³ 1975.734888  
модуль упругости, ГПа 739.923233  
Количество отвердителя, м.% 110.564328  
Содержание эпоксидных групп,%_2 22.244390  
Температура вспышки, С_2 285.882151  
Поверхностная плотность, г/м² 482.731833  
Модуль упругости при растяжении, ГПа 73.328571  
Прочность при растяжении, МПа 2466.928843  
Потребление смолы, г/м² 218.423144  
Угол нашивки 0.491691  
Шаг нашивки 6.89922  
Плотность нашивки 57.153929  
dtype: float64
```

In [34]: # медианное значение

In [35]: df.median()

```
Соотношение матрица-наполнитель 2.986978  
Плотность, кг/м³ 1977.621657  
модуль упругости, ГПа 739.664328  
Количество отвердителя, м.% 110.564348  
Содержание эпоксидных групп,%_2 22.238744  
Температура вспышки, С_2 285.896012  
Поверхностная плотность, г/м² 451.864365  
Модуль упругости при растяжении, ГПа 73.268805  
Прочность при растяжении, МПа 2459.524526  
Потребление смолы, г/м² 219.198882  
Угол нашивки 0.899899  
Шаг нашивки 6.316144  
Плотность нашивки 57.341928  
dtype: float64
```

In [36]: # Вычисляем коэффициенты ранговой корреляции Кендалла. Статистической зависимости не наблюдаем.

In [36]: df.corr(method = 'kendall')

```
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  
Соотношение матрица-наполнитель 1.000000 -0.001315 0.021247 0.001410 0.010180 -0.009480 -0.002060 -0.004157 0.011614 0.035145 -0.021395 0.022723 0.002788  
Плотность, кг/м³ -0.001315 1.000000 -0.008059 -0.021963 -0.007758 -0.019947 0.037302 -0.021151 -0.047426 -0.017079 -0.051525 -0.031220 0.002935  
модуль упругости, ГПа 0.021247 -0.008059 1.000000 0.022382 0.002351 0.021028 -0.000442 0.005458 0.022959 0.005169 -0.031695 -0.008305 0.049347  
Количество отвердителя, м.% 0.001410 -0.021963 0.022382 1.000000 0.000010 0.059034 0.031110 -0.043140 0.046507 -0.003677 0.024690 0.006232 0.016607  
Содержание эпоксидных групп,%_2 0.010180 -0.007758 0.003351 0.000010 1.000000 -0.002170 -0.006859 0.041994 -0.013441 0.009756 0.004668 -0.004539 -0.021968  
Температура вспышки, С_2 -0.009480 -0.019947 0.021028 0.059034 0.000000 0.017196 0.016481 -0.019106 0.019106 0.035313 0.017880 0.029552 0.005268  
Поверхностная плотность, г/м² -0.002060 0.037302 -0.000442 0.031110 -0.006859 0.017196 1.000000 0.024051 0.005099 -0.004446 0.045452 0.025514 -0.022320  
Модуль упругости при растяжении, ГПа -0.004157 -0.021151 0.005458 -0.043140 0.041994 0.016481 0.024051 1.000000 0.006599 0.034814 0.022431 -0.010024 0.002600  
Прочность при растяжении, МПа 0.011614 -0.047426 0.022959 -0.046507 -0.013441 -0.019106 -0.005099 1.000000 0.013580 0.020609 -0.048049 0.009821  
Потребление смолы, г/м² 0.035145 -0.017079 0.005169 -0.003677 0.009756 0.035313 -0.004446 0.022431 1.000000 -0.002402 0.005962 0.010792  
Угол нашивки -0.021395 -0.051525 -0.031220 0.022723 0.006232 -0.004539 0.021178 0.002230 0.002402 1.000000 0.021178 0.082142  
Шаг нашивки 0.002788 0.002935 0.002952 0.002952 0.002952 -0.002320 0.009821 0.0013580 0.0013580 0.0013580 1.000000 0.000658  
Плотность нашивки 0.002788 0.002935 0.002952 0.002952 0.002952 -0.002320 0.009821 0.0013580 0.0013580 0.0013580 0.000658 1.000000
```

In [37]: #Вычисляем коэффициенты корреляции Пирсона. Статистической зависимости не наблюдаем.

In [37]: df.corr(method = 'pearson')

```
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  
Соотношение матрица-наполнитель 1.000000 0.003841 0.031700 -0.006445 0.019766 -0.004776 -0.006272 -0.008411 0.024148 0.072531 -0.031073 0.036437 -0.004652  
Плотность, кг/м³ 0.003841 1.000000 -0.009547 -0.035911 -0.008278 -0.020695 0.044930 -0.017602 -0.069981 -0.015937 -0.068474 -0.061015 0.080304  
модуль упругости, ГПа 0.031700 -0.009547 1.000000 0.024049 -0.006804 0.031174 -0.005306 0.023267 0.041868 0.001840 -0.025417 -0.009875 0.056346  
Количество отвердителя, м.% -0.006445 -0.035911 0.024049 1.000000 -0.000684 0.095193 0.05198 0.065929 -0.075375 0.007440 0.038570 0.014887 0.017248  
Содержание эпоксидных групп,%_2 0.019766 -0.008278 -0.006804 0.000684 1.000000 -0.009769 -0.012340 0.056828 -0.023899 0.015165 0.008052 0.003022 -0.039073  
Температура вспышки, С_2 -0.004776 -0.020695 0.031174 0.0095193 -0.009769 0.000000 0.020121 0.028414 0.031763 0.059954 0.020695 0.025795 0.011391  
Поверхностная плотность, г/м² -0.006272 0.044930 -0.005306 0.005198 -0.012940 0.020121 1.000000 0.036702 0.003210 0.015692 0.052299 0.038332 -0.049923  
Модуль упругости при растяжении, ГПа -0.008411 -0.017602 0.023267 -0.065929 0.056828 0.028414 0.036702 1.000000 -0.009009 0.050938 0.023003 -0.023848 0.006476  
Прочность при растяжении, МПа 0.024148 -0.069981 0.041868 -0.075375 -0.023899 0.031763 -0.003210 0.009009 1.000000 0.028602 0.023398 -0.059547 0.019604  
Потребление смолы, г/м² 0.072531 -0.015937 0.001840 0.007446 0.015165 0.059954 0.015692 0.050938 0.028602 1.000000 -0.015334 0.013394 0.012239  
Угол нашивки -0.031073 -0.068474 -0.025417 0.038570 0.080502 0.020695 0.052299 0.023398 -0.015334 0.000000 1.000000 0.023616 0.0107947  
Шаг нашивки 0.036437 -0.061015 -0.009875 0.014867 0.003022 0.03795 0.036332 0.029468 -0.059547 0.013394 0.023616 1.000000 0.003487  
Плотность нашивки -0.004652 0.080304 0.056346 0.017248 -0.039073 0.011391 -0.049923 0.019604 0.012239 0.0107947 0.003487 0.000000 1.000000
```

In [38]: #Создадим переменную для называния всех столбцов. Это нам придется при построении коделей. И передадим к визуализации данных

```
df.columns  
column_names = ["Соотношение матрица-наполнитель", "Плотность, кг/м³", "модуль упругости, ГПа", "Количество отвердителя, м.%","Содержание эпоксидных групп,%_2", "Temperatura 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"]
```

Визуализируем сырье данные и проведем анализ

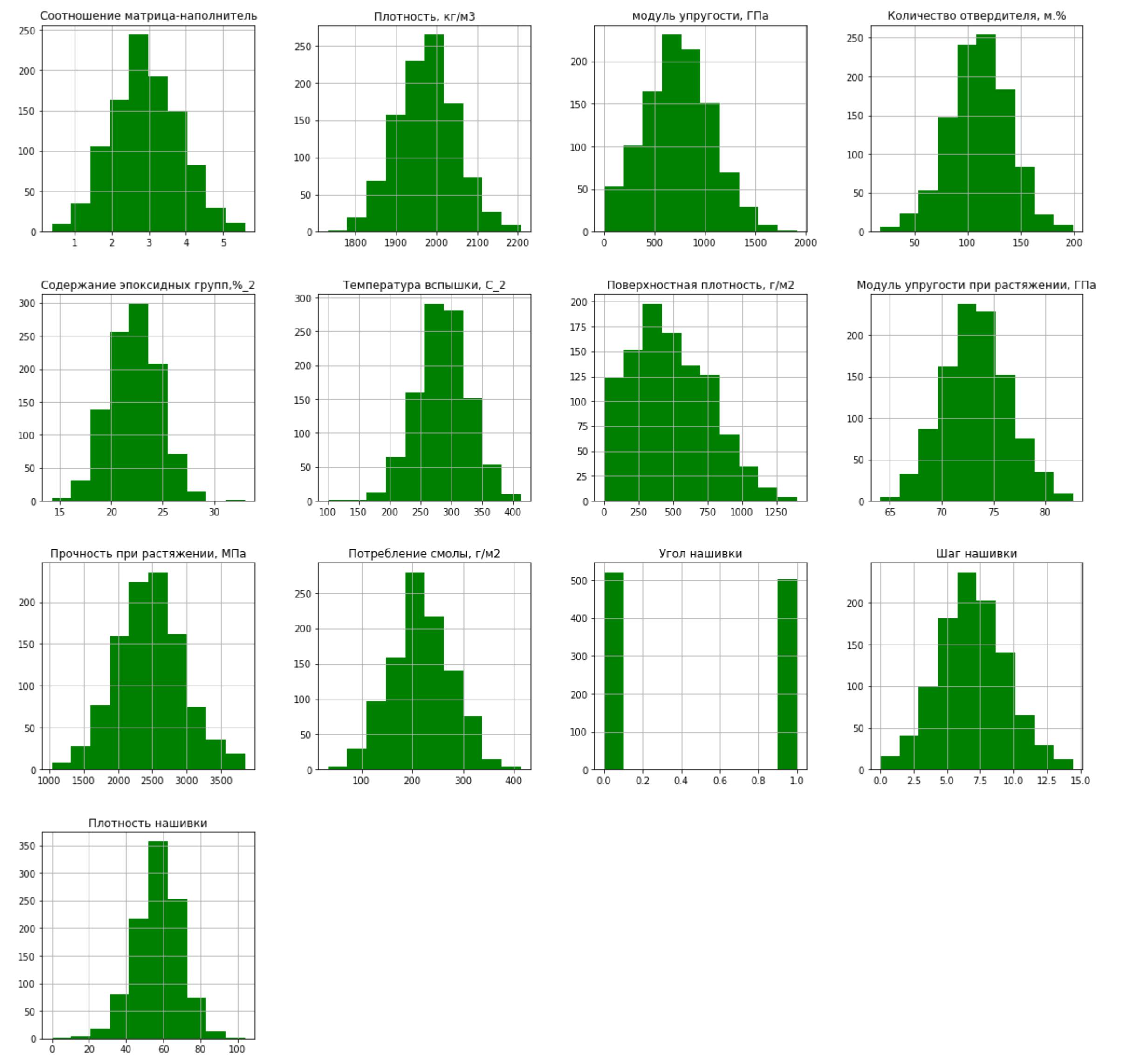
- Построим гистограммы распределения каждой из переменных и боксplotы (несколько разных способов визуализации),
- диаграммы "ящиков с усами" (несколько вариантов),
- попарные графики рассеяния точек (несколько вариантов)
- графики квантиль-квантиль

без нормализации и исключения шумов

Т.к. первый взгляд на общий файл и дополнительный анализ в excel не дал каких-то явных и бросающихся в глаза закономерностей, то используем разные варианты визуализации в надежде, что получится увидеть какую-то корреляцию. И разные варианты одного и того же типа визуализации используются для отображения результата, потому что какие-то графики отображаются в jupyter, но не работают в colab, какие-то не работают в Github

In [39]:

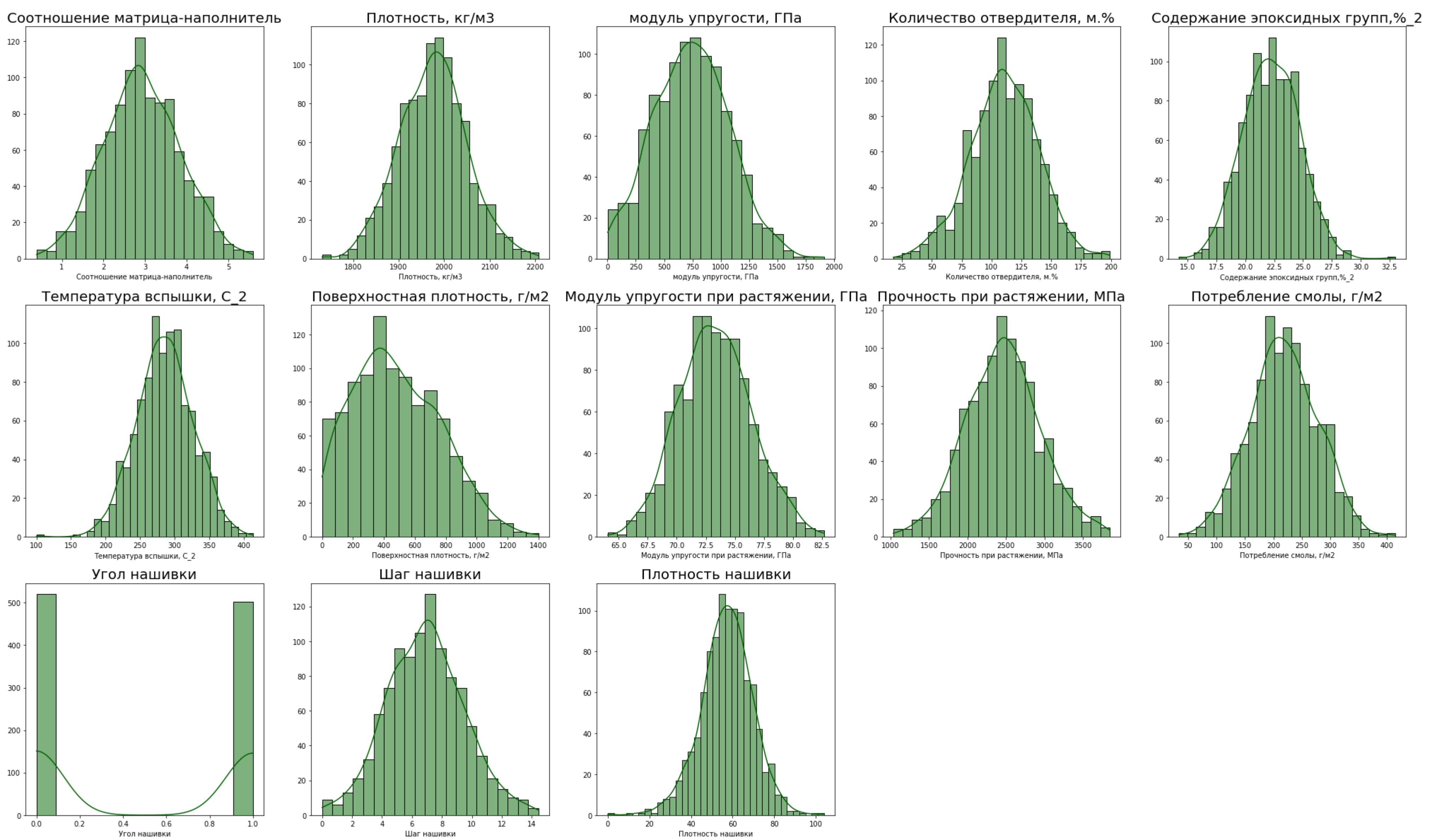
```
#Построим гистограммы распределения каждой из переменных без нормализации и исключения шумов  
df.hist(figsize = (20,20), color = "g")  
plt.show()
```



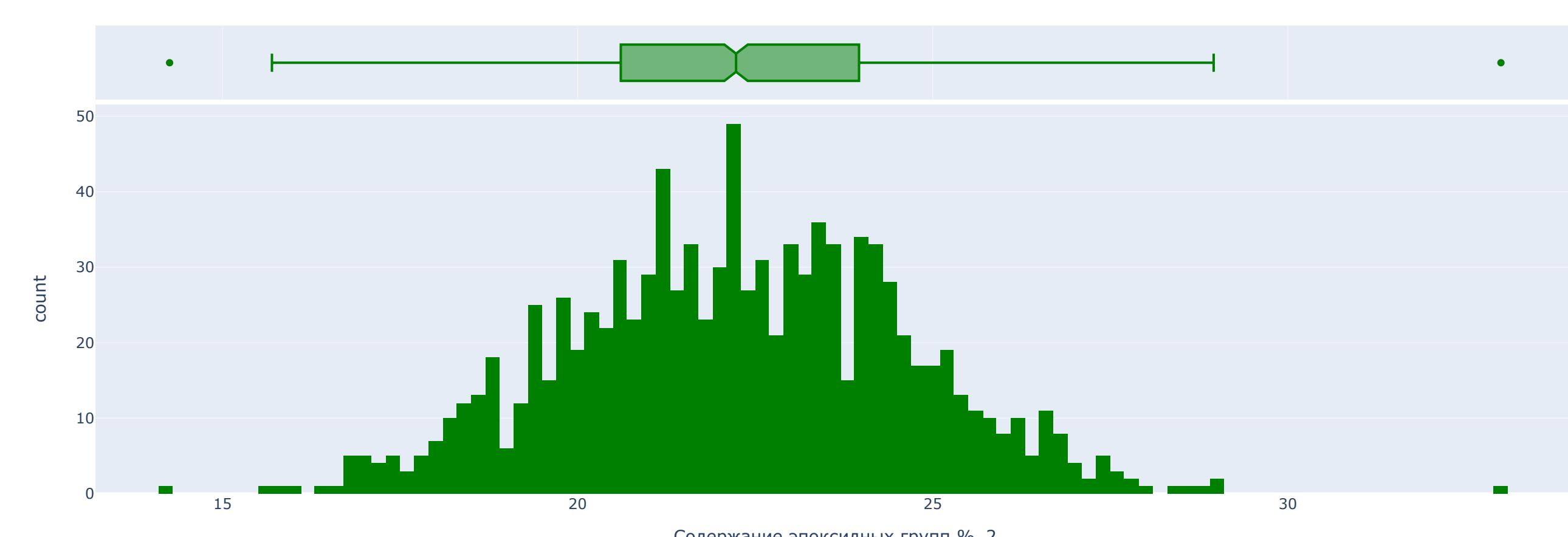
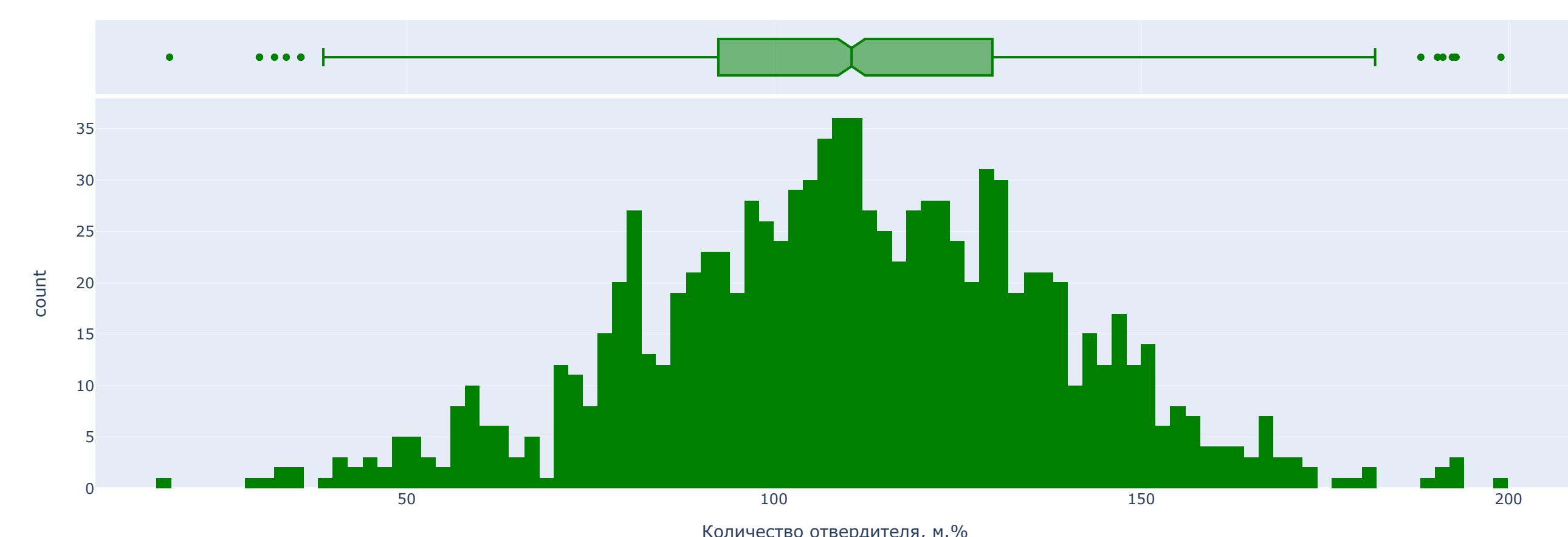
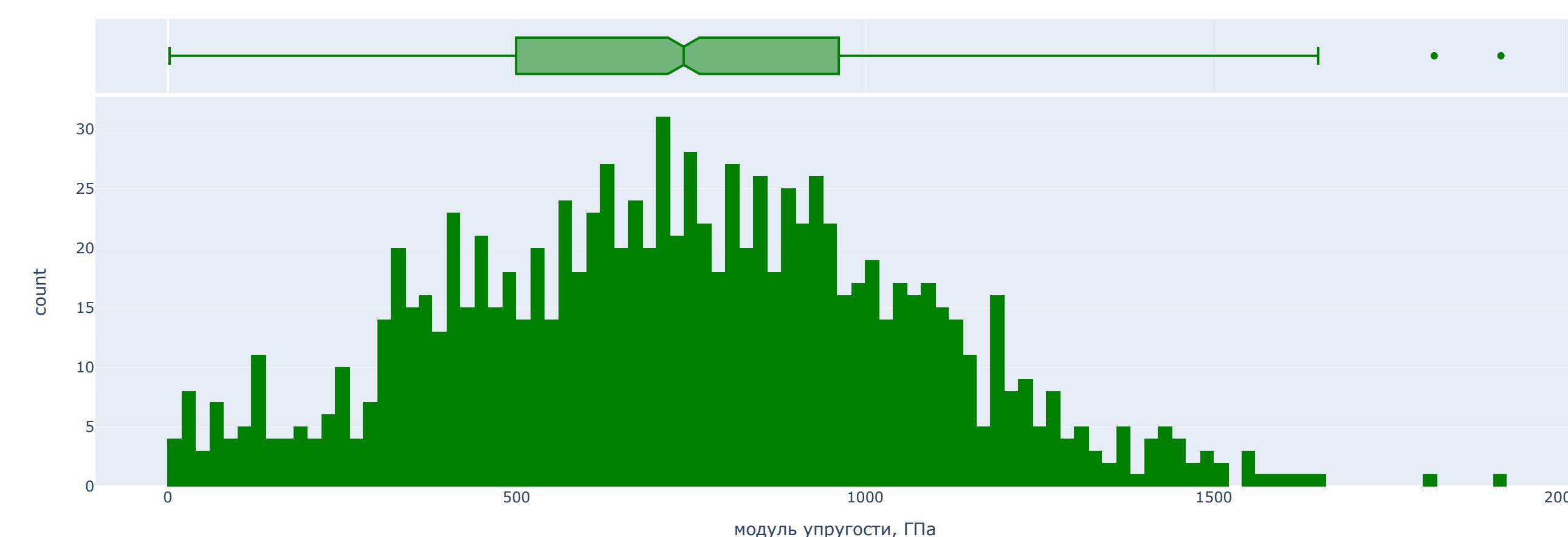
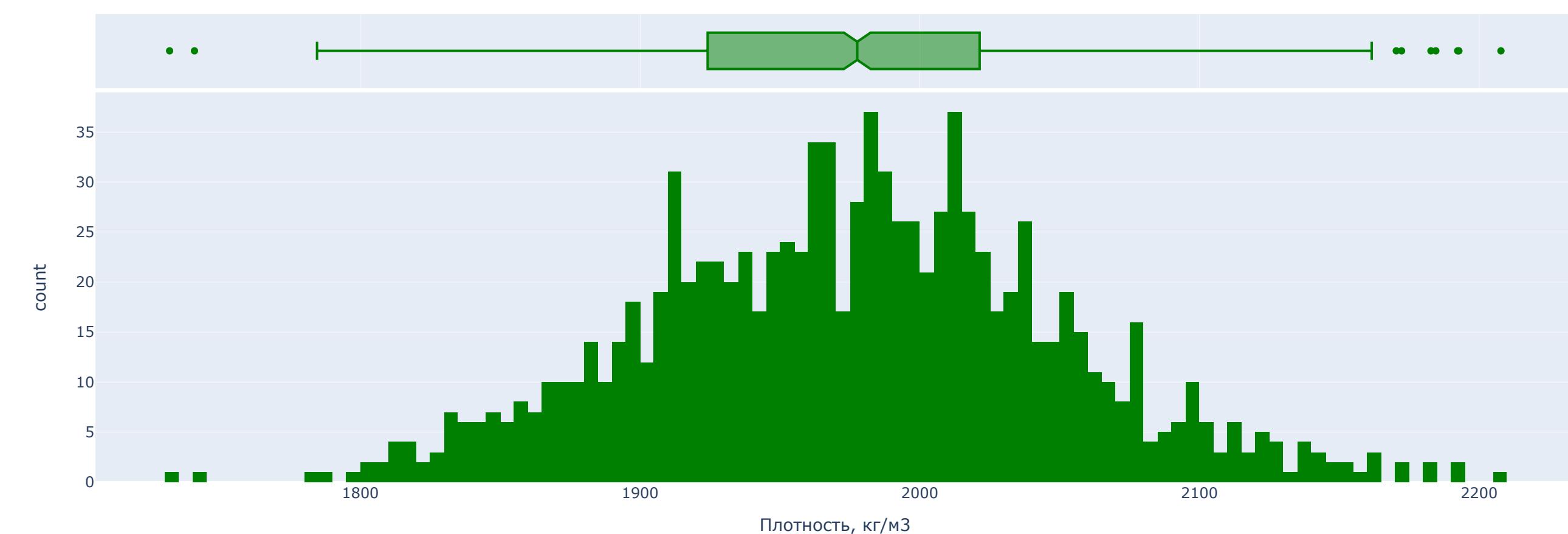
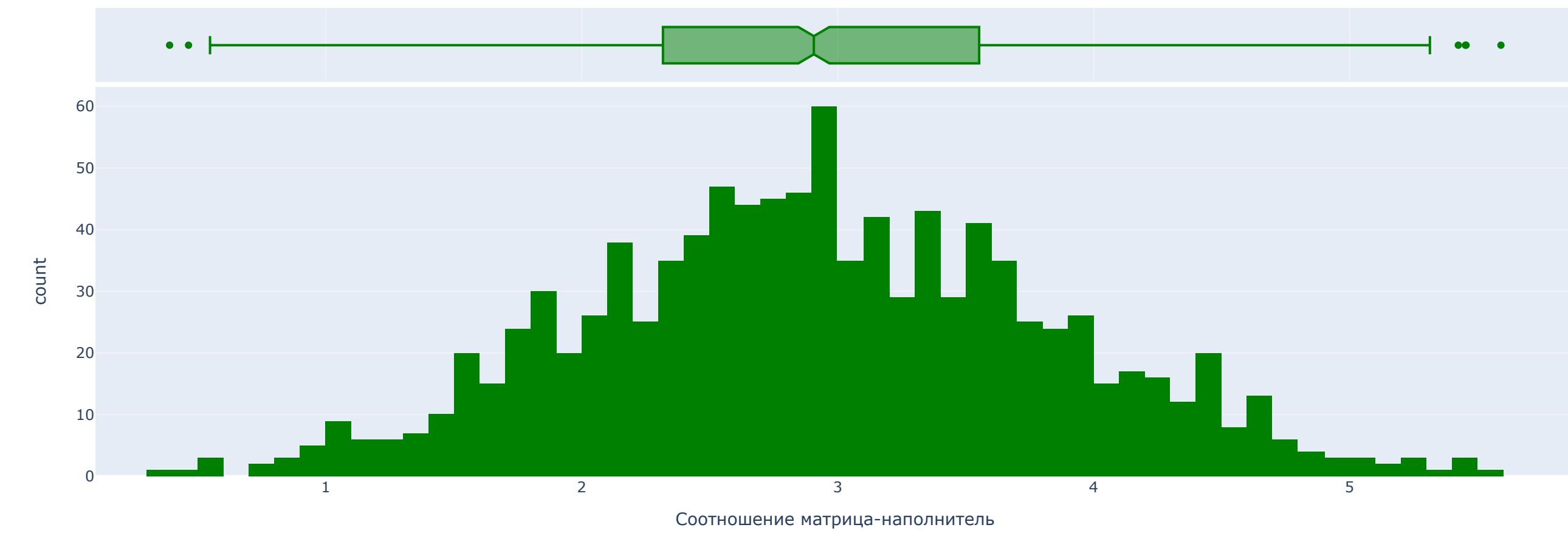
```
In [40]: # Гистограммы распределения (второй вариант)
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # количество ячеек
plt.figure(figsize = (35,35))
plt.suptitle('Гистограммы переменных', fontsize = 30)
for col in df.columns:
    plt.subplot(a, b, c)
    plt.figure(figsize=(7,5))
    sns.histplot(data = df[col], kde=True, color = "darkgreen")
    plt.title(col, size = 20)
    plt.show()
    c += 1
```

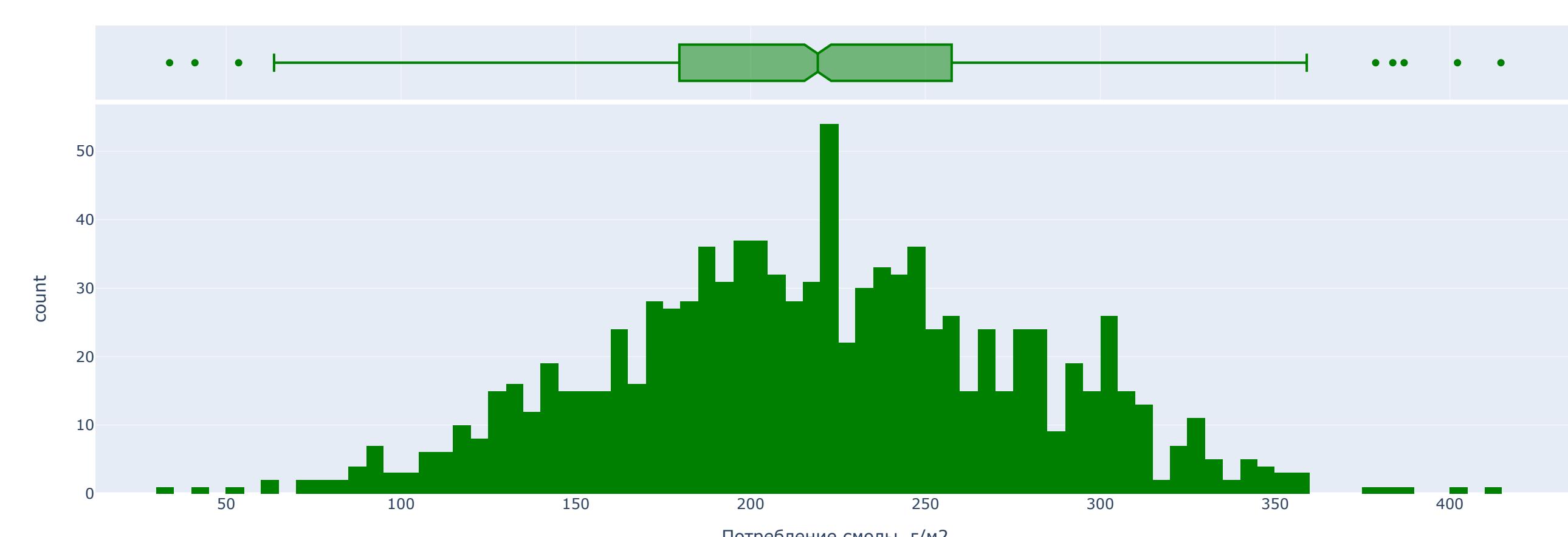
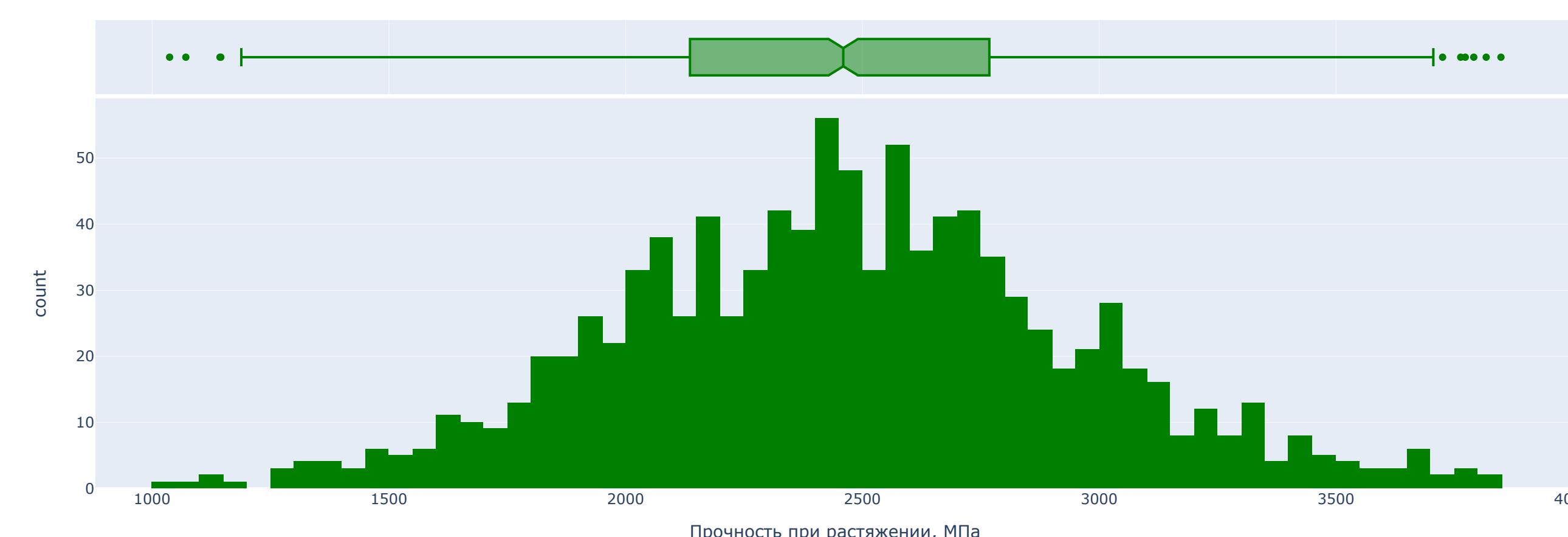
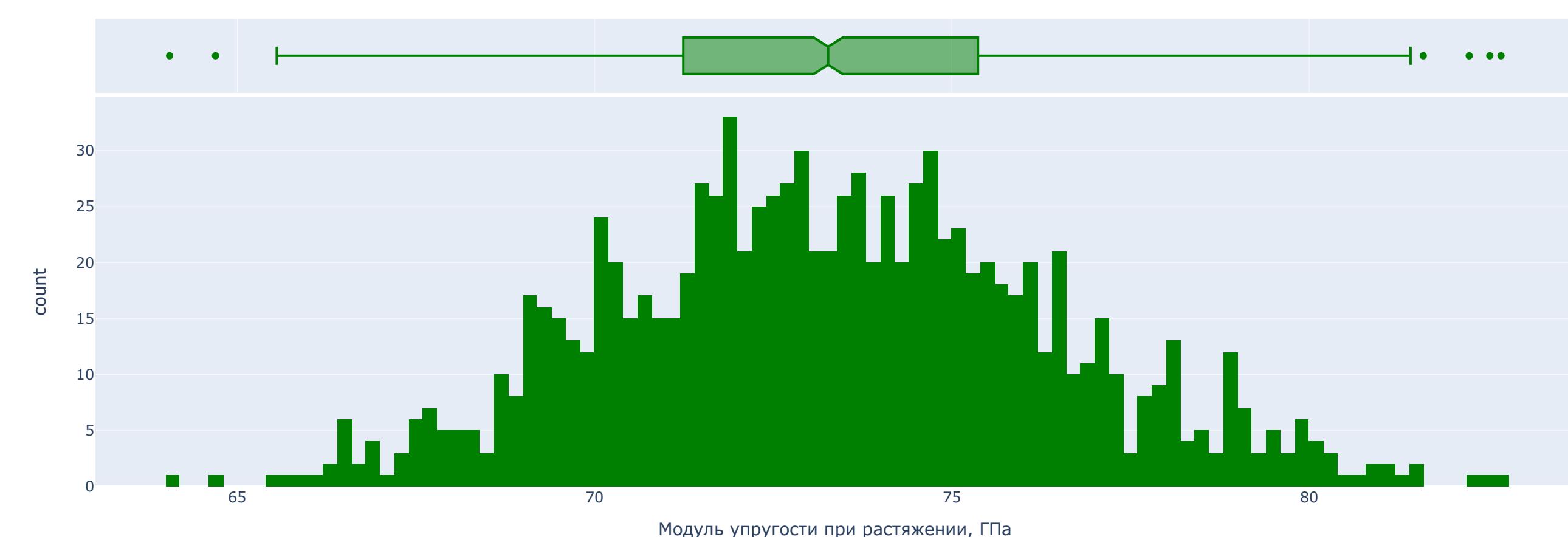
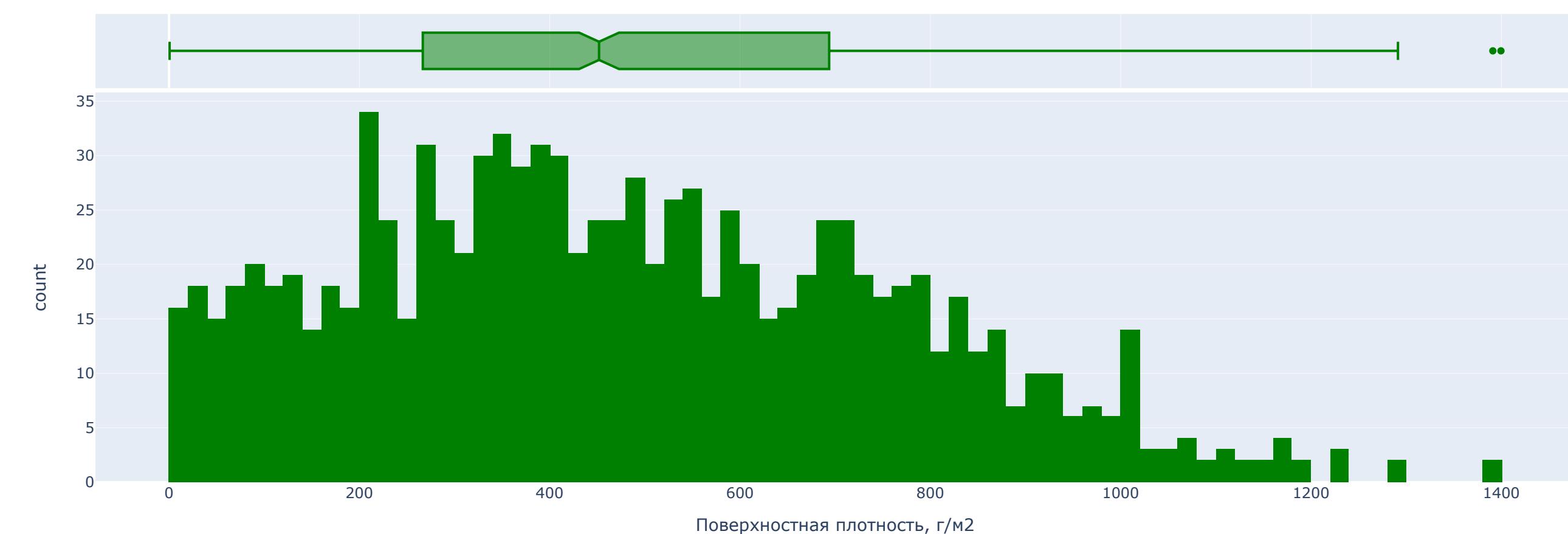
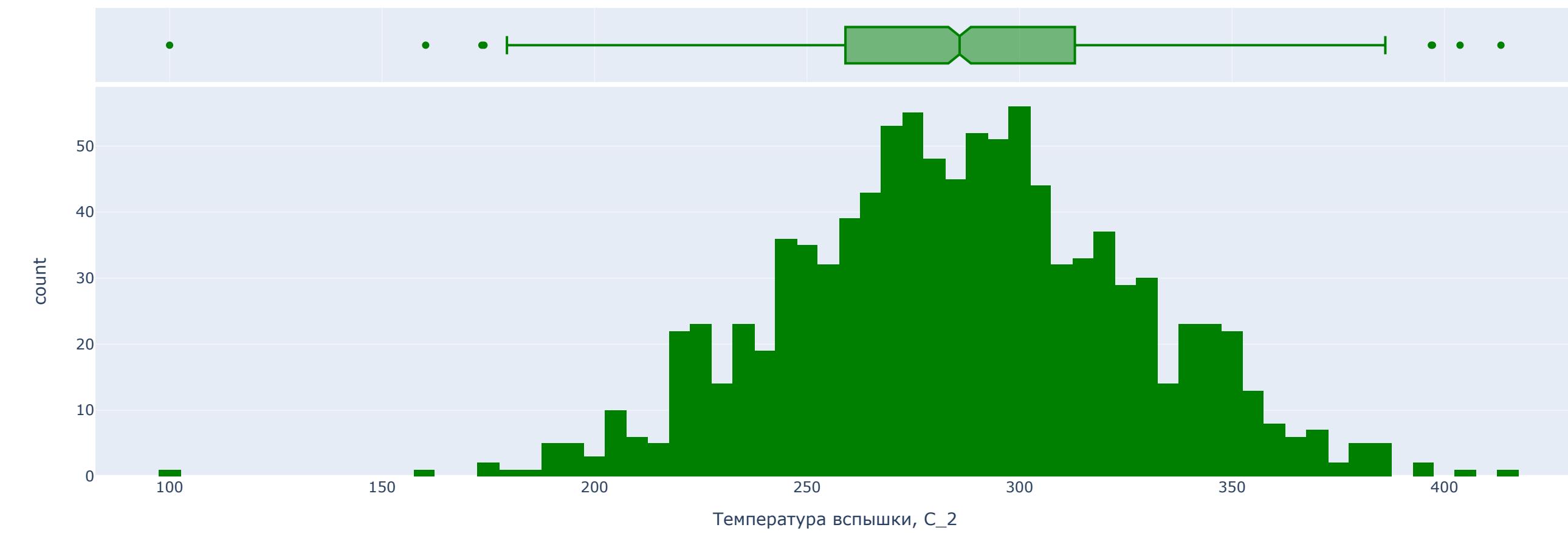
Гистограммы показывают приближенные выбросы в столбцах: плотность, содержание эпоксидных групп, температура вспышки, плотность нашивки.
Данные стремятся к нормальному распределению практически всегда, кроме угла нашивки, имеющим только 2 значения, с которым мы уже работали ранее.

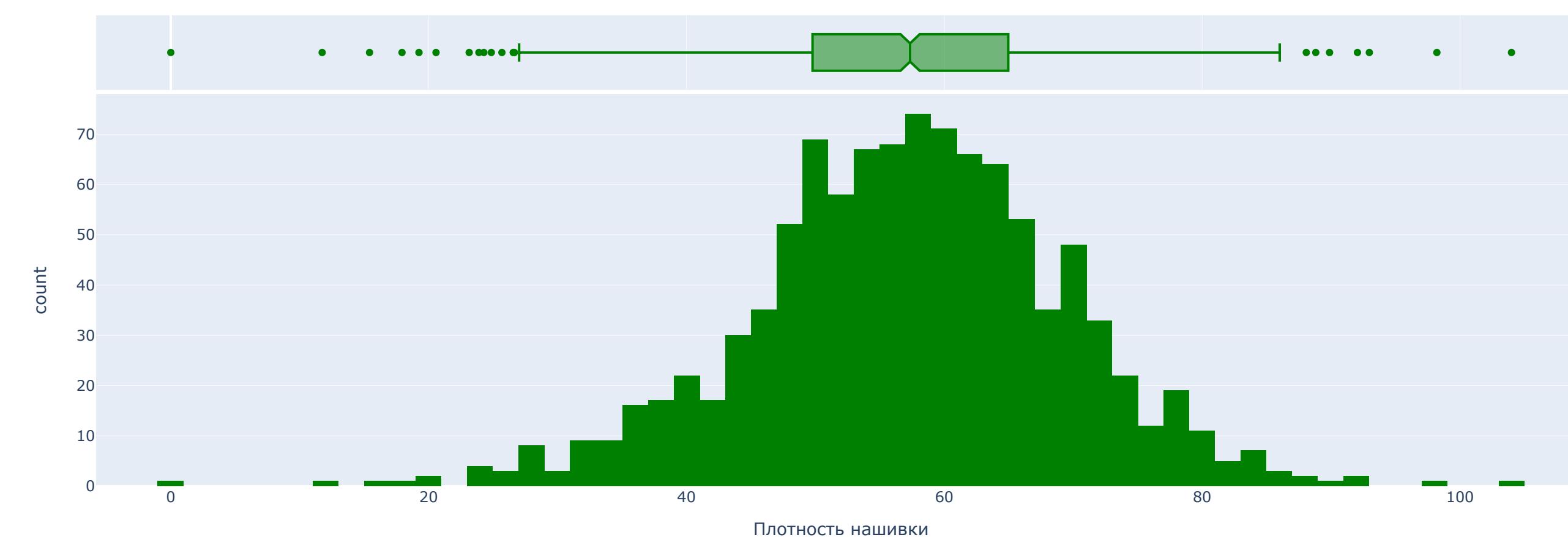
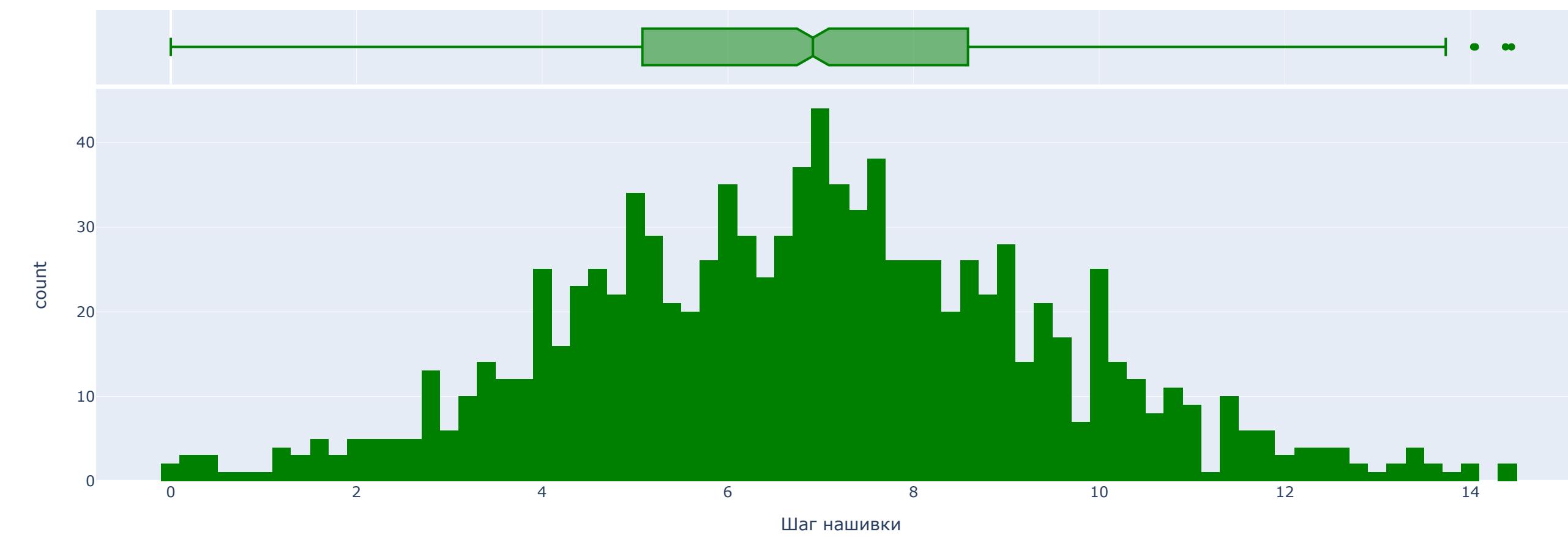
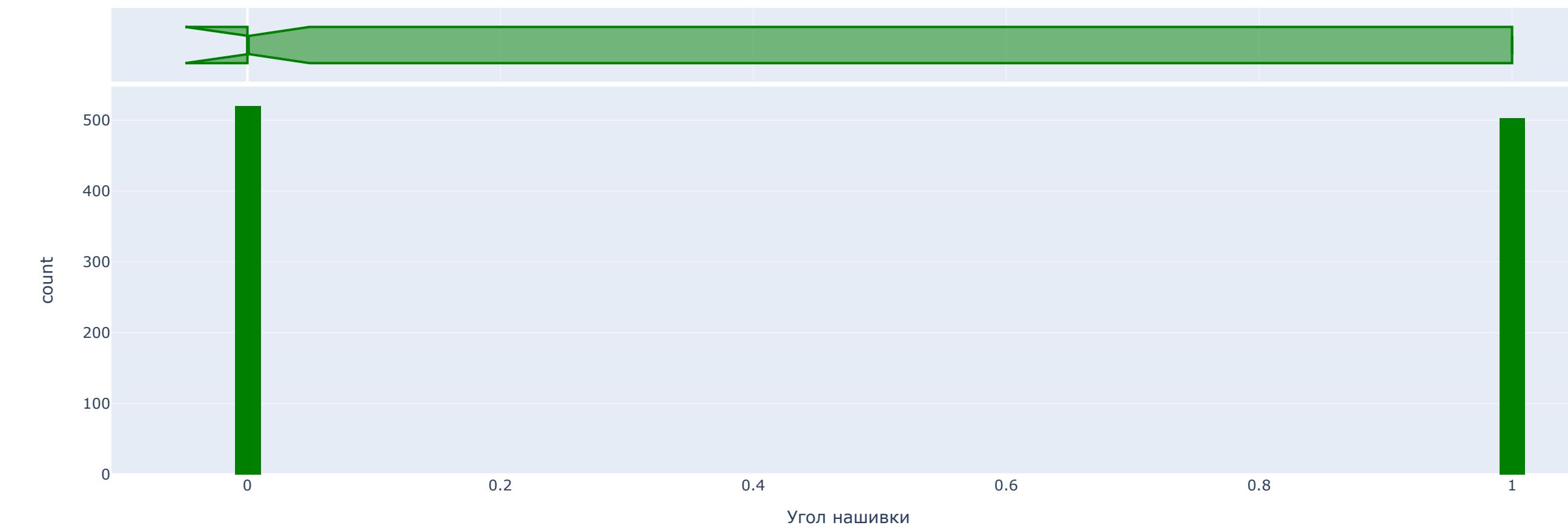
Гистограммы переменных



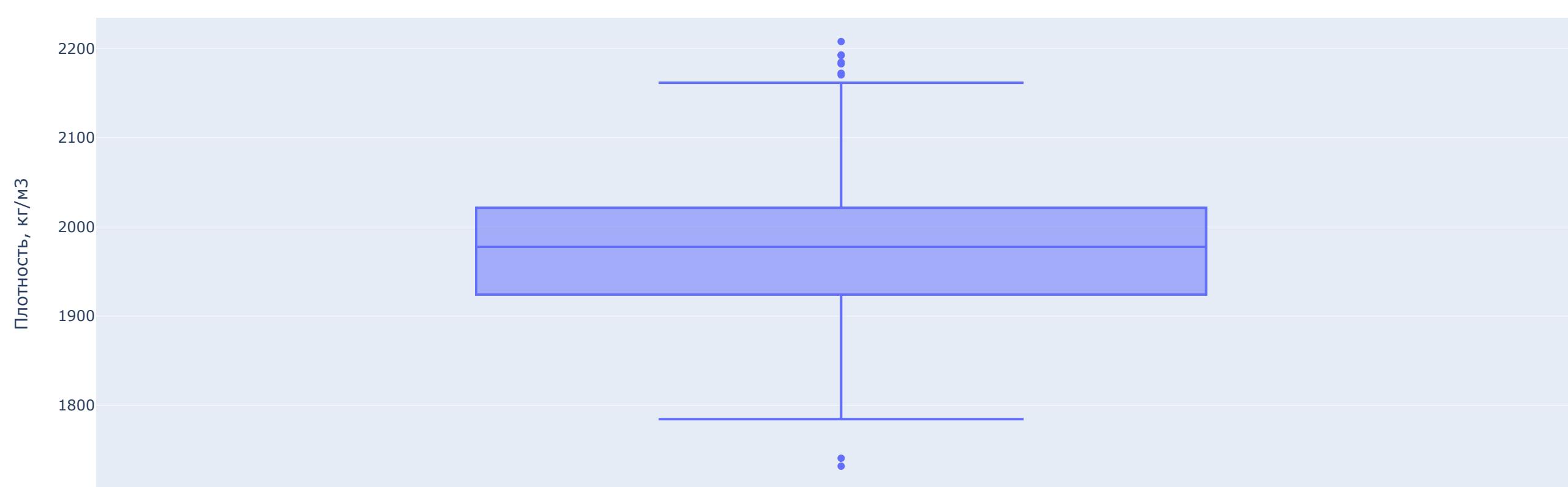
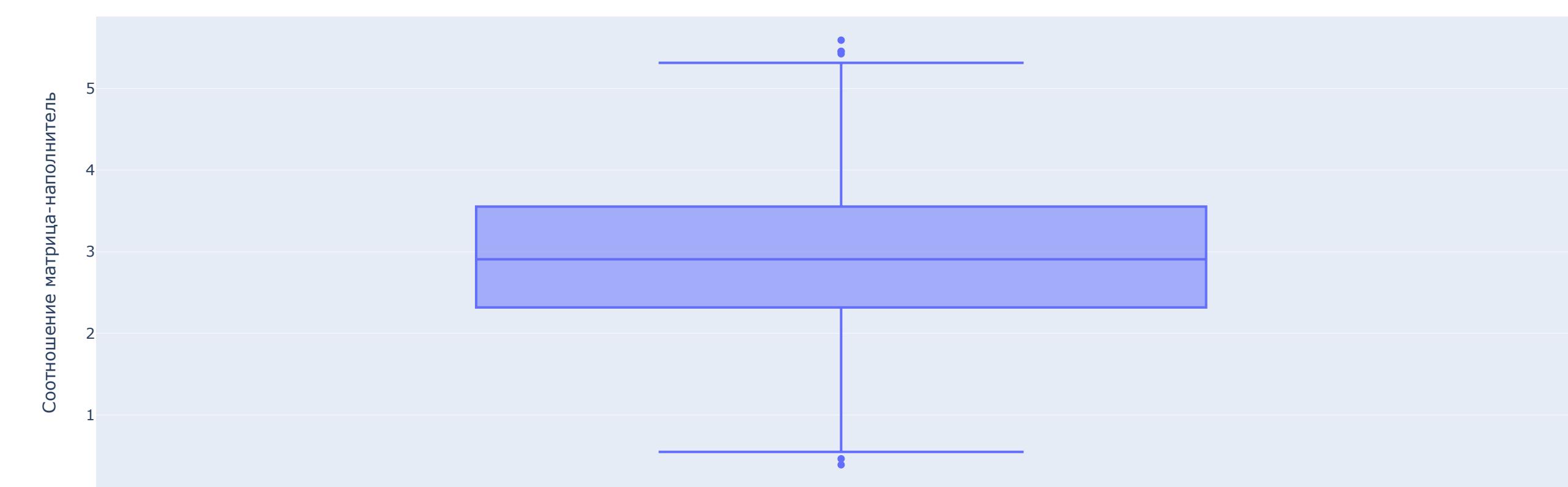
```
In [41]: # гистограмма распределения и боксплоты (третий вариант)
for column in df.columns:
    fig = px.histogram(df, x = column, color_discrete_sequence = ['green'], nbins = 100, marginal = "box")
    fig.show()
```

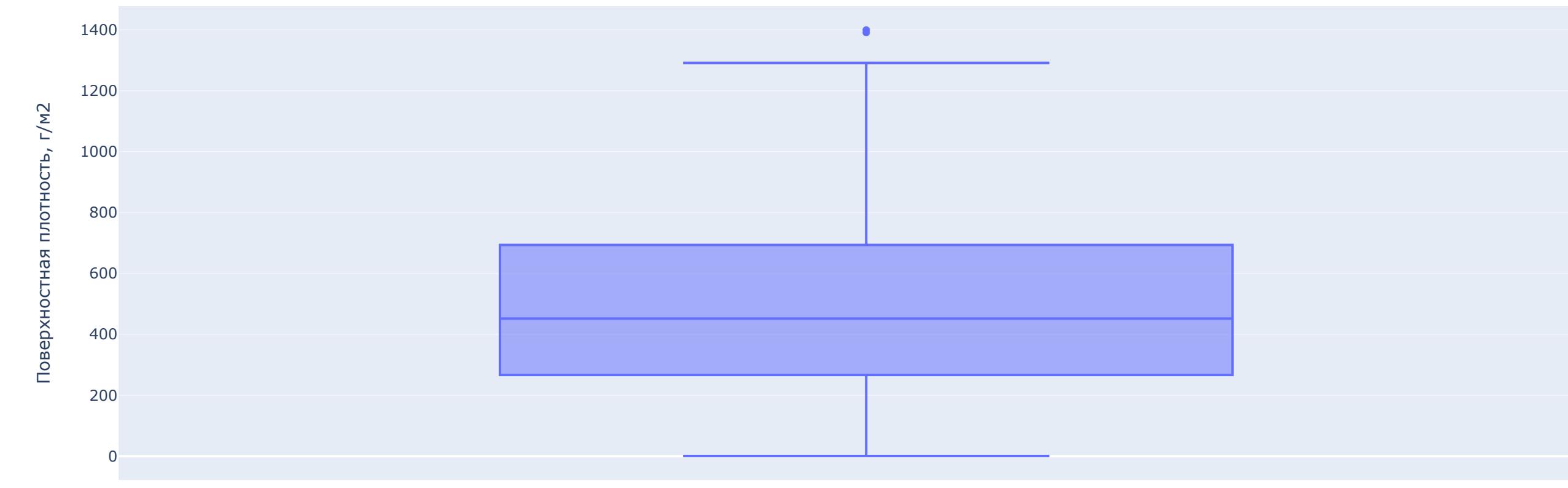
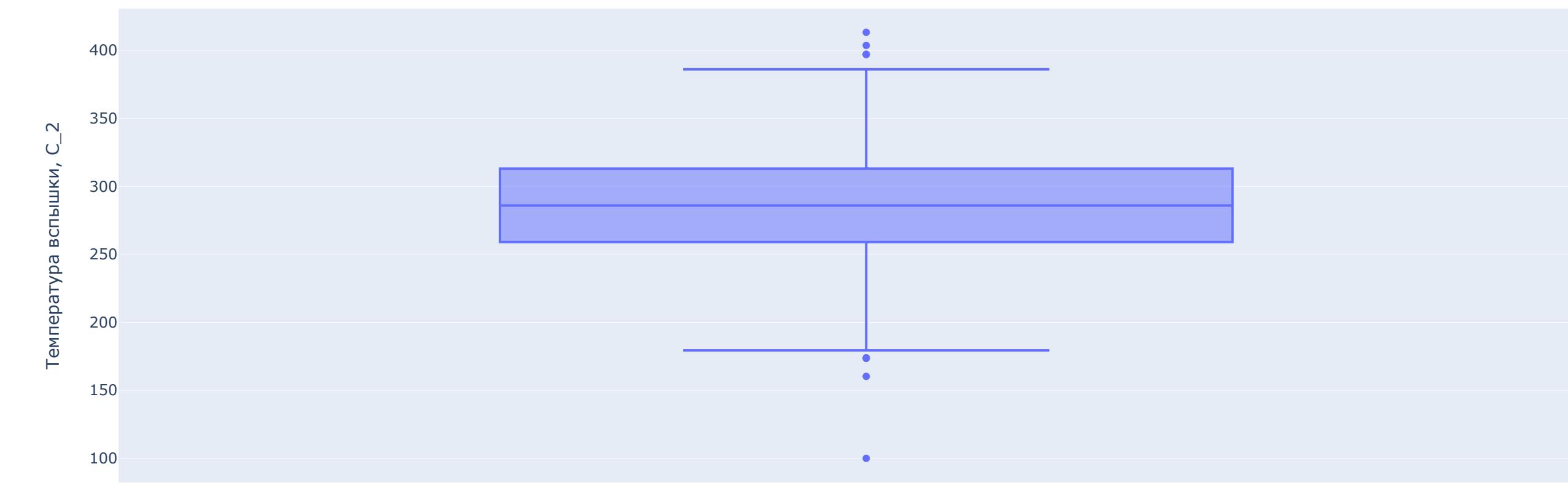
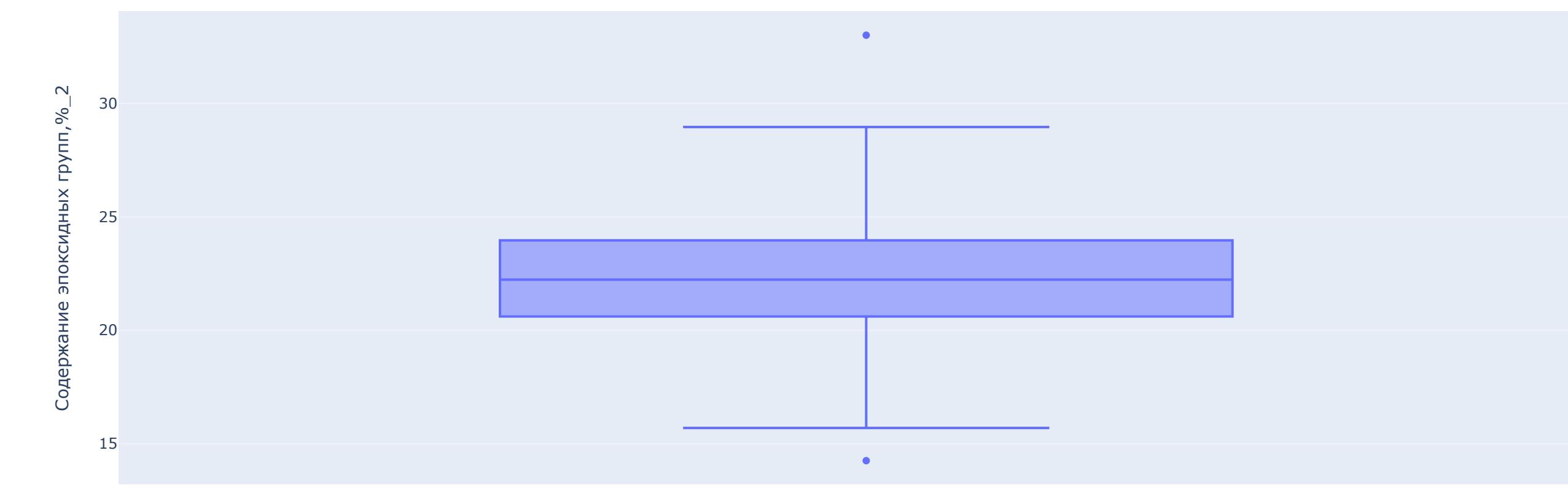
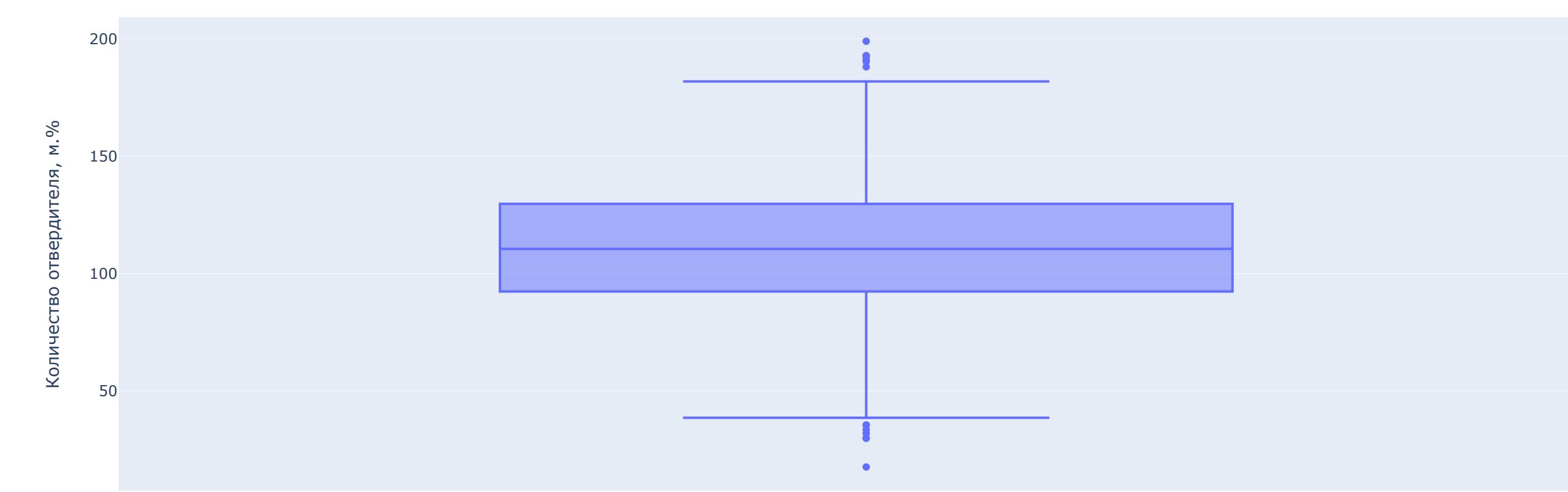
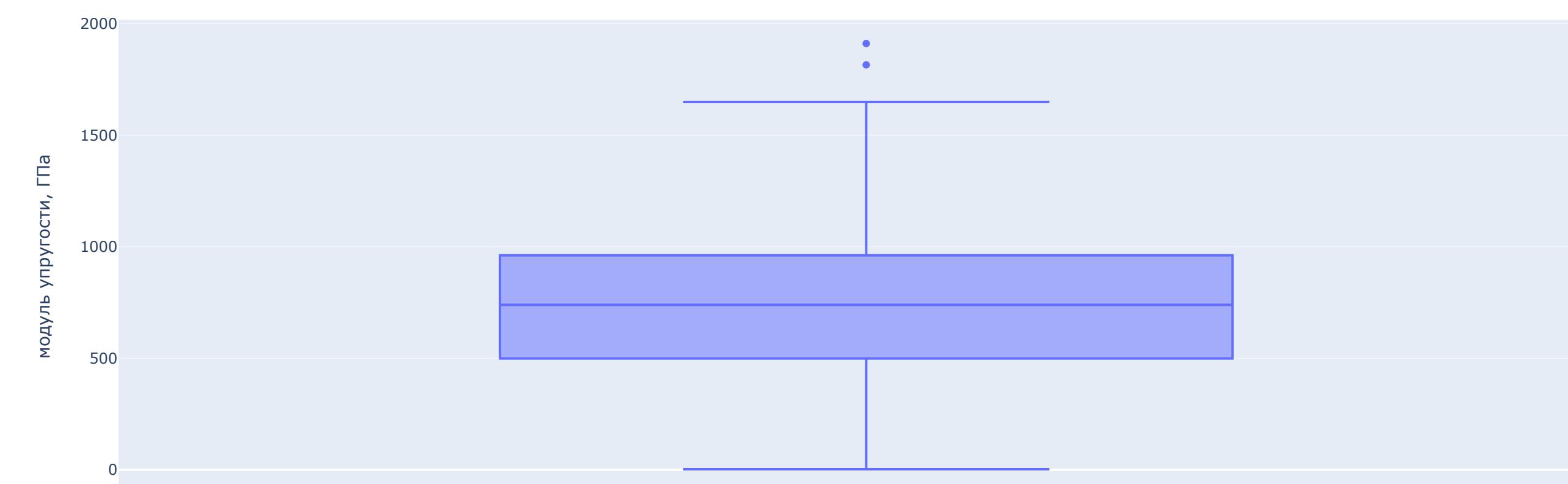


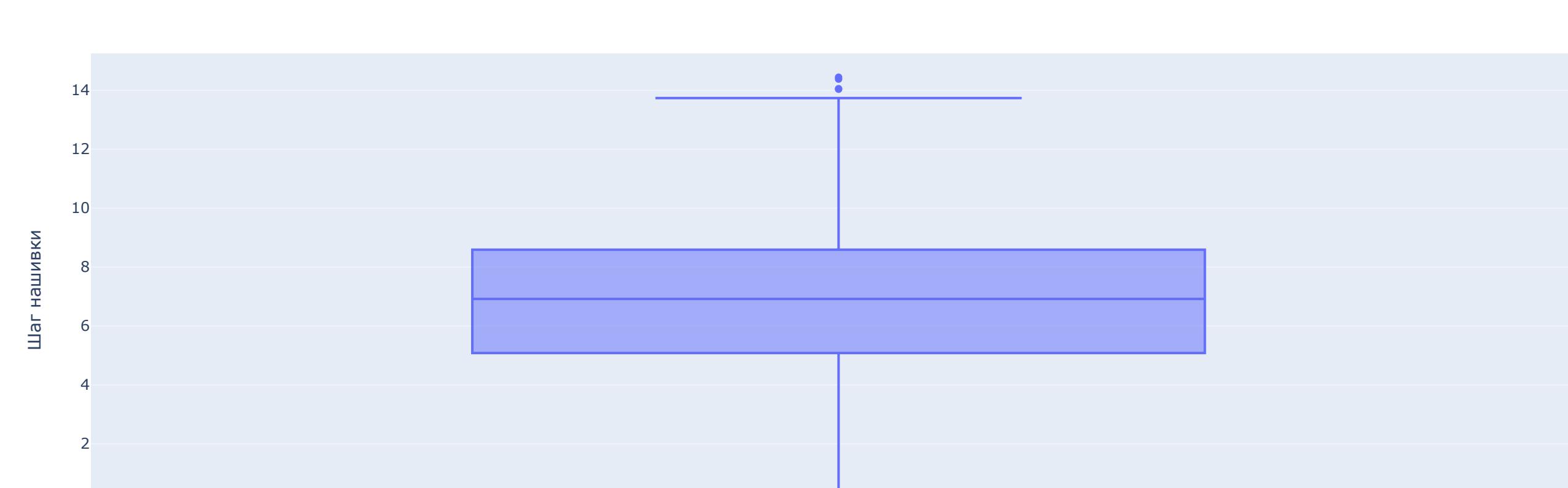
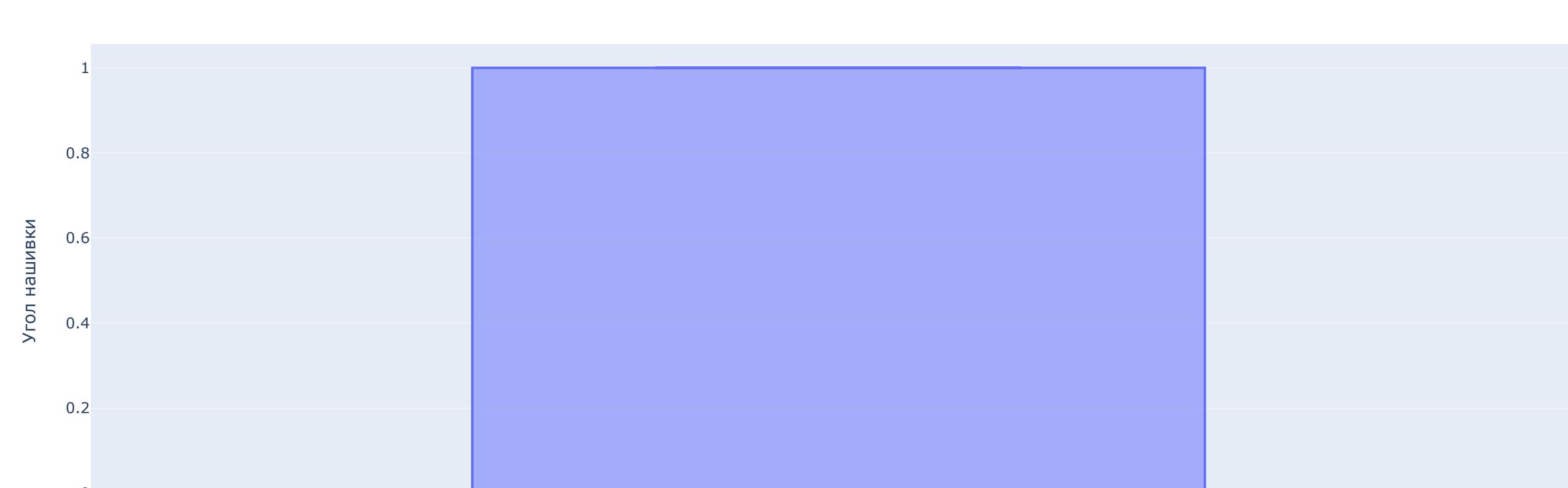
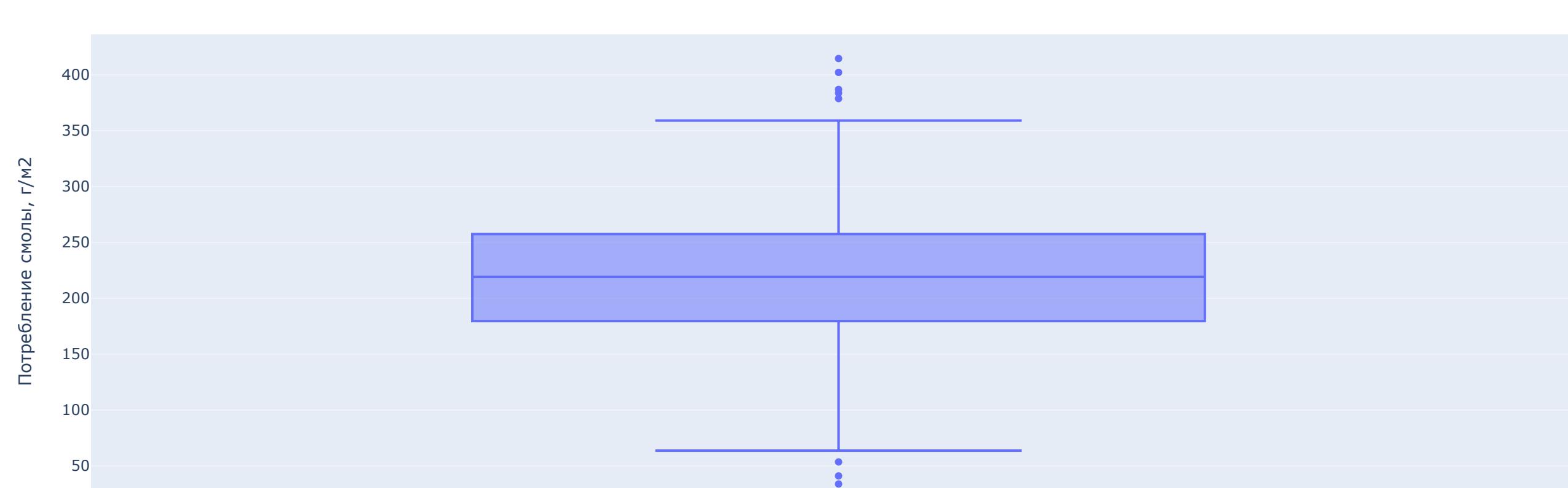
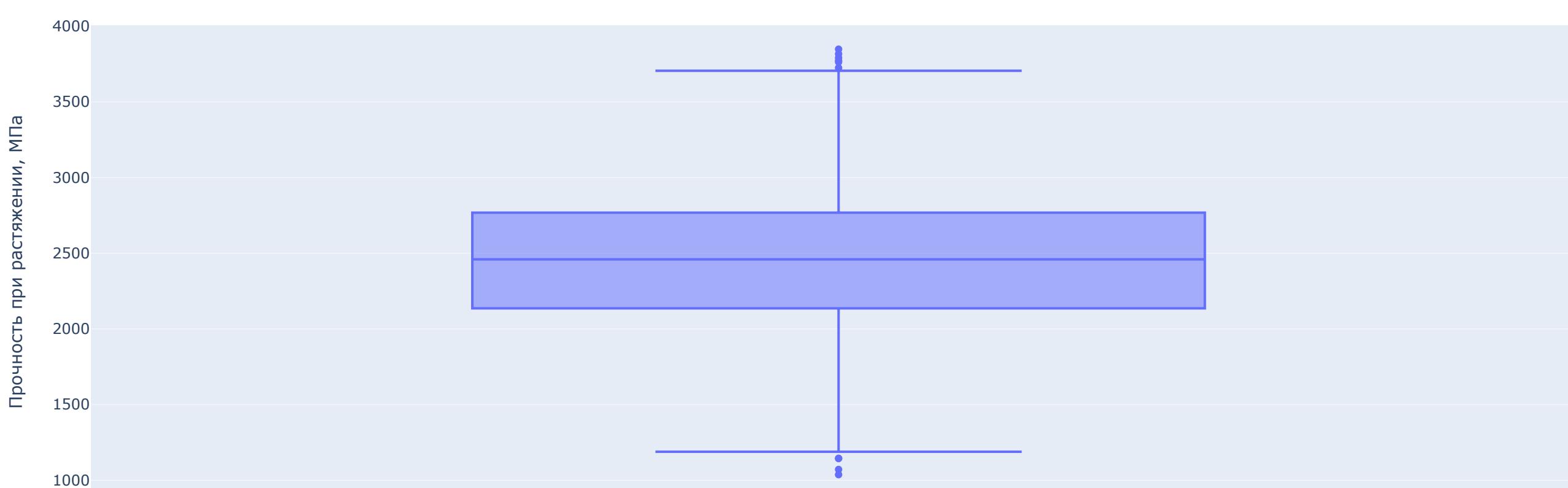
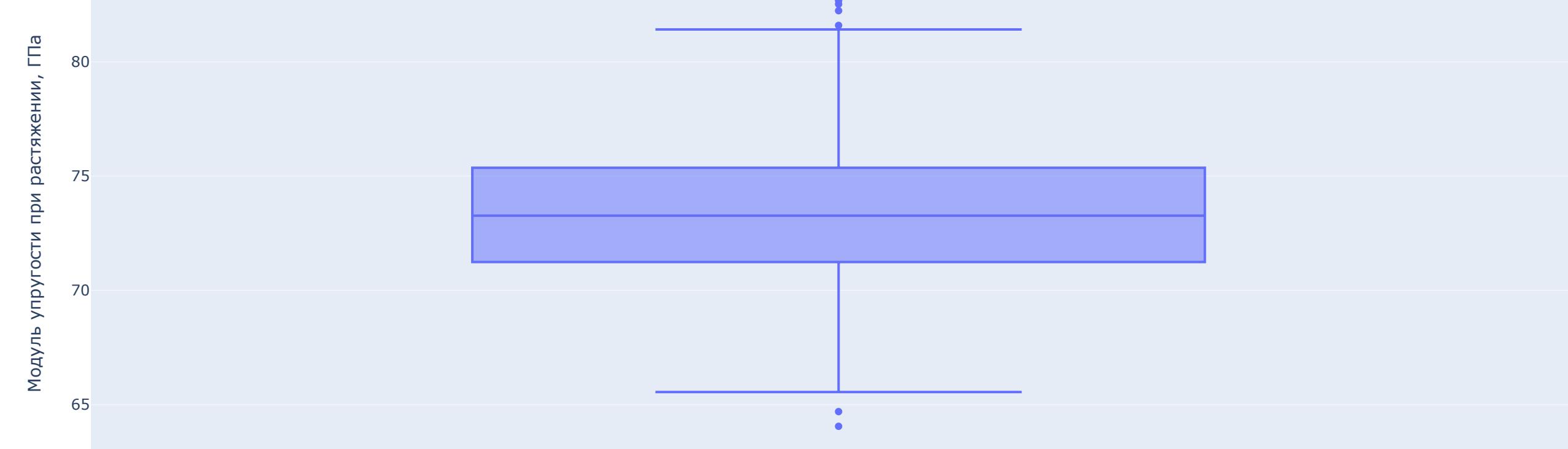


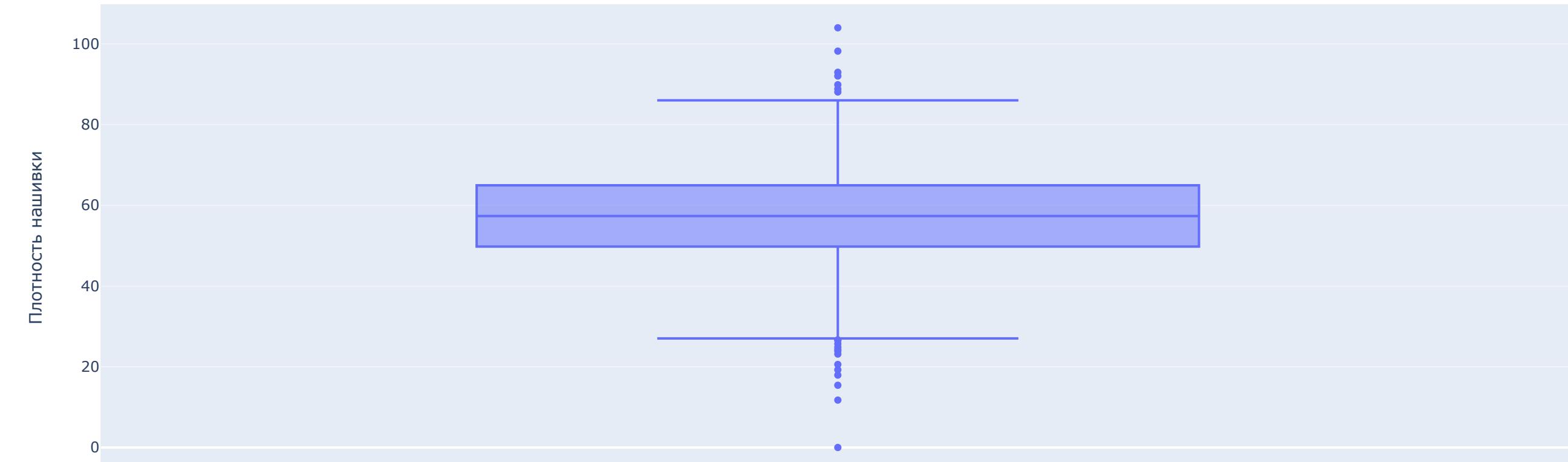


```
In [42]: for column in df.columns:  
    fig = px.box(df, y = column)  
    fig.show()
```

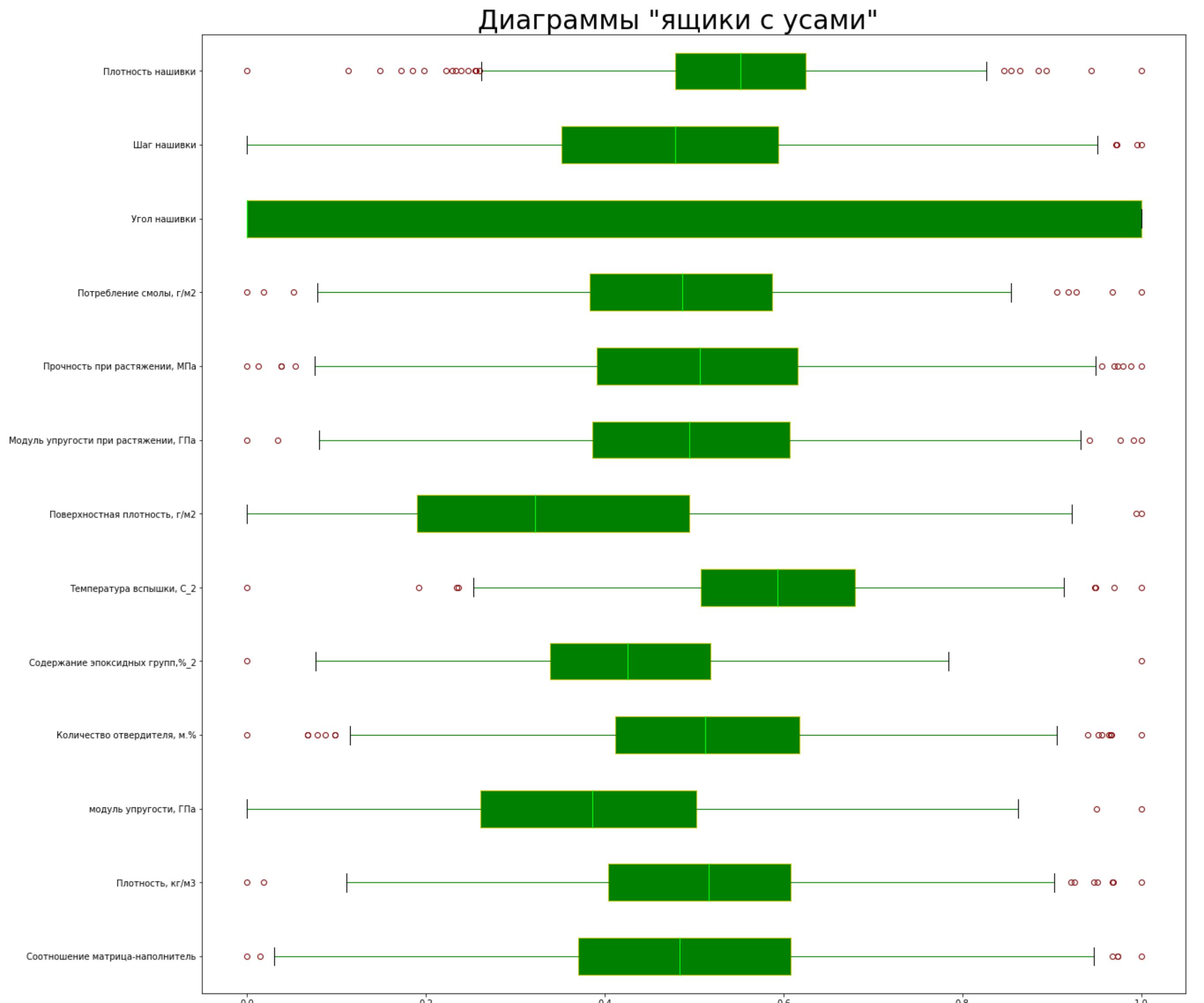




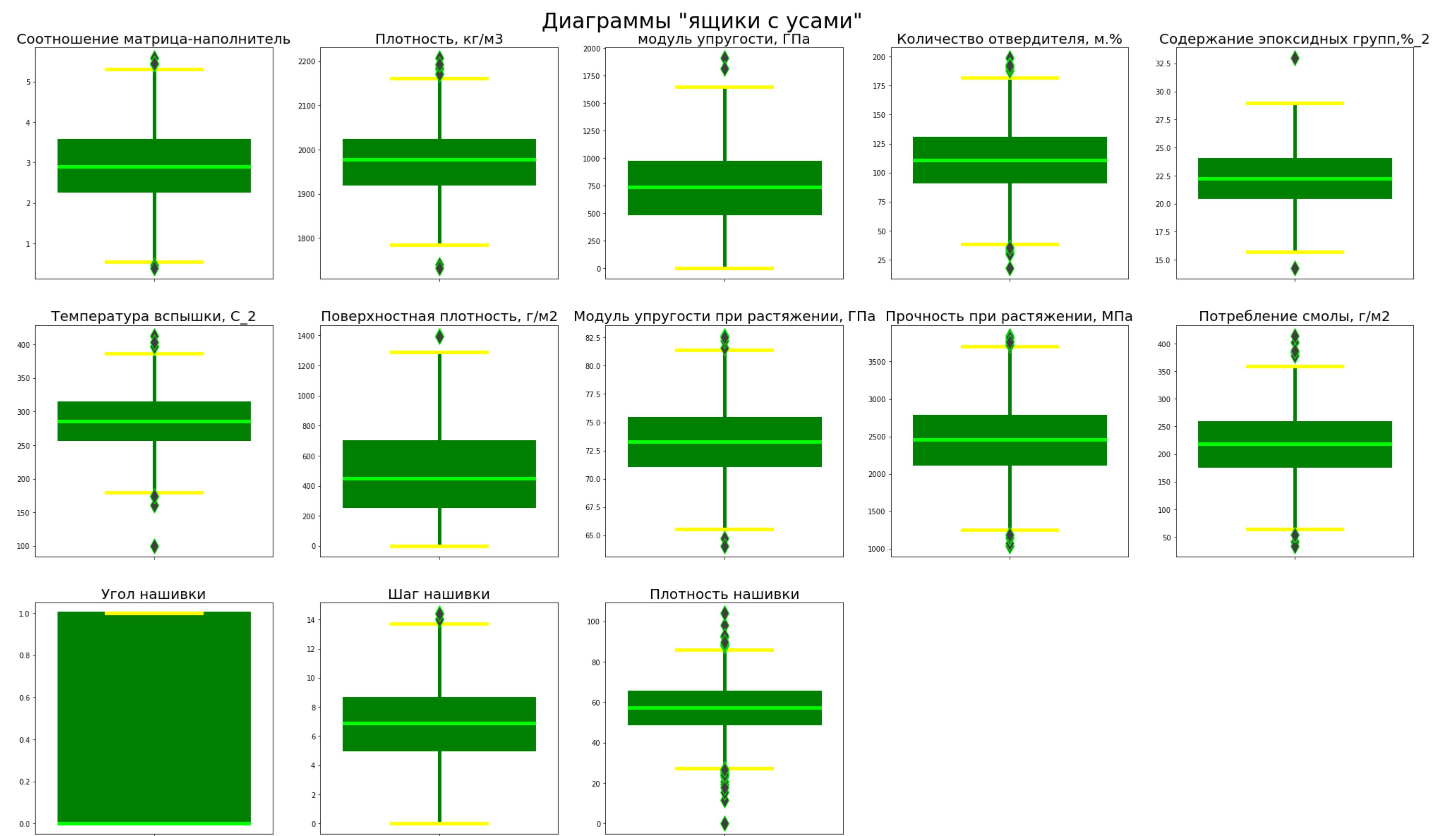




```
In [43]: # "Ящики с усами" (первый вариант)
scaler = MinMaxScaler()
scaler.fit(df)
plt.figure(figsize=(20, 20))
plt.suptitle("Диаграммы \"ящики с усами\"", y=0.9, fontsize=30)
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels=df.columns, patch_artist=True, meanline=True, vert=False, boxprops=dict(facecolor='g', color='y'), medianprops=dict(color='lime'), whiskerprops=dict(color='g'), capprops=dict(color='black'), flierprops=dict(color='y', markeredgecolor='maroon'))
plt.show()
```



```
In [44]: # Ящики с усами (второй вариант)
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.boxplot(data=df, y=df[col], figsize=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.xlabel('')
    plt.title(col, size=20)
    plt.show()
    c += 1
# "ящики с усами" показывают наличие выбросов во всех столбцах, кроме угла нащивки, значит, с ними будем работать
```

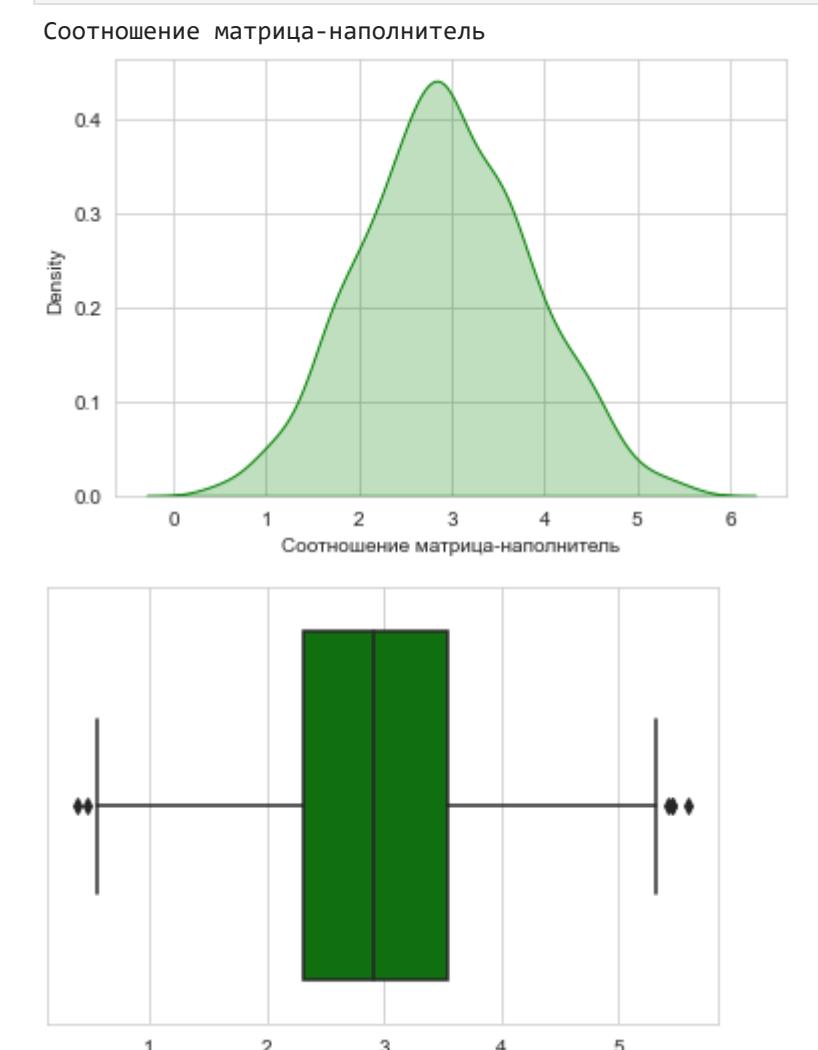


```
In [45]: # Гистограмма распределения и диаграмма "ящик с усами" вместе с данными по каждому столбцу
for column_name in column_names:
    print(column_name)

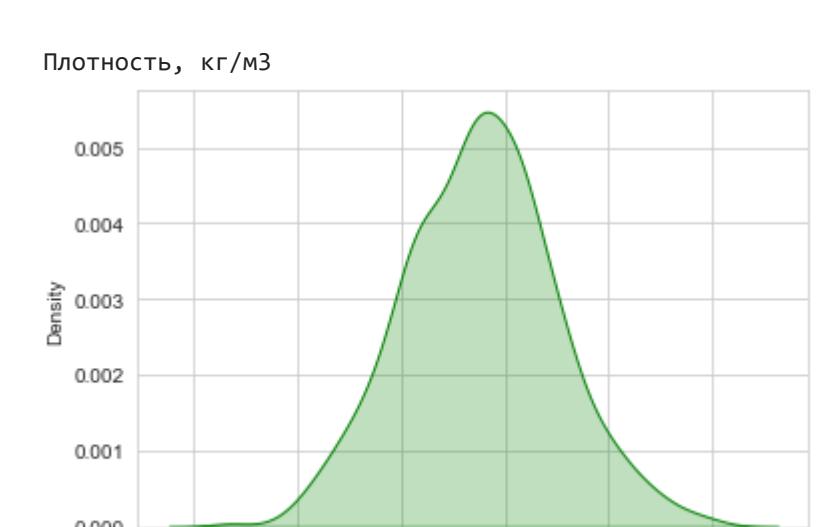
# Гистограмма распределения
g1s = df[column_name]
sns.set_style("whitegrid")
sns.kdeplot(data = g1s, shade = True, palette = "colorblind", color = "g")
plt.show()

# Диаграмма "ящик с усами"
sns.boxplot(x=g1s, color = "g");
plt.show()

# Значения (min max)
print("Минимальное значение: ", end = " ")
print(np.min(g1s))
print("Максимальное значение: ", end = " ")
print(np.max(g1s))
print("Среднее значение: ", end = " ")
print(np.mean(g1s))
print("Медианное значение: ", end = " ")
print(np.median(g1s))
print("\n\n")
# Кроме "угол нашивки, град" и "Поверхностная плотность, г/м²" остальные переменные относительно хорошо соответствуют нормальному распределению
```

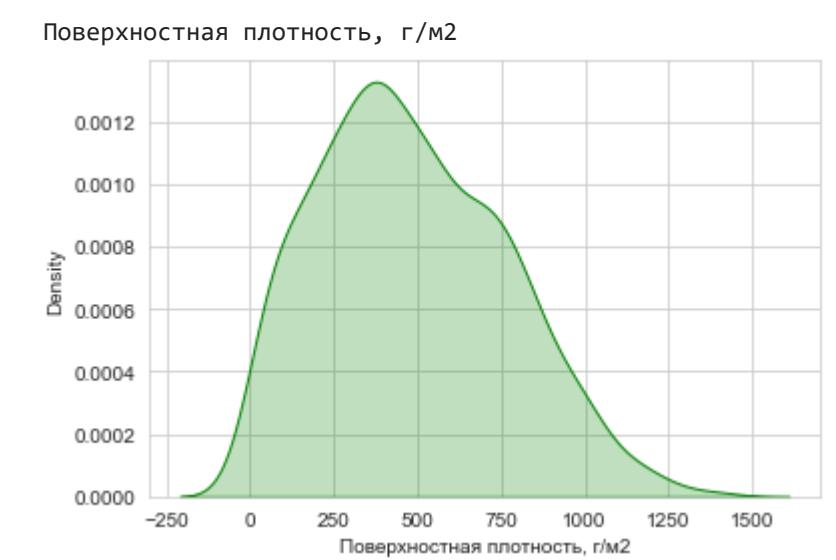
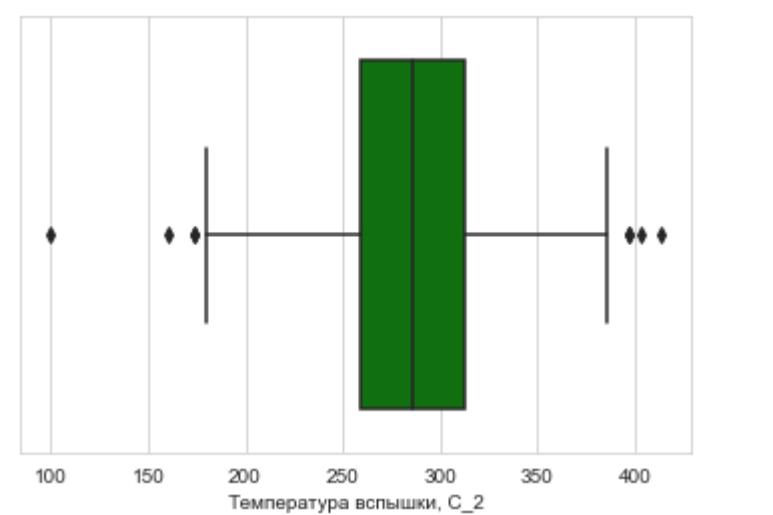
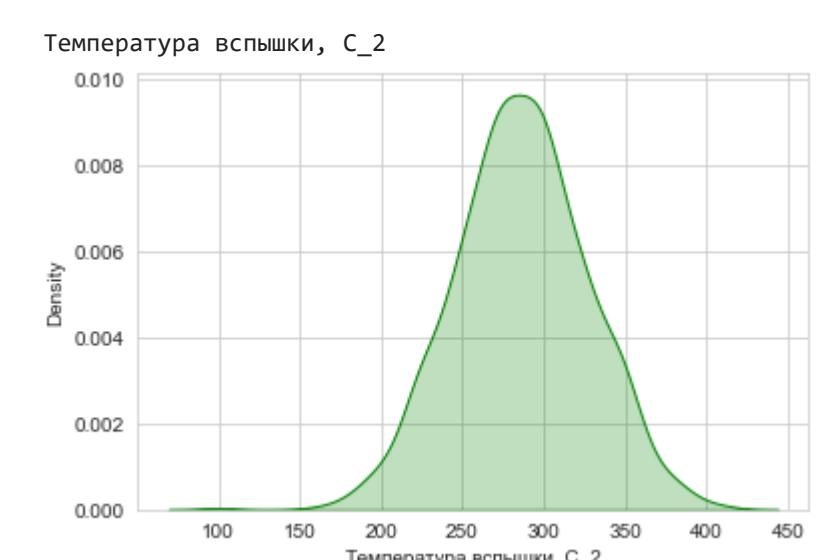
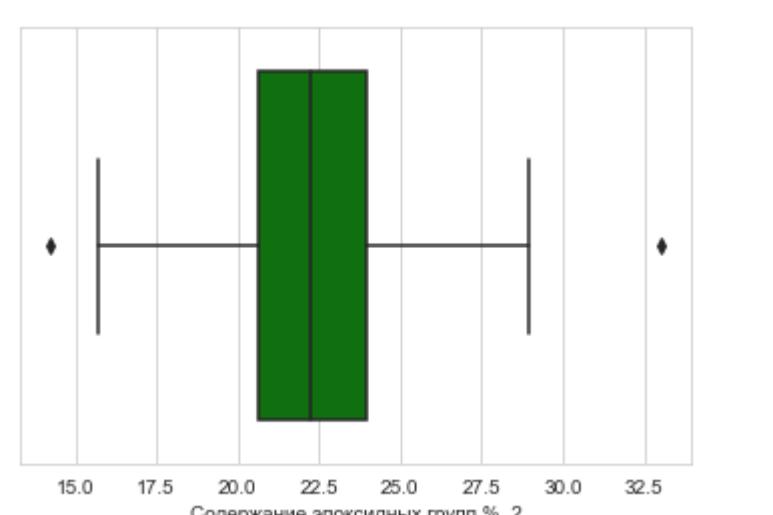
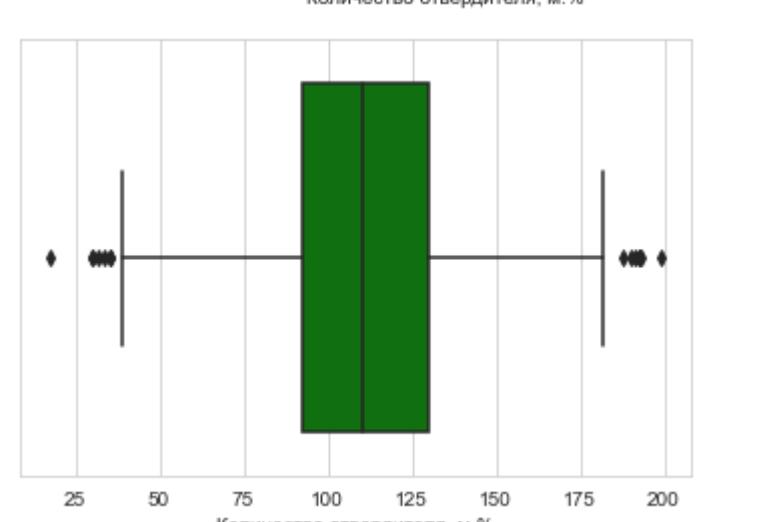
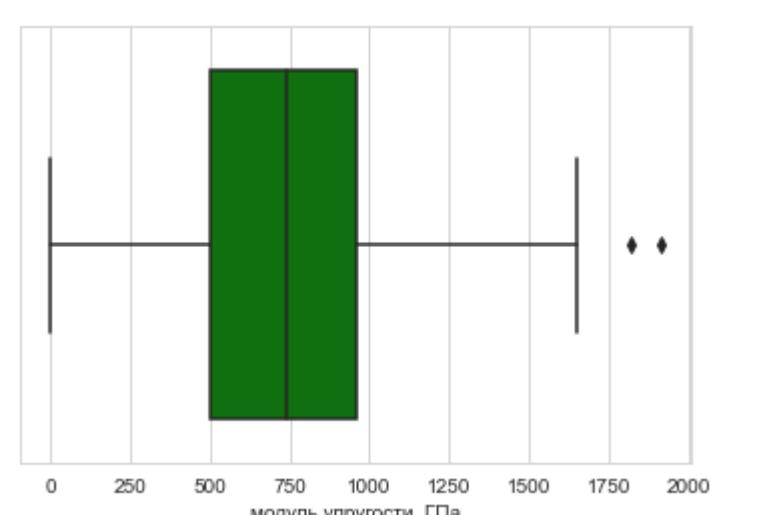
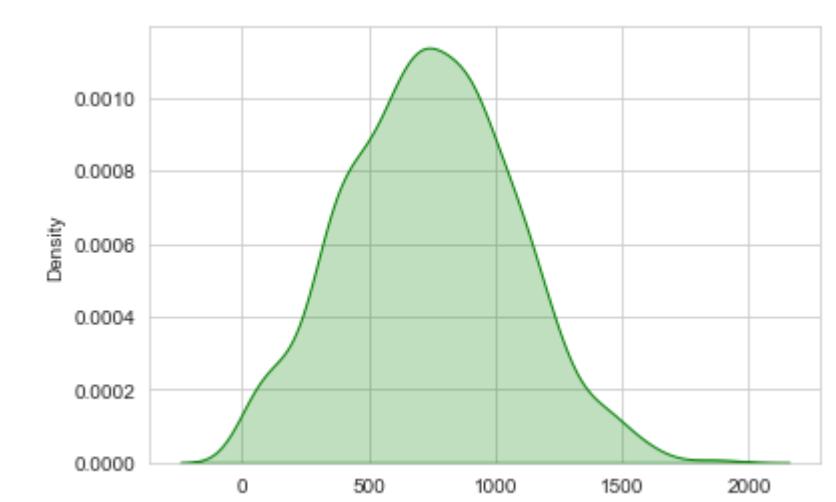


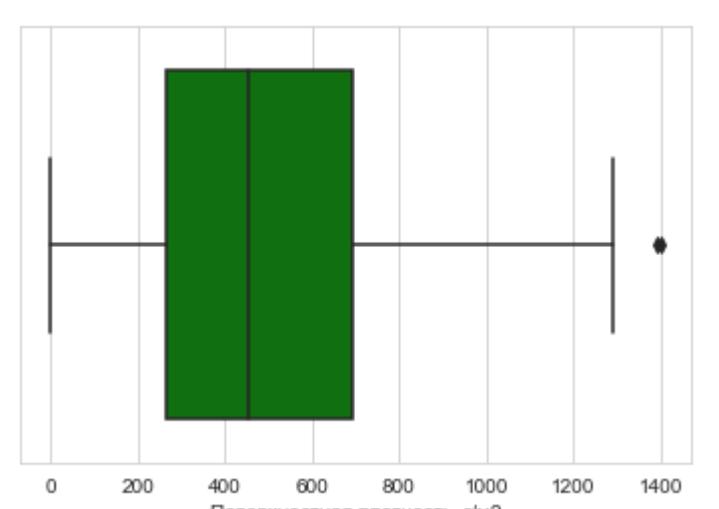
Минимальное значение: 0.398409589179414
Максимальное значение: 5.591741515860754
Среднее значение: 2.9383657334325586
Медианное значение: 2.90687765033521



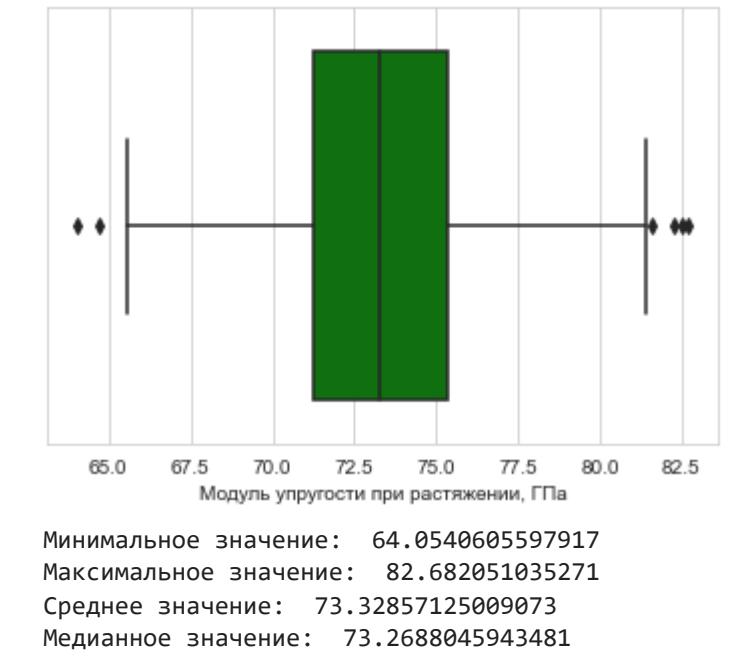
Минимальное значение: 1727.773488611119
Максимальное значение: 2307.773488611119
Среднее значение: 1975.7348881101548
Медианное значение: 1977.62165679058

Модуль упругости, ГПа

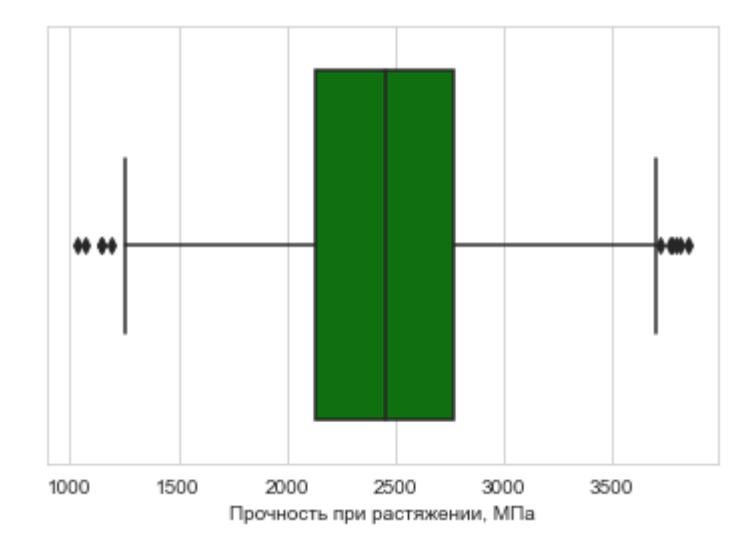




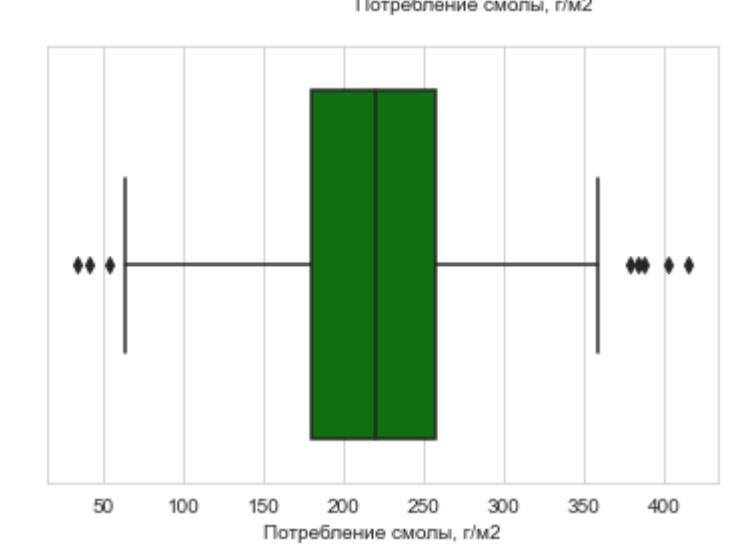
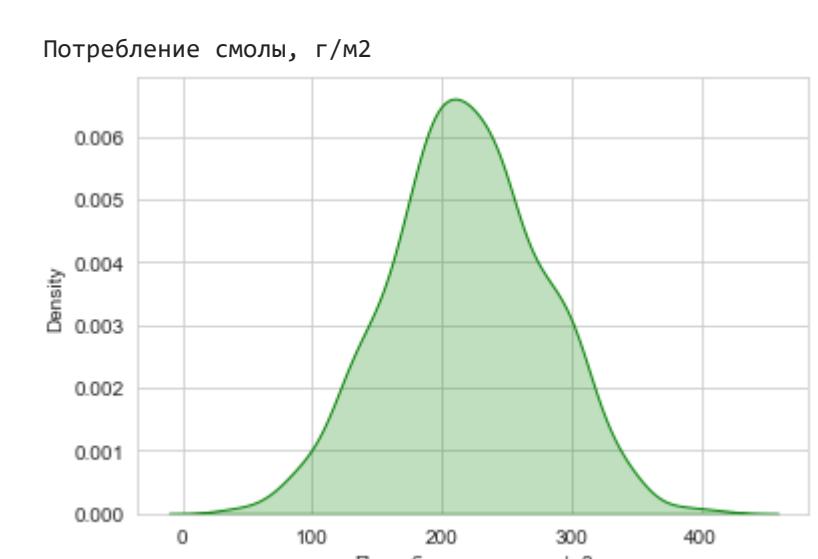
Минимальное значение: 0.603739925153949
Максимальное значение: 1399.54236233989
Среднее значение: 482.718138484181
Медианное значение: 451.86436518306



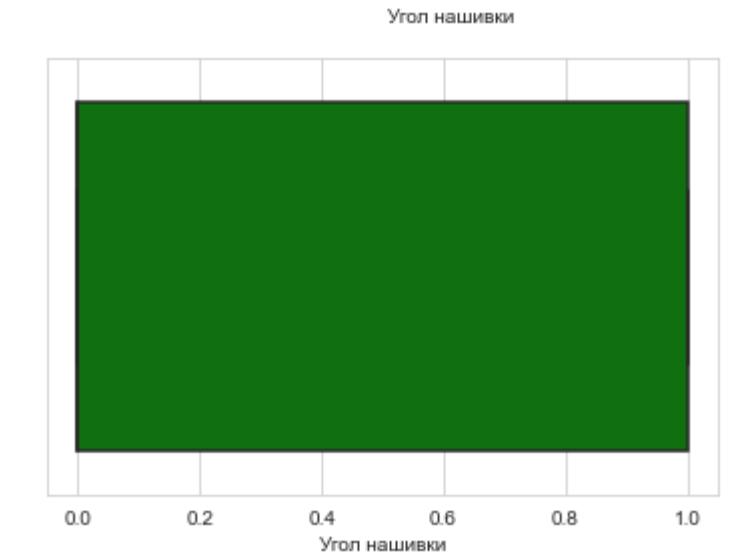
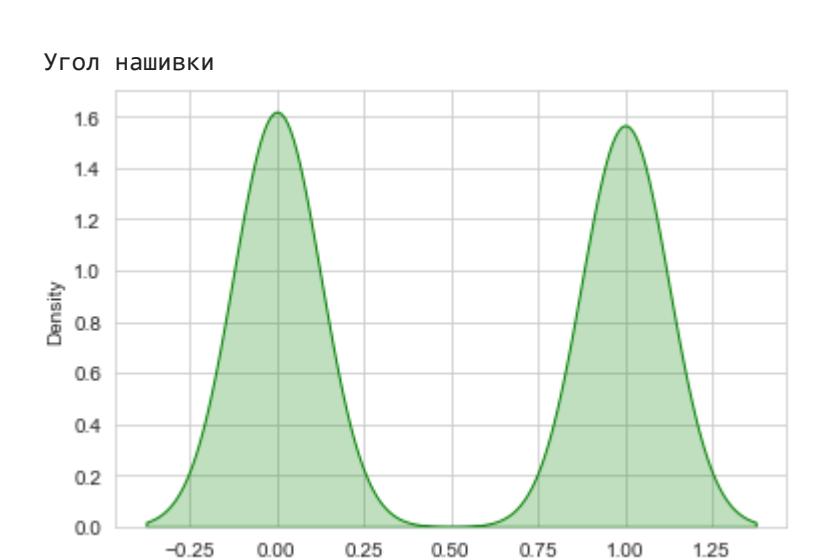
Минимальное значение: 64.054065597917
Максимальное значение: 82.682051353271
Среднее значение: 73.32857125809073
Медианное значение: 73.2888643543481



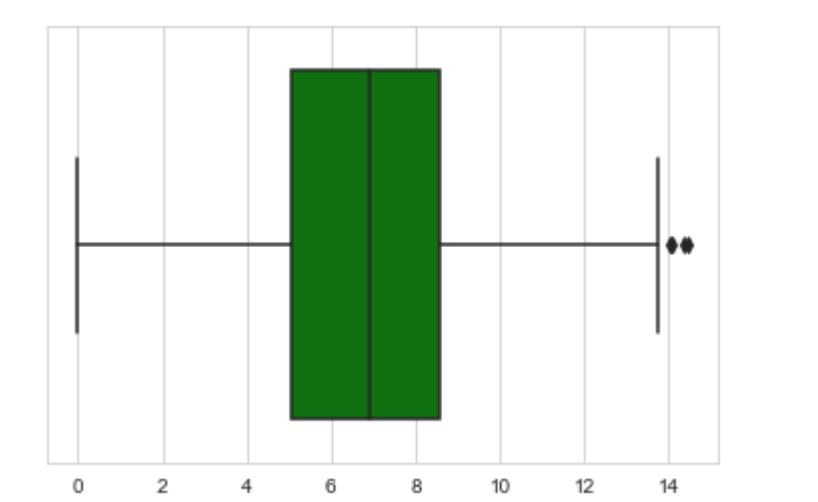
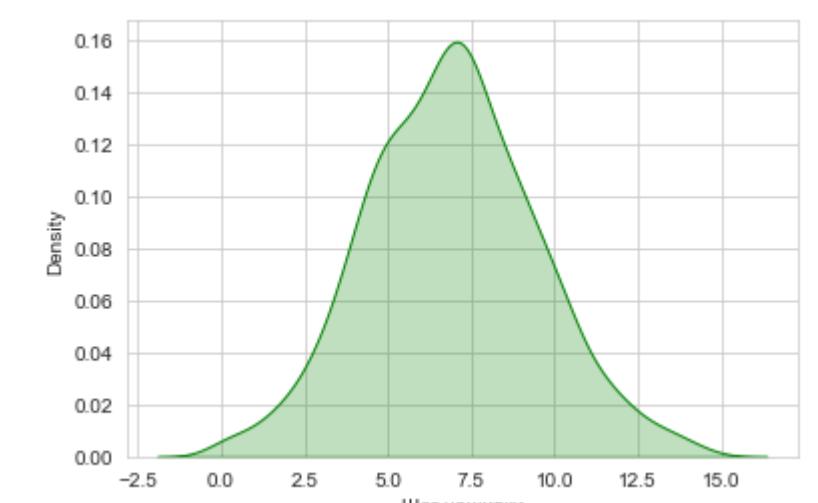
Минимальное значение: 1036.85669835
Максимальное значение: 3848.43673187618
Среднее значение: 2466.9228426979025
Медианное значение: 2459.52452680389



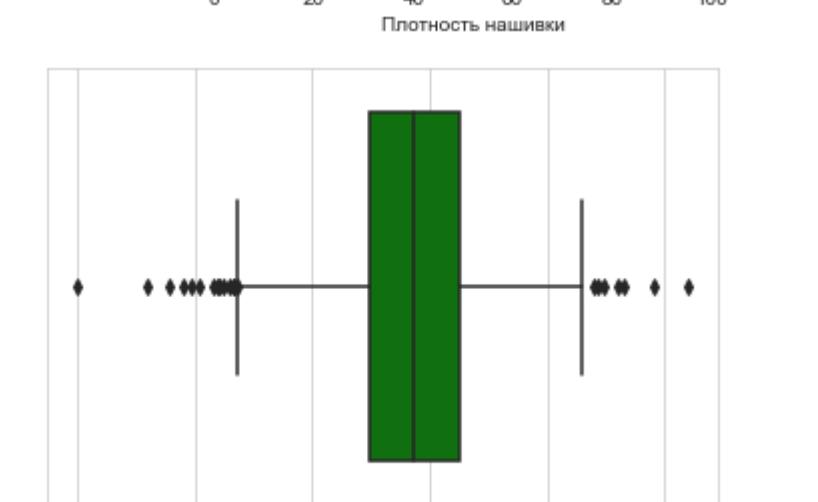
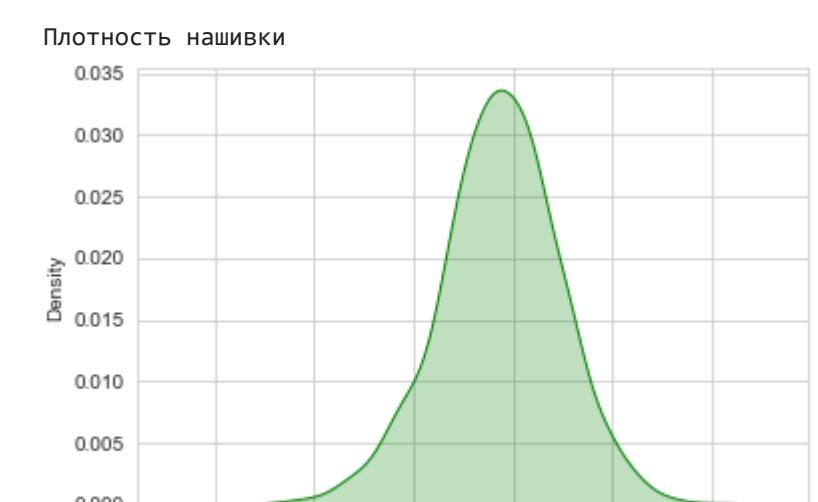
Минимальное значение: 33.893055329625
Максимальное значение: 414.59628361534
Среднее значение: 218.42314367654265
Медианное значение: 219.198882195134



Минимальное значение: 0
Максимальное значение: 1
Среднее значение: 0.4916911045943384
Медианное значение: 0.8



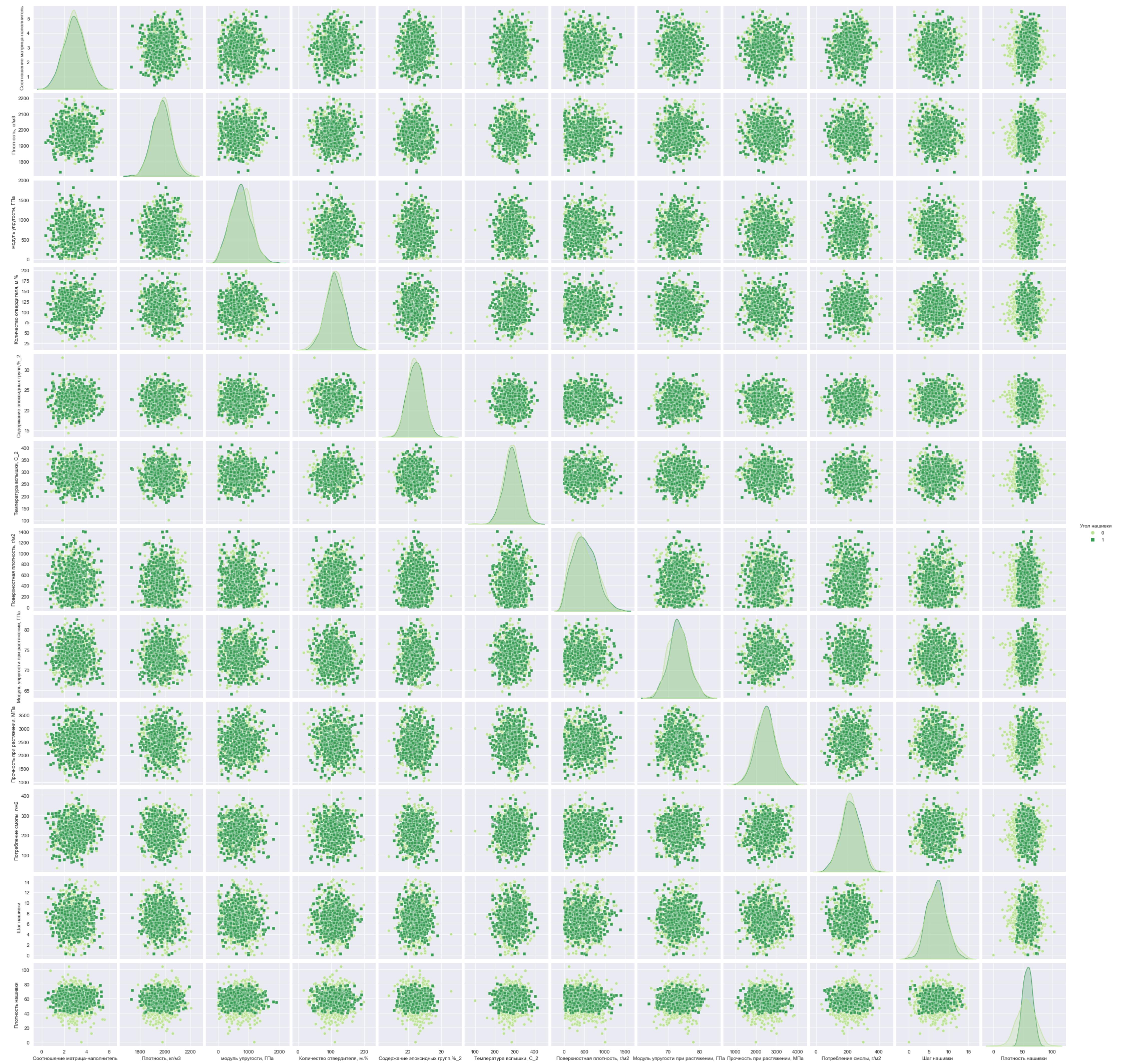
```
Минимальное значение: 0.0
Максимальное значение: 14.4495218753969
Среднее значение: 6.899222077675824
Медианное значение: 6.9161438559491
```



```
Минимальное значение: 0.0
Максимальное значение: 103.988901301494
Среднее значение: 57.15392543285763
Медианное значение: 57.34191984669929
```

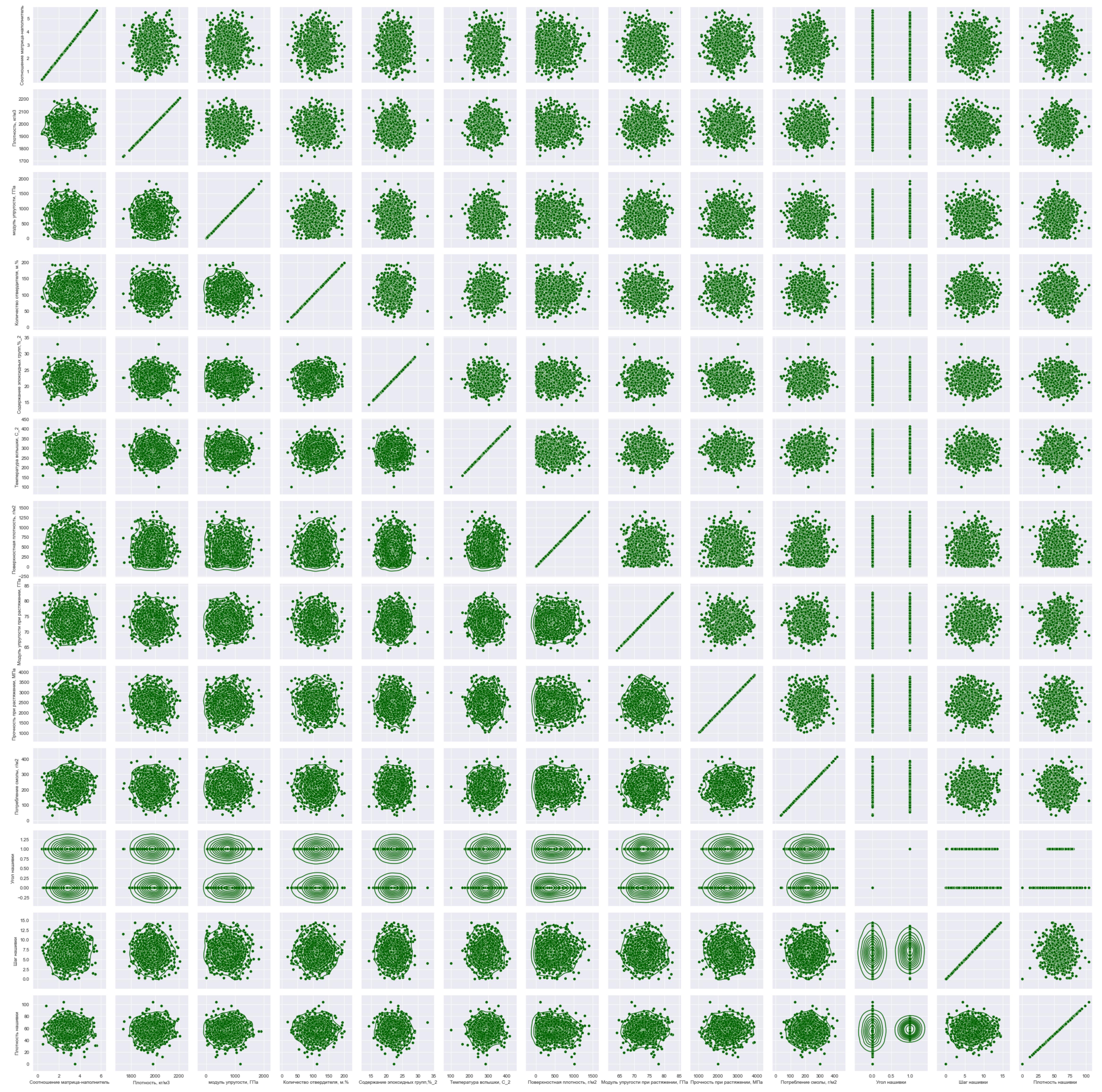
```
In [46]: # Построение распределения точек (матрица диаграмм рассеяния) (первый вариант)
sns.set_style('darkgrid')
sns.pairplot(df, hue = "Угол навигки", markers = ["o", "s"], diag_kind = 'auto', palette='YIGn')
# Попарные графики рассеяния точек так же не показывают какую-либо зависимость между данными. Зависимость между показателями не линейная, взаимосвязь отсутствует, необходимо использовать несколько показателей.
# из графиков можно наблюдать выбросы, потому что некоторые точки расположены далеко от общего облака
# Отсутствие линейной корреляции наборами подтверждается при построении регрессии?
```

```
Out[46]: <seaborn.axisgrid.PairGrid at 0x1dc2e419b50>
```

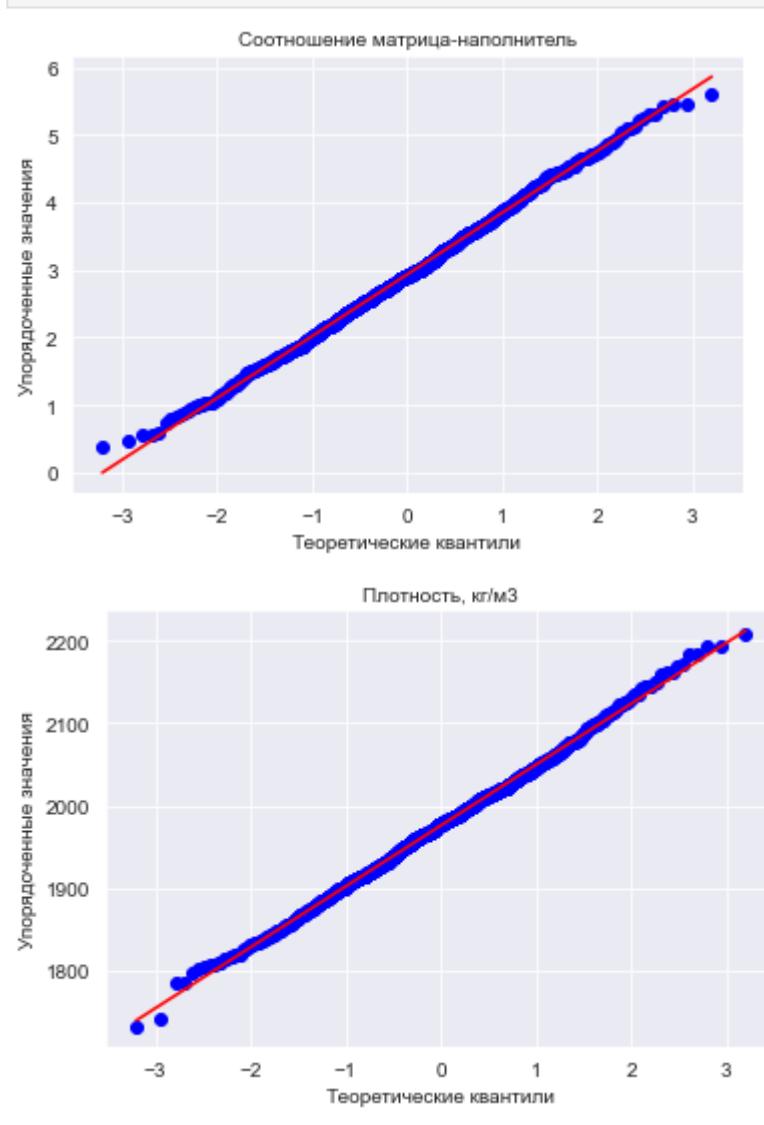


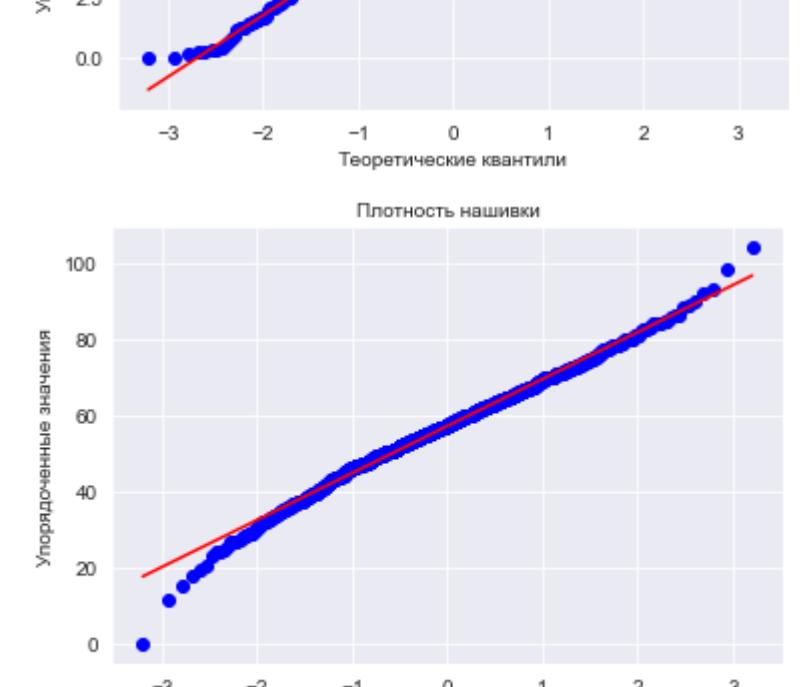
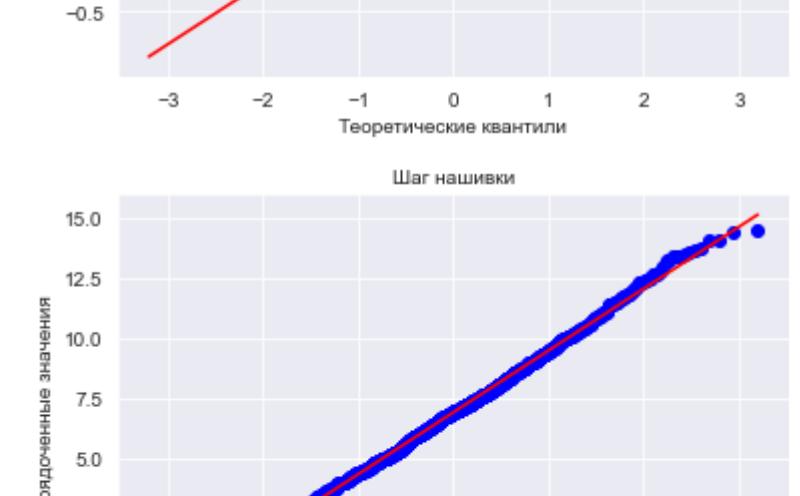
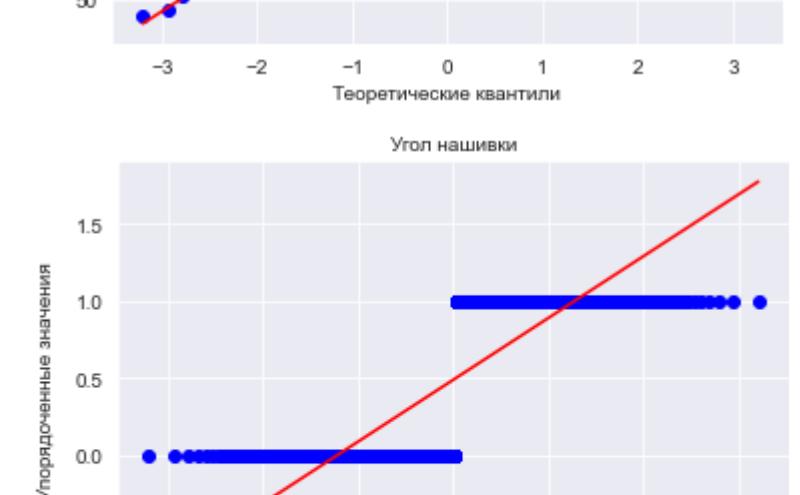
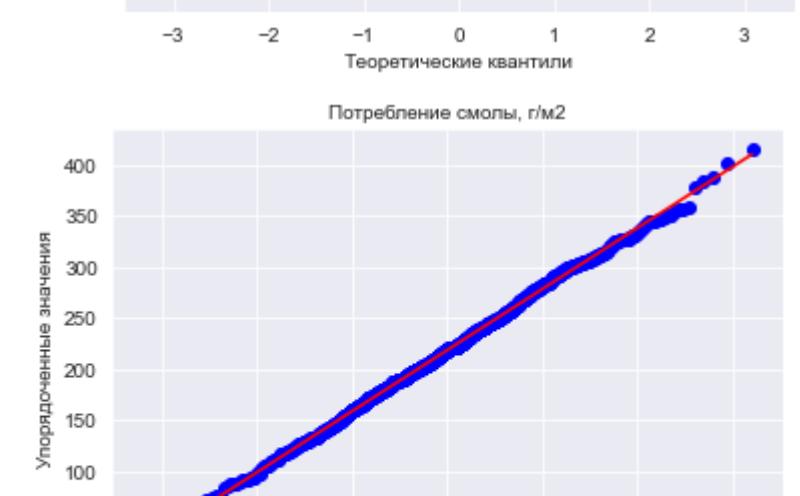
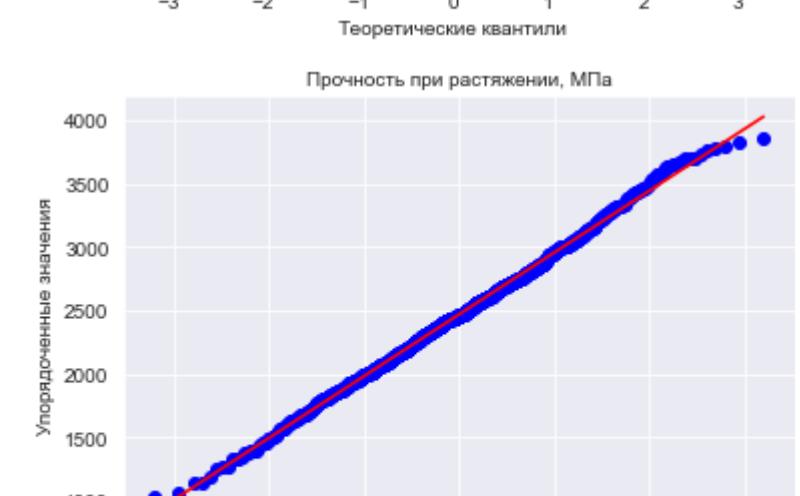
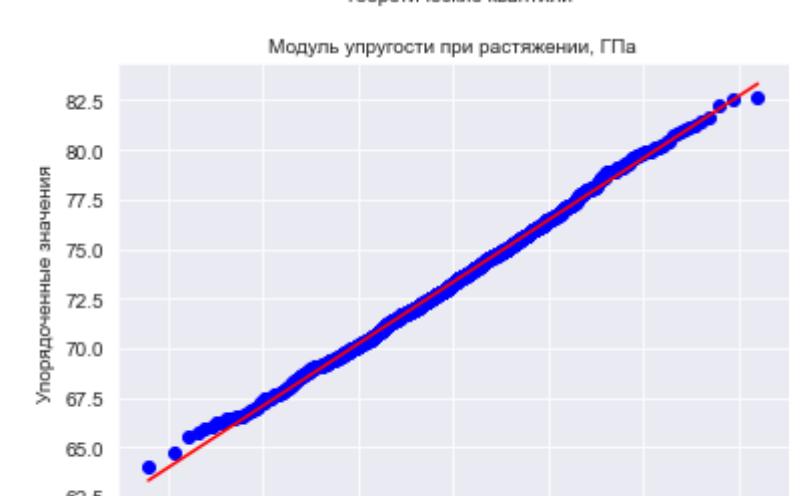
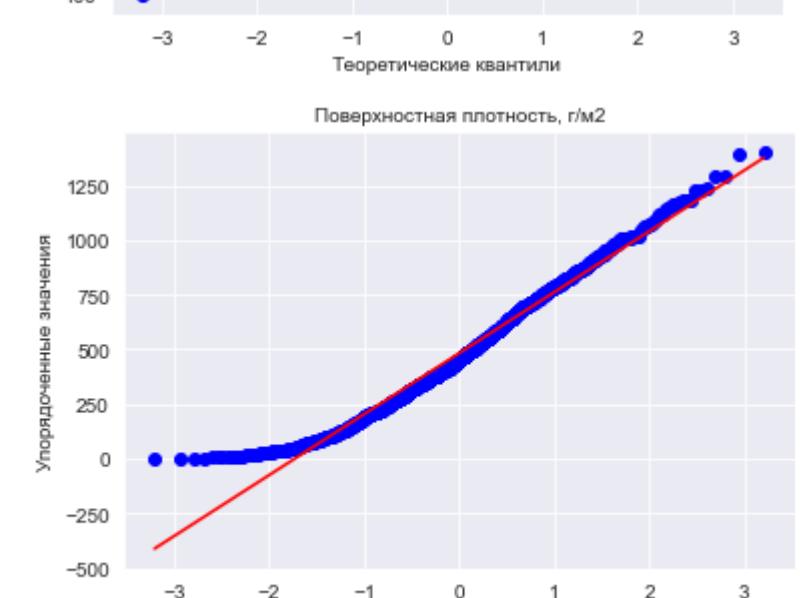
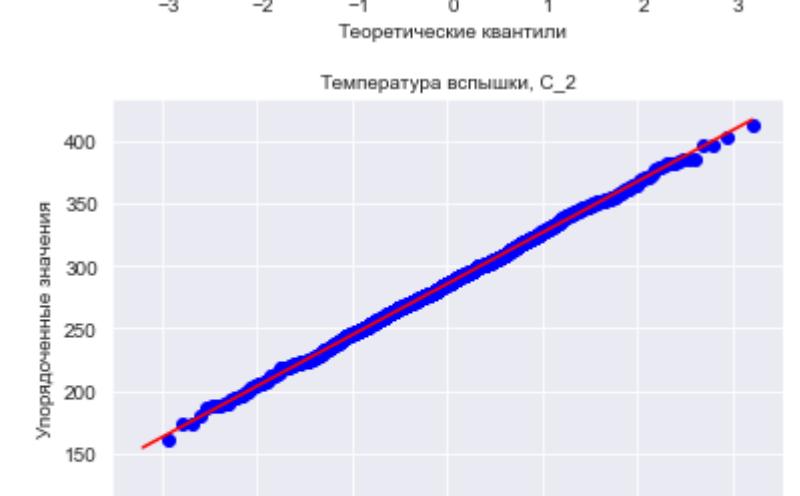
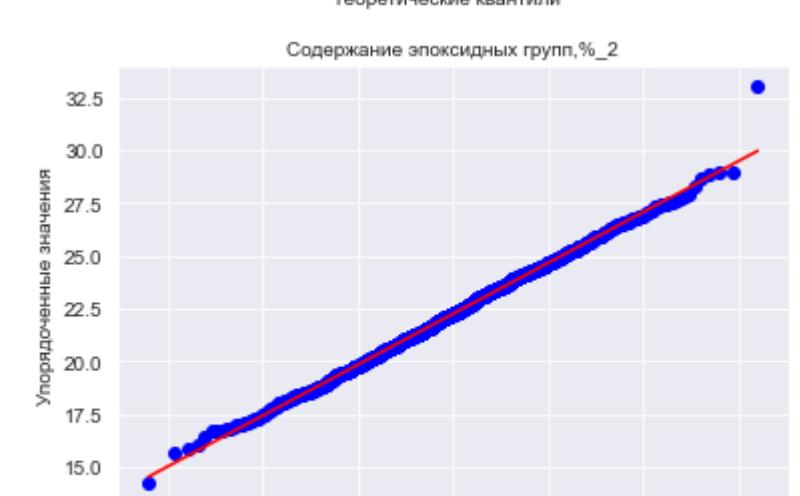
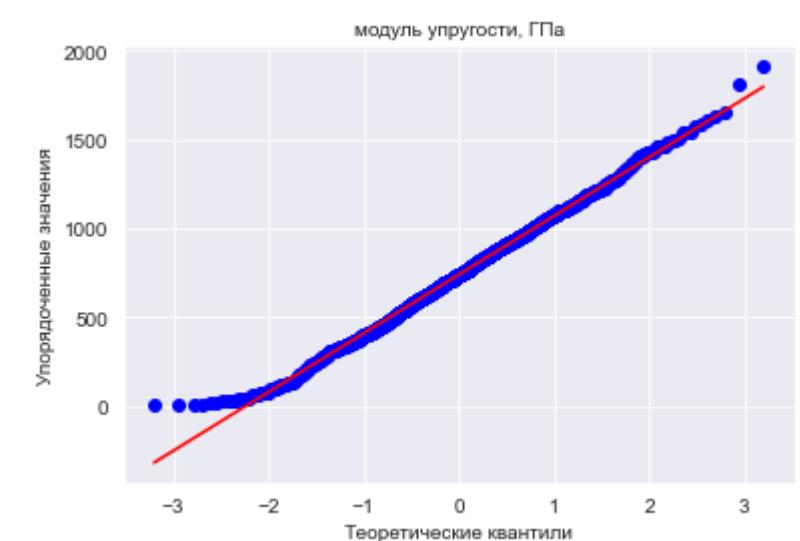
```
In [47]: # Проверим, сколько в ячейках - скatterплоты (бледой бархат)
g = sns.PairGrid(df[df.columns])
g.map_upper(sns.scatterplot, color = 'darkgreen')
g.map_lower(sns.kdeplot, color = 'darkgreen')
g.map_diag(sns.kdeplot, color = 'darkgreen')
plt.show()
# Корреляции нам
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

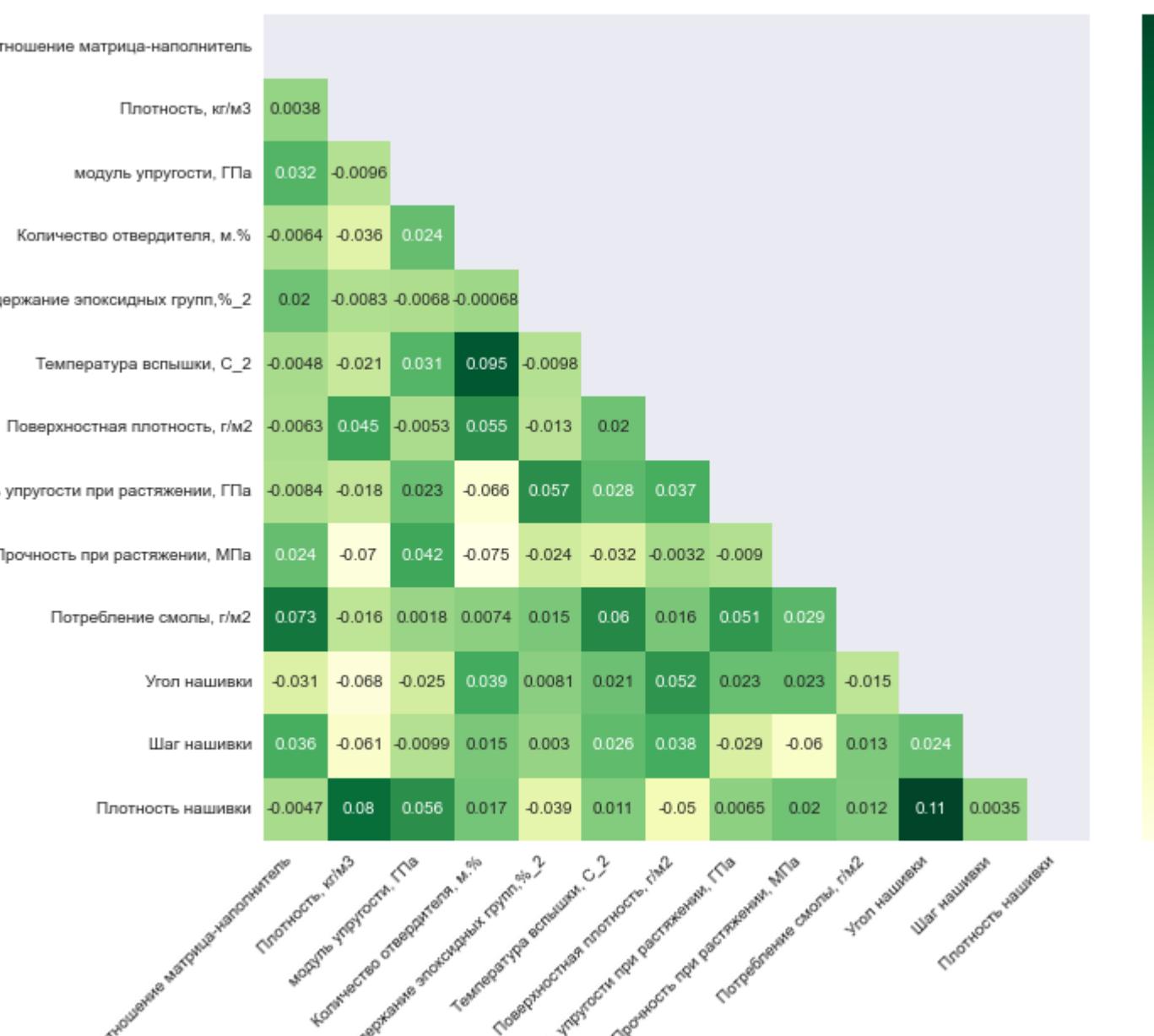


```
In [48]: # график qq
for i in df.columns:
    plt.figure(figsize = (6, 4))
    res = stats.probplot(df[i], plot = plt)
    plt.title(i, fontsize = 10)
    plt.xlabel('Теоретические квантили')
    plt.ylabel('Упорядоченные значения')
    plt.show()
```





```
# Создаем пустое для отображения большого графика
f, ax = plt.subplots(figsize = (11, 9))
# Визуализируем данные корреляции и сделаем цветобаю палинту
sns.heatmap(df.corr(), mask = mask, annot = True, square = True, cmap = 'YlGn')
plt.suptitle('Максимальная корреляция между наивысшими показателями')
plt.show()
# Корреляция между всеми параметрами очень близка к 0, что говорит об отсутствии зависимости между этими данными.
```



In [50]: # График корреляции подтверждает данные теории композитных материалов. Мы видим, что на качество материала влияет температура вспышки и количество отвердителя с матрицей и наполнителем под влиянием температуры. Угол наивысши и плотность наивысши несомненно оказывают влияние на свойства материала. А потребление смолы и соотношение матрицы-наполнителя, плотности и плотности наивысши, модуль упругости и плотности наивысши имеют не особенно выраженную корреляцию.

Выход на данном этапе работы: На наших "сырых" данных мы наблюдаем выбросы в каждом столбце, кроме столбца "Угол наивысши" и корреляция входных переменных очень слабая.

Проведем предобработку данных

```
#Вспомнили, что пропуски в данных сплошные. Значит, сразу приступаем к работе с выбросами
df = df.fillna(0)
print("Удалено выбросов. Посчитаем, сколько значений у нас в каждом столбце выбывающих из распределения.")

#Т.к. значений не очень много, их вполне можно исключить.
```

```
Out[51]: Соотношение матрица-наполнитель : 0
Плотность, кг/м³ : 0
модуль упругости, ГПа : 0
Количество отвердителя, м.% : 0
Содержание эпоксидных групп,%_2 : 0
Температура вспышки, °C_2 : 0
Поверхностная плотность, г/м² : 0
Модуль упругости при растяжении, ГПа : 0
Прочность при растяжении, МПа : 0
Потребление смолы, г/м² : 0
угол наивысши : 0
шаг наивысши : 0
плотность наивысши : 0
старт: int64
```

In [52]: #Для удаления выбросов существует 2 основных метода - метод 3-х сигм и межквартильных расстояний. Сравним эти 2 метода.

```
def method_3s(df):
    метод_3s = 0
    count_3s = [] # Список, куда записывается количество выбросов по каждой колонке датабрейна методом.
    count_3s.append(df['3s'].sum())
    print('Метод 3-х сигм, выбросов:', метод_3s)

for column in df:
    if column != '3s':
        # метод 3-х Сигм
        zscore = (df[column] - df[column].mean()) / df[column].std()
        з3s = np.abs(zscore) >= 3
        метод_3s = df[~з3s].sum()
        count_3s.append(df[~з3s].sum())
        print(column,'3s:', ' ', df['3s'].sum())

# метод 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
df['iq'] = (df[column] <= lower) | (df[column] >= upper)
метод_iq = df['iq'].sum()
count_iq.append(df['iq'].sum())
print('Метод 3-х сигм, выбросов:', метод_3s)
print('Метод межквартильных расстояний, выбросов:', метод_iq)
```

Соотношение матрица-наполнитель: 0

Плотность, кг/м³: 0

модуль упругости, ГПа: 0

Количество отвердителя, м.%: 0

Содержание эпоксидных групп,%_2: 0

Температура вспышки, °C_2: 0

Поверхностная плотность, г/м²: 0

Модуль упругости при растяжении, ГПа: 0

Прочность при растяжении, МПа: 0

Потребление смолы, г/м²: 0

угол наивысши: 0

шаг наивысши: 0

плотность наивысши: 0

старт: 0

Метод 3-х сигм, выбросов: 24

Метод межквартильных расстояний, выбросов: 93

In [53]: # С целью предотвращения удаления особенностью признака или допущения ошибки, посчитаем распределение выбросов по каждому столбцу.

```
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 {m.sum().sum()} выбросов')
```

0 выбросов в признаке матрица-наполнитель

2 выбросов в признаке Плотность

2 выбросов в признаке модуль упругости, ГПа

2 выбросов в признаке Количество отвердителя, м.%

2 выбросов в признаке Содержание эпоксидных групп,%_2

2 выбросов в признаке Температура вспышки, °C

2 выбросов в признаке Поверхностная плотность, г/м²

0 выбросов в признаке Модуль упругости при растяжении, ГПа

3 выбросов в признаке Прочность при растяжении, МПа

0 выбросов в признаке Угол наивысши

7 выбросов в признаке Плотность наивысши

Всего 24 выброса

In [55]: # Результаты показывают, что выбросы распределены по разным признакам. Какого-то скопления выбросов в одном из признаков не обнаружено. Поэтому можно удалять выбросы, потому что глобальных изменений в зависимостях они не дают.

```
#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, °C",
"Поверхностная плотность, г/м²",
"Модуль упругости при растяжении, ГПа",
"Прочность при растяжении, МПа",
"Потребление смолы, г/м²",
"шаг наивысши",
"плотность наивысши"]
```

In [57]: # Исключим выбросы, очистим данные от выбросов методом межквартильного рассстояния (далее 1,5 межквартильных размахов)
Выше сделано вручную, это не методом, потому что хотим добиться в данной работе полностью избавления от выбросов, и метод межквартильного рассстояния позволяет удалить практически 10% датасета сразу

```
for i in column_list_drop:
    q75, q25 = np.percentile(df.loc[:, i], [75, 25])
    инт_3s = q75 + 1.5 * инт_3s
    max_3s = q75 + (1.5 * инт_3s)
    min_3s = q25 - (1.5 * инт_3s)
    df.loc[df[i] < min_3s, i] = np.nan
    df.loc[df[i] > max_3s, i] = np.nan
```

In [58]: # Так же для удаления выбросов можно использовать этот формулу, но мне приличней вариант выше:

```
df = df[(df>(min_3s)) & (df<(max_3s))].dropna(1)
```

In [59]: # Можно так же удалить выбросы методом 3-х сигм, но мы не будем. Просто оставим код здесь на будущее

```
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
```

Out[59]: (1023, 13)

```
#Посмотрим на сумму выбросов по каждому из столбцов
df.isnull().sum()
```

Многие столбцы можно их удалить.

```
Out[60]: Соотношение матрица-наполнитель      6
Плотность, кг/м3        9
Модуль упругости, ГПа     2
Количество отвердителя, м.%       14
Содержание эпоксидных групп,%_2      2
Температура вспышки, С_2          2
Поверхностная плотность, г/м2        2
Модуль упругости при растяжении, ГПа   6
Прочность при растяжении, МПа       11
Потребление смолы, г/м2           8
Угол нашивки                   0
Шаг нашивки                     4
Плотность нашивки                 21
dtype: int64
```

In [61]: #Удаляем строки с выбросами

```
df = df.dropna(axis=0)
```

In [62]: #И еще раз посмотрим на сумму выбросов по каждому из столбцов, чтобы убедиться, что все работает

```
df.isnull().sum()
```

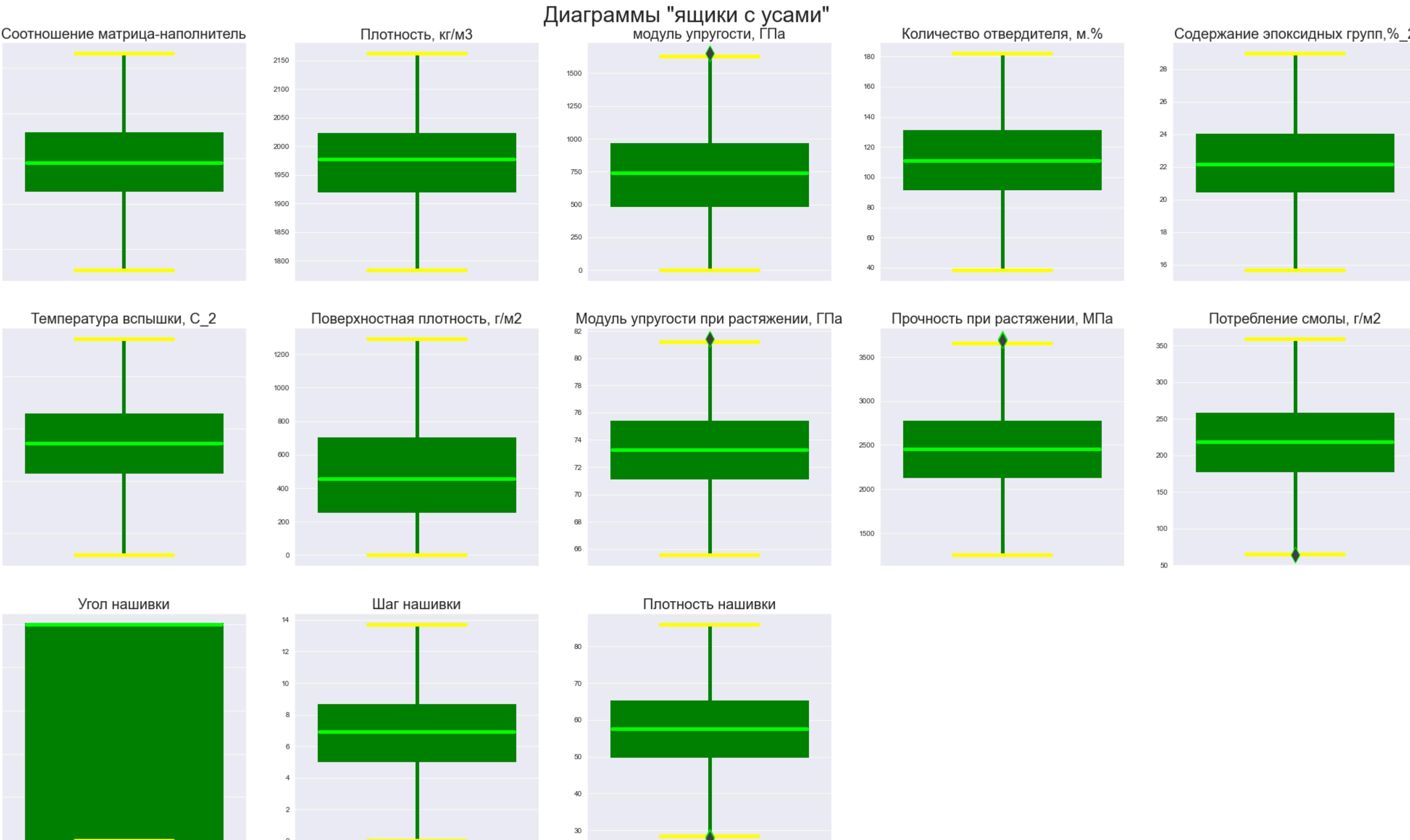
```
Out[62]: Соотношение матрица-наполнитель      0
Плотность, кг/м3        0
Модуль упругости, ГПа     0
Количество отвердителя, м.%       0
Содержание эпоксидных групп,%_2      0
Температура вспышки, С_2          0
Поверхностная плотность, г/м2        0
Модуль упругости при растяжении, ГПа   0
Прочность при растяжении, МПа       0
Потребление смолы, г/м2           0
Угол нашивки                   0
Шаг нашивки                     0
Плотность нашивки                 0
dtype: int64
```

In [63]: #Проанализируя информацию о "числом" датафрейме после удаления пропусков. Видим, что строк стало меньше
df.info()

```
# После удаления выбросов в датафрейме осталось 936 строк и 13 колонок
<class 'pandas.core.frame.DataFrame'>
Int64Index: 936 entries, 1 to 1022
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
 --- 
 0   Соотношение матрица-наполнитель    936 non-null   float64
 1   Плотность, кг/м3                  936 non-null   float64
 2   Модуль упругости, ГПа            936 non-null   float64
 3   Количество отвердителя, м.%       936 non-null   float64
 4   Содержание эпоксидных групп,%_2    936 non-null   float64
 5   Температура вспышки, С_2          936 non-null   float64
 6   Поверхностная плотность, г/м2        936 non-null   float64
 7   Модуль упругости при растяжении, ГПа 936 non-null   float64
 8   Прочность при растяжении, МПа       936 non-null   float64
 9   Потребление смолы, г/м2           936 non-null   float64
 10  Угол нашивки                   936 non-null   int32
 11  Шаг нашивки                     936 non-null   float64
 12  Плотность нашивки                 936 non-null   float64
dtypes: float64(12), int32(1)
memory usage: 98.7 KB
```

In [64]: # Яшки с усами (Внедрение ящиков с усами)
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # количество ящиков plot counter

```
plt.figure(figsize = (35,35))
plt.suptitle('Диаграммы "ящики с усами"', y = 0.9 , fontweight = 'bold', fontsize = 30)
for col in df.columns:
    plt.subplot(a, b, c)
    plt.title(col, size = 20)
    sns.boxplot(data = df, y = df[col], fliersize = 15, linewidth = 5, boxprops = dict(facecolor = 'y', color = 'lime'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = 'g'), capprops = dict(color = 'yellow'), flierprops = dict(color = 'y', markeredgecolor = 'lime'))
    c += 1
```



In [65]: #Посмотрим на "ящики с усами" и наблюдаем все еще наличие выбросов

```
scaler = StandardScaler()
```

```
scaler.fit(df)
```

```
plt.figure(figsize = (20, 20))
```

#Посмотрим на "ящики 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 = 'g', color = 'y'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = 'g'), capprops = dict(color = 'black'), flierprops = dict(color = 'y', markeredgecolor = 'maroon'))
```

```
plt.show()
```



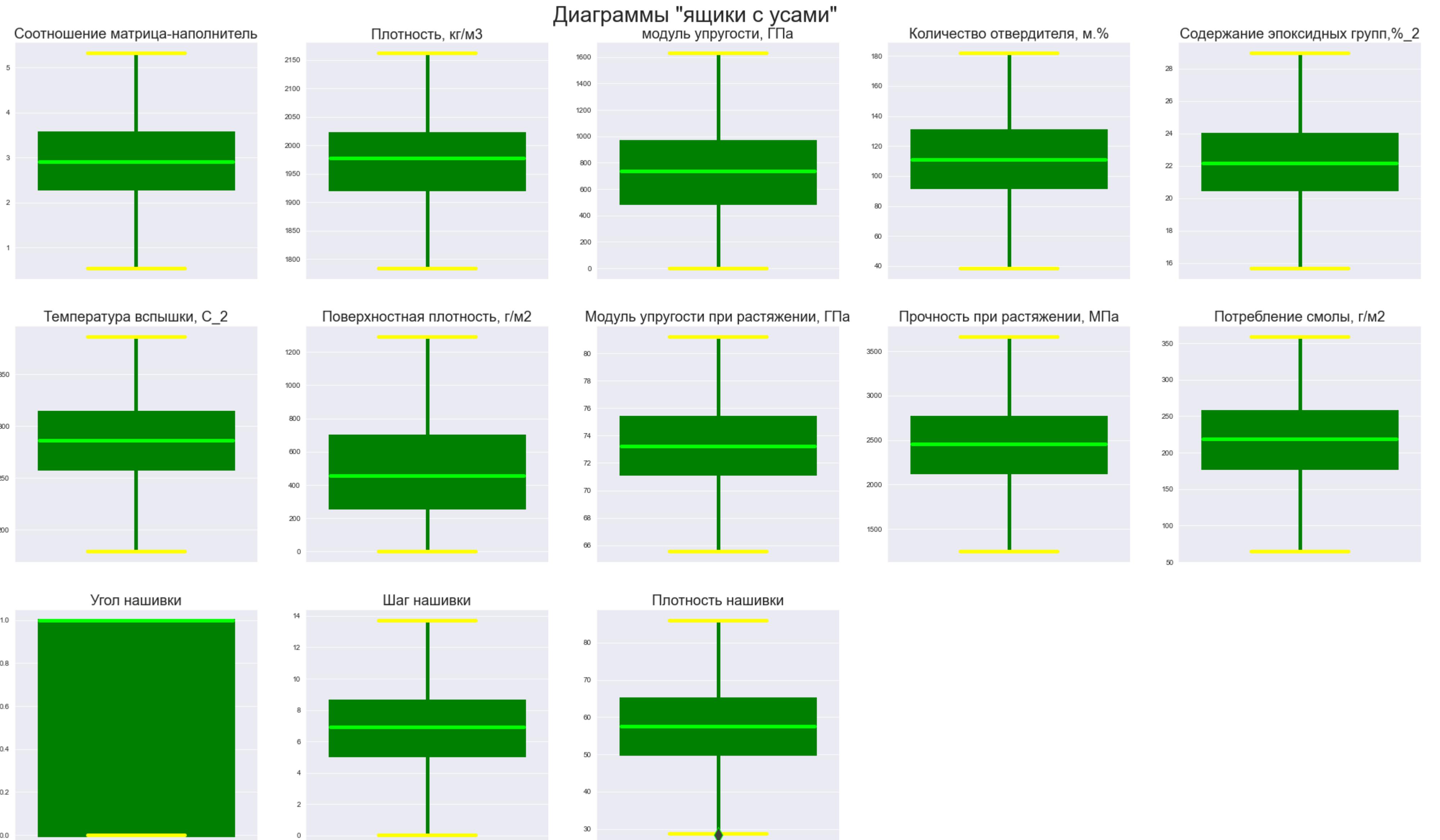
In [66]: # На графиках выше мы видим выбросы в некоторых столбцах. Они всё ещё есть, поэтому повторяю удаление выбросов

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

In [67]: # Видим с усами (Видори Баранов)

```
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,
            fontweight='bold')
for col in df.columns:
    plt.subplot(a, b, c)
    #plt.figure(figsize=(7,5))
    #boxplot(data = df, y = df[col], flierprops = dict(facecolor = 'y', color = 'g'), medianprops = dict(color = 'lime'), whiskerprops = dict(color = "g"), capprops = dict(color="yellow"), filerprops = dict(color = "y", markeredgecolor = "lime"))
    plt.boxplot(df[col], showfliers=True)
    plt.title(col, size = 20)
    plt.show()
    c += 1
```



In [68]: df.skew()# каскадия по всем колонкам. Никак не использовал в дальнейшем.

```
Соотношение матрица-наполнитель      0.058917
Плотность, кг/м3                      0.002822
Модуль упругости при растяжении, ГПа   0.070520
Количество отвердителя, м.%             0.128059
Содержание эпоксидных групп,%_2        0.027975
Температура вспышки, С_2                0.007521
Поверхностная плотность, г/м2           0.257398
Модуль упругости при растяжении, ГПа   0.197138
Прочность при растяжении, МПа          0.063861
Потребление смолы, г/м2                 0.026155
Угол нашивки                            0.847497
Шаг нашивки                             0.935338
Плотность нашивки                      0.047335
dtype: float64
```

In [69]: df.kurt()# эксцесс по всем колонкам. Никак не использовал в дальнейшем

```

Out[69]: Соотношение матрица-наполнитель -0.388987
Плотность, кг/м3 -0.219527
Модуль упругости, ГПа -0.37935
Количество отвердителя, м.% -0.349075
Содержание эпоксидных групп,%_2 0.318182
Температура вспышки, С_2 -0.393724
Поверхностная плотность, г/м2 -0.548925
Модуль упругости при растяжении, ГПа -0.308785
Прочность при растяжении, МПа -0.217998
Поглощение смолы, г/м2 -0.393795
Угол наклона 2.08264
Шаг нашивки -0.114436
Плотность нашивки -0.193872
dtype: float64

In [70]: #проверим сумму выбросов по каждому из столбцов
df.isnull().sum()

Out[70]: Соотношение матрица-наполнитель 0
Плотность, кг/м3 0
Модуль упругости, ГПа 1
Количество отвердителя, м.% 0
Содержание эпоксидных групп,%_2 0
Температура вспышки, С_2 0
Поверхностная плотность, г/м2 0
Модуль упругости при растяжении, ГПа 1
Прочность при растяжении, МПа 4
Поглощение смолы, г/м2 1
Угол наклона 0
Шаг нашивки 0
Плотность нашивки 0
dtype: int64

In [71]: #и снова удалаем строки, которые содержат выбросы
df = df.dropna(axis=0)

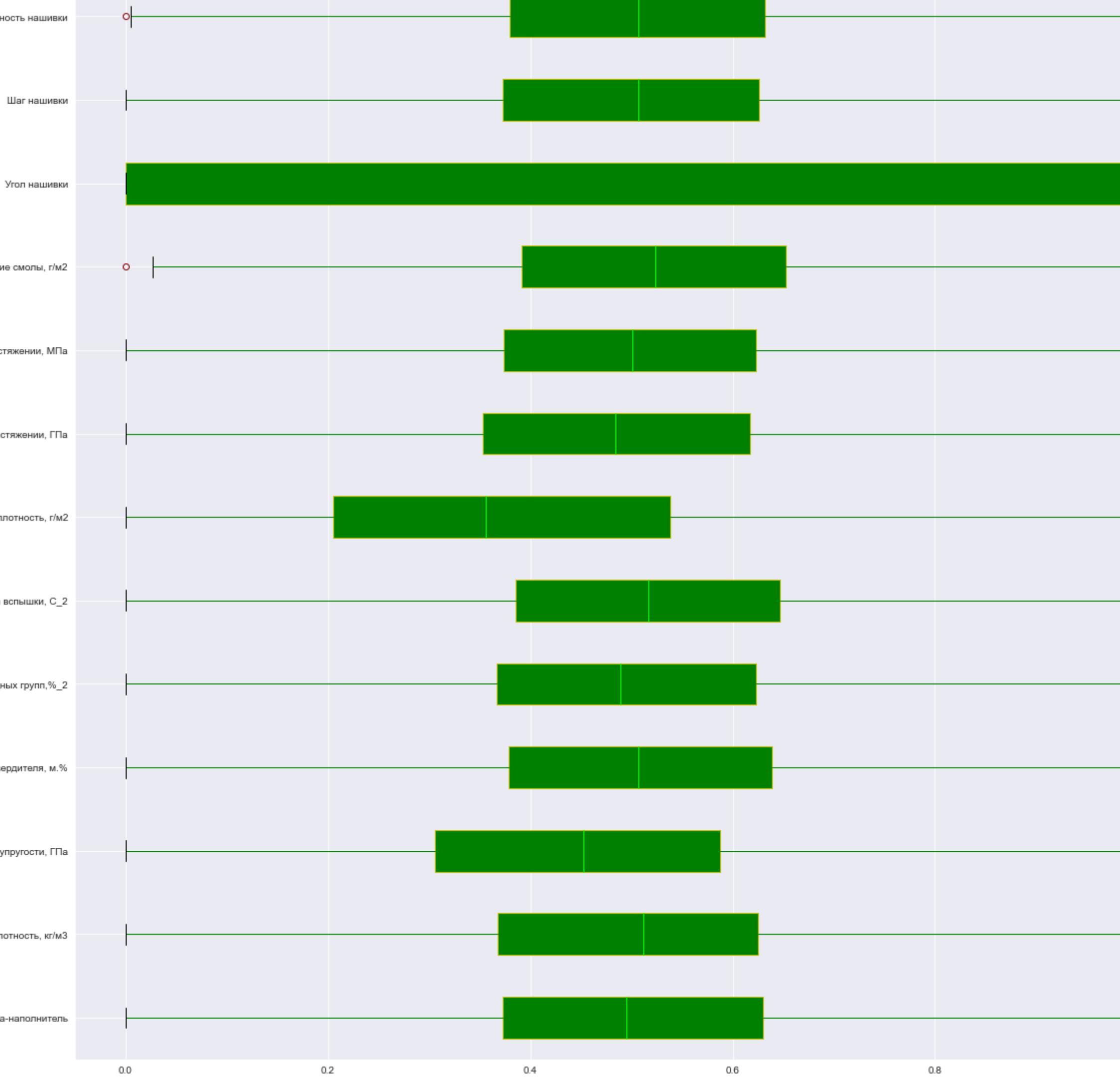
Out[71]: #Третий раз проверим сумму выбросов по каждому столбцу
df.isnull().sum()

Out[72]: 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
Поглощение смолы, г/м2 0
Угол наклона 0
Шаг нашивки 0
Плотность нашивки 0
dtype: int64

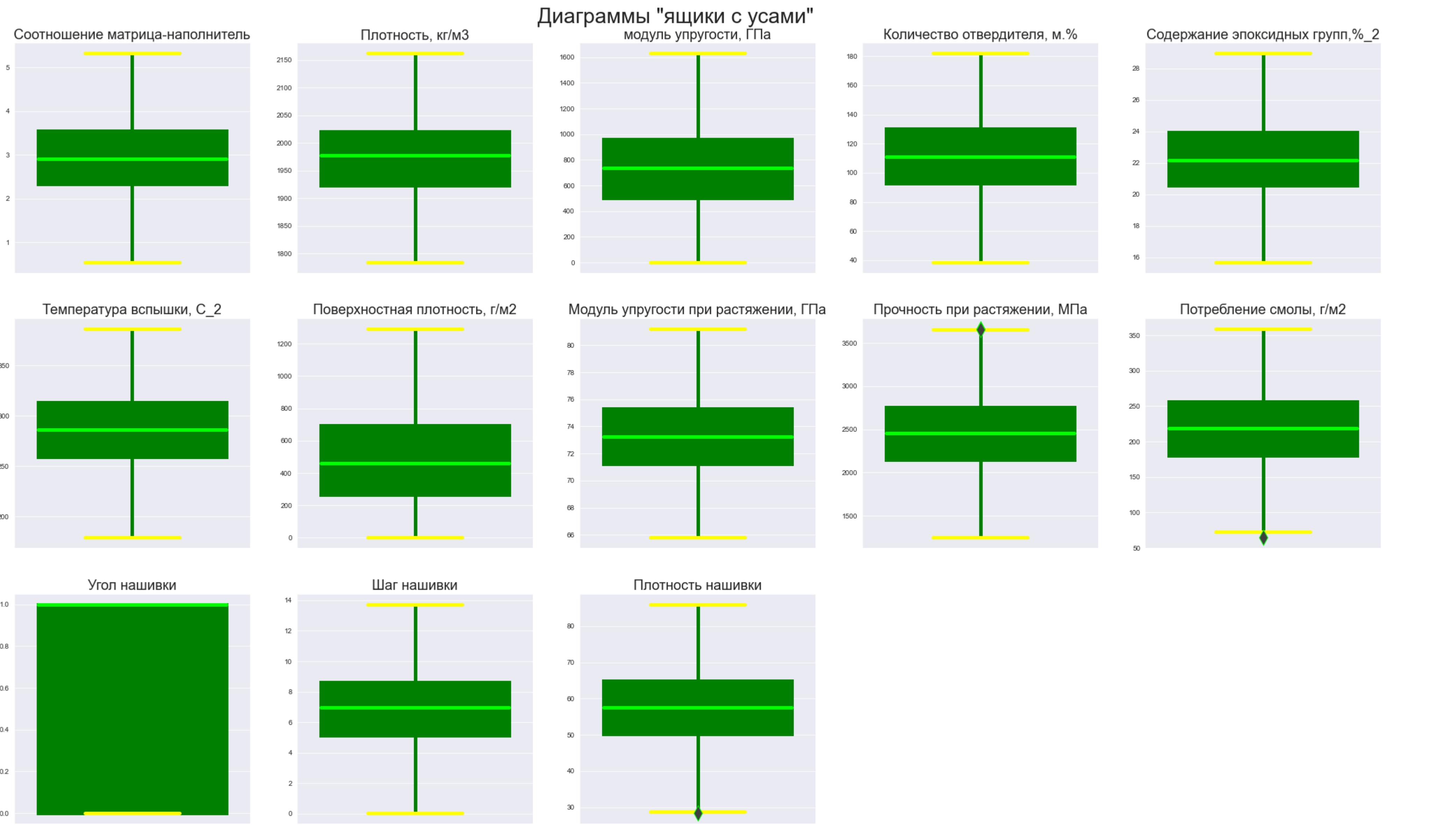
In [73]: #просмотрим информацию о нашем датасете после еще одного удаления пропусков. Видим, что строк стало еще меньше
df.info()

<class 'pandas.core.frame.DataFrame'>
Integers: 926 entries, 1 to 1022
Floats: 13 columns (total 13 columns):
 # Column Non-Null Count Dtype  
--- 
 0 Соотношение матрица-наполнитель 926 non-null float64
 1 Плотность, кг/м3 926 non-null float64
 2 Модуль упругости, ГПа 926 non-null float64
 3 Количество отвердителя, м.% 926 non-null float64
 4 Содержание эпоксидных групп,%_2 926 non-null float64
 5 Температура вспышки, С_2 926 non-null float64
 6 Поверхностная плотность, г/м2 926 non-null float64
 7 Модуль упругости при растяжении, ГПа 926 non-null float64
 8 Прочность при растяжении, МПа 926 non-null float64
 9 Поглощение смолы, г/м2 926 non-null float64
 10 Угол наклона 926 non-null int32
 11 Шаг нашивки 926 non-null float64
 12 Плотность нашивки 926 non-null float64
dtypes: float64(12), int32(1)
memory usage: 97.7 KB

In [74]: #В первый раз построим на "лапки с усами"
scaler = MinMaxScaler()
scaler.fit(df)
plt.figure(figsize = (20, 20))
#Видим "лапки"
sns.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 [76]: # И снова видим, что выбросы остались в некоторых исходных данных. Это вообще когда-нибудь закончится???

```
In [77]: # Подберем процедуру с выбросами еще раз. Надеюсь последний
for i in column_list_drop:
    q1 = df[i].quantile(0.25)
    intr_qn = df[i].quantile(0.75) - q1
    max = q75 + (1.5*intr_qn)
    min = q25 - (1.5*intr_qn)
    df.loc[df[i] < min,i] = np.nan
    df.loc[df[i] > max,i] = np.nan
```

In [78]: #Еще раз проверим сумму выбросов в каждом столбце
df.isnull().sum()

```
Out[78]:
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 [79]: #Еще раз удалаем строки с выбросами
df = df.dropna(axis=0)

In [80]: #И проверим снова сумму выбросов
df.isnull().sum()

```
Out[80]:
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 [81]: #Посмотрим на числовой датасет
df.info()

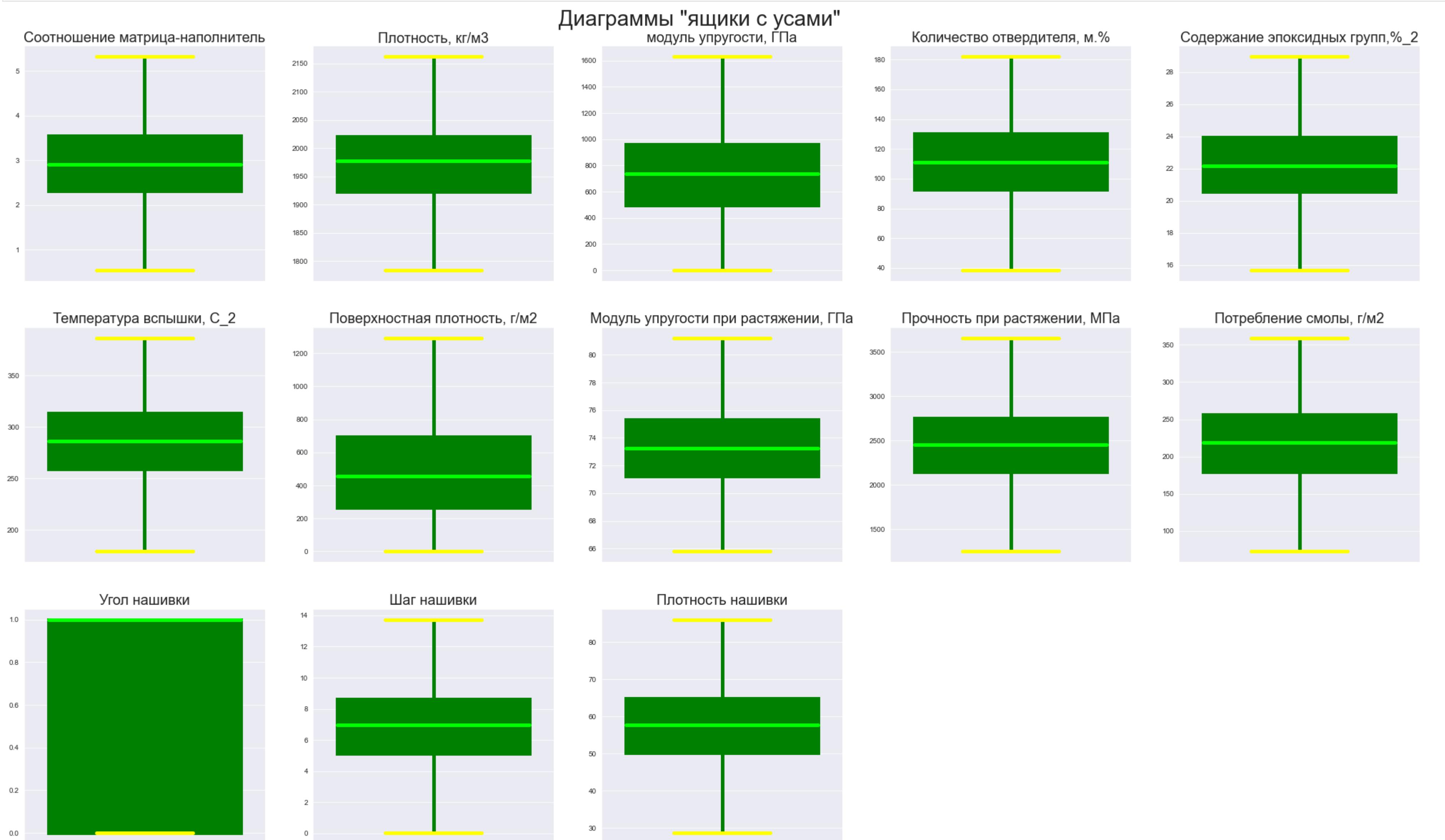
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 922 entries, 1 to 1022
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
 --- 
 0   Соотношение матрица-наполнитель      922 non-null   float64
 1   Плотность, кг/м3                   922 non-null   float64
 2   Модуль упругости, ГПа              922 non-null   float64
 3   Количество отвердителя, м.%        922 non-null   float64
 4   Содержание эпоксидных групп, %_2   922 non-null   float64
 5   Температура вспышки, С_2          922 non-null   float64
 6   Поверхностная плотность, г/м2      922 non-null   float64
 7   Модуль упругости при растяжении, ГПа 922 non-null   float64
 8   Прочность при растяжении, МПа     922 non-null   float64
 9   Потребление смолы, г/м2          922 non-null   float64
 10  Угол нашивки                     922 non-null   int32  
 11  Шаг нашивки                      922 non-null   float64
 12  Плотность нашивки                922 non-null   float64
dtypes: float64(12), int32(1)
memory usage: 97.2 KB
```

In [82]: # "ящики с усами"(боксплоты) (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", markeredgcolor = "maroon"))
plt.show()

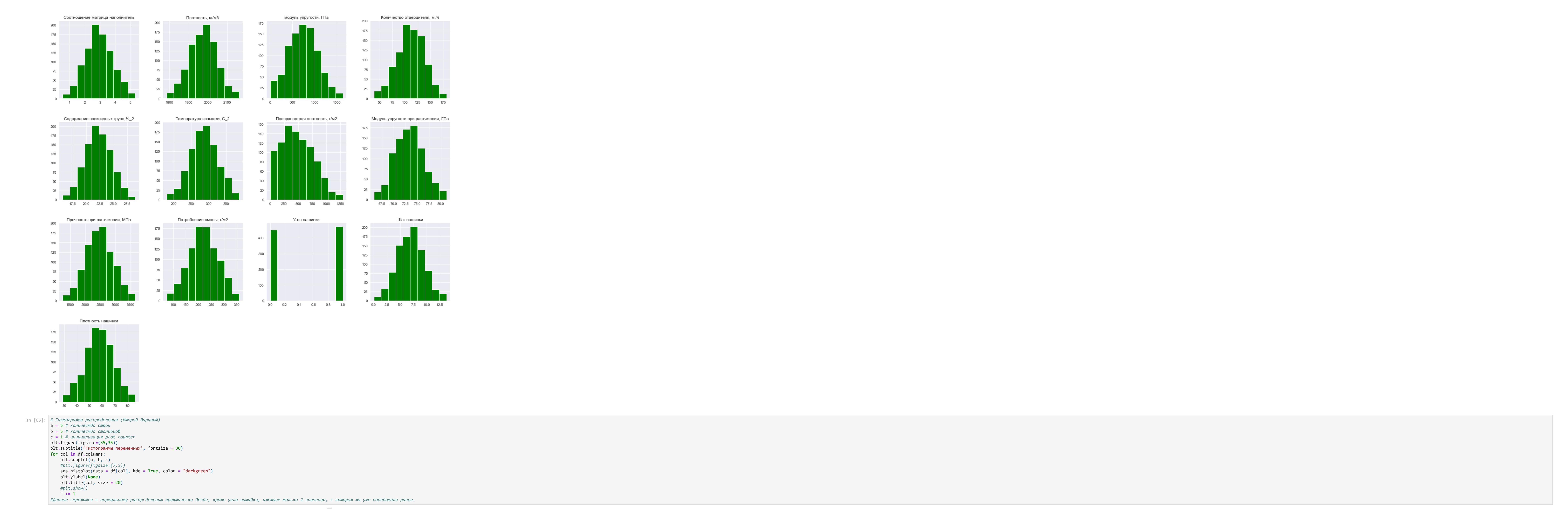
Диаграммы "ящики с усами"



```
In [83]: # Ящики с усами (Вероятность)
# 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))
    plt.title(col, size = 20)
    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.show()
    c += 1
```

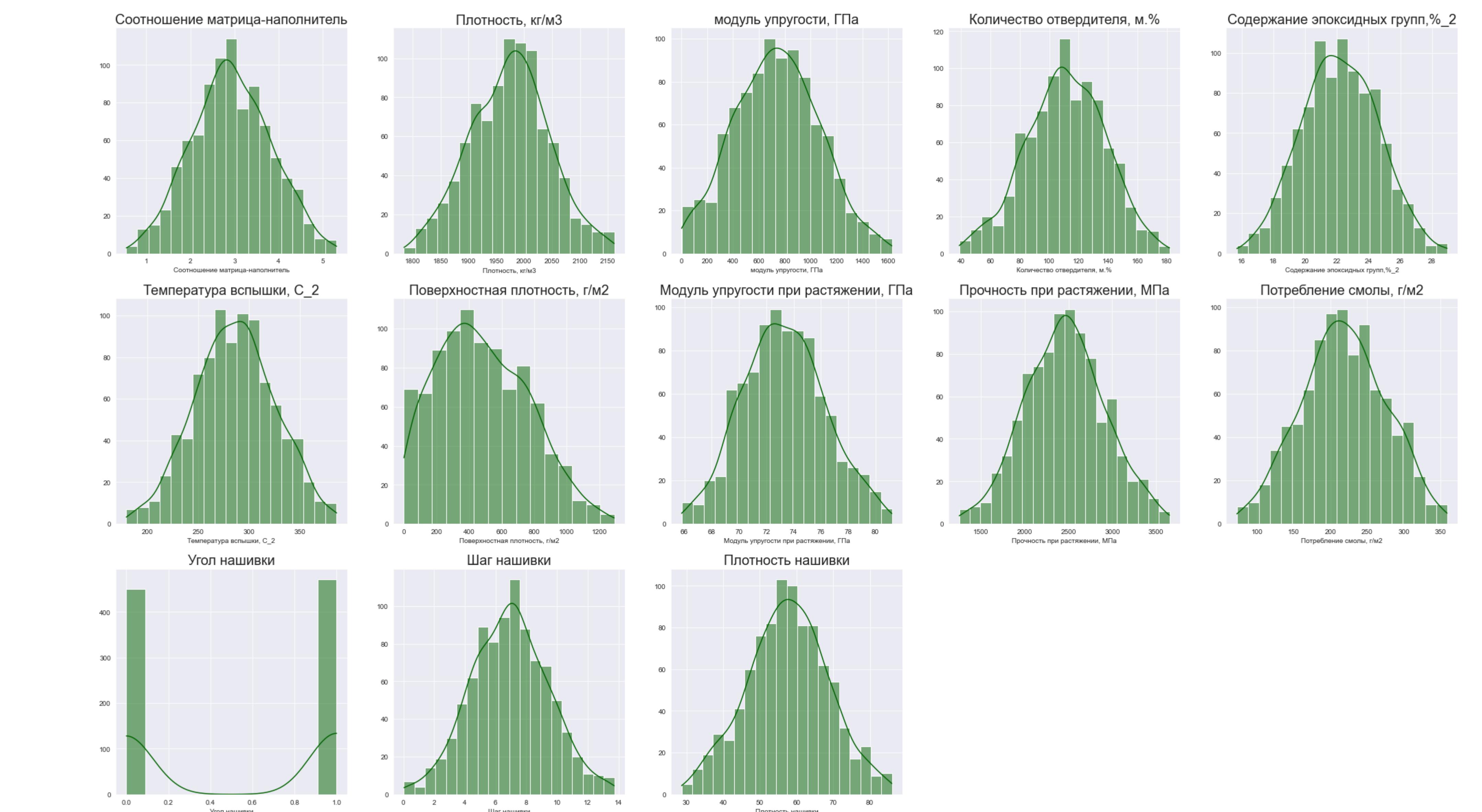


```
In [84]: # Построим гистограммы распределения каждой из переменных без нормализации
df.hist(figsize = (20,20), color = "g")
plt.show()
```

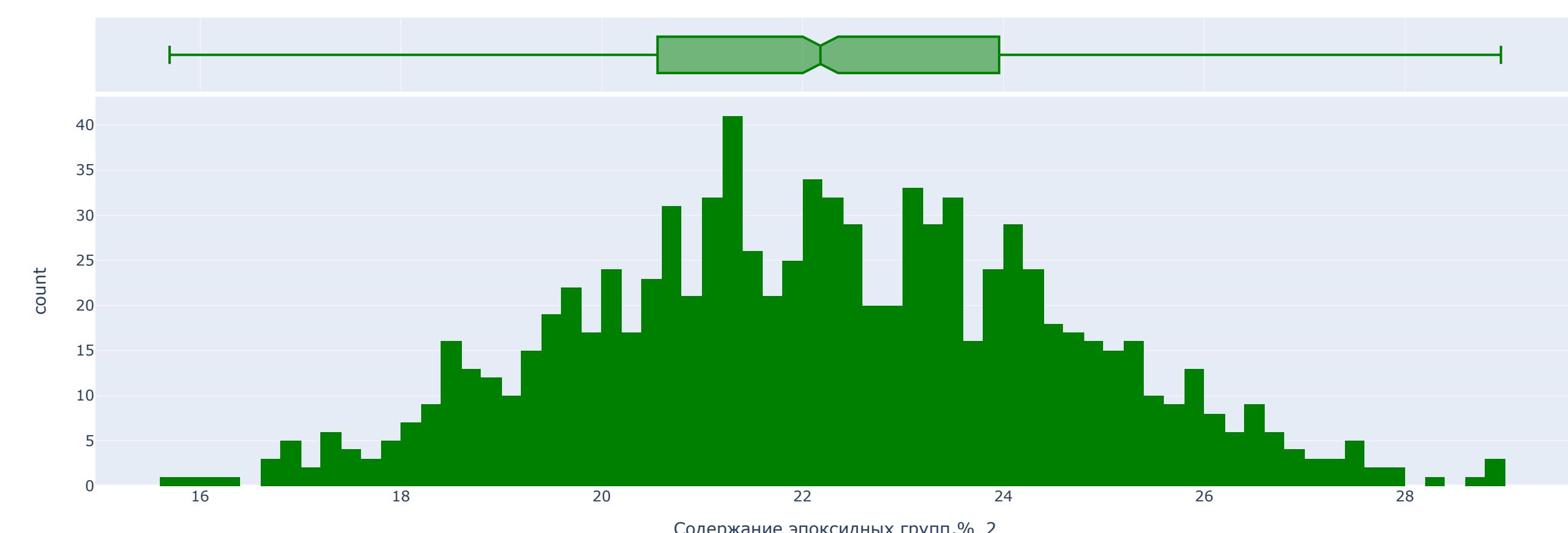
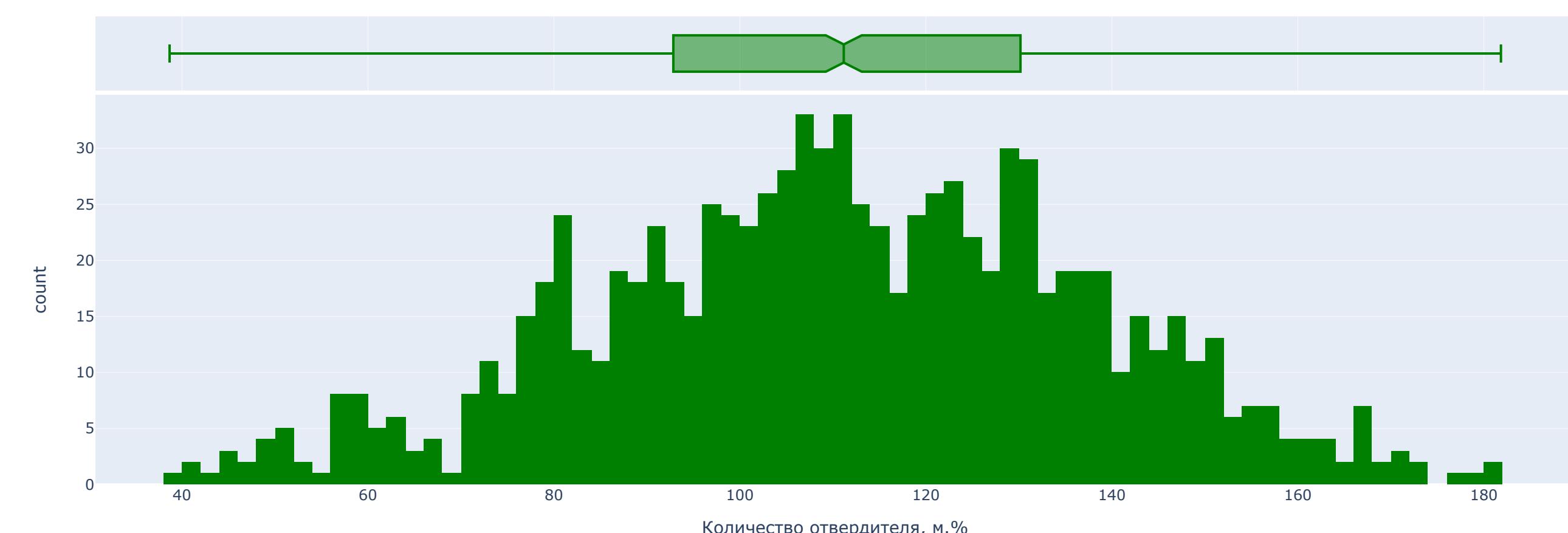
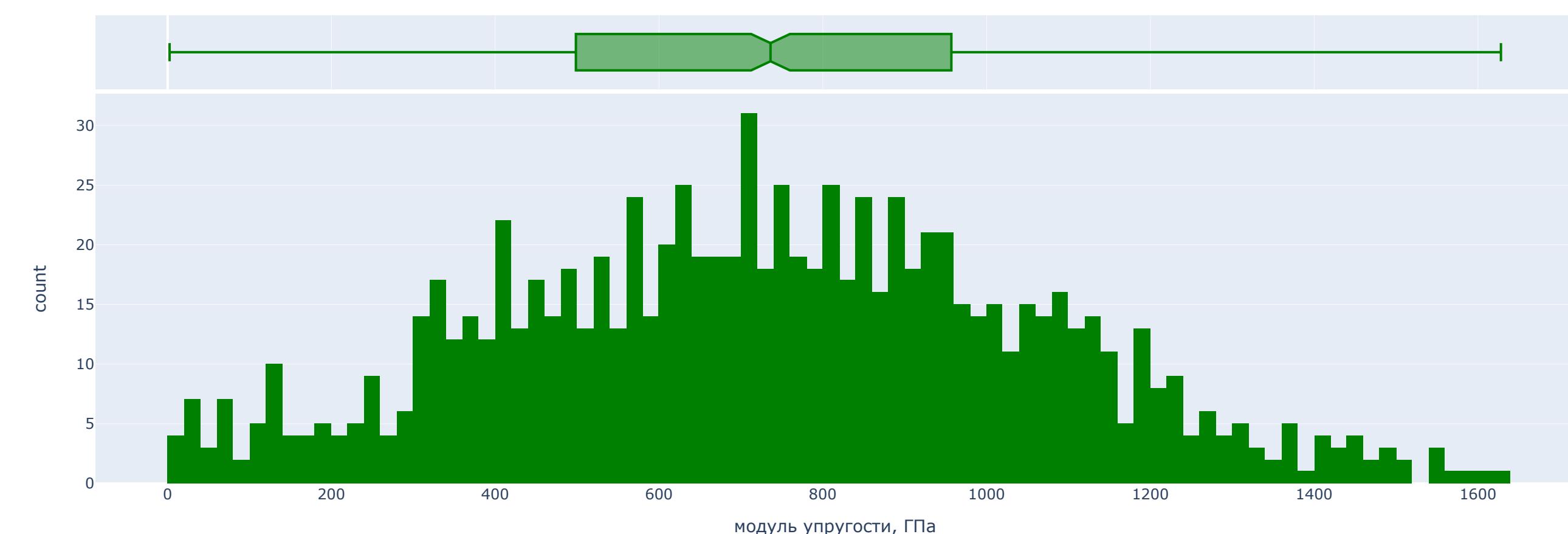
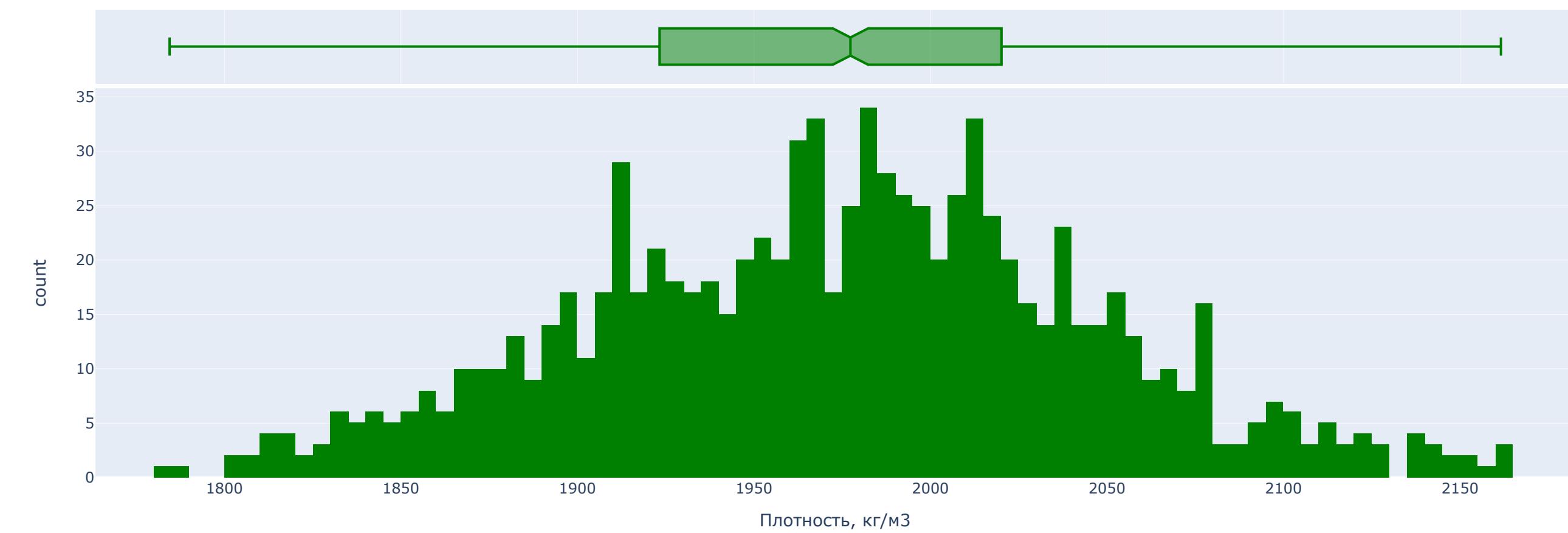
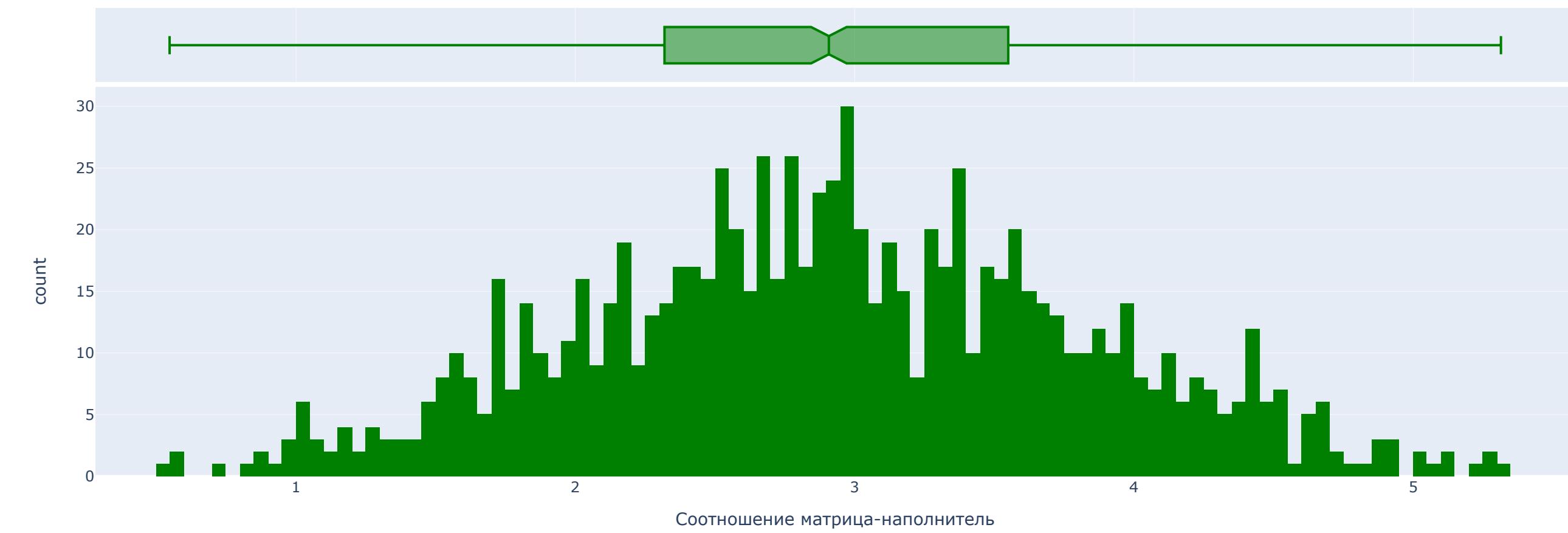


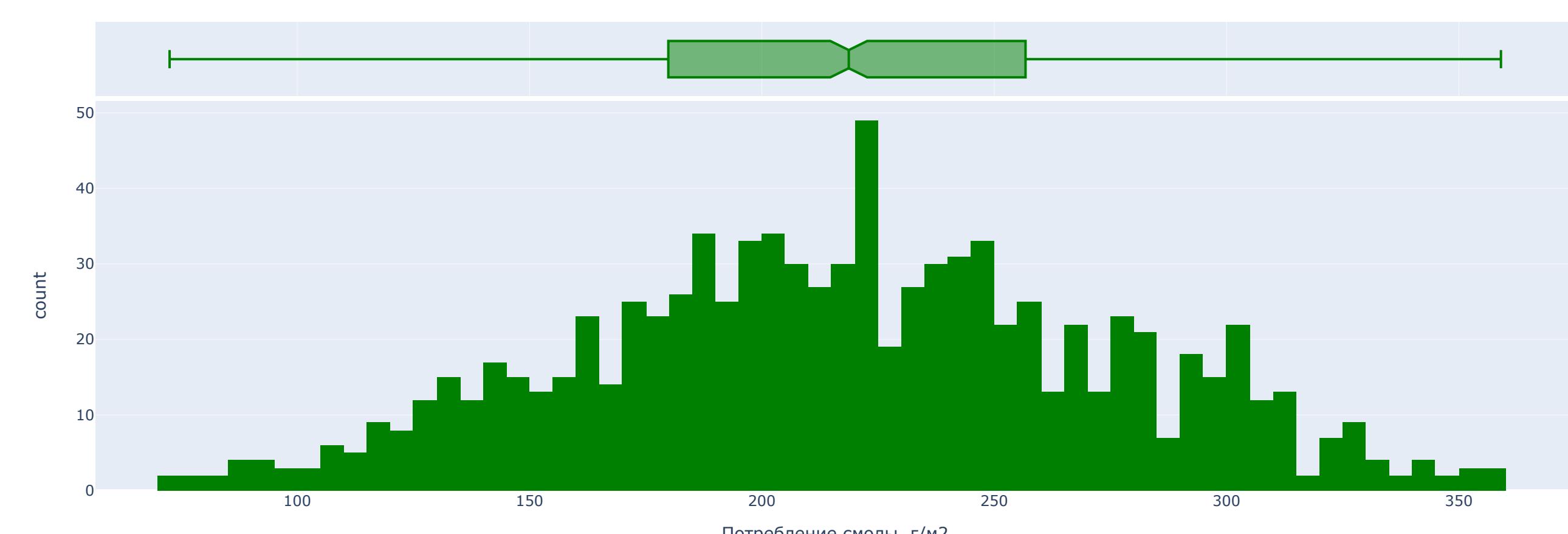
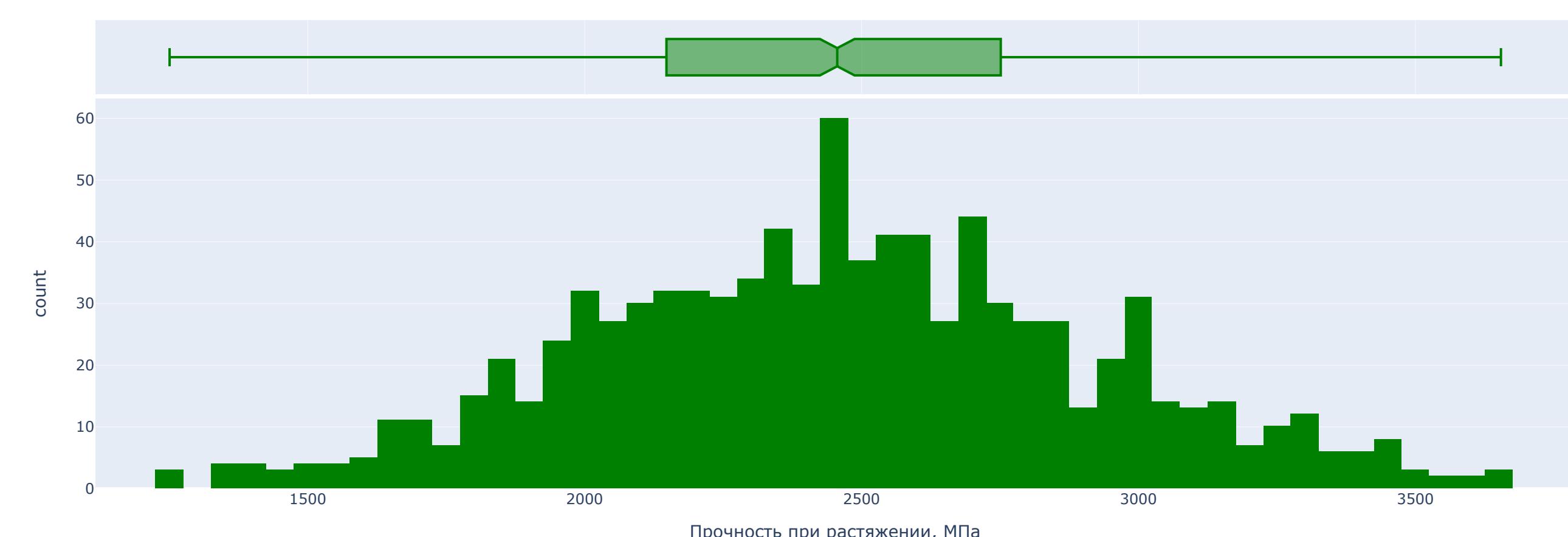
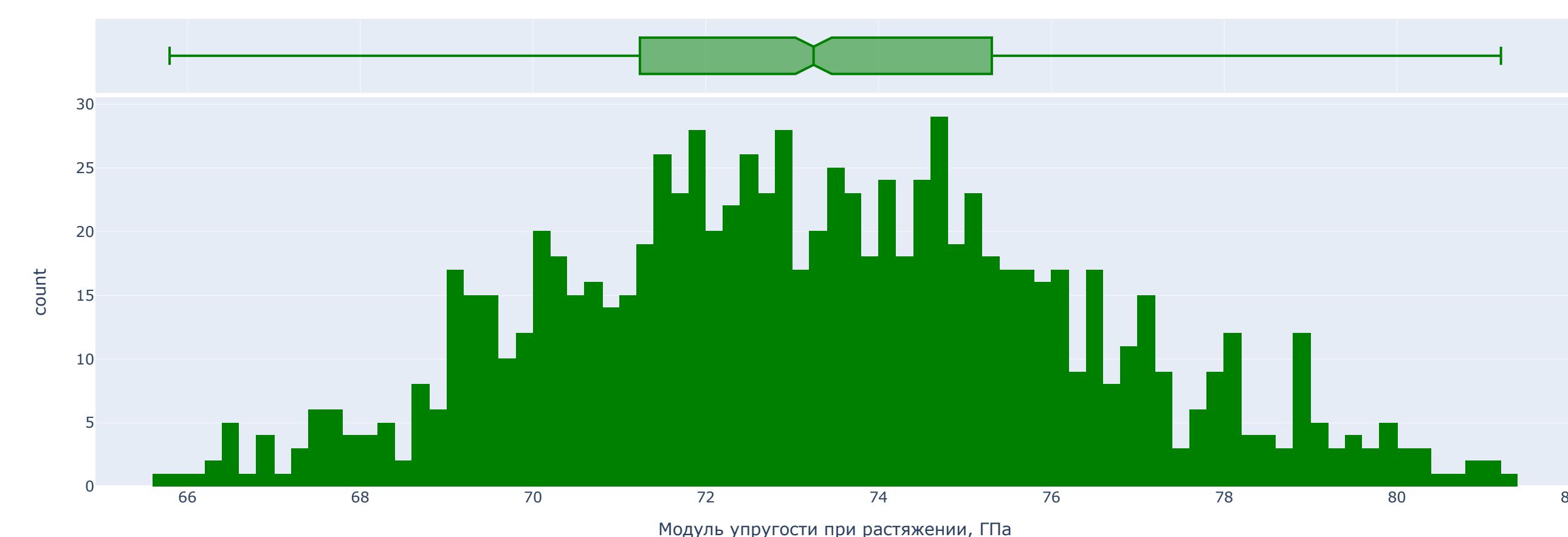
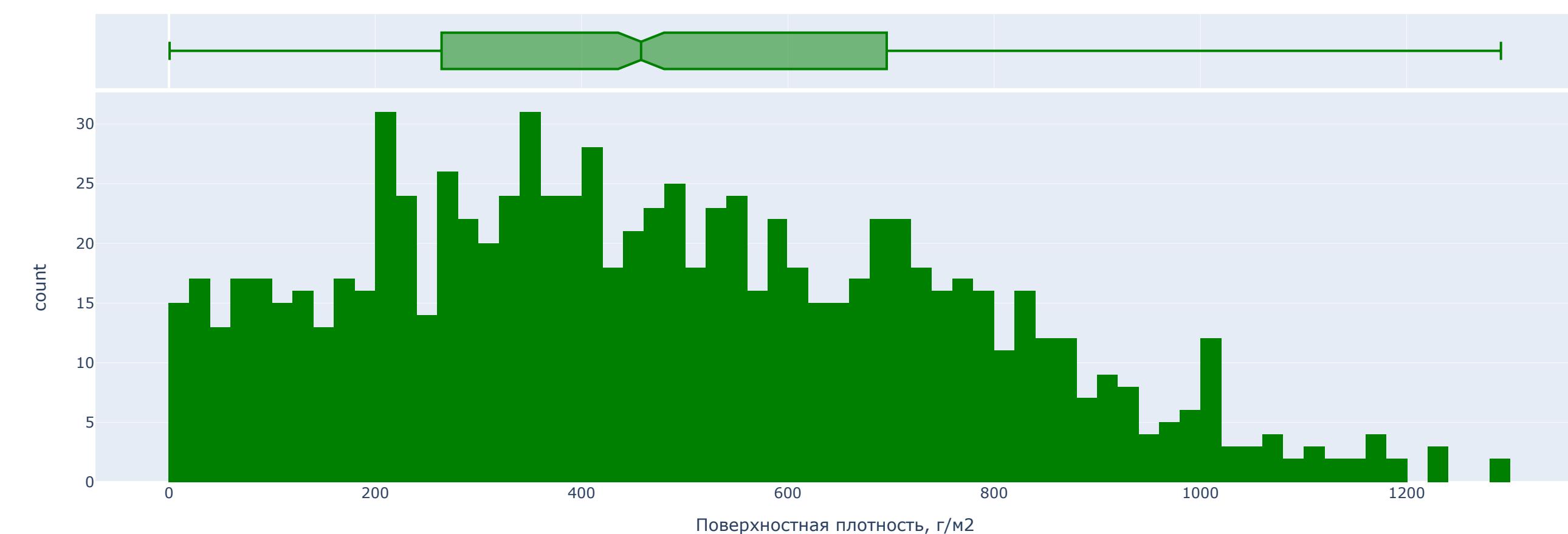
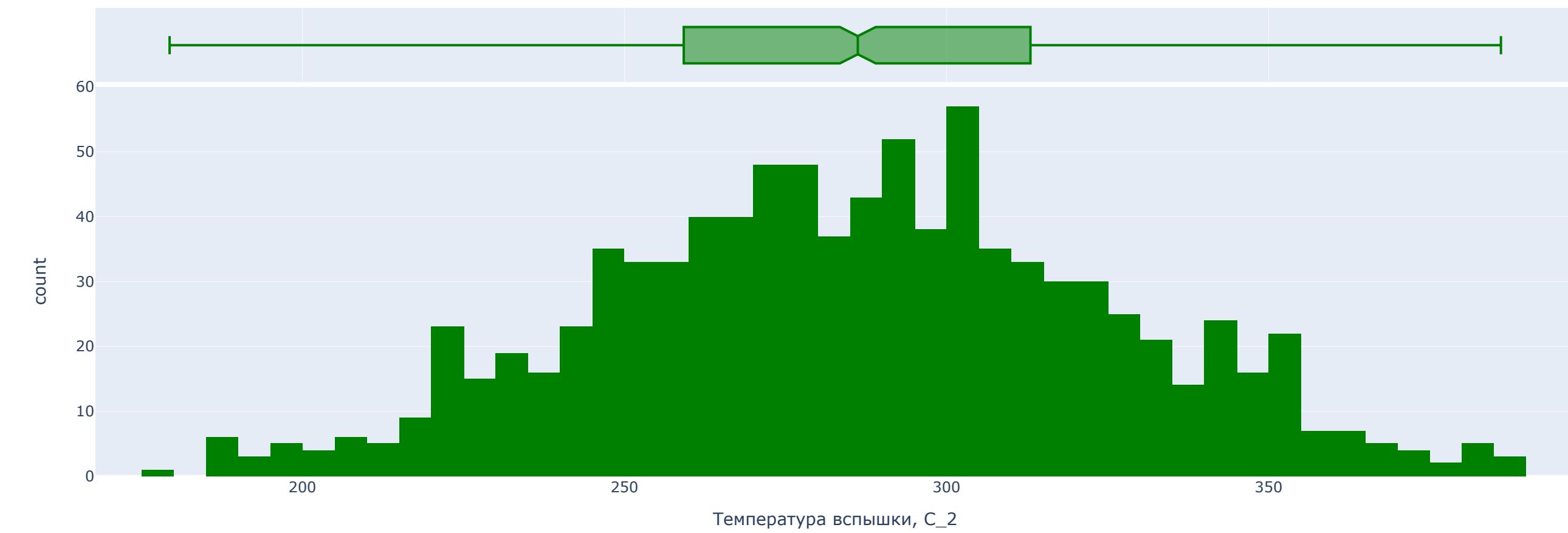
```
In [85]: # Гистограммы распределения (второй вариант)
a = 5 # количество столбцов
b = 3 # количество строк
c = 1 # инициализация plot counter
plt.figure(figsize=(35,35))
plt.suptitle('Гистограммы переменных', fontsize = 30)
for col in range(a):
    for row in range(b):
        plt.subplot(a, b, c)
        plt.figure(figsize=(7,5))
        sns.histplot(data = df[col], kde = True, color = "darkgreen")
        plt.title(col, size = 20)
        plt.show()
        c += 1
```

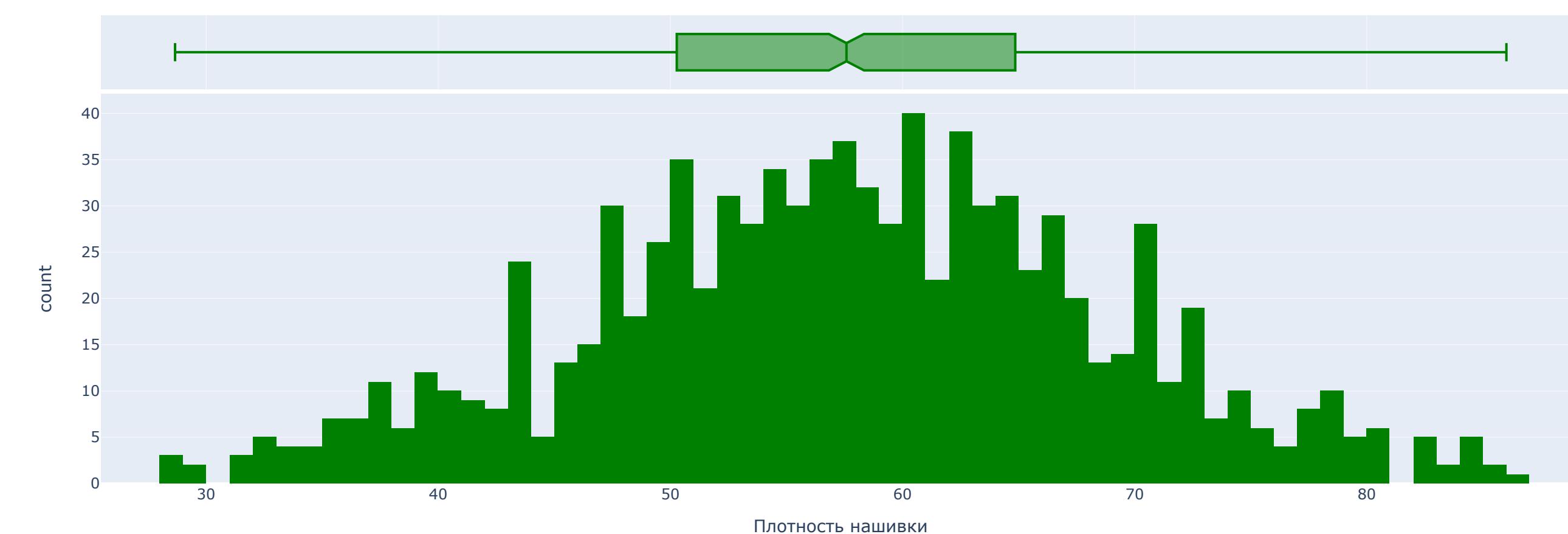
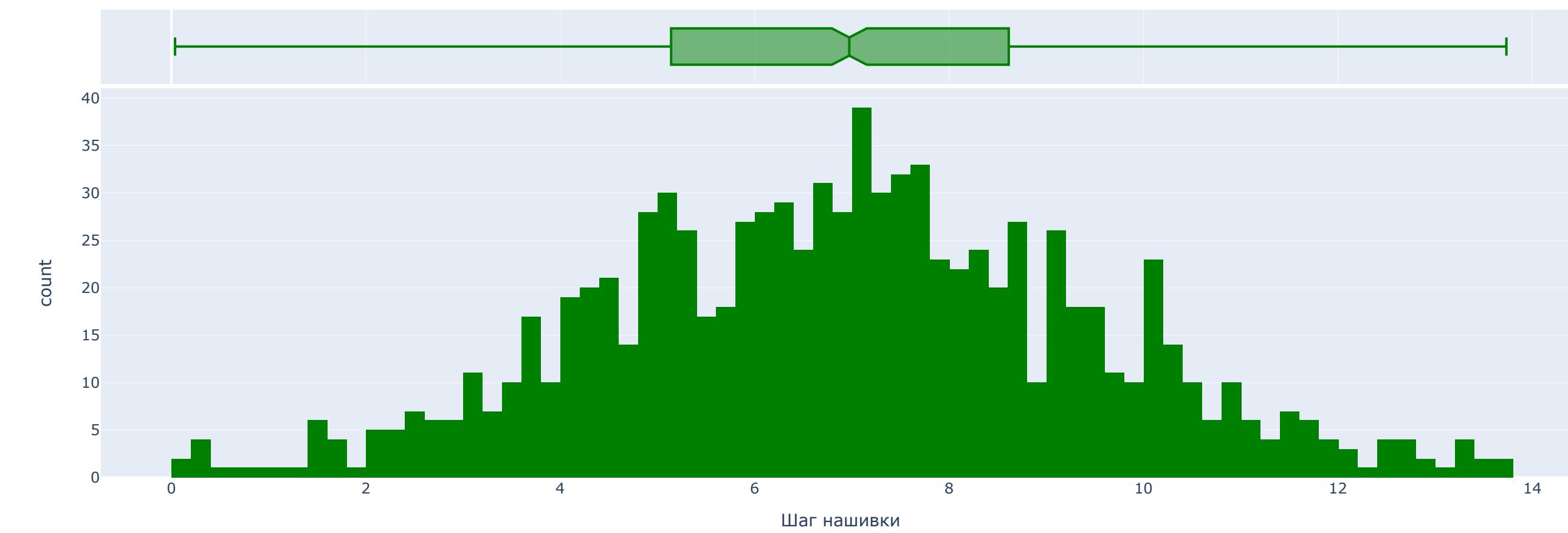
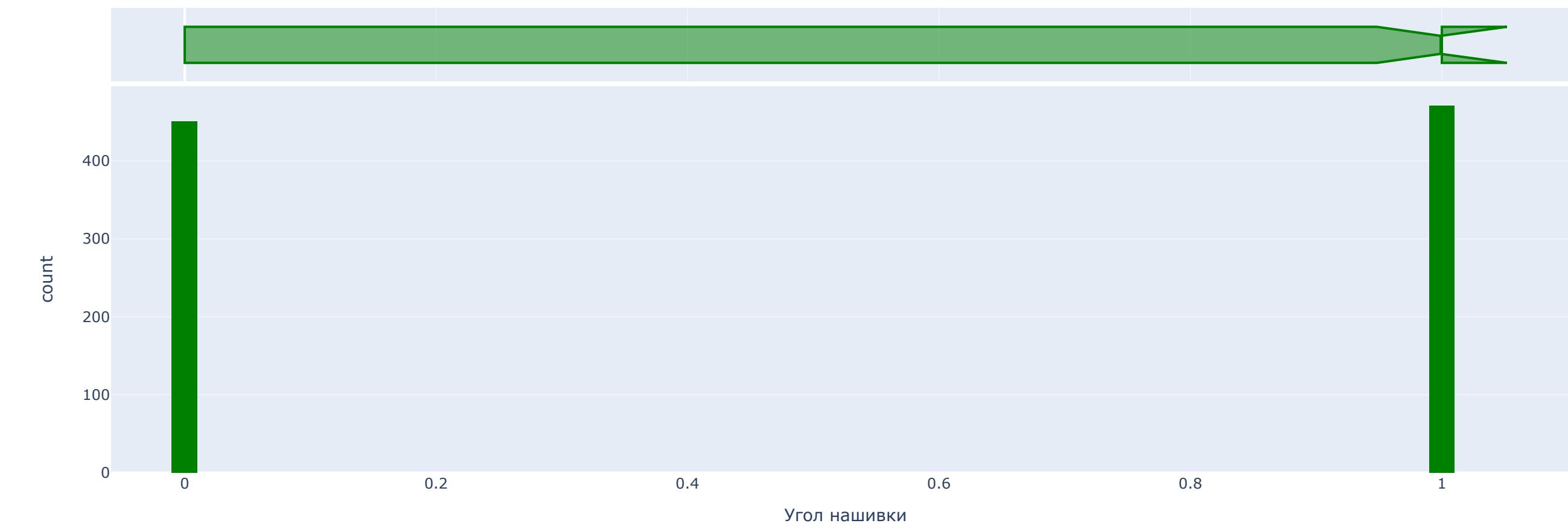
Гистограммы переменных



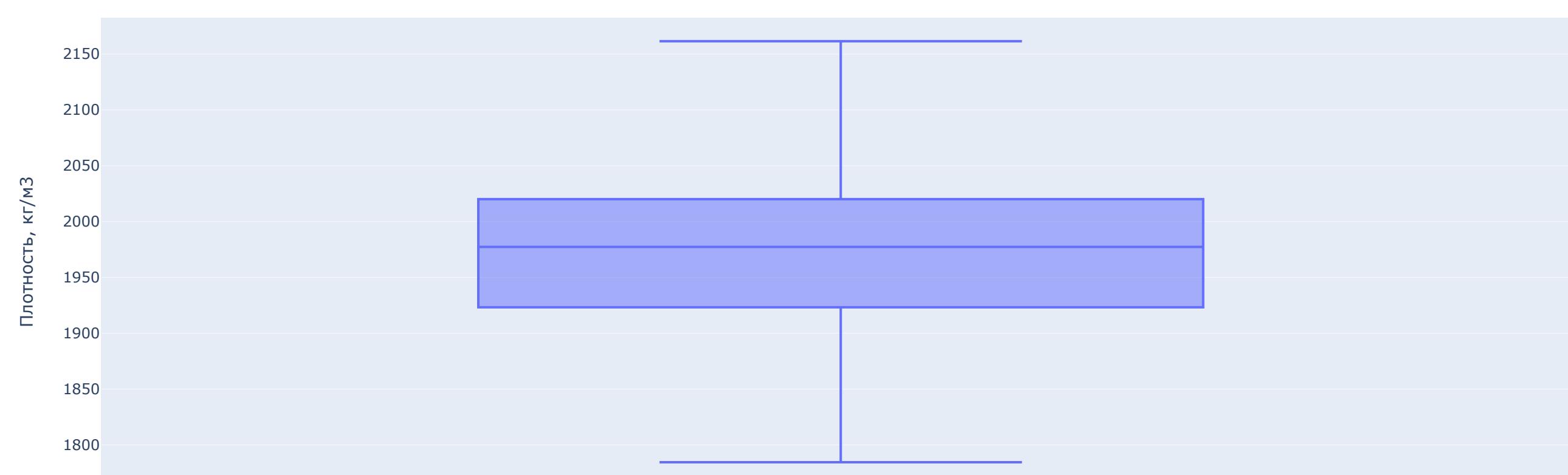
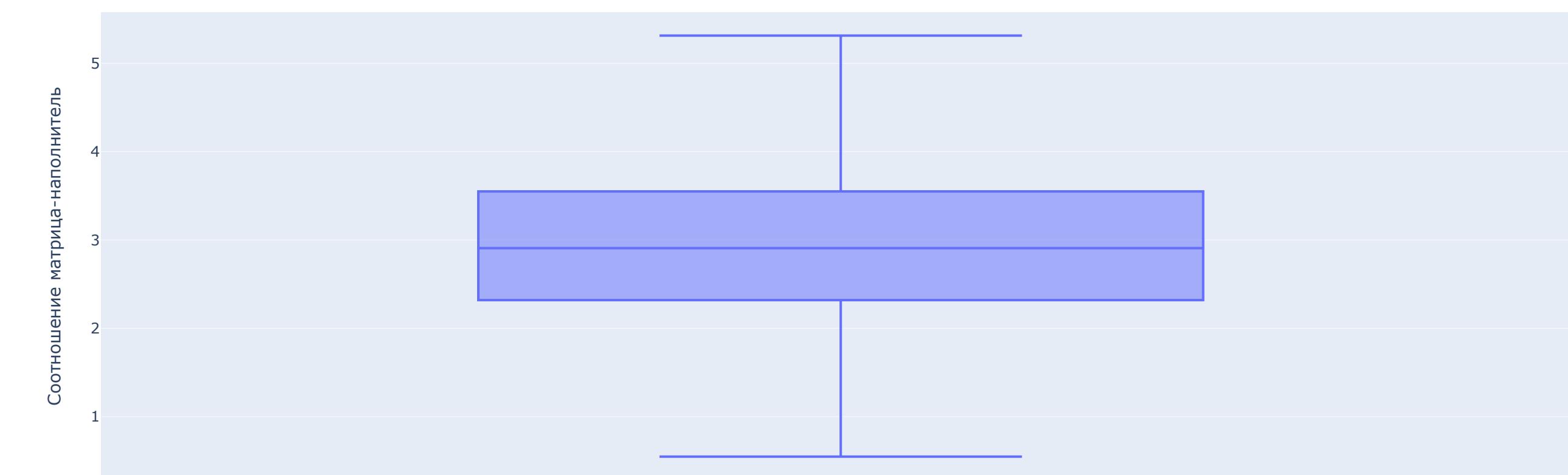
```
In [86]: # гистограмма распределения с бокслотом (третий вариант)
for column in df.columns:
    fig = px.histogram(df, x=column, color_discrete_sequence=['green'], nbins=100, marginal="box")
    fig.show()
```

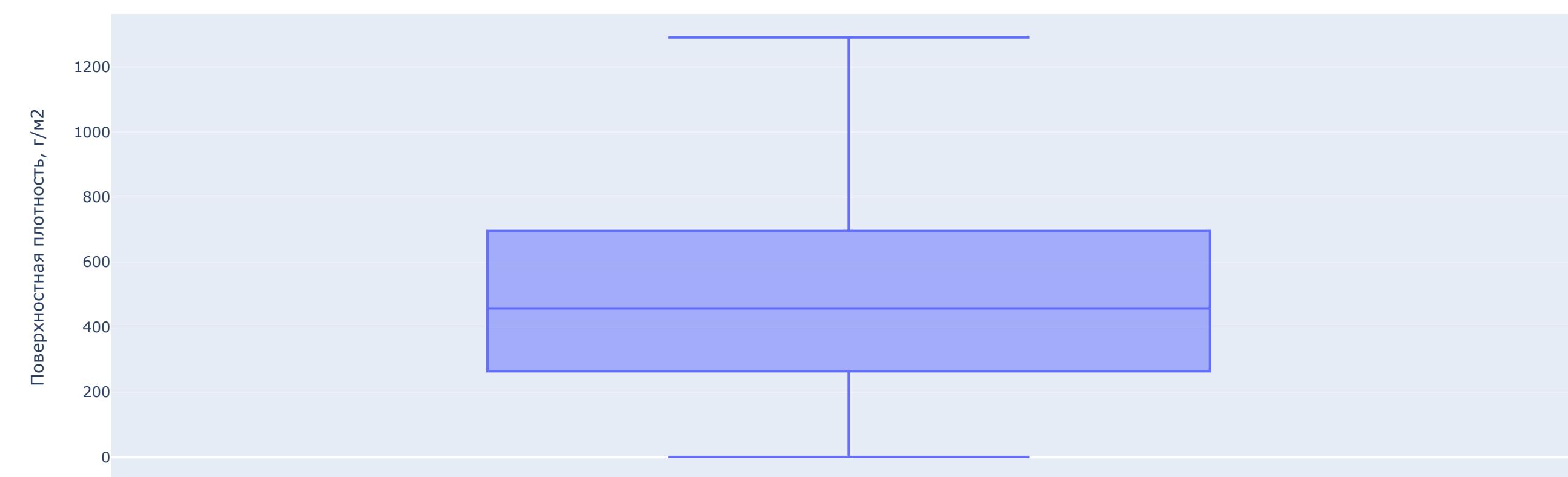
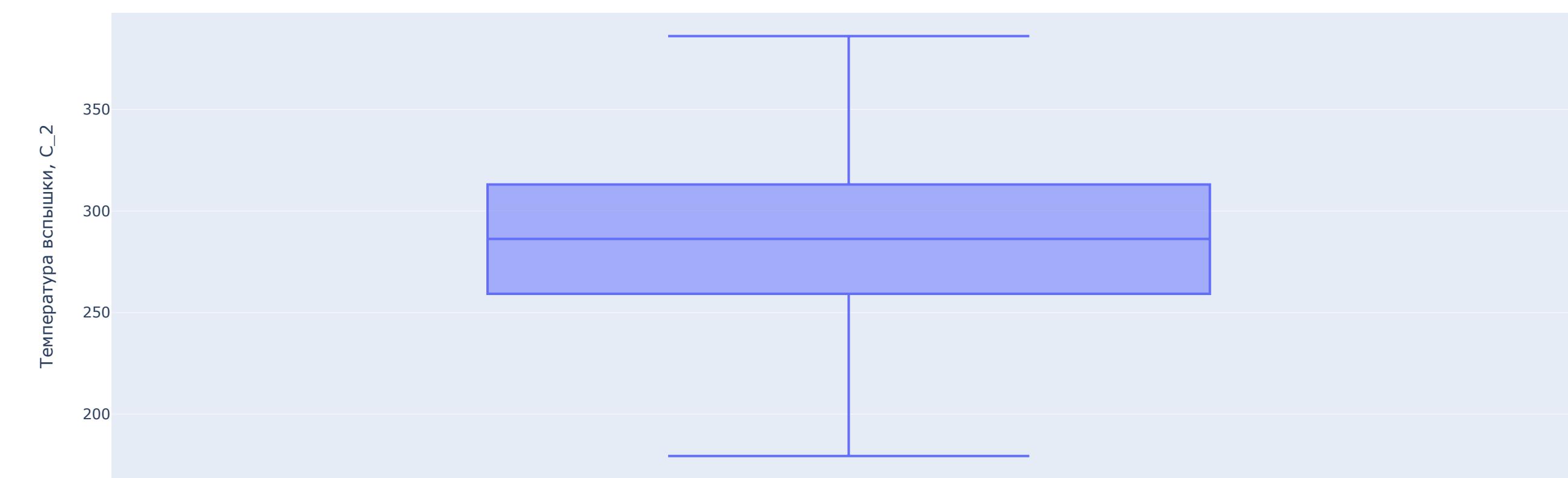
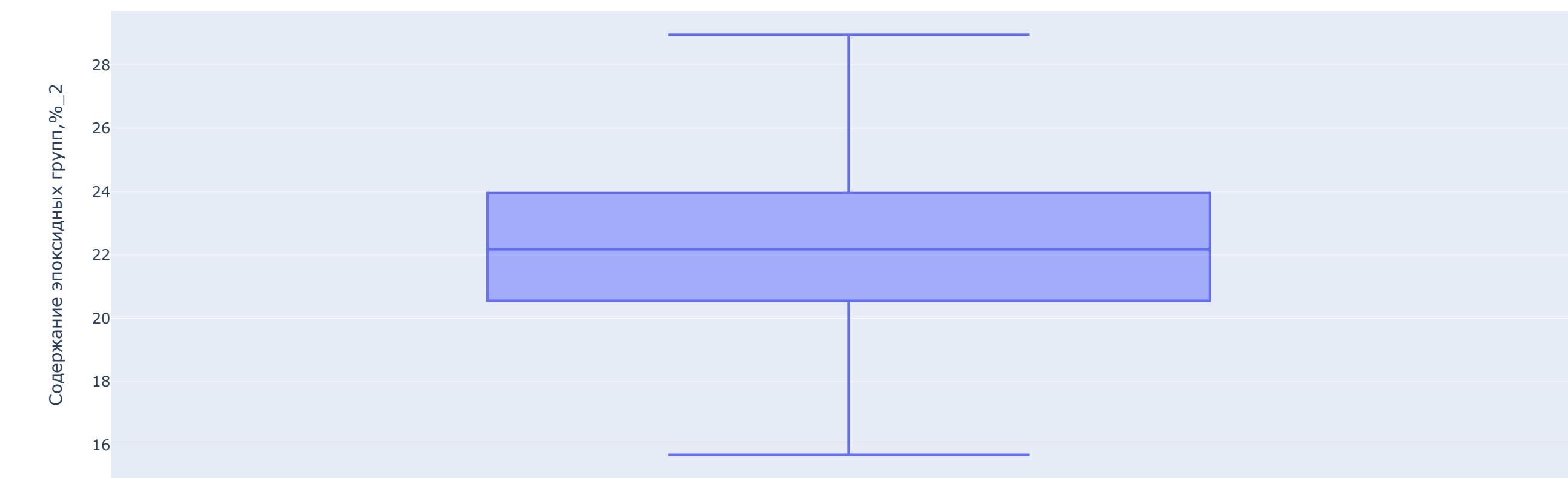
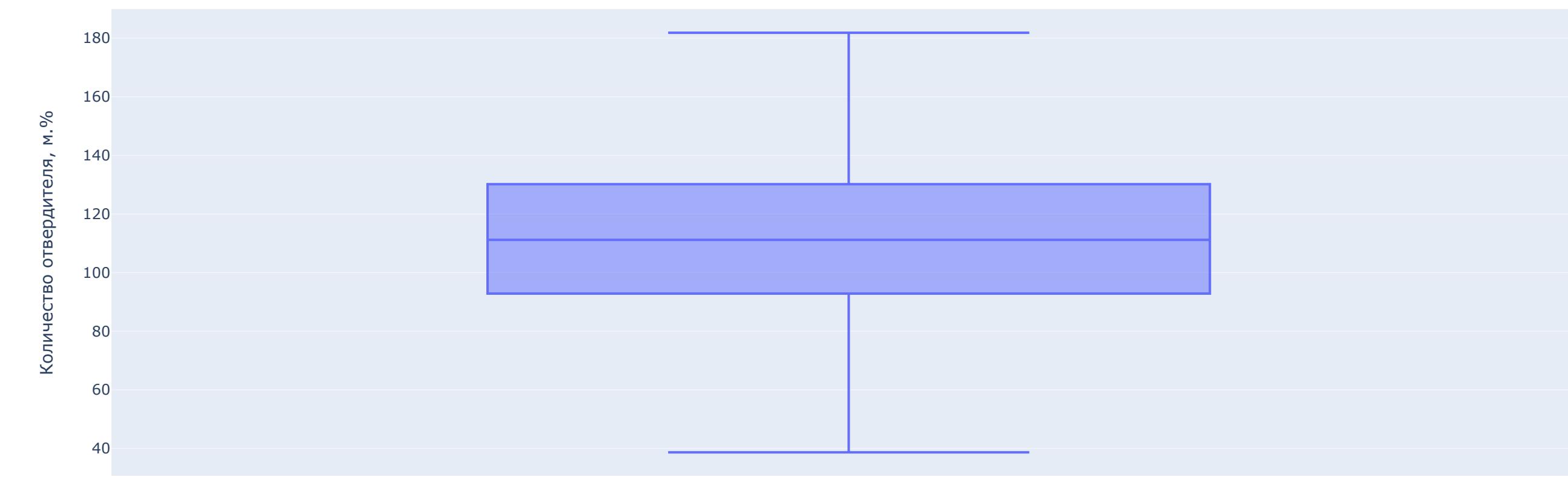
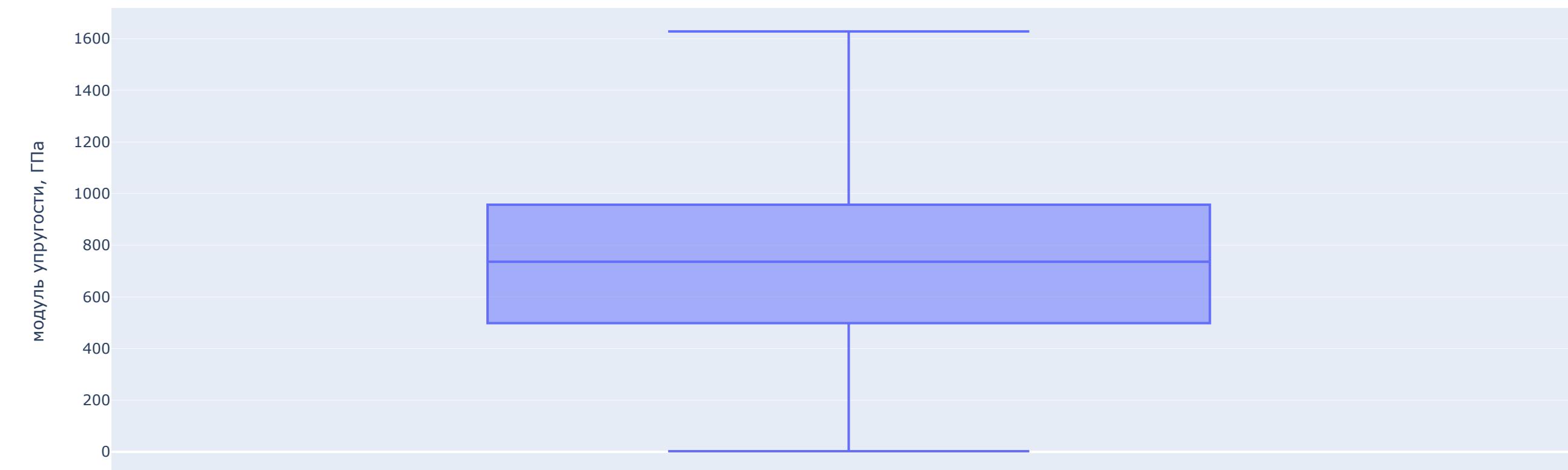


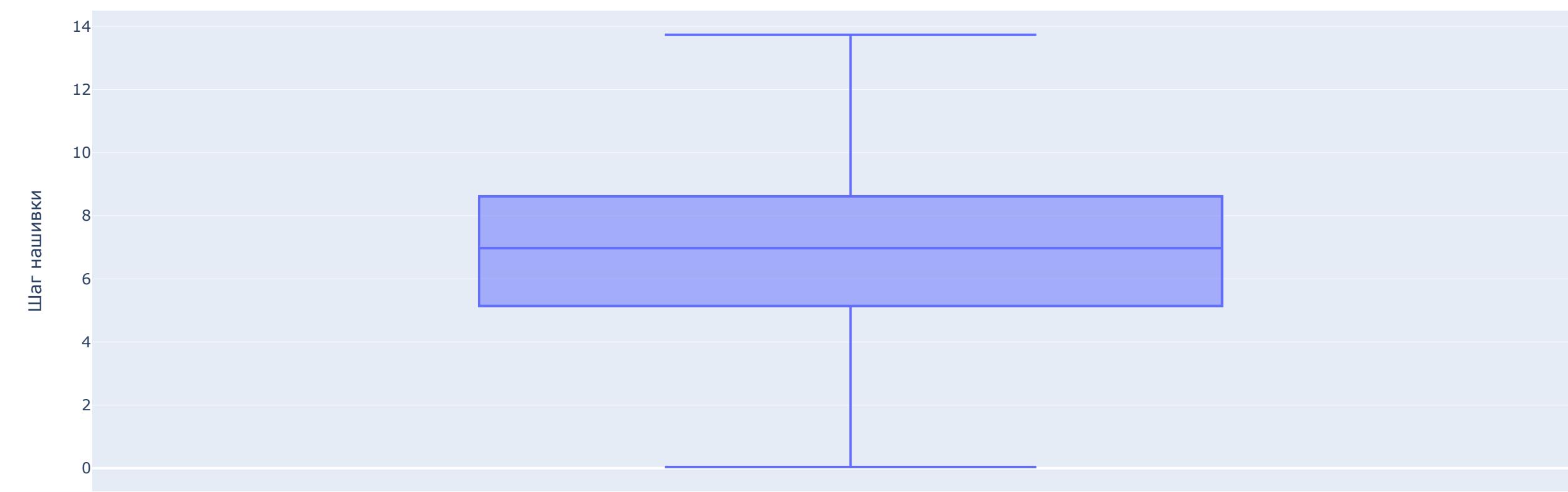
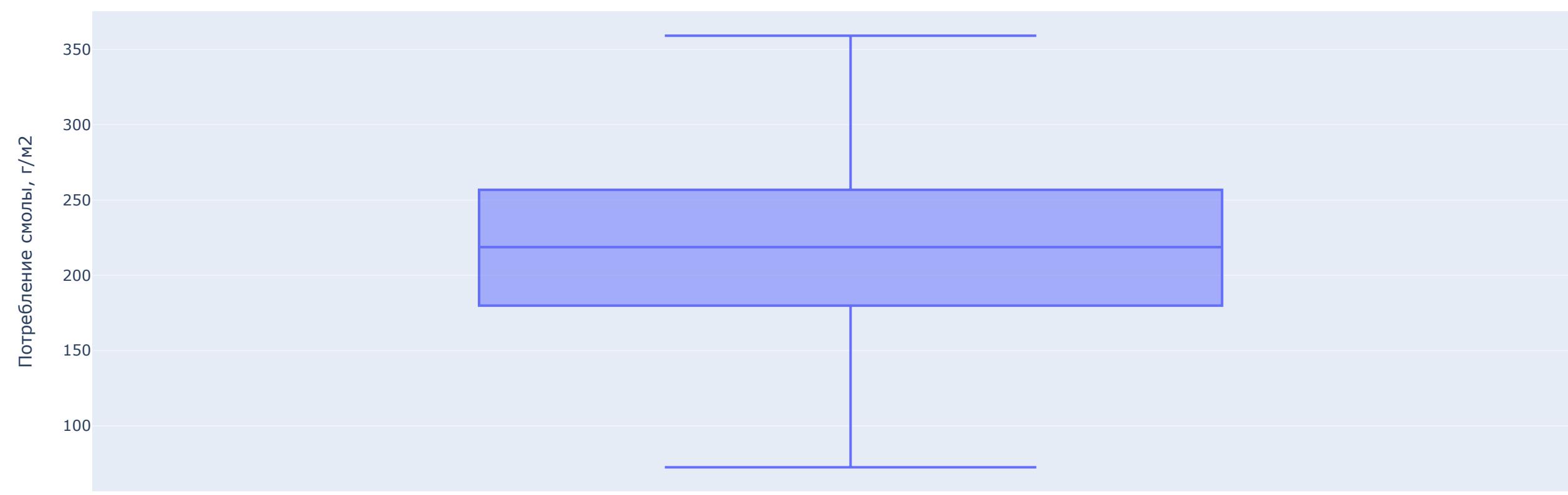
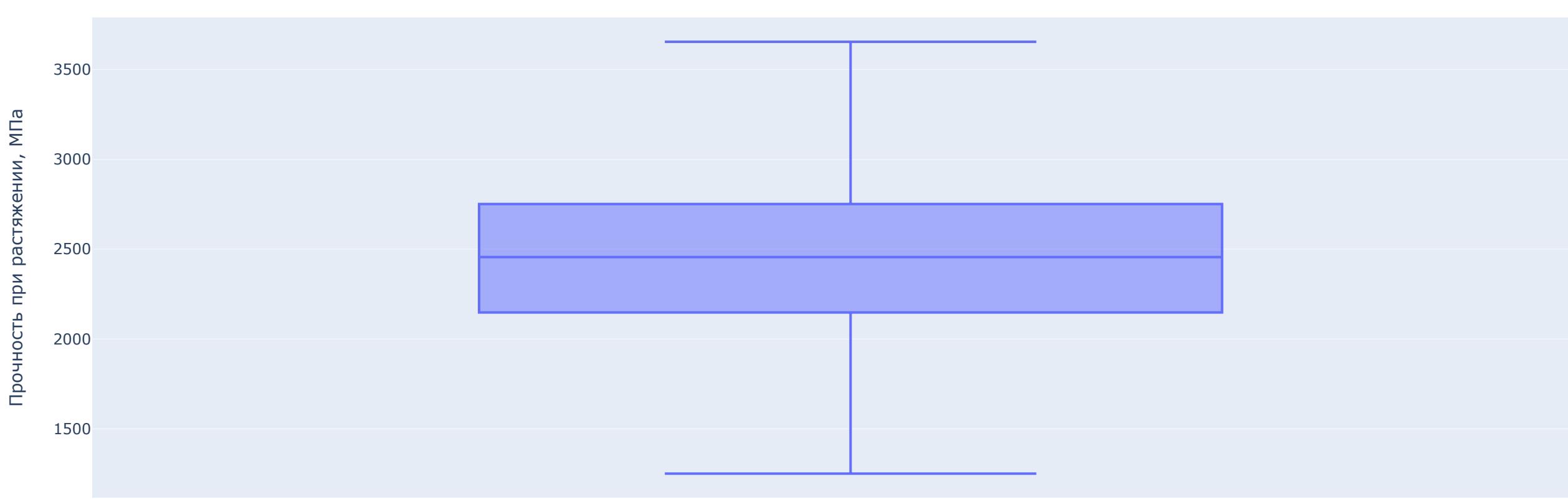
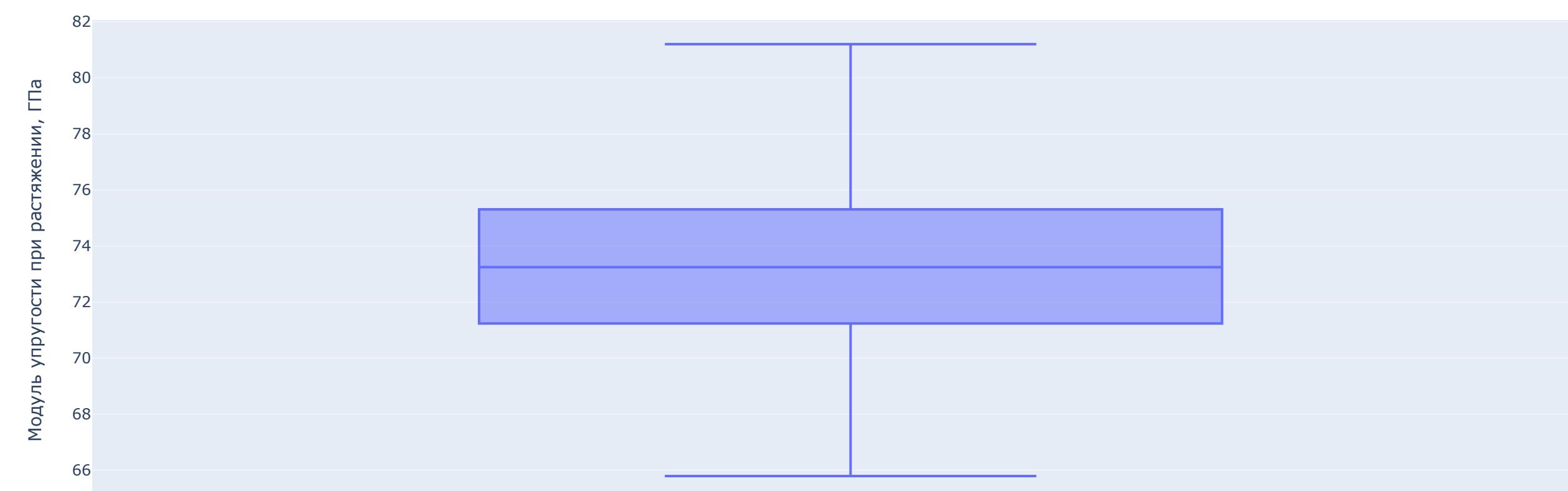


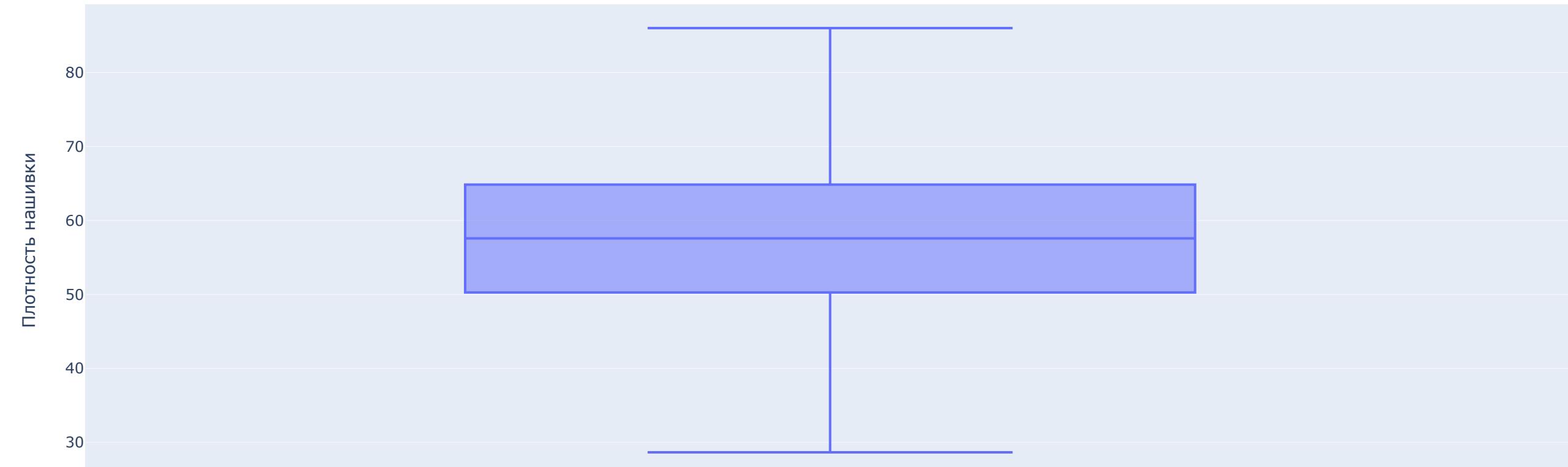


```
In [87]: for column in df.columns:  
    fig = px.box(df, y=column)  
    fig.show()
```

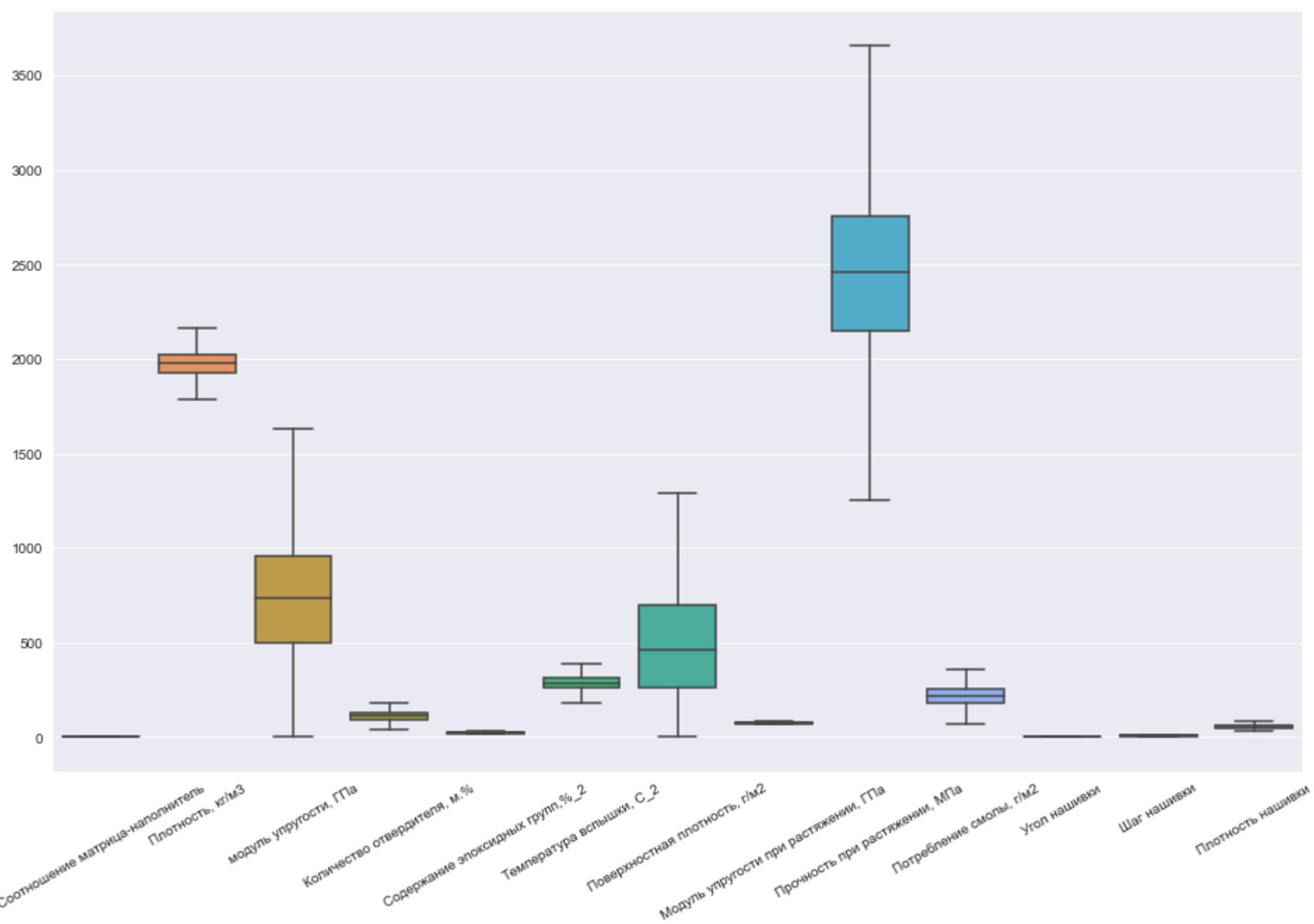








```
In [88]: #данные с условием на один рисунок
plt.figure(figsize=(16,10))
ax = sns.boxplot(data=df)
ax.set_xticklabels(ax.get_xticklabels(), rotation=30);
```



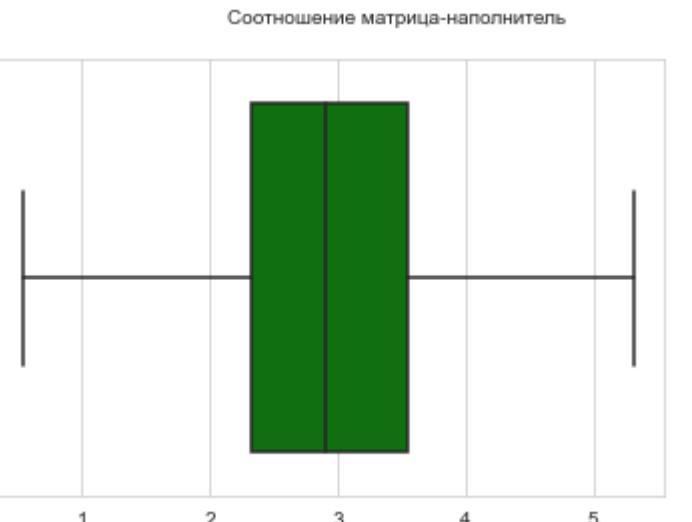
```
for column_name in column_names:
    print(column_name)

#Построение распределения
gis = df[column_name]
sns.set_style("whitegrid")
sns.kdeplot(data=gis, shade=True, palette="colorblind", color="g")
plt.show()

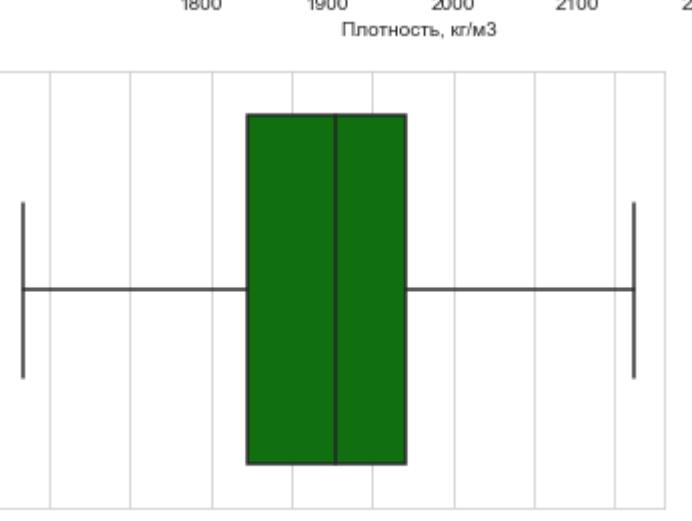
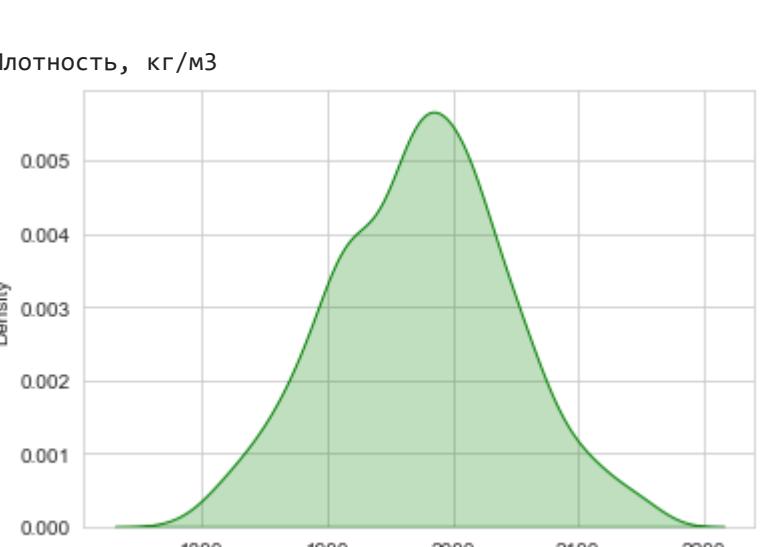
#Графиком "Лист с условии"
sns.boxplot(x=gis, color="g");
plt.show()

#Значения (мин максср)
print("Минимальное значение: ", end=" ")
print(np.min(gis))
print("Максимальное значение: ", end=" ")
print(np.max(gis))
print("Среднее значение: ", end=" ")
print(np.mean(gis))

print("Медианное значение: ", end=" ")
print(np.median(gis))
print("\n\n")
```

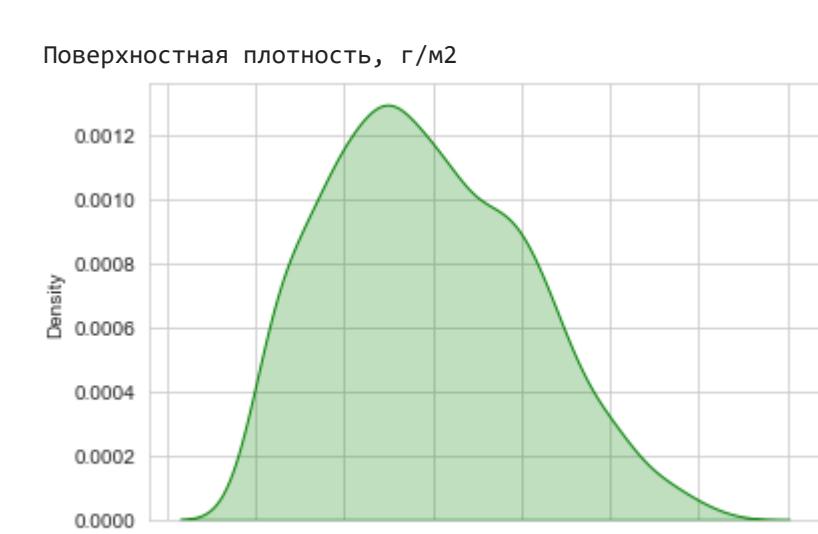
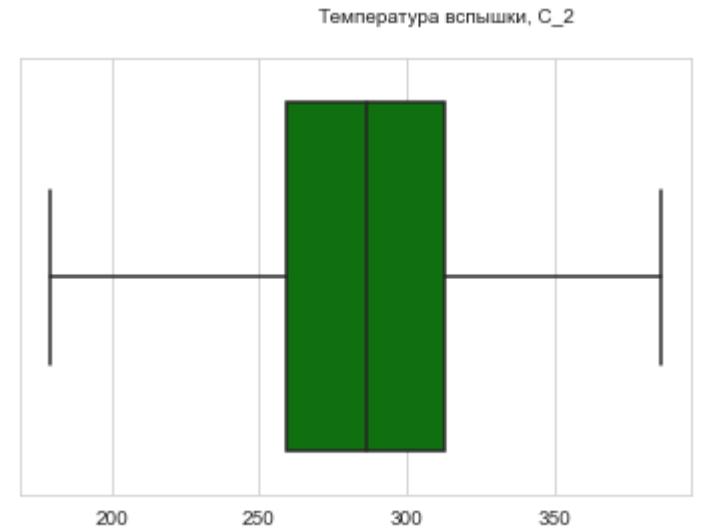
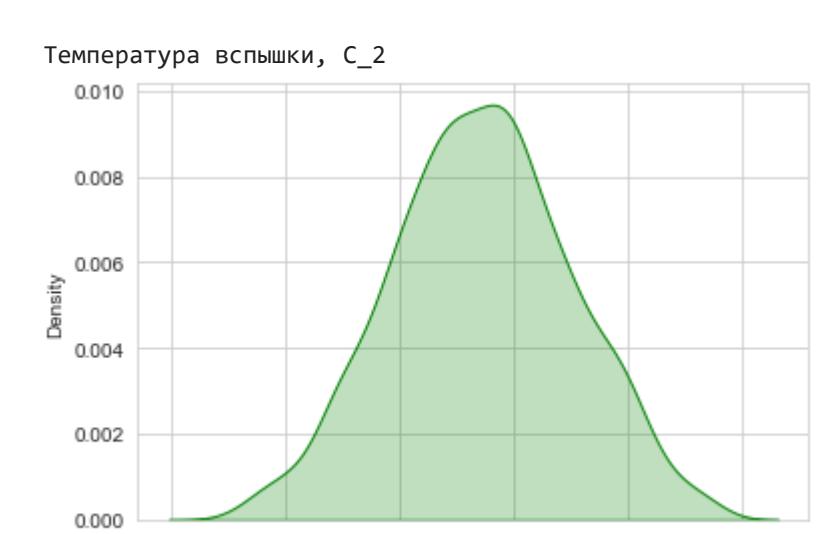
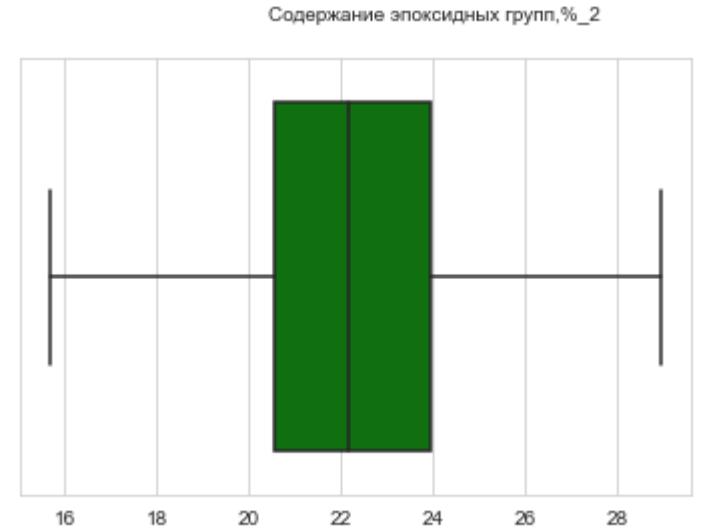
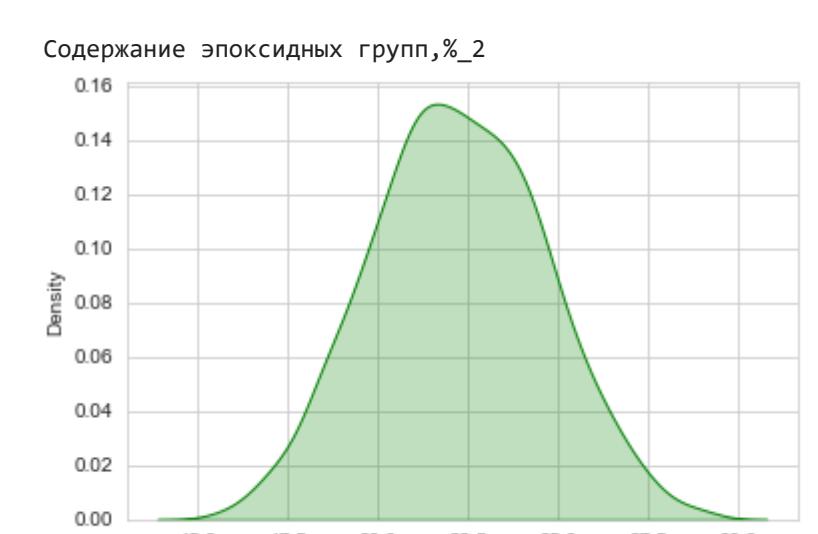
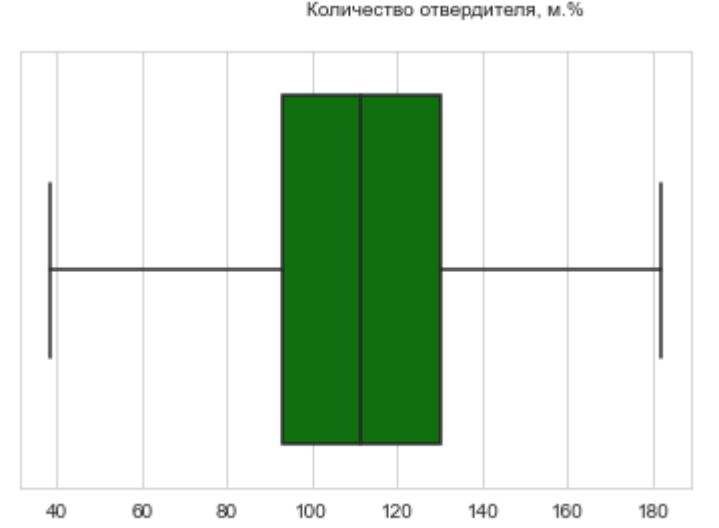
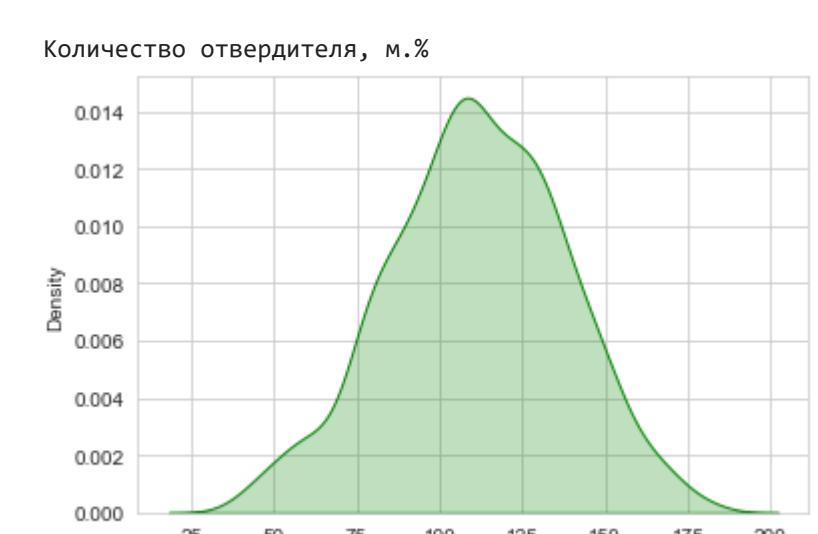
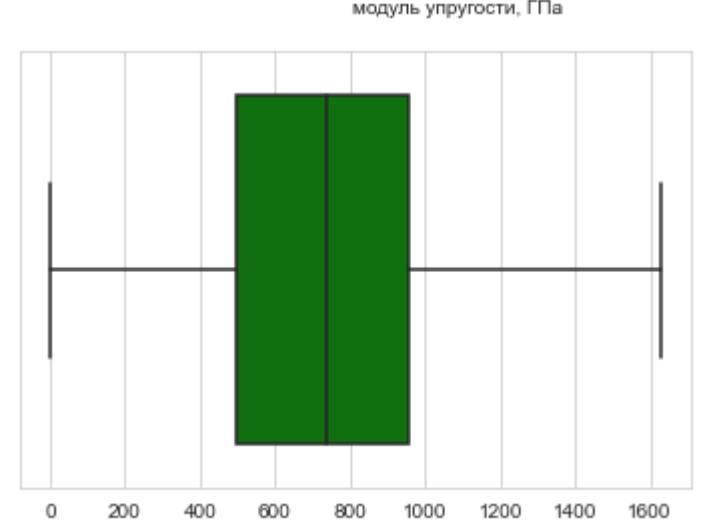
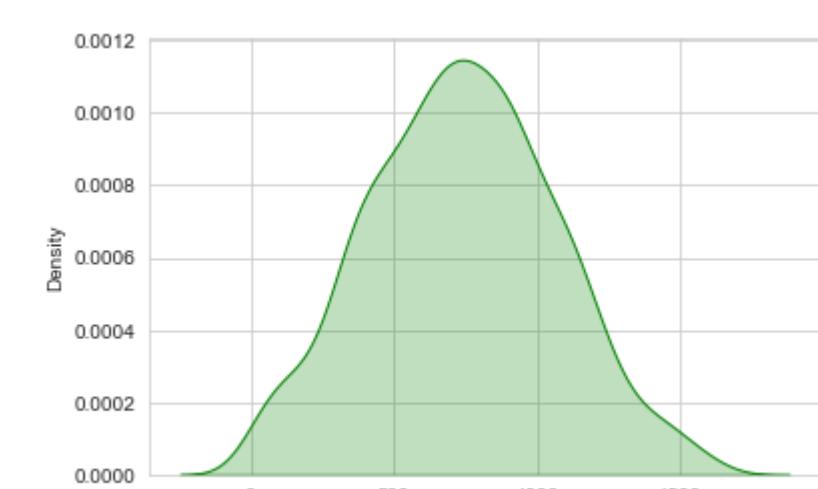


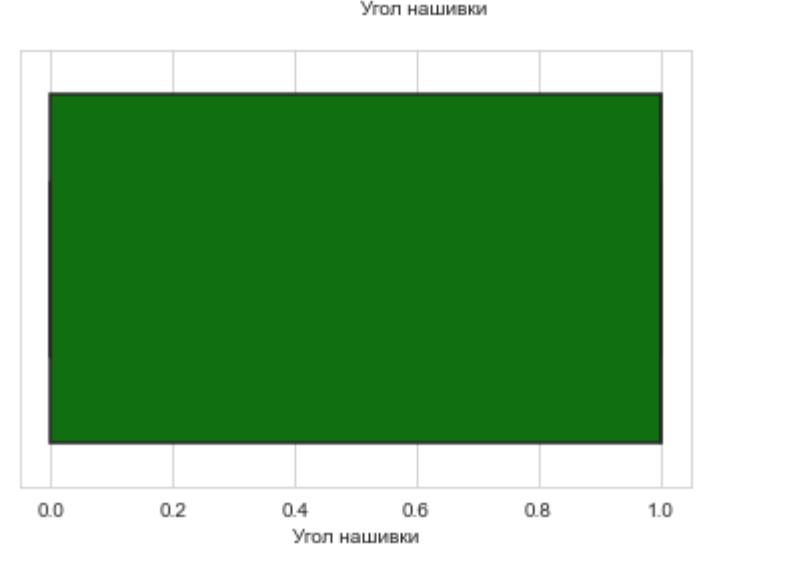
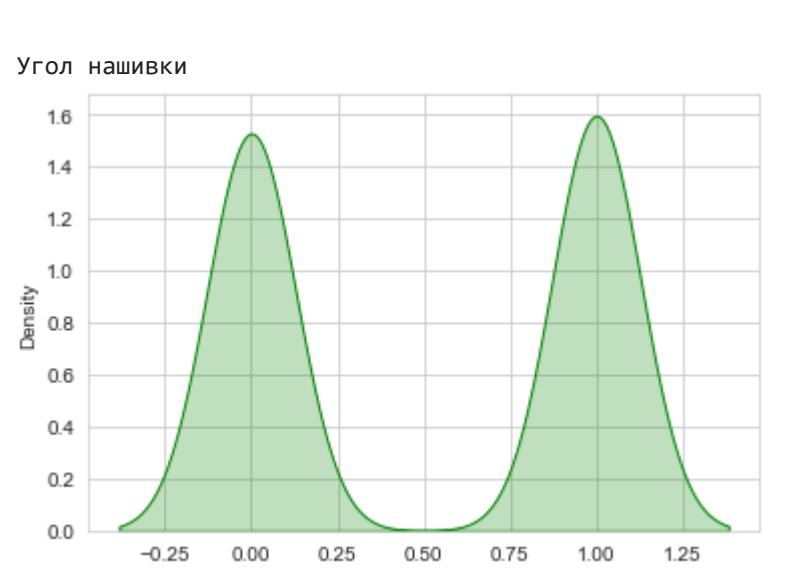
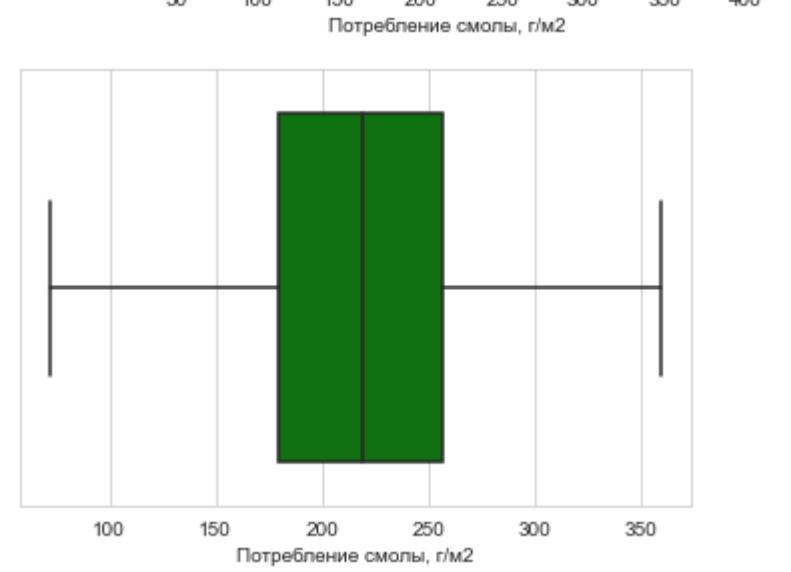
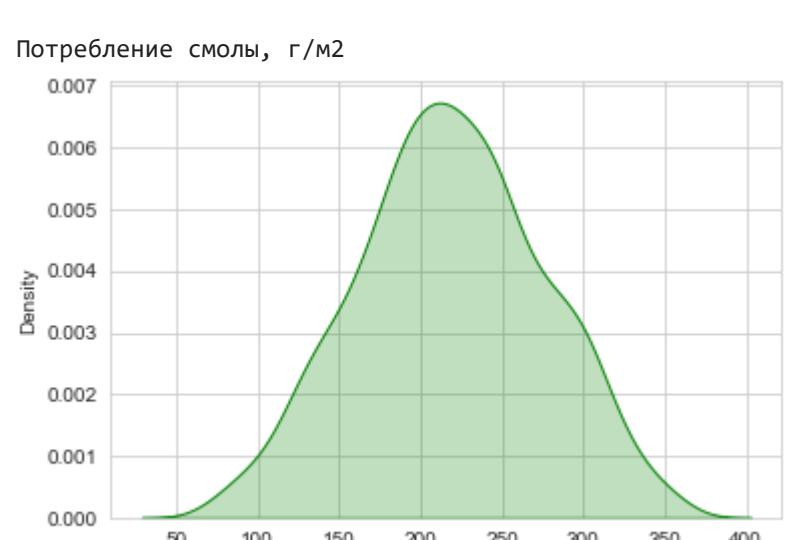
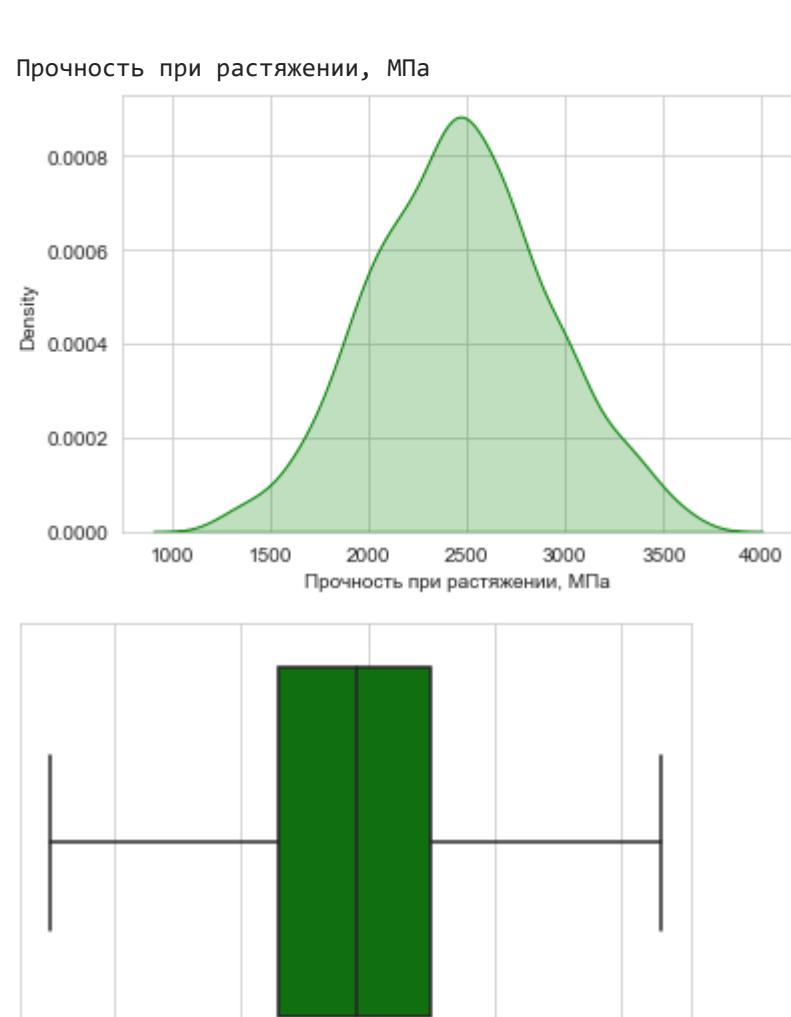
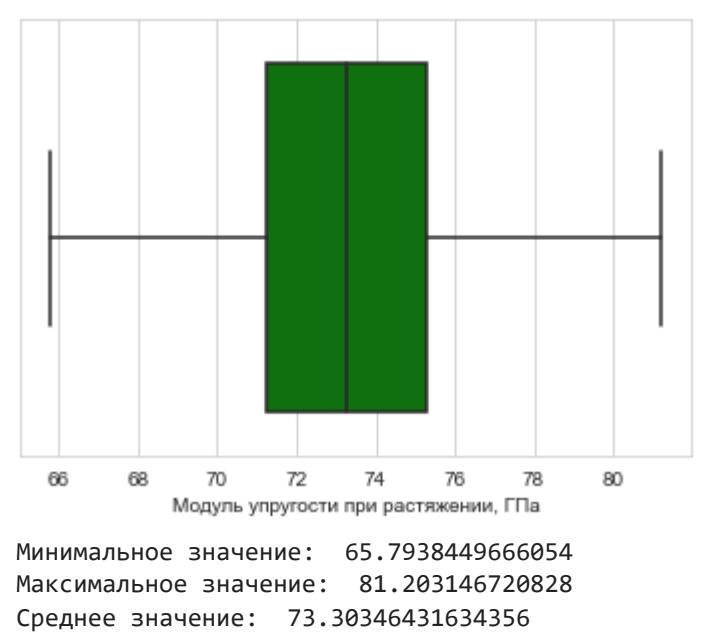
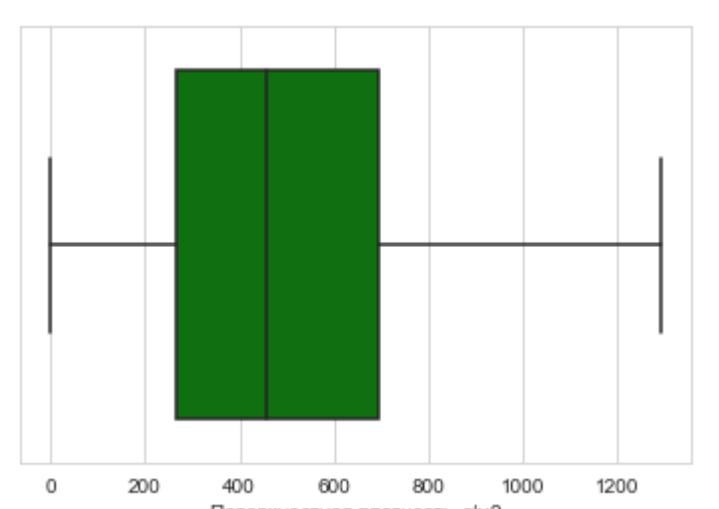
Минимальное значение: 0.547393809795624
Максимальное значение: 3.31413305151035
Среднее значение: 2.0279639384545688
Медианное значение: 2.987832228399705

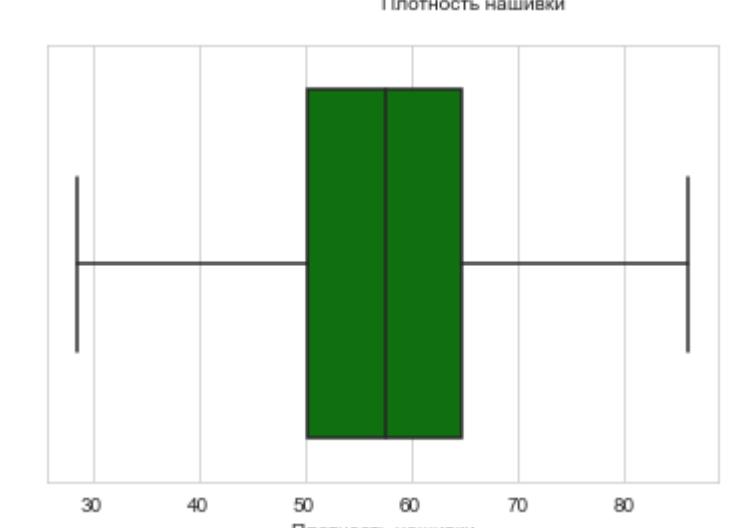
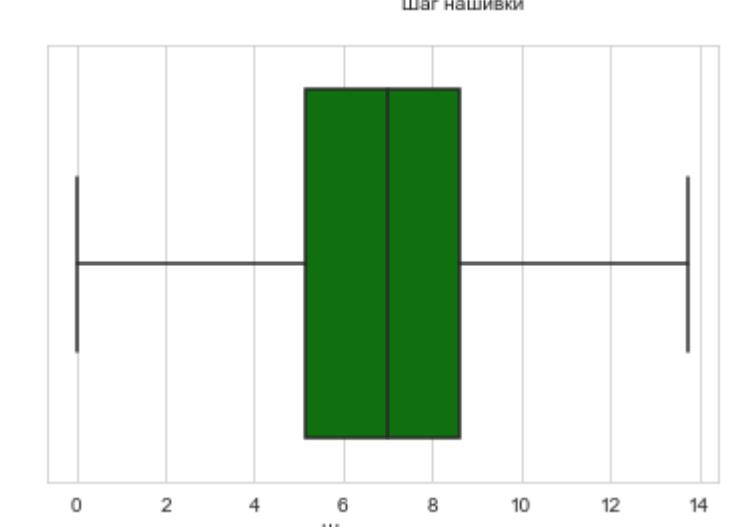
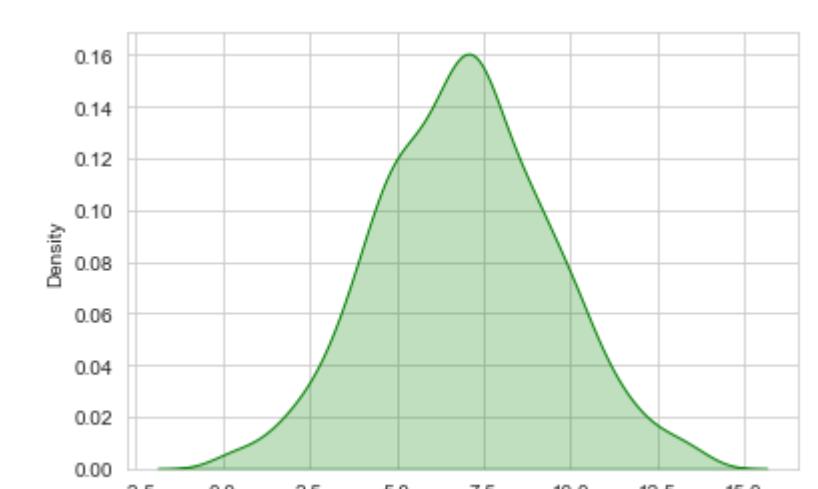


Минимальное значение: 1784.25424548658
Максимальное значение: 3652.68639388
Среднее значение: 1974.1187441112247
Медианное значение: 1977.3219816945

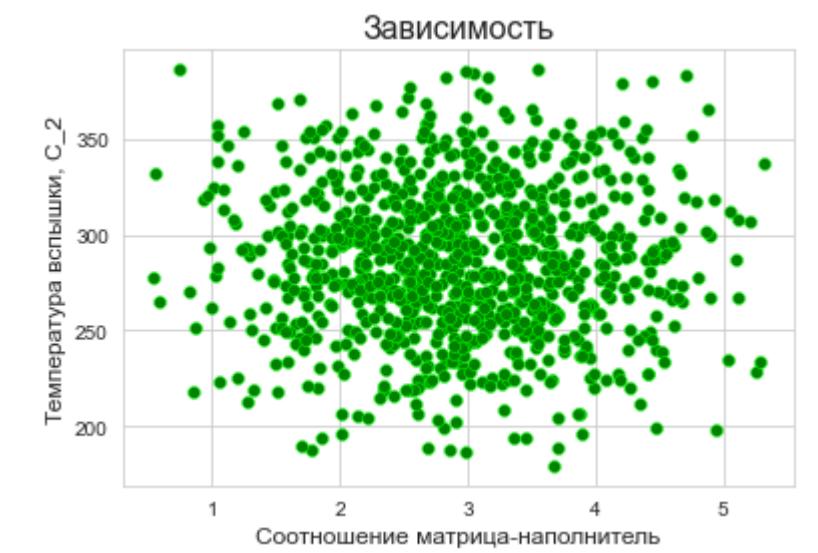
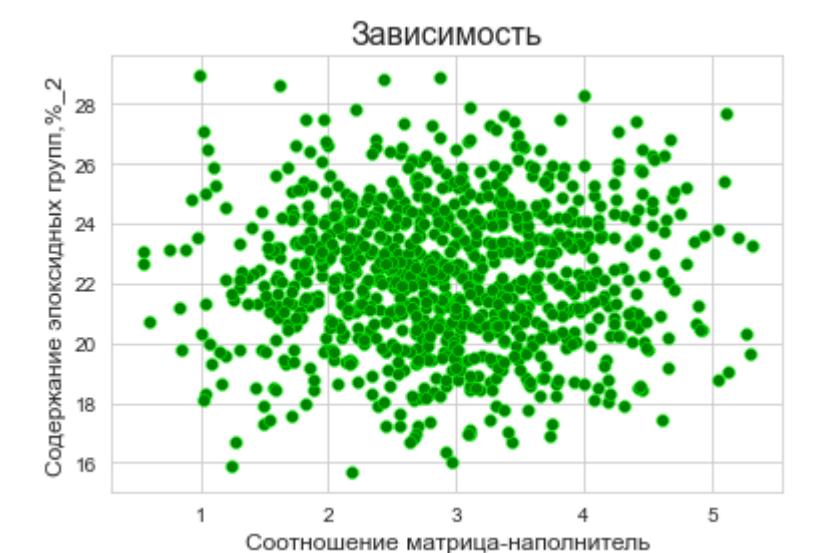
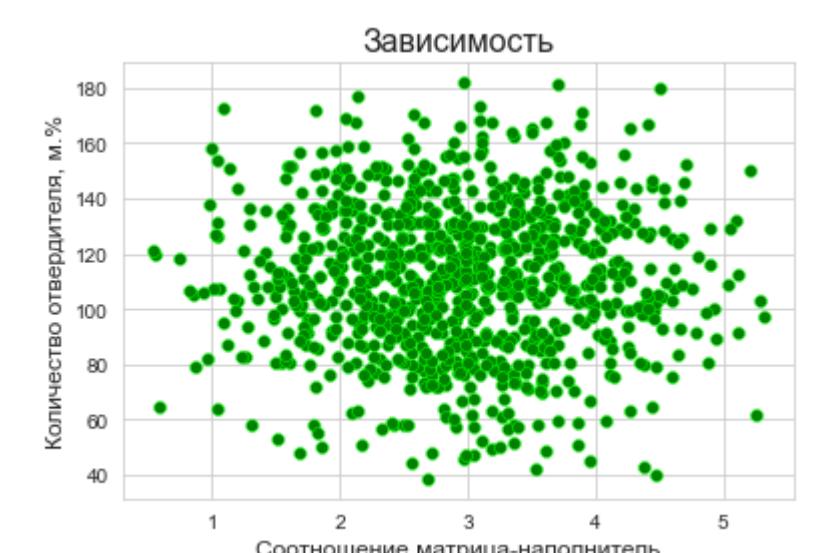
модуль упругости, ГПа

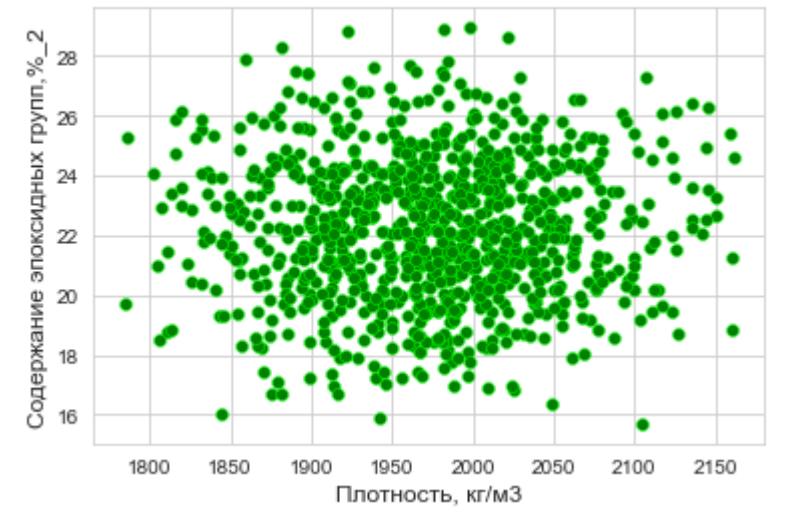
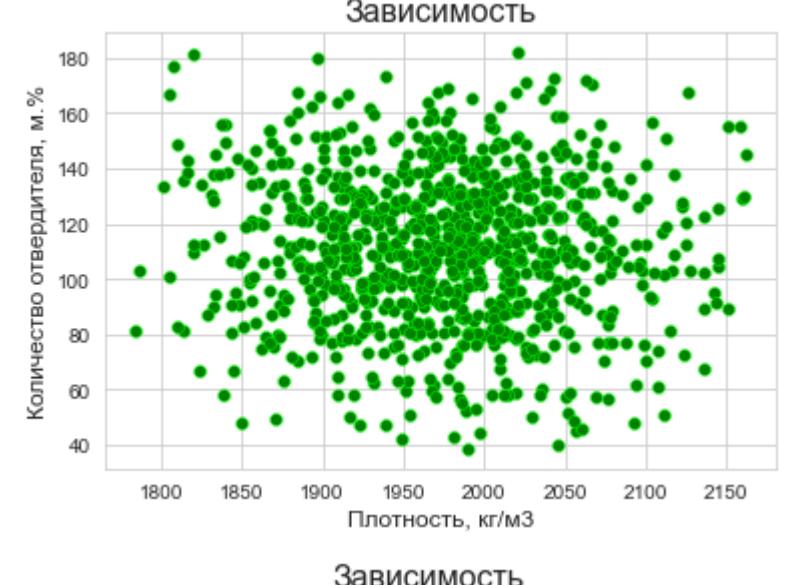
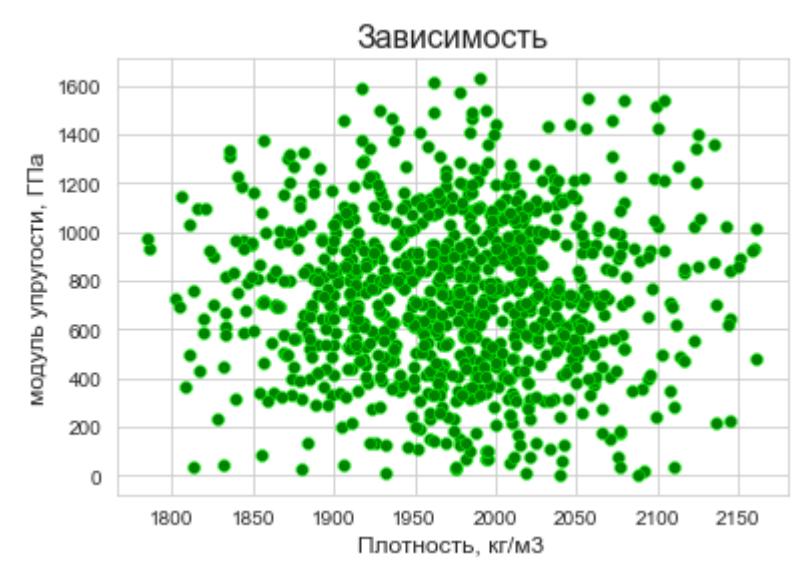
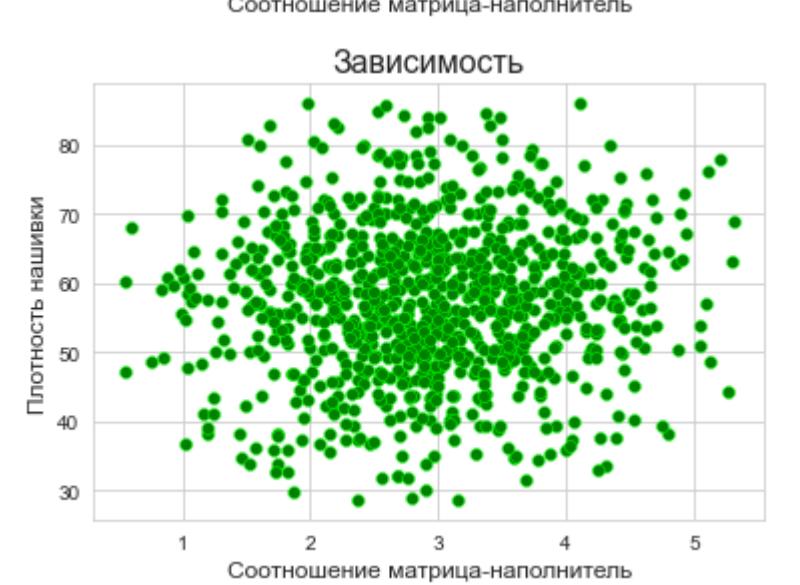
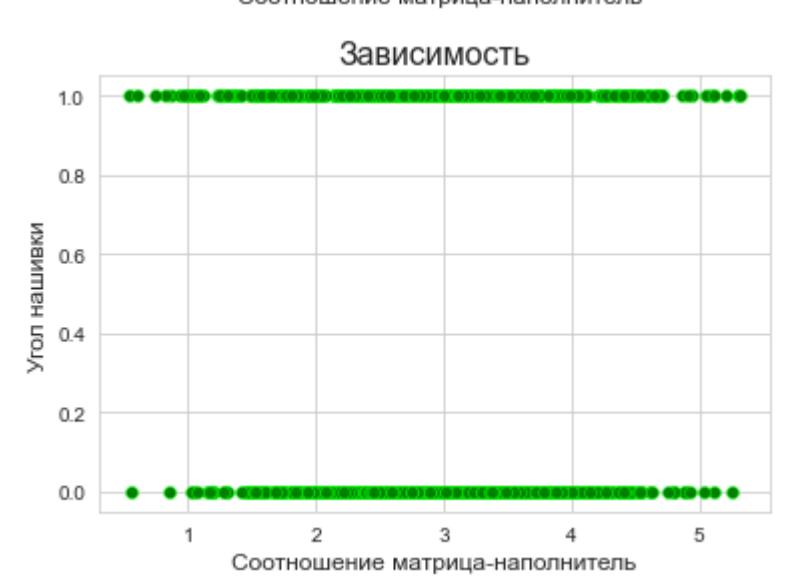
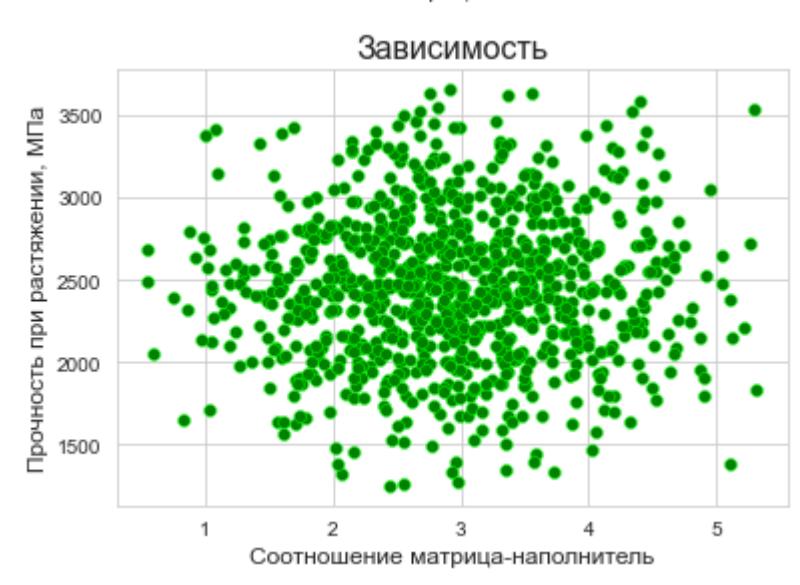
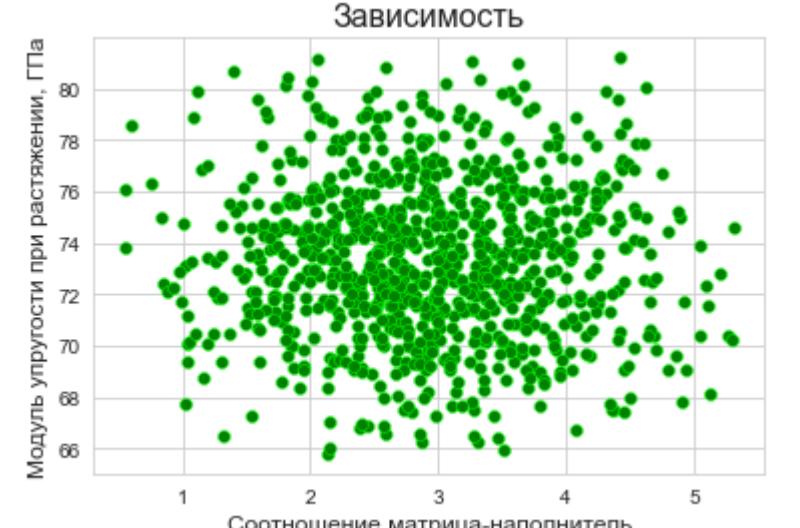
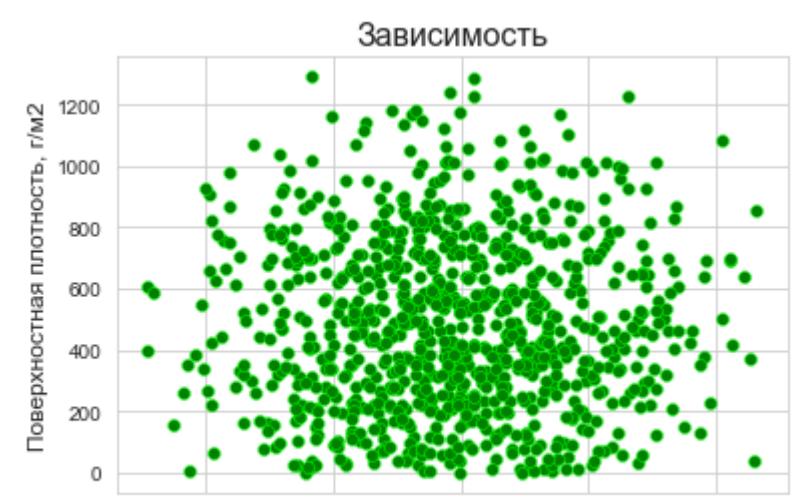


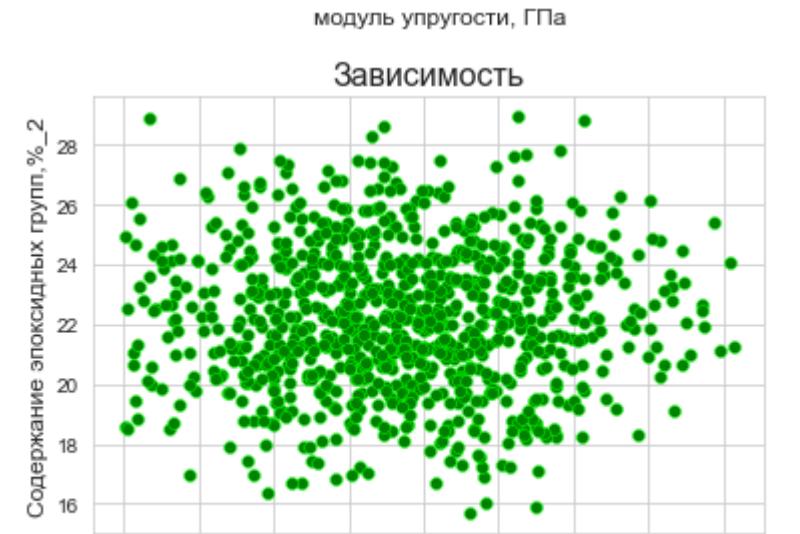
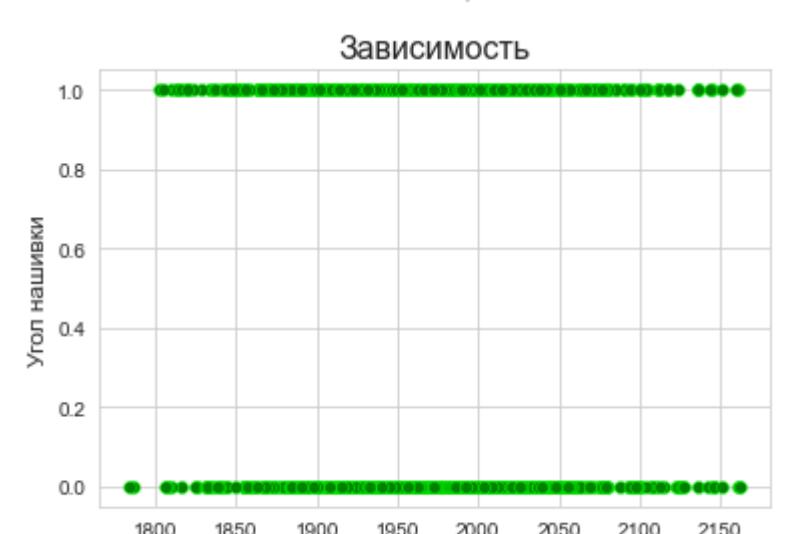
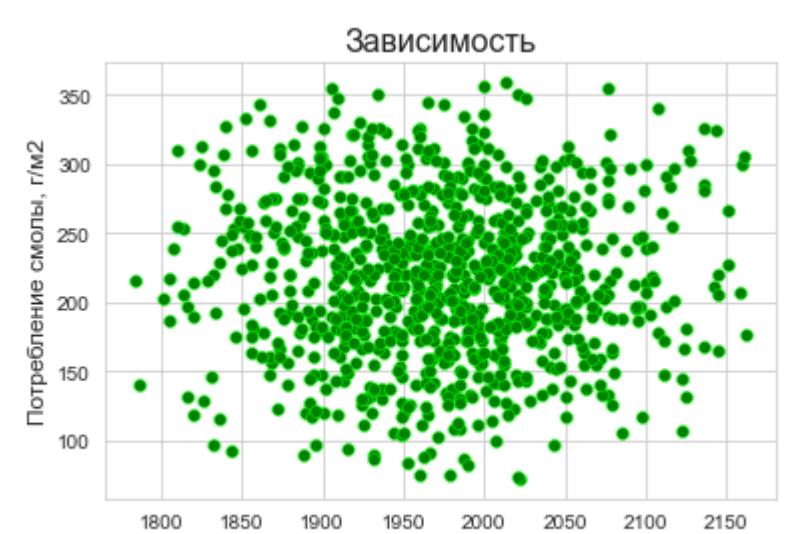
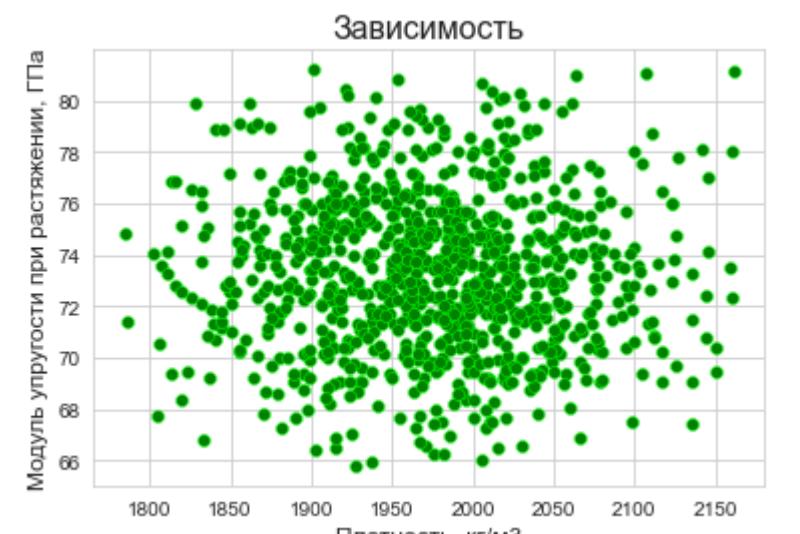
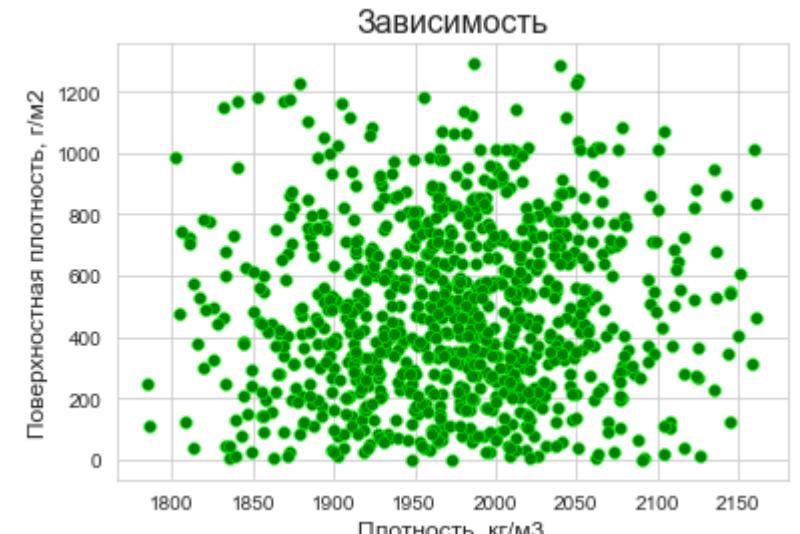
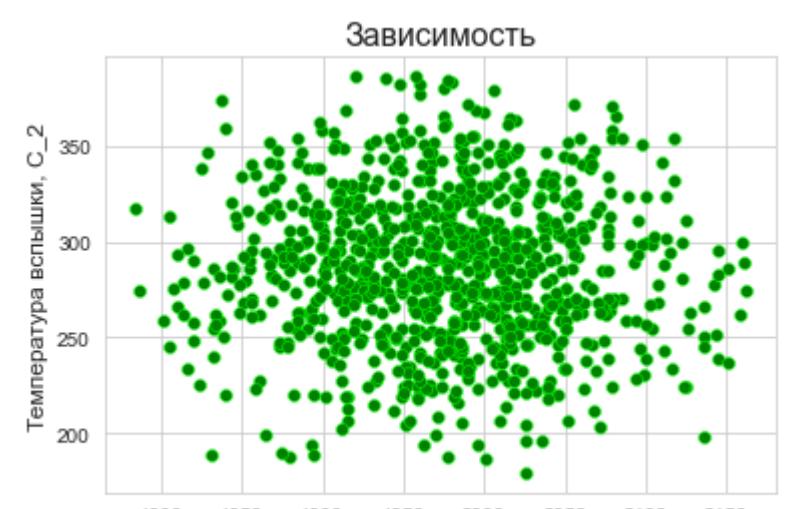


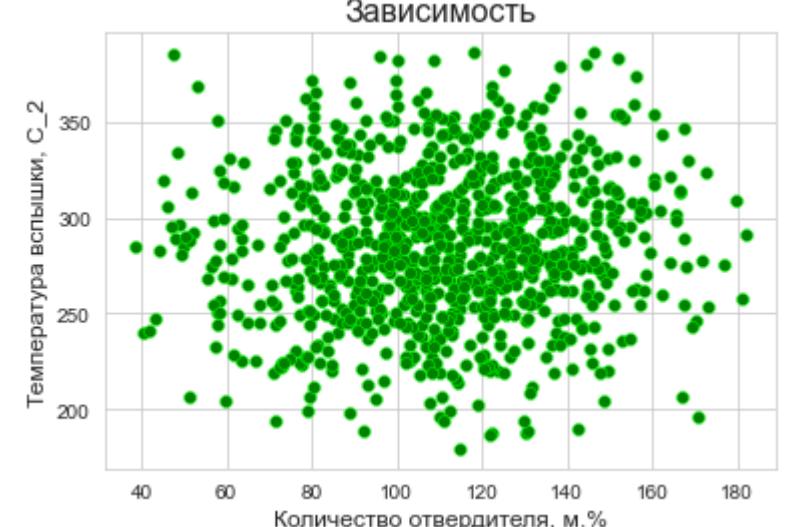
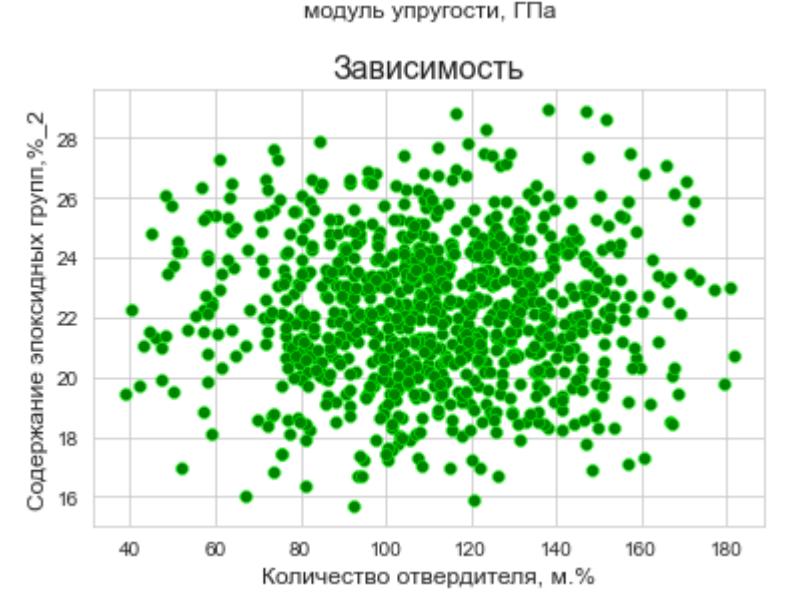
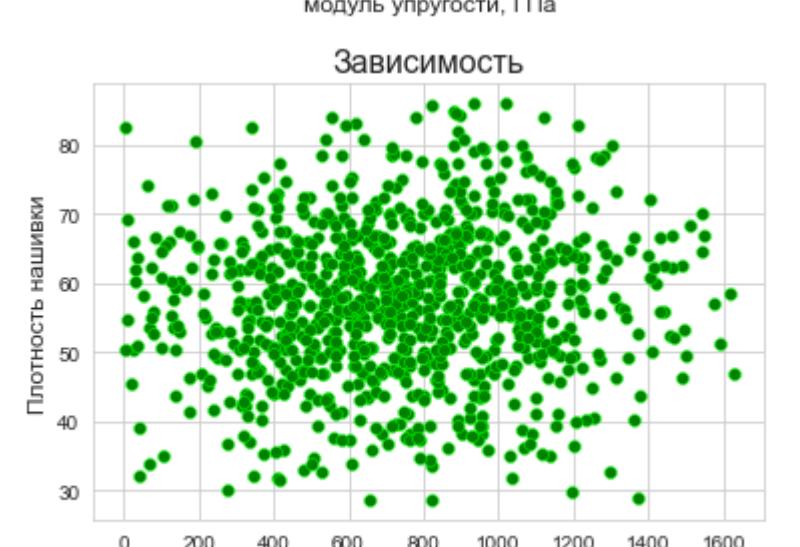
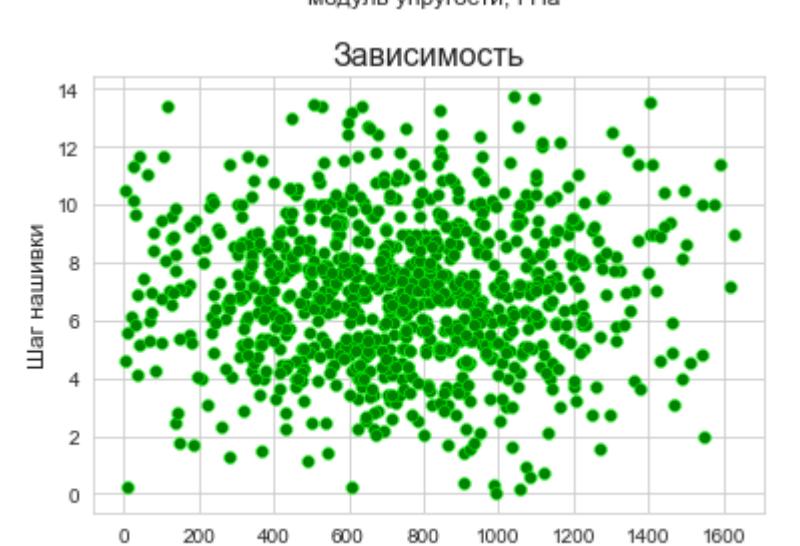
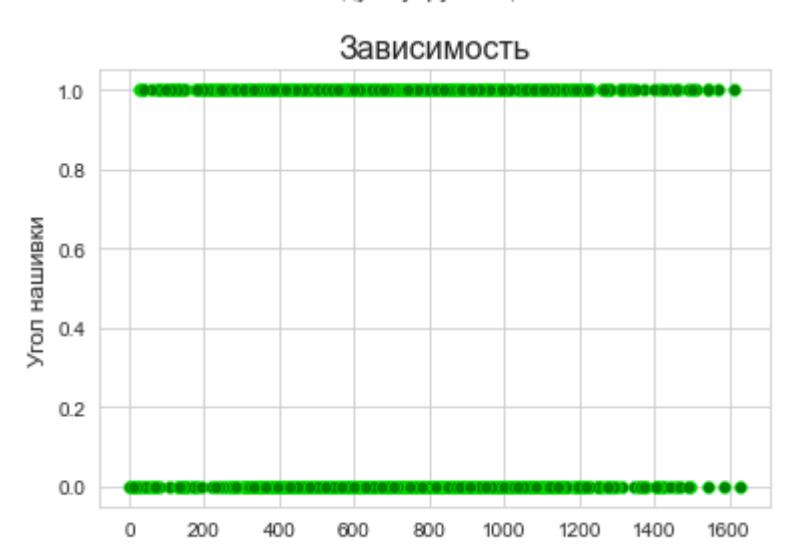
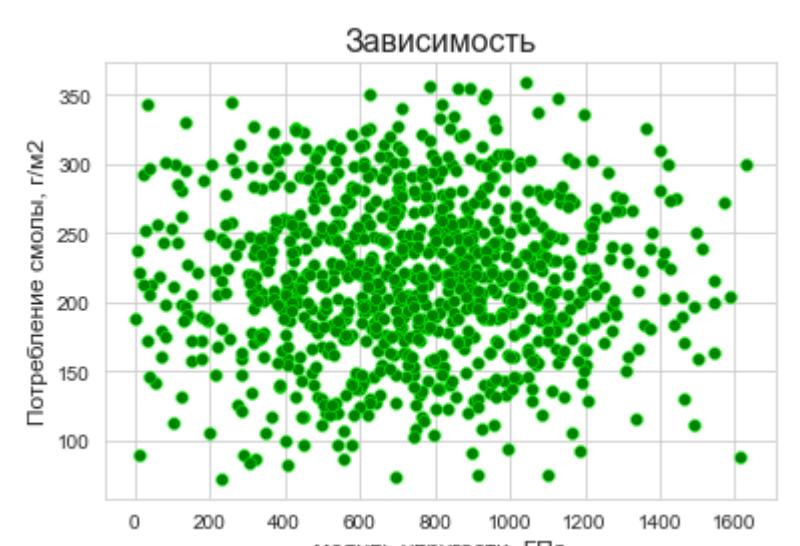
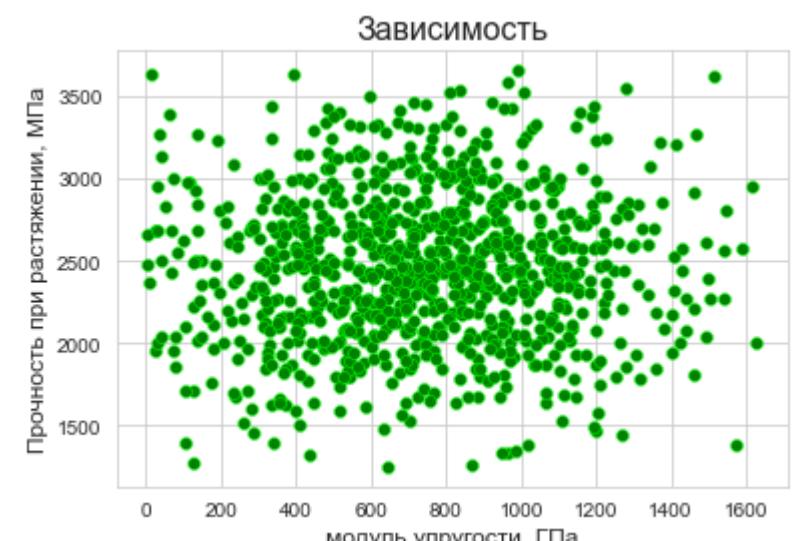
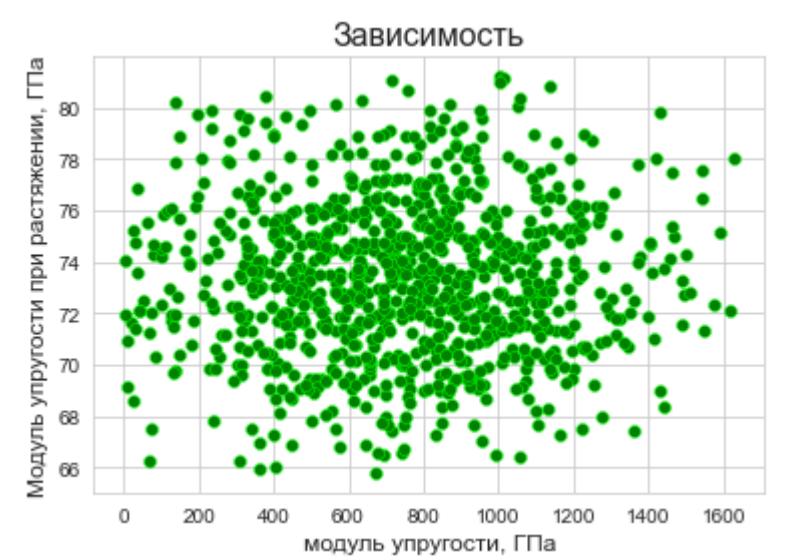
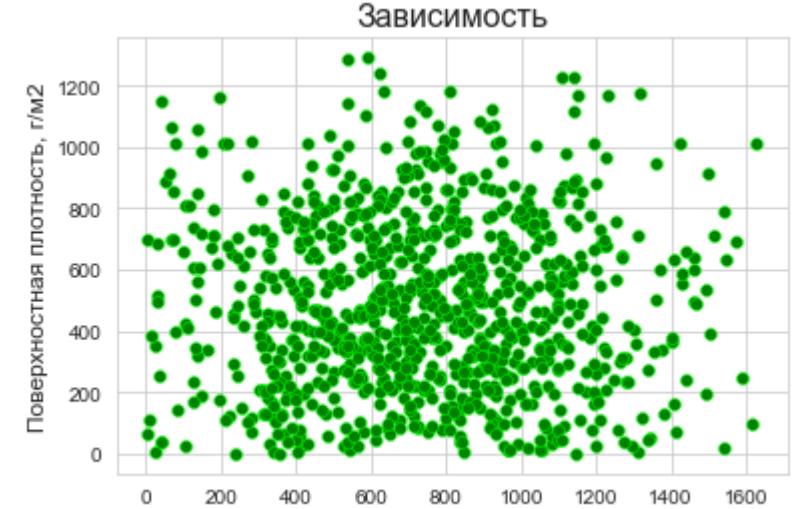
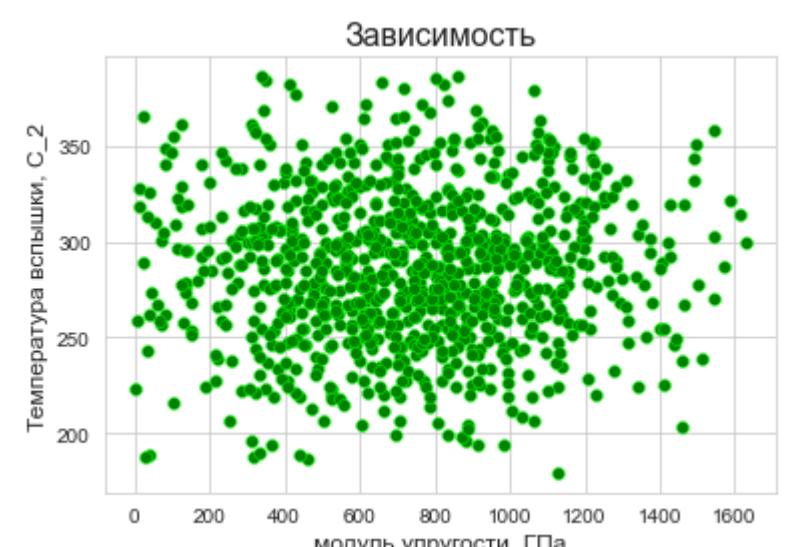


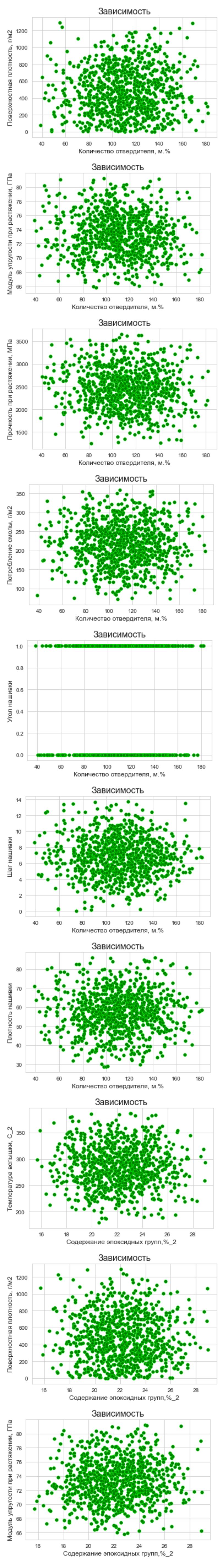
```
In [98]: n = 0
while n < len(column_names):
    b = n + 1
    while b < len(column_names):
        sns.set(style='whitegrid')
        plt.title('Зависимость', size = 16)
        plt.xlabel(column_names[n], size = 12)
        plt.ylabel(column_names[b], size = 12)
        sns.scatterplot(x = column_names[n], y = df, color = "green", edgecolor = 'lime', palette = 'cubehelix')
        plt.show()
        b += 1
    n += 1
```

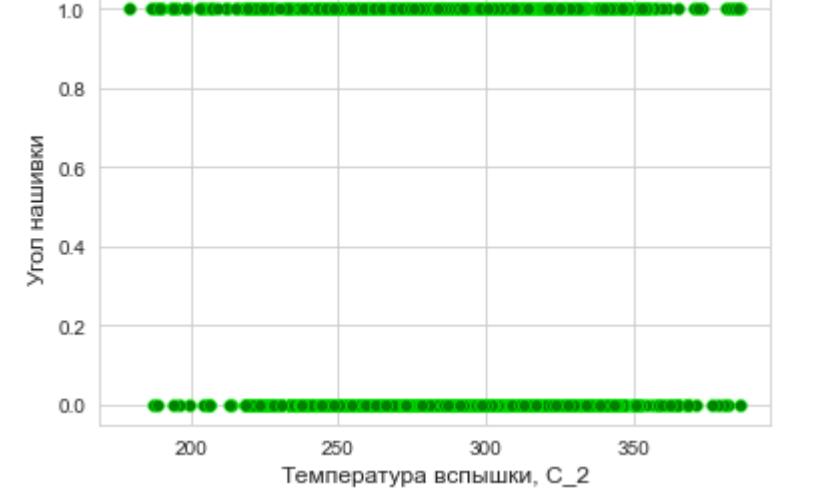
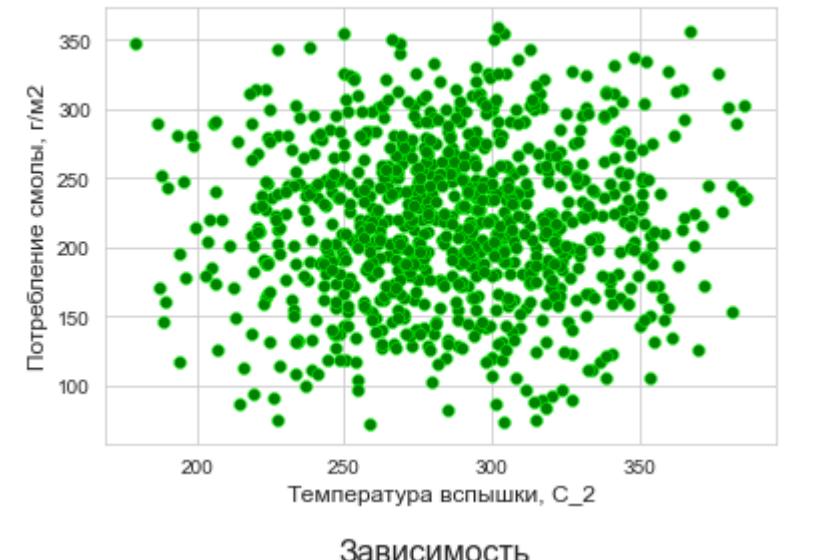
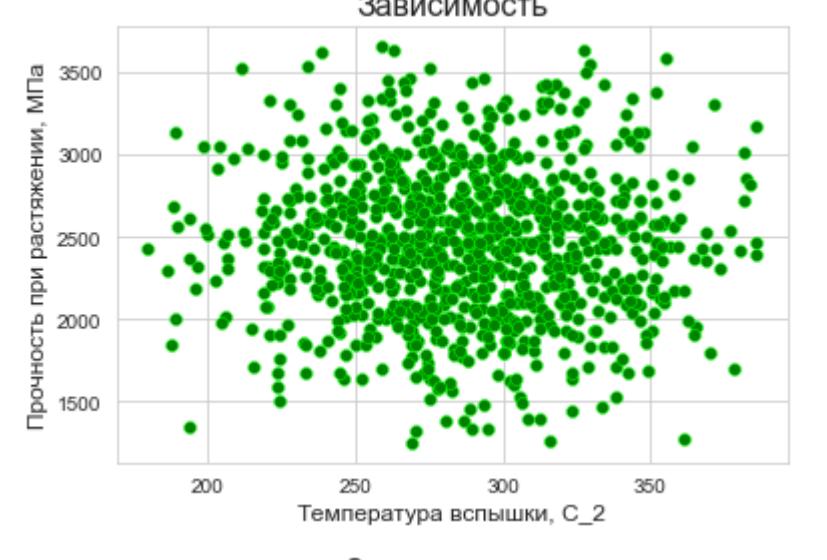
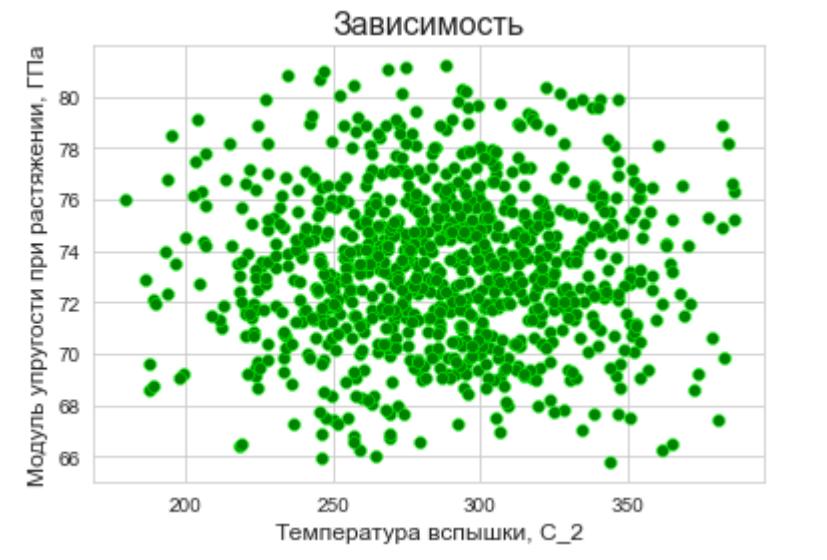
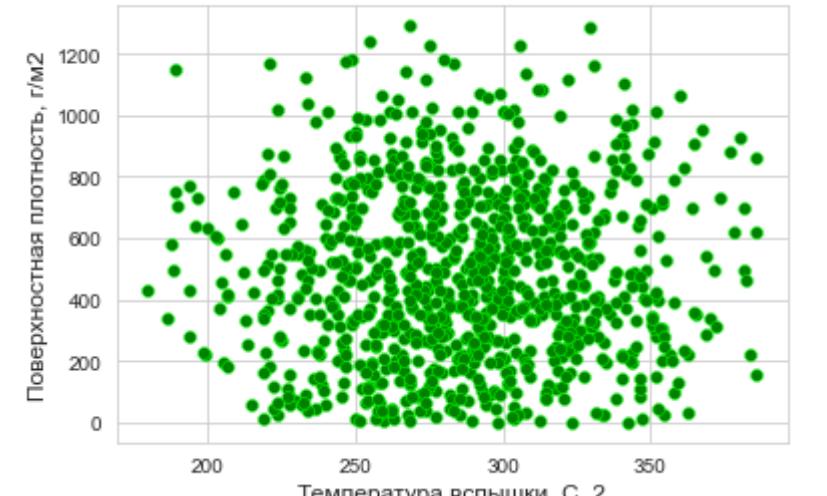
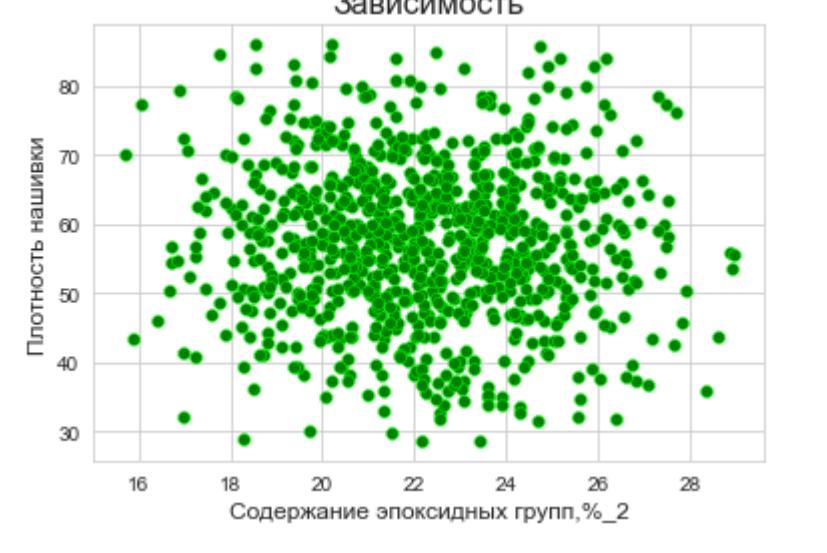
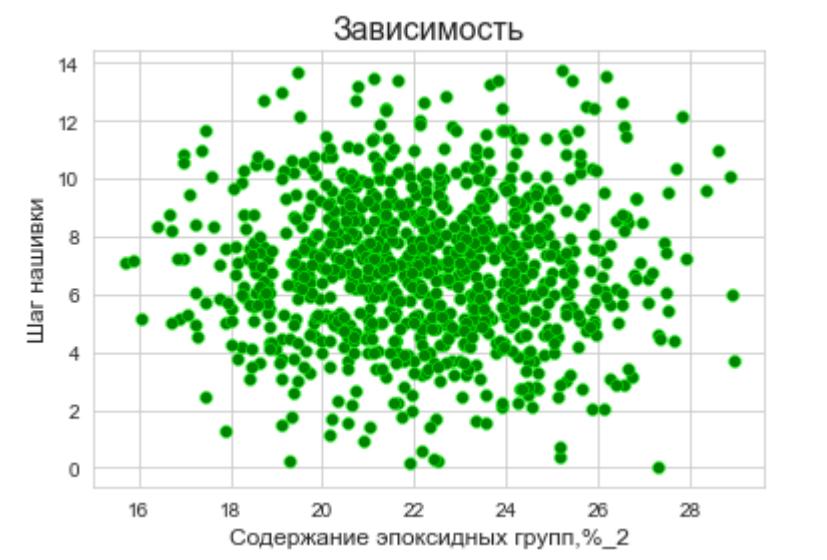
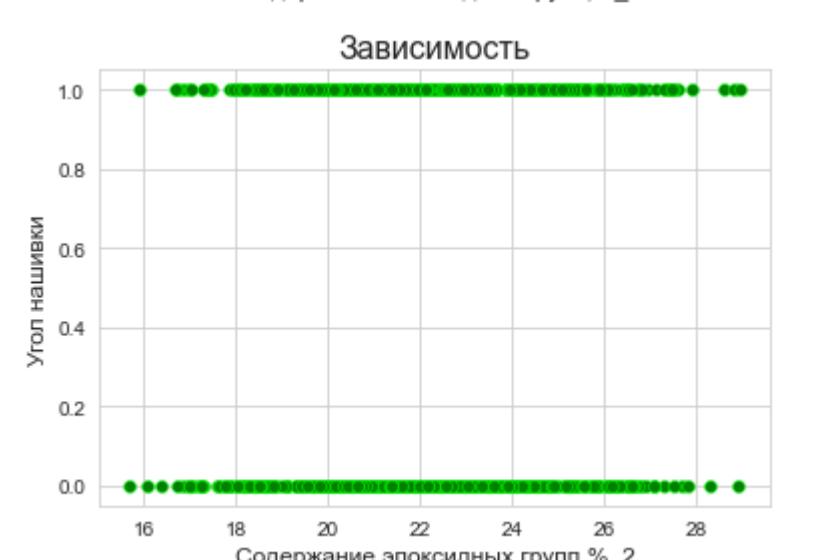
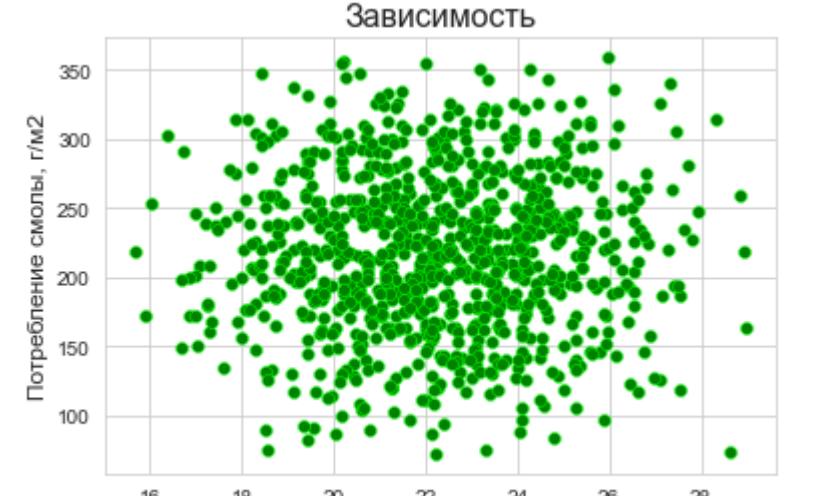
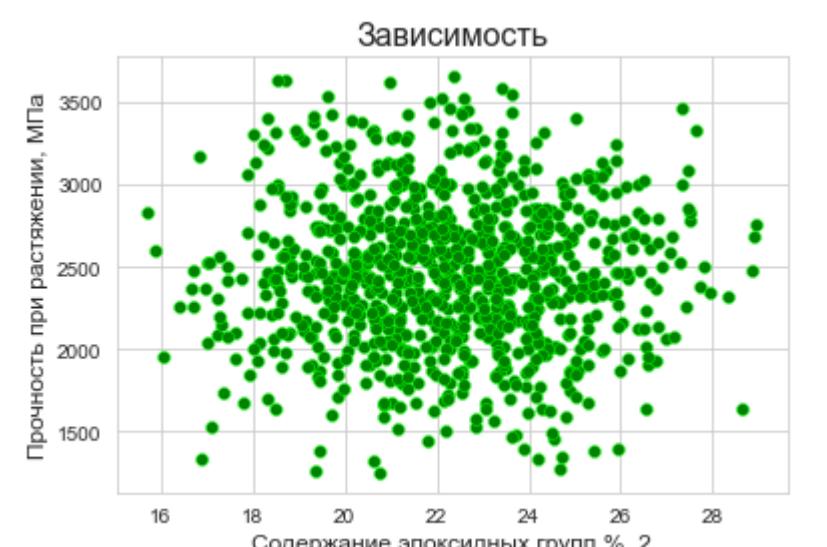


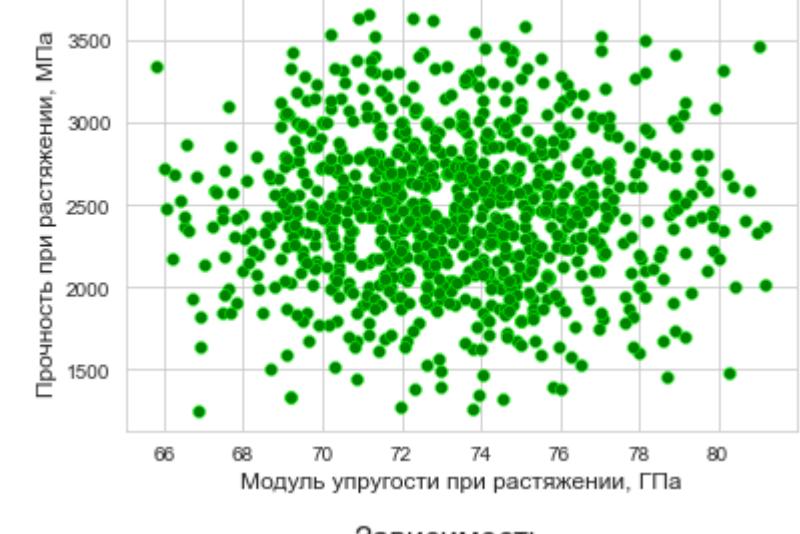
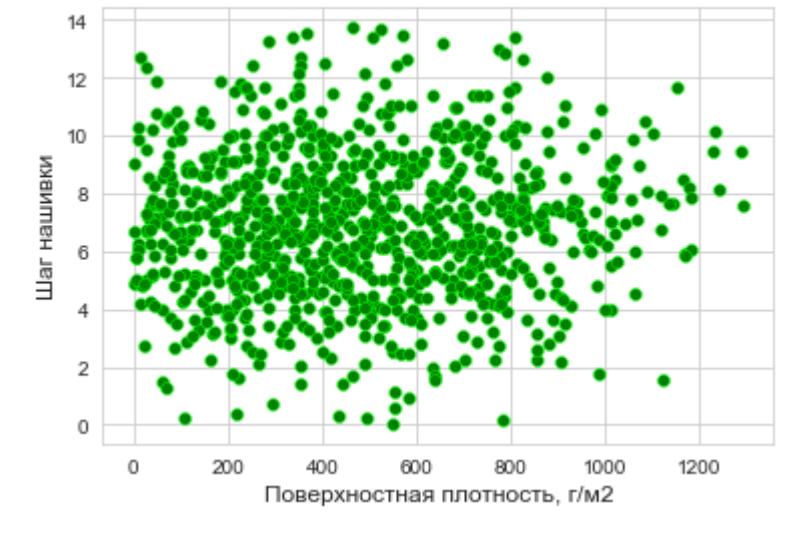
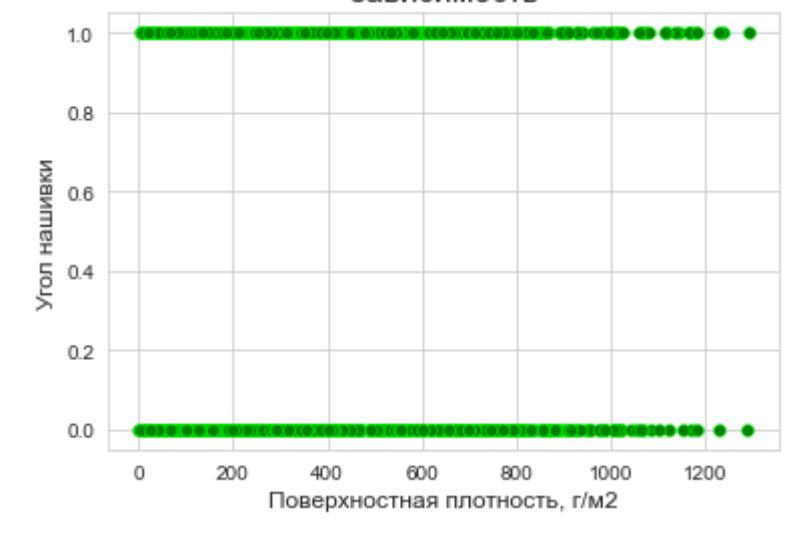
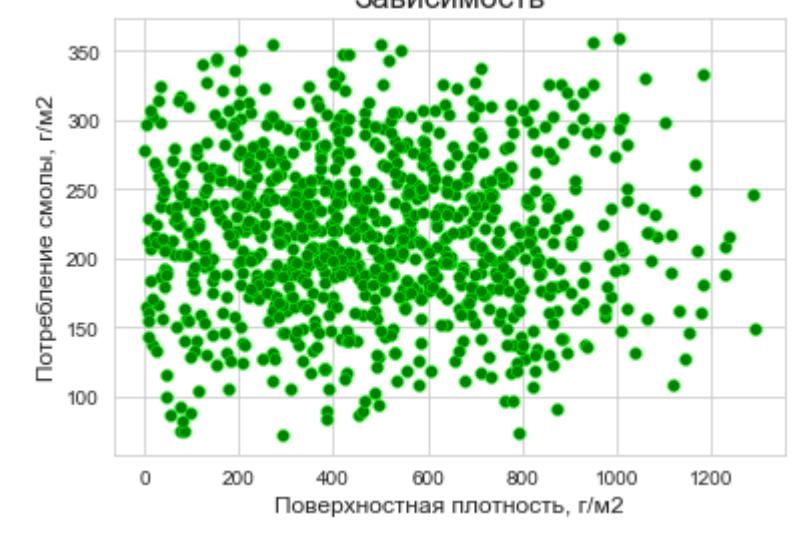
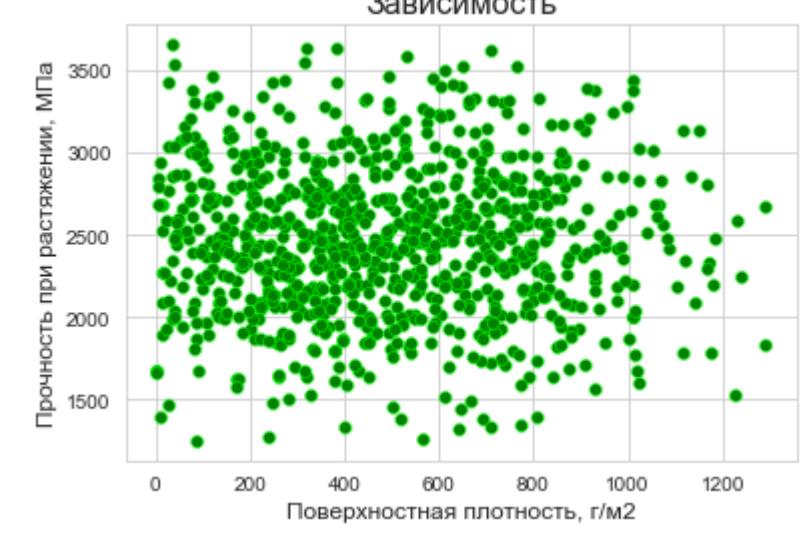
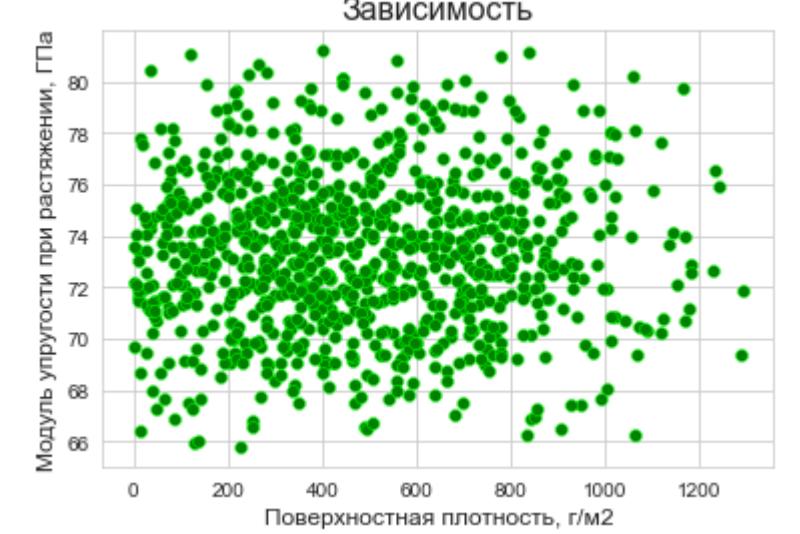
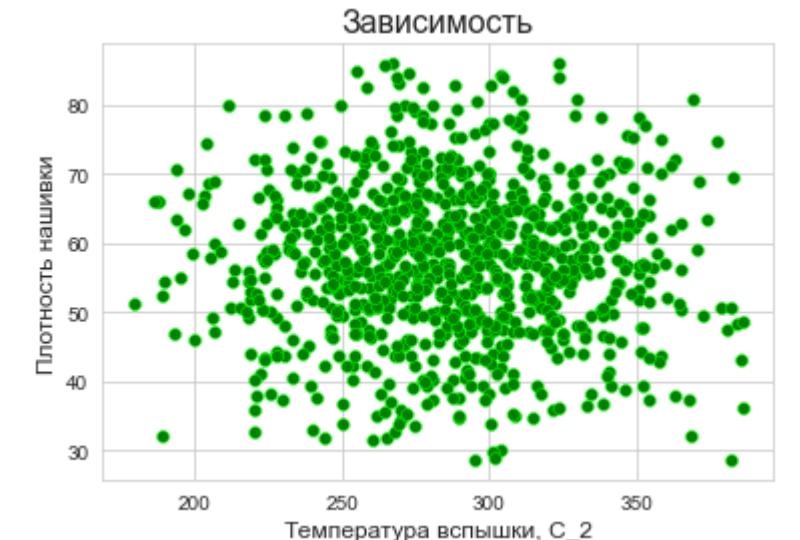
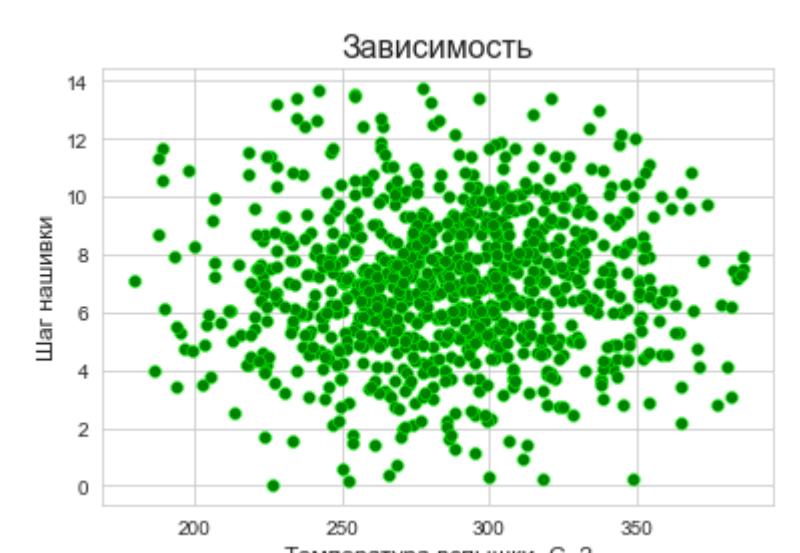


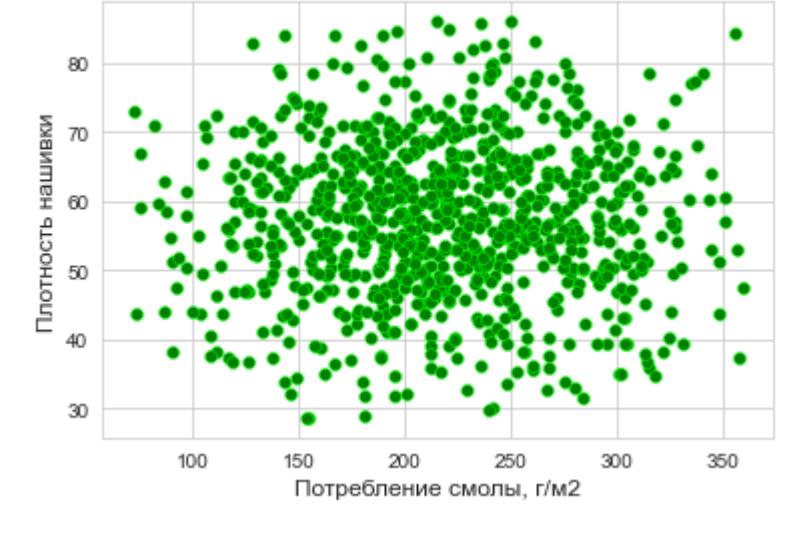
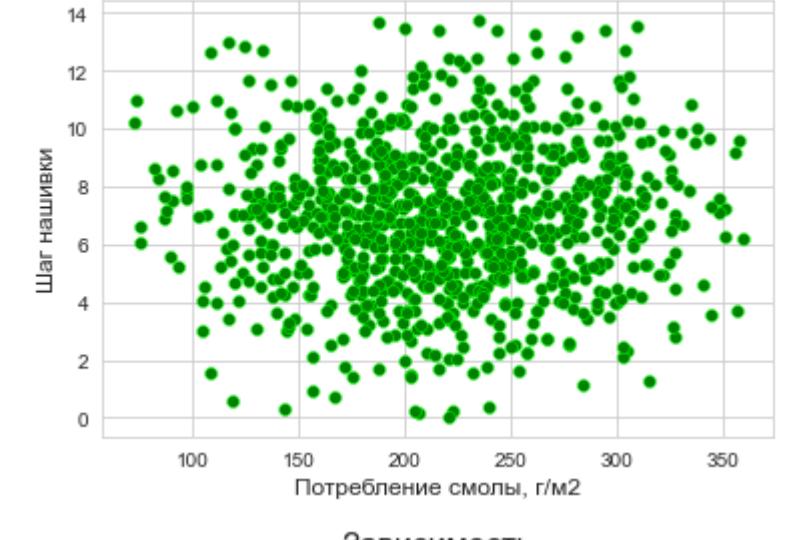
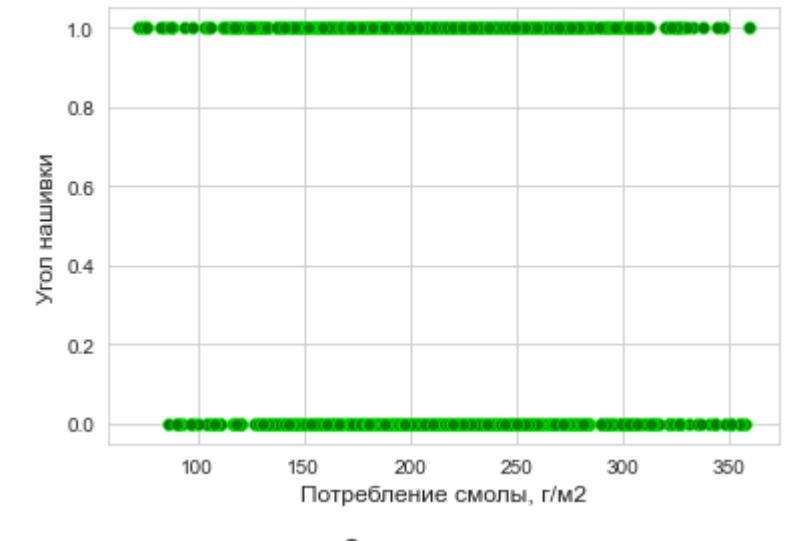
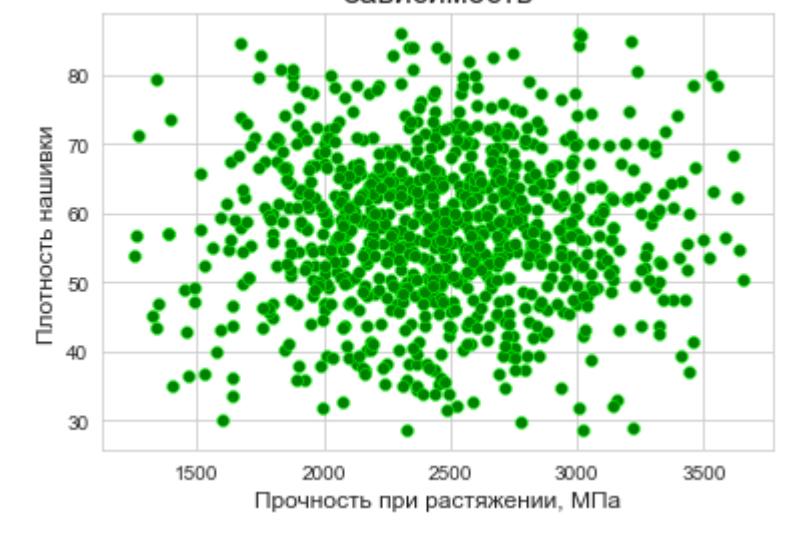
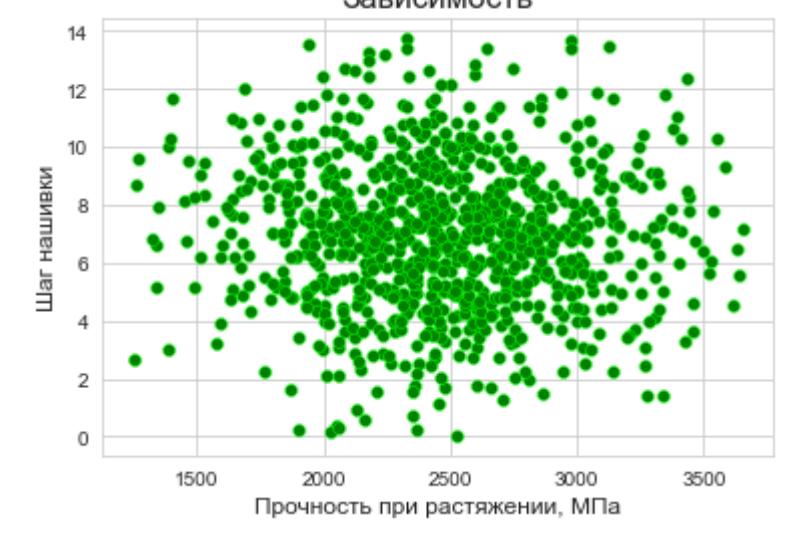
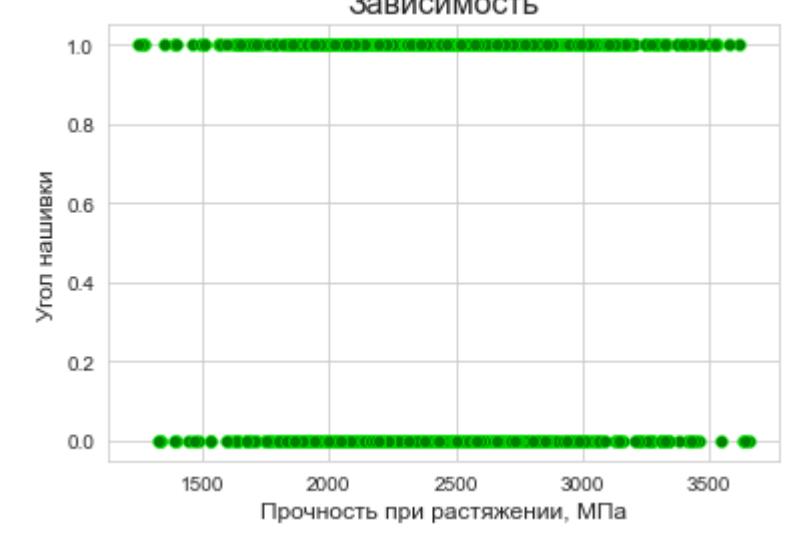
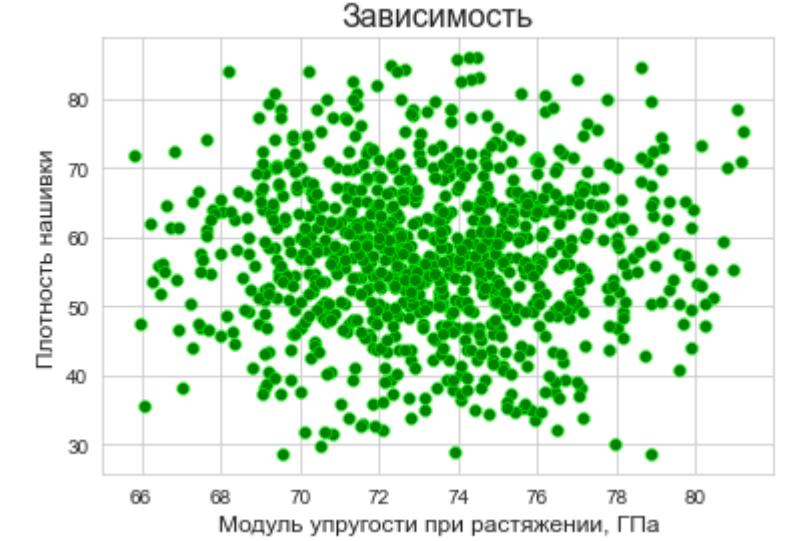
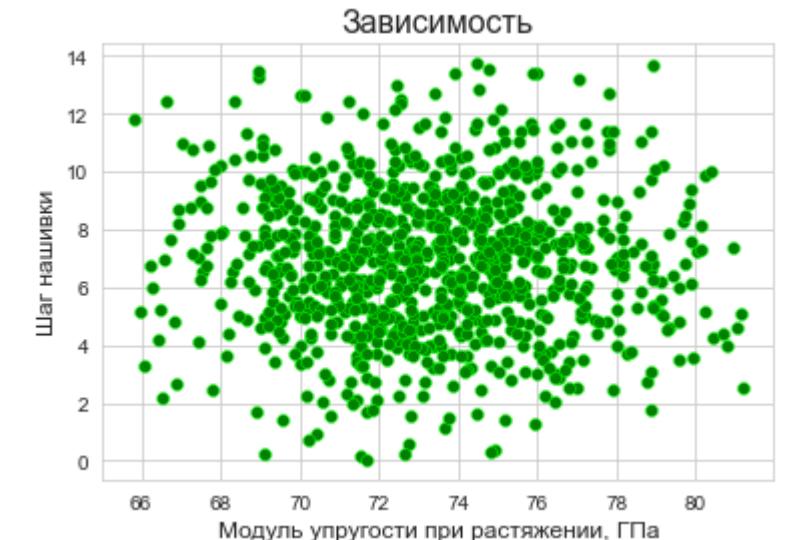


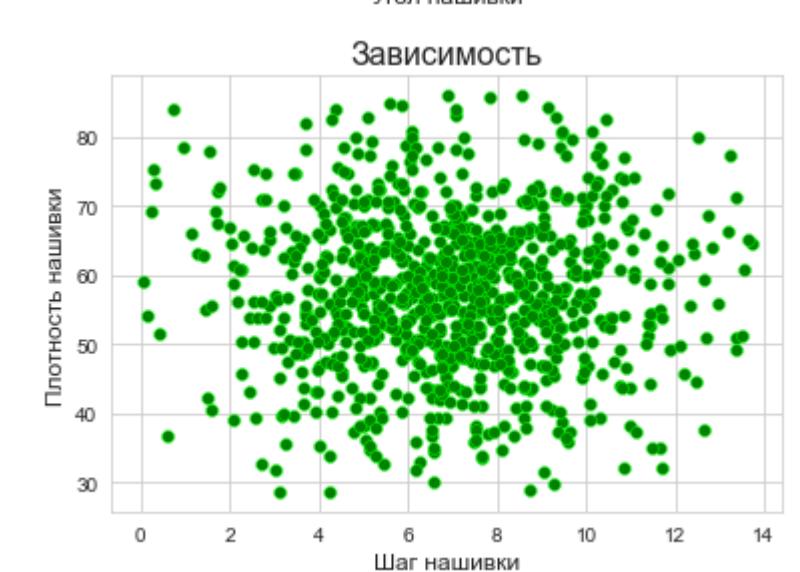
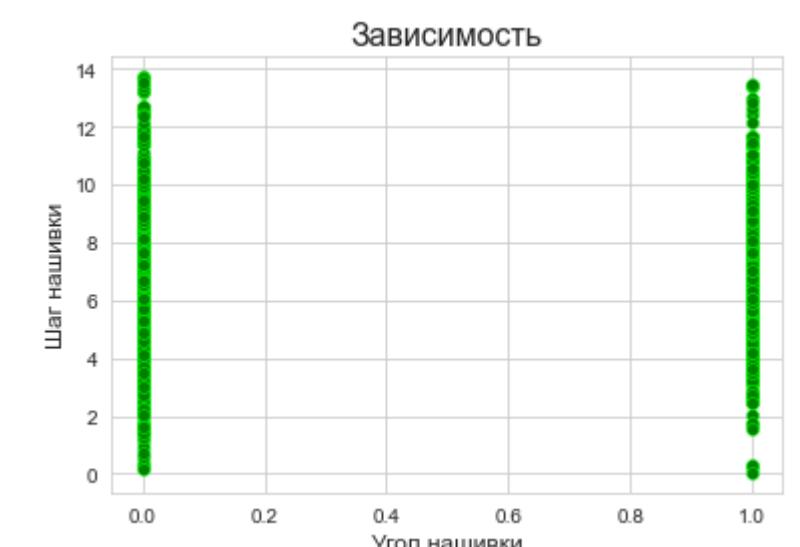






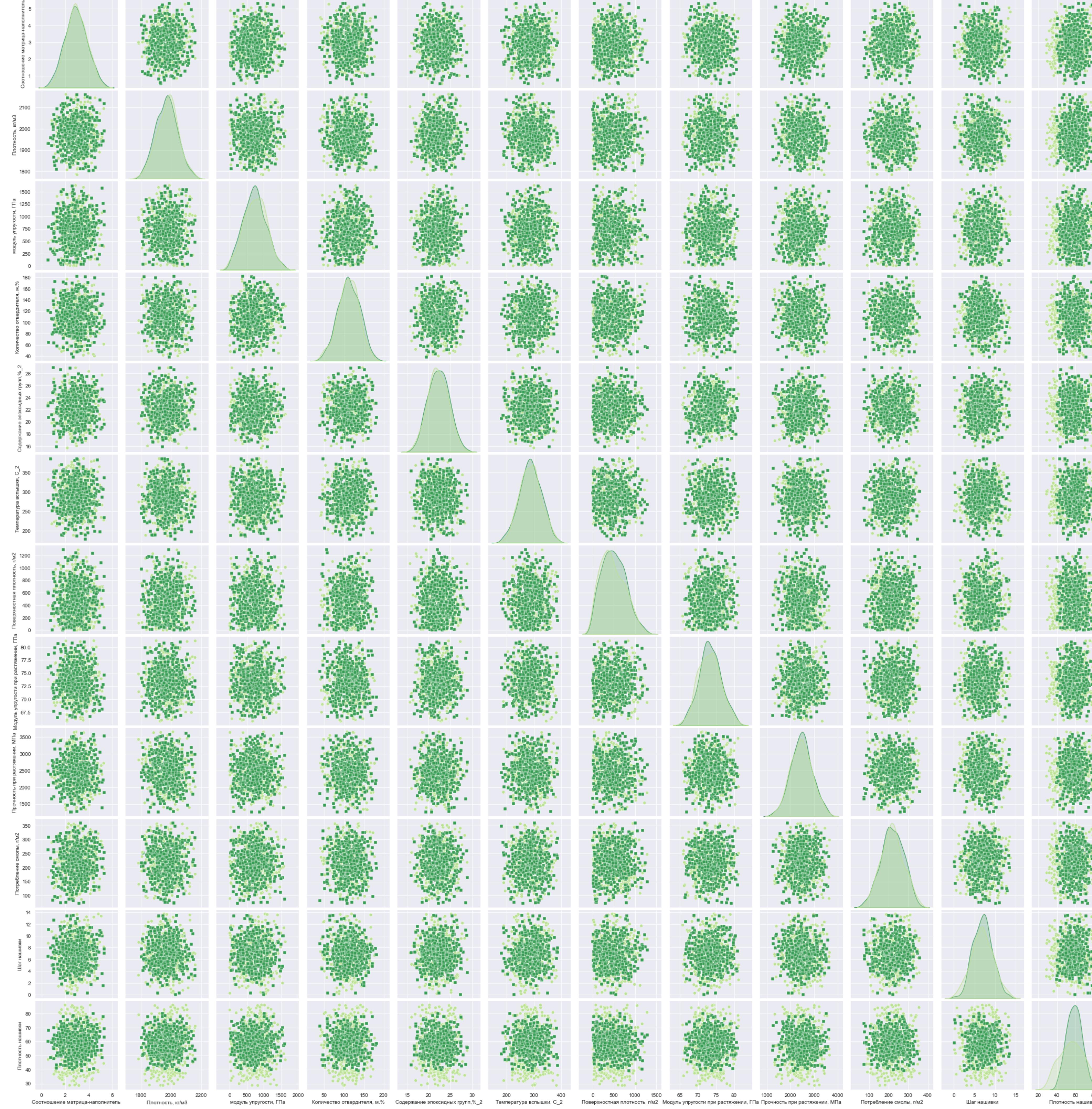






In [91]: sns.set_style('darkgrid')
sns.pairplot(df, hue = 'Угол нашивки', markers = ["o", "s"], diag_kind = 'auto', palette = 'YIGn')

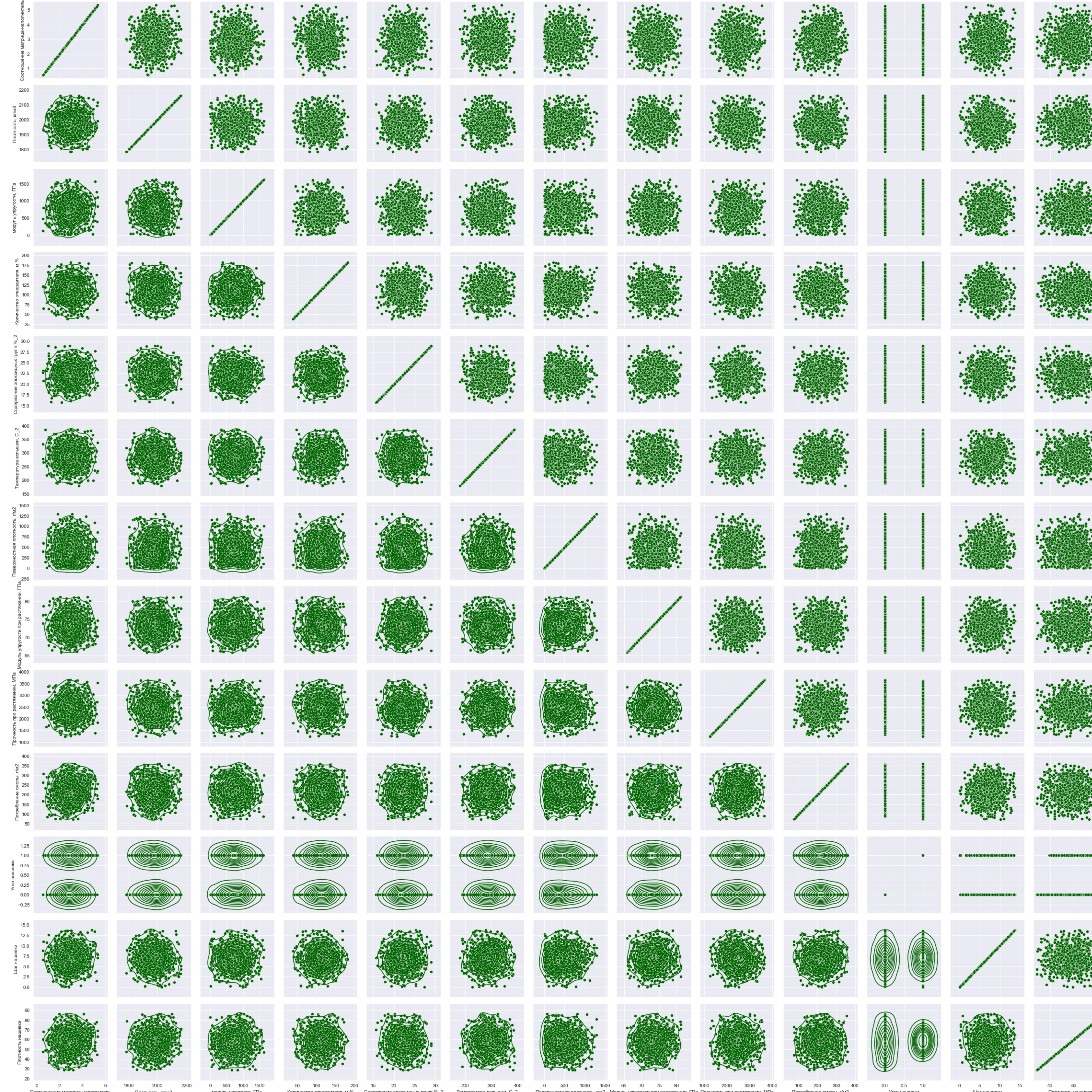
Out[91]: <seaborn.axisgrid.PairGrid at 0x1dc47c19df0>



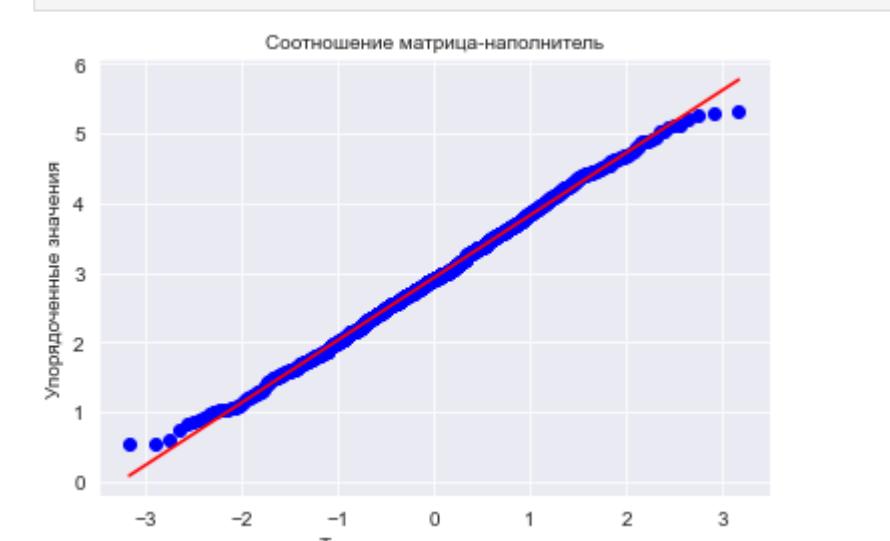
In [92]: # Порядковые графики рассеяния точек - скatterплоты (второй вариант)

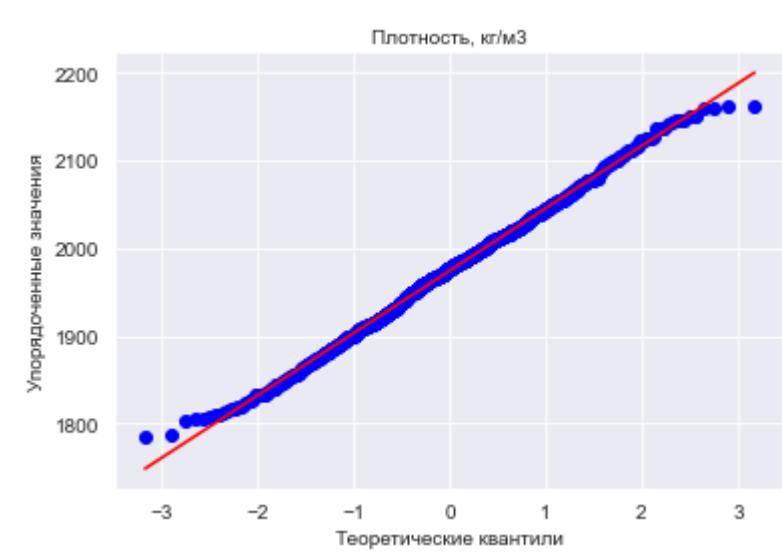
```
g = sns.PairGrid(df[df.columns])
g.map_upper(sns.scatterplot, color = "darkgreen")
g.map_lower(sns.kdeplot, color = "darkgreen")
g.map_diag(sns.kdeplot, color = "darkgreen")
plt.show()
# Копирование hem
```

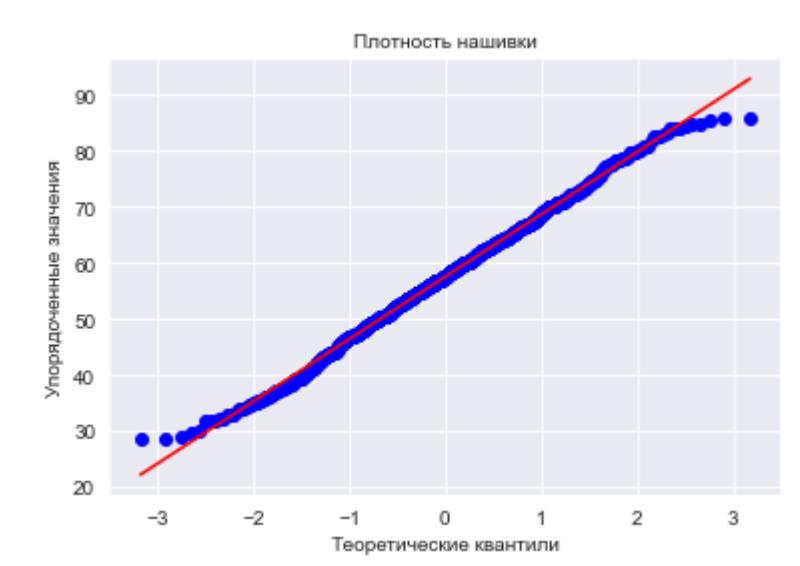
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



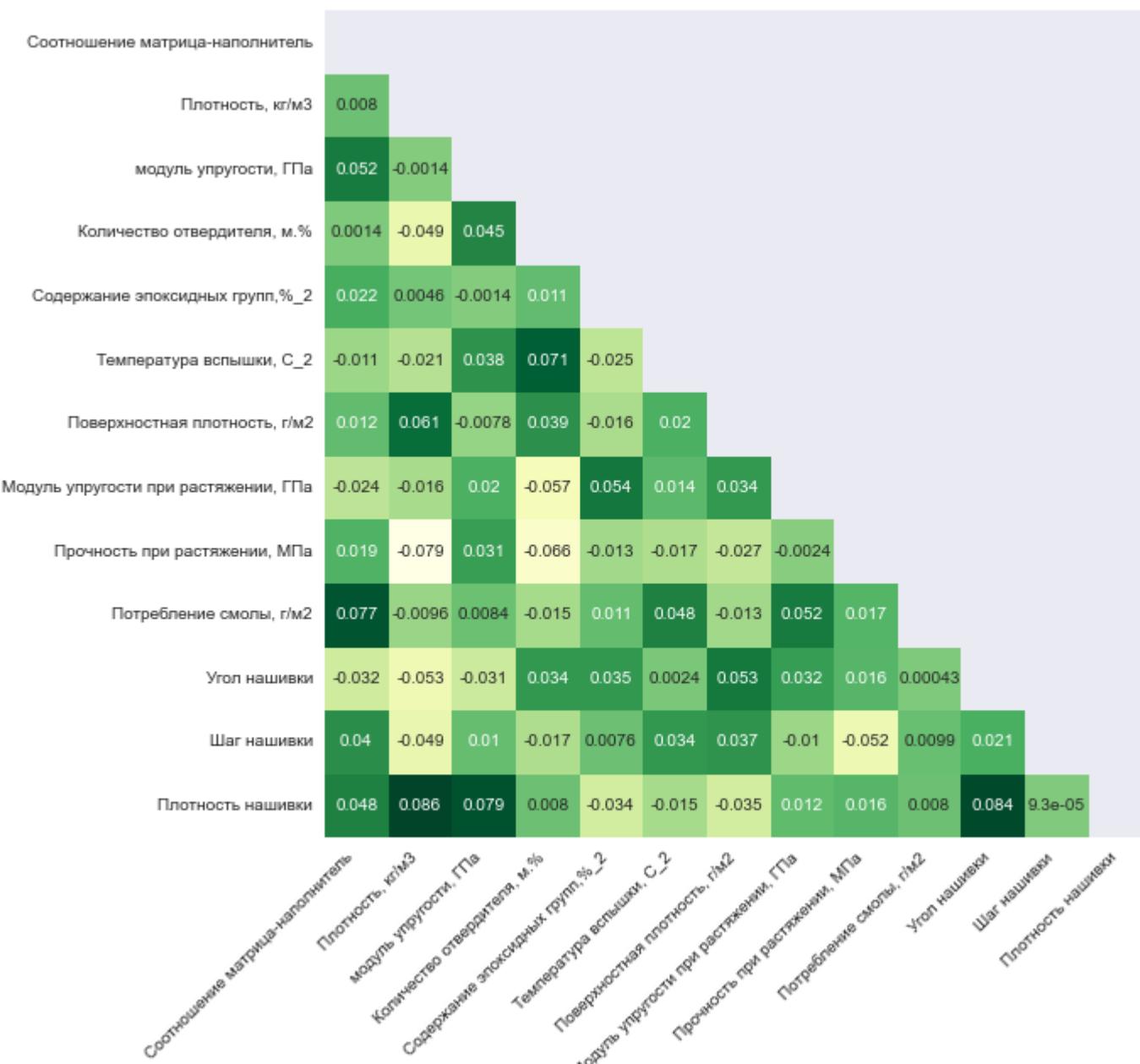
```
In [93]:
for i in df.columns:
    plt.figure(figsize = (6, 4))
    res = stats.probplot(df[i], plot = plt)
    plt.title(i, fontsize = 10)
    plt.xlabel("Теоретические значения", fontsize = 10)
    plt.ylabel("Упорядоченные значения", fontsize = 10)
    plt.show()
```







```
In [94]: #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))
# Устанавливаем фоновые параметры и создаем тепловую карту
sns.heatmap(df.corr(), mask=mask, annot=True, square=True, cmap='YlGn')
plt.xticks(rotation=45, ha='right')
plt.show()
```



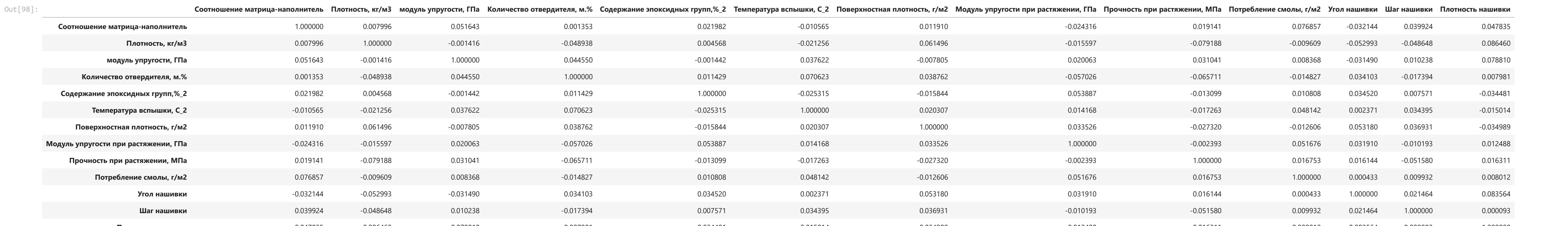
```
In [95]: # Уфф, наконец-то, убеждаюсь, что багровод не осталось. В итоге осталось всего 922 строки
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Integers dtype: 922 entries, 1 to 1822
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   Соотношение матрица-наполнитель    922 non-null   float64
 1   Плотность, кг/м3                  922 non-null   float64
 2   модуль упругости, ГПа              922 non-null   float64
 3   Количество отвердителя, м.%       922 non-null   float64
 4   Содержание эпоксидных групп, %_2  922 non-null   float64
 5   Температура вспышки, С_2          922 non-null   float64
 6   Поверхностная плотность, г/м2     922 non-null   float64
 7   Модуль упругости при растяжении, ГПа 922 non-null   float64
 8   Прочность при растяжении, МПа      922 non-null   float64
 9   Потребление смолы, г/м2            922 non-null   float64
 10  Угол нашивки                      922 non-null   int32  
 11  Шаг нашивки                        922 non-null   float64
 12  Плотность нашивки                 922 non-null   float64
dtypes: float64(12), Int32(1)
memory usage: 129.5 KB
```

```
In [96]: #Сортируем идеальный, без багровода датасет, чтобы в excel проверить дополнительные данные
df.to_csv('Itog1Itog.csv', encoding = 'cp1251')
df.to_excel('Itog1Itog.xlsx')
```

```
In [97]: #Продолжим повторный разбивочный анализ уже без багровод
```

```
In [98]: #Выборкам корреляции между параметрами
df.corr()
```



```
In [99]: #Посмотрим на среднее и медианное значение датасета после багровод
mean_and_50 = df.describe()
mean_and_50.loc['mean', '50%']
#Увидеть, что после удаления багровод среднее и медианное значение осталось в пределах предыдущих значений
```

```
Out[99]: 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
mean           2.927964        1974.118744      736.119982      111.136066     22.200570     286.181128      482.429070      73.303464      2461.491315     218.048059      0.510846      6.931939      57.562887
50%            2.907832        1977.321002      736.178435      111.162090     22.177681     286.220763      457.732246      73.247594      2455.974462     218.697660      1.000000      6.972862      57.584225
```

Приготовим переменные и проверим данные перед нормализацией данных

```
In [100]: df_norm = df.copy()
```

```
In [101]: df_norm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Integers dtype: 922 entries, 1 to 1822
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   Соотношение матрица-наполнитель    922 non-null   float64
 1   Плотность, кг/м3                  922 non-null   float64
 2   модуль упругости, ГПа              922 non-null   float64
 3   Количество отвердителя, м.%       922 non-null   float64
 4   Содержание эпоксидных групп, %_2  922 non-null   float64
 5   Температура вспышки, С_2          922 non-null   float64
 6   Поверхностная плотность, г/м2     922 non-null   float64
 7   Модуль упругости при растяжении, ГПа 922 non-null   float64
 8   Прочность при растяжении, МПа      922 non-null   float64
 9   Потребление смолы, г/м2            922 non-null   float64
 10  Угол нашивки                      922 non-null   int32  
 11  Шаг нашивки                        922 non-null   float64
 12  Плотность нашивки                 922 non-null   float64
dtypes: float64(12), Int32(1)
memory usage: 129.5 KB
```

Предобработка данных.

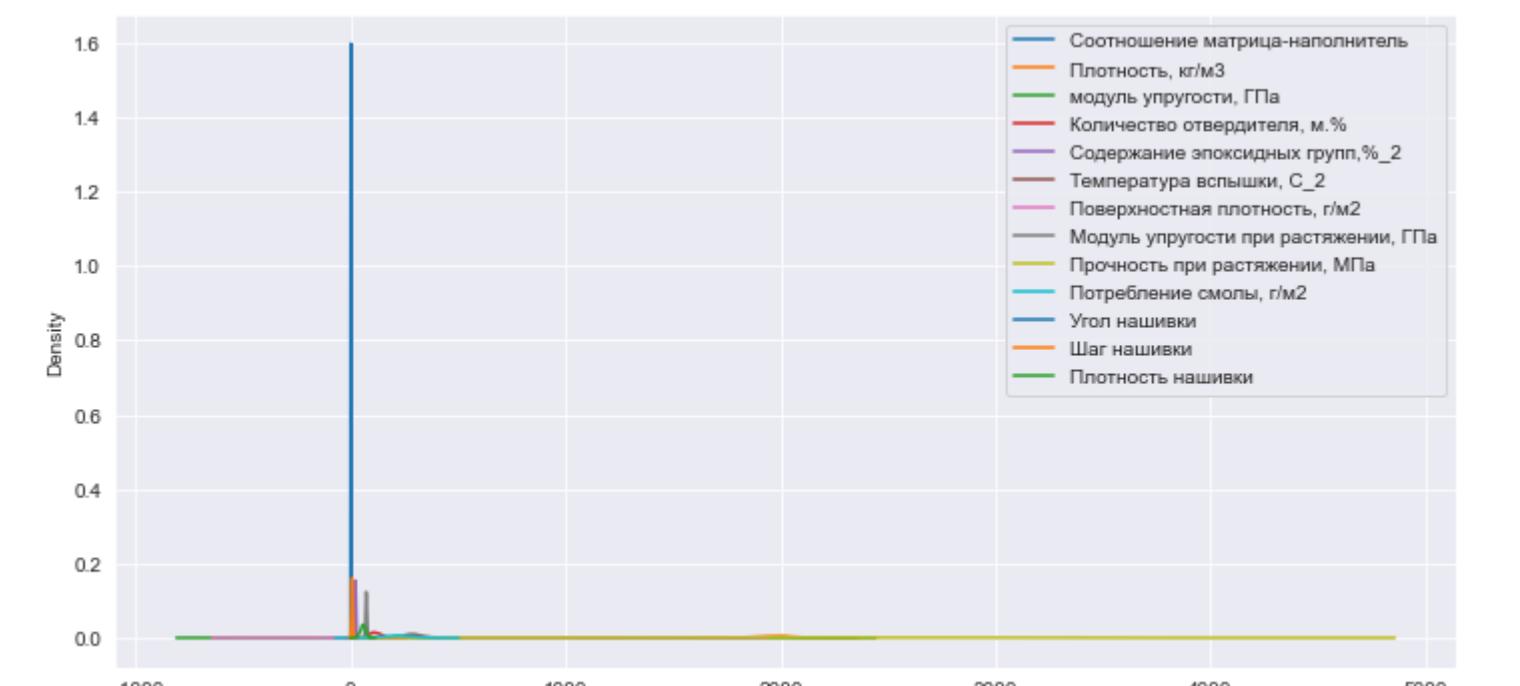
Нормализуем данные

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 [102]: #На данный момент в нашем датасете всего 922 строки. Они все без багровод, 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.
```

```
Out[102]: <AxesSubplot: xlabel='Density'>
```



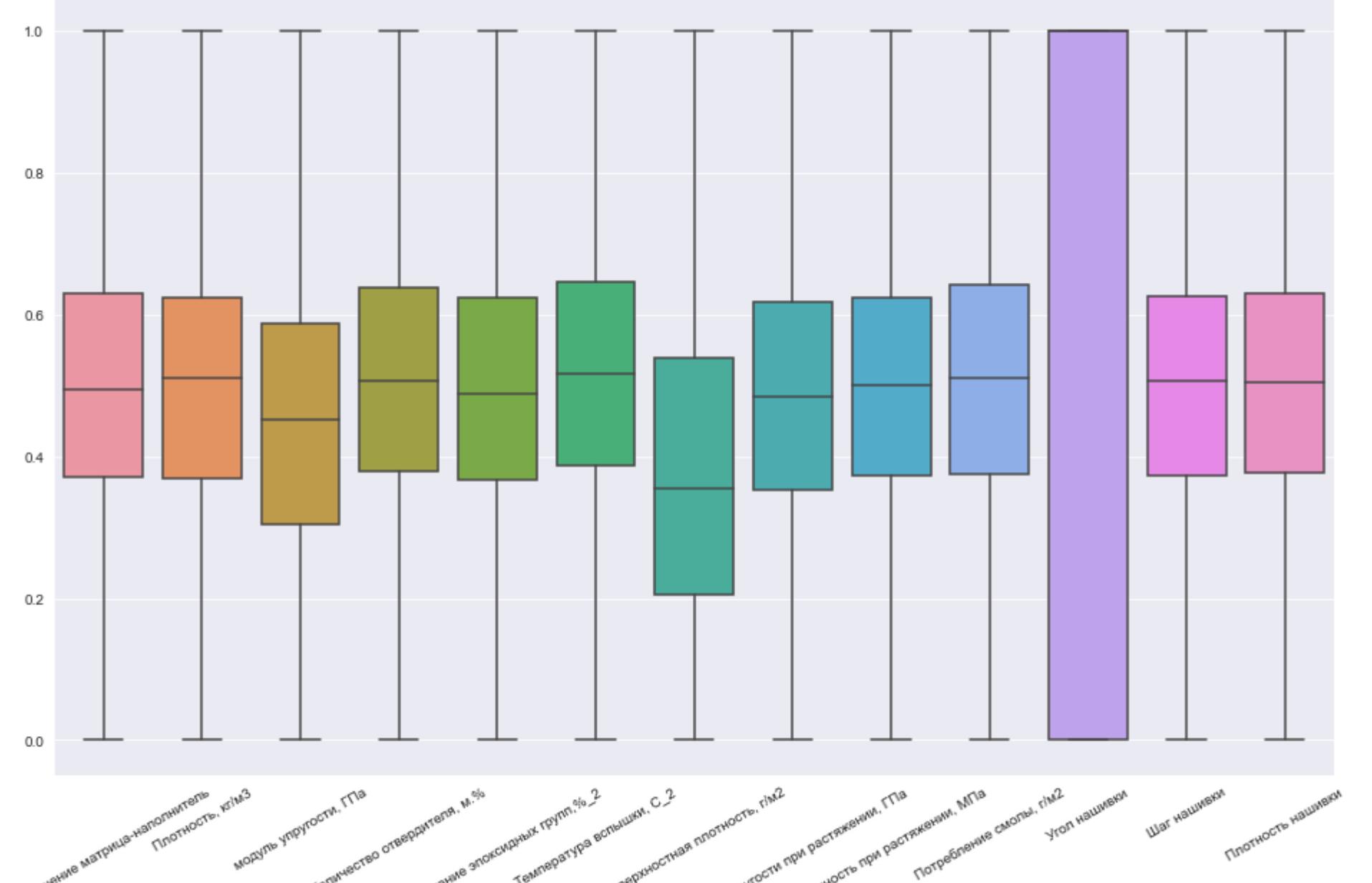
```
In [103]: #Нормализуем данные с помощью MinMaxScaler()
scaler = preprocessing.MinMaxScaler()
col = df.columns
result = scaler.fit_transform(df)

df_minmax_n = pd.DataFrame(result, columns = col)
df_minmax_n.describe()
```

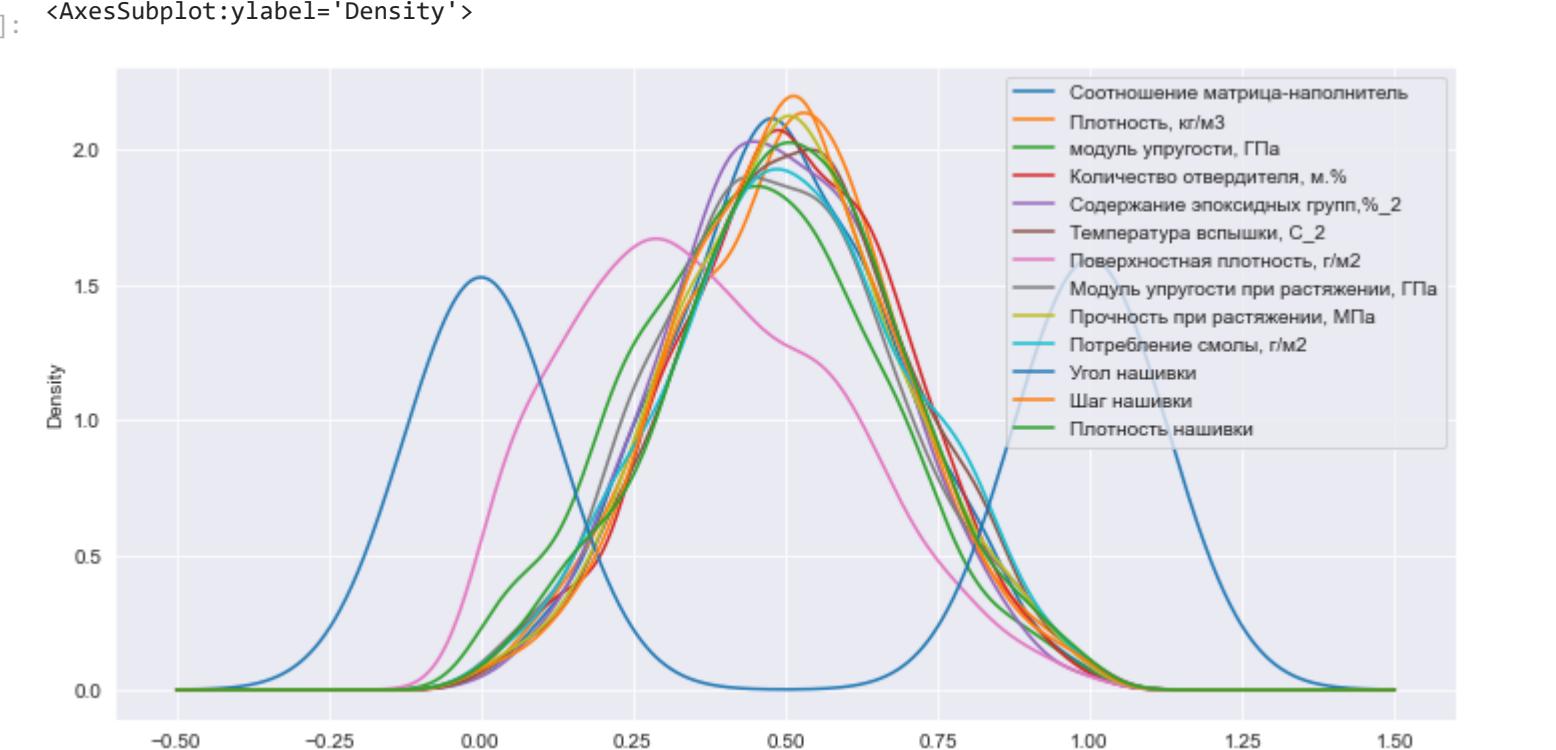
```
Out[103]: Соотношение матрица-наполнитель Плотность, кг/м3 модуль упругости, ГПа Количество отвердителя, м.% Содержание эпоксидных групп, %_2 Температура вспышки, С_2 Поверхностная плотность, г/м2 Модуль упругости при растяжении, ГПа Прочность при растяжении, МПа Погребение смолы, г/м2 Угол нашивки 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
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    0.5030938
std          0.187858    0.168395    0.201534    0.166876    0.180548    0.190721    0.217269    0.196366    0.188668    0.199418    0.500154    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
25%         0.371909    0.368184    0.305188    0.378514    0.366571    0.386228    0.204335    0.353512    0.373447    0.374647    0.000000    0.372844    0.376969
50%         0.495189    0.511396    0.451377    0.506382    0.488852    0.516931    0.354161    0.483718    0.501048    0.510143    1.000000    0.506414    0.504310
75%         0.629774    0.624719    0.567193    0.638735    0.623046    0.646553    0.538897    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
```

```
In [104]: plt.figure(figsize = (16,10))
ax = sns.boxplot(data = df_minmax_n)
ax.set_xticklabels(ax.get_xticklabels(), rotation=30);
```

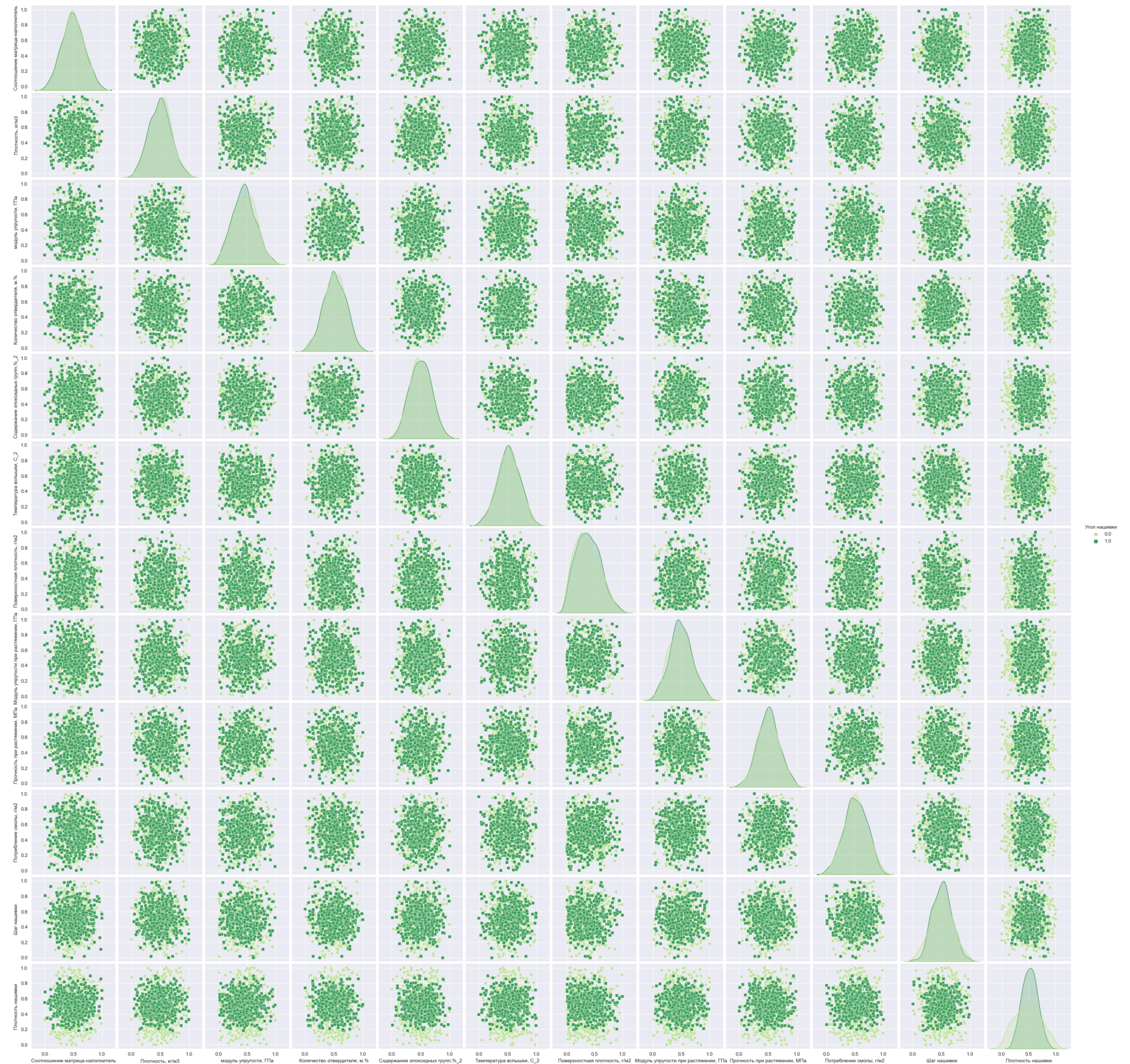


```
In [105]: fig, ax = plt.subplots(figsize = (12, 6))
df_minmax_n.plot(knd = 'kd', ax = ax)
```

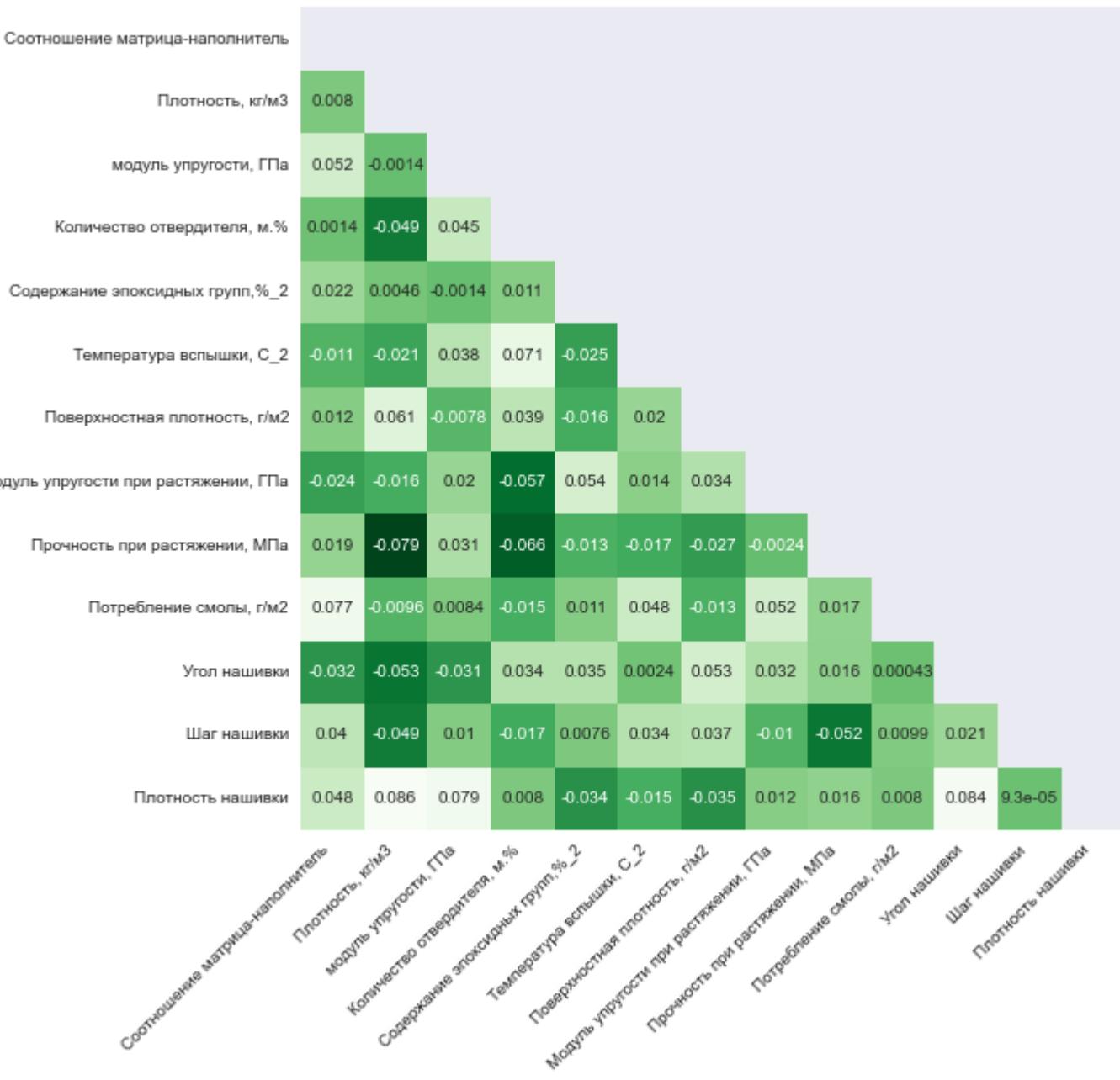


```
In [106]: sns.pairplot(df_minmax_n, hue = 'Угол нашивки', markers = ["o", "*"], diag_kind = 'auto', palette = 'YlGn')
```

```
Out[106]: 
```



```
In [107]:
mask = np.triu(df_simax_n.corr())
f, ax = plt.subplots(figsize=(11, 9))
sns.heatmap(df_simax_n.corr(), mask=mask, annot=True, square=True, cmap='Greens_r')
plt.xticks(rotation=45, ha='right')
plt.show()
```



```
In [108]:
#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[108]:	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	0.000499 0.545436 0.198490 0.013434 0.006381 0.076473 0.056424 0.018808 0.059111 0.000000 0.001075 0.016121
1	0.000499 0.545011 0.198335 0.034634 0.005705 0.080543 0.056380 0.018793 0.050453 0.059065 0.000000 0.001342 0.012618
2	0.000744 0.544829 0.202097 0.03022 0.005976 0.076388 0.056362 0.018787 0.050167 0.059046 0.000000 0.001342 0.015298
3	0.000746 0.539271 0.201687 0.030161 0.006004 0.076742 0.056523 0.018874 0.0508906 0.059320 0.000000 0.001348 0.016178
4	0.000699 0.519919 0.219673 0.030449 0.006062 0.077475 0.057164 0.019055 0.0516627 0.059868 0.000000 0.001361 0.019055
...	...
917	0.000700 0.601751 0.281397 0.026816 0.006203 0.100115 0.064488 0.022531 0.0735908 0.038535 0.000308 0.002798 0.014494
918	0.001078 0.641795 0.139227 0.045701 0.006136 0.079584 0.109777 0.022828 0.07388938 0.036856 0.000313 0.003308 0.016827
919	0.000953 0.573123 0.121122 0.032118 0.006961 0.072186 0.15067 0.021716 0.0773775 0.068752 0.000291 0.001209 0.019652
920	0.001192 0.664667 0.238453 0.045473 0.006190 0.088689 0.206291 0.023812 0.0665248 0.063394 0.000322 0.002030 0.018736
921	0.001071 0.531728 0.117381 0.036336 0.007728 0.084651 0.213418 0.020902 0.0803417 0.054780 0.000281 0.001710 0.021780

922 rows × 13 columns

In [109]: fig, ax = plt.subplots(figsize=(12, 6))
df_norm_n.plot(kind='kde', ax=ax)

Out[109]:

In [110]: #Сравнение с данными до нормализации
df.head(10)

Out[110]:

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
1 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
3 1.857143 2030.0 738.736842 129.00 21.250000 300.000000 210.0 70.0 3000.0 220.0 0 5.0 47.0
4 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
5 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
6 2.59620 1910.0 807.000000 111.86 22.267857 284.615385 210.0 70.0 3000.0 220.0 0 5.0 70.0
7 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
8 3.557018 1930.0 889.000000 129.00 21.250000 300.000000 380.0 75.0 1800.0 120.0 0 7.0 57.0
9 3.552338 2100.0 1421.000000 129.00 21.250000 300.000000 1010.0 78.0 2000.0 300.0 0 7.0 60.0
10 2.919678 2160.0 933.000000 129.00 21.250000 300.000000 1010.0 78.0 2000.0 300.0 0 7.0 70.0
11 2.877358 1990.0 1628.000000 129.00 21.250000 300.000000 1010.0 78.0 2000.0 300.0 0 9.0 47.0

In [111]: #Получаем первые данные из нормализованных б 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[111]:

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 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 1.857143 2030.0 738.736842 129.00 21.250000 300.000000 210.0 70.0 3000.0 220.0 0 5.0 47.0
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 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 2.56620 1910.0 807.000000 111.86 22.267857 284.615385 210.0 70.0 3000.0 220.0 0 5.0 70.0
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 3.557018 1930.0 889.000000 129.00 21.250000 300.000000 380.0 75.0 1800.0 120.0 0 7.0 57.0
7 3.532338 2100.0 1421.000000 129.00 21.250000 300.000000 1010.0 78.0 2000.0 300.0 0 7.0 60.0
8 2.919678 2160.0 933.000000 129.00 21.250000 300.000000 1010.0 78.0 2000.0 300.0 0 7.0 70.0
9 2.877358 1990.0 1628.000000 129.00 21.250000 300.000000 1010.0 78.0 2000.0 300.0 0 9.0 47.0

In [112]: #Рассмотрим несколько вариантов корреляции между параметрами после нормализации (первый вариант)
df_norm_n[df_norm_n.columns].corr()

Out[112]:

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 1.000000 0.281512 1.000000 -0.042233 0.390253 0.658281 0.067392 0.075269 -0.037849 0.314219 0.082452 0.291974 0.445379
1 0.281512 1.000000 -0.042233 1.000000 0.049364 -0.028060 0.003038 -0.026006 -0.026914 -0.036917 -0.009529 -0.039522 0.004268
2 0.390253 1.000000 0.049364 1.000000 0.0409364 0.030595 0.011569 0.067273 0.0376425 -0.039828 0.143166 0.064597 0.121374 0.203167
3 0.658281 1.000000 0.028060 0.049364 1.000000 0.0404877 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886
4 0.067392 1.000000 0.030595 0.0409364 1.000000 0.0404877 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886
5 0.075269 1.000000 0.011569 0.0404877 1.000000 0.03895 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886
6 0.0376425 1.000000 0.067273 0.0404877 1.000000 0.03895 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886
7 0.030595 1.000000 0.030595 0.03895 1.000000 0.03895 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886
8 0.0404877 1.000000 0.0404877 0.03895 1.000000 0.03895 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886
9 0.003895 1.000000 0.003895 0.03895 1.000000 0.03895 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886
10 0.0100000 1.000000 0.0100000 0.03895 1.000000 0.03895 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886
11 0.060206 1.000000 0.060206 0.03895 1.000000 0.03895 0.003895 0.0100000 0.060206 -0.051282 0.244822 0.098492 0.236752 0.260886

In [113]: #Рассмотрим второй вариант корреляции между параметрами после нормализации (второй вариант)
df_minmax_n[df_minmax_n.columns].corr()

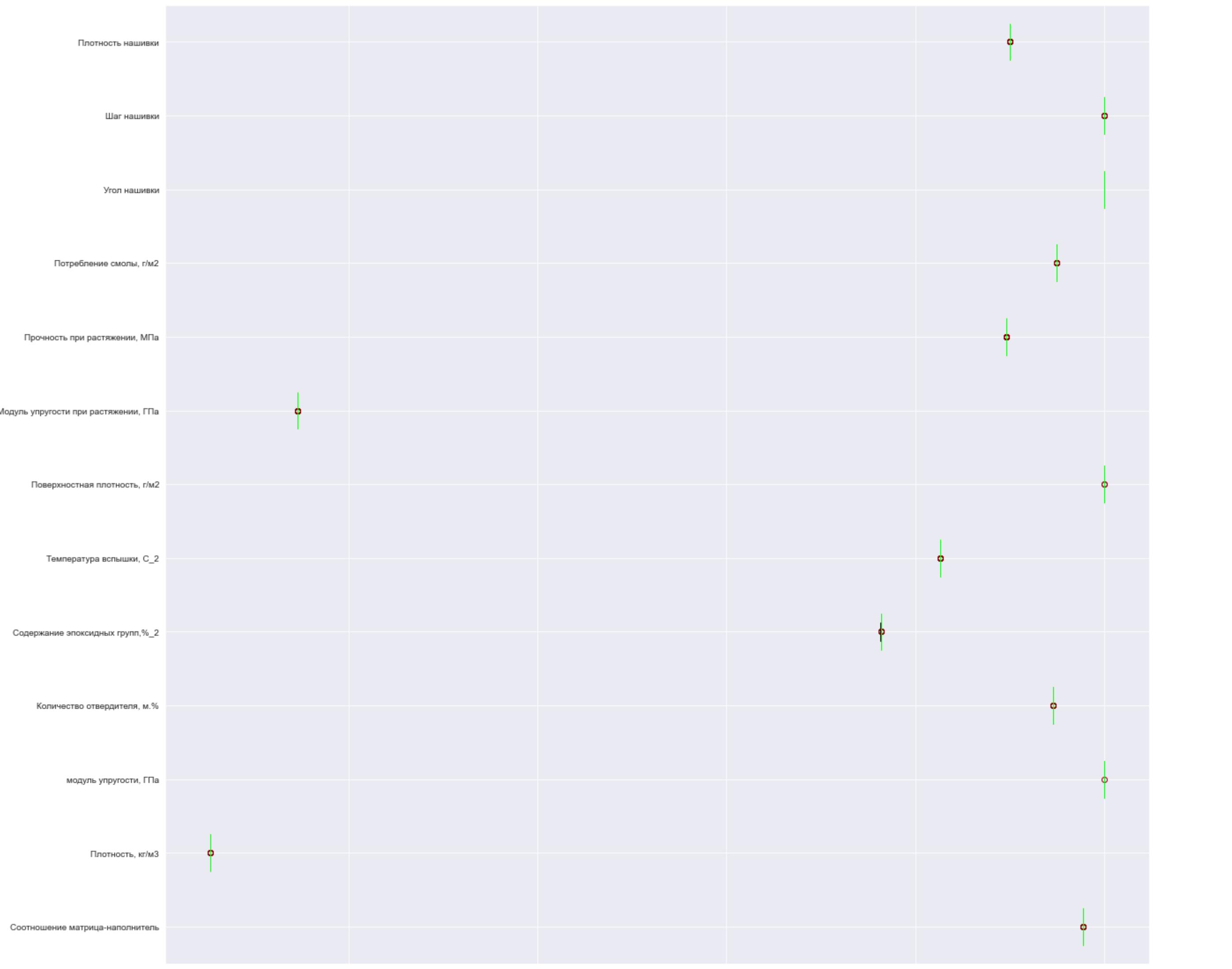
Out[113]:

Соотношение матрица-наполнитель Плотность, кг/м³ модуль упругости, ГПа Количество отвердителя, м.% Содержание эпоксидных групп,%_2 Температура вспышки, С_2

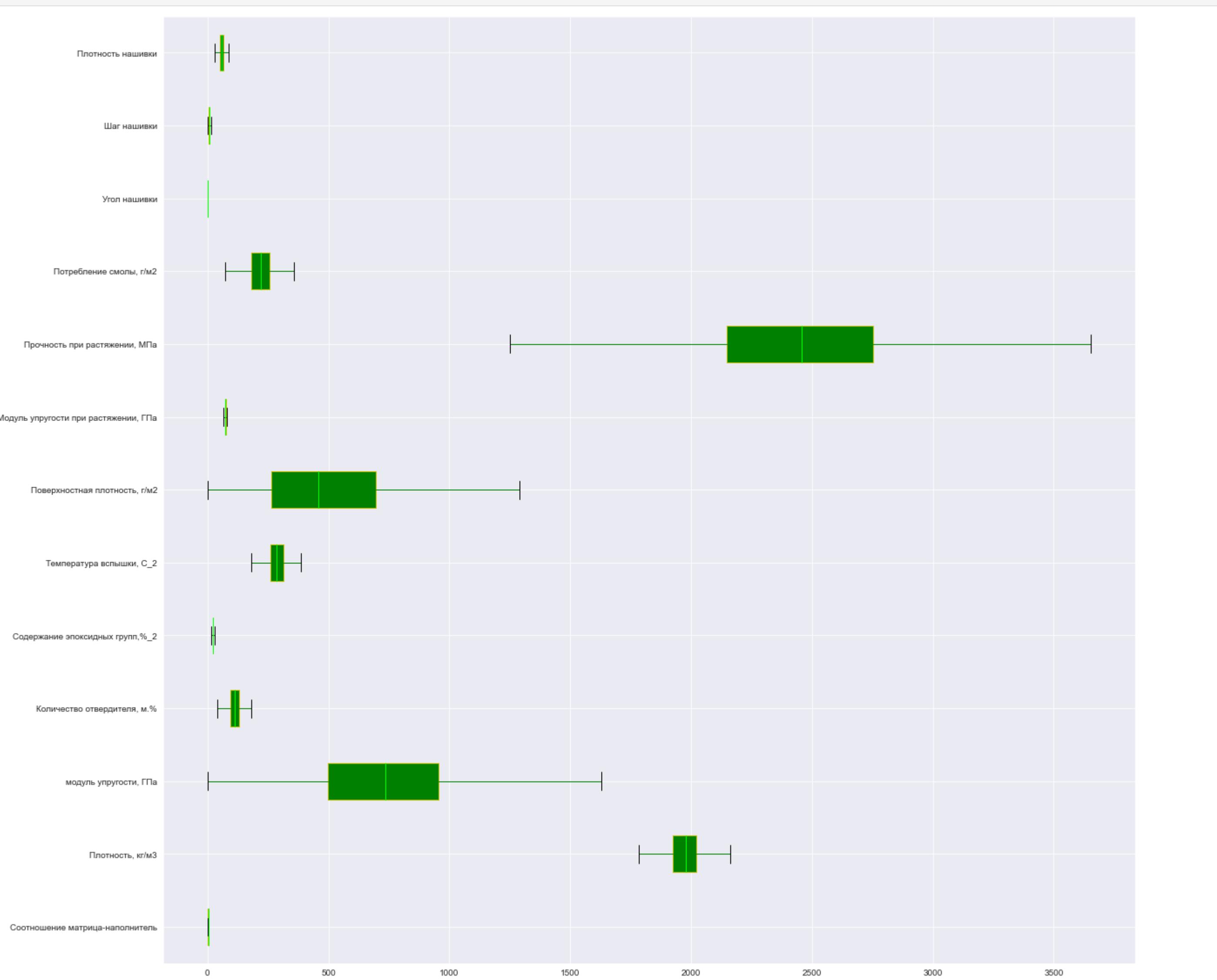
Out[115]:	Соотношение матрица-наполнитель	Плотность, кг/м³	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %_2	Температура вспышки, С_2	Поверхностная плотность, г/м²	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м²	Угол нашивки	Шаг нашивки	Плотность нашивки
0	0.000499	0.545436	0.198490	0.013434	0.006381	0.076473	0.056424	0.018800	0.806064	0.059111	0.000000	0.001075	0.016121
1	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	0.000744	0.544829	0.202097	0.03022	0.005976	0.076388	0.056362	0.018787	0.805167	0.059046	0.000000	0.001342	0.015298
3	0.000746	0.539271	0.201687	0.030161	0.006004	0.076742	0.056623	0.018874	0.808906	0.059320	0.000000	0.001348	0.016178
4	0.000699	0.519919	0.219673	0.030449	0.006062	0.077475	0.057164	0.019055	0.816627	0.059865	0.000000	0.001361	0.019055
...
917	0.000700	0.601751	0.281397	0.026816	0.006203	0.100115	0.064488	0.022531	0.735908	0.038535	0.000308	0.002798	0.014494
918	0.001078	0.641795	0.139227	0.045701	0.006136	0.079584	0.109777	0.022828	0.738898	0.036856	0.000313	0.003308	0.016827
919	0.000953	0.573123	0.121122	0.032118	0.006961	0.072186	0.215067	0.021716	0.773775	0.068752	0.000291	0.001209	0.019652
920	0.001192	0.664667	0.238453	0.045473	0.006190	0.088689	0.206291	0.023812	0.666248	0.063394	0.000322	0.002030	0.018736
921	0.001071	0.531728	0.117381	0.036336	0.007728	0.084651	0.213418	0.020902	0.803417	0.054780	0.000281	0.001710	0.021780

922 rows × 13 columns

```
In [116]: #Построение ящиков с усами
scaler = MinMaxScaler()
scaler.fit(df)
plt.figure(figsize=(20, 20))
#Рисование ящиков
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 [117]: #Помимо ящиков посмотрим на "ящики с усами"
scaler = MinMaxScaler()
scaler.fit(df_minmax_n)
plt.figure(figsize = (20, 20))
#Рисование ящиков
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()
```




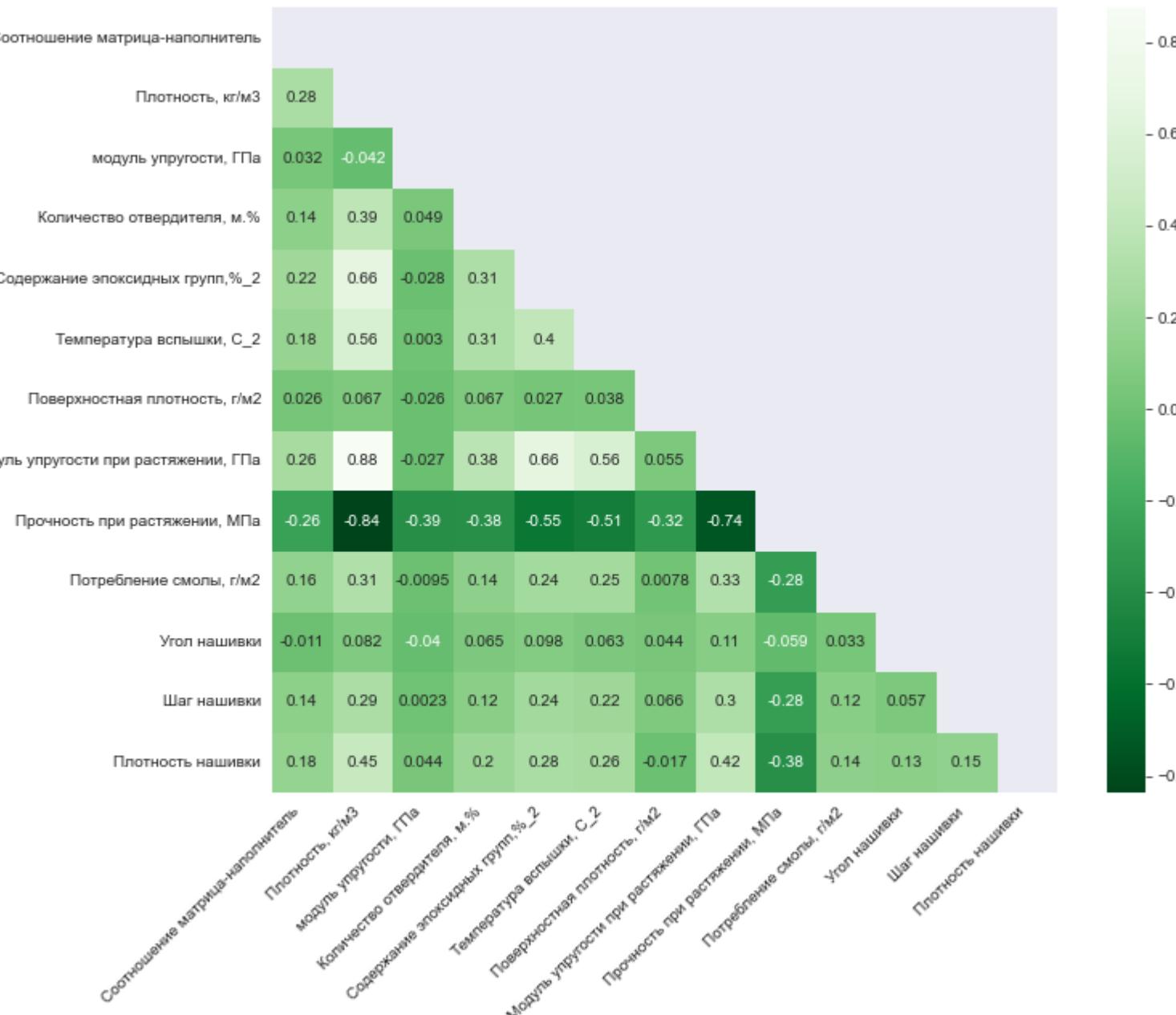
```
In [118]: # Визуализация графиком показывает, что нормализация при помощи "Normalizer" дает нам большое количество выбросов
```

Стандартизируем данные

```
In [119]: X1 = df_minmax_n.copy()
X2 = df_norm_n.copy()
```

```
In [120]: df_std_X1 = preprocessing.StandardScaler().fit(X1)
```





In [128]: `df_norm_n.describe()`

```
Out[128]:
   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
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     0.920085  0.97049  0.220075  0.033657  0.006717  0.086572  0.144885  0.022174  0.731093  0.065909  0.000155  0.002101  0.017401
std      0.000286  0.064504  0.095713  0.008973  0.001016  0.015045  0.083872  0.002485  0.069235  0.018509  0.000153  0.000808  0.003795
min     0.000163  0.444650  0.000709  0.011339  0.004113  0.049402  0.000230  0.016108  0.463136  0.021630  0.000000  0.000011  0.007195
25%     0.000685  0.552643  0.150901  0.027607  0.005990  0.076193  0.080748  0.020407  0.688177  0.025457  0.000000  0.001543  0.014860
50%     0.000872  0.592130  0.219805  0.033559  0.006654  0.085003  0.140105  0.021973  0.740666  0.065577  0.000244  0.002053  0.017293
75%     0.001068  0.640228  0.289231  0.039565  0.007346  0.096585  0.204309  0.023759  0.780691  0.077899  0.000301  0.002630  0.019756
max     0.001803  0.824241  0.525102  0.062919  0.010887  0.147961  0.414271  0.030620  0.877580  0.122973  0.000419  0.004519  0.03027
```

In [129]: `df_maxmax_n.describe()`

```
Out[129]:
   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
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      0.187858  0.188395  0.201534  0.186876  0.180548  0.190721  0.217269  0.196366  0.188668  0.199418  0.500154  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
25%     0.371909  0.368184  0.305188  0.378514  0.366571  0.386228  0.204335  0.373447  0.374647  0.000000  0.372844  0.376869
50%     0.495189  0.511396  0.451377  0.506382  0.488852  0.516931  0.354161  0.483718  0.501043  1.000000  0.506414  0.504310
75%     0.629774  0.624719  0.567193  0.638735  0.623046  0.646553  0.538397  0.617568  0.624511  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
```

Разработка и обучение нескольких моделей для прогноза прочности при растяжении

In [130]: `#Данные в нашем игровом датасете в основном непрерывные и на них приходит сразу решение данной задачи с использованием регрессионных моделей. Но попарные графики рассеивания точек и тепловая карта 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 выборки и посмотрим, что получилось

In [131]: `# Проверяя прочность при растяжении`

```
In [132]:
#разделение на тестовую, валидационную выборку, добавляя 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, МПа'],
    df['Прочность при растяжении, МПа'],
    test_size = 0.3,
    random_state = 42)
```

In [133]: `#Поборка проблемности разбиения`

```
In [133]:
df_norm_n.shape[0] - x_train_1.shape[0] + x_test_1.shape[0]
```

Out[133]: 0

In [134]: `x_train_1.head()`

```
Out[134]:
   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 Потребление смолы, г/м² Угол нашивки Шаг нашивки Плотность нашивки
481     0.000528  0.589066  0.132149  0.020997  0.006254  0.071450  0.186083  0.020219  0.073037  0.000291  0.002285  0.015563
650     0.000430  0.617300  0.232509  0.027252  0.007346  0.075579  0.081347  0.024837  0.056839  0.000308  0.001350  0.018310
483     0.000580  0.585434  0.170519  0.043402  0.007074  0.096550  0.132807  0.021197  0.041487  0.000295  0.002249  0.018929
355     0.000911  0.545896  0.306130  0.024469  0.006610  0.070488  0.159918  0.018527  0.050148  0.000000  0.001774  0.0012540
850     0.000573  0.595720  0.263992  0.034166  0.007519  0.089582  0.095597  0.023521  0.086336  0.000322  0.002765  0.017180
```

In [135]: `y_train_1`

```
Out[135]:
   Прочность при растяжении, МПа
552     2641.571967
735     2404.068921
554     2619.854215
403     2793.783901
948     2298.985700
...
120     1994.674603
306     2419.732206
959     2758.141767
497     2347.135204
113     1529.694423
```

645 rows × 1 columns

In [136]: `y_test_1`

```
Out[136]:
   Прочность при растяжении, МПа
360     2167.533030
427     2705.819718
611     2952.639631
336     2305.241225
604     1399.118555
...
479     3305.886922
149     2085.866383
561     2461.609016
632     2616.114331
411     2478.484767
```

277 rows × 1 columns

In [137]: `y_train_1.shape`

Out[137]: (645, 1)

```
In [138]:
#функция для сглаживания результатов предсказаний с моделью, выдающей среднее значение по тестовой выборке
def mean_model(y_test_1):
    return [np.mean(y_test_1)] * len(y_test_1)

y_1_pred_mean = mean_model(y_test_1)
```

In [139]: `#Проверка различных моделей при стандартных параметрах
#Метод опорных векторов`

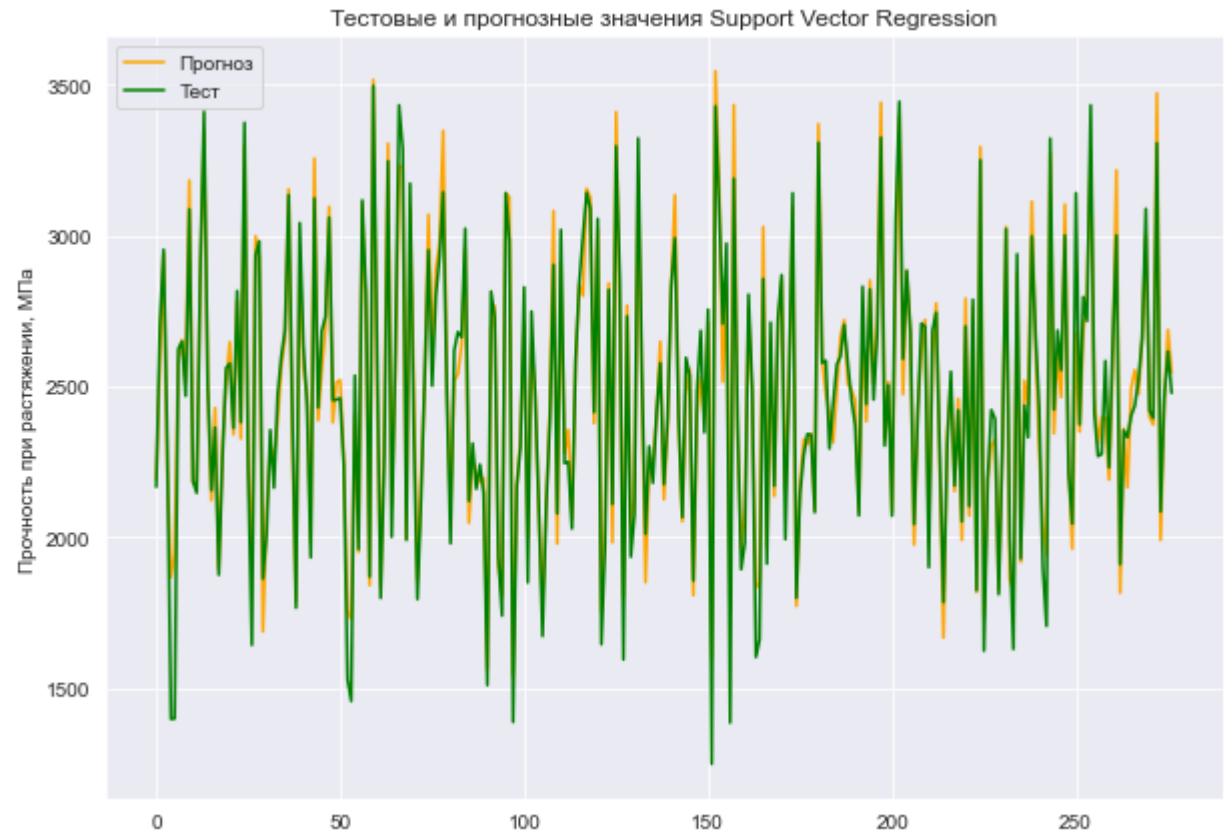
```
In [140]: svr = make_pipeline(StandardScaler(), SVR(kernel = 'rbf', C = 500.0, epsilon = 1.0))
```

```
#Подгружаем модель
svr = fit(x_train_1, np.ravel(y_train_1))
#Вычисляем коэффициенты демпфирования
y_pred_svr = svr.predict(x_test_1)
mae_svr = mean_absolute_error(y_pred_svr, y_test_1)
mse_svr = mean_squared_error(y_test_1, y_pred_svr)
print("Support Vector Regression Results Train:")
print("Test score: {:.2f}".format(svr.score(x_train_1, y_train_1))) # Скор для тренировочной выборки
print("Support Vector Regression Results:")
print("SVR MAE: 0.95")
print("SVR MPE: 0.04")
print("SVR MSE: 11671.63")
print("SVR RMSE: 108.04")
print("Test score: 0.95")
```

```
In [141]: #Проверка ошибок, введенных в модели
mse_lin_elast_mean = 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_mean)
print("RMSE for mean target: ", np.sqrt(mse_lin_elast_mean))

MAE for mean target: 368.96801863398946
MSE for mean target: 210404.3755454378
RMSE for mean target: 462.05146227526157
```

```
In [142]: 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 = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Промежуточное значение, МНв")
plt.legend()
plt.grid()
```

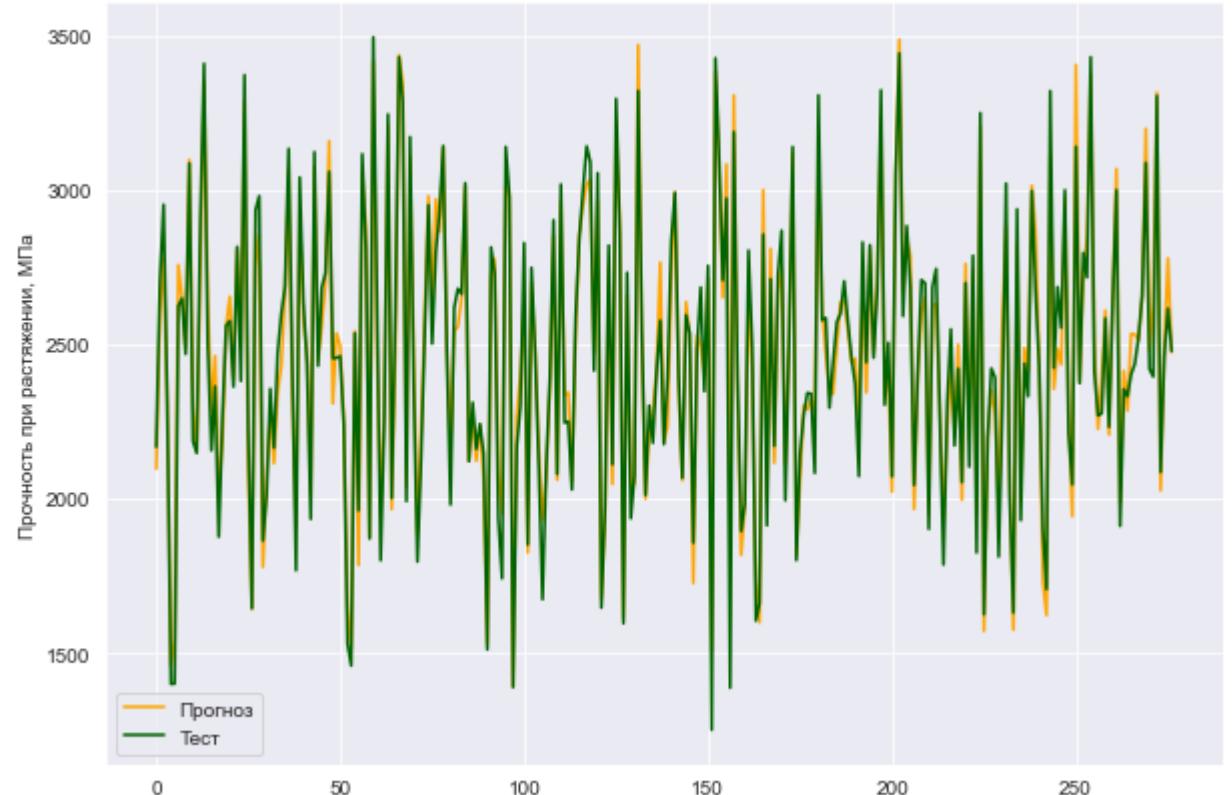


```
In [143]: #Метод случайного леса - Random Forest Regressor - 2
```

```
In [144]: #испорченные модели и фундаментализация метода случайный лес
rfr = RandomForestRegressor(n_estimators=15, max_depth=7, random_state=33)
rfr.fit(x_train_1, y_train_1)
y_pred_forest = rfr.predict(x_test_1)
mae_rfr_elast = mean_absolute_error(y_test_1, y_pred_forest)
mse_rfr_elast = mean_squared_error(y_test_1, y_pred_forest)
print("Random Forest Regression Results Train:")
print("Test score: {:.2f}".format(rfr.score(x_train_1, y_train_1))) # Скор для тренировочной выборки
print("RF MAE: ", round(mean_absolute_error(y_test_1, y_pred_forest)))
print("RF MPE: {:.2f}".format(mean_absolute_percentage_error(y_test_1, y_pred_forest)))
print("RF MSE: {:.2f}".format(mse_rfr_elast))
print("RF RMSE: {:.2f}".format(np.sqrt(mse_rfr_elast)))
print("Test score: {:.2f}".format(rfr.score(x_test_1, y_test_1))) # Скор для тестовой выборки
```

```
Rand Forest Regression Results Train:
Test score: 0.98
Rand Forest Regression Results:
RF MAE: 77
RF MPE: 0.03
RF MSE: 9595.97
RF RMSE: 97.50
Test score: 0.96
```

```
In [145]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Random Forest Regression")
plt.plot(y_pred_forest, label = "Прогноз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Промежуточность при расщеплении, МНв")
plt.legend()
plt.grid()
```



```
In [146]: #Метод линейной регрессии - Linear Regression - 3
```

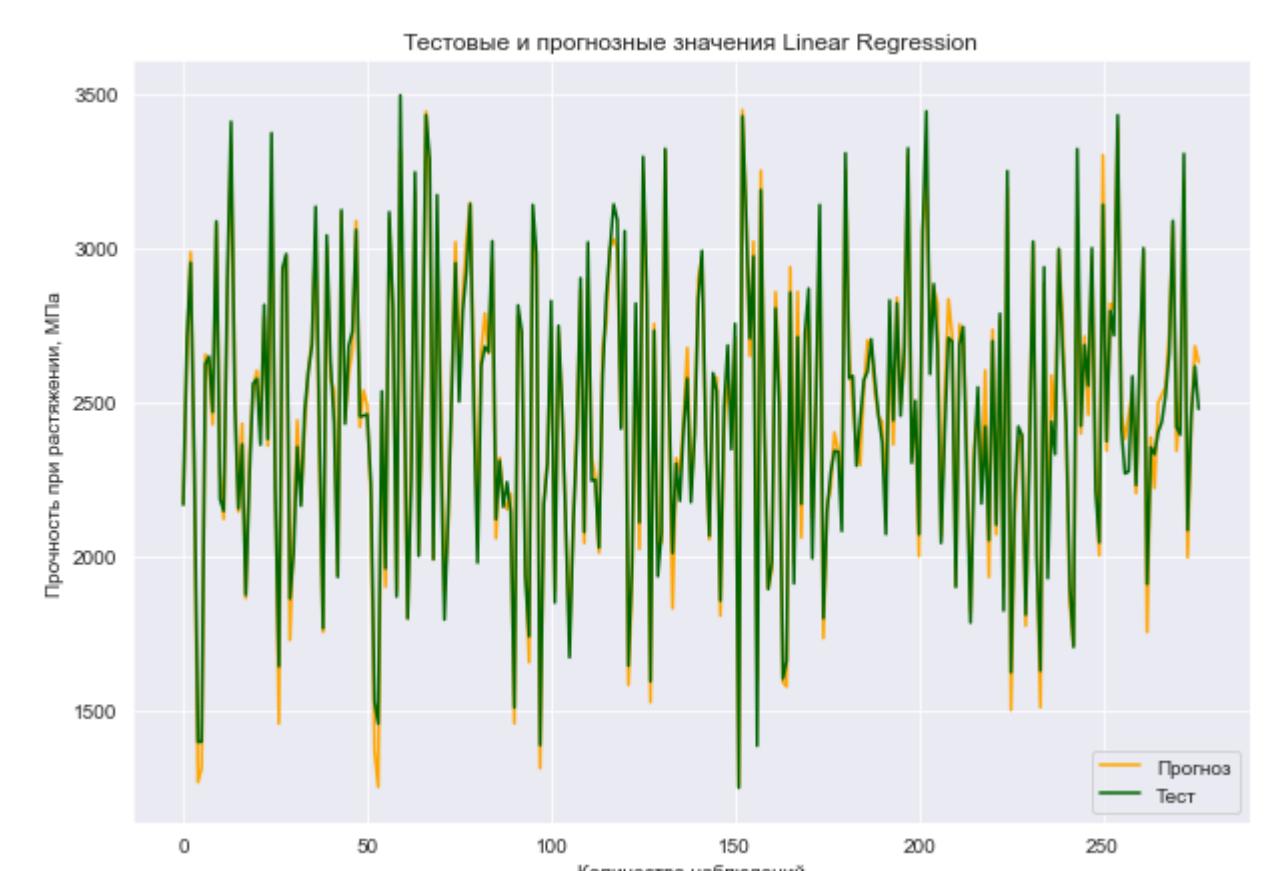
```
In [147]: #испорченные модели и фундаментализация линейной регрессии
lr = LinearRegression()
lr.fit(x_train_1, y_train_1)
y_pred_lr = lr.predict(x_test_1)
mae_lr_elast = mean_absolute_error(y_pred_lr, y_test_1)
mse_lr_elast = mean_squared_error(y_test_1, y_pred_lr)
print("Linear Regression Results 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: ", mean_absolute_error(y_test_1, y_pred_lr))
print("LR MPE: {:.2f}".format(mean_absolute_percentage_error(y_test_1, y_pred_lr)))
print("LR MSE: {:.2f}".format(mse_lr_elast))
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: 0.03
LR MPE: 0.0031
LR MSE: 6249.41
LR RMSE: 79.07
Test score: 0.97
```

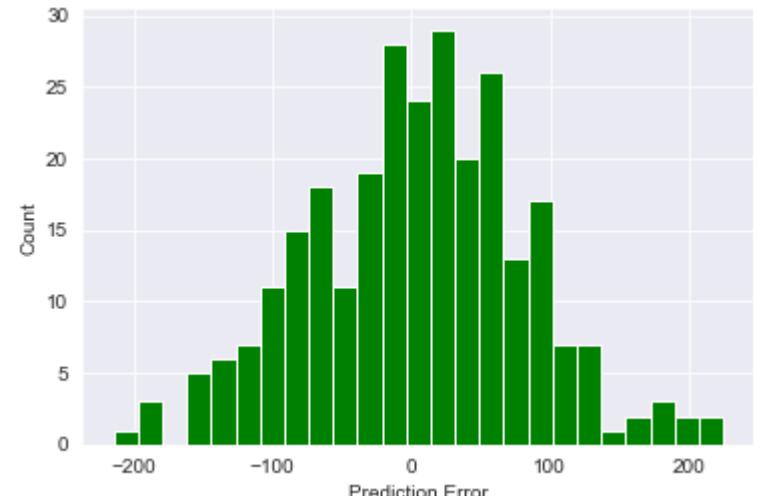
```
In [148]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Linear Regression")
plt.plot(y_pred_lr, label = "Прогноз", color = "orange")
plt.plot(y_test_1.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Промежуточность при расщеплении, МНв")
plt.legend()
plt.grid()
```

Линейная регрессия с задачей справилась в 97 % случаев.



In [149]: #Визуализация гистограммы распределения ошибки

```
In [150]:
error = y_test_1 - y_pred_lr
plt.hist(error, bins = 25, color = "g")
plt.xlabel("Prediction Error")
_ = plt.ylabel("Count")
```

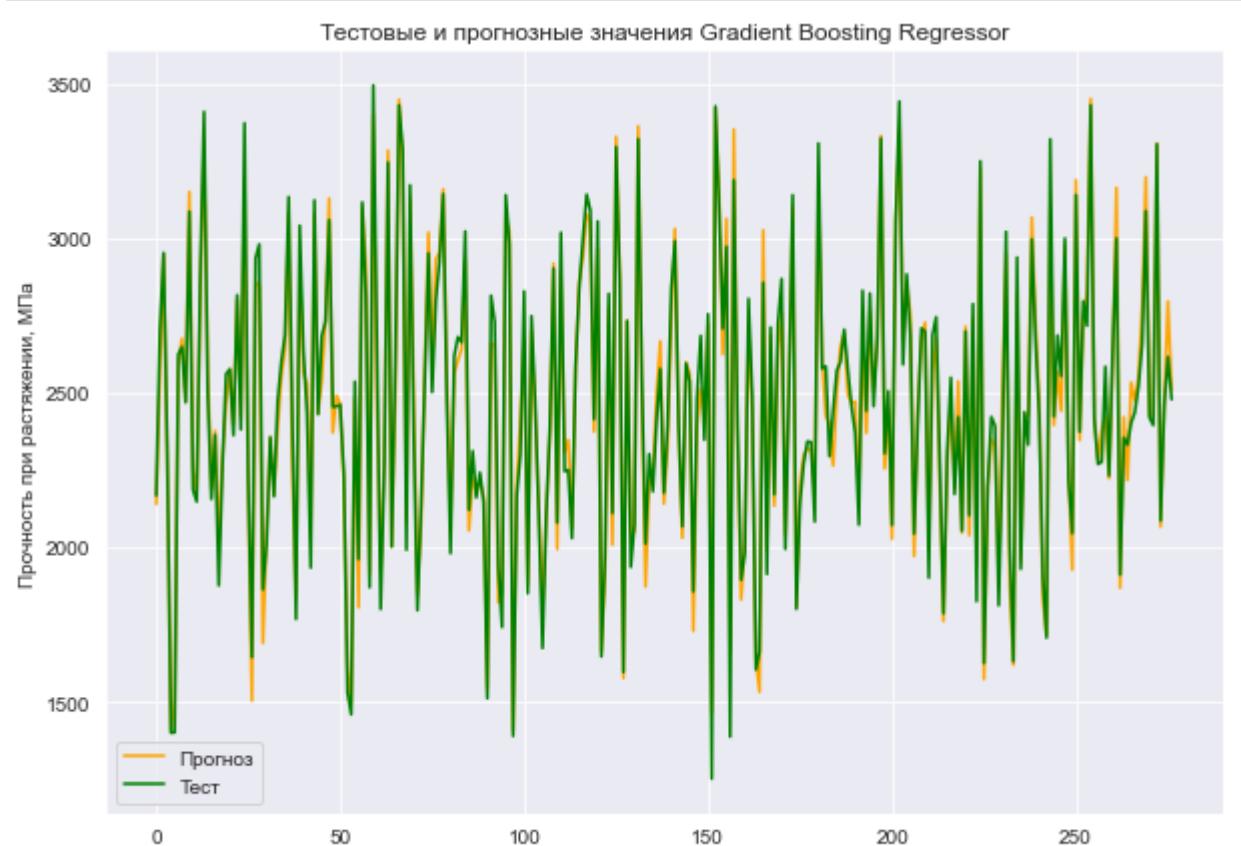


In [151]: #Метода градиентного бустинга - Gradient Boosting Regressor - 4

```
In [152]:
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:")
print("Test score: (%.2f)"%gbr.score(x_train_1, y_train_1)) # Скор для тренировочной выборки
print("Gradient Boosting Regressor Results:")
print("GBR_MAE: ", round(mae_gbr, 2))
print("GBR_MAPE: (%.2f)"%format(mean_absolute_percentage_error(y_test_1, y_pred_gbr)))
print("GBR_MSE: ", round(mse_gbr, 2))
print("GBR_RMSE: (%.2f)"%format(np.sqrt(mse_gbr)))
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: 65
GBR_MAPE: 0.03
GBR_MSE: 6552.07
GBR_RMSE: 80.94
Test score: 0.97

```
In [153]:
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Gradient Boosting Regressor")
plt.plot(y_pred_gbr, label = "Прогноз", color = "orange")
plt.plot(y_test_1, label = "Тест", color = "green")
plt.xlabel("Количество наблюдений")
plt.ylabel("Проницаемость при растяжении, МПа")
plt.legend()
plt.grid(True);
#Градиентный бустинг с заданной спрэдом в 87 % случаев.
```

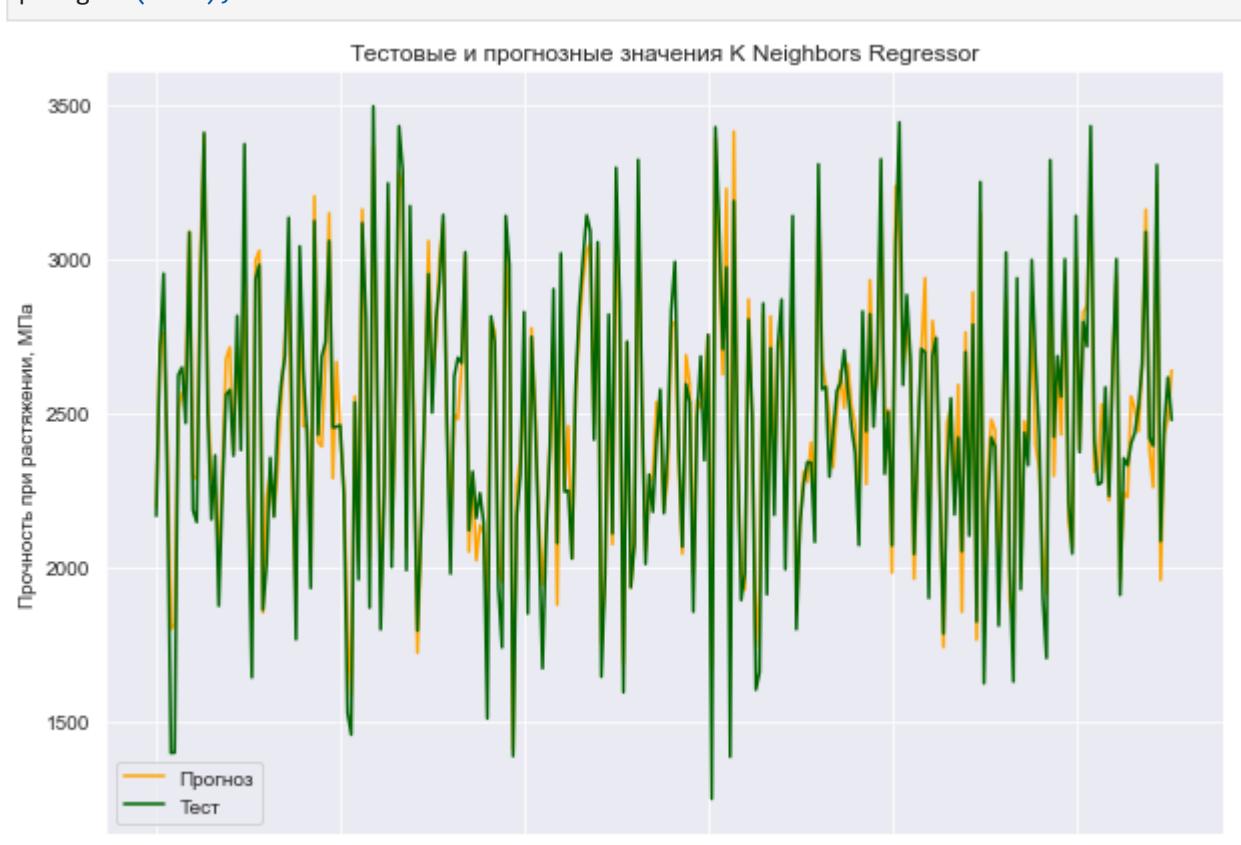


In [154]: # Метод K ближайших соседей - K Neighbors Regressor - 5

```
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 Regression - Results Train:")
print("K Neighbors Regression - Results:")
print("KNN_MAE: (%.2f)"%round(mae_knn, 2))
print("KNN_MAPE: (%.2f)"%format(mean_absolute_percentage_error(y_test_1, y_pred_knn)))
print("KNN_MSE: (%.2f)"%format(mse_knn))
print("KNN_RMSE: (%.2f)"%format(np.sqrt(mse_knn)))
print("Test score: (%.2f)".format(knn.score(x_test_1, y_test_1)))# Скор для тестовой выборки
```

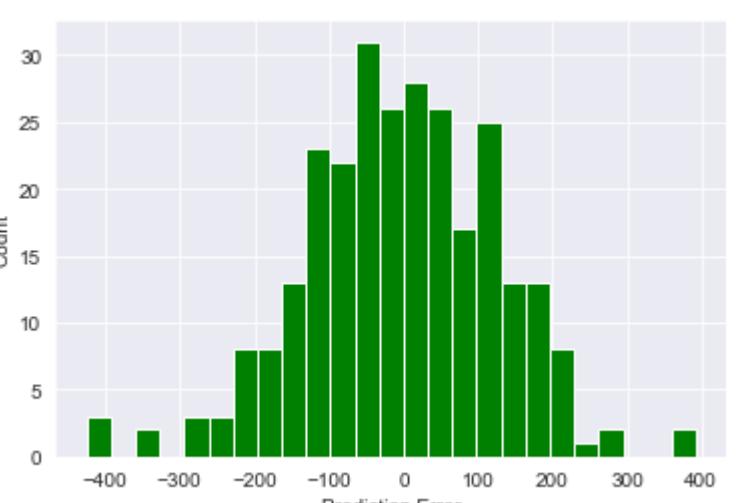
K Neighbors Regression - Results Train:
Test score: 0.94
K Neighbors Regression - Results:
KNN_MAE: 102
KNN_MAPE: 0.04
KNN_MSE: 16723.93
KNN_RMSE: 129.32
Test score: 0.92

```
In [155]:
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения K Neighbors Regressor")
plt.plot(y_pred_knn, label = "Прогноз", color = "orange")
plt.plot(y_test_1, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Проницаемость при растяжении, МПа")
plt.legend()
plt.grid(True);
```



In [156]: #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 [157]: #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_test_1, y_pred_dtr)
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(np.sqrt(mse_dtr)))
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("DTR MAPE: {:.2f}%".format(dtr.score(x_test_1, y_test_1)))# Скор для тестовой выборки
print("Test score: {:.2f}").format(dtr.score(x_test_1, y_test_1))
```

Decision Tree Regressor Results Train:

Test score: 0.94

Decision Tree Regressor Results:

DTR MAE: 100

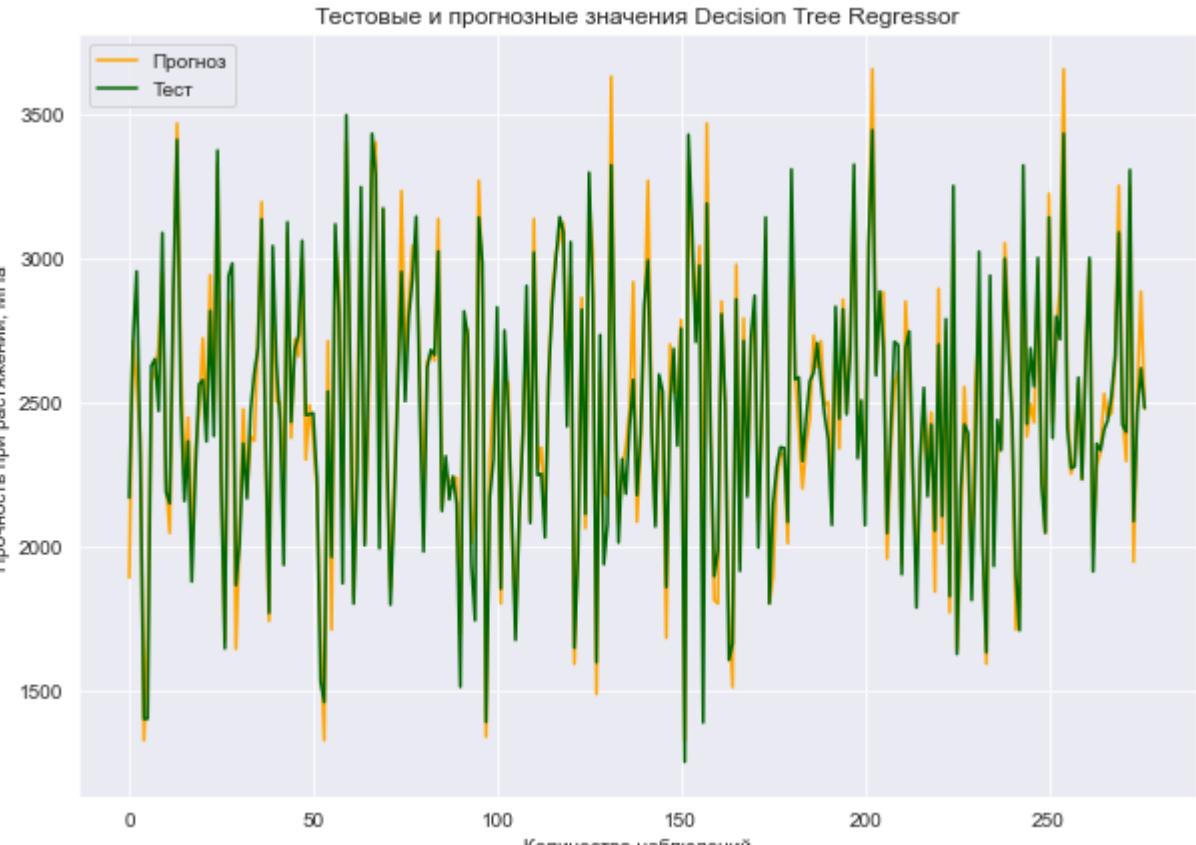
DTR MSE: 16212.91

DTR RMSE: 127.33

DTR MAPE: 0.44

Test score: 0.92

In [158]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Decision Tree Regression")
plt.plot(y_pred_dtr, 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 [159]: # 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(np.sqrt(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("SGD MAPE: {:.2f}%".format(sgd.score(x_test_1, y_test_1)))# Скор для тестовой выборки
print("Test score: {:.2f}").format(sgd.score(x_test_1, y_test_1))
```

Stochastic Gradient Descent Regressor Results Train:

Test score: 0.74

Stochastic Gradient Descent Regressor Results:

SGD MAE: 181

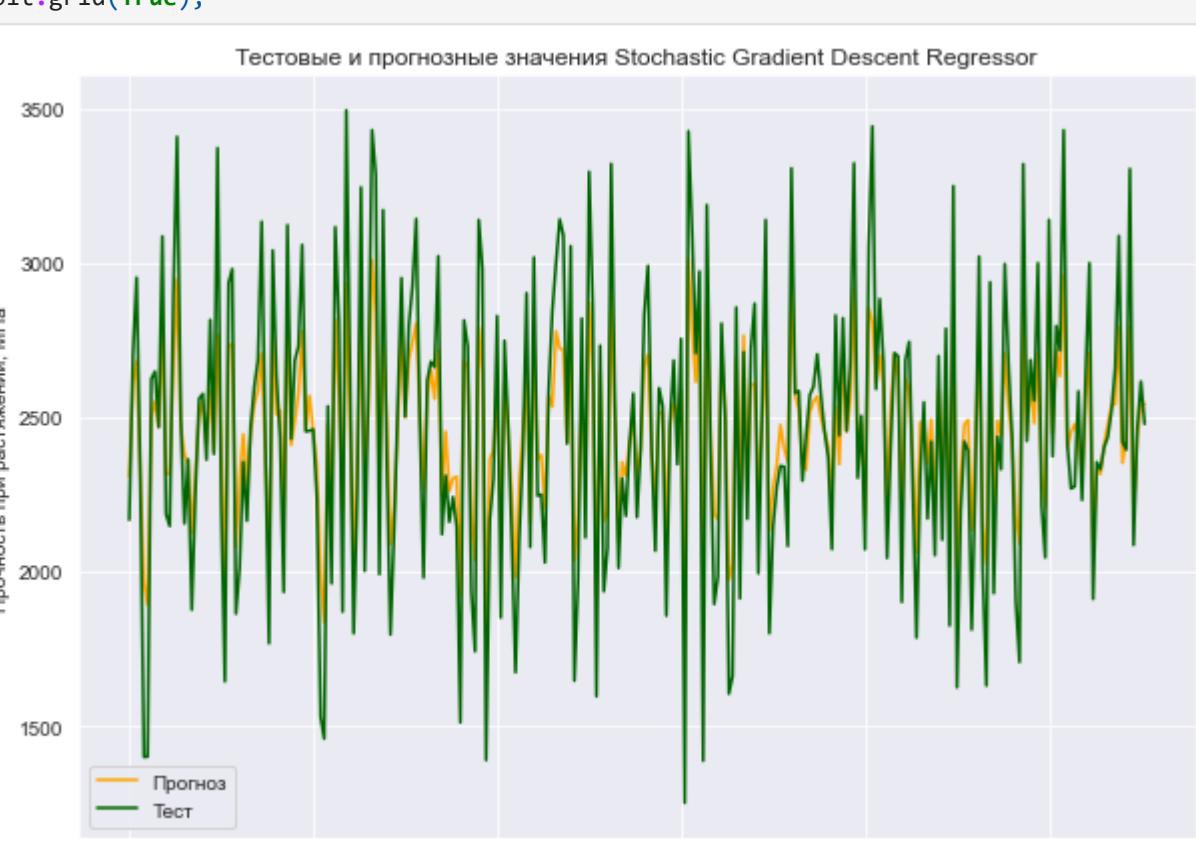
SGD MSE: 52213.29

SGD RMSE: 228.59

SGD MAPE: 0.68

Test score: 0.76

In [160]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Stochastic Gradient Descent Regressor")
plt.plot(y_pred_sgd, 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 [161]: # Многослойный перцептрон - Multi-Layer Perceptron regression - 8

```
mlp = MLPRegressor(random_state = 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_test_1, y_pred_mlp)
print("Multi-layer Perceptron Results Train:")
print("Test score: {:.2f}%".format(mlp.score(x_train_1, y_train_1)))# Скор для тренировочной выборки
print("Multi-layer Perceptron Results:")
print("SGD MAE: {:.2f}%".format(mae_mlp))
print("SGD MSE: {:.2f}%".format(np.sqrt(mse_mlp_elast)))
print("SGD RMSE: {:.2f}%".format(np.sqrt(mse_mlp_elast)))
print("SGD MAPE: {:.2f}%".format(mean_absolute_percentage_error(y_test_1, y_pred_mlp)))
print("SGD MAPE: {:.2f}%".format(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))
```

Multi-layer Perceptron Results Train:

Test score: -16.28

Multi-layer Perceptron Results:

SGD MAE: 1800

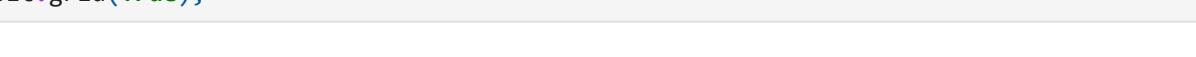
SGD MAPE: 0.72

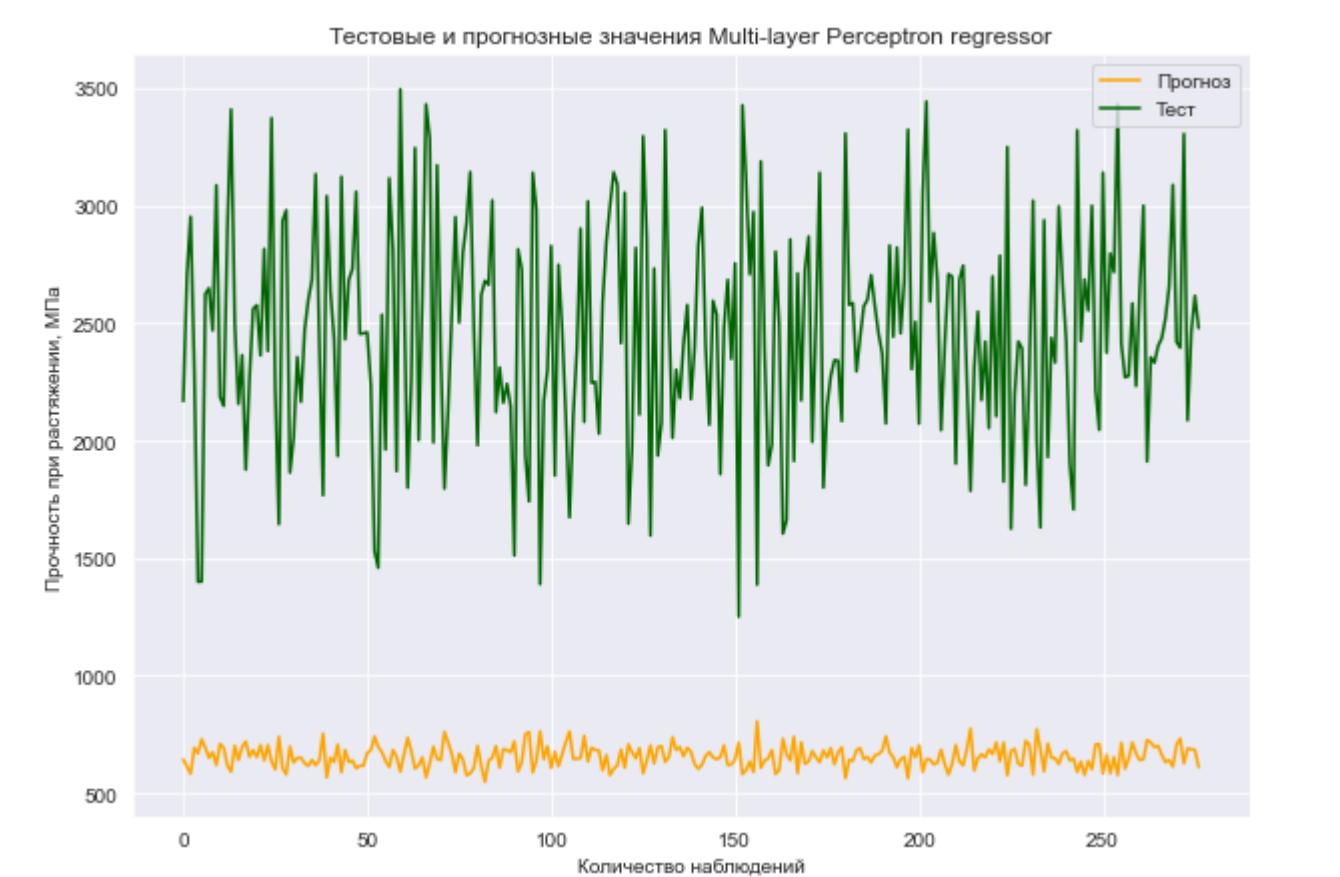
SGD MSE: 3519593.40

SGD RMSE: 1876.03

Test score: -15.44

In [162]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Multi-layer Perceptron regression")
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 [163]: # Лассо регрессия - the Lasso - 9
cif = 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("Test score: {:.2f}".format(clf.score(x_train_1, y_train_1)))# Скор для тренировочной выборки
print("Lasso regressor Results:")
print("SGD_MAE: ", round(mean_absolute_error(y_test_1, y_pred_clf)))
print("SGD_MAPE: ", round((mean_absolute_error(y_test_1, y_pred_clf)*100)/mean(y_test_1)))
print("SGD_RMSE: {:.2f}".format(np.sqrt(mse_clf_elast)))
print("SGD_RMSE: {:.2f}".format (np.sqrt(mse_clf_elast)))
print("Test score: {:.2f}".format(clf.score(x_test_1, y_test_1)))# Скор для тестовой выборки
print("Test score: 0.96")

In [164]:
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, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);


```

```
In [165]: #сравнив наши модели 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_gbc, mae_knr, mae_dtr, mae_sdg, mae_mlp, mae_cif]}
mae_df = pd.DataFrame(mae_df)
```

```
In [166]: mae_df
```

```
Out[166]:


|   | Перцептор         | MAE         |
|---|-------------------|-------------|
| 0 | Support Vector    | 78.477914   |
| 1 | RandomForest      | 76.589025   |
| 2 | Linear Regression | 61.986894   |
| 3 | GradientBoosting  | 64.902175   |
| 4 | KNeighbors        | 102.030259  |
| 5 | DecisionTree      | 100.359294  |
| 6 | SGD               | 181.506953  |
| 7 | MLP               | 1808.547264 |
| 8 | Lasso             | 69.474334   |


```

```
In [167]: # Поиск оптимальных гиперпараметров.
# В машинном обучении гиперпараметры называют параметры алгоритмов, значения которых устанавливаются перед запуском процесса обучения.
# В этом смысле они отличаются от обычных параметров, численных в процессе обучения. Гиперпараметры используются для управления процессом обучения.
# Одним из способов настройки гиперпараметров является grid, чтобы задавать компьютеру потребовать все возможные комбинации значений параметров.
# Используя модуль GridSearchCV из библиотеки Sklearn. Попытаемся найти наилучшую комбинацию гиперпараметров для построения классификатора для нашего набора данных.
#ПРИМЕЧАНИЕ: Этот метод получает требуемые оптимальные гиперпараметры.
#ВНИМАНИЕ: Операция ведется с использованием кросс-валидации. Если диапазон или число гиперпараметров велики, то вероятности могут исчисляться миллионами, и на завершение потребуется довольно много времени.
```

```
In [168]: # Проделаем поиск по схеме гиперпараметров с перекрестной проверкой, количеством блоков равно 10 (cv = 10), для
```

```
# модели случайного леса - Random Forest Regressor
parameters = {'n_estimators': [200, 300],
              'max_depth': [10, 15],
              'max_features': ['auto'],
              'criterion': ['mse']}
grid = GridSearchCV(estimator = rfr, param_grid = parameters, cv = 10)
grid.fit(x_train_1, y_train_1)
GridSearchCV(cv=10,
            estimator=RandomForestRegressor(max_depth=7, n_estimators=15),
            param_grid=[{"n_estimators": [200, 300], "max_depth": [10, 15], "max_features": ["auto"]}], refit=True, return_train_score=False, scoring='neg_mean_squared_error', error_score=0)
```

```
Out[168]:
{'criterion': 'mse',
 'max_depth': 15,
 'max_features': 'auto',
 'n_estimators': 200}
```

```
In [169]: #Находим оптимальные гиперпараметры для оптимальной модели
```

```
grid.best_params_
{'criterion': 'mse',
 'max_depth': 15,
 'max_features': 'auto',
 'n_estimators': 200}
```

```
Out[169]: #Находим оптимальные гиперпараметры для оптимальной модели
```

```
grid.best_estimator_
RandomForestRegressor(criterion='mse', max_depth=15, n_estimators=200)
```

```
R2-score RFR для прочности при растяжении, МПа: 0.963
```

```
#находим оптимальные гиперпараметры в новой модели 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)
```

```
#Очищаем память на тестовых наборах
```

```
mae_rfr_grid = mean_absolute_error(predictions_rfr_grid, y_test_1)
```

```
mae_rfr_grid
```

```
68.8708651237421
```

```
Out[171]: 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 [172]: # поиск гиперпараметров не дал улучшений для уже имеющейся модели RandomForestRegressor
```

```
# Вероятно, нужно указать более варийные параметры при работе с GridSearchCV
```

```
mae_df
```

```

Out[173]: Perceptron MAE
0 Support Vector 78.477914
1 RandomForest 76.589025
2 Linear Regression 61.986894
3 GradientBoosting 64.902175
4 KNeighbors 102.030259
5 DecisionTree 100.359294
6 SGD 181.506953
7 MLP 1808.547264
8 Lasso 69.474334
9 RandomForest_GridSearchCV 68.870307

In [174]: # Проведем поиск по сетке гиперпараметров с перекрестной проверкой, количеством блоков равно 10 (cv = 10), для
# Метода k ближайших соседей - K Neighbors Regressor - 5
knn = KNeighborsRegressor()
knn_params = {'n_neighbors': range(1, 301, 2),
              'weights': ['uniform', 'distance'],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']}
#Запускаем обучение модели. В конечном итоге модели будут использовать коэффициент детерминации (R^2)
# Если R^2 < 0, это значит, что разработанная модель даёт прогноз 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_3 = gs.best_estimator_
gs.best_params_
Fitting 10 folds for each of 1200 candidates, totalling 12000 fits
{'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'distance'}

Out[174]: {'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'distance'}

In [175]: #Выводим гиперпараметры для оптимальной модели
print(gs.best_estimator_.get_params())
gs1 = gs.best_estimator_
print("R2-score KNN для прочности при растяжении, MAE: (gs1.score(x_test_1, y_test_1).round(3))")
#NeighborsRegressor(algorithm='brute', n_neighbors=7, weights='distance')
R2_score_KNN для прочности при растяжении, MAE: 0.933
R2-score KNN для прочности при растяжении, MAE: 0.933

In [176]: #подаем оптимальные гиперпараметры в нашу модель метода k ближайших соседей
knn_grid = KNeighborsRegressor(algorithm = 'brute', n_neighbors = 7, weights = 'distance')
#обнуляем множества на текущий набор
mae_knn_grid = mean_absolute_error(predictions_knn_grid, y_test_1)
mae_knn_grid
2816938003547

Out[176]: 99.2816938003547

In [177]: 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[177]: Perceptron MAE
0 Support Vector 78.477914
1 RandomForest 76.589025
2 Linear Regression 61.986894
3 GradientBoosting 64.902175
4 KNeighbors 102.030259
5 DecisionTree 100.359294
6 SGD 181.506953
7 MLP 1808.547264
8 Lasso 69.474334
9 RandomForest_GridSearchCV 68.870307
10 KNeighbors_GridSearchCV 99.281694

In [178]: #Проведем поиск по сетке гиперпараметров с перекрестной проверкой, количеством блоков равно 10 (cv = 10), для
#дерева решений - Decision Tree Regressor - 6
criterion = ['mse', 'friedman_mse', 'absolute_error', 'poisson']
splitter = ['best', 'random']
max_depth = [3, 5, 7, 9, 11]
min_samples_leaf = [100, 150, 200]
min_samples_split = [200, 250, 300]
max_features = ['auto', 'sqrt', 'log2']
param_grid = {'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 конечном итоге модели будут использовать коэффициент детерминации (R^2)
# Если R^2 < 0, это значит, что разработанная модель даёт прогноз хуже, чем простое усреднение.
dt4 = DecisionTreeRegressor(criterion='mse', max_depth=5, min_samples_split=200)
gs4 = GridSearchCV(dt4, param_grid, cv = 10, verbose = 1, n_jobs =-1, scoring = 'r2')
gs4.fit(x_train_1, y_train_1)
dt4_3 = gs4.best_estimator_
gs4.best_params_
Fitting 10 folds for each of 1000 candidates, totalling 10000 fits
{'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'distance'}

Out[178]: {'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'distance'}

In [179]: #Выводим гиперпараметры для оптимальной модели
print(gs4.best_estimator_.get_params())
gs1 = gs4.best_estimator_
print("R2-score DTR для прочности при растяжении, MAE: (gs1.score(x_test_1, y_test_1).round(3))")
#DecisionTreeRegressor(criterion='poisson', max_depth=5, min_samples_leaf=100, min_samples_split=200)
R2-score DTR для прочности при растяжении, MAE: 0.834
R2-score DTR для прочности при растяжении, MAE: 0.834

In [180]: #подаем оптимальные гиперпараметры в нашу модель метода дерева решений
dtr_grid = DecisionTreeRegressor(criterion = 'poisson', max_depth = 7, max_features = 'auto',
                                  min_samples_leaf = 100, min_samples_split = 250)
#обнуляем модель
dtr_grid.fit(x_train_1, y_train_1)
predictions_dtr_grid = dtr_grid.predict(x_test_1)
#обнуляем множества на текущий набор
mae_dtr_grid = mean_absolute_error(predictions_dtr_grid, y_test_1)
mae_dtr_grid
6269974156563

Out[180]: 168.6269974156563

In [181]: 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

Out[181]: Perceptron MAE
0 Support Vector 78.477914
1 RandomForest 76.589025
2 Linear Regression 61.986894
3 GradientBoosting 64.902175
4 KNeighbors 102.030259
5 DecisionTree 100.359294
6 SGD 181.506953
7 MLP 1808.547264
8 Lasso 69.474334
9 RandomForest_GridSearchCV 68.870307
10 KNeighbors_GridSearchCV 99.281694
11 DecisionTree_GridSearchCV 168.626997

In [182]: pipe = Pipeline([('preprocessing', StandardScaler()), ('regression', 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__C__': [0.001, 0.01, 0.1, 1, 10, 100],
     'preprocessing__': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [RandomForestRegressor(n_estimators = 100)],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None],
     'regressor__n_estimators': [100],
     'preprocessing__': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [GradientBoostingRegressor()],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None],
     'regressor__n_estimators': [100],
     'preprocessing__': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [KNeighborsRegressor()],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None],
     'regressor__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
     'preprocessing__': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [DecisionTreeRegressor()],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None],
     'regressor__criterion': ['mse', 'friedman_mse', 'absolute_error', 'poisson'],
     'preprocessing__': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [MLPRegressor(random_state = 1, max_iter = 500)],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None],
     'regressor__alpha': [0.001, 0.01, 0.1, 1, 10, 100],
     'preprocessing__': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [LinearRegression(pipe_alpha = 0.1)],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]},
    {'grid': GridSearchCV(pipe, param_grid, cv = 10)},
    {'grid': fit(x_train_1, np.ravel(y_train_1)),
     'grid': best_params_},
    {'grid': best_params_['grid'].best_params_},
    {'grid': best_params_['grid'].best_score_},
    {'grid': best_params_['grid'].best_params_['regressor__C']},
    {'grid': best_params_['grid'].best_params_['regressor__epsilon']}]

Начальные параметры:
{'preprocessing': MinMaxScaler(), 'regressor': Lasso(alpha=0.1)}

Найденное значение правильности перекрестной проверки: 0.97
Правильность на тестовом наборе: 0.97

In [183]: print("Наилучшая модель:\n{}\n".format(grid.best_estimator_))

```

```
Напечатанная модель:  
Pipeline(steps=[('preprocessing', MinMaxScaler()),  
              ('regressor', Lasso(alpha=0.1))])
```

Прогнозируем модуль упругости при растяжении, ГПа

```
In [184]: #разделяем на тестовую, тренировочную выборки, делая предварительные обработки и случайные перестановки  
x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(  
    df[["Количество наблюдений"]],  
    df[["Модуль упругости при растяжении, ГПа"]],  
    test_size = 0.3,  
    random_state = 42)
```

```
In [185]: #Подработка приближности размера  
df_norm.shape[0] * x_train_2.shape[0] * x_test_2.shape[0]
```

```
Out[185]: 0
```

```
In [186]: x_train_2.head()
```

```
Out[186]: Соотношение матрица-наполнитель Плотность, кг/м3 модуль упругости, ГПа Количество ствердителя, м.% Содержание эпоксидных групп, %_2 Температура вспышки, С_2 Поверхностная плотность, г/м2 Прочность при растяжении, МПа Потребление смолы, г/м2 Угол нашивки Шаг нашивки Плотность нашивки  
481 0.000528 0.589066 0.132149 0.020997 0.006254 0.071450 0.186083 0.767682 0.073037 0.000291 0.002285 0.015563  
650 0.000430 0.617300 0.232509 0.027252 0.007346 0.075579 0.081347 0.739980 0.056839 0.000308 0.001350 0.018310  
483 0.000580 0.585434 0.170519 0.043402 0.007074 0.096550 0.132807 0.772305 0.041487 0.000295 0.002249 0.018929  
355 0.000911 0.545896 0.306130 0.024469 0.006610 0.070488 0.159918 0.757679 0.050148 0.000000 0.001774 0.012540  
850 0.000573 0.595720 0.263992 0.034166 0.007519 0.089582 0.095597 0.740762 0.086336 0.000322 0.002765 0.017180
```

```
In [187]: y_train_2
```

```
Out[187]: Модуль упругости при растяжении, ГПа  
552 69.573625  
735 80.614999  
554 71.873767  
403 68.314525  
948 72.997468  
... ...  
120 74.191919  
306 70.325533  
959 77.959289  
497 70.199234  
113 72.625213  
645 rows × 1 columns
```

```
In [188]: y_train_2.shape
```

```
Out[188]: (645, 1)
```

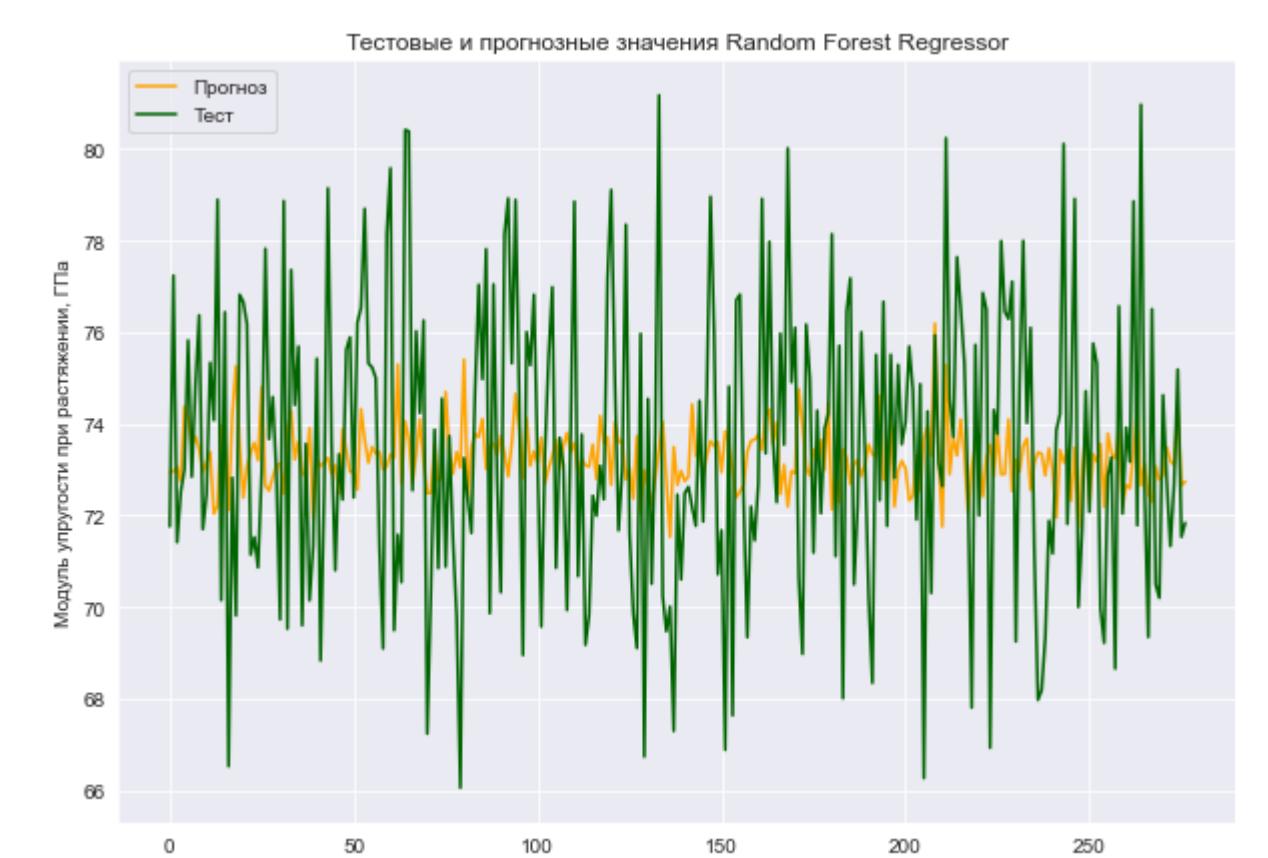
```
In [189]: #Функция для сглаживания результатов предсказаний с моделью, выдающей среднее значение по тестовой выборке  
def mean_model(y_test_2):  
    return np.mean(y_test_2) for _ in range(len(y_test_2))  
y_2_pred_mean = mean_model(y_test_2)
```

```
In [190]: #Проверка различных моделей при стандартных параметрах  
#Метод опорных векторов - 1
```

```
In [191]: svr = make_pipeline(StandardScaler(), SVR(kernel = 'rbf', C = 500.0, epsilon = 1.0))  
#Подготовка модели  
svr.fit(x_train_2, np.ravel(y_train_2))  
#Вычисление коэффициента детерминации  
y_pred_svr = svr.predict(x_test_2)  
mse_svr = mean_squared_error(y_pred_svr, y_test_2)  
mse_svr_alert = mean_squared_error(y_test_2, y_pred_svr)  
print("Support Vector Regression Results Train:")  
print("Test score: {:.2f}".format(svr.score(x_train_2, y_train_2))) #Скор для тренировочной выборки  
print("Support Vector Regression Results Test:")  
print("MAE: {:.2f} ".format(svr.MAE))  
print("MSE: {:.2f} ".format(mse_svr))  
print("RMSE: {:.2f} ".format(np.sqrt(mse_svr)))  
print("Support Vector Regression Results Test:")  
print("Test score: {:.2f}".format(svr.score(x_test_2, y_test_2))) #Скор для тестовой выборки  
Support Vector Regression Results Train:  
Test score: 0.505356442  
Support Vector Regression Results:  
SVR_MAE: 3  
SVR_MSE: 0.05  
SVR_RMSE: 0.21  
SVR(score: 0.505356442  
Test score: 446546.79  
In [192]: #Проверка различных моделей, выдающей среднее значение  
mse_lin_elast2_mean = mean_squared_error(y_test_2, y_2_pred_mean)  
print("MAE for mean target: ", mse_lin_elast2_mean)  
print("MSE for mean target: ", mse_lin_elast2_mean)  
print("RMSE for mean target: ", np.sqrt(mse_lin_elast2_mean))  
MAE for mean target: 2.57849935756179  
MSE for mean target: 9.910360742108628  
RMSE for mean target: 3.1408754397442  
In [193]: 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(x_test_2, y_test_2, label = "Тест", color = "green")  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Модуль упругости при растяжении, ГПа")  
plt.legend()  
plt.grid(True);
```

```
In [194]: #Метод случайного леса - Random Forest Regressor - 2
```

```
In [195]: #Построение модели и фузуллизация метода случайной леса  
rf2 = RandomForestRegressor(n_estimators = 15, max_depth = 7, random_state = 33)  
rf2.fit(x_train_2, y_train_2)  
y2_pred_forest = rf2.predict(x_test_2)  
mse_rf2 = mean_squared_error(y2_pred_forest, y_test_2)  
mse_rf2_elast2 = mean_squared_error(y_test_2,y2_pred_forest)  
print("Random Forest Regressor Results Train:")  
print("Test score: {:.2f}".format(rf2.score(x_train_2, y_train_2))) #Скор для тренировочной выборки  
print("Random Forest Regressor Results")  
print("RF_MAE: ", round(mean_absolute_error(y_test_2, y2_pred_forest)))  
print("RF_MSE: {:.2f} ".format(mean_squared_error(y_test_2, y2_pred_forest)))  
print("RF_RMSE: {:.2f} ".format(np.sqrt(mse_rf2_elast2)))  
print("Test score: {:.2f} ".format(rf2.score(x_test_2, y_test_2))) #Скор для тестовой выборки  
Random Forest Regressor Results:  
Test score: 0.63052889  
Random Forest Regressor Results:  
RF_MAE: 3  
RF_MSE: 0.48  
RF_RMSE: 10.26  
RF_RMSE: 3.20  
Test score: 0.56293745  
In [196]: plt.figure(figsize=(10, 7))  
plt.title("Тестовые и прогнозные значения Random Forest Regressor")  
plt.plot(y2_pred_forest, label = "Прогноз", color = "orange")  
plt.plot(y_test_2.values, label = "Тест", color = "darkgreen")  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Модуль упругости при растяжении, ГПа")  
plt.legend()  
plt.grid(True);
```



In [197]: #Метод линейной регрессии - Linear Regression - 3

```
In [198]: #Моделирование и визуализация линейной регрессии
l2 = fitLinearRegression(x_train_1)
y_pred_l2 = l2.predict(x_test_1)
mae_lr2 = mean_absolute_error(y_pred_l2, y_test_2)
mse_lr2 = mean_squared_error(y_pred_l2, y_test_2)
print("Linear Regression Results Train:")
print("Linear Regression Results test:")
print("Test score: {:.2f}".format(l2.score(x_train_2, y_train_2)))
print("Linear Regression Results:")
print("lr.MAE: ", round(mae_lr2))
print("lr.MAPE: {:.2f}".format(mean_absolute_percentage_error(y_test_2, y_pred_l2)))
print("lr.MSE: {:.2f}".format(np.sqrt(mse_lr2)))
print("lr.RMSE: {:.2f}".format(np.sqrt(mse_lr2)))
print("Test score: {:.2f}".format(l2.score(x_test_2, y_test_2))) # Скор для тестовой выборки
```

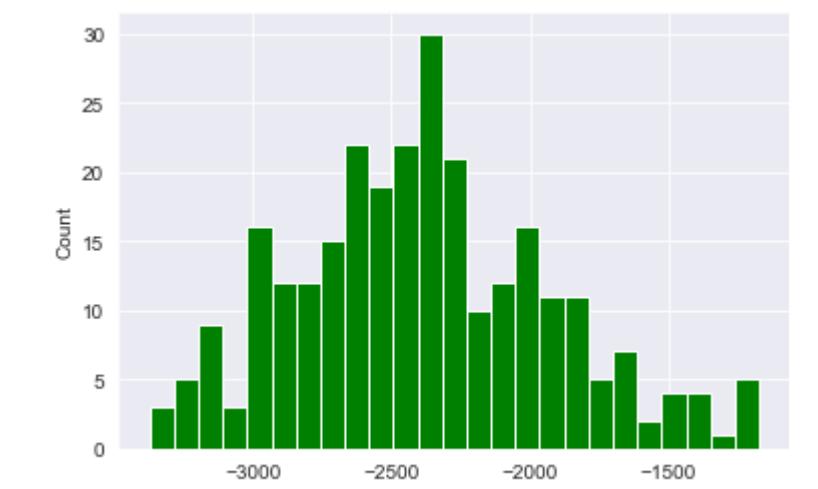
```
Linear Regression Results Train:
Test score: 79193846.55
Linear Regression Results:
lr.MAE: 2387
lr.MAPE: 0.15
lr.MSE: 5911046.86
lr.RMSE: 2431.26
Test score: 70414400.77
```

```
In [199]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Linear Regression")
plt.plot(y_pred_l2, label = "Прогноз", color = "orange")
plt.plot(y_test_2.values, label = "Тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



In [200]: #Визуализация гистограммы распределения ошибки

```
In [201]: error = y_test_2 - y_pred_l2
plt.hist(error, bins = 25, color = "g")
plt.xlabel("Prediction Error")
_ = plt.ylabel("Count")
```



In [202]: #Метод градиентного бустинга - Gradient Boosting Regressor - 4

```
In [203]: gbr2 = make_pipeline(GradientBoostingRegressor())
gbr2.fit(x_train_2, y_train_2)
y_pred_gbr2 = gbr2.predict(x_test_2)
mae_gbr2 = mean_absolute_error(y_pred_gbr2, y_test_2)
mse_gbr2 = mean_squared_error(y_pred_gbr2, y_test_2)
print("Gradient Boosting Regressor Results Train:")
print("Gradient Boosting Regressor Results test:")
print("Gbr.MAE: ", round(mae_gbr2))
print("Gbr.MAPE: {:.2f}".format(mean_absolute_percentage_error(y_test_2, y_pred_gbr2)))
print("Gbr.MSE: {:.2f}".format(np.sqrt(mse_gbr2)))
print("Gbr.RMSE: {:.2f}".format(np.sqrt(mse_gbr2)))
print("Test score: {:.2f}".format(gbr2.score(x_test_2, y_test_2))) # Скор для тестовой выборки
```

```
Gradient Boosting Regressor Results Train:
Test score: 575259.38
Gradient Boosting Regressor Results:
Gbr.MAE: 0.04
Gbr.MAPE: 0.04
Gbr.MSE: 11.05
Gbr.RMSE: 3.32
Test score: 509411.39
```

```
In [204]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Gradient Boosting Regressor")
plt.plot(y_pred_gbr2, label = "Прогноз", color = "orange")
plt.plot(y_test_2.values, label = "Тест", color = "green")
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```

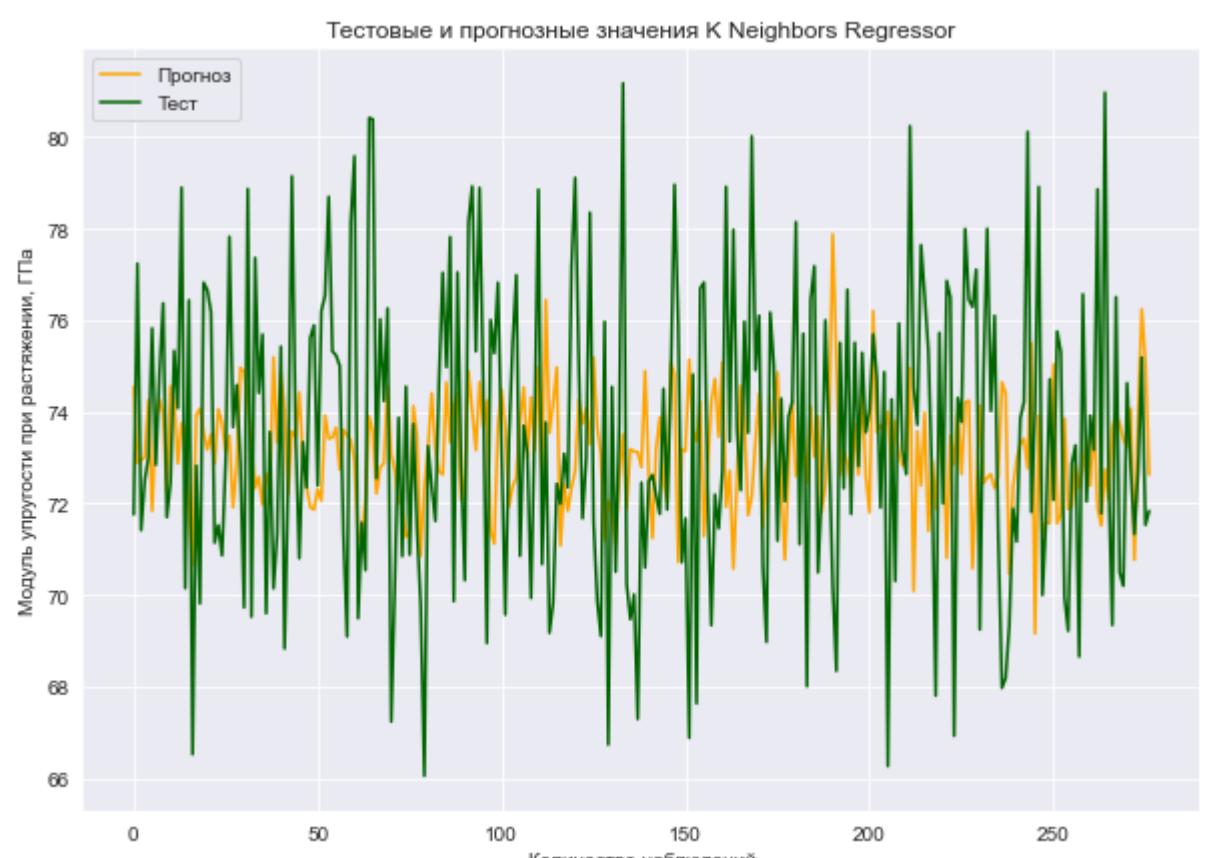


```
In [205]: # Метод K ближайших соседей - KNeighbors Regressor - 5
knn2 = KNeighborsRegressor(n_neighbors=5)
knn2.fit(x_train_2, y_train_2)
y_pred_knn2 = knn2.predict(x_test_2)
mae_knn2 = mean_absolute_error(y_pred_knn2, y_test_2)
mse_knn2 = mean_squared_error(y_pred_knn2, y_test_2)
print("K Neighbors Regressor Results Train:")
print("Test score: {:.2f}".format(knn2.score(x_train_2, y_train_2))) # Скор для тренировочной выборки
print("KNeighbors Regressor Results:")
print("KNN.MAE: ", round(mae_knn2))
print("KNN.MAPE: {:.2f}".format(mean_absolute_percentage_error(y_test_2, y_pred_knn2)))
print("KNN.MSE: {:.2f}".format(np.sqrt(mse_knn2)))
print("KNN.RMSE: {:.2f}".format(np.sqrt(mse_knn2)))
```

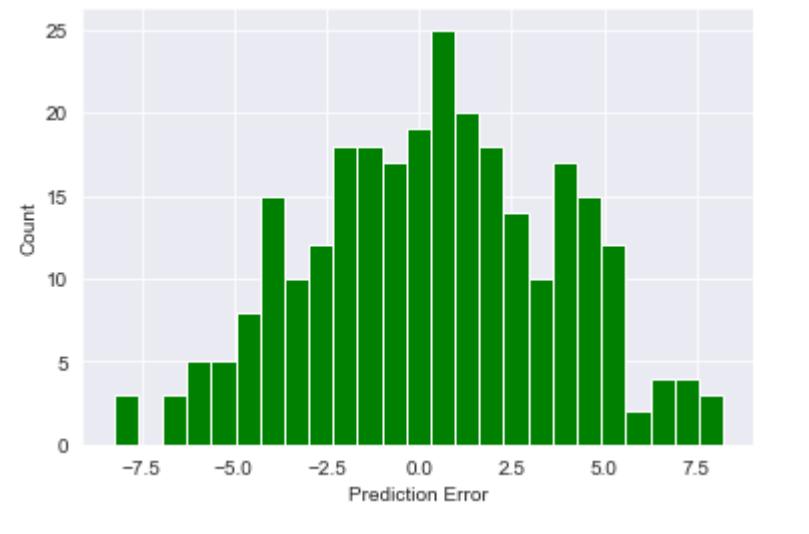
```
print("KNN_RMSE: {:.2f}".format(np.sqrt(msse_knn_elast2)))
print("Test score: {:.2f}").format(knn2.score(x_test_2, y_test_2))# Скор для тестовой выборки
```

```
K Neighbors Regressor - Results Train:
Test score: 0.16
K Neighbors Regressor - Results:
KNN_MAE: 3
KNN_MAPE: 0.04
KNN_RMSE: 0.47
KNN_RMSLE: 0.40
Test score: 0.17
```

```
In [206]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения K Neighbors Regressor")
plt.plot(y_pred_knn2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



```
In [207]: #Визуализация гистограммы распределения ошибки
err = y_test_2 - y_pred_knn2
plt.hist(err, bins = 25, color = "g")
plt.xlabel('Prediction Error')
plt.ylabel('Count')
```



```
In [208]: #Дерево решений - Decision Tree Regressor - 6
dtr2 = DecisionTreeRegressor()
dtr2.fit(x_train_2, y_train_2.values)
y_pred_dtr2 = dtr2.predict(x_test_2)
mae_dtr2 = mean_absolute_error(y_pred_dtr2, y_test_2)
mse_dtr_elast2 = mean_squared_error(y_test_2,y_pred_dtr2)
print("Decision Tree Regression Results Train:")
print("Decision Tree Regression Results: (.2f)".format(dtr2.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print("DTR_MAE: ", round(mean_absolute_error(y_test_2, y_pred_dtr2)))
print("DTR_RMSE: {:.2f}".format(np.sqrt(mse_dtr_elast2)))
print("DTR_MAPE: {:.2f}").format((np.sum(np.abs((y_test_2 - y_pred_dtr2)))) / np.sum(np.abs(y_test_2)))
print("DTR_RMSLE: {:.2f}").format(np.sqrt(mean_absolute_percentage_error(y_test_2, y_pred_dtr2)))
print("Test score: {:.2f}").format(dtr2.score(x_test_2, y_test_2))# Скор для тестовой выборки
```

```
Decision Tree Regressor Results Train:
Test score: 1.00
Decision Tree Regressor Results:
DTR_MAE: 2362
DTR_RMSE: 2405.58
DTR_RMSLE: 2405.89
DTR_MAPE: 32.17
Test score: 582706.91
```

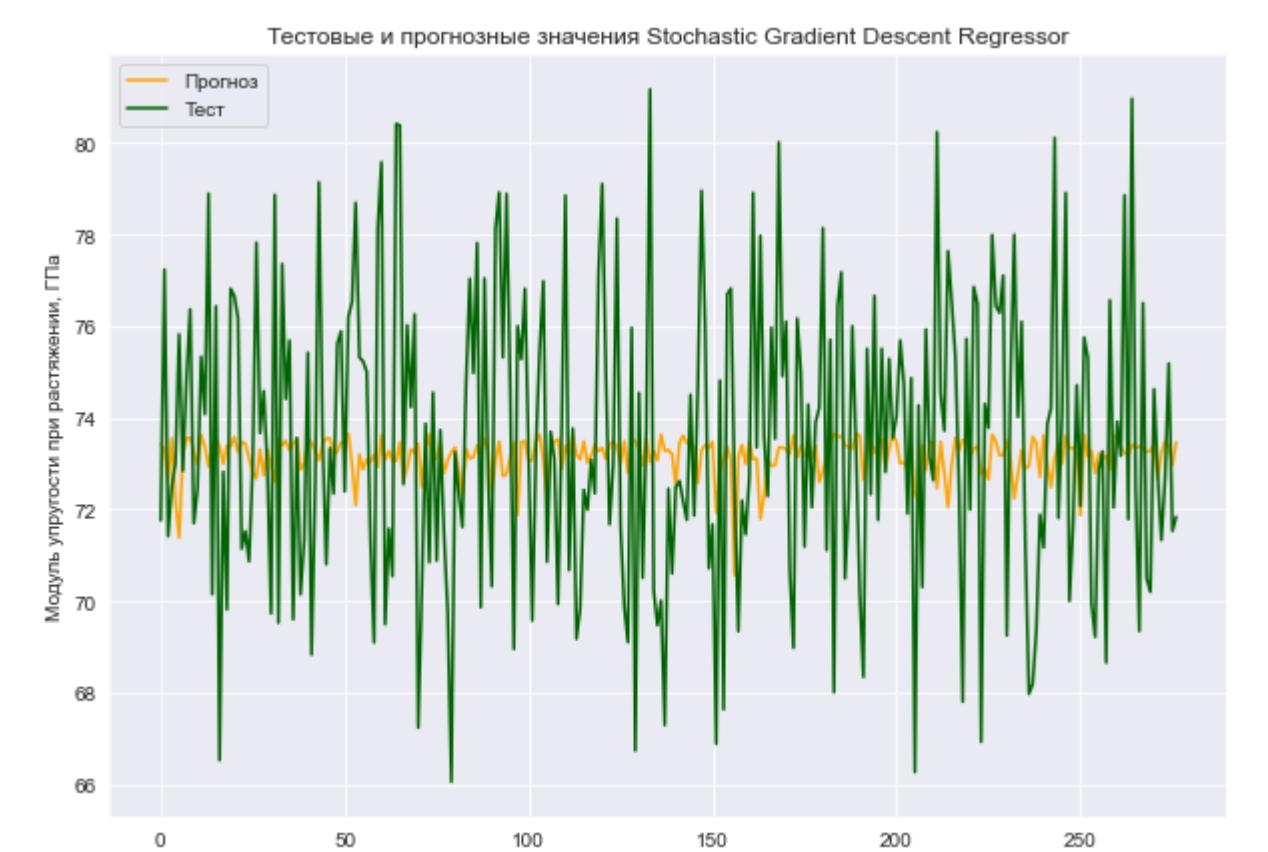
```
In [209]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Decision Tree Regressor")
plt.plot(y_pred_dtr2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



```
In [210]: # Стохастический градиентный спуск (SGD) - Stochastic Gradient Descent Regressor - 7
sgd2 = SGDRegressor()
sgd2.fit(x_train_2, y_train_2)
y_pred_sdg2 = sgd2.predict(x_test_2)
mae_sdg2 = mean_absolute_error(y_pred_sdg2, y_test_2)
mse_sdg_elast2 = mean_squared_error(y_test_2,y_pred_sdg2)
print("Stochastic Gradient Descent Regressor Results Train:")
print("Stochastic Gradient Descent Regressor Results: (.2f)".format(sgd2.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print("SGD_MAE: ", round(mean_absolute_error(y_test_2, y_pred_sdg2)))
print("SGD_RMSE: {:.2f}").format(np.sqrt(mse_sdg_elast2))
print("SGD_MSE: {:.2f}").format((np.sum(np.abs((y_test_2 - y_pred_sdg2)))) / np.sum(np.abs(y_test_2)))
print("SGD_RMSLE: {:.2f}").format(np.sqrt(mean_absolute_percentage_error(y_test_2, y_pred_sdg2)))
print("Test score: (.2f)".format(sgd2.score(x_test_2, y_test_2)))# Скор для тестовой выборки
```

```
Stochastic Gradient Descent Regressor Results Train:
Test score: 602607.03
Stochastic Gradient Descent Regressor Results:
SGD_MAE: 3
SGD_MSE: 101.24
SGD_RMSLE: 10.20
SGD_MAPE: 0.04
Test score: -536731.25
```

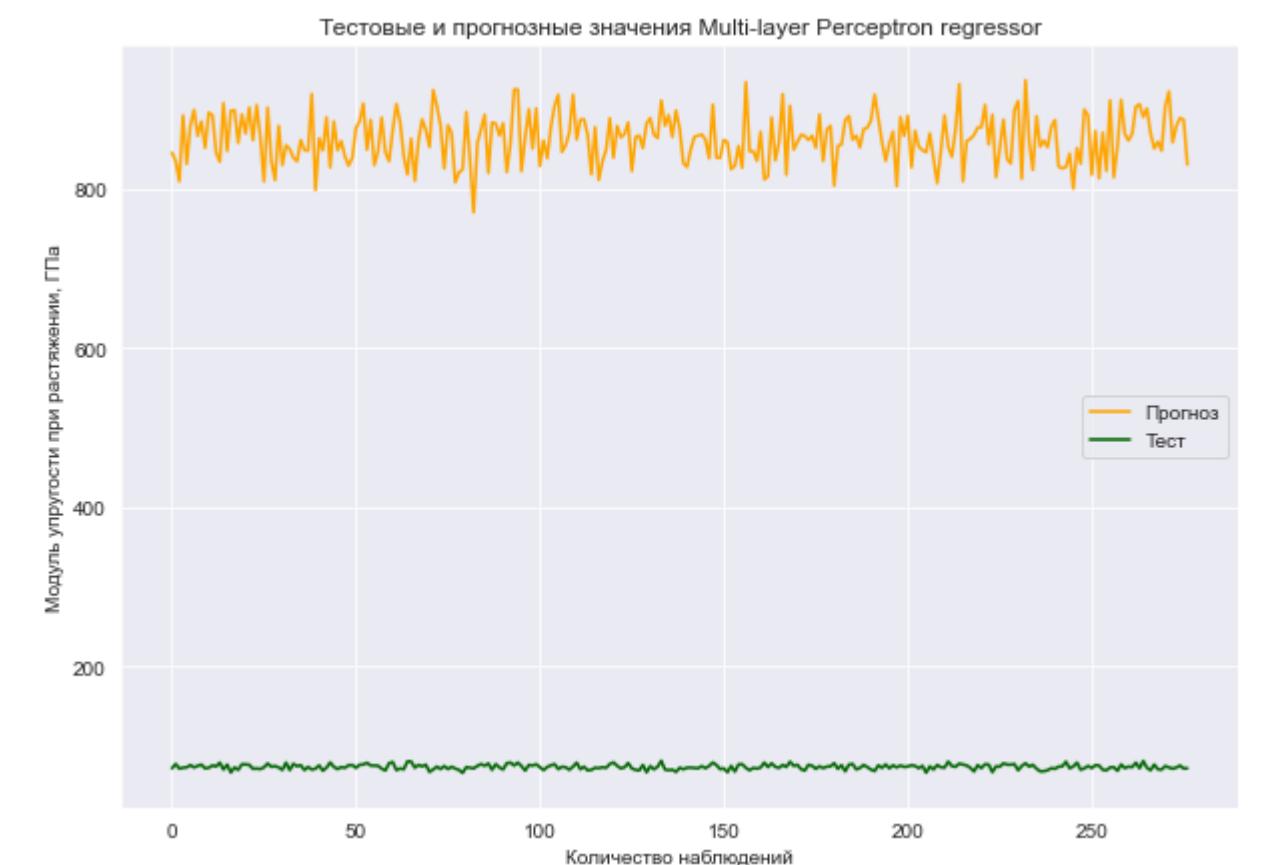
```
In [211]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Stochastic Gradient Descent Regressor")
plt.plot(y_pred_sdg2, label = "Прогноз", color = 'orange')
plt.plot(y_test_2.values, label = "тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



```
In [212]: # Многослойный перцептрон - Multi-layer Perceptron regressor - 8
mlp = MLPRegressor(random_state = 1, max_iter = 500)
mlp.fit(x_train_2, y_train_2)
y_pred_mlp2 = mlp.predict(x_test_2)
mae_mlp2 = mean_absolute_error(y_pred_mlp2, y_test_2)
mse_mlp_elast2 = mean_squared_error(y_pred_mlp2, y_test_2, y_true=y_test_2)
print("Multi-layer Perceptron regressor Results Train:")
print("Test score: {:.2f}".format(mlp.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print("SGD_MAE: ", round(mean_absolute_error(y_test_2, y_pred_mlp2)))
print("SGD_MAPE: ", round(mean_absolute_percentage_error(y_test_2, y_pred_mlp2)))
print("SGD_MSE: {:.2f}".format(mse_mlp_elast2))
print("SGD_RMSE: {:.2f}".format(np.sqrt(mse_mlp_elast2)))
print("Test score: {:.2f}".format(mlp.score(x_test_2, y_test_2)))# Скор для тестовой выборки
print("Test score: {:.2f}").format(mlp.score(x_test_2, y_test_2))

Multi-layer Perceptron regressor Results Train:
Test score: 71118.12
Multi-layer Perceptron regressor Results:
SGD_MAE: 19.75
SGD_MAPE: 19.75
SGD_MSE: 622539.68
SGD_RMSE: 789.01
Test score: 62816.06
```

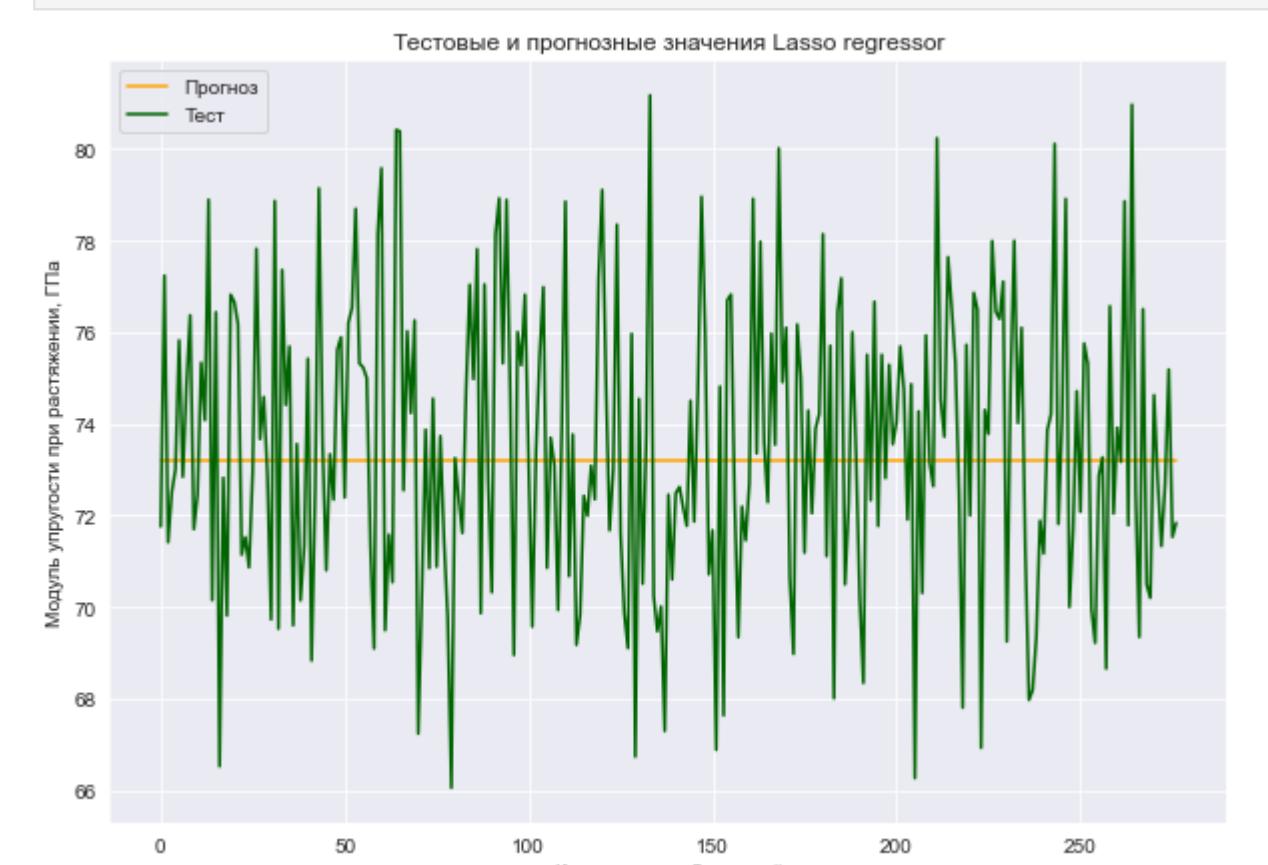
```
In [213]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Multi-layer Perceptron regressor")
plt.plot(y_pred_mlp2, label = "Прогноз", color = "orange")
plt.plot(y_test_2, label = "тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Модель упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



```
In [214]: # Лассо регрессия - the Lasso - 9
clf2 = linear_model.Lasso(alpha = 0.1)
clf2.fit(x_train_2, y_train_2)
y_pred_clf2 = clf2.predict(x_test_2)
mae_clf2 = mean_absolute_error(y_pred_clf2, y_test_2)
mse_clf_elast2 = mean_squared_error(y_test_2,y_pred_clf2)
print("Lasso regressor Results Train:")
print("Test score: {:.2f}".format(clf2.score(x_train_2, y_train_2)))# Скор для тренировочной выборки
print("Lasso regression Results:")
print("SGD_MAE: ", round(mean_absolute_error(y_test_2, y_pred_clf2)))
print("SGD_MAPE: {:.2f}".format(mean_absolute_percentage_error(y_test_2, y_pred_clf2)))
print("SGD_MSE: {:.2f}".format(mse_clf_elast2))
print("SGD_RMSE: {:.2f}".format(np.sqrt(mse_clf_elast2)))
print("Test score: {:.2f}".format(clf2.score(x_test_2, y_test_2)))# Скор для тестовой выборки

Lasso regressor Results Train:
Test score: 669857.32
Lasso regressor Results:
SGD_MAE: 0.04
SGD_MAPE: 0.04
SGD_MSE: 10.00
SGD_RMSE: 3.16
Test score: 597457.66
```

```
In [215]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Lasso regressor")
plt.plot(y_pred_clf2, label = "Прогноз", color = "orange")
plt.plot(y_test_2.values, label = "тест", color = "darkgreen")
plt.xlabel("Количество наблюдений")
plt.ylabel("Модель упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);
```



```
In [216]: # Создание новой модели по методу MAE
mae_df2 = {'Perceptron': ['Support Vector', 'RandomForest', 'Linear Regression', 'GradientBoosting', 'KNeighbors', 'DecisionTree', 'SGD', 'MLP', 'Lasso'], 'MAE': [mae_svr2, mae_rfr2, mae_lr2, mae_gbr2, mae_knr2, mae_dtr2, mae_sdg2, mae_mlp2, mae_clf2]}

In [217]: # Проблема поиск по семье 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],
               'min_features': ['auto'],
               'random_state': [33] }
grid21 = GridSearchCV(estimator = rfr2, param_grid = parameters, cv=10)
grid21.fit(x_train_2, y_train_2)

Out[217]: GridSearchCV(cv=10,
                     estimator=RandomForestRegressor(max_depth=7, n_estimators=15,
                                                     random_state=33),
                     param_grid=[{'criterion': 'mse', 'max_depth': [9, 15], 'min_features': ['auto'], 'n_estimators': [200, 300]}])
```

```
In [218]: grid21.best_params_
Out[218]: {'criterion': 'mse',
           'max_depth': 9,
           'min_features': 'auto',
           'n_estimators': 300}

In [219]: # Будем гиперпараметры для оптимальной модели
print(grid21.best_estimator_)
knr_u = grid21.best_estimator_
print("R2-score RFR для модели упругости при растяжении: ({knr_u.score(x_test_2, y_test_2).round(3)})")

RandomForestRegressor(criterion='mse', max_depth=9, n_estimators=300,
                      random_state=33)
```

```
R2-score RFR для модели упругости при растяжении: -0.035

In [220]: # подберём оптимальные гиперпараметры в нашу модель случайного леса
rfr21_grid = RandomForestRegressor(n_estimators=200, criterion="mse", max_depth=15, max_features="auto")
rfr21_grid.fit(x_train_2, y_train_2)
```

```
predictions_rfr21_grid = rfr21_grid.predict(x_test_2)
#Оцениваем точность на тестовом наборе
mae_rfr21_grid = mean_absolute_error(predictions_rfr21_grid, y_test_2)
mae_rfr21_grid
```

Out[228]: 2.6241009351035056

```
In [221]: new_row_in_mae_df = {'Perceptron': 'RandomForest1_GridSearchCV', 'MAE': mae_rfr21_grid}
mae_df.append(new_row_in_mae_df, ignore_index = True)
```

In [222]: mae_df

	Перескор	МАЕ
0	Support Vector	78.477914
1	RandomForest	76.590025
2	Linear Regression	61.986894
3	GradientBoosting	64.902175
4	KNeighbors	102.030259
5	DecisionTree	100.359294
6	SGD	181.506953
7	MLP	1808.547264
8	Lasso	69.474334
9	RandomForest_GridSearchCV	68.870307
10	KNeighbors_GridSearchCV	99.281694
11	DecisionTree_GridSearchCV	168.624997
12	RandomForest1_GridSearchCV	2.624101

```
In [223]: # Проведем поиск по семи гиперпараметров с перекрестной проверкой, количеством блоков равно 10 (cv = 10), для
# каждого из ближайших соседей - K Neighbors Regressor - 5
knn21 = KNeighborsRegressor()
knn21_params = {"n_neighbors": range(1, 301, 2),
                 "weights": ['uniform', 'distance'],
                 "algorithm": ['auto', 'ball_tree', 'kd_tree', 'brute']}
#Запускаем обучение модели. В конечном итоге модели будут использовать коэффициент детерминации (R^2)
# Если R^2 < 0, это значит, что разработанная модель даёт прогноз даже хуже, чем простое усреднение.
gs21 = GridSearchCV(gs121, param_grid=knn21_params, cv = 10, verbose = 1, n_jobs=-1, scoring = 'r2')
gs21.fit(x_train_2, y_train_2)
knn_21 = gs21.best_estimator_
gs21.best_params_
```

Fitting 10 folds for each of 1200 candidates, totalling 12000 fits

```
Out[223]: {'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'distance'}
```

#Выборка гиперпараметров для оптимальной модели

```
print(gs21.best_estimator_)
gs21 = gs21.best_estimator_
print("R2-score KNN для модуля упрости при растяжении: (gs121.score(x_test_2, y_test_2).round(3))")
```

KNeighborsRegressor(n_neighbors=133)

R2-score KNN для модуля упрости при растяжении: -0.037

```
In [224]: #Поставим оптимальные гиперпараметры в новую модель метода к ближайшим соседям
knn21_grid = KNeighborsRegressor(algorithm = "brute", n_neighbors = 7, weights = "distance")
#Выборка гиперпараметров
knn21_grid.fit(x_train_2, y_train_2)
```

```
predictions_knn21_grid = knn21_grid.predict(x_test_2)
#Оцениваем точность на тестовом наборе
mae_knn21_grid = mean_absolute_error(predictions_knn21_grid, y_test_2)
mae_knn21_grid
```

Out[224]: 2.7577880199083345

```
In [225]: new_row_in_mae_df = {'Perceptron': 'KNeighbors1_GridSearchCV', 'MAE': mae_knn21_grid}
```

```
mae_df.append(new_row_in_mae_df, ignore_index=True)
```

mae_df

	Перескор	МАЕ
0	Support Vector	78.477914
1	RandomForest	76.590025
2	Linear Regression	61.986894
3	GradientBoosting	64.902175
4	KNeighbors	102.030259
5	DecisionTree	100.359294
6	SGD	181.506953
7	MLP	1808.547264
8	Lasso	69.474334
9	RandomForest_GridSearchCV	68.870307
10	KNeighbors_GridSearchCV	99.281694
11	DecisionTree_GridSearchCV	168.624997
12	RandomForest1_GridSearchCV	2.624101
13	KNeighbors1_GridSearchCV	2.757781

```
In [226]: # Проведем поиск по семи гиперпараметров с перекрестной проверкой, количеством блоков равно 10 (cv = 10), для
# дерева решений - Decision Tree Regression - 6
criterion1 = ["squared_error", "friedman_mse", "absolute_error", "poisson"]
splitter1 = ["best", "random"]
max_depth1 = [2, 3, 5, 7, 10]
min_samples_leaf21 = [100, 150, 200]
min_samples_split21 = [200, 250, 300]
max_features21 = ['auto', 'sqrt', 'log2']
param_grid21 = {"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 конечном итоге модели будут использовать коэффициент детерминации (R^2)
# Если R^2 < 0, это значит, что разработанная модель даёт прогноз даже хуже, чем простое усреднение.
gs21 = GridSearchCV(gs121, param_grid21, cv = 10, verbose = 1, n_jobs=-1, scoring = 'r2')
gs21.fit(x_train_2, y_train_2)
dtr_21 = gs21.best_estimator_
gs21.best_params_
```

Fitting 10 folds for each of 1088 candidates, totalling 10880 fits

```
Out[226]: {'criterion': 'friedman_mse',
            'max_depth': 9,
            'min_features': 'auto',
            'min_samples_leaf': 150,
            'min_samples_split': 250,
            'splitter': 'random'}
```

#Выборка гиперпараметров для оптимальной модели

```
print(gs21.best_estimator_)
gs21 = gs21.best_estimator_
print("R2-score DTR для модуля упрости при растяжении: (gs121.score(x_test_2, y_test_2).round(3))")
```

DecisionTreeRegressor(criterion='friedman_mse', max_depth=9,

max_features='auto', min_samples_leaf=150,

min_samples_split=250, splitter='random')

R2-score DTR для модуля упрости при растяжении: -0.027

```
In [227]: #Поставим оптимальные гиперпараметры в новую модель метода деревьев решений
dtr21_grid = DecisionTreeRegressor(criterion='poisson', max_depth=7, max_features='auto',
                                    min_samples_leaf=100, min_samples_split=200)
#Обучаем модель
dtr21_grid.fit(x_train_2, y_train_2)
```

predictions_dtr21_grid = dtr21_grid.predict(x_test_2)

#Оцениваем точность на тестовом наборе

mae_dtr21_grid = mean_absolute_error(predictions_dtr21_grid, y_test_2)

mae_dtr21_grid

Out[227]: 2.6148420213878336

```
In [228]: new_row_in_mae_df = {'Perceptron': 'DecisionTree_GridSearchCV', 'MAE': mae_dtr21_grid}
```

```
mae_df.append(new_row_in_mae_df, ignore_index=True)
```

mae_df

```

Out[230]:
      Perceptor    MAE
0   Support Vector  78477914
1   RandomForest  76590025
2  Linear Regression  61986894
3  GradientBoosting  64902175
4   KNeighbors  102.03059
5  DecisionTree  100.359294
6      SGD  1815.06953
7      MLP  1808.547264
8      Lasso  69.474334
9  RandomForest_GridSearchCV  68.870307
10 KNeighbors_GridSearchCV  99.281694
11 DecisionTree_GridSearchCV  168.624997
12 RandomForest_GridSearchCV  2.624101
13 KNeighbors1_GridSearchCV  2.757781
14 DecisionTree1_GridSearchCV  2.614842

In [231]:
pipe2 = Pipeline([('preprocessing', StandardScaler()), ('regressor', SVR()))]
param_grid2 = [
    {'regressor': [SVR()], 'preprocessing': [StandardScaler(), MinMaxScaler(), None],
     'regressor__C': [0.001, 0.01, 0.1, 1, 10, 100],
     'regressor__gamma': [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=max_iter=300)],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]},
    {'regressor': [linear_model.Lasso(alpha=0.1)],
     'preprocessing': [StandardScaler(), MinMaxScaler(), None]}]
grid2 = GridSearchCV(pipe2, param_grid2, cv=10)
grid2.fit(x_train_1, np.ravel(y_train_2))
print("Наилучшее значение правильности перекрестной проверки: {:.2f}".format(grid2.best_score_))
print("Правильность на тестовом наборе: {:.2f}".format(grid2.score(x_test_2, y_test_2)))
print("Наилучшие параметры: " + str(grid2.best_params_))
print("Наилучшее значение правильности перекрестной проверки: 0.68")
print("Правильность на тестовом наборе: 70466489.04")

In [232]:
print("Наилучшая модель:\n" + str(grid2.best_estimator_))

Наилучшая модель:
Pipeline(steps=[('preprocessing', MinMaxScaler()),
               ('regressor', Lasso(alpha=0.1))])

Написать нейронную сеть, которая будет рекомендовать соотношение матрица-наполнитель.

In [233]:
# Сформируем входы и выход для модели
tv = df['Соотношение матрица-наполнитель']
tr_v = tv.loc[:, tv.columns != 'Соотношение матрица-наполнитель']

# Разделим выборку на обучающую и тестовую
x_train, x_test, y_train, y_test = train_test_split(tr_v, tv, test_size = 0.3, random_state = 14)

# Нормализуем данные
x_train_n = tf.keras.layers.Normalization(axis = -1)
x_train_n.adapt(np.array(x_train))

In [234]:
def create_model(layers=[32], act="softmax", opt="SGD", dr=0.1):
    seed = 7
    np.random.seed(seed)
    tf.random.set_seed(seed)

    model = Sequential()
    model.add(Dense(layers[0], input_dim=x_train.shape[1], activation=act))
    for i in range(1,len(layers)):
        model.add(Dense(layers[i], activation=act))
    model.add(Dropout(dr))
    model.add(Dense(3, activation='tanh')) # выходной слой
    model.compile(loss="binary_crossentropy", optimizer=opt, metrics=['mae', 'accuracy'])

    return model

In [235]:
# спарим модель
model = KerasClassifier(build_fn=create_model, verbose=0)

# определим параметры
batch_size = [4, 10, 20, 50, 100]
epochs = [10, 50, 100, 200, 300]
param_grid = dict(batch_size=batch_size, epochs=epochs)

# поиск оптимальных параметров
grid = GridSearchCV(estimator=model,
                    param_grid=param_grid,
                    cv=10,
                    verbose=1, n_jobs=-1)

grid_result = grid.fit(x_train, y_train)
Fitting 10 folds for each of 25 candidates, totalling 250 fits

In [236]:
# результаты
print("Best: %s using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, std, param in zip(means, stds, params):
    print("%s with: %s (%s, %s)" % (mean, std, param))

Best: 0.001538 using {batch_size: 4, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 50}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 100}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 200}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 300}
0.001538 (0.004615) with: {batch_size: 10, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 10, 'epochs': 50}
0.001538 (0.004615) with: {batch_size: 10, 'epochs': 100}
0.001538 (0.004615) with: {batch_size: 10, 'epochs': 200}
0.001538 (0.004615) with: {batch_size: 20, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 20, 'epochs': 50}
0.001538 (0.004615) with: {batch_size: 20, 'epochs': 100}
0.001538 (0.004615) with: {batch_size: 50, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 50, 'epochs': 50}
0.001538 (0.004615) with: {batch_size: 100, 'epochs': 100}
0.001538 (0.004615) with: {batch_size: 100, 'epochs': 200}
0.001538 (0.004615) with: {batch_size: 100, 'epochs': 300}

In [237]:
# результаты
print("Best: %s using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, std, param in zip(means, stds, params):
    print("%s with: %s (%s, %s)" % (mean, std, param))

Best: 0.001538 using {batch_size: 4, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 50}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 100}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 200}
0.001538 (0.004615) with: {batch_size: 4, 'epochs': 300}
0.001538 (0.004615) with: {batch_size: 10, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 10, 'epochs': 50}
0.001538 (0.004615) with: {batch_size: 10, 'epochs': 100}
0.001538 (0.004615) with: {batch_size: 10, 'epochs': 200}
0.001538 (0.004615) with: {batch_size: 20, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 20, 'epochs': 50}
0.001538 (0.004615) with: {batch_size: 20, 'epochs': 100}
0.001538 (0.004615) with: {batch_size: 50, 'epochs': 10}
0.001538 (0.004615) with: {batch_size: 50, 'epochs': 50}
0.001538 (0.004615) with: {batch_size: 100, 'epochs': 100}
0.001538 (0.004615) with: {batch_size: 100, 'epochs': 200}
0.001538 (0.004615) with: {batch_size: 100, 'epochs': 300}

In [238]:
model = KerasClassifier(build_fn=create_model, epochs=50, batch_size=4, verbose=0)

optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Nadam']
param_grid = dict(optimizer=optimizer)
grid_result = grid.fit(x_train, y_train)

```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits:  
[CV] END ..... .opt=SGD; total time= 12.5s  
[CV] END ..... .opt=SGD; total time= 11.8s  
[CV] END ..... .opt=SGD; total time= 11.5s  
[CV] END ..... .opt=SGD; total time= 11.5s  
[CV] END ..... .opt=SGD; total time= 12.2s  
[CV] END ..... .opt=SGD; total time= 13.0s  
[CV] END ..... .opt=SGD; total time= 12.8s  
[CV] END ..... .opt=SGD; total time= 12.7s  
[CV] END ..... .opt=SGD; total time= 13s  
[CV] END ..... .opt=SGD; total time= 12.9s  
[CV] END ..... .opt=SGDprop; total time= 12.2s  
[CV] END ..... .opt=SGDprop; total time= 12.2s  
[CV] END ..... .opt=SGDprop; total time= 12s  
[CV] END ..... .opt=SGDprop; total time= 11.9s  
[CV] END ..... .opt=SGDprop; total time= 12.3s  
[CV] END ..... .opt=SGDprop; total time= 13.6s  
[CV] END ..... .opt=SGDprop; total time= 13.2s  
[CV] END ..... .opt=SGDprop; total time= 13.6s  
[CV] END ..... .opt=SGDprop; total time= 13.4s  
[CV] END ..... .opt=SGDprop; total time= 13.7s  
[CV] END ..... .opt=Adagrad; total time= 15.7s  
[CV] END ..... .opt=Adagrad; total time= 15.5s  
[CV] END ..... .opt=Adagrad; total time= 11.3s  
[CV] END ..... .opt=Adagrad; total time= 12.0s  
[CV] END ..... .opt=Adagrad; total time= 11.8s  
[CV] END ..... .opt=Adagrad; total time= 12.0s  
[CV] END ..... .opt=Adagrad; total time= 13.8s  
[CV] END ..... .opt=Adagrad; total time= 12.7s  
[CV] END ..... .opt=Adagrad; total time= 12.8s  
[CV] END ..... .opt=Adagrad; total time= 13.0s  
[CV] END ..... .opt=Adagrad; total time= 11.5s  
[CV] END ..... .opt=Adadelta; total time= 11.9s  
[CV] END ..... .opt=Adadelta; total time= 11.6s  
[CV] END ..... .opt=Adadelta; total time= 12.3s  
[CV] END ..... .opt=Adadelta; total time= 13.9s  
[CV] END ..... .opt=Adadelta; total time= 13.1s  
[CV] END ..... .opt=Adadelta; total time= 13.0s  
[CV] END ..... .opt=Adadelta; total time= 13.0s  
[CV] END ..... .opt=Adadelta; total time= 13.8s  
[CV] END ..... .opt=Adadelta; total time= 13.4s  
[CV] END ..... .opt=Adadelta; total time= 13.2s  
[CV] END ..... .opt=Adam; total time= 12.2s  
[CV] END ..... .opt=Adam; total time= 12.4s  
[CV] END ..... .opt=Adam; total time= 11.8s  
[CV] END ..... .opt=Adam; total time= 12.5s  
[CV] END ..... .opt=Adam; total time= 12.3s  
[CV] END ..... .opt=Adam; total time= 13.5s  
[CV] END ..... .opt=Adam; total time= 13.5s  
[CV] END ..... .opt=Adam; total time= 13.6s  
[CV] END ..... .opt=Adam; total time= 13.6s  
[CV] END ..... .opt=Adam; total time= 13.3s  
[CV] END ..... .opt=Adam; total time= 13.6s  
[CV] END ..... .opt=Adam; total time= 12.8s  
[CV] END ..... .opt=Adam; total time= 12.7s  
[CV] END ..... .opt=Adam; total time= 12.5s  
[CV] END ..... .opt=Adam; total time= 12.2s  
[CV] END ..... .opt=Adam; total time= 12.5s  
[CV] END ..... .opt=Adam; total time= 12.5s  
[CV] END ..... .opt=Adam; total time= 14.9s  
[CV] END ..... .opt=Adam; total time= 14.4s  
[CV] END ..... .opt=Adam; total time= 14.0s  
[CV] END ..... .opt=Nadam; total time= 14.3s
```

```
In [239]: # per seymour  
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))  
means = grid_result.cv_results_['mean_test_score']  
stds = grid_result.cv_results_['std_test_score']  
params = grid_result.cv_results_['params']  
for mean, stdev, param in zip(means, stds, params):  
    print("Mean (%f) with: %s" % (mean, stdev, param))
```

```
Best: 0.003104 using 'opt': 'Adagrad'  
0.001538 (0.004631) with: {'opt': 'SGD'}  
0.001538 (0.004631) with: {'opt': 'SGDprop'}  
0.003101 (0.00602) with: {'opt': 'Adagrad'}  
0.003101 (0.00602) with: {'opt': 'Adadelta'}  
0.000000 (0.000000) with: {'opt': 'Adam'}  
0.003077 (0.00931) with: {'opt': 'Nadam'}
```

```
In [240]: model = KerasClassifier(build_fn=create_model, epochs=50, batch_size=4, verbose=0)  
layers = [[8],[16, 4],[32, 8, 3],[12, 6, 3], [64, 64, 3], [128, 64, 16, 3]]  
param_grid = dict(layers=layers)  
grid_result = GridSearchCV(estimator=model, param_grid=param_grid, cv=10, verbose=2)
```

```
# per seymour  
grid_result = grid.fit(x_train, y_train)  
Fitting 10 folds for each of 6 candidates, totalling 60 fits:  
[CV] END ..... .lyrs=[8]; total time= 11.1s  
[CV] END ..... .lyrs=[8]; total time= 11.8s  
[CV] END ..... .lyrs=[8]; total time= 11.4s  
[CV] END ..... .lyrs=[8]; total time= 10.9s  
[CV] END ..... .lyrs=[8]; total time= 11.7s  
[CV] END ..... .lyrs=[8]; total time= 13.8s  
[CV] END ..... .lyrs=[8]; total time= 13.3s  
[CV] END ..... .lyrs=[8]; total time= 13.7s  
[CV] END ..... .lyrs=[8]; total time= 12.5s  
[CV] END ..... .lyrs=[8]; total time= 14.1s  
[CV] END ..... .lyrs=[8]; total time= 13.9s  
[CV] END ..... .lyrs=[16, 4]; total time= 12.2s  
[CV] END ..... .lyrs=[16, 4]; total time= 11.8s  
[CV] END ..... .lyrs=[16, 4]; total time= 11.4s  
[CV] END ..... .lyrs=[16, 4]; total time= 12.2s  
[CV] END ..... .lyrs=[16, 4]; total time= 13.7s  
[CV] END ..... .lyrs=[16, 4]; total time= 13.5s  
[CV] END ..... .lyrs=[16, 4]; total time= 13.6s  
[CV] END ..... .lyrs=[16, 4]; total time= 13.2s  
[CV] END ..... .lyrs=[16, 4]; total time= 12.5s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 12.6s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 12.4s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 13s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 12.4s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 14.3s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 13.9s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 14.6s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 14.2s  
[CV] END ..... .lyrs=[32, 8, 3]; total time= 14.8s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 11.9s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 12.5s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 13s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 13.6s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 12.6s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 13.7s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 13.6s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 14.2s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 14.0s  
[CV] END ..... .lyrs=[12, 6, 3]; total time= 13.2s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 12.4s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 12.9s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 12.9s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 12.5s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 14.6s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 15.4s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 15.3s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 15.5s  
[CV] END ..... .lyrs=[64, 64, 3]; total time= 15.7s  
[CV] END ..... .lyrs=[128, 64, 16, 3]; total time= 15.5s
```

```
In [241]: # per seymour  
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))  
means = grid_result.cv_results_['mean_test_score']  
stds = grid_result.cv_results_['std_test_score']  
params = grid_result.cv_results_['params']  
for mean, stdev, param in zip(means, stds, params):  
    print("Mean (%f) with: %s" % (mean, stdev, param))
```

```
Best: 0.004639 using 'lyrs': [128, 64, 16, 3]
```

```
0.001538 (0.004631) with: {'lyrs': [8]}
```

```
0.001538 (0.004631) with: {'lyrs': [16, 4]}
```

```
0.001538 (0.004631) with: {'lyrs': [32, 8, 3]}
```

```
0.001538 (0.004631) with: {'lyrs': [12, 6, 3]}
```

```
0.001538 (0.004631) with: {'lyrs': [64, 64, 3]}
```

```
0.004639 (0.009877) with: {'lyrs': [128, 64, 16, 3]}
```

```
In [242]: model = KerasClassifier(build_fn=create_model, epochs=50, batch_size=4, verbose=0)  
activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear']  
param_grid = dict(act=activation)  
grid_result = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=10)
```

```
# per seymour  
grid_result = grid.fit(x_train, y_train)
```

```
In [243]: # per seymour  
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))  
means = grid_result.cv_results_['mean_test_score']  
stds = grid_result.cv_results_['std_test_score']  
params = grid_result.cv_results_['params']  
for mean, stdev, param in zip(means, stds, params):  
    print("Mean (%f) with: %s" % (mean, stdev, param))
```

```
Best: 0.001538 using 'act': 'softmax'
```

```
0.001538 (0.004631) with: {'act': 'softmax'}
```

```
0.001538 (0.004631) with: {'act': 'softplus'}
```

```
0.001538 (0.004631) with: {'act': 'softsign'}
```

```
0.001538 (0.004631) with: {'act': 'relu'}
```

```
0.001538 (0.004631) with: {'act': 'tanh'}
```

```
0.001538 (0.004631) with: {'act': 'sigmoid'}
```

```
0.001538 (0.004631) with: {'act': 'hard_sigmoid'}
```

```
0.001538 (0.004631) with: {'act': 'linear'}
```

```
drops = [0.0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5]
param_grid = dict(dr=drops)

grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=10, verbose=2)
grid_result = grid.fit(x_train, y_train)

Fitting 10 folds for each of 7 candidates, totalling 70 fits
[CV] dr=0.0: total time= 11.6s
[CV] dr=0.01: total time= 12.2s
[CV] dr=0.05: total time= 11.6s
[CV] dr=0.1: total time= 11.3s
[CV] dr=0.2: total time= 11.1s
[CV] dr=0.3: total time= 12.9s
[CV] dr=0.5: total time= 12.7s
[CV] dr=0.0: total time= 12.3s
[CV] dr=0.01: total time= 12.7s
[CV] dr=0.05: total time= 12.9s
[CV] dr=0.1: total time= 16.7s
[CV] dr=0.2: total time= 21.9s
[CV] dr=0.3: total time= 14.8s
[CV] dr=0.5: total time= 15.4s
[CV] dr=0.0: total time= 17.8s
[CV] dr=0.01: total time= 19.5s
[CV] dr=0.05: total time= 19.5s
[CV] dr=0.1: total time= 19.5s
[CV] dr=0.2: total time= 18.8s
[CV] dr=0.3: total time= 18.5s
[CV] dr=0.5: total time= 18.4s
[CV] dr=0.0: total time= 17.1s
[CV] dr=0.01: total time= 18.2s
[CV] dr=0.05: total time= 19.6s
[CV] dr=0.1: total time= 15.7s
[CV] dr=0.2: total time= 13.5s
[CV] dr=0.3: total time= 16.8s
[CV] dr=0.5: total time= 16.6s
[CV] dr=0.0: total time= 18.8s
[CV] dr=0.01: total time= 18.5s
[CV] dr=0.05: total time= 18.5s
[CV] dr=0.1: total time= 18.4s
[CV] dr=0.2: total time= 18.3s
[CV] dr=0.3: total time= 18.3s
[CV] dr=0.5: total time= 18.2s
[CV] dr=0.0: total time= 14.9s
[CV] dr=0.01: total time= 15.3s
[CV] dr=0.05: total time= 15.4s
[CV] dr=0.1: total time= 17.8s
[CV] dr=0.2: total time= 18.8s
[CV] dr=0.3: total time= 16.7s
[CV] dr=0.5: total time= 15.6s
[CV] dr=0.0: total time= 18.3s
[CV] dr=0.01: total time= 15.5s
[CV] dr=0.05: total time= 15.5s
[CV] dr=0.1: total time= 15.3s
[CV] dr=0.2: total time= 15.1s
[CV] dr=0.3: total time= 14.7s
[CV] dr=0.5: total time= 21.8s
[CV] dr=0.0: total time= 18.0s
[CV] dr=0.01: total time= 13.9s
[CV] dr=0.05: total time= 17.6s
[CV] dr=0.1: total time= 15.9s
[CV] dr=0.2: total time= 16.6s
[CV] dr=0.3: total time= 15.9s
[CV] dr=0.5: total time= 17.8s
[CV] dr=0.0: total time= 15.4s
[CV] dr=0.01: total time= 12.4s
[CV] dr=0.05: total time= 14.5s
[CV] dr=0.1: total time= 14.5s
[CV] dr=0.2: total time= 14.7s
[CV] dr=0.3: total time= 15.7s
[CV] dr=0.5: total time= 15.8s
[CV] dr=0.0: total time= 16.8s
[CV] dr=0.01: total time= 16.8s
[CV] dr=0.05: total time= 16.8s
[CV] dr=0.1: total time= 17.3s

In [ ]: # построение
print("Best: %s using Xs" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, std, param in zip(means, stds, params):
    print("%f (%f) with: %s" % (mean, std, param))

In [ ]: # построение окончательного модели
model = create_model(lyrs=[128, 64, 16, 3], dr=0.05)
print(model.summary())

In [ ]: # обучение нейросети, 80/20 CV
model_hist = model.fit(x_train,
y_train,
batch_size = 100,
verbose = 1,
validation_split = 0.2)

In [ ]: # оценка модели
scores = model.evaluate(x_test, y_test)
print("\n\nss: %.2f%% (%d)" % (model.metrics_names[1], scores[1]*100))

In [ ]: # Посмотрим на историю модели
model_hist.history

In [ ]: # Посмотрим на график потеря на тренировочной и тестовой выборках
def model_loss_plot(model_hist):
    plt.figure(figsize = (17,5))
    plt.plot(model_hist.history['loss'],
            label = 'Ошибка на обучающей выборке')
    plt.plot(model_hist.history['val_loss'],
            label = 'Ошибка на тестовой выборке')
    plt.title('График потери модели')
    plt.xlabel('Эпохи')
    plt.ylabel('Значение ошибки')
    plt.legend(loc='best')
    plt.show()
model_loss_plot(model_hist)

In [ ]: # Задача функция для визуализации фок/прогноз для результатов моделей
# Посмотрим на график результатов работы модели
def actual_and_predicted_plot(orig, predict, var, model_name):
    plt.figure(figsize=(17,5))
    title(f'Тестовые и прогнозные значения: ({model_name})')
    plot(orig, label = 'Тест')
    plot(predict, label = 'Прогноз')
    plot(var, label = 'Количество наблюдений')
    plt.legend(loc = 'best')
    plt.show()
actual_and_predicted_plot(y_test.values, model.predict(x_test.values), 'Соотношение матрица/наполнитель', 'Keras_neuronet')

In [ ]: # Сконфигурируем другую модель, зададим слои
model1 = tf.keras.Sequential([
    layers.Dense(128, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1)
])
model1.compile(optimizer = tf.keras.optimizers.Adam(0.001), loss = 'mean_squared_error', metrics = [tf.keras.metrics.RootMeanSquaredError()])
# Посмотрим на архитектуру модели
model1.summary()

In [ ]: # Обучим модель
model_hist1 = model1.fit(
x_train,
y_train,
epochs = 100,
verbose = 1,
validation_split = 0.2)

In [ ]: # оценка модели
y_pred_model = model1.predict(x_test_1)

print("Model Results:")
print(Model.MAE: , round(mean_absolute_error(y_test_1, y_pred_model)))
print(Model.MAPE: {:.2f},format(mean_absolute_percentage_error(y_test_1, y_pred_model)))
print("Test score: {:.2f}.format(mean_squared_error(y_test_1, y_pred_model)))

In [ ]: # Посмотрим на историю модели
model_hist1.history

In [ ]: # Посмотрим на график потеря на тренировочной и тестовой выборках
def model_loss_plot(model_hist1):
    plt.figure(figsize = (17,5))
    plt.plot(model_hist1.history['loss'],
            label = 'Ошибка на обучающей выборке')
    plt.plot(model_hist1.history['val_loss'],
            label = 'Ошибка на тестовой выборке')
    plt.title('График потери модели')
    plt.xlabel('Эпохи')
    plt.ylabel('Значение ошибки')
    plt.legend(loc='best')
    plt.show()
model_loss_plot(model_hist1)

In [ ]: # Задача функция для визуализации фок/прогноз для результатов моделей
# Посмотрим на график результатов работы модели
def actual_and_predicted_plot(orig, predict, var, model_name):
    plt.figure(figsize=(17,5))
    title(f'Тестовые и прогнозные значения: ({model_name})')
    plot(orig, label = 'Тест')
    plot(predict, label = 'Прогноз')
    plot(var, label = 'Количество наблюдений')
    plt.legend(loc = 'best')
    plt.show()
actual_and_predicted_plot(y_test_1, model1.predict(x_test_1), 'Соотношение матрица/наполнитель', 'Keras_neuronet')
```

```
plt.show()
actual_and_predicted_plot(y_test.values, model1.predict(x_test.values), 'Соотношение матрица/наполнитель', 'Keras_neuronet')

In [1]: # оценка модели MSE
model1.evaluate(x_test, y_test, verbose = 1)

In [2]: test_predictions = model1.predict(x_test).flatten()

a = plt.axes(aspect = 'equal')
plt.scatter(y_test, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
lims = [0, 5]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)

In [3]: # Сохранение моделя для разработки веб-приложения для прогнозирования соотношения "матрица-наполнитель" в фреймворке Flask
model1.save('App/s.model/k_m')

In [4]: """ Заключение
В данной работе, стоит сказать, что машинное обучение в задачах моделирования – довольно сложный процесс, требующий не только навыков программирования, но и профессионального подхода к сфере самих композитных материалов.
Необходимо понимать, на какие атрибуты нужно в первую очередь обратить внимание, чтобы сунуть впоследствии грамотно и четко спрогнозировать тот или иной признак. И, естественно, обладать всеми необходимыми знаниями, умениями и навыками для прогнозов и расчетов.
В ходе работы был задействован дата-сет с реальными данными, произведена его подробная опись и сопутствующий анализ; построено множество разнообразных графиков; осуществлено разбиение данных на обучавшую и тестовую выборки с использованием множества вспомогательных модулей из библиотеки SkLearn, которая во многом облегчила процесс машинного обучения и в целом была очень полезным инструментом в ходе работы над выпускной квалификационной работой.
В рамках машинного обучения и поиска гиперпараметров были задействованы несколько алгоритмов: линейная регрессия, градиентный бустинг, К ближайших соседей, деревья решений, стохастический градиентный спуск, многослойный перцептрон, лasso регрессия, а также опорные вектора и случайный лес.
Поиск гиперпараметров осуществлялся при помощи таких методов, как «GridSearch». Для каждой из выборок были составлены классификационные отчеты, содержащие в себе основополагающие метрики, оценивающие качество проводимого обучения.
В ходе работы было получено следующее значение коэффициента детерминации R^2 = 0.85, что свидетельствует о высокой точности предсказаний.
Обучена нейронная сеть и разработано пользовательское приложение, предсказывающее вероятный прогноз по заданным параметрам.
Что касается перспектив решения данной проблемы композитных материалов, то я думаю, что в таких случаях необходимо уделять больше внимания изучению самой проблемы композитных материалов, углубить знания по статистике и регрессиям, поискать иные варианты решений с данным датасетом, создать плодотворную команду программистов и сотрудников, работающих с природными материалами, способную к совместной работе над усовершенствованием уже существующих разработок и поддержанием их качественного и бесперебойного функционирования.""""

In [5]:
```