



Выпускная квалификационная работа Кирпичева Артема Игоревича

слушателя курса "Data Science Pro" Образовательного центра Московского государственного технического университета им. Н.Э. Баумана

Тема исследования: Прогнозирование конечных свойств новых материалов (композиционных материалов). Цель исследования: построение моделей прогнозирования следующих параметров: «модуль упругости при растяжении»; «прочность при растяжении»; «соотношение матрица-наполнитель» Итог работы: Разработка приложения с графическим интерфейсом, которое будет выдавать прогноз параметра «соотношение матрица-наполнитель».

Приложение 1 Подробный план работы: 1. Загружаем и обрабатываем входящие датасеты 1.1. Удаляем неинформативные столбцы 1.2. Объединяем датасеты по методу INNER 2. Проводим разведочный анализ данных: 2.1. Данные в столбце "Угол нашивки" приведём к 0 и 1 2.2. Изучим описательную статистику каждой переменной - среднее, медиана, стандартное отклонение, минимум, максимум, квартили 2.3. Проверим датасет на пропуски и дубликаты данных 2.4. Получим среднее, медианное значение для каждой колонки (по заданию необходимо получить их отдельно, поэтому продублируем их только отдельно) 2.5. Вычислим коэффициенты ранговой корреляции Кендалла 2.6. Вычислим коэффициенты корреляции Пирсона 3. Визуализируем наш разведочный анализ сырых данных (до выбросов и нормализации) 3.1. Построим несколько вариантов гистограмм распределения каждой переменной 3.2. Построим несколько вариантов диаграмм ящиков с усами каждой переменной 3.3. Построим гистограмму распределения и диаграмма "ящик с усами" одновременно вместе с данными по каждому столбцу 3.4. Построим несколько вариантов попарных графиков рассеяния точек (матрицы диаграмм рассеяния) 3.5. Построим графики квантиль-квантиль 3.6. Построим корреляционную матрицу с помощью тепловой карты 4. Проведём предобработку данных (в данном пункте только очистка датасета от выбросов) 4.1. Проверим выбросы по 2 методам: 3-х сигм или межквартильных расстояний 4.2. Посчитаем распределение выбросов по каждому столбцу (с целью предотвращения удаления особенностей признака или допущения ошибки) 4.3. Исключим выбросы методом межквартильного расстояния 4.4. Удалим строки с выбросами 4.5. Визуализируем датасет без выбросов, и убедимся, что выбросы еще есть. 4.6. Для полной очистки датасета от выбросов повторим пункты (4.3 – 4.5) ещё 3 раза. 4.7. Сохраняем идеальный, без выбросов датасет 4.8. Изучим чистые данные по всем параметрам 4.9. Визуализируем «чистый» датасет (без выбросов) 5. Проведём нормализацию и стандартизацию (продолжим предобработку данных) 5.1. Визуализируем плотность ядра 5.2. Нормализуем данные с помощью MinMaxScaler() 5.3. Нормализуем данные с помощью Normalizer() 5.4. Сравним с данными до нормализации 5.5. Проверим перевод данных из нормализованных в исходные 5.6. Рассмотрим несколько вариантов корреляции между параметрами после нормализации 5.7. Стандартизуем данные 5.8. Визуализируем данные корреляции 5.9. Посмотрим на описательную статистику после нормализации и после стандартизации 6. Разработаем и обучим нескольких моделей прогноза прочности при растяжении (с 30% тестовой выборки) 6.1. Определим входы и выходы для моделей 6.2. Разобьём данные на обучающую и тестовую выборки 6.3. Проверим правильность разбивки 6.4. Построим модели и найдём лучшие гиперпараметры (задача по заданию): 6.5. Построим и визуализируем результат работы метода опорных векторов 6.6. Построим и визуализируем результат работы метода случайного леса 6.7. Построим и визуализируем результат работы линейной регрессии 6.8. Построим и визуализируем результат работы метода градиентного бустинга 6.9. Построим и визуализируем результат работы метода К ближайших соседей 6.10. Построим и визуализируем результат работы метода деревья решений 6.11. Построим и визуализируем результат работы стохастического градиентного спуска 6.12. Построим и визуализируем результат работы многослойного перцептрона 6.13. Построим и визуализируем результат работы лasso регрессии 6.14. Сравним наши модели по метрике MAE 6.15. Найдём лучшие гиперпараметры для случайного леса 6.16. Подставим значения в нашу модель случайного

леса 6.17. Найдём лучшие гиперпараметры для К ближайших соседей 6.18. Подставим значения в нашу модель К ближайших соседей 6.19. Найдём лучшие гиперпараметры метода дерева решений 6.20. Подставим значения в нашу модель метода дерева решений 6.21. Проверим все модели и процессинги и выведем лучшую модель и процессинг 7. Разработаем и обучим нескольких моделей прогноза модуля упругости при растяжении (с 30% тестовой выборки) 7.1. Определим входы и выходы для моделей 7.2. Разобьём данные на обучающую и тестовую выборки 7.3. Проверим правильность разбивки 7.4. Построим модели и найдём лучшие гиперпараметры (задача по заданию): 7.5. Построим и визуализируем результат работы метода опорных векторов 7.6. Построим и визуализируем результат работы метода случайного леса 7.7. Построим и визуализируем результат работы линейной регрессии 7.8. Построим и визуализируем результат работы метода градиентного бустинга 7.9. Построим и визуализируем результат работы метода К ближайших соседей 7.10. Построим и визуализируем результат работы метода дерева решений 7.11. Построим и визуализируем результат работы стохастического градиентного спуска 7.12. Построим и визуализируем результат работы многослойного перцептрона 7.13. Построим и визуализируем результат работы лasso регрессии 7.14. Сравним наши модели по метрике МАЕ 7.15. Найдём лучшие гиперпараметры для случайного леса 7.16. Подставим значения в нашу модель случайного леса 7.17. Найдём лучшие гиперпараметры для К ближайших соседей 7.18. Подставим значения в нашу модель К ближайших соседей 7.19. Найдём лучшие гиперпараметры метода дерева решений 7.20. Подставим значения в нашу модель метода дерева решений 7.21. Проверим все модели и процессинги и выведем лучшую модель и процессинг 8. Нейронная сеть для рекомендации соотношения матрица-наполнитель 8.1. Сформируем входы и выход для модели 8.2. Нормализуем данные 8.3. Построим модель, определим параметры 8.4. Найдем оптимальные параметры для модели 8.5. Посмотрим на результаты 8.6. Повторим шаги 8.4 – 8.5 до построения окончательной модели 8.7. Обучим нейросеть 80/20 8.8. Оценим модель 8.9. Посмотрим на потери модели 8.10. Посмотрим на график результата работы модели 8.11. Посмотрим на график потерь на тренировочной и тестовой выборках 8.12. Сконфигурируем другую модель, зададим слои 8.13. Посмотрим на архитектуру другой модели 8.14. Обучим другую модель 8.15. Посмотрим на потери другой модели 8.16. Посмотрим на график потерь на тренировочной и тестовой выборках 8.17. Зададим функцию для визуализации факт/прогноз для результатов моделей 8.18. Посмотрим на график результата работы модели 8.19. Оценим модель MSE 8.20. Сохраняем вторую модель для разработки веб-приложения для прогнозирования соотношения "матрица-наполнитель" в фреймворке Flask 9. Создаём приложение 9.1. Импортируем необходимые библиотеки 9.2. Загрузим модель и определим параметры функции 9.3. Получим данные из наших форм и положим их в список 9.4. Укажем шаблон и прототип сайта для вывода 9.5. Запустим приложение 9.6. Откроем <http://127.0.0.1:5000/> 10. Создание удалённого репозитория и загрузка результатов работы на него. 10.1. <https://github.com/artem270Z/VKR> 10.2. Создадим README (<https://github.com/artem270Z/VKR#readme>) 10.3. Выгрузим все необходимые файлы и репозиторий

Импортируем сразу все необходимые библиотеки для нашего исследования

```
In [1]: # Базовые библиотеки
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

# Визуализация
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# TensorFlow и Keras
#import tensorflow as tf
#from tensorflow import keras
#from tensorflow.keras import layers
#from tensorflow.keras.layers import (
    #Dense,
    #Flatten,
```

```

# Dropout,
#BatchNormalization,
#Activation
#)
#from tensorflow.keras.models import Sequential

# Импорт Keras wrappers из отдельного модуля
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense

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

# Дополнительные модули
from pandas import read_excel, DataFrame, Series
from scipy import stats

```

Загружаем исходные данные из обеих excel таблиц и удаляем колонку с индексом

In [2]: #Загружаем первый датасет и посмотрим на названия столбцов

```
df_bp = pd.read_excel(r"C:\Users\МАЭСТРО\Desktop\ЗащитаДиплома\Datasets\X_bp.xlsx")
df_bp.shape
df_bp.head()
```

Out[2]:

Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%
0	0	1.857143	2030.0	738.736842	30.00
1	1	1.857143	2030.0	738.736842	50.00
2	2	1.857143	2030.0	738.736842	49.90
3	3	1.857143	2030.0	738.736842	129.00
4	4	2.771331	2030.0	753.000000	111.86

In [3]:

```
#Удаляем первый неинформативный столбец
df_bp.drop(['Unnamed: 0'], axis=1, inplace=True)
#Посмотрим на первые 5 строк первого датасета и убедимся, что первый столбец у
df_bp.head()
```

Out[3]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки
0	1.857143	2030.0	738.736842	30.00	22.267857	100.0
1	1.857143	2030.0	738.736842	50.00	23.750000	284.6
2	1.857143	2030.0	738.736842	49.90	33.000000	284.6
3	1.857143	2030.0	738.736842	129.00	21.250000	300.0
4	2.771331	2030.0	753.000000	111.86	22.267857	284.6

In [4]:

```
# Проверим размерность первого файла
df_bp.shape
```

Out[4]: (1023, 10)

In [5]:

```
# Загружаем второй датасет
df_nup = pd.read_excel(r"C:\Users\МАЭСТРО\Desktop\ЗащитаДиплома\Datasets\X_nup.xlsx")
df_nup.shape
df_nup.head()
```

Out[5]:

	Unnamed: 0	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	0	0	4.0	57.0
1	1	0	4.0	60.0
2	2	0	4.0	70.0
3	3	0	5.0	47.0
4	4	0	5.0	57.0

In [6]:

```
#Удаляем первый неинформативный столбец
df_nup.drop(['Unnamed: 0'], axis=1, inplace=True)
#Посмотрим на первые 5 строк второго датасета и убедимся, что и здесь не нужнъ
df_nup.head()
```

Out[6]:

	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	0	4.0	57.0
1	0	4.0	60.0
2	0	4.0	70.0
3	0	5.0	47.0
4	0	5.0	57.0

In [7]:

```
# Проверим размерность второго файла
df_nup.shape
```

Out[7]: (1040, 3)

Объединим по индексу, тип объединения INNER, смотрим итоговый датасет

In [8]:

```
# два датасета имеют разный объем строк.
# объединяем их по типу INNER.
df = df_bp.merge(df_nup, left_index = True, right_index = True, how = 'inner')
df.head().T
```

Out[8]:

	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 [9]:

```
#Посмотрим количество колонок и столбцов
df.shape
# Итоговый датасет имеет 13 столбцов и 1023 строки, 17 строк из таблицы X_pir
```

Out[9]: (1023, 13)

Проведем разведочный анализ.

Знакомство с данными

```
In [10]: # Посмотрим на начальные и конечные строки нашего датасета на данном этапе работы
df
```

Out[10]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2
0	1.857143	2030.000000	738.736842	30.000000	22.267857	1
1	1.857143	2030.000000	738.736842	50.000000	23.750000	2
2	1.857143	2030.000000	738.736842	49.900000	33.000000	2
3	1.857143	2030.000000	738.736842	129.000000	21.250000	3
4	2.771331	2030.000000	753.000000	111.860000	22.267857	2
...
1018	2.271346	1952.087902	912.855545	86.992183	20.123249	3
1019	3.444022	2050.089171	444.732634	145.981978	19.599769	2
1020	3.280604	1972.372865	416.836524	110.533477	23.957502	2
1021	3.705351	2066.799773	741.475517	141.397963	19.246945	2
1022	3.808020	1890.413468	417.316232	129.183416	27.474763	3

1023 rows × 13 columns

```
In [11]: #Просмотрим информацию о датасете
df.info()
# все переменные содержат значения float64, качественные характеристики отсутствуют
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1023 entries, 0 to 1022
Data columns (total 13 columns):
 #   Column           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   int64  
 11  Шаг нашивки                      1023 non-null   float64
 12  Плотность нашивки                1023 non-null   float64
dtypes: float64(12), int64(1)
memory usage: 111.9 KB
```

```
In [12]: #Поиск уникальных значений с помощью функции unique  
df.unique()  
#Видим в основном общее число уникальных значений в каждом столбце, но в столб
```

```
Out[12]: Соотношение матрица-наполнитель      1014  
Плотность, кг/м3                          1013  
модуль упругости, ГПа                      1020  
Количество отвердителя, м.%                1005  
Содержание эпоксидных групп,%_2          1004  
Температура вспышки, С_2                  1003  
Поверхностная плотность, г/м2            1004  
Модуль упругости при растяжении, ГПа    1004  
Прочность при растяжении, МПа            1004  
Потребление смолы, г/м2                  1003  
Угол нашивки, град                         2  
Шаг нашивки                                989  
Плотность нашивки                         988  
dtype: int64
```

```
In [13]: # Поработаем со столбцом "Угол нашивки"
```

```
In [14]: df['Угол нашивки, град'].unique()  
#Так как кол-во уникальных значений в колонке Угол нашивки равно 2, можем прив
```

```
Out[14]: 2
```

```
In [15]: #Проверим кол-во элементов, где Угол нашивки равен 0 градусов  
df['Угол нашивки, град'][df['Угол нашивки, град'] == 0.0].count()
```

```
Out[15]: np.int64(520)
```

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

```
In [17]: #Переименуем столбец  
df = df.rename(columns={'Угол нашивки, град' : 'Угол нашивки'})  
df
```

Out[17]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура кurenia, С
0	1.857143	2030.000000	738.736842	30.000000	22.267857	1
1	1.857143	2030.000000	738.736842	50.000000	23.750000	2
2	1.857143	2030.000000	738.736842	49.900000	33.000000	2
3	1.857143	2030.000000	738.736842	129.000000	21.250000	3
4	2.771331	2030.000000	753.000000	111.860000	22.267857	2
...
1018	2.271346	1952.087902	912.855545	86.992183	20.123249	3
1019	3.444022	2050.089171	444.732634	145.981978	19.599769	2
1020	3.280604	1972.372865	416.836524	110.533477	23.957502	2
1021	3.705351	2066.799773	741.475517	141.397963	19.246945	2
1022	3.808020	1890.413468	417.316232	129.183416	27.474763	3

1023 rows × 13 columns

In [18]: `#Посчитаем количество элементов, где угол нашивки равен 0 градусов и убедимся,
df['Угол нашивки'][df['Угол нашивки'] == 0.0].count()
#После преобразования колонки Угол нашивки к значениям 0 и 1, кол-во элементов`

Out[18]: np.int64(520)

In [19]: `# Переведем столбец с нумерацией в integer
df.index = df.index.astype('int')`

In [20]: `# Сохраним итоговый датасет в отдельную папку с данными
df.to_excel("Itog\itog.xlsx")`

In [21]: `#Изучим описательную статистику наших данных (максимальное, минимальное, квартильные значения)
df.describe()`

Out[21]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Те
count	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000
mean	2.930366	1975.734888	739.923233	110.570769	22.244390	1
std	0.913222	73.729231	330.231581	28.295911	2.406301	
min	0.389403	1731.764635	2.436909	17.740275	14.254985	
25%	2.317887	1924.155467	500.047452	92.443497	20.608034	
50%	2.906878	1977.621657	739.664328	110.564840	22.230744	
75%	3.552660	2021.374375	961.812526	129.730366	23.961934	
max	5.591742	2207.773481	1911.536477	198.953207	33.000000	

In [22]: `a = df.describe()
a.T`

	count	mean	std	min	25%
Соотношение матрица-наполнитель	1023.0	2.930366	0.913222	0.389403	2.317887 2
Плотность, кг/м3	1023.0	1975.734888	73.729231	1731.764635	1924.155467 1977
модуль упругости, ГПа	1023.0	739.923233	330.231581	2.436909	500.047452 739
Количество отвердителя, м.%	1023.0	110.570769	28.295911	17.740275	92.443497 110
Содержание эпоксидных групп,%_2	1023.0	22.244390	2.406301	14.254985	20.608034 22
Температура вспышки, С_2	1023.0	285.882151	40.943260	100.000000	259.066528 285
Поверхностная плотность, г/м2	1023.0	482.731833	281.314690	0.603740	266.816645 451
Модуль упругости при растяжении, ГПа	1023.0	73.328571	3.118983	64.054061	71.245018 73
Прочность при растяжении, МПа	1023.0	2466.922843	485.628006	1036.856605	2135.850448 2459
Потребление смолы, г/м2	1023.0	218.423144	59.735931	33.803026	179.627520 219
Угол нашивки	1023.0	0.491691	0.500175	0.000000	0.000000 0
Шаг нашивки	1023.0	6.899222	2.563467	0.000000	5.080033 6
Плотность нашивки	1023.0	57.153929	12.350969	0.000000	49.799212 57

Описательная статистика содержит по каждому столбцу (по каждой переменной):

- count - количество значений
- mean - среднее значение
- std - стандартное отклонение
- min - минимум

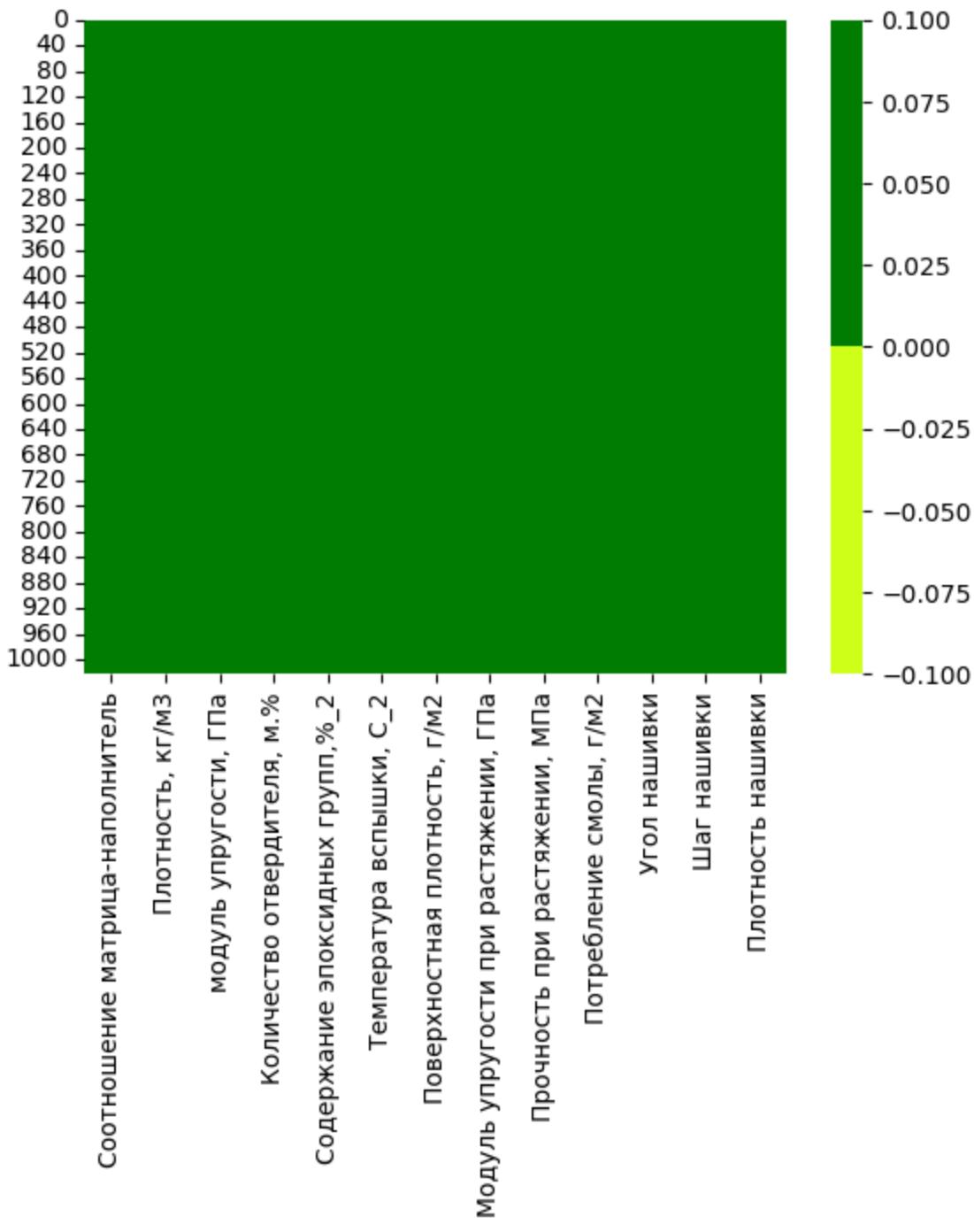
- 25% - верхнее значение первого квартиля
- 50% - медиана
- 75% - верхнее значение третьего квартиля
- max - максимум

```
In [23]: # Проверим на пропущенные данные
df.isnull().sum()
# Пропущенных данных нет = нулевых значений нет, очистка не требуется
```

```
Out[23]: 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 [24]: #светло-зеленый - не пропущенные, темнозеленый - пропущенные данные
cols = df.columns
colours = ['#ceff1d', '#008000']
sns.heatmap(df[cols].isnull(), cmap = sns.color_palette(colours))
#Тепловая карта, так же как info() и функция ISNULL() показывает, что пропуска
```

```
Out[24]: <Axes: >
```



```
In [25]: #анализ пропусков
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    print('{} - {}%'.format(col, round(pct_missing*100)))
```

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%

In [26]: # Проверим датасет на дубликаты
df.duplicated().sum()
#Дубликатов нет

Out[26]: np.int64(0)

In [27]: #получим среднее и медианное значения данных в колонках
mean_and_50 = df.describe()
mean_and_50.loc[['mean', '50%']]
#в целом мы видим близкие друг к другу значения

Out[27]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2
mean	2.930366	1975.734888	739.923233	110.570769	22.244390	285.882151
50%	2.906878	1977.621657	739.664328	110.564840	22.230744	482.731833

In [28]: # среднее значение
df.mean()

Out[28]: Соотношение матрица-наполнитель 2.930366
Плотность, кг/м3 1975.734888
модуль упругости, ГПа 739.923233
Количество отвердителя, м.% 110.570769
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 2466.922843
Потребление смолы, г/м2 218.423144
Угол нашивки 0.491691
Шаг нашивки 6.899222
Плотность нашивки 57.153929
dtype: float64

```
In [29]: # медианное значение  
df.median()
```

```
Out[29]: Соотношение матрица-наполнитель      2.906878  
Плотность, кг/м3                          1977.621657  
модуль упругости, ГПа                      739.664328  
Количество отвердителя, м.%                110.564840  
Содержание эпоксидных групп,%_2           22.230744  
Температура вспышки, С_2                  285.896812  
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                                0.000000  
Шаг нашивки                                 6.916144  
Плотность нашивки                         57.341920  
dtype: float64
```

```
In [30]: # Вычисляем коэффициенты ранговой корреляции Кендалла  
df.corr(method = 'kendall')
```

Out[30]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп
Соотношение матрица-наполнитель	1.000000	-0.003135	0.021247	0.001410	0.01
Плотность, кг/м3	-0.003135	1.000000	-0.008059	-0.021963	-0.00
модуль упругости, ГПа	0.021247	-0.008059	1.000000	0.022382	0.00
Количество отвердителя, м.%	0.001410	-0.021963	0.022382	1.000000	0.00
Содержание эпоксидных групп,%_2	0.010180	-0.007758	0.002351	0.000010	1.00
Температура вспышки, С_2	-0.009480	-0.019947	0.021028	0.059034	-0.00
Поверхностная плотность, г/м2	-0.002060	0.037302	-0.000442	0.033110	-0.00
Модуль упругости при растяжении, ГПа	-0.004157	-0.021151	0.005458	-0.043140	0.04
Прочность при растяжении, МПа	0.011614	-0.047426	0.022959	-0.046507	-0.01
Потребление смолы, г/м2	0.035145	-0.017079	0.005169	-0.003677	0.00
Угол нашивки	-0.021395	-0.051525	-0.031695	0.024690	0.00
Шаг нашивки	0.022723	-0.031220	-0.008305	0.006232	-0.00
Плотность нашивки	0.002788	0.052935	0.049347	0.016607	-0.02

In [31]: #Вычисляем коэффициенты корреляции Пирсона
df.corr(method ='pearson')

Out[31]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп
Соотношение матрица-наполнитель	1.000000	0.003841	0.031700	-0.006445	0.01
Плотность, кг/м3	0.003841	1.000000	-0.009647	-0.035911	-0.00
модуль упругости, ГПа	0.031700	-0.009647	1.000000	0.024049	-0.00
Количество отвердителя, м.%	-0.006445	-0.035911	0.024049	1.000000	-0.00
Содержание эпоксидных групп,%_2	0.019766	-0.008278	-0.006804	-0.000684	1.00
Температура вспышки, С_2	-0.004776	-0.020695	0.031174	0.095193	-0.00
Поверхностная плотность, г/м2	-0.006272	0.044930	-0.005306	0.055198	-0.01
Модуль упругости при растяжении, ГПа	-0.008411	-0.017602	0.023267	-0.065929	0.05
Прочность при растяжении, МПа	0.024148	-0.069981	0.041868	-0.075375	-0.02
Потребление смолы, г/м2	0.072531	-0.015937	0.001840	0.007446	0.01
Угол нашивки	-0.031073	-0.068474	-0.025417	0.038570	0.00
Шаг нашивки	0.036437	-0.061015	-0.009875	0.014887	0.00
Плотность нашивки	-0.004652	0.080304	0.056346	0.017248	-0.03

In [32]: #Создадим переменную для названия всех столбцов. Это нам пригодится при построении графиков

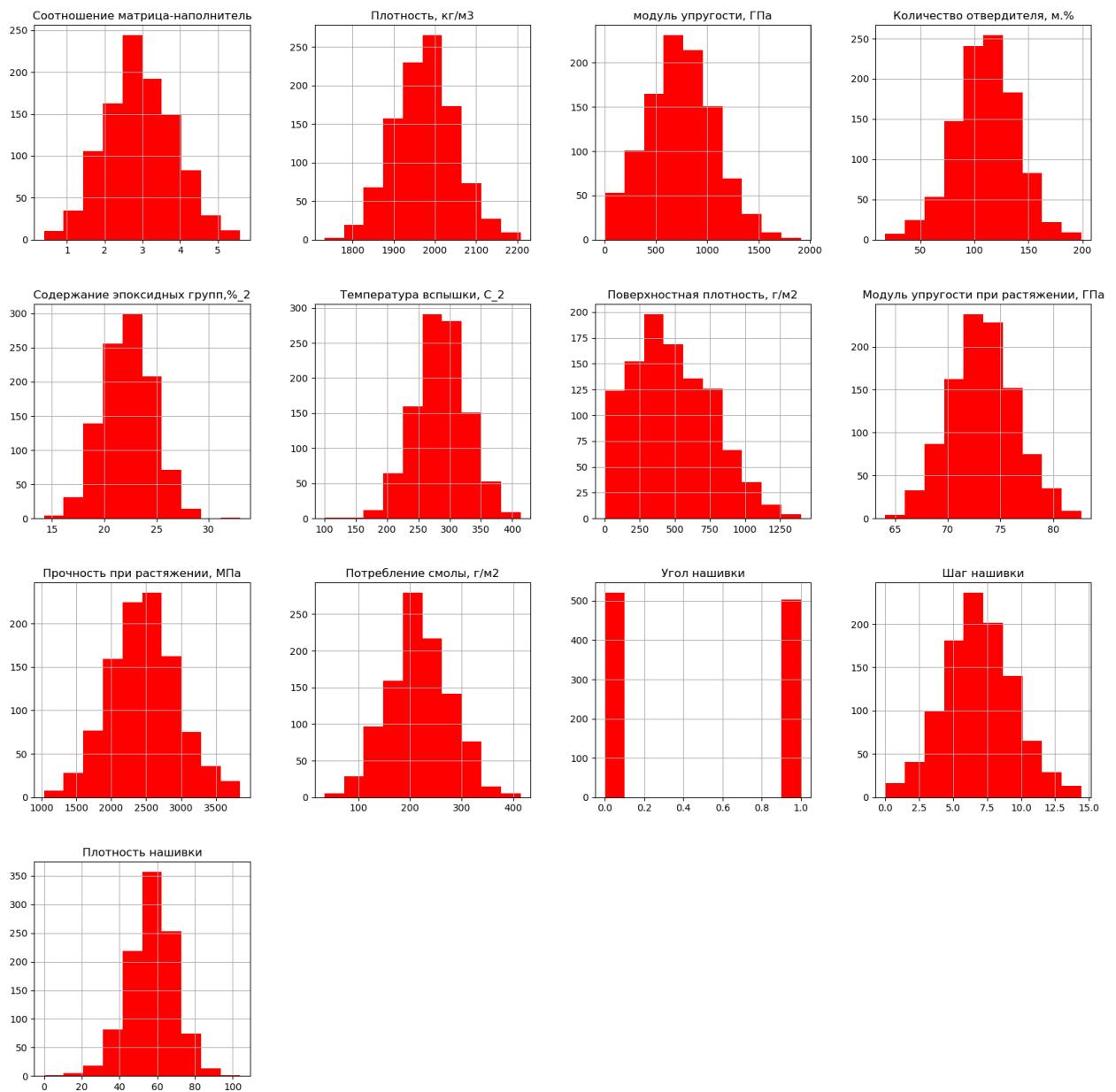
```
#column_names = ["Соотношение матрица-наполнитель", "Плотность, кг/м3", "модуль упругости, ГПа", "Содержание эпоксидных групп,%_2", "Температура вспышки, С_2", "Поверхностная плотность, г/м2", "Модуль упругости при растяжении, ГПа", "Прочность при растяжении, МПа", "Угол нашивки, град", "Шаг нашивки", "Плотность нашивки"]
column_names = df.columns
```

Визуализируем сырые данные и проведем анализ

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

Показатели описательной статистики и визуализация гистограмм и/или диаграмм размаха («ящик с усами») позволяют получить наглядное представление о характеристах распределений переменных. Такое частотное распределение показывает, какие именно конкретные значения или диапазоны значений исследуемой переменной встречаются наиболее часто, насколько различаются эти значения, расположено ли большинство наблюдений около среднего значения, является распределение симметричным или асимметричным, многомодальным (т.е. имеет две или более вершины) или одномодальным и т.д. По форме распределения можно судить о природе исследуемой переменной (например, бимодальное распределение позволяет предположить, что выборка не является однородной и содержит наблюдения, принадлежащие двум различным множествам, которые в свою очередь нормально распределены).

```
In [33]: # Построим гистограммы распределения каждой из переменных без нормализации и их
df.hist(figsize = (20,20), color = "r")
plt.show()
```



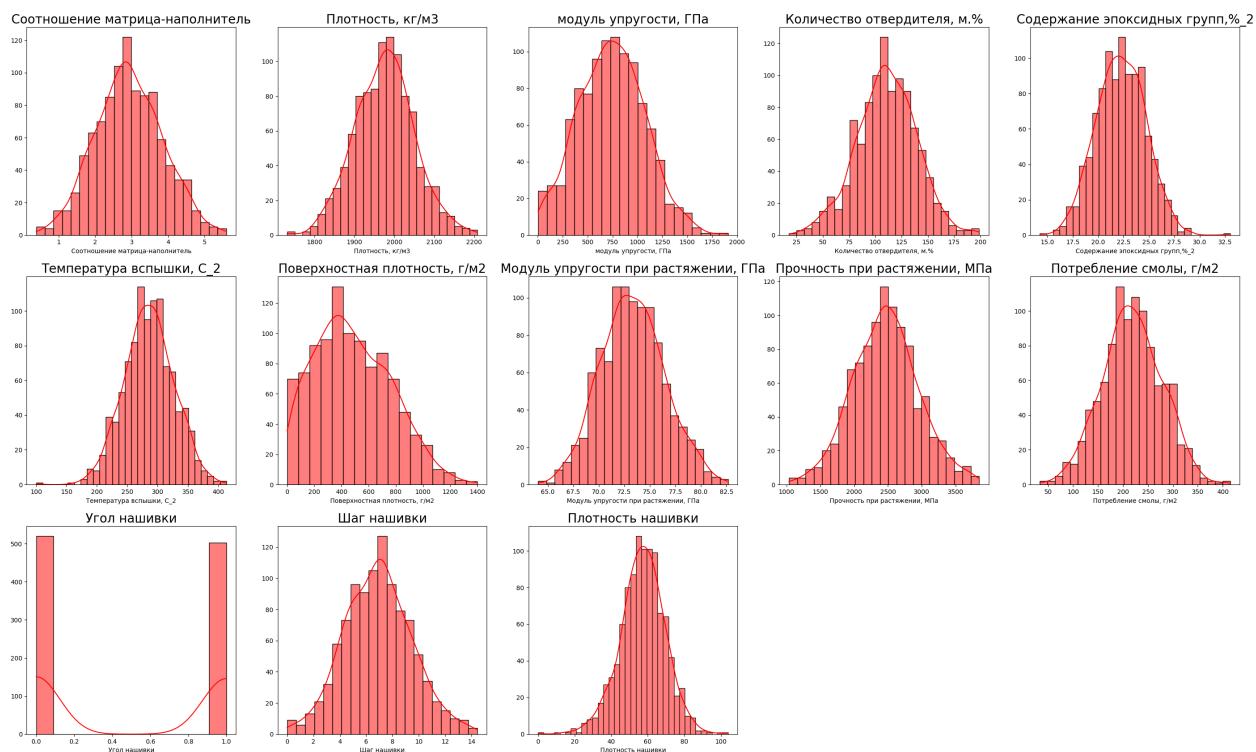
При проведении анализа выявлены параметры близкие к нормальному:
 Соотношение матрица-наполнитель; Плотность, кг/м³; Модуль упругости, ГПа; Количество отвердителя, м.%; Содержание эпоксидных групп, %_2; Температура вспышки, С_2; Поверхностная плотность, г/м²; Модуль упругости при растяжении, ГПа; Прочность при растяжении, МПа; Потребление смолы, г/м²; Шаг нашивки; Плотность нашивки.

Преимущественно данные стремятся кциальному распределению. Угол нашивки, как и отражено в датасете, имеет только два значения 90 градусов и 0 градусов, что отражает общий подход к проведению нашивки материалов, а также может быть использовано при обработке данных. Учитывая отсутствие иных показателей для угла нашивки, предлагаем в прогнозе использовать категориальный, а не непрерывный подход при

анализе данного параметра.

```
In [34]: # Гистограмма распределения (второй вариант)
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # инициализация plot counter
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 = "red")
    plt.ylabel(None)
    plt.title(col, size = 20)
    #plt.show()
    c += 1
#Гистограммы показывают ярковыраженные выбросы в столбцах: плотность, содержание отвердителя, содержание эпоксидных групп
#Данные стремятся к нормальному распределению практически везде, кроме угла нашивки
```

Гистограммы переменных



```
In [35]: # гистограмма распределения и боксплоты (третий вариант)
```

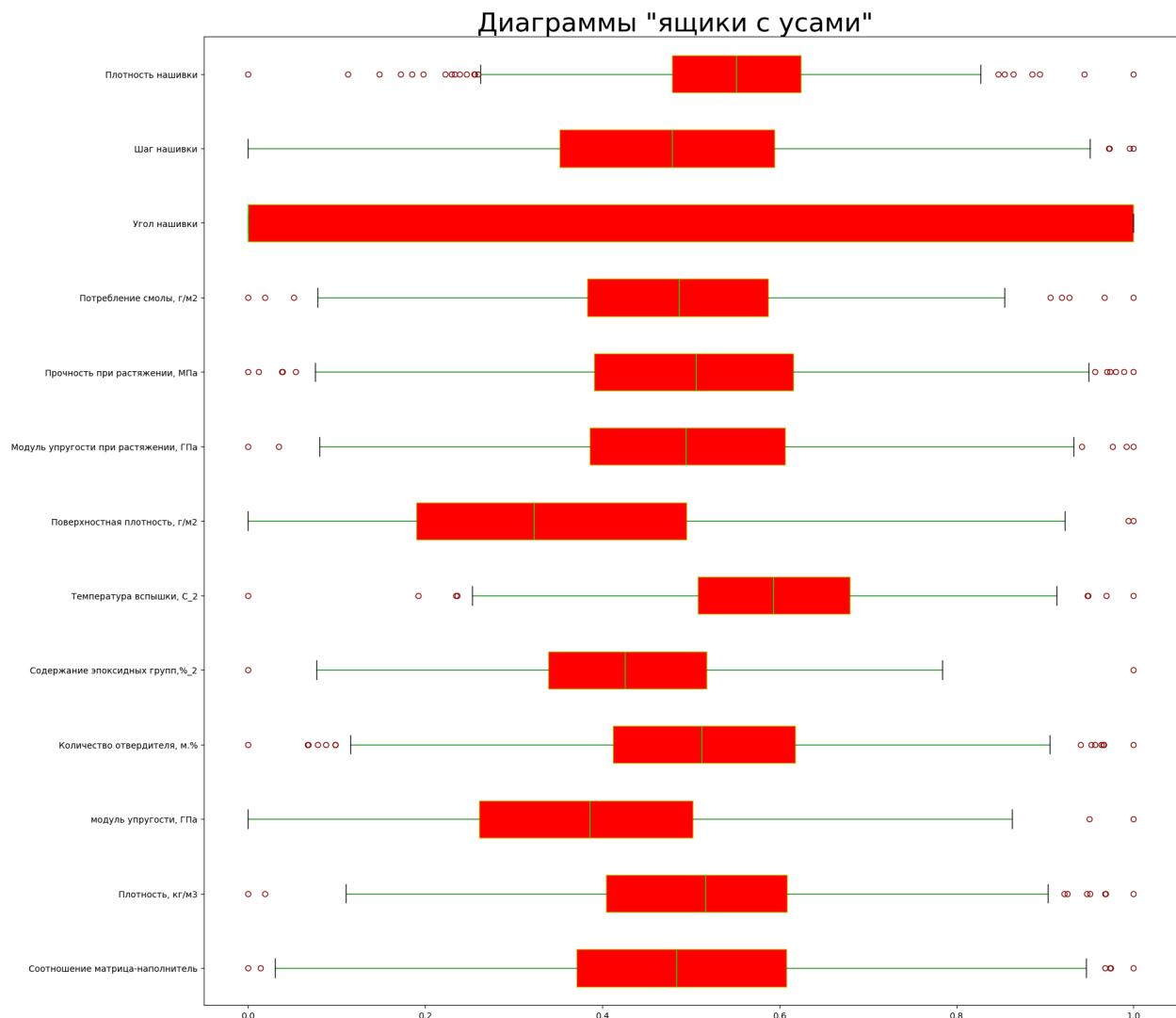
```
for column in df.columns:
    fig = px.histogram(df, x = column, color_discrete_sequence = ['blue'], nbins=50)
    fig.show()
```

```
In [36]: for column in df.columns:
```

```
fig = px.box(df, y = column)
fig.show()
```

In [37]:

```
# "Ящики с усами" (боксплоты) (первый вариант)
from sklearn.preprocessing import MinMaxScaler # Добавляем эту строку
import matplotlib.pyplot as plt
import pandas as pd
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)
plt.show()
```

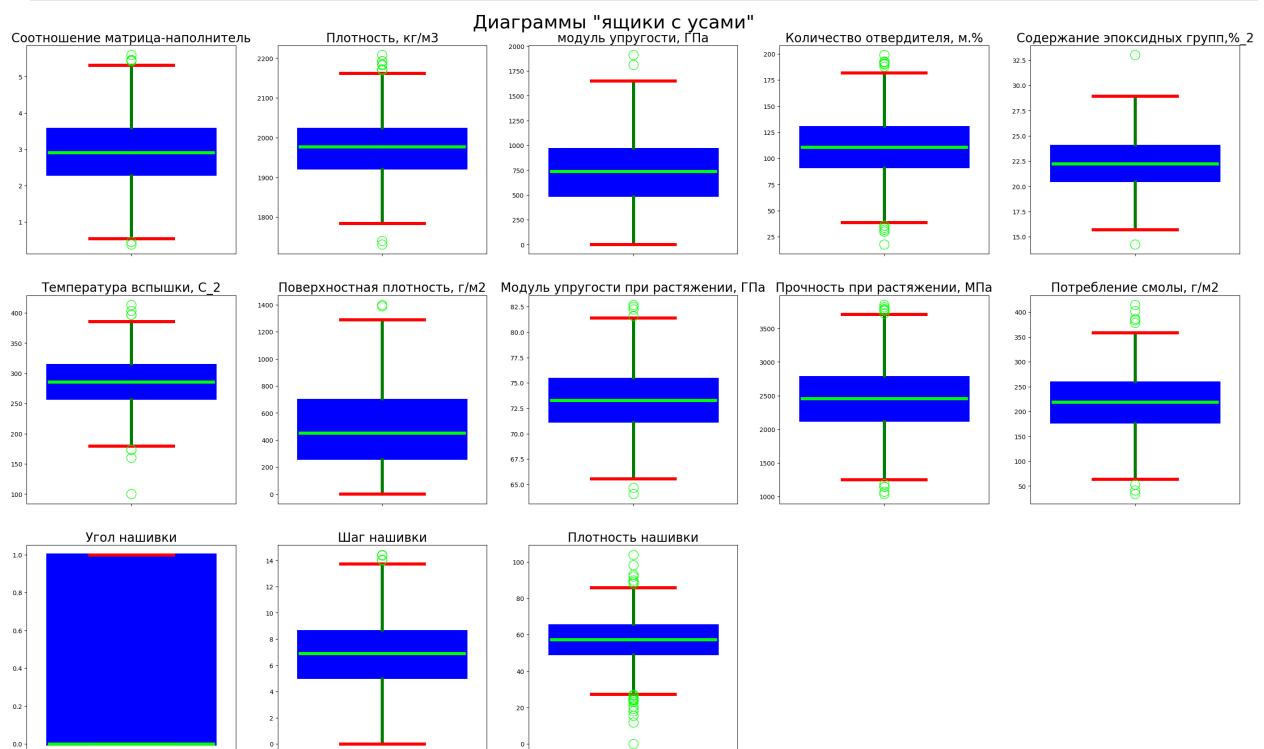


Многие алгоритмы машинного обучения чувствительны к разбросу и распределению значений признаков обрабатываемых объектов. Соответственно, выбросы во входных данных могут исказить и ввести в заблуждение процесс обучения алгоритмов машинного обучения, что приводит к увеличению

времени обучения, снижению точности моделей и, в конечном итоге, к снижению результатов.

```
In [38]: # 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.figure(figsize=(7,5))
    sns.boxplot(data = df, y = df[col], fliersize = 15, linewidth = 5, boxprops = {'color': 'blue'}, whiskerprops = {'color': 'red'}, medianprops = {'color': 'green'}, notch = True)
    plt.ylabel(None)
    plt.title(col, size = 20)
    #plt.show()
    c += 1
# "Ящики с усами" показывают наличие выбросов во всех столбцах, кроме углов на
```



```
In [39]: # Гистограмма распределения и диаграмма "ящик с усами" вместе с данными по каждому столбцу
import seaborn as sns

# Настройка стиля графиков
sns.set_style("whitegrid")

# Функция для визуализации и анализа столбца
def analyze_column(column_data, column_name):
    # Заголовок для вывода
```

```

print(f"\n--- Анализ столбца: {column_name} ---\n")

# Гистограмма распределения
plt.figure(figsize=(12, 6))
sns.kdeplot(
    data=column_data,
    shade=True,
    palette='colorblind',
    color='g',
    alpha=0.7
)
plt.title(f'Распределение значений для {column_name}')
plt.xlabel(column_name)
plt.ylabel('Плотность')
plt.show()

# Диаграмма "Ящик с усами"
plt.figure(figsize=(8, 4))
sns.boxplot(
    x=column_data,
    color='g'
)
plt.title(f'Boxplot для {column_name}')
plt.show()

# Статистические показатели
stats = {
    'Минимум': np.min(column_data),
    'Максимум': np.max(column_data),
    'Среднее': np.mean(column_data),
    'Медиана': np.median(column_data)
}

# Вывод статистики
for stat, value in stats.items():
    print(f'{stat}: {value:.4f}')
print("\n")

# Основной цикл обработки
for column_name in column_names:
    try:
        # Извлекаем данные столбца
        column_data = df[column_name]

        # Проверяем, что данные числовые
        if np.issubdtype(column_data.dtype, np.number):
            analyze_column(column_data, column_name)
        else:
            print(f'Пропускаем столбец {column_name}: не числовой тип данных\n')

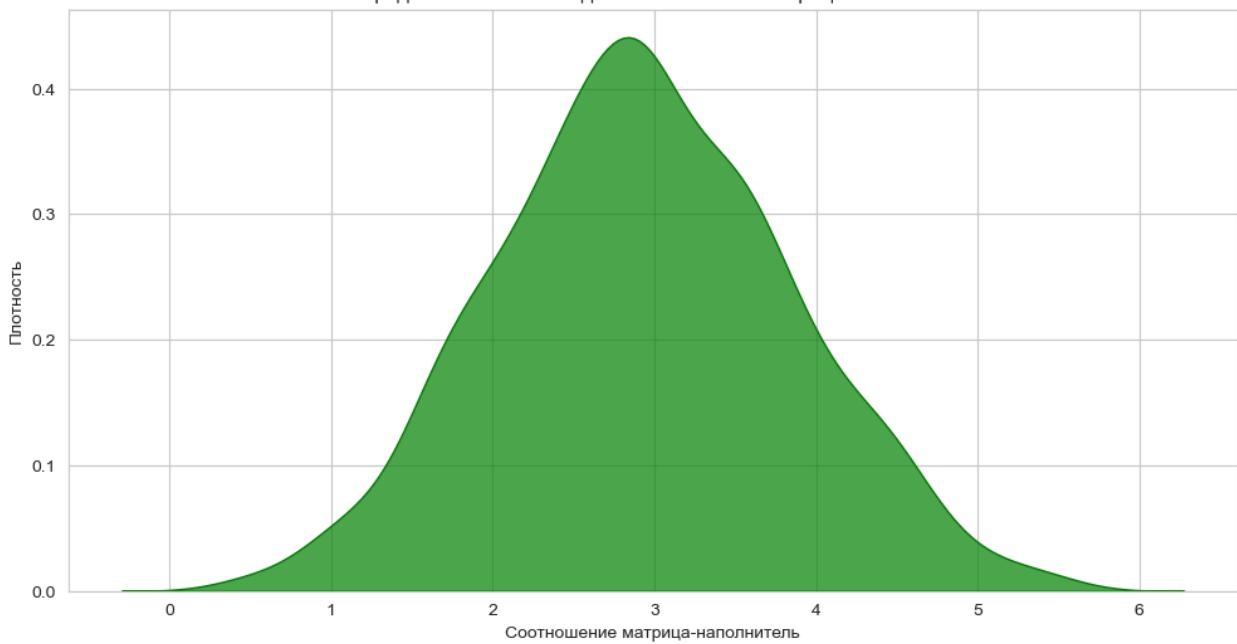
    except Exception as e:
        print(f'Ошибка при обработке столбца {column_name}: {str(e)}\n')

# Кроме "Угол нашивки, град" и "Поверхностная плотность, г/м²" остальные переменные

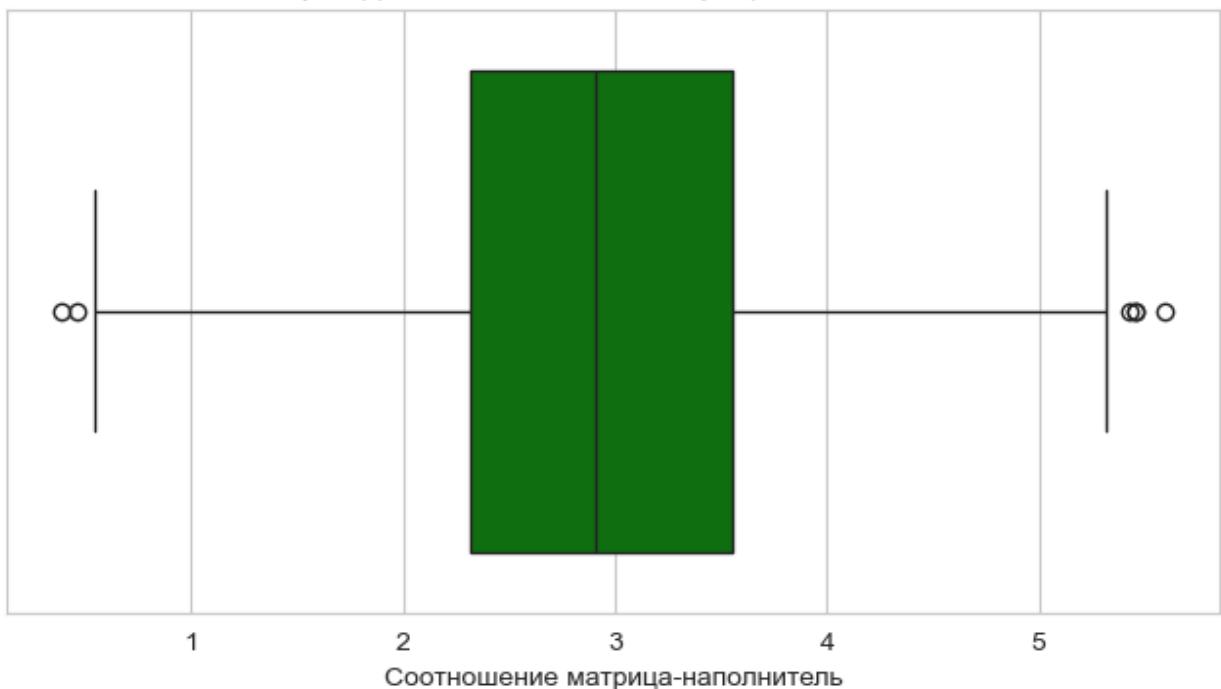
```

--- Анализ столбца: Соотношение матрица-наполнитель ---

Распределение значений для Соотношение матрица-наполнитель



Вохплот для Соотношение матрица-наполнитель



Минимум: 0.3894

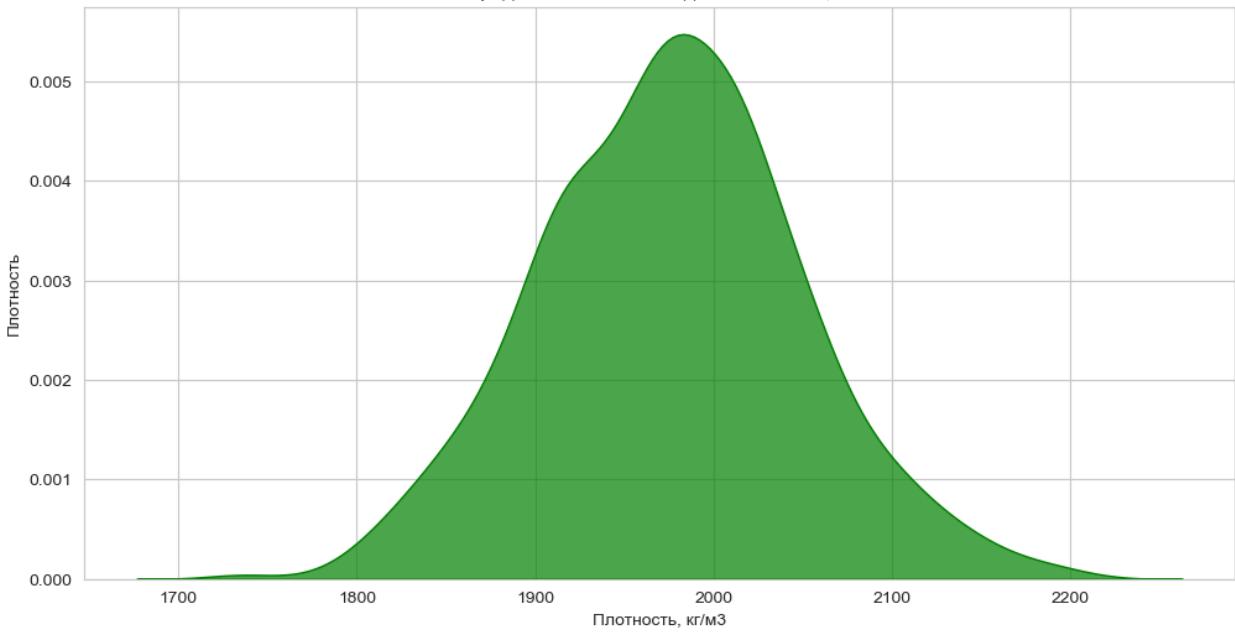
Максимум: 5.5917

Среднее: 2.9304

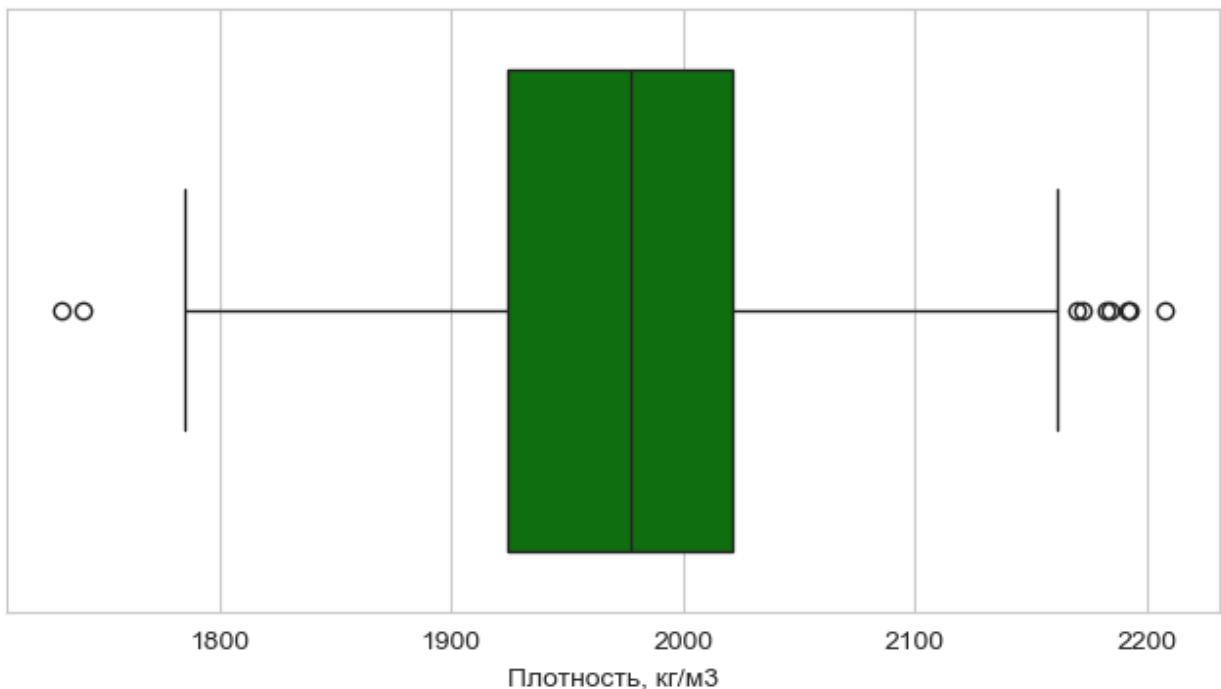
Медиана: 2.9069

--- Анализ столбца: Плотность, кг/м3 ---

Распределение значений для Плотность, кг/м³



Boxplot для Плотность, кг/м³



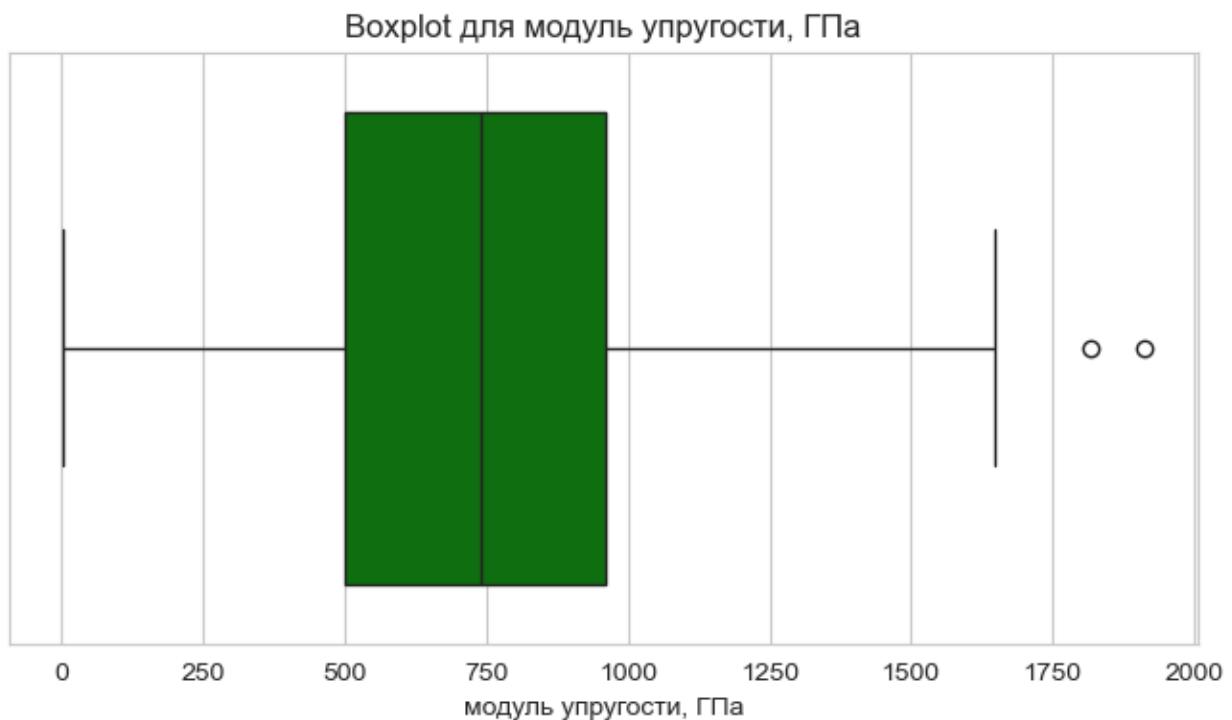
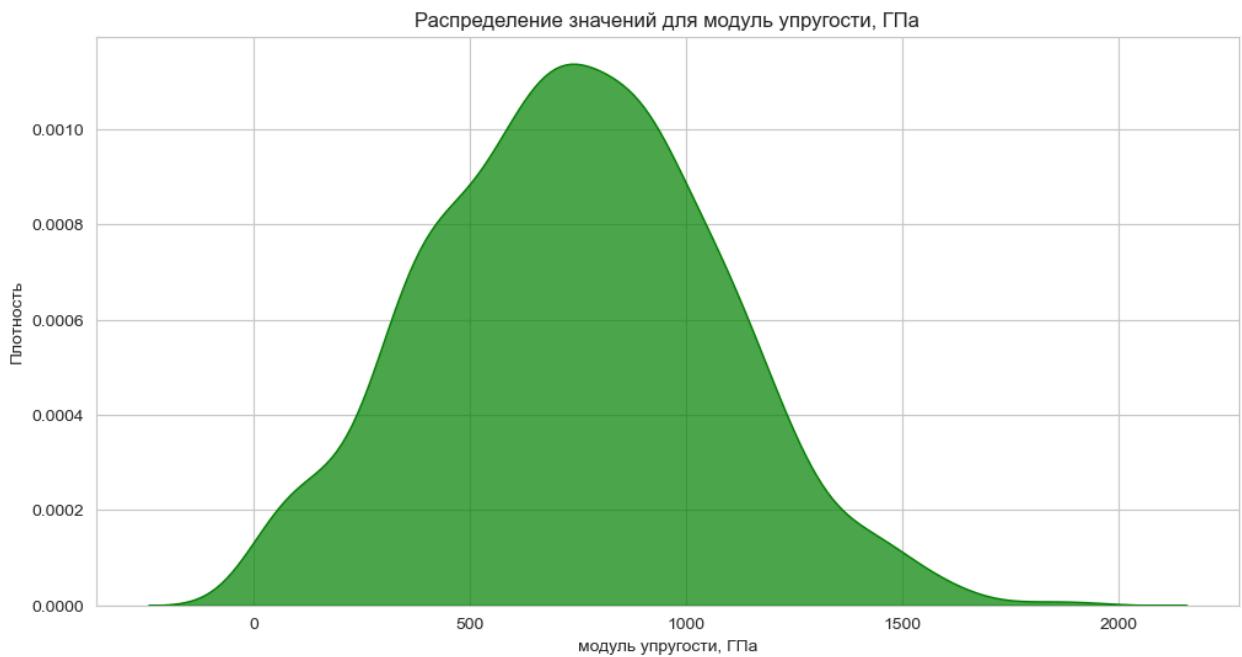
Минимум: 1731.7646

Максимум: 2207.7735

Среднее: 1975.7349

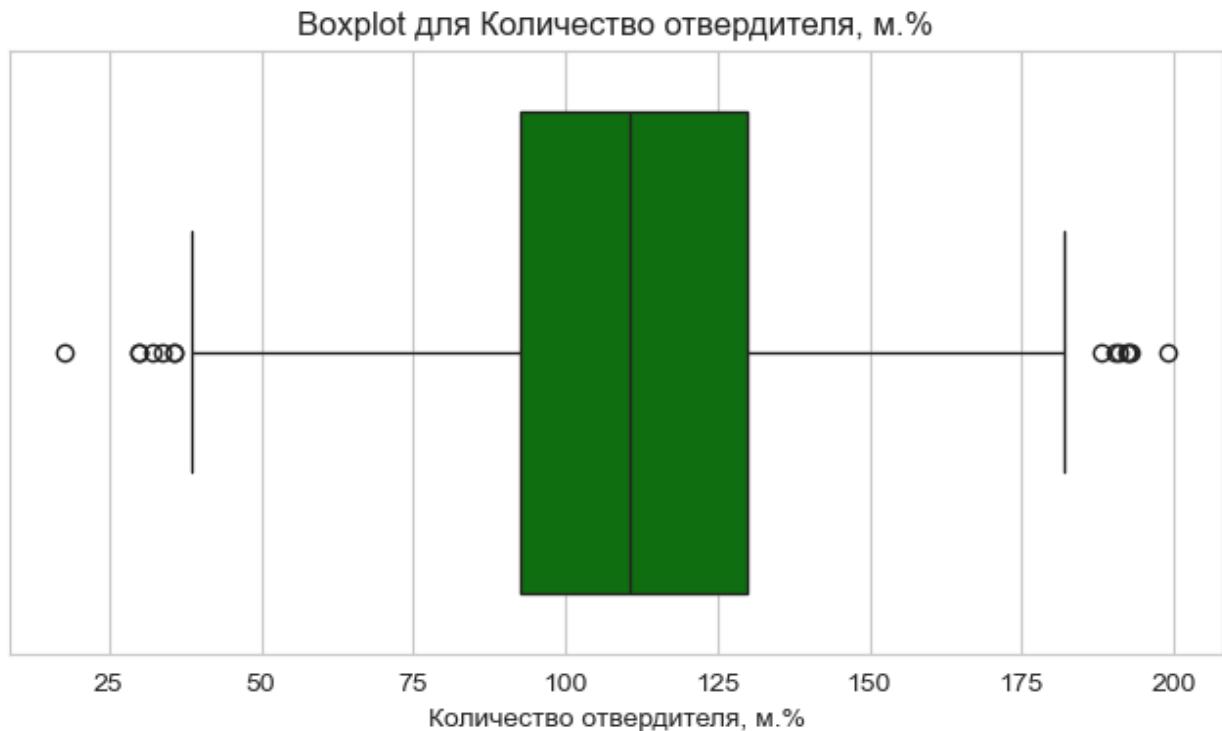
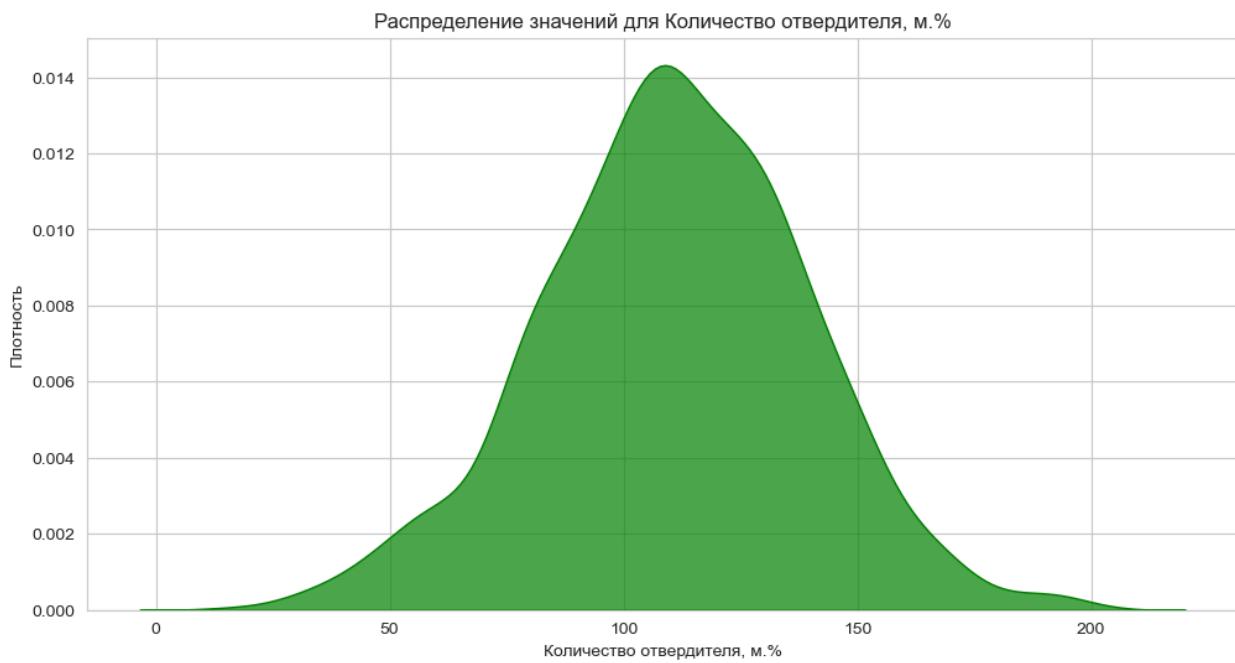
Медиана: 1977.6217

--- Анализ столбца: модуль упругости, ГПа ---



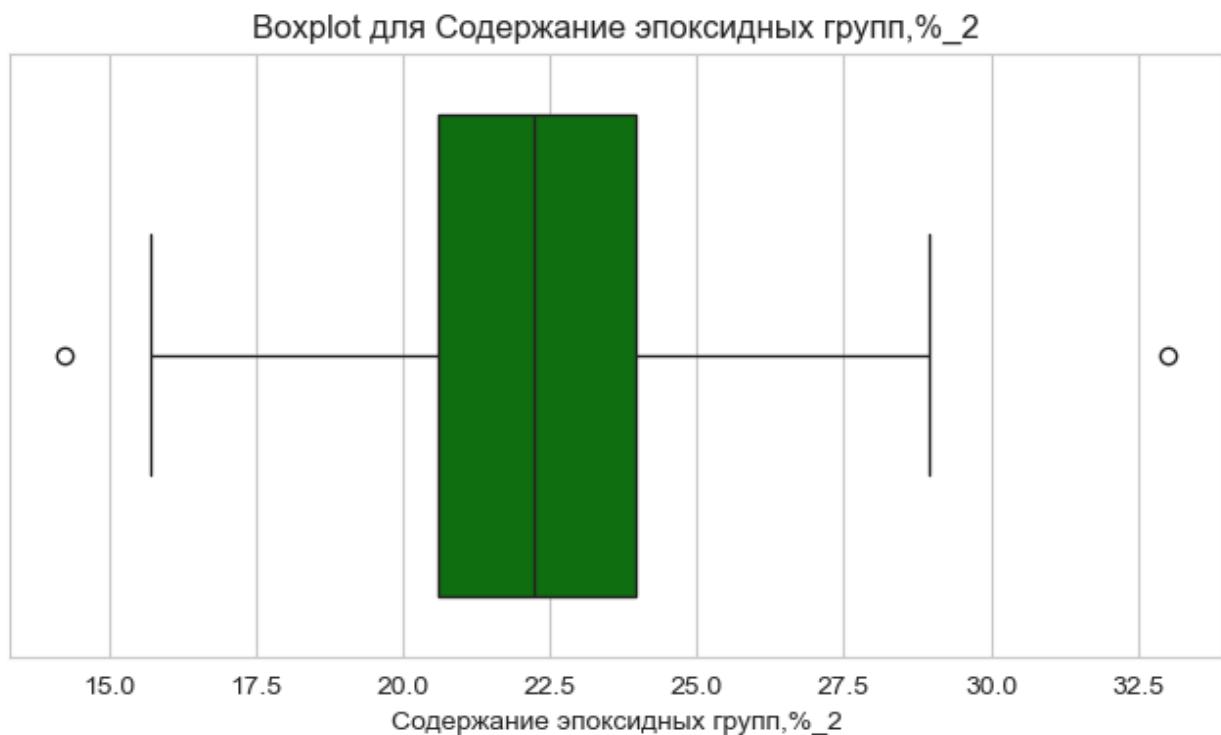
Минимум: 2.4369
Максимум: 1911.5365
Среднее: 739.9232
Медиана: 739.6643

--- Анализ столбца: Количество отвердителя, м.% ---



Минимум: 17.7403
Максимум: 198.9532
Среднее: 110.5708
Медиана: 110.5648

--- Анализ столбца: Содержание эпоксидных групп,%_2 ---



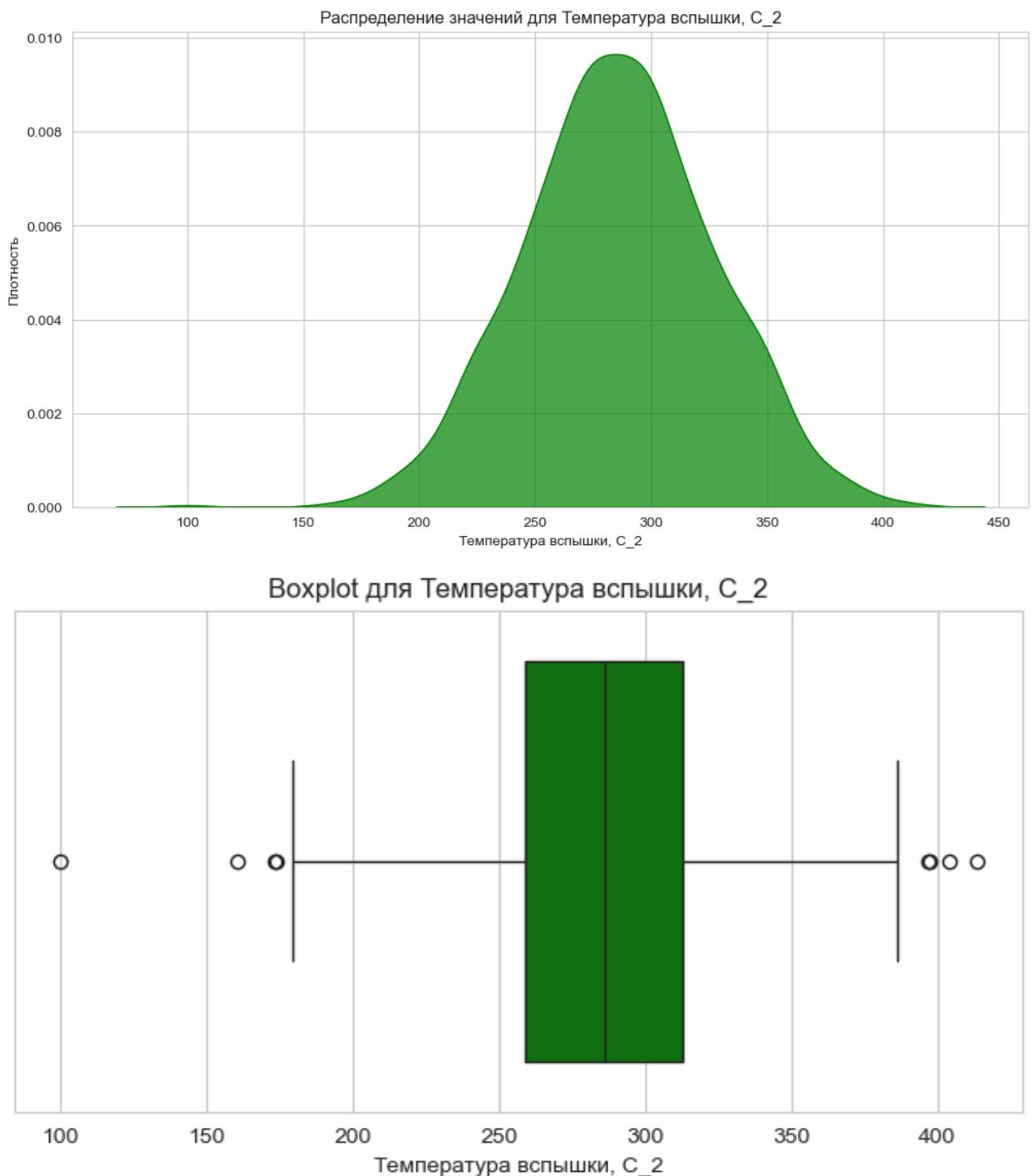
Минимум: 14.2550

Максимум: 33.0000

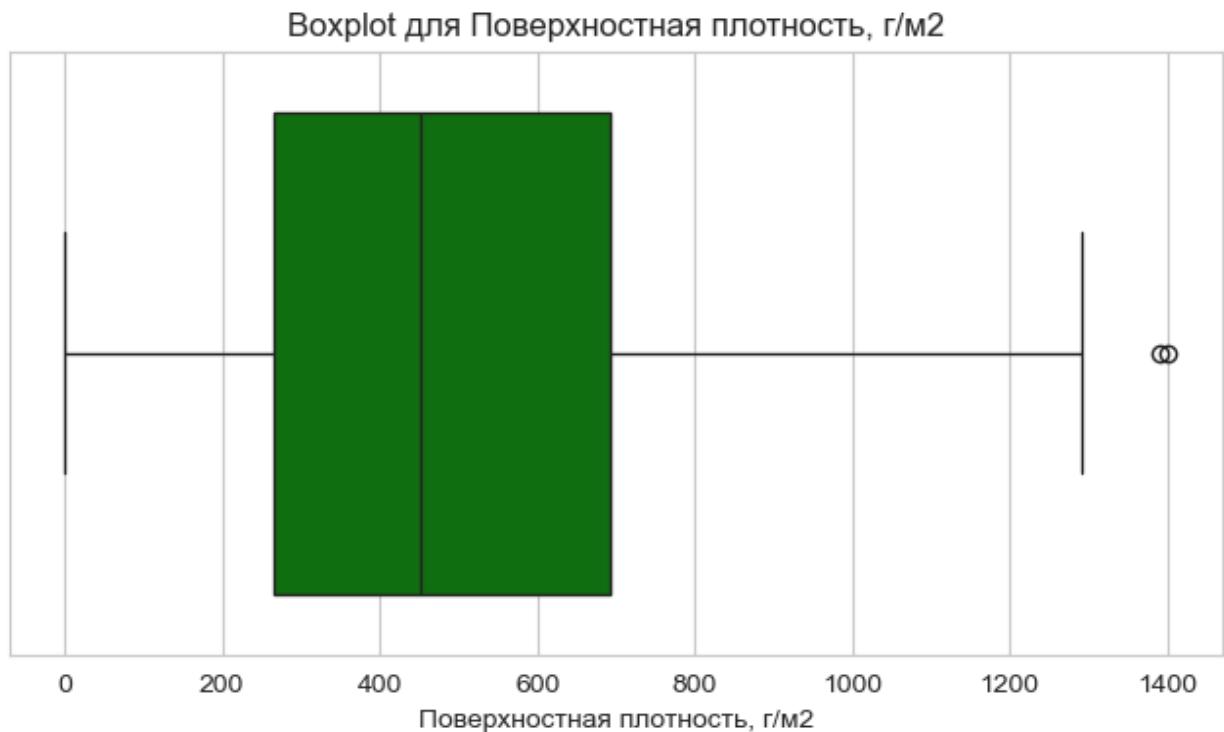
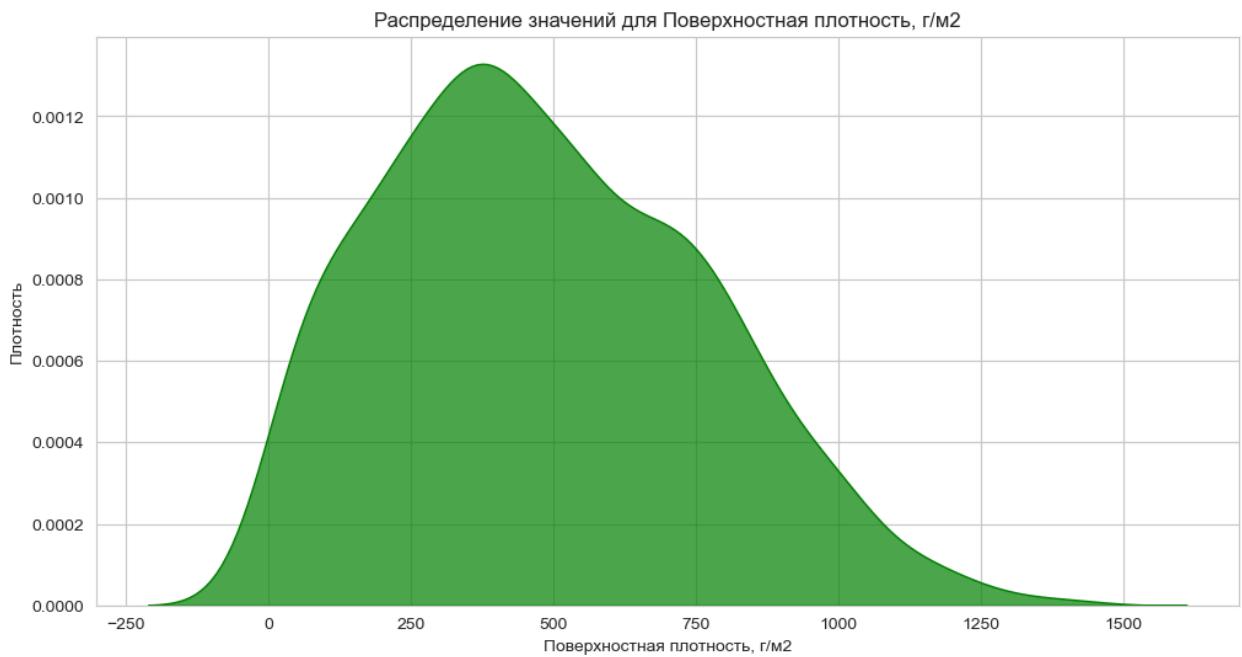
Среднее: 22.2444

Медиана: 22.2307

--- Анализ столца: Температура вспышки, C_2 ---



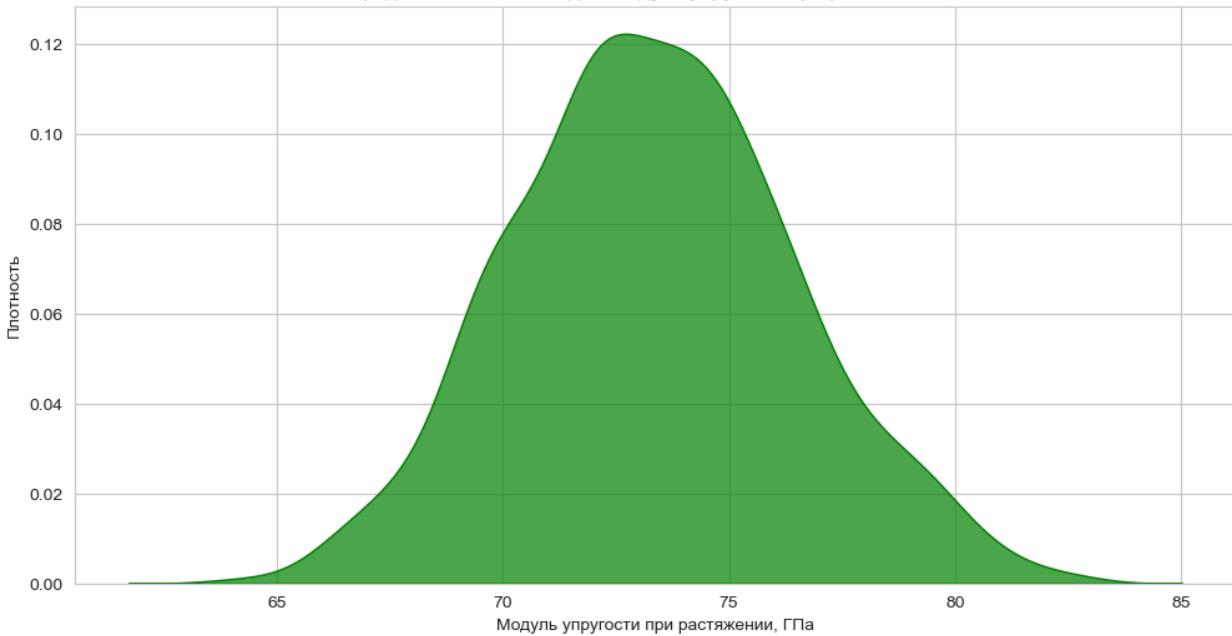
--- Анализ столбца: Поверхностная плотность, г/м² ---



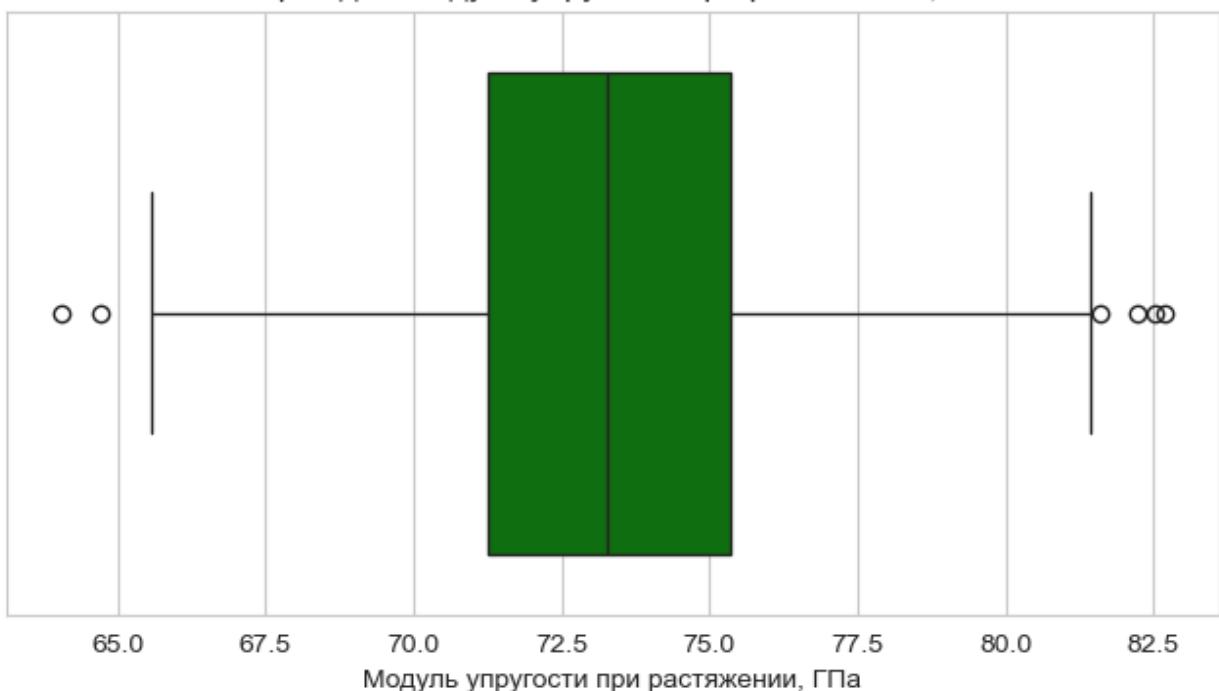
Минимум: 0.6037
Максимум: 1399.5424
Среднее: 482.7318
Медиана: 451.8644

--- Анализ столбца: Модуль упругости при растяжении, ГПа ---

Распределение значений для Модуль упругости при растяжении, ГПа



Вохplot для Модуль упругости при растяжении, ГПа



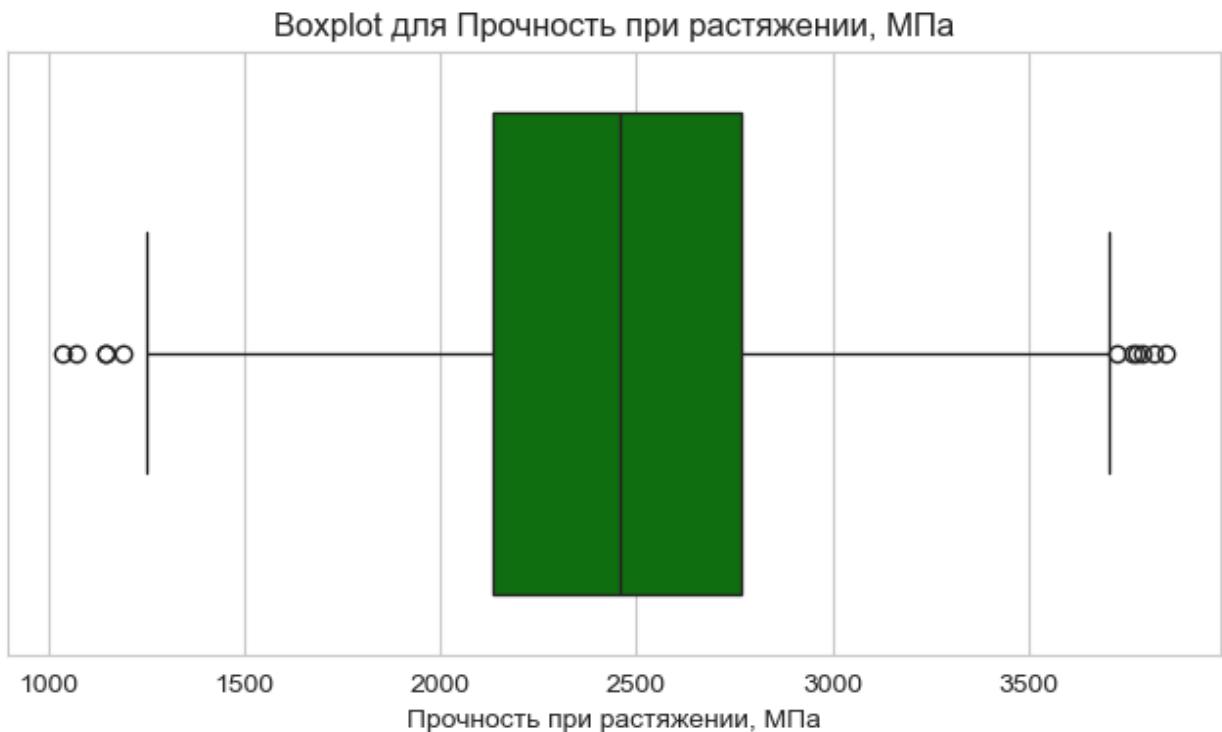
Минимум: 64.0541

Максимум: 82.6821

Среднее: 73.3286

Медиана: 73.2688

--- Анализ столбца: Прочность при растяжении, МПа ---



Минимум: 1036.8566

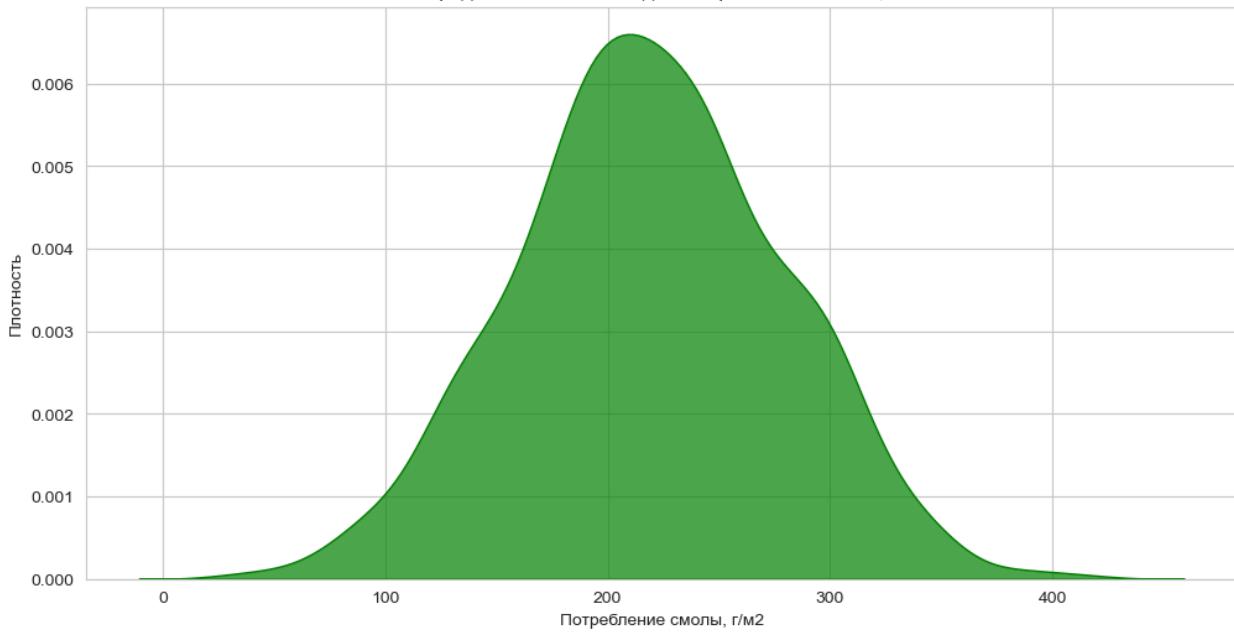
Максимум: 3848.4367

Среднее: 2466.9228

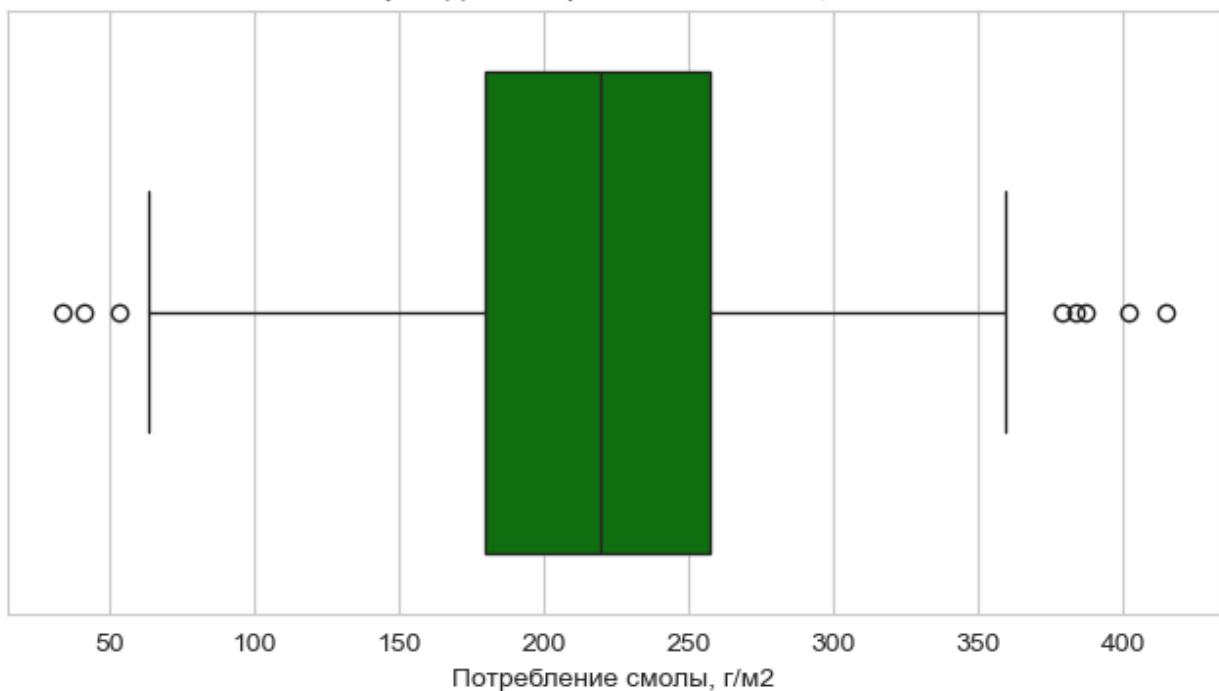
Медиана: 2459.5245

--- Анализ столбца: Потребление смолы, г/м² ---

Распределение значений для Потребление смолы, г/м²



Boxplot для Потребление смолы, г/м²



Минимум: 33.8030

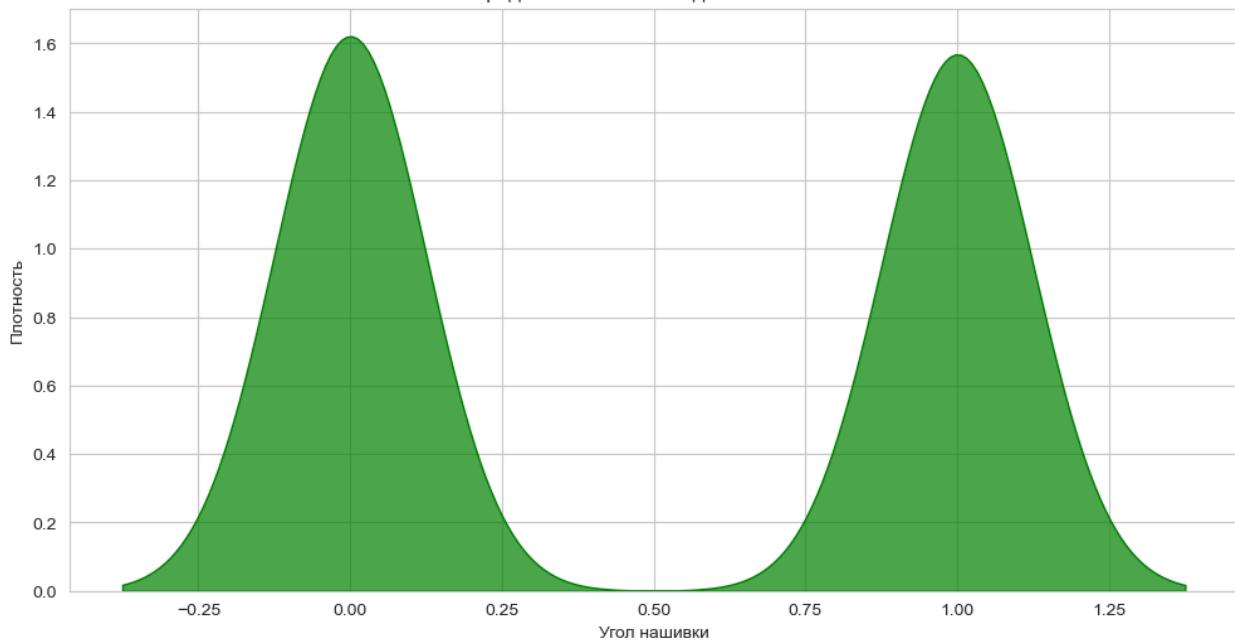
Максимум: 414.5906

Среднее: 218.4231

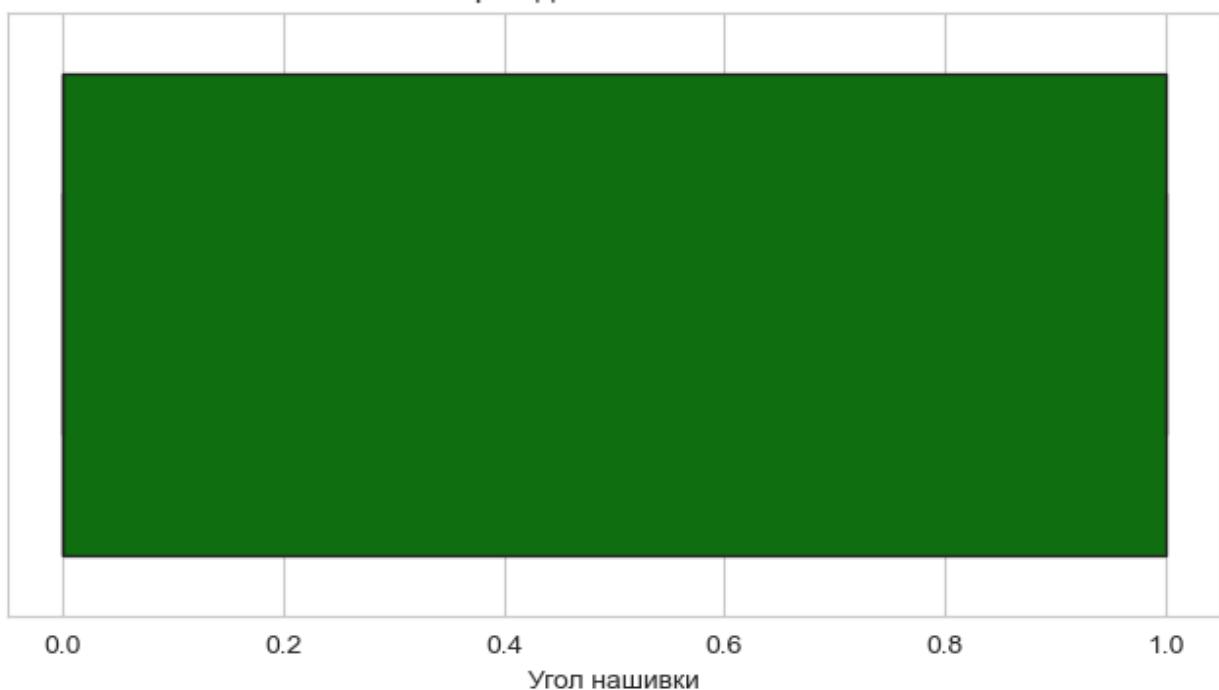
Медиана: 219.1989

--- Анализ столбца: Угол нашивки ---

Распределение значений для Угол нашивки



Boxplot для Угол нашивки



Минимум: 0.0000

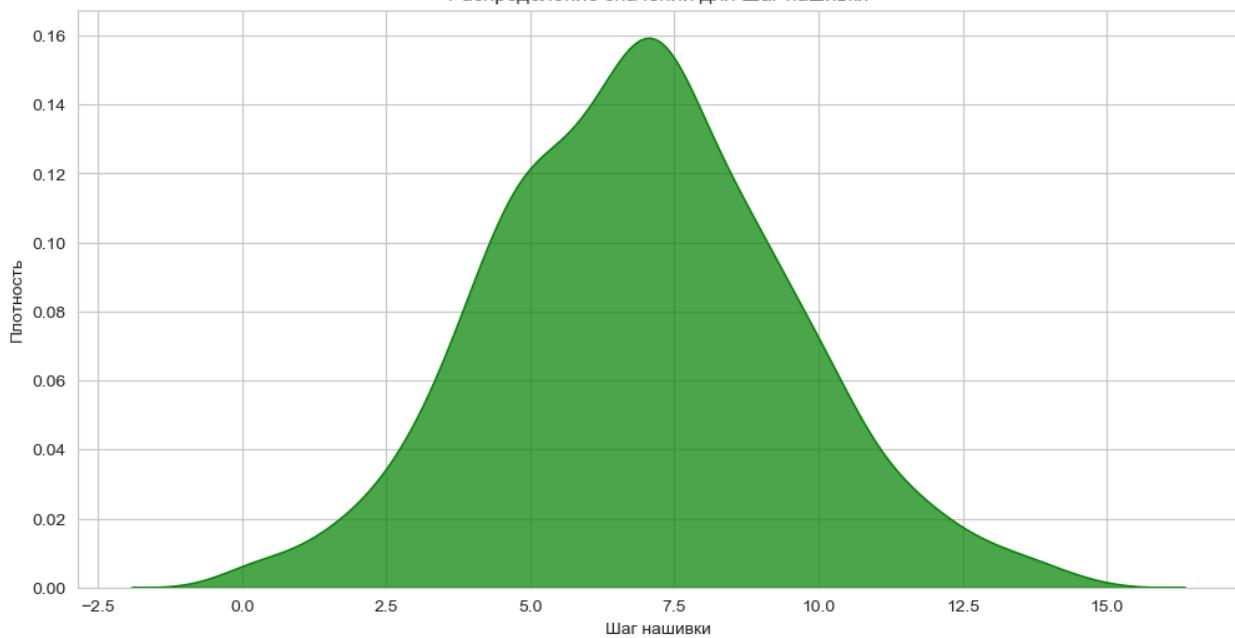
Максимум: 1.0000

Среднее: 0.4917

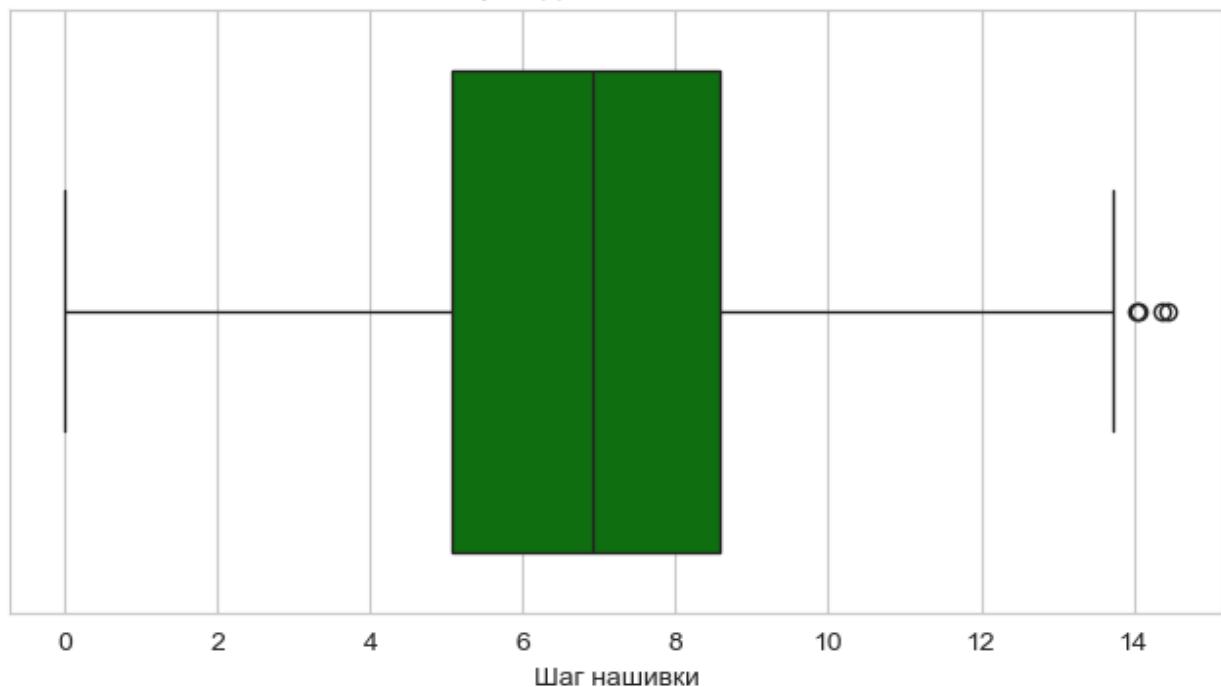
Медиана: 0.0000

--- Анализ столбца: Шаг нашивки ---

Распределение значений для Шаг нашивки



Boxplot для Шаг нашивки



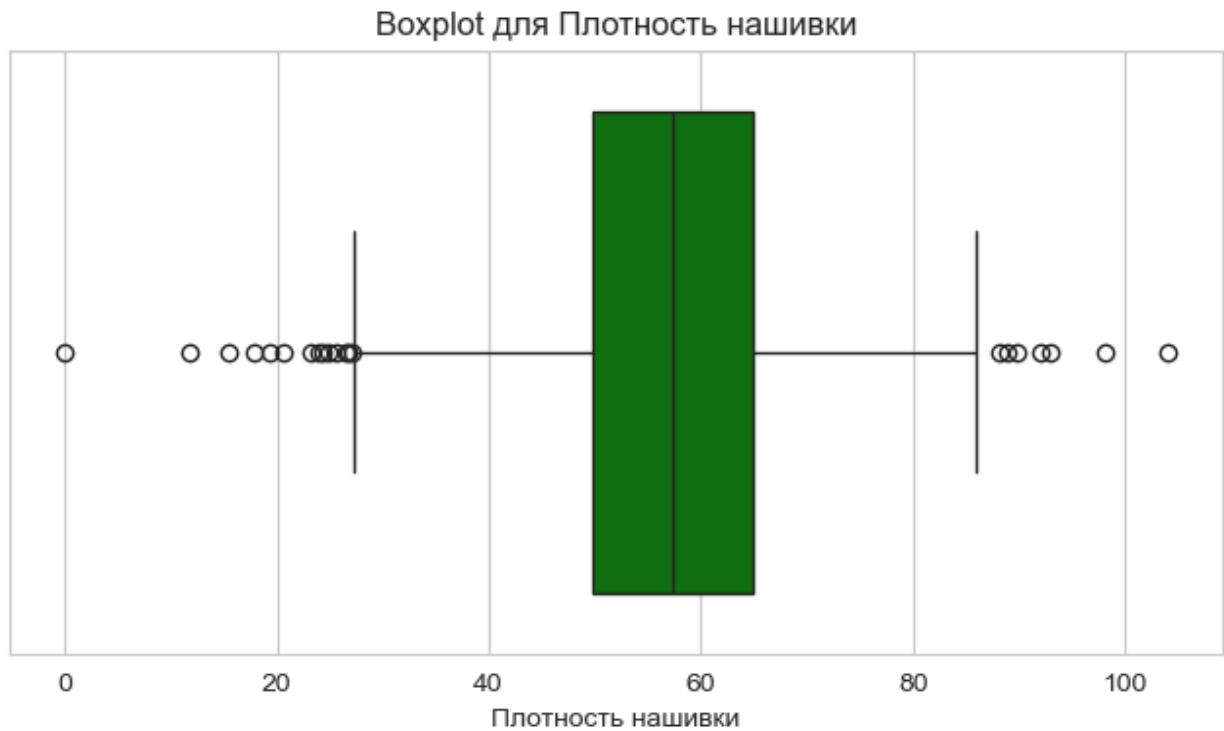
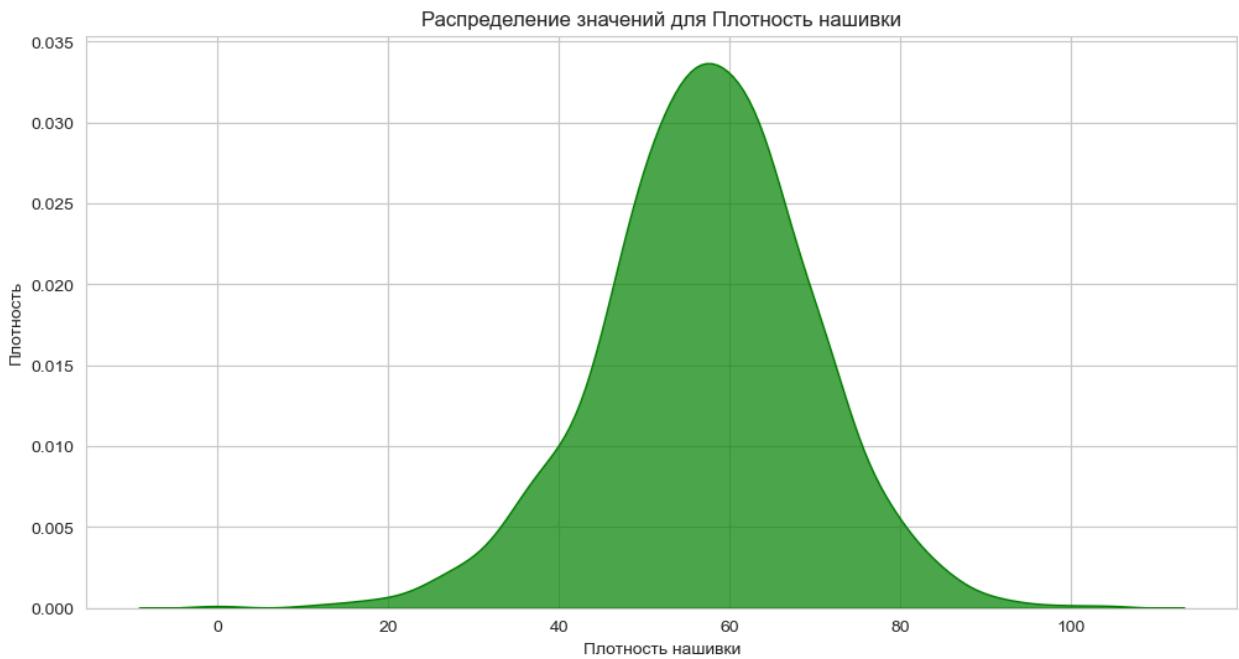
Минимум: 0.0000

Максимум: 14.4405

Среднее: 6.8992

Медиана: 6.9161

--- Анализ столбца: Плотность нашивки ---

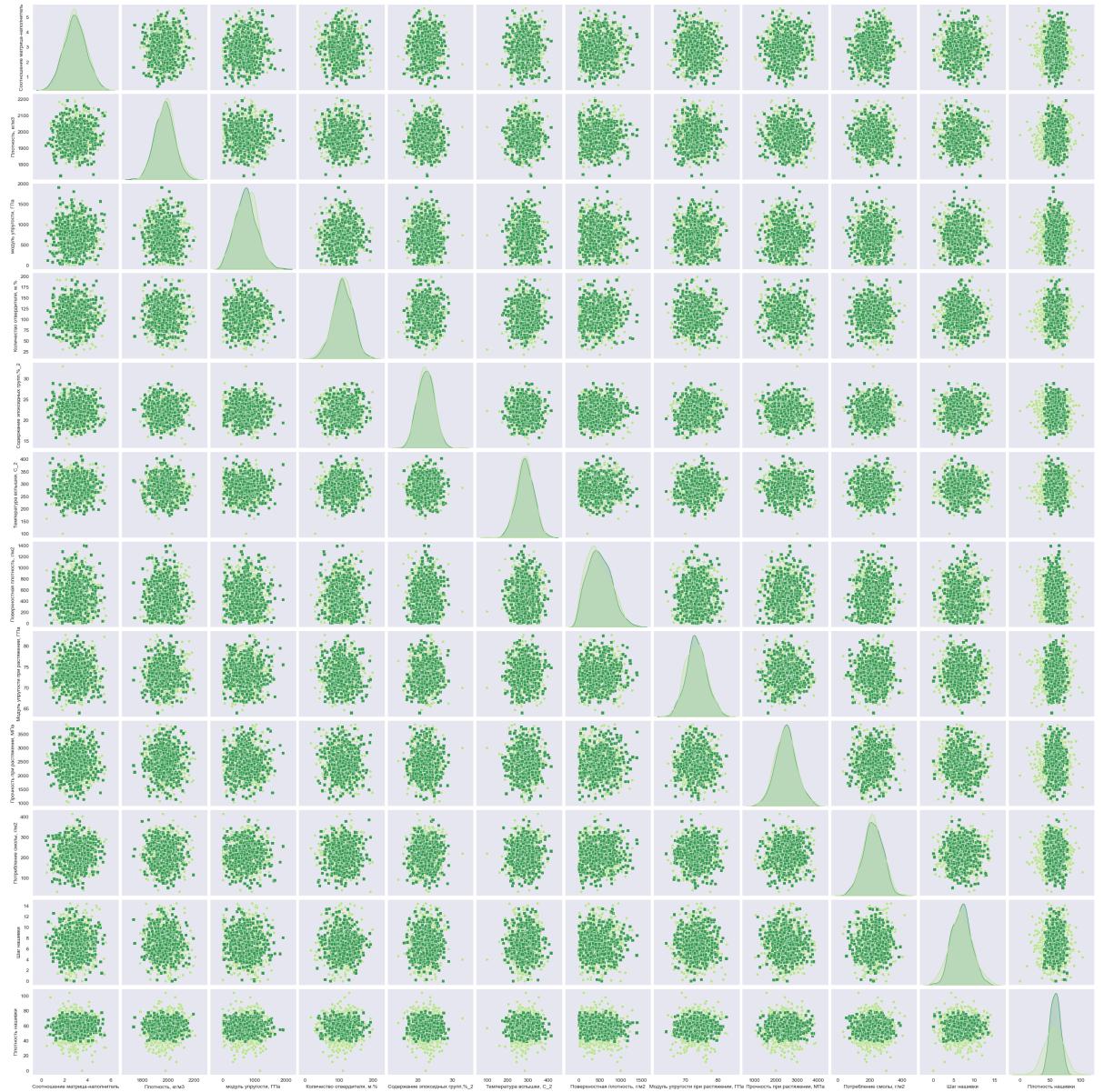


Минимум: 0.0000
Максимум: 103.9889
Среднее: 57.1539
Медиана: 57.3419

In [40]: # Попарные графики рассеяния точек (матрица диаграмм рассеяния) (первый вариант)
sns.set_style('dark')
sns.pairplot(df, hue = 'Угол нашивки', markers = ["o", "s"], diag_kind = 'auto')
Попарные графики рассеяния точек так же не показывают какой-либо зависимости

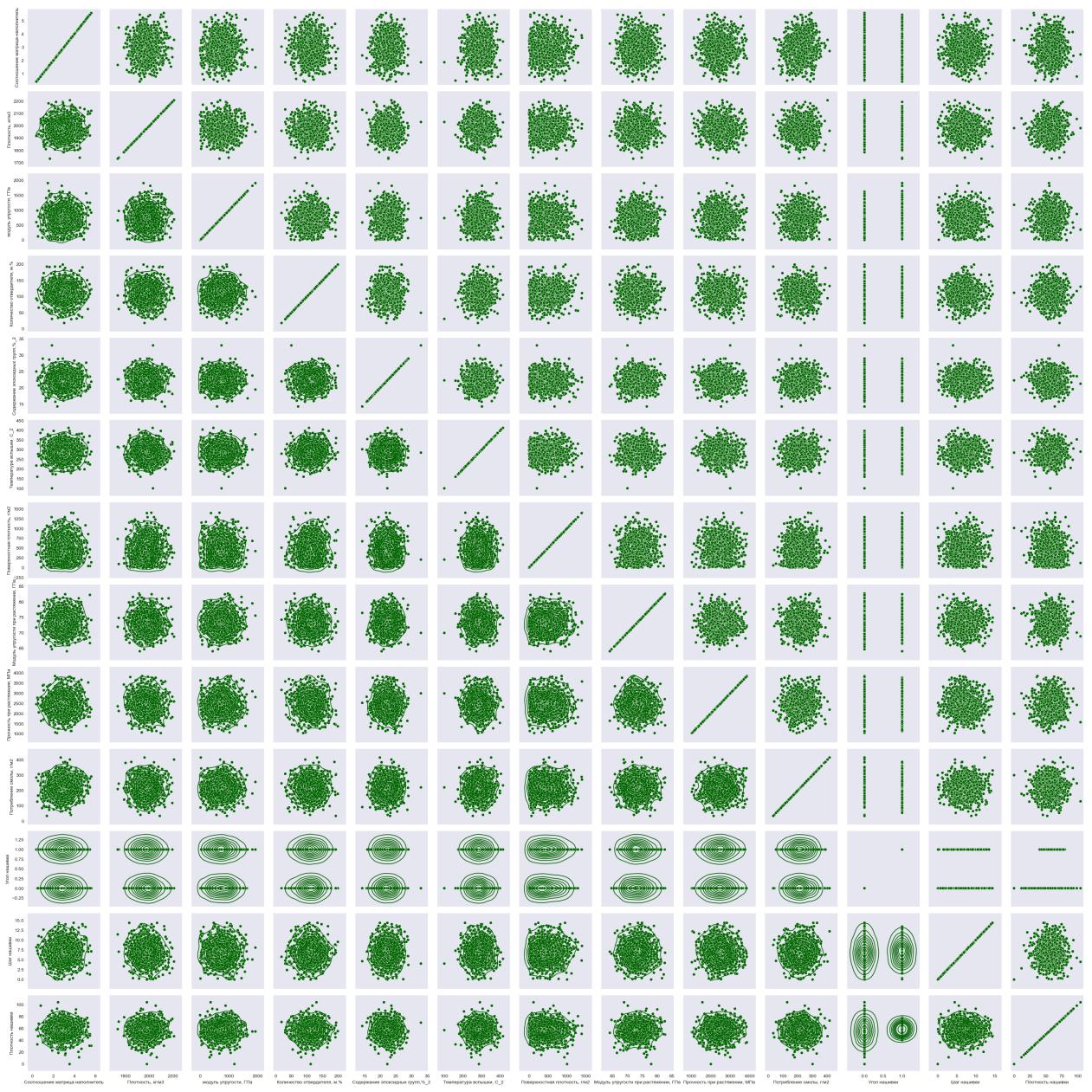
```
# из графиков можно наблюдать выбросы, потому что некоторые точки располагают
```

Out[40]: <seaborn.axisgrid.PairGrid at 0x258e5c39400>



```
In [41]: # графики рассеяния точек
g = sns.PairGrid(df[df.columns])
g.map(sns.scatterplot, color = 'darkgreen')
g.map_upper(sns.scatterplot, color = 'darkgreen')
g.map_lower(sns.kdeplot, color = 'darkgreen')
plt.show
# Корреляции нет
```

Out[41]: <function matplotlib.pyplot.show(close=None, block=None)>



```
In [42]: import matplotlib.pyplot as plt
import scipy.stats as stats # Добавляем импорт stats
```

```
# Функция для построения QQ-графиков
def plot_qq_plots(dataframe):
    for column_name in dataframe.columns:
        # Проверяем, что столбец числовой
        if pd.api.types.is_numeric_dtype(dataframe[column_name]):
            plt.figure(figsize=(6, 4))

            # Строим QQ-график
            res = stats.probplot(
                dataframe[column_name],
                plot=plt,
                fit=True # Добавляем фитинг
            )
```

```

# Настраиваем отображение
plt.title(column_name, fontsize=10)
plt.xlabel("Теоретические квантили", fontsize=10)
plt.ylabel("Упорядоченные значения", fontsize=10)

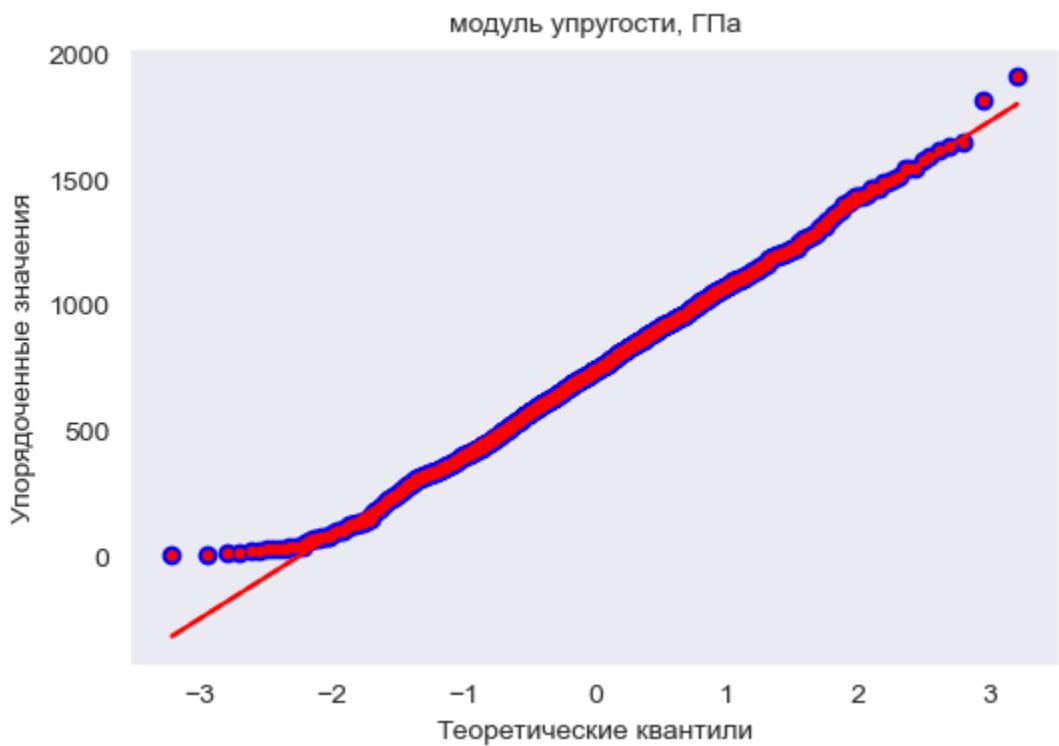
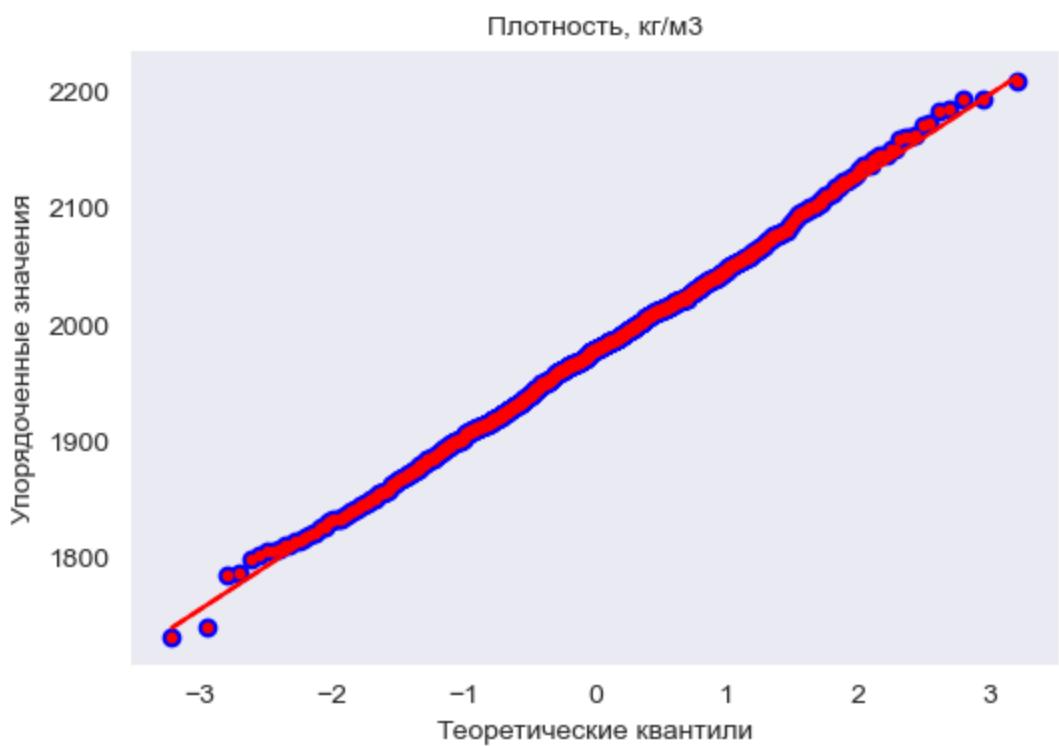
# Добавляем линию регрессии
plt.plot(res[0][0], res[0][1], 'ro', markersize=3)
plt.plot(res[0][0], res[1][0] * res[0][0] + res[1][1], 'r--')

plt.show()
else:
    print(f"Пропускаем столбец {column_name}: не числовой тип данных")

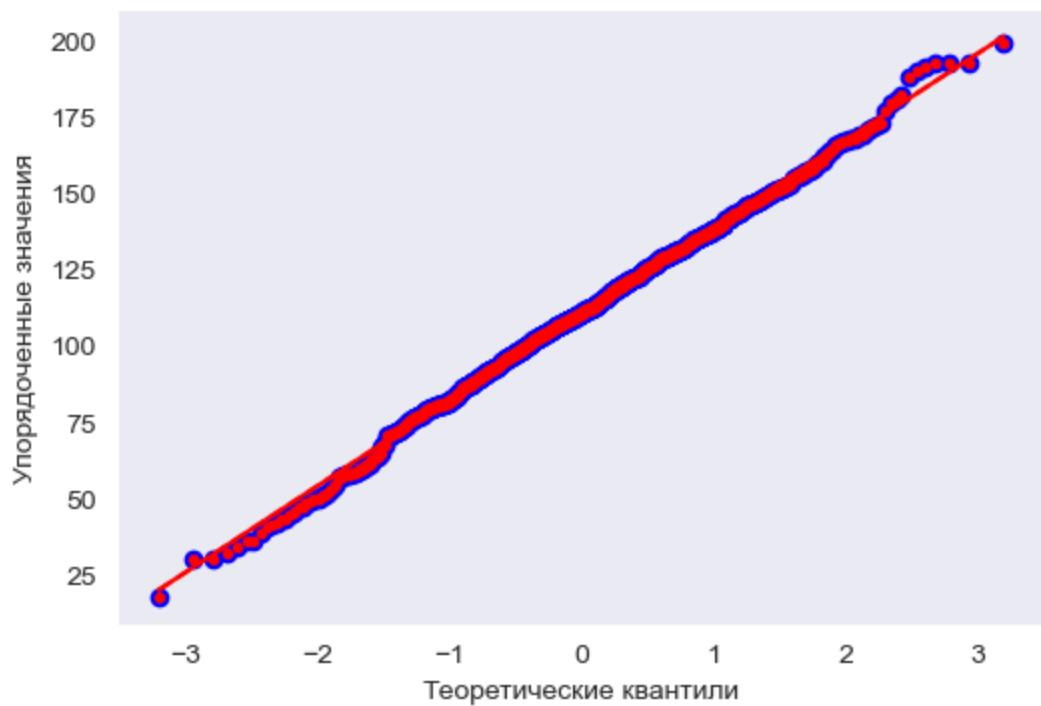
# Вызов функции
plot_qq_plots(df)

```

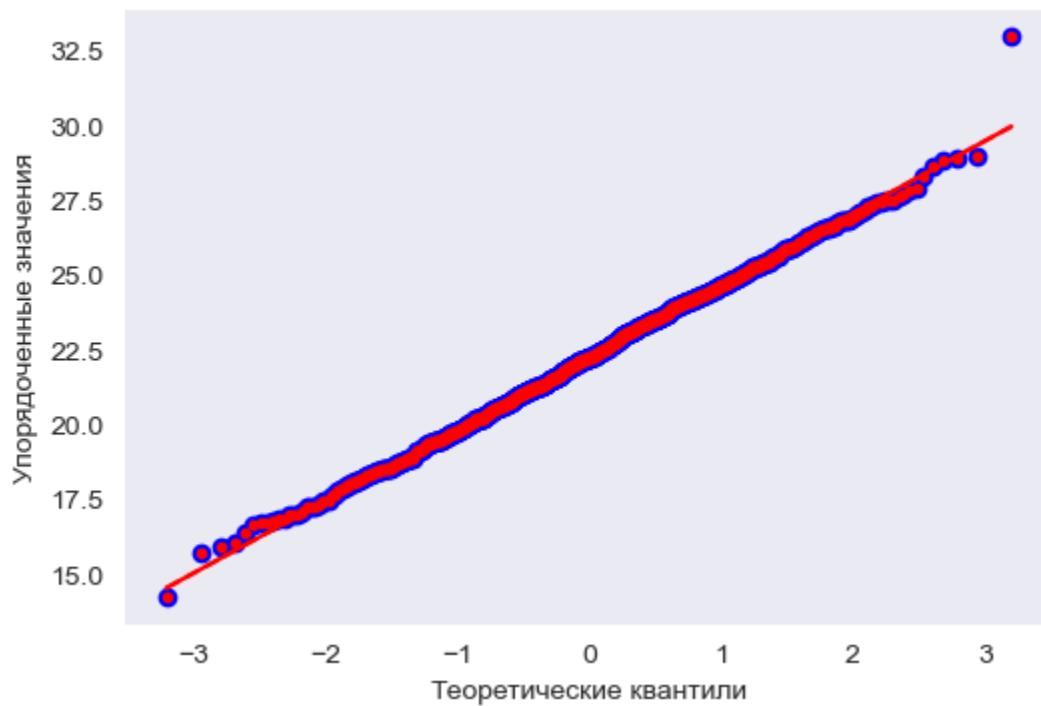




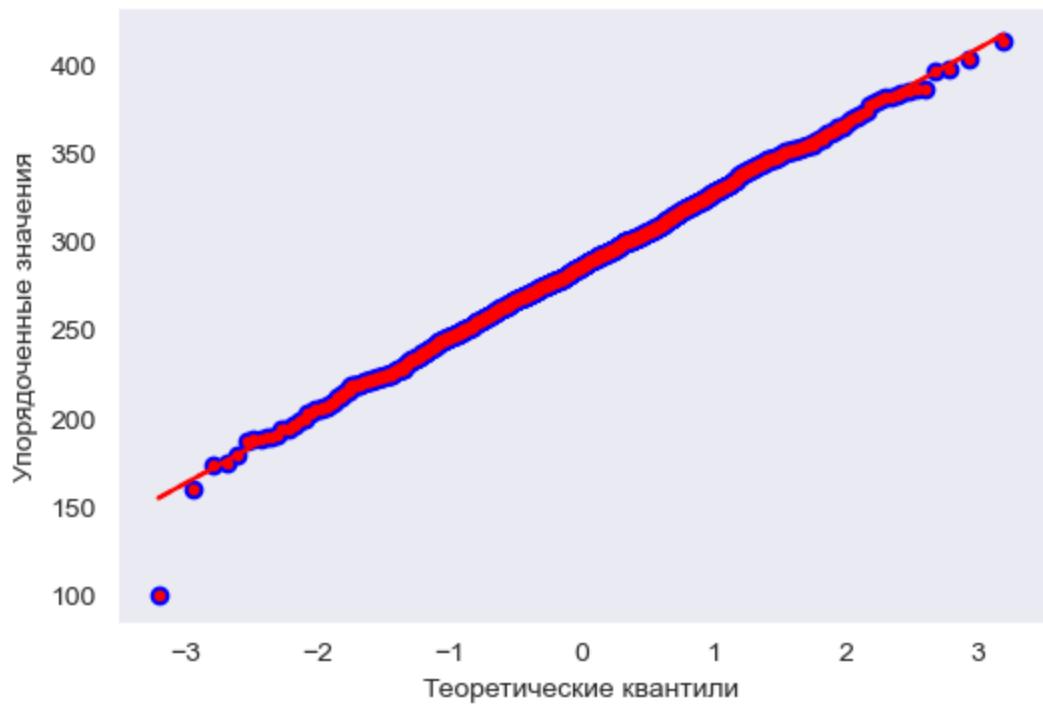
Количество отвердителя, м.%



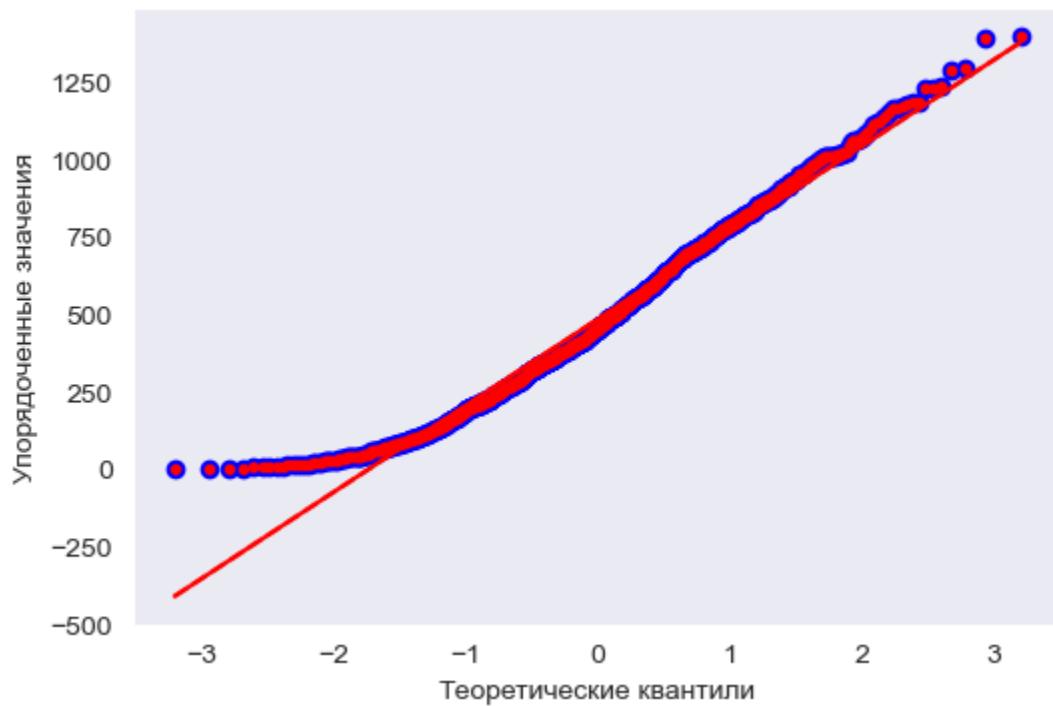
Содержание эпоксидных групп,%_2



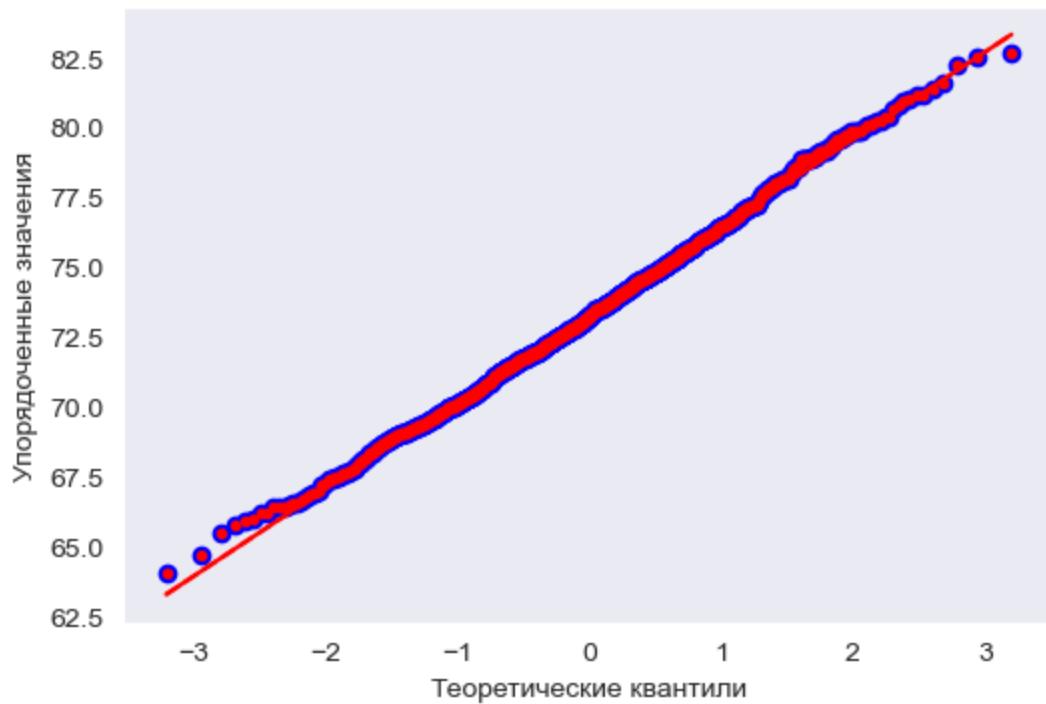
Температура вспышки, С_2



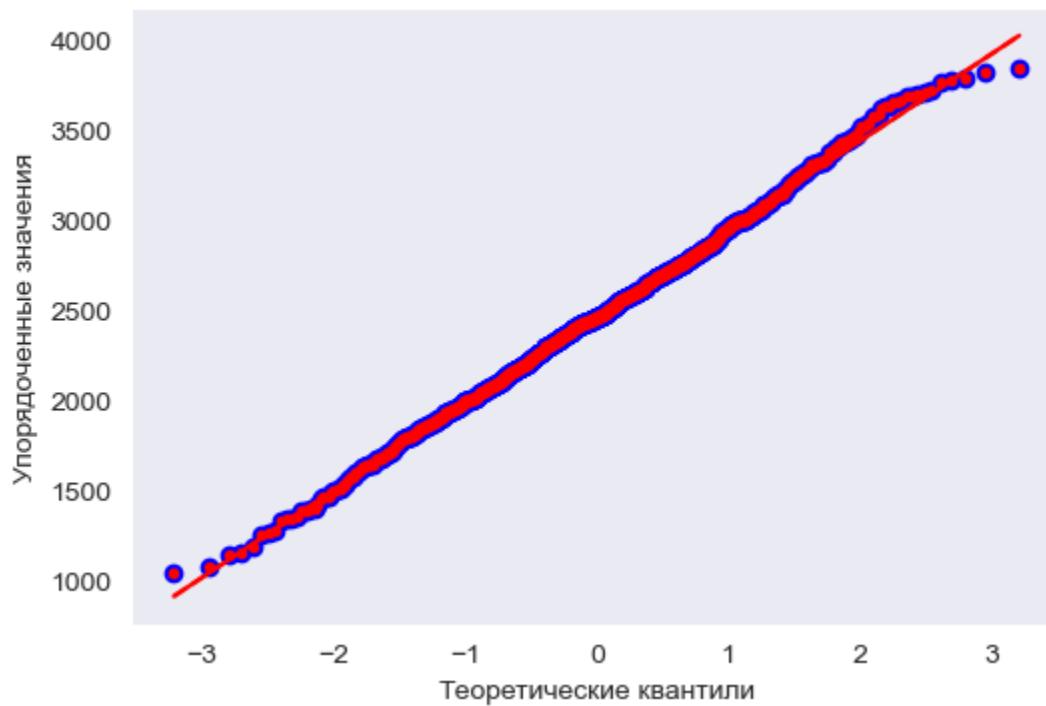
Поверхностная плотность, г/м²



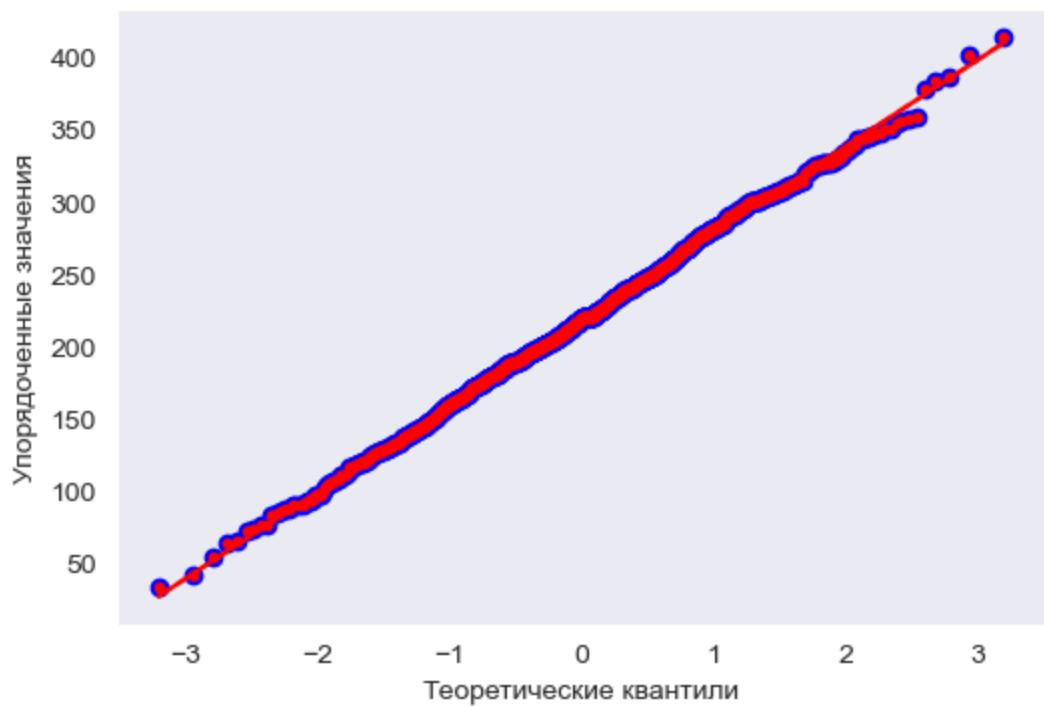
Модуль упругости при растяжении, ГПа



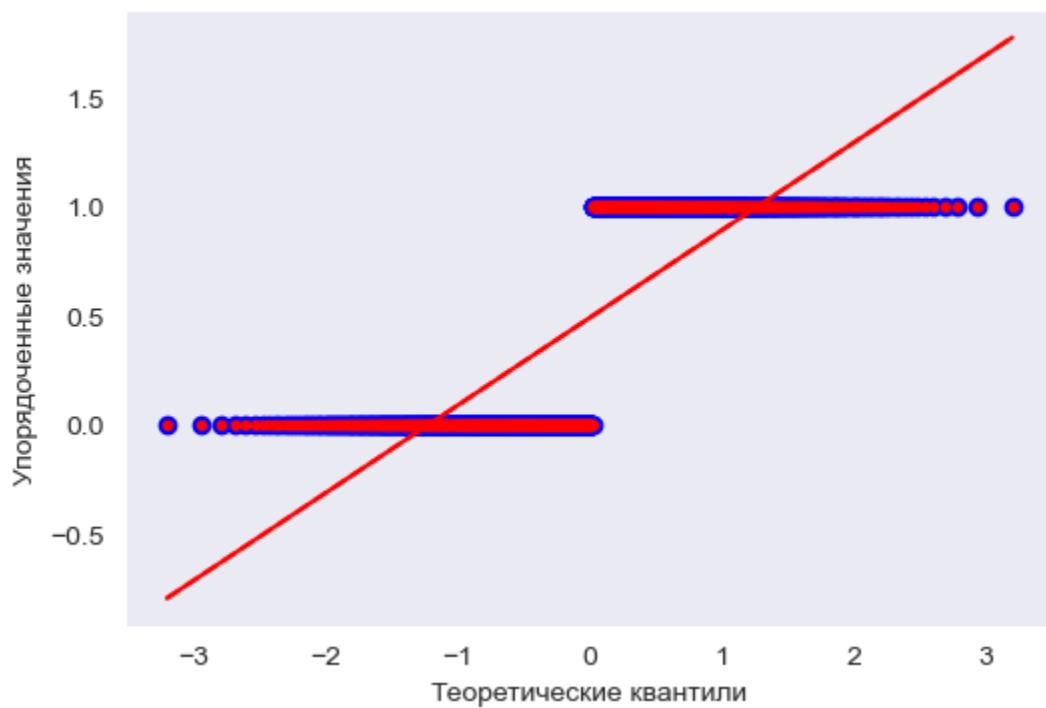
Прочность при растяжении, МПа

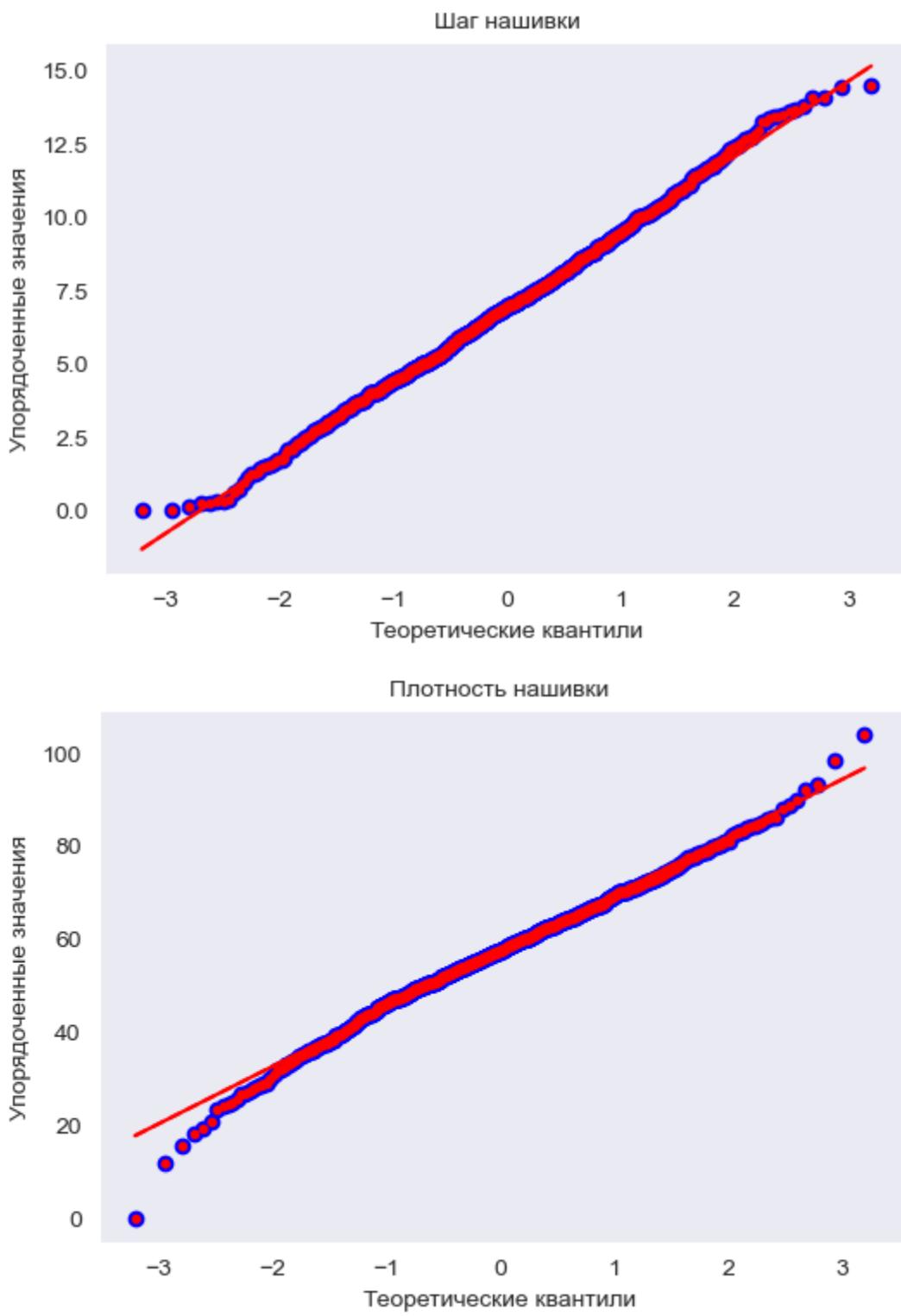


Потребление смолы, г/м²



Угол нашивки



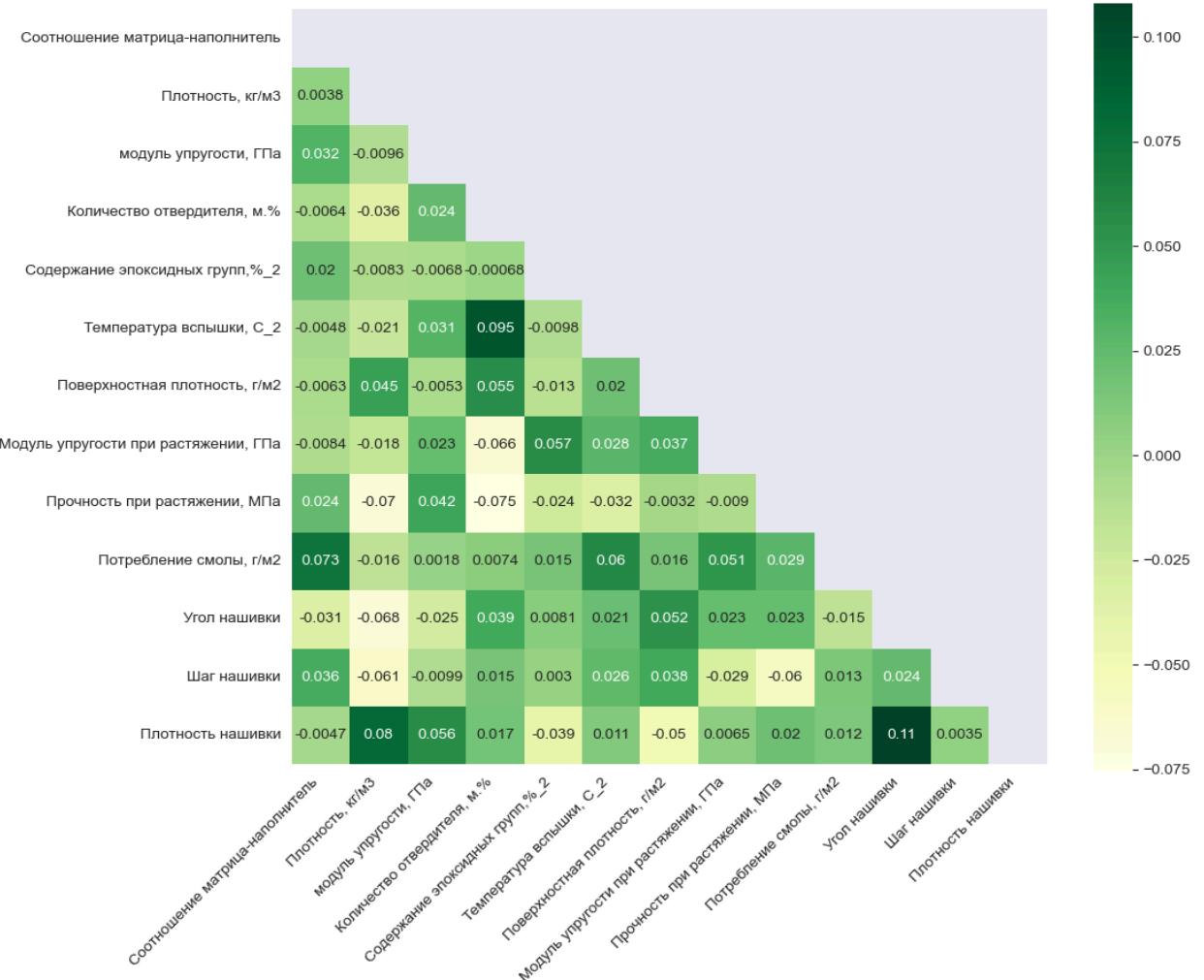


```
In [43]: #Визуализация корреляционной матрицы с помощью тепловой карты
mask = np.triu(df.corr())
# Создаем полотно для отображения большого графика
f, ax = plt.subplots(figsize = (11, 9))
# # Визуализируем данные корреляции и создаем цветовую палитру
sns.heatmap(df.corr(), mask = mask, annot = True, square = True, cmap = 'YlGn'
plt.xticks(rotation = 45, ha='right')
```

```

plt.show()
# Максимальная корреляция между Плотностью нашивки и углом нашивки и составляет
# Корреляция между всеми параметрами очень близка к 0, что говорит об отсутствии

```



```

In [44]: # Создадим переменную для названия всех столбцов. Это нам пригодится при построении
df.columns
#column_names = ["Соотношение матрица-наполнитель", "Плотность, кг/м³", "модуль упругости, ГПа",
#                 "Содержание эпоксидных групп, %_2", "Температура вспышки, С_2", "Поверхностная плотность, г/м2",
#                 "Модуль упругости при растяжении, ГПа", "Прочность при растяжении, МПа",
#                 "Угол нашивки, град", "Шаг нашивки", "Плотность нашивки"]
column_names = df.columns

```

Проведем предобработку данных

```

In [45]: # Шаг 1 Удаление выбросов. Посчитаем, сколько значений у нас в каждом столбце
df.isna().sum()

```

```
Out[45]: 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 [46]: #Для удаления выбросов существует 2 основных метода - метод 3-х сигм и межквартильных расстояний
metod_3s = 0
metod_iq = 0
count_iq = [] # Список, куда записывается количество выбросов по каждой колонке
count_3s = [] # Список, куда записывается количество выбросов по каждой колонке
for column in df:
    d = df.loc[:, [column]]
    # методом 3-х СИГМ
    zscore = (df[column] - df[column].mean()) / df[column].std()
    d['3s'] = zscore.abs() > 3
    metod_3s += d['3s'].sum()
    count_3s.append(d['3s'].sum())
    print(column, '3s', ': ', d['3s'].sum())

    # методом межквартильных расстояний
    q1 = np.quantile(df[column], 0.25)
    q3 = np.quantile(df[column], 0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    d['iq'] = (df[column] <= lower) | (df[column] >= upper)
    metod_iq += d['iq'].sum()
    count_iq.append(d['iq'].sum())
    print(column, ': ', d['iq'].sum())
print('Метод 3-х сигм, выбросов:', metod_3s)
print('Метод межквартильных расстояний, выбросов:', metod_iq)
```

```
Соотношение матрица-наполнитель 3s : 0
Соотношение матрица-наполнитель : 6
Плотность, кг/м3 3s : 3
Плотность, кг/м3 : 9
модуль упругости, ГПа 3s : 2
модуль упругости, ГПа : 2
Количество отвердителя, м.% 3s : 2
Количество отвердителя, м.% : 14
Содержание эпоксидных групп,%_2 3s : 2
Содержание эпоксидных групп,%_2 : 2
Температура вспышки, С_2 3s : 3
Температура вспышки, С_2 : 8
Поверхностная плотность, г/м2 3s : 2
Поверхностная плотность, г/м2 : 2
Модуль упругости при растяжении, ГПа 3s : 0
Модуль упругости при растяжении, ГПа : 6
Прочность при растяжении, МПа 3s : 0
Прочность при растяжении, МПа : 11
Потребление смолы, г/м2 3s : 3
Потребление смолы, г/м2 : 8
Угол нашивки 3s : 0
Угол нашивки : 0
Шаг нашивки 3s : 0
Шаг нашивки : 4
Плотность нашивки 3s : 7
Плотность нашивки : 21
Метод 3-х сигм, выбросов: 24
Метод межквартильных расстояний, выбросов: 93
```

```
In [47]: # С целью предотвращения удаления особенностей признака или допущения ошибки,
m = df.copy()
for i in df.columns:
    m[i] = abs((df[i] - df[i].mean()) / df[i].std())
    print(f"sum(m[{i}] > 3) выбросов в признаке {i}")
print(f' Всего sum(sum(m.values > 3)) выброса')
```

```
0 выбросов в признаке Соотношение матрица-наполнитель
3 выбросов в признаке Плотность, кг/м3
2 выбросов в признаке модуль упругости, ГПа
2 выбросов в признаке Количество отвердителя, м.%
2 выбросов в признаке Содержание эпоксидных групп,%_2
3 выбросов в признаке Температура вспышки, С_2
2 выбросов в признаке Поверхностная плотность, г/м2
0 выбросов в признаке Модуль упругости при растяжении, ГПа
0 выбросов в признаке Прочность при растяжении, МПа
3 выбросов в признаке Потребление смолы, г/м2
0 выбросов в признаке Угол нашивки
0 выбросов в признаке Шаг нашивки
7 выбросов в признаке Плотность нашивки
Всего 24 выброса
```

```
In [48]: #Создадим переменную со списком всех параметров, в которых есть выбросы
df.columns
column_list_drop = ["Соотношение матрица-наполнитель",
```

```
"Плотность, кг/м3",
"модуль упругости, ГПа",
"Количество отвердителя, м.%",
"Содержание эпоксидных групп,%_2",
"Температура вспышки, С_2",
"Поверхностная плотность, г/м2",
"Модуль упругости при растяжении, ГПа",
"Прочность при растяжении, МПа",
"Потребление смолы, г/м2",
"Шаг нашивки",
"Плотность нашивки"]
```

```
In [49]: # Исключим выбросы, очистим данные от выбросов методом межквартильного расстояния
# Выбор сделан в пользу этого метода, потому что хотим добиться в данной работе
for i in column_list_drop:
    q75, q25 = np.percentile(df.loc[:,i], [75,25])
    intr_qr = q75 - q25
    max = q75 + (1.5 * intr_qr)
    min = q25 - (1.5 * intr_qr)
    df.loc[df[i] < min, i] = np.nan
    df.loc[df[i] > max, i] = np.nan
```

```
In [50]: #Посмотрим на сумму выбросов по каждому из столбцов
df.isnull().sum()
#Всего 64 выброса, можно их удалить.
```

```
Out[50]: 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                    8
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 [51]: #Удаляем строки с выбросами
df = df.dropna(axis=0)
```

```
In [52]: #И еще раз посмотрим на сумму выбросов по каждому из столбцов, чтобы убедиться
df.isnull().sum()
```

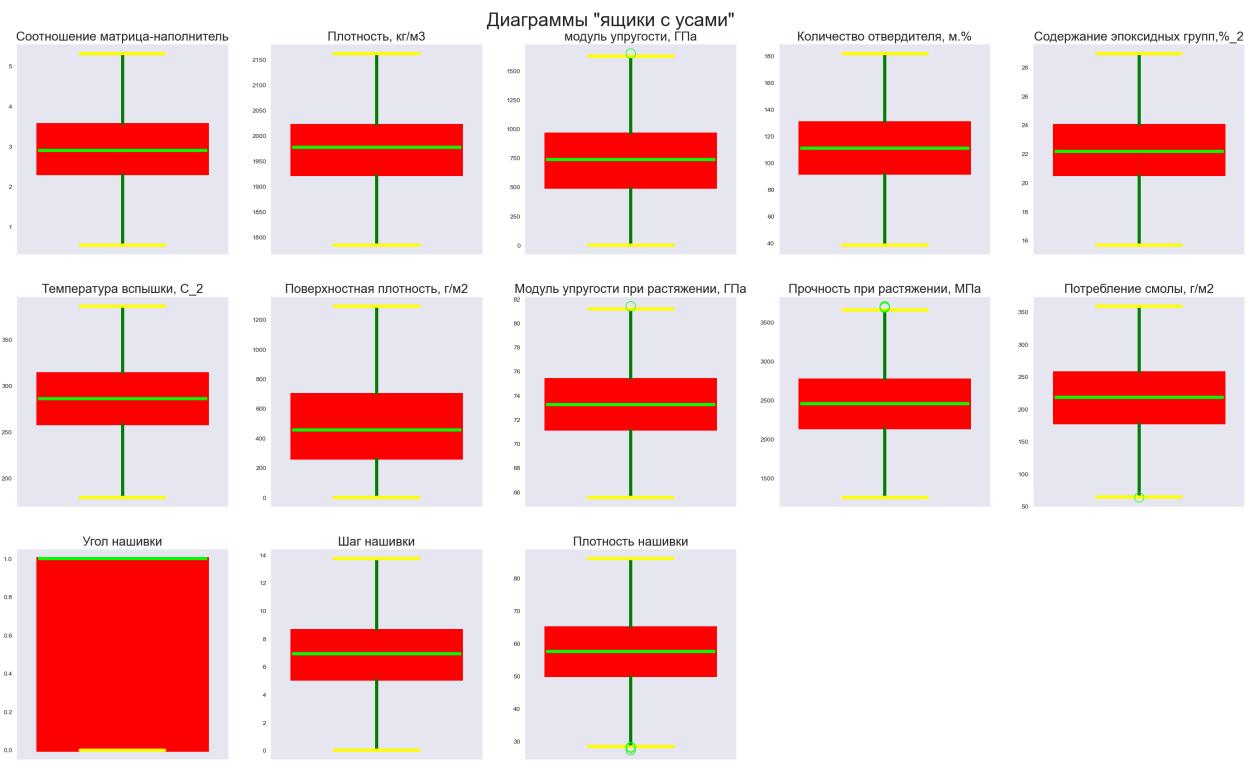
```
Out[52]: Соотношение матрица-наполнитель      0
          Плотность, кг/м3                      0
          модуль упругости, ГПа                  0
          Количество отвердителя, м.%            0
          Содержание эпоксидных групп,%_2        0
          Температура вспышки, С_2                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 [53]: df.info()
```

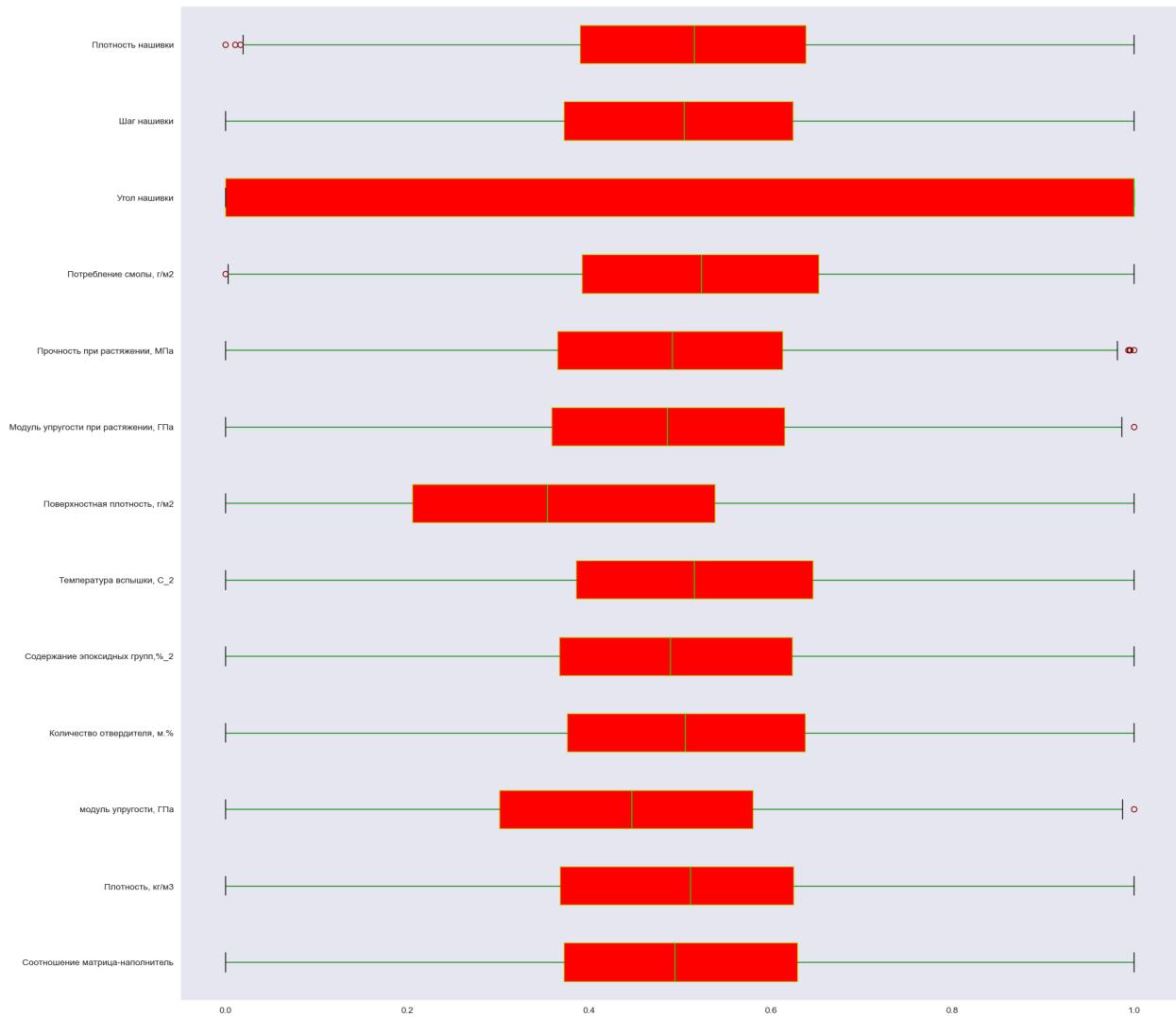
```
<class 'pandas.core.frame.DataFrame'>
Index: 936 entries, 1 to 1022
Data columns (total 13 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   Соотношение матрица-наполнитель    936 non-null   float64 
 1   Плотность, кг/м3                   936 non-null   float64 
 2   модуль упругости, 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   int64  
 11  Шаг нашивки                     936 non-null   float64 
 12  Плотность нашивки              936 non-null   float64 
dtypes: float64(12), int64(1)
memory usage: 102.4 KB
```

```
In [54]: # 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.figure(figsize=(7,5))
    sns.boxplot(data = df, y = df[col], fliersize = 15, linewidth = 5, boxprops
    plt.ylabel(None)
    plt.title(col, size = 20)
    #plt.show()
    c += 1
```



```
In [55]: scaler = MinMaxScaler()
scaler.fit(df)
plt.figure(figsize = (20, 20))
#Посмотрим на "ящики с усами", чтобы наглядно увидеть, что выбросов нет, но он
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels = df.columns, patch_artist=True)
plt.show()
```



```
In [56]: # На графиках выше мы видим выбросы в некоторых столбцах. Они все ещё есть, по

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

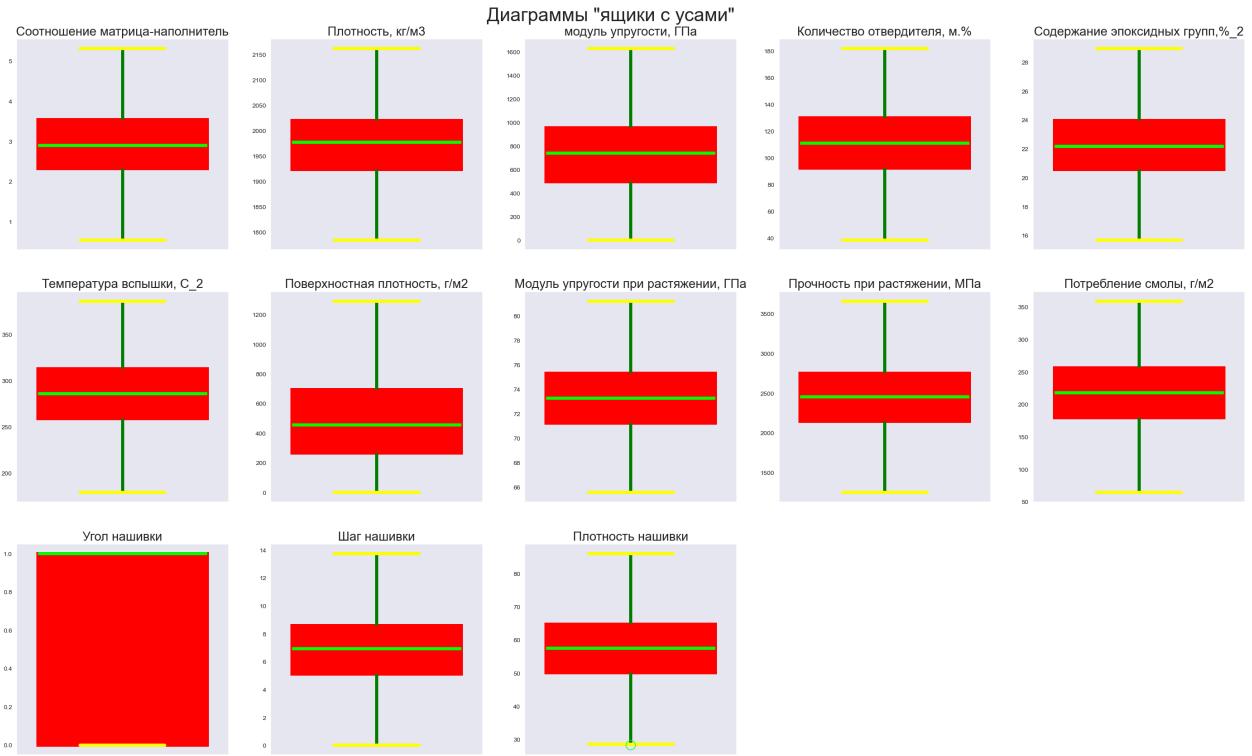
```
In [57]: # Ящики с усами (второй вариант)
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # инициализация plot counter

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.figure(figsize=(7,5))
sns.boxplot(data = df, y = df[col], fliersize = 15, linewidth = 5, boxprops = dict(facecolor='red'), medianprops = dict(color='green'), whiskerprops = dict(color='darkblue'), capprops = dict(color='darkblue'), notch = True)
plt.ylabel(None)
plt.title(col, size = 20)
#plt.show()
c += 1

```



In [58]: `#Проверим сумму выбросов по каждому из столбцов
df.isnull().sum()`

Out[58]:

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
Потребление смолы, г/м2	1
Угол нашивки	0
Шаг нашивки	0
Плотность нашивки	3

`dtype: int64`

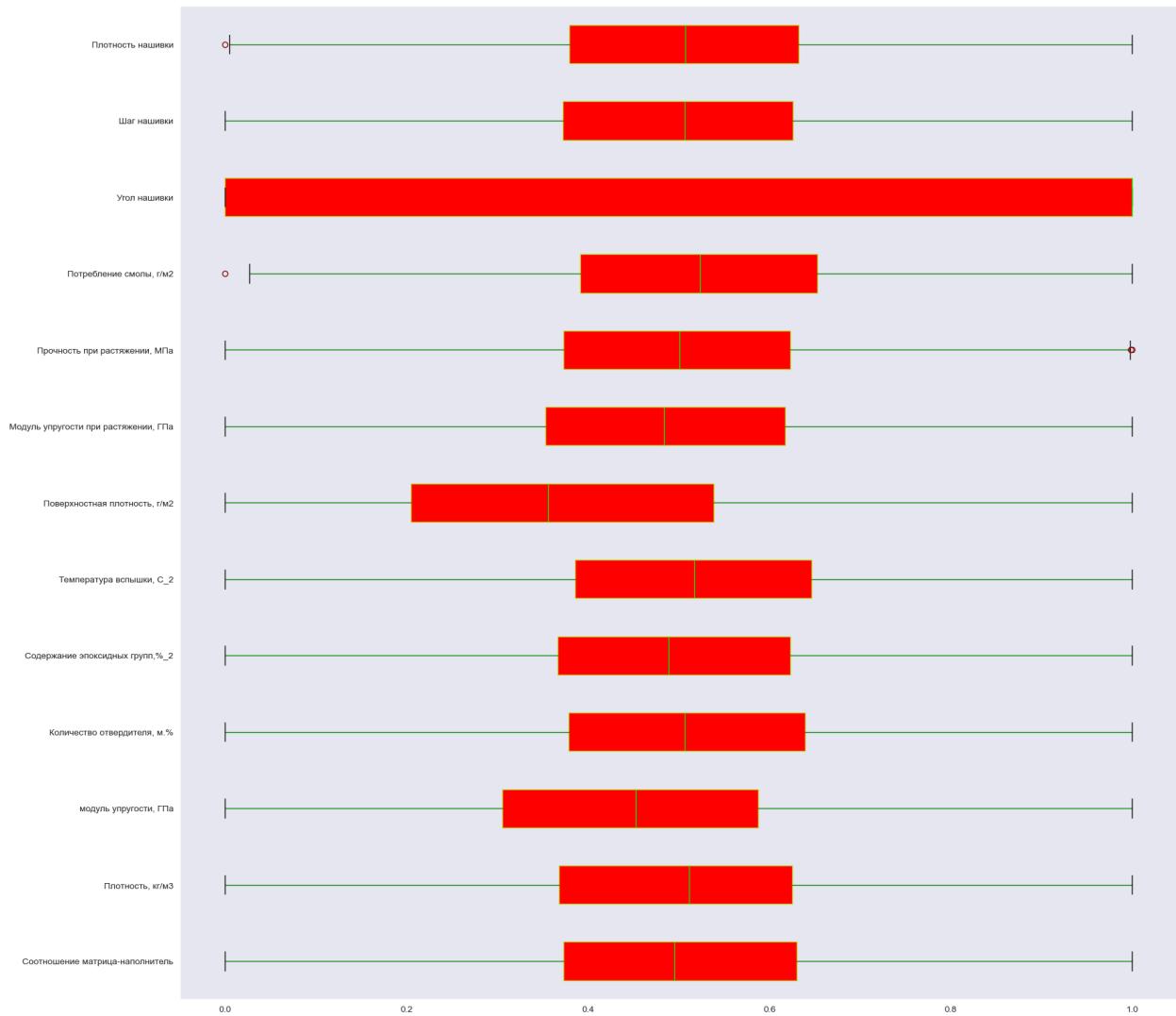
In [59]: `df = df.dropna(axis=0)
df.isnull().sum()`

```
Out[59]: 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 [60]: df.info()
```

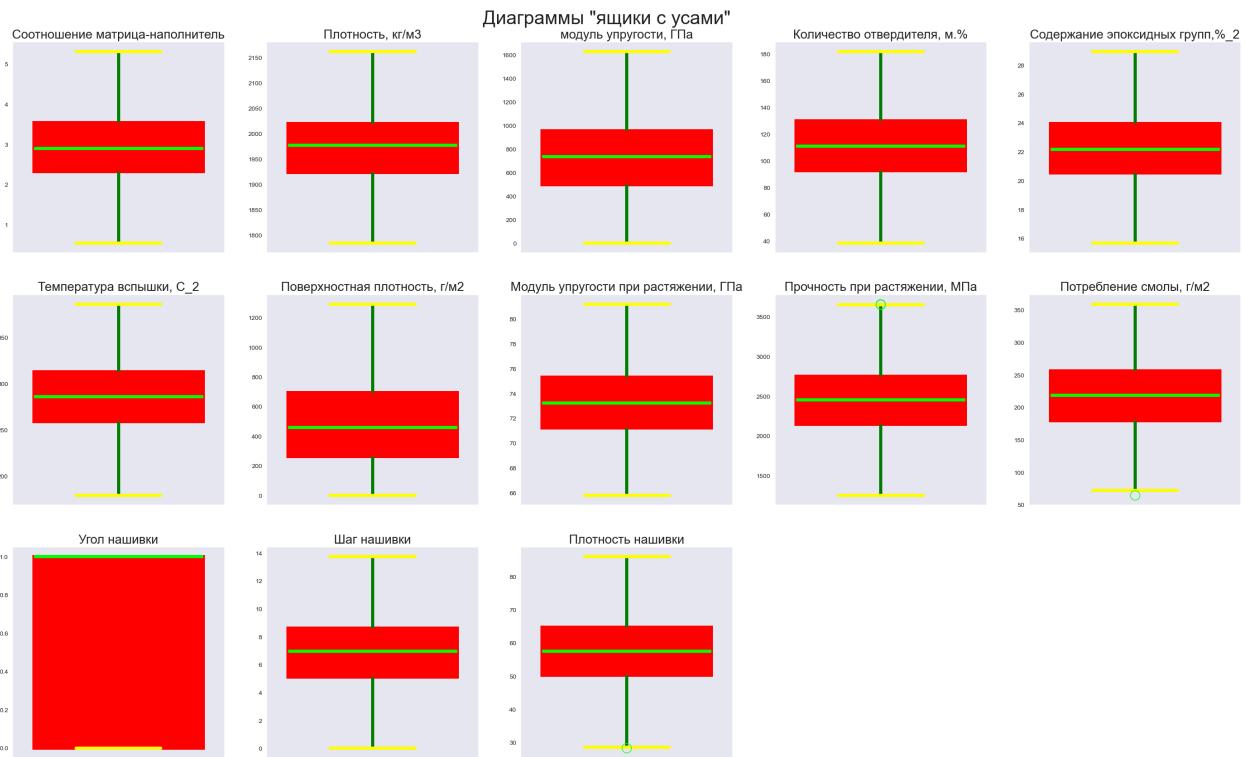
```
<class 'pandas.core.frame.DataFrame'>  
Index: 926 entries, 1 to 1022  
Data columns (total 13 columns):  
 #   Column           Non-Null Count  Dtype    
---  --    
 0   Соотношение матрица-наполнитель    926 non-null   float64  
 1   Плотность, кг/м3                   926 non-null   float64  
 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   Потребление смолы, г/м2            926 non-null   float64  
 10  Угол нашивки                     926 non-null   int64  
 11  Шаг нашивки                      926 non-null   float64  
 12  Плотность нашивки                926 non-null   float64  
dtypes: float64(12), int64(1)  
memory usage: 101.3 KB
```

```
In [61]: #В третий раз построим на "ящики с усами"  
scaler = MinMaxScaler()  
scaler.fit(df)  
plt.figure(figsize = (20, 20))  
#Выводим "ящики"  
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels = df.columns, patch_artist=True)  
plt.show()
```



```
In [62]: # 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.figure(figsize=(7,5))
    sns.boxplot(data = df, y = df[col], fliersize = 15, linewidth = 5, boxprops = {'color': 'red'}, whiskerprops = {'color': 'green'}, medianprops = {'color': 'green'}, capprops = {'color': 'green'})
    plt.ylabel(None)
    plt.title(col, size = 20)
    #plt.show()
    c += 1
```



```
In [63]: # Повторяем процедуру с выбросами еще раз.
q75, q25 = np.percentile(df.loc[:,i],[75, 25])
intr_qr = q75 - q25
max = q75 + (1.5*intr_qr)
min = q25 - (1.5*intr_qr)
df.loc[df[i] < min,i] = np.nan
df.loc[df[i] > max,i] = np.nan
```

```
Cell In[63], line 2
    q75, q25 = np.percentile(df.loc[:,i],[75, 25])
 ^
IndentationError: unexpected indent
```

```
In [64]: #Еще раз проверим сумму выбросов в каждом столбце
df.isnull().sum()
```

```
Out[64]: 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 [65]: #Еще раз удаляем строки с выбросами  
df = df.dropna(axis=0)
```

```
In [66]: #И проверочно посмотрим сумму выбросов  
df.isnull().sum()
```

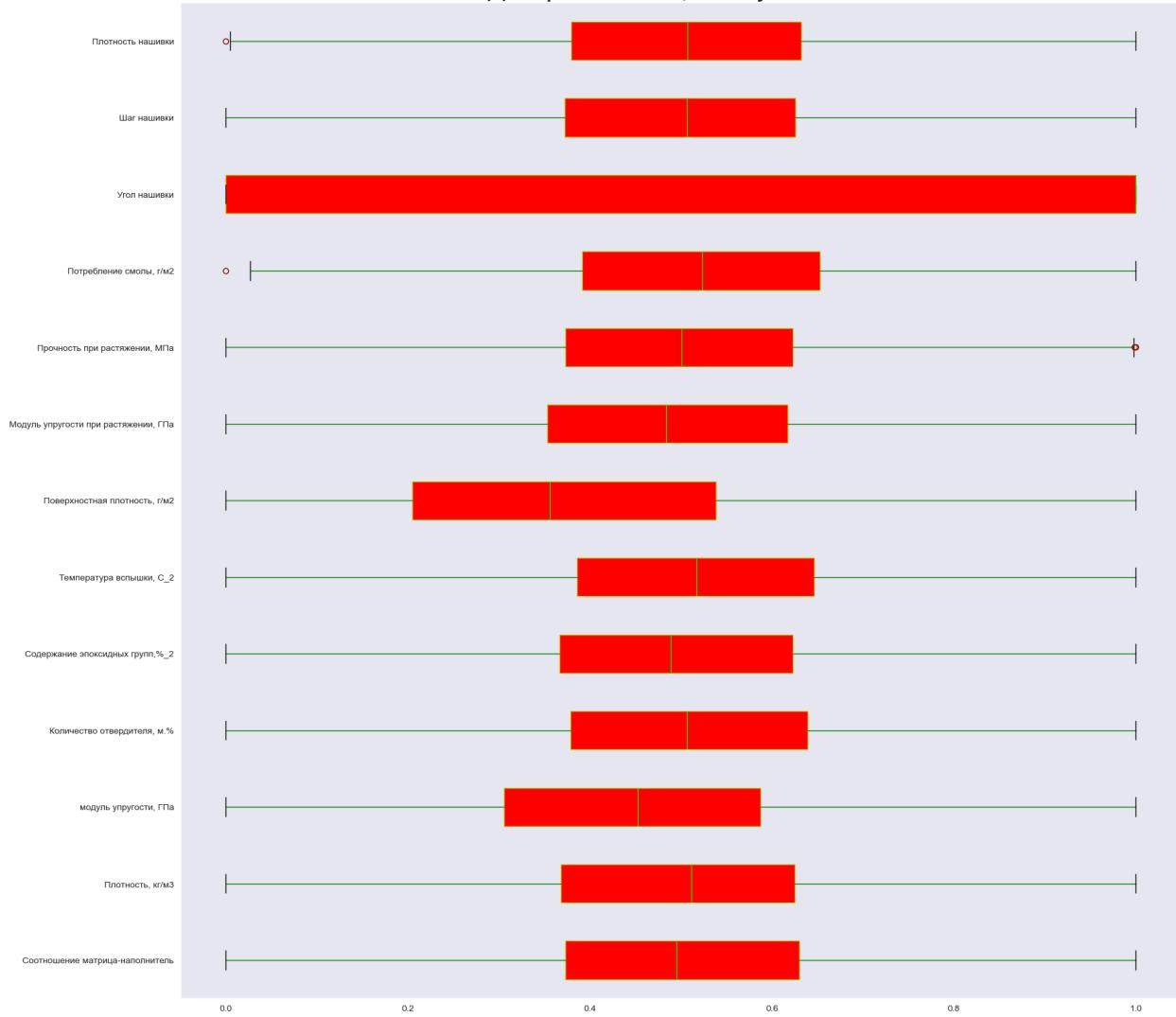
```
Out[66]: Соотношение матрица-наполнитель          0  
Плотность, кг/м3           0  
модуль упругости, ГПа      0  
Количество отвердителя, м.% 0  
Содержание эпоксидных групп,%_2    0  
Температура вспышки, С_2        0  
Поверхностная плотность, г/м2     0  
Модуль упругости при растяжении, ГПа 0  
Прочность при растяжении, МПа      0  
Потребление смолы, г/м2         0  
Угол нашивки                  0  
Шаг нашивки                   0  
Плотность нашивки            0  
dtype: int64
```

```
In [67]: #Просмотрим на чистый датасет  
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 926 entries, 1 to 1022  
Data 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   Потребление смолы, г/м2         926 non-null   float64  
 10  Угол нашивки                  926 non-null   int64  
 11  Шаг нашивки                   926 non-null   float64  
 12  Плотность нашивки            926 non-null   float64  
dtypes: float64(12), int64(1)  
memory usage: 101.3 KB
```

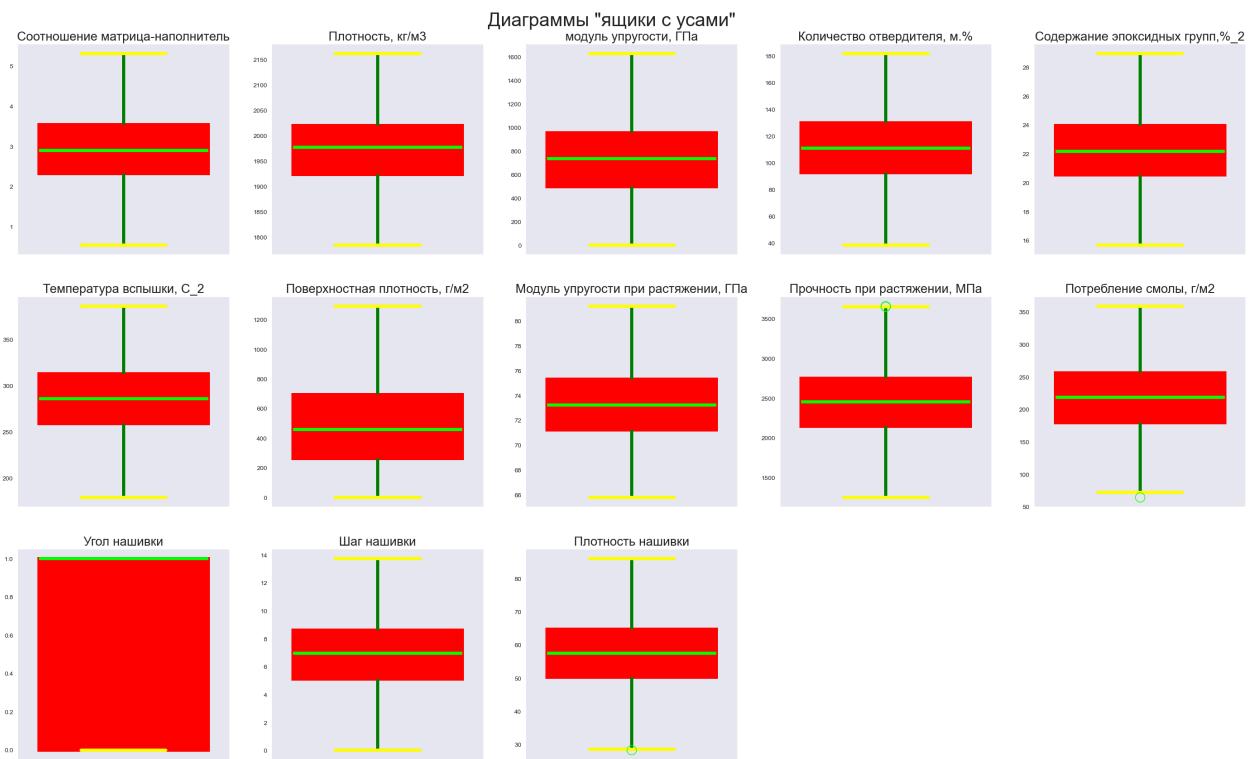
```
In [68]: # "Ящики с усами"(боксплоты) (первый вариант)  
scaler = MinMaxScaler()  
scaler.fit(df)  
plt.figure(figsize = (20, 20))  
plt.suptitle('Диаграммы "ящики с усами"', y = 0.9 ,  
             fontsize = 30)  
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels = df.columns,patch_arti  
plt.show()
```

Диаграммы "ящики с усами"

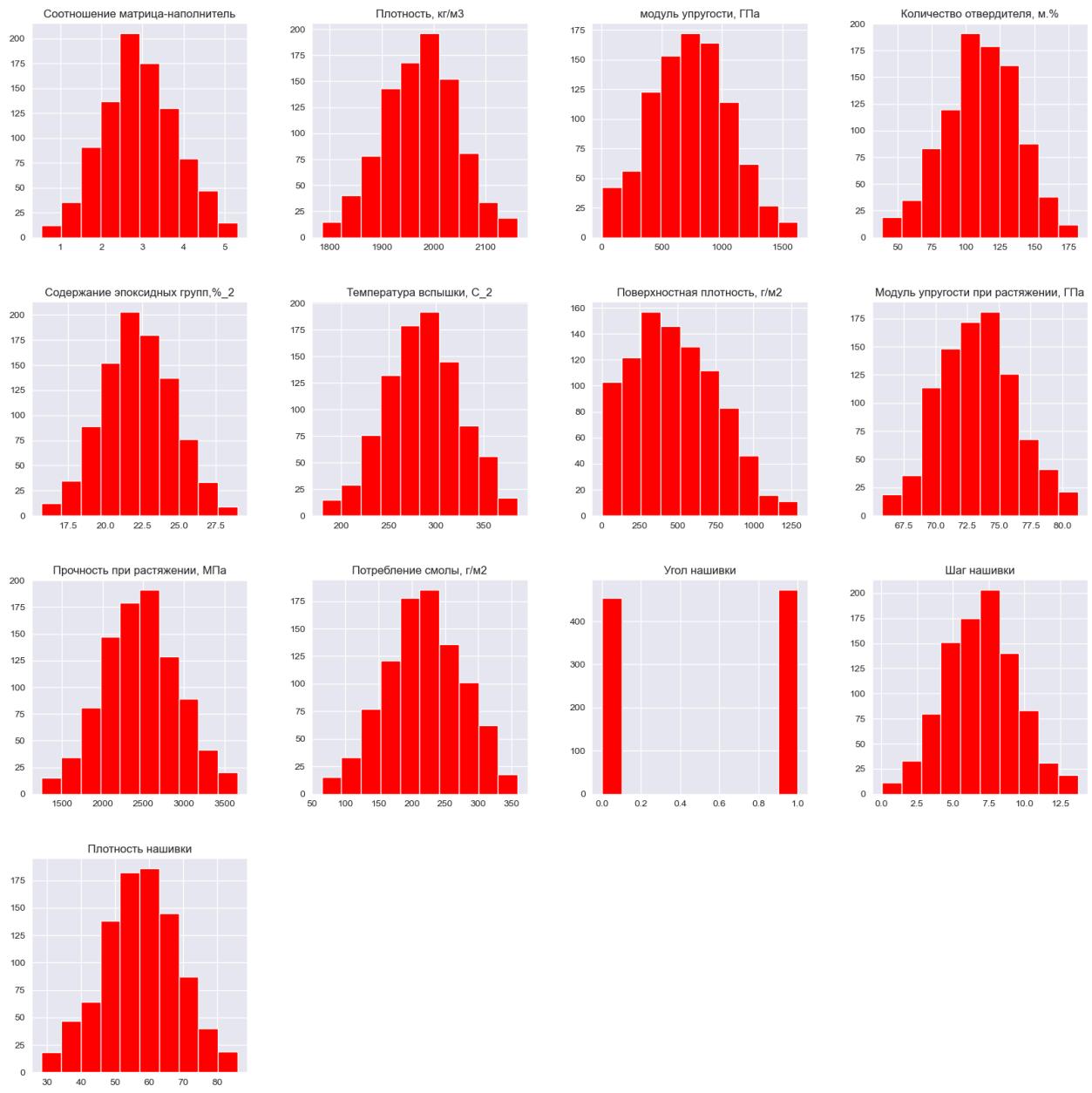


```
In [69]: # Ящики с усами (второй вариант)
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # инициализация plot counter

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.figure(figsize=(7,5))
    sns.boxplot(data = df, y = df[col], fliersize = 15, linewidth = 5, boxprops = dict(facecolor='red'))
    plt.ylabel(None)
    plt.title(col, size = 20)
    #plt.show()
    c += 1
```

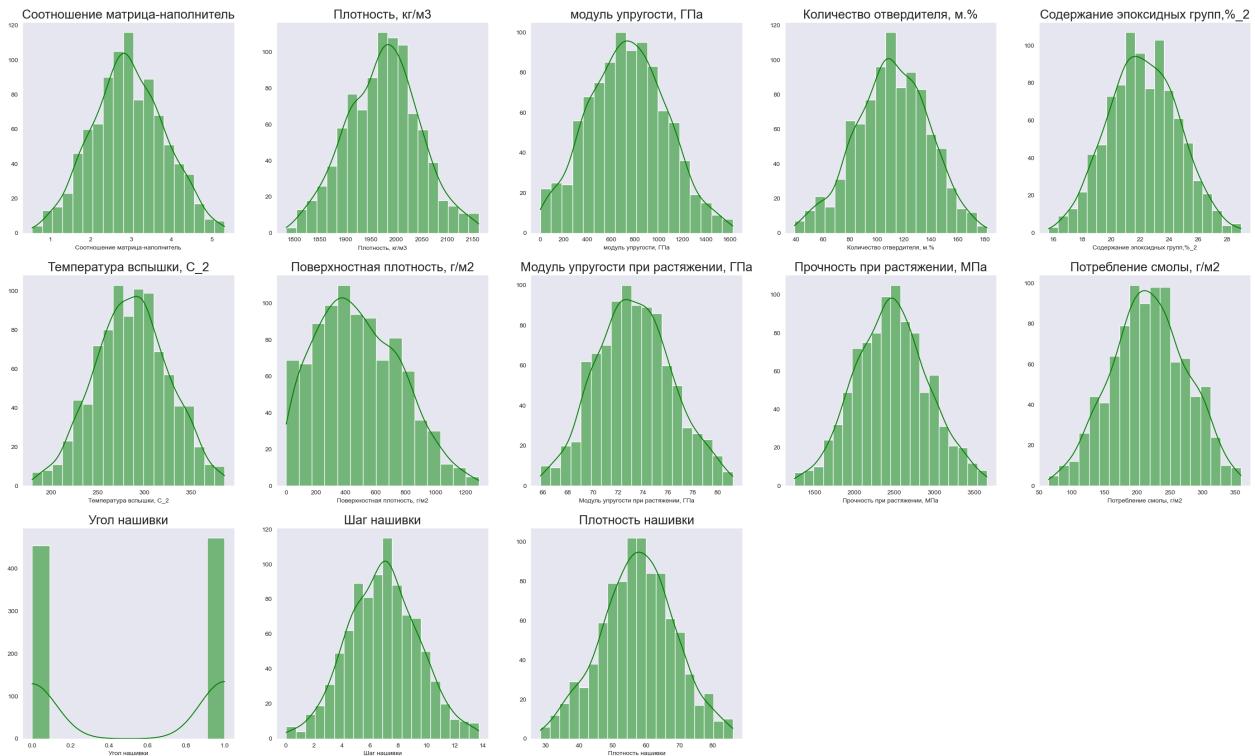


```
In [70]: # Построим гистограммы распределения каждой из переменных без нормализации
df.hist(figsize = (20,20), color = "r")
plt.show()
```



```
In [71]: # Гистограмма распределения (второй вариант)
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # инициализация plot counter
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 = "green")
    plt.ylabel(None)
    plt.title(col, size = 20)
    #plt.show()
    c += 1
#Данные стремятся к нормальному распределению практически везде, кроме угла на
```

Гистограммы переменных

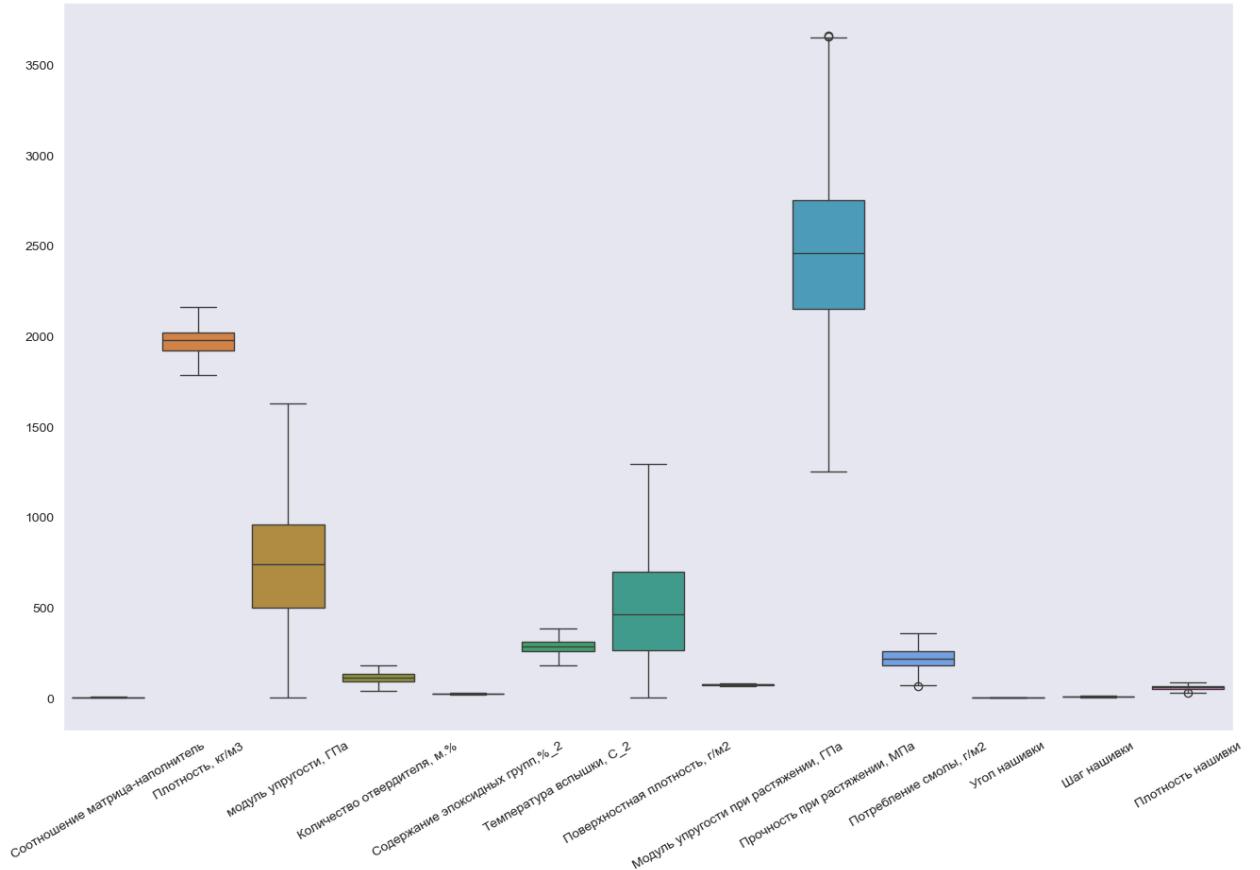


```
In [72]: # гистограмма распределения и боксплоты (третий вариант)
```

```
for column in df.columns:
    fig = px.histogram(df, x=column, color_discrete_sequence=['red'], nbins=10
    fig.show()
```

```
In [73]: for column in df.columns:
    fig = px.box(df, y=column)
    fig.show()
```

```
In [74]: # Ящики с усами на одном рисунке
plt.figure(figsize=(16,10))
ax = sns.boxplot(data=df)
ax.set_xticklabels(ax.get_xticklabels(), rotation=30);
```



```
In [75]: # поэтому выбросы проверим еще другими способами
for column_name in column_names:
    print(column_name)

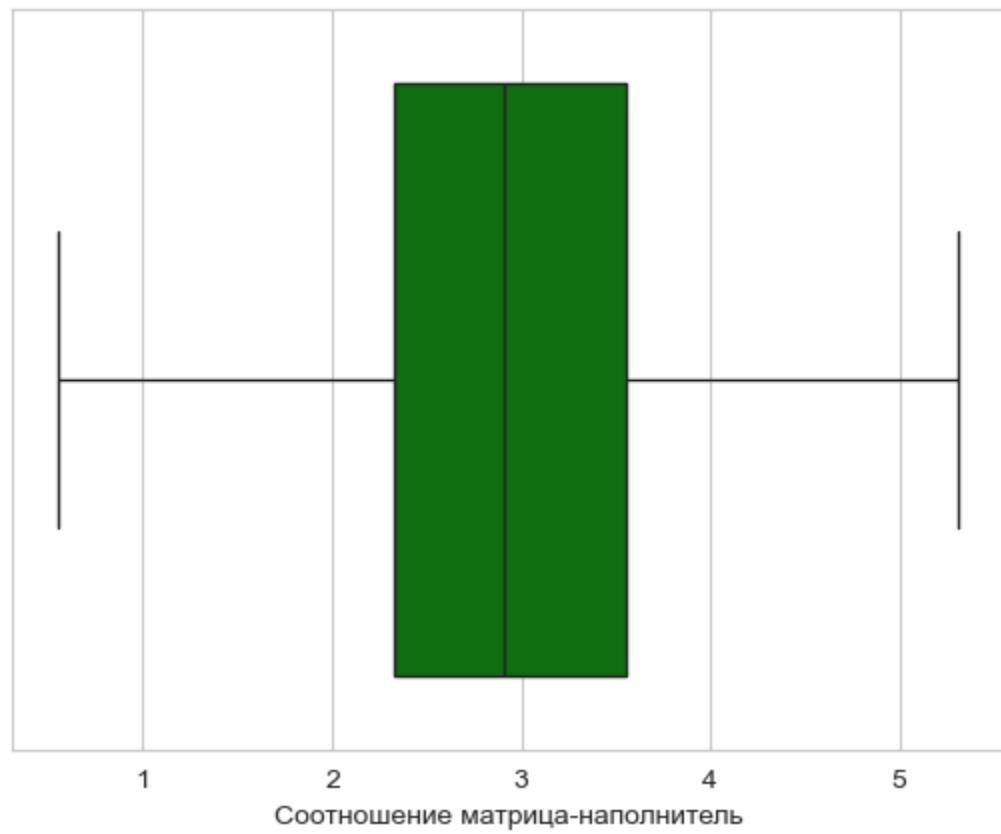
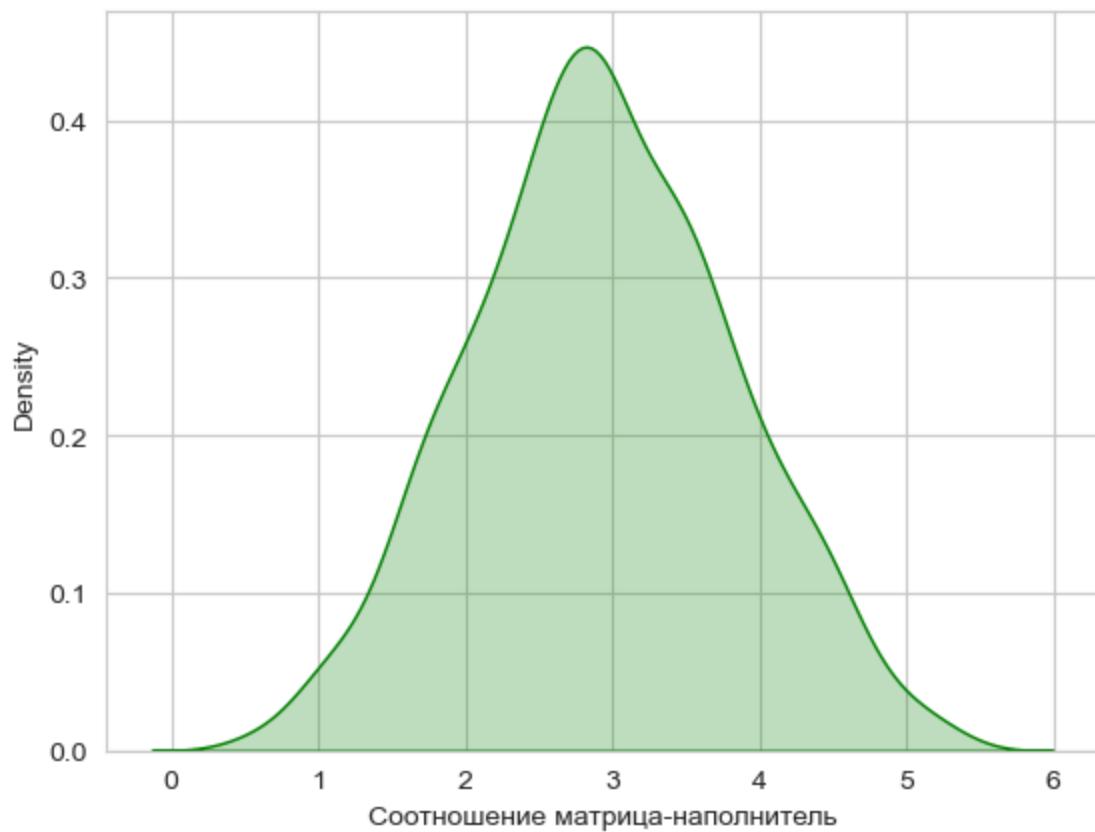
        #Гистограмма распределения
    gis = df[column_name]
    sns.set_style("whitegrid")
    sns.kdeplot(data = gis, shade = True, palette = 'colorblind', color = "g")
    plt.show()

        #Диаграмма "Ящик с усами"
    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")
```

Соотношение матрица-наполнитель



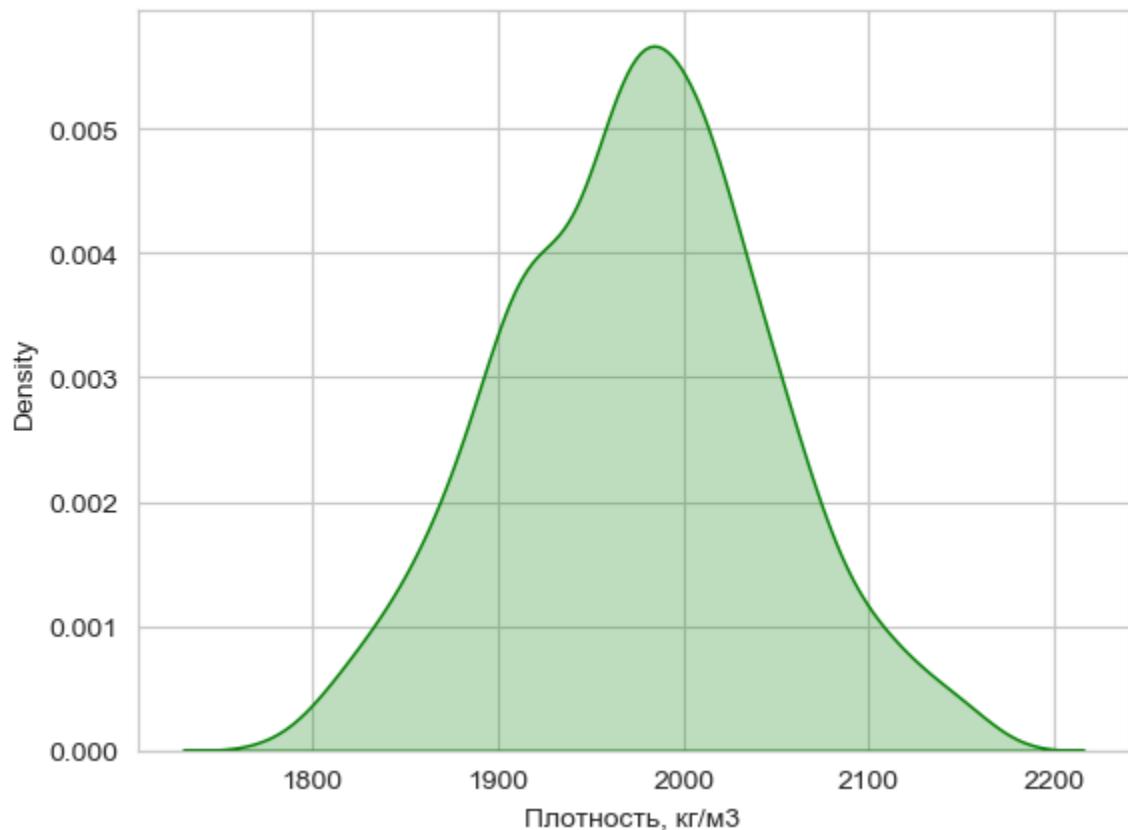
Минимальное значение: 0.547391007365624

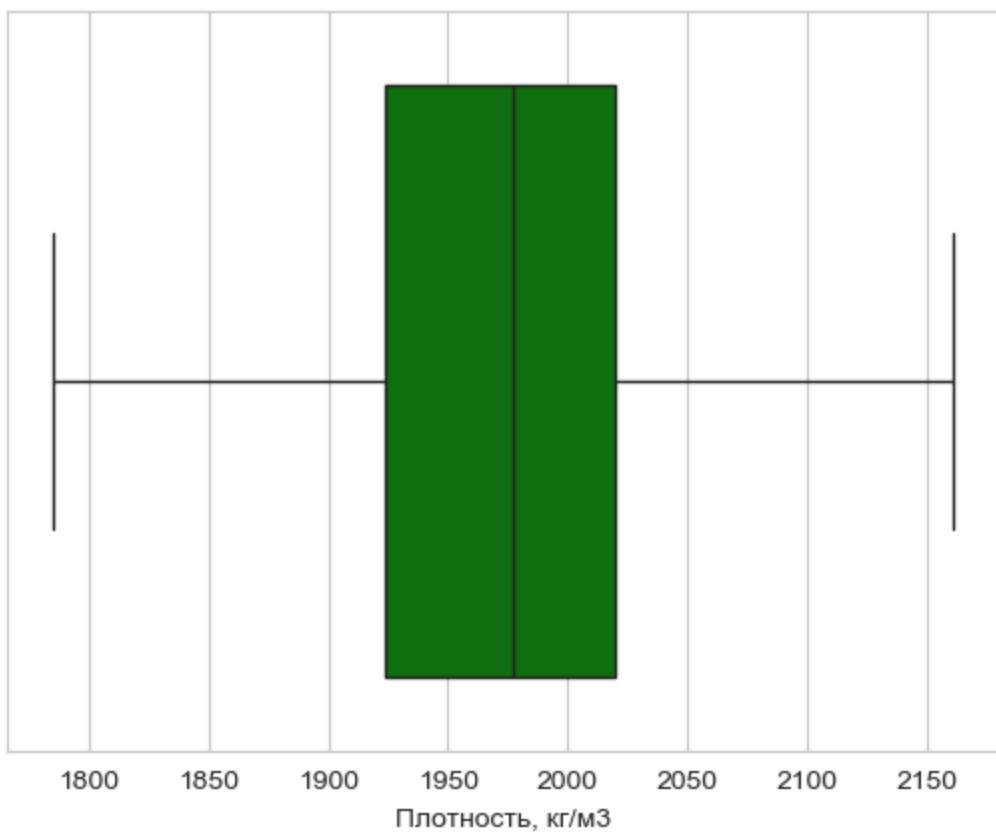
Максимальное значение: 5.3141436851035

Среднее значение: 2.9292736002637914

Медианное значение: 2.906630378786655

Плотность, кг/м3





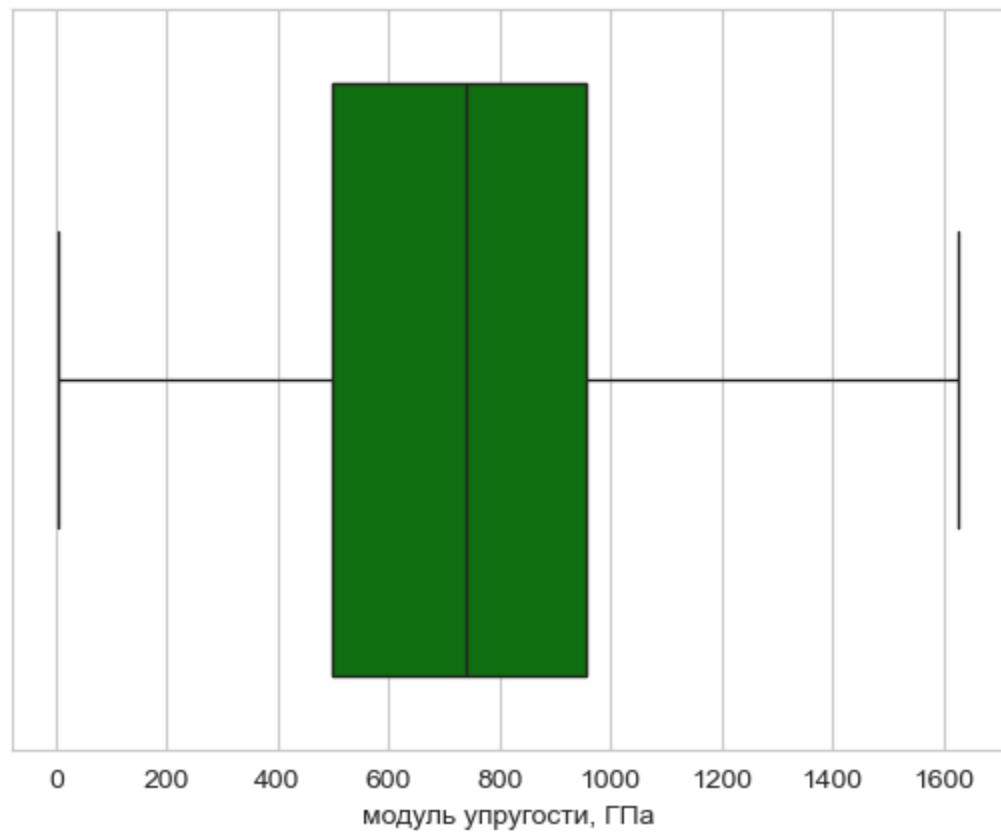
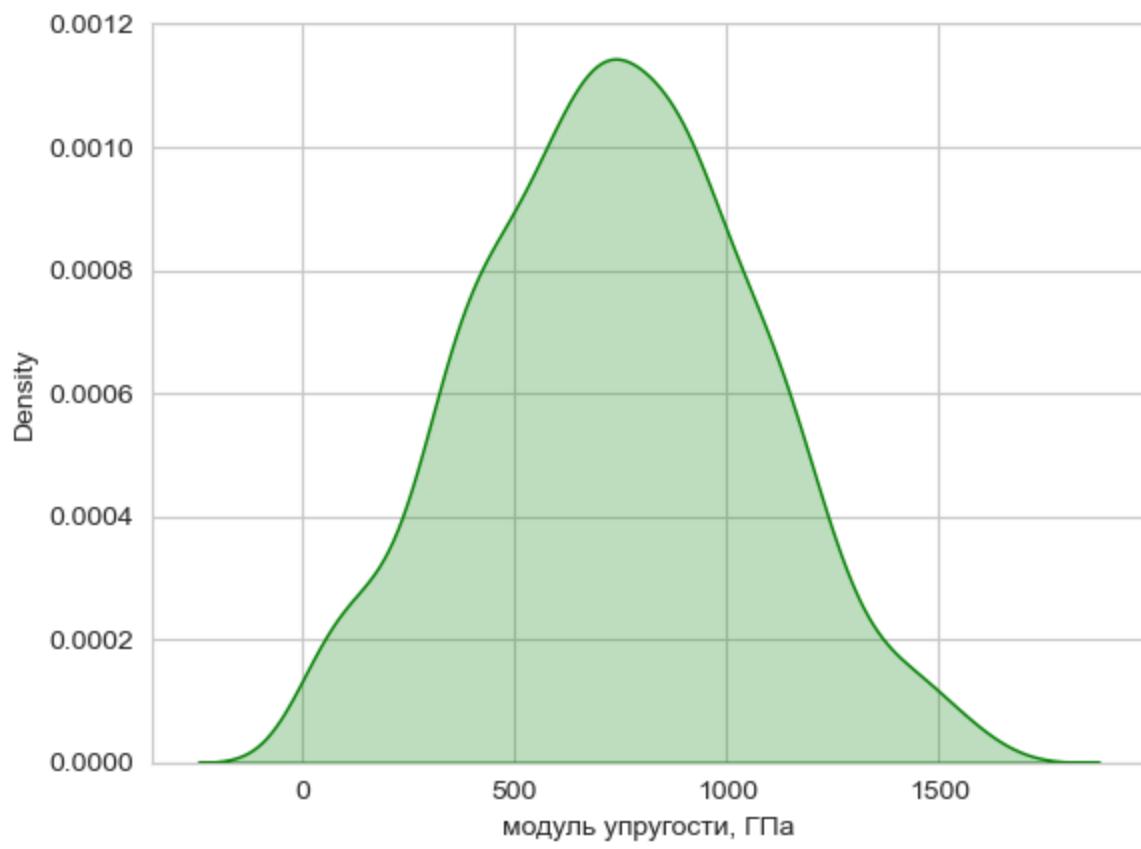
Минимальное значение: 1784.48224524858

Максимальное значение: 2161.56521646388

Среднее значение: 1974.1705357934077

Медианное значение: 1977.45068378571

модуль упругости, ГПа



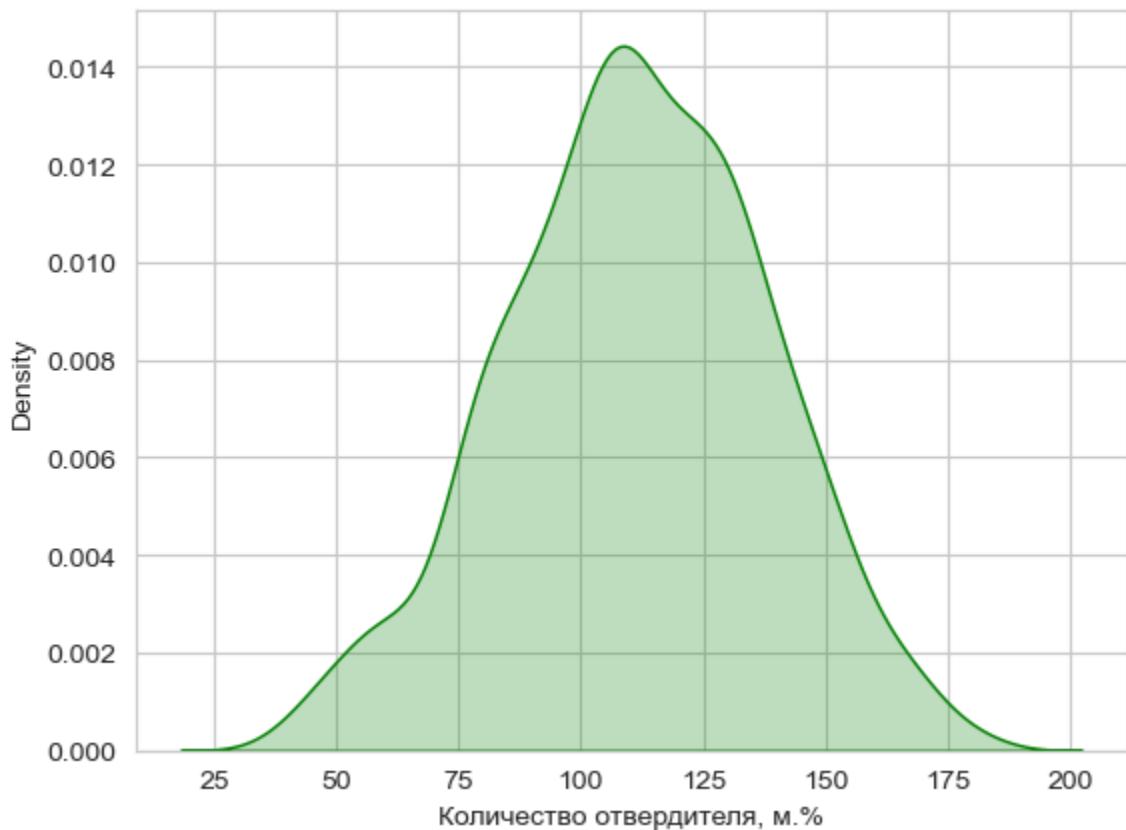
Минимальное значение: 2.4369087535075

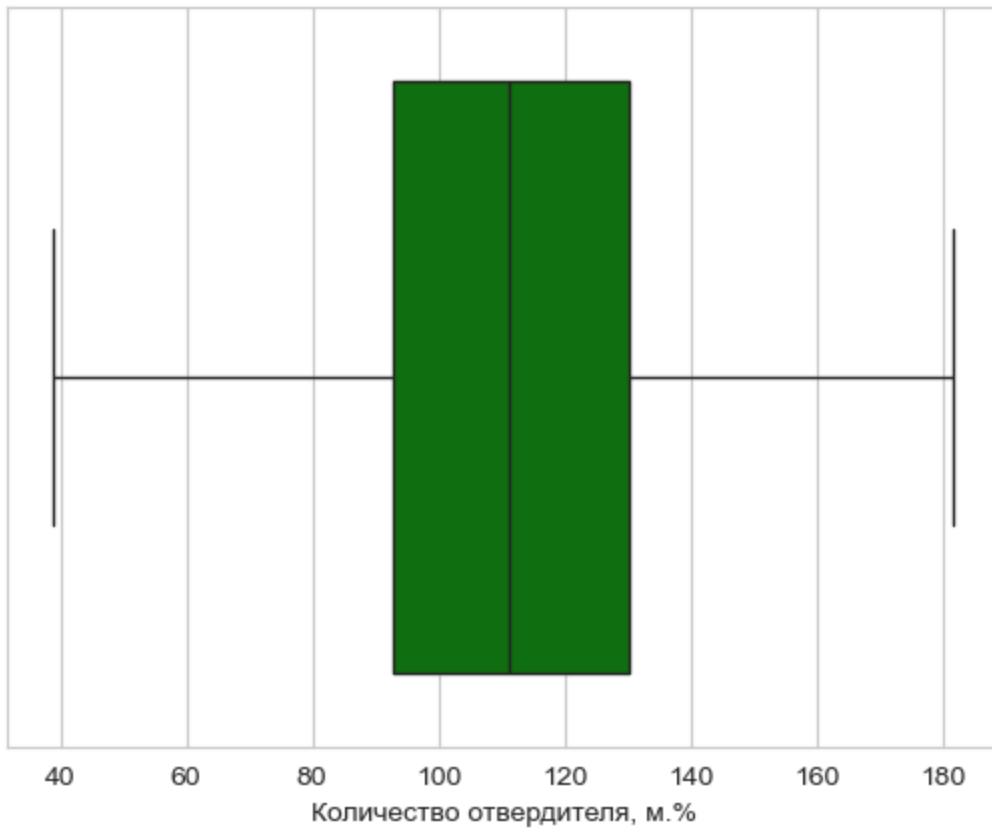
Максимальное значение: 1628.0

Среднее значение: 737.0096041105552

Медианное значение: 738.164863750196

Количество отвердителя, м.%





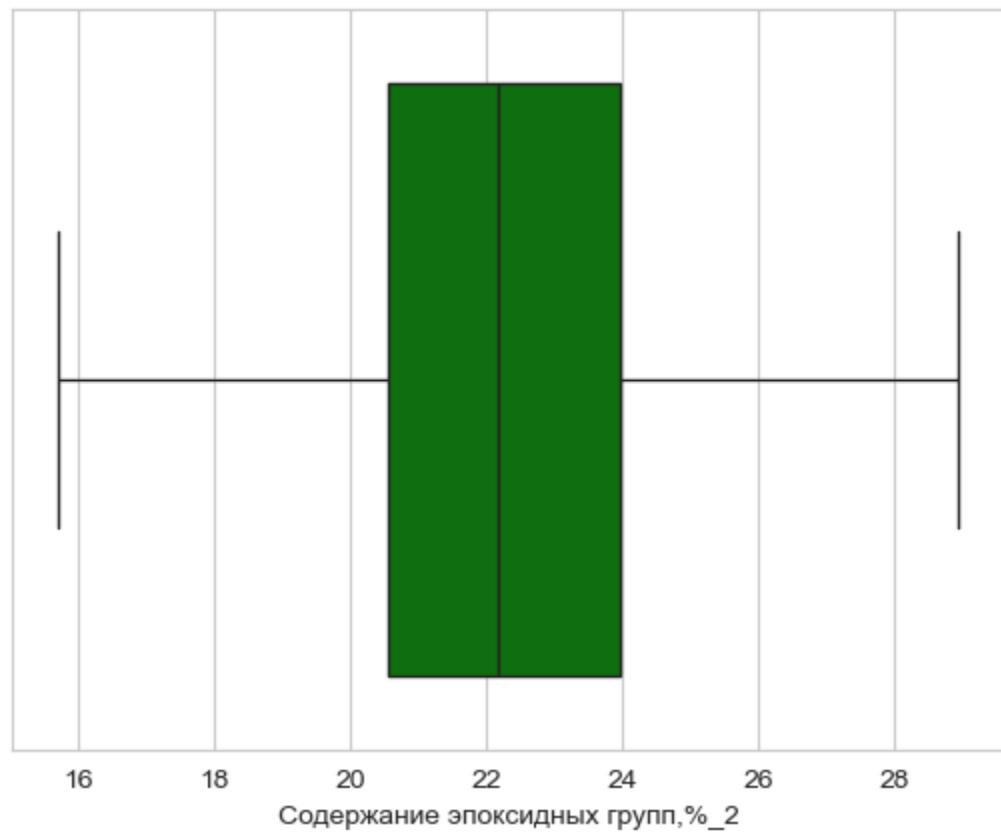
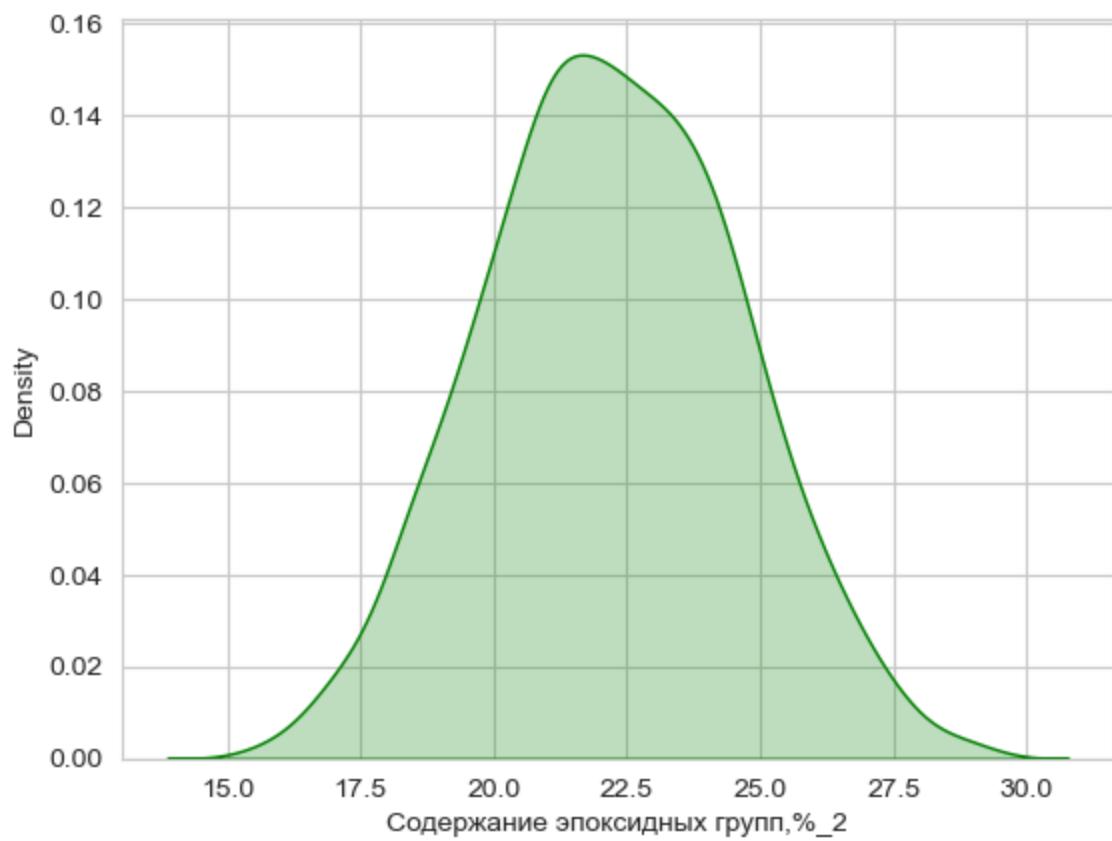
Минимальное значение: 38.6685003343557

Максимальное значение: 181.82844779488

Среднее значение: 111.1874278281093

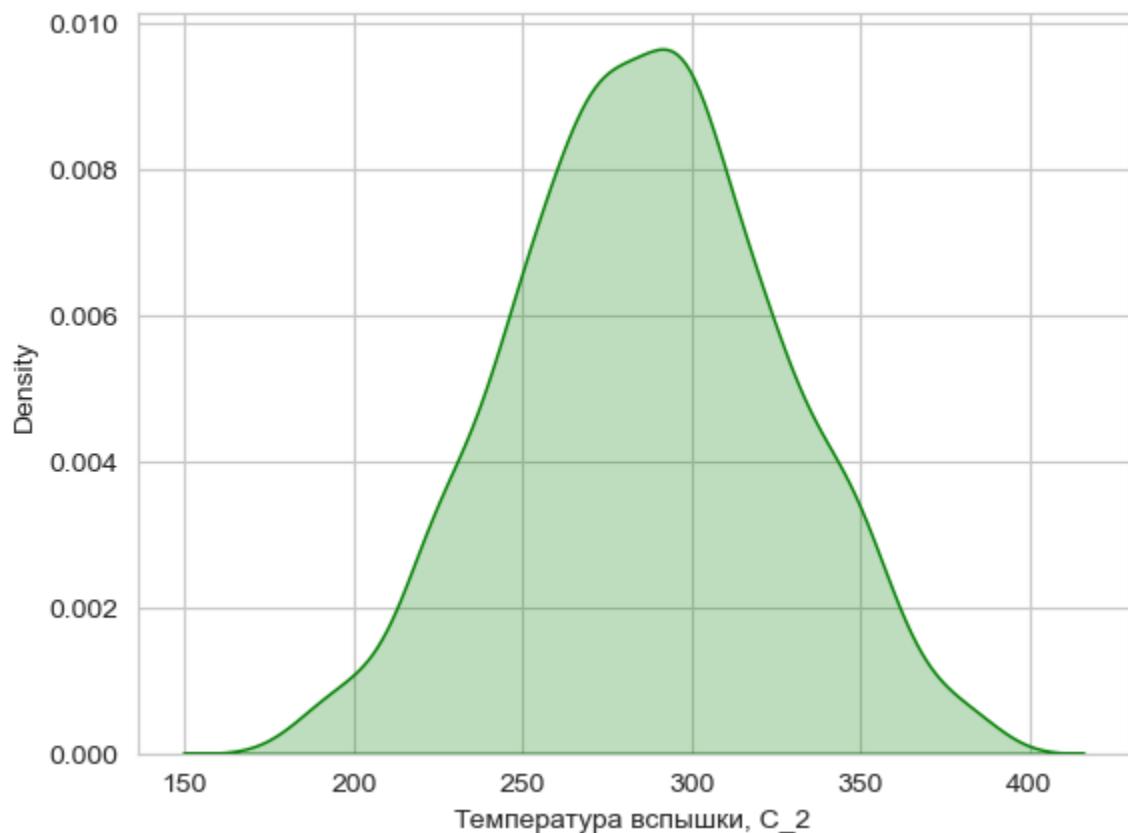
Медианное значение: 111.1836272889665

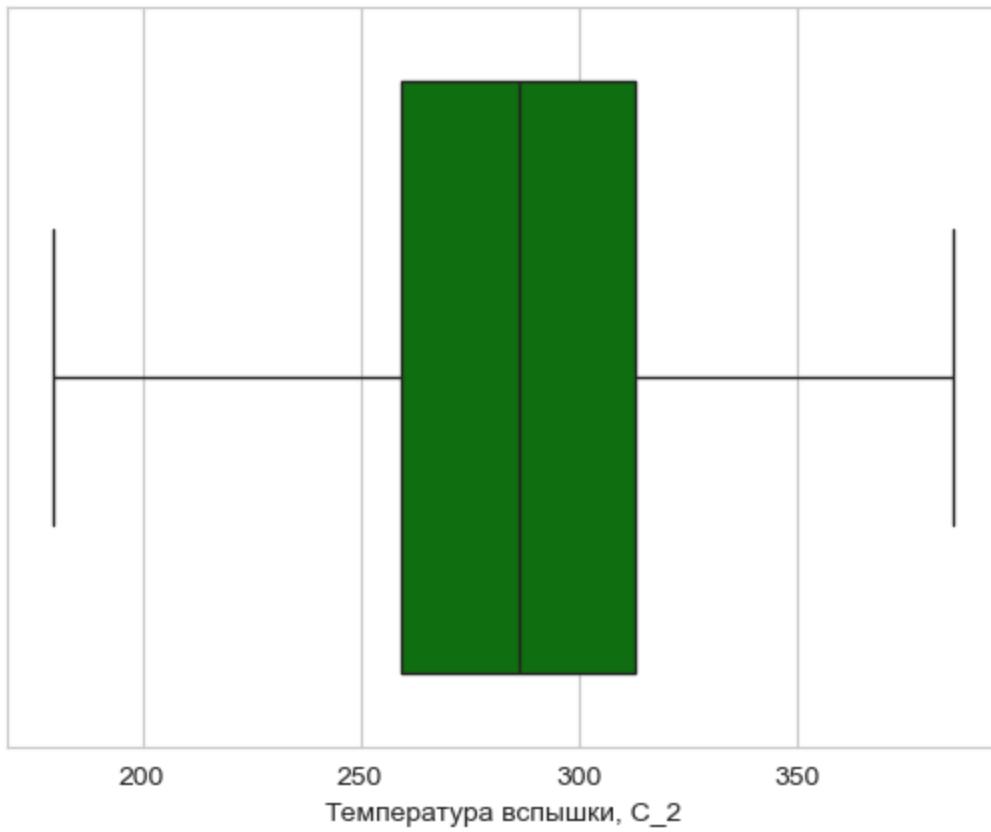
Содержание эпоксидных групп,%_2



Минимальное значение: 15.6958938036288
Максимальное значение: 28.9550943746499
Среднее значение: 22.20699936446046
Медианное значение: 22.1784714399792

Температура вспышки, С_2





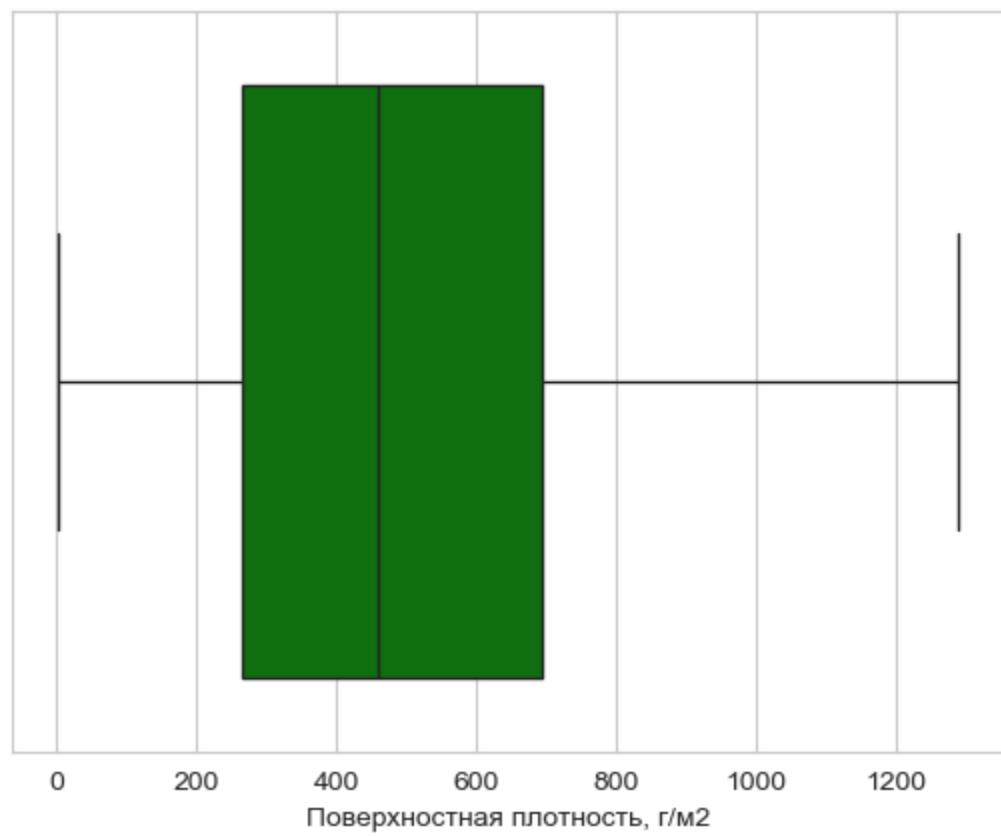
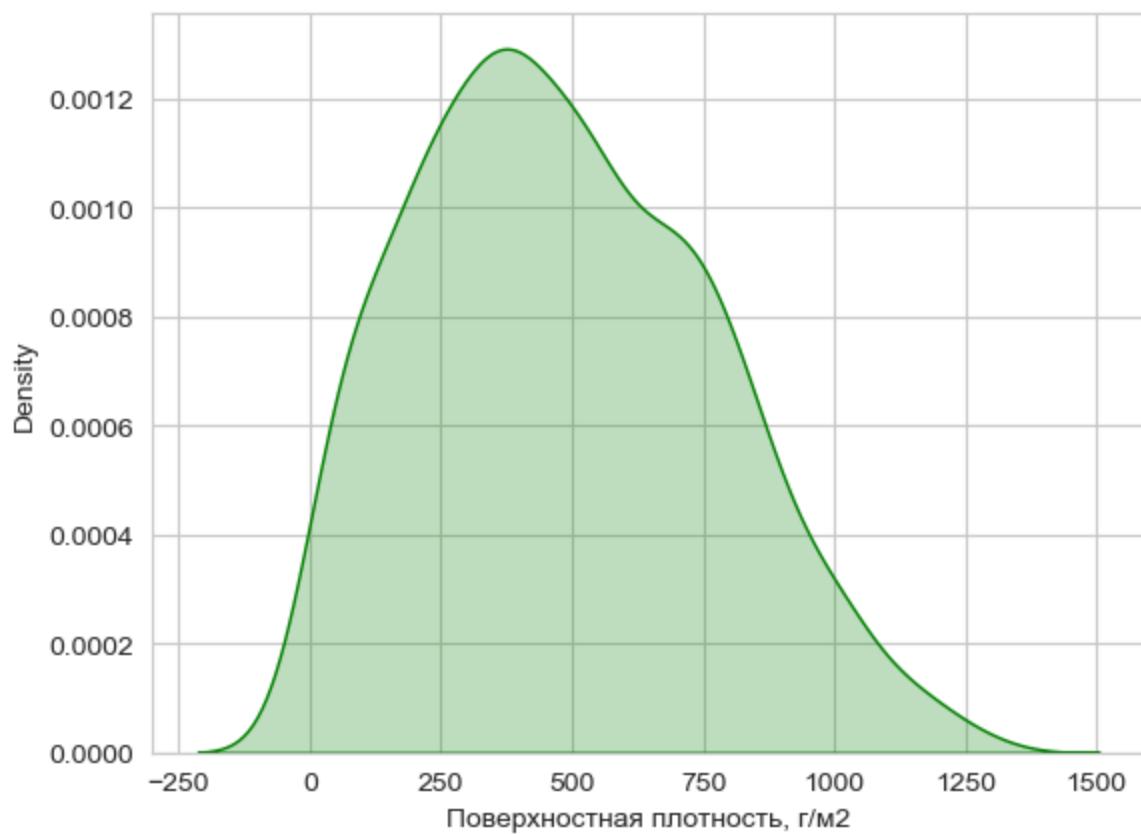
Минимальное значение: 179.37439137039

Максимальное значение: 386.067991779505

Среднее значение: 286.1198125639639

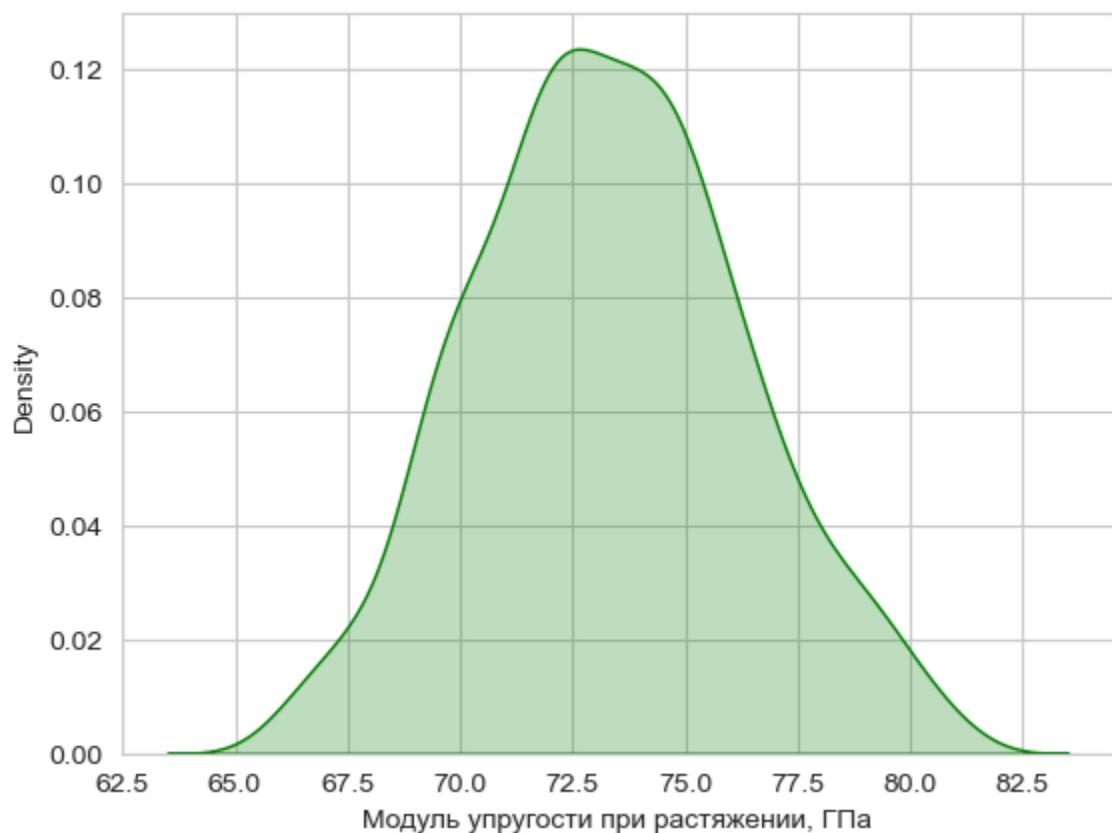
Медианное значение: 286.2207634450395

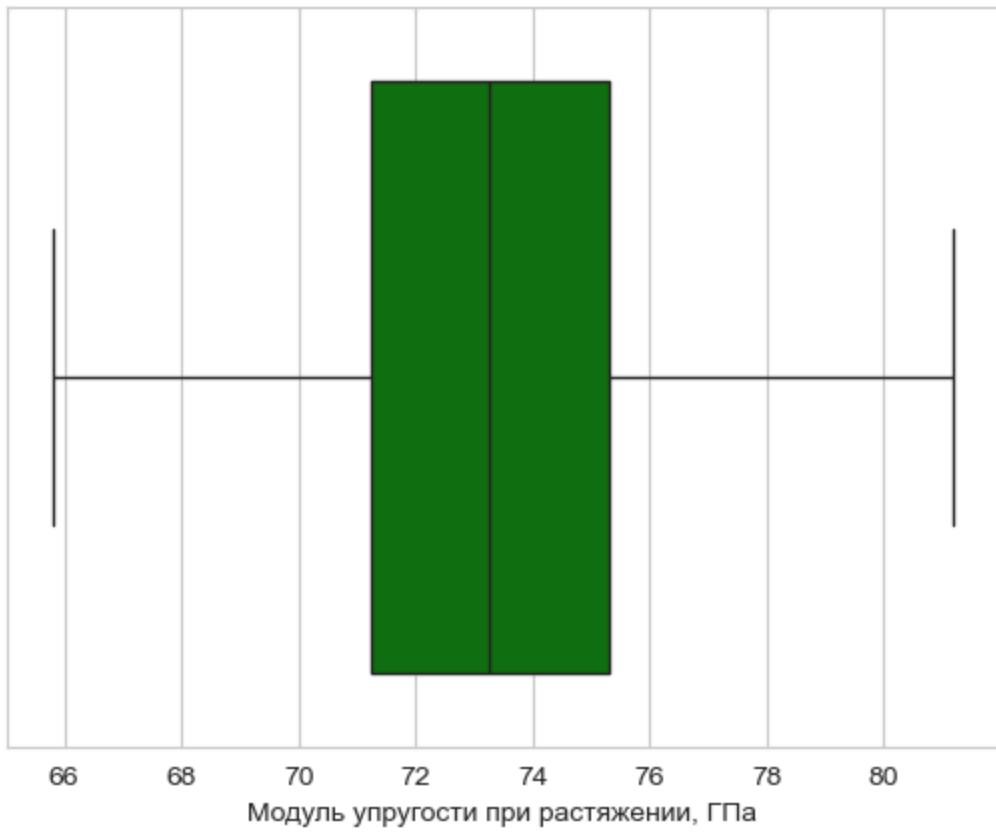
Поверхностная плотность, г/м²



Минимальное значение: 0.603739925153945
Максимальное значение: 1291.34011463545
Среднее значение: 482.8565301437093
Медианное значение: 460.33929415998796

Модуль упругости при растяжении, ГПа





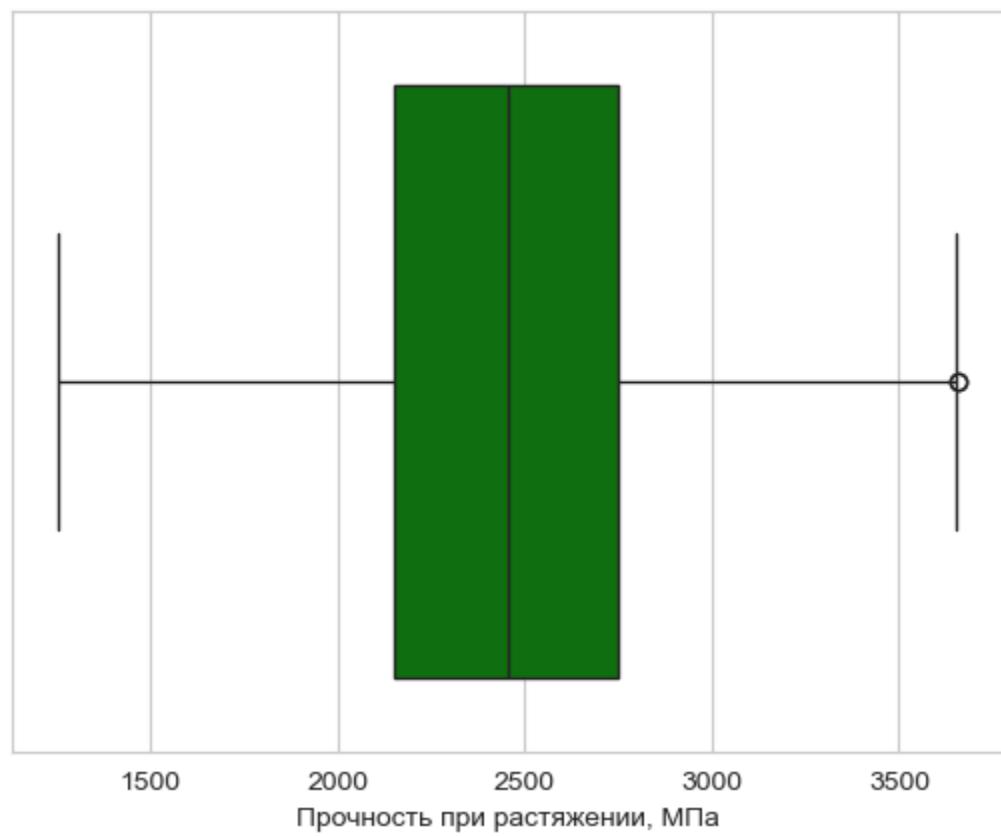
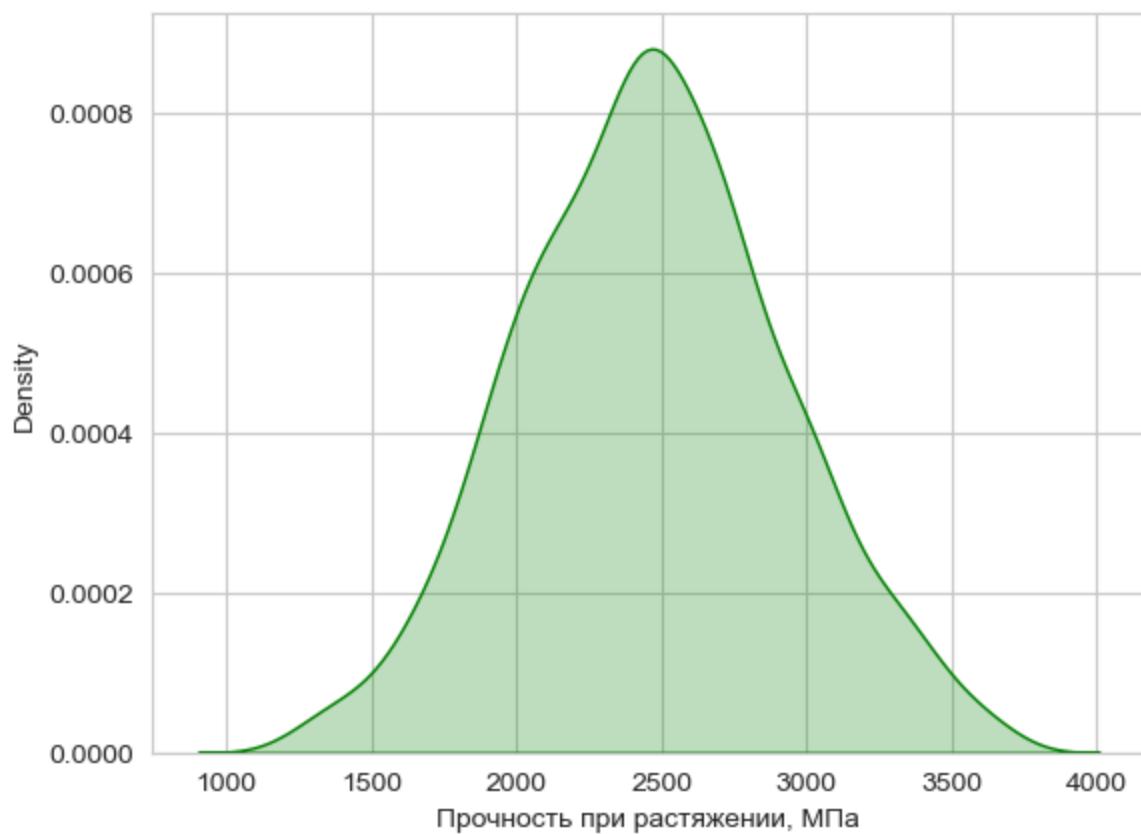
Минимальное значение: 65.7938449666054

Максимальное значение: 81.203146720828

Среднее значение: 73.30325783251774

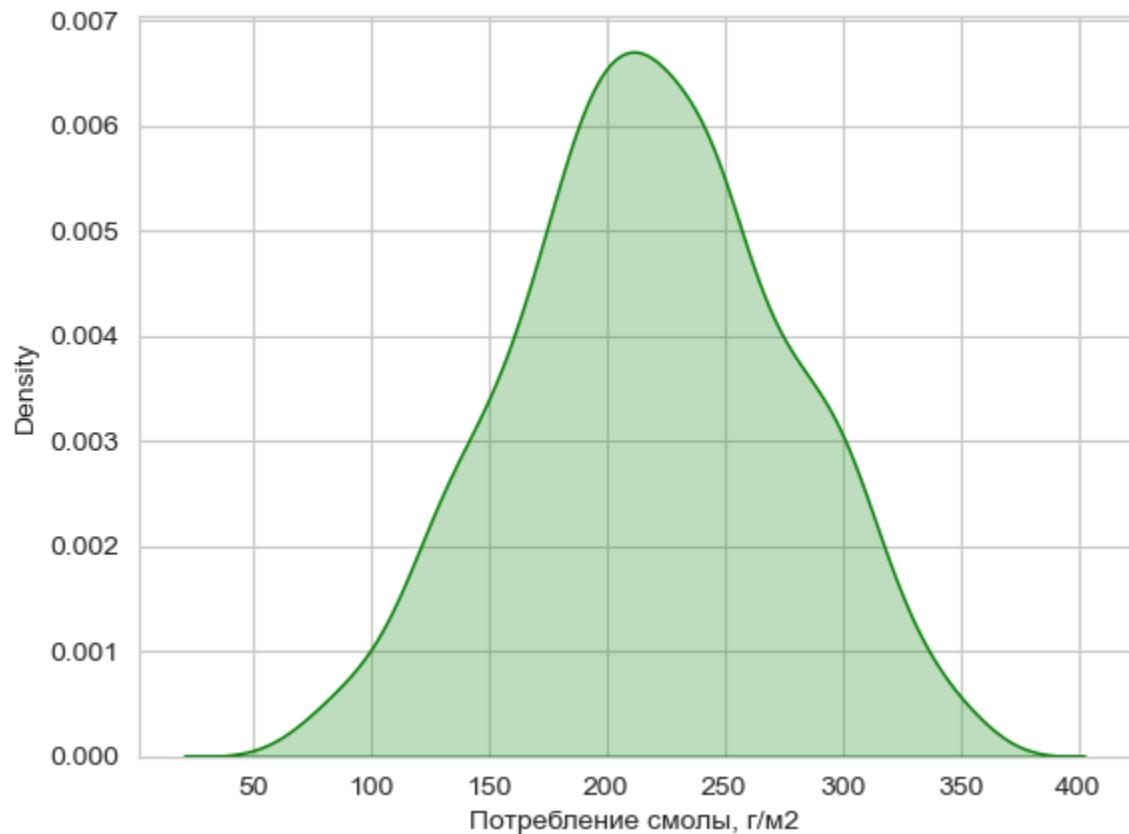
Медианное значение: 73.24759433381435

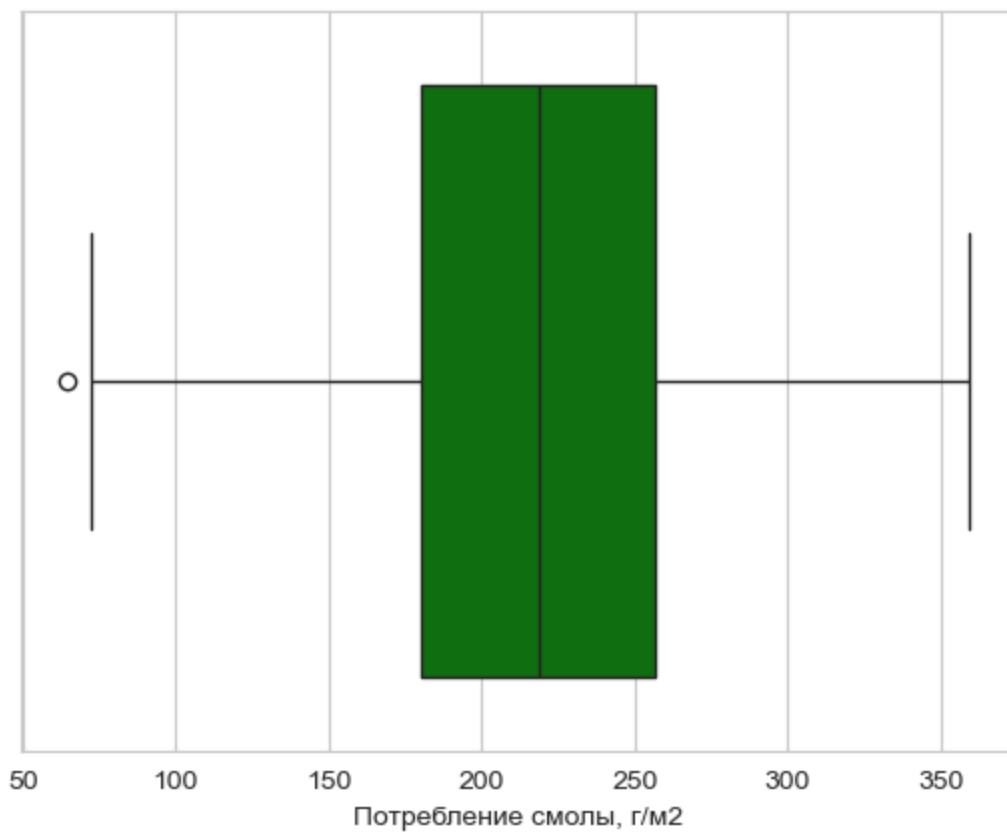
Прочность при растяжении, МПа



Минимальное значение: 1250.39280220501
Максимальное значение: 3660.45020981796
Среднее значение: 2465.1040891087805
Медианное значение: 2457.9597673317303

Потребление смолы, г/м²





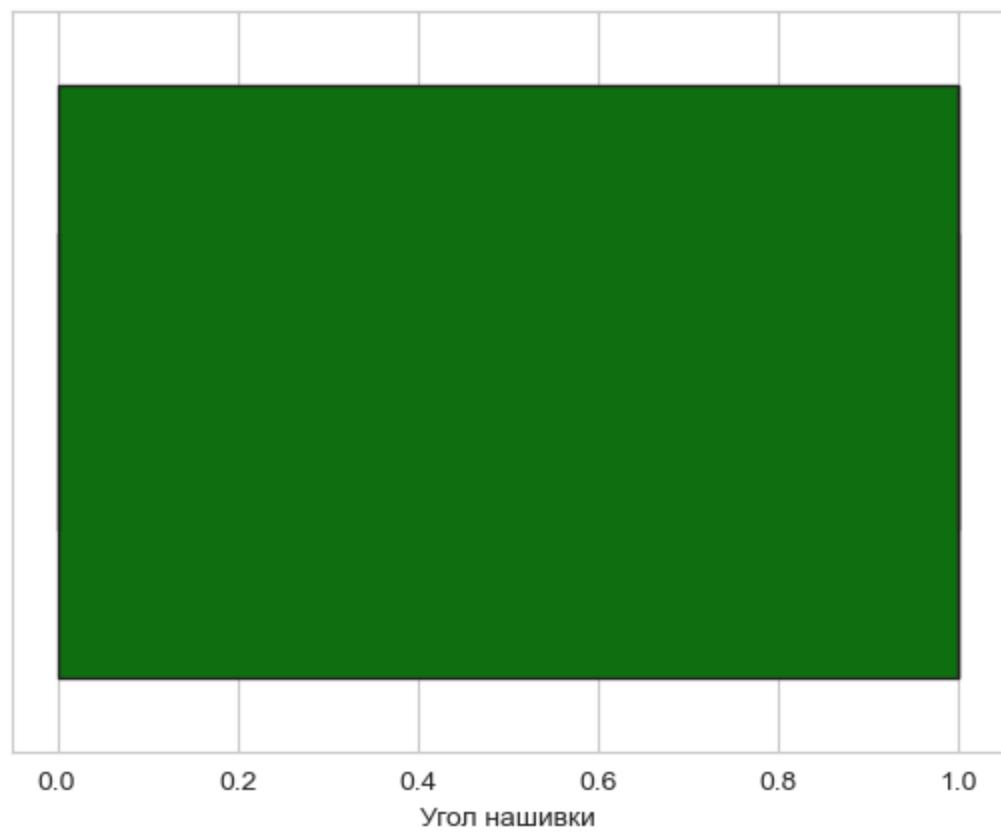
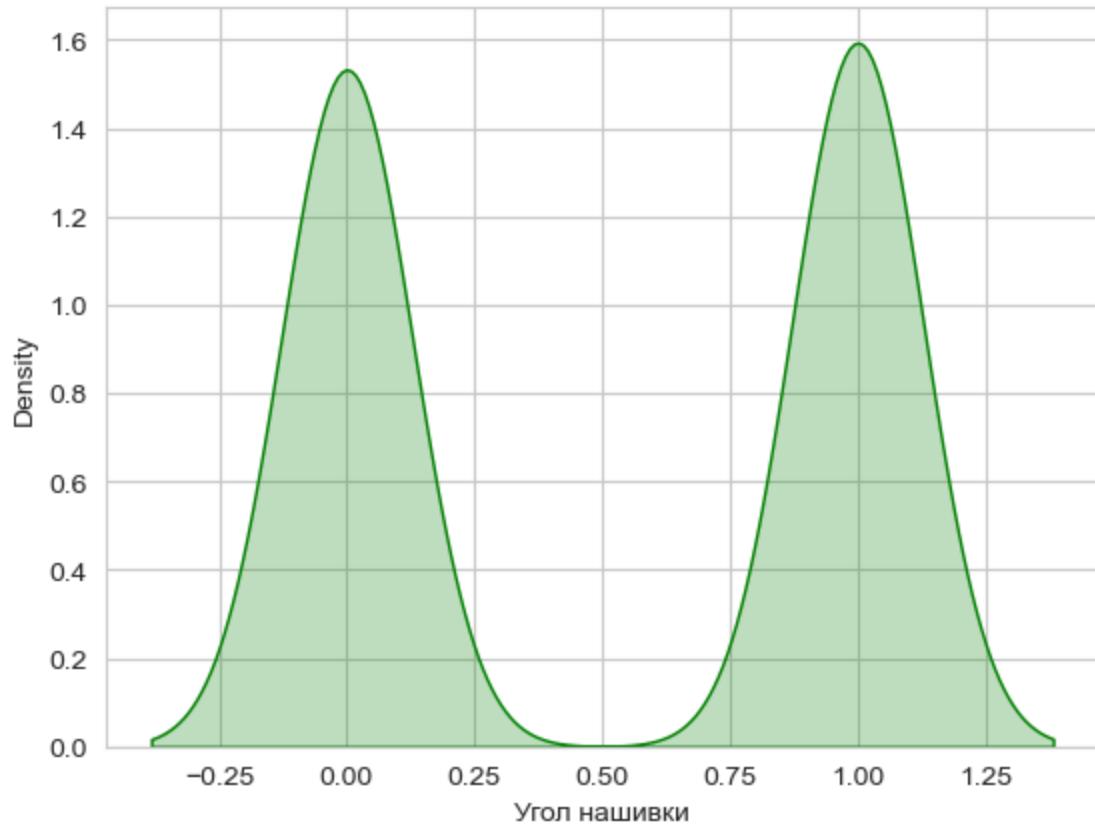
Минимальное значение: 64.5241796378105

Максимальное значение: 359.052219789673

Среднее значение: 218.01221267541476

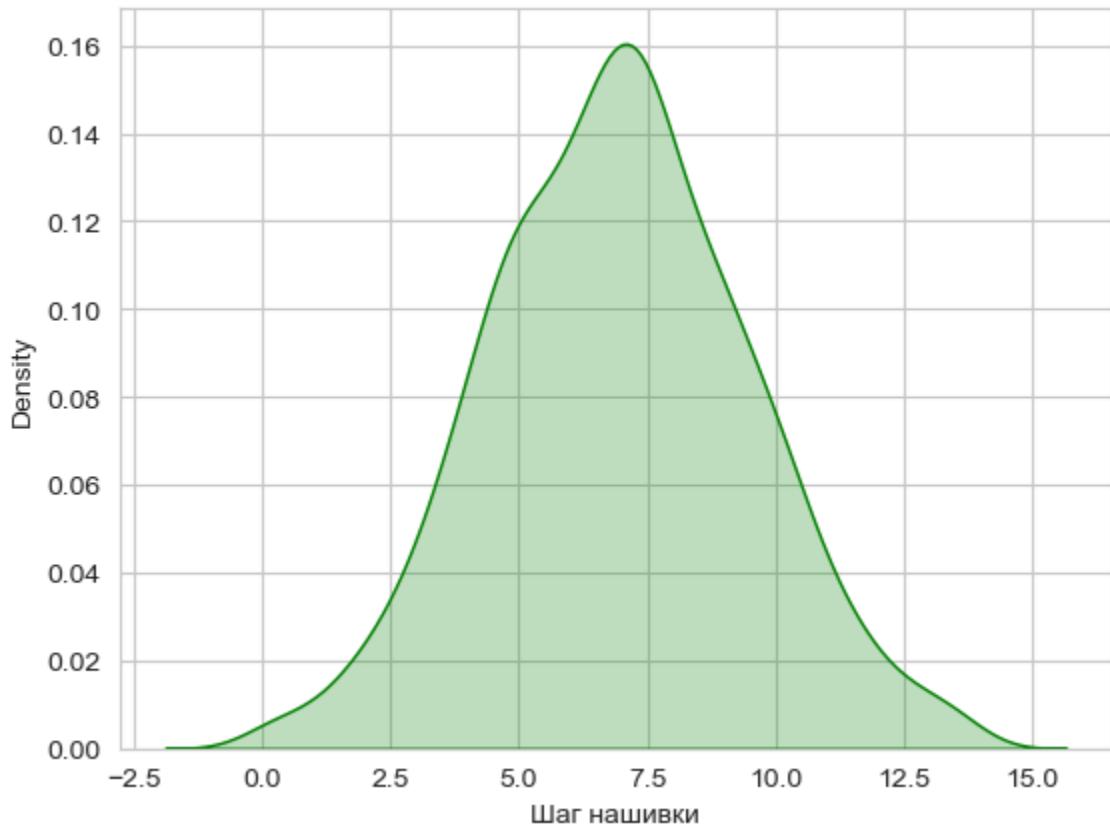
Медианное значение: 218.697659676914

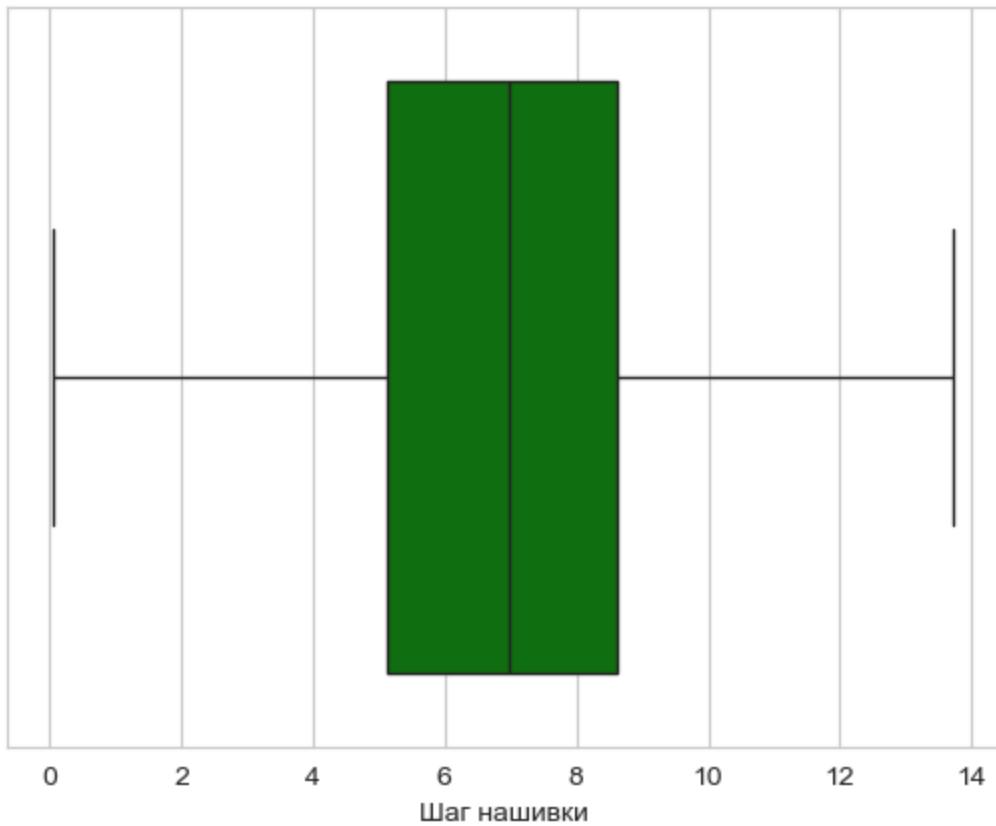
Угол нашивки



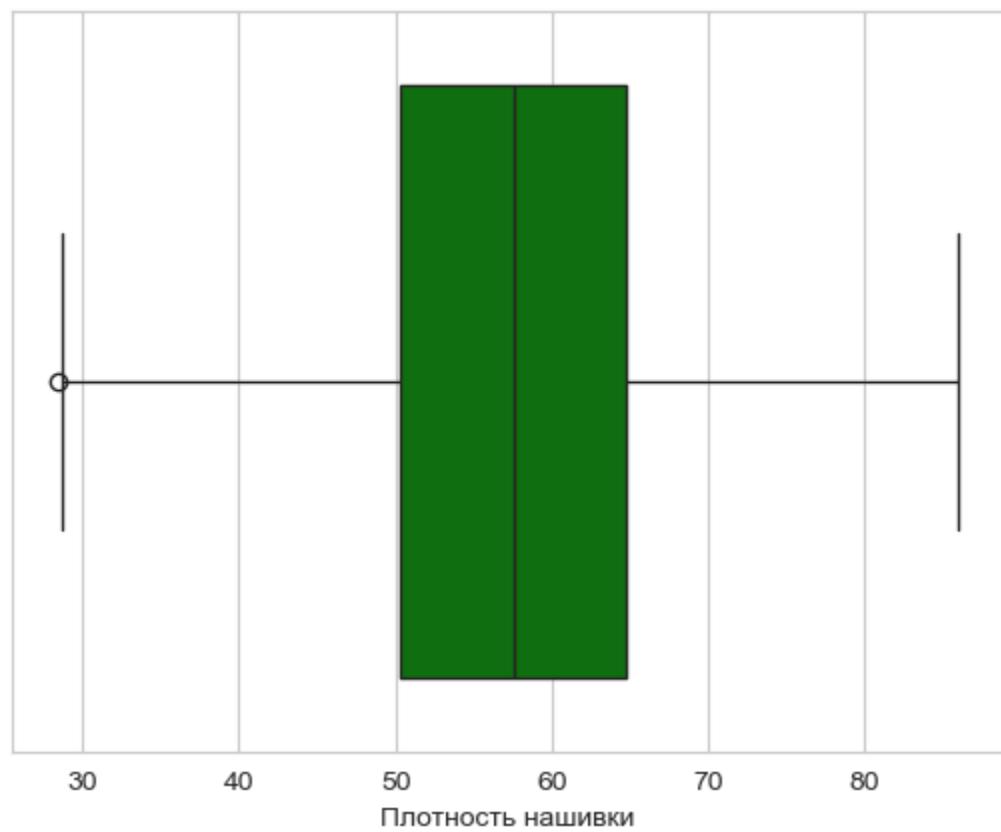
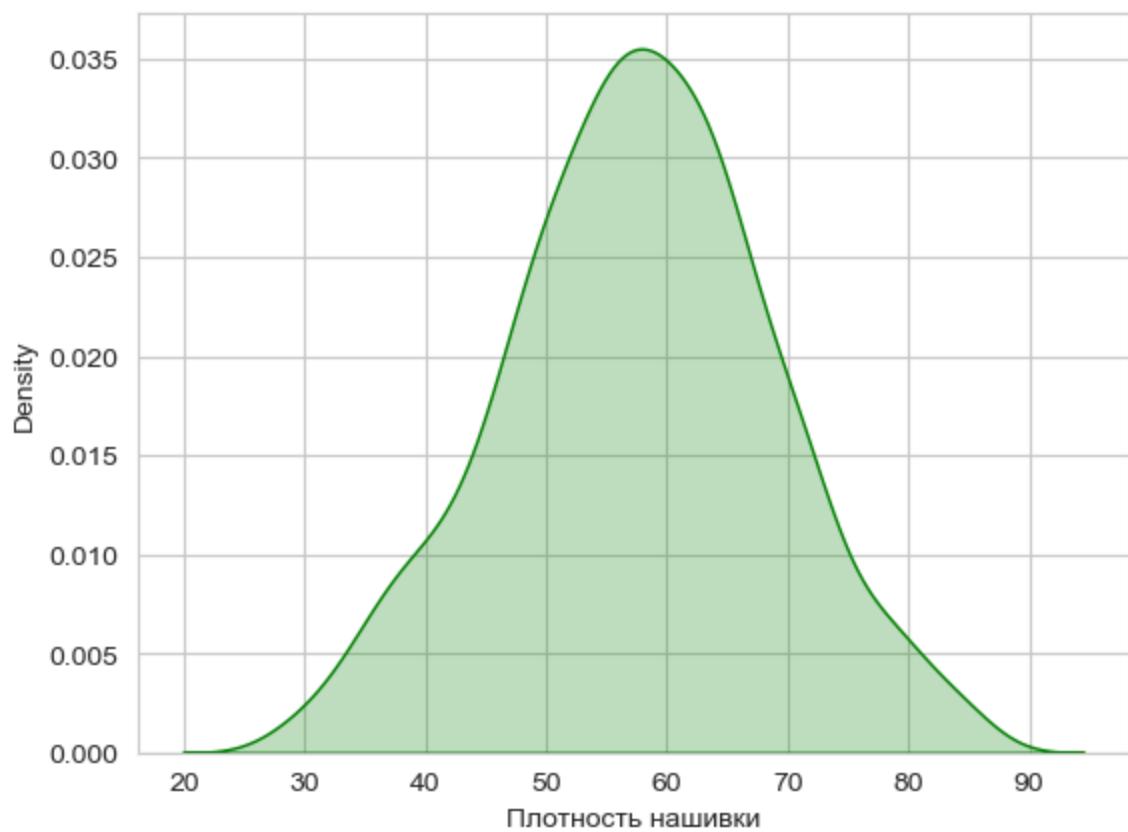
Минимальное значение: 0
Максимальное значение: 1
Среднее значение: 0.509719222462203
Медианное значение: 1.0

Шаг нашивки



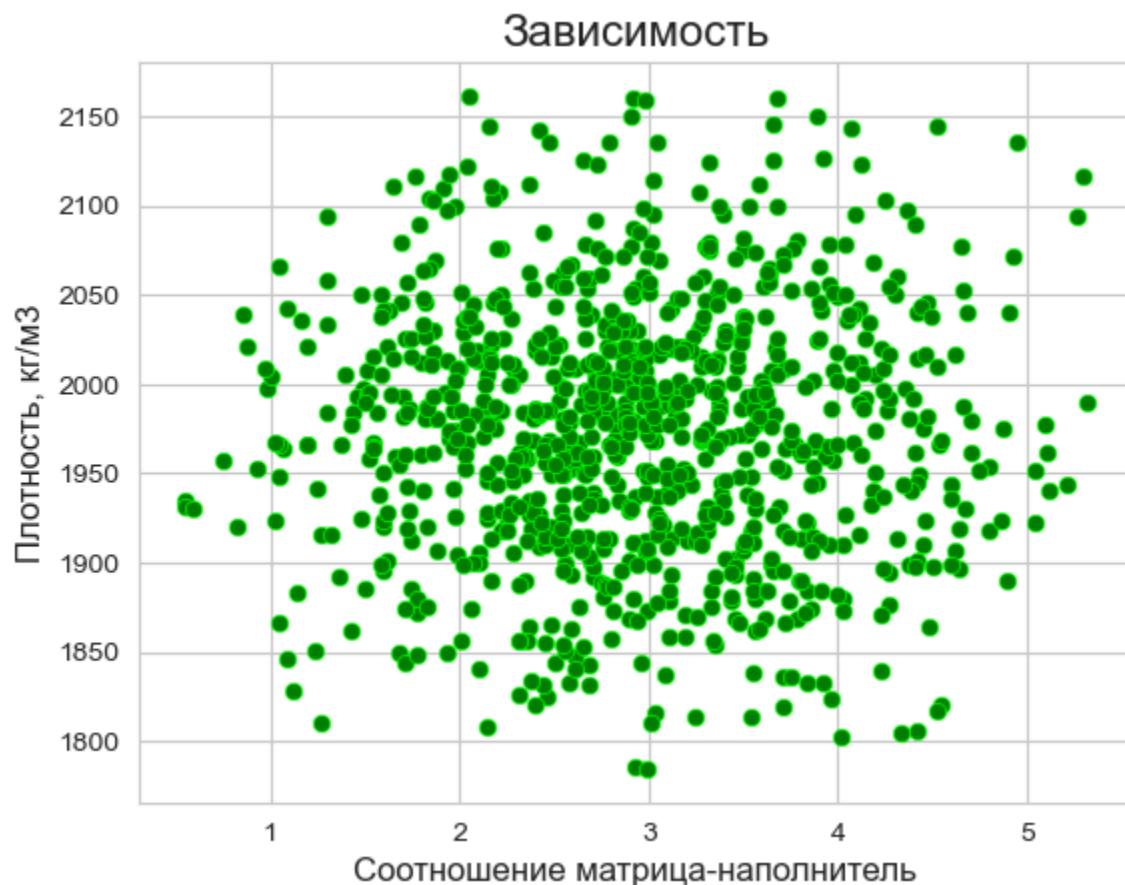


Плотность нашивки

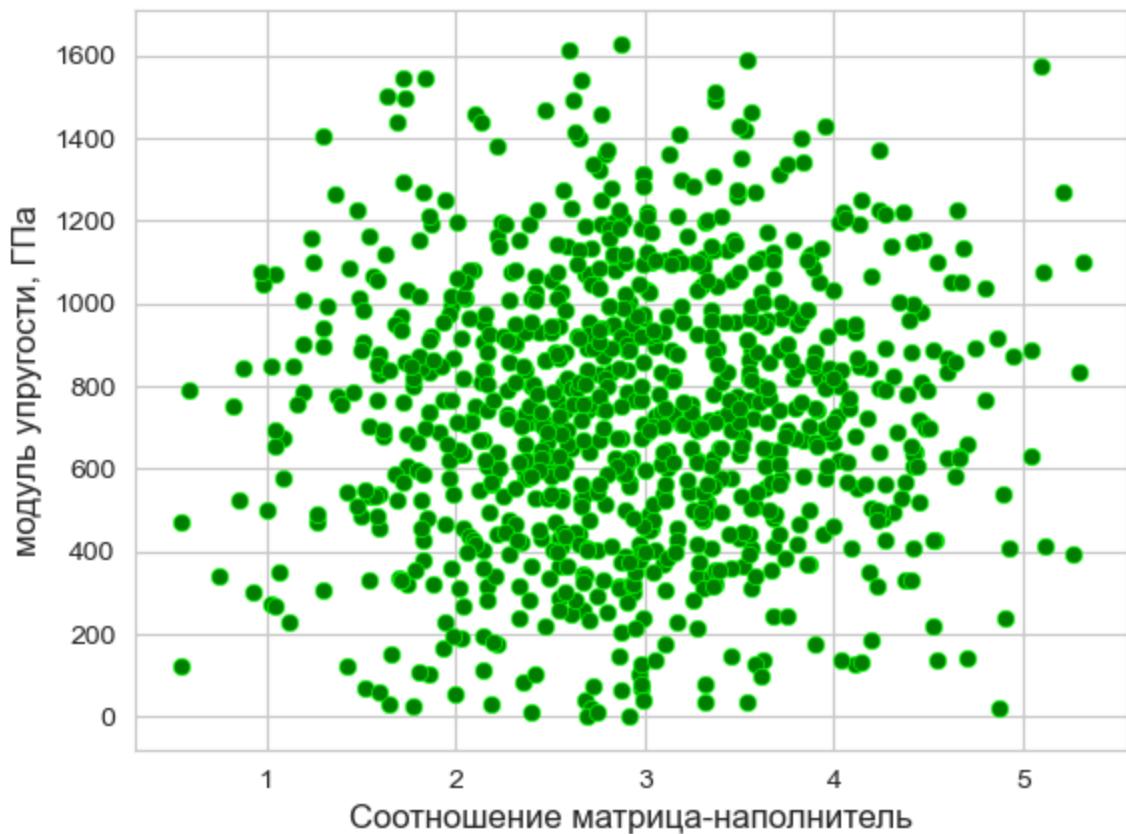


Минимальное значение: 28.3824774111668
Максимальное значение: 86.0124270098611
Среднее значение: 57.539051695078584
Медианное значение: 57.60929450498985

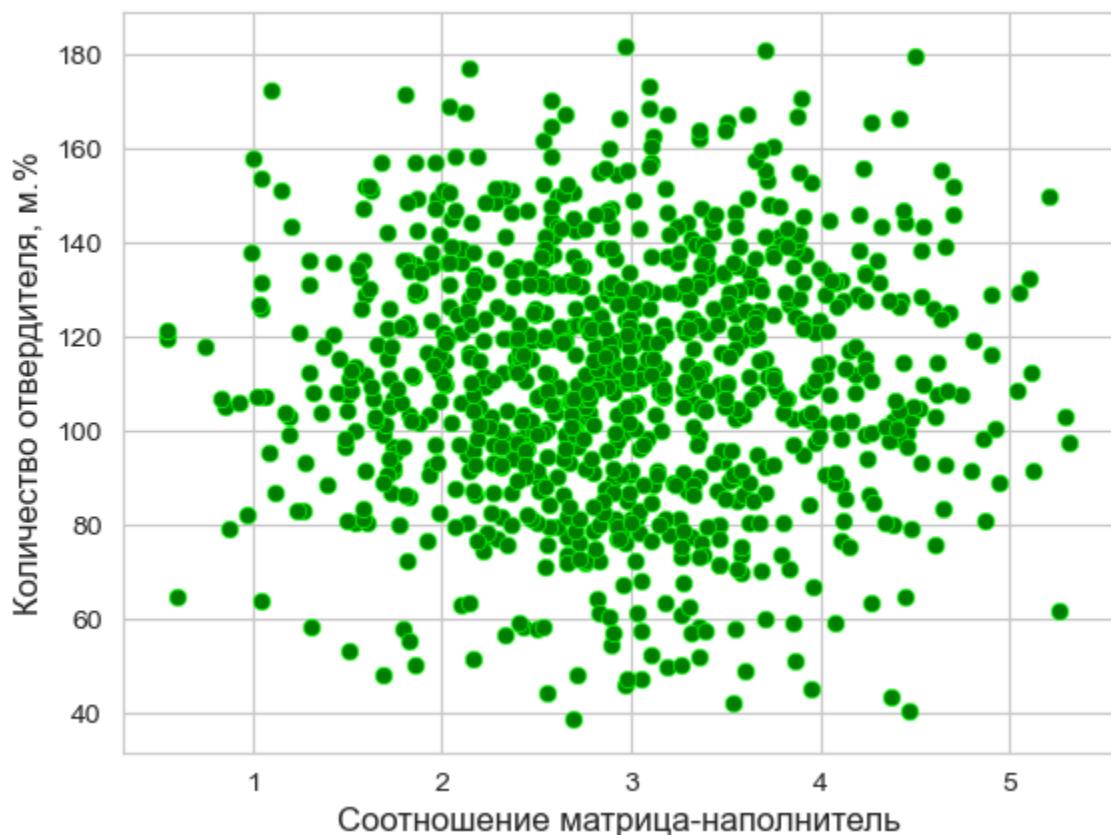
```
In [76]: n = 0
while n < len(column_names):
    b = n + 1
    while b < len(column_names):
        sns.set_style('whitegrid')
        plt.title('Зависимость', size = 16)
        plt.xlabel(column_names[n], size = 12)
        plt.ylabel(column_names[b], size = 12)
        sns.scatterplot(x = column_names[n], y = column_names[b], data = df, color = 'green')
        plt.show()
        b += 1
    n += 1
```



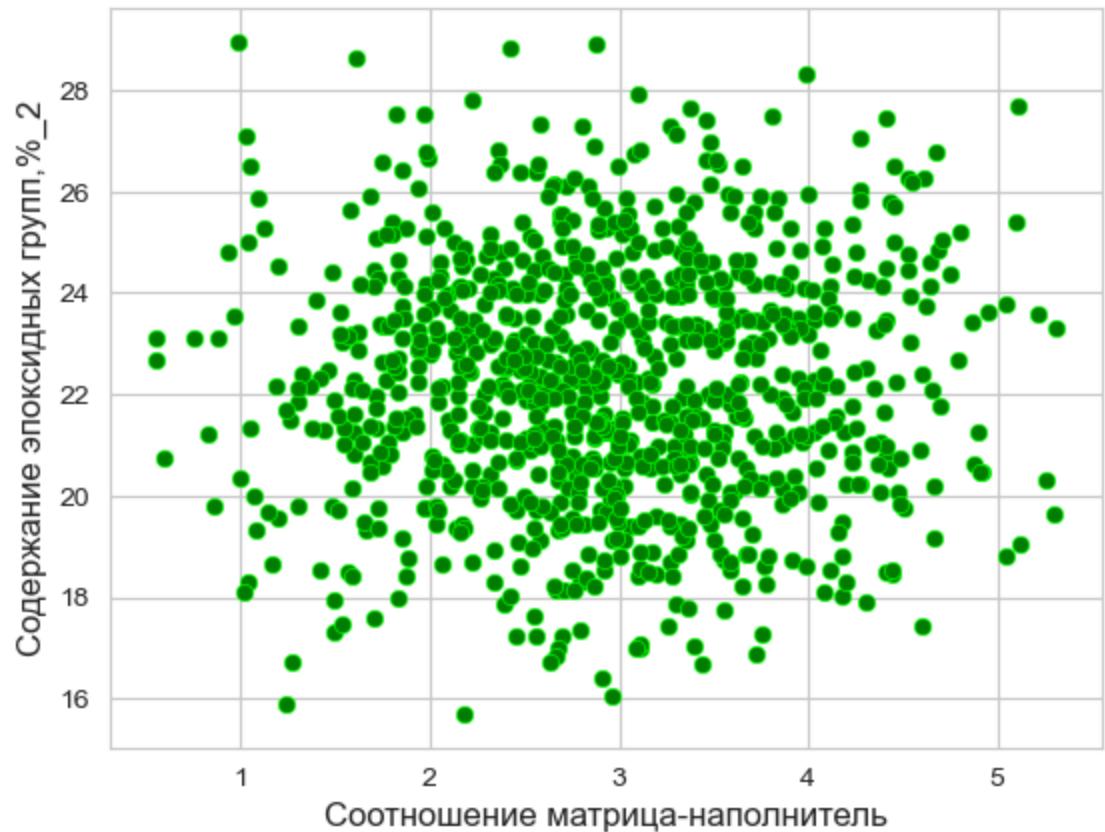
Зависимость



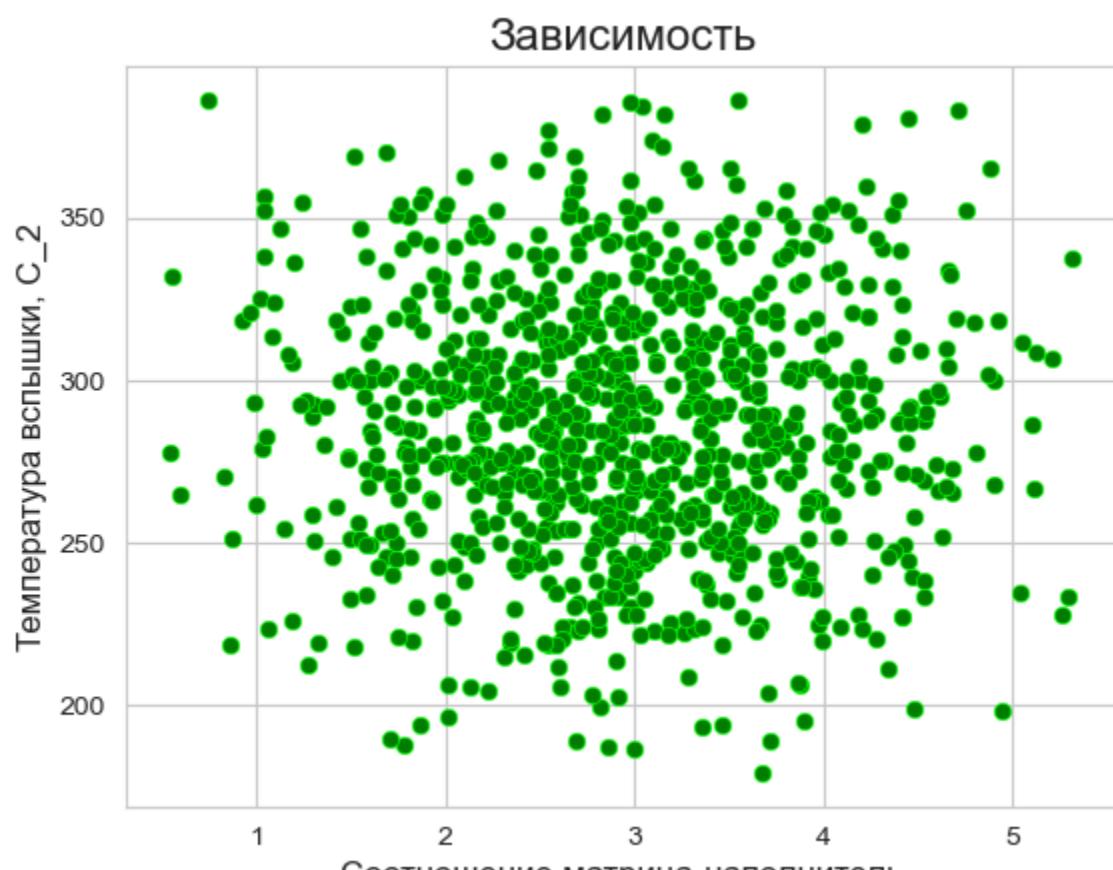
Зависимость



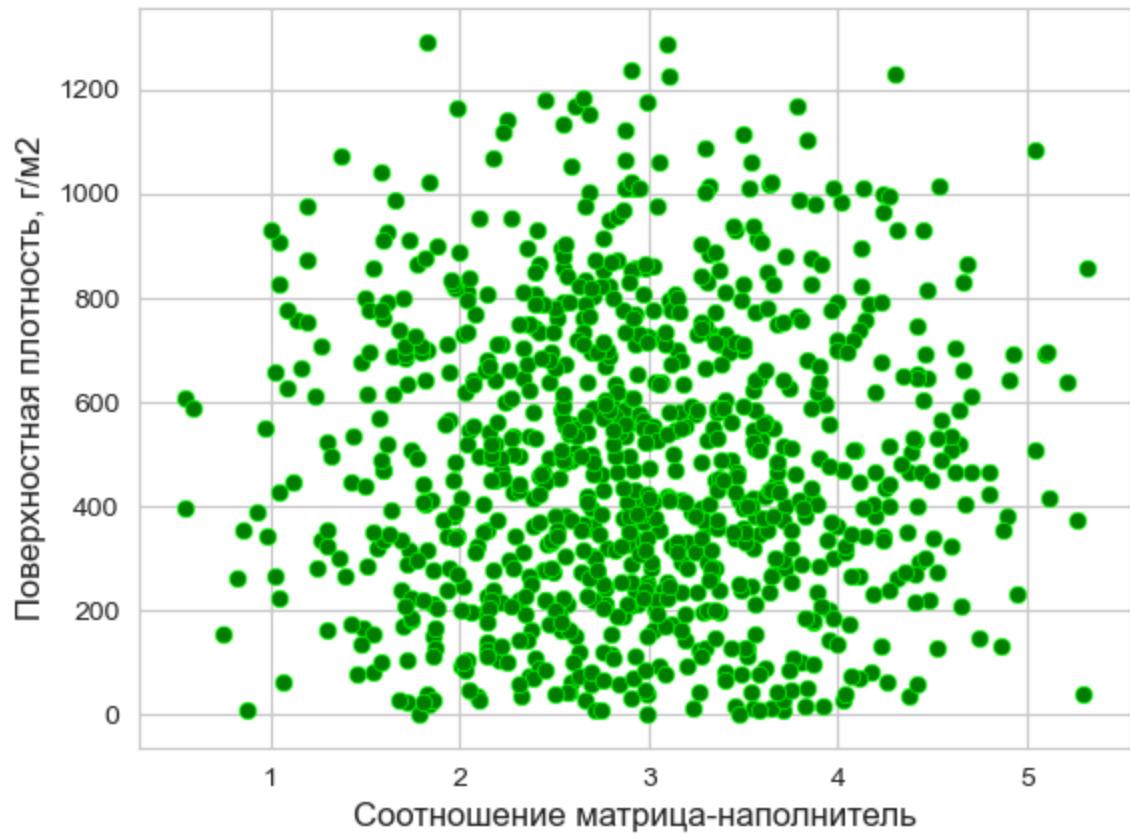
Зависимость



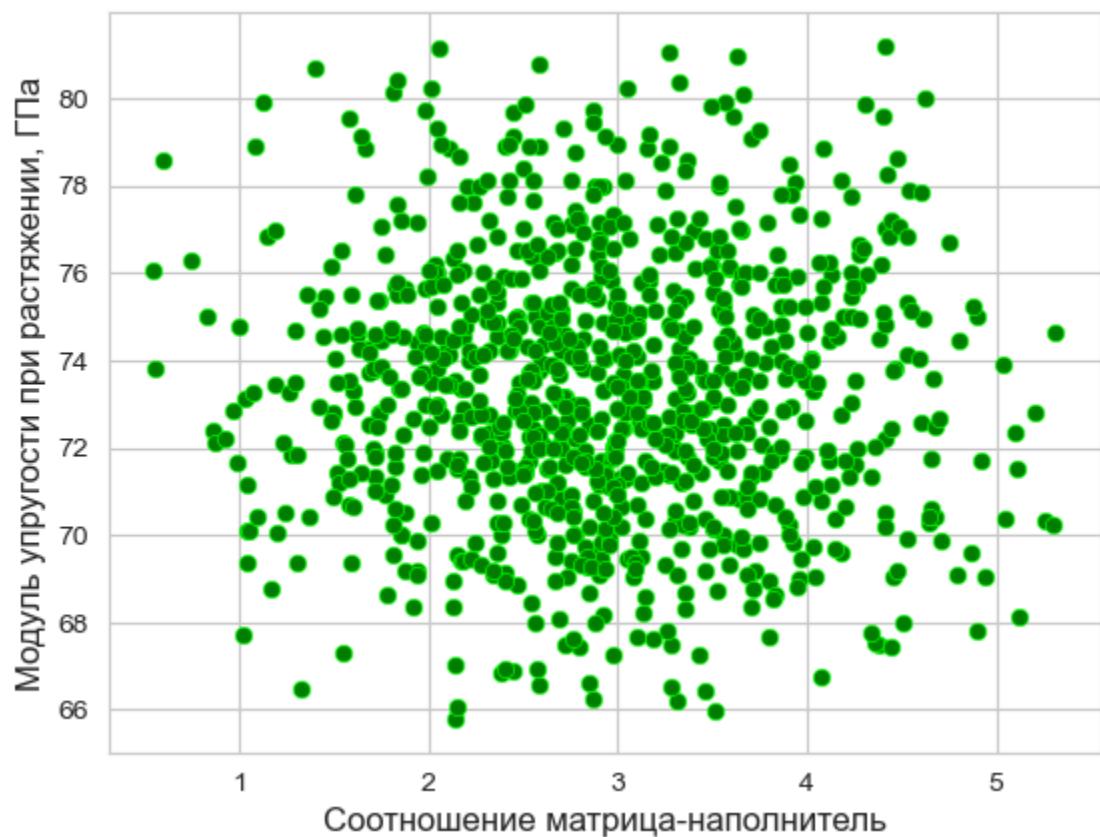
Зависимость



Зависимость



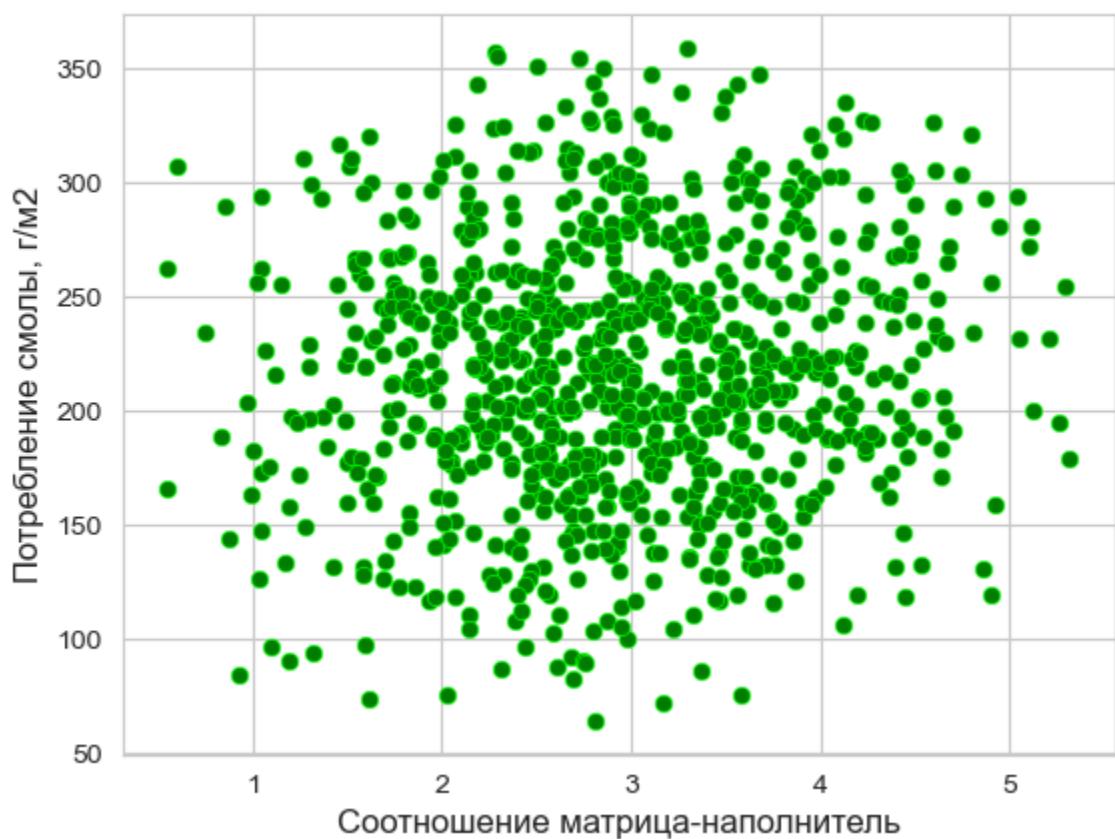
Зависимость



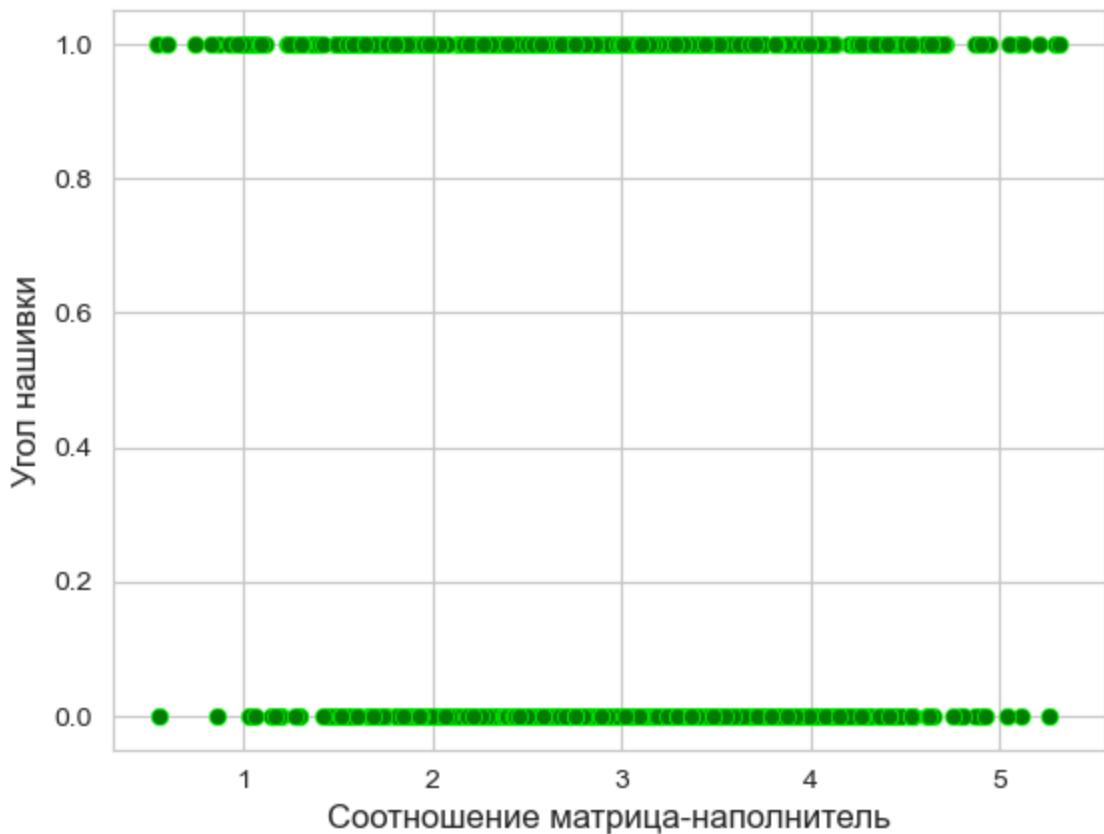
Зависимость



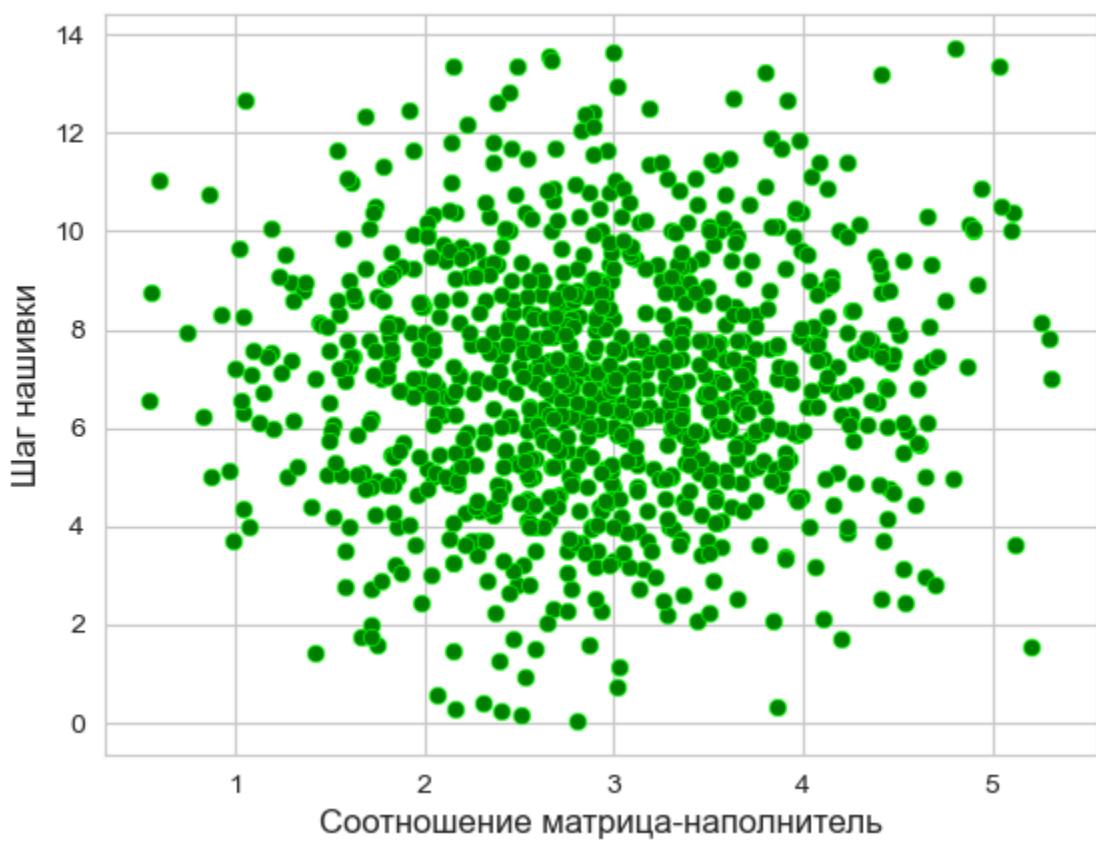
Зависимость



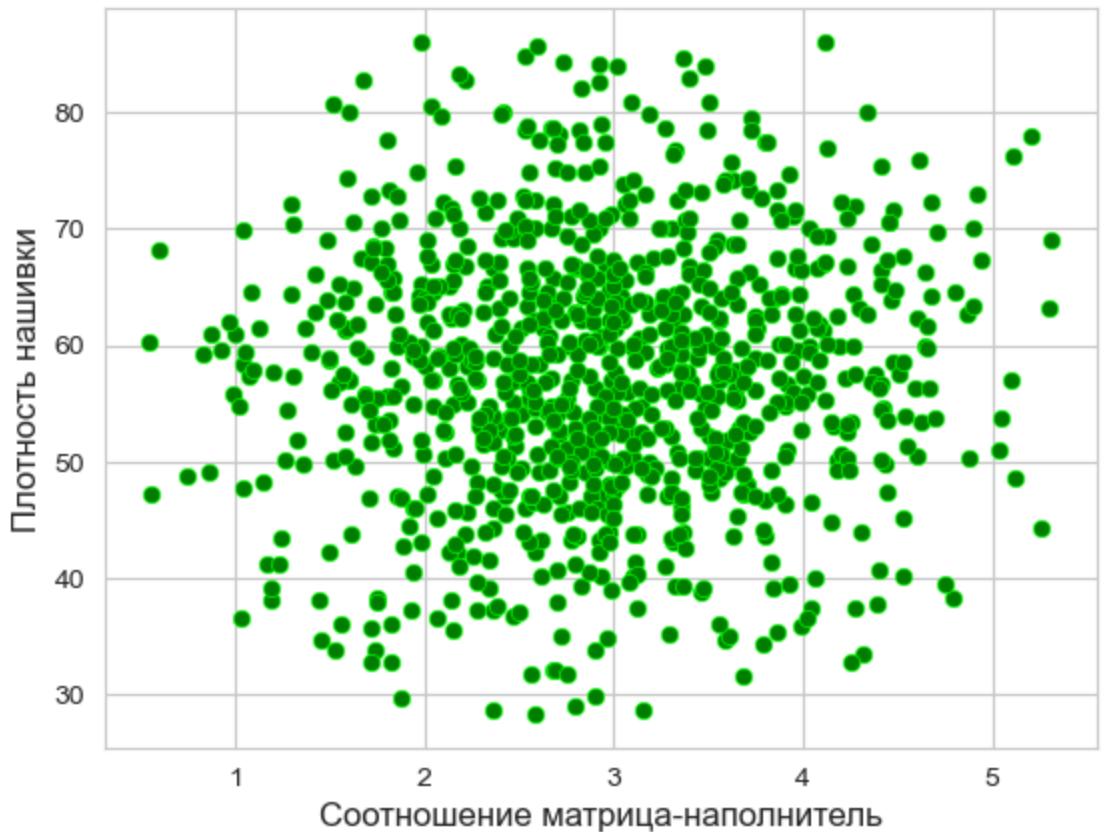
Зависимость



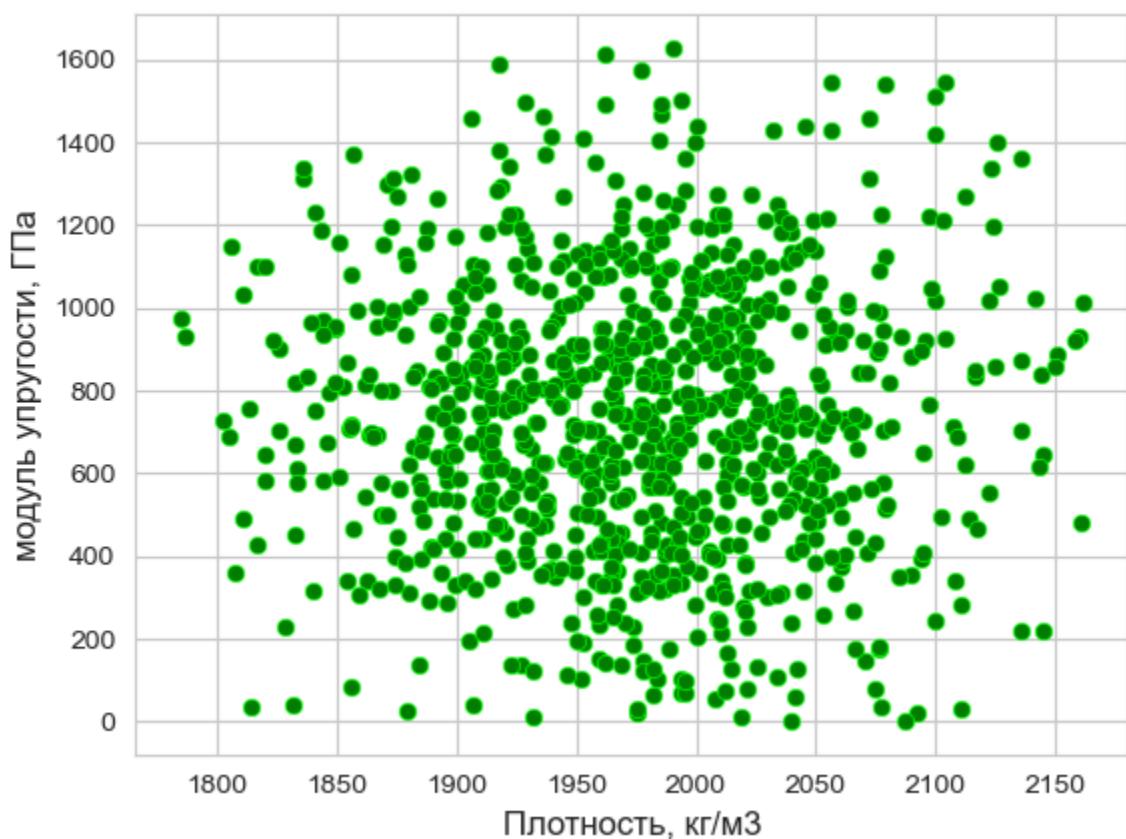
Зависимость



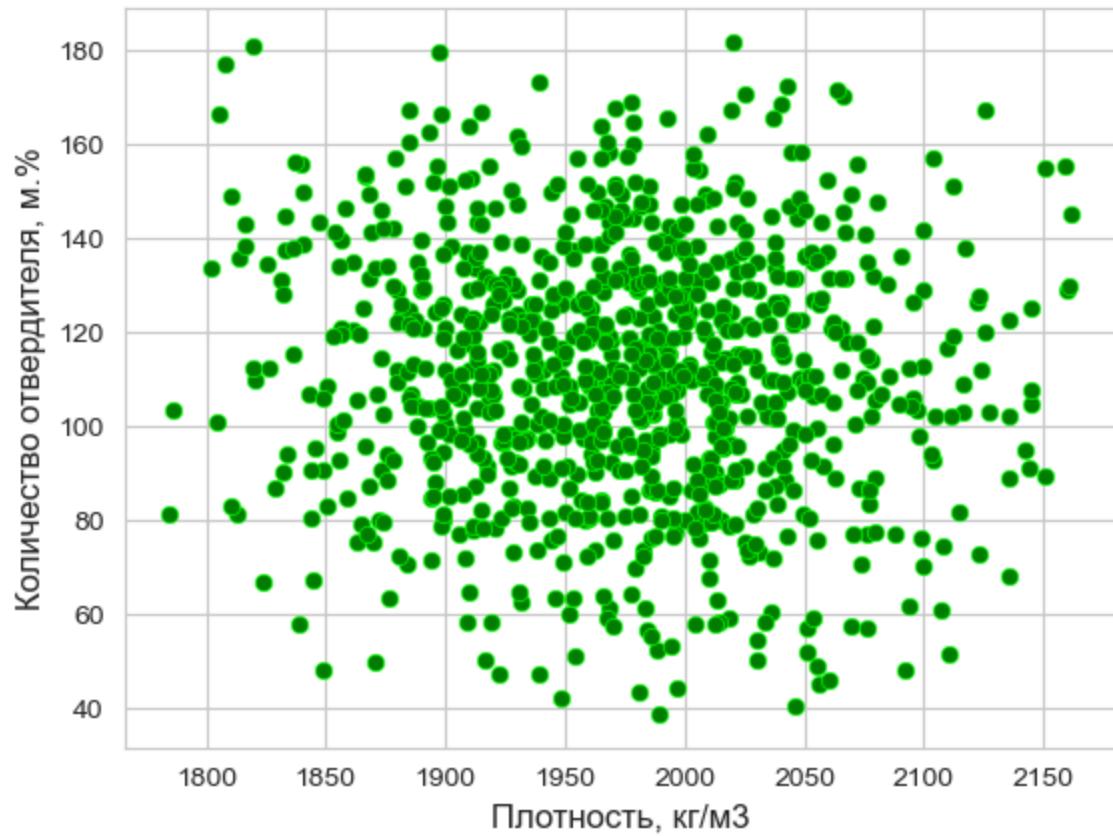
Зависимость



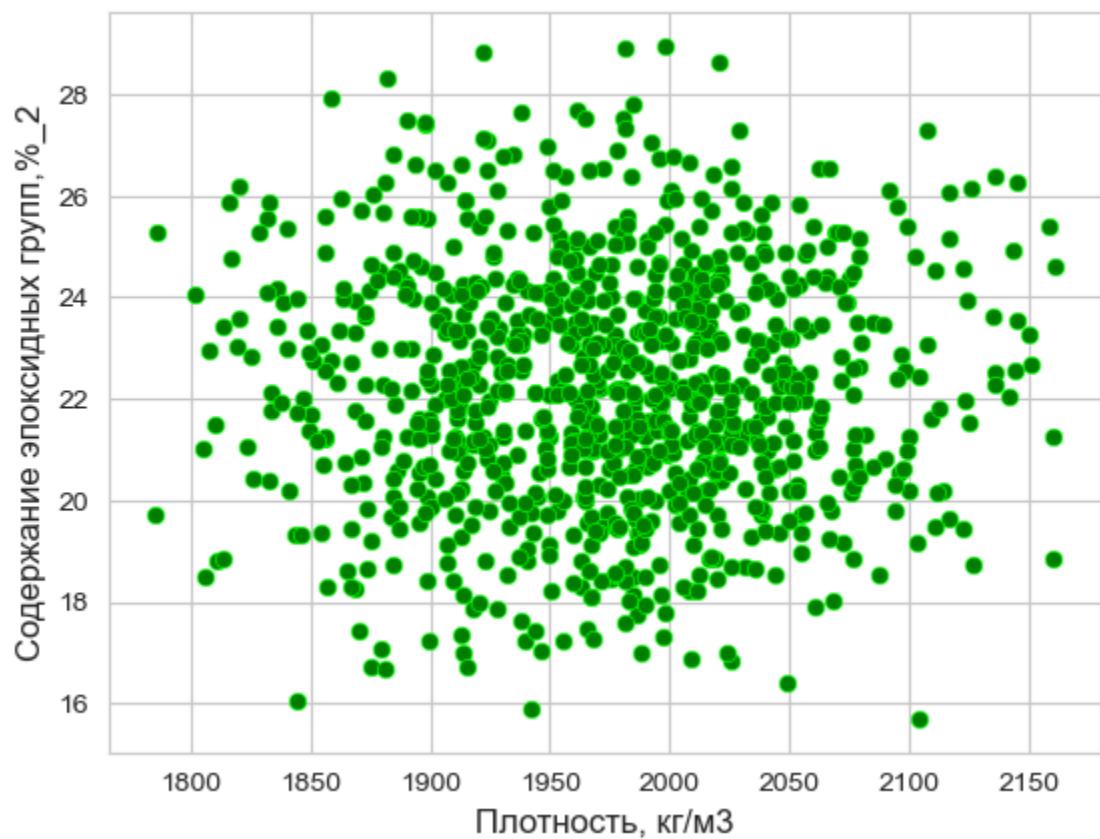
Зависимость



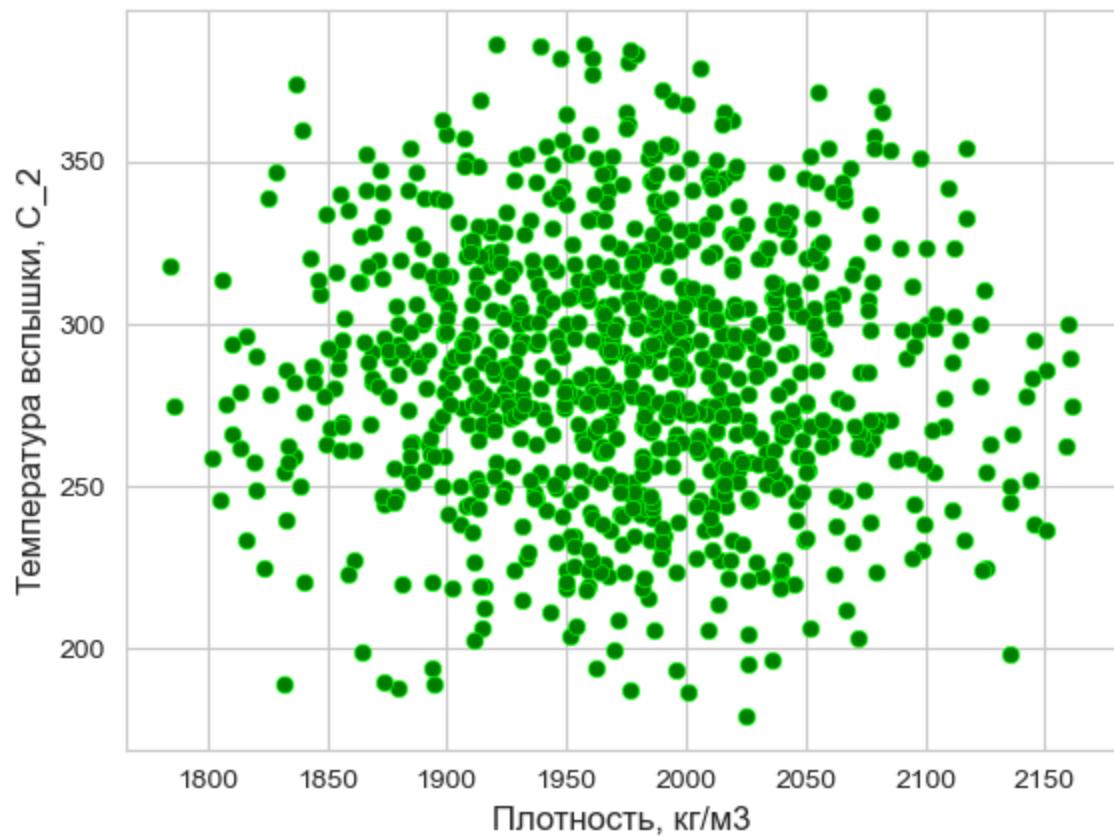
Зависимость



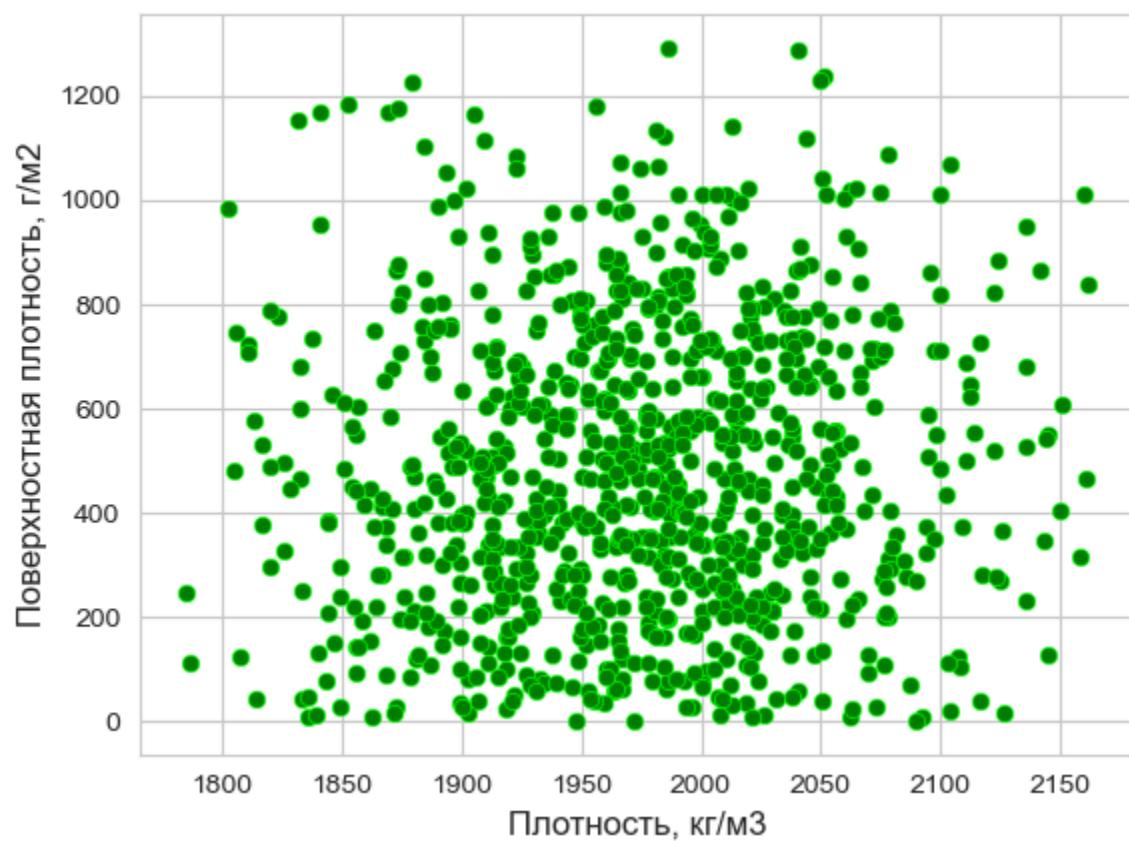
Зависимость



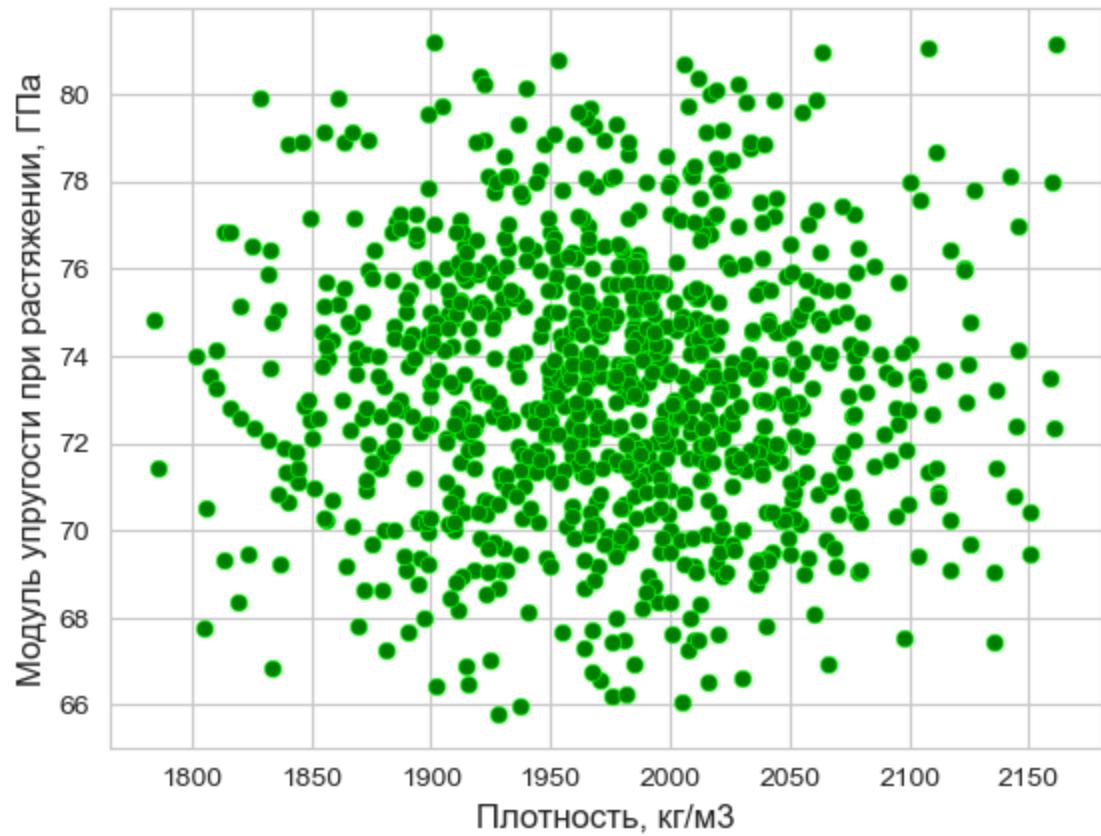
Зависимость



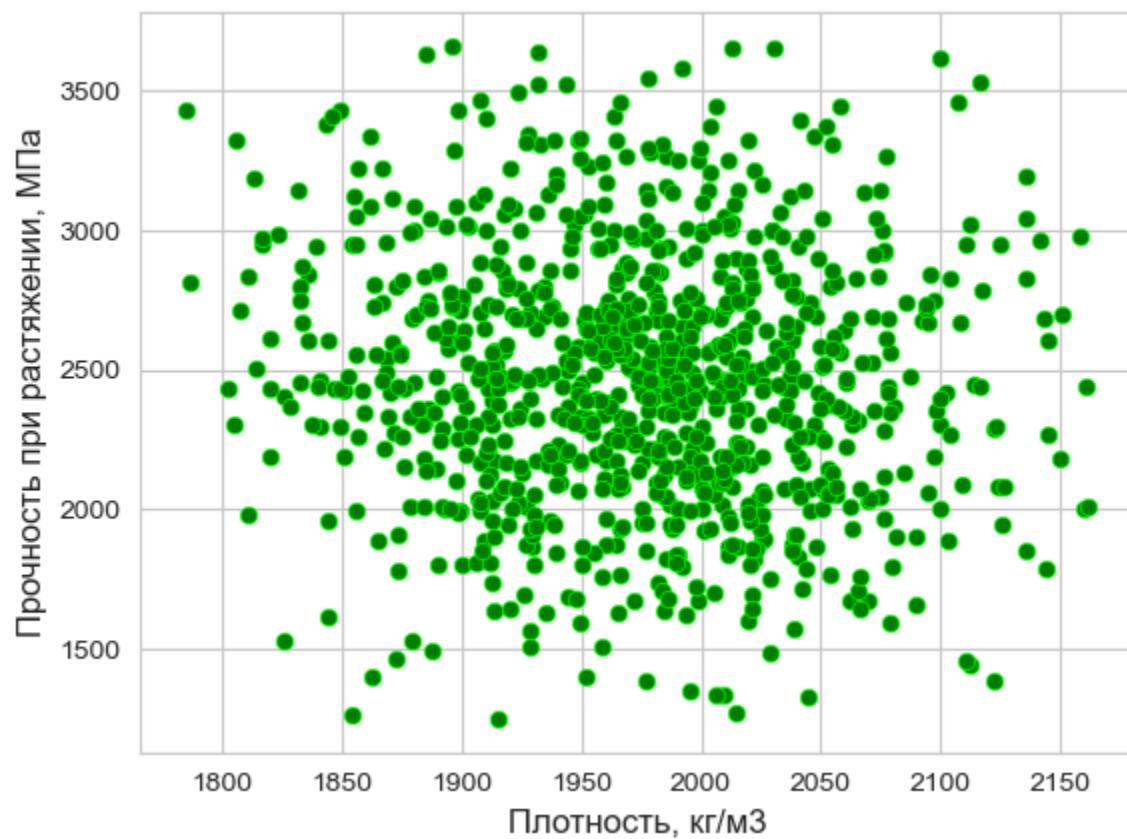
Зависимость



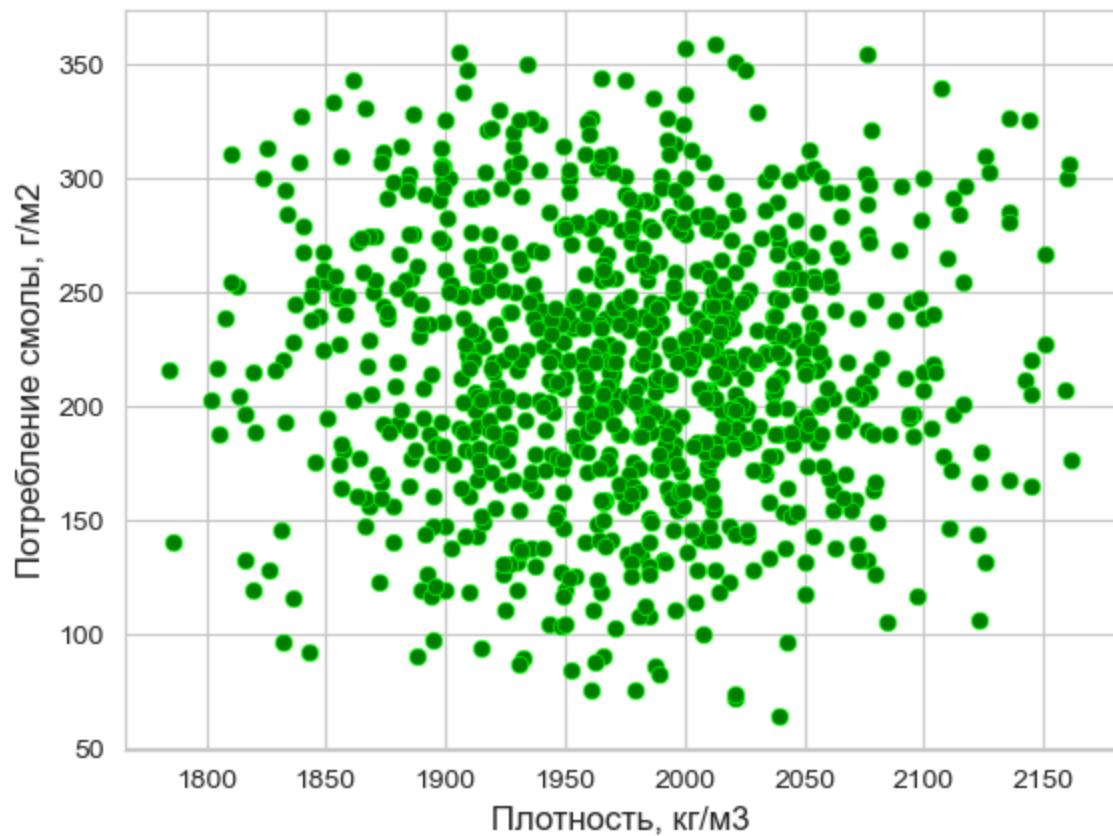
Зависимость



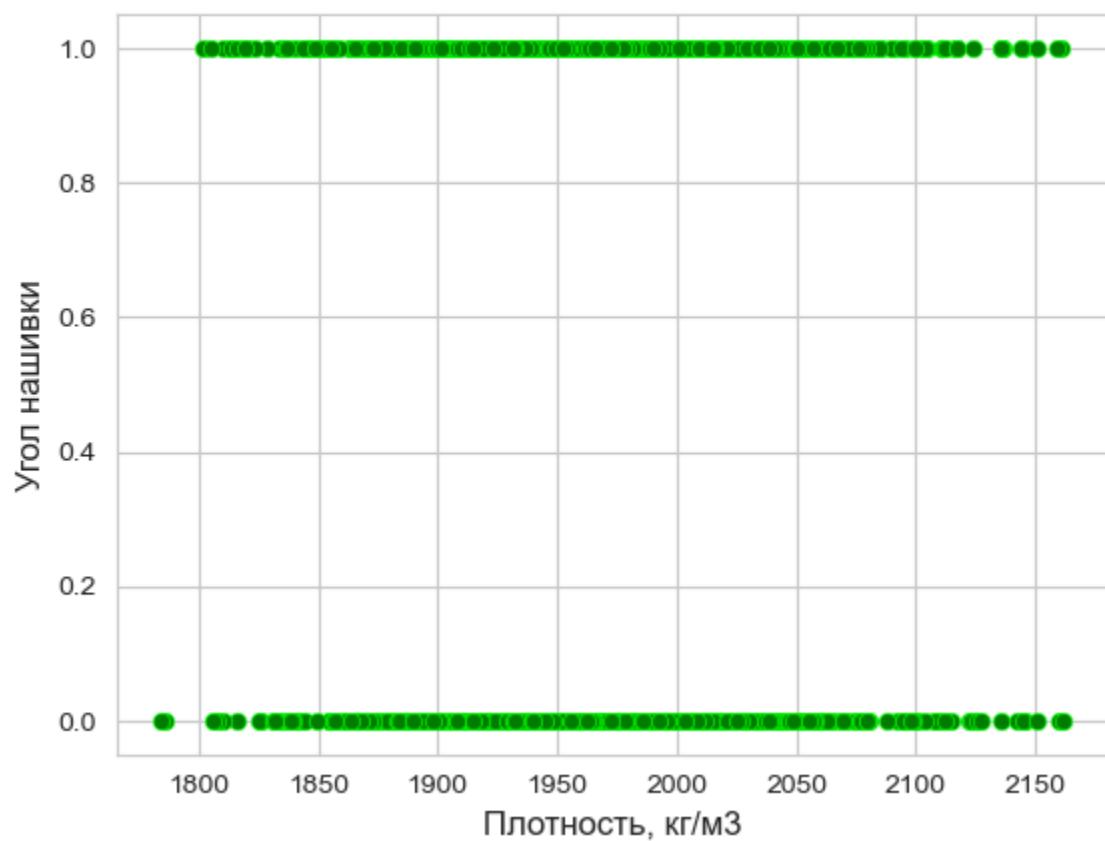
Зависимость



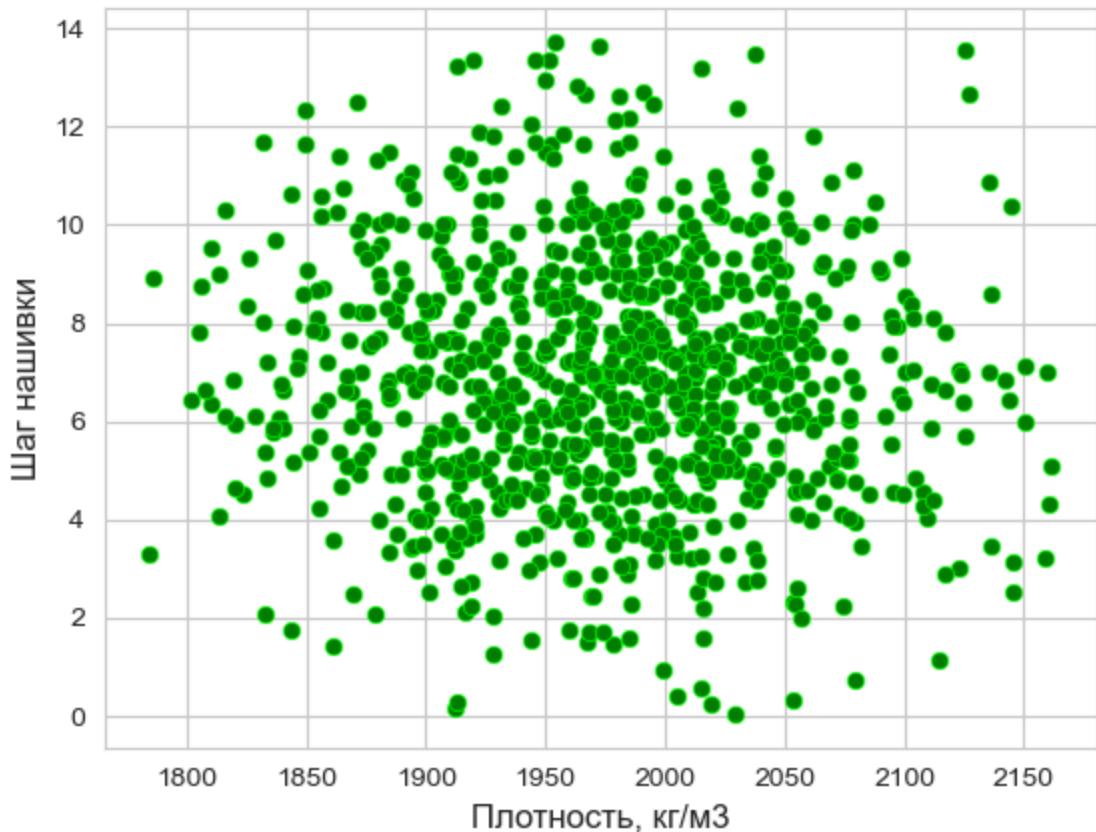
Зависимость



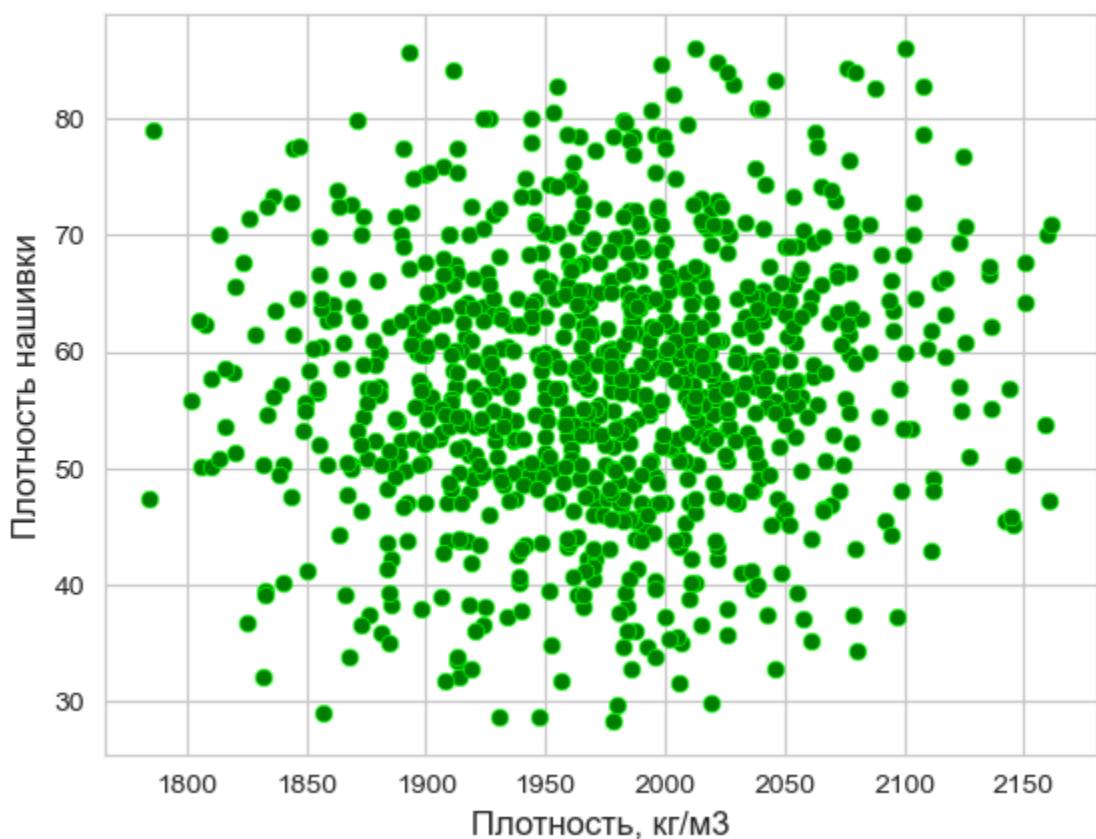
Зависимость



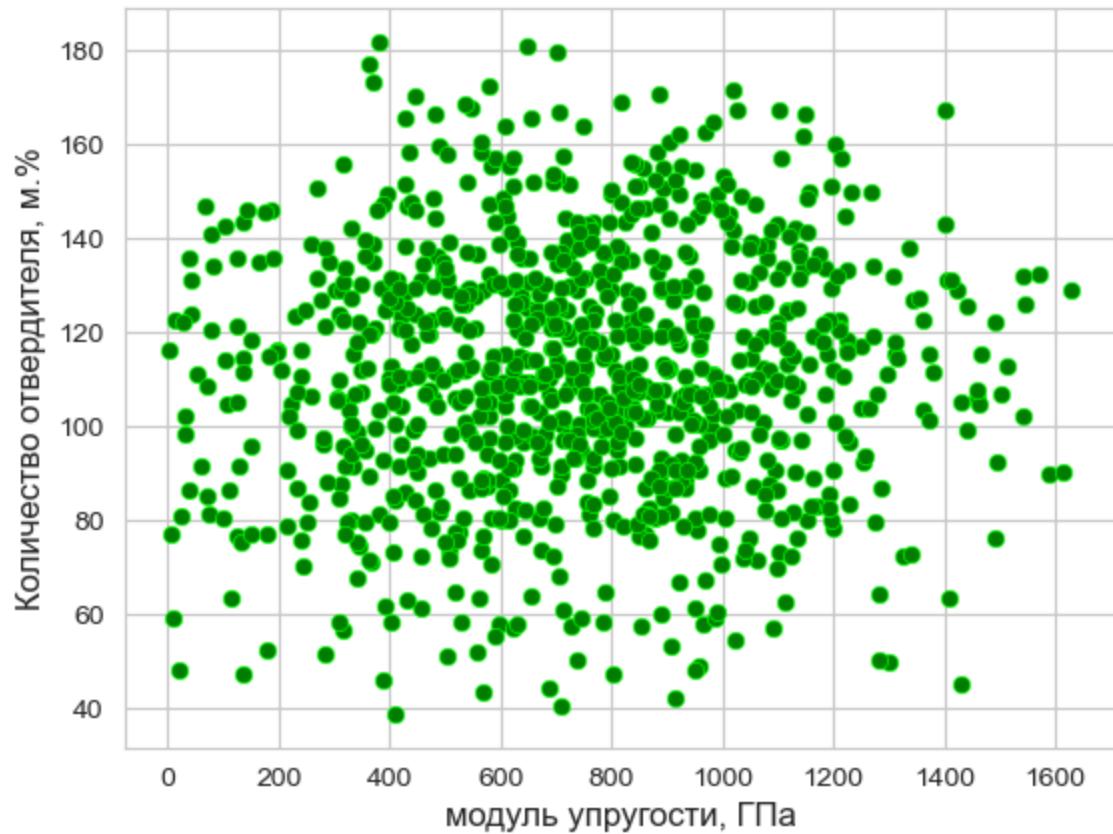
Зависимость



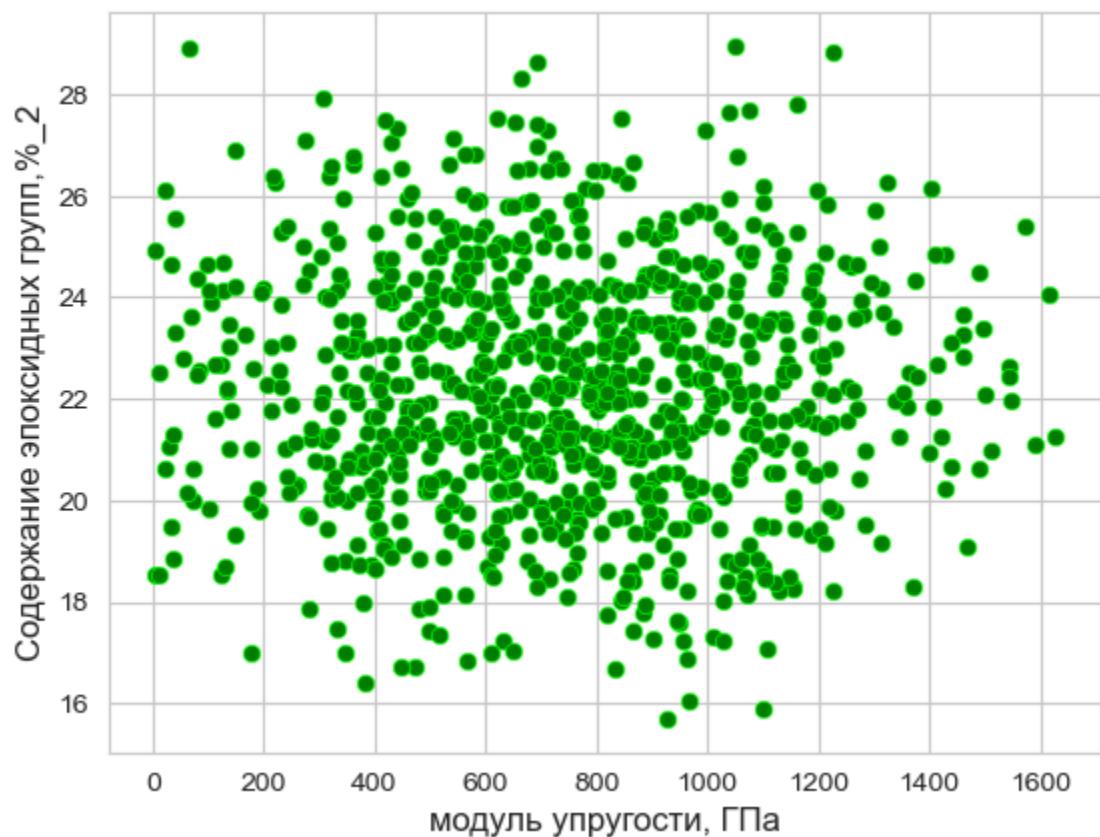
Зависимость



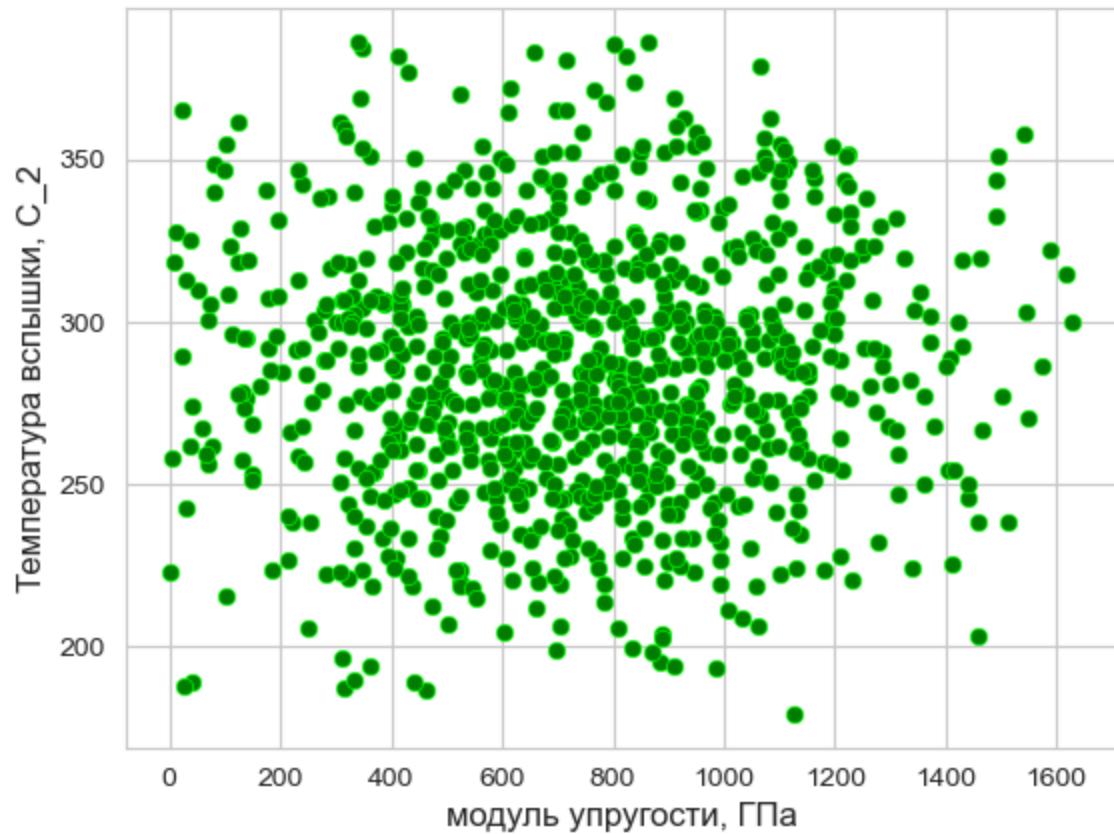
Зависимость



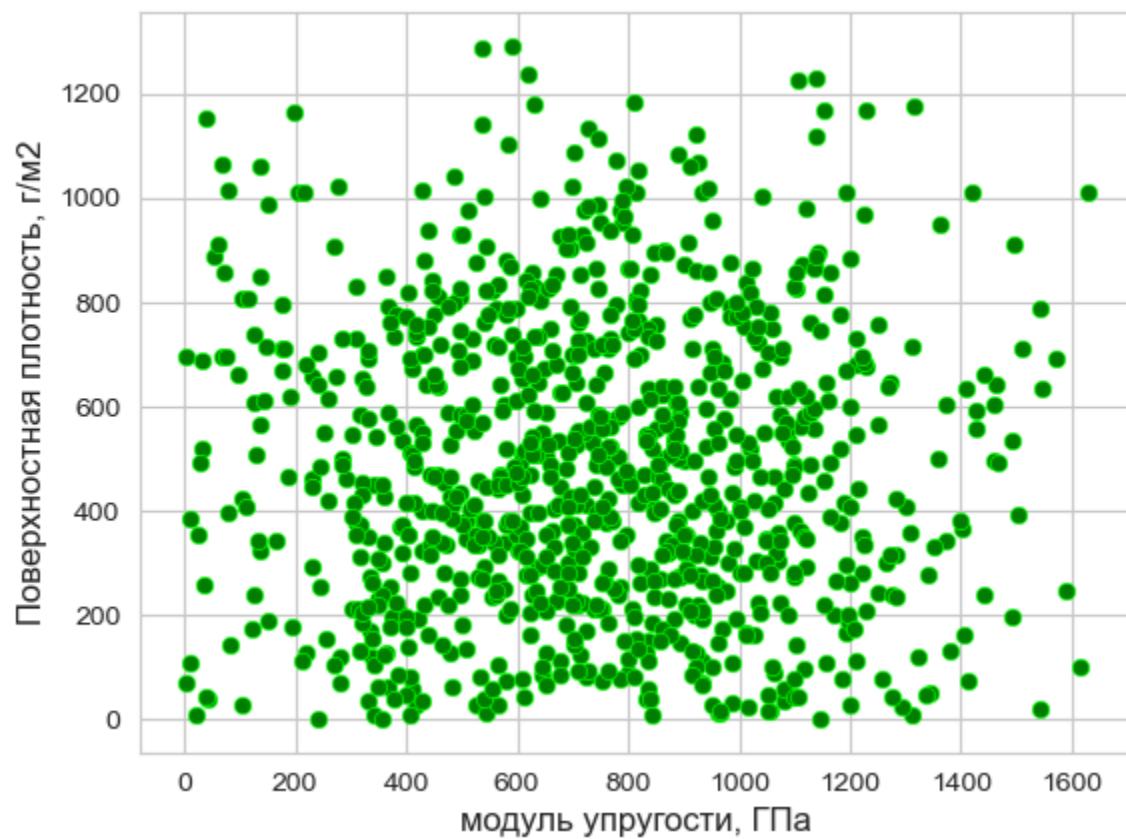
Зависимость



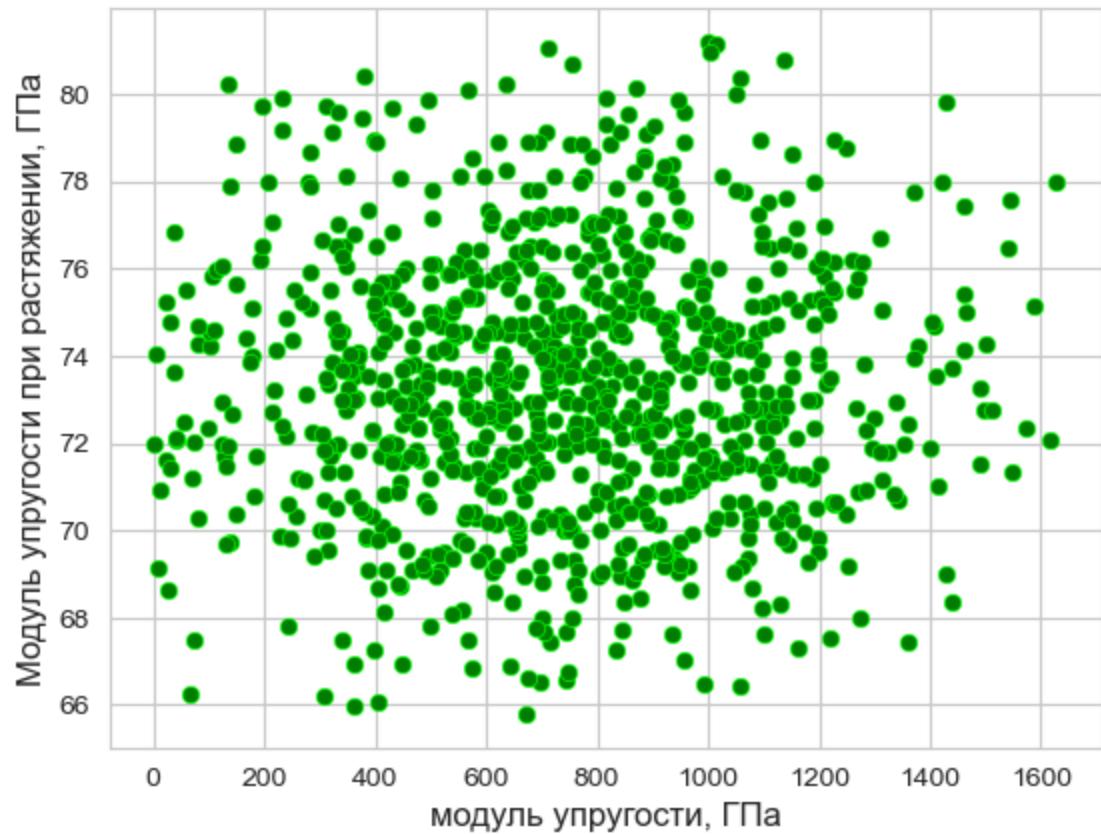
Зависимость



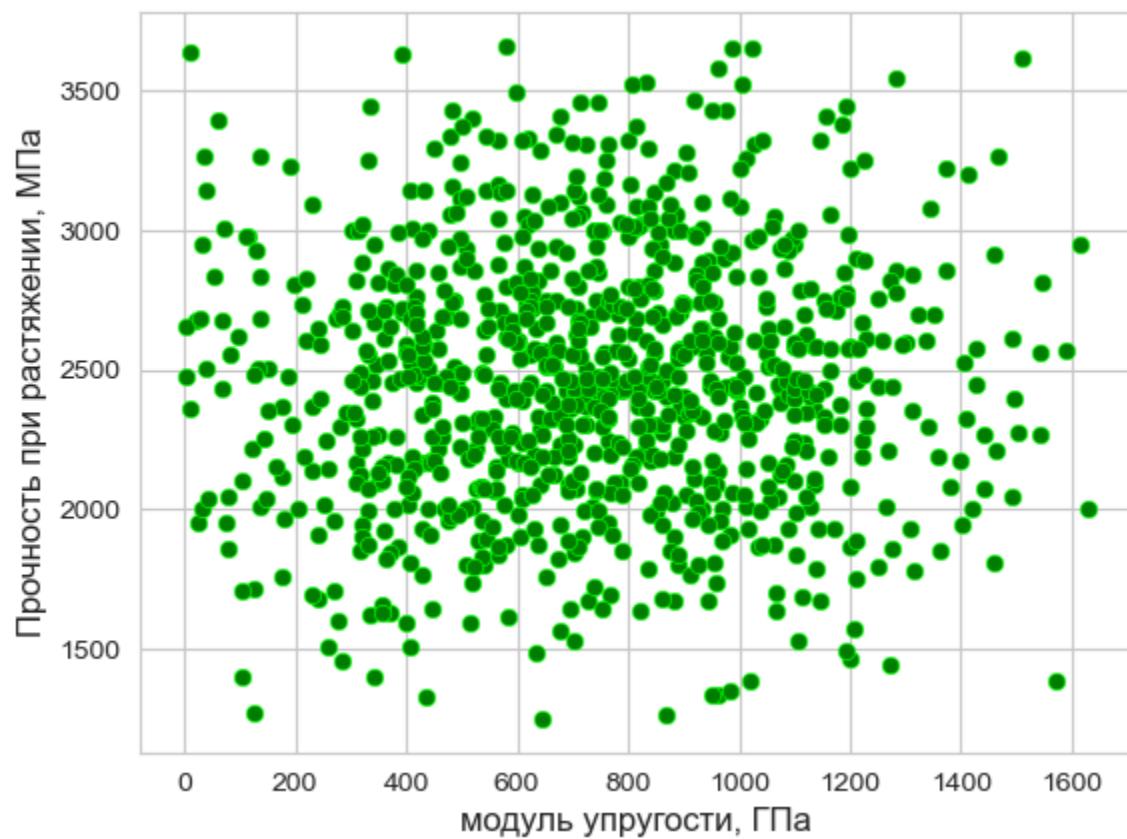
Зависимость



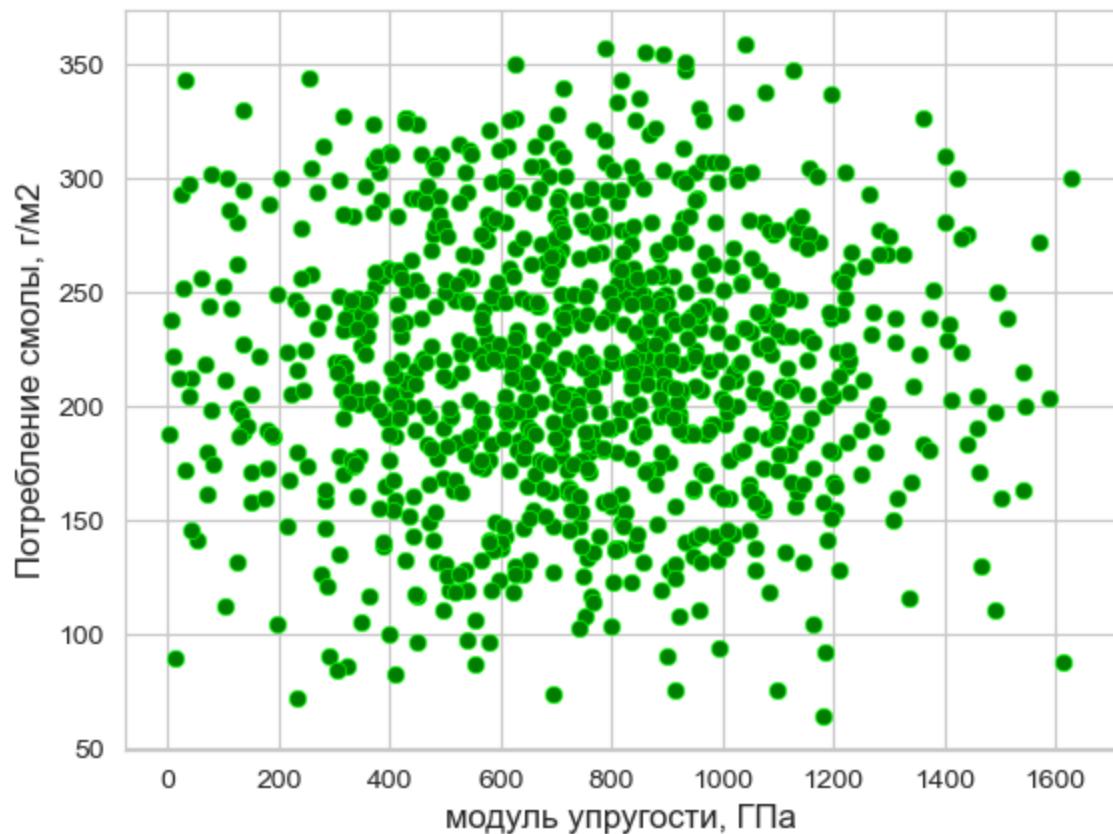
Зависимость



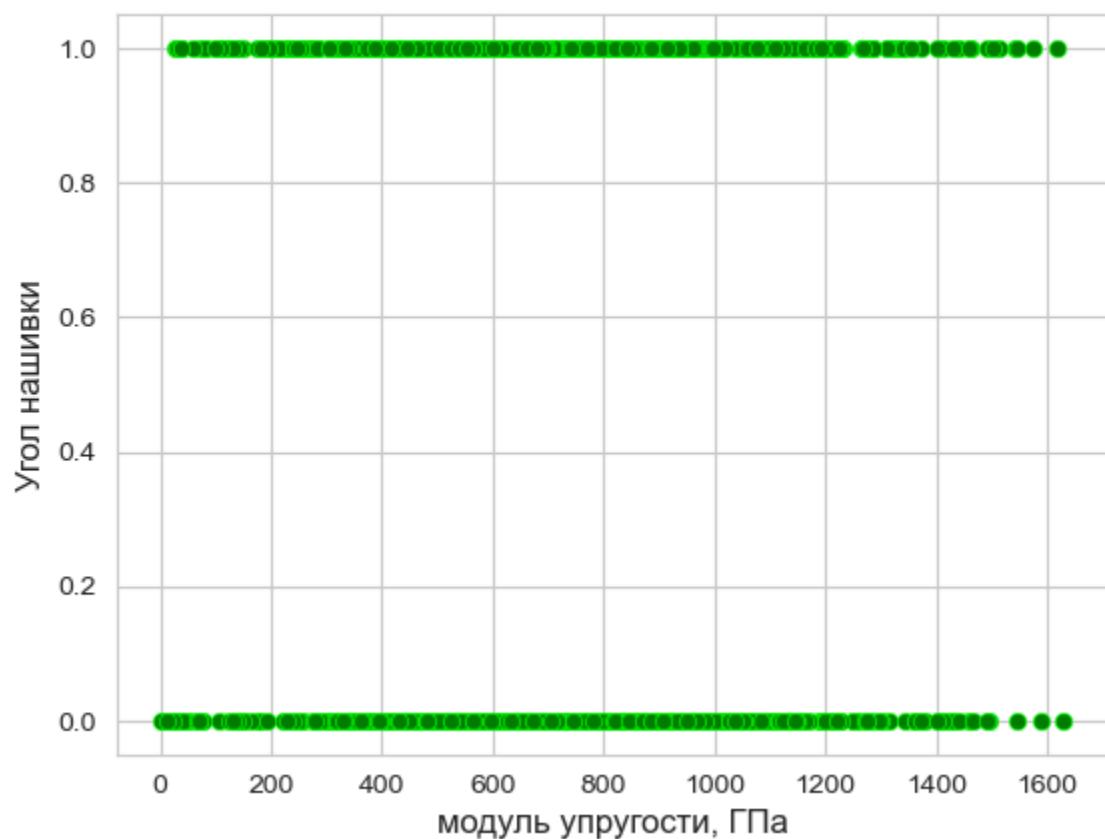
Зависимость



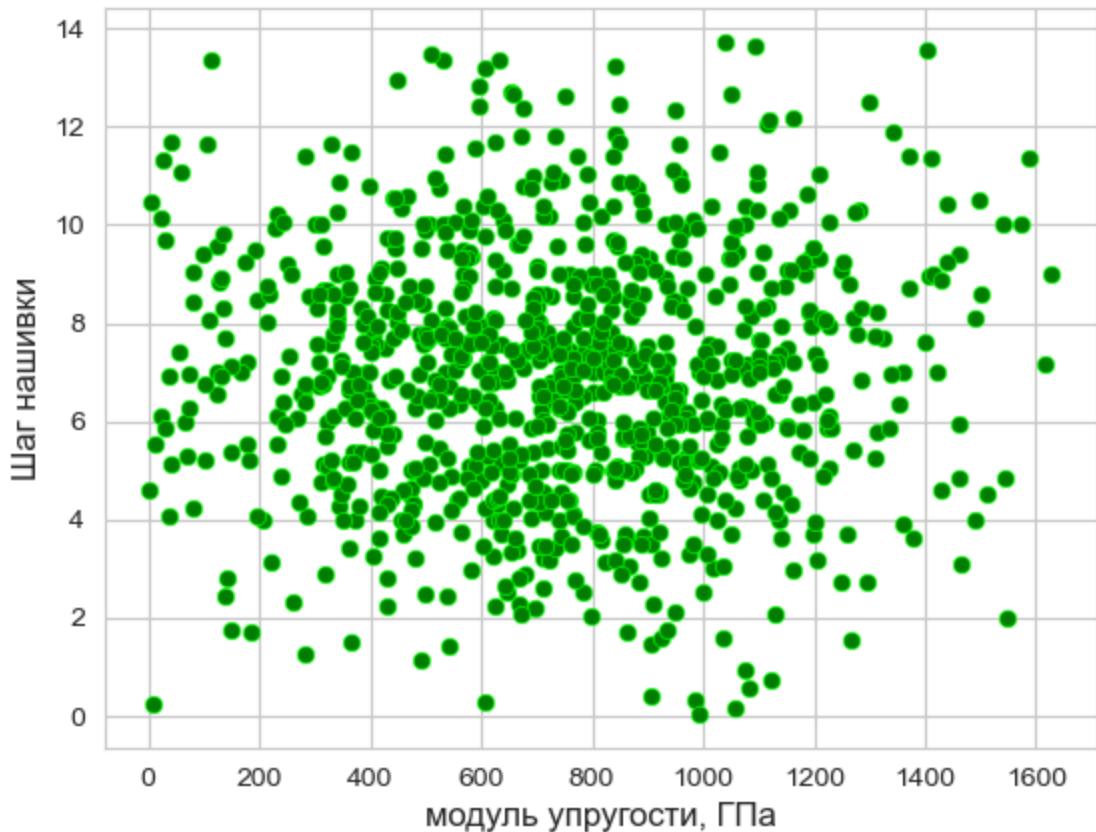
Зависимость



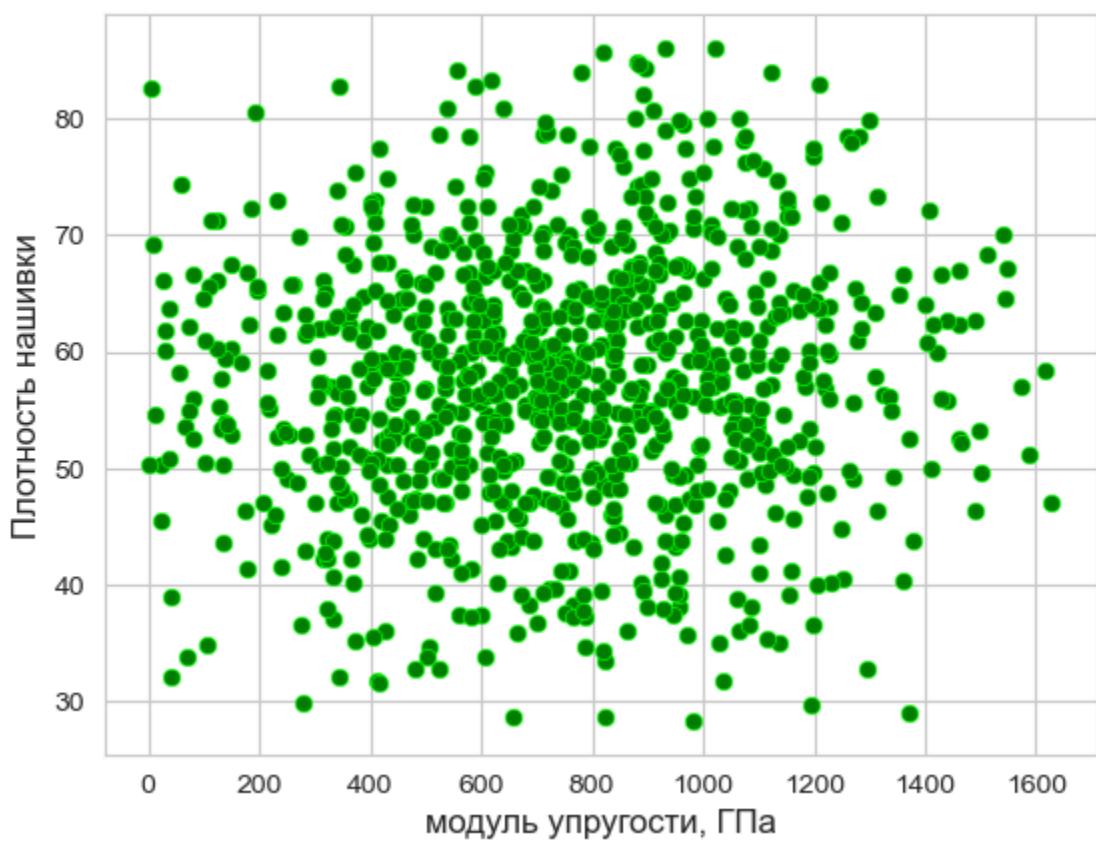
Зависимость



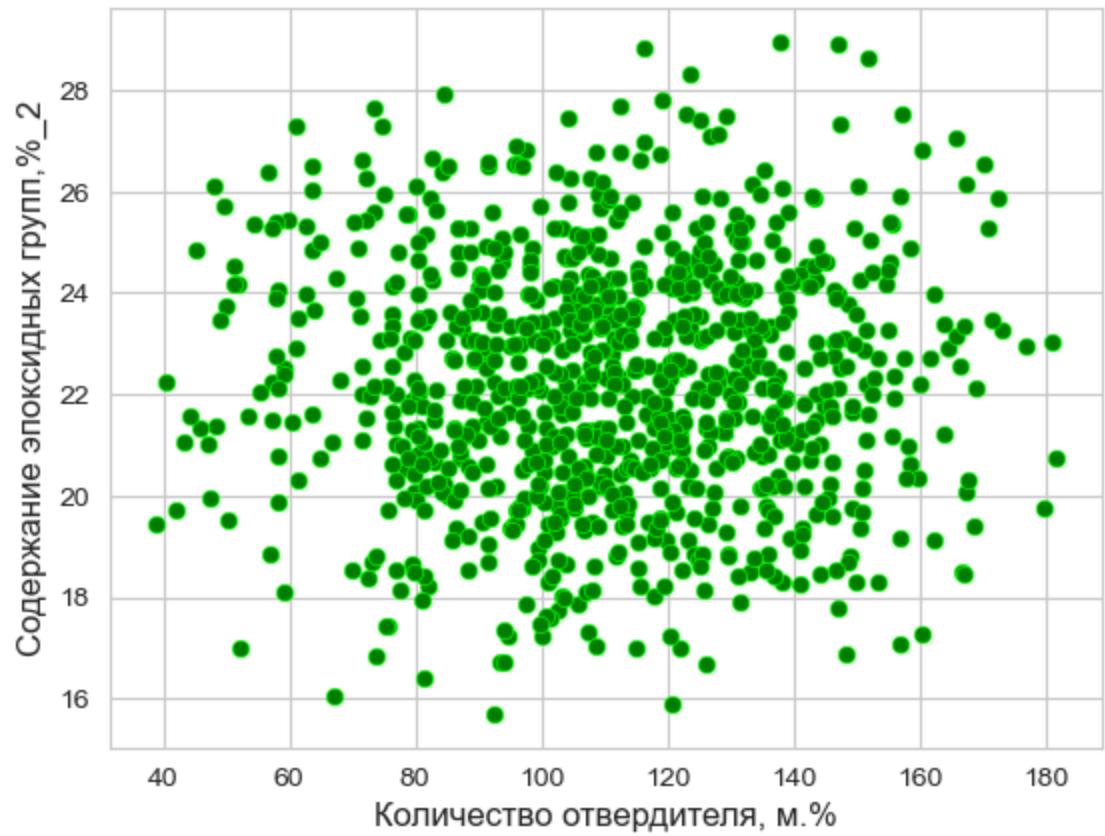
Зависимость



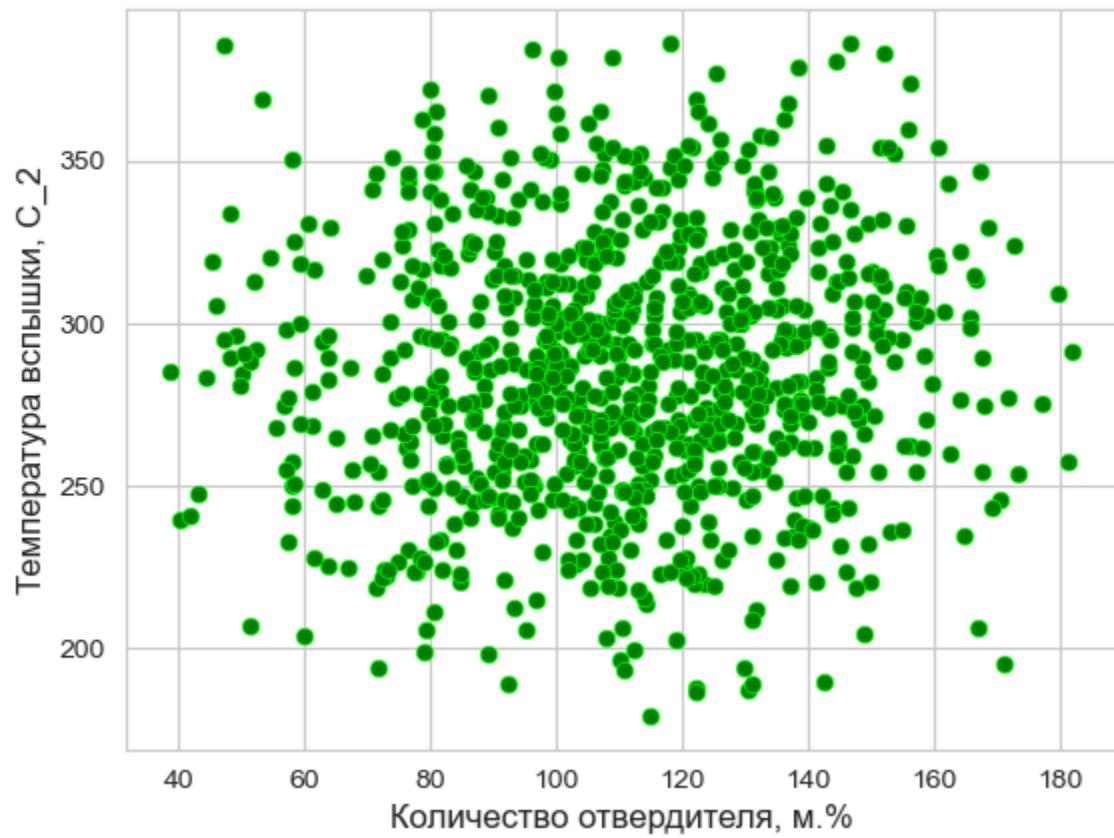
Зависимость



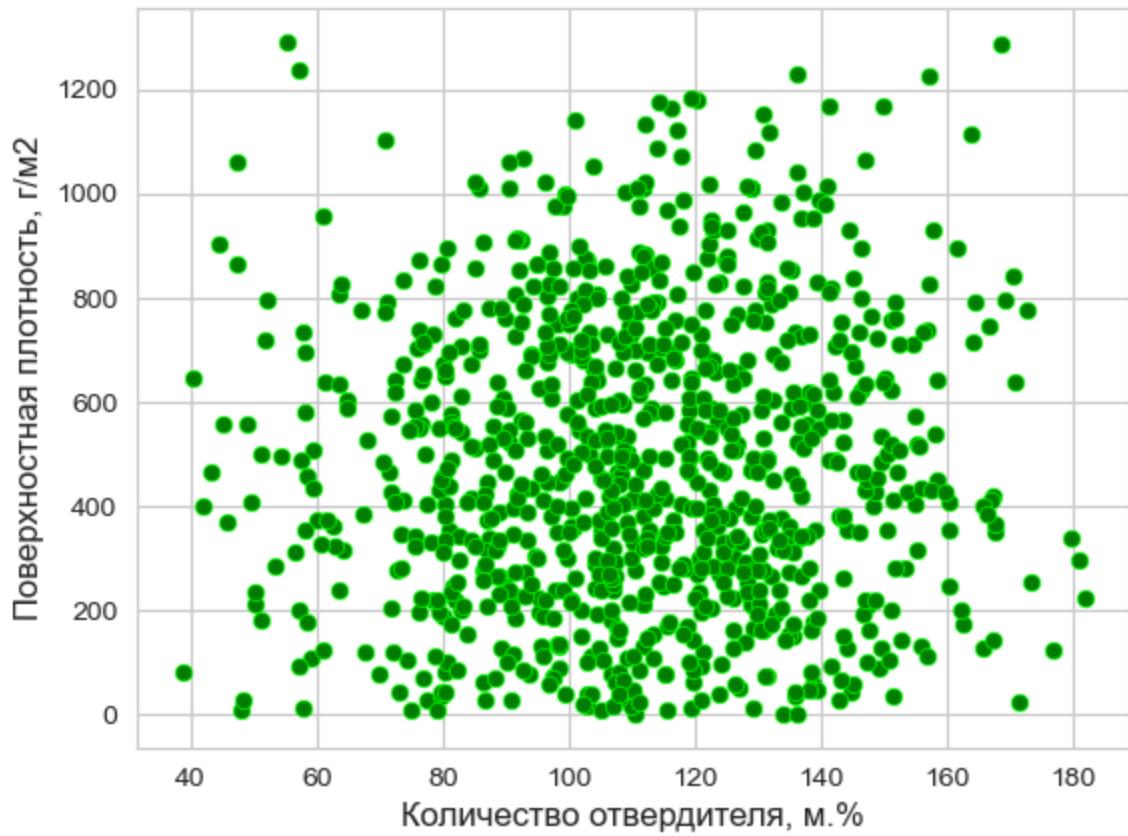
Зависимость



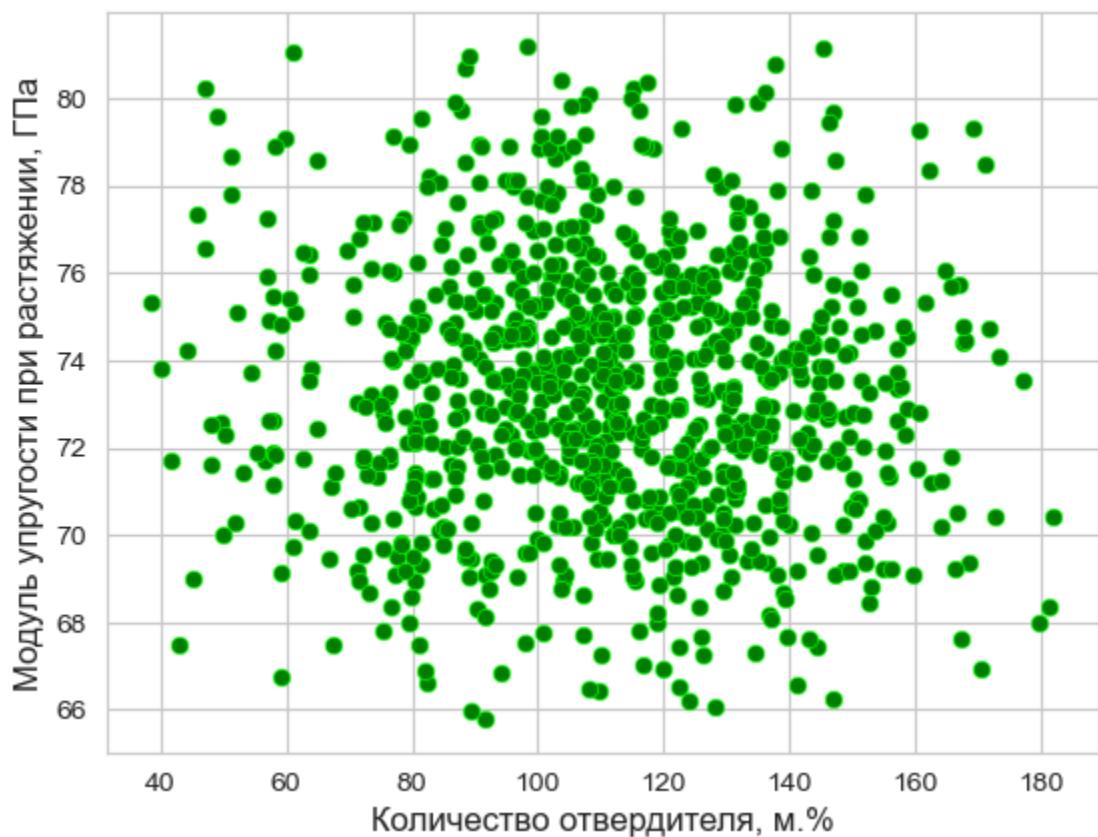
Зависимость



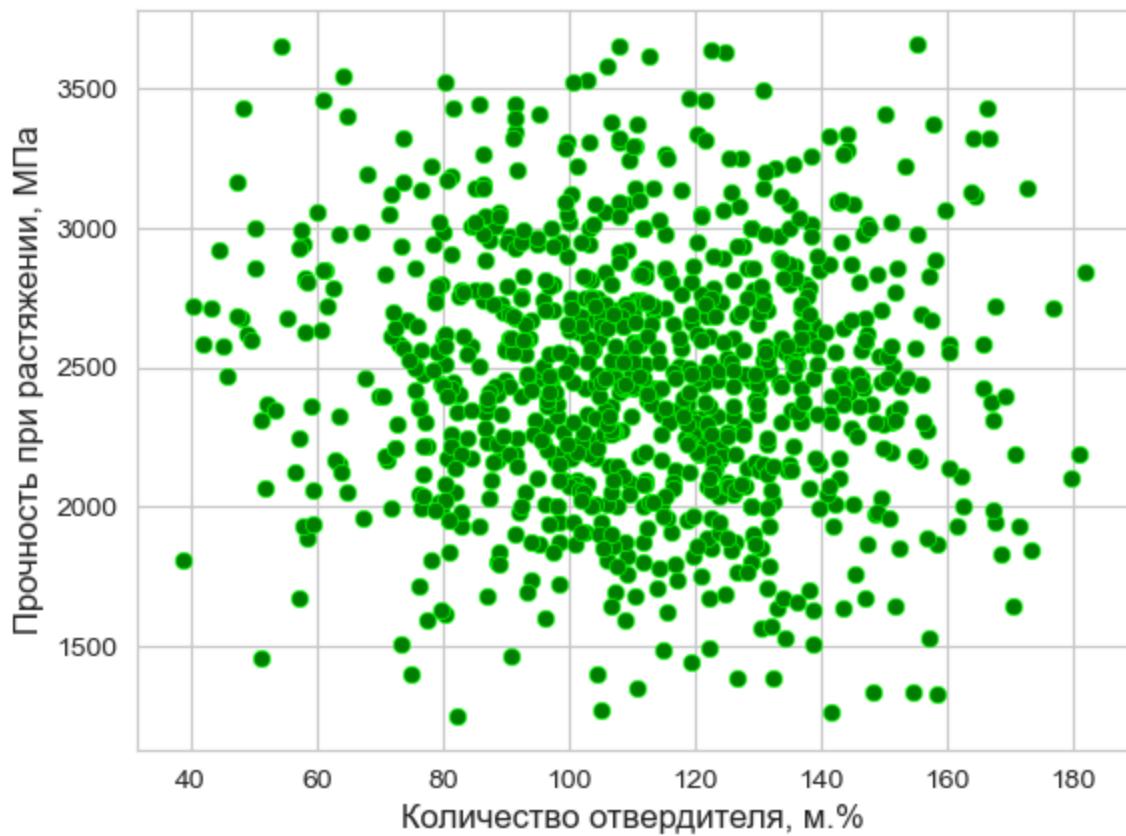
Зависимость



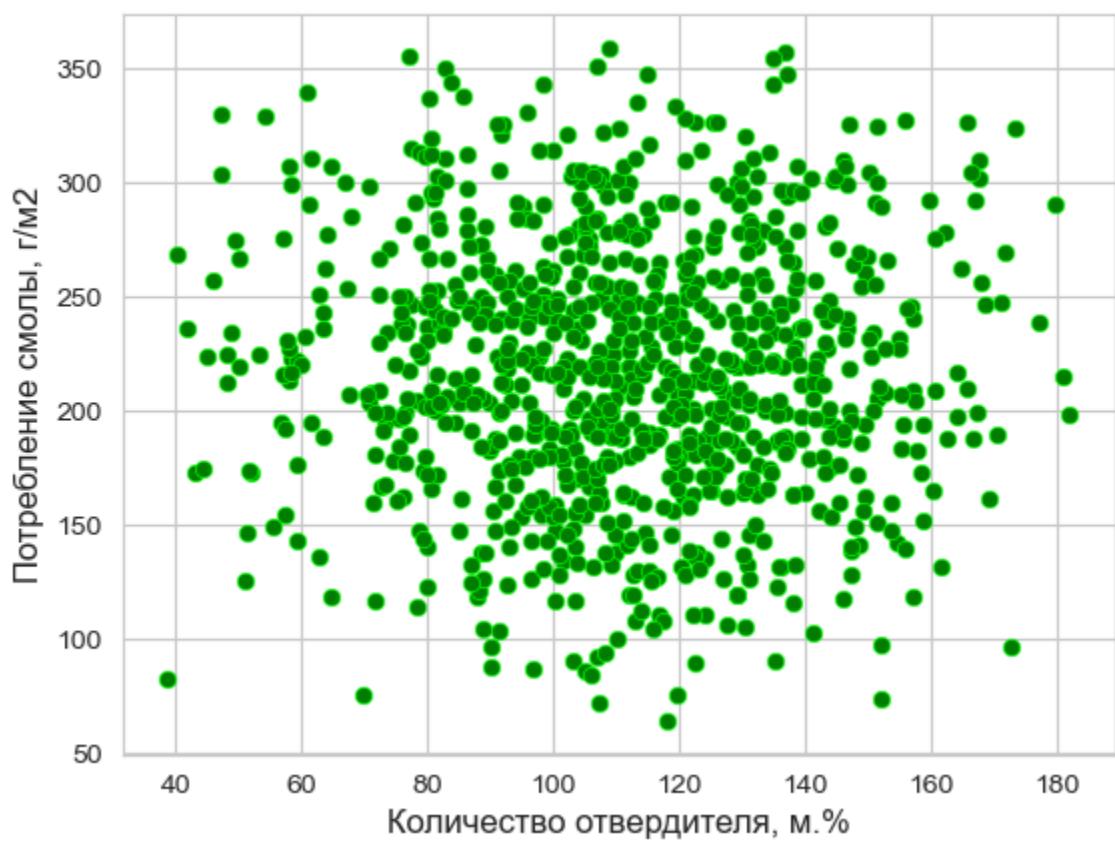
Зависимость



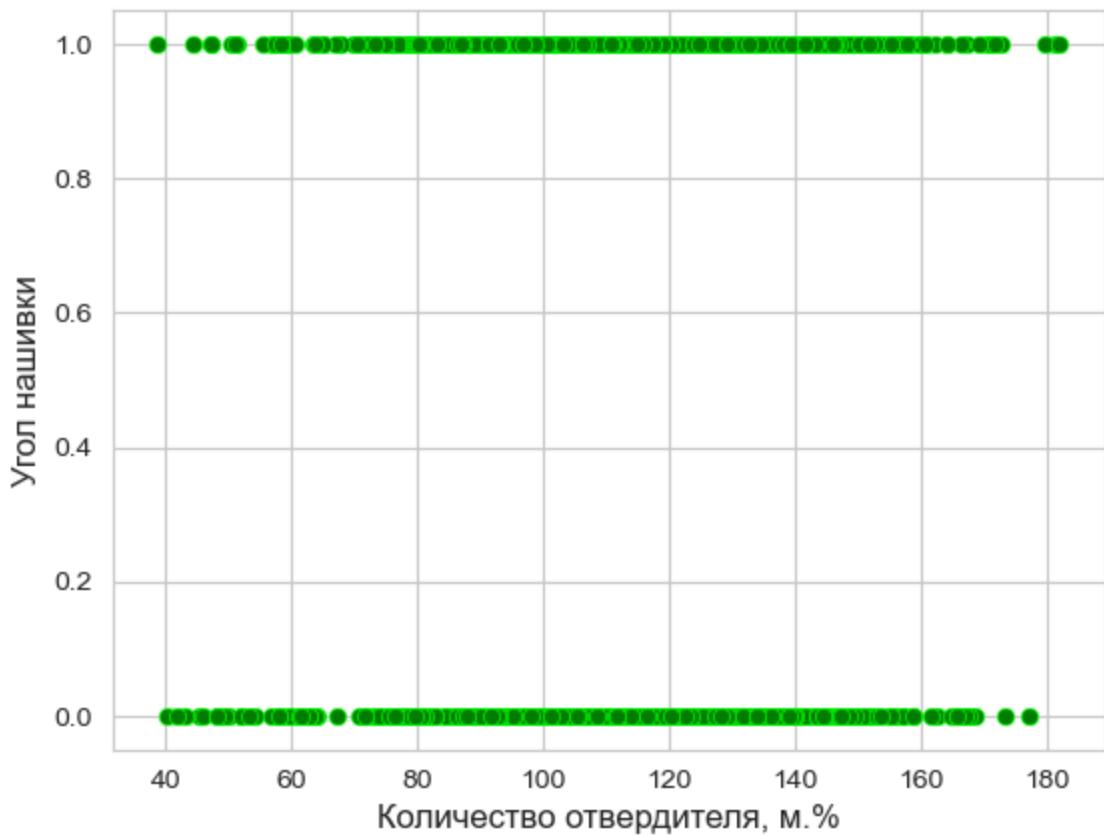
Зависимость



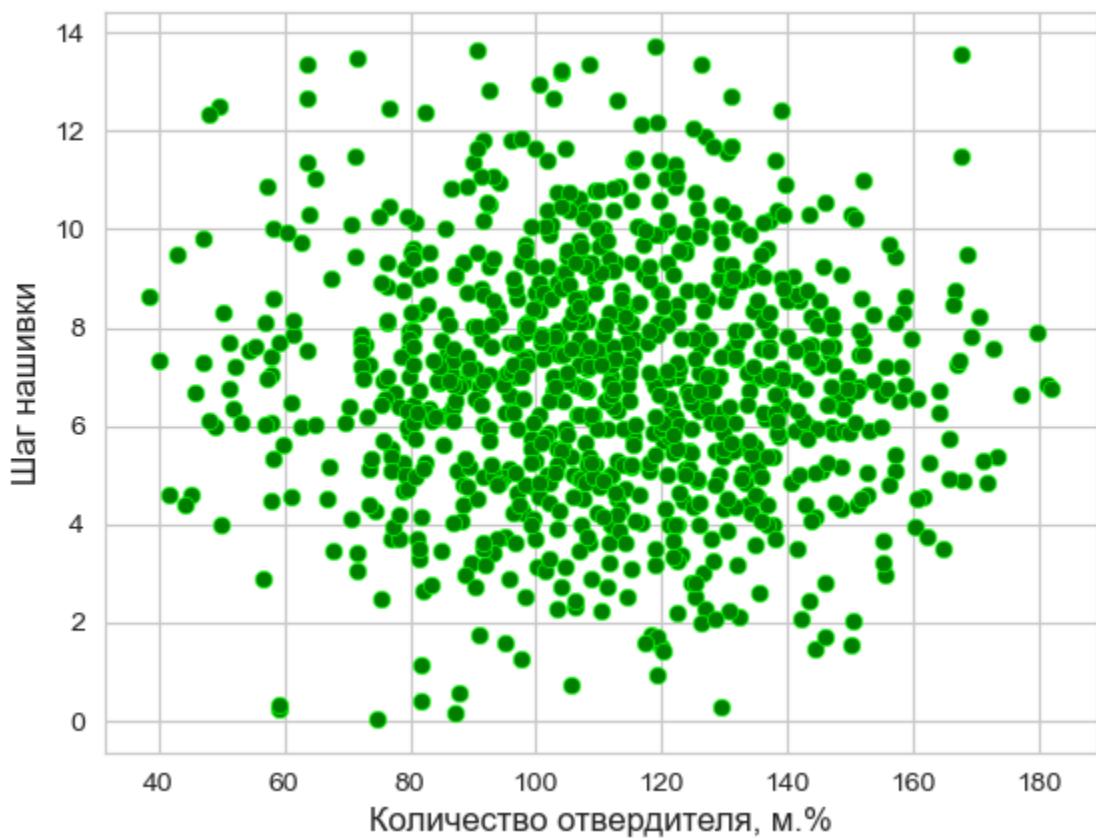
Зависимость



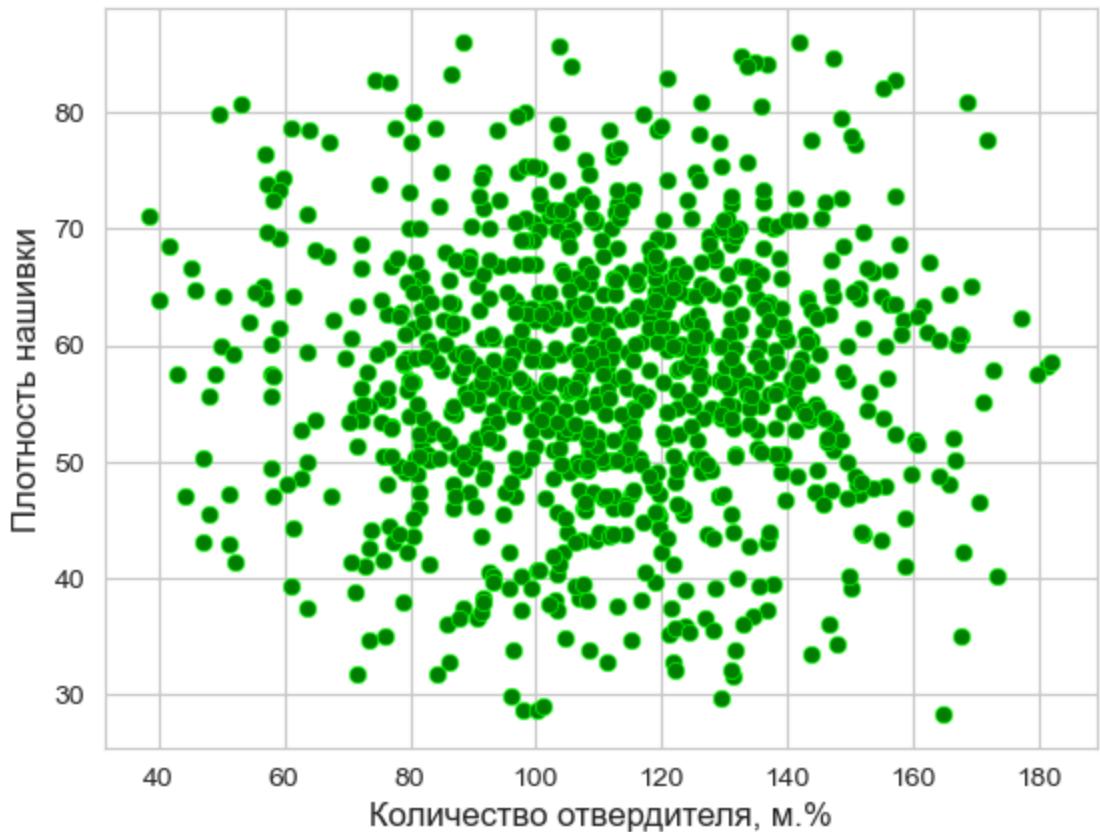
Зависимость



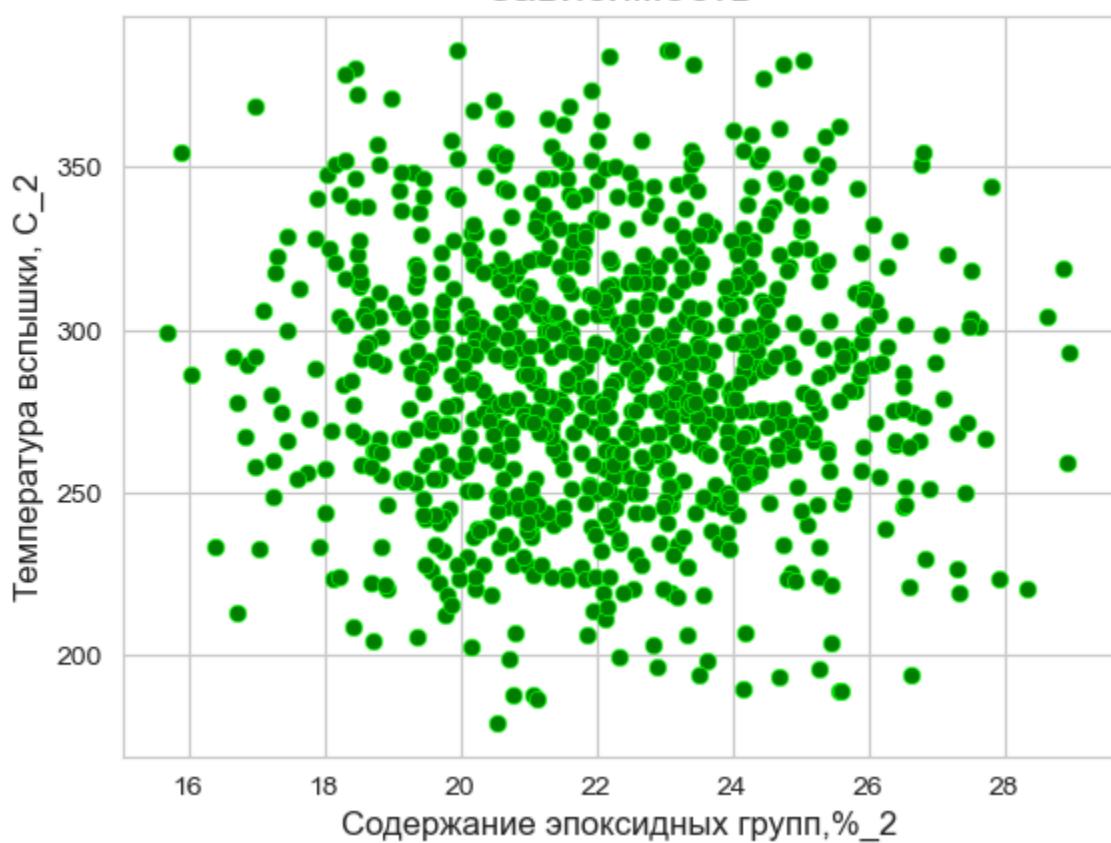
Зависимость



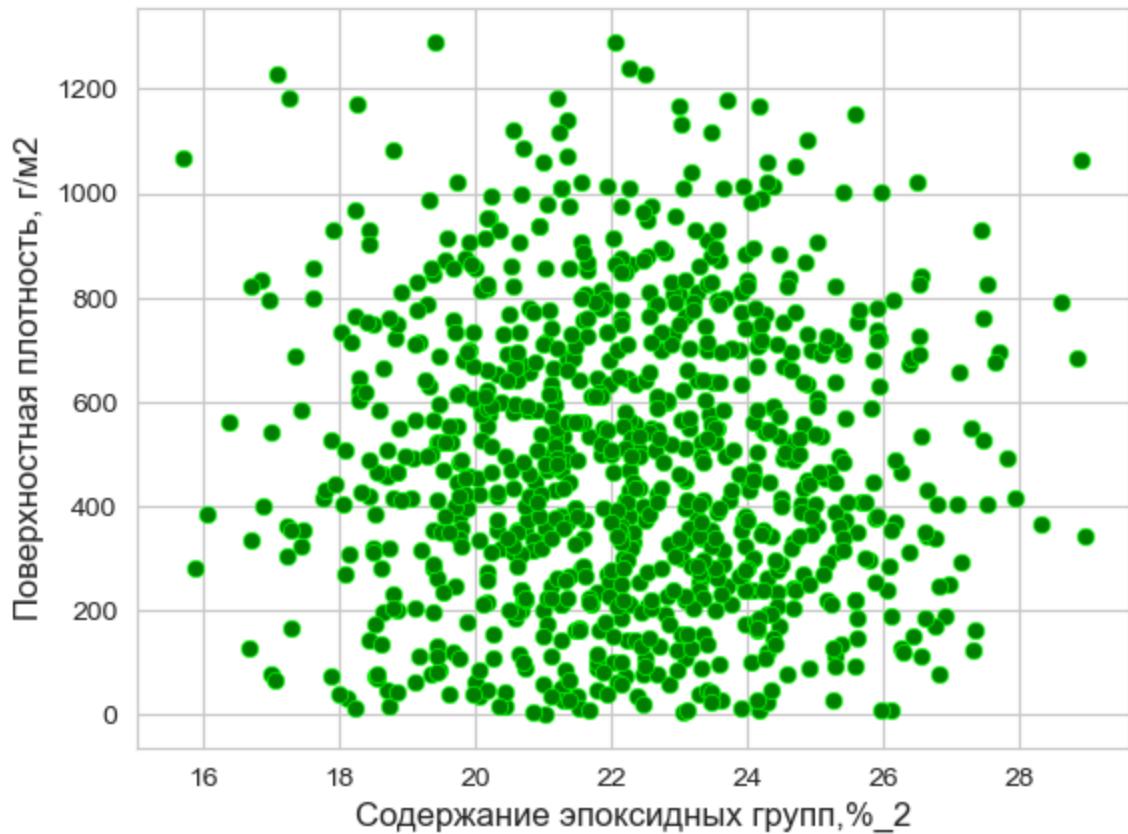
Зависимость



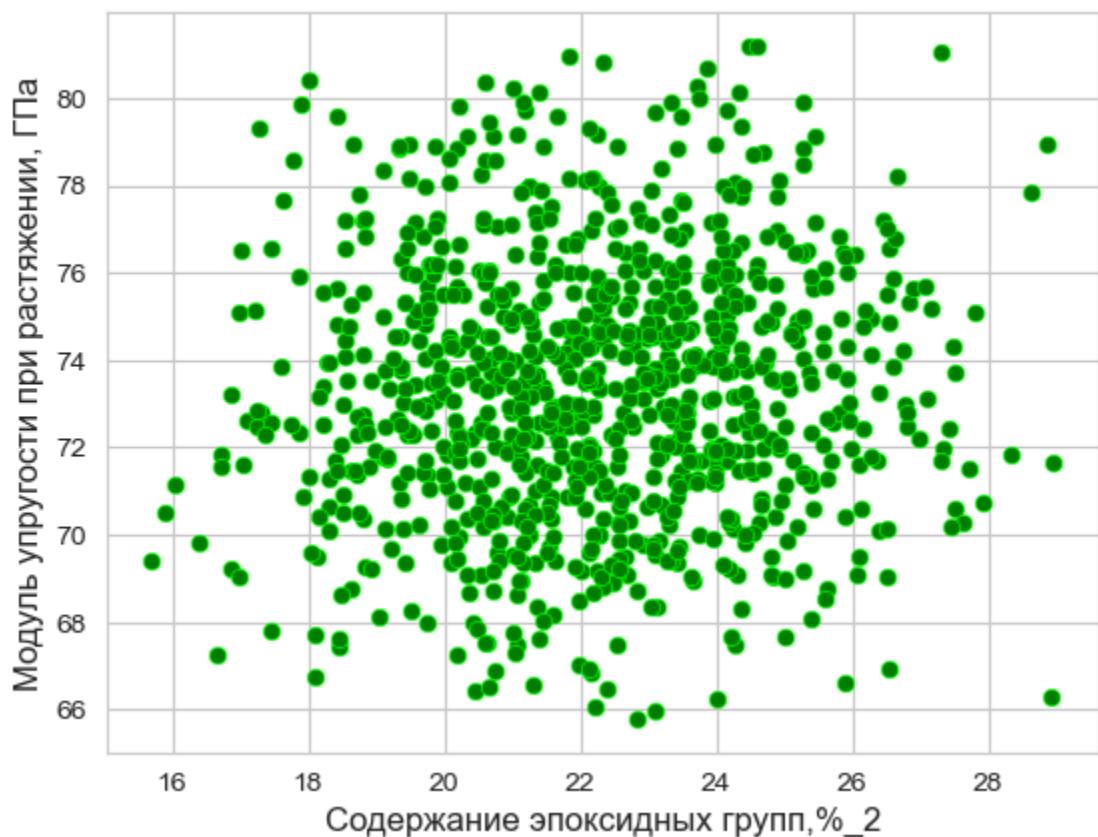
Зависимость



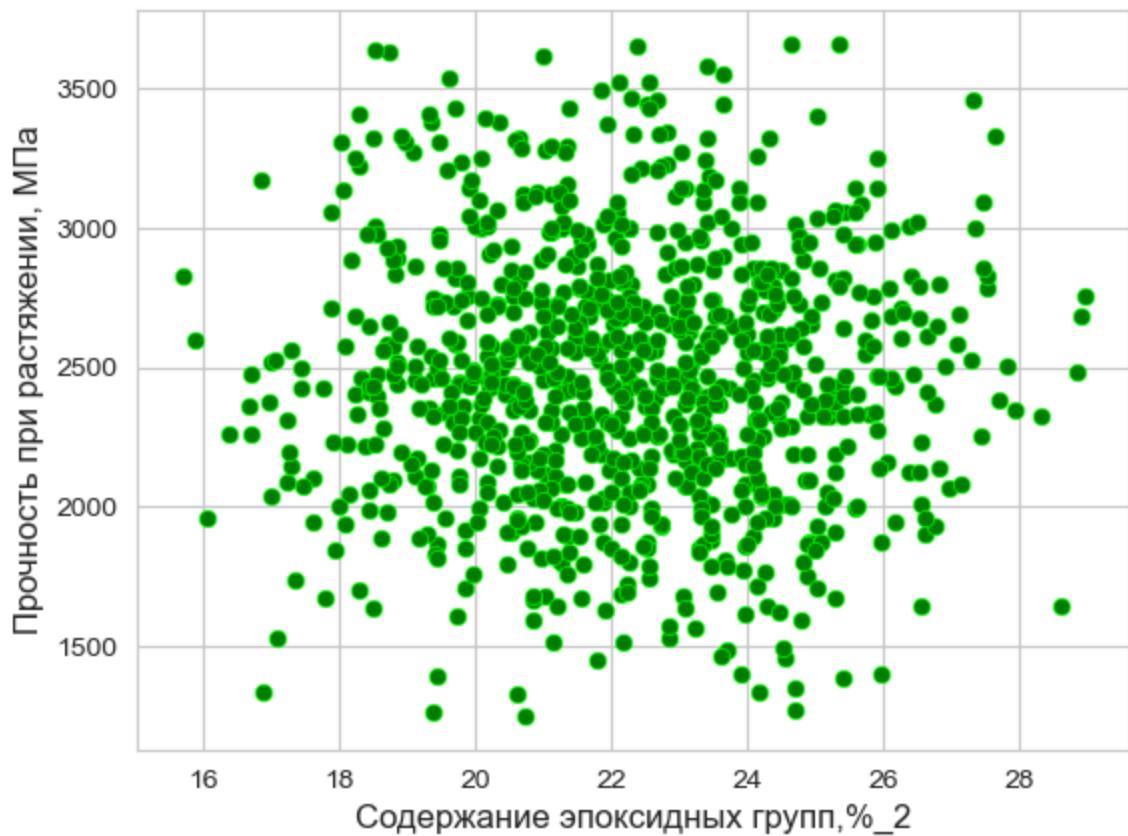
Зависимость



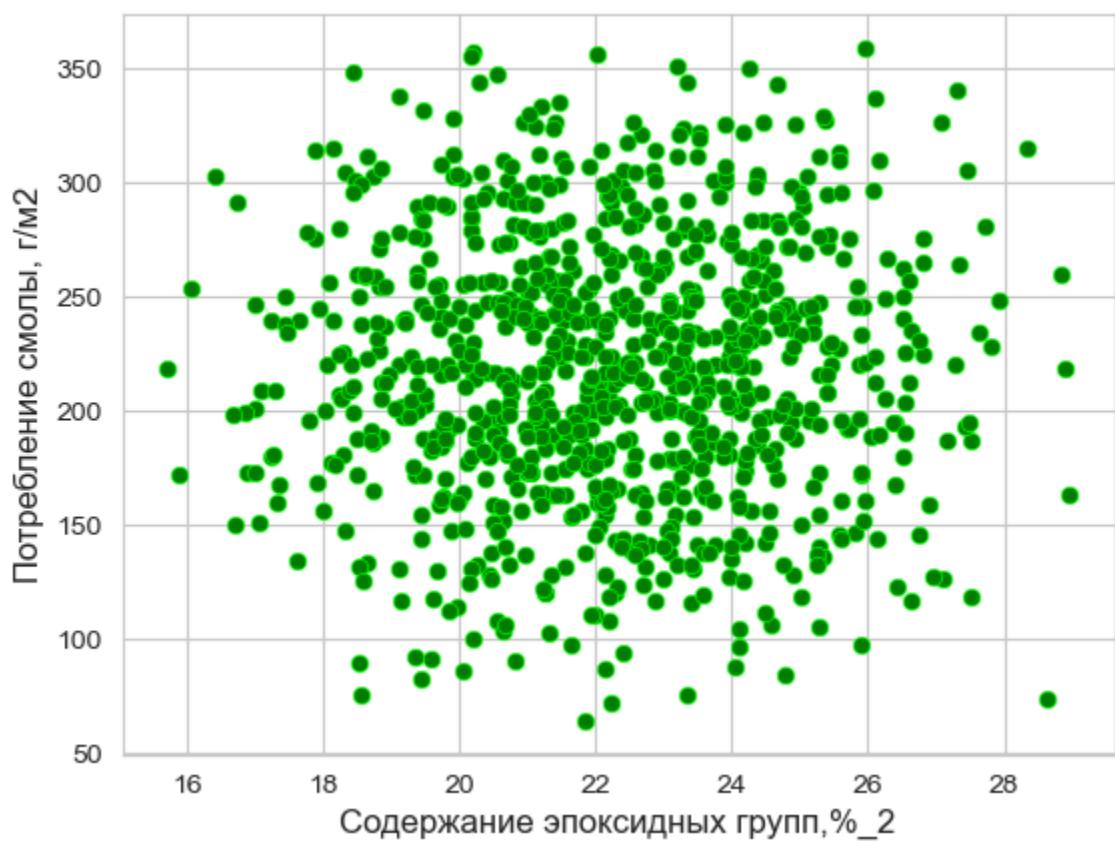
Зависимость



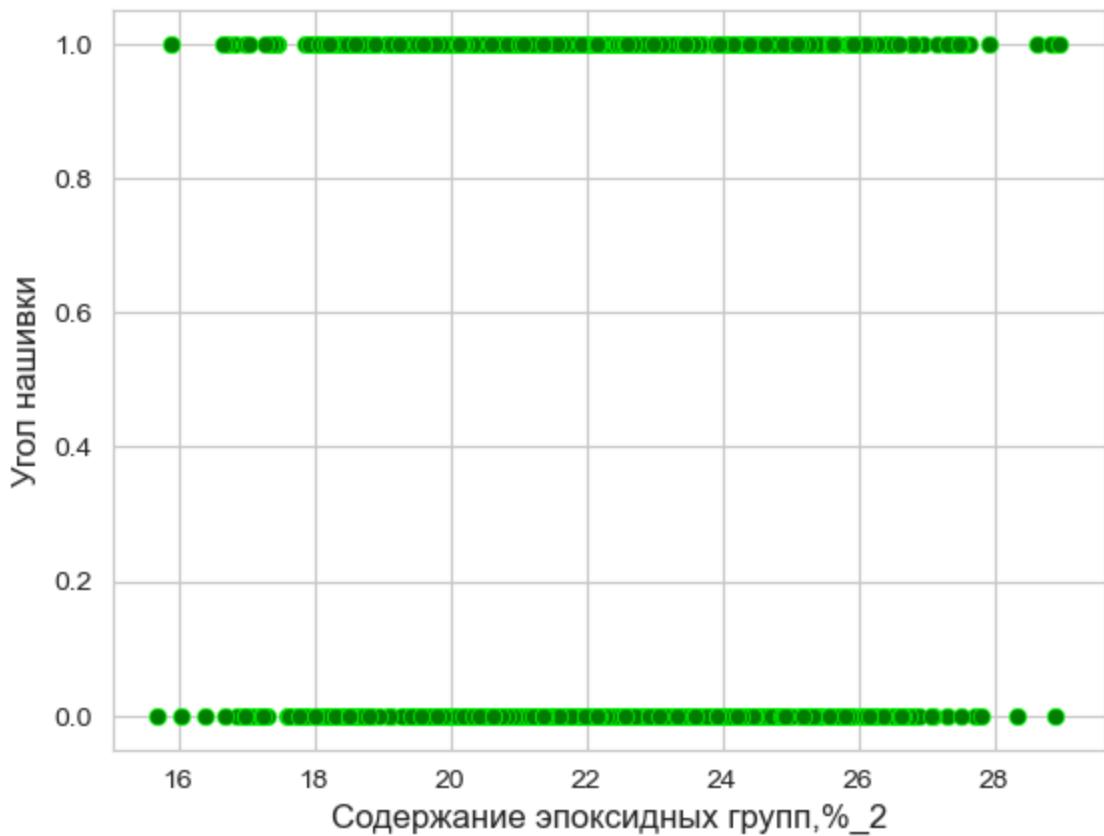
Зависимость



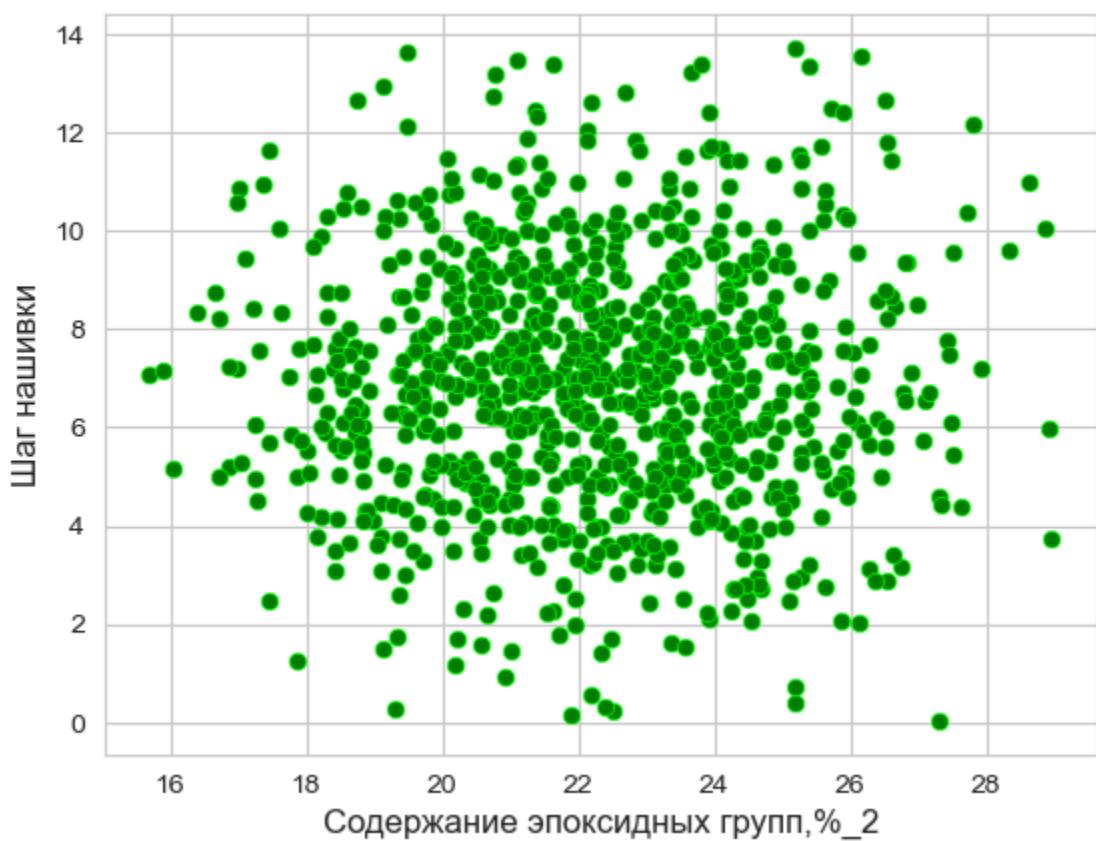
Зависимость



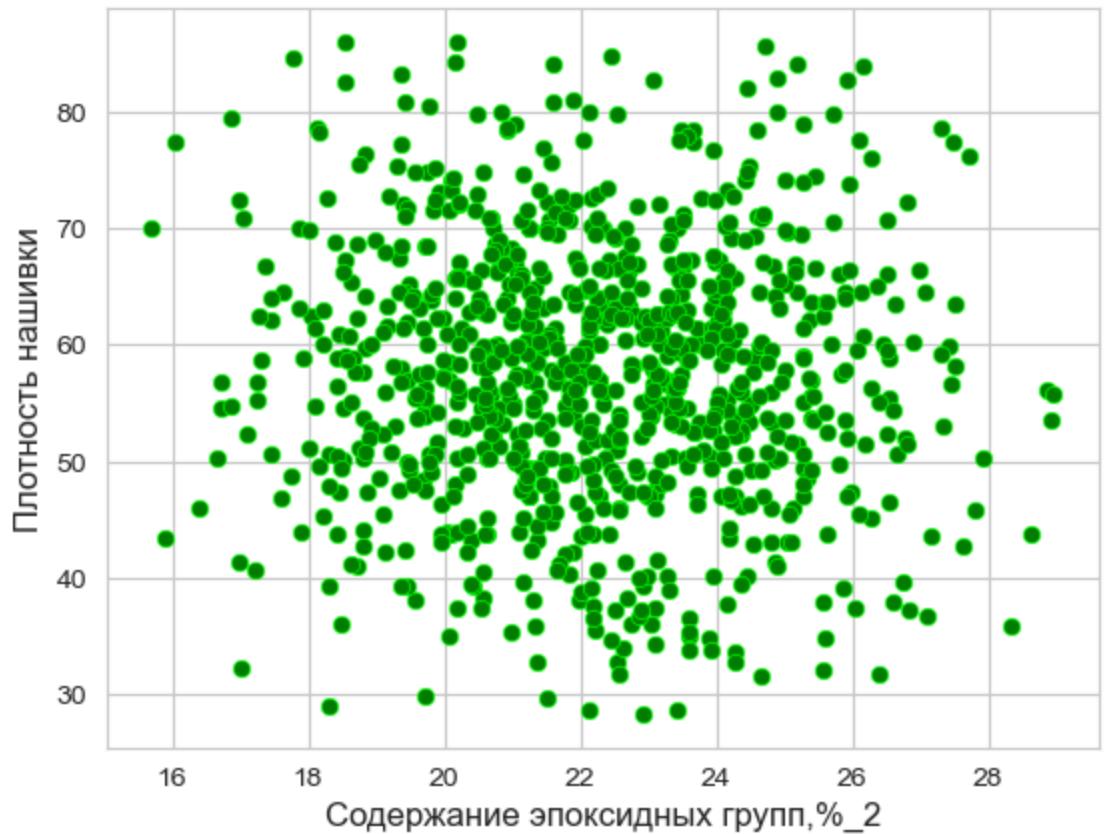
Зависимость



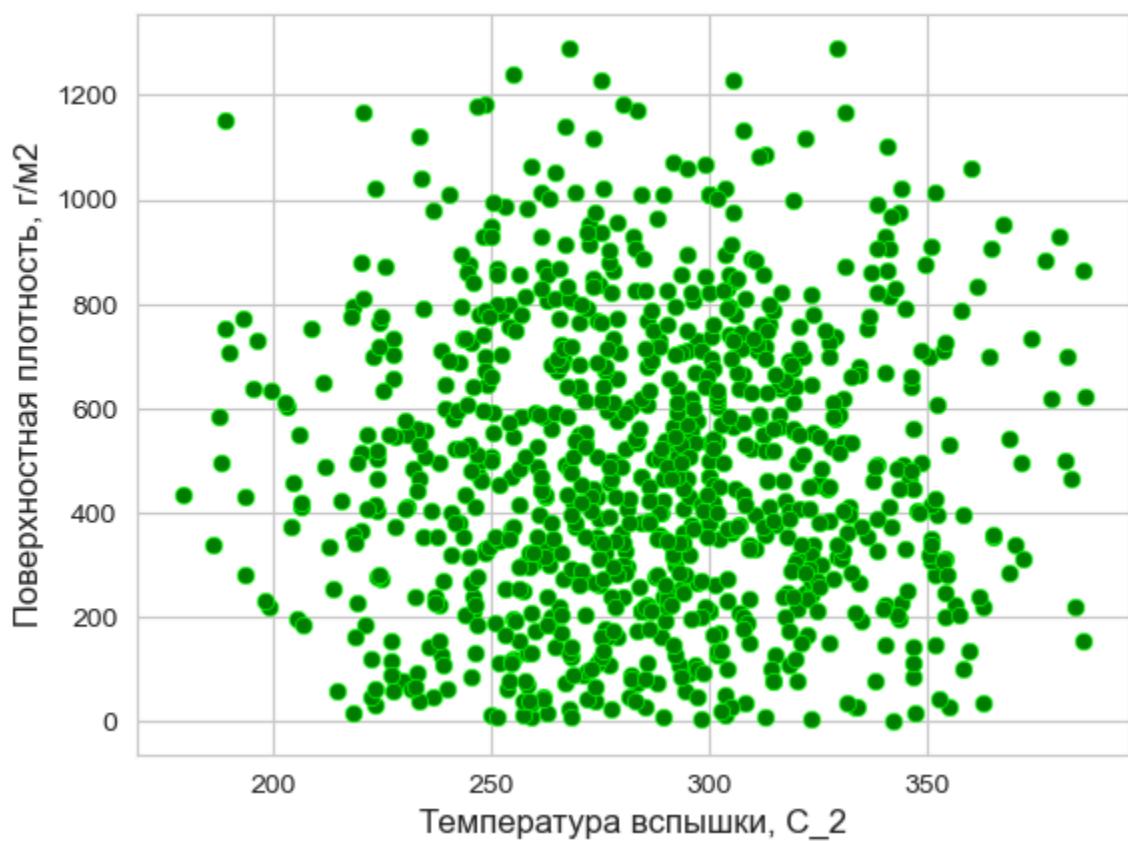
Зависимость



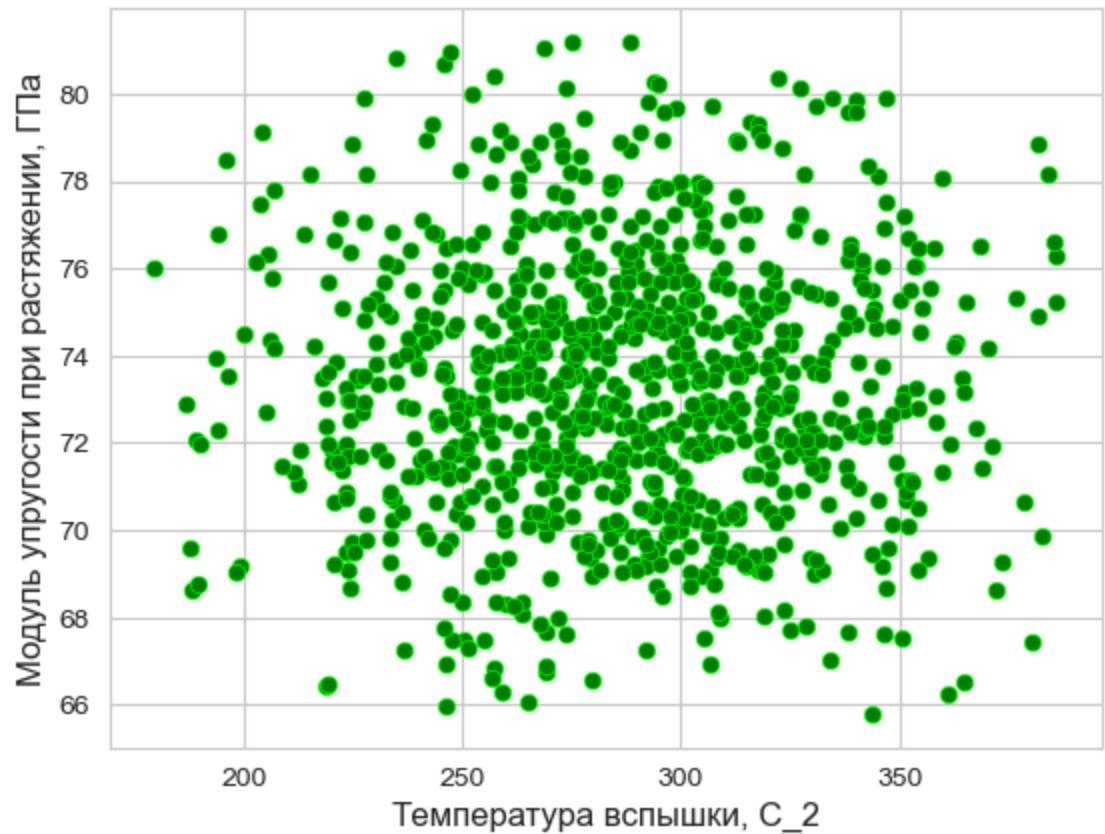
Зависимость



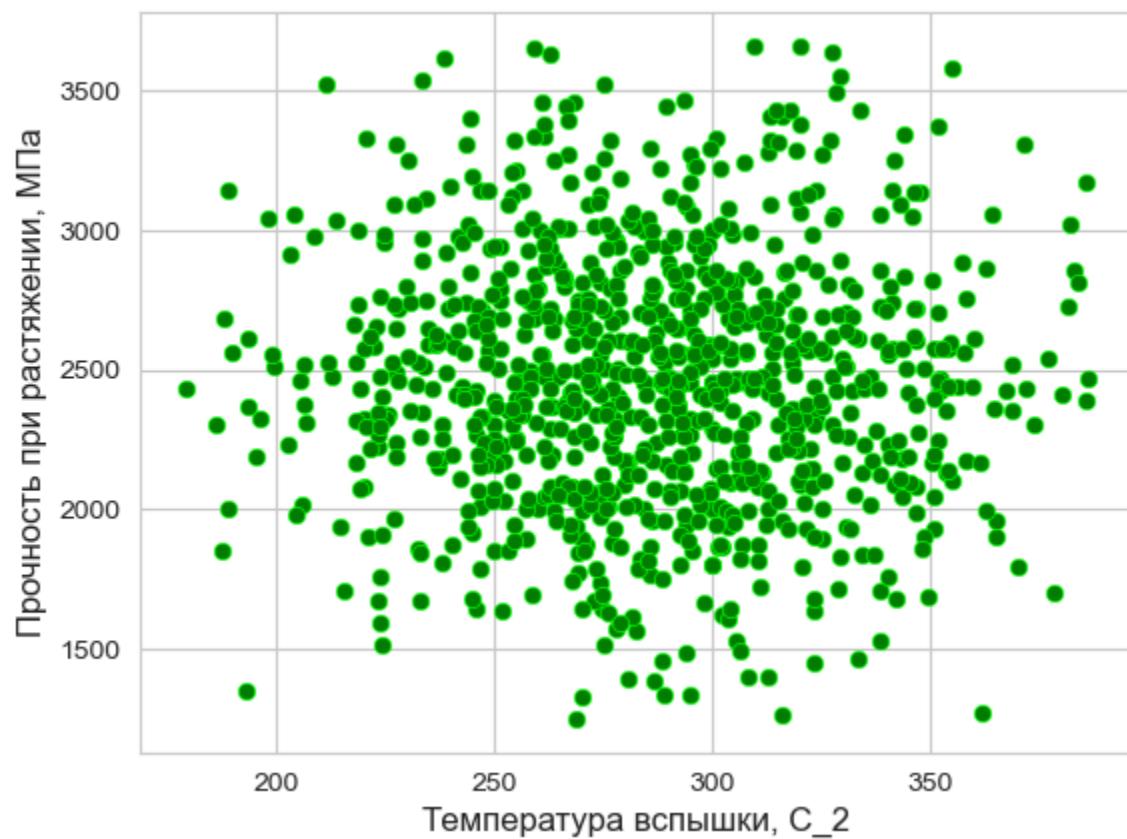
Зависимость



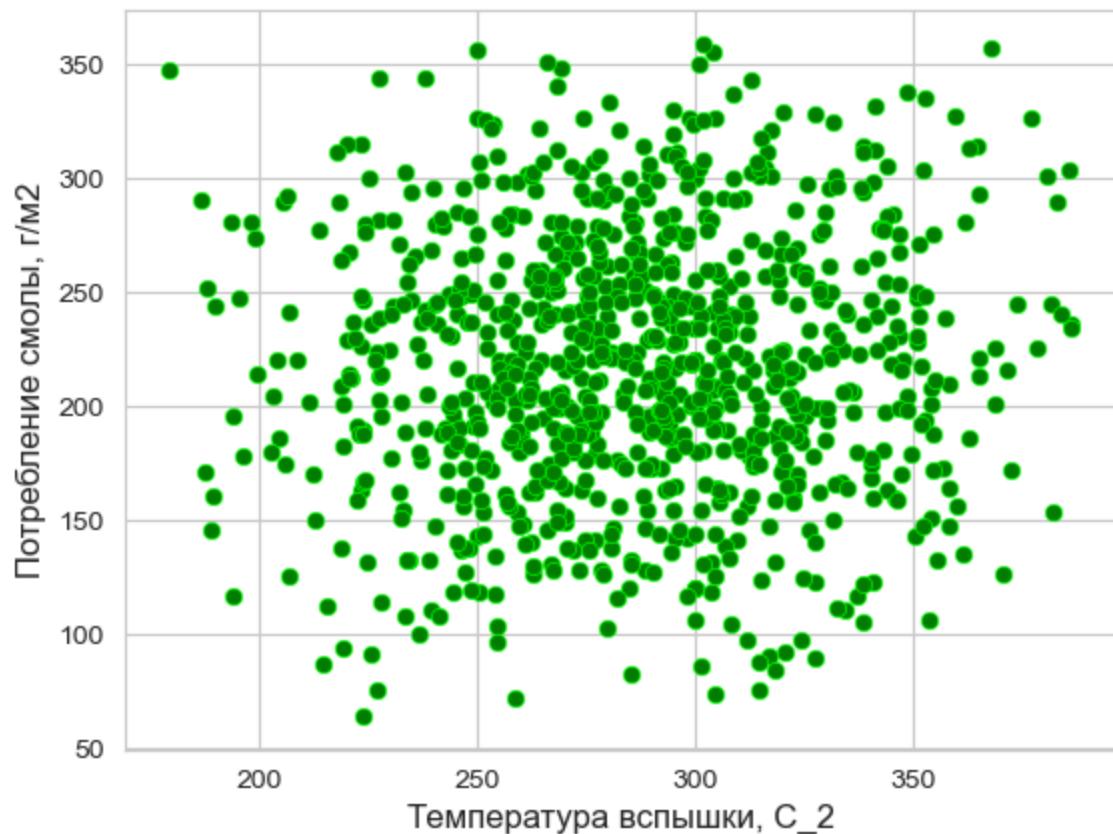
Зависимость



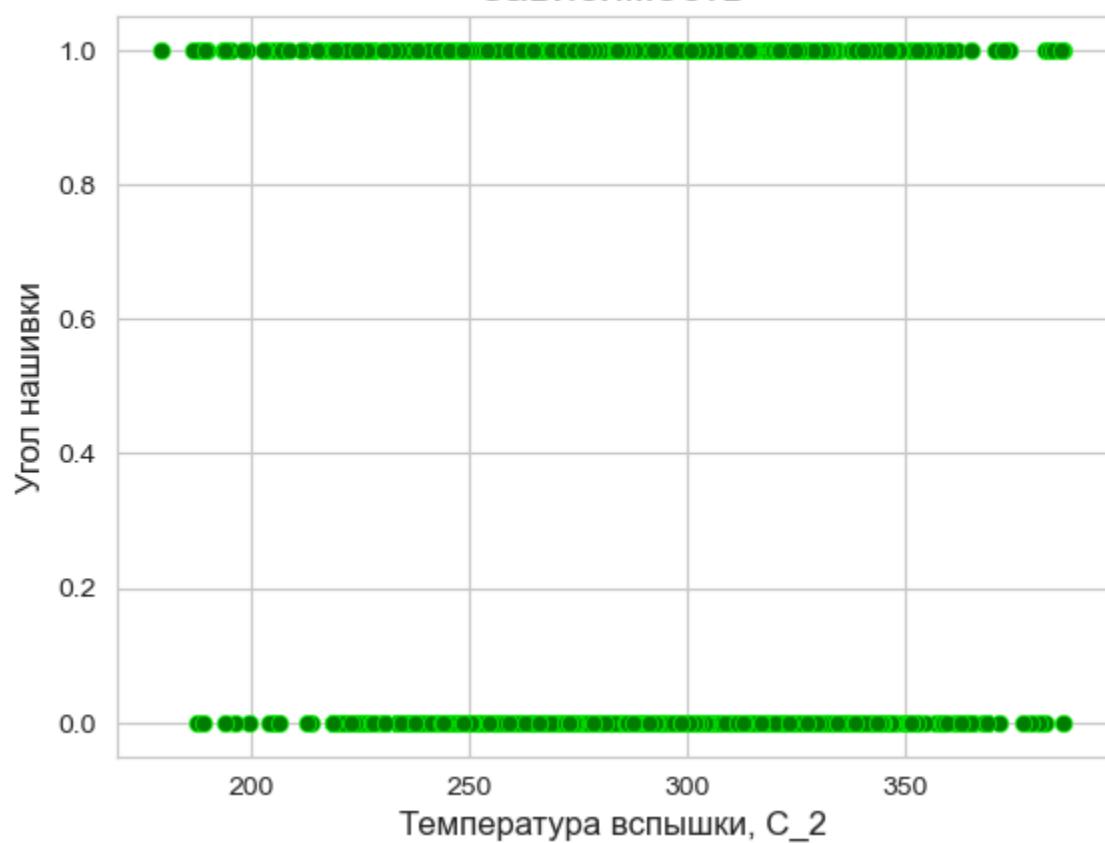
Зависимость



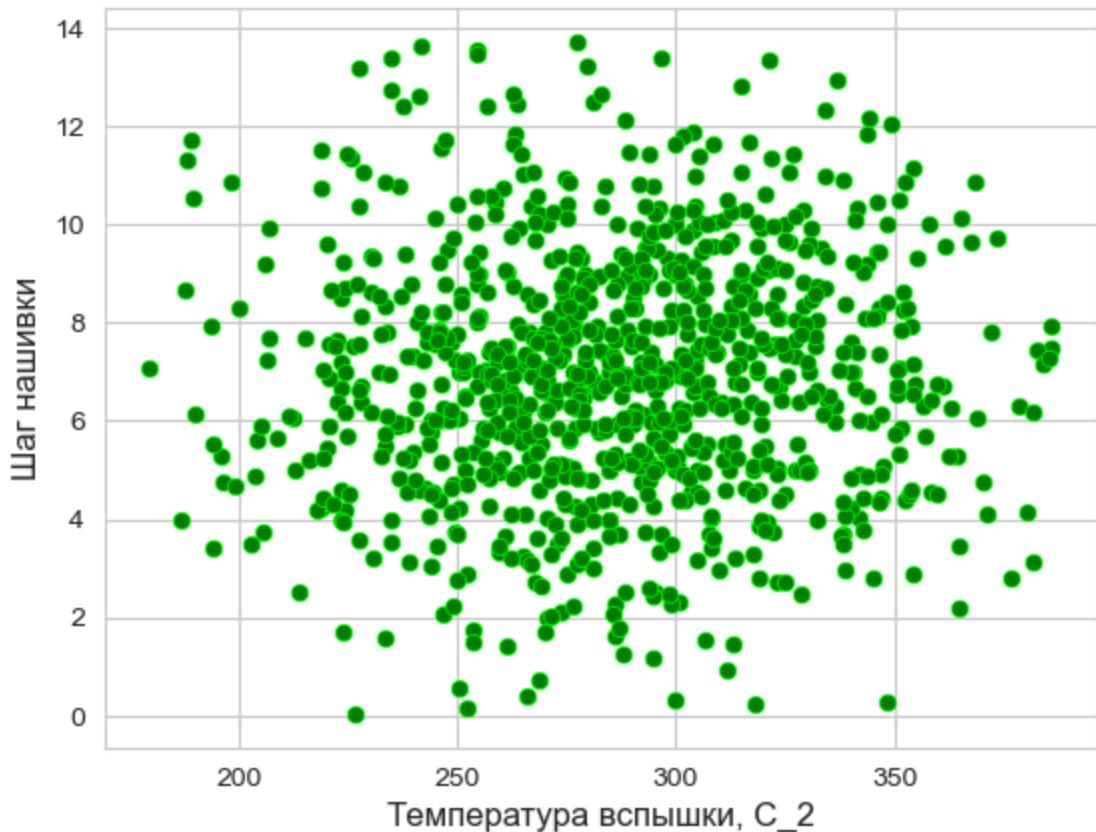
Зависимость



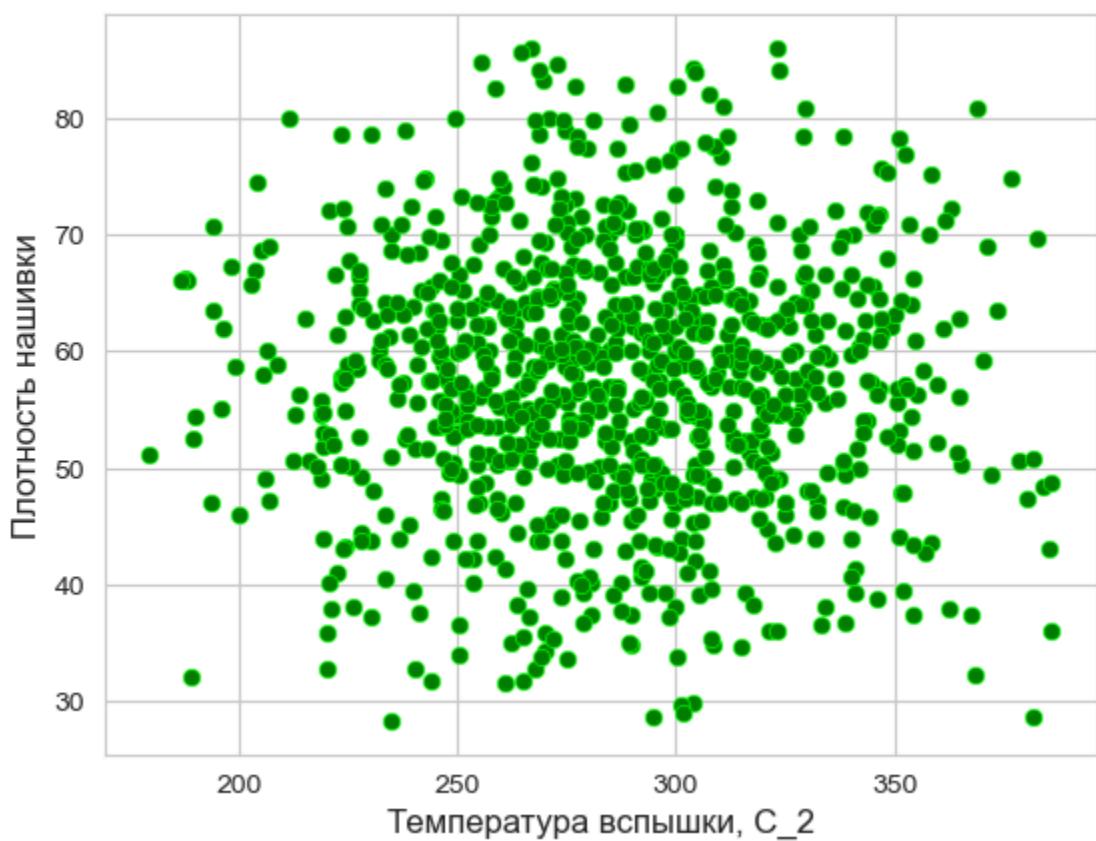
Зависимость



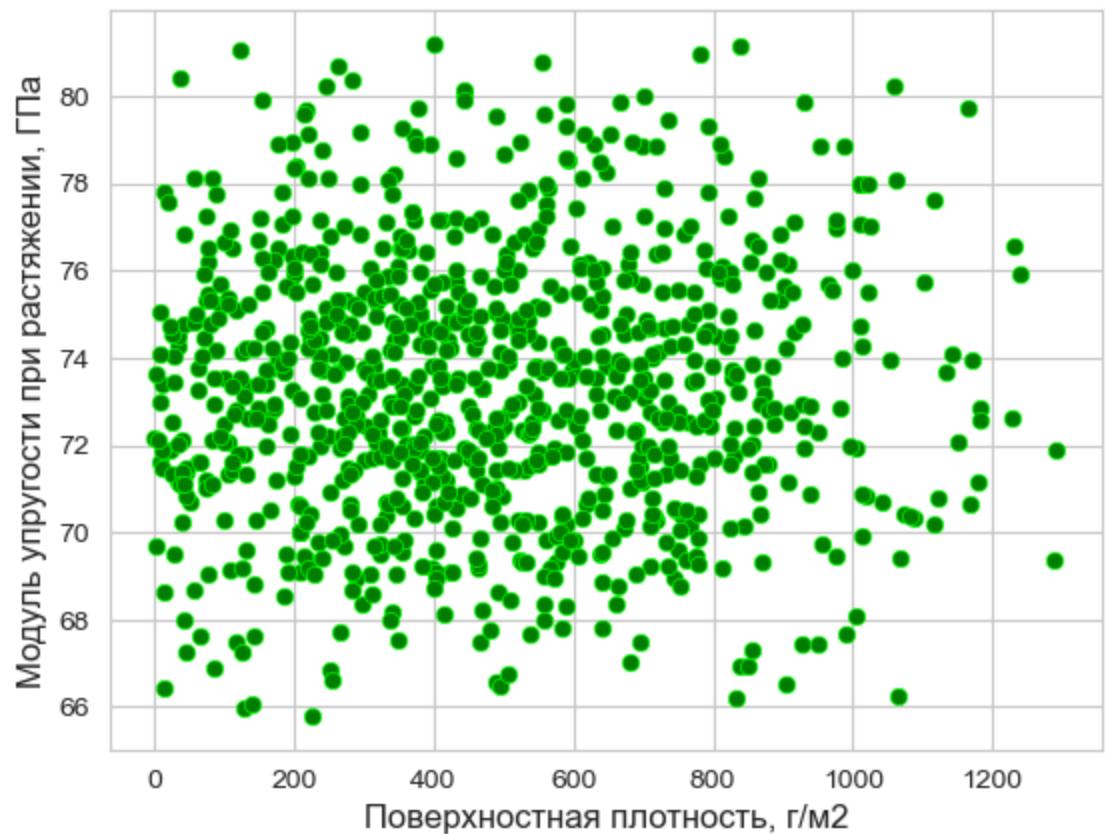
Зависимость



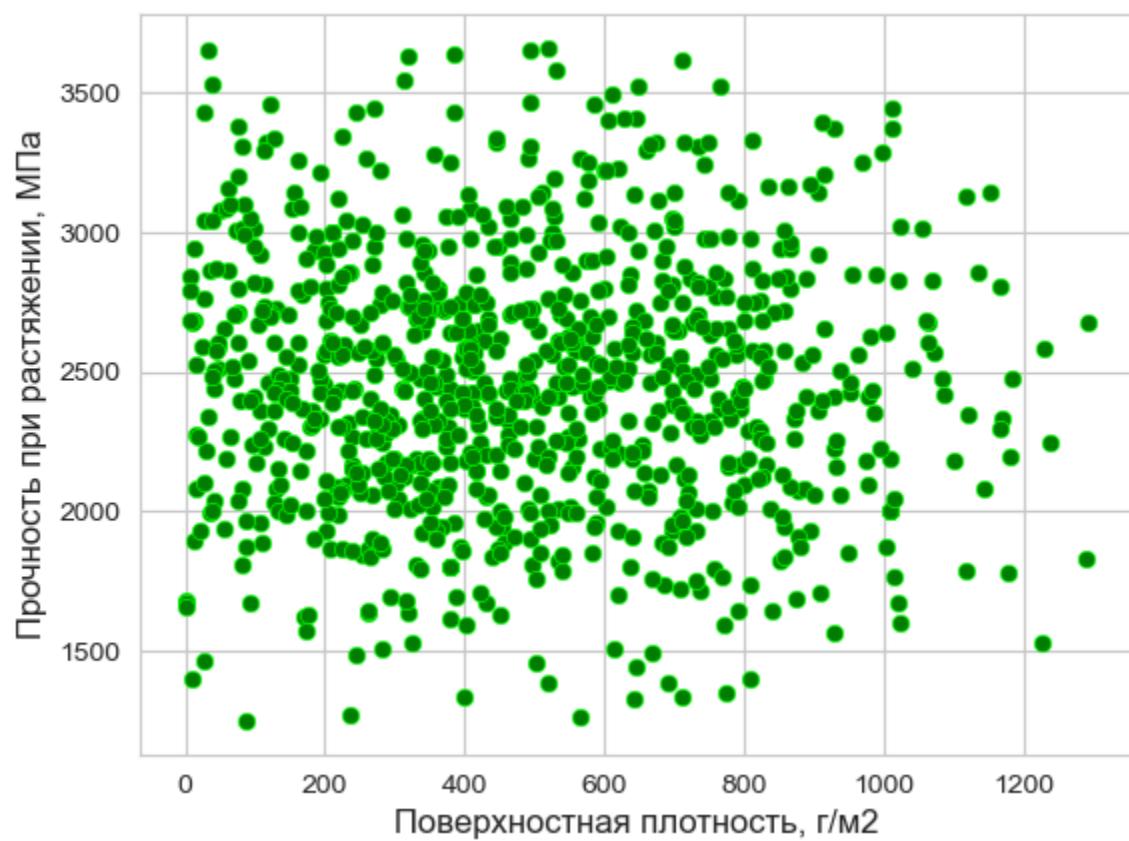
Зависимость



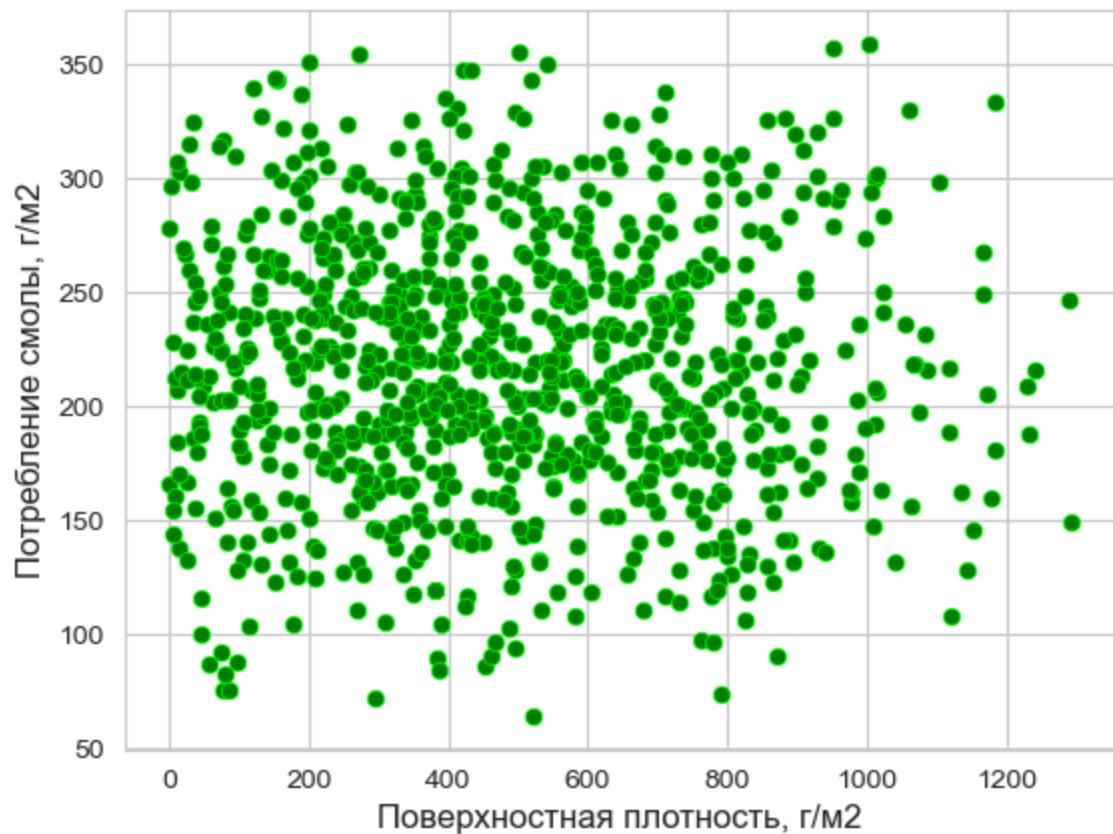
Зависимость



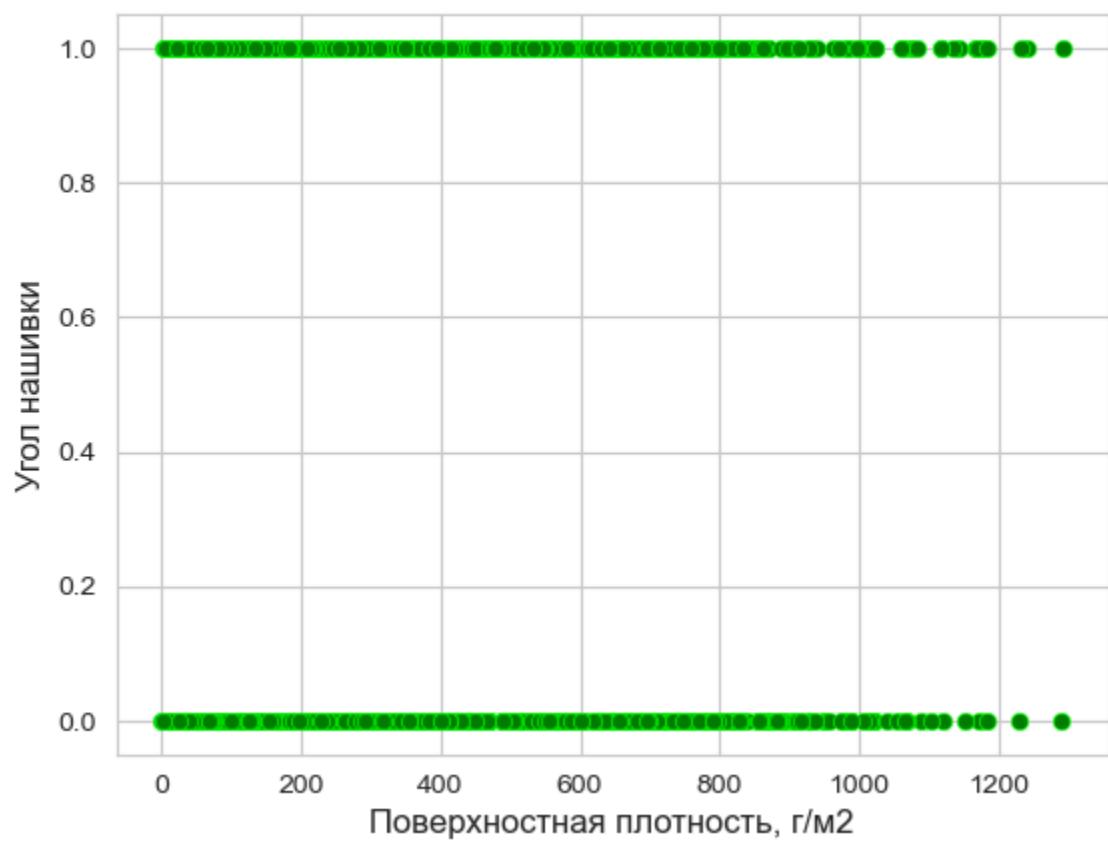
Зависимость



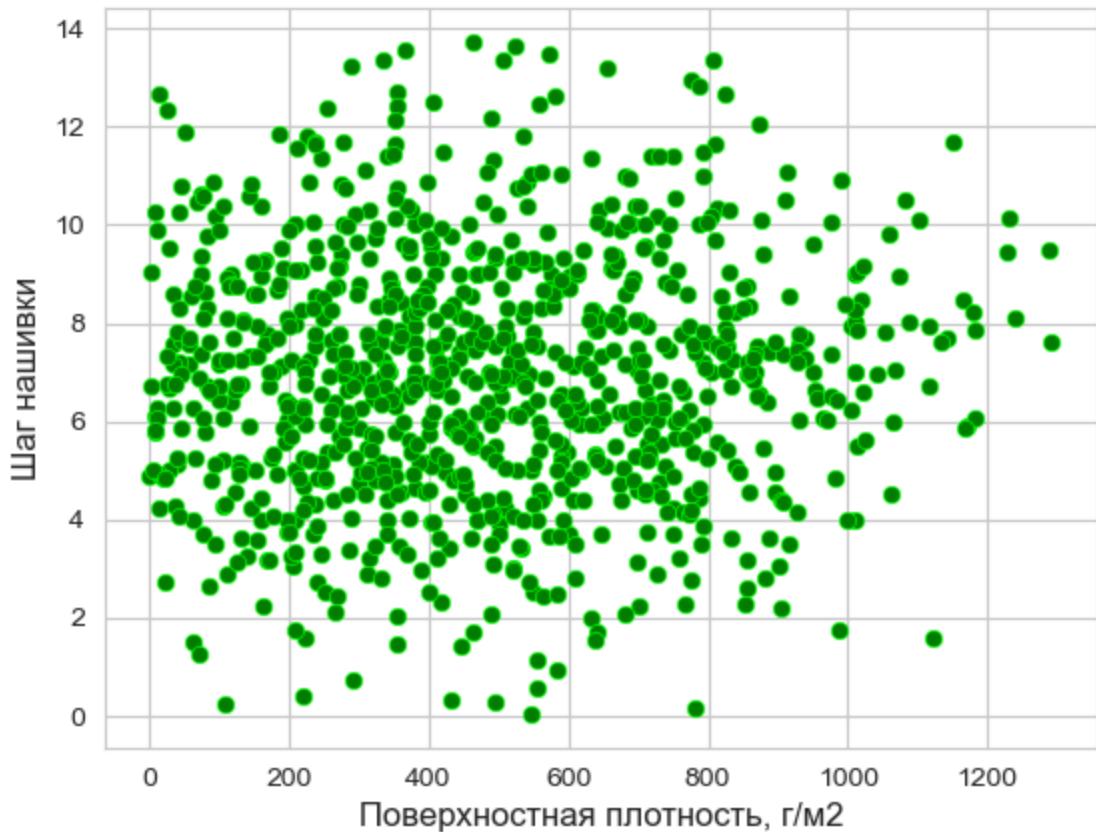
Зависимость



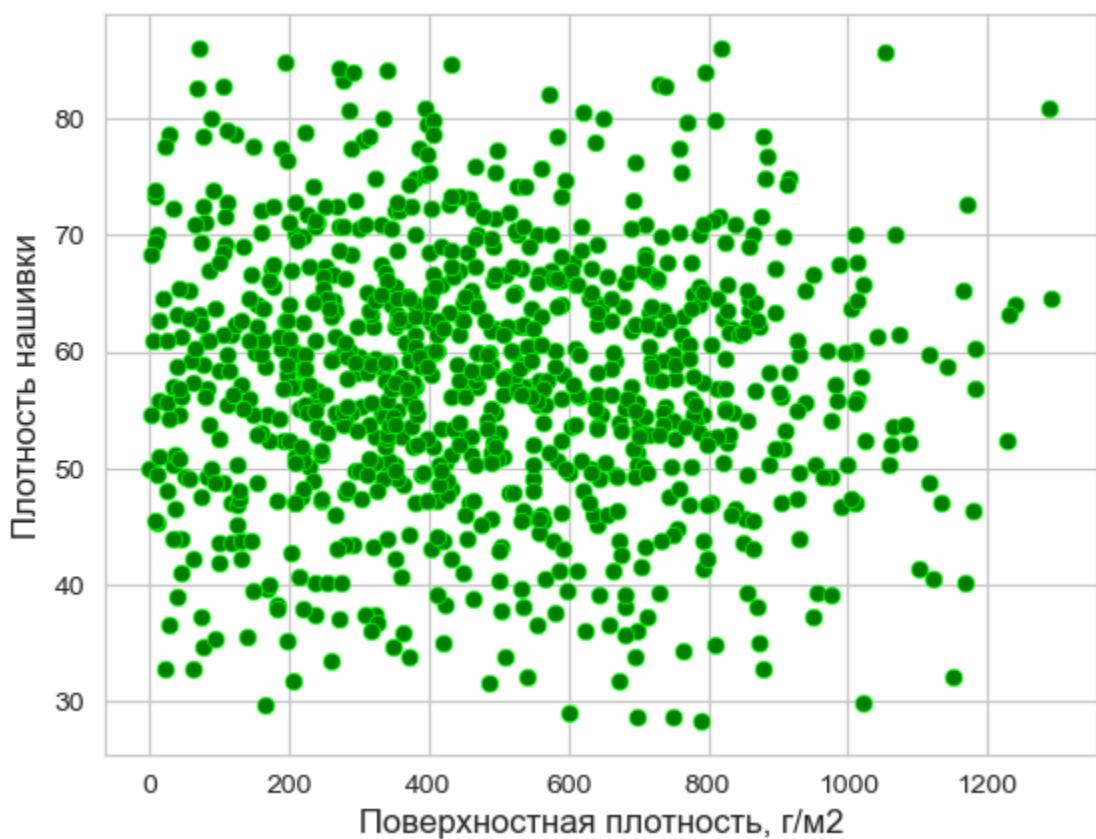
Зависимость



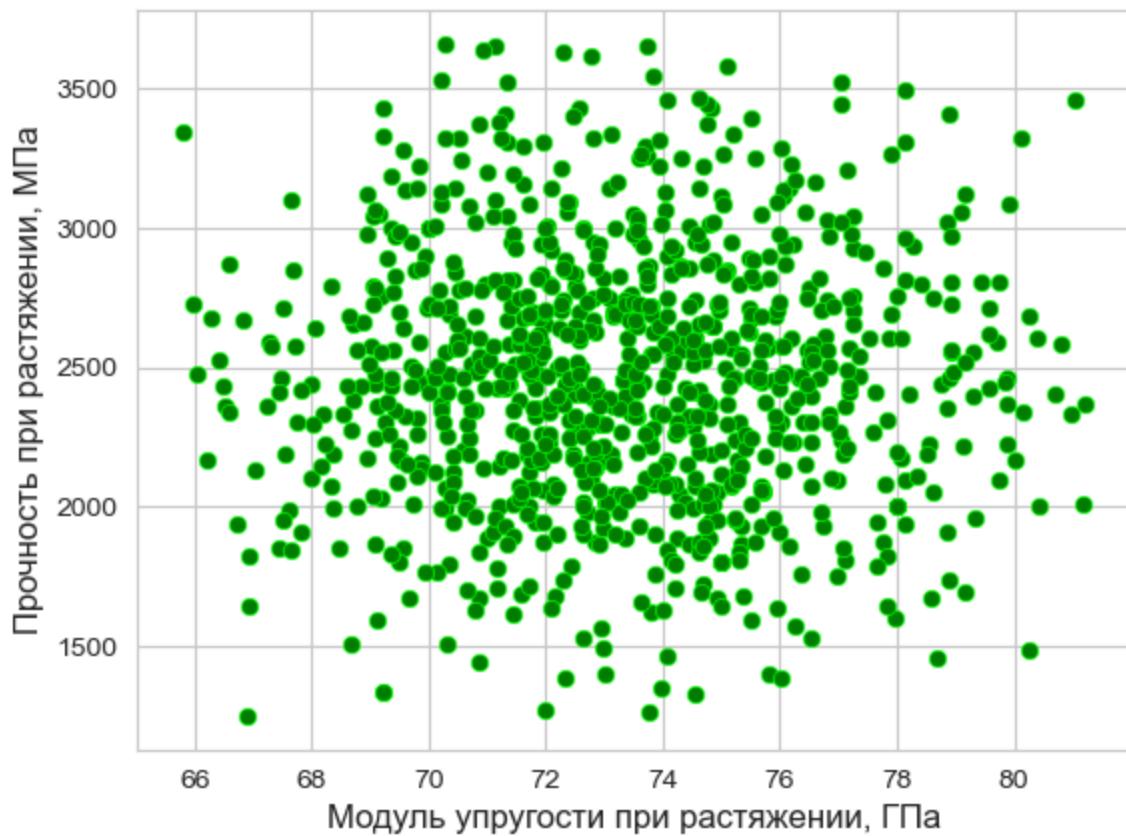
Зависимость



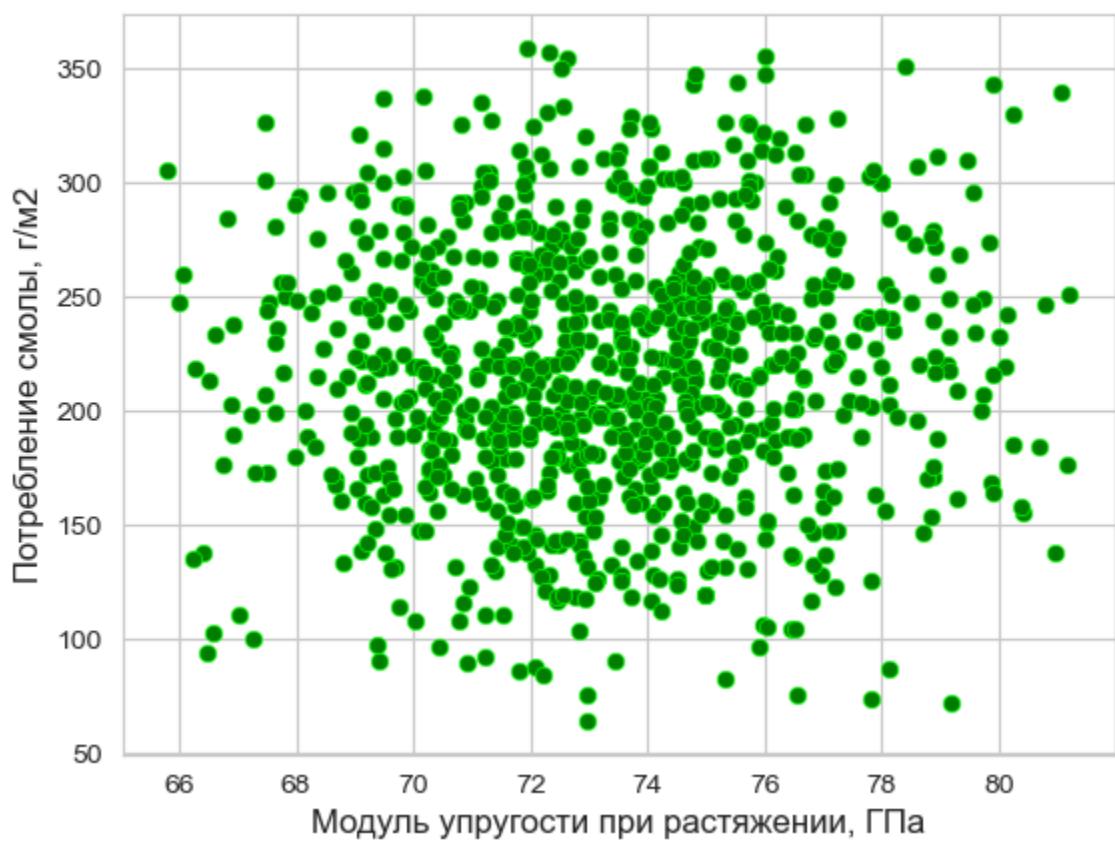
Зависимость

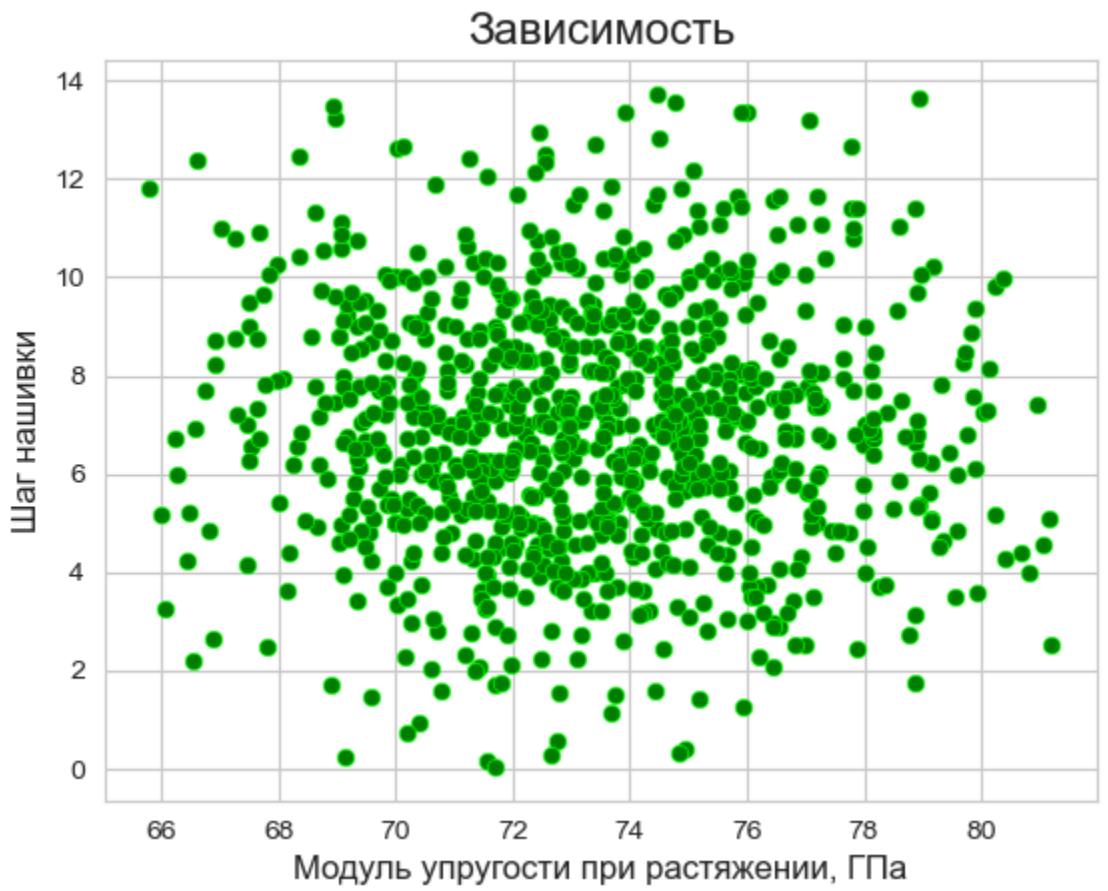


Зависимость

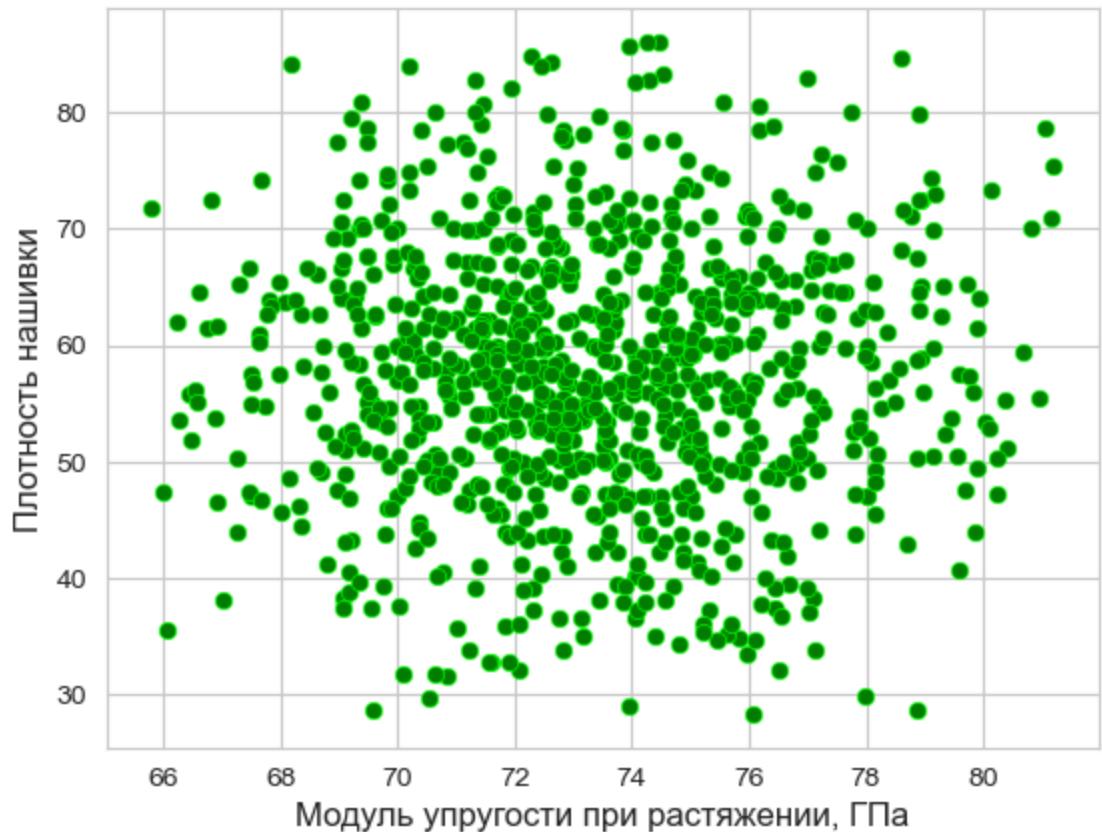


Зависимость

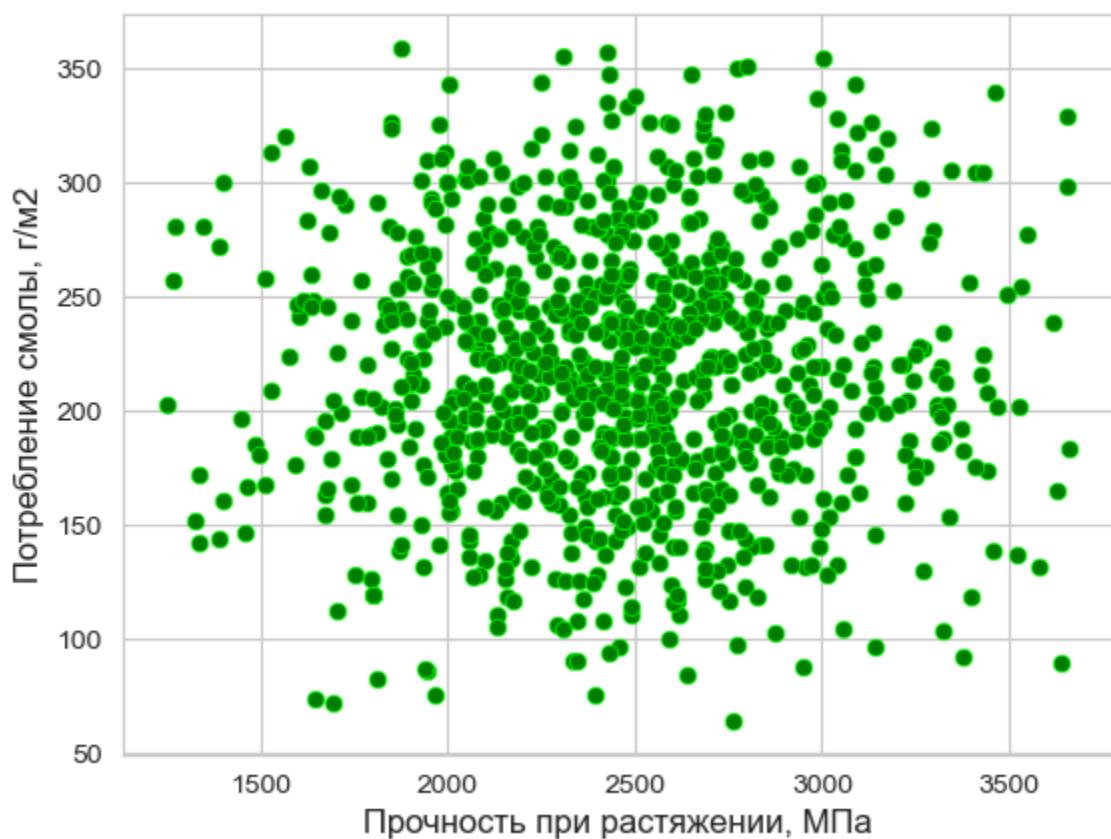




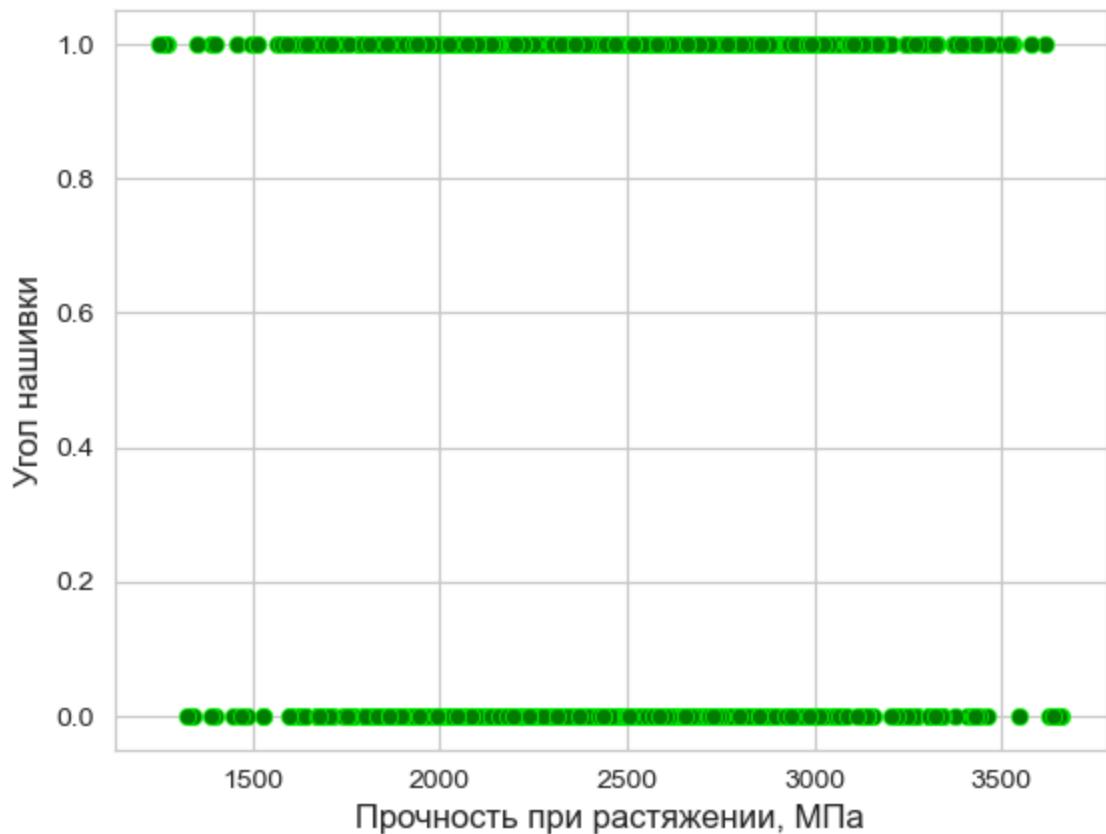
Зависимость



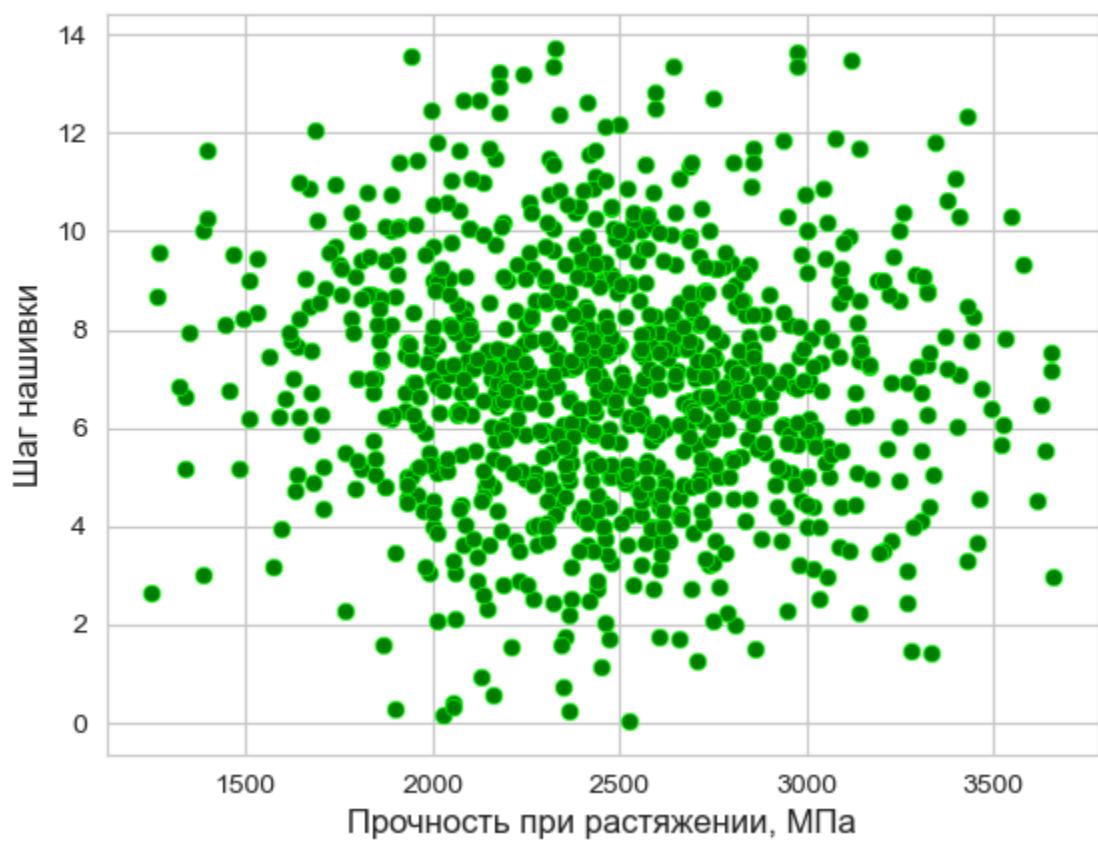
Зависимость



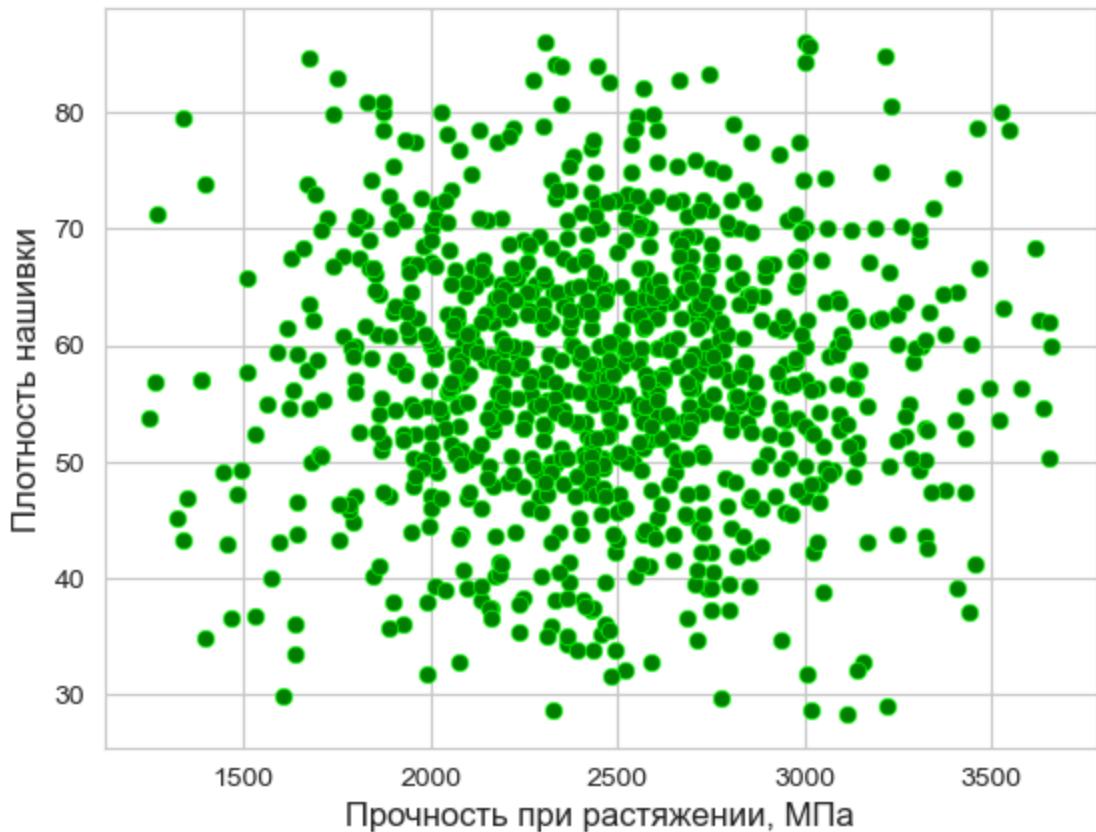
Зависимость



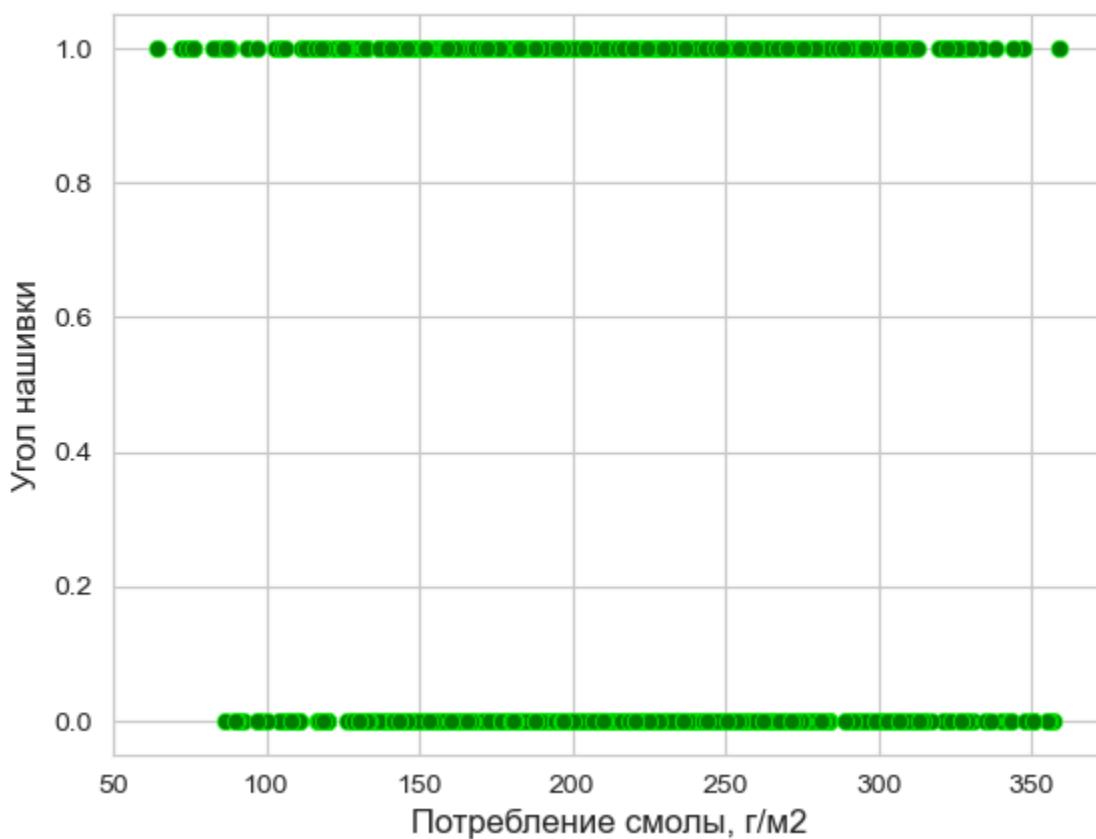
Зависимость



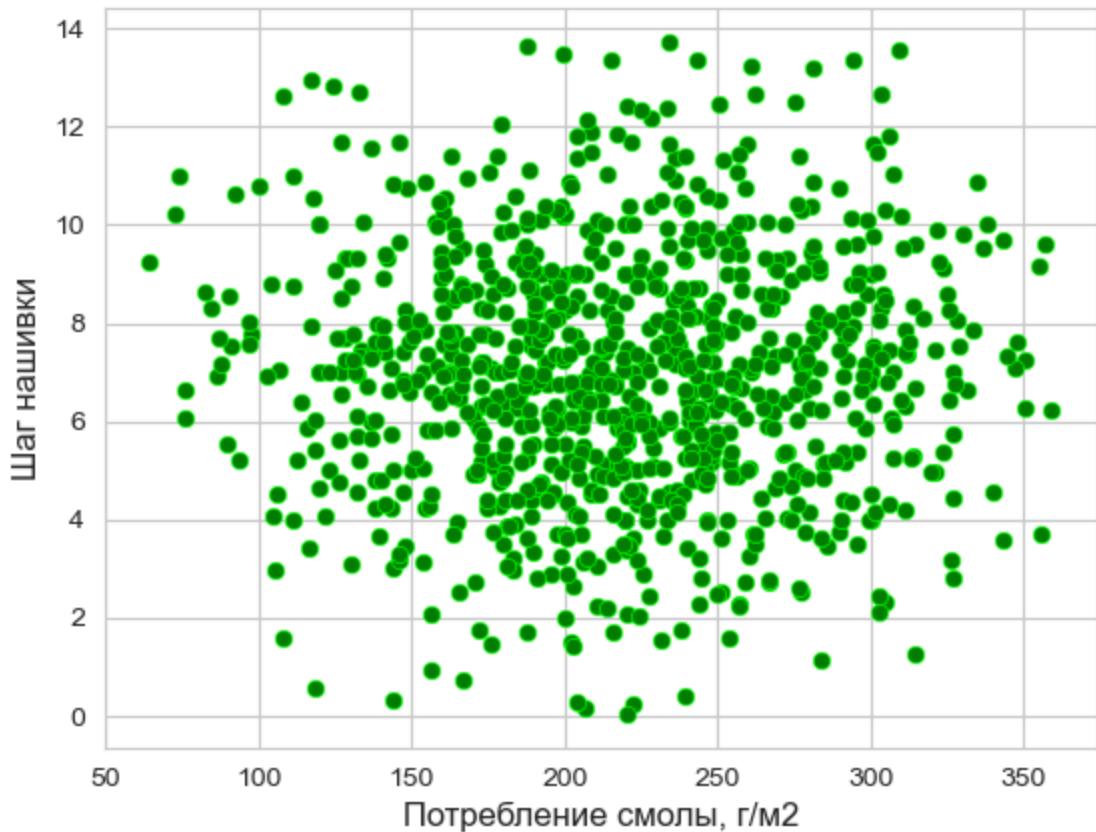
Зависимость



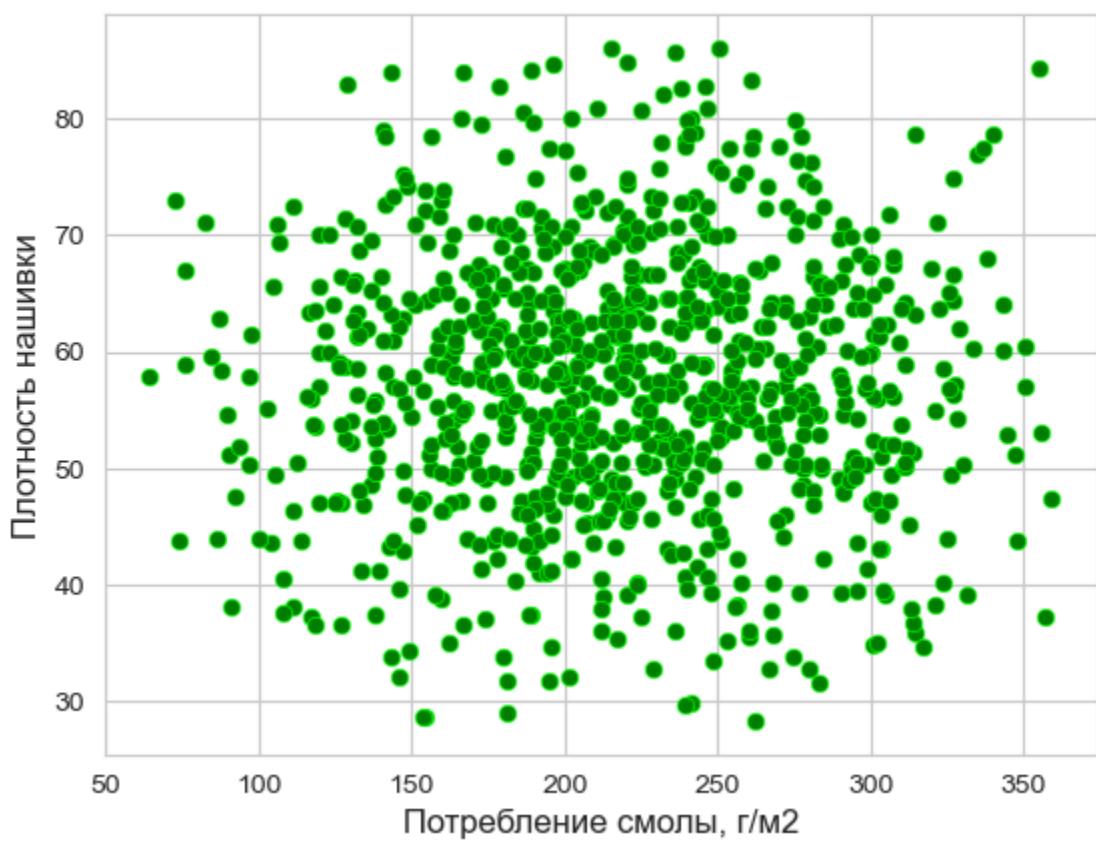
Зависимость



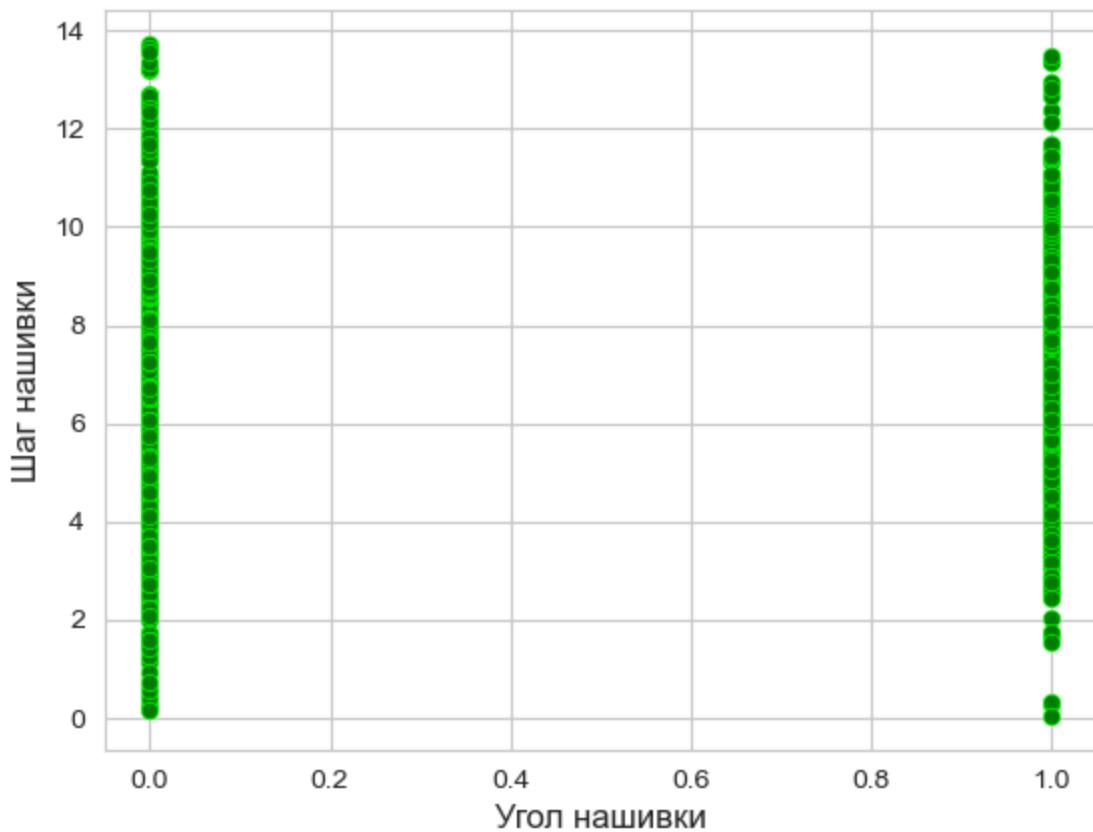
Зависимость



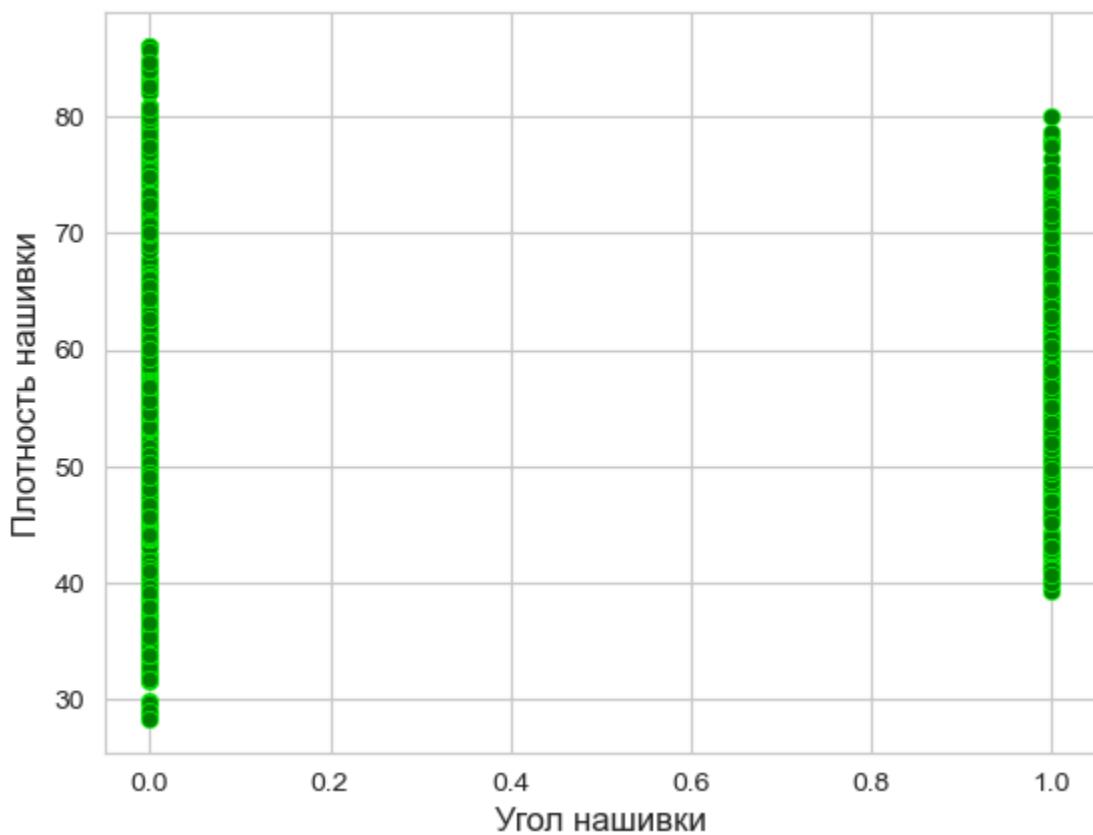
Зависимость

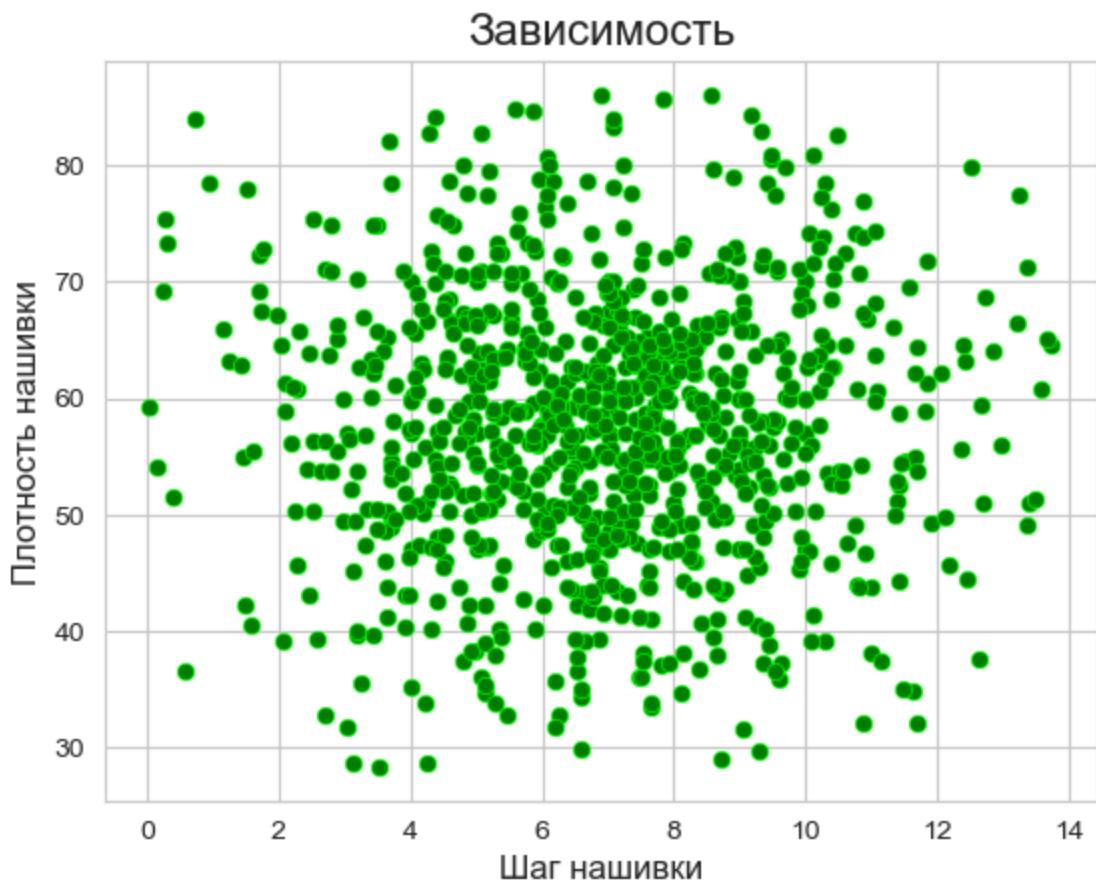


Зависимость



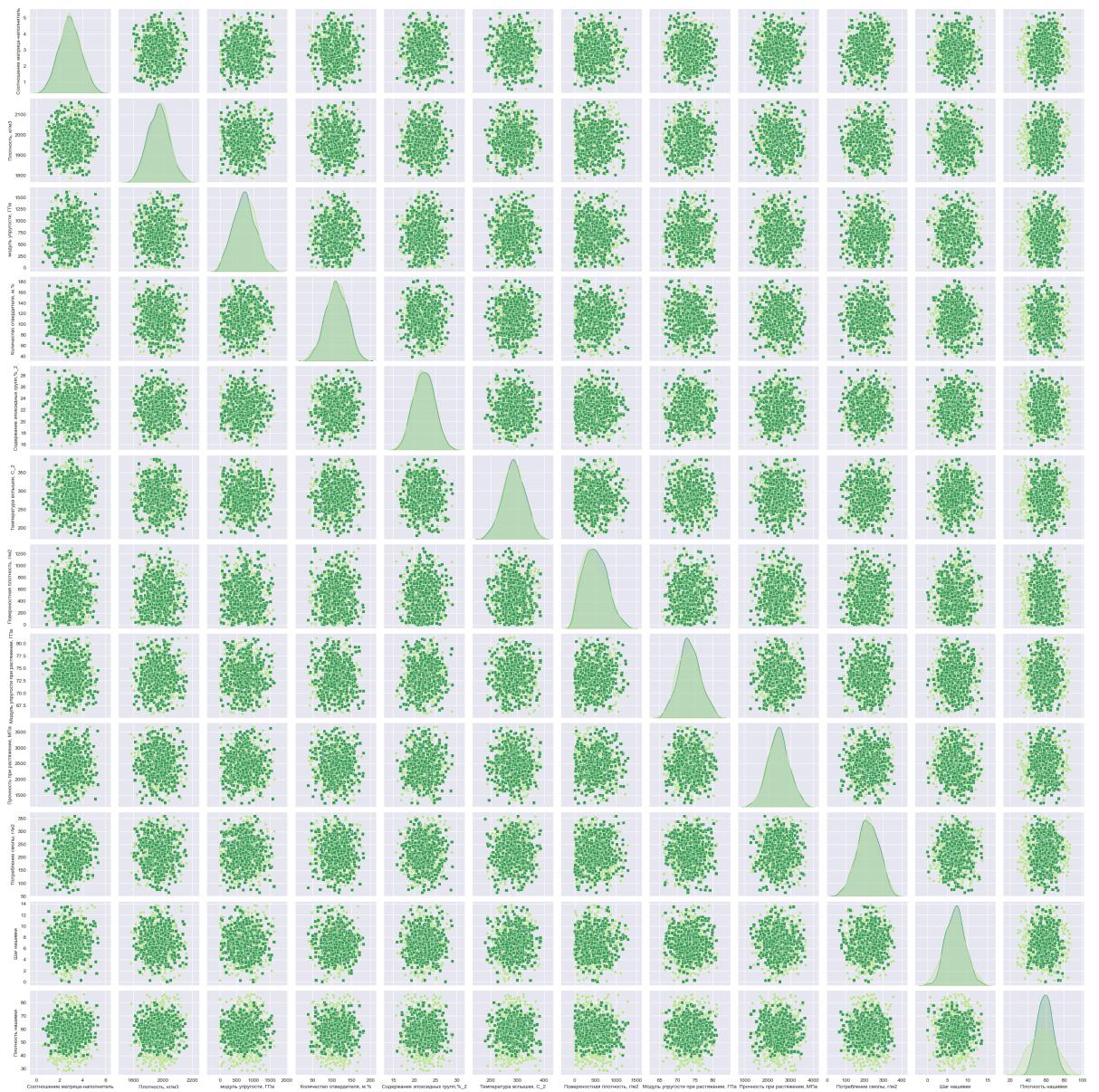
Зависимость





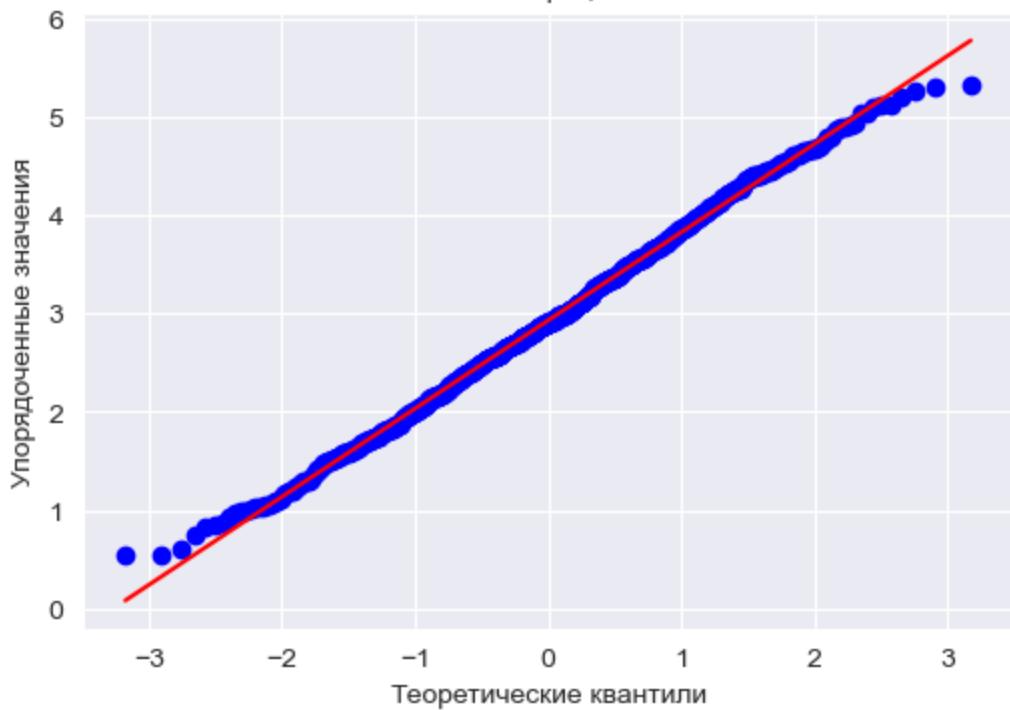
```
In [77]: sns.set_style('darkgrid')
sns.pairplot(df, hue = 'Угол нашивки', markers = ["o", "s"], diag_kind = 'auto')
```

```
Out[77]: <seaborn.axisgrid.PairGrid at 0x25888da6210>
```

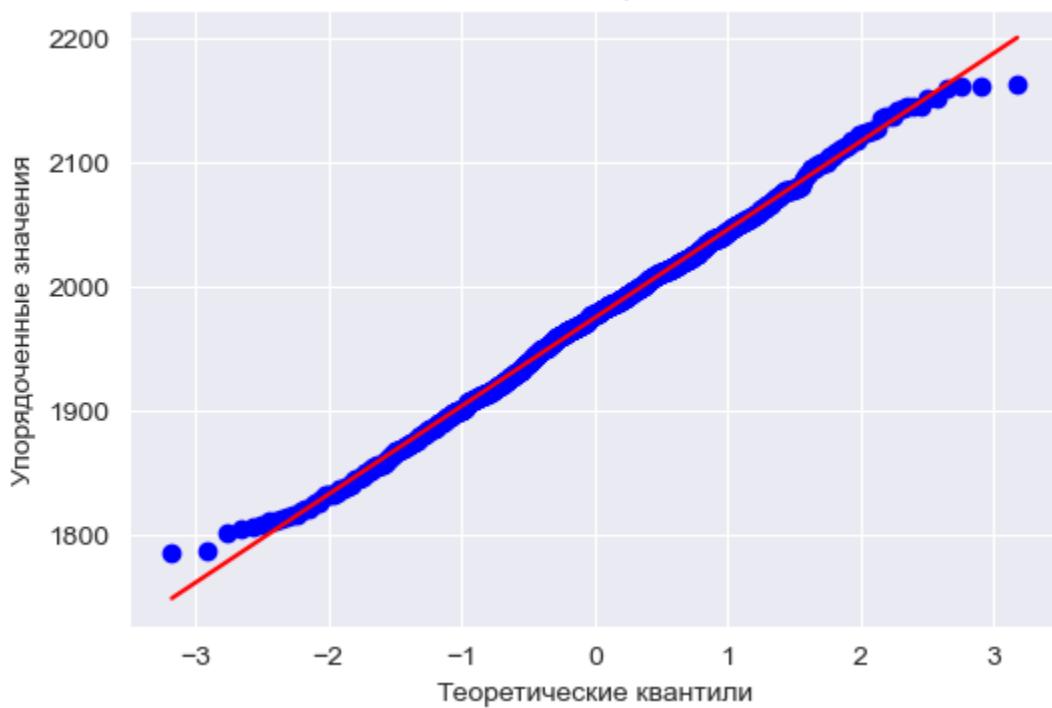


```
In [78]: for i in df.columns:  
    plt.figure(figsize = (6, 4))  
    res = stats.probplot(df[i], plot = plt)  
    plt.title(i, fontsize = 10)  
    plt.xlabel("Теоретические квантили", fontsize = 10)  
    plt.ylabel("Упорядоченные значения", fontsize = 10)  
    plt.show()
```

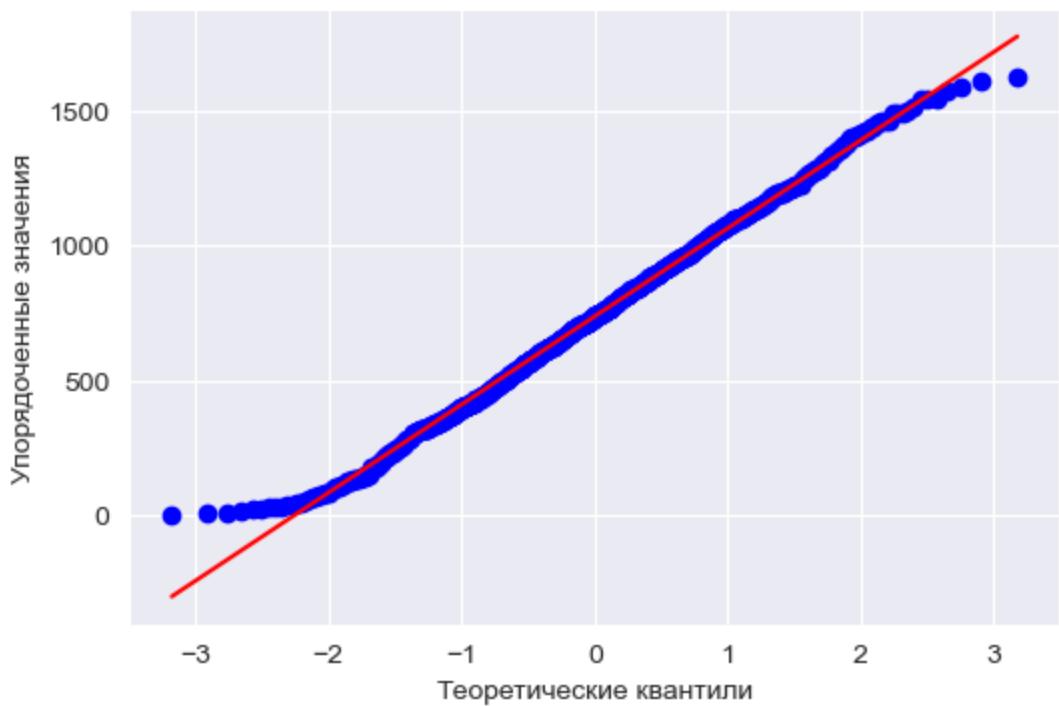
Соотношение матрица-наполнитель



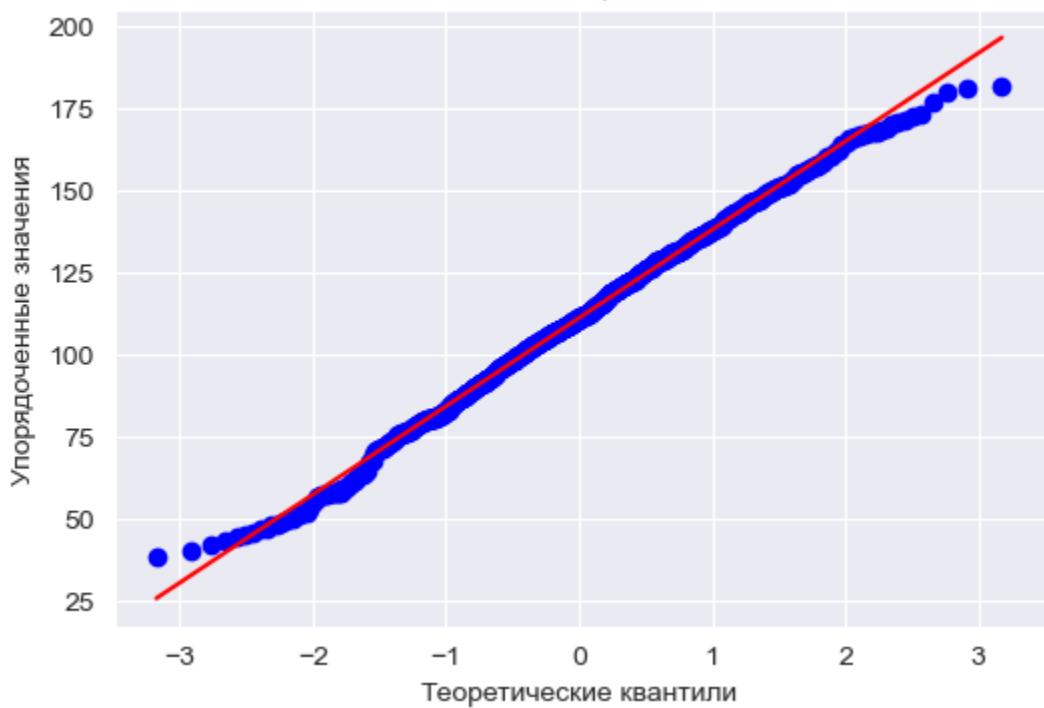
Плотность, кг/м³



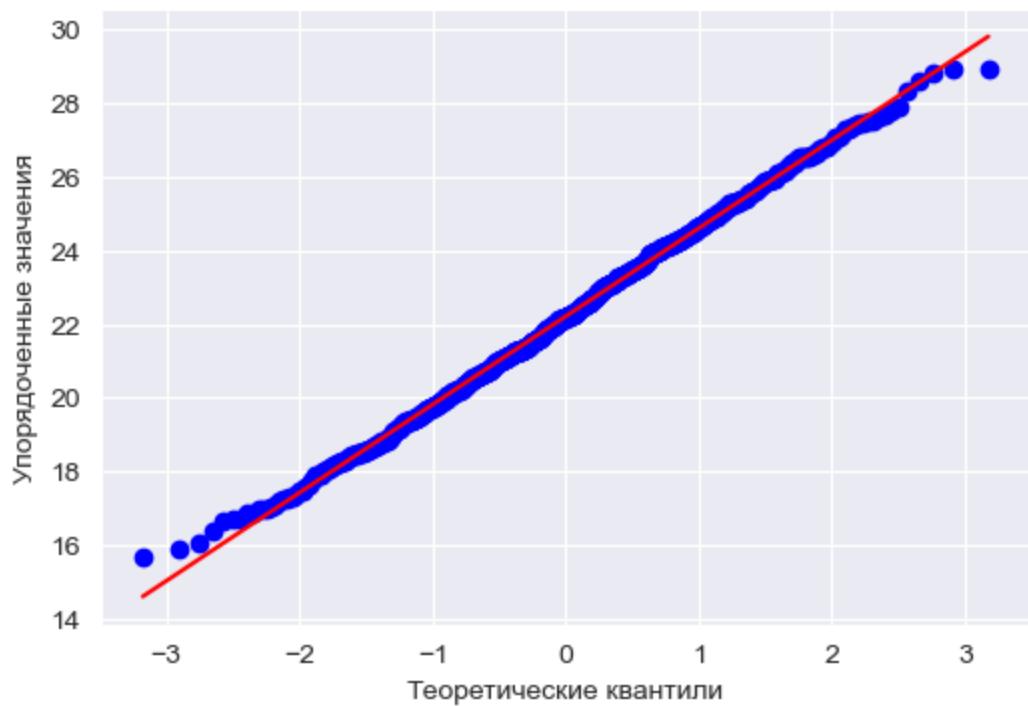
модуль упругости, ГПа



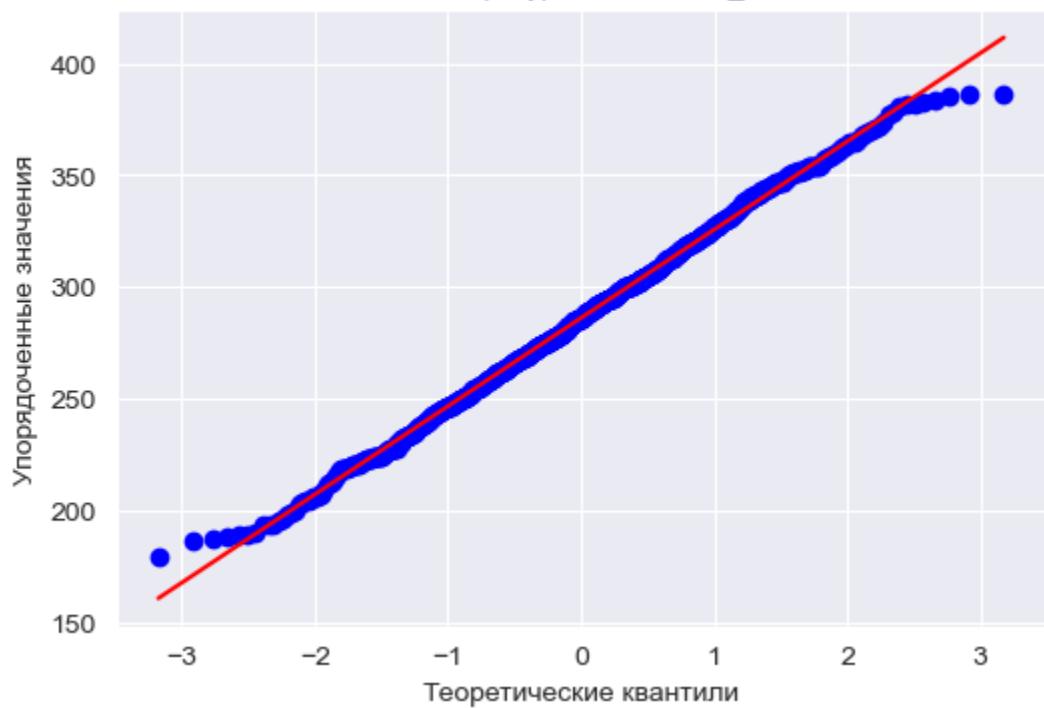
Количество отвердителя, м. %



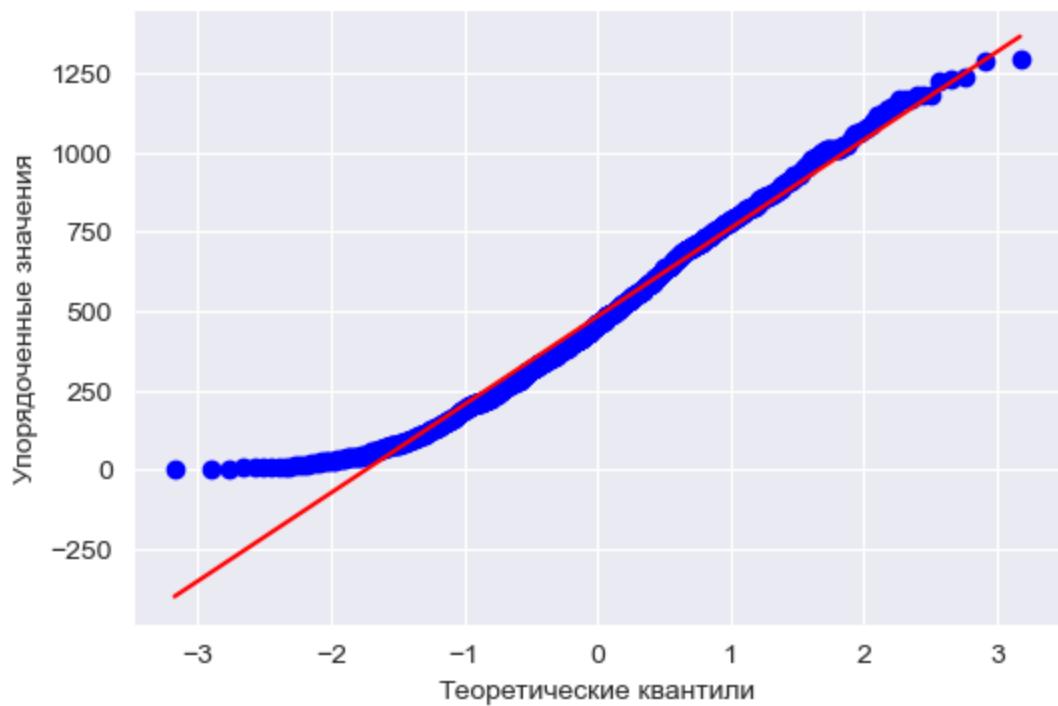
Содержание эпоксидных групп, %_2



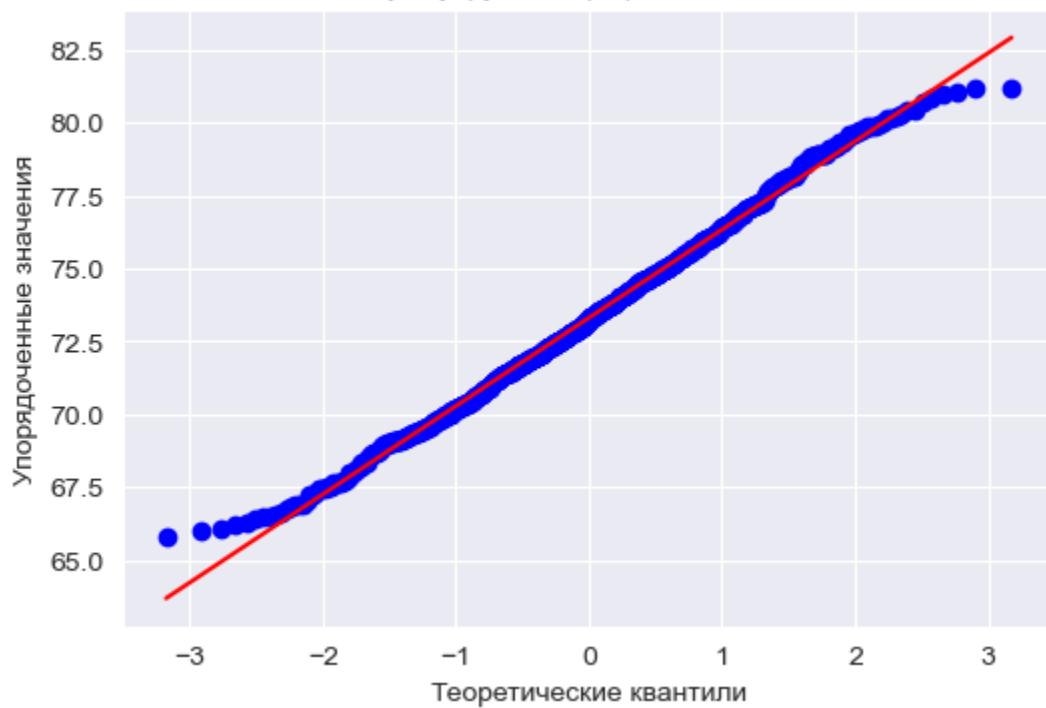
Температура вспышки, С_2



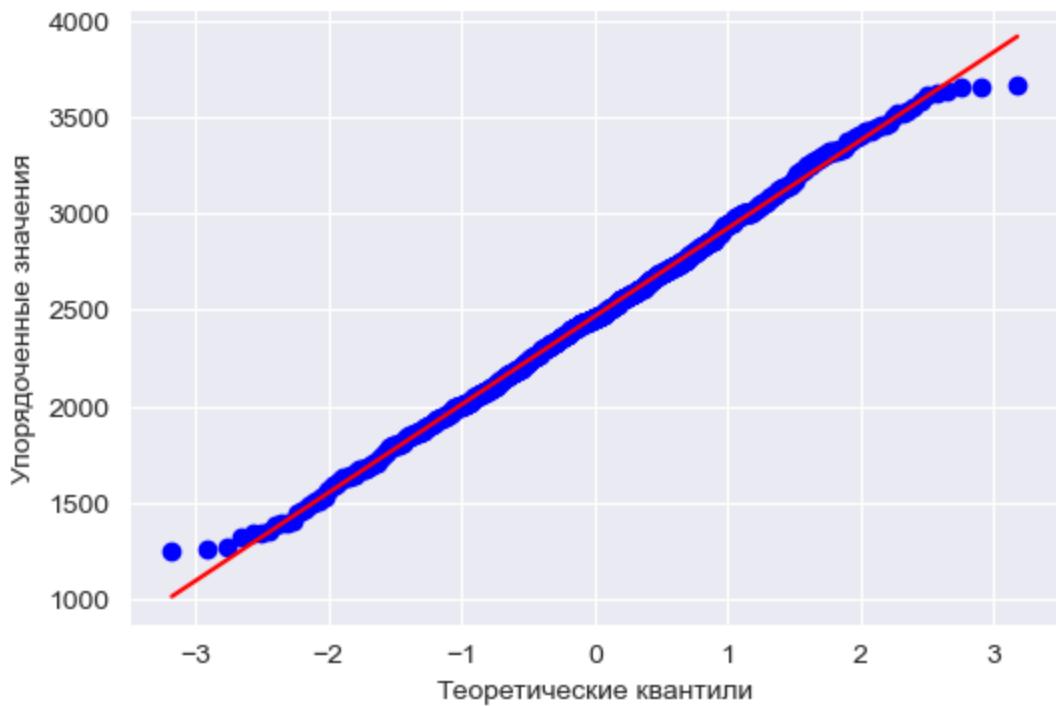
Поверхностная плотность, г/м²



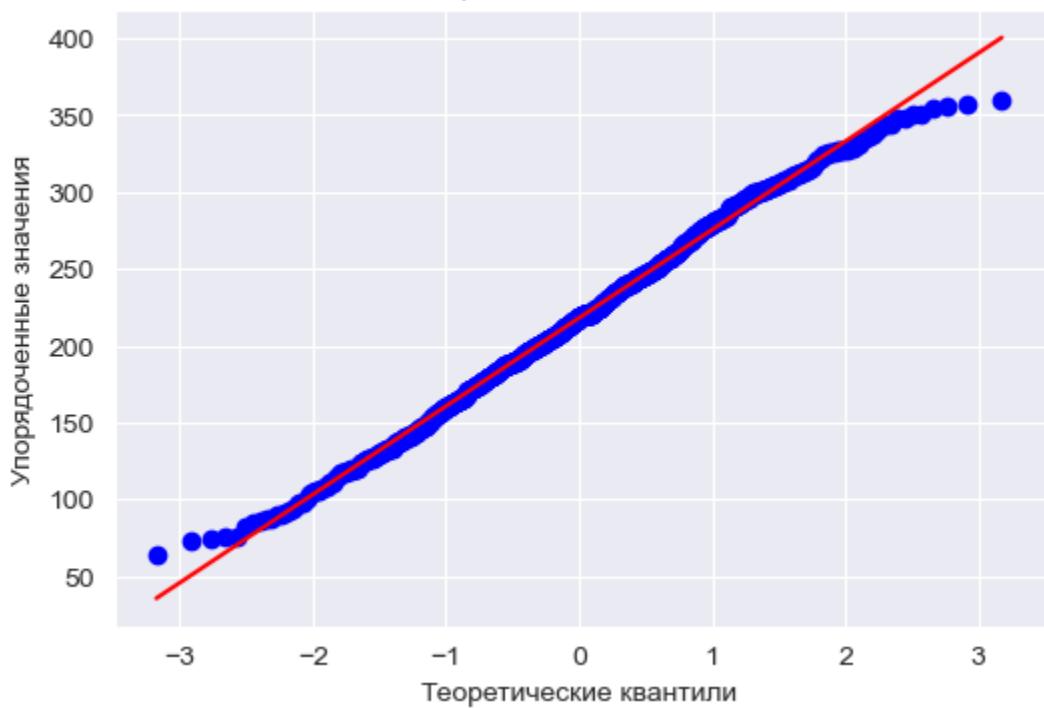
Модуль упругости при растяжении, ГПа

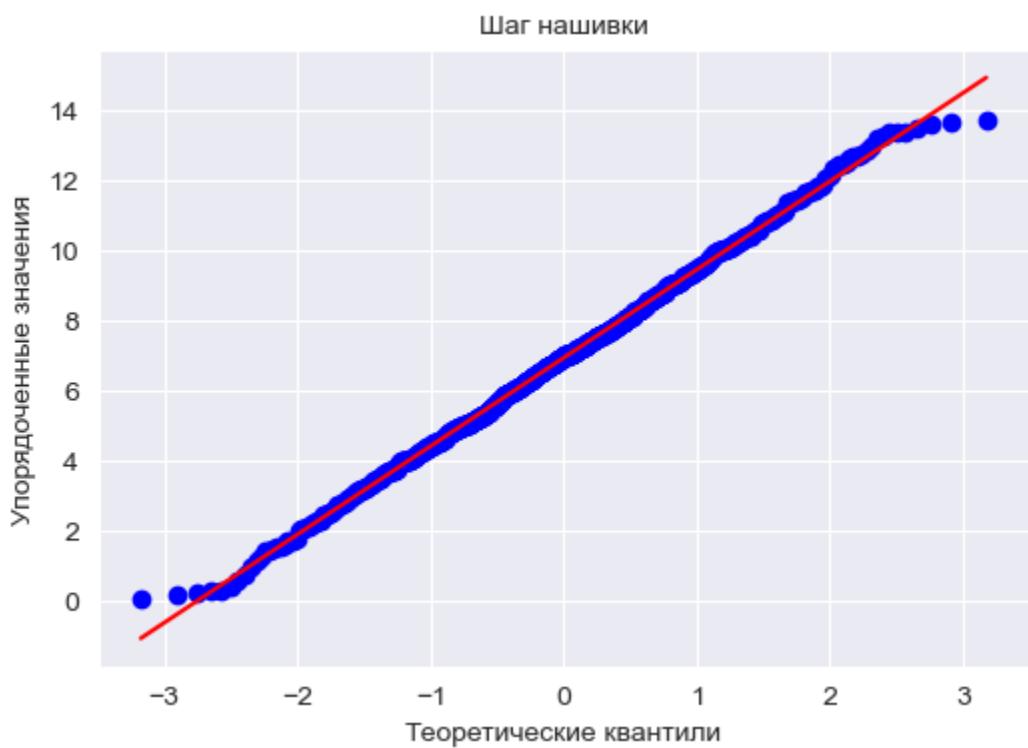
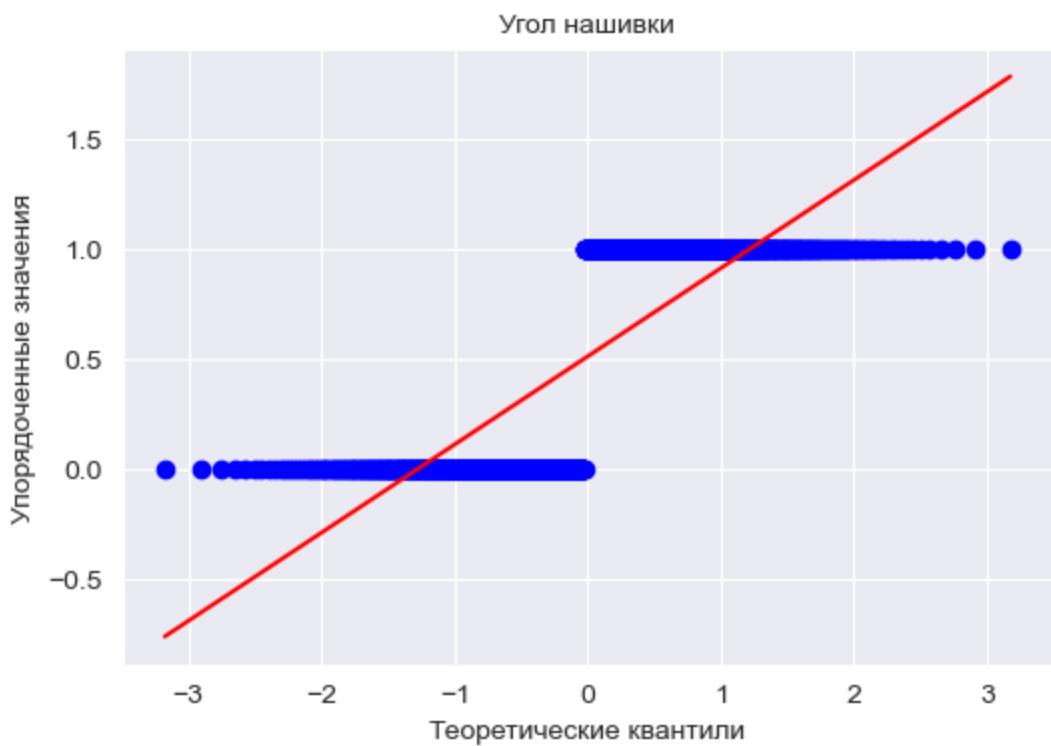


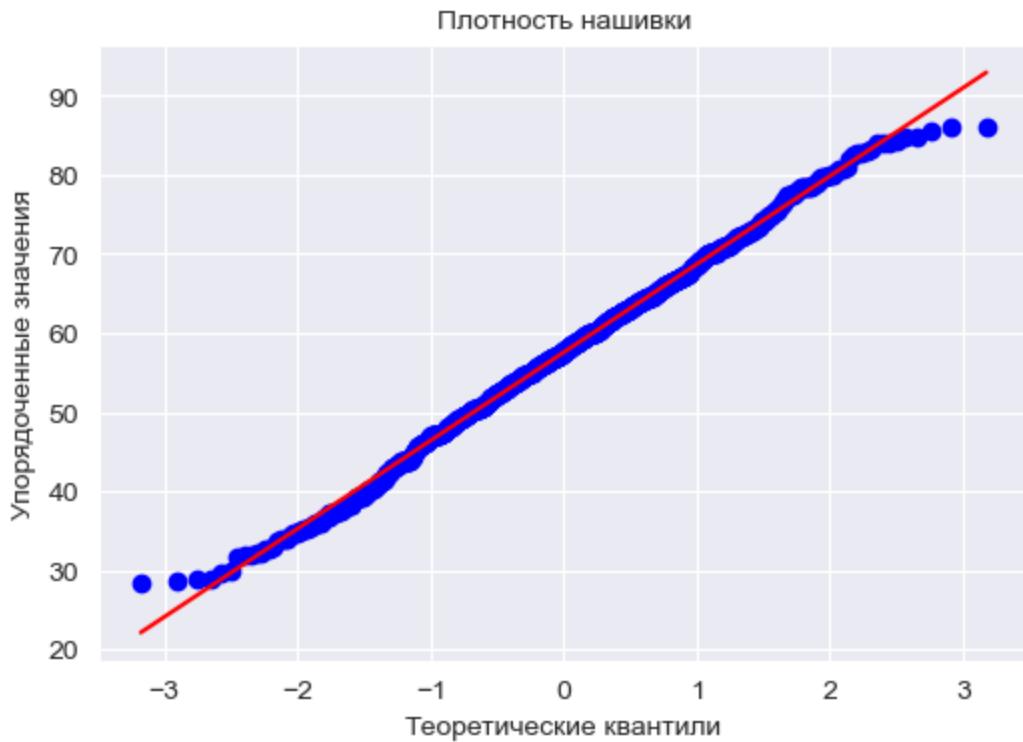
Прочность при растяжении, МПа



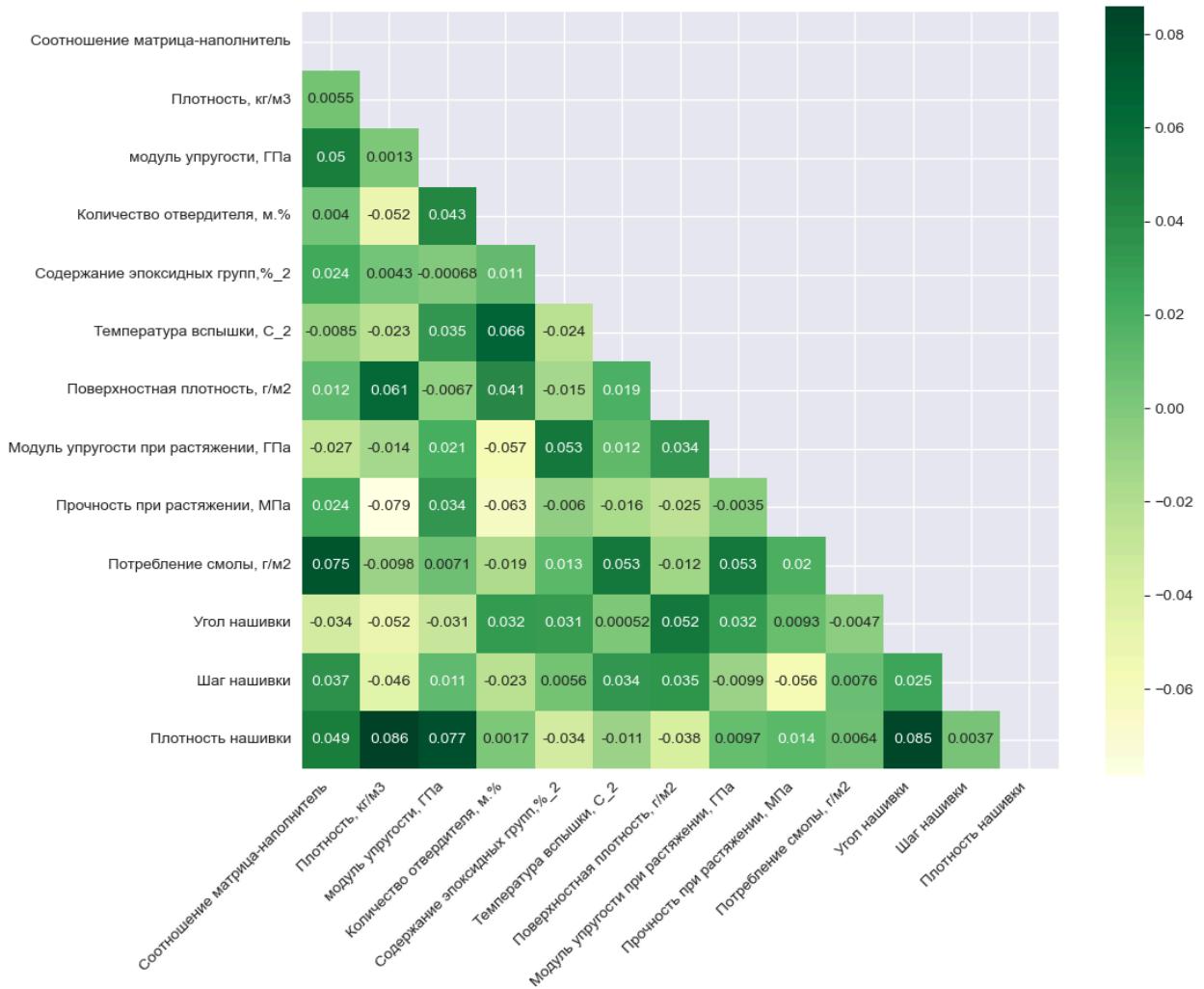
Потребление смолы, г/м²







```
In [79]: #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))
# # Визуализируем данные корреляции и создаем цветовую политику
sns.heatmap(df.corr(), mask = mask, annot = True, square = True, cmap = 'YlGn')
plt.xticks(rotation = 45, ha = 'right')
plt.show()
```



In [80]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 926 entries, 1 to 1022
Data 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   Потребление смолы, г/м2          926 non-null   float64 
 10  Угол нашивки                   926 non-null   int64   
 11  Шаг нашивки                   926 non-null   float64 
 12  Плотность нашивки             926 non-null   float64 

dtypes: float64(12), int64(1)
memory usage: 101.3 KB
```

```
In [81]: #Сохраняем отредактированный датасет в excel
df.to_csv('Itog\Itog.csv', encoding = 'cp1251' )
df.to_excel("Itog\Itog.xlsx")
```

```
In [82]: #Проводим повторный разведочный анализ
#Выведем корреляции между параметрами
df.corr()
```

Out[82]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп
Соотношение матрица-наполнитель	1.000000	0.005504	0.049910	0.003969	0.02
Плотность, кг/м3	0.005504	1.000000	0.001303	-0.051965	0.00
модуль упругости, ГПа	0.049910	0.001303	1.000000	0.043327	-0.00
Количество отвердителя, м.%	0.003969	-0.051965	0.043327	1.000000	0.01
Содержание эпоксидных групп,%_2	0.023834	0.004350	-0.000682	0.010752	1.00
Температура вспышки, С_2	-0.008536	-0.022796	0.034568	0.065849	-0.02
Поверхностная плотность, г/м2	0.011675	0.061442	-0.006727	0.041044	-0.01
Модуль упругости при растяжении, ГПа	-0.026694	-0.014301	0.021232	-0.056787	0.05
Прочность при растяжении, МПа	0.023550	-0.078511	0.033958	-0.062819	-0.00
Потребление смолы, г/м2	0.074926	-0.009771	0.007146	-0.019212	0.01
Угол нашивки	-0.033784	-0.051552	-0.031199	0.032415	0.03
Шаг нашивки	0.036905	-0.045528	0.011484	-0.023263	0.00
Плотность нашивки	0.049060	0.085970	0.076559	0.001731	-0.03

```
In [83]: # Посмотрим на средние и медианные значения датасета после выбросов
```

```
mean_and_50 = df.describe()  
mean_and_50.loc[['mean', '50%']]
```

Out[83]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2
mean	2.929274	1974.170536	737.009604	111.187428	22.206999	140.0
50%	2.906630	1977.450684	738.164864	111.183627	22.178471	140.0

```
In [86]: df_norm = df.copy()  
df_norm.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 926 entries, 1 to 1022  
Data 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   Потребление смолы, г/м2            926 non-null   float64  
 10  Угол нашивки                      926 non-null   int64  
 11  Шаг нашивки                        926 non-null   float64  
 12  Плотность нашивки                  926 non-null   float64  
dtypes: float64(12), int64(1)  
memory usage: 101.3 KB
```

```
In [84]: ### Предобработка данных.  
#Нормализуем данные
```

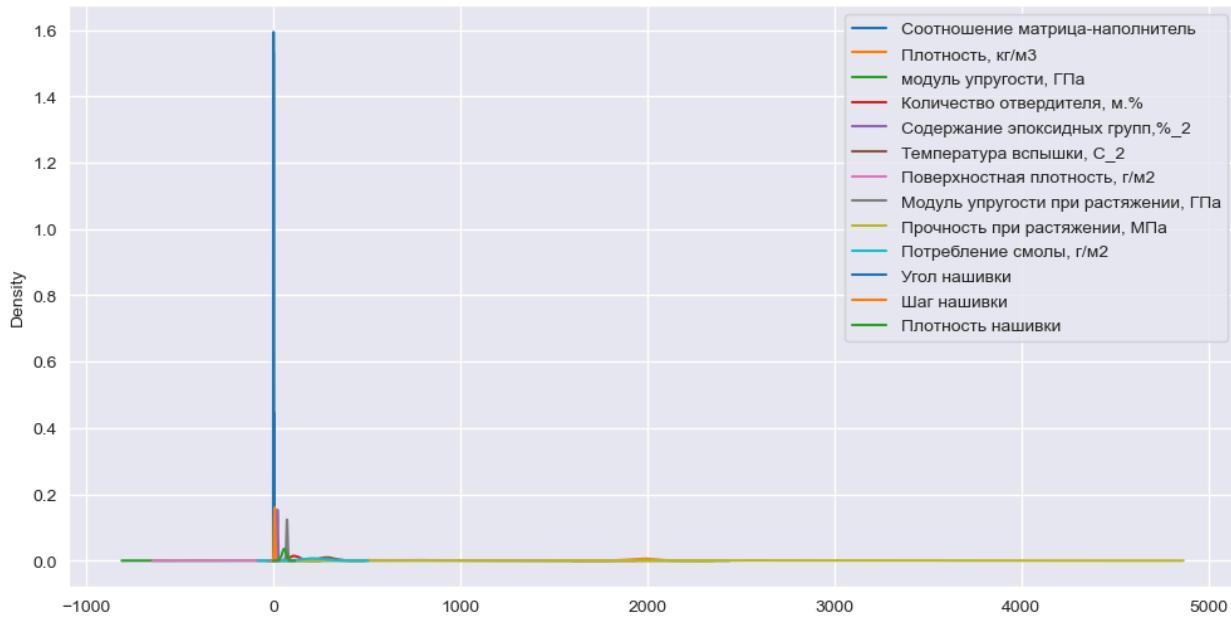
#У нас в основном количественные признаки, поэтому можно применить нормализацию

#Этап предобработки данных нужен нам и для введенных данных в будущем приложении

In [87]:

```
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
```

Out[87]: <Axes: ylabel='Density'>



In [88]: #Нормализуем данные с помощью MinMaxScaler()

```
scaler = preprocessing.MinMaxScaler()
col = df.columns
result = scaler.fit_transform(df)

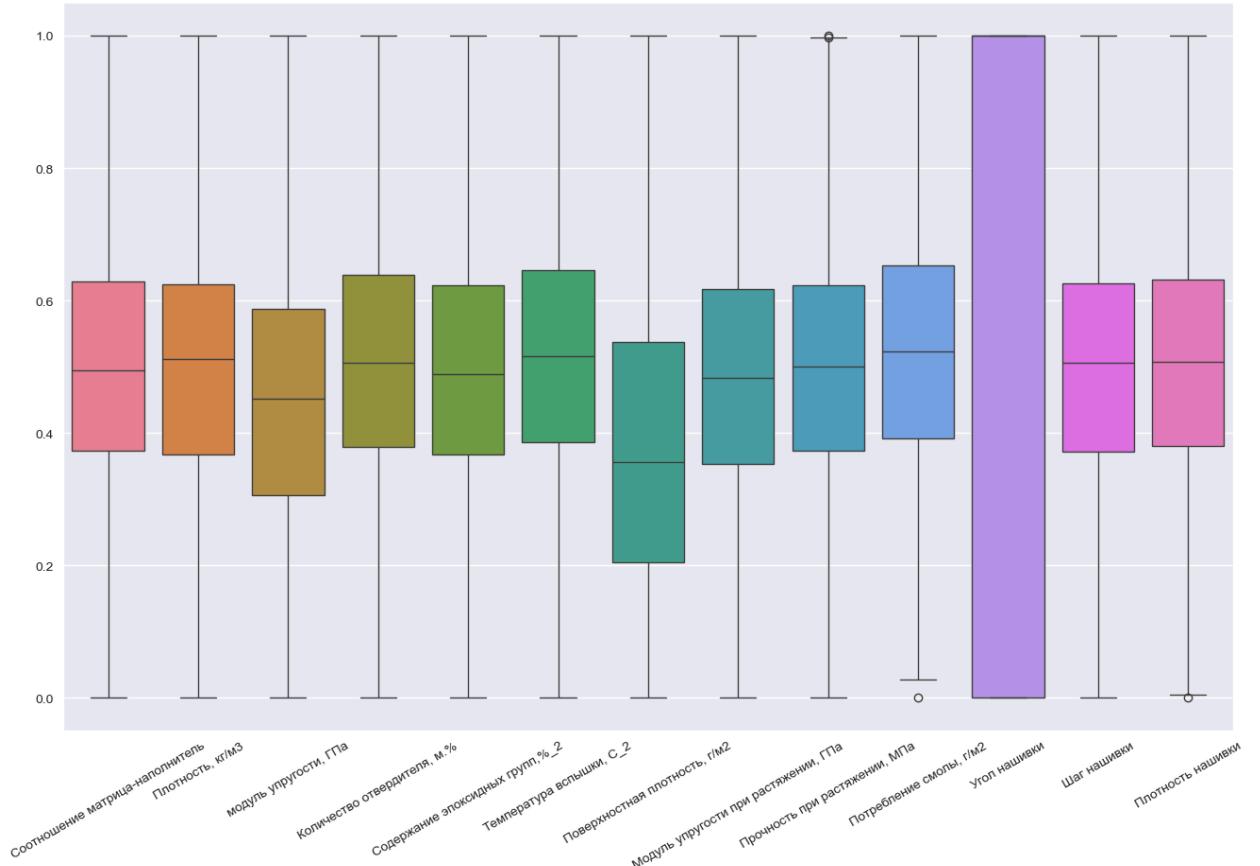
df_minmax_n = pd.DataFrame(result, columns = col)
df_minmax_n.describe()
```

Out[88]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2
count	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000
mean	0.499687	0.503041	0.451888	0.506559	0.491063	0.491063
std	0.187841	0.188261	0.201469	0.187611	0.180438	0.180438
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.373077	0.368184	0.305669	0.378514	0.367101	0.367101
50%	0.494936	0.511740	0.452599	0.506532	0.488912	0.488912
75%	0.629774	0.625266	0.587461	0.639120	0.623296	0.623296
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

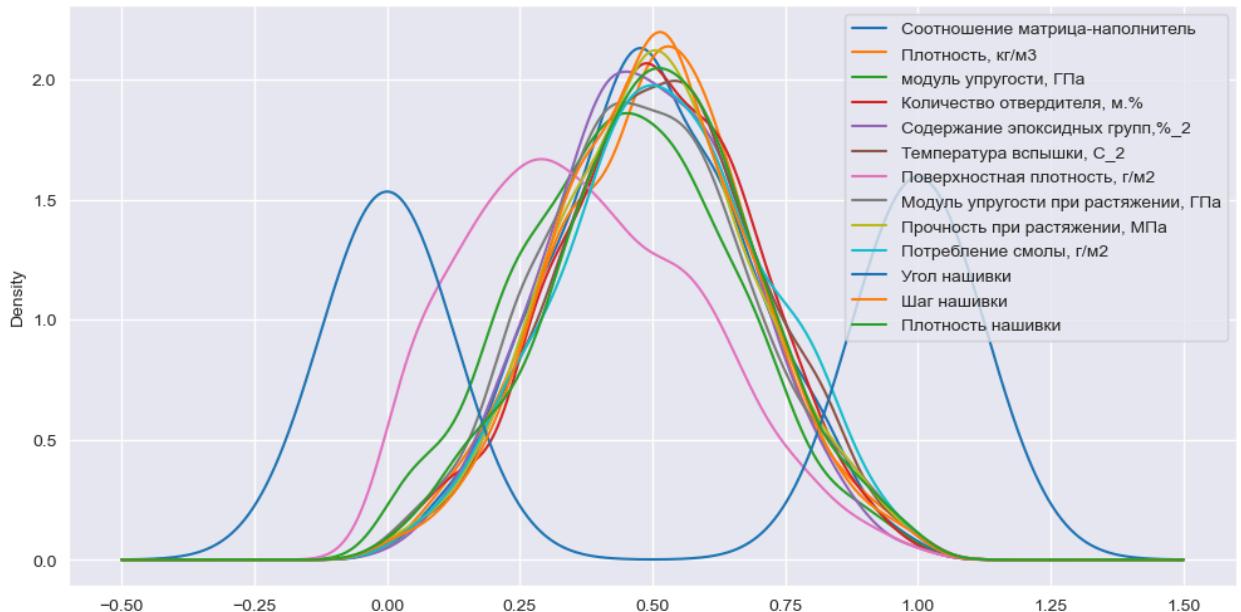
In [89]: plt.figure(figsize = (16,10))

```
ax = sns.boxplot(data = df_minmax_n)
ax.set_xticklabels(ax.get_xticklabels(),rotation=30);
```



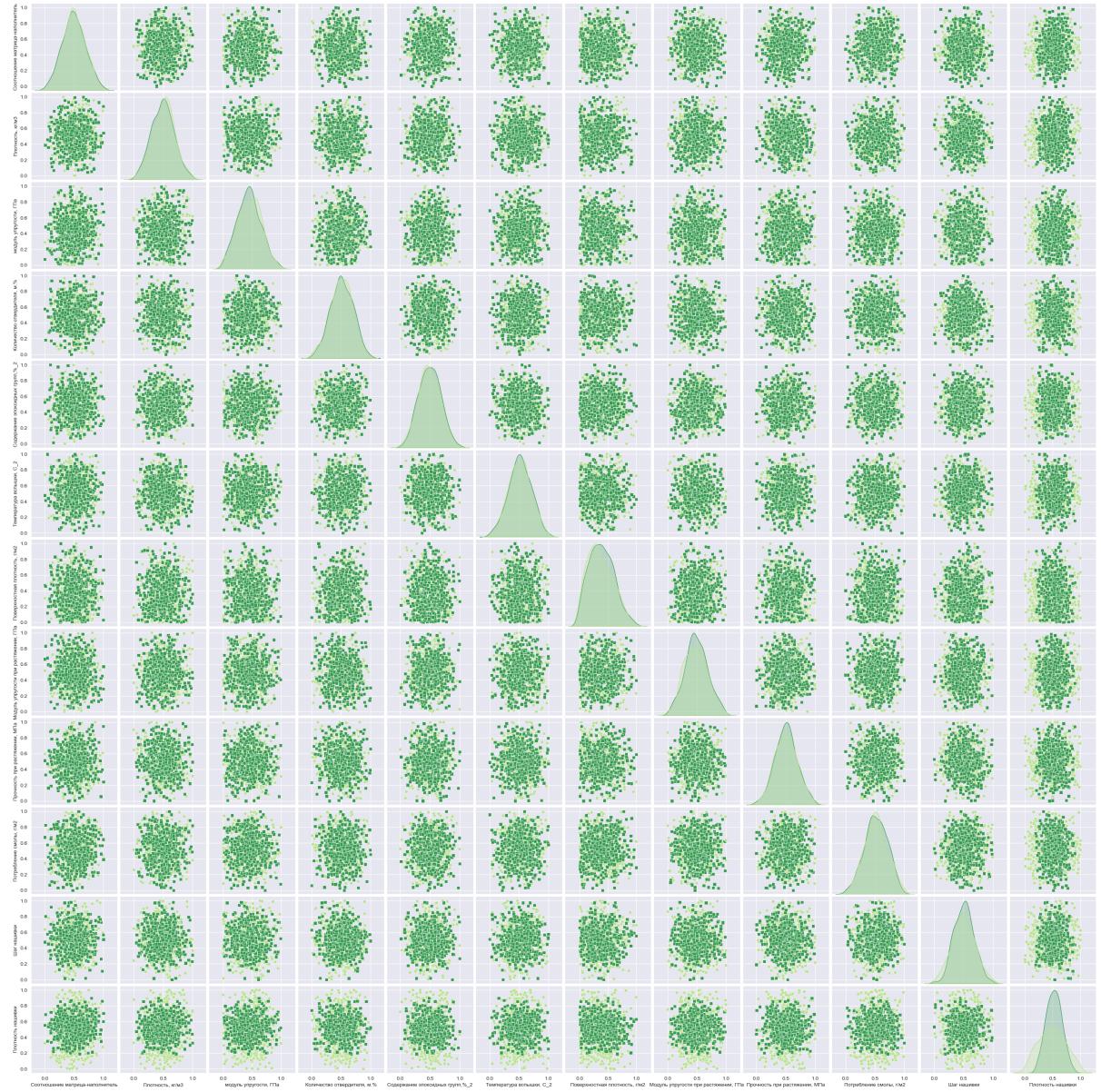
```
In [90]: fig, ax = plt.subplots(figsize = (12, 6))
df_minmax_n.plot(kind = 'kde', ax = ax)
```

Out[90]: <Axes: ylabel='Density'>

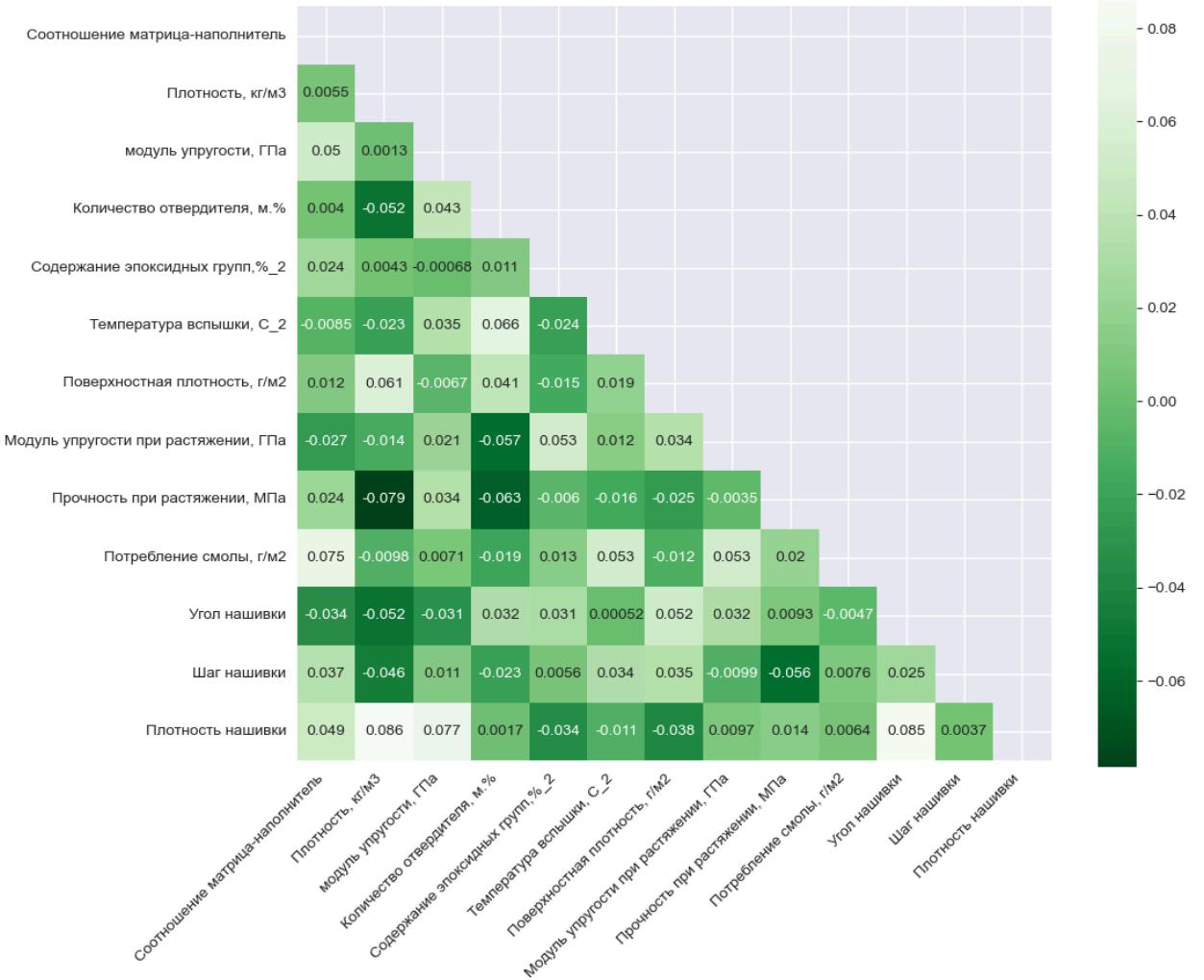


```
In [92]: sns.pairplot(df_minmax_n, hue = 'Угол нашивки', markers = ["o", "s"], diag_kir
```

Out[92]: <seaborn.axisgrid.PairGrid at 0x258e21d7bb0>



```
In [93]: mask = np.triu(df_minmax_n.corr())
f, ax = plt.subplots(figsize = (11, 9))
sns.heatmap(df_minmax_n.corr(), mask = mask, annot = True, square = True, cmap = 'RdYlGn')
plt.xticks(rotation = 45, ha = 'right')
plt.show()
```



In [94]: #Нормализуем данные с помощью Normalizer()

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

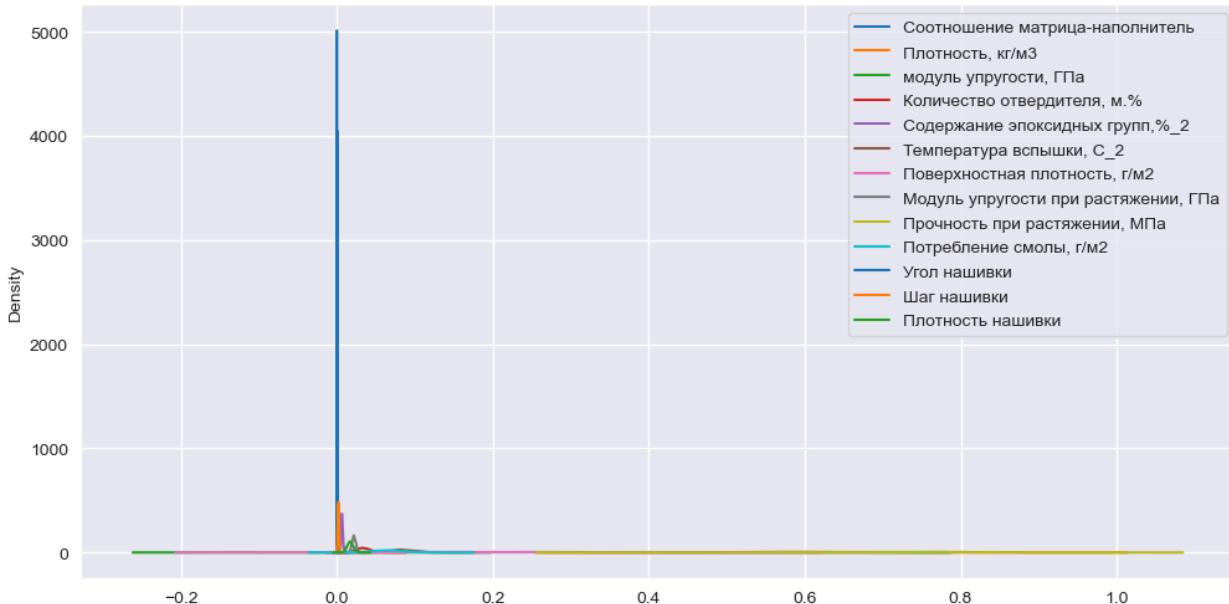
Out[94]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Темп вс
0	0.000499	0.545436	0.198490	0.013434	0.006381	(
1	0.000499	0.545011	0.198335	0.034634	0.005705	(
2	0.000744	0.544829	0.202097	0.030022	0.005976	(
3	0.000746	0.539271	0.201687	0.030161	0.006004	(
4	0.000699	0.519919	0.219673	0.030449	0.006062	(
...
921	0.000700	0.601751	0.281397	0.026816	0.006203	(
922	0.001078	0.641795	0.139227	0.045701	0.006136	(
923	0.000953	0.573123	0.121122	0.032118	0.006961	(
924	0.001192	0.664667	0.238453	0.045473	0.006190	(
925	0.001071	0.531728	0.117381	0.036336	0.007728	(

926 rows × 13 columns

In [95]: `fig, ax = plt.subplots(figsize = (12, 6))
df_norm_n.plot(kind = 'kde', ax = ax)`

Out[95]: <Axes: ylabel='Density'>



In [96]: `#Сравним с данными до нормализации
df.head()`

Out[96]:

Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки
1	1.857143	2030.0	738.736842	50.00	23.750000
3	1.857143	2030.0	738.736842	129.00	21.250000
4	2.771331	2030.0	753.000000	111.86	22.267857
5	2.767918	2000.0	748.000000	111.86	22.267857
6	2.569620	1910.0	807.000000	111.86	22.267857

In [97]:

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

Out[97]:

Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки
0	1.857143	2030.0	738.736842	50.00	23.750000
1	1.857143	2030.0	738.736842	129.00	21.250000
2	2.771331	2030.0	753.000000	111.86	22.267857
3	2.767918	2000.0	748.000000	111.86	22.267857
4	2.569620	1910.0	807.000000	111.86	22.267857
5	2.561475	1900.0	535.000000	111.86	22.267857
6	3.557018	1930.0	889.000000	129.00	21.250000
7	3.532338	2100.0	1421.000000	129.00	21.250000
8	2.919678	2160.0	933.000000	129.00	21.250000
9	2.877358	1990.0	1628.000000	129.00	21.250000

In [98]:

```
#Рассмотрим несколько вариантов корреляции между параметрами после нормализации
df_norm_n[df_norm_n.columns].corr()
```

Out[98]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп
Соотношение матрица-наполнитель	1.000000	0.280778	0.030799	0.138298	0.22
Плотность, кг/м3	0.280778	1.000000	-0.041385	0.389383	0.65
модуль упругости, ГПа	0.030799	-0.041385	1.000000	0.048529	-0.02
Количество отвердителя, м.%	0.138298	0.389383	0.048529	1.000000	0.30
Содержание эпоксидных групп,%_2	0.222080	0.658997	-0.028501	0.305066	1.00
Температура вспышки, С_2	0.181389	0.566959	0.001122	0.309626	0.40
Поверхностная плотность, г/м2	0.025020	0.067377	-0.025607	0.068559	0.02
Модуль упругости при растяжении, ГПа	0.261448	0.876721	-0.026469	0.376364	0.66
Прочность при растяжении, МПа	-0.257279	-0.839121	-0.385935	-0.379318	-0.55
Потребление смолы, г/м2	0.156483	0.314371	-0.011220	0.140773	0.24
Угол нашивки	-0.010049	0.087455	-0.038104	0.065263	0.10
Шаг нашивки	0.136283	0.296371	0.003858	0.119476	0.23
Плотность нашивки	0.177313	0.448746	0.043269	0.200689	0.28

In [99]: #Рассмотрим второй вариант корреляции между параметрами после нормализации (в df_minmax_n[df_minmax_n.columns].corr()

Out[99]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп
Соотношение матрица-наполнитель	1.000000	0.005504	0.049910	0.003969	0.02
Плотность, кг/м3	0.005504	1.000000	0.001303	-0.051965	0.00
модуль упругости, ГПа	0.049910	0.001303	1.000000	0.043327	-0.00
Количество отвердителя, м.%	0.003969	-0.051965	0.043327	1.000000	0.01
Содержание эпоксидных групп,%_2	0.023834	0.004350	-0.000682	0.010752	1.00
Температура вспышки, С_2	-0.008536	-0.022796	0.034568	0.065849	-0.02
Поверхностная плотность, г/м2	0.011675	0.061442	-0.006727	0.041044	-0.01
Модуль упругости при растяжении, ГПа	-0.026694	-0.014301	0.021232	-0.056787	0.05
Прочность при растяжении, МПа	0.023550	-0.078511	0.033958	-0.062819	-0.00
Потребление смолы, г/м2	0.074926	-0.009771	0.007146	-0.019212	0.01
Угол нашивки	-0.033784	-0.051552	-0.031199	0.032415	0.03
Шаг нашивки	0.036905	-0.045528	0.011484	-0.023263	0.00
Плотность нашивки	0.049060	0.085970	0.076559	0.001731	-0.03

In [100...]

df_minmax_n

Out[100...]

Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Темп, °C	Время, с
0	0.274768	0.651097	0.452951	0.079153	0.607435	(
1	0.274768	0.651097	0.452951	0.630983	0.418887	(
2	0.466552	0.651097	0.461725	0.511257	0.495653	(
3	0.465836	0.571539	0.458649	0.511257	0.495653	(
4	0.424236	0.332865	0.494944	0.511257	0.495653	(
...
921	0.361662	0.444480	0.560064	0.337550	0.333908	(
922	0.607674	0.704373	0.272088	0.749605	0.294428	(
923	0.573391	0.498274	0.254927	0.501991	0.623085	(
924	0.662497	0.748688	0.454635	0.717585	0.267818	(
925	0.684036	0.280923	0.255222	0.632264	0.888354	(

926 rows × 13 columns

In [101...]

df_norm_n

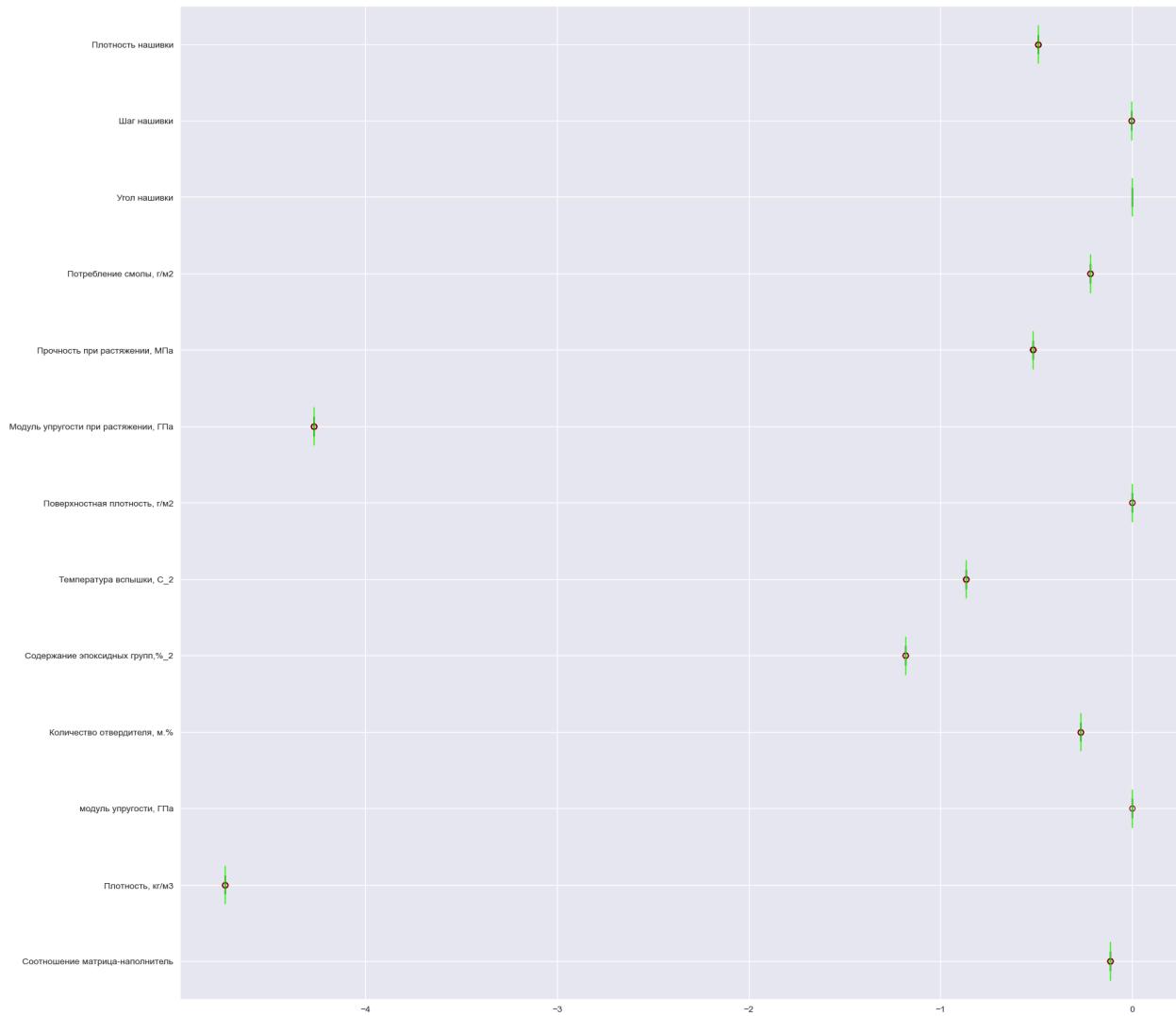
Out[101...]

Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Темп, °C	Время выдержки, с
0	0.000499	0.545436	0.198490	0.013434	0.006381	(0, 10)
1	0.000499	0.545011	0.198335	0.034634	0.005705	(0, 10)
2	0.000744	0.544829	0.202097	0.030022	0.005976	(0, 10)
3	0.000746	0.539271	0.201687	0.030161	0.006004	(0, 10)
4	0.000699	0.519919	0.219673	0.030449	0.006062	(0, 10)
...
921	0.000700	0.601751	0.281397	0.026816	0.006203	(0, 10)
922	0.001078	0.641795	0.139227	0.045701	0.006136	(0, 10)
923	0.000953	0.573123	0.121122	0.032118	0.006961	(0, 10)
924	0.001192	0.664667	0.238453	0.045473	0.006190	(0, 10)
925	0.001071	0.531728	0.117381	0.036336	0.007728	(0, 10)

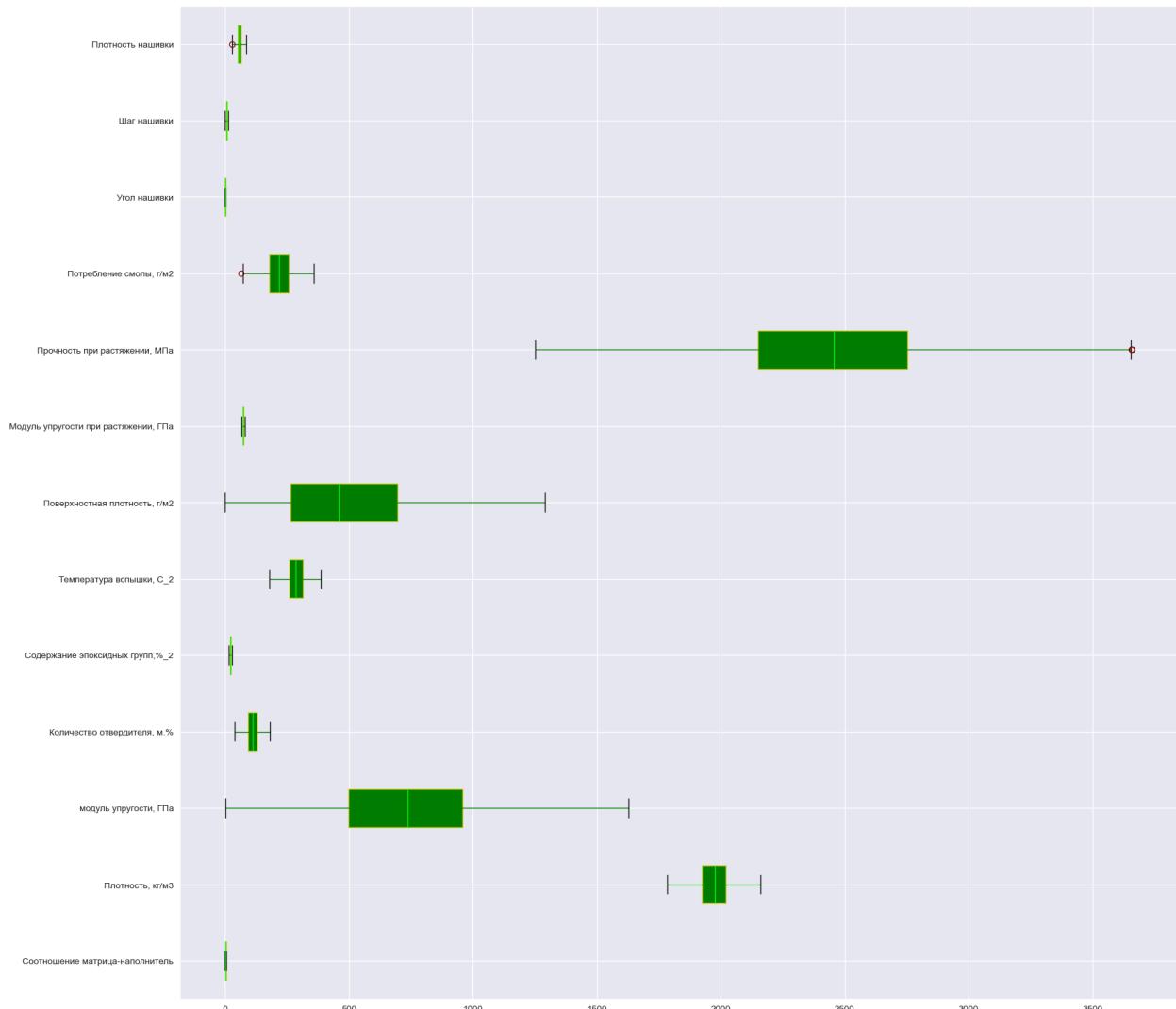
926 rows × 13 columns

In [102...]

```
#Построим на "ящики с усами"
scaler = MinMaxScaler()
scaler.fit(df)
plt.figure(figsize=(20, 20))
#Выводим "ящики"
plt.boxplot(pd.DataFrame(scaler.transform(df_norm_n)), labels = df_norm_n.columns)
plt.show()
```



```
In [103]: # "ящики с усами"
scaler = MinMaxScaler()
scaler.fit(df_minmax_n)
plt.figure(figsize = (20, 20))
#Выводим "ящики"
plt.boxplot(pd.DataFrame(scaler.transform(df)), labels = df_minmax_n.columns,
plt.show()
```

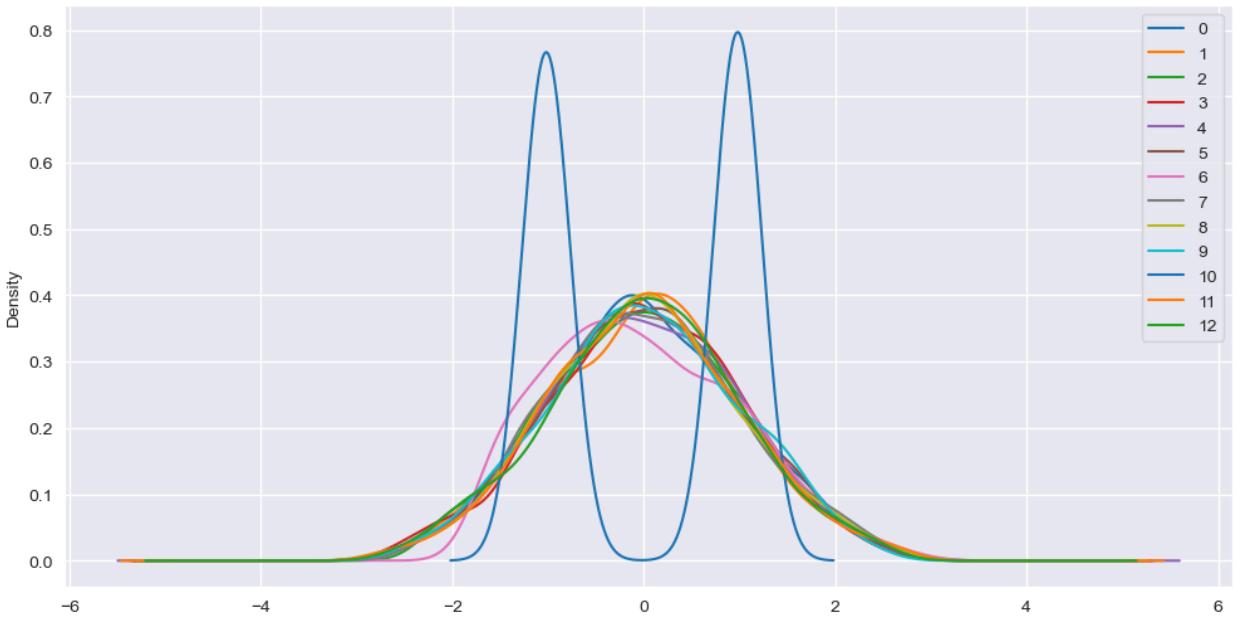


```
In [104]: # Визуализация графиков показывает, что нормализация при помощи "Normalizer" д
```

```
In [105]: ### Стандартизируем данные
```

```
In [106]: X1 = df_minmax_n.copy()
X2 = df_norm_n.copy()
df_std_X1 = preprocessing.StandardScaler().fit(X1)
df_standart_X1 = df_std_X1.transform(X1)
df_standart_1 = pd.DataFrame(df_standart_X1)
fig, ax = plt.subplots(figsize = (12, 6))
df_standart_1.plot(kind = 'kde', ax = ax)
```

```
Out[106]: <Axes: ylabel='Density'>
```



In [107... df_standart_1

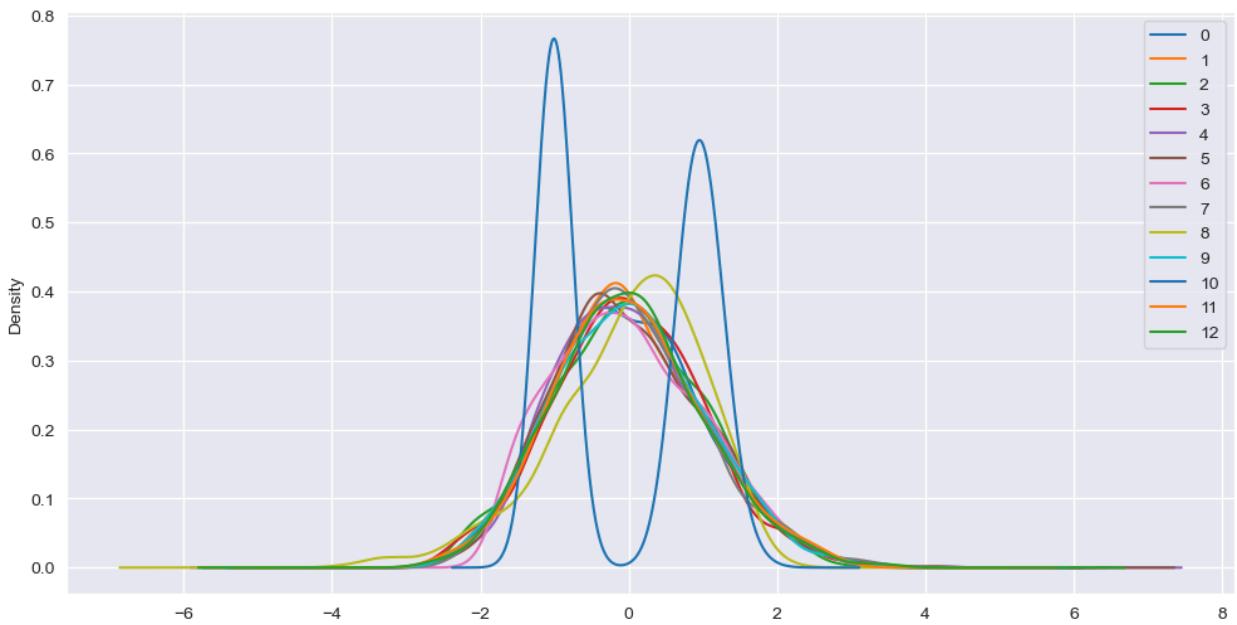
Out[107...

	0	1	2	3	4	5	6
0	-1.198036	0.786865	0.005277	-2.279380	0.645292	-0.038157	-0.974948
1	-1.198036	0.786865	0.005277	0.663561	-0.400223	0.352042	-0.974948
2	-0.176491	0.786865	0.048852	0.025055	0.025451	-0.038157	-0.974948
3	-0.180304	0.364043	0.033577	0.025055	0.025451	-0.038157	-0.974948
4	-0.401889	-0.904425	0.213827	0.025055	0.025451	-0.038157	-0.974948
...
921	-0.735191	-0.311235	0.537224	-0.901331	-0.871437	0.980399	-0.977811
922	0.575198	1.070004	-0.892931	1.296181	-1.090360	-0.809190	-0.472350
923	0.392589	-0.025337	-0.978155	-0.024361	0.732071	-0.956101	0.919313
924	0.867216	1.305524	0.013644	1.125415	-1.237913	-0.262252	0.566738
925	0.981942	-1.180479	-0.976690	0.670394	2.203011	0.376206	0.985792

926 rows × 13 columns

```
In [108... df_std_X2 = preprocessing.StandardScaler().fit(X2)
df_standart_X2 = df_std_X2.transform(X2)
df_standart_2 = pd.DataFrame(df_standart_X2)
fig, ax = plt.subplots(figsize = (12, 6))
df_standart_2.plot(kind = 'kde', ax=ax)
```

Out[108... <Axes: ylabel='Density'>



In [109]: df_standart_2

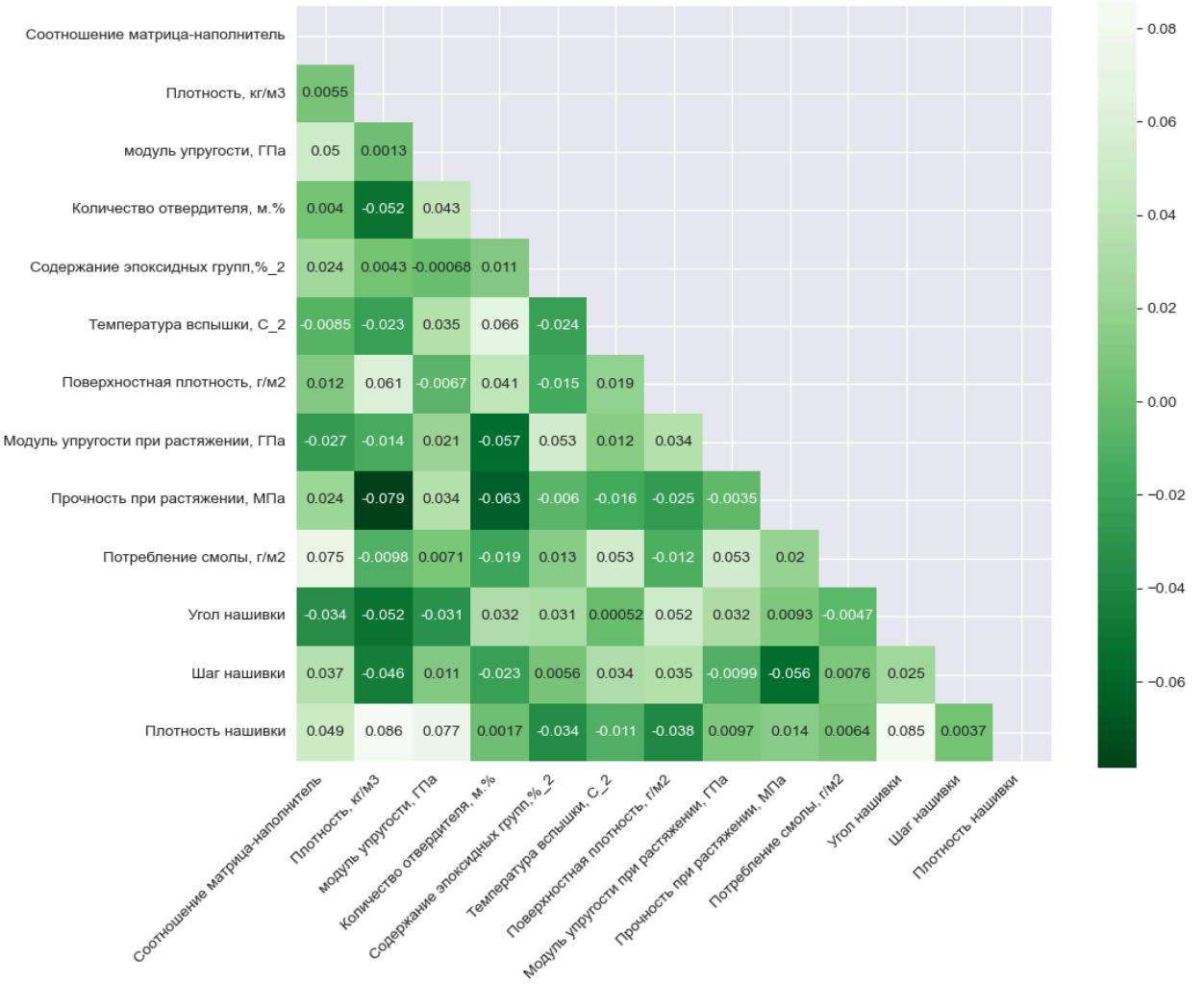
Out[109]:

	0	1	2	3	4	5	6
0	-1.350640	-0.790345	-0.226617	-2.250426	-0.326699	-0.664698	-1.057186
1	-1.352004	-0.796919	-0.228238	0.110139	-0.992819	-0.394499	-1.057712
2	-0.493036	-0.799722	-0.188868	-0.403381	-0.725571	-0.670347	-1.057936
3	-0.484158	-0.885585	-0.193155	-0.387855	-0.698228	-0.646800	-1.054808
4	-0.648303	-1.184499	-0.004938	-0.355799	-0.641771	-0.598181	-1.048349
...
921	-0.645876	0.079523	0.641009	-0.760342	-0.502186	0.904576	-0.960823
922	0.678386	0.698056	-0.846802	1.342463	-0.568538	-0.458188	-0.419577
923	0.240784	-0.362686	-1.036263	-0.169952	0.244798	-0.949255	0.838736
924	1.075785	1.051353	0.191598	1.317056	-0.515512	0.146145	0.733856
925	0.653612	-1.002095	-1.075417	0.299713	0.999930	-0.121878	0.819020

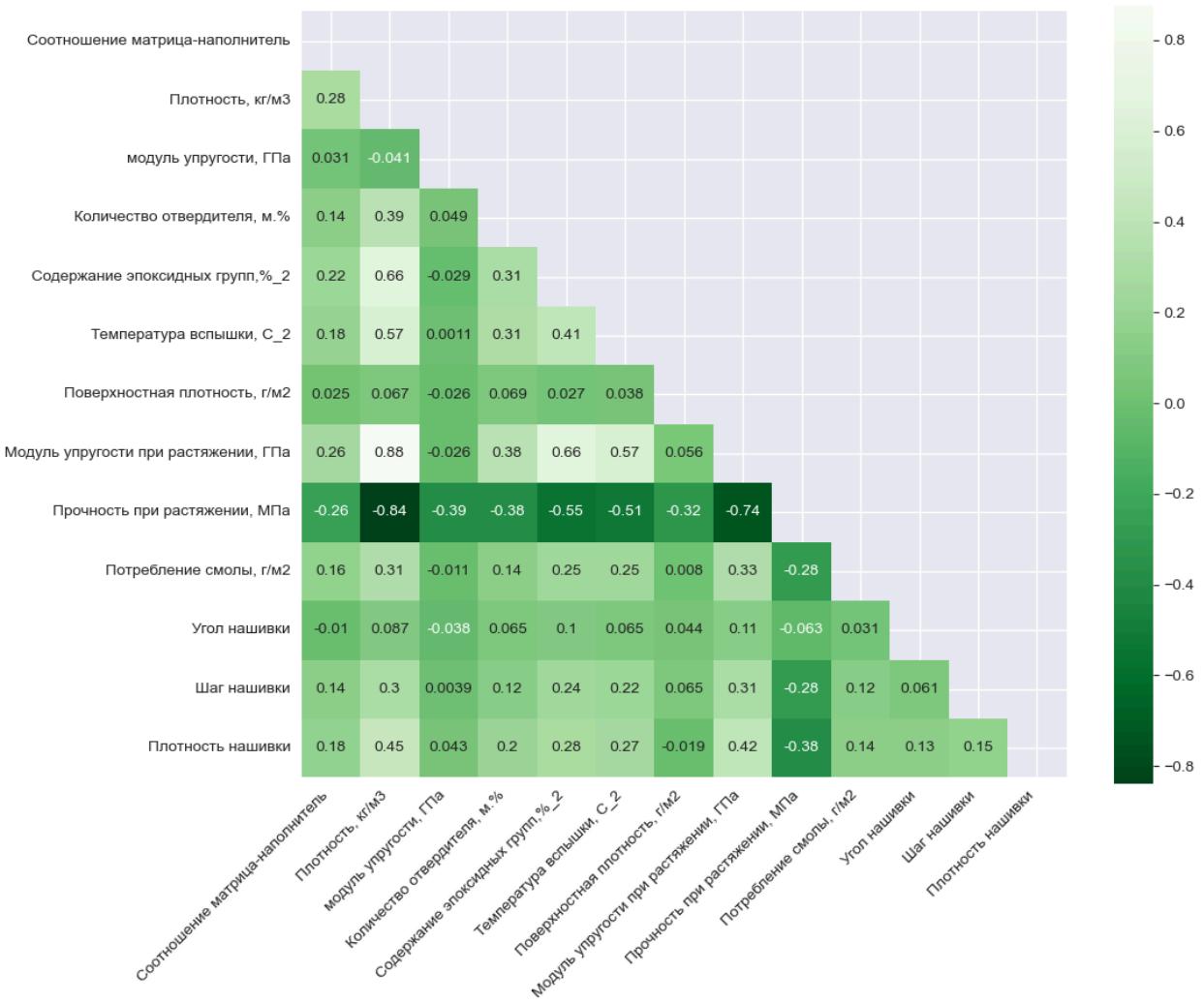
926 rows × 13 columns

In [110]:

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



```
In [111]: mask = np.triu(X2.corr())
f, ax = plt.subplots(figsize=(11, 9))
sns.heatmap(X2.corr(), mask=mask, annot=True, square=True, cmap='Greens_r')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [112]: df_norm_n.describe()

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
count	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000
mean	0.000885	0.596603	0.220144	0.033645	0.006713	0.000163	0.444650	0.000709	0.011339	0.004113	0.000163	0.001068	0.001803
std	0.000286	0.064775	0.095609	0.008985	0.001016	0.000685	0.552236	0.150901	0.027607	0.005979	0.000871	0.001068	0.002421
min	0.000163	0.444650	0.000709	0.011339	0.004113	0.000163	0.444650	0.000709	0.011339	0.004113	0.000163	0.001068	0.001803
25%	0.000685	0.552236	0.150901	0.027607	0.005979	0.000871	0.591816	0.219954	0.033559	0.006648	0.001068	0.001068	0.002421
50%	0.000871	0.591816	0.219954	0.033559	0.006648	0.001068	0.640169	0.289231	0.039565	0.007345	0.001068	0.001068	0.002421
75%	0.001068	0.640169	0.289231	0.039565	0.007345	0.001068	0.640169	0.325102	0.0462919	0.010887	0.001068	0.001068	0.002421
max	0.001803	0.824241	0.525102	0.062919	0.010887	0.001803	0.824241	0.525102	0.062919	0.010887	0.001803	0.001803	0.002421

```
In [113]: df_minmax_n.describe()
```

Out[113]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура кurenia, С
count	926.000000	926.000000	926.000000	926.000000	926.000000	926.000000
mean	0.499687	0.503041	0.451888	0.506559	0.491063	100.000000
std	0.187841	0.188261	0.201469	0.187611	0.180438	100.000000
min	0.000000	0.000000	0.000000	0.000000	0.000000	100.000000
25%	0.373077	0.368184	0.305669	0.378514	0.367101	100.000000
50%	0.494936	0.511740	0.452599	0.506532	0.488912	100.000000
75%	0.629774	0.625266	0.587461	0.639120	0.623296	100.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	100.000000

```
In [114]: ### Разработка и обучение нескольких моделей для прогноза прочности при растяжении
```

```
In [115]: #Данные в нашем итоговом датасете в основном непрерывные, используем регрессию
#Но попарные графики рассеивания точек и тепловая карта не дают нам реальной картины
#поэтому будем использовать и категориальные подходы к прогнозированию (например,
#или обучение со скрытыми слоями, чтобы выявить дополнительные взаимосвязи.
```

```
In [116]: # Прогнозируем прочность при растяжении. Для обучения используем 70 % данных,
from sklearn.model_selection import train_test_split # Добавляем импорт

# Разделение данных
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_norm_n[['Прочность при растяжении, МПа']], # Исправляем на df_norm_n
    test_size=0.3,
    random_state=42
)

# Проверка корректности разделения
assert df_norm_n.shape[0] == x_train_1.shape[0] + x_test_1.shape[0], "Ошибка в разделении"

# Проверка размеров
print(f"Исходный размер: {df_norm_n.shape[0]}")
print(f"Размер train: {x_train_1.shape[0]}")
print(f"Размер test: {x_test_1.shape[0]}")
print(f"Разница: {df_norm_n.shape[0] - x_train_1.shape[0] - x_test_1.shape[0]}")

# Дополнительная проверка на совпадение индексов
assert x_train_1.index.equals(y_train_1.index), "Индексы признаков и целевой груп"
assert x_test_1.index.equals(y_test_1.index), "Индексы признаков и целевой групп"
```

Исходный размер: 926

Размер train: 648

Размер test: 278

Разница: 0

In [118... x_train_1.head()

Out[118...

Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Темп, вс
888	0.000962	0.611056	0.231597	0.035655	0.005740
368	0.001038	0.684067	0.313007	0.030022	0.008396
651	0.000687	0.578411	0.070642	0.022260	0.006792
652	0.000981	0.570442	0.401106	0.029510	0.005677
490	0.000681	0.588044	0.059991	0.028152	0.007266

In [120... y_train_1

Out[120... Прочность при растяжении, МПа

888	0.726716
368	0.609143
651	0.777192
652	0.686935
490	0.778695
...	...
106	0.711398
270	0.769917
860	0.600012
435	0.673210
102	0.516461

648 rows × 1 columns

In [121... y_test_1

Out[121...]

Прочность при растяжении, МПа

323	0.723626
601	0.711288
30	0.813965
823	0.834088
294	0.760643
...	...
552	0.760595
404	0.794416
309	0.687828
133	0.672247
497	0.595539

278 rows × 1 columns

In [122...]

y_train_1.shape

Out[122...]

(648, 1)

In [123...]

```
def mean_model(y_test_1):
    return [np.mean(y_test_1) for _ in range(len(y_test_1))]
y_1_pred_mean = mean_model(y_test_1)
```

In [124...]

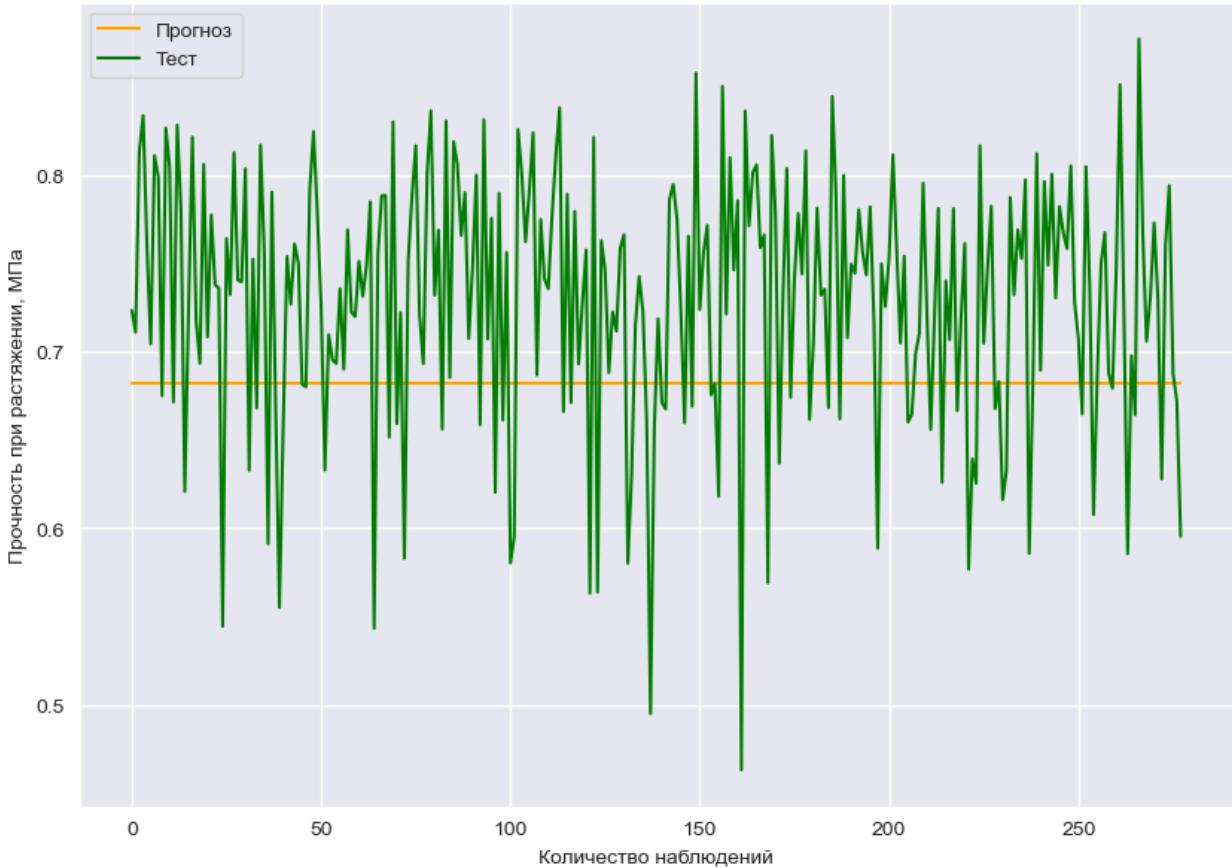
```
svr = make_pipeline(StandardScaler(), SVR(kernel = 'rbf', C = 500.0, epsilon =
#обучаем модель
svr.fit(x_train_1, np.ravel(y_train_1))
#вычисляем коэффициент детерминации
y_pred_svr=svr.predict(x_test_1)
mae_svr = mean_absolute_error(y_pred_svr, y_test_1)
mse_svr_elast = mean_squared_error(y_test_1,y_pred_svr)
print('Support Vector Regression Results Train:')
print("Test score: {:.2f}".format(svr.score(x_train_1, y_train_1))) # Скор для
print('Support Vector Regression Results:')
print('SVR_MAE: ', round(mean_absolute_error(y_test_1, y_pred_svr)))
print('SVR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_svr)))
print('SVR_MSE: {:.2f}'.format(mse_svr_elast))
print("SVR_RMSE: {:.2f}".format(np.sqrt(mse_svr_elast)))
print("Test score: {:.2f}".format(svr.score(x_test_1, y_test_1))) # Скор для т
```

```
Support Vector Regression Results Train:  
Test score: -0.52  
Support Vector Regression Results:  
SVR_MAE: 0  
SVR_MAPE: 0.10  
SVR_MSE: 0.01  
SVR_RMSE: 0.09  
Test score: -0.45
```

```
In [125...]: #Результаты модели, выдающей среднее значение  
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: 0.05606395529005198  
MSE for mean target: 0.005011156544381145  
RMSE for mean target: 0.0707895228432933
```

```
In [126...]: plt.figure(figsize = (10, 7))  
plt.title("Тестовые и прогнозные значения Support Vector Regression")  
plt.plot(y_pred_svr, label = "Прогноз", color = "orange")  
plt.plot(y_test_1.values, label = "Тест", color = "green")  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Прочность при растяжении, МПа")  
plt.legend()  
plt.grid(True);
```

Тестовые и прогнозные значения Support Vector Regression



```
In [127...]: #построение модели и визуализация метода случайный лес
rfr = RandomForestRegressor(n_estimators=15,max_depth=7, random_state=33)
rfr.fit(x_train_1, y_train_1.values)
y_pred_forest = rfr.predict(x_test_1)
mae_rfr = mean_absolute_error(y_pred_forest, y_test_1)
mse_rfr_elast = mean_squared_error(y_test_1,y_pred_forest)
print('Random Forest Regressor Results Train:')
print("Test score: {:.2f}".format(rfr.score(x_train_1, y_train_1))) # Скор для тренировки
print('Random Forest Regressor Results:')
print('RF_MAE: ', round(mean_absolute_error(y_test_1, y_pred_forest)))
print('RF_MAPE: {:.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 теста
```

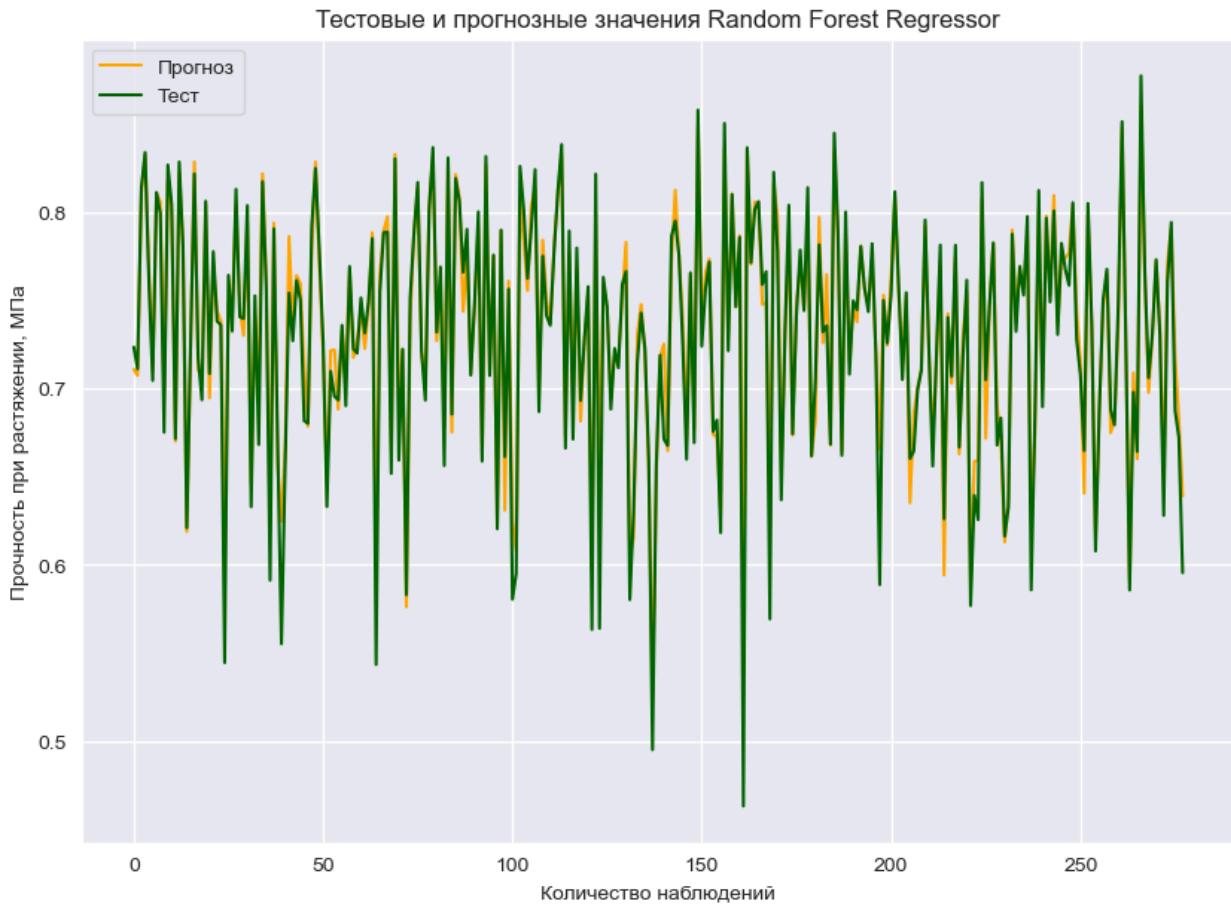
```
Random Forest Regressor Results Train:
Test score: 0.99
Random Forest Regressor Results:
RF_MAE:  0
RF_MAPE: 0.02
RF_MSE: 0.00
RF_RMSE: 0.02
Test score: 0.94
```

```
In [128...]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Random Forest Regressor")
```

```

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("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);

```



In [129...]:

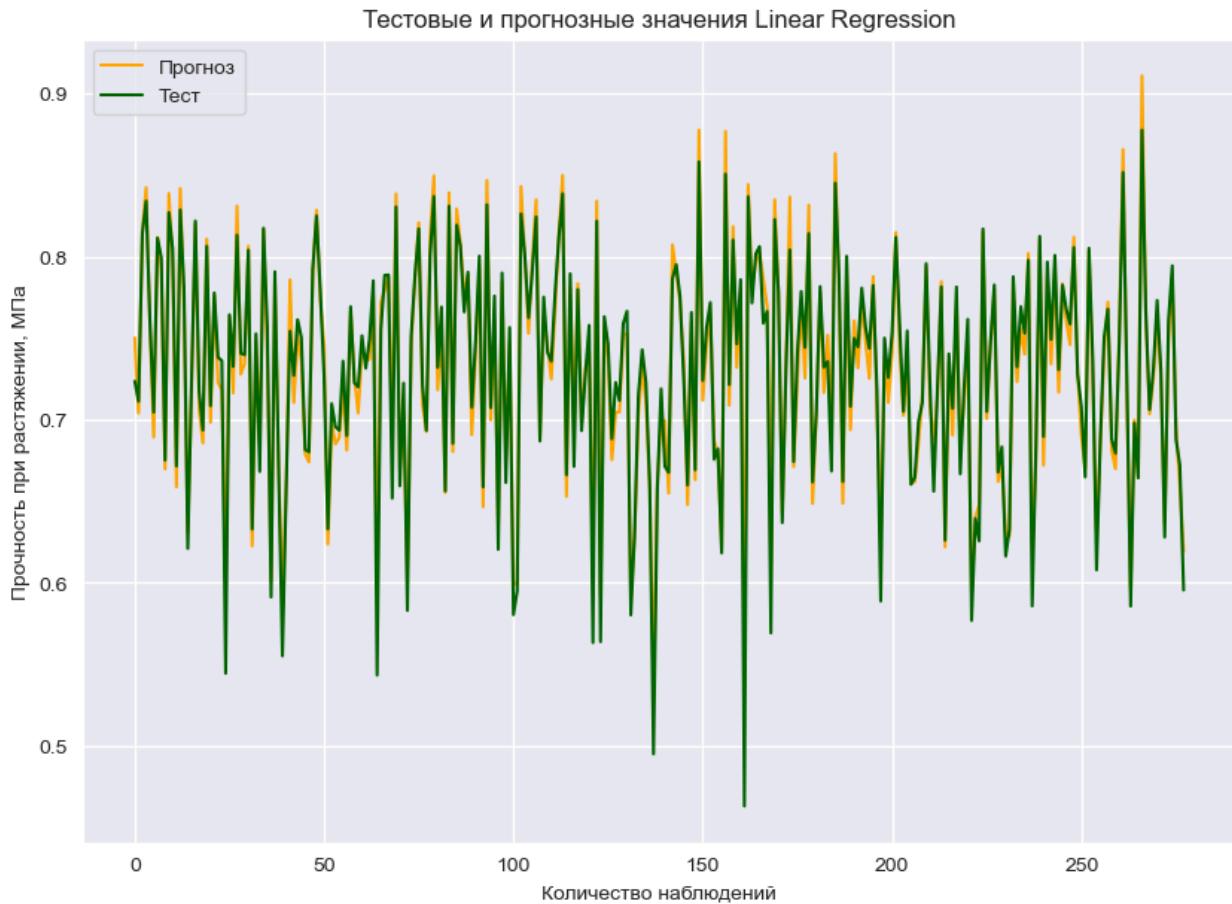
```

#построение модели и визуализация Линейной регрессии
lr = LinearRegression()
lr.fit(x_train_1, y_train_1)
y_pred_lr = lr.predict(x_test_1)
mae_lr = mean_absolute_error(y_pred_lr, y_test_1)
mse_lin_elast = mean_squared_error(y_test_1, y_pred_lr)
print('Linear Regression Results Train: ') # Скор для тренировочной выборки
print("Test score: {:.2f}".format(lr.score(x_train_1, y_train_1)))
print('Linear Regression Results:')
print('lr_MAE: ', round(mean_absolute_error(y_test_1, y_pred_lr)))
print('lr_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_lr)))
print('lr_MSE: {:.2f}'.format(mse_lin_elast))
print("lr_RMSE: {:.2f}".format(np.sqrt(mse_lin_elast)))
print("Test score: {:.2f}".format(lr.score(x_test_1, y_test_1))) # Скор для тес

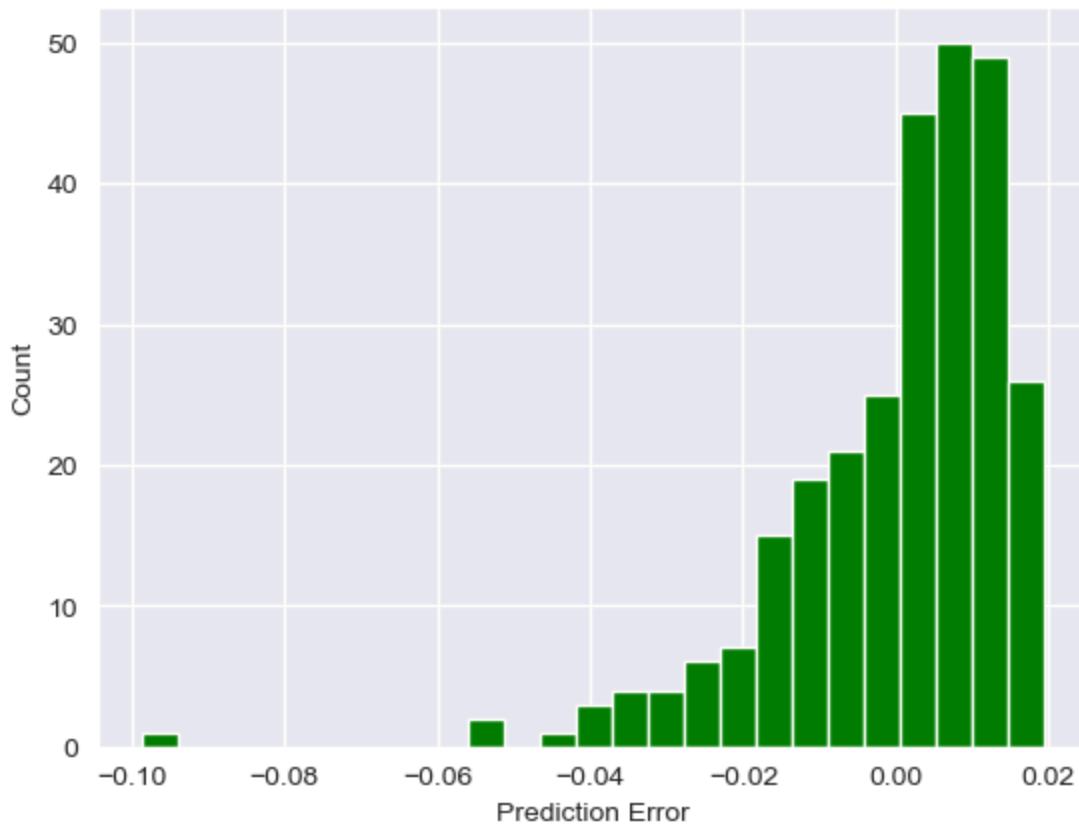
```

```
Linear Regression Results Train:  
Test score: 0.96  
Linear Regression Results:  
lr_MAE: 0  
lr_MAPE: 0.02  
lr_MSE: 0.00  
lr_RMSE: 0.02  
Test score: 0.95
```

```
In [130...]: 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("Прочность при растяжении, МПа")  
plt.legend()  
plt.grid(True);
```



```
In [134...]: #Визуализация гистограммы распределения ошибки  
error = y_test_1 - y_pred_lr  
plt.hist(error, bins = 25, color = "g")  
plt.xlabel('Prediction Error')  
_ = plt.ylabel('Count')
```



```
In [135]: #Метод градиентного бустинга
gbr = make_pipeline(StandardScaler(), GradientBoostingRegressor())
gbr.fit(x_train_1, np.ravel(y_train_1))
y_pred_gbr = gbr.predict(x_test_1)
mae_gbr = mean_absolute_error(y_pred_gbr, y_test_1)
mse_gbr_elast = mean_squared_error(y_test_1,y_pred_gbr)
print('Gradient Boosting Regressor Results Train:')
print("Test score: {:.2f}".format(gbr.score(x_train_1, y_train_1))) # Скор для
print('Gradient Boosting Regressor Results:')
print('GBR_MAE: ', round(mean_absolute_error(y_test_1, y_pred_gbr)))
print('GBR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_gbr)))
print('GBR_MSE: {:.2f}'.format(mse_gbr_elast))
print("GBR_RMSE: {:.2f}".format(np.sqrt(mse_gbr_elast)))
print("Test score: {:.2f}".format(gbr.score(x_test_1, y_test_1)))# Скор для тес
```

Gradient Boosting Regressor Results Train:

Test score: 1.00

Gradient Boosting Regressor Results:

GBR_MAE: 0

GBR_MAPE: 0.01

GBR_MSE: 0.00

GBR_RMSE: 0.01

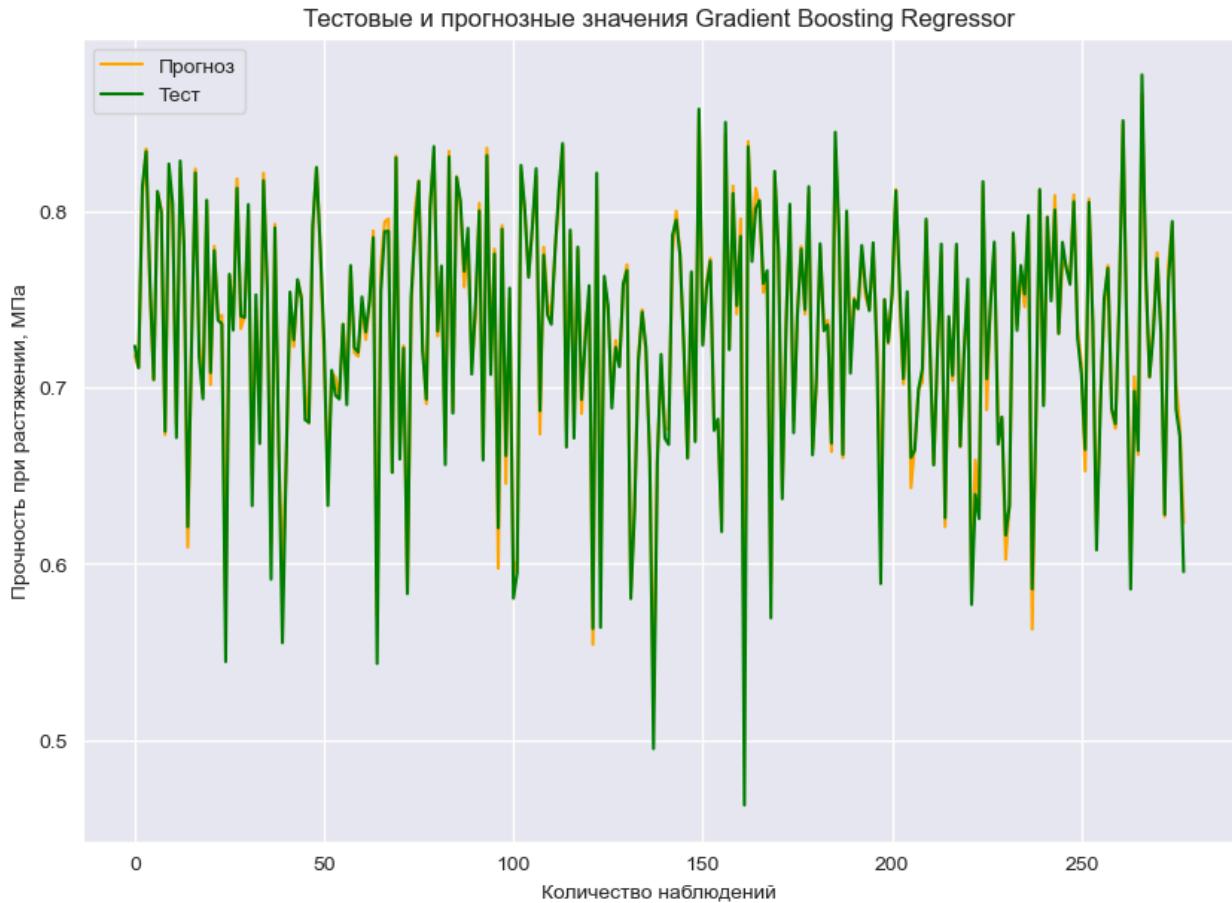
Test score: 0.98

```
In [136]: 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.values, label = "Тест", color = "green")
```

```

plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);

```



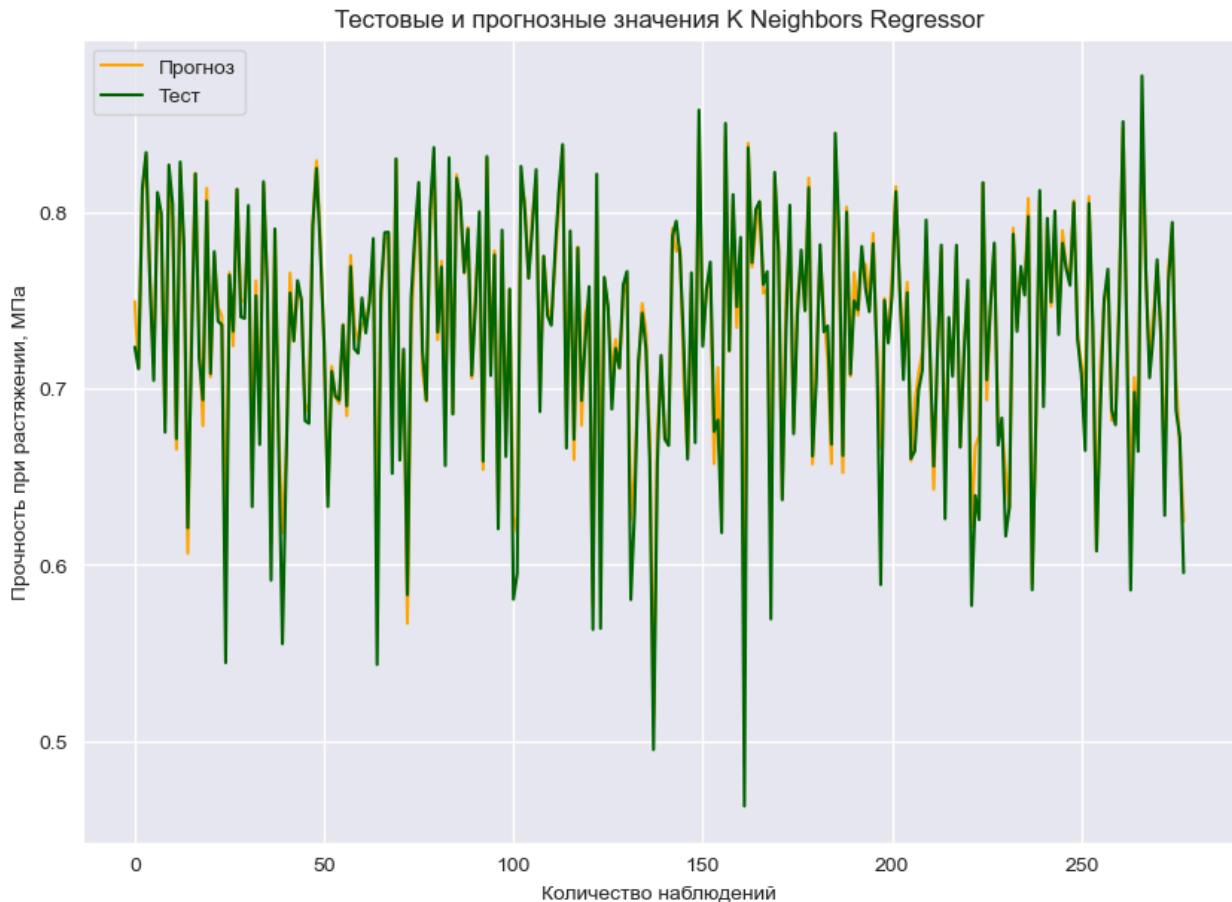
```

In [137]: # 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_knr = mean_absolute_error(y_pred_knn, y_test_1)
mse_knn_elast = mean_squared_error(y_test_1,y_pred_knn)
print('K Neighbors Regressor Results Train:')
print("Test score: {:.2f}".format(knn.score(x_train_1, y_train_1)))# Скор для
print('K Neighbors Regressor Results:')
print('KNN_MAE: ', round(mean_absolute_error(y_test_1, y_pred_knn)))
print('KNN_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_knn)))
print('KNN_MSE: {:.2f}'.format(mse_knn_elast))
print("KNN_RMSE: {:.2f}".format(np.sqrt(mse_knn_elast)))
print("Test score: {:.2f}".format(knn.score(x_test_1, y_test_1)))# Скор для тэ

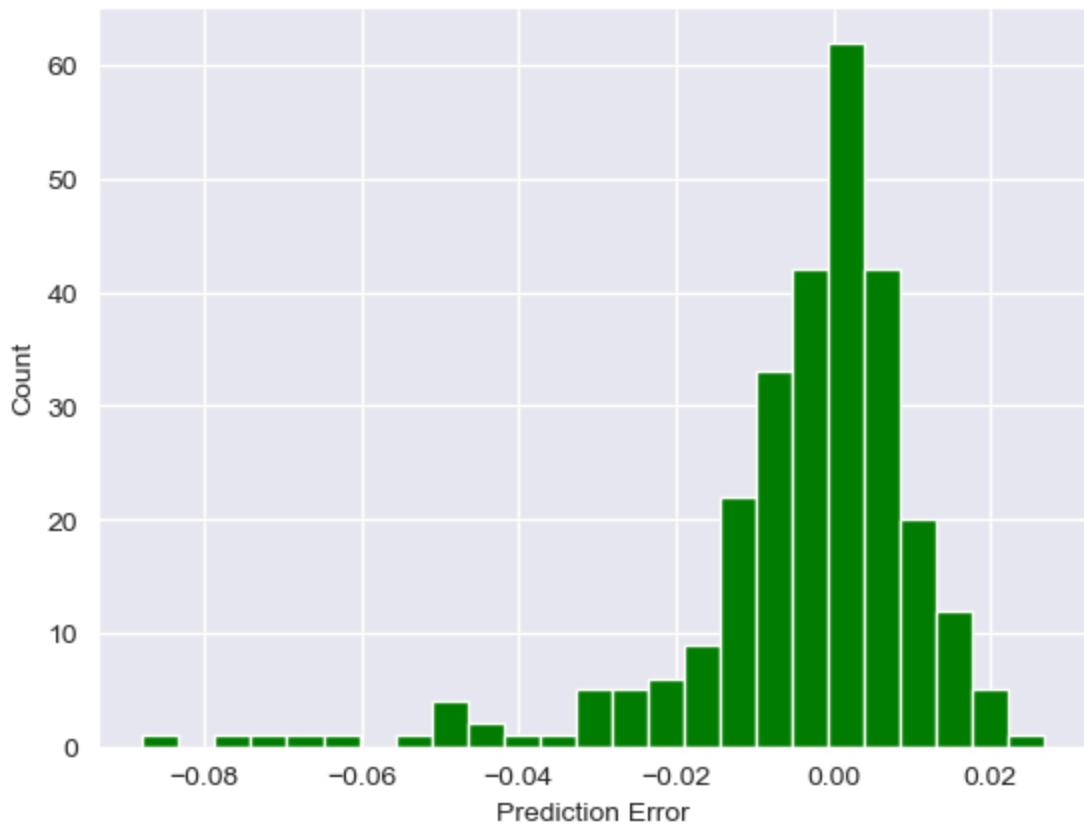
```

```
K Neighbors Regressor Results Train:  
Test score: 0.97  
K Neighbors Regressor Results:  
KNN_MAE: 0  
KNN_MAPE: 0.02  
KNN_MSE: 0.00  
KNN_RMSE: 0.02  
Test score: 0.95
```

```
In [138...]: 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.values, label = "Тест", color = 'darkgreen')  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Прочность при растяжении, МПа")  
plt.legend()  
plt.grid(True);
```



```
In [139...]: #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 [140...]: #Деревья решений - Decision Tree Regressor - 6

```
dtr = DecisionTreeRegressor()
dtr.fit(x_train_1, y_train_1.values)
y_pred_dtr = dtr.predict(x_test_1)
mae_dtr = mean_absolute_error(y_pred_dtr, y_test_1)
mse_dtr_elast = mean_squared_error(y_test_1,y_pred_dtr)
print('Decision Tree Regressor Results Train:')
print("Test score: {:.2f}".format(knn.score(x_train_1, y_train_1)))# Скор для
print('Decision Tree Regressor Results:')
print('DTR_MAE: ', round(mean_absolute_error(y_test_1, y_pred_dtr)))
print('DTR_MSE: {:.2f}'.format(mse_dtr_elast))
print("DTR_RMSE: {:.2f}".format (np.sqrt(mse_dtr_elast)))
print('DTR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_dtr)))
print("Test score: {:.2f}".format(dtr.score(x_test_1, y_test_1)))# Скор для тес
```

Decision Tree Regressor Results Train:

Test score: 0.97

Decision Tree Regressor Results:

DTR_MAE: 0

DTR_MSE: 0.00

DTR_RMSE: 0.02

DTR_MAPE: 0.02

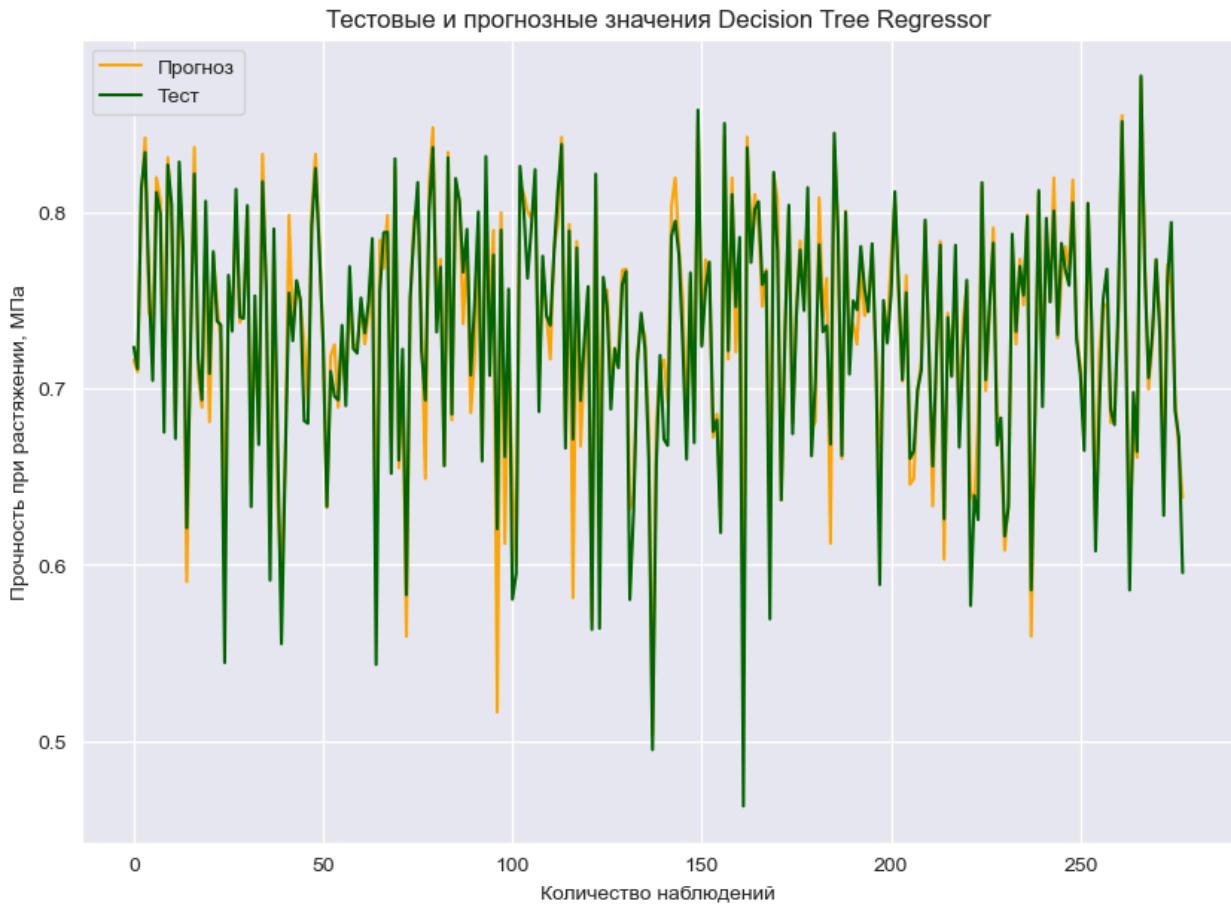
Test score: 0.90

In [141...]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Decision Tree Regressor")
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 [142...]: # Стохастический градиентный спуск (SGD) - Stochastic Gradient Descent Regressor

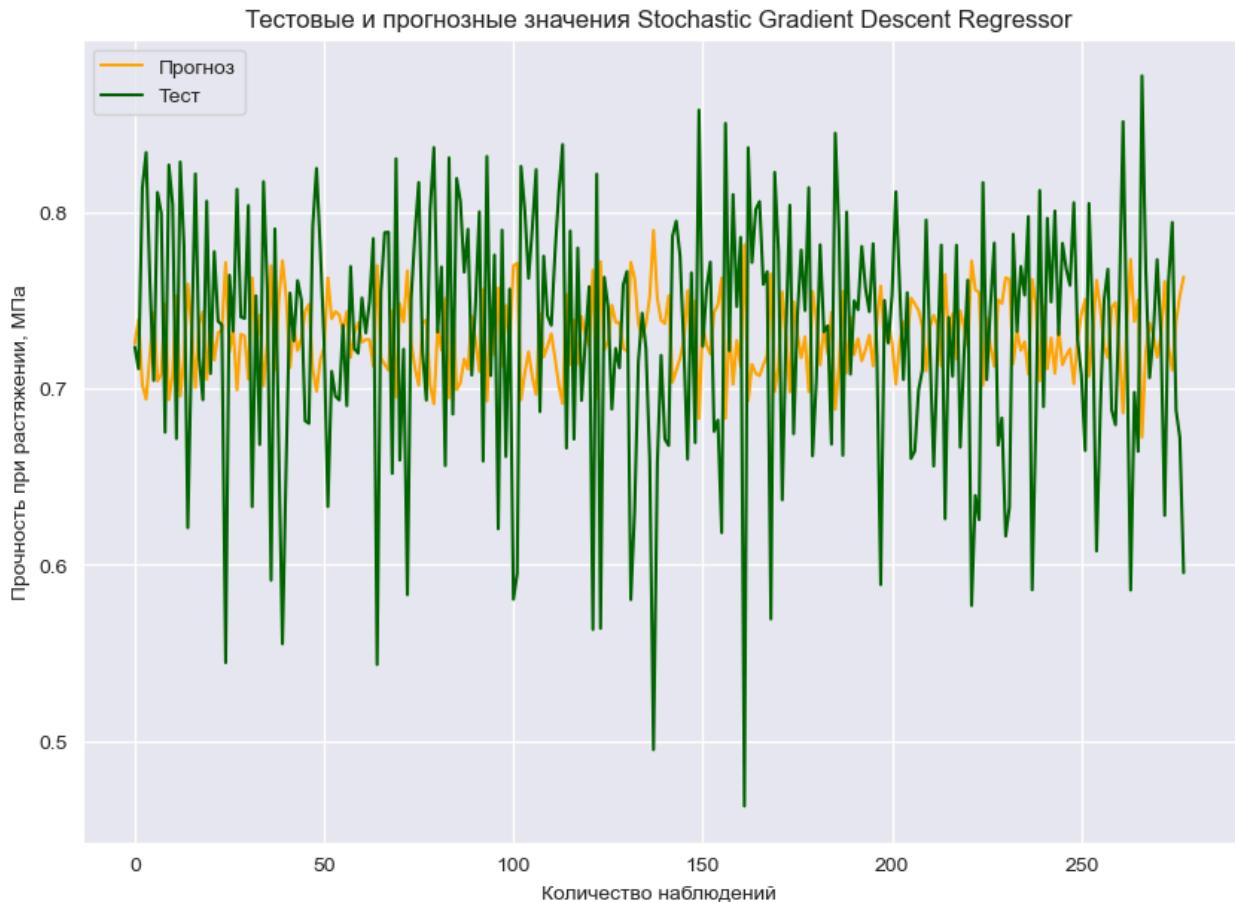
```

sgd = SGDRegressor()
sgd.fit(x_train_1, y_train_1)
y_pred_sdg = sgd.predict(x_test_1)
mae_sdg = mean_absolute_error(y_pred_sdg, y_test_1)
mse_sdg_elast = mean_squared_error(y_test_1,y_pred_sdg)
print('Stochastic Gradient Descent Regressor Results Train:')
print("Test score: {:.2f}".format(sgd.score(x_train_1, y_train_1)))# Скор для тренировки
print('Stochastic Gradient Descent Regressor Results:')
print('SGD_MAE: ', round(mean_absolute_error(y_test_1, y_pred_sdg)))
print('SGD_MSE: {:.2f}'.format(mse_sdg_elast))
print("SGD_RMSE: {:.2f}".format(np.sqrt(mse_sdg_elast)))
print('SGD_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_sdg)))
print("Test score: {:.2f}".format(sgd.score(x_test_1, y_test_1)))# Скор для теста

```

```
Stochastic Gradient Descent Regressor Results Train:  
Test score: -0.70  
Stochastic Gradient Descent Regressor Results:  
SGD_MAE: 0  
SGD_MSE: 0.01  
SGD_RMSE: 0.09  
SGD_MAPE: 0.10  
Test score: -0.67
```

```
In [143...]: plt.figure(figsize = (10, 7))  
plt.title("Тестовые и прогнозные значения Stochastic Gradient Descent Regressor")  
plt.plot(y_pred_sdg, 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 [144...]: # Многослойный перцептрон - Multi-layer Perceptron regressor - 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 regressor Results Train:')
```

```

print("Test score: {:.2f}".format(mlp.score(x_train_1, y_train_1)))# Скор для
print('Multi-layer Perceptron regressor Results:')
print('SGD_MAE: ', round(mean_absolute_error(y_test_1, y_pred_mlp)))
print('SGD_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_mlp)))
print('SGD_MSE: {:.2f}'.format(mse_mlp_elast))
print("SGD_RMSE: {:.2f}".format(np.sqrt(mse_mlp_elast)))
print("Test score: {:.2f}".format(mlp.score(x_test_1, y_test_1)))# Скор для тес

```

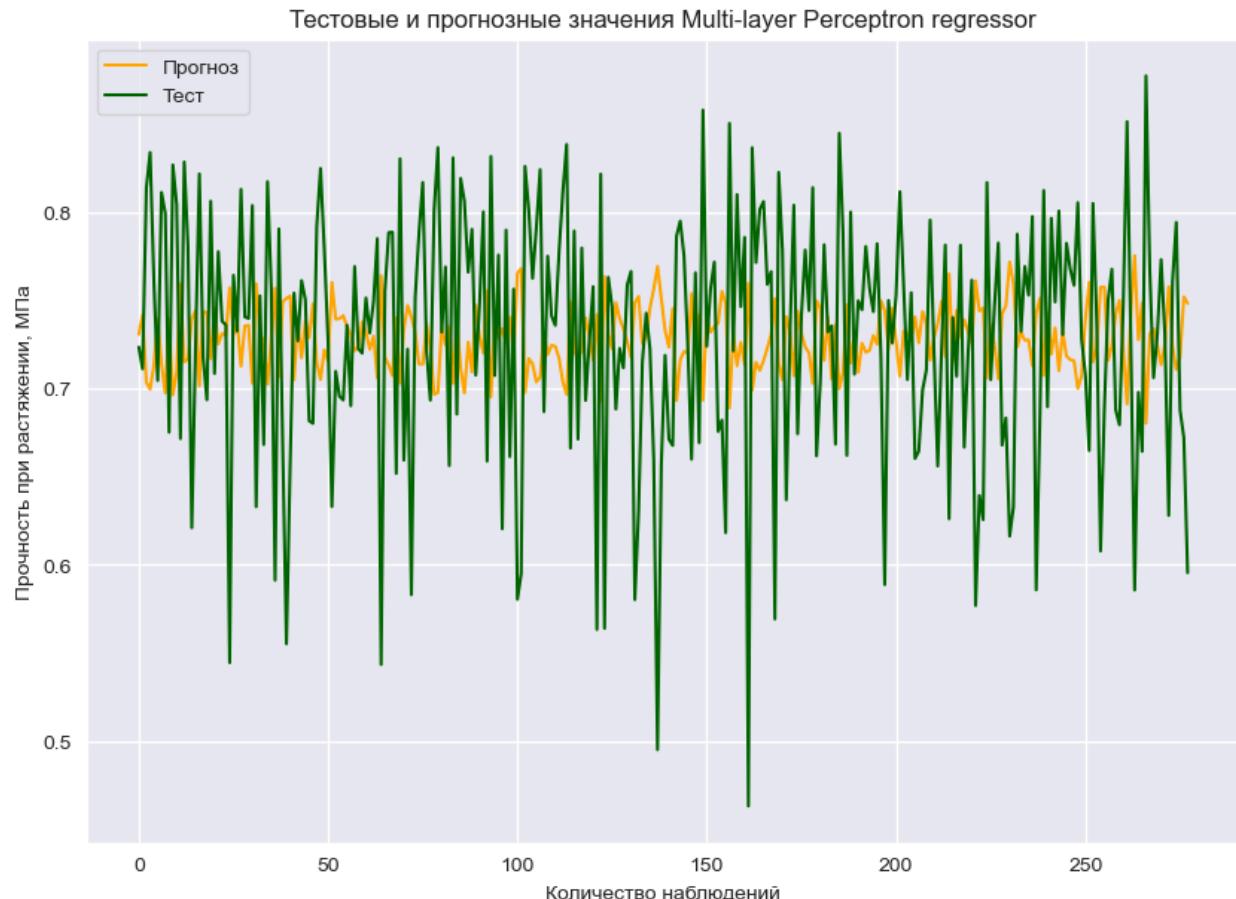
Multi-layer Perceptron regressor Results Train:
 Test score: -0.55
 Multi-layer Perceptron regressor Results:
 SGD_MAE: 0
 SGD_MAPE: 0.10
 SGD_MSE: 0.01
 SGD_RMSE: 0.09
 Test score: -0.50

In [153...]:

```

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

```



```
In [157]: # Лассо регрессия - the Lasso - 9
from sklearn.linear_model import Lasso
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    mean_absolute_percentage_error
)
import numpy as np

# Инициализация модели Lasso
clf = Lasso(alpha=0.1)

# Обучение модели
clf.fit(x_train_1, y_train_1.values.ravel())

# Предсказания
y_pred_clf = clf.predict(x_test_1)

# Расчет метрик
mae_clf = mean_absolute_error(y_test_1, y_pred_clf)
mse_clf = mean_squared_error(y_test_1, y_pred_clf)
mape_clf = mean_absolute_percentage_error(y_test_1, y_pred_clf)
rmse_clf = np.sqrt(mse_clf)

# Вывод результатов
print("Lasso Regression Results:")
print("\nМетрики на обучающей выборке:")
print(f"Train R2: {clf.score(x_train_1, y_train_1):.4f}")

print("\nМетрики на тестовой выборке:")
print(f"Test R2: {clf.score(x_test_1, y_test_1):.4f}")
print(f"MAE: {mae_clf:.2f}")
print(f"MAPE: {mape_clf:.2%}")
print(f"MSE: {mse_clf:.2f}")
print(f"RMSE: {rmse_clf:.2f}")

# Дополнительно: проверка важности признаков
print("\nВажность признаков (коэффициенты):")
for feature, coef in zip(x_train_1.columns, clf.coef_):
    print(f"{feature}: {coef:.4f}")
```

Lasso Regression Results:

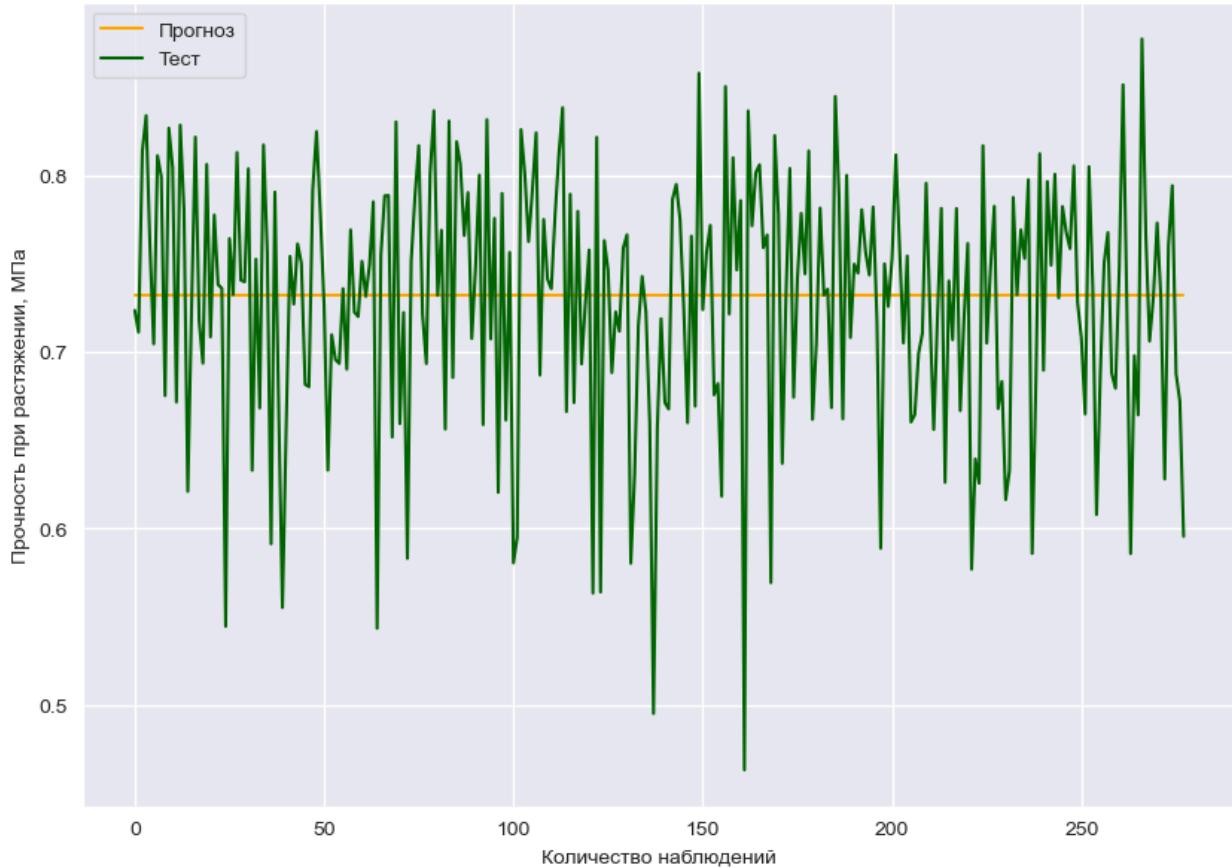
Метрики на обучающей выборке:
Train R²: 0.0000

Метрики на тестовой выборке:
Test R²: -0.0012
MAE: 0.06
MAPE: 8.05%
MSE: 0.01
RMSE: 0.07

Важность признаков (коэффициенты):
Соотношение матрица-наполнитель: -0.0000
Плотность, кг/м3: -0.0000
модуль упругости, ГПа: -0.0000
Количество отвердителя, м.%: -0.0000
Содержание эпоксидных групп,%_2: -0.0000
Температура вспышки, С_2: -0.0000
Поверхностная плотность, г/м2: -0.0000
Модуль упругости при растяжении, ГПа: -0.0000
Потребление смолы, г/м2: -0.0000
Угол нашивки: -0.0000
Шаг нашивки: -0.0000
Плотность нашивки: -0.0000

```
In [158]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Lasso regressor")
plt.plot(y_pred_clf, label = "Прогноз", color = 'orange')
plt.plot(y_test_1.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);
```

Тестовые и прогнозные значения Lasso regressor



```
In [160...]: #сравним наши модели по метрике MAE
mae_df = {'Регрессор': ['Support Vector', 'RandomForest', 'Linear Regression',
                        'GradientBoosting'],
          'MAE': [0.072198, 0.010823, 0.011203, 0.005985]}

mae_df = pd.DataFrame(mae_df)
mae_df
```

	Регрессор	MAE
0	Support Vector	0.072198
1	RandomForest	0.010823
2	Linear Regression	0.011203
3	GradientBoosting	0.005985
4	KNeighbors	0.010085
5	DecisionTree	0.014322
6	SGD	0.073114
7	MLP	0.069747
8	Lasso	0.055824

```
In [ ]: #GradientBoosting показывает превосходный результат с минимальным MAE
```

```
#RandomForest и KNeighbors демонстрируют сопоставимую точность
```

```
#Linear Regression также показывает хорошие результаты
```

In [168...]

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Инициализация модели
rfr = RandomForestRegressor(random_state=42)

# Определение сетки гиперпараметров
param_grid = {
    'n_estimators': [200, 300],
    'max_depth': [9, 15],
    'max_features': ['sqrt'], # Исправлено значение
    'criterion': ['squared_error'],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Настройка GridSearchCV
grid_search = GridSearchCV(
    estimator=rfr,
    param_grid=param_grid,
    cv=10,
    scoring='neg_mean_absolute_error', # Используем MAE как метрику
    n_jobs=-1, # Используем все ядра процессора
    verbose=1
)

# Обучение модели
grid_search.fit(x_train_1, y_train_1.values.ravel())

# Вывод лучших параметров
print("Лучшие параметры:")
print(grid_search.best_params_)

# Оценка модели на тестовой выборке
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test_1)

mae = mean_absolute_error(y_test_1, y_pred)
mse = mean_squared_error(y_test_1, y_pred)
rmse = np.sqrt(mse)

print("\nРезультаты на тестовой выборке:")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
```

```

# Важность признаков
feature_importances = best_model.feature_importances_
print("\nВажность признаков:")
for feature, importance in zip(x_train_1.columns, feature_importances):
    print(f"{feature}: {importance:.4f}")

```

Fitting 10 folds for each of 16 candidates, totalling 160 fits

Лучшие параметры:

```
{'criterion': 'squared_error', 'max_depth': 15, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

Результаты на тестовой выборке:

MAE: 0.0159

MSE: 0.0006

RMSE: 0.0236

Важность признаков:

Соотношение матрица-наполнитель: 0.0159

Плотность, кг/м3: 0.3369

модуль упругости, ГПа: 0.1343

Количество отвердителя, м.%: 0.0258

Содержание эпоксидных групп,%_2: 0.0693

Температура вспышки, С_2: 0.0726

Поверхностная плотность, г/м2: 0.0585

Модуль упругости при растяжении, ГПа: 0.1661

Потребление смолы, г/м2: 0.0170

Угол нашивки: 0.0547

Шаг нашивки: 0.0197

Плотность нашивки: 0.0291

In [175... mae_df

Out[175... Регрессор MAE

	Регрессор	MAE
0	Support Vector	0.072198
1	RandomForest	0.010823
2	Linear Regression	0.011203
3	GradientBoosting	0.005985
4	KNeighbors	0.010085
5	DecisionTree	0.014322
6	SGD	0.073114
7	MLP	0.069747
8	Lasso	0.055824

In []: #набор показывает хорошие значения

#Использовать GradientBoosting как основную модель, так как: Показывает минимальное значение MAE. Хорошая точность. Более стабильная работа

```
In [178...]: #Прогнозируем модуль упругости при растяжении, ГПа
```

```
In [190...]: #разбиваем на тестовую, тренировочную выборки, выделяя предикторы и целевые пе  
normalizer = Normalizer()  
res = normalizer.fit_transform(df)  
df_norm_n = pd.DataFrame(res, columns = df.columns)  
x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(  
    df_norm_n.loc[:, df_norm_n.columns != 'Модуль упругости при растяжении, ГГ  
    df[['Модуль упругости при растяжении, ГПа']],  
    test_size = 0.3,  
    random_state = 42)
```

```
In [191...]: df_norm_n.shape[0] - x_train_2.shape[0] - x_test_2.shape[0]  
x_train_2.head()
```

Out[191...]

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Темп, С
888	0.000962	0.611056	0.231597	0.035655	0.005740	(
368	0.001038	0.684067	0.313007	0.030022	0.008396	(
651	0.000687	0.578411	0.070642	0.022260	0.006792	(
652	0.000981	0.570442	0.401106	0.029510	0.005677	(
490	0.000681	0.588044	0.059991	0.028152	0.007266	(

```
In [192...]: y_train_2
```

Out[192...]

Модуль упругости при растяжении, ГПа

983	73.537663
416	69.492433
732	74.862468
733	79.828646
558	73.241464
...	...
120	74.519119
305	70.361924
954	68.673273
493	72.747541
113	72.625213

648 rows × 1 columns

In [193...]

y_train_2.shape

Out[193...]

(648, 1)

In [187...]

```
# Функция для сравнения результатов предсказаний с моделью, выдающей среднее значение
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 [194...]

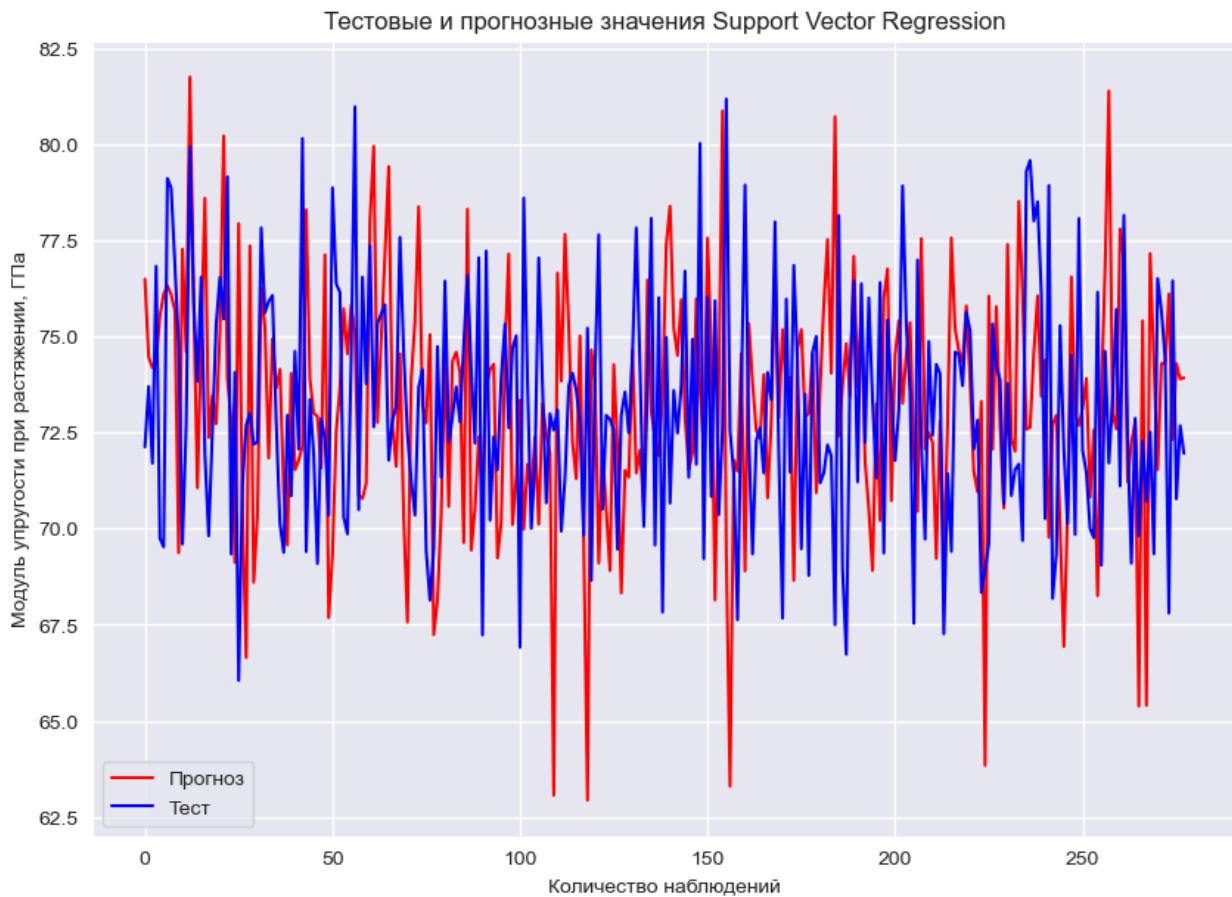
```
# Метод опорных векторов - 1
svr2 = make_pipeline(StandardScaler(), SVR(kernel = 'rbf', C = 500.0, epsilon = 0.1))
#обучаем модель
svr2.fit(x_train_2, np.ravel(y_train_2))
#вычисляем коэффициент детерминации
y_pred_svr2 = svr2.predict(x_test_2)
mae_svr2 = mean_absolute_error(y_pred_svr2, y_test_2)
mse_svr_elast2 = mean_squared_error(y_test_2,y_pred_svr2)
print('Support Vector Regression Results Train:')
print("Test score: {:.2f}".format(svr2.score(x_train_2, y_train_2))) # Скор для обучения
print('Support Vector Regression Results:')
print('SVR_MAE: ', round(mean_absolute_error(y_test_2, y_pred_svr2)))
print('SVR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_svr2)))
print('SVR_MSE: {:.2f}'.format(mse_svr_elast2))
print("SVR_RMSE: {:.2f}".format(np.sqrt(mse_svr_elast2)))
print("Test score: {:.2f}".format(svr2.score(x_test_2, y_test_2))) # Скор для проверки
```

```
Support Vector Regression Results Train:  
Test score: 0.90  
Support Vector Regression Results:  
SVR_MAE: 3  
SVR_MAPE: 0.05  
SVR_MSE: 18.61  
SVR_RMSE: 4.31  
Test score: -1.03
```

```
In [195...]: #Результаты модели, выдающей среднее значение  
mse_lin_elast2_mean = mean_squared_error(y_test_2, y_2_pred_mean)  
print("MAE for mean target: ", mean_absolute_error(y_test_2, y_2_pred_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: 73.13603017252109  
MSE for mean target: 5358.032787908149  
RMSE for mean target: 73.19858460317487
```

```
In [196...]: #SVR показывает значительно лучшие результаты по всем метрикам по сравнению с
```

```
In [199...]: plt.figure(figsize = (10, 7))  
plt.title("Тестовые и прогнозные значения Support Vector Regression")  
plt.plot(y_pred_svr2, label = "Прогноз", color = "red")  
plt.plot(y_test_2.values, label = "Тест", color = "blue")  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Модуль упругости при растяжении, ГПа")  
plt.legend()  
plt.grid(True);
```



```
In [200]: #построение модели и визуализация метода случайный лес
rfr2 = RandomForestRegressor(n_estimators = 15,max_depth = 7, random_state = 3)
rfr2.fit(x_train_2, y_train_2.values)
y2_pred_forest = rfr2.predict(x_test_2)
mae_rfr2 = mean_absolute_error(y2_pred_forest, y_test_2)
mse_rfr_elast2 = mean_squared_error(y_test_2,y2_pred_forest)
print('Random Forest Regressor Results Train:')
print("Test score: {:.2f}".format(rfr2.score(x_train_2, y_train_2))) # Скор для
print('Random Forest Regressor Results:')
print('RF_MAE: ', round(mean_absolute_error(y_test_2, y2_pred_forest)))
print('RF_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y2_pred_forest)))
print('RF_MSE: {:.2f}'.format(mse_rfr_elast2))
print("RF_RMSE: {:.2f}".format(np.sqrt(mse_rfr_elast2)))
print("Test score: {:.2f}".format(rfr2.score(x_test_2, y_test_2))) # Скор для
```

Random Forest Regressor Results Train:
Test score: 0.44
Random Forest Regressor Results:
RF_MAE: 3
RF_MAPE: 0.03
RF_MSE: 9.92
RF_RMSE: 3.15
Test score: -0.08

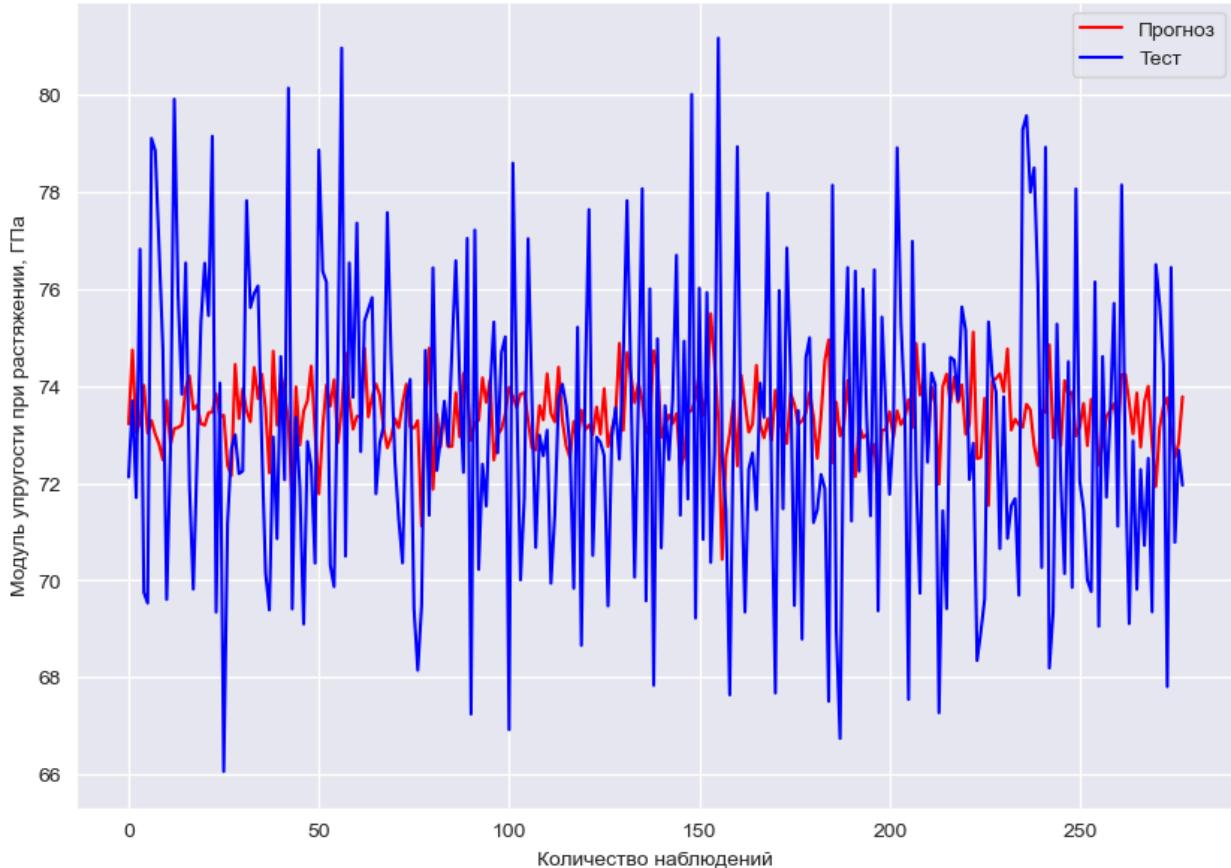
```
In [202]: plt.figure(figsize=(10, 7))
plt.title("Тестовые и прогнозные значения Random Forest Regressor")
```

```

plt.plot(y2_pred_forest, label = "Прогноз", color = "red")
plt.plot(y_test_2.values, label = "Тест", color = 'blue')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);

```

Тестовые и прогнозные значения Random Forest Regressor



In [203...]

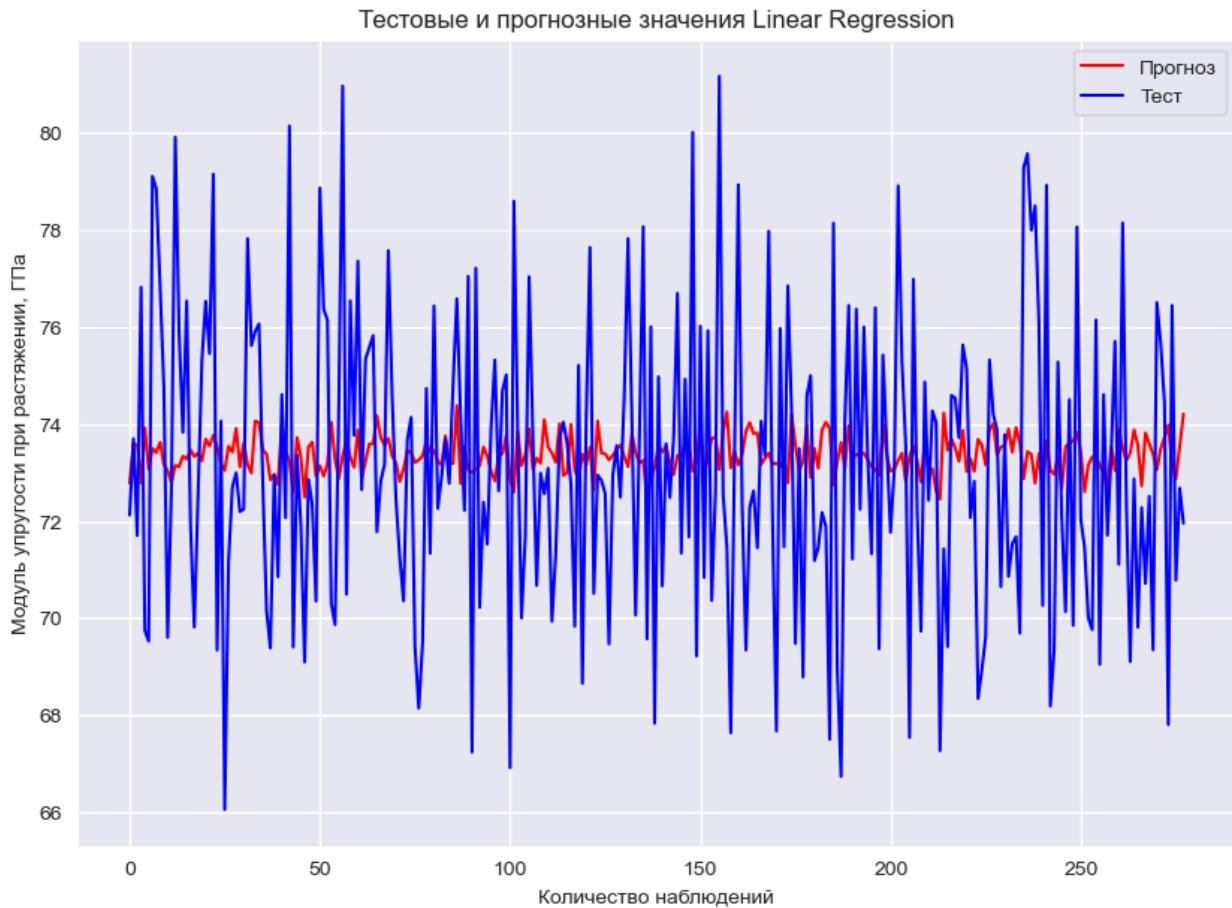
```

#построение модели и визуализация Линейной регрессии
lr2 = LinearRegression()
lr2.fit(x_train_2, y_train_2)
y_pred_lr2 = lr2.predict(x_test_2)
mae_lr2 = mean_absolute_error(y_pred_lr2, y_test_2)
mse_lin_elast2 = mean_squared_error(y_test_2, y_pred_lr2)
print('Linear Regression Results Train: ') # Скор для тренировочной выборки
print("Test score: {:.2f}".format(lr2.score(x_train_2, y_train_2)))
print('Linear Regression Results: ')
print('lr_MAE: ', round(mean_absolute_error(y_test_2, y_pred_lr2)))
print('lr_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_lr2)))
print('lr_MSE: {:.2f}'.format(mse_lin_elast2))
print("lr_RMSE: {:.2f}".format(np.sqrt(mse_lin_elast2)))
print("Test score: {:.2f}".format(lr2.score(x_test_2, y_test_2))) # Скор для т

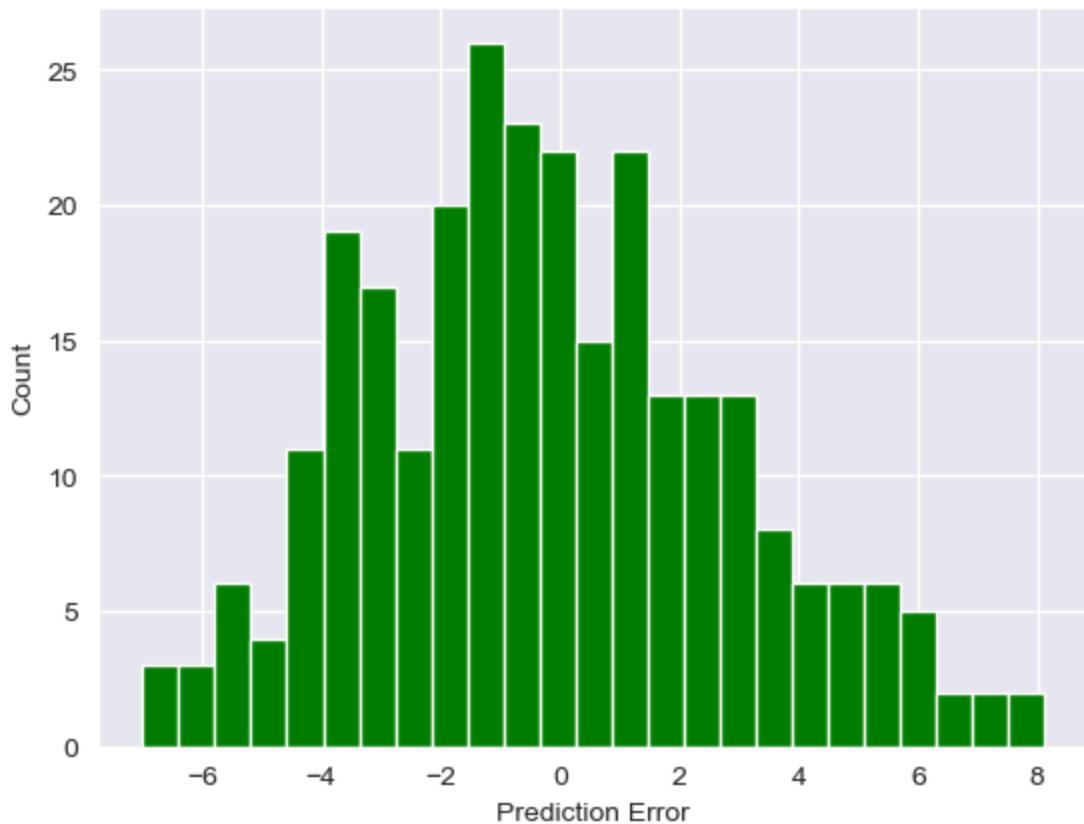
```

```
Linear Regression Results Train:  
Test score: 0.02  
Linear Regression Results:  
lr_MAE: 2  
lr_MAPE: 0.03  
lr_MSE: 9.23  
lr_RMSE: 3.04  
Test score: -0.01
```

```
In [205...]: plt.figure(figsize = (10, 7))  
plt.title("Тестовые и прогнозные значения Linear Regression")  
plt.plot(y_pred_lr2, label = "Прогноз", color = 'red')  
plt.plot(y_test_2.values, label = "Тест", color = 'blue')  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Модуль упругости при растяжении, ГПа")  
plt.legend()  
plt.grid(True);
```



```
In [206...]: error = y_test_2 - y_pred_lr2  
plt.hist(error, bins = 25, color = "g")  
plt.xlabel('Prediction Error')  
_ = plt.ylabel('Count')
```



```
In [207]: gbr2 = make_pipeline(StandardScaler(), GradientBoostingRegressor())
gbr2.fit(x_train_2, np.ravel(y_train_2))
y_pred_gbr2 = gbr2.predict(x_test_2)
mae_gbr2 = mean_absolute_error(y_pred_gbr2, y_test_2)
mse_gbr_elast2 = mean_squared_error(y_test_2,y_pred_gbr2)
print('Gradient Boosting Regressor Results Train:')
print("Test score: {:.2f}".format(gbr2.score(x_train_2, y_train_2))) # Скор для
print('Gradient Boosting Regressor Results:')
print('GBR_MAE: ', round(mean_absolute_error(y_test_2, y_pred_gbr2)))
print('GBR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_gbr2)))
print('GBR_MSE: {:.2f}'.format(mse_gbr_elast2))
print("GBR_RMSE: {:.2f}".format (np.sqrt(mse_gbr_elast2)))
print("Test score: {:.2f}".format(gbr2.score(x_test_2, y_test_2)))
```

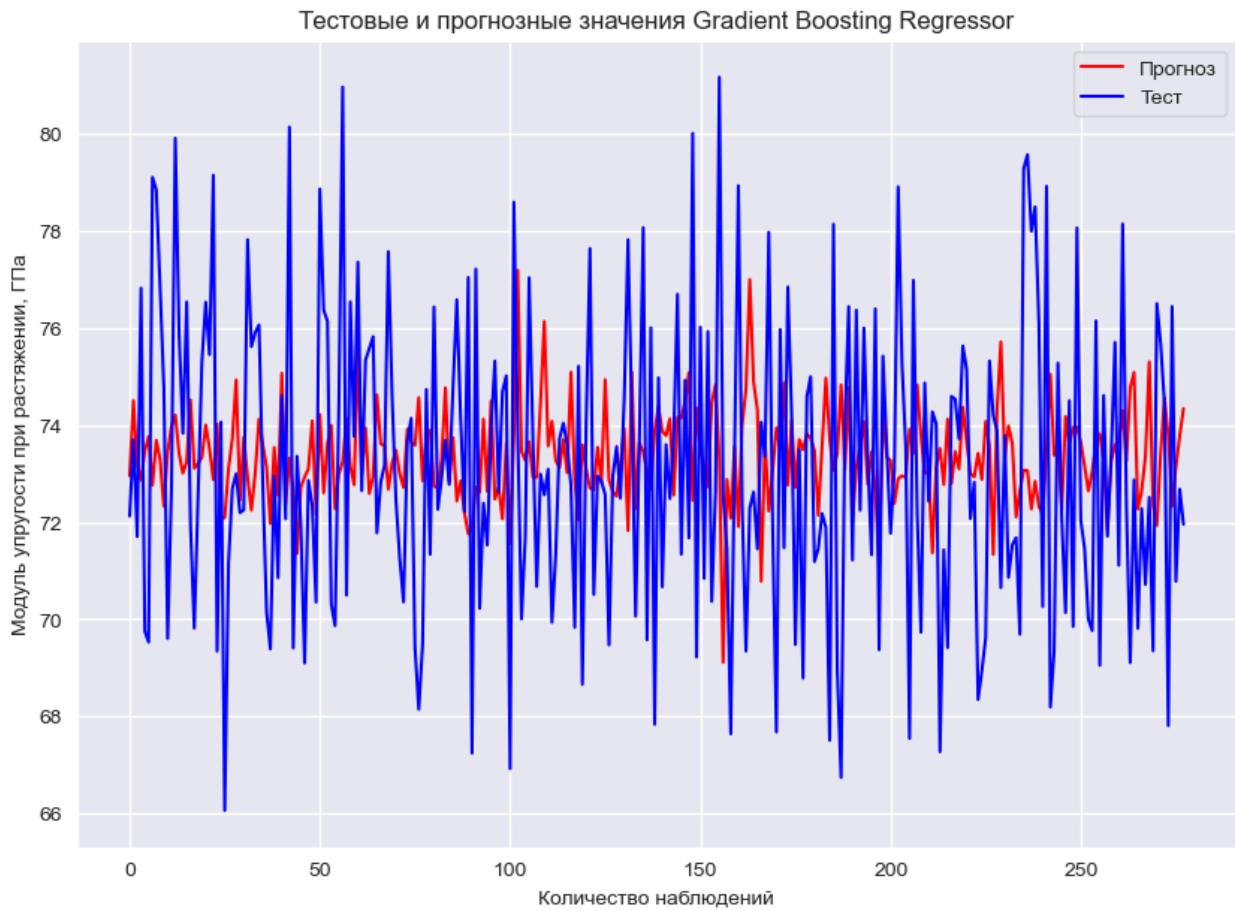
Gradient Boosting Regressor Results Train:
Test score: 0.54
Gradient Boosting Regressor Results:
GBR_MAE: 3
GBR_MAPE: 0.04
GBR_MSE: 10.41
GBR_RMSE: 3.23
Test score: -0.14

```
In [209]: plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Gradient Boosting Regressor")
plt.plot(y_pred_gbr2, label = "Прогноз", color = "red")
plt.plot(y_test_2.values, label = "Тест", color = "blue")
plt.xlabel("Количество наблюдений")
```

```

plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);

```



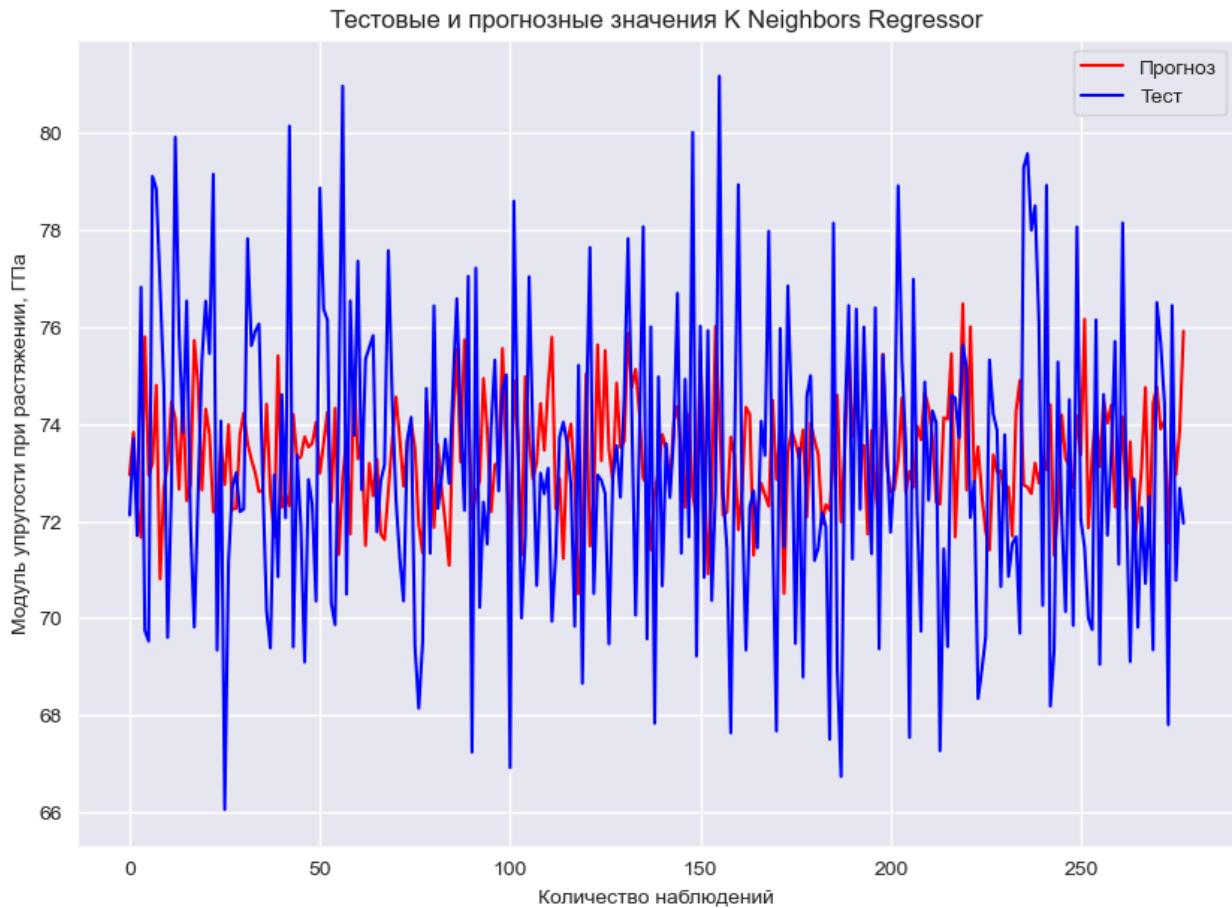
```

In [210...]: # Метод K ближайших соседей - K Neighbors Regressor - 5
knn2 = KNeighborsRegressor(n_neighbors=5)
knn2.fit(x_train_2, y_train_2)
y_pred_knn2 = knn2.predict(x_test_2)
mae_knr2 = mean_absolute_error(y_pred_knn2, y_test_2)
mse_knn_elast2 = mean_squared_error(y_test_2,y_pred_knn2)
print('K Neighbors Regressor Results Train:')
print("Test score: {:.2f}".format(knn2.score(x_train_2, y_train_2)))# Скор для
print('K Neighbors Regressor Results:')
print('KNN_MAE: ', round(mean_absolute_error(y_test_2, y_pred_knn2)))
print('KNN_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pre
print('KNN_MSE: {:.2f}'.format(mse_knn_elast2))
print("KNN_RMSE: {:.2f}".format(np.sqrt(mse_knn_elast2)))
print("Test score: {:.2f}".format(knn2.score(x_test_2, y_test_2)))

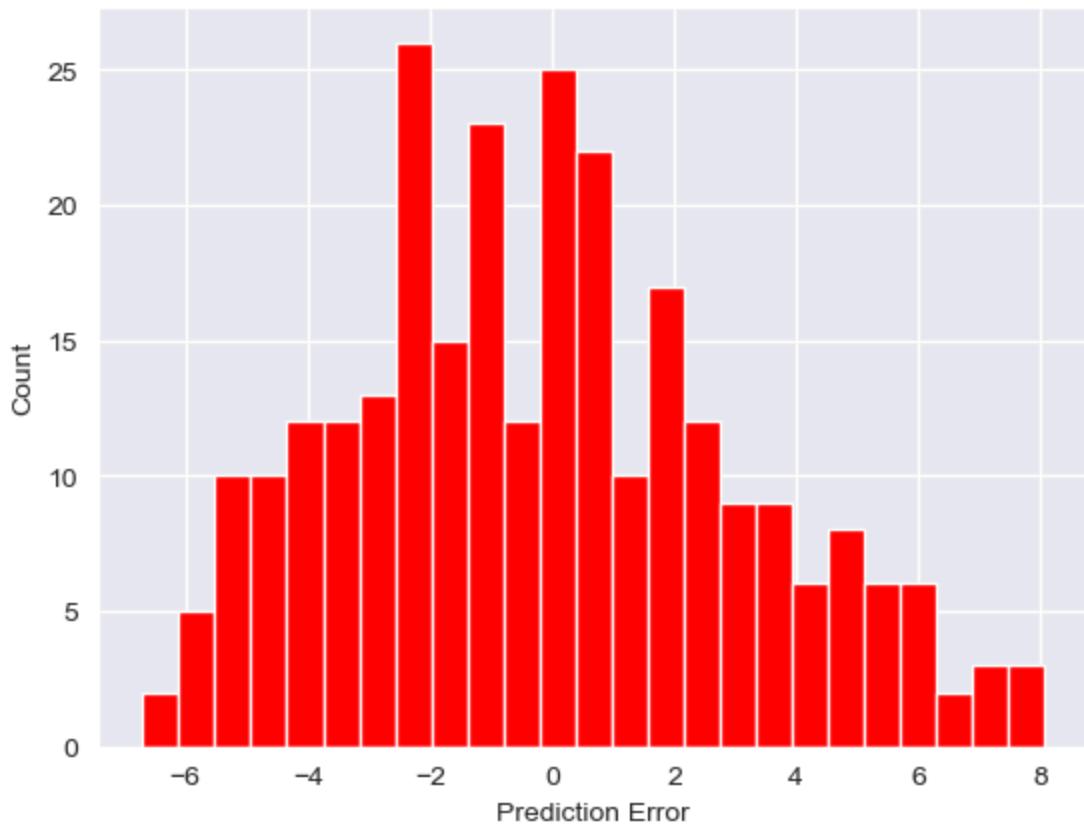
```

```
K Neighbors Regressor Results Train:  
Test score: 0.19  
K Neighbors Regressor Results:  
KNN_MAE: 3  
KNN_MAPE: 0.04  
KNN_MSE: 10.26  
KNN_RMSE: 3.20  
Test score: -0.12
```

```
In [211]: plt.figure(figsize = (10, 7))  
plt.title("Тестовые и прогнозные значения K Neighbors Regressor")  
plt.plot(y_pred_knn2, label = "Прогноз", color = 'red')  
plt.plot(y_test_2.values, label = "Тест", color = 'blue')  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Модуль упругости при растяжении, ГПа")  
plt.legend()  
plt.grid(True);
```



```
In [213]: error = y_test_2 - y_pred_knn2  
plt.hist(error, bins = 25, color = "r")  
plt.xlabel('Prediction Error')  
_ = plt.ylabel('Count')
```



In [214]: #Деревья 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 Regressor Results Train:')
print("Test score: {:.2f}".format(dtr2.score(x_train_2, y_train_2)))# Скор для
print('Decision Tree Regressor Results:')
print('DTR_MAE: ', round(mean_absolute_error(y_test_2, y_pred_dtr2)))
print('DTR_MSE: {:.2f}'.format(mse_dtr_elast2))
print("DTR_RMSE: {:.2f}".format (np.sqrt(mse_dtr_elast2)))
print('DTR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_dtr2)))
print("Test score: {:.2f}".format(dtr2.score(x_test_2, y_test_2)))
```

Decision Tree Regressor Results Train:

Test score: 1.00

Decision Tree Regressor Results:

DTR_MAE: 4

DTR_MSE: 20.06

DTR_RMSE: 4.48

DTR_MAPE: 0.05

Test score: -1.19

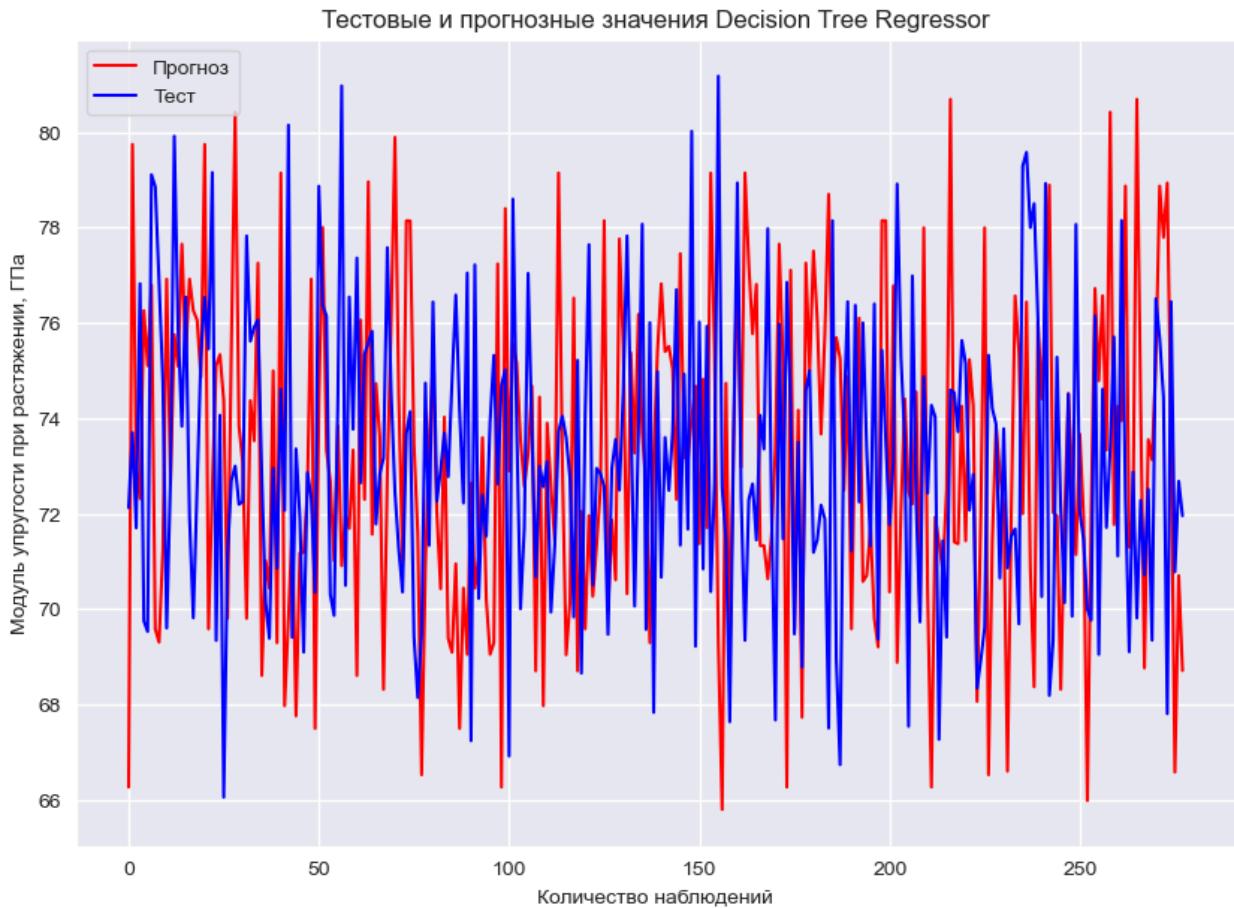
In [215]:

```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Decision Tree Regressor")
plt.plot(y_pred_dtr2, label = "Прогноз", color = 'red')
```

```

plt.plot(y_test_2.values, label = "Тест", color = 'blue')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);

```



In [216...]: # Стохастический градиентный спуск (SGD) - Stochastic Gradient Descent Regressor

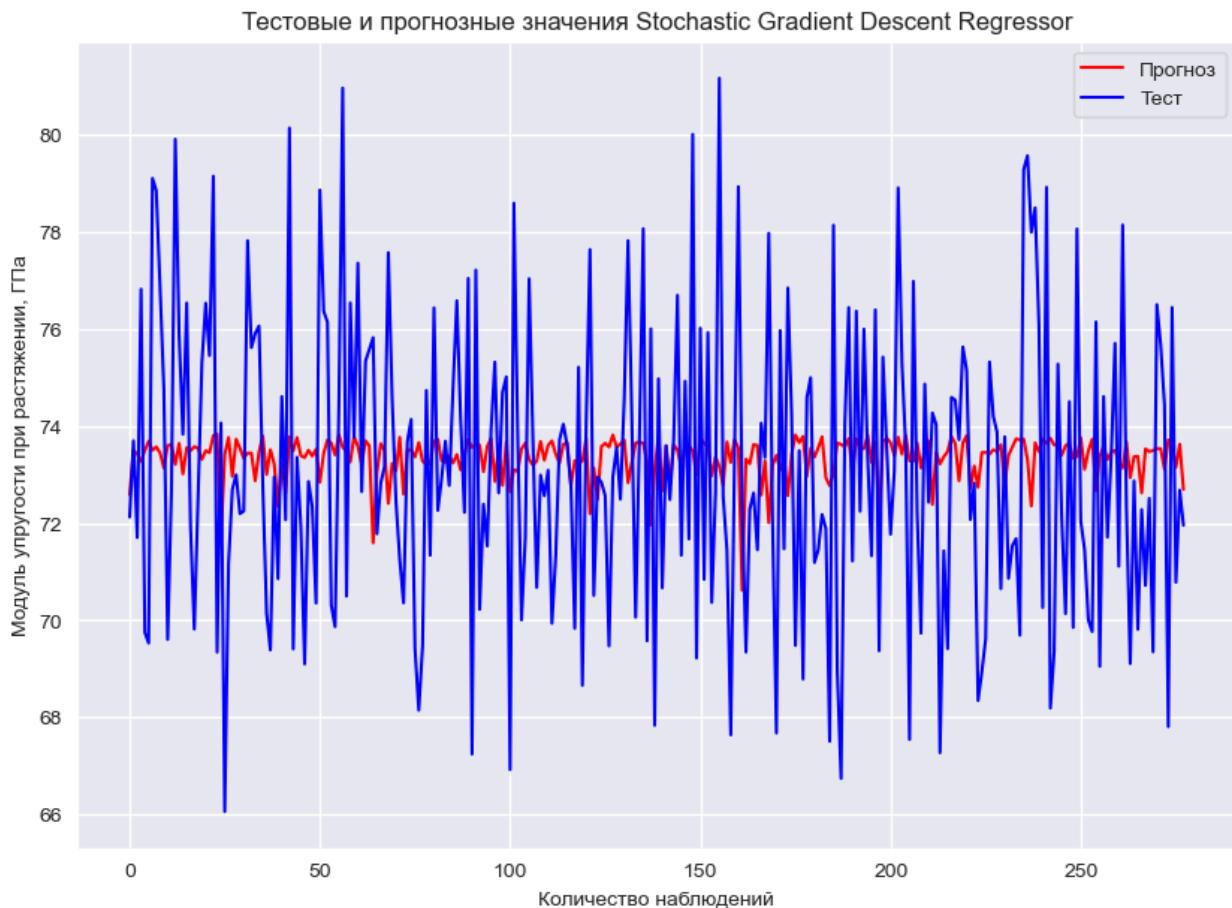
```

sdg2 = SGDRegressor()
sdg2.fit(x_train_2, y_train_2)
y_pred_sdg2 = sdg2.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("Test score: {:.2f}".format(sdg2.score(x_train_2, y_train_2)))# Скор для
print('Stochastic Gradient Descent Regressor Results:')
print('SGD_MAE: ', round(mean_absolute_error(y_test_2, y_pred_sdg2)))
print('SGD_MSE: {:.2f}'.format(mse_sdg_elast2))
print("SGD_RMSE: {:.2f}".format (np.sqrt(mse_sdg_elast2)))
print('SGD_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_2, y_pred_sdg2)))
print("Test score: {:.2f}".format(sdg2.score(x_test_2, y_test_2)))

```

```
Stochastic Gradient Descent Regressor Results Train:  
Test score: -0.02  
Stochastic Gradient Descent Regressor Results:  
SGD_MAE: 2  
SGD_MSE: 9.42  
SGD_RMSE: 3.07  
SGD_MAPE: 0.03  
Test score: -0.03
```

```
In [217...]: plt.figure(figsize = (10, 7))  
plt.title("Тестовые и прогнозные значения Stochastic Gradient Descent Regressor")  
plt.plot(y_pred_sdg2, label = "Прогноз", color = 'red')  
plt.plot(y_test_2.values, label = "Тест", color = 'blue')  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Модуль упругости при растяжении, ГПа")  
plt.legend()  
plt.grid(True);
```



```
In [218...]: # Многослойный перцептрон - Multi-layer Perceptron regressor - 8  
  
mlp2 = MLPRegressor(random_state = 1, max_iter = 500)  
mlp2.fit(x_train_2, y_train_2)  
y_pred_mlp2 = mlp2.predict(x_test_2)  
mae_mlp2 = mean_absolute_error(y_pred_mlp2, y_test_2)  
mse_mlp_elast2 = mean_squared_error(y_test_2,y_pred_mlp2)  
print('Multi-layer Perceptron regressor Results Train:')
```

```

print("Test score: {:.2f}".format(mlp2.score(x_train_2, y_train_2)))# Скор для
print('Multi-layer Perceptron regressor Results:')
print('SGD_MAE: ', round(mean_absolute_error(y_test_2, y_pred_mlp2)))
print('SGD_MAPE: {:.2f}'.format(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(mlp2.score(x_test_2, y_test_2)))

```

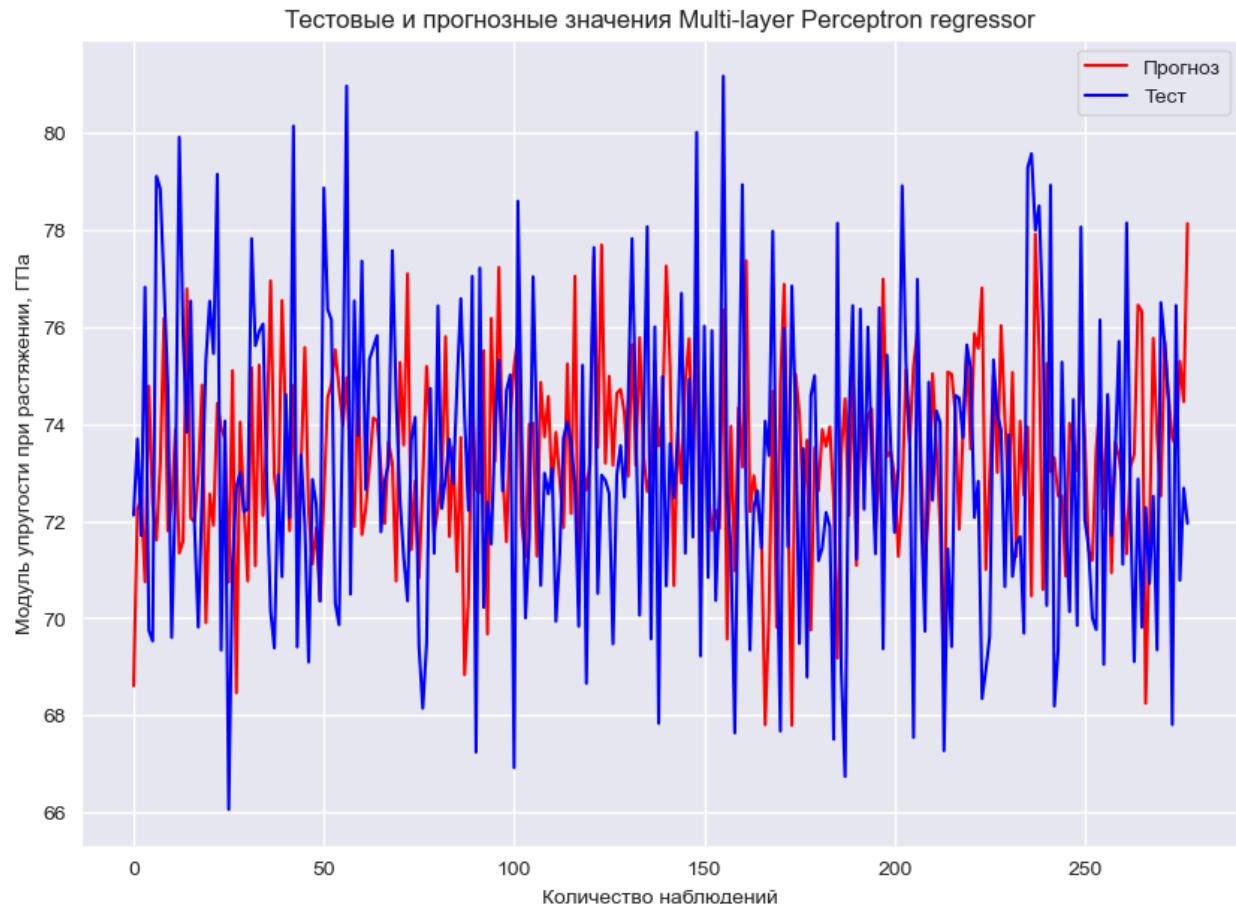
Multi-layer Perceptron regressor Results Train:
 Test score: -0.40
 Multi-layer Perceptron regressor Results:
 SGD_MAE: 3
 SGD_MAPE: 0.04
 SGD_MSE: 12.19
 SGD_RMSE: 3.49
 Test score: -0.33

In [219]:

```

plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Multi-layer Perceptron regressor")
plt.plot(y_pred_mlp2, label = "Прогноз", color = 'red')
plt.plot(y_test_2.values, label = "Тест", color = 'blue')
plt.xlabel("Количество наблюдений")
plt.ylabel("Модуль упругости при растяжении, ГПа")
plt.legend()
plt.grid(True);

```



```
In [223...]:  
from sklearn.linear_model import Lasso  
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error  
import numpy as np  
  
# Создаем модель Lasso  
clf2 = Lasso(alpha=0.1) # Используем прямой импорт класса  
  
# Обучение модели  
clf2.fit(x_train_2, y_train_2)  
  
# Предсказания  
y_pred_clf2 = clf2.predict(x_test_2)  
  
# Расчет метрик  
mae_clf2 = mean_absolute_error(y_test_2, y_pred_clf2)  
mse_clf2 = mean_squared_error(y_test_2, y_pred_clf2)  
rmse_clf2 = np.sqrt(mse_clf2)  
mape_clf2 = mean_absolute_percentage_error(y_test_2, y_pred_clf2)  
  
# Вывод результатов  
print('Lasso Regression Results Train:')print("Train score: {:.4f}".format(clf2.score(x_train_2, y_train_2)))  
print("Test score: {:.4f}".format(clf2.score(x_test_2, y_test_2)))  
  
print('\nLasso Regression Results:')print('Lasso_MAE: {:.4f}'.format(mae_clf2))  
print('Lasso_MAPE: {:.4f}'.format(mape_clf2))  
print('Lasso_MSE: {:.4f}'.format(mse_clf2))  
print('Lasso_RMSE: {:.4f}'.format(rmse_clf2))
```

Lasso Regression Results Train:

Train score: 0.0000

Test score: -0.0047

Lasso Regression Results:

Lasso_MAE: 2.4611

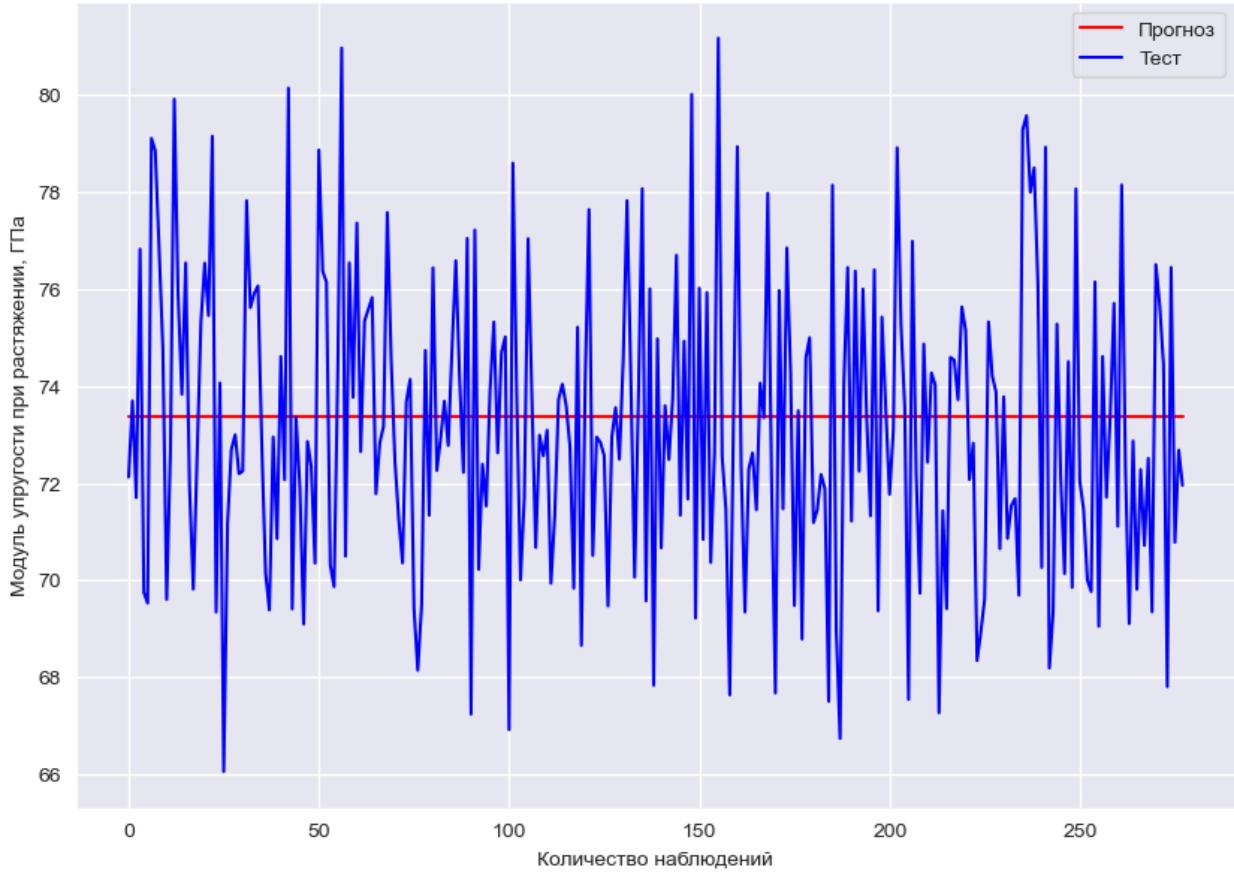
Lasso_MAPE: 0.0337

Lasso_MSE: 9.1969

Lasso_RMSE: 3.0326

```
In [224...]:  
plt.figure(figsize = (10, 7))  
plt.title("Тестовые и прогнозные значения Lasso regressor")  
plt.plot(y_pred_clf2, label = "Прогноз", color = 'red')  
plt.plot(y_test_2.values, label = "Тест", color = 'blue')  
plt.xlabel("Количество наблюдений")  
plt.ylabel("Модуль упругости при растяжении, ГПа")  
plt.legend()  
plt.grid(True);
```

Тестовые и прогнозные значения Lasso regressor



```
In [ ]: # Сравнение моделей по метрике MAE

mae_results = {
    '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]
}

mae_df2 = pd.DataFrame(mae_results)

mae_df2 = mae_df2.sort_values(by='MAE', ascending=True)

print("Сравнение моделей по метрике MAE:")
print(mae_df2)

from sklearn.model_selection import GridSearchCV
```

```

parametrs = {
    'n_estimators': [200, 300],
    'max_depth': [9, 15, 20],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['squared_error'],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

grid21 = GridSearchCV(
    estimator=rfr2,
    param_grid=parametrs,
    cv=10, # 10-кратная кросс-валидация
    scoring='neg_mean_absolute_error', # Используем отрицательную MAE
    verbose=1, # Добавляем вывод прогресса
    n_jobs=-1 # Используем все ядра процессора
)

# Обучаем модель
grid21.fit(x_train_2, y_train_2)

# Выводим лучшие параметры
print("\nЛучшие параметры для RandomForest:")
print(grid21.best_params_)

# Выводим лучшую оценку
print("Лучшая MAE:", -grid21.best_score_)

```

Сравнение моделей по метрике MAE:

	Регрессор	MAE
8	Lasso	2.461084
2	Linear Regression	2.462732
6	SGD	2.485237
1	RandomForest	2.544859
4	KNeighbors	2.608872
3	GradientBoosting	2.633531
7	MLP	2.796519
0	Support Vector	3.446165
5	DecisionTree	3.635743

Fitting 10 folds for each of 48 candidates, totalling 480 fits

Лучшие параметры для RandomForest:

```
{'criterion': 'squared_error', 'max_depth': 9, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
Лучшая MAE: 2.4522487631720375
```

In [234... mae_df

Out[234...]

	Перрессор	MAE
0	Support Vector	0.072198
1	RandomForest	0.010823
2	Linear Regression	0.011203
3	GradientBoosting	0.005985
4	KNeighbors	0.010085
5	DecisionTree	0.014322
6	SGD	0.073114
7	MLP	0.069747
8	Lasso	0.055824

In []: #Лучший результат показывает GradientBoosting с MAE = 0.005985

In [250...]: # Написать нейронную сеть, которая будет рекомендовать соотношение матрица-наполнитель

In [272...]

```

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

# Сформируем входы и выход для модели
tv = df['Соотношение матрица-наполнитель']
tr_v = df.drop('Соотношение матрица-наполнитель', axis=1)

# Разбиваем выборки 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
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Создаем модель нейронной сети
mlp = MLPRegressor(random_state=42)

# Определяем параметры для поиска
param_grid = {
    'hidden_layer_sizes': [(32,), (64, 32), (128, 64, 32)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001, 0.01],
}

```

```
'learning_rate': ['constant', 'adaptive'],
'max_iter': [200, 300, 500]
}

# Поиск оптимальных параметров
grid = GridSearchCV(
    estimator=mlp,
    param_grid=param_grid,
    cv=5,
    n_jobs=-1,
    verbose=1,
    scoring='neg_mean_absolute_error'
)

grid_result = grid.fit(x_train, y_train)

# Результаты
print("Best: %.4f using %s" % (grid_result.best_score_, grid_result.best_params_))

# Лучшая модель
best_model = grid_result.best_estimator_

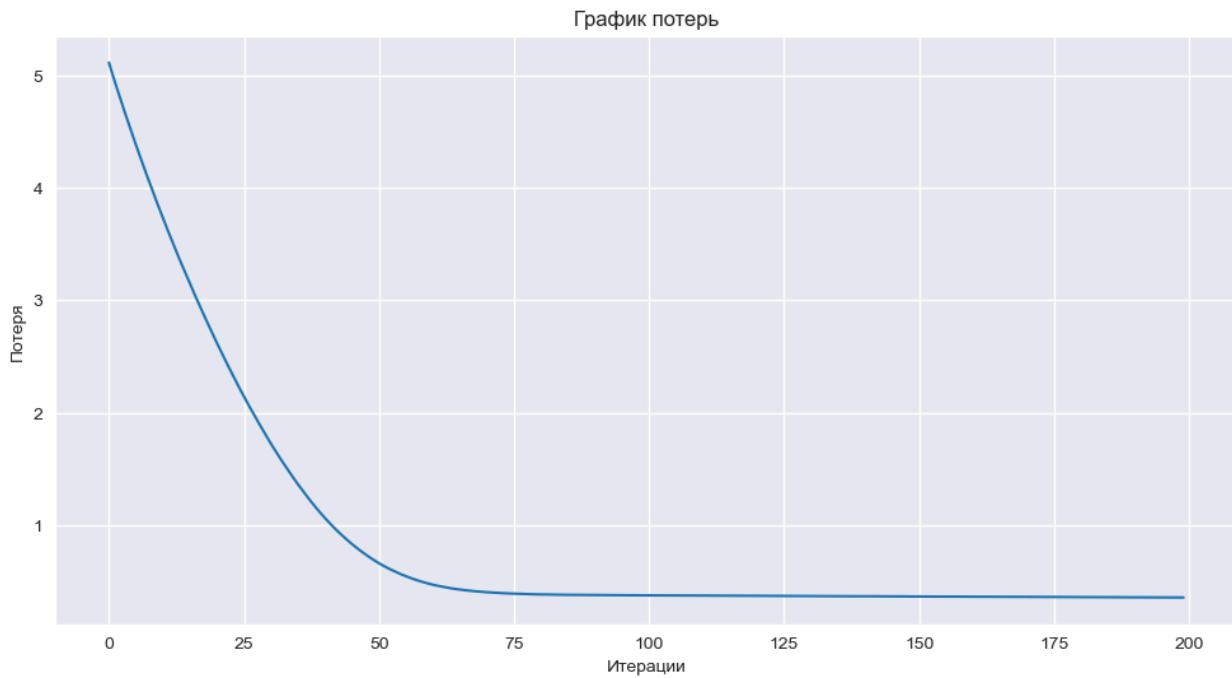
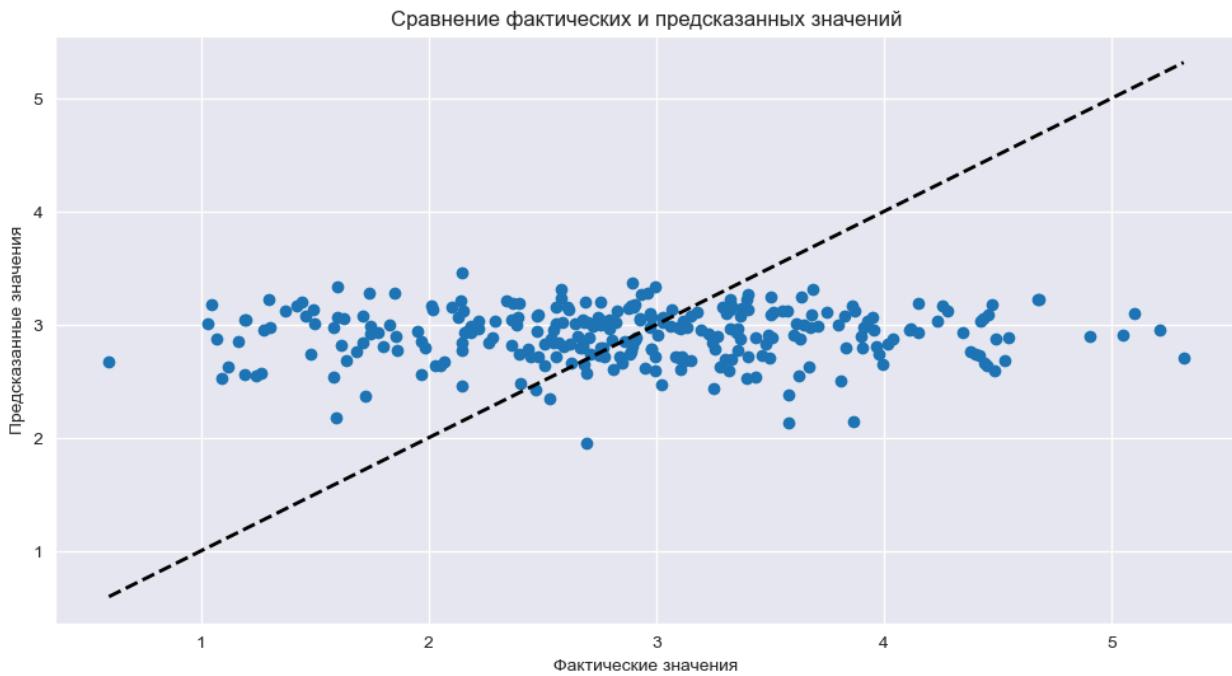
# Оценка модели
y_pred = best_model.predict(x_test)
mae = mean_absolute_error(y_test, y_pred)
print(f"\nMean Absolute Error: {mae:.4f}")

# Визуализация результатов
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel('Фактические значения')
plt.ylabel('Предсказанные значения')
plt.title('Сравнение фактических и предсказанных значений')
plt.show()

# График ошибок
plt.figure(figsize=(12, 6))
plt.plot(best_model.loss_curve_)
plt.title('График потерь')
plt.xlabel('Итерации')
plt.ylabel('Потеря')
plt.show()
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best: -0.7323 using {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes':
(32,), 'learning_rate': 'constant', 'max_iter': 200, 'solver': 'adam'}
```

Mean Absolute Error: 0.7397



```
In [277]: y_pred = best_model.predict(x_test)
mae = mean_absolute_error(y_test, y_pred)
print(f"\nMean Absolute Error: {mae:.4f}")
```

Mean Absolute Error: 0.7397

Лучшие найденные параметры

Best: -0.7323 using {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_scores': (32,), 'learning_rate': 'constant', 'max_iter': 200, 'solver': 'adam'}

Здесь:

Best score (-0.7323) - значение метрики (отрицательное МАЕ, так как GridSearchCV максимизирует значение)

Найденные оптимальные параметры:

Функция активации: tanh

Коэффициент регуляризации: 0.01

Архитектура сети: один скрытый слой с 32 нейронами

Скорость обучения: постоянная

Максимальное число итераций: 200

Оптимизатор: adam

Качество модели

Mean Absolute Error: 0.7397

Это означает, что в среднем предсказания модели отличаются от реальных значений на 0.7397 единицы измерения целевой переменной.

Анализ результатов

Качество модели:

MAE около 0.74 может быть приемлемым

Рекомендации по улучшению:

Проверить масштаб целевой переменной

Попробовать другие метрики оценки

Рассмотреть возможность увеличения числа нейронов

Проверить наличие переобучения

Заключение.

Итоговый анализ исследования Основные выводы Машинное обучение в сфере прогнозирования характеристик композитных материалов представляет собой комплексную задачу, которая требует высокого уровня программистских компетенций, глубокого понимания специфики композитных материалов, способности к анализу и интерпретации данных, а также профессионального подхода к построению прогностических моделей.

Методология исследования В процессе работы были реализованы следующие ключевые этапы. Прежде всего, проведен тщательный анализ данных, включающий использование реального датасета, детальное описание и исследование характеристик, построение различных типов визуализаций, а также разделение данных на обучающую и тестовую выборки.

Особое внимание было уделено инструментам анализа. В работе активно применялась библиотека Scikit-learn, использовались вспомогательные модули и осуществлялась автоматизация процессов обработки данных.

Алгоритмический инструментарий В ходе исследования были применены различные методы машинного обучения. Среди них линейная регрессия, градиентный бустинг, метод К ближайших соседей, деревья решений, стохастический градиентный спуск, многослойный перцептрон, лассо-регрессия, метод опорных векторов и случайный лес.

Оптимизация и оценка Процесс оптимизации включал несколько важных компонентов. Был применен метод GridSearch для поиска гиперпараметров, составлен подробный классификационный отчет, проведен тщательный анализ ключевых метрик качества, а также выполнена визуальная оценка результатов.

Практические результаты В результате проделанной работы удалось достичь значимых результатов. Было успешно обучено нейронное сеть, разработано функциональное пользовательское приложение, создана эффективная система прогнозирования параметров и сформирована наглядная отчетность.

Перспективные направления Для дальнейшего развития проекта предлагается несколько направлений работы. Необходимо углубленно изучить свойства композитных материалов, расширить статистическую базу данных, исследовать альтернативные подходы к анализу, сформировать междисциплинарную команду специалистов, совершенствовать существующие решения и обеспечить качественную техническую

поддержку.

Таким образом, проведенное исследование демонстрирует значительный потенциал применения методов машинного обучения для прогнозирования характеристик композитных материалов, открывая новые возможности для оптимизации процессов их разработки и производства.