```
# Импорт библиотек
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)


# загрузка датасета 1
data1 = pd.read_excel('X_bp.xlsx')
data1
```

	Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Темпе вс
0	0.0	1.857143	2030.000000	738.736842	30.000000	22.267857	100.
1	1.0	1.857143	2030.000000	738.736842	50.000000	23.750000	284.
2	2.0	1.857143	2030.000000	738.736842	49.900000	33.000000	284.
3	3.0	1.857143	2030.000000	738.736842	129.000000	21.250000	300.
4	4.0	2.771331	2030.000000	753.000000	111.860000	22.267857	284.
...	...	...	...	...	...	...	
1018	1018.0	2.271346	1952.087902	912.855545	86.992183	20.123249	324.
1019	1019.0	3.444022	2050.089171	444.732634	145.981978	19.599769	254.
1020	1020.0	3.280604	1972.372865	416.836524	110.533477	23.957502	248.
1021	1021.0	3.705351	2066.799773	741.475517	141.397963	19.246945	275.

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1023 entries, 0 to 1022
Data columns (total 11 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Unnamed: 0                            1023 non-null   float64
 1   Соотношение матрица-наполнитель       1023 non-null   float64
 2   Плотность, кг/м3                      1023 non-null   float64
 3   модуль упругости, ГПа                 1023 non-null   float64
 4   Количество отвердителя, м.%           1023 non-null   float64
 5   Содержание эпоксидных групп,%_2       1023 non-null   float64
 6   Температура вспышки, С_2              1023 non-null   float64
 7   Поверхностная плотность, г/м2         1023 non-null   float64
```

```
 8   Модуль упругости при растяжении, ГПа  1023 non-null   float64
 9   Прочность при растяжении, МПа         1023 non-null   float64
 10  Потребление смолы, г/м2               1023 non-null   float64
dtypes: float64(11)
memory usage: 88.0 KB
```

```
# загрузка датасета 2
data2 = pd.read_excel('X_nup.xlsx')
data2
```

	Unnamed: 0	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	0.0	0.0	4.000000	57.000000
1	1.0	0.0	4.000000	60.000000
2	2.0	0.0	4.000000	70.000000
3	3.0	0.0	5.000000	47.000000
4	4.0	0.0	5.000000	57.000000
...	...	...	...	...
1035	1035.0	90.0	8.088111	47.759177
1036	1036.0	90.0	7.619138	66.931932
1037	1037.0	90.0	9.800926	72.858286
1038	1038.0	90.0	10.079859	65.519479
1039	1039.0	90.0	9.021043	66.920143

1040 rows × 4 columns

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1040 entries, 0 to 1039
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          1040 non-null   float64
 1   Угол нашивки, град  1040 non-null   float64
 2   Шаг нашивки         1040 non-null   float64
 3   Плотность нашивки   1040 non-null   float64
dtypes: float64(4)
memory usage: 32.6 KB
```

```
# объединение датасетов по индексу, тип объединения inner
df = pd.merge(data1, data2, how = 'inner', left_index = True, right_index = True)
df.head()
```

	Unnamed: 0_x	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температу вспыш
0	0.0	1.857143	2030.0	738.736842	30.00	22.267857	100.000
1	1.0	1.857143	2030.0	738.736842	50.00	23.750000	284.615

```
# в инфо видим лишние колонки
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1023 entries, 0 to 1022
Data columns (total 15 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   Unnamed: 0_x                           1023 non-null   float64
 1   Соотношение матрица-наполнитель        1023 non-null   float64
 2   Плотность, кг/м3                       1023 non-null   float64
 3   модуль упругости, ГПа                  1023 non-null   float64
 4   Количество отвердителя, м.%            1023 non-null   float64
 5   Содержание эпоксидных групп,%_2        1023 non-null   float64
 6   Температура вспышки, С_2               1023 non-null   float64
 7   Поверхностная плотность, г/м2          1023 non-null   float64
 8   Модуль упругости при растяжении, ГПа   1023 non-null   float64
 9   Прочность при растяжении, МПа          1023 non-null   float64
 10  Потребление смолы, г/м2                1023 non-null   float64
 11  Unnamed: 0_y                           1023 non-null   float64
 12  Угол нашивки, град                     1023 non-null   float64
 13  Шаг нашивки                            1023 non-null   float64
 14  Плотность нашивки                      1023 non-null   float64
dtypes: float64(15)
memory usage: 127.9 KB
```

```
# дропаем их
df1 = df.drop(df.columns[[0,11]], axis=1)
df1.head()
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверх плотно
0	1.857143	2030.0	738.736842	30.00	22.267857	100.000000	
1	1.857143	2030.0	738.736842	50.00	23.750000	284.615385	
2	1.857143	2030.0	738.736842	49.90	33.000000	284.615385	
3	1.857143	2030.0	738.736842	129.00	21.250000	300.000000	

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 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   float64
 11  Шаг нашивки                           1023 non-null   float64
 12  Плотность нашивки                     1023 non-null   float64
dtypes: float64(13)
memory usage: 111.9 KB
```

```
# проверяем пропуски
df1.isnull().sum()
```

```
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, град                      0
Шаг нашивки                             0
Плотность нашивки                       0
dtype: int64
```

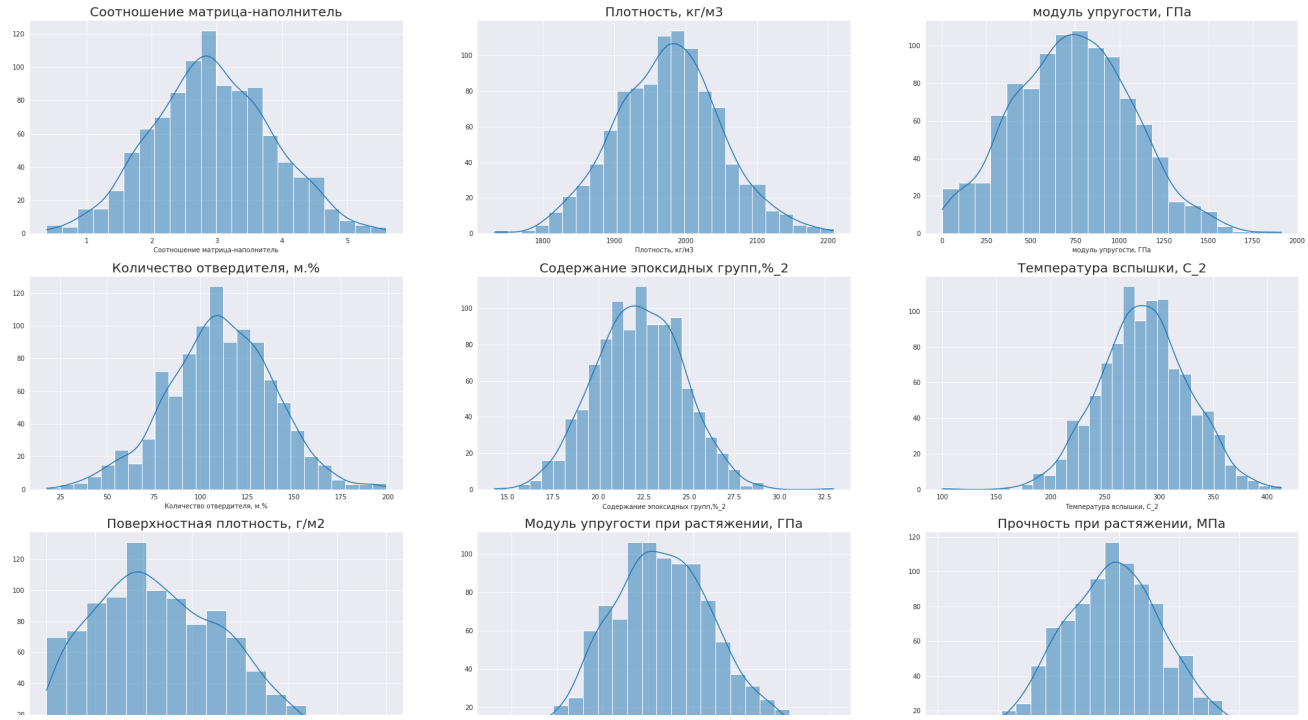
```
# проверяем дубли
df.duplicated().sum()
```

```
0
```

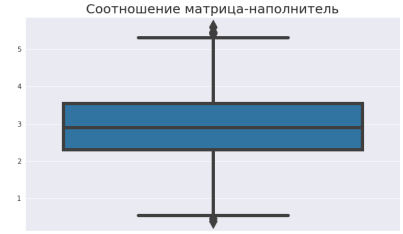
```
# смотрим описание (для каждой колонки получаем среднее, медианное значение и т.д.)
df1.describe()
```

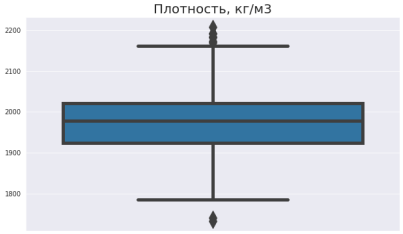
	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2
count	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000

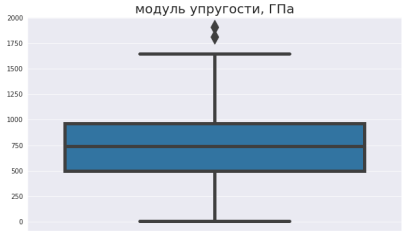
```
# гистрограмма распределения каждой переменной
plt.figure(figsize=(35,35))
a = 5
b = 3
c = 1

for col in df1.columns:
    plt.subplot(a, b, c)
    sns.histplot(data = df1[col], kde=True)
    plt.ylabel(None)
    plt.title(col, size = 20)
    c+=1
```

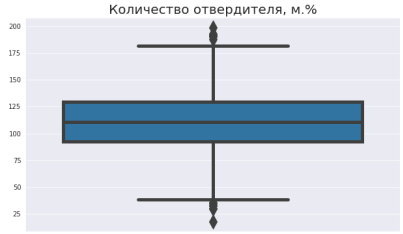


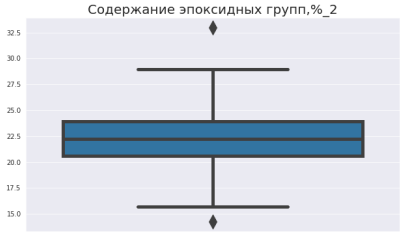

```
# диаграмма ящика с усами по каждой переменной
plt.figure(figsize=(35,35))
a = 5
b = 3
c = 1

for col in df1.columns:
    plt.subplot(a, b, c)
    sns.boxplot(data = df1, y=df1[col], fliersize=15, linewidth=5)
    plt.ylabel(None)
    plt.title(col, size = 20)
    c+=1
```

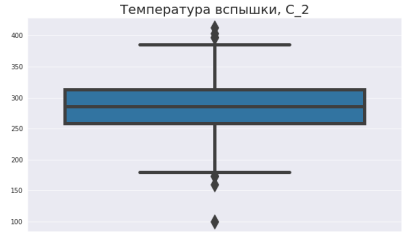

Соотношение матрица-наполнитель
5
4
3
2
1

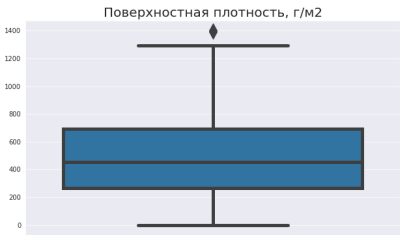

Плотность, кг/м3
2200
2100
2000
1900
1800

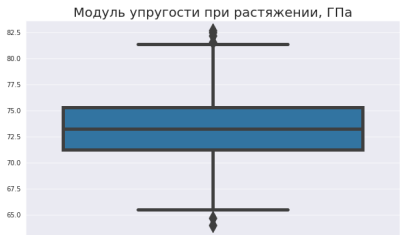

модуль упругости, ГПа
2000
1750
1500
1250
1000
750
500
250
0

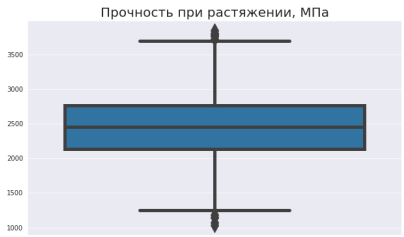

Количество отвердителя, м.%
200
175
150
125
100
75
50
25

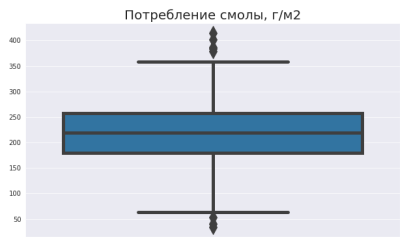

Содержание эпоксидных групп,%_2
32.5
30.0
27.5
25.0
22.5
20.0
17.5
15.0


Температура вспышки, С_2
400
350
300
250
200
150
100

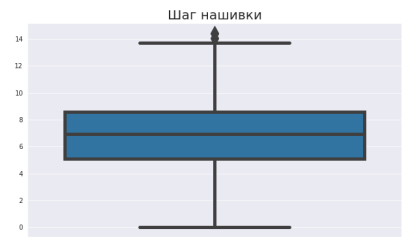

Поверхностная плотность, г/м2
1400
1200
1000
800
600
400
200
0

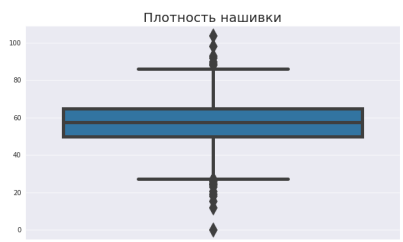

Модуль упругости при растяжении, ГПа
82.5
80.0
77.5
75.0
72.5
70.0
67.5
65.0


Прочность при растяжении, МПа
3500
3000
2500
2000
1500
1000


Потребление смолы, г/м2
400
350
300
250
200
150
100
50

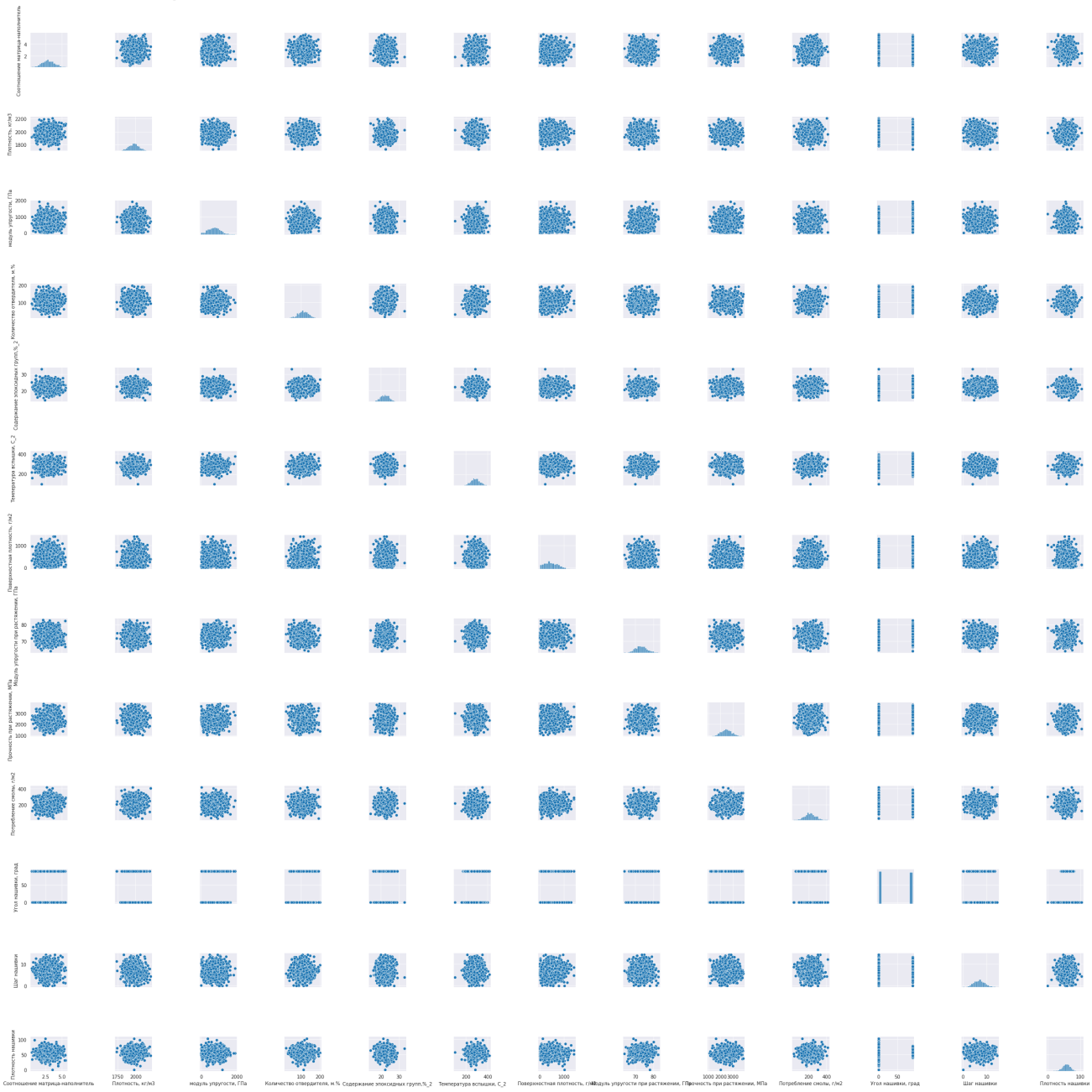

Угол нашивки, град
80
60
40
20
0


Шаг нашивки
14
12
10
8
6
4
2
0

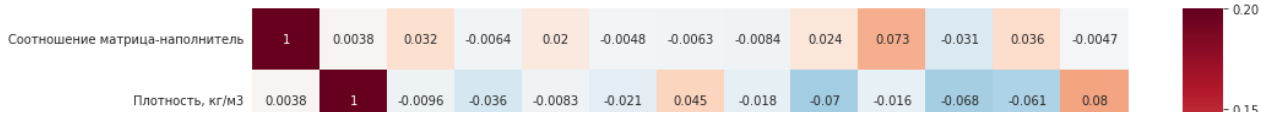

Плотность нашивки
100
80
60
40
20
0

```
# попарные графики рассеяния точек
sns.pairplot (df1)
```

<seaborn.axisgrid.PairGrid at 0x7f58e7228990>



```
# смотрим корреляцию между переменными
plt.figure(figsize=(16,12))
sns.heatmap(
    df1.corr(),
    cmap='RdBu_r',
    annot=True,
    vmin=-0.2, vmax=0.2)
plt.xticks(rotation=50, ha='right');
```




```
# ищем выбросы в разных колонках
for col in df1.columns:
  q3,q1 = np.percentile(df1.loc[:,col],[75,25])
  iqr = q3 - q1

  upper = q3 + 1.5 * iqr
  lower = q1 - 1.5 * iqr

  df1.loc[df1[col] < lower,col] = np.nan
  df1.loc[df1[col] > upper,col] = np.nan

df1.isnull().sum()
```

```
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, град                       0
Шаг нашивки                              4
Плотность нашивки                       21
dtype: int64
```

```
# дропаем их
df1 = df1.dropna(axis=0)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 936 entries, 1 to 1022
Data columns (total 13 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   Соотношение матрица-наполнитель        936 non-null    float64
 1   Плотность, кг/м3                       936 non-null    float64
 2   модуль упругости, ГПа                  936 non-null    float64
 3   Количество отвердителя, м.%            936 non-null    float64
 4   Содержание эпоксидных групп,%_2        936 non-null    float64
 5   Температура вспышки, С_2               936 non-null    float64
 6   Поверхностная плотность, г/м2          936 non-null    float64
 7   Модуль упругости при растяжении, ГПа   936 non-null    float64
 8   Прочность при растяжении, МПа          936 non-null    float64
 9   Потребление смолы, г/м2                936 non-null    float64
 10  Угол нашивки, град                     936 non-null    float64
 11  Шаг нашивки                            936 non-null    float64
 12  Плотность нашивки                      936 non-null    float64
dtypes: float64(13)
memory usage: 102.4 KB
```

```
# нормализация данных
mms = MinMaxScaler()
df2 = pd.DataFrame(mms.fit_transform(df1), columns = df1.columns, index=df1.index)
df2.describe()
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	По пл
count	936.000000	936.000000	936.000000	936.000000	936.000000	936.000000	
mean	0.498933	0.502695	0.446764	0.504664	0.491216	0.516059	
std	0.187489	0.187779	0.199583	0.188865	0.180620	0.190624	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.372274	0.368517	0.301243	0.376190	0.367716	0.386128	
50%	0.494538	0.511229	0.447061	0.506040	0.489382	0.515980	
75%	0.629204	0.624999	0.580446	0.637978	0.623410	0.646450	

Переходим к построению моделей для прогноза модуля упругости при растяжении и прочности при растяжении

```
#добавляем библиотеки
from sklearn.model_selection import train_test_split  # При построении модели необходимо З
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV # При построении моде
# GridSearchCV исчерпывающе рассматриваются все комбинации параметров, а RandomizedSearchC

from sklearn.linear_model import LinearRegression   # линейная регрессия
from sklearn.neighbors import KNeighborsRegressor  # метод ближайших соседей
from sklearn.ensemble import RandomForestRegressor # случайный лес
from sklearn.neural_network import MLPRegressor # многослойный персепртрон, нейросеть с уч

from sklearn.metrics import mean_absolute_percentage_error # метрика для проверки


# делаем разбивку данных для обучения
x_uprugost = df2.drop(['Модуль упругости при растяжении, ГПа'], axis=1)
x_prochnost = df2.drop(['Прочность при растяжении, МПа'], axis=1)
y_uprugost = df2[['Модуль упругости при растяжении, ГПа']]
y_prochnost = df2[['Прочность при растяжении, МПа']]

X_train_uprugost, X_test_uprugost, y_train_uprugost, y_test_uprugost = train_test_split(x_
X_train_prochnost, X_test_prochnost, y_train_prochnost, y_test_prochnost = train_test_spli


# линейная регрессия (упругость)
lr = LinearRegression()
lr_parameters = {
    'fit_intercept' : ['True', 'False']
```

```
}       # Когда fit_intercept=True линия наилучшего соответствия может «соответствовать» о
GSCV_lr_uprugost = GridSearchCV(lr, lr_parameters, n_jobs=-1, cv=10) # Количество заданий
GSCV_lr_uprugost.fit(X_train_uprugost, y_train_uprugost)
GSCV_lr_uprugost.best_params_
```

```
     {'fit_intercept': 'True'}
```

```
lr_uprugost = GSCV_lr_uprugost.best_estimator_
lr_uprugost_result = pd.DataFrame({
   'Model': 'LR_uprugost',
   'MAPE': mean_absolute_percentage_error(y_test_uprugost, lr_uprugost.predict(X_test_upru
   'R2': lr_uprugost.score(X_test_uprugost, y_test_uprugost).round(3)
}, index=['1'])
```

```
# метод ближайших соседей (упругость)
kn = KNeighborsRegressor()
kn_parameters = {'n_neighbors' : range(1, 301, 2),  # кол-во соседей
          'weights' : ['uniform', 'distance'],   # веса, как однородные имеющие один вес,
          'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute']   # все алгоритмы
          }
GSCV_kn_uprugost = GridSearchCV(kn, kn_parameters, n_jobs=-1, cv=10)
GSCV_kn_uprugost.fit(X_train_uprugost, y_train_uprugost)
GSCV_kn_uprugost.best_params_
```

```
     {'algorithm': 'auto', 'n_neighbors': 183, 'weights': 'uniform'}
```

```
kn_uprugost = GSCV_kn_uprugost.best_estimator_
kn_uprugost_result = pd.DataFrame({
   'Model': 'KN_uprugost',
   'MAPE': mean_absolute_percentage_error(y_test_uprugost, kn_uprugost.predict(X_test_upru
   'R2': kn_uprugost.score(X_test_uprugost, y_test_uprugost).round(3)
}, index=['2'])
```

```
# случайный лес (упругость)
rf = RandomForestRegressor()
rf_parameters = {
    'n_estimators' : range(10, 1000, 10),
    'criterion' : ['squared_error', 'absolute_error', 'poisson'],
    'max_depth' : range(1, 7),
    'min_samples_split' : range(20, 50, 5),
    'min_samples_leaf' : range(2, 8),
    'bootstrap' : ['True', 'False']
}
RSCV_rf_uprugost = RandomizedSearchCV(rf, rf_parameters, n_jobs=-1, cv=10, verbose=4)
RSCV_rf_uprugost.fit(X_train_uprugost, np.ravel(y_train_uprugost))
RSCV_rf_uprugost.best_params_
```

```
     Fitting 10 folds for each of 10 candidates, totalling 100 fits
     {'n_estimators': 920,
      'min_samples_split': 30,
      'min_samples_leaf': 3,
      'max_depth': 1,
```

```
      'criterion': 'absolute_error',
      'bootstrap': 'False'}
```

```
rf_uprugost = RSCV_rf_uprugost.best_estimator_
rf_uprugost_result = pd.DataFrame({
   'Model': 'RF_uprugost',
   'MAPE': mean_absolute_percentage_error(y_test_uprugost, rf_uprugost.predict(X_test_upru
   'R2': rf_uprugost.score(X_test_uprugost, y_test_uprugost).round(3)
}, index=["3"])
```

```
# собираем все метрики для оценки эффективности моделей
metrics_models_uprugost = pd.DataFrame()
metrics_models_uprugost = pd.concat([metrics_models_uprugost, lr_uprugost_result, kn_uprug
metrics_models_uprugost
```

	Model	MAPE	R2
1	LR_uprugost	0.453354	-0.005
2	KN_uprugost	0.449671	-0.003
3	RF_uprugost	0.445775	-0.005

Средняя абсолютная процентная ошибка между значениями, предсказанными моделью, и фактическими значениями составляет 45%. А коэфициент детерминации ~0 , что говорит о том, что данные прогнозируемые моделями равны усреднённым значениям. Такие низкие показатели работы моделей обусловлены слабой корреляцией данных.

```
# линейная регрессия (прочность)
GSCV_lr_prochnost = GridSearchCV(lr, lr_parameters, n_jobs=-1, cv=10)
GSCV_lr_prochnost.fit(X_train_prochnost, y_train_prochnost)
GSCV_lr_prochnost.best_params_
```

```
     {'fit_intercept': 'True'}
```

```
lr_prochnost = GSCV_lr_prochnost.best_estimator_
lr_prochnost_result = pd.DataFrame({
   'Model': 'LR_prochnost',
   'MAPE': mean_absolute_percentage_error(y_test_prochnost, lr_prochnost.predict(X_test_pr
   'R2': lr_prochnost.score(X_test_prochnost, y_test_prochnost).round(3)
}, index=['1'])
```

```
# метод ближайших соседей (прочность)
GSCV_kn_prochnost = GridSearchCV(kn, kn_parameters, n_jobs=-1, cv=10)
GSCV_kn_prochnost.fit(X_train_prochnost, y_train_prochnost)
GSCV_kn_prochnost.best_params_
```

```
     {'algorithm': 'auto', 'n_neighbors': 243, 'weights': 'uniform'}
```

```
kn_prochnost = GSCV_kn_prochnost.best_estimator_
kn_prochnost_result = pd.DataFrame({
   'Model': 'KN_prochnost',
   'MAPE': mean_absolute_percentage_error(y_test_prochnost, kn_prochnost.predict(X_test_pr
   'R2': kn_prochnost.score(X_test_prochnost, y_test_prochnost).round(3)
}, index=['2'])
```

```
# случайный лес (прочность)
RSCV_rf_prochnost = RandomizedSearchCV(rf, rf_parameters, n_jobs=-1, cv=10, verbose=4)
RSCV_rf_prochnost.fit(X_train_prochnost, y_train_prochnost)
RSCV_rf_prochnost.best_params_
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:926: DataC
  self.best_estimator_.fit(X, y, **fit_params)
{'n_estimators': 470,
 'min_samples_split': 35,
 'min_samples_leaf': 7,
 'max_depth': 1,
 'criterion': 'absolute_error',
 'bootstrap': 'True'}
```

```
rf_prochnost = RSCV_rf_prochnost.best_estimator_
rf_prochnost_result = pd.DataFrame({
   'Model': 'RF_prochnost',
   'MAPE': mean_absolute_percentage_error(y_test_prochnost, rf_prochnost.predict(X_test_pr
   'R2': rf_prochnost.score(X_test_prochnost, y_test_prochnost).round(3)
}, index=['3'])
```

```
metrics_models_prochnost = pd.DataFrame()
metrics_models_prochnost = pd.concat([metrics_models_prochnost, lr_prochnost_result, kn_pr
metrics_models_prochnost
```

	Model	MAPE	R2
1	LR_prochnost	0.473412	-0.047
2	KN_prochnost	0.472405	-0.028
3	RF_prochnost	0.469157	-0.045

Показатели модели прочности не лучше чем упругости.

```
# импортируем библиотеки для построения нейросети (Написать нейронную сеть, которая будет
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, LeakyReLU, Activation, Drop
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
# готовим слой предварительной обработки, который нормализует непрерывные функции.
normalizer = tf.keras.layers.Normalization(axis=-1)


# создаем обучающую и тестовую выборку
x_mn = df1.drop(['Соотношение матрица-наполнитель'], axis=1)
y_mn = df1[['Соотношение матрица-наполнитель']]

X_train_mn, X_test_mn, y_train_mn, y_test_mn = train_test_split(x_mn, y_mn, test_size=0.3,


# на обучающей выборке нормализуем данные
X_train_mn_norm = normalizer.adapt(np.array(X_train_mn))


# формируем слои нейросети
model_mn = Sequential(X_train_mn_norm)   # создаем последовательную модель

model_mn.add(Dense(128)) # добавляем полносвязный слой
model_mn.add(BatchNormalization()) # нормализация входных данных
model_mn.add(LeakyReLU()) # расширенный активационный слой для создания небольшого градиен
model_mn.add(Dense(128, activation='selu')) #  самонормализация нейронной сети. При умноже
model_mn.add(BatchNormalization())
model_mn.add(Dense(64, activation='selu'))
model_mn.add(BatchNormalization())
model_mn.add(Dense(32, activation='selu')) # 128 слои не очень точные, но быстрые, оптимал
model_mn.add(BatchNormalization())
model_mn.add(LeakyReLU())
model_mn.add(Dense(16, activation='selu'))
model_mn.add(BatchNormalization())
model_mn.add(Dense(1))
model_mn.add(Activation('selu'))


early_mn = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1, mode='au


# готовим компиляцию модели. Вызывать будем оптимизатор по имени, использовать стохастичес
# параметр создающий некую инерцию по методу импульсов - 0.5 (чтобы не застревать на миним
# и функция ошибки - "средняя абсолютная ошибка"
model_mn.compile(
    optimizer=tf.optimizers.SGD(learning_rate=0.02, momentum=0.5),   #вообще надо загонять
    loss='mean_absolute_error')


# итого
itogo_mn = model_mn.fit(          # призываем нашу модель с оптимизатором
    X_train_mn,     # входящая и целевая выборки
    y_train_mn,
    batch_size = 64,    # вычисляем градиенты для каждых 64 наблюдений
    epochs=100, # 100 эпох берём, стопер у нас есть, а это потолок.
    verbose=1, # индикатор выполнения
    validation_split = 0.2, # доля обучающих данных, которые будут использоваться в качест
```

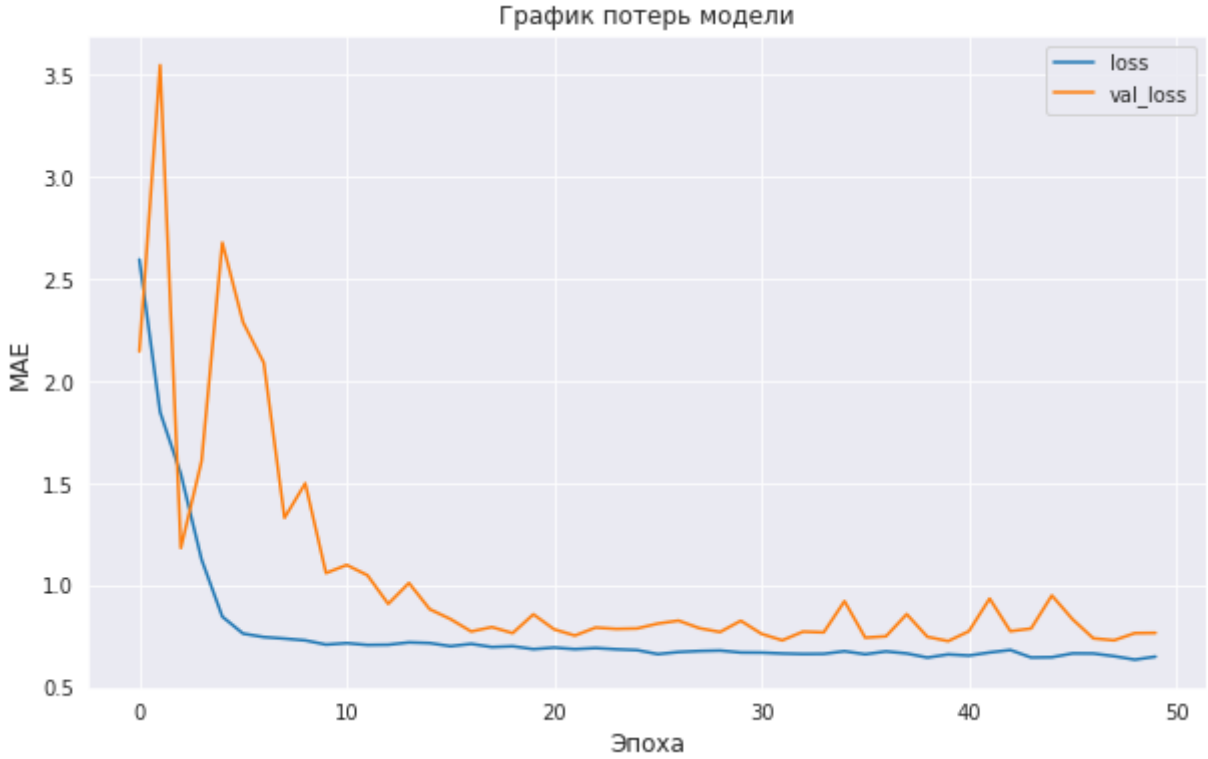
```
callbacks = [early_mn]  # колбэки в соответствии с нашими стопами
)
```

```
Epoch 1/100
9/9 [==============================] - 2s 39ms/step - loss: 2.5947 - val_loss: 2.14
Epoch 2/100
9/9 [==============================] - 0s 9ms/step - loss: 1.8478 - val_loss: 3.54
Epoch 3/100
9/9 [==============================] - 0s 9ms/step - loss: 1.5448 - val_loss: 1.18
Epoch 4/100
9/9 [==============================] - 0s 9ms/step - loss: 1.1261 - val_loss: 1.608
Epoch 5/100
9/9 [==============================] - 0s 7ms/step - loss: 0.8468 - val_loss: 2.67
Epoch 6/100
9/9 [==============================] - 0s 9ms/step - loss: 0.7639 - val_loss: 2.287
Epoch 7/100
9/9 [==============================] - 0s 7ms/step - loss: 0.7468 - val_loss: 2.089
Epoch 8/100
9/9 [==============================] - 0s 7ms/step - loss: 0.7394 - val_loss: 1.330
Epoch 9/100
9/9 [==============================] - 0s 7ms/step - loss: 0.7306 - val_loss: 1.499
Epoch 10/100
9/9 [==============================] - 0s 7ms/step - loss: 0.7101 - val_loss: 1.060
Epoch 11/100
9/9 [==============================] - 0s 7ms/step - loss: 0.7172 - val_loss: 1.099
Epoch 12/100
9/9 [==============================] - 0s 9ms/step - loss: 0.7083 - val_loss: 1.049
Epoch 13/100
9/9 [==============================] - 0s 9ms/step - loss: 0.7094 - val_loss: 0.909
Epoch 14/100
9/9 [==============================] - 0s 7ms/step - loss: 0.7208 - val_loss: 1.01
Epoch 15/100
9/9 [==============================] - 0s 9ms/step - loss: 0.7175 - val_loss: 0.88
Epoch 16/100
9/9 [==============================] - 0s 9ms/step - loss: 0.7022 - val_loss: 0.83
Epoch 17/100
9/9 [==============================] - 0s 9ms/step - loss: 0.7136 - val_loss: 0.774
Epoch 18/100
9/9 [==============================] - 0s 8ms/step - loss: 0.6981 - val_loss: 0.79
Epoch 19/100
9/9 [==============================] - 0s 7ms/step - loss: 0.7021 - val_loss: 0.76
Epoch 20/100
9/9 [==============================] - 0s 7ms/step - loss: 0.6878 - val_loss: 0.858
Epoch 21/100
9/9 [==============================] - 0s 9ms/step - loss: 0.6954 - val_loss: 0.784
Epoch 22/100
9/9 [==============================] - 0s 8ms/step - loss: 0.6878 - val_loss: 0.754
Epoch 23/100
9/9 [==============================] - 0s 7ms/step - loss: 0.6932 - val_loss: 0.79
Epoch 24/100
9/9 [==============================] - 0s 7ms/step - loss: 0.6870 - val_loss: 0.78
Epoch 25/100
9/9 [==============================] - 0s 7ms/step - loss: 0.6831 - val_loss: 0.789
Epoch 26/100
9/9 [==============================] - 0s 8ms/step - loss: 0.6635 - val_loss: 0.81
Epoch 27/100
9/9 [==============================] - 0s 7ms/step - loss: 0.6738 - val_loss: 0.827
Epoch 28/100
9/9 [==============================] - 0s 7ms/step - loss: 0.6785 - val_loss: 0.790
Epoch 29/100
```

```
# смотрим результаты работы нашей модели
model_mn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               1664

 batch_normalization (BatchN  (None, 128)              512
 ormalization)

 leaky_re_lu (LeakyReLU)     (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 batch_normalization_1 (Batc  (None, 128)              512
 hNormalization)

 dense_2 (Dense)             (None, 64)                8256

 batch_normalization_2 (Batc  (None, 64)               256
 hNormalization)

 dense_3 (Dense)             (None, 32)                2080

 batch_normalization_3 (Batc  (None, 32)               128
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 32)                0

 dense_4 (Dense)             (None, 16)                528

 batch_normalization_4 (Batc  (None, 16)               64
 hNormalization)

 dense_5 (Dense)             (None, 1)                 17

 activation (Activation)     (None, 1)                 0

=================================================================
Total params: 30,529
Trainable params: 29,793
Non-trainable params: 736
_________________________________________________________________
```

```
# визуализируем потери модели
history = itogo_mn
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('График потерь модели', size=12)
plt.ylabel('MAE', size=12)    # наша средняя квадратичная ошибка
plt.xlabel('Эпоха', size=12)
plt.legend(['loss', 'val_loss'], loc='best') ; # ошибка на обучаемой и на тестовой выборке
```




```
# делаем предикт
pred_mn = model_mn.predict(np.array((X_test_mn)))
original_mn = y_test_mn.values
predicted_mn = pred_mn
```

```
     9/9 [==============================] - 0s 2ms/step
```

```
# визуализируем график оригинального и предсказанного значения У
plt.figure(figsize=(10,6))
plt.title('Тестовые и прогнозные значения', size=12)
plt.plot(original_mn, color='blue', label = 'Тестовые значения')
plt.plot(predicted_mn, color='yellow', label = 'Прогнозные значения')
plt.legend(loc='best') ;
```