ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу «Data Science»

Тема: Прогнозирование конечных свойств новых материалов (композиционных материалов).

Слушатель: Киселев Дмитрий Владимирович

Постановка задачи:

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

- ознакомление с предоставленными материалами
- объединение в один датасет и удаление не нужных столбцов
- разведочный анализ данных и дополнительные преобразования
- предобработка данных и их разбивка на тренировочные и тестовые
- обучение модели на различных алгоритмах с гиперпараметрами по умолчанию
- подбор лучших гиперпараметров, поиск по сетке с перекрестной проверкой
- сравнение результатов до и после в каждом алгоритме, выбор лучшего результата
- сравнение результатов между используемыми алгоритмами, выбор лучшего
- оценить качество лучшей модели на тренировочной и тестовой выборках
- создать приложение для предсказания значения целевой переменной
- оформление проделанной работы

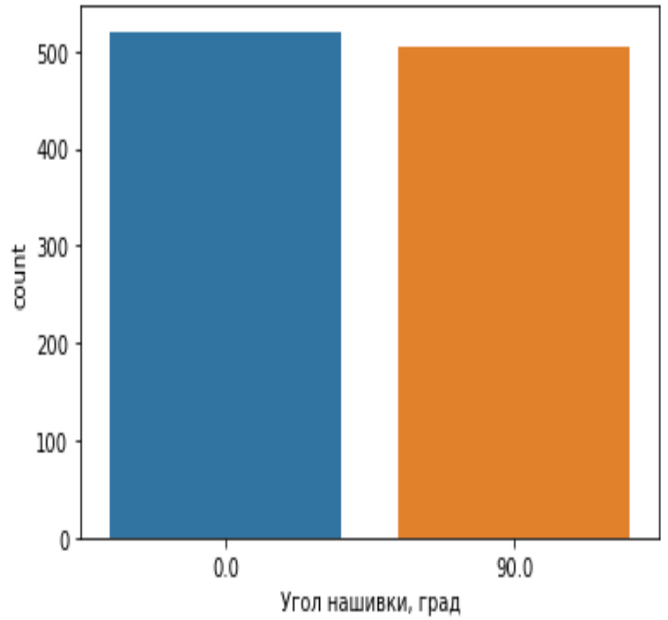
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 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: 104.0 KB
```

```
df.nunique()
```

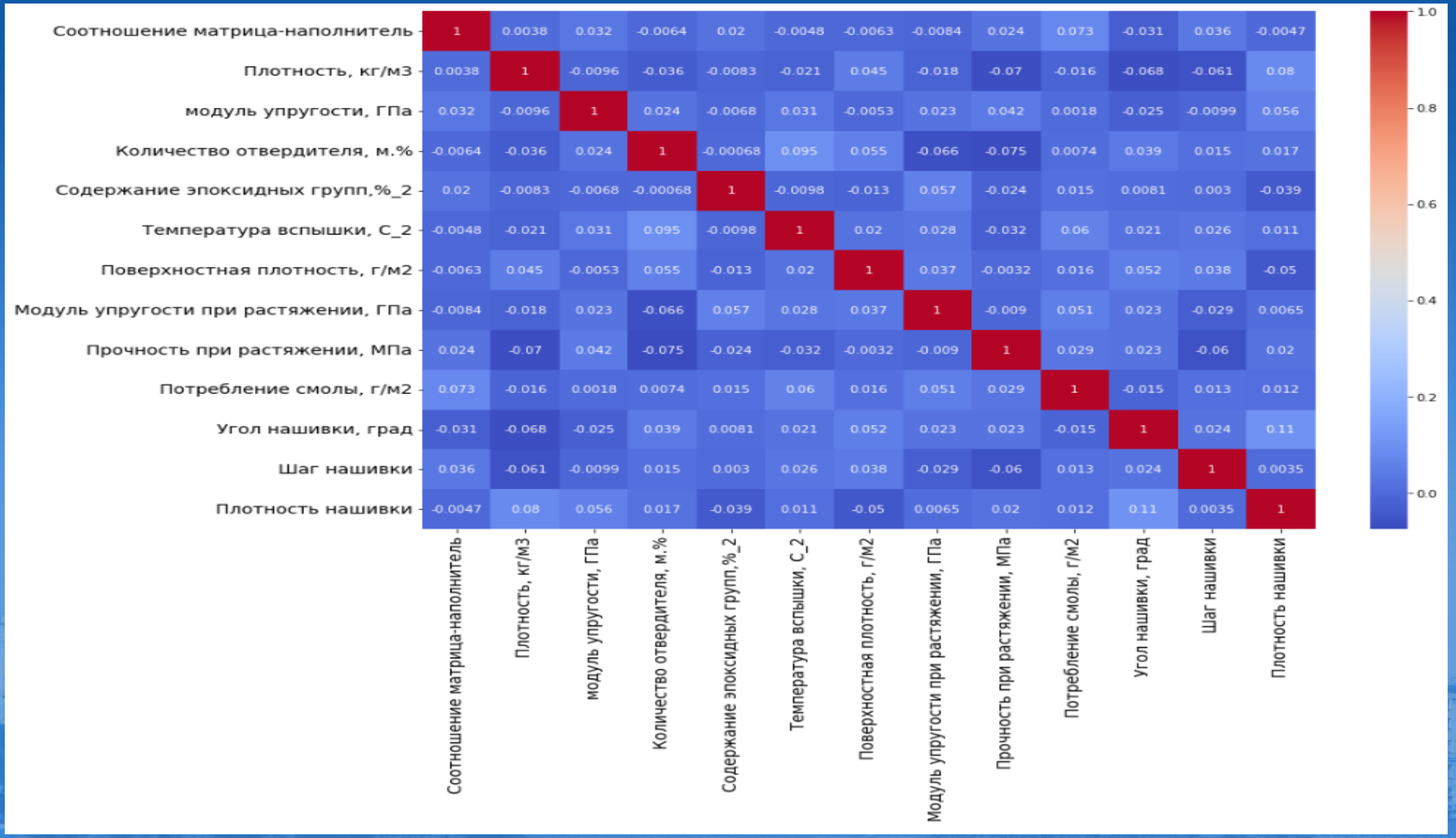
```
Соотношение матрица-наполнитель          1014
Плотность, кг/м3                         1013
модуль упругости, ГПа                    1020
Количество отвердителя, м.%              1005
Содержание эпоксидных групп,%_2          1004
Температура вспышки, С_2                 1003
Поверхностная плотность, 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
```

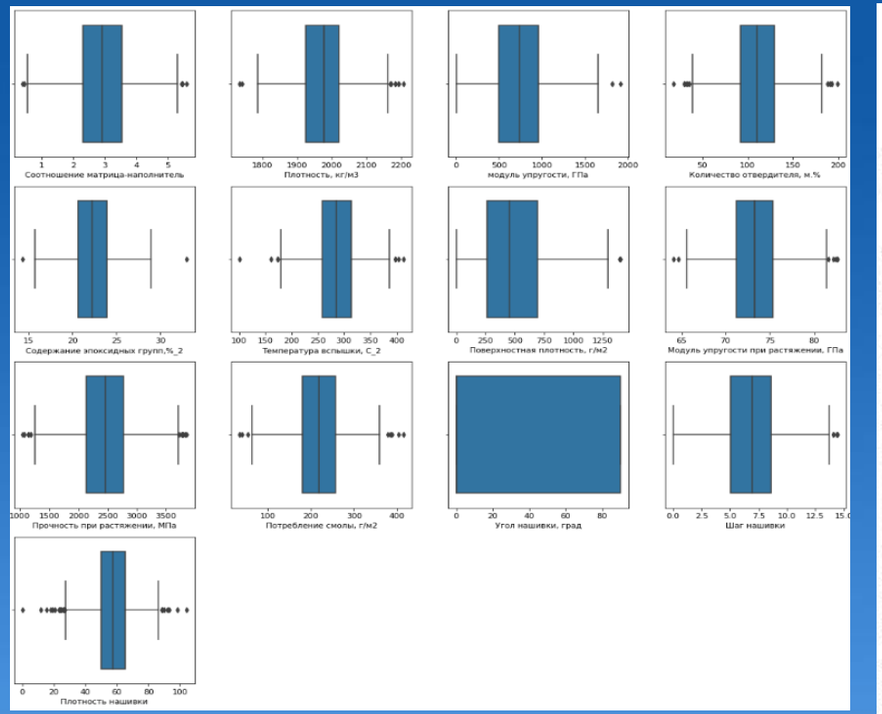
```
<AxesSubplot:xlabel='Угол нашивки, град', ylabel='count'>
```

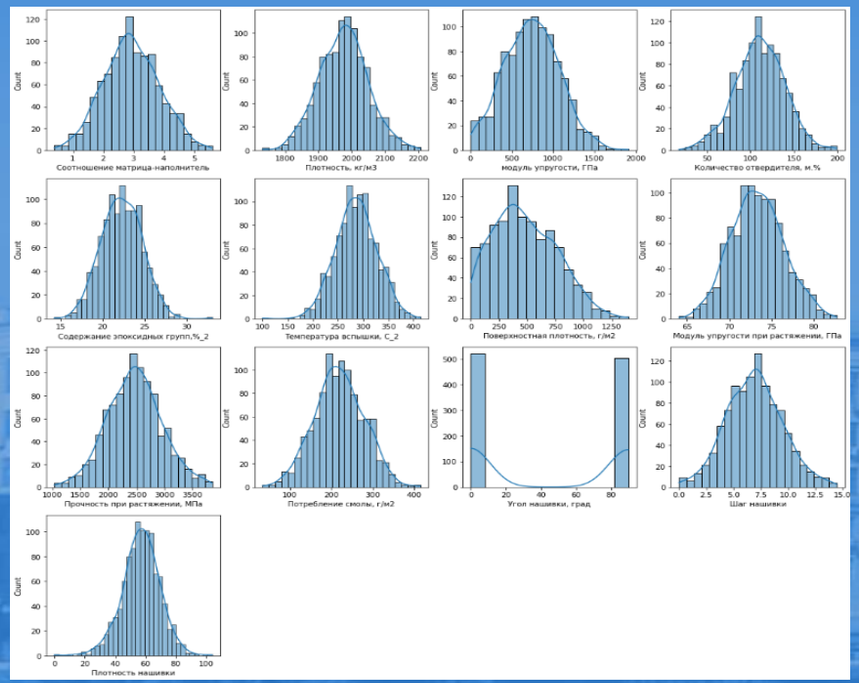


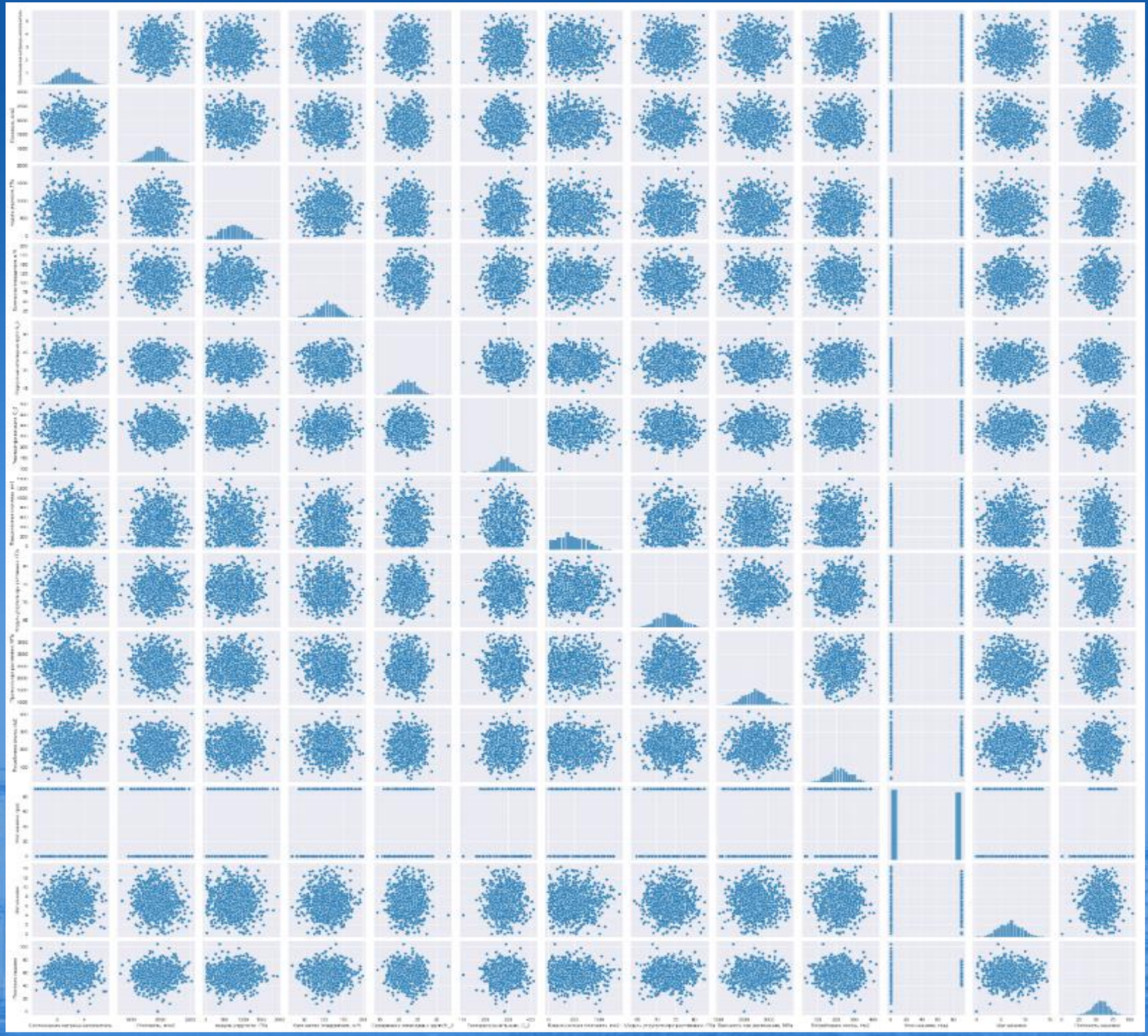

```
df.describe()
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град	Шаг нашивки	Плотность нашивки
count	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000
mean	2.930366	1975.734888	739.923233	110.570769	22.244390	285.882151	482.731833	73.328571	2466.922843	218.423144	44.252199	6.899222	57.153929
std	0.913222	73.729231	330.231581	28.295911	2.406301	40.943260	281.314690	3.118983	485.628006	59.735931	45.015793	2.563467	12.350969
min	0.389403	1731.764635	2.436909	17.740275	14.254985	100.000000	0.603740	64.054061	1036.856605	33.803026	0.000000	0.000000	0.000000
25%	2.317887	1924.155467	500.047452	92.443497	20.608034	259.066528	266.816645	71.245018	2135.850448	179.627520	0.000000	5.080033	49.799212
50%	2.906878	1977.621657	739.664328	110.564840	22.230744	285.896812	451.864365	73.268805	2459.524526	219.198882	0.000000	6.916144	57.341920
75%	3.552660	2021.374375	961.812526	129.730366	23.961934	313.002106	693.225017	75.356612	2767.193119	257.481724	90.000000	8.586293	64.944961
max	5.591742	2207.773481	1911.536477	198.953207	33.000000	413.273418	1399.542362	82.682051	3848.436732	414.590628	90.000000	14.440522	103.988901


Соотношение матрица-наполнитель 1 0.0038 0.032 -0.0064 0.02 -0.0048 -0.0063 -0.0084 0.024 0.073 -0.031 0.036 -0.0047
Плотность, кг/м3 0.0038 1 -0.0096 -0.036 -0.0083 -0.021 0.045 -0.018 -0.07 -0.016 -0.068 -0.061 0.08
модуль упругости, ГПа 0.032 -0.0096 1 0.024 -0.0068 0.031 -0.0053 0.023 0.042 0.0018 -0.025 -0.0099 0.056
Количество отвердителя, м.% -0.0064 -0.036 0.024 1 -0.00068 0.095 0.055 -0.066 -0.075 0.0074 0.039 0.015 0.017
Содержание эпоксидных групп,%_2 0.02 -0.0083 -0.0068 -0.00068 1 -0.0098 -0.013 0.057 -0.024 0.015 0.0081 0.003 -0.039
Температура вспышки, С_2 -0.0048 -0.021 0.031 0.095 -0.0098 1 0.02 0.028 -0.032 0.06 0.021 0.026 0.011
Поверхностная плотность, г/м2 -0.0063 0.045 -0.0053 0.055 -0.013 0.02 1 0.037 -0.0032 0.016 0.052 0.038 -0.05
Модуль упругости при растяжении, ГПа -0.0084 -0.018 0.023 -0.066 0.057 0.028 0.037 1 -0.009 0.051 0.023 -0.029 0.0065
Прочность при растяжении, МПа 0.024 -0.07 0.042 -0.075 -0.024 -0.032 -0.0032 -0.009 1 0.029 0.023 -0.06 0.02
Потребление смолы, г/м2 0.073 -0.016 0.0018 0.0074 0.015 0.06 0.016 0.051 0.029 1 -0.015 0.013 0.012
Угол нашивки, град -0.031 -0.068 -0.025 0.039 0.0081 0.021 0.052 0.023 0.023 -0.015 1 0.024 0.11
Шаг нашивки 0.036 -0.061 -0.0099 0.015 0.003 0.026 0.038 -0.029 -0.06 0.013 0.024 1 0.0035
Плотность нашивки -0.0047 0.08 0.056 0.017 -0.039 0.011 -0.05 0.0065 0.02 0.012 0.11 0.0035 1
Соотношение матрица-наполнитель
Плотность, кг/м3
модуль упругости, ГПа
Количество отвердителя, м.%
Содержание эпоксидных групп,%_2
Температура вспышки, С_2
Поверхностная плотность, г/м2
Модуль упругости при растяжении, ГПа
Прочность при растяжении, МПа
Потребление смолы, г/м2
Угол нашивки, град
Шаг нашивки
Плотность нашивки
1.0
0.8
0.6
0.4
0.2
0.0


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


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



```
[ ]  zz = abs(stats.zscore(df))
     print(np.where(zz > 3))
```

```
(array([  0,   2,  19,  41, 100, 100, 140, 211, 257, 279, 298, 375, 378,
        412, 438, 461, 464, 503, 592, 692, 718, 770, 791, 873, 918],
       dtype=int64), array([ 5,  4, 12, 12,  1,  9, 12,  9,  3, 12,  4,  7,  5,  3, 12, 12, 12,
         9,  2,  6,  6,  2,  5,  1,  1], dtype=int64))
```

Как видно, получилось два массива: первый - номера (индексы) строк в которых есть выбросы; второй - номера (индексы) столбцов выбросов соответствующие номерам (индексам) строк из первого. Проще говоря, в какой строке и по какому столбцу имеется выброс.

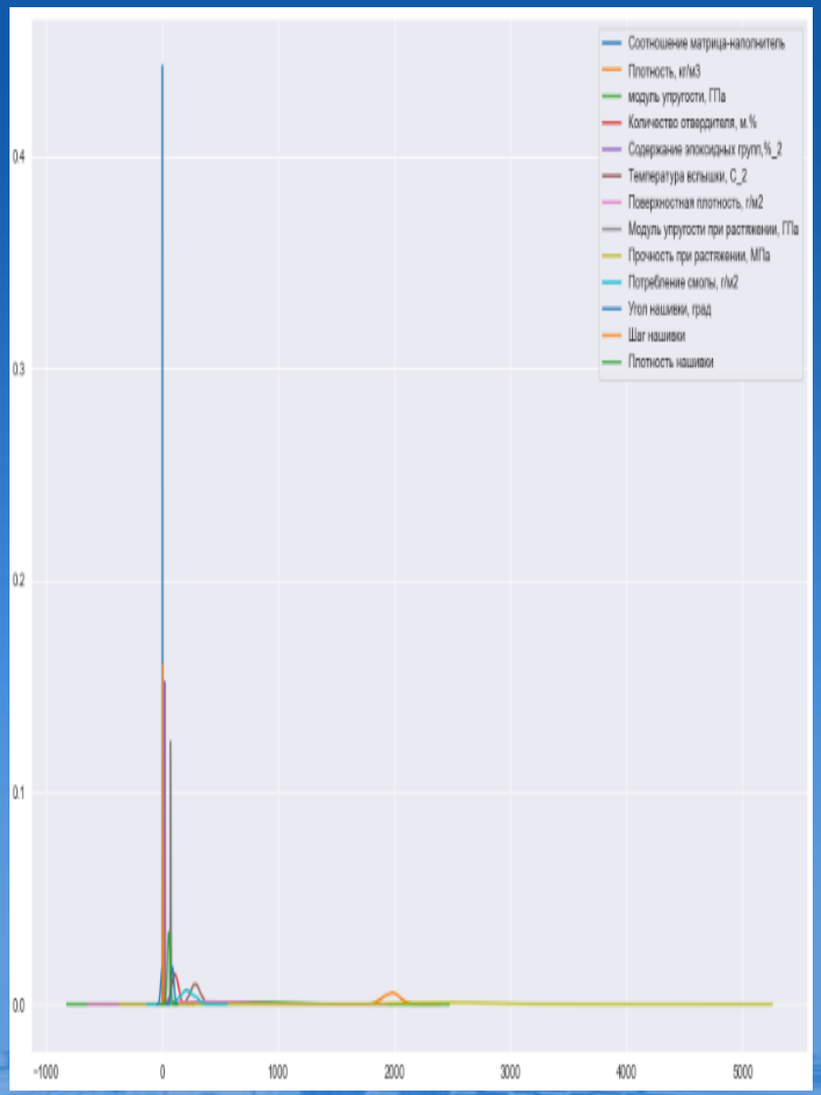
Также видно, что дважды прописана строка с номером (индексом) 100, это значит что в данной строке имеется 2 выброса, по столбцам с номерами (индексами) 1 и 9, т.е. в столбце плотность и в столбце потребление смолы выбросы в 101 строке (она же строка с индексом 100).

```
#посчитаем выбросы z = np.abs(stats.zscore(df[col]))
total_z = 0
count_z = []
for col in df:
    df_z = df.loc[:, [col]]
    z = np.abs(stats.zscore(df[col]))
    df_z['3sig'] = z > 3
    total_z += df_z['3sig'].sum()
    count_z.append(df_z['3sig'].sum())
    print(col, '=', df_z['3sig'].sum())
print('Количество выбросов по столбцам:', count_z, sep='\n')
print(f'Общее количество выбросов = {total_z}')
print(np.where(abs(stats.zscore(df)) > 3))
'''
каждый раз создаётся новый дата фрейм из 2-х столбцов, первый перебирает текущие столбцы по порядку,
второй - df_z['3sig'] присваивает значения либо False, либо True.
total_z += df_z['3sig'].sum() - количество значений True в столбце df_z['3sig'] суммируются в переменную total_z.
'''
```

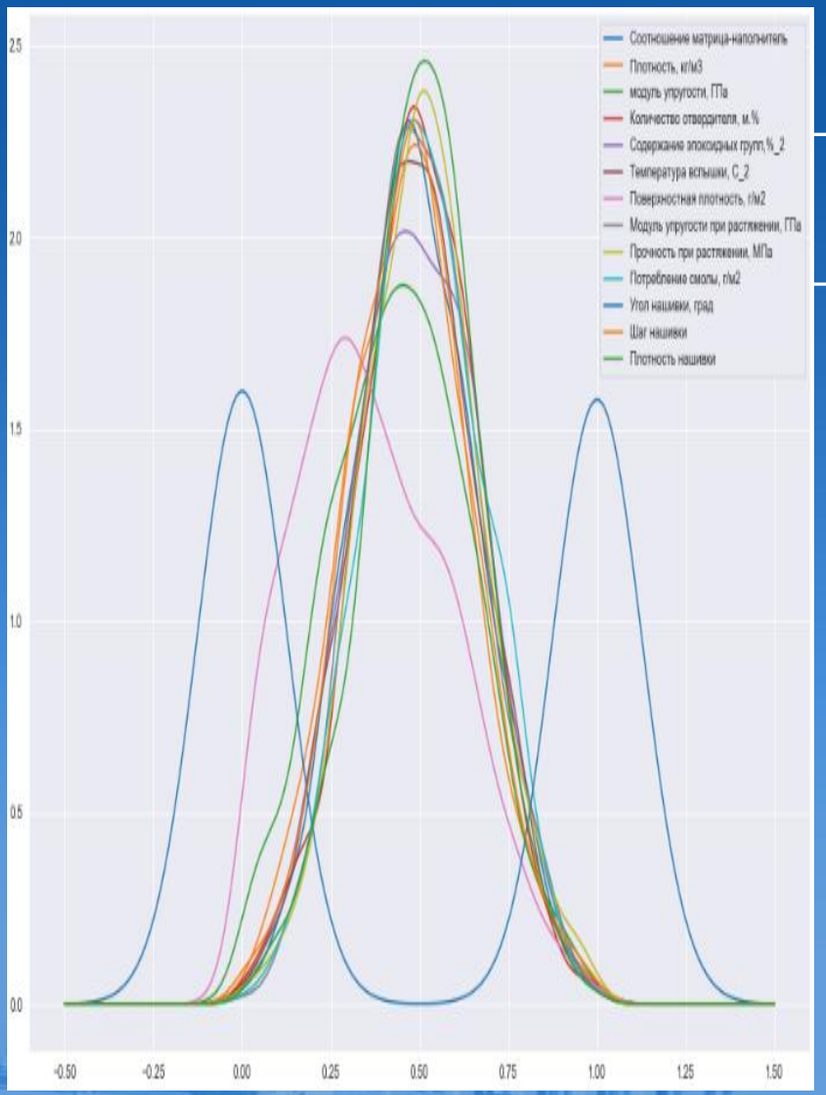
```
#удаление выбросов
df_z = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]
df_z
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град	Шаг нашивки	Плотность нашивки
1	1.857143	2030.000000	738.736842	50.000000	23.750000	284.615385	210.000000	70.000000	3000.000000	220.000000	0.0	4.000000	60.000000
3	1.857143	2030.000000	738.736842	129.000000	21.250000	300.000000	210.000000	70.000000	3000.000000	220.000000	0.0	5.000000	47.000000
4	2.771331	2030.000000	753.000000	111.860000	22.267857	284.615385	210.000000	70.000000	3000.000000	220.000000	0.0	5.000000	57.000000
5	2.767918	2000.000000	748.000000	111.860000	22.267857	284.615385	210.000000	70.000000	3000.000000	220.000000	0.0	5.000000	60.000000
6	2.569620	1910.000000	807.000000	111.860000	22.267857	284.615385	210.000000	70.000000	3000.000000	220.000000	0.0	5.000000	70.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1018	2.271346	1952.087902	912.855545	86.992183	20.123249	324.774576	209.198700	73.090961	2387.292495	125.007669	90.0	9.076380	47.019770
1019	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401	350.660830	72.920827	2360.392784	117.730099	90.0	10.565614	53.750790
1020	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047	740.142791	74.734344	2662.906040	236.606764	90.0	4.161154	67.629684
1021	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840	641.468152	74.042708	2071.715856	197.126067	90.0	6.313201	58.261074
1022	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708	758.747882	74.309704	2856.328932	194.754342	90.0	6.078902	77.434468

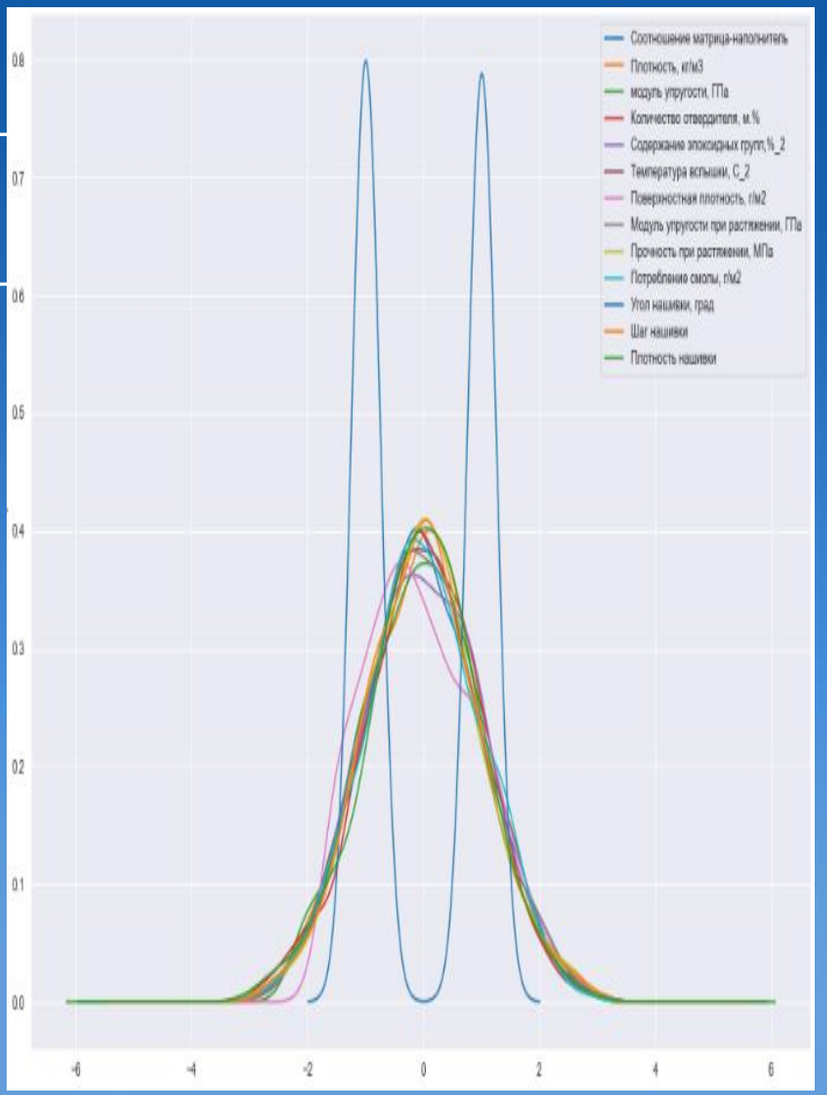
999 rows × 13 columns



kde (оценка плотности ядра) показала, что в датасете, признаки находятся в разных диапазонах, значит будем нормализовывать их.



Как видно (строки: min равны 0; max равны 1.) и из графика оценки плотности ядра, данные нормализовались.



Как видно (строки: mean стремится к 0; std стремится к 1.) и из графика оценки плотности ядра, данные стандартизировались.

```
ss = StandardScaler()
ss = ss.fit_transform(df_z)
#в результате получается numpy массив, который преобразуем в DataFrame
df_ss = pd.DataFrame(data=ss, columns=df_z.columns)
df_ss.describe()
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град	Шаг нашивки	Плотность нашивки
count	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02	9.990000e+02
mean	1.505858e-16	-3.097733e-15	6.756913e-17	-1.092442e-16	-2.911696e-16	-1.969840e-16	1.549200e-16	2.858797e-15	5.794497e-16	9.251859e-17	2.283125e-15	5.606682e-17	-7.245900e-17
std	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00	1.000501e+00
min	-2.804633e+00	-2.618774e+00	-2.247590e+00	-2.908544e+00	-2.742801e+00	-2.795859e+00	-1.725710e+00	-2.985712e+00	-2.953238e+00	-3.007020e+00	-9.930174e-01	-2.687171e+00	-3.099817e+00
25%	-6.781908e-01	-7.097124e-01	-7.280190e-01	-6.576547e-01	-6.925708e-01	-6.685947e-01	-7.664917e-01	-6.658067e-01	-6.832385e-01	-6.522420e-01	-9.930174e-01	-7.086606e-01	-6.195198e-01
50%	-3.114775e-02	2.571301e-02	7.899558e-03	-6.624289e-03	-5.819450e-03	-1.686254e-03	-1.048987e-01	-2.887993e-02	-1.965750e-02	4.413767e-03	-9.930174e-01	6.847284e-03	1.824727e-02
75%	6.801307e-01	6.267731e-01	6.748517e-01	6.833178e-01	7.311338e-01	6.728703e-01	7.612871e-01	6.495422e-01	6.089493e-01	6.644458e-01	1.007032e+00	6.570172e-01	6.468672e-01
max	2.922296e+00	2.979312e+00	2.781848e+00	2.946635e+00	2.819592e+00	2.925323e+00	2.925070e+00	2.973525e+00	2.857099e+00	2.863982e+00	1.007032e+00	2.943918e+00	3.012626e+00

```
mms = MinMaxScaler()
norm = mms.fit_transform(df_z)
#в результате получается numpy массив, который преобразуем в DataFrame
df_norm = pd.DataFrame(data=norm, columns=df_z.columns)
df_norm.describe()
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град	Шаг нашивки	Плотность нашивки
count	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000	999.000000
mean	0.489727	0.467798	0.446887	0.496747	0.493097	0.488685	0.371058	0.501023	0.508273	0.512182	0.496496	0.477203	0.507132
std	0.174701	0.178722	0.198929	0.170875	0.179869	0.174877	0.215125	0.167891	0.172193	0.170414	0.500238	0.177675	0.163683
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.371306	0.341020	0.302135	0.384427	0.368588	0.371822	0.206249	0.389296	0.390683	0.401086	0.000000	0.351355	0.405778
50%	0.484288	0.472391	0.448458	0.495616	0.492051	0.488391	0.348503	0.496176	0.504890	0.512933	0.000000	0.478419	0.510118
75%	0.608487	0.579760	0.581067	0.613450	0.624540	0.606296	0.534748	0.610020	0.613078	0.625356	1.000000	0.593879	0.612960
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Используемые методы:

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

1. Метод ближайших соседей k-NN.
2. Метод опорных векторов SVR.
3. Метод Решающих деревьев (Decision tree).
4. Метод Случайного леса (Random forest).
5. Линейная регрессия.
6. Гребневая регрессия RidgeCV.
7. Повышение градиента дерева (GradientBoostingRegressor).
8. Стохастический градиентный спуск (SGD) SGDRegressor.
9. Регрессор, который делает прогнозы, используя простые правила. DummyRegressor(*, strategy='mean', constant=None, quantile=None).
10. AdaBoostRegressor.

Оценки качества:

- MSE (Mean Squared Error) средняя квадратичная ошибка
- MAE (Mean Absolute Error) средняя абсолютная ошибка
- R2 коэф-т детерминации

Создам для каждой целевой переменной свою модель:

модуль упругости при растяжении

прочность при растяжении

соотношение матрица-наполнитель

```
parametrs_svr1 = {'kernel':['linear', 'poly', 'rbf'],
                  'gamma':['scale', 'auto']}
metrics_svr1 = ['neg_mean_squared_error', 'neg_mean_absolute_error', 'r2']
for elem1 in metrics_svr1:
    grid_svr1 = GridSearchCV(svr1, parametrs_svr1, cv=10, scoring=elem1)
    grid_svr1.fit(X1_train, y1_train)
    print(elem1, ':', grid_svr1.best_params_)
```

```
neg_mean_squared_error : {'gamma': 'auto', 'kernel': 'poly'}
neg_mean_absolute_error : {'gamma': 'auto', 'kernel': 'poly'}
r2 : {'gamma': 'auto', 'kernel': 'poly'}
```

```
svr1_best = SVR(kernel='poly', gamma='auto')
svr1_best.fit(X1_train, y1_train)
y_pred_svr1_best = svr1_best.predict(X1_test)
estimates_svr1_best = {
    'MSE: ': mean_squared_error(y1_test, y_pred_svr1_best),
    'MAE: ': mean_absolute_error(y1_test, y_pred_svr1_best),
    'R2: ': r2_score(y1_test, y_pred_svr1_best)
}
estimates_svr1_best
```

```
{'MSE: ': 0.02625366400407608,
 'MAE: ': 0.13056059523819394,
 'R2: ': 0.007325511935029616}
```

```
parametrs_lin_reg1 = {'fit_intercept':[True, False],
                      'copy_X':[True, False],
                      'n_jobs':[-1, 1],
                      'positive':[True, False]}
metrics_lin_reg1 = ['neg_mean_squared_error', 'neg_mean_absolute_error', 'r2']
for elem1 in metrics_lin_reg1:
    grid_lin_reg1 = GridSearchCV(lin_reg1, parametrs_lin_reg1, cv=10, scoring=elem1)
    grid_lin_reg1.fit(X1_train, y1_train)
    print(elem1, ':', grid_lin_reg1.best_params_)
```

```
neg_mean_squared_error : {'copy_X': True, 'fit_intercept': True, 'n_jobs': -1, 'positive': True}
neg_mean_absolute_error : {'copy_X': True, 'fit_intercept': True, 'n_jobs': -1, 'positive': True}
r2 : {'copy_X': True, 'fit_intercept': True, 'n_jobs': -1, 'positive': True}
```

sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True, copy_X=True, n_jobs=None, positive=False*)

```
parametrs_gbr1 = {'n_estimators':range(50, 151, 50),
                  'max_depth':[3, 5, 7],
                  'random_state':[1]}
metrics_gbr1 = ['neg_mean_squared_error', 'neg_mean_absolute_error', 'r2']
for elem1 in metrics_gbr1:
    grid_gbr1 = GridSearchCV(gbr1, parametrs_gbr1, cv=10, scoring=elem1)
    grid_gbr1.fit(X1_train, y1_train)
    print(elem1, ':', grid_gbr1.best_params_)
```

```
neg_mean_squared_error : {'max_depth': 3, 'n_estimators': 50, 'random_state': 1}
neg_mean_absolute_error : {'max_depth': 3, 'n_estimators': 50, 'random_state': 1}
r2 : {'max_depth': 3, 'n_estimators': 50, 'random_state': 1}
```

```
parametrs_sgdr1 = {'penalty':['l2', 'l1', 'elasticnet', None],
                   'shuffle':[True, False],
                   'learning_rate':['constant', 'optimal', 'invscaling', 'adaptive'],
                   'random_state':[1]}
metrics_sgdr1 = ['neg_mean_squared_error', 'neg_mean_absolute_error', 'r2']
for elem1 in metrics_sgdr1:
    grid_sgdr1 = GridSearchCV(sgdr1, parametrs_sgdr1, cv=10, scoring=elem1)
    grid_sgdr1.fit(X1_train, y1_train)
    print(elem1, ':', grid_sgdr1.best_params_)
```

```
neg_mean_squared_error : {'learning_rate': 'adaptive', 'penalty': 'l1', 'random_state': 1, 'shuffle': True}
neg_mean_absolute_error : {'learning_rate': 'adaptive', 'penalty': 'l1', 'random_state': 1, 'shuffle': True}
r2 : {'learning_rate': 'adaptive', 'penalty': 'l1', 'random_state': 1, 'shuffle': True}
```

```
parametrs_abr1 = {'n_estimators':range(25, 76, 25),
                  'learning_rate':np.arange(0.1, 2.2, 0.2),
                  'random_state':[1]}
metrics_abr1 = ['neg_mean_squared_error', 'neg_mean_absolute_error', 'r2']
for elem1 in metrics_abr1:
    grid_abr1 = GridSearchCV(abr1, parametrs_abr1, cv=10, scoring=elem1)
    grid_abr1.fit(X1_train, y1_train)
    print(elem1, ':', grid_abr1.best_params_)
```

```
neg_mean_squared_error : {'learning_rate': 0.1, 'n_estimators': 25, 'random_state': 1}
neg_mean_absolute_error : {'learning_rate': 0.1, 'n_estimators': 25, 'random_state': 1}
r2 : {'learning_rate': 0.1, 'n_estimators': 25, 'random_state': 1}
```

Модуль упругости при растяжении

	MSE:	MAE:	R2:
KNeighborsRegressor(n_neighbors=30)	0.026989	0.132739	-0.020474
SVR(kernel="poly", gamma="auto")	0.026254	0.130561	0.007326
DecisionTreeRegressor(criterion="absolute_error", splitter="random", max_depth=1, random_state=1)	0.026285	0.130917	0.006147
RandomForestRegressor(n_estimators=150, criterion="absolute_error", random_state=1)	0.027026	0.132052	-0.021895
LinearRegression(fit_intercept=True, copy_X=True, n_jobs=-1, positive=True)	0.026172	0.130441	0.010415
RidgeCV(fit_intercept=True, gcv_mode="eigen")	0.026343	0.130956	0.003947
GradientBoostingRegressor(n_estimators=50, max_depth=3, random_state=1)	0.028132	0.136365	-0.063705
SGDRegressor(penalty="l1", shuffle=False, learning_rate="adaptive", random_state=1)	0.026533	0.132072	-0.003246
DummyRegressor(strategy="median", quantile=0.1)	0.026454	0.130975	-0.000254
AdaBoostRegressor(n_estimators=25, learning_rate=0.1, random_state=1)	0.026307	0.130463	0.005294

Прочность при растяжении

	MSE:	MAE:	R2:
KNeighborsRegressor(n_neighbors=15)	1.113348	0.829424	-0.069217
SVR(kernel="linear", gamma="scale")	1.059066	0.828067	-0.017088
DecisionTreeRegressor(criterion="squared_error", splitter="best", max_depth=1, random_state=1)	1.067445	0.814039	-0.025134
RandomForestRegressor(n_estimators=50, criterion="absolute_error", random_state=1)	1.050733	0.813191	-0.009085
LinearRegression(fit_intercept=False, copy_X=True, n_jobs=-1, positive=True)	1.066604	0.823477	-0.024326
RidgeCV(fit_intercept=False, gcv_mode="auto")	1.065749	0.822930	-0.023506
GradientBoostingRegressor(n_estimators=50, max_depth=3, random_state=1)	1.084678	0.819900	-0.041684
SGDRegressor(penalty="l1", shuffle=True, learning_rate="invscaling", random_state=1)	1.066826	0.823175	-0.024540
DummyRegressor(strategy="mean", quantile=0.1)	1.041286	0.802264	-0.000012
AdaBoostRegressor(n_estimators=25, learning_rate=0.3, random_state=1)	1.051999	0.812497	-0.010301

Соотношение матрица-наполнитель

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

```
model1 = Sequential()
#model1.add(Input(shape=X_train.shape[1]))
model1.add(Dense(128, activation='sigmoid', input_shape=[X_train.shape[1]]))
model1.add(Dense(128, activation='sigmoid'))
model1.add(Dense(1, activation='relu'))
#Выведем полученную модель на экран
model1.summary()
```

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

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 18,305
Trainable params: 18,305
Non-trainable params: 0
```

```
model2 = Sequential()
model2.add(Input(shape=X_train.shape[1]))
model2.add(Dense(8, activation='sigmoid'))
model2.add(Dense(8, activation='sigmoid'))
model2.add(Dense(1, activation='linear'))
#Выведем полученную модель на экран
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 8)                 104

 dense_4 (Dense)             (None, 8)                 72

 dense_5 (Dense)             (None, 1)                 9

=================================================================
Total params: 185
Trainable params: 185
Non-trainable params: 0
```

```
model3 = Sequential()
model3.add(Input(shape=X_train.shape[1]))
#добавим смещение, чтобы модель стала более гибкой
model3.add(Dense(128, activation='softmax', use_bias=True))
model3.add(Dense(64, activation='softmax', use_bias=True))
model3.add(Dense(32, activation='softmax', use_bias=True))
model3.add(Dense(1, activation='sigmoid', use_bias=True))
#Выведем полученную модель на экран
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 128)               1664

 dense_7 (Dense)             (None, 64)                8256

 dense_8 (Dense)             (None, 32)                2080

 dense_9 (Dense)             (None, 1)                 33

=================================================================
Total params: 12,033
Trainable params: 12,033
Non-trainable params: 0
```

```
model4 = Sequential()
model4.add(Input(shape=X_train.shape[1]))
model4.add(Dense(1028, activation='sigmoid'))
model4.add(Dense(512, activation='sigmoid'))
model4.add(Dense(256, activation='sigmoid'))
model4.add(Dense(128, activation='sigmoid'))
model4.add(Dense(64, activation='sigmoid'))
model4.add(Dense(1, activation='relu'))
#Выведем полученную модель на экран
model4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 1028)              13364

 dense_11 (Dense)            (None, 512)               526848

 dense_12 (Dense)            (None, 256)               131328

 dense_13 (Dense)            (None, 128)               32896

 dense_14 (Dense)            (None, 64)                8256

 dense_15 (Dense)            (None, 1)                 65

=================================================================
Total params: 712,757
Trainable params: 712,757
Non-trainable params: 0
```

```
model5 = Sequential()
model5.add(Input(shape=X_train.shape[1]))
model5.add(Dense(2048, activation='tanh'))
model5.add(Dense(256, activation='tanh'))
model5.add(Dense(64, activation='tanh'))
model5.add(Dense(1, activation='linear'))
#Выведем полученную модель на экран
model5.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_16 (Dense)            (None, 2048)              26624

 dense_17 (Dense)            (None, 256)               524544

 dense_18 (Dense)            (None, 64)                16448

 dense_19 (Dense)            (None, 1)                 65

=================================================================
Total params: 567,681
Trainable params: 567,681
Non-trainable params: 0
```

```
model6 = Sequential()
model6.add(Input(shape=X_train.shape[1]))
model6.add(Dense(512, activation='sigmoid'))
model6.add(Dense(64, activation='sigmoid'))
model6.add(Dense(8, activation='sigmoid'))
model6.add(Dense(1, activation='linear'))
#Выведем полученную модель на экран
model6.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 512)               6656

 dense_21 (Dense)            (None, 64)                32832

 dense_22 (Dense)            (None, 8)                 520

 dense_23 (Dense)            (None, 1)                 9

=================================================================
Total params: 40,017
Trainable params: 40,017
Non-trainable params: 0
```

Соотношение матрица-наполнитель

```
model7 = Sequential()
model7.add(Input(shape=X_train.shape[1]))
model7.add(Dense(128, activation='sigmoid'))
model7.add(Dense(64, activation='sigmoid'))
model7.add(Dense(32, activation='sigmoid'))
model7.add(Dense(16, activation='sigmoid'))
model7.add(Dense(8, activation='sigmoid'))
model7.add(Dense(1, activation='linear'))
#выведем полученную модель на экран
model7.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_24 (Dense)            (None, 128)               1664

 dense_25 (Dense)            (None, 64)                8256

 dense_26 (Dense)            (None, 32)                2080

 dense_27 (Dense)            (None, 16)                528

 dense_28 (Dense)            (None, 8)                 136

 dense_29 (Dense)            (None, 1)                 9

=================================================================
Total params: 12,673
Trainable params: 12,673
Non-trainable params: 0
```

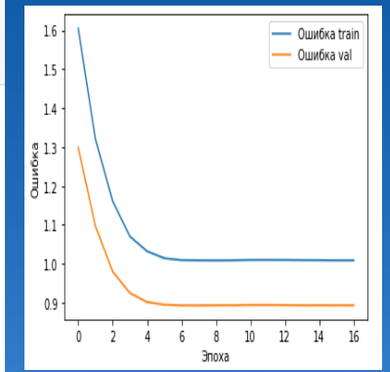
```
X = df_ss.drop('Соотношение матрица-наполнитель', axis=1)
y = df_ss['Соотношение матрица-наполнитель']
X_tr, X_test, y_tr, y_test = train_test_split(X, y, test_size=0.15, random_state=1)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_tr, y_tr, test_size=0.15, random_state=1)
```

```
model7.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_squared_error'])
```

```
#функция ранней остановки, patience=5 - если в течение 5-ти эпох потери на валидации (val_loss) не будут уменьшаться
stop7 = tf.keras.callbacks.EarlyStopping(monitor='val_loss', verbose=1, patience=5)
```

```
history7_train = model7.fit(X_train, y_train,
                            batch_size=70,
                            epochs=30,
                            verbose=1,
                            callbacks=[stop7],
                            validation_data=(X_val, y_val))
```



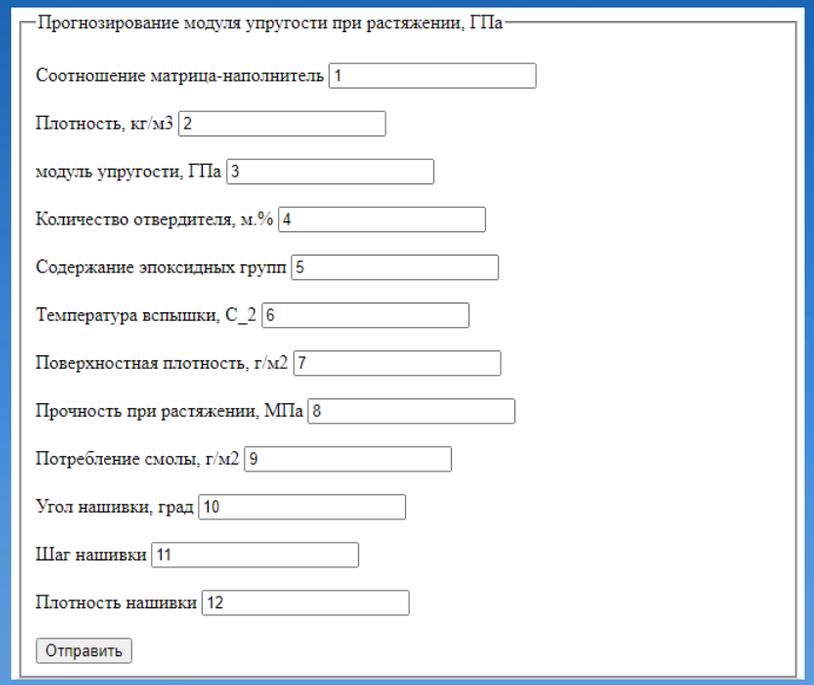
	0	1
history1_test	0.848025	1.050867
history2_test	0.860793	1.068046
history3_test	0.905366	1.186593
history4_test	0.848025	1.050867
history5_test	0.919941	1.220210
history6_test	0.849433	1.052788
history7_test	0.847440	1.050008

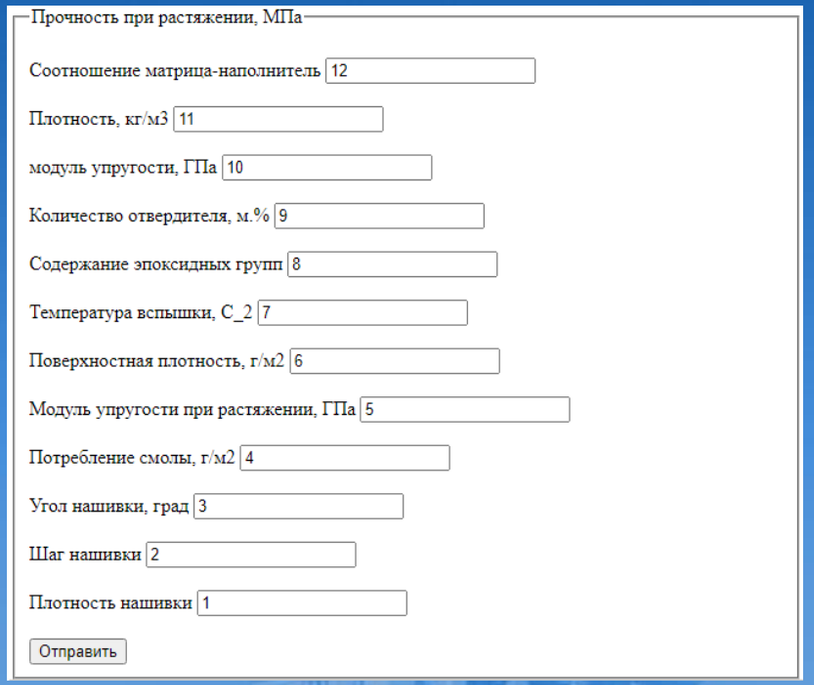
	0
r2_score1	-0.001128
r2_score2	-0.017495
r2_score3	-0.130430
r2_score4	-0.001128
r2_score5	-0.162457
r2_score6	-0.002959
r2_score7	-0.000310

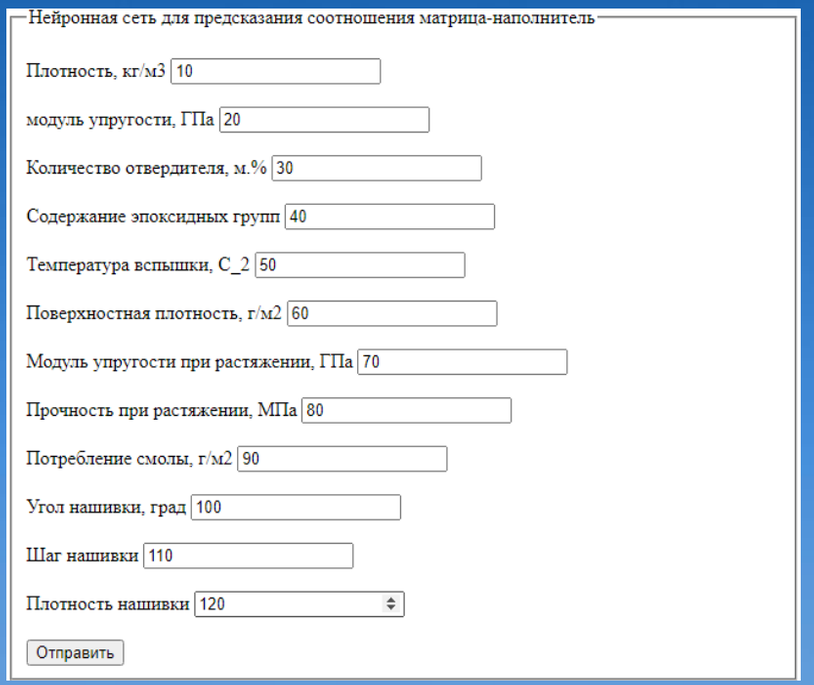
```
Epoch 1/30
11/11 [==============================] - 2s 30ms/step - loss: 1.0264 - mean_squared_error: 1.6047 - val_loss: 0.8985 - val_mean
_squared_error: 1.2987
Epoch 2/30
11/11 [==============================] - 0s 9ms/step - loss: 0.9177 - mean_squared_error: 1.3203 - val_loss: 0.8250 - val_mean_
squared_error: 1.0967
Epoch 3/30
11/11 [==============================] - 0s 8ms/step - loss: 0.8547 - mean_squared_error: 1.1611 - val_loss: 0.7818 - val_mean_
squared_error: 0.9801
Epoch 4/30
11/11 [==============================] - 0s 10ms/step - loss: 0.8187 - mean_squared_error: 1.0705 - val_loss: 0.7615 - val_mean
_squared_error: 0.9252
Epoch 5/30
11/11 [==============================] - 0s 6ms/step - loss: 0.8048 - mean_squared_error: 1.0323 - val_loss: 0.7516 - val_mean_
squared_error: 0.9017
Epoch 6/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7983 - mean_squared_error: 1.0145 - val_loss: 0.7485 - val_mean_
squared_error: 0.8952
Epoch 7/30
11/11 [==============================] - 0s 6ms/step - loss: 0.7972 - mean_squared_error: 1.0096 - val_loss: 0.7487 - val_mean_
squared_error: 0.8935
Epoch 8/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7976 - mean_squared_error: 1.0091 - val_loss: 0.7489 - val_mean_
squared_error: 0.8934
Epoch 9/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7971 - mean_squared_error: 1.0088 - val_loss: 0.7484 - val_mean_
squared_error: 0.8937
Epoch 10/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7972 - mean_squared_error: 1.0092 - val_loss: 0.7484 - val_mean_
squared_error: 0.8937
Epoch 11/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7975 - mean_squared_error: 1.0100 - val_loss: 0.7483 - val_mean_
squared_error: 0.8945
Epoch 12/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7973 - mean_squared_error: 1.0101 - val_loss: 0.7483 - val_mean_
squared_error: 0.8944
Epoch 13/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7972 - mean_squared_error: 1.0100 - val_loss: 0.7483 - val_mean_
squared_error: 0.8941
Epoch 14/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7973 - mean_squared_error: 1.0096 - val_loss: 0.7486 - val_mean_
squared_error: 0.8936
Epoch 15/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7972 - mean_squared_error: 1.0094 - val_loss: 0.7485 - val_mean_
squared_error: 0.8937
Epoch 16/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7971 - mean_squared_error: 1.0090 - val_loss: 0.7486 - val_mean_
squared_error: 0.8936
Epoch 17/30
11/11 [==============================] - 0s 7ms/step - loss: 0.7972 - mean_squared_error: 1.0090 - val_loss: 0.7486 - val_mean_
squared_error: 0.8936
Epoch 17: early stopping
```

Приложения

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана







Модуль упругости при растяжении, ГПа

[2.04292512]

Прочность при растяжении, МПа

[0.00418093]

Соотношение матрица-наполнитель

[[0.00019164]]

Заключение

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Распределение данных в датасете очень близко к нормальному, коэффициенты корреляции признаков стремятся к нулю. Использованные при обучении моделей методы не позволили получить достоверных предсказаний. Модели регрессии не показали высокой точности в предсказании целевых переменных. «Лучше» всего, показали себя методы машинного обучения: для модуля упругости при растяжении, ГПа – метод линейной регрессии; для прочности при растяжении, МПа – DummyRegressor. Предсказание целевых переменных в данном датасете не дало желаемых результатов. Самое главное из-за очень слабой корреляции между признаками. Поэтому проведя огромную работу с данными и применив довольно большое количество методов к ним, выявление лучших настраиваемых гиперпараметров, хорошего результата всё равно достигнуто не было. Немного улучшить результат можно путём увеличения тренировочной выборки и уменьшения тестовой выборки, но значительных улучшений не произойдёт. Поэтому необходимы дополнительные данные, получение новых признаков, консультации экспертов в предметной области, новые исследования, эффективная работа команды, состоящей из различных экспертов и дополнительные опыты с данными материалами. Задачу решить так, чтобы получить хороший результат на имеющихся данных не возможно ни одним из использованных методов.