

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА по курсу «Data Science»

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

Слушатель: Моисеева Мария Алексеевна

Загрузка и разведывательный анализ данных

Загружаем данные

```
# загружаем первый датасет
df1 = pd.read_excel('X_bp.xlsx')
df1
```

Unnamed: 0	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2
0	0	1.857143	2030.000000	738.736842	30.000000	22.267857	100.000000	210.000000	70.000000	3000.000000
1	1	1.857143	2030.000000	738.736842	50.000000	23.750000	284.615385	210.000000	70.000000	3000.000000
2	2	1.857143	2030.000000	738.736842	49.900000	33.000000	284.615385	210.000000	3000.000000	220.000000
3	3	1.857143	2030.000000	738.736842	129.000000	21.250000	300.000000	210.000000	3000.000000	220.000000
4	4	2.771331	2030.000000	753.000000	111.860000	22.267857	284.615385	210.000000	3000.000000	220.000000
...
1018	1018	2.271346	1952.087902	912.855545	86.992183	20.123249	324.774576	209.198700	73.090961	2387.292495
1019	1019	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401	350.660830	72.920827	2360.392784
1020	1020	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047	740.142791	74.734344	2662.906040
1021	1021	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840	641.468152	74.042708	2071.715856
1022	1022	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708	758.747882	74.309704	2856.328932

1023 rows x 11 columns

```
# загрузим второй датасет
df2 = pd.read_excel('X_nup.xlsx')
df2
```

Unnamed: 0	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	0	0	4.000000
1	1	0	4.000000
2	2	0	4.000000
3	3	0	5.000000
4	4	0	5.000000
...
1035	1035	90	8.088111
1036	1036	90	7.619138
1037	1037	90	9.800926
1038	1038	90	10.079859
1039	1039	90	9.021043

1040 rows x 4 columns

```
df3.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   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
```

```
# Объединяем по индексу, тип объединения INNER
df3 = pd.merge(df1,df2, how = 'inner', on = 'Index')
df3
```

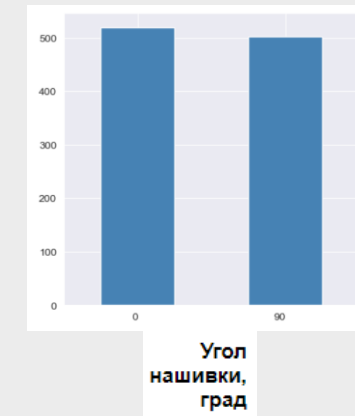
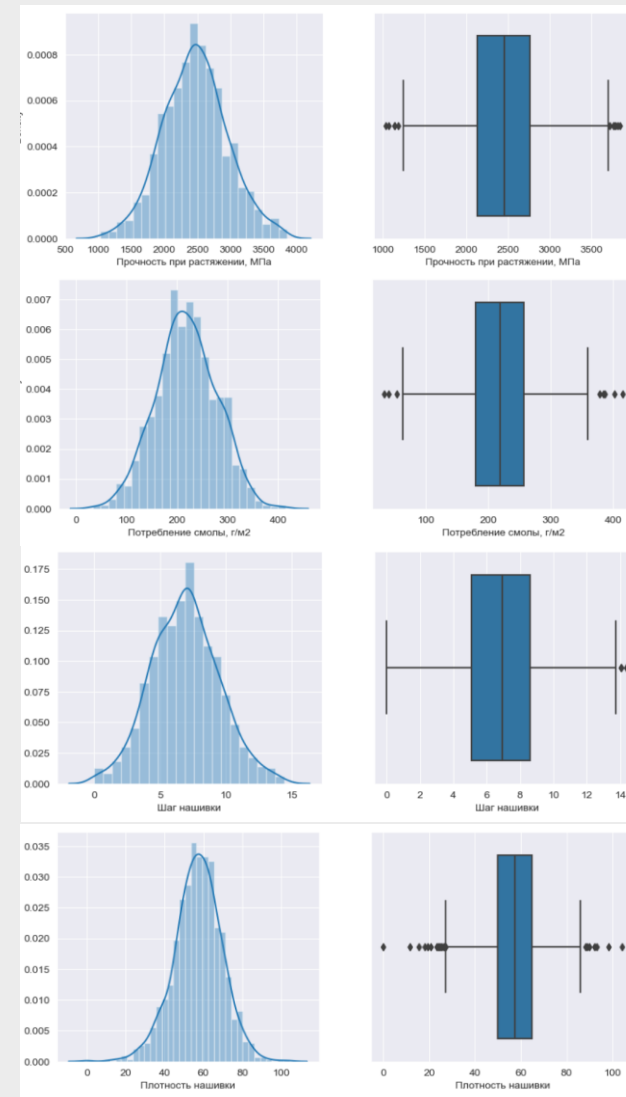
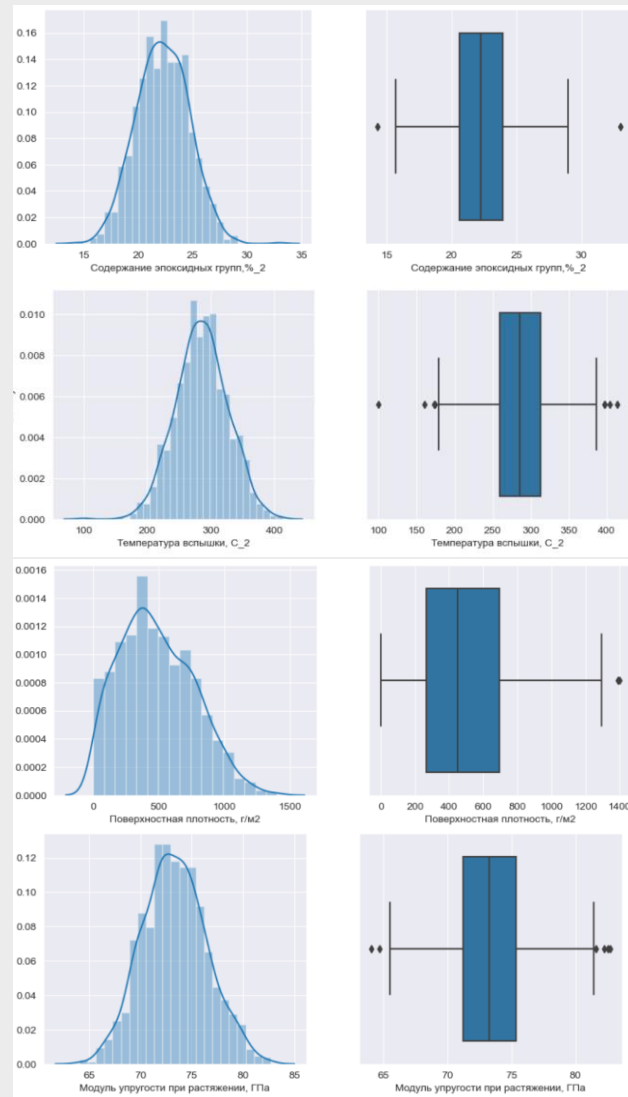
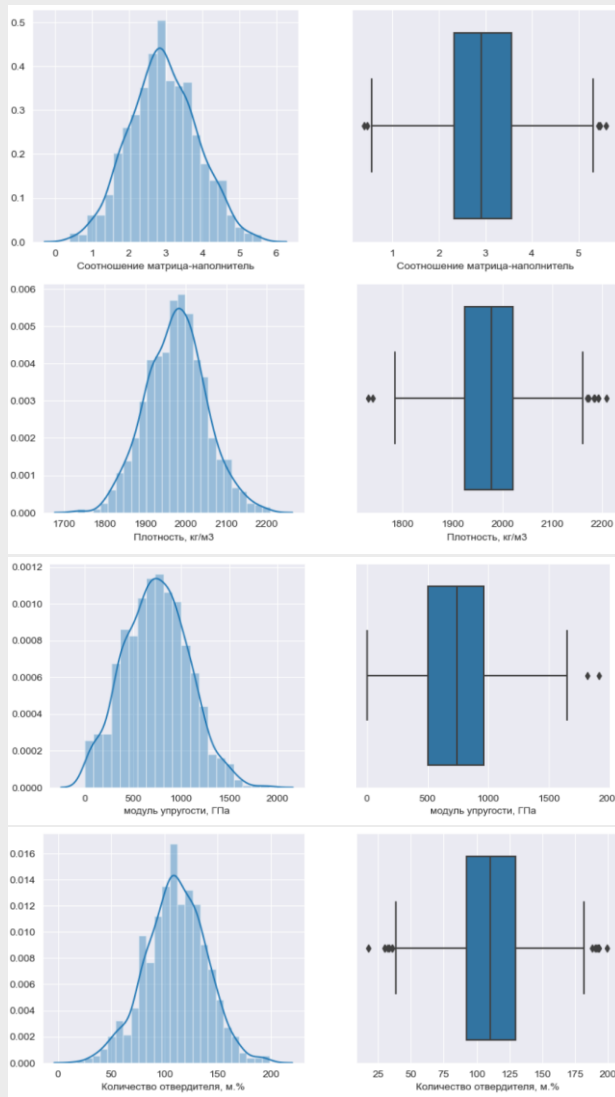
Index	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град
0	1.857143	2030.000000	738.736842	30.000000	22.267857	100.000000	210.000000	70.000000	3000.000000	220.000000	0
1	1.857143	2030.000000	738.736842	50.000000	23.750000	284.615385	210.000000	70.000000	3000.000000	220.000000	0
2	1.857143	2030.000000	738.736842	49.900000	33.000000	284.615385	210.000000	70.000000	3000.000000	220.000000	0
3	1.857143	2030.000000	738.736842	129.000000	21.250000	300.000000	210.000000	70.000000	3000.000000	220.000000	0
4	2.771331	2030.000000	753.000000	111.860000	22.267857	284.615385	210.000000	70.000000	3000.000000	220.000000	0
...
1018	2.271346	1952.087902	912.855545	86.992183	20.123249	324.774576	209.198700	73.090961	2387.292495	125.007669	90
1019	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401	350.660830	72.920827	2360.392784	117.730099	90
1020	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047	740.142791	74.734344	2662.906040	236.606764	90
1021	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840	641.468152	74.042708	2071.715856	197.126067	90
1022	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708	758.747882	74.309704	2856.328932	194.754342	90

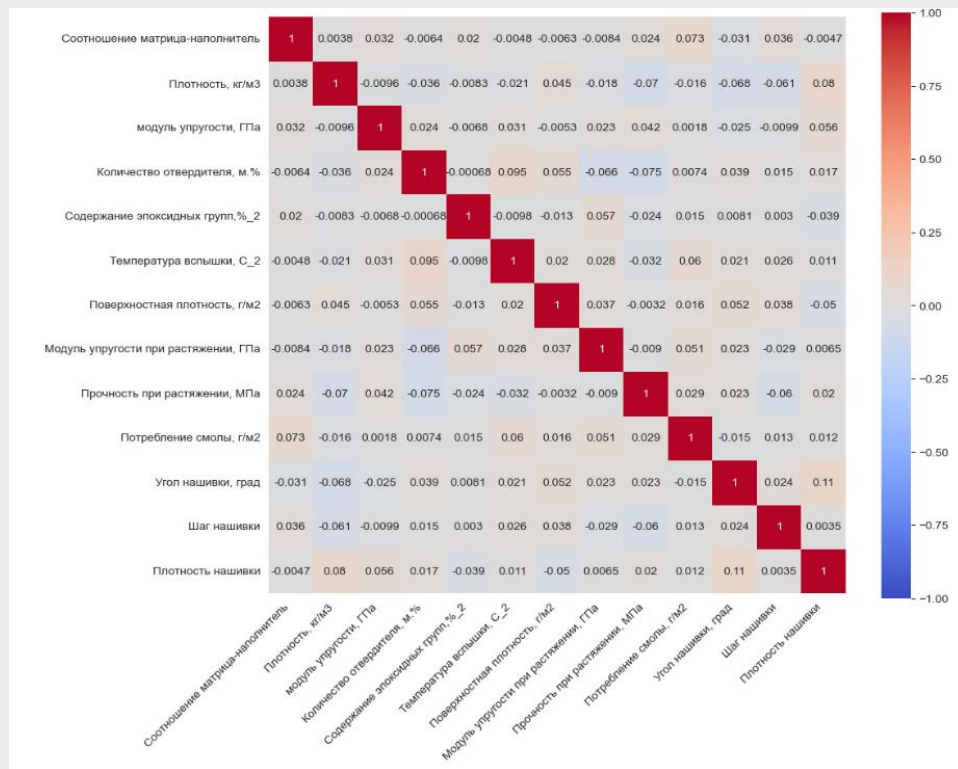
1023 rows x 13 columns

```
df3.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Соотношение матрица-наполнитель	1023.0	2.930366	0.913222	0.389403	2.317887	2.906878	3.552660	5.591742
Плотность, кг/м3	1023.0	1975.734888	73.729231	1731.764635	1924.155467	1977.621657	2021.374375	2207.773481
модуль упругости, ГПа	1023.0	739.923233	330.231581	2.436909	500.047452	739.664328	961.812526	1911.536477
Количество отвердителя, м.%	1023.0	110.570769	28.295911	17.740275	92.443497	110.564840	129.730366	198.953207
Содержание эпоксидных групп,%_2	1023.0	22.244390	2.406301	14.254985	20.608034	22.230744	23.961934	33.000000
Температура вспышки, С_2	1023.0	285.882151	40.943260	100.000000	259.066528	285.896812	313.002106	413.273418
Поверхностная плотность, г/м2	1023.0	482.731833	281.314690	0.603740	266.816645	451.864365	693.225017	1399.542362
Модуль упругости при растяжении, ГПа	1023.0	73.328571	3.118983	64.054061	71.245018	73.268805	75.356612	82.682051
Прочность при растяжении, МПа	1023.0	2466.922843	485.628006	1036.856605	2135.850448	2459.524526	2767.193119	3848.436732
Потребление смолы, г/м2	1023.0	218.423144	59.735931	33.803026	179.627520	219.198882	257.481724	414.590628
Угол нашивки, град	1023.0	44.252199	45.015793	0.000000	0.000000	0.000000	90.000000	90.000000
Шаг нашивки	1023.0	6.899222	2.563467	0.000000	5.080033	6.916144	8.586293	14.440522
Плотность нашивки	1023.0	57.153929	12.350969	0.000000	49.799212	57.341920	64.944961	103.988901

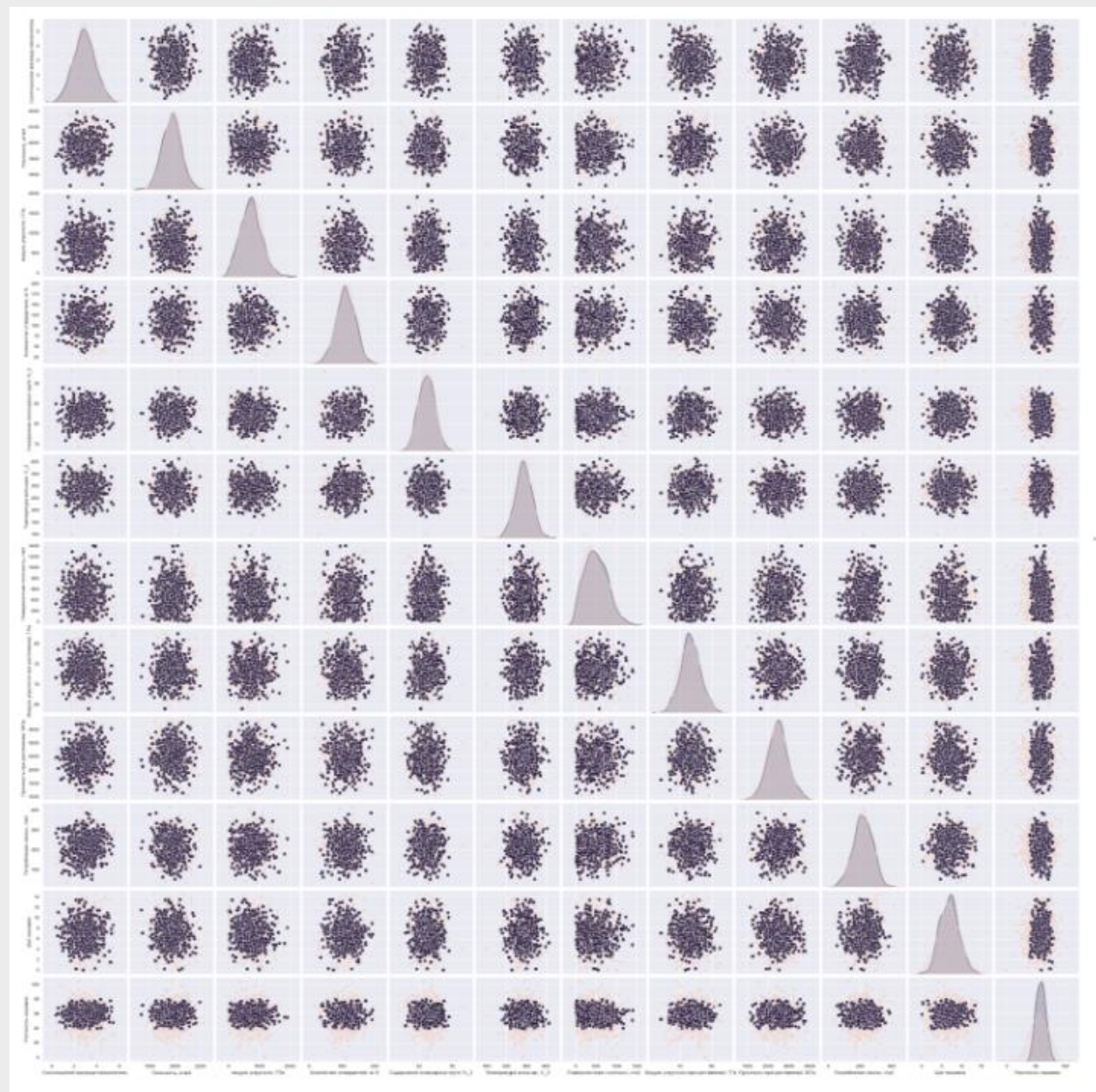
Гистограммы распределения переменных. Ящики с усами





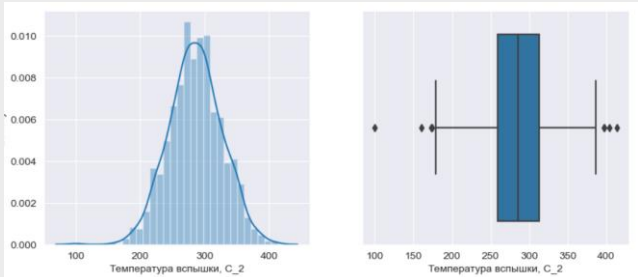
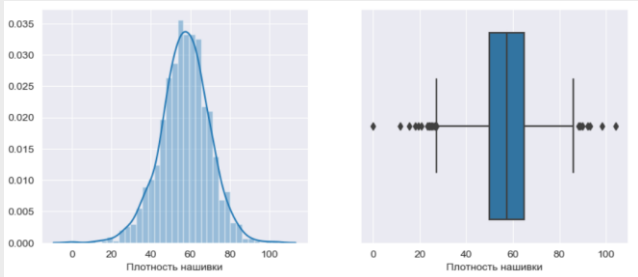
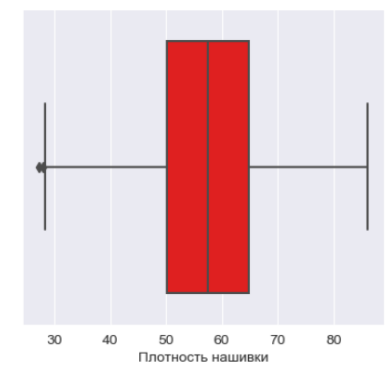
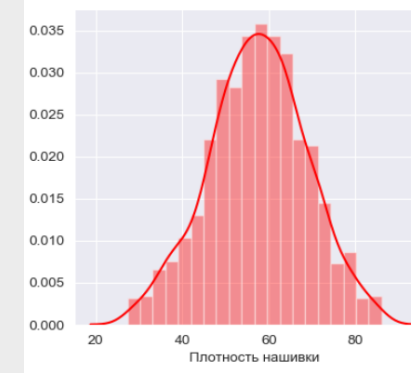
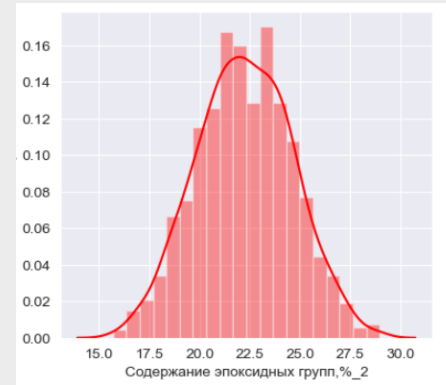
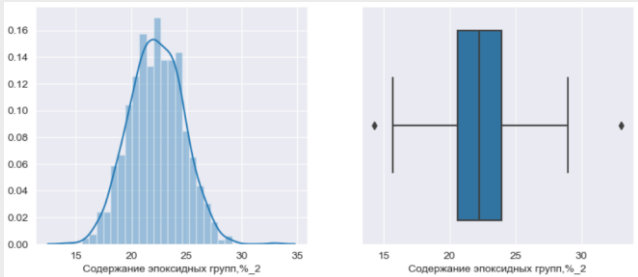
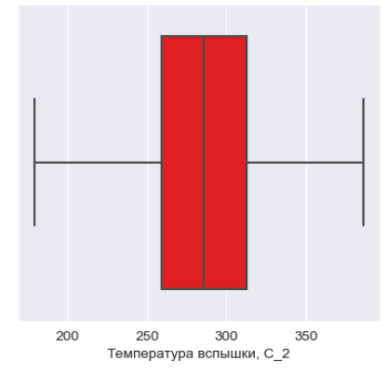
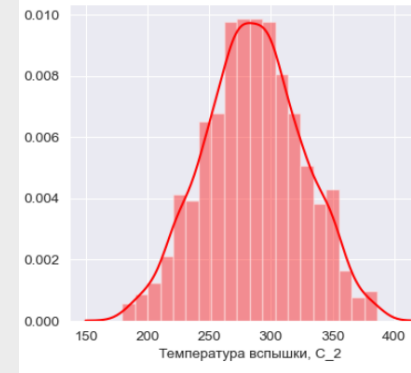
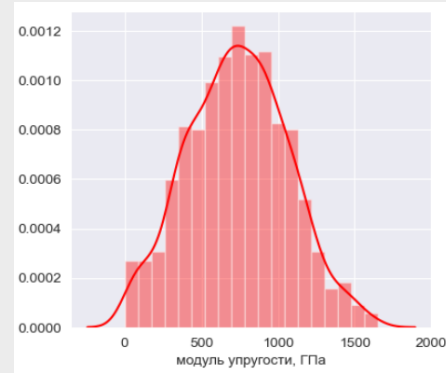
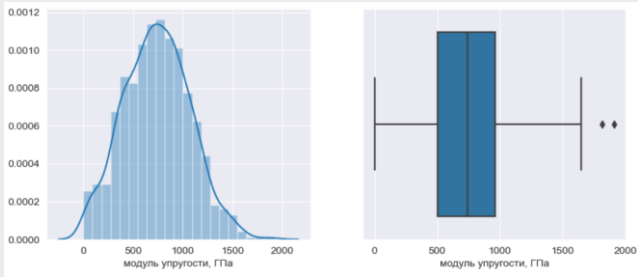
Тепловая карта корреляции

1. Соотношение матрица-наполнитель:	mean = 2.93 ,	median = 2.91
2. Плотность, кг/м3:	mean = 1975.73 ,	median = 1977.62
3. Модуль упругости, ГПа:	mean = 739.92 ,	median = 739.66
4. Количество отвердителя, м.%:	mean = 110.57 ,	median = 110.56
5. Содержание эпоксидных групп,%_2:	mean = 22.24 ,	median = 22.23
6. Температура вспышки, С_2:	mean = 285.88 ,	median = 285.9
7. Поверхностная плотность, г/м2:	mean = 482.73 ,	median = 451.86
8. Модуль упругости при растяжении, ГПа:	mean = 73.33 ,	median = 73.27
9. Прочность при растяжении, МПа:	mean = 2466.92 ,	median = 2459.52
10. Потребление смолы, г/м2:	mean = 218.42 ,	median = 219.2
11. Шаг нашивки:	mean = 6.9 ,	median = 6.92
12. Плотность нашивки:	mean = 57.15 ,	median = 57.34



Попарные графики рассеяния точек

Удаление выбросов



В нашем датасете после удаления выбросов осталось 990 строк.
Этого достаточно для дальнейшей работы.
С выбросами мы удалили всего 4% данных.

Нормализация данных

```
minmax_scaler = MinMaxScaler()

# применяем нормализацию
df_norm = minmax_scaler.fit_transform(np.array(df7))
df_norm

array([[0.28213084, 0.62653324, 0.44706097, ..., 0.27510888,
        0.55715613],
       [0.28213084, 0.62653324, 0.44706097, ..., 0.34453943,
        0.33583998],
       [0.45785722, 0.62653324, 0.45572116, ..., 0.34453943,
        0.50608317],
       ...,
       [0.55575038, 0.50547008, 0.25161199, ..., 1.28629789,
        0.68704631],
       [0.63739572, 0.70384225, 0.44872381, ..., 1.43571567,
        0.5275521 ],
       [0.65713085, 0.33328967, 0.25190326, ..., 1.41944815,
        0.85396608]])
```

df3.describe().T								
	count	mean	std	min	25%	50%	75%	max
Соотношение матрица-наполнитель	1023.0	2.930366	0.913222	0.389403	2.317887	2.906878	3.552660	5.591742
Плотность, кг/м3	1023.0	1975.734888	73.729231	1731.764635	1924.155467	1977.621657	2021.374375	2207.773481
модуль упругости, ГПа	1023.0	739.923233	330.231581	2.436909	500.047452	739.664328	961.812526	1911.536477
Количество отвердителя, м.%	1023.0	110.570769	28.295911	17.740275	92.443497	110.564840	129.730366	198.953207
Содержание эпоксидных групп,%_2	1023.0	22.244390	2.406301	14.254985	20.608034	22.230744	23.961934	33.000000
Температура вспышки, С_2	1023.0	285.882151	40.943260	100.000000	259.066528	285.896812	313.002106	413.273418
Поверхностная плотность, г/м2	1023.0	482.731833	281.314690	0.603740	266.816645	451.864365	693.225017	1399.542362
Модуль упругости при растяжении, ГПа	1023.0	73.328571	3.118983	64.054061	71.245018	73.268805	75.356612	82.682051
Прочность при растяжении, МПа	1023.0	2466.922843	485.628006	1036.856605	2135.850448	2459.524526	2767.193119	3848.436732
Потребление смолы, г/м2	1023.0	218.423144	59.735931	33.803026	179.627520	219.198882	257.481724	414.590628
Угол нашивки, град	1023.0	44.252199	45.015793	0.000000	0.000000	0.000000	90.000000	90.000000
Шаг нашивки	1023.0	6.899222	2.563467	0.000000	5.080033	6.916144	8.586293	14.440522
Плотность нашивки	1023.0	57.153929	12.350969	0.000000	49.799212	57.341920	64.944961	103.988901

	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
0	0.282131	0.626533	0.447061	0.178021	0.607435	0.509164	0.149682	0.319194	0.698235	0.488979	0.0
1	0.282131	0.626533	0.447061	0.613972	0.418887	0.583596	0.149682	0.319194	0.698235	0.488979	0.0
2	0.457857	0.626533	0.455721	0.519387	0.495653	0.509164	0.149682	0.319194	0.698235	0.488979	0.0
3	0.457201	0.563509	0.452685	0.519387	0.495653	0.509164	0.149682	0.319194	0.698235	0.488979	0.0
4	0.419084	0.374437	0.488508	0.519387	0.495653	0.509164	0.149682	0.319194	0.698235	0.488979	0.0
...
985	0.361750	0.462855	0.552781	0.382158	0.333908	0.703458	0.149109	0.485125	0.480312	0.239516	1.0
986	0.587163	0.668737	0.268550	0.707685	0.294428	0.362087	0.250230	0.475992	0.470745	0.220404	1.0
987	0.555750	0.505470	0.251612	0.512067	0.623085	0.334063	0.528643	0.573346	0.578340	0.532590	1.0
988	0.637396	0.703842	0.448724	0.682389	0.267818	0.466417	0.458108	0.536217	0.368070	0.428909	1.0
989	0.657131	0.333290	0.251903	0.614984	0.888354	0.588206	0.541942	0.550550	0.647135	0.422680	1.0

990 rows × 13 columns

В датасете есть значения, которые исчисляются в единицах, а есть значения, исчисляемые в тысячах. Чтобы с ними было удобно работать, нужно провести нормализацию — привести различные данные в самых разных единицах измерения и диапазонах значений к единому виду, который позволит сравнивать их между собой или использовать для расчёта схожести объектов.

	count	mean	std	min	25%	50%	75%	max
Соотношение матрица-наполнитель	990.0	0.490171	0.174705	0.0	0.372206	0.484780	0.609422	1.0
Плотность, кг/м3	990.0	0.512143	0.155353	0.0	0.403171	0.516539	0.608290	1.0
модуль упругости, ГПа	990.0	0.446680	0.198786	0.0	0.303017	0.448041	0.579752	1.0
Количество отвердителя, м.%	990.0	0.514348	0.155730	0.0	0.412869	0.514198	0.620400	1.0
Содержание эпоксидных групп,%_2	990.0	0.492322	0.179336	0.0	0.368579	0.491612	0.623428	1.0
Температура вспышки, С_2	990.0	0.515877	0.191180	0.0	0.386228	0.515628	0.646553	1.0
Поверхностная плотность, г/м2	990.0	0.343991	0.201677	0.0	0.189017	0.322631	0.495412	1.0
Модуль упругости при растяжении, ГПа	990.0	0.497039	0.166423	0.0	0.388086	0.493533	0.604256	1.0
Прочность при растяжении, МПа	990.0	0.508326	0.173281	0.0	0.390869	0.504740	0.614477	1.0
Потребление смолы, г/м2	990.0	0.484999	0.156819	0.0	0.383197	0.486941	0.587429	1.0
Угол нашивки, град	990.0	0.502020	0.500249	0.0	0.000000	1.000000	1.000000	1.0
Шаг нашивки	990.0	0.477793	0.178097	0.0	0.350214	0.479096	0.594753	1.0
Плотность нашивки	990.0	0.513352	0.191836	0.0	0.389191	0.515188	0.639569	1.0

Модели машинного обучения

Необходимо обучить нескольких моделей для прогноза модуля упругости при растяжении и прочности при растяжении. При построении модели необходимо 30% данных оставить на тестирование модели, на остальных происходит обучение моделей. При построении моделей проведем поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10.

Гиперпараметры - это все параметры, которые могут быть произвольно установлены перед началом обучения модели.

#Разбиваем данные на обучающую и тестовую выборки

```
# Датасет для обучения моделей для предсказания модуля упругости при растяжении  
X1 = df_norm_df.drop(['Модуль упругости при растяжении, ГПа'], axis = 1)  
# целевая переменная для предсказания модуля упругости при растяжении  
y1 = df_norm_df[['Модуль упругости при растяжении, ГПа']]
```

```
# Датасет для обучения моделей для предсказания прочности при растяжении  
X2 = df_norm_df.drop(['Прочность при растяжении, МПа'], axis = 1)  
# целевая переменная для предсказания прочности при растяжении  
y2 = df_norm_df[['Прочность при растяжении, МПа']]
```

#30% данных оставим на тестирование модели, на остальных проведем обучение моделей

```
X1_train, X1_test, y1_train, y1_test = train_test_split( X1, y1, test_size = 0.3, random_state = 42)  
X2_train, X2_test, y2_train, y2_test = train_test_split( X2, y2, test_size = 0.3, random_state = 42)
```

```
# проверим размерности наших выборок  
print(X1_train.shape)  
print(X2_train.shape)  
print(X1_test.shape)  
print(X2_test.shape)
```

```
(693, 12)  
(693, 12)  
(297, 12)  
(297, 12)
```

```
print(y1_train.shape)  
print(y2_train.shape)  
print(y1_test.shape)  
print(y2_test.shape)
```

```
(693, 1)  
(693, 1)  
(297, 1)  
(297, 1)
```

1. Линейная регрессия

```
# создание и обучение модели Линейной регрессии для модуля упругости при растяжении
LR_model1 = LinearRegression()
```

```
# поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
LR_model1_params = {
    'fit_intercept': ['True', 'False']
}
GSCV_LR_model1 = GridSearchCV(LR_model1, LR_model1_params, n_jobs=-1, cv=10)
GSCV_LR_model1.fit(X1_train, y1_train)
print('Лучшие параметры LinearRegression для предсказания модуля упругости при растяжении: ')
GSCV_LR_model1.best_params_
```

Лучшие параметры LinearRegression для предсказания модуля упругости при растяжении:

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

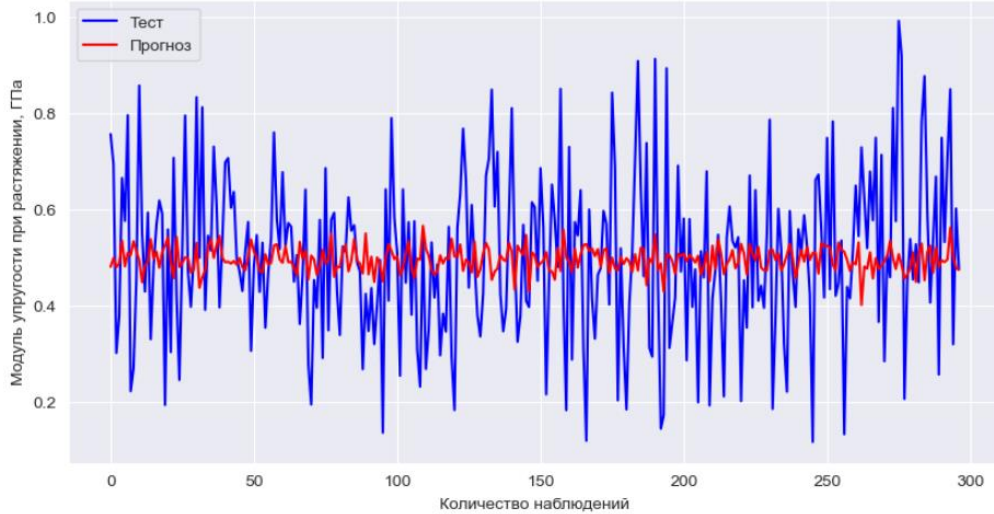
```
LR_model1_upr = GSCV_LR_model1.best_estimator_
# предсказанные значения нашей модели
y1_pred = LR_model1_upr.predict(X1_test)
```

```
# сравним точность модели с помощью метрик MAE, MSE и RMSE
print('Точность модели Линейной регрессии для предсказания модуля упругости при растяжении составляет: \nMAE ',
      mae(y1_test, y1_pred),
      '\nMSE ', mse(y1_test, y1_pred),
      '\nRMSE ', sqrt(mse(y1_test, y1_pred)),
      '\nr2 ', r2_score(y1_test, y1_pred))
```

Точность модели Линейной регрессии для предсказания модуля упругости при растяжении составляет:

```
MAE    0.13638454242788745
MSE    0.029612529068892425
RMSE    0.17208291335543
r2     -0.012292650952422157
```

Тестовые и прогнозные значения LinearRegression



```
# создание и обучение модели Линейной регрессии для прочности при растяжении
LR_model2 = LinearRegression()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
LR_model2_params = {
    'fit_intercept': ['True', 'False']
}
GSCV_LR_model2 = GridSearchCV(LR_model2, LR_model2_params, n_jobs=-1, cv=10)
GSCV_LR_model2.fit(X2_train, y2_train)
print('Лучшие параметры LinearRegression для предсказания прочности при растяжении: ')
GSCV_LR_model2.best_params_
```

Лучшие параметры LinearRegression для предсказания прочности при растяжении:

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

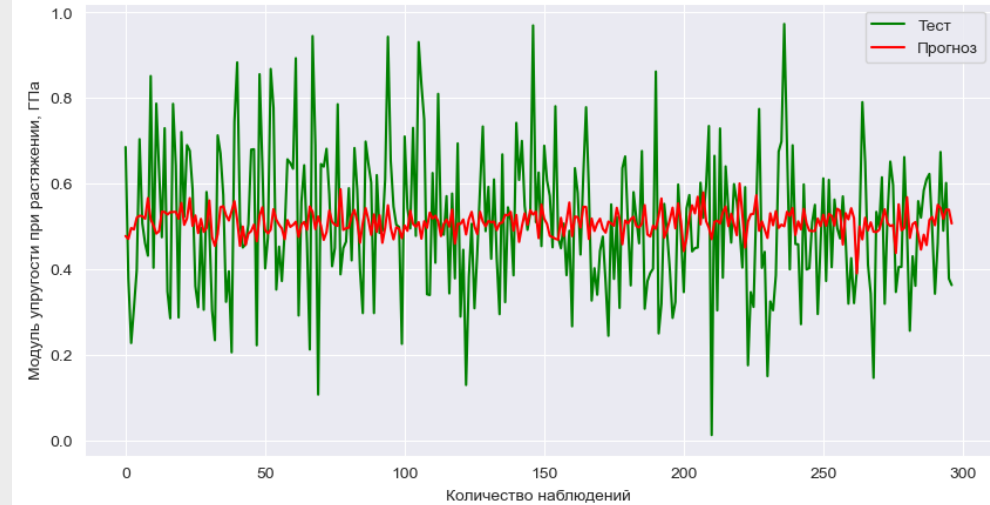
```
LR_model2_pro = GSCV_LR_model2.best_estimator_
# предсказанные значения нашей модели
y2_pred = LR_model2_pro.predict(X2_test)
```

```
# сравним точность модели с помощью метрик MAE, MSE и RMSE
print('Точность модели Линейной регрессии для предсказания прочности при растяжении составляет: \nMAE ',
      mae(y2_test, y2_pred),
      '\nMSE ', mse(y2_test, y2_pred),
      '\nRMSE ', sqrt(mse(y2_test, y2_pred)),
      '\nr2 ', r2_score(y2_test, y2_pred))
```

Точность модели Линейной регрессии для предсказания прочности при растяжении составляет:

```
MAE    0.13622058094797934
MSE    0.028019234151330532
RMSE    0.16738946846002747
r2     0.001458601420238237
```

Тестовые и прогнозные значения LinearRegression



2. Дерево решений

```
# создание и обучение модели DecisionTreeRegressor для модуля упругости при растяжении
tree1 = DecisionTreeRegressor()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
tree1_params = {
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'],
    'splitter': ['best', 'random'],
    'max_depth': range(1, 20, 2),
    'min_samples_split': range(10, 100, 5),
    'min_samples_leaf': range(1, 200, 50),
    'max_features': ['auto', 'sqrt', 'log2']
}
GSCV_tree1 = GridSearchCV(tree1, tree1_params, n_jobs=-1, cv=10)
GSCV_tree1.fit(X1_train, y1_train)
print('Лучшие параметры DecisionTreeRegressor для предсказания модуля упругости при растяжении: ')
GSCV_tree1.best_params_
```

Лучшие параметры DecisionTreeRegressor для предсказания модуля упругости при растяжении:

```
{'criterion': 'friedman_mse',
 'max_depth': 9,
 'max_features': 'auto',
 'min_samples_leaf': 101,
 'min_samples_split': 85,
 'splitter': 'random'}
```

```
tree1_upr = GSCV_tree1.best_estimator_
# предсказанные значения нашей модели
y3_pred = tree1_upr.predict(X1_test)
```

```
# сравним точность модели с помощью метрик MAE, MSE и RMSE
print('Точность модели DecisionTreeRegressor для предсказания модуля упругости при растяжении составляет: \nMAE - ',
      'mae(y1_test, y3_pred).round(3),
      '\nMSE - ', mse(y1_test, y3_pred).round(3),
      '\nRMSE - ', sqrt(mse(y1_test, y3_pred).round(3)),
      '\nr2 - ', r2_score(y1_test, y3_pred))
```

Точность модели DecisionTreeRegressor для предсказания модуля упругости при растяжении составляет:

```
MAE - 0.135
MSE - 0.029
RMSE - 0.17029386365926402
r2 - 0.00762113222736871
```



```
# создание и обучение модели DecisionTreeRegressor для прочности при растяжении
tree2 = DecisionTreeRegressor()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
tree2_params = {
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'],
    'splitter': ['best', 'random'],
    'max_depth': range(1, 20, 2),
    'min_samples_split': range(10, 100, 5),
    'min_samples_leaf': range(1, 200, 50),
    'max_features': ['auto', 'sqrt', 'log2']
}
GSCV_tree2 = GridSearchCV(tree2, tree2_params, n_jobs=-1, cv=10)
GSCV_tree2.fit(X2_train, y2_train)
print('Лучшие параметры DecisionTreeRegressor для предсказания прочности при растяжении:')
GSCV_tree2.best_params_
```

Лучшие параметры DecisionTreeRegressor для предсказания прочности при растяжении:

```
{'criterion': 'friedman_mse',
 'max_depth': 13,
 'max_features': 'log2',
 'min_samples_leaf': 101,
 'min_samples_split': 80,
 'splitter': 'best'}
```

```
tree2_pro = GSCV_tree2.best_estimator_
# предсказанные значения нашей модели
y4_pred = tree2_pro.predict(X2_test)
```

```
print('Точность модели DecisionTreeRegressor для предсказания прочности при растяжении составляет: \nMAE - ',
      'mae(y2_test, y4_pred).round(3),
      '\nMSE - ', mse(y2_test, y4_pred).round(3),
      '\nRMSE - ', sqrt(mse(y2_test, y4_pred).round(3)),
      '\nr2 - ', r2_score(y2_test, y4_pred))
```

Точность модели DecisionTreeRegressor для предсказания прочности при растяжении составляет:

```
MAE - 0.138
MSE - 0.028
RMSE - 0.1673320053068151
r2 - 0.005559716162631512
```



3. Метод ближайших соседей

```
# создание и обучение модели KNeighborsRegressor для модуля упругости при растяжении
knr1 = KNeighborsRegressor()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
knr1_params = {
    'n_neighbors': range(1, 100, 2),
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}
GSCV_knr1 = GridSearchCV(knr1, knr1_params, n_jobs=-1, cv=10)
GSCV_knr1.fit(X1_train, y1_train)
print('Лучшие параметры KNeighborsRegressor для предсказания модуля упругости при растяжении:')
GSCV_knr1.best_params_
```

Лучшие параметры KNeighborsRegressor для предсказания модуля упругости при растяжении:

```
{'algorithm': 'auto', 'n_neighbors': 77, 'weights': 'uniform'}
```

```
knr1_upr = GSCV_knr1.best_estimator_
# предсказанные значения нашей модели
y5_pred = knr1_upr.predict(X1_test)
```

```
print('Точность модели KNeighborsRegressor для предсказания модуля упругости при растяжении составляет: \nMAE ',
      mae(y1_test, y5_pred).round(3),
      '\nMSE ', mse(y1_test, y5_pred).round(3),
      '\nRMSE ', sqrt(mse(y1_test, y5_pred).round(3)),
      '\nr2 ', r2_score(y1_test, y5_pred))
```

Точность модели KNeighborsRegressor для предсказания модуля упругости при растяжении составляет:

```
MAE  0.136
MSE  0.029
RMSE  0.17029386365926402
r2    0.004487509611533103
```



```
# создание и обучение модели KNeighborsRegressor для прочности при растяжении
knr2 = KNeighborsRegressor()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
knr2_params = {
    'n_neighbors': range(1, 100, 2),
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}
GSCV_knr2 = GridSearchCV(knr2, knr2_params, n_jobs=-1, cv=10)
GSCV_knr2.fit(X2_train, y2_train)
print('Лучшие параметры KNeighborsRegressor для предсказания прочности при растяжении:')
GSCV_knr2.best_params_
```

Лучшие параметры KNeighborsRegressor для предсказания прочности при растяжении:

```
{'algorithm': 'auto', 'n_neighbors': 97, 'weights': 'uniform'}
```

```
knr2_pro = GSCV_knr2.best_estimator_
# предсказанные значения нашей модели
y6_pred = knr2_pro.predict(X2_test)
```

```
print('Точность модели KNeighborsRegressor для предсказания прочности при растяжении составляет:\nMAE ',
      mae(y2_test, y6_pred).round(3),
      '\nMSE ', mse(y2_test, y6_pred).round(3),
      '\nRMSE ', sqrt(mse(y2_test, y6_pred).round(3)),
      '\nr2 ', r2_score(y2_test, y6_pred))
```

Точность модели KNeighborsRegressor для предсказания прочности при растяжении составляет:

```
MAE  0.134
MSE  0.028
RMSE  0.1673320053068151
r2    0.0072424437070057746
```



4. Метод случайного леса - Random Forest Regressor

```
# создание и обучение модели Random Forest Regressor для модуля упругости при растяжении
rfr1 = RandomForestRegressor()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
rfr1_params = {
    'n_estimators': range(1, 500, 10),
    'criterion': ['squared_error', 'absolute_error', 'poisson'],
    'max_depth': range(1, 8),
    'min_samples_split': range(10, 50, 5),
    'min_samples_leaf': range(2, 9),
    'bootstrap': ['True', 'False'],
    'oob_score': ['True', 'False'],
    'warm_start': ['True', 'False']
}
GSCV_rfr1 = RandomizedSearchCV(rfr1, rfr1_params, n_jobs=-1, cv=10)
GSCV_rfr1.fit(X1_train, np.ravel(y1_train))
print('Лучшие параметры RandomForestRegressor для предсказания модуля упругости при растяжении: ')
GSCV_rfr1.best_params_
```

Лучшие параметры RandomForestRegressor для предсказания модуля упругости при растяжении:

```
{'warm_start': 'False',
 'oob_score': 'False',
 'n_estimators': 451,
 'min_samples_split': 30,
 'min_samples_leaf': 2,
 'max_depth': 2,
 'criterion': 'absolute_error',
 'bootstrap': 'False'}
```

```
rfr1_upr = GSCV_rfr1.best_estimator_
# предсказанные значения нашей модели
y7_pred = rfr1_upr.predict(X1_test)
```

```
print('Точность модели RandomForestRegressor для предсказания модуля упругости при растяжении составляет: \nMAE ',
      mae(y1_test, y7_pred).round(3),
      '\nMSE ', mse(y1_test, y7_pred).round(3),
      '\nRMSE ', sqrt(mse(y1_test, y7_pred).round(3)),
      '\nr2 ', r2_score(y1_test, y7_pred))
```

Точность модели RandomForestRegressor для предсказания модуля упругости при растяжении составляет:
MAE 0.137
MSE 0.03
RMSE 0.17320508075688773
r2 -0.013782187128413392



```
# создание и обучение модели RandomForestRegressor для прочности при растяжении
rfr2 = RandomForestRegressor()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
rfr2_params = {
    'n_estimators': range(1, 4),
    'criterion': ['squared_error', 'absolute_error', 'poisson'],
    'max_depth': range(1, 4),
    'min_samples_split': range(1, 4),
    'min_samples_leaf': range(1, 4),
    'bootstrap': ['True', 'False'],
    'oob_score': ['True', 'False'],
    'warm_start': ['True', 'False']
}
GSCV_rfr2 = RandomizedSearchCV(rfr2, rfr2_params, n_jobs=-1, cv=10)
GSCV_rfr2.fit(X2_train, y2_train)
print('Лучшие параметры RandomForestRegressor для предсказания прочности при растяжении: ')
GSCV_rfr2.best_params_
```

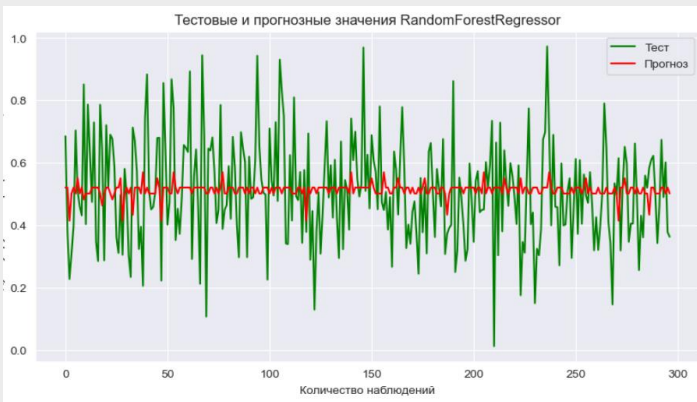
Лучшие параметры RandomForestRegressor для предсказания прочности при растяжении:

```
{'warm_start': 'True',
 'oob_score': 'False',
 'n_estimators': 3,
 'min_samples_split': 3,
 'min_samples_leaf': 1,
 'max_depth': 1,
 'criterion': 'squared_error',
 'bootstrap': 'False'}
```

```
rfr2_pro = GSCV_rfr2.best_estimator_
# предсказанные значения нашей модели
y8_pred = rfr2_pro.predict(X2_test)
```

```
print('Точность модели RandomForestRegressor для предсказания прочности при растяжении составляет: \nMAE ',
      mae(y2_test, y8_pred).round(3),
      '\nMSE ', mse(y2_test, y8_pred).round(3),
      '\nRMSE ', sqrt(mse(y2_test, y8_pred).round(3)),
      '\nr2 ', r2_score(y2_test, y8_pred))
```

Точность модели RandomForestRegressor для предсказания прочности при растяжении составляет:
MAE 0.134
MSE 0.028
RMSE 0.1673320053068151
r2 0.005193704024973433



5. Метод градиентного бустинга - Gradient Boosting Regressor

```
# создание и обучение модели Gradient Boosting Regressor для модуля упругости при растяжении
gbr1 = GradientBoostingRegressor()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
gbr1_params = {
    'n_estimators': range(1, 10),
    'loss': ['squared_error', 'absolute_error', 'huber', 'quantile'],
    'criterion': ['friedman_mse', 'squared_error', 'mse'],
    'max_depth': range(1, 5),
    'min_samples_split': range(1, 10),
    'min_samples_leaf': range(1, 10, 2),
    'max_features': range(1, 10)
}
GSCV_gbr1 = RandomizedSearchCV(gbr1, gbr1_params, n_jobs=-1, cv=10)
GSCV_gbr1.fit(X1_train, y1_train)
print('Лучшие параметры Gradient Boosting Regressor для предсказания модуля упругости при растяжении:')
GSCV_gbr1.best_params_
```

Лучшие параметры Gradient Boosting Regressor для предсказания модуля упругости при растяжении:

```
{'n_estimators': 7,
 'min_samples_split': 8,
 'min_samples_leaf': 5,
 'max_features': 2,
 'max_depth': 1,
 'loss': 'absolute_error',
 'criterion': 'mse'}
```

```
gbr1_upr = GSCV_gbr1.best_estimator_
# предсказанные значения нашей модели
y9_pred = gbr1_upr.predict(X1_test)
```

```
print('Точность модели Gradient Boosting Regressor для предсказания модуля упругости при растяжении составляет: \nMAE ',
      mae(y1_test, y9_pred).round(3),
      '\nMSE ', mse(y1_test, y9_pred).round(3),
      '\nRMSE ', sqrt(mse(y1_test, y9_pred).round(3)),
      '\nr2 ', r2_score(y1_test, y9_pred))
```

Точность модели Gradient Boosting Regressor для предсказания модуля упругости при растяжении составляет:

```
MAE    0.136
MSE     0.029
RMSE   0.17029386365926402
r2     -0.005486966182762787
```



```
# создание и обучение модели Gradient Boosting Regressor для прочности при растяжении
gbr2 = GradientBoostingRegressor()
```

```
#поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10
gbr2_params = {
    'n_estimators': range(1, 10),
    'loss': ['squared_error', 'absolute_error', 'huber', 'quantile'],
    'criterion': ['friedman_mse', 'squared_error', 'mse'],
    'max_depth': range(1, 5),
    'min_samples_split': range(1, 10),
    'min_samples_leaf': range(1, 10, 2),
    'max_features': range(1, 10)
}
GSCV_gbr2 = RandomizedSearchCV(gbr2, gbr2_params, n_jobs=-1, cv=10)
GSCV_gbr2.fit(X2_train, y2_train)
print('Лучшие параметры Gradient Boosting Regressor для предсказания прочности при растяжении:')
GSCV_gbr2.best_params_
```

Лучшие параметры Gradient Boosting Regressor для предсказания прочности при растяжении:

```
{'n_estimators': 6,
 'min_samples_split': 3,
 'min_samples_leaf': 9,
 'max_features': 1,
 'max_depth': 2,
 'loss': 'squared_error',
 'criterion': 'friedman_mse'}
```

```
gbr2_pro = GSCV_gbr2.best_estimator_
# предсказанные значения нашей модели
y10_pred = gbr2_pro.predict(X2_test)
```

```
print('Точность модели Gradient Boosting Regressor для предсказания прочности при растяжении составляет: \nMAE ',
      mae(y2_test, y10_pred).round(3),
      '\nMSE ', mse(y2_test, y10_pred).round(3),
      '\nRMSE ', sqrt(mse(y2_test, y10_pred).round(3)),
      '\nr2 ', r2_score(y2_test, y10_pred))
```

Точность модели Gradient Boosting Regressor для предсказания прочности при растяжении составляет:

```
MAE    0.135
MSE     0.028
RMSE   0.1673320053068151
r2     -0.00019076706941989485
```



Сравнение качества моделей

	Модель для модуля упругости при растяжении	MAE	MSE	Score	r2
2	KNeighborsRegressor	0.136126	0.029122	0.004488	0.004488
3	RandomForestRegressor	0.136587	0.029393	-0.004792	-0.004792
1	DecisionTree	0.135020	0.029476	-0.007621	-0.007621
4	GradientBoostingRegressor	0.136832	0.029477	-0.007663	-0.007663
0	LinearRegression	0.136385	0.029613	-0.012293	-0.012293

	Модель для прочности при растяжении	MAE	MSE	Score	r2
2	KNeighborsRegressor	0.134416	0.027857	0.007242	0.007242
4	GradientBoostingRegressor	0.135653	0.027990	0.002511	0.002511
0	LinearRegression	0.136221	0.028019	0.001459	0.001459
3	RandomForestRegressor	0.135000	0.028000	-0.013890	-0.013890
1	DecisionTree	0.138764	0.029548	-0.053019	-0.053019

Нейронная сеть для предсказания соотношения «Матрица-наполнитель»

1 вариант

```
# нормализуем данные для подачи в нейросеть
normalizer = tf.keras.layers.Normalization(axis=-1)
X_train_ns_norm = normalizer.adapt(np.array(X_train_ns))

# воспользуемся классом Sequential библиотеки Keras, который укажет,
# что мы задаём последовательно связанные между собой слои
model_mn = Sequential(X_train_ns_norm)
model_mn.add(Dense(128))
model_mn.add(BatchNormalization())
model_mn.add(LeakyReLU())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(128, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(64, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(32, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(LeakyReLU())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(16, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dense(1))
model_mn.add(Activation('sigmoid'))

# Настройка будет три:
# 1) тип функции потерь (loss function) определяет, как мы будем считать
# 2) способ или алгоритм оптимизации этой функции (optimizer) поможет
# 3) метрика (metric) покажет, насколько точна наша модель
model_mn.compile(
    optimizer='Adam',
    loss='mse', metrics = ['mae'])

history = model_mn.fit( X_train_ns,
                        y_train_ns,
                        epochs = 100,
                        verbose = 1,
                        validation_split = 0.2)
```



4.82

4.49

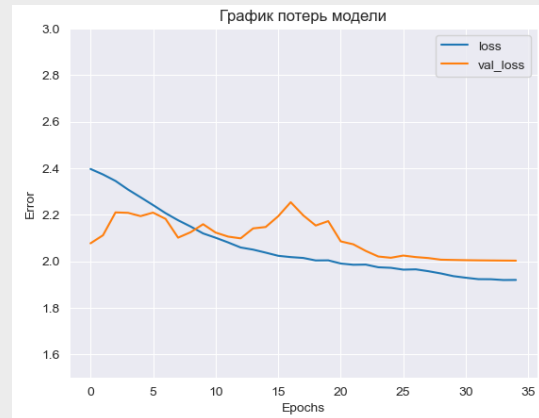
2 вариант

```
# увеличим количество слоев, нейронов и уменьшим кол
model_mn = Sequential(X_train_ns_norm)

model_mn.add(Dense(256))
model_mn.add(BatchNormalization())
model_mn.add(LeakyReLU())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(256, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(128))
model_mn.add(BatchNormalization())
model_mn.add(LeakyReLU())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(128, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(64, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(32, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(LeakyReLU())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(16, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dense(1))
model_mn.add(Activation('sigmoid'))
```

```
model_mn.compile(
    optimizer='Adam',
    loss='mean_absolute_error', metrics = ['mae'])
```

```
history = model_mn.fit( X_train_ns,
                        y_train_ns,
                        epochs = 35,
                        verbose = 1,
                        validation_split = 0.2)
```



2.00

1.93

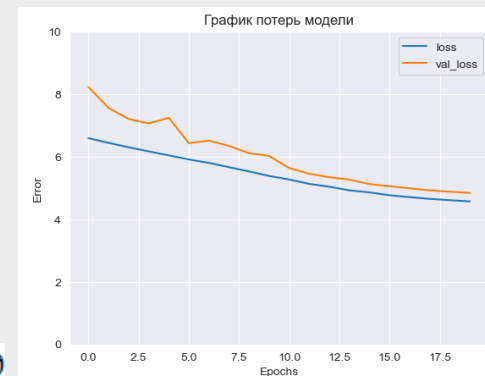
3 вариант

```
# уменьшим количество слоев, нейронов и эпох. Из
model_mn = Sequential(X_train_ns_norm)

model_mn.add(Dense(64, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(32, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(LeakyReLU())
model_mn.add(Dropout(0.18))
model_mn.add(Dense(16, activation='relu'))
model_mn.add(BatchNormalization())
model_mn.add(Dense(1))
model_mn.add(Activation('sigmoid'))

model_mn.compile(
    optimizer='rmsprop',
    loss='mse', metrics = ['mae'])

history = model_mn.fit( X_train_ns,
                        y_train_ns,
                        epochs = 20,
                        verbose = 1,
                        validation_split = 0.2)
```



4.94

4.60

4 вариант

```
# подадим в нейросеть данные после нормализации после MinMaxScaler
#определяем X y
y_ns = np.array(df_norm_df['Соотношение матрица-наполнитель'])
X_ns = np.array(df_norm_df.drop(['Соотношение матрица-наполнитель'], axis = 1))

#train_test_split
X_train_ns, X_test_ns, y_train_ns, y_test_ns = train_test_split(X_ns, y_ns,
                                                                test_size=0.3,
                                                                random_state = 1)

#архитектура модели
model_ns = Sequential()
model_ns.add(Dense(128, input_shape = (X_train_ns.shape[1],), activation = 'relu'))
model_ns.add(BatchNormalization())
model_ns.add(Dense(128, activation='relu'))
model_ns.add(Dropout(0.18))
model_ns.add(Dense(64, activation='relu'))
model_ns.add(Dropout(0.18))
model_ns.add(Dense(32, activation='relu'))
model_ns.add(Dense(16, activation='relu'))
model_ns.add(BatchNormalization())
model_ns.add(Dense(1, activation = 'sigmoid'))

#компиляция
model_ns.compile(
    optimizer='Adam',
    loss='mean_absolute_error', metrics = ['mae'])

history = model_ns.fit( X_train_ns,
                        y_train_ns,
                        epochs = 100,
                        verbose = 1,
                        validation_split = 0.2)
```



0.15

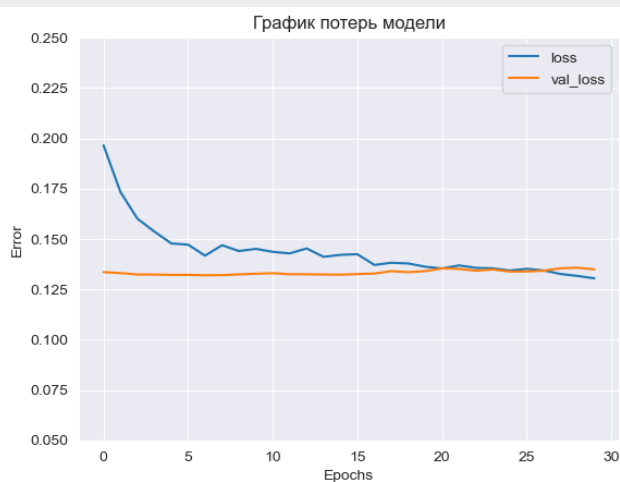
0.08

5 вариант

```
# увеличим количество слоев и нейронов и уменьшим количество эпох
#архитектура модели
model_ns = Sequential()
model_ns.add(Dense(256, input_shape = (X_train_ns.shape[1],), activation = 'relu'))
model_ns.add(BatchNormalization())
model_ns.add(Dense(128, activation='relu'))
model_ns.add(Dropout(0.18))
model_ns.add(Dense(128, activation='relu'))
model_ns.add(Dropout(0.18))
model_ns.add(Dense(64, activation='relu'))
model_ns.add(Dropout(0.18))
model_ns.add(Dense(64, activation='relu'))
model_ns.add(Dropout(0.18))
model_ns.add(Dense(32, activation='relu'))
model_ns.add(Dense(16, activation='relu'))
model_ns.add(BatchNormalization())
model_ns.add(Dense(1, activation = 'sigmoid'))

#компиляция
model_ns.compile(
    optimizer='Adam',
    loss='mean_absolute_error', metrics = ['mae'])

history = model_ns.fit( X_train_ns,
                        y_train_ns,
                        epochs = 30,
                        verbose = 1,
                        validation_split = 0.2)
```



0.14

0.13

Разработка Flask-приложения

```
app.py > main
1  import flask
2  from flask import render_template
3  import tensorflow as tf
4  from tensorflow import keras
5  import sklearn
6  import keras
7
8  app = flask.Flask(__name__, template_folder='templates')
9
10 @app.route('/', methods=['GET', 'POST'])
11 @app.route('/index', methods=['GET', 'POST'])
12
13 def main():
14     temp = 1
15     param_lst = []
16
17     if flask.request.method == 'GET':
18         return render_template('mn.html' )
19
20     if flask.request.method == 'POST':
21         loaded_model = keras.models.load_model("model_mn")
22         for i in range(1,13,1):
23
24
25
26             experience = flask.request.form.get(f'experience{i}')
27
28             param_lst.append(float(experience))
29
30         temp = loaded_model.predict([param_lst])
31         return render_template('mn.html', message = temp)
32
33 if __name__ == '__main__':
34     app.run()
```

```
(base) m1@WIN-67VNPTIGQOB:/mnt/c/Users/user/Desktop/Diplom/ML_FLASK$ conda activate ml_flask
(ml_flask) m1@WIN-67VNPTIGQOB:/mnt/c/Users/user/Desktop/Diplom/ML_FLASK$ python3 app.py
2022-12-19 00:01:30.031991: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary
is optimized for one or more CPU architectures: AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-12-19 00:01:30.930457: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:44]
Failed to find the library for loading OpenCL. This warning can be safely ignored. See
file or directory
2022-12-19 00:01:30.930584: I tensorflow/compiler/xla/stream_executor/cuda/cudart_stub.cc:29] Ignore
d
2022-12-19 00:01:34.424674: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:44]
Failed to find the library for loading OpenCL. This warning can be safely ignored. See
file or directory
2022-12-19 00:01:34.424822: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:44]
Failed to find the library for loading OpenCL. This warning can be safely ignored. See
file or directory
2022-12-19 00:01:34.424872: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning:
The following TensorRT features are not supported: No such file or directory
ies mentioned above are installed properly.
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production W
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
□
```

Код файла app.py и демонстрация запуска приложения из виртуальной среды в программе Visual Code Studio

Демонстрация работы приложения

Расчет соотношения матрица-наполнитель

Введите параметры

Введите Плотность, кг/м³

Введите Модуль упругости, ГПа

Введите Количество отвердителя, м.%

Введите Содержание эпоксидных групп, %

Введите Температура вспышки, С

Введите Поверхностная плотность, г/м²

Введите Модуль упругости при растяжении, ГПа

Введите Прочность при растяжении, МПа

Введите Потребление смолы, г/м²

Введите Угол нашивки, град

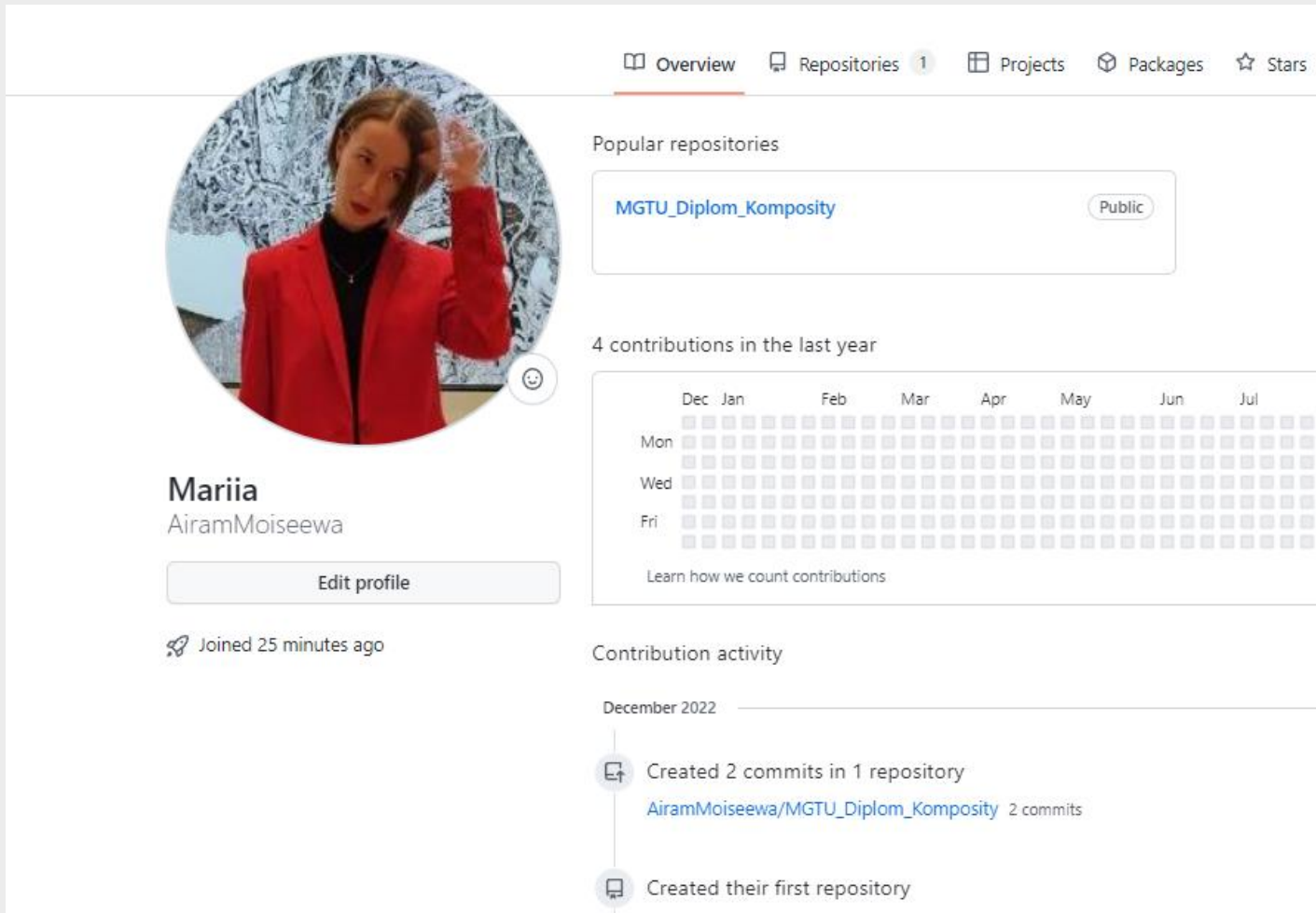
Введите Шаг нашивки

Введите Плотность нашивки

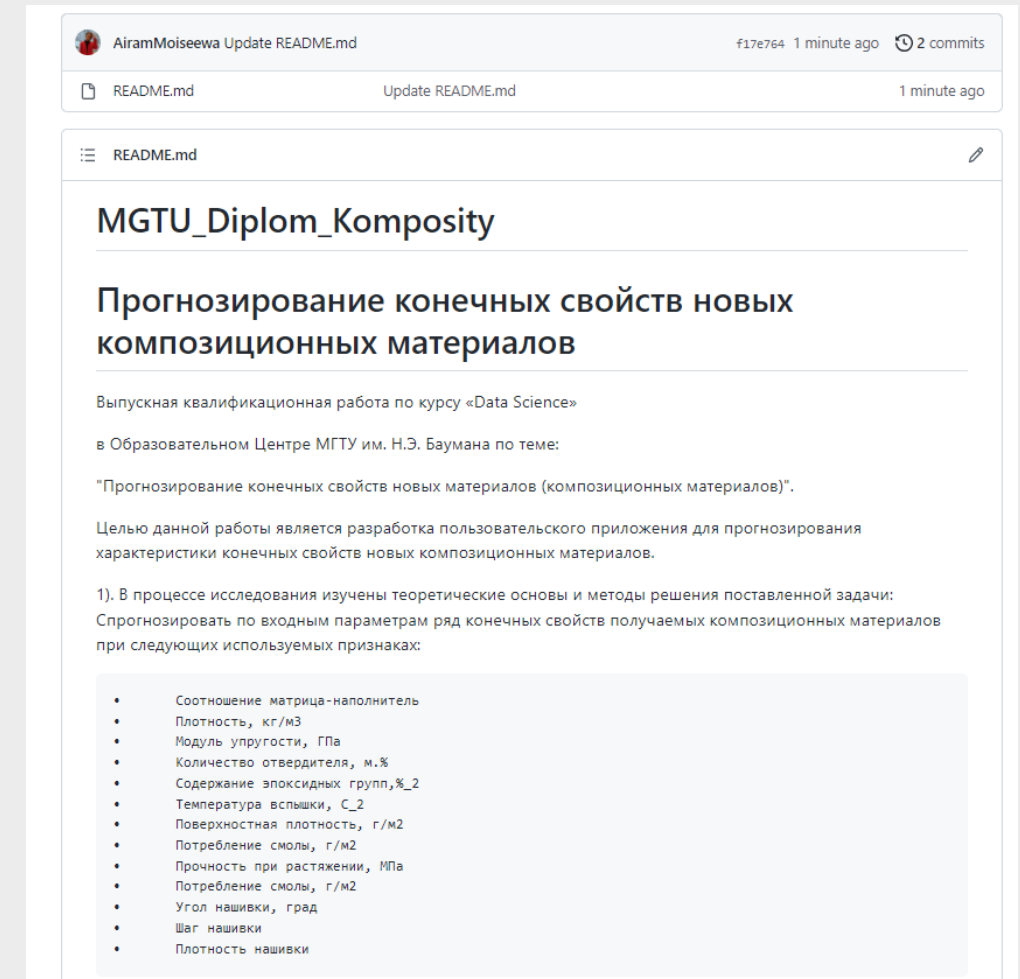
В открывшемся окне пользователю необходимо ввести в соответствующие ячейки требуемые значения и нажать на кнопку «Рассчитать». На выходе пользователь получает результат прогноза для значения параметра «Соотношение «матрица – наполнитель».

[[4.797463e-10]]

Создание репозитория на Github.com



The screenshot shows the GitHub profile of Mariia AiramMoiseewa. The profile includes a circular profile picture of a woman in a red jacket, the name 'Mariia AiramMoiseewa', and a bio. The navigation bar shows 'Overview' as the active tab, with 'Repositories' (1), 'Projects', 'Packages', and 'Stars' also visible. Under 'Popular repositories', the repository 'MGTU_Diplom_Komposity' is listed as 'Public'. A contribution graph shows '4 contributions in the last year' with a grid for the months Dec, Jan, Feb, Mar, Apr, May, Jun, and Jul. The 'Contribution activity' section for December 2022 shows two events: 'Created 2 commits in 1 repository' (AiramMoiseewa/MGTU_Diplom_Komposity, 2 commits) and 'Created their first repository'.



The screenshot shows the README.md file for the repository 'MGТУ_Diplom_Komposity'. The file content includes the title 'Прогнозирование конечных свойств новых композиционных материалов', a subtitle 'Выпускная квалификационная работа по курсу «Data Science»', and a description of the work. The README also lists the following parameters used for the analysis:

- Соотношение матрица-наполнитель
- Плотность, кг/м3
- Модуль упругости, ГПа
- Количество отвердителя, м.%
- Содержание эпоксидных групп, %_2
- Температура вспышки, C_2
- Поверхностная плотность, г/м2
- Потребление смолы, г/м2
- Прочность при растяжении, МПа
- Потребление смолы, г/м2
- Угол нашивки, град
- Шаг нашивки
- Плотность нашивки

- https://github.com/AiramMoiseewa/MGTU_Diplom_Komposity

Спасибо за внимание!



edu.bmstu.ru

+7 495 182-83-85

edu@bmstu.ru

Москва, Госпитальный переулок ,
д. 4-6, с.3