ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР *МГТУ им. Н. Э. Баумана*

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу

«Data Science»

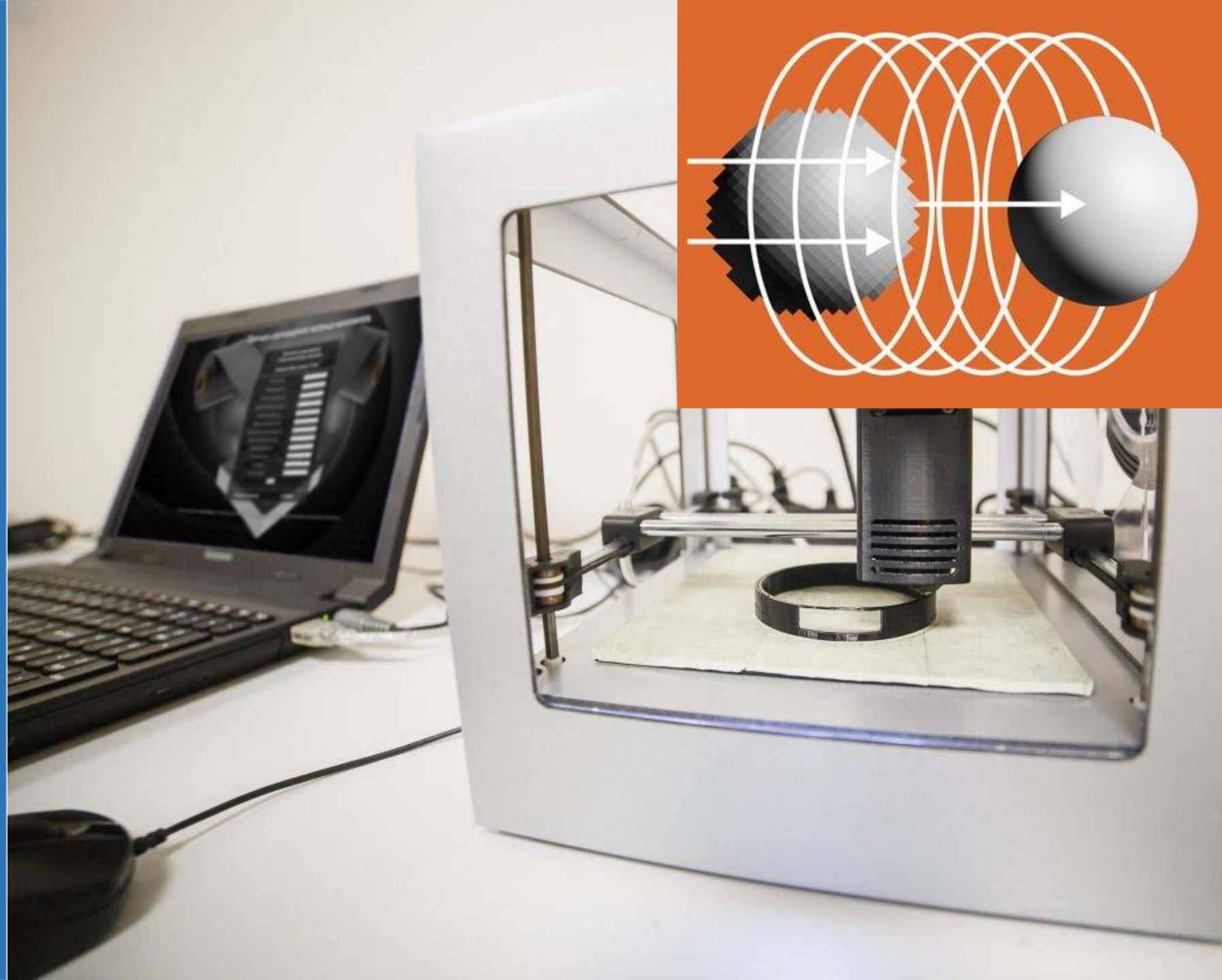
Слушатель: Глазунов Игорь Владимирович

Цели и задачи работы.

- Цель – прогнозирование отношения матрица-наполнитель в композиционном материале (в данной работе рассматривается базальтопластик);
- Задача №1 изучить теоретические вопросы работы и предоставленные таблицы с данными;
- Задача №2 построить лучшую модель данных для прогнозирования;
- Задача №3 обучить нейросеть которая способна прогнозировать отношение матрица-наполнитель с высокой вероятностью;
- Задача №4 написать пользовательское приложение для облегчения взаимодействия с моделью данных.

Актуальность работы.

- Работа позволяет на практике пройти все этапы работы с данными;
- Сами модели позволяют сократить количество реально проводимых испытаний и тем самым сэкономить время и другие ресурсы;
- Поиск и предсказание оптимальных параметров позволяет обнаружить новые направления исследований.



Начало работы:

✓ План работы:

1. Изучение теоретических основ и методов решения поставленной задачи;

2. Разведочный анализ предложенных данных;

3. Предобработка данных;

4. Разработка и обучение нескольких моделей для прогноза прочности при растяжении и модуля упругости при растяжении;

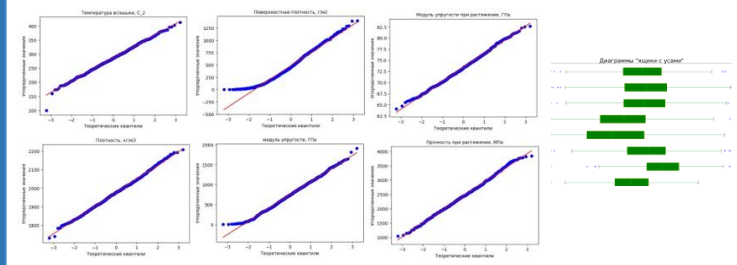
5. Разработка нейронной сети для рекомендации соотношения матрица-наполнитель;

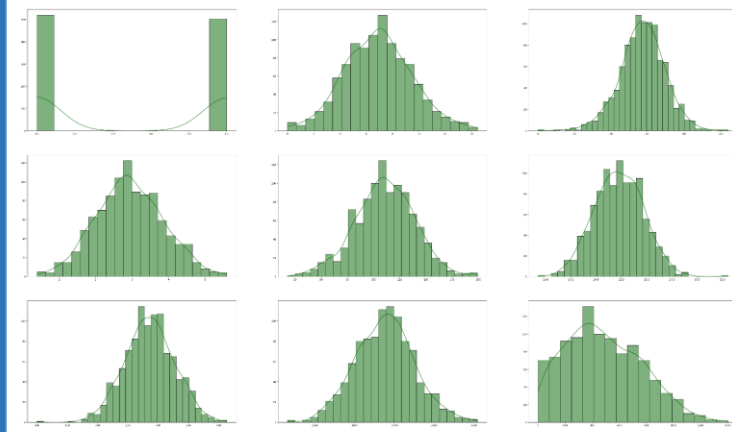
6. Разработка пользовательского приложения на базе Flask;

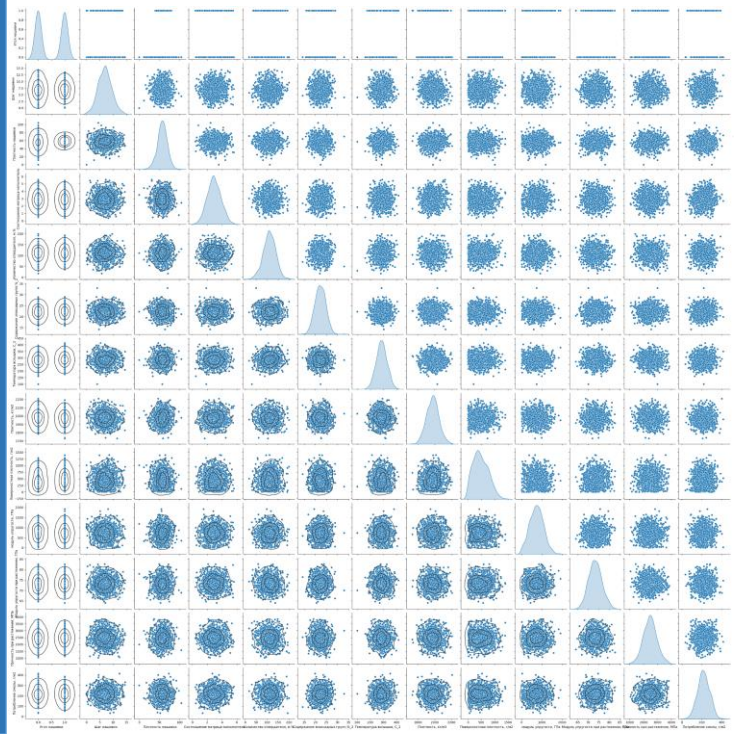
7. Создание репозитория на Github.

✓ Методы применённые в ходе работы:

- Работа с дата фреймами при помощи pandas.DataFrame;
- Построение графиков при помощи seaborn, plotly, matplotlib.pyplot;
- Предобработка данных при помощи sklearn.preprocessing: Normalizer, LabelEncoder, MinMaxScaler, StandardScaler

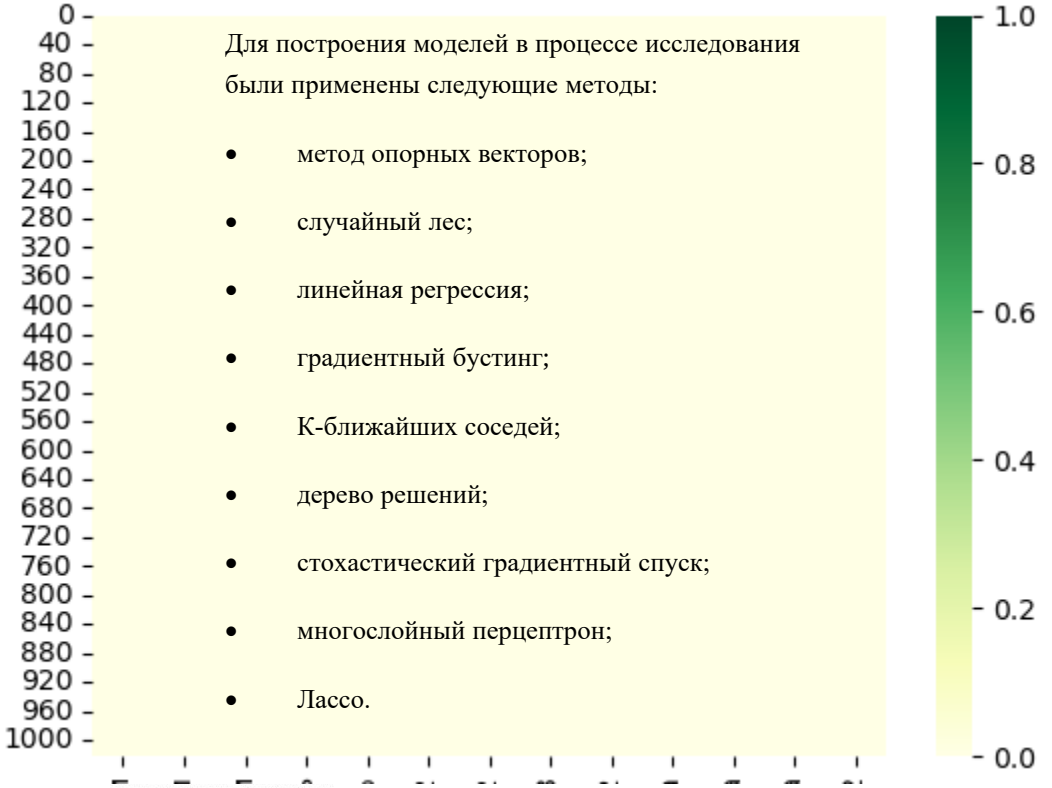


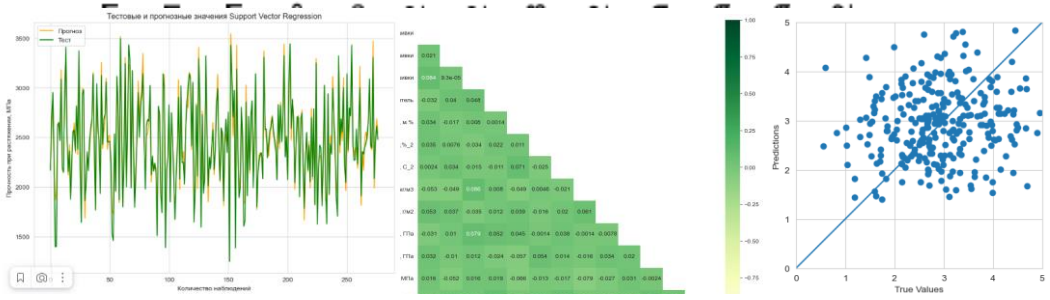


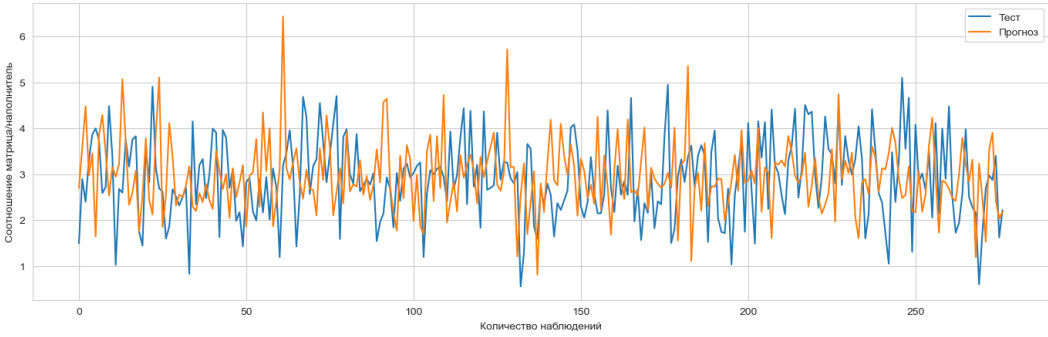


Для построения моделей в процессе исследования были применены следующие методы:

- метод опорных векторов;
- случайный лес;
- линейная регрессия;
- градиентный бустинг;
- К-ближайших соседей;
- дерево решений;
- стохастический градиентный спуск;
- многослойный перцептрон;
- Лассо.







ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Разведочный анализ предложенных данных

При загрузке и обработке датасетов, было произведено:

Объединение датасетов по методу INNER.

Изучение описательной статистики каждой переменной - среднее, медиана, стандартное отклонение, минимум, максимум, квартили.

Проверка данных и приведение их к удобному для работы виду (угол нашивки приведён к двоичному виду), Проверка на пропуски и дубликаты.

Построение гистограмм и «ящиков с усами». Удаление выбросов.

Разведочный анализ очищенного от выбросов объединённого датасета.:

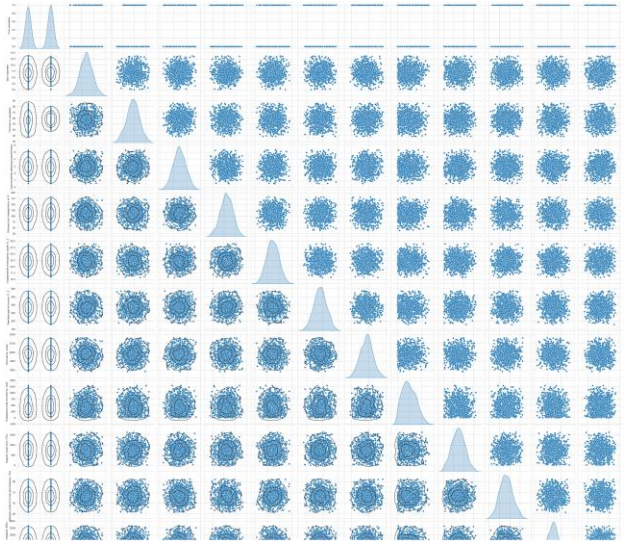
Среднее, медианное значение было рассмотрено отдельно для каждой колонки. Были построены гистограммы распределения и Диаграммы «ящиков с усами» (боксплоты). Были выведены попарные графики рассеяния точек (матрицы диаграмм рассеяния) и графики квантиль-квантиль. Зависимости между колонками не выявлено.

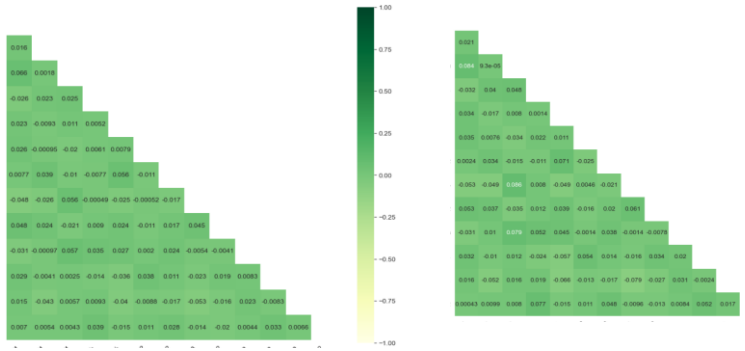
В ходе разведочного анализа явной корреляции не было выявлено. Это иллюстрируют корреляционные матрица в виде тепловой карты для коэффициента ранговой корреляции Кендалла и коэффициента корреляции Пирсона.


```
<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   int64
 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(10), int64(1)
memory usage: 88.0 KB
<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   int64
 1   Угол нашивки, град  1040 non-null   int64
 2   Шаг нашивки         1040 non-null   float64
 3   Плотность нашивки   1040 non-null   float64
dtypes: float64(2), int64(2)
memory usage: 32.6 KB
```

```
#Объединим 2 датасета по индексу тип объединения INNER
df_rw = df_X_nup_rnd.merge(df_X_bp_rnd, left_index = True, right_index = True, how = 'inner')
df_rw.head(10).T
```

	0	1	2	3	4	5	6
Угол нашивки, град	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Шаг нашивки	4.000000	4.000000	4.000000	5.000000	5.000000	5.000000	5.000000
Плотность нашивки	57.000000	60.000000	70.000000	47.000000	57.000000	60.000000	70.000000
Соотношение матрица-наполнитель	1.857143	1.857143	1.857143	1.857143	2.771331	2.767918	2.569620
Количество отвердителя, м.%	30.000000	50.000000	49.900000	129.000000	111.860000	111.860000	111.860000
Содержание эпоксидных групп,%_2	22.267857	23.750000	33.000000	21.250000	22.267857	22.267857	22.267857
Температура вспышки, С_2	100.000000	284.615385	284.615385	300.000000	284.615385	284.615385	284.615385
Плотность, кг/м3	2030.000000	2030.000000	2030.000000	2030.000000	2030.000000	2000.000000	1910.000000
Поверхностная плотность, г/м2	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000
модуль упругости, ГПа	738.736842	738.736842	738.736842	738.736842	753.000000	748.000000	807.000000
Модуль упругости при растяжении, ГПа	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000
Прочность при растяжении, МПа	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000
Потребление смолы, г/м2	220.000000	220.000000	220.000000	220.000000	220.000000	220.000000	220.000000



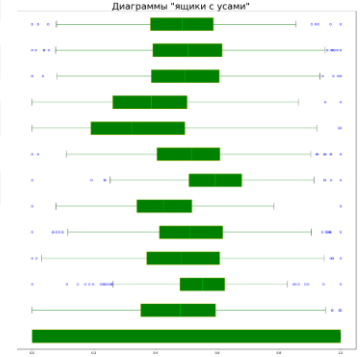


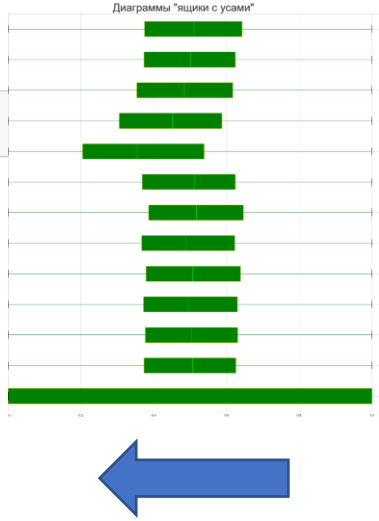
```
# Среднее значение
df.mean()
```

```
Угол нашивки                          0.510846
Шаг нашивки                           6.931939
Плотность нашивки                    57.562887
Соотношение матрица-наполнитель       2.927964
Количество отвердителя, м.%         111.136066
Содержание эпоксидных групп,%_2      22.200570
Температура вспышки, С_2            286.181128
Плотность, кг/м3                   1974.118744
Поверхностная плотность, г/м2       482.429070
модуль упругости, ГПа               736.119982
Модуль упругости при растяжении, ГПа  73.303464
Прочность при растяжении, МПа      2461.491315
Потребление смолы, г/м2             218.048059
dtype: float64
```

```
# Медианное значение
df.median()
```

```
Угол нашивки                          1.000000
Шаг нашивки                           6.972862
Плотность нашивки                    57.584225
Соотношение матрица-наполнитель       2.907832
Количество отвердителя, м.%         111.162090
Содержание эпоксидных групп,%_2      22.177681
Температура вспышки, С_2            286.220763
Плотность, кг/м3                   1977.321002
Поверхностная плотность, г/м2       457.732246
модуль упругости, ГПа               736.178435
Модуль упругости при растяжении, ГПа  73.247594
Прочность при растяжении, МПа      2455.974462
Потребление смолы, г/м2             218.697660
dtype: float64
```





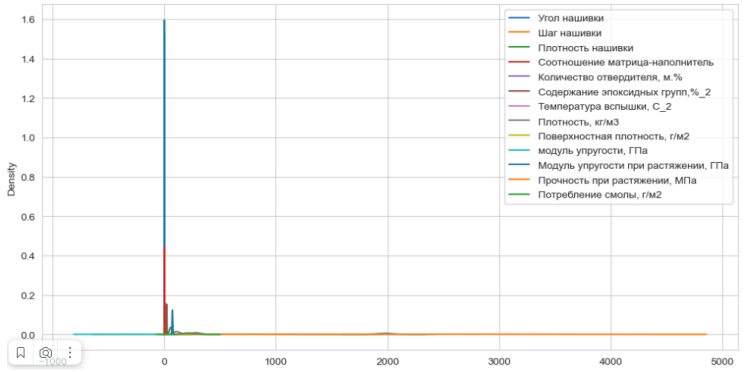
Предобработка данных:

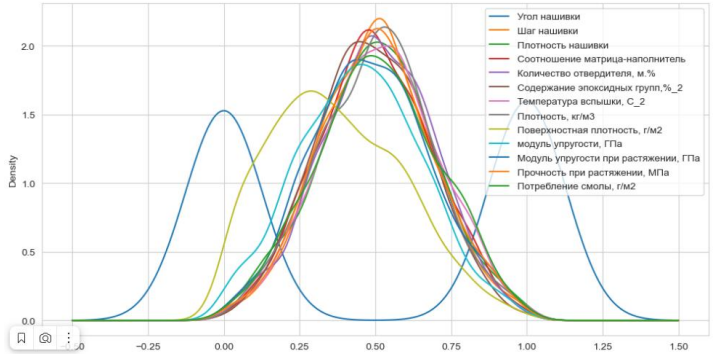
✓ Нормализация данных:

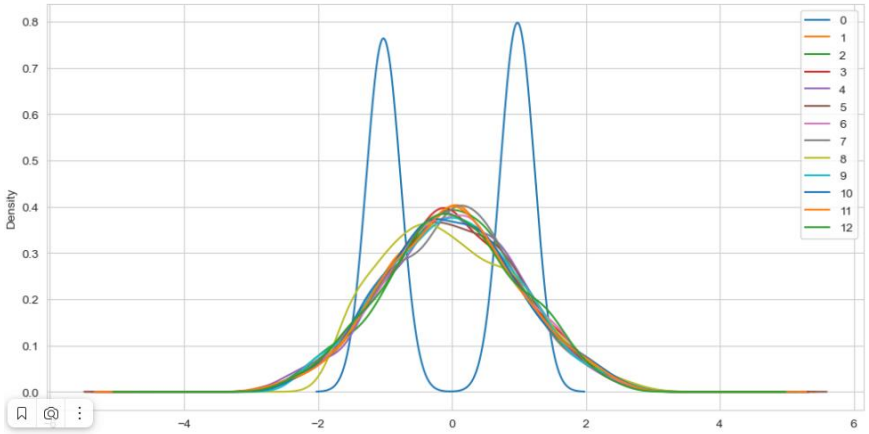
- Нормализуем данные MinMaxScaler();
- Построим график плотности ядра;
- Проверим результат MinMaxScaler();
- Построим графики MinMaxScaler();
- Нормализуем данные с помощью Normalizer();
- Проверим результат Normalizer();
- Построим графики Normalizer();

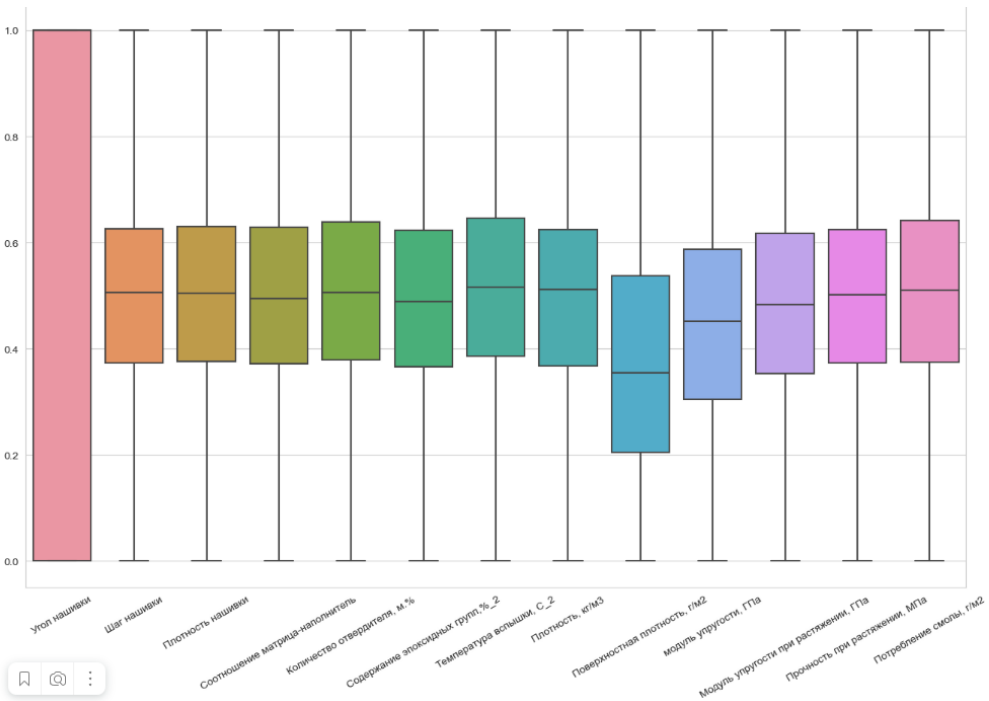
✓ Стандартизация данных:

- Стандартизируем данные с помощью StandardScaler();
- Проверим результат StandardScaler();
- Построим графики StandardScaler();










```
df_standart_1
```

	0	1	2	3	4	5	6
0	-1.021932	-1.166792	0.219240	-1.196467	-2.286425	0.647585	-0.039740
1	-1.021932	-0.768833	-0.950227	-1.196467	0.668092	-0.397291	0.350738
2	-1.021932	-0.768833	-0.050637	-0.175012	0.027074	0.028123	-0.039740
3	-1.021932	-0.768833	0.219240	-0.178825	0.027074	0.028123	-0.039740
4	-1.021932	-0.768833	1.118831	-0.400390	0.027074	0.028123	-0.039740
...	...	...	...	...	...	...	...
917	0.978538	0.853400	-0.948449	-0.733662	-0.902956	-0.868217	0.979545
918	0.978538	1.446055	-0.342933	0.576611	1.303201	-1.087006	-0.811326
919	0.978538	-1.102660	0.905599	0.394018	-0.022536	0.734311	-0.958342
920	0.978538	-0.246233	0.062808	0.868603	1.131763	-1.234469	-0.263996
921	0.978538	-0.339474	1.787628	0.983318	0.674952	2.204352	0.374919

922 rows × 13 columns

Разработка и обучение моделей для прогноза прочности при растяжении:

- Разбиваем данные на тестовую и тренировочную выборки;
- Обучаем модель;
- Вычисляем коэффициент детерминации;
- Считаем MAE, MAPE, MSE, RMSE, test score train и test score test;
- Сравниваем с результатами модели, выдающей среднее значение;
- Построим графики для тестовых и прогнозных значений;
- Построим гистограмму распределения ошибки
- Использованные работы и их результаты описаны справа.

```
Support Vector Regression Results Train:
Test score: 0.99
Support Vector Regression Results:
SVR_MAE: 78
SVR_MAPE: 0.04
SVR_MSE: 11671.63
SVR_RMSE: 108.04
Test score: 0.95
```

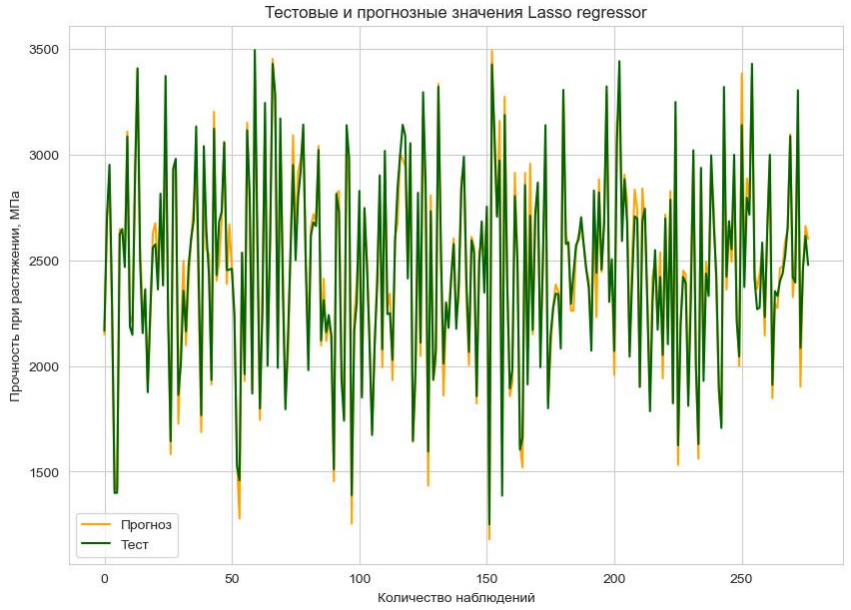
```
Random Forest Regressor Results Train:
Test score: 0.98
Random Forest Regressor Results:
RF_MAE:  75
RF_MAPE: 0.03
RF_MSE: 9144.40
RF_RMSE: 95.63
Test score: 0.96
```

```
Linear Regression Results Train:
Test score: 0.97
Linear Regression Results:
lr_MAE:  62
lr_MAPE: 0.03
lr_MSE: 6149.31
lr_RMSE: 78.42
Test score: 0.97
```

```
Gradient Boosting Regressor Results Train:
Test score: 0.99
Gradient Boosting Regressor Results:
GBR_MAE:  65
GBR_MAPE: 0.03
GBR_MSE: 6580.70
GBR_RMSE: 81.12
Test score: 0.97
```

- метод опорных векторов;
- случайный лес;
- линейная регрессия;
- градиентный бустинг;
- К-ближайших соседей;
- дерево решений;
- стохастический градиентный спуск;
- многослойный перцептрон;
- Лассо.

	Регрессор	MAE	Test score
0	Support Vector	78.477914	0.945472
1	RandomForest	75.456863	0.957278
2	Linear Regression	61.986894	0.971271
3	GradientBoosting	65.018011	0.969256
4	KNeighbors	102.030259	0.921868
5	DecisionTree	104.624022	0.916426
6	SGD	69.976071	0.962955
7	MLP	67.113346	0.965526
8	Lasso	69.475066	0.963792




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

```
Decision Tree Regressor Results Train:
Test score: 1.00
Decision Tree Regressor Results:
DTR_MAE:  105
DTR_MSE: 17888.80
DTR_RMSE: 133.75
DTR_MAPE: 0.05
Test score: 0.92
```

```
Stochastic Gradient Descent Regressor Results Train:
Test score: 0.95
Stochastic Gradient Descent Regressor Results:
SGD_MAE:  70
SGD_MSE: 7929.30
SGD_RMSE: 89.05
SGD_MAPE: 0.03
Test score: 0.96
```

```
Multi-layer Perceptron regressor Results Train:
Test score: 0.96
Multi-layer Perceptron regressor Results:
SGD_MAE:  67
SGD_MAPE: 0.03
SGD_MSE: 7378.94
SGD_RMSE: 85.90
Test score: 0.97
```

```
Lasso regressor Results Train:
Test score: 0.95
Lasso regressor Results:
SGD_MAE:  69
SGD_MAPE: 0.03
SGD_MSE: 7750.18
SGD_RMSE: 88.04
Test score: 0.96
```

Поиск гиперпараметров:

- Поиск гиперпараметров методом GridSearchCV с перекрёстной проверкой с количеством блоков 10;
- Выводим гиперпараметры для оптимальной модели;
- Подставляем оптимальные гиперпараметры в модель случайного леса;
- Обучаем модель;
- Оцениваем точность на тестовом наборе;
- Выводим наилучшее значение правильности перекрёстной проверки , наилучшие параметры, наилучшую модель по всем 9 методам;
- Проверяем правильность на тестовом наборе

	Регрессор	MAE	Test score
0	Support Vector	78.477914	0.945472
1	RandomForest	75.456863	0.957278
2	Linear Regression	61.986894	0.971271
3	GradientBoosting	65.018011	0.969256
4	KNeighbors	102.030259	0.921868
5	DecisionTree	104.624022	0.916426
6	SGD	69.976071	0.962955
7	MLP	67.113346	0.965526
8	Lasso	69.475066	0.963792
9	RandomForest_GridSearchCV	68.825594	0.963944
10	KNeighbors_GridSearchCV	99.281694	0.923491
11	DecisionTree_GridSearchCV	168.624997	0.778642

```
pipe = Pipeline([('preprocessing', StandardScaler()), ('regressor', SVR())])
param_grid = [
{'regressor': [SVR()], 'preprocessing': [StandardScaler(), MinMaxScaler(), None],
'regressor__gamma': [0.001, 0.01, 0.1, 1, 10, 100],
'regressor__C': [0.001, 0.01, 0.1, 1, 10, 100]},
{'regressor': [RandomForestRegressor(n_estimators = 100)],
'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [LinearRegression()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [GradientBoostingRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [KNeighborsRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [DecisionTreeRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [SGDRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [MLPRegressor(random_state = 1, max_iter = 500)], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [linear_model.Lasso(alpha = 0.1)], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},]
grid = GridSearchCV(pipe, param_grid, cv = 10)
grid.fit(x_train_1, np.ravel(y_train_1))
print("Наилучшие параметры:\n{}\n".format(grid.best_params_))
print("Наилучшее значение правильности перекрестной проверки: {:.2f}".format(grid.best_score_))
print("Правильность на тестовом наборе: {:.2f}".format(grid.score(x_test_1, y_test_1)))
```

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

Наилучшее значение правильности перекрестной проверки: 0.97
Правильность на тестовом наборе: 0.97
```

```
print("Наилучшая модель:\n{}".format(grid.best_estimator_))
```

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

Лучший препроцессинг - MinMaxScaler. Лучшая модель - лассо регрессии.

Разработка и обучение моделей для прогноза модуль упругости при растяжении:

- Разбиваем данные на тестовую и тренировочную выборки;
- Обучаем модель;
- Вычисляем коэффициент детерминации;
- Считаем MAE, MAPE, MSE, RMSE, test score train и test score test;
- Сравниваем с результатами модели, выдающей среднее значение;
- Построим графики для тестовых и прогнозных значений;
- Построим гистограмму распределения ошибки
- Использованные работы и их результаты описаны справа.


```
Support Vector Regression Results Train:
Test score: 0.90
Support Vector Regression Results:
SVR_MAE: 3
SVR_MAPE: 0.05
SVR_MSE: 16.94
SVR_RMSE: 4.12
Test score: -0.71
```

```
Random Forest Regressor Results Train:
Test score: 0.38
Random Forest Regressor Results:
RF_MAE:  3
RF_MAPE: 0.04
RF_MSE: 10.49
RF_RMSE: 3.24
Test score: -0.06
```

```
Linear Regression Results Train:
Test score: 0.02
Linear Regression Results:
lr_MAE:  3
lr_MAPE: 0.04
lr_MSE: 10.17
lr_RMSE: 3.19
Test score: -0.03
```

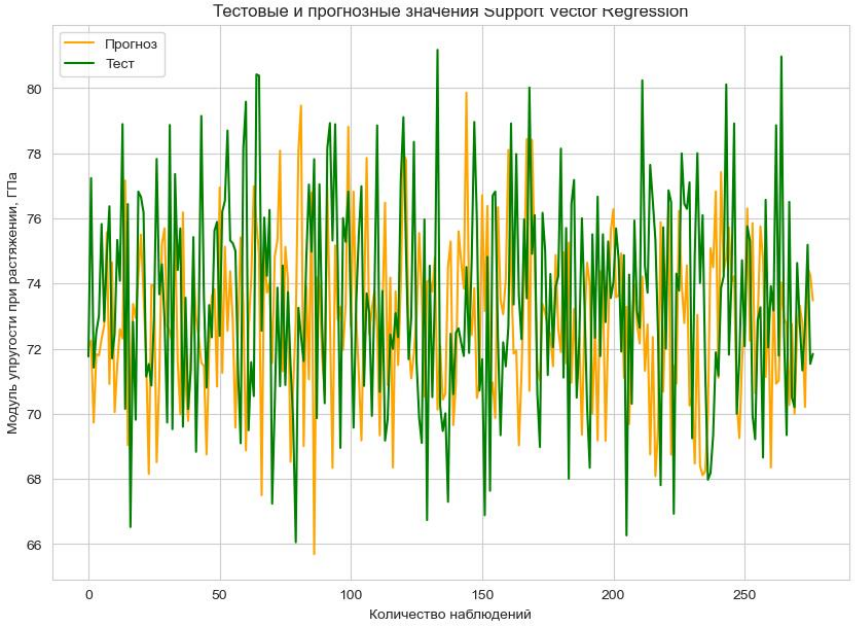
```
Gradient Boosting Regressor Results Train:
Test score: 0.49
Gradient Boosting Regressor Results:
GBR_MAE:  3
GBR_MAPE: 0.04
GBR_MSE: 10.92
GBR_RMSE: 3.30
Test score: -0.10
```

- метод опорных векторов;
- случайный лес;
- линейная регрессия;
- градиентный бустинг;
- К-ближайших соседей;
- дерево решений;
- стохастический градиентный спуск;
- многослойный перцептрон;
- Лассо.

	Регрессор	MAE	Test score
0	Support Vector	3.328027	-0.709412
1	RandomForest	2.654599	-0.058527
2	Linear Regression	2.612192	-0.026243
3	GradientBoosting	2.683693	-0.101453
4	KNeighbors	2.826817	-0.219121
5	DecisionTree	3.793327	-1.193823
6	SGD	2.622215	-0.059723
7	MLP	2.612881	-0.034282
8	Lasso	2.580193	-0.008879

```
K Neighbors Regressor  Results Train:
Test score: 0.18
K Neighbors Regressor  Results:
KNN_MAE:  3
KNN_MAPE: 0.04
KNN_MSE: 12.08
KNN_RMSE: 3.48
Test score: -0.22
```

```
Decision Tree Regressor Results Train:
Test score: 1.00
Decision Tree Regressor Results:
DTR_MAE:  4
DTR_MSE: 21.74
DTR_RMSE: 4.66
DTR_MAPE: 0.05
Test score: -1.19
```




```
Stochastic Gradient Descent Regressor Results Train:
Test score: -0.02
Stochastic Gradient Descent Regressor Results:
SGD_MAE:  3
SGD_MSE: 10.50
SGD_RMSE: 3.24
SGD_MAPE: 0.04
Test score: -0.06
```

```
Multi-layer Perceptron regressor Results Train:
Test score: 0.02
Multi-layer Perceptron regressor Results:
SGD_MAE:  3
SGD_MAPE: 0.04
SGD_MSE: 10.25
SGD_RMSE: 3.20
Test score: -0.03
```

```
Lasso regressor Results Train:
Test score: 0.00
Lasso regressor Results:
SGD_MAE:  3
SGD_MAPE: 0.04
SGD_MSE: 10.00
SGD_RMSE: 3.16
Test score: -0.01
```

Поиск гиперпараметров :

- Поиск гиперпараметров методом GridSearchCV с перекрёстной проверкой с количеством блоков 10;
- Выводим гиперпараметры для оптимальной модели;
- Подставляем оптимальные гиперпараметры в модель случайного леса;
- Обучаем модель;
- Оцениваем точность на тестовом наборе;
- Выводим наилучшее значение правильности перекрёстной проверки , наилучшие параметры, наилучшую модель по всем 9 методам;
- Проверяем правильность на тестовом наборе

	Регрессор	MAE	Test score
0	Support Vector	3.328027	-0.709412
1	RandomForest	2.654599	-0.058527
2	Linear Regression	2.612192	-0.026243
3	GradientBoosting	2.683693	-0.101453
4	KNeighbors	2.826817	-0.219121
5	DecisionTree	3.793327	-1.193823
6	SGD	2.622215	-0.059723
7	MLP	2.612881	-0.034282
8	Lasso	2.580193	-0.008879
9	RandomForest_GridSearchCV	2.613278	-0.035022
10	KNeighbors_GridSearchCV	2.792868	-0.014218
11	DecisionTree_GridSearchCV	2.600754	-0.005112

```
pipe2 = Pipeline([('preprocessing', StandardScaler()), ('regressor', SVR())])
param_grid2 = [
{'regressor': [SVR()], 'preprocessing': [StandardScaler(), MinMaxScaler(), None],
'regressor__gamma': [0.001, 0.01, 0.1, 1, 10, 100],
'regressor__C': [0.001, 0.01, 0.1, 1, 10, 100]},
{'regressor': [RandomForestRegressor(n_estimators=100)],
'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [LinearRegression()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [GradientBoostingRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [KNeighborsRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [DecisionTreeRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [SGDRegressor()], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [MLPRegressor(random_state=1, max_iter=500)], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},
{'regressor': [linear_model.Lasso(alpha=0.1)], 'preprocessing':  [StandardScaler(), MinMaxScaler(), None]},]
grid2 = GridSearchCV(pipe2, param_grid2, cv=10)
grid2.fit(x_train_2, np.ravel(y_train_2))
print("Наилучшие параметры:\n{}\n".format(grid2.best_params_))
print("Наилучшее значение правильности перекрестной проверки: {:.2f}".format(grid2.best_score_))
print("Правильность на тестовом наборе: {:.2f}".format(grid2.score(x_test_2, y_test_2)))
```

```
Наилучшие параметры:
{'preprocessing': StandardScaler(), 'regressor': SVR(C=10, gamma=100), 'regressor__C': 10, 'regressor__gamma': 100}

Наилучшее значение правильности перекрестной проверки: -0.01
Правильность на тестовом наборе: -0.01
```

```
print("Наилучшая модель:\n{}".format(grid2.best_estimator_))
```

```
Наилучшая модель:
Pipeline(steps=[('preprocessing', StandardScaler()),
                ('regressor', SVR(C=10, gamma=100))])
```

Лучший препроцессинг - StandardScaler. Лучшая модель - Метод опорных векторов.

Нейронная сеть для соотношения «матрица-наполнитель»:

✓ Первая модель:

- Сформируем входы и выход для модели.
- Разобьём выборки на обучающую и тестовую.
- Нормализуем данные.
- Создадим функцию для поиска наилучших параметров и слоёв.
- Построим модель, определим параметры, найдем оптимальные параметры посмотрим на результаты;
- Повторим все эти этапы до построения окончательной модели;
- Обучим нейросеть;
- Посмотрим на потери модели;
- Построим график потерь на тренировочной и тестовой выборках.
- Построим график результата работы модели.

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

5.1 Входы и выход для модели.

```
# Сформируем входы и выход для модели

tv = df['Соотношение матрица-наполнитель']
tr_v = df.loc[:, df.columns != 'Соотношение матрица-наполнитель']

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

5.2 Нормализация данных.

```
# Нормализуем данные

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

```
def create_model(lyrs=[32], act='softmax', opt='SGD', dr=0.1):

    seed = 7
    np.random.seed(seed)
    tf.random.set_seed(seed)

    model = Sequential()
    model.add(Dense(lyrs[0], input_dim=x_train.shape[1], activation=act))
    for i in range(1,len(lyrs)):
        model.add(Dense(lyrs[i], activation=act))

    model.add(Dropout(dr))
    model.add(Dense(3, activation='tanh'))  # выходной слой

    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['mae', 'accuracy'])

    return model
```

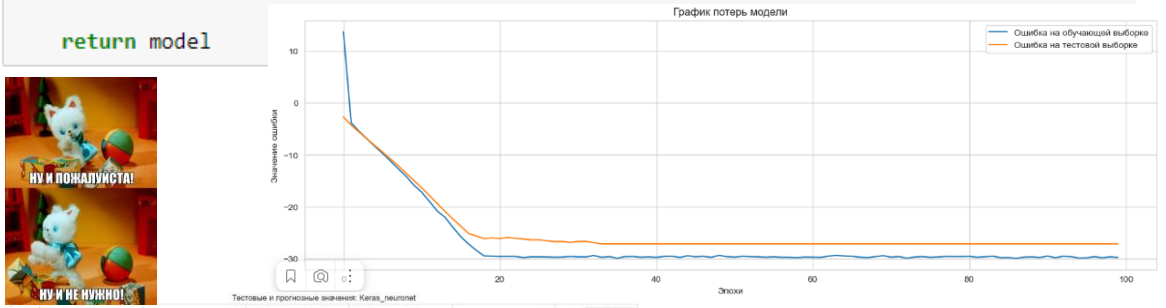
```
Model: "sequential_195"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_490 (Dense)           (None, 128)               1664

 dense_491 (Dense)           (None, 64)                8256

 dense_492 (Dense)           (None, 16)                1040

 dense_493 (Dense)           (None, 3)                 51

 dropout_195 (Dropout)       (None, 3)                 0

 dense_494 (Dense)           (None, 3)                 12

=================================================================
Total params: 11,023
Trainable params: 11,023
Non-trainable params: 0
_________________________________________________________________
```

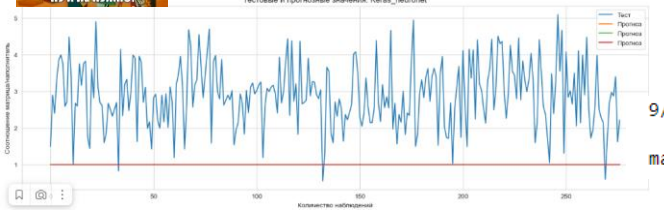
```
Best: 0.004639 using {'batch_size': 50, 'epochs': 200}
0.003077 (0.009231) with: {'batch_size': 4, 'epochs': 10}
0.003077 (0.009231) with: {'batch_size': 4, 'epochs': 50}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 100}
```

```
Best: 0.004639 using {'opt': 'Adagrad'}
0.001538 (0.004615) with: {'opt': 'SGD'}
0.001538 (0.004615) with: {'opt': 'RMSprop'}
0.004639 (0.009877) with: {'opt': 'Adagrad'}
0.001563 (0.004688) with: {'opt': 'Adadelta'}
0.000000 (0.000000) with: {'opt': 'Adam'}
0.001538 (0.004615) with: {'opt': 'Nadam'}
```

```
Best: 0.001538 using {'lyrs': [8]}
0.001538 (0.004615) with: {'lyrs': [8]}
0.001538 (0.004615) with: {'lyrs': [16, 4]}
0.001538 (0.004615) with: {'lyrs': [32, 8, 3]}
0.001538 (0.004615) with: {'lyrs': [12, 6, 3]}
0.001538 (0.004615) with: {'lyrs': [64, 64, 3]}
0.001538 (0.004615) with: {'lyrs': [128, 64, 16, 3]}
```

```
Best: 0.003101 using {'dr': 0.2}
0.001538 (0.004615) with: {'dr': 0.0}
0.001538 (0.004615) with: {'dr': 0.01}
0.001538 (0.004615) with: {'dr': 0.05}
0.001538 (0.004615) with: {'dr': 0.1}
0.003101 (0.006202) with: {'dr': 0.2}
0.001538 (0.004615) with: {'dr': 0.3}
0.001538 (0.004615) with: {'dr': 0.5}
```






```
9/9 [==============================] - 0s 1ms/step - loss: -28.9481 - mae: 1.9057 - accuracy: 0.0000e+00

mae: 190.57%
```

Нейронная сеть для соотношения «матрица-наполнитель»:

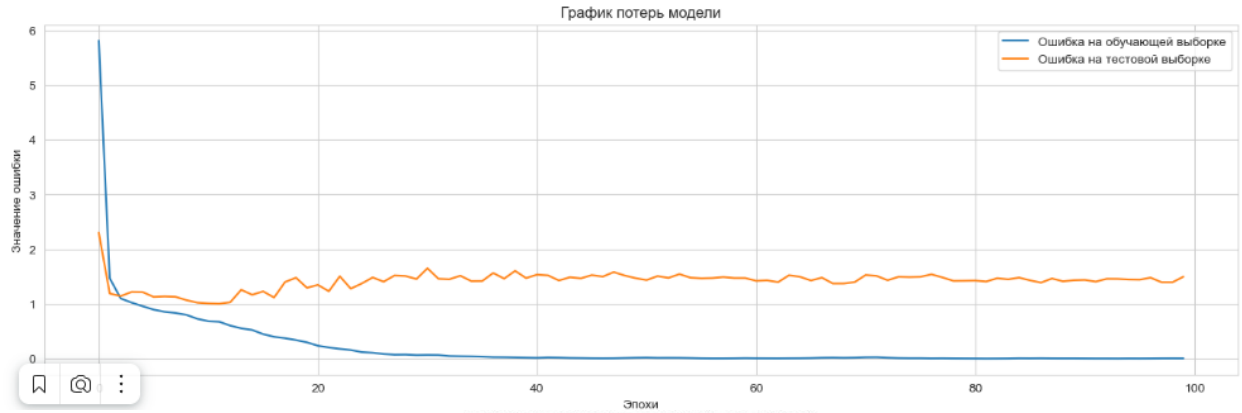
✓ Вторая модель:

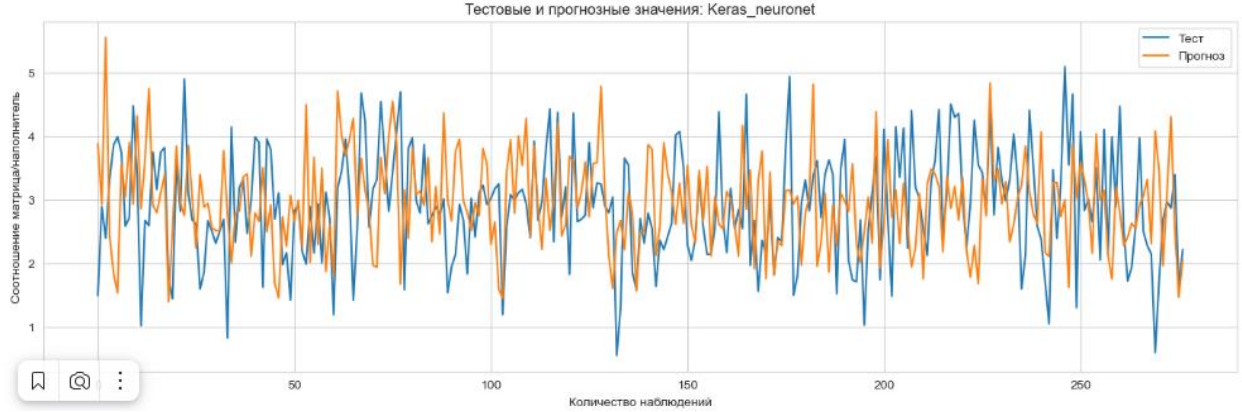
- Сформируем входы и выход для модели.
- Разобьём выборки на обучающую и тестовую.
- Нормализуем данные.
- Сконфигурируем модель, зададим слои, посмотрим на архитектуру модели.
- Обучим модель.
- Посмотрим на MAE, MAPE, Test score и на потери модели.
- Построим график потерь на тренировочной и тестовой выборках.
- Построим график результата работы модели.
- Оценим модель по MSE.

```
model1 = Sequential([x_train_n, Dense(128, activation='relu'),
                     Dense(128, activation='relu'),
                     Dense(128, activation='relu'),
                     Dense(64, activation='relu'),
                     Dense(64, activation='relu'),
                     Dense(32, activation='relu'),
                     Dense(16, activation='relu'),
                     Dense(1)])

model1.compile(optimizer = tf.keras.optimizers.Adam(0.001), loss =
# Посмотрим на архитектуру модели

model1.summary()
```

```
Model: "sequential_196"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 12)               25
 n)

 dense_495 (Dense)           (None, 128)               1664

 dense_496 (Dense)           (None, 128)               16512

 dense_497 (Dense)           (None, 128)               16512

 dense_498 (Dense)           (None, 64)                8256

 dense_499 (Dense)           (None, 64)                4160

 dense_500 (Dense)           (None, 32)                2080

 dense_501 (Dense)           (None, 16)                528

 dense_502 (Dense)           (None, 1)                 17

=================================================================
Total params: 49,754
Trainable params: 49,729
Non-trainable params: 25
```

```
Model Results:
Model_MAE:  1
Model_MAPE: 0.37
Test score: 1.20
```

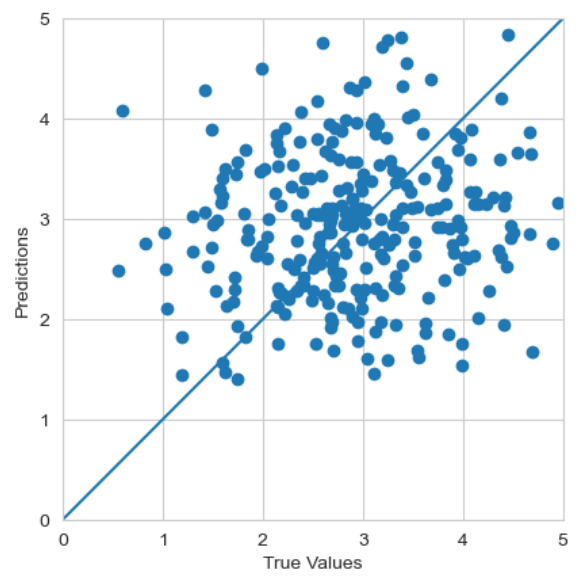




Вывод: Эта модель в отличие от первой, хотя бы работает. Она даёт высокий уровень ошибки, но не понятно, из-за чего так происходит. Скорее всего начальных данных было мало для получения хорошей модели, однако несмотря на слабую корреляцию данных модель способна делать предсказания, график справа показывает распределение предсказанных и настоящих значений, они довольно похожи.

```
# оценка модели MSE
model1.evaluate(x_test, y_test, verbose = 1)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 1.2014 - root_mean_squared_error: 1.0961

[1.2014341354370117, 1.0960994958877563]
```



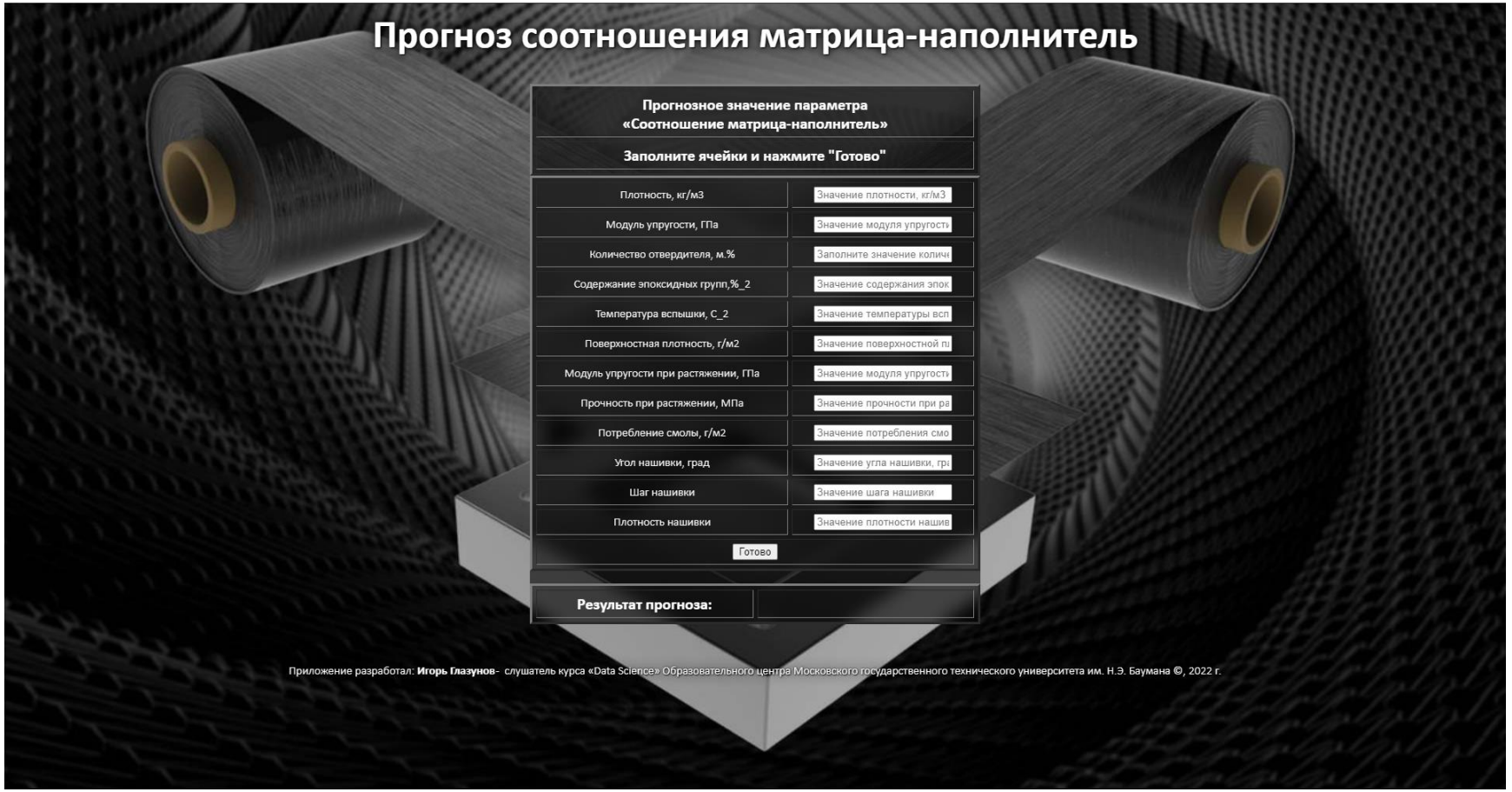
Приложение:

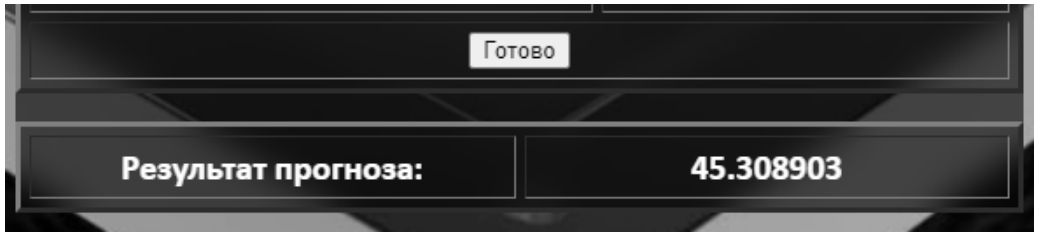
✓ Пользовательское приложение

- Сохранил вторую модель нейронной сети для разработки веб-приложения для прогнозирования соотношения "матрица-наполнитель" в фреймворке Flask;
- При запуске приложения, пользователь переходит на: http://127.0.0.1:5000/;
- В открывшемся окне пользователю необходимо ввести в соответствующие ячейки требуемые значения и нажать на кнопку «Готово».
- На выходе пользователь получает результат прогноза для значения параметра «Соотношение «матрица – наполнитель»».
- Приложение успешно работает

✓ Репозиторий на github.com

- https://github.com/IamGlazz/VKR_DPO
- https://colab.research.google.com/drive/1NLeeWBgoBLtHXUPzjtj2eGFPnxHF9dma?usp=sharing





ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Спасибо за внимание

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Трудности и ошибки

- Опечатки, описки, пропуски скобок.
- Из-за этого модели не работали, я шёл разными путями: искал ошибки в написанном коде и пробовал другие формулы, поэтому в работе одни и те же задачи решены разными (иногда практически одинаковыми способами) вариантами.
- Большой стопор возник при переносе ноутбука с jupiter в colab, потому что часть графиков не отображалась и результат был всегда разный в процессе работы.
- Но когда все ошибки, которые я смог найти, были устранены оба ноутбука заработали (кроме одного графика, так он и не захотел отображаться в colab).

Заключение

- Использованные при разработке моделей подходы не позволили получить сколько-нибудь достоверных прогнозов.
- Применённые модели регрессии не показали высокой эффективности в прогнозировании свойств композитов.
- Невозможно определить из свойств материалов соотношение «матрица – наполнитель»
- Текущим набором алгоритмов задача эффективно не решается.