

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**по курсу
«Data Science»**

**Тема: «Прогнозирование конечных свойств новых материалов
(композиционных материалов)»**

Слушатель

Зюликов Олег Александрович

Москва, 2022

Содержание:

1. Аналитическая часть	4
1.1. Постановка задачи	4
1.2. Описание используемых методов	7
1.3. Разведочный анализ данных	15
2. Практическая часть	16
2.1. Предобработка данных	16
2.2. Разработка и обучение модели	23
2.3. Тестирование модели	26
2.4. Нейронная сеть	27
2.5. Разработка приложения	31
2.6. Создание удаленного репозитория	32
2.7. Заключение	33
2.8. Список использованной литературы	34

Введение

Композиционные материалы — это искусственно созданные материалы, состоящие из нескольких других с четкой границей между ними. Композиты обладают теми свойствами, которые не наблюдаются у компонентов по отдельности. При этом композиты являются монолитным материалом, т. е. компоненты материала неотделимы друг от друга без разрушения конструкции в целом.

Яркий пример композита - железобетон. Бетон прекрасно сопротивляется сжатию, но плохо растяжению. Стальная арматура внутри бетона компенсирует его неспособность сопротивляться сжатию, формируя тем самым новые, уникальные свойства. Современные композиты изготавливаются из других материалов: полимеры, керамика, стеклянные и углеродные волокна, но данный принцип сохраняется. У такого подхода есть и недостаток: даже если мы знаем характеристики исходных компонентов, определить характеристики композита, состоящего из этих компонентов, достаточно проблематично. Для решения этой проблемы есть два пути: физические испытания образцов материалов, или прогнозирование характеристик.

Суть прогнозирования заключается в симуляции представительного элемента объема композита, на основе данных о характеристиках входящих компонентов (связующего и армирующего компонента). На входе имеются данные о начальных свойствах компонентов композиционных материалов (количество связующего, наполнителя, температурный режим отверждения и т.д.). На выходе необходимо спрогнозировать ряд конечных свойств получаемых композиционных материалов.

Актуальность: Созданные прогнозные модели помогут сократить количество проводимых испытаний, а также пополнить базу данных материалов возможными

новыми характеристиками материалов, и цифровыми двойниками новых композитов.

В процессе исследовательской работы были разработаны несколько моделей для предсказания модуля упругости при растяжении и прочности при растяжении, а также была создана нейронная сеть, которая рекомендует соотношение матрица–наполнитель при определенных параметрах. На основе этой нейронной сети было создано веб-приложение с использованием фреймворка Flask, которое запрашивает у пользователя значения входных параметров и выдает рекомендацию по соотношению матрица-наполнитель.

1. Аналитическая часть

1.1. Постановка задачи

Для исследовательской работы были даны два файла: X_bp.xlsx и X_nup.xlsx. Необходимо было загрузить эти файлы и сделать объединение в один датасет по индексу. Для этого использовать тип объединения INNER.

Целью работы является создать модели машинного обучения для прогноза модуля упругости при растяжении и прочности при растяжении. А также разработать нейронную сеть, которая будет рекомендовать соотношение матрица-наполнитель при заданных параметрах.

Перед созданием моделей необходимо провести разведочный анализ данных. Для каждого признака получить среднее, медианное значение, провести анализ и исключение выбросов, проверить наличие пропусков; перед обработкой данные: удалить шумы и выбросы, сделать нормализацию и стандартизацию.

После разработки моделей нужно создать приложение на Flask для рекомендации значения соотношения матрица-наполнитель, а также загрузить результаты работы на удаленный репозиторий GitHub.

Перед началом работы был произведен импорт необходимых для работы библиотек. Также необходимые библиотеки добавлялись в этот список по мере надобности в процессе работы над заданием.

▼ Импортируем необходимые библиотеки

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter('ignore')
%matplotlib inline

from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn import svm
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import sklearn.metrics as metrics
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, mean_absolute_percentage_error

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from tensorflow.keras.utils import normalize
from keras.utils.np_utils import normalize
from tensorflow.keras import optimizers
```

Рисунок 1 - начало работы – импорт необходимых библиотек

После этого были загружены наши файлы с данными, которые были объединены в один датасет.

	Unnamed: 0	Соотношение матрица- наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, C_2
0	0.0	1.857143	2030.000000	738.736842	30.000000	22.267857	100.000000
1	1.0	1.857143	2030.000000	738.736842	50.000000	23.750000	284.615385
2	2.0	1.857143	2030.000000	738.736842	49.900000	33.000000	284.615385
3	3.0	1.857143	2030.000000	738.736842	129.000000	21.250000	300.000000
4	4.0	2.771331	2030.000000	753.000000	111.860000	22.267857	284.615385
...
1018	1018.0	2.271346	1952.087902	912.855545	86.992183	20.123249	324.774576
1019	1019.0	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401
1020	1020.0	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047
1021	1021.0	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840
1022	1022.0	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708

1023 rows × 11 columns

Рисунок 2 – часть датасета, загруженного из файла

Для объединения файлов мы использовали тип объединения INNER, а объединение проводили по индексу .

```
df = df1.merge(df2, how='inner', left_index=True, right_index=True)
df
```

	Unnamed: 0_x	Соотношение матрица- наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, C_2
0	0.0	1.857143	2030.000000	738.736842	30.000000	22.267857	100.000000
1	1.0	1.857143	2030.000000	738.736842	50.000000	23.750000	284.615385
2	2.0	1.857143	2030.000000	738.736842	49.900000	33.000000	284.615385
3	3.0	1.857143	2030.000000	738.736842	129.000000	21.250000	300.000000
4	4.0	2.771331	2030.000000	753.000000	111.860000	22.267857	284.615385
...
1018	1018.0	2.271346	1952.087902	912.855545	86.992183	20.123249	324.774576
1019	1019.0	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401
1020	1020.0	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047
1021	1021.0	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840
1022	1022.0	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708

1023 rows × 15 columns

Рисунок 3 – объединение файлов в один датасет

Изначально данные из файла X_br.xlsx имели размер 1023 строки и 11 колонок, а данные из файла X_nur.xlsx – 1040 строк и 4 колонки. После

объединения наш датасет приобрел размер 1023 строки и 15 колонок, причем две колонки были одинаковы – это индексы из обоих файлов. Было принято решение удалить эти колонки. После удаления размер датасета стал 1023 строки и 13 колонок. То есть у нас получилось 13 признаков и 1023 строки их значений.

```
df.columns = ['соот_матр_нап', 'плотность', 'мод_упр', 'кол_отв', 'сод_эп_гр', 'темп_всп', 'пов_плотн',  
df
```

	соот_матр_нап	плотность	мод_упр	кол_отв	сод_эп_гр	темп_всп	пов_плотн	мод_упр_раст
0	1.857143	2030.000000	738.736842	30.000000	22.267857	100.000000	210.000000	70.000000

Рисунок 4 – корректировка заголовков признаков

Для удобства работы мы произвели корректировку заголовков признаков, переименовав их и сделав более короткими.

После этого наш датасет стал готов к разведочному анализу.

1.2 Описание используемых методов

Для решения этой задачи использовались методы машинного обучения с учителем, именно методы регрессии. В процессе работы были применены следующие методы:

- линейная регрессия;
- метод опорных векторов;
- градиентный бустинг;
- случайный лес;

Линейная регрессия — это алгоритм машинного обучения, основанный на контролируемом обучении, рассматривающий зависимость между одной входной и выходными переменными. Это один из самых простых и эффективных инструментов статистического моделирования. Она определяет зависимость одной (объясняемой, зависимой) переменной от другой или нескольких других

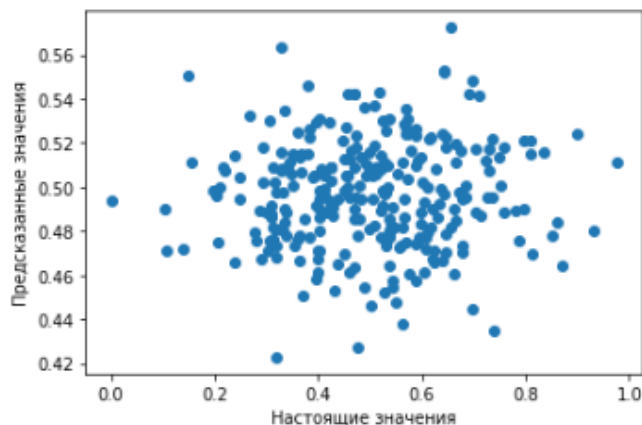
переменных (факторов, регрессоров, независимых переменных) с линейной функцией зависимости.

Модель линейной регрессии является часто используемой и наиболее изученной. А именно изучены свойства оценок параметров, получаемых различными методами при предположениях о вероятностных характеристиках факторов, и случайных ошибок модели

Достоинства метода: быстр и прост в реализации; легко интерпретируем; имеет меньшую сложность по сравнению с другими алгоритмами;

```
plt.scatter(y_test, y_mod_pred)
plt.xlabel('Настоящие значения')
plt.ylabel('Предсказанные значения')
plt.show()

print('Средняя абсолютная ошибка:', mean_absolute_error(y_test, y_mod_pred))
print('Средняя квадратическая ошибка:', metrics.mean_squared_error(y_test, y_mod_pred))
print('Корень из среднеквадратичной ошибки:', np.sqrt(metrics.mean_squared_error(y_test, y_mod_pred)))
print('Train score:', lr_mod.score(X_train, y_train))
print('Test score:', lr_mod.score(X_test, y_test))
```

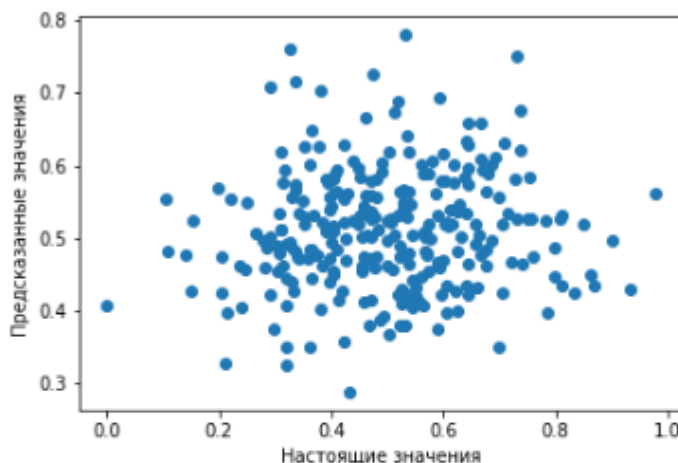


Средняя абсолютная ошибка: 0.1314576268724851
Средняя квадратическая ошибка: 0.02653044502216099
Корень из среднеквадратичной ошибки: 0.16288169026063362
Train score: 0.02354957271791802
Test score: -0.013409993292819111

Рисунок 5 – результаты и график рассеяния настоящих и предсказанных значений методом линейной регрессии

Недостатки метода: моделирует только прямые линейные зависимости; требует прямую связь между зависимыми и независимыми переменными; выбросы оказывают огромное влияние.

Метод опорных векторов - это метод машинного обучения, целью которого является попытка классифицировать входные наборы данных в один из двух классов. Для эффективной работы метода сначала необходимо использовать обучающую выборку, состоящую из входных и выходных данных, которая необходима для построения модели метода опорных векторов, и которую в дальнейшем можно будет использовать для классификации новых данных.



Средняя абсолютная ошибка: 0.14080290512299304
Средняя квадратическая ошибка: 0.03074812596151863
Корень из среднеквадратичной ошибки: 0.17535143558442465
Train score: 0.4319282793427546
Test score: -0.17451697845252467

Рисунок 6 – результаты и график рассеяния настоящих и предсказанных значений методом опорных векторов

Для построения модели метода опорных векторов нужно взять обучающие входные данные, отобразить их в многомерное пространство, а затем использовать регрессию, чтобы найти гиперплоскость (гиперплоскость - это поверхность в n -мерном пространстве, которая разделяет его на два подпространства), которая лучше всего разделяла бы два класса входных данных. После обучения модели она способна классифицировать новые входные данные в один из классов при помощи разделяющей гиперплоскости.

По существу, метод опорных векторов является методом входов/выходов. Пользователь вводит входные данные, и на основе разработанной (при помощи обучения) модели получает выходные результаты. Теоретически, число входов для метода опорных векторов лежит в диапазоне от одного до бесконечности. Однако, в практическом применении, есть определенные ограничения на размер входной выборки, которые зависят от вычислительной мощности. Метод опорных векторов – это бинарный линейный классификатор, который хорошо работает на небольших датасетах. Данный алгоритм – это алгоритм обучения с учителем, использующихся для задач классификации и регрессионного анализа.

Достоинства метода: для классификации достаточно небольшого набора данных. При правильной работе модели, построенной на тестовом множестве, вполне возможно применение данного метода на реальных данных

Недостатки метода: неустойчивость к шуму

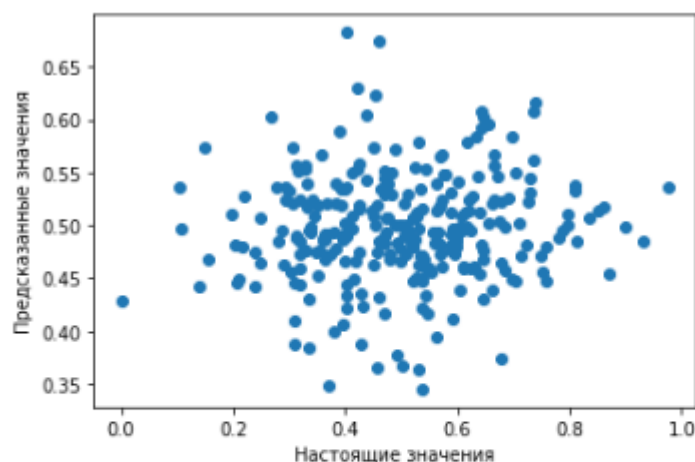
Градиентный бустинг — метод машинного обучения, который создает решающую модель прогнозирования в виде ансамбля слабых моделей прогнозирования, обычно деревьев решений. Он строит модель поэтапно, позволяя оптимизировать произвольную дифференцируемую функцию потерь.

Основная идея градиентного бустинга: строятся последовательно несколько базовых классификаторов, каждый из которых как можно лучше компенсирует

недостатки предыдущих. Финальный классификатор является линейной композицией этих базовых классификаторов.

Достоинства метода: новые алгоритмы учатся на ошибках предыдущих; требуется меньше итераций, чтобы приблизиться к фактическим прогнозам; наблюдения выбираются на основе ошибки; прост в настройке темпа обучения и применения; легко интерпретируем.

Недостатки метода: необходимо тщательно выбирать критерии остановки, иначе это может привести к переобучению; наблюдения с наибольшей ошибкой появляются чаще; слабее и менее гибко чем нейронные сети.

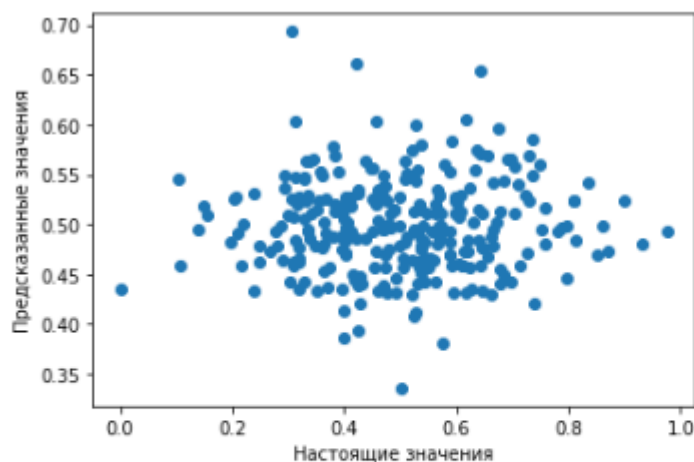


Средняя абсолютная ошибка: 0.13383360968515376
Средняя квадратическая ошибка: 0.027403045830818352
Корень из среднеквадратичной ошибки: 0.16553865358525288
Train score: 0.4906933248119718
Test score: -0.04674160076906397

Рисунок 7 – результаты и график рассеяния настоящих и предсказанных значений методом градиентного бустинга

Случайный лес – ансамблиевый метод, способный выполнять как задачи регрессии, так и классификации с использованием нескольких деревьев решений и техники, называемой начальной загрузкой и агрегацией, обычно известной как bagging. Основная идея, лежащая в основе этого, заключается в объединении нескольких деревьев решений для определения конечного результата, а не в том, чтобы полагаться на отдельные деревья решений.

Случайный лес имеет несколько деревьев решений в качестве базовых моделей обучения. Мы случайным образом выполняем выборку объектов из набора данных, формируя образцы наборов данных для каждой модели. Эта часть называется Bootstrap. Достоинства метода: не переобучается; не требует предобработки входных данных; эффективно обрабатывает пропущенные данные, данные с большим числом классов и признаков; имеет высокую точность предсказания и внутреннюю оценку обобщающей способности модели, а также высокую масштабируемость.



Средняя абсолютная ошибка: 0.13528209334394553
Средняя квадратическая ошибка: 0.028025463599794666
Корень из среднеквадратичной ошибки: 0.16740807507344044
Train score: 0.6427939276105532
Test score: -0.07051671598317877

Рисунок 8 – результаты и график рассеяния настоящих и предсказанных значений методом случайного леса

Недостатки метода: построение занимает много времени; сложно интерпретируемый; не обладает возможностью экстраполяции; может недообучаться; трудоёмко прогнозируемый; иногда работает хуже, чем линейные методы.

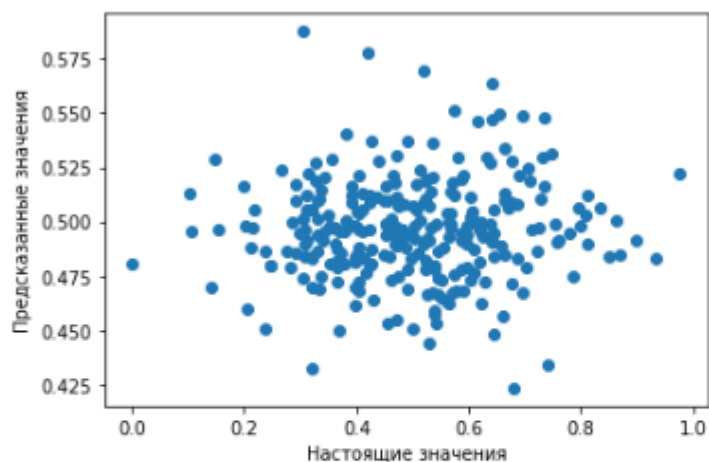
При построении модели случайного леса также был произведен поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой.

```
parameters = { 'n_estimators': [100, 500],
               'max_depth': [5, 20],
               'max_features': ['auto'],
               'criterion': ['mse'] }
grid = GridSearchCV(estimator = rfr_mod, param_grid = parameters, cv = 10)
grid.fit(X_train, y_train)

GridSearchCV(cv=10,
             estimator=RandomForestRegressor(max_depth=10, n_estimators=20),
             param_grid={'criterion': ['mse'], 'max_depth': [5, 20],
                        'max_features': ['auto'], 'n_estimators': [100, 500]})
```

Рисунок 9 – поиск гиперпараметров для модели случайного леса

После подстановки найденных гиперпараметров в модель, алгоритм выдал более точный результат предсказания на наших данных.



Средняя абсолютная ошибка: 0.13085290669282332
 Средняя квадратическая ошибка: 0.026335264811484127
 Корень из среднеквадратичной ошибки: 0.16228143705145123
 Train score: 0.2690318405781257
 Test score: -0.00595449920564084

Рисунок 10 – результаты и график рассеяния настоящих и предсказанных значений методом случайного леса с подбором гиперпараметров

При сравнении эффективности алгоритмов нами были применены следующие метрики: средняя абсолютная ошибка (MAE), средняя квадратическая ошибка (MSE) и корень из среднеквадратической ошибки (RMSE), а также мы рассчитывали score на тренировочной и тестовой выборках.

Практически во всех моделях score на тренировочной выборке был значительно выше, чем на тестовой. Что говорит о недостаточном обучении нашей модели для эффективного предсказания результата.

Наиболее точные прогнозы были сделаны с помощью метода линейной регрессии, как в предсказаниях модуля упругости при растяжении, так и для предсказания прочности при растяжении.

Ниже представлены сравнительные показатели ошибок используемых моделей с помощью метрики MAE.

Средняя абсолютная ошибка для прогноза модуля упругости при растяжении:

- Линейная регрессия - 0,146.
- Метод опорных векторов - 0,153.
- Метод градиентного бустинга - 0,149.
- Случайный лес - 0,15.
- Случайный лес с подборкой гиперпараметров - 0,147.

Рисунок 11 – средняя абсолютная ошибка для прогноза модуля упругости при растяжении

Средняя абсолютная ошибка для прогноза прочности при растяжении:

- Линейная регрессия - 0,1467.
- Метод опорных векторов - 0,1528.
- Метод градиентного бустинга - 0,1502.
- Случайный лес - 0,1484.
- Случайный лес с подборкой гиперпараметров - 0,1471.

Рисунок 12 – средняя абсолютная ошибка для прогноза прочности при растяжении

1.3 Разведочный анализ данных

Для того, чтобы работа с данными была эффективна, необходимо их соответствующим образом обработать: выявить пропущенные значения, дубликаты, выбросы, очистить данные. Эти задачи решает разведочный анализ данных.

```
df.duplicated().sum()
```

0

Рисунок 13 – поиск дубликатов в нашем датасете

С помощью него мы можем посмотреть на наши данные с различных сторон, изучить их для дальнейшей эффективной обработки.

В нашем наборе данных ни пропусков, ни дубликатов не оказалось.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1023 entries, 0 to 1022
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   соот_матр_нап         1023 non-null   float64
1   плотность             1023 non-null   float64
2   мод_упр               1023 non-null   float64
3   кол_отв               1023 non-null   float64
4   сод_эп_гр             1023 non-null   float64
5   темп_всп              1023 non-null   float64
6   пов_плотн             1023 non-null   float64
7   мод_упр_раст          1023 non-null   float64
8   проч_раст             1023 non-null   float64
9   потр_смолы            1023 non-null   float64
10  угол_наш              1023 non-null   float64
11  шаг_наш               1023 non-null   float64
12  плотн_наш             1023 non-null   float64
dtypes: float64(13)
memory usage: 111.9 KB
```

Рисунок 14 – изучение данных, поиск пропущенных значений

При разведочном анализе данных мы также в обязательном порядке должны посмотреть описательные статистики нашего датасета. С помощью описательных статистик мы провели оценку статистических характеристик датасета: среднее и медианное значение для каждой колонки, минимальное и максимальное значение, а также квантили.

```
df.describe()
```

	соот_матр_нап	плотность	мод_упр	кол_отв	сод_эп_гр	темп_всп	пов_плотн	мод_упр_раст	проч_раст	потр_смолы	угол_наш	шаг_наш	плотн_наш
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

Рисунок 15 – изучение описательных статистик с помощью метода describe()

В качестве инструментов разведочного анализа данных также используются: гистограммы распределения каждой из переменной; диаграммы ящика с усами; попарные графики рассеяния точек.

2 Практическая часть

2.2 Предобработка данных

Построим гистограммы для изучения распределения наших переменных.

```
fig, axes = plt.subplots(nrows = 7, ncols = 2, figsize=(20,50))
l = 0
for i in range(7):
    for j in range(2):
        axes[i, j].hist(df[int_var[l]], bins = 50, color = "seagreen", edgecolor='black', linewidth=1.2)
        axes[i, j].set_title("Гистограмма \n для признака <<{}>>".format(int_var[l]))
        axes[i, j].set_xlabel(int_var[l])
        l+=1
    if l > 12:
        break
```

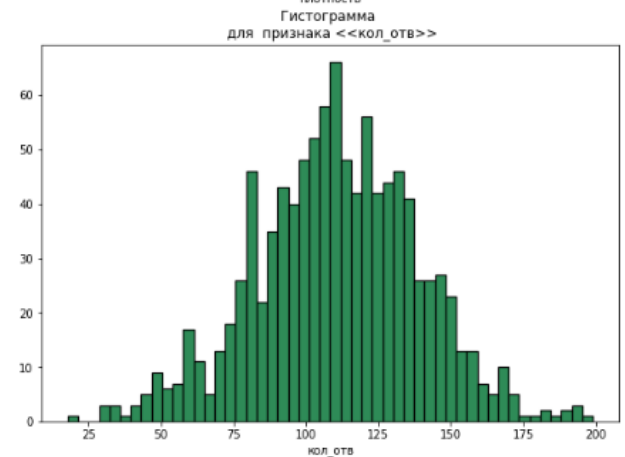
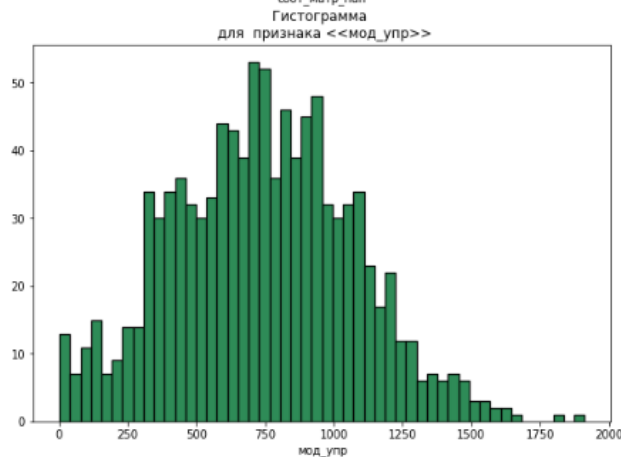
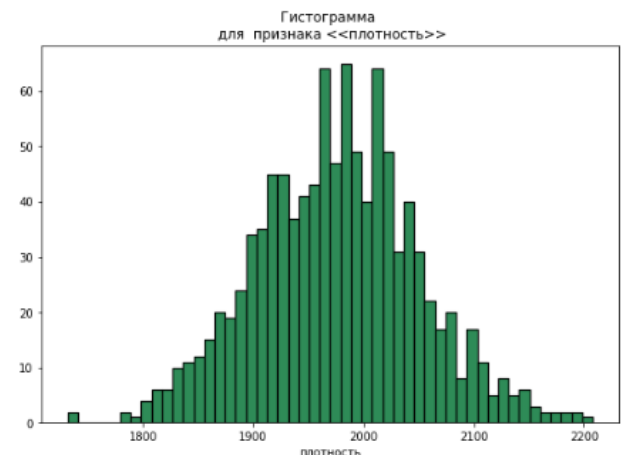
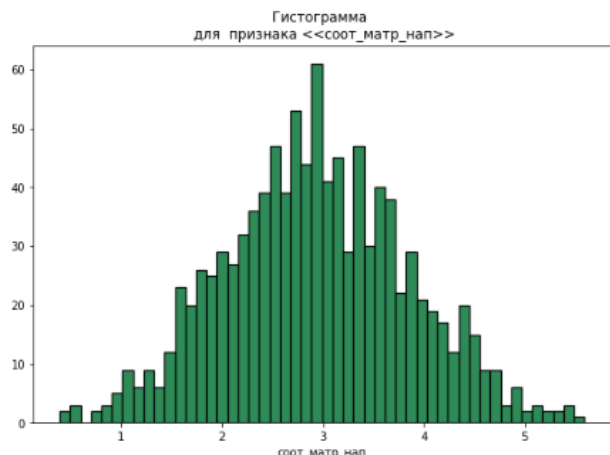


Рисунок 16 – гистограммы распределения переменных

Также для изучения наших признаков построим попарные графики рассеяния.

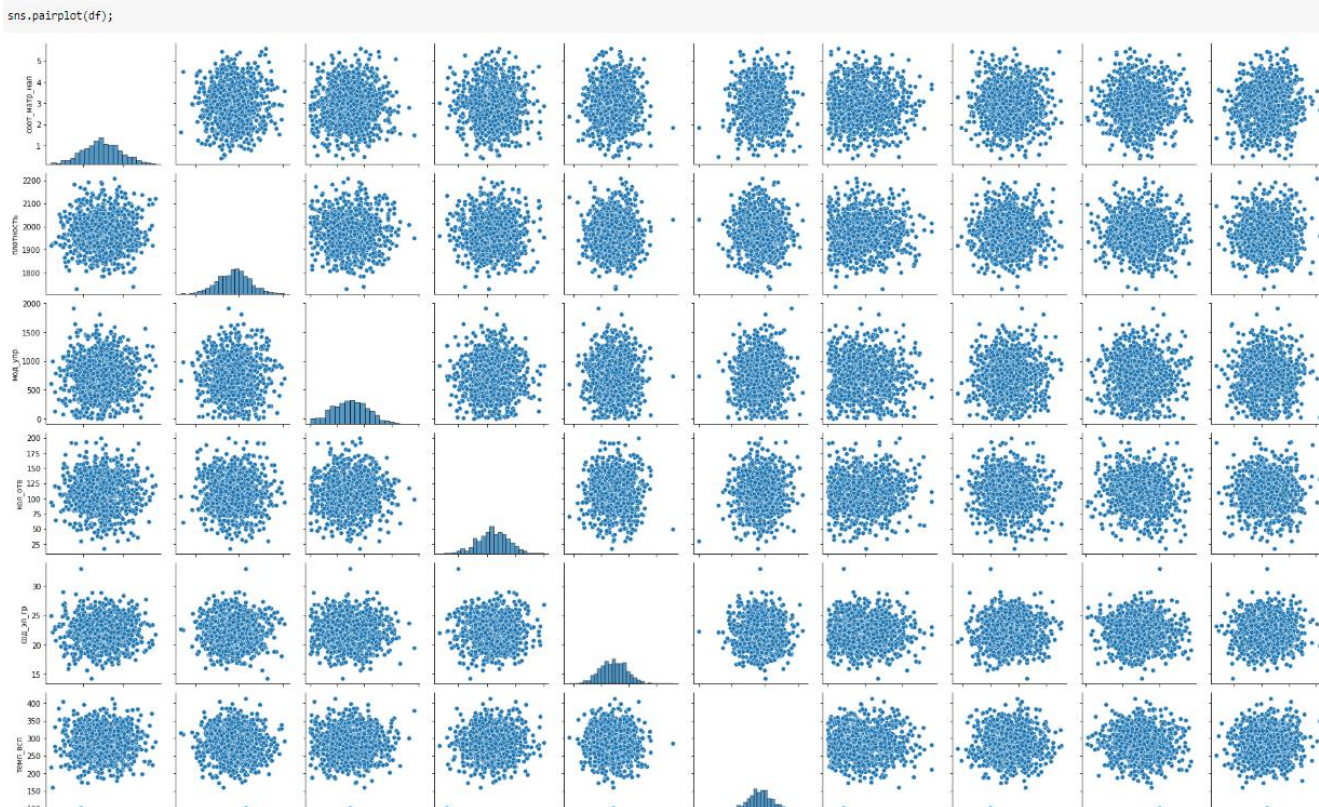


Рисунок 17 – попарные графики рассеяния

Здесь мы можем наблюдать, что какая-либо значимая корреляция между признаками визуально отсутствует.

Важным этапом обработки данных является удаление выбросов из нашего набора данных. Это необходимо для более точного предсказания наших целевых переменных, поскольку выбросы будут очень сильно влиять на итог.

Для этой цели мы будем использовать диаграммы ящика с усами. Эти диаграммы позволяют визуально определить выбросы в наших данных. Мы построим диаграммы, с их помощью определим числовые значения наших выбросов и удалим их, чтобы они не влияли на точность наших предсказаний.

```
fig, axes = plt.subplots(2, sharey=True, figsize=(20,50))
for i, col in enumerate(int_var):
    plt.subplot(2,2,i+1)
    sns.boxplot(data=df[col], whis = 1.5, color = "seagreen")
    plt.title("Ящик с усами \n для признака <<{}>>".format(col))
    plt.xlabel(col)
```

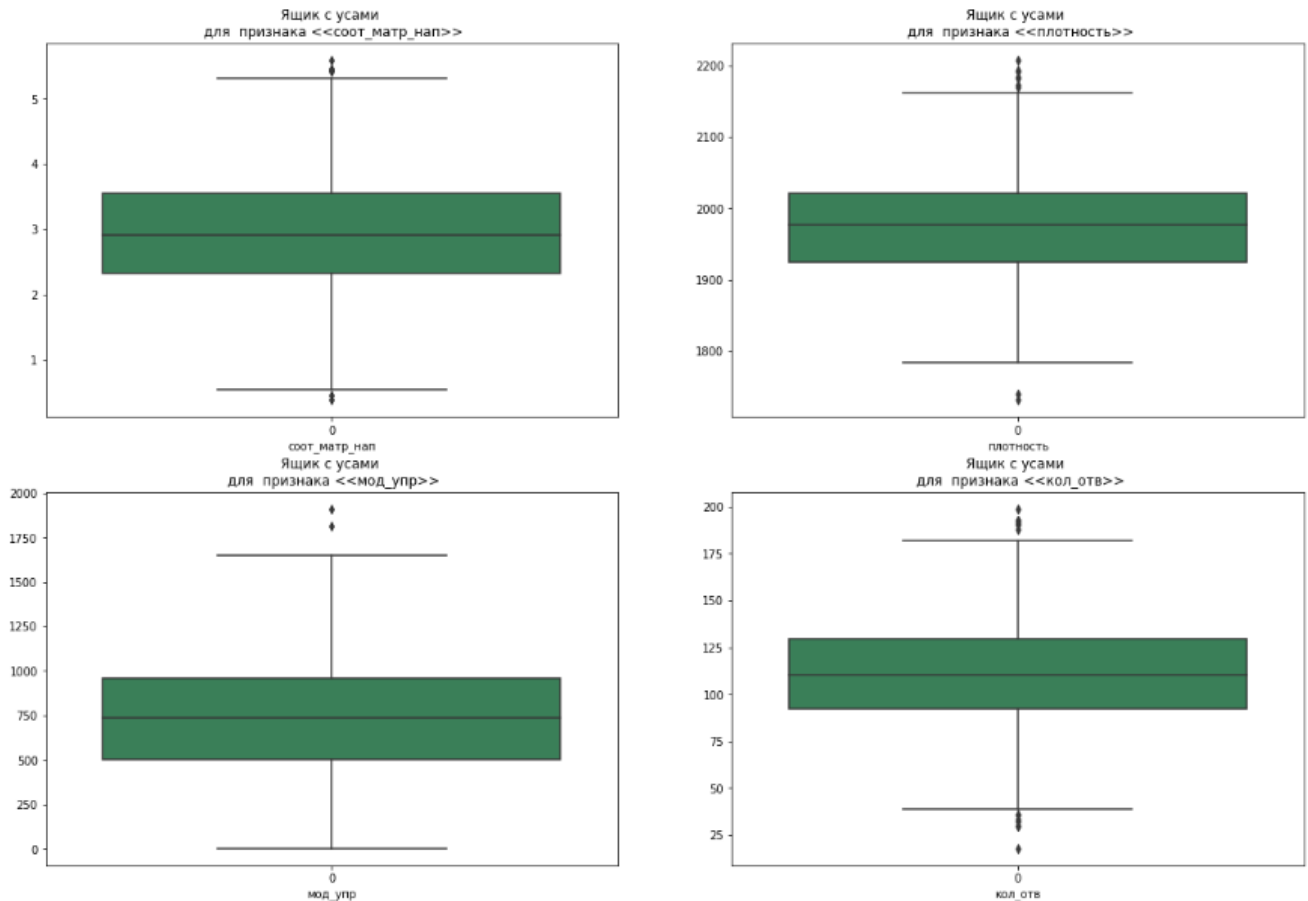


Рисунок 18 – диаграммы ящика с усами

На диаграммах видны точки за пределами – это выбросы, мы их удалим, чтобы их влияние не сказывалось негативно на предсказаниях наших моделей. Причем наши целевые признаки – соотношение матрица-наполнитель, модуль упругости при растяжении и прочность при растяжении мы не будем трогать, поскольку есть вероятность, что это не выбросы, а просто нам не хватает данных для более точной статистической оценки.

После удаления выбросов в нашем датасете осталось 924 строки.

	соот_матр_нап	плотность	мод_упр	кол_отв	сод_эл_гр	темп_всп	пов_плотн	мод_упр_раст	проч_раст	потр_смолы	угол_наш	шаг_наш	плотн_наш
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

924 rows × 13 columns

Рисунок 19 – датасет после удаления выбросов

Следующим нашим шагом будет нормализация данных. Для того, чтобы привести данные в диапазон между 0 и 1. Это позволит моделям более точно сравнивать данные друг с другом.

```

mms = MinMaxScaler()
data_norm = mms.fit_transform(np.array(df))

df_norm = pd.DataFrame(data = data_norm, columns = ['соот_матр_нап', 'плотность', 'мод_упр', 'кол_отв', 'сод_эл_гр', 'темп_всп', 'пов_плотн', 'мод_упр_раст', 'проч_раст', 'потр_смолы', 'угол_наш', 'шаг_наш', 'плотн_наш'])
df_norm

```

	соот_матр_нап	плотность	мод_упр	кол_отв	сод_эл_гр	темп_всп	пов_плотн	мод_упр_раст	проч_раст	потр_смолы	угол_наш	шаг_наш	плотн_наш
0	0.282131	0.617489	0.447061	0.068506	0.596696	0.511042	0.169158	0.319194	0.698235	0.514688	0.0	0.291000	0.573774
1	0.282131	0.617489	0.447061	0.626716	0.402989	0.591181	0.169158	0.319194	0.698235	0.514688	0.0	0.364441	0.350716
2	0.457857	0.617489	0.455721	0.505606	0.481855	0.511042	0.169158	0.319194	0.698235	0.514688	0.0	0.364441	0.522299
3	0.457201	0.529218	0.452685	0.505606	0.481855	0.511042	0.169158	0.319194	0.698235	0.514688	0.0	0.364441	0.573774
4	0.419084	0.264403	0.488508	0.505606	0.481855	0.511042	0.169158	0.319194	0.698235	0.514688	0.0	0.364441	0.745356
...
919	0.361750	0.388242	0.552781	0.329891	0.315685	0.720233	0.168511	0.485125	0.480312	0.183151	1.0	0.663815	0.351056
920	0.587163	0.676600	0.268550	0.746710	0.275125	0.352687	0.282789	0.475992	0.470745	0.157752	1.0	0.773186	0.466548
921	0.555750	0.447928	0.251612	0.496233	0.612773	0.322515	0.597427	0.573346	0.578340	0.572648	1.0	0.302836	0.704686
922	0.637396	0.725769	0.448724	0.714320	0.247787	0.465017	0.517714	0.536217	0.368070	0.434855	1.0	0.460884	0.543937
923	0.657131	0.206771	0.251903	0.628012	0.885300	0.596144	0.612457	0.550550	0.647135	0.426577	1.0	0.443677	0.872919

924 rows × 13 columns

Рисунок 20 – датасет после нормализации

Проведем визуализацию наших данных после обработки.

Построим гистограммы распределения.

```
fig, axes = plt.subplots(nrows = 7, ncols = 2, figsize=(20,50))
l = 0
for i in range(7):
    for j in range(2):
        axes[i, j].hist(df_norm[int_var[l]], bins = 50, color = "seagreen", edgecolor='black', linewidth=1.2)
        axes[i, j].set_title("Гистограмма \n для признака <<{}>>".format(int_var[l]))
        axes[i, j].set_xlabel(int_var[l])
        l+=1
    if l > 12:
        break
```

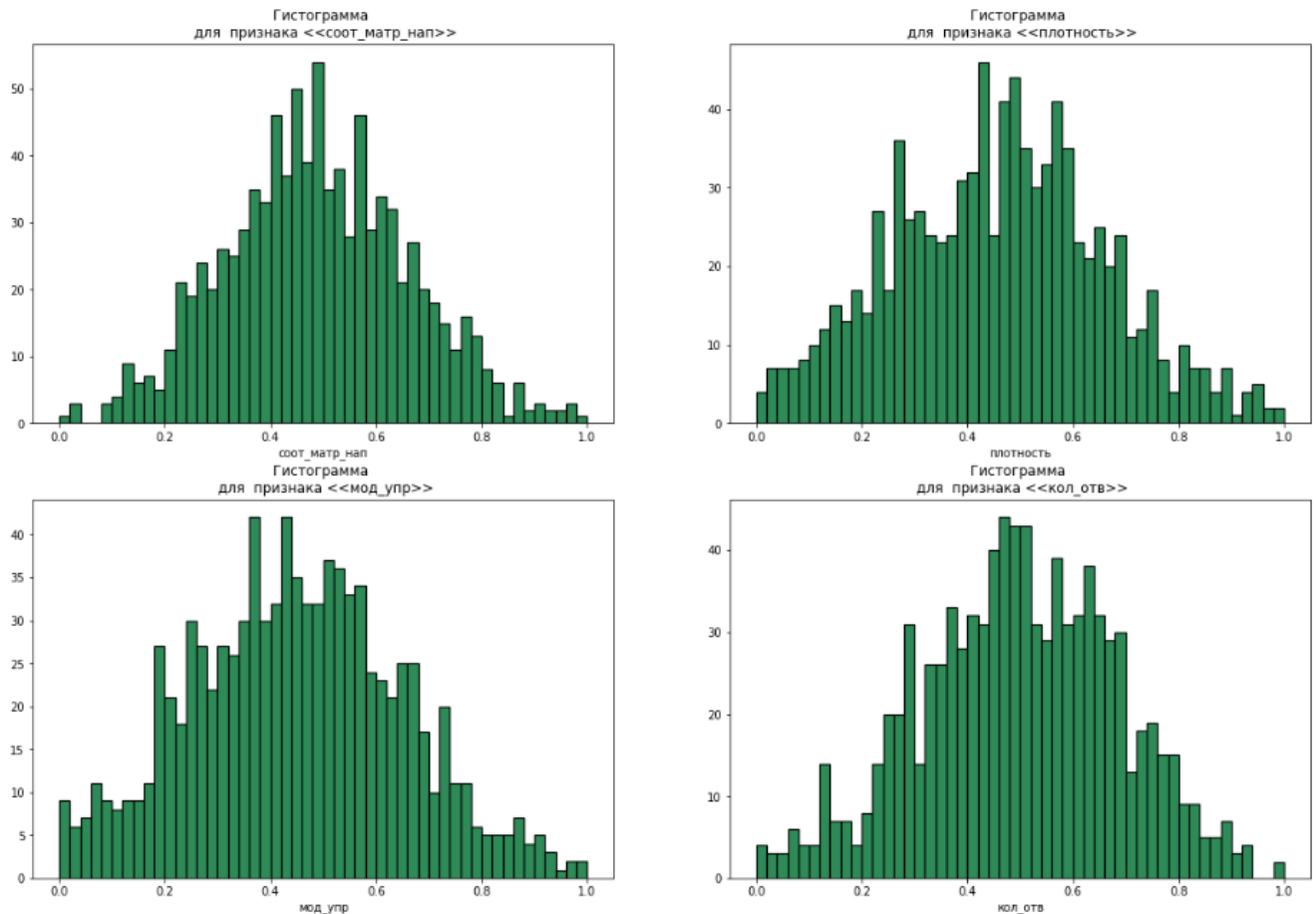


Рисунок 21 – гистограммы распределения признаков после обработки

Визуально мы наблюдаем, что после обработки у наших признаков распределение стало ближе к нормальному.

Построим попарные графики рассеяния и диаграммы ящика с усами.


```
sns.pairplot(df_norm);
```

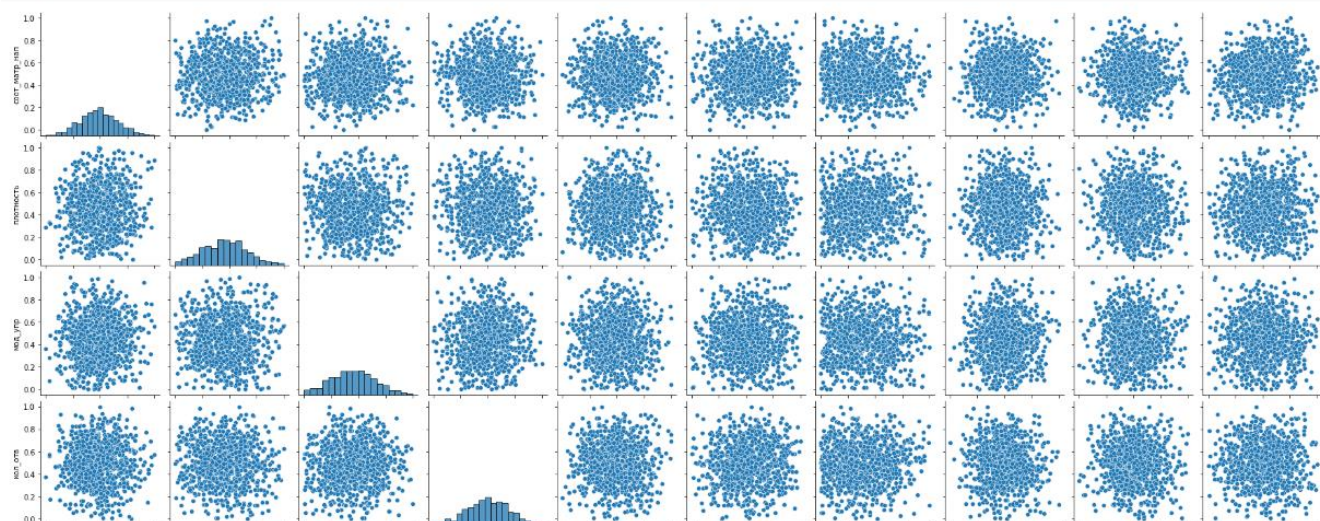


Рисунок 22 – попарные графики рассеяния после обработки

```
fig, axes = plt.subplots(2, sharey=True, figsize=(20,50))
for i, col in enumerate(int_var):
    plt.subplot(7,2,i+1)
    sns.boxplot(data=df_norm[col], whis = 1.5, color = "seagreen")
    plt.title("Ящик с усами \n для признака <<{}>>".format(col))
    plt.xlabel(col)
```

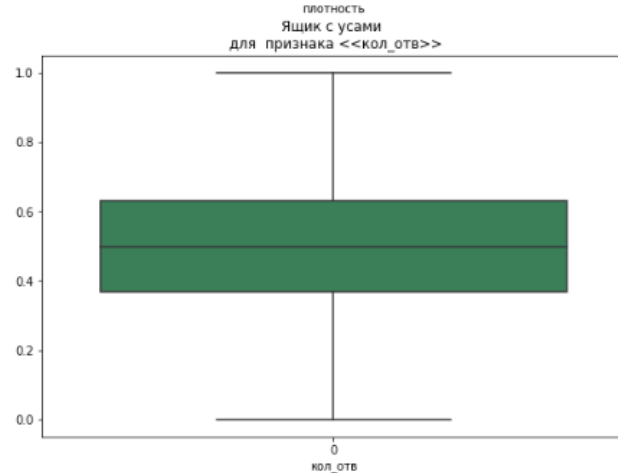
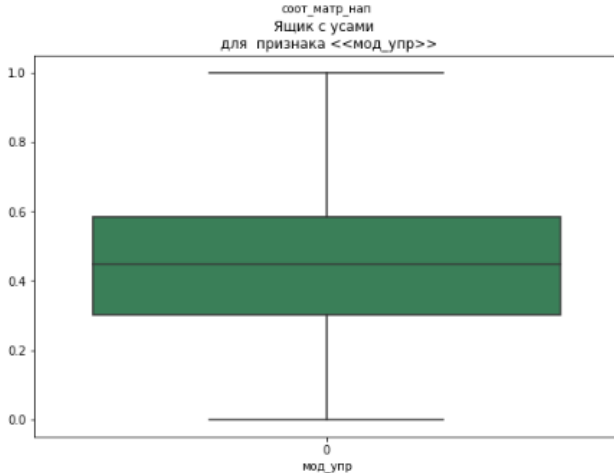
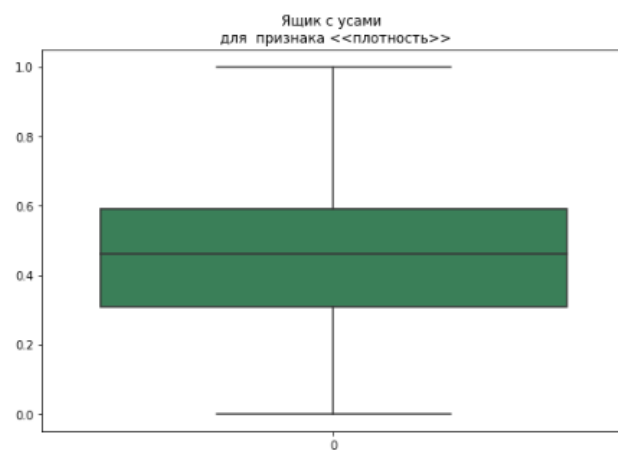
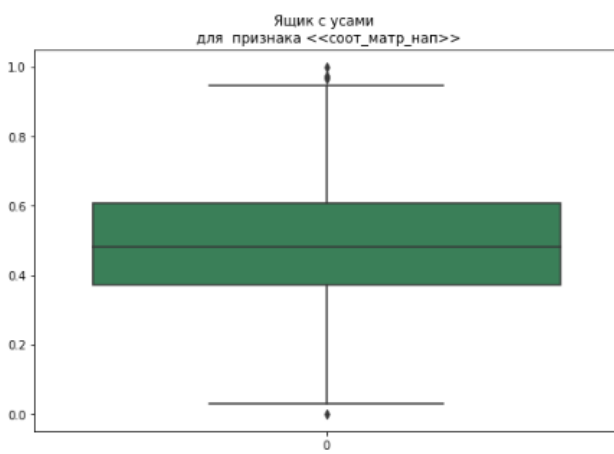


Рисунок 23 – диаграммы ящика с усами после обработки

На диаграммах ящика с усами мы видим, что у тех признаков, которые мы корректировали, выбросы отсутствуют.

Посмотрим на корреляцию наших признаков.

```
corr = df_norm.corr()
corr_10 = corr[corr>=.1]
corr_10
```

	соот_матр_нап	плотность	мод_упр	кол_отв	сод_эп_гр	темп_всп	пов_плотн	мод_упр_раст	проч_раст	потр_смолы	угол_наш	шаг_наш	плотн_наш
соот_матр_нап	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
плотность	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
мод_упр	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
кол_отв	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
сод_эп_гр	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
темп_всп	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
пов_плотн	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN
мод_упр_раст	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN
проч_раст	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
потр_смолы	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN
угол_наш	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	NaN	0.120142
шаг_наш	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN
плотн_наш	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.120142	NaN	1.000000

```
sns.heatmap(corr_10, cmap='Reds', annot=True);
```

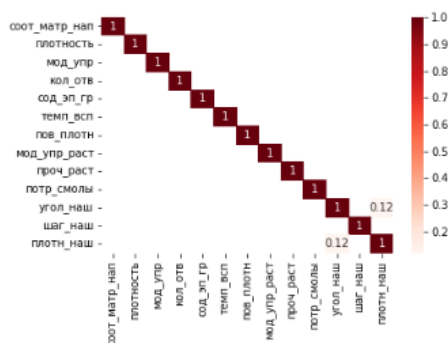


Рисунок 24 – корреляция признаков

Мы видим, что практически все признаки между собой имеют корреляцию менее 0,1.

2.2 Разработка и обучение модели

Разработка и обучение моделей машинного обучения осуществлялась для двух целевых переменных - прочность при растяжении и модуль упругости при растяжении.

Для решения были применены методы, описанные выше. Модели строились отдельно для каждой из целевых переменных.

Для предсказаний мы использовали датасет без наших целевых признаков. То есть признаки соотношение матрица-наполнитель, модуль упругости при растяжении и прочность при растяжении мы убрали из датасета.

Линейная регрессия:

```
# Линейная регрессия

X = df_norm.drop(['соот_матр_нан', 'мод_упр_паст', 'проч_паст'], axis=1)
y_mod = df_norm[['мод_упр_паст']]

X_train, X_test, y_train, y_test = train_test_split(X, y_mod,
                                                    test_size=0.3,
                                                    random_state=0)

lr_mod = LinearRegression()
lr_mod.fit(X_train, y_train)
y_mod_pred = lr_mod.predict(X_test)

lr_mod.coef_

array([[ -0.03930401,  0.01305379, -0.04890765,  0.00927354,  0.01789968,
         0.0393792 ,  0.07879292,  0.01233273, -0.04506183,  0.046259 ]])
```

Рисунок 25 – построение модели линейной регрессии

Метод опорных векторов:

```
# Метод опорных векторов

mov_mod = svm.SVR()
mov_mod.fit(X_train, y_train)
y_mod_pred_mov = mov_mod.predict(X_test)
```

Рисунок 26 – построение модели по методу опорных векторов

Метод градиентного бустинга:


```
# Метод градиентного бустинга

gbr_mod = GradientBoostingRegressor()
gbr_mod.fit(X_train, y_train)
y_mod_pred_gbr = gbr_mod.predict(X_test)
```

Рисунок 27 – построение модели градиентного бустинга

Случайный лес:

```
# Random Forest Regressor

rfr_mod = RandomForestRegressor(n_estimators=20,max_depth=10)
rfr_mod.fit(X_train, y_train)
y_mod_pred_forest = rfr_mod.predict(X_test)
```

Рисунок 28 – построение модели случайного леса

Для модели случайного леса мы применили поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10.

```
parameters = { 'n_estimators': [100, 500],
               'max_depth': [5, 20],
               'max_features': ['auto'],
               'criterion': ['mse'] }
grid = GridSearchCV(estimator = rfr_mod, param_grid = parameters, cv = 10)
grid.fit(X_train, y_train)

GridSearchCV(cv=10,
             estimator=RandomForestRegressor(max_depth=10, n_estimators=20),
             param_grid={'criterion': ['mse'], 'max_depth': [5, 20],
                         'max_features': ['auto'], 'n_estimators': [100, 500]})

grid.best_params_

{'criterion': 'mse',
 'max_depth': 5,
 'max_features': 'auto',
 'n_estimators': 500}
```

Рисунок 29 – поиск гиперпараметров для модели случайного леса

Нужно заметить, что после применения лучших гиперпараметров, модель выдала более хороший результат, чем прежде.

2.3 Тестирование модели

После обучения моделей была проведена оценка точности этих моделей, в качестве параметра оценки модели использовалась средняя абсолютная ошибка (MAE).

Средняя абсолютная ошибка для прогноза модуля упругости при растяжении:

- Линейная регрессия - 0,146.
- Метод опорных векторов - 0,153.
- Метод градиентного бустинга - 0,149.
- Случайный лес - 0,15.
- Случайный лес с подборкой гиперпараметров - 0,147.

Средняя абсолютная ошибка для прогноза прочности при растяжении:

- Линейная регрессия - 0,1467.
- Метод опорных векторов - 0,1528.
- Метод градиентного бустинга - 0,1502.
- Случайный лес - 0,1484.
- Случайный лес с подборкой гиперпараметров - 0,1471.

Рисунок 30 – сравнение эффективности моделей машинного обучения

Лучший результат показала модель линейной регрессии, незначительно уступила ей модель случайного леса с подобранными гиперпараметрами. На третьем месте модель случайного леса до подбора гиперпараметров. Модели, разработанные с применением метода опорных векторов и метода градиентного бустинга показали худшие результаты.

2.4 Нейронная сеть

Обучение нейронной сети - это процесс, в котором параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена. Тип обучения определяется способом подстройки параметров. В процессе обучения

происходит подбор оптимальных параметров модели, с точки зрения минимизации функционала ошибки.

В нашей работе для предсказания соотношения матрица-наполнитель мы будем использовать многослойный персептрон.

```
train_dataset = df.sample(frac=0.8, random_state=0)
test_dataset = df.drop(train_dataset.index)

train_features = train_dataset.copy()
test_features = test_dataset.copy()

train_labels = train_features.pop('соот_матр_нан')
test_labels = test_features.pop('соот_матр_нан')

normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(np.array(train_features))

# Построим многослойный персептрон
def compile_model(norm):
    model = keras.Sequential([
        norm,
        layers.Dense(128, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(1),
    ])
    model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.Adam(0.001))
    return model

itog_model = compile_model(normalizer)
itog_model.summary()

Model: "sequential_2"

```

Layer (type)	Output Shape	Param #
normalization_2 (Normalization)	(None, 12)	25
dense_12 (Dense)	(None, 128)	1664
dense_13 (Dense)	(None, 64)	8256
dense_14 (Dense)	(None, 64)	4160
dense_15 (Dense)	(None, 32)	2080
dense_16 (Dense)	(None, 16)	528
dense_17 (Dense)	(None, 1)	17

```

=====
Total params: 16,730
Trainable params: 16,705
Non-trainable params: 25
=====

```

Рисунок 31 – построение нейронной сети

После построения нейронной сети, обучим ее.

```
%time
history = itog_model.fit(
    train_features,
    train_labels,
    validation_split=0.2,
    verbose=1, epochs=100)

CPU times: user 6 µs, sys: 0 ns, total: 6 µs
Wall time: 11 µs
Epoch 1/100
19/19 [=====] - 1s 12ms/step - loss: 2.3401 - val_loss: 1.1385
Epoch 2/100
19/19 [=====] - 0s 5ms/step - loss: 0.9725 - val_loss: 0.8935
Epoch 3/100
19/19 [=====] - 0s 5ms/step - loss: 0.8481 - val_loss: 0.8500
Epoch 4/100
19/19 [=====] - 0s 5ms/step - loss: 0.8073 - val_loss: 0.8081
Epoch 5/100
19/19 [=====] - 0s 6ms/step - loss: 0.7982 - val_loss: 0.8048
Epoch 6/100
19/19 [=====] - 0s 6ms/step - loss: 0.7598 - val_loss: 0.8001
Epoch 7/100
19/19 [=====] - 0s 4ms/step - loss: 0.7346 - val_loss: 0.7865
Epoch 8/100
19/19 [=====] - 0s 4ms/step - loss: 0.7101 - val_loss: 0.8102
Epoch 9/100
19/19 [=====] - 0s 4ms/step - loss: 0.7368 - val_loss: 0.8274
Epoch 10/100
19/19 [=====] - 0s 4ms/step - loss: 0.6785 - val_loss: 0.7861
Epoch 11/100
19/19 [=====] - 0s 4ms/step - loss: 0.6570 - val_loss: 0.8073
Epoch 12/100
19/19 [=====] - 0s 6ms/step - loss: 0.6352 - val_loss: 0.8141
Epoch 13/100
19/19 [=====] - 0s 4ms/step - loss: 0.6233 - val_loss: 0.8281
Epoch 14/100
19/19 [=====] - 0s 5ms/step - loss: 0.6029 - val_loss: 0.8122
Epoch 15/100
19/19 [=====] - 0s 6ms/step - loss: 0.5792 - val_loss: 0.8603
Epoch 16/100
19/19 [=====] - 0s 5ms/step - loss: 0.5592 - val_loss: 0.8149
Epoch 17/100
19/19 [=====] - 0s 4ms/step - loss: 0.5354 - val_loss: 0.8481
Epoch 18/100
19/19 [=====] - 0s 4ms/step - loss: 0.5260 - val_loss: 0.8135
Epoch 19/100
19/19 [=====] - 0s 4ms/step - loss: 0.5000 - val_loss: 0.8242
Epoch 20/100
19/19 [=====] - 0s 5ms/step - loss: 0.4695 - val_loss: 0.8282
```

Рисунок 32 – обучение нейронной сети

Посмотрим на историю обучения нашей нейронной сети и на график потерь модели на тренировочной и тестовой выборках.

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

	loss	val_loss	epoch
95	0.148348	0.902362	95
96	0.139414	0.883634	96
97	0.129413	0.866384	97
98	0.119439	0.893686	98
99	0.109648	0.879466	99

Рисунок 33 – история обучения нейронной сети

```
plt.figure(figsize = (12,5))
plt.plot(history.history['loss'], label = 'ошибка на обучающей выборке')
plt.plot(history.history['val_loss'], label = 'ошибка на тестовой выборке')
plt.ylim([0, 10])
plt.title('График потерь модели')
plt.ylabel('Значение ошибки')
plt.xlabel('Эпохи')
plt.legend()
plt.grid(True)
plt.show()
```

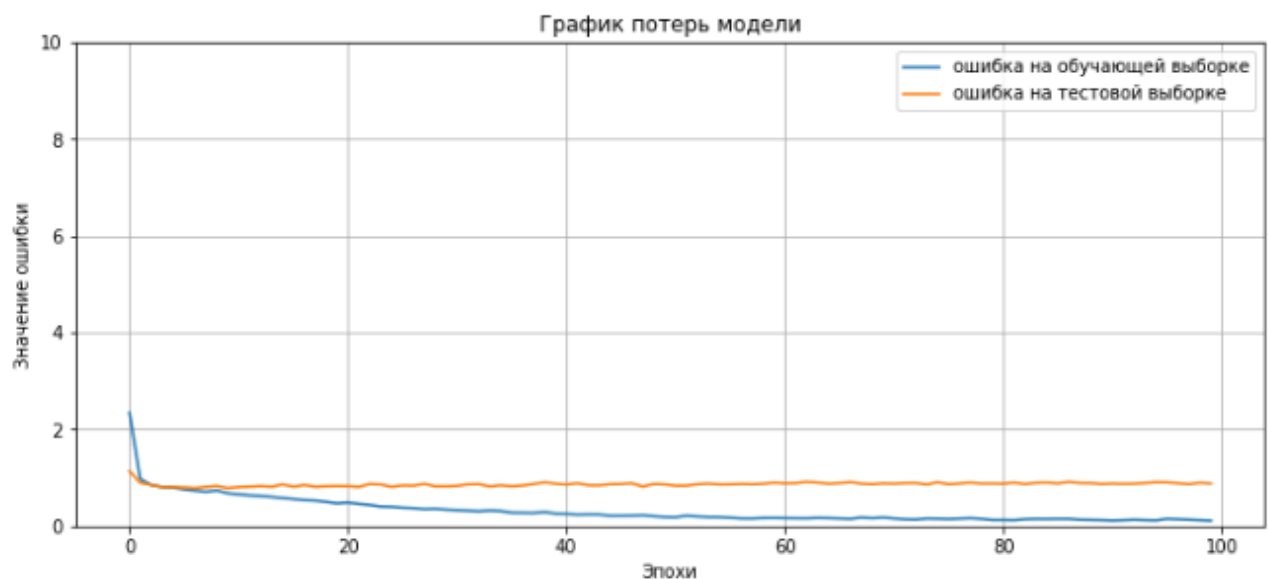


Рисунок 34 – график потерь модели

Из графика мы видим, что на обучающей выборке модель успешно обучилась и показала хороший результат, но на тестовой выборке результат предсказания оставляет желать лучшего. Это также подтверждается и графиком рассеяния.

```
test_predictions = itog_model.predict(test_features).flatten()

plt.figure(figsize = (17,5))
a = plt.axes(aspect = 'equal')
plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
lims = [0, 5]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)
```

6/6 [=====] - 0s 3ms/step

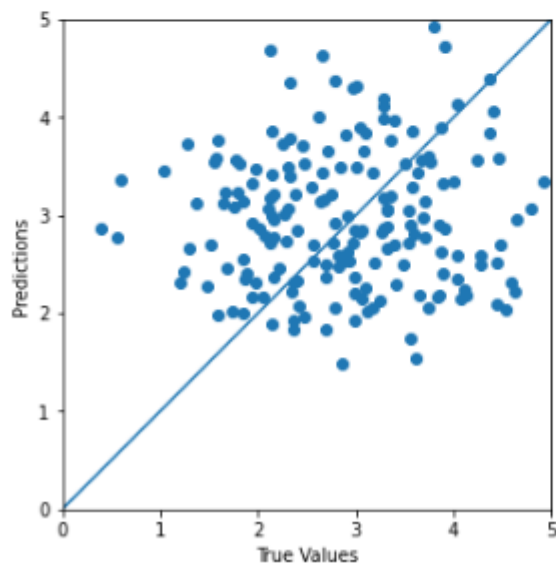
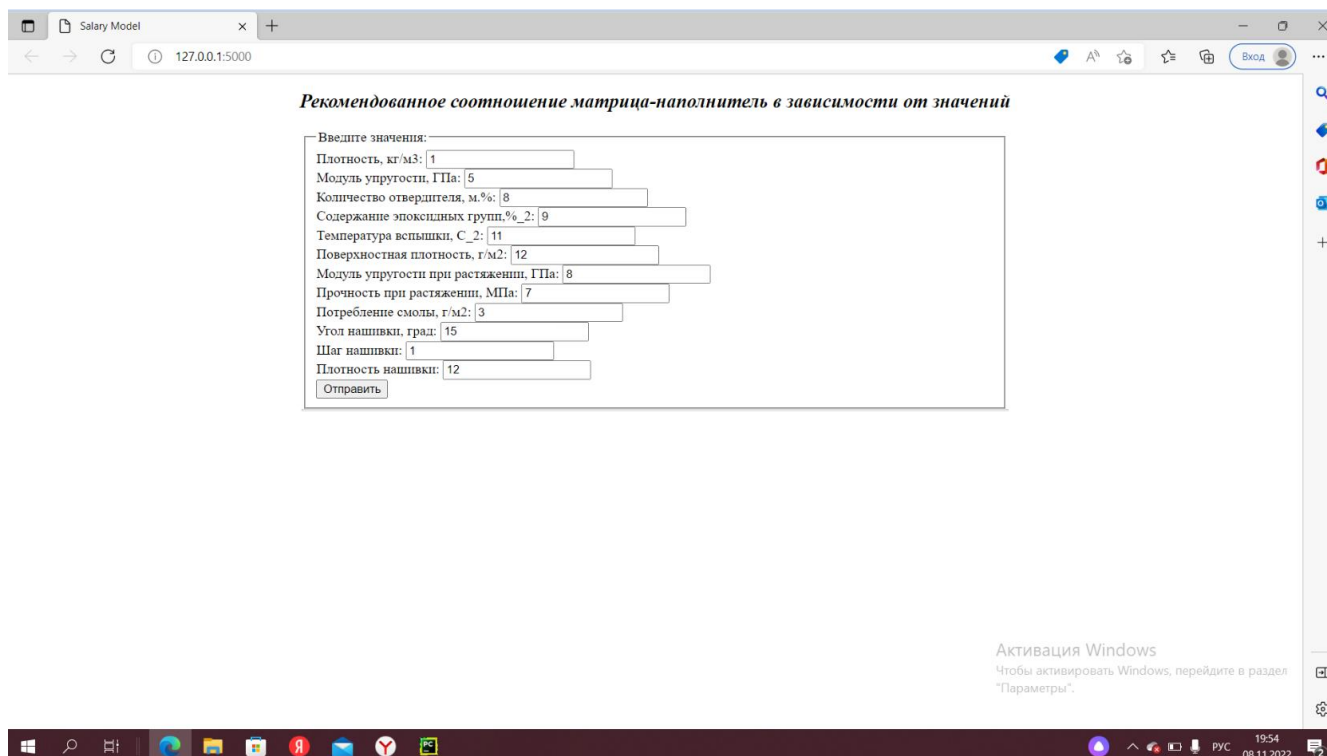


Рисунок 35 – график рассеяния предсказания и реального значения

2.5 Разработка приложения

Следующим этапом нашей работы стала разработка приложения на фреймворке Flask для рекомендации соотношения матрица-наполнитель по заданным пользователем параметрам.

Пользователь вводит в окно ввода значение показателей и нажимает на кнопку «Отправить».



Рекомендованное соотношение матрица-наполнитель в зависимости от значений

Введите значения:

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:

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок 36 – окно ввода пользовательских значений

После этого программа выдает результат с рекомендацией соотношения матрица-наполнитель, рассчитанный на основе сохраненной модели нашей нейронной сети.

Salary Model

127.0.0.1:5000/index

Рекомендованное соотношение матрица-наполнитель в зависимости от значений

Введите значения:

Плотность, кг/м3:

Модуль упругости, ГПа:

Количество отвердителя, м.г:

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:

Рекомендуемое соотношение:

[[18.756414]]

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

19:54
08.11.2022

Рисунок 37 – результат работы приложения

2.6 Создание удалённого репозитория

Для загрузки результатов работы был создан репозиторий github.com по адресу: <https://github.com/OlegZiulikov/composite-forecast>.

Туда были загружены файлы с итоговой работой, файл созданного приложения, html-файл для приложения Flask, а также файлы с набором данных.

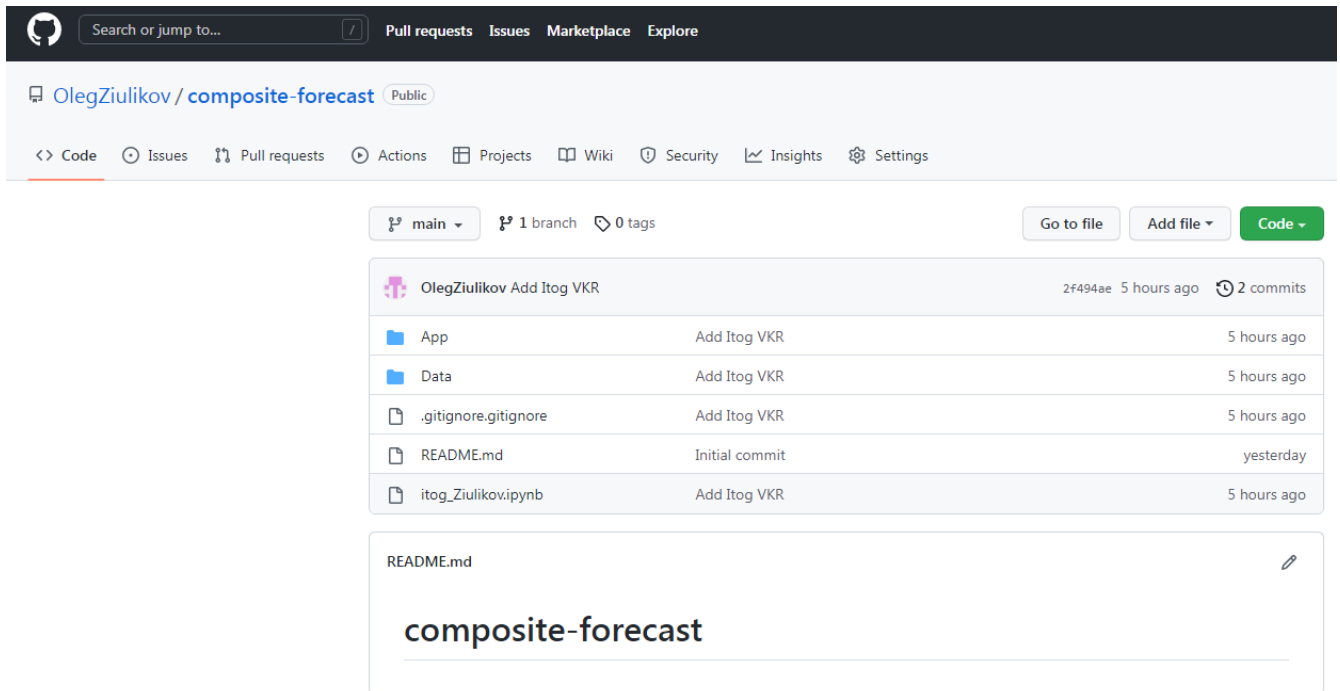


Рисунок 38 – скриншот созданного репозитория

2.7 Заключение

В работа был проведен анализ данных на основе объединения данных из двух файлов. Распределение полученных данных в объединённом датасете близко к нормальному, но коэффициенты корреляции между парами признаков стремятся к нулю. Поэтому применение моделей регрессии было не очень эффективно. Использованные при разработке моделей подходы не позволили получить сколько-нибудь достоверных прогнозов. Лучшей моделью для предсказания модуля упругости при растяжении и прочности при растяжении стала линейная регрессия.

Соотношение матрица – наполнитель тоже не удалось предсказать с высокой точностью. На тренировочной выборке модель показывала хорошие результаты, но на тестовой выборке результат заметно падал.

На основе полученных данных, можно сделать вывод, что для более точного предсказания нужны дополнительные данные, возможно синтезированные из уже

имеющихся путем каких-либо математических преобразований. А также изучение экспертами взаимосвязи различных компонентов и получение из них дополнительного набора данных

2.8 Список используемой литературы

1. Грас Д. Data Science. Наука о данных с нуля: Пер. с англ. - 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2021. - 416 с.: ил.
2. Плас Дж. Вандер, Python для сложных задач: наука о данных и машинное обучение.
3. Скиена, Стивен С. С42 Наука о данных: учебный курс.: Пер. с англ. - СПб.: ООО "Диалектика", 2020. - 544 с. : ил.