ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу

«Data Science»

Слушатель: Шортанова Регина Исфандияровна

Начало работы:

✓Подробный план работы:

★ Составлен подробный план;

★ Изучена теоретическая основа, методы решения и практические составляющие поставленной задачи;

★ Некоторые пункты плана повторялись несколько раз, чтобы добиться лучшего результата;

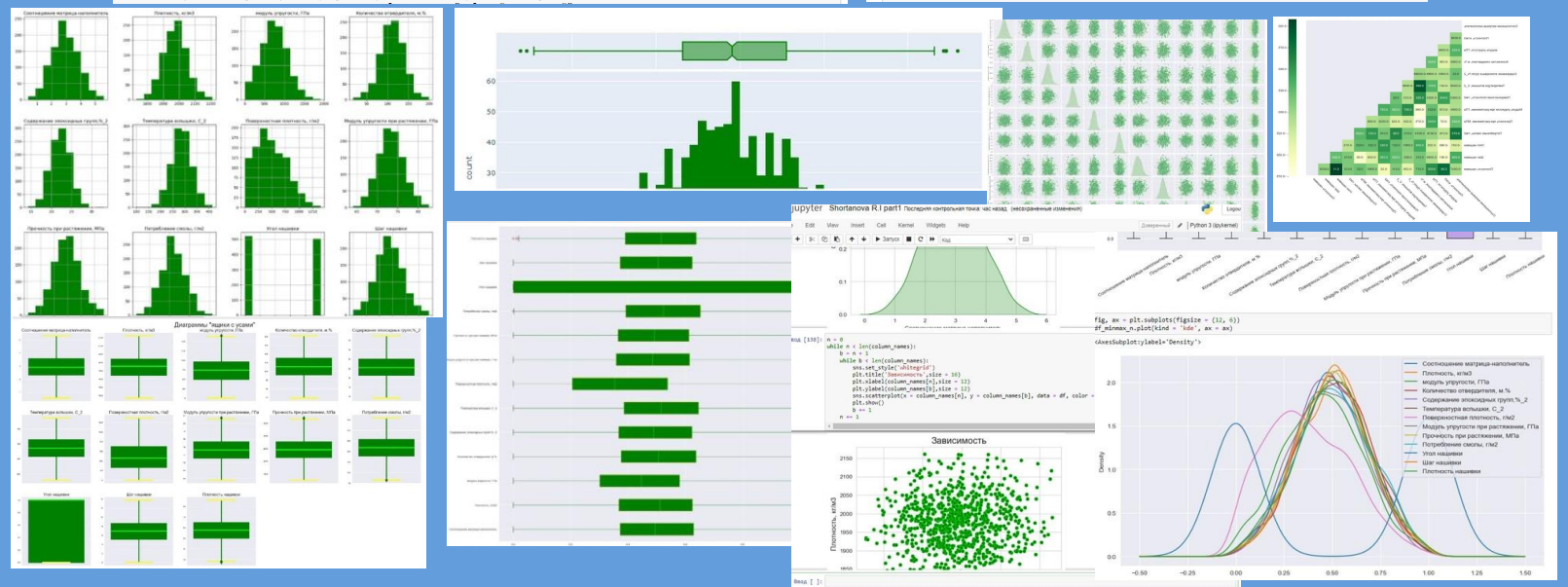
★ Использовано 9 разных методов регрессий для каждой из моделей.

✓Графики:

★ Построенно много графиков;

★ Несколько подобных графиков для одних и тех же переменных.

Приложение 1
Подробный план работы:
1. Загрузить и обработать входящие датасеты
1.1. Удалить неинформативные столбцы
1.2. Объединить датасеты по методу INNER
2. Провести разведочный анализ данных:
2.1. Данные в столбце "Угол нашивки» привести к 0 и 1
2.2. Изучить описательную статистику каждой переменной - "среднее", "медиана", "стандартное отклонение", "минимум", "максимум", "квартили"
2.3. Проверить датасет на пропуски и дубликаты данных
2.4. Получить среднее, медианное значение для каждой колонки
2.5. Вычислить коэффициенты ранговой корреляции Кендалла
2.6. Вычислить коэффициенты корреляции Пирсона
3. Визуализировать разведочный анализ сырых данных (до выбросов и нормализации)
3.1. Построить несколько вариантов гистограмм распределения каждой переменной
3.2. Построить несколько вариантов диаграмм "ящиков с усами" каждой переменной
3.3. Построить гистограмму распределения и диаграмма "ящик с усами" одновременно вместе с данными по каждому столбцу
3.4. Построить несколько вариантов попарных графиков рассеяния точек (матрицы диаграмм рассеяния)
3.5. Построить графики "квантиль-квантиль"
3.6. Построить корреляционную матрицу с помощью тепловой карты
4. Провести предобработку данных (в данном пункте - очистка датасета от выбросов)
4.1. Проверить выбросы по 2 методам: 3-х сигм или межквартильных расстояний
4.2. Посчитать распределение выбросов по каждому столбцу (с целью предотвращения удаления особенностей признака или допущения ошибки)
4.3. Исключить выбросы методом межквартильного расстояния
4.4. Удалить строки с выбросами
4.5. Визуализировать датасет без выбросов, и убедиться, что выбросы еще есть.
4.6. Для полной очистки датасета от выбросов повторить пункты (4.3 – 4.5) ещё 3 раза.
4.7. Сохранить идеальный, без выбросов датасет
4.8. Изучить чистые данные по всем параметрам
4.9. Визуализировать «чистый» датасет (без выбросов)
5. Провести нормализацию и стандартизацию
5.1. Визуализировать плотность ядра
5.2. Нормализовать данные с помощью MinMaxScaler()
5.3. Нормализовать данные с помощью Normalizer()
5.4. Сравнить с данными до нормализации
5.5. Проверить перевод данных из нормализованных в исходные
5.6. Рассмотреть несколько вариантов корреляции между параметрами после нормализации
5.7. Стандартизировать данные
5.8. Визуализировать данные корреляции
5.9. Посмотреть на описательную статистику после нормализации и после стандартизации
6. Разработать и обучить нескольких моделей прогноза прочности при растяжении
6.1. Определить входы и выходы для моделей
6.2. Разобить данные на обучающую и тестовую выборки
6.3. Проверить правильность разбивки
6.4. Построить модели и найти лучшие гиперпараметры
6.5. Построить и визуализировать результат работы метода "опорных векторов"
6.6. Построить и визуализировать результат работы метода "случайного леса"
6.7. Построить и визуализировать результат работы линейной регрессии
6.8. Построить и визуализировать результат работы метода "градиентного бустинга"
6.9. Построить и визуализировать результат работы метода "К ближайших соседей"
6.10. Построить и визуализировать результат работы метода "деревья решений"
6.11. Построить и визуализировать результат работы стохастического градиентного спуска
6.12. Построить и визуализировать результат работы многослойного перцептрона
6.13. Построить и визуализировать результат работы лассо регрессии
6.14. Сравнить модели по метрике MAE
6.15. Найти лучшие гиперпараметры для "случайного леса"
6.16. Подставить значения в модель "случайного леса"
6.17. Найти лучшие гиперпараметры для "К ближайших соседей"
6.18. Подставить значения в модель "К ближайших соседей
6.19. Найти лучшие гиперпараметры метода "деревья решений"
6.20. Подставить значения в модель метода "деревья решений"
6.21. Проверить все модели и процессинги и вывести лучшую модель и процессинг
7. Разработать и обучить нескольких моделей прогноза модуля упругости при растяжении
7.1. Определить входы и выходы для моделей
7.2. Разобить данные на обучающую и тестовую выборки
7.3. Проверить правильность разбивки
7.4. Построить модели и найти лучшие гиперпараметры
7.5. Построить и визуализировать результат работы метода "опорных векторов"
7.6. Построить и визуализировать результат работы метода "случайного леса"
7.7. Построить и визуализировать результат работы линейной регрессии
7.8. Построить и визуализировать результат работы метода градиентного бустинга
7.9. Построить и визуализировать результат работы метода "К ближайших соседей"
7.10. Построить и визуализировать результат работы метода "деревья решений"
7.11. Построить и визуализировать результат работы стохастического градиентного спуска
7.12. Построить и визуализировать результат работы многослойного перцептрона
7.13. Построить и визуализировать результат работы лассо регрессии
7.14. Сравнить модели по метрике MAE
7.15. Найти лучшие гиперпараметры для случайного леса
7.16. Подставить значения в модель "случайного леса"
7.17. Найти лучшие гиперпараметры для "К ближайших соседей"
7.18. Подставить значения в модель "К ближайших соседей"
7.19. Найти лучшие гиперпараметры метода "деревья решений"
7.20. Подставить значения в модель метода "деревья решений"
7.21. Проверить все модели и процессинги и вывести лучшую модель и процессинг
8. Нейронная сеть для рекомендации соотношения матрица-наполнитель
8.1. Сформировать входы и выход для модели
8.2. Нормализовать данные
8.3. Построить модель, определить параметры
8.4. Найти оптимальные параметры для модели
8.5. Посмотреть на результаты
8.6. Повторить шаги 8.4 – 8.5 до построения окончательной модели
8.7. Обучить нейросеть 80/20
8.8. Оценить модель
8.9. Посмотреть на потери модели
8.10. Посмотреть на график результата работы модели
8.11. Посмотреть на график потерь на тренировочной и тестовой выборках
8.12. Сконфигурировать другую модель, задать слои
8.13. Посмотреть на архитектуру другой модели
8.14. Обучить другую модель
8.15. Посмотреть на потери другой модели
8.16. Посмотреть на график потерь на тренировочной и тестовой выборках
8.17. Задать функцию для визуализации факт/прогноз для результатов моделей
8.18. Посмотреть на график результата работы модели
8.19. Оценить модель MSE



ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Объединение файлов и разведочный анализ:

✓Объединение по индексу:

- ★ Импортируем необходимые библиотеки;
- ★ Загружаем файлы;
- ★ Смотрим размерность;
- ★ Объединяем оба файла по индексу по типу объединения INNER

✓Разведочный анализ данных:

- ★ Посмотрим на начальные и конечные строки нашего датасета;
- ★ Изучаем информацию о датасете;
- ★ Проверяем типы данных в каждом столбце;
- ★ Проверяем пропуски;
- ★ Ищем уникальные значения с помощью функции nunique

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import plotly.express as px
import tensorflow as tf
import sklearn

from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error, mean_absolute_error
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer, LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

from tensorflow import keras as keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, Activation
from pandas import read_excel, DataFrame, Series
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from tensorflow.keras.models import Sequential
from numpy.random import seed
from scipy import stats
import warnings
warnings.filterwarnings("ignore")
```

1.2 Объединить датасеты по методу INNER

```
[48]: # Представленные датасеты имеют разный объем строк.
      # Собрать исходные данные файлы в один, единый набор данных.
      df = df_bp.merge(df_nup, left_index = True, right_index = True, how = 'inner')
      df.head().T
```

t[48]:

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

	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	1.857143	2030.0	738.736842	30.00	22.267857	100.000000	210.0	70.0	3000.0	220.0
1	1.857143	2030.0	738.736842	50.00	23.750000	284.615385	210.0	70.0	3000.0	220.0
2	1.857143	2030.0	738.736842	49.90	33.000000	284.615385	210.0	70.0	3000.0	220.0
3	1.857143	2030.0	738.736842	129.00	21.250000	300.000000	210.0	70.0	3000.0	220.0
4	2.771331	2030.0	753.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0	220.0
5	2.767918	2000.0	748.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0	220.0
6	2.569620	1910.0	807.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0	220.0
7	2.561475	1900.0	535.000000	111.86	22.267857	284.615385	380.0	75.0	1800.0	120.0
8	3.557018	1930.0	889.000000	129.00	21.250000	300.000000	380.0	75.0	1800.0	120.0
9	3.532338	2100.0	1421.000000	129.00	21.250000	300.000000	1010.0	78.0	2000.0	300.0

```
[51]: #Просмотрим информацию о датасете, проверим тип данных в каждом столб
      df.info()
      # Пропусков не имеется.Ни одна из записей не является NaN, очистка не
```

```
<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
```

	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	0	4.0	57.0
1	0	4.0	60.0
2	0	4.0	70.0
3	0	5.0	47.0
4	0	5.0	57.0
5	0	5.0	60.0
6	0	5.0	70.0
7	0	7.0	47.0
8	0	7.0	57.0
9	0	7.0	60.0

```
[56]:
```

	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
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	1
1019	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401	350.660830	72.920827	2360.392784	117.730099	1
1020	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047	740.142791	74.734344	2662.906040	236.606764	1
1021	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840	641.468152	74.042708	2071.715856	197.126067	1
1022	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708	758.747882	74.309704	2856.328932	194.754342	1

1023 rows × 13 columns

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

«Угол нашивки» и описательная статистика:

✓Работа со столбцом "Угол нашивки":

- ★ Проверяем количество элементов со значением 0 градусов;
- ★ Приводим к значениям 0 и 1;
- ★ Убеждаемся в неизменном количестве элементов.

✓Описательная статистика:

- ★ Изучим описательную статистику данных (максимальное, минимальное, квартили, медиана, стандартное отклонение, среднее значение и т.д.),
- ★ Посмотрим на основные параметры анализа данных;
- ★ Проверяем датасет на пропущенные и дублирующие данные;
- ★ Вычисляем коэффициенты ранговой корреляции Кендалла и Пирсона

```
Ввод [18]: #Посчитаем количество элементов, где угол нашивки равен 0 градусов
           df['Угол нашивки'][df['Угол нашивки'] == 0.0].count()
           #После преобразования колонки Угол нашивки к значениям 0 и 1, кол-во элементов, где угол нашив
Out[18]: 520

Ввод [19]: # Переведем столбец с нумерацией в integer
           df.index = df.index.astype('int')

Ввод [20]: # Сохраним итоговый датасет в отдельную папку с данными
           df.to_excel(r'C:\Users\user\Desktop\МГТУ учеба\vkr\itogoviidataset\itogoviidataset.xlsx')
```

[56]:

	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
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	1
1019	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401	350.660830	72.920827	2360.392784	117.730099	1
1020	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047	740.142791	74.734344	2662.906040	236.606764	1
1021	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840	641.468152	74.042708	2071.715856	197.126067	1
1022	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708	758.747882	74.309704	2856.328932	194.754342	1

1023 rows × 13 columns

```
# Пропуски данных

# Проверим на пропущенные данные
df.isnull().sum()
# Пропущенных данных нет = нулевых значений
```

```
Соотношение матрица-наполнитель          0
Плотность, кг/м3                         0
модуль упругости, ГПа                    0
Количество отвердителя, м.%              0
Содержание эпоксидных групп,%_2          0
Температура вспышки, С_2                 0
Поверхностная плотность, г/м2            0
Модуль упругости при растяжении, ГПа     0
Прочность при растяжении, МПа            0
Потребление смолы, г/м2                  0
Угол нашивки                             0
Шаг нашивки                              0
Плотность нашивки                        0
dtype: int64
```

```
# Поработаем со столбцом "Угол нашивки"

df['Угол нашивки, град'].nunique()
#Так как кол-во уникальных значений в колонке Угол нашивки равно 2
```

```
2
```

```
#Проверим кол-во элементов, где Угол нашивки равен 0 градусов
df['Угол нашивки, град'][df['Угол нашивки, град'] == 0.0].count()
```

```
520
```

```
# Приведем столбец "Угол нашивки" к значениям 0 и 1 и integer
df = df.replace({'Угол нашивки, град': {0.0 : 0, 90.0 : 1}})
df['Угол нашивки, град'] = df['Угол нашивки, град'].astype(int)

#Переименуем столбец
df = df.rename(columns={'Угол нашивки, град' : 'Угол нашивки'})
df
```

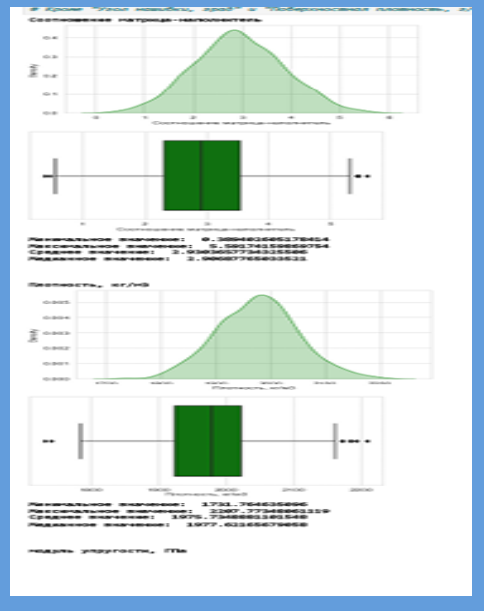
```
a = df.describe()
a.T
```

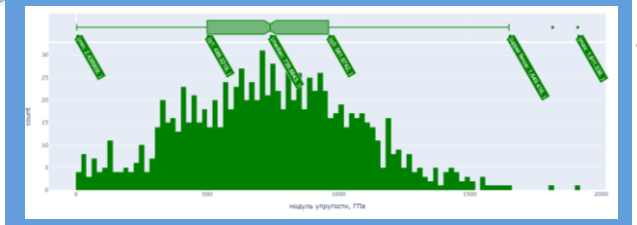
	count	mean	std	min	25%	50%	75%	max
Соотношение матрица-наполнитель	1023.0	2.930366	0.913222	0.389403	2.317887	2.906878	3.552660	5.591742
Плотность, кг/м3	1023.0	1975.734888	73.729231	1731.764635	1924.155467	1977.621657	2021.374375	2207.773481
модуль упругости, ГПа	1023.0	739.923233	330.231581	2.436909	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	1023.0	0.491691	0.500175	0.000000	0.000000	0.000000	1.000000	1.000000
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

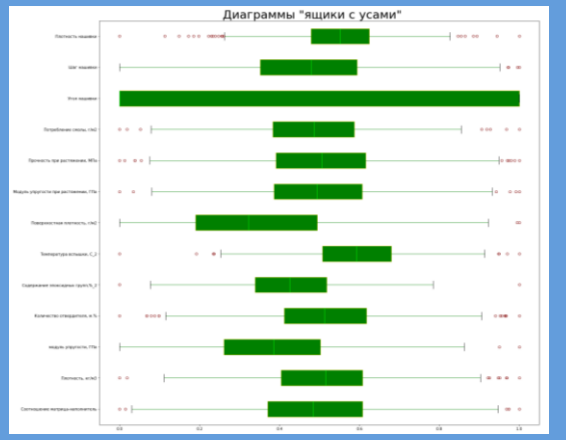
Визуализация «сырых» данных:

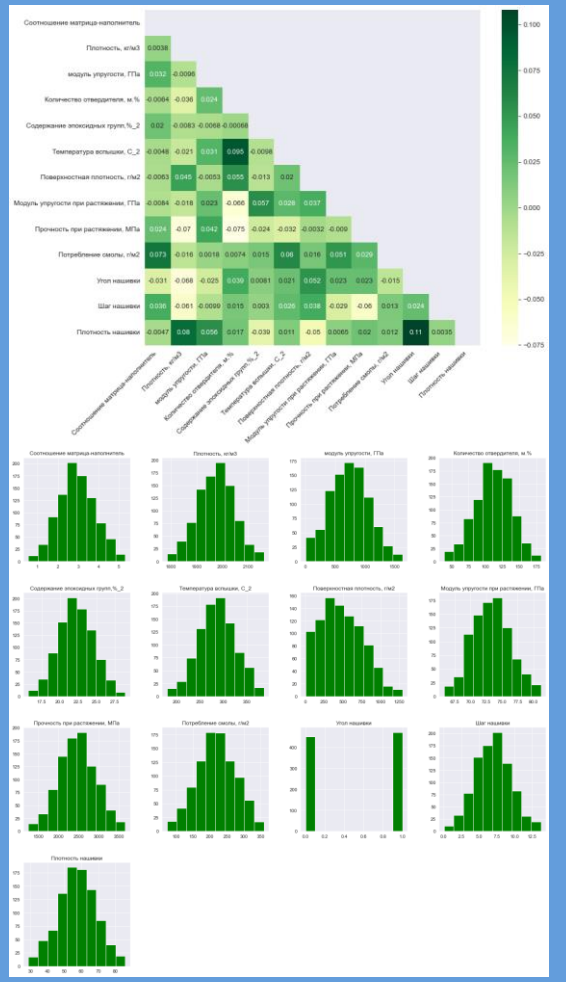
✓**Графики без нормализации и исключения шумов :**

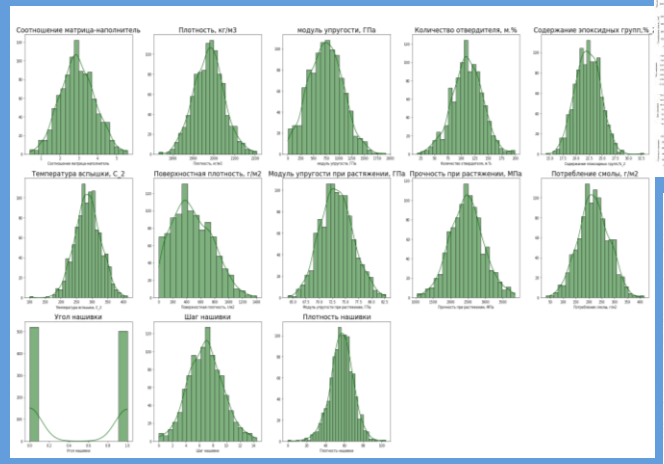
- ★ Построим гистограммы распределения каждой из переменных (несколько вариантов);
- ★ Диаграммы "ящиков с усами" (несколько вариантов);
- ★ Попарные графики рассеяния точек (несколько вариантов);
- ★ Графики квантиль-квантиль; тепловые карты (несколько вариантов).

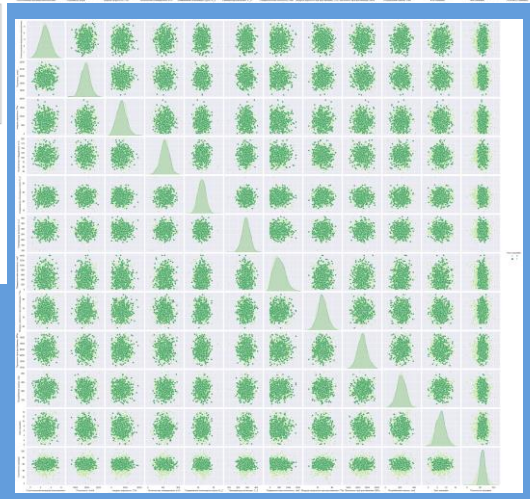


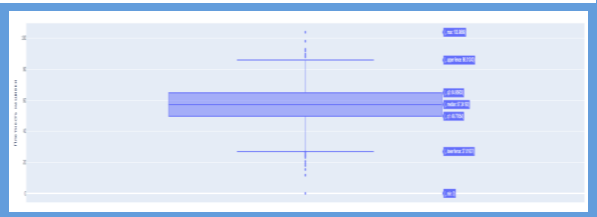










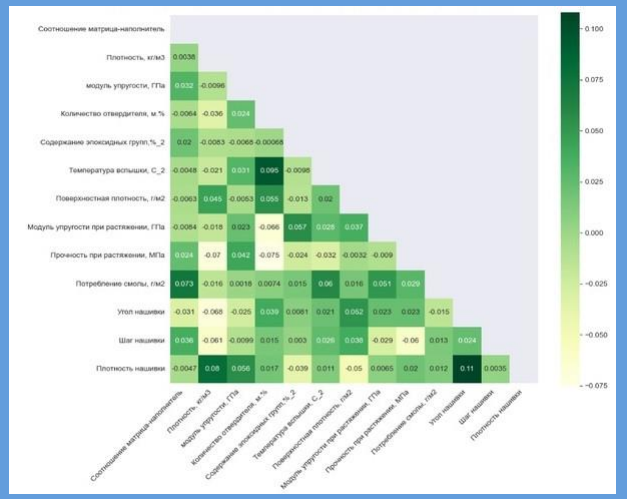


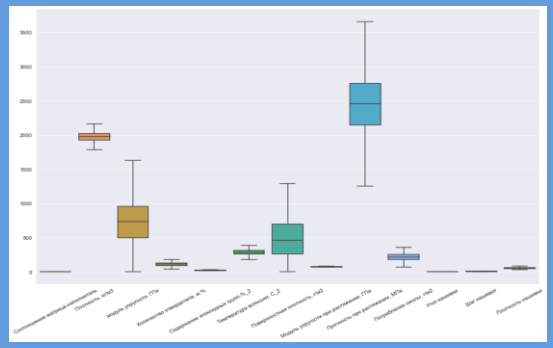
ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

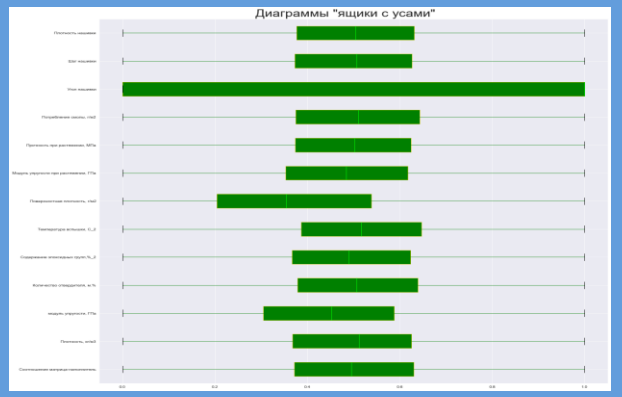
Предобработка данных:

✓Исключение выбросов:

- ★ Посчитаем количество значений методом 3 сигм и методом межквартильных расстояний;
- ★ Исключаем выбросы методом межквартильного расстояния ;
- ★ Проверяем результат;
- ★ Строим графики;
- ★ Проверяем на наличие оставшихся выбросов;
- ★ Повторяем удаление выбросов ещё 4 раза до полного удаления;
- ★ Проверяем чистоту датасета от выбросов;
- ★ Строим все возможные графики «чистого» датасета.








```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 922 entries, 1 to 1022
Data columns (total 13 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Соотношение матрица-наполнитель    922 non-null    float64
 1   Плотность, кг/м3                   922 non-null    float64
 2   модуль упругости, ГПа              922 non-null    float64
 3   Количество отвердителя, м.%        922 non-null    float64
 4   Содержание эпоксидных групп,%_2    922 non-null    float64
 5   Температура вспышки, С_2           922 non-null    float64
 6   Поверхностная плотность, г/м2      922 non-null    float64
 7   Модуль упругости при растяжении, ГПа  922 non-null  float64
 8   Прочность при растяжении, МПа      922 non-null    float64
 9   Потребление смолы, г/м2            922 non-null    float64
 10  Угол нашивки                       922 non-null    int32
 11  Шаг нашивки                        922 non-null    float64
 12  Плотность нашивки                  922 non-null    float64
dtypes: float64(12), int32(1)
memory usage: 129.5 KB
```

```
Соотношение матрица-наполнитель          6
Плотность, кг/м3                         9
модуль упругости, ГПа                    2
Количество отвердителя, м.%             14
Содержание эпоксидных групп,%_2          2
Температура вспышки, С_2                 8
Поверхностная плотность, г/м2            2
Модуль упругости при растяжении, ГПа     6
Прочность при растяжении, МПа           11
Потребление смолы, г/м2                  8
Угол нашивки                             0
Шаг нашивки                              4
Плотность нашивки                        21
dtype: int64
```

```
#Для удаления выбросов существует 2 основных метода - метод 3-х с
metod_3s = 0
metod_iq = 0
count_iq = [] # Список, куда записывается количество выбросов по
count_3s = [] # Список, куда записывается количество выбросов по
for column in df:
    d = df.loc[:, [column]]
    # методом 3-х сигм
    zscore = (df[column] - df[column].mean()) / df[column].std()
    d['3s'] = zscore.abs() > 3
    metod_3s += d['3s'].sum()
    count_3s.append(d['3s'].sum())
    print(column,'3s', ': ', d['3s'].sum())

    # методом межквартильных расстояний
    q1 = np.quantile(df[column], 0.25)
    q3 = np.quantile(df[column], 0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    d['iq'] = (df[column] <= lower) | (df[column] >= upper)
    metod_iq += d['iq'].sum()
    count_iq.append(d['iq'].sum())
    print(column, ': ', d['iq'].sum())
print('Метод 3-х сигм, выбросов:', metod_3s)
print('Метод межквартильных расстояний, выбросов:', metod_iq)
```

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

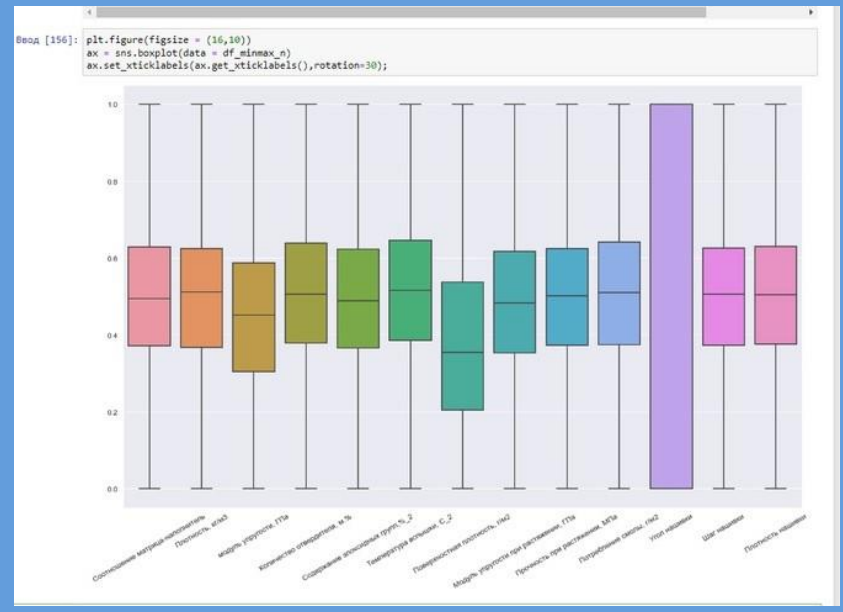
Предобработка данных:

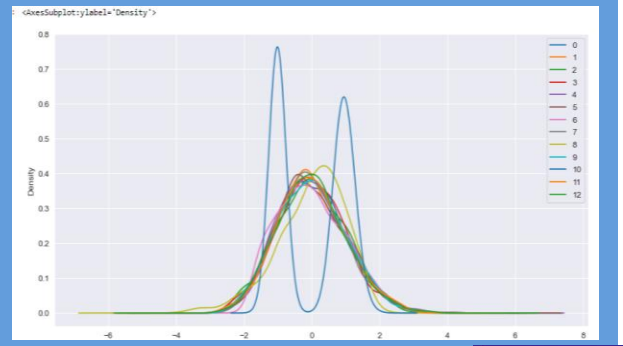
✓Нормализация данных:

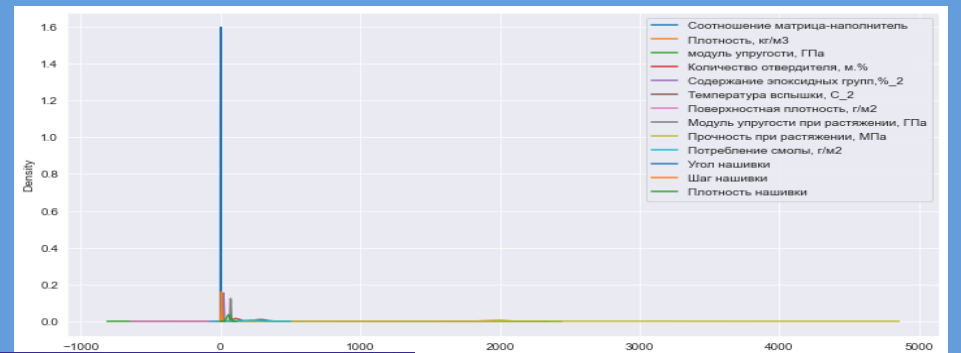
- ★ Нормализуем данные MinMaxScaler();
- ★ Построим график плотности ядра;
- ★ Проверим результат MinMaxScaler();
- ★ Построим графики MinMaxScaler();
- ★ Нормализуем данные с помощью Normalizer();
- ★ Проверим результат Normalizer();
- ★ Построим графики Normalizer().

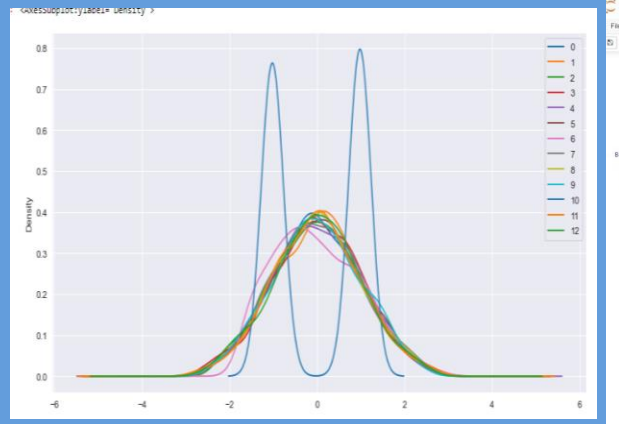
✓Стандартизация данных:

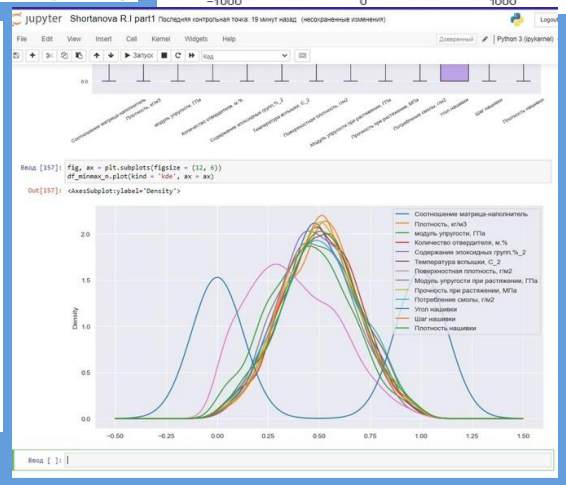
- ★ Стандартизируем данные с помощью StandardScaler();
- ★ Проверим результат StandardScaler();
- ★ Построим графики StandardScaler().

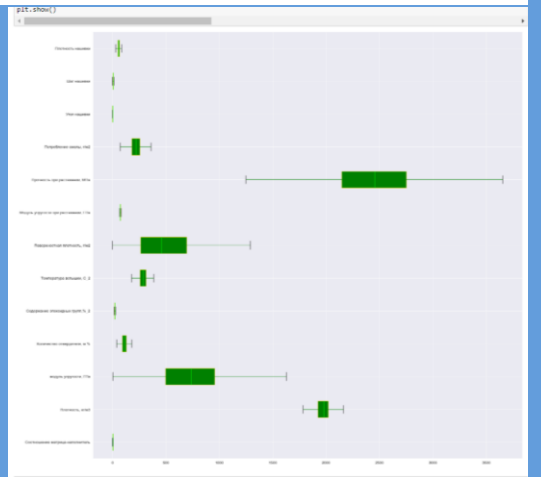












ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Разработка и обучение моделей для прогноза прочности при растяжении:

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

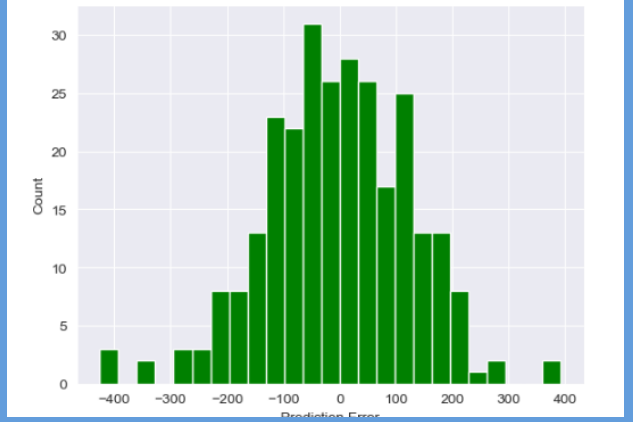
✓Метод К ближайших соседей:

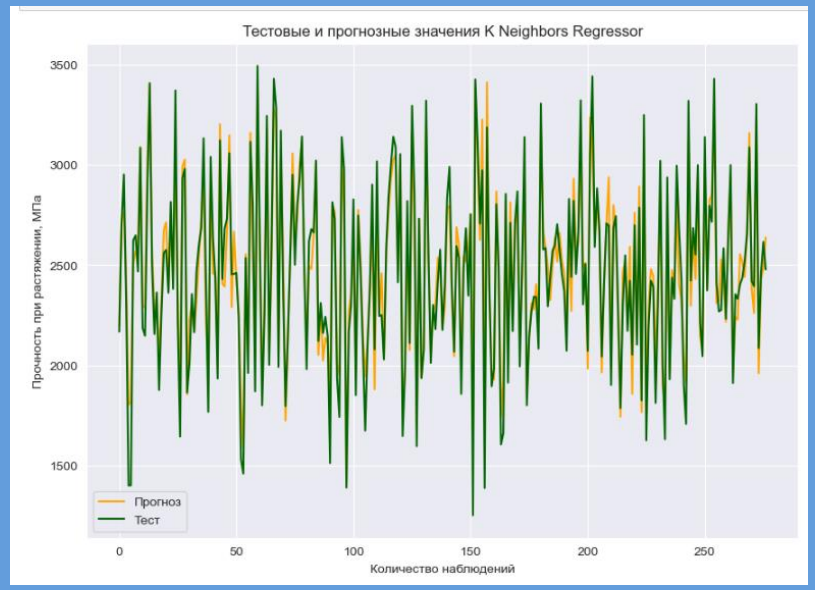
- ★ Разбиваем данные на тестовую и тренировочную выборки;
- ★ Обучаем модель;
- ★ Вычисляем коэффициент детерминации;
- ★ Считаем MAE, MAPE, MSE, RMSE, test score train и test score test;
- ★ Сравниваем с результатами модели, выдающей среднее значение;
- ★ Строим графики для тестовых и прогнозных значений;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

6.9. Построим и визуализируем результат работы метода К ближайших соседей

```
Ввод [139]: knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train_1, y_train_1)
y_pred_knn = knn.predict(x_test_1)
mae_knr = mean_absolute_error(y_pred_knn, y_test_1)
mse_knn_elast = mean_squared_error(y_test_1,y_pred_knn)
print('K Neighbors Regressor  Results Train:')
print("Test score: {:.2f}".format(knn.score(x_train_1, y_train_1)))# Скор для тренировочной выборки
print('K Neighbors Regressor  Results:')
print('KNN_MAE: ', round(mean_absolute_error(y_test_1, y_pred_knn)))
print('KNN_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_knn)))
print('KNN_MSE: {:.2f}'.format(mse_knn_elast))
print("KNN_RMSE: {:.2f}".format (np.sqrt(mse_knn_elast)))
print("Test score: {:.2f}".format(knn.score(x_test_1, y_test_1)))# Скор для тестовой выборки

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
```

```
#Визуализация гистограммы распределения ошибки
error = y_test_1 - y_pred_knn
plt.hist(error, bins = 25, color = "g")
plt.xlabel('Prediction Error')
_ = plt.ylabel('Count')
```






```
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
```

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

✓Для метода «Деревья решений»:

- ★ 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.

6.10. Построим и визуализируем результат работы метода деревья решений

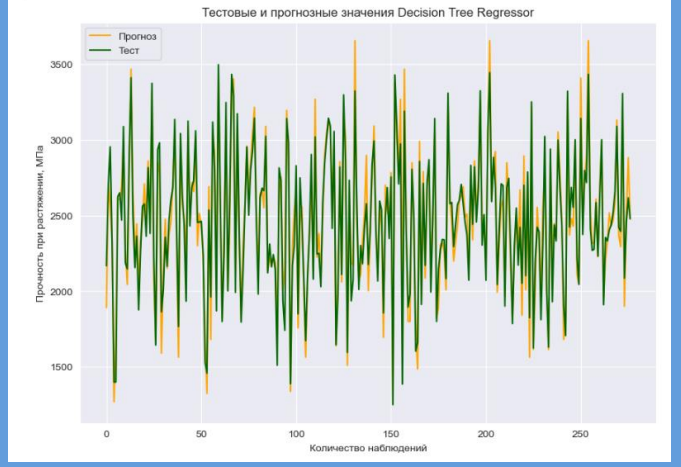
Ввод [142]:
```
dtr = DecisionTreeRegressor()
dtr.fit(x_train_1, y_train_1.values)
y_pred_dtr = dtr.predict(x_test_1)
mae_dtr = mean_absolute_error(y_pred_dtr, y_test_1)
mse_dtr_elast = mean_squared_error(y_test_1,y_pred_dtr)
print('Decision Tree Regressor Results Train:')
print("Test score: {:.2f}".format(knn.score(x_train_1, y_train_1)))# Скор для тренировочной выборки
print('Decision Tree Regressor Results:')
print('DTR_MAE: ', round(mean_absolute_error(y_test_1, y_pred_dtr)))
print('DTR_MSE: {:.2f}'.format(mse_dtr_elast))
print("DTR_RMSE: {:.2f}".format (np.sqrt(mse_dtr_elast)))
print('DTR_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_dtr)))
print("Test score: {:.2f}".format(dtr.score(x_test_1, y_test_1)))# Скор для тестовой выборки
```

```
Decision Tree Regressor Results Train:
Test score: 0.94
Decision Tree Regressor Results:
DTR_MAE:  103
DTR_MSE: 17320.98
DTR_RMSE: 131.61
DTR_MAPE: 0.04
Test score: 0.92
```

Out[166]:

	Perpeccop	MAE
0	Support Vector	78.477914
1	RandomForest	76.589025
2	Linear Regression	61.986894
3	GradientBoosting	65.140165
4	KNeighbors	102.030259
5	DecisionTree	102.722429
6	SGD	181.710781
7	MLP	1808.547264
8	Lasso	69.474334
9	RandomForest_GridSearchCV	68.326303
10	KNeighbors_GridSearchCV	99.281694
11	DecisionTree_GridSearchCV	168.624997

```
plt.figure(figsize = (10, 7))
plt.title("Тестовые и прогнозные значения Decision Tree Regressor")
plt.plot(y_pred_dtr, label = "Прогноз", color = 'orange')
plt.plot(y_test_1.values, label = "Тест", color = 'darkgreen')
plt.xlabel("Количество наблюдений")
plt.ylabel("Прочность при растяжении, МПа")
plt.legend()
plt.grid(True);
```



6.19. Найдём лучшие гиперпараметры метода деревья решений

Ввод [163]:
```
criterion = ['squared_error', 'friedman_mse', 'absolute_error', 'poisson']
splitter = ['best', 'random']
max_depth = [3,5,7,9,11]
min_samples_leaf = [100,150,200]
min_samples_split = [200,250,300]
max_features = ['auto', 'sqrt', 'log2']
param_grid = {'criterion': criterion,
              'splitter': splitter,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf,
              'max_features': max_features}
#Запустим обучение модели. В качестве оценки модели будем использовать коэффициент детерминации (R^2)
# Если R2<0, это значит, что разработанная модель даёт прогноз даже хуже, чем простое усреднение.
gs4 = GridSearchCV(dtr, param_grid, cv = 10, verbose = 1, n_jobs =-1, scoring = 'r2')
gs4.fit(x_train_1, y_train_1)
dtr_3 = gs4.best_estimator_
gs.best_params_
```

```
Fitting 10 folds for each of 1080 candidates, totalling 10800 fits
```

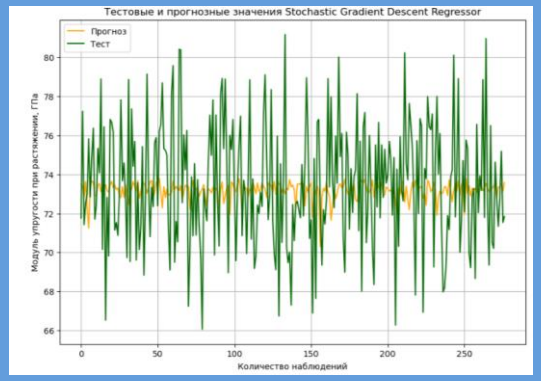
Out[163]: {'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'distance'}

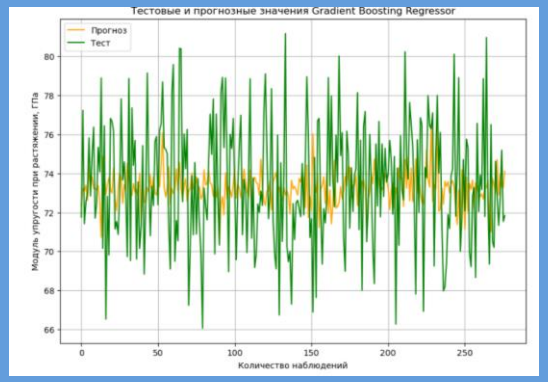
ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

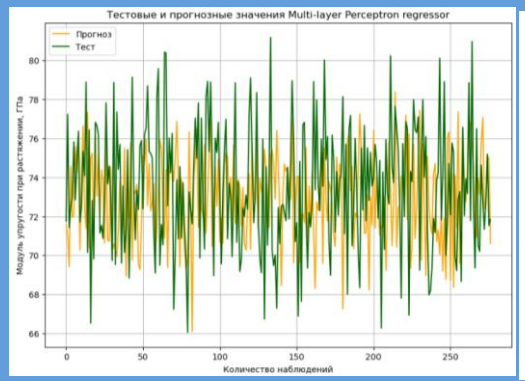
Разработка и обучение моделей для прогноза модуль упругости при растяжении:

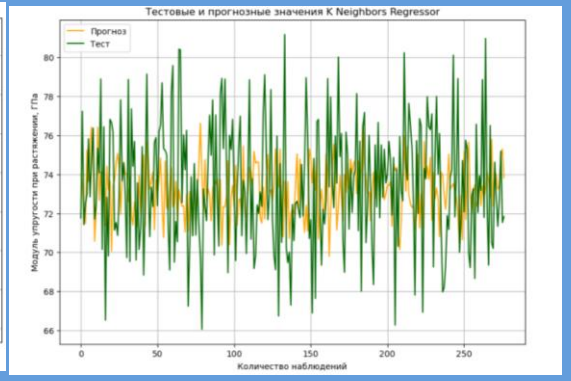
✓**Графики тестовых и прогнозных значений для разных методов (слева - направо и сверху – вниз):**

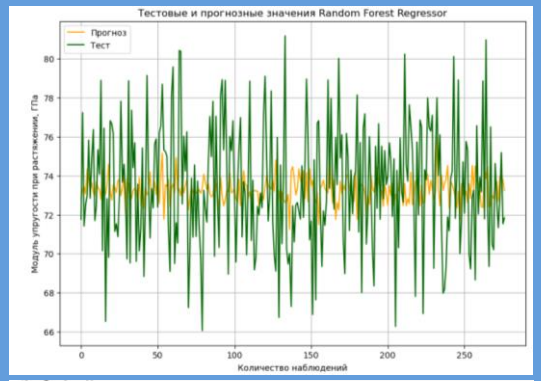
- ★ Метод опорных векторов;
- ★ Линейная регрессия;
- ★ Стохастический градиентный спуск;
- ★ Многослойный перцептрон;
- ★ К-ближайших соседей;
- ★ Градиентный бустинг;
- ★ «Случайный лес»;
- ★ Дерево принятия решений;
- ★ Лассо.

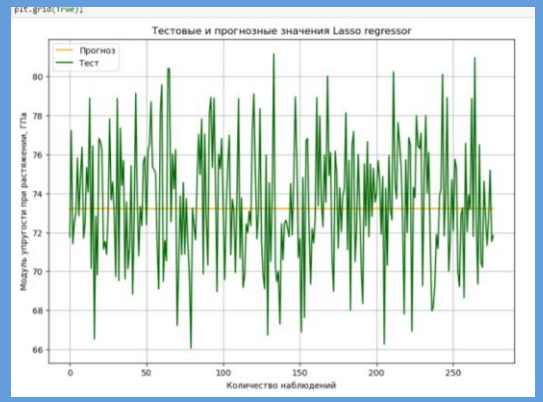


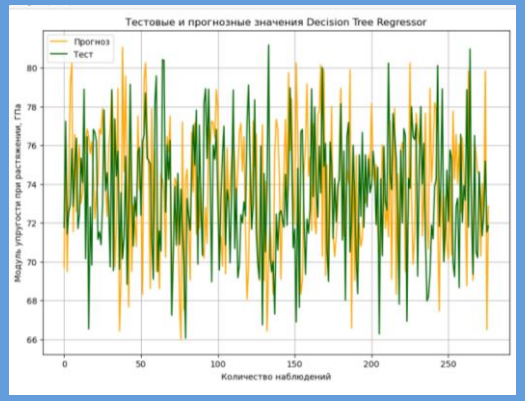


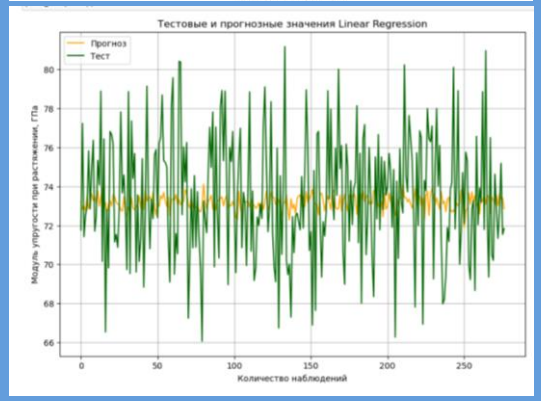












ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Поиск гиперпараметров: для прогноза модуль упругости при растяжении:

✓Для метода «Случайный лес»:

- ★ Поиск гиперпараметров методом GridSearchCV с перекрёстной проверкой с количеством блоков 10;
- ★ Вывод гиперпараметров 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

Ввод [41]:
```
mae_df
```

Out[41]:

	Perpeccop	MAE
0	Support Vector	3.450178
1	RandomForest	2.669425
2	Linear Regression	2.612429
3	GradientBoosting	2.667008
4	KNeighbors	2.818041
5	DecisionTree	3.571074
6	SGD	2.607170
7	MLP	3.133701
8	Lasso	2.580193
9	RandomForest1_GridSearchCV	2.631907

[50]:
```
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=1, gamma=1), 'regressor__C': 1, 'regressor__gamma': 1}

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

[51]:
```
print("Наилучшая модель:\n{}".format(grid2.best_estimator_))
```

```
Наилучшая модель:
Pipeline(steps=[('preprocessing', StandardScaler()),
                ('regressor', SVR(C=1, gamma=1))])
```

После обучения моделей была проведена оценка точности этих моделей на обучающей и тестовых выборках.
В качестве параметра оценки модели использовалась средняя абсолютная ошибка (MAE).
Обе модели даже на тренировочном датасете не смогли обучиться и приблизиться к исходным данным.
Поэтому ошибка на тестовом датасете выше.

7.16. Подставим значения в нашу модель случайного леса

Ввод [39]:
```
rfr21_grid = RandomForestRegressor(n_estimators=200, criterion='mse', max_depth=15, max_features='auto')
#Обучаем модель
rfr21_grid.fit(x_train_2, y_train_2)

predictions_rfr21_grid = rfr21_grid.predict(x_test_2)
#Оценивaем точность на тестовом наборе
mae_rfr21_grid = mean_absolute_error(predictions_rfr21_grid, y_test_2)
mae_rfr21_grid
```

Out[39]: 2.631907012135688

Ввод [40]:
```
new_row_in_mae_df = {'Perpeccop': 'RandomForest1_GridSearchCV', 'MAE': mae_rfr21_grid}

mae_df = mae_df2.append(new_row_in_mae_df, ignore_index = True)
```

Ввод [41]:
```
mae_df
```

Out[41]:

	Perpeccop	MAE
0	Support Vector	3.450178
1	RandomForest	2.669425
2	Linear Regression	2.612429
3	GradientBoosting	2.667008
4	KNeighbors	2.818041
5	DecisionTree	3.571074
6	SGD	2.607170
7	MLP	3.133701
8	Lasso	2.580193
9	RandomForest1_GridSearchCV	2.631907

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.

```
# построение окончательной модели
model = create_model(lyrs=[128, 64, 16, 3], dr=0.05)

print(model.summary())
```

```
Model: "sequential_195"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_493 (Dense)           (None, 128)               1792

 dense_494 (Dense)           (None, 64)                8256

 dense_495 (Dense)           (None, 16)                1040

 dense_496 (Dense)           (None, 3)                 51

 dropout_195 (Dropout)       (None, 3)                 0

 dense_497 (Dense)           (None, 3)                 12

=================================================================
Total params: 11,151
Trainable params: 11,151
Non-trainable params: 0
_________________________________________________________________
None
```

8.2. Нормализуем данные

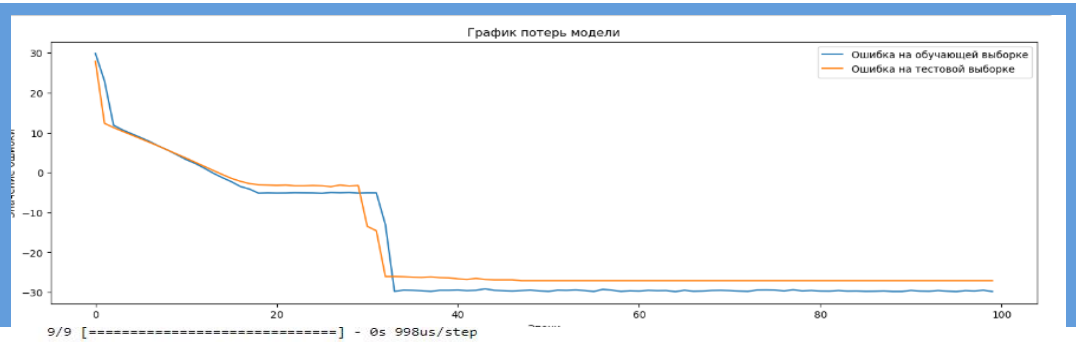
Ввод [55]:
```
x_train_n = tf.keras.layers.Normalization(axis =-1)
x_train_n.adapt(np.array(x_train))
```

Ввод [56]:
```
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
```

8.8. Оценим модель

Ввод [69]:
```
scores = model.evaluate(x_test, y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

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

mae: 190.57%
```



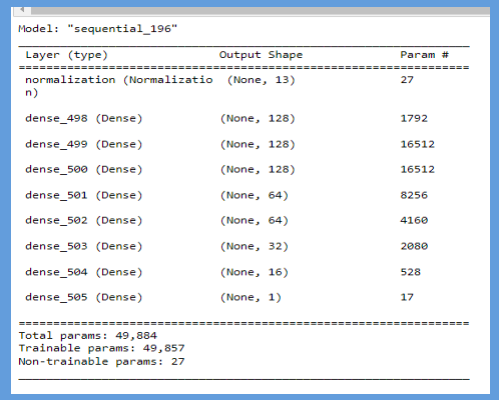

```
9/9 [==============================] - 0s 998us/step
```

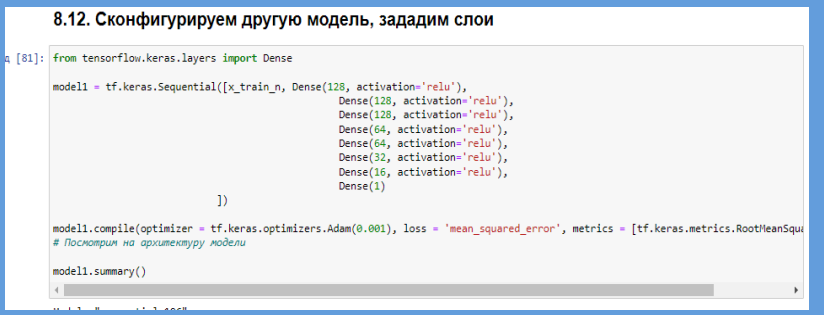
ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

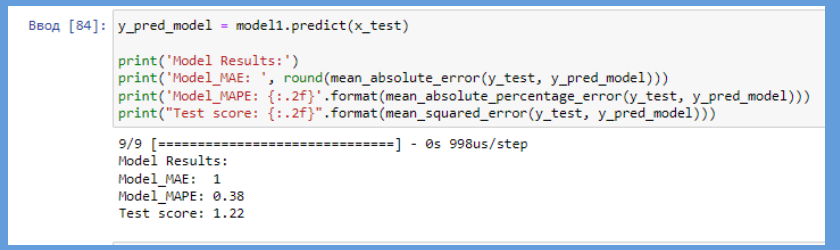
Нейронная сеть для соотношения «матрица-наполнитель»:

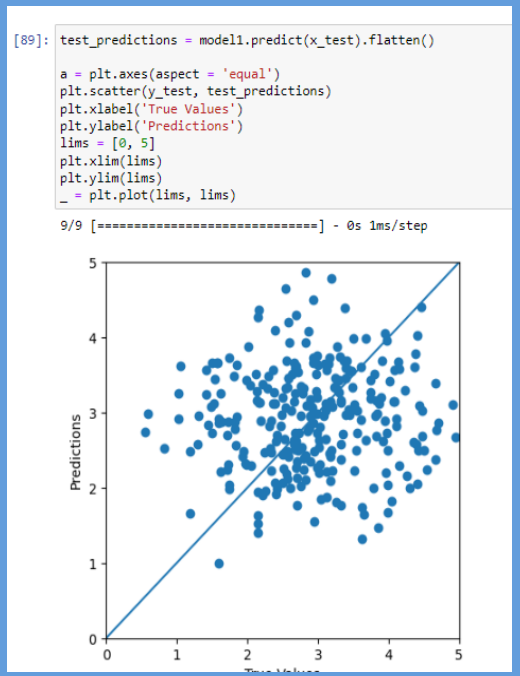
✓Вторая модель:

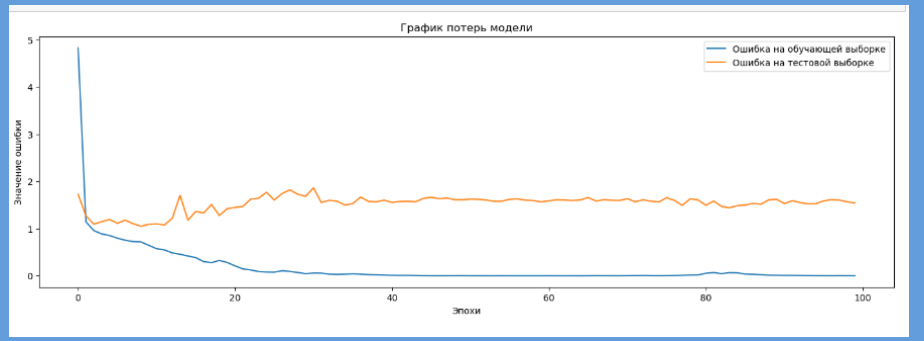
- ★ Сформируем входы и выход для модели.
- ★ Разобьём выборки на обучающую и тестовую.
- ★ Нормализуем данные.
- ★ Сконфигурируем модель, зададим слои, посмотрим на архитектуру модели.
- ★ Обучим модель.
- ★ Посмотрим на MAE, MAPE, Test score и на потери модели.
- ★ Построим график потерь на тренировочной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

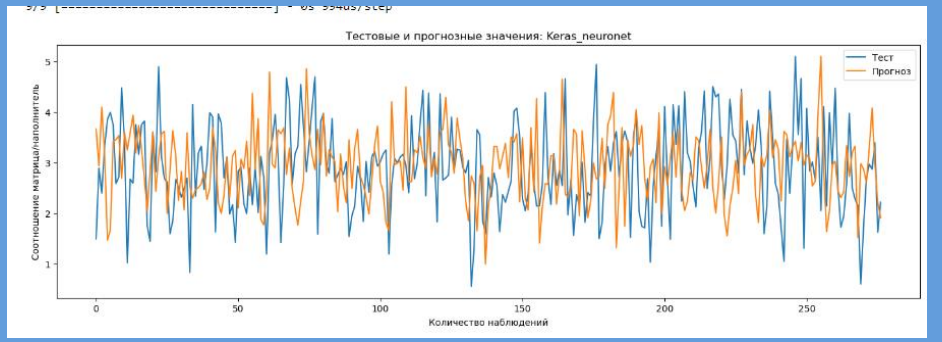












ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

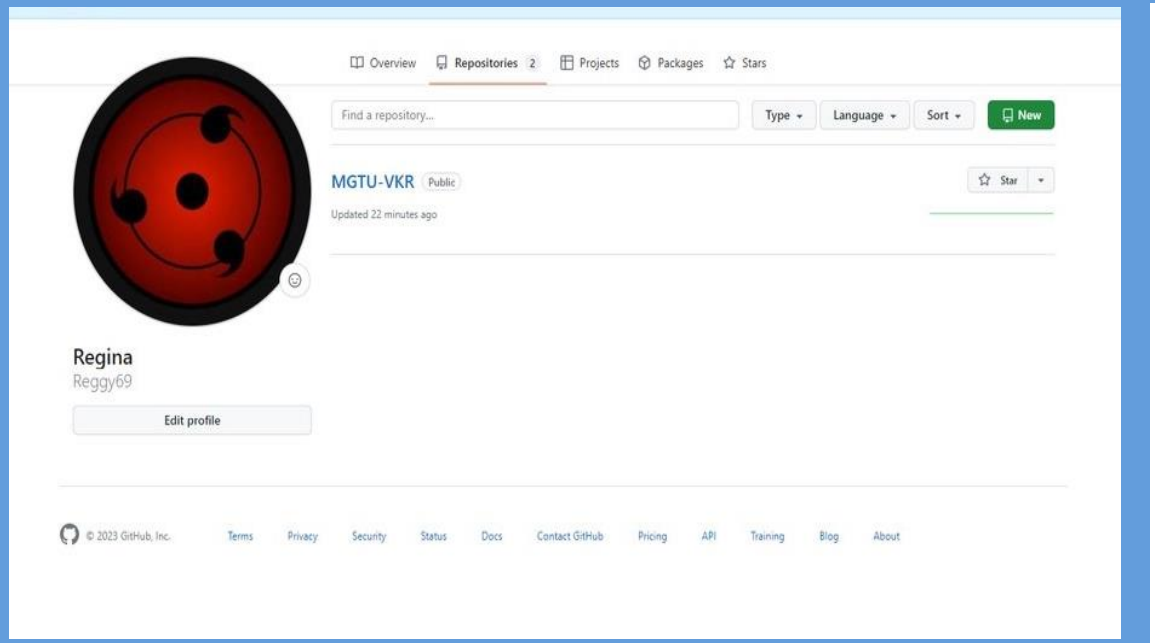
Приложение:

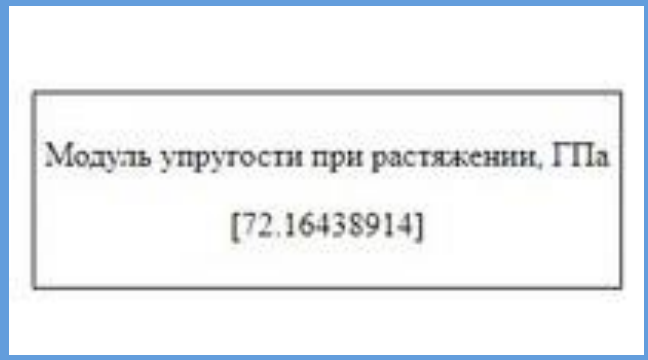
✓Пользовательское приложение

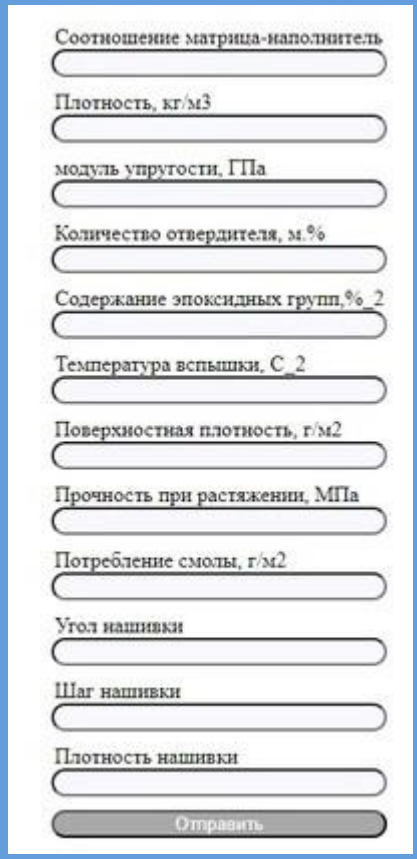
- ★ Сохраняем вторую модель нейронной сети для разработки веб-приложения для прогнозирования соотношения "матрица-наполнитель" в фреймворке Flask;
- ★ При запуске приложения, пользователь переходит на:http://127.0.0.1:5000/
- ★ В открывшемся окне пользователю необходимо ввести в соответствующие ячейки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

✓Репозиторий на github.com

https://github.com/Reggy69/MGTU-VKR







Спасибо за внимание

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.
- Опечатки, описки, пропуски скобок.
- Из-за этого модели не работали, пришлось идти разными путями: нахождение ошибок в написанном коде и пробовала другие формулы, поэтому в работе одни и те же задачи решены разными (иногда практически одинаковыми способами) вариантами.