ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу «Data Science»

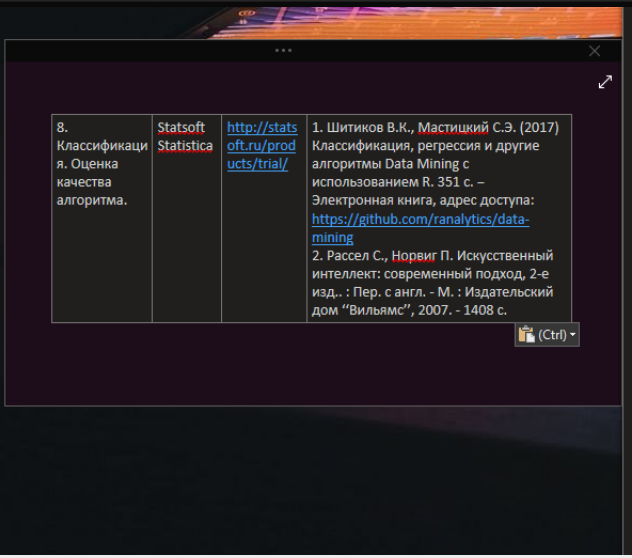
Слушатель: Едренников Николай Сергеевич

План работы:

1. Загрузить и обработать входящие датасеты
2. Провести разведочный анализ данных:
3. Визуализировать разведочный анализ сырых данных (до выбросов и нормализации)
4. Провести предобработку данных
5. Провести нормализацию и стандартизацию (продолжим предобработку данных)
6. Разработать и обучить несколько моделей прогноза прочности и модуля упругости при растяжении (с 30% тестовой выборки)
7. Создание нейронной сети для рекомендации соотношения матрица-наполнитель
8. Создание приложения
9. Создание удалённого репозитория и загрузка результатов работы на него

```
2.4.    Получим среднее, медианное значение для каждой колонки
2.5.    Вычислим коэффициенты ранговой корреляции Кендалла
2.6.    Вычислим коэффициенты корреляции Пирсона
3.  Визуализируем наш разведочный анализ сырых данных (до выбросов и нормализации)
3.1.    Построим несколько вариантов гистограмм распределения каждой переменной
3.2.    Построим несколько вариантов диаграмм ящиков с усами каждой переменной
3.3.    Построим гистограмму распределения и диаграмма "ящик с усами"
одновременно вместе с данными по каждому столбцу
3.4.    Построим несколько вариантов попарных графиков рассеяния точек (матрицы диаграмм рассеяния)
3.5.    Построим графики квантиль-квантиль
3.6.    Построим корреляционную матрицу с помощью тепловой карты
4.  Проведём предобработку данных (в данном пункте только очистка датасета от выбросов)
4.1.    Проверим выбросы по 2 методам: 3-х сигм или межквартильных расстояний
4.2.    Посчитаем распределение выбросов по каждому столбцу
(с целью предотвращения удаления особенностей признака или допущения ошиб-ки)
4.3.    Исключим выбросы методом межквартильного расстояния
4.4.    Удалим строки с выбросами
4.5.    Визуализируем датасет без выбросов, и убедимся, что выбросы еще есть.
4.6.    Для полной очистки датасета от выбросов повторим пункты (4.3 – 4.5) ещё 3 раза.
4.7.    Сохраняем идеальный, без выбросов датасет
4.8.    Изучим чистые данные по всем параметрам
4.9.    Визуализируем «чистый» датасет (без выбросов)
5.  Проведём нормализацию и стандартизацию (продолжим предобработку данных)
5.1.    Визуализируем плотность ядра
5.2.    Нормализуем данные с помощью MinMaxScaler()
5.3.    Нормализуем данные с помощью Normalizer()
5.4.    Сравним с данными до нормализации
5.5.    Проверим перевод данных из нормализованных в исходные
5.6.    Рассмотрим несколько вариантов корреляции между параметрами после нормализации
5.7.    Стандартизируем данные
5.8.    Визуализируем данные корреляции
5.9.    Посмотрим на описательную статистику после нормализации и по-сле стандартизации
6.  Разработаем и обучим нескольких моделей прогноза прочности при растя-жении (с 30% тестовой выборки)
6.1.    Определим входы и выходы для моделей
6.2.    Разобьём данные на обучающую и тестовую выборки
6.3.    Проверим правильность разбивки
6.4.    Построим модели и найдём лучшие гиперпараметры (задача по за-данию):
6.5.    Построим и визуализируем результат работы метода опорных век-торов
6.6.    Построим и визуализируем результат работы метода случайного леса
6.7.    Построим и визуализируем результат работы линейной регрессии
6.8.    Построим и визуализируем результат работы метода градиентного бустинга
6.9.    Построим и визуализируем результат работы метода K ближайших соседей
```

Normal text file | length : 12 077 | lines : 118 | Ln : 1 | Col : 15 | Pos : 28 | Windows (CR LF) | UTF-8



8. Классификация. Оценка качества алгоритма.	Statsoft Statistica	http://statsoft.ru/products/trial/	1. Шитиков В.К., Мастицкий С.Э. (2017) Классификация, регрессия и другие алгоритмы Data Mining с использованием R. 351 с. – Электронная книга, адрес доступа: https://github.com/ranalytics/data-mining 2. Рассел С., Норвиг П. Искусственный интеллект: современный подход, 2-е изд.. : Пер. с англ. - М. : Издательский дом "Вильямс", 2007. - 1408 с.

(Ctrl)

10.1 Литература и ПО

1 декабря 2022 г. 22:55

Регрессия	Deductor Studio	https://basegroup.ru/sites/default/files/deductor_academic_5.3.0.88.zip	1. Box G.E.P., Jenkins G.M., Reinsel G.C., Ljung G.M. Time Series Analysis: Forecasting and Control - 5th Edition. — Wiley, 2015. — 712 p. — ISBN: 978-1-118-67502-1. 2. Nisbet R., Elder J., Miner G. Handbook of Statistical Analysis and Data Mining Applications. - Academic Press, 2009. — 864 p. — ISBN: 0123747651 3. В. Ш. Берикашвили, С. П.Оськин Статистическая обработка данных, планирование эксперимента и случайные процессы : учебное пособие для вузов - 2-е изд., испр. и доп. - М. : Юрайт, 2021. - 163 с. 4. Документация по программной системе Deductor. Режим доступа: https://basegroup.ru/deductor/manual 5. Документация по программной системе Loginom. Режим доступа: https://help.loginom.ru/userguide/
	Loginom	https://loginom.ru/download	
	Google Colab (требуется	https://colab.research.google.com	

- 3.1 Литература и ПО
- заметки
- Установка postGreSQL
- pgAdmin
  - Видео 2.03 моделирование реляционных БД
  - Создание таблиц
  - Видео 2.09
- Питон Python
- Практика (тема 3)
- Практика (тема 2 (2.4))
- 4.1 Литература
- 4.3 Практика ленивые вычисления
- 5.1 Литература
- 5.7 Практика
- 6.2 Литература
- 6.3 Практика
- 7.1 Литература
- 7.2 Практика
- 8.1 Литература и ПО
- 8.2 Практика
- 9.2 Практика
- 10.1 Литература и ПО

Начало работы:

Подробный план работы:

- Подробный план позволил не пропускать этапы работ и снизил количество ошибок;
- Вспомнил курс материаловедения, чтобы понимать суть проблемы;
- Иногда приходилось переделывать или повторять шаги для поиска ошибок;
- Использовал 9 методов регрессий для каждой модели;
- Приложение скорее работает, но требует доработки

Вспомогательные приложения

- Справочный материал, собранный во время курса, приложения для заметок и текстовые редакторы помогали структурировать данные и быстро находить решение.

1 Загружаем данные из таблиц и удаляем колонку с индексом

```
[6]: # первый датасет (базальтопластик). Смотрим на названия столбцов
df_bp = pd.read_excel(r"G:\vkr-Big\DS\X_bp.xlsx")
df_bp.shape
```

```
[6]: (1023, 11)
```

```
[7]: #Удаляем первый неинформативный столбец
df_bp.drop(['Unnamed: 0'], axis=1, inplace=True)
#Проверяем, что первый столбец удалился
df_bp.head()
```

[7]:

	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

```
[8]: # Проверим размерность первого файла
df_bp.shape
```

```
[8]: (1023, 10)
```

2 Объединяем по индексу (тип объединения INNER) получаем датасет, имеющий данные из обоих исходных файлов

```
[14]: # Датасеты имеют разный объем строк.
# Для объединения данных, объединяем их (по inner - по условию).
df = df_bp.merge(df_nup, left_index = True, right_index = True, how = 'inner')
df.head().T
```

[14]:

	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

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:

- Сравним начальные и конечные строки датасета;
- Изучим информацию о датасете;
- Проверим типы данных в каждом столбце;
- Убедимся в отсутствии пропусков;
- Выполним поиск уникальных значений с помощью функции nunique

```
[18]: #Просмотрим информацию о датасете, проверим тип данных в столбцах
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1023 entries, 0 to 1022
Data columns (total 13 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   Соотношение матрица-наполнитель        1023 non-null   float64
 1   Плотность, кг/м3                       1023 non-null   float64
 2   модуль упругости, ГПа                  1023 non-null   float64
 3   Количество отвердителя, м.%            1023 non-null   float64
 4   Содержание эпоксидных групп,%_2        1023 non-null   float64
 5   Температура вспышки, С_2               1023 non-null   float64
 6   Поверхностная плотность, г/м2          1023 non-null   float64
 7   Модуль упругости при растяжении, ГПа   1023 non-null   float64
 8   Прочность при растяжении, МПа          1023 non-null   float64
 9   Потребление смолы, г/м2                1023 non-null   float64
 10  Угол нашивки, град                     1023 non-null   float64
 11  Шаг нашивки                            1023 non-null   float64
 12  Плотность нашивки                      1023 non-null   float64
dtypes: float64(13)
memory usage: 111.9 KB
```

```
[19]: #Поиск уникальных значений (nunique)
df.nunique()
```

```
[19]: Соотношение матрица-наполнитель        1014
Плотность, кг/м3                       1013
модуль упругости, ГПа                  1020
Количество отвердителя, м.%            1005
Содержание эпоксидных групп,%_2        1004
Температура вспышки, С_2               1003
Поверхностная плотность, г/м2          1004
Модуль упругости при растяжении, ГПа   1004
Прочность при растяжении, МПа          1004
Потребление смолы, г/м2                1003
Угол нашивки, град                        2
Шаг нашивки                             989
Плотность нашивки                       988
dtype: int64
```

```
[49]: # Изучим столбец "Угол нашивки"

[50]: df['Угол нашивки, град'].nunique()
      #Количество уникальных значений в строке "Угол нашивки" равно двум, приведем данные к значениям 0 и 1

[50]: 2

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

[51]: 520

[52]: # 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)

[53]: #Переименование столбца
      df = df.rename(columns={'Угол нашивки, град' : 'Угол нашивки'})
      df

[89]: #Посчитаем количество элементов, где угол нашивки равен 0 градусов, проверим, что количество не изменилось после наших манипуляций
      df['Угол нашивки'][df['Угол нашивки'] == 0.0].count()
      #После преобразования колонки Угол нашивки к значениям 0 и 1, кол-во элементов, где угол нашивки равен 0 не изменилось (520 до и после пр

[89]: 520

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

[91]: # Сохраним итоговый датасет в отдельную папку с данными, чтобы долго не искать
      df.to_excel('itog_V3.xlsx')
      # Почему-то не получается указать путь к месту сохренения файла. Еще вернусь к этому. Пока сохраняет в дерикторияю пользователя...
```

[94]:

	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

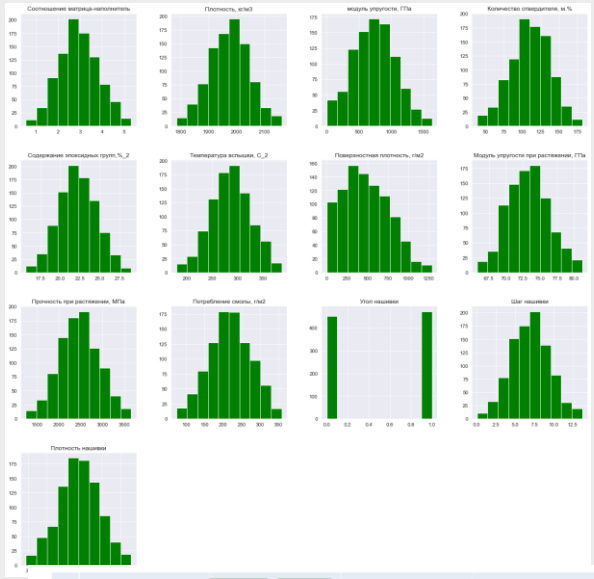
«Угол нашивки» и описательная статистика:

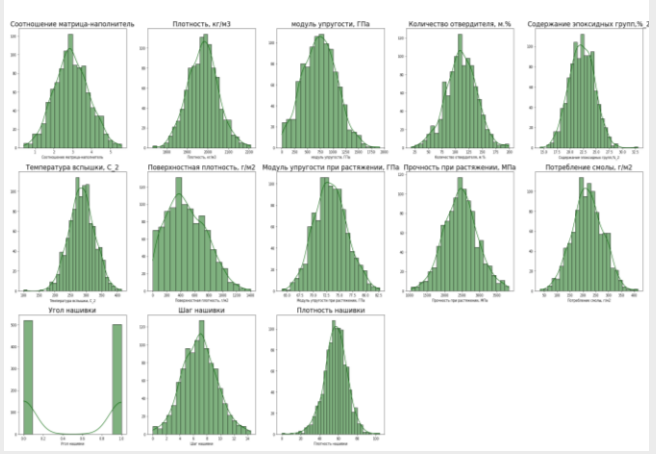
Работа со столбцом "Угол нашивки":

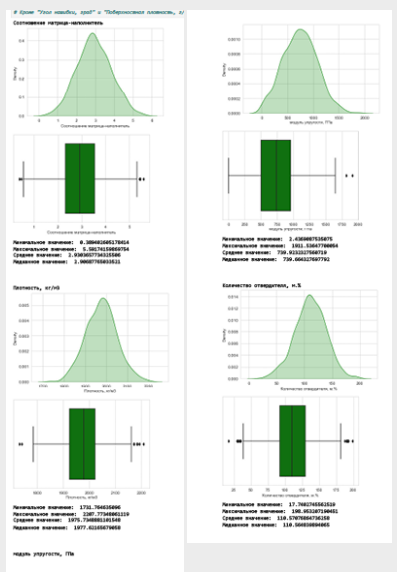
- Проверяем количество элементов со значением 0 градусов;
- Приводим к значениям 0 и 1;
- Убедимся в неизменном количестве элементов

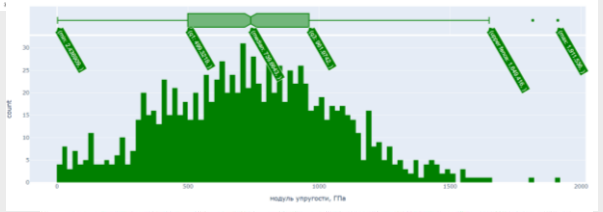
Описательная статистика:

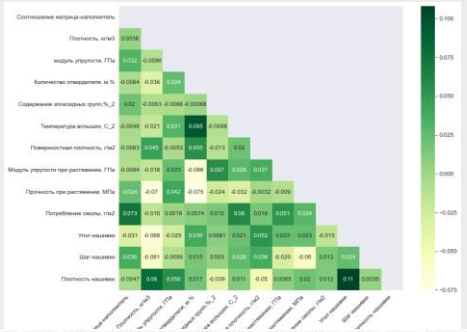
- Изучим описательную статистику данных
- Посмотрим на основные параметры анализа данных;
- Проверим датасет на пропущенные и дублирующие данные;
- Вычислим коэффициенты ранговой корреляции Кендалла и Пирсона

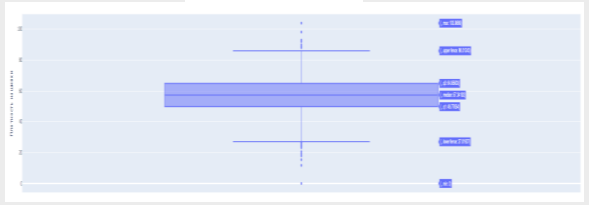


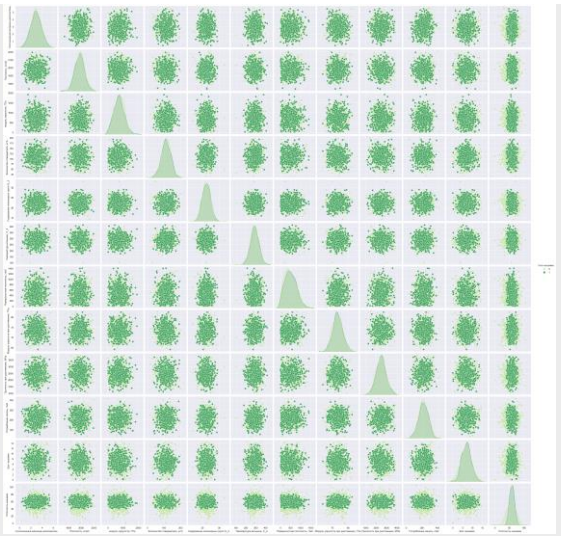


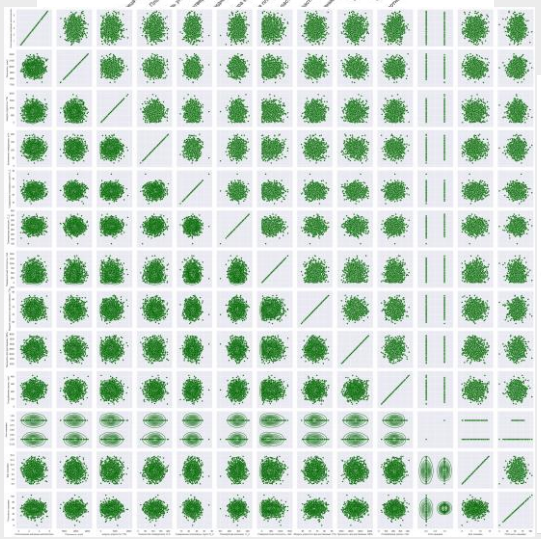


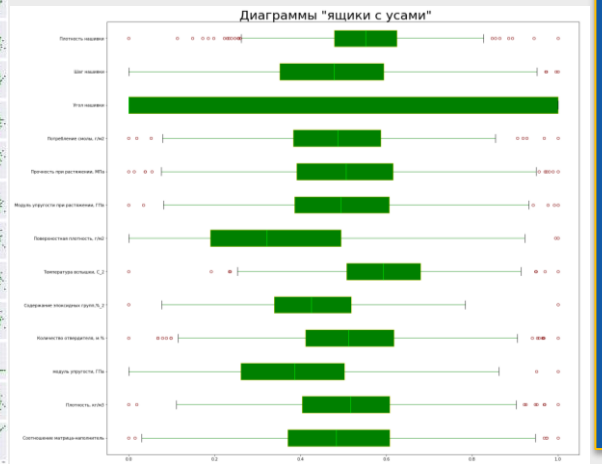












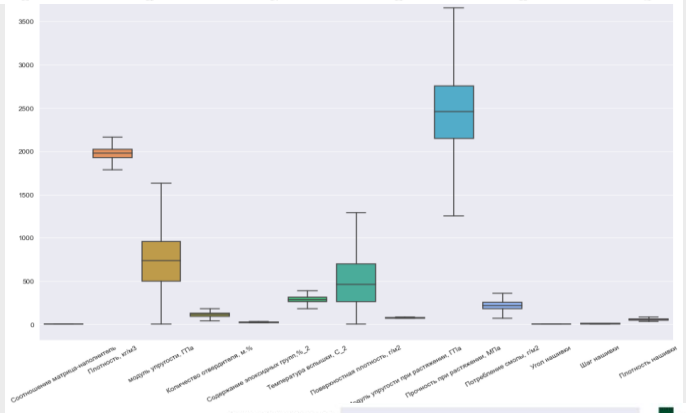
Визуализация «сырых» данных:

Графики без нормализации и исключения шумов :

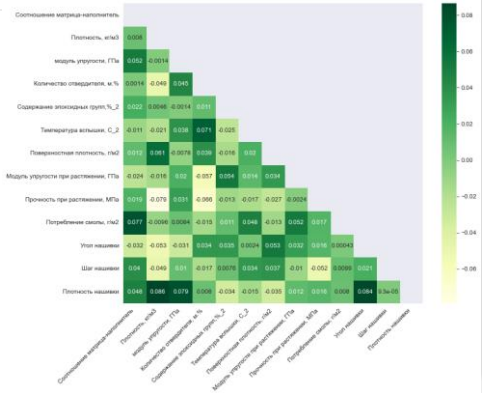
- Построим гистограммы распределения каждой из переменных (несколько вариантов);
- диаграммы "ящиков с усами" (несколько вариантов);
- попарные графики рассеяния точек (несколько вариантов);
- графики квантиль-квантиль;
- тепловые карты (несколько вариантов)

```python
#Для удаления выбросов существует 2 основных метода - метод 3-х си
metod_3s = 0
metod_iq = 0
count_iq = [] # Список, куда записывается количество выбросов по к
count_3s = [] # Список, куда записывается количество выбросов по к
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)
```

```
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
```



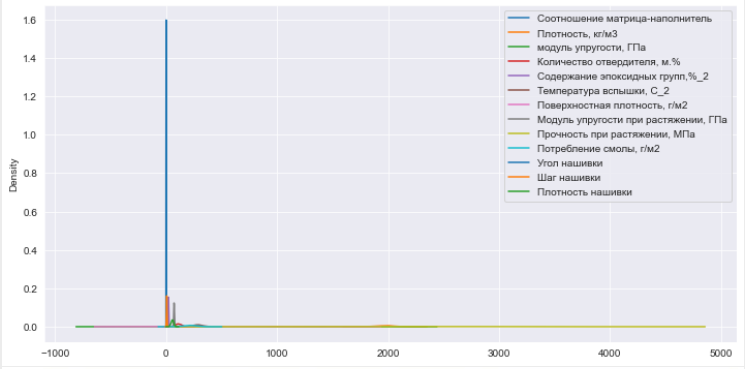
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 922 entries, 1 to 1022
Data columns (total 13 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Соотношение матрица-наполнитель    922 non-null    float64
 1   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
```

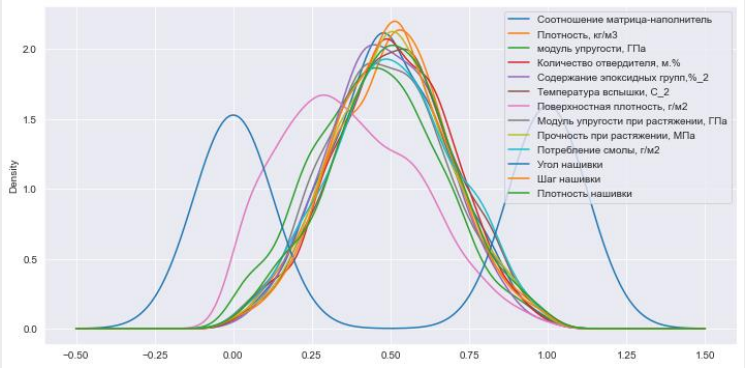


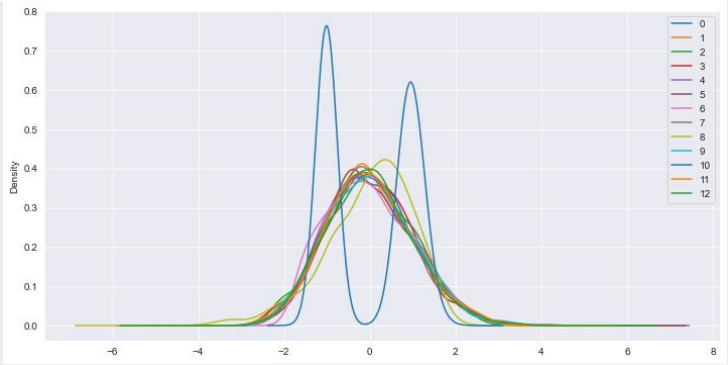
Предобработка данных:

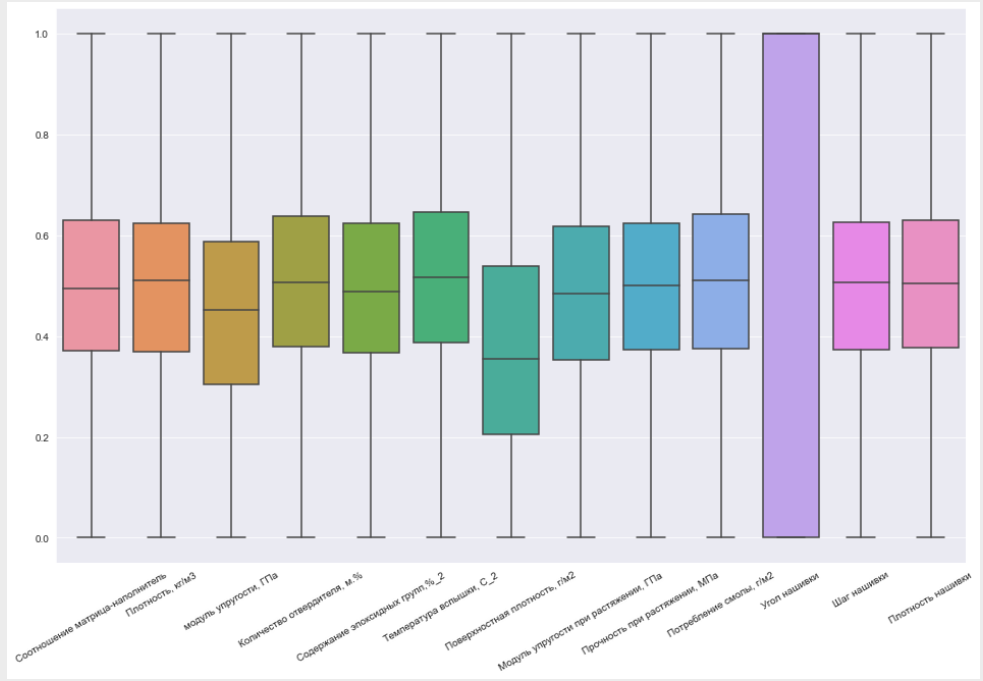
Исключение выбросов:

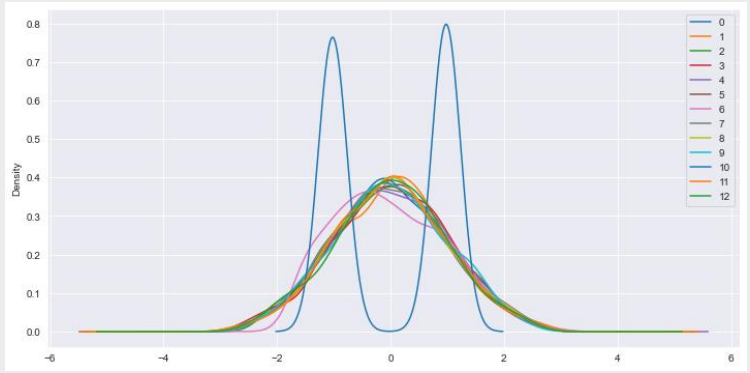
- Посчитаем количество значений методом 3 сигм и методом межквартильных расстояний;
- Исключим выбросы методом межквартильного расстояния ;
- Проверим результат;
- Построим графики;
- Убедимся, что выбросы ещё остались;
- Повторим удаление выбросов ещё 4 раза до полного удаления;
- Проверим чистоту датасета от выбросов
- Построим все возможные графики «чистого» датасета











df_standart_1

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	-1.196467	0.787037	0.007992	-2.286425	0.647585	-0.039740	-0.971971	-1.092335	1.187925	0.034181	-1.021932	-1.166792	0.219240
1	-1.196467	0.787037	0.007992	0.668092	-0.397291	0.350738	-0.971971	-1.092335	1.187925	0.034181	-1.021932	-0.768833	-0.950227
2	-0.175012	0.787037	0.051553	0.027074	0.028123	-0.039740	-0.971971	-1.092335	1.187925	0.034181	-1.021932	-0.768833	-0.050637
3	-0.178825	0.364514	0.036283	0.027074	0.028123	-0.039740	-0.971971	-1.092335	1.187925	0.034181	-1.021932	-0.768833	0.219240
4	-0.400390	-0.903054	0.216474	0.027074	0.028123	-0.039740	-0.971971	-1.092335	1.187925	0.034181	-1.021932	-0.768833	1.118831

Предобработка данных:

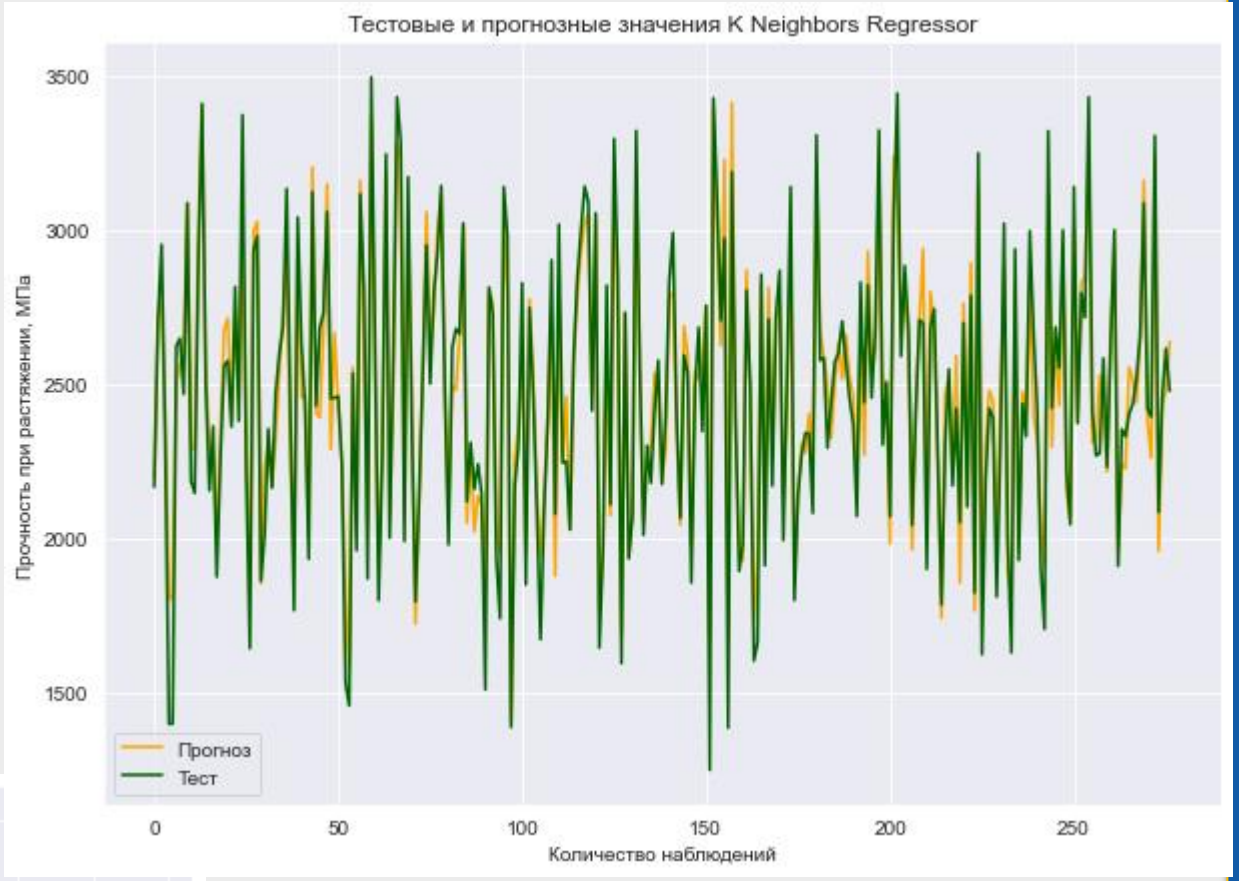
✓ Нормализация данных:

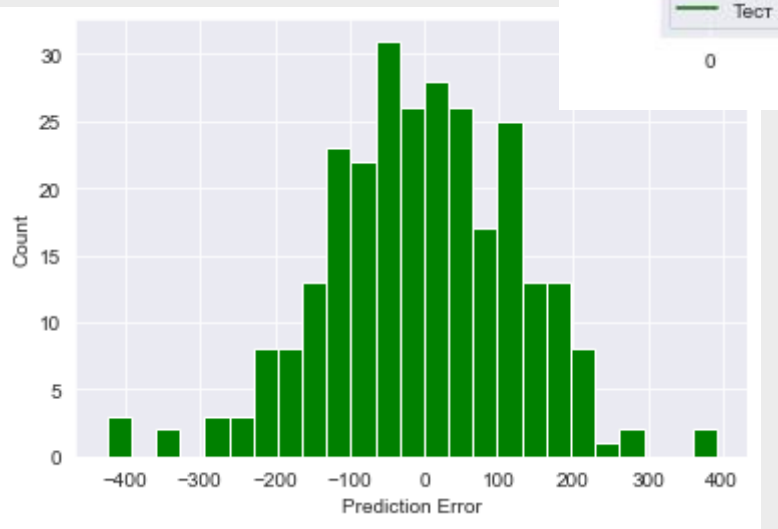
- Нормализуем данные MinMaxScaler();
- Построим график плотности ядра;
- Проверим результат MinMaxScaler();
- Построим графики MinMaxScaler();
- Нормализуем данные с помощью Normalizer();
- Проверим результат Normalizer();
- Построим графики Normalizer();

✓ Стандартизация данных:

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

```
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
```






```python
# Метод К ближайших соседей - K Neighbors Regressor - 5
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
```

Разработка и обучение моделей для прогноза прочности при растяжении:

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;
- Построим графики для тестовых и прогнозных значений;
- Построим гистограмму распределения ошибки

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

```
Наилучшие параметры:
{'preprocessing': StandardScaler(), 'regressor': SGDRegressor()}

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

```
# Проведем поиск  по сетке гиперпараметров с перекрестной проверкой, количество блоков равно 10 (cv = 10), для
#Деревья решений - Decision Tree Regressor - 6
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
{'algorithm': 'brute', 'n_neighbors': 7, 'weights': 'distance'}
```

```
#Выводим гиперпараметры для оптимальной модели
print(gs4.best_estimator_)
gs1 = gs4.best_estimator_
print(f'R2-score DTR для прочности при растяжении, МПа: {gs4.score(x_test_1, y_test_1).round(3)}')
```

```
DecisionTreeRegressor(criterion='poisson', max_depth=5, max_features='auto',
                      min_samples_leaf=100, min_samples_split=250)
R2-score DTR для прочности при растяжении, МПа: 0.779
```

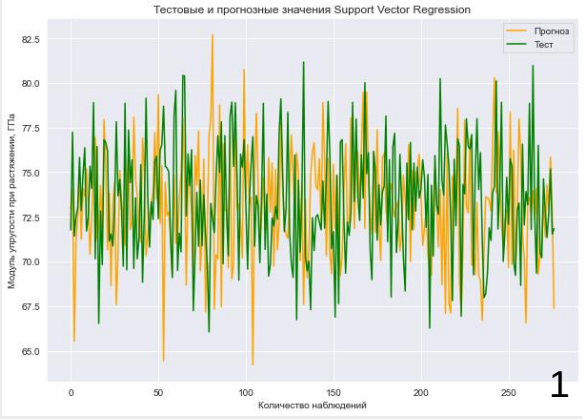
```
#подставим оптимальные гиперпараметры в нашу модель метода деревья решений
dtr_grid = DecisionTreeRegressor(criterion = 'poisson', max_depth = 7, max_features = 'auto',
                                 min_samples_leaf = 100, min_samples_split = 250)
#Обучаем модель
dtr_grid.fit(x_train_1, y_train_1)

predictions_dtr_grid = dtr_grid.predict(x_test_1)
#Оценивае точность на тестовом наборе
mae_dtr_grid = mean_absolute_error(predictions_dtr_grid, y_test_1)
mae_dtr_grid
```

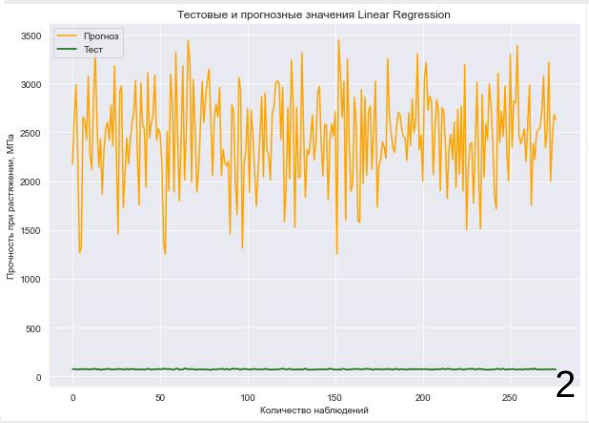
```
168.6249974156563
```

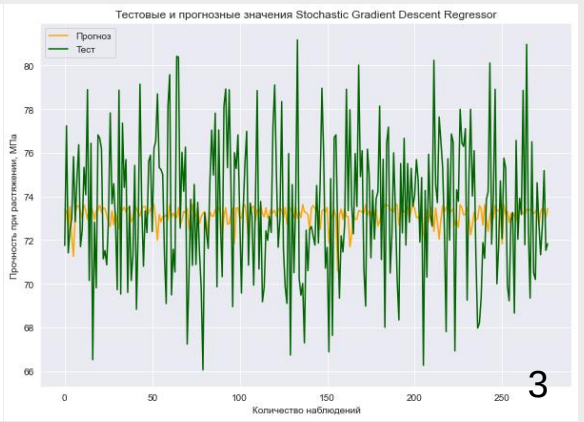
Поиск гиперпараметров:

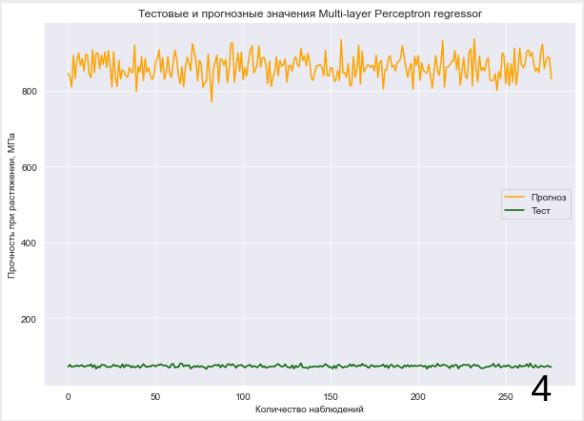
✓ **Для метода «Деревья решений»:**

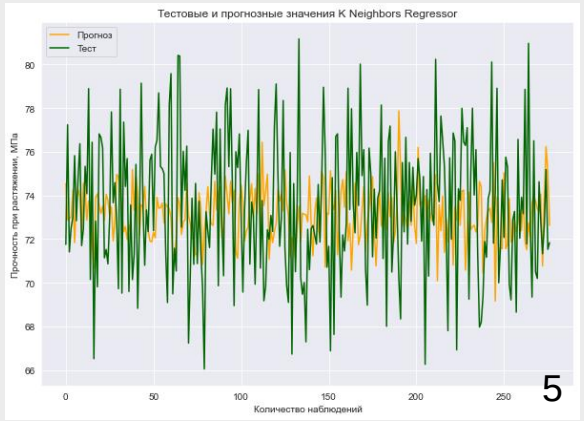
- Поиск гиперпараметров методом GridSearchCV с перекрёстной проверкой с количеством блоков 10;
- Выводим гиперпараметры для оптимальной модели;
- Подставляем оптимальные гиперпараметры в модель случайного леса;
- Обучаем модель;
- Оцениваем точность на тестовом наборе;
- Выводим наилучшее значение правильности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

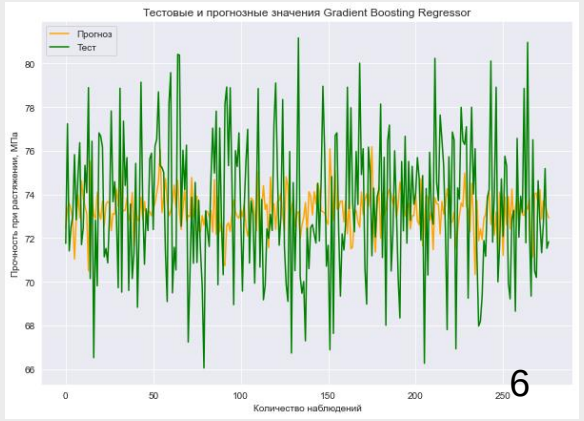


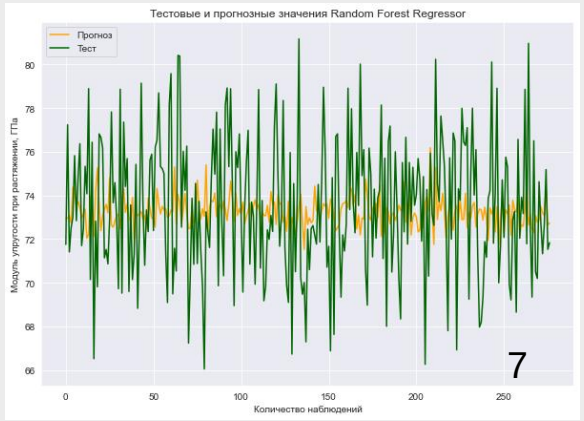


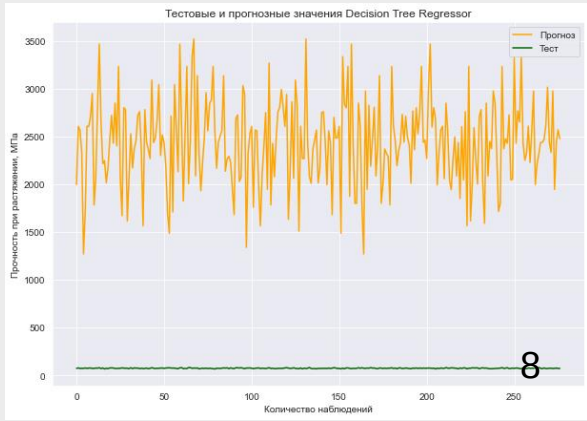


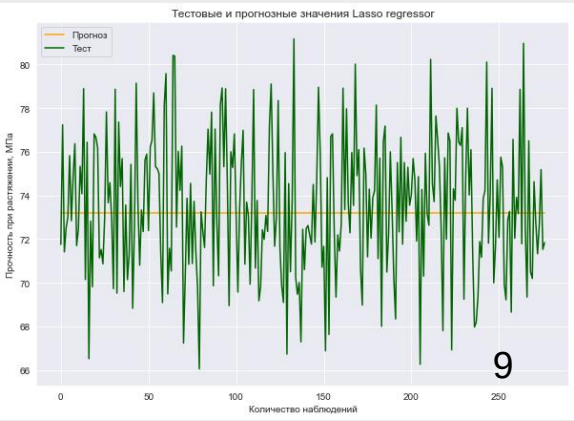












Разработка и обучение моделей для прогноза модуль упругости при растяжении:

Графики тестовых и прогнозных значений для разных методов:

1. Метод опорных векторов
2. Линейная регрессия
3. Стохастический градиентный спуск
4. Многослойный перцептрон
5. К-ближайших соседей
6. Градиентный бустинг
7. «Случайный лес»
8. Дерево принятия решений
9. Лассо

```
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_1, 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(grid.score(x_test_2, y_test_2)))
```

```
Наилучшие параметры:
{'preprocessing': MinMaxScaler(), 'regressor': SVR(C=100, gamma=1), 'regressor__C': 100, 'regressor__gamma': 1}

Наилучшее значение правильности перекрестной проверки: 0.68
Правильность на тестовом наборе: -79805487.66
```

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

```
Наилучшая модель:
Pipeline(steps=[('preprocessing', StandardScaler()),
                ('regressor', SGDRegressor())])
```

	Perpeccop	MAE
0	Support Vector	78.477914
1	RandomForest	76.589025
2	Linear Regression	61.986894
3	GradientBoosting	64.728717
4	KNeighbors	102.030259
5	DecisionTree	107.158013
6	SGD	181.624450
7	MLP	1808.547264
8	Lasso	69.474334
9	RandomForest_GridSearchCV	67.603567
10	KNeighbors_GridSearchCV	99.281694
11	DecisionTree_GridSearchCV	168.624997
12	RandomForest1_GridSearchCV	2.627032

```
# Проведем поиск  по сетке гиперпараметров с перекрестной проверкой, количество блоков равно 10 (cv = 10), для
# модели случайного леса - Random Forest Regressor - 2

parametrs = { 'n_estimators': [200, 300],
              'max_depth': [9, 15],
              'max_features': ['auto'],
              'criterion': ['mse'] }
grid21 = GridSearchCV(estimator = rfr2, param_grid  = parametrs, cv=10)
grid21.fit(x_train_2, y_train_2)
```

```
GridSearchCV(cv=10,
             estimator=RandomForestRegressor(max_depth=7, n_estimators=15,
                                             random_state=33),
             param_grid={'criterion': ['mse'], 'max_depth': [9, 15],
                         'max_features': ['auto'], 'n_estimators': [200, 300]})
```

```
#Выводим гиперпараметры для оптимальной модели
print(grid21.best_estimator_)
knr_u = grid21.best_estimator_
print(f'R2-score RFR для модуля упругости при растяжении: {knr_u.score(x_test_2, y_test_2).round(3)}')
```

```
RandomForestRegressor(criterion='mse', max_depth=9, n_estimators=300,
                      random_state=33)
R2-score RFR для модуля упругости при растяжении: -0.035
```

Поиск гиперпараметров: для прогноза модуль упругости при растяжении:

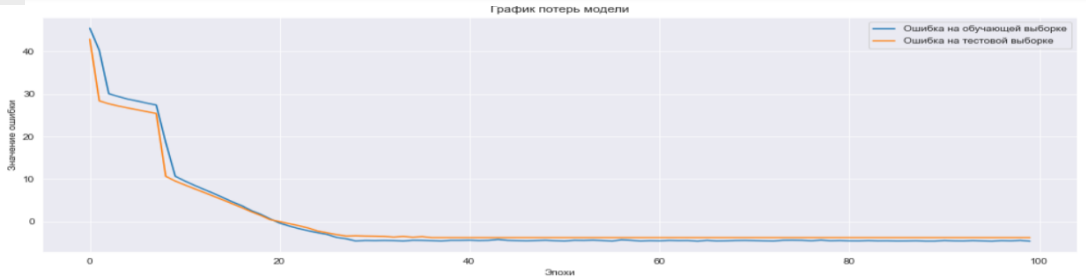
Для метода «Случайный лес»:

- 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

```python
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
```

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

print(model.summary())
```

```
Model: "sequential_405"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1077 (Dense)          (None, 128)               1664

 dense_1078 (Dense)          (None, 64)                8256

 dense_1079 (Dense)          (None, 16)                1040

 dense_1080 (Dense)          (None, 3)                 51

 dropout_405 (Dropout)       (None, 3)                 0

 dense_1081 (Dense)          (None, 3)                 12

=================================================================
Total params: 11,023
Trainable params: 11,023
Non-trainable params: 0
_________________________________________________________________
None
```




```
Best: 0.001538 using {'batch_size': 4, 'epochs': 10}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 10}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 50}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 100}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 200}
0.001538 (0.004615) with: {'batch_size': 4, 'epochs': 300}
```

```
Best: 0.004639 using {'lyrs': [128, 64, 16, 3]}
0.001538 (0.004615) with: {'lyrs': [8]}
0.001538 (0.004615) with: {'lyrs': [16, 4]}
0.001538 (0.004615) with: {'lyrs': [32, 8, 3]}
0.001538 (0.004615) with: {'lyrs': [12, 6, 3]}
0.001538 (0.004615) with: {'lyrs': [64, 64, 3]}
0.004639 (0.009877) with: {'lyrs': [128, 64, 16, 3]}
```

```
Best: 0.001538 using {'act': 'softmax'}
0.001538 (0.004615) with: {'act': 'softmax'}
0.001538 (0.004615) with: {'act': 'softplus'}
0.001538 (0.004615) with: {'act': 'softsign'}
0.001538 (0.004615) with: {'act': 'relu'}
0.001538 (0.004615) with: {'act': 'tanh'}
0.001538 (0.004615) with: {'act': 'sigmoid'}
0.001538 (0.004615) with: {'act': 'hard_sigmoid'}
0.001538 (0.004615) with: {'act': 'linear'}
```

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

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.

```
# Сконфигурируем модель, зададим слои

model = tf.keras.Sequential([x_train_n, layers.Dense(128, activation='relu'),
                                        layers.Dense(128, activation='relu'), Dropout(0.8),
                                        layers.Dense(128, activation='relu'),
                                        layers.Dense(64, activation='relu'),
                                        layers.Dense(32, activation='relu'),
                                        layers.Dense(16, activation='relu'),
                                        layers.Dense(1)
                             ])

model.compile(optimizer = tf.keras.optimizers.Adam(0.001), loss = 'mean_squared_error', metrics = [tf.keras.metrics.RootMeanSquaredError()])
# Посмотрим на архитектуру модели

model.summary()
```

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

 dense (Dense)               (None, 128)               1664

 dense_1 (Dense)             (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 32)                2080

 dense_5 (Dense)             (None, 16)                528

 dense_6 (Dense)             (None, 1)                 17

=================================================================
Total params: 45,594
Trainable params: 45,569
Non-trainable params: 25
_________________________________________________________________
```

```
model.evaluate(x_test, y_test)
```

```
9/9 [==============================] - 0s 3ms/step - loss: 1.5056 - root_mean_squared_error: 1.2270
[1.5056190490722656, 1.227036714553833]
```


```
# Обучим модель

model_hist = model.fit(
    x_train,
    y_train,
    epochs = 100,
    verbose = 1,
    validation_split = 0.3)
```

```
Model Results:
Model_MAE:  1
Model_MAPE: 0.37
Test score: 1.25
```

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.

Приложение:
Пользовательское приложение

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

edu.bmstu.ru

+7 495 182-83-85

edu@bmstu.ru

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

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана