

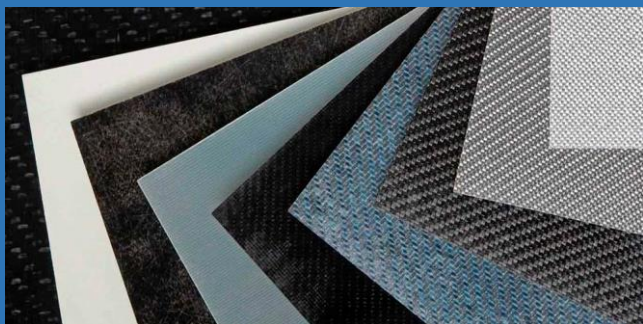


**ОБРАЗОВАТЕЛЬНЫЙ
ЦЕНТР** МГТУ им. Н. Э. Баумана

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу
«Data Science»

Слушатель: Митрошкина Кристина Вячеславовна



Тема выпускной квалификационной работы: «Прогнозирование конечных свойств новых материалов (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.

Объектом исследования является процесс прогнозирования конечных свойств новых материалов.

Предметом исследования – автоматизация процесса прогнозирования конечных свойств новых материалов.

Цели данной выпускной квалификационной работы:

1. Обучить алгоритм машинного обучения, который будет определять значения:
 - Модуль упругости при растяжении, ГПа
 - Прочность при растяжении, МПа
2. Написать нейронную сеть, которая будет рекомендовать:
 - Соотношение матрица-наполнитель
3. Написать приложение, которое будет выдавать прогнозное значение параметра «Соотношение матрица-наполнитель».



Начало работы:

✓ Подробный план работы:

- Составим подробный план;
- Изучим теоретические основы, методы решения и практические составляющие поставленной задачи;
- Некоторые пункты плана потребовали повторения , для достижения результата;
- В ВКР использовано 9 разных методов регрессий для каждой из моделей;
- Приложение успешно работает

✓ Графики:

- Построено много графиков для визуального анализа данных;
- Несколько подобных графиков для одних и тех же переменных;
- Графики преведены в одном стиле



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

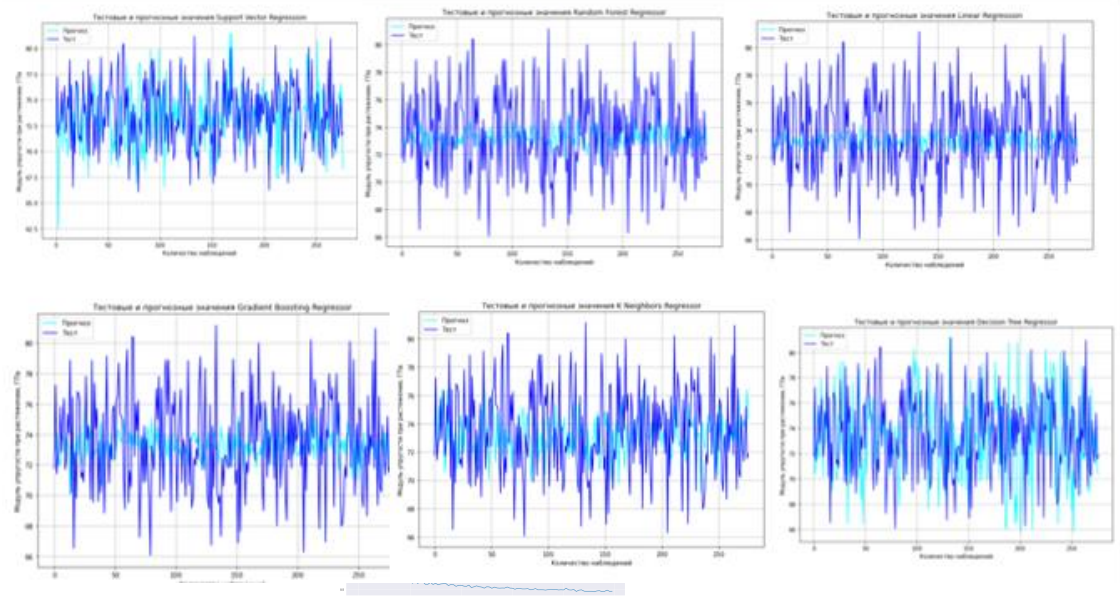
Приложение 1

Подробный план работы:

1. Загружаем и обрабатываем входящие **данные**
- 1.1. Удаляем неинформативные столбцы
- 1.2. Объединяем **данные** по методу INNER
2. Проводим разведочный анализ данных:
 - 2.1. Данные в столбце "Угол наклона" приведём к 0 и 1
 - 2.2. Изучим описательную статистику каждой переменной - среднее, медиана, стандартное отклонение, минимум, максимум, квартили
 - 2.3. Проверим **данные** на пропуски и дубликаты данных
 - 2.4. Получим среднее, медианное значение для каждой колонки (по заданию необходимо получить их отдельно, поэтому продублируем их только отдельно)
 - 2.5. Вычислим коэффициенты ранговой корреляции **Сpearman**
 - 2.6. Вычислим коэффициенты корреляции Пирсона
3. Визуализируем наш разведочный анализ сериями 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 "ящики 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 **данных** от выбросов)
 - 4.1. Проверим выбросы по 2 методам: 3-х сигм или **методом расстояний**
 - 4.2. Посчитаем распределение выбросов по каждому столбцу (с целью предотвращения удаления особенностей признака или допущения ошибки)
 - 4.3. Исключим выбросы методом **устранения расстояния**
 - 4.4. Удалили строки с выбросами
 - 4.5. Визуализируем **данные** без выбросов, и убедимся, что выбросы **есть**
 - 4.6. Для полной очистки **данных** от выбросов повторим пункты (4.3 – 4.5) ещё 3 раза.
 - 4.7. Сохраним идеальный, без выбросов **данные**
 - 4.8. Изучим чистые данные по всем параметрам
 - 4.9. Визуализируем чистый **данные** (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 **StandardScaler()**
 - 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 после стандартизации тестовой выборки
6. Разработаем и обучим несколько моделей прогноза прочности при растяжении (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 разбиения

- 6.4. Построим модели и найдём лучшие **параметры** (задача по заданию):
- 6.5. Построим и визуализируем результат работы метода опорных векторов
- 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
- 6.10. Построим и визуализируем результат работы метода деревьев решений
- 6.11. Построим и визуализируем результат работы стохастического градиентного спуска
- 6.12. Построим и визуализируем результат работы многослойного перцептрона
- 6.13. Построим и визуализируем результат работы лассо регрессии
- 6.14. Сравним наши модели по метрике MAE
- 6.15. Найдём лучшие **параметры** для случайного леса
- 6.16. Подставим значения в нашу модель случайного леса
- 6.17. Найдём лучшие **параметры** для K ближайших соседей
- 6.18. Подставим значения в нашу модель K ближайших соседей
- 6.19. Найдём лучшие **параметры** метода деревьев решений
- 6.20. Подставим значения в нашу модель метода деревьев решений
- 6.21. Проверим все модели и процессинг и выведем лучшую модель и процессинг
7. Разработаем и обучим несколько моделей прогноза модуля упругости при растяжении (с 30% тестовой выборки)
 - 7.1. Определим входы и выходы для моделей
 - 7.2. Разобьём данные на обучающую и тестовую выборки
 - 7.3. Проверим правильность разбиения
 - 7.4. Построим модели и найдём лучшие **параметры** (задача по заданию):
 - 7.5. Построим и визуализируем результат работы метода опорных векторов
 - 7.6. Построим и визуализируем результат работы метода случайного леса
 - 7.7. Построим и визуализируем результат работы линейной регрессии
 - 7.8. Построим и визуализируем результат работы метода градиентного **бустинга**
 - 7.9. Построим и визуализируем результат работы метода K ближайших соседей
 - 7.10. Построим и визуализируем результат работы метода деревьев решений
 - 7.11. Построим и визуализируем результат работы стохастического градиентного спуска
 - 7.12. Построим и визуализируем результат работы многослойного перцептрона
 - 7.13. Сравним наши модели по метрике MAE
 - 7.14. Найдём лучшие **параметры** для случайного леса
 - 7.15. Подставим значения в нашу модель случайного леса
 - 7.16. Подставим значения в нашу модель K ближайших соседей
 - 7.17. Найдём лучшие **параметры** для K ближайших соседей
 - 7.18. Подставим значения в нашу модель K ближайших соседей
 - 7.19. Найдём лучшие **параметры** метода деревьев решений
 - 7.20. Подставим значения в нашу модель метода деревьев решений
 - 7.21. Проверим все модели и процессинг и выведем лучшую модель и процессинг
8. Нейронная сеть для рекомендации соотношения матрица-наполнителя
 - 8.1. Оформим входы и выходы для модели
 - 8.2. Нормализуем данные
 - 8.3. Построим модель, определим параметры
 - 8.4. Найдём оптимальные параметры для модели
 - 8.5. Посмотрим на результаты
 - 8.6. Повторим шаги 8.4 – 8.5 до построения окончательной модели
 - 8.7. Обучим нейросеть 80/20
 - 8.8. Оценим модель

- 8.9. Посмотрим на потери модели
- 8.10. Посмотрим на график результата работы модели
- 8.11. Посмотрим на график потерь на тренировочной и тестовой выборках
- 8.12. Сконфигурируем другую модель, зададим слои
- 8.13. Посмотрим на архитектуру другой модели
- 8.14. Обучим другую модель
- 8.15. Посмотрим на потери другой модели
- 8.16. Посмотрим на график потерь на тренировочной и тестовой выборках
- 8.17. Зададим функцию для визуализации факт/прогноза для результатов моделей
- 8.18. Посмотрим на график результата работы модели
- 8.19. Оценим модель MSE
- 8.20. Сохраним вторую модель для разработки веб-приложения для прогнозирования соотношения "матрица-наполнителя" в **фреймворке Flask**
9. Создаём приложение
 - 9.1. Импортируем необходимые **библиотеки**
 - 9.2. Загрузим модель и определим параметры функции
 - 9.3. Получим данные из наших форм и положим их в список
 - 9.4. Укажем шаблон и прототип сайта для вывода
 - 9.5. Запустим приложение
 - 9.6. Откроем <http://127.0.0.1:5000/>
10. Создание удалённого репозитория и загрузка результатов работы на него.
 - 10.1. <https://github.com/MikhailinKV/KOMPPOST>
 - 10.2. Создадим README
 - 10.3. Выгрузим все необходимые файлы в репозиторий



Объединение файлов и разведочный анализ:

✓ Объединение по индексу:

- Импортируем необходимые библиотеки;
- Загружаем файлы;
- Посмотрим размерность;
- Объединим оба файла по индексу по типу объединения INNER

✓ Разведочный анализ данных:

- Посмотрим на начальные и конечные строки нашего датасета;
- Изучим информацию о датасете;
- Проверим типы данных в каждом столбце;
- Проверим пропуски;
- Поищем уникальные значения с помощью функции nunique

Объединяем по индексу, тип объединения INNER, смотрим итоговый датасет

```
In [9]: # Понимаем, что эти два датасета имеют разный объем строк.  
# Но наша задача собрать исходные данные файлы в один, единый набор данных.  
# По условию задачи объединяем их по типу INNER.  
df = df_bp.merge(df_nup, left_index = True, right_index = True, how = 'inner')  
df.head().T
```

Out[9]:	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, C_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

```
#Удаляем первый неинформативный столбец  
df_nup.drop(['Unnamed: 0'], axis=1, inplace=True)
```

```
#Посмотрим на первые 5 строк второго датасета и убедимся, что и здесь не нужный первый столбец успешно удалился  
df_nup.head()
```

	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	0.0	4.0	57.0
1	0.0	4.0	60.0
2	0.0	4.0	70.0
3	0.0	5.0	47.0
4	0.0	5.0	57.0

```
# Проверим размерность второго файла  
df_nup.shape
```

(1040, 3)

```
<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, C_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

Соотношение матрица-наполнитель	1014
Плотность, кг/м3	1013
модуль упругости, ГПа	1020
Количество отвердителя, м.%	1005
Содержание эпоксидных групп,%_2	1004
Температура вспышки, C_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	



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

«Угол нашивки» и описательная статистика:

✓ Работа со столбцом "Угол нашивки":

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

✓ Описательная статистика:

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



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

```
# Поработаем со столбцом "Угол нашивки"
```

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

```
#Посчитаем количество элементов, где угол нашивки равен 0  
df['Угол нашивки'][df['Угол нашивки'] == 0.0].count()  
#После преобразования колонки Угол нашивки к значениям
```

520

```
# Переведем столбец с нумерацией в integer  
df.index = df.index.astype('int')
```

```
# Сохраним итоговый датасет в отдельную папку с данными  
df.to_excel("Itog\itog.xlsx")
```

```
# 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, C_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
```

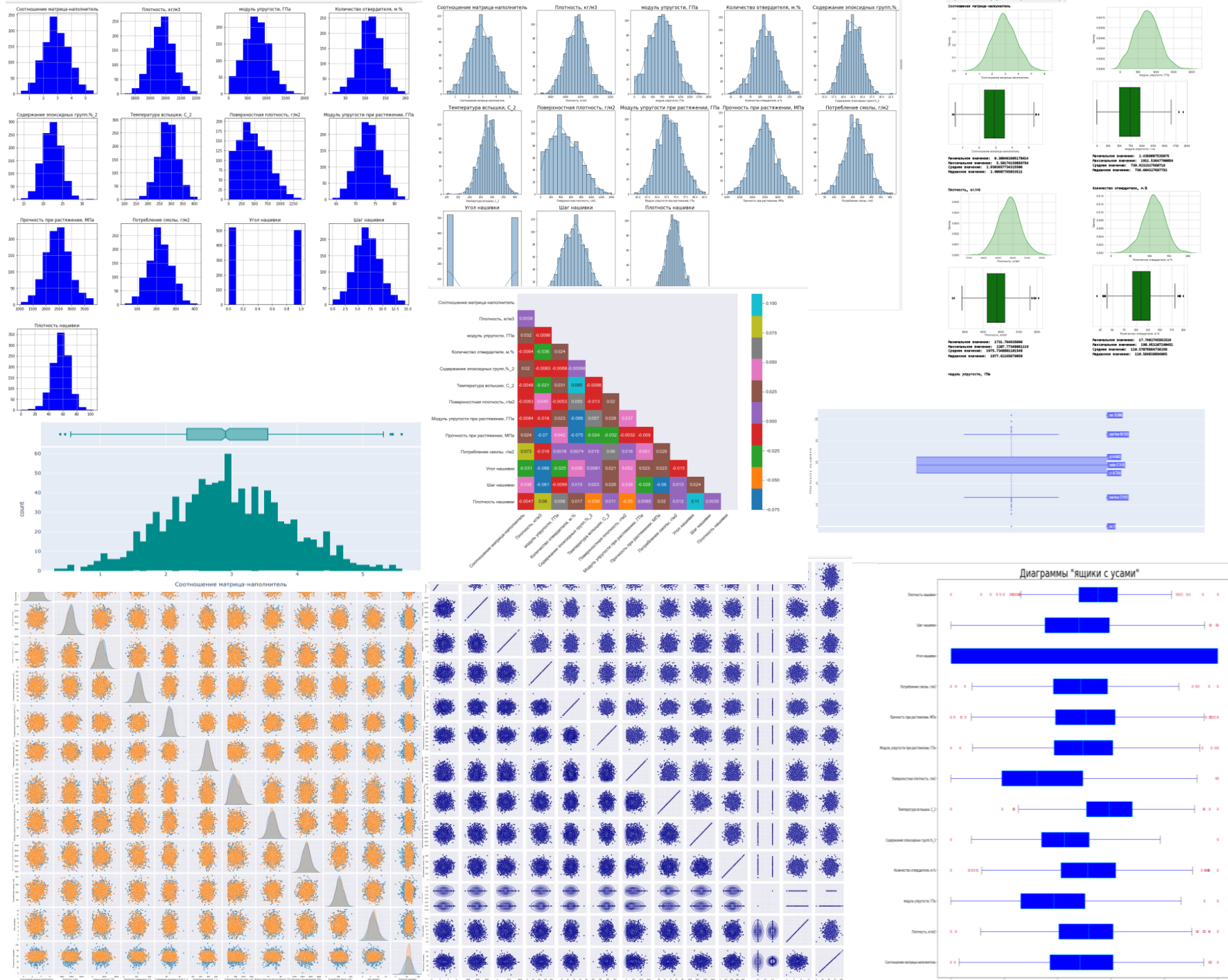
Визуализация «сырых» данных:

✓ 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** МГУ им. Н. Э. Баумана



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



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

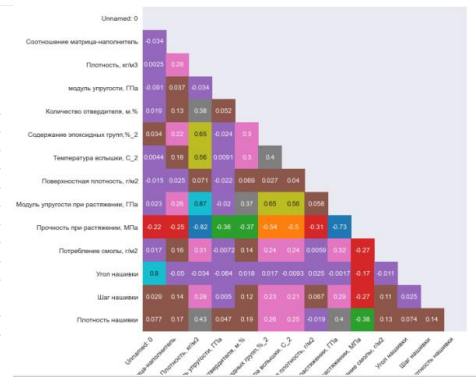
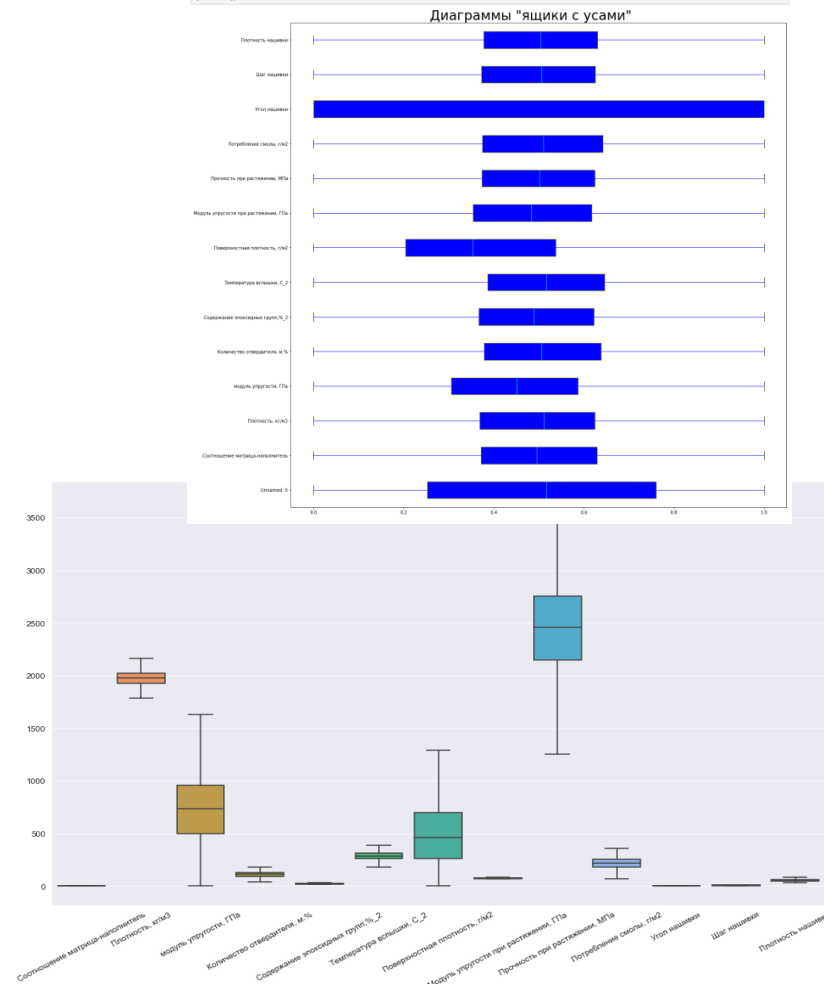
```
#Для удаления выбросов существует 2 основных метода - метод 3-х сигм
metod_3s = 0
metod_iq = 0
count_iq = [] # Список, куда записывается количество выбросов по методу 3-х сигм
count_3s = [] # Список, куда записывается количество выбросов по методу межквартильных расстояний
for column in df:
    d = 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, 'iq', ': ', d['iq'].sum())

print('Метод 3-х сигм, выбросов:', metod_3s)
print('Метод межквартильных расстояний, выбросов:', metod_iq)
```

Соотношение матрица-наполнитель
Плотность, кг/м3
модуль упругости, ГПа
Количество отвердителя, м. %
Содержание эпоксидных групп, %₂
Температура вспышки, C₂
Поверхностная плотность, г/м2
Модуль упругости при растяжении, ГПа
Прочность при растяжении, МПа
Потребление смолы, г/м2
Угол нашивки
Шаг нашивки
Плотность нашивки
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, C2                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
```



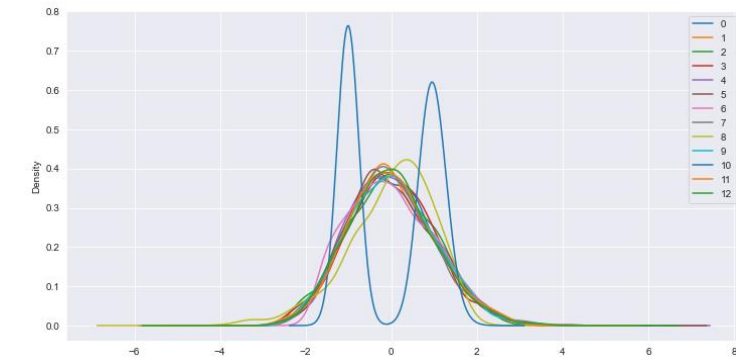
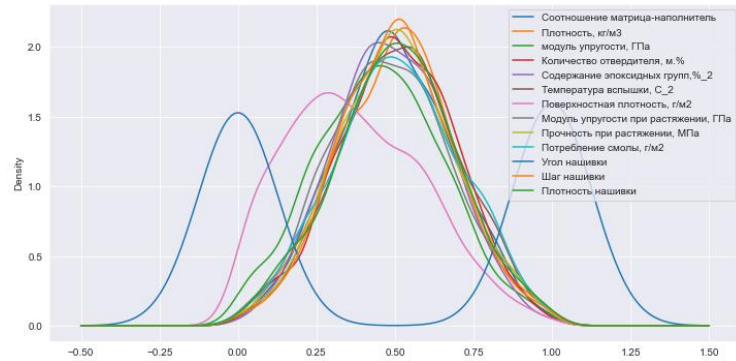
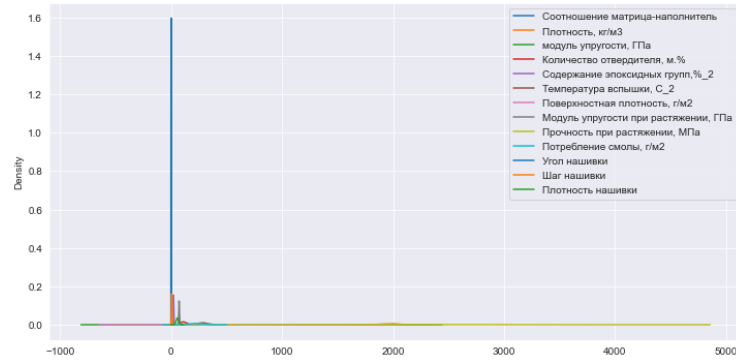
Предобработка данных:

✓ Нормализация данных:

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

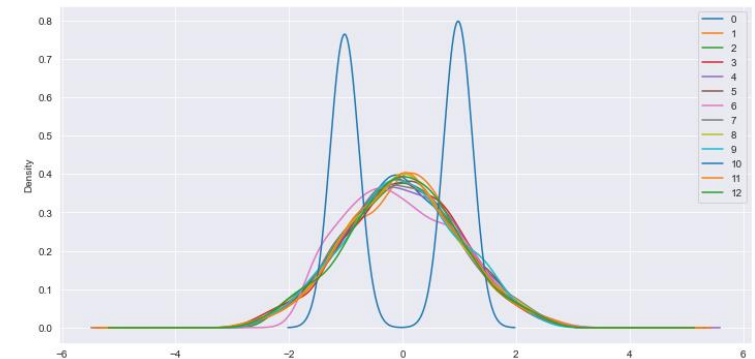
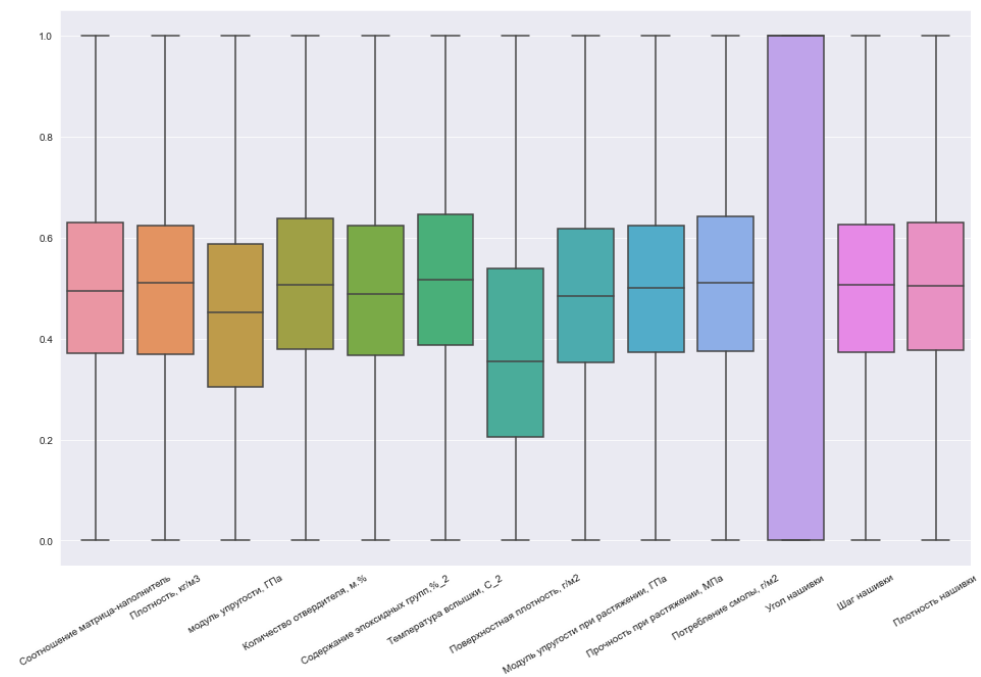
✓ Стандартизация данных:

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



df_standart_1

	0	1	2	3	4	5	6	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

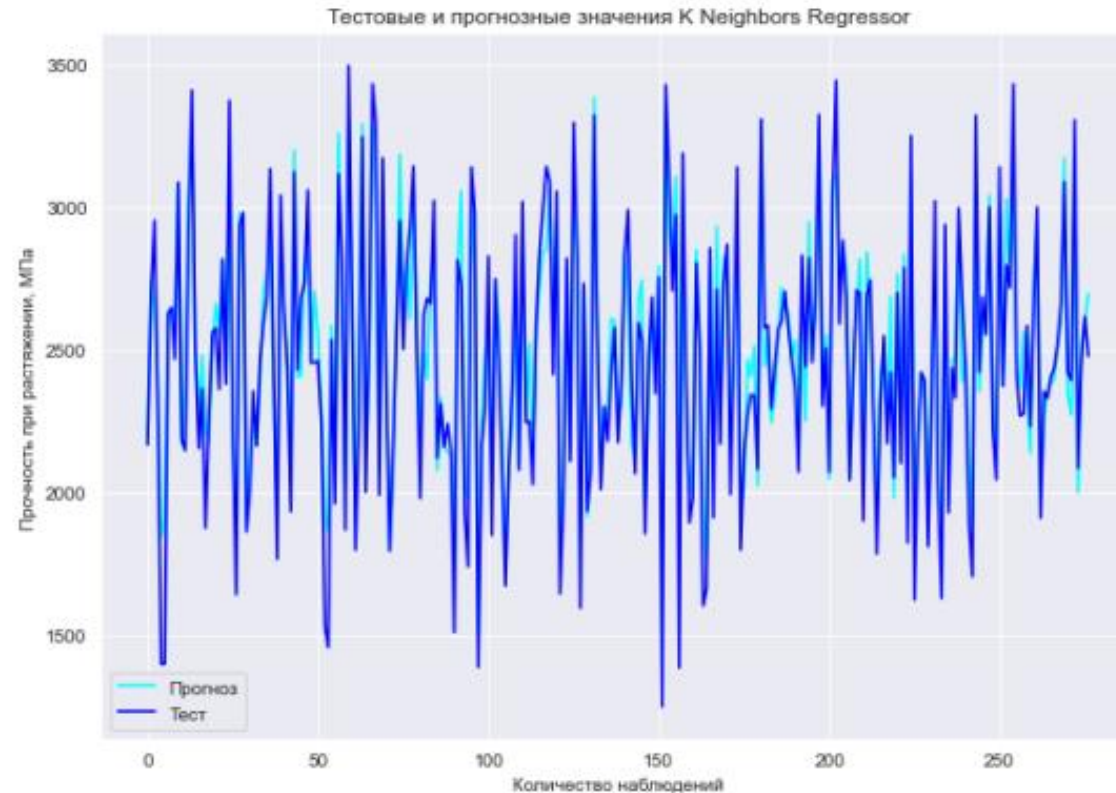
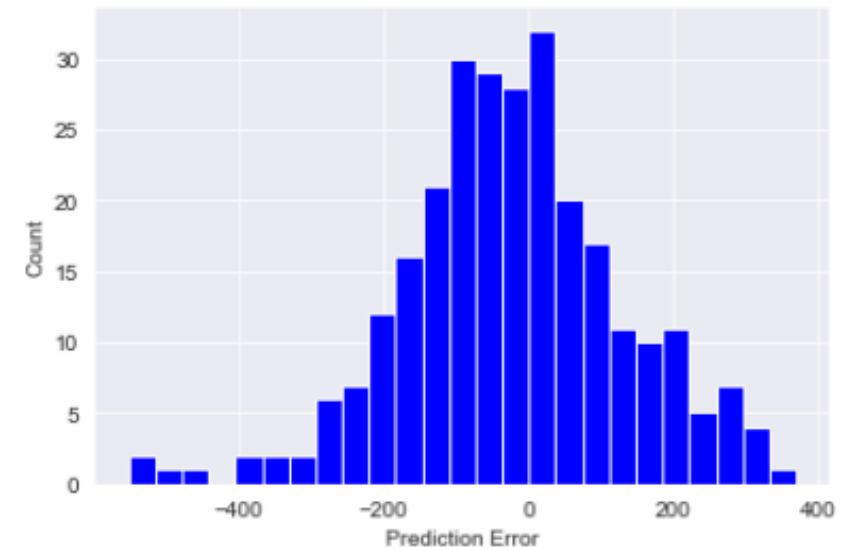


Разработка и обучение моделей для прогноза прочности при растяжении:

✓ 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

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



```
# 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)
mse_knn = mean_squared_error(y_test_1, y_pred_knn)
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_RMSE: {:.2f}'.format(np.sqrt(mse_knn)))
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_RMSE: 129.32
Test score: 0.92



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

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



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

```
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 R2
# Если R2 < 0, это значит, что разработанная модель даёт прогноз даже хуже, чем просто
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 10800 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 дерева 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)

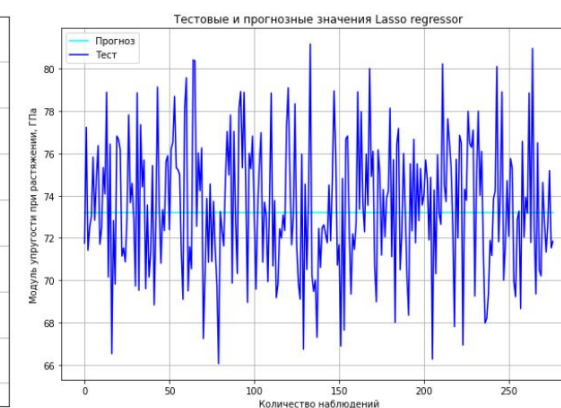
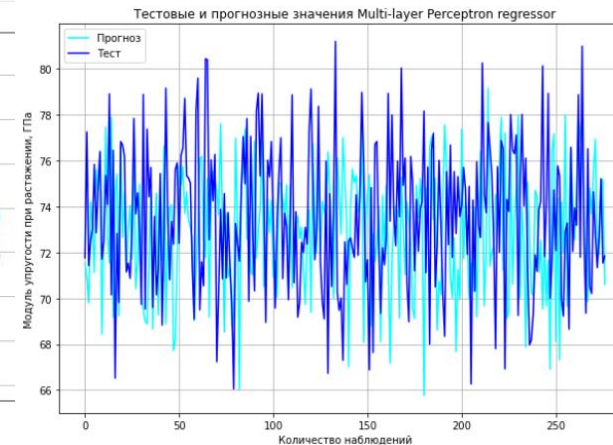
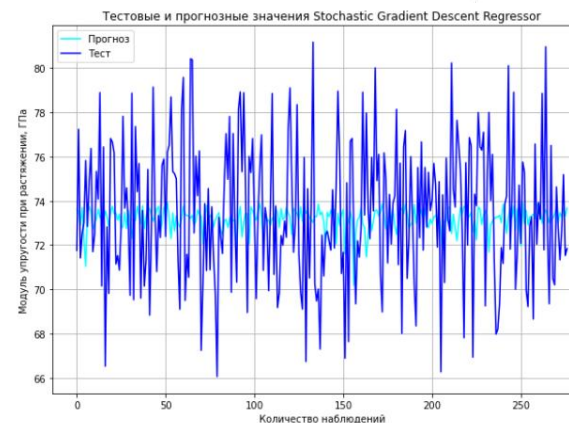
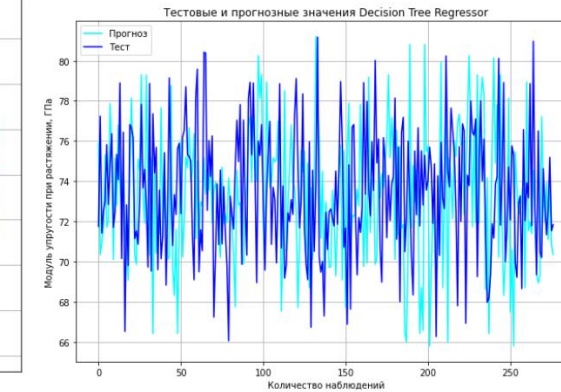
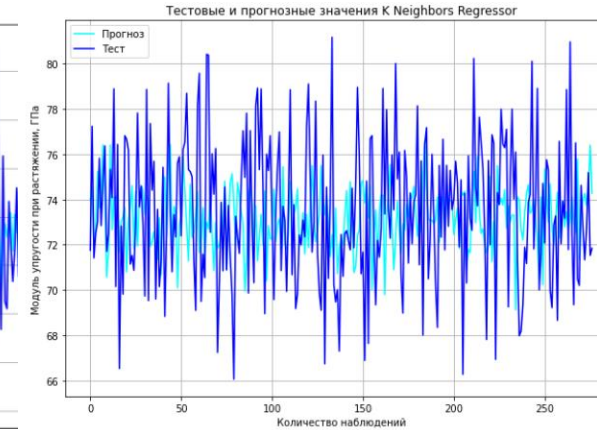
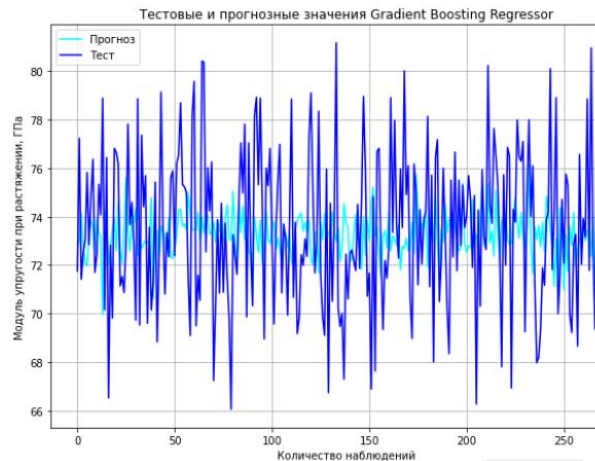
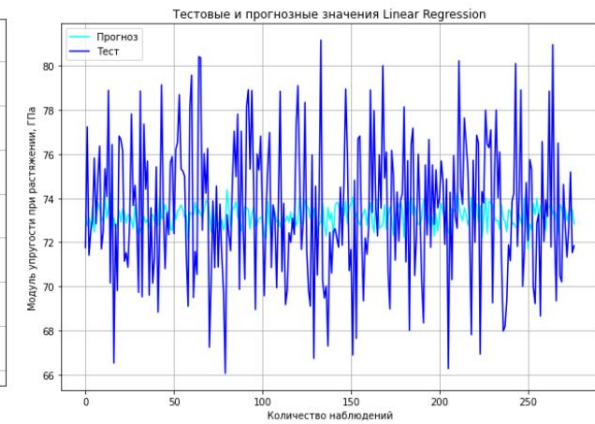
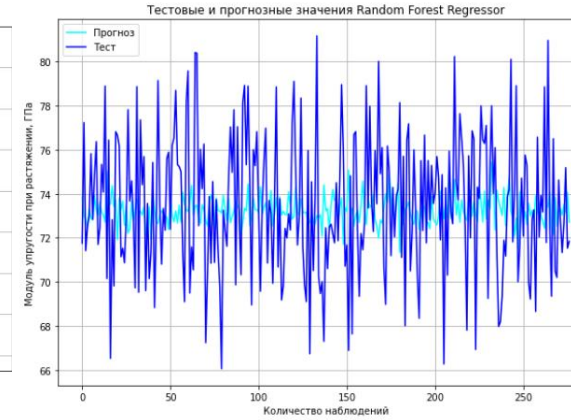
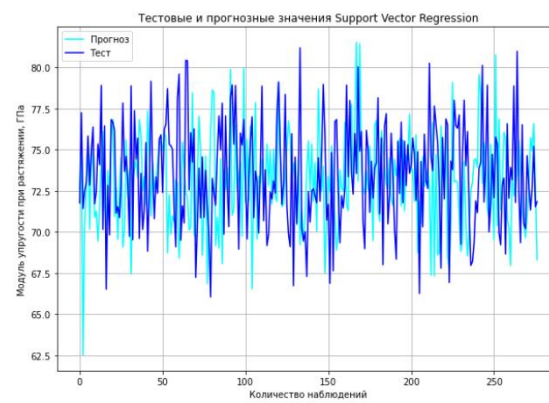
predictions_dtr_grid = dtr_grid.predict(x_test_1)
# Оцениваем точность на тестовом наборе
mae_dtr_grid = mean_absolute_error(predictions_dtr_grid, y_test_1)
mae_dtr_grid
```

168.6249974156563

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;
- 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



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

```
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())])

```
# Проведем поиск по сетке гиперпараметров с перекрестной проверкой, количество блоков равно 2
# модели случайного леса - Random Forest Regressor - 2
```

```
parameters = { 'n_estimators': [200, 300],
                'max_depth': [9, 15],
                'max_features': ['auto'],
                'criterion': ['mse'] }
grid21 = GridSearchCV(estimator = rfr2, param_grid = parameters, 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

	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

Нейронная сеть для соотношения «матрица-наполнитель»:

✓ Первая модель:

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



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

```
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
```

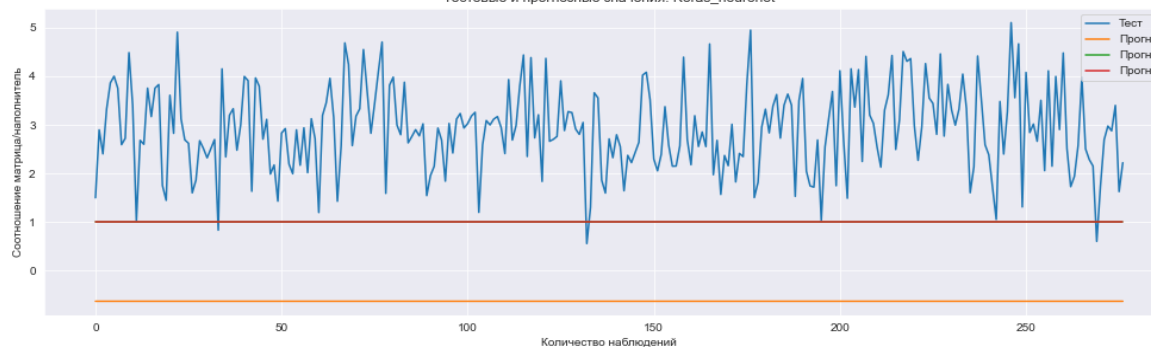
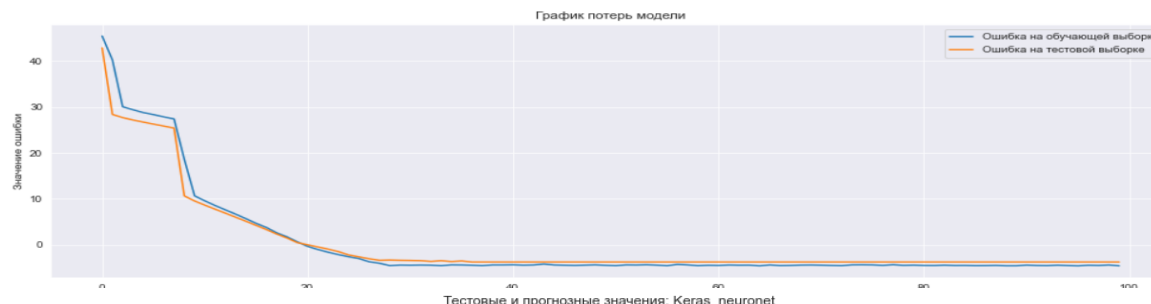
```
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['mae', 'accuracy'])
```

```
    return model
```

```
# 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»:

✓ 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

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.evaluate(x_test, y_test)

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



Model: "sequential"

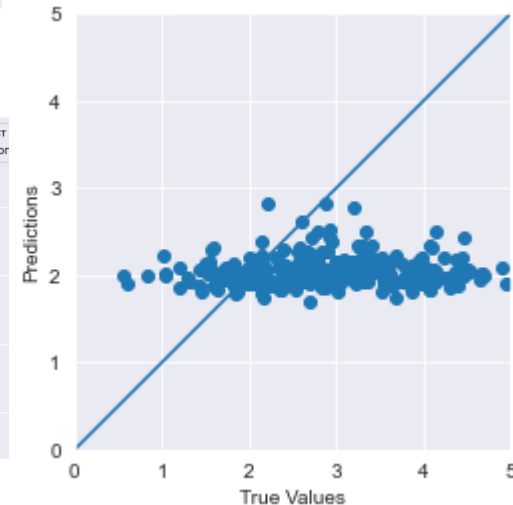
Layer (type)	Output Shape	Param #
normalization (Normalization)	(None, 12)	25
dense (Dense)	(None, 128)	1664
dense_1 (Dense)	(None, 128)	16512
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 32)	2080
dense_5 (Dense)	(None, 16)	528
dense_6 (Dense)	(None, 1)	17

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

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



Приложение:

✓ Пользовательское приложение

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

- Приложение успешно работает

✓ Репозиторий на github.com

- <https://github.com/MitroshkinaKV/KOMPOSIT>



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

Приложение разработано: Митрошкина Кристина Вячеславовна - слушатель курса «Data Science» Образовательного центра Московского государственного технического университета им. Н.Э. Баумана ©, 2023г.

Прогнозное значение параметра «Соотношение матрица-наполнитель»	
Заполните ячейки и нажмите "Готово"	
Плотность, кг/м3	1.8
Модуль упругости, ГПа	738.73
Количество отвердителя, м. %	30
Содержание эпоксидных групп, % 2	22.2678
Температура вспышки, С 2	100
Поверхностная плотность, г/м2	210
Модуль упругости при растяжении, ГПа	70
Прочность при растяжении, МПа	3000
Потребление смолы, г/м2	220
Угол нашивки, град	0
Шаг нашивки	4
Плотность нашивки	57
Готово	
Результат прогноза:	
14.428331	

Overview Repositories 1 Projects Packages Stars

Popular repositories

KOMPOSIT Public

Прогнозирование конечных свойств новых материалов (композиционных материалов)

3 contributions in the last year

Mar Apr May Jun Jul Aug Sep Oct Nov Dec

Mon

Wed

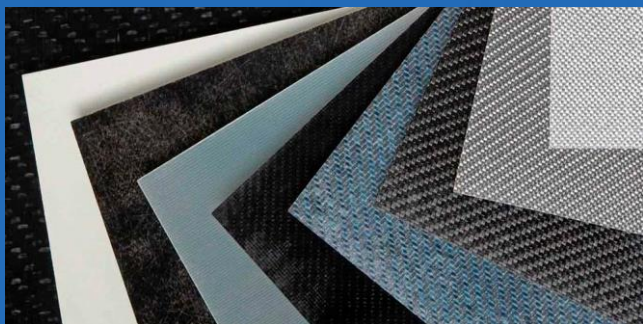
Fri

Learn how we count contributions

Contribution activity

March 2023

Модуль упругости при растяжении, ГПа	70
Прочность при растяжении, МПа	3000
Потребление смолы, г/м2	220
Угол нашивки, град	0
Шаг нашивки	4
Плотность нашивки	57
Готово	
Результат прогноза:	
14.428331	



Заключение

- Используемые 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.

Спасибо
за
Внимание!



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