ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу

«Data Science»

на тему

«Прогнозирование конечных свойств новых материалов (композиционных материалов)

Слушатель: Яманкин Андрей Геннадьевич

Постановка задачи

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».

Актуальность: Созданные прогнозные модели помогут сократить количество проводимых испытаний, а также пополнить базу данных материалов возможными новыми характеристиками материалов, и цифровыми двойниками новых композитов.

Объект исследования - процесс прогнозирования конечных свойств новых материалов.

Предмет исследования – автоматизация процесса прогнозирования конечных свойств новых материалов.

Исходные данные для анализа:

- Использованы производственные данные Центра НТИ «Цифровое материаловедение: новые материалы и вещества».
- Информация представлена в виде двух файлов формата excel: X_bp.xlsx (характеристики базальтопластика) и X_nup.xlsx (характеристики нашивки из углепластика).
- Для работы указанные датасеты объединены в один по индексу по типу объединения INNER с удалением 17 строк второго датасета.

Ввод [4]:
```
# Загрузка датасета из файла "X_bp.xlsx" и вывод 5 первых строк
dataset_bp = pd.read_excel('C:/Users/AYAmankin/Desktop/Курс Data science МГТУ/ВКР/ВКР/Datasets/X_bp.xlsx')
dataset_bp.head()
```

Out[4]:

	Unnamed: 0	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	0	1.857143	2030.0	738.736842	30.00	22.267857	100.000000	210.0	70.0	3000.0	220.0
1	1	1.857143	2030.0	738.736842	50.00	23.750000	284.615385	210.0	70.0	3000.0	220.0
2	2	1.857143	2030.0	738.736842	49.90	33.000000	284.615385	210.0	70.0	3000.0	220.0
3	3	1.857143	2030.0	738.736842	129.00	21.250000	300.000000	210.0	70.0	3000.0	220.0
4	4	2.771331	2030.0	753.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0	220.0

Ввод [5]:
```
# вывод размерности первого датасета
dataset_bp.shape
```

Out[5]:
```
(1023, 11)
```

Ввод [11]:
```
# Объединим датасеты с помощью метода merge(), тип объединения INNER и выведем первые 5 строк
dataset = pd.merge(dataset_bp, dataset_nup,
                   left_index=True,
                   right_index=True,
                   how = "inner")
dataset.head()
```

Out[11]:

	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, град	наш
0	1.857143	2030.0	738.736842	30.00	22.267857	100.000000	210.0	70.0	3000.0	220.0	0	
1	1.857143	2030.0	738.736842	50.00	23.750000	284.615385	210.0	70.0	3000.0	220.0	0	
2	1.857143	2030.0	738.736842	49.90	33.000000	284.615385	210.0	70.0	3000.0	220.0	0	
3	1.857143	2030.0	738.736842	129.00	21.250000	300.000000	210.0	70.0	3000.0	220.0	0	
4	2.771331	2030.0	753.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0	220.0	0	

Ввод [12]:
```
# выведем размерность объединенного датасета
dataset.shape
```

Out[12]:
```
(1023, 13)
```

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Разведочный анализ данных

- ✓ **Разведочный анализ данных:**
- Статистические характеристики объединенного датасета;
- Оценка типов данных;
- Проверка на наличие пропусков данных;
- Подсчет уникальных значений для каждой характеристики.

Ввод [14]:
```
# проверка на полноту данных по столбцам
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   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
```

Ввод [18]:
```
# Статистические характеристики датасета
df.describe()
```

Out[18]:

	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	У нашив г
count	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000
mean	2.930366	1975.734888	739.923233	110.570769	22.244390	285.882151	482.731833	73.328571	2466.922843	218.423144	0.491
std	0.913222	73.729231	330.231581	28.295911	2.406301	40.943260	281.314690	3.118983	485.628006	59.735931	0.500
min	0.389403	1731.764635	2.436909	17.740275	14.254985	100.000000	0.603740	64.054061	1036.856605	33.803026	0.000
25%	2.317887	1924.155467	500.047452	92.443497	20.608034	259.066528	266.816645	71.245018	2135.850448	179.627520	0.000
50%	2.906878	1977.621657	739.664328	110.564840	22.230744	285.896812	451.864365	73.268805	2459.524526	219.198882	0.000
75%	3.552660	2021.374375	961.812526	129.730366	23.961934	313.002106	693.225017	75.356612	2767.193119	257.481724	1.000
max	5.591742	2207.773481	1911.536477	198.953207	33.000000	413.273418	1399.542362	82.682051	3848.436732	414.590628	1.000

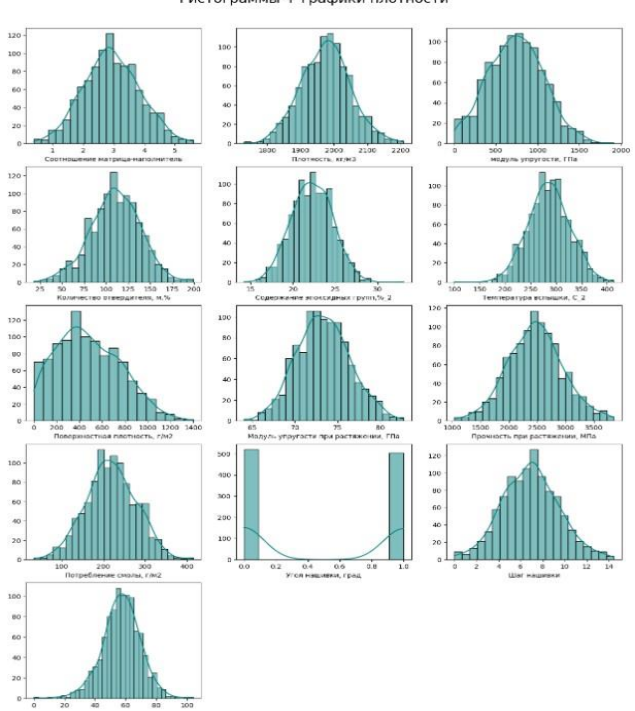
Ввод [15]:
```
# Оценим кол-во уникальных значений для признаков
df.nunique()
```

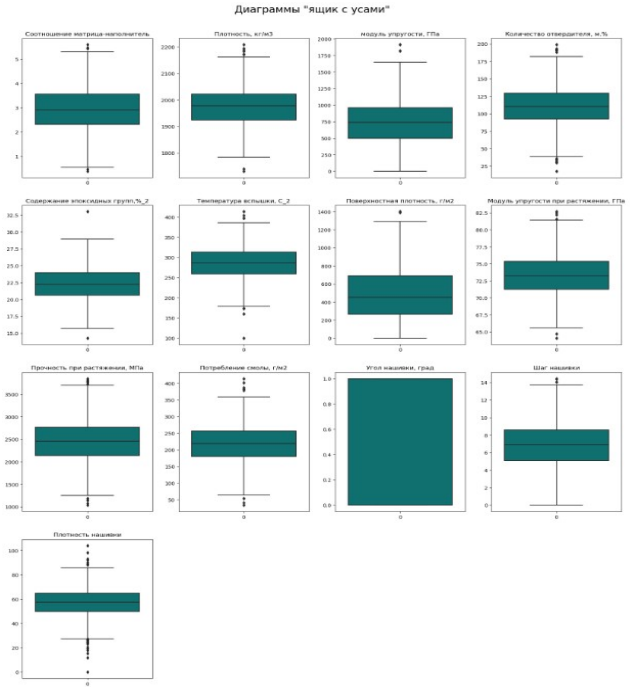
Out[15]:
```
Соотношение матрица-наполнитель        1014
Плотность, кг/м3                       1013
модуль упругости, ГПа                  1020
Количество отвердителя, м.%            1005
Содержание эпоксидных групп,%_2        1004
Температура вспышки, С_2               1003
Поверхностная плотность, 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
```

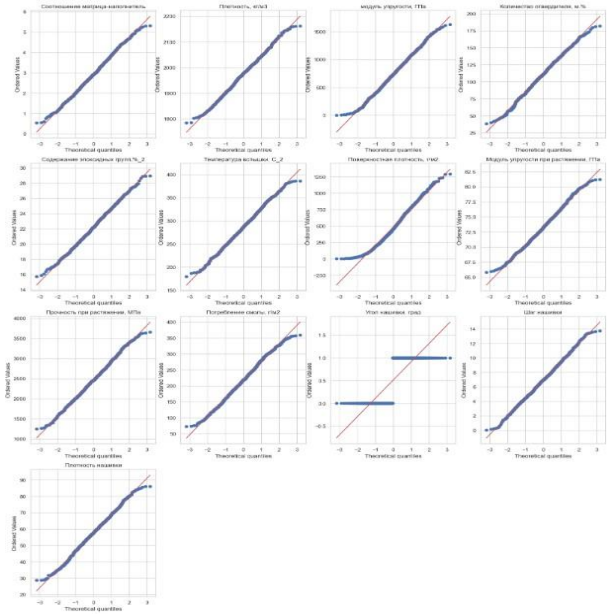
Визуализация исходных данных:

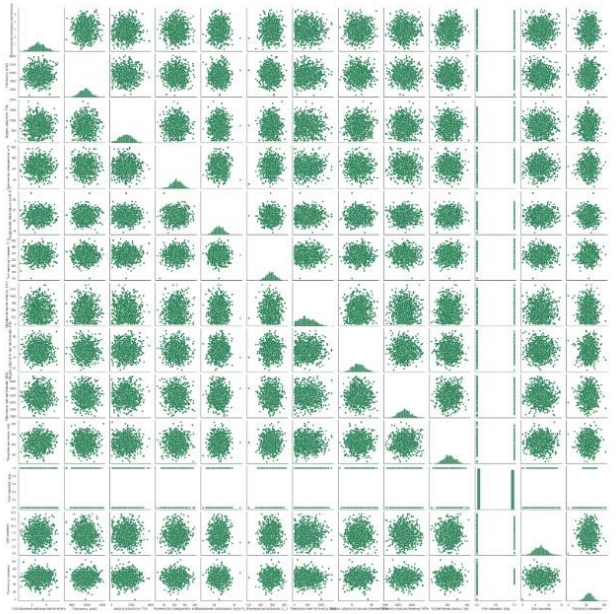
- гистограммы и графики плотности
- диаграмма «ящик с усами»;
- попарные графики рассеяния точек;
- график квантиль-квантиль;
- тепловая карта.

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

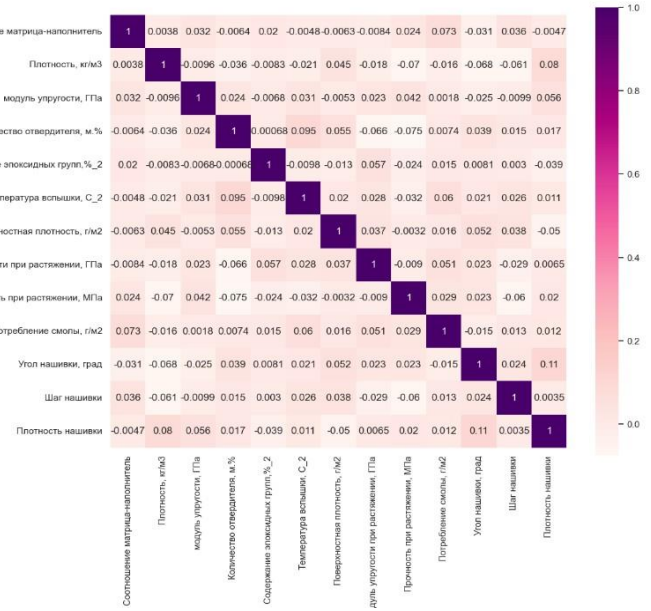








Ввод [26]:
```
# Визуализация матрицы корреляции в виде тепловой карты
f, ax = plt.subplots(figsize=(12,10))
sns.heatmap(df.corr(), annot=True, ax=ax, square = True, cmap='RdPu')
plt.show()
```



Предобработка данных:

✓ Исключение выбросов:

- Проверка датасета на выбросы;
- Выбор и обоснование метода очистки от выбросов;
- Исключение выбросов методом межквартильного диапазона (3 итерации);
- Проверка очищенного датасета на выбросы.

Ввод [27]:

```
# Метод межквартильного диапазона, предварительная оценка кол-ва выбросов
def detect_outliers_IQR(data):
    outliers = []
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    for i in data:
        if (i <= lower_bound) | (i >= upper_bound):
            outliers.append(i)
    return outliers

sum_of_outliers = 0
for col in df.columns:
    df_outliers_IQR = detect_outliers_IQR(df[col])
    sum_of_outliers += len(df_outliers_IQR)
    print("Выбросов в столбце ", df[col].name, ": ", len(df_outliers_IQR))
print("Итого выбросов: ", sum_of_outliers)
```

```
Выбросов в столбце  Соотношение матрица-наполнитель :  6
Выбросов в столбце  Плотность, кг/м3 :  9
Выбросов в столбце  модуль упругости, ГПа :  2
Выбросов в столбце  Количество отвердителя, м.% :  14
Выбросов в столбце  Содержание эпоксидных групп,%_2 :  2
Выбросов в столбце  Температура вспышки, С_2 :  8
Выбросов в столбце  Поверхностная плотность, г/м2 :  2
Выбросов в столбце  Модуль упругости при растяжении, ГПа :  6
Выбросов в столбце  Прочность при растяжении, МПа :  11
Выбросов в столбце  Потребление смолы, г/м2 :  8
Выбросов в столбце  Угол нашивки, град :  0
Выбросов в столбце  Шаг нашивки :  4
Выбросов в столбце  Плотность нашивки :  21
Итого выбросов:  93
```

Ввод [35]:

```
sum_of_outliers_4 = 0
for col in df_clean.columns:
    df_outliers_IQR = detect_outliers_IQR(df_clean[col])
    sum_of_outliers_4 += len(df_outliers_IQR)
    print("Выбросов в столбце ", df_clean[col].name, ": ", len(df_outliers_IQR))
print("Итого выбросов: ", sum_of_outliers_4)
```

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

Предобработка данных:

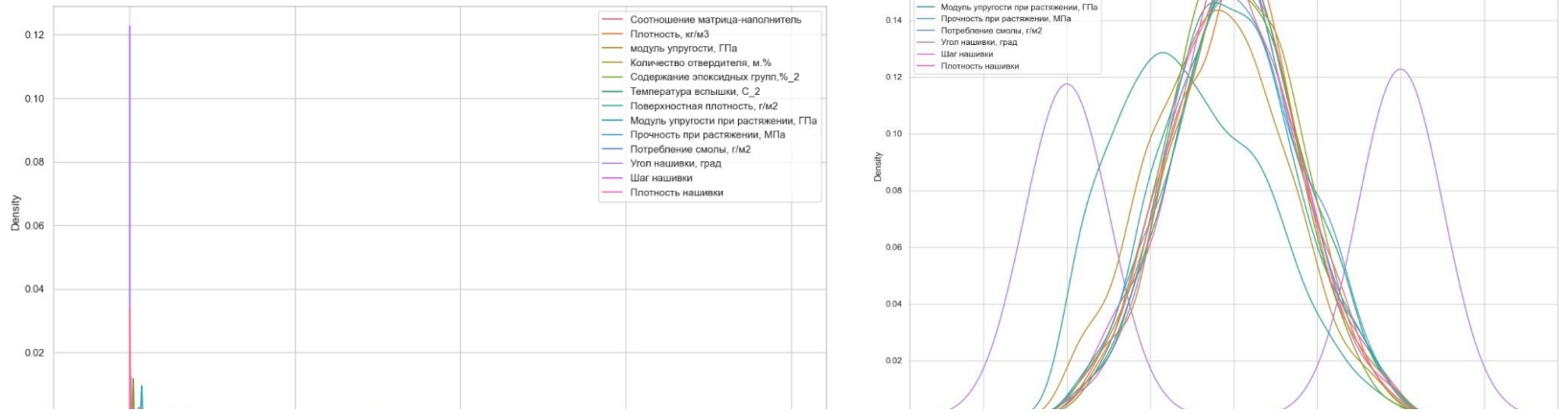
✓ Нормализация и стандартизация данных:

- Тестирование на нормальность исходного датасета;
- Нормализация данных с использованием MinMaxScaler;
- Стандартизация данных с использованием StandartScaler;
- Оценка полученного результата

Ввод [41]:
```
# Проведем тест Шапиро-Уилка на нормальность
for col in df_clean.columns:
    print(df_clean[col].name, shapiro(df_clean[col]))
```

```
Соотношение матрица-наполнитель ShapiroResult(statistic=0.9971880316734314, pvalue=0.10904344916343689)
Плотность, кг/м3 ShapiroResult(statistic=0.997011661529541, pvalue=0.08352091163396835)
модуль упругости, ГПа ShapiroResult(statistic=0.995254397392273, pvalue=0.0058130305260419846)
Количество отвердителя, м.% ShapiroResult(statistic=0.9966756105422974, pvalue=0.05000615119934082)
Содержание эпоксидных групп,%_2 ShapiroResult(statistic=0.9977133870124817, pvalue=0.23541033267974854)
Температура вспышки, С_2 ShapiroResult(statistic=0.9971266984939575, pvalue=0.09941922873258591)
Поверхностная плотность, г/м2 ShapiroResult(statistic=0.9776217937469482, pvalue=1.0688074730813568e-10)
Модуль упругости при растяжении, ГПа ShapiroResult(statistic=0.9955782890319824, pvalue=0.009416559711098671)
Прочность при растяжении, МПа ShapiroResult(statistic=0.9973730444908142, pvalue=0.14375117421150208)
Потребление смолы, г/м2 ShapiroResult(statistic=0.9955164790153503, pvalue=0.008584197610616684)
Угол нашивки, град ShapiroResult(statistic=0.6364129781723022, pvalue=2.8589291269154918e-40)
Шаг нашивки ShapiroResult(statistic=0.9980173110961914, pvalue=0.3559962809085846)
Плотность нашивки ShapiroResult(statistic=0.9967383742332458, pvalue=0.05505112186074257)
```

Ввод [39]:
```
# Построим графики распределения всех параметров
plt.figure(figsize=(15,8))
sns.set(context='notebook', style='whitegrid')
sns.kdeplot(data=df_clean)
plt.show()
```



Ввод [57]:
```
# Тест Шапиро-Уилка на нормальность
for col in df_norm_minmax.columns:
    print(df_norm_minmax[col].name, shapiro(df_norm_minmax[col]))
```

```
Соотношение матрица-наполнитель ShapiroResult(statistic=0.9971882104873657, pvalue=0.10907094180583954)
Плотность, кг/м3 ShapiroResult(statistic=0.997011661529541, pvalue=0.08352091163396835)
модуль упругости, ГПа ShapiroResult(statistic=0.9952542185783386, pvalue=0.005811706185340881)
Количество отвердителя, м.% ShapiroResult(statistic=0.9966757893562317, pvalue=0.050019025802612305)
Содержание эпоксидных групп,%_2 ShapiroResult(statistic=0.9977133870124817, pvalue=0.23541033267974854)
Температура вспышки, С_2 ShapiroResult(statistic=0.9971266984939575, pvalue=0.09941922873258591)
Поверхностная плотность, г/м2 ShapiroResult(statistic=0.9776219129562378, pvalue=1.0689168300492824e-10)
Модуль упругости при растяжении, ГПа ShapiroResult(statistic=0.9955782890319824, pvalue=0.009416559711098671)
Прочность при растяжении, МПа ShapiroResult(statistic=0.9973730444908142, pvalue=0.14375117421150208)
Потребление смолы, г/м2 ShapiroResult(statistic=0.995516300201416, pvalue=0.008582196198403835)
Угол нашивки, град ShapiroResult(statistic=0.6364129781723022, pvalue=2.8589291269154918e-40)
Шаг нашивки ShapiroResult(statistic=0.9980174899101257, pvalue=0.3560756742954254)
Плотность нашивки ShapiroResult(statistic=0.9967382550239563, pvalue=0.055037494748830795)
```

Разработка и обучение моделей:

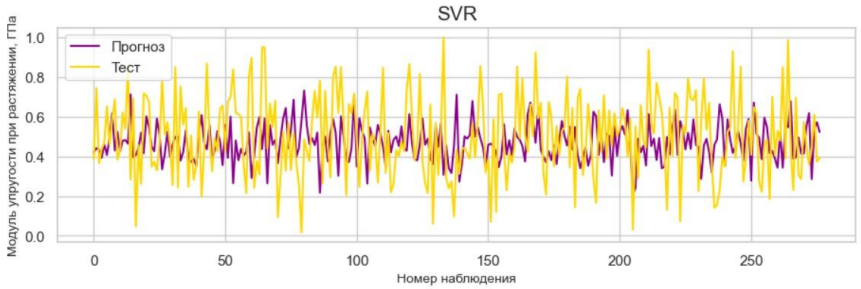
- ✓ **Прогнозирование модуля упругости при растяжении:**

- Разбиение данных на тестовую и тренировочную выборки;
- Анализ работы различных моделей на стандартных параметрах.

Ввод [61]:
```
# Разделим датасет на тренировочную и тестовую выборки.
# При построении модели 30% данных оставим на тестирование модели, на остальных происходит обучение моделей.
X_train_elastic, X_test_elastic, y_train_elastic, y_test_elastic = train_test_split(
                                                                   df_norm.drop(['Модуль упругости при растяжении, ГПа', 'Прочно
                                                                   df_norm[['Модуль упругости при растяжении, ГПа']],
                                                                   test_size = 0.3,
                                                                   random_state = 42,
                                                                   shuffle = True
                                                                   )
```

Ввод [62]:
```
# Посмотрим размерность тренировочной и тестовой выборок
print('Размер тренировочного датасета: {}\nРазмер тестового датасета:{}'.format(X_train_elastic.shape, X_test_elastic.shape))
```

```
Размер тренировочного датасета: (645, 11)
Размер тестового датасета:(277, 11)
```



Разработка и обучение моделей:

✓ **Прогнозирование модуля упругости при растяжении:**

- Поиск гиперпараметров моделей (метод поиска по сетке GridSearch с перекрестной проверкой, кол-во блоков = 10)

```
# Создаем словарь с наборами гиперпараметров всех моделей
all_params = {'kneighborsregressor':       {'kneighborsregressor__n_neighbors': [i for i in range(1, 201, 2)],
                                            'kneighborsregressor__weights':     ['uniform', 'distance'],
                                            'kneighborsregressor__algorithm':   ['auto', 'ball_tree', 'kd_tree', 'brute']
                                           },
              'svr':                       {'svr__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
                                            'svr__C':      [0.01, 0.1, 1],
                                            'svr__gamma':  [0.01, 0.1, 1]
                                           },
              'linearregression':          {'linearregression__fit_intercept': [True, False]},
              'decisiontreeregressor':     {'decisiontreeregressor__max_depth':         [3, 5, 7, 9, 11, 13, 15],
                                            'decisiontreeregressor__min_samples_leaf':  [1, 2, 5, 10, 20, 50, 100, 150, 200],
                                            'decisiontreeregressor__min_samples_split': [200, 250, 300],
                                            'decisiontreeregressor__max_features':      ['auto', 'sqrt', 'log2']
                                           },
              'adaboostregressor':         {'adaboostregressor__base_estimator__max_depth':        [i for i in range(2, 11,
                                            'adaboostregressor__base_estimator__min_samples_leaf': [5,  10],
                                            'adaboostregressor__n_estimators':                     [10, 50, 100, 250, 1000],
                                            'adaboostregressor__learning_rate':                    [0.01, 0.05, 0.1, 0.5]
                                           },
              'gradientboostingregressor': {'gradientboostingregressor__learning_rate': [0.01,  0.02, 0.03, 0.04],
                                            'gradientboostingregressor__subsample'    : [0.9, 0.5, 0.2, 0.1],
                                            'gradientboostingregressor__n_estimators' : [100, 500, 1000, 1500],
                                            'gradientboostingregressor__max_depth'    : [4, 6, 8, 10]
                                           },
              'xgbregressor':              {'xgbregressor__learning_rate':    [0.05, 0.10, 0.15],
                                            'xgbregressor__max_depth':        [ 3, 4, 5, 6, 8],
                                            'xgbregressor__min_child_weight': [ 1, 3, 5, 7],
                                            'xgbregressor__gamma':            [ 0.0, 0.1, 0.2],
                                            'xgbregressor__colsample_bytree': [ 0.3, 0.4]
                                           },
              'randomforestregressor':     {'randomforestregressor__n_estimators':     [30, 100, 200, 300, 500],
                                            'randomforestregressor__max_depth':        [1, 2, 3, 4, 5, 6, 7, 8],
                                            'randomforestregressor__min_samples_leaf': [1, 2],
                                            'randomforestregressor__max_features':     ['auto', 'sqrt', 'log2']
                                           },
              'sgdregressor':              {'sgdregressor__penalty': ['l2', 'l1', 'elasticnet', None],
                                            'sgdregressor__alpha':   [0.0001, 0.001, 0.01, 0.1]
                                           },
              'lasso':                     {'lasso__alpha': [0.01, 0.02, 0.1, 0.2, 0.03, 0.3, 0.05, 0.5, 0.07, 0.7, 1]}
```

```
Регрессор с лучшим значением R2 = -0.0025 на тестовой выборке: AdaBoostRegressor(base_estimator=DecisionTreeRegressor())

Лучший алгоритм:
Pipeline(steps=[('adaboostregressor',
                 AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=2,
                                                                        min_samples_leaf=10),
                                   learning_rate=0.5, n_estimators=10))])
```

Разработка и обучение моделей:

✓ **Прогнозирование модуля упругости при растяжении:**

- Оценка качества работы моделей

✓ **Прогнозирование прочности при растяжении:**

- Оценка качества работы моделей

Ввод [70]:

```
# Рассчитаем метрики для моделей с их лучшими параметрами:
KNR_best = KNeighborsRegressor(algorithm='auto', n_neighbors=199, weights='uniform')
SVReg_best = SVR(C=0.01, gamma=1, kernel='rbf')
LR_best = LinearRegression(fit_intercept=True)
DTR_best = DecisionTreeRegressor(max_depth=9, max_features='sqrt', min_samples_leaf=200, min_samples_split=250)
Abr_best = AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=2, min_samples_leaf=10), learning_rate=0.01, n_estima
Gbr_best = GradientBoostingRegressor(learning_rate=0.01, max_depth=4, n_estimators=100, subsample=0.5)
XgbR_best = XGBRegressor(colsample_bytree=0.3, gamma=0.2, learning_rate=0.05, max_depth=3, min_child_weight=5)
RFR_best = RandomForestRegressor(max_depth=1, max_features='log2', min_samples_leaf=2, n_estimators=100)
SGDR_best = SGDRegressor(alpha=0.1, penalty='l1')
LassoR_best = Lasso(alpha=0.01)

Model_Comparision_Train_Test([KNR_best, SVReg_best, LR_best, DTR_best, Abr_best, Gbr_best, XgbR_best, RFR_best, SGDR_best, LassoF
```

Out[70]:

Model	R2_score	MAE	MSE	RMSE	MAPE
KNeighborsRegressor	-0.0105 (0.0091)	0.17 (0.15)	0.04 (0.04)	0.21 (0.19)	0.65 (3324567063626.71)
SVR	-0.0146 (0.0326)	0.17 (0.15)	0.04 (0.04)	0.21 (0.19)	0.65 (3293971898525.6)
LinearRegression	-0.0245 (0.0172)	0.17 (0.15)	0.04 (0.04)	0.21 (0.19)	0.65 (3530183592566.66)
DecisionTreeRegressor	0.006 (0.0061)	0.17 (0.15)	0.04 (0.04)	0.2 (0.19)	0.65 (3479792181714.39)
AdaBoostRegressor	-0.0066 (0.0228)	0.17 (0.15)	0.04 (0.04)	0.2 (0.19)	0.65 (3231476794929.06)
GradientBoostingRegressor	-0.017 (0.1559)	0.17 (0.14)	0.04 (0.03)	0.21 (0.18)	0.65 (2959719650427.48)
XGBRegressor	-0.0218 (0.0918)	0.17 (0.15)	0.04 (0.03)	0.21 (0.18)	0.66 (3274354675248.97)
RandomForestRegressor	-0.0106 (0.0171)	0.17 (0.15)	0.04 (0.04)	0.21 (0.19)	0.66 (3366865472577.18)
SGDRegressor	-0.0092 (-0.0)	0.17 (0.16)	0.04 (0.04)	0.21 (0.19)	0.65 (3359840987530.74)
Lasso	-0.0089 (0.0)	0.17 (0.16)	0.04 (0.04)	0.21 (0.19)	0.65 (3362407323544.2)

Ввод [79]:

```
# Рассчитаем метрики для моделей с их лучшими параметрами:
KNR_best = KNeighborsRegressor(algorithm='auto', n_neighbors=159, weights='distance')
SVReg_best = SVR(C=0.01, gamma=0.1, kernel='poly')
LR_best = LinearRegression(fit_intercept=True)
DTR_best = DecisionTreeRegressor(max_depth=15, max_features='sqrt', min_samples_leaf=100, min_samples_split=250)
Abr_best = AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=2, min_samples_leaf=10), learning_rate=0.01, n_estima
Gbr_best = GradientBoostingRegressor(learning_rate=0.01, max_depth=8, n_estimators=100, subsample=0.1)
XgbR_best = XGBRegressor(colsample_bytree=0.3, gamma=0.2, learning_rate=0.05, max_depth=3, min_child_weight=5)
RFR_best = RandomForestRegressor(max_depth=1, max_features='log2', min_samples_leaf=2, n_estimators=30)
SGDR_best = SGDRegressor(alpha=0.1, penalty='l1')
LassoR_best = Lasso(alpha=0.01)

Model_Comparision_Train_Test([KNR_best, SVReg_best, LR_best, DTR_best, Abr_best, Gbr_best, XgbR_best, RFR_best, SGDR_best, LassoF
```

Out[79]:

Model	R2_score	MAE	MSE	RMSE	MAPE
KNeighborsRegressor	-0.002 (1.0)	0.15 (0.0)	0.04 (0.0)	0.19 (0.0)	8176160847752.77 (0.0)
SVR	-0.0012 (0.0012)	0.15 (0.15)	0.04 (0.03)	0.19 (0.19)	8151410332325.56 (0.69)
LinearRegression	-0.0035 (0.017)	0.15 (0.15)	0.04 (0.03)	0.19 (0.19)	9069348467609.18 (0.68)
DecisionTreeRegressor	-0.0116 (0.0284)	0.15 (0.15)	0.04 (0.03)	0.19 (0.18)	8145511233200.83 (0.68)
AdaBoostRegressor	-0.0072 (0.0501)	0.15 (0.15)	0.04 (0.03)	0.19 (0.18)	8684037540203.58 (0.67)
GradientBoostingRegressor	-0.0226 (0.161)	0.16 (0.14)	0.04 (0.03)	0.19 (0.17)	8432488024396.78 (0.63)
XGBRegressor	0.0005 (0.1155)	0.15 (0.14)	0.04 (0.03)	0.19 (0.18)	8709751775321.21 (0.65)
RandomForestRegressor	-0.0011 (0.0231)	0.15 (0.15)	0.04 (0.03)	0.19 (0.18)	8410926210985.66 (0.68)
SGDRegressor	-0.0002 (-0.0)	0.15 (0.15)	0.04 (0.03)	0.19 (0.19)	8168139964879.41 (0.69)
Lasso	-0.0001 (0.0)	0.15 (0.15)	0.04 (0.03)	0.19 (0.19)	8183534682026.43 (0.69)

Нейронная сеть для соотношения «матрица-наполнитель»:

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Ввод [83]:
```
# Создадим функцию для генерации слоев нейронной сети
def create_NN_model(layers, activation, drop, opt):
        model= Sequential()
        for i, neurons in enumerate(layers):
            if i==0:
                model.add(Dense(neurons, input_dim=X_train_matrix.shape[1], activation = activation))
            else:
                model.add(Dense(neurons, activation))
                model.add(Dropout(drop))
            model.add(Dense(1))

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

            return model
```

Ввод [85]:
```
# Построим нейронную сети с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 5 (cv = 5)
# Воспользуемся методом GridSearchCV

reg = KerasRegressor(model = create_NN_model, layers = [128], activation = 'relu', drop = 0.1, opt = 'Adam', verbose = 2)

# Зададим параметры для модели
param_grid = {'activation': ['relu', 'softmax', 'sigmoid'],
                  'layers': [[128, 64, 16], [128, 128, 64, 32], [128, 128, 64, 16]],
                    'opt': ['Adam', 'SGD'],
                   'drop': [0.0, 0.1, 0.2],
             'batch_size': [10, 20, 40],
                 'epochs': [10, 50, 100]
              }

# Произведем поиск лучших параметров
grid = GridSearchCV(estimator = reg,
                    param_grid = param_grid,
                    cv = 5,
                    verbose = 0,
                    n_jobs = -1)

grid_result = grid.fit(X_train_matrix, np.ravel(y_train_matrix))
```

Ввод [86]:
```
print('Лучший коэффициент R2: {:.4f} при использовании модели с параметрами {} \n'.format(grid_result.best_score_, grid_result.be
```

```
Лучший коэффициент R2: -0.0025 при использовании модели с параметрами {'activation': 'softmax', 'batch_size': 10, 'drop': 0.2,
'epochs': 50, 'layers': [128, 128, 64, 32], 'opt': 'SGD'}
```

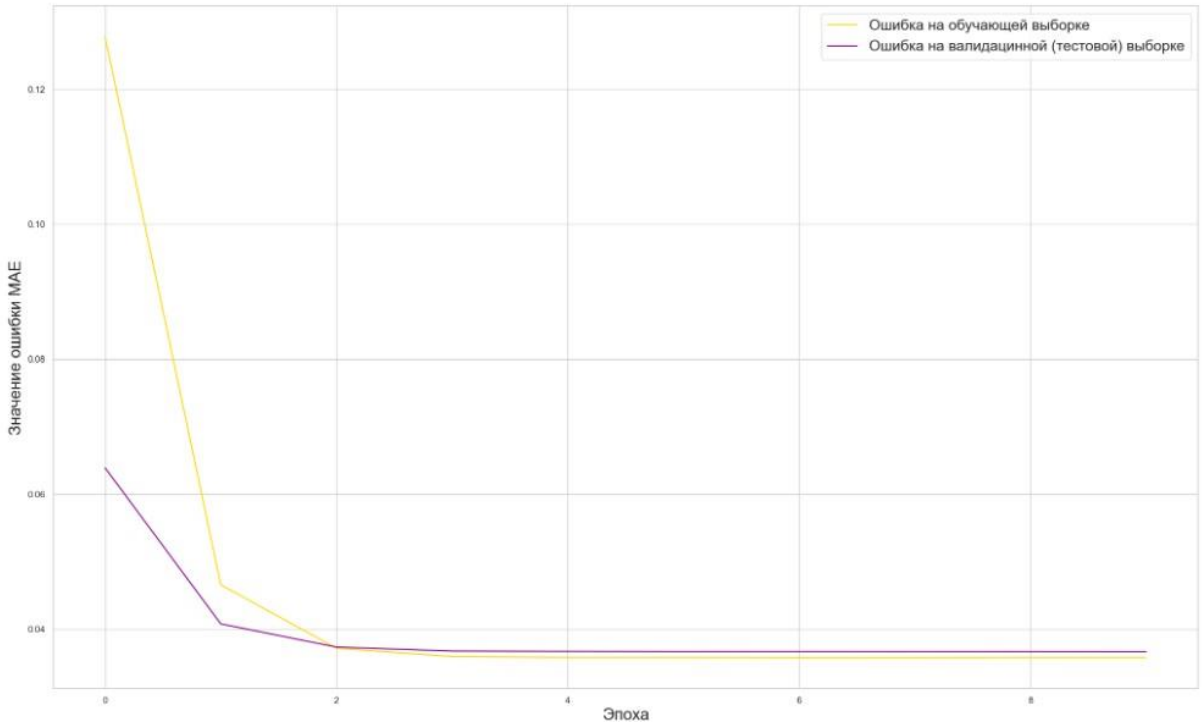
Ввод [87]:
```
# Создадим модель с полученными значениями
best_model = Sequential()
best_model.add(Dense(128, input_dim = X_train_matrix.shape[1], activation = 'softmax'))      # входной слой
best_model.add(Dense(128, activation = 'softmax'))                                           # добавляем полносвязный слой
best_model.add(Dropout(0.0))                                                                 # исключаем переобучения
best_model.add(Dense(64, activation = 'softmax'))                                            # добавляем полносвязный слой
best_model.add(Dropout(0.0))                                                                 # исключаем переобучения
best_model.add(Dense(32, activation = 'softmax'))                                            # добавляем полносвязный слой
best_model.add(Dropout(0.0))                                                                 # исключаем переобучения
best_model.add(Dense(1))                                                                     # выходной слой

# Компиляция модели: определяем метрики и алгоритм оптимизации
best_model.compile(loss = 'mse',
                   optimizer = 'SGD',
                   metrics = ['mae'])

# Обучение модели
best_history = best_model.fit(X_train_matrix, np.ravel(y_train_matrix),
                              epochs=10,
                              batch_size=10,
                              verbose=1,
                              validation_split=0.2)
```

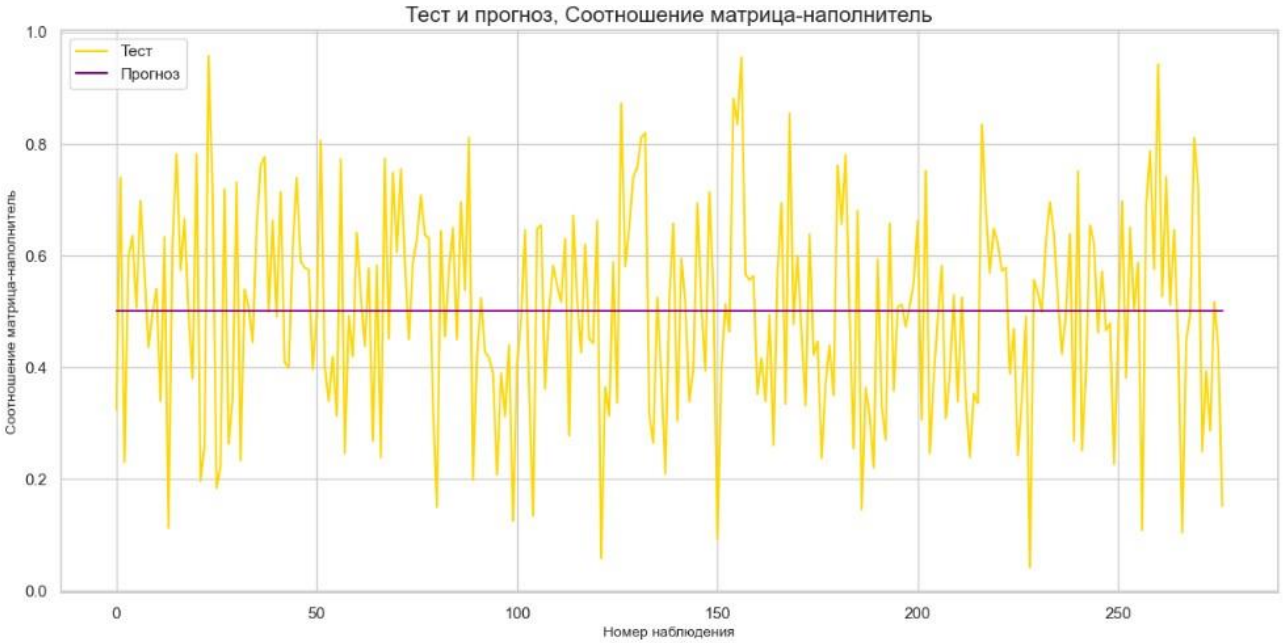
Ввод [89]:
```
# Структура нейронной сети
best_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 128)               1664

 dense_3 (Dense)             (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 32)                2080

 dropout_2 (Dropout)         (None, 32)                0

 dense_6 (Dense)             (None, 1)                 33

=================================================================
Total params: 28,545
Trainable params: 28,545
Non-trainable params: 0
_________________________________________________________________
```

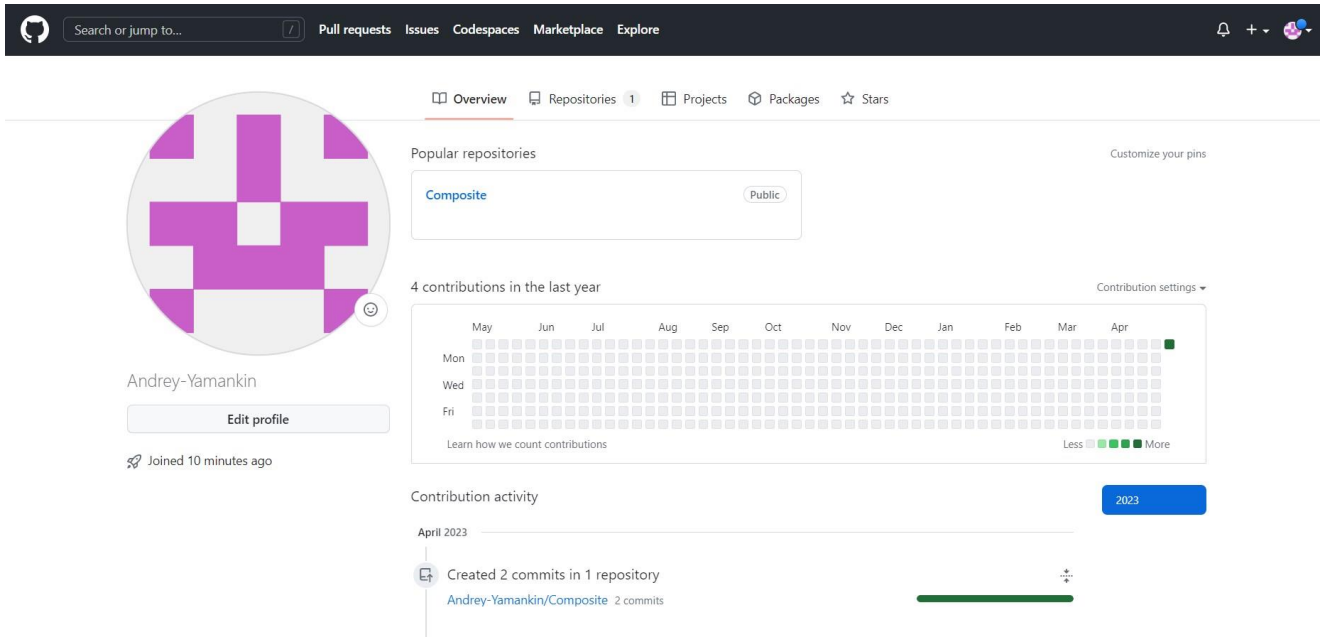
Нейронная сеть для соотношения «матрица-наполнитель»:

- График потерь на тренировочных и тестовых выборках
- Визуализация прогнозных данных для модели.





- ✓ **Пользовательское приложение**
- ✓ **Репозиторий на github.com**



ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана

Спасибо
за
Внимание!