Выпускная квалификационная работа по курсу “Data Science”

Слушатель: Андрюшина Марина Викторовна

Введение

Выпускная квалификационная работа выполнена по теме: “Прогнозирование конечных свойств новых материалов (композиционных материалов).

На входе были даны два датасета с данными о параметрах базальтопластика и данными нашивок углепластика, которые мы в дальнейшем объединили и исследовали.

В процесс работы было произведено объединение датасетов, разведочный анализ данных, выполнена предобработка данных, разработан список моделей, которые будут использоваться для прогноза модуля упругости при растяжении и прочности при растяжении, проведено тестирование модели, написана нейросеть.

Объединение файлов и разведочный анализ

Импортируем необходимые библиотеки, загружаем файлы, проверяем размерность. Объединяем по типу объединения INNER, смотрим итоговый датасет.

```
[5] #Удаление неинформативного столбца с нумерацией
    df_bp.drop(['Unnamed: 0'], axis=1, inplace=True)
    #Выведение на экран первых 5 строк первого датасета и проверка, что первый столбец удалился
    df_bp.head()
```

	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

```
#Проверка размерности после удаления первого столбца
df_bp.shape
```

```
(1023, 10)
```

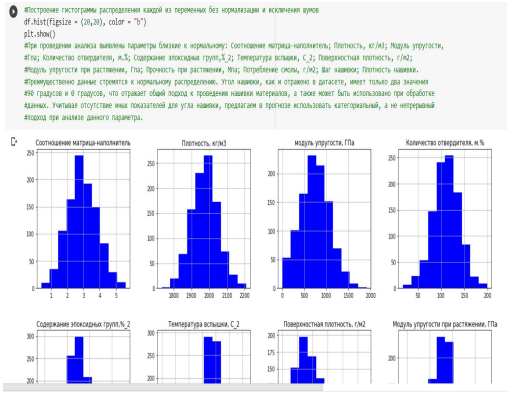
Объединение датасетов по индексу тип объединения INNER

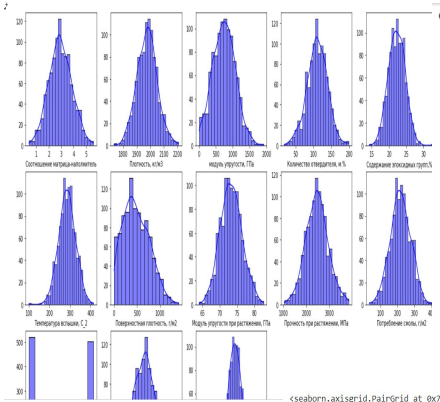
```
df = df_bp.merge(df_nup, left_index = True, right_index = True, how = 'inner')
df.head().T
```

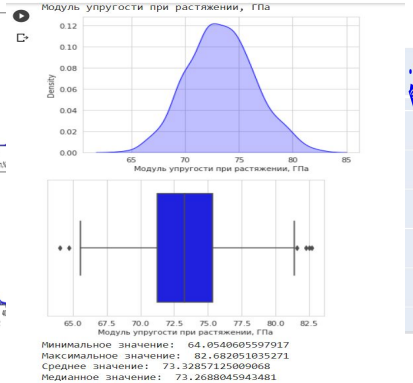
	0	1	2	3	4
Соотношение матрица-наполнитель	1.857143	1.857143	1.857143	1.857143	2.771331
Плотность, кг/м3	2030.000000	2030.000000	2030.000000	2030.000000	2030.000000
модуль упругости, ГПа	738.736842	738.736842	738.736842	738.736842	753.000000
Количество отвердителя, м.%	30.000000	50.000000	49.900000	129.000000	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

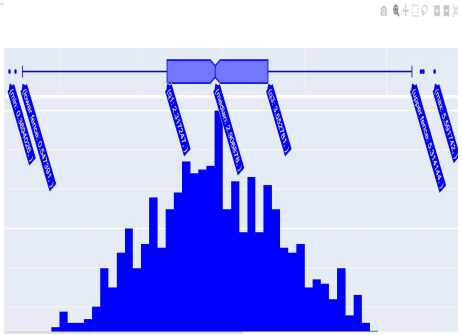
Визуализация исходных данных

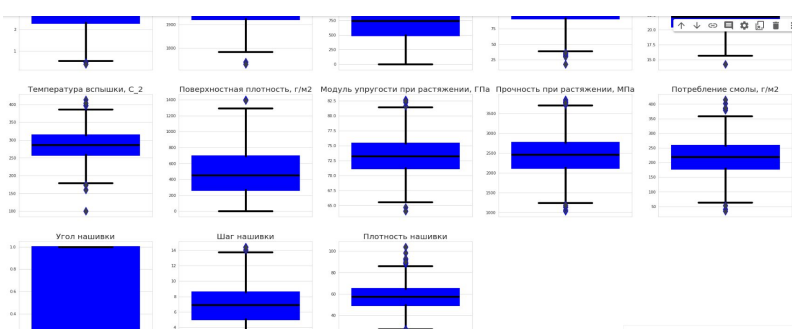
Построим графики различных типов для исходных данных: гистограммы, диаграммы боксплот, попарные графики рассеяния точек, графики квантиль-квантиль, тепловые карты

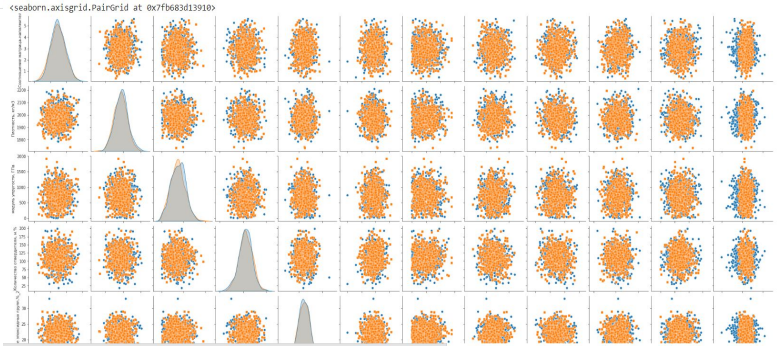












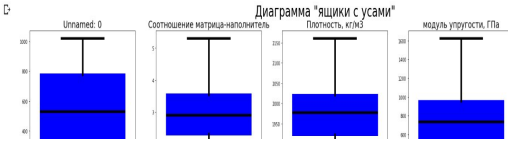
Предобработка данных, исключение выбросов

Исключение выбросов: посчитаем количество значений методом 3 сигм и методом межквартильных интервалов, исключим выбросы методом межквартильного расстояния, построим ящики с усами, проверим выбросы. Удаляем выбросы несколько раз до их окончательного удаления. Получаем датасет без выбросов.

```
#Для удаления выбросов существует 2 основных метода - метод 3-х сигм и межквартильных расстояний. Сравнение этих 2 методов
metod_3s = 0
metod_iq = 0
count_iq = [] # Список, куда записывается количество выбросов по каждой колонке датафрейма методом.
count_3s = [] # Список, куда записывается количество выбросов по каждой колонке датафрейма.
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)
```

```
 13  Плотность нашивки          923 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 108.2 KB
```

```
# Ящики с усами (второй вариант)
a = 5 # количество строк
b = 5 # количество столбцов
c = 1 # инициализация plot counter

plt.figure(figsize = (35,35))
plt.suptitle('Диаграмма "ящики с усами"', y = 0.9 ,
             fontsize = 30)
for col in df.columns:
    plt.subplot(a, b, c)
    #plt.figure(figsize=(7,5))
    sns.boxplot(data = df, y = df[col], fliersize = 15, linewidth = 5, boxprops = dict(facecolor = 'red', color = 'b'), medianprops = dict(
    plt.ylabel(None)
    plt.title(col, size = 20)
    #plt.show()
    c += 1
# "Ящики с усами" показывают отсутствие выбросов
```




```
df.isnull().sum()
```

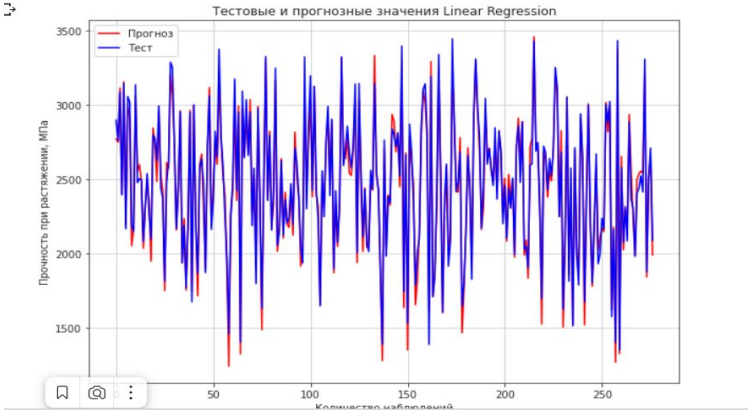
```
Unnamed: 0                             0
Соотношение матрица-наполнитель        0
Плотность, кг/м3                       0
модуль упругости, ГПа                  0
Количество отвердителя, м.%            0
Содержание эпоксидных групп,%_2        0
Температура вспышки, С_2               0
Поверхностная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

```
[ ]  # Построение гистограммы распределения каждой из перемен
```

Разработка и обучение моделей для прогноза прочности при растяжении

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

```
#Линейная регрессия с задачей справилась в 97 % случаев.
```

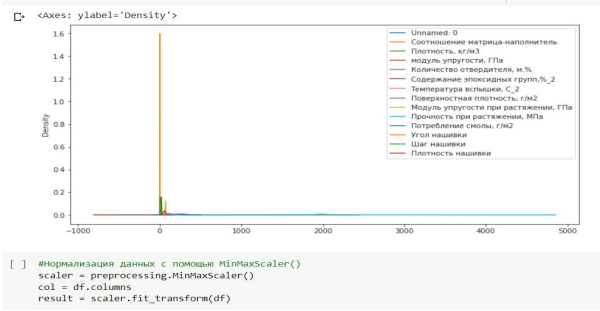



```
#Метод линейной регрессии - Linear Regression - 3

#построение модели и вузуализация Линейной регрессии
lr = LinearRegression()
lr.fit(x_train_1, y_train_1)
y_pred_lr = lr.predict(x_test_1)
mae_lr = mean_absolute_error(y_pred_lr, y_test_1)
mse_lin_elast = mean_squared_error(y_test_1, y_pred_lr)
print('Linear Regression Results Train:') # Скор для тренировочной выборки
print("Test score: {:.2f}".format(lr.score(x_train_1, y_train_1)))
print('Linear Regression Results:')
print('lr_MAE: ', round(mean_absolute_error(y_test_1, y_pred_lr)))
print('lr_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test_1, y_pred_lr)))
print('lr_MSE: {:.2f}'.format(mse_lin_elast))
print("lr_RMSE: {:.2f}".format (np.sqrt(mse_lin_elast)))
print("Test score: {:.2f}".format(lr.score(x_test_1, y_test_1))) # Скор для тестовой выборки
```

```
Linear Regression Results Train:
Test score: 0.97
Linear Regression Results:
lr_MAE:  64
lr_MAPE: 0.03
lr_MSE: 6502.47
lr_RMSE: 80.64
Test score: 0.97
```

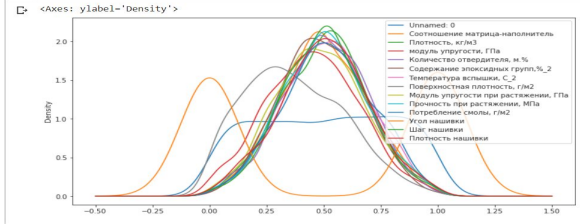
Предобработка данных, нормализация данных, стандартизация данных

Проведем нормализацию данных: MinMaxScaler, построим график плотности ядра, проверим результат MinMaxScaler, построим графики MinMaxScaler, нормализируем данные с помощью Normalazer, проверим результат Normalizer, построим графики Normalizer



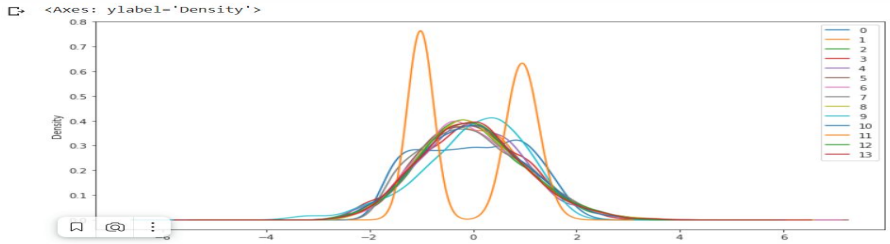

```
[ ]  #Нормализация данных с помощью MinMaxScaler()
     scaler = preprocessing.MinMaxScaler()
     col = df.columns
     result = scaler.fit_transform(df)
```

```
fig, ax = plt.subplots(figsize = (12, 6))
df_minmax_n.plot(kind = 'kde', ax = ax)
```



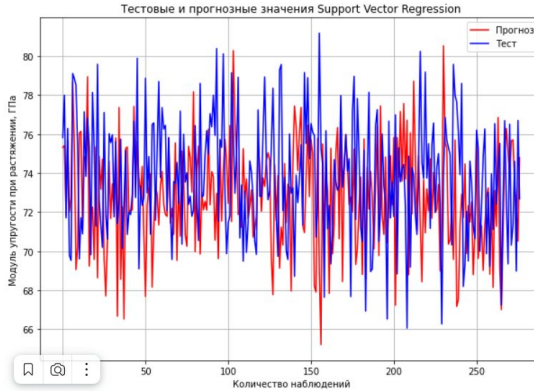
Проведем стандартизацию данных: стандартизируем данные с помощью StandardScaler, проверим результат StandardScaler, построим графики StandardScaler

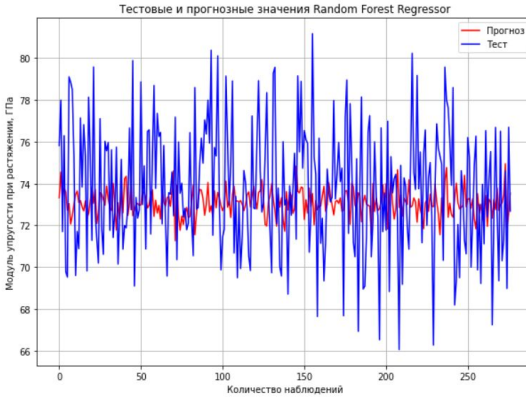
```
fig, ax = plt.subplots(figsize = (12, 6))
df_standart_2.plot(kind = 'kde', ax=ax)
```

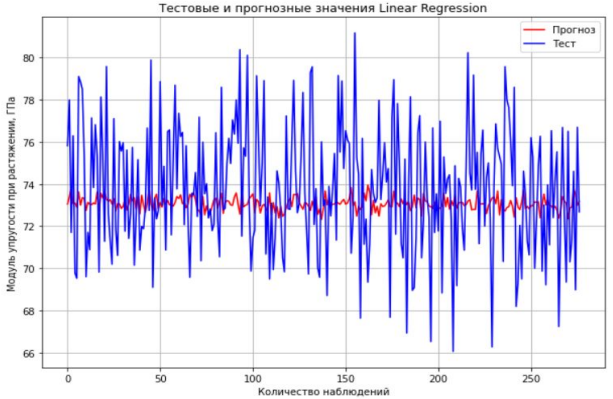


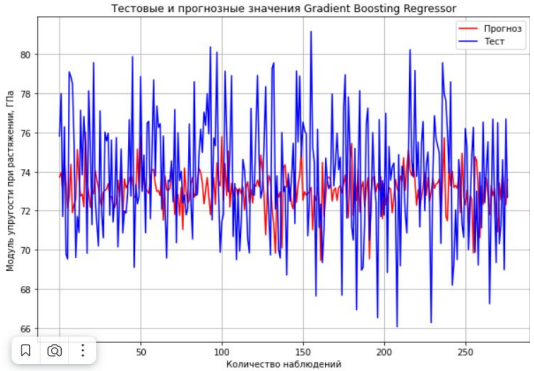
Разработка и обучение моделей для прогноза модуль упругости при растяжении:

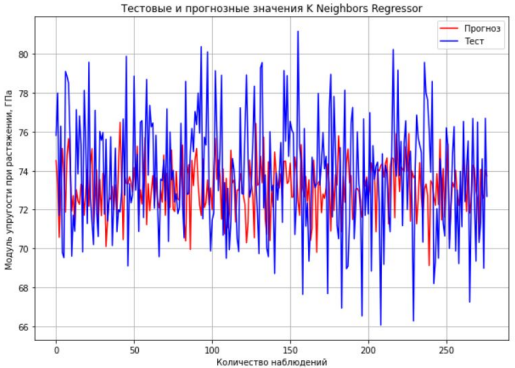
Графики тестовых и прогнозных значений для методов: опорных векторов, случайного леса, линейной регрессии, градиентного бустинга, K-ближайших соседей, дерева принятия решений

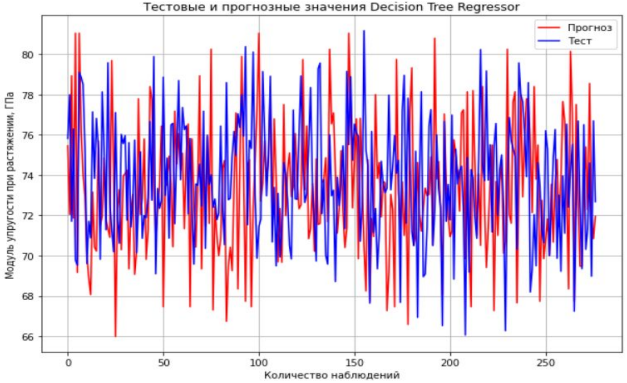












Поиск гиперпараметров для прогноза модуль упругости при растяжении для метода: “Дерево принятия решений”

```
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]},]
```

```
# Проведение поиска  по сетке гиперпараметров с перекрестной проверкой, количество блоков равно 10 (cv = 10), для
#Деревья решений - Decision Tree Regressor - 6
criterion21 = ['squared_error', 'friedman_mse', 'absolute_error', 'poisson']
splitter21 = ['best', 'random']
max_depth21 = [3,5,7,9,11]
min_samples_leaf21 = [100,150,200]
min_samples_split21 = [200,250,300]
max_features21 = ['auto', 'sqrt', 'log2']
param_grid21 = {'criterion': criterion21,
                'splitter': splitter21,
                'max_depth': max_depth21,
                'min_samples_split': min_samples_split21,
                'min_samples_leaf': min_samples_leaf21,
                'max_features': max_features21}
#Запуск 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.
gs21 = GridSearchCV(dtr2, param_grid21, cv = 10, verbose = 1, n_jobs=-1, scoring = 'r2')
gs21.fit(x_train_2, y_train_2)
dtr_21 = gs21.best_estimator_
gs21.best_params_
```

```
Fitting 10 folds for each of 1080 candidates, totalling 10800 fits
{'criterion': 'friedman_mse',
 'max_depth': 11,
 'max_features': 'log2',
 'min_samples_leaf': 100,
 'min_samples_split': 250,
```

	Регрессор	MAE
0	Support Vector	3.295316
1	RandomForest	2.648371
2	Linear Regression	2.535163
3	GradientBoosting	2.660585
4	KNeighbors	2.673409
5	DecisionTree	3.460973
6	SGD	2.591770
7	MLP	3.052449
8	Lasso	2.530635
9	KNeighbors1_GridSearchCV	2.627784
10	DecisionTree1_GridSearchCV	2.588066

Нейронная сеть для соотношения “матрица - наполнитель” (первая модель):

```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
```

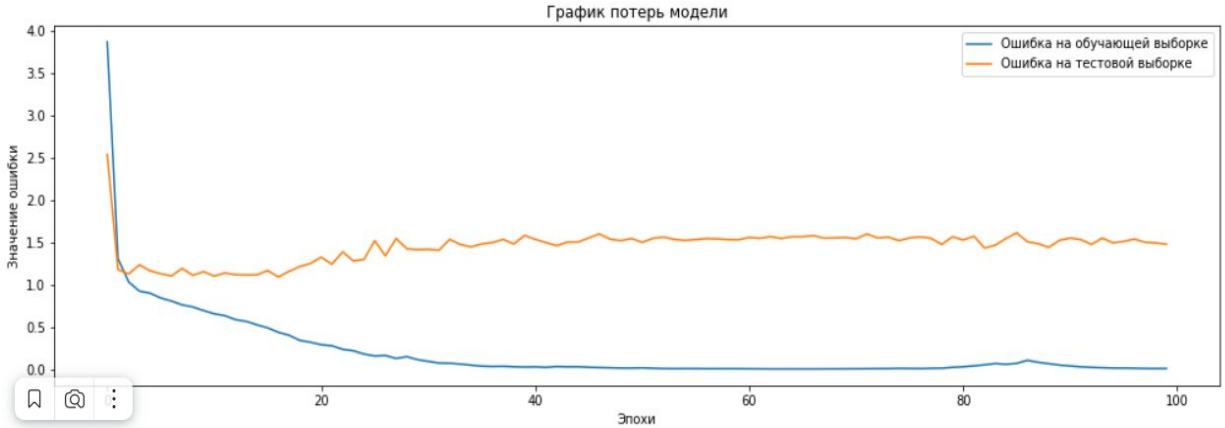
```python
# построение окончательной модели (результаты равны между собой)
model = create_model(lyrs=[12, 6, 3], dr=0.05)

print(model.summary())
```

```
Model: "sequential_135"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_342 (Dense)           (None, 12)                180

 dense_343 (Dense)           (None, 6)                 78

 dense_344 (Dense)           (None, 3)                 21

 dropout_135 (Dropout)       (None, 3)                 0

 dense_345 (Dense)           (None, 3)                 12

=================================================================
Total params: 291
Trainable params: 291
Non-trainable params: 0
_________________________________________________________________
```

Нейронная сеть для соотношения “матрица - наполнитель” (вторая модель)

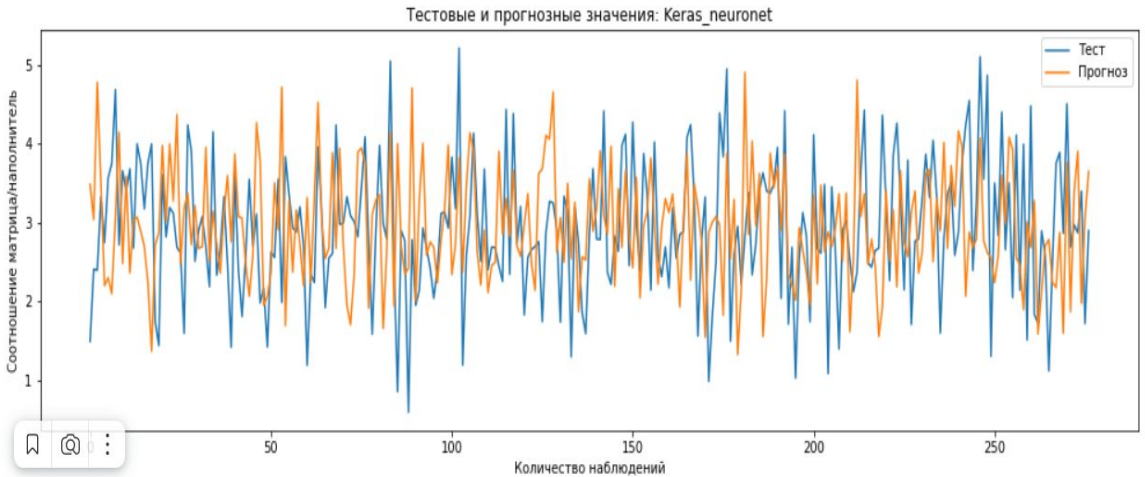
```
# Сконфигурируем другую модель, зададим слои
from tensorflow import keras
from keras import layers
from tensorflow.keras.layers import Dense
model1 = tf.keras.Sequential([x_train_n, layers.Dense(128, activation='relu'),
                                         layers.Dense(128, activation='relu'),
                                         layers.Dense(64, activation='relu'),
                                         layers.Dense(32, activation='relu'),
                                         layers.Dense(16, activation='relu'),
                                         layers.Dense(1)
                              ])

model1.compile(optimizer = tf.keras.optimizers.Adam(0.001), loss = 'mean_squared_error', metrics = [tf.keras.metrics.RootMeanSquaredError()])
# Посмотрим на архитектуру модели

model1.summary()
```

```
Model: "sequential_138"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 14)               29
 n)

 dense_361 (Dense)           (None, 128)               1920

 dense_362 (Dense)           (None, 128)               16512

 dense_363 (Dense)           (None, 64)                8256

 dense_364 (Dense)           (None, 32)                2080

 dense_365 (Dense)           (None, 16)                528

 dense_366 (Dense)           (None, 1)                 17

=================================================================
Total params: 29,342
Trainable params: 29,313
Non-trainable params: 29
_________________________________________________________________
```




```
# Обучение модели

model_hist1 = model1.fit(
    x_train,
    y_train,
    epochs = 100,
    verbose = 1,
    validation_split = 0.2)
```

Нейронная сеть для соотношения “матрица - наполнитель” (вторая модель)




```
# оценка модели MSE
model1.evaluate(x_test, y_test, verbose = 1)
```

```
9/9 [==============================] - 0s 4ms/step - loss: 1.1285 - root_mean_squared_error: 1.0623
[1.1284537315368652, 1.0622869729995728]
```

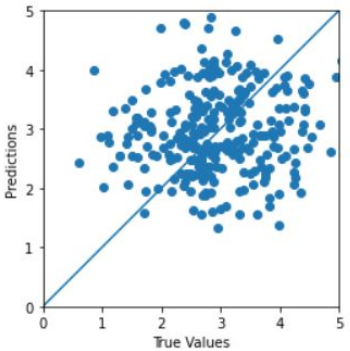
```
test_predictions = model1.predict(x_test).flatten()

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

```
model1.evaluate(x_test, y_test)
```

```
9/9 [==============================] - 0s 4ms/step - loss: 1.1285 - root_mean_squared_error: 1.0623
[1.1284537315368652, 1.0622869729995728]
```

```
y_pred_model = model1.predict(x_test)

print('Model Results:')
print('Model_MAE: ', round(mean_absolute_error(y_test, y_pred_model)))
print('Model_MAPE: {:.2f}'.format(mean_absolute_percentage_error(y_test, y_pred_model)))
print("Test score: {:.2f}".format(mean_squared_error(y_test, y_pred_model)))
```

```
9/9 [==============================] - 0s 3ms/step
Model Results:
Model_MAE:  1
Model_MAPE: 0.35
Test score: 1.13
```



Спасибо за внимание!

Заключение:

- Использованные при разработке моделей подходы не позволили получить какие-либо достоверные прогнозы.
- Примененные модели регрессии не показали высокой эффективности в прогнозировании свойств композитов.
- Невозможно определить из свойств материалов соотношение “матрица - наполнитель”
- Текущим набором задача эффективно не решается