ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по курсу

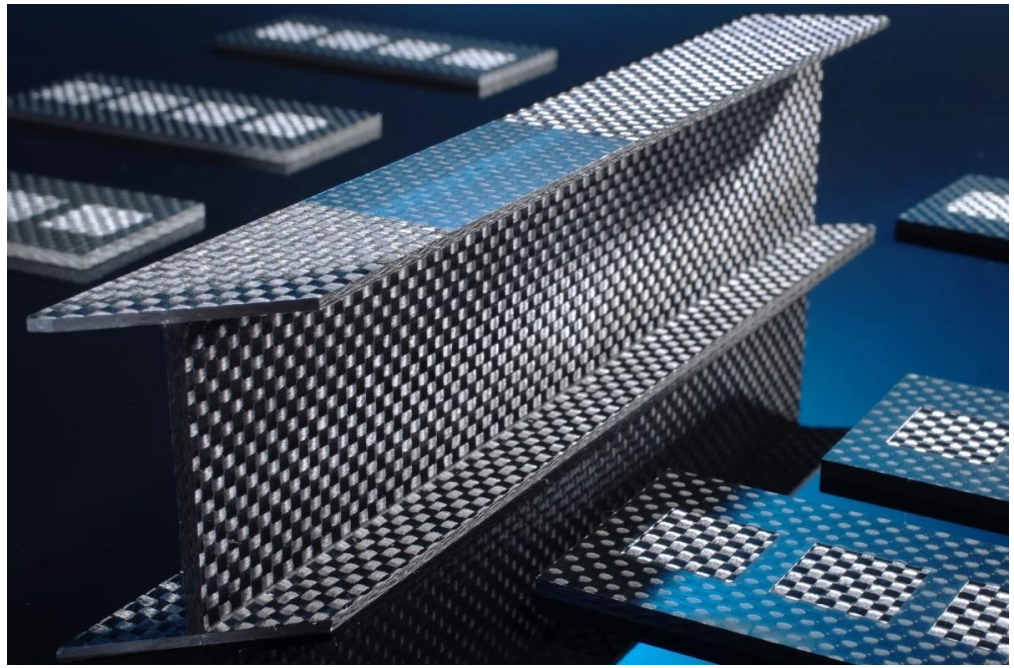
«Data Science»

Слушатель:
Дряев Феликс Аланович

ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

ВВЕДЕНИЕ

Композиционные материалы, представляют собой металлические или неметаллические матрицы (основы) с заданным распределением в них упрочнителей (волокон дисперсных частиц и др.); при этом эффективно используются индивидуальные свойства составляющих композиции.



Важнейшими технологическими методами изготовления композиционных материалов являются: пропитка армирующих волокон матричным материалом; формирование в пресс-форме лент упрочнителя и матрицы, получаемых намоткой; холодное прессование обоих компонентов с последующим спеканием;

ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Постановка задачи

Импортирование нам необходимых библиотек

In [1]:
```
#Импортируем нам необходимые библиотеки!
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import plotly.express as px
import tensorflow as tf
import sklearn
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error, mean_absolute_error
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer, LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from tensorflow import keras as keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, Activation
from pandas import read_excel, DataFrame, Series
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from tensorflow.keras.models import Sequential
from numpy.random import seed
from scipy import stats
import warnings
warnings.filterwarnings("ignore")
```

Объединение файлов и разведочный анализ

In [2]:
```
#Загружаем первый файл эксель
df_bp = pd.read_excel("C:/Users/fdrya/anaconda3/X_bp.xlsx")
df_bp.shape
```

Out[2]: (1023, 11)

In [3]:
```
#Удаляем первый столбец из первого файла
df_bp.drop(['Unnamed: 0'], axis=1, inplace=True)
#Посмотрим на первые 5 строк первого файла
df_bp.head()
```

Out[3]:

	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

In [4]:
```
# Размерность первого файла
df_bp.shape
```

Out[4]: (1023, 10)

In [5]:
```
# Загружаем второй файл
df_nup = pd.read_excel("C:/Users/fdrya/anaconda3/X_nup.xlsx")
df_nup.shape
```

Out[5]: (1040, 4)

In [6]:
```
#Удаляем первый столбец
df_nup.drop(['Unnamed: 0'], axis=1, inplace=True)
#Посмотрим на первые 5 строк второго файла
df_nup.head()
```

Out[6]:

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

In [7]:
```
# Проверим размерность второго файла
df_nup.shape
```

Out[7]: (1040, 3)

In [8]:
```
# Два датасета имеют разный объем строк.
# Надо собрать исходные данные файлы в единый набор данных.
# Объединяем их по типу INNER.
df = df_bp.merge(df_nup, left_index = True, right_index = True, how = 'inner')
df.head().T
```

Out[8]:

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

ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Работа со столбцом "Угол нашивки«

Так как ак как кол-во уникальных значений в колонке Угол нашивки равно 2, можем привести данные в этой колонке к значениям 0 и 1.

```
In [12]: # Поработаем со столбцом "Угол нашивки"

In [13]: df['Угол нашивки, град'].nunique()
Out[13]: 2

In [14]: #Проверим кол-во элементов где равен 0 угол
         df['Угол нашивки, град'][df['Угол нашивки, град'] == 0.0].count()
Out[14]: 520

In [15]: # Приведем Угол нашивки к 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)

In [16]: #Переименуем столбец
         df = df.rename(columns={'Угол нашивки, град' : 'Угол нашивки'})
         df
```

Out[16]:

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки
0	1.857143	2030.000000	738.736842	30.000000	22.267857	100.000000	210.000000	70.000000	3000.000000	220.000000	0
1	1.857143	2030.000000	738.736842	50.000000	23.750000	284.615385	210.000000	70.000000	3000.000000	220.000000	0
2	1.857143	2030.000000	738.736842	49.900000	33.000000	284.615385	210.000000	70.000000	3000.000000	220.000000	0

Проверим где пропущенные данные

```
In [26]: # Проверим где пропущенные данные
         df.isnull().sum()
         # Пропущенных данных нет и нулевых значений нет
         # Очистку проводить не будем
```

Out[26]:

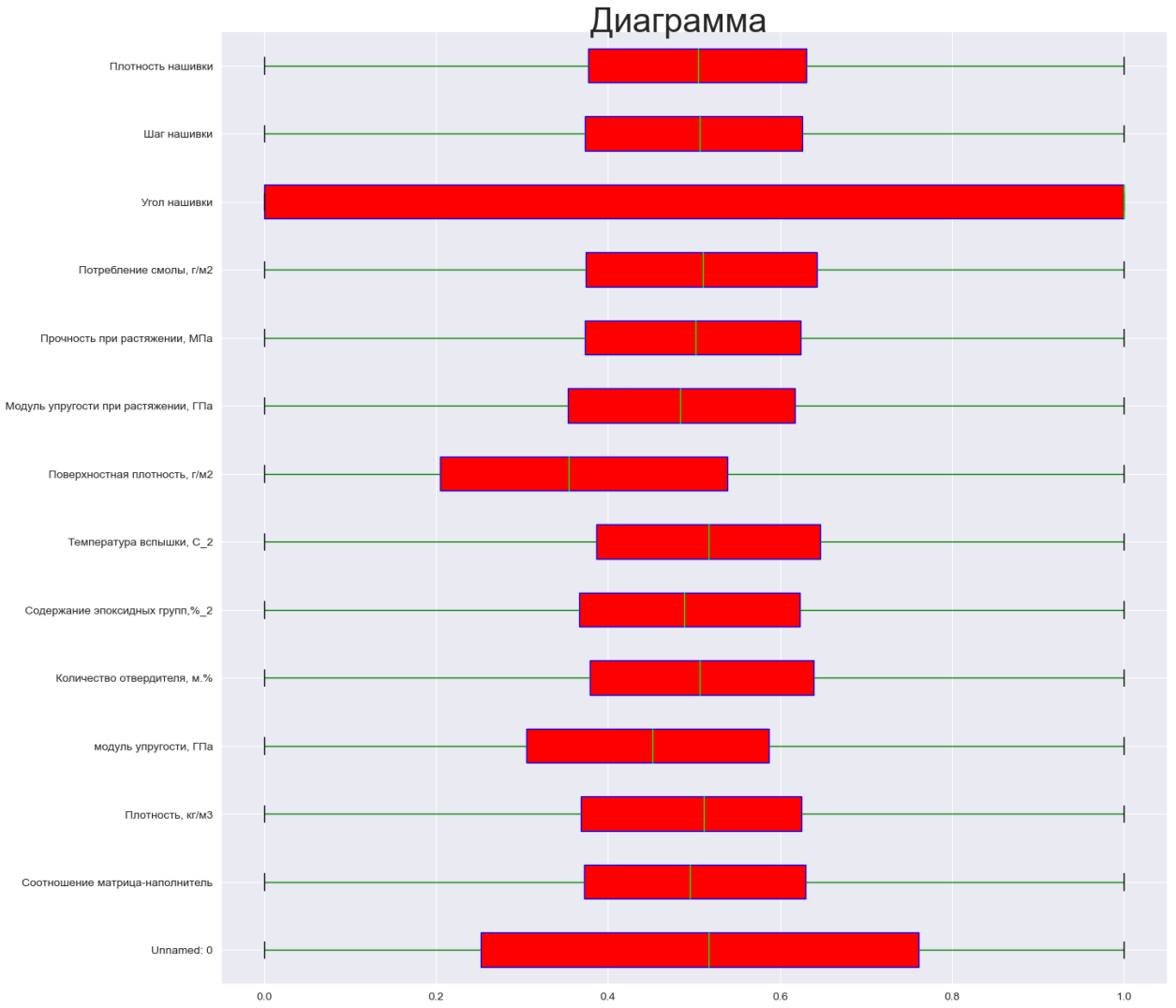
```
Соотношение матрица-наполнитель          0
Плотность, кг/м3                         0
модуль упругости, ГПа                    0
Количество отвердителя, м.%              0
Содержание эпоксидных групп,%_2          0
Температура вспышки, С_2                 0
Поверхностная плотность, г/м2            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
```

ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Выбросы в датасете

Построим первый график с выбросами по-нашему датасету

```
# "Диаграмы" - первый вариант
scaler = MinMaxScaler()
scaler.fit(df)
plt.figure(figsize = (15, 15))
plt.suptitle('Диаграмма', y = 0.9 ,
             fontsize = 30)
plt.boxplot(pd.DataFrame(scaler.transform(df)),
            labels = df.columns,patch_artist = True,
            meanline = True, vert = False,
            boxprops = dict(facecolor = 'r', color = 'b'),
            medianprops = dict(color = 'lime'),
            whiskerprops = dict(color="g"),
            capprops = dict(color = "black"),
            flierprops = dict(color = "y", markeredgecolor = "maroon"))
plt.show()
```

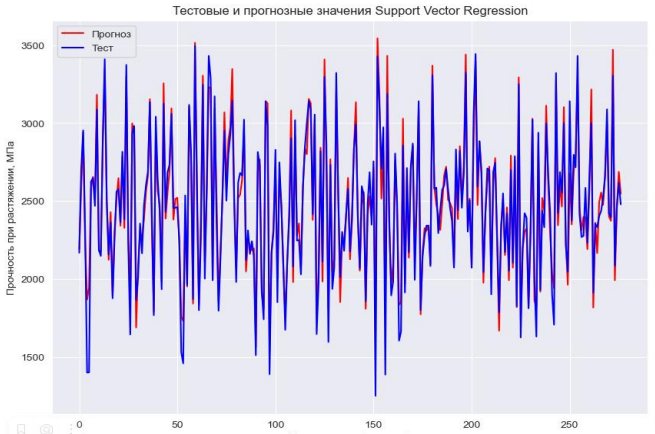


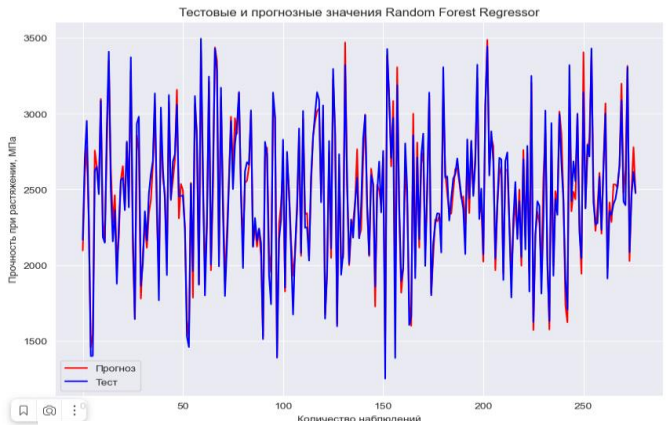
ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

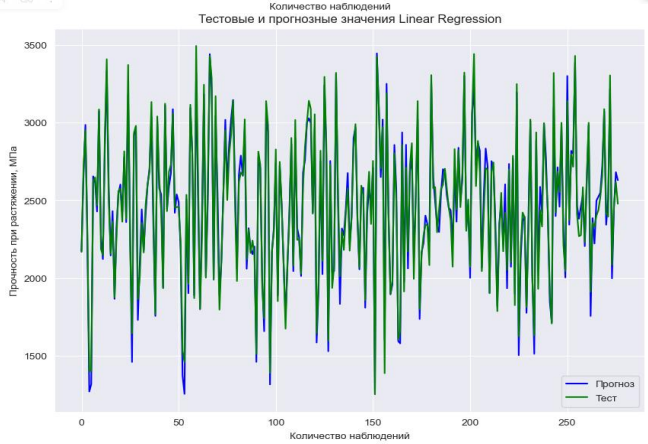
Описание используемых методов в обучении моделей

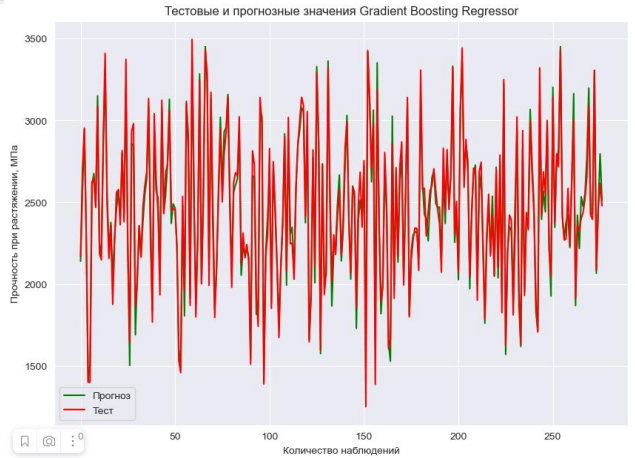
Задача в рамках классификации категорий машинного обучения относится к машинному обучению с учителем и традиционно это задача регрессии. Цель любого алгоритма обучения с учителем — определить функцию потерь и минимизировать её, поэтому для наилучшего решения в процессе исследования были применены следующие методы:

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





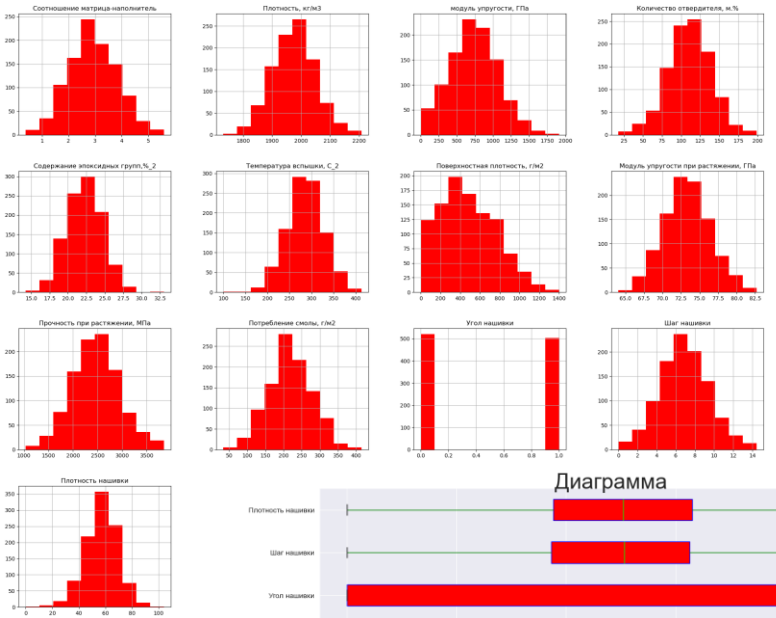


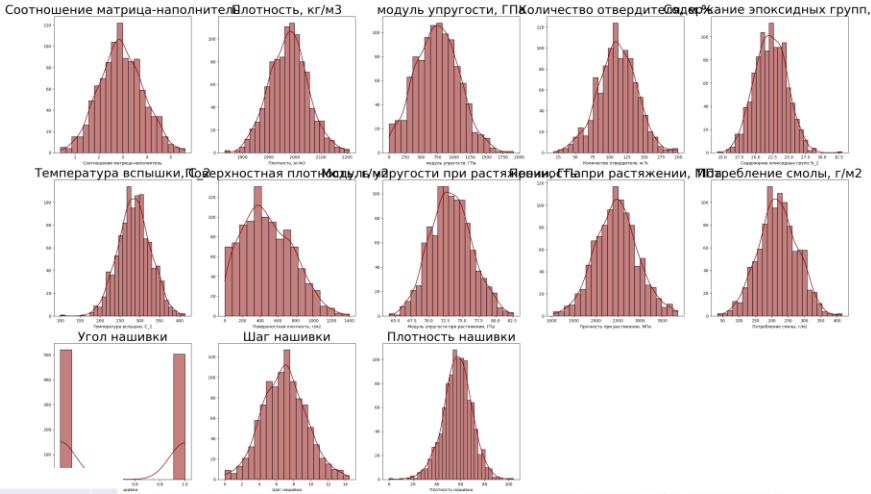


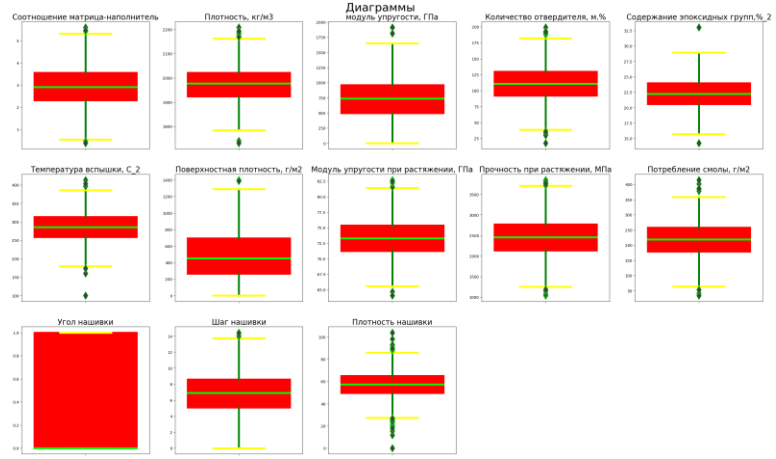
ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

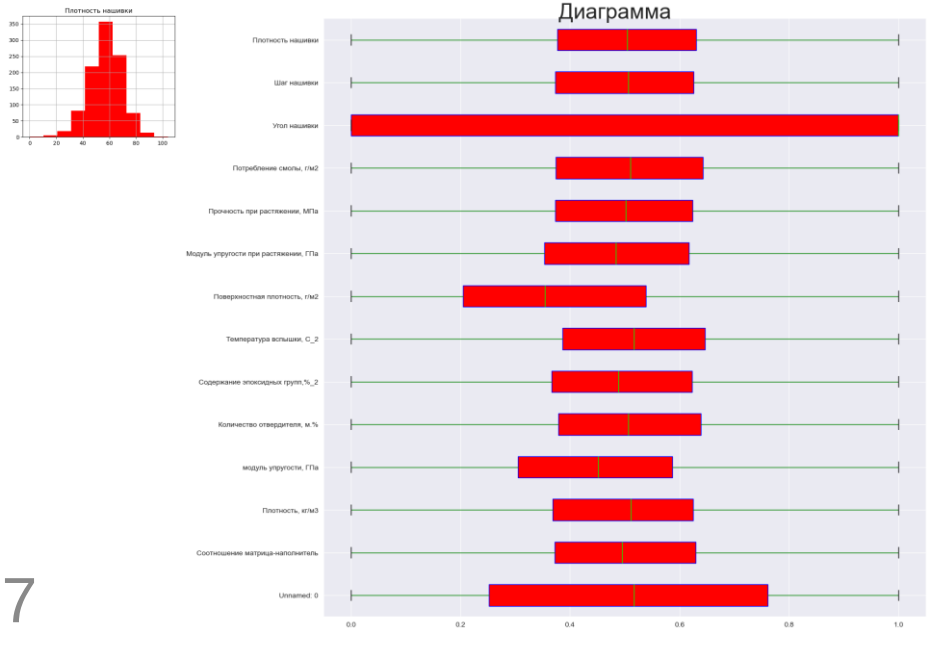
Визуализация имеющихся данных

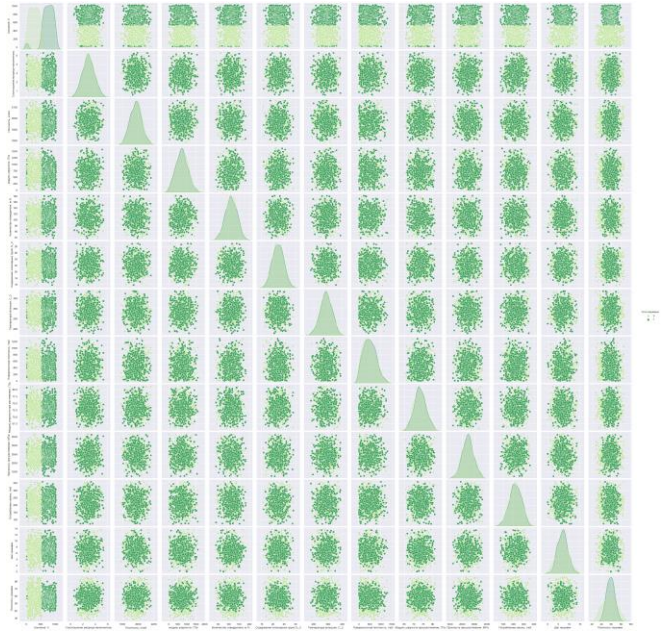
Гистограммы переменных

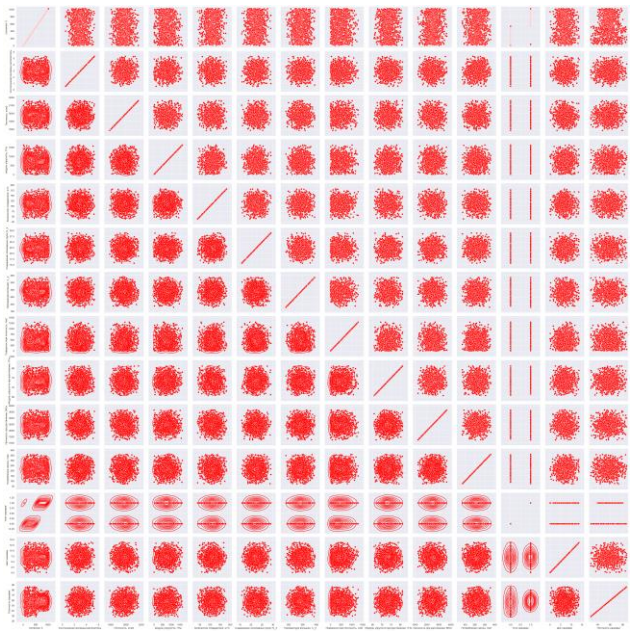










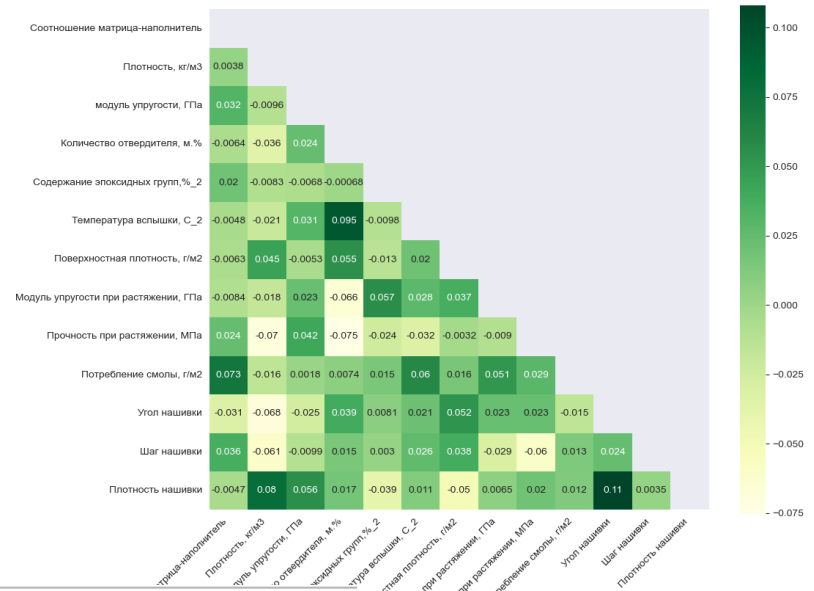


ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Исключение выбросов

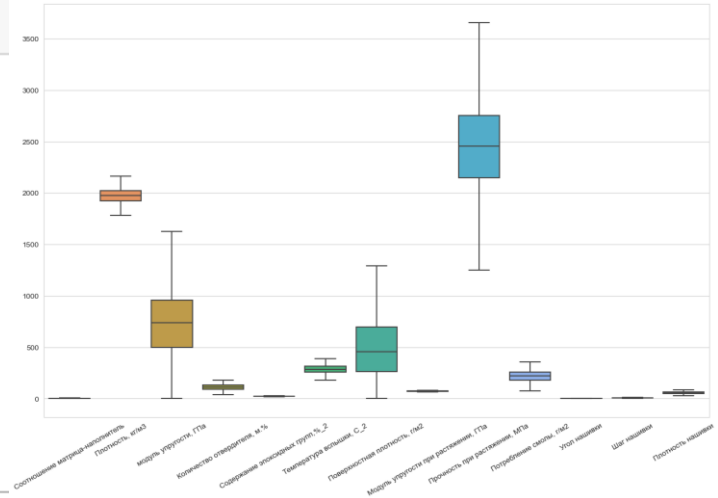
```
In [64]: #Удаления выбросов
metod_3s = 0
metod_iq = 0
count_iq = [] # Список, куда записывается количество выбросов по каждой колонке датафрейма методом.
count_3s = [] # Список, куда записывается количество выбросов по каждой колонке датафрейма.
for column in df:
 d = df.loc[:, [column]]
 # методом 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)
```

```
Соотношение матрица-наполнитель 3s :  0
Соотношение матрица-наполнитель :  6
Плотность, кг/м3 3s :  3
Плотность, кг/м3 :  9
модуль упругости, ГПа 3s :  2
модуль упругости, ГПа :  2
Количество отвердителя, м.% 3s :  2
Количество отвердителя, м.% :  14
Содержание эпоксидных групп,%_2 3s :  2
Содержание эпоксидных групп,%_2 :  2
Температура вспышки, С_2 3s :  3
Температура вспышки, С_2 :  8
Поверхностная плотность, г/м2 3s :  2
Поверхностная плотность, г/м2 :  2
Модуль упругости при растяжении, ГПа 3s :  0
Модуль упругости при растяжении, ГПа :  6
Прочность при растяжении, МПа 3s :  0
Прочность при растяжении, МПа :  11
Потребление смолы, г/м2 3s :  3
Потребление смолы, г/м2 :  8
Угол нашивки 3s :  0
Угол нашивки :  0
Шаг нашивки 3s :  0
Шаг нашивки :  4
Плотность нашивки 3s :  7
Плотность нашивки :  21
Метод 3-х сигм, выбросов: 24
Метод межквартильных расстояний, выбросов: 93
```




```
In [73]: #Сумма выбросов по каждому из столбцов
df.isnull().sum()
#Всего 60 выбросов, их необходимо удалить.
```

```
Out[73]: Соотношение матрица-наполнитель      6
Плотность, кг/м3                      9
модуль упругости, ГПа                 2
Количество отвердителя, м.%          14
Содержание эпоксидных групп,%_2       2
Температура вспышки, С_2              8
Поверхностная плотность, г/м2         2
Модуль упругости при растяжении, ГПа  6
Прочность при растяжении, МПа        11
Потребление смолы, г/м2               8
Угол нашивки                          0
Шаг нашивки                           4
Плотность нашивки                    21
dtype: int64
```



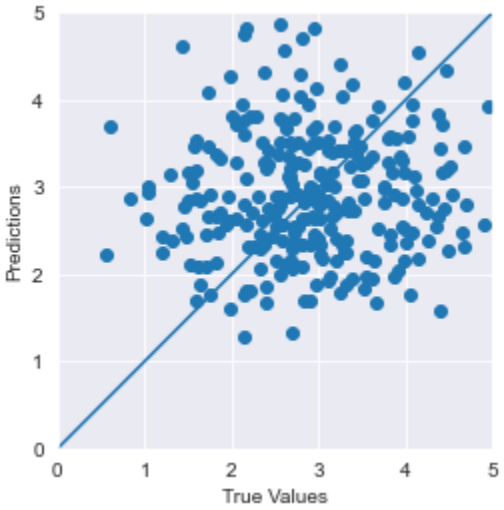
ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Написать нейронную сеть, которая будет рекомендовать соотношение «матрица – наполнитель»

```
# 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.RootMean
# Посмотрим на архитектуру модели

model.summary()
```

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

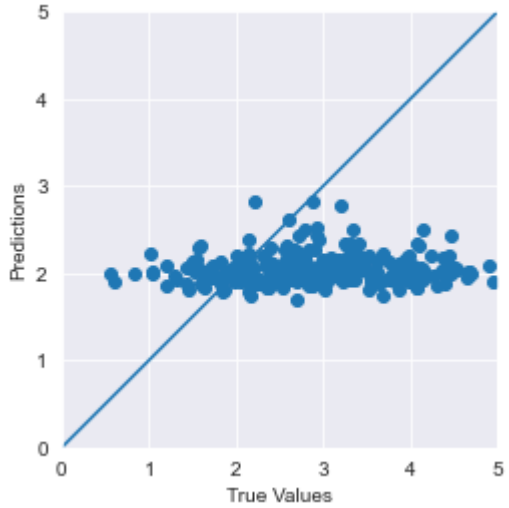
```
9/9 [==============================] - 0s 3ms/step - loss: 1.5056 - root_mean_squared_error
[1.5056190490722656, 1.227036714553833]
```




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

 dense (Dense)               (None, 128)               1664

 dense_1 (Dense)             (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 32)                2080

 dense_5 (Dense)             (None, 16)                528

 dense_6 (Dense)             (None, 1)                 17

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

```
...дель

= model.fit(

= 100,
= 1,
validation_split = 0.3)
```

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



ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Ошибки

- Ошибки такие как: «name '....' is not defined», «Failed to convert a NumPy array to a Tensor (Unsupported object type Normalization)» и т.п.

```
In [53]: # Нормализуем данные
         x_train_n = tf.keras.layers.Normalization(axis =-1)
         x_train_n.adapt(np.array(x_train))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_15664\1606877082.py in <module>
      1 # Нормализуем данные
      2 x_train_n = tf.keras.layers.Normalization(axis =-1)
----> 3 x_train_n.adapt(np.array(x_train))

~\anaconda3\lib\site-packages\keras\layers\preprocessing\normalization.py in adapt(self, data, batch_size, steps)
    284                 argument is not supported with array inputs.
    285         """
--> 286         super().adapt(data, batch_size=batch_size, steps=steps)
    287
    288     def update_state(self, data):

~\anaconda3\lib\site-packages\keras\engine\base_preprocessing_layer.py in adapt(self, data, batch_size, steps)
    244         if self.built:
    245             self.reset_state()
--> 246         data_handler = data_adapter.DataHandler(
    247             data,
    248             batch_size=batch_size,
```

ЦЕНТР
ДОПОЛНИТЕЛЬНОГО
ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

do.bmstu.ru