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

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

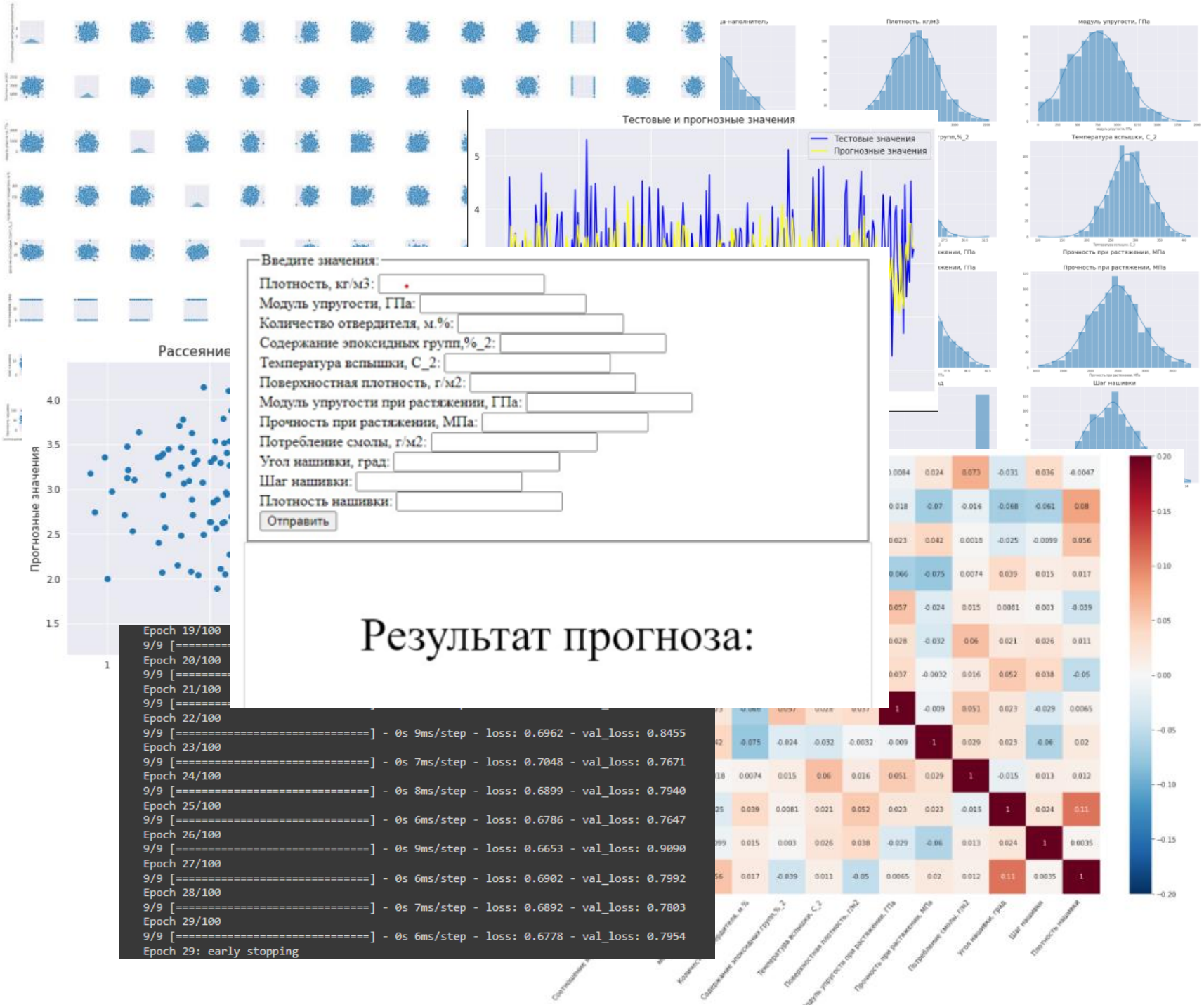
по курсу

«Data Science»

Слушатель: Кулабухов Константин Витальевич
группа 3.1 11785 МГТУ им. Н.Э. Баумана

От и До:

- У нас было 2 датасета с данными о композитах;
- Цель – написать программу для прогнозирования свойств новых композитов;



Начало пути

Заливаем и «моем» датасеты

- визуализируем
- получаем инсайты

```
[3]  data1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1023 entries, 0 to 1022
Data columns (total 11 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Unnamed: 0                          1023 non-null   float64
 1   Соотношение матрица-наполнитель     1023 non-null   float64
 2   Плотность, кг/м3                    1023 non-null   float64
 3   модуль упругости, ГПа               1023 non-null   float64
 4   Количество отвердителя, м.%         1023 non-null   float64
 5   Содержание эпоксидных групп,%_2     1023 non-null   float64
 6   Температура вспышки, С_2            1023 non-null   float64
 7   Поверхностная плотность, г/м2       1023 non-null   float64
 8   Модуль упругости при растяжении, ГПа 1023 non-null  float64
 9   Прочность при растяжении, МПа       1023 non-null   float64
 10  Потребление смолы, г/м2             1023 non-null   float64
dtypes: float64(11)
memory usage: 88.0 KB
```

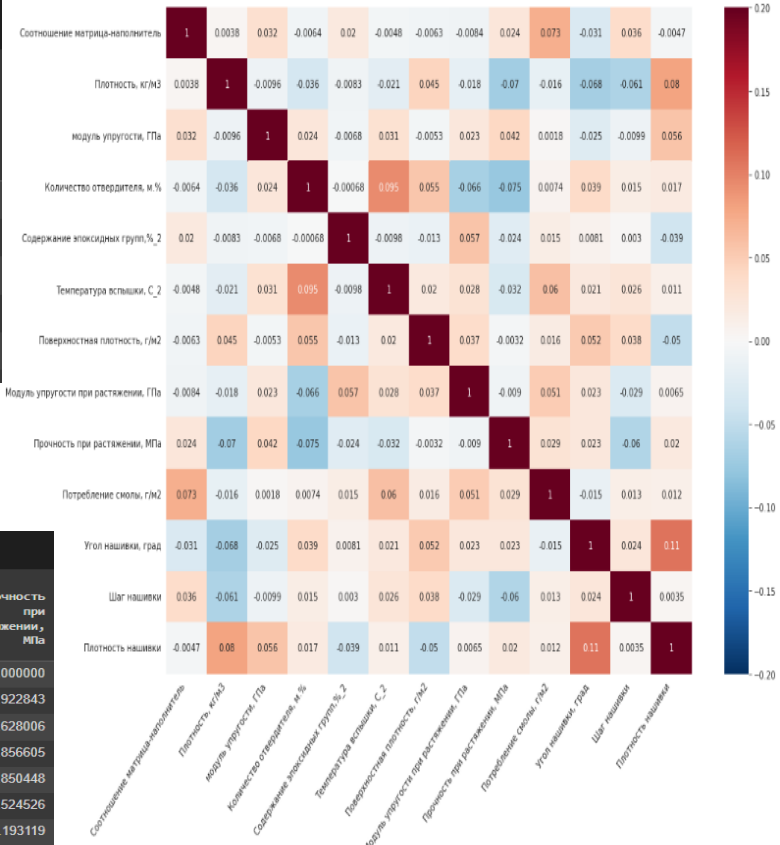
```
# проверяем пропуски
df1.isnull().sum()

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, град                       0
Шаг нашивки                              0
Плотность нашивки                        0
dtype: int64
```

```
data2.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1040 entries, 0 to 1039
Data columns (total 4 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            1040 non-null   float64
 1   Угол нашивки, град    1040 non-null   float64
 2   Шаг нашивки           1040 non-null   float64
 3   Плотность нашивки     1040 non-null   float64
dtypes: float64(4)
memory usage: 32.6 KB
```

```
# объединение датасетов по индексу, тип объединения inner
df = pd.merge(data1, data2, how = 'inner', left_index = True, right_index = True)
df.head()
```

	Unnamed: 0_x	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
0	0.0	1.857143	2030.0	738.736842	30.00	22.267857	100.000000	210.0	70.0	3000.0
1	1.0	1.857143	2030.0	738.736842	50.00	23.750000	284.615385	210.0	70.0	3000.0
2	2.0	1.857143	2030.0	738.736842	49.90	33.000000	284.615385	210.0	70.0	3000.0
3	3.0	1.857143	2030.0	738.736842	129.00	21.250000	300.000000	210.0	70.0	3000.0
4	4.0	2.771331	2030.0	753.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0

```
# проверяем дубли
df.duplicated().sum()

0
```




```
# смотрим описание (для каждой колонки получаем среднее, медианное значение и т.д.)
df1.describe()
```

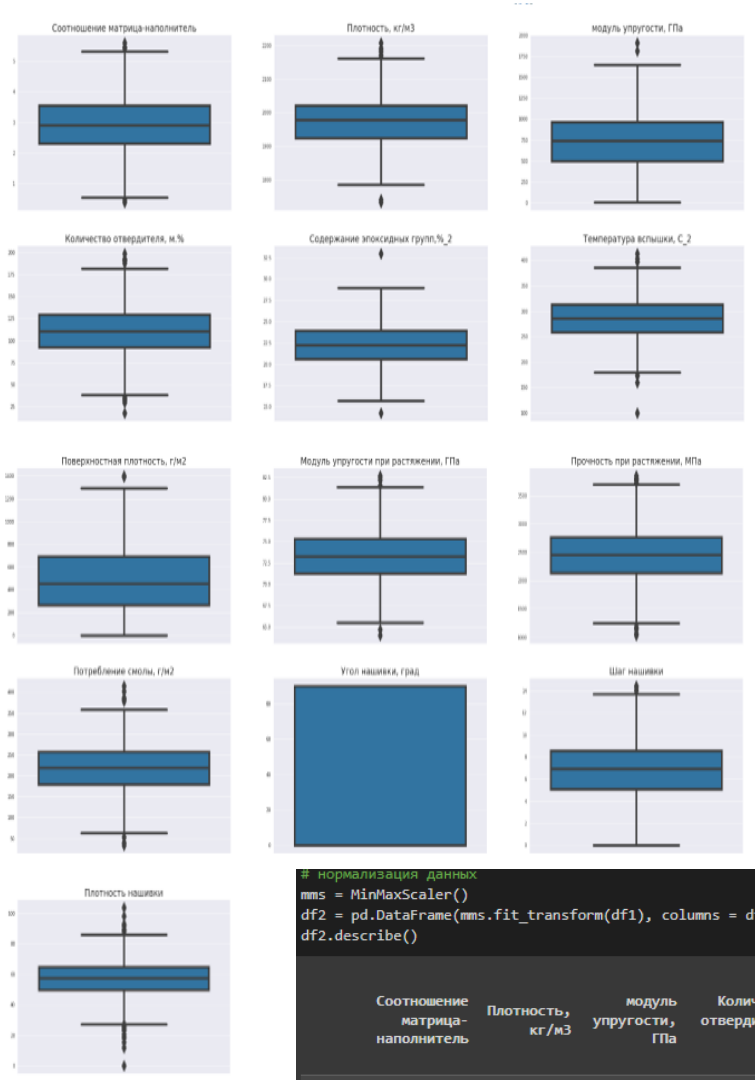
	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа
count	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000
mean	2.930366	1975.734888	739.923233	110.570769	22.244390	285.882151	482.731833	73.328571	2466.922843
std	0.913222	73.729231	330.231581	28.295911	2.406301	40.943260	281.314690	3.118983	485.628006
min	0.389403	1731.764635	2.436909	17.740275	14.254985	100.000000	0.603740	64.054061	1036.856605
25%	2.317887	1924.155467	500.047452	92.443497	20.608034	259.066528	266.816645	71.245018	2135.850448
50%	2.906878	1977.621657	739.664328	110.564840	22.230744	285.896812	451.864365	73.268805	2459.524526
75%	3.552660	2021.374375	961.812526	129.730366	23.961934	313.002106	693.225017	75.356612	2767.193119
max	5.591742	2207.773481	1911.536477	198.953207	33.000000	413.273418	1399.542362	82.682051	3848.436732

Первые шаги

Оцениваем результат

Отсекаем лишнее

Нормализуем данные




```
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, град                   0
Шаг нашивки                          4
Плотность нашивки                   21
dtype: int64
```

```
# дропаем их
df1 = df1.dropna(axis=0)
df1.info()
```

```
# нормализация данных
mms = MinMaxScaler()
df2 = pd.DataFrame(mms.fit_transform(df1), columns = df1.columns, index=df1.index)
df2.describe()
```

	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
count	936.000000	936.000000	936.000000	936.000000	936.000000	936.000000	936.000000	936.000000	936.000000	936.000000	936.000000
mean	0.498933	0.502695	0.446764	0.504664	0.491216	0.516059	0.373733	0.488647	0.495706	0.521141	0.511752
std	0.187489	0.187779	0.199583	0.188865	0.180620	0.190624	0.217078	0.191466	0.188915	0.195781	0.500129
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.372274	0.368517	0.301243	0.376190	0.367716	0.386128	0.205619	0.359024	0.365149	0.392067	0.000000
50%	0.494538	0.511229	0.447061	0.506040	0.489382	0.515980	0.354161	0.485754	0.491825	0.523766	1.000000
75%	0.629204	0.624999	0.580446	0.637978	0.623410	0.646450	0.538683	0.615077	0.612874	0.652447	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Обучаем модели

Делим данные на два лагеря

Учим - тестируем

Повторить N раз

Выбираем победителя

- линейная регрессия (Linear regression);
- метод ближайших соседей (kNN - k Nearest Neighbours);
- случайный лес (RandomForest);

GridSearchCV исчерпывающе рассматриваются все комбинации параметров, а RandomizedSearchCV можно выбрать заданное количество кандидатов из пространства параметров с указанным распределением.

	Model	MAPE	R2
1	LR_uprugost	0.453354	-0.005
2	KN_uprugost	0.449671	-0.003
3	RF_uprugost	0.450596	-0.006

	Model	MAPE	R2
1	LR_prochnost	0.473412	-0.047
2	KN_prochnost	0.472405	-0.028
3	RF_prochnost	0.475677	-0.034

Создаем нейросеть

```
# создаем обучающую и тестовую выборку
x_mn = df1.drop(['Соотношение матрица-наполнитель'], axis=1)
y_mn = df1[['Соотношение матрица-наполнитель']]

X_train_mn, X_test_mn, y_train_mn, y_test_mn = train_test_split(x_mn, y_mn, test_size=0.3, random_state=42)
```

```
itogo_mn = model_mn.fit(
    X_train_mn,      # входя
    y_train_mn,
    batch_size = 64,     # в
    epochs=100, # 100 эпох
    verbose=1, # индикатор
    validation_split = 0.2,
    callbacks = [early_mn]
    )
```

```
# формируем слои нейросети
model_mn = Sequential(X_train_mn_norm)   #

model_mn.add(Dense(128)) # добавляем полно
model_mn.add(BatchNormalization()) # норма
model_mn.add(LeakyReLU()) # расширенный ак
model_mn.add(Dense(128, activation='selu'))
model_mn.add(BatchNormalization())
model_mn.add(Dense(64, activation='selu'))
model_mn.add(BatchNormalization())
model_mn.add(Dense(32, activation='selu'))
model_mn.add(BatchNormalization())
model_mn.add(LeakyReLU())
model_mn.add(Dense(16, activation='selu'))
model_mn.add(BatchNormalization())
model_mn.add(Dense(1))
model_mn.add(Activation('selu'))
```

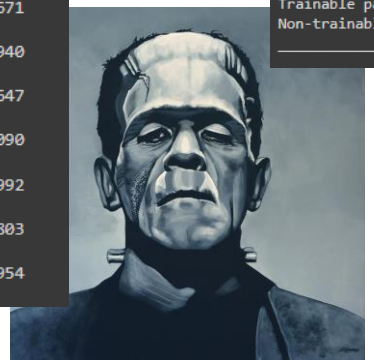
```
model_mn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 128)               1664

batch_normalization_10 (Bat  (None, 128)               512
chNormalization)

leaky_re_lu_4 (LeakyReLU)    (None, 128)               0

dense_13 (Dense)             (None, 128)               16512

batch_normalization_11 (Bat  (None, 128)               512
chNormalization)

dense_14 (Dense)             (None, 64)                8256

batch_normalization_12 (Bat  (None, 64)                256
chNormalization)

dense_15 (Dense)             (None, 32)                2080

batch_normalization_13 (Bat  (None, 32)                128
chNormalization)

leaky_re_lu_5 (LeakyReLU)    (None, 32)                0

dense_16 (Dense)             (None, 16)                528

batch_normalization_14 (Bat  (None, 16)                64
chNormalization)

dense_17 (Dense)             (None, 1)                 17

activation_2 (Activation)    (None, 1)                 0

=================================================================
Total params: 30,529
Trainable params: 29,793
Non-trainable params: 736
_________________________________________________________________
```

```
early_mn = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1
```

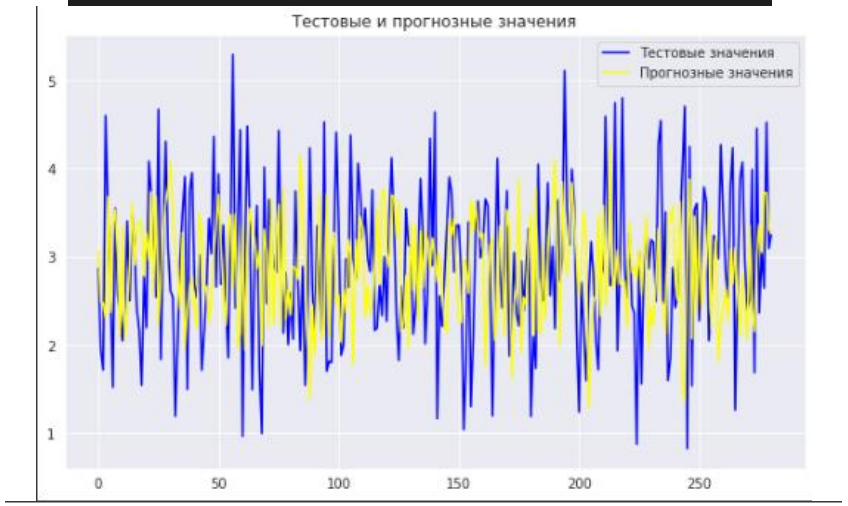
```
model_mn.compile(
    optimizer=tf.optimizers.SGD(learning_rate=0.02, momentum=0.5),
    loss='mean_absolute_error')
```

```
Epoch 19/100
9/9 [==============================] - 0s 5ms/step - loss: 0.7223 - val_loss: 0.7473
Epoch 20/100
9/9 [==============================] - 0s 6ms/step - loss: 0.6894 - val_loss: 0.8947
Epoch 21/100
9/9 [==============================] - 0s 5ms/step - loss: 0.7046 - val_loss: 0.7508
Epoch 22/100
9/9 [==============================] - 0s 9ms/step - loss: 0.6962 - val_loss: 0.8455
Epoch 23/100
9/9 [==============================] - 0s 7ms/step - loss: 0.7048 - val_loss: 0.7671
Epoch 24/100
9/9 [==============================] - 0s 8ms/step - loss: 0.6899 - val_loss: 0.7940
Epoch 25/100
9/9 [==============================] - 0s 6ms/step - loss: 0.6786 - val_loss: 0.7647
Epoch 26/100
9/9 [==============================] - 0s 9ms/step - loss: 0.6653 - val_loss: 0.9090
Epoch 27/100
9/9 [==============================] - 0s 6ms/step - loss: 0.6902 - val_loss: 0.7992
Epoch 28/100
9/9 [==============================] - 0s 7ms/step - loss: 0.6892 - val_loss: 0.7803
Epoch 29/100
9/9 [==============================] - 0s 6ms/step - loss: 0.6778 - val_loss: 0.7954
Epoch 29: early stopping
```



Предсказание

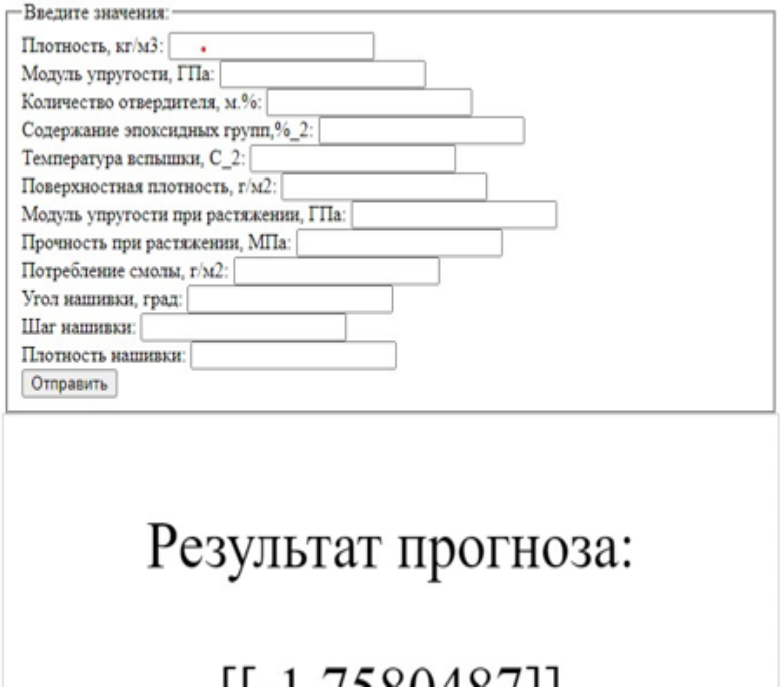
```
pred_mn = model_mn.predict(np.array((X_test_mn)))
original_mn = y_test_mn.values
predicted_mn = pred_mn
```



Завершаем проект

app.py > main

```python
import flask
from flask import render_template
import tensorflow as tf
from tensorflow import keras
import sklearn
import keras

app = flask.Flask(__name__, template_folder='templates')

@app.route('/', methods=['GET', 'POST'])
@app.route('/index', methods=['GET', 'POST'])

def main():
    temp = 1
    param_lst = []

    if flask.request.method == 'GET':
        return render_template('main0.html' )

    if flask.request.method == 'POST':
        loaded_model = keras.models.load_model("model_vkr1")
        for i in range(1,13,1):
            experience = float(flask.request.form.get(f'experience{i}'))
            param_lst.append(experience)

        temp = loaded_model.predict([param_lst])
        return render_template('main0.html', result = temp)

if __name__ == '__main__':
    app.run()
```



edu.bmstu.ru

+7 495 182-83-85

edu@bmstu.ru

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

ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР МГТУ им. Н. Э. Баумана