Импорт Бибилиотек

In [1]:
```
import numpy as np
import pandas as pd
import scipy
```

In [2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import pyplot
```

In [3]:
```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer, LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.metrics import accuracy_score
```

Анализ датасета. Общая оценка содержимого

In [4]:
```
df1 = pd.read_csv(r"D:\Downloads\МГТУ\ВКР\X_bp.csv")
df2 = pd.read_csv(r"D:\Downloads\МГТУ\ВКР\X_nup.csv")
df1.shape,df2.shape
```

Out[4]: ((1023, 10), (1040, 3))

In [5]:
```
# Объединениепо INNER по индексу
df = df1.merge(df2, left_index=True, right_index=True)
df.head()
```

Out[5]:

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

In [6]:
```
df.shape,df.columns
```

Out[6]:
```
((1023, 13),
 Index(['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', 'Шаг нашивки', 'Плотность нашивки'],
       dtype='object'))
```

Подготовка датасета

In [7]:
```
y = df['Соотношение матрица-наполнитель']
X = df.drop('Соотношение матрица-наполнитель', axis = 1)
y.shape,X.shape
```

Out[7]: ((1023,), (1023, 12))

In [8]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,shuffle=True)
```

Построение нейронной сети на Keras

Нейронная сеть — это метод в искусственном интеллекте, который учит компьютеры обрабатывать данные таким же способом, как и человеческий мозг. Это тип процесса машинного обучения, называемый глубоким обучением, который использует взаимосвязанные узлы или нейроны в слоистой структуре, напоминающей человеческий мозг.

In [9]:
```
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, LeakyReLU, Activation, Dropout, LSTM
from keras.callbacks import EarlyStopping, ModelCheckpoint

print(tf.__version__)
```

```
WARNING:tensorflow:From C:\Users\Phoenix\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.loss
es.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy inst
ead.

2.15.0
```

In [10]:
```
normalizer = tf.keras.layers.Normalization(axis=-1)
```

```
WARNING:tensorflow:From C:\Users\Phoenix\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name tf.get_
default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

In [11]:
```
X_train_norm = normalizer.adapt(np.array(X_train))
```

In [12]:
```
model = Sequential(X_train_norm)

model.add(Dense(128))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dense(64))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dense(64))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dense(32))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dense(32))
model.add(BatchNormalization())
model.add(LeakyReLU())
model.add(Dense(1))
model.add(Activation(activation='elu'))
```

In [13]:
```
model.compile(
    optimizer=tf.optimizers.SGD(learning_rate=0.01, momentum=0.9, nesterov=False),
    loss='mean_absolute_error')
```

In [14]:

```
%%time
history = model.fit(
    X_train,
    y_train,
    batch_size = 64,
    epochs=40,
    verbose=1,
    validation_split = 0.2
    )
```

```
Epoch 1/40
WARNING:tensorflow:From C:\Users\Phoenix\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name
tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

9/9 [==============================] - 1s 29ms/step - loss: 2.8209 - val_loss: 5.2234
Epoch 2/40
9/9 [==============================] - 0s 6ms/step - loss: 1.8540 - val_loss: 1.6177
Epoch 3/40
9/9 [==============================] - 0s 5ms/step - loss: 1.3260 - val_loss: 2.8281
Epoch 4/40
9/9 [==============================] - 0s 5ms/step - loss: 0.8647 - val_loss: 1.0158
Epoch 5/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7953 - val_loss: 2.9903
Epoch 6/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7880 - val_loss: 1.7962
Epoch 7/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7724 - val_loss: 1.4744
Epoch 8/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7662 - val_loss: 1.5125
Epoch 9/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7519 - val_loss: 0.8289
Epoch 10/40
9/9 [==============================] - 0s 6ms/step - loss: 0.7174 - val_loss: 1.4532
Epoch 11/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7266 - val_loss: 0.9085
Epoch 12/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7205 - val_loss: 0.8261
Epoch 13/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7194 - val_loss: 0.8907
Epoch 14/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7172 - val_loss: 0.8589
Epoch 15/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6958 - val_loss: 1.1785
Epoch 16/40
9/9 [==============================] - 0s 5ms/step - loss: 0.7002 - val_loss: 1.1723
Epoch 17/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6973 - val_loss: 0.9737
Epoch 18/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6808 - val_loss: 0.9903
Epoch 19/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6891 - val_loss: 0.8615
Epoch 20/40
9/9 [==============================] - 0s 6ms/step - loss: 0.6923 - val_loss: 1.0909
Epoch 21/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6899 - val_loss: 0.9034
Epoch 22/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6835 - val_loss: 0.8161
Epoch 23/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6895 - val_loss: 0.7421
Epoch 24/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6790 - val_loss: 0.8262
Epoch 25/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6738 - val_loss: 0.8404
Epoch 26/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6600 - val_loss: 0.7424
Epoch 27/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6747 - val_loss: 0.7419
Epoch 28/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6958 - val_loss: 0.7261
Epoch 29/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6655 - val_loss: 0.8312
Epoch 30/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6529 - val_loss: 0.7147
Epoch 31/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6510 - val_loss: 1.0434
Epoch 32/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6700 - val_loss: 0.7462
Epoch 33/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6591 - val_loss: 0.8440
Epoch 34/40
9/9 [==============================] - 0s 4ms/step - loss: 0.6632 - val_loss: 0.8473
Epoch 35/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6546 - val_loss: 0.7752
Epoch 36/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6341 - val_loss: 0.7305
Epoch 37/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6452 - val_loss: 0.9022
Epoch 38/40
```

```
9/9 [==============================] - 0s 5ms/step - loss: 0.6776 - val_loss: 0.7316
Epoch 39/40
9/9 [==============================] - 0s 4ms/step - loss: 0.6428 - val_loss: 0.7598
Epoch 40/40
9/9 [==============================] - 0s 5ms/step - loss: 0.6422 - val_loss: 0.7909
CPU times: total: 3.62 s
Wall time: 3.12 s
```

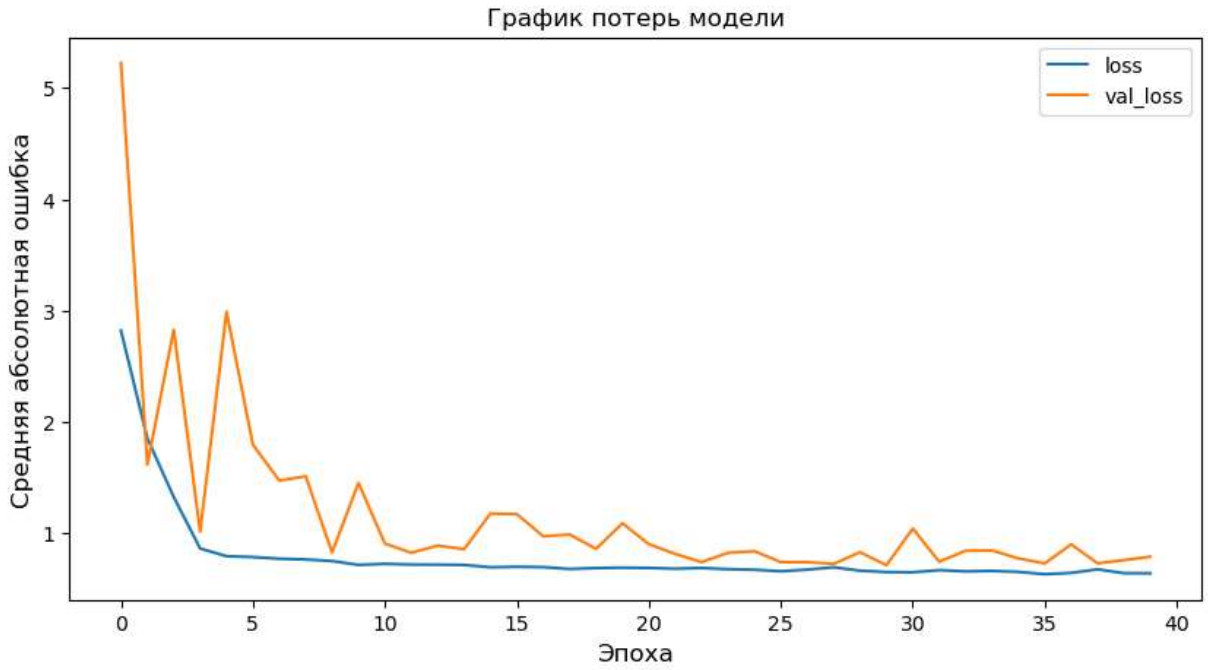
In [16]:

```
1 model.summary()
```

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

 batch_normalization (Batch  (None, 128)               512
 Normalization)

 leaky_re_lu (LeakyReLU)     (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 batch_normalization_1 (Bat  (None, 64)                256
 chNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 64)                0

 dense_2 (Dense)             (None, 64)                4160

 batch_normalization_2 (Bat  (None, 64)                256
 chNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 64)                0

 dense_3 (Dense)             (None, 32)                2080

 batch_normalization_3 (Bat  (None, 32)                128
 chNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 32)                0

 dense_4 (Dense)             (None, 32)                1056

 batch_normalization_4 (Bat  (None, 32)                128
 chNormalization)

 leaky_re_lu_4 (LeakyReLU)   (None, 32)                0

 dense_5 (Dense)             (None, 1)                 33

 activation (Activation)     (None, 1)                 0

=================================================================
Total params: 18529 (72.38 KB)
Trainable params: 17889 (69.88 KB)
Non-trainable params: 640 (2.50 KB)
_________________________________________________________________
```

In [19]:

```
# потери модели на тренировочной и тестовой выборках
def model_loss_plot(model_history):
    plt.figure(figsize=(10, 5))
    plt.plot(model_history.history['loss'])
    plt.plot(model_history.history['val_loss'])
    plt.title('График потерь модели', size=12)
    plt.ylabel('Средняя абсолютная ошибка', size=12)
    plt.xlabel('Эпоха', size=12)
    plt.legend(['loss', 'val_loss'], loc='best')
    plt.show()

model_loss_plot(history)
```

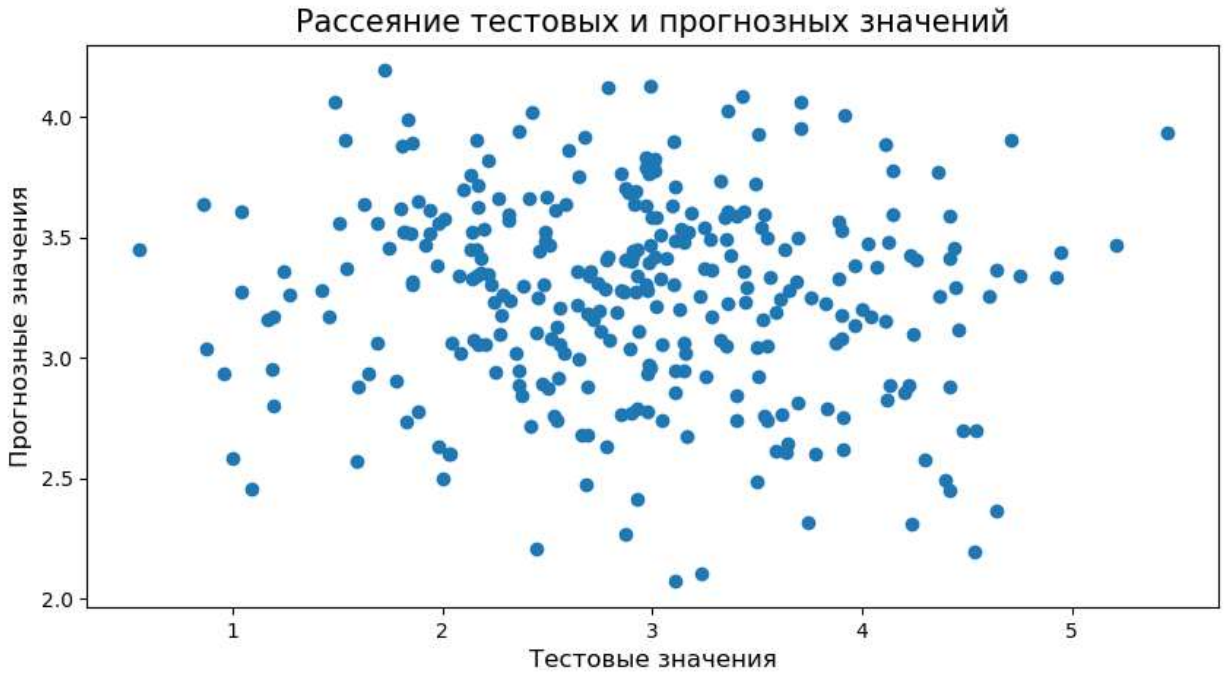



```
# оригинальное и предсказанное значения y
def actual_and_predicted_plot(original_y, predicted_y):
    plt.figure(figsize=(10,5))
    plt.title('Тестовые и прогнозные значения', size=12)
    plt.plot(original_y, color='blue', label = 'Тестовые значения')
    plt.plot(predicted_y, color='red', label = 'Прогнозные значения')
    plt.legend(loc='best')
    plt.show()
```

In [21]:
```
pred = model.predict(np.array((X_test)))
original = y_test.values
predicted = pred

actual_and_predicted_plot(original, predicted)
```

```
10/10 [==============================] - 0s 1ms/step
```

In [22]:
```
# точечный график оригинального и предсказанного значения y
def actual_and_predicted_scatter(original_y, predicted_y):
    plt.figure(figsize=(10,5))
    plt.title('Рассеяние тестовых и прогнозных значений', size=15)
    plt.scatter(original_y, predicted_y)
    plt.xlabel('Тестовые значения', size=12)
    plt.ylabel('Прогнозные значения', size=12)
    plt.show()

actual_and_predicted_scatter(original, predicted)
```



In [25]:
```
print(f'Model MAE: {model.evaluate(X_test, y_test, verbose=1)}')
#print(f'MAE среднего значения: {np.mean(np.abs(y_test-np.mean(y_test)))}')
```

```
10/10 [==============================] - 0s 1ms/step - loss: 0.8766
Model MAE: 0.8765799403190613
MAE среднего значения: 0.7108250025157616
```

In [29]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error, mean_absolute_error
```

In [37]:
```
# оценка модели
mae_train = mean_absolute_error(y_train, model.predict(X_train.values))
mae_test = mean_absolute_error(y_test, model.predict(X_test.values))
mse_train = mean_squared_error(y_train, model.predict(X_train.values))
mse_test = mean_squared_error(y_test, model.predict(X_test.values))
R2_train = r2_score(y_train, model.predict(X_train.values))
R2_test = r2_score(y_test, model.predict(X_test.values))
```

```
23/23 [==============================] - 0s 926us/step
10/10 [==============================] - 0s 1ms/step
23/23 [==============================] - 0s 961us/step
10/10 [==============================] - 0s 1ms/step
23/23 [==============================] - 0s 1ms/step
10/10 [==============================] - 0s 1ms/step
```

In [39]:
```
#сведем результаты работы нейросети по основным исследуемым метрикам в таблицу:
df_fin = {'Модель': ['X_train', 'X_test'], 'MAE': [mae_train, mae_test], 'MSE':[mse_train, mse_test], 'R2':[
df_fin = pd.DataFrame(df_fin)
df_fin
```

Out[39]:

	Модель	MAE	MSE	R2
0	X_train	0.788442	0.969012	-0.145699
1	X_test	0.876580	1.146314	-0.426842

In [ ]:
```

```

Коэффициент детерминации близкий к 1 указывает на то, что модель работает очень хорошо (имеет высокую значимость),

Коэффициент детерминации близкий к 0 означает низкую значимость модели - входная переменная плохо "объясняет" поведение выходной, т.е. линейная зависимость между ними отсутствует. Очевидно, что такая модель будет иметь низкую эффективность.

Отрицательные значения коэффициента детерминации показывает модель "бесполезной" и ее предсказания хуже, чем оценки на основе среднего значения.

Обученные модели не решают поставленных задачпрогнозирования. Неудовлетворительно описывают исходные данные. Построенная и обученная нейронная сеть также не справились с задачей рекомендации соотношения матрица-наполнитель.