# Импорт Бибилиотек

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_va
from sklearn import preprocessing
from sklearn.preprocessing import Normalizer, LabelEncoder, MinMaxScaler, StandardS
```

# Загрузка датасета

In [3]:
```
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[3]:
```
((1023, 10), (1040, 3))
```

In [4]:
```
# Объединениепо INNER по индексу
df = df1.merge(df2, left_index=True, right_index=True)
df.head()
```

Out[4]:

| | Соотношение матрица-наполнитель | Плотность, кг/м3 | модуль упругости, ГПа | Количество отвердителя, м.% | Содержание эпоксидных групп,%_2 | Температура вспышки, С_2 | Поверхност плотност |
|---|---|---|---|---|---|---|---|
| **0** | 1.857143 | 2030.0 | 738.736842 | 30.00 | 22.267857 | 100.000000 | 2 |
| **1** | 1.857143 | 2030.0 | 738.736842 | 50.00 | 23.750000 | 284.615385 | 2 |
| **2** | 1.857143 | 2030.0 | 738.736842 | 49.90 | 33.000000 | 284.615385 | 2 |
| **3** | 1.857143 | 2030.0 | 738.736842 | 129.00 | 21.250000 | 300.000000 | 2 |
| **4** | 2.771331 | 2030.0 | 753.000000 | 111.86 | 22.267857 | 284.615385 | 2 |

In [5]:
```
df.shape
```

Out[5]:
```
(1023, 13)
```

In [6]:
```
df.describe().T.round(2)
```

Out[6]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Соотношение матрица-наполнитель** | 1023.0 | 2.93 | 0.91 | 0.39 | 2.32 | 2.91 | 3.55 | 5.59 |
| **Плотность, кг/м3** | 1023.0 | 1975.73 | 73.73 | 1731.76 | 1924.16 | 1977.62 | 2021.37 | 2207.77 |
| **модуль упругости, ГПа** | 1023.0 | 739.92 | 330.23 | 2.44 | 500.05 | 739.66 | 961.81 | 1911.54 |
| **Количество отвердителя, м.%** | 1023.0 | 110.57 | 28.30 | 17.74 | 92.44 | 110.56 | 129.73 | 198.95 |
| **Содержание эпоксидных групп,%_2** | 1023.0 | 22.24 | 2.41 | 14.25 | 20.61 | 22.23 | 23.96 | 33.00 |
| **Температура вспышки, С_2** | 1023.0 | 285.88 | 40.94 | 100.00 | 259.07 | 285.90 | 313.00 | 413.27 |
| **Поверхностная плотность, г/м2** | 1023.0 | 482.73 | 281.31 | 0.60 | 266.82 | 451.86 | 693.23 | 1399.54 |
| **Модуль упругости при растяжении, ГПа** | 1023.0 | 73.33 | 3.12 | 64.05 | 71.25 | 73.27 | 75.36 | 82.68 |
| **Прочность при растяжении, МПа** | 1023.0 | 2466.92 | 485.63 | 1036.86 | 2135.85 | 2459.52 | 2767.19 | 3848.44 |
| **Потребление смолы, г/м2** | 1023.0 | 218.42 | 59.74 | 33.80 | 179.63 | 219.20 | 257.48 | 414.59 |
| **Угол нашивки, град** | 1023.0 | 44.25 | 45.02 | 0.00 | 0.00 | 0.00 | 90.00 | 90.00 |
| **Шаг нашивки** | 1023.0 | 6.90 | 2.56 | 0.00 | 5.08 | 6.92 | 8.59 | 14.44 |
| **Плотность нашивки** | 1023.0 | 57.15 | 12.35 | 0.00 | 49.80 | 57.34 | 64.94 | 103.99 |

## Нормализация

In [7]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
columns = df.columns
df = scaler.fit_transform(np.array(df))
df = pd.DataFrame(df, columns=columns)
df.head()
```

Out[7]:

| | Соотношение матрица-наполнитель | Плотность, кг/м3 | модуль упругости, ГПа | Количество отвердителя, м.% | Содержание эпоксидных групп,%_2 | Температура вспышки, С_2 | Поверхност плотност |
|---|---|---|---|---|---|---|---|
| **0** | 0.282131 | 0.626533 | 0.385679 | 0.067654 | 0.427467 | 0.000000 | 0.14 |
| **1** | 0.282131 | 0.626533 | 0.385679 | 0.178021 | 0.506535 | 0.589311 | 0.14 |
| **2** | 0.282131 | 0.626533 | 0.385679 | 0.177469 | 1.000000 | 0.589311 | 0.14 |
| **3** | 0.282131 | 0.626533 | 0.385679 | 0.613972 | 0.373167 | 0.638420 | 0.14 |
| **4** | 0.457857 | 0.626533 | 0.393150 | 0.519387 | 0.427467 | 0.589311 | 0.14 |

In [8]:
```
df.describe().T.round(2)
```

Out[8]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Соотношение матрица-наполнитель** | 1023.0 | 0.49 | 0.18 | 0.0 | 0.37 | 0.48 | 0.61 | 1.0 |
| **Плотность, кг/м3** | 1023.0 | 0.51 | 0.15 | 0.0 | 0.40 | 0.52 | 0.61 | 1.0 |
| **модуль упругости, ГПа** | 1023.0 | 0.39 | 0.17 | 0.0 | 0.26 | 0.39 | 0.50 | 1.0 |
| **Количество отвердителя, м.%** | 1023.0 | 0.51 | 0.16 | 0.0 | 0.41 | 0.51 | 0.62 | 1.0 |
| **Содержание эпоксидных групп,%_2** | 1023.0 | 0.43 | 0.13 | 0.0 | 0.34 | 0.43 | 0.52 | 1.0 |
| **Температура вспышки, С_2** | 1023.0 | 0.59 | 0.13 | 0.0 | 0.51 | 0.59 | 0.68 | 1.0 |
| **Поверхностная плотность, г/м2** | 1023.0 | 0.34 | 0.20 | 0.0 | 0.19 | 0.32 | 0.50 | 1.0 |
| **Модуль упругости при растяжении, ГПа** | 1023.0 | 0.50 | 0.17 | 0.0 | 0.39 | 0.49 | 0.61 | 1.0 |
| **Прочность при растяжении, МПа** | 1023.0 | 0.51 | 0.17 | 0.0 | 0.39 | 0.51 | 0.62 | 1.0 |
| **Потребление смолы, г/м2** | 1023.0 | 0.48 | 0.16 | 0.0 | 0.38 | 0.49 | 0.59 | 1.0 |
| **Угол нашивки, град** | 1023.0 | 0.49 | 0.50 | 0.0 | 0.00 | 0.00 | 1.00 | 1.0 |
| **Шаг нашивки** | 1023.0 | 0.48 | 0.18 | 0.0 | 0.35 | 0.48 | 0.59 | 1.0 |
| **Плотность нашивки** | 1023.0 | 0.55 | 0.12 | 0.0 | 0.48 | 0.55 | 0.62 | 1.0 |

## Очистка данных от выбросов

In [9]:
```
for col in df.columns:
    q75,q25 = np.percentile(df.loc[:,col],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    df.loc[df[col] < min,col] = np.nan
    df.loc[df[col] > max,col] = np.nan
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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
```

In [11]:
```
df = df.dropna(axis=0)
```

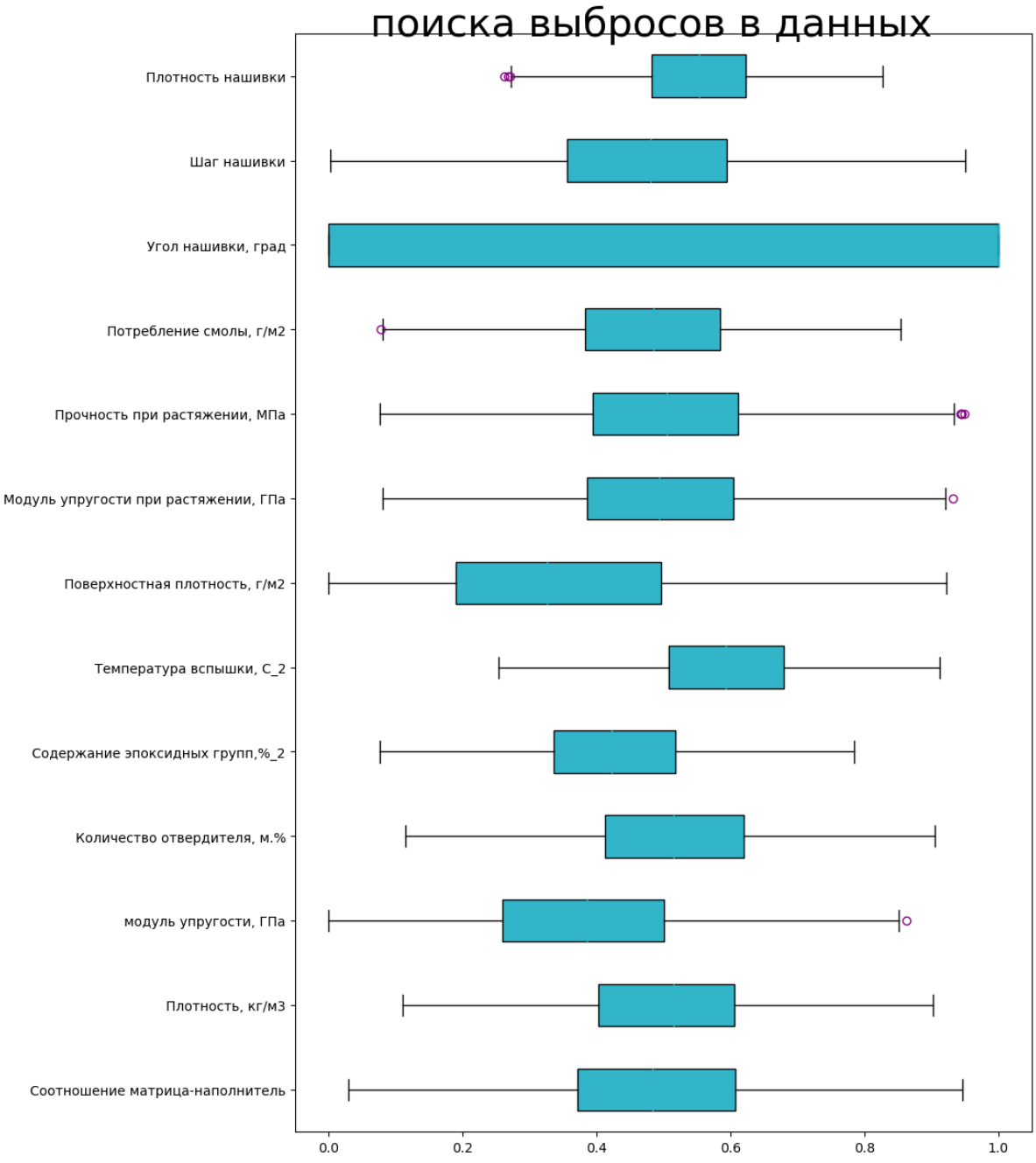
In [12]:
```
df.isnull().sum()
```

Out[12]:
```
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
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 936 entries, 1 to 1022
Data columns (total 13 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Соотношение матрица-наполнитель       936 non-null    float64
 1   Плотность, кг/м3                      936 non-null    float64
 2   модуль упругости, ГПа                 936 non-null    float64
 3   Количество отвердителя, м.%           936 non-null    float64
 4   Содержание эпоксидных групп,%_2       936 non-null    float64
 5   Температура вспышки, С_2              936 non-null    float64
 6   Поверхностная плотность, г/м2         936 non-null    float64
 7   Модуль упругости при растяжении, ГПа  936 non-null    float64
 8   Прочность при растяжении, МПа         936 non-null    float64
 9   Потребление смолы, г/м2               936 non-null    float64
 10  Угол нашивки, град                    936 non-null    float64
 11  Шаг нашивки                           936 non-null    float64
 12  Плотность нашивки                     936 non-null    float64
dtypes: float64(13)
memory usage: 102.4 KB
```

In [14]:
```
plt.figure(figsize = (10, 15))
plt.suptitle('поиска выбросов в данных', y = 0.9, fontsize = 30)
plt.boxplot(pd.DataFrame(df), labels = df.columns,patch_artist = True, meanline = T
plt.show()
```




# Создание и обучение моделей

## Разделение выборки

Разделяем данные на тестовую и тренировочную

Изначально она обучается на отдельных данных (обучающей выборке), а замет проверяем качество работы модели на тестовой выборке. Таким образом, можно измерить способность модели обобщать новые данные. test_size, train_size - необязательные параметры, отвечающие за количество данных в %, которые пойдут в обучающую или тестовую выборки.

random_state - отвечает за перемешивание данных. Туда передают какое-либо целочисленное занчение, чтобы при каждом запуске выборка мешалась одинаково. Это гарантирует воспроизводимость эксперимента.

In [15]:
```
y_el = df['Модуль упругости при растяжении, ГПа']
X_el = df.drop('Модуль упругости при растяжении, ГПа', axis=1)
y_el.shape,X_el.shape
```

Out[15]:
```
((936,), (936, 12))
```

In [16]:
```
X_el_train, X_el_test, y_el_train, y_el_test = train_test_split(X_el, y_el, test_si
                                                                random_state=17, sh
X_el_train.shape, X_el_test.shape
```

Out[16]:
```
((655, 12), (281, 12))
```

In [17]:
```
y_str = df['Прочность при растяжении, МПа']
X_str = df.drop(['Прочность при растяжении, МПа'], axis = 1)
y_str.shape,X_str.shape
```

Out[17]:
```
((936,), (936, 12))
```

In [18]:
```
X_str_train, X_str_test, y_str_train, y_str_test = train_test_split(X_str, y_str, t
                                                                    random_state=17
X_str_train.shape, X_str_test.shape
```

Out[18]:
```
((655, 12), (281, 12))
```

In [19]:
```
models = pd.DataFrame()
```

In [20]:
```
# график сравнения оригинального и предсказанного значения
def plot_act_pr(original_y, predicted_y):
    plt.figure(figsize=(15,7))
    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()
```

## Метод К ближайших соседей

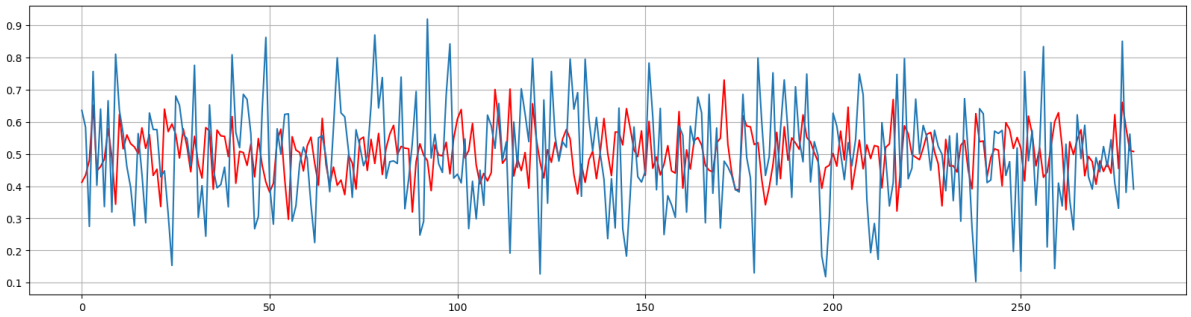
Метод работает с помощью поиска кратчайшей дистанции между тестируемым объектом и ближайшими к нему классифицированным объектами из обучающего набора. Классифицируемый объект будет относится к тому классу, к которому принадлежит ближайший объект набора.

In [21]:
```
from sklearn.neighbors import KNeighborsRegressor
```

In [22]:
```
knn = KNeighborsRegressor().fit(X_el_train, y_el_train)
y_el_pred_knn = knn.predict(X_el_test)
```

In [23]:
```
# оценка качества модели
knr_result = pd.DataFrame({
   'Model': 'KNeighborsRegressor',
   'MSE': mean_squared_error(y_el_test, y_el_pred_knn),
   'MAE': mean_absolute_error(y_el_test, y_el_pred_knn),
   'R2 score': r2_score(y_el_test, y_el_pred_knn).round(3)
}, index=['Модуль упругости при растяжении'])
models = pd.concat([models, knr_result])
```

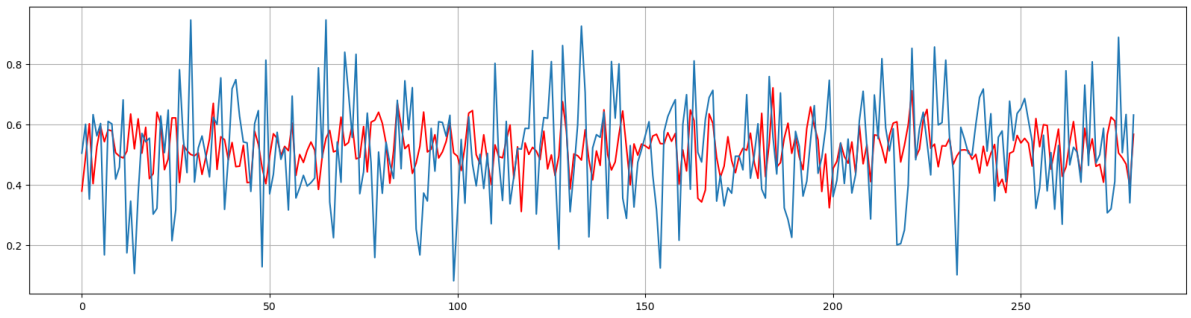
In [24]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_el_pred_knn, "r", label="prediction")
plt.plot(y_el_test.values, label="actual")
plt.grid(True);
```



In [25]:
```
knn = KNeighborsRegressor().fit(X_str_train, y_str_train)
y_str_pred_knn = knn.predict(X_str_test)
```

In [26]:
```
# оценка качества модели
knr_result = pd.DataFrame({
    'Model': 'KNeighborsRegressor',
    'MSE': mean_squared_error(y_str_test, y_str_pred_knn),
    'MAE': mean_absolute_error(y_str_test, y_str_pred_knn),
    'R2 score': r2_score(y_str_test, y_str_pred_knn).round(3)
}, index=['Прочность при растяжении, МПа'])
models = pd.concat([models, knr_result])
```

In [27]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_str_pred_knn, "r", label="prediction")
plt.plot(y_str_test.values, label="actual")
plt.grid(True);
```



## Линейная регрессия

модель зависимости переменной X от одной или нескольких других переменных (факторов, регрессоров, независимых переменных) с линейной функцией зависимости.

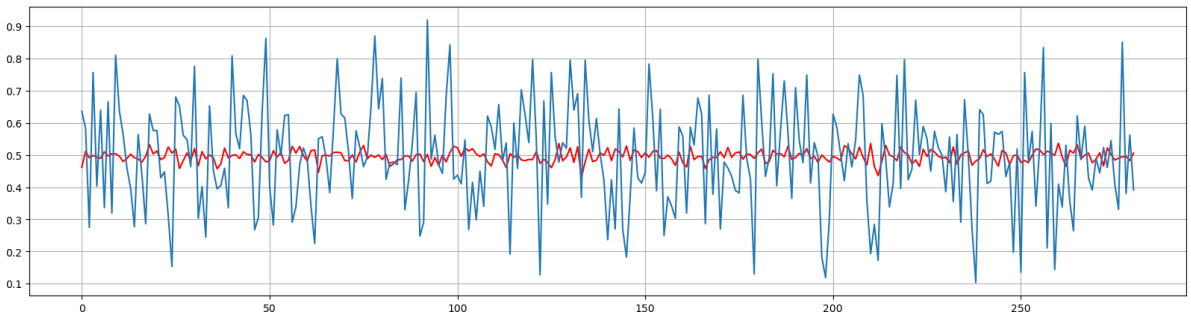
In [28]:
```
from sklearn.linear_model import LinearRegression
```

In [29]:
```
lr = LinearRegression().fit(X_el_train, y_el_train)
y_el_pred_lr = lr.predict(X_el_test)
```

In [30]:
```
# оценка качества модели
lr_result = pd.DataFrame({
    'Model': 'LinearRegression',
    'MSE': mean_squared_error(y_el_test, y_el_pred_lr),
    'MAE': mean_absolute_error(y_el_test, y_el_pred_lr),
    'R2 score': r2_score(y_el_test, y_el_pred_lr).round(3)
```

```
}, index=['Модуль упругости при растяжении'])
models = pd.concat([models, lr_result])
```

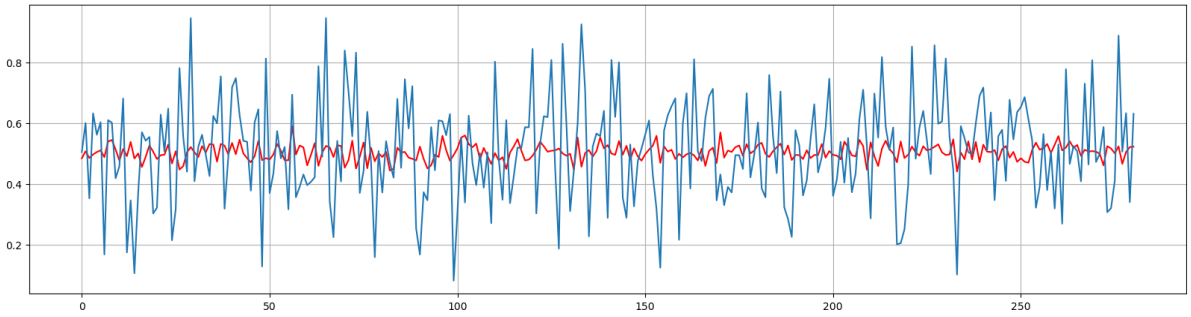
In [31]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_el_pred_lr, "r", label="prediction")
plt.plot(y_el_test.values, label="actual")
plt.grid(True);
```



In [32]:
```
lr = LinearRegression().fit(X_str_train, y_str_train)
y_str_pred_lr = lr.predict(X_str_test)
```

In [33]:
```
# оценка качества модели
lr_result = pd.DataFrame({
   'Model': 'LinearRegression',
   'MSE': mean_squared_error(y_str_test, y_str_pred_lr),
   'MAE': mean_absolute_error(y_str_test, y_str_pred_lr),
   'R2 score': r2_score(y_str_test, y_str_pred_lr).round(3)
}, index=['Прочность при растяжении, МПа'])
models = pd.concat([models, lr_result])
```

In [34]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_str_pred_lr, "r", label="prediction")
plt.plot(y_str_test.values, label="actual")
plt.grid(True);
```



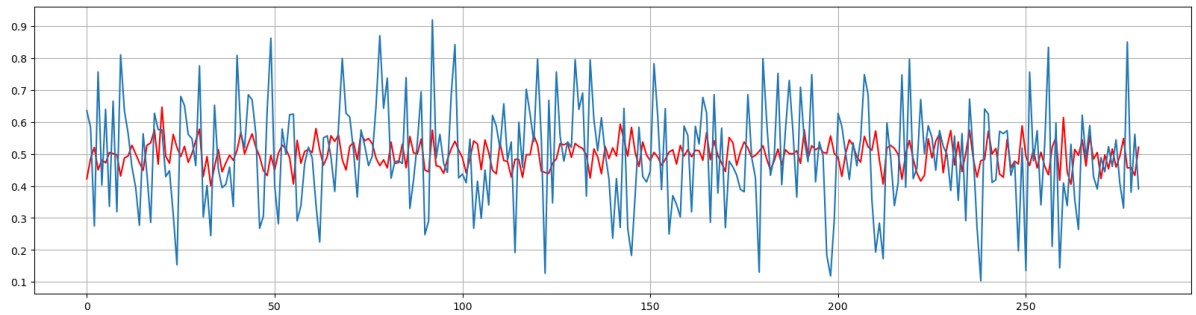
## Многослойный перцептрон

In [35]:
```
from sklearn.neural_network import MLPRegressor
```

In [36]:
```
mlp = MLPRegressor(random_state = 1, max_iter = 500).fit(X_el_train, y_el_train)
y_el_pred_mlp = mlp.predict(X_el_test)
```

In [37]:
```
# оценка качества модели
mlp_result = pd.DataFrame({
    'Model': 'MLPRegressor',
    'MSE': mean_squared_error(y_el_test, y_el_pred_mlp),
    'MAE': mean_absolute_error(y_el_test, y_el_pred_mlp),
    'R2 score': r2_score(y_el_test, y_el_pred_mlp).round(3)
}, index=['Модуль упругости при растяжении'])
models = pd.concat([models, mlp_result])
```

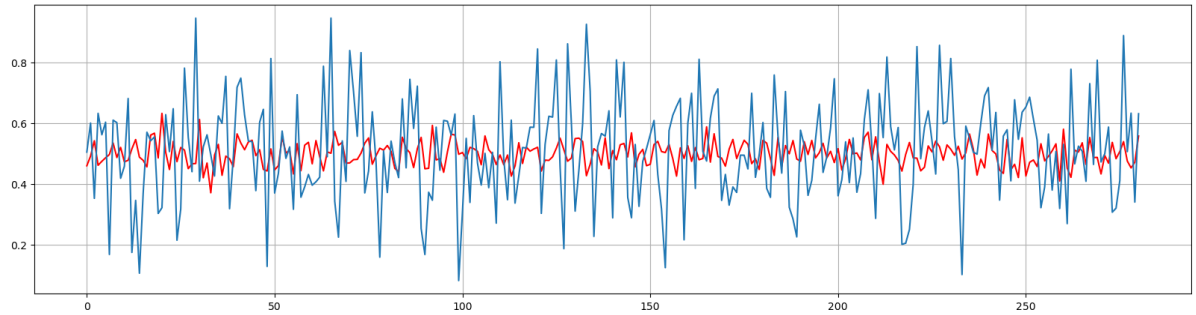
In [38]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_el_pred_mlp, "r", label="prediction")
plt.plot(y_el_test.values, label="actual")
plt.grid(True);
```




In [39]:
```
mlp = MLPRegressor(random_state = 1, max_iter = 500).fit(X_str_train, y_str_train)
y_str_pred_mlp = mlp.predict(X_str_test)
```

In [40]:
```
# оценка качества модели
mpl_result = pd.DataFrame({
    'Model': 'MLPRegressor',
    'MSE': mean_squared_error(y_str_test, y_str_pred_mlp),
    'MAE': mean_absolute_error(y_str_test, y_str_pred_mlp),
    'R2 score': r2_score(y_str_test, y_str_pred_mlp).round(3)
}, index=['Прочность при растяжении, МПа'])
models = pd.concat([models, mlp_result])
```

In [41]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_str_pred_mlp, "r", label="prediction")
plt.plot(y_str_test.values, label="actual")
plt.grid(True);
```




## Дерево решений (Decision Trees)

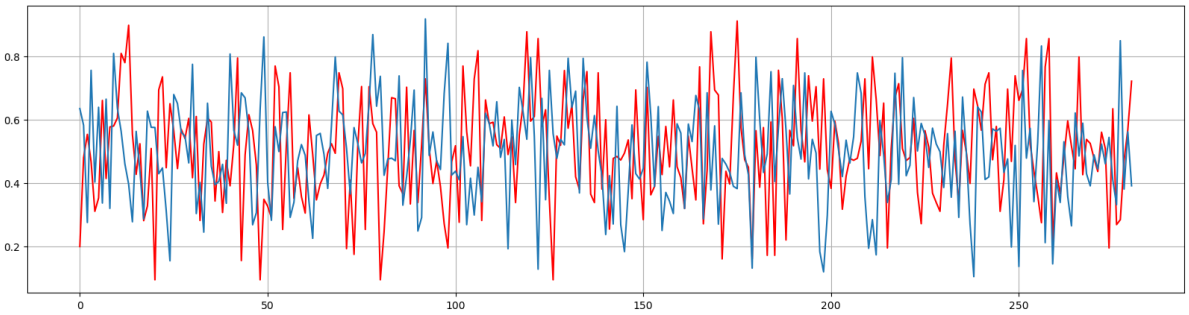
В основе работы дерева решений лежит процесс рекурсивного разбиения исходного множества объектов на подмножества, ассоциированные с предварительно заданными классами. Разбиение производится с помощью решающих правил, в которых осуществляется проверка значений атрибутов по заданному условию.

In [42]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [43]:
```
dcs = DecisionTreeRegressor().fit(X_el_train, y_el_train)
y_el_pred_dcs = dcs.predict(X_el_test)
```

In [44]:
```
# оценка качества модели
dcs_result = pd.DataFrame({
    'Model': 'DecisionTreeRegressor',
    'MSE': mean_squared_error(y_el_test, y_el_pred_dcs),
    'MAE': mean_absolute_error(y_el_test, y_el_pred_dcs),
    'R2 score': r2_score(y_el_test, y_el_pred_dcs).round(3)
}, index=['Модуль упругости при растяжении'])
models = pd.concat([models, dcs_result])
```

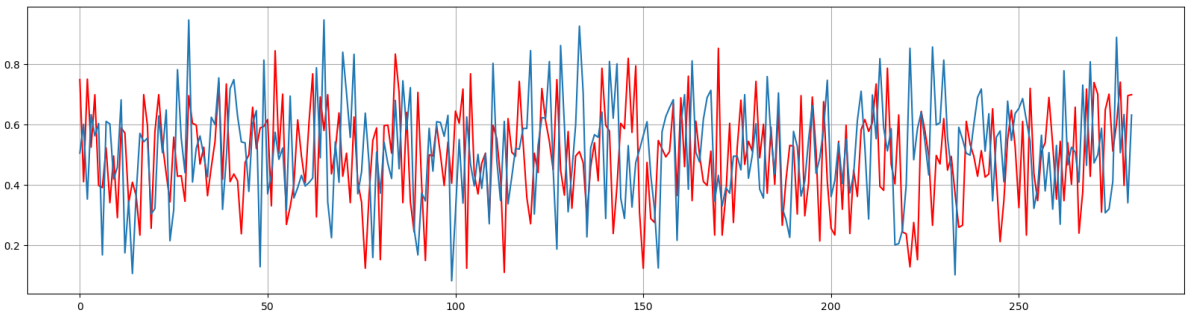
In [45]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_el_pred_dcs, "r", label="prediction")
plt.plot(y_el_test.values, label="actual")
plt.grid(True);
```




In [46]:
```
dcs = DecisionTreeRegressor().fit(X_str_train, y_str_train)
y_str_pred_dcs = dcs.predict(X_str_test)
```

In [47]:
```
# оценка качества модели
dcs_result = pd.DataFrame({
    'Model': 'DecisionTreeRegressor',
    'MSE': mean_squared_error(y_str_test, y_str_pred_dcs),
    'MAE': mean_absolute_error(y_str_test, y_str_pred_dcs),
    'R2 score': r2_score(y_str_test, y_str_pred_dcs).round(3)
}, index=['Прочность при растяжении, МПа'])
models = pd.concat([models, dcs_result])
```

In [48]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_str_pred_dcs, "r", label="prediction")
plt.plot(y_str_test.values, label="actual")
plt.grid(True);
```



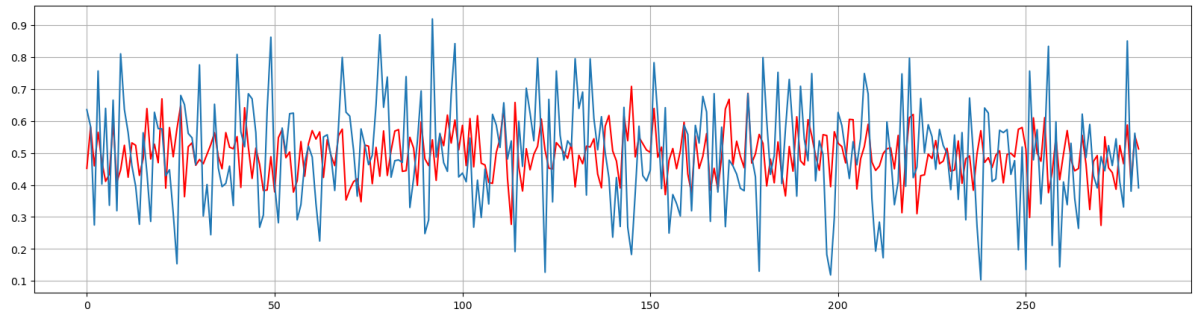

## Случайный лес

In [49]:
```
from sklearn.svm import SVR
```

In [50]:
```
svr = SVR().fit(X_el_train, y_el_train)
y_el_pred_svr = svr.predict(X_el_test)
```

In [51]:
```
# оценка качества модели
svr_result = pd.DataFrame({
    'Model': 'SVR',
    'MSE': mean_squared_error(y_el_test, y_el_pred_svr),
    'MAE': mean_absolute_error(y_el_test, y_el_pred_svr),
    'R2 score': r2_score(y_el_test, y_el_pred_svr).round(3)
}, index=['Модуль упругости при растяжении'])
models = pd.concat([models, svr_result])
```

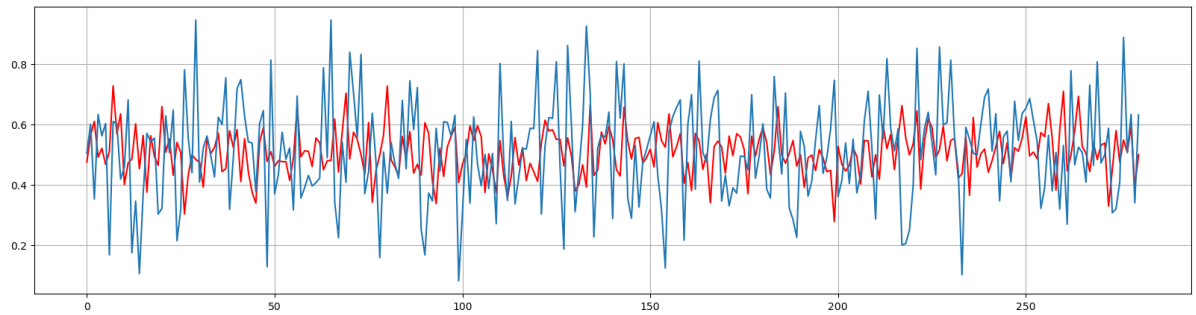
In [52]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_el_pred_svr, "r", label="prediction")
plt.plot(y_el_test.values, label="actual")
plt.grid(True);
```




In [53]:
```
svr = SVR().fit(X_str_train, y_str_train)
y_str_pred_svr = svr.predict(X_str_test)
```

In [54]:
```
# оценка качества модели
svr_result = pd.DataFrame({
    'Model': 'SVR',
    'MSE': mean_squared_error(y_str_test, y_str_pred_svr),
    'MAE': mean_absolute_error(y_str_test, y_str_pred_svr),
    'R2 score': r2_score(y_str_test, y_str_pred_svr).round(3)
}, index=['Прочность при растяжении, МПа'])
models = pd.concat([models, svr_result])
```

In [55]:
```
plt.figure(figsize=(20, 5))
plt.plot(y_str_pred_svr, "r", label="prediction")
plt.plot(y_str_test.values, label="actual")
plt.grid(True);
```




In [61]:
```
models.sort_values(by=['R2 score'], ascending=False).round(3)
```

Out[61]:

| | Model | MSE | MAE | R2 score |
|---|---|---|---|---|
| **Модуль упругости при растяжении** | LinearRegression | 0.027 | 0.132 | -0.007 |
| **Прочность при растяжении, МПа** | LinearRegression | 0.029 | 0.136 | -0.012 |
| **Модуль упругости при растяжении** | MLPRegressor | 0.028 | 0.135 | -0.062 |
| **Модуль упругости при растяжении** | MLPRegressor | 0.028 | 0.135 | -0.062 |
| **Прочность при растяжении, МПа** | KNeighborsRegressor | 0.033 | 0.143 | -0.155 |
| **Прочность при растяжении, МПа** | SVR | 0.034 | 0.146 | -0.185 |
| **Модуль упругости при растяжении** | SVR | 0.032 | 0.144 | -0.207 |
| **Модуль упругости при растяжении** | KNeighborsRegressor | 0.034 | 0.150 | -0.284 |
| **Прочность при растяжении, МПа** | DecisionTreeRegressor | 0.054 | 0.189 | -0.898 |
| **Модуль упругости при растяжении** | DecisionTreeRegressor | 0.059 | 0.191 | -1.231 |

Если R2<0, это значит, что разработанная модель даёт прогноз даже хуже, чем простое усреднение.

In [ ]: