Построение модели машинного обучения для прогноза модуля упругости при растяжении и модели машинного обучения для прогноза прочности при растяжении

Импортируем необходимые библиотеки

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler

import warnings
warnings.filterwarnings("ignore")
```

Загружаем итоговый датасет

```
df = pd.read_excel('db_itog_bez_norm.xlsx')

df.drop('Unnamed: 0', axis=1,inplace=True)
df.head()
```

```
   Соотношение матрица-наполнитель  Плотность, кг/м3  модуль
упругости, ГПа  \
0                         1.857143            2030.0
738.736842
1                         1.857143            2030.0
738.736842
2                         1.857143            2030.0
738.736842
3                         2.771331            2030.0
753.000000
4                         2.767918            2000.0
748.000000

   Количество отвердителя, м.%  Поверхностная плотность, г/м2  \
0                        50.00                          210.0
1                        49.90                          210.0
2                       129.00                          210.0
3                       111.86                          210.0
4                       111.86                          210.0
```

```
   Модуль упругости при растяжении, ГПа  Прочность при растяжении, МПа
\
0                                  70.0                           3000.0

1                                  70.0                           3000.0

2                                  70.0                           3000.0

3                                  70.0                           3000.0

4                                  70.0                           3000.0


   Шаг нашивки  Плотность нашивки  Угол нашивки, град_0  \
0          4.0               60.0                     1
1          4.0               70.0                     1
2          5.0               47.0                     1
3          5.0               57.0                     1
4          5.0               60.0                     1

   Угол нашивки, град_90  Содержание эпоксидных групп, г  \
0                      0                       52.250000
1                      0                       72.600000
2                      0                       46.750000
3                      0                       48.989286
4                      0                       48.989286

   Потребление смолы, г/м2 без ЭГ
0                      167.750000
1                      147.400000
2                      173.250000
3                      171.010714
4                      171.010714
```

Подготавливаем датафреймы для создания обучающей и тестовой выборки, а именно исключаем и выносим в отдельный датафрейм переменные в отношении которых необходимо подготовить прогнозирующую модель

```
data_for_models = df.copy()
X_proch = df.drop("Прочность при растяжении, МПа", axis=1)
y_proch = df['Прочность при растяжении, МПа']

X_mod_uprug = df.drop("Модуль упругости при растяжении, ГПа", axis=1)
y_mod_uprug = df["Модуль упругости при растяжении, ГПа"]
```

Масштабирование данных

Масштабирование данных — это процесс преобразования числовых признаков в наборе данных к одному масштабу, чтобы они имели одинаковый диапазон значений. Это необходимо, потому что многие алгоритмы машинного обучения чувствительны к разным

масштабам признаков, и масштабирование помогает избежать ситуации, когда один признак доминирует над другими. Масштабирование применяется для: Улучшение работы алгоритмов. Многие алгоритмы, особенно те, что основаны на расстоянии (например, KNN, K-means, SVM) или градиентном спуске, работают лучше, когда признаки имеют одинаковый масштаб. Предотвращение доминирования признаков. Если один признак имеет большой диапазон значений, а другой – маленький, то первый может оказывать большее влияние на модель, даже если его важность не выше. Масштабирование выравнивает вклад каждого признака. Оптимизация обучения. Масштабирование может ускорить процесс обучения модели и повысить ее точность.

Нормализация (Max-Min Normalization, Min-Max Scaling) – техника преобразования значений признака, масштабирующая значения таким образом, что они располагаются в диапазоне от 0 до 1. Цель такого преобразования – изменить значения числовых столбцов в наборе данных так, чтобы сохранить различия их диапазонов. В Машинном обучении датасет требует нормализации, когда признаки имеют разные диапазоны и тем самым способствуют искажению восприятия взаимоотношений между Переменными-предикторами (Predictor Variable) и Целевой переменной (Target Variable).

Принимаем решения о нормализации данных. Используем MinMaxScaler из модуля sklearn.

Нормализуем данные дата фрейма X_proch с помощью MinMaxScaler()

```
mms_proch = MinMaxScaler()
col = X_proch.columns
result = mms_proch.fit_transform(X_proch)

X_proch = pd.DataFrame(result, columns = col)
X_proch.describe()
```

```
       Соотношение матрица-наполнитель  Плотность, кг/м3  \
count                       925.000000        925.000000
mean                          0.497546          0.503436
std                           0.188196          0.188667
min                           0.000000          0.000000
25%                           0.370724          0.368017
50%                           0.494224          0.511348
75%                           0.629154          0.626775
max                           1.000000          1.000000

       модуль упругости, ГПа  Количество отвердителя, м.%  \
count             925.000000                   925.000000
mean                0.450854                     0.505851
std                 0.201595                     0.187418
min                 0.000000                     0.000000
25%                 0.305176                     0.378362
50%                 0.452951                     0.506352
75%                 0.586757                     0.639114
max                 1.000000                     1.000000

       Поверхностная плотность, г/м2  Модуль упругости при растяжении,
```

```
ГПа  \
count                      925.000000
925.000000
mean                         0.374074
0.487438
std                          0.216236
0.195421
min                          0.000000
0.000000
25%                          0.206500
0.355991
50%                          0.356477
0.484115
75%                          0.537667
0.617537
max                          1.000000
1.000000

       Шаг нашивки  Плотность нашивки  Угол нашивки, град_0  \
count   925.000000          925.000000             925.000000
mean      0.502436            0.504042               0.486486
std       0.183617            0.192942               0.500088
min       0.000000            0.000000               0.000000
25%       0.370933            0.376961               0.000000
50%       0.504549            0.504853               0.000000
75%       0.625697            0.631086               1.000000
max       1.000000            1.000000               1.000000

       Угол нашивки, град_90  Содержание эпоксидных групп, г  \
count              925.000000                       925.000000
mean                 0.513514                         0.486999
std                  0.500088                         0.190326
min                  0.000000                         0.000000
25%                  0.000000                         0.347856
50%                  1.000000                         0.486436
75%                  1.000000                         0.615519
max                  1.000000                         1.000000

       Потребление смолы, г/м2 без ЭГ
count                      925.000000
mean                         0.505240
std                          0.190173
min                          0.000000
25%                          0.380455
50%                          0.504801
75%                          0.637710
max                          1.000000

y_proch.describe()
```

```
count     925.000000
mean     2460.116851
std       450.916069
min      1250.392802
25%      2149.974687
50%      2456.395009
75%      2751.235671
max      3636.892992
Name: Прочность при растяжении, МПа, dtype: float64
```

Нормализуем данные дата фрейма X_mod_uprug с помощью MinMaxScaler()

```
mms_mod_uprug = MinMaxScaler()
col = X_mod_uprug.columns
result = mms_mod_uprug.fit_transform(X_mod_uprug)

X_mod_uprug = pd.DataFrame(result, columns = col)
X_mod_uprug.describe()
```

```
       Соотношение матрица-наполнитель  Плотность, кг/м3  \
count                        925.000000        925.000000
mean                           0.497546          0.503436
std                            0.188196          0.188667
min                            0.000000          0.000000
25%                            0.370724          0.368017
50%                            0.494224          0.511348
75%                            0.629154          0.626775
max                            1.000000          1.000000

       модуль упругости, ГПа  Количество отвердителя, м.%  \
count             925.000000                   925.000000
mean                0.450854                     0.505851
std                 0.201595                     0.187418
min                 0.000000                     0.000000
25%                 0.305176                     0.378362
50%                 0.452951                     0.506352
75%                 0.586757                     0.639114
max                 1.000000                     1.000000

       Поверхностная плотность, г/м2  Прочность при растяжении, МПа  \
count                     925.000000                     925.000000
mean                        0.374074                       0.506903
std                         0.216236                       0.188944
min                         0.000000                       0.000000
25%                         0.206500                       0.376946
50%                         0.356477                       0.505343
75%                         0.537667                       0.628889
max                         1.000000                       1.000000

       Шаг нашивки  Плотность нашивки  Угол нашивки, град_0  \
```

```
count   925.000000          925.000000              925.000000
mean      0.502436            0.504042                0.486486
std       0.183617            0.192942                0.500088
min       0.000000            0.000000                0.000000
25%       0.370933            0.376961                0.000000
50%       0.504549            0.504853                0.000000
75%       0.625697            0.631086                1.000000
max       1.000000            1.000000                1.000000

       Угол нашивки, град_90  Содержание эпоксидных групп, г  \
count             925.000000                      925.000000
mean                0.513514                        0.486999
std                 0.500088                        0.190326
min                 0.000000                        0.000000
25%                 0.000000                        0.347856
50%                 1.000000                        0.486436
75%                 1.000000                        0.615519
max                 1.000000                        1.000000

       Потребление смолы, г/м2 без ЭГ
count                      925.000000
mean                         0.505240
std                          0.190173
min                          0.000000
25%                          0.380455
50%                          0.504801
75%                          0.637710
max                          1.000000
```

```
y_mod_uprug.describe()
```

```
count    925.000000
mean      73.304924
std        3.011295
min       65.793845
25%       71.279418
50%       73.253725
75%       75.309657
max       81.203147
Name: Модуль упругости при растяжении, ГПа, dtype: float64
```

Создание обучающей и тестирующей выборки для прогноза по переменной "Прочность при растяжении, МПа". Разделение по принципу 70% данных относится к обучающей выборки и 30% относится к тестирующей выборки.

```
X_train_proch, X_test_proch, y_train_proch, y_test_proch =
train_test_split(X_proch, y_proch, test_size=0.3, random_state=0,
shuffle=True)
```

Проверям что обучающая и тестирующая выборки для прогноза по переменной "Прочность при растяжении, МПа" созданы корректно.

```
X_train_proch.head()
```

```
     Соотношение матрица-наполнитель  Плотность, кг/м3  модуль
упругости, ГПа  \
48                            0.158102          0.725190
0.576675
306                           0.619850          0.769181
0.263654
263                           0.532021          0.559514
0.443638
749                           0.722379          0.619887
0.435784
648                           0.376244          0.491648
0.146501

     Количество отвердителя, м.%  Поверхностная плотность, г/м2  \
48                      0.682142                       0.405384
306                     0.501200                       0.541645
263                     0.559515                       0.129828
749                     0.441711                       0.542370
648                     0.259451                       0.545220

     Модуль упругости при растяжении, ГПа  Шаг нашивки  Плотность
нашивки  \
48                                 0.232913     0.446045
0.726996
306                                0.473322     0.160618
0.379040
263                                0.547563     0.229111
0.191757
749                                0.503371     0.755744
0.420222
648                                0.588516     0.502273
0.224795

     Угол нашивки, град_0  Угол нашивки, град_90  \
48                    1.0                     0.0
306                   1.0                     0.0
263                   1.0                     0.0
749                   0.0                     1.0
648                   0.0                     1.0

     Содержание эпоксидных групп, г  Потребление смолы, г/м2 без ЭГ
48                          0.529288                         0.502252
306                         0.517291                         0.468870
263                         0.359648                         0.240430
```

```
749                          0.522485                          0.588697
648                          0.601278                          0.583609
```

```
X_train_proch.shape
```

```
(647, 12)
```

```
X_test_proch.head()
```

```
     Соотношение матрица-наполнитель  Плотность, кг/м3  модуль
упругости, ГПа  \
308                           0.330840          0.493907
0.334887
352                           0.520203          0.825374
0.564928
665                           0.571982          0.487762
0.569393
909                           0.301038          0.491187
0.328963
812                           0.871067          0.470143
0.086004

     Количество отвердителя, м.%  Поверхностная плотность, г/м2  \
308                     0.901624                       0.271493
352                     0.468778                       0.665379
665                     0.462942                       0.262282
909                     0.472701                       0.208149
812                     0.748675                       0.472314

     Модуль упругости при растяжении, ГПа  Шаг нашивки  Плотность
нашивки  \
308                                0.561482     0.354639
0.236418
352                                0.430624     0.328661
0.608536
665                                0.587542     0.514526
0.421655
909                                0.570038     0.176512
0.251281
812                                0.444866     0.202120
0.438186

     Угол нашивки, град_0  Угол нашивки, град_90  \
308                   1.0                    0.0
352                   1.0                    0.0
665                   0.0                    1.0
909                   0.0                    1.0
812                   0.0                    1.0

     Содержание эпоксидных групп, г  Потребление смолы, г/м2 без ЭГ
308                         0.541704                          0.655402
```

```
352                          0.378438                              0.450856
665                          0.721889                              0.714210
909                          0.875925                              0.752797
812                          0.396252                              0.423092
```

```
X_test_proch.shape
```

```
(278, 12)
```

```
y_train_proch.head()
```

```
48     2561.544646
306    3141.616040
263    2370.338363
749    2647.588058
648    2646.858411
Name: Прочность при растяжении, МПа, dtype: float64
```

```
y_train_proch.shape
```

```
(647,)
```

```
y_test_proch.head()
```

```
308    2723.281123
352    2844.655510
665    2648.655464
909    2322.432527
812    2253.684145
Name: Прочность при растяжении, МПа, dtype: float64
```

```
y_test_proch.shape
```

```
(278,)
```

Создание обучающей и тестирующей выборки для прогноза по переменной "Модуль упругости при растяжении, ГПа". Разделение по принципу 70% данных относится к обучающей выборки и 30% относится к тестирующей выборки.

```
X_train_mod_uprug, X_test_mod_uprug, y_train_mod_uprug,
y_test_mod_uprug = train_test_split(X_mod_uprug, y_mod_uprug,
test_size=0.3, random_state=0, shuffle=True)
```

Проверяем что обучающие и тестовые выборки дял прогноза по переменной "Модуль упругости при растяжении, ГПа" созданны корректно.

```
X_train_mod_uprug.head()
```

```
     Соотношение матрица-наполнитель  Плотность, кг/м3  модуль
упругости, ГПа  \
48                          0.158102          0.725190
```

```
0.576675
306                          0.619850          0.769181
0.263654
263                          0.532021          0.559514
0.443638
749                          0.722379          0.619887
0.435784
648                          0.376244          0.491648
0.146501

     Количество отвердителя, м.%  Поверхностная плотность, г/м2  \
48                      0.682142                       0.405384
306                     0.501200                       0.541645
263                     0.559515                       0.129828
749                     0.441711                       0.542370
648                     0.259451                       0.545220

     Прочность при растяжении, МПа  Шаг нашивки  Плотность нашивки  \
48                        0.549404     0.446045           0.726996
306                       0.792467     0.160618           0.379040
263                       0.469284     0.229111           0.191757
749                       0.585458     0.755744           0.420222
648                       0.585152     0.502273           0.224795

     Угол нашивки, град_0  Угол нашивки, град_90  \
48                    1.0                     0.0
306                   1.0                     0.0
263                   1.0                     0.0
749                   0.0                     1.0
648                   0.0                     1.0

     Содержание эпоксидных групп, г  Потребление смолы, г/м2 без ЭГ
48                          0.529288                         0.502252
306                         0.517291                         0.468870
263                         0.359648                         0.240430
749                         0.522485                         0.588697
648                         0.601278                         0.583609

X_train_mod_uprug.shape

(647, 12)

X_test_mod_uprug.head()

     Соотношение матрица-наполнитель  Плотность, кг/м3  модуль
упругости, ГПа  \
308                          0.330840          0.493907
0.334887
352                          0.520203          0.825374
0.564928
665                          0.571982          0.487762
```

```
0.569393
909                          0.301038          0.491187
0.328963
812                          0.871067          0.470143
0.086004

     Количество отвердителя, м.%  Поверхностная плотность, г/м2  \
308                     0.901624                        0.271493
352                     0.468778                        0.665379
665                     0.462942                        0.262282
909                     0.472701                        0.208149
812                     0.748675                        0.472314

     Прочность при растяжении, МПа  Шаг нашивки  Плотность нашивки  \
308                        0.617175     0.354639           0.236418
352                        0.668034     0.328661           0.608536
665                        0.585905     0.514526           0.421655
909                        0.449210     0.176512           0.251281
812                        0.420403     0.202120           0.438186

     Угол нашивки, град_0  Угол нашивки, град_90  \
308                   1.0                     0.0
352                   1.0                     0.0
665                   0.0                     1.0
909                   0.0                     1.0
812                   0.0                     1.0

     Содержание эпоксидных групп, г  Потребление смолы, г/м2 без ЭГ
308                         0.541704                         0.655402
352                         0.378438                         0.450856
665                         0.721889                         0.714210
909                         0.875925                         0.752797
812                         0.396252                         0.423092

X_test_mod_uprug.shape

(278, 12)

y_train_mod_uprug.head()

48     69.382866
306    73.087407
263    74.231408
749    73.550433
648    74.862468
Name: Модуль упругости при растяжении, ГПа, dtype: float64

y_train_mod_uprug.shape

(647,)
```

```
y_test_mod_uprug.head()
```

```
308    74.445891
352    72.429467
665    74.847458
909    74.577734
812    72.648916
Name: Модуль упругости при растяжении, ГПа, dtype: float64
```

```
y_test_mod_uprug.shape
```

```
(278,)
```

Метрики

Метрики в машинном обучении – это количественные показатели, используемые для оценки производительности моделей. Они позволяют оценить, насколько хорошо модель выполняет поставленную задачу, будь то классификация, регрессия или другие типы задач. Выбор правильной метрики критически важен для оценки и сравнения различных моделей, а также для оптимизации их работы.

При оценке моделей будем использовать следующие метрики:

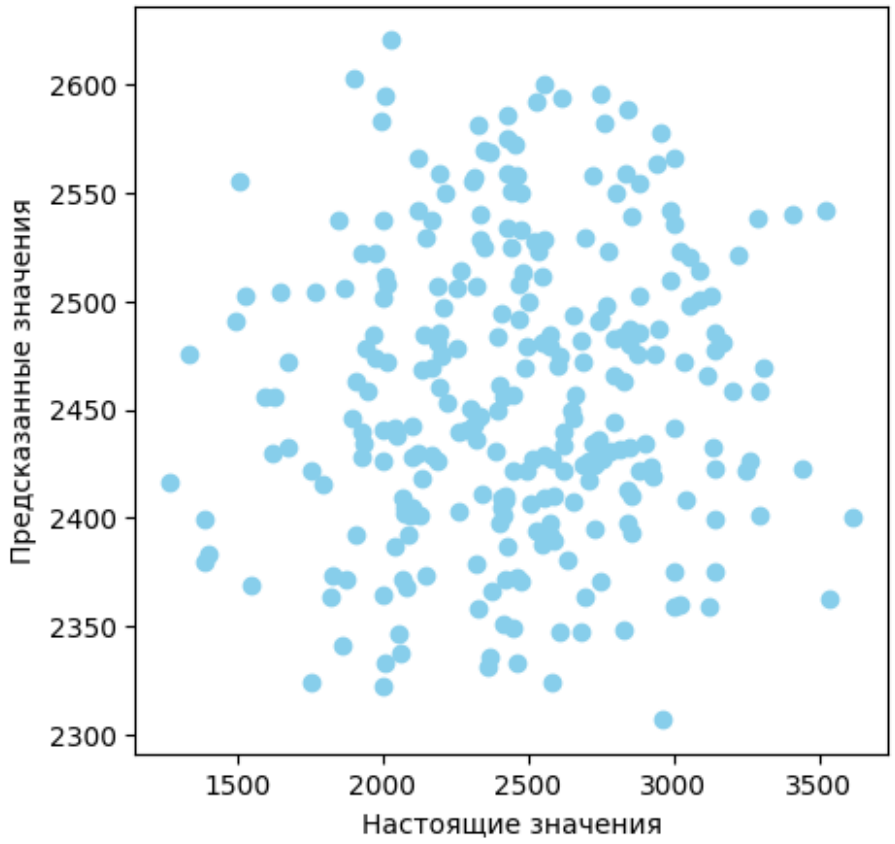
1. Корень средней квадратичной ошибки (RMSE) является одним из двух основных показателей эффективности для моделей прогнозирования регрессии. Это исследование устанавливает среднюю разницу между значениями, спрогнозированными моделями прогнозирования и фактическими значениями. Он дает оценку того, что хорошая модель может предсказать целевое значение (оценку точности).
2. MAE, или Mean Absolute Error (Средняя абсолютная ошибка), это метрика, которая используется для оценки точности моделей машинного обучения, в частности, регрессионных моделей. Она показывает среднее значение абсолютных разностей между прогнозируемыми и фактическими значениями. Чем меньше значение MAE, тем лучше модель предсказывает данные.
3. R2 - Статистический показатель, отражающий объясняющую способность регрессии f: X→Y и определяемый как доля дисперсии зависимой переменной, объясненная регрессионной моделью с данным набором независимых переменных. Коэффициент детерминации изменяется в диапазоне от – ∞ до 1. Если он равен 1, это соответствует идеальной модели, когда все точки наблюдений лежат точно на линии регрессии, т.е. сумма квадратов их отклонений равна 0. Если коэффициент детерминации равен 0, это означает, что связь между переменными регрессионной модели отсутствует, и вместо нее для оценки значения выходной переменной можно использовать простое среднее ее наблюдаемых значений.
4. MAPE, или средняя абсолютная процентная ошибка (Mean Absolute Percentage Error), это метрика, используемая для оценки точности моделей прогнозирования. Она показывает среднее значение абсолютной процентной разницы между фактическими и прогнозируемыми значениями. Другими словами, MAPE измеряет, насколько в среднем прогнозы отличаются от реальных значений в процентах.

Построение модели на основе Линейной регрессии (LinearRegression)

Линейная регрессия - это метод статистического анализа, который используется для моделирования взаимосвязи между зависимой переменной и одной или несколькими независимыми переменными путем подбора линейного уравнения. Проще говоря, она позволяет предсказать значения зависимой переменной, основываясь на значениях независимых переменных.

Построение и тестирование модели Линейной регрессии (LinearRegression) для предсказания переменной "Прочность при растяжении, МПа"

```
# При построении модели пользуемся библиотекой sklearn.linear_model
regressor = LinearRegression()
regressor.fit(X_train_proch, y_train_proch)
y_pred_proch = regressor.predict(X_test_proch)


# Визуализируем на диаграмме рассеяния соотношение предсказанных и
настоящих значений
plt.figure(figsize=(5, 5))
plt.scatter(y_test_proch, y_pred_proch, color="skyblue")
plt.xlabel("Настоящие значения")
plt.ylabel("Предсказанные значения")
plt.show()

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch)))
print("MAE:", metrics.mean_absolute_error(y_test_proch, y_pred_proch))
print("R2", metrics.r2_score(y_test_proch, y_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch))
print("Коэффициенты регрессии:", regressor.coef_)
print("Константа:", regressor.intercept_)

# Создаем датафрейм в который занесем результаты тестирования каждой
модели
# Данный датафрейм используем при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch))],
    [metrics.mean_absolute_error(y_test_proch, y_pred_proch)],
    [metrics.r2_score(y_test_proch, y_pred_proch)],
    [metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch)],
]

metrics_index = ['RMSE','MAE','R2','MAPE']
df_sravnenie = pd.DataFrame(data=data_sravn, index=metrics_index)
df_sravnenie.rename(columns={0:'LinearRegression_proch'},inplace=True)
```

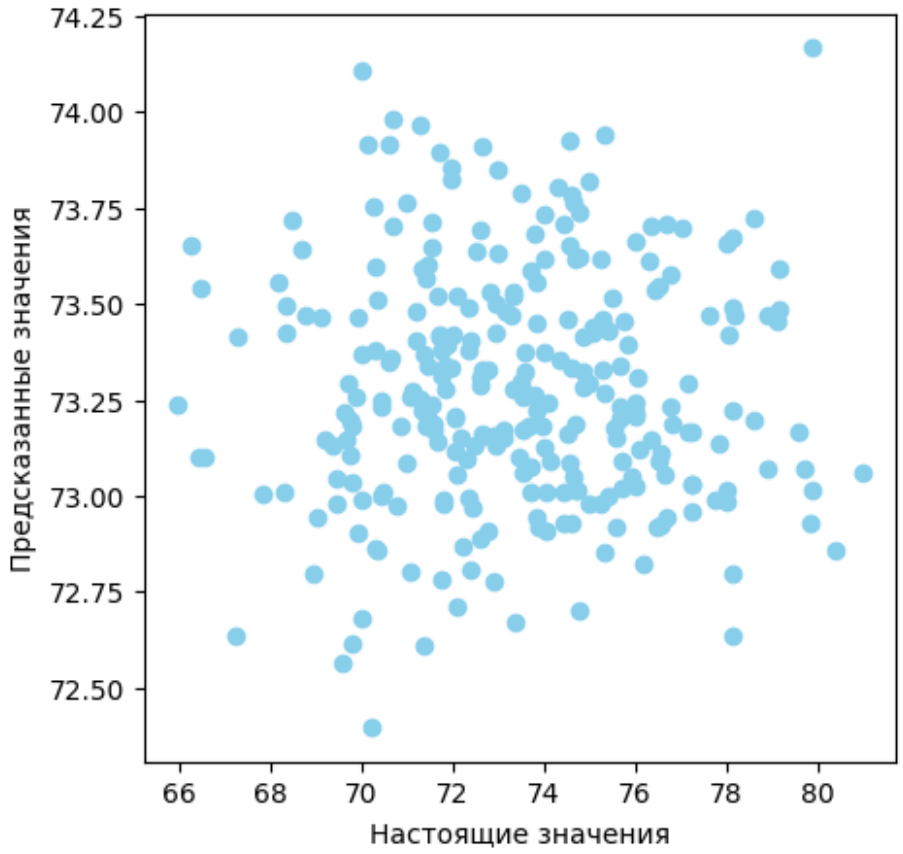



```
Корень из среднеквадратичной ошибки: 451.34204146327596
MAE: 365.02993743964964
R2 -0.003964508608367945
MAPE: 0.15986495520338406
Коэффициенты регрессии: [   3.71788611 -286.22167735   -7.51855558  -
91.27900555   -8.27752849
  -76.00292956 -190.5991545    53.25922147    2.33782713   -2.33782713
   15.19405669  -27.37189604]
Константа: 2766.9711967652634
```

Построение и тестирование модели Линейной регрессии (LinearRegression) для предсказания переменной "Модуль упругости при растяжении, ГПа"

```
# При построении модели пользуемся библиотекой sklearn.linear_model
regressor = LinearRegression()
regressor.fit(X_train_mod_uprug, y_train_mod_uprug)
y_pred_mod_uprug = regressor.predict(X_test_mod_uprug)

# Визуализируем на диаграмме рассеяния соотношение предсказанных и
настоящих значений
plt.figure(figsize=(5, 5))
plt.scatter(y_test_mod_uprug, y_pred_mod_uprug, color="skyblue")
plt.xlabel("Настоящие значения")
```

```
plt.ylabel("Предсказанные значения")
plt.show()

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(y_test_mod_uprug,
y_pred_mod_uprug))
print("R2", metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug))
print("Коэффициенты регрессии:", regressor.coef_)
print("Константа:", regressor.intercept_)

# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug))],
    [metrics.mean_absolute_error(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug)],
]

df_sravnenie['LinearRegression_uprug'] = np.array(data_sravn)
```




```
Корень из среднеквадратичной ошибки: 2.986900936096138
MAE: 2.421377839386351
R2 -0.01208198695063989
MAPE: 0.03298137627511868
Коэффициенты регрессии: [-0.58338399 -0.58997389 -0.17812112 -0.220153
0.1402872  -0.5368078
 -0.61777874  0.47785495 -0.0805161   0.0805161   1.70074587 -
1.30625937]
Константа: 74.16702344988468
```

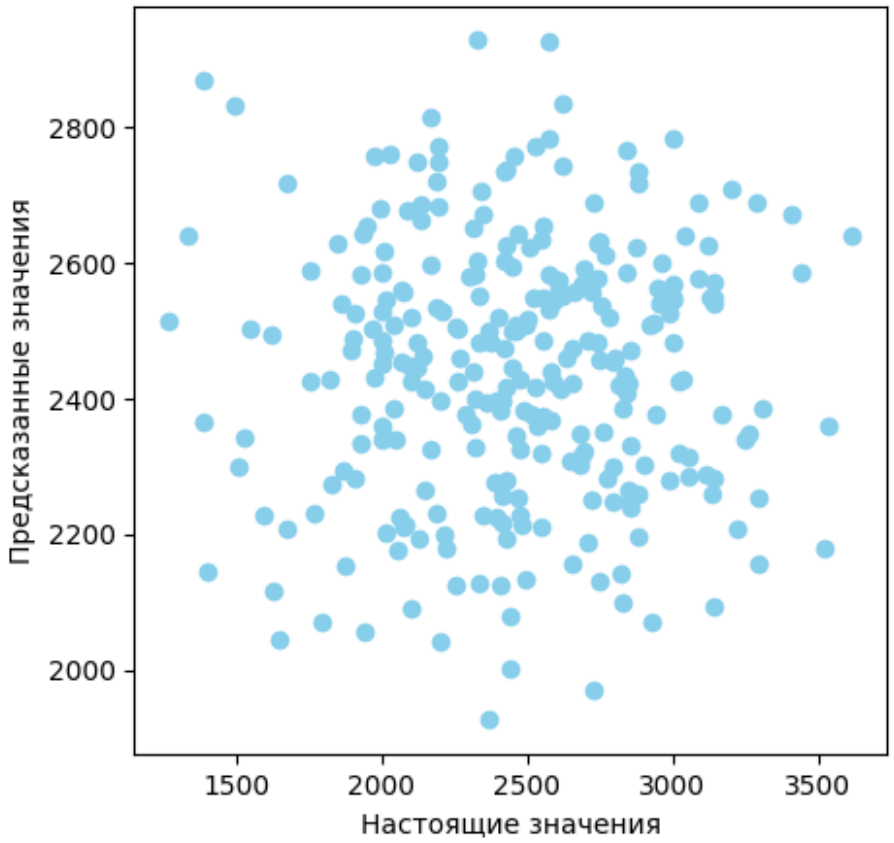
Построение модели Регрессии на основе k-ближайших соседей (KNeighborsRegressor)

K-ближайших соседей (K-Nearest Neighbors или просто KNN) — алгоритм классификации и регрессии, основанный на гипотезе компактности, которая предполагает, что расположенные близко друг к другу объекты в пространстве признаков имеют схожие значения целевой переменной или принадлежат к одному классу.

Построение и тестирование модели Регрессии на основе k-ближайших соседей (KNeighborsRegressor) для предсказания переменной "Прочность при растяжении, МПа"

```
# при построении модели используем библиотеку sklearn
knn = KNeighborsRegressor()
```

```
knn.fit(X_train_proch, y_train_proch)
y_pred_proch = knn.predict(X_test_proch)

# Визуализируем на диаграмме рассеяния соотношение предсказанных и
настоящих значений
plt.figure(figsize=(5, 5))
plt.scatter(y_test_proch, y_pred_proch, color='skyblue')
plt.xlabel("Настоящие значения")
plt.ylabel("Предсказанные значения")
plt.show()

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch)))
print("MAE:", metrics.mean_absolute_error(y_test_proch, y_pred_proch))
print("R2", metrics.r2_score(y_test_proch, y_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch))
```




```
Корень из среднеквадратичной ошибки: 488.60301188371545
MAE: 392.86910688443356
R2 -0.1765734982566982
MAPE: 0.17042995789958992
```

Проведем поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10. Используем объект GridSearchCV из библиотеки sklearn, который поможет выбрать наилучшие параметры.

```
# Подберём параметры используя GridSearchCV
n_neighbors = range(1, 50)
p = [1, 2]
weights = ["uniform", "distance"]
metric = ["mahalanobis", "minkowski", "cosine", "chebyshev",
"correlation", "euclidean"]
algorithm = ["ball_tree", "kd_tree", "brute"]

hyperparameters = dict(n_neighbors=n_neighbors, weights=weights, p=p,
metric=metric, algorithm=algorithm)

knn = KNeighborsRegressor()
search = GridSearchCV(knn, hyperparameters, cv=10, verbose=1)
best_model = search.fit(X_train_proch, y_train_proch)
```

```
Fitting 10 folds for each of 3528 candidates, totalling 35280 fits
```

```
best_model.best_estimator_
```

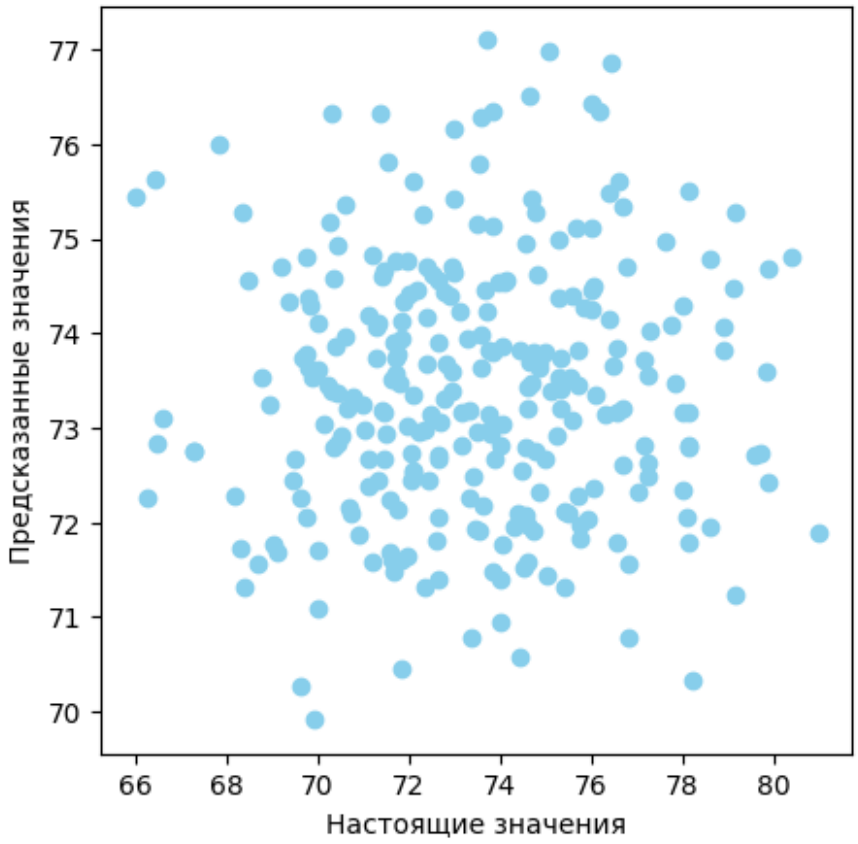
```
KNeighborsRegressor(algorithm='ball_tree', n_neighbors=48, p=1)
```

```
knn = best_model.best_estimator_
knn.fit(X_train_proch, y_train_proch)
y_pred_proch = knn.predict(X_test_proch)

# Выводим значения метрик (после поиска гиперпараметров)
print("Корень из среднеквадратичной
ошибки:",np.sqrt(metrics.mean_squared_error(y_test_proch,
y_pred_proch)))
print("MAE:", metrics.mean_absolute_error(y_test_proch, y_pred_proch))
print("R2", metrics.r2_score(y_test_proch, y_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch))

# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch))],
    [metrics.mean_absolute_error(y_test_proch, y_pred_proch)],
    [metrics.r2_score(y_test_proch, y_pred_proch)],
    [metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch)],
]

df_sravnenie["KNR_GSCV_proch"] = np.array(data_sravn)
```

```
Корень из среднеквадратичной ошибки: 452.4531325358677
MAE: 365.86009954452174
R2 -0.00891361122238754
MAPE: 0.16084622619523814
```

Построение и тестирование модели Регрессии на основе k-ближайших соседей (KNeighborsRegressor) для предсказания переменной "Модуль упругости при растяжении, ГПа"

```
# при построении модели используем библиотеку sklearn
knn = KNeighborsRegressor()

knn.fit(X_train_mod_uprug, y_train_mod_uprug)
y_pred_mod_uprug = knn.predict(X_test_mod_uprug)

# Визуализируем на диаграмме рассеяния соотношение предсказанных и
настоящих значений
plt.figure(figsize=(5, 5))
plt.scatter(y_test_mod_uprug, y_pred_mod_uprug, color="skyblue")
plt.xlabel("Настоящие значения")
plt.ylabel("Предсказанные значения")
plt.show()
# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(y_test_mod_uprug,
y_pred_mod_uprug))
print("R2", metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug))
```




```
Корень из среднеквадратичной ошибки: 3.23140577094596
MAE: 2.603313484473943
R2 -0.18455996317736534
MAPE: 0.03549890556868933
```

Проведем поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10. Используем объект GridSearchCV из библиотеки sklearn, который поможет выбрать наилучшие параметры.

```
# Подберём параметры используя GridSearchCV
n_neighbors = range(1, 50)
p = [1, 2]
weights = ["uniform", "distance"]
metric = ["mahalanobis", "minkowski", "cosine", "chebyshev",
"correlation", "euclidean"]
algorithm = ["ball_tree", "kd_tree", "brute"]

hyperparameters = dict(n_neighbors=n_neighbors, weights=weights, p=p,
metric=metric, algorithm=algorithm)

knn = KNeighborsRegressor()
search = GridSearchCV(knn, hyperparameters, cv=10, verbose=1)
best_model = search.fit(X_train_mod_uprug, y_train_mod_uprug)
```

```
Fitting 10 folds for each of 3528 candidates, totalling 35280 fits
```

```
best_model.best_estimator_
```

```
KNeighborsRegressor(algorithm='ball_tree', n_neighbors=33, p=1)
```

```
knn = best_model.best_estimator_
knn.fit(X_train_mod_uprug, y_train_mod_uprug)
y_pred_mod_uprug = knn.predict(X_test_mod_uprug)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(y_test_mod_uprug,
y_pred_mod_uprug))
print("R2", metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug))

# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug))],
    [metrics.mean_absolute_error(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug)],
]

df_sravnenie["KNR_GSCV_upr"] = np.array(data_sravn)
```

```
Корень из среднеквадратичной ошибки: 2.9824205901887875
MAE: 2.4341991282788467
R2 -0.009048021893739877
MAPE: 0.03314252939637089
```

Построение модели Регрессии на основе Случайного леса (RandomForestRegressor)

Random Forest - это мощный и широко используемый метод ансамблевого обучения в Machine Learning (ML). Он работает путем построения множества деревьев принятия решений в процессе обучения и вывода класса, который является модой классов (классификация) или средним предсказанием (регрессия) отдельных деревьев.

Построение и тестирование модели Регрессии на основе Случайного леса (RandomForestRegressor) для предсказания переменной "Прочность при растяжении, МПа"Прочность

```
# строим модель регрессии RandomForestRegressor с использованием
библиотеки sklearn
rfr = RandomForestRegressor(max_depth=2, random_state=0)
rfr.fit(X_train_proch, y_train_proch)
y_pred_proch = rfr.predict(X_test_proch)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch)))
print("MAE:", metrics.mean_absolute_error(y_test_proch, y_pred_proch))
print("R2", metrics.r2_score(y_test_proch, y_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch))
```

```
Корень из среднеквадратичной ошибки: 450.5104395258723
MAE: 364.0192828490871
R2 -0.000268288893126023
MAPE: 0.15975546800381185
```

Проведем поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10. Используем объект GridSearchCV из библиотеки sklearn, который поможет выбрать наилучшие параметры.

```
parametrs = {
    "n_estimators": [100, 200],
    "max_depth": [9, 12],
    "criterion": ['absolute_error', 'poisson'],
}
grid = GridSearchCV(estimator=rfr, param_grid=parametrs, cv=10)
grid.fit(X_train_proch, y_train_proch)
```

```
GridSearchCV(cv=10,
             estimator=RandomForestRegressor(max_depth=2,
random_state=0),
             param_grid={'criterion': ['absolute_error', 'poisson'],
                         'max_depth': [9, 12], 'n_estimators': [100,
200]})
```

```
rfr = grid.best_estimator_
rfr.fit(X_train_proch, y_train_proch)
y_pred_proch = rfr.predict(X_test_proch)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch)))
print("MAE:", metrics.mean_absolute_error(y_test_proch, y_pred_proch))
print("R2", metrics.r2_score(y_test_proch, y_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch))
```

```
# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch))],
    [metrics.mean_absolute_error(y_test_proch, y_pred_proch)],
    [metrics.r2_score(y_test_proch, y_pred_proch)],
    [metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch)],
]

df_sravnenie["RFR_GSCV_proch"] = np.array(data_sravn)
```

```
Корень из среднеквадратичной ошибки: 449.62387150557396
MAE: 365.95479561510143
R2 0.0036647311565284335
MAPE: 0.1602830113148621
```

Построение и тестирование модели Регрессии на основе Случайного леса (RandomForestRegressor) для предсказания переменной "Модуль упругости при растяжении, ГПа"

```
# строим модель регрессии RandomForestRegressor с использованием
библиотеки sklearn
rfr = RandomForestRegressor(max_depth=2, random_state=0)
rfr.fit(X_train_mod_uprug, y_train_mod_uprug)
y_pred_mod_uprug = rfr.predict(X_test_mod_uprug)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(y_test_mod_uprug,
y_pred_mod_uprug))
print("R2", metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug))
```

```
Корень из среднеквадратичной ошибки: 2.96855495505229
MAE: 2.4082318622147367
R2 0.00031254154346593843
MAPE: 0.03279879316773634
```

Проведем поиск гиперпараметров модели с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 10. Используем объект GridSearchCV из библиотеки sklearn, который поможет выбрать наилучшие параметры.

```
parametrs = {
    "n_estimators": [200, 300],
```

```
    "max_depth": [9, 15],
    "criterion": ['friedman_mse'],
}
grid = GridSearchCV(estimator=rfr, param_grid=parametrs, cv=10)
grid.fit(X_train_mod_uprug, y_train_mod_uprug)
```

```
GridSearchCV(cv=10,
             estimator=RandomForestRegressor(max_depth=2,
random_state=0),
             param_grid={'criterion': ['friedman_mse'], 'max_depth':
[9, 15],
                         'n_estimators': [200, 300]})
```

```
rfr = grid.best_estimator_
rfr.fit(X_train_mod_uprug, y_train_mod_uprug)
y_pred_mod_uprug = rfr.predict(X_test_mod_uprug)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(y_test_mod_uprug,
y_pred_mod_uprug))
print("R2", metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug))

# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug))],
    [metrics.mean_absolute_error(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug)],
]

df_sravnenie["RFR_GSCV_upr"] = np.array(data_sravn)
```

```
Корень из среднеквадратичной ошибки: 2.988807571848938
MAE: 2.426733808727847
R2 -0.013374488863761691
MAPE: 0.0330512642179237
```

Построение модели Байесовская гребневая регрессия (BayesianRidge)

Байесовская линейная регрессия — это подход в линейной регрессии, в котором статистический анализ проводится в контексте байесовского вывода: когда регрессионная модель имеет ошибки, имеющие нормальное распределение, и, если принимается

определённая форма априорного распределения, доступны явные результаты для апостериорных распределений вероятностей параметров модели. Методы байесовской регрессии можно использовать для включения параметров регуляризации в процедуру оценки: параметр регуляризации не задается жестко, а настраивается на имеющиеся данные. Преимущества байесовской регрессии: -Она адаптируется к имеющимся данным. -Его можно использовать для включения параметров регуляризации в процедуру оценки. К недостаткам байесовской регрессии относятся: -Вывод модели может занять много времени.

Построение и тестирование модели Байесовской гребней регрессии (BayesianRidge) для предсказания переменной "Прочность при растяжении, МПа"

```
# Строим модель Байесовской гребней регрессии (BayesianRidge) с
помощью библиотеки sklearn
bsr = BayesianRidge()
bsr.fit(X_train_proch, y_train_proch)
y_pred_proch = bsr.predict(X_test_proch)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch)))
print("MAE:", metrics.mean_absolute_error(y_test_proch, y_pred_proch))
print("R2", metrics.r2_score(y_test_proch, y_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch))

# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch))],
    [metrics.mean_absolute_error(y_test_proch, y_pred_proch)],
    [metrics.r2_score(y_test_proch, y_pred_proch)],
    [metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch)],
]

df_sravnenie["BayesR_proch"] = np.array(data_sravn)
```

```
Корень из среднеквадратичной ошибки: 450.4504014395061
MAE: 361.27922150739045
R2 -1.701542271970169e-06
MAPE: 0.15881184773465556
```

Построение и тестирование модели Байесовской гребней регрессии (BayesianRidge) для предсказания переменной "Модуль упругости при растяжении, ГПа"

```
# Строим модель Байесовской гребней регрессии (BayesianRidge) с
помощью библиотеки sklearn
bsr = BayesianRidge()
bsr.fit(X_train_mod_uprug, y_train_mod_uprug)
```

```
y_pred_mod_uprug = bsr.predict(X_test_mod_uprug)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(y_test_mod_uprug,
y_pred_mod_uprug))
print("R2", metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug))

# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug))],
    [metrics.mean_absolute_error(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug)],
]

df_sravnenie["BayesR_upr"] = np.array(data_sravn)
```

```
Корень из среднеквадратичной ошибки: 2.971975379211404
MAE: 2.4160873421657194
R2 -0.00199250256622463
MAPE: 0.03290899342370176
```

Построение модели на основе Лассо-регрессии (Lasso)

Лассо́-регре́ссия (англ. lasso или LASSO, least absolute shrinkage and selection operator), вариация линейной регрессии, которая используется в статистике и эконометрике для решения проблемы мультиколлинеарности (наличие линейной зависимости между объясняющими переменными) и отбора наиболее информативных (с точки зрения способности объяснять дисперсию зависимой переменной) признаков.

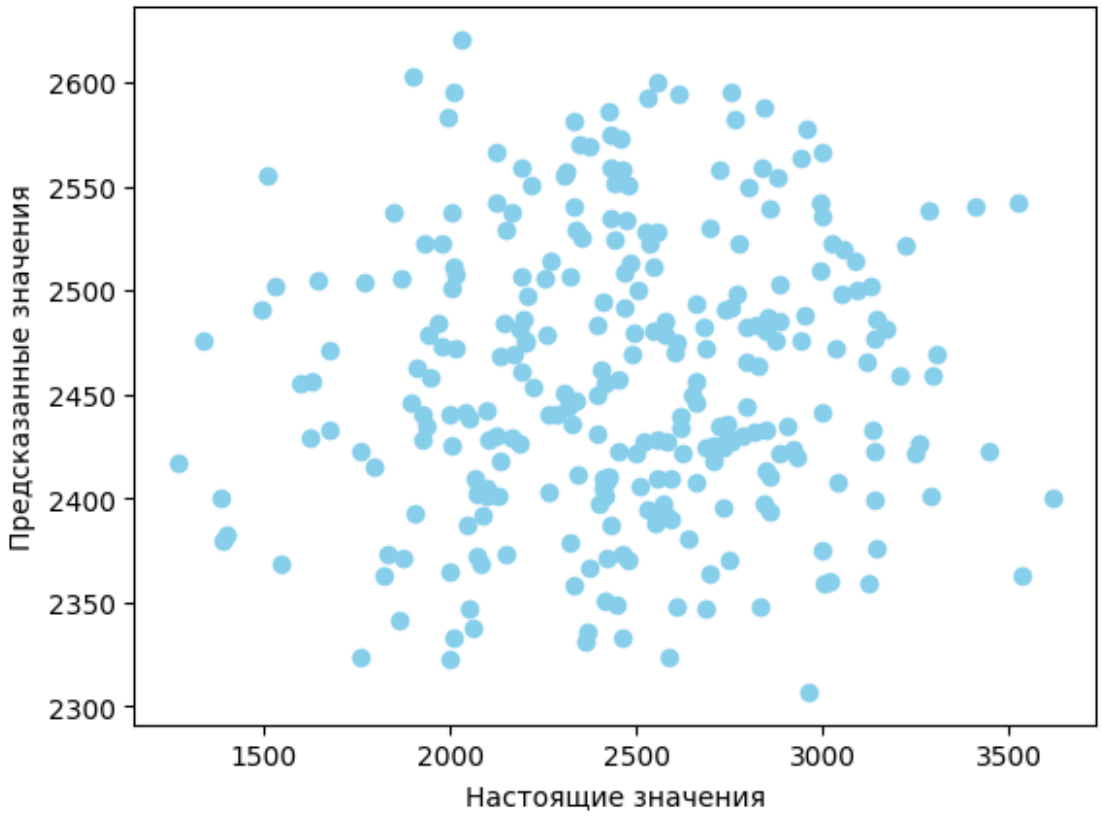
Построение и тестирование модели Лассо-регресии (Lasso) для предсказания переменной "Прочность при растяжении, МПа"

```
# Для построения модели используем библиотеку sklearn
lasso_regressor = Lasso(alpha=0.001)
lasso_regressor.fit(X_train_proch, y_train_proch)
```

```
Lasso(alpha=0.001)
```

```
# Визуализируем на диаграмме рассеяния соотношение предсказанных и
настоящих значений
```

```
y_pred_proch = lasso_regressor.predict(X_test_proch)
plt.scatter(y_test_proch, y_pred_proch, color="skyblue")
plt.xlabel("Настоящие значения")
plt.ylabel("Предсказанные значения")
plt.show()
```




```
# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch)))
print("MAE:", metrics.mean_absolute_error(y_test_proch, y_pred_proch))
print("R2", metrics.r2_score(y_test_proch, y_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch))

# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_proch, y_pred_proch))],
    [metrics.mean_absolute_error(y_test_proch, y_pred_proch)],
    [metrics.r2_score(y_test_proch, y_pred_proch)],
    [metrics.mean_absolute_percentage_error(y_test_proch,
y_pred_proch)],
]
```

```
df_sravnenie["Lasso_proch"] = np.array(data_sravn)
```

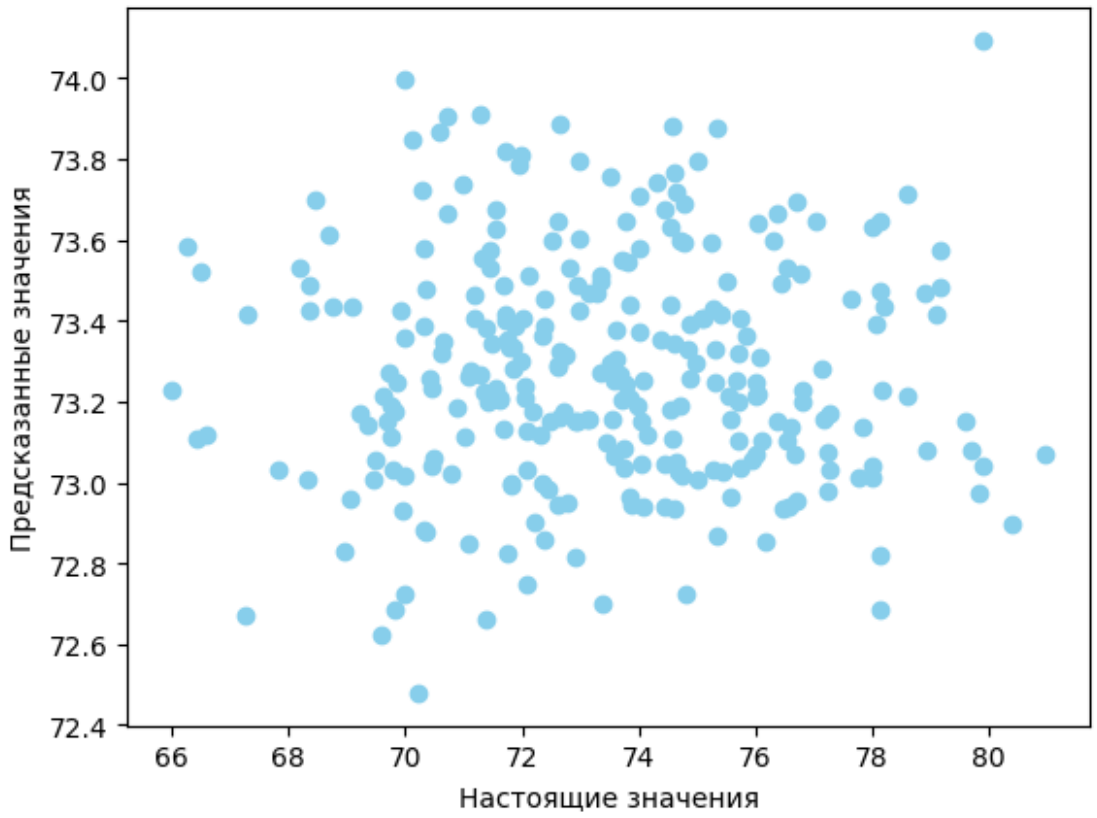
```
Корень из среднеквадратичной ошибки: 451.3393469595164
MAE: 365.0278829816321
R2 -0.003952521344521731
MAPE: 0.15986395083226296
```

Построение и тестирование модели Лассо-регрессии (Lasso) для предсказания переменной "Модуль упругости при растяжении, ГПа"

```
# Для построения модели используем библиотеку sklearn
lasso_regressor = Lasso(alpha=0.001)
lasso_regressor.fit(X_train_mod_uprug, y_train_mod_uprug)
```

```
Lasso(alpha=0.001)
```

```
# Визуализируем на диаграмме рассеяния соотношение предсказанных и настоящих значений
y_pred_mod_uprug = lasso_regressor.predict(X_test_mod_uprug)
plt.scatter(y_test_mod_uprug, y_pred_mod_uprug, color="skyblue")
plt.xlabel("Настоящие значения")
plt.ylabel("Предсказанные значения")
plt.show()
```



```
# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(y_test_mod_uprug,
y_pred_mod_uprug))
print("R2", metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug))

# Переносим полученные метрики в датафрейм, который будет
использоваться при итоговом сравнении моделей
data_sravn = [
    [np.sqrt(metrics.mean_squared_error(y_test_mod_uprug,
y_pred_mod_uprug))],
    [metrics.mean_absolute_error(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.r2_score(y_test_mod_uprug, y_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(y_test_mod_uprug,
y_pred_mod_uprug)],
]

df_sravnenie["Lasso_upr"] = np.array(data_sravn)
```

```
Корень из среднеквадратичной ошибки: 2.984351229664753
MAE: 2.421021489179655
R2 -0.0103548385744614
MAPE: 0.032975563340768836
```

Сравнение моделей

Выводим датафрейм, в который сохранялись значения метрик и для удобства построения диаграмм транспонируем его.

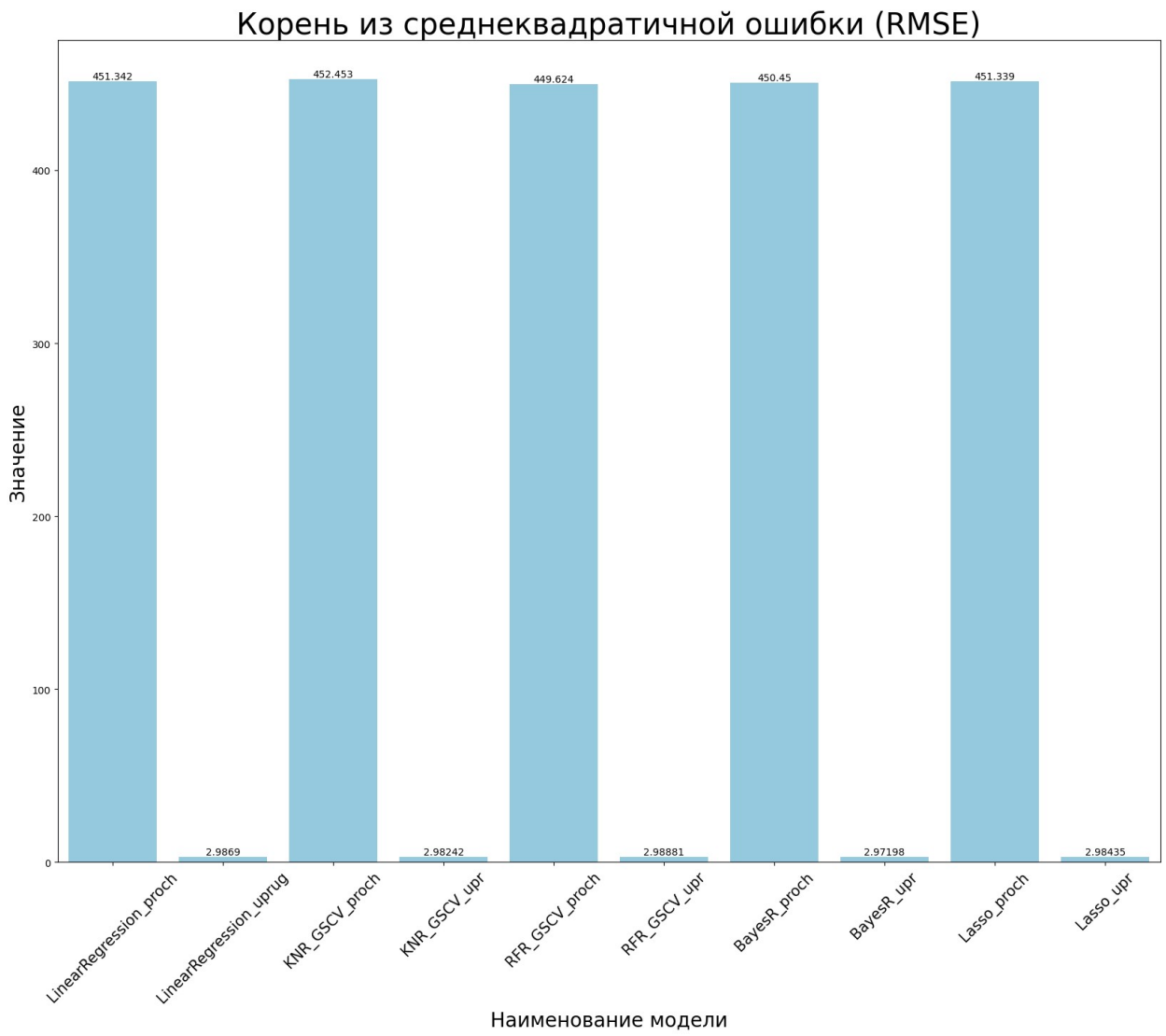
```
df_sravnenie = df_sravnenie.transpose()

df_sravnenie
```

	RMSE	MAE	R2	MAPE
LinearRegression_proch	451.342041	365.029937	-0.003965	0.159865
LinearRegression_uprug	2.986901	2.421378	-0.012082	0.032981
KNR_GSCV_proch	452.453133	365.860100	-0.008914	0.160846
KNR_GSCV_upr	2.982421	2.434199	-0.009048	0.033143
RFR_GSCV_proch	449.623872	365.954796	0.003665	0.160283
RFR_GSCV_upr	2.988808	2.426734	-0.013374	0.033051
BayesR_proch	450.450401	361.279222	-0.000002	0.158812
BayesR_upr	2.971975	2.416087	-0.001993	0.032909
Lasso_proch	451.339347	365.027883	-0.003953	0.159864
Lasso_upr	2.984351	2.421021	-0.010355	0.032976

Сравнение моделей. Метрика - "Корень из среднеквадратичной ошибки (RMSE)"

```
plt.figure(figsize=(20,15))
ax = sns.barplot(df_sravnenie["RMSE"], color="skyblue")
plt.title("Корень из среднеквадратичной ошибки (RMSE)", fontsize=30)
plt.ylabel("Значение", fontsize=20)
plt.xlabel("Наименование модели", fontsize=20)
plt.xticks(rotation=45, fontsize=15)
ax.bar_label(ax.containers[0], fontsize=10)
```

```
[Text(0, 0, '451.342'),
 Text(0, 0, '2.9869'),
 Text(0, 0, '452.453'),
 Text(0, 0, '2.98242'),
 Text(0, 0, '449.624'),
 Text(0, 0, '2.98881'),
 Text(0, 0, '450.45'),
 Text(0, 0, '2.97198'),
 Text(0, 0, '451.339'),
 Text(0, 0, '2.98435')]
```

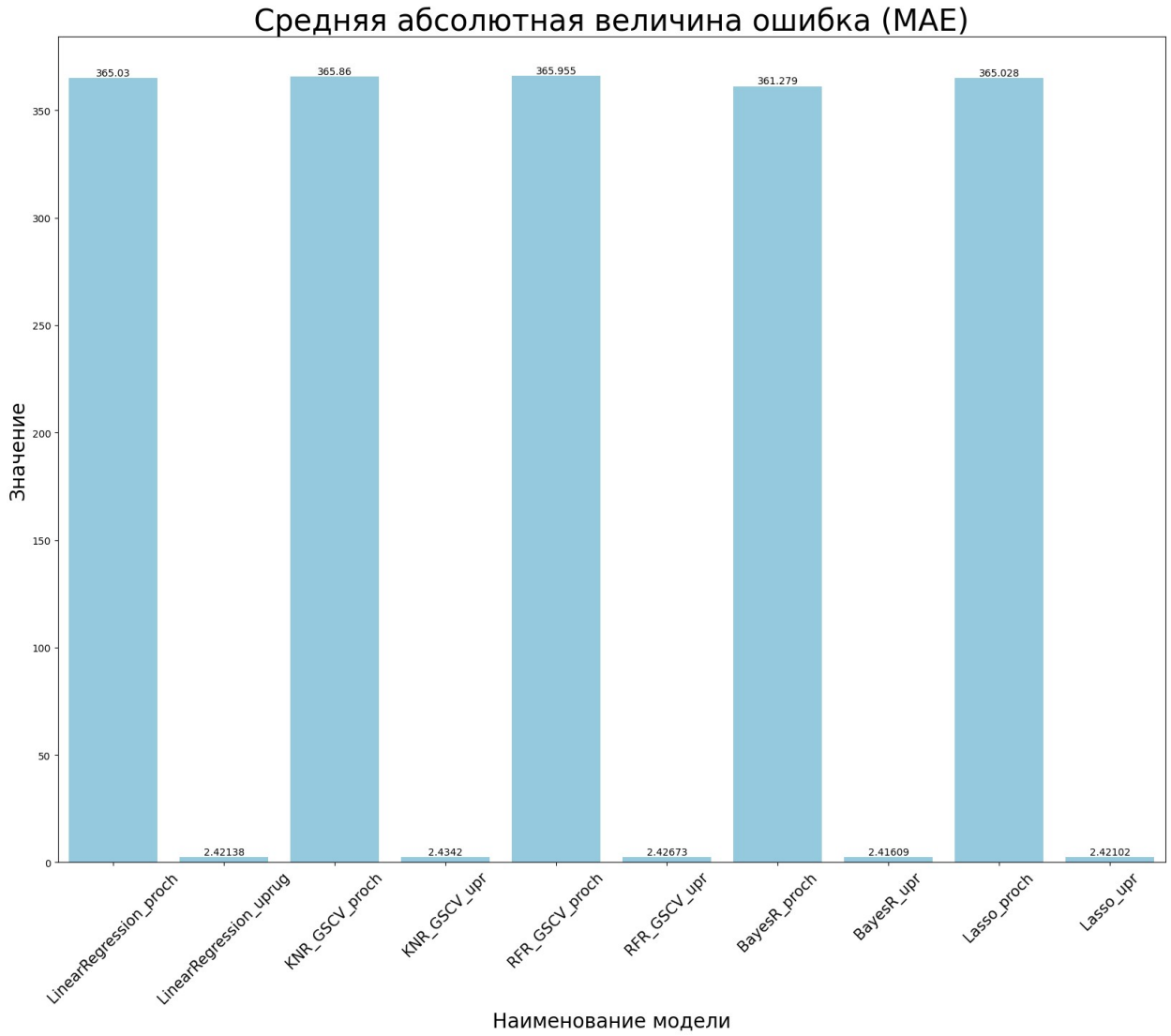


Сравнение моделей. Метрика - "Средняя абсолютная величина ошибка (MAE)"

```
plt.figure(figsize=(20, 15))
ax = sns.barplot(df_sravnenie["MAE"], color="skyblue")
plt.title("Средняя абсолютная величина ошибка (MAE)", fontsize=30)
plt.ylabel("Значение", fontsize=20)
plt.xlabel("Наименование модели", fontsize=20)
plt.xticks(rotation=45, fontsize=15)
ax.bar_label(ax.containers[0], fontsize=10)
```

```
[Text(0, 0, '365.03'),
 Text(0, 0, '2.42138'),
 Text(0, 0, '365.86'),
 Text(0, 0, '2.4342'),
 Text(0, 0, '365.955'),
 Text(0, 0, '2.42673'),
 Text(0, 0, '361.279'),
```

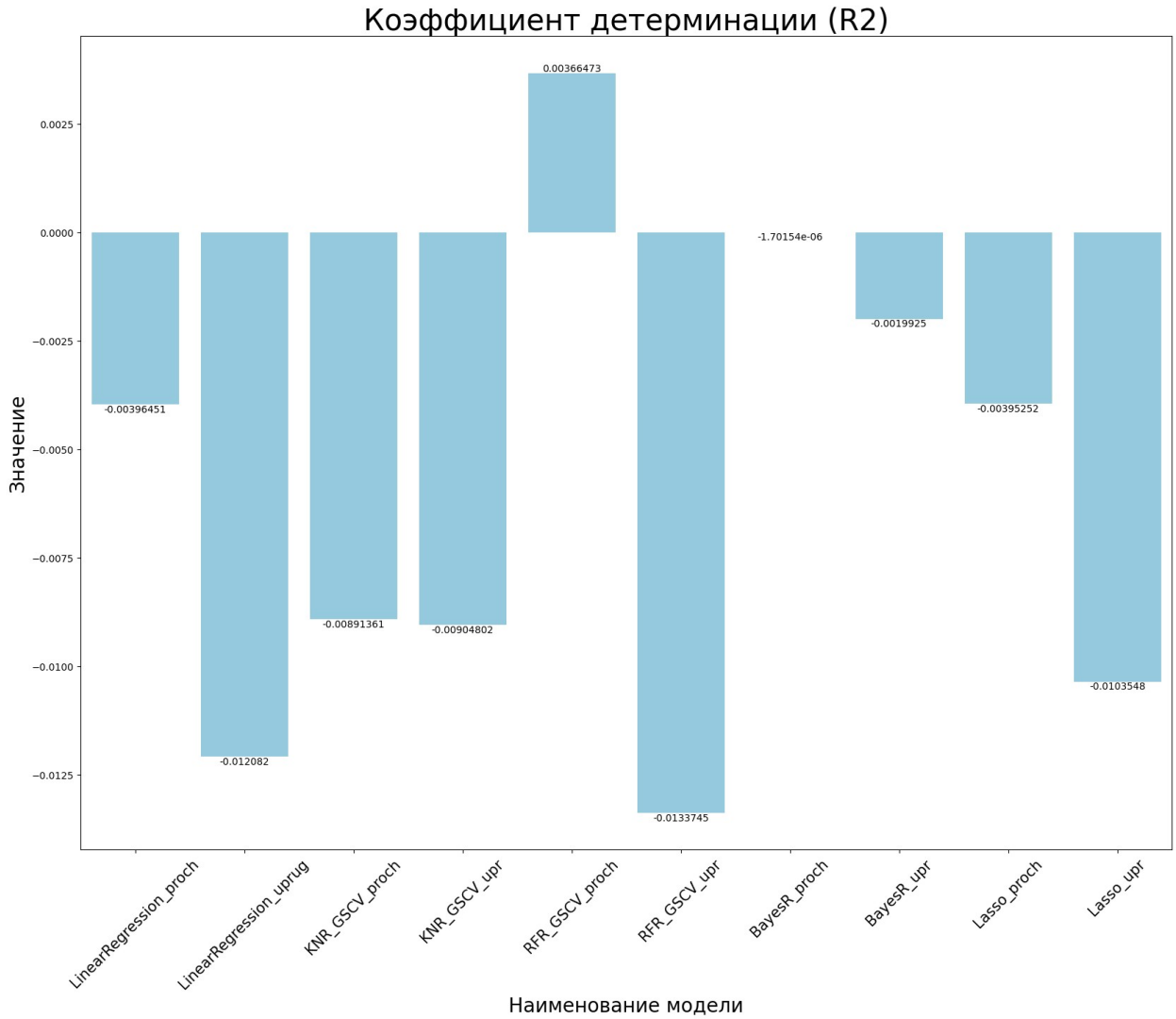
```
 Text(0, 0, '2.41609'),
 Text(0, 0, '365.028'),
 Text(0, 0, '2.42102')]
```



Сравнение моделей. Метрика - "Коэффициент детерминации (R2)"

```
plt.figure(figsize=(20, 15))
ax = sns.barplot(df_sravnenie["R2"], color="skyblue")
plt.title("Коэффициент детерминации (R2)", fontsize=30)
plt.ylabel("Значение", fontsize=20)
plt.xlabel("Наименование модели", fontsize=20)
plt.xticks(rotation=45, fontsize=15)
ax.bar_label(ax.containers[0], fontsize=10)

[Text(0, 0, '-0.00396451'),
 Text(0, 0, '-0.012082'),
```

```
Text(0, 0, '-0.00891361'),
Text(0, 0, '-0.00904802'),
Text(0, 0, '0.00366473'),
Text(0, 0, '-0.0133745'),
Text(0, 0, '-1.70154e-06'),
Text(0, 0, '-0.0019925'),
Text(0, 0, '-0.00395252'),
Text(0, 0, '-0.0103548')]
```

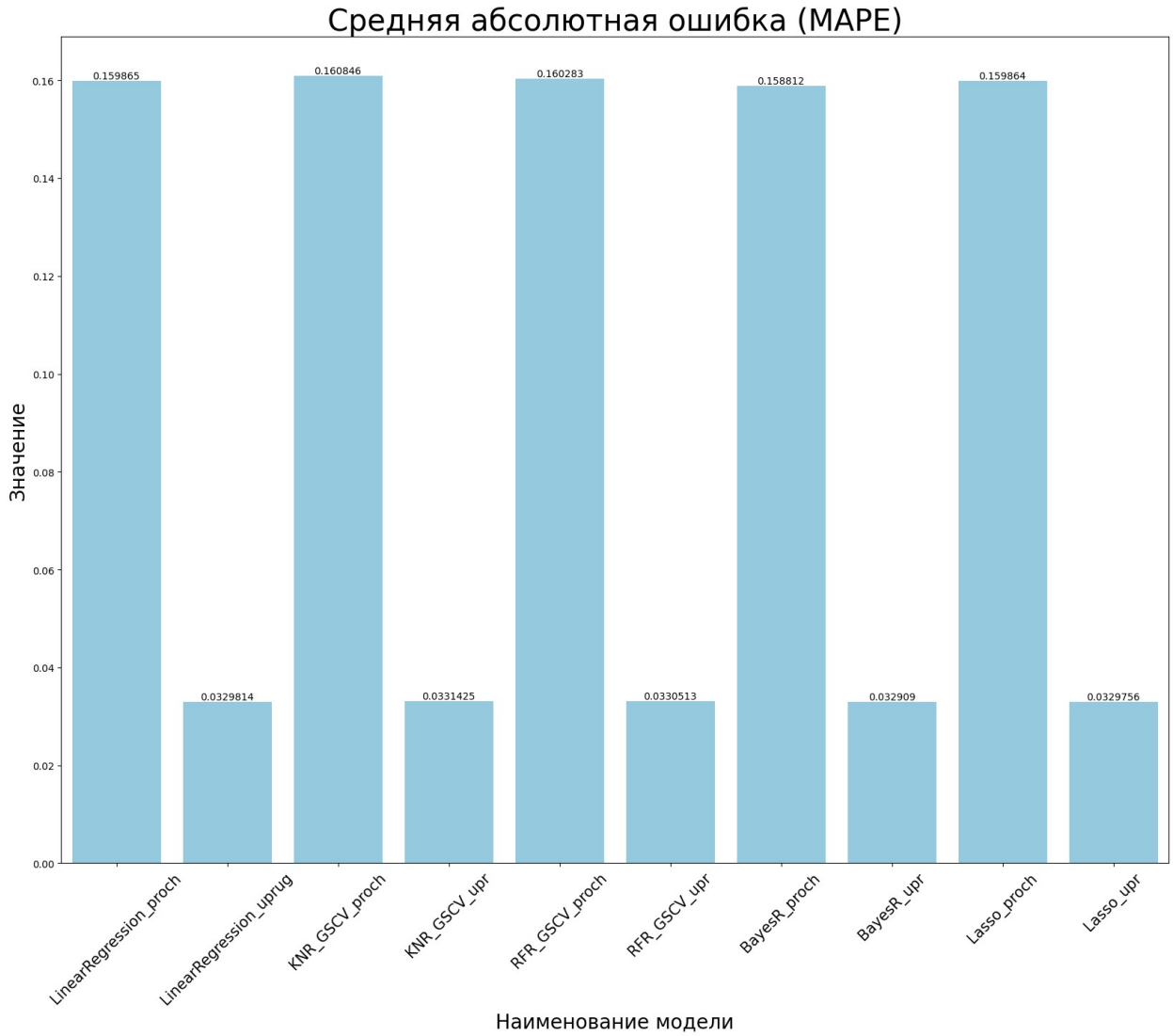


Сравнение моделей. Метрика - "Средняя абсолютная ошибка (MAPE)"

```
plt.figure(figsize=(20, 15))
ax = sns.barplot(df_sravnenie["MAPE"], color="skyblue")
plt.title("Средняя абсолютная ошибка (MAPE)", fontsize=30)
plt.ylabel("Значение", fontsize=20)
plt.xlabel("Наименование модели", fontsize=20)
```

```
plt.xticks(rotation=45, fontsize=15)
ax.bar_label(ax.containers[0], fontsize=10)
```

```
[Text(0, 0, '0.159865'),
 Text(0, 0, '0.0329814'),
 Text(0, 0, '0.160846'),
 Text(0, 0, '0.0331425'),
 Text(0, 0, '0.160283'),
 Text(0, 0, '0.0330513'),
 Text(0, 0, '0.158812'),
 Text(0, 0, '0.032909'),
 Text(0, 0, '0.159864'),
 Text(0, 0, '0.0329756')]
```



Дополнительный тест прогноза

Дополнительно проверим работу Лассо регрессора. Возьмем данные из 308 строки нашего датафрейма (которая в последствии попали в тестовую выборку)

```
df[308:309]
```

```
     Соотношение матрица-наполнитель  Плотность, кг/м3  модуль
упругости, ГПа  \
308                          2.124424       1970.726352
546.816051

     Количество отвердителя, м.%  Поверхностная плотность, г/м2  \
308                   167.744937                      351.029659

     Модуль упругости при растяжении, ГПа  Прочность при растяжении,
МПа  \
308                               74.445891
2723.281123

     Шаг нашивки  Плотность нашивки  Угол нашивки, град_0  \
308     4.894335          42.220404                      1

     Угол нашивки, град_90  Содержание эпоксидных групп, г  \
308                      0                       52.057788

     Потребление смолы, г/м2 без ЭГ
308                      204.054293
```

```
# Вводим нормализованные данные из строки 308 тестового датасета
x = [[0.33084,  0.493907,  0.334887,  0.901624,  0.271493,  0.561482,
      0.354639,  0.236418,  1.0,  0.0,  0.541704,  0.655402]]
# прогнорзируем значение "Модуля упругости при растяжении, ГПа"
print(lasso_regressor.predict(x))
```

```
[73.07251385]
```

Как видим прогнозное значение составило 73.07, а фактическое указанное в тестовом датафрейме 74.45. В данном случае необходимо отметить, что фактическое значение является очень близким по отношению к среднему значению "Модуля упругости при растяжении, ГПа".

```
# Вводим не нормализованные из строки 308 тестового датасета
g =[[2.124424,  1970.726352,    546.816051,     167.744937,
      351.029659,     2723.281123,    4.894335,  42.220404, 1,   0,
      52.057788, 204.054293]]
# прогнорзируем значение "Модуля упругости при растяжении, ГПа"
print(lasso_regressor.predict(g))
```

```
[-2565.92324078]
```

Передача Лассо регрессору не нормализованных данных ведет к некорректному прогнозу, т.к. модель обучалась на нормализованных данных.

Вывод

На основании сравнения можем сделать вывод о том, что в целом все использованные модели машинного обучения дали приблизительно равный результат. Ни одна из моделей не дала достаточно точного проноза для того, чтобы ее можно было рекомендовать к применению. Коэффициент детерминации R2 у всех моделей близок к 0 и указывает что связь между переменными регрессионной модели отсутствует, и вместо нее для оценки значения выходной переменной можно использовать простое среднее ее наблюдаемых значений. Ввиду приблизительно равных прогнозтических значений можно рекомендовать использование Линейной регрессии (т.к. метод более простой, легкий и максимально понятно интерпритируем) либо среднего арифметического значения.

Можно отметить, что на результаты прогнозной способности моделей могли повлиять:

- отсутствие линейной связи / корреляции между переменными (что было выявлено на этапе исследования данных);
- отсутствие нормального распределения у отдельных переменных;
- измерение переменных в разных единицах (проценты, г/м3, м.%, С_2, %_2, г/м2, ГПа, МПа);
- отсутствие четкого понимания как происходил сбор данных и какие переменные могут быть производными или соотноситься между собой;
- не все переменные достаточно объяснены и замерены (например, угол нашивки);
- кроме того, необходимо учитывать, что композитные материалы являются сложными материалами и в результате объединения компонентов их свойства могут изменяться существенным образом.

Таким образом, в целом необходимо проработать и более детально проанализировать датасет, получить обратную связь о том, как были собраны данные, получить экспертное мнение о том, как фактически связаны между собой отдельные переменные (возможно требуется их группировка, изменение единиц измерения).

Дополнительное исследование от 05.07.2025

При повторном рассмотрении итогового датасета было обнаружено, что первые 37 строк датасета существенно отличаются от остальных. В частности многие их значения выражены в виде целых чисел, а иные значения представлены в виде чисел с плавающей точкой. Поэтому делаю предположение, что эти данные могли быть синтезированы (созданы искусственно). С целью проверки данной гипотезы проведем краткое исследование.

```
dop_df = pd.read_excel('db_itog_bez_norm.xlsx')

dop_df.drop('Unnamed: 0', axis=1,inplace=True)

dop_df.iloc[0:38]
```

```
    Соотношение матрица-наполнитель  Плотность, кг/м3  модуль
упругости, ГПа  \
0                          1.857143       2030.000000
738.736842
```

```
1                               1.857143        2030.000000
738.736842
2                               1.857143        2030.000000
738.736842
3                               2.771331        2030.000000
753.000000
4                               2.767918        2000.000000
748.000000
5                               2.569620        1910.000000
807.000000
6                               2.561475        1900.000000
535.000000
7                               3.557018        1930.000000
889.000000
8                               3.532338        2100.000000
1421.000000
9                               2.919678        2160.000000
933.000000
10                              2.877358        1990.000000
1628.000000
11                              1.598174        1950.000000
827.000000
12                              2.919678        1980.000000
568.000000
13                              4.029126        1910.000000
800.000000
14                              2.934783        2030.000000
302.000000
15                              3.557018        1880.000000
313.000000
16                              4.193548        1950.000000
506.000000
17                              4.897959        1890.000000
540.000000
18                              2.877358        2000.000000
205.000000
19                              1.598174        1920.000000
456.000000
20                              4.029126        1880.000000
622.000000
21                              2.587348        1953.274926
1136.596135
22                              2.046471        2037.631811
707.570887
23                              1.856476        2018.220332
836.294382
24                              3.305535        1917.907506
478.286247
25                              2.709554        1892.071124
```

```
641.052549
26                          2.282825       2008.357592
393.967325
27                          1.978140       1973.629097
991.724095
28                          1.771436       1872.491560
801.033883
29                          3.277087       2010.047012
339.550423
30                          2.984362       1912.315437
1183.091845
31                          2.916150       1879.969846
1003.270178
32                          3.247617       1813.234600
757.874479
33                          2.423876       1908.940601
530.228686
34                          5.098993       1977.339047
1572.096042
35                          2.444177       2085.495837
931.310636
36                          2.667697       2078.894676
1542.168458
37                          3.034399       1968.401388
455.871019

    Количество отвердителя, м.%  Поверхностная плотность, г/м2  \
0                     50.000000                     210.000000
1                     49.900000                     210.000000
2                    129.000000                     210.000000
3                    111.860000                     210.000000
4                    111.860000                     210.000000
5                    111.860000                     210.000000
6                    111.860000                     380.000000
7                    129.000000                     380.000000
8                    129.000000                    1010.000000
9                    129.000000                    1010.000000
10                   129.000000                    1010.000000
11                   129.000000                     470.000000
12                   129.000000                     470.000000
13                   129.000000                     470.000000
14                   129.000000                     210.000000
15                   129.000000                     210.000000
16                   129.000000                     380.000000
17                   129.000000                     380.000000
18                   111.860000                    1010.000000
19                   111.860000                     470.000000
20                   111.860000                     470.000000
21                   137.627420                     555.893453
```

22	101.617251	547.601219
23	135.401697	150.961449
24	105.786930	526.692159
25	96.563293	804.592621
26	149.372832	535.371459
27	149.372128	485.453778
28	79.794548	864.929184
29	67.498993	117.535234
30	133.549001	377.389009
31	109.239530	408.354239
32	81.379871	575.062857
33	58.262414	456.908047
34	132.343060	690.364836
35	110.564840	278.230020
36	132.147403	787.299217
37	61.421297	637.376893

	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа
0	70.000000	3000.000000
1	70.000000	3000.000000
2	70.000000	3000.000000
3	70.000000	3000.000000
4	70.000000	3000.000000
5	70.000000	3000.000000
6	75.000000	1800.000000
7	75.000000	1800.000000
8	78.000000	2000.000000
9	78.000000	2000.000000
10	78.000000	2000.000000
11	73.333333	2455.555556
12	73.333333	2455.555556
13	73.333333	2455.555556
14	70.000000	3000.000000

```
15                                  70.000000
3000.000000
16                                  75.000000
1800.000000
17                                  75.000000
1800.000000
18                                  78.000000
2000.000000
19                                  73.333333
2455.555556
20                                  73.333333
2455.555556
21                                  80.803222
2587.342983
22                                  73.817067
2624.026407
23                                  77.210762
2473.187195
24                                  72.345709
3059.032991
25                                  74.511359
2288.967377
26                                  72.244924
2704.445081
27                                  75.665701
2448.943079
28                                  70.947592
2796.785402
29                                  67.478707
2462.605386
30                                  75.290452
2303.770656
31                                  71.700856
3086.546196
32                                  69.341133
3188.136358
33                                  74.244354
1890.505807
34                                  72.341640
1386.578973
35                                  71.479060
2740.229631
36                                  76.471788
2559.643047
37                                  75.090372
2848.490078

    Шаг нашивки  Плотность нашивки  Угол нашивки, град_0  \
0      4.000000          60.000000                     1
```

1	4.000000	70.000000	1
2	5.000000	47.000000	1
3	5.000000	57.000000	1
4	5.000000	60.000000	1
5	5.000000	70.000000	1
6	7.000000	47.000000	1
7	7.000000	57.000000	1
8	7.000000	60.000000	1
9	7.000000	70.000000	1
10	9.000000	47.000000	1
11	9.000000	57.000000	1
12	9.000000	60.000000	1
13	9.000000	70.000000	1
14	10.000000	47.000000	1
15	10.000000	57.000000	1
16	10.000000	60.000000	1
17	10.000000	70.000000	1
18	4.000000	47.000000	0
19	4.000000	57.000000	0
20	4.000000	60.000000	0
21	4.000000	70.000000	0
22	5.000000	57.000000	0
23	5.000000	60.000000	0
24	5.000000	70.000000	0
25	7.000000	47.000000	0
26	7.000000	57.000000	0
27	7.000000	60.000000	0
28	7.000000	70.000000	0
29	9.000000	47.000000	0
30	9.000000	57.000000	0
31	9.000000	60.000000	0
32	9.000000	70.000000	0
33	10.000000	47.000000	0
34	10.000000	57.000000	0
35	10.000000	60.000000	0
36	10.000000	70.000000	0
37	7.856167	64.301964	1

	Угол нашивки, град_90	Содержание эпоксидных групп, г
0	0	52.250000
1	0	72.600000
2	0	46.750000
3	0	48.989286
4	0	48.989286
5	0	48.989286
6	0	26.721429
7	0	25.500000
8	0	63.750000
9	0	63.750000

10	0	63.750000
11	0	46.750000
12	0	46.750000
13	0	46.750000
14	0	46.750000
15	0	46.750000
16	0	25.500000
17	0	25.500000
18	1	66.803571
19	1	48.989286
20	1	48.989286
21	1	55.104551
22	1	41.246538
23	1	32.606770
24	1	49.256708
25	1	29.154177
26	1	56.553864
27	1	32.093444
28	1	27.503887
29	1	50.265372
30	1	46.662581
31	1	49.359994
32	1	59.228521
33	1	53.611671
34	1	69.054800
35	1	44.123263
36	1	37.124158
37	0	73.068457

	Потребление смолы, г/м2 без ЭГ
0	167.750000
1	147.400000
2	173.250000
3	171.010714
4	171.010714
5	171.010714
6	93.278571
7	94.500000
8	236.250000
9	236.250000
10	236.250000
11	173.250000
12	173.250000
13	173.250000
14	173.250000
15	173.250000
16	94.500000
17	94.500000
18	233.196429

```
19                          171.010714
20                          171.010714
21                          191.508566
22                          136.952018
23                           90.737792
24                          226.319171
25                           97.662162
26                          204.523207
27                          130.400250
28                           95.852377
29                          156.753209
30                          153.917668
31                          142.831168
32                          193.642048
33                          169.087817
34                          202.846594
35                          143.738110
36                          126.778620
37                          237.983941
```

Исключаем искуственные столбцы.

```
list = range(0,37,1)
dop_df.drop([*list], inplace=True)

dop_df.head()
```

```
    Соотношение матрица-наполнитель  Плотность, кг/м3  модуль
упругости, ГПа  \
37                          3.034399       1968.401388
455.871019
38                          2.664389       1996.159145
525.057774
39                          1.193530       1965.929227
899.603701
40                          2.914333       2049.373404
382.263359
41                          4.315666       1913.379677
822.918735

    Количество отвердителя, м.%  Поверхностная плотность, г/м2  \
37                     61.421297                     637.376893
38                     77.506883                      28.658102
39                    102.959069                     871.088955
40                     81.352047                     561.992131
41                    143.576937                     260.859341

    Модуль упругости при растяжении, ГПа  Прочность при растяжении,
МПа  \
37                              75.090372
```

```
2848.490078
38                              69.489773
2220.587445
39                              73.454695
2335.541792
40                              69.814615
2262.784366
41                              75.957329
1639.912525

    Шаг нашивки  Плотность нашивки  Угол нашивки, град_0  \
37     7.856167          64.301964                     1
38     6.675780          78.623299                     1
39     7.526398          38.176975                     1
40     8.325699          46.045428                     1
41     7.656211          33.571024                     1

    Угол нашивки, град_90  Содержание эпоксидных групп, г  \
37                      0                       73.068457
38                      0                       57.056756
39                      0                       17.815035
40                      0                       49.678900
41                      0                       60.262771

    Потребление смолы, г/м2 без ЭГ
37                       237.983941
38                       257.719913
39                        73.232612
40                       253.396553
41                       187.981559

dop_df.shape

(888, 13)
```

Подготавливаем датафреймы

```
Xd_proch = dop_df.drop("Прочность при растяжении, МПа", axis=1)
yd_proch = dop_df['Прочность при растяжении, МПа']

Xd_mod_uprug = dop_df.drop("Модуль упругости при растяжении, ГПа",
axis=1)
yd_mod_uprug = dop_df["Модуль упругости при растяжении, ГПа"]
```

Нормализация

```
mms_dproch = MinMaxScaler()
col = Xd_proch.columns
result = mms_dproch.fit_transform(Xd_proch)
```

```
Xd_proch = pd.DataFrame(result, columns = col)
Xd_proch.describe()
```

```
       Соотношение матрица-наполнитель  Плотность, кг/м3  \
count                        888.000000        888.000000
mean                           0.498170          0.504063
std                            0.188600          0.188239
min                            0.000000          0.000000
25%                            0.370991          0.370092
50%                            0.494913          0.511692
75%                            0.629517          0.625981
max                            1.000000          1.000000

       модуль упругости, ГПа  Количество отвердителя, м.%  \
count             888.000000                   888.000000
mean                0.453269                     0.505054
std                 0.202600                     0.187826
min                 0.000000                     0.000000
25%                 0.306972                     0.376190
50%                 0.453027                     0.501726
75%                 0.591871                     0.640147
max                 1.000000                     1.000000

       Поверхностная плотность, г/м2  Модуль упругости при растяжении,
ГПа  \
count                     888.000000
888.000000
mean                        0.374192
0.487271
std                         0.216893
0.195308
min                         0.000000
0.000000
25%                         0.206504
0.357440
50%                         0.356181
0.480836
75%                         0.538857
0.617610
max                         1.000000
1.000000

       Шаг нашивки  Плотность нашивки  Угол нашивки, град_0  \
count   888.000000         888.000000            888.000000
mean      0.501530           0.502884              0.486486
std       0.184373           0.194733              0.500099
min       0.000000           0.000000              0.000000
25%       0.373371           0.376780              0.000000
50%       0.502075           0.504310              0.000000
75%       0.622800           0.630352              1.000000
```

```
max       1.000000           1.000000              1.000000

       Угол нашивки, град_90  Содержание эпоксидных групп, г  \
count             888.000000                      888.000000
mean                0.513514                        0.487567
std                 0.500099                        0.190908
min                 0.000000                        0.000000
25%                 0.000000                        0.347834
50%                 1.000000                        0.486314
75%                 1.000000                        0.618045
max                 1.000000                        1.000000

       Потребление смолы, г/м2 без ЭГ
count                      888.000000
mean                         0.506469
std                          0.190361
min                          0.000000
25%                          0.380550
50%                          0.503554
75%                          0.640242
max                          1.000000

mms_dmod_uprug = MinMaxScaler()
col = Xd_mod_uprug.columns
result = mms_dmod_uprug.fit_transform(Xd_mod_uprug)

Xd_mod_uprug = pd.DataFrame(result, columns = col)
Xd_mod_uprug.describe()

       Соотношение матрица-наполнитель  Плотность, кг/м3  \
count                       888.000000        888.000000
mean                          0.498170          0.504063
std                           0.188600          0.188239
min                           0.000000          0.000000
25%                           0.370991          0.370092
50%                           0.494913          0.511692
75%                           0.629517          0.625981
max                           1.000000          1.000000

       модуль упругости, ГПа  Количество отвердителя, м.%  \
count             888.000000                    888.000000
mean                0.453269                      0.505054
std                 0.202600                      0.187826
min                 0.000000                      0.000000
25%                 0.306972                      0.376190
50%                 0.453027                      0.501726
75%                 0.591871                      0.640147
max                 1.000000                      1.000000

       Поверхностная плотность, г/м2  Прочность при растяжении, МПа  \
```

```
count                       888.000000                       888.000000
mean                          0.374192                         0.506406
std                           0.216893                         0.188674
min                           0.000000                         0.000000
25%                           0.206504                         0.377215
50%                           0.356181                         0.505343
75%                           0.538857                         0.627759
max                           1.000000                         1.000000
```

```
       Шаг нашивки  Плотность нашивки  Угол нашивки, град_0  \
count   888.000000          888.000000            888.000000
mean      0.501530            0.502884              0.486486
std       0.184373            0.194733              0.500099
min       0.000000            0.000000              0.000000
25%       0.373371            0.376780              0.000000
50%       0.502075            0.504310              0.000000
75%       0.622800            0.630352              1.000000
max       1.000000            1.000000              1.000000
```

```
       Угол нашивки, град_90  Содержание эпоксидных групп, г  \
count             888.000000                      888.000000
mean                0.513514                        0.487567
std                 0.500099                        0.190908
min                 0.000000                        0.000000
25%                 0.000000                        0.347834
50%                 1.000000                        0.486314
75%                 1.000000                        0.618045
max                 1.000000                        1.000000
```

```
       Потребление смолы, г/м2 без ЭГ
count                      888.000000
mean                         0.506469
std                          0.190361
min                          0.000000
25%                          0.380550
50%                          0.503554
75%                          0.640242
max                          1.000000
```

```
Xd_train_proch, Xd_test_proch, yd_train_proch, yd_test_proch =
train_test_split(Xd_proch, yd_proch, test_size=0.3, random_state=0,
shuffle=True)

Xd_train_mod_uprug, Xd_test_mod_uprug, yd_train_mod_uprug,
yd_test_mod_uprug = train_test_split(Xd_mod_uprug, yd_mod_uprug,
test_size=0.3, random_state=0, shuffle=True)
```

Линейная регрессия

```
# При построении модели пользуемся библиотекой sklearn.linear_model
regressor = LinearRegression()
regressor.fit(Xd_train_proch, yd_train_proch)
yd_pred_proch = regressor.predict(Xd_test_proch)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_proch, yd_pred_proch)))
print("MAE:", metrics.mean_absolute_error(yd_test_proch,
yd_pred_proch))
print("R2", metrics.r2_score(yd_test_proch, yd_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch))
print("Коэффициенты регрессии:", regressor.coef_)
print("Константа:", regressor.intercept_)


# Создаем датафрейм в который занесем результаты тестирования каждой
модели
# Данный датафрейм используем при итоговом сравнении моделей
dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_proch,
yd_pred_proch))],
    [metrics.mean_absolute_error(yd_test_proch, yd_pred_proch)],
    [metrics.r2_score(yd_test_proch, yd_pred_proch)],
    [metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch)],
]

metrics_index = ['RMSE','MAE','R2','MAPE']
dop_df_sravnenie = pd.DataFrame(data=dop_data_sravn,
index=metrics_index)
dop_df_sravnenie.rename(columns={0:'LinearRegression_dproch'},inplace=
True)
```

```
Корень из среднеквадратичной ошибки: 465.89100157687966
MAE: 378.4267858184801
R2 -0.011597094210259229
MAPE: 0.16662095058868923
Коэффициенты регрессии: [  13.71718046 -263.65716472   18.10416732 -
251.41969885   48.08560556
   -5.72416939 -124.43391044   47.19799526  -12.59643546   12.59643546
 -130.99185478   49.9816475 ]
Константа: 2771.8628811628037
```

```
# При построении модели пользуемся библиотекой sklearn.linear_model
regressor = LinearRegression()
regressor.fit(Xd_train_mod_uprug, yd_train_mod_uprug)
yd_pred_mod_uprug = regressor.predict(Xd_test_mod_uprug)
```

```
# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug))
print("R2", metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug))
print("Коэффициенты регрессии:", regressor.coef_)
print("Константа:", regressor.intercept_)

dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug))],
    [metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
    [metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
]

dop_df_sravnenie["LinearRegression_dupr"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 3.005482159995833
MAE: 2.454921985418782
R2 -0.0220329833757702
MAPE: 0.03361677717653914
Коэффициенты регрессии: [-0.30318348 -0.02278777  0.18222356 -
1.86900664  0.70899896 -0.04117522
 -0.1139297   0.26505708 -0.0629704   0.0629704   1.66253719 -
0.48187011]
Константа: 73.44659197911224
```

KNeighborsRegressor

```
knn = KNeighborsRegressor()

knn.fit(Xd_train_proch, yd_train_proch)
yd_pred_proch = knn.predict(Xd_test_proch)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_proch, yd_pred_proch)))
print("MAE:", metrics.mean_absolute_error(yd_test_proch,
yd_pred_proch))
print("R2", metrics.r2_score(yd_test_proch, yd_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch))
```

```
dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_proch,
yd_pred_proch))],
    [metrics.mean_absolute_error(yd_test_proch, yd_pred_proch)],
    [metrics.r2_score(yd_test_proch, yd_pred_proch)],
    [metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch)],
]

dop_df_sravnenie["KNN_dproch"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 508.5981407078656
MAE: 406.4759811484183
R2 -0.20555898669111516
MAPE: 0.17889904674604915
```

```
# при построении модели используем библиотеку sklearn
knn = KNeighborsRegressor()

knn.fit(Xd_train_mod_uprug, yd_train_mod_uprug)
yd_pred_mod_uprug = knn.predict(Xd_test_mod_uprug)

print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug))
print("R2", metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug))

dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug))],
    [metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
    [metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
]

dop_df_sravnenie["KNN_dupr"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 3.2812801367352455
MAE: 2.672771238948065
R2 -0.21821297816752416
MAPE: 0.03666331242314573
```

Случайный лес

```
# строим модель регрессии RandomForestRegressor с использованием
библиотеки sklearn
rfr = RandomForestRegressor(max_depth=2, random_state=0)
rfr.fit(Xd_train_proch, yd_train_proch)
yd_pred_proch = rfr.predict(Xd_test_proch)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_proch, yd_pred_proch)))
print("MAE:", metrics.mean_absolute_error(yd_test_proch,
yd_pred_proch))
print("R2", metrics.r2_score(yd_test_proch, yd_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch))

dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_proch,
yd_pred_proch))],
    [metrics.mean_absolute_error(yd_test_proch, yd_pred_proch)],
    [metrics.r2_score(yd_test_proch, yd_pred_proch)],
    [metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch)],
]

dop_df_sravnenie["RFR_dproch"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 464.21536261377867
MAE: 377.9401589360584
R2 -0.004333492959275764
MAPE: 0.16678429721523605
```

```
# строим модель регрессии RandomForestRegressor с использованием
библиотеки sklearn
rfr = RandomForestRegressor(max_depth=2, random_state=0)
rfr.fit(Xd_train_mod_uprug, yd_train_mod_uprug)
yd_pred_mod_uprug = rfr.predict(Xd_test_mod_uprug)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug))
print("R2", metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug))

dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
```

```
yd_pred_mod_uprug))],
    [metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
    [metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
]

dop_df_sravnenie["RFR_dupr"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 3.009875405663134
MAE: 2.45561371092279
R2 -0.02502306841565849
MAPE: 0.03363232206077254
```

```
# Строим модель Байесовской гребней регрессии (BayesianRidge) с
помощью библиотеки sklearn
bsr = BayesianRidge()
bsr.fit(Xd_train_proch, yd_train_proch)
yd_pred_proch = bsr.predict(Xd_test_proch)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_proch, yd_pred_proch)))
print("MAE:", metrics.mean_absolute_error(yd_test_proch,
yd_pred_proch))
print("R2", metrics.r2_score(yd_test_proch, yd_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch))


dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_proch,
yd_pred_proch))],
    [metrics.mean_absolute_error(yd_test_proch, yd_pred_proch)],
    [metrics.r2_score(yd_test_proch, yd_pred_proch)],
    [metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch)],
]

dop_df_sravnenie["BR_dproch"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 463.47835099767036
MAE: 376.11626491686496
R2 -0.0011469639615064864
MAPE: 0.16564304006411398
```

```
# Строим модель Байесовской гребней регрессии (BayesianRidge) с
помощью библиотеки sklearn
bsr = BayesianRidge()
bsr.fit(Xd_train_mod_uprug, yd_train_mod_uprug)
```

```
yd_pred_mod_uprug = bsr.predict(Xd_test_mod_uprug)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug))
print("R2", metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug))

dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug))],
    [metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
    [metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
]

dop_df_sravnenie["BR_dupr"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 2.972967065879132
MAE: 2.427031660589892
R2 -3.8682412638824104e-05
MAPE: 0.033241733087847225
```

```
lasso_regressor = Lasso(alpha=0.001)
lasso_regressor.fit(Xd_train_proch, yd_train_proch)
yd_pred_proch = lasso_regressor.predict(Xd_test_proch)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_proch, yd_pred_proch)))
print("MAE:", metrics.mean_absolute_error(yd_test_proch,
yd_pred_proch))
print("R2", metrics.r2_score(yd_test_proch, yd_pred_proch))
print("MAPE:", metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch))

dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_proch,
yd_pred_proch))],
    [metrics.mean_absolute_error(yd_test_proch, yd_pred_proch)],
    [metrics.r2_score(yd_test_proch, yd_pred_proch)],
    [metrics.mean_absolute_percentage_error(yd_test_proch,
yd_pred_proch)],
```

```
]

dop_df_sravnenie["LASSO_dproch"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 465.88999555712246
MAE: 378.42372266165194
R2 -0.01159272543919121
MAPE: 0.16661971921007115
```

```
# Для построения модели используем библиотеку sklearn
lasso_regressor = Lasso(alpha=0.001)
lasso_regressor.fit(Xd_train_mod_uprug, yd_train_mod_uprug)
yd_pred_mod_uprug = lasso_regressor.predict(Xd_test_mod_uprug)

# Выводим значения метрик
print("Корень из среднеквадратичной ошибки:",
np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug)))
print("MAE:", metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug))
print("R2", metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug))
print("MAPE:",
metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug))


dop_data_sravn = [
    [np.sqrt(metrics.mean_squared_error(yd_test_mod_uprug,
yd_pred_mod_uprug))],
    [metrics.mean_absolute_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
    [metrics.r2_score(yd_test_mod_uprug, yd_pred_mod_uprug)],
    [metrics.mean_absolute_percentage_error(yd_test_mod_uprug,
yd_pred_mod_uprug)],
]

dop_df_sravnenie["LASSO_dupr"] = np.array(dop_data_sravn)
```

```
Корень из среднеквадратичной ошибки: 3.0048583052335553
MAE: 2.454836401012962
R2 -0.021608735993227768
MAPE: 0.03361806320404101
```

```
dop_df_sravnenie = df_sravnenie.transpose()

dop_df_sravnenie
```

```
      LinearRegression_proch  LinearRegression_uprug
KNR_GSCV_proch  \
RMSE              451.342041                2.986901      452.453133
```

```
MAE               365.029937                2.421378      365.860100

R2                 -0.003965               -0.012082       -0.008914

MAPE                0.159865                0.032981        0.160846


      KNR_GSCV_upr  RFR_GSCV_proch  RFR_GSCV_upr  BayesR_proch
BayesR_upr  \
RMSE      2.982421      449.623872      2.988808    450.450401
2.971975
MAE       2.434199      365.954796      2.426734    361.279222
2.416087
R2       -0.009048        0.003665     -0.013374     -0.000002   -
0.001993
MAPE      0.033143        0.160283      0.033051      0.158812
0.032909

      Lasso_proch  Lasso_upr
RMSE   451.339347   2.984351
MAE    365.027883   2.421021
R2      -0.003953  -0.010355
MAPE     0.159864   0.032976
```

Существенного улучшения прогноза у моделей не появилось. В данном случае можно предположить, что гипотеза о том, что данные были синтезированы не верна, либо на предсказательную способность существенно повлияли изменения в данные связанные с исключением, добавлением или изменением переменных.