

Итоговый проект: Процесс электронно-лучевой сварки

Курс: Аналитик данных (Data scientist)

МГТУ им. Н.Э. Баумана

Студент: Григорев Александр

2 поток: 04.10-20.12.2022

Оглавление

- 1 Описание задачи
- 2 Подключение библиотек и загрузка данных
- 3 Исследование данных
- 4 Предобработка
- 5 Разделение данных на выборки
- 6 Тестирование моделей
 - 6.1 Модель линейной регрессии
 - 6.2 Ridge
 - 6.3 Дерево решений
 - 6.4 Случайный лес
 - 6.5 Случайный лес Extra
 - 6.6 Градиентный бустинг
 - 6.7 Модель полносвязной НС
 - 6.8 Проверка на адекватность
 - 6.9 Выводы по моделям
- 7 Создание итоговой модели
 - 7.1 Экспорт модели
- 8 Выводы

Описание задачи

В качестве исходных данных были взяты результаты экспериментальных исследований, проводимых в целях улучшения технологического процесса электронно-лучевой сварки изделия, сборка которого состоит из элементов, состоящих из разнородного материала.

Установка электронно-лучевой сварки, на которой проводились исследования, предназначена для сварки электронным лучом в глубоком вакууме деталей сборочных единиц из нержавеющей сталей, титановых, алюминиевых и специальных сплавов.

Существующая установка электронно-лучевой сварки обеспечивает повторяемость режимов в рамках возможностей реализованной системы управления. Работы по сварке выполнялись на образцах-имитаторах, соответствующих технологическому изделию. Для уменьшения вложения энергии при сварке:

1. Снижалась величина сварочного тока **IW** ;
2. Увеличивался ток фокусировки электронного пучка **IF** ;
3. Увеличивалась скорость сварки **VW** ;
4. Менялось расстояние от поверхности образцов до электронно-оптической системы **FP** .

По совокупности параметров технологических режимов обеспечивались минимально возможные размеры сварных швов: глубина шва **Depth** и ширина шва **Width** .

В процессе выполнения работ была произведена электронно-лучевая сварка 18-ти единиц образцов. Результаты металлографического контроля по размерам сварочного шва для каждого образца проводились в 4-х поперечных сечениях сварочного шва. Ускоряющее напряжение было постоянным в диапазоне 19,8 – 20 кВ. Набор полученных данных собраны в составе режимов сварки, размеров сварочных швов в поперечных сечениях всех образцов. Статистические показатели набора обучающих данных указаны в табл. 1.

Таблица 1

Статистические показатели набора обучающих данных

Показатель	IW	IF	VW	FP	Depth	Width
Количество	72	72	72	72	72	72
Среднее выборочное	45,666	141,333	8,639	78,333	1,196	1,970
Среднее квадратичное отклонение	1,678	5,146	2,061	21,494	0,225	0,279
Минимум	43	131	4,5	50	0,80	1,68
25%	44	139	8	60	1,08	1,76
50%	45	141	9	80	1,20	1,84
75%	47	146	10	80	1,29	2,05
Максимум	49	150	12	125	1,76	2,60

Задача:

Провести прогнозирование глубины **Depth** и ширины **Width** сварного шва в зависимости от параметров технологического процесса **IW** , **IF** , **VW** , **FP** .

Т.к. прогнозируемые значения **Depth** и **Width** представляют собой непрерывные случайные величины, то для предсказания данных величин будем использовать модели регрессии.

Для оценки качества моделей будем использовать две метрики из пакет sklearn:

- стандартная для задач регрессии, метрика R2,

- метрика RMSE.

Наша задача получить наименьшее значение метрики RMSE и наиболее близкое к "1" значение метрики R2. Таким образом, мы сможем с определенной точностью прогнозировать значения параметров сварного шва исходя из заданных параметров технологического процесса.

Порядок выполнения работы:

1. Загрузим данные.
2. Выполним анализ данных.
3. Выполним предобработку данных.
4. Раздели данные на обучающую и тестовую выборки.
5. Опробуем несколько моделей и выберем лучшую.
6. Выполним обучение выбранной модели на полной выборке методом кросс-валидации и подбором гиперпараметров.
7. Сохраним полученную модель для использования в приложении прогнозирования параметров сварного шва.
8. Сделаем выводы.

Подключение библиотек и загрузка данных

```
In [1]: #!pip install -U imbalanced-learn
```

```
In [2]: # common
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style='whitegrid')

# ML
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, Normalizer
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import plot_tree

# TF.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.metrics import RootMeanSquaredError, LogCoshError

#models export
import pickle

# const
RND = 33333
```

```
In [3]: # общие процедуры и функции

# комбинированный график для визуализации плотности значений столбца
def value_density_plot(df_column, mean_line_height=1, figsize=(5, 5),
                       title='График плотности значений', xlabel=''):

    median = df_column.median()
    avg = df_column.mean()
    left = df_column.min() * 0.85
    right = df_column.max() * 1.15
    # гистограмма и плотность
    plt.figure(figsize=figsize)
    plt.subplot(2, 1, 1)
    plt.title(title)
    plt.xlim(left, right)
    plt.xticks([])
    plt.plot([median, median], [0, mean_line_height], 'r--')
    plt.plot([avg, avg], [0, mean_line_height], 'g--')
    plt.legend(['медиана', 'среднее'])
    df_column.plot(kind='hist', density=True)
    df_column.plot(kind='kde',)
    plt.ylabel('Плотность')
    # ящик с усами
    plt.subplot(2, 1, 2)
    plt.xlim(left, right)
    plt.xlabel(xlabel)
    plt.boxplot(df_column, vert=False)
    plt.show()
```

```
In [4]: # читаем данные из файла в датасет
df = pd.read_csv('ebw_data.csv')
```

Исследование данных

```
In [5]: # оценим данные
df
```

```
Out[5]:
```

	IW	IF	VW	FP	Depth	Width
0	47	139	4.5	80	1.60	2.54
1	47	139	4.5	80	1.62	2.50
2	47	139	4.5	80	1.68	2.60
3	47	139	4.5	80	1.58	2.52
4	45	140	4.5	80	1.76	2.48
...
67	44	146	9.0	60	1.20	1.72
68	45	146	9.0	60	1.36	1.76
69	45	146	9.0	60	1.28	1.76
70	45	146	9.0	60	1.28	1.76
71	45	146	9.0	60	1.32	1.76

72 rows × 6 columns

```
In [6]: # общая информация
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72 entries, 0 to 71
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0    IW      72 non-null      int64
 1    IF      72 non-null      int64
 2    VW      72 non-null      float64
 3    FP      72 non-null      int64
 4   Depth  72 non-null      float64
 5   Width  72 non-null      float64
dtypes: float64(3), int64(3)
memory usage: 3.5 KB
```

```
In [7]: # проверим данные на дубли
df.duplicated().sum()
```

```
Out[7]: 7
```

```
In [8]: # посмотрим дубли
df[df.duplicated(keep=False)].sort_values('IW')
```

```
Out[8]:
```

	IW	IF	VW	FP	Depth	Width
53	43	150	9.0	50	1.08	1.82
54	43	150	9.0	50	1.08	1.82
55	43	150	9.0	50	1.08	1.82
56	44	146	9.0	60	1.20	1.76
59	44	146	9.0	60	1.20	1.76
9	45	140	8.0	80	1.20	1.96
11	45	140	8.0	80	1.20	1.96
69	45	146	9.0	60	1.28	1.76
70	45	146	9.0	60	1.28	1.76
45	46	146	10.0	60	1.36	1.76
46	46	146	10.0	60	1.36	1.76
29	47	139	4.5	80	1.36	2.48
31	47	139	4.5	80	1.36	2.48

Данные содержат всего 72 строки, в каждой по 6 признаков. Все данные имеют числовой тип, из них:

- целочисленные признаки: IW , IF , FP ;
- признаки с плавающей точкой: FP , Depth , Width ;

Выборка для обучения модели достаточно маленькая.

Пропуски в данных отсутствуют, но имеются дубли.

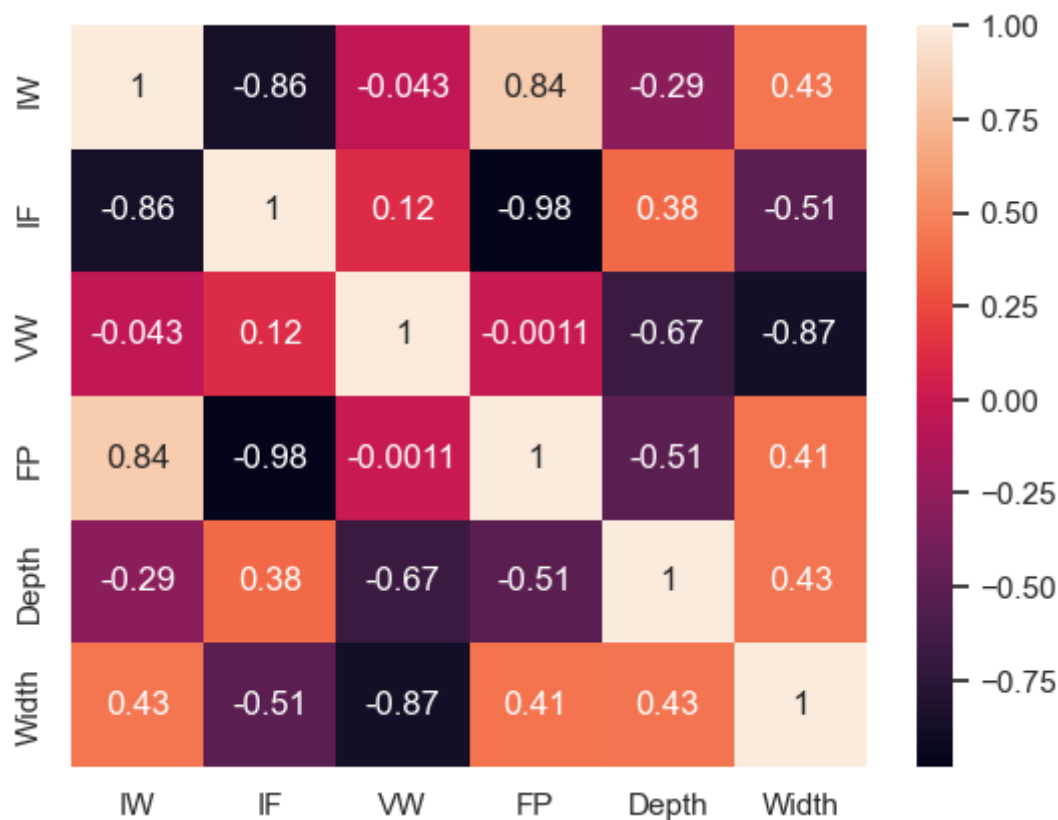
```
In [9]: # проверим корреляцию в данных
```

```
df_corr = df.corr()
df_corr
```

Out[9]:

	IW	IF	VW	FP	Depth	Width
IW	1.000000	-0.861073	-0.043430	0.835530	-0.289568	0.434869
IF	-0.861073	1.000000	0.115093	-0.980562	0.376084	-0.510167
VW	-0.043430	0.115093	1.000000	-0.001060	-0.671437	-0.874257
FP	0.835530	-0.980562	-0.001060	1.000000	-0.510748	0.412962
Depth	-0.289568	0.376084	-0.671437	-0.510748	1.000000	0.425391
Width	0.434869	-0.510167	-0.874257	0.412962	0.425391	1.000000

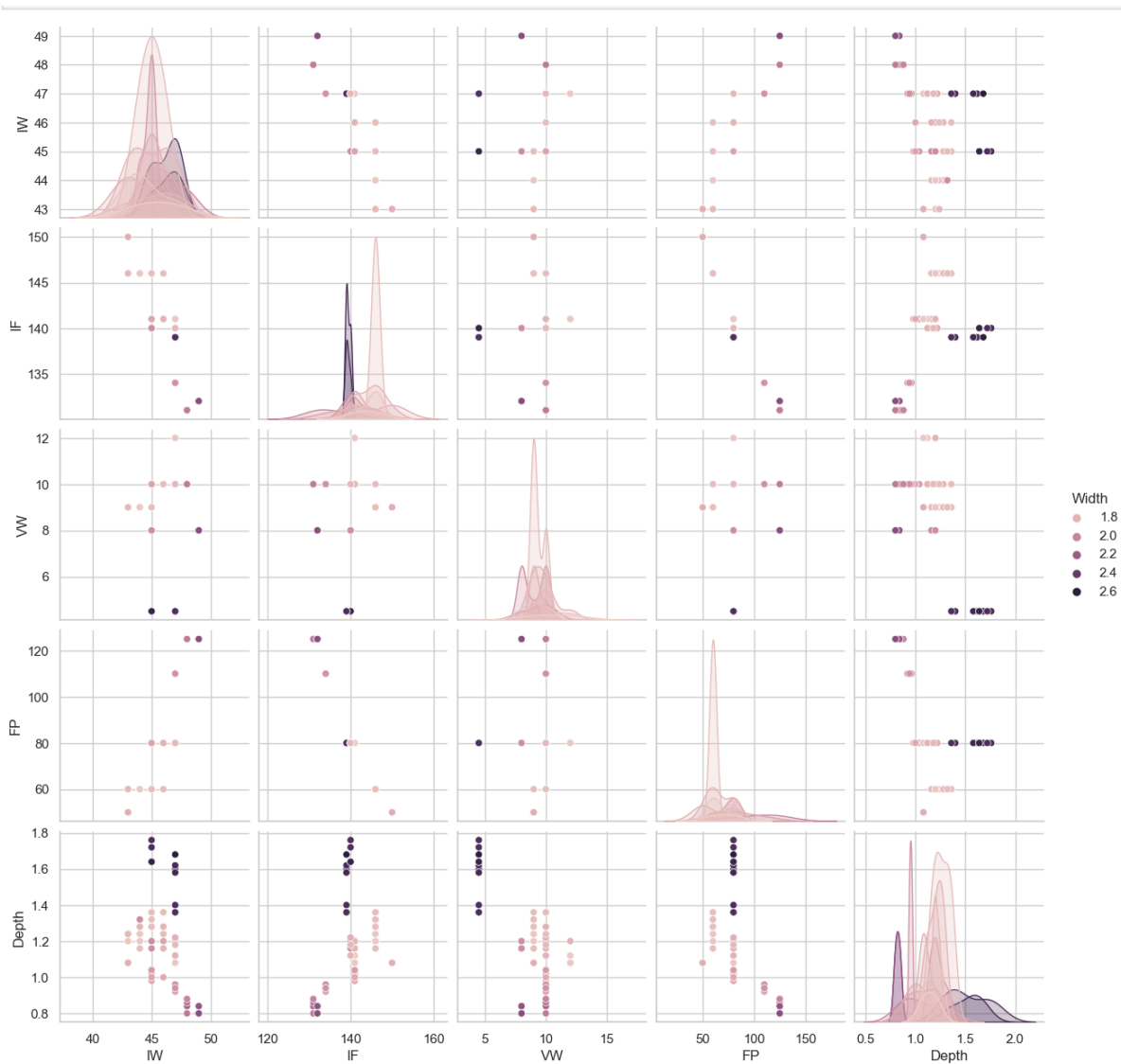
```
In [10]: # построим температурную карту корреляции
sns.heatmap(df_corr, annot=True);
```



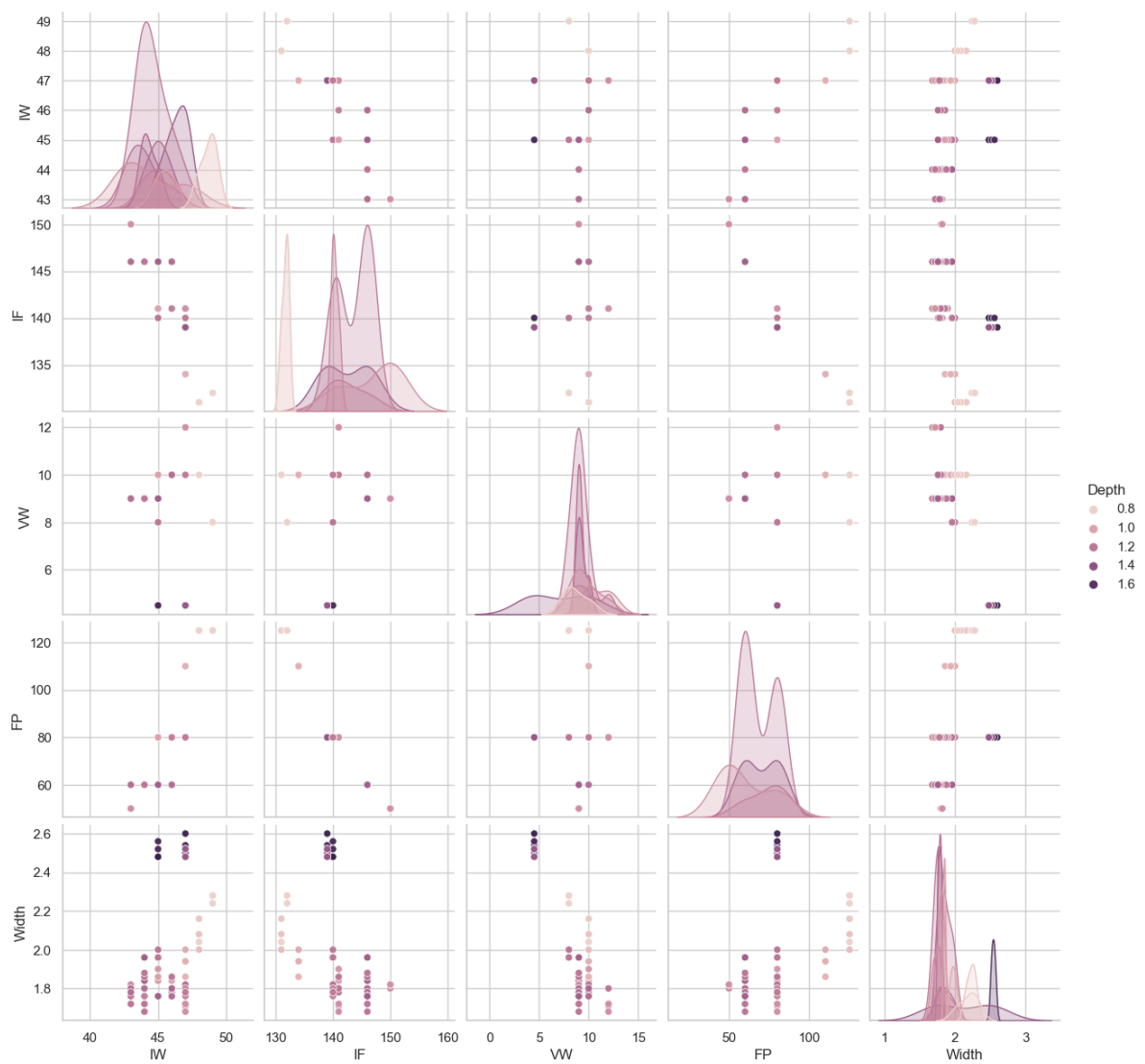
Для показателей ширины **Width** и глубины **Depth** шва, видим очевидную отрицательную корреляцию с увеличением скорости сварки **VW**. Больше выражена данная зависимость относительно ширины шва. Т.е. данную зависимость можно интерпретировать так: *при увеличении скорости сварки, ширина и глубина сварки уменьшаются и наоборот.*

Наблюдаются очень высокая отрицательная корреляция между увеличением тока фокусировки электронного пучка **IF** и изменением расстояния от поверхности образцов до электронно-оптической системы **FP** и снижением величины сварочного тока **IW**. Т.е. *при увеличении тока фокусировки, расстояние до поверхности образцов и величина сварочного тока снижаются.*

```
In [11]: # диаграммы рассеяния в зависимости от ширины шва
sns.pairplot(df, hue='Width');
```



```
In [12]: # диаграммы рассеяния в зависимости от глубины шва
sns.pairplot(df, hue='Depth');
```



На диаграммах рассеивания заметна зависимость целевых параметров от других параметров сварки.

```
In [13]: # характеристики
df.describe()
```

Out[13]:

	IW	IF	VW	FP	Depth	Width
count	72.000000	72.000000	72.000000	72.000000	72.000000	72.000000
mean	45.666667	141.333333	8.638889	78.333333	1.195556	1.970417
std	1.678363	5.145763	2.061078	21.493530	0.225081	0.279040
min	43.000000	131.000000	4.500000	50.000000	0.800000	1.680000
25%	44.000000	139.000000	8.000000	60.000000	1.080000	1.760000
50%	45.500000	141.000000	9.000000	80.000000	1.200000	1.840000
75%	47.000000	146.000000	10.000000	80.000000	1.290000	2.050000
max	49.000000	150.000000	12.000000	125.000000	1.760000	2.600000

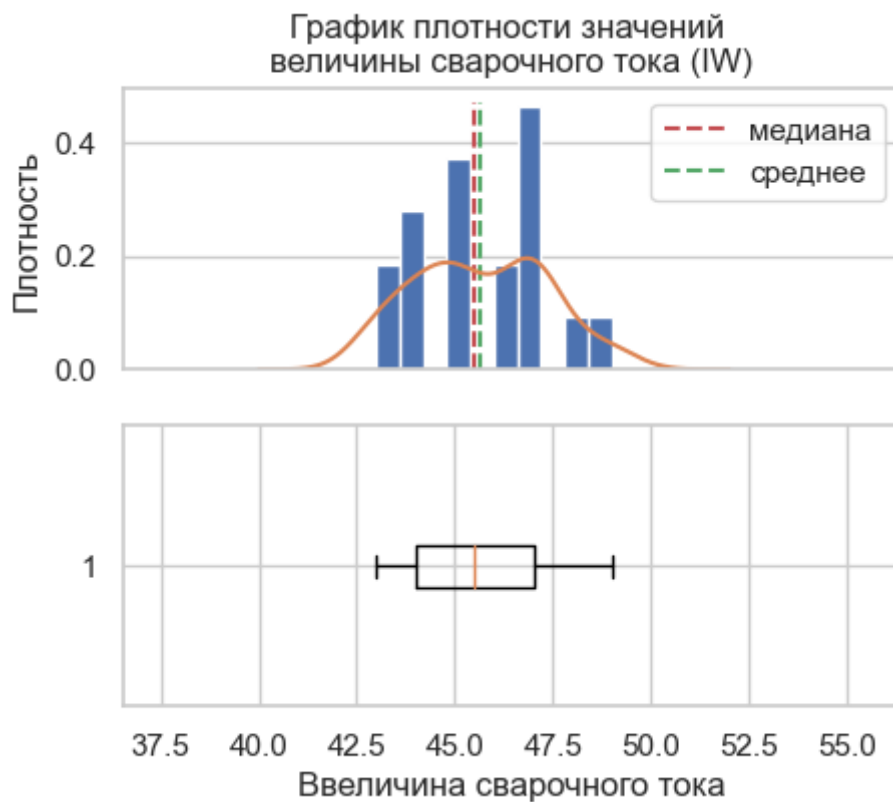
Рассмотрим признаки по отдельности.

```
In [14]: # график плотности значений величины сварочного тока
value_density_plot(df['IW'], mean_line_height=0.47, figsize=(5, 4),
```



```
title='График плотности значений \nвеличины сварочного тока (IW)'
xlabel='Ввеличина сварочного тока')
```

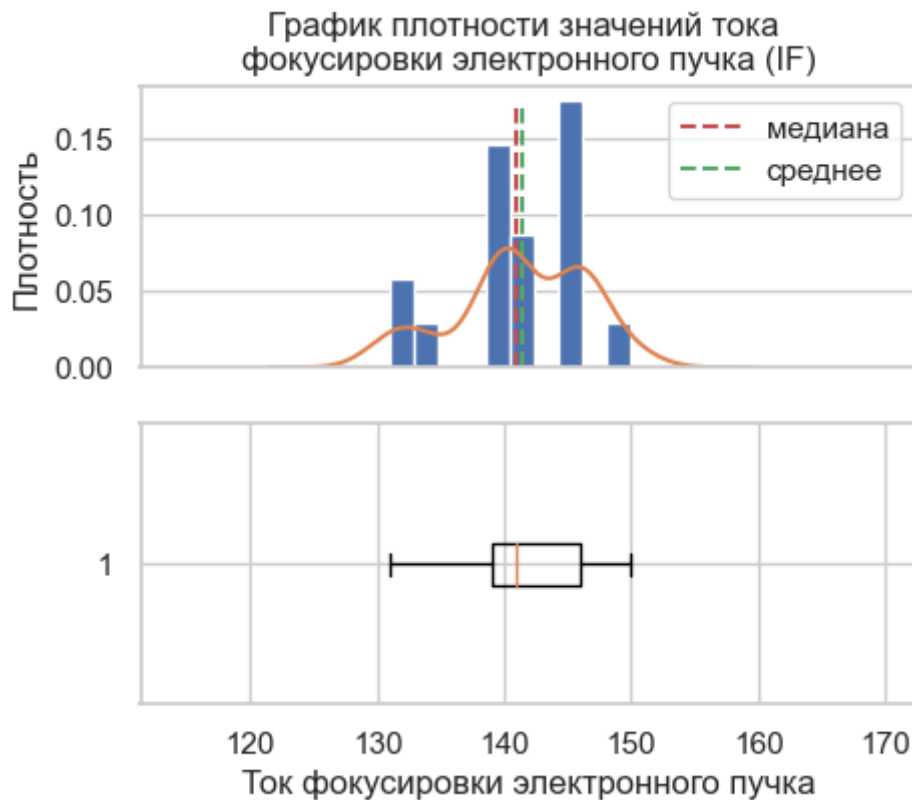
```
df['IW'].value_counts()
```



```
Out[14]: 47    20
         45    16
         44    12
         46     8
         43     8
         48     4
         49     4
         Name: IW, dtype: int64
```

```
In [15]: # график плотности значений тока фокусировки электронного пучка
value_density_plot(df['IF'], mean_line_height=0.17, figsize=(5, 4),
                  title='График плотности значений тока \nфокусировки электронного пучка',
                  xlabel='Ток фокусировки электронного пучка')

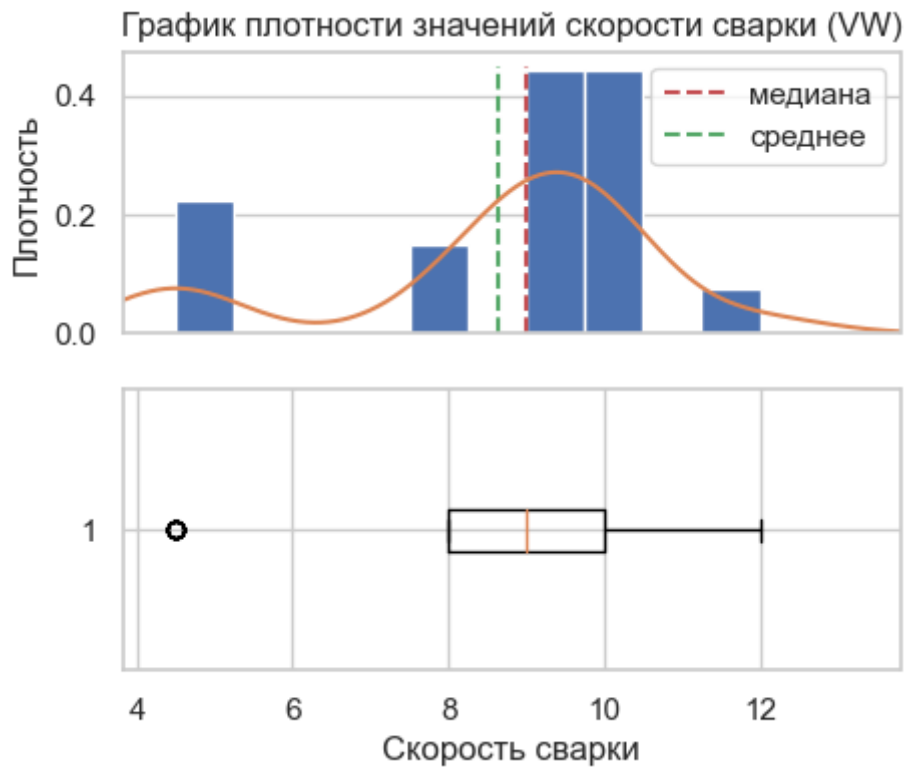
df['IF'].value_counts()
```



```
Out[15]: 146    24
         140    12
         141    12
         139     8
         134     4
         131     4
         132     4
         150     4
         Name: IF, dtype: int64
```

```
In [16]: # график плотности значений скорости сварки
value_density_plot(df['VW'], mean_line_height=0.45, figsize=(5, 4),
                  title='График плотности значений скорости сварки (VW)',
                  xlabel='Скорость сварки')

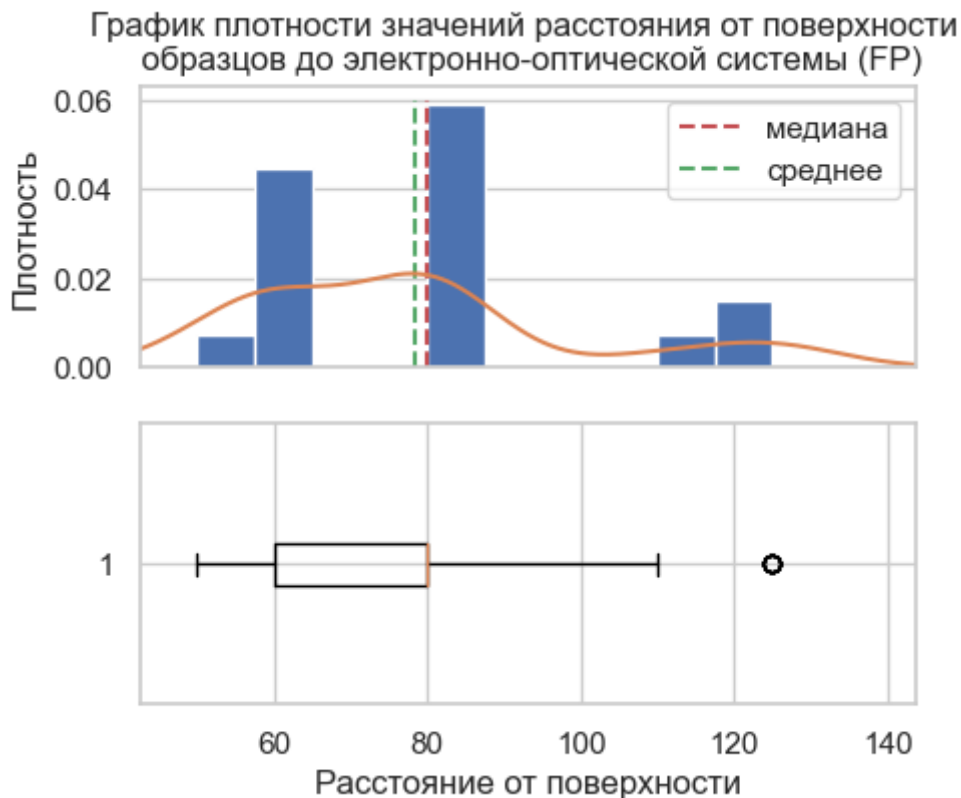
df['VW'].value_counts()
```



```
Out[16]: 10.0    24
          9.0    24
          4.5    12
          8.0     8
          12.0    4
          Name: VW, dtype: int64
```

```
In [17]: # график плотности значений расстояния от поверхности
# образцов до электронно-оптической системы FP
value_density_plot(df['FP'], mean_line_height=0.06, figsize=(5, 4),
                  title='График плотности значений расстояния от поверхности \n \
образцов до электронно-оптической системы (FP)',
                  xlabel='Расстояние от поверхности')

df['FP'].value_counts()
```

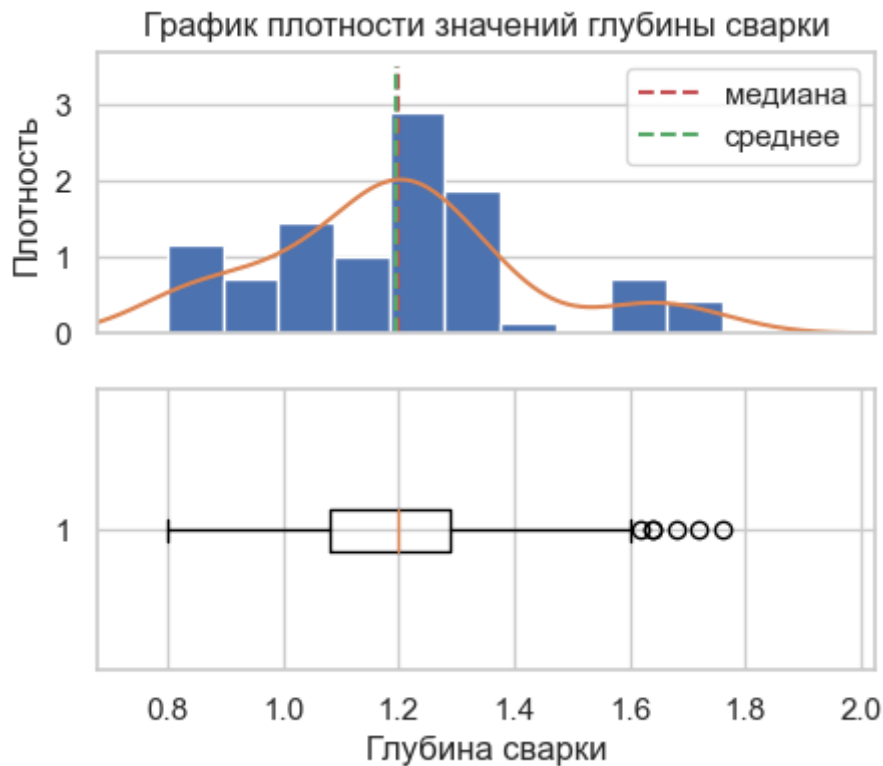


```
Out[17]: 80      32
          60      24
          125     8
          110     4
          50      4
          Name: FP, dtype: int64
```

Т.к. данных в датасете достаточно мало, то сложно судить о характере распределения значений, но учитывая, что медианные и средние значения выборок практически совпадают, то будем считать распределения нормальными.

Те, незначительные выбросы, которые мы видим на диаграммах распределения в данных **FP** и **VW**, нельзя считать выбросами, т.к. это фактические данные параметров технологического процесса сварки. Оставляем данные как есть.

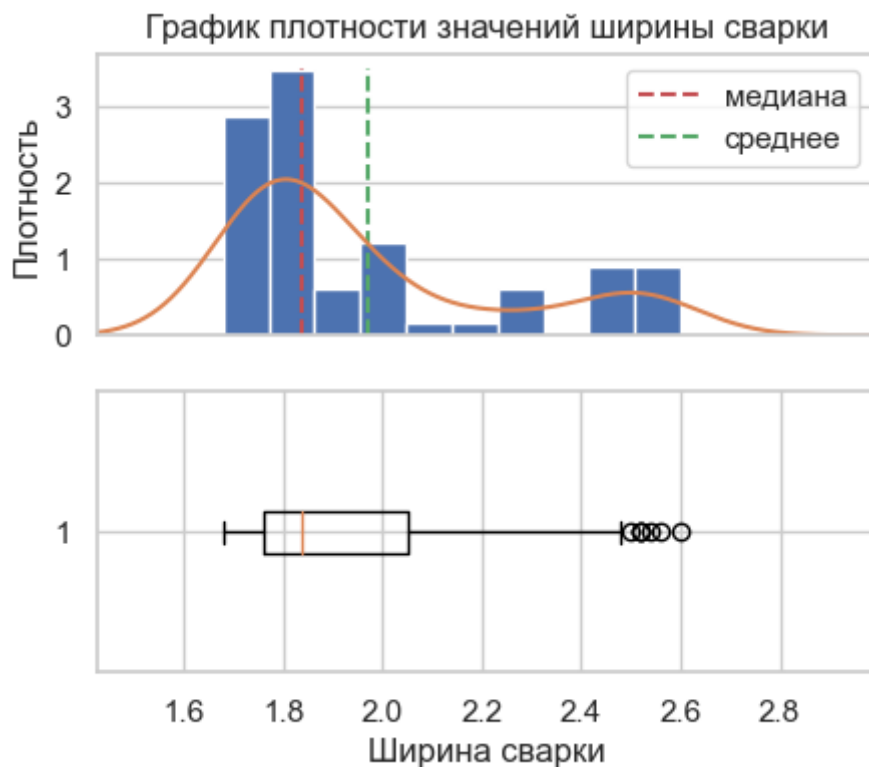
```
In [18]: # график плотности значений целевого признака Depth
value_density_plot(df['Depth'], mean_line_height=3.5, figsize=(5, 4),
                  title='График плотности значений глубины сварки',
                  xlabel='Глубина сварки')
```



```
In [19]: # число уникальных значений
df['Depth'].nunique()
```

Out[19]: 29

```
In [20]: # график плотности значений целевого признака Width
value_density_plot(df['Width'], mean_line_height=3.5, figsize=(5, 4),
                  title='График плотности значений ширины сварки',
                  xlabel='Ширина сварки')
```



```
In [21]: # число уникальных значений
df['Width'].nunique()
```

Out[21]: 25

Выводы по исследованию данных:

1. Пропусков в данных нет.
2. Признаки параметров технологического процесса имеют дискретные значения.
3. Есть дубли, их необходимо удалить.
4. Слишком маленький датасет, желательно его увеличить.

Предобработка

```
In [22]: # удаляем дубликаты
df.drop_duplicates(inplace=True, ignore_index=True)
```

```
In [23]: # проверяем результат
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65 entries, 0 to 64
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  -
 0    IW      65 non-null    int64  
 1    IF      65 non-null    int64  
 2    VW      65 non-null    float64 
 3    FP      65 non-null    int64  
 4    Depth   65 non-null    float64 
 5    Width   65 non-null    float64 
dtypes: float64(3), int64(3)
memory usage: 3.2 KB
```

Пока оставляем выборку как есть, если качество модели окажется низким, то попробуем увеличить выборку синтезировав новые объекты.

Разделение данных на выборки

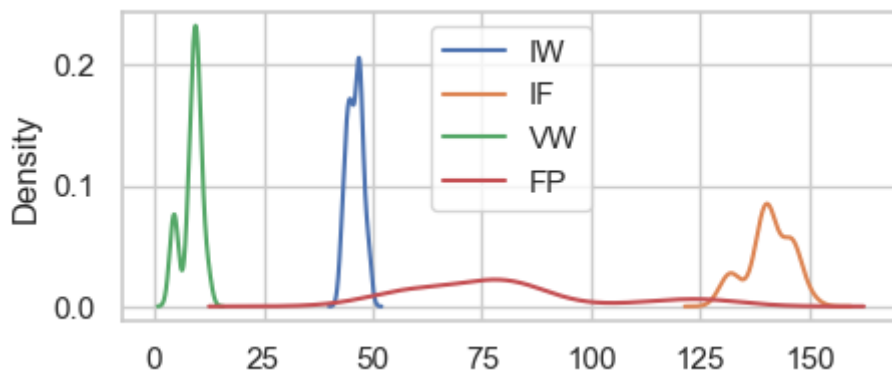
```
In [24]: # разделяем объекты и целевые признаки
X = df.drop(['Depth', 'Width'], axis=1)
y = df[['Depth', 'Width']]
```

```
In [25]: # разделение на выборки
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2,
                                                    random_state=RND,
                                                    shuffle=True)

# размеры
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[25]: ((52, 4), (52, 2), (13, 4), (13, 2))

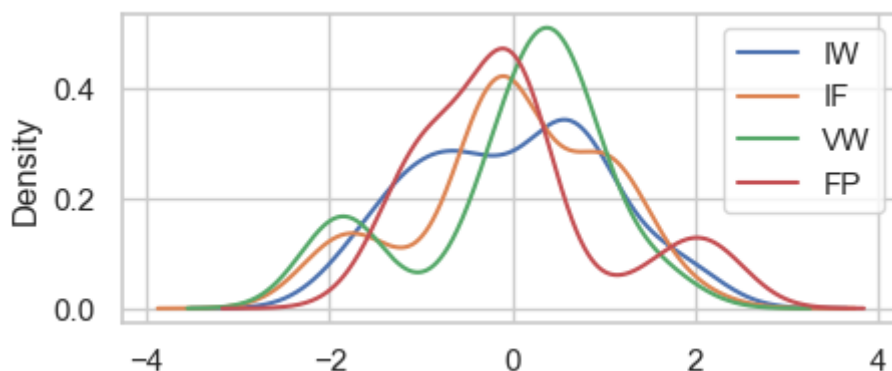
```
In [26]: # распределение значений признаков
X_train.plot(kind='kde', figsize=(5,2));
```



Для "линейных" моделей могут потребоваться стандартизованные, нормализованные или масштабированные признаки. Подготовим соответствующие выборки и будем их использовать по мере необходимости.

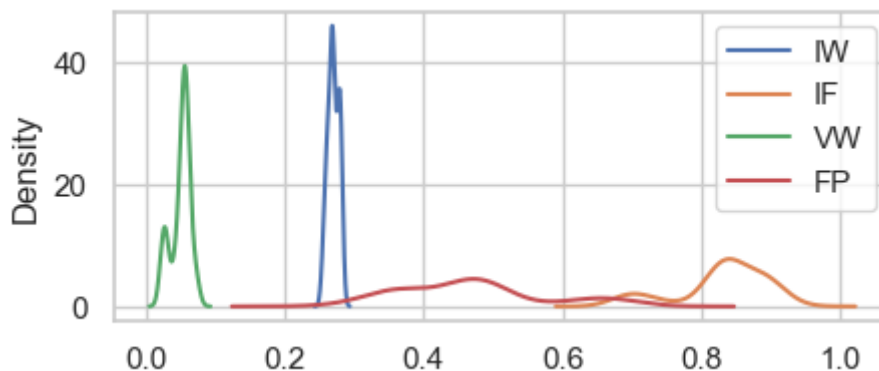
```
In [27]: # выполним стандартизацию объектов выборок
scaler = StandardScaler().fit(X_train, y_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# распределение значений признаков после стандартизации
pd.DataFrame(
    X_train_scaled, columns=X.columns
).plot(kind='kde', figsize=(5,2));
```



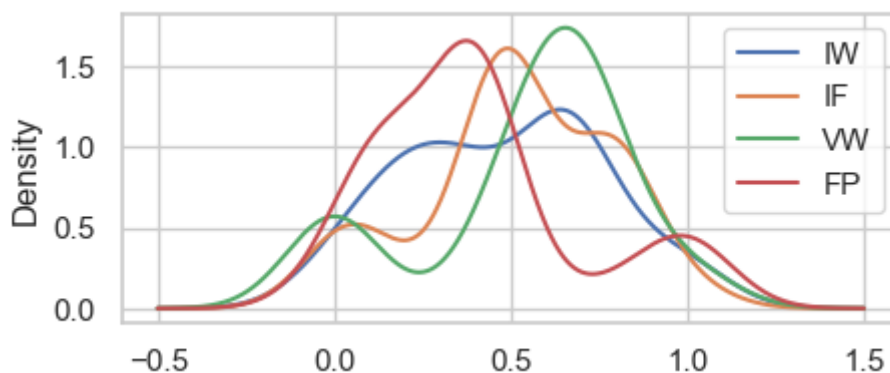
```
In [28]: # выполним нормализацию объектов выборок
normalizer = Normalizer().fit(X_train, y_train)
X_train_norm = normalizer.transform(X_train)
X_test_norm = normalizer.transform(X_test)

# распределение значений признаков после нормализации
pd.DataFrame(
    X_train_norm, columns=X.columns
).plot(kind='kde', figsize=(5,2));
```



```
In [29]: # выполним масштабирование объектов выборки
min_max_scaler = MinMaxScaler().fit(X_train, y_train)
X_train_minmax = min_max_scaler.transform(X_train)
X_test_minmax = min_max_scaler.transform(X_test)

# распределение значений признаков после масштабирования
pd.DataFrame(
    X_train_minmax, columns=X.columns
).plot(kind='kde', figsize=(5,2));
```



Тестирование моделей

```
In [30]: # создадим список для сохранения параметров моделей
models = []
# параметры списка
model_params = ['Название модели', 'Метрика R2', 'Метрика RMSE']
```

Модель линейной регрессии

```
In [31]: # создадим и обучим модель
lr_model = LinearRegression().fit(X_train, y_train)
lr_preds = lr_model.predict(X_test)
lr_rmse = mean_squared_error(y_test, lr_preds, squared=False)
lr_r2 = lr_model.score(X_test, y_test)
print('LinearRegression, R2 на тестовой выборке:', lr_r2)
print('LinearRegression, RMSE на тестовой выборке:', lr_rmse)
```

```
LinearRegression, R2 на тестовой выборке: 0.8519073860015265
LinearRegression, RMSE на тестовой выборке: 0.08135804333621585
```

```
In [32]: # проверим модель на стандартизованных данных выборки
lr_model_scaled = LinearRegression().fit(X_train_scaled, y_train)
lr_preds_scaled = lr_model_scaled.predict(X_test_scaled)
lr_rmse_scaled = mean_squared_error(y_test, lr_preds_scaled, squared=False)
```



```
lr_r2_scaled = lr_model_scaled.score(X_test_scaled, y_test)
print('LinearRegression, R2 на стандартизированной выборке:', lr_r2_scaled)
print('LinearRegression, RMSE на стандартизированной выборке:', lr_rmse_scaled)
```

LinearRegression, R2 на стандартизированной выборке: 0.8519073860015247
 LinearRegression, RMSE на стандартизированной выборке: 0.0813580433362163

```
In [33]: # проверим модель на нормализованных данных выборки
lr_model_norm = LinearRegression().fit(X_train_norm, y_train)
lr_preds_norm = lr_model_norm.predict(X_test_norm)
lr_rmse_norm = mean_squared_error(y_test, lr_preds_norm, squared=False)
lr_r2_norm = lr_model_norm.score(X_test_norm, y_test)
print('LinearRegression, R2 на нормализованной выборке:', lr_r2_norm)
print('LinearRegression, RMSE на нормализованной выборке:', lr_rmse_norm)
```

LinearRegression, R2 на нормализованной выборке: 0.8172064326838904
 LinearRegression, RMSE на нормализованной выборке: 0.08946464134107285

```
In [34]: # проверим модель на масштабированных данных выборки
lr_model_minmax = LinearRegression().fit(X_train_minmax, y_train)
lr_preds_minmax = lr_model_minmax.predict(X_test_minmax)
lr_rmse_minmax = mean_squared_error(y_test, lr_preds_minmax, squared=False)
lr_r2_minmax = lr_model_minmax.score(X_test_minmax, y_test)
print('LinearRegression, R2 на масштабированной выборке:', lr_r2_minmax)
print('LinearRegression, RMSE на масштабированной выборке:', lr_rmse_minmax)
```

LinearRegression, R2 на масштабированной выборке: 0.8519073860015248
 LinearRegression, RMSE на масштабированной выборке: 0.08135804333621627

Метрики, хоть и не значительно, но улучшились на стандартизованных и масштабированных данных. На нормализованных данных метрики просели.

```
In [35]: # сохраняем лучшие метрики в список
models.append(['LinearRegression', lr_r2_minmax, lr_rmse_minmax])
```

Ridge

Линейный метод наименьших квадратов с регуляризацией l2.

```
In [36]: # создадим и обучим модель
ridge_model = Ridge(random_state=RND).fit(X_train, y_train)
ridge_preds = ridge_model.predict(X_test)
ridge_rmse = mean_squared_error(y_test, ridge_preds, squared=False)
ridge_r2 = ridge_model.score(X_test, y_test)
print('Ridge, R2 на тестовой выборке:', ridge_r2)
print('Ridge, RMSE на тестовой выборке:', ridge_rmse)
```

Ridge, R2 на тестовой выборке: 0.8540111619283008
 Ridge, RMSE на тестовой выборке: 0.0807994812992135

```
In [37]: # проверим модель на стандартизованных данных выборки
ridge_model_scaled = Ridge().fit(X_train_scaled, y_train)
ridge_preds_scaled = ridge_model_scaled.predict(X_test_scaled)
ridge_rmse_scaled = mean_squared_error(y_test, ridge_preds_scaled, squared=False)
ridge_r2_scaled = ridge_model_scaled.score(X_test_scaled, y_test)
print('Ridge, R2 на стандартизированной выборке:', ridge_r2_scaled)
print('Ridge, RMSE на стандартизированной выборке:', ridge_rmse_scaled)
```

Ridge, R2 на стандартизированной выборке: 0.877648291932538
 Ridge, RMSE на стандартизированной выборке: 0.07390788656398409

```
In [38]: # проверим модель на нормализованных данных выборки
ridge_model_norm = Ridge().fit(X_train_norm, y_train)
```

```
ridge_preds_norm = ridge_model_norm.predict(X_test_norm)
ridge_rmse_norm = mean_squared_error(y_test, ridge_preds_norm, squared=False)
ridge_r2_norm = ridge_model_norm.score(X_test_norm, y_test)
print('Ridge, R2 на нормализованной выборке:', ridge_r2_norm)
print('Ridge, RMSE на нормализованной выборке:', ridge_rmse_norm)
```

Ridge, R2 на нормализованной выборке: 0.11721189535758636
 Ridge, RMSE на нормализованной выборке: 0.19858311868402645

```
In [39]: # проверим модель на масштабированных данных выборки
ridge_model_minmax = Ridge().fit(X_train_minmax, y_train)
ridge_preds_minmax = ridge_model_minmax.predict(X_test_minmax)
ridge_rmse_minmax = mean_squared_error(y_test, ridge_preds_minmax, squared=False)
ridge_r2_minmax = ridge_model_minmax.score(X_test_minmax, y_test)
print('Ridge, R2 на масштабированной выборке:', ridge_r2_minmax)
print('Ridge, RMSE на масштабированной выборке:', ridge_rmse_minmax)
```

Ridge, R2 на масштабированной выборке: 0.8187107504707556
 Ridge, RMSE на масштабированной выборке: 0.09002937568260751

Наилучший результат получен на стандартизованных данных выборки, а нормализация данных привела к ухудшению метрик качества модели.

```
In [40]: # сохраняем лучшие метрики в список
models.append(['Ridge', ridge_r2_scaled, ridge_rmse_scaled])
```

Дерево решений

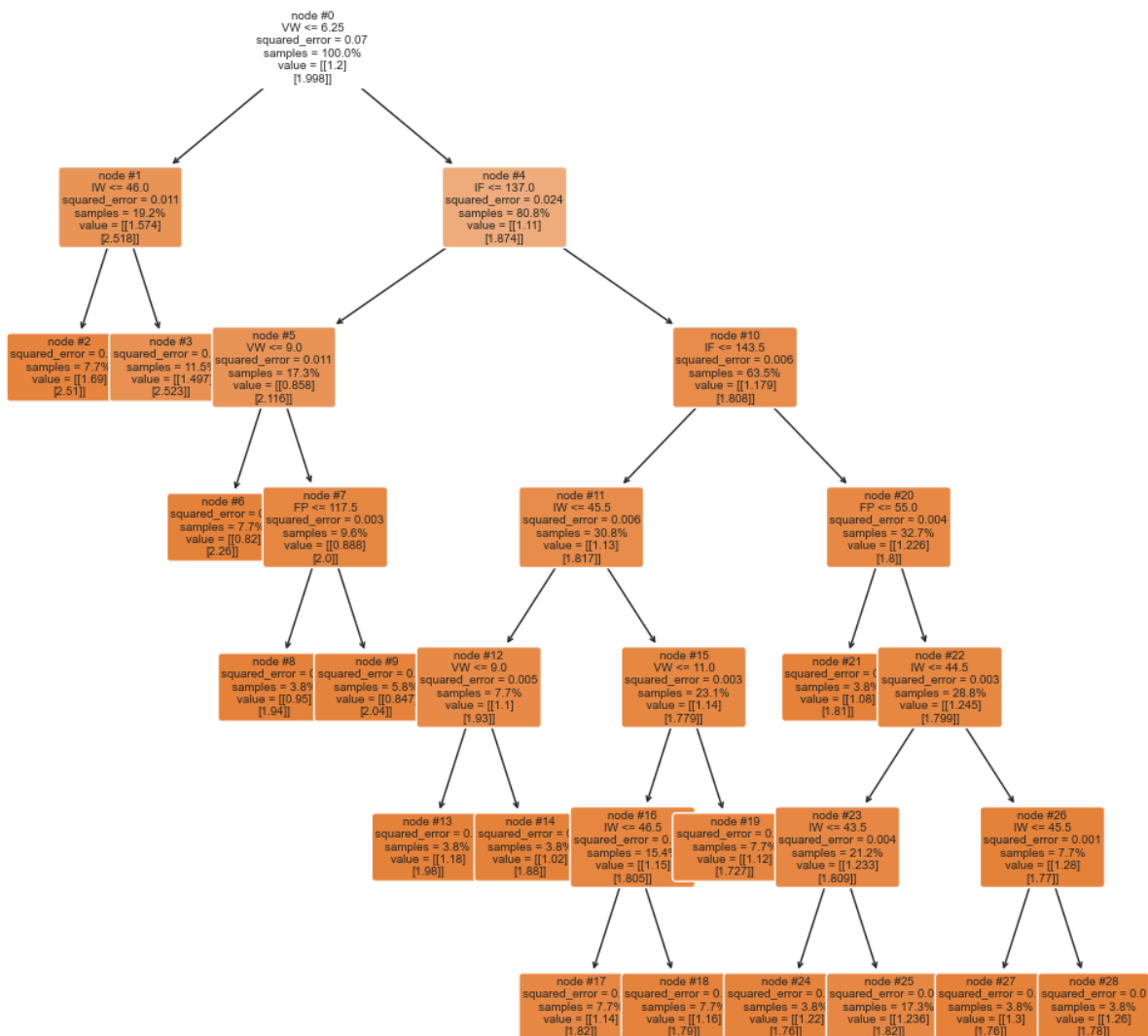
```
In [41]: # создадим и обучим модель
tree_model = DecisionTreeRegressor(random_state=RND).fit(X_train, y_train)
tree_preds = tree_model.predict(X_test)
tree_rmse = mean_squared_error(y_test, tree_preds, squared=False)
tree_r2 = tree_model.score(X_test, y_test)
print('DecisionTreeRegressor, R2 на тестовой выборке:', tree_r2)
print('DecisionTreeRegressor, RMSE на тестовой выборке:', tree_rmse)
```

DecisionTreeRegressor, R2 на тестовой выборке: 0.9209118633208002
 DecisionTreeRegressor, RMSE на тестовой выборке: 0.05921584750391146

"Деревянные" модели не требуют масштабирования и стандартизации данных, поэтому нет смысла проверять данную модель на преобразованных выборках.

```
In [42]: # сохраняем полученные метрики в список
models.append(['DecisionTreeRegressor', tree_r2, tree_rmse])
```

```
In [43]: # визуализируем полученное дерево решений
plt.figure(figsize=(12,12))
plot_tree(tree_model,
          feature_names=X.columns,
          class_names=y.columns,
          filled=True,
          node_ids=True,
          proportion=True,
          rounded=True,
          fontsize=7);
```



Случайный лес

```
In [44]: # создадим и обучим модель
forest_model = RandomForestRegressor(random_state=RND).fit(X_train, y_train)
forest_preds = forest_model.predict(X_test)
forest_rmse = mean_squared_error(y_test, forest_preds, squared=False)
forest_r2 = forest_model.score(X_test, y_test)
print('RandomForestRegressor, R2 на тестовой выборке:', forest_r2)
print('RandomForestRegressor, RMSE на тестовой выборке:', forest_rmse)
```

RandomForestRegressor, R2 на тестовой выборке: 0.9189176752625283

RandomForestRegressor, RMSE на тестовой выборке: 0.06017913162196182

```
In [45]: # сохраняем полученные метрики в список
models.append(['RandomForestRegressor', forest_r2, forest_rmse])
```

Случайный лес Extra

```
In [46]: # создадим и обучим модель
extra_model = ExtraTreesRegressor(random_state=RND).fit(X_train, y_train)
extra_preds = extra_model.predict(X_test)
extra_rmse = mean_squared_error(y_test, extra_preds, squared=False)
extra_r2 = extra_model.score(X_test, y_test)
```

```
print('ExtraTreesRegressor, R2 на тестовой выборке:', extra_r2)
print('ExtraTreesRegressor, RMSE на тестовой выборке:', extra_rmse)
```

ExtraTreesRegressor, R2 на тестовой выборке: 0.9209118633208027
 ExtraTreesRegressor, RMSE на тестовой выборке: 0.05921584750391055

```
In [47]: # сохраняем полученные метрики в список
models.append(['ExtraTreesRegressor', extra_r2, extra_rmse])
```

Градиентный бустинг

Метод GradientBoostingRegressor не поддерживает многоцелевые предсказания. Для этих целей оборачиваем его в специальный метод MultiOutputRegressor.

```
In [48]: # создадим и обучим модель
boost_model = MultiOutputRegressor(
    GradientBoostingRegressor(random_state=RND)
).fit(X_train, y_train)
boost_preds = boost_model.predict(X_test)
boost_rmse = mean_squared_error(y_test, boost_preds, squared=False)
boost_r2 = boost_model.score(X_test, y_test)
print('GradientBoostingRegressor, R2 на тестовой выборке:', boost_r2)
print('GradientBoostingRegressor, RMSE на тестовой выборке:', boost_rmse)
```

GradientBoostingRegressor, R2 на тестовой выборке: 0.9208391375663638
 GradientBoostingRegressor, RMSE на тестовой выборке: 0.059278376639489934

```
In [49]: # сохраняем полученные метрики в список
models.append(['GradientBoostingRegressor', boost_r2, boost_rmse])
```

Модель полносвязной НС

```
In [50]: # создаем модель полносвязной нейронной сети
nn_model = Sequential()
nn_model.add(Dense(16, activation='relu', input_dim=4))
nn_model.add(Dense(2, activation='exponential'))
```

```
In [51]: # архитектура сети
nn_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 16)	80
dense_1 (Dense)	(None, 2)	34

=====
 Total params: 114
 Trainable params: 114
 Non-trainable params: 0

```
In [52]: # компилируем модель
nn_model.compile(optimizer='nadam',
                 loss='mse',
                 metrics=['accuracy'])
```

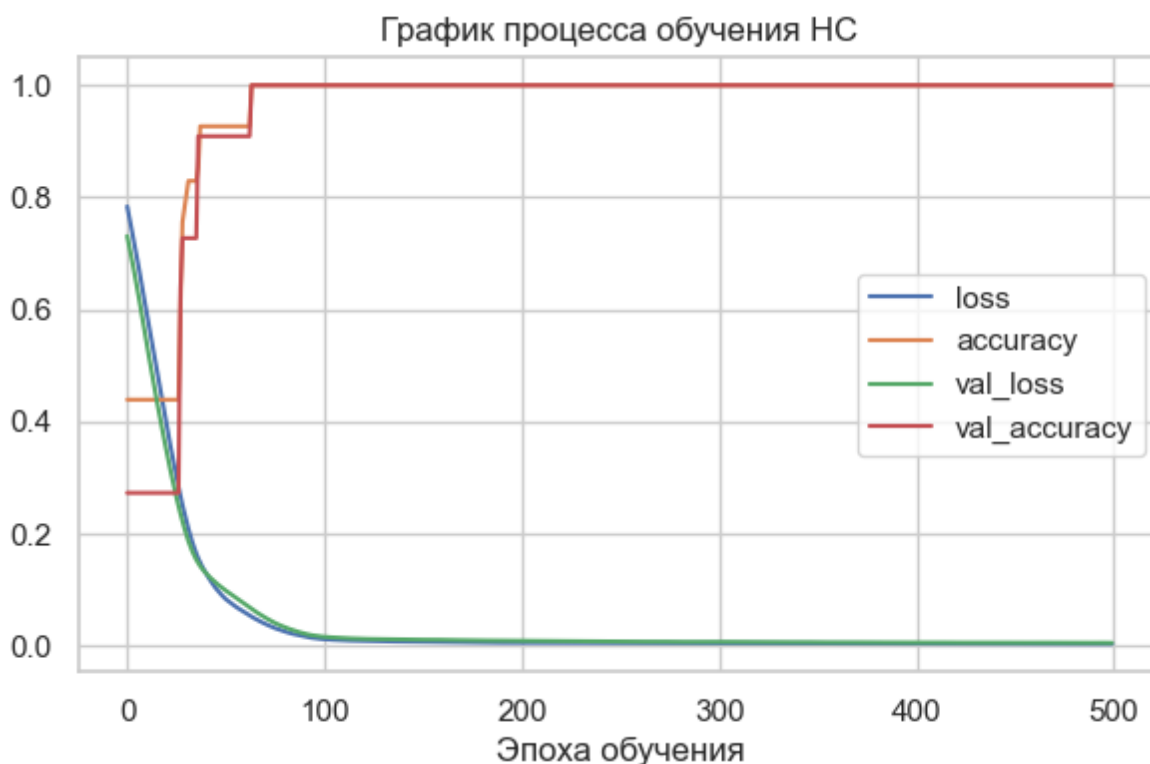
```
In [53]: %%time
```

```
# обучаем модель на стандартизированных данных
history = nn_model.fit(X_train_scaled, y_train,
                       batch_size=16,
                       epochs=500,
                       validation_split=0.2,
                       verbose=0)
```

CPU times: total: 31.3 s

Wall time: 27 s

```
In [54]: # визуализация обучения
plt.figure(figsize=(7,4))
plt.title('График процесса обучения НС')
for key in history.history.keys():
    plt.plot(history.history[key], label=key)
plt.xlabel('Эпоха обучения')
plt.legend();
```



```
In [55]: # оценим метрики
nn_preds = nn_model.predict(X_test_scaled)
nn_rmse = mean_squared_error(y_test, nn_preds, squared=False)
nn_r2 = r2_score(y_test, nn_preds)
print('Sequential NN, R2 на тестовой выборке:', nn_r2)
print('Sequential NN, RMSE на тестовой выборке:', nn_rmse)
```

1/1 [=====] - 0s 135ms/step

Sequential NN, R2 на тестовой выборке: 0.8984681697390569

Sequential NN, RMSE на тестовой выборке: 0.06736830784736625

```
In [56]: # сохраняем полученные метрики в список
models.append(['Sequential NN', nn_r2, nn_rmse])
```

Модель полносвязной нейронной сети показала не плохой результат. Есть возможность улучшить ее качество путем подбора архитектуры и гиперпараметров. Это требует достаточно продолжительного времени для проведения экспериментов. На обычном ПК без использования GPU даже такая небольшая модель со 114

параметрами обучалась около 30 секунд. В рамках данного проекта мы не будем этого делать.

Увеличение размера выборки должно положительным образом сказаться на качестве подготовки модели.

Проверка на адекватность

```
In [57]: # создадим и обучим модель
dummy_model = DummyRegressor(strategy='median').fit(X_train, y_train)
dummy_preds = dummy_model.predict(X_test)
dummy_rmse = mean_squared_error(y_test, dummy_preds, squared=False)
dummy_r2 = dummy_model.score(X_test, y_test)
print('DummyRegressor, R2 на тестовой выборке:', dummy_r2)
print('DummyRegressor, RMSE на тестовой выборке:', dummy_rmse)
```

```
DummyRegressor, R2 на тестовой выборке: -0.026847176843300402
DummyRegressor, RMSE на тестовой выборке: 0.21440839534079803
```

```
In [58]: # сохраняем полученные метрики в список
models.append(['DummyRegressor', dummy_r2, dummy_rmse])
```

Данная "случайная" модель выдаёт медианные значения целевых признаков на все объекты тестовой выборки.

Сравнивая метрики "случайной" модели с метриками рассмотренных нами моделей, можно сделать вывод, что все модели обучились и дают адекватный ответ.

Выводы по моделям

```
In [59]: # соберём все данные по моделям в общую таблицу
models_df = pd.DataFrame(models, columns=model_params)
```

```
In [60]: # отсортируем модели по убыванию метрики R2
models_df.sort_values('Метрика R2', ascending=False)
```

```
Out[60]:
```

	Название модели	Метрика R2	Метрика RMSE
4	ExtraTreesRegressor	0.920912	0.059216
2	DecisionTreeRegressor	0.920912	0.059216
5	GradientBoostingRegressor	0.920839	0.059278
3	RandomForestRegressor	0.918918	0.060179
6	Sequential NN	0.898468	0.067368
1	Ridge	0.877648	0.073908
0	LinearRegression	0.851907	0.081358
7	DummyRegressor	-0.026847	0.214408

Лучший результат с дефолтными гиперпараметрами показала модель `ExtraTreesRegressor`.

В целом, "деревянные" модели и модель градиентного бустинга показали очень похожие результаты и все могут быть использованы в качестве итоговой модели.

Хорошие результаты показала полносвязная нейронная сеть. У данной модели явно есть потенциал для улучшения метрик. Но для этого потребуются дополнительные исследования и увеличение размера обучающей выборки.

Создание итоговой модели

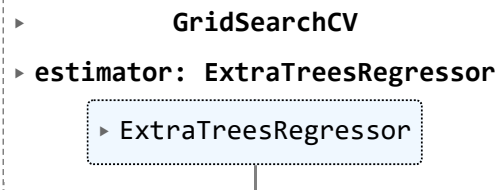
Для получения итоговой модели выполним обучение модели на полной выборке с использованием механизма кросс-валидации. Для этого воспользуемся методом GridSearchCV из библиотеки sklearn.model_selection.

```
In [61]: # итоговая модель
model = ExtraTreesRegressor(random_state=RND)

# добавим модель для кросс-валидации
grid = GridSearchCV(model, {}, scoring='r2', n_jobs=-2)

# обучим модель
grid.fit(X, y)
```

```
Out[61]:
```

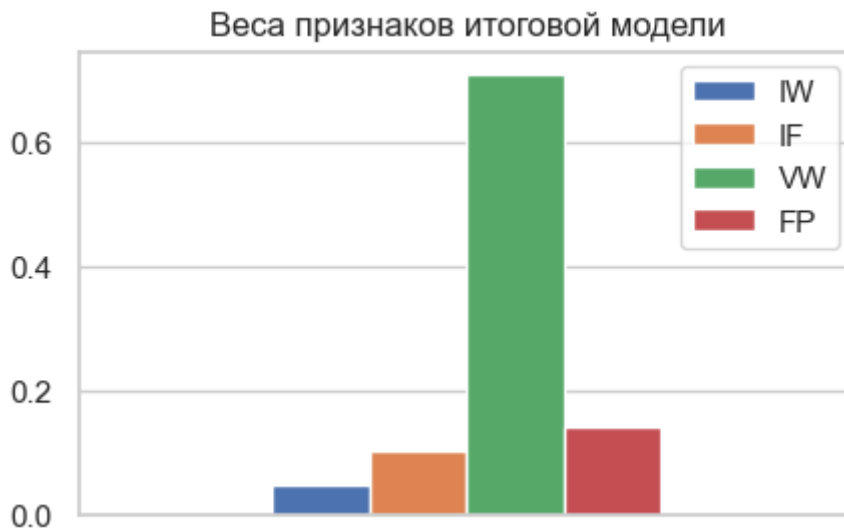


```
In [62]: print('Качество итоговой модели на кросс-валидации:', grid.score(X, y))
```

Качество итоговой модели на кросс-валидации: 0.9565116636363731

```
In [63]: # запомним лучшую модель
best_model = grid.best_estimator_
```

```
In [64]: # оценим веса признаков итоговой модели
pd.DataFrame(
    [best_model.feature_importances_], columns=X.columns
).plot(
    kind='bar', grid=True, figsize=(5, 3), xticks=[],
    title='Веса признаков итоговой модели'
);
```



Экспорт модели

Сохраним полученную модель в файл. В дальнейшем её можно будет загрузить и использовать для предсказания параметров сварного шва в зависимости от поданных на вход параметров технологического процесса.

```
In [65]: # имя файла модели
model_file = 'welding.pkl'

# сохраним итоговую модель в файл
with open(model_file, 'wb') as F:
    pickle.dump(best_model, F)
```

Проверим сохраненную модель на работоспособность.

```
In [66]: # случайный номер строки
row = np.random.randint(65)

# пример исходных данных
print('Параметры технологического процесса:', X.loc[row].values)
print('Истинные значения глубины и ширины шва:', y.loc[row].values)
```

Параметры технологического процесса: [44. 146. 9. 60.]
Истинные значения глубины и ширины шва: [1.2 1.88]

```
In [67]: # предсказания итоговой модели
print('Предсказания итоговой модели:', best_model.predict(X[row:row+1]))
```

Предсказания итоговой модели: [[1.23272727 1.79818182]]

```
In [68]: # загрузим маодель из файла
with open(model_file, 'rb') as F:
    loaded_model = pickle.load(F)
```

```
In [69]: # предсказания модели загруженной из файла
print('Предсказания загруженной модели:', loaded_model.predict(X[row:row+1]))
```

Предсказания загруженной модели: [[1.23272727 1.79818182]]

Предсказания модели загруженной из файла совпадают с предсказаниями итоговой модели.

Можно сделать вывод, что сохранение модели выполнено успешно и данная модель, в дальнейшем, может быть загружена и использована в коде стороннего приложения.

Выводы

Нам в качестве исходных данных были предоставлены результаты экспериментальных исследований технологического процесса электронно-лучевой сварки.

Поставлена задача провести прогнозирование глубины и ширины сварного шва в зависимости от параметров технологического процесса:

- Снижение величины сварочного тока IW ;
- Увеличение тока фокусировки электронного пучка IF ;
- Увеличение скорости сварки VW ;
- Изменение расстояния от поверхности образцов до электронно-оптической системы FP .

Для прогнозирования значений ширины и глубины сварного соединения решено было использовать регрессионные модели машинного обучения и выбраны соответствующие метрики для оценки качества.

Проведён анализ и предобработка предоставленных данных. Данные были подготовлены и разбиты на выборки для применения в моделях машинного обучения.

На подготовленных данных протестировали несколько различных классов моделей и оценили полученные результаты на адекватность. В результате тестирования была выбрана итоговая модель с наилучшими показателями качества выбранных метрик - ExtraTreesRegressor.

Итоговая модель, для улучшения качества, была обучена на полном наборе данных с использованием кросс-валидации.

Так же оценили уровень влияния параметров технологического процесса на целевые признаки, построили график. У итоговой модели наиболее влияние на предсказания ширины и глубины сварного шва оказывает скорость сварки VW , а наименьшее - величина сварочного тока IW .

Усреднённое качество итоговой модели на кросс-валидации, метрика R^2 : 0.9565.

Таким образом, задача получить наиболее близкое к "1" значение метрики R^2 выполнена успешно и мы можем с определенной точностью прогнозировать значения параметров сварного шва исходя из заданных параметров технологического процесса.

Итоговая модель была экспортирована в rkl-файл для дальнейшего применения её в программном обеспечении для прогнозирования глубины и ширины сварного шва.

In []: