Проект "Промышленность"

Table of Contents

Описание проекта

Исходные данные

Цель: Уменьшить энергопотребление
Задача: Построить модель, предсказывающую температуру стали

Описание этапа обработки

Сталь обрабатывают в металлическом ковше вместимостью около 100 тонн. Чтобы ковш выдерживал высокие температуры, изнутри его облицовывают огнеупорным кирпичом. Расплавленную сталь заливают в ковш и подогревают до нужной температуры графитовыми электродами. Они установлены в крышке ковша.

Из сплава выводится сера (десульфурация), добавлением примесей корректируется химический состав и отбираются пробы. Сталь легируют — изменяют её состав — подавая куски сплава из бункера для сыпучих материалов или проволоку через специальный трайб-аппарат (англ. tribe, «масса»).

Перед тем как первый раз ввести легирующие добавки, измеряют температуру стали и производят её химический анализ. Потом температуру на несколько минут повышают, добавляют легирующие материалы и продувают сплав инертным газом. Затем его перемешивают и снова проводят измерения. Такой цикл повторяется до достижения целевого химического состава и оптимальной температуры плавки.

Тогда расплавленная сталь отправляется на доводку металла или поступает в машину непрерывной разливки. Оттуда готовый продукт выходит в виде заготовок-слябов (англ. *slab*, «плита»).

Описание данных

Данные состоят из файлов, полученных из разных источников:

- `data_arc.csv` — данные об электродах;
- `data_bulk.csv` — данные о подаче сыпучих материалов (объём);
- `data_bulk_time.csv` — данные о подаче сыпучих материалов (время);
- `data_gas.csv` — данные о продувке сплава газом;
- `data_temp.csv` — результаты измерения температуры;
- `data_wire.csv` — данные о проволочных материалах (объём);
- `data_wire_time.csv` — данные о проволочных материалах (время).

Во всех файлах столбец `key` содержит номер партии. В файлах может быть несколько строк с одинаковым значением `key`: они соответствуют разным итерациям обработки.

Дополнительная информация из внешнего источника:

основной целью продувки металла инертными газами является его дегазация, удаление неметаллических включений, выравнивание химического состава и температуры по всему объему металла.

Предварительная гипотеза:

Предполагается что температура сплава является показателем качества сплава стали либо используется как параметр при его расчете. Затраты электроенергии связаны с нагревом сплава. Т.к. для введения добавок сплав необходимо нагревать, количество итераций (процедур введения примесей и последующее измерение качества) непосредственно влияет на затраты электроенергии. Модель, предсказывающая температуру сплава позволит заранее подобрать необходимое количество примесей и минимизировать количество итераций.

Подключение библиотек

In [1]:
```
import os
from pathlib import Path
import urllib
import pandas as pd
import numpy as np
```

```
# библиотека для графиков
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Подготовка данных к обучению моделей

# библиотека для разбиения данных на выборки
from sklearn.model_selection import train_test_split
# библиотеки для масштабирования признаков
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
# библиотека для перемешивания датафреймов
from sklearn.utils import shuffle
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [3]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

from sklearn.metrics import fbeta_score, make_scorer
```

In [4]:
```
from xgboost import XGBRegressor
```

Загрузка и просмотр данных

In [5]:
```
data_arc=pd.read_csv('/datasets/final_steel/data_arc.csv')
data_bulk=pd.read_csv('/datasets/final_steel/data_bulk.csv')
data_bulk_time=pd.read_csv('/datasets/final_steel/data_bulk_time.csv')
data_gas=pd.read_csv('/datasets/final_steel/data_gas.csv')
data_temp=pd.read_csv('/datasets/final_steel/data_temp.csv')
data_wire=pd.read_csv('/datasets/final_steel/data_wire.csv')
data_wire_time=pd.read_csv('/datasets/final_steel/data_wire_time.csv')
```

In [6]:
```
data_arc.head()
```

Out[6]:

	key	Начало нагрева дугой	Конец нагрева дугой	Активная мощность	Реактивная мощность
0	1	2019-05-03 11:02:14	2019-05-03 11:06:02	0.976059	0.687084
1	1	2019-05-03 11:07:28	2019-05-03 11:10:33	0.805607	0.520285
2	1	2019-05-03 11:11:44	2019-05-03 11:14:36	0.744363	0.498805
3	1	2019-05-03 11:18:14	2019-05-03 11:24:19	1.659363	1.062669
4	1	2019-05-03 11:26:09	2019-05-03 11:28:37	0.692755	0.414397

In [7]:
```
data_arc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14876 entries, 0 to 14875
Data columns (total 5 columns):
key                     14876 non-null int64
Начало нагрева дугой    14876 non-null object
Конец нагрева дугой     14876 non-null object
Активная мощность       14876 non-null float64
Реактивная мощность     14876 non-null float64
dtypes: float64(2), int64(1), object(2)
memory usage: 581.2+ KB
```

In [8]:
```
print(len(data_arc['key'].unique()))
print(len(data_arc['key']))
```

```
3214
14876
```

Промежуточный вывод:

В файле с данными об электродах содержится информация о мощности и времени работы электродов для 3214 партий. Всего 14876 строк. Пропусков нет. По данным можно оценить затраты электроенергии на партии.

In [9]:
```
data_bulk.head()
```

Out[9]:

	key	Bulk 1	Bulk 2	Bulk 3	Bulk 4	Bulk 5	Bulk 6	Bulk 7	Bulk 8	Bulk 9	Bulk 10	Bulk 11	Bulk 12	Bulk 13	Bul 1
0	1	NaN	NaN	NaN	43.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	206.0	NaN	150.
1	2	NaN	NaN	NaN	73.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	206.0	NaN	149.
2	3	NaN	NaN	NaN	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	205.0	NaN	152.
3	4	NaN	NaN	NaN	81.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	207.0	NaN	153.
4	5	NaN	NaN	NaN	78.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	203.0	NaN	151.

In [10]:
```
data_bulk.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3129 entries, 0 to 3128
Data columns (total 16 columns):
key         3129 non-null int64
Bulk 1      252 non-null float64
Bulk 2      22 non-null float64
Bulk 3      1298 non-null float64
Bulk 4      1014 non-null float64
Bulk 5      77 non-null float64
Bulk 6      576 non-null float64
Bulk 7      25 non-null float64
Bulk 8      1 non-null float64
Bulk 9      19 non-null float64
Bulk 10     176 non-null float64
Bulk 11     177 non-null float64
Bulk 12     2450 non-null float64
Bulk 13     18 non-null float64
Bulk 14     2806 non-null float64
Bulk 15     2248 non-null float64
dtypes: float64(15), int64(1)
memory usage: 391.2 KB
```

In [11]:
```
print(len(data_bulk['key'].unique()))
```

```
print(len(data_bulk['key']))
```

```
3129
3129
```

In [12]:
```
data_bulk_time.head()
```

Out[12]:

	key	Bulk 1	Bulk 2	Bulk 3	Bulk 4	Bulk 5	Bulk 6	Bulk 7	Bulk 8	Bulk 9	Bulk 10	Bulk 11	Bulk 12	Bull 13
0	1	NaN	NaN	NaN	2019-05-03 11:21:30	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019-05-03 11:03:52	NaN
1	2	NaN	NaN	NaN	2019-05-03 11:46:38	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019-05-03 11:40:20	NaN
2	3	NaN	NaN	NaN	2019-05-03 12:31:06	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019-05-03 12:09:40	NaN
3	4	NaN	NaN	NaN	2019-05-03 12:48:43	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019-05-03 12:41:24	NaN
4	5	NaN	NaN	NaN	2019-05-03 13:18:50	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2019-05-03 13:12:56	NaN

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3129 entries, 0 to 3128
Data columns (total 16 columns):
key         3129 non-null int64
Bulk 1      252 non-null object
Bulk 2      22 non-null object
Bulk 3      1298 non-null object
Bulk 4      1014 non-null object
Bulk 5      77 non-null object
Bulk 6      576 non-null object
Bulk 7      25 non-null object
Bulk 8      1 non-null object
Bulk 9      19 non-null object
Bulk 10     176 non-null object
Bulk 11     177 non-null object
Bulk 12     2450 non-null object
Bulk 13     18 non-null object
Bulk 14     2806 non-null object
Bulk 15     2248 non-null object
dtypes: int64(1), object(15)
memory usage: 391.2+ KB
```

In [14]:
```
print(len(data_bulk_time['key'].unique()))
print(len(data_bulk_time['key']))
```

```
3129
3129
```

Промежуточный вывод:

В файлах с данными о подаче сыпучих материалов содержится информация об объёме и времени различных сыпучих материалов для 3129 партий. Всего 3129 строк - т.е. одна строка соответствует одной партии.

По данным таблицам можно оценить отдельные добавки и общий объем сыпучих материалов, содержащихся в париях в анализируемый момент времени.

In [15]:
```
data_gas.head()
```

Out[15]:

	key	Газ 1
0	1	29.749986
1	2	12.555561
2	3	28.554793
3	4	18.841219
4	5	5.413692

In [16]:
```
data_gas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3239 entries, 0 to 3238
Data columns (total 2 columns):
key       3239 non-null int64
Газ 1     3239 non-null float64
dtypes: float64(1), int64(1)
memory usage: 50.7 KB
```

In [17]:
```
print(len(data_gas['key'].unique()))
print(len(data_gas['key']))
```

```
3239
3239
```

Промежуточный вывод:
В файлах с данными о продувке сплава инертным газом содержится некоторая информация о газе для 3239 партий. Всего 3239 строк - т.е. одна строка соответствует одной партии.
Вероятно значения содержат некоторый количественный показатель. Т.к. "... основной целью продувки металла инертными газами является его дегазация, удаление неметаллических включений, выравнивание химического состава и температуры по всему объему металла..." эти данные вероятно не влияют на химический состав, однако могут влиять на затраты электроенергии.

In [18]:
```
data_temp.head()
```

Out[18]:

	key	Время замера	Температура
0	1	2019-05-03 11:16:18	1571.0
1	1	2019-05-03 11:25:53	1604.0
2	1	2019-05-03 11:29:11	1618.0
3	1	2019-05-03 11:30:01	1601.0
4	1	2019-05-03 11:30:39	1613.0

In [19]:
```
data_temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15907 entries, 0 to 15906
Data columns (total 3 columns):
key              15907 non-null int64
Время замера     15907 non-null object
Температура      13006 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 372.9+ KB
```

In [20]:
```
print(len(data_temp['key'].unique()))
print(len(data_temp['key']))
```

```
3216
15907
```

Промежуточный вывод:

В файле с данными о результатах измерения температуры содержится информация о температуре сплава и времени работы измерения для 3216 партий. Всего 15907 строк. Есть пропуски. По данным можно оценить температуру сплава партии на различных стадиях, финальную (предположительно целевую) температуру сплава партий.

In [21]:
```
data_wire.head()
```

Out[21]:

	key	Wire 1	Wire 2	Wire 3	Wire 4	Wire 5	Wire 6	Wire 7	Wire 8	Wire 9
0	1	60.059998	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2	96.052315	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	3	91.160157	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	89.063515	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	5	89.238236	9.11456	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [22]:
```
data_wire.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3081 entries, 0 to 3080
Data columns (total 10 columns):
key       3081 non-null int64
Wire 1    3055 non-null float64
Wire 2    1079 non-null float64
Wire 3    63 non-null float64
Wire 4    14 non-null float64
Wire 5    1 non-null float64
Wire 6    73 non-null float64
Wire 7    11 non-null float64
Wire 8    19 non-null float64
Wire 9    29 non-null float64
dtypes: float64(9), int64(1)
memory usage: 240.8 KB
```

In [23]:
```
print(len(data_wire['key'].unique()))
print(len(data_wire['key']))
```

```
3081
3081
```

In [24]:
```
data_wire_time.head()
```

Out[24]:

	key	Wire 1	Wire 2	Wire 3	Wire 4	Wire 5	Wire 6	Wire 7	Wire 8	Wire 9
0	1	2019-05-03 11:11:41	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2	2019-05-03 11:46:10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	3	2019-05-03 12:13:47	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	2019-05-03 12:48:05	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	5	2019-05-03 13:18:15	2019-05-03 13:32:06	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [25]:
```
data_wire_time.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3081 entries, 0 to 3080
Data columns (total 10 columns):
key       3081 non-null int64
Wire 1    3055 non-null object
Wire 2    1079 non-null object
Wire 3    63 non-null object
Wire 4    14 non-null object
Wire 5    1 non-null object
Wire 6    73 non-null object
Wire 7    11 non-null object
Wire 8    19 non-null object
Wire 9    29 non-null object
dtypes: int64(1), object(9)
memory usage: 240.8+ KB
```

In [26]:
```
print(len(data_wire_time['key'].unique()))
print(len(data_wire_time['key']))
```

```
3081
3081
```

Промежуточный вывод:

В файлах с данными о проволочных материалах содержится время и объем различных проволочных материалов для 3081 партий. Всего 3081 строк - т.е. одна строка соответствует одной партии.

По данным таблицам можно оценить общий объем проволочных материалов, содержащихся в париях в анализируемый момент времени.

Вывод:

В файле с данными об электродах содержится информация о мощности и времени работы электродов для 3214 партий. Всего 14876 строк. Пропусков нет. По данным можно оценить затраты электроенергии на партии.

В файлах с данными о подаче сыпучих материалов содержится информация об объёме и времени различных сыпучих материалов для 3129 партий. Всего 3129 строк - т.е. одна строка соответствует одной партии.

По данным таблицам можно оценить общий объем сыпучих материалов, содержащихся в париях в анализируемый момент времени.

В файлах с данными о продувке сплава инертным газом содержится некоторая информация о газе для 3239 партий. Всего 3239 строк - т.е. одна строка соответствует одной партии.
Вероятно значения содержат некоторый количественный показатель. Т.к. "... основной целью продувки металла инертными газами является его дегазация, удаление неметаллических включений, выравнивание химического состава и температуры по всему объему металла..." эти данные вероятно не влияют на химический состав, однако могут влиять на затраты электроенергии.

В файле с данными о результатах измерения температуры содержится информация о температуре сплава и времени работы измерения для 3216 партий. Всего 15907 строк. Есть пропуски. По данным можно оценить температуру сплава партии на различных стадиях, финальну (предположительно целевую) температуру сплава партий.

В файлах с данными о проволочных материалах содержится время и объем различных проволочных материалов для 3081 партий. Всего 3081 строк - т.е. одна строка соответствует одной партии.
По данным таблицам можно оценить общий объем проволочных материалов, содержащихся в париях в анализируемый момент времени.

Анализ данных об измерениях температуры

Обработка измерений

In [27]:
```
data_temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15907 entries, 0 to 15906
Data columns (total 3 columns):
key             15907 non-null int64
Время замера    15907 non-null object
Температура     13006 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 372.9+ KB
```

In [28]:
```
data_temp['Температура'].isna().sum()
```

Out[28]:
```
2901
```

В данных об измерениях температуры сплава 2901 пропуск

In [29]:
```
len(data_temp['key'].unique())
```

Out[29]:
```
3216
```

В данных об измерениях температуры сплава содержится информация о 3216 партиях

In [30]:
```
print(len(data_temp.dropna()['Время замера']))
print(len(data_temp.dropna()['Время замера'].unique()))
```

```
13006
13006
```

В данных об измерениях температуры сплава время всех измерений уникально

In [31]:
```
# список партий, имеющих пропуски в значениях температуры
a=list(data_temp[data_temp['Температура'].isna()]['key'].values)
```

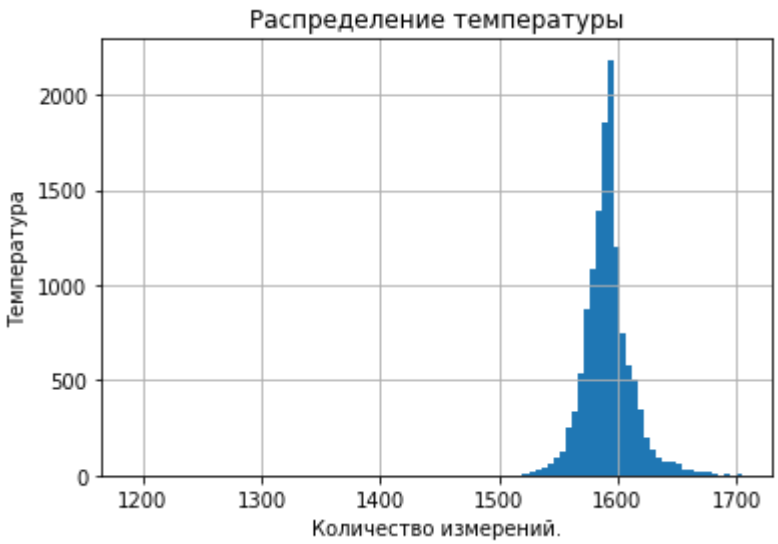
In [32]:
```
# срез партий, имеющих пропуски в значениях температуры, группировка по номе
# и агрегация по максимальному значению температуры
# суммирование пропусков

data_temp.query('key in @a').groupby('key')['Температура'].max().isna().sum(
```

Out[32]:
```
0
```

Анализ пропусков показал, что для каждой партии в таблице есть хотя бы одно значение температуры

In [33]:
```
data_temp_prep=data_temp.dropna().reset_index(drop=True)
```

In [34]:
```
data_temp['Температура'].hist(bins=100)
plt.xlabel('Количество измерений.')
plt.ylabel('Температура')
plt.grid(True)
plt.title('Распределение температуры')
plt.show()
```



In [35]:
```
data_temp_prep.describe()
```

Out[35]:

	key	Температура
count	13006.000000	13006.000000
mean	1328.447793	1591.840920
std	804.740001	21.375851
min	1.000000	1191.000000
25%	638.000000	1581.000000
50%	1315.000000	1591.000000
75%	1977.750000	1601.000000
max	3241.000000	1705.000000

Создание таблицы с финальными температурами

In [36]:
```
data_temp_last=data_temp_prep.pivot_table(index='key',values=['Время замера'
                                'Температура': 'last'})
data_temp_last.columns=['time','temp']
data_temp_last
```

Out[36]:

key	time	temp
1	2019-05-03 11:30:39	1613.0
2	2019-05-03 11:59:12	1602.0
3	2019-05-03 12:34:57	1599.0
4	2019-05-03 12:59:25	1625.0
5	2019-05-03 13:36:01	1602.0
...	...	...
3237	2019-08-31 22:44:04	1569.0
3238	2019-08-31 23:30:31	1584.0
3239	2019-09-01 01:31:47	1598.0
3240	2019-09-01 02:39:01	1617.0
3241	2019-09-01 04:03:30	1586.0

3216 rows × 2 columns

Создание таблицы с исходными температурами

In [37]:
```
data_temp_first=data_temp_prep.pivot_table(index='key',values=['Время замера
                                'Температура': 'first',
                                 'key':'count'})
data_temp_first.columns=['iter','time','temp']
data_temp_first
```

Out[37]:

key	iter	time	temp
1	5	2019-05-03 11:16:18	1571.0
2	6	2019-05-03 11:37:27	1581.0
3	5	2019-05-03 12:13:17	1596.0
4	3	2019-05-03 12:52:57	1601.0
5	2	2019-05-03 13:23:19	1576.0
...	...	...	...
3237	1	2019-08-31 22:44:04	1569.0
3238	1	2019-08-31 23:30:31	1584.0
3239	1	2019-09-01 01:31:47	1598.0
3240	1	2019-09-01 02:39:01	1617.0
3241	1	2019-09-01 04:03:30	1586.0

3216 rows × 3 columns

Ниже долгий алгоритм создающий таблицу с споследними температурами для каждой партии для случая не отсортированных по времени данных проверка эквивалентности пердыдущей таблице

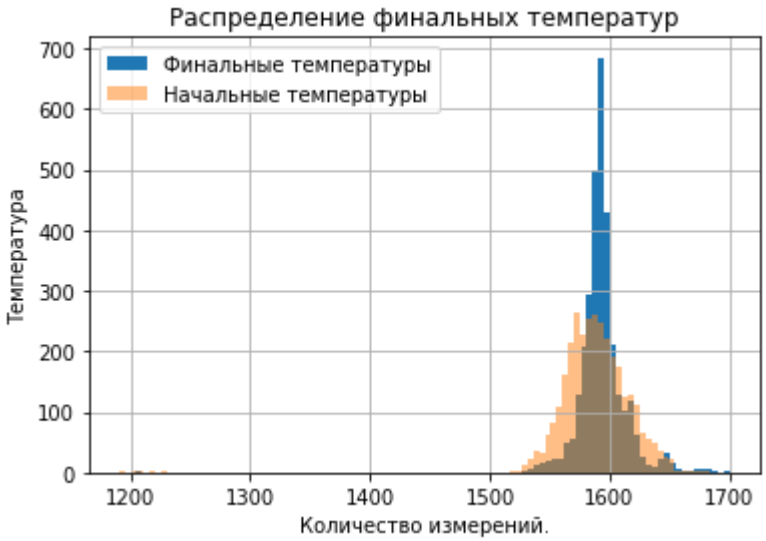
In [38]:
```
# data_temp_gr=data_temp_prep.groupby('key')['Время замера'].max()
```

In [39]:
```
# def final_t(key):
#     return data_temp_prep[data_temp_prep['Время замера']==data_temp_gr[key
```

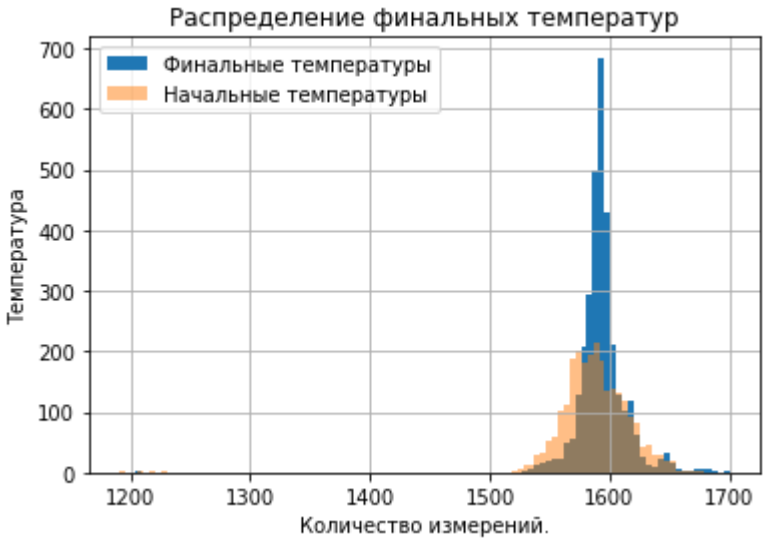
In [40]:
```
# data_temp_last_=data_temp_prep.pivot_table(index='key',values=['Время заме
# data_temp_last_['key']=data_temp_last_.index

# data_temp_last_['temp']=data_temp_last_['key'].apply(final_t)
```

In [41]:
```
# import numpy as np
# np.sum(data_temp_last_['temp']==data_temp_last['Температура'])
```

In [42]:
```
data_temp_last['temp'].hist(bins=100)
data_temp_first['temp'].hist(bins=100,alpha=0.5)
plt.xlabel('Количество измерений.')
plt.ylabel('Температура')
plt.grid(True)
plt.title('Распределение финальных температур')
plt.legend(['Финальные температуры','Начальные температуры'])
plt.show()
```



In [43]:
```
data_temp_last['temp'].hist(bins=100)
data_temp_first[data_temp_first['iter']>1]['temp'].hist(bins=100,alpha=0.5)
plt.xlabel('Количество измерений.')
plt.ylabel('Температура')
plt.grid(True)
plt.title('Распределение финальных температур')
plt.legend(['Финальные температуры','Начальные температуры'])
plt.show()
```



Вывод:

В данных об измерениях температуры сплава:

- 2901 пропуск
- содержится информация о 3216 партиях
- время всех измерений уникально

  Анализ пропусков показал, что для каждой партии в таблице есть хотя бы одно значение температуры

  Получены таблицы с начальными и финальными температурами сплава для каждой партии

  Распределения финальных температур имеет меньше дисперсию чем распределение исходных температур. Медианные значения близки.

Объединение таблиц. Создание признаков

Создание таблицы со всеми измерениями температуры и количеством загруженных примесей

In [44]:
```
def my_row(row):
    s=0
    for i in range(3,3+(len(row)-3)//2):
        if not(row[i]==0):
            if row[i]<row[1]:
                s+=row[i+(len(row)-3)//2]
            else:
                row[i+(len(row)-3)//2]=0
    row[-1]=s
    return row
```

Добавление признаков - проволочный компонент (для каждого вида компонента) и суммарное значение

In [45]:
```
data_wire_time=data_wire_time.fillna(0)
data_wire=data_wire.fillna(0)
df1=data_temp_prep.merge(data_wire_time,on='key',how='left')
df1=df1.merge(data_wire,on='key',how='left')
df1=df1.fillna(0)
```

In [46]:
```
df1['wire']=0
df1=df1.apply(my_row,axis=1)
```

In [47]:
```
col=data_wire_time.columns
col_x=[s+'_x' for s in col[1:]]
df1.drop([*col_x], axis=1, inplace=True)
df1=df1.reset_index(drop=True)
```

Добавление признаков - сыпучий компонент (для каждого вида компонента) и суммарное значение

In [48]:
```
data_bulk_time=data_bulk_time.fillna(0)
data_bulk=data_bulk.fillna(0)
df2=data_temp_prep.merge(data_bulk_time,on='key',how='left')
df2=df2.merge(data_bulk,on='key',how='left')
df2=df2.fillna(0)
```

In [49]:
```
df2['bulk']=0
df2=df2.apply(my_row,axis=1)
```

In [50]:
```
col=data_bulk_time.columns
col_x=[s+'_x' for s in col[1:]]
df2.drop([*col_x], axis=1, inplace=True)
df2=df2.reset_index(drop=True)
```

In [51]:
```
df2_col=df2.columns[3:]
df2_col=['Время замера']+[*df2_col]
```

In [52]:
```
df=df1.merge(df2.loc[:,df2_col], on='Время замера', how='left')
```

Добавление признаков - начальная температура, время итерации

In [53]:
```
df['start_t']=df['key'].apply(lambda x:  data_temp_first.loc[x]['temp'])
df['start_time']=df['key'].apply(lambda x:  data_temp_first.loc[x]['time'])
```

In [54]:
```
df['Температура']=df['Температура'].astype('int')
df['start_t']=df['start_t'].astype('int')
```

In [55]:
```
df['Время замера']=pd.to_datetime(df['Время замера'])
df['start_time']=pd.to_datetime(df['start_time'])
```

In [56]:
```
df['process_time']=(df['Время замера']-df['start_time']).dt.seconds
```

In [57]:
```
df['process_time']=df['process_time'].astype('int')
```

обработка признаков добавок - удаление добавок добавленных до начала измерений температуры

In [58]:
```
ind=df.loc[df['Температура']==df['start_t']].index
# ind
cols=df.columns[3:-3]
cols
for col in cols:
    df['odd']=0.0
    df.loc[ind,['odd']]=df.loc[ind][col]
    gr=df.groupby('key')['odd'].max()
    df['odd']=df.apply(lambda x:gr[x['key']],axis=1)
    df[col]=df[col]-df['odd']
```

In [59]:
```
df_big=df.drop(['odd'],axis=1)
df_big= df_big.drop(df_big[df_big['Температура']==df_big['start_t']].index,a
# df_big.columns=['final_t','wire','bulk','start_t']
df_big=df_big.reset_index(drop=True)
```

Добавление признака - газ

In [60]:
```
data_gas.columns=['key','gas']
```

In [61]:
```
df_big=df_big.merge(data_gas,on='key',how='left')
```

Добавление признака - мощность

In [62]:
```
data_arc.columns=['key','start','finish','WA','WR']
```

In [63]:
```
data_arc['start']=pd.to_datetime(data_arc['start'])
data_arc['finish']=pd.to_datetime(data_arc['finish'])
```

In [64]:
```
def power_calc(row):
    t1=row["start_time"]
    t2=row["Время замера"]
    p=data_arc.query('start>@t1 and finish<@t2')['WA'].sum()
    row[-1]=p
    return row
```

In [65]:
```
df_big['power']=0
df_big=df_big.apply(power_calc,axis=1)
```

Добавление признака - финальная итерация

In [66]:
```
df_big['final_t']=df_big['key'].apply(lambda x:  data_temp_last.loc[x]['temp
df_big['final_t']=df_big['final_t'].astype('int')
df_big['is_final']=0
df_big.loc[df_big['final_t']==df_big['Температура'],['is_final']]=1
df_big=df_big.drop('final_t',axis=1)
```

Получившаяся таблица

In [67]:
```
df_big.columns=[''.join(ss.split()) for ss in [s.lower() for s in df_big.col
print(df_big.columns)
print(len(df_big.columns))
```

```
Index(['key', 'времязамера', 'температура', 'wire1_y', 'wire2_y', 'wire3_y',
       'wire4_y', 'wire5_y', 'wire6_y', 'wire7_y', 'wire8_y', 'wire9_y',
       'wire', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y', 'bulk5_y',
       'bulk6_y', 'bulk7_y', 'bulk8_y', 'bulk9_y', 'bulk10_y', 'bulk11_y',
       'bulk12_y', 'bulk13_y', 'bulk14_y', 'bulk15_y', 'bulk', 'start_t',
       'start_time', 'process_time', 'gas', 'power', 'is_final'],
      dtype='object')
35
```

In [68]:
```
df_big=df_big.reset_index(drop=True)
df_big.columns=['key', 'time', 'temp', 'wire1_y', 'wire2_y', 'wire3_y',
       'wire4_y', 'wire5_y', 'wire6_y', 'wire7_y', 'wire8_y', 'wire9_y',
       'wire', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y', 'bulk5_y',
       'bulk6_y', 'bulk7_y', 'bulk8_y', 'bulk9_y', 'bulk10_y', 'bulk11_y',
       'bulk12_y', 'bulk13_y', 'bulk14_y', 'bulk15_y', 'bulk',
       'start_t', 'start_time', 'process_time', 'gas', 'power', 'is_final']
df_big.head()
```

Out[68]:

	key	time	temp	wire1_y	wire2_y	wire3_y	wire4_y	wire5_y	wire6_y	wire7_y	...	b
0	1	2019-05-03 11:25:53	1604	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
1	1	2019-05-03 11:29:11	1618	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
2	1	2019-05-03 11:30:01	1601	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
3	1	2019-05-03 11:30:39	1613	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
4	2	2019-05-03 11:38:00	1577	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

5 rows × 35 columns

Создание таблицы с финальным значением температуры и соответствующими ему количеством загруженных примесей и исходной температуры сплава

In [69]:
```
col_max=df_big.columns[3:-6]
col_sum=['power','process_time']
cols=[*col_max]+[*col_sum]+[*['time', 'temp','gas','start_t']]
col_d={c:'max' for c in col_max}
```

```
col_d['power']='sum'
col_d['process_time']='sum'
col_d['time']='max'
col_d['temp']='last'
col_d['gas']='first'
col_d['start_t']='first'
```

In [70]:
```
df_small=df_big.pivot_table(index='key',values=cols,aggfunc=col_d)
```

In [71]:
```
print(df_small.columns)
print(len(df_small.columns))
```

```
Index(['bulk', 'bulk10_y', 'bulk11_y', 'bulk12_y', 'bulk13_y', 'bulk14_y',
       'bulk15_y', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y', 'bulk5_y',
       'bulk6_y', 'bulk7_y', 'bulk8_y', 'bulk9_y', 'gas', 'power',
       'process_time', 'start_t', 'temp', 'time', 'wire', 'wire1_y', 'wire2_
y',
       'wire3_y', 'wire4_y', 'wire5_y', 'wire6_y', 'wire7_y', 'wire8_y',
       'wire9_y'],
      dtype='object')
32
```

In [72]:
```
df_small=df_small.reset_index(drop=True)
df_small
```

Out[72]:

	bulk	bulk10_y	bulk11_y	bulk12_y	bulk13_y	bulk14_y	bulk15_y	bulk1_y	bulk2_y	b
0	43.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	582.0	0.0	0.0	206.0	0.0	149.0	154.0	0.0	0.0	
2	34.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
2467	111.0	90.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2468	796.0	122.0	0.0	256.0	0.0	129.0	226.0	0.0	0.0	
2469	85.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2470	191.0	101.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2471	47.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

2472 rows × 32 columns

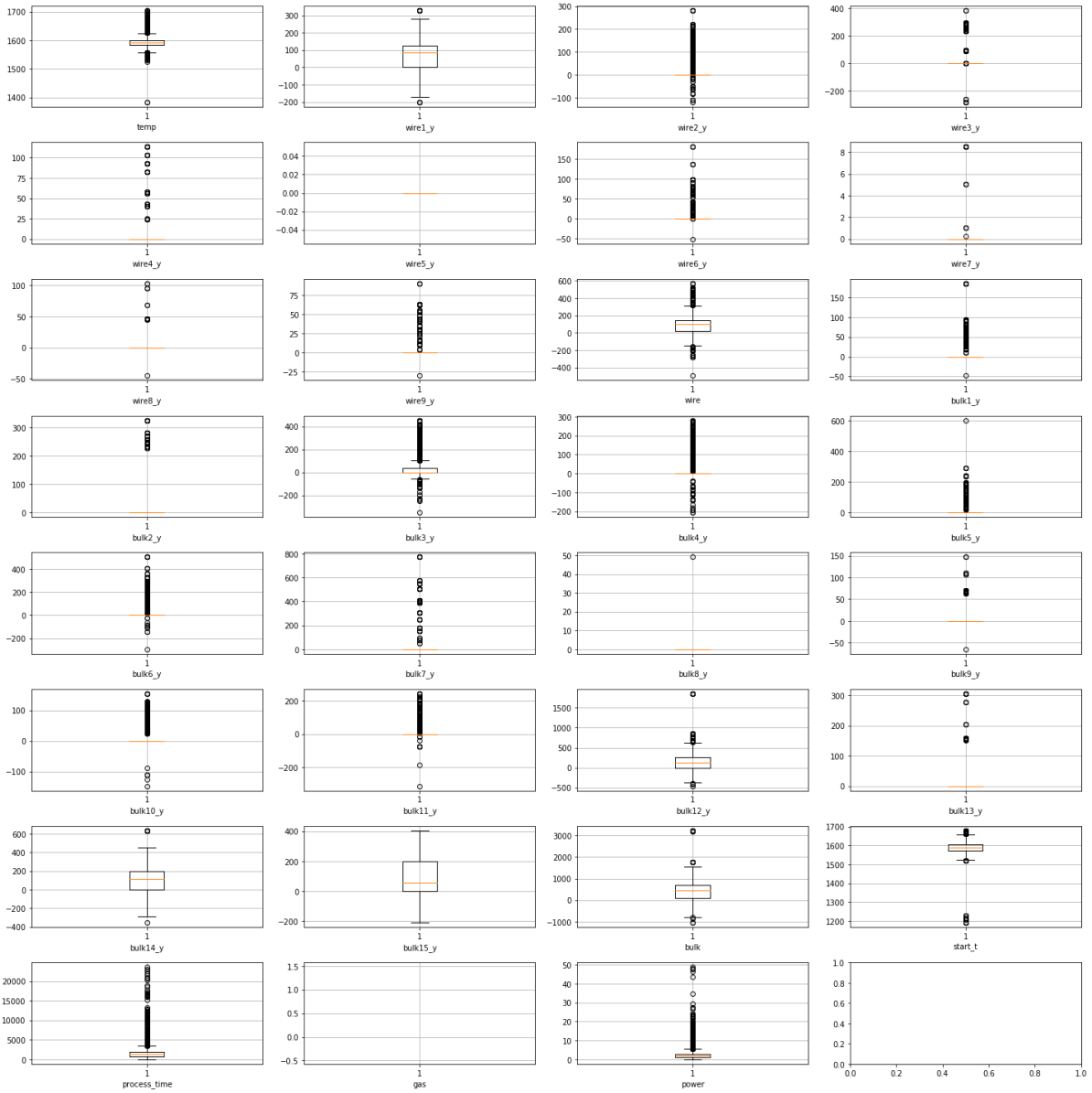
Анализ признаков

Таблица df_big

In [73]:
```
df_big.columns
```

Out[73]:
```
Index(['key', 'time', 'temp', 'wire1_y', 'wire2_y', 'wire3_y', 'wire4_y',
       'wire5_y', 'wire6_y', 'wire7_y', 'wire8_y', 'wire9_y', 'wire',
       'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y', 'bulk5_y', 'bulk6_y',
       'bulk7_y', 'bulk8_y', 'bulk9_y', 'bulk10_y', 'bulk11_y', 'bulk12_y',
       'bulk13_y', 'bulk14_y', 'bulk15_y', 'bulk', 'start_t', 'start_time',
       'process_time', 'gas', 'power', 'is_final'],
      dtype='object')
```

In [74]:
```
s1, s2=8,4
fig, axs = plt.subplots(s1, s2,figsize=(20,20))
k=0
col_name=['temp', 'wire1_y', 'wire2_y', 'wire3_y', 'wire4_y',
       'wire5_y', 'wire6_y', 'wire7_y', 'wire8_y', 'wire9_y', 'wire',
       'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y', 'bulk5_y', 'bulk6_y',
       'bulk7_y', 'bulk8_y', 'bulk9_y', 'bulk10_y', 'bulk11_y', 'bulk12_y',
       'bulk13_y', 'bulk14_y', 'bulk15_y', 'bulk', 'start_t',
       'process_time', 'gas', 'power']

for i in range(0,s1):
    for j in range(0,s2):
        axs[i,j].boxplot(x=df_big[col_name[k]])
        axs[i,j].set_xlabel(col_name[k])
        axs[i,j].grid(True)
        k+=1
        if k>30:
            break

fig.tight_layout()
plt.show()
```



In [75]:
```
for k in range(len(col_name)):
    print(col_name[k],': ',df_big.loc[df_big[col_name[k]]>0][col_name[k]].co
```

```
temp :  9485
wire1_y :  7012
wire2_y :  2309
wire3_y :  126
wire4_y :  64
wire5_y :  0
wire6_y :  184
wire7_y :  15
wire8_y :  29
wire9_y :  87
wire :  7314
bulk1_y :  636
bulk2_y :  81
bulk3_y :  2705
bulk4_y :  2260
bulk5_y :  208
bulk6_y :  1320
bulk7_y :  86
bulk8_y :  1
bulk9_y :  28
bulk10_y :  398
bulk11_y :  374
bulk12_y :  5196
bulk13_y :  60
bulk14_y :  5972
bulk15_y :  4780
bulk :  7699
start_t :  9485
process_time :  9485
gas :  9477
power :  8808
```

In [76]:
```
cols=['bulk', 'gas', 'power','process_time', 'start_t', 'temp', 'wire']
for col in cols:
    df_big=df_big.drop(df_big[df_big[col]>df_big[col].quantile(0.99)].index,
    df_big=df_big.drop(df_big[df_big[col]<df_big[col].quantile(0.01)].index,
```

In [77]:
```
df_big
```

Out[77]:

	key	time	temp	wire1_y	wire2_y	wire3_y	wire4_y	wire5_y	wire6_y	wire
0	1	2019-05-03 11:25:53	1604	0.000000	0.00000	0.0	0.0	0.0	0.0	
1	1	2019-05-03 11:29:11	1618	0.000000	0.00000	0.0	0.0	0.0	0.0	
2	1	2019-05-03 11:30:01	1601	0.000000	0.00000	0.0	0.0	0.0	0.0	
3	1	2019-05-03 11:30:39	1613	0.000000	0.00000	0.0	0.0	0.0	0.0	
4	2	2019-05-03 11:49:38	1589	96.052315	0.00000	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
8430	2498	2019-08-06 02:12:00	1580	118.110717	0.00000	0.0	0.0	0.0	0.0	
8431	2498	2019-08-06 02:19:26	1593	118.110717	0.00000	0.0	0.0	0.0	0.0	
8432	2498	2019-08-06 02:25:31	1594	118.110717	0.00000	0.0	0.0	0.0	0.0	
8433	2499	2019-08-06 02:54:24	1604	0.000000	50.00528	0.0	0.0	0.0	0.0	
8434	2499	2019-08-06 03:00:28	1603	0.000000	50.00528	0.0	0.0	0.0	0.0	

8435 rows × 35 columns

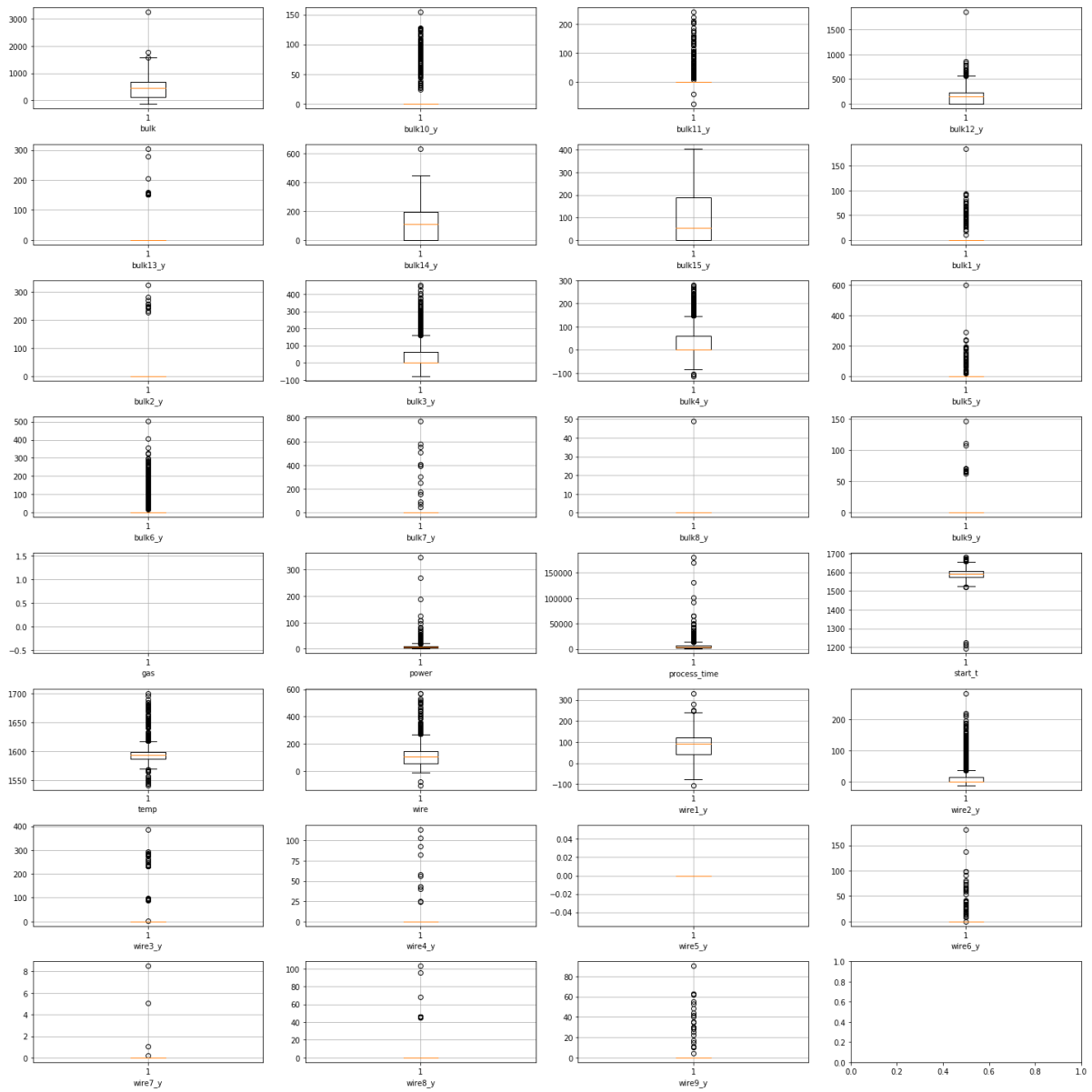
Таблица df_small

In [78]:
```
df_small.columns
```

Out[78]:
```
Index(['bulk', 'bulk10_y', 'bulk11_y', 'bulk12_y', 'bulk13_y', 'bulk14_y',
       'bulk15_y', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y', 'bulk5_y',
       'bulk6_y', 'bulk7_y', 'bulk8_y', 'bulk9_y', 'gas', 'power',
       'process_time', 'start_t', 'temp', 'time', 'wire', 'wire1_y', 'wire2_
y',
       'wire3_y', 'wire4_y', 'wire5_y', 'wire6_y', 'wire7_y', 'wire8_y',
       'wire9_y'],
      dtype='object')
```

In [79]:
```
s1, s2=8,4
fig, axs = plt.subplots(s1, s2,figsize=(20,20))
k=0
col_name=['bulk', 'bulk10_y', 'bulk11_y', 'bulk12_y', 'bulk13_y', 'bulk14_y'
       'bulk15_y', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y', 'bulk5_y',
       'bulk6_y', 'bulk7_y', 'bulk8_y', 'bulk9_y', 'gas', 'power',
       'process_time', 'start_t', 'temp', 'wire', 'wire1_y', 'wire2_y',
       'wire3_y', 'wire4_y', 'wire5_y', 'wire6_y', 'wire7_y', 'wire8_y',
```

```
        'wire9_y']

for i in range(0,s1):
    for j in range(0,s2):
        axs[i,j].boxplot(x=df_small[col_name[k]])
        axs[i,j].set_xlabel(col_name[k])
        axs[i,j].grid(True)
        k+=1
        if k>30:
            break

fig.tight_layout()
plt.show()
```



In [80]:
```
for k in range(len(col_name)):
    print(col_name[k],': ',df_small.loc[df_small[col_name[k]]>0][col_name[k]
```

```
bulk :  2106
bulk10_y :  130
bulk11_y :  111
bulk12_y :  1372
bulk13_y :  13
bulk14_y :  1564
bulk15_y :  1248
bulk1_y :  153
bulk2_y :  12
bulk3_y :  873
bulk4_y :  741
bulk5_y :  48
bulk6_y :  379
bulk7_y :  14
bulk8_y :  1
bulk9_y :  12
gas :  2470
power :  2438
process_time :  2472
start_t :  2472
temp :  2472
wire :  2078
wire1_y :  1989
wire2_y :  733
wire3_y :  34
wire4_y :  11
wire5_y :  0
wire6_y :  41
wire7_y :  4
wire8_y :  12
wire9_y :  26
```

In [81]:
```
cols=['bulk', 'gas', 'power','process_time', 'start_t', 'temp', 'wire']
for col in cols:
    df_small=df_small.drop(df_small[df_small[col]>df_small[col].quantile(0.9
    df_small=df_small.drop(df_small[df_small[col]<df_small[col].quantile(0.0
```

In [82]:
```
df_small
```

Out[82]:

	bulk	bulk10_y	bulk11_y	bulk12_y	bulk13_y	bulk14_y	bulk15_y	bulk1_y	bulk2_y	b
0	43.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	582.0	0.0	0.0	206.0	0.0	149.0	154.0	0.0	0.0	
2	34.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
2204	111.0	90.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2205	796.0	122.0	0.0	256.0	0.0	129.0	226.0	0.0	0.0	
2206	85.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2207	191.0	101.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2208	47.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

2209 rows × 32 columns

Вывод:

Созданы 2 таблицы:

- Таблица df_small содержит финальные температуры для каждой партии и соответствующие им значения стартовой температуры и количества примесей, а также газ, используемый для данной партии
- Таблица df_big содержит все значения температуры (кроме стартовых) и соответствующие им значения стартовой температуры и количества примесей к текущему моменту времени добавленных в состав*.

Таблица df_small содержит 1974 сэмпла, в т.ч. данные по газу. Таблица df_big содержит 8746 сэмпла.

*Идея создания таблицы df_big заключается в существенном увеличении количества сэмплов без дублирования за счет использования промежуточных измерений. Промежуточные измерения принимаются как финальные и используются для установления влияния различных факторов на температуру сплава. При этом утечки информации не происходит, т.к. сами значения температуры в промежуточных стадиях не используются как признаки. Предполагается, что такой подход позволит расширить диапазон значений целевого и признакового пространства и тем самым уменьшить эффект переобучения.

Построение моделей

In [83]:
```
Models_train_score=pd.DataFrame(columns=['Dummy_small','DT_small','RF_small'
Models_test_score=pd.DataFrame(columns=['Dummy_small','DT_small','RF_small',
```

In [84]:
```
class Dummy_model_mean:
    def fit(self, train, target):
        self.y = target.mean()

    def predict(self, test):
        return np.zeros(test.shape[0])+(self.y)
```

In [85]:
```
def sMAPE(pred, targ):
    """Рассчитывает значения метрики sMAPE

    pred - предсказания модели
    targ - реальные значения

    Возвращает:
    значения метрики sMAPE
    """
    return 100/len(pred) * np.sum(2 * np.abs(pred - targ) / (np.abs(pred) +
```

In [86]:
```
# score = make_scorer(sMAPE, greater_is_better=False)
```

Таблица df_small

In [87]:
```
df_small.dropna(inplace=True)
df_small=df_small.reset_index(drop=True)
```

```
df_small.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2207 entries, 0 to 2206
Data columns (total 32 columns):
bulk             2207 non-null float64
bulk10_y         2207 non-null float64
bulk11_y         2207 non-null float64
bulk12_y         2207 non-null float64
bulk13_y         2207 non-null float64
bulk14_y         2207 non-null float64
bulk15_y         2207 non-null float64
bulk1_y          2207 non-null float64
bulk2_y          2207 non-null float64
bulk3_y          2207 non-null float64
bulk4_y          2207 non-null float64
bulk5_y          2207 non-null float64
bulk6_y          2207 non-null float64
bulk7_y          2207 non-null float64
bulk8_y          2207 non-null float64
bulk9_y          2207 non-null float64
gas              2207 non-null float64
power            2207 non-null float64
process_time     2207 non-null int64
start_t          2207 non-null int64
temp             2207 non-null int64
time             2207 non-null datetime64[ns]
wire             2207 non-null float64
wire1_y          2207 non-null float64
wire2_y          2207 non-null float64
wire3_y          2207 non-null float64
wire4_y          2207 non-null float64
wire5_y          2207 non-null float64
wire6_y          2207 non-null float64
wire7_y          2207 non-null float64
wire8_y          2207 non-null float64
wire9_y          2207 non-null float64
dtypes: datetime64[ns](1), float64(28), int64(3)
memory usage: 551.9 KB
```

In [88]:
```
f_train_s, f_test_s, t_train_s, t_test_s =train_test_split(df_small.drop(['t
                                                           df_small['temp'],
```

In [89]:
```
col1=f_train_s.columns
```

In [90]:
```
# преобразования признаков StandardScaler
scaler1 = ColumnTransformer(transformers=[
    ('std', StandardScaler(),col1),
], remainder='passthrough')
```

DecisionTreeRegressor

In [91]:
```
r=17
pipe_r = Pipeline(steps=[('scale', scaler1),  # Трансформеры
                         ('DT', DecisionTreeRegressor(random_state=r))])

pipe_params = {'DT__criterion': ['mse', 'mae'],
               'DT__max_depth': range(1,20)}
prs = RandomizedSearchCV(pipe_r, pipe_params, cv=5, n_iter=10, scoring='neg_

prs.fit(f_train_s,t_train_s)
```

```
# print(sMAPE(prs.predict(f_test_s),t_test_s))
print(abs(prs.best_score_))
prs.best_params_
```

Out[91]:
```
103.89622356495468
{'DT__max_depth': 5, 'DT__criterion': 'mae'}
```

In [92]:
```
Models_train_score.loc['cv_best_score',['DT_small']]=abs(prs.best_score_)
Models_train_score.loc['mae',['DT_small']]=mean_absolute_error(prs.predict(f
Models_train_score.loc['MAPE',['DT_small']]=sMAPE(prs.predict(f_train_s),t_t
Models_train_score.loc['mse',['DT_small']]= mean_squared_error(prs.predict(f
```

RandomForestRegressor

RandomForestRegressor - подбор гиперпараметров, оценка признаков

In [93]:
```
r=17
pipe_r = Pipeline(steps=[('scale', scaler1),  # Трансформеры
                         ('RF', RandomForestRegressor(random_state=r))])

pipe_params = {'RF__criterion':  ['mse', 'mae'],
               'RF__max_depth': range(3,20),
               'RF__n_estimators': range(100,200)}
prs = RandomizedSearchCV(pipe_r, pipe_params, cv=5, n_iter=10, scoring='neg_
prs.fit(f_train_s,t_train_s)

# print(sMAPE(prs.predict(f_test_s),t_test_s))
print(abs(prs.best_score_))
prs.best_params_
```

Out[93]:
```
68.96908244408331
{'RF__n_estimators': 137, 'RF__max_depth': 16, 'RF__criterion': 'mse'}
```

In [94]:
```
Models_train_score.loc['cv_best_score',['RF_small']]=abs(prs.best_score_)
```

In [95]:
```
std_scale=StandardScaler()
```

In [96]:
```
RF_model=RandomForestRegressor(random_state=r,criterion= 'mse', max_depth= 1
f_train_s_scale=std_scale.fit_transform(f_train_s)
RF_model.fit(f_train_s_scale,t_train_s)
```

Out[96]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=16,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=137,
                      n_jobs=None, oob_score=False, random_state=17, verbose
=0,
                      warm_start=False)
```

In [97]:
```
importance = RF_model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %s, Score: %.5f' % (f_train_s.columns[i],v))
```

```
Feature: bulk, Score: 0.03822
Feature: bulk10_y, Score: 0.00468
Feature: bulk11_y, Score: 0.00329
Feature: bulk12_y, Score: 0.02617
Feature: bulk13_y, Score: 0.00002
Feature: bulk14_y, Score: 0.03874
Feature: bulk15_y, Score: 0.01602
Feature: bulk1_y, Score: 0.00349
Feature: bulk2_y, Score: 0.00003
Feature: bulk3_y, Score: 0.01578
Feature: bulk4_y, Score: 0.01557
Feature: bulk5_y, Score: 0.00131
Feature: bulk6_y, Score: 0.04162
Feature: bulk7_y, Score: 0.00003
Feature: bulk8_y, Score: 0.00000
Feature: bulk9_y, Score: 0.00013
Feature: gas, Score: 0.05029
Feature: power, Score: 0.15898
Feature: process_time, Score: 0.08851
Feature: start_t, Score: 0.23454
Feature: wire, Score: 0.13214
Feature: wire1_y, Score: 0.09967
Feature: wire2_y, Score: 0.02874
Feature: wire3_y, Score: 0.00063
Feature: wire4_y, Score: 0.00025
Feature: wire5_y, Score: 0.00000
Feature: wire6_y, Score: 0.00087
Feature: wire7_y, Score: 0.00000
Feature: wire8_y, Score: 0.00016
Feature: wire9_y, Score: 0.00014
```

In [98]:
```
f_train_s.columns
```

Out[98]:
```
Index(['bulk', 'bulk10_y', 'bulk11_y', 'bulk12_y', 'bulk13_y', 'bulk14_y',
       'bulk15_y', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y', 'bulk5_y',
       'bulk6_y', 'bulk7_y', 'bulk8_y', 'bulk9_y', 'gas', 'power',
       'process_time', 'start_t', 'wire', 'wire1_y', 'wire2_y', 'wire3_y',
       'wire4_y', 'wire5_y', 'wire6_y', 'wire7_y', 'wire8_y', 'wire9_y'],
      dtype='object')
```

RandomForestRegressor - лучшая модель

In [99]:
```
features=['bulk', 'bulk12_y', 'bulk14_y', 'bulk15_y', 'bulk3_y', 'bulk4_y',
       'bulk6_y', 'gas', 'power',
       'process_time', 'start_t', 'wire', 'wire1_y', 'wire2_y']
RF_model=RandomForestRegressor(random_state=r,criterion= 'mse', max_depth= 1
f_train_s_scale=std_scale.fit_transform(f_train_s[features])
RF_model.fit(f_train_s_scale,t_train_s)
```

Out[99]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=16,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=137,
                      n_jobs=None, oob_score=False, random_state=17, verbose
=0,
                      warm_start=False)
```

In [100…
```
Models_train_score.loc['mae',['RF_small']]=mean_absolute_error(RF_model.pred
Models_train_score.loc['MAPE',['RF_small']]=sMAPE(RF_model.predict(f_train_s
Models_train_score.loc['mse',['RF_small']]= mean_squared_error(RF_model.pred
```

XGBRegressor

In [101…
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

XGBRegressor - подбор гиперпараметров, оценка признаков

In [102…
```
r=17
pipe_r = Pipeline(steps=[('scale', scaler1),  # Трансформеры
                         ('XGBR', XGBRegressor(random_state=r))])

pipe_params = {'XGBR__objective': ['reg:squarederror'],
               'XGBR__reg_alpha': [1, 1.5, 2],
               'XGBR__n_estimators': range(150,250,5)}
prs = RandomizedSearchCV(pipe_r, pipe_params, cv=5, n_iter=10, scoring='neg_
prs.fit(f_train_s,t_train_s)


print(abs(prs.best_score_))
prs.best_params_
```

```
61.11995308144395
```

Out[102]:
```
{'XGBR__reg_alpha': 1.5,
 'XGBR__objective': 'reg:squarederror',
 'XGBR__n_estimators': 245}
```

In [103…
```
Models_train_score.loc['cv_best_score',['XGB_small']]=abs(prs.best_score_)
```

In [104…
```
XGBR_model=XGBRegressor(n_estimators=245, reg_alpha=1.5,objective='reg:squar

f_train_s_scale=std_scale.fit_transform(f_train_s)
XGBR_model.fit(f_train_s_scale,t_train_s)
```

Out[104]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=24
5,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=17, reg_alpha=1.5, reg_lambda=1, scale_pos_weight
=1,
             seed=None, silent=None, subsample=1, verbosity=1)
```

In [105…
```
importance = XGBR_model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %s, Score: %.5f' % (f_train_s.columns[i],v))
```

```
Feature: bulk, Score: 0.03025
Feature: bulk10_y, Score: 0.02690
Feature: bulk11_y, Score: 0.04885
Feature: bulk12_y, Score: 0.04854
Feature: bulk13_y, Score: 0.00892
Feature: bulk14_y, Score: 0.03771
Feature: bulk15_y, Score: 0.04294
Feature: bulk1_y, Score: 0.04282
Feature: bulk2_y, Score: 0.01537
Feature: bulk3_y, Score: 0.02403
Feature: bulk4_y, Score: 0.07307
Feature: bulk5_y, Score: 0.04071
Feature: bulk6_y, Score: 0.05874
Feature: bulk7_y, Score: 0.00000
Feature: bulk8_y, Score: 0.00000
Feature: bulk9_y, Score: 0.00000
Feature: gas, Score: 0.02422
Feature: power, Score: 0.06299
Feature: process_time, Score: 0.05321
Feature: start_t, Score: 0.08414
Feature: wire, Score: 0.09810
Feature: wire1_y, Score: 0.08123
Feature: wire2_y, Score: 0.07805
Feature: wire3_y, Score: 0.00741
Feature: wire4_y, Score: 0.00000
Feature: wire5_y, Score: 0.00000
Feature: wire6_y, Score: 0.01178
Feature: wire7_y, Score: 0.00000
Feature: wire8_y, Score: 0.00000
Feature: wire9_y, Score: 0.00000
```

XGBRegressor - лучшая модель

In [106…

```
features=['bulk', 'bulk10_y', 'bulk11_y', 'bulk12_y',  'bulk14_y',
       'bulk15_y', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y','bulk6_y', 'ga
       'process_time', 'start_t', 'wire', 'wire1_y', 'wire2_y', 'wire6_y', '

XGBR_model=XGBRegressor(n_estimators=245, reg_alpha=1.5,objective='reg:squar

f_train_s_scale=std_scale.fit_transform(f_train_s[features])
XGBR_model.fit(f_train_s_scale,t_train_s)
```

Out[106]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=24
5,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=17, reg_alpha=1.5, reg_lambda=1, scale_pos_weight
=1,
             seed=None, silent=None, subsample=1, verbosity=1)
```

In [107…

```
Models_train_score.loc['mae',['XGB_small']]=mean_absolute_error(XGBR_model.p
Models_train_score.loc['MAPE',['XGB_small']]=sMAPE(XGBR_model.predict(f_trai
Models_train_score.loc['mse',['XGB_small']]= mean_squared_error(XGBR_model.p
```

Dummy-model

In [108…

```
Dummy_small=Dummy_model_mean()
Dummy_small.fit(f_train_s,t_train_s)
print(mean_absolute_error(Dummy_small.predict(f_test_s),t_test_s))
```

```
8.459072201059591
```

In [109…
```
Models_train_score.loc['mae',['Dummy_small']]=mean_absolute_error(Dummy_smal
Models_train_score.loc['MAPE',['Dummy_small']]=sMAPE(Dummy_small.predict(f_t
Models_train_score.loc['mse',['Dummy_small']]= mean_squared_error(Dummy_smal
```

Таблица df_big

In [110…
```
df_big.dropna(inplace=True)
df_big=df_big.reset_index(drop=True)
df_big.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8427 entries, 0 to 8426
Data columns (total 35 columns):
key             8427 non-null int64
time            8427 non-null datetime64[ns]
temp            8427 non-null int64
wire1_y         8427 non-null float64
wire2_y         8427 non-null float64
wire3_y         8427 non-null float64
wire4_y         8427 non-null float64
wire5_y         8427 non-null float64
wire6_y         8427 non-null float64
wire7_y         8427 non-null float64
wire8_y         8427 non-null float64
wire9_y         8427 non-null float64
wire            8427 non-null float64
bulk1_y         8427 non-null float64
bulk2_y         8427 non-null float64
bulk3_y         8427 non-null float64
bulk4_y         8427 non-null float64
bulk5_y         8427 non-null float64
bulk6_y         8427 non-null float64
bulk7_y         8427 non-null float64
bulk8_y         8427 non-null float64
bulk9_y         8427 non-null float64
bulk10_y        8427 non-null float64
bulk11_y        8427 non-null float64
bulk12_y        8427 non-null float64
bulk13_y        8427 non-null float64
bulk14_y        8427 non-null float64
bulk15_y        8427 non-null float64
bulk            8427 non-null float64
start_t         8427 non-null int64
start_time      8427 non-null datetime64[ns]
process_time    8427 non-null int64
gas             8427 non-null float64
power           8427 non-null float64
is_final        8427 non-null int64
dtypes: datetime64[ns](2), float64(28), int64(5)
memory usage: 2.3 MB
```

In [111…
```
df_big_f=df_big[df_big['is_final']==1]
df_big_m=df_big[df_big['is_final']==0]
```

In [112…
```
f_train_b, f_test_b, t_train_b, t_test_b =train_test_split(df_big_f, \
                                                           df_big_f['temp'],
f_train_b=f_train_b.append(df_big_m)
t_train_b=t_train_b.append(df_big_m['temp'])
```

In [113…
```
f_train_b=f_train_b.drop(['temp', 'key', 'time','start_time','is_final'],axi
f_test_b=f_test_b.drop(['temp', 'key', 'time','start_time','is_final'],axis=
```

In [114…
```
col2=f_train_b.columns
```

In [115…
```
# преобразования признаков StandardScaler
scaler2 = ColumnTransformer(transformers=[
    ('std', StandardScaler(),col2),
], remainder='passthrough')
```

DecisionTreeRegressor

In [116…
```
r=17
pipe_r = Pipeline(steps=[('scale', scaler2),  # Трансформеры
                    ('DT', DecisionTreeRegressor(random_state=r))])

pipe_params = {'DT__criterion': ['mse', 'mae'],
               'DT__max_depth': range(3, 20)}
prs = RandomizedSearchCV(pipe_r, pipe_params, cv=5, n_iter=10, scoring='neg_

prs.fit(f_train_b,t_train_b)

print(abs(prs.best_score_))
prs.best_params_
```

Out[116]:
```
107.04980203166465
 {'DT__max_depth': 8, 'DT__criterion': 'mse'}
```

In [117…
```
Models_train_score.loc['cv_best_score',['DT_big']]=abs(prs.best_score_)
Models_train_score.loc['mae',['DT_big']]=mean_absolute_error(prs.predict(f_t
Models_train_score.loc['MAPE',['DT_big']]=sMAPE(prs.predict(f_train_b),t_tra
Models_train_score.loc['mse',['DT_big']]= mean_squared_error(prs.predict(f_t
```

RandomForestRegressor

RandomForestRegressor - подбор гиперпараметров, оценка признаков

In [118…
```
r=17
pipe_r = Pipeline(steps=[('scale', scaler2),  # Трансформеры
                    ('RF', RandomForestRegressor(random_state=r))])

pipe_params = {'RF__criterion':  ['mse'],
               'RF__max_depth': range(10,20),
               'RF__n_estimators': range(100,150)}
prs = RandomizedSearchCV(pipe_r, pipe_params, cv=3, n_iter=10, scoring='neg_
prs.fit(f_train_b,t_train_b)

print(abs(prs.best_score_))
prs.best_params_
```

Out[118]:
```
77.02378299868607
 {'RF__n_estimators': 107, 'RF__max_depth': 17, 'RF__criterion': 'mse'}
```

In [119…
```
Models_train_score.loc['cv_best_score',['RF_big']]=abs(prs.best_score_)
```

In [120…
```
std_scale=StandardScaler()
```

In [121…
```
RF_model=RandomForestRegressor(random_state=r,criterion= 'mse', max_depth= 1
f_train_b_scale=std_scale.fit_transform(f_train_b)
RF_model.fit(f_train_b_scale,t_train_b)
```

Out[121]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=17,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=107,
                      n_jobs=None, oob_score=False, random_state=17, verbos
e=0,
                      warm_start=False)
```

In [122…
```
importance = RF_model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %s, Score: %.5f' % (f_train_b.columns[i],v))
```

```
Feature: wire1_y, Score: 0.03261
Feature: wire2_y, Score: 0.01484
Feature: wire3_y, Score: 0.00006
Feature: wire4_y, Score: 0.00004
Feature: wire5_y, Score: 0.00000
Feature: wire6_y, Score: 0.00046
Feature: wire7_y, Score: 0.00000
Feature: wire8_y, Score: 0.00005
Feature: wire9_y, Score: 0.00017
Feature: wire, Score: 0.08261
Feature: bulk1_y, Score: 0.00407
Feature: bulk2_y, Score: 0.00005
Feature: bulk3_y, Score: 0.00892
Feature: bulk4_y, Score: 0.00897
Feature: bulk5_y, Score: 0.00049
Feature: bulk6_y, Score: 0.00895
Feature: bulk7_y, Score: 0.00003
Feature: bulk8_y, Score: 0.00000
Feature: bulk9_y, Score: 0.00002
Feature: bulk10_y, Score: 0.00332
Feature: bulk11_y, Score: 0.00153
Feature: bulk12_y, Score: 0.01967
Feature: bulk13_y, Score: 0.00044
Feature: bulk14_y, Score: 0.02949
Feature: bulk15_y, Score: 0.01618
Feature: bulk, Score: 0.03351
Feature: start_t, Score: 0.36512
Feature: process_time, Score: 0.11300
Feature: gas, Score: 0.04944
Feature: power, Score: 0.20597
```

RandomForestRegressor - лучшая модель

In [123…
```
features=['wire1_y', 'wire2_y','wire', 'bulk12_y',  'bulk14_y',
       'bulk15_y', 'bulk', 'start_t', 'process_time', 'gas', 'power']
RF_model=RandomForestRegressor(random_state=r,criterion= 'mse', max_depth= 1
f_train_b_scale=std_scale.fit_transform(f_train_b[features])
RF_model.fit(f_train_b_scale,t_train_b)
```

Out[123]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=17,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=107,
                      n_jobs=None, oob_score=False, random_state=17, verbos
e=0,
                      warm_start=False)
```

In [124…
```
Models_train_score.loc['mae',['RF_big']]=mean_absolute_error(RF_model.predic
Models_train_score.loc['MAPE',['RF_big']]=sMAPE(RF_model.predict(f_train_b_s
```

```
Models_train_score.loc['mse',['RF_big']]= mean_squared_error(RF_model.predic
```

XGBRegressor

XGBRegressor - подбор гиперпараметров, оценка признаков

In [125…
```
r=17
pipe_r = Pipeline(steps=[('scale', scaler2),  # Трансформеры
                         ('XGBR', XGBRegressor(random_state=r))])

pipe_params = {'XGBR__objective': ['reg:squarederror'],
               'XGBR__reg_alpha': [1, 1.5, 2],
               'XGBR__n_estimators': range(150,250)}
prs = RandomizedSearchCV(pipe_r, pipe_params, cv=5, n_iter=10, scoring='neg_
prs.fit(f_train_b,t_train_b)


print(abs(prs.best_score_))
prs.best_params_
```

```
74.49828694326514
```

Out[125]:
```
{'XGBR__reg_alpha': 2,
 'XGBR__objective': 'reg:squarederror',
 'XGBR__n_estimators': 246}
```

In [126…
```
Models_train_score.loc['cv_best_score',['XGB_big']]=abs(prs.best_score_)
```

In [127…
```
XGBR_model=XGBRegressor(n_estimators=246, reg_alpha=2,objective='reg:squared

f_train_b_scale=std_scale.fit_transform(f_train_b)
XGBR_model.fit(f_train_b_scale,t_train_b)
```

Out[127]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=24
6,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=17, reg_alpha=2, reg_lambda=1, scale_pos_weight=
1,
             seed=None, silent=None, subsample=1, verbosity=1)
```

In [128…
```
importance = XGBR_model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %s, Score: %.5f' % (f_train_b.columns[i],v))
```

```
Feature: wire1_y, Score: 0.03550
Feature: wire2_y, Score: 0.07009
Feature: wire3_y, Score: 0.00725
Feature: wire4_y, Score: 0.00260
Feature: wire5_y, Score: 0.00000
Feature: wire6_y, Score: 0.00994
Feature: wire7_y, Score: 0.00000
Feature: wire8_y, Score: 0.00000
Feature: wire9_y, Score: 0.00723
Feature: wire, Score: 0.11175
Feature: bulk1_y, Score: 0.04285
Feature: bulk2_y, Score: 0.01162
Feature: bulk3_y, Score: 0.01690
Feature: bulk4_y, Score: 0.07837
Feature: bulk5_y, Score: 0.01091
Feature: bulk6_y, Score: 0.07378
Feature: bulk7_y, Score: 0.00000
Feature: bulk8_y, Score: 0.00000
Feature: bulk9_y, Score: 0.00600
Feature: bulk10_y, Score: 0.02118
Feature: bulk11_y, Score: 0.01118
Feature: bulk12_y, Score: 0.03119
Feature: bulk13_y, Score: 0.03171
Feature: bulk14_y, Score: 0.03197
Feature: bulk15_y, Score: 0.04154
Feature: bulk, Score: 0.04360
Feature: start_t, Score: 0.13532
Feature: process_time, Score: 0.05896
Feature: gas, Score: 0.02080
Feature: power, Score: 0.08777
```

XGBRegressor - лучшая модель

In [129…

```
features=['wire1_y', 'wire2_y',  'wire', 'bulk1_y', 'bulk2_y',
       'bulk3_y', 'bulk4_y', 'bulk5_y', 'bulk6_y','bulk10_y', 'bulk11_y', 'b
       'bulk15_y', 'bulk', 'start_t', 'process_time', 'gas', 'power']

XGBR_model=XGBRegressor(n_estimators=246, reg_alpha=2,objective='reg:squared

f_train_b_scale=std_scale.fit_transform(f_train_b[features])
XGBR_model.fit(f_train_b_scale,t_train_b)
```

Out[129]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=24
6,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=17, reg_alpha=2, reg_lambda=1, scale_pos_weight=
1,
             seed=None, silent=None, subsample=1, verbosity=1)
```

In [130…

```
Models_train_score.loc['mae',['XGB_big']]=mean_absolute_error(XGBR_model.pre
Models_train_score.loc['MAPE',['XGB_big']]=sMAPE(XGBR_model.predict(f_train_
Models_train_score.loc['mse',['XGB_big']]= mean_squared_error(XGBR_model.pre
```

Dummy-model

In [131…

```
Dummy_big=Dummy_model_mean()
Dummy_big.fit(f_train_b,t_train_b)
print(mean_absolute_error(Dummy_big.predict(f_test_b),t_test_b))
```

```
8.436383259191999
```

In [132…
```
Models_train_score.loc['mae',['Dummy_big']]=mean_absolute_error(Dummy_big.pr
Models_train_score.loc['MAPE',['Dummy_big']]=sMAPE(Dummy_big.predict(f_train
Models_train_score.loc['mse',['Dummy_big']]= mean_squared_error(Dummy_big.pr
```

Вывод

Анализ факторов

По результатам анализ а факторов на модели типа RF, обученной на таблице df_small, значимые факторы (коэффициент >= 0,01):
'bulk', 'bulk12_y', 'bulk14_y', 'bulk15_y', 'bulk3_y', 'bulk4_y', 'bulk6_y', 'gas', 'power','process_time', 'start_t', 'wire', 'wire1_y', 'wire2_y'

По результатам анализ а факторов на модели типа XGB, обученной на таблице df_small, значимые факторы (коэффициент >= 0,01):
'bulk', 'bulk10_y', 'bulk11_y', 'bulk12_y', 'bulk14_y','bulk15_y', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y','bulk6_y', 'gas', 'power', 'process_time', 'start_t', 'wire', 'wire1_y', 'wire2_y', 'wire6_y', 'wire3_y'

По результатам анализ а факторов на модели типа RF, обученной на таблице df_big, значимые факторы (коэффициент >= 0,01):
'wire1_y', 'wire2_y','wire', 'bulk12_y', 'bulk14_y','bulk15_y', 'bulk', 'start_t', 'process_time', 'gas', 'power'

По результатам анализ а факторов на модели типа XGB, обученной на таблице df_big, значимые факторы (коэффициент >= 0,01):
'wire1_y', 'wire2_y', 'wire', 'bulk1_y', 'bulk2_y','bulk3_y', 'bulk4_y', 'bulk5_y', 'bulk6_y','bulk10_y', 'bulk11_y', 'bulk12_y', 'bulk13_y', 'bulk14_y','bulk15_y', 'bulk', 'start_t', 'process_time', 'gas', 'power'

вывод: Важные факторы: 'wire', 'bulk', 'start_t', 'process_time', 'gas', 'power', 'bulk12_y', 'bulk14_y', 'bulk15_y', 'wire1_y', 'wire2_y'

Анализ моделей

In [133…
```
Models_train_score
```

Out[133]:

	Dummy_small	DT_small	RF_small	XGB_small	Dummy_big	DT_big	RF_bi
mae	8.29664	6.05982	2.4682	3.86525	11.0173	6.09539	2.2317
MAPE	0.519639	0.379755	0.154616	0.242232	0.691403	0.382756	0.14009
mse	143.261	74.9553	10.3835	25.6317	215.211	62.9997	8.9974
cv_best_score	NaN	103.896	68.9691	61.12	NaN	107.05	77.023

Лучшее значение метрик у модели типа XGBRegressor, обученной на датасете df_small:
mse = 61.12 по результатам кросс-валидации (на обучающей выборке - 25.63)
также относительно хорошее значение метрик у модели типа XGBRegressor, обученной на датасете df_big:
mse = 74.49 по результатам кросс-валидации (на обучающей выборке - 51.25)

У модели XGB_big разница метрик между обучающей и тестовой выборкой меньше, вероятно, она меньше переобучена.

Тестирование лучших моделей

In [134…
```
features=['bulk', 'bulk10_y', 'bulk11_y', 'bulk12_y',  'bulk14_y',
       'bulk15_y', 'bulk1_y', 'bulk2_y', 'bulk3_y', 'bulk4_y','bulk6_y', 'ga
       'process_time', 'start_t', 'wire', 'wire1_y', 'wire2_y', 'wire6_y', '

XGBR_model=XGBRegressor(n_estimators=245, reg_alpha=1.5,objective='reg:squar

f_train_s_scale=std_scale.fit_transform(f_train_s[features])
XGBR_model.fit(f_train_s_scale,t_train_s)
f_test_s_scale=std_scale.transform(f_test_s[features])
```

In [135…
```
Models_test_score.loc['mae',['XGB_small']]=mean_absolute_error(XGBR_model.pr
Models_test_score.loc['MAPE',['XGB_small']]=sMAPE(XGBR_model.predict(f_test_
Models_test_score.loc['mse',['XGB_small']]= mean_squared_error(XGBR_model.pr
```

In [136…
```
Models_test_score.loc['mae',['Dummy_small']]=mean_absolute_error(Dummy_small
Models_test_score.loc['MAPE',['Dummy_small']]=sMAPE(Dummy_small.predict(f_te
Models_test_score.loc['mse',['Dummy_small']]= mean_squared_error(Dummy_small
```

In [137…
```
features=['wire1_y', 'wire2_y',  'wire', 'bulk1_y', 'bulk2_y',
       'bulk3_y', 'bulk4_y', 'bulk5_y', 'bulk6_y','bulk10_y', 'bulk11_y', 'b
       'bulk15_y', 'bulk', 'start_t', 'process_time', 'gas', 'power']

XGBR_model=XGBRegressor(n_estimators=246, reg_alpha=2,objective='reg:squared

f_train_b_scale=std_scale.fit_transform(f_train_b[features])
XGBR_model.fit(f_train_b_scale,t_train_b)
f_test_b_scale=std_scale.transform(f_test_b[features])
```

In [138…
```
Models_test_score.loc['mae',['XGB_big']]=mean_absolute_error(XGBR_model.pred
Models_test_score.loc['MAPE',['XGB_big']]=sMAPE(XGBR_model.predict(f_test_b_
Models_test_score.loc['mse',['XGB_big']]= mean_squared_error(XGBR_model.pred
```

In [139…
```
Models_test_score.loc['mae',['Dummy_big']]=mean_absolute_error(Dummy_big.pre
Models_test_score.loc['MAPE',['Dummy_big']]=sMAPE(Dummy_big.predict(f_test_b
Models_test_score.loc['mse',['Dummy_big']]= mean_squared_error(Dummy_big.pre
```

In [140…
```
Models_test_score.dropna(axis=1)
```

Out[140]:

	Dummy_small	XGB_small	Dummy_big	XGB_big
mae	8.45907	5.52908	8.43638	5.00937
MAPE	0.529088	0.346	0.528254	0.313824
mse	167.045	54.9916	160.973	43.0366

Вывод

Лучшее значение метрики (mae) на тестовой выборке показала модель XGB_big - 5, что существенно лучше (в 1.5 раза) dummy-model XGB_big - модель типа XGBRegressor, обученная на датасете df_big

Заключение:

Проект "Промышленность"
Цель: Уменьшить энергопотребление
Задача: Построить модель, предсказывающую температуру стали

Описание этапа обработки

Сталь обрабатывают в металлическом ковше вместимостью около 100 тонн. Чтобы ковш выдерживал высокие температуры, изнутри его облицовывают огнеупорным кирпичом. Расплавленную сталь заливают в ковш и подогревают до нужной температуры графитовыми электродами. Они установлены в крышке ковша.

Из сплава выводится сера (десульфурация), добавлением примесей корректируется химический состав и отбираются пробы. Сталь легируют — изменяют её состав — подавая куски сплава из бункера для сыпучих материалов или проволоку через специальный трайб-аппарат (англ. tribe, «масса»).

Перед тем как первый раз ввести легирующие добавки, измеряют температуру стали и производят её химический анализ. Потом температуру на несколько минут повышают, добавляют легирующие материалы и продувают сплав инертным газом. Затем его перемешивают и снова проводят измерения. Такой цикл повторяется до достижения целевого химического состава и оптимальной температуры плавки.

Тогда расплавленная сталь отправляется на доводку металла или поступает в машину непрерывной разливки. Оттуда готовый продукт выходит в виде заготовок-слябов (англ. *slab*, «плита»).

Описание данных

Данные состоят из файлов, полученных из разных источников:

- `data_arc.csv` — данные об электродах;
- `data_bulk.csv` — данные о подаче сыпучих материалов (объём);
- `data_bulk_time.csv` — данные о подаче сыпучих материалов (время);
- `data_gas.csv` — данные о продувке сплава газом;
- `data_temp.csv` — результаты измерения температуры;
- `data_wire.csv` — данные о проволочных материалах (объём);
- `data_wire_time.csv` — данные о проволочных материалах (время).

Предварительная гипотеза:
Предполагается что температура сплава является показателем качества сплава стали либо используется как параметр при его рассчете. Затраты электроенергии связаны с нагревом сплава. Т.к. для введения добавок сплав необходимо нагревать, количество итераций (процедур введения примесей и последующее измерение качества) непосредственно влияет на затраты электроенергии. Модель, предсказывающая температуру сплава позволит заранее подобрать необходимое количество примесей и минимизировать количество итераций.

Загрузка и просмотр данных
В файле с данными об электродах содержится информация о мощности и времени работы электродов для 3214 партий. Всего 14876 строк. Пропусков нет. По данным можно оценить затраты электроенергии на партии.

В файлах с данными о подаче сыпучих материалов содержится информация об объёме и времени различных сыпучих материалов для 3129 партий. Всего 3129 строк - т.е. одна строка соответствует одной партии.
По данным таблицам можно оценить общий объем сыпучих материалов, содержащихся в париях в анализируемый момент времени.

В файлах с данными о продувке сплава инертным газом содержится некоторая информация о газе для 3239 партий. Всего 3239 строк - т.е. одна строка соответствует одной партии.
Вероятно значения содержат некоторый количественный показатель. Т.к. "... основной целью продувки металла инертными газами является его дегазация, удаление неметаллических включений, выравнивание химического состава и температуры по всему объему металла..." эти данные вероятно не влияют на химический состав, однако могут влиять на затраты электроенергии.

В файле с данными о результатах измерения температуры содержится информация о температуре сплава и времени работы измерения для 3216 партий. Всего 15907 строк. Есть пропуски. По данным можно оценить температуру сплава партии на различных стадиях, финальну (предположительно целевую) температуру сплава партий.

В файлах с данными о проволочных материалах содержится время и объем различных проволочных материалов для 3081 партий. Всего 3081 строк - т.е. одна строка соответствует одной партии.
По данным таблицам можно оценить общий объем проволочных материалов, содержащихся в париях в анализируемый момент времени.

Анализ данных об измерении температуры:
В данных об измерениях температуры сплава:

- 2901 пропуск
- содержится информация о 3216 партиях
- время всех измерений уникально
  Анализ пропусков показал, что для каждой партии в таблице есть хотя бы одно значение температуры
  Получены таблицы с начальными и финальными температурами сплава для каждой партии
  Распределения финальных температур имеет меньше дисперсию чем распределение исходных температур. Медианные значения близки.

Генерация признаков:
Созданы 2 таблицы:

- Таблица df_small содержит финальные температуры для каждой партии и соответствующие им значения стартовой температуры и количества примесей, а также газ, используемый для данной партии
- Таблица df_big содержит все значения температуры (кроме стартовых) и соответствующие им значения стартовой температуры и количества примесей к текущему моменту времени добавленных в состав*.

Таблица df_small содержит 1974 сэмпла, в т.ч. данные по газу. Таблица df_big содержит 8746 сэмпла.

*Идея создания таблицы df_big заключается в существенном увеличении количества сэмплов без дублирования за счет использования промежуточных измерений. Промежуточные измерения принимаются как финальные и используются для установления влияния различных факторов на температуру сплава. При этом утечки информации не происходит, т.к. сами значения температуры в промежуточных стадиях не используются как признаки. Предполагается, что такой подход позволит расширить диапазон значений целевогои признакового пространства и тем самым уменьшить эффект переобучения.

Анализ факторов
По результатам анализа факторов на различных данных и моделях важными факторами являются: суммарное количество проволочных компонентов, суммарное количество сыпучих компонентов, исходная температура, время технологического процесса, количественный показатель газа, активная мощность, сыпучие компоненты - 12, 14, 15, проволочные компоненты 1 и 2.

Анализ моделей
Лучшее значение метрик у модели типа XGBRegressor, обученной на датасете df_small:
mse = 61.12 по результатам кросс-валидации (на обучающей выборке - 25.63)
также относительно хорошее значение метрик у модели типа XGBRegressor, обученной на датасете df_big:
mse = 74.49 по результатам кросс-валидации (на обучающей выборке - 51.25)
У модели XGB_big разница метрик между обучающей и тестовой выборкой меньше, вероятно, она меньше переобучена.

Тестирование моделей

In [141…
```
Models_test_score.dropna(axis=1)
```

Out[141]:

	Dummy_small	XGB_small	Dummy_big	XGB_big
mae	8.45907	5.52908	8.43638	5.00937
MAPE	0.529088	0.346	0.528254	0.313824
mse	167.045	54.9916	160.973	43.0366

Лучшее значение метрики (mae) на тестовой выборке показала модель XGB_big - 5, что существенно лучше (в 1.5 раза) dummy-model XGB_big - модель типа

XGBRegressor, обученная на датасете df_big

Модель может быть использована для предсказания температуры сплава по известным параметрам технологического процесса, а также для подбора по модели значений параметров технологического процесса для обеспечения необходимой температуры сплава при минимизации мощности.

Отчет по проекту:

Согласованный план действий:

1) Выделить 20% данных для тестовой выборки. Обучение и выбор модели и подбор гиперпараметров проводить на оставшихся данных.
2) Построить несколько моделей для задачи регрессии на данных таблицы df_small (целевой признак - финальная температура, признаки - количество сыпучих примесей, количество проволочных примесей, количество газа, стартовая температура). Исследовать модели типа **Линейная регрессия**, **Случайный лес**. В качестве метрики использовать MSE.
3) Построить несколько моделей для задачи регрессии на данных таблицы df_big (целевой признак - финальная температура, признаки - количество сыпучих примесей, количество проволочных примесей, стартовая температура). Исследовать модели типа **Линейная регрессия**, **Случайный лес**. В качестве метрики использовать MSE.
4) Построить дамми-модель предсказывающую всегда среднее значение финальной температуры обучающей выборки.
5) Выбрать лучшую модель оценить ее качество сравнив с дамми-моделью и по результатам тестирования на тестовой выборке.

Отчет:

- Все пункты плана выполнены
- На этапе **Обработка данных. Создание обучающей / тестовой выборок:** принято решение содать 2 таблицы:
  - Таблица df_small содержит финальные температуры для каждой партии и соответствующие им значения стартовой температуры и количества примесей, а также газ, используемый для данной партии (1974 сэмпла)
  - Таблица df_big содержит все значения температуры (кроме стартовых) и соответствующие им значения стартовой температуры и количества примесей к текущему моменту времени добавленных в состав (8746 сэмплов)*.

*Идея создания таблицы df_big заключается в существенном увеличении количества сэмплов без дублирования за счет использования промежуточных измерений. Промежуточные измерения принимаются как финальные и используются для установления влияния различных факторов на температуру сплава. При этом утечки информации не происходит, т.к. сами значения температуры в промежуточных стадиях не используются как признаки.

Предполагается, что такой подход позволит расширить диапазон значений целевогои признакового пространства и тем самым уменьшить эффект переобучения.

- На этапе очистки данных - оставлены данные от 1 до 99 квантиля для признаков - суммарное количество проволочных компонентов, суммарное количество сыпучих компонентов, исходная температура, время технологического процесса, количественный показатель газа, активная мощность. Принято решение по отдельным добавкам не удалялять выбросы т.к. данных по отдельным добавкам мало.

- По результатам анализа факторов на различных данных и моделях важными факторами являются: суммарное количество проволочных компонентов, суммарное количество сыпучих компонентов, исходная температура, время технологического процесса, количественный показатель газа, активная мощность, сыпучие компоненты - 12, 14, 15, проволочные компоненты 1 и 2.

- Выделено 25 % данных таблицы df_small для теста (493 сэмпла) и 50 % данных среди финальных стадий для таблицы df_big, что соответствует 11% данных (987 сэмплов). Для таблицы df_small на тест выделено 25% данных и 50 % данных среди финальных стадий для таблицы df_big, а не 20% планируемых, с тем что бы повысить надежность результатов тестирования. Для таблицы df_big на тест выбраны только финальные стадии, т.к. в будущем модель будет использована для прогноза финальной температуры. При этом, в тест ушло лишь 11% всех данных.

- На этапе **Анализ моделей** рассмотрены модели вида **DecisionTreeRegressor**, **RandomForestRegressor**, **XGBRegressor** Модель вида Линейная регрессия показала изначально плохое значение метрики и в дальнейшем не рассматривалась.

- Гиперпараметры моделей подбирались методом крос-валидации с использованием метрики MSE (т.к. она более обьективна чем целевая метрика MAE)
- По результатам кросс-валидации лучшее значение метрик у модели типа XGBRegressor, обученной на датасете df_small(mse = 61.12 по результатам кросс-валидации (на обучающей выборке - 25.63)). также относительно хорошее значение метрик у модели типа XGBRegressor, обученной на датасете df_big (mse = 74.49 по результатам кросс-валидации (на обучающей выборке - 51.25)). Т.к. у модели XGB_big разница метрик между обучающей и тестовой выборкой меньше, вероятно, она меньше переобучена. Обе модели отпралены на этап тестирования.
- Лучшее значение метрики (mae) на тестовой выборке показала модель **XGB_big - 5**, что существенно лучше (в 1.5 раза) dummy-model. **XGB_big - модель типа XGBRegressor, обученная на датасете df_big**

In [ ]: