Integrated Project 2

Recovered gold share prediction model

Table of Contents

Goal

Develop a machine learning model for Zyfra, efficiency solutions for heavy industry company, that would analyze data on extraction and purification of gold from an ore and predict the amount of recovered gold. The model will help to optimize the production and eliminate unprofitable parameters.

Data description

Technological process

- *Rougher feed* — raw material
- *Rougher additions* (or *reagent additions*) — flotation reagents: *Xanthate, Sulphate, Depressant*
  - *Xanthate* — promoter or flotation activator;
  - *Sulphate* — sodium sulphide for this particular process;
  - *Depressant* — sodium silicate.
- *Rougher process* — flotation
- *Rougher tails* — product residues
- *Float banks* — flotation unit
- *Cleaner process* — purification
- *Rougher Au* — rougher gold concentrate
- *Final Au* — final gold concentrate

Parameters of stages

- *air amount — volume of air*
- *fluid levels*
- *feed size* — feed particle size
- *feed rate*

Feature naming

Here's how we name the features:

`[stage].[parameter_type].[parameter_name]`

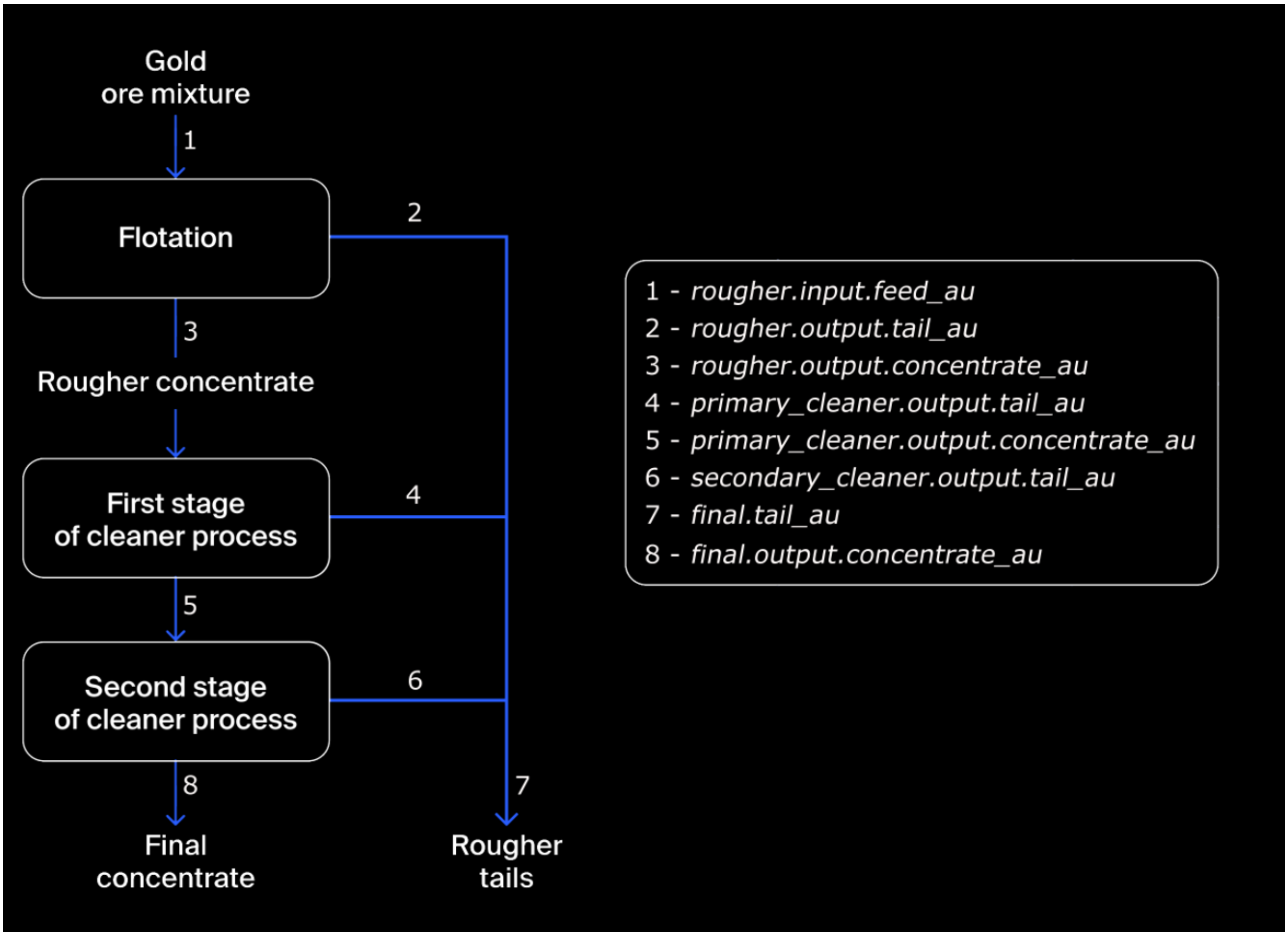
Example: `rougher.input.feed_ag`

Possible values for `[stage]` :

- *rougher* — flotation
- *primary_cleaner* — primary purification
- *secondary_cleaner* — secondary purification
- *final* — final characteristics

Possible values for `[parameter_type]` :

- *input* — raw material parameters
- *output* — product parameters
- *state* — parameters characterizing the current state of the stage
- *calculation* — calculation characteristics


Gold
ore mixture
1
Flotation
2
3
Rougher concentrate
First stage
of cleaner process
4
5
Second stage
of cleaner process
6
8
7
Final
concentrate
Rougher
tails
1 - rougher.input.feed_au
2 - rougher.output.tail_au
3 - rougher.output.concentrate_au
4 - primary_cleaner.output.tail_au
5 - primary_cleaner.output.concentrate_au
6 - secondary_cleaner.output.tail_au
7 - final.tail_au
8 - final.output.concentrate_au

Imports

In [2]:

```
import pandas as pd
import matplotlib
import numpy as np
import seaborn as sns

from sklearn.preprocessing import StandardScaler as ss

from sklearn.dummy import DummyRegressor
from sklearn import linear_model
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import svm

from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer

import matplotlib.pyplot as plt
%matplotlib inline

import sys
import warnings
if not sys.warnoptions:
       warnings.simplefilter("ignore")

pd.set_option('display.max_rows', None, 'display.max_columns', None)

print("Setup Complete")
```

```
Setup Complete
```

Input data

In [3]:

```
try:
    df_train = pd.read_csv('gold_recovery_train.csv')
    df_test = pd.read_csv('gold_recovery_test.csv')
    df_full = pd.read_csv('gold_recovery_full.csv')

except:
    df_train = pd.read_csv('/datasets/gold_recovery_train.csv')
    df_test = pd.read_csv('/datasets/gold_recovery_test.csv')
    df_full = pd.read_csv('/datasets/gold_recovery_full.csv')
```

Descriptive statistics

In [4]:

```
df_train.head()
```

Out[4]:

	date	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concentrate_sol
0	2016-01-15 00:00:00	6.055403	9.889648	5.507324
1	2016-01-15 01:00:00	6.029369	9.968944	5.257781
2	2016-01-15 02:00:00	6.055926	10.213995	5.383759
3	2016-01-15 03:00:00	6.047977	9.977019	4.858634
4	2016-01-15 04:00:00	6.148599	10.142511	4.939416

In [5]:

```
df_test.head()
```

Out[5]:

	date	primary_cleaner.input.sulfate	primary_cleaner.input.depressant	primary_cleaner.inpu
0	2016-09-01 00:59:59	210.800909	14.993118	
1	2016-09-01 01:59:59	215.392455	14.987471	
2	2016-09-01 02:59:59	215.259946	12.884934	
3	2016-09-01 03:59:59	215.336236	12.006805	
4	2016-09-01 04:59:59	199.099327	10.682530	

Notes for preprocessing:

- All features are numerical, except for the `date` variable.

In [6]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16860 entries, 0 to 16859
Data columns (total 87 columns):
 #   Column                                              Non-Null Co
unt  Dtype
---  ------                                              -----------
---  -----
 0   date                                                16860 non-n
ull  object
 1   final.output.concentrate_ag                         16788 non-n
ull  float64
 2   final.output.concentrate_pb                         16788 non-n
ull  float64
 3   final.output.concentrate_sol                        16490 non-n
ull  float64
 4   final.output.concentrate_au                         16789 non-n
ull  float64
 5   final.output.recovery                               15339 non-n
ull  float64
 6   final.output.tail_ag                                16794 non-n
ull  float64
 7   final.output.tail_pb                                16677 non-n
ull  float64
 8   final.output.tail_sol                               16715 non-n
ull  float64
 9   final.output.tail_au                                16794 non-n
ull  float64
 10  primary_cleaner.input.sulfate                       15553 non-n
ull  float64
 11  primary_cleaner.input.depressant                    15598 non-n
ull  float64
 12  primary_cleaner.input.feed_size                     16860 non-n
ull  float64
 13  primary_cleaner.input.xanthate                      15875 non-n
ull  float64
 14  primary_cleaner.output.concentrate_ag               16778 non-n
ull  float64
 15  primary_cleaner.output.concentrate_pb               16502 non-n
ull  float64
 16  primary_cleaner.output.concentrate_sol              16224 non-n
ull  float64
 17  primary_cleaner.output.concentrate_au               16778 non-n
ull  float64
 18  primary_cleaner.output.tail_ag                      16777 non-n
ull  float64
 19  primary_cleaner.output.tail_pb                      16761 non-n
ull  float64
 20  primary_cleaner.output.tail_sol                     16579 non-n
ull  float64
 21  primary_cleaner.output.tail_au                      16777 non-n
ull  float64
 22  primary_cleaner.state.floatbank8_a_air              16820 non-n
ull  float64
 23  primary_cleaner.state.floatbank8_a_level            16827 non-n
ull  float64
 24  primary_cleaner.state.floatbank8_b_air              16820 non-n
ull  float64
 25  primary_cleaner.state.floatbank8_b_level            16833 non-n
ull  float64
 26  primary_cleaner.state.floatbank8_c_air              16822 non-n
ull  float64
```

```
 27  primary_cleaner.state.floatbank8_c_level             16833 non-n
ull  float64
 28  primary_cleaner.state.floatbank8_d_air               16821 non-n
ull  float64
 29  primary_cleaner.state.floatbank8_d_level             16833 non-n
ull  float64
 30  rougher.calculation.sulfate_to_au_concentrate        16833 non-n
ull  float64
 31  rougher.calculation.floatbank10_sulfate_to_au_feed  16833 non-n
ull  float64
 32  rougher.calculation.floatbank11_sulfate_to_au_feed  16833 non-n
ull  float64
 33  rougher.calculation.au_pb_ratio                      15618 non-n
ull  float64
 34  rougher.input.feed_ag                                16778 non-n
ull  float64
 35  rougher.input.feed_pb                                16632 non-n
ull  float64
 36  rougher.input.feed_rate                              16347 non-n
ull  float64
 37  rougher.input.feed_size                              16443 non-n
ull  float64
 38  rougher.input.feed_sol                               16568 non-n
ull  float64
 39  rougher.input.feed_au                                16777 non-n
ull  float64
 40  rougher.input.floatbank10_sulfate                    15816 non-n
ull  float64
 41  rougher.input.floatbank10_xanthate                   16514 non-n
ull  float64
 42  rougher.input.floatbank11_sulfate                    16237 non-n
ull  float64
 43  rougher.input.floatbank11_xanthate                   14956 non-n
ull  float64
 44  rougher.output.concentrate_ag                        16778 non-n
ull  float64
 45  rougher.output.concentrate_pb                        16778 non-n
ull  float64
 46  rougher.output.concentrate_sol                       16698 non-n
ull  float64
 47  rougher.output.concentrate_au                        16778 non-n
ull  float64
 48  rougher.output.recovery                              14287 non-n
ull  float64
 49  rougher.output.tail_ag                               14610 non-n
ull  float64
 50  rougher.output.tail_pb                               16778 non-n
ull  float64
 51  rougher.output.tail_sol                              14611 non-n
ull  float64
 52  rougher.output.tail_au                               14611 non-n
ull  float64
 53  rougher.state.floatbank10_a_air                      16807 non-n
ull  float64
 54  rougher.state.floatbank10_a_level                    16807 non-n
ull  float64
 55  rougher.state.floatbank10_b_air                      16807 non-n
ull  float64
 56  rougher.state.floatbank10_b_level                    16807 non-n
ull  float64
 57  rougher.state.floatbank10_c_air                      16807 non-n
```

```
ull  float64
 58  rougher.state.floatbank10_c_level                    16814 non-n
ull  float64
 59  rougher.state.floatbank10_d_air                      16802 non-n
ull  float64
 60  rougher.state.floatbank10_d_level                    16809 non-n
ull  float64
 61  rougher.state.floatbank10_e_air                      16257 non-n
ull  float64
 62  rougher.state.floatbank10_e_level                    16809 non-n
ull  float64
 63  rougher.state.floatbank10_f_air                      16802 non-n
ull  float64
 64  rougher.state.floatbank10_f_level                    16802 non-n
ull  float64
 65  secondary_cleaner.output.tail_ag                     16776 non-n
ull  float64
 66  secondary_cleaner.output.tail_pb                     16764 non-n
ull  float64
 67  secondary_cleaner.output.tail_sol                    14874 non-n
ull  float64
 68  secondary_cleaner.output.tail_au                     16778 non-n
ull  float64
 69  secondary_cleaner.state.floatbank2_a_air             16497 non-n
ull  float64
 70  secondary_cleaner.state.floatbank2_a_level           16751 non-n
ull  float64
 71  secondary_cleaner.state.floatbank2_b_air             16705 non-n
ull  float64
 72  secondary_cleaner.state.floatbank2_b_level           16748 non-n
ull  float64
 73  secondary_cleaner.state.floatbank3_a_air             16763 non-n
ull  float64
 74  secondary_cleaner.state.floatbank3_a_level           16747 non-n
ull  float64
 75  secondary_cleaner.state.floatbank3_b_air             16752 non-n
ull  float64
 76  secondary_cleaner.state.floatbank3_b_level           16750 non-n
ull  float64
 77  secondary_cleaner.state.floatbank4_a_air             16731 non-n
ull  float64
 78  secondary_cleaner.state.floatbank4_a_level           16747 non-n
ull  float64
 79  secondary_cleaner.state.floatbank4_b_air             16768 non-n
ull  float64
 80  secondary_cleaner.state.floatbank4_b_level           16767 non-n
ull  float64
 81  secondary_cleaner.state.floatbank5_a_air             16775 non-n
ull  float64
 82  secondary_cleaner.state.floatbank5_a_level           16775 non-n
ull  float64
 83  secondary_cleaner.state.floatbank5_b_air             16775 non-n
ull  float64
 84  secondary_cleaner.state.floatbank5_b_level           16776 non-n
ull  float64
 85  secondary_cleaner.state.floatbank6_a_air             16757 non-n
ull  float64
 86  secondary_cleaner.state.floatbank6_a_level           16775 non-n
ull  float64
dtypes: float64(86), object(1)
memory usage: 11.2+ MB
```

In [7]:

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5856 entries, 0 to 5855
Data columns (total 53 columns):
 #   Column                                      Non-Null Count  Dty
pe
---  ------                                      --------------  ---
--
 0   date                                        5856 non-null   obj
ect
 1   primary_cleaner.input.sulfate               5554 non-null   flo
at64
 2   primary_cleaner.input.depressant            5572 non-null   flo
at64
 3   primary_cleaner.input.feed_size             5856 non-null   flo
at64
 4   primary_cleaner.input.xanthate              5690 non-null   flo
at64
 5   primary_cleaner.state.floatbank8_a_air      5840 non-null   flo
at64
 6   primary_cleaner.state.floatbank8_a_level    5840 non-null   flo
at64
 7   primary_cleaner.state.floatbank8_b_air      5840 non-null   flo
at64
 8   primary_cleaner.state.floatbank8_b_level    5840 non-null   flo
at64
 9   primary_cleaner.state.floatbank8_c_air      5840 non-null   flo
at64
 10  primary_cleaner.state.floatbank8_c_level    5840 non-null   flo
at64
 11  primary_cleaner.state.floatbank8_d_air      5840 non-null   flo
at64
 12  primary_cleaner.state.floatbank8_d_level    5840 non-null   flo
at64
 13  rougher.input.feed_ag                       5840 non-null   flo
at64
 14  rougher.input.feed_pb                       5840 non-null   flo
at64
 15  rougher.input.feed_rate                     5816 non-null   flo
at64
 16  rougher.input.feed_size                     5834 non-null   flo
at64
 17  rougher.input.feed_sol                      5789 non-null   flo
at64
 18  rougher.input.feed_au                       5840 non-null   flo
at64
 19  rougher.input.floatbank10_sulfate           5599 non-null   flo
at64
 20  rougher.input.floatbank10_xanthate          5733 non-null   flo
at64
 21  rougher.input.floatbank11_sulfate           5801 non-null   flo
at64
 22  rougher.input.floatbank11_xanthate          5503 non-null   flo
at64
 23  rougher.state.floatbank10_a_air             5839 non-null   flo
at64
 24  rougher.state.floatbank10_a_level           5840 non-null   flo
at64
 25  rougher.state.floatbank10_b_air             5839 non-null   flo
at64
 26  rougher.state.floatbank10_b_level           5840 non-null   flo
at64
```

```
 27  rougher.state.floatbank10_c_air             5839 non-null   flo
at64
 28  rougher.state.floatbank10_c_level           5840 non-null   flo
at64
 29  rougher.state.floatbank10_d_air             5839 non-null   flo
at64
 30  rougher.state.floatbank10_d_level           5840 non-null   flo
at64
 31  rougher.state.floatbank10_e_air             5839 non-null   flo
at64
 32  rougher.state.floatbank10_e_level           5840 non-null   flo
at64
 33  rougher.state.floatbank10_f_air             5839 non-null   flo
at64
 34  rougher.state.floatbank10_f_level           5840 non-null   flo
at64
 35  secondary_cleaner.state.floatbank2_a_air    5836 non-null   flo
at64
 36  secondary_cleaner.state.floatbank2_a_level  5840 non-null   flo
at64
 37  secondary_cleaner.state.floatbank2_b_air    5833 non-null   flo
at64
 38  secondary_cleaner.state.floatbank2_b_level  5840 non-null   flo
at64
 39  secondary_cleaner.state.floatbank3_a_air    5822 non-null   flo
at64
 40  secondary_cleaner.state.floatbank3_a_level  5840 non-null   flo
at64
 41  secondary_cleaner.state.floatbank3_b_air    5840 non-null   flo
at64
 42  secondary_cleaner.state.floatbank3_b_level  5840 non-null   flo
at64
 43  secondary_cleaner.state.floatbank4_a_air    5840 non-null   flo
at64
 44  secondary_cleaner.state.floatbank4_a_level  5840 non-null   flo
at64
 45  secondary_cleaner.state.floatbank4_b_air    5840 non-null   flo
at64
 46  secondary_cleaner.state.floatbank4_b_level  5840 non-null   flo
at64
 47  secondary_cleaner.state.floatbank5_a_air    5840 non-null   flo
at64
 48  secondary_cleaner.state.floatbank5_a_level  5840 non-null   flo
at64
 49  secondary_cleaner.state.floatbank5_b_air    5840 non-null   flo
at64
 50  secondary_cleaner.state.floatbank5_b_level  5840 non-null   flo
at64
 51  secondary_cleaner.state.floatbank6_a_air    5840 non-null   flo
at64
 52  secondary_cleaner.state.floatbank6_a_level  5840 non-null   flo
at64
dtypes: float64(52), object(1)
memory usage: 2.4+ MB
```

Notes for preprocessing:

- There are 16860 observations in the train set and 5856 observations in the test set, so the test set is around 26% of the full dataset;
- There are 87 features in the train set and only 53 features in the test set. Some parameters are not available in the test set because they were measured and/or calculated much later. We will analyze this point further in more detail;
- There are 2 target variables: `rougher.output.recovery` and `final.output.recovery` ;
- Each feature data type is float except for the `date` column, it should be converted to datetime format;
- There are quite a few missing values to fill.

Recovery calculation check

Let's use the following formula to check whether `rougher.output.recovery` was calculated correctly in the train set:

$$\text{Recovery} = \frac{C \times (F - T)}{F \times (C - T)} \times 100\%$$

where:

- *C* — share of gold in the concentrate right after flotation (for finding the rougher concentrate recovery)
- *F* — share of gold in the feed before flotation (for finding the rougher concentrate recovery)
- *T* — share of gold in the rougher tails right after flotation (for finding the rougher concentrate recovery)

In [8]:

```
C = df_train['rougher.output.concentrate_au']
F = df_train['rougher.input.feed_au']
T = df_train['rougher.output.tail_au']

df_train['recovery_calc'] = (100* ((C*(F-T)) / (F*(C-T)))).round(6)
```

In [9]:

```
df_train[['recovery_calc', 'rougher.output.recovery']][0:5]
```

Out[9]:

	recovery_calc	rougher.output.recovery
0	87.107763	87.107763
1	86.843261	86.843261
2	86.842308	86.842308
3	87.226430	87.226430
4	86.688794	86.688794

The first 5 rows are identical, let's calculate MAE to make sure it holds for all observations.

In [10]:

```
MAE = (df_train['recovery_calc'] - df_train['rougher.output.recovery']).abs().me
an()
MAE
```

Out[10]:

```
2.4482453097445186e-07
```

The MAE value is very close to 0, which means the `rougher.output.recovery` was calculated correctly.

Missing features analysis

In [11]:

```
missing_col = list(set(df_train.columns)-set(df_test.columns))
sorted(missing_col)
```

Out[11]:

```
['final.output.concentrate_ag',
 'final.output.concentrate_au',
 'final.output.concentrate_pb',
 'final.output.concentrate_sol',
 'final.output.recovery',
 'final.output.tail_ag',
 'final.output.tail_au',
 'final.output.tail_pb',
 'final.output.tail_sol',
 'primary_cleaner.output.concentrate_ag',
 'primary_cleaner.output.concentrate_au',
 'primary_cleaner.output.concentrate_pb',
 'primary_cleaner.output.concentrate_sol',
 'primary_cleaner.output.tail_ag',
 'primary_cleaner.output.tail_au',
 'primary_cleaner.output.tail_pb',
 'primary_cleaner.output.tail_sol',
 'recovery_calc',
 'rougher.calculation.au_pb_ratio',
 'rougher.calculation.floatbank10_sulfate_to_au_feed',
 'rougher.calculation.floatbank11_sulfate_to_au_feed',
 'rougher.calculation.sulfate_to_au_concentrate',
 'rougher.output.concentrate_ag',
 'rougher.output.concentrate_au',
 'rougher.output.concentrate_pb',
 'rougher.output.concentrate_sol',
 'rougher.output.recovery',
 'rougher.output.tail_ag',
 'rougher.output.tail_au',
 'rougher.output.tail_pb',
 'rougher.output.tail_sol',
 'secondary_cleaner.output.tail_ag',
 'secondary_cleaner.output.tail_au',
 'secondary_cleaner.output.tail_pb',
 'secondary_cleaner.output.tail_sol']
```

The missing features are directly connected to the target variables (types **output** and **calculation**). It's only logical that they are not included in the test set as we are developing a model precisely to predict those 2 targets.

In order to be able to use an ML model, we need to remove these extra features from the train set, so that both train and test set have the same shapes.

Preprocessing

Data type change

As mentioned above, let's convert the `date` column into the datetime type.

In [12]:

```
df_train['date'] = pd.to_datetime(df_train['date'])
df_test['date'] = pd.to_datetime(df_test['date'])
```

Test targets

Let's use the full dataset to extract test targets and include them in the test set to be able to compare our predictions to the actual values.

In [13]:

```
df_full['date'] = pd.to_datetime(df_full['date'])
```

In [14]:

```
df_test = df_test.merge(df_full[['date', 'final.output.recovery', 'rougher.outpu
t.recovery']],
                        how='left', on='date')
```

Missing values

In [15]:

```
df_train.isnull().sum().mean()/len(df_train)
```

Out[15]:

```
0.021974414968187212
```

In [16]:

```
df_test.isnull().sum().mean()/len(df_test)
```

Out[16]:

```
0.010394932935916543
```

In both datasets there are a few missing values in almost each column, but their share is not significant (no more than 2%, on average), we will simply drop them.

In [17]:

```
df_train = df_train.dropna(how='any', axis=0)
df_train.reset_index(drop=True, inplace=True)

df_test = df_test.dropna(how='any', axis=0)
df_test.reset_index(drop=True, inplace=True)
```

In [18]:

```
df_train.isnull().sum().sum()
```

Out[18]:

0

In [19]:

```
df_train.shape
```

Out[19]:

```
(11017, 88)
```

In [20]:

```
df_test.isnull().sum().sum()
```

Out[20]:

0

In [21]:

```
df_test.shape
```

Out[21]:

```
(5229, 55)
```

All missing values were removed.

Duplicates

Let's check if any rows are duplicated.

In [22]:

```
df_train.duplicated().sum()
```

Out[22]:

0

In [23]:

```
df_test.duplicated().sum()
```

Out[23]:

0

EDA

Concentrations of metals

Let's see how the concentrations of metals (Au, Ag, Pb) change depending on the purification stage.

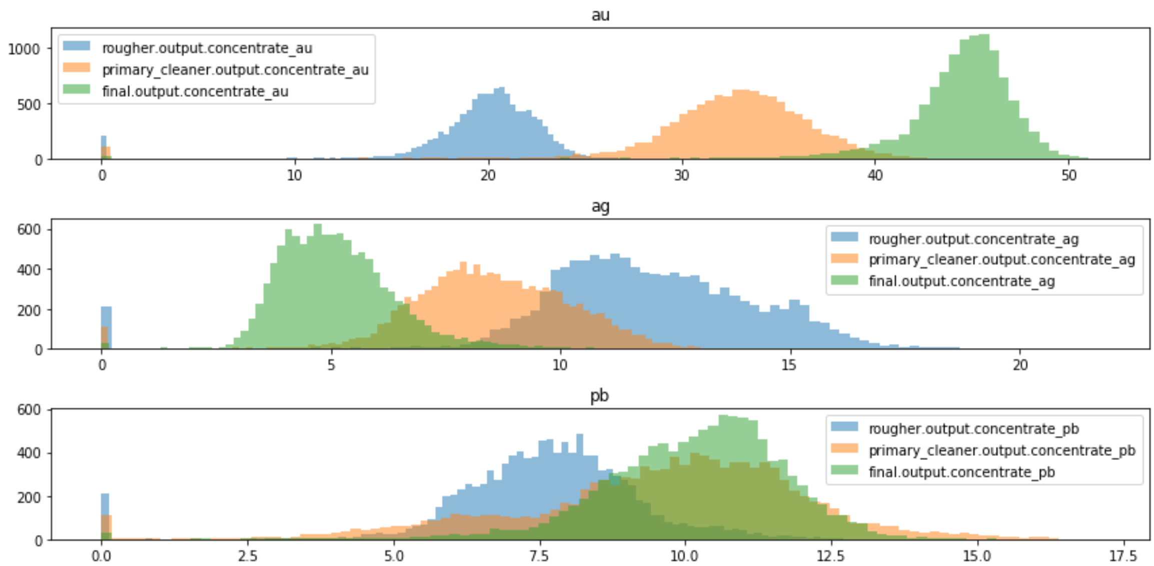
In [24]:

```
metals = ['au', 'ag', 'pb']
stages = ['rougher.output.concentrate', 'primary_cleaner.output.concentrate', 'f
inal.output.concentrate']
plt.figure(figsize=(12,6))

for i,metal in enumerate(metals):
    for stage in stages:
        plt.subplot(3,1,i+1)
        plt.hist(df_train[stage+'_'+metal], bins=100, label=stage+'_'+metal, alp
ha=.5);
    plt.title(metal)

    plt.legend()

plt.tight_layout()
```



First of all, we see around 2000 outliers for each metal and stage - values with 0 concentration of metals. We will need to remove them as they directly correlate with our targets.

As for the gold concentration (`au`), there is a clear trend towards quality improvement after each stage: the share of gold is higher and higher on average as we proceed with purification (from 20% to more than 45%, on average).

Interestingly, we see the opposite trend for silver concentration (`ag`): the more we purify the feed, the lower gets the share of this metal (from around 11% to less than 5%, on average).

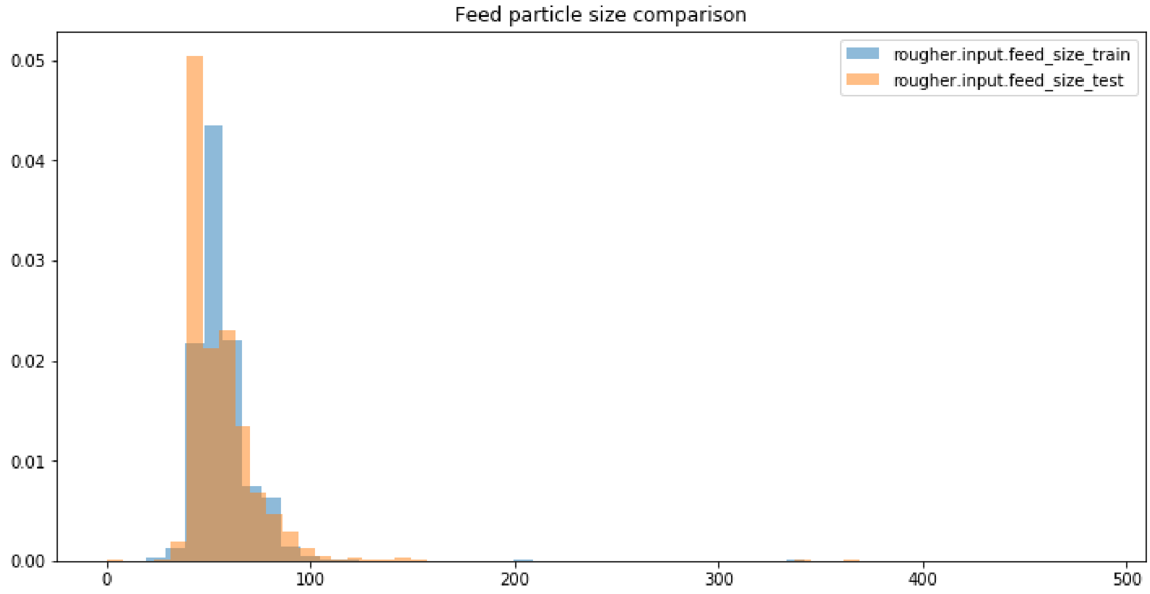
As for the lead concentration (`pb`), we can say that there is probably no need for the second purification stage, as the quality of this metal doesn't improve, on average. However, we see a slight improvement after the first stage (from around 8% to almost 11%, on average).

Feed particle size comparison

Let's compare the feed particle size distributions in the training set and in the test set. If the distributions vary significantly, the model evaluation will be incorrect.

In [25]:

```
plt.figure(figsize=(12,6))
plt.hist(df_train['rougher.input.feed_size'],bins=50,label='rougher.input.feed_size_train',alpha=.5, density=1)
plt.hist(df_test['rougher.input.feed_size'],bins=50,label='rougher.input.feed_size_test',alpha=.5, density=1)
plt.legend()
plt.title('Feed particle size comparison');
```



The two distributions are very close to each other, which means that we will not have a problem of applying a model trained on the train set to the test set.

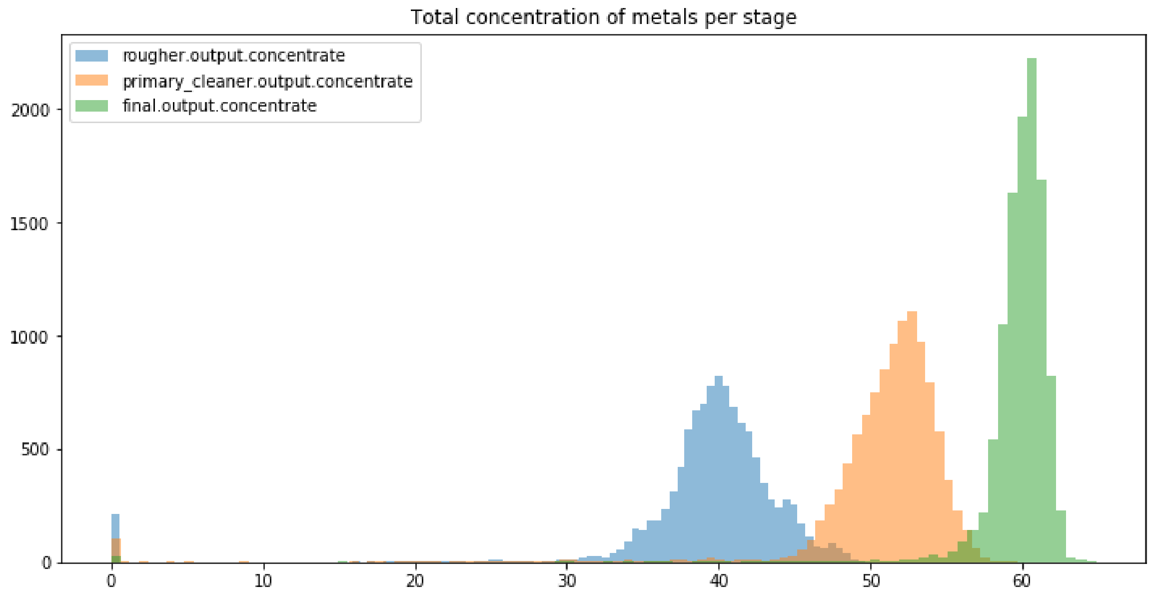
Outliers

Let's remove the outliers in the target variables. We will first transform any 0 values to NaNs and then drop them from the data sets because there amount is insignificant.

In [26]:

```
plt.figure(figsize=(12,6))

for stage in stages:
    df_train[stage+'_all_metals'] = 0
    for metal in metals:
        df_train[stage+'_all_metals'] += df_train[stage+'_'+metal]
    plt.hist(df_train[stage+'_all_metals'], bins=100, label=stage, alpha=.5)
    plt.title('Total concentration of metals per stage')
    plt.legend();
```



We can see that the total concentration of metals per stage does get better, on average.

Just as before we see multiple outliers around 0, let's remove them.

In [27]:

```
for i,metal in enumerate(metals):
    for stage in stages:
        df_train[df_train[stage+'_'+metal]<0.01] = np.nan
df_train = df_train.dropna(how='any', axis=0)
```

In [28]:

```
df_train.isnull().sum().sum()
```

Out[28]:

0

In [29]:

```
df_train.shape
```

Out[29]:

```
(10676, 91)
```

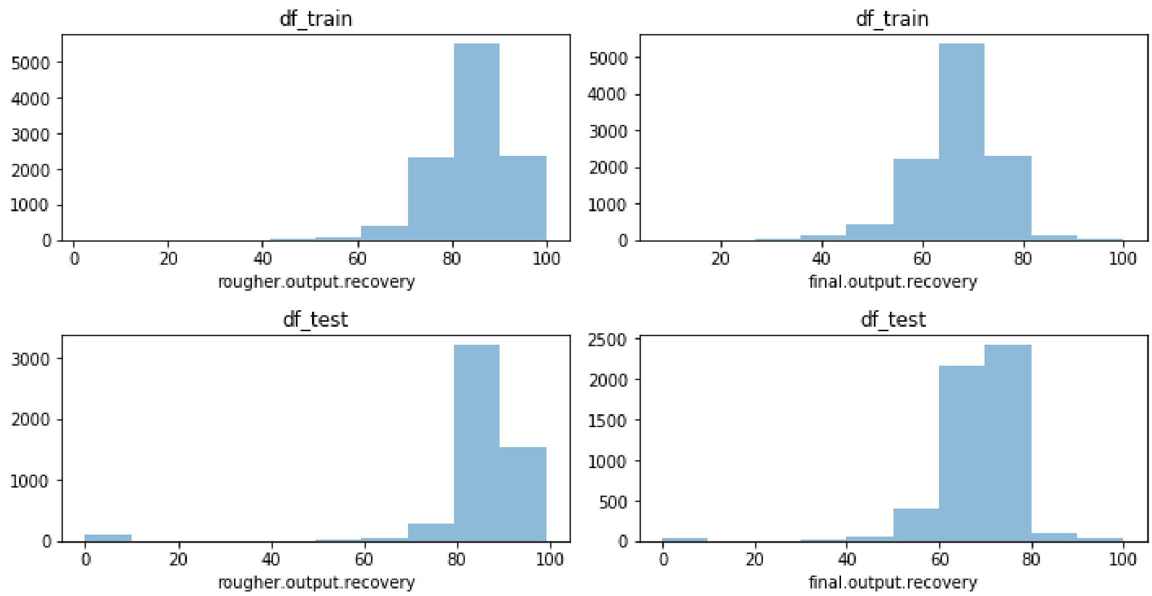
Target analysis

Finally, let's analyze our targets.

In [30]:

```
targets = ['rougher.output.recovery', 'final.output.recovery']
dfs = [df_train, df_test]
df_train.name = 'df_train'
df_test.name = 'df_test'

plt.figure(figsize=(10,10))
c=1
for i, df in enumerate(dfs):
    for target in targets:
        plt.subplot(4,2,c)
        plt.hist(df[target], alpha=.5)
        plt.title(df.name)
        plt.xlabel(target)
        c = c + 1
plt.tight_layout()
```



Train distribution looks ok, both targets in the test set have outliers around 0, let's remove them.

In [31]:

```
for target in targets:
    df_test[df_test[target]<0.01] = np.nan

df_test = df_test.dropna(how='any', axis=0)
```

In [32]:

```
df_test.isnull().sum().sum()
```

Out[32]:

0

In [33]:

```
df_test.shape
```

Out[33]:

(5105, 55)

Common columns

In the end, we will get rid of extra features in the train set.

In [34]:

```
common_columns =  list(set(df_train.columns).intersection(set(df_test.columns)))
df_train_filtered = df_train[common_columns]
df_train_filtered[['rougher.output.recovery', 'final.output.recovery']] = df_tra
in[['rougher.output.recovery', 'final.output.recovery']]
```

Standard scaling

Let's scale the features before modeling to be able to compare their coefficients in the later sections.

In [35]:

```
X_train = df_train_filtered.drop(['rougher.output.recovery', 'final.output.recov
ery', 'date'], axis=1)
X_test =  df_test.drop(['rougher.output.recovery', 'final.output.recovery', 'dat
e'], axis=1)
y_train = df_train_filtered[['rougher.output.recovery', 'final.output.recovery'
]].values
y_test = df_test[['rougher.output.recovery', 'final.output.recovery']].values

sc = ss()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Evaluation metric

Let's write a function to calculate the final sMAPE value.

sMAPE is a symmetric Mean Absolute Percentage Error.

It is similar to MAE, but is expressed in relative values instead of absolute ones. It equally takes into account the scale of both the target and the prediction.

Here’s how *sMAPE* is calculated:

$$sMAPE = \frac{1}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|) / 2} \times 100\%$$

Denotation:

- Value of target for the observation with the *i* index in the sample used to measure quality.
- Value of prediction for the observation with the *i* index, for example, in the test sample.
- Number of observations in the sample.
- Summation over all observations of the sample (***i*** takes values from 1 to *N*).

We need to predict two values:

1. rougher concentrate recovery `rougher.output.recovery`
2. final concentrate recovery `final.output.recovery`

The final metric includes the two values:

$$\text{Final sMAPE} = 25\% \times sMAPE(rougher) + 75\% \times sMAPE(final)$$

In [36]:

```
def smape(y_true, y_pred):
    return (np.abs(y_true-y_pred)/((np.abs(y_true) + np.abs(y_pred))/2)).mean()
```

In [37]:

```
def smape_final(y_true,y_pred):
    smape_rougher = smape(y_true[:,0], y_pred[:,0])
    smape_final = smape(y_true[:,1], y_pred[:,1])
    return 0.25*smape_rougher + 0.75*smape_final
```

Model selection

Now let's train our models on the train set and select the best model using the cross-validation technique.

In [38]:

```
LR = LinearRegression()
DT = DecisionTreeRegressor(random_state=12345)
RF = RandomForestRegressor(random_state=12345)
Lasso = linear_model.Lasso()
Ridge = linear_model.Ridge()
base_model = DummyRegressor(strategy='mean')
```

In [39]:

```
def crossval(model, X_train, y_train, cv):
    smape_score = make_scorer(smape_final)
    scores = cross_val_score(model, X_train, y_train, cv=cv, scoring=smape_score
)
    return scores.mean()
```

In [40]:

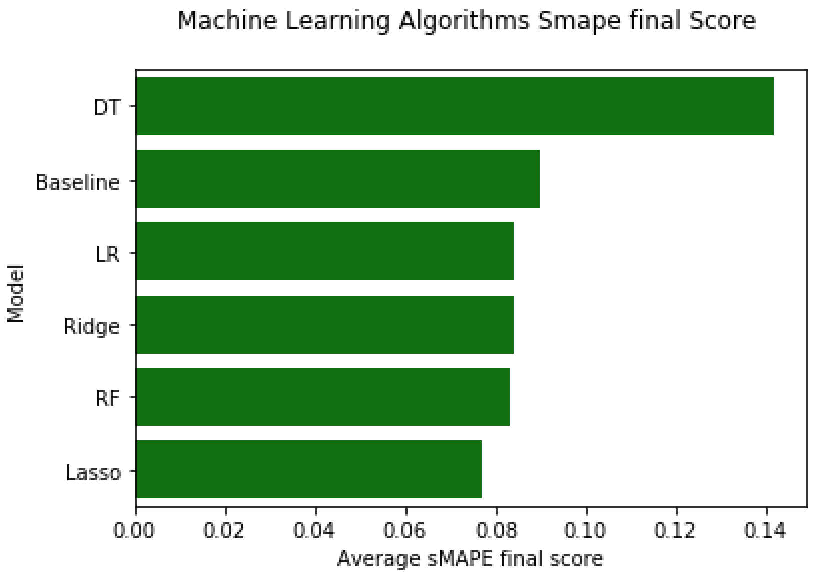
```
smape_final_Lasso = crossval(Lasso, X_train, y_train, 5)
smape_final_Ridge = crossval(Ridge, X_train, y_train, 5)
smape_final_LR = crossval(LR, X_train, y_train, 5)
smape_final_DT = crossval(DT, X_train, y_train, 5)
smape_final_RF = crossval(RF, X_train, y_train, 5)
smape_final_base = crossval(base_model, X_train, y_train, 5)
```

In [41]:

```
models = pd.DataFrame({
    'Model': ['Baseline', 'LR', 'DT', 'RF', 'Lasso', 'Ridge'],
    'Score': [smape_final_base, smape_final_LR, smape_final_DT, smape_final_RF,
smape_final_Lasso, smape_final_Ridge]})
sorted_by_score = models.sort_values(by='Score', ascending=False)
```

In [42]:

```
sns.barplot(x='Score', y = 'Model', data = sorted_by_score, color = 'g')
plt.title('Machine Learning Algorithms Smape final Score \n')
plt.xlabel('Average sMAPE final score')
plt.ylabel('Model');
```



Lasso regression model shows the best average score. Let's try to tune its hyperparameters.

Hyperparameter tuning

In [43]:

```
d = []
for alpha in np.arange(0.1,1,0.1):
    for tol in [0.01, 0.05, 0.001]:
        for max_iter in (100, 500, 1000):
            Lasso = linear_model.Lasso(alpha=alpha, tol=tol, max_iter=max_iter)
            smape_final_Lasso = crossval(Lasso, X_train, y_train, 5)
            d.append(
            {
                'alpha': alpha,
                'tol': tol,
                'max_iter': max_iter,
                'smape_final_Lasso':  smape_final_Lasso
            }
            )

best_param = pd.DataFrame(d).nlargest(1, ['smape_final_Lasso'], keep='first')
smape_final_RF = best_param['smape_final_Lasso'].values
best_param
```

Out[43]:

	alpha	tol	max_iter	smape_final_Lasso
3	0.1	0.05	100	0.08001

Test the model

In [44]:

```
Lasso = linear_model.Lasso(alpha=0.1, tol=0.001, max_iter=100)
Lasso.fit(X_train, y_train)
y_pred = Lasso.predict(X_test)
smape_final(y_test, y_pred)
```

Out[44]:

```
2.0
```

The test score is pretty bad even after hyper parameters tuning. The model seems to be overfitted to the train set. Let's try feature selection to reduce overfitting.

Feature importance

One of the goals of this report is to eliminate unprofitable features. We can do that by comparing feature importances.

In [45]:

```
LR.fit(X_train, y_train)
coeff_df = pd.DataFrame()
coeff_df['Feature'] = df_train_filtered.drop(['rougher.output.recovery','final.output.recovery','date'], axis=1).columns.values
coeff_df["Correlation"] = pd.Series(LR.coef_[0])

coeff_df.sort_values(by='Correlation', ascending=False)
```

Out[45]:

	Feature	Correlation
50	rougher.state.floatbank10_f_air	2.873959
41	rougher.input.feed_ag	1.969526
29	rougher.state.floatbank10_b_level	1.882530
9	rougher.input.feed_sol	1.829609
30	rougher.state.floatbank10_a_level	1.695842
20	primary_cleaner.input.sulfate	1.635028
43	secondary_cleaner.state.floatbank5_a_air	1.521347
35	rougher.input.floatbank11_sulfate	1.245344
38	primary_cleaner.state.floatbank8_b_air	1.223162
17	secondary_cleaner.state.floatbank3_a_air	1.167750
42	rougher.input.floatbank10_xanthate	1.153397
33	secondary_cleaner.state.floatbank2_a_air	1.106496
45	secondary_cleaner.state.floatbank5_b_level	1.069566
7	primary_cleaner.state.floatbank8_c_level	0.695207
36	secondary_cleaner.state.floatbank3_b_level	0.582116
3	secondary_cleaner.state.floatbank4_b_level	0.532246
6	rougher.input.floatbank11_xanthate	0.347046
13	primary_cleaner.input.feed_size	0.299679
46	rougher.input.feed_pb	0.237812
34	secondary_cleaner.state.floatbank2_a_level	0.175052
37	secondary_cleaner.state.floatbank4_a_level	0.161707
4	secondary_cleaner.state.floatbank4_b_air	0.125723
16	rougher.state.floatbank10_b_air	0.060345
51	rougher.input.feed_au	0.049011
21	primary_cleaner.state.floatbank8_d_level	0.047284
28	rougher.input.feed_size	-0.015142
18	rougher.state.floatbank10_f_level	-0.116173
47	primary_cleaner.input.xanthate	-0.136632
39	primary_cleaner.state.floatbank8_a_air	-0.140311
5	secondary_cleaner.state.floatbank5_b_air	-0.175401
27	primary_cleaner.input.depressant	-0.200300
15	primary_cleaner.state.floatbank8_b_level	-0.247245
40	rougher.state.floatbank10_d_level	-0.258105
0	primary_cleaner.state.floatbank8_a_level	-0.261074
10	secondary_cleaner.state.floatbank5_a_level	-0.357582
11	primary_cleaner.state.floatbank8_c_air	-0.359393

	Feature	Correlation
26	secondary_cleaner.state.floatbank4_a_air	-0.384321
23	secondary_cleaner.state.floatbank2_b_level	-0.545032
32	rougher.state.floatbank10_d_air	-0.593025
22	rougher.state.floatbank10_e_level	-0.630099
12	secondary_cleaner.state.floatbank2_b_air	-0.705941
24	secondary_cleaner.state.floatbank6_a_air	-0.742980
44	secondary_cleaner.state.floatbank6_a_level	-0.753929
31	rougher.state.floatbank10_c_air	-0.773621
19	primary_cleaner.state.floatbank8_d_air	-0.988076
49	secondary_cleaner.state.floatbank3_a_level	-1.069232
1	secondary_cleaner.state.floatbank3_b_air	-1.101443
25	rougher.state.floatbank10_c_level	-1.157193
48	rougher.state.floatbank10_a_air	-1.270705
14	rougher.input.feed_rate	-1.903286
8	rougher.state.floatbank10_e_air	-1.986217
2	rougher.input.floatbank10_sulfate	-2.558315

From the above table we can see that `rougher.state.floatbank10_e_air` and `rougher.state.floatbank10_f_air` have the highest impact on the target variables. The features with scores close to 0 have almost no impact on the targets and can be eliminated.

In [46]:

```
profitable_params = coeff_df.loc[abs(coeff_df["Correlation"]) > 0.1 , 'Feature']
```

In [47]:

```
df_train_params = df_train_filtered[profitable_params]
df_test_params = df_test[profitable_params]

df_train_params[['rougher.output.recovery', 'final.output.recovery']] = df_train
_filtered[['rougher.output.recovery', 'final.output.recovery']]
df_test_params[['rougher.output.recovery', 'final.output.recovery']] = df_test[[
'rougher.output.recovery', 'final.output.recovery']]
```

In [48]:

```
X_train = df_train_params.drop(['rougher.output.recovery', 'final.output.recover
y'], axis=1)
X_test =  df_test_params.drop(['rougher.output.recovery', 'final.output.recover
y'], axis=1)
y_train = df_train_params[['rougher.output.recovery', 'final.output.recovery']].
values
y_test = df_test_params[['rougher.output.recovery', 'final.output.recovery']].va
lues

sc = ss()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [49]:

```
Lasso = linear_model.Lasso(alpha=0.1, tol=0.001, max_iter=100)
Lasso.fit(X_train, y_train)
y_pred = Lasso.predict(X_test)
smape_final_score = smape_final(y_test, y_pred)
smape_final_score
```

Out[49]:

```
0.06794437817892128
```

Sanity check

Let's calculate the test baseline score to perform this check.

In [50]:

```
base_model = DummyRegressor(strategy='mean')
base_model.fit(X_train, y_train)
y_pred = base_model.predict(X_test)
smape_final_base = smape_final(y_test, y_pred)
smape_final_base
```

Out[50]:

```
0.07637178944673645
```

In [51]:

```
print(round((smape_final_base - smape_final_score)/smape_final_base,2)*100,'%')
```

```
11.0 %
```

Lasso model cross-validation score is lower than the baseline score by 11%.

Conclusion

In this project we have **developed a machine learning model for Zyfra, efficiency solutions for heavy industry company, that analyzes data on extraction and purification of gold from an ore and predicts the amount of recovered gold. The model helps to eliminate unprofitable parameters**.

First of all, we have familiarized ourselves with the data by performing the descriptive statistics. We found missing values, wrong `date` column data type.

Next, we checked the recovery calculation in the dataset. It turned out to be performed correctly as the MAE value between the `rougher.output.recovery` column and our calculations was very close to 0.

We found that some features are missing from the test set as they are directly connected to the target variables (types output and calculation).

In the preprocessing step we have converted `date` column data type to datetime, filled missing tenure values with the mean per each column and checked for duplicated values. In order to be able to use an ML model, we removed those extra features from the train set, so that both train and test set have the same shapes.

In the following section we have performed an **exploratory data analysis** and reached the following conclusions:

- As for the gold concentration (au), there is a clear trend towards quality improvement after each stage: the share of gold is higher and higher on average as we proceed with purification (from 20% to more than 45%, on average).
- Interestingly, we see the opposite trend for silver concentration (ag): the more we purify the feed, the lower gets the share of this metal (from around 11% to less than 5%, on average).
- As for the lead concentration (pb), we can say that there is probably no need for the second purification stage, as the quality of this metal doesn't improve, on average. However, we see a slight improvement after the first stage (from around 8% to almost 11%, on average).
- The train and test distributions of feed particle sizes are very close to each other, which means that we will not have a problem of applying a model trained on the train set to the test set.
- We found some outliers in the target variables and removed them.

In the next step we developed and tested, using cross-validation, several ML algorithms and tuned the best model's hyperparameters. Lasso regression model showed the lowest score.

The selected model appeared to be overfitting to the train set, so we decided to implement feature selection method to reduce the overfitting. We have measured feature importances using the coefficients from the Linear Regression model and identified the most profitable parameters.

Then we have tested our model on the test set. We have reached 6.8% error rate on the test set.

Finally, we have checked our model for sanity by comparing the final score to the baseline score. The final score of our model is lower (by 11%) than the baseline score.