Project Description

Prepare a prototype of a machine learning model for Zyfra. The company develops efficiency solutions for heavy industry. The model should predict the amount of gold recovered from gold ore. You have the data on extraction and purification. The mode l will help to optimize the production and eliminate unprofitable parameters.

Project instructions

1. Prepare the data

1.1. Open the files and look into the data.

Path to files:

/datasets/gold_recovery_train.csv
/datasets/gold_recovery_test.csv
/datasets/gold_recovery_full.csv

1.2. Check that recovery is calculated correctly. Using the training set, calculate recovery for the rougher.output.recovery feature. Find the MAE between your calculations and the feature values. Provide findings.

1.3. Analyze the features not available in the test set. What are these parameters? What is their type?
1.4. Perform data preprocessing.

2. Analyze the data

2.1. Take note of how the concentrations of metals (Au, Ag, Pb) change depending on the purification stage.

2.2. 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.
2.3. Consider the total concentrations of all substances at different stages: raw feed, rougher concentrate, and final concentrate. Do you notice any abnormal values in the total distribution? If you do, is it worth removing such values from both samples? Describe the findings and eliminate anomalies.

3. Build the model

3.1. Write a function to calculate the final sMAPE value.
3.2. Train different models. Evaluate them using cross-validation. Pick the best model and test it using the test sample. Provide findings.

In [1]:
```
import pandas as pd

import numpy as np
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error as mae
import plotly.express as px
from sklearn.metrics import mean_absolute_error, make_scorer
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
import warnings
warnings.filterwarnings('ignore')
import plotly.io as pio
pio.renderers.default = "SVG"

#Load our data sets
```

In [2]:
```
train_gold = pd.read_csv('gold_recovery_train.csv',parse_dates=True, infer_datetime_format=True, index_col='da

te
test_gold = pd.read_csv('gold_recovery_test.csv',parse_dates=True, infer_datetime_format=True, index_col='date'
)
gold = pd.read_csv('gold_recovery_full.csv',parse_dates=True, infer_datetime_format=True, index_col='date')
```

In [3]:
```
#Creating auxiliary functions
def miss_dup(data):
    print('Number of missing values in our data set:')
    print(data.isnull().sum())
    print()
    print('Number of duplicates in our data set:',data.duplicated().sum())
def show(data):
    print(data.info())
    print(data.describe())
    return data.sample(5)
```

In [4]:
```
#Checking what we have in our first data set
miss_dup(train_gold)
```

```
Number of missing values in our data set:
final.output.concentrate_ag                72
final.output.concentrate_pb                72
final.output.concentrate_sol              370
final.output.concentrate_au                71
final.output.recovery                    1521
                                           ...
secondary_cleaner.state.floatbank5_a_level    85
secondary_cleaner.state.floatbank5_b_air      85
secondary_cleaner.state.floatbank5_b_level    84
secondary_cleaner.state.floatbank6_a_air     103
secondary_cleaner.state.floatbank6_a_level    85
Length: 86, dtype: int64

Number of duplicates in our data set: 16
```

In [5]:
```
show(train_gold)
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 16860 entries, 2016-01-15 00:00:00 to 2018-08-18 10:59:59
Data columns (total 86 columns):
 #   Column                                      Non-Null Count  Dtype
---  ------                                      --------------  -----
 0   final.output.concentrate_ag                 16788 non-null  float64
 1   final.output.concentrate_pb                 16788 non-null  float64
```

```
 2    final.output.concentrate_sol                        16490 non-null  float64
 3    final.output.concentrate_au                         16789 non-null  float64
 4    final.output.recovery                               15339 non-null  float64
 5    final.output.tail_ag                                16794 non-null  float64
 6    final.output.tail_pb                                16677 non-null  float64
 7    final.output.tail_sol                               16715 non-null  float64
 8    final.output.tail_au                                16794 non-null  float64
 9    primary_cleaner.input.sulfate                       15553 non-null  float64
 10   primary_cleaner.input.depressant                    15598 non-null  float64
 11   primary_cleaner.input.feed_size                     16860 non-null  float64
 12   primary_cleaner.input.xanthate                      15875 non-null  float64
 13   primary_cleaner.output.concentrate_ag               16778 non-null  float64
 14   primary_cleaner.output.concentrate_pb               16502 non-null  float64
 15   primary_cleaner.output.concentrate_sol              16224 non-null  float64
 16   primary_cleaner.output.concentrate_au               16778 non-null  float64
 17   primary_cleaner.output.tail_ag                      16777 non-null  float64
 18   primary_cleaner.output.tail_pb                      16761 non-null  float64
 19   primary_cleaner.output.tail_sol                     16579 non-null  float64
 20   primary_cleaner.output.tail_au                      16777 non-null  float64
 21   primary_cleaner.state.floatbank8_a_air              16820 non-null  float64
 22   primary_cleaner.state.floatbank8_a_level            16827 non-null  float64
 23   primary_cleaner.state.floatbank8_b_air              16820 non-null  float64
 24   primary_cleaner.state.floatbank8_b_level            16833 non-null  float64
 25   primary_cleaner.state.floatbank8_c_air              16822 non-null  float64
 26   primary_cleaner.state.floatbank8_c_level            16833 non-null  float64
 27   primary_cleaner.state.floatbank8_d_air              16821 non-null  float64
 28   primary_cleaner.state.floatbank8_d_level            16833 non-null  float64
 29   rougher.calculation.sulfate_to_au_concentrate       16833 non-null  float64
 30   rougher.calculation.floatbank10_sulfate_to_au_feed  16833 non-null  float64
 31   rougher.calculation.floatbank11_sulfate_to_au_feed  16833 non-null  float64
 32   rougher.calculation.au_pb_ratio                     15618 non-null  float64
 33   rougher.input.feed_ag                               16778 non-null  float64
 34   rougher.input.feed_pb                               16632 non-null  float64
 35   rougher.input.feed_rate                             16347 non-null  float64
 36   rougher.input.feed_size                             16443 non-null  float64
 37   rougher.input.feed_sol                              16568 non-null  float64
 38   rougher.input.feed_au                               16777 non-null  float64
 39   rougher.input.floatbank10_sulfate                   15816 non-null  float64
 40   rougher.input.floatbank10_xanthate                  16514 non-null  float64
 41   rougher.input.floatbank11_sulfate                   16237 non-null  float64
 42   rougher.input.floatbank11_xanthate                  14956 non-null  float64
 43   rougher.output.concentrate_ag                       16778 non-null  float64
 44   rougher.output.concentrate_pb                       16778 non-null  float64
 45   rougher.output.concentrate_sol                      16698 non-null  float64
 46   rougher.output.concentrate_au                       16778 non-null  float64
 47   rougher.output.recovery                             14287 non-null  float64
 48   rougher.output.tail_ag                              14610 non-null  float64
 49   rougher.output.tail_pb                              16778 non-null  float64
 50   rougher.output.tail_sol                             14611 non-null  float64
 51   rougher.output.tail_au                              14611 non-null  float64
 52   rougher.state.floatbank10_a_air                     16807 non-null  float64
 53   rougher.state.floatbank10_a_level                   16807 non-null  float64
 54   rougher.state.floatbank10_b_air                     16807 non-null  float64
 55   rougher.state.floatbank10_b_level                   16807 non-null  float64
 56   rougher.state.floatbank10_c_air                     16807 non-null  float64
 57   rougher.state.floatbank10_c_level                   16814 non-null  float64
 58   rougher.state.floatbank10_d_air                     16802 non-null  float64
 59   rougher.state.floatbank10_d_level                   16809 non-null  float64
 60   rougher.state.floatbank10_e_air                     16257 non-null  float64
 61   rougher.state.floatbank10_e_level                   16809 non-null  float64
 62   rougher.state.floatbank10_f_air                     16802 non-null  float64
 63   rougher.state.floatbank10_f_level                   16802 non-null  float64
 64   secondary_cleaner.output.tail_ag                    16776 non-null  float64
 65   secondary_cleaner.output.tail_pb                    16764 non-null  float64
 66   secondary_cleaner.output.tail_sol                   14874 non-null  float64
 67   secondary_cleaner.output.tail_au                    16778 non-null  float64
 68   secondary_cleaner.state.floatbank2_a_air            16497 non-null  float64
 69   secondary_cleaner.state.floatbank2_a_level          16751 non-null  float64
 70   secondary_cleaner.state.floatbank2_b_air            16705 non-null  float64
 71   secondary_cleaner.state.floatbank2_b_level          16748 non-null  float64
 72   secondary_cleaner.state.floatbank3_a_air            16763 non-null  float64
 73   secondary_cleaner.state.floatbank3_a_level          16747 non-null  float64
 74   secondary_cleaner.state.floatbank3_b_air            16752 non-null  float64
 75   secondary_cleaner.state.floatbank3_b_level          16750 non-null  float64
 76   secondary_cleaner.state.floatbank4_a_air            16731 non-null  float64
 77   secondary_cleaner.state.floatbank4_a_level          16747 non-null  float64
 78   secondary_cleaner.state.floatbank4_b_air            16768 non-null  float64
 79   secondary_cleaner.state.floatbank4_b_level          16767 non-null  float64
 80   secondary_cleaner.state.floatbank5_a_air            16775 non-null  float64
 81   secondary_cleaner.state.floatbank5_a_level          16775 non-null  float64
 82   secondary_cleaner.state.floatbank5_b_air            16775 non-null  float64
 83   secondary_cleaner.state.floatbank5_b_level          16776 non-null  float64
 84   secondary_cleaner.state.floatbank6_a_air            16757 non-null  float64
 85   secondary_cleaner.state.floatbank6_a_level          16775 non-null  float64
dtypes: float64(86)
memory usage: 11.2 MB
None
       final.output.concentrate_ag  final.output.concentrate_pb  \
count                 16788.000000                 16788.000000
```

```
mean                 4.716907              9.113559
std                  2.096718              3.389495
min                  0.000000              0.000000
25%                  3.971262              8.825748
50%                  4.869346             10.065316
75%                  5.821176             11.054809
max                 16.001945             17.031899

       final.output.concentrate_sol  final.output.concentrate_au  \
count                16490.000000                 16789.000000
mean                     8.301123                    39.467217
std                      3.825760                    13.917227
min                      0.000000                     0.000000
25%                      6.939185                    42.055722
50%                      8.557228                    44.498874
75%                     10.289741                    45.976222
max                     18.124851                    53.611374

          final.output.recovery   final.output.tail_ag   final.output.tail_pb   \
count                15339.000000           16794.000000           16677.000000
mean                    67.213166               8.757048               2.360327
std                     11.960446               3.634103               1.215576
min                      0.000000               0.000000               0.000000
25%                     62.625685               7.610544               1.641604
50%                     67.644601               9.220393               2.453690
75%                     72.824595              10.971110               3.192404
max                    100.000000              19.552149               6.086532

       final.output.tail_sol  final.output.tail_au  \
count           16715.000000          16794.000000
mean                9.303932              2.687512
std                 4.263208              1.272757
min                 0.000000              0.000000
25%                 7.870275              2.172953
50%                10.021968              2.781132
75%                11.648573              3.416936
max                22.317730              9.789625

       primary_cleaner.input.sulfate  ...  \ count                15553.000000  ...
mean                    129.479789  ...    std                    45.386931  ...
min                       0.000003  ...    25%                   103.064021  ...
50%                     131.783108  ...    75%                   159.539839  ...
max                     251.999948  ...

       secondary_cleaner.state.floatbank4_a_air  \
count                     16731.000000
mean                         19.101874    std                      6.883163
min                           0.000000    25%                     14.508299
50%                          19.986958    75%                     24.983961
max                          60.000000

       secondary_cleaner.state.floatbank4_a_level  \
count                      16747.000000
mean                        -494.164481
std                           84.803334
min                         -799.920713
25%                         -500.837689
50%                         -499.778379
75%                         -494.648754
max                         -127.692333

       secondary_cleaner.state.floatbank4_b_air  \
count                     16768.000000
mean                         14.778164    std                      5.999149
min                           0.000000    25%                     10.741388
50%                          14.943933    75%                     20.023751
max                          28.003828

       secondary_cleaner.state.floatbank4_b_level  \
count                      16767.000000
mean                        -476.600082
std                           89.381172
min                         -800.021781
25%                         -500.269182
50%                         -499.593286
75%                         -400.137948
max                          -71.472472

       secondary_cleaner.state.floatbank5_a_air  \

count                     16775.000000
 mean                         15.779488
std                           6.834703   min                    -0.423260
 25%                          10.977713   50%                    15.998340
 75%                          20.000701   max                    63.116298
```

c

```
       secondary_cleaner.state.floatbank5_a_level  \
count                                16775.000000
mean                                  -500.230146
std                                     76.983542
min                                   -799.741097
25%                                   -500.530594
50%                                   -499.784231
75%                                   -496.531781
max                     -275.073125

     secondary_cleaner.state.floatbank5_b_air  \
count                     16775.000000
mean                         12.377241
std                           6.219989
min                           0.427084
25%                           8.925586
50%                          11.092839
75%                          15.979467
max                          39.846228

       secondary_cleaner.state.floatbank5_b_level  \
count                                16776.000000
mean                                  -498.956257
std                                     82.146207
min                                   -800.258209
25%                                   -500.147603
50%                                   -499.933330
75%                                   -498.418000
max                     -120.190931

     secondary_cleaner.state.floatbank6_a_air  \
count                     16757.000000
mean                         18.429208
std                           6.958294
min                           0.024270
25%                          13.977626
50%                          18.034960
75%                          24.984992
max                          54.876806

     secondary_cleaner.state.floatbank6_a_level
count                     16775.000000
mean                       -521.801826
std                         77.170888
min                       -810.473526
25%                       -501.080595
50%                       -500.109898
75%                       -499.565540
max                        -39.784927

[8 rows x 86 columns]
```

Out[5]:

date	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concentrate_sol	final.output.concentrate_au	final.output.recovery
2018-06-13 11:59:59	3.933171	11.037830	6.727559	46.584006	71.990753
2018-06-17 21:59:59	4.202885	11.905214	6.588792	44.883515	68.642465
2017-07-01 11:59:59	5.375617	10.588572	7.620606	44.205162	61.719110
2018-07-02 19:59:59	3.480208	10.644975	6.703221	46.822778	70.860520
2017-01-11 19:59:59	4.129018	8.477903	13.136784	47.539575	62.695083

5 rows × 86 columns

In [6]:
```
#Test data set
miss_dup(test_gold)
```

```
Number of missing values in our data set:
primary_cleaner.input.sulfate               302
primary_cleaner.input.depressant            284
primary_cleaner.input.feed_size               0
primary_cleaner.input.xanthate              166
primary_cleaner.state.floatbank8_a_air       16
primary_cleaner.state.floatbank8_a_level     16
primary_cleaner.state.floatbank8_b_air       16
primary_cleaner.state.floatbank8_b_level     16
primary_cleaner.state.floatbank8_c_air       16
primary_cleaner.state.floatbank8_c_level     16
primary_cleaner.state.floatbank8_d_air       16
primary_cleaner.state.floatbank8_d_level     16
rougher.input.feed_ag                       16
rougher.input.feed_pb                       16
rougher.input.feed_rate                     40
rougher.input.feed_size                     22
rougher.input.feed_sol                      67
rougher.input.feed_au                       16
rougher.input.floatbank10_sulfate          257
rougher.input.floatbank10_xanthate         123
rougher.input.floatbank11_sulfate           55
rougher.input.floatbank11_xanthate         353
rougher.state.floatbank10_a_air             17
rougher.state.floatbank10_a_level           16
rougher.state.floatbank10_b_air             17
rougher.state.floatbank10_b_level           16
rougher.state.floatbank10_c_air             17
rougher.state.floatbank10_c_level           16
rougher.state.floatbank10_d_air             17
rougher.state.floatbank10_d_level           16
rougher.state.floatbank10_e_air             17
rougher.state.floatbank10_e_level           16
rougher.state.floatbank10_f_air             17
rougher.state.floatbank10_f_level           16
secondary_cleaner.state.floatbank2_a_air    20
secondary_cleaner.state.floatbank2_a_level  16
secondary_cleaner.state.floatbank2_b_air    23
secondary_cleaner.state.floatbank2_b_level  16
secondary_cleaner.state.floatbank3_a_air    34
secondary_cleaner.state.floatbank3_a_level  16
secondary_cleaner.state.floatbank3_b_air    16
secondary_cleaner.state.floatbank3_b_level  16
secondary_cleaner.state.floatbank4_a_air    16
secondary_cleaner.state.floatbank4_a_level  16
secondary_cleaner.state.floatbank4_b_air    16
secondary_cleaner.state.floatbank4_b_level  16
secondary_cleaner.state.floatbank5_a_air    16
secondary_cleaner.state.floatbank5_a_level  16
secondary_cleaner.state.floatbank5_b_air    16
secondary_cleaner.state.floatbank5_b_level  16
secondary_cleaner.state.floatbank6_a_air    16
secondary_cleaner.state.floatbank6_a_level  16
dtype: int64

Number of duplicates in our data set: 6
```

#Test data set

In [7]:
```
show(test_gold)
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5856 entries, 2016-09-01 00:59:59 to 2017-12-31 23:59:59
Data columns (total 52 columns):
 #   Column                                      Non-Null Count  Dtype
---  ------                                      --------------  -----
 0   primary_cleaner.input.sulfate               5554 non-null   float64
 1   primary_cleaner.input.depressant            5572 non-null   float64
 2   primary_cleaner.input.feed_size             5856 non-null   float64
 3   primary_cleaner.input.xanthate              5690 non-null   float64
 4   primary_cleaner.state.floatbank8_a_air      5840 non-null   float64
 5   primary_cleaner.state.floatbank8_a_level    5840 non-null   float64
 6   primary_cleaner.state.floatbank8_b_air      5840 non-null   float64
 7   primary_cleaner.state.floatbank8_b_level    5840 non-null   float64
 8   primary_cleaner.state.floatbank8_c_air      5840 non-null   float64
 9   primary_cleaner.state.floatbank8_c_level    5840 non-null   float64
 10  primary_cleaner.state.floatbank8_d_air      5840 non-null   float64
 11  primary_cleaner.state.floatbank8_d_level    5840 non-null   float64
 12  rougher.input.feed_ag                       5840 non-null   float64
 13  rougher.input.feed_pb                       5840 non-null   float64
 14  rougher.input.feed_rate                     5816 non-null   float64
 15  rougher.input.feed_size                     5834 non-null   float64
 16  rougher.input.feed_sol                      5789 non-null   float64
 17  rougher.input.feed_au                       5840 non-null   float64
 18  rougher.input.floatbank10_sulfate           5599 non-null   float64
 19  rougher.input.floatbank10_xanthate          5733 non-null   float64
 20  rougher.input.floatbank11_sulfate           5801 non-null   float64
 21  rougher.input.floatbank11_xanthate          5503 non-null   float64
 22  rougher.state.floatbank10_a_air             5839 non-null   float64
 23  rougher.state.floatbank10_a_level           5840 non-null   float64
```

```
 24  rougher.state.floatbank10_b_air          5839 non-null   float64
 25  rougher.state.floatbank10_b_level        5840 non-null   float64
 26  rougher.state.floatbank10_c_air          5839 non-null   float64
 27  rougher.state.floatbank10_c_level        5840 non-null   float64
 28  rougher.state.floatbank10_d_air          5839 non-null   float64
 29  rougher.state.floatbank10_d_level        5840 non-null   float64
 30  rougher.state.floatbank10_e_air          5839 non-null   float64
 31  rougher.state.floatbank10_e_level        5840 non-null   float64
 32  rougher.state.floatbank10_f_air          5839 non-null   float64
 33  rougher.state.floatbank10_f_level        5840 non-null   float64
 34  secondary_cleaner.state.floatbank2_a_air   5836 non-null   float64
 35  secondary_cleaner.state.floatbank2_a_level  5840 non-null   float64
 36  secondary_cleaner.state.floatbank2_b_air   5833 non-null   float64
 37  secondary_cleaner.state.floatbank2_b_level  5840 non-null   float64
 38  secondary_cleaner.state.floatbank3_a_air   5822 non-null   float64
 39  secondary_cleaner.state.floatbank3_a_level  5840 non-null   float64
 40  secondary_cleaner.state.floatbank3_b_air   5840 non-null   float64
 41  secondary_cleaner.state.floatbank3_b_level  5840 non-null   float64
 42  secondary_cleaner.state.floatbank4_a_air   5840 non-null   float64
 43  secondary_cleaner.state.floatbank4_a_level  5840 non-null   float64
 44  secondary_cleaner.state.floatbank4_b_air   5840 non-null   float64
 45  secondary_cleaner.state.floatbank4_b_level  5840 non-null   float64
 46  secondary_cleaner.state.floatbank5_a_air   5840 non-null   float64
 47  secondary_cleaner.state.floatbank5_a_level  5840 non-null   float64
 48  secondary_cleaner.state.floatbank5_b_air   5840 non-null   float64
 49  secondary_cleaner.state.floatbank5_b_level  5840 non-null   float64
 50  secondary_cleaner.state.floatbank6_a_air   5840 non-null   float64
 51  secondary_cleaner.state.floatbank6_a_level  5840 non-null   float64
dtypes: float64(52)
memory usage: 2.4 MB
None
       primary_cleaner.input.sulfate   primary_cleaner.input.depressant  \
count                    5554.000000                       5572.000000
mean                      170.515243                          8.482873
std                        49.608602                          3.353105
min                         0.000103                          0.000031
25%                       143.340022                          6.411500
50%                       176.103893                          8.023252
75%                       207.240761                         10.017725
max                  274.409626                  40.024582

       primary_cleaner.input.feed_size   primary_cleaner.input.xanthate  \
count                     5856.000000                       5690.000000
mean                        7.264651                          1.321420
std                         0.611526                          0.693246
min                         5.650000                          0.000003
25%                         6.885625                          0.888769
50%                         7.259333                          1.183362
75%                         7.650000                          1.763797
max                  15.500000                  5.433169

     primary_cleaner.state.floatbank8_a_air  \
count                     5840.000000
mean                      1481.990241    std                     310.453166
min                          0.000000    25%                    1497.190681
50%                       1554.659783    75%                    1601.681656
max                       2212.432090

     primary_cleaner.state.floatbank8_a_level  \
count                     5840.000000
mean                      -509.057796
std                         61.339256
min                       -799.773788
25%                       -500.455211
50%                       -499.997402
75%                       -499.575313
max                        -57.195404

     primary_cleaner.state.floatbank8_b_air  \
count                     5840.000000
mean                      1486.908670
std                       313.224286    min                       0.000000
25%                       1497.150234
50%                       1553.268084
75%                       1601.784707
max                       1975.147923

     primary_cleaner.state.floatbank8_b_level  \
count                     5840.000000
mean                      -511.743956
std                         67.139074
min                       -800.029078
25%                       -500.936639
50%                       -500.066588
75%                       -499.323361
```

max	-142.527229

	primary_cleaner.state.floatbank8_c_air
count	5840.000000
mean	1468.495216
std	309.980748
min	0.000000
25%	1437.050321
50%	1546.160672
75%	1600.785573
max	1715.053773

	primary_cleaner.state.floatbank8_c_level	...
count	5840.000000	...
mean	-509.741212	...
std	62.671873	...
min	-799.995127	...
25%	-501.300441	...
50%	-500.079537	...
75%	-499.009545	...
max	-150.937035	...

	secondary_cleaner.state.floatbank4_a_air
count	5840.000000
mean	15.636031
std	4.660835
min	0.000000
25%	12.057838
50%	17.001867
75%	18.030985
max	30.051797

	secondary_cleaner.state.floatbank4_a_level
count	5840.000000
mean	-516.266074
std	62.756748
min	-799.798523
25%	-501.054741
50%	-500.160145
75%	-499.441529
max	-401.565212

	secondary_cleaner.state.floatbank4_b_air
count	5840.000000
mean	13.145702
std	4.304086
min	0.000000
25%	11.880119
50%	14.952102
75%	15.940011
max	31.269706

	secondary_cleaner.state.floatbank4_b_level
count	5840.000000
mean	-476.338907
std	105.549424
min	-800.836914
25%	-500.419113
50%	-499.644328
75%	-401.523664
max	-6.506986

	secondary_cleaner.state.floatbank5_a_air
count	5840.000000
mean	12.308967
std	3.762827
min	-0.223393
25%	10.123459
50%	12.062877
75%	15.017881
max	25.258848

	secondary_cleaner.state.floatbank5_a_level
count	5840.000000
mean	-512.208126
std	58.864651
min	-799.661076
25%	-500.879383
50%	-500.047621
75%	-499.297033
max	-244.483566

	secondary_cleaner.state.floatbank5_b_air
count	5840.000000
mean	9.470986
std	3.312471
min	0.528083
25%	7.991208
50%	9.980774

```
75%                          11.992176
max                          14.090194

       secondary_cleaner.state.floatbank5_b_level  \
count                                5840.000000
mean                                  -505.017827
std                                     68.785898
min                                   -800.220337
25%                                   -500.223089
50%                                   -500.001338
75%                                   -499.722835
max                           -126.463446

     secondary_cleaner.state.floatbank6_a_air  \
count                         5840.000000
mean                            16.678722
std                              5.404514
min                             -0.079426
25%                             13.012422
50%                             16.007242
75%                             21.009076
max                             26.705889

     secondary_cleaner.state.floatbank6_a_level
count                          5840.000000
mean                           -512.351694
std                              69.919839
min                            -809.859706
25%                            -500.833821
50%                            -500.041085
75%                            -499.395621
max                             -29.093593

[8 rows x 52 columns]
```

Out[7]:

date	primary_cleaner.input.sulfate	primary_cleaner.input.depressant	primary_cleaner.input.feed_size	primary_cleaner.input.xanthate	primary
2016-12-16 05:59:59	180.404185	12.534573	6.90	1.314723	
2016-09-04 00:59:59	123.453451	8.050887	6.92	0.904880	
2016-10-10 20:59:59	197.793527	8.904348	7.57	1.000186	
2017-09-24 22:59:59	173.544887	5.991566	6.75	1.995377	
2017-11-27 09:59:59	219.979518	10.990236	7.10	2.217427	

5 rows × 52 columns

In [8]:
```
#Our main gold data set
miss_dup(gold)
```

```
Number of missing values in our data set:
final.output.concentrate_ag                89
final.output.concentrate_pb                87
final.output.concentrate_sol              385
final.output.concentrate_au                86
final.output.recovery                    1963
                                          ...
secondary_cleaner.state.floatbank5_a_level  101
secondary_cleaner.state.floatbank5_b_air    101
secondary_cleaner.state.floatbank5_b_level  100
secondary_cleaner.state.floatbank6_a_air    119
secondary_cleaner.state.floatbank6_a_level  101
Length: 86, dtype: int64

Number of duplicates in our data set: 22
```

In [9]:
```
#Main data set
show(gold)
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 22716 entries, 2016-01-15 00:00:00 to 2018-08-18 10:59:59
Data columns (total 86 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   final.output.concentrate_ag         22627 non-null  float64
 1   final.output.concentrate_pb         22629 non-null  float64
 2   final.output.concentrate_sol        22331 non-null  float64
```

```
 3    final.output.concentrate_au                         22630 non-null  float64
 4    final.output.recovery                               20753 non-null  float64
 5    final.output.tail_ag                                22633 non-null  float64
 6    final.output.tail_pb                                22516 non-null  float64
 7    final.output.tail_sol                               22445 non-null  float64
 8    final.output.tail_au                                22635 non-null  float64
 9    primary_cleaner.input.sulfate                       21107 non-null  float64
 10   primary_cleaner.input.depressant                    21170 non-null  float64
 11   primary_cleaner.input.feed_size                     22716 non-null  float64
 12   primary_cleaner.input.xanthate                      21565 non-null  float64
 13   primary_cleaner.output.concentrate_ag               22618 non-null  float64
 14   primary_cleaner.output.concentrate_pb               22268 non-null  float64
 15   primary_cleaner.output.concentrate_sol              21918 non-null  float64
 16   primary_cleaner.output.concentrate_au               22618 non-null  float64
 17   primary_cleaner.output.tail_ag                      22614 non-null  float64
 18   primary_cleaner.output.tail_pb                      22594 non-null  float64
 19   primary_cleaner.output.tail_sol                     22365 non-null  float64
 20   primary_cleaner.output.tail_au                      22617 non-null  float64
 21   primary_cleaner.state.floatbank8_a_air              22660 non-null  float64
 22   primary_cleaner.state.floatbank8_a_level            22667 non-null  float64
 23   primary_cleaner.state.floatbank8_b_air              22660 non-null  float64
 24   primary_cleaner.state.floatbank8_b_level            22673 non-null  float64
 25   primary_cleaner.state.floatbank8_c_air              22662 non-null  float64
 26   primary_cleaner.state.floatbank8_c_level            22673 non-null  float64
 27   primary_cleaner.state.floatbank8_d_air              22661 non-null  float64
 28   primary_cleaner.state.floatbank8_d_level            22673 non-null  float64
 29   rougher.calculation.sulfate_to_au_concentrate       22672 non-null  float64
 30   rougher.calculation.floatbank10_sulfate_to_au_feed  22672 non-null  float64
 31   rougher.calculation.floatbank11_sulfate_to_au_feed  22672 non-null  float64
 32   rougher.calculation.au_pb_ratio                     21089 non-null  float64
 33   rougher.input.feed_ag                               22618 non-null  float64
 34   rougher.input.feed_pb                               22472 non-null  float64
 35   rougher.input.feed_rate                             22163 non-null  float64
 36   rougher.input.feed_size                             22277 non-null  float64
 37   rougher.input.feed_sol                              22357 non-null  float64
 38   rougher.input.feed_au                               22617 non-null  float64
 39   rougher.input.floatbank10_sulfate                   21415 non-null  float64
 40   rougher.input.floatbank10_xanthate                  22247 non-null  float64
 41   rougher.input.floatbank11_sulfate                   22038 non-null  float64
 42   rougher.input.floatbank11_xanthate                  20459 non-null  float64
 43   rougher.output.concentrate_ag                       22618 non-null  float64
 44   rougher.output.concentrate_pb                       22618 non-null  float64
 45   rougher.output.concentrate_sol                      22526 non-null  float64
 46   rougher.output.concentrate_au                       22618 non-null  float64
 47   rougher.output.recovery                             19597 non-null  float64
 48   rougher.output.tail_ag                              19979 non-null  float64
 49   rougher.output.tail_pb                              22618 non-null  float64
 50   rougher.output.tail_sol                             19980 non-null  float64
 51   rougher.output.tail_au                              19980 non-null  float64
 52   rougher.state.floatbank10_a_air                     22646 non-null  float64
 53   rougher.state.floatbank10_a_level                   22647 non-null  float64
 54   rougher.state.floatbank10_b_air                     22646 non-null  float64
 55   rougher.state.floatbank10_b_level                   22647 non-null  float64
 56   rougher.state.floatbank10_c_air                     22646 non-null  float64
 57   rougher.state.floatbank10_c_level                   22654 non-null  float64
 58   rougher.state.floatbank10_d_air                     22641 non-null  float64
 59   rougher.state.floatbank10_d_level                   22649 non-null  float64
 60   rougher.state.floatbank10_e_air                     22096 non-null  float64
 61   rougher.state.floatbank10_e_level                   22649 non-null  float64
 62   rougher.state.floatbank10_f_air                     22641 non-null  float64
 63   rougher.state.floatbank10_f_level                   22642 non-null  float64
 64   secondary_cleaner.output.tail_ag                    22616 non-null  float64
 65   secondary_cleaner.output.tail_pb                    22600 non-null  float64
 66   secondary_cleaner.output.tail_sol                   20501 non-null  float64
 67   secondary_cleaner.output.tail_au                    22618 non-null  float64
 68   secondary_cleaner.state.floatbank2_a_air            22333 non-null  float64
 69   secondary_cleaner.state.floatbank2_a_level          22591 non-null  float64
 70   secondary_cleaner.state.floatbank2_b_air            22538 non-null  float64
 71   secondary_cleaner.state.floatbank2_b_level          22588 non-null  float64
 72   secondary_cleaner.state.floatbank3_a_air            22585 non-null  float64
 73   secondary_cleaner.state.floatbank3_a_level          22587 non-null  float64
 74   secondary_cleaner.state.floatbank3_b_air            22592 non-null  float64
 75   secondary_cleaner.state.floatbank3_b_level          22590 non-null  float64
 76   secondary_cleaner.state.floatbank4_a_air            22571 non-null  float64
 77   secondary_cleaner.state.floatbank4_a_level          22587 non-null  float64
 78   secondary_cleaner.state.floatbank4_b_air            22608 non-null  float64
 79   secondary_cleaner.state.floatbank4_b_level          22607 non-null  float64
 80   secondary_cleaner.state.floatbank5_a_air            22615 non-null  float64
 81   secondary_cleaner.state.floatbank5_a_level          22615 non-null  float64
 82   secondary_cleaner.state.floatbank5_b_air            22615 non-null  float64
 83   secondary_cleaner.state.floatbank5_b_level          22616 non-null  float64
 84   secondary_cleaner.state.floatbank6_a_air            22597 non-null  float64
 85   secondary_cleaner.state.floatbank6_a_level          22615 non-null  float64
dtypes: float64(86)
memory usage: 15.1 MB
None
       final.output.concentrate_ag  final.output.concentrate_pb  \
count                 22627.000000                 22629.000000
mean                      4.781559                     9.095308
```

```
std                 2.030128              3.230797
min                 0.000000              0.000000
25%                 4.018525              8.750171
50%                 4.953729              9.914519
75%                 5.862593             10.929839
max                16.001945             17.031899

       final.output.concentrate_sol  final.output.concentrate_a
u \
count             22331.000000            22630.000000
mean                  8.640317               40.001172
std                   3.785035               13.398062
min                   0.000000                0.000000
25%                   7.116799               42.383721
50%                   8.908792               44.653436
75%                  10.705824               46.111999
max                  19.615720               53.611374

        final.output.recovery  final.output.tail_ag  final.output.t
ail_pb                                                            \
count         20753.000000         22633.000000         22516.0
00000
mean             67.447488            8.923690            2.48825
2
std              11.616034            3.517917            1.189407
min               0.000000            0.000000              0.00000
0
25%              63.282393            7.684016            1.805376

50%              68.322258            9.484369            2.653001

75%              72.950836           11.084557            3.287790

max             100.000000           19.552149             6.086532


       final.output.tail_sol  final.output.tail_au \
count         22445.000000         22635.000000
mean              9.523632             2.827459
std               4.079739             1.262834
min               0.000000             0.000000
25%               8.143576             2.303108
50%              10.212998             2.913794
75%              11.860824             3.555077
max              22.861749             9.789625

       primary_cleaner.input.sulfate  ... \
count               21107.000000  ...
mean                  140.277672  ...
std                    49.919004  ...
min                     0.000003  ...
25%                   110.177081  ...
50%                   141.330501  ...
75%                   174.049914  ...
max                   274.409626  ...

       secondary_cleaner.state.floatbank4_a_air \
count                      22571.000000
mean                          18.205125
std                            6.560700
min                            0.000000
25%                           14.095940
50%                           18.007326
75%                           22.998194
max                           60.000000

       secondary_cleaner.state.floatbank4_a_level \
count                       22587.000000
mean                         -499.878977
std                            80.273964
min                          -799.920713
25%                          -500.896232
50%                          -499.917108
75%                          -498.361545
max                          -127.692333

       secondary_cleaner.state.floatbank4_b_air \
count                      22608.000000
mean                          14.356474
std                            5.655791
min                            0.000000
25%                           10.882675
50%                           14.947646
75%                           17.977502
max                           31.269706

       secondary_cleaner.state.floatbank4_b_level \
count                       22607.000000
mean                         -476.532613
std                            93.822791
min                          -800.836914
25%                          -500.309169
50%                          -499.613292
```

```
mean                                 14.883276
std                                   6.372811
min                                  -0.423260
25%                                  10.941299
50%                                  14.859117
75%                                  18.014914
max                                  63.116298

       secondary_cleaner.state.floatbank5_a
_level                                       \
count                            22615.000000
mean                              -503.323288
std                                 72.925589
min                               -799.741097
25%                               -500.628697
50%                               -499.865158
75%                               -498.489381
max                               -244.483566

      secondary_cleaner.state.floatbank5_b
_air  \
count                        22615.000000
mean                            11.626743
std                              5.757449
min                              0.427084
25%                              8.037533
50%                             10.989756
75%                             14.001193
max                             39.846228

       secondary_cleaner.state.floatbank5_b
_level                                       \
count                            22616.000000
mean                              -500.521502
std                                 78.956292
min                               -800.258209
25%                               -500.167897
50%                               -499.951980
75%                               -499.492354
max                               -120.190931

      secondary_cleaner.state.floatbank6_a
_air  \
count                        22597.000000
mean                            17.976810
std                              6.636203
min                             -0.079426
25%                             13.968418
50%                             18.004215
75%                             23.009704
max                             54.876806
      secondary_cleaner.state.floatbank6_a
_level
count                            22615.000000
mean                              -519.361465
std                                 75.477151
min                               -810.473526
25%                               -500.981671
50%                               -500.095463
75%                               -499.526388
max                                -29.093593

[
```

Out[9]:

date	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concentrate_sol	final.output.concentrate_au	final.output.recovery
2017-01-03 21:59:59	5.823118	6.695543	13.384066	46.296351	64.456032
2017-02-23 16:59:59	7.034332	10.241551	15.757617	42.534733	76.218580
2018-04-28 15:59:59	6.220393	11.318218	7.748058	42.778664	67.219417
2017-07-17 19:59:59	4.807913	9.806378	12.527713	45.180306	61.390046
2017-09-29 07:59:59	4.433671	12.400607	6.570356	45.121775	72.276698

5 rows × 86 columns

> Coclusion: What we see from the previous step that we have in each variable missing values.Let's check if all alright with our Target variable Recovery by recalculating those values and check how they match with our recovery column in data set

Step 1.2. Checking that recovery is calculated correctly. Using the training set, calculate recovery for the rougher.output.recovery

RECOVERY = ((C *(F - T)) / (F* (C - T))) * 100%

- C — share of gold in the concentrate right after flotation
- F — share of gold in the feed before flotation
- T — share of gold in the rougher tails right after flotation

Here we need to calculate with our primary data set where we didn't clean our missing values it will provide to use better accuracy

In [10]:
```
def recovery(c,f,t):
    value = ((c * (f - t)) / (f * (c - t)))
    recovery = value * 100
    recovery[recovery < 0] = np.nan
    recovery[recovery > 100] = np.nan
    return recovery


c = train_gold['rougher.output.concentrate_au']
f = train_gold['rougher.input.feed_au']
t = train_gold['rougher.output.tail_au']

recovery_val = recovery(c,f,t)
#And then I propose to reffill missing values of each of our variable by "0"
MAE_val = mae(train_gold['rougher.output.recovery'].fillna(0),recovery_val.fillna(0))
print('Our MAE number between recovery calculations and the feature values is equal to: {:.3}'.format(MAE_val))
```

```
Our MAE number between recovery calculations and the feature values is equal to: 7.88e-15
```

> Conclusion: MAE of calcualted and provided data is very very close to zero. That means, we don't need to worry about errors in our data, because the provided data was correctly done!

Step 1.3 Analyzing the features not available in the test set.

In [11]:
```
#Creating a list with names of all features from our train data set
train_name = train_gold.columns.tolist()
#The same operation as we did prevous but for the test data set
test_name = test_gold.columns.tolist()
f
#Looping from all features in the test data set and comparing them with features
in the train data set features_not_test = []
for i in train_name:
    if i not in test_name:
        features_not_test.append(i)
print('Our features which no in Test set are:')
features_not_test
```

```
Our features which no in Test set are:
```

Out[11]:
```
['final.output.concentrate_ag',
 'final.output.concentrate_pb',
 'final.output.concentrate_sol',
 'final.output.concentrate_au',
 'final.output.recovery',
 'final.output.tail_ag',
 'final.output.tail_pb',
 'final.output.tail_sol',
 'final.output.tail_au',
 'primary_cleaner.output.concentrate_ag',
 'primary_cleaner.output.concentrate_pb',
 'primary_cleaner.output.concentrate_sol',
 'primary_cleaner.output.concentrate_au',
 'primary_cleaner.output.tail_ag',
 'primary_cleaner.output.tail_pb',
 'primary_cleaner.output.tail_sol',
 'primary_cleaner.output.tail_au',
 'rougher.calculation.sulfate_to_au_concentrate',
 'rougher.calculation.floatbank10_sulfate_to_au_feed',
 'rougher.calculation.floatbank11_sulfate_to_au_feed',
 'rougher.calculation.au_pb_ratio',
 'rougher.output.concentrate_ag',
 'rougher.output.concentrate_pb',
 'rougher.output.concentrate_sol',
 'rougher.output.concentrate_au',
 'rougher.output.recovery',
 'rougher.output.tail_ag',
 'rougher.output.tail_pb',
 'rougher.output.tail_sol',
 'rougher.output.tail_au',
 'secondary_cleaner.output.tail_ag',
 'secondary_cleaner.output.tail_pb',
 'secondary_cleaner.output.tail_sol',
 'secondary_cleaner.output.tail_au']
```

> Conclusion: We see that in our test_gold data set we don't have all output features that we have in the train set. They were intentionally removed. Because these columns have values that will cause data leakage for our future models.

Step 1.4. Perform data preprocessing

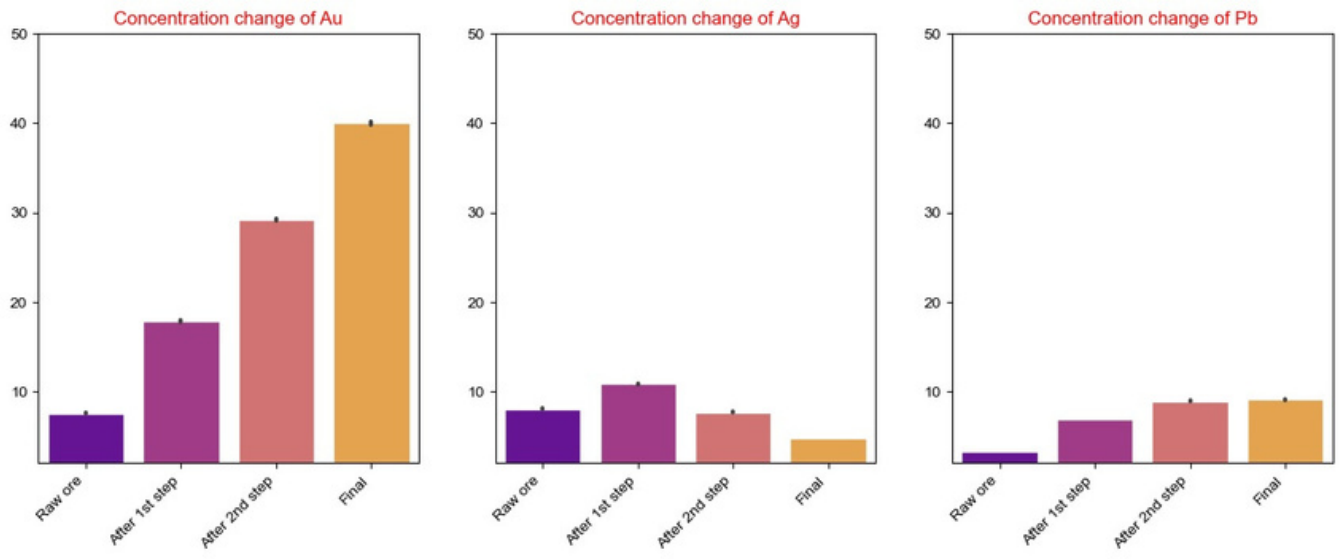
In [12]:
```
#prepare imputer for filling missing values
imputer =SimpleImputer()


#fit and fill missing values of training dataset using imputer
train_gold_upd = pd.DataFrame(imputer.fit_transform(train_gold))

#imputer removes column names and index, get them back
train_gold_upd.columns = train_gold.columns
train_gold_upd.index = train_gold.index


#use fitted imputer to fill missing values of source dataset
gold_upd = pd.DataFrame(imputer.transform(gold))
gold_upd.columns = gold.columns
gold_upd.index = gold.index

#get filled values for testing file from source
test_gold_upd = gold_upd.loc[test_gold.index,test_gold.columns]
```

> Conclusion: In the last steps, we filled using mean values of the training dataset. After all, the steps were done, we can sa y that we cleared and prepared our data sets for the next moves and can start working with them.

Step 2 Analyze the data

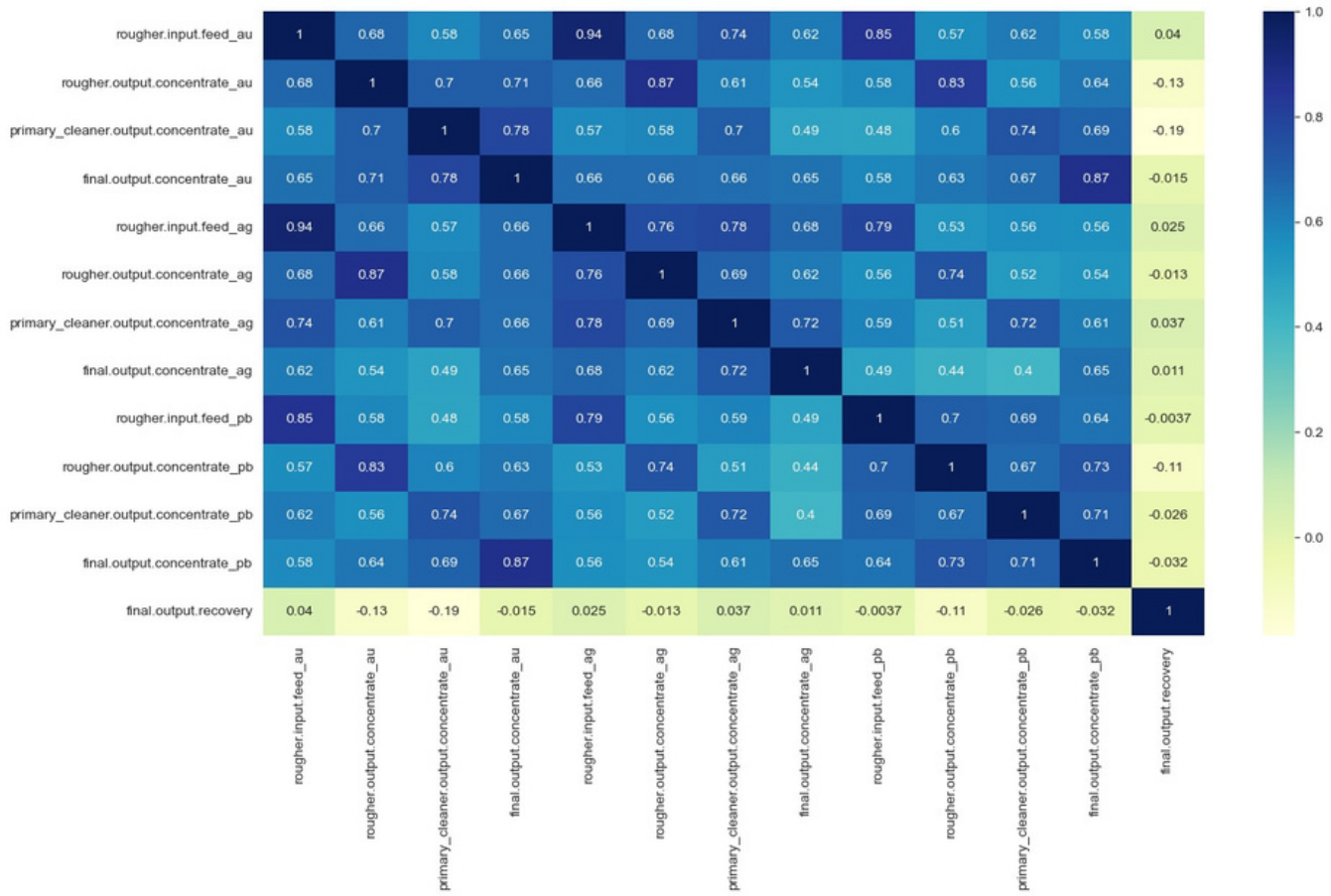
Step 2.1. Take note of how the concentrations of metals (Au, Ag, Pb) change depending on the purification stage.

In [13]:
```
#columns showing each metals concentration at different stages

au_conc = ['rougher.input.feed_au','rougher.output.concentrate_au',
       'primary_cleaner.output.concentrate_au','final.output.concentrate_au']
ag_conc = ['rougher.input.feed_ag','rougher.output.concentrate_ag',
       'primary_cleaner.output.concentrate_ag','final.output.concentrate_ag']
pb_conc = ['rougher.input.feed_pb','rougher.output.concentrate_pb',
       'primary_cleaner.output.concentrate_pb','final.output.concentrate_pb']

#using a for loop, prepare bar charts for the change of concentration of every metal at each stage
metals=[au_conc, ag_conc, pb_conc]
titles=['Concentration change of Au', 'Concentration change of Ag', 'Concentration change of Pb']

fig, ax = plt.subplots(1,3, figsize=(15, 5))
for i in range(3):
   sns.set_style('darkgrid')
   sns.barplot(data=gold_upd[metals[i]], ax=ax[i], palette='plasma')
   ax[i].set_xticklabels(labels=['Raw ore', 'After 1st step', 'After 2nd step', 'Final'], rotation=45, ha='righ
   ax[i].set_ylim(2, 50)
   ax[i].set_title(titles[i],color='r')
```



In [14]:
```
#Let's show what type of corraletion we have bettwen our  variables (Au, Ag, Pb) and final recovery output
corr_data = gold_upd[['rougher.input.feed_au','rougher.output.concentrate_au',
       'primary_cleaner.output.concentrate_au','final.output.concentrate_au','rougher.input.feed_ag','rougher.
       'primary_cleaner.output.concentrate_ag','final.output.concentrate_ag','rougher.input.feed_pb','rougher.
       'primary_cleaner.output.concentrate_pb','final.output.concentrate_pb','final.output.recovery']]
```

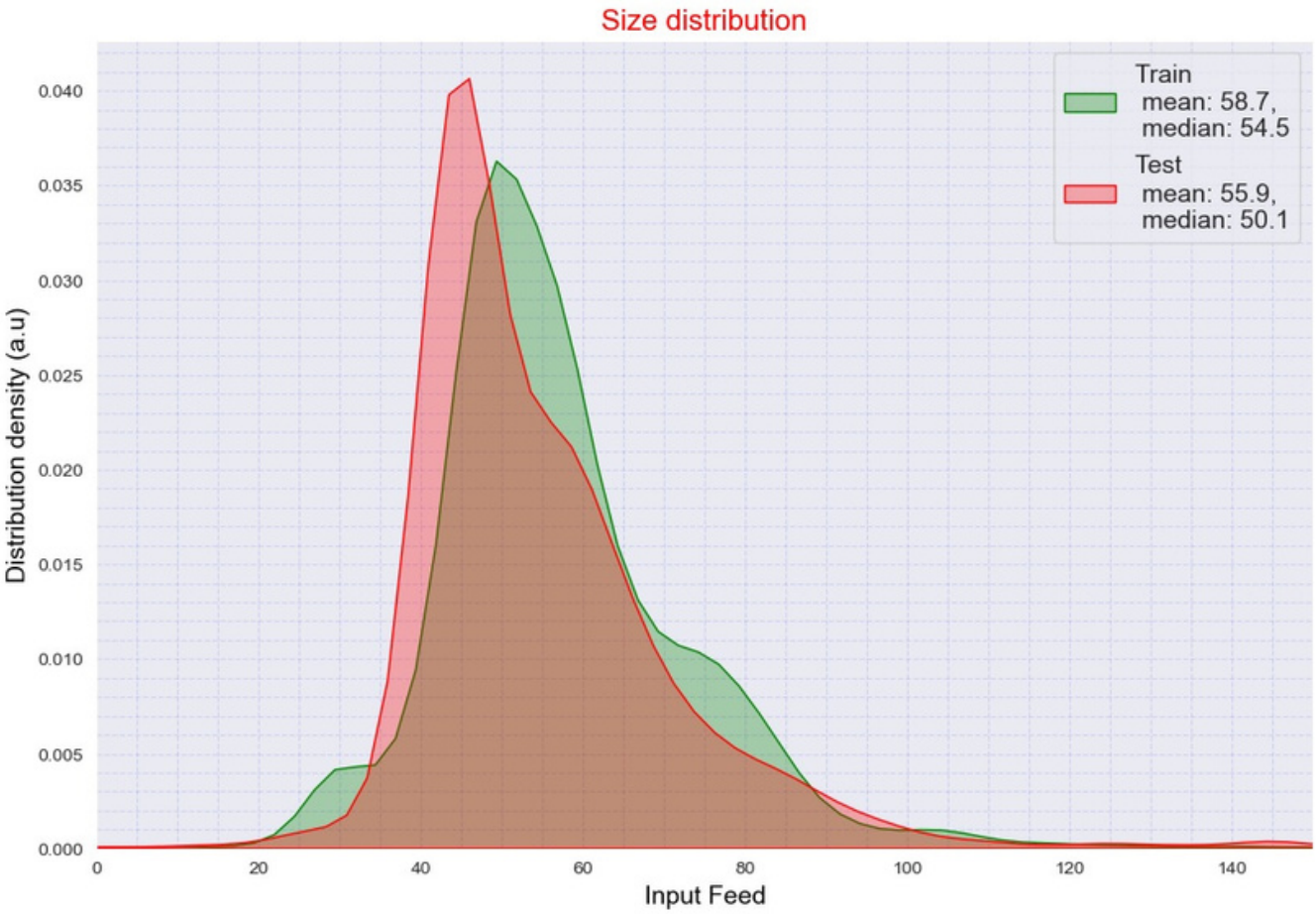
```
plt.figure(figsize=(15,8))
sns.heatmap(corr_data.corr(),annot=True, cmap='YlGnBu');
```



Step 2.2. 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 [15]:
```
#Creating a new variables which will represent our necessary distribution
dist_train = train_gold_upd['rougher.input.feed_size']
dist_test = test_gold_upd['rougher.input.feed_size']
```

In [16]:
```
#Now let's show our distribution and by
#create figure object
plt.figure(figsize=(12,8))

#plot distribution curves for each data sets conditions
sns.distplot(dist_train, hist=False, kde_kws={'shade':True, 'color':'green', 'alpha':0.3}, label=('Train \n mean
sns.distplot(dist_test, hist=False, kde_kws={'shade':True, 'color':'red', 'alpha':0.3}, label=('Test \n mean: {:

#customize the figure
plt.title('Size distribution', fontsize=16, color='r')
plt.xlabel('Input Feed', fontsize=14, color='k')
plt.ylabel('Distribution density (a.u)', fontsize=14, color='k')
plt.xlim(0, 150) #there are some extreme values, we don't need to show them for now
plt.legend(fontsize=14)
plt.grid(b=True, which='both', color='b', linestyle='--', alpha=0.1)
plt.minorticks_on()
plt.show()
```



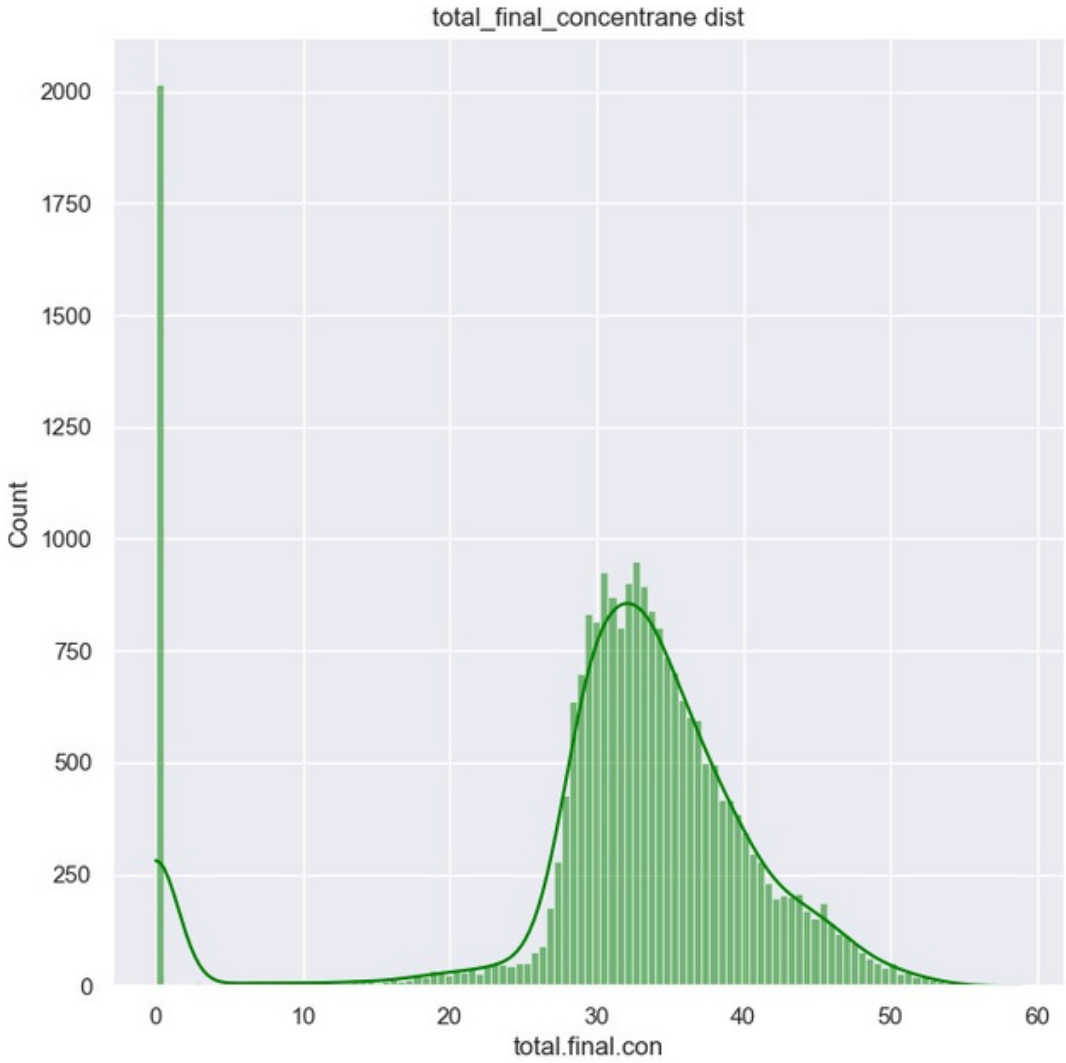
> Conclusion: From our density distribution graph, we see that we have mean particle size of training data 58.7 and the mean particle size of testing data was 55.9. If we compare those mean values we can say that they are a little bit different, but general distribution of particle sizes are similar for two datasets.

Step 2.3.

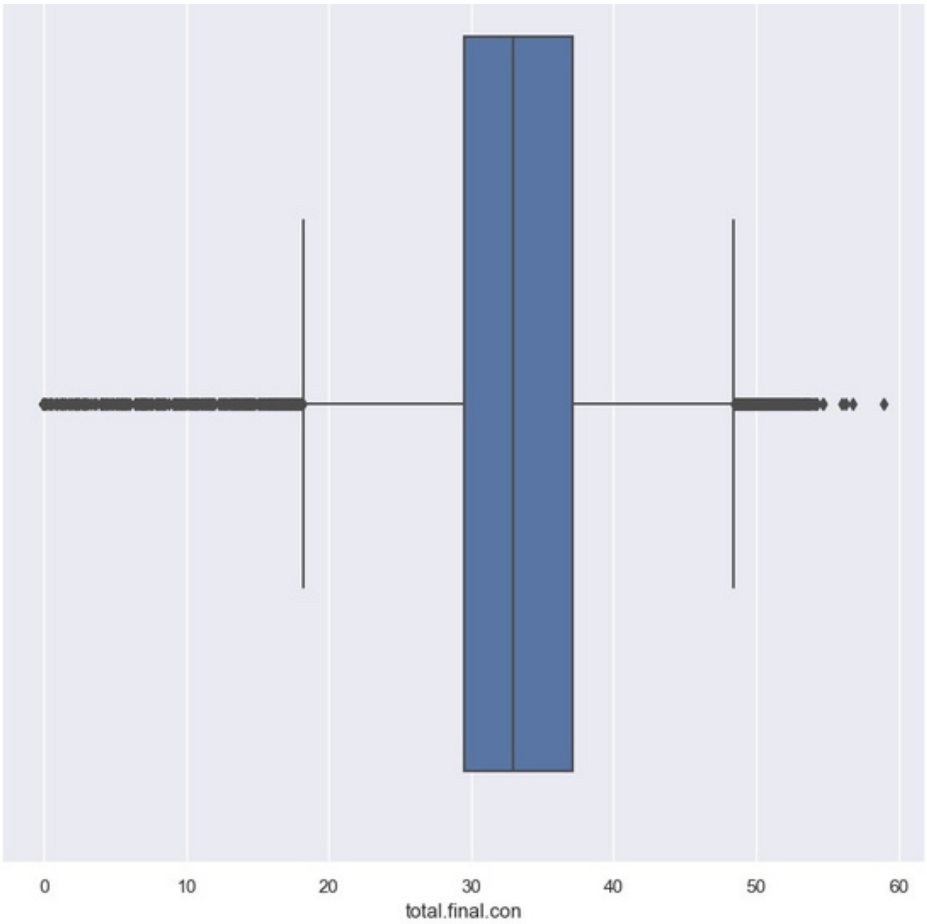
Consider the total concentrations of all substances at different stages: raw feed, rougher concentrate, and final concentrate.

In [17]:
```
#Creating our variables

gold_upd['total.raw.feed'] = gold_upd[['rougher.input.feed_ag',
                    'rougher.input.feed_pb',
                    'rougher.input.feed_au',
                    'rougher.output.tail_sol']].sum(axis=1)
gold_upd['total.roug.con'] = gold_upd[['rougher.output.concentrate_ag',
                    'rougher.output.concentrate_pb',
                    'rougher.output.concentrate_sol',
                    'rougher.output.concentrate_au']].sum(axis=1)
gold_upd['total.final.con'] = gold_upd[['final.output.concentrate_ag',
                          'final.output.concentrate_pb',
                          'final.output.concentrate_sol',
                          'final.output.concentrate_sol']].sum(axis=1)
```

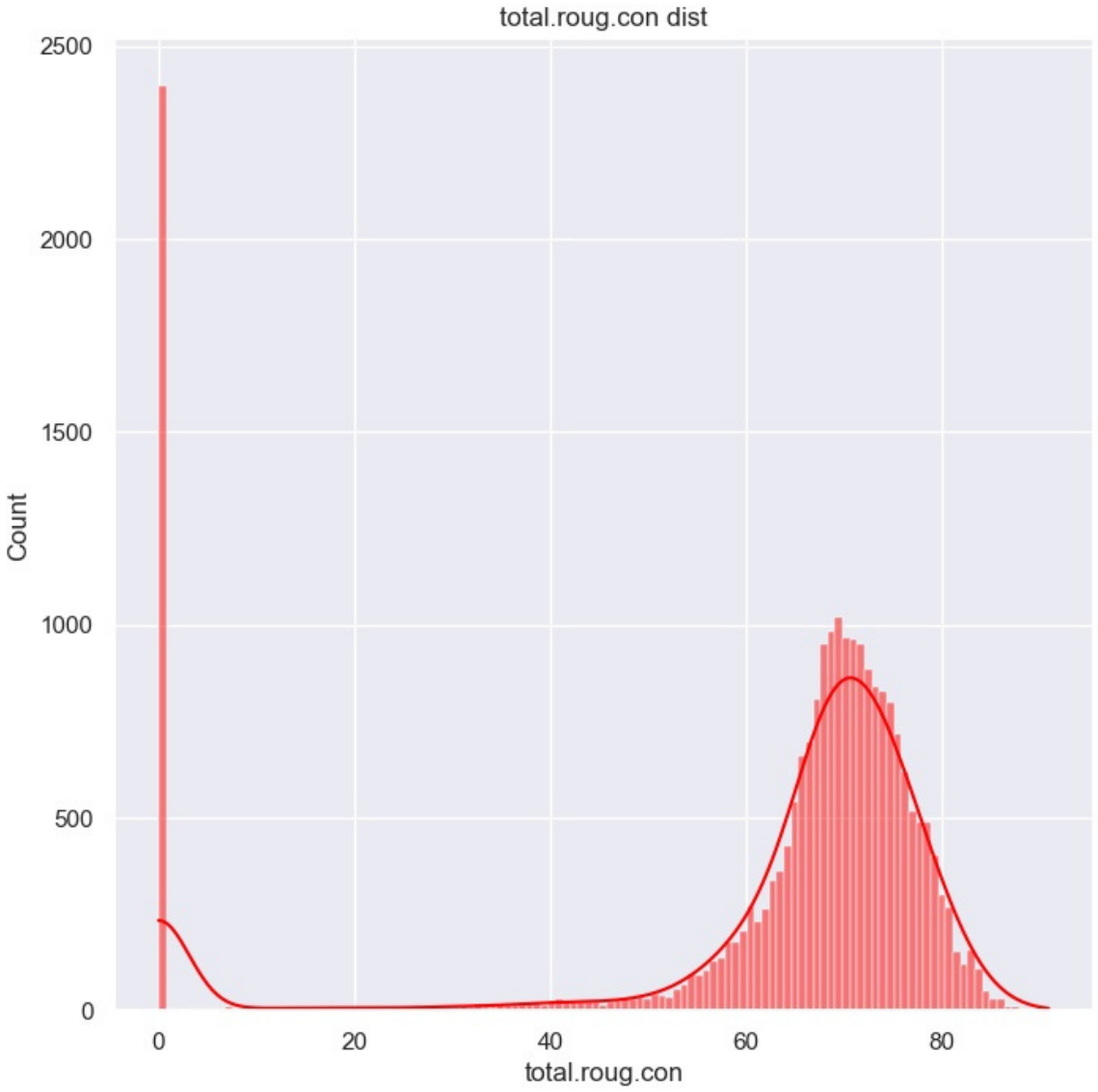
In [18]:
```
#Plotting first variable distribution


sns.set(style='darkgrid',font_scale=1.25)
sns.set(rc={'figure.figsize':(8,8)})
sns.histplot(data=gold_upd['total.final.con'],color='green',kde=True)
plt.title('total_final_concentrane dist');
```



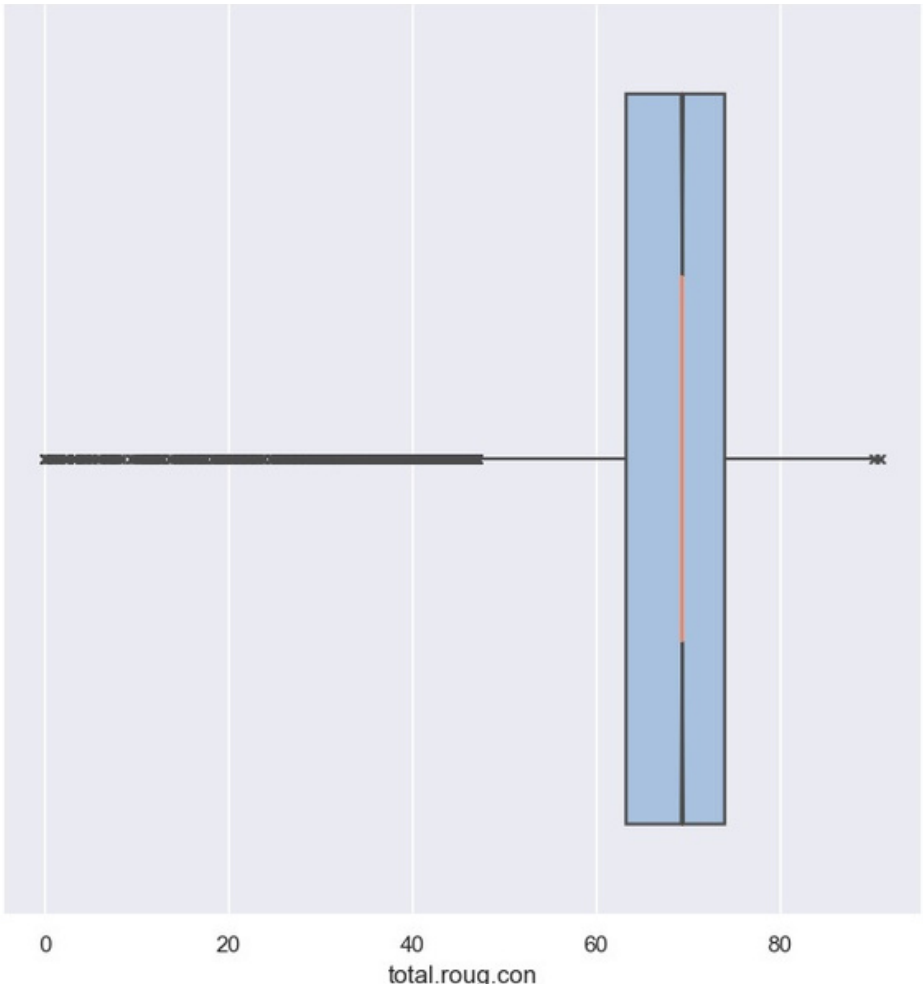
In [19]:
```
#Providing deeper analyzing by boxplot
sns.set(rc={'figure.figsize':(10,10)})
sns.boxplot(gold_upd['total.final.con']);
```



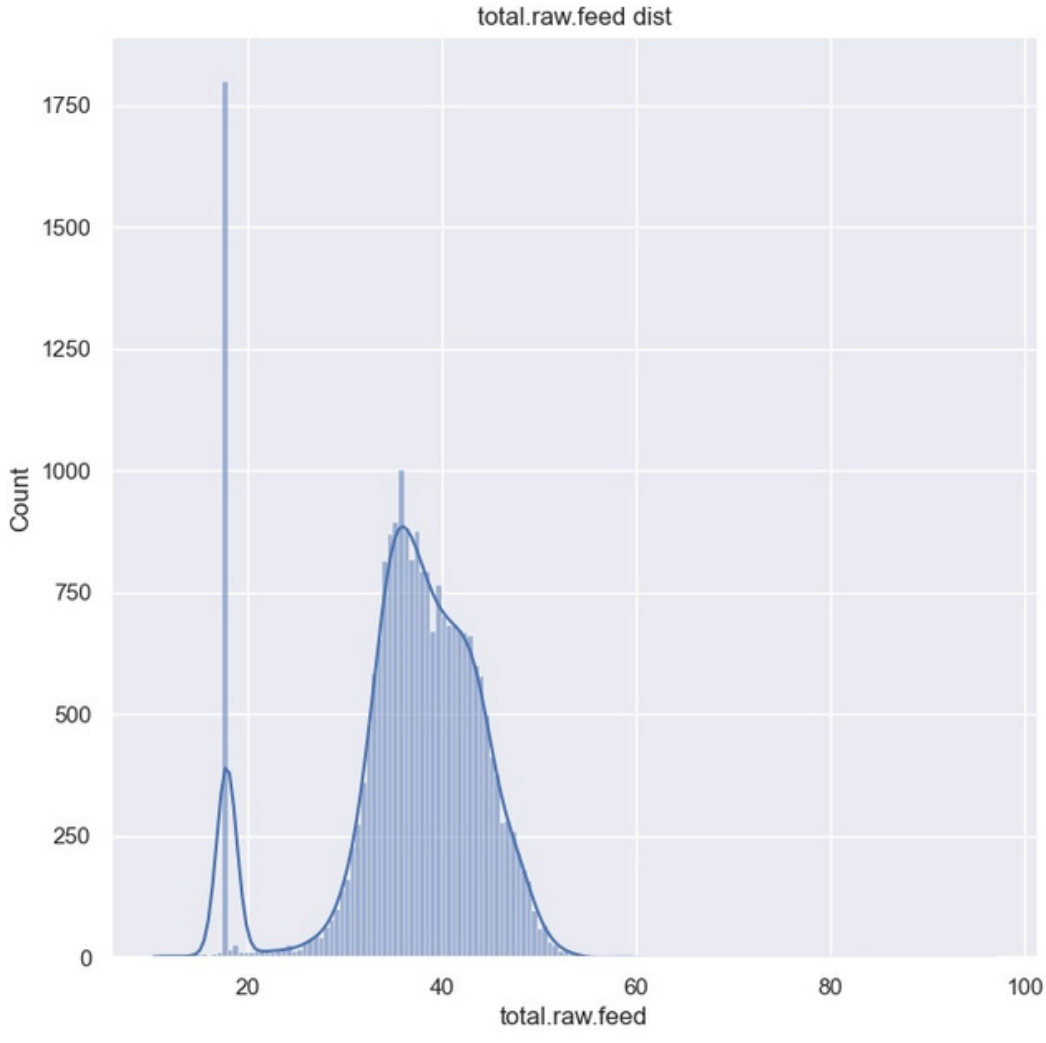
In [20]:
```
#Second variable distribution
sns.set(rc={'figure.figsize':(8,8)})
sns.histplot(gold_upd['total.roug.con'],color='red',kde=True)
plt.title('total.roug.con dist');
```



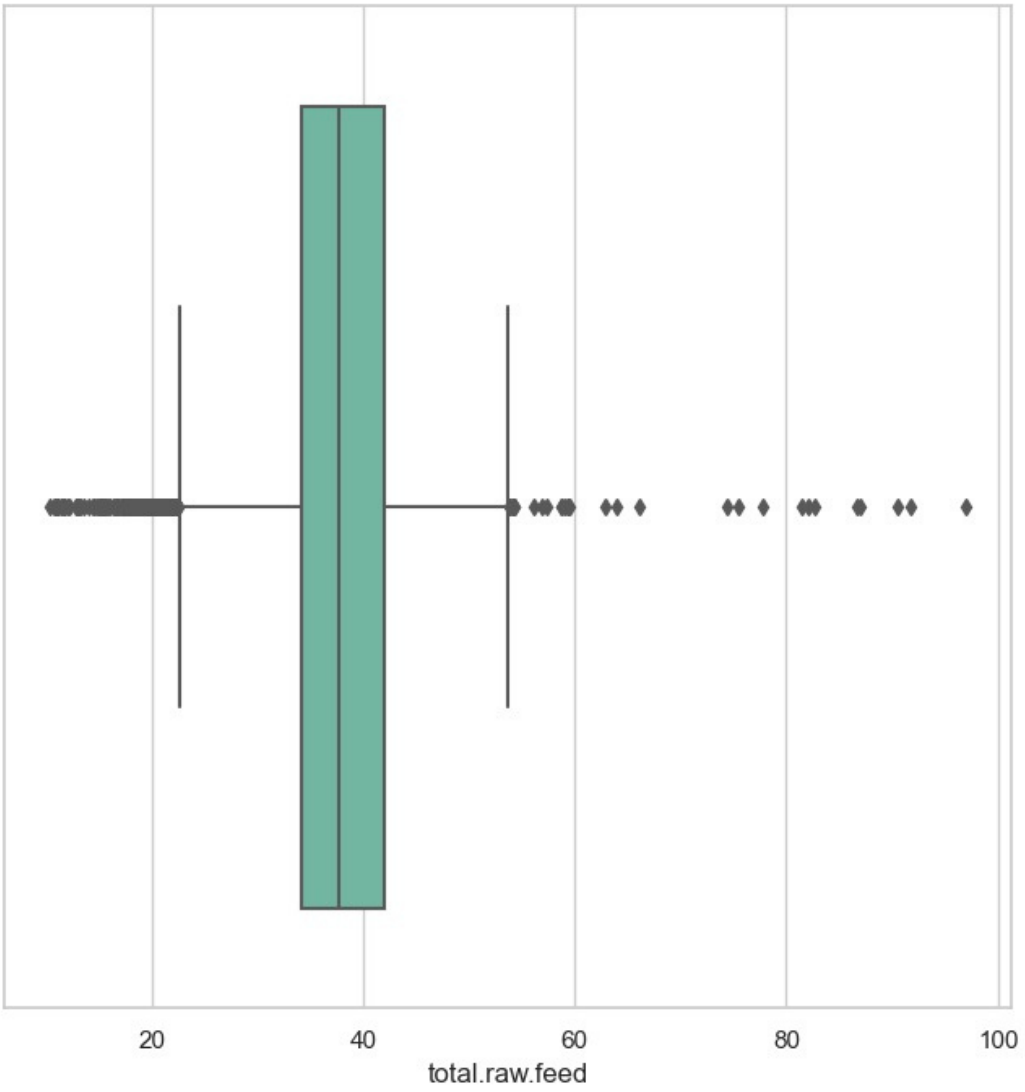
In [21]:
```
sns.boxplot(
    gold_upd['total.roug.con'],
    notch=True, showcaps=False,
    flierprops={"marker": "x"},
    boxprops={"facecolor": (.4, .6, .8, .5)},
    medianprops={"color": "coral"},
);
```



In [22]:
```
#And the last one variable
sns.set(rc={'figure.figsize':(8,8)})
sns.histplot(gold_upd['total.raw.feed'],kde=True)
plt.title('total.raw.feed dist');
```



In [23]:
```
sns.set(style="whitegrid")
sns.boxplot(gold_upd['total.raw.feed'],
            palette="Set2",
            dodge=True);
```



> Conclusion: From our histograms, we can see that we have a lot of values equal to 0 and outliers which can bring us wrong misleading and our model evaluation will be incorrect. That is why I propose to remove all anomaly values that are equal to 0.

In [24]:
```
#Let's remove all an anomaly values that we will provide better work for our model
anomaly_values = gold_upd[(gold_upd['total.raw.feed'] == 0) |\
          (gold_upd['total.roug.con'] == 0) |\
           (gold_upd['total.final.con'] == 0)].index


test_rows_drop = []
train_rows_drop = []
for i in anomaly_values:
   if i in test_gold_upd.index:
      test_rows_drop.append(i)
   elif i in train_gold_upd.index:
      train_rows_drop.append(i)

print('We dropped around {:.2%} of our train data'.format(len(train_rows_drop)/len(train
_gold_upd))
)
print('We dropped around {:.2%} of our test data'.format(len(test_rows_drop)/len(test_g
old_upd))
)
```

```
We dropped around 9.76% of our train data
We dropped around 9.29% of our test data
```

In [25]:
```
# drop all rows with total conc equal to 0
train_gold_upd.drop(train_rows_drop, inplace=True)
test_gold_upd.drop(test_rows_drop, inplace=True)
```

Step 3. Build the model

Step 3.1. Write a function to calculate the final sMAPE value

In [26]:
```
#function to calcualte sMAPE score for predicted and actual values
def calc_metric(y, y_pred):
   smape = 1/len(y) *np.sum(np.abs(y - y_pred)*2*100/((np.abs(y) + np.abs(y_pred))))
   return smape
```

```
#Calculation final_sMAPE
def calc_metric_final(rough,final):
   final_sMAPE = 0.25 * rough + 0.75 * final
   return final_sMAPE

#convert it to sklearn function so it can be directly attached to CV scoring parameter
my_scorer=make_scorer(calc_metric, greater_is_better=False)
```

Step3.2. Train, different models. Evaluate them using cross-validation. Pick the best model and test it using the test sample. Provide findings.

In this step, I propose to work with linear regression, Decision Tree Regression, and Random Forest Regression models. But first, we will need to standardize our features because we see that our features have different scales, that means if one feature will lies in the range from 0 to 10 and another from 0 to 100 then the algorithm will find the features with the biggest range to be more important.

In [27]:
```
#Assigned our variables
features_final = test_gold_upd.columns.tolist()
features_rougher= [x for x in features_final if 'rougher' in x.split('.')]

print(len(features_final))
print()
print(len(features_rougher))
```

```
52

22
```

-

In [28]:
```
#Our rougher Train features
X_train_rog = train_gold_upd[features_rougher].reset_index(drop=True)
Y_train_rog = train_gold_upd['rougher.output.recovery'].reset_index(drop=True)

#Our rougher Test Features
X_test_rog = gold_upd.loc[test_gold_upd.index,features_rougher].reset_index(dr
op=True)
Y_test_rog = gold_upd.loc[test_gold_upd.index,'rougher.output.recovery'].reset_i
ndex(drop=True)
```

In [29]:
```
#Our Final Train features
X_train_final = train_gold_upd[features_final].reset_index(drop=True)
Y_train_final = train_gold_upd['final.output.recovery'].reset_index(drop=True)
#Our Final Test features
X_test_final = gold_upd.loc[test_gold_upd.index,features_final].reset_index(drop
=True)
Y_test_final = gold_upd.loc[test_gold_upd.index,'final.output.recovery'].reset_in
dex(drop=True)

print(X_train_rog.shape)
print(Y_train_rog.shape)
print(X_test_rog.shape)
print(Y_test_rog.shape)
print(X_train_final.shape)
print(Y_train_final.shape)
print(X_test_final.shape)
print(Y_test_final.shape)
```

```
(15215, 22)
(15215,)
(5312, 22)
(5312,)
(15215, 52)
(15215,)
(5312, 52)
(5312,)
```

- Now we can start work with our models

In [30]
```
#construct a pipeline
pipeline = Pipeline([('scaler',StandardScaler()), ('model', LinearRegression())])

#parameters need to be checkec here we used only three different algorithms
params = [{'model': [RandomForestRegressor(n_estimators = 200, max_depth=10, random_state=12
       {'model': [LinearRegression()]},
       {'model':[DecisionTreeRegressor(max_depth=20, random_state=12345)]}]

#construct a grid to search best algorithm
grid_r = GridSearchCV(pipeline, param_grid = params, cv=5, n_jobs = -1, scoring = my_scorer)

#train each model using 5 cross_validation steps
grid_r.fit(X_train_rog, Y_train_rog)


#display results by ranking best to worst
results_r = pd.DataFrame(grid_r.cv_results_)
```

```
results_r = results_r.sort_values('rank_test_score')
 results_r
```

Out[30]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_model	params
1	0.024199	0.006676	0.003800	0.000749	LinearRegression()	{'model': LinearRegression()}
0	34.544354	5.001904	0.049600	0.005609	RandomForestRegressor(max_depth=10, n_estimato...	{'model': RandomForestRegressor(max_depth=10, ...
2	0.457432	0.023591	0.003601	0.001020	DecisionTreeRegressor(max_depth=20, random_sta...	{'model': DecisionTreeRegressor(max_depth=20, ...

In [31]:

```
# The same operations as previous but for the final set
#train each model using 5 cross_validation steps
#construct a grid to search best algorithm
grid_f = GridSearchCV(pipeline, param_grid = params, cv=5, n_jobs = -1, scoring = my_scorer)
grid_f.fit(X_train_final, Y_train_final)


#display results by ranking best to worst
results_f = pd.DataFrame(grid_r.cv_results_)
results_f = results_r.sort_values('rank_test_score')
results_f
```

Out[31]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_model	params
1	0.024199	0.006676	0.003800	0.000749	LinearRegression()	{'model': LinearRegression()}
0	34.544354	5.001904	0.049600	0.005609	RandomForestRegressor(max_depth=10, n_estimato...	{'model': RandomForestRegressor(max_depth=10, ...
2	0.457432	0.023591	0.003601	0.001020	DecisionTreeRegressor(max_depth=20, random_sta...	{'model': DecisionTreeRegressor(max_depth=20, ...

- After we have done all calculations can say that our best model is "LinearRegression". Let's start working with our model LinearRegression

Step 3.3. Final model training and evaluation using test set

In [35]:

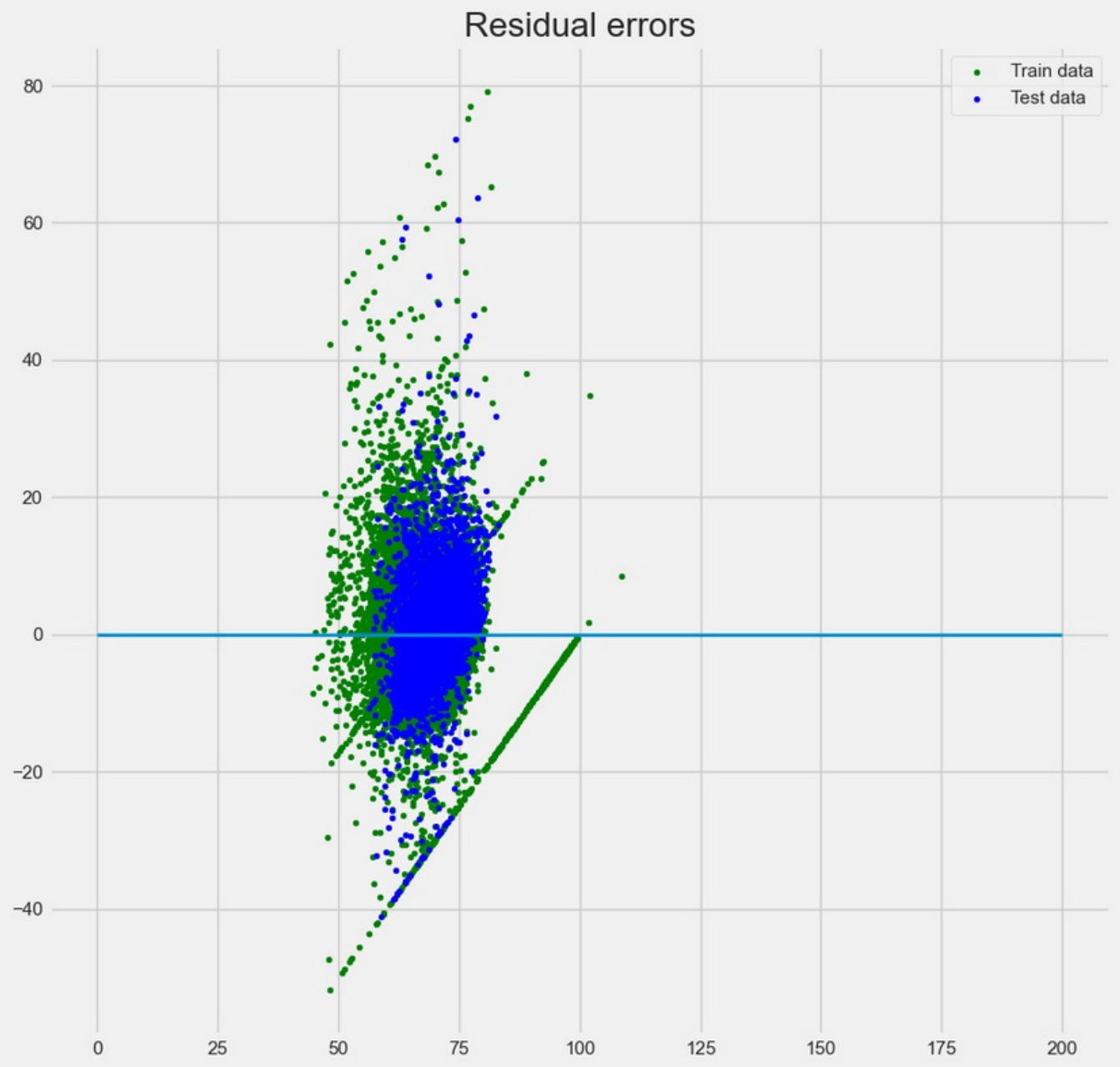
```
#Starting working with our Linear Regression algorithm for rougher data sets model = LinearRegression()

#Finding predictions values for the final set
model_f = model.fit(X_train_final,Y_train_final)
predictions_final = pd.Series(model_f.predict(X_test_final), index=Y_test_final.index)


#Finding predictions values for rough set

model_r = model.fit(X_train_rog,Y_train_rog)

predictions_rog = pd.Series(model_r.predict(X_test_rog), index=Y_test_rog.index)

model.fit(X_train_final,Y_train_final)
```

In [42]:

```

# regression coefficients
print('Coefficients: ', model_f.coef_)
print()
# variance score: 1 means perfect prediction
print('Variance score: {}'.format(model_f.score(X_test_final, Y_test_final)))

# plot for residual error
plt.figure(figsize=(10,10))
# setting plot style
plt.style.use('fivethirtyeight')

# plotting residual errors in training data
plt.scatter(model_f.predict(X_train_final),
            model_f.predict(X_train_final) - Y_train_final,
            color="green", s=10,
            label='Train data')

# plotting residual errors in test data
plt.scatter(model_f.predict(X_test_final),
            model_f.predict(X_test_final) - Y_test_final,
            color="blue", s=10,
            label='Test data')
```

```
# plotting line for zero residual error
plt.hlines(y=0, xmin=0, xmax=200, linewidth=2)

# plotting legend
plt.legend(loc='upper right')

# plot title
plt.title("Residual errors")

# method call for showing the plot
plt.show()
```

```
Coefficients:  [ 9.69936524e-02 -1.54442518e-01  9.87463619e-01 -7.05883348e-01
  4.91692772e-03 -7.59159249e-04  8.79266651e-03 -8.38356299e-03
 -1.29176980e-02  1.00306042e-02 -3.04229535e-03 -6.81796525e-03
 -5.85822922e-01 -2.08038768e+00 -9.22659821e-03 -9.59933910e-03
 -1.14461229e-01  1.13957557e+00 -1.69277556e-01 -8.17634341e-01
 -8.49373023e-02  2.17170157e+00  2.13723371e-03 -3.13676173e-03
  6.27377121e-03  1.56628687e-02 -6.52322256e-03  9.13793870e-03
 -8.64303483e-03  6.94836895e-03 -1.12977205e-02 -2.67609580e-02
  1.58168460e-02  6.32074035e-03  2.04874345e-01 -9.43847232e-04
 -9.55510968e-02 -1.52380103e-02  9.16273380e-02 -1.05766977e-02
 -1.64535705e-01  6.75963954e-03  2.96794364e-01  9.09686277e-03
  1.01430531e-01 -9.04360224e-03  6.11299898e-03 -1.02227466e-02
 -1.45746049e-01  3.44488368e-04 -2.95359074e-02 -9.34274565e-03]

Variance score: -0.053058204650828555
```



In [40]:
```
#Calculating our Final sMAPE

def calc_metric_final(rough,final):
    final_sMAPE = 0.25 * rough + 0.75 * final
    return final_sMAPE

rough = calc_metric(Y_test_rog,predictions_rog )
```

```
final = calc_metric(Y_test_final, predictions_final)

print('sMAPE score for rough set: {:.3}'.format(rough))
print('sMAPE score for final set: {:.3}'.format(final))
print('the final sMAPE score is {:.3}'.format(calc_metric_final(rough,final)))
```

```
sMAPE score for rough set: 4.72
sMAPE score for final set: 7.94
the final sMAPE score is 7.13
```

Conclusion:

- 3.1 Three different algorithms were tested (LinearRegression, RandomForestRegressor, and DecisionTreeRegressor) using the cross_validation technique. After we made the calculation we found which of our models predict with the highest rate of accuracy or with the smallest number of MAE
- 3.2 LinearRegression were chosen as the best model with its sMAPE score for rougher and final output recoveries was equal to 4.72 and 7.94. respectively.
- 3.3 The LinearRegression model as described above was trained with the whole training dataset and tested with the testing dataset. Ultimately, the final sMAPE value is equal to 7.13!

Overall Conclusion:

- 1.1 Three data files have been successfully loaded. The source file (gold) has 22716 rows and 86 columns. It was observed that the testing file (test_gold) was obtained from two different time periods of the source file and test_gold has 5856 rows, 52 columns. The rest of the source file was used for the preparation of the training file (train_gold) with 16860 rows, 86 columns.
- 1.2 There were missing values in all data files, these missing values were filled using mean values of training dataset.
- 1.3 Several columns were not present in the testing file, however, when we check these columns, we found that they were intentionally removed. Because these columns have values that will cause data leakage for our future models.
- 2.1 It was observed that the concentration of gold (Au) increased at each stage of purification. In the final output stage, the concentration of gold reached around 40. Lead (Pb) also follows the same trend, however, the final concentration is just around 9. But, for the silver (Ag), the concentration increased first and then dropped, where at the final stage its concentration was around 5.
- 2.2 Particle size distribution of raw feed of training and testing data sets compared by visualizing density plots. It was observed that they have a similar distribution, even though their mean and median values differ slightly.
- 2.3 Finally we have checked the total concentration of substances at each stage and visualized their distribution. Surprisingly, it was found that there were a lot of rows with a total concentration equal to zero. For the sake of not putting more biases into our data, we decided to drop all rows (in training and testing datasets) when the total concentration is equal to zero. Consequently, above 9% of rows were dropped from training and testing datasets.
- 3.1 Three different algorithms were tested (LinearRegression, RandomForestRegressor, and DecisionTreeRegressor) using the cross_validation technique.
- 3.2 LinearRegression were chosen as the best model with its sMAPE score for rougher and final output recoveries was equal to 4.72 and 7.94 respectively.
- 3.3 Ultimately, the final sMAPE value is equal to 7.13!
-

In [ ]: