Содержание ¶

Восстановление золота из руды (учебный проект)

Прототип модели машинного обучения для «Цифры».
Модель предсказывает коэффициент восстановления золота из золотосодержащей руды.

Этапы:
rougher — флотация
primary_cleaner — первичная очистка
secondary_cleaner — вторичная очистка
final — финальные характеристики

Типы параметров:
input — параметры сырья
output — параметры продукта
state — параметры, характеризующие текущее состояние этапа
calculation — расчётные характеристики

Технологический процесс
Rougher feed — исходное сырье
Rougher additions (или reagent additions) — флотационные реагенты: Xanthate, Sulphate, Depressant
Xanthate — ксантогенат (промотер, или активатор флотации);
Sulphate — сульфат (на данном производстве сульфид натрия);
Depressant — депрессант (силикат натрия).
Rougher process (англ. «грубый процесс») — флотация
Rougher tails — отвальные хвосты
Float banks — флотационная установка
Cleaner process — очистка
Rougher Au — черновой концентрат золота
Final Au — финальный концентрат золота

Параметры этапов
air amount — объём воздуха
fluid levels — уровень жидкости
feed size — размер гранул сырья
feed rate — скорость подачи

Подготовка данных

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import random

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_absolute_error, make_scorer
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

```

In [3]:

```
train = pd.read_csv('/datasets/gold_recovery_train_new.csv')
test = pd.read_csv('/datasets/gold_recovery_test_new.csv')
full = pd.read_csv('/datasets/gold_recovery_full_new.csv')

```

In [4]:

```
print(len(train.columns), len(test.columns), len(full.columns)) #длина выборок
```

```
87 53 87
```

In [5]:

```
train_c = train.columns.to_list()
full_c = full.columns.to_list()
test_c = test.columns.to_list()
if train_c== full_c:
    print('Столбцы обучающей и полной выборки равны') #столбцы обучающей и полно

```

```
Столбцы обучающей и полной выборки равны
```

In [6]:

```
deleted = [i for i in full_c if not i in test_c]
len(deleted) #признаки, отсутствующие в тестовой выборке
```

Out[6]:

```
34
```

In [7]:

```
nodeleted = [i for i in full_c if not i in deleted]
test_c==nodeleted
```

Out[7]:

```
True
```

Тестовая выборка содержит:

Дата

Первичная очистка:

- параметры сырья: сульфаты, депрессант, ксантогенат, размер гранул
- текущее состояние этапа на 8 флотационной установке: объем воздуха и уровень жидкости (4 измерения)

Флотация:

- входящий размер 4 элементов, размер гранул, скорость подачи
- параметры сырья на 10 и 11 флотационной установке (сульфат, ксантогенат)
- текущее состояние этапа на 10 флотационной установке: объем воздуха и уровень жидкости (6 измерений)

Вторичная очистка:

- текущее состояние этапа на 2, 3, 4, 5, 6 установках: объем воздуха и уровень жидкости (2 измерения)

Отсутствующие столбцы

Финальная:

- параметры продукта: концентрация золота, свинца, серебра, соли
- параметры продукта: эффективность восстановления
- параметры продукта: концентрация 4 элементов в хвостах

Первичная очистка:

- параметры продукта: параметры сырья: сульфаты, депрессант, ксантогенат, размер гранул сырья
- параметры продукта: концентрация 4 элементов после очистки
- параметры продукта: концентрация 4 элементов в хвостах после очистки

Флотация:

- отгношение сульфата к концентрации золота
- отношение сульфата к размеру гранул золота, флотационные установки 10 и 11
- соотношение золота и свинца
- параметры продукта: размер гранул серебра, свинца, соли, золота
- параметры продукта: концентрация 4 элементов
- параметры продукта: эффективность восстановления
- параметры продукта: концентрация 4 элементов в хвостах

Вторичная очистка:

- параметры продукта: концентрация 4 элементов в хвостах

Вывод:
В тестовой выборке отсутствуют столбцы, которые являются результатами работы этапа

Список недоступных признаков на тесте:
Финальный концентрат:
параметры продукта: концентрация золота, серебра, свинца, соли
параметры продукта: концентрация 4 элементов в хвостах
параметры продукта: эффективность обогащения

Первичная очистка:
параметры продукта: концентрация 4 элементов
параметры продукта: концентрация 4 элементов в хвостах

Флотация:
расчетные характеристики: концентрация сульфата по отношению к золоту
расчетные характеристики: флотационная установка 10, отношения сульфата к золоту
расчетные характеристики: флотационная установка 11, отношения сульфата к золоту
расчетные характеристики: флотационная установка 10, соотношение свинца
параметры продукта: концентрация 4 элементов
параметры продукта: концентрация 4 элементов в хвостах
параметры продукта: эффективность обогащения

Вторичная очистка:
параметры продукта: концентрация 4 элементов
параметры продукта: концентрация 4 элементов в хвостах

In [8]:

```
1 full.columns[~full.columns.isin(test.columns.to_list())]
```

Out[8]:

```
Index(['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_so
l',
       '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.t
ail_pb',
       'secondary_cleaner.output.tail_sol',
       'secondary_cleaner.output.tail_au'],
      dtype='object')
```

In [9]:

```
#Recovery = (c*(f-t))/(f*(c-t))*100%<br><br>

#C — доля золота в концентрате после флотации/очистки;<br>
#F — доля золота в сырье/концентрате до флотации/очистки;<br>
#T — доля золота в отвальных хвостах после флотации/очистки.m<br>

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

full['rougher.recovery_test'] = (c*(f-t))/(f*(c-t))*100

#print(full['rougher.recovery_test'][16])
#print(full['rougher.output.recovery'][16])
print(full[['rougher.recovery_test', 'rougher.output.recovery']])


```

```
       rougher.recovery_test  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
...                      ...                      ...
19434              89.574376                89.574376
19435              87.724007                87.724007
19436              88.890579                88.890579
19437              89.858126                89.858126
19438              89.514960                89.514960

[19439 rows x 2 columns]
```

In [10]:

```
c = full['final.output.concentrate_au']
t = full['final.output.tail_au']
f = full['primary_cleaner.output.concentrate_au']

full['final.recovery_test'] = (c*(f-t))/(f*(c-t))*100

print(full[['final.recovery_test', 'final.output.recovery']].sort_values('final.
```

```
       final.recovery_test  final.output.recovery
14746                 -0.0                    0.0
14359                 -0.0                    0.0
14360                 -0.0                    0.0
10026                 -0.0                    0.0
7585                  -0.0                    0.0
...                    ...                    ...
15787                100.0                  100.0
6279                 100.0                  100.0
6163                 100.0                  100.0
18037                100.0                  100.0
11977                100.0                  100.0

[19439 rows x 2 columns]
```

In [11]:

```
full[(full['final.recovery_test']<0)&(full['primary_cleaner.output.concentrate_a
```

Out[11]:

	primary_cleaner.output.concentrate_au	final.output.concentrate_au	final.output.tail_au	final.recovery_test	fi
19	0.0	42.509402	2.272460	-inf	
14308	0.0	41.347015	3.060066	-inf	
14113	0.0	35.303158	5.471755	-inf	
14112	0.0	43.186241	4.907862	-inf	
14111	0.0	43.963810	4.374566	-inf	
...	...	...	...	...	
8263	0.0	42.691757	3.655060	-inf	
8262	0.0	41.862333	4.064143	-inf	
8261	0.0	43.046942	4.322626	-inf	
7443	0.0	44.528214	3.199278	-inf	

In [12]:

```
#Найдите MAE между вашими расчётами и значением признака.
mae = mean_absolute_error(full['rougher.recovery_test'], full['rougher.output.re
mae
```

Out[12]:

```
9.874045668302637e-15
```

В 'rougher.output.recovery', расчитанном по формуле различия с исходными данными несущественные. Возможно, они возникли в следствии округления данных в расчетах.
'final.recovery_test' - значения отличаются сильно и также я заметила, что если на входе была концентрация 0, то формула дает результат -inf.
Замена данных формулой приводит к сильному ухудшению качества модели. Возможно, при расчетах формула отличалась?

In [13]:

```
full['rougher.output.recovery'] = full['rougher.recovery_test']
#full.loc[full['primary_cleaner.output.concentrate_au']!=0,'final.output.recover
```

In [14]:

```
#full[full['final.output.recovery']<0]['primary_cleaner.output.concentrate_au']
```

In [15]:

```
print(full.shape,test.shape,train.shape) #в фул я добавляла одну колонку для тес
```

```
(19439, 89) (5290, 53) (14149, 87)
```

Дубликатов нет

In [16]:

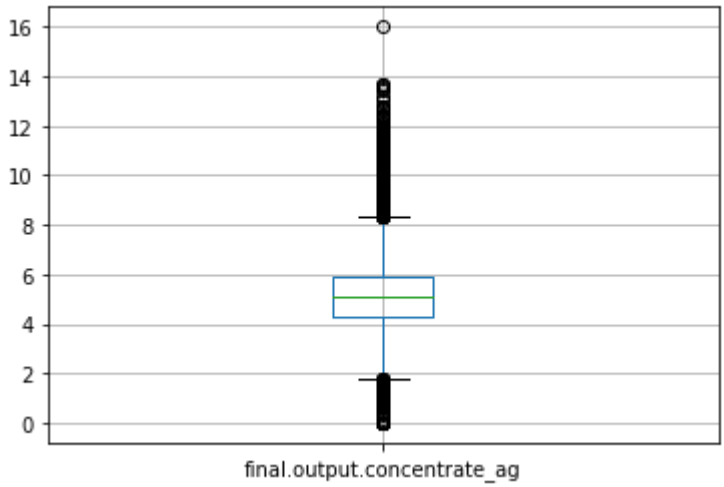
```
for one in [full,test,train]:
    print(one.duplicated().sum())
```

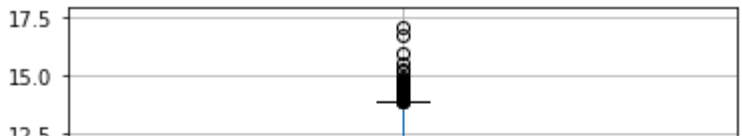
```
0
0
0
```

Выпады в данных есть, я не могу их удалить без консультации со специалистом

In [17]:

```
for one in full.columns.drop('date'):
    full[[one]].boxplot()
    plt.show()
```





Пропуски

параметры продукта заполню ближайшими значениями, параметры сырья - медианами

In [18]:

```
def passing(data):
    #продукт
    for elem in ['ag','sol','pb', 'au']:
        for output in['final.output.concentrate_',
                      'final.output.tail_',
                      'primary_cleaner.output.tail_',
                      'primary_cleaner.output.concentrate_']:
            if (output+elem) in data.columns:
                data.loc[data[output+elem].isna(), output+elem] = data[output+el

    if 'rougher.output.concentrate_sol' in data.columns:
        data.loc[data['rougher.output.concentrate_sol'].isna(),
                 'rougher.output.concentrate_sol'] = data['rougher.output.concer
    if 'rougher.output.tail_ag' in data.columns:
        data.loc[data['rougher.output.tail_ag'].isna(),
                 'rougher.output.tail_ag'] = data['rougher.output.tail_ag'].ffil

    for elem in ['ag','pb','sol']:
        if ('secondary_cleaner.output.tail_'+elem) in data.columns:
            data.loc[data['secondary_cleaner.output.tail_'+elem].isna(),
                     'secondary_cleaner.output.tail_'+elem] = data['secondary_cl
                                                                   +elem].ffill(
    #сырье
    for reagent in ['sulfate','depressant','xanthate']:
        if ('primary_cleaner.input.'+reagent) in data.columns:
                data.loc[data['primary_cleaner.input.'+reagent].isna(),
                         'primary_cleaner.input.'+reagent] = data['primary_clear
    for reagent in ['sulfate','xanthate']:
        for inpu in ['rougher.input.floatbank10_','rougher.input.floatbank11_']:
            if (inpu+reagent) in data.columns:
                data.loc[data[inpu+reagent].isna(),inpu+reagent] = data[inpu+rea
    for feed in ['feed_pb','feed_rate','feed_size','feed_sol']:
        if ('rougher.input.'+feed) in data.columns:
                data.loc[data['rougher.input.'+feed].isna(), 'rougher.input.'+fe


    #текущий этап
    for state in ['primary_cleaner.state.floatbank8_']:
        for letter in ['a','b','c','d']:
            for item in ['_air','_level']:
                if (state+letter+item) in data.columns:
                    data.loc[data[state+letter+item].isna(),
                             state+letter+item] = data[state+letter+item].ffill(

    for letter in ['a','b','c']:
        for item in ['_air','_level']:
            if ('rougher.state.floatbank10_'+letter+item) in data.columns:
                data.loc[data['rougher.state.floatbank10_'+letter+item].isna(),
                     'rougher.state.floatbank10_'+letter+item] = data['rougher.s
                                                                     +letter+it
    if 'rougher.state.floatbank10_e_air' in data.columns:
        data.loc[data['rougher.state.floatbank10_e_air'].isna(),
             'rougher.state.floatbank10_e_air'] = data['rougher.state.floatbank1

    sec = 'secondary_cleaner.state.floatbank'
    for fb in range(2,6):
        fb = str(fb)
        for letter in ['_a_','_b_']:
            for item in ['air','level']:
```

```
                if (sec+fb+letter+item) in data.columns:
                    data.loc[data[sec+fb+letter+item].isna(),
                             sec+fb+letter+item] = data[sec+fb+letter+item].ffil
    for item in ['air','level']:
        if ('secondary_cleaner.state.floatbank6_a_'+item) in data.columns:
            data.loc[data['secondary_cleaner.state.floatbank6_a_'+item].isna(),
                     'secondary_cleaner.state.floatbank6_a_'+item] = data['secor
                                                                         +item]


    #калькуляция

    for conc in ['sulfate_to_au_concentrate',
                     'floatbank10_sulfate_to_au_feed',
                     'floatbank11_sulfate_to_au_feed']:
        if ('rougher.calculation.'+conc) in data.columns:
                data.loc[data['rougher.calculation.'+conc].isna(),
                         'rougher.calculation.'+conc] = data['rougher.calculatic

    return data
full = passing(full)
train = passing(train)
test = passing(test)

```

In [19]:

```
#test.isna().sum()
```

Анализ данных

Как меняется концентрация металлов (Au, Ag, Pb) на различных этапах очистки

In [20]:

```
concentrate_au = pd.DataFrame()
concentrate_ag = pd.DataFrame()
concentrate_pb = pd.DataFrame()

concentrate_au['input_au'] = full['rougher.input.feed_au']
concentrate_ag['input_ag'] = full['rougher.input.feed_ag']
concentrate_pb['input_pb'] = full['rougher.input.feed_pb']
for one in ['rougher', 'primary_cleaner','final']:
    lev = '.output.concentrate_'

    concentrate_au[one+'_au'] = full[one+lev+'au']
    concentrate_ag[one+'_ag'] = full[one+lev+'ag']
    concentrate_pb[one+'_pb'] = full[one+lev+'pb']

concentrate_au
```

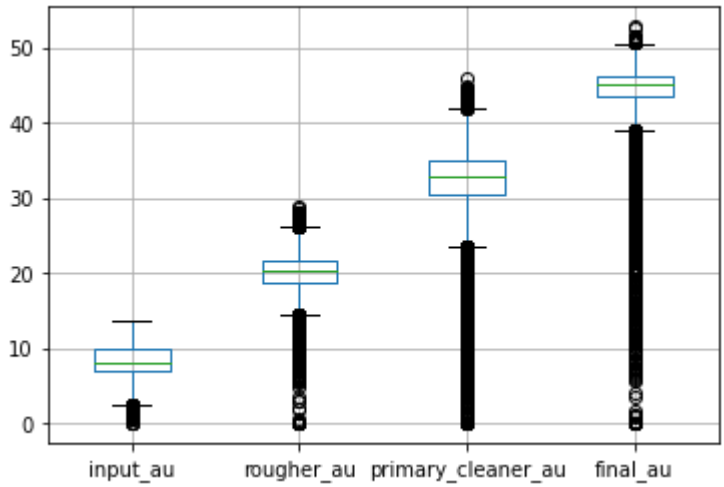
Out[20]:

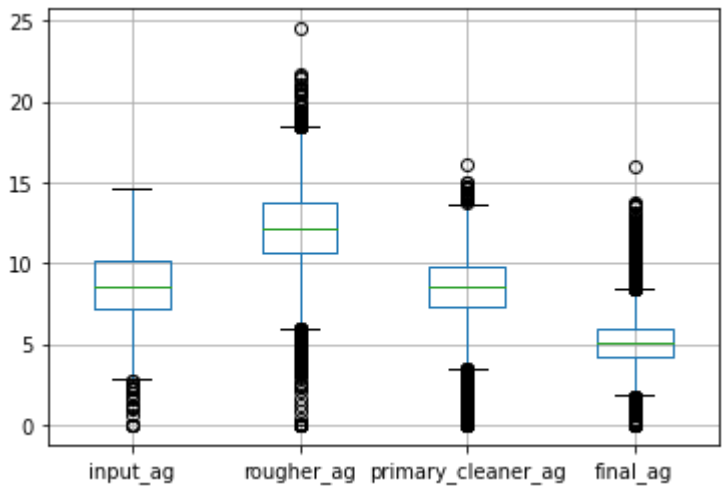
	input_au	rougher_au	primary_cleaner_au	final_au
0	6.486150	19.793808	34.174427	42.192020
1	6.478583	20.050975	34.118526	42.701629
2	6.362222	19.737170	33.969464	42.657501
3	6.118189	19.320810	28.260743	42.689819
4	5.663707	19.216101	33.044932	42.774141
...	...	...	...	...
19434	5.335862	18.603550	32.940215	46.713954
19435	4.838619	18.441436	32.925325	46.866780
19436	4.525061	15.111231	31.856742	46.795691
19437	4.362781	17.834772	30.770892	46.408188
19438	4.365491	17.804134	30.356618	46.299438

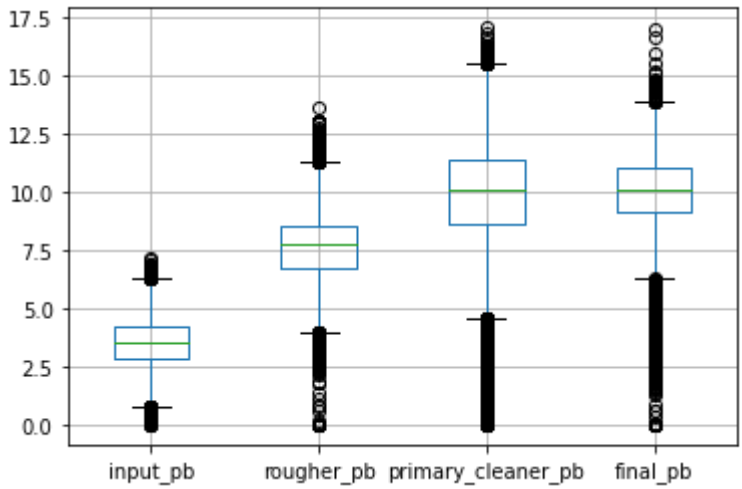
19439 rows × 4 columns

In [21]:

```
concentrate_au.boxplot()
plt.show()
concentrate_ag.boxplot()
plt.show()
concentrate_pb.boxplot()
plt.show()
```





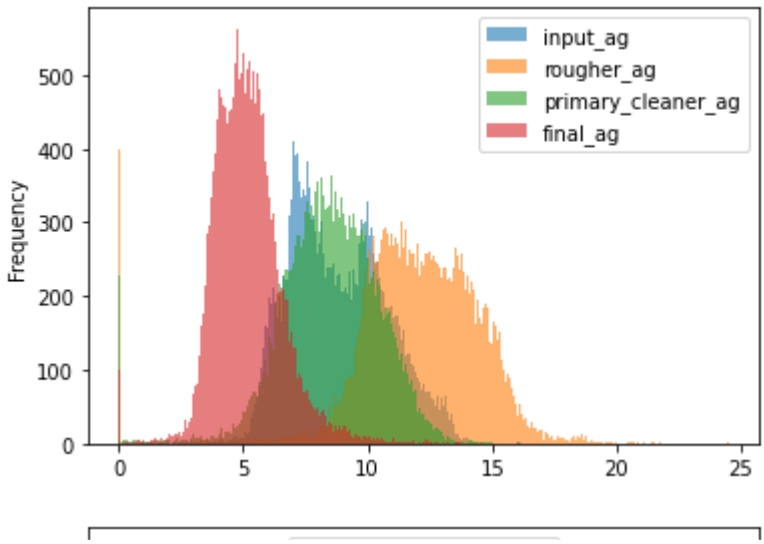


In [22]:

```
concentrate_ag.plot(kind = 'hist',alpha=0.6, bins=300)
concentrate_au.plot(kind = 'hist',alpha=0.6, bins=300)
concentrate_pb.plot(kind = 'hist',alpha=0.6, bins=300)
```

Out[22]:

```
<AxesSubplot:ylabel='Frequency'>
```

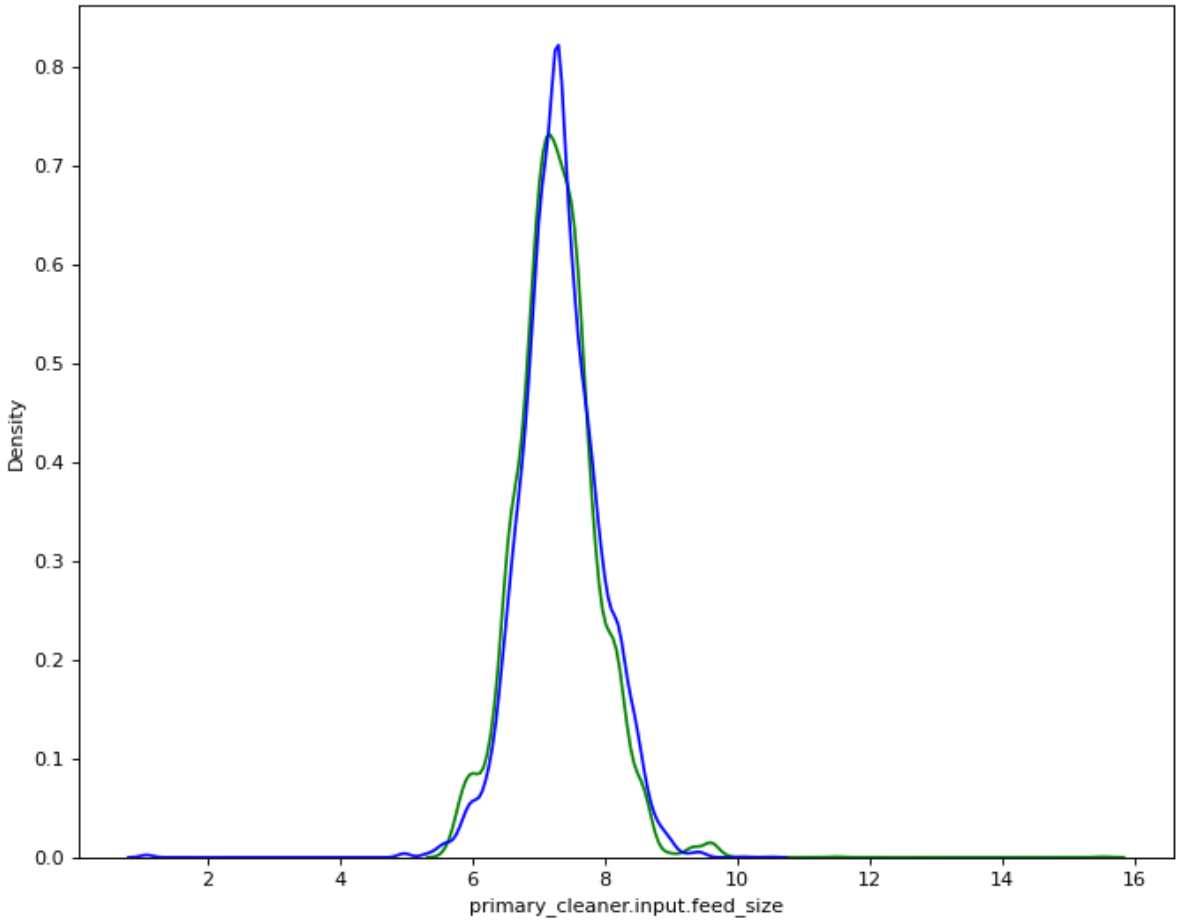


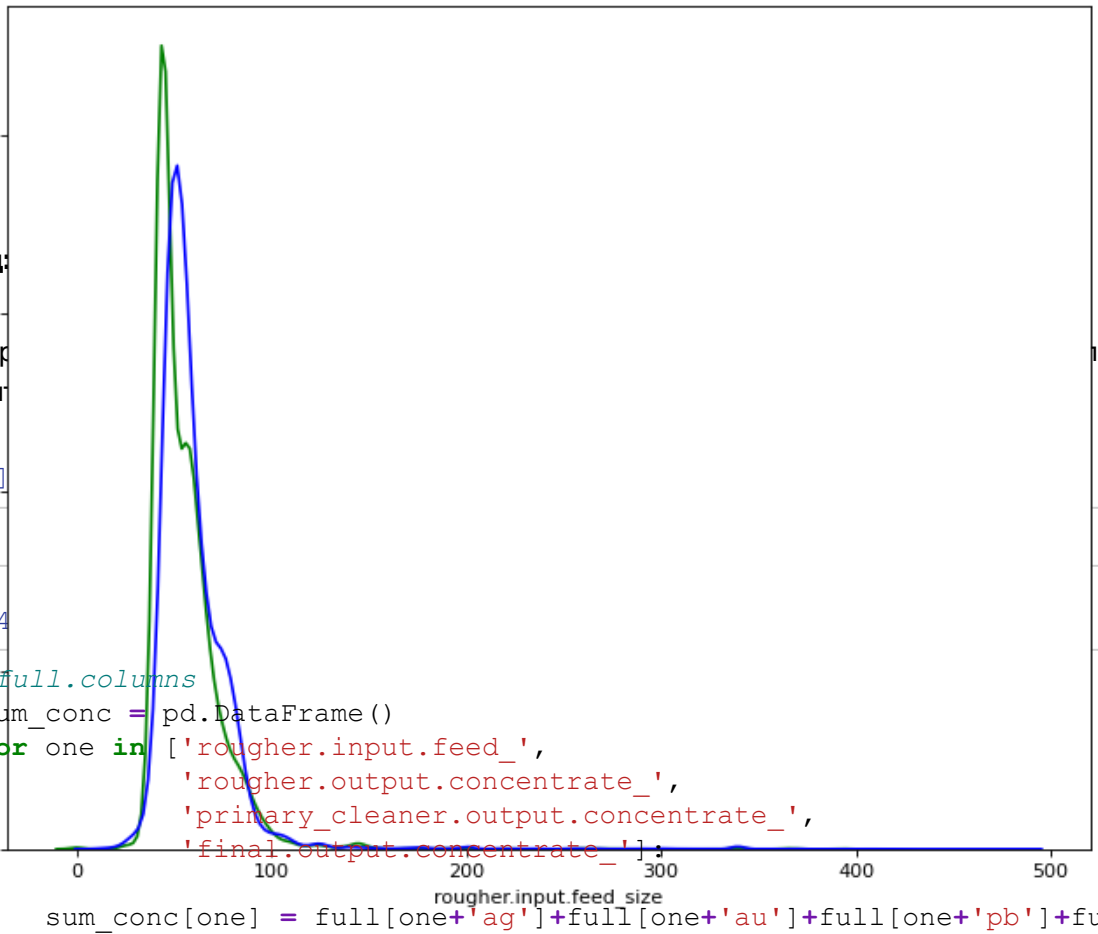
Вывод: концентрация золота увеличивается на каждом этап, концентрация серебра увеличивается на флотации и уменьшается на каждом следующем этапе, концентрация свинца растет на флотации и на первом этапе очистки и не увеличивается во время второй очистки

Сравниваю распределение размеров гранул сырья на обучающей и тестовой выборках.

In [23]:

```
plt.figure(figsize=(10,8), dpi= 80)
sns.kdeplot(test['primary_cleaner.input.feed_size'],color="g", alpha=.7)
sns.kdeplot(train['primary_cleaner.input.feed_size'], color="b", alpha=.7)
plt.show()

plt.figure(figsize=(10,8), dpi= 80)
sns.kdeplot(test['rougher.input.feed_size'],color="g", alpha=.7)
sns.kdeplot(train['rougher.input.feed_size'], color="b", alpha=.7)
plt.show()
```





Вывод:

Суммар ьном
концен

In [ ]:

```
1
```

In [24]:

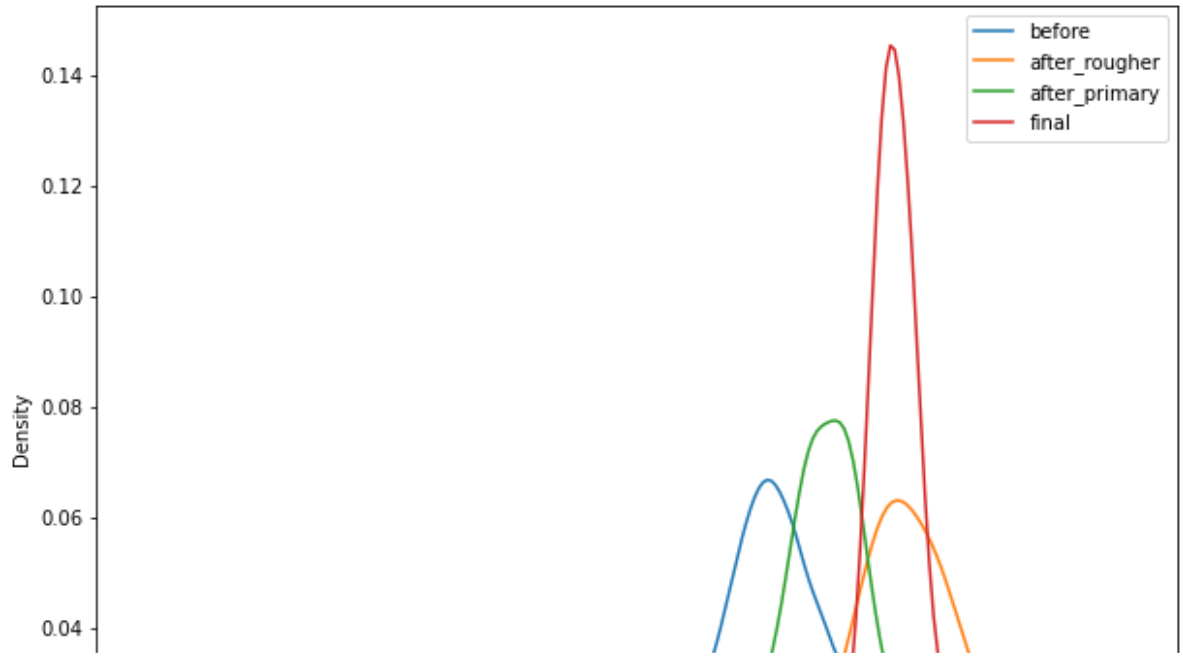
```
#full.columns
sum_conc = pd.DataFrame()
for one in ['rougher.input.feed_',
            'rougher.output.concentrate_',
            'primary_cleaner.output.concentrate_',
            'final.output.concentrate_']:
    sum_conc[one] = full[one+'ag']+full[one+'au']+full[one+'pb']+full[one+'sol']

sum_conc.columns = ['rougher.input','rougher.output','primary.output','final.out
sum_conc
```

Out[24]:

	rougher.input	rougher.output	primary.output	final.output
0	51.680034	66.424950	72.640924	63.644396
1	50.659114	67.012710	72.543485	63.957723
2	50.609929	66.103793	72.095042	64.311180
3	51.061546	65.752751	59.957723	63.573449
4	47.859163	65.908382	71.321010	64.004667
...	...	...	...	...
19434	53.415050	70.781325	59.001692	68.098589
19435	53.696482	70.539603	59.703912	68.274362
19436	54.589604	55.376330	59.561096	68.226068
19437	54.027355	69.201689	57.216686	68.200449

In [25]:

```
plt.figure(figsize=(10,8))
legend = True
sns.kdeplot(sum_conc['rougher.input'],alpha=.7, label = 'before')
sns.kdeplot(sum_conc['rougher.output'], alpha=.7, label = 'after_rougher')
sns.kdeplot(sum_conc['primary.output'], alpha=.7, label = 'after_primary')
sns.kdeplot(sum_conc['final.output'], alpha=.7, label='final')
plt.legend()
plt.show()
```



In [ ]:

```
1
```

Выводы: Есть небольшое количество данных с практически нулевой концентрацией веществ на всех этапах. Суммарная концентрация меняется в зависимости от этапа обработки. Сначала снижается после флотации, затем возвращается после первого этапа очистки. Стремительно возрастает после второго этапа очистки

Заменим X на Z и вычислим, чему будет равено предсказание и вектор весов.

$$a_1 = Zw_1$$
$$qquad(2.2)$$
$$w_1 = (Z^T Z)^{-1} Z^T y$$

Подставим уравнение 2.2 правую часть уравнения 2.3 и получим следущее:

$$a1 = Z(Z^T Z)^{-1} Z^T y$$

Заменим все Z правой частью уравнения 2.1:

$$a_1 = XP((XP)^T(XP))^{-1}(XP)^T y$$
$$qquad(2.5)$$

Для следующего шага понадобится следующее свойсво обратной матрицы:

$$(AB)^{-1} = B^{-1}A^{-1}$$

Раскроем $((XP)^T(XP))^{-1}$ в два шага:

$$a_1 = XP((XP)^T(XP))^{-1}(XP)^T y = XP(XP)^{-1}((XP)^T)^{-1}(XP)^T y = XPP^{-1}X^{-1}((XP)^T)^{-1}(XP)^T y$$
$$qquad(2.6)$$

Умножение приведет к $PP^{-1} = E$. Для следуюзего этапа воспользуемся свойством транспорнированной матрицы:

$$(AB)^T = B^T A^T$$

Умножение на единичную матрицу ничего не меняет. Раскроем $((XP)^T)^{-1}(XP)^T$ в три шага:

$$a_1 = XEX^{-1}((XP)^T)^{-1}(XP)^T y = XX^{-1}(P^T X^T)^{-1}P^T X^T y = XX^{-1}(X^T)^{-1}(P^T)^{-1}P^T X^T y$$
$$qquad(2.7)$$

Умножение приведет к $(P^T)^{-1}P^T = E$. Посмотрим, что осталось от уравнения 2.7:

$$a_1 = XX^{-1}(X^T)^{-1}EX^T y = X(X^T X)^{-1}X^T y = Xw = a$$
$$qquad(2.8)$$

"Как видно, значение предсказания a не меняется, если умножать матрицу признаков на обратимую матрицу."

In [26]:

```
for one in ['rougher.input.feed_',
            'rougher.output.concentrate_',
            'primary_cleaner.output.concentrate_',
            'final.output.concentrate_']:
    train = train[train[one+'au']+train[one+'ag']+train[one+'pb']+train[one+'sol
train[train[one+'au']+train[one+'ag']+train[one+'pb']+train[one+'sol'] < 1]
```

Out[26]:

date	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concentrate_sol	final.o

0 rows × 87 columns

Модель

Метрики качества

In [27]:

```
#перенесла отсюда функцию поближе к коду
```

In [28]:

```
#train.columns

```

In [29]:

```
train.index = train.date
test.index = test.date
full.index = full.date

test_target = pd.DataFrame()
test_target['date'] = test['date']
test_target.index = test_target.date
test_target[['rougher.output.recovery','final.output.recovery']] = full[['roughe

train[['rougher.output.recovery','final.output.recovery']] = full[['rougher.outp
train_target = train[['rougher.output.recovery','final.output.recovery']]
train = train[test.columns]


train.shape, train_target.shape, test.shape, test_target.shape
```

Out[29]:

```
((13599, 53), (13599, 2), (5290, 53), (5290, 3))
```

In [30]:

```
test_target[test_target['final.output.recovery'].isna()]=0
test_target[test_target['final.output.recovery'].isna()]
```

Out[30]:

date	rougher.output.recovery	final.output.recovery

date

удалю из обучающей выборки нули в целевом признаке (это ухудшило качество модели)

In [31]:

```
#train_target = train_target[(train_target['rougher.output.recovery']!=0)&(trair
#train_target = train_target.dropna()
#
#train = train[train.index.isin(train_target.index)]
#train.shape, train_target.shape
```

In [32]:

```
train_target[train_target['final.output.recovery'].isna()]
```

Out[32]:

date	rougher.output.recovery	final.output.recovery

date

Посмотрю на корреляцию признаков

In [33]:

```
#pd.set_option('display.max_rows',None)
#pd.set_option('display.max_columns',None)
full.corr()
#primary_cleaner.output.concentrate_ag
```

Out[33]:

	final.output.concentrate_ag	final.output.concentrate_pb
final.output.concentrate_ag	1.000000	0.063618
final.output.concentrate_pb	0.063618	1.000000
final.output.concentrate_sol	0.345467	-0.041863
final.output.concentrate_au	-0.038185	0.374439
final.output.recovery	0.185964	0.151998
...	...	...
secondary_cleaner.state.floatbank5_b_level	0.156545	-0.059971
secondary_cleaner.state.floatbank6_a_air	0.162637	-0.042138
secondary_cleaner.state.floatbank6_a_level	0.059583	0.067421
rougher.recovery_test	0.084139	0.039727
final.recovery_test	-0.000850	0.037343

88 rows × 88 columns

Мне кажется, нет признаков, которые бы сильно влияли на целевые, а те, которые могли бы влиять, уже удалены в тестовой выборке

In [34]:

```
train = train.drop('date', axis=1)
test = test.drop('date', axis=1)
test_target = test_target.drop('date', axis=1)
```

3.2. Обучите разные модели и оцените их качество кросс-валидацией. Выберите лучшую модель и проверьте её на тестовой выборке. Опишите выводы.

In [35]:

```
def calc_smape(target, prediction):
    #функция принимает на вход целевой признак и предсказание и вычисляет smape
    #симметричное среднее абсолютное процентное отклонение
    smape = 0
    for i in range(len(target)):
        smape += (1/len(target)) * (abs(target[i]-prediction[i])) / ((abs(target
    return smape

def calc_itog_smape(target, prediction):
    target= np.array(target)
    #функция вычисляет итоговую метрику, принимая на вход smape двух величин
    smape_one = calc_smape(target[:,0],prediction[:,0])
    smape_two = calc_smape(target[:,1], prediction[:,1])
    smape_itog = 0.25*smape_one+0.75*smape_two

    return smape_itog


```

Константная модель

In [36]:

```
constant_predict = pd.DataFrame(index = train_target.index, columns = ['rougher.
constant_predict['rougher.output.recovery'] = constant_predict['rougher.output.r
constant_predict['final.output.recovery'] = constant_predict['final.output.recov
constant_predict  = constant_predict.to_numpy()
constant_smape = calc_itog_smape(train_target, constant_predict)
constant_smape
```

Out[36]:

```
9.339089690445489
```

Случайный лес

In [ ]:

```
в

```

In [37]:

```
best_est = 0
best_depth = 0
best_cross = -10000
best_split = 0
best_leaf = 0
prediction = pd.DataFrame()
for est in range(25,26): #подобрала лучшие гиперпарамы
    for depth in range(3,4):
        for split in range(1,2):
            for leaf in range(136,137):
                model_f = RandomForestRegressor(n_estimators=est,
                                                max_depth=depth,
                                                random_state=12345,
                                                min_samples_split=split/10,
                                                min_samples_leaf=leaf)

                scor = make_scorer(calc_itog_smape, greater_is_better=False)
                cross = cross_val_score(model_f, train, train_target, cv=5, scor
                if cross.mean()>best_cross:
                    best_est = est
                    best_depth = depth
                    best_cross = cross.mean()
                    best_split = split
                    best_leaf = leaf
                #print(cross.mean(), est, depth, split, leaf)
print(best_cross*(-1), 'Лучшие гиперпараметры:', best_est,  best_depth, best_spl


```

```
8.662527213987541 Лучшие гиперпараметры: 25 3 1 136
```

Стандартизация - не влияет на результат

In [38]:

```
pd.options.mode.chained_assignment = None
scaler = StandardScaler()
scaler.fit(train)
train_scaled = scaler.transform(train)
test_scaled = scaler.transform(test)

```

In [46]:

```
prediction_l = pd.DataFrame()
model_l = LinearRegression()
model_l.fit(train, train_target)
scor = make_scorer(calc_itog_smape, greater_is_better=False)
cross_l = cross_val_score(model_l, train, train_target, cv=5, scoring = scor)

print(cross_l.mean(), cross_l)

```

```
-10.014241894204414 [-12.03005649  -8.86985447  -9.30815727  -7.843783
54 -12.01935769]
```

In [45]:

```
model_f.fit(train,train_target)
predict_test_f = model_f.predict(test)
smape_test_f = calc_itog_smape(test_target, predict_test_f)
smape_test_f
```

Out[45]:

```
9.32012722153264
```

Лучшая модель - случайный лес. SMAPE на тестовой выборке - 9.32