

Ирина, привет! 🤖

Меня зовут Алексей Гриб, и я буду ревьюером твоего проекта.

Сразу хочу предложить в дальнейшем общаться на "ты" - надеюсь, так будет комфортнее:) Но если это неудобно, обязательно дай знать, и мы придумаем что-нибудь ещё!

Цель ревью - не искать ошибки в твоём проекте, а помочь тебе сделать твою работу ещё лучше, устранив недочёты и приблизив её к реальным задачам специалиста по работе с данными. Поэтому не расстраивайся, если что-то не получилось с первого раза - это нормально, и это поможет тебе вырасти!

Ты можешь найти мои комментарии, обозначенные **зеленым**, **желтым** и **красным** цветами, например:

Комментарий ревьюера

Все отлично! 🍌: похвала, рекомендации «со звёздочкой», полезные лайфхаки, которые сделают и без того красивое решение ещё более элегантным.

Комментарий ревьюера

Некоторые замечания и рекомендации ⚠️: некритичные ошибки или развивающие рекомендации на будущее.

Комментарий ревьюера

На доработку ✖️: Критичные ошибки, которые обязательно нужно исправить.

Пожалуйста, не удаляй мои комментарии, они будут особенно полезны для нашей работы в случае повторной проверки проекта.

Ты также можешь задавать свои вопросы, реагировать на мои комментарии, делать пометки и пояснения - полная творческая свобода! Но маленькая просьба - пускай они будут отличаться от моих комментариев, это поможет избежать путаницы в нашем общении:) Например, вот так:

Комментарий студента

твой текст

Давай посмотрим на твой проект!

Содержание

- [1 Подготовка данных](#)
 - [1.1 Изучение данных](#)
 - [1.2 Расчет эффективности обогащения - коэффициент восстановления золота из золотосодержащей руды](#)
 - [1.3 Анализ параметров не указанных в тестовой выборке](#)
 - [1.4 Предобработка данных](#)
 - [1.4.1 Обработка нулей](#)
 - [1.4.2 Обработка пропусков](#)
 - [1.4.3 Изучение пропущенных в тестовой выборке данных](#)
- [2 Анализ данных](#)
 - [2.1 Концентрации металлов на различных этапах очистки](#)
 - [2.2 Распределение размеров гранул сырья в выборках](#)
 - [2.3 Суммарная концентрация веществ на разных стадиях](#)
- [3 Модель](#)
 - [3.1 Выбор признаков для обучения](#)
 - [3.2 Функция для расчета sMAPE](#)
 - [3.3 Разделение данных на выборки](#)
 - [3.4 Модель решающее дерево](#)
 - [3.5 Случайный лес](#)
 - [3.6 Линейная регрессия](#)
 - [3.7 Проверка наилучшей модели на тестовой выборке](#)
 - [3.8 Адекватность модели](#)

✓ Восстановление золота из руды

Подготовьте прототип модели машинного обучения для «Цифры». Компания разрабатывает решения для эффективной работы промышленных предприятий.

Модель должна предсказать коэффициент восстановления золота из золотосодержащей руды. Используйте данные с параметрами добычи и очистки.

Модель поможет оптимизировать производство, чтобы не запускать предприятие с убыточными характеристиками.

Вам нужно:

1. Подготовить данные;
2. Провести исследовательский анализ данных;
3. Построить и обучить модель.

Чтобы выполнить проект, обращайтесь к библиотекам *pandas*, *matplotlib* и *sklearn*. Вам поможет их документация.

✓ Описание проекта

Задача проекта

Подготовьте прототип модели машинного обучения для золотодобывающей отрасли.

Модель должна предсказать коэффициент восстановления золота из золотосодержащей руды.

Модель должна использовать данные с параметрами добычи и очистки.

Цели проекта

Модель поможет оптимизировать производство, чтобы не запускать предприятие с убыточными характеристиками.

Заказчик проекта

Компания "Цифры", разрабатывающая решения для промышленных предприятий, в данном случае для золотодобывающей отрасли

Входные данные:

Исходные данные:

- `gold_recovery_train_new.csv` — обучающая выборка;
- `gold_recovery_test_new.csv` — тестовая выборка;
- `gold_recovery_full_new.csv` — исходные данные.

Данные индексируются датой и временем получения информации (признак `date`). Соседние по времени параметры часто похожи. Некоторые параметры недоступны, потому что замеряются и/или рассчитываются значительно позже. Из-за этого в тестовой выборке отсутствуют некоторые признаки, которые могут быть в обучающей. Также в тестовом наборе нет целевых признаков. Исходный датасет содержит обучающую и тестовую выборки со всеми признаками. В вашем распоряжении сырые данные: их просто выгрузили из хранилища. Необходима проверка данных на корректность.

Описание данных:

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

Наименование признаков: [этап].[тип_параметра].[название_параметра]

Пример: rougher.input.feed_ag

Возможные значения для блока [этап]:

- rougher — флотация
- primary_cleaner — первичная очистка
- secondary_cleaner — вторичная очистка
- final — финальные характеристики

Возможные значения для блока [тип_параметра]:

- input — параметры сырья
- output — параметры продукта
- state — параметры, характеризующие текущее состояние этапа
- calculation — расчётные характеристики

План исследования:

- 1) Подготовка данных
- 2) Анализ данных
- 3) Модель
- 4) Общие выводы

Комментарий ревьюера

Все отлично! 🍌: Хорошее вступление!

В нём есть всё, что необходимо, чтобы понять суть проекта с первых строк отчёта!

✓ Подготовка данных

#подключаемые библиотеки

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec #для сетки построения графиков
```

```
#библиотеки для построения моделей регрессии (целевой признак - количественный)
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
```


```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error # для метрики mrse
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.dummy import DummyRegressor
```

Комментарий ревьюера

Все отлично! 🍌: Библиотеки импортировали - отлично!

✓ Изучение данных

```
#исходные данные
df_full = pd.read_csv('/datasets/gold_recovery_full_new.csv')
df_full.head()
```



	date	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concentrate_sol	final.output.concentrate_au	fi
0	2016-01-15 00:00:00	6.055403	9.889648	5.507324	42.192020	
1	2016-01-15 01:00:00	6.029369	9.968944	5.257781	42.701629	
2	2016-01-15 02:00:00	6.055926	10.213995	5.383759	42.657501	
3	2016-01-15 03:00:00	6.047977	9.977019	4.858634	42.689819	
4	2016-01-15 04:00:00	6.148599	10.142511	4.939416	42.774141	

5 rows × 87 columns

Комментарий ревьюера

Все отлично!👍

Данные загрузили.

При считывании данных из файла здорово перестраховывать себя от ошибок, связанных, например, с неверным указанием пути к файлу. А иногда бывает, что работаешь с файлом локально, выгружаешь его на сервер, ожидая, что он будет принимать данные, которые лежат на том же сервере, а код падает с ошибкой, потому что путь к файлу не поменялся с локального на серверный.

Для этого, например, можно использовать конструкцию `try-except`: сначала пробуешь локальный путь, при возникновении ошибки используется серверный путь (подробнее можешь почитать тут: <https://pythonworld.ru/typy-dannyx-v-python/isklyucheniya-v-python-konstrukciya-try-except-dlya-obrabotki-isklyuchenij.html>).

Но еще лучше использовать библиотеку `os` - её использование позволит тебе проверять существование указанных директорий (что может быть актуально при одновременной работа на локальном и сетевом окружении) и загружать данные из существующей директории, избегая ошибок. Как пример:


```
import os

pth1 = '/folder_1/data.csv'
pth2 = '/folder_2/data.csv'

if os.path.exists(pth1):
    query_1 = pd.read_csv(pth1)
elif os.path.exists(pth2):
    query_1 = pd.read_csv(pth2)
else:
    print('Something is wrong')
```

Ещё на этапе считывания данных можно спарсить дату: за это действие отвечает параметр `parse_dates` метода `read_csv()`, в него нужно передать список с названием полей-дат, и в большинстве случаев дата будет корректно преобразована в нужный формат сразу.) Также на этапе считывания данных задать индекс-столбец- за это действие отвечает параметр `index_col`.

```
df_full.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19439 entries, 0 to 19438
Data columns (total 87 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   date                                19439 non-null  object
1   final.output.concentrate_ag         19438 non-null  float64
2   final.output.concentrate_pb         19438 non-null  float64
3   final.output.concentrate_sol        19228 non-null  float64
4   final.output.concentrate_au         19439 non-null  float64
5   final.output.recovery               19439 non-null  float64
6   final.output.tail_ag                19438 non-null  float64
7   final.output.tail_pb                19338 non-null  float64
8   final.output.tail_sol               19433 non-null  float64
9   final.output.tail_au                19439 non-null  float64
10  primary_cleaner.input.sulfate       19415 non-null  float64
11  primary_cleaner.input.depressant    19402 non-null  float64
```

```

12 primary_cleaner.input.feed_size      19439 non-null float64
13 primary_cleaner.input.xanthate       19335 non-null float64
14 primary_cleaner.output.concentrate_ag 19439 non-null float64
15 primary_cleaner.output.concentrate_pb 19323 non-null float64
16 primary_cleaner.output.concentrate_sol 19069 non-null float64
17 primary_cleaner.output.concentrate_au 19439 non-null float64
18 primary_cleaner.output.tail_ag        19435 non-null float64
19 primary_cleaner.output.tail_pb        19418 non-null float64
20 primary_cleaner.output.tail_sol        19377 non-null float64
21 primary_cleaner.output.tail_au         19439 non-null float64
22 primary_cleaner.state.floatbank8_a_air 19435 non-null float64
23 primary_cleaner.state.floatbank8_a_level 19438 non-null float64
24 primary_cleaner.state.floatbank8_b_air 19435 non-null float64
25 primary_cleaner.state.floatbank8_b_level 19438 non-null float64
26 primary_cleaner.state.floatbank8_c_air 19437 non-null float64
27 primary_cleaner.state.floatbank8_c_level 19438 non-null float64
28 primary_cleaner.state.floatbank8_d_air 19436 non-null float64
29 primary_cleaner.state.floatbank8_d_level 19438 non-null float64
30 rougher.calculation.sulfate_to_au_concentrate 19437 non-null float64
31 rougher.calculation.floatbank10_sulfate_to_au_feed 19437 non-null float64
32 rougher.calculation.floatbank11_sulfate_to_au_feed 19437 non-null float64
33 rougher.calculation.au_pb_ratio        19439 non-null float64
34 rougher.input.feed_ag                  19439 non-null float64
35 rougher.input.feed_pb                  19339 non-null float64
36 rougher.input.feed_rate                19428 non-null float64
37 rougher.input.feed_size                19294 non-null float64
38 rougher.input.feed_sol                  19340 non-null float64
39 rougher.input.feed_au                  19439 non-null float64
40 rougher.input.floatbank10_sulfate       19405 non-null float64
41 rougher.input.floatbank10_xanthate      19431 non-null float64
42 rougher.input.floatbank11_sulfate       19395 non-null float64
43 rougher.input.floatbank11_xanthate      18986 non-null float64
44 rougher.output.concentrate_ag           19439 non-null float64
45 rougher.output.concentrate_pb           19439 non-null float64
46 rougher.output.concentrate_sol          19416 non-null float64
47 rougher.output.concentrate_au           19439 non-null float64
48 rougher.output.recovery                 19439 non-null float64
49 rougher.output.tail_ag                  19438 non-null float64
50 rougher.output.tail_pb                  19439 non-null float64
51 rougher.output.tail_sol                  19439 non-null float64
52 rougher.output.tail_au                  19439 non-null float64

```

Исходные данные содержат дату и 86 параметров, строк в данных: 19439

```
df_full.describe()
```

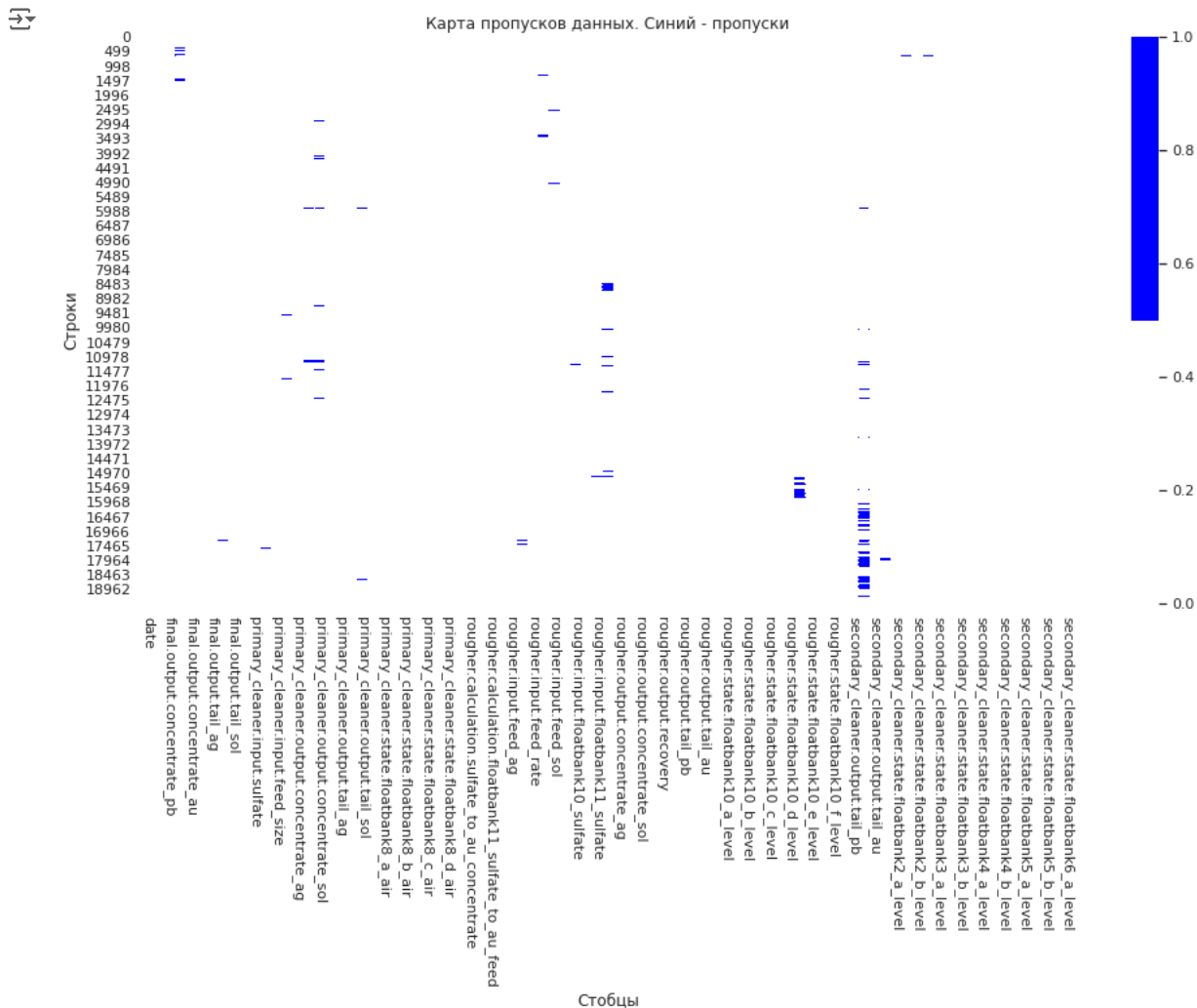
	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concentrate_sol	final.output.concentrate_au	final.output.tail_ag
count	19438.000000	19438.000000	19228.000000	19439.000000	19439.000000
mean	5.168470	9.978895	9.501224	44.076513	19.438470
std	1.372348	1.669240	2.787537	5.129784	19.438470
min	0.000000	0.000000	0.000000	0.000000	19.438470
25%	4.251240	9.137262	7.722820	43.402215	19.438470
50%	5.066094	10.102433	9.218961	45.011244	19.438470
75%	5.895527	11.035769	10.947813	46.275313	19.438470
max	16.001945	17.031899	19.615720	52.756638	19.438470

8 rows × 6 columns

```

#карта пропусков данных
cols = df_full.columns
# определяем цвета
# синий - пропущенные данные, белый - не пропущенные
colours = ['#ffffff', '#0000ff']
sns.set(rc = {'figure.figsize':(16,8)})
sns.heatmap(df_full[cols].isnull(), cmap=sns.color_palette(colours))
plt.title('Карта пропусков данных. Синий - пропуски')
plt.xticks(rotation=-90)
plt.xlabel('Столбцы')
plt.ylabel('Строки')
plt.grid()
plt.show()

```




```
# посмотрим какие параметры содержат нули
# параметры с содержанием золота не должны быть нулевыми
for col in df_full:
    count_nul = df_full.loc[ df_full[col]==0, col].count()
    if count_nul != 0:
        print(col,':', count_nul)
```

```
final.output.concentrate_ag : 98
final.output.concentrate_pb : 98
final.output.concentrate_sol : 98
final.output.concentrate_au : 98
final.output.recovery : 98
final.output.tail_ag : 91
final.output.tail_pb : 91
final.output.tail_sol : 91
final.output.tail_au : 91
primary_cleaner.input.depressant : 10
primary_cleaner.output.concentrate_ag : 108
primary_cleaner.output.concentrate_pb : 108
primary_cleaner.output.concentrate_sol : 108
primary_cleaner.output.concentrate_au : 108
primary_cleaner.output.tail_ag : 124
primary_cleaner.output.tail_pb : 124
primary_cleaner.output.tail_sol : 124
primary_cleaner.output.tail_au : 124
rougher.output.concentrate_ag : 394
rougher.output.concentrate_pb : 394
rougher.output.concentrate_sol : 394
rougher.output.concentrate_au : 394
rougher.output.recovery : 394
secondary_cleaner.output.tail_ag : 849
secondary_cleaner.output.tail_pb : 849
secondary_cleaner.output.tail_sol : 849
secondary_cleaner.output.tail_au : 849
secondary_cleaner.state.floatbank2_b_air : 8
secondary_cleaner.state.floatbank3_a_air : 2
secondary_cleaner.state.floatbank3_b_air : 1
secondary_cleaner.state.floatbank4_a_air : 1
secondary_cleaner.state.floatbank4_b_air : 1
```

В значениях параметров обнаружены нули, но в параметрах золотоносной смеси их быть не может (хоть сколько-то, но должно быть золота) Хорошо бы вывести эти строки и посмотреть, что там с данными. Здесь явно закралась какая-то ошибка и 108 строк данных точно можно без сожаления удалить!

- final.output.concentrate_au : 98
- final.output.recovery : 98
- primary_cleaner.output.concentrate_au : 108


```
#обучающая выборка
df_train = pd.read_csv('/datasets/gold_recovery_train_new.csv')
df_train.head()
```



	date	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concentrate_sol	final.output.concentrate_au	fir
0	2016-01-15 00:00:00	6.055403	9.889648	5.507324	42.192020	
1	2016-01-15 01:00:00	6.029369	9.968944	5.257781	42.701629	
2	2016-01-15 02:00:00	6.055926	10.213995	5.383759	42.657501	
3	2016-01-15 03:00:00	6.047977	9.977019	4.858634	42.689819	
4	2016-01-15 04:00:00	6.148599	10.142511	4.939416	42.774141	

5 rows × 87 columns

```
df_train.info()
```



```
31 rougher.calculation.floatbank10_sulfate_to_au_feed 14148 non-null float64
32 rougher.calculation.floatbank11_sulfate_to_au_feed 14148 non-null float64
33 rougher.calculation.au_pb_ratio 14149 non-null float64
34 rougher.input.feed_ag 14149 non-null float64
35 rougher.input.feed_pb 14049 non-null float64
36 rougher.input.feed_rate 14141 non-null float64
37 rougher.input.feed_size 14005 non-null float64
38 rougher.input.feed_sol 14071 non-null float64
39 rougher.input.feed_au 14149 non-null float64
40 rougher.input.floatbank10_sulfate 14120 non-null float64
```

```
// secondary_cleaner.state.floatbank4_a_air
78 secondary_cleaner.state.floatbank4_a_level
79 secondary_cleaner.state.floatbank4_b_air
80 secondary_cleaner.state.floatbank4_b_level
81 secondary_cleaner.state.floatbank5_a_air
82 secondary_cleaner.state.floatbank5_a_level
83 secondary_cleaner.state.floatbank5_b_air
84 secondary_cleaner.state.floatbank5_b_level
85 secondary_cleaner.state.floatbank6_a_air
86 secondary_cleaner.state.floatbank6_a_level
dtypes: float64(86), object(1)
memory usage: 9.4+ MB
```

14143	non-null	float64
14148	non-null	float64
14148	non-null	float64
14148	non-null	float64
14148	non-null	float64
14148	non-null	float64
14148	non-null	float64
14148	non-null	float64
14147	non-null	float64
14148	non-null	float64

Размер обучающей выборки: 14149 строк

```
#тестовая выборка
df_test = pd.read_csv('/datasets/gold_recovery_test_new.csv')
df_test.head()
```

	date	primary_cleaner.input.sulfate	primary_cleaner.input.depressant	primary_cleaner.input.feed_size	primary_cleaner.input
0	2016-09-01 00:59:59	210.800909	14.993118	8.080000	
1	2016-09-01 01:59:59	215.392455	14.987471	8.080000	
2	2016-09-01 02:59:59	215.259946	12.884934	7.786667	
3	2016-09-01 03:59:59	215.336236	12.006805	7.640000	
4	2016-09-01 04:59:59	199.099327	10.682530	7.530000	

5 rows × 53 columns

```
df_test.info()
```

Data columns (total 53 columns):				
#	Column	Non-Null Count		Dtype
0	date	5290 non-null		object
1	primary_cleaner.input.sulfate	5286 non-null		float64
2	primary_cleaner.input.depressant	5285 non-null		float64
3	primary_cleaner.input.feed_size	5290 non-null		float64
4	primary_cleaner.input.xanthate	5286 non-null		float64
5	primary_cleaner.state.floatbank8_a_air	5290 non-null		float64
6	primary_cleaner.state.floatbank8_a_level	5290 non-null		float64
7	primary_cleaner.state.floatbank8_b_air	5290 non-null		float64
8	primary_cleaner.state.floatbank8_b_level	5290 non-null		float64
9	primary_cleaner.state.floatbank8_c_air	5290 non-null		float64
10	primary_cleaner.state.floatbank8_c_level	5290 non-null		float64
11	primary_cleaner.state.floatbank8_d_air	5290 non-null		float64
12	primary_cleaner.state.floatbank8_d_level	5290 non-null		float64
13	rougher.input.feed_ag	5290 non-null		float64
14	rougher.input.feed_pb	5290 non-null		float64
15	rougher.input.feed_rate	5287 non-null		float64
16	rougher.input.feed_size	5289 non-null		float64
17	rougher.input.feed_sol	5269 non-null		float64
18	rougher.input.feed_au	5290 non-null		float64
19	rougher.input.floatbank10_sulfate	5285 non-null		float64
20	rougher.input.floatbank10_xanthate	5290 non-null		float64
21	rougher.input.floatbank11_sulfate	5282 non-null		float64
22	rougher.input.floatbank11_xanthate	5265 non-null		float64
23	rougher.state.floatbank10_a_air	5290 non-null		float64
24	rougher.state.floatbank10_a_level	5290 non-null		float64
25	rougher.state.floatbank10_b_air	5290 non-null		float64
26	rougher.state.floatbank10_b_level	5290 non-null		float64
27	rougher.state.floatbank10_c_air	5290 non-null		float64
28	rougher.state.floatbank10_c_level	5290 non-null		float64
29	rougher.state.floatbank10_d_air	5290 non-null		float64
30	rougher.state.floatbank10_d_level	5290 non-null		float64
31	rougher.state.floatbank10_e_air	5290 non-null		float64
32	rougher.state.floatbank10_e_level	5290 non-null		float64
33	rougher.state.floatbank10_f_air	5290 non-null		float64
34	rougher.state.floatbank10_f_level	5290 non-null		float64
35	secondary_cleaner.state.floatbank2_a_air	5287 non-null		float64
36	secondary_cleaner.state.floatbank2_a_level	5290 non-null		float64


```

39 secondary_cleaner.state.floatbank3_a_air 5290 non-null float64
40 secondary_cleaner.state.floatbank3_a_level 5290 non-null float64
41 secondary_cleaner.state.floatbank3_b_air 5290 non-null float64
42 secondary_cleaner.state.floatbank3_b_level 5290 non-null float64
43 secondary_cleaner.state.floatbank4_a_air 5290 non-null float64
44 secondary_cleaner.state.floatbank4_a_level 5290 non-null float64
45 secondary_cleaner.state.floatbank4_b_air 5290 non-null float64
46 secondary_cleaner.state.floatbank4_b_level 5290 non-null float64
47 secondary_cleaner.state.floatbank5_a_air 5290 non-null float64
48 secondary_cleaner.state.floatbank5_a_level 5290 non-null float64
49 secondary_cleaner.state.floatbank5_b_air 5290 non-null float64
50 secondary_cleaner.state.floatbank5_b_level 5290 non-null float64
51 secondary_cleaner.state.floatbank6_a_air 5290 non-null float64
52 secondary_cleaner.state.floatbank6_a_level 5290 non-null float64
dtypes: float64(52), object(1)
memory usage: 2.1+ MB

```

Тестовая выборка размер: 5290, 1-дата и 52 - параметра

Выводы

- Размер исходных данных: (19439; 87)
- Размер обучающей выборки: (14149; 87)
- Размер тестовой выборки: (5290; 53) - не содержит целевой признак
- В данных наблюдаются пропуски
- Количество колонок в тестовой выборке не соответствует количеству колонок в полных данных
- Наблюдаются нулевые данные для золотосодержащего концентрата, что свидетельствует об ошибках в данных

Комментарий ревьюера

Все отлично! 👍

Данные рассмотрели.

✓ Расчет эффективности обогащения - коэффициент восстановления золота из золотосодержащей руды

Проверим, что эффективность обогащения рассчитана правильно.

Вычислим эффективность на обучающей выборке для признака `rougher.output.recovery` - коэффициент обогащения для чернового концентрата.

Эффективность обогащения рассчитывается по формуле:

$$\text{RECOVERY} = C(F-T)/[F(C-T)]$$

где:

- C — доля золота в концентрате после флотации/очистки;
- F — доля золота в сырье/концентрате до флотации/очистки;
- T — доля золота в отвальных хвостах после флотации/очистки.

Для прогноза коэффициента нужно найти долю золота в концентратах и хвостах. Причём важен не только финальный продукт, но и черновой концентрат.

Коэффициент обогащения для чернового концентрата:

$$\text{RECOVERY}_{\text{rougher}} = C1(F1-T1)/[F1(C1-T1)]$$

где:

- C1 — доля золота в черновом концентрате;
- F1 — доля золота в сырье/концентрате до флотации;
- T1 — доля золота в отвальных хвостах после флотации.

Коэффициент обогащения для финального концентрата:

$$\text{RECOVERY}_{\text{final}} = C2(F2-T2)/[F2(C2-T2)]$$

где:

- C2 — доля золота в концентрате после очистки, в финальном концентрате;
- F2=C1 — доля золота в черновом концентрате;
- T2 — доля золота в финальных отвальных хвостах после очистки (за вычетом T1 доли золота в отвальных хвостах после флотации)

```
print('Концентрация золота в смеси в отвальных хвостах после каждого этапа:')
display(df_train[['rougher.output.tail_au',
                  'primary_cleaner.output.tail_au',
                  'secondary_cleaner.output.tail_au',
                  'final.output.tail_au']].head(2))

print('Концентрация золота в смеси после каждого этапа:')

display(df_train[['rougher.input.feed_au',
                  'rougher.output.concentrate_au',
                  'primary_cleaner.output.concentrate_au',
                  'final.output.concentrate_au']].head(2))
```

↗ Концентрация золота в смеси в отвальных хвостах после каждого этапа:

	rougher.output.tail_au	primary_cleaner.output.tail_au	secondary_cleaner.output.tail_au	final.output.tail_au
0	1.170244	2.106679	2.606185	2.143149
1	1.184827	2.353017	2.488248	2.224930

Концентрация золота в смеси после каждого этапа:

	rougher.input.feed_au	rougher.output.concentrate_au	primary_cleaner.output.concentrate_au	final.output.concentrate_au
0	6.486150	19.793808	34.174427	42.192020
1	6.478583	20.050975	34.118526	42.701629

По данным видим, что концентрация золота в отвальных хвостах увеличивается от этапа к этапу, значит в данных указывается не прибавка, а абсолютное число.

Уточним, как будет выглядеть формула для коэффициента обогащения для чернового концентрата:

Коэффициент обогащения для чернового концентрата:

$$\text{RECOVERY}_{\text{rougher}} = C1(F1-T1)/[F1(C1-T1)]$$

где:

- C1 = rougher.output.concentrate_au — доля золота в черновом концентрате;
- F1 = rougher.input.feed_au — доля золота в сырье/концентрате до флотации;
- T1 = rougher.output.tail_au — доля золота в отвальных хвостах после флотации.

Коэффициент обогащения для финального концентрата:

$$\text{RECOVERY}_{\text{final}} = C2(F2-T2)/[F2(C2-T2)]$$

где:

- C2 = final.output.concentrate_au — доля золота в концентрате после очистки, в финальном концентрате;
- F2=C1 = rougher.output.concentrate_au — доля золота в черновом концентрате;
- T2 = (final.output.tail_au - rougher.output.tail_au) — доля золота в финальных отвальных хвостах после очистки (за вычетом T1 доли золота в отвальных хвостах после флотации)

#расчитаем коэффициент обогащения для чернового концентрата

```
c1=df_train['rougher.output.concentrate_au']
f1 = df_train['rougher.input.feed_au']
t1 = df_train['rougher.output.tail_au']
```

```
recovery_rougher = c1*(f1-t1)*100/(f1*(c1-t1))
recovery_rougher.head()
```

↗

```
0    87.107763
1    86.843261
2    86.842308
3    87.226430
4    86.688794
dtype: float64
```

#расчитаем коэффициент обогащения для финального концентрата

```
t2 = df_train['final.output.tail_au'] - df_train['rougher.output.tail_au']
c2 = df_train['final.output.concentrate_au']
f2 = df_train['rougher.output.concentrate_au']
recovery_final = c2*(f2- t2)*100/(f2*(c2-t2))
recovery_final.head()
```

↗

```
0    97.329110
1    97.179758
2    96.939576
3    96.899102
4    96.936989
dtype: float64
```

Найдем MAE между расчётом и значением признака из таблицы данных.

MAE - метрика, которая сообщает нам среднюю абсолютную разницу между прогнозируемыми значениями и фактическими значениями в наборе данных. Чем ниже MAE, тем лучше модель соответствует набору данных.

$$MAE = \text{SUM}(\text{abs}(\text{rougher.output.recovery}[i] - \text{RECOVERY_rougher}[i])) / N$$

N= 14149

```
print('RECOVERY_rougher из предоставленных данных:')
print(df_train['rougher.output.recovery'].head())
print(recovery_rougher.head())

print('RECOVERY_rougher расчетные значения:')
mae_rougher = 0
for i in range(len(df_train)):
    mae_rougher += abs(df_train.loc[i, 'rougher.output.recovery'] - recovery_rougher[i])

mae_rougher = mae_rougher / len(df_train)
print('MAE:', mae_rougher, 'N =', len(df_train))
```

```
RECOVERY_rougher из предоставленных данных:
0    87.107763
1    86.843261
2    86.842308
3    87.226430
4    86.688794
Name: rougher.output.recovery, dtype: float64
0    87.107763
1    86.843261
2    86.842308
3    87.226430
4    86.688794
dtype: float64
RECOVERY_rougher расчетные значения:
MAE: 1.1131451184435918e-14 N = 14149
```

Выводы

- предоставленные значения рассчитанного коэффициента восстановления для флотации совпадают с рассчитанным значением.
- небольшое значение в MAE свидетельствует в пользу погрешности из-за округления данных

Комментарий ревьюера

Все отлично! 🙌: Оценили MAE между исходным и расчётным значением эффективности обогащения и убедились, что эффективность обогащения рассчитана правильно - отлично!

✓ Анализ параметров не указанных в тестовой выборке

В тестовой выборке 1 параметр - дата и 52 параметра В полных данных 1 - дата и 86 параметров В тестовой выборке не хватает 34 параметра

выведем названия столбцов, которых нет в тестовой выборке

```
temp = pd.DataFrame({'columns': df_full.columns})
print(temp.query('columns not in @df_test'))
print('Количество пропущенных параметров', temp.query('columns not in @df_test').count())
```

```
columns
1      final.output.concentrate_ag
2      final.output.concentrate_pb
3      final.output.concentrate_sol
4      final.output.concentrate_au
5      final.output.recovery
6      final.output.tail_ag
7      final.output.tail_pb
8      final.output.tail_sol
9      final.output.tail_au
14     primary_cleaner.output.concentrate_ag
15     primary_cleaner.output.concentrate_pb
16     primary_cleaner.output.concentrate_sol
17     primary_cleaner.output.concentrate_au
18     primary_cleaner.output.tail_ag
19     primary_cleaner.output.tail_pb
20     primary_cleaner.output.tail_sol
21     primary_cleaner.output.tail_au
30     rougher.calculation.sulfate_to_au_concentrate
```

```

31 rougher.calculation.floatbank10_sulfate_to_au_...
32 rougher.calculation.floatbank11_sulfate_to_au_...
33     rougher.calculation.au_pb_ratio
44     rougher.output.concentrate_ag
45     rougher.output.concentrate_pb
46     rougher.output.concentrate_sol
47     rougher.output.concentrate_au
48     rougher.output.recovery
49     rougher.output.tail_ag
50     rougher.output.tail_pb
51     rougher.output.tail_sol
52     rougher.output.tail_au
65     secondary_cleaner.output.tail_ag
66     secondary_cleaner.output.tail_pb
67     secondary_cleaner.output.tail_sol
68     secondary_cleaner.output.tail_au
Количество пропущенных параметров columns    34
dtype: int64

```

Выводы

В тестовой выборке пропущены параметры относящиеся к:

- параметрам продукта - output
 - rougher.output - продукты флотации, в том числе целевой признак
 - rougher.output.recovery
 - primary_cleaner.output - продукты первичной очистки
 - secondary_cleaner.output - продукты вторичной очистки
 - final.output - финальные характеристики продукта, в том числе целевой признак:
 - final.output.recovery
- 4 расчетных параметра для процесса флотации - rougher.calculation()
- параметры пропущенные в тестовой выборке, присутствуют в обучающей т.к. расчет их производился значительно позже.
- данные тестовой выборки можно дополнить из исходных данных соотнеся их по дате

Комментарий ревьюера

Все отлично! 🍌: Проанализировали разницу в признаках между выборками. Вывод верный: действительно, в тестовой выборке присутствуют только те параметры техпроцесса, которые мы можем получить непосредственно в ходе процесса или по его окончании.

✓ Предобработка данных

1. Обработка нулей
2. Обработка пропусков
3. Добавление параметров в тестовую выборку

✓ Обработка нулей

Выше при изучении данных были обнаружены нулевые значения в смесях в которых должно содержаться золото:

- final.output.concentrate_au : 98 строк - концентрация золота в финальном продукте
- final.output.recovery : 98 строк - коэффициент восстановления финальной смеси
- primary_cleaner.output.concentrate_au : 108 строк - концентрация золота в смеси после первичной очистки
- rougher.output.concentrate_au : 394 строк - концентрация золота в смеси после флотации
- rougher.output.recovery : 394 строк - коэффициент восстановления после флотации

посмотрим какие параметры содержат нули

параметры с содержанием золота не должны быть нулевые

```

for col in df_full:
    count_nul = df_full.loc[ df_full[col]==0, col].count()
    if count_nul != 0:
        print(col,':', count_nul)

```

```

🔗 final.output.concentrate_ag : 98
   final.output.concentrate_pb : 98
   final.output.concentrate_sol : 98
   final.output.concentrate_au : 98

```

```
final.output.recovery : 98
final.output.tail_ag : 91
final.output.tail_pb : 91
final.output.tail_sol : 91
final.output.tail_au : 91
primary_cleaner.input.depressant : 10
primary_cleaner.output.concentrate_ag : 108
primary_cleaner.output.concentrate_pb : 108
primary_cleaner.output.concentrate_sol : 108
primary_cleaner.output.concentrate_au : 108
primary_cleaner.output.tail_ag : 124
primary_cleaner.output.tail_pb : 124
primary_cleaner.output.tail_sol : 124
primary_cleaner.output.tail_au : 124
rougher.output.concentrate_ag : 394
rougher.output.concentrate_pb : 394
rougher.output.concentrate_sol : 394
rougher.output.concentrate_au : 394
rougher.output.recovery : 394
secondary_cleaner.output.tail_ag : 849
secondary_cleaner.output.tail_pb : 849
secondary_cleaner.output.tail_sol : 849
secondary_cleaner.output.tail_au : 849
secondary_cleaner.state.floatbank2_b_air : 8
secondary_cleaner.state.floatbank3_a_air : 2
secondary_cleaner.state.floatbank3_b_air : 1
secondary_cleaner.state.floatbank4_a_air : 1
secondary_cleaner.state.floatbank4_b_air : 1

#выведем строки с нулевыми значениями
df_full.loc[ df_full['final.output.concentrate_au'] == 0, ['rougher.input.feed_au',
                                                             'final.output.tail_au',
                                                             'rougher.output.concentrate_au',
                                                             'rougher.output.recovery',
                                                             'primary_cleaner.output.concentrate_au',
                                                             'secondary_cleaner.output.tail_au',
                                                             'final.output.concentrate_au',
                                                             'final.output.recovery']]
```

	rougher.input.feed_au	final.output.tail_au	rougher.output.concentrate_au	rougher.output.recovery	primary_cleaner.output.concentrate_au
644	5.598971	1.779052	15.935190	68.960993	
1193	7.786147	2.984504	18.682162	82.787322	
1194	7.499248	2.747294	18.231121	83.515262	
1195	7.032278	2.791440	18.190655	83.702224	
1196	7.021635	2.682127	18.194323	84.836821	
...
14951	11.604771	5.061847	19.442973	79.705229	
14952	11.763586	4.291648	17.784973	83.325755	
15183	7.345464	5.819537	20.047747	26.959908	
15184	7.687805	3.837274	22.632637	78.786772	
15310	6.009365	1.210944	13.743304	65.003857	

98 rows × 8 columns

Удивительные данные. В серье золото есть, в финальном концентрате золото есть, а на различных этапах оно вдруг пропадает. Похоже в данные промежутки времени были ошибки в измерениях. Следует сообщить об обнаруженной ошибке менеджерам, предоставившим данные

Так же мало похоже на правду что в начале могут быть нулевые значения добавляемых реагентов. Всего 10 таких строк, удалим и их.

```
# список наименований столбцов, где есть нулевые значения содержания золота
items = ['rougher.output.concentrate_au',
         'primary_cleaner.output.concentrate_au',
         'primary_cleaner.output.tail_au',
         'secondary_cleaner.output.tail_au',
         'final.output.concentrate_au',
         'final.output.tail_au',
         'rougher.output.recovery',
         'final.output.recovery',
         'primary_cleaner.input.depressant']
```

```
full = df_full.copy()
#удаляем нулевые строки везде где они есть для смесей содержащих золото
for i in items:
    full = full.drop(index = full.loc[ full[i]==0].index)
    print(i, len(df_full) - len(full))
```

```
rougher.output.concentrate_au 394
primary_cleaner.output.concentrate_au 497
primary_cleaner.output.tail_au 614
secondary_cleaner.output.tail_au 1403
final.output.concentrate_au 1481
final.output.tail_au 1535
rougher.output.recovery 1535
final.output.recovery 1535
primary_cleaner.input.depressant 1545
```

```
# проверим, что удалили нулевые значения
for i in items:
    count_nul = full.loc[ full[i]==0, col].count()
    print(i, ': ', count_nul)
```

```
rougher.output.concentrate_au : 0
primary_cleaner.output.concentrate_au : 0
primary_cleaner.output.tail_au : 0
secondary_cleaner.output.tail_au : 0
final.output.concentrate_au : 0
final.output.tail_au : 0
rougher.output.recovery : 0
final.output.recovery : 0
primary_cleaner.input.depressant : 0
```

```
# проверим в остальных данных остались ли нули
is_nul = 0
for col in full:
    count_nul = full.loc[ df_full[col]==0, col].count()
    if count_nul != 0:
        print(col, ': ', count_nul)
        is_nul=1
if (is_nul == 0):
    print('Нулей не осталось')
```

```
Нулей не осталось
```

```
# доля удаленных данных
print(f'Доля удаленных данных: {(len(df_full) - len(full))/len(df_full):.0%}' )
```

```
Доля удаленных данных: 8%
```

Выводы

- обнаружены ошибки в данных концентрации золота в смеси и концентрате. Нулевых значений быть не может, что свидетельствует о нарушении данных. Требуется установить причину этих нарушений. Это может быть как неверная работа датчиков, замеряющих значения, так и ошибки, возникшие при выгрузке данных. На данном этапе:
- удалены 1545 строк с нулевыми значениями золота в различных смесях, что составило 8% исходной выборки

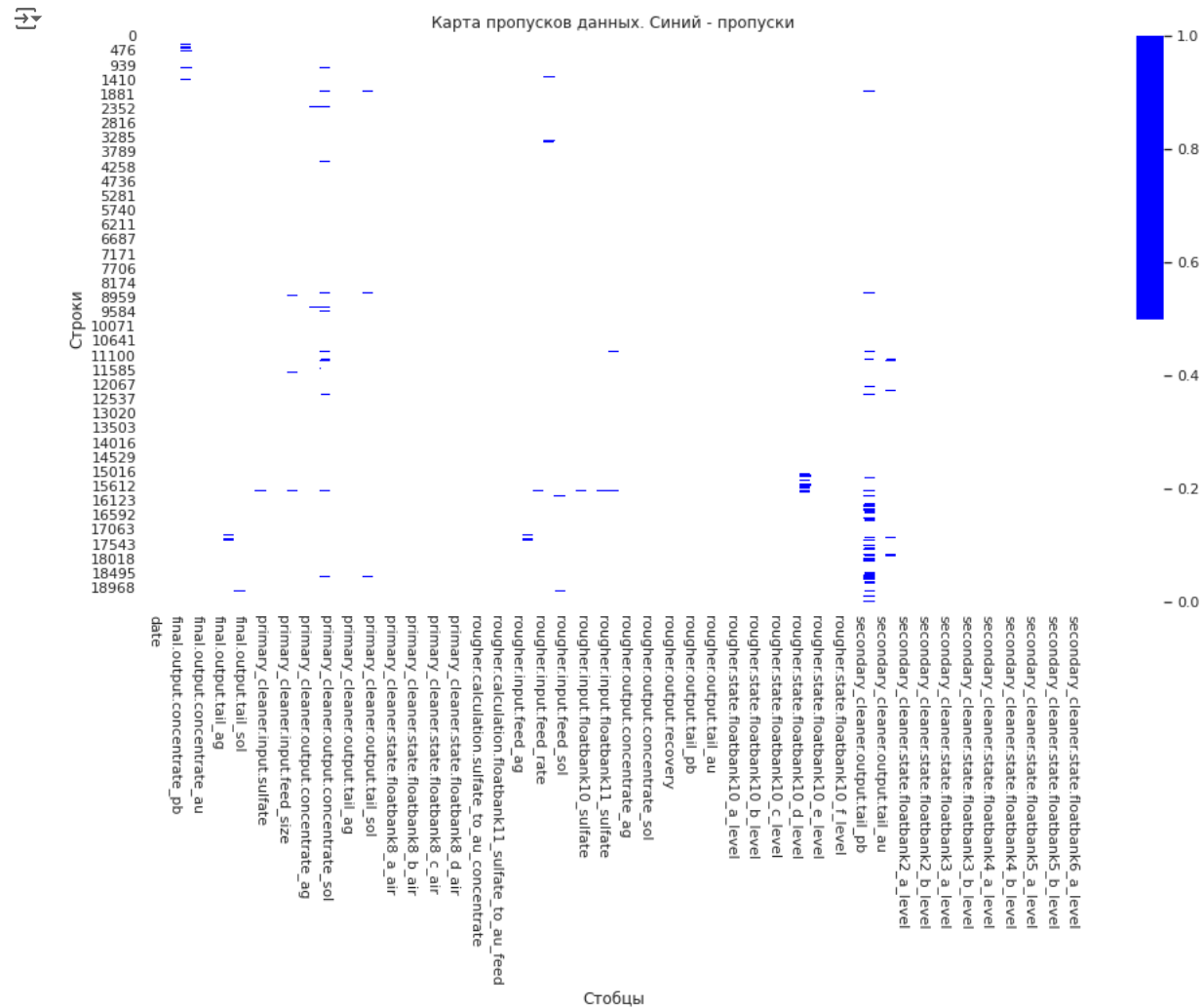
Комментарий ревьюера

Все отлично! 🍌: Удалены нулевые значения - отлично!

✓ Обработка пропусков

Еще раз взглянем на карту пропусков с учетом удаленных данных с нулями

```
#карта пропусков данных
cols = full.columns
# определяем цвета
# синий - пропущенные данные, белый - не пропущенные
colours = ['#ffffff', '#0000ff']
sns.set(rc = {'figure.figsize':(16,8)})
sns.heatmap(full[cols].isnull(), cmap=sns.color_palette(colours))
plt.title('Карта пропусков данных. Синий - пропуски')
plt.xticks(rotation=-90)
plt.xlabel('Столбцы')
plt.ylabel('Строки')
plt.grid()
plt.show()
```



```
#подсчитаем число пропусков в каждом столбце sum - сумма пропусков в столбце
temp = pd.DataFrame({'sum':full.isnull().sum().sort_values(ascending=False)})
print('Число столбцов с пропусками')
print(temp.loc[temp['sum']!=0].count())
temp.loc[temp['sum']!=0]
```

Число столбцов с пропусками
sum 62
dtype: int64

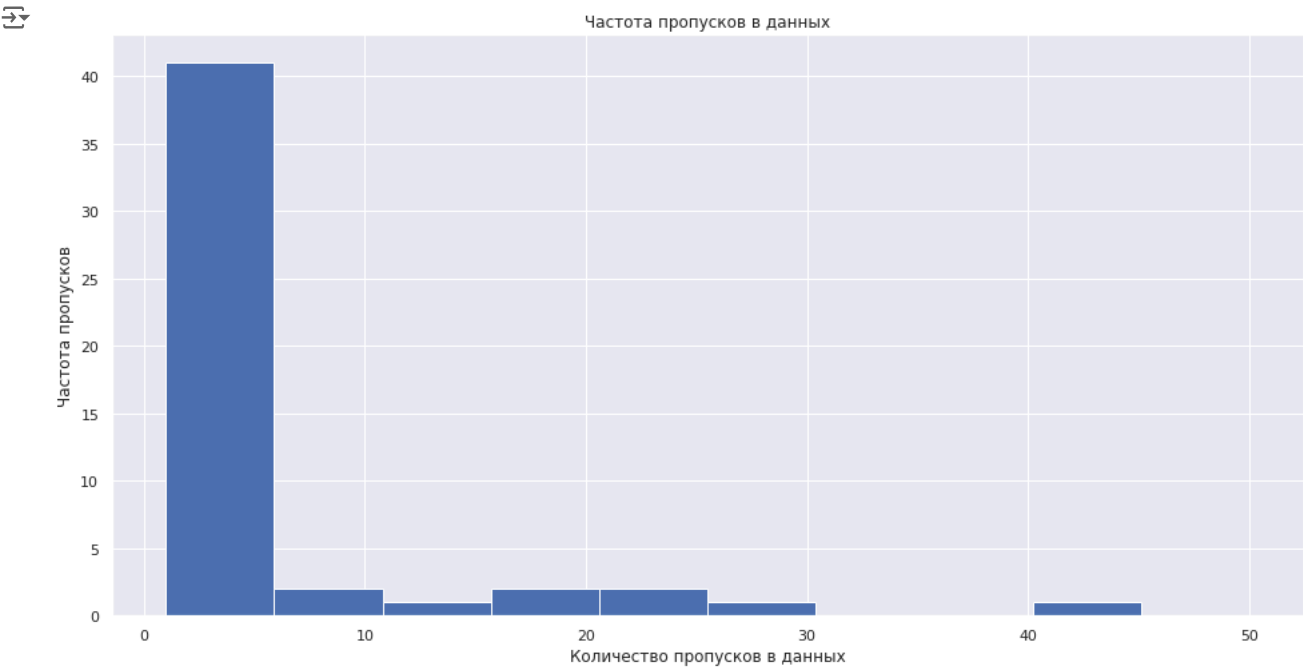
	sum
secondary_cleaner.output.tail_sol	1674
rougher.state.floatbank10_e_air	356
primary_cleaner.output.concentrate_sol	307
final.output.concentrate_sol	201
secondary_cleaner.state.floatbank2_a_air	200
...	...
rougher.input.floatbank10_xanthate	1
primary_cleaner.state.floatbank8_a_air	1
primary_cleaner.state.floatbank8_d_level	1
rougher.output.tail_ag	1
final.output.concentrate_pb	1

62 rows × 1 columns

Комментарий Ирины 1

Исправлен заголовок графика

```
temp.hist(bins=10,range=(1,50))
plt.xlabel('Количество пропусков в данных')
plt.ylabel('Частота пропусков')
plt.title('Частота пропусков в данных')
plt.show()
```



Комментарий ревьюера

На доработку ❌: Графику стоит дать более интерпретируемое название.

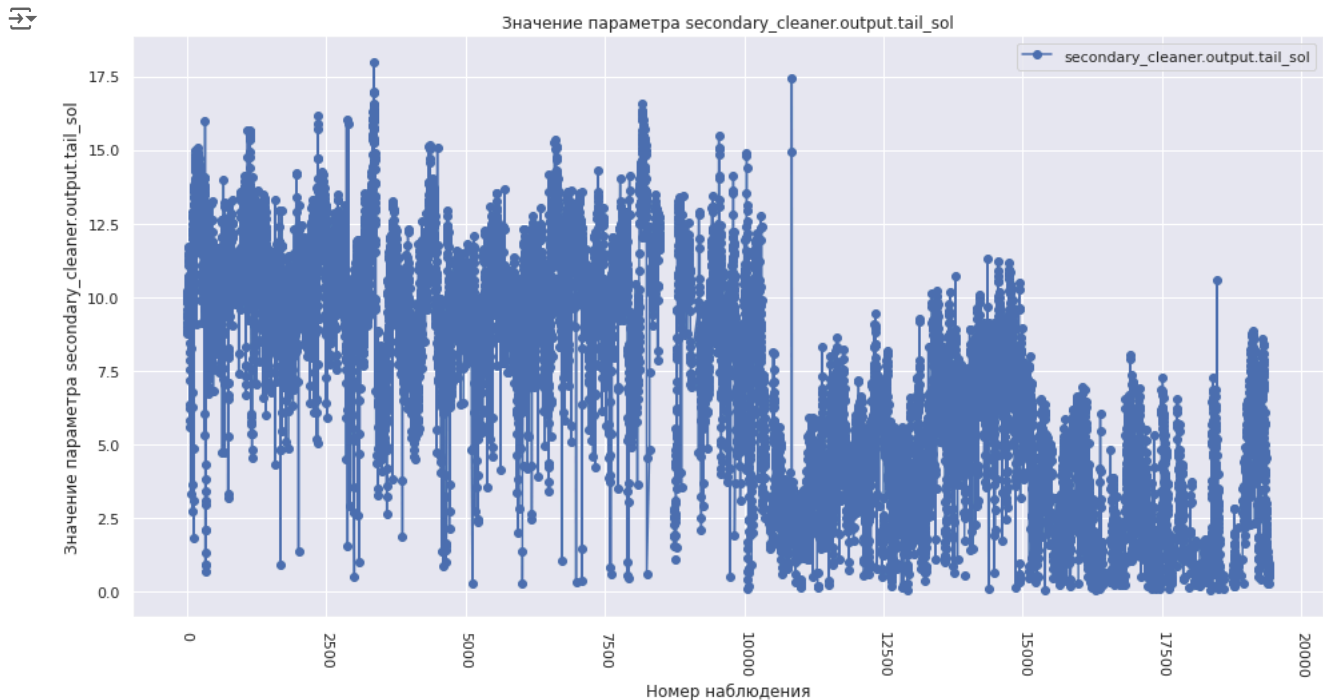
Комментарий Ирины 1

Исправлен заголовок графика

Комментарий ревьюера v.2

Все отлично! 🍌: Учтено.

```
full.plot(style='o-', y='secondary_cleaner.output.tail_sol', )
plt.xlabel('Номер наблюдения')
plt.ylabel('Значение параметра secondary_cleaner.output.tail_sol')
plt.xticks(rotation=-90)
plt.title('Значение параметра secondary_cleaner.output.tail_sol')
plt.show()
```



Комментарий ревьюера

На доработку ❌: Тут название нужно подписать.

Комментарий ревьюера v.2

Все отлично! 🍌: Учтено.

Характер данных имеет равномерный вид. На первый взгляд можно заполнить данные медианным значением. Но смущает что наблюдения после 10000 явно имеют другое медианное значение, чем наблюдения до 10000

Попробуем заполнить пропуски secondary_cleaner.output.tail_sol медианным значением и средним между предыдущим и последующим

```
#заполнение медианой
median_f = full['secondary_cleaner.output.tail_sol'].median()
full1 = full['secondary_cleaner.output.tail_sol'].fillna(median_f).copy()
full1.isnull().sum()
```

0

```
# попробуем заполнить предыдущим
fbfill_f = full['secondary_cleaner.output.tail_sol'].ffill()
full2 = full['secondary_cleaner.output.tail_sol'].fillna(fbbfill_f).copy()
full2.isnull().sum()
```

0

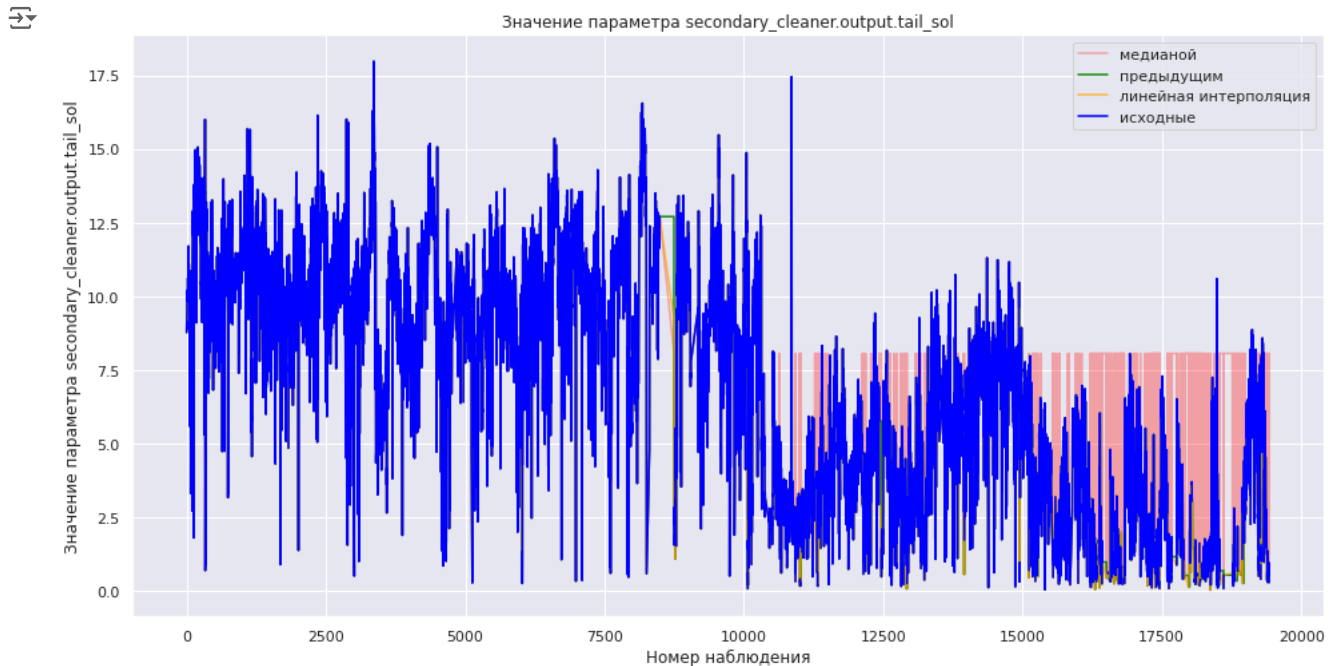
```
#линейная интерполяция
full3 = full['secondary_cleaner.output.tail_sol'].interpolate(method='linear', limit_direction='forward', axis=0).copy()
print(full['secondary_cleaner.output.tail_sol'].isnull().sum())
full3.isnull().sum()
```

1674
0

```

ax = full1.plot( y='secondary_cleaner.output.tail_sol', alpha=0.3, label = 'медианой',grid=True, legend=True, color='red')
full2.plot(y='secondary_cleaner.output.tail_sol', alpha=0.9, ax=ax, label = 'предыдущим',grid=True, legend=True, color='green')
full3.plot(y='secondary_cleaner.output.tail_sol', alpha=0.7, ax=ax, label = 'линейная интерполяция',grid=True, legend=True, color='orange')
full.plot(y='secondary_cleaner.output.tail_sol', figsize=(16,8), ax=ax, label = 'исходные',grid=True, legend=True, color='blue')
plt.xlabel('Номер наблюдения')
plt.ylabel('Значение параметра secondary_cleaner.output.tail_sol')
plt.title('Значение параметра secondary_cleaner.output.tail_sol')
plt.show()

```



Комментарий ревьюера

На доработку ❌: Тут название нужно подписать.

Комментарий ревьюера v.2

Все отлично! 🍌: Учтено.

Как видно из графика значения после 10000 наблюдения плохо описываются общей медианой и заполнение пропусков медианой искажает данные.

Заполним все пропуски линейной интерполяцией

```

full_without_nan = full.copy()
names = temp.loc[temp['sum']!=0].index
count = 0
for name in names:
    ffill_std = full[name].ffill()
    full_without_nan[name] = full_without_nan[name].interpolate(method='linear', limit_direction='forward', axis=0).copy()

```

Комментарий ревьюера

Некоторые замечания и рекомендации ⚠️: Замена на средние или медианы на всегда будет корректно работать, так как состояние одного и того же объекта в разное время может быть разным, и это не очень корректный шаг для данных с временной меткой.

Лучше использовать методы заполнения предыдущим или следующим значением (из условий проекта мы знаем, что соседние значения похожи) или использовать инструменты машинного обучения для заполнения пропусков из семейства `sklearn.impute`.

#подсчитаем число пропусков в каждом столбце sum - сумма пропусков в столбце

```

print('Проверим, что все столбцы не содержат пропуски')
print(full_without_nan.isnull().sum().sort_values(ascending = False))

```

Проверим, что все столбцы не содержат пропуски
date 0

```

rougher.state.floatbank10_b_air      0
rougher.state.floatbank10_f_air      0
rougher.state.floatbank10_e_level    0
rougher.state.floatbank10_e_air      0
..
primary_cleaner.state.floatbank8_c_level  0
primary_cleaner.state.floatbank8_c_air    0
primary_cleaner.state.floatbank8_b_level  0
primary_cleaner.state.floatbank8_b_air    0
secondary_cleaner.state.floatbank6_a_level 0
Length: 87, dtype: int64

```

Выводы

- больше всего пропусков у параметра secondary_cleaner.output.tail_sol = 1674 - 9% всех уже почищенных данных
- около 45 столбцов с не более 10 пропусками
- пропуски хорошо заполняются предыдущим значением и линейной интерполяцией

Комментарий ревьюера

Все отлично!👍: Пропуски обработаны.

✓ Изучение пропущенных в тестовой выборке данных

```

# выведем названия столбцов, которых нет в тестовой выборке
temp = pd.DataFrame({'columns': full_without_nan.columns})
print(temp.query('columns not in @df_test'))
print('Количество пропущенных параметров', temp.query('columns not in @df_test').count())

```

```

↔ columns
1      final.output.concentrate_ag
2      final.output.concentrate_pb
3      final.output.concentrate_sol
4      final.output.concentrate_au
5      final.output.recovery
6      final.output.tail_ag
7      final.output.tail_pb
8      final.output.tail_sol
9      final.output.tail_au
14     primary_cleaner.output.concentrate_ag
15     primary_cleaner.output.concentrate_pb
16     primary_cleaner.output.concentrate_sol
17     primary_cleaner.output.concentrate_au
18     primary_cleaner.output.tail_ag
19     primary_cleaner.output.tail_pb
20     primary_cleaner.output.tail_sol
21     primary_cleaner.output.tail_au
30     rougher.calculation.sulfate_to_au_concentrate
31     rougher.calculation.floatbank10_sulfate_to_au...
32     rougher.calculation.floatbank11_sulfate_to_au...
33     rougher.calculation.au_pb_ratio
44     rougher.output.concentrate_ag
45     rougher.output.concentrate_pb
46     rougher.output.concentrate_sol
47     rougher.output.concentrate_au
48     rougher.output.recovery
49     rougher.output.tail_ag
50     rougher.output.tail_pb
51     rougher.output.tail_sol
52     rougher.output.tail_au
65     secondary_cleaner.output.tail_ag
66     secondary_cleaner.output.tail_pb
67     secondary_cleaner.output.tail_sol
68     secondary_cleaner.output.tail_au
Количество пропущенных параметров columns    34
dtype: int64

```

```


# извлечем даты из тестовой выборки
test_date = pd.DataFrame({'date': df_test['date']})
test_date = test_date.set_index('date')
print(test_date.shape)
test_date.head()

```

 (5290, 0)

date
2016-09-01 00:59:59
2016-09-01 01:59:59
2016-09-01 02:59:59
2016-09-01 03:59:59
2016-09-01 04:59:59

```
#извлечем даты из обучающей выборки
train_date = pd.DataFrame({'date':df_train['date']})
train_date = train_date.set_index('date')
print(train_date.shape)
train_date.head()
```


 (14149, 0)

date
2016-01-15 00:00:00
2016-01-15 01:00:00
2016-01-15 02:00:00
2016-01-15 03:00:00
2016-01-15 04:00:00

Комментарий Ирины 1

Оставим в обучающих и тестовых данных только 52 параметра из тестовой выборки, т.к. остальные параметры нам на входе не доступны (они будут рассчитаны позднее). Так же оставим в них целевые признаки, выделим их в отдельные массивы данных позднее

```
# установим в качестве индексов - дату для очищенных данных
full_clearn = full_without_nan.copy() # очищенные данные со всеми параметрами
full_clearn_input = full_clearn[df_test.columns].copy() # очищенные даынные только с параметрами доступными на входе
full_clearn_input['rougher.output.recovery'] = full_clearn['rougher.output.recovery'].copy()#восстановим целевой признак
full_clearn_input['final.output.recovery'] = full_clearn['final.output.recovery'].copy()#восстановим целевой признак
full_clearn = full_clearn.set_index('date')
full_clearn_input = full_clearn_input.set_index('date')
print('full_clearn shape:', full_clearn.shape)
print('full_clearn_input shape:', full_clearn_input.shape)
full_clearn.head()
```

 full_clearn shape: (17894, 86)
full_clearn_input shape: (17894, 54)

	final.output.concentrate_ag	final.output.concentrate_pb	final.output.concentrate_sol	final.output.concentrate_au	final
date					
2016-01-15 00:00:00	6.055403	9.889648	5.507324	42.192020	
2016-01-15 01:00:00	6.029369	9.968944	5.257781	42.701629	
2016-01-15 02:00:00	6.055926	10.213995	5.383759	42.657501	
2016-01-15 03:00:00	6.047977	9.977019	4.858634	42.689819	
2016-01-15 04:00:00	6.148599	10.142511	4.939416	42.774141	

5 rows × 86 columns

```
#очищенные тестовые данные + целевые признаки
test = full_clearn_input.join(test_date, on='date', how='inner', lsuffix='', rsuffix='_test')
print(test.shape)
test.head()
```

↗ (4958, 54)

	primary_cleaner.input.sulfate	primary_cleaner.input.depressant	primary_cleaner.input.feed_size	primary_cleaner.input.xa
date				
2016-09-01 00:59:59	210.800909	14.993118	8.080000	1.
2016-09-01 01:59:59	215.392455	14.987471	8.080000	0.
2016-09-01 02:59:59	215.259946	12.884934	7.786667	0.
2016-09-01 03:59:59	215.336236	12.006805	7.640000	0.
2016-09-01 04:59:59	199.099327	10.682530	7.530000	0.

5 rows × 54 columns

Комментарий ревьюера

На доработку ❌: С учётом ранее сделанных выводов о доступности некоторых признаков:

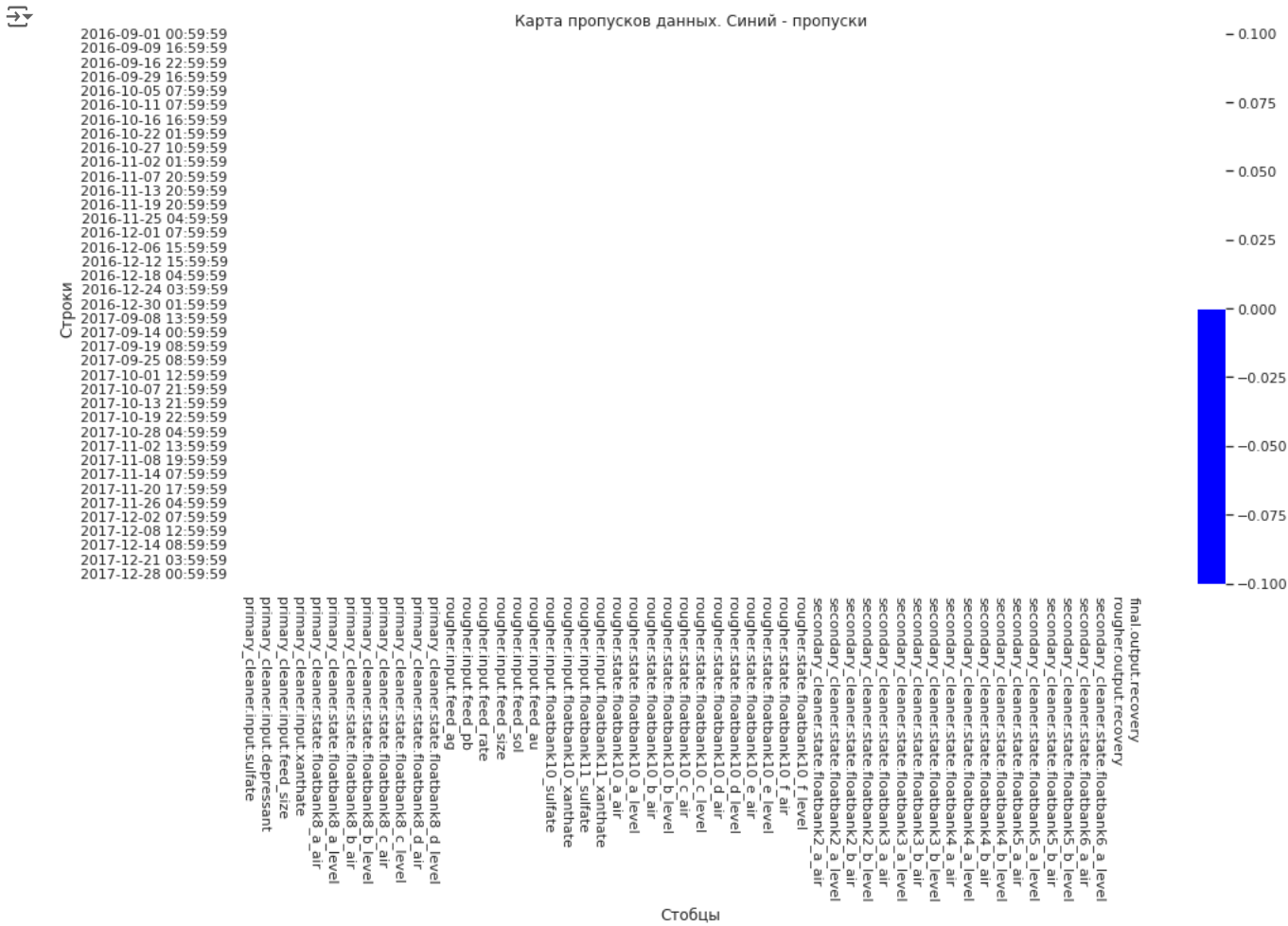
параметры пропущенные в тестовой выборке, присутствуют в обучающей т.к. расчет их производился значительно позже.

как мы будем воспроизводить эти признаки в начале техпроцесса для получения прогноза, если мы узнаем их только тогда, когда процесс будет завершён?

Комментарий ревьюера v.2

Все отлично! 👍: Учтено.

```
#карта пропусков данных
cols = test.columns
# определяем цвета
# синий - пропущенные данные, белый - не пропущенные
colours = ['#0000ff', '#ffffff']
sns.set(rc = {'figure.figsize':(16,8)})
sns.heatmap(test[cols].isnull(), cmap=sns.color_palette(colours))
plt.title('Карта пропусков данных. Синий - пропуски')
plt.xticks(rotation=-90)
plt.xlabel('Столбцы')
plt.ylabel('Строки')
plt.grid()
plt.show()
```



```
#очищенные обучающие данные + 2 целевых признака
# в обучающей выборке так же оставим только 52 параметра из тестовой выборки
train = full_clearn_input.join(train_date, on='date', how='inner', lsuffix='', rsuffix='_train')
print(train.shape)
test.head()
```

(12936, 54)

	primary_cleaner.input.sulfate	primary_cleaner.input.depressant	primary_cleaner.input.feed_size	primary_cleaner.input.xa
date				
2016-09-01 00:59:59	210.800909	14.993118	8.080000	1.
2016-09-01 01:59:59	215.392455	14.987471	8.080000	0.
2016-09-01 02:59:59	215.259946	12.884934	7.786667	0.
2016-09-01 03:59:59	215.336236	12.006805	7.640000	0.
2016-09-01 04:59:59	199.099327	10.682530	7.530000	0.

5 rows × 54 columns

Выводы

- в обучающей выборке оставленные 52 параметра, которые можно получить в начале технологического процесса
- размер исходных очищенных данных изменился с (19439,87) до (17894,87)
- размер полученной тестовой выборки изменился с (5290, 53) до (4958, 52)
- размер обучающей выборки изменился с (14149, 87) до (12936, 52)
- параметр даты стал использоваться в качестве индекса

Выводы по разделу подготовка данных

- Изучение данных
 - Размер исходных данных: (19439; 87)
 - Размер обучающей выборки: (14149; 87)
 - Размер тестовой выборки: (5290; 53) - не содержит целевой признак
 - В данных наблюдаются пропуски
 - Количество колонок в тестовой выборке не соответствует количеству колонок в полных данных
 - Наблюдаются нулевые данные для золотосодержащего концентрата, что свидетельствует об ошибках в данных
- Обработка аномалий:
 - обнаружены ошибки в данных концентрации золота в смеси и концентрате. Нулевых значений быть не может, что свидетельствует о нарушении данных. Требуется установить причину этих нарушений. Это может быть как неверная работа датчиков, измеряющих значения, так и ошибки, возникшие при выгрузке данных. На данном этапе:
 - удалены 1545 строк с нулевыми значениями золота в различных смесях, что составило 8% исходной выборки
- Обработка пропусков:
 - больше всего пропусков у параметра `secondary_cleaner.output.tail_sol` = 1674 - 9% всех уже почищенных данных
 - около 45 столбцов с не более 10 пропусками
 - пропуски хорошо заполняются линейной интерполяцией
- Добавление параметров:
 - в обучающей выборке оставлены 52 параметра, которые можно получить в начале технологического процесса
 - размер исходных очищенных данных изменился с (19439,87) до (17894,87)
 - размер полученной тестовой выборки изменился с (5290, 53) до (4958, 52)
 - размер обучающей выборки изменился с (14149, 87) до (12936, 52)
 - параметр даты стал использоваться в качестве индекса

Комментарий ревьюера

Все отлично! 🍌: Молодец, что не забываешь делать промежуточные выводы.

✓ Анализ данных

✓ Концентрации металлов на различных этапах очистки

Посмотрим, как меняется концентрация металлов (Au, Ag, Pb) на различных этапах очистки.

```

"""
функция получает исходные данные и название металлов
возвращает, датафрейм со следующими столбцами:
    название стадии,
    название столбца из исходных данных
    номер последовательности стадии
    класс данных: c - концентрат, t - хвосты
    медианное значение концентрации металла на каждой стадии
"""
def get_metal_info( data, name):
    col = data.columns
    info = []
    if name == 'Золото': name = '_au'
    elif name == 'Серебро': name = '_ag'
    elif name == 'Свинец': name = '_pb'
    for c in col:
        if ('rougher.input.feed'+name in c):
            info.append(['исходное сырье',c, 1, 'c', data[c].median()])
        elif ('rougher.output.concentrate'+name in c ):
            info.append(['после флотации',c,2, 'c', data[c].median()])
        elif ('primary_cleaner.output.concentrate'+name in c ):
            info.append(['после 1 очистки',c, 3, 'c', data[c].median()])
        elif ('final.output.concentrate'+name in c ):
            info.append(['финальный концентрат',c, 4, 'c', data[c].median()])
        elif ('rougher.output.tail'+name in c ):
            info.append(['хвосты после флотации',c, 5, 't', data[c].median()])
        elif ('primary_cleaner.output.tail'+name in c ):
            info.append(['хвосты после 1 очистки',c, 6, 't', data[c].median()])
        elif ('secondary_cleaner.output.tail'+name in c ):
            info.append(['хвосты после 2 очистки',c, 7, 't', data[c].median()])
        elif ('final.output.tail'+name in c ):
            info.append(['финальные хвосты',c, 8, 't', data[c].median()])
    metal_info = pd.DataFrame(info)
    metal_info.columns = [ 'name','column_name', 'stage','class', 'median']
    metal_info = metal_info.sort_values(by='stage', ascending = True).reset_index(drop='True')
    return metal_info

```

```
display(get_metal_info (full_clearn, 'Золото'))
```

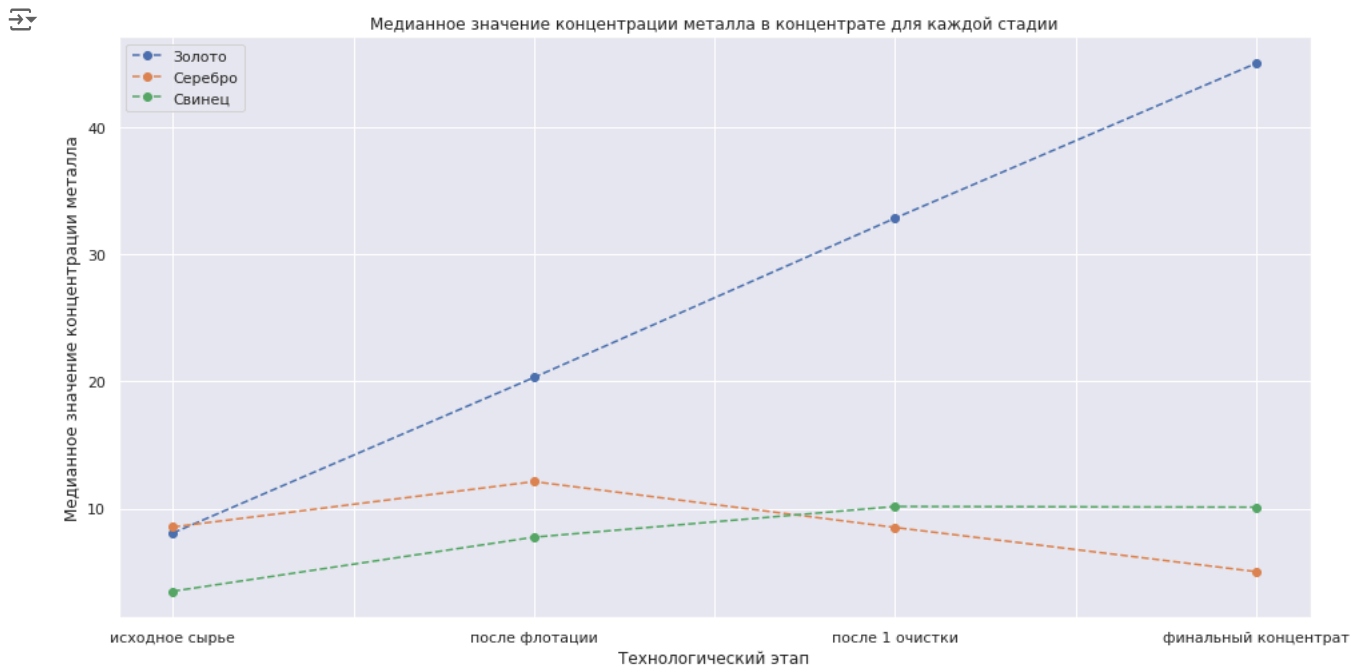


	name	column_name	stage	class	median
0	исходное сырье	rougher.input.feed_au	1	c	8.086987
1	после флотации	rougher.output.concentrate_au	2	c	20.314645
2	после 1 очистки	primary_cleaner.output.concentrate_au	3	c	32.837665
3	финальный концентрат	final.output.concentrate_au	4	c	45.009343
4	хвосты после флотации	rougher.output.tail_au	5	t	1.808189
5	хвосты после 1 очистки	primary_cleaner.output.tail_au	6	t	3.595376
6	хвосты после 2 очистки	secondary_cleaner.output.tail_au	7	t	4.129036
7	финальные хвосты	final.output.tail_au	8	t	2.958955

```

#медианные значения концентрации для концентрата
names = ['Золото', 'Серебро', 'Свинец']
fig,ax = plt.subplots()
for n in names:
    get_metal_info(full_clearn, n)\
    .loc[get_metal_info(full_clearn, n)['class'] == 'c']\
    .plot(x= 'name', y='median', style='o--', ax=ax, label = n)
plt.title('Медианное значение концентрации металла в концентрате для каждой стадии')
plt.xlabel('Технологический этап')
plt.ylabel('Медианное значение концентрации металла')
plt.show()

```

Комментарий ревьюера

На доработку ❌: Тут и далее не подписаны оси Y.

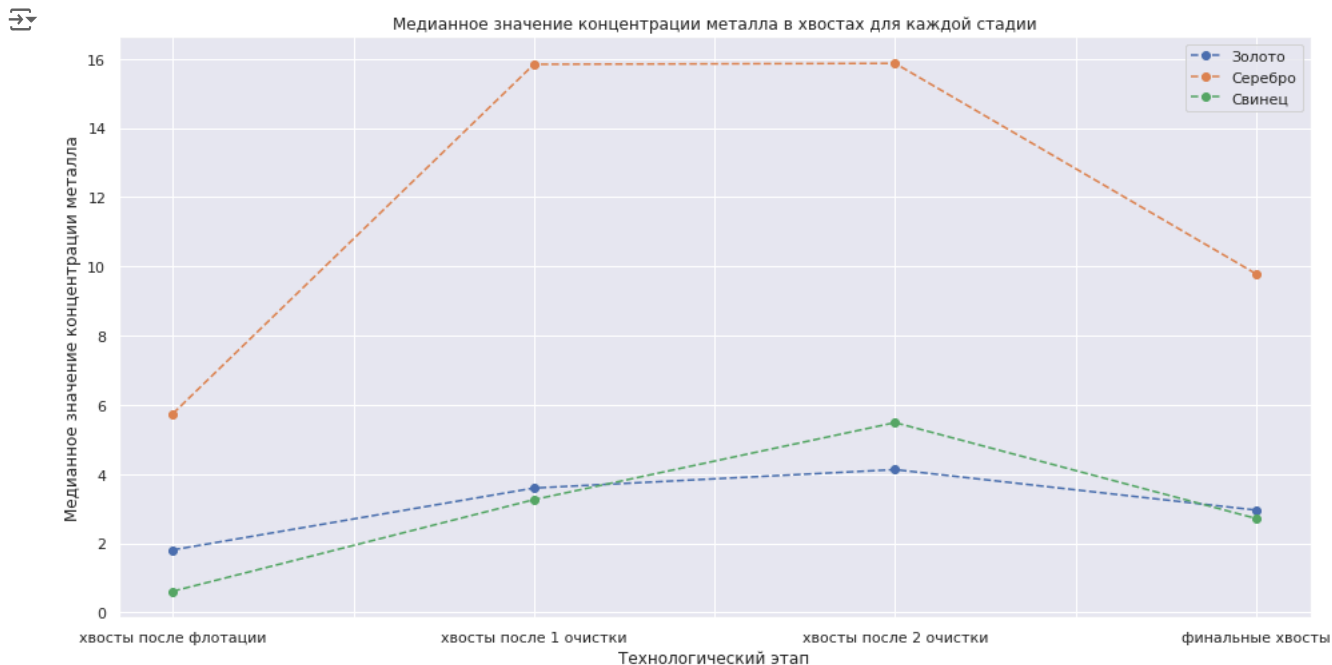
Комментарий ревьюера v.2

Все отлично! 👍: Учтено.

Выводы

- медианное значение концентрации золота в концентрате от стадии к стадии растёт
- медианное значение концентрации свинца в концентрате растёт
- медианное значение концентрации серебра после флотации возрастает, после очистки снижается

```
#медианные значения концентрации металлов для хвостов
names = ['Золото', 'Серебро', 'Свинец']
fig, ax = plt.subplots()
for n in names:
    get_metal_info(full_clearn, n)\
        .loc[get_metal_info(full_clearn, n)['class'] == 't']\
        .plot(x='name', y='median', style='o--', ax=ax, label=n)
plt.title('Медианное значение концентрации металла в хвостах для каждой стадии')
plt.xlabel('Технологический этап')
plt.ylabel('Медианное значение концентрации металла')
plt.show()
```

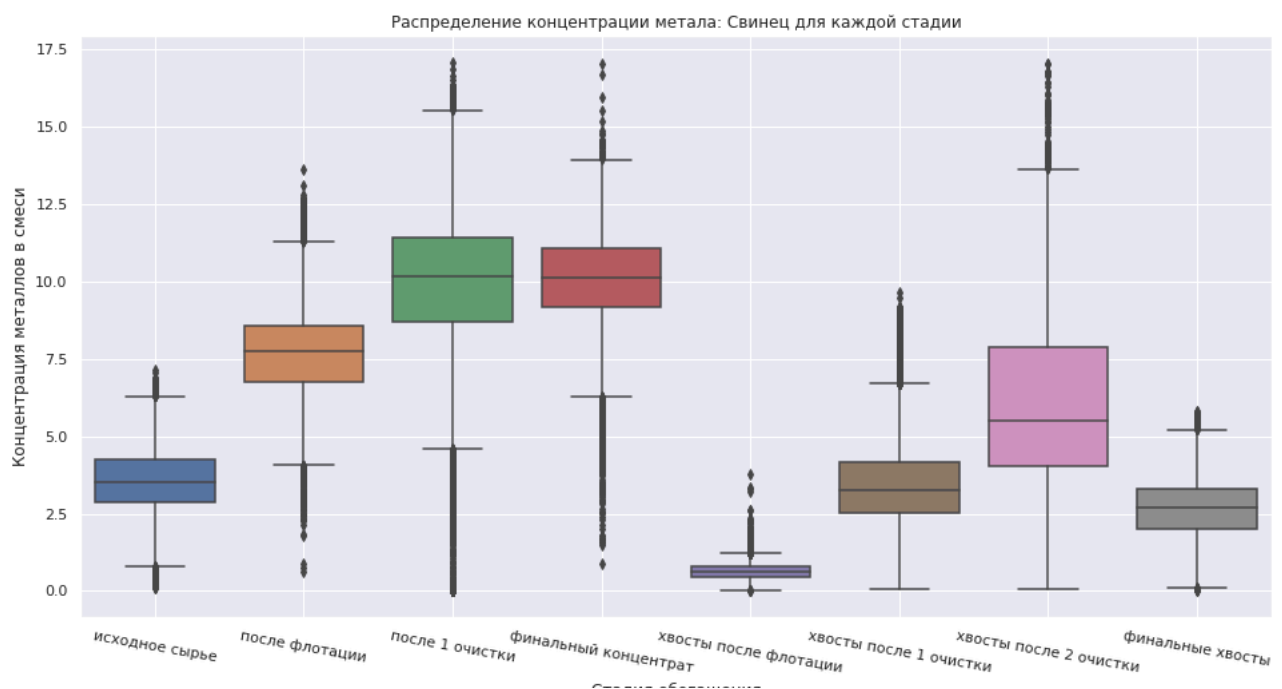
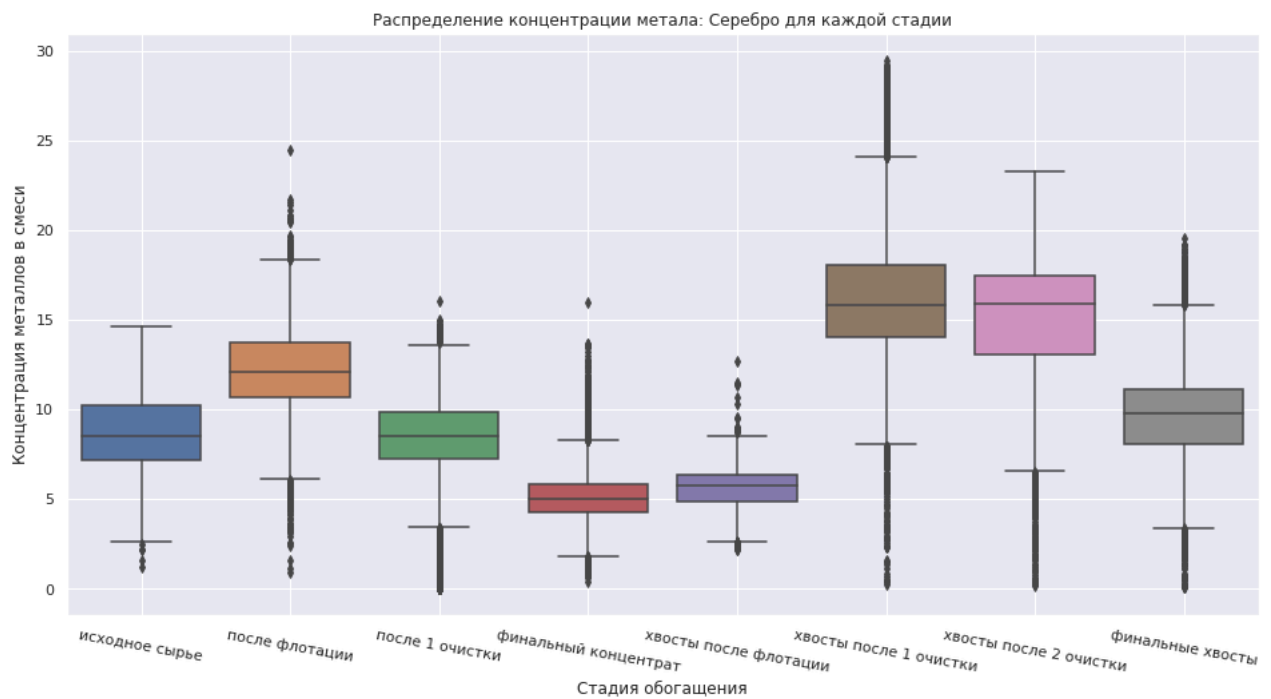
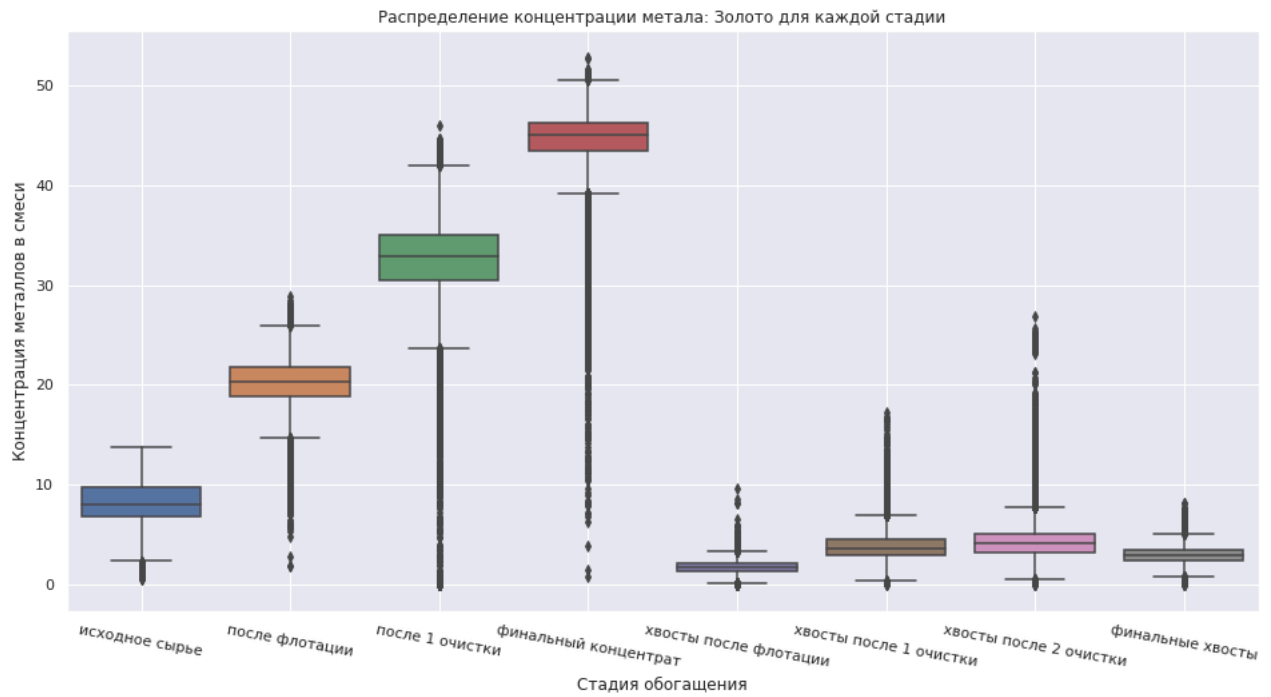


Выводы

- медианное значение концентрации золота в хвостах увеличивается от стадии к стадии
- медианное значение концентрации золота в финальных хвостах чуть ниже чем на стадии очистки, что можно объяснить более существенными выбросами в измерениях на стадии очистки
- медианное значение концентрации свинца в хвостах растет в процессе очистки, но в финальных хвостах снижается
- медианное значение концентрации серебра в хвостах после флотации возрастает, затем в последующих стадиях после очистки снижается

```
names = ['Золото', 'Серебро', 'Свинец']
fig, ax = plt.subplots()

for n in names:
    metals_info = get_metal_info(full_clearn, n)
    columns = metals_info['column_name']
    x_name = metals_info['name']
    #чтобы переименовать подписи графиков
    data = full_clearn[columns].copy()
    data.columns = x_name
    sns.boxplot(data = data)
    plt.xticks(rotation=-10)
    plt.title(f'Распределение концентрации металла: {n} для каждой стадии')
    plt.ylabel('Концентрация металлов в смеси')
    plt.xlabel('Стадия обогащения')
    plt.grid(True)
    plt.show()
```



Выводы

Концетрат:

- медианное значение концентрации золота в концетрате от стадии к стадии растёт
- медианное значение концентрации свинца в концетрате растёт
- медианное значение концентрации серебра после флотации возрастает, после очистки снижается

Хвосты:

- медианное значение концентрации золота в хвостах увеличивается от стадии к стадии
- медианное значение концетрации золота в финальных хвостах чуть ниже чем на стадии очистки, что можно объяснить более существенными выбросами в измерениях на стадии очистки
- медианное значение концентрации свинца в хвостах растёт в процессе очистки, но в финальных хвостах снижается
- медианное значение концентрации серебра в хвостах после флотации возрастает, затем в оследующих стадиях после очистки снижается

Комментарий ревьюера

Все отлично! 🙌: Исследована концентрация металлов на разных стадиях обработки, проанализирована динамика концентрации в зависимости от этапа техпроцесса - отлично, тут всё верно.

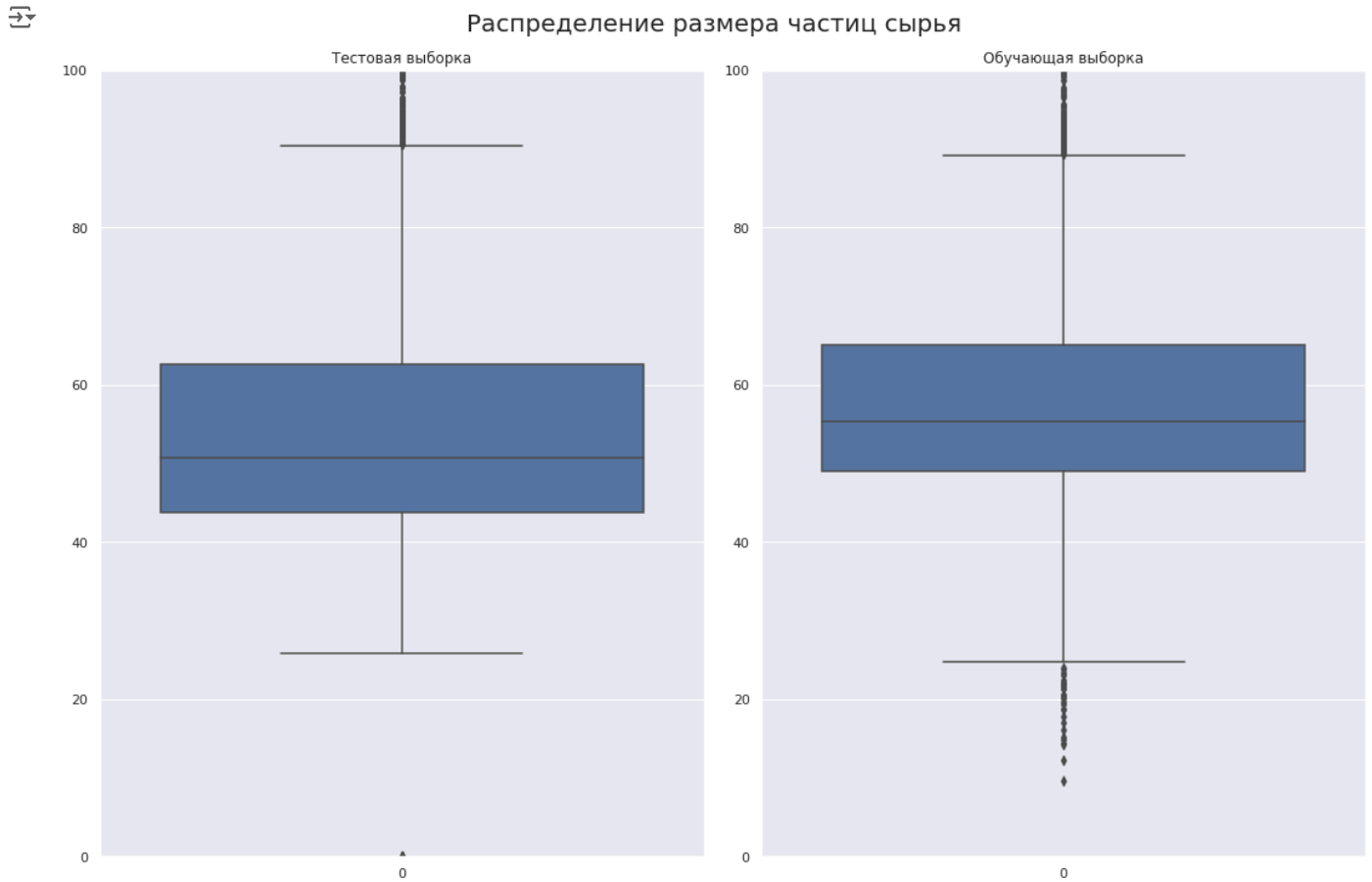
✓ Распределение размеров гранул сырья в выборках

Сравним распределения размеров гранул сырья на обучающей и тестовой выборках.

Если распределения сильно отличаются друг от друга, оценка модели будет неправильной.

```
#сетка
fig = plt.figure(figsize=(15, 10), constrained_layout=True)
gs = gridspec.GridSpec(ncols=2, nrows=1, figure=fig)
#первый график в сетке
fig_ax_1 = fg.add_subplot(gs[0, 0])
sns.boxplot(data = test['rougher.input.feed_size'])
plt.ylim(0, 100)
plt.title('Тестовая выборка')
#второй график в сетке
fig_ax_2 = fg.add_subplot(gs[0, 1])
sns.boxplot(data = train['rougher.input.feed_size'])
plt.ylim(0, 100)
plt.title('Обучающая выборка')

plt.suptitle('Распределение размера частиц сырья', fontsize=20)
plt.show()
```



```
display(test['rougher.input.feed_size'].describe())
train['rougher.input.feed_size'].describe()
```

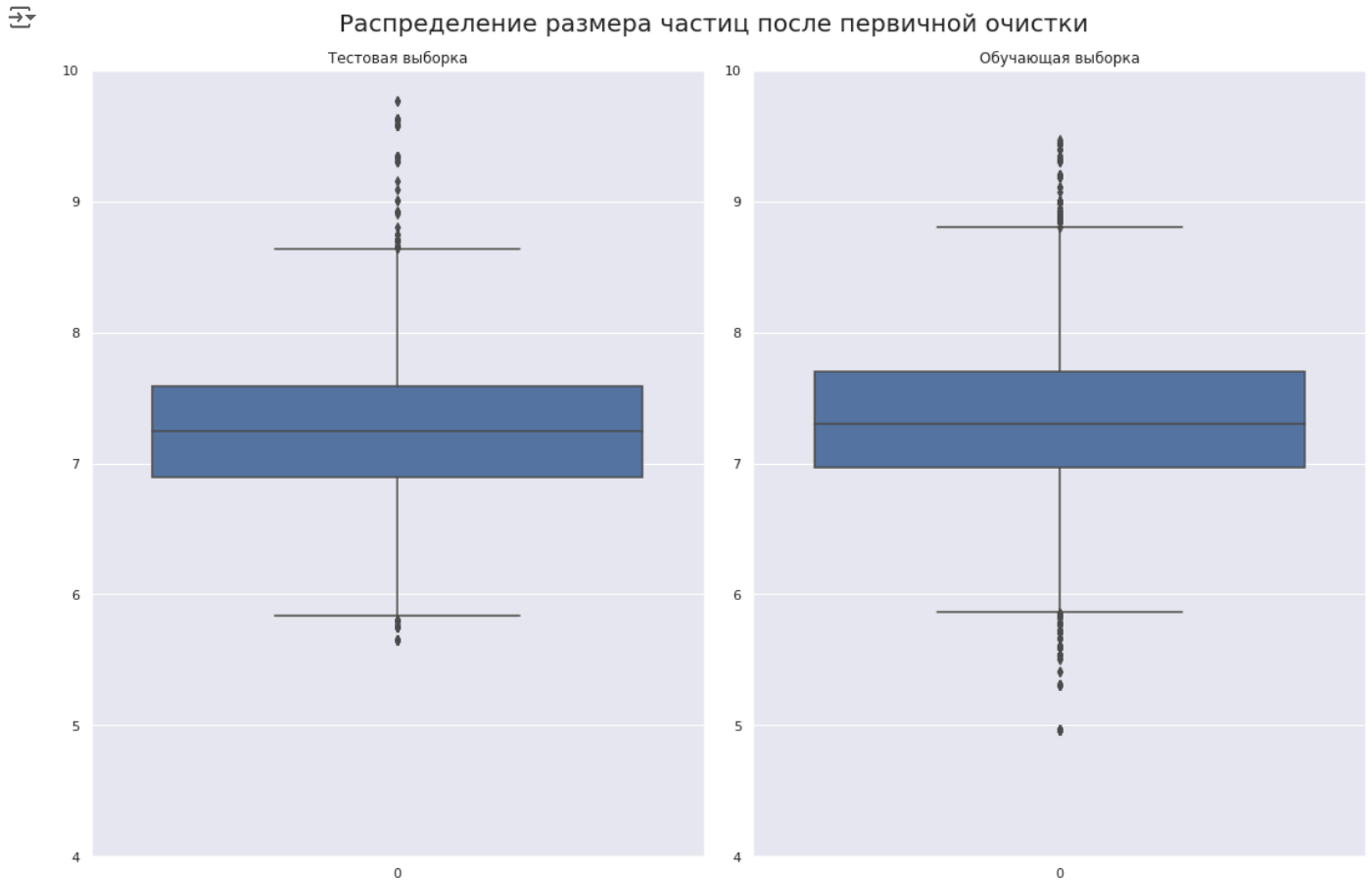
```
count    4958.000000
mean      56.003990
std       19.309900
min        0.046369
25%       43.836458
50%       50.691959
75%       62.527815
max       392.494040
Name: rougher.input.feed_size, dtype: float64
count    12936.000000
mean      59.424724
std       22.285530
min        9.659576
25%       48.982282
50%       55.277053
75%       65.135448
max       484.967466
Name: rougher.input.feed_size, dtype: float64
```

Комментарий Ирины 1

Распределение размера гранул на этапе первичной очистки

```
#сетка
fg = plt.figure(figsize=(15, 10), constrained_layout=True)
gs = gridspec.GridSpec(ncols=2, nrows=1, figure=fg)
#первый график в сетке
fig_ax_1 = fg.add_subplot(gs[0, 0])
sns.boxplot(data = test['primary_cleaner.input.feed_size'])
plt.ylim(4, 10)
plt.title('Тестовая выборка')
#второй график в сетке
fig_ax_2 = fg.add_subplot(gs[0, 1])
sns.boxplot(data = train['primary_cleaner.input.feed_size'])
plt.ylim(4, 10)
plt.title('Обучающая выборка')

plt.suptitle('Распределение размера частиц после первичной очистки', fontsize=20)
plt.show()
```



```
display(test['primary_cleaner.input.feed_size'].describe())
train['primary_cleaner.input.feed_size'].describe()
```

```
count    4958.000000
mean      7.259991
std       0.606376
min       5.650000
25%      6.890000
50%      7.250000
75%      7.593333
max      15.500000
Name: primary_cleaner.input.feed_size, dtype: float64
count    12936.000000
mean      7.329537
std       0.611677
min       1.080000
25%      6.965000
50%      7.300000
75%      7.700000
max      10.470000
Name: primary_cleaner.input.feed_size, dtype: float64
```

Выводы

- Размер гранул сырья:

- средний размер гранул сырья тестовой выборки 56, 75% гранул имеют размер в диапазоне [43;62]
- средний размер гранул сырья обучающей выборки 59, 75% гранул имеют размер в диапазоне [48;65]
- распределения размеров гранул сырья тестовой и обучающей выборки не имеют сильного различия
- Размер гранул после первичной очистки:
 - средний размер гранул в тестовой выборке 7,2 в диапазоне [6.8; 7.6]
 - средний размер гранул в обучающей выборке 7,3 в диапазоне [7,0; 7.7]
 - распределения размеров гранул после первичной очистки в тестовой и обучающей выборках не имеют сильного различия

Комментарий ревьюера

Все отлично! 🍌: Сравнили размеры гранул сырья на обучающей и тестовой выборках для этапа `rougher.input.feed_size` - отлично, распределения между выборками действительно схожи, следовательно модель будет корректно работать.

Чуть более интересным решением было бы использование статистического теста (например, `ttest`) для сравнения распределений в выборках.

Комментарий ревьюера

На доработку ❌: Нет анализа размера гранул сырья на этапе первичной очистки.

Комментарий ревьюера v.2

Все отлично! 🍌: Учтено.

✓ Суммарная концентрация веществ на разных стадиях

Исследуйте суммарную концентрацию всех веществ на разных стадиях: в сырье, в черновом и финальном концентратах. 1) исходное сырье

- `'rougher.input.feed_ag'`,
- `'rougher.input.feed_pb'`,
- `'rougher.input.feed_sol'`,
- `'rougher.input.feed_au'`,

2) после флотации

- `'rougher.output.concentrate_ag'`,
- `'rougher.output.concentrate_pb'`,
- `'rougher.output.concentrate_sol'`,
- `'rougher.output.concentrate_au'`,

3) после очистки

- `'primary_cleaner.output.concentrate_ag'`,
- `'primary_cleaner.output.concentrate_pb'`,
- `'primary_cleaner.output.concentrate_sol'`,
- `'primary_cleaner.output.concentrate_au'`,

4) финальные

- `'final.output.concentrate_ag'`,
- `'final.output.concentrate_pb'`,
- `'final.output.concentrate_sol'`,
- `'final.output.concentrate_au'`,

```
#добавим в данные столбцы с суммарными концентрациями веществ на каждой стадии
#если разкомментировать '_sol' то концентрация вещества в расчетах не будет учитываться
data= full_clearn.copy()
# заголовки стадий
stages = ['rougher.input.feed', 'rougher.output.concentrate', 'primary_cleaner.output.concentrate', 'final.output.concentrate' ]
col = data.columns
columns_stage = []
columns_sum = []
#пройдемся по всем стадиям
for s in stages:
    # создадим новую колонку с суммарным значением для текущей стадии
    name = s+'_sum'
    data[name] = 0
    #пройдемся по всем колонкам и найдем все колонки относящиеся к текущей стадии
    for c in col:
        if (s in c)&('_size' not in c)&('_rate' not in c): #&('_sol' not in c):- можно разкомментировать и перестроить графики
            columns_stage.append(c)
            # занесем сумму по всем колонкам соответствующим текущей стадии в специальную колонку
            data[name] +=data[c]
    #columns_stage.append(name)
    columns_sum.append(name)
#выведем столбцы и отдельно суммы по ним, чтобы проверить правильность расчетов
display(data[columns_stage].head())

data_new = data[columns_sum].copy()
display(data_new.head())
data_new.columns = ['исходное', 'флотация', 'очистка', 'финальное']
display(data_new.head())
```

	rougher.input.feed_ag	rougher.input.feed_pb	rougher.input.feed_sol	rougher.input.feed_au	rougher.output.concentrate_ag
date					
2016-01-15 00:00:00	6.100378	2.284912	36.808594	6.486150	11.500771
2016-01-15 01:00:00	6.161113	2.266033	35.753385	6.478583	11.615865
2016-01-15 02:00:00	6.116455	2.159622	35.971630	6.362222	11.695753
2016-01-15 03:00:00	6.043309	2.037807	36.862241	6.118189	11.915047
2016-01-15 04:00:00	6.060915	1.786875	34.347666	5.663707	12.411054

	rougher.input.feed_sum	rougher.output.concentrate_sum	primary_cleaner.output.concentrate_sum	final.output.concentrate_s
date				
2016-01-15 00:00:00	51.680034	66.424950		63.6443
2016-01-15 01:00:00	50.659114	67.012710		63.9577
2016-01-15 02:00:00	50.609929	66.103793		64.3111
2016-01-15 03:00:00	51.061546	65.752751		63.5734
2016-01-15 04:00:00	47.859163	65.908382		64.0046

исходное

флотация

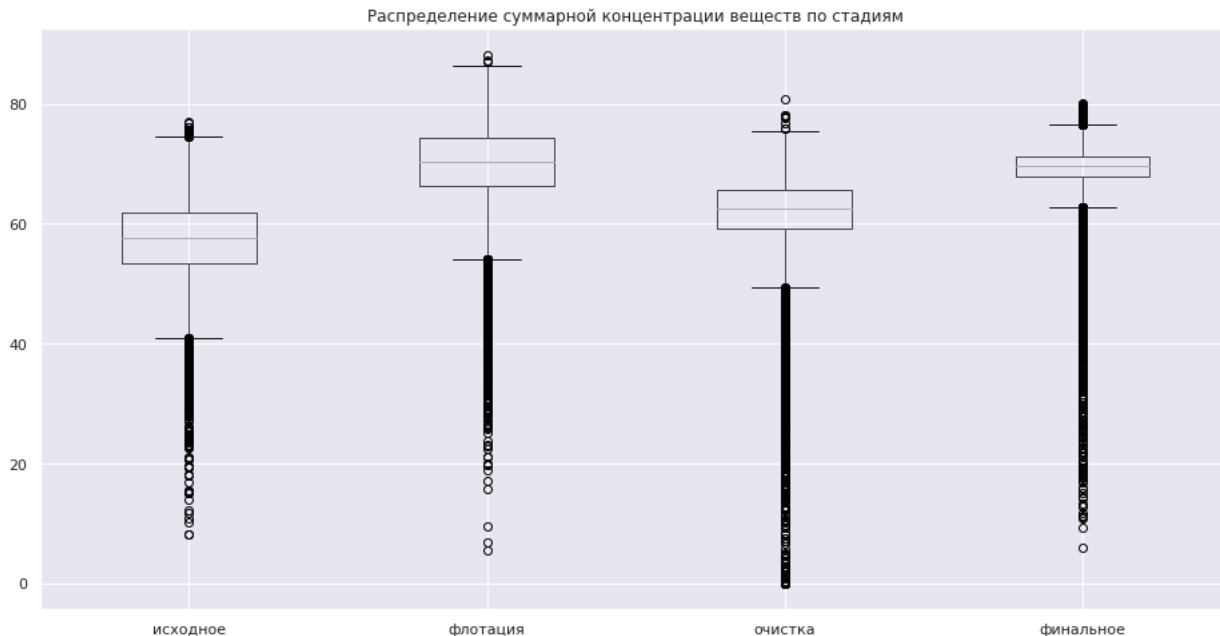
очистка

финальное

date

```
data_new.boxplot()
plt.title('Распределение суммарной концентрации веществ по стадиям');
```


↻ Text(0.5, 1.0, 'Распределение суммарной концентрации веществ по стадиям')

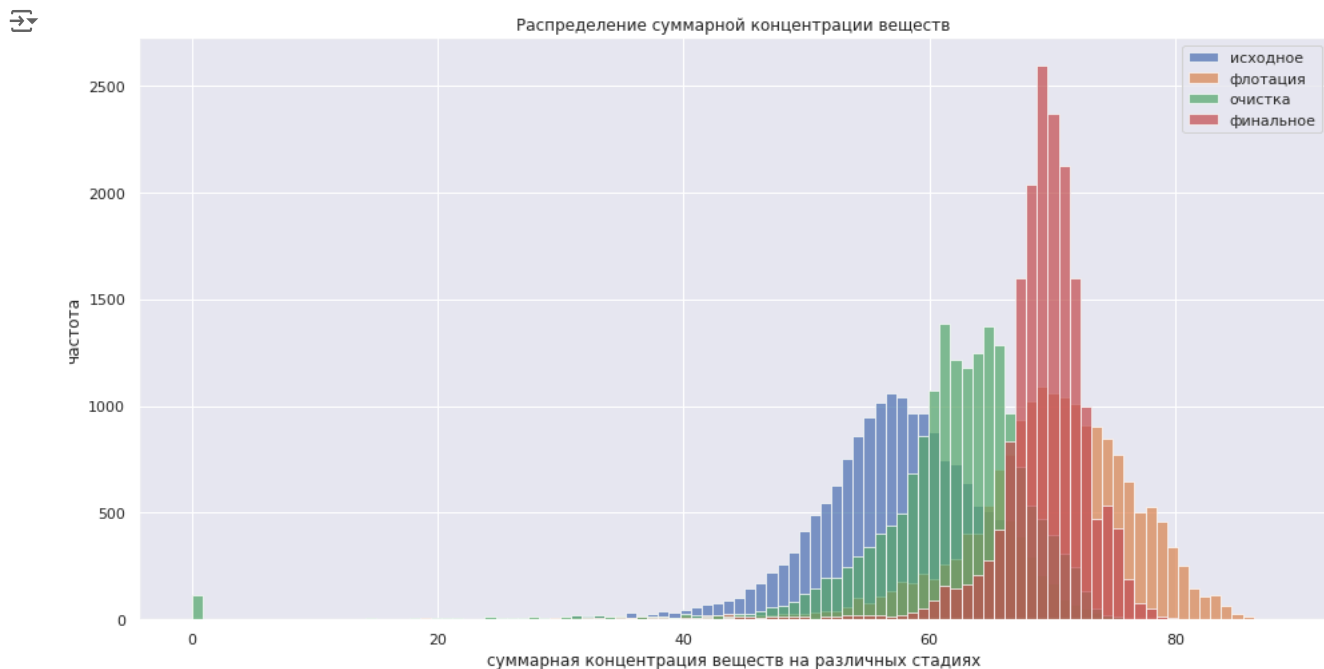


Комментарий Ирины 1

Рассмотрим распределение суммарной концентрации веществ

```
data_sum = data[columns_sum].copy()
data_sum.columns = ['исходное', 'флотация', 'очистка', 'финальное']
```

```
data_sum.plot(kind='hist',
              bins=100,
              alpha = 0.7,
              legend=True)
plt.xlabel('суммарная концентрация веществ на различных стадиях')
plt.ylabel('частота')
plt.title('Распределение суммарной концентрации веществ')
plt.show()
```



Комментарий Ирины 1

[Новые выводы](#)

Выводы

- суммарная концентрация веществ не сильно изменяется от процесса к процессу из-за вещества обозначенного '_sol'
- если не учитывать концентрацию '_sol', то суммарная концентрация веществ в смеси увеличивается от процесса к процессу
- на финальной стадии разброс данных существенно снижается, суммарная концентрация веществ имеет более узкий характер распределения концентрации, что ожидаемо получить, т.к. является целью всего процесса.

Комментарий ревьюера

Все отлично! 🍌: Исследована суммарная концентрация металлов на разных стадиях техпроцесса - отлично!

Комментарий ревьюера

На доработку ❌:

Довольно важный момент - линейный график не очень подходит для ответа на вопрос о динамике концентрации (как частной, так и суммарной): линейный график подходит для тех данных, когда мы исследуем динамику единичного процесса. В нашем случае мы работаем с большим количеством итераций процесса, который имеет отдельное начало и отдельный конец (то есть по сути своей автономен и не зависит от других процессов), а временная метка - просто дата снятия показаний с оборудования. Поэтому вместо линейного графика стоит исследовать распределение индивидуальных наблюдений концентрации металлов. Оптимальнее всего организовать визуализацию так, чтобы на каждый металл приходился отдельный график, и на каждом графике было бы отображено распределение индивидуальной концентрации на каждом этапе техпроцесса.

Комментарий ревьюера v.2

Все отлично! 🍌: Учтено.

Выводы по разделу Анализ данных

- Концентрации металлов на различных этапах:
 - медианное значение концентрации золота в концентрате от стадии к стадии растет
 - медианное значение концентрации свинца в концентрате растет
 - медианное значение концентрации серебра после флотации возрастает, после очистки снижается
- Размер гранул сырья:
 - средний размер гранул сырья тестовой выборки 56, 75% гранул имеют размер в диапазоне [43;62]
 - средний размер гранул сырья обучающей выборки 59, 75% гранул имеют размер в диапазоне [48;65]
 - распределения размеров гранул сырья тестовой и обучающей выборки не имеют сильного различия
- Размер гранул после первичной очистки:
 - средний размер гранул в тестовой выборке 7,2 в диапазоне [6.8; 7.6]
 - средний размер гранул в обучающей выборке 7,3 в диапазоне [7,0; 7.7]
 - распределения размеров гранул после первичной очистки в тестовой и обучающей выборках не имеют сильного различия
- Суммарная концентрация веществ:
 - суммарная концентрация веществ не сильно изменяется от процесса к процессу из-за вещества обозначенного '_sol'
 - если не учитывать концентрацию '_sol', то суммарная концентрация веществ в смеси увеличивается от процесса к процессу
 - на финальной стадии разброс данных существенно снижается, суммарная концентрация веществ имеет более узкий характер распределения концентрации, что ожидаемо получить, т.к. является целью всего процесса.

Комментарий ревьюера

Все отлично! 🍌: Хороший промежуточный вывод!

✓ Модель

В результате предыдущих шагов у нас есть три массива данных

- full_clean - все очищенные данные, дата - переведена в индексы, объем(17894,86)
- train - обучающая очищенная выборка +2 цел.признака, объем (12936, 54)
- test - тестовая выборка + 2 цел.признака, объем (4958, 54)

Модель будет оцениваться по метрике sMAPE:

$$sMAPE = 0.25sMAPE(rougher) + 0.75sMAPE(final)$$

sMAPE будем рассчитывать на валидационной выборке. Для этого из train выделим валидационную выборку 25% данных

✓ Выбор признаков для обучения

Комментарий Ирины 1

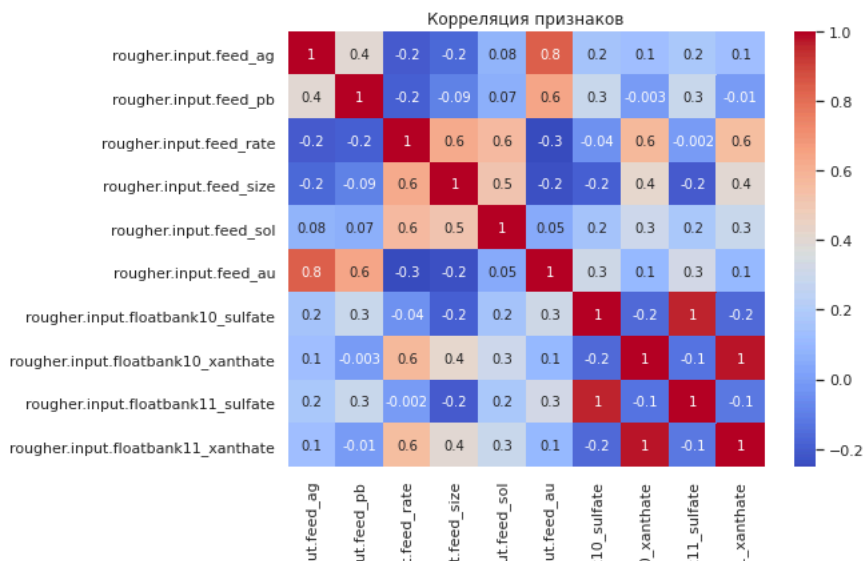
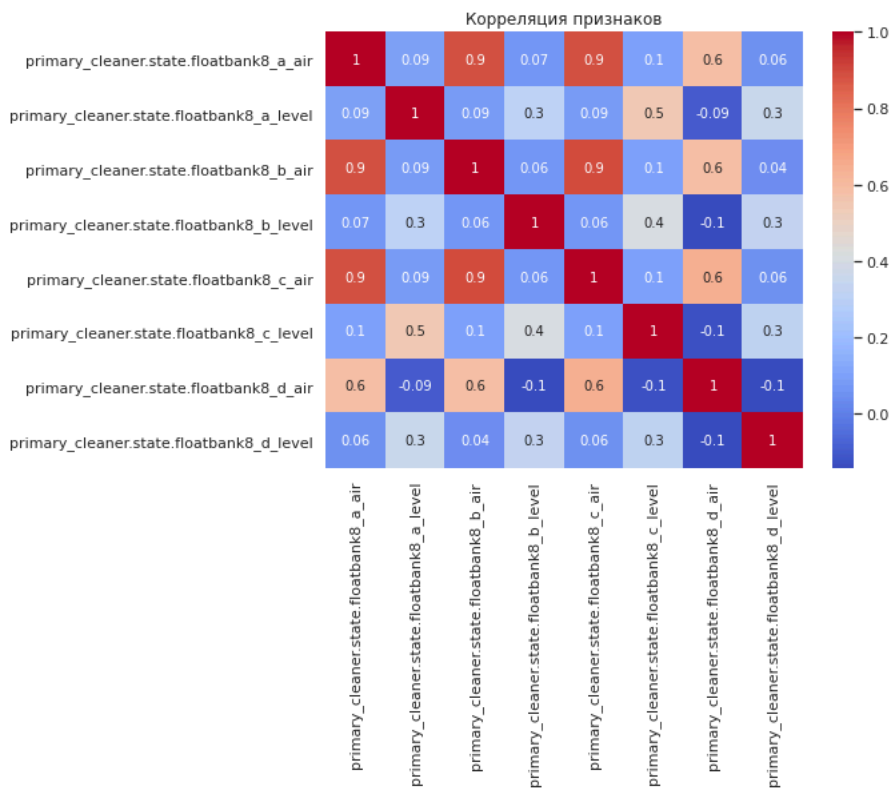
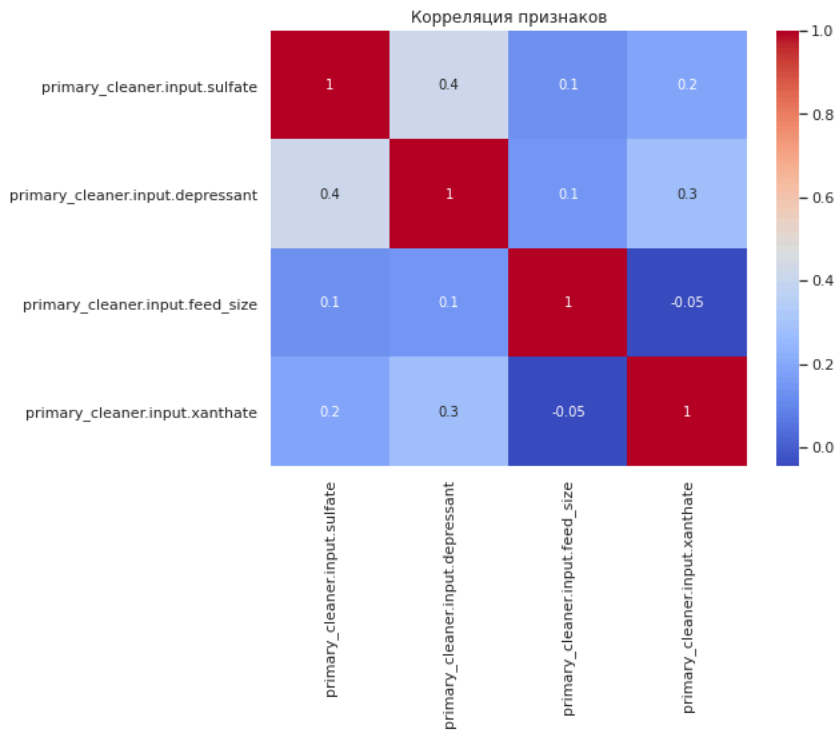
Ниже корректирую выбираемые параметры

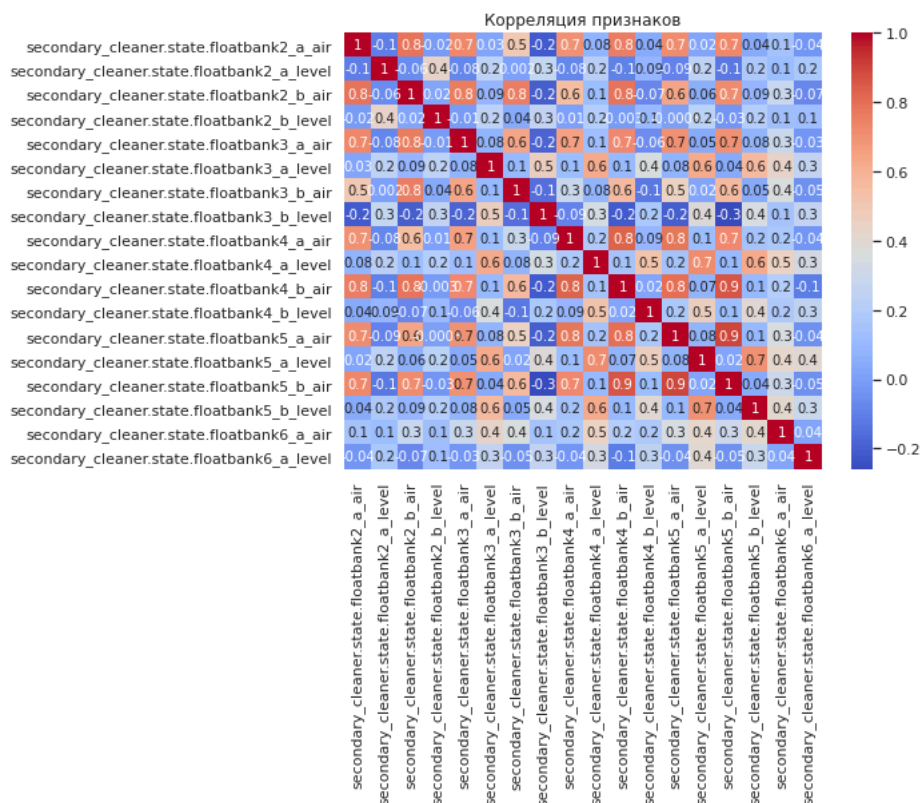
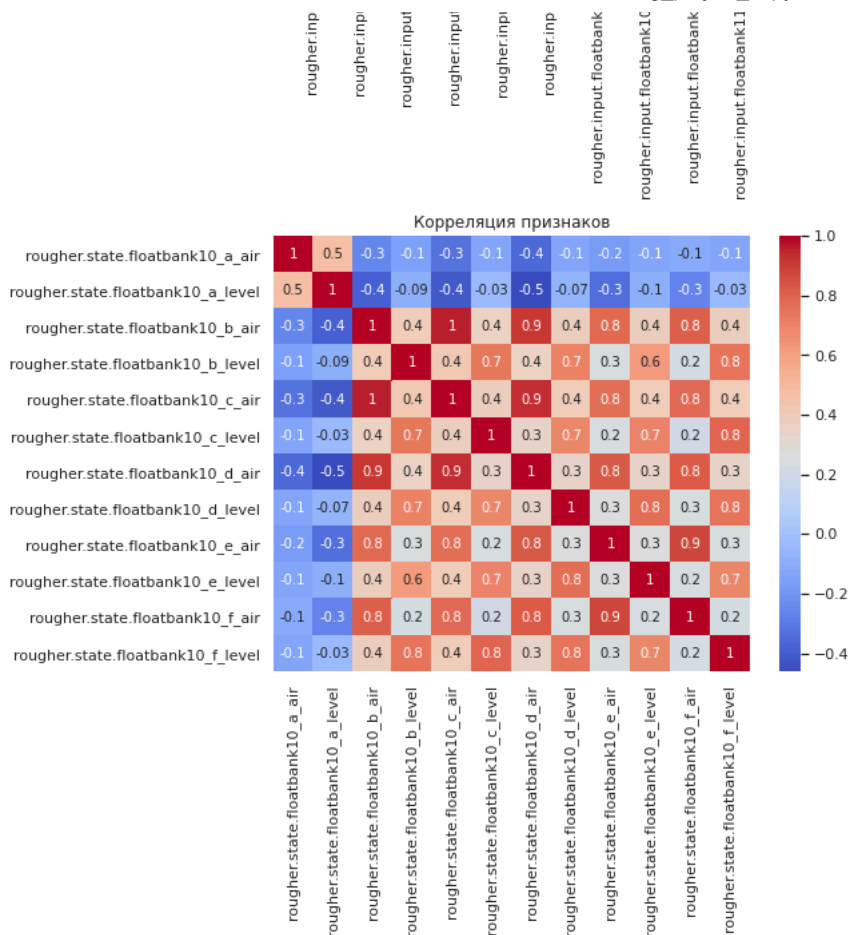
Необходимо понять, какие признаки оставить для обучения модели. Признаки не должны коррелировать между собой, удалим те, которые имеют коэффициент корреляции не менее 0.8

```
# посмотрим корреляцию внутри групп параметров
dict = [ 'primary_cleaner.input', 'primary_cleaner.state',
        'rougher.input', 'rougher.state', 'secondary_cleaner.state']

col = train.columns
for d in dict:
    temp = pd.DataFrame()
    for c in col:
        if(d in c):
            temp[c] = train[c]

plt.figure(figsize = (8,6))
plt.title('Корреляция признаков')
sns.heatmap(temp.corr(method='spearman'), annot = True,\
            cmap = 'coolwarm', annot_kws={'size':10}, fmt='.1g')
```





следует удалить следующие параметры имеющие корреляцию выше 0.8:

- 'primary_cleaner.state.floatbank8_b_air'
- 'primary_cleaner.state.floatbank8_c_air'
- 'rougher.input.feed_ag'
- 'rougher.state.floatbank10_d_air',
- 'rougher.state.floatbank10_c_air',
- 'rougher.state.floatbank10_e_air',
- 'rougher.state.floatbank10_f_air',

- 'secondary_cleaner.state.floatbank2_b_air',
- 'secondary_cleaner.state.floatbank4_b_air',
- 'rougher.input.floatbank10_sulfate',
- 'rougher.input.floatbank10_xanthate'
- 'rougher.state.floatbank10_e_level',
- 'rougher.state.floatbank10_f_level'

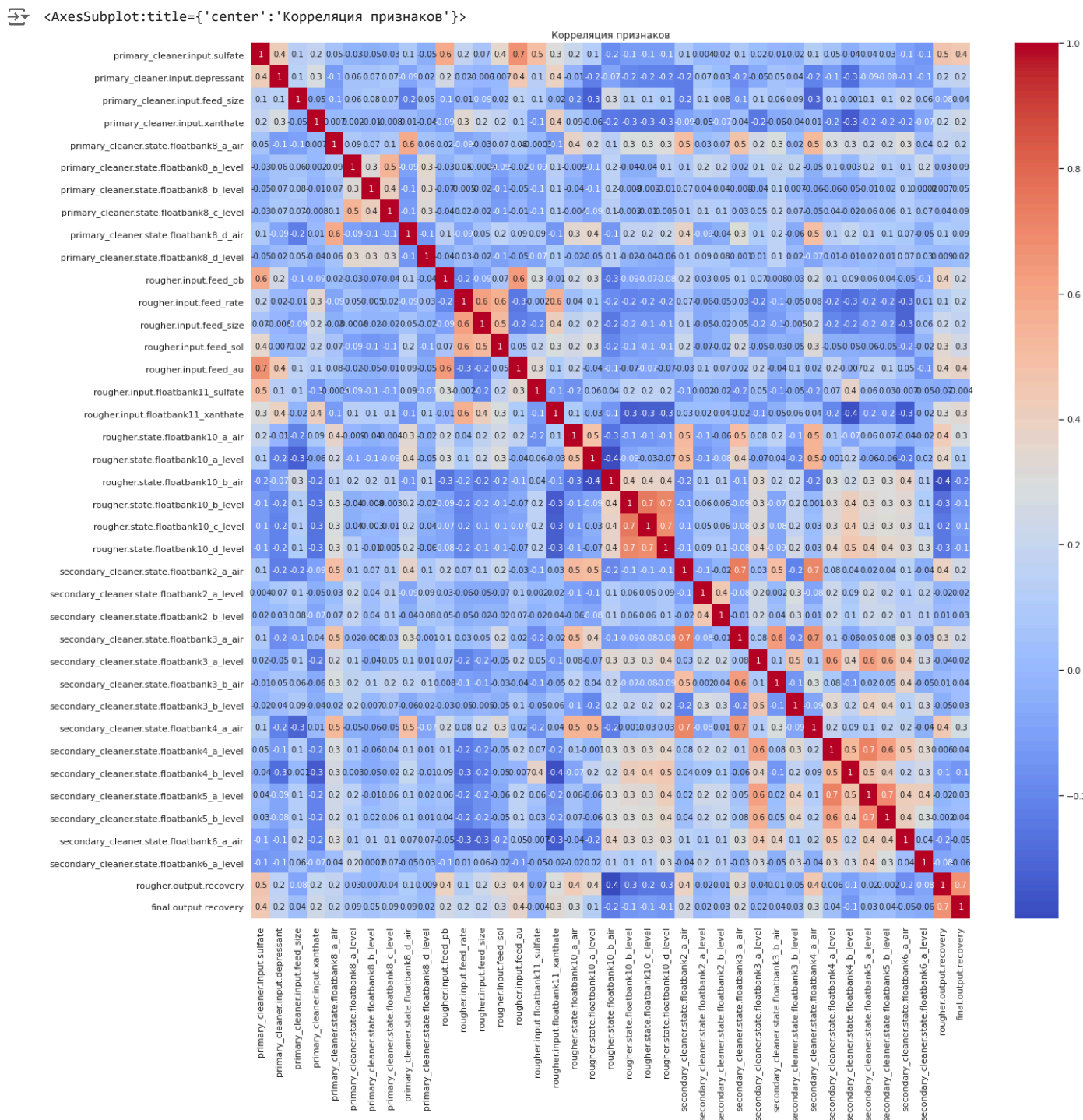
```
dict_for_delete = ['primary_cleaner.state.floatbank8_b_air',
                  'primary_cleaner.state.floatbank8_c_air',
                  'rougher.input.feed_ag',
                  'rougher.state.floatbank10_d_air',
                  'rougher.state.floatbank10_c_air',
                  'rougher.state.floatbank10_e_air',
                  'rougher.state.floatbank10_f_air',
                  'secondary_cleaner.state.floatbank2_b_air',
                  'secondary_cleaner.state.floatbank4_b_air',
                  'secondary_cleaner.state.floatbank5_a_air',
                  'secondary_cleaner.state.floatbank5_b_air',
                  'rougher.input.floatbank10_sulfate',
                  'rougher.input.floatbank10_xanthate',
                  'rougher.state.floatbank10_e_level',
                  'rougher.state.floatbank10_f_level']
```

```
try:
    train = train.drop(dict_for_delete, axis = 1)
except:
    print("Таких нет данных, возможно они уже удалены")
print('train shape: ', train.shape)
try:
    test = test.drop(dict_for_delete, axis = 1)
except:
    print("Таких нет данных, возможно они уже удалены")
print('test shape: ', test.shape)
```

```
→ train shape: (12936, 39)
   test shape: (4958, 39)
```

Теперь посмотрим корреляцию всех параметров между собой

```
plt.figure(figsize = (20,20))
plt.title('Корреляция признаков')
sns.heatmap(train.corr(method='spearman'), annot = True,\
            cmap = 'coolwarm', annot_kws={'size':10}, fmt='.1g');
```



Параметров с корреляцией более и равной 0.8 не осталось. В обучение и в тест пойдут 37 параметров

Комментарий ревьюера

На доработку ❌:

Идея удалить скоррелированные предикторы может быть неплохой, но принято искать мультиколлинеарность по границе ≥ 0.8 - нас интересует сильная линейная зависимость, а по шкале Чеддока в диапазоне 0.5-0.7 находится средняя зависимость.

Комментарий ревьюера v.2

Все отлично! 🍌: Учтено.

Выводы

- После удаления коррелирующих признаков с коэффициентом корреляции более 0.8 осталось 37 параметров, на которых будем обучать модель и тестировать модель

Комментарий ревьюера

На доработку ❌:

Нужно будет скорректировать с учётом замечания о воспроизводимости признаков.

Комментарий ревьюера v.2

Все отлично! 🍌: Учтено.

✓ Функция для расчета sMAPE

Для решения задачи введём новую метрику качества — sMAPE (англ. Symmetric Mean Absolute Percentage Error, «симметричное среднее абсолютное процентное отклонение»).

Она похожа на MAE, но выражается не в абсолютных величинах, а в относительных. Почему симметричная? Она одинаково учитывает масштаб и целевого признака, и предсказания.

Метрика sMAPE вычисляется так:

Комментарий Ирины 1

Переписанная в векторном виде функция smape

```
def sMAPE(data, data_predict):
    n = len(data)
    x=abs(data-data_predict)*2
    y=abs(data)+abs(data_predict)
    z= x/y
    smape = z.sum()*100/n
    return smape
```

Комментарий ревьюера

На доработку ❌:

1. Использование циклов в Pandas - антипаттерн, от которого стоит избавляться: итерирование по строкам может быть ресурсозатратным, особенно если речь о больших таблицах. Если переменные равны друг другу по размерам, стоит использовать векторные вычисления и оценивать метрику напрямую.
2. Метрика sMAPE измеряется в процентах - результат стоит умножить на 100.

Комментарий ревьюера v.2

Все отлично! 🍌: Учтено.

```
def sMAPE_total(data_r,data_f,data_predict_r, data_predict_f):
    s_r = sMAPE(data_r,data_predict_r)
    s_f = sMAPE(data_f,data_predict_f)
    smape_total = 0.25*s_r+0.75*s_f
    return smape_total
```


Выводы

- Написаны две функции для расчета sMAPE

Комментарий ревьюера

Все отлично! 🍌: Есть функции для частного и взвешенного sMAPE.

✓ **Разделение данных на выборки**

Комментарий Ирины 1

Количество параметров изменено, исследование ниже с новыми параметрами

Для начала необходимо оделить целевые признаки - это:

- rougher.output.recovery
- final.output.recovery

```
target_rougher = train['rougher.output.recovery'] # отделяем первый целевой признак
target_final = train['final.output.recovery'] # отделяем второй целевой признак
print(target_rougher.shape, target_final.shape)
```

```
(12936,) (12936,)
```

Признаки для обучения features возьмем из train

```
features = train.copy()
features = features.drop(['rougher.output.recovery', 'final.output.recovery'], axis = 1)
features.shape
```

```
(12936, 37)
```

```
# выделим по две выборки для валидации по 20% и выборки для обучения
features_rougher_train, features_rougher_valid, target_rougher_train, target_rougher_valid = train_test_split(
    features, target_rougher, test_size=0.20, random_state=12345)
```

```
features_final_train, features_final_valid, target_final_train, target_final_valid = train_test_split(
    features, target_final, test_size=0.20, random_state=12345)
```

```
print('выборки: ')
print('features_rougher_train', features_rougher_train.shape)
print('features_rougher_valid', features_rougher_valid.shape)
print()
print('target_rougher_train', target_rougher_train.shape)
print('target_rougher_valid', target_rougher_valid.shape)
print()
print('features_final_train', features_final_train.shape)
print('features_final_valid', features_final_valid.shape)
print()
print('target_final_train', target_final_train.shape)
print('target_final_valid', target_final_valid.shape)
```

```
выборки:
features_rougher_train (10348, 37)
features_rougher_valid (2588, 37)

target_rougher_train (10348,)
target_rougher_valid (2588,)

features_final_train (10348, 37)
features_final_valid (2588, 37)

target_final_train (10348,)
target_final_valid (2588,)
```

Выводы

- Так как необходимо предсказать два целевых признака то данные из массива train разбиваются на 4 выборки:
 - признаки и целевой признак для rougher коэффициента восстановления: обучающие и валидационные

- features_rougher_train (10348, 37)
- features_rougher_valid (2588, 37)
- target_rougher_train (10348,)
- target_rougher_valid (2588,)
- признаки и целевой признак для final коэффициента восстановления: обучающие и валидационные
 - features_final_train (10348, 37)
 - features_final_valid (2588, 37)
 - target_final_train (10348,)
 - target_final_valid (2588,)

Комментарий ревьюера

Все отлично! 🍌: Выделили признаки для обучения и целевые признаки - отлично!

Комментарий ревьюера

Некоторые замечания и рекомендации ⚠️: Валидационная выборка как форма промежуточной оценки качества моделей сопряжена с некоторыми недостатками:

- валидационная выборка формируется за счёт части тренировочных данных - как итог, модель теряет часть данных, на которых могла бы обучаться;
- валидационная выборка не очень эффективно контролирует моменты, связанные с недо- или переобучением.

Вместо валидационной выборки рекомендую использовать кросс-валидацию для промежуточной оценки модели:

- она разбивает данные на фолды, соответственно модель итеративно обучится на полном наборе обучающих данных;
- так как каждый фолд будет участвовать и в обучении, и в тестировании, мы снизим риск недо- или переобучения, получив более надёжную метрику.

Комментарий Ирины 1

Алексей, какую статью ты рекомендуешь прочитать про кросс-валидацию, чтобы в следующий раз я могла ее применить?

Комментарий ревьюера v.2

Все отлично! 🍌: По обучающим материалам лучше обратиться к преподавателю - он сможет дать материал уровня, соответствующий уровню погружения на протяжении курса. Так будет лучше, потому что я могу случайно дать сложный материал - преподаватель лучше знает нагрузку:)

✓ Модель решающее дерево

Обучим модель решающее дерево. Обучим модели с глубиной ветвления дерева от 1 до 26 и посмотрим метрику sMare для каждой модели. Чем она будет меньше, тем лучше.

Создадим структуру данных, в которую будем записывать результаты для всех моделей в проекте: df_models_results

```

#модель решающее дерево для rougher
#гиперпараметр глубина дерева

best_deth_dtree = 0
best_result_dtree = 100
best_model_dtree_rougher = None
best_model_dtree_final = None

def model_dtree_result(max_depth_i,
                       features_rougher_train,
                       target_rougher_train,
                       features_final_train,
                       target_final_train,
                       features_rougher_valid,
                       features_final_valid,
                       target_rougher_valid,
                       target_final_valid):
    model_dtree_rougher = DecisionTreeRegressor(max_depth=max_depth_i, random_state=12345)
    model_dtree_final = DecisionTreeRegressor(max_depth=max_depth_i, random_state=12345)
    # обучение модели
    model_dtree_rougher.fit(features_rougher_train, target_rougher_train)
    model_dtree_final.fit(features_final_train, target_final_train)
    # предсказания модели на валидационной выборке
    predictions_dtree_rougher_valid=model_dtree_rougher.predict(features_rougher_valid)
    predictions_dtree_final_valid=model_dtree_final.predict(features_final_valid)
    # расчет SMAPE на валидационной выборке
    result = SMAPE_total(target_rougher_valid, target_final_valid , predictions_dtree_rougher_valid, predictions_dtree_final_valid)
    return [result,model_dtree_rougher,model_dtree_final]

#проверим модель с max_depth = none
results = model_dtree_result(None,
                             features_rougher_train,
                             target_rougher_train,
                             features_final_train,
                             target_final_train,
                             features_rougher_valid,
                             features_final_valid,
                             target_rougher_valid,
                             target_final_valid)

best_result_dtree = results[0]
print(f'глубина None Итоговая метрика качества SMAPE верных предсказаний: {best_result_dtree:.3}')
```

цикл для max_depth от 2 до 13 >


```

for i in range(2,13) :
    results = model_dtree_result(i,
                                 features_rougher_train,
                                 target_rougher_train,
                                 features_final_train,
                                 target_final_train,
                                 features_rougher_valid,
                                 features_final_valid,
                                 target_rougher_valid,
                                 target_final_valid)
    print(f'глубина {i} Итоговая метрика качества SMAPE верных предсказаний: {results[0]:.3}')
```

```

if (best_result_dtree > results[0]):
    best_result_dtree = results[0]
    best_deth_dtree = i
    best_model_dtree_rougher = results[1]
    best_model_dtree_final = results[2]

print()
print(f'Глубина дерева для наилучшей модели max_depth = {best_deth_dtree} Лучшая итоговая метрика SMAPE:{ best_result_dtree:.3}')
```


 глубина None Итоговая метрика качества SMAPE верных предсказаний: 7.09
 глубина 2 Итоговая метрика качества SMAPE верных предсказаний: 8.29
 глубина 3 Итоговая метрика качества SMAPE верных предсказаний: 7.74
 глубина 4 Итоговая метрика качества SMAPE верных предсказаний: 7.4
 глубина 5 Итоговая метрика качества SMAPE верных предсказаний: 7.14
 глубина 6 Итоговая метрика качества SMAPE верных предсказаний: 7.04
 глубина 7 Итоговая метрика качества SMAPE верных предсказаний: 6.84
 глубина 8 Итоговая метрика качества SMAPE верных предсказаний: 6.65
 глубина 9 Итоговая метрика качества SMAPE верных предсказаний: 6.7
 глубина 10 Итоговая метрика качества SMAPE верных предсказаний: 6.35
 глубина 11 Итоговая метрика качества SMAPE верных предсказаний: 6.44
 глубина 12 Итоговая метрика качества SMAPE верных предсказаний: 6.52

Глубина дерева для наилучшей модели max_depth = 10 Лучшая итоговая метрика SMAPE:6.35

Комментарий ревьюера

Некоторые замечания и рекомендации ⚠: Для гиперпараметра `max_depth` рассматривается очень длинная последовательность: деревья с большой глубиной склонны к переобучению, а обучаются и предсказывают результат они дольше, поэтому делать их слишком глубокими не стоит - оптимальное значение почти всегда лежит в диапазоне от 2 до 5-6. Кроме того, можно попробовать значение `None` - в итоге оптимальная последовательность может выглядеть как `[None] + [i for i in range(2, 7)]`.

```
# запишем результаты по первой модели
df_models_results = pd.DataFrame()
if (len(df_models_results.index)<1):
    df_models_results = df_models_results.append(
        {
            'model_name': "решающее дерево",
            'best_depth': best_deth_dtree,
            'best_est': None,
            'best_result': round(best_result_dtree,1)}, ignore_index=True)
df_models_results
```

	best_depth	best_est	best_result	model_name
0	10.0	None	6.4	решающее дерево

Выводы

- Перебраны 12 значений параметров для модели Решающее дерево для глубины ветвления
- Наилучшее значение показала модель с глубиной ветвления 10
- Метрика sMape составила 6,4%

Комментарий ревьюера

Все отлично! 🍌: Для `DecisionTreeRegressor` оптимизировали гиперпараметры и вывели оценку на `valid`.

✓ Случайный лес

Обучим модель случайный лес. Исследуем влияние двух гиперпараметров на предсказания модели: глубина ветвления дерева и количество деревьев.