Реальная задача Data Science из золотодобывающей отрасли. Проект предоставлен компанией «Цифра».

Постановка задачи

Подготовьте прототип модели машинного обучения для «Цифры». Компания разрабатывает решения для эффективной работы промышленных предприятий. Модель должна предсказать коэффициент восстановления золота из золотосодержащей руды. В вашем распоряжении данные с параметрами добычи и очистки. Модель поможет оптимизировать производство, чтобы не запускать предприятие с убыточными характеристиками.

Технологический процесс

Когда добытая руда проходит первичную обработку, получается дроблёная смесь. Её отправляют на флотацию (обогащение) и двухэтапную очистку.

1. Флотация

Во флотационную установку подаётся смесь золотосодержащей руды. После обогащения получается черновой концентрат и «отвальные хвосты», то есть остатки продукта с низкой концентрацией ценных металлов. На стабильность этого процесса влияет непостоянное и неоптимальное физико-химическое состояние флотационной пульпы (смеси твёрдых частиц и жидкости).

2. Очистка

Черновой концентрат проходит две очистки. На выходе получается финальный концентрат и новые отвальные хвосты.

Описание данных

Технологический процесс

- `Rougher feed` — исходное сырье
- `Rougher additions` (или reagent additions) — флотационные реагенты:
  - `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`

Значения для блока [этап]:

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

Значения для блока [тип_параметра]:

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

Расчёт эффективности

Необходимо смоделировать процесс восстановления золота из золотосодержащей руды. Эффективность обогащения рассчитывается по формуле

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

, 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.

Метрика качества

Для решения задачи введём новую метрику качества — `sMAPE` (англ. `Symmetric Mean Absolute Percentage Error` , «симметричное среднее абсолютное процентное отклонение»). Она одинаково учитывает масштаб и целевого признака, и предсказания.

$$sMAPE = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{\frac{|y_i|+|\hat{y}_i|}{2}}$$

, где

- y_i - значение целевого признака для объекта с порядковым номером i в выборке, на которой измеряется качество.
- $\hat{y}_i$ - значение предсказания для объекта с порядковым номером i, например, в тестовой выборке.

Нужно спрогнозировать сразу две величины:

- эффективность обогащения чернового концентрата `rougher.output.recovery` ;
- эффективность обогащения финального концентрата `final.output.recovery` .

Итоговая метрика складывается из двух величин:

$$sMAPE_{result} = 25\% * sMAPE(rougher) + 75\% * sMAPE(final)$$

План

1. Подготовка данных

```
1.1 Получение файлов
1.2 Проверка верности расчёта эффективности обогащения
1.3 Анализ признаков, которых нет в тестовой выборке
1.4 Предобработка данных
```

2. Исследовательский анализ данных

```
2.1 Динамика изменения концентрации металлов (Au, Ag, Pb) на различных э
тапах очистки.
2.2 Сравнение распределения размеров гранул сырья на обучающей и тестово
й выборках.
2.3 Исследование суммарной концентрации всех веществ на разных стадиях:
 в сырье, в черновом и финальном концентратах.
```

3. Построение модели и её обучение

3.1 Функция нахождения итоговой sMAPE.
3.2 Обучение моделей и оценка их качеств кросс-валидацией. Выбор лучшей модели и её проверка на тестовой выборке.

1. Подготовка данных

1.1 Получение файлов

B [1]:
```
import pandas as pd
import numpy as np

from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import make_scorer

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression

import os
import urllib

import matplotlib.pyplot as plt

pd.options.display.float_format = "{:.3f}".format
PLOT = True
```

B [2]:
```
datas = [
    ['datasets/gold_recovery_train', 'https://code.s3.yandex.net/datasets/gold_re
    ['datasets/gold_recovery_test', 'https://code.s3.yandex.net/datasets/gold_rec
    ['datasets/gold_recovery_full', 'https://code.s3.yandex.net/datasets/gold_rec
]

os.makedirs(datas[0][0].split('/')[0], exist_ok=True)
_ = [urllib.request.urlretrieve(url, name) for name, url in datas if not os.path.
```

B [3]:
```
class DF:
    def __init__(self, df):
        self.df = df

    def get(self, part_of_name):
        return self.df.loc[:, list(filter(lambda x: part_of_name in x, self.df.co
```

B [4]:
```
train_origin, test_origin, full_origin = [DF(pd.read_csv(name)) for name, _ in da
train, test, full = [DF(pd.read_csv(name)) for name, _ in datas]
target_col_1, target_col_2 = 'rougher.output.recovery', 'final.output.recovery'
```

1.2 Проверка верности расчёта эффективности обогащения

Вычислите её на обучающей выборке для признака rougher.output.recovery. Найдите MAE между вашими расчётами и значением признака. Опишите выводы.

B [5]:
```
C = train.get('rougher.output.concentrate_au').iloc[:, 0]
F = train.get('rougher.input.feed_au').iloc[:, 0]
T = train.get('rougher.output.tail_au').iloc[:, 0]
Recovery_calc = ((C * (F - T)) / (F * (C - T)) * 100)
Recovery_actu = train.get('rougher.output.recovery').iloc[:, 0]
res = pd.concat([Recovery_calc, Recovery_actu], axis=1)
res = res.dropna()
res.columns = 'calculated actual'.split()
print('Расчёт признака Recovery верный' if mae(res.iloc[:, 0], res.iloc[:, 1]) <
```

```
Расчёт признака Recovery верный
```

1.3 Анализ признаков, которых нет в тестовой выборке

B [6]:
```
cols_not_in_test = sorted(list(set(full.df.columns) - set(test.df.columns)))
print(*cols_not_in_test, sep='\n')
print('#' * 50)
print(len(cols_not_in_test))
print("Is gen features in test -> ", target_col_1 in cols_not_in_test, target_col
```

```
final.output.concentrate_ag
final.output.concentrate_au
final.output.concentrate_pb
final.output.concentrate_sol
final.output.recovery
final.output.tail_ag
final.output.tail_au
final.output.tail_pb
final.output.tail_sol
primary_cleaner.output.concentrate_ag
primary_cleaner.output.concentrate_au
primary_cleaner.output.concentrate_pb
primary_cleaner.output.concentrate_sol
primary_cleaner.output.tail_ag
primary_cleaner.output.tail_au
primary_cleaner.output.tail_pb
primary_cleaner.output.tail_sol
rougher.calculation.au_pb_ratio
rougher.calculation.floatbank10_sulfate_to_au_feed
rougher.calculation.floatbank11_sulfate_to_au_feed
rougher.calculation.sulfate_to_au_concentrate
rougher.output.concentrate_ag
rougher.output.concentrate_au
rougher.output.concentrate_pb
rougher.output.concentrate_sol
rougher.output.recovery
rougher.output.tail_ag
rougher.output.tail_au
rougher.output.tail_pb
rougher.output.tail_sol
secondary_cleaner.output.tail_ag
secondary_cleaner.output.tail_au
secondary_cleaner.output.tail_pb
secondary_cleaner.output.tail_sol
##################################################
34
Is gen features in test ->  True True
```

В тестовой выборке не хватает 34 признаков, включая `rougher.output.recovery` , `final.output.recovery` (целевые прирнаки). Все признаки, кроме целевых, а точнее их величины могут быть получены лишь при работающем механизме фильтрации (концентрации, излишки)

1.4 Предобработка данных

B [7]:
```
train.df.columns.shape[0], test.df.columns.shape[0], full.df.columns.shape[0]
```

Out[7]:
```
(87, 53, 87)
```

В датасетах различное количество столбцов, что недопустимо для обучения модели (`train` должен иметь такие же столбцы, как и `test` или `valid`). Проверим, совпадают ли столбцы из `train` с `full` на предмет того, что все столбцы рассматриваются.

B [8]:
```
set(train.df.columns) == set(full.df.columns)
```

Out[8]:
```
True
```

B [9]:
```
target_col_1 in full.df.columns and target_col_2 in full.df.columns
```

Out[9]:
```
True
```

B [10]:
```
train.df = train.df.loc[:, list(test.df.columns) + [target_col_1, target_col_2]]
```

B [11]:
```
for df in train.df, test.df, full.df:
    df.index = df.date
    df.drop(['date'], axis=1, inplace=True)
```

B [12]:
```
test.df = test.df.join(full.df[[target_col_1, target_col_2]])
```

B [13]:
```
print('Nan values in train (max) {:.3f} %'.format(train.df.isna().sum().max() / t
print('Nan values in test (max) {:.3f} %'.format(test.df.isna().sum().max() / tes
```

```
Nan values in train (max) 15.261 %
Nan values in test (max) 9.324 %
```

B [14]:
```
%%time
f = s = t = 1
while f or s or t:
    if f:
        train.df = train.df.ffill(limit=5)
        train.df = train.df.bfill(limit=5)
    if s:
        test.df = test.df.ffill(limit=5)
        test.df = test.df.bfill(limit=5)
    if t:
        full.df = full.df.ffill(limit=5)
        full.df = full.df.bfill(limit=5)
    f, s, t = (train.df.isna().sum().sum(),\
               test.df.isna().sum().sum(),\
               full.df.isna().sum().sum())
```

```
Wall time: 1.42 s
```

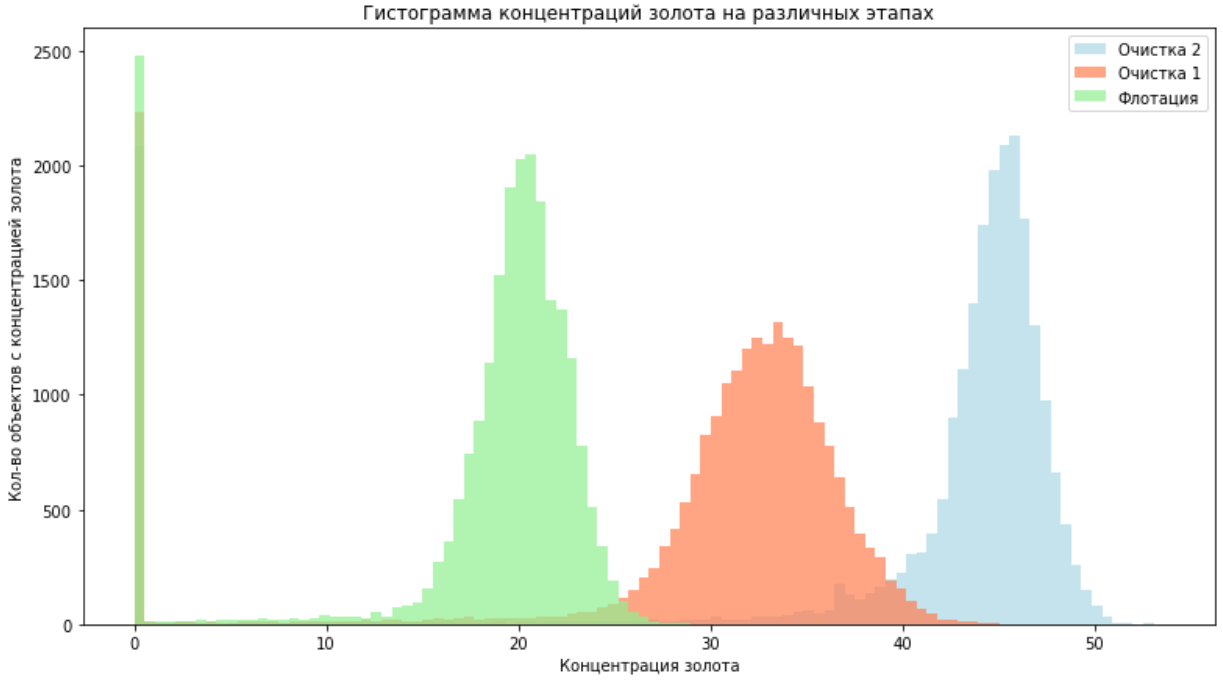
B [15]:
```
print(train.df.isna().sum().sum(), test.df.isna().sum().sum(), full.df.isna().sum
```

```
0 0 0
```

2. Исследовательский анализ данных

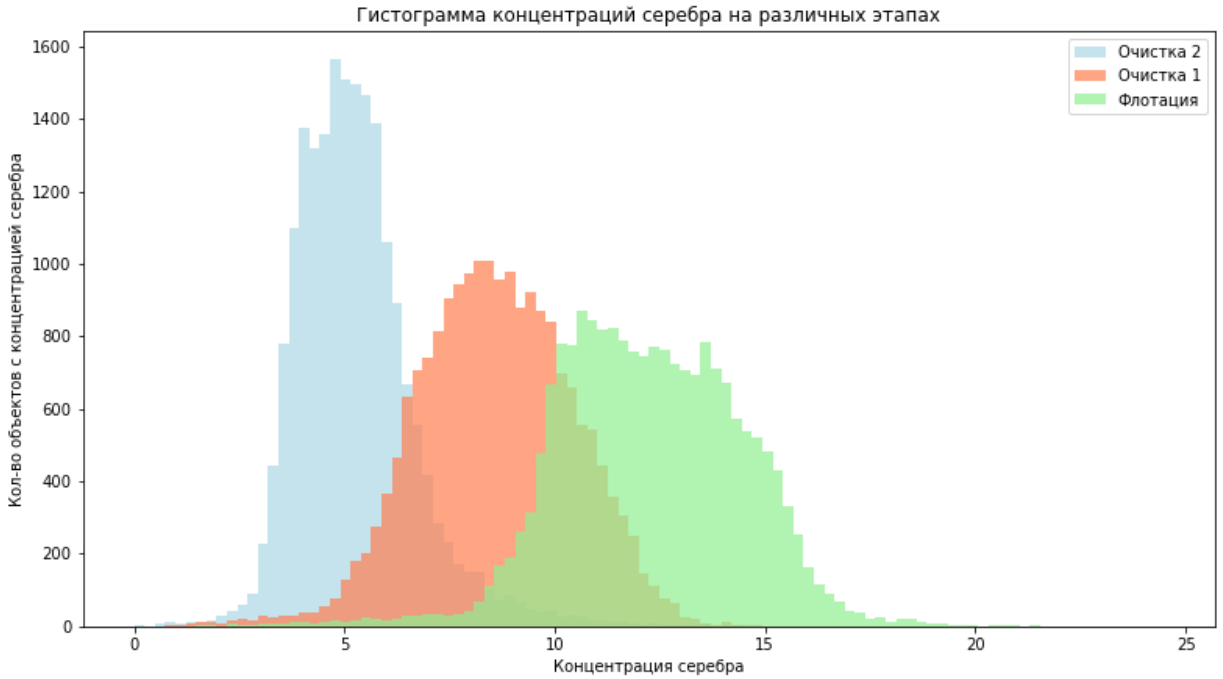
2.1 Динамика изменения концентрации металлов (Au, Ag, Pb) на различных этапах очистки.

B [16]:
```
if PLOT:
    df = full.get('concentrate_au').reset_index(drop=True)
    _ = df.plot.hist(figsize=(13, 7), y=df.columns, bins=100,\
                 color=('lightblue', 'coral', 'lightgreen'), alpha=.7,\
                    label=('Очистка 2', 'Очистка 1', 'Флотация'))
    plt.title('Гистограмма концентраций золота на различных этапах')
    plt.xlabel('Концентрация золота')
    plt.ylabel('Кол-во объектов с концентрацией золота')
    plt.show()
```

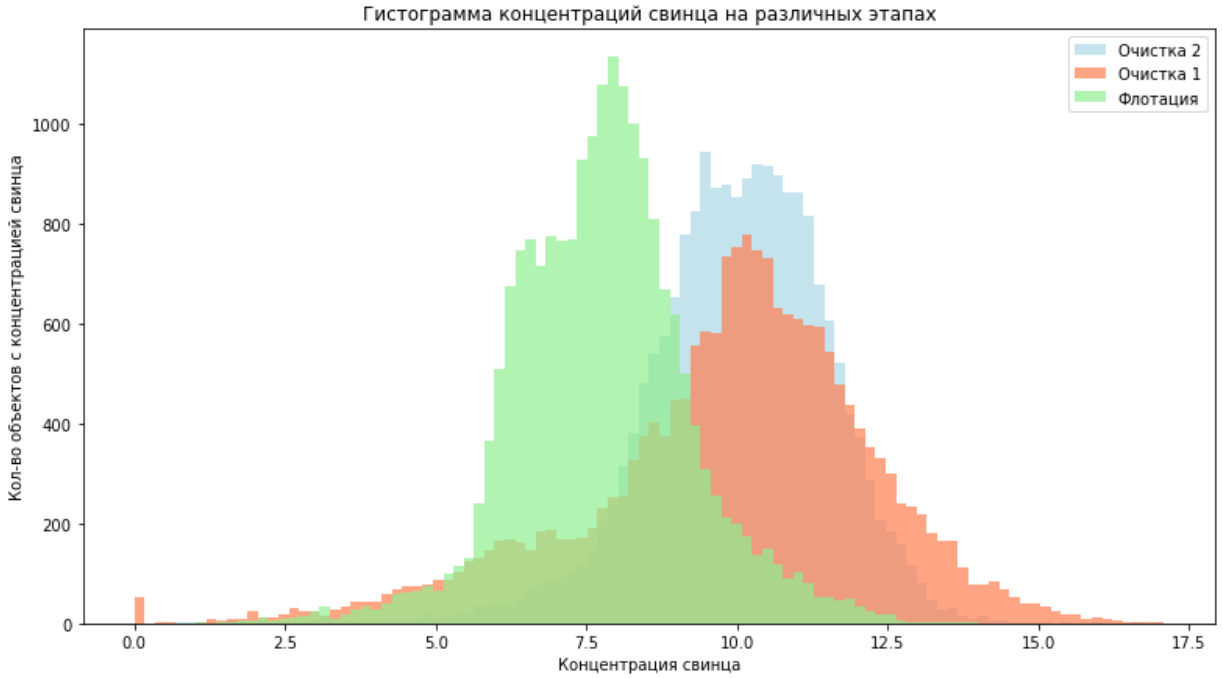


B [17]:
```
if PLOT:
    full.df = full.df.loc[(full.df['final.output.concentrate_au'] > 5) &
                          (full.df['primary_cleaner.output.concentrate_au'] > 5)
                          (full.df['rougher.output.concentrate_au'] > 5), :]
```

In [18]:
```
if PLOT:
    df = full.get('concentrate_ag').reset_index(drop=True)
    _ = df.plot.hist(figsize=(13, 7), y=df.columns, bins=100,\
                 color=('lightblue', 'coral', 'lightgreen'), alpha=.7,\
                label=('Очистка 2', 'Очистка 1', 'Флотация'))
    plt.title('Гистограмма концентраций серебра на различных этапах')
    plt.xlabel('Концентрация серебра')
    plt.ylabel('Кол-во объектов с концентрацией серебра')
    plt.show()
```



In [19]:
```
if PLOT:
    df = full.get('concentrate_pb').reset_index(drop=True)
    _ = df.plot.hist(figsize=(13, 7), y=df.columns, bins=100,\
                 color=('lightblue', 'coral', 'lightgreen'), alpha=.7,\
                label=('Очистка 2', 'Очистка 1', 'Флотация'))
    plt.title('Гистограмма концентраций свинца на различных этапах')
    plt.xlabel('Концентрация свинца')
    plt.ylabel('Кол-во объектов с концентрацией свинца')
    plt.show()
```

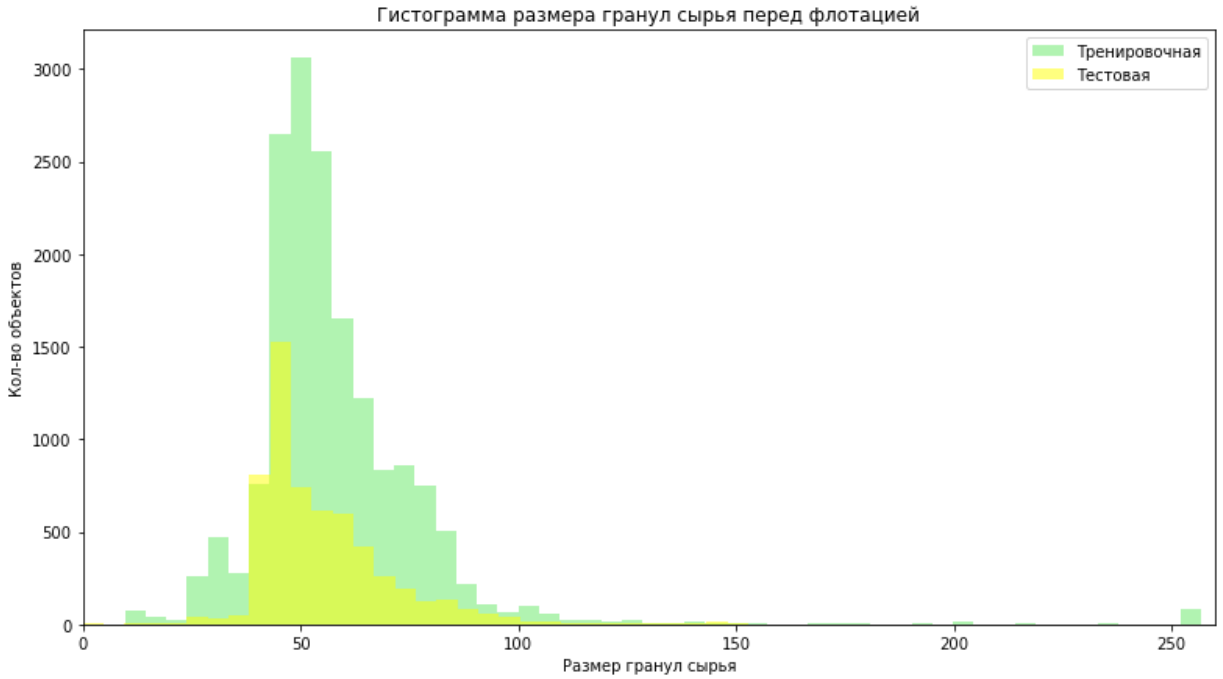


Концентраци серебра при прохождении руды через такую последовательность этапов: флотация, первая очистка и вторая очистка уменьшается. При таком же следовании этапов концетрация золота увеличивается. В свою очередь после флотации концетрация свинца увеличивается, но разница не так хорошо прослеживается между очистками.

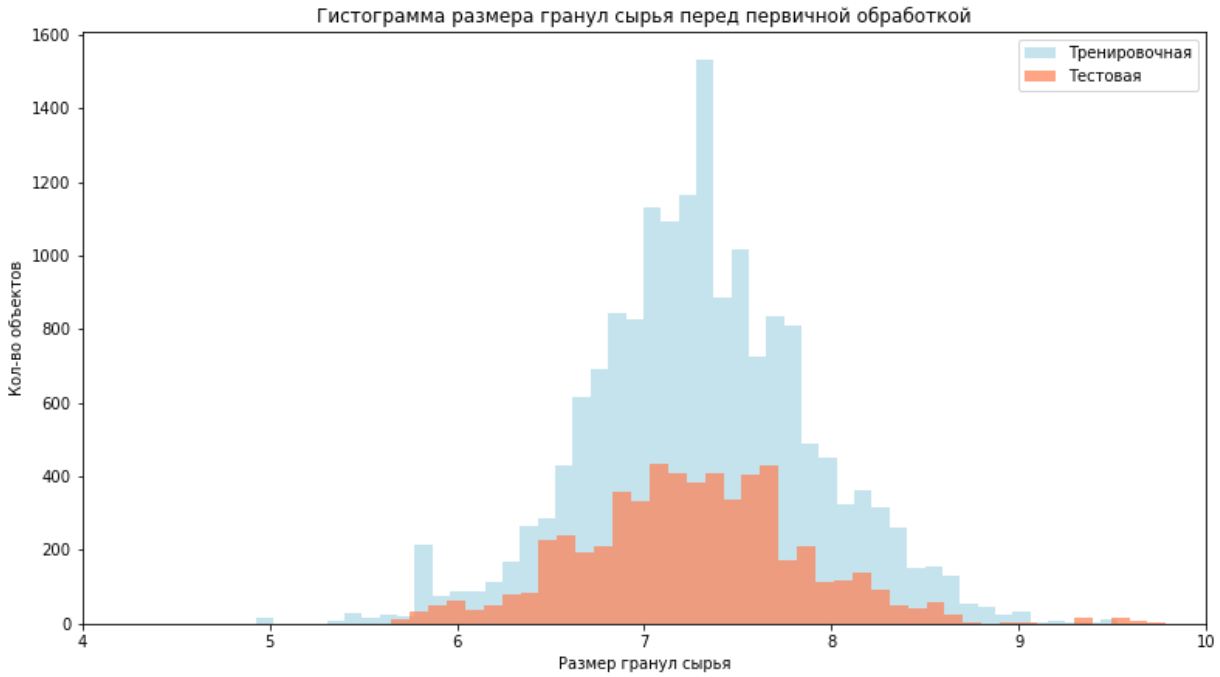
2.2 Сравнение распределения размеров гранул сырья на обучающей и тестовой выборках.

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

B [20]:
```
if PLOT:
    df_train = train.df[['rougher.input.feed_size']].reset_index(drop=True)
    df_test = test.df[['rougher.input.feed_size']].reset_index(drop=True)

    df_train.columns = ['Тренировочная']
    df_test.columns = ['Тестовая']

    axis = df_train.plot.hist(figsize=(13, 7), bins=100,\
                color='lightgreen', alpha=.7)
    axis = df_test.plot(kind='hist', figsize=(13, 7), bins=100,\
                color='yellow', alpha=.5, ax=axis)
    plt.title('Гистограмма размера гранул сырья перед флотацией')
    plt.xlabel('Размер гранул сырья')
    plt.ylabel('Кол-во объектов')
    plt.xlim((0, 260))
    plt.show()
```



In [21]:
```
if PLOT:
    df_train = train.df[['primary_cleaner.input.feed_size']]
    df_test = test.df[['primary_cleaner.input.feed_size']]

    df_train.columns = ['Тренировочная']
    df_test.columns = ['Тестовая']

    axis = df_train.plot.hist(figsize=(13, 7), bins=100,\
                color='lightblue', alpha=.7)
    axis = df_test.plot.hist(figsize=(13, 7), bins=100,\
                color='coral', alpha=.7, ax=axis)
    plt.title('Гистограмма размера гранул сырья перед первичной обработкой')
    plt.xlabel('Размер гранул сырья')
    plt.ylabel('Кол-во объектов')
    plt.xlim((4, 10))
    plt.show()
```



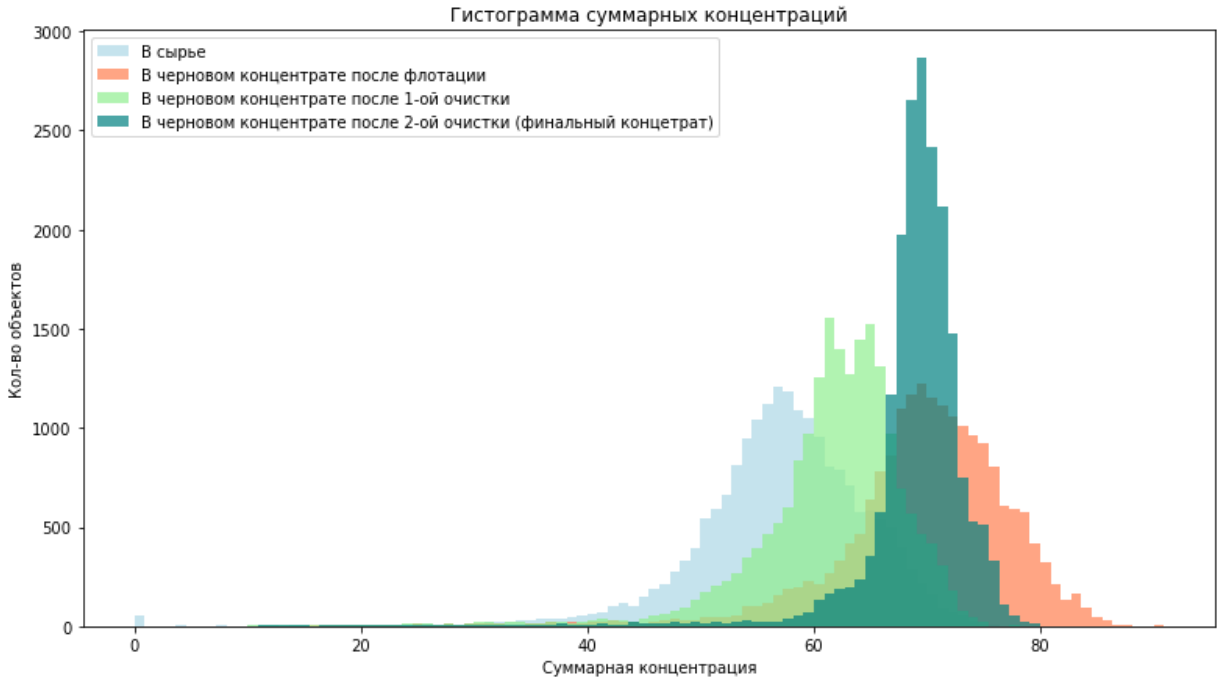
Размер гранул сырья перед флотацией на обеих выборках имеют распределения похожие на распределение Пуссона, в свою очередь распределения перед первой очисткой похожи на нормальное распределение

2.3 Исследование суммарной концентрации всех веществ на разных стадиях: в сырье, в черновом и финальном концентратах.

B [22]:
```
if PLOT:
    full.df.loc[:, 'rougher.input.feed_sum'] = (full
        .get('rougher.input.feed')
        .drop(['rougher.input.feed_size', 'rougher.input.feed_rate'], axis=1)
        .apply(lambda x: x.sum(), axis=1)
    )
```

B [23]:
```
if PLOT:
    for name in ['rougher.output.concentrate', 'primary_cleaner.output.concentrat
        full.df.loc[:, name + '_sum'] = (full
            .get(name)
            .apply(lambda x: x.sum(), axis=1)
        )
```

B [24]:
```
if PLOT:
    df_sum_conc = full.get('sum')
    df_sum_conc.columns = ['В сырье', 'В черновом концентрате после флотации',\
                           'В черновом концентрате после 1-ой очистки', 'В чернов
    _ = df_sum_conc.plot(kind='hist', bins=100, figsize=(13, 7),\
                         color=('lightblue', 'coral', 'lightgreen', 'teal'), alpha=.7
    plt.title('Гистограмма суммарных концентраций')
    plt.xlabel('Суммарная концентрация')
    plt.ylabel('Кол-во объектов')
    plt.show()
```



Как и ожидалось, концентрация металлов увеличивается к концу всех этапов.

3. Построение модели и её обучение

3.1 Функция нахождения итоговой sMAPE.

B [25]:
```
def sMAPE(predicted, target):
    res = np.sum(np.abs(predicted - target) / (np.abs(predicted) + np.abs(target)
    return res / len(predicted) * 100

def sMAPE_result(predicted_rougher, target_rougher, predicted_final, target_final
    sMAPE_rougher = sMAPE(predicted_rougher, target_rougher)
    sMAPE_final = sMAPE(predicted_final, target_final)
    return .25 * sMAPE_rougher + .75 * sMAPE_final

sMAPE_score = make_scorer(sMAPE, greater_is_better=False)
```

3.2 Обучение моделей и оценка их качеств кросс-валидацией. Выбор лучшей модели и её проверка на тестовой выборке.

B [26]:
```
features_train = train.df.drop([target_col_1, target_col_2], axis=1).reset_index(
target_train = train.df[[target_col_1, target_col_2]].reset_index(drop=True)

features_test = test.df.drop([target_col_1, target_col_2], axis=1).reset_index(dr
target_test = test.df[[target_col_1, target_col_2]].reset_index(drop=True)
```

B [27]:
```
print(features_train.shape, target_train.shape)
print(features_test.shape, target_test.shape)
```

```
(16860, 52) (16860, 2)
(5856, 52) (5856, 2)
```

Линейная регрессия

B [28]:
```
model_lin = LinearRegression()
model_lin.fit(features_train, target_train)
predicts_lin = model_lin.predict(features_test)
```

B [29]:
```
sMAPE_result(predicts_lin[:, 0], target_test.iloc[:, 0], predicts_lin[:, 1], targ
```

Out[29]:
```
9.140700746751737
```

Дерево решений

B [30]:
```
features_train_1 = train.df.drop([target_col_1], axis=1).reset_index(drop=True)
target_train_1 = train.df[target_col_1].reset_index(drop=True)
features_train_2 = train.df.drop([target_col_2], axis=1).reset_index(drop=True)
target_train_2 = train.df[target_col_2].reset_index(drop=True)

features_test_1 = test.df.drop([target_col_1], axis=1).reset_index(drop=True)
target_test_1 = test.df[target_col_1].reset_index(drop=True)
features_test_2 = test.df.drop([target_col_2], axis=1).reset_index(drop=True)
target_test_2 = test.df[target_col_2].reset_index(drop=True)
```

B [31]:
```
param_grid_tree = {
    'max_depth': [i for i in range(3, 30)]
}

model_tree = DecisionTreeRegressor(random_state=1)
model_tree_gscv = GridSearchCV(estimator=model_tree, param_grid=param_grid_tree,\
                               n_jobs=-1, cv=3, verbose=2,\
                               scoring=sMAPE_score)
model_tree_gscv.fit(features_train_1, target_train_1)
model_tree_best_model_1 = model_tree_gscv.best_estimator_
predicts_tree_1 = model_tree_best_model_1.predict(features_test_1)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.8s
[Parallel(n_jobs=-1)]: Done  81 out of  81 | elapsed:   17.2s finished
```

B [32]:
```
model_tree_gscv.fit(features_train_2, target_train_2)
model_tree_best_model_2 = model_tree_gscv.best_estimator_
predicts_tree_2 = model_tree_best_model_2.predict(features_test_2)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    3.1s
[Parallel(n_jobs=-1)]: Done  81 out of  81 | elapsed:   15.3s finished
```

B [33]:
```
sMAPE_result(predicts_tree_1, target_test_1, predicts_tree_2, target_test_2)
```

Out[33]:
```
9.045763502493083
```

Случайный лес

B [34]:
```
param_grid = {
    'bootstrap': [True],
    'max_depth': [i for i in range(3, 20, 4)],
    'max_features': [3, 7],
    'min_samples_leaf': [3, 7],
    'min_samples_split': [4, 8],
    'n_estimators': [i for i in range(20, 120, 20)]
}

model = RandomForestRegressor()
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,\
                           cv=3, verbose=2, n_jobs=-1,\
                           scoring=make_scorer(sMAPE, greater_is_better=False))
grid_search.fit(features_train_1, target_train_1)
model_forest_best_1 = grid_search.best_estimator_
predicts_forest_1 = model_forest_best_1.predict(features_test_1)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   31.6s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 600 out of 600 | elapsed:  5.4min finished
```

B [35]:
```
grid_search.fit(features_test_2, target_test_2)
model_forest_best_2 = grid_search.best_estimator_
predicts_forest_2 = model_forest_best_2.predict(features_test_2)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.1s
[Parallel(n_jobs=-1)]: Done 194 tasks      | elapsed:   18.5s
[Parallel(n_jobs=-1)]: Done 397 tasks      | elapsed:   56.2s
[Parallel(n_jobs=-1)]: Done 600 out of 600 | elapsed:  1.8min finished
```

B [36]:
```
sMAPE_result(predicts_forest_1, target_test_1, predicts_forest_2, target_test_2)
```

Out[36]:
```
8.087939651378415
```

Вывод

Лучшая модель (2 модели для двух целевых признаков) на тестовой выборке - `RandomForestRegressor`

B [37]:
```
model_forest_best_1.get_params()
```

Out[37]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 11,
 'max_features': 3,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 7,
 'min_samples_split': 4,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

B [38]:
```
model_forest_best_2.get_params()
```

Out[38]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 3,
 'max_features': 7,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 7,
 'min_samples_split': 4,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 60,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

С показетелем `sMAPE_reslt = 8.09`