Выбор локации для скважины

Оглавление

Введение

Допустим, вы работаете в добывающей компании «ГлавРосГосНефть». Нужно решить, где бурить новую скважину.

Вам предоставлены пробы нефти в трёх регионах: в каждом 10 000 месторождений, где измерили качество нефти и объём её запасов. Постройте модель машинного обучения, которая поможет определить регион, где добыча принесёт наибольшую прибыль. Проанализируйте возможную прибыль и риски техникой `Bootstrap`

Шаги для выбора локации:

- В избранном регионе ищут месторождения, для каждого определяют значения признаков (собирают характеристики для скважин: качество нефти и объём её запасов);
- Строят модель для предсказания объёма запасов в новых скважинах и оценивают объём запасов;
- Выбирают месторождения с самым высокими оценками значений. Количество месторождений зависит от бюджета компании и стоимости разработки одной скважины;

- Определяют регион с максимальной суммарной прибылью отобранных скважин (Прибыль равна суммарной прибыли отобранных месторождений)

Входные данные

Данные геологоразведки трёх регионов находятся в файлах:

- `/datasets/geo_data_0.csv`
- `/datasets/geo_data_1.csv`
- `/datasets/geo_data_2.csv`
- `id` — уникальный идентификатор скважины;
- `f0`, `f1`, `f2` — три признака точек (неважно, что они означают, но сами признаки значимы);
- `product` — объём запасов в скважине (тыс. баррелей).

Условия задачи

- Для обучения модели подходит только линейная регрессия (остальные — недостаточно предсказуемые).
- При разведке региона исследуют `500` точек, из которых с помощью машинного обучения выбирают `200` лучших для разработки.
- Бюджет на разработку скважин в регионе — `10 млрд рублей`.
- При нынешних ценах один баррель сырья приносит `450 рублей` дохода. Доход с каждой единицы продукта составляет `450 тыс. рублей`, поскольку объём указан в тысячах баррелей.
- После оценки рисков нужно оставить лишь те регионы, в которых вероятность убытков меньше `2.5%`. Среди них выбирают регион с наибольшей средней прибылью.

Данные синтетические: детали контрактов и характеристики месторождений не разглашаются.

Ход исследования

1. Загрузите и подготовьте данные. Поясните порядок действий.
2. Обучите и проверьте модель для каждого региона:
   - Разбейте данные на обучающую и валидационную выборки в соотношении `75:25`.
   - Обучите модель и сделайте предсказания на валидационной выборке.
   - Сохраните предсказания и правильные ответы на валидационной выборке.
   - Напечатайте на экране средний запас предсказанного сырья и `RMSE` модели.
   - Проанализируйте результаты.
3. Подготовьтесь к расчёту прибыли:
   - Все ключевые значения для расчётов сохраните в отдельных переменных.
   - Рассчитайте достаточный объём сырья для безубыточной разработки новой скважины. Сравните полученный объём сырья со средним запасом в каждом регионе.
   - Напишите выводы по этапу подготовки расчёта прибыли.
4. Напишите функцию для расчёта прибыли по выбранным скважинам и предсказаниям модели:
   - Выберите скважины с максимальными значениями предсказаний.
   - Просуммируйте целевое значение объёма сырья, соответствующее этим предсказаниям.
   - Рассчитайте прибыль для полученного объёма сырья.
5. Посчитайте риски и прибыль для каждого региона:
   - Примените технику `Bootstrap` с `1000` выборок, чтобы найти распределение прибыли.
   - Найдите среднюю прибыль, `95%-й` доверительный интервал и риск убытков. Убыток — это отрицательная прибыль.
   - Напишите выводы: предложите регион для разработки скважин и обоснуйте выбор.

1. Обзор данных

1.1 Импорт библиотек и датасета

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

from sklearn.metrics import mean_squared_error
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [2]:
```
# для того чтобы код работал локально и на Практикуме применим конструкцию try-except

try:
    gd_0 = pd.read_csv('/datasets/geo_data_0.csv') # для Практикума
    gd_1 = pd.read_csv('/datasets/geo_data_1.csv')
    gd_2 = pd.read_csv('/datasets/geo_data_2.csv')
except:
    gd_0 = pd.read_csv('datasets/geo_data_0.csv') # локально
    gd_1 = pd.read_csv('datasets/geo_data_1.csv')
    gd_2 = pd.read_csv('datasets/geo_data_2.csv')
```

Зададим имена датасетам.

In [3]:
```
gd_0.name = 'Регион 1'
gd_1.name = 'Регион 2'
gd_2.name = 'Регион 3'
```

1.2 Обзор данных

Для предварительного обзора данных используем заранее заготовленную функцию:

In [4]:
```
def overview(o_df):
    print(o_df.name)

    print('\nОбщий вид')
    display(o_df)

    print('\n.info()\n')
    print(o_df.info())

    print('\nЗаголовки')
    display(list(o_df.columns))

    print('\n.describe(числовых значений датафрейма)')
    display(o_df.describe())

    print('\n.describe(категориальных значений и дат)')
    display(o_df.select_dtypes(include=['object', 'datetime']).describe())
```

Регион 1

In [5]:
```
overview(gd_0)
```

```
Регион 1

Общий вид
```

	id	f0	f1	f2	product
0	txEyH	0.705745	-0.497823	1.221170	105.280062
1	2acmU	1.334711	-0.340164	4.365080	73.037750
2	409Wp	1.022732	0.151990	1.419926	85.265647
3	iJLyR	-0.032172	0.139033	2.978566	168.620776
4	Xdl7t	1.988431	0.155413	4.751769	154.036647
...	...	...	...	...	...
99995	DLsed	0.971957	0.370953	6.075346	110.744026
99996	QKivN	1.392429	-0.382606	1.273912	122.346843
99997	3rnvd	1.029585	0.018787	-1.348308	64.375443
99998	7kl59	0.998163	-0.528582	1.583869	74.040764
99999	1CWhH	1.764754	-0.266417	5.722849	149.633246

100000 rows × 5 columns

```
.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 5 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   id       100000 non-null  object 
 1   f0       100000 non-null  float64
 2   f1       100000 non-null  float64
 3   f2       100000 non-null  float64
 4   product  100000 non-null  float64
dtypes: float64(4), object(1)
memory usage: 3.8+ MB
None

Заголовки
['id', 'f0', 'f1', 'f2', 'product']
.describe(числовых значений датафрейма)
```

	f0	f1	f2	product
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	0.500419	0.250143	2.502647	92.500000
std	0.871832	0.504433	3.248248	44.288691
min	-1.408605	-0.848218	-12.088328	0.000000
25%	-0.072580	-0.200881	0.287748	56.497507
50%	0.502360	0.250252	2.515969	91.849972
75%	1.073581	0.700646	4.715088	128.564089
max	2.362331	1.343769	16.003790	185.364347

```
.describe(категориальных значений и дат)
```

	id
count	100000
unique	99990
top	fiKDv
freq	2

Регион 2

In [6]:
```
overview(gd_1)
```

```
Регион 2

Общий вид
```

	id	f0	f1	f2	product
0	kBEdx	-15.001348	-8.276000	-0.005876	3.179103
1	62mP7	14.272088	-3.475083	0.999183	26.953261
2	vyE1P	6.263187	-5.948386	5.001160	134.766305
3	KcrkZ	-13.081196	-11.506057	4.999415	137.945408
4	AHL4O	12.702195	-8.147433	5.004363	134.766305
...	...	...	...	...	...
99995	QywKC	9.535637	-6.878139	1.998296	53.906522
99996	ptvty	-10.160631	-12.558096	5.005581	137.945408
99997	09gWa	-7.378891	-3.084104	4.998651	137.945408
99998	rqwUm	0.665714	-6.152593	1.000146	30.132364
99999	relB0	-3.426139	-7.794274	-0.003299	3.179103

100000 rows × 5 columns

```
.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 5 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   id       100000 non-null  object 
 1   f0       100000 non-null  float64
 2   f1       100000 non-null  float64
 3   f2       100000 non-null  float64
 4   product  100000 non-null  float64
dtypes: float64(4), object(1)
memory usage: 3.8+ MB
None

Заголовки
['id', 'f0', 'f1', 'f2', 'product']
.describe(числовых значений датафрейма)
```

	f0	f1	f2	product
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	1.141296	-4.796579	2.494541	68.825000
std	8.965932	5.119872	1.703572	45.944423
min	-31.609576	-26.358598	-0.018144	0.000000
25%	-6.298551	-8.267985	1.000021	26.953261
50%	1.153055	-4.813172	2.011479	57.085625
75%	8.621015	-1.332816	3.999904	107.813044
max	29.421755	18.734063	5.019721	137.945408

```
.describe(категориальных значений и дат)
```

	id
count	100000
unique	99996
top	wt4Uk
freq	2

Регион 3

In [7]:
```
overview(gd_2)
```

```
Регион 3

Общий вид
```

	id	f0	f1	f2	product
0	fwXo0	-1.146987	0.963328	-0.828965	27.758673
1	WJtFt	0.262778	0.269839	-2.530187	56.069697
2	ovLUW	0.194587	0.289035	-5.586433	62.871910
3	q6cA6	2.236060	-0.553760	0.930038	114.572842
4	WPMUX	-0.515993	1.716266	5.899011	149.600746
...	...	...	...	...	...
99995	4GxBu	-1.777037	1.125220	6.263374	172.327046
99996	YKFjq	-1.261523	-0.894828	2.524545	138.748846
99997	tKPY3	-1.199934	-2.957637	5.219411	157.080080
99998	nmxp2	-2.419896	2.417221	-5.548444	51.795253
99999	V9kWn	-2.551421	-2.025625	6.090891	102.775767

100000 rows × 5 columns

```
.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 5 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   id       100000 non-null  object 
 1   f0       100000 non-null  float64
 2   f1       100000 non-null  float64
 3   f2       100000 non-null  float64
 4   product  100000 non-null  float64
dtypes: float64(4), object(1)
memory usage: 3.8+ MB
None

Заголовки
['id', 'f0', 'f1', 'f2', 'product']
.describe(числовых значений датафрейма)
```

	f0	f1	f2	product
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	0.002023	-0.002081	2.495128	95.000000
std	1.732045	1.730417	3.473445	44.749921
min	-8.760004	-7.084020	-11.970335	0.000000
25%	-1.162288	-1.174820	0.130359	59.450441
50%	0.009424	-0.009482	2.484236	94.925613
75%	1.158535	1.163678	4.858794	130.595027
max	7.238262	7.844801	16.739402	190.029838

```
.describe(категориальных значений и дат)
```

	id
count	100000
unique	99996
top	VF7Jo
freq	2

Выводы ▲

На первый взгляд данные в полном порядке.

Есть несколько дубликатов **`id`**, но их количество не превышает **`10 шт`**. что в объеме всего датасета всего **`0.01 %`** и никак не повлияет на обучение модели. Можно пренебречь их удалением.

2. Подготовка данных

2.1 Подготовка датасетов ▲

Проверим наши датасеты на полные дубликаты:

In [8]:
```
print(f'{gd_0.name}: Количество полных дубликатов в датасете = {gd_0.duplicated().sum()}')
print(f'{gd_1.name}: Количество полных дубликатов в датасете = {gd_1.duplicated().sum()}')
print(f'{gd_2.name}: Количество полных дубликатов в датасете = {gd_2.duplicated().sum()}')
```

```
Регион 1: Количество полных дубликатов в датасете = 0
Регион 2: Количество полных дубликатов в датасете = 0
Регион 3: Количество полных дубликатов в датасете = 0
```

Полные дубликаты отсутствуют.

Теперь удалим стобец **`id`**, он будет мешать обучению нашей модели, так как не несет полезную информацию о скважинах. Для того, чтобы не потерять **`id`** скважин, сохраним их в отдельных переменных. В случае необходимости их всегда можно будет прицепить обратно методом **`.join()`** через индексы строк.

In [9]:
```
gd_0_id = gd_0['id'].copy()
gd_1_id = gd_1['id'].copy()
```

```
gd_2_id = gd_2['id'].copy()
```

In [10]:
```
for i in [gd_0, gd_1, gd_2]:
    i.drop(['id'], axis=1, inplace=True)
```

2.2 Разбивка на признаки

Напишем небольшую функцию, которая разбивает на целевые и обычные признаки указанный датасет.

На вход принимает:

- датасет
- имя целевого признака

Возвращает объекты:

- `features` - датафрейм с признаками
- `target` - Series с целевыми признаками

In [11]:
```
def feat_target(df, attribute):
    features = df.drop([attribute], axis=1)
    target = df[attribute]
    return features, target
```

In [12]:
```
gd_0_features, gd_0_target = feat_target(gd_0, 'product')
gd_1_features, gd_1_target = feat_target(gd_1, 'product')
gd_2_features, gd_2_target = feat_target(gd_2, 'product')
```

In [13]:
```
display('Признаки', gd_0_features.head())
display('Целевые признаки', gd_0_target.head())
```

'Признаки'

	f0	f1	f2
0	0.705745	-0.497823	1.221170
1	1.334711	-0.340164	4.365080
2	1.022732	0.151990	1.419926
3	-0.032172	0.139033	2.978566
4	1.988431	0.155413	4.751769

```
'Целевые признаки'
0    105.280062
1     73.037750
2     85.265647
3    168.620776
4    154.036647
Name: product, dtype: float64
```

2.3 Создание обучающей и валидационной выборки

Для создания тестовой и валидационной выборки напишем функцию:

На вход принимает:

- Таблицу с признаками
- Series с целевыми признаками
- Необходимую долю валидационной выборки

Возвращает выборки в нужной пропорции:

- `features_train`
- `features_valid`
- `target_train`
- `target_valid`

In [14]:
```
def splitter(features, target, size):
    features_train, features_valid, target_train, target_valid = train_test_split(
        features, target,
        random_state=12345,
        test_size=size)
    return features_train, features_valid, target_train, target_valid
```

Применяем функцию:

In [15]:
```
gd_0_features_train, gd_0_features_valid, gd_0_target_train, gd_0_target_valid = splitter(gd_0_features, gd_0_targ
gd_1_features_train, gd_1_features_valid, gd_1_target_train, gd_1_target_valid = splitter(gd_1_features, gd_1_targ
gd_2_features_train, gd_2_features_valid, gd_2_target_train, gd_2_target_valid = splitter(gd_2_features, gd_2_targ
```

Проверим верность разбивки:

In [16]:
```
print(gd_0_features_train.shape, gd_0_features_valid.shape, gd_0_target_train.shape, gd_0_target_valid.shape)
print(gd_1_features_train.shape, gd_1_features_valid.shape, gd_1_target_train.shape, gd_1_target_valid.shape)
print(gd_2_features_train.shape, gd_2_features_valid.shape, gd_2_target_train.shape, gd_2_target_valid.shape)
```

```
(75000, 3) (25000, 3) (75000,) (25000,)
(75000, 3) (25000, 3) (75000,) (25000,)
(75000, 3) (25000, 3) (75000,) (25000,)
```

3. Обучение и проверка модели

Для обучения выбераем **`Линейную регрессию`**

In [17]:
```
model = LinearRegression()
```

3.1 Предсказания для валидационной выборки - Регион 1 ▲

In [18]:
```
model.fit(gd_0_features_train, gd_0_target_train) # обучаем
gd_0_predicted_valid = model.predict(gd_0_features_valid) # предсказываем
rmse_gd_0 = mean_squared_error(gd_0_target_valid, gd_0_predicted_valid) ** 0.5 # считаем RMSE

# Сохраняем наши предсказания в отдельный датафрейм,
# вместе с целевыми признаками и их индексами
gd_0_target_predict = pd.DataFrame(data={'target': gd_0_target_valid, 'predicted': gd_0_predicted_valid})

display('Регион 1 - Валидационная выборка', gd_0_target_predict)
```

```
'Регион 1 - Валидационная выборка'
```

	target	predicted
71751	10.038645	95.894952
80493	114.551489	77.572583
2655	132.603635	77.892640
53233	169.072125	90.175134
91141	122.325180	70.510088
...	...	...
12581	170.116726	103.037104
18456	93.632175	85.403255
73035	127.352259	61.509833
63834	99.782700	118.180397
43558	177.821022	118.169392

25000 rows × 2 columns

In [19]:
```
print('Регион 1 - Валидационная выборка')
print('Средний запас предсказанного сырья:', gd_0_target_predict['predicted'].mean())
print('RMSE =', rmse_gd_0)
```

```
Регион 1 - Валидационная выборка
Средний запас предсказанного сырья: 92.59256778438005
```

```
RMSE = 37.5794217150813
```

3.2 Предсказания для валидационной выборки - Реигион 2

In [20]:
```
model.fit(gd_1_features_train, gd_1_target_train)
gd_1_predicted_valid = model.predict(gd_1_features_valid)
rmse_gd_1 = mean_squared_error(gd_1_target_valid, gd_1_predicted_valid) ** 0.5

gd_1_target_predict = pd.DataFrame(data={'target': gd_1_target_valid, 'predicted': gd_1_predicted_valid})

display('Регион 2 - Валидационная выборка', gd_1_target_predict)
```

```
'Регион 2 - Валидационная выборка'
```

	target	predicted
71751	80.859783	82.663314
80493	53.906522	54.431786
2655	30.132364	29.748760
53233	53.906522	53.552133
91141	0.000000	1.243856
...	...	...
12581	137.945408	136.869211
18456	110.992147	110.693465
73035	137.945408	137.879341
63834	84.038886	83.761966
43558	53.906522	53.958466

25000 rows × 2 columns

In [21]:
```
print('Регион 2 - Валидационная выборка')
print('Средний запас предсказанного сырья:', gd_1_target_predict['predicted'].mean())
print('RMSE =', rmse_gd_1)
```

```
Регион 2 - Валидационная выборка
Средний запас предсказанного сырья: 68.7285468954458
RMSE = 0.8930992867756168
```

3.3 Предсказания для валидационной выборки - Регион 3

In [22]:
```
model.fit(gd_2_features_train, gd_2_target_train)
gd_2_predicted_valid = model.predict(gd_2_features_valid)
rmse_gd_2 = mean_squared_error(gd_2_target_valid, gd_2_predicted_valid) ** 0.5

gd_2_target_predict = pd.DataFrame(data={'target': gd_2_target_valid, 'predicted': gd_2_predicted_valid})

display('Регион 3 - Валидационная выборка', gd_2_target_predict)
```

```
'Регион 3 - Валидационная выборка'
```

	target	predicted
71751	61.212375	93.599633
80493	41.850118	75.105159
2655	57.776581	90.066809
53233	100.053761	105.162375
91141	109.897122	115.303310
...	...	...
12581	28.492402	78.765887
18456	21.431303	95.603394
73035	125.487229	99.407281
63834	99.422903	77.779912

	target	predicted
43558	127.445075	129.032417

25000 rows × 2 columns

In [23]:
```
print('Регион 3 - Валидационная выборка')
print('Средний запас предсказанного сырья:', gd_2_target_predict['predicted'].mean())
print('RMSE =', rmse_gd_2)
```

```
Регион 3 - Валидационная выборка
Средний запас предсказанного сырья: 94.96504596800504
RMSE = 40.02970873393434
```

Выводы ▲

Посмотрим результаты полученные на валидационных выборках:

Локация	Средний запас сырья	RMSE
Регион 1	92.59	37.579
Регион 2	68.72	0.893
Регион 3	94.96	40.029

В **Регионе 2** предсказанный средний запас сырья немного меньше чем у 1 и 3 региона. А вот ошибка предсказания меньше в десятки раз. Посмотрим дальше как это отразится на результатах исследования.

4. Подготовка к расчёту прибыли

4.1 Создание переменных ▲

In [24]:
```
budget = 10000000000
income_thousand_barrel = 450000
number_of_wells_to_develop = 200
minimum_count_of_product = budget / income_thousand_barrel
minimum_mean_count_of_product = budget / (income_thousand_barrel * number_of_wells_to_develop)
```

In [25]:
```
print(f'Бюджет на разработку месторождения: {budget / 1000000000:.1f} млрд. руб.')
print(f'Доход с единицы продукта: {income_thousand_barrel / 1000} тыс. руб.')
print(f'Количество скважин на разработку: {number_of_wells_to_develop} шт.')
print(f'Минимальный объем сырья для безубыточной разработки: {minimum_count_of_product:.2f} тыс. баррелей')
print(f'Средний минимальный объем сырья для безубыточной разработки: {minimum_mean_count_of_product:.2f} тыс. барр
```

```
Бюджет на разработку месторождения: 10.0 млрд. руб.
Доход с единицы продукта: 450.0 тыс. руб.
Количество скважин на разработку: 200 шт.
Минимальный объем сырья для безубыточной разработки: 22222.22 тыс. баррелей
Средний минимальный объем сырья для безубыточной разработки: 111.11 тыс. баррелей
```

4.2 Расчет объемов сырья ▲

Посчитаем средний объем сырья в одной точке, для каждого региона.

In [26]:
```
print(f'Регион 1: Средний объем сырья с одной точки: {gd_0["product"].mean():.2f} тыс. баррелей')
print(f'Регион 2: Средний объем сырья с одной точки: {gd_1["product"].mean():.2f} тыс. баррелей')
print(f'Регион 3: Средний объем сырья с одной точки: {gd_2["product"].mean():.2f} тыс. баррелей')
print(f'Средний объем сырья с одной точки для безубыточной разработки: {minimum_mean_count_of_product:.2f} тыс. ба
```

```
Регион 1: Средний объем сырья с одной точки: 92.50 тыс. баррелей
Регион 2: Средний объем сырья с одной точки: 68.83 тыс. баррелей
Регион 3: Средний объем сырья с одной точки: 95.00 тыс. баррелей
Средний объем сырья с одной точки для безубыточной разработки: 111.11 тыс. баррелей
```

Выводы ▲

Как мы видим, средний запас сырья в одной точке меньше, чем среднее значение безубыточного запаса. Это означает, что если мы будем бурить скважины наугад, то практически гарантированно получим убытки.

5. Расчёт прибыли

Для начала найдем предсказания для всего датасета, и добавим эту информацию в исходную таблицу.

5.1 Предсказания для всего датасета - Регион 1

In [27]:
```
model.fit(gd_0_features_train, gd_0_target_train) # обучаем модель на тренировочных данных
gd_0_predicted_full = model.predict(gd_0_features) # предсказываем значения для всего датасета
rmse_gd_0_full = mean_squared_error(gd_0_target, gd_0_predicted_full) ** 0.5

gd_0['predicted'] = gd_0_predicted_full # создаем дополнительный столбец с предсказаниями
```

In [28]:
```
print('Регион 1 - Весь датасет')
print('RMSE =', rmse_gd_0_full)
display(gd_0)
```

```
Регион 1 - Весь датасет
RMSE = 37.69240596744535
```

	f0	f1	f2	product	predicted
0	0.705745	-0.497823	1.221170	105.280062	95.461973
1	1.334711	-0.340164	4.365080	73.037750	116.227394
2	1.022732	0.151990	1.419926	85.265647	88.750254
3	-0.032172	0.139033	2.978566	168.620776	95.419237
4	1.988431	0.155413	4.751769	154.036647	114.138969
...	...	...	...	...	...
99995	0.971957	0.370953	6.075346	110.744026	116.174856
99996	1.392429	-0.382606	1.273912	122.346843	96.652519
99997	1.029585	0.018787	-1.348308	64.375443	72.401340
99998	0.998163	-0.528582	1.583869	74.040764	99.337548
99999	1.764754	-0.266417	5.722849	149.633246	125.684762

100000 rows × 5 columns

5.2 Предсказания для всего датасета - Регион 2

In [29]:
```
model.fit(gd_1_features_train, gd_1_target_train)
gd_1_predicted_full = model.predict(gd_1_features)
rmse_gd_1_full = mean_squared_error(gd_1_target, gd_1_predicted_full) ** 0.5

gd_1['predicted'] = gd_1_predicted_full
```

In [30]:
```
print('Регион 2 - Весь датасет')
print('RMSE =', rmse_gd_1_full)
display(gd_1)
```

```
Регион 2 - Весь датасет
RMSE = 0.890380265029757
```

	f0	f1	f2	product	predicted
0	-15.001348	-8.276000	-0.005876	3.179103	3.853530
1	14.272088	-3.475083	0.999183	26.953261	26.592376
2	6.263187	-5.948386	5.001160	134.766305	135.665691
3	-13.081196	-11.506057	4.999415	137.945408	138.544872
4	12.702195	-8.147433	5.004363	134.766305	134.867164
...	...	...	...	...	...
99995	9.535637	-6.878139	1.998296	53.906522	54.281135
99996	-10.160631	-12.558096	5.005581	137.945408	138.310908
99997	-7.378891	-3.084104	4.998651	137.945408	137.512258

	f0	f1	f2	product	predicted
99998	0.665714	-6.152593	1.000146	30.132364	28.649447
99999	-3.426139	-7.794274	-0.003299	3.179103	2.234663

100000 rows × 5 columns

5.3 Предсказания для всего датасета - Регион 3

In [31]:
```
model.fit(gd_2_features_train, gd_2_target_train)
gd_2_predicted_full = model.predict(gd_2_features)
rmse_gd_2_full = mean_squared_error(gd_2_target, gd_2_predicted_full) ** 0.5

gd_2['predicted'] = gd_2_predicted_full
```

In [32]:
```
print('Регион 3 - Весь датасет')
print('RMSE =', rmse_gd_2_full)
display(gd_2)
```

```
Регион 3 - Весь датасет
RMSE = 40.055624818752136
```

	f0	f1	f2	product	predicted
0	-1.146987	0.963328	-0.828965	27.758673	75.968506
1	0.262778	0.269839	-2.530187	56.069697	66.329664
2	0.194587	0.289035	-5.586433	62.871910	48.880039
3	2.236060	-0.553760	0.930038	114.572842	86.178035
4	-0.515993	1.716266	5.899011	149.600746	114.363434
...	...	...	...	...	...
99995	-1.777037	1.125220	6.263374	172.327046	116.429156
99996	-1.261523	-0.894828	2.524545	138.748846	95.186430
99997	-1.199934	-2.957637	5.219411	157.080080	110.658478
99998	-2.419896	2.417221	-5.548444	51.795253	48.926897
99999	-2.551421	-2.025625	6.090891	102.775767	115.552517

100000 rows × 5 columns

5.4 Расчет масксимально возможной прибыли

Теперь рассчитаем максимальную прибыль с 200 точек с самым большим запасом сырья на предсказанных данных.

Для этого напишем фукнцию, которая принимает на вход:

- Датафрейм с предсказаниями

Возвращает:

- Максимальную прибыль с 200 точек на предсказанных данных

In [33]:
```
def max_profit_calc(df):
    profit = df['predicted'].sort_values(ascending=False).head(200).sum() * income_thousand_barrel - budget
    return profit
```

In [34]:
```
print(f'Регион 1: Предсказанная максимально возможная прибыль: {max_profit_calc(gd_0) / 1000000:.3f}, млн руб.')
print(f'Регион 2: Предсказанная максимально возможная прибыль: {max_profit_calc(gd_1) / 1000000:.3f}, млн руб.')
print(f'Регион 3: Предсказанная максимально возможная прибыль: {max_profit_calc(gd_2) / 1000000:.3f}, млн руб.')
```

```
Регион 1: Предсказанная максимально возможная прибыль: 4689.596, млн руб.
Регион 2: Предсказанная максимально возможная прибыль: 2524.606, млн руб.
Регион 3: Предсказанная максимально возможная прибыль: 4063.786, млн руб.
```

Выводы

Максимальная прибыль для 1 и 3 региона сильно превышает аналогичный показатель для 2 региона. Однако мы помним, что показатель ошибок предсказания для 1 и 3 региона был выше практически в 40 раз. Оценим связанные с этим риски.

6. Подсчет рисков

6.1 Функция для расчета показателей

Напишем функцию, которая будет считать **среднюю прибыль**, **95%** доверительный интервал, а так же **вероятность убытков**, с помощью техники **Bootstrap** на **1000** выборках.

In [35]:
```
def calc(df):
    target = df['product']
    predict = df['predicted']

    # функция принимает на вход целевые признаки и предсказанные значения, а так же количество
    # строк, для которых необходимо посчитать прибыль
    def profit(target, predict, count):
        predict_sorted = predict.sort_values(ascending=False) # сортируем количество предсказанного сырья по убыва
        selected = target[predict_sorted.index][:count] # оставляем 200 первых целевых признаков
        return (income_thousand_barrel * selected.sum()) - budget # считаем прибыль для этих 200 строк

    state = np.random.RandomState(12345) # фиксируем RandomState для воспроизводимости эксперимента

    values = [] # пустой список для значений прибыли
    for i in range(1000):
        target_subsample = target.sample(n=500, replace=True, random_state=state) # Берем случайную выборку из 500
        predict_subsample = predict[target_subsample.index] # создаем из этой выборки по индексам выборку предсказ
        res = profit(target_subsample, predict_subsample, 200) # считаем прибыль для 200 строк с самым большим зап
        values.append(res) # добавляем значение прибыли в заготовленный список
        # и так 1000 раз

    values = pd.Series(values) # Переводим список в Series

    lower = values.quantile(0.025) / 1000000 # Считаем 2.5% квантиль
    upper = values.quantile(0.975) / 1000000 # Считаем 97.5% квантиль
    mean = values.mean() / 1000000 # считаем среднюю прибыль

    print(df.name)
    print(f'Средняя прибыль: {mean:.3f} млн руб.')
    print(f'2.5% - квантиль: {lower:.3f} млн руб.')
    print(f'97.5% - квантиль: {upper:.3f} млн руб.')
    print(f'Вероятность убытков = {values[values < 0].count() / values.shape[0]:.1%}') # считаем вероятность убытк
```

6.2 Оценка рисков и выбор региона

Прогоним функцией все три региона:

In [36]:
```
calc(gd_0)
```

```
Регион 1
Средняя прибыль: 431.092 млн руб.
2.5% - квантиль: -70.542 млн руб.
97.5% - квантиль: 979.020 млн руб.
Вероятность убытков = 5.1%
```

In [37]:
```
calc(gd_1)
```

```
Регион 2
Средняя прибыль: 462.957 млн руб.
2.5% - квантиль: 55.989 млн руб.
97.5% - квантиль: 845.839 млн руб.
Вероятность убытков = 1.3%
```

In [38]:
```
calc(gd_2)
```

```
Регион 3
Средняя прибыль: 380.554 млн руб.
2.5% - квантиль: -177.061 млн руб.
97.5% - квантиль: 896.985 млн руб.
Вероятность убытков = 9.1%
```

Вероятность убытков во втором регионе составила `1.3%` что полностью нам подходит. Регионы 1 и 3 не смотря на большие запасы сырья наоборот, не преодолели порога в `2.5%` и отбрасываются из-за высокого риска убытков при разработке.

Итого, для разработки мы выбираем 2 регион, так как шанс получить убытки минимальны. Вот его характеристики:

- Средняя прогнозируемая прибыль: **`462.957 млн руб.`**
- 95% доверительный интервал составляет: от **`55.989`** до **`845.839 млн руб`**.
- Вероятность убытков: **`1.3%`**

6.3 Выбор скважин

После выбора региона отберем 200 самых перспективных скважин.

Для начала вернем **`id`** скважин в исходный датафрейм.

In [39]:
```
gd_1_full = gd_1.join(gd_1_id)
```

In [40]:
```
gd_1_full.head()
```

Out[40]:

	f0	f1	f2	product	predicted	id
0	-15.001348	-8.276000	-0.005876	3.179103	3.853530	kBEdx
1	14.272088	-3.475083	0.999183	26.953261	26.592376	62mP7
2	6.263187	-5.948386	5.001160	134.766305	135.665691	vyE1P
3	-13.081196	-11.506057	4.999415	137.945408	138.544872	KcrkZ
4	12.702195	-8.147433	5.004363	134.766305	134.867164	AHL4O

Теперь найдем 200 скважин с наибольшим запасом сырья.

In [41]:
```
wells_for_development = gd_1_full.sort_values('predicted', ascending=False).head(200)
```

Итоговый перечень скважин:

In [42]:
```
display(wells_for_development)
```

	f0	f1	f2	product	predicted	id
26531	-26.646255	-1.531112	5.001941	137.945408	140.359367	yLbdW
80439	-23.884180	-3.773158	5.001008	137.945408	139.983257	kpPCd
55165	-22.107811	-5.342910	5.006254	137.945408	139.901759	wcKNk
42738	-19.384167	-15.314765	5.010972	137.945408	139.853808	XLXvP
38665	-19.348001	-14.265995	5.010731	137.945408	139.818970	vz1OW
...	...	...	...	...	...	...
74361	-16.945577	-7.769416	4.994912	137.945408	138.901322	XkDo3
41969	-16.670778	-5.580829	4.998175	137.945408	138.901202	qhtWm
31949	-16.982960	-3.750930	4.997991	137.945408	138.901184	4esiy
71942	-15.922463	-0.995106	5.005805	137.945408	138.897365	TQoii
96878	-15.280313	-8.946434	5.002709	137.945408	138.895998	7dLVk

200 rows × 6 columns

Выводы

Мы рассмотрели 3 региона для разработки скважин.

Входные данные, оказались достаточно чистыми и мы смогли сразу приступить к анализу показателей и постройке моделей.

При проверке моделей на валидационной выборке мы выяснили следующее:

- В регионе 1 и 3 средний предсказанный запас сырья почти на треть выше чем в регионе 2. Однако при этом и RMSE в десятки раз выше чем во втором регионе
- Во 2 регионе напротив, RMSE оказался крайне малым. Несмотря на меньший средний объем сырья, регион выглядел более перспективным.

Так же мы выяснили, что бурение в случайном месте практически гарантированно принесет нам убытки, так как среднй объем сырья для безубыточной разработки был ощутимо выше чем просто среднее количество сырья в любом из регионов.

После подсчета риска убытков мы получили следующие значения:

- Регион 1: 5.1%
- Регион 2: 1.3%
- Регион 3: 9.1%

В результате чего, для разработки был выбран 2 регион. Не смотря на более скромные запасы сырья, вероятность получить прибыль при его разработке намного выше чем у 1 и 3 региона.