Описание проекта

Для добывающей компании «ГлавРосГосНефть» необходимо выяснить, где бурить новую скважину.

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

Шаги для выбора локации:

- В избранном регионе ищут месторождения, для каждого определяют значения признаков;
- Строят модель и оценивают объём запасов;
- Выбирают месторождения с самым высокими оценками значений. Количество месторождений зависит от бюджета компании и стоимости разработки одной скважины;
- Прибыль равна суммарной прибыли отобранных месторождений.

Описание данных

id — уникальный идентификатор месторождения;

f0, f1, f2 — три признака точек (неважно, что они означают, но сами признаки значимы);

product — объём запасов в месторождении (тыс. баррелей).

Условия задачи:

Для обучения модели подходит только линейная регрессия (остальные — недостаточно предсказуемые).

При разведке региона проводится исследование 500 точек.

Бюджет на разработку месторождений — 10 млрд рублей, стоимость бурения одной скважины — 50 млн рублей.

Один баррель сырья приносит 4500 рублей прибыли.

Не рассматривать регионы, в которых риск убытков выше 2.5%. Из оставшихся выбирается регион с наибольшей средней прибылью.

1. Загрузка и подготовка данных

In [67]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split #импортируем функцию train_test_split из библиотеки sklearn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from scipy import stats as st
from scipy.stats import t
import warnings
warnings.filterwarnings('ignore')
```

In [68]:
```
df_0=pd.read_csv('/datasets/geo_data_0.csv') #прочитаем файлы
df_1=pd.read_csv('/datasets/geo_data_1.csv')
df_2=pd.read_csv('/datasets/geo_data_2.csv')
```

In [69]:
```
df=[df_0,df_1,df_2]
```

In [70]:
```
for k in df:
    print('Для региона')
    print("")
    print(k.info())
    print(k.head(100)) #изучим датасеты
```

```
Для региона

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 5 columns):
id         100000 non-null object
f0         100000 non-null float64
f1         100000 non-null float64
f2         100000 non-null float64
product    100000 non-null float64
dtypes: float64(4), object(1)
memory usage: 3.8+ MB
None
       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
..    ...       ...       ...       ...         ...
95  U2v01 -0.299399  1.080699  1.296372   20.701885
96  6bstl  0.332135 -0.140657  2.808669   63.864110
97  PfuFQ  1.682084 -0.139053  2.703813  168.087330
98  D1vZK -1.072434  0.350963  5.353898  114.833418
99  vT7FD  1.929955  0.116360 -1.044324  158.819062

[100 rows x 5 columns]
Для региона

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 5 columns):
id         100000 non-null object
f0         100000 non-null float64
f1         100000 non-null float64
f2         100000 non-null float64
product    100000 non-null float64
dtypes: float64(4), object(1)
memory usage: 3.8+ MB
None
       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
..    ...        ...        ...       ...         ...
95  9plKW  15.411156   1.117979  1.002825   26.953261
96  FFBRr  -9.631558  -3.142134 -0.000449    3.179103
97  NJgyC  -7.317779  -6.071897  4.006200  110.992147
98  0m7Gs  11.877073   4.792457  3.001572   80.859783
99  n93UD  18.452972  -8.879619  4.003719  107.813044

[100 rows x 5 columns]
Для региона

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 5 columns):
id         100000 non-null object
f0         100000 non-null float64
f1         100000 non-null float64
f2         100000 non-null float64
product    100000 non-null float64
dtypes: float64(4), object(1)
memory usage: 3.8+ MB
None
       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
..    ...       ...       ...       ...         ...
95  9v3QN -2.133221  0.362536  5.722595  168.507922
96  a0qWE -2.227204  0.624627 -0.804711   77.504331
97  0oX8v -3.375801 -2.021755  3.999755  108.882155
98  opNAr  0.058635 -1.347149  3.400361   81.555711
99  QMk6k  2.490791 -3.454526  1.841053   40.063656

[100 rows x 5 columns]
```

In [71]:
```
def features_target(df): #создадим функцию для присвоения признаков и целевого значения
    features=df.drop(['id','product'], axis=1)
    target=df['product']
    return features, target
```

In [72]:
```
features_0, target_0=features_target(df_0) #присвоим признакам и цел показателю соотвествующий номер
features_1, target_1=features_target(df_1)
features_2, target_2=features_target(df_2)
```

In [73]:
```
def split(x,y): #функция для разделения выборки, которая возвращает значения для разделенных выборок признаков и целев признака
    x_train, x_valid, y_train, y_valid = train_test_split(x,y, test_size=0.25, random_state=12345)
    return x_train, x_valid, y_train, y_valid
```

In [74]:
```
features_train_0, features_valid_0, target_train_0, target_valid_0 = split(features_0, target_0)
```

In [75]:
```
features_train_0 # посмотрим как выглядит обучающая выборка первого региона
```

Out[75]:

	f0	f1	f2
27212	0.022450	0.951034	2.197333
7866	1.766731	0.007835	6.436602
62041	0.724514	0.666063	1.840177
70185	-1.104181	0.255268	2.026156
82230	-0.635263	0.747990	6.643327
...	...	...	...
4094	1.863680	-0.298123	1.621324
85412	-1.162682	-0.014822	6.819941
2177	0.862688	-0.403776	1.867662
77285	0.846235	-0.489533	1.058786
86498	2.019850	0.263887	11.497239

75000 rows × 3 columns

In [76]:
```
features_train_1, features_valid_1, target_train_1, target_valid_1 = split(features_1, target_1)
```

In [77]:
```
features_train_2, features_valid_2, target_train_2, target_valid_2 = split(features_2, target_2)
```

In [78]:
```
#Произведем масштабирование
#создадим функцию для масштабирования числовых данных, возвращающую отмасштабированные выборки признаков

def scaler (features_train, features_valid):
    numeric = ['f0', 'f1', 'f2']
    scaler = StandardScaler()
    scaler.fit(features_train[numeric])
    features_train[numeric] = scaler.transform(features_train[numeric])
    features_valid[numeric] = scaler.transform(features_valid[numeric])
    return features_train,features_valid
```

In [79]:
```
#применим для выборок каждого региона
features_train_0,features_valid_0=scaler(features_train_0, features_valid_0)
features_train_1,features_valid_1=scaler(features_train_1, features_valid_1)
features_train_2,features_valid_2=scaler(features_train_2, features_valid_2)
```

In [80]:
```
#сравним с тем, что было до масштабирования
features_train_0
```

Out[80]:

	f0	f1	f2
27212	-0.544828	1.390264	-0.094959
7866	1.455912	-0.480422	1.209567
62041	0.260460	0.825069	-0.204865
70185	-1.837105	0.010321	-0.147634
82230	-1.299243	0.987558	1.273181
...	...	...	...
4094	1.567114	-1.087243	-0.272211
85412	-1.904207	-0.525360	1.327530
2177	0.418949	-1.296788	-0.196407
77285	0.400077	-1.466874	-0.445317
86498	1.746246	0.027415	2.766848

75000 rows × 3 columns

Вывод:

Данные выгружены, прошли масштабирование числовых значений, разделили на выборки, присвоили значения признакам.

2. Обучение и проверка модели

In [81]:
```
#создадим функцию, которая обучает линейную регрессию,
#находит средний запас сырья в регионе, расчитывает качество модели
#возвращает предсказания и правильные ответы валидационной выборки
def log_reg(features_train, features_valid, target_train, target_valid):
    model = LinearRegression()
    model.fit(features_train, target_train)
    predicted_valid = model.predict(features_valid)
    predicted_mean = predicted_valid.mean()
    print('Средний запас сырья', predicted_mean)
    rmse = mean_squared_error(target_valid, predicted_valid)**0.5
    print('Квадратный корень из средней квадратичной ошибки модели =', rmse)
    r2 = r2_score(target_valid,predicted_valid)
    print("Коэффициент детерминации =", r2)
    mae=mean_absolute_error(target_valid, predicted_valid)
    print("Среднее абсолютное отклонение =", mae)
    return predicted_valid, target_valid, predicted_mean,rmse,r2, mae
```

In [82]:
```
predicted_valid_0, target_valid_0, predicted_mean_0,rmse_0,r2_0, mae_0=log_reg (features_train_0, features_valid_0, target_train_0, target_valid_0)
```

```
Средний запас сырья 92.59256778438038
Квадратный корень из средней квадратичной ошибки модели = 37.5794217150813
Коэффициент детерминации = 0.27994321524487786
Среднее абсолютное отклонение = 30.919600777151313
```

In [83]:
```
predicted_valid_1, target_valid_1, predicted_mean_1,rmse_1,r2_1,mae_1=log_reg (features_train_1, features_valid_1, target_train_1, target_valid_1)
```

```
Средний запас сырья 68.728546895446
Квадратный корень из средней квадратичной ошибки модели = 0.8930992867756158
Коэффициент детерминации = 0.9996233978805127
Среднее абсолютное отклонение = 0.718766244212475
```

In [84]:
```
predicted_valid_2, target_valid_2, predicted_mean_2,rmse_2,r2_2,mae_2=log_reg (features_train_2, features_valid_2, target_train_2, target_valid_2)
```

```
Средний запас сырья 94.96504596800489
Квадратный корень из средней квадратичной ошибки модели = 40.02970873393434
Коэффициент детерминации = 0.20524758386040443
Среднее абсолютное отклонение = 32.792652105481814
```

Вывод

модель 1 и 3 переоценивает объем запасов. модель 1 в среднем ошибается на 30,9 тыс. баррелей, модель 3 ошибается на 32,7 тыс. баррелей. Вторая модель самая точная. Средний запас 68,7 тыс. баррелей.

3. Подготовка к расчёту прибыли

Исходя из условий задачи рассчитаем минимальный средний объём сырья в месторождениях региона, достаточный для его разработки. Стоимость бурения одной скважины — 50 млн рублей. Один баррель сырья приносит 4500 рублей прибыли.

In [85]:
```
#Сохраним в коде все ключевые значения для расчётов

budget=10000000
expan_one=50000
one_barrel=4.5
```

In [86]:
```
def revenue(target_valid, predicted_valid, count):
    sorted_predicted = predicted_valid.sort_values(ascending=False)
    selected = target_valid[sorted_predicted.index][:count]
    total_expan=expan_one*count
    revenue=total_expan-one_barrel*selected.sum()
    return revenue
```

In [87]:
```
from numpy.random import RandomState
state = np.random.RandomState(12345)

def bootstrap_min_barrels (target_valid, predicted_valid):

    values = []

    for i in range(1000):
        predicted_valid=pd.Series(predicted_valid)
        target_valid=pd.Series(target_valid)
        target_subsample=target_valid.sample(n=500,replace=True,random_state=state)
        predicted_subsample=predicted_valid[target_subsample.index]
        values.append(revenue(target_subsample,predicted_subsample,200))

    values = pd.Series(values)
    lower = values.quantile(0.025)
    mean_revenue = values.mean()
    budget=10000000
    print("2.5%-квантиль:", lower)
    if values.mean()>budget:
        return print("Регион рассматриваем, риск убытков меньше 2.5%")
```

In [88]:
```
bootstrap_min_barrels(target_valid_0, predicted_valid_0)
```

```
2.5%-квантиль: 9911678.913158959
```

In [89]:
```
bootstrap_min_barrels(target_valid_1, predicted_valid_1)
```

```
2.5%-квантиль: 9932394.30160146
```

In [90]:
```
bootstrap_min_barrels(target_valid_2, predicted_valid_2)
```

```
2.5%-квантиль: 9909186.18235849
```

У всех регионов значение 2,5% квантиля позволяет сделать вывод о безубыточности регионов.

4. Расчёт прибыли и рисков

In [91]:
```
from numpy.random import RandomState
state = np.random.RandomState(12345)

def bootstrap (target_valid, predicted_valid):

    values = []

    for i in range(1000):
        predicted_valid=pd.Series(predicted_valid)
        target_valid=pd.Series(target_valid)
        target_subsample=target_valid.sample(n=500,replace=True,random_state=state)
        predicted_subsample=predicted_valid[target_subsample.index]
        values.append(revenue(target_subsample,predicted_subsample,200))

    values = pd.Series(values)
    lower = values.quantile(0.025)
    upper= values.quantile(0.975)
    mean_revenue = values.mean()
    confidence_interval = st.t.interval(0.95, len(values)-1, values.mean(), values.sem())

    print("Средняя выручка:", mean_revenue)
    print("95%-ый доверительный интервал:", confidence_interval)
    return mean_revenue, confidence_interval
```

In [92]:
```
mean_revenue_0, confidence_interval_0= bootstrap (target_valid_0, predicted_valid_0)
```

```
Средняя выручка: 9917270.74054416
95%-ый доверительный интервал: (9917090.642103251, 9917450.83898507)
```

In [93]:
```
mean_revenue_1, confidence_interval_1= bootstrap (target_valid_1, predicted_valid_1)
```

```
Средняя выручка: 9938115.14804936
95%-ый доверительный интервал: (9937931.357645914, 9938298.938452806)
```

In [94]:
```
mean_revenue_2, confidence_interval_2= bootstrap (target_valid_2, predicted_valid_2)
```

```
Средняя выручка: 9914804.177735426
95%-ый доверительный интервал: (9914619.846902106, 9914988.508568745)
```

Вывод:

Для обучения модели была использована линейная регрессия (остальные — недостаточно предсказуемые). В результате анализа 3х регионов методом bootstrap лучшие показатели демонстрирует регион №2 2 и по средней выручке, и по 95%-й доверительному интервалу.