```
!pip install boruta
```

```
Collecting boruta
  Downloading Boruta-0.4.3-py3-none-any.whl.metadata (8.8 kB)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.11/dist-packages (from boruta) (2.0.2)
Requirement already satisfied: scikit-learn>=0.17.1 in /usr/local/lib/python3.11/dist-packages (from boruta) (1.6.1)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.11/dist-packages (from boruta) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.17.1->boru
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.17.
Downloading Boruta-0.4.3-py3-none-any.whl (57 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.9/57.9 kB 3.1 MB/s eta 0:00:00
Installing collected packages: boruta
Successfully installed boruta-0.4.3
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from boruta import BorutaPy
from itertools import combinations
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.linear_model import Ridge, Lasso, LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, ExtraTreesRegressor
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, roc_auc_score, accuracy_score, mean_squared_error, r2_score
```

```
df = pd.read_excel('/content/Параметры_для_многомерной_регрессии_Добычные_параметры_20250624.xlsx', index_col=0, header=2)
```

```
df
```

№	Скважины	Qж, м3/сут	Qн, т/сут	Qв, м3/сут	Обв, %	ГФ, м3/т	Кпрод	Кратность	Выборка по кратности > 40	Выборка по кратности > 70	...	Средняя .1	Линейная по MD.1
1	202_1_h	84.69	69.5960	0.25407	0.3	12.156	27.000000	31.631	0	0	...	NaN	NaN
2	202_1_zbs	75.00	43.1250	2.25000	3.0	229.350	2.000000	32.640	0	0	...	NaN	NaN
3	237_1_h	229.86	179.2793	11.49300	5.0	194.000	35.978370	29.693	0	0	...	NaN	NaN
4	272_1_h	316.10	233.2430	3.47710	1.1	194.000	191.877689	35.421	0	0	...	NaN	NaN
5	782_1_h	13.00	79.4892	0.78000	6.0	197.846	24.000000	25.391	0	0	...	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
271	657_53_h2	95.50	41.5549	44.88500	47.0	499.000	25.000000	62.755	1	0	...	NaN	NaN
272	362_54_h	121.00	41.6239	70.30100	58.1	372.670	13.000000	59.706	1	0	...	0.00	0.2286
273	413_54_h	94.30	68.9410	10.37300	11.0	565.627	15.000000	93.852	1	1	...	0.00	0.2730
274	378_57_h	29.20	23.8533	0.14600	0.5	13312.000	2.200000	73.766	1	1	...	0.01	0.3466

```
print(df.columns.tolist())
```

```
['Скважины', 'Qж, м3/сут', 'Qн, т/сут', 'Qв, м3/сут', 'Обв, %', 'ГФ, м3/т', 'Кпрод', 'Кратность ', 'Выборка по кратности
```

```
df = df.drop(columns=[
    "№", "Кпрод", "Qн, т/сут", "Qв, м3/сут", "Обв, %", "ГФ, м3/т",
    "Кратность ", "Выборка по кратности > 40", "Выборка по кратности > 70",
    "Снятие значений в рифее",

    # Геометрия коллектора
    "Общая проходка", "Длина коллектора, м", "Мощность коллектора, м",
    "Длина коллектора с поро>5%, м", "Длина глинистых участков, м", "Доля глинистости",

    # Все Средняя и Линейная по MD
    'Средняя ',"Средняя .1", "Линейная по MD", "Средняя .2", "Линейная по MD.1",
    "Средняя .3", "Линейная по MD.2", "Средняя .4", "Линейная по MD.3",
    "Средняя .5", "Линейная по MD.4", "Средняя .6", "Линейная по MD.5"
], errors="ignore")
```

```
df.reset_index(drop=True)
```

	Скважины	Qж, м3/сут	Качество коллектора	Акустический импеданс (PSTM)	RMS амплитуды	Расстояние от разлома R4 (интерпретатор)	Расстояние от разлома (вероятность разломов)	Расстояние от разлома (интерпретатор + вероятности)	Расстояние от вреза	Рас выкл
0	202_1_h	84.69	0.555	18329.34	1157.065	181.482	198.743	103.458	4509.707	
1	202_1_zbs	75.00	0.571	19037.81	333.230	546.018	177.107	177.107	4802.865	
2	237_1_h	229.86	0.639	17914.69	469.790	1298.976	83.969	83.969	3289.179	
3	272_1_h	316.10	0.646	19384.03	1423.487	756.552	295.196	295.196	3124.844	
4	782_1_h	13.00	0.625	17551.41	523.814	195.287	273.674	195.287	3765.022	
...	...	...	...	...	...	...	...	...	...	
270	657_53_h2	95.50	0.617	18469.91	1110.697	891.204	78.603	78.603	156.518	
271	362_54_h	121.00	0.557	18127.75	1122.852	193.434	161.894	119.790	217.242	
272	413_54_h	94.30	0.630	17116.49	543.458	213.632	54.275	54.008	171.722	
273	378_57_h	29.20	0.547	19868.97	1128.926	1255.104	170.126	170.126	255.657	

```
df.describe()
```

	Qж, м3/сут	Качество коллектора	Акустический импеданс (PSTM)	RMS амплитуды	Расстояние от разлома R4 (интерпретатор)	Расстояние от разлома (вероятность разломов)	Расстояние от разлома (интерпретатор + вероятности)	Расстояние от вреза	Рассто выклинив т
count	275.000000	275.000000	275.000000	275.000000	275.000000	275.000000	275.000000	275.000000	275.00
mean	118.440818	0.560625	18025.461964	1250.799927	784.099164	581.056745	379.676578	931.491015	920.63
std	104.659026	0.068420	1870.397750	1043.291816	551.062290	561.465995	371.696412	1220.572160	1084.73
min	1.000000	0.347000	11488.030000	81.338000	34.174000	13.577000	13.577000	0.000000	34.09
25%	34.245000	0.519000	17009.280000	543.910500	326.308000	174.458500	122.224000	138.991000	172.06
50%	86.400000	0.570000	18329.340000	927.013000	679.602000	376.020000	259.476000	360.277000	391.84

```
df = df.dropna()
```

```
df.isnull().sum()
```

	0
Скважины	0
Qж, м3/сут	0
Качество коллектора	0
Акустический импеданс (PSTM)	0
RMS амплитуды	0
Расстояние от разлома R4 (интерпретатор)	0
Расстояние от разлома (вероятность разломов)	0
Расстояние от разлома (интерпретатор + вероятности)	0
Расстояние от вреза	0
Расстояние от выклинивания толши	0
Глубина проводки (от эрозионной поверхности)	0
Толщина Б-Ro	0
Толщина R0-R4	0

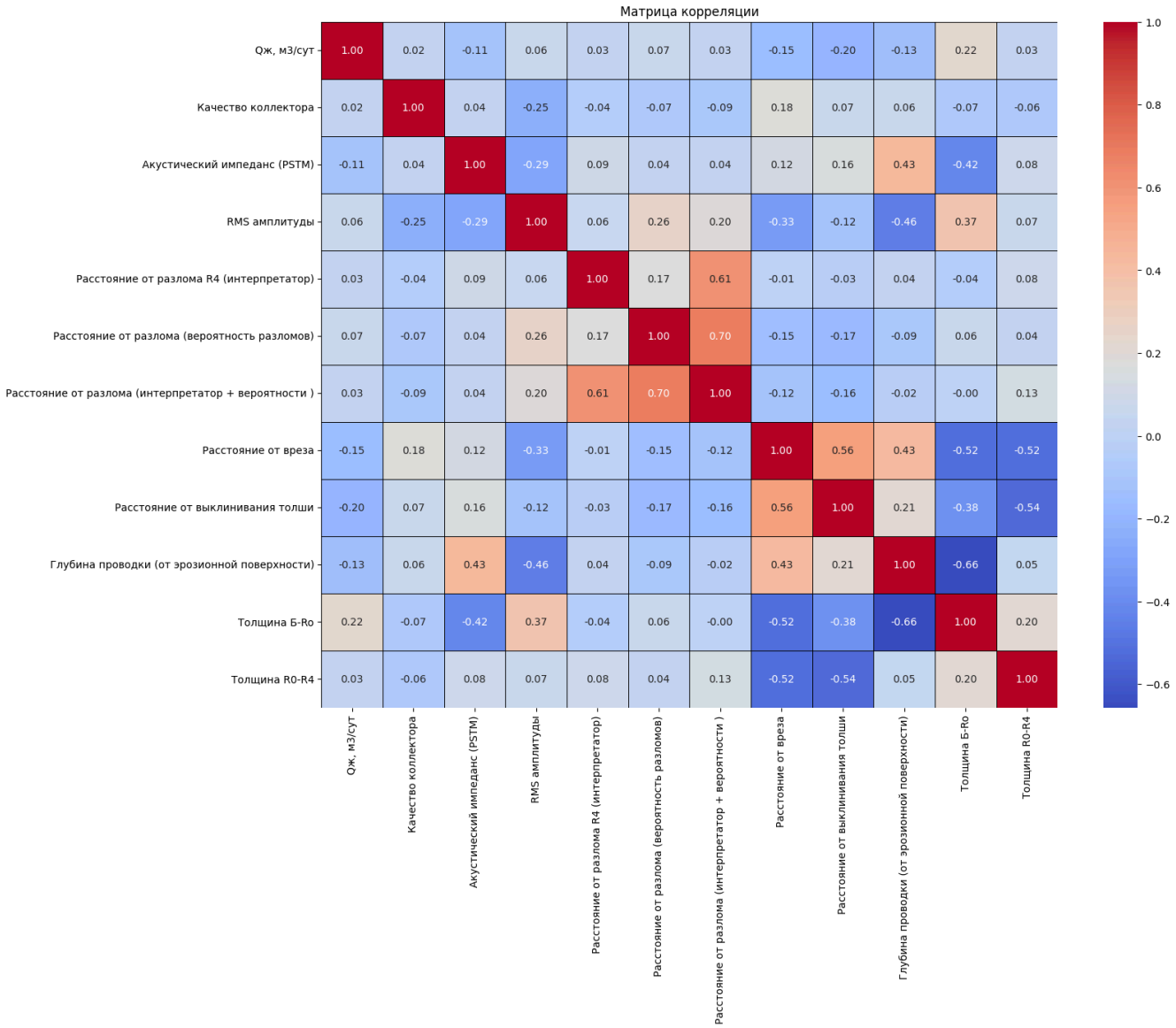
dtype: int64

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 275 entries, 1 to 275
Data columns (total 13 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   Скважины                                      275 non-null    object
 1   Qж, м3/сут                                    275 non-null    float64
```

```
 2   Качество коллектора                                275 non-null    float64
 3   Акустический импеданс (PSTM)                       275 non-null    float64
 4   RMS амплитуды                                      275 non-null    float64
 5   Расстояние от разлома R4 (интерпретатор)           275 non-null    float64
 6   Расстояние от разлома (вероятность разломов)       275 non-null    float64
 7   Расстояние от разлома (интерпретатор + вероятности ) 275 non-null  float64
 8   Расстояние от среза                                275 non-null    float64
 9   Расстояние от выклинивания толши                   275 non-null    float64
 10  Глубина проводки (от эрозионной поверхности)       275 non-null    float64
 11  Толщина Б-Ro                                       275 non-null    float64
 12  Толщина R0-R4                                      275 non-null    float64
dtypes: float64(12), object(1)
memory usage: 30.1+ KB
```

```
plt.figure(figsize=(16, 12))
sns.heatmap(
    df.corr(numeric_only=True),
    annot=True,
    cmap='coolwarm',
    fmt='.2f',
    linecolor='black',
    linewidths=0.5
)
plt.title('Матрица корреляции')
plt.show()
```



Т.к. по данным мы видим, что их мало сразу отметаем пункт с эксперементами нелинейных моделей, а работаем с тремя линейными моделями и выясним какая отработаем лучше всех!

Ручной выбор признаков (Качество коллектора → Толщина R0-R4)

[ ] ↳ Скрыто 9 ячеек.

Автоматический отбор признаков через SelectKBest

[ ] ↳ Скрыто 4 ячейки.

Подбор параметров через boruta

```
X_boruta = df[[
    'Качество коллектора',
    'Акустический импеданс (PSTM)',
    'RMS амплитуды',
    'Расстояние от разлома R4 (интерпретатор)',
    'Расстояние от разлома (вероятность разломов)',
    'Расстояние от разлома (интерпретатор + вероятности )',
    'Расстояние от вреза',
    'Расстояние от выклинивания толши',
    'Глубина проводки (от эрозионной поверхности)',
    'Толщина Б-Ro',
    'Толщина R0-R4'
]]
y_boruta = df['Qж, м3/сут']

# Делим на обучающую и тестовую выборку
X_train_boruta, X_test_boruta, y_train_boruta, y_test_boruta = train_test_split(X_boruta, y_boruta, test_size=0.2, random_st

# Стандартизация (для линейных моделей)
scaler_boruta = StandardScaler()
X_train_scaled = scaler_boruta.fit_transform(X_train_boruta)
X_test_scaled = scaler_boruta.transform(X_test_boruta)
```

```
rf_boruta = RandomForestRegressor(n_estimators=100, random_state=42)
boruta_selector = BorutaPy(estimator=rf_boruta,
                           n_estimators='auto',
                           max_iter=200,
                           perc=85,
                           random_state=42)

boruta_selector.fit(X_train_scaled, y_train_boruta.values)
selected_features = X_boruta.columns[boruta_selector.support_].tolist()

# 5. Проверка
if len(selected_features) == 0:
    raise ValueError("Boruta не выбрала ни одного признака. Попробуй изменить параметры отбора или выбрать признаки вручную.

print("Выбранные признаки Boruta:")
for feat in selected_features:
    print("-", feat)
```

```
Выбранные признаки Boruta:
- Расстояние от выклинивания толши
- Толщина Б-Ro
```

```
X_train_lin = X_train_scaled[:, boruta_selector.support_]
X_test_lin = X_test_scaled[:, boruta_selector.support_]
```

```
results_linear_boruta = []

# Ridge
ridge_boruta = GridSearchCV(Ridge(), {'alpha': [0.1, 1.0, 10.0, 100.0]}, cv=3)
ridge_boruta.fit(X_train_lin, y_train_boruta)
results_linear_boruta.append({
    'Model': 'Ridge',
    'Best Param': ridge_boruta.best_params_,
    'CV R2': cross_val_score(ridge_boruta.best_estimator_, X_train_lin, y_train_boruta, cv=3).mean(),
    'Test R2': r2_score(y_test_boruta, ridge_boruta.predict(X_test_lin)),
    'Train R2': r2_score(y_train_boruta, ridge_boruta.predict(X_train_lin)),
    'Test MSE': mean_squared_error(y_test_boruta, ridge_boruta.predict(X_test_lin)),
    'Train MSE': mean_squared_error(y_train_boruta, ridge_boruta.predict(X_train_lin))
})
```

```
# Lasso
lasso_boruta = GridSearchCV(Lasso(max_iter=10000), {'alpha': [0.01, 0.1, 1.0, 10.0]}, cv=3)
lasso_boruta.fit(X_train_lin, y_train_boruta)
results_linear_boruta.append({
    'Model': 'Lasso',
    'Best Param': lasso_boruta.best_params_,
    'CV R2': cross_val_score(lasso_boruta.best_estimator_, X_train_lin, y_train_boruta, cv=3).mean(),
    'Test R2': r2_score(y_test_boruta, lasso_boruta.predict(X_test_lin)),
    'Train R2': r2_score(y_train_boruta, lasso_boruta.predict(X_train_lin)),
    'Test MSE': mean_squared_error(y_test_boruta, lasso_boruta.predict(X_test_lin)),
    'Train MSE': mean_squared_error(y_train_boruta, lasso_boruta.predict(X_train_lin))
})

# Linear Regression
linear_boruta = LinearRegression()
linear_boruta.fit(X_train_lin, y_train_boruta)
results_linear_boruta.append({
    'Model': 'LinearRegression',
    'Best Param': None,
    'CV R2': cross_val_score(linear_boruta, X_train_lin, y_train_boruta, cv=3).mean(),
    'Test R2': r2_score(y_test_boruta, linear_boruta.predict(X_test_lin)),
    'Train R2': r2_score(y_train_boruta, linear_boruta.predict(X_train_lin)),
    'Test MSE': mean_squared_error(y_test_boruta, linear_boruta.predict(X_test_lin)),
    'Train MSE': mean_squared_error(y_train_boruta, linear_boruta.predict(X_train_lin))
})
```

```
selected_features = X_boruta.columns[boruta_selector.support_].tolist()

print("Выбранные признаки Boruta:")
for feat in selected_features:
    print("-", feat)
```

```
Выбранные признаки Boruta:
- Расстояние от выклинивания толщи
- Толщина Б-Ro
```

```
results_df_boruta = pd.DataFrame(results_linear_boruta)

# Сохраняем в CSV
results_df_boruta.to_csv('признаки борута линейные модели.csv', index=False)

# Показываем в ячейке
display(results_df_boruta)
```

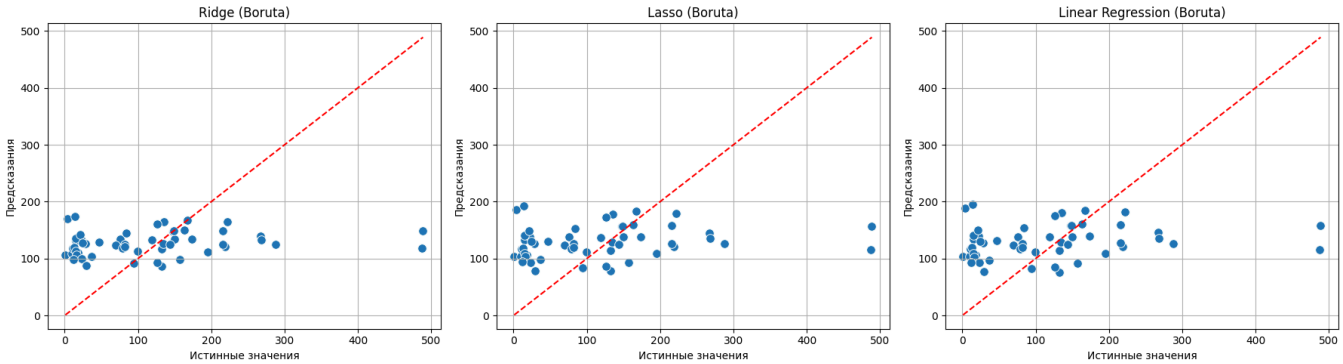
	Model	Best Param	CV R2	Test R2	Train R2	Test MSE	Train MSE
0	Ridge	{'alpha': 100.0}	0.052106	0.025271	0.070748	11126.115412	10012.856008
1	Lasso	{'alpha': 1.0}	0.049971	0.017107	0.075465	11219.300762	9962.031438
2	LinearRegression	None	0.049861	0.015046	0.075601	11242.827676	9960.568113

Далее: Создать код с переменной `results_df_boruta` | Посмотреть рекомендованные графики | New interactive sheet

```
# Предсказания по Boruta
y_pred_ridge_boruta = ridge_boruta.predict(X_test_lin)
y_pred_lasso_boruta = lasso_boruta.predict(X_test_lin)
y_pred_linear_boruta = linear_boruta.predict(X_test_lin)
```

```
fig, axs = plt.subplots(1, 3, figsize=(18, 5))

# Ridge
sns.scatterplot(ax=axs[0], x=y_test_boruta, y=y_pred_ridge_boruta, s=70)
min_r, max_r = min(y_test_boruta.min(), y_pred_ridge_boruta.min()), max(y_test_boruta.max(), y_pred_ridge_boruta.max())
axs[0].plot([min_r, max_r], [min_r, max_r], color='red', linestyle='--')
axs[0].set_title("Ridge (Boruta)")
axs[0].set_xlabel("Истинные значения")
axs[0].set_ylabel("Предсказания")
axs[0].grid(True)

# Lasso
sns.scatterplot(ax=axs[1], x=y_test_boruta, y=y_pred_lasso_boruta, s=70)
min_l, max_l = min(y_test_boruta.min(), y_pred_lasso_boruta.min()), max(y_test_boruta.max(), y_pred_lasso_boruta.max())
axs[1].plot([min_l, max_l], [min_l, max_l], color='red', linestyle='--')
axs[1].set_title("Lasso (Boruta)")
axs[1].set_xlabel("Истинные значения")
axs[1].set_ylabel("Предсказания")
axs[1].grid(True)

# Linear Regression
```

```
sns.scatterplot(ax=axs[2], x=y_test_boruta, y=y_pred_linear_boruta, s=70)
min_lin, max_lin = min(y_test_boruta.min(), y_pred_linear_boruta.min()), max(y_test_boruta.max(), y_pred_linear_boruta.max()
axs[2].plot([min_lin, max_lin], [min_lin, max_lin], color='red', linestyle='--')
axs[2].set_title("Linear Regression (Boruta)")
axs[2].set_xlabel("Истинные значения")
axs[2].set_ylabel("Предсказания")
axs[2].grid(True)

plt.tight_layout()
plt.show()
```



Отбор признаков и работа с лучшей моделью (Ridge)

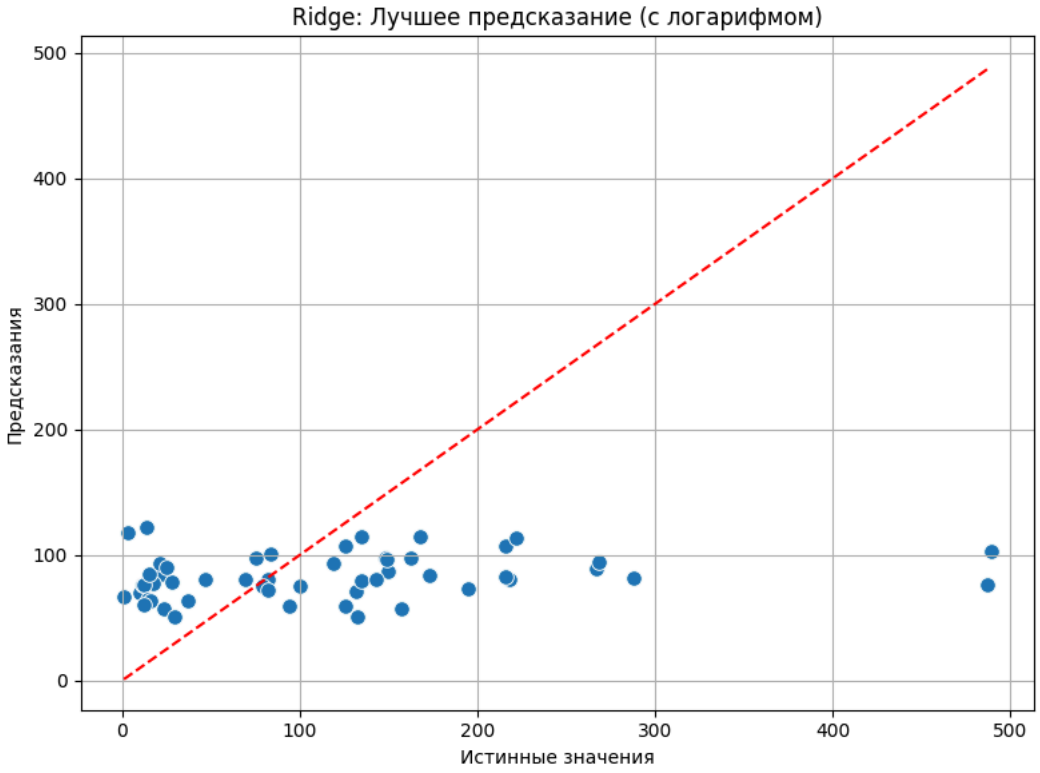
Отлично, борута помог нам определить математические лучшие коррелирующие признаки: Толщина Б-Ro, Расстояние от выталкивания толши; поэтому пострим цикл с наилучшим сочетанием признаков. Также мы видим, что ридж отрабатывает стабильнее всего, поэтомому работаем с ним

Комбинированный подбор признаков(гиперпараметр: alpha 100, логарифмирование)

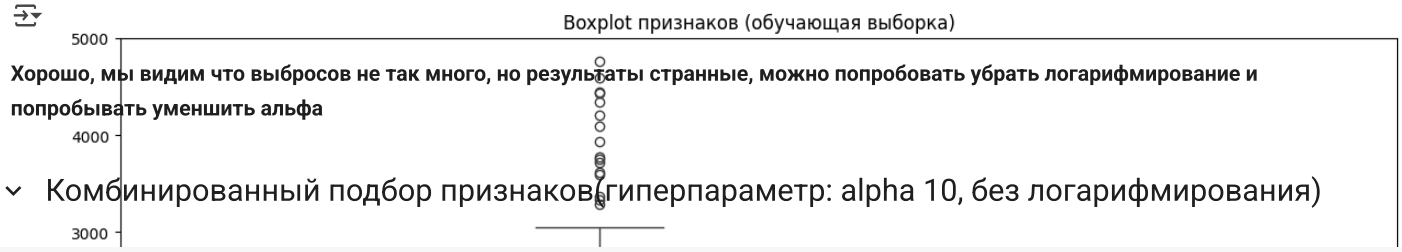
```
# Все признаки из диапазона
feature_pool = df.loc[:, 'Качество коллектора':'Толщина R0-R4'].columns.tolist()

# Два признака от Boruta
boruta_fixed = ['Толщина Б-Ro', 'Толщина R0-R4']
remaining_features = [f for f in feature_pool if f not in boruta_fixed]

# Комбинации: 2 и 3 доп. признака → итог 4 или 5
combo_additional_2 = list(combinations(remaining_features, 2))
combo_additional_3 = list(combinations(remaining_features, 3))
all_combos = combo_additional_2 + combo_additional_3

# Храним только лучший результат
best_result = None
best_score = -np.inf

for additional in all_combos:
    combo = boruta_fixed + list(additional)
    X = df[combo]
    y = df['Qж, м3/сут']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Логарифмируем целевую переменную
    y_train_log = np.log1p(y_train)
    y_test_log = np.log1p(y_test)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

```
    ridge = GridSearchCV(Ridge(), {'alpha': [0.01, 0.1, 1.0, 10.0, 100.0]}, cv=3)
    ridge.fit(X_train_scaled, y_train_log)

    y_pred_log_test = ridge.predict(X_test_scaled)
    y_pred_log_train = ridge.predict(X_train_scaled)

    # Обратное логарифмирование
    y_pred_test = np.expm1(y_pred_log_test)
    y_pred_train = np.expm1(y_pred_log_train)

    # Оценка на оригинальных значениях
    test_r2 = r2_score(y_test, y_pred_test)

    if test_r2 > best_score:
        best_score = test_r2
        best_result = {
            'Features': combo,
            'Best alpha': ridge.best_params_['alpha'],
            'CV R2': cross_val_score(ridge.best_estimator_, X_train_scaled, y_train_log, cv=3).mean(),
            'Test R2': test_r2,
            'Train R2': r2_score(y_train, y_pred_train),
            'Test MSE': mean_squared_error(y_test, y_pred_test),
            'Train MSE': mean_squared_error(y_train, y_pred_train)
        }

        # Сохраняем данные для графика
        best_y_test = y_test
        best_y_pred_test = y_pred_test

# Выводим лучший результат
best_df = pd.DataFrame([best_result])
display(best_df)

# Сохраняем только лучший результат в CSV
best_df.to_csv('best_ridge_combination_log.csv', index=False)

# График предсказаний
plt.figure(figsize=(8, 6))
sns.scatterplot(x=best_y_test, y=best_y_pred_test, s=70)
plt.plot([best_y_test.min(), best_y_test.max()],
         [best_y_test.min(), best_y_test.max()],
         color='red', linestyle='--')
plt.xlabel("Истинные значения")
plt.ylabel("Предсказания")
plt.title("Ridge: Лучшее предсказание (с логарифмом)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

	Features	Best alpha	CV R2	Test R2	Train R2	Test MSE	Train MSE
0	[Толщина Б-Ro, Толщина R0-R4, Расстояние от ра...	100.0	0.024321	-0.022523	-0.106495	11671.659626	11922.689518




```
plt.figure(figsize=(15, 6))
sns.boxplot(data=X_train)  # или df[features] если еще не делил
plt.title("Boxplot признаков (обучающая выборка)")
plt.xticks(rotation=45)
plt.show()
```



Хорошо, мы видим что выбросов не так много, но результаты странные, можно попробовать убрать логарифмирование и попробывать уменьшить альфа

Комбинированный подбор признаков(гиперпараметр: alpha 10, без логарифмирования)

```
# Все признаки из диапазона
feature_pool_nolog = df.loc[:, 'Качество коллектора':'Толщина R0-R4'].columns.tolist()

# Два признака от Boruta
boruta_fixed_nolog = ['Толщина Б-Ro', 'Толщина R0-R4']
remaining_features_nolog = [f for f in feature_pool_nolog if f not in boruta_fixed_nolog]

# Комбинации: 2 и 3 доп. признака → итог 4 или 5
combo_add_2_nolog = list(combinations(remaining_features_nolog, 2))
combo_add_3_nolog = list(combinations(remaining_features_nolog, 3))
all_combos_nolog = combo_add_2_nolog + combo_add_3_nolog

# Храним только лучший результат
best_result_nolog = None
best_score_nolog = -np.inf

for additional_nolog in all_combos_nolog:
    combo_nolog = boruta_fixed_nolog + list(additional_nolog)
    X_nolog = df[combo_nolog]
    y_nolog = df['Qж, м3/сут']

    X_train_nolog, X_test_nolog, y_train_nolog, y_test_nolog = train_test_split(X_nolog, y_nolog, test_size=0.2, random_stat

    scaler_nolog = StandardScaler()
    X_train_scaled_nolog = scaler_nolog.fit_transform(X_train_nolog)
    X_test_scaled_nolog = scaler_nolog.transform(X_test_nolog)

    ridge_nolog = GridSearchCV(Ridge(), {'alpha': [0.01, 0.1, 1.0, 10.0, 100.0]}, cv=3)
    ridge_nolog.fit(X_train_scaled_nolog, y_train_nolog)

    y_pred_test_nolog = ridge_nolog.predict(X_test_scaled_nolog)
    y_pred_train_nolog = ridge_nolog.predict(X_train_scaled_nolog)

    test_r2_nolog = r2_score(y_test_nolog, y_pred_test_nolog)

    if test_r2_nolog > best_score_nolog:
        best_score_nolog = test_r2_nolog
        best_result_nolog = {
            'Features': combo_nolog,
            'Best alpha': ridge_nolog.best_params_['alpha'],
            'CV R2': cross_val_score(ridge_nolog.best_estimator_, X_train_scaled_nolog, y_train_nolog, cv=3).mean()
```