

```
In [409... import time
import warnings
from tqdm.notebook import tqdm
from collections import defaultdict
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from catboost import CatBoostRanker, Pool, MetricVisualizer
from copy import deepcopy

import pandas as pd
import scipy.stats as sps
```

# Тестовое задание

## Обзор

```
In [387... data = pd.read_csv('intern_task.csv')
```

```
In [360... data.head()
```

Out[360]:

	rank	query_id	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	...	fe
0	0	10	1.0	0.0	1.0	3.0	3.0	0.333333	0.0	0.333333	...	
1	1	10	3.0	0.0	3.0	0.0	3.0	1.000000	0.0	1.000000	...	
2	0	10	3.0	0.0	2.0	0.0	3.0	1.000000	0.0	0.666667	...	
3	1	10	3.0	0.0	3.0	0.0	3.0	1.000000	0.0	1.000000	...	
4	2	10	3.0	0.0	3.0	1.0	3.0	1.000000	0.0	1.000000	...	

5 rows × 146 columns

```
In [361... data.describe()
```

Out[361]:

	rank	query_id	feature_0	feature_1	feature_2	feature_3	featu
count	235258.000000	235258.000000	235258.000000	235258.000000	235258.000000	235258.000000	235258.000
mean	0.677869	14828.413401	1.911960	0.206233	1.189847	0.550272	1.960
std	0.830033	8193.945170	1.237374	0.579089	1.037233	0.790947	1.200
min	0.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	8215.000000	1.000000	0.000000	0.000000	0.000000	1.000
50%	0.000000	14935.000000	2.000000	0.000000	1.000000	0.000000	2.000
75%	1.000000	21580.000000	3.000000	0.000000	2.000000	1.000000	3.000
max	4.000000	29995.000000	31.000000	18.000000	27.000000	9.000000	31.000

8 rows × 146 columns

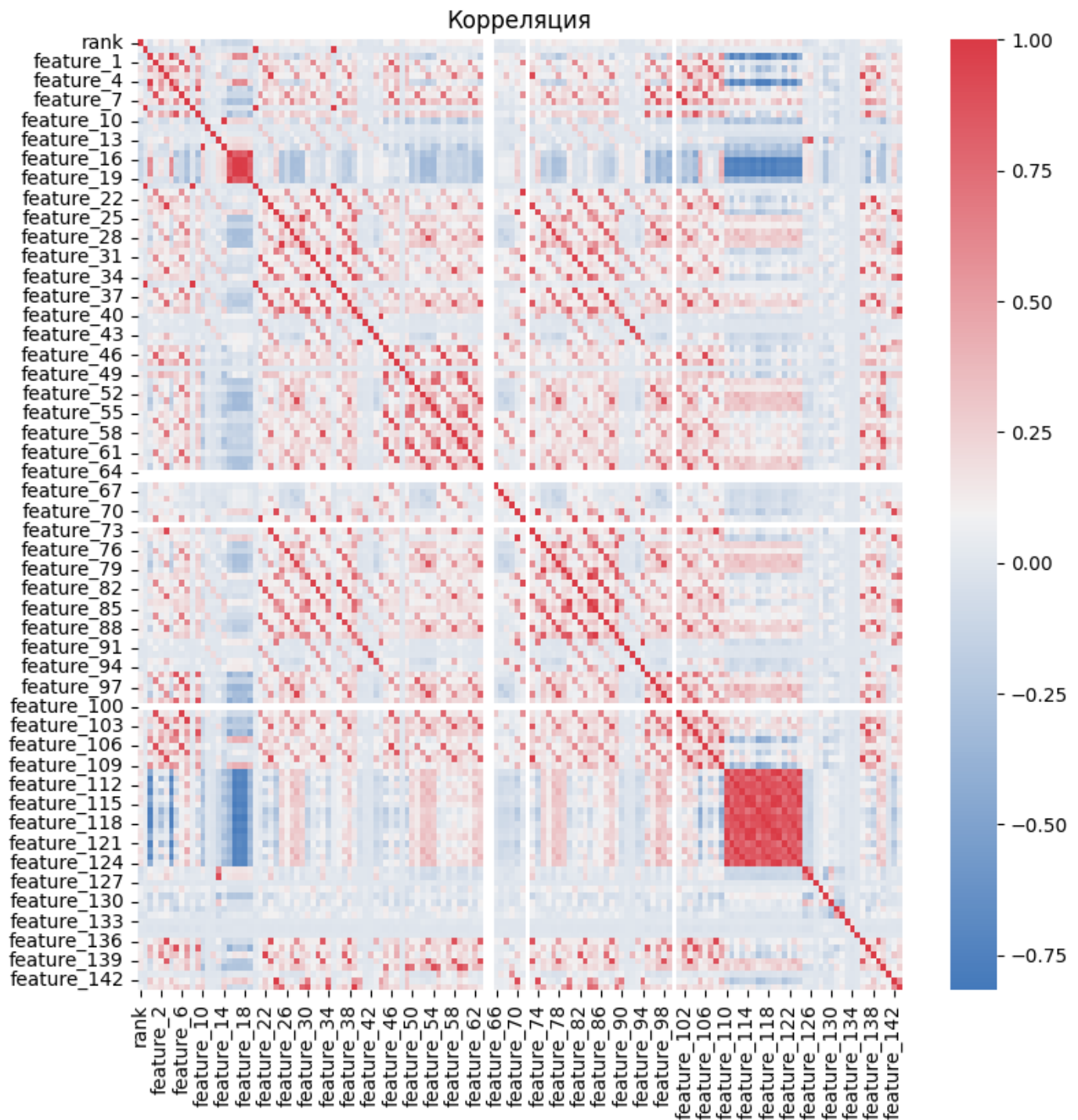
```
In [362... print(f'Кол-во пропусков: {data.isna().sum().sum()}')
```

Кол-во пропусков: 0

## Анализ фич

В нашем случае, размер выборки не мал. Если размер выборки велик, а распределение признаков не нормальное, то коэффициент корреляции Пирсона выборки остается примерно несмещенным, но может быть неэффективным.

```
In [58]: plt.figure(figsize=(9, 9))
смап = sns.diverging_palette(250, 10, as_cmap=True)
sns.heatmap(data.corr(), cmap=смап)
plt.title("Корреляция")
plt.show()
```



Заметим, что есть фичи:

1. которые вообще не коррелируют;
2. которые сильно коррелируют с некоторыми (может быть линейная зависимость между ними → мультиколлинеарность); Возможно, это одна и та же фишка (по смыслу), но в другом формате или представлении

Посмотрим отдельно на них

```
In [363... useless_feats = []

for i in range(144):
    feat = f'feature_{i}'
    minn = data[feat].min()
    maxx = data[feat].max()
    if np.isclose(minn, maxx) or data[feat].std() < 0.0001:
        useless_feats.append(feat)
        print(f'Неинформативный признак {feat}: мин. {minn}, макс. {maxx}')
```

useless\_feats

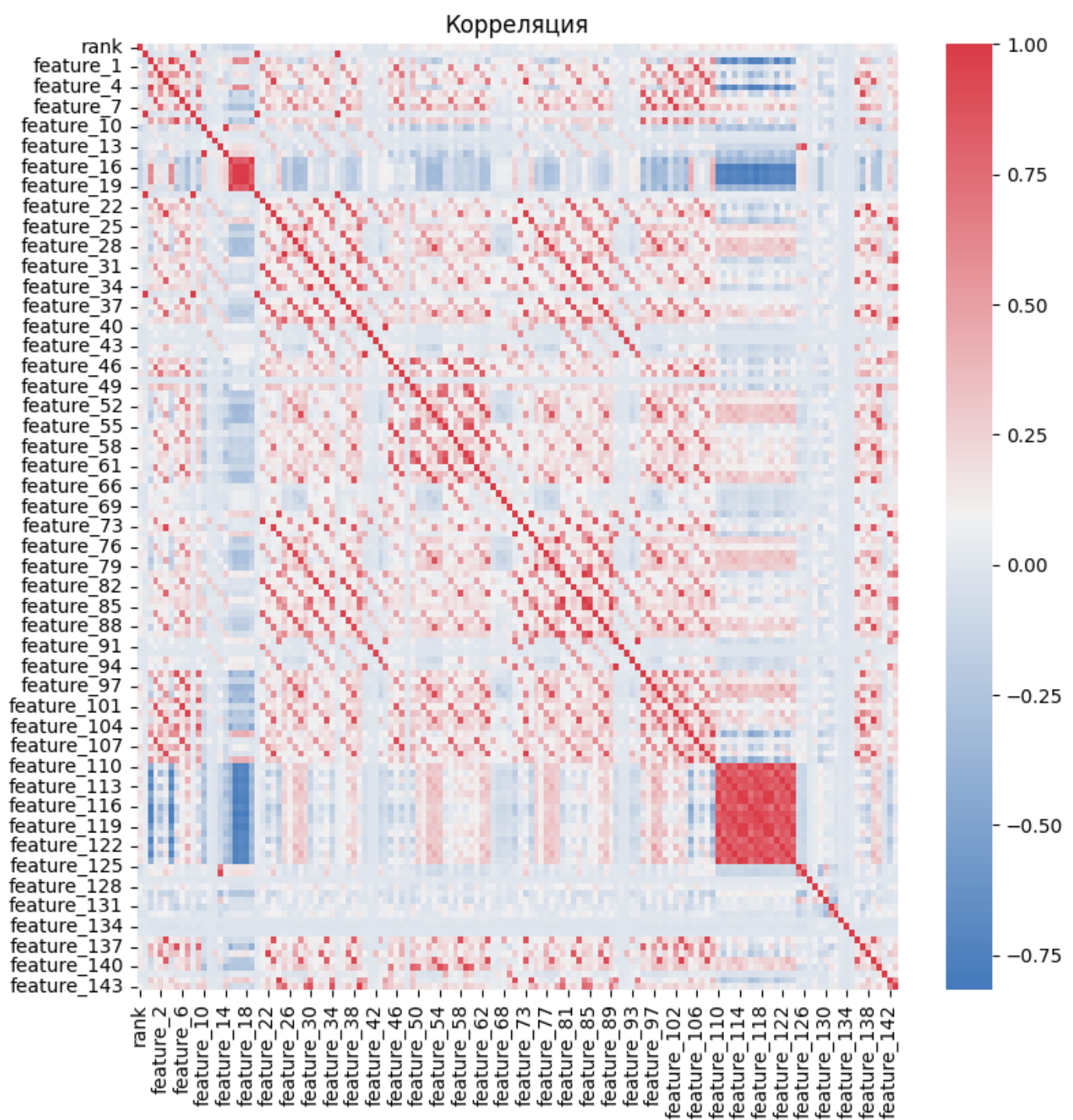
Неинформативный признак feature\_64: мин. 0, макс. 0  
Неинформативный признак feature\_65: мин. 0, макс. 0  
Неинформативный признак feature\_72: мин. 1, макс. 1  
Неинформативный признак feature\_100: мин. 0, макс. 0

```
Out[363]: ['feature_64', 'feature_65', 'feature_72', 'feature_100']
```

Удалим их

```
In [388... data = data.drop(columns=useless_feats)
```

```
In [87]: plt.figure(figsize=(9, 9))
snsmap = sns.diverging_palette(250, 10, as_cmap=True)
sns.heatmap(data.corr(), cmap=snsmap)
plt.title("Корреляция")
plt.show()
```



Попробуем задетектить категориальные признаки. Целочисленных признаков много. Пусть это будут те признаки, у которых всего макс. 3 значения

```
In [389... cat_feats = []

for feat in data.columns:
    unique_a = np.unique(data[feat])
    if (unique_a.astype(int) == unique_a).sum() == len(unique_a) and len(unique_a) < 4:
        print(f'Предположительно категориальная {feat} имеет {len(unique_a)} значений {n
        cat_feats.append(feat)
```

```
Предположительно категориальная feature_95 имеет 2 значений [0. 1.]
Предположительно категориальная feature_96 имеет 2 значений [0. 1.]
Предположительно категориальная feature_97 имеет 2 значений [0. 1.]
Предположительно категориальная feature_98 имеет 2 значений [0. 1.]
Предположительно категориальная feature_99 имеет 2 значений [0. 1.]
```

Если же нет, то их много

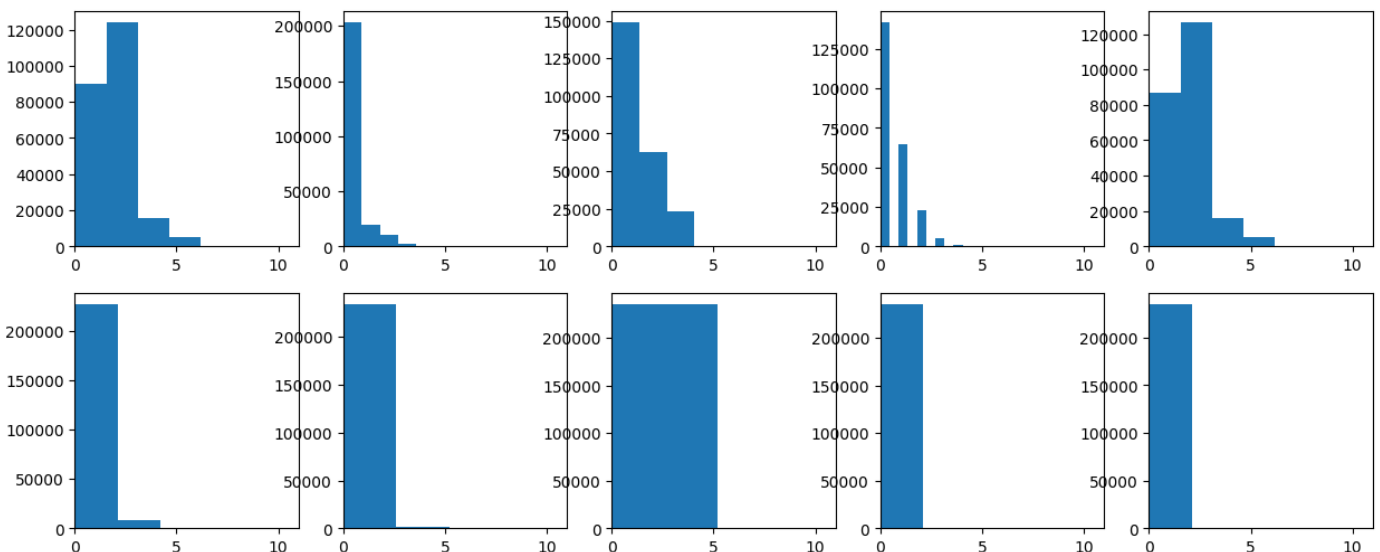
```
In [355... cat_feats = []

for feat in data.columns:
    unique_a = np.unique(data[feat])
    if (unique_a.astype(int).sum() == len(unique_a) and len(unique_a) < 30
        print(f'Предположительно категориальная {feat} имеет {len(unique_a)} значений {n
            cat_feats.append(feat)
```

```
Предположительно категориальная feature_0 имеет 16 значений [ 0.  1.  2.  3.  4.  5.  6.
 7.  8.  9. 10. 11. 24. 26. 27. 31.]
Предположительно категориальная feature_1 имеет 10 значений [ 0.  1.  2.  3.  4.  5.  6.
 7.  9. 18.]
Предположительно категориальная feature_2 имеет 15 значений [ 0.  1.  2.  3.  4.  5.  6.
 7.  8.  9. 19. 20. 22. 24. 27.]
Предположительно категориальная feature_3 имеет 10 значений [0.  1.  2.  3.  4.  5.  6.  7.  8.
 9.]
Предположительно категориальная feature_4 имеет 16 значений [ 0.  1.  2.  3.  4.  5.  6.
 7.  8.  9. 10. 11. 24. 26. 27. 31.]
Предположительно категориальная feature_23 имеет 13 значений [ 0.  1.  2.  3.  4.  5.
 6.  7.  8.  9. 10. 12. 42.]
Предположительно категориальная feature_26 имеет 26 значений [ 0.  1.  2.  3.  4.  5.
 6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 22. 24. 25. 28. 49. 52.]
Предположительно категориальная feature_27 имеет 24 значений [ 0.  1.  2.  3.  4.
 5.  6.  7.  8.  9. 10. 11. 12. 13.
 14. 15. 21. 22. 27. 30. 32. 38. 53. 104.]
Предположительно категориальная feature_28 имеет 10 значений [ 0.  1.  2.  3.  4.  5.
 6. 10. 12. 42.]
Предположительно категориальная feature_33 имеет 11 значений [ 0.  1.  2.  3.  4.  5.
 6.  8. 10. 12. 42.]
Предположительно категориальная feature_95 имеет 2 значений [0.  1.]
Предположительно категориальная feature_96 имеет 2 значений [0.  1.]
Предположительно категориальная feature_97 имеет 2 значений [0.  1.]
Предположительно категориальная feature_98 имеет 2 значений [0.  1.]
Предположительно категориальная feature_99 имеет 2 значений [0.  1.]
Предположительно категориальная feature_125 имеет 19 значений [ 1.  2.  3.  4.  5.  6.
 7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.
 31.]
```

```
In [318... plt.figure(figsize=(15, 6))

for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.hist(data[cat_feats[i]], bins=20)
    plt.xlim(0, 11)
```



Причем у некоторых есть критические значения которых мало. Так как у нас нет описания признаков,

я тоже ничего не буду с ними делать + не уверен что они вообще категориальные

Пока возьмем за категориальные фичи у которых всего 2 значения

```
In [390... cat_data = data[cat_feats].astype(int)
data = data.drop(columns=cat_feats)
```

```
In [394... cat_data.describe()
```

```
Out[394]:
```

	feature_95	feature_96	feature_97	feature_98	feature_99
count	235258.000000	235258.000000	235258.000000	235258.000000	235258.000000
mean	0.699912	0.061779	0.383192	0.184946	0.733046
std	0.458297	0.240754	0.486166	0.388254	0.442369
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000	1.000000
75%	1.000000	0.000000	1.000000	0.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

Что касается выбросов (если они есть): Их можно задетектить с помощью Robust Random Cut Forest или других методов. Но так как у нас нет никакого описания признаков, я не буду этого делать.

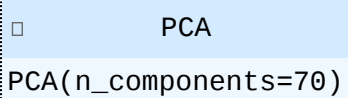
## Подход 1: PCA

```
In [395... Y_data = data['rank']
Q_data = data['query_id']
X_data = data.drop(columns=['rank', 'query_id'])
```

```
In [396... scaler = StandardScaler()
scaled_X_data = scaler.fit_transform(X_data)
```

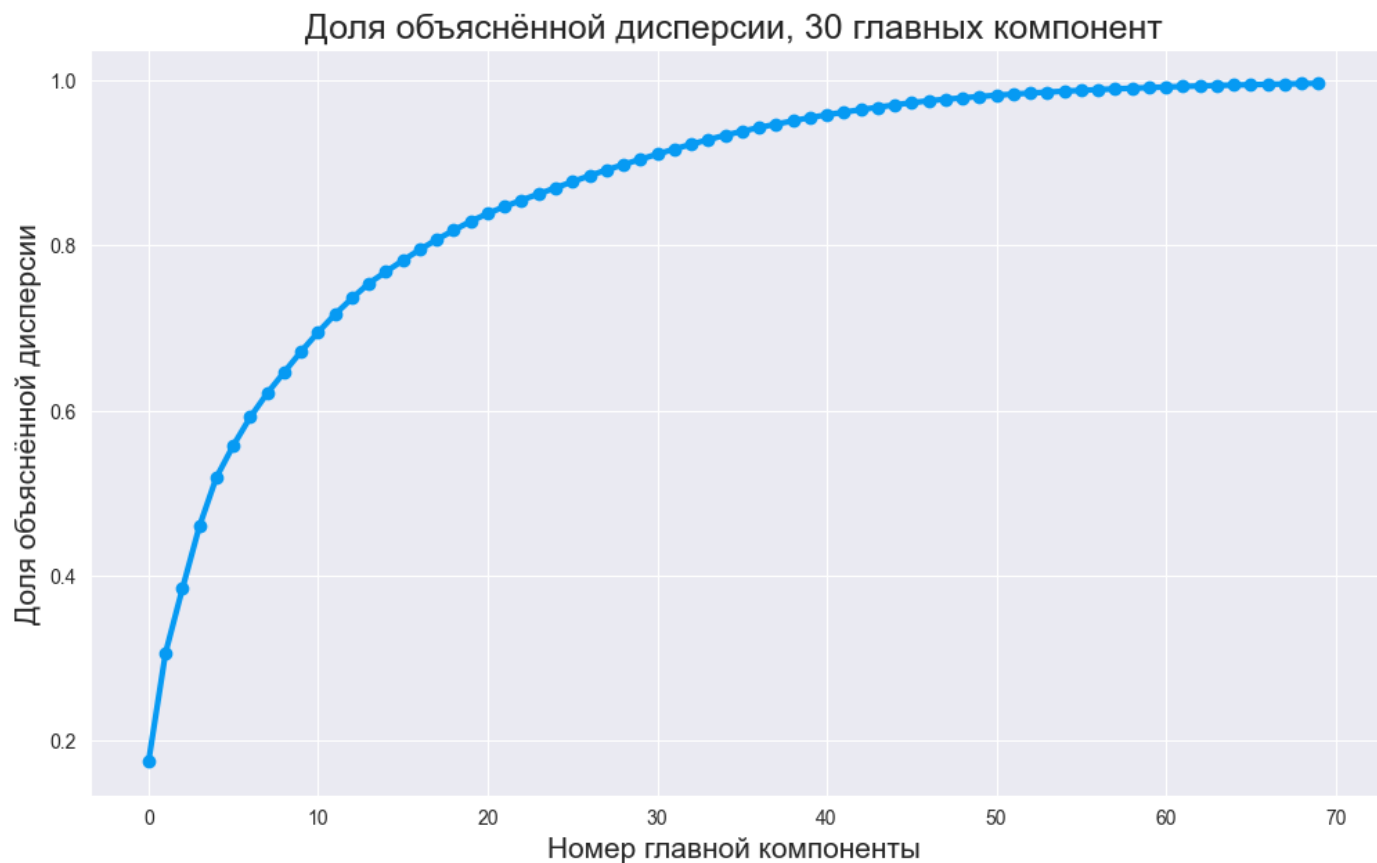
```
In [397... pca = PCA(n_components=70)
pca.fit(scaled_X_data)
```

```
Out[397]:
```



```
In [371... sns.set_style('darkgrid')

plt.figure(figsize=(12, 7))
plt.title("Доля объяснённой дисперсии, 30 главных компонент", fontsize=18)
plt.scatter(np.arange(70), pca.explained_variance_ratio_.cumsum(), color="xkcd:azure");
plt.plot(np.arange(70), pca.explained_variance_ratio_.cumsum(), color="xkcd:azure", lw=3)
plt.xlabel("Номер главной компоненты", fontsize=15)
plt.ylabel("Доля объяснённой дисперсии", fontsize=15)
plt.show()
```



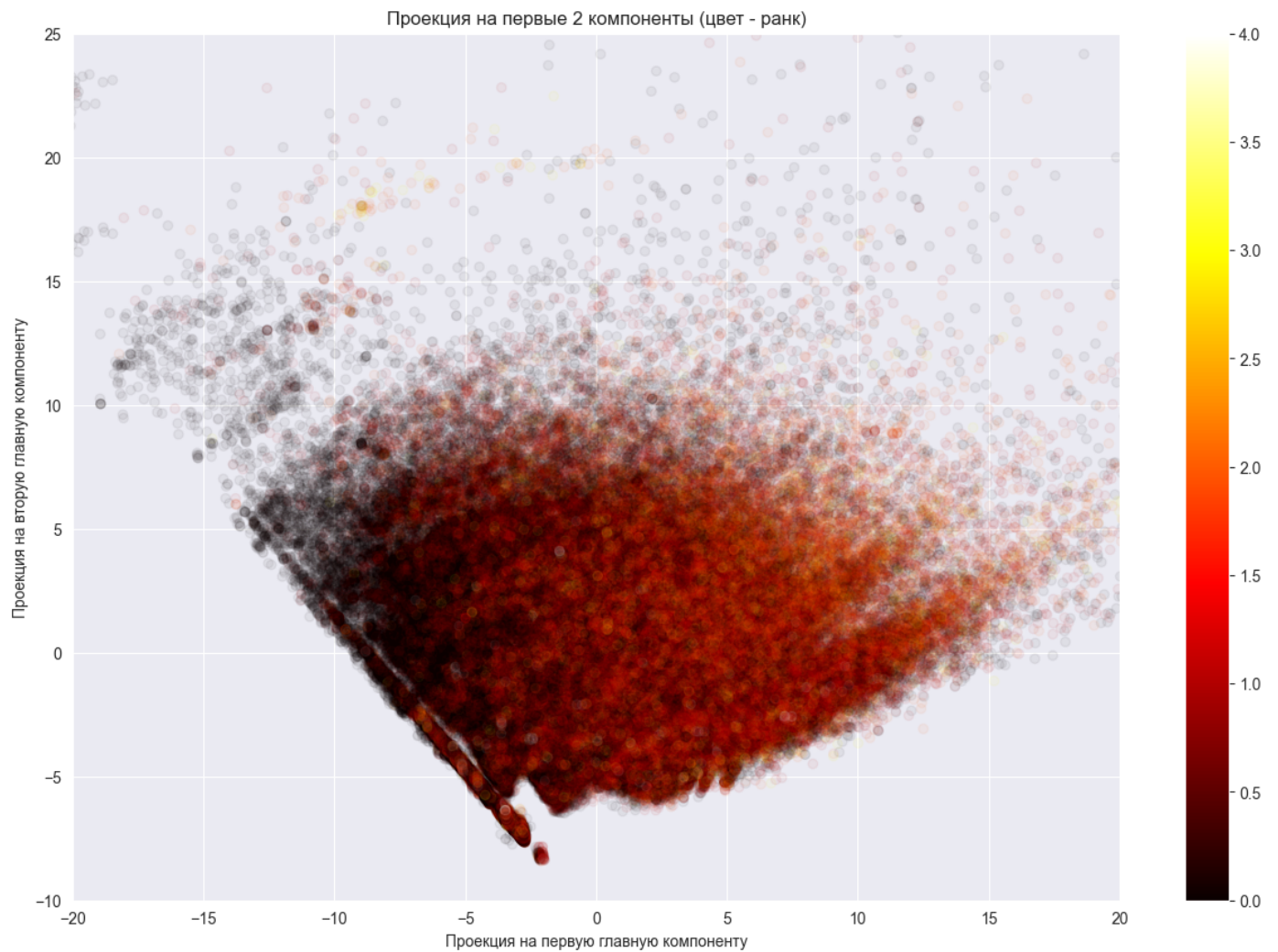
Выберем 30 компонент

```
In [398... n_components=30
```

```
In [399... pca = PCA(n_components=n_components)
scaled_tranformed_X = pca.fit_transform(scaled_X_data)
```

```
In [164... pca = PCA(n_components=3)
Y = pca.fit_transform(scaled_X_data)
plt.figure(figsize=(15, 10))
p = plt.scatter(Y[:, 0], Y[:, 1], c=Y_data, cmap='hot', alpha=1)
plt.xlabel('Проекция на первую главную компоненту')
plt.ylabel('Проекция на вторую главную компоненту')
plt.title('Проекция на первые 2 компоненты (цвет - ранк)')
plt.colorbar()
p.set_alpha(0.05)
plt.xlim(-20, 20)
plt.ylim(-10, 25)
plt.show()
```





Заметно смещение по рангу

## Распределение рангов

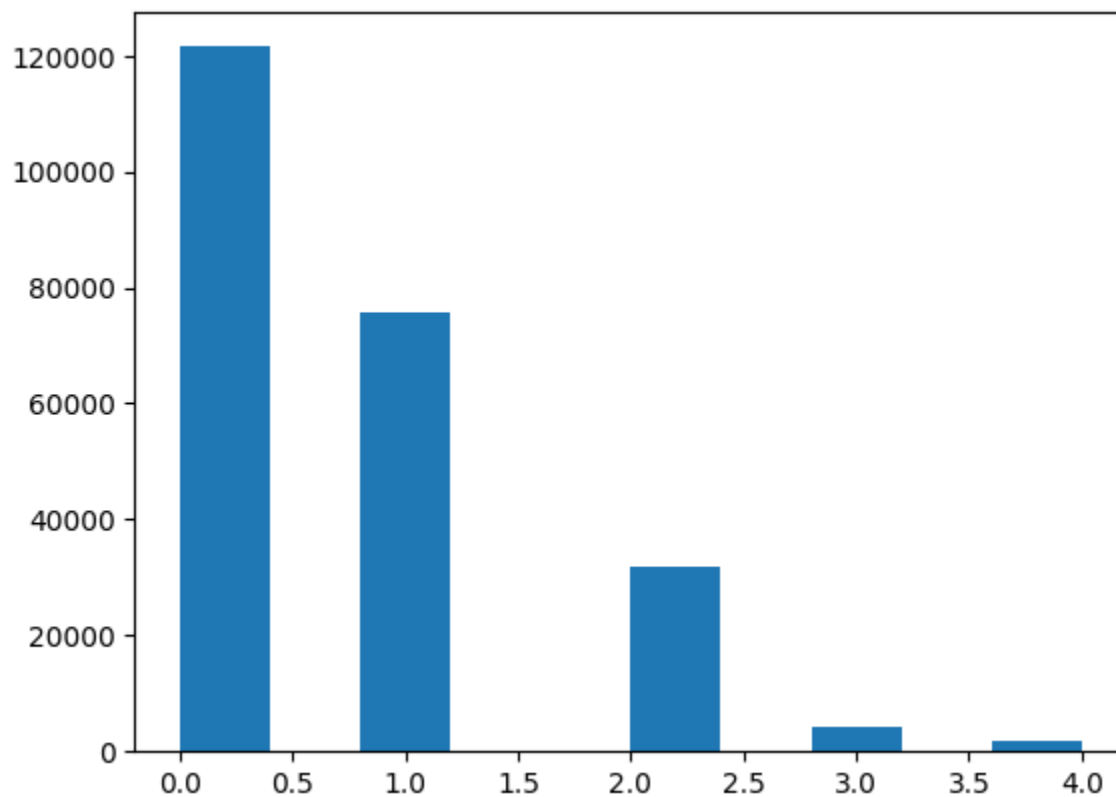
```
In [33]: data.groupby('rank')['query_id'].count()
```

```
Out[33]: rank
0      121521
1       75815
2       31910
3        4209
4         1803
Name: query_id, dtype: int64
```

```
In [32]: data['rank'].hist(grid=False)
```

```
Out[32]: <Axes: >
```





## Подготовка к обучению

```
In [400...] X_df = pd.DataFrame(scaled_tranformed_X)
X_df = pd.concat([X_df, cat_data], axis=1)
X_df.head()
```

```
Out[400]:
```

	0	1	2	3	4	5	6	7	8	9
0	4.052559	2.344586	-9.747515	-3.416371	1.336300	-3.534182	0.909853	0.955754	-1.093650	1.376543
1	-0.432786	2.264945	3.081714	-1.804683	-2.302030	0.409672	0.450202	-1.160240	-0.467964	0.512443
2	-2.645617	1.129820	1.719140	-0.952188	-1.674765	0.443441	-0.317642	-1.520690	-0.558013	0.809746
3	-0.452983	1.162849	2.626443	-2.370404	-4.113124	-1.693650	-0.483230	-1.024103	0.312562	0.223809
4	2.739472	3.112070	-1.074635	-3.516672	-2.758475	-0.750276	2.074385	-1.620698	-1.663656	1.713078

5 rows × 35 columns

```
In [375...] Q_data.unique().shape
```

```
Out[375]: (2000,)
```

Всего 2000 сессии. На оценку качества отведем 200 (10%).

```
In [401...] Q_test_indexes = np.random.choice(Q_data.unique(), size=200, replace=False)
Q_train_indexes = np.array([i for i in Q_data.unique() if i not in Q_test_indexes])
assert(Q_train_indexes.shape[0] == 1800 and Q_test_indexes.shape[0] == 200)
```

```
In [402...] X_train_df = X_df.loc[np.where(Q_data.isin(Q_train_indexes))]
X_test_df = X_df.loc[np.where(Q_data.isin(Q_test_indexes))]

y_train = Y_data.loc[np.where(Q_data.isin(Q_train_indexes))]
y_test = Y_data.loc[np.where(Q_data.isin(Q_test_indexes))]
```

```
Q_train = Q_data.loc[np.where(Q_data.isin(Q_train_indexes))]  
Q_test = Q_data.loc[np.where(Q_data.isin(Q_test_indexes))]
```

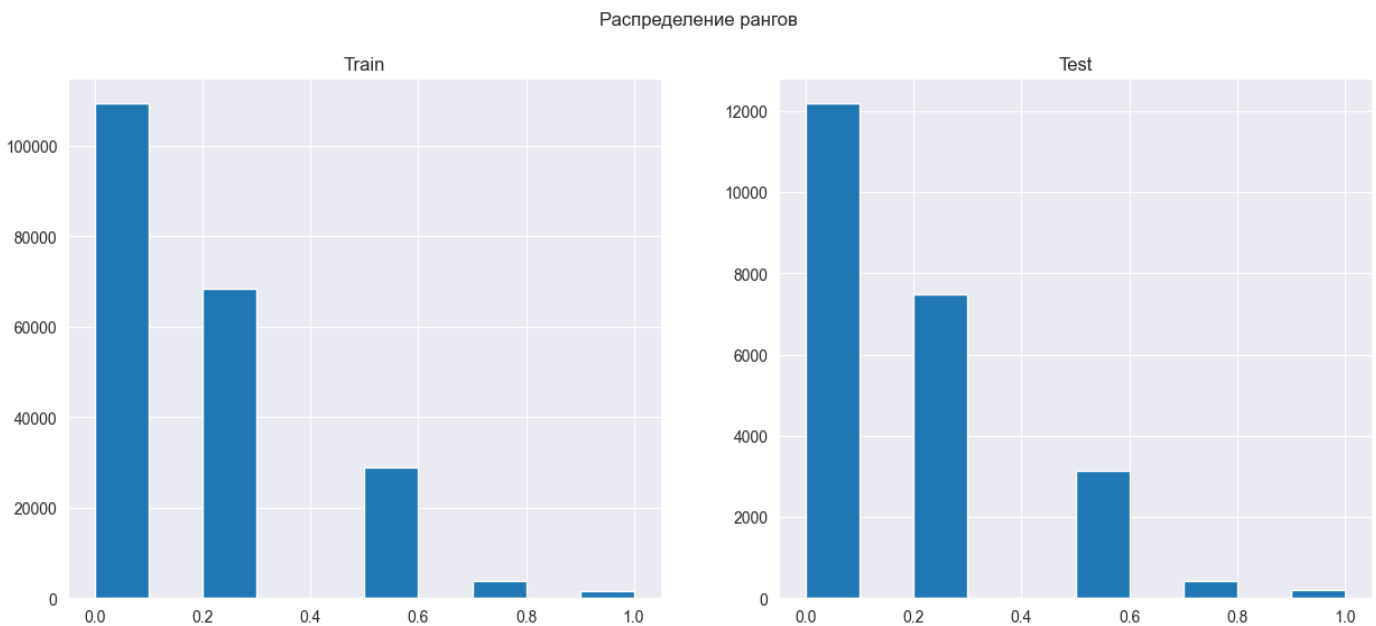
```
In [403... X_train_df.columns = X_train_df.columns.astype(str)  
X_test_df.columns = X_test_df.columns.astype(str)
```

```
In [382... X_train_df.shape
```

```
Out[382]: (209800, 35)
```

Убедимся, что распределение классов одинаково

```
In [490... plt.figure(figsize=(15, 6))  
plt.suptitle('Распределение рангов')  
plt.subplot(121)  
plt.title('Train')  
y_train.hist()  
plt.subplot(122)  
plt.title('Test')  
y_test.hist()  
plt.show()
```



Они *почти* совпадают, т.к. мы брали рандомное разбиение

## Обучение

### CatBoostRanker

Для подсчета метрик отобразим ранги на отрезок [0, 1]

```
In [404... max_rank = np.max(Y_data)  
y_train /= max_rank  
y_test /= max_rank
```

```
In [405... train = Pool(  
    data=X_train_df,  
    label=y_train,  
    group_id=Q_train,  
    cat_features=np.array(cat_data.columns)  
)
```

```
test = Pool(
    data=X_test_df,
    label=y_test,
    group_id=Q_test,
    cat_features=np.array(cat_data.columns)
)
```

```
In [510... params = {
    'iterations': 2000,
    'custom_metric': ['NDCG:top=5', 'NDCG:top=10', 'PFound', 'PrecisionAt:top=5', 'MAP:t
    'verbose': False,
    'random_seed': 42,
}
```

```
In [511... def train_ranker(loss_function, plot=False):
    parameters = deepcopy(params)
    parameters['loss_function'] = loss_function
    parameters['train_dir'] = loss_function

    model = CatBoostRanker(**parameters)
    model.fit(train, eval_set=test, plot=plot, use_best_model=True)

    return model.best_score_
```

```
In [527... def dict_to_dataframe(metrics_dict, model_name, df=None):
    learn_metrics = metrics_dict.get('learn', {})
    validation_metrics = metrics_dict.get('validation', {})

    df_learn = pd.DataFrame(learn_metrics, index=[model_name])
    df_valid = pd.DataFrame(validation_metrics, index=[model_name])

    df_learn.columns = [col for col in df_learn.columns]
    df_valid.columns = [col for col in df_valid.columns]

    d = {'Train' : df_learn, 'Test' : df_valid}

    df_concat = pd.concat(d.values(), axis=1, keys=d.keys())
    if df is not None:
        df_concat = pd.concat([df, df_concat], axis=0)
    return df_concat
```

```
In [533... losses = ['RMSE', 'QueryRMSE', 'PairLogit', 'YetiRank']
```

```
In [534... result_df = None
for loss_func in tqdm(losses):
    metrics = train_ranker(loss_func)
    if result_df is None:
        result_df = dict_to_dataframe(metrics, loss_func)
    else:
        result_df = dict_to_dataframe(metrics, loss_func, result_df)
```

```
0%|          | 0/4 [00:00<?, ?it/s]
```

```
C:\Users\ztimu\anaconda3\lib\site-packages\catboost\core.py:6219: RuntimeWarning: Regres
sion loss ('RMSE') ignores an important ranking parameter 'group_id'
  warnings.warn("Regression loss ('{}') ignores an important ranking parameter 'group_i
d'".format(loss_function), RuntimeWarning)
```

Метрики качества для разных loss функции

```
In [600... result_df.index.name = 'CatBoostRanker'
```

```
In [601... np.round(result_df['Train'][['MAP:top=5', 'PrecisionAt:top=5']], 2)
```

Out[601]: MAP:top=5 PrecisionAt:top=5

CatBoostRanker		
RMSE	0.23	0.19
QueryRMSE	0.17	0.15
PairLogit	0.11	0.11
YetiRank	0.19	0.16

In [602... np.round(result\_df['Test'][['NDCG:top=10;type=Base', 'PFound', 'NDCG:top=5;type=Base', 'MAP:top=5', 'PrecisionAt:top=5']])

Out[602]: NDCG:top=10;type=Base PFound NDCG:top=5;type=Base MAP:top=5 PrecisionAt:top=5

CatBoostRanker					
RMSE	0.48	0.68	0.48	0.11	0.10
QueryRMSE	0.50	0.68	0.48	0.12	0.10
PairLogit	0.49	0.66	0.47	0.10	0.10
YetiRank	0.50	0.69	0.50	0.13	0.11

Корректность: подадим на вход неинформативные данные.

In [566... probs = y\_train.value\_counts().values / y\_train.value\_counts().sum()

In [567... probs

Out[567]: array([0.51606493, 0.32253527, 0.13585451, 0.0179223 , 0.00762299])

In [582... X\_df\_dummy = np.random.random(size=(211859, 35))  
y\_dummy = np.random.choice([0, 0.25, 0.5, 0.75, 1], p=probs, replace=True, size=211859)

In [583... X\_df\_dummy = pd.DataFrame(X\_df\_dummy).rename(columns={30: 'feature\_95', 31: 'feature\_96'})

In [584... for feat in cat\_feats:  
X\_df\_dummy[feat] = sps.bernoulli.rvs(size=211859, p=0.5)

In [586... dummy = Pool(  
data=X\_df\_dummy,  
label=y\_dummy,  
group\_id=Q\_train,  
cat\_features=np.array(cat\_data.columns)  
)

In [ ]: parameters['loss\_function'] = 'QueryRMSE'  
parameters['train\_dir'] = 'QueryRMSE'  
  
model = CatBoostRanker(\*\*parameters)  
model.fit(dummy, eval\_set=test, plot=False, use\_best\_model=True)

In [596... model.best\_score\_['validation']

Out[596]: {'NDCG:top=10;type=Base': 0.3308435636209059,  
'PFound': 0.5481857128359304,  
'NDCG:top=5;type=Base': 0.31221416099814264,  
'QueryRMSE': 0.19102692323384216,  
'MAP:top=5': 0.04714722222222221,  
'PrecisionAt:top=5': 0.047000000000000014}

Получаем `NDCG:top=5;type=Base` = 0.312

При информативном входе `NDCG:top=5;type=Base` = 0.5

## Подход 2: Отбор признаков

Использовать one-hot проблематично для категориальных, потому что каждая фича будет закодирована как 0/1, и подача ее в линейную регрессию не позволит выбрать категориальную переменную в целом. Конечно, приблизительно можно брать сумму или еще что-то другое (они хотя-бы сонаправлены);

Попробуем использовать target encoding

```
In [603... from sklearn.linear_model import Lasso
import statsmodels.api as sm
```

```
In [659... data = pd.read_csv('intern_task.csv')
```

```
In [660... data = data.drop(columns=useless_feats)
```

```
In [661... max_rank = np.max(data['rank'])
data['rank'] /= max_rank
```

```
In [607... Q_test_indexes = np.random.choice(data['query_id'].unique(), size=200, replace=False)
Q_train_indexes = np.array([i for i in data['query_id'].unique() if i not in Q_test_indexes])
```

```
In [611... X_df = data.loc[np.where(data['query_id'].isin(Q_train_indexes))]
```

```
In [613... for feat in cat_feats:
    means = X_df.groupby(feat)['rank'].mean()
    X_df[f'encoded_{feat}'] = X_df[feat].map(means)
X_df = X_df.drop(columns=cat_feats)
```

```
In [614... X_df.head()
```

```
Out[614]:
```

	rank	query_id	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	...	fe
0	0.00	10	1.0	0.0	1.0	3.0	3.0	0.333333	0.0	0.333333	...	
1	0.25	10	3.0	0.0	3.0	0.0	3.0	1.000000	0.0	1.000000	...	
2	0.00	10	3.0	0.0	2.0	0.0	3.0	1.000000	0.0	0.666667	...	
3	0.25	10	3.0	0.0	3.0	0.0	3.0	1.000000	0.0	1.000000	...	
4	0.50	10	3.0	0.0	3.0	1.0	3.0	1.000000	0.0	1.000000	...	

5 rows × 142 columns

```
In [618... X_df.drop(columns=['rank', 'query_id']).shape
```

```
Out[618]: (211337, 140)
```

Гауссовская линейная модель

```
model = sm.OLS(X_df['rank'], X_df.drop(columns=['rank', 'query_id']))

results = model.fit()

print(results.summary())
```

OLS Regression Results						
=====						
Dep. Variable:	rank	R-squared (uncentered):	0.480			
Model:	OLS	Adj. R-squared (uncentered):	0.480			
Method:	Least Squares	F-statistic:	1436.			
Date:	Thu, 25 Apr 2024	Prob (F-statistic):	0.00			
Time:	02:47:47	Log-Likelihood:	46992.			
No. Observations:	211337	AIC:	-9.371e+04			
Df Residuals:	211201	BIC:	-9.232e+04			
Df Model:	136					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]
-----						
feature_0	-0.0437	0.008	-5.728	0.000	-0.059	-0.029
feature_1	0.0064	0.004	1.449	0.147	-0.002	0.015
feature_2	-0.0097	0.003	-2.970	0.003	-0.016	-0.003
feature_3	0.0379	0.007	5.419	0.000	0.024	0.052
feature_4	0.0564	0.008	7.200	0.000	0.041	0.072
feature_5	-0.0227	0.023	-1.000	0.317	-0.067	0.022
feature_6	0.0519	0.020	2.608	0.009	0.013	0.091
feature_7	0.0680	0.011	6.225	0.000	0.047	0.089
feature_8	7.8717	8.706	0.904	0.366	-9.192	24.935
feature_9	-0.0711	0.023	-3.082	0.002	-0.116	-0.026
feature_10	-0.4636	0.513	-0.903	0.366	-1.470	0.542
feature_11	-0.4637	0.513	-0.903	0.366	-1.470	0.542
feature_12	-0.4636	0.513	-0.903	0.366	-1.470	0.542
feature_13	-0.4661	0.513	-0.908	0.364	-1.472	0.540
feature_14	0.4636	0.513	0.903	0.366	-0.542	1.470
feature_15	-0.0080	0.005	-1.566	0.117	-0.018	0.002
feature_16	-0.0067	0.001	-9.539	0.000	-0.008	-0.005
feature_17	0.0054	0.001	8.797	0.000	0.004	0.007
feature_18	0.0007	0.000	1.747	0.081	-8.96e-05	0.002
feature_19	0.0081	0.005	1.581	0.114	-0.002	0.018
feature_20	-0.3830	0.424	-0.904	0.366	-1.213	0.447
feature_21	0.7521	0.805	0.934	0.350	-0.825	2.330
feature_22	0.7274	0.805	0.904	0.366	-0.850	2.305
feature_23	0.7025	0.805	0.873	0.383	-0.875	2.280
feature_24	-0.7281	0.805	-0.905	0.366	-2.306	0.849
feature_25	-0.0066	0.002	-3.068	0.002	-0.011	-0.002
feature_26	0.0165	0.012	1.347	0.178	-0.008	0.041
feature_27	-0.0165	0.006	-2.840	0.005	-0.028	-0.005
feature_28	-0.0364	0.020	-1.861	0.063	-0.075	0.002
feature_29	0.0085	0.002	3.998	0.000	0.004	0.013
feature_30	0.0067	0.002	3.916	0.000	0.003	0.010
feature_31	-0.0258	0.010	-2.714	0.007	-0.044	-0.007
feature_32	-0.0096	0.004	-2.407	0.016	-0.017	-0.002
feature_33	0.0050	0.014	0.349	0.727	-0.023	0.033
feature_34	-0.0062	0.002	-3.650	0.000	-0.010	-0.003
feature_35	-0.0039	0.004	-0.904	0.366	-0.012	0.005
feature_36	-1689.1257	1868.135	-0.904	0.366	-5350.623	1972.372
feature_37	-1689.0766	1868.135	-0.904	0.366	-5350.574	1972.421
feature_38	-1689.0770	1868.134	-0.904	0.366	-5350.574	1972.420
feature_39	1689.1297	1868.135	0.904	0.366	-1972.368	5350.627
feature_40	4.793e-05	1.7e-05	2.825	0.005	1.47e-05	8.12e-05
feature_41	0.0013	0.001	2.434	0.015	0.000	0.002
feature_42	0.0003	9.7e-05	3.272	0.001	0.000	0.001
feature_43	-0.0100	0.015	-0.674	0.500	-0.039	0.019
feature_44	-4.875e-05	1.69e-05	-2.893	0.004	-8.18e-05	-1.57e-05

feature_45	0.1960	0.043	4.553	0.000	0.112	0.280
feature_46	-0.0406	0.015	-2.682	0.007	-0.070	-0.011
feature_47	0.0269	0.009	2.870	0.004	0.009	0.045
feature_48	-0.0002	0.000	-0.860	0.390	-0.001	0.000
feature_49	-0.4213	0.054	-7.823	0.000	-0.527	-0.316
feature_50	-0.3035	0.188	-1.615	0.106	-0.672	0.065
feature_51	0.0379	0.035	1.091	0.275	-0.030	0.106
feature_52	0.0237	0.027	0.892	0.373	-0.028	0.076
feature_53	0.4043	0.143	2.829	0.005	0.124	0.684
feature_54	-0.5264	0.217	-2.421	0.015	-0.953	-0.100
feature_55	-0.8555	0.174	-4.918	0.000	-1.196	-0.515
feature_56	0.0308	0.023	1.364	0.172	-0.013	0.075
feature_57	0.0009	0.021	0.045	0.964	-0.040	0.042
feature_58	-0.5969	0.102	-5.850	0.000	-0.797	-0.397
feature_59	0.8781	0.197	4.459	0.000	0.492	1.264
feature_60	0.8238	0.346	2.380	0.017	0.145	1.502
feature_61	-0.0104	0.045	-0.233	0.815	-0.098	0.077
feature_62	-0.0697	0.041	-1.691	0.091	-0.151	0.011
feature_63	0.4416	0.189	2.340	0.019	0.072	0.812
feature_66	0.0647	0.073	0.884	0.377	-0.079	0.208
feature_67	0.0910	0.066	1.372	0.170	-0.039	0.221
feature_68	5.9867	1.050	5.701	0.000	3.928	8.045
feature_69	-6.5696	1.034	-6.351	0.000	-8.597	-4.542
feature_70	0.0001	0.001	0.272	0.785	-0.001	0.001
feature_71	-0.0032	0.001	-3.942	0.000	-0.005	-0.002
feature_73	0.0010	0.001	0.892	0.373	-0.001	0.003
feature_74	-0.0002	0.001	-0.348	0.728	-0.001	0.001
feature_75	-0.0028	0.001	-5.126	0.000	-0.004	-0.002
feature_76	-0.0043	0.002	-2.778	0.005	-0.007	-0.001
feature_77	-0.0010	0.001	-1.297	0.195	-0.003	0.001
feature_78	-0.0025	0.002	-1.523	0.128	-0.006	0.001
feature_79	0.0024	0.001	4.605	0.000	0.001	0.003
feature_80	0.0002	0.000	0.510	0.610	-0.001	0.001
feature_81	0.0038	0.001	3.260	0.001	0.002	0.006
feature_82	0.0026	0.001	4.551	0.000	0.002	0.004
feature_83	0.0077	0.001	5.994	0.000	0.005	0.010
feature_84	-0.0004	0.000	-1.027	0.304	-0.001	0.000
feature_85	-0.0030	0.001	-2.378	0.017	-0.005	-0.001
feature_86	0.0002	0.003	0.079	0.937	-0.005	0.006
feature_87	-0.0082	0.002	-5.195	0.000	-0.011	-0.005
feature_88	-0.0117	0.003	-4.147	0.000	-0.017	-0.006
feature_89	0.0036	0.001	2.934	0.003	0.001	0.006
feature_90	-1.912e-06	1.18e-06	-1.620	0.105	-4.22e-06	4.01e-07
feature_91	-2.023e-05	8.3e-06	-2.437	0.015	-3.65e-05	-3.96e-06
feature_92	-6.184e-06	2.49e-06	-2.480	0.013	-1.11e-05	-1.3e-06
feature_93	-0.0002	0.000	-1.768	0.077	-0.000	2.39e-05
feature_94	1.899e-06	1.18e-06	1.616	0.106	-4.05e-07	4.2e-06
feature_101	-0.0792	0.013	-6.292	0.000	-0.104	-0.055
feature_102	-0.0677	0.007	-10.060	0.000	-0.081	-0.055
feature_103	-0.0571	0.009	-6.149	0.000	-0.075	-0.039
feature_104	0.0029	0.014	0.206	0.837	-0.025	0.030
feature_105	0.0004	0.000	1.015	0.310	-0.000	0.001
feature_106	0.0016	0.001	3.064	0.002	0.001	0.003
feature_107	0.0016	0.000	6.268	0.000	0.001	0.002
feature_108	0.0008	0.001	0.892	0.372	-0.001	0.002
feature_109	0.0005	0.000	1.264	0.206	-0.000	0.001
feature_110	-0.0042	0.002	-2.431	0.015	-0.008	-0.001
feature_111	-0.0025	0.001	-3.178	0.001	-0.004	-0.001
feature_112	0.0059	0.001	5.171	0.000	0.004	0.008
feature_113	0.0195	0.002	9.994	0.000	0.016	0.023
feature_114	0.0249	0.002	13.194	0.000	0.021	0.029
feature_115	0.0090	0.002	5.125	0.000	0.006	0.012
feature_116	-0.0007	0.000	-2.326	0.020	-0.001	-0.000
feature_117	-0.0064	0.001	-11.731	0.000	-0.008	-0.005
feature_118	-0.0041	0.001	-5.674	0.000	-0.006	-0.003
feature_119	-0.0128	0.002	-6.809	0.000	-0.017	-0.009



feature_120	-0.0008	0.002	-0.460	0.645	-0.004	0.003
feature_121	0.0032	0.001	5.030	0.000	0.002	0.004
feature_122	0.0001	0.001	0.158	0.874	-0.002	0.002
feature_123	-0.0137	0.001	-10.010	0.000	-0.016	-0.011
feature_124	-0.0134	0.002	-6.837	0.000	-0.017	-0.010
feature_125	-0.0063	0.000	-15.428	0.000	-0.007	-0.005
feature_126	0.0004	4.98e-05	7.720	0.000	0.000	0.000
feature_127	-3.834e-10	1.35e-10	-2.837	0.005	-6.48e-10	-1.19e-10
feature_128	0.0013	5.25e-05	24.506	0.000	0.001	0.001
feature_129	7.116e-07	2.21e-08	32.150	0.000	6.68e-07	7.55e-07
feature_130	1.226e-07	2.54e-08	4.821	0.000	7.28e-08	1.72e-07
feature_131	6.569e-05	1.68e-05	3.916	0.000	3.28e-05	9.86e-05
feature_132	-0.0003	1.22e-05	-21.267	0.000	-0.000	-0.000
feature_133	5.054e-08	9.38e-09	5.386	0.000	3.21e-08	6.89e-08
feature_134	-6.209e-08	2.25e-08	-2.754	0.006	-1.06e-07	-1.79e-08
feature_135	-5.161e-11	1.51e-10	-0.342	0.732	-3.47e-10	2.44e-10
feature_136	-0.0792	0.047	-1.673	0.094	-0.172	0.014
feature_137	-0.0119	0.014	-0.867	0.386	-0.039	0.015
feature_138	-0.0002	0.001	-0.360	0.719	-0.001	0.001
feature_139	0.0019	0.018	0.105	0.916	-0.033	0.037
feature_140	0.1999	0.394	0.507	0.612	-0.572	0.972
feature_141	1.2725	0.392	3.249	0.001	0.505	2.040
feature_142	0.7284	0.805	0.905	0.365	-0.849	2.306
feature_143	-1689.1319	1868.135	-0.904	0.366	-5350.630	1972.366
encoded_feature_95	0.7522	0.173	4.352	0.000	0.413	1.091
encoded_feature_96	-0.1287	0.098	-1.316	0.188	-0.320	0.063
encoded_feature_97	0.3932	0.048	8.277	0.000	0.300	0.486
encoded_feature_98	0.4344	0.111	3.923	0.000	0.217	0.651
encoded_feature_99	-0.3024	0.154	-1.963	0.050	-0.604	-0.000

---

Omnibus:	33681.060	Durbin-Watson:	1.836
Prob(Omnibus):	0.000	Jarque-Bera (JB):	56909.588
Skew:	1.061	Prob(JB):	0.00
Kurtosis:	4.400	Cond. No.	1.06e+16

---

Notes:

[1]  $R^2$  is computed without centering (uncentered) since the model does not contain a constant.

[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[3] The smallest eigenvalue is 1.84e-14. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

Применим МПГ. Для 48 признаков гипотеза о незначимости отвергается. Попробуем обучить модель на них.

```
In [662... from statsmodels.stats.multitest import multipletests

(multipletests(results.pvalues)[0] == True).sum()
```

Out[662]: 48

```
In [663... indexes = np.where(multipletests(results.pvalues)[0] == True)
feats = data.drop(columns=['rank', 'query_id']).columns[indexes[0]]
```

```
In [664... feats
```

```
Out[664]: Index(['feature_0', 'feature_3', 'feature_4', 'feature_7', 'feature_16',
        'feature_17', 'feature_29', 'feature_30', 'feature_34', 'feature_45',
        'feature_49', 'feature_55', 'feature_58', 'feature_59', 'feature_68',
        'feature_69', 'feature_71', 'feature_75', 'feature_79', 'feature_82',
        'feature_83', 'feature_87', 'feature_88', 'feature_95', 'feature_96',
        'feature_97', 'feature_102', 'feature_107', 'feature_108',
```

```

'feature_109', 'feature_110', 'feature_112', 'feature_113',
'feature_114', 'feature_116', 'feature_118', 'feature_119',
'feature_120', 'feature_121', 'feature_123', 'feature_124',
'feature_125', 'feature_126', 'feature_127', 'feature_128',
'feature_139', 'feature_141', 'feature_142'],
dtype='object')

```

```

In [665... data[list(set(cat_feats) & set(feats))] = data[list(set(cat_feats) & set(feats))].astype
data.head()

```

```

Out[665]:

```

	rank	query_id	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	...	fe
0	0.00	10	1.0	0.0	1.0	3.0	3.0	0.333333	0.0	0.333333	...	
1	0.25	10	3.0	0.0	3.0	0.0	3.0	1.000000	0.0	1.000000	...	
2	0.00	10	3.0	0.0	2.0	0.0	3.0	1.000000	0.0	0.666667	...	
3	0.25	10	3.0	0.0	3.0	0.0	3.0	1.000000	0.0	1.000000	...	
4	0.50	10	3.0	0.0	3.0	1.0	3.0	1.000000	0.0	1.000000	...	

5 rows × 142 columns

```

In [666... Y_data = data['rank']
Q_data = data['query_id']
X_data = data.drop(columns=['rank', 'query_id'])

```

## Обучение

```

In [667... Q_test_indexes = np.random.choice(Q_data.unique(), size=200, replace=False)
Q_train_indexes = np.array([i for i in Q_data.unique() if i not in Q_test_indexes])
assert(Q_train_indexes.shape[0] == 1800 and Q_test_indexes.shape[0] == 200)

```

```

In [668... X_train_df = X_data.loc[np.where(Q_data.isin(Q_train_indexes))]
X_test_df = X_data.loc[np.where(Q_data.isin(Q_test_indexes))]

y_train = Y_data.loc[np.where(Q_data.isin(Q_train_indexes))]
y_test = Y_data.loc[np.where(Q_data.isin(Q_test_indexes))]

Q_train = Q_data.loc[np.where(Q_data.isin(Q_train_indexes))]
Q_test = Q_data.loc[np.where(Q_data.isin(Q_test_indexes))]

```

```

In [669... X_train_df.columns = X_train_df.columns.astype(str)
X_test_df.columns = X_test_df.columns.astype(str)

```

```

In [671... train = Pool(
    data=X_train_df,
    label=y_train,
    group_id=Q_train,
    cat_features=np.array(list(set(cat_feats) & set(feats)))
)

test = Pool(
    data=X_test_df,
    label=y_test,
    group_id=Q_test,
    cat_features=np.array(list(set(cat_feats) & set(feats)))
)

```

```

In [672... losses = ['RMSE', 'QueryRMSE']

```

```

In [673... result_df_2 = None
for loss_func in tqdm(losses):
    metrics = train_ranker(loss_func)
    if result_df_2 is None:
        result_df_2 = dict_to_dataframe(metrics, loss_func)
    else:
        result_df_2 = dict_to_dataframe(metrics_dict, loss_func, result_df_2)

0%|          | 0/2 [00:00<?, ?it/s]
C:\Users\ztimu\anaconda3\lib\site-packages\catboost\core.py:6219: RuntimeWarning: Regression loss ('RMSE') ignores an important ranking parameter 'group_id'
  warnings.warn("Regression loss ('{}') ignores an important ranking parameter 'group_id'".format(loss_function), RuntimeWarning)

```

```

In [674... np.round(result_df_2['Test'][['NDCG:top=10;type=Base', 'PFound', 'NDCG:top=5;type=Base',

```

```

Out[674]:

```

	NDCG:top=10;type=Base	PFound	NDCG:top=5;type=Base	MAP:top=5	PrecisionAt:top=5
RMSE	0.57	0.75	0.57	0.17	0.15
Model1	0.50	0.68	0.48	0.12	0.10

## Совсем без обработки

```

In [675... data = pd.read_csv('intern_task.csv')

```

```

In [676... data = data.drop(columns=useless_feats)

```

```

In [677... Y_data = data['rank']
Q_data = data['query_id']
X_data = data.drop(columns=['rank', 'query_id'])

```

```

In [680... Q_test_indexes = np.random.choice(Q_data.unique(), size=200, replace=False)
Q_train_indexes = np.array([i for i in Q_data.unique() if i not in Q_test_indexes])
assert(Q_train_indexes.shape[0] == 1800 and Q_test_indexes.shape[0] == 200)

X_train_df = X_data.loc[np.where(Q_data.isin(Q_train_indexes))]
X_test_df = X_data.loc[np.where(Q_data.isin(Q_test_indexes))]

y_train = Y_data.loc[np.where(Q_data.isin(Q_train_indexes))]
y_test = Y_data.loc[np.where(Q_data.isin(Q_test_indexes))]

y_test /= max(y_test)
y_train /= max(y_train)

Q_train = Q_data.loc[np.where(Q_data.isin(Q_train_indexes))]
Q_test = Q_data.loc[np.where(Q_data.isin(Q_test_indexes))]

train = Pool(
    data=X_train_df,
    label=y_train,
    group_id=Q_train
)

test = Pool(
    data=X_test_df,
    label=y_test,
    group_id=Q_test
)

```

```

In [681... losses = ['RMSE', 'QueryRMSE']

```

```

result_df_3 = None
for loss_func in tqdm(losses):
    metrics = train_ranker(loss_func)
    if result_df_3 is None:
        result_df_3 = dict_to_dataframe(metrics, loss_func)
    else:
        result_df_3 = dict_to_dataframe(metrics_dict, model_name, result_df_3)

```

0%| | 0/2 [00:00<?, ?it/s]

C:\Users\ztimu\anaconda3\lib\site-packages\catboost\core.py:6219: RuntimeWarning: Regression loss ('RMSE') ignores an important ranking parameter 'group\_id'  
warnings.warn("Regression loss ('{}') ignores an important ranking parameter 'group\_id'".format(loss\_function), RuntimeWarning)

Выводы:

С удалением неинформативных признаков, добав. катег. признаки.  
PCA 30 компонент.

In [684... np.round(result\_df[['Test']][['NDCG:top=10;type=Base', 'PFound', 'NDCG:top=5;type=Base', 'MAP:top=5', 'PrecisionAt:top=5']])

Out[684]:

	NDCG:top=10;type=Base	PFound	NDCG:top=5;type=Base	MAP:top=5	PrecisionAt:top=5
<b>CatBoostRanker</b>					
RMSE	0.48	0.68	0.48	0.11	0.10
QueryRMSE	0.50	0.68	0.48	0.12	0.10
PairLogit	0.49	0.66	0.47	0.10	0.10
YetiRank	0.50	0.69	0.50	0.13	0.11

С удалением неинформативных признаков, добав. катег. признаки.  
Отбор признаков через ttest.

In [685... np.round(result\_df\_2[['Test']][['NDCG:top=10;type=Base', 'PFound', 'NDCG:top=5;type=Base', 'MAP:top=5', 'PrecisionAt:top=5']])

Out[685]:

	NDCG:top=10;type=Base	PFound	NDCG:top=5;type=Base	MAP:top=5	PrecisionAt:top=5
RMSE	0.57	0.75	0.57	0.17	0.15
Model1	0.50	0.68	0.48	0.12	0.10

С удалением неинформативных признаков.

In [686... np.round(result\_df\_3[['Test']][['NDCG:top=10;type=Base', 'PFound', 'NDCG:top=5;type=Base', 'MAP:top=5', 'PrecisionAt:top=5']])

Out[686]:

	NDCG:top=10;type=Base	PFound	NDCG:top=5;type=Base	MAP:top=5	PrecisionAt:top=5
RMSE	0.58	0.77	0.59	0.20	0.18
Model1	0.50	0.68	0.48	0.12	0.10

Лучшее качество по NDCG:top=5 0.59 с минимальной обработкой данных