Table of Contents

Task description

The first challenge of the Playground series in 2024 is offering a binary classification problem for predicting bank customer churn. Submissions are evaluated by ROC-AUC metric.

My goal is to try to find any insights in the data and build a solid model which might be 1-2% below the best score.

IMPORTS

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from catboost import CatBoostClassifier
from sklearn.metrics import roc_auc_score
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/playground-series-s4e1/sample_submission.csv
/kaggle/input/playground-series-s4e1/train.csv
/kaggle/input/playground-series-s4e1/test.csv
```

In [2]:
```
data=pd.read_csv("/kaggle/input/playground-series-s4e1/train.csv")
test=pd.read_csv("/kaggle/input/playground-series-s4e1/test.csv")
sample_submission=pd.read_csv("/kaggle/input/playground-series-s4e1/sample_submission.cs
train = data.copy()
```

Dataset overview

In [3]:
```
print('Train shape:', train.shape)
print('Test shape: ', test.shape)
```

```
Train shape: (165034, 14)
Test shape:  (110023, 13)
```

In [4]:
```
train.head()
```

Out[4]:

	id	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	H
0	0	15674932	Okwudilichukwu	668	France	Male	33.0	3	0.00	2	
1	1	15749177	Okwudiliolisa	627	France	Male	33.0	1	0.00	2	
2	2	15694510	Hsueh	678	France	Male	40.0	10	0.00	2	
3	3	15741417	Kao	581	France	Male	34.0	2	148882.54	1	
4	4	15766172	Chiemenam	716	Spain	Male	33.0	5	0.00	2	

In [5]:
```
test.head()
```

Out[5]:

	id	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	Ha
0	165034	15773898	Lucchese	586	France	Female	23.0	2	0.00	2	
1	165035	15782418	Nott	683	France	Female	46.0	2	0.00	1	
2	165036	15807120	K?	656	France	Female	34.0	7	0.00	2	
3	165037	15808905	O'Donnell	681	France	Male	36.0	8	0.00	1	
4	165038	15607314	Higgins	752	Germany	Male	38.0	10	121263.62	1	

In [6]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 165034 entries, 0 to 165033
Data columns (total 14 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   id               165034 non-null  int64  
 1   CustomerId       165034 non-null  int64  
 2   Surname          165034 non-null  object 
 3   CreditScore      165034 non-null  int64  
 4   Geography        165034 non-null  object 
 5   Gender           165034 non-null  object 
 6   Age              165034 non-null  float64
 7   Tenure           165034 non-null  int64  
 8   Balance          165034 non-null  float64
 9   NumOfProducts    165034 non-null  int64  
 10  HasCrCard        165034 non-null  float64
 11  IsActiveMember   165034 non-null  float64
 12  EstimatedSalary  165034 non-null  float64
 13  Exited           165034 non-null  int64  
dtypes: float64(5), int64(6), object(3)
memory usage: 17.6+ MB
```

In [7]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110023 entries, 0 to 110022
Data columns (total 13 columns):
 #   Column           Non-Null Count   Dtype  
```

```
---  ------          --------------   -----
 0   id              110023 non-null  int64
 1   CustomerId      110023 non-null  int64
 2   Surname         110023 non-null  object
 3   CreditScore     110023 non-null  int64
 4   Geography       110023 non-null  object
 5   Gender          110023 non-null  object
 6   Age             110023 non-null  float64
 7   Tenure          110023 non-null  int64
 8   Balance         110023 non-null  float64
 9   NumOfProducts   110023 non-null  int64
 10  HasCrCard       110023 non-null  float64
 11  IsActiveMember  110023 non-null  float64
 12  EstimatedSalary 110023 non-null  float64
dtypes: float64(5), int64(5), object(3)
memory usage: 10.9+ MB
```

In [8]:
```
train.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	
id	165034.0	8.251650e+04	47641.356500	0.00	41258.25	82516.5	1.237748e+05	165
CustomerId	165034.0	1.569201e+07	71397.816791	15565701.00	15633141.00	15690169.0	1.575682e+07	1581
CreditScore	165034.0	6.564544e+02	80.103340	350.00	597.00	659.0	7.100000e+02	
Age	165034.0	3.812589e+01	8.867205	18.00	32.00	37.0	4.200000e+01	
Tenure	165034.0	5.020353e+00	2.806159	0.00	3.00	5.0	7.000000e+00	
Balance	165034.0	5.547809e+04	62817.663278	0.00	0.00	0.0	1.199395e+05	250
NumOfProducts	165034.0	1.554455e+00	0.547154	1.00	1.00	2.0	2.000000e+00	
HasCrCard	165034.0	7.539537e-01	0.430707	0.00	1.00	1.0	1.000000e+00	
IsActiveMember	165034.0	4.977702e-01	0.499997	0.00	0.00	0.0	1.000000e+00	
EstimatedSalary	165034.0	1.125748e+05	50292.865585	11.58	74637.57	117948.0	1.551525e+05	199
Exited	165034.0	2.115988e-01	0.408443	0.00	0.00	0.0	0.000000e+00	

In [9]:
```
test.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	
id	110023.0	2.200450e+05	31761.048671	165034.00	1.925395e+05	220045.00	2.475505e+05	
CustomerId	110023.0	1.569210e+07	71684.990992	15565701.00	1.563286e+07	15690175.00	1.575693e+07	15
CreditScore	110023.0	6.565308e+02	80.315415	350.00	5.970000e+02	660.00	7.100000e+02	
Age	110023.0	3.812221e+01	8.861550	18.00	3.200000e+01	37.00	4.200000e+01	
Tenure	110023.0	4.996637e+00	2.806148	0.00	3.000000e+00	5.00	7.000000e+00	
Balance	110023.0	5.533361e+04	62788.519675	0.00	0.000000e+00	0.00	1.201456e+05	
NumOfProducts	110023.0	1.553321e+00	0.544714	1.00	1.000000e+00	2.00	2.000000e+00	
HasCrCard	110023.0	7.530425e-01	0.431244	0.00	1.000000e+00	1.00	1.000000e+00	
IsActiveMember	110023.0	4.952328e-01	0.499980	0.00	0.000000e+00	0.00	1.000000e+00	
EstimatedSalary	110023.0	1.123151e+05	50277.048244	11.58	7.444033e+04	117832.23	1.546314e+05	

In [10]:
```
print('Train NONE elements:', train.isnull().sum().sum())
```

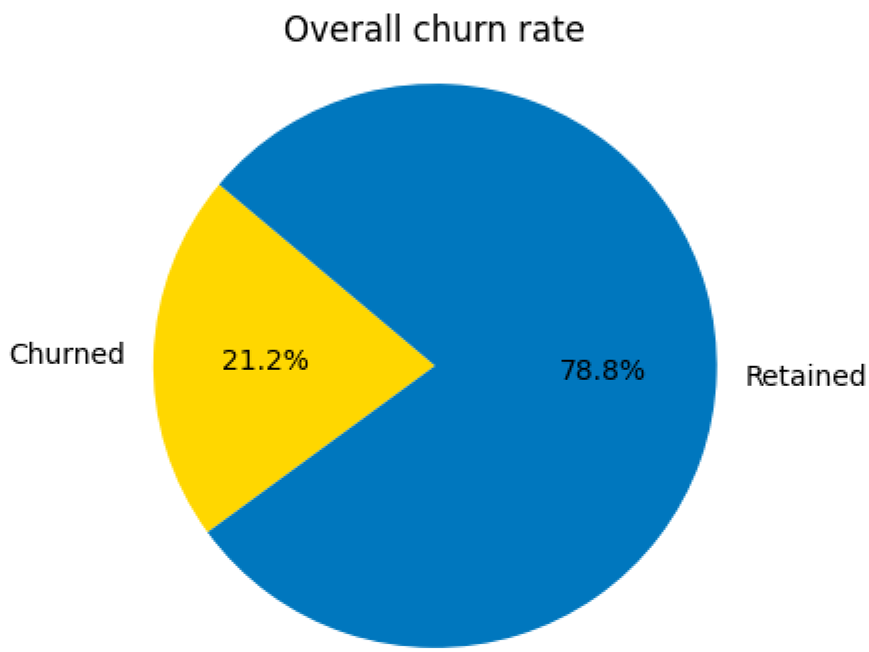
```
print('Test NONE elements:', test.isnull().sum().sum())
```

```
Train NONE elements: 0
Test NONE elements: 0
```

A large dataset without None values, most features are already numeric.

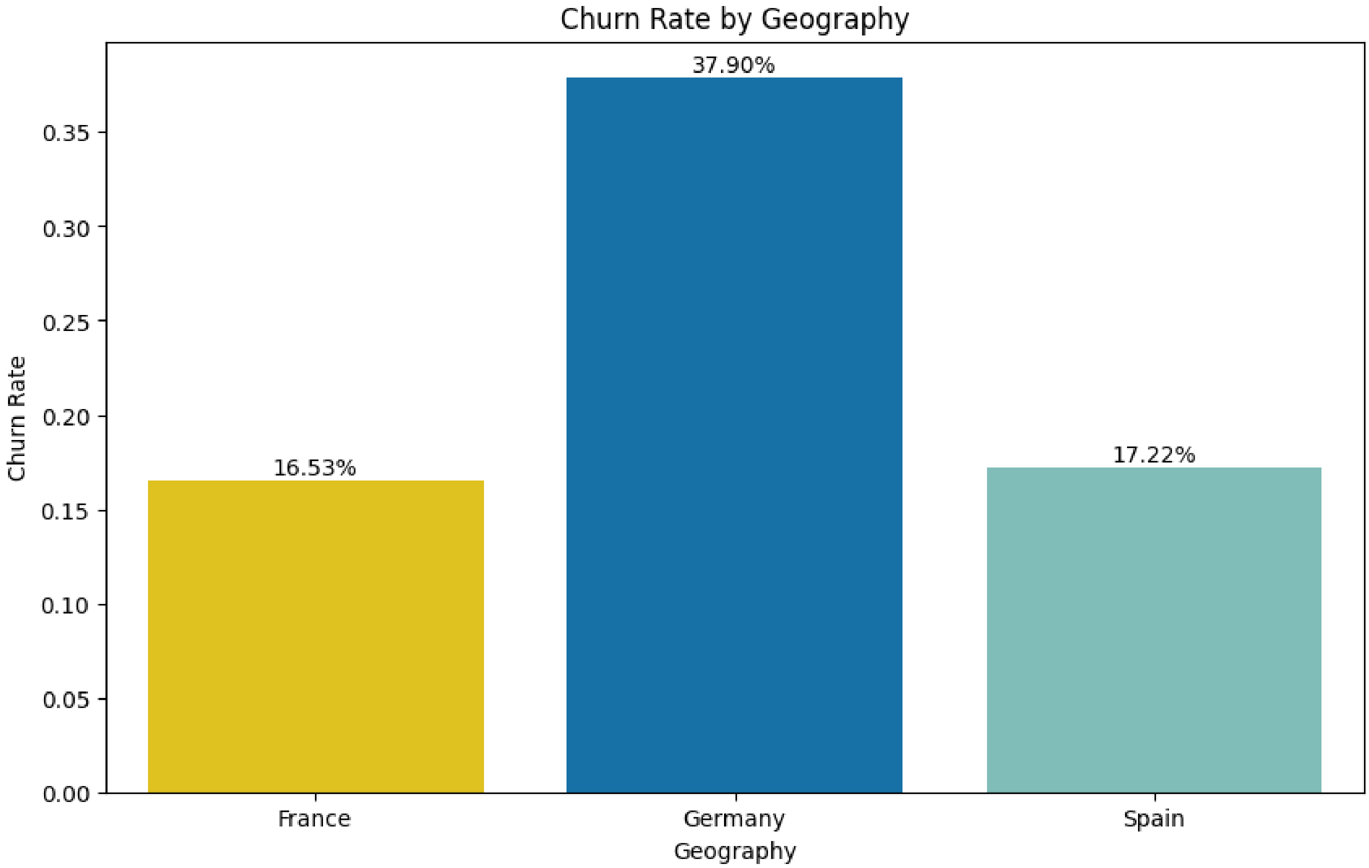
Dataset analysis (EDA)

In [11]:
```
# Churn rate
churned_cnt = data['Exited'].sum()
retained_cnt = len(data) - churned_cnt
plt.figure(figsize=(4, 4))
plt.pie([churned_cnt, retained_cnt], labels=['Churned', 'Retained'], colors=['#FFD700',
plt.axis('equal')
plt.title('Overall churn rate')
plt.show()
```



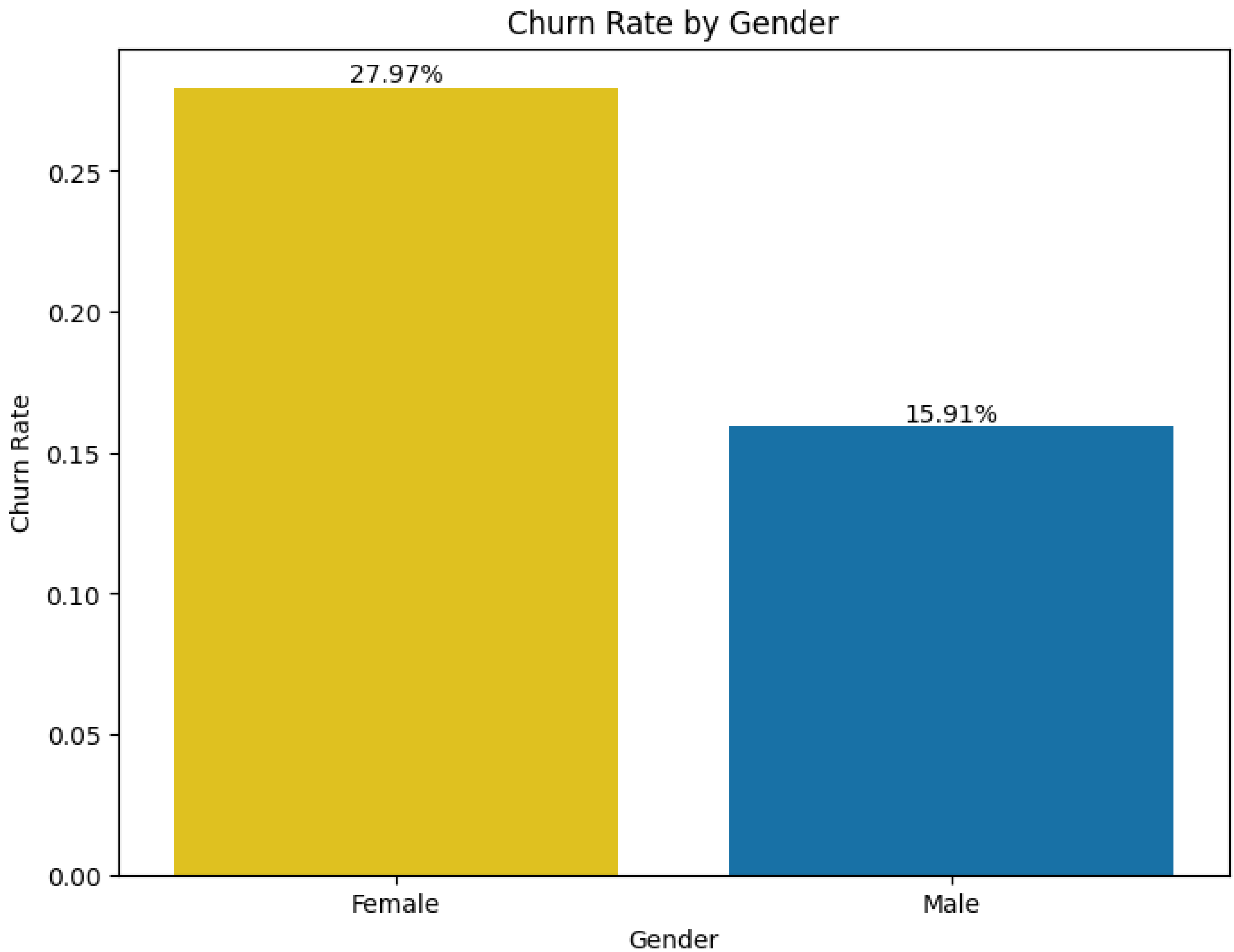
The overall churn rate from the train dataset is about 21%.

In [12]:
```
# Churn rate by country
country_churn = data.groupby('Geography')['Exited'].mean()
plt.figure(figsize=(10, 6))
sns.barplot(x=country_churn.index, y=country_churn.values, palette=['#FFD700', '#0077BE'
plt.title('Churn Rate by Geography')
plt.ylabel('Churn Rate')
for index, value in enumerate(country_churn.values):
    plt.text(index, value, f'{value:.2%}', ha='center', va='bottom')
plt.show()
```



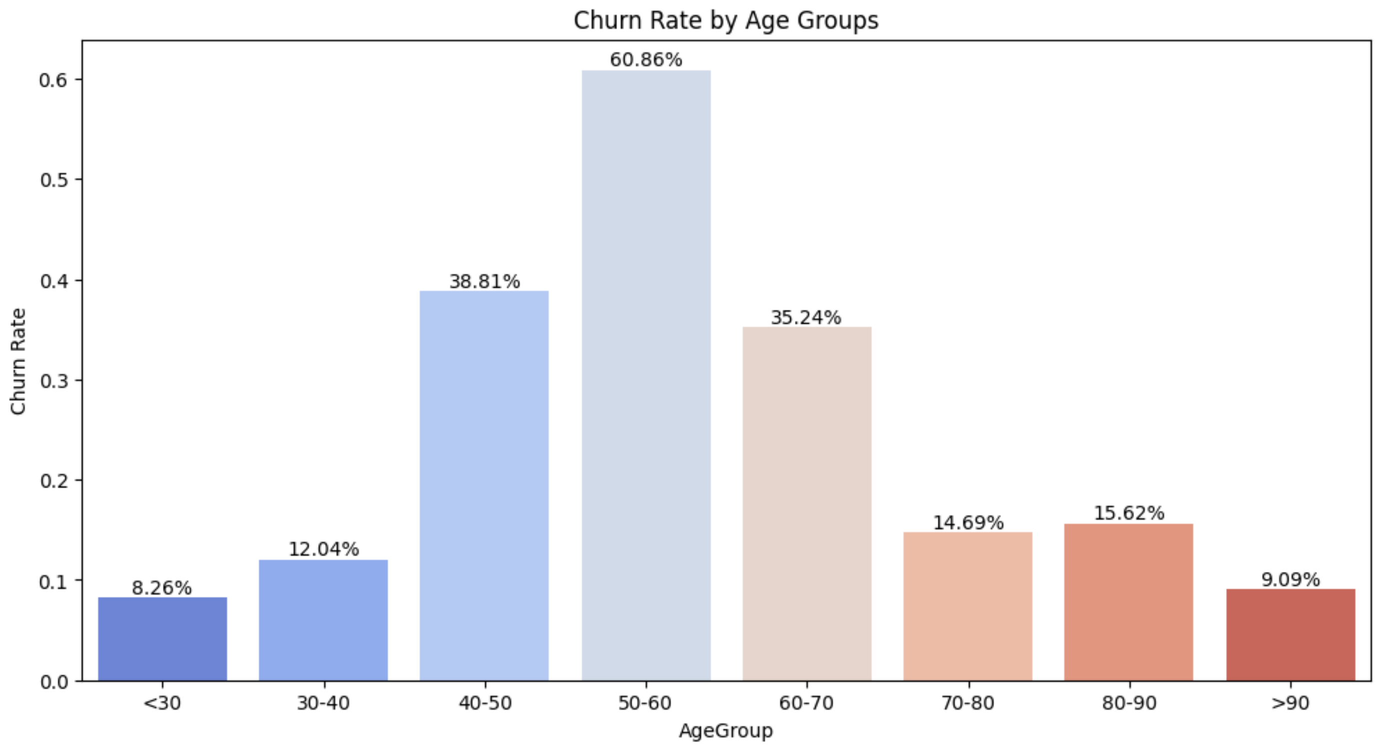
While churn rate in France and Spain is almost the same, Germany is the leader with almost 38%.

In [13]:
```
# Churn rate by gender
gender_churn = data.groupby('Gender')['Exited'].mean()
plt.figure(figsize=(8, 6))
sns.barplot(x=gender_churn.index, y=gender_churn.values, palette=['#FFD700', '#0077BE'])
plt.title('Churn Rate by Gender')
plt.ylabel('Churn Rate')
for index, value in enumerate(gender_churn.values):
    plt.text(index, value, f'{value:.2%}', ha='center', va='bottom')
plt.show()
```



Female customers have a higher churn rate.

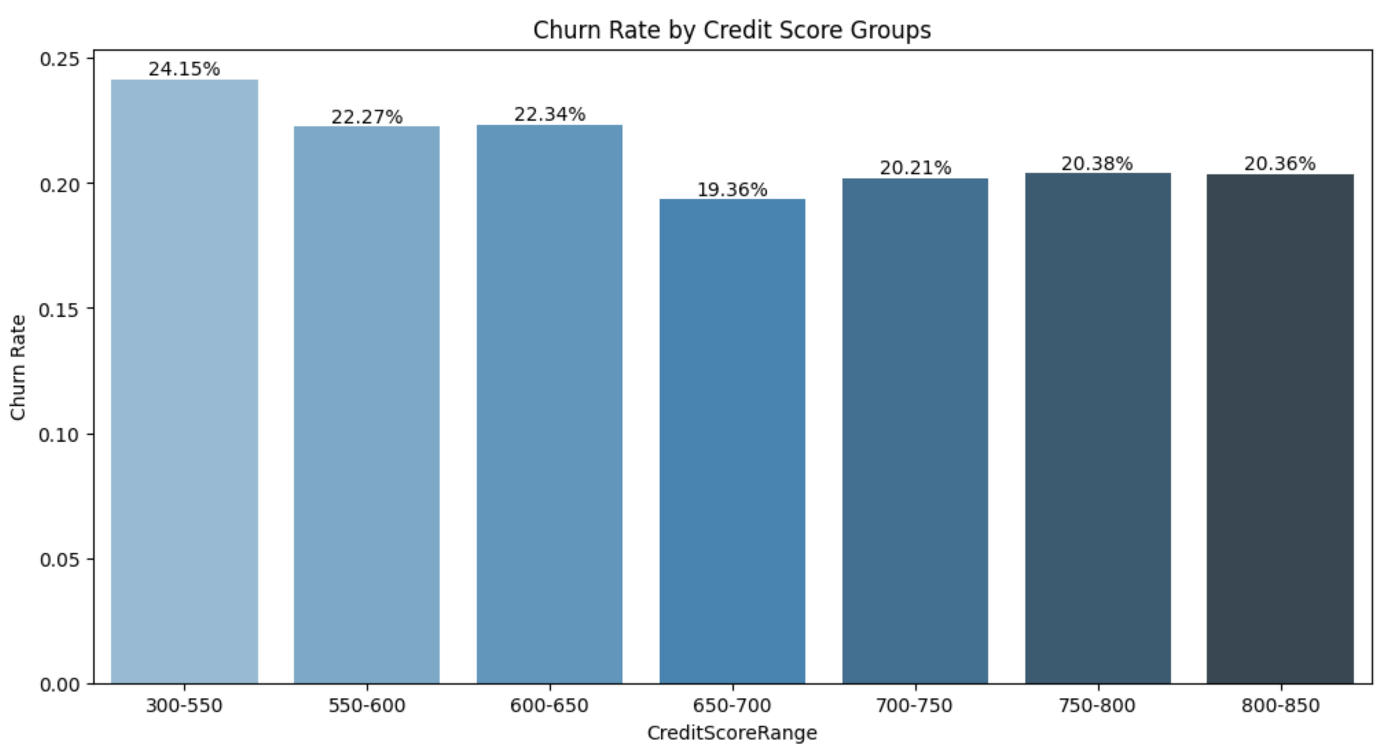
In [14]:
```
# Churn by Age groups
train['AgeGroup'] = pd.cut(train['Age'], bins=[0, 30, 40, 50, 60, 70, 80, 90, 100],
                           labels=['<30', '30-40', '40-50', '50-60', '60-70', '70-80', '8
age_group_churn = train.groupby('AgeGroup')['Exited'].mean()
plt.figure(figsize=(12, 6))
sns.barplot(x=age_group_churn.index, y=age_group_churn.values, palette=sns.color_palette
plt.title('Churn Rate by Age Groups')
plt.ylabel('Churn Rate')
for index, value in enumerate(age_group_churn.values):
    plt.text(index, value, f'{value:.2%}', ha='center', va='bottom')
plt.show()
```



Churn rates by age seems like a normal distribution with the peak at 50-60 group.

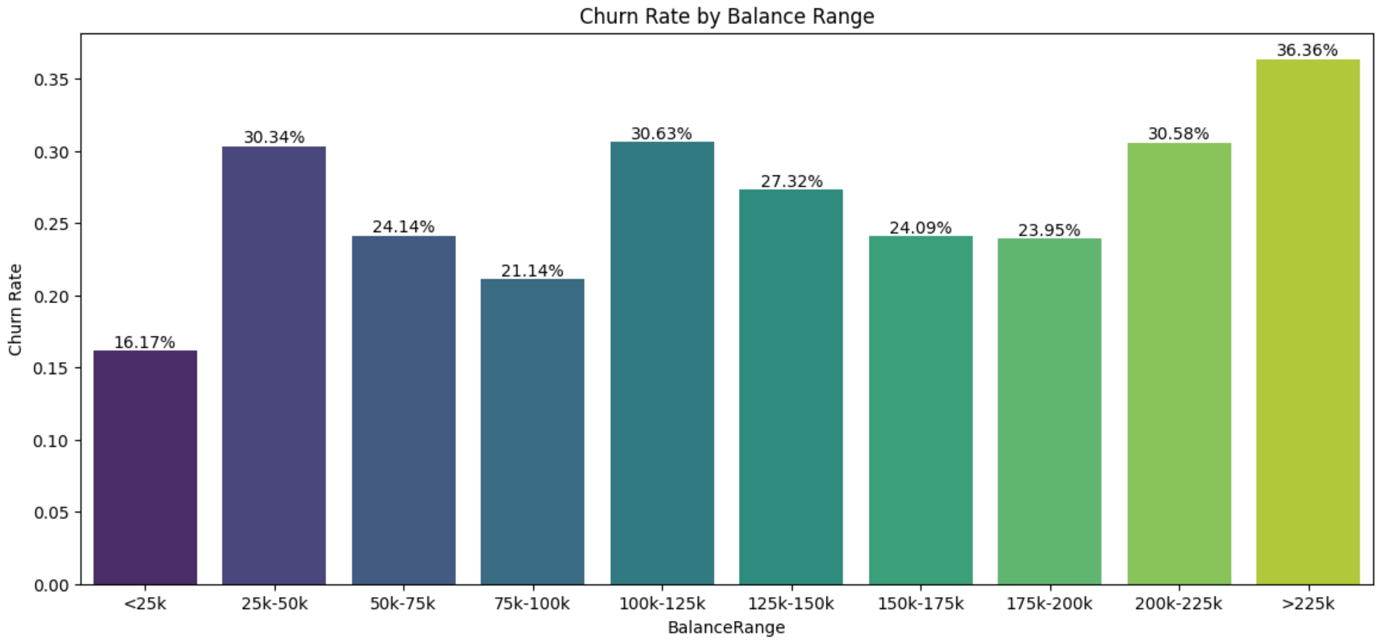
In [15]:

```
# Credit score analysis
train['CreditScoreRange'] = pd.cut(train['CreditScore'], bins=[300, 550, 600, 650, 700,
                                   labels=['300-550', '550-600', '600-650', '650-700', '7
cc_churn = train.groupby('CreditScoreRange')['Exited'].mean()
plt.figure(figsize=(12, 6))
sns.barplot(x=cc_churn.index, y=cc_churn.values, palette="Blues_d")
plt.title('Churn Rate by Credit Score Groups')
plt.ylabel('Churn Rate')
for index, value in enumerate(cc_churn.values):
    plt.text(index, value, f'{value:.2%}', ha='center', va='bottom')
plt.show()
```



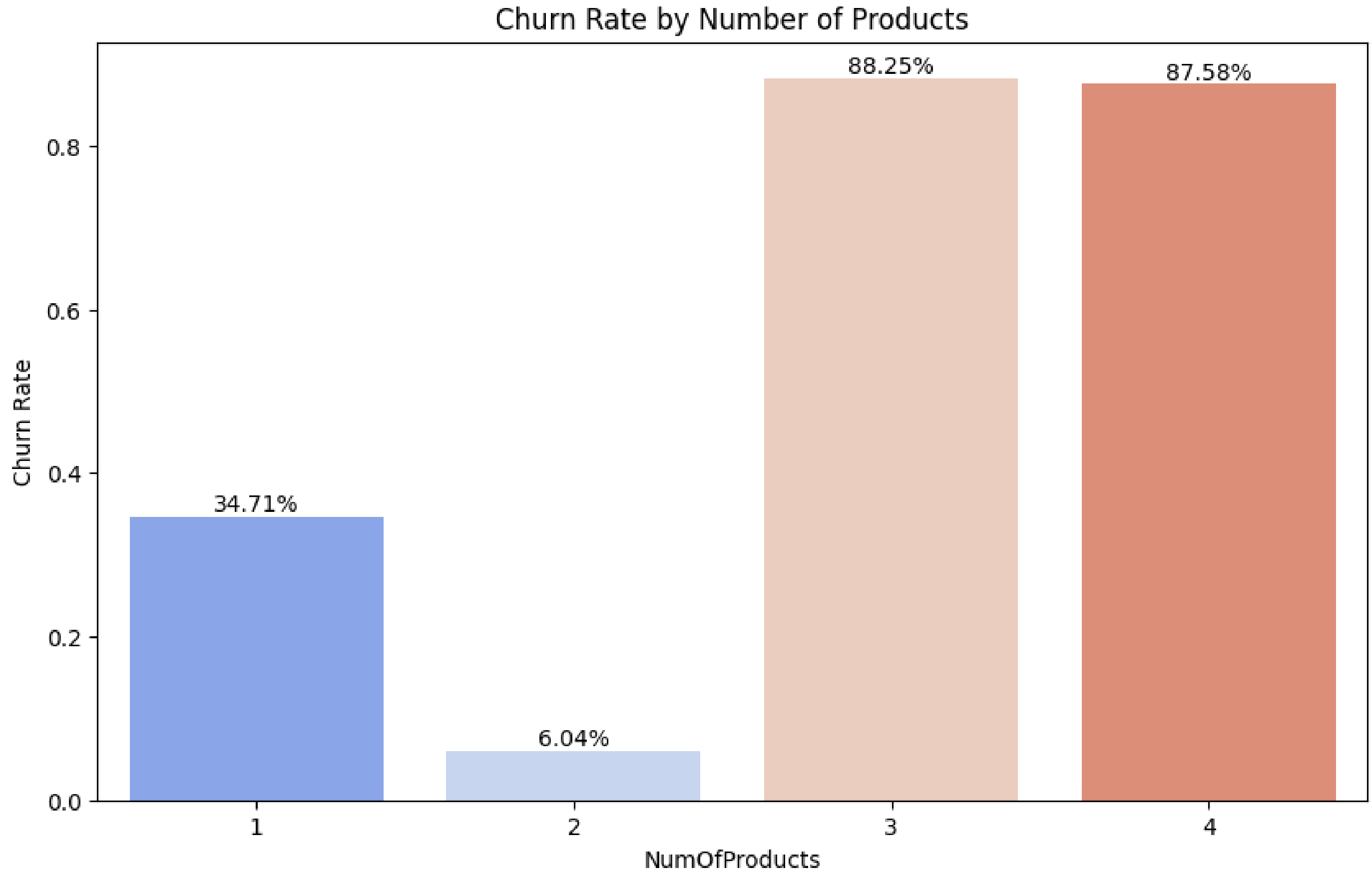
Almost no changes between Credit Score groups.

In [16]:
```
# Balance analysis
train['BalanceRange'] = pd.cut(train['Balance'], bins=[-1, 25000, 50000, 75000, 100000,
                              labels=['<25k', '25k-50k', '50k-75k', '75k-100k', '100k-12
b_churn = train.groupby('BalanceRange')['Exited'].mean()
plt.figure(figsize=(14, 6))
sns.barplot(x=b_churn.index, y=b_churn.values, palette="viridis")
plt.title('Churn Rate by Balance Range')
plt.ylabel('Churn Rate')
for index, value in enumerate(b_churn.values):
    plt.text(index, value, f'{value:.2%}', ha='center', va='bottom')
plt.show()
```



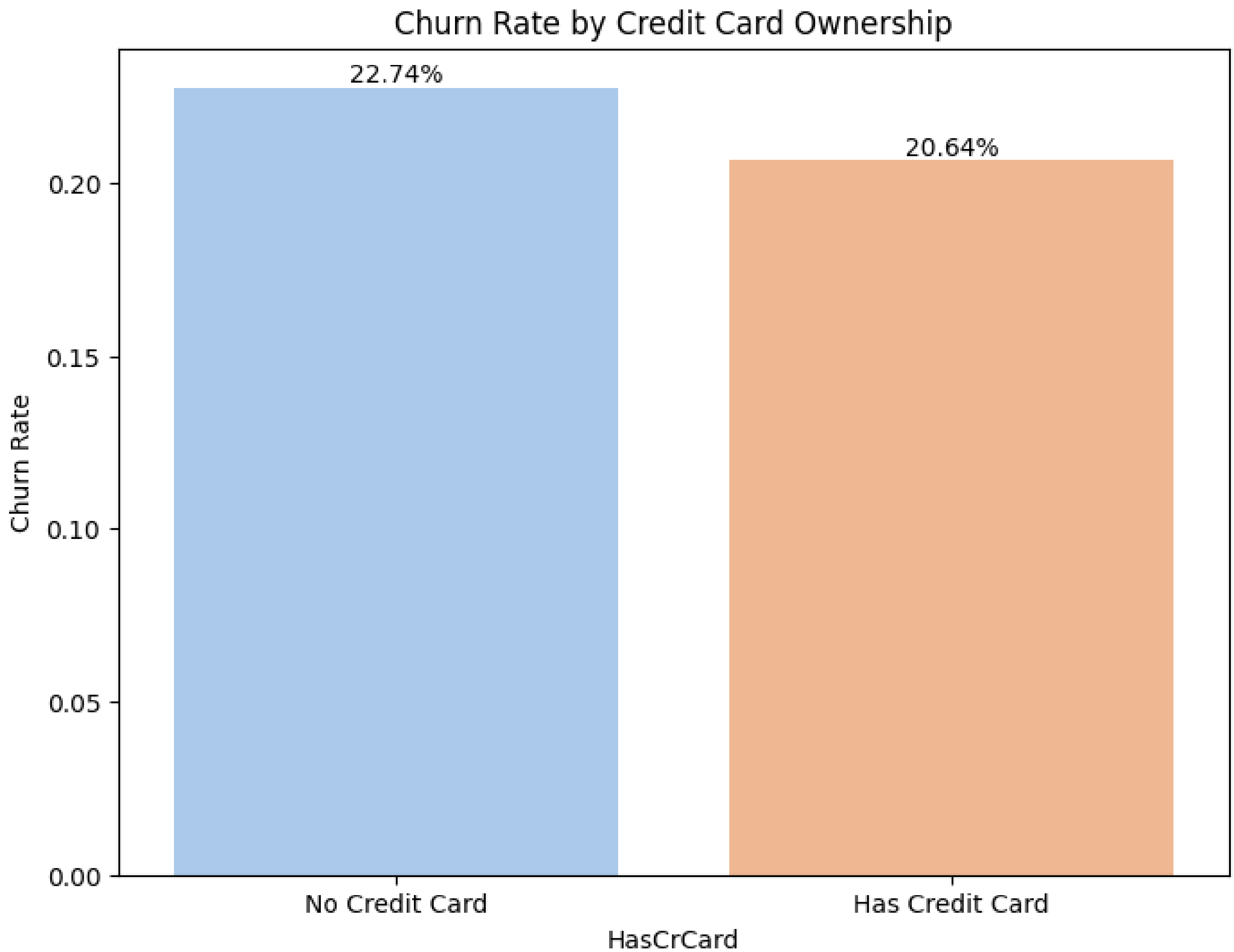
Customers with high balances have a big churn rate.

In [17]:
```
# Churn by number of products
churn_pn = train.groupby('NumOfProducts')['Exited'].mean()
plt.figure(figsize=(10, 6))
sns.barplot(x=churn_pn.index, y=churn_pn.values, palette="coolwarm")
plt.title('Churn Rate by Number of Products')
plt.ylabel('Churn Rate')
for index, value in enumerate(churn_pn.values):
    plt.text(index, value, f'{value:.2%}', ha='center', va='bottom')
plt.show()
```



Customers using 2 product has the smallest churn rate, just 6%. Customers with only one might be considered as an average, 34%, which is much less than 3-4 products users with more than 87%.

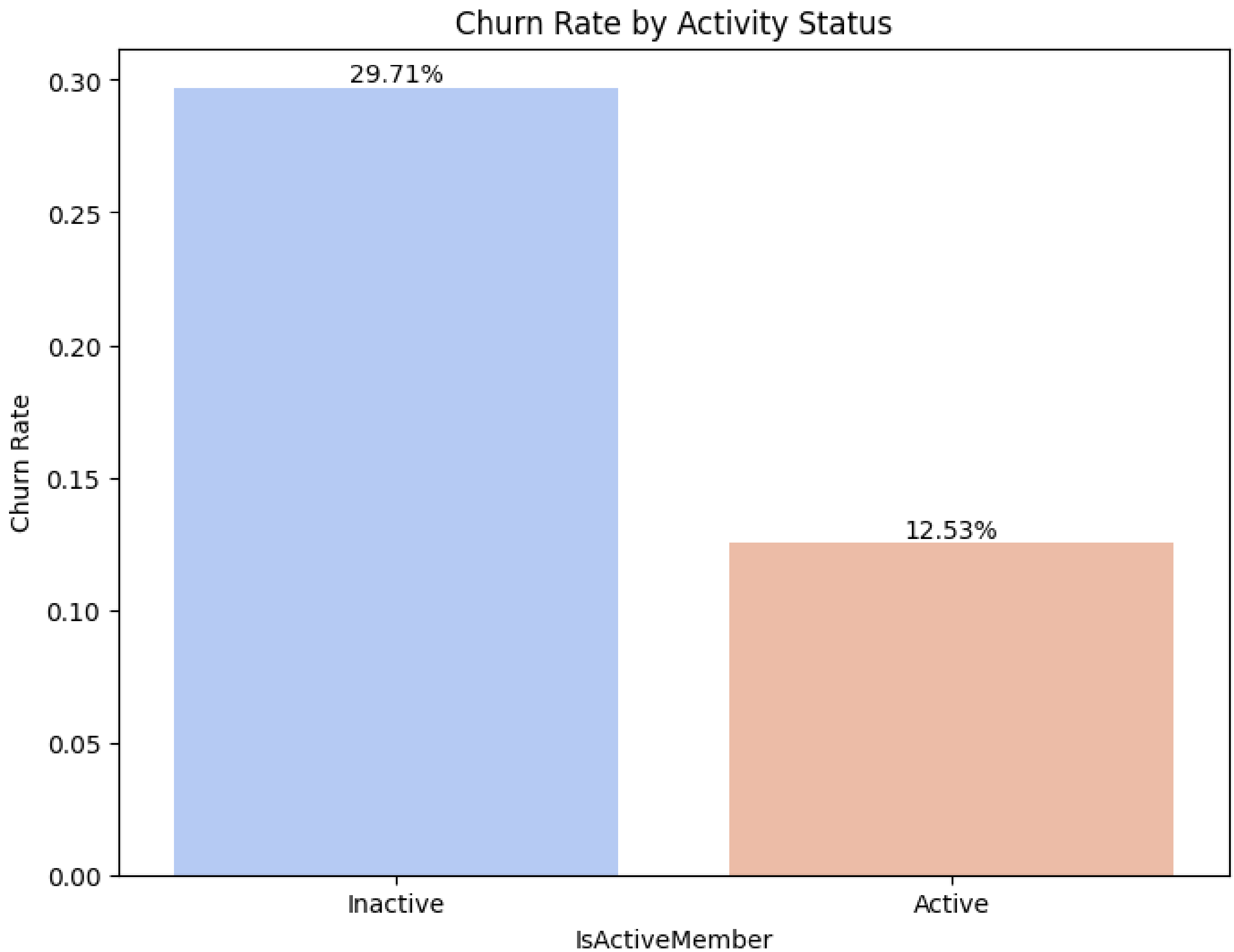
In [18]:
```
# Churn rate by credit card ownership
cco = train.groupby('HasCrCard')['Exited'].mean()
plt.figure(figsize=(8, 6))
sns.barplot(x=cco.index.map({0: 'No Credit Card', 1: 'Has Credit Card'}), y=cco.values, 
plt.title('Churn Rate by Credit Card Ownership')
plt.ylabel('Churn Rate')
for index, value in enumerate(cco.values):
    plt.text(index, value, f'{value:.2%}', ha='center', va='bottom')
plt.show()
```



Just a slight difference between Credit card holders and non-holders.

In [19]:

```
# Churn by activity status
as_c = train.groupby('IsActiveMember')['Exited'].mean()
plt.figure(figsize=(8, 6))
sns.barplot(x=as_c.index.map({0: 'Inactive', 1: 'Active'}), y=as_c.values, palette="cool
plt.title('Churn Rate by Activity Status')
plt.ylabel('Churn Rate')
for index, value in enumerate(as_c.values):
    plt.text(index, value, f'{value:.2%}', ha='center', va='bottom')
plt.show()
```



Inactive members have a significantly higher churn rate, compared to active members.

Conclusion: Country, gender, age, number of products and activity status are significant factors related to churn.

Preprocessing, data transformation, features engineering

I planned to drop Surname column as well, but let's inspect how many unique values are there:

In [20]:
```
sur_train = data['Surname'].value_counts()
print(sur_train)
sur_test = test['Surname'].value_counts()
print(sur_test)
```

```
Surname
Hsia         2456
T'ien        2282
Hs?          1611
Kao          1577
Maclean      1577
```

```
                ...
Samaniego        1
Lawley           1
Bonwick          1
Tennant          1
Elkins           1
Name: count, Length: 2797, dtype: int64
Surname
Hsia          1606
T'ien         1484
Hs?           1124
Maclean       1042
Ts'ui         1017
              ...
Wallwork         1
Praed            1
Kentish          1
Younger          1
Distefano        1
Name: count, Length: 2708, dtype: int64
```

Seems it might have some insights, let's keep it.

In [21]:
```
# Feature engineering

# Customer Engagement Score - assuming IsActiveMember and NumOfProducts are indicators o
data['EngagementScore'] = data['IsActiveMember'] * (data['NumOfProducts'] / data['NumOfP

# Wealth Segment by Balance
bins = [-1, 50000, 150000, float('inf')]
labels = ['Low', 'Medium', 'High']
data['WealthSegment'] = pd.cut(data['Balance'], bins=bins, labels=labels).astype(object)

# Product Concentration Index - inverse relationship with the number of products
data['ProductConcentration'] = 1 / (data['NumOfProducts'] + 0.01)

# Age Groups
age_bins = [0, 18, 30, 40, 50, 60, 70, 80, 90, 100]
age_labels = ['<18', '19-30', '31-40', '41-50', '51-60', '61-70', '71-80', '81-90', '>90
data['AgeGroup'] = pd.cut(data['Age'], bins=age_bins, labels=age_labels, right=False).as

# Same for the Test dataset

# Customer Engagement Score
test['EngagementScore'] = test['IsActiveMember'] * (test['NumOfProducts'] / test['NumOfP

# Wealth Segment by Balance
test['WealthSegment'] = pd.cut(test['Balance'], bins=bins, labels=labels).astype(object)

# Product Concentration Index
test['ProductConcentration'] = 1 / (test['NumOfProducts'] + 0.01)

# Age Groups
test['AgeGroup'] = pd.cut(test['Age'], bins=age_bins, labels=age_labels, right=False).as
```

In [22]:
```
y = data.Exited
X = data.drop(['id', 'CustomerId', 'Exited'], axis=1)

testY = test.id
test = test.drop(['id', 'CustomerId'], axis=1)

categorical_features = np.where(X.dtypes == 'object')[0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
```

Model training

I'm using CatBoost Classifier as from the past challanges I know this model perfomance is very high and it can handle categorical features as is.

In [23]:
```
cat = CatBoostClassifier(iterations=3500,
                         depth=5,
                         learning_rate=0.01,
                         early_stopping_rounds=1000,
                         loss_function='MultiClass',
                         eval_metric='AUC',
                         random_seed=122,
                         l2_leaf_reg=1,
                         max_ctr_complexity=15,
                         grow_policy='Lossguide',
                         max_leaves=128,
                         min_data_in_leaf=5,
                         verbose=500,
                         cat_features=categorical_features)

cat.fit(X_train, y_train)

y_pred = cat.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
auc_val = roc_auc_score(y_test, y_pred)

print("Model Accuracy:",accuracy)
print("AUC:",auc_val)
```

```
0:      total: 196ms    remaining: 11m 27s
500:    total: 55.8s    remaining: 5m 33s
1000:   total: 1m 50s   remaining: 4m 34s
1500:   total: 2m 43s   remaining: 3m 37s
2000:   total: 3m 36s   remaining: 2m 42s
2500:   total: 4m 30s   remaining: 1m 47s
3000:   total: 5m 23s   remaining: 53.8s
3499:   total: 6m 17s   remaining: 0us
Model Accuracy: 0.8685342651128032
AUC: 0.7579511706368511
```

Submission and conclusion

In [24]:
```
submission = pd.DataFrame({
    "id": testY,
    "Exited": cat.predict_proba(test)[:,1]})
submission.to_csv("s4e01_0131_1.csv", index=False)
```

Submission public score: 0.88951

Conclusion: even the training metrics weren't so hight, I'm satisfied with the public score, which is in 1.5% from the current leader (0.90182), while the model is easy and fast.