Churn Prediction Using Classification

In here, I am going to use two methods, using the actual samples and using resampling method to make each class have equal samples. The expectation when using resampling method is to avoid bias because when testing the models with actual samples, they tend to predict all samples to not churn (because the data contain more `0` class).

In [1]:

```
# importing libraries
# main libraries
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# model building libraries
from sklearn.feature_selection import RFE, RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.pipeline import Pipeline
%matplotlib inline
plt.rcParams['figure.figsize']=(10,6)

from warnings import filterwarnings
filterwarnings('ignore')
```

In [15]:

```
# initialise categorical columns
cat_cols = ['HasCrCard', 'IsActiveMember', 'Complain', 'Card Type', 'Female', 'Male',
       'France','Germany','Spain','zero_balance','high_balance_low_salary','high_salary_l
```

In [2]:

```
data = pd.read_csv('PrepChurnData.csv')
data.head()
```

Out[2]:

	CustomerId	Surname	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estima
0	15634602	Hargrave	619	42	2	0.00	1	1	1	
1	15647311	Hill	608	41	1	83807.86	1	0	1	
2	15619304	Onio	502	42	8	159660.80	3	1	0	
3	15701354	Boni	699	39	1	0.00	2	0	0	
4	15737888	Mitchell	850	43	2	125510.82	1	1	1	

Feature Engineering

In [3]:

```
from sklearn.preprocessing import normalize, LabelEncoder, StandardScaler
```

In [5]:

```
data.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%

	count	mean	std	min	25%	50%	75%	
CustomerId	10000.0	1.569094e+07	71936.186123	15565701.00	15628528.25	1.569074e+07	1.575323e+07	15815
CreditScore	10000.0	6.505288e+02	96.653299	350.00	584.00	6.520000e+02	7.180000e+02	[illegible]
Age	10000.0	3.892180e+01	10.487806	18.00	32.00	3.700000e+01	4.400000e+01	
Tenure	10000.0	5.012800e+00	2.892174	0.00	3.00	5.000000e+00	7.000000e+00	
Balance	10000.0	7.648589e+04	62397.405202	0.00	0.00	9.719854e+04	1.276442e+05	250
NumOfProducts	10000.0	1.530200e+00	0.581654	1.00	1.00	1.000000e+00	2.000000e+00	
HasCrCard	10000.0	7.055000e-01	0.455840	0.00	0.00	1.000000e+00	1.000000e+00	
IsActiveMember	10000.0	5.151000e-01	0.499797	0.00	0.00	1.000000e+00	1.000000e+00	
EstimatedSalary	10000.0	1.000902e+05	57510.492818	11.58	51002.11	1.001939e+05	1.493882e+05	199
Exited	10000.0	2.038000e-01	0.402842	0.00	0.00	0.000000e+00	0.000000e+00	
Complain	10000.0	2.044000e-01	0.403283	0.00	0.00	0.000000e+00	0.000000e+00	
Satisfaction Score	10000.0	3.013800e+00	1.405919	1.00	2.00	3.000000e+00	4.000000e+00	
Point Earned	10000.0	6.065151e+02	225.924839	119.00	410.00	6.050000e+02	8.010000e+02	1
Female	10000.0	4.543000e-01	0.497932	0.00	0.00	0.000000e+00	1.000000e+00	
Male	10000.0	5.457000e-01	0.497932	0.00	0.00	1.000000e+00	1.000000e+00	
France	10000.0	5.014000e-01	0.500023	0.00	0.00	1.000000e+00	1.000000e+00	
Germany	10000.0	2.509000e-01	0.433553	0.00	0.00	0.000000e+00	1.000000e+00	
Spain	10000.0	2.477000e-01	0.431698	0.00	0.00	0.000000e+00	0.000000e+00	

We want to transform feature(s) that have high values and normalise them for our models. This will make our models interpret all features with similar weights.

In [6]:
```
def engineer(df):
    data = df.copy()
    high_values = ['Tenure','CreditScore', 'Balance', 'EstimatedSalary', 'Point Earned']

    data['zero_balance'] = data['Balance']==0
    data['female_male_ratio'] = data.Female.sum() / data.Male.sum()

    for col in high_values:
        data[col] = normalize(np.array(data[col]).reshape(1,-1)).reshape(-1,1)

    data['tenure_balance_ratio'] = np.where(data.zero_balance==False, data.Tenure/data.Ba
    data['salary_balance_ratio'] = np.where(data.zero_balance==False, data.EstimatedSalary
    data['high_balance_low_salary'] = np.where(data.salary_balance_ratio==0, 1, 0)
    data['high_salary_low_balance'] = np.where(data.salary_balance_ratio>=1, 1, 0)
    enc = LabelEncoder()
    data['Card Type'] = enc.fit_transform(data['Card Type'])
#     data['complain_score'] = np.where(data['Complain']==1,
#                                       data.loc[data['Complain']==1,'Satisfaction Score'
#                                       0)
#     data['complain_score'] = np.where(data['complain_score']==0,
#                                       data.loc[data['Complain']==0,'Satisfaction Score'
#                                       data['complain_score'])

    return data
```

In [7]:
```
eng_data = engineer(data)
eng_data.head()
```

Out[7]:

	CustomerId	Surname	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estim
0	15634602	Hargrave	0.009412	42	0.003456	0.000000	1	1	1	
1	15647311	Hill	0.009245	41	0.001728	0.008491	1	0	1	
2	15619304	Onio	0.007633	42	0.013824	0.016175	3	1	0	
3	15701354	Boni	0.010628	39	0.001728	0.000000	2	0	0	
4	15737888	Mitchell	0.012924	43	0.003456	0.012715	1	1	1	

5 rows × 26 columns

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

Out[8]:

	count	mean	std	min	25%	50%	7
CustomerId	10000.0	1.569094e+07	7.193619e+04	1.556570e+07	1.562853e+07	1.569074e+07	1.575323e+
CreditScore	10000.0	9.891430e-03	1.469634e-03	5.321825e-03	8.879845e-03	9.913800e-03	1.091734e-
Age	10000.0	3.892180e+01	1.048781e+01	1.800000e+01	3.200000e+01	3.700000e+01	4.400000e+
Tenure	10000.0	8.661834e-03	4.997513e-03	0.000000e+00	5.183830e-03	8.639716e-03	1.209560e-
Balance	10000.0	7.748754e-03	6.321456e-03	0.000000e+00	0.000000e+00	9.847145e-03	1.293159e-
NumOfProducts	10000.0	1.530200e+00	5.816544e-01	1.000000e+00	1.000000e+00	1.000000e+00	2.000000e+
HasCrCard	10000.0	7.055000e-01	4.558405e-01	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+
IsActiveMember	10000.0	5.151000e-01	4.997969e-01	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+
EstimatedSalary	10000.0	8.670720e-03	4.982078e-03	1.003164e-06	4.418263e-03	8.679701e-03	1.294136e-
Exited	10000.0	2.038000e-01	4.028421e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
Complain	10000.0	2.044000e-01	4.032827e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
Satisfaction Score	10000.0	3.013800e+00	1.405919e+00	1.000000e+00	2.000000e+00	3.000000e+00	4.000000e+
Card Type	10000.0	1.498000e+00	1.118356e+00	0.000000e+00	0.000000e+00	1.000000e+00	2.000000e+
Point Earned	10000.0	9.371039e-03	3.490681e-03	1.838625e-03	6.334757e-03	9.347630e-03	1.237595e-
Female	10000.0	4.543000e-01	4.979320e-01	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+
Male	10000.0	5.457000e-01	4.979320e-01	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+
France	10000.0	5.014000e-01	5.000230e-01	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+
Germany	10000.0	2.509000e-01	4.335527e-01	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+
Spain	10000.0	2.477000e-01	4.316982e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
female_male_ratio	10000.0	8.325087e-01	2.116191e-13	8.325087e-01	8.325087e-01	8.325087e-01	8.325087e-
tenure_balance_ratio	10000.0	4.892430e-01	5.755533e-01	0.000000e+00	0.000000e+00	3.027190e-01	8.620636e-
salary_balance_ratio	10000.0	5.007213e-01	7.014547e-01	0.000000e+00	0.000000e+00	3.186064e-01	8.764975e-
high_balance_low_salary	10000.0	3.617000e-01	4.805166e-01	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+
high_salary_low_balance	10000.0	1.951000e-01	3.962975e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+

Splitting Data

In [9]:
```
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, Strat

train, test = train_test_split(eng_data, test_size=0.2,random_state=42)
```

In [10]:
```
features = eng_data.drop(['CustomerId','Surname','Exited'], axis=1).columns.to_list()
target = 'Exited'
```

In [11]:
```
X_train = train[features].astype('float')
y_train = train[target].astype('float')
X_test = test[features].astype('float')
y_test = test[target].astype('float')
```

In [12]:
```
X_train
```

Out[12]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Complai
9254	0.010431	32.0	0.010368	0.000000	2.0	1.0	1.0	0.015515	0.
1561	0.009610	42.0	0.006912	0.012119	2.0	1.0	1.0	0.016977	0.
1670	0.008500	24.0	0.005184	0.011624	1.0	1.0	0.0	0.007441	1.
6087	0.008530	27.0	0.015551	0.013741	1.0	1.0	0.0	0.013261	1.
6669	0.007861	56.0	0.015551	0.014401	1.0	0.0	0.0	0.003421	1.
...	...	...	...	...	...	...	...	...	.
5734	0.011678	54.0	0.013824	0.007063	1.0	1.0	1.0	0.006010	0.
5191	0.010370	58.0	0.001728	0.000000	1.0	1.0	1.0	0.000061	0.
5390	0.011176	38.0	0.001728	0.000000	3.0	0.0	0.0	0.007989	1.
860	0.010142	43.0	0.013824	0.019272	1.0	1.0	0.0	0.008447	1.
7270	0.010598	51.0	0.001728	0.014985	1.0	1.0	1.0	0.004642	0.

8000 rows × 23 columns

In [187…
```
print('Number of training samples:', X_train.shape[0])
print('Number of test samples:', X_test.shape[0])
```

```
Number of training samples: 8000
Number of test samples: 2000
```

Building Models

In [23]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_a
```

In [24]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class custom_scaler(BaseEstimator, TransformerMixin):
```

```
    def __init__(self, variables=None, excludeComplain=None):
        self.variables = variables
        self.excludeComplain = excludeComplain

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        scaler = StandardScaler()
        num_cols = [col for col in X.columns if col not in self.variables]
        if (self.excludeComplain) & ('Complain' in self.variables):
            self.variables.remove('Complain')
        X_scaled = pd.DataFrame(scaler.fit_transform(X[num_cols]), index=X.index,columns=n
        X = pd.concat([X_scaled, X[self.variables]], axis=1)
        return X
```

In [25]:

```
def plot_cm(y_hat):
    matrix = confusion_matrix(y_test,y_hat)
    ax = sns.heatmap(matrix, annot=True, fmt='d', cmap='crest')
    ax.set(title='Confusion Matrix', xlabel='Predicted values', ylabel='Actual Values',
           xticklabels=['Did not churn', 'Churn'], yticklabels=['Did not churn', 'Churn'])
    plt.show()
```

In [26]:

```
def build_model(model_name, parameters, excludeComplain=False):
    cat_cols = ['HasCrCard', 'IsActiveMember', 'Complain', 'Card Type', 'Female', 'Male',
            'France','Germany','Spain','zero_balance','high_balance_low_salary','high_sala
    scale_num_cols = custom_scaler(variables=cat_cols, excludeComplain=excludeComplain)
    pipe = Pipeline([('scaler', scale_num_cols),('model',model_name)])
    pipe_cv = GridSearchCV(pipe, param_grid=parameters, refit=True, verbose=1, scoring='ro
    pipe_cv.fit(X_train,y_train)
    return pipe_cv
```

In [27]:

```
def evaluate(model_name, excludeComplain=False):
    features = X_test.columns.to_list()
    if excludeComplain:
        features.remove('Complain')

    print('The best parameters are:', model_name.best_params_)
    yhat_train = model_name.predict(X_train[features])
    yhat = model_name.predict(X_test[features])
    train_acc = accuracy_score(yhat_train, y_train)
    test_acc = accuracy_score(yhat, y_test)
    pos_probs = model_name.predict_proba(X_test[features])[:,1]
    roc_score = roc_auc_score(y_test, pos_probs)
    print('The best score on training set is:', train_acc)
    print('Accuracy on test set: ', test_acc)
    print('ROC score: ', roc_score)
    print('The classification scores are:')
    print(classification_report(yhat, y_test))
    scores['train_acc'].append(train_acc)
    scores['test_acc'].append(test_acc)
    scores['roc_auc'].append(roc_score)
    plot_cm(yhat)
```

Experiment with complain vs without complain as a feature

In [28]:

```
# initialise score metrics
```

```
scores = {'test_acc':[], 'train_acc':[], 'roc_auc':[]}
```

Logistic Regression

With Complain

In [370…

```
parameters = {'model__C':[0.01,0.1,1],
              'model__penalty':['l2'],
              'model__solver':['lbfgs','newton-cholesky']}
LR_model = build_model(LogisticRegression(random_state=42), parameters, excludeComplain=Fa
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

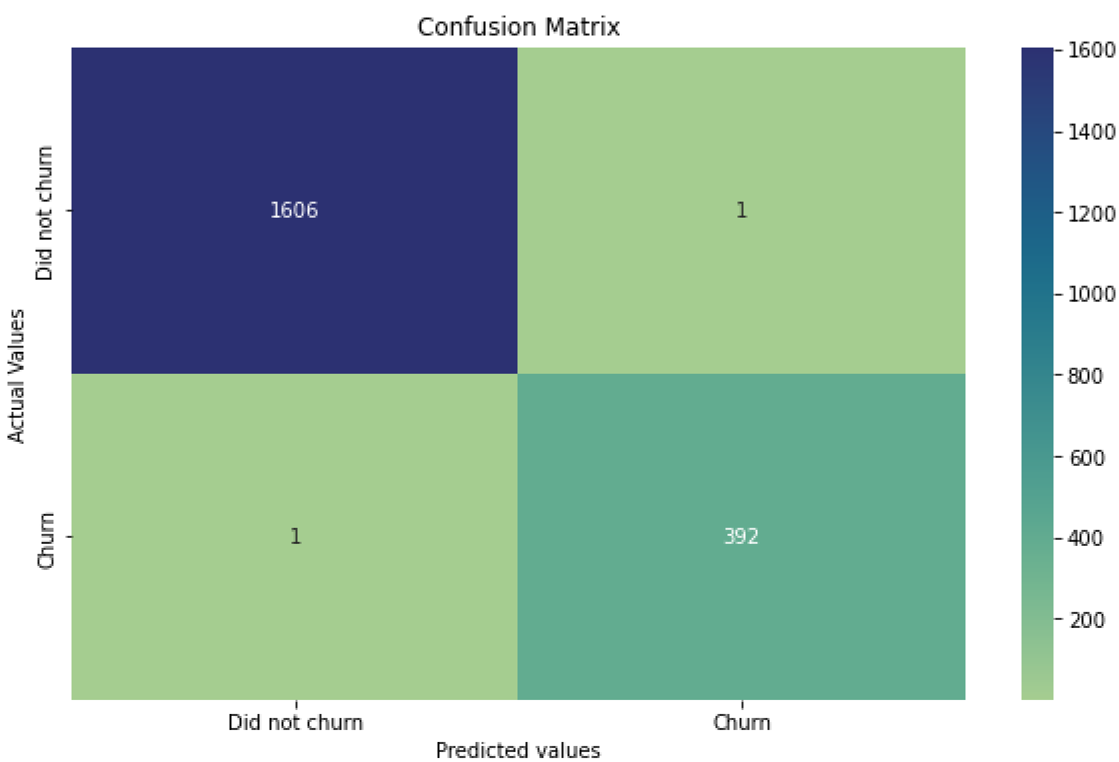
In [373…

```
evaluate(LR_model)
```

```
The best parameters are: {'model__C': 0.01, 'model__penalty': 'l2', 'model__solver': 'lbfg
s'}
The best score on training set is: 0.99775
Accuracy on test set:  0.999
ROC score:  0.9993017191010702
The classification scores are:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1607
         1.0       1.00      1.00      1.00       393

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```



Without Complain

In [375…

```
parameters = {'model__C':[0.01,0.1,1],
              'model__penalty':['l2'],
```

```
              'model__solver':['lbfgs','newton-cholesky']}
LR_model2 = build_model(LogisticRegression(random_state=42), parameters, excludeComplain=T
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

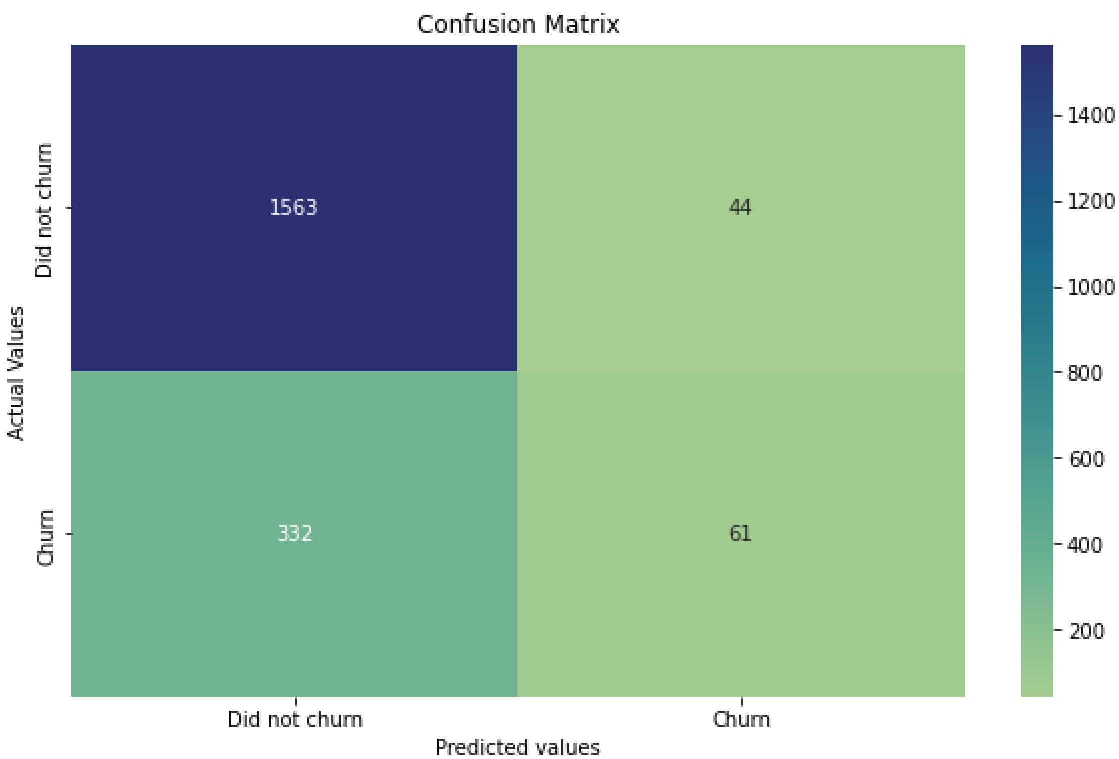
In [376…

```
evaluate(LR_model2, True)
```

```
The best parameters are: {'model__C': 0.01, 'model__penalty': 'l2', 'model__solver': 'lbfg
s'}
The best score on training set is: 0.806
Accuracy on test set:  0.812
ROC score:  0.7810928966940122
The classification scores are:
              precision    recall  f1-score   support

         0.0       0.97      0.82      0.89      1895
         1.0       0.16      0.58      0.24       105

    accuracy                           0.81      2000
   macro avg       0.56      0.70      0.57      2000
weighted avg       0.93      0.81      0.86      2000
```



We can see that the model performance dropped significantly when Complain is not included as a one of the features. We know that Complain is one of the key drivers for churning customers. Thus, now we want to find out whether there is other customer segments that drives churning customers. Now we are going to exclude Complain as a feature.

Logistic Regression is our base model. It is a good indicator whether other classifiers can improve from Logistic Regression and make comparison.

Decision Tree Classification

In [377…

```
parameters = {'model__min_samples_split':[2,8],
              'model__max_depth':[2,4,8],
              'model__max_features':[5,10],
```

```
                'model__class_weight':[{0:1,1:2}, {0:1,1:1.5}]}
tree_model = build_model(DecisionTreeClassifier(random_state=42), parameters, excludeComp
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

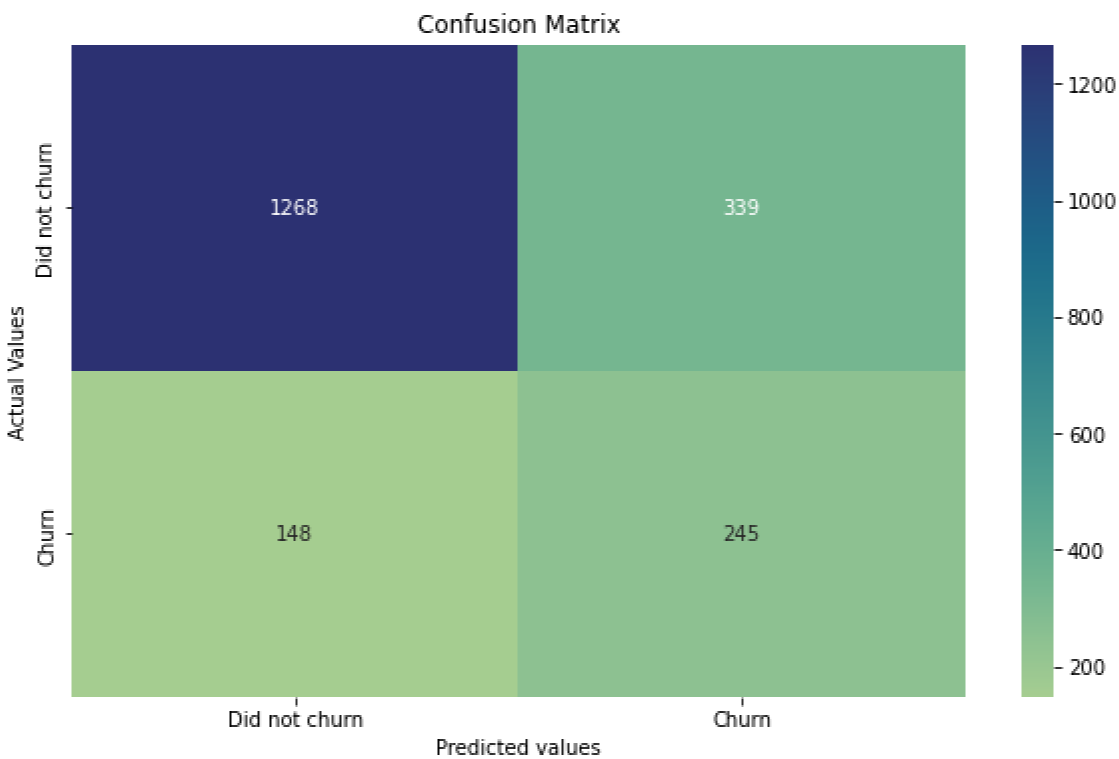
In [378…

```
evaluate(tree_model, True)
```

```
The best parameters are: {'model__class_weight': {0: 1, 1: 2}, 'model__max_depth': 2, 'mod
el__max_features': 5, 'model__min_samples_split': 2}
The best score on training set is: 0.745625
Accuracy on test set:  0.7565
ROC score:  0.732778508782347
The classification scores are:
              precision    recall  f1-score   support

         0.0       0.79      0.90      0.84      1416
         1.0       0.62      0.42      0.50       584

    accuracy                           0.76      2000
   macro avg       0.71      0.66      0.67      2000
weighted avg       0.74      0.76      0.74      2000
```



Random Forest Classification

Ensemble models can boost performance by combining multiple algorithms to learn the data. Random Forest is one of example of Ensemble models. It uses bagging technique which uses multiple decision trees.

In [383…

```
parameters={'model__n_estimators':[100,200],
            'model__class_weight':[{0:1,1:2}, {0:1,1:1.5}],
            'model__max_depth':[2,4,8],
            'model__max_features':[5,15]}
rf_model = build_model(RandomForestClassifier(random_state=42), parameters, excludeCompla
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

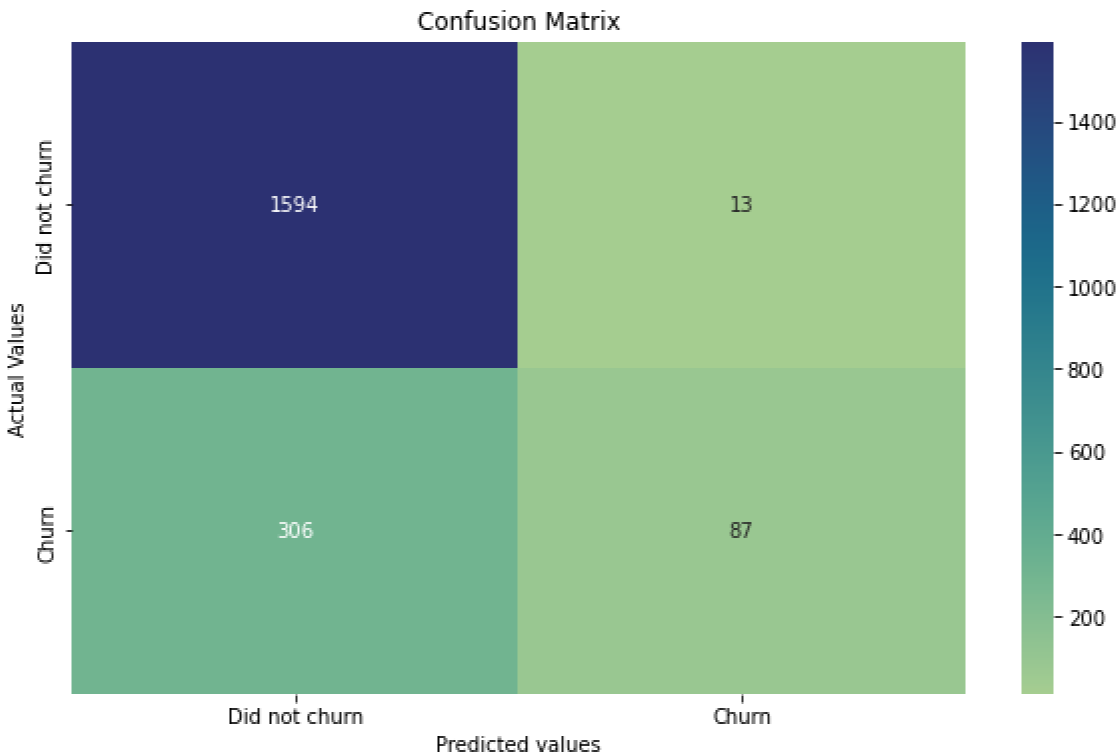
In [384…

```
evaluate(rf_model, True)
```

```
The best parameters are: {'model__class_weight': {0: 1, 1: 2}, 'model__max_depth': 2, 'mod
el__max_features': 5, 'model__n_estimators': 100}
The best score on training set is: 0.8355
Accuracy on test set:  0.8405
ROC score:  0.8249476289325803
The classification scores are:
              precision    recall  f1-score   support

         0.0       0.99      0.84      0.91      1900
         1.0       0.22      0.87      0.35       100

    accuracy                           0.84      2000
   macro avg       0.61      0.85      0.63      2000
weighted avg       0.95      0.84      0.88      2000
```



Gradient Boosting

Gradient boosting is another ensemble model that utilises boosting technique.

In [388…

```
parameters = {'model__n_estimators':[100,150],
              'model__learning_rate':[0.1,0.2],
              'model__max_features':[5,10],
              'model__subsample':[1,0.9]}
gb_model = build_model(GradientBoostingClassifier(random_state=42), parameters, excludeCor
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

In [389…

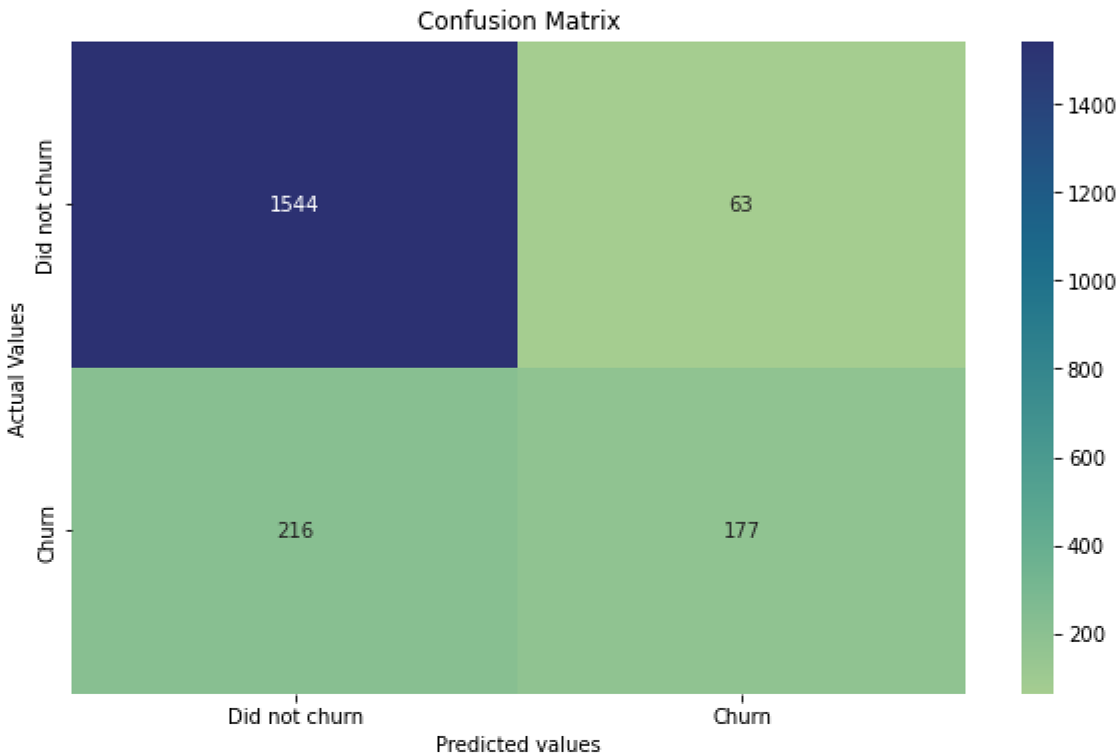
```
evaluate(gb_model, True)
```

```
The best parameters are: {'model__learning_rate': 0.1, 'model__max_features': 5, 'model__n
_estimators': 100, 'model__subsample': 1}
The best score on training set is: 0.87025
Accuracy on test set:  0.8605
ROC score:  0.8683819675687316
The classification scores are:
              precision    recall  f1-score   support

         0.0       0.96      0.88      0.92      1760
```

```
         1.0       0.45      0.74      0.56       240

    accuracy                           0.86      2000
   macro avg       0.71      0.81      0.74      2000
weighted avg       0.90      0.86      0.87      2000
```



Model Performance Comparison

In [393…

```
scores_df = pd.DataFrame(scores, index=['Logistic Regression', 'Decision Tree', 'Random Fo
scores_df
```

Out[393…

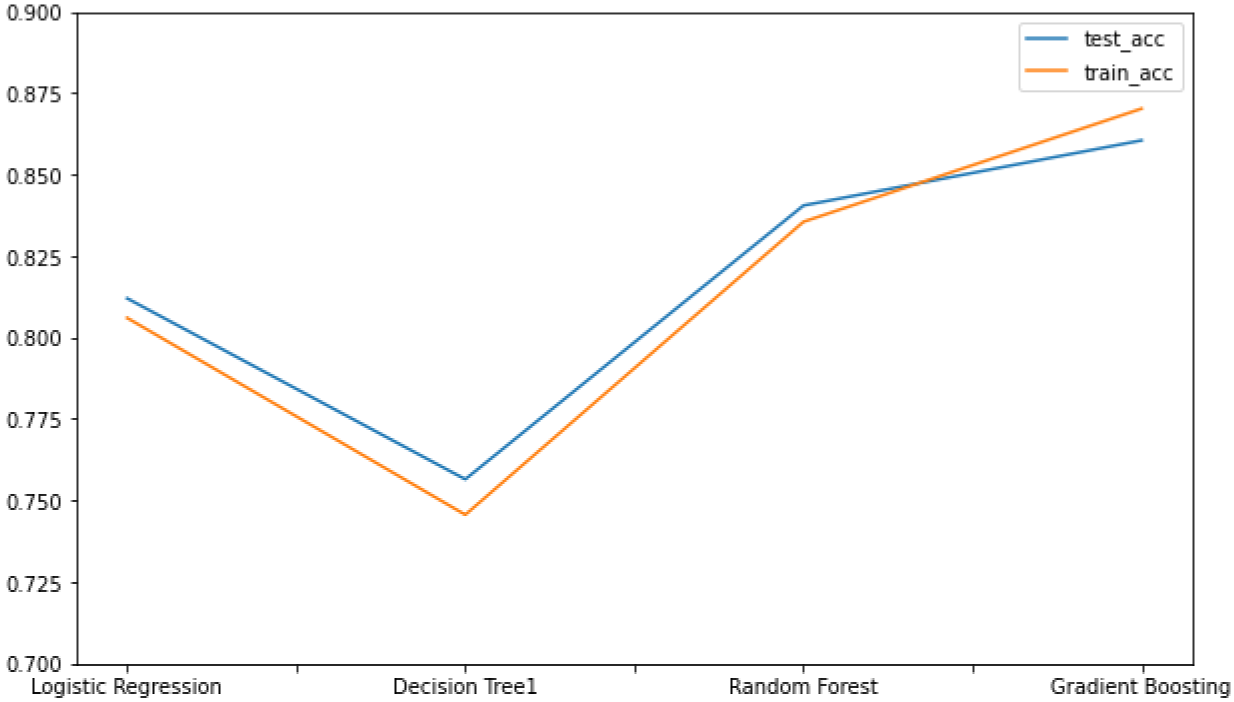
	test_acc	train_acc	roc_auc
Logistic Regression	0.8120	0.806000	0.781093
Decision Tree1	0.7565	0.745625	0.732779
Random Forest	0.8405	0.835500	0.824948
Gradient Boosting	0.8605	0.870250	0.868382

In [394…

```
ax = scores_df.drop('roc_auc',axis=1).plot(kind='line', rot=0, figsize=(10,6))
ax.set(ylim=[0.7,0.9])
```

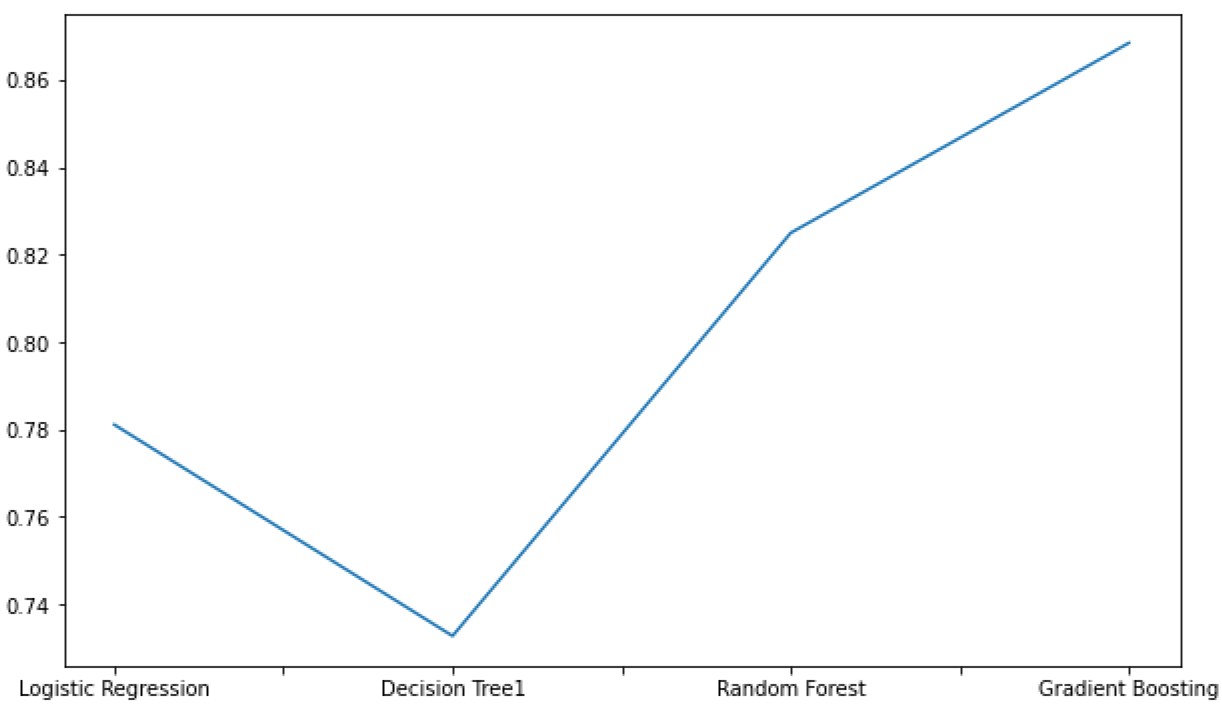
Out[394…

```
[(0.7, 0.9)]
```



In [395…

```
ax = scores_df['roc_auc'].plot(kind='line', rot=0, figsize=(10,6))
```



There is a small improvement in validation score from logistic regression model. Ensemble methods are great here. Random Forest and Gradient Boosting models have significant increase in performance. The ROC score also increases from logistic regression and decision tree.

In conclusion, one of the main driver of churning customer is complain. We know this finding based on our exploration and further proved it with building ML models. When building models without complain as a feature, performance dropped significantly. So, I decided to investigate whether there are other features that interact with churn well.

Several models were built including Logistic Regression (our base model), decision tree, random forest, and gradient boosting. Overall, gradient boosting model is chosen here as the best model for predicting churn.

While random forest has slightly higher test accuracy, gradient boosting has better ROC score. ROC score is a measure to evaluate whether model can distinguish binary classes.

The main features that drive customer churn will be discovered in the next phase using recursive feature elimination (RFE) and cross validation (CV). I will also experiment using resample method to find out if it will improve the f1-score for `1` class (churn).

Current model performance benchmark (GB):

```
The best score on training set is: 0.87025
Accuracy on test set:  0.8605
ROC score:  0.8683819675687316
The classification scores are:
              precision    recall  f1-score   support

         0.0       0.96      0.88      0.92      1760
         1.0       0.45      0.74      0.56       240

    accuracy                           0.86      2000
   macro avg       0.71      0.81      0.74      2000
weighted avg       0.90      0.86      0.87      2000
```

In [396…

```
import pickle
pickle.dump(gb_model, open('best_model.pkl', 'wb'))
```

RFECV

Best model params:

The best parameters are: {'model**learning_rate': 0.1, 'model**max_features': 5, 'model**n_estimators': 100, 'model**subsample': 1}

In [403…

```
# Use all data
X = eng_data[features]
y = eng_data.Exited
# Scale data
cat_cols = ['HasCrCard', 'IsActiveMember', 'Complain', 'Card Type', 'Female', 'Male',
            'France','Germany','Spain','zero_balance','high_balance_low_salary','high_sala
scale_num_cols = custom_scaler(variables=cat_cols, excludeComplain=True)
X_scaled = scale_num_cols.fit_transform(X)
# Build rfe
estimator = GradientBoostingClassifier(random_state=42, learning_rate= 0.1,
                                       max_features=5, n_estimators=100, subsample=1)
rfecv = RFECV(estimator=estimator, cv=StratifiedKFold(n_splits=5), scoring='roc_auc')
rfecv.fit(X_scaled, y)
```

Out[403…

```
RFECV
estimator: GradientBoostingClassifier
GradientBoostingClassifier
```

In [406…

```
selected_features = X_scaled.columns[rfecv.support_]
```

```
selected_features
```

Out[406…

```
Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
       'EstimatedSalary', 'Satisfaction Score', 'Point Earned',
       'tenure_balance_ratio', 'salary_balance_ratio', 'IsActiveMember',
       'Female', 'Male', 'France', 'Germany', 'Spain', 'zero_balance',
       'high_balance_low_salary'],
      dtype='object')
```

In [409…

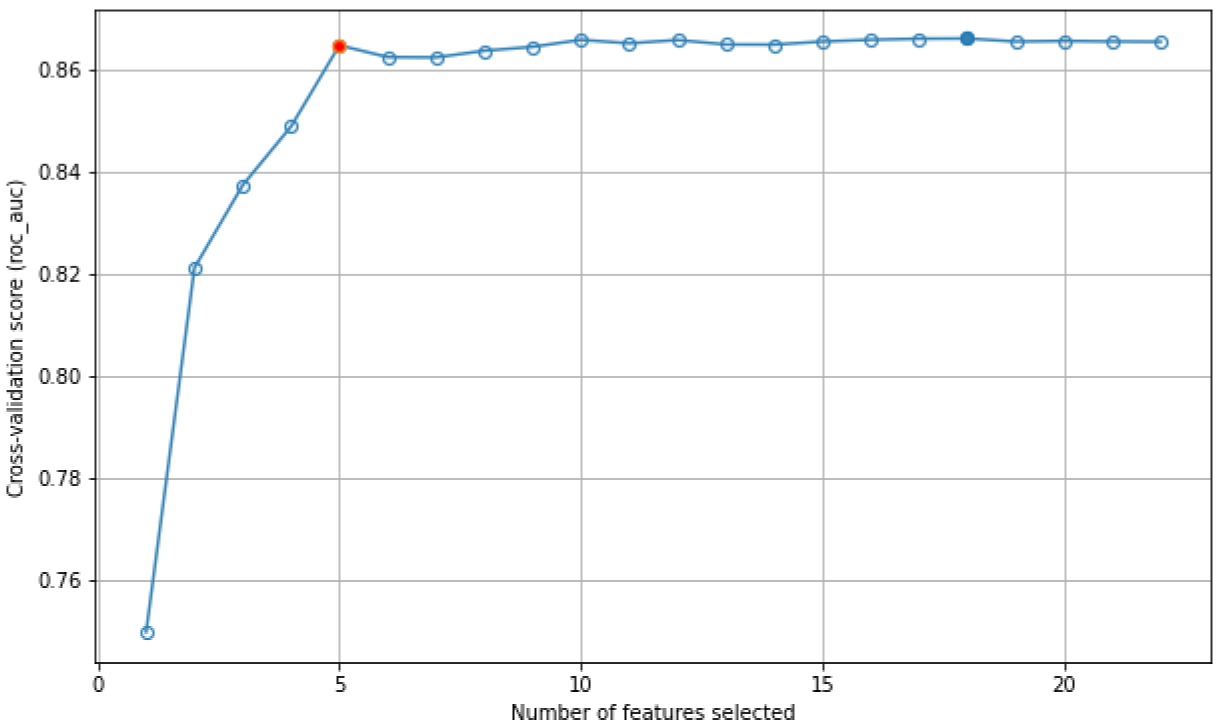
```
np.argmax(rfecv.cv_results_['mean_test_score'])
```

Out[409…

```
17
```

In [416…

```
plt.figure()
plt.grid(True)
plt.xlabel("Number of features selected")
plt.ylabel("Cross-validation score (roc_auc)")
plt.plot(range(1, len(rfecv.cv_results_['mean_test_score']) + 1), rfecv.cv_results_['mean_
        marker='o', color='#3282b8', markerfacecolor = 'None', markeredgecolor = '#3282b8'
plt.plot(18, rfecv.cv_results_['mean_test_score'][17], marker='o', markerfacecolor = '#328
plt.plot(5, rfecv.cv_results_['mean_test_score'][4], marker='o', markerfacecolor = 'r', ma
plt.show()
```



The roc_auc score peaks with 18 features selected. The score stops increasing after 5 features and this is the optimal number of features.

Features importance

In [455…

```
f1 = features.copy()
f1.remove('Complain')
X_train_sc = scale_num_cols.fit_transform(X_train[f1])
estimator.fit(X_train_sc, y_train)
```

In [456…

```
X_train_sc
```

Out[456…

	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	Satisfaction Score	Point Earned	female_
9254	0.356500	-0.655786	0.345680	-1.218471	0.808436	1.367670	-0.720010	-0.430193	-3.
1561	-0.203898	0.294938	-0.348369	0.696838	0.808436	1.661254	0.704342	1.565908	-3.
1670	-0.961472	-1.416365	-0.695393	0.618629	-0.916688	-0.252807	0.704342	-1.243749	-3.
6087	-0.940717	-1.131148	1.386753	0.953212	-0.916688	0.915393	-0.720010	-0.176791	-3.
6669	-1.397337	1.625953	1.386753	1.057449	-0.916688	-1.059600	-0.007834	0.534515	-3.
...	...	...	...	...	...	...	...	...	
5734	1.207474	1.435808	1.039728	-0.102301	-0.916688	-0.539860	-0.007834	-0.167899	-3.
5191	0.314989	1.816097	-1.389442	-1.218471	-0.916688	-1.733882	0.704342	0.454493	-3.
5390	0.865009	-0.085351	-1.389442	-1.218471	2.533560	-0.142765	1.416518	0.316678	-3.
860	0.159323	0.390011	1.039728	1.827259	-0.916688	-0.050826	-0.007834	0.325569	-3.
7270	0.470655	1.150590	-1.389442	1.149720	-0.916688	-0.814568	0.704342	0.143297	-3.

8000 rows × 22 columns

In [431…

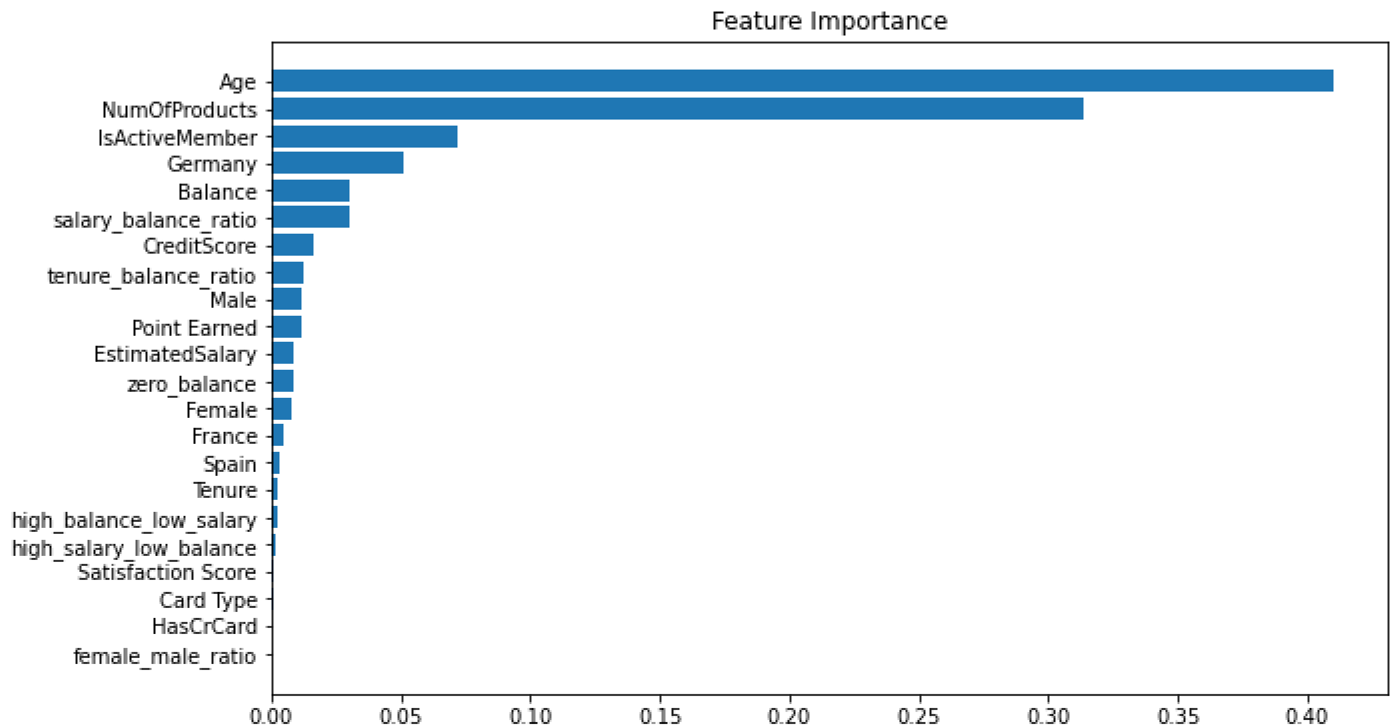
```
sort_indices = estimator.feature_importances_.argsort()
X_train_sc.columns[sort_indices]
```

Out[431…

```
Index(['female_male_ratio', 'HasCrCard', 'Card Type', 'Satisfaction Score',
       'high_salary_low_balance', 'high_balance_low_salary', 'Tenure', 'Spain',
       'France', 'Female', 'zero_balance', 'EstimatedSalary', 'Point Earned',
       'Male', 'tenure_balance_ratio', 'CreditScore', 'salary_balance_ratio',
       'Balance', 'Germany', 'IsActiveMember', 'NumOfProducts', 'Age'],
      dtype='object')
```

In [432…

```
plt.figure()
plt.title("Feature Importance")
plt.barh(range(X_train_sc.shape[1]), estimator.feature_importances_[sort_indices])
plt.yticks(range(X_train_sc.shape[1]), X_train_sc.columns[sort_indices], rotation=0)
plt.show()
```



From the chart it can be seen that the top 3 features are age, number of products with the bank, and active member status. However, age and number of products are the most influential features in determining churn due to its contribution value.

Overall, the bank institution should focus on building better experience for their customer. This may be solved by resolving how the bank handles complains received as complain is the main driver. Furthermore, age is also important in determining churn, the further exploration of which age group churned in the past will be conducted. Lastly, number of products that the customer has with the bank is also crucial. Increasing bank products as well as its quality will prevent customer from churning.

Exploring Age group

In [437…

```
data['Age'].describe()
```

Out[437…

```
count    10000.000000
mean        38.921800
std         10.487806
min         18.000000
25%         32.000000
50%         37.000000
75%         44.000000
max         92.000000
Name: Age, dtype: float64
```

In [436…

```
group = [18,20,30,40,50,60,70,93]
labels = ['<20', '20s', '30s','40s','50s','60s','>70']
data['Age_group']=pd.cut(data['Age'], bins=group, labels=labels, right=False)
```

In [438…

```
data.head(10)
```

Out[438…

	CustomerId	Surname	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estima
0	15634602	Hargrave	619	42	2	0.00	1	1	1	

	CustomerId	Surname	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estima
1	15647311	Hill	608	41	1	83807.86	1	0	1	
2	15619304	Onio	502	42	8	159660.80	3	1	0	
3	15701354	Boni	699	39	1	0.00	2	0	0	
4	15737888	Mitchell	850	43	2	125510.82	1	1	1	
5	15574012	Chu	645	44	8	113755.78	2	1	0	
6	15592531	Bartlett	822	50	7	0.00	2	1	1	
7	15656148	Obinna	376	29	4	115046.74	4	1	0	
8	15792365	He	501	44	4	142051.07	2	0	1	
9	15592389	H?	684	27	2	134603.88	1	1	1	

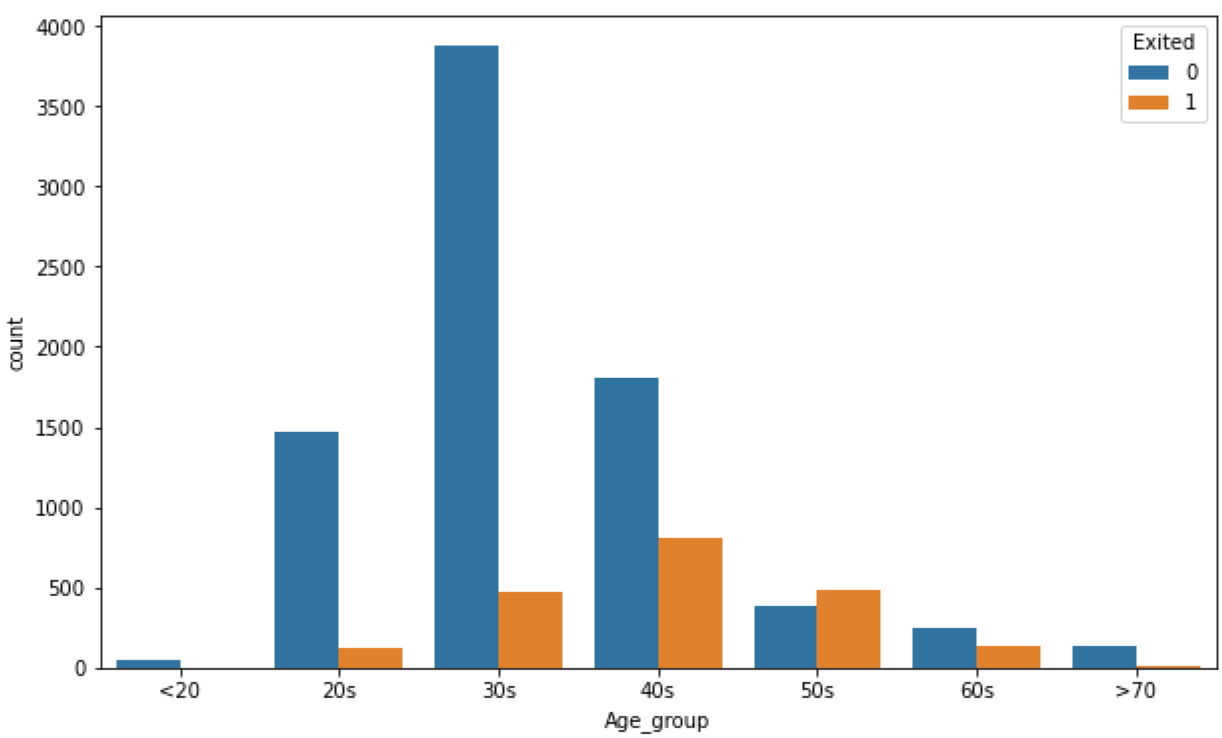
10 rows × 21 columns

In [439…

```
sns.countplot(x=data['Age_group'], hue=data['Exited'])
```

Out[439…

```
<AxesSubplot:xlabel='Age_group', ylabel='count'>
```



Most of the customers who churn are located in the 40s age group.

Resampling Method (Equal samples for each class target)

Using resample technique

In [506…

```
from sklearn.utils import resample

# combined features and target data
train_data = pd.concat([X_train, y_train], axis=1)
major_class = train_data.loc[train_data['Exited']==0]
minor_class = train_data.loc[train_data['Exited']==1]
```

```
# resample until minor class reaches half of major class
minor_upsampled = resample(minor_class, replace=True, n_samples=int(len(major_class)/2),
                           random_state=42)
upsampled_df = pd.concat([major_class, minor_upsampled])
# define feature and target
X_train = upsampled_df.drop('Exited', axis=1)
y_train = upsampled_df['Exited']
```

In [507…

```
y_train_re.value_counts()
```

Out[507…

```
0.0    6355
1.0    3177
Name: Exited, dtype: int64
```

In [508…

```
X_train
```

Out[508…

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Complai
9254	0.010431	32.0	0.010368	0.000000	2.0	1.0	1.0	0.015515	0.
1561	0.009610	42.0	0.006912	0.012119	2.0	1.0	1.0	0.016977	0.
8829	0.010385	40.0	0.001728	0.000000	2.0	0.0	0.0	0.006563	0.
7945	0.011541	45.0	0.013824	0.000000	2.0	1.0	1.0	0.008598	0.
3508	0.006477	34.0	0.005184	0.000000	2.0	1.0	1.0	0.005304	0.
...	...	...	...	...	...	...	...	...	.
6564	0.009914	47.0	0.000000	0.012826	2.0	1.0	1.0	0.003361	1.
1685	0.009321	20.0	0.000000	0.011889	1.0	0.0	0.0	0.009837	1.
916	0.007222	39.0	0.010368	0.000000	1.0	1.0	1.0	0.004938	1.
43	0.012681	49.0	0.003456	0.013312	1.0	0.0	0.0	0.016838	1.
1003	0.009823	42.0	0.005184	0.017745	2.0	0.0	0.0	0.005815	1.

9532 rows × 23 columns

Training Model

In [516…

```
parameters = {'model__n_estimators':[100,150],
              'model__learning_rate':[0.1,0.2],
              'model__max_features':[5,10],
              'model__subsample':[1,0.9]}
gb_model = build_model(GradientBoostingClassifier(random_state=42), parameters, excludeCor
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

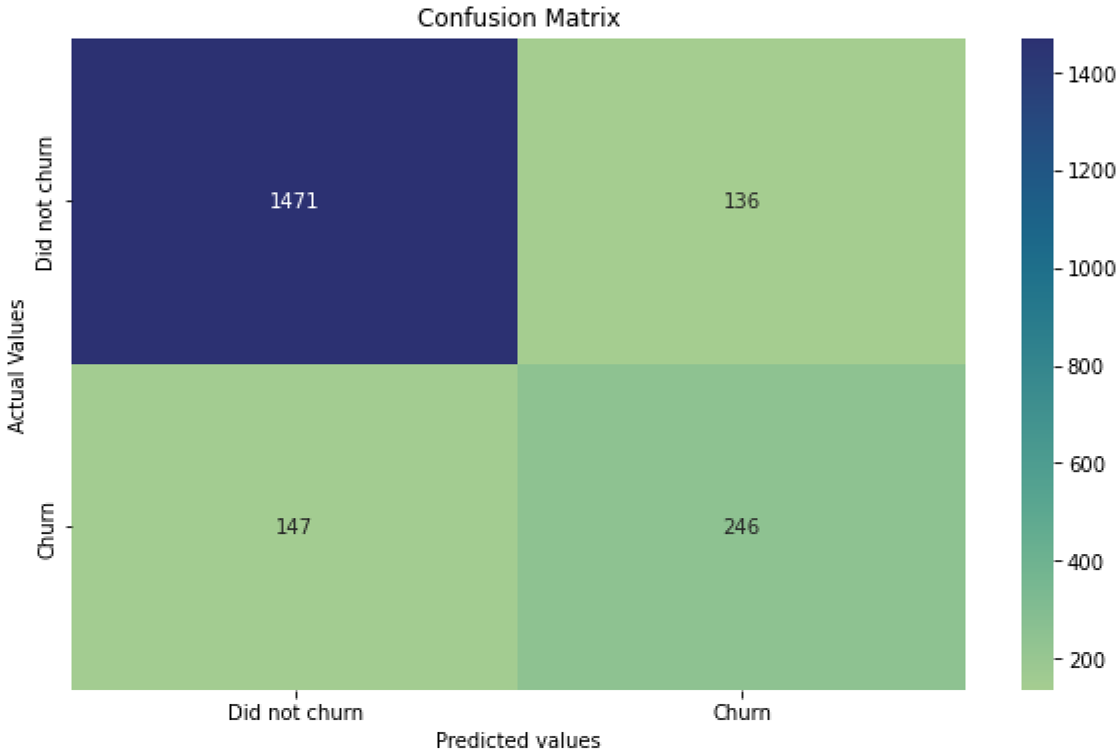
In [517…

```
evaluate(gb_model, True)
```

```
The best parameters are: {'model__learning_rate': 0.1, 'model__max_features': 5, 'model__n
_estimators': 100, 'model__subsample': 1}
The best score on training set is: 0.8268988669744021
Accuracy on test set:  0.8585
ROC score:  0.8697690289461975
The classification scores are:
              precision    recall  f1-score   support
```

```
         0.0       0.92      0.91      0.91      1618
         1.0       0.63      0.64      0.63       382

    accuracy                           0.86      2000
   macro avg       0.77      0.78      0.77      2000
weighted avg       0.86      0.86      0.86      2000
```



Comparison to without resampling model:

```
The best score on training set is: 0.87025
Accuracy on test set:  0.8605
ROC score:  0.8683819675687316
The classification scores are:
              precision    recall  f1-score   support

         0.0       0.96      0.88      0.92      1760
         1.0       0.45      0.74      0.56       240

    accuracy                           0.86      2000
   macro avg       0.71      0.81      0.74      2000
weighted avg       0.90      0.86      0.87      2000
```

The ROC score has increased slighly. However, we see a slight drop in accuracy in test set. The major improvement seen is the f1-score on true class from 0.56 -> 0.63 (~11% increase).

Using SMOTE technique

In [13]:

```
# Re-initialise X and y train
X_train = train[features].astype('float')
y_train = train[target].astype('float')
```

In [18]:

```python
from imblearn.over_sampling import SMOTENC, SMOTE
cat_inds = [X_train.columns.get_loc(c) for c in cat_cols]
smote = SMOTENC(sampling_strategy=.5, categorical_features=cat_inds)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
print(y_resampled.value_counts())
X_resampled
```

```
0.0    6355
1.0    3177
Name: Exited, dtype: int64
```

Out[18]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Complai
0	0.010431	32.0	0.010368	0.000000	2.0	1.0	1.0	0.015515	0.
1	0.009610	42.0	0.006912	0.012119	2.0	1.0	1.0	0.016977	0.
2	0.008500	24.0	0.005184	0.011624	1.0	1.0	0.0	0.007441	1.
3	0.008530	27.0	0.015551	0.013741	1.0	1.0	0.0	0.013261	1.
4	0.007861	56.0	0.015551	0.014401	1.0	0.0	0.0	0.003421	1.
...	...	...	...	...	...	...	...	...	.
9527	0.010360	51.0	0.004039	0.009829	1.0	1.0	0.0	0.008981	1.
9528	0.010324	51.0	0.007355	0.010819	1.0	0.0	0.0	0.014652	1.
9529	0.009199	40.0	0.003297	0.011715	1.0	0.0	0.0	0.012154	1.
9530	0.011276	49.0	0.013993	0.005109	1.0	1.0	0.0	0.005498	1.
9531	0.008915	40.0	0.004871	0.007028	1.0	0.0	1.0	0.002329	1.

9532 rows × 23 columns

In [19]:

```python
X_train = X_resampled
y_train = y_resampled
```

In [29]:

```python
parameters = {'model__n_estimators':[100,150],
              'model__learning_rate':[0.1,0.2],
              'model__max_features':[5,10],
              'model__subsample':[1,0.9]}
gb_model = build_model(GradientBoostingClassifier(random_state=42), parameters, excludeCor
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

In [30]:

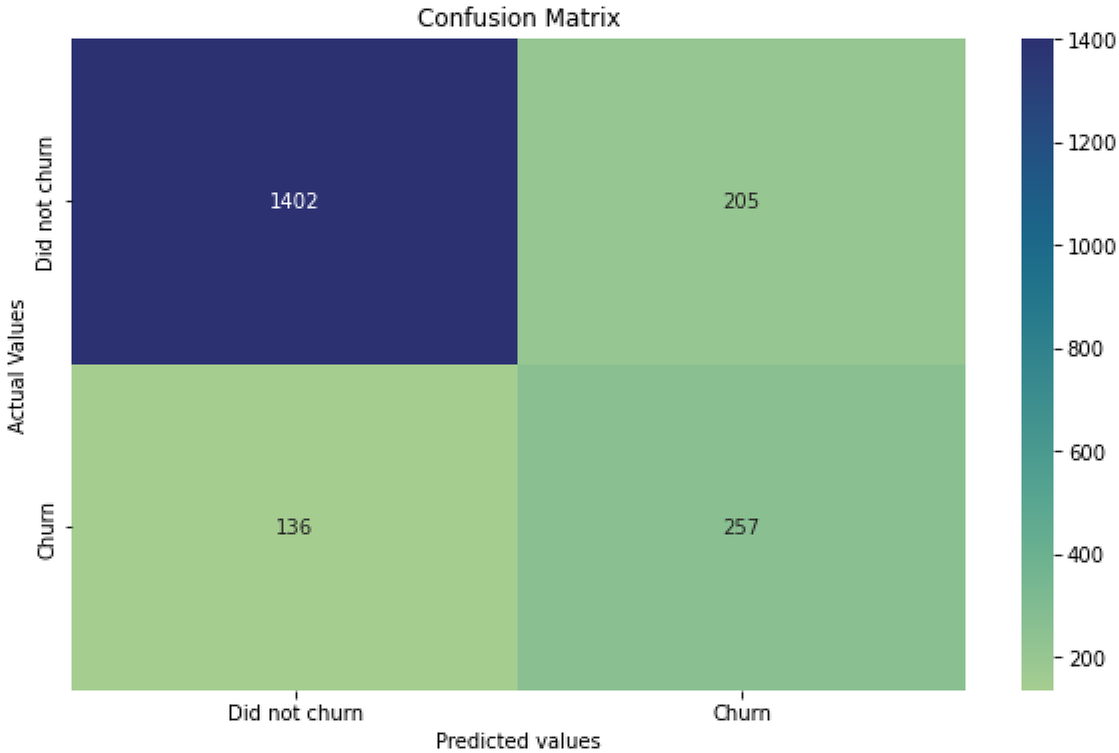
```python
evaluate(gb_model, True)
```

```
The best parameters are: {'model__learning_rate': 0.1, 'model__max_features': 5, 'model__n
_estimators': 100, 'model__subsample': 1}
The best score on training set is: 0.8522870331514897
Accuracy on test set:  0.8295
ROC score:  0.8503881713432487
The classification scores are:
              precision    recall  f1-score   support

         0.0       0.87      0.91      0.89      1538
         1.0       0.65      0.56      0.60       462

    accuracy                           0.83      2000
   macro avg       0.76      0.73      0.75      2000
```

```
weighted avg       0.82      0.83      0.82      2000
```



Compared to using resampling technique, when using SMOTE the model evaluation metrics are slighlty lower.

Overall, when using resampling technique, model performance slightly improves. However the improvement is not significant, so it might be not necessary to resample the data to cover the imbalanceness. For further research, I would suggest getting more data for analysis.

In [ ]: