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).

Building Models

In [34]:

```
# 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.model_selection import train_test_split, cross_val_score, GridSearchCV, Strat
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, mean_
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 make_pipeline, Pipeline
from sklearn.preprocessing import StandardScaler
%matplotlib inline
plt.rcParams['figure.figsize']=(10,6)
```

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	

In [3]:

```
features = ['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMe
            'Satisfaction Score','Point Earned','Female','Male','France','Germany','Spain'
target = 'Exited'
```

In [4]:

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

In [5]:

```
X_train = train[features]
y_train = train[target].astype(str)
X_test = test[features]
y_test = test[target].astype(str)
```

In [6]:

```
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
```

In [7]:

```
scores = {'test_score':[], 'val_score':[], 'rmse':[]}
```

In [8]:

```
def plot_cm(model, model_name):
    yhat = model.predict(X_test)
    matrix = confusion_matrix(yhat, y_test)
    ax = sns.heatmap(matrix, annot=True, fmt='d', cmap='crest')
    ax.set(title='Confusion Matrix for '+model_name, xlabel='Predicted values', ylabel='Ac
           xticklabels=['Did not churn', 'Churn'], yticklabels=['Did not churn', 'Churn'])
    plt.show()
```

In [9]:

```
def build_model(model_name, parameters):
    pipe = Pipeline([('scaler', StandardScaler()),('model',model_name)])
    pipe_cv = GridSearchCV(pipe, param_grid=parameters, refit=True, verbose=1)
    pipe_cv.fit(X_train,y_train)
    return pipe_cv
```

In [10]:

```
def evaluate(model_name):
    print('The best parameters are:', model_name.best_params_)
    print('The best score on training set is:', model_name.best_score_)
    yhat = model_name.predict(X_test)
    test_acc = accuracy_score(yhat, y_test)
    print('Accuracy on test set: ', test_acc)
    print('The classification scores are:')
    print(classification_report(yhat, y_test))
    rmse = np.sqrt(mean_squared_error(yhat, y_test))
    scores['val_score'].append(model_name.best_score_)
    scores['test_score'].append(test_acc)
    scores['rmse'].append(rmse)
```

Logistic Regression

In [11]:

```
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)
```

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

In [12]:

```
evaluate(LR_model)
```

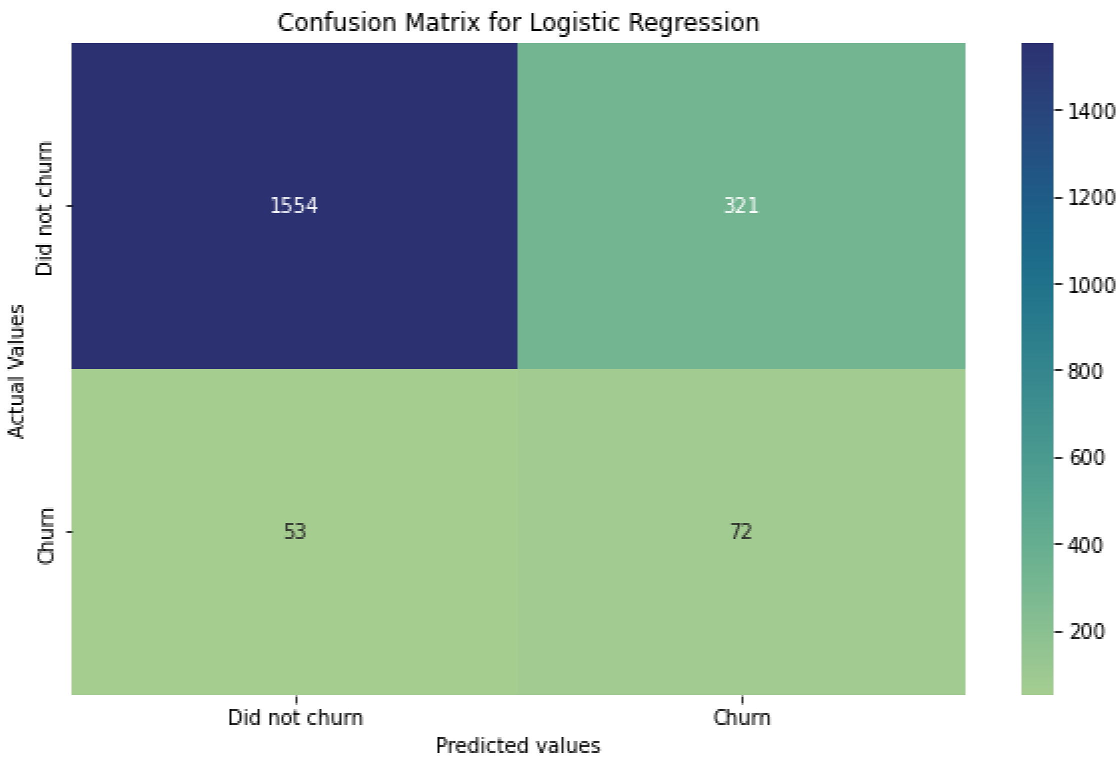
```
The best parameters are: {'model__C': 0.01, 'model__penalty': 'l2', 'model__solver': 'lbfg
s'}
The best score on training set is: 0.8105
Accuracy on test set:  0.813
The classification scores are:
              precision    recall  f1-score   support

           0       0.97      0.83      0.89      1875
           1       0.18      0.58      0.28       125
```

```
    accuracy                           0.81      2000
   macro avg       0.58      0.70      0.59      2000
weighted avg       0.92      0.81      0.85      2000
```

In [13]:

```python
plot_cm(LR_model, 'Logistic Regression')
```



Decision Tree Classification

In [14]:

```python
parameters = {'model__criterion':['gini','entropy'],
              'model__max_depth':[2,4,8],
              'model__max_features':[5,10,None,'sqrt']}
tree_model = build_model(DecisionTreeClassifier(random_state=42), parameters)
```

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

In [15]:

```python
evaluate(tree_model)
```

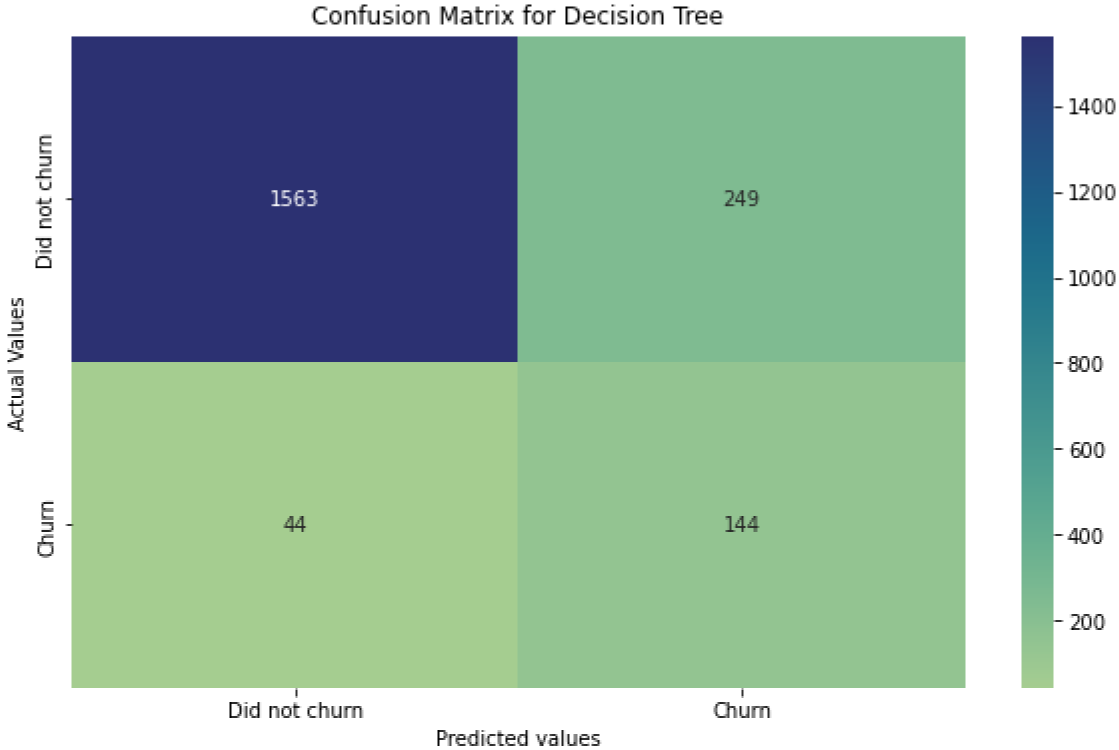
```
The best parameters are: {'model__criterion': 'gini', 'model__max_depth': 4, 'model__max_f
eatures': None}
The best score on training set is: 0.8491250000000001
Accuracy on test set:  0.8535
The classification scores are:
              precision    recall  f1-score   support

           0       0.97      0.86      0.91      1812
           1       0.37      0.77      0.50       188

    accuracy                           0.85      2000
   macro avg       0.67      0.81      0.70      2000
weighted avg       0.92      0.85      0.87      2000
```

In [16]:

```python
plot_cm(tree_model, 'Decision Tree')
```



Random Forest Classification

In [17]:

```
parameters={'model__n_estimators':[100,150],
            'model__criterion':['gini','entropy'],
            'model__max_depth':[2,4],
            'model__max_features':[5,10,'sqrt',None]}
rf_model = build_model(RandomForestClassifier(random_state=42), parameters)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

In [18]:

```
evaluate(rf_model)
```

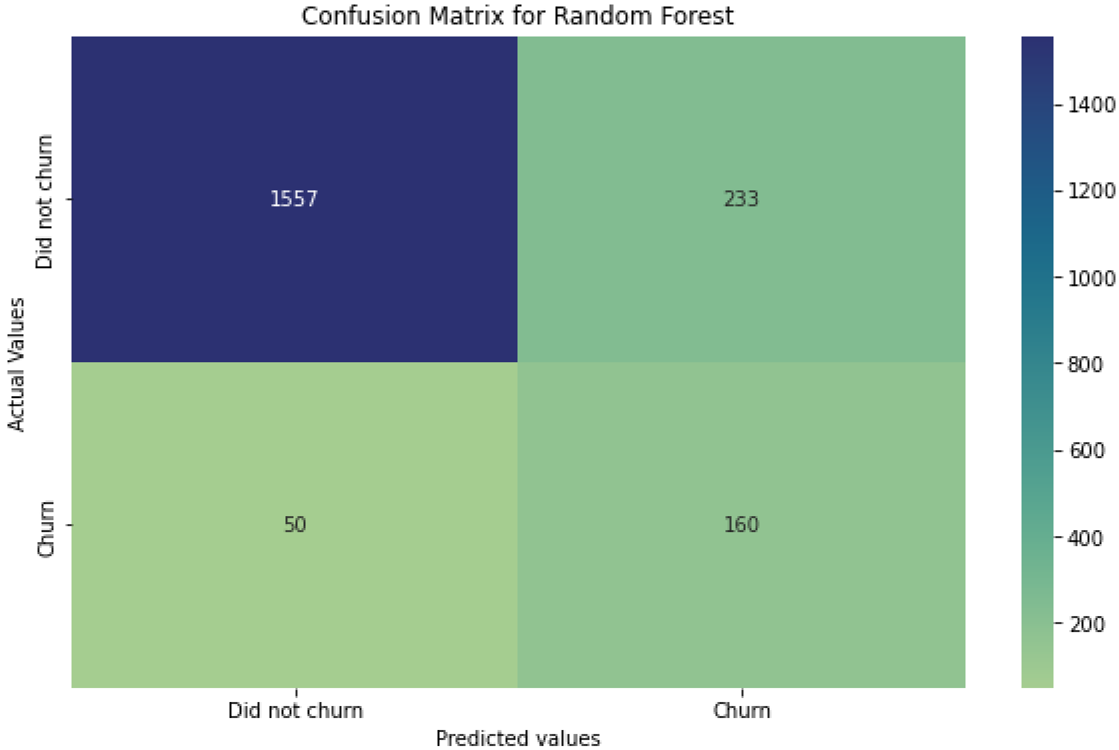
```
The best parameters are: {'model__criterion': 'gini', 'model__max_depth': 4, 'model__max_f
eatures': 10, 'model__n_estimators': 150}
The best score on training set is: 0.85625
Accuracy on test set:  0.8585
The classification scores are:
              precision    recall  f1-score   support

           0       0.97      0.87      0.92      1790
           1       0.41      0.76      0.53       210

    accuracy                           0.86      2000
   macro avg       0.69      0.82      0.72      2000
weighted avg       0.91      0.86      0.88      2000
```

In [19]:

```
plot_cm(rf_model, 'Random Forest')
```



Gradient Boosting

In [20]:

```
parameters = {'model__n_estimators':[100,150],
              'model__criterion':['friedman_mse', 'squared_error'],
              'model__learning_rate':[0.1,0.2],
              'model__max_features':[5,10,None],
              'model__subsample':[1,0.9]}
gb_model = build_model(GradientBoostingClassifier(random_state=42), parameters)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

In [21]:

```
evaluate(gb_model)
```

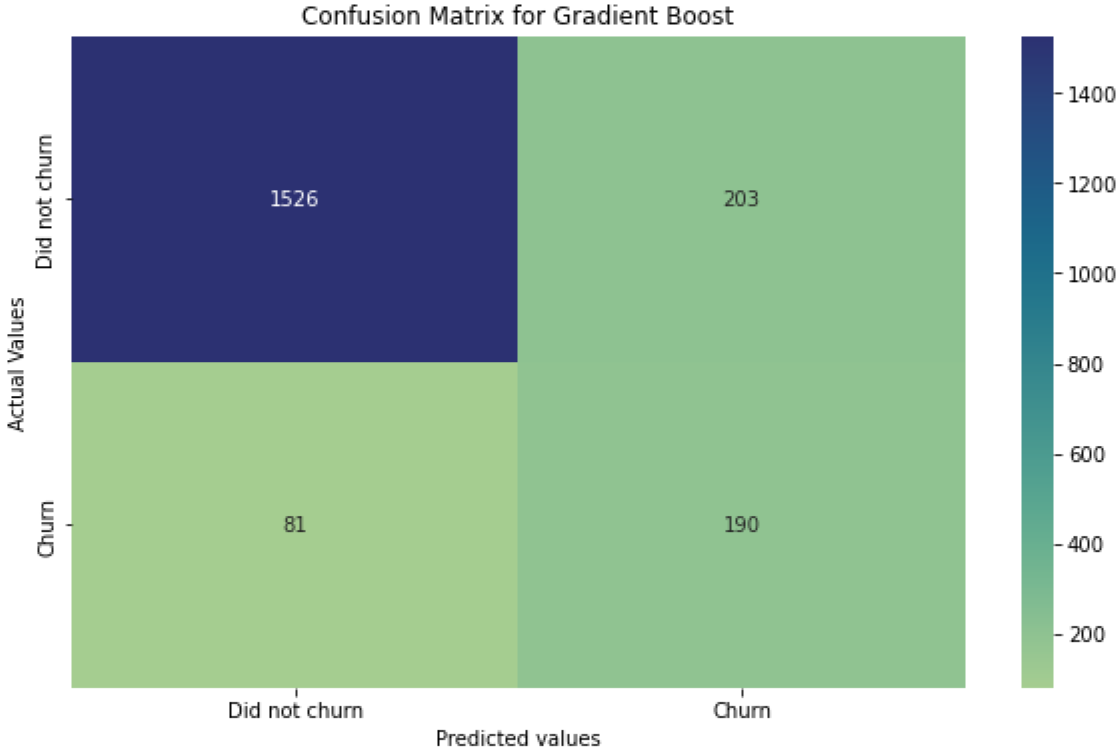
```
The best parameters are: {'model__criterion': 'friedman_mse', 'model__learning_rate': 0.2,
'model__max_features': 10, 'model__n_estimators': 100, 'model__subsample': 0.9}
The best score on training set is: 0.8626250000000001
Accuracy on test set:  0.858
The classification scores are:
              precision    recall  f1-score   support

           0       0.95      0.88      0.91      1729
           1       0.48      0.70      0.57       271

    accuracy                           0.86      2000
   macro avg       0.72      0.79      0.74      2000
weighted avg       0.89      0.86      0.87      2000
```

In [22]:

```
plot_cm(gb_model, 'Gradient Boost')
```



Model Performance Comparison

In [23]:

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

In [24]:

```
scores_df
```

Out[24]:

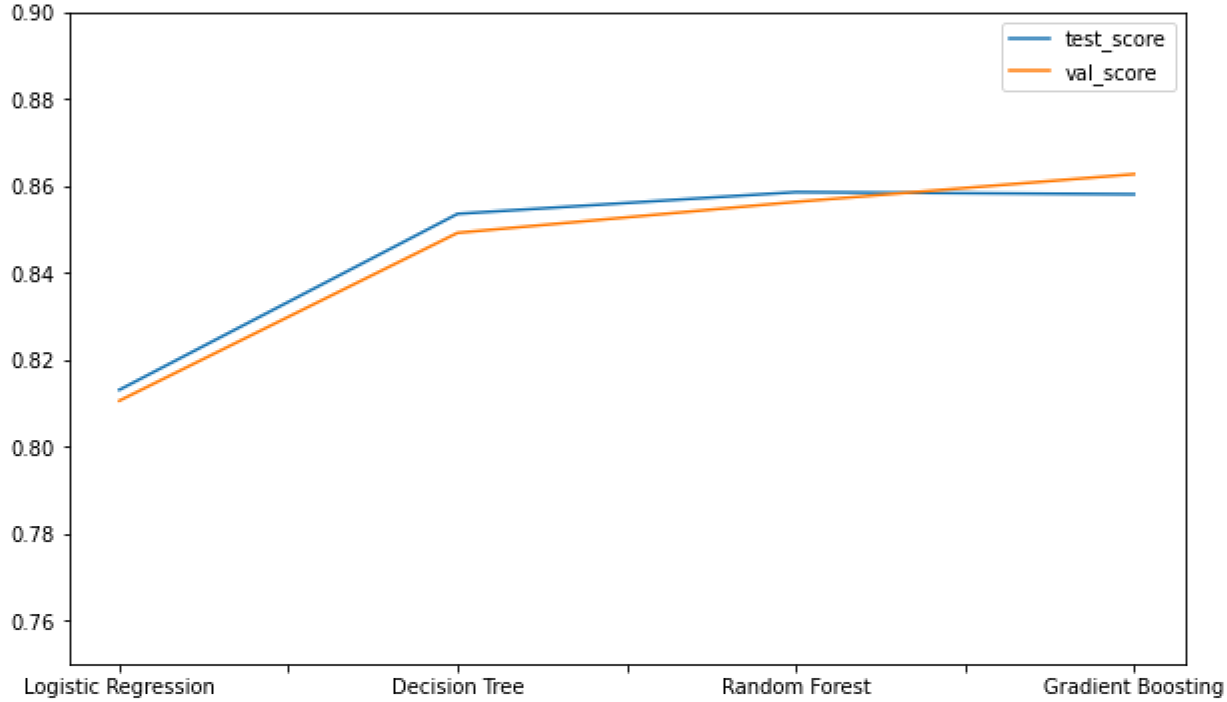
	test_score	val_score	rmse
Logistic Regression	0.8130	0.810500	0.432435
Decision Tree	0.8535	0.849125	0.382753
Random Forest	0.8585	0.856250	0.376165
Gradient Boosting	0.8580	0.862625	0.376829

In [57]:

```
ax = scores_df.drop('rmse',axis=1).plot(kind='line', rot=0, figsize=(10,6))
ax.set(ylim=[0.75,0.9])
```

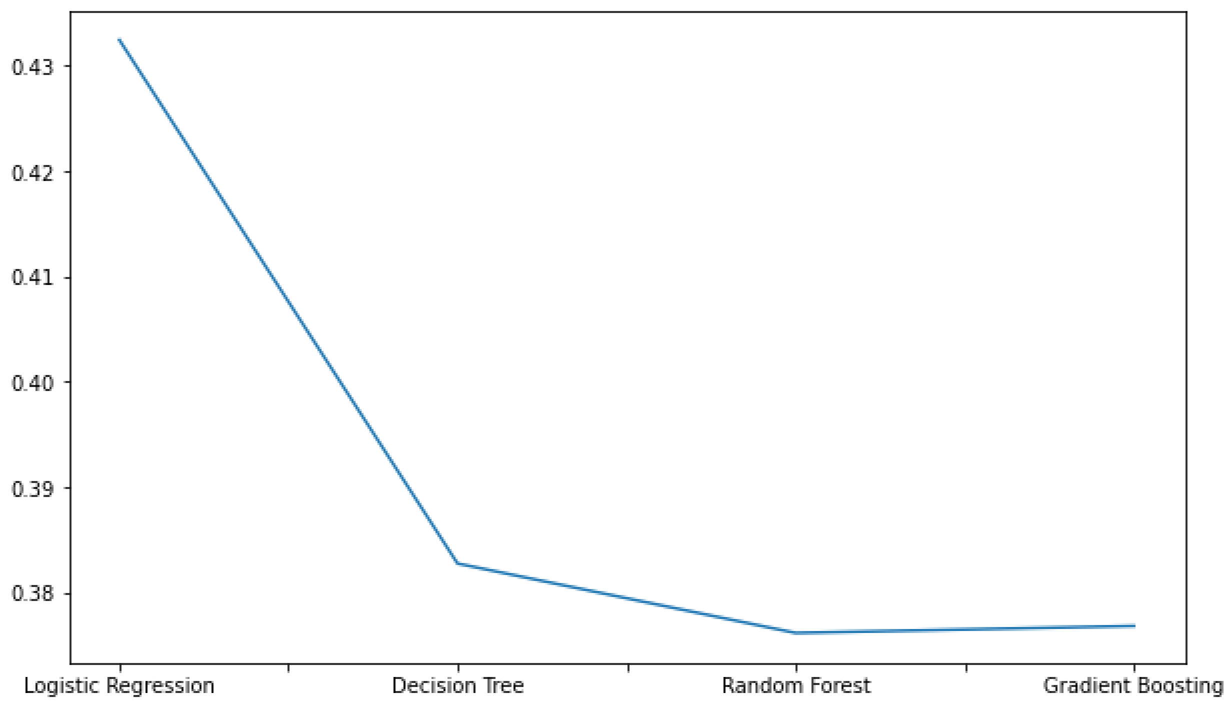
Out[57]:

```
[(0.75, 0.9)]
```



In [59]:

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



There is a small improvement in validation score from logistic regression model. With hyperparameter tuning, gradient boost model achieved a better validation score. The root mean squared error (RMSE) also decreases from logistic regression. From the confusion matrix, the type 2 error increases but the true positive also increases. In conclusion, I found out that `complain` was the main driver for customer to churn. This is based on findings in the exploration phase. Furthermore, gradient boosting model is chosen here as the best model for predicting churn. 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)

RFECV

In [35]:

```
estimator = GradientBoostingClassifier(random_state=42, criterion='friedman_mse', learning
                                       max_features=10, n_estimators=100, subsample=0.9)

rfecv = RFECV(estimator=estimator, cv=StratifiedKFold(n_splits=5))
rfecv.fit(X_train, y_train)
```

Out[35]:

```
RFECV
estimator: GradientBoostingClassifier
    GradientBoostingClassifier
```

In [36]:

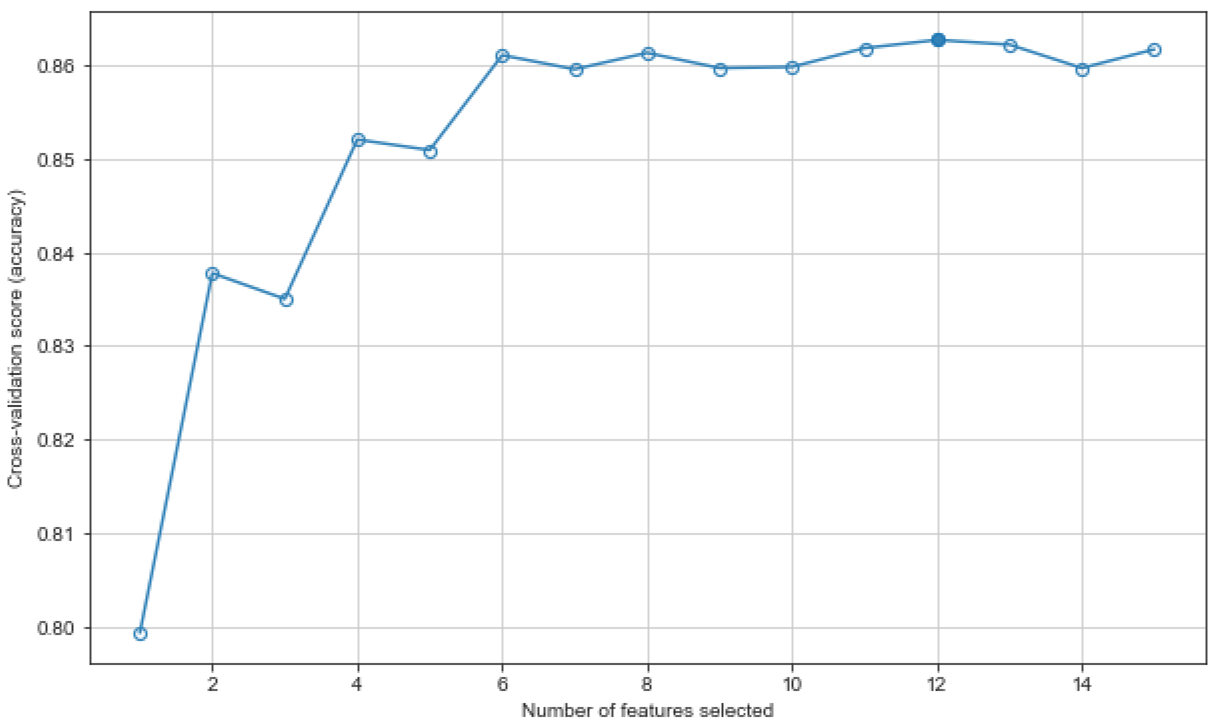
```
selected_features = X_train.columns[rfecv.support_]
selected_features
```

Out[36]:

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

In [61]:

```
plt.figure()
plt.grid(True)
plt.xlabel("Number of features selected")
plt.ylabel("Cross-validation score (accuracy)")
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(12, rfecv.cv_results_['mean_test_score'][11], marker='o', markerfacecolor = '#328
plt.show()
```



The accuracy peaks with 12 features selected. The accuracy scores starts increasing after 1 feature and stopped after 6 features.

Features importance

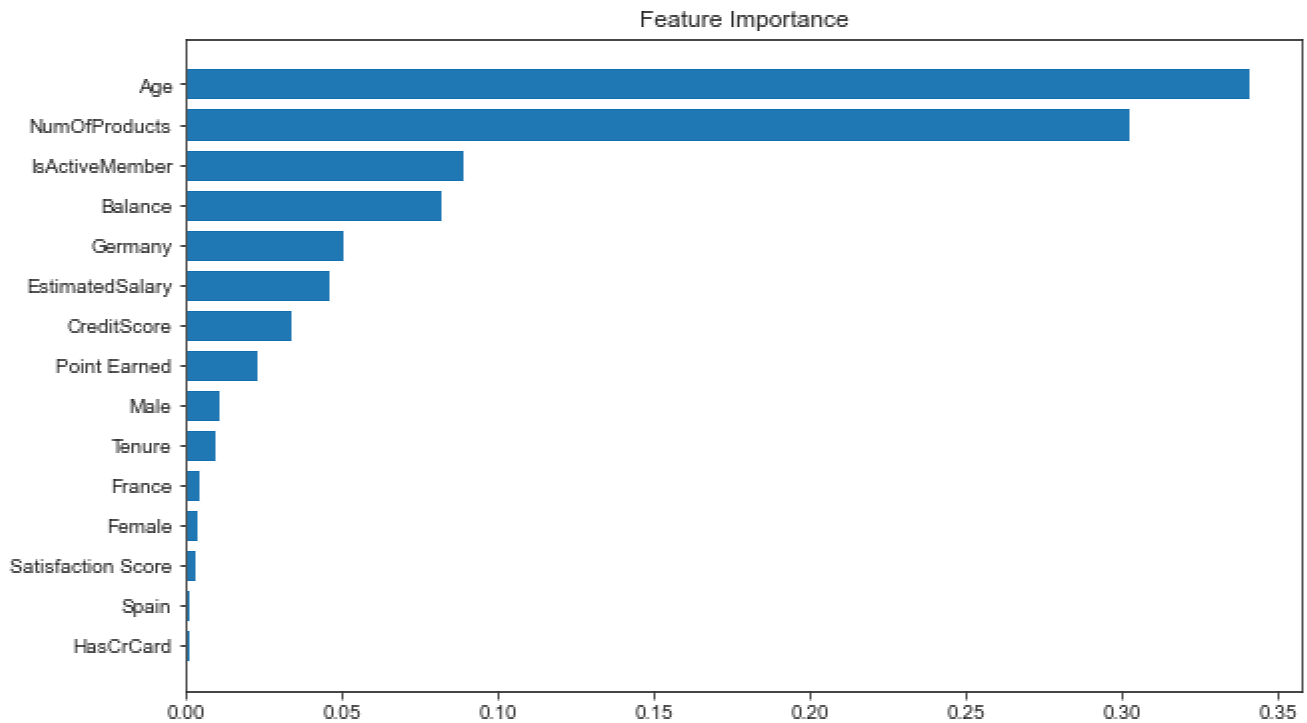
In [69]:

```
estimator.fit(X_train, y_train)
```

In [88]:
```python
sort_indices = estimator.feature_importances_.argsort()
X_train.columns[sort_indices]
```

Out[88]:
```
Index(['HasCrCard', 'Spain', 'Satisfaction Score', 'Female', 'France',
       'Tenure', 'Male', 'Point Earned', 'CreditScore', 'EstimatedSalary',
       'Germany', 'Balance', 'IsActiveMember', 'NumOfProducts', 'Age'],
      dtype='object')
```

In [89]:
```python
plt.figure()
plt.title("Feature Importance")
plt.barh(range(X_train.shape[1]), estimator.feature_importances_[sort_indices])
plt.yticks(range(X_train.shape[1]), X_train.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 the contribution value it holds.

Overall, the bank should focus on building better experience for their customer. This can be solved from the complains that the bank have received as this 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 [92]:
```python
data['Age'].describe()
```

Out[92]:
```
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 [103…

```
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 [106…

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

Out[106…

	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	
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	

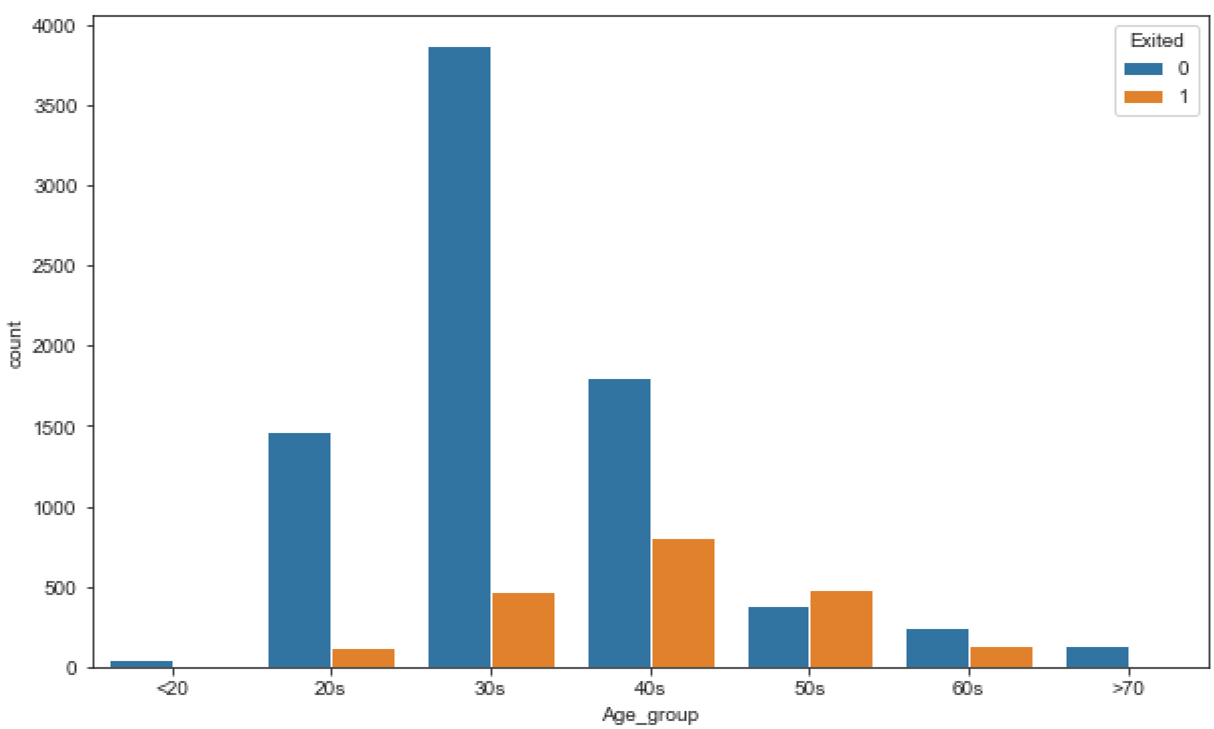
10 rows × 21 columns

In [108…

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

Out[108…

```
<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)

In [121…

```
from sklearn.utils import resample

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']
minor_upsampled = resample(minor_class, replace=True, n_samples=len(major_class),
                          random_state=42)
upsampled_df = pd.concat([major_class, minor_upsampled])
X_train = upsampled_df.drop('Exited', axis=1)
y_train = upsampled_df['Exited']
```

Logistic Regression

In [123…

```
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)
```

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

In [124…

```
evaluate(LR_model)
```

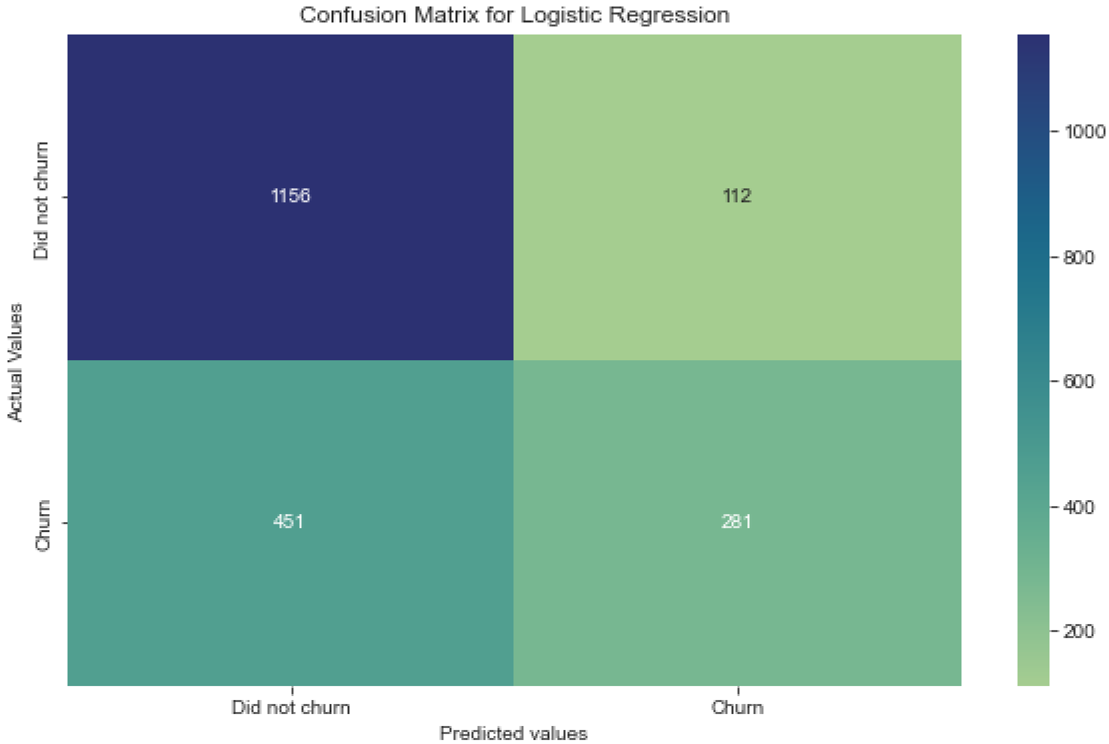
```
The best parameters are: {'model__C': 0.01, 'model__penalty': 'l2', 'model__solver': 'lbfg
s'}
The best score on training set is: 0.6986624704956728
Accuracy on test set:  0.7185
The classification scores are:
              precision    recall  f1-score   support

           0       0.72      0.91      0.80      1268
           1       0.72      0.38      0.50       732

    accuracy                           0.72      2000
   macro avg       0.72      0.65      0.65      2000
weighted avg       0.72      0.72      0.69      2000
```

In [125…

```
plot_cm(LR_model, 'Logistic Regression')
```



The accuracy score is lower than with imbalanced dataset. This can indicate that earlier model may not generalise to outside data.

Gradient Boosting Classification

In [127…

```
parameters = {'model__n_estimators':[100],
              'model__criterion':['friedman_mse', 'squared_error'],
              'model__learning_rate':[0.1,0.2],
              'model__max_features':[5,10,None],
              'model__subsample':[1,0.9]}
gb_model = build_model(GradientBoostingClassifier(random_state=42), parameters)
```

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

In [128…

```
evaluate(gb_model)
```

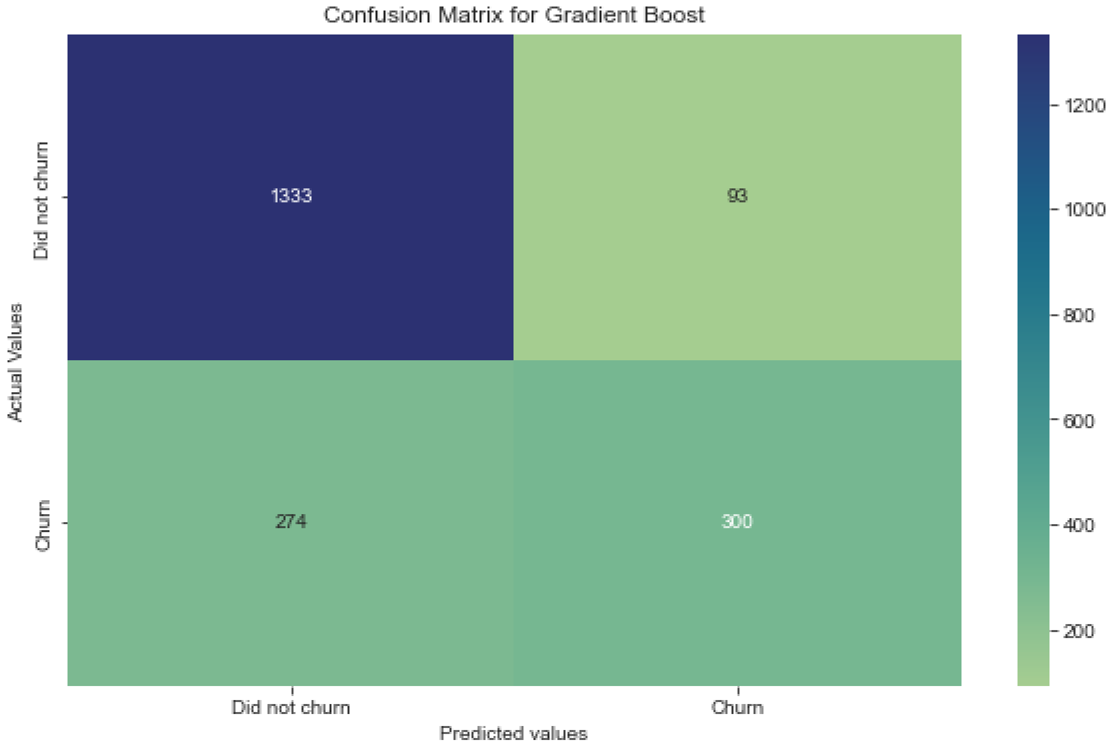
```
The best parameters are: {'model__criterion': 'friedman_mse', 'model__learning_rate': 0.2,
'model__max_features': None, 'model__n_estimators': 100, 'model__subsample': 1}
The best score on training set is: 0.8146341463414635
Accuracy on test set:  0.8165
The classification scores are:
              precision    recall  f1-score   support

           0       0.83      0.93      0.88      1426
           1       0.76      0.52      0.62       574

    accuracy                           0.82      2000
   macro avg       0.80      0.73      0.75      2000
weighted avg       0.81      0.82      0.80      2000
```

In [130…

```
plot_cm(gb_model, 'Gradient Boost')
```



The accuracy score is better now. Recall has increased greatly with gradient boosting. Now, we need to repeat the same process using RFECV.

In [131…

```
estimator = GradientBoostingClassifier(random_state=42, criterion='friedman_mse', learning
                                       max_features=None, n_estimators=100, subsample=1)

rfecv = RFECV(estimator=estimator, cv=StratifiedKFold(n_splits=5))
rfecv.fit(X_train, y_train)
```

Out[131…

```
RFECV
estimator: GradientBoostingClassifier
    GradientBoostingClassifier
```

In [132…

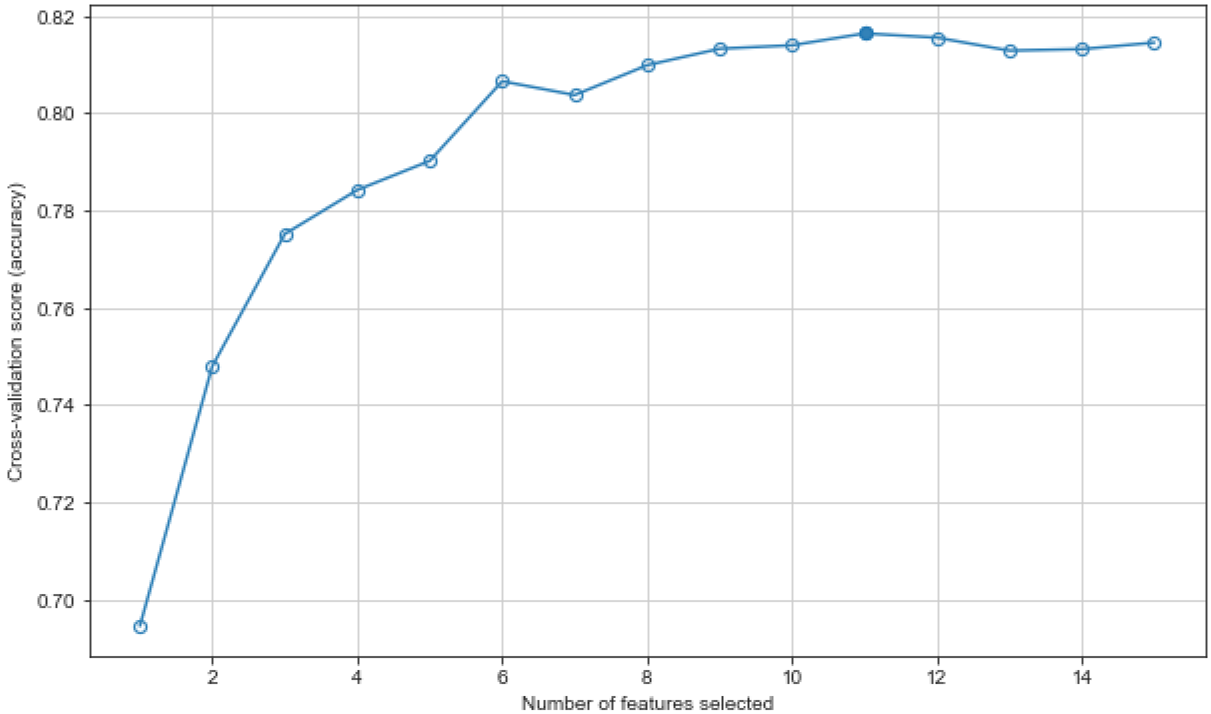
```
selected_features = X_train.columns[rfecv.support_]
selected_features
```

Out[132…

```
Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
       'IsActiveMember', 'EstimatedSalary', 'Satisfaction Score',
       'Point Earned', 'Female', 'Germany'],
      dtype='object')
```

In [138…

```
plt.figure()
plt.grid(True)
plt.xlabel("Number of features selected")
plt.ylabel("Cross-validation score (accuracy)")
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(11, rfecv.cv_results_['mean_test_score'][10], marker='o', markerfacecolor = '#328
plt.show()
```



With balanced dataset/resample, the accuracy score peaks with 11 features. The accuracy stopped increasing after 6 features which is the same with earlier experiment.

In [139…

```
estimator.fit(X_train, y_train)
```

Out[139…

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.2, random_state=42, subsample=1)
```

In [140…

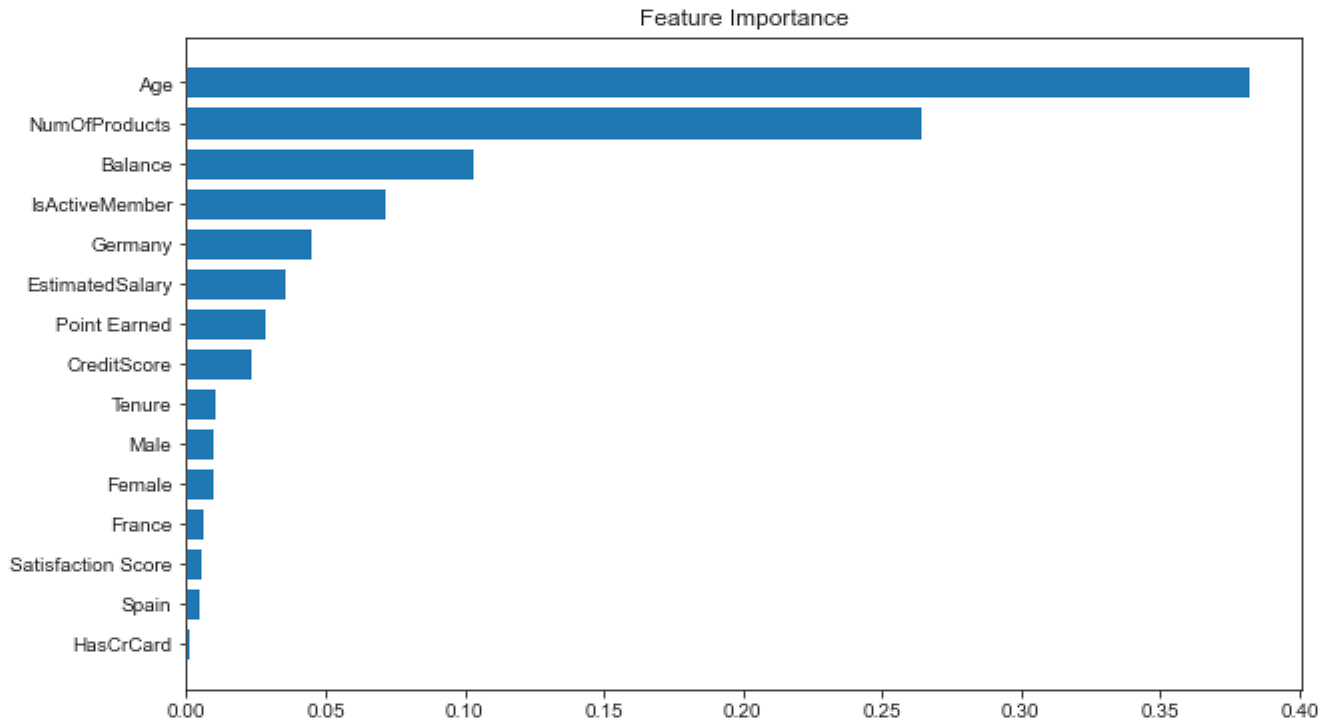
```
sort_indices = estimator.feature_importances_.argsort()
X_train.columns[sort_indices]
```

Out[140…

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

In [141…

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



The top 2 features are still age and number of products. The difference is in the third place which now has became balance.

In [ ]: