# Predicting Customer Churn using Bank Customers

## 1. Abstract

Customer churn (also known as customer attrition) occurs when a customer stops using a company's products or services.

The goal of this notebook is to understand and predict customer churn for a bank usig Random forest and SHAP values.

Intially, performed Exploratory Data Analysis (EDA) to identify and visualise the factors that contribute to customer churn. Then I used Random Forest to predict whether a customer will churn or not. Accuracy of the model is 84.1%.

Most important factors are age, gender, geography, number of products and if a customer is active member or not

# Problem with this notebook

1. Gender: one-hot encoding
2. Didn't normalize the data
3. Summary plot -> not as two different class i.e remained ad churned
4. Output from summary plot does not support the EDA graphs

## 2. Import libraries

Importing libraries:

Pandas for handling and analysing data, Seaborn and Matplotlib for data visualization, and Scikit-learn for building Machine Learning models.

```
import warnings
warnings.simplefilter(action = 'ignore', category = FutureWarning)

import numpy as np
import pandas as pd
pd.set_option('precision', 3)
pd.options.mode.chained_assignment = None

import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'

import seaborn as sns
sns.set_style('darkgrid')

from scipy.stats import chi2_contingency
from collections import Counter
from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import cross_val_score, cross_val_predict, GridSearchCV, learning_curve
from sklearn.metrics import accuracy_score, recall_score, precision_score, auc, roc_auc_score, roc_curve
from sklearn.metrics import confusion_matrix, plot_confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import scikitplot as skplt

label_size = 17

plt.rcParams['axes.labelsize'] = label_size
plt.rcParams['xtick.labelsize'] = label_size - 2
plt.rcParams['ytick.labelsize'] = label_size - 2
plt.rcParams['axes.titlesize'] = label_size
plt.rcParams['legend.fontsize'] = label_size - 2

random_state = 42
scoring_metric = 'recall'
comparison_dict = {}
comparison_test_dict = {}
```

```
df = pd.read_csv('Churn_Modelling.csv')

print ('This dataset contains {} rows and {} columns.'.format(df.shape[0], df.shape[1]))
df.head()
```

```
This dataset contains 10000 rows and 14 columns.
```

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Bal |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | |
| **1** | 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 8380 |
| **2** | 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 15966 |
| **3** | 4 | 15701354 | Boni | 699 | France | Female | 39 | 1 | |
| **4** | 5 | 15737888 | Mitchell | 850 | Spain | Female | 43 | 2 | 12551 |

Details about the dataset:

It consists of 10000 observations and 12 variables. Independent variables contain information about customers. Dependent variable refers to customer abandonment status.

Variables:

RowNumber — corresponds to the record (row) number and has no effect on the output. This column will be removed.

CustomerId — contains random values and has no effect on customer leaving the bank. This column will be removed.

Surname — the surname of a customer has no impact on their decision to leave the bank. This column will be removed.

CreditScore — can have an effect on customer churn, since a customer with a higher credit score is less likely to leave the bank.

Geography — a customer's location can affect their decision to leave the bank. We'll keep this column.

Gender — it's interesting to explore whether gender plays a role in a customer leaving the bank. We'll include this column, too.

Age — this is certainly relevant, since older customers are less likely to leave their bank than younger ones.

Tenure — refers to the number of years that the customer has been a client of the bank. Normally, older clients are more loyal and less likely to leave a bank.

Balance — also a very good indicator of customer churn, as people with a higher balance in their accounts are less likely to leave the bank compared to those with lower balances.

NumOfProducts — refers to the number of products that a customer has purchased through the bank.

HasCrCard — denotes whether or not a customer has a credit card. This column is also relevant, since people with a credit card are less likely to leave the bank. (0=No,1=Yes)

IsActiveMember — active customers are less likely to leave the bank, so we'll keep this. (0=No,1=Yes)

EstimatedSalary — as with balance, people with lower salaries are more likely to leave the bank compared to those with higher salaries.

Exited — whether or not the customer left the bank. This is what we have to predict. (0=No,1=Yes)

The info() method can give us valuable information such as the number of non-null values and the type of each feature:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

There are no missing values in the dataset.

Columns 'RowNumber', 'CustomerID' and 'Surname' are specific to each customer and can be dropped:

```
df.drop(['RowNumber','CustomerId','Surname'],axis=1, inplace = True)
df.columns
```

```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

## 3. Exploratory data analysis

```
import pandas_profiling
df.profile_report()
```

```
HBox(children=(HTML(value='Summarize dataset'), FloatProgress(value=0.0, max=23.0),
HTML(value='')))

HBox(children=(HTML(value='Generate report structure'), FloatProgress(value=0.0,
max=1.0), HTML(value='')))

HBox(children=(HTML(value='Render HTML'), FloatProgress(value=0.0, max=1.0),
HTML(value='')))
```

Exited

Boolean

| | |
|---|---|
| **Distinct count** | 2 |
| **Unique (%)** | < 0.1% |
| **Missing** | 0 |
| **Missing (%)** | 0.0% |
| **Memory size** | 78.2 KiB |

| | |
|---|---|
| 0 | 7963 |
| 1 | 2037 |

Toggle details

# Interactions

CreditScore Age Tenure Balance EstimatedSalary

CreditScore Age Tenure Balance EstimatedSalary


```
df.describe().T
```

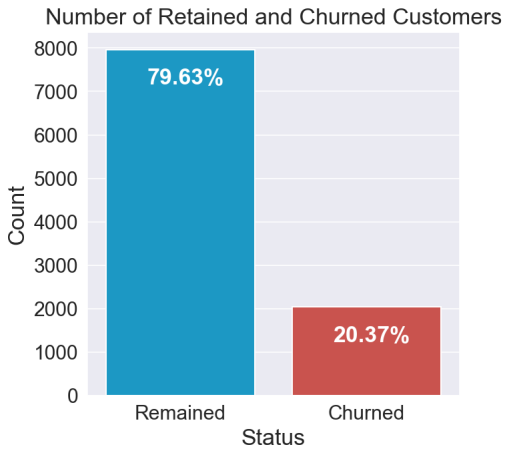
| | count | mean | std | min | 25% | 50% | 75% |
|---|---|---|---|---|---|---|---|
| **CreditScore** | 10000.0 | 650.529 | 96.653 | 350.00 | 584.00 | 652.000 | 718.000 |

The most important things to note are:

1. The age of customers ranges from 18 to 92, with a mean value approximately equal to 40,
2. The mean (and median) tenure is 5 years, so the majority of customers is loyal (tenure > 3), and
3. Approximately 50% of customers are active.

```
colors = ['#00A5E0', '#DD403A']

fig = plt.figure(figsize = (5, 5))

# plot the graph
sns.countplot(x = 'Exited', data = df, palette = colors)

for index, value in enumerate(df['Exited'].value_counts()):
    label =  '{}%'.format(round( (value/df['Exited'].shape[0])*100, 2))
    plt.annotate(label, xy = (index - 0.18, value - 800), color = 'w', fontweight = 'bold', size = label_size)

plt.title('Number of Retained and Churned Customers')
plt.xticks([0, 1], ['Remained', 'Churned'])
plt.xlabel('Status')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```




The bank kept 80% of its clientele.

The dataset is skewed/imbalanced since the number of instances in the 'Remained' class outnumbers the number of instances in the 'Churned' class by a lot.

```
## feature importance
from scipy.stats import ttest_1samp
import numpy as np

ages = np.genfromtxt(“Churn_Modelling.csv”)
print(ages)
ages_mean = np.mean(ages)
print(ages_mean)
tset, pval = ttest_1samp(ages, 30)
print(“p-values”,pval)
if pval < 0.05:    # alpha value is 0.05 or 5%
   print(" we are rejecting null hypothesis")
else:
  print("we are accepting null hypothesis")
```

```
  File "<ipython-input-8-17073aa0c97b>", line 5
    ages = np.genfromtxt(“Churn_Modelling.csv”)
                                               ^
SyntaxError: invalid character in identifier
```

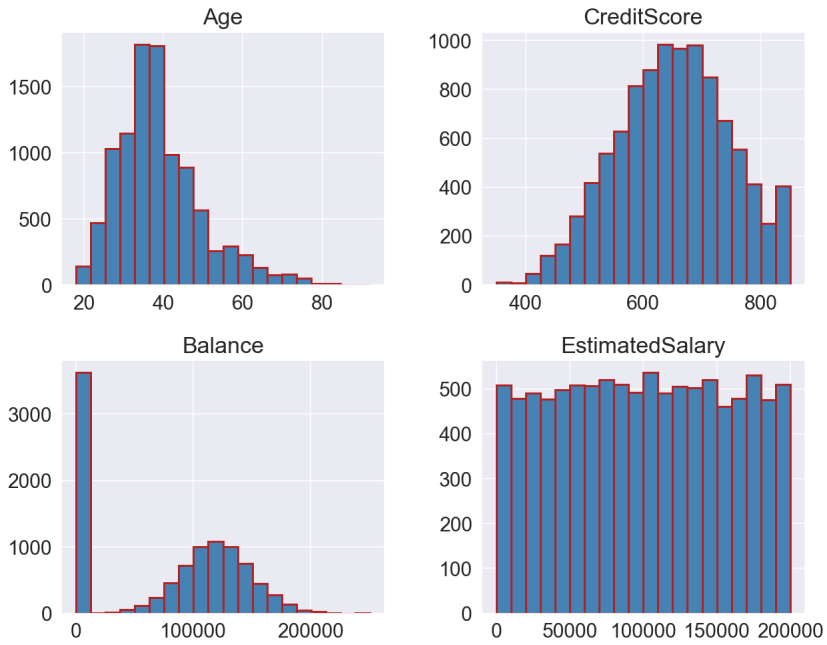
SEARCH STACK OVERFLOW

## Continuous Variables - Age, CreditScore, Balance, EstimatedSalary

By calling the hist() method we can plot a histogram for each of the four continuous numeric features:

```
continuous = ['Age','CreditScore','Balance','EstimatedSalary']
df[continuous].hist(figsize =(10,8), bins = 20, layout=(2,2), color = 'steelblue', edgecolor = 'firebrick', linewidth = 1.5)
```

```
array([[<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'CreditScore'}>],
       [<AxesSubplot:title={'center':'Balance'}>,
        <AxesSubplot:title={'center':'EstimatedSalary'}>]], dtype=object)
```



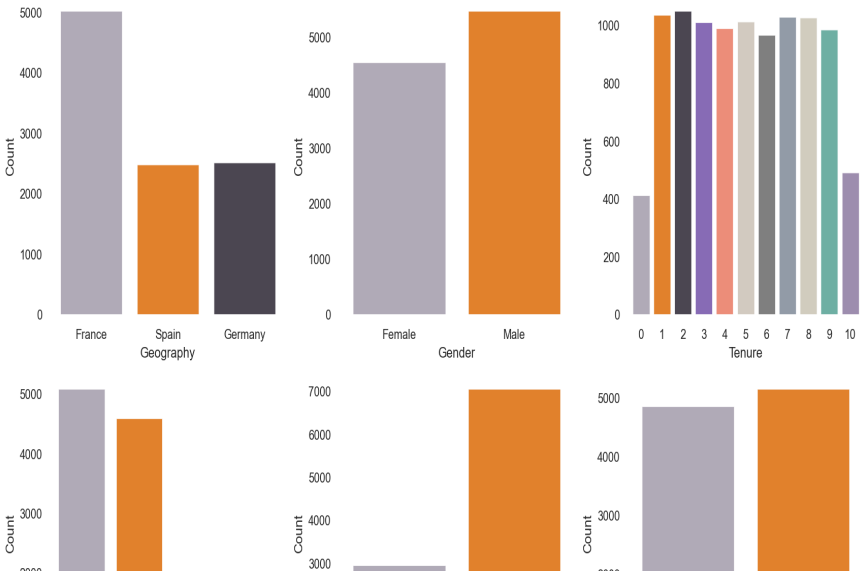

### Observations:

1. 'Age' is slightly tail-heavy, i.e. it extends more further to the right of the median than to the left
2. Most values for 'CreditScore' are above 600
3. If we ignore the fist bin, 'Balance' follows a fairly normal distribution
4. The distribution of 'EstimatedSalary' is more or less uniform and provides little information.

## Categorical Variables - Geography, Gender, HasCreditCard,Tenure,NumberOfProducts, IsAxtiveMember

Let's plot a countplot for each categorical feature:

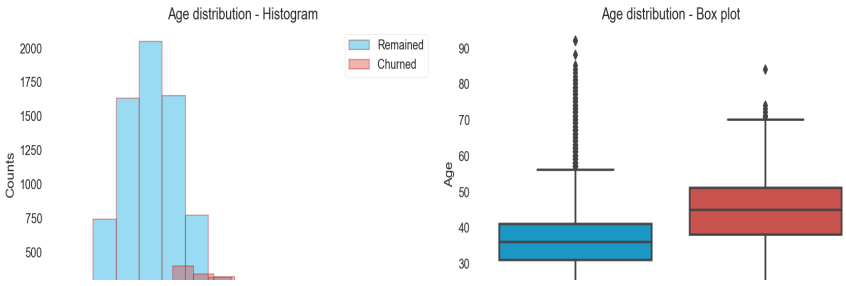
```
colors2 = ['#B0A8B9', '#ff7f0e', '#4B4453', '#845EC2', '#FF8066', '#D5CABD',
           '#7f7f7f', '#8f9aaa', '#d4cebb',  '#63BAAA', '#9D88B3']

cat_vars = ['Geography', 'Gender', 'Tenure', 'NumOfProducts', 'HasCrCard', 'IsActiveMember']
df_cat = df[cat_vars]

fig, ax = plt.subplots(2, 3, figsize = (12, 8))

for index, column in enumerate(df_cat.columns):

    plt.subplot(2, 3, index + 1)
    sns.countplot(x = column, data = df, palette = colors2)

    plt.ylabel('Count')
    if (column == 'HasCrCard' or column == 'IsActiveMember'):
        plt.xticks([0, 1], ['No', 'Yes'])

plt.tight_layout();
```




## Observations:

1. The bank has customers in three countries (France, Spain, and Germany). Most customers are in France.
2. There are more male customers than females
3. Only a small percentage leaves within the first year. The count of customers in tenure years between 1 and 9 is almost the same,
4. Most of the customers have purchased 1 or 2 products, while a small portion has purchased 3 and 4,
5. A significant majority of customers has a credit card, and
6. Almost 50% of customers are not active.

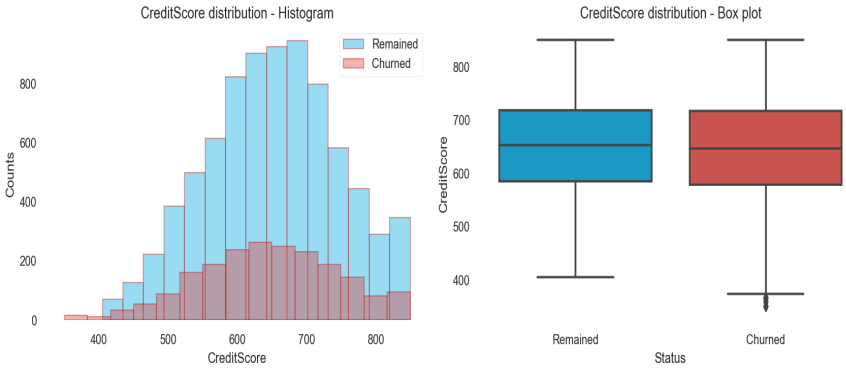
```
def plot_continuous(feature):

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 4))

    sns.distplot(df_remained[feature], bins = 15, color = colors[0], label = 'Remained',
                 hist_kws = dict(edgecolor = 'firebrick', linewidth = 1), ax = ax1, kde = False)
    sns.distplot(df_churned[feature], bins = 15, color = colors[1], label = 'Churned',
                 hist_kws = dict(edgecolor = 'firebrick', linewidth = 1), ax = ax1, kde = False)
    ax1.set_title('{} distribution - Histogram'.format(feature))
    ax1.set_ylabel('Counts')
    ax1.legend()

    sns.boxplot(x = 'Exited', y = feature, data = df, palette = colors, ax = ax2)
    ax2.set_title('{} distribution - Box plot'.format(feature))
    ax2.set_xlabel('Status')
    ax2.set_xticklabels(['Remained', 'Churned'])

    plt.tight_layout();


def plot_categorical(feature):

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 4))
    sns.countplot(x = feature, hue = 'Exited', data = df, palette = colors, ax = ax1)
    ax1.set_ylabel('Counts')
    ax1.legend(labels = ['Retained', 'Churned'])

    sns.barplot(x = feature, y = 'Exited', data = df, palette = colors2 , ci = None, ax = ax2)
    ax2.set_ylabel('Churn rate')

    if (feature == 'HasCrCard' or feature == 'IsActiveMember'):
        ax1.set_xticklabels(['No', 'Yes'])
        ax2.set_xticklabels(['No', 'Yes'])

    plt.tight_layout();


df_churned = df[df['Exited'] == 1]
df_remained = df[df['Exited'] == 0]

plot_continuous('Age')
```




There is a clear difference between age groups since older customers are more likely to churn. This could potentially indicate that preferences change with age, and the bank hasn't adapted its strategy to meet the requirements of older customers.

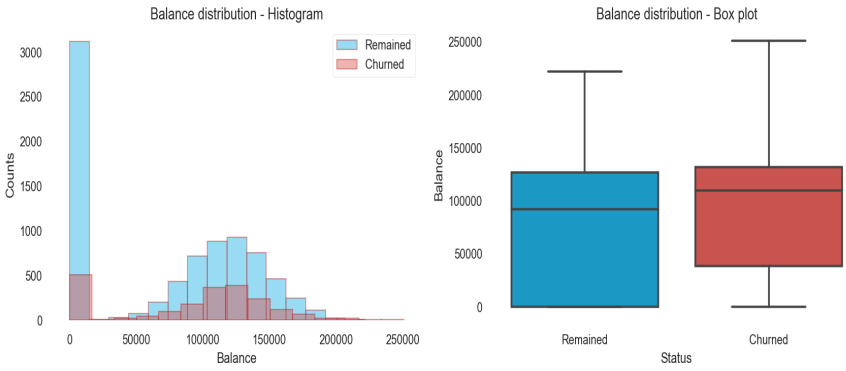
It can be clearly observed that 50% of the people likely to churn at the age of 50.

```
plot_continuous('CreditScore')
```



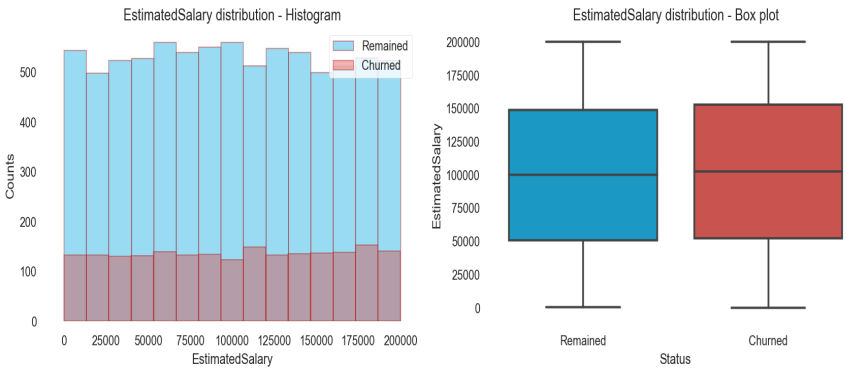

There is no significant difference between retained and churned customers in terms of their credit score.

```
plot_continuous('Balance')
```



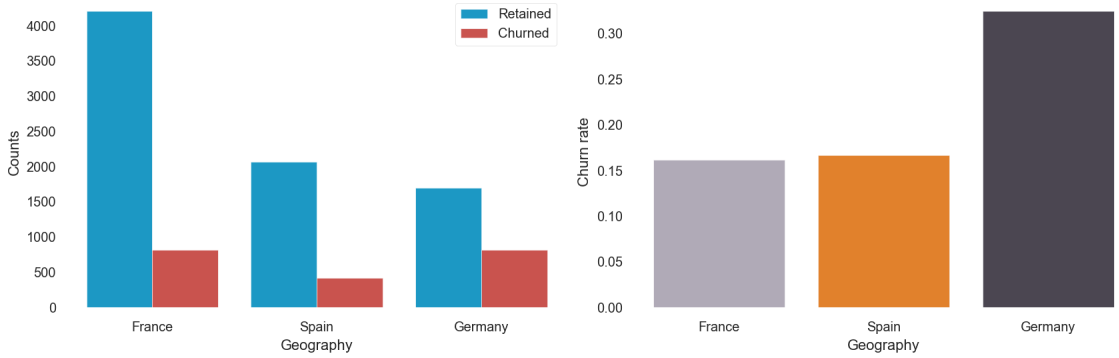

The two distributions are quite similar. There is a big percentage of non-churned customers with a low account balance.

```
plot_continuous('EstimatedSalary')
```



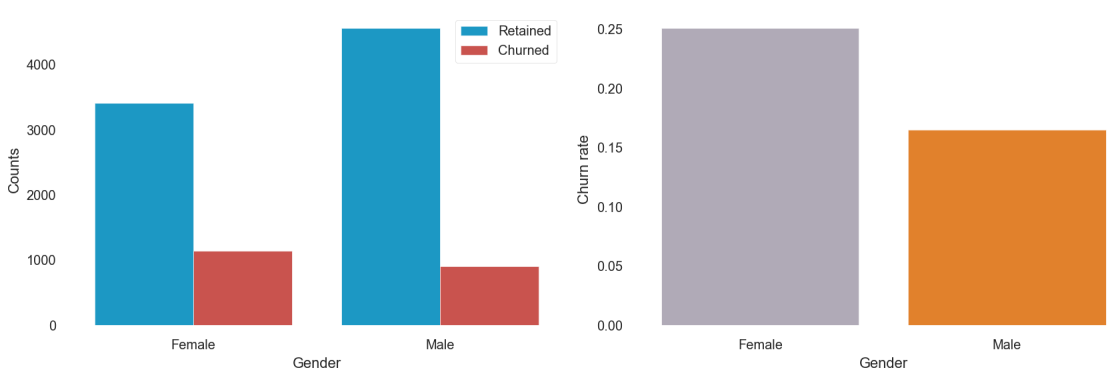

Both churned and retained customers display a similar uniform distribution for their salary. Consequently, we can conclude that salary doesn't have a significant effect on the likelihood to churn.

```
plot_categorical('Geography')
```



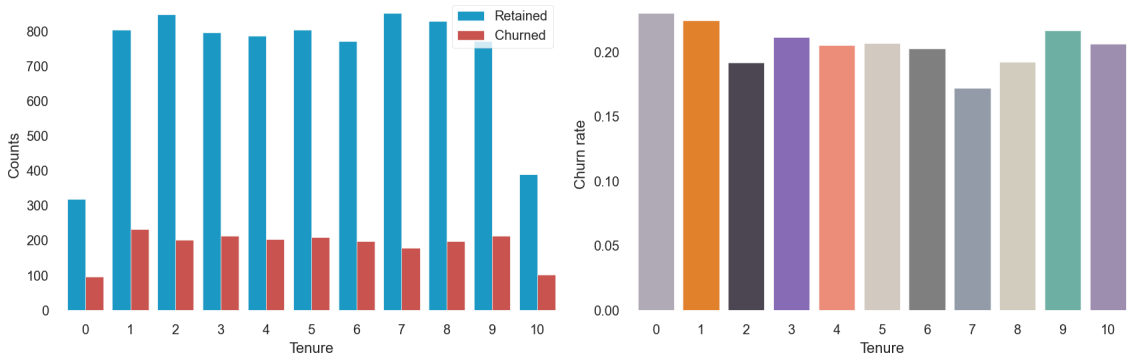

Customers in Germany are more likely to churn than customers in the other two countries (the churn rate is almost double compared to Spain and France). Many reasons could explain this finding such as higher competition or different preferences for German customers.

```
plot_categorical('Gender')
```




Female customers are more likely to churn. ( which is more imp age or gender)

```
plot_categorical('Tenure')
```



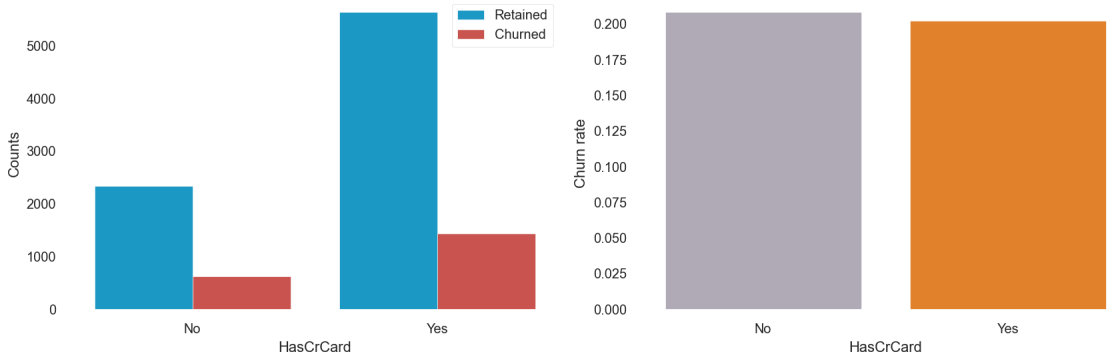

Tenure does not seem to affect the customer churn. It is most likely for customers to churn either at the start or end of the tenure

```
plot_categorical('NumOfProducts')
```

having 3 or 4 products significantly increases the likelihood of churn.

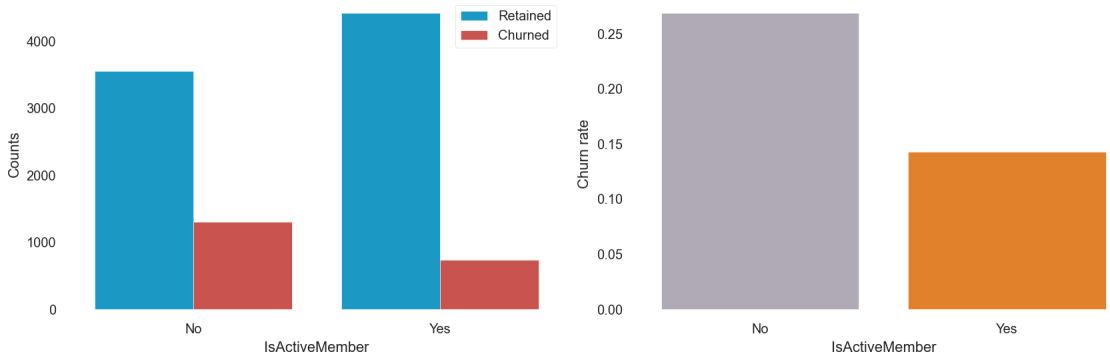
I am not sure how to interpret this result. It could potentially mean that the bank is unable to properly support customers with more products which in turn increases customer dissatisfaction.

```
plot_categorical('HasCrCard')
```




Having a credit card doesn't seem to affect the churn rate.

```
plot_categorical('IsActiveMember')
```




It's not a surprise that inactive customers are more likely to churn. A big portion of the clientele is inactive; therefore, the bank will definitely benefit from changing its policy so that more customers become active.

## 4. Data Preprocessing

Data preprocessing is the process of converting raw data into a well-readable format that is suitable for building and training Machine Learning models.

Encoding Categorical Features:

Machine learning algorithms usually require that all input (and output) features are numeric. Consequently, categorical features need to be converted (encoded) to numbers before using them for building models.

In this dataset, geography and gender was encoded.

```
# One-Hot encoding the categorical attributes
list_cat = ['Geography', 'Gender']
df = pd.get_dummies(df, columns = list_cat, prefix = list_cat)
#df.head()
```

```
# Arrange columns by data type for easier manipulation
continuous_vars = ['CreditScore',  'Age', 'Tenure', 'Balance','NumOfProducts', 'EstimatedSalary']
cat_vars = ['HasCrCard', 'IsActiveMember','Gender_Female','Gender_Male','Geography_France',
            'Geography_Germany','Geography_Spain']
df = df[continuous_vars + cat_vars + ['Exited']]
df.head()
```

| | CreditScore | Age | Tenure | Balance | NumOfProducts | EstimatedSalary | HasCrCard | IsActiveMember | Gender_Female | Gender_Male | Geogra |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 619 | 42 | 2 | 0.00 | 1 | 101348.88 | 1 | 1 | 1 | 0 | |
| **1** | 608 | 41 | 1 | 83807.86 | 1 | 112542.58 | 0 | 1 | 1 | 0 | |
| **2** | 502 | 42 | 8 | 159660.80 | 3 | 113931.57 | 1 | 0 | 1 | 0 | |
| **3** | 699 | 39 | 1 | 0.00 | 2 | 93826.63 | 0 | 0 | 1 | 0 | |

```
# New dataframe with important features
df_bank = df[['CreditScore','Tenure','EstimatedSalary','HasCrCard','Geography_Germany','IsActiveMember',
              'Gender_Female','Gender_Male','Age', 'Geography_France','Balance','Geography_Spain','NumOfProducts','Exited']]

df_bank.head()
```

| | CreditScore | Tenure | EstimatedSalary | HasCrCard | Geography_Germany | IsActiveMember | Gender_Female | Gender_Male | Age | Geography_Fr |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 619 | 2 | 101348.88 | 1 | 0 | 1 | 1 | 0 | 42 | |
| **1** | 608 | 1 | 112542.58 | 0 | 0 | 1 | 1 | 0 | 41 | |
| **2** | 502 | 8 | 113931.57 | 1 | 0 | 0 | 1 | 0 | 42 | |
| **3** | 699 | 1 | 93826.63 | 0 | 0 | 0 | 1 | 0 | 39 | |
| **4** | 850 | 2 | 79084.10 | 1 | 0 | 1 | 1 | 0 | 43 | |

## 7. Train test split

The code below splits the training data. Using 75% of the training data for actual training purposes, and once training is completed, using the remaining 25% of the training data to check the training accuracy of the trained model.

```
x = df_bank.drop('Exited', axis=1)
y = df_bank.Exited

# Splitting the dataset in training and test set
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)



# Because it's an unstable data set,increase the number of samples.
# References: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.combine.SMOTETomek.html
from imblearn.combine import SMOTETomek

smk = SMOTETomek()
# Oversample training  data
X_train, y_train = smk.fit_sample(X_train, y_train)

# Oversample validation data
X_test, y_test = smk.fit_sample(X_test, y_test)


from xgboost import XGBClassifier
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
```

```
C:\Users\prana\anaconda3\envs\deeplearning\lib\site-packages\xgboost\sklearn.py:888: UserWarning: The use of label encoder in XGBCla
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[21:42:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=12, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
print(model)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=12, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
```

```
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 87.87%
```

## 7. Interpretation using SHAP

TreeExplainer is a package for explaining and interpreting predictions of tree-based machine learning models. The notion of interpretability is based on how close the inclusion of a feature takes the model toward its final prediction. For this reason, the result of this approach is "feature contributions" to the predictions.

The basic idea is to decompose each prediction into feature contribution components. For a dataset with n features, each prediction on the dataset is calculated as

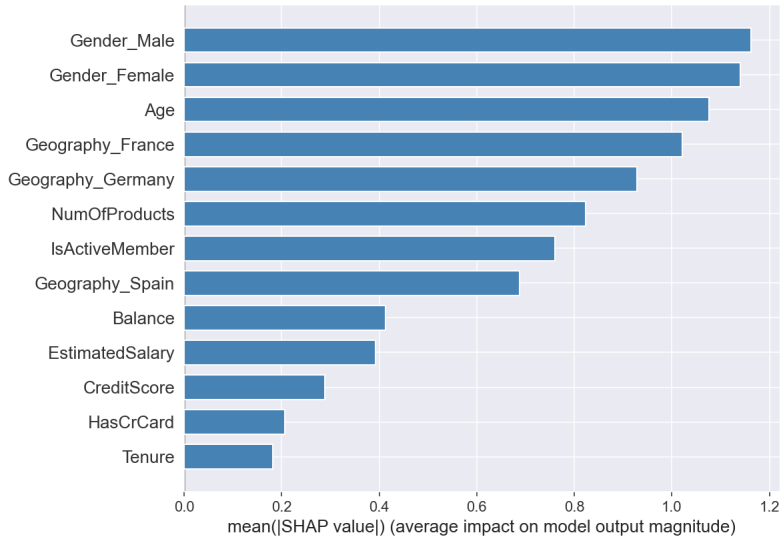
prediction ≈ baseline probability at tree root + contribution feature 1 + ... + contribution feature n

Feature contributions are provided at the level of observations, features, and targets.

```
#SHAP import
import shap
```

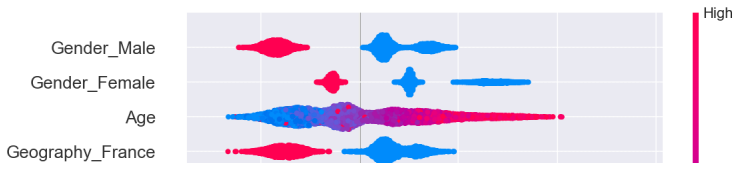
```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_train,approximate=False, check_additivity=False)
```

```
# summarize the effects of all the features
shap.summary_plot(shap_values, X_train, plot_type="bar", color='steelblue')
```




Most important factors are age, gender, geography, number of products and if a customer is active member or not

```
# summarize the effects of all the features
shap.summary_plot(shap_values, X_train)
```



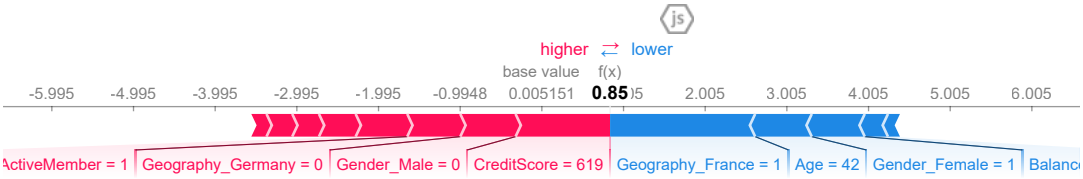

age : high age remain wth bank?? (wrong)

male: if a person is not male then he remains as customers

female: if a person is not female the she remains as cutomers (not correct)....


## Force plot

```
shap.initjs()
# visualize the first prediction's explanation (use matplotlib=True to avoid Javascript)
shap.force_plot(explainer.expected_value, shap_values[0,:], x.iloc[0,:])
```




```
# visualize the first prediction's explanation (use matplotlib=True to avoid Javascript)
shap.force_plot(explainer.expected_value, shap_values[0,:], x.iloc[1,:])
```

**Visualization omitted, Javascript library not loaded!**
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

```
# visualize the first prediction's explanation (use matplotlib=True to avoid Javascript)
shap.force_plot(explainer.expected_value, shap_values[1,:], x.iloc[0,:])
```

**Visualization omitted, Javascript library not loaded!**
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

```
shap.force_plot(explainer.expected_value, shap_values[1,:], x.iloc[1,:])
```

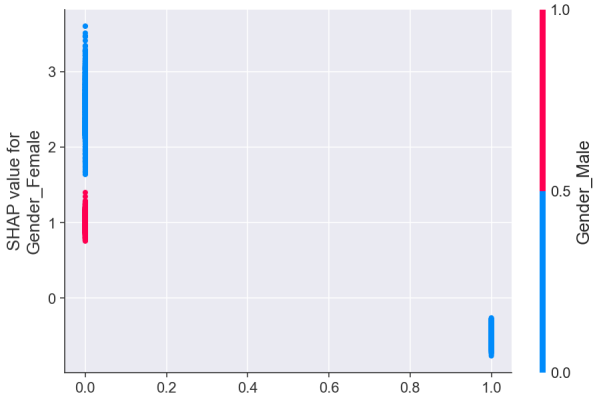
**Visualization omitted, Javascript library not loaded!**
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

```
# create a dependence plot to show the effect of a single feature across the whole dataset
shap.dependence_plot("Age", shap_values, X_train)
```

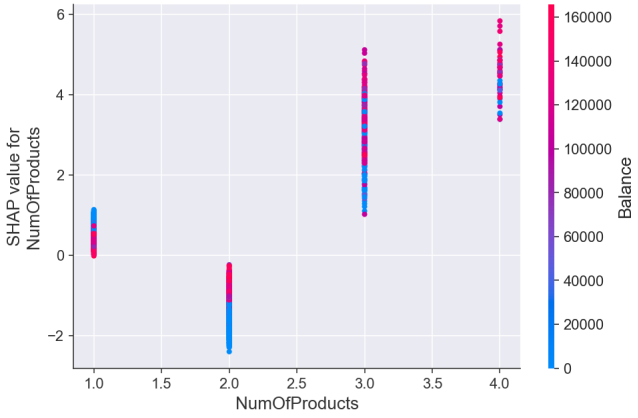
```
Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and will become an error two minor releases later. Plea
```


```
# create a dependence plot to show the effect of a single feature across the whole dataset
shap.dependence_plot("Gender_Female", shap_values, X_train)
```

```
Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and will become an error two minor releases later. Plea
```




```
# create a dependence plot to show the effect of a single feature across the whole dataset
shap.dependence_plot("NumOfProducts", shap_values, X_train)
```




```
shap.TreeExplainer(model).shap_interaction_values(x)
```

```
array([[[ 6.82297051e-02,  2.01554559e-02, -3.72037292e-04, ...,
          4.87112552e-02, -2.57200003e-03, -2.61183679e-02],
        [ 2.01554447e-02,  1.04926601e-02,  4.76177186e-02, ...,
          3.86958942e-02, -1.51947141e-03,  1.26273036e-02],
        [-3.71985137e-04,  4.76176962e-02, -1.76827356e-01, ...,
         -6.46543726e-02,  1.91392154e-02, -3.12244892e-02],
        ...,
        [ 4.87112775e-02,  3.86958420e-02, -6.46543950e-02, ...,
         -5.33494592e-01, -5.34186885e-02,  2.02614889e-01],
        [-2.57201493e-03, -1.51946023e-03,  1.91392154e-02, ...,
         -5.34186997e-02,  3.29292029e-01, -1.61749125e-03],
        [-2.61183828e-02,  1.26273595e-02, -3.12243849e-02, ...,
          2.02615038e-01, -1.61758065e-03,  4.61240351e-01]],

       [[-5.89127541e-02,  1.13666710e-02,  9.47502479e-02, ...,
          7.99134374e-02,  7.86826611e-02, -3.36419865e-02],
        [ 1.13666207e-02,  4.70400900e-02,  1.17738992e-02, ...,
          7.67384470e-03,  1.23783350e-02,  9.62841511e-03],
        [ 9.47502106e-02,  1.17739253e-02, -1.55037731e-01, ...,
          1.79256767e-01,  1.30410194e-01, -2.34648213e-02],
        ...,
        [ 7.99135491e-02,  7.67387077e-03,  1.79256901e-01, ...,
         -8.14906001e-01,  3.15824747e-02, -1.65081769e-01],
        [ 7.86829218e-02,  1.23785995e-02,  1.30410284e-01, ...,
          3.15824598e-02, -6.24578178e-01, -7.98818395e-02],
        [-3.36419716e-02,  9.62843001e-03, -2.34647840e-02, ...,
         -1.65081799e-01, -7.98816681e-02,  4.82486546e-01]],

       [[-6.49828494e-01, -6.95594326e-02,  4.64168787e-02, ...,
          4.40789834e-02,  3.72560471e-02,  2.21275330e-01],
        [-6.95594624e-02, -1.58568025e-01,  5.90233356e-02, ...,
         -7.43162632e-02,  1.23836696e-02,  1.29813194e-01],
        [ 4.64168489e-02,  5.90233356e-02, -1.32834032e-01, ...,
          2.27665603e-02, -3.74414027e-03,  1.11708164e-01],
        ...,
        [ 4.40790206e-02, -7.43162483e-02,  2.27665715e-02, ...,
         -7.52008915e-01, -2.43807435e-02,  7.79776812e-01],
        [ 3.72560471e-02,  1.23836622e-02, -3.74413282e-03, ...,
         -2.43806988e-02,  3.24047744e-01, -6.18820190e-02],
        [ 2.21275270e-01,  1.29812673e-01,  1.11708902e-01, ...,
          7.79776096e-01, -6.18820935e-02,  3.34143829e+00]],

       ...,

       [[-3.87993604e-02, -5.63476980e-03, -1.22786060e-01, ...,
```

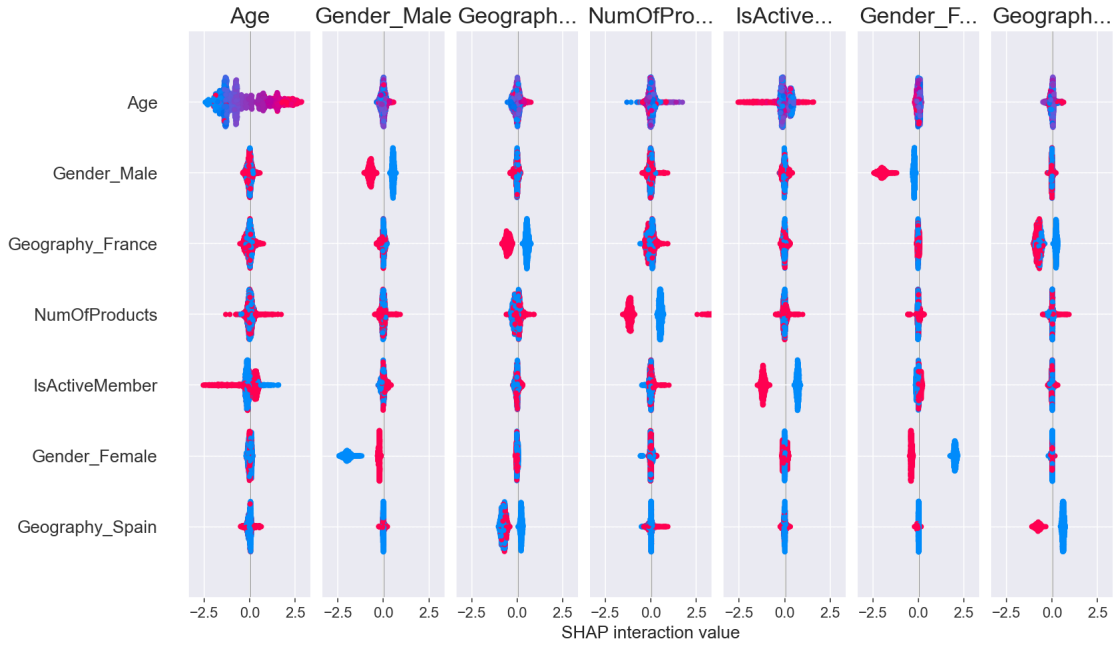
```
        -5.36407903e-02,  2.01304406e-02, -1.11176670e-01],
       [-5.63476980e-03, -4.58560511e-02, -1.80543140e-02, ...,
        -6.01031408e-02,  5.10337949e-03,  2.00942606e-02],
       [-1.22786053e-01, -1.80543587e-02,  2.90554017e-02, ...,
        -3.99993211e-02,  1.66174173e-02, -1.01180911e-01],
       ...,
       [-5.36407754e-02, -6.01031110e-02, -3.99993360e-02, ...,
        -3.59667063e-01, -6.57482445e-02,  2.07900360e-01],
       [ 2.01304406e-02,  5.10334224e-03,  1.66174173e-02, ...,
        -6.57482445e-02,  3.37009847e-01, -1.24450028e-03],
       [-1.11176610e-01,  2.00942382e-02, -1.01180926e-01, ...,
         2.07900316e-01, -1.24445558e-03,  5.11531293e-01]],
```

```
# takes a couple minutes since SHAP interaction values take a factor of 2 * # features
# more time than SHAP values to compute, since this is just an example we only explain
# the first 2,000 people in order to run quicker
shap_interaction_values = shap.TreeExplainer(model).shap_interaction_values(x.iloc[:2000,:])
```

## SHAP Interaction Value Summary Plot

A summary plot of a SHAP interaction value matrix plots a matrix of summary plots with the main effects on the diagonal and the interaction effects off the diagonal.

```
shap.summary_plot(shap_interaction_values, x.iloc[:2000,:])
```




## SHAP Interaction Value Dependence Plots

Running a dependence plot on the SHAP interaction values a allows us to separately observe the main effects and the interaction effects.

Below we plot the main effects for age and some of the interaction effects for age. It is informative to compare the main effects plot of age with the earlier SHAP value plot for age. The main effects plot has no vertical dispersion because the interaction effects are all captured in the off-diagonal terms.

```
shap.dependence_plot(
    ("Age", "Gender_Female"),
    shap_interaction_values, x.iloc[:2000,:]
    #display_features=x_display.iloc[:2000,:]
)
```

```
Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and will become an error two minor releases later. Plea
```


## Summary plot for churn customers




```
from matplotlib import colors as plt_colors

# class names
classes = ['Remained','Churned']

# set RGB tuple per class
colors = ['#4682b4','#b22222']

# get class ordering from shap values
class_inds = np.argsort([-np.abs(shap_values[i]).mean() for i in range(len(shap_values))])

# create listed colormap
cmap = plt_colors.ListedColormap(np.array(colors)[class_inds])


shap.summary_plot(shap_values, X_test, plot_type="bar", color=cmap, class_names=classes)
```

## Summary plot for each class - Remained and Churned

```
# set RGB tuple per class
colors = ['#4682b4','#b22222']

# get class ordering from shap values
#class_inds = np.argsort([-np.abs(shap_values[i]).mean() for i in range(len(shap_values))])

# create listed colormap
#cmap = plt_colors.ListedColormap(np.array(colors)[class_inds])

for i in range(len(classes)):
    plot = shap.summary_plot(shap_values, X_test,class_names=classes,class_inds=[i])
    print("Class: "+classes[i] +'  ' +str(i))



# class names
classes = ['Remained','Churned']

for i in range(len(classes)):
    print("Customer:"+classes[i])
    shap.summary_plot(shap_values, X_test)
```

Age : older people were likely to churn and younger people remained with the bank

Number of Products: if customers bought products i.e it is too high ( product no. 4, 3) or too low (product no. 1) then they are likely to churn whereas customers who bought product no. 2 are likely to increase the chances of continuing being a customer.

IsActiveMember: High feature value (red,1 ) indicates customer being active and will remain with the bank and those customers who are not active member are going to churn

```
X_output = X_test.copy()
X_output.loc[:,'predict if customer will churn'] = np.round(model.predict(X_output),2)

random_picks = np.arange(1,330,50)
S = X_output.iloc[random_picks]
S
```

```
shap.initjs()

def shap_plot(j):
    # compute SHAP values
    explainerModel = shap.TreeExplainer(model)
    shap_values_Model = explainerModel.shap_values(S)
    p = shap.force_plot(explainerModel.expected_value[j], shap_values_Model[j], S.iloc[[j]])
    return(p)


# mean of X train
X_train.mean()


# mean of Y test
y_test.mean()


shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[0,:], X_test.iloc[0,:])


shap.summary_plot(shap_values, X_test, plot_type="bar")


shap.force_plot(explainer.expected_value[0], shap_values[0], link='logit')


shap_plot(2)









X_test.loc[[0]]


y_test.loc[[0]]


#choosen_instance = X_train.loc[[0]]
#shap_values = explainer.shap_values(X_train.loc[[0]])
shap.initjs()
shap.force_plot(explainer.expected_value[0], shap_values[0])
```

The prediction value is 1.00 which is greater than the base value(0.5017) which imples that there is high predicted chance of customer churing.

Is not an active member pushes the prediction to churn

```
X_test.loc[[421]]









shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[0,:], X_train.iloc[0,:])
```