In [81]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [82]:
```
churn_data = pd.read_csv('C:\\Users\\Neeraj\\Downloads\\Churn_
```

In [58]:
```
churn_data.head()
```

Out[58]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Ge
0	1	15634602	Hargrave	619	France	Fe
1	2	15647311	Hill	608	Spain	Fe
2	3	15619304	Onio	502	France	Fe
3	4	15701354	Boni	699	France	Fe
4	5	15737888	Mitchell	850	Spain	Fe

In [59]:
```
churn_data.columns.values
```

Out[59]:
```
array(['RowNumber', 'CustomerId', 'Surname', 'CreditScore',
'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'], dtype=
object)
```

In [60]:
```
churn_data.isnull().sum()
```

Out[60]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [61]:
```
churn_data = churn_data.drop('RowNumber',axis =1)
```

In [62]:
```
churn_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CustomerId       10000 non-null  int64  
 1   Surname          10000 non-null  object 
 2   CreditScore      10000 non-null  int64  
 3   Geography        10000 non-null  object 
 4   Gender           10000 non-null  object 
 5   Age              10000 non-null  int64  
 6   Tenure           10000 non-null  int64  
 7   Balance          10000 non-null  float64
 8   NumOfProducts    10000 non-null  int64  
 9   HasCrCard        10000 non-null  int64  
 10  IsActiveMember   10000 non-null  int64  
 11  EstimatedSalary  10000 non-null  float64
 12  Exited           10000 non-null  int64  
dtypes: float64(2), int64(8), object(3)
memory usage: 1015.8+ KB
```

In [63]:
```
churn_data['Gender'].unique
```

Out[63]:
```
<bound method Series.unique of 0       Female
1       Female
2       Female
3       Female
4       Female
         ...
9995      Male
9996      Male
9997    Female
9998      Male
9999    Female
Name: Gender, Length: 10000, dtype: object>
```

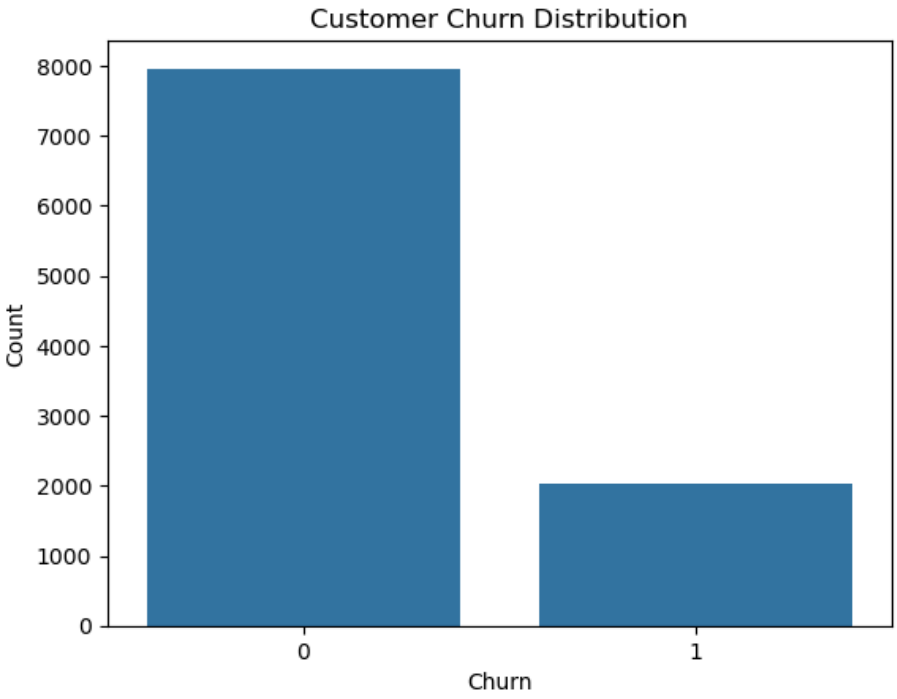
In [64]:
```
churn_data['Gender'] = churn_data['Gender'].replace({'Male' :
```

In [65]:
```
churn_data.info()
```

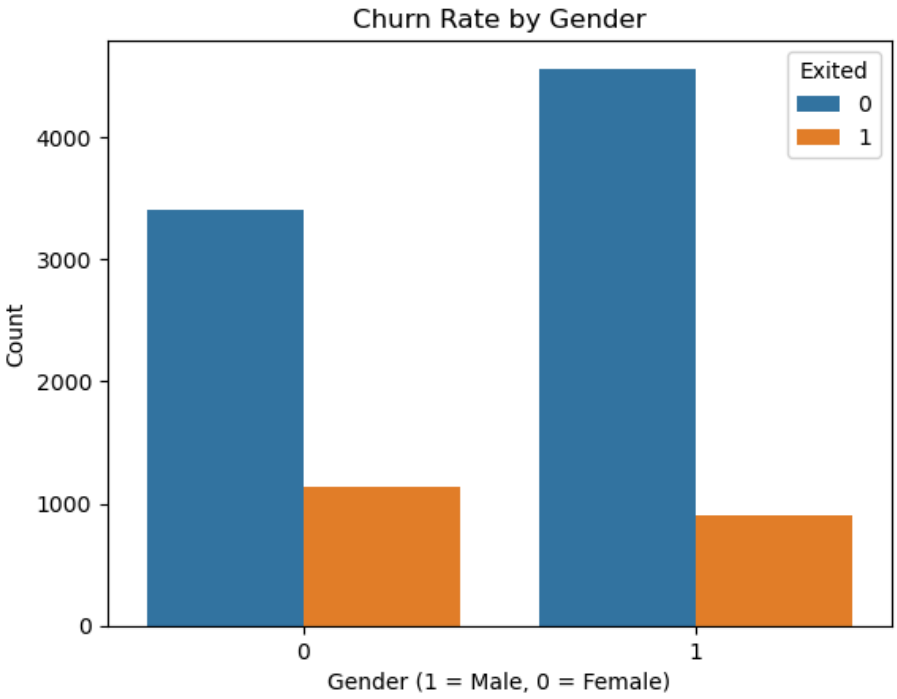
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CustomerId       10000 non-null  int64  
 1   Surname          10000 non-null  object 
 2   CreditScore      10000 non-null  int64  
 3   Geography        10000 non-null  object 
 4   Gender           10000 non-null  int64  
 5   Age              10000 non-null  int64  
 6   Tenure           10000 non-null  int64  
 7   Balance          10000 non-null  float64
 8   NumOfProducts    10000 non-null  int64  
 9   HasCrCard        10000 non-null  int64  
 10  IsActiveMember   10000 non-null  int64  
 11  EstimatedSalary  10000 non-null  float64
 12  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(2)
memory usage: 1015.8+ KB
```

In [66]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

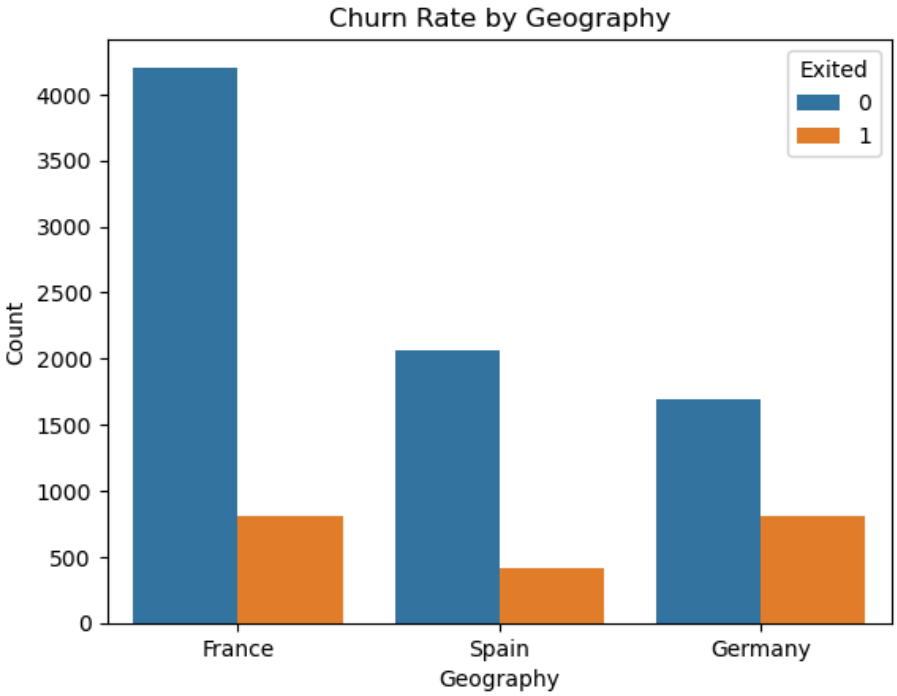
In [67]:
```
plt1 = sns.countplot(data = churn_data, x = 'Exited')
plt.title('Customer Churn Distribution')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
```



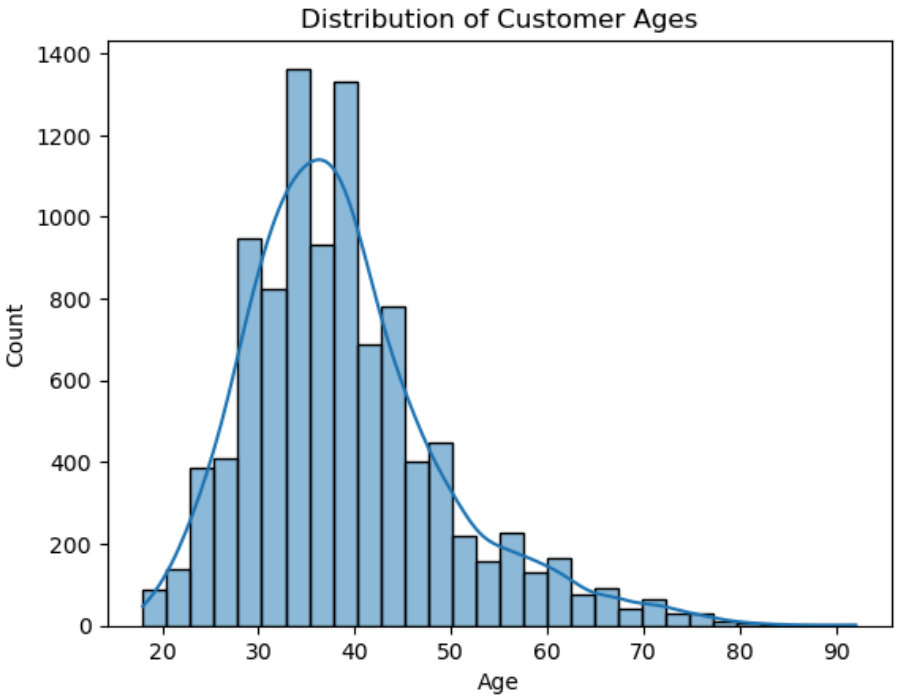
In [68]:
```
sns.countplot(x='Gender',hue = 'Exited', data = churn_data)
plt.title('Churn Rate by Gender')
plt.xlabel('Gender (1 = Male, 0 = Female)')
plt.ylabel('Count')
plt.show()
```



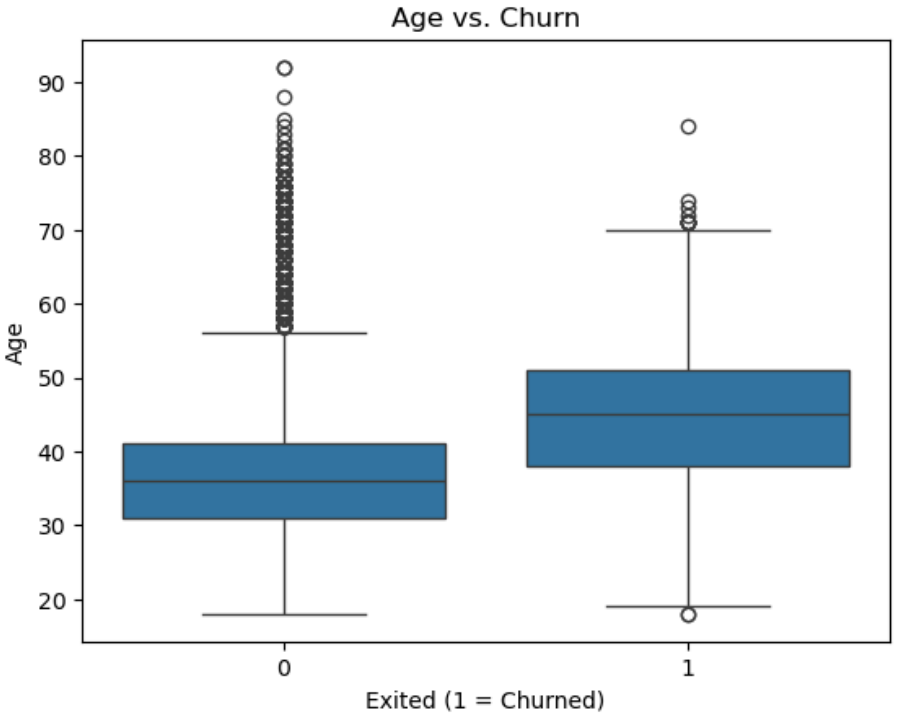
In [69]:
```
sns.countplot(x='Geography', hue='Exited', data=churn_data)
plt.title("Churn Rate by Geography")
plt.ylabel("Count")
plt.show()
```



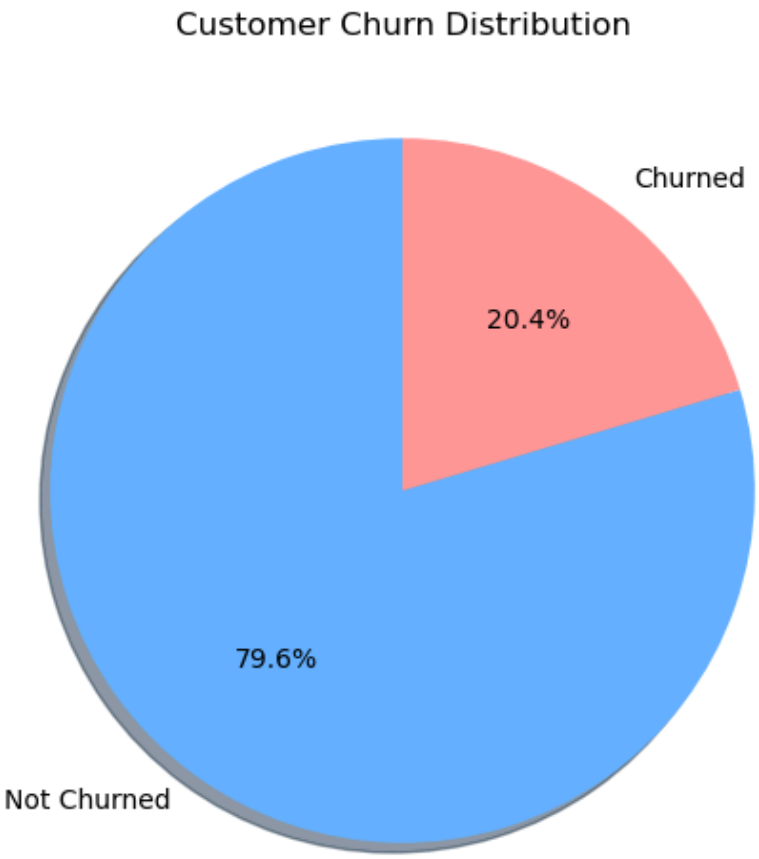
In [70]:
```
sns.histplot(churn_data['Age'], bins=30, kde=True)
plt.title("Distribution of Customer Ages")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```



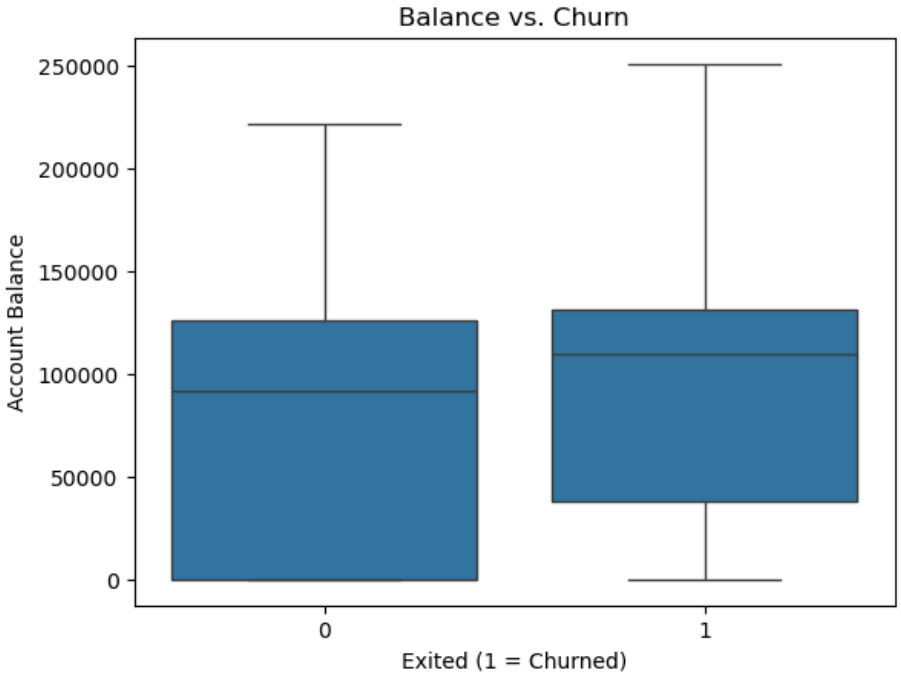
In [71]:
```
sns.boxplot(x='Exited', y='Age', data=churn_data)
plt.title("Age vs. Churn")
plt.xlabel("Exited (1 = Churned)")
plt.ylabel("Age")
plt.show()
```



In [72]:
```
import matplotlib.pyplot as plt
churn_counts = churn_data['Exited'].value_counts()
labels = ['Not Churned', 'Churned']
colors = ['#66b3ff', '#ff9999']
plt.figure(figsize=(6,6))
plt.pie(churn_counts, labels=labels, autopct='%1.1f%%', starta
plt.title('Customer Churn Distribution')
plt.show()
```

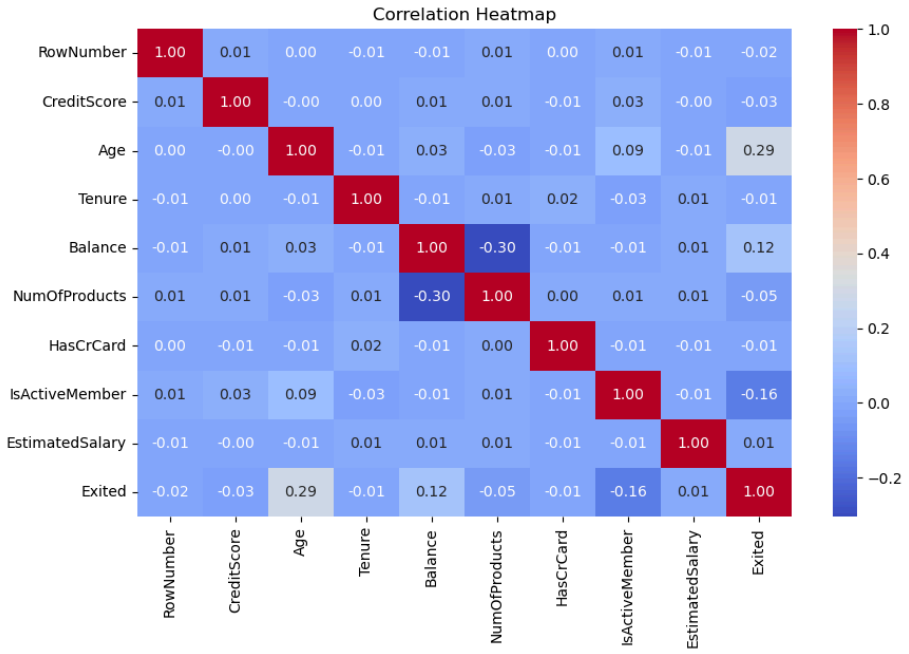


In [73]:
```
sns.boxplot(x='Exited', y='Balance', data=churn_data)
plt.title("Balance vs. Churn")
plt.xlabel("Exited (1 = Churned)")
plt.ylabel("Account Balance")
plt.show()
```

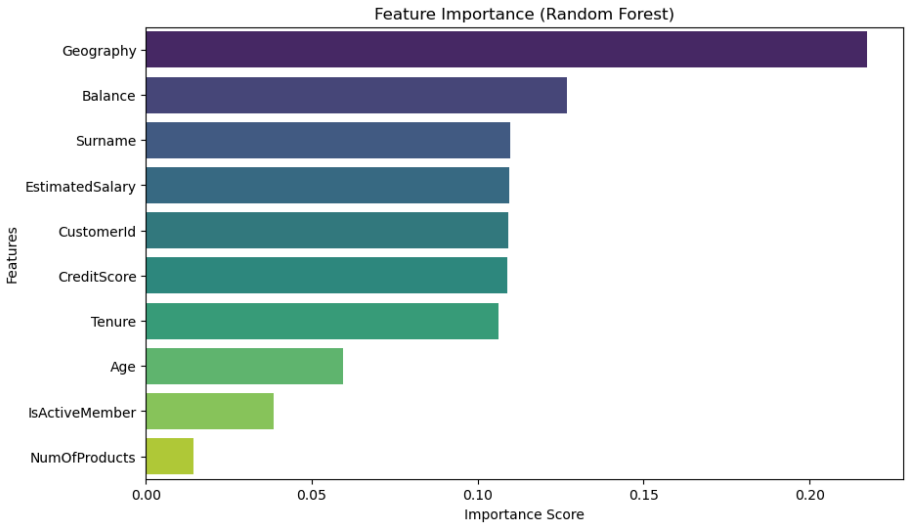


In [83]:
```
churn_data_new = churn_data.drop(['Surname','Gender','Customer
plt.figure(figsize=(10,6))
sns.heatmap(churn_data_new.corr(), annot=True, cmap='coolwarm'
```

```
plt.title("Correlation Heatmap")
plt.show()
```

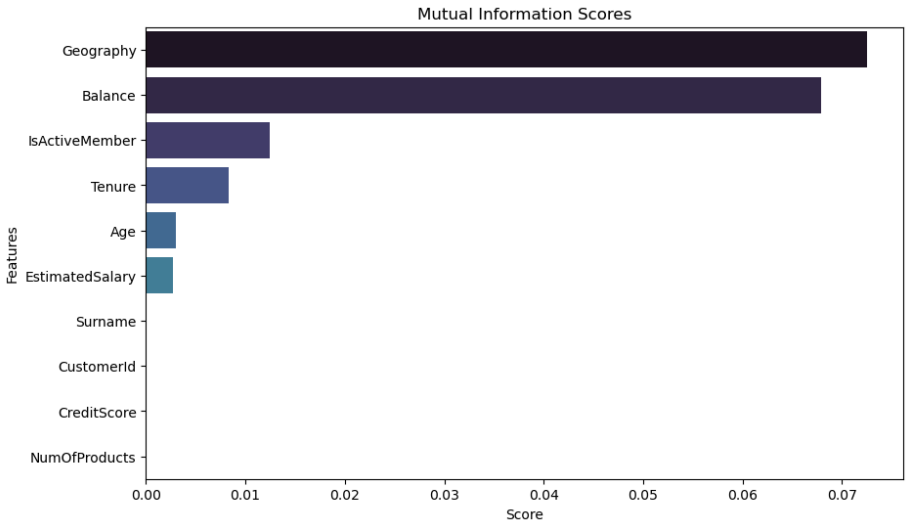


In [86]:
```
from sklearn.ensemble import RandomForestClassifier
x1 = churn_data.drop(['Exited','Geography','Gender','Surname']
y1 = churn_data['Exited']
model = RandomForestClassifier(random_state=42)
model.fit(x1, y1)
importance = pd.Series(model.feature_importances_, index=X.col
importance.sort_values(ascending=False, inplace=True)
plt.figure(figsize=(10,6))
sns.barplot(x=importance, y=importance.index, palette='viridis
plt.title('Feature Importance (Random Forest)')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```



In [87]:
```
from sklearn.feature_selection import mutual_info_classif
x2 = churn_data.drop(['Exited','Geography','Gender','Surname']
y2 = churn_data_new['Exited']
mi_scores = mutual_info_classif(x2, y2, discrete_features='aut
mi_scores = pd.Series(mi_scores, index=X.columns).sort_values(
plt.figure(figsize=(10,6))
```

```
sns.barplot(x=mi_scores, y=mi_scores.index, palette='mako')
plt.title('Mutual Information Scores')
plt.xlabel('Score')
plt.ylabel('Features')
plt.show()
```



In [88]:
```
churn_data.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
```

In [89]:
```
churn_data = churn_data.drop(['HasCrCard','Gender','Surname','
```

In [90]:
```
churn_data.shape
```

Out[90]:
```
(10000, 9)
```

In [94]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
df = churn_data.copy()
le = LabelEncoder()
df['Geography'] = le.fit_transform(df['Geography'])
```

In [95]:
```
X = df.drop('Exited',axis = 1)
Y = df['Exited']
```

In [96]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size
```

In [97]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_ma
```

In [98]:
```
rfc = RandomForestClassifier()
rfc.fit(X_train,Y_train)
```

Out[98]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [99]:
```
predictions = rfc.predict(X_test)
```

In [101…
```
print(classification_report(Y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      2378
           1       0.79      0.50      0.61       622

    accuracy                           0.87      3000
   macro avg       0.84      0.73      0.77      3000
weighted avg       0.86      0.87      0.86      3000
```

In [102…
```
print(confusion_matrix(Y_test,predictions))
```

```
[[2297   81]
 [ 314  308]]
```

In [103…
```
print(accuracy_score(Y_test,predictions))
```

```
0.8683333333333333
```

In [104…
```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
```

In [109…
```
param_grid = {
    'max_depth' : [3,8,1],
    'min_samples_leaf' : range(1,3,4),
    'min_samples_split' : range(2,5,1),
    'n_estimators'      : [5,10,5],
    'max_features'      : [5,10,1]
}
rf = RandomForestClassifier()
grid_search = GridSearchCV(estimator = rf,param_grid=param_gri
```

In [110…
```
grid_search.fit(X_train,Y_train)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
```

Out[110…

GridSearchCV

best_estimator_:
RandomForestClassifier

RandomForestClassifier

In [130…
```
rfc = RandomForestClassifier(bootstrap=True,
                             max_depth = 5,
                             min_samples_split=4,
                             max_features = 5,
                             n_estimators = 9)
```

In [131…
```
rfc.fit(X_train,Y_train)
```

Out[131…

RandomForestClassifier

```
RandomForestClassifier(max_depth=5, max_features=5, mi
n_samples_split=4,
                       n_estimators=9)
```

In [132…
```
prediction = rfc.predict(X_test)
```

In [133…
```
print(confusion_matrix(Y_test,prediction))
```

```
[[2317   61]
 [ 363  259]]
```

In [134…
```
print(accuracy_score(Y_test,prediction))
```

```
0.8586666666666667
```

In [120…
```
from xgboost import XGBClassifier
```

In [123…
```
model2 = XGBClassifier()
model2.fit(X_train,Y_train)
```

Out[123…

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callba
cks=None,
              colsample_bylevel=None, colsample_byn
ode=None,
              colsample_bytree=None, device=None, e
arly_stopping_rounds=None,
              enable_categorical=False, eval_metric
=None, feature_types=None,
              feature_weights=None, gamma=None, gro
```

In [124…
```
predictions2 = model2.predict(X_test)
```

In [128…
```
print(confusion_matrix(Y_test,predictions2))
```

```
[[2240  138]
 [ 299  323]]
```

print(accuracy_score(Y_test,predictions2))

In [ ]:

In [ ]: