Customer Churn prediction is predicting which customers are at high risk of leaving your company or canceling a subscription to a service, based on their behavior with your product.For many companies, this is an important prediction. This is because acquiring new customers often costs more than retaining existing ones. Once you've identified customers at risk of churn, you need to know exactly what marketing efforts you should make with each customer to maximize their likelihood of staying.

Our model to build will predict bank customer churn in which we use Classification Machine learning models.

The dataset we will use consists of the following features:-

1. RowNumber : row number of the data
2. CustomerId : Bank Id of the customer
3. Surname: Customer's surname
4. CreditScore: the credit score of the customer
5. Geography: location of customer
6. Gender: whether the customer is male or female
7. Age: the age of the customer
8. Tenure: From how many years customer is in bank
9. Balance: Average balance of customer
10. NumOfProducts: Number of bank product facilities customer is using
11. HasCrCard : Whether the customer has a credit card or not
12. IsActiveMember: whether the customer is active or not
13. EstimatedSalary: the expected salary of the customer
14. Exited: Whether the customer left or not

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler

import warnings
warnings.filterwarnings("ignore")

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import TomekLinks
from collections import Counter
from imblearn.over_sampling import SMOTE

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion
```

Reading Data

In [2]:

```
df = pd.read_csv("/Users/HP/Desktop/Churn_Modelling.csv")
df
```

Out[2]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProdu

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProdu
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	
...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	

10000 rows × 14 columns

In [3]:

```
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
```

In [4]:

```
df.describe()
```

Out[4]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	Has(
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	Has(
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1

Checking for nulls & duplicates

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
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 [6]:

```
df.duplicated().sum()
```

Out[6]:

```
0
```

Data Cleaning

In [7]:

```
df.columns
```

Out[7]:

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

- We can find that columns such as Rownumber, CustomerID and Surname are useless data as it won't affect our model performance so we can drop them.

In [8]:

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

Out[8]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estin
0	619	France	Female	42	2	0.00	1	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	1	
2	502	France	Female	42	8	159660.80	3	1	0	
3	699	France	Female	39	1	0.00	2	0	0	
4	850	Spain	Female	43	2	125510.82	1	1	1	

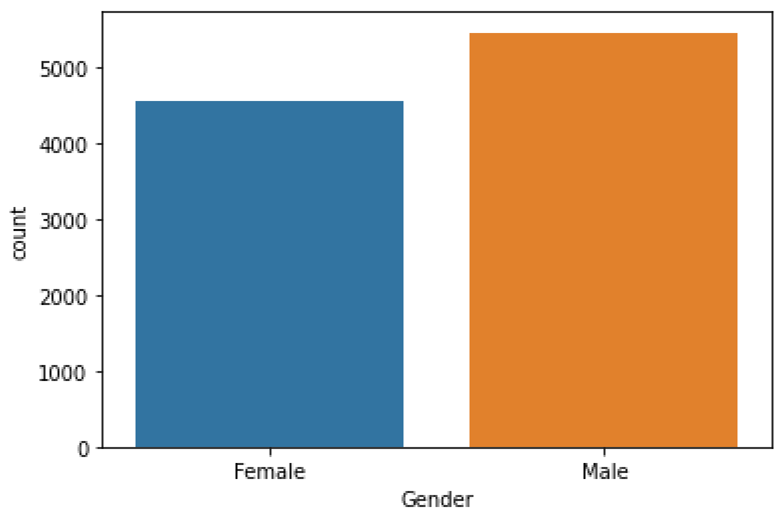
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estir
...	...	...	...	...	...	...	...	...	...	
9995	771	France	Male	39	5	0.00	2	1	0	
9996	516	France	Male	35	10	57369.61	1	1	1	
9997	709	France	Female	36	7	0.00	1	0	1	
9998	772	Germany	Male	42	3	75075.31	2	1	0	
9999	792	France	Female	28	4	130142.79	1	1	0	

10000 rows × 11 columns

EDA

In [9]:
```
sns.countplot(x = df['Gender'])
```

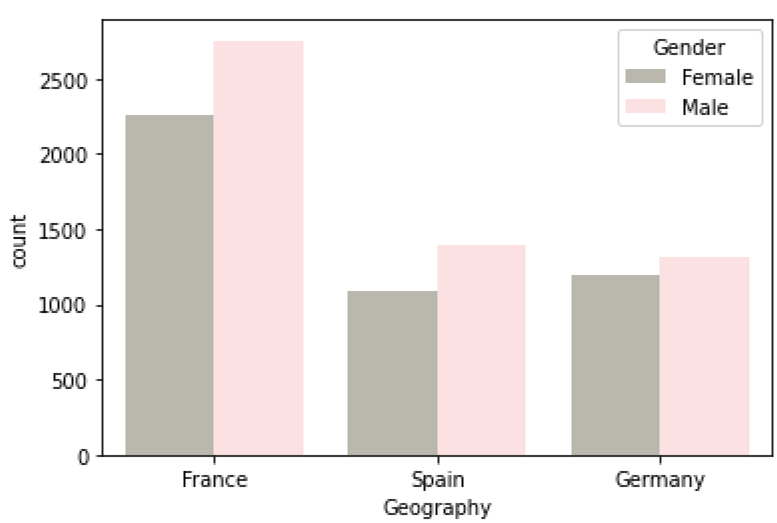
Out[9]:
```
<Axes: xlabel='Gender', ylabel='count'>
```



- We can find that is a slight little difference in the numbers of Males and females

In [10]:
```
sns.countplot( data = df , x= df['Geography'] , hue= df['Gender'], palette=["#bcbaaa", "#f
```

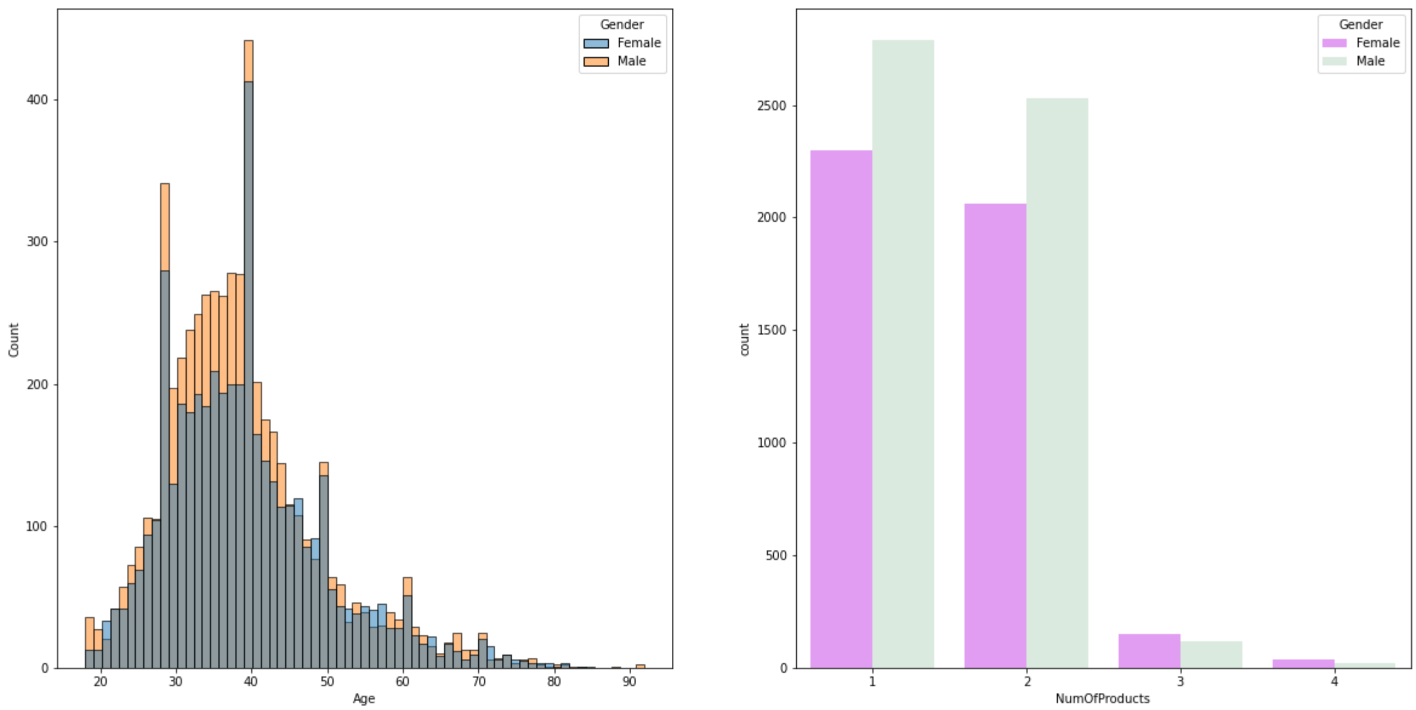
Out[10]:
```
<Axes: xlabel='Geography', ylabel='count'>
```



- We can find that in the respective countries that we have which are France, Spain and Germany, that there are more males than females customers in the bank in the three countries.

In [11]:
```
plt.figure(figsize = (20,10))

# subplot 1
plt.subplot(1, 2, 1)
sns.histplot(x = df['Age'] ,hue = df['Gender'])

# subplot 2
plt.subplot(1, 2, 2)
sns.countplot(x = df['NumOfProducts'] , data = df, hue = df['Gender'] , palette=["#ea8fff'
```

Out[11]:
```
<Axes: xlabel='NumOfProducts', ylabel='count'>
```



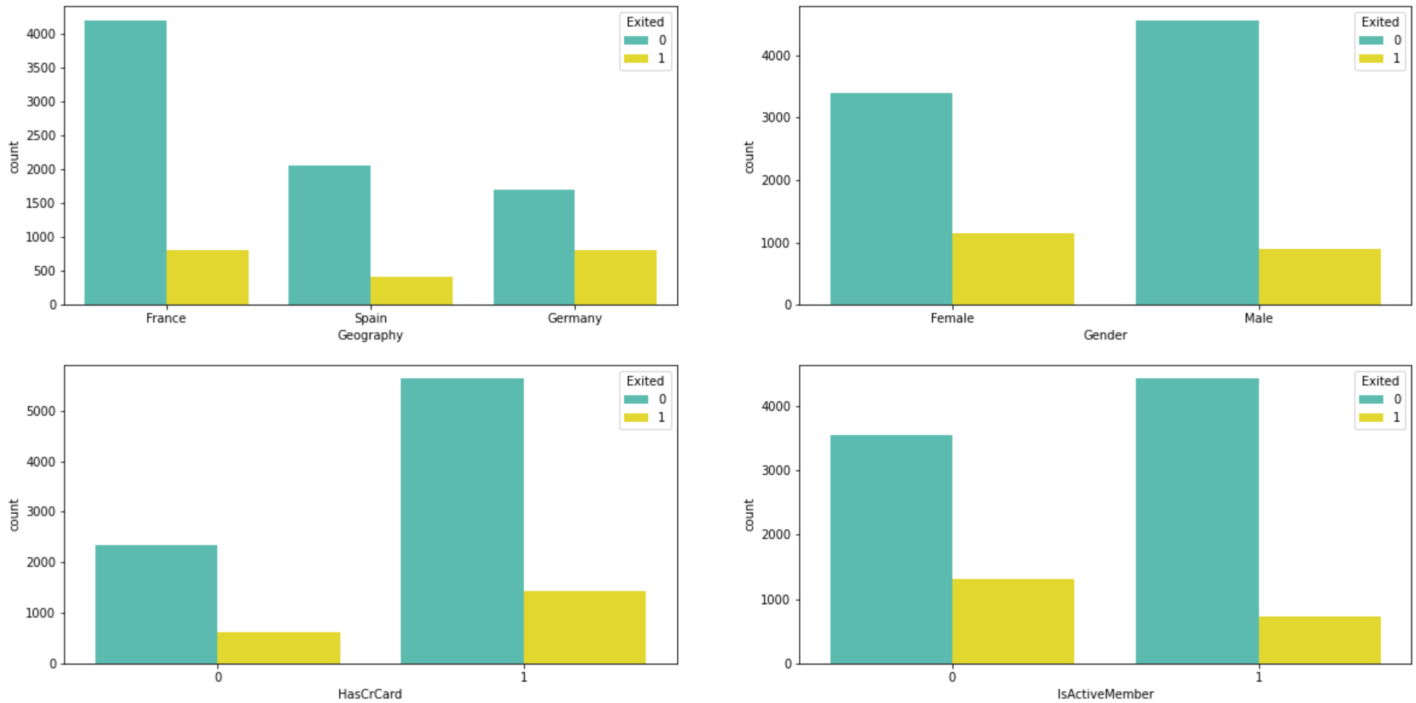
- Majority of the customers age range from around 28 years to 45 years with Males representing the most in this range
- there is major decrease in the customers count from ages of range 50 to 85
- Most of the customers have 1 or 2 products in which they use from the bank with males being higher than females

In [12]:
```
plt.figure(figsize = (20,10))

# subplot 1
plt.subplot(2, 2, 1)
sns.countplot(x='Geography', hue = 'Exited',data = df , palette = ["#4ccbbb", "#fff111"])

# subplot 2
plt.subplot(2, 2, 2)
sns.countplot(x = 'Gender', hue = 'Exited',data = df, palette = ["#4ccbbb", "#fff111"])

# subplot 3
plt.subplot(2, 2, 3)
sns.countplot(x = 'HasCrCard', hue = 'Exited',data = df, palette = ["#4ccbbb", "#fff111"])

# subplot 4
```

```
plt.subplot(2, 2, 4)
sns.countplot(x = 'IsActiveMember', hue = 'Exited',data = df, palette = ["#4ccbbb", "#fff
```

Out[12]:

```
<Axes: xlabel='IsActiveMember', ylabel='count'>
```



- We find that the number of customers leaving in germany and france is higher than in spain
- we find too that number of customers staying in france is higher than in germany and spain by alot
- More Females left the bank than males and more males stayed as customers in the bank than females
- Number of customers left while having a credit card is higher than customers left without having one
- Number of customers stayed while having credit card is higher than customers stayed without having one
- Number of customers left the bank with not being active is higher than customers left while being active
- Number of customers stayed with being active is higher than customers stayed without being active

Encoding of categorical features (Geography & Gender)

In [13]:

```
df['Geography'].unique()
```

Out[13]:

```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [14]:

```
df['Geography'].value_counts()
```

Out[14]:

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [15]:

```
pd.get_dummies(df['Geography'], drop_first = False)
```

Out[15]:

	France	Germany	Spain
0	1	0	0

	France	Germany	Spain
1	0	0	1
2	1	0	0
3	1	0	0
4	0	0	1
...	...	...	...
9995	1	0	0
9996	1	0	0
9997	1	0	0
9998	0	1	0
9999	1	0	0

10000 rows × 3 columns

- This feature is a nominal one which is best dealt with one hot encoding

In [16]:

```
df = df.join(pd.get_dummies(df['Geography'], drop_first = False)).drop(['Geography'], axis
df
```

Out[16]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	Female	42	2	0.00	1	1	1	101348.88
1	608	Female	41	1	83807.86	1	0	1	112542.58
2	502	Female	42	8	159660.80	3	1	0	113931.57
3	699	Female	39	1	0.00	2	0	0	93826.63
4	850	Female	43	2	125510.82	1	1	1	79084.10
...	...	...	...	...	...	...	...	...	...
9995	771	Male	39	5	0.00	2	1	0	96270.64
9996	516	Male	35	10	57369.61	1	1	1	101699.77
9997	709	Female	36	7	0.00	1	0	1	42085.58
9998	772	Male	42	3	75075.31	2	1	0	92888.52
9999	792	Female	28	4	130142.79	1	1	0	38190.78

10000 rows × 13 columns

In [17]:

```
df['Gender'].unique()
```

Out[17]:

```
array(['Female', 'Male'], dtype=object)
```

In [18]:

```
df['Gender'] = df['Gender'].map({'Male':1 , 'Female':0})
df['Gender'].unique()
```

Out[18]:

```
array([0, 1], dtype=int64)
```

- replaced the gender values of (male,female) with values of (1 for male , 0 for female)

In [19]:
```python
print(df['Gender'].value_counts())
df
```

```
1    5457
0    4543
Name: Gender, dtype: int64
```

Out[19]:

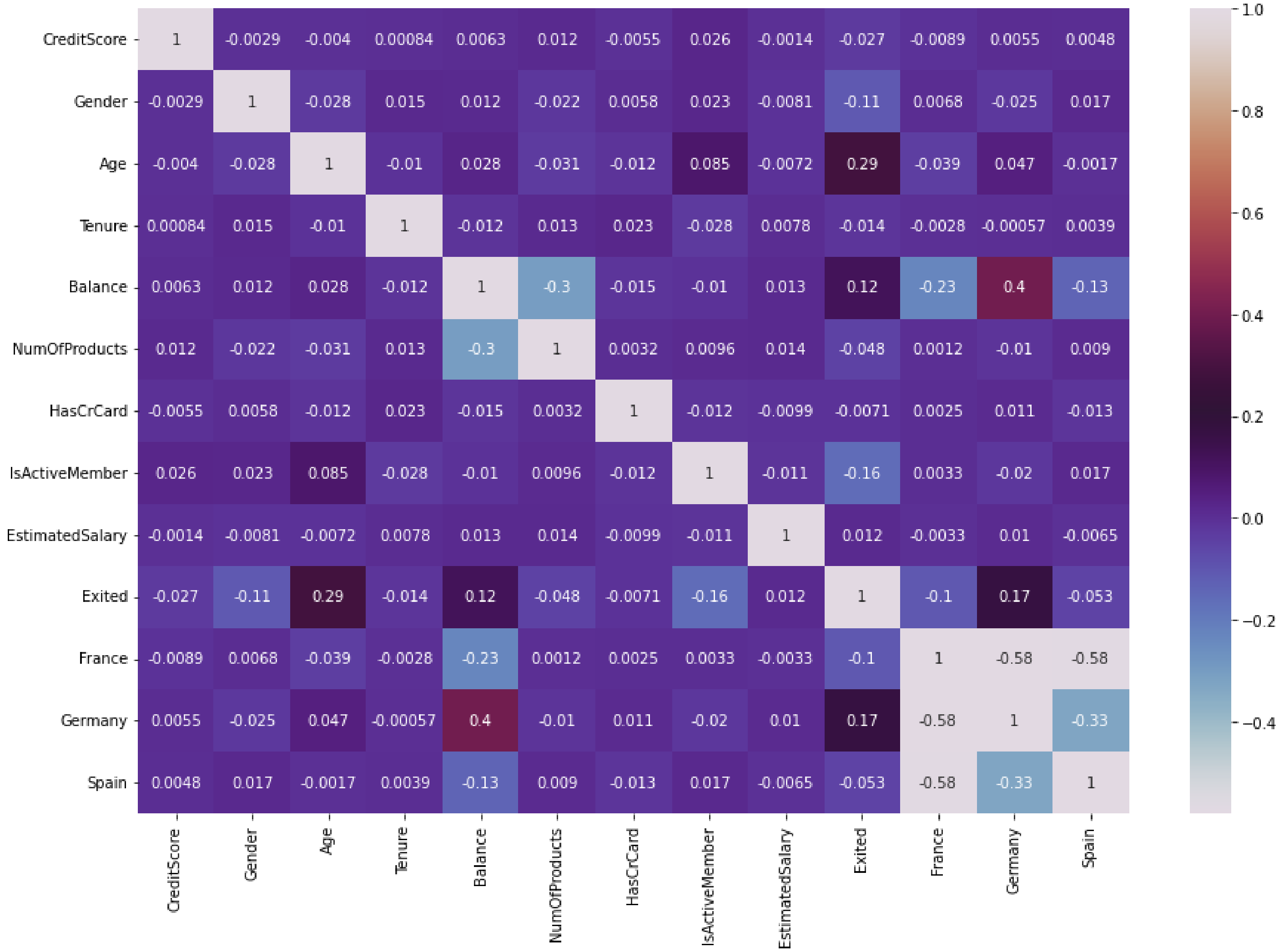
	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	0	42	2	0.00	1	1	1	101348.88
1	608	0	41	1	83807.86	1	0	1	112542.58
2	502	0	42	8	159660.80	3	1	0	113931.57
3	699	0	39	1	0.00	2	0	0	93826.63
4	850	0	43	2	125510.82	1	1	1	79084.10
...	...	...	...	...	...	...	...	...	...
9995	771	1	39	5	0.00	2	1	0	96270.64
9996	516	1	35	10	57369.61	1	1	1	101699.77
9997	709	0	36	7	0.00	1	0	1	42085.58
9998	772	1	42	3	75075.31	2	1	0	92888.52
9999	792	0	28	4	130142.79	1	1	0	38190.78

10000 rows × 13 columns

Correlation

In [20]:
```python
corr = df.corr()
plt.figure(figsize = (15,10))
sns.heatmap(corr , cmap = 'twilight', annot = True)
```

Out[20]:
```
<Axes: >
```



- We can find that there is no features that is heavily correlated to each other
- This is great for our performance of the model
- No feature selection is needed

Splitting of Data

In [21]:

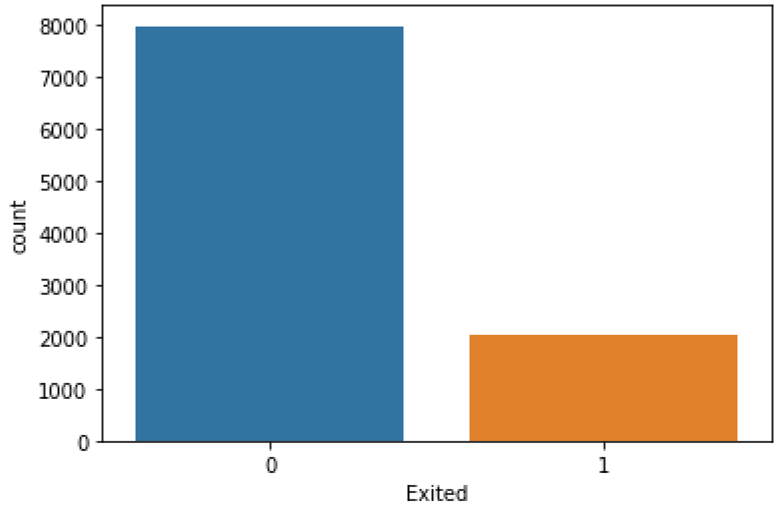
```
x = df.drop(['Exited'], axis = 1)
y = df['Exited']
```

Check for data imbalancement

In [22]:

```
sns.countplot(x = df['Exited'])
print(Counter(y))
```

```
Counter({0: 7963, 1: 2037})
```



- We find that the number of people to leave the bank is too low compared to number of people to stay in the bank
- we can solve the problem by applying SMOTE or RandomOverSampler.

Handling imbalanced data

In [23]:
```
Counter(y)
```
Out[23]:
```
Counter({1: 2037, 0: 7963})
```

In [24]:
```
smote = SMOTE(sampling_strategy = 'minority', k_neighbors = 4)
x,y = smote.fit_resample(x,y)
```

In [25]:
```
Counter(y)
```
Out[25]:
```
Counter({1: 7963, 0: 7963})
```

- Now, it is solved so we can do the train test split.

Train test Split & Scaling

In [26]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, random_state =42
```

In [27]:
```
scaler = StandardScaler()

x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

- We just scaled the the input features
- We applied a fit_transform for the x_train
- For the x_test, we just applied a transform function

Bagging (with KNN, DT, Logistic Reg.)

In [28]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

In [29]:
```
tree_clf = DecisionTreeClassifier()
log_clf = LogisticRegression()
knn_clf = KNeighborsClassifier()
```

In [30]:
```
tree_bag = BaggingClassifier(
    base_estimator = tree_clf,
    n_estimators = 500,
    bootstrap = True,
    n_jobs = -1,
    random_state = 42
)
log_bag = BaggingClassifier(
    base_estimator = log_clf,
    n_estimators = 500,
    bootstrap = True,
    n_jobs = -1,
    random_state = 42
)
knn_bag = BaggingClassifier(
    base_estimator = knn_clf,
    n_estimators = 500,
    bootstrap = True,
    n_jobs = -1,
    random_state = 42
)
```

Bagging (with KNN, DT, Logistic Reg.) Evaluation

In [31]:
```
#tree
tree_bag.fit(x_train,y_train)
tree_pred = tree_bag.predict(x_test)

#logstic
log_bag.fit(x_train,y_train)
log_pred = log_bag.predict(x_test)

#knn
knn_bag.fit(x_train,y_train)
knn_pred = knn_bag.predict(x_test)
```

In [32]:
```
print(tree_pred)
print('***********')
print(log_pred)
print('***********')
print(knn_pred)
```

```
[0 1 0 ... 0 1 1]
***********
[0 1 0 ... 1 1 1]
***********
[0 1 0 ... 0 1 1]
```

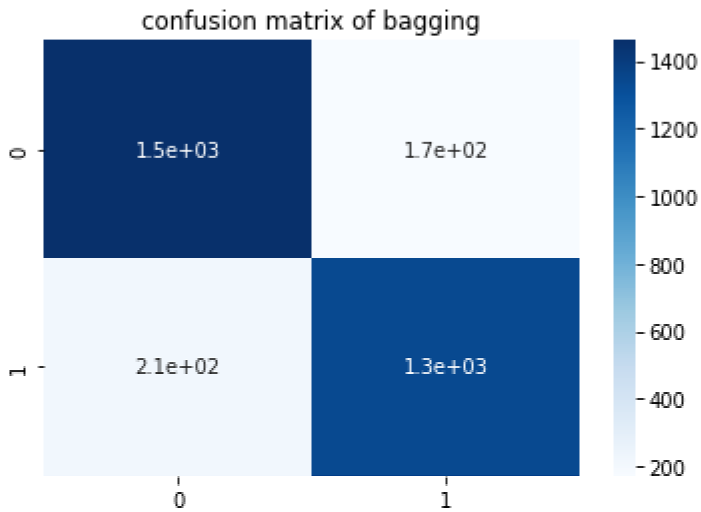
In [33]:

```python
# as it is classification we will do majority voting

final_pred = ((tree_pred + log_pred + knn_pred) / 3)
final_pred
```

Out[33]:
```
array([0.        , 1.        , 0.        , ..., 0.33333333, 1.        ,
       1.        ])
```

In [34]:
```python
cnf_mat = confusion_matrix(y_test,final_pred.round())
sns.heatmap( cnf_mat , annot = True , cmap = 'Blues')
plt.title('confusion matrix of bagging')
```

Out[34]:
```
Text(0.5, 1.0, 'confusion matrix of bagging')
```



In [36]:
```python
accuracy_bag = accuracy_score(y_test, final_pred.round())
print(f'the accuracy of the bagging model is = {accuracy_bag*100} %')
recall = recall_score(y_test, final_pred.round())
print(f'the recall of the bagging model is = {recall * 100} %')
precision = precision_score(y_test, final_pred.round())
print(f'the precision of the bagging model is = {precision * 100} %')
f1 = f1_score(y_test, final_pred.round())
print(f'the f1_score of the bagging model is = {f1 * 100} %')
```

```
the accuracy of the bagging model is = 87.88449466415568 %
the recall of the bagging model is = 86.22021893110109 %
the precision of the bagging model is = 88.6168100595632 %
the f1_score of the bagging model is = 87.40208877284596 %
```

In [37]:
```python
report = classification_report(y_test, final_pred.round())
print(report)
```

```
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      1633
           1       0.89      0.86      0.87      1553

    accuracy                           0.88      3186
   macro avg       0.88      0.88      0.88      3186
weighted avg       0.88      0.88      0.88      3186
```

Random Forest

In [38]:
```python
from sklearn.ensemble import RandomForestClassifier
```

In [39]:

```
rf = RandomForestClassifier(
    n_estimators = 500,
    bootstrap = True,
    n_jobs = -1,
    random_state =42
)

rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
```

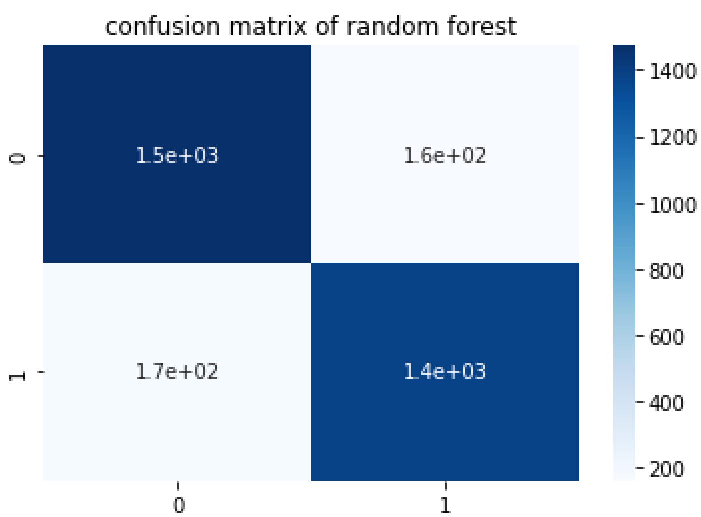
Random Forest Evaluation

In [40]:

```
cnf_mat = confusion_matrix(y_test,y_pred)
sns.heatmap( cnf_mat , annot = True , cmap = 'Blues')
plt.title('confusion matrix of random forest')
```

Out[40]:

```
Text(0.5, 1.0, 'confusion matrix of random forest')
```



In [41]:

```
accuracy_random = accuracy_score(y_test, y_pred)
print(f'the accuracy of the random forest model is = {accuracy_random * 100} %')
recall = recall_score(y_test, y_pred)
print(f'the recall of the random forest model is = {recall * 100} %')
precision = precision_score(y_test, y_pred)
print(f'the precision of the random forest model is = {precision * 100} %')
f1 = f1_score(y_test, y_pred)
print(f'the f1_score of the random forest model is = {f1 * 100} %')
```

```
the accuracy of the random forest model is = 89.51663527934714 %
the recall of the random forest model is = 88.79587894397939 %
the precision of the random forest model is = 89.60363872644575 %
the f1_score of the random forest model is = 89.19793014230272 %
```

In [42]:

```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.90      1633
           1       0.90      0.89      0.89      1553

    accuracy                           0.90      3186
   macro avg       0.90      0.89      0.90      3186
```

```
weighted avg       0.90      0.90      0.90      3186
```

Adaboost

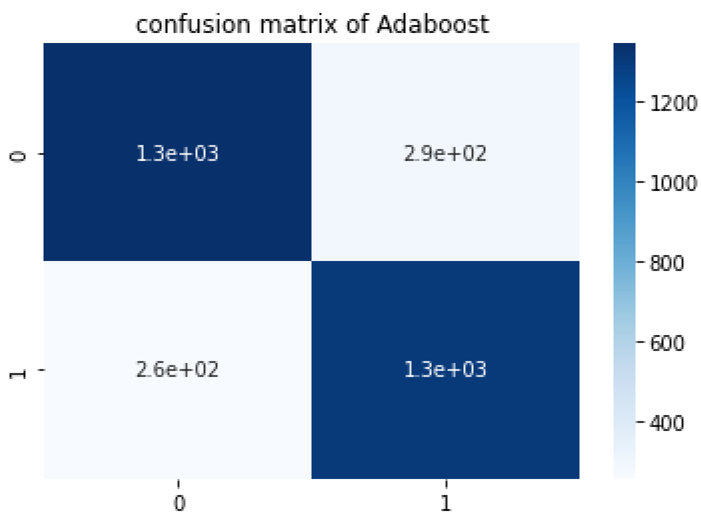
In [43]:
```python
from sklearn.ensemble import AdaBoostClassifier
```

In [44]:
```python
Adaboost = AdaBoostClassifier(
    base_estimator = DecisionTreeClassifier(),
    n_estimators = 200,
    learning_rate = 0.5,
    random_state = 42
)

Adaboost.fit(x_train,y_train)
y_pred = Adaboost.predict(x_test)
```

Adaboost Evaluation

In [45]:
```python
cnf_mat = confusion_matrix(y_test,y_pred)
sns.heatmap( cnf_mat , annot = True , cmap = 'Blues')
plt.title('confusion matrix of Adaboost')
```

Out[45]:
```
Text(0.5, 1.0, 'confusion matrix of Adaboost')
```



In [46]:
```python
accuracy_boost = accuracy_score(y_test, y_pred)
print(f'the accuracy of the Adaboost model is = {accuracy_boost * 100} %')
recall = recall_score(y_test, y_pred)
print(f'the recall of the Adaboost model is = {recall * 100} %')
precision = precision_score(y_test, y_pred)
print(f'the precision of the Adaboost model is = {precision * 100} %')
f1 = f1_score(y_test, y_pred)
print(f'the f1_score of the Adaboost model is = {f1 * 100} %')
```

```
the accuracy of the Adaboost model is = 82.86252354048965 %
the recall of the Adaboost model is = 83.51577591757888 %
the precision of the Adaboost model is = 81.72652804032766 %
the f1_score of the Adaboost model is = 82.61146496815286 %
```

In [47]:
```python
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83      1633
           1       0.82      0.84      0.83      1553

    accuracy                           0.83      3186
   macro avg       0.83      0.83      0.83      3186
weighted avg       0.83      0.83      0.83      3186
```

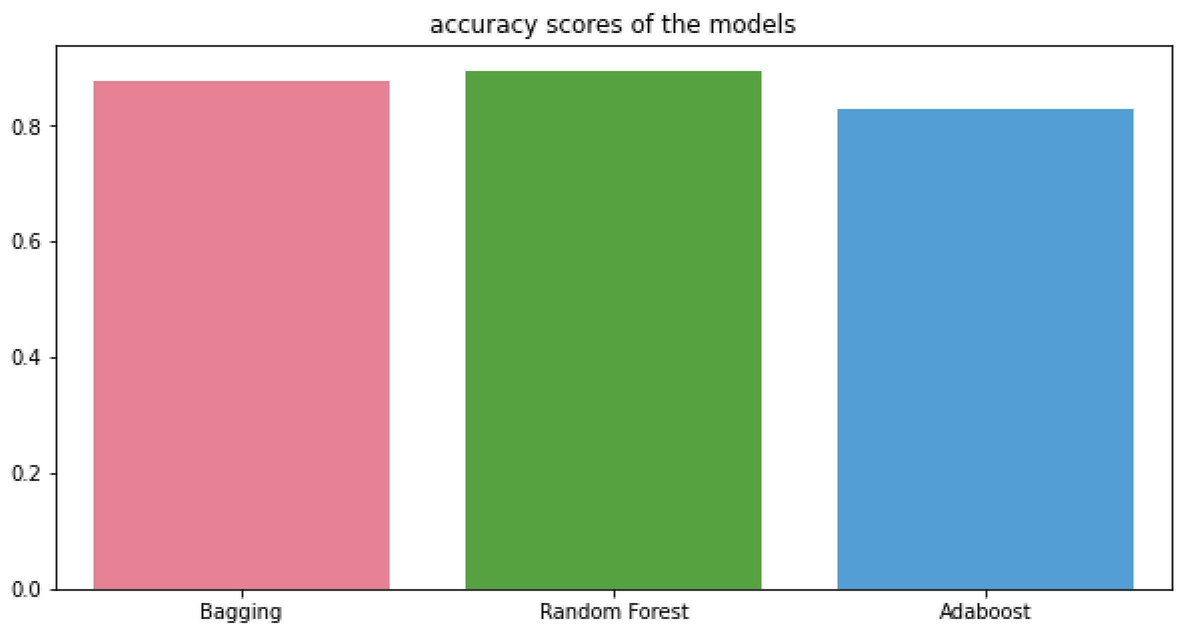
Comparsion of Models Evaluation (Accuracy)

In [48]:

```
models = ['Bagging', 'Random Forest', 'Adaboost']
scores = [accuracy_bag, accuracy_random, accuracy_boost]

plt.figure(figsize = (10,5))
sns.barplot(x = models, y = scores, data =df , palette = 'husl')
plt.title('accuracy scores of the models')
```

Out[48]:

```
Text(0.5, 1.0, 'accuracy scores of the models')
```



- We can conclude from the plot above that Random forest classifier scored the highest accuracy
- Bagging Classifier in which we used the (Decision Tree, KNN and Logistic Regression) models was much near to the accuracy scored by the random forest classifier
- Adaboost Classifier had the lowest score among all of the classifiers

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: