In [1]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
#Importing required modules
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```python
#Importing Churn_Modelling file
churn = pd.read_csv("Churn_Modelling.csv")
```

In [4]:
```python
#Number of rows and columns-(data points) present in the data frame
churn.shape
```

Out[4]: (10000, 14)

In [5]:
```python
#Display first 5 columns using head()
churn.head()
```

Out[5]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	

In [6]:

```
#Display last 5 columns using tail()
churn.tail()
```

Out[6]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiv
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

In [7]:

```
#Checking for null values
churn.isnull().sum()
```

Out[7]:

```
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 [8]:
```
#List of features present in the dataset
churn.columns
```

Out[8]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [9]:
```
#Number of features present in the dataset
len(churn.columns)
```

Out[9]:
```
14
```

In [10]:
```
#Type of each features
churn.dtypes
```

Out[10]:
```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [11]:
```
#Summary Statistics
churn.describe()
```

Out[11]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.00000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.51510
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.49979
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.00000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.00000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.00000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.00000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.00000

In [12]:
```
#Dependent variable - Exited
#The Number of classes present in 'Exited'
len(churn['Exited'].unique())
```

Out[12]:
```
2
```

In [13]:
```
#Name of two classes and the number of data points in each
churn['Exited'].value_counts()
```

Out[13]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

Observations:

1.The are 1000 rows and 14 columns in the given dataset

2.The is no null values present.

```
3.Number of features present in the data frame = 14

4.The target variable is of type 'int64' which means it is numerical,: Binary flag 1 if the customer cl
osed account with bank and 0 if the customer is retained.

5.There are 2 classes present in the target column/variable,so our problem is a classification problem

6.The dataset is an imbalanced dataset as the values of datapoints in each class is not distributed equ
ally
```

Objective

Our main objective is to find whether the customer closed the account with bank (1) or the customer retained(0)

Visualizing The Data

Univariate Analysis

```
Its all about finding out one feature which suits best with/which helps to clearly find whether the cus
tomer exited or   not
```

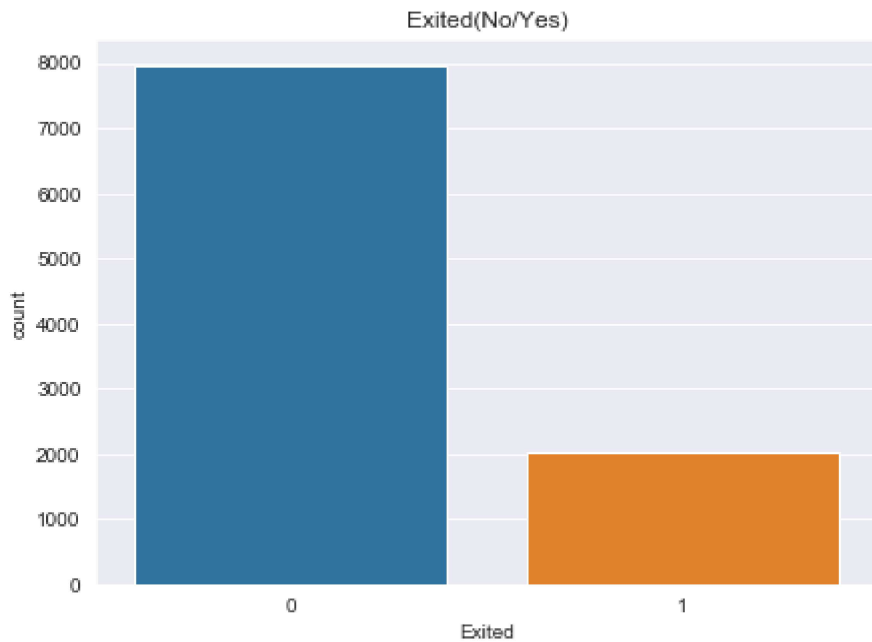
Counter Plot

In [14]:

```
plt.figure(figsize =(7,5))
sns.set_style('darkgrid')
sns.countplot(x = 'Exited', data = churn)
plt.title('Exited(No/Yes)')
churn['Exited'].value_counts()
```

Out[14]:

```
0    7963
1    2037
Name: Exited, dtype: int64
```



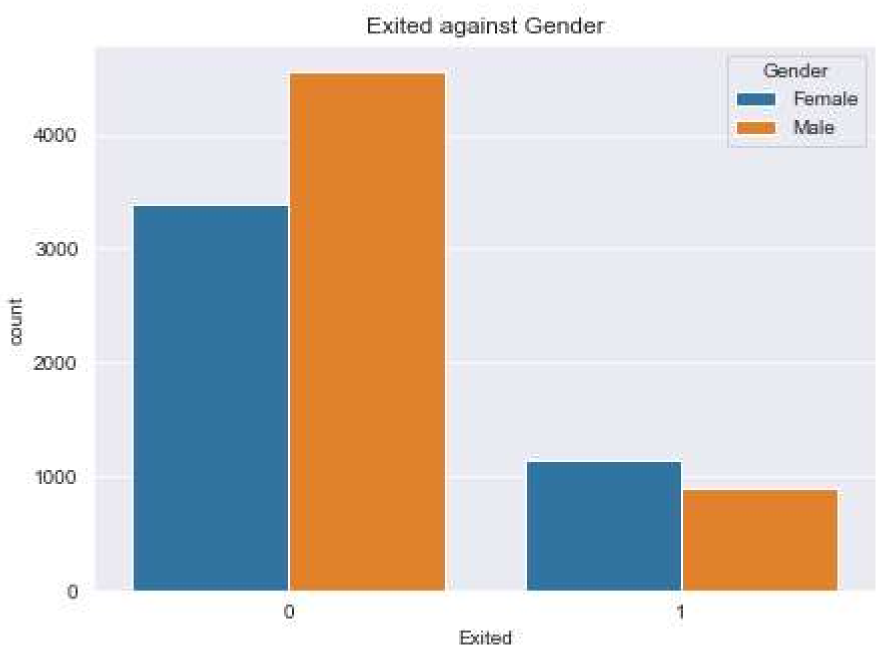
The above bar graph shows that 2000 customers closed the account with the bank and about 8000 customers retained

In [15]:

```
#Exited against Gender
plt.figure(figsize =(7,5))
sns.set_style('darkgrid')
sns.countplot(x = 'Exited', hue = 'Gender', data = churn)
plt.title('Exited against Gender');
pd.DataFrame(churn.groupby(['Gender', 'Exited'])['Exited'].count())
```

Out[15]:

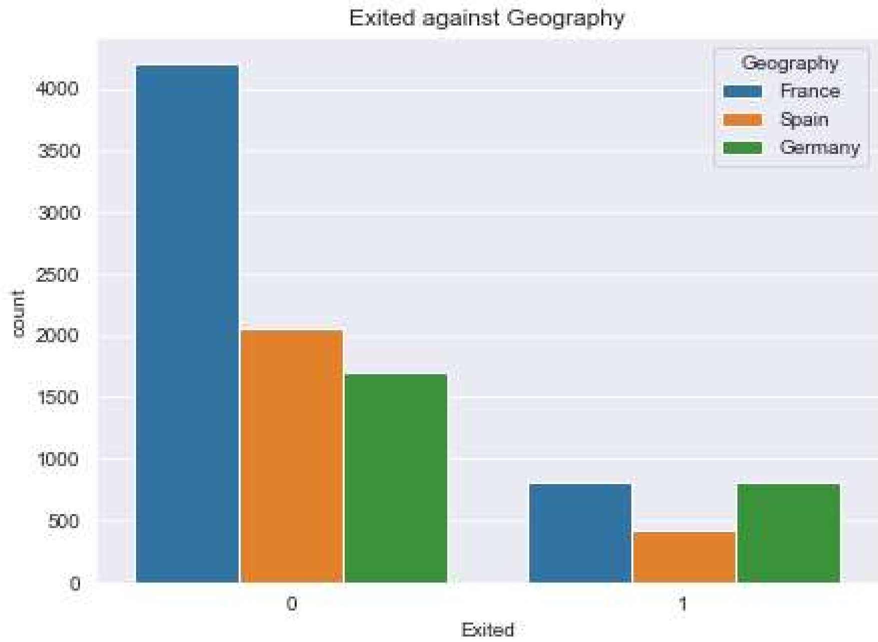
		Exited
Gender	**Exited**	
Female	0	3404
	1	1139
Male	0	4559
	1	898



In [16]:
```
#Exited against Country
plt.figure(figsize =(7,5))
sns.set_style('darkgrid')
sns.countplot(x = 'Exited', hue = 'Geography', data = churn)
plt.title('Exited against Geography');
pd.DataFrame(churn.groupby(['Geography', 'Exited'])['Exited'].count())
```

Out[16]:

		Exited
Geography	**Exited**	
France	0	4204
	1	810
Germany	0	1695
	1	814
Spain	0	2064
	1	413



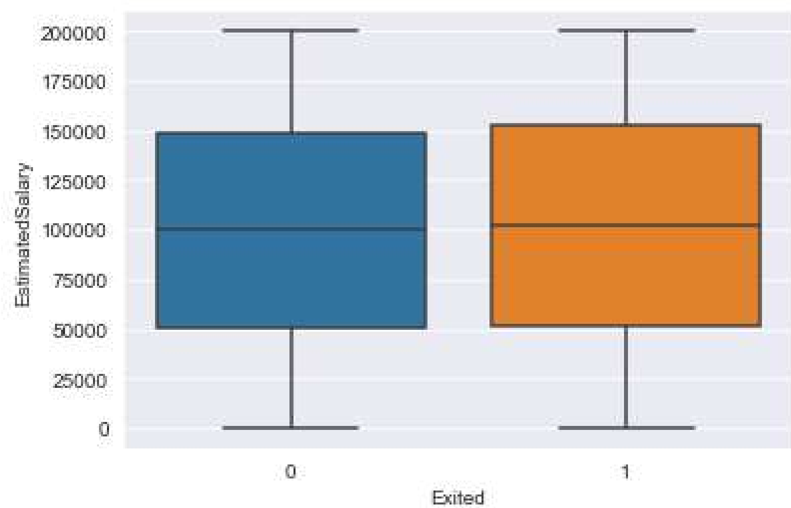
Exited against Geography
Geography
France
Spain
Germany
4000
3500
3000
2500
2000
1500
1000
500
0
count
0
1
Exited


```
In [ ]:  1
```

Boxplot

In [17]:

```
sns.boxplot(x="Exited" , y ="EstimatedSalary",data =churn)
plt.show()
sns.boxplot(x="Exited" , y ="IsActiveMember",data =churn)
plt.show()
sns.boxplot(x="Exited" , y ="HasCrCard",data =churn)
plt.show()
sns.boxplot(x="Exited" , y ="NumOfProducts",data =churn)
plt.show()
sns.boxplot(x="Exited" , y ="Balance",data =churn)
plt.show()
sns.boxplot(x="Exited" , y ="Tenure",data =churn)
plt.show()
sns.boxplot(x="Exited" , y ="Age",data =churn)
plt.show()
sns.boxplot(x="Exited" , y ="CreditScore",data =churn)
plt.show()

```



From the above box plots we can say that ,

```
1.Customers of age 37 to 50 are more likely to close their account with the bank
```

```
2.Customers staying with the bank for 2-3 years or 7-8 years have highest probabilty of closing their a
ccount with
the bank.
```

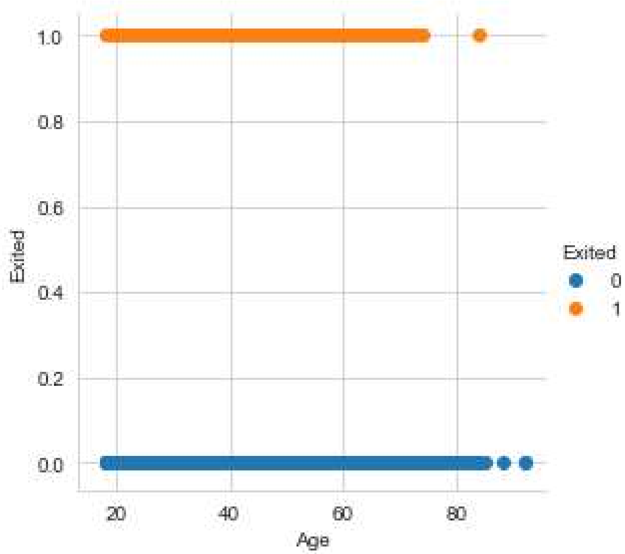
Bivariate Analysis

Its aboubt finding two best features which helps to find the Exited status of a customer

Scatter Plot

In [18]:

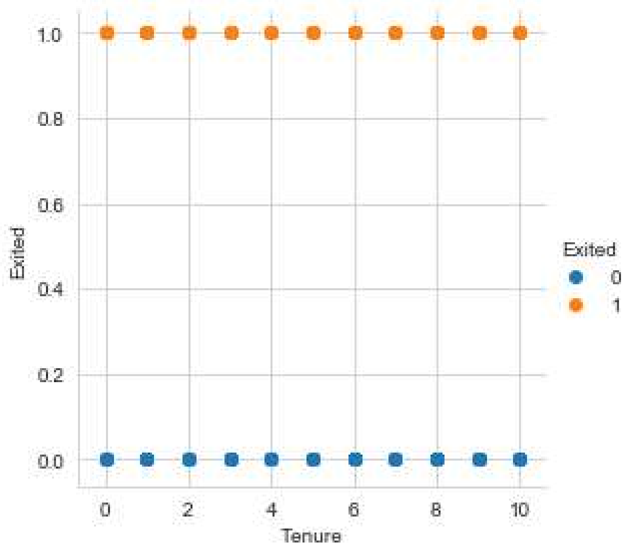
```
sns.set_style("whitegrid");
sns.FacetGrid(churn, hue="Exited", size=4) \
 .map(plt.scatter, "Age", "Exited") \
 .add_legend();

plt.show();

```



Customers aged 75+ has more chance of closing their account

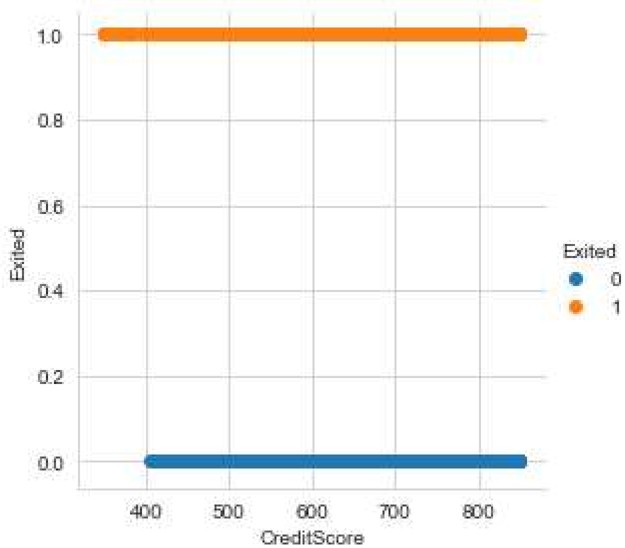
In [19]:

```
sns.set_style("whitegrid");
sns.FacetGrid(churn, hue="Exited", size=4) \
 .map(plt.scatter, "Tenure", "Exited") \
 .add_legend();

plt.show();
```



In [20]:

```
sns.set_style("whitegrid");
sns.FacetGrid(churn, hue="Exited", size=4) \
 .map(plt.scatter, "CreditScore", "Exited") \
 .add_legend();

plt.show();
```

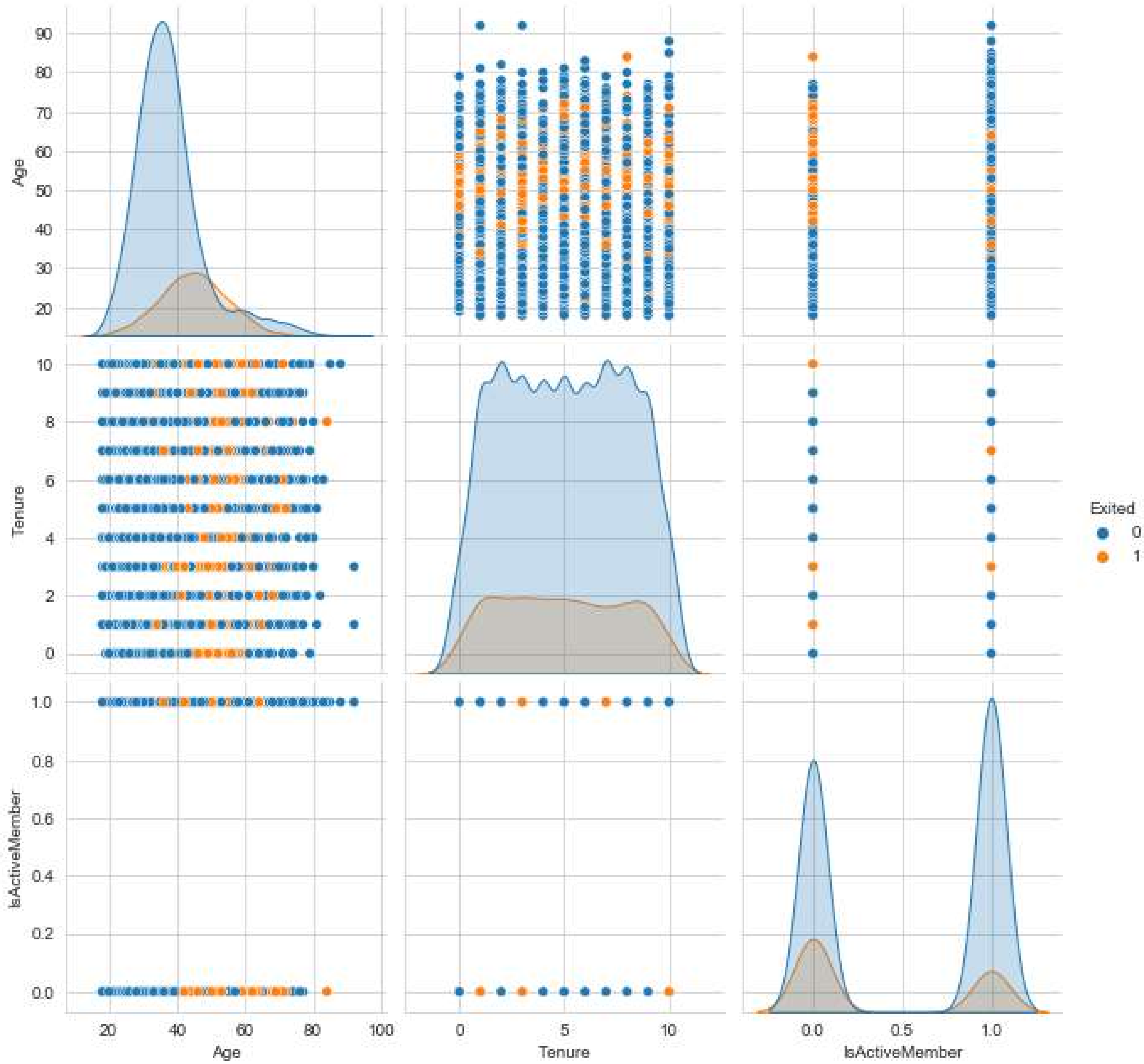


Customer who has credit less than 400 are more likely to close their account with the bank

Pair Plot

In [21]:

```
sns.pairplot(churn, hue="Exited", size=3,vars=["Age", "Tenure", "IsActiveMember"]);

plt.show()
```



Descriptive Statistics

In [22]:

```
#Mean
churn.mean()
```

Out[22]:

```
RowNumber          5.000500e+03
CustomerId         1.569094e+07
CreditScore        6.505288e+02
Age                3.892180e+01
Tenure             5.012800e+00
Balance            7.648589e+04
NumOfProducts      1.530200e+00
HasCrCard          7.055000e-01
IsActiveMember     5.151000e-01
EstimatedSalary    1.000902e+05
Exited             2.037000e-01
dtype: float64
```

In [23]:
```
#Median
churn.median()
```

Out[23]:
```
RowNumber          5.000500e+03
CustomerId         1.569074e+07
CreditScore        6.520000e+02
Age                3.700000e+01
Tenure             5.000000e+00
Balance            9.719854e+04
NumOfProducts      1.000000e+00
HasCrCard          1.000000e+00
IsActiveMember     1.000000e+00
EstimatedSalary    1.001939e+05
Exited             0.000000e+00
dtype: float64
```

In [24]:
```
#Standard Deviation
churn.std()
```

Out[24]:
```
RowNumber           2886.895680
CustomerId         71936.186123
CreditScore           96.653299
Age                   10.487806
Tenure                 2.892174
Balance            62397.405202
NumOfProducts          0.581654
HasCrCard              0.455840
IsActiveMember         0.499797
EstimatedSalary    57510.492818
Exited                 0.402769
dtype: float64
```

In [25]:
```
#Range of Age
minimum = churn['Age'].min()
maximum = churn['Age'].max()
range1 = maximum-minimum
range1

```

Out[25]: 74

In [26]:

```python
#Quantiles
Q1 = churn.quantile(0.25)
Q2 = churn.quantile(0.50)
Q3 = churn.quantile(0.75)
print("25th Percentile :\n ",Q1)
print("50th Percentile :\n ",Q2)
print("75th Percentile :\n ",Q3)
```

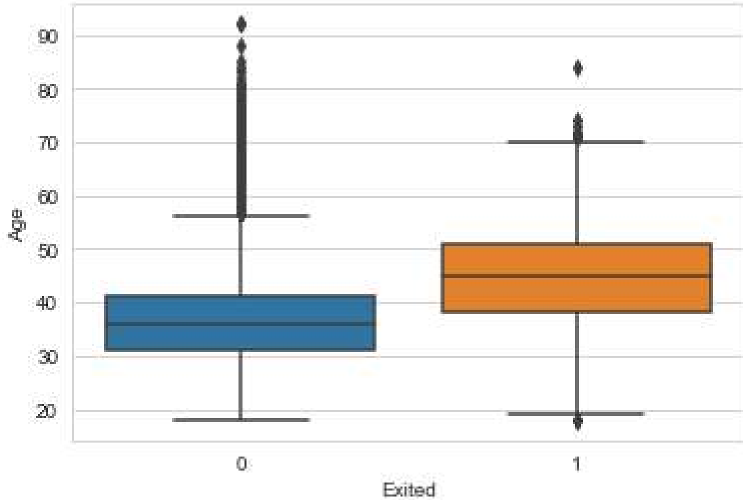
```
25th Percentile :
  RowNumber              2500.75
CustomerId         15628528.25
CreditScore             584.00
Age                      32.00
Tenure                    3.00
Balance                   0.00
NumOfProducts             1.00
HasCrCard                 0.00
IsActiveMember            0.00
EstimatedSalary       51002.11
Exited                    0.00
Name: 0.25, dtype: float64
50th Percentile :
  RowNumber          5.000500e+03
CustomerId         1.569074e+07
CreditScore        6.520000e+02
Age                3.700000e+01
Tenure             5.000000e+00
Balance            9.719854e+04
NumOfProducts      1.000000e+00
HasCrCard          1.000000e+00
IsActiveMember     1.000000e+00
EstimatedSalary    1.001939e+05
Exited             0.000000e+00
Name: 0.5, dtype: float64
75th Percentile :
  RowNumber          7.500250e+03
CustomerId         1.575323e+07
CreditScore        7.180000e+02
Age                4.400000e+01
Tenure             7.000000e+00
Balance            1.276442e+05
NumOfProducts      2.000000e+00
```

```
HasCrCard          1.000000e+00
IsActiveMember     1.000000e+00
EstimatedSalary    1.493882e+05
Exited             0.000000e+00
Name: 0.75, dtype: float64
```

Outliers

In [27]:

```
#Age against Exited

sns.boxplot(x="Exited" , y ="Age",data =churn)
plt.show()
```



The points above the upper and lower whiskers are called a outliers

We can use IQR to remove these outliers

In [28]:

```
Q1 = churn['Age'].quantile(0.25)
Q2 = churn['Age'].quantile(0.50)
Q3 = churn['Age'].quantile(0.75)
IQR = Q3 - Q1
print("25th Percentile :\n ",Q1)
print("75th Percentile :\n ",Q3)
print("IQR: ",IQR)
```

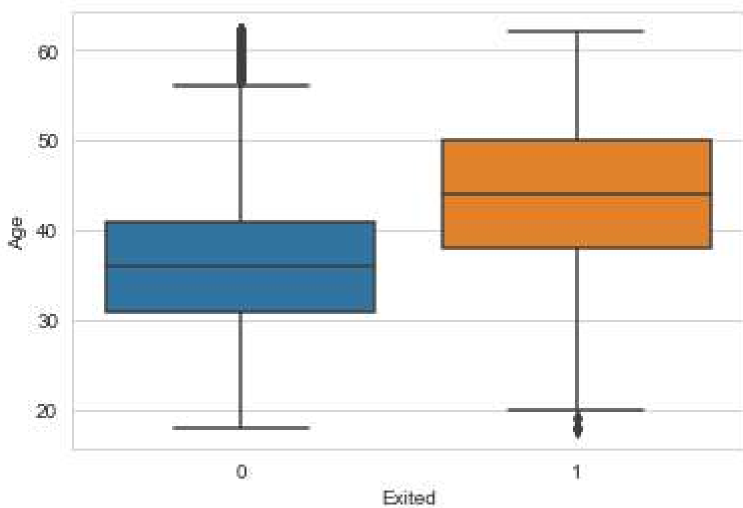
```
25th Percentile :
  32.0
75th Percentile :
  44.0
IQR:  12.0
```

In [29]:

```
#Removing Outliers
churn1 = churn[(churn['Age']>=Q1-(1.5*IQR)) & (churn['Age']<= Q3+(1.5*IQR))]
```

In [30]:

```
#Age after removing outliers
sns.boxplot(x="Exited" , y ="Age",data =churn1)
plt.show()
```



Encoding of Categorical Variables

In [31]:

```
churn2 = churn
```

In [32]:

```
#Encoding Age
#Female-1 and Male-0
for i in churn2.index:
    if(churn2['Gender'][i]=="Female"):
        churn2['Gender'][i]=1
    else:
        churn2['Gender'][i]=0
```

In [33]:

```python
#After Encoding
churn2['Gender'][0:40]
```

Out[33]:

```
0     1
1     1
2     1
3     1
4     1
5     0
6     0
7     1
8     0
9     0
10    0
11    0
12    1
13    1
14    1
15    0
16    0
17    1
18    0
19    1
20    0
21    1
22    1
23    0
24    1
25    0
26    0
27    0
28    1
29    0
30    1
31    0
32    0
33    1
34    1
35    1
36    0
37    0
38    0
```

```
39    0
Name: Gender, dtype: object
```

In [34]:
```
churn2['Geography'].value_counts()
```

Out[34]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [35]:
```
#Encoding Geography
#France - 1
#Germany - 2
#Spain - 3
for i in churn2['Geography'].index:
    if(churn2['Geography'][i]=="France"):
        churn2['Geography'][i]=1
    elif(churn2['Geography'][i]=="Germany"):
        churn2['Geography'][i]=2
    else:
        churn2['Geography'][i]=3
```

In [36]:
```
#After Encoding 'Geography'
churn2['Geography']
```

Out[36]:
```
0       1
1       3
2       1
3       1
4       3
       ..
9995    1
9996    1
9997    1
9998    2
9999    1
Name: Geography, Length: 10000, dtype: object
```

In [37]:
```
churn2
```

Out[37]:

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

) rows × 14 columns

Splitting Dependent and Independent variables

In [38]:
```
#Deleting unnecessary features
churn2.drop(['RowNumber','CustomerId','Surname'],axis = 1,inplace = True)
```

In [39]:
```
churn2
```

Out[39]:

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

10000 rows × 11 columns

In [40]:
```
#Dependent variable
dependent = churn2.iloc[0:10000, 10:11]
```

In [41]:
```
dependent
```

Out[41]:

	Exited
0	1
1	0
2	1
3	0
4	0
...	...
9995	0
9996	0
9997	1
9998	1
9999	0

10000 rows × 1 columns

In [42]:
```
#Independent Variable
independent = churn2.iloc[0:10000, 0:10]
```

In [43]:
```
independent
```

Out[43]:

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

10000 rows × 10 columns

Splitting of Train and Test

In [44]:
```
from sklearn.model_selection import train_test_split
```

In [45]:
```
x_train,x_test,y_train,y_test = train_test_split(independent,dependent,test_size = 0.2 ,random_state = 0)
```

In [46]:
```
x_train.shape
```

Out[46]: (8000, 10)

In [47]:
```
x_test.shape
```
Out[47]:
```
(2000, 10)
```
In [48]:
```
y_train.shape
```
Out[48]:
```
(8000, 1)
```
In [49]:
```
y_test.shape
```
Out[49]:
```
(2000, 1)
```
In [50]:
```
x_train.head()
```
Out[50]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
7389	667	3	1	34	5	0.00	2	1	0	163830.64
9275	427	2	0	42	1	75681.52	1	1	1	57098.00
2995	535	1	1	29	2	112367.34	1	1	0	185630.76
5316	654	3	0	40	5	105683.63	1	1	0	173617.09
356	850	3	1	57	8	126776.30	2	1	1	132298.49

Scaling the Independent Variable - Feature Scaling

In [51]:
```
from sklearn.preprocessing import StandardScaler
```
In [52]:
```
sc = StandardScaler()
```
In [53]:
```
x_train = sc.fit_transform(x_train)
```

In [54]:
```
print(x_train)
```

```
[[ 0.16958176  1.51919821  1.09168714 ...  0.64259497 -1.03227043
   1.10643166]
 [-2.30455945  0.3131264  -0.91601335 ...  0.64259497  0.9687384
  -0.74866447]
 [-1.19119591 -0.89294542  1.09168714 ...  0.64259497 -1.03227043
   1.48533467]
 ...
 [ 0.9015152  -0.89294542 -0.91601335 ...  0.64259497 -1.03227043
   1.41231994]
 [-0.62420521  1.51919821  1.09168714 ...  0.64259497  0.9687384
   0.84432121]
 [-0.28401079  0.3131264   1.09168714 ...  0.64259497 -1.03227043
   0.32472465]]
```

In [55]:
```
x_test = sc.transform(x_test)
```

In [56]:
```
x_test[0:5,:]
```

Out[56]:
```
array([[-0.55204276,  0.3131264 ,  1.09168714, -0.36890377,  1.04473698,
         0.8793029 , -0.92159124,  0.64259497,  0.9687384 ,  1.61085707],
       [-1.31490297, -0.89294542,  1.09168714,  0.10961719, -1.031415  ,
         0.42972196, -0.92159124,  0.64259497, -1.03227043,  0.49587037],
       [ 0.57162971,  1.51919821,  1.09168714,  0.30102557,  1.04473698,
         0.30858264, -0.92159124,  0.64259497,  0.9687384 , -0.42478674],
       [ 1.41696129, -0.89294542, -0.91601335, -0.65601634, -0.33936434,
         0.57533623, -0.92159124, -1.55619021, -1.03227043, -0.18777657],
       [ 0.57162971,  0.3131264 , -0.91601335, -0.08179119,  0.00666099,
         1.38961097,  0.8095029 ,  0.64259497,  0.9687384 ,  0.61684179]])
```