In [33]:

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

2.Load the Data

In [83]:

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

Out[83]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	838
2	3	15619304	Onio	502	France	Female	42	8	1596
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1255
...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	
9996	9997	15569892	Johnstone	516	France	Male	35	10	573
9997	9998	15584532	Liu	709	France	Female	36	7	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	750
9999	10000	15628319	Walker	792	France	Female	28	4	1301

10000 rows × 14 columns

In [35]:

```
df.head()
```

Out[35]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balanc
0	1	15634602	Hargrave	619	France	Female	42	2	0.0
1	2	15647311	Hill	608	Spain	Female	41	1	83807.8
2	3	15619304	Onio	502	France	Female	42	8	159660.8
3	4	15701354	Boni	699	France	Female	39	1	0.0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.8

3. Performing Visualization

Univariate analysis

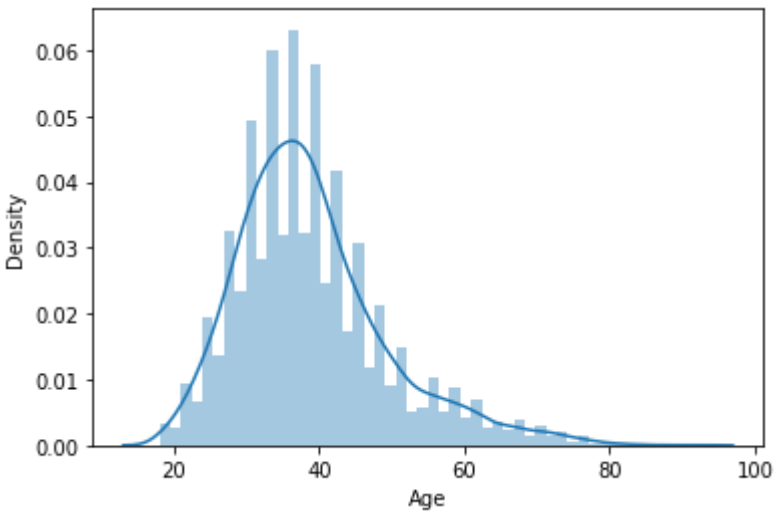
In [36]:

```
sns.distplot(df.Age)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level func
tion with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa71205fb10>
```



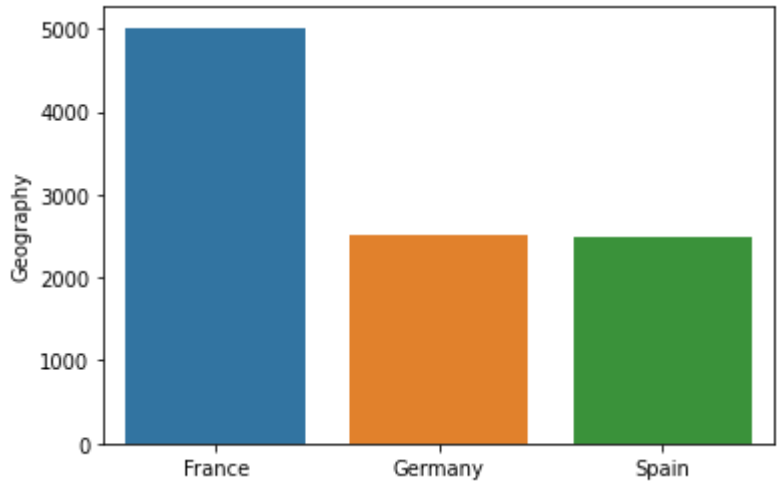
In [37]:

```
sns.barplot(df.Geography.value_counts().index, df.Geography.value_counts())
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
ing: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa711ef2a10>
```

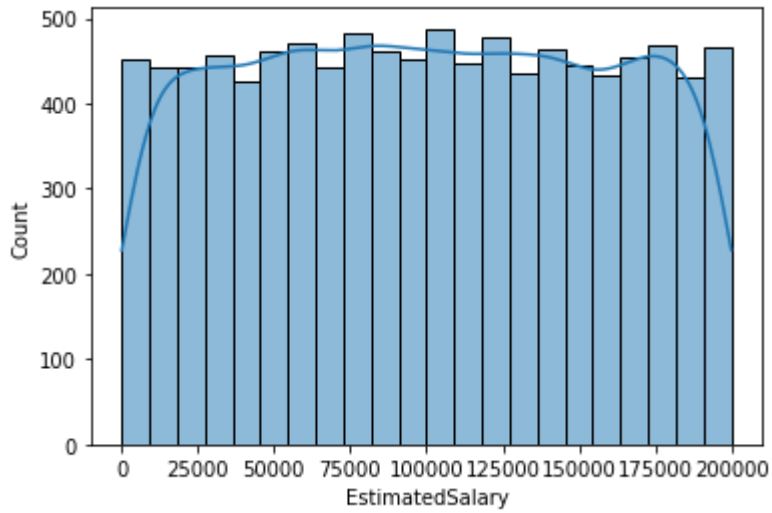


In [38]:

```
sns.histplot(df.EstimatedSalary,kde=True)
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa711ec1d10>
```



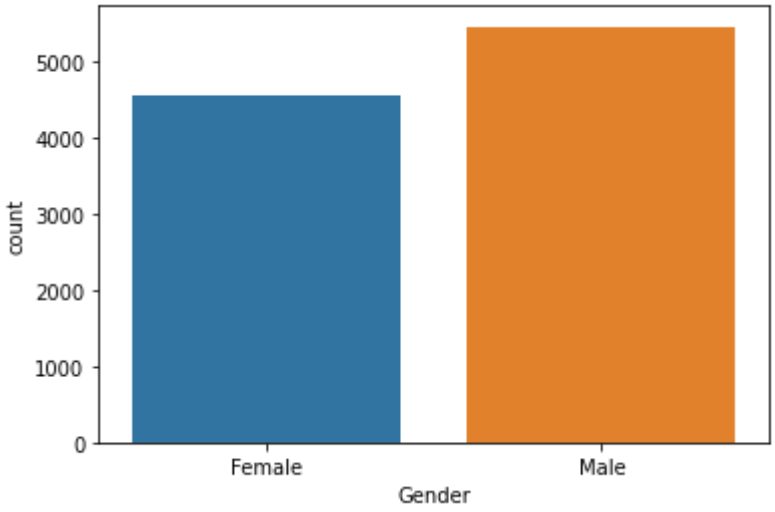
In [39]:

```
sns.countplot(df.Gender)  ## univariant analysis for categorial datas
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
ing: Pass the following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments w
ithout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa711eb1fd0>
```



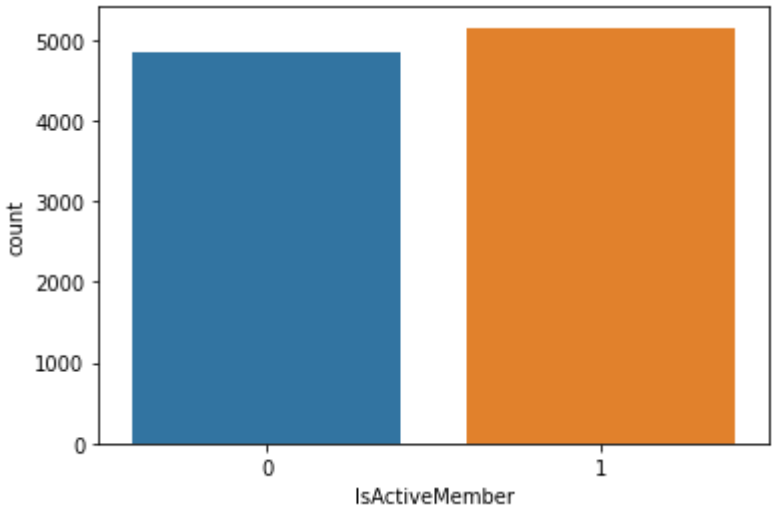
In [40]:

```
sns.countplot(df.IsActiveMember) ## univariant analysis for categorial datas
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
ing: Pass the following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments w
ithout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa711dcfa90>
```



Bivariate Analysis

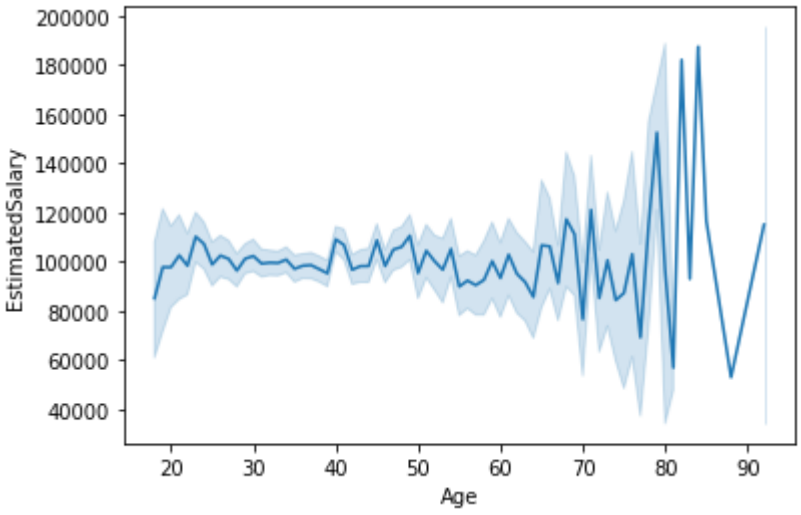
In [41]:

```
sns.lineplot(df.Age,df.EstimatedSalary) # bivariate analysis for continuous data
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
ing: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

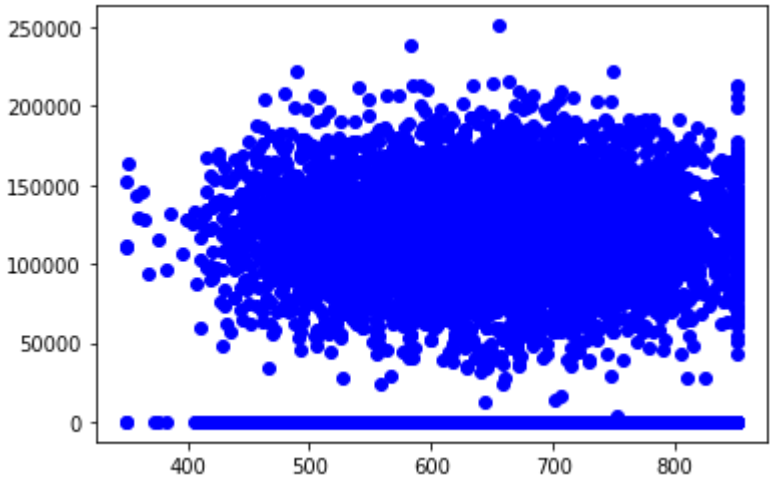
Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa711d32c50>
```



In [42]:

```
plt.scatter(df.CreditScore,df.Balance,c="blue")
plt.show()
```

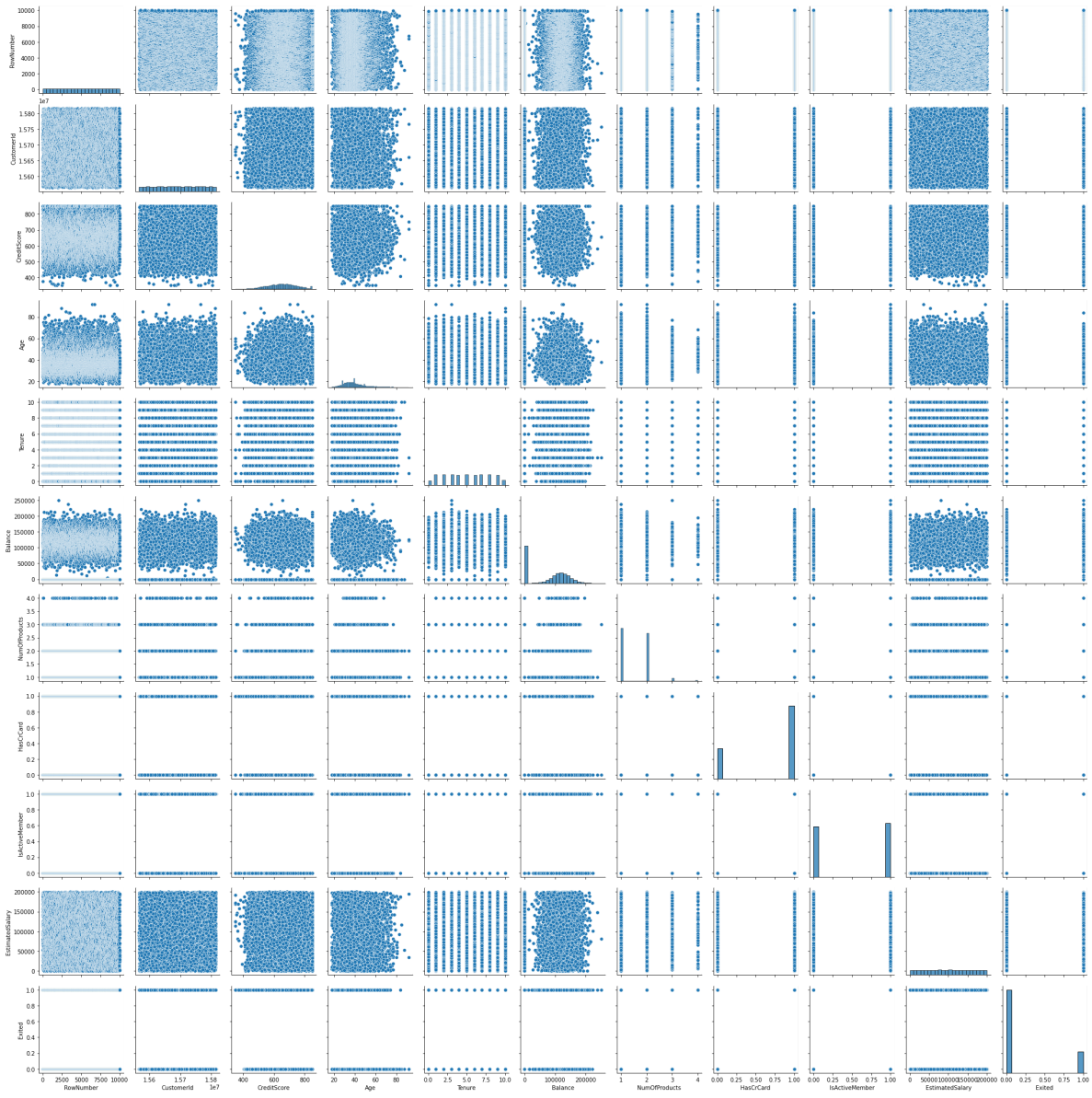


In [43]:

```
sns.pairplot(df)
```

Out[43]:

```
<seaborn.axisgrid.PairGrid at 0x7fa711ca3c50>
```



Multivariant analysis

In [44]:

```
df.corr()
```

Out[44]:

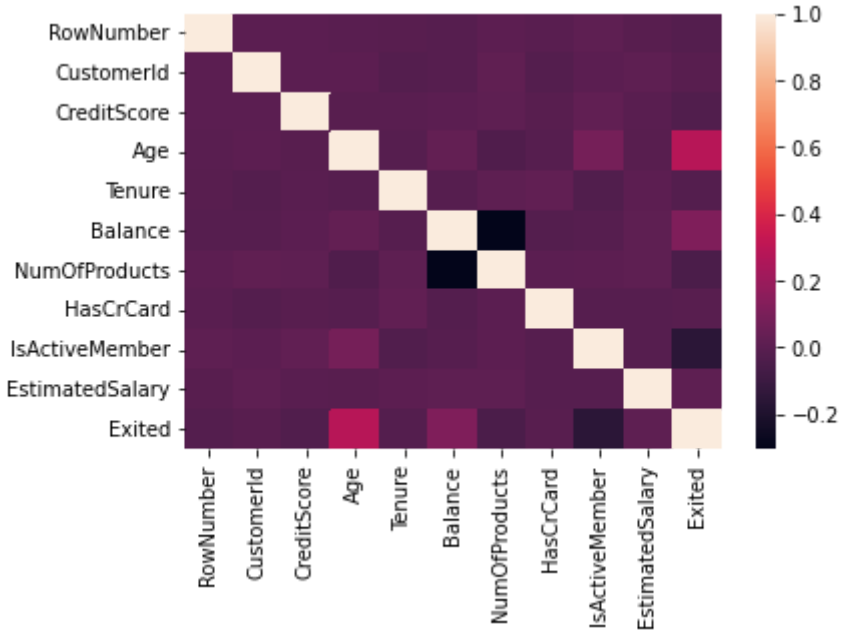
	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOf
RowNumber	1.000000	0.004202	0.005840	0.000783	-0.006495	-0.009067	
CustomerId	0.004202	1.000000	0.005308	0.009497	-0.014883	-0.012419	
CreditScore	0.005840	0.005308	1.000000	-0.003965	0.000842	0.006268	
Age	0.000783	0.009497	-0.003965	1.000000	-0.009997	0.028308	-
Tenure	-0.006495	-0.014883	0.000842	-0.009997	1.000000	-0.012254	
Balance	-0.009067	-0.012419	0.006268	0.028308	-0.012254	1.000000	-
NumOfProducts	0.007246	0.016972	0.012238	-0.030680	0.013444	-0.304180	
HasCrCard	0.000599	-0.014025	-0.005458	-0.011721	0.022583	-0.014858	
IsActiveMember	0.012044	0.001665	0.025651	0.085472	-0.028362	-0.010084	
EstimatedSalary	-0.005988	0.015271	-0.001384	-0.007201	0.007784	0.012797	
Exited	-0.016571	-0.006248	-0.027094	0.285323	-0.014001	0.118533	-

In [45]:

```
sns.heatmap(df.corr())
```

Out[45]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa70ea8a650>
```

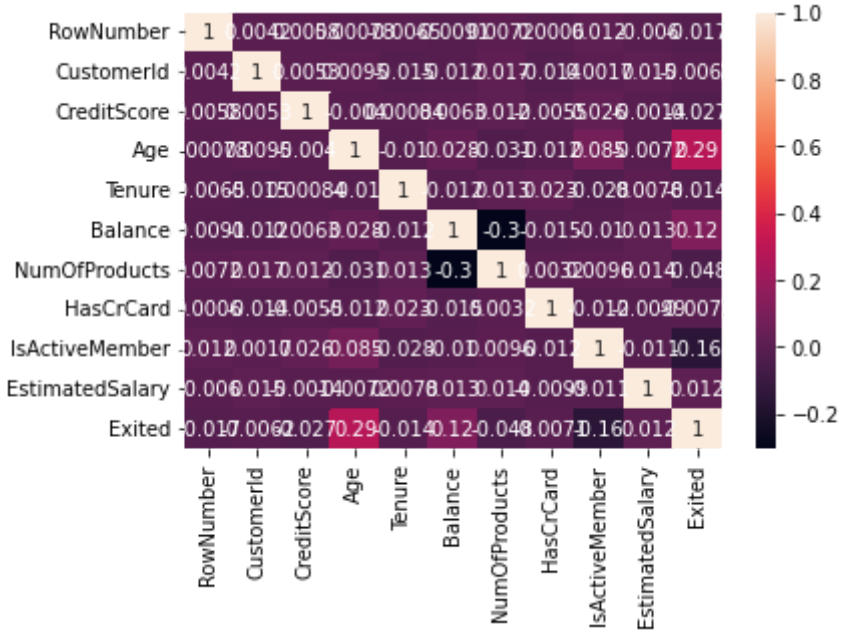


In [46]:

```
sns.heatmap(df.corr(),annot=True)
```

Out[46]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa70e9bf890>
```



4.Descriptive statistics on the data

In [47]:

```
df.describe()
```

Out[47]:

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

In [48]:

```
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 [49]:

```
df.shape
```

Out[49]:

```
(10000, 14)
```

In [50]:

```
df.isnull().any()
```

Out[50]:

```
RowNumber          False
CustomerId         False
Surname            False
CreditScore        False
Geography          False
Gender             False
Age                False
Tenure             False
Balance            False
NumOfProducts      False
HasCrCard          False
IsActiveMember     False
EstimatedSalary    False
Exited             False
dtype: bool
```

In [51]:

```
df.Geography.value_counts()
```

Out[51]:

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

In [52]:

```
df.HasCrCard.value_counts()
```

Out[52]:

```
1    7055
0    2945
Name: HasCrCard, dtype: int64
```

In [53]:

```
df.IsActiveMember .value_counts()
```

Out[53]:

```
1    5151
0    4849
Name: IsActiveMember, dtype: int64
```

5.Handling the missing data

In [54]:

```
## for numerical column use mean or median for null value replacement
## for categoriccal columns use mode for null value replacement.
##HasCrCard,IsActiveMember,Exited are categorial columns
```

In [55]:

```
df['IsActiveMember'].fillna(df['IsActiveMember'].mode(),inplace=True)
```

In [56]:

```
## for numerical column
df['Age'].fillna(df['Age'].mean(),inplace=True)
```

In [57]:

```
df.isnull().any()
```

Out[57]:

```
RowNumber          False
CustomerId         False
Surname            False
CreditScore        False
Geography          False
Gender             False
Age                False
Tenure             False
Balance            False
NumOfProducts      False
HasCrCard          False
IsActiveMember     False
EstimatedSalary    False
Exited             False
dtype: bool
```

#6.Finding outliners and replacing it

In [109]:

```
np.where(df['Balance']>200000)
```

Out[109]:

```
(array([ 138,  520,  720, 1067, 1174, 1365, 1533, 2092, 2597, 2709, 3150,
        3280, 3588, 3920, 4436, 4533, 5254, 5871, 6271, 6497, 6717, 6913,
        7353, 7492, 7632, 7887, 8027, 8063, 8702, 8733, 8794, 8982, 9833,
        9920]),)
```

In [111]:

```
df[9833:9834]
```

Out[111]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bal
9833	9834	15807245	McKay	699	Germany	Female	41	1	2001

In [116]:

```
Q1 = np.percentile(df['Balance'], 50,
                   interpolation = 'midpoint')

Q3 = np.percentile(df['Balance'], 90,
                   interpolation = 'midpoint')
IQR = Q3 - Q1
IQR
```

Out[116]:

```
52069.54000000001
```

In [117]:

```
print("Old Shape: ", df.shape)

# Upper bound
upper = np.where(df['Balance'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(df['Balance'] <= (Q1-1.5*IQR))

''' Removing the Outliers '''
df.drop(upper[0], inplace = True)
df.drop(lower[0], inplace = True)

print("New Shape: ", df.shape)
```

```
Old Shape:  (10000, 14)
New Shape:  (6377, 14)
```

#7.Performing encoding for categorical values

One hot Encoding

In [58]:

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

Out[58]:

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

In [59]:

```
from sklearn.preprocessing import OneHotEncoder
country_encoder = OneHotEncoder()
```

In [65]:

```
shaped_country=np.array(df["Geography"]).reshape(-1,1)
values = country_encoder.fit_transform(shaped_country)
values.toarray()[:10]
```

Out[65]:

```
array([[1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

Checking the correctness of One Hot encoder

In [66]:

```
print(country_encoder.inverse_transform(values)[:10])
```

```
[['France']
 ['Spain']
 ['France']
 ['France']
 ['Spain']
 ['Spain']
 ['France']
 ['Germany']
 ['France']
 ['France']]
```

Label Encoding

In [67]:

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

Out[67]:

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

In [68]:

```
from sklearn.preprocessing import LabelEncoder
gender_encoder = LabelEncoder()
```

In [69]:

```
gender_encoder.fit(df['Gender'])
```

Out[69]:

```
LabelEncoder()
```

In [70]:

```
values1 = gender_encoder.transform(df['Gender'])
values1[:10]
```

Out[70]:

```
array([0, 0, 0, 0, 0, 1, 1, 0, 1, 1])
```

In [72]:

```
list(df['Gender'][:10])
```

Out[72]:

```
['Female',
 'Female',
 'Female',
 'Female',
 'Female',
 'Male',
 'Male',
 'Female',
 'Male',
 'Male']
```

8.Spliting Dependent and Independent variables

Independent Variable

In [78]:

```
x=df.iloc[:,3:13].values
x
```

Out[78]:

```
array([[619, 'France', 'Female', ..., 1, 1, 101348.88],
       [608, 'Spain', 'Female', ..., 0, 1, 112542.58],
       [502, 'France', 'Female', ..., 1, 0, 113931.57],
       ...,
       [709, 'France', 'Female', ..., 0, 1, 42085.58],
       [772, 'Germany', 'Male', ..., 1, 0, 92888.52],
       [792, 'France', 'Female', ..., 1, 0, 38190.78]], dtype=object)
```

Dependent Variable

In [82]:

```
y=df.iloc[:,-1].values
y
```

Out[82]:

```
array([1, 0, 1, ..., 1, 1, 0])
```

#9.Scaling Independent Variables

Scaling Balance column

In [84]:

```
from sklearn.preprocessing import StandardScaler
```

In [85]:

```
scaler = StandardScaler()
```

In [89]:

```
val=np.array(df['Balance']).reshape(-1,1)
scaled = scaler.fit_transform(val)
print(scaled)
```

```
[[-1.22584767]
 [ 0.11735002]
 [ 1.33305335]
 ...
 [-1.22584767]
 [-0.02260751]
 [ 0.85996499]]
```

#10.Spliting Training and Testing data

In [90]:

```
from sklearn.model_selection import train_test_split
```

In [104]:

```
x_train,y_train,x_test,y_test=train_test_split(x,y,test_size=0.3,shuffle=True)
```

In [105]:

```
x_train.shape
```

Out[105]:

```
(7000, 10)
```

In [106]:

```
y_train.shape
```

Out[106]:

```
(3000, 10)
```

In [107]:

```
x_test.shape
```

Out[107]:

```
(7000,)
```

In [108]:

```
y_test.shape
```

Out[108]:

```
(3000,)
```