In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
import xgboost
```

Loading Dataset

In [2]:

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

Out[2]:

	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

In [3]:

```
df.tail(10)
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure
9990	9991	15798964	Nkemakonam	714	Germany	Male	33	3
9991	9992	15769959	Ajuluchukwu	597	France	Female	53	4
9992	9993	15657105	Chukwualuka	726	Spain	Male	36	2
9993	9994	15569266	Rahman	644	France	Male	28	7
9994	9995	15719294	Wood	800	France	Female	29	2
9995	9996	15606229	Obijiaku	771	France	Male	39	5
9996	9997	15569892	Johnstone	516	France	Male	35	10
9997	9998	15584532	Liu	709	France	Female	36	7
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3
9999	10000	15628319	Walker	792	France	Female	28	4

Collecting dataset information using inbuilt functions

In [4]:

```
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 [5]:

```
df.describe()
```

Out[5]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
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 [6]:

```
df.shape
```

Out[6]:

```
(10000, 14)
```

In [7]:

```
df.dtypes# avoid
```

Out[7]:

```
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
```

Here We will check null values in our dataset

In [8]:

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

Out[8]:

```
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 [9]:

```
# now will see unique values in each parameters

df.nunique()
```

Out[9]:

```
RowNumber          10000
CustomerId         10000
Surname             2932
CreditScore          460
Geography              3
Gender                 2
Age                   70
Tenure                11
Balance             6382
NumOfProducts          4
HasCrCard              2
IsActiveMember         2
EstimatedSalary     9999
Exited                 2
dtype: int64
```

From above operation we can see that some parameters showing very high null values although some of the columns still required like balance and estimated salary

In [10]:

```
# Now we will remove top 3 parameters
df=df.drop(['RowNumber','CustomerId','Surname'],axis=1)
```

In [11]:

```
# checking the new dataset
df.head()
```

Out[11]:

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

In [12]:

```
df[['Gender','Geography','Exited']].value_counts() # geography has 3 categories
```

Out[12]:

```
Gender  Geography  Exited
Male    France     0         2403
Female  France     0         1801
Male    Spain      0         1206
        Germany    0          950
Female  Spain      0          858
        Germany    0          745
        France     1          460
        Germany    1          448
Male    Germany    1          366
        France     1          350
Female  Spain      1          231
Male    Spain      1          182
dtype: int64
```

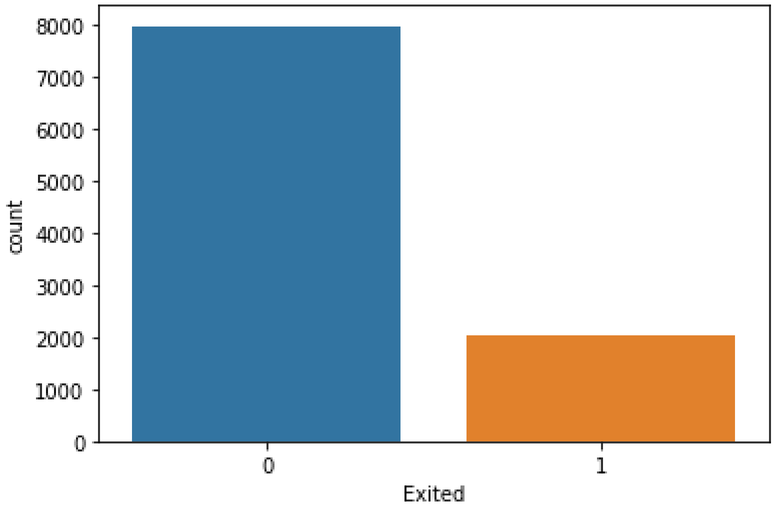
In exited parameter it can be seen that there is some thing biased in this parameter

In [13]:

```
# lets visualize exited parameter

sns.countplot(data=df,x='Exited')
```

Out[13]:

```
<AxesSubplot:xlabel='Exited', ylabel='count'>
```



EDA With Univariate and Bivariate analysis

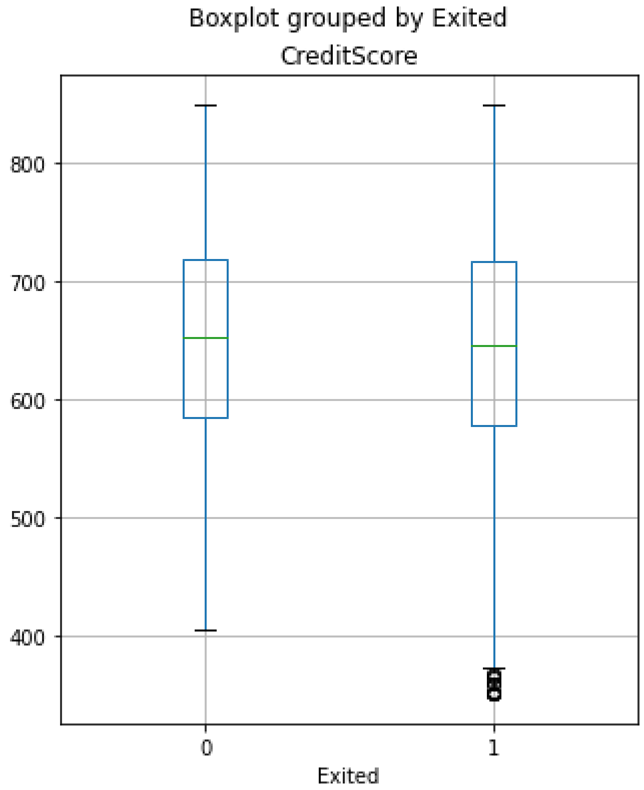
With all parameters

In [14]:

```
df.boxplot(column='CreditScore',by='Exited',figsize=(5,6))
```

Out[14]:

```
<AxesSubplot:title={'center':'CreditScore'}, xlabel='Exited'>
```



In [15]:

```
df.CreditScore.unique()
```

Out[15]:

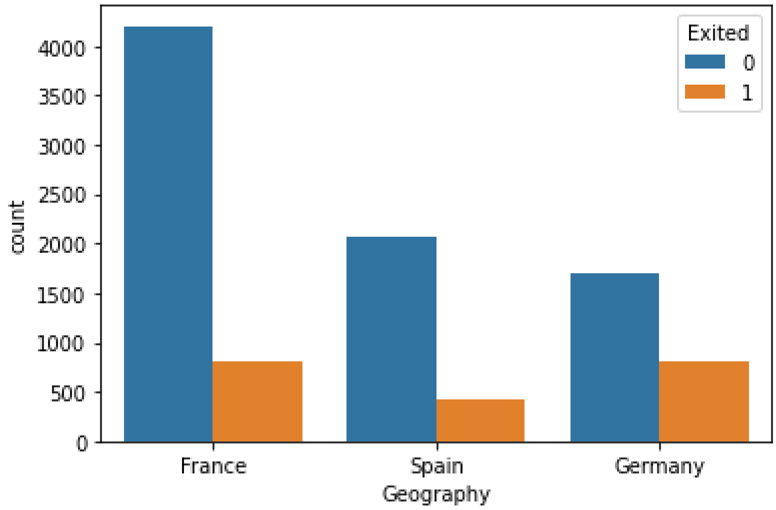
```
array([619, 608, 502, 699, 850, 645, 822, 376, 501, 684, 528, 497, 476,
       549, 635, 616, 653, 587, 726, 732, 636, 510, 669, 846, 577, 756,
       571, 574, 411, 591, 533, 553, 520, 722, 475, 490, 804, 582, 472,
       465, 556, 834, 660, 776, 829, 637, 550, 698, 585, 788, 655, 601,
       656, 725, 511, 614, 742, 687, 555, 603, 751, 581, 735, 661, 675,
       738, 813, 657, 604, 519, 664, 678, 757, 416, 665, 777, 543, 506,
       493, 652, 750, 729, 646, 647, 808, 524, 769, 730, 515, 773, 814,
       710, 413, 623, 670, 622, 785, 605, 479, 685, 538, 562, 721, 628,
       668, 828, 674, 625, 432, 770, 758, 795, 686, 789, 589, 461, 584,
       579, 663, 682, 793, 691, 485, 650, 754, 535, 716, 539, 706, 586,
       631, 717, 800, 683, 704, 615, 667, 484, 480, 578, 512, 606, 597,
       778, 514, 525, 715, 580, 807, 521, 759, 516, 711, 618, 643, 671,
       689, 620, 676, 572, 695, 592, 567, 694, 547, 594, 673, 610, 767,
       763, 712, 703, 662, 659, 523, 772, 545, 634, 739, 771, 681, 544,
       696, 766, 727, 693, 557, 531, 498, 651, 791, 733, 811, 707, 714,
       782, 775, 799, 602, 744, 588, 747, 583, 627, 731, 629, 438, 642,
       806, 474, 559, 429, 680, 749, 734, 644, 626, 649, 805, 718, 840,
       630, 654, 762, 568, 613, 522, 737, 648, 443, 640, 540, 460, 593,
       801, 611, 802, 745, 483, 690, 492, 709, 705, 560, 752, 701, 537,
       487, 596, 702, 486, 724, 548, 464, 790, 534, 748, 494, 590, 468,
       509, 818, 816, 536, 753, 774, 621, 569, 658, 798, 641, 542, 692,
       639, 765, 570, 638, 599, 632, 779, 527, 564, 833, 504, 842, 508,
       417, 598, 741, 607, 761, 848, 546, 439, 755, 760, 526, 713, 700,
       666, 566, 495, 688, 612, 477, 427, 839, 819, 720, 459, 503, 624,
       529, 563, 482, 796, 445, 746, 786, 554, 672, 787, 499, 844, 450,
       815, 838, 803, 736, 633, 600, 679, 517, 792, 743, 488, 421, 841,
       708, 507, 505, 456, 435, 561, 518, 565, 728, 784, 552, 609, 764,
       697, 723, 551, 444, 719, 496, 541, 830, 812, 677, 420, 595, 617,
       809, 500, 826, 434, 513, 478, 797, 363, 399, 463, 780, 452, 575,
       837, 794, 824, 428, 823, 781, 849, 489, 431, 457, 768, 831, 359,
       820, 573, 576, 558, 817, 449, 440, 415, 821, 530, 350, 446, 425,
       740, 481, 783, 358, 845, 451, 458, 469, 423, 404, 836, 473, 835,
       466, 491, 351, 827, 843, 365, 532, 414, 453, 471, 401, 810, 832,
       470, 447, 422, 825, 430, 436, 426, 408, 847, 418, 437, 410, 454,
       407, 455, 462, 386, 405, 383, 395, 467, 433, 442, 424, 448, 441,
       367, 412, 382, 373, 419], dtype=int64)
```

In [16]:

```
# with geography
sns.countplot(data=df,x='Geography',hue='Exited')
```

Out[16]:

```
<AxesSubplot:xlabel='Geography', ylabel='count'>
```

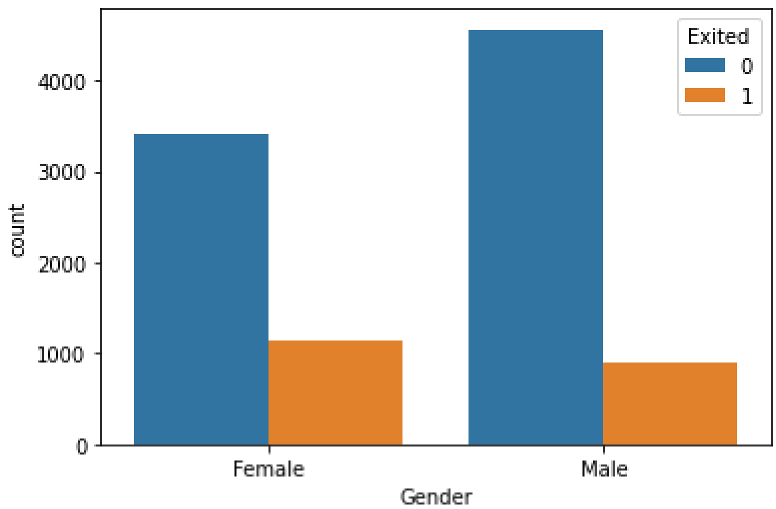


In [17]:

```
# with Gender
sns.countplot(data=df,x='Gender',hue='Exited')
```

Out[17]:

```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```



From gender v/s churn graph we can see that, female has exited more and male has retained more.

In [ ]:

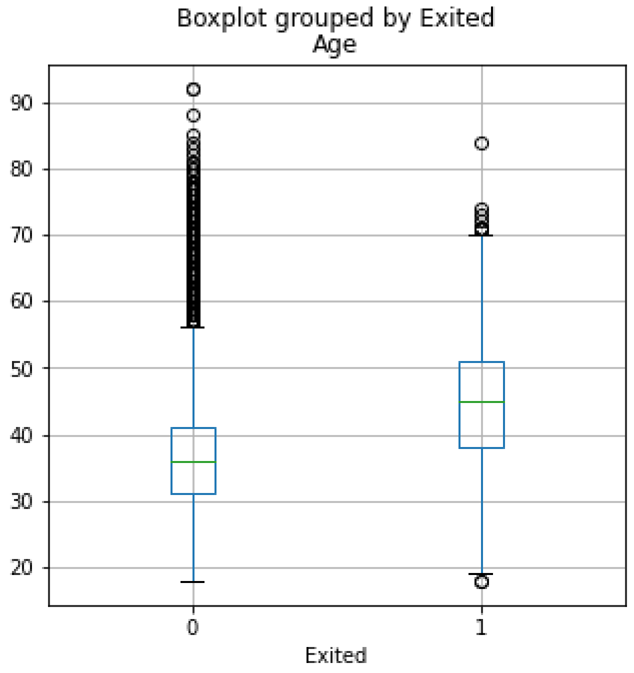
```

```

In [18]:

```
# with age
df.boxplot(column='Age',by='Exited',figsize=(5,5))
```

Out[18]:

```
<AxesSubplot:title={'center':'Age'}, xlabel='Exited'>
```

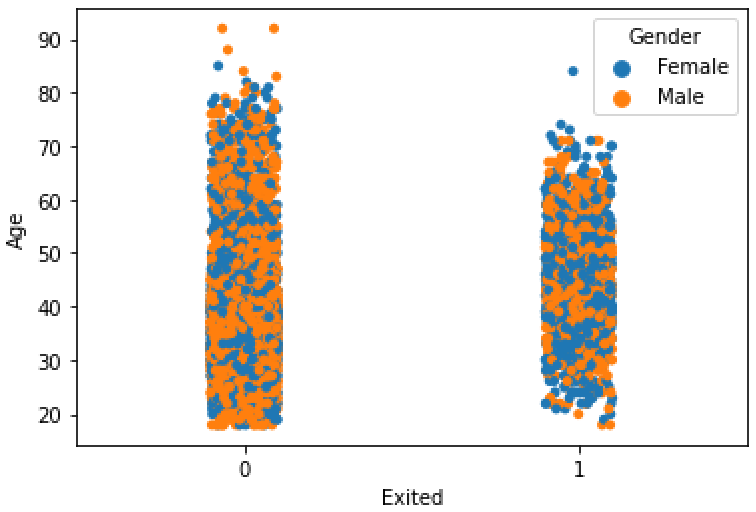


In [19]:

```
sns.stripplot(data=df, x="Exited", y="Age", hue="Gender")
```

Out[19]:

```
<AxesSubplot:xlabel='Exited', ylabel='Age'>
```

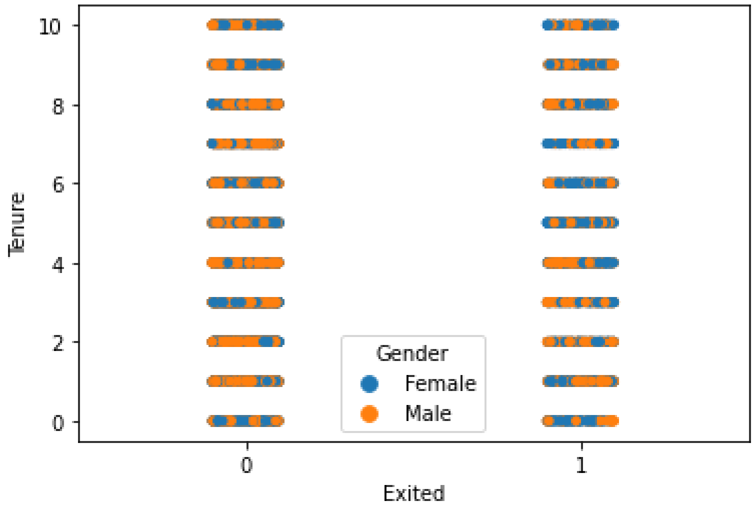


In [20]:

```python
sns.stripplot(data=df, x="Exited", y="Tenure", hue="Gender")
```

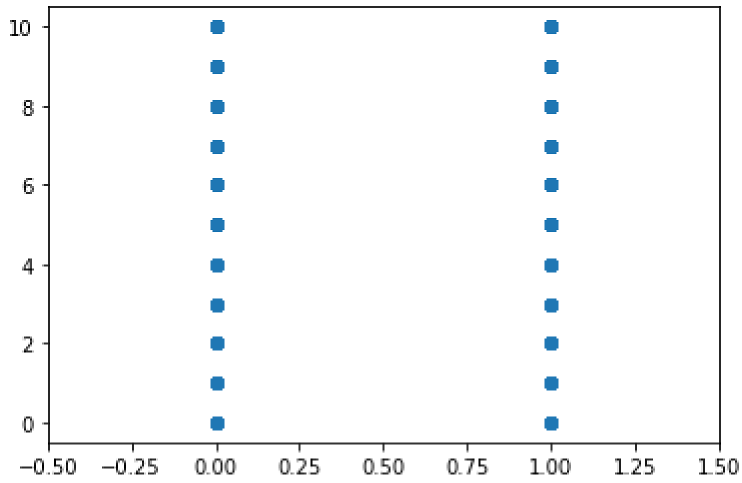
Out[20]:

```
<AxesSubplot:xlabel='Exited', ylabel='Tenure'>
```



In [21]:

```python
plt.scatter(df['Exited'],df['Tenure'])
plt.margins(x=0.5)
plt.show()
```

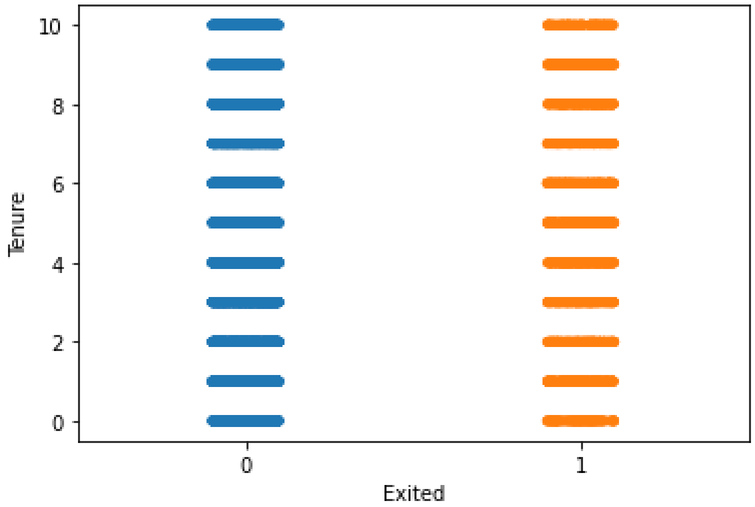


In [22]:

```
sns.stripplot(x="Exited", y="Tenure", data=df)
```

Out[22]:

```
<AxesSubplot:xlabel='Exited', ylabel='Tenure'>
```



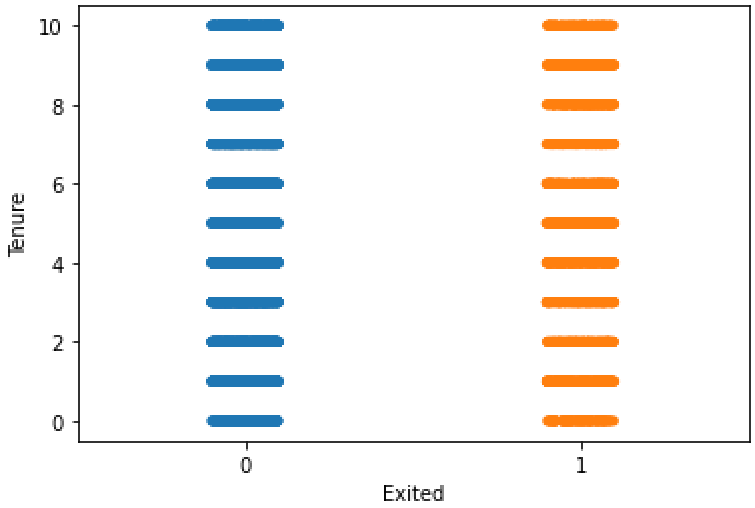
In below and above graph there is only difference of representation (visualization)

In [23]:

```
sns.stripplot(data=df,x="Exited", y="Tenure")
```

Out[23]:

```
<AxesSubplot:xlabel='Exited', ylabel='Tenure'>
```

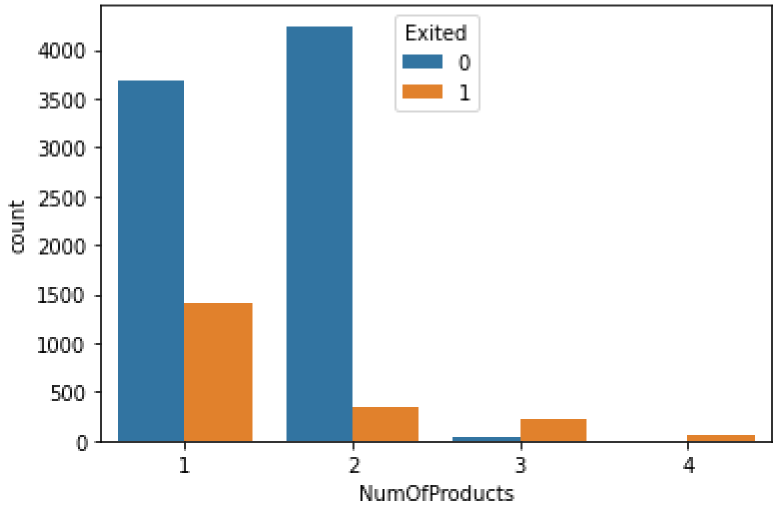


In [24]:

```
sns.countplot(data=df,x='NumOfProducts',hue='Exited')
```

Out[24]:

<AxesSubplot:xlabel='NumOfProducts', ylabel='count'>

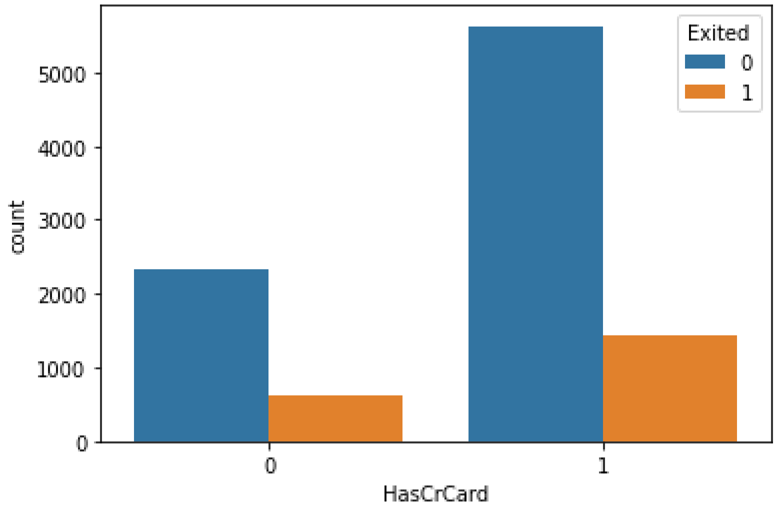


In [25]:

```
sns.countplot(data=df,x='HasCrCard',hue='Exited')
```

Out[25]:

<AxesSubplot:xlabel='HasCrCard', ylabel='count'>

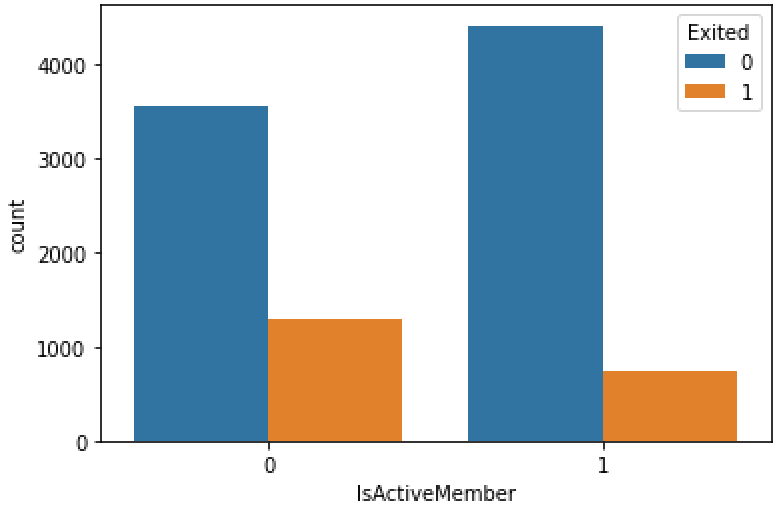


In [26]:

```
sns.countplot(data=df,x='IsActiveMember',hue='Exited')
```

Out[26]:

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



In [27]:

```
# for Balance grouping them in a range
df['Balance'].unique()
```

Out[27]:

```
array([     0.  ,  83807.86, 159660.8 , ...,  57369.61,  75075.31,
       130142.79])
```

In [28]:

```
Category=pd.cut(df.Balance,bins=[-1,0,10000,20000,30000,40000,50000,60000,70000,80000,90
               labels=['No_balance','below_10k','below_20k','below_30k','below_40k','bel
                       'below_70k','below_80k','below_90k','below_1lac','below_1.1lac','
                       'below_1.4lac','below_1.5lac','below_1.6lac','below_1.7lac','belo
                       'below_2lac','below_2.1lac','below_2.2lac','below_2.3lac','below_
```

In [29]:

```
Category
```

Out[29]:

```
0         No_balance
1          below_90k
2       below_1.6lac
3         No_balance
4       below_1.3lac
            ...
9995      No_balance
9996       below_60k
9997      No_balance
9998       below_80k
9999    below_1.4lac
Name: Balance, Length: 10000, dtype: category
Categories (27, object): ['No_balance' < 'below_10k' < 'below_20k' < 'belo
w_30k' ... 'below_2.3lac' < 'below_2.4lac' < 'below_2.5lac' < 'below_2.6la
c']
```

In [30]:

```
df.insert(6,'Balance_label',Category)
```

In [31]:

```
df.sample(5)
```

Out[31]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	Balance_label	NumOfProdu
2719	569	Germany	Female	42	9	146100.75	below_1.5lac	
5597	670	France	Female	42	6	112333.63	below_1.2lac	
1688	601	France	Female	41	1	0.00	No_balance	
611	650	France	Female	27	6	0.00	No_balance	
440	626	France	Female	35	3	0.00	No_balance	

In [32]:

```
# to perform above operation we need max value in Balance and Salary
mean = df.max(axis=0)
print(mean)
```

```
CreditScore                850
Geography                Spain
Gender                    Male
Age                         92
Tenure                      10
Balance              250898.09
Balance_label      below_2.6lac
NumOfProducts                4
HasCrCard                    1
IsActiveMember               1
EstimatedSalary      199992.48
Exited                       1
dtype: object
```

In [33]:

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

Out[33]:

```
No_balance      3617
below_1.3lac     898
below_1.2lac     832
below_1.1lac     786
below_1.4lac     734
below_1lac       599
below_1.5lac     580
below_90k        400
below_1.6lac     386
below_80k        274
below_1.7lac     264
below_1.8lac     159
below_70k        156
below_1.9lac      86
below_60k         80
below_50k         46
below_2lac        40
below_2.1lac      21
below_40k         17
below_2.2lac       9
below_30k          8
below_20k          3
below_2.3lac       2
below_10k          1
below_2.4lac       1
below_2.6lac       1
below_2.5lac       0
Name: Balance_label, dtype: int64
```

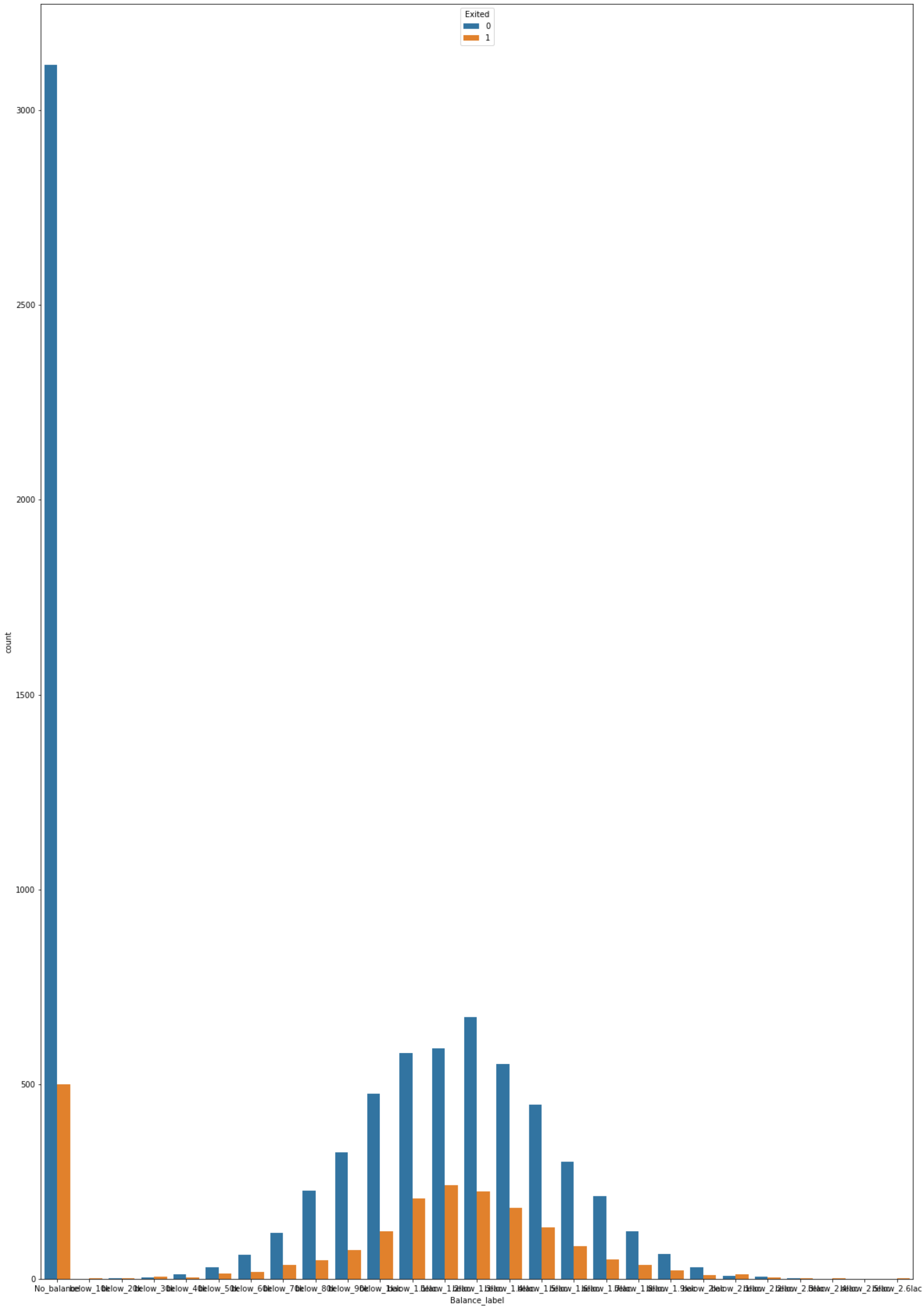
From This we can see the balance label v/s exited content

In [34]:

```python
# Now we made the balance feature as required for visualization
plt.figure(figsize=(20,30))

sns.countplot(data=df,x='Balance_label',hue='Exited')
```

Out[34]:

```
<AxesSubplot:xlabel='Balance_label', ylabel='count'>
```



SAME PROCESS WE WILL APPLY ON eSTIMATED SALARY FEATURE

In [35]:

```
mean = df.min(axis=0)
print(mean)
```

```
CreditScore               350
Geography              France
Gender                 Female
Age                        18
Tenure                      0
Balance                   0.0
Balance_label      No_balance
NumOfProducts               1
HasCrCard                   0
IsActiveMember              0
EstimatedSalary         11.58
Exited                      0
dtype: object
```

In [36]:

```
Category2=pd.cut(df.EstimatedSalary,bins=[0,10000,20000,30000,40000,50000,60000,70000,80
               labels=['below_10k','below_20k','below_30k','below_40k','below_50k','belo
                       'below_70k','below_80k','below_90k','below_1lac','below_1.1lac','
                       'below_1.4lac','below_1.5lac','below_1.6lac','below_1.7lac','belo
                       'below_2lac'])
```

In [37]:

```
Category2
```

Out[37]:

```
0       below_1.1lac
1       below_1.2lac
2       below_1.2lac
3         below_1lac
4          below_80k
            ...
9995      below_1lac
9996    below_1.1lac
9997       below_50k
9998      below_1lac
9999       below_40k
Name: EstimatedSalary, Length: 10000, dtype: category
Categories (20, object): ['below_10k' < 'below_20k' < 'below_30k' < 'below
_40k' ... 'below_1.7lac' < 'below_1.8lac' < 'below_1.9lac' < 'below_2lac']
```

In [38]:

```
df.insert(11,'SalaryRange',Category2)
```

In [39]:

```
df.head(2)
```

Out[39]:

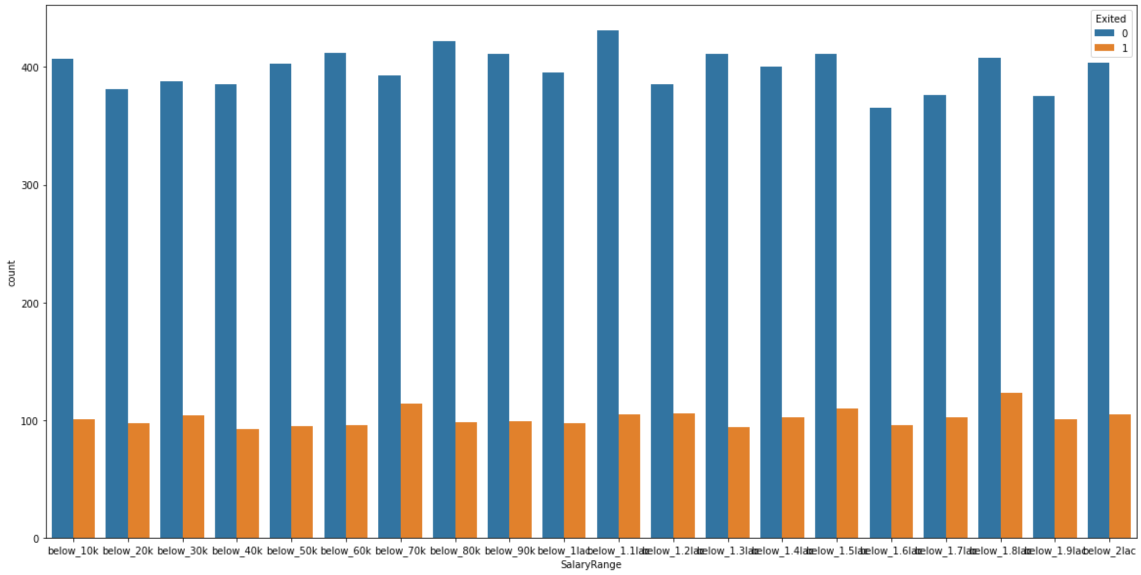
	CreditScore	Geography	Gender	Age	Tenure	Balance	Balance_label	NumOfProducts
0	619	France	Female	42	2	0.00	No_balance	1
1	608	Spain	Female	41	1	83807.86	below_90k	1

In [40]:

```
plt.figure(figsize=(20,10))
sns.countplot(data=df,x='SalaryRange',hue='Exited')# delete
```

Out[40]:

<AxesSubplot:xlabel='SalaryRange', ylabel='count'>



Till Now We Have Finished EDA Work So From Now We Will Drop Additional Parameters Which We Added For Visualization Purpose

In [41]:

```
df.head(5)
```

Out[41]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	Balance_label	NumOfProducts
0	619	France	Female	42	2	0.00	No_balance	1
1	608	Spain	Female	41	1	83807.86	below_90k	1
2	502	France	Female	42	8	159660.80	below_1.6lac	3
3	699	France	Female	39	1	0.00	No_balance	2
4	850	Spain	Female	43	2	125510.82	below_1.3lac	1

In [42]:

```
# balance_label and SalaryRange has to be dropped
df.drop(['Balance_label','SalaryRange'],axis=1)
```

Out[42]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	619	France	Female	42	2	0.00	1	1
1	608	Spain	Female	41	1	83807.86	1	0
2	502	France	Female	42	8	159660.80	3	1
3	699	France	Female	39	1	0.00	2	0
4	850	Spain	Female	43	2	125510.82	1	1
...	...	...	...	...	...	...	...	...
9995	771	France	Male	39	5	0.00	2	1
9996	516	France	Male	35	10	57369.61	1	1
9997	709	France	Female	36	7	0.00	1	0
9998	772	Germany	Male	42	3	75075.31	2	1
9999	792	France	Female	28	4	130142.79	1	1

10000 rows × 11 columns

In [43]:

```
df.describe()
```

Out[43]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCa
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000(
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.705
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.455
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.000(
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.000(
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.000(
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.000(
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.000(

Now We Will See outliers in our parameters

In [44]:

```
fig=px.histogram(df,x='CreditScore')
fig.show()
```

In [45]:

```
fig=px.box(df,x='CreditScore')
fig.show()
```

From box plotting we can se that some outliers are there below 382 credit score

Now we will try to fix outliers in creditScore

In [46]:

```
# Finding The IQR
percentile25=df['CreditScore'].quantile(0.25)
percentile75=df['CreditScore'].quantile(0.75)
```

In [47]:

```
percentile75
```

Out[47]:

```
718.0
```

In [48]:

```
percentile25
```

Out[48]:

```
584.0
```

In [49]:

```
iqr=percentile75-percentile25
iqr
```

Out[49]:

```
134.0
```

In [50]:

```
upper_limit=percentile75+1.5*iqr
lower_limit=percentile25-1.5*iqr
```

In [51]:

```
print(upper_limit)
print(lower_limit)
```

```
919.0
383.0
```

Finding Outliers

In [52]:

```
df[df['CreditScore']<lower_limit]
```

Out[52]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	Balance_label	NumOfProdu
7	376	Germany	Female	29	4	115046.74	below_1.2lac	
942	376	France	Female	46	6	0.00	No_balance	
1193	363	Spain	Female	28	6	146098.43	below_1.5lac	
1405	359	France	Female	44	6	128747.69	below_1.3lac	
1631	350	Spain	Male	54	1	152677.48	below_1.6lac	
1838	350	Germany	Male	39	0	109733.20	below_1.1lac	
1962	358	Spain	Female	52	8	143542.36	below_1.5lac	
2473	351	Germany	Female	57	4	163146.46	below_1.7lac	
2579	365	Germany	Male	30	0	127760.07	below_1.3lac	
8154	367	Spain	Male	42	6	93608.28	below_1lac	
8723	350	France	Male	51	10	0.00	No_balance	
8762	350	France	Female	60	3	0.00	No_balance	
9210	382	Spain	Male	36	0	0.00	No_balance	
9356	373	France	Male	42	7	0.00	No_balance	
9624	350	France	Female	40	0	111098.85	below_1.2lac	

In [53]:

```
df[df['CreditScore']>upper_limit]
```

Out[53]:

CreditScore	Geography	Gender	Age	Tenure	Balance	Balance_label	NumOfProducts	H

In [54]:

```
# deleting balance label and estimated salary
df=df.drop(['Balance_label','SalaryRange'],axis=1)
```

Now We Will do Trimming To Delete Those Columns Which Have Outliers

In [55]:

```
new_df=df[df['CreditScore']>lower_limit]
```

In [56]:

```
new_df.shape
```

Out[56]:

(9984, 11)

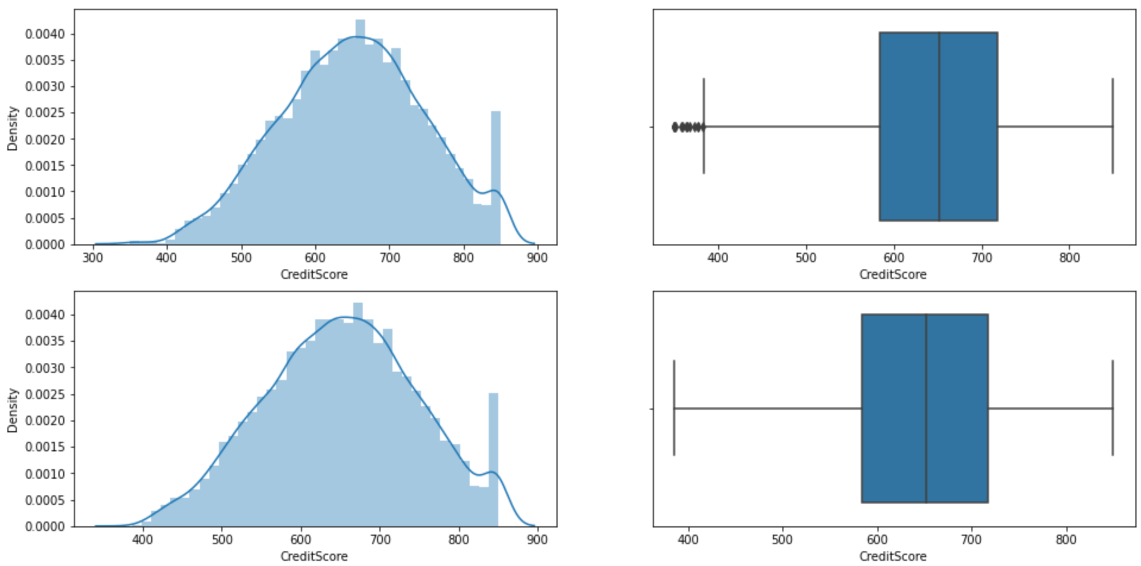
Comparing

In [57]:

```
plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
sns.distplot(df['CreditScore'])

plt.subplot(2,2,2)
sns.boxplot(df['CreditScore'])

plt.subplot(2,2,3)
sns.distplot(new_df['CreditScore'])

plt.subplot(2,2,4)
sns.boxplot(new_df['CreditScore'])
```

Out[57]:

<AxesSubplot:xlabel='CreditScore'>



So from the above graph we can see outliers has been removed from CreditScore column

Now we will apply this in all continuos parameters

In [58]:

```
new_df.head(5)
```

Out[58]:

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

In [59]:

```
# Age
percentile25=df['Age'].quantile(0.25)
percentile75=df['Age'].quantile(0.75)
```

In [60]:

```
print(percentile25)
print(percentile75)
```

```
32.0
44.0
```

In [61]:

```
iqr=percentile75-percentile25
```

In [62]:

```
iqr
```

Out[62]:

```
12.0
```

In [63]:

```
upperlimit=percentile75+1.5*iqr
lowerlimit=percentile25-1.5*iqr
```

In [64]:

```
print(lowerlimit)
print(upperlimit)
```

```
14.0
62.0
```

In [65]:

```
df[df['Age']<lowerlimit]
```

Out[65]:

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAct

In [66]:

```
df[df['Age']>upperlimit]
```

Out[66]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
58	511	Spain	Female	66	4	0.00	1	1
85	652	Spain	Female	75	10	0.00	2	1
104	670	Spain	Female	65	1	0.00	1	1
158	646	France	Female	73	6	97259.25	1	0
181	510	France	Male	65	2	0.00	2	1
...	...	...	...	...	...	...	...	...
9753	656	Germany	Male	68	7	153545.11	1	1
9765	445	France	Male	64	2	136770.67	1	0
9832	595	Germany	Female	64	2	105736.32	1	1
9894	521	France	Female	77	6	0.00	2	1
9936	609	France	Male	77	1	0.00	1	0

359 rows × 11 columns

In [67]:

```
# Trimming

new_df1=df[df['Age']<upperlimit]
```

In [68]:

```
new_df1.shape
```

Out[68]:

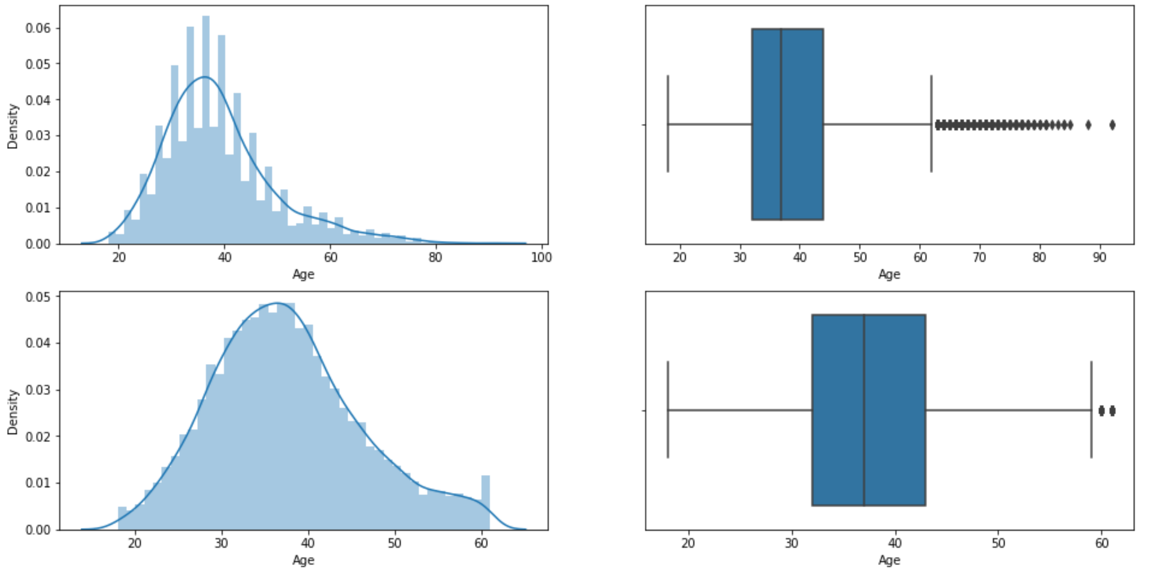
```
(9589, 11)
```

In [69]:

```
# COMPARING
plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
sns.distplot(df['Age'],kde=True)

plt.subplot(2,2,2)
sns.boxplot(df['Age'])

plt.subplot(2,2,3)
sns.distplot(new_df1['Age'])

plt.subplot(2,2,4)
sns.boxplot(new_df1['Age'])
```

Out[69]:

```
<AxesSubplot:xlabel='Age'>
```



In [70]:

```
# For Balance also we will do
percentile25=new_df1['Balance'].quantile(0.25)
percentile75=new_df1['Balance'].quantile(0.75)
```

In [71]:

```
print(percentile25)
print(percentile75)
```

```
0.0
127661.69
```

In [72]:

```
iqr=percentile75-percentile25
iqr
```

Out[72]:

```
127661.69
```

In [73]:

```
upperlimit_balance=percentile75+1.5*iqr
lowerlimit_balance=percentile25-1.5*iqr
```

In [74]:

```
print(upperlimit_balance)
print(lowerlimit_balance)
```

```
319154.225
-191492.535
```

In [75]:

```
new_df1[new_df1['Balance']<lowerlimit_balance]
```

Out[75]:

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAct

In [76]:

```
new_df1[new_df1['Balance']>upperlimit_balance]
```

Out[76]:

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAct

SO IT CAN BE SEEN THAT IN BALANCE PARAMETER THERE ARE NO OUTLIERS

OUTLIER REMOVAL FOR ESTIMATED SALARY USING Z-SCORE

In [77]:

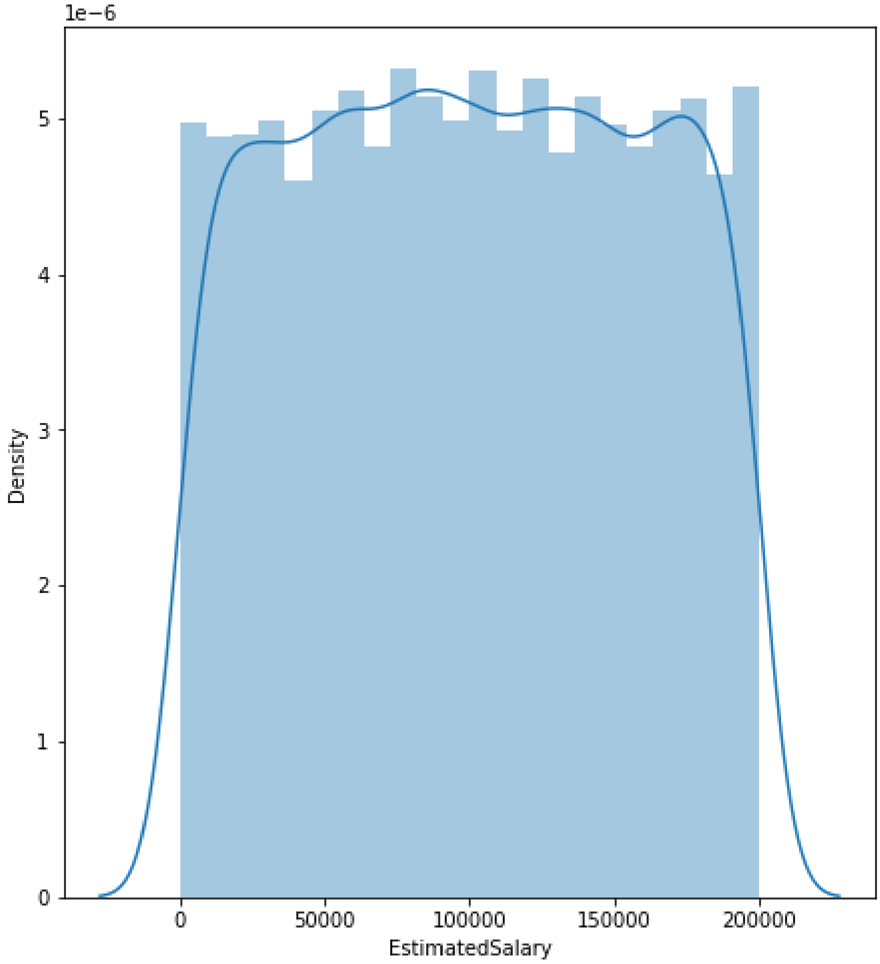
```
new_df1.head(2)
```

Out[77]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsA
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	

In [78]:

```
# FOR USING Z-SCORE WE MUST SURE THAT PARAMETER IS FOLLOWING NORMAL DISTRIBUTION PATTERN
plt.figure(figsize=(16,8))

plt.subplot(1,2,1)
sns.distplot(new_df1['EstimatedSalary'])

plt.show()
```



So it can be seen that this parameter is not following normal distribution. so, now we will apply iqr method

In [79]:

```
percentile25=new_df1['EstimatedSalary'].quantile(0.25)
percentile75=new_df1['EstimatedSalary'].quantile(0.75)
```

In [80]:

```
print(percentile25)
print(percentile75)
```

```
51226.32
149458.73
```

In [81]:

```
iqr=percentile75-percentile25
print(iqr)
```

```
98232.41
```

In [82]:

```
upperlimit_salary=percentile75+1.5*iqr
lowerlimit_salary=percentile25-1.5*iqr
```

In [83]:

```
print(upperlimit_salary)
print(lowerlimit_salary)
```

```
296807.345
-96122.29499999998
```

In [84]:

```
new_df1[new_df1['EstimatedSalary']<lowerlimit_salary]
```

Out[84]:

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAct

In [85]:

```
new_df1[new_df1['EstimatedSalary']>upperlimit_salary]
```

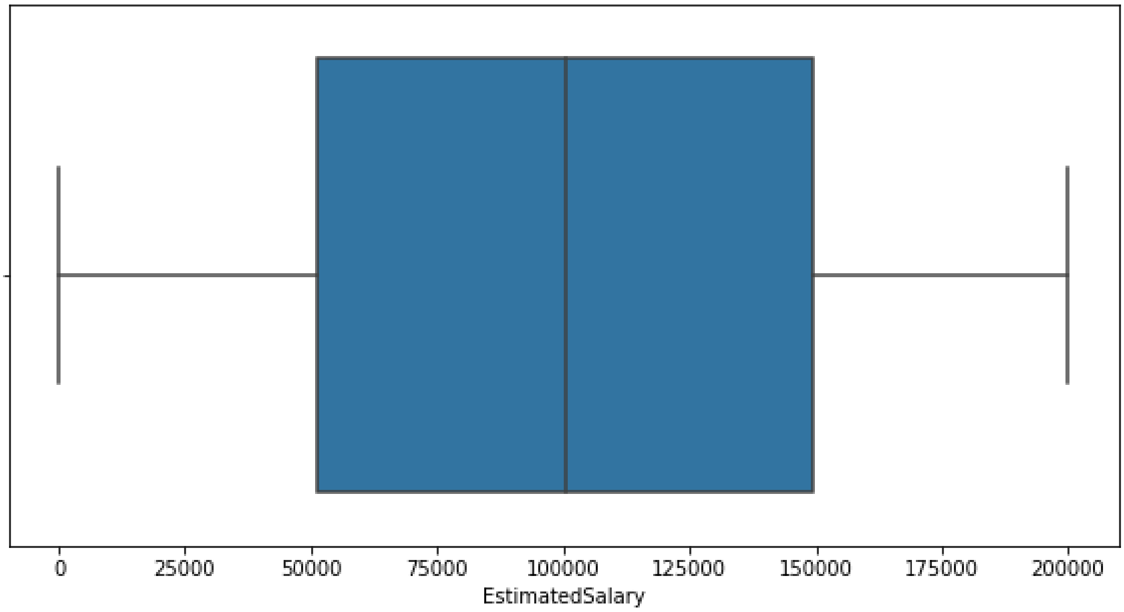
Out[85]:

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAct

So, from applied maths and box plot we can see that there are no outliers present in estimated salary parameter

In [86]:

```
plt.figure(figsize=(10,5))

sns.boxplot(new_df1['EstimatedSalary'])
plt.show()
```



Now we have to handle imbalanced features specially in categorical parameters

In [87]:

```
new_df1.columns
```

Out[87]:

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

In [88]:

```
new_df1.dtypes
```

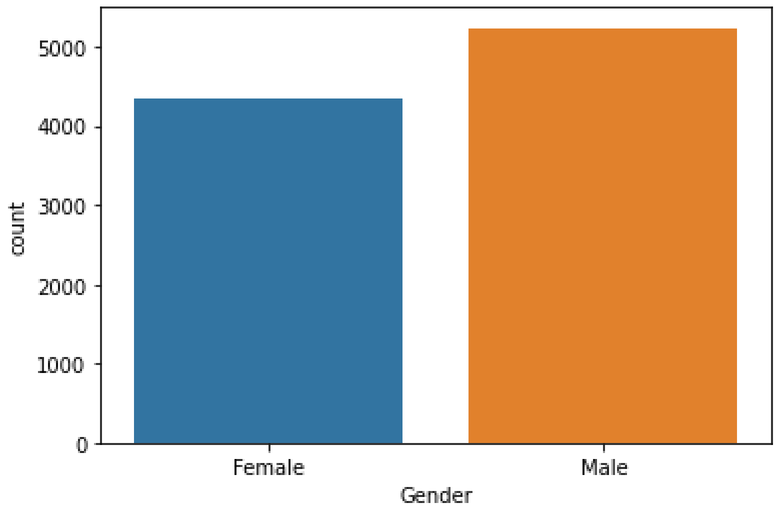
Out[88]:

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

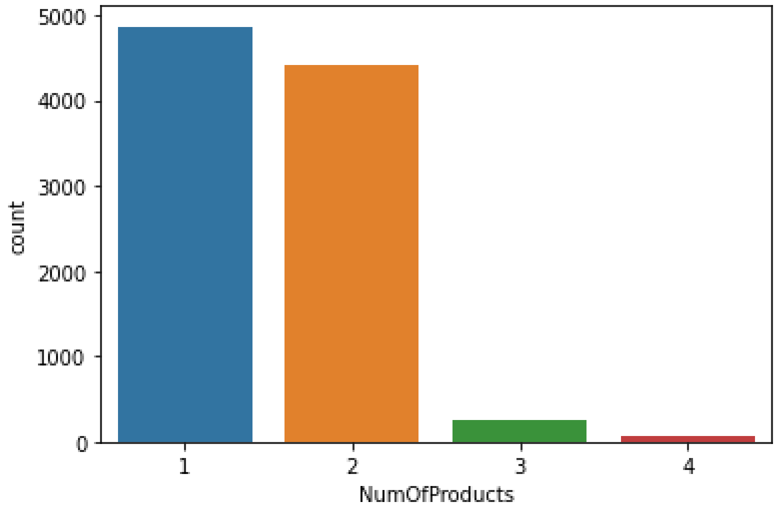
In [89]:

```
for i in new_df1[['Gender','NumOfProducts','HasCrCard','IsActiveMember','Exited']]:

    fig,ax=plt.subplots()
    feature_plot=sns.countplot(data=new_df1,x=i,ax=ax)
    print(feature_plot)
    plt.show()
```

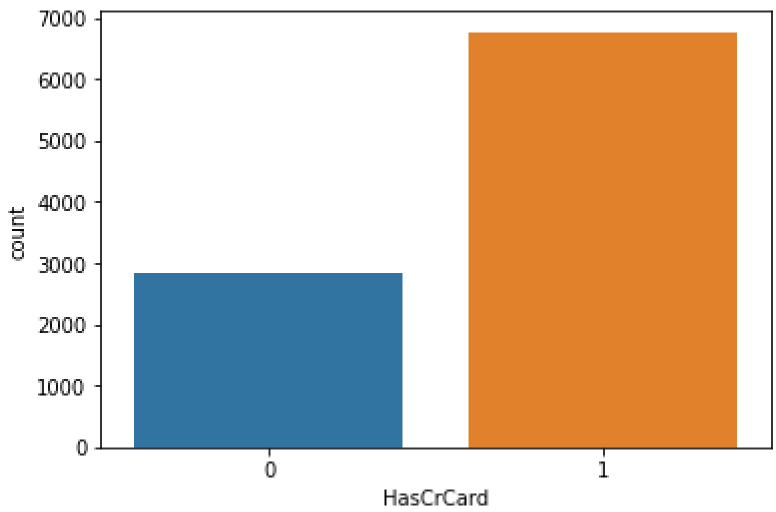
AxesSubplot(0.125,0.125;0.775x0.755)



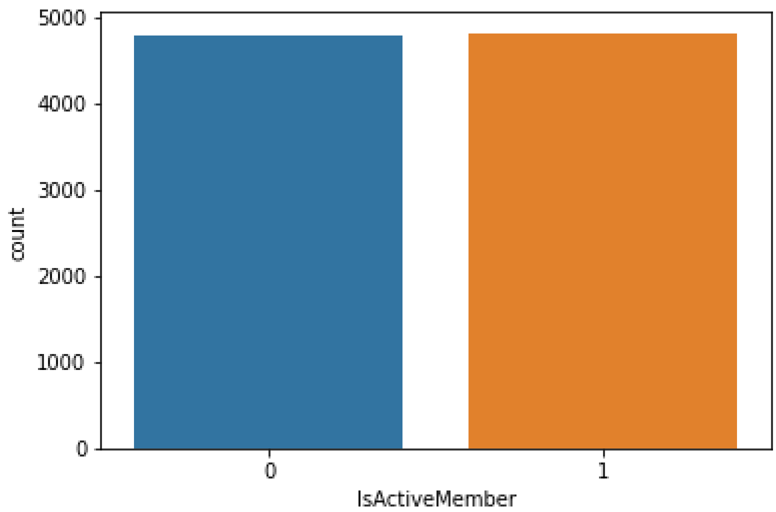
AxesSubplot(0.125,0.125;0.775x0.755)



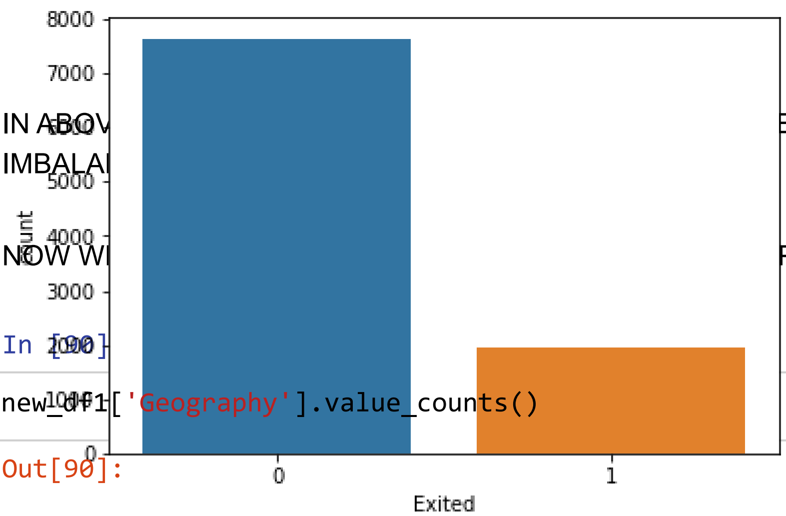
AxesSubplot(0.125,0.125;0.775x0.755)



AxesSubplot(0.125,0.125;0.775x0.755)



AxesSubplot(0.125,0.125;0.775x0.755)



IN ABOV EXITED, NUMBER OF PRODUCTS HAS IMBALAI

NOW WI PHY AND GENDER PARAMETER

In [90]:

```
new_df1['Geography'].value_counts()
```

Out[90]:

```
France     4808
Germany    2411
Spain      2370
Name: Geography, dtype: int64
```

In [91]:

```
new_df1['Gender'].value_counts()
```

Out[91]:

```
Male      5236
Female    4353
Name: Gender, dtype: int64
```

In [92]:

```
# applying label encoding
labelenc=LabelEncoder()
```

In [93]:

```
new_df1[['Geography','Gender']]=new_df1[['Geography','Gender']].apply(LabelEncoder().fit
```

In [94]:

```
new_df1.head(5)
```

Out[94]:

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

In [95]:

```
new_df1.tail(5)
```

Out[95]:

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

Type *Markdown* and LaTeX: α^2

Now We Will Do Feature Scaling Task First

In [96]:

```
# we will use either of the two that is normalization or standardization
# standardization main point is to have normal distribution type of data
# normalization when we are not sure about data distribution
# to check which scaling we have to use first we will check the distribution of our data
# we can say that not all the parameters are normally distributed.So, we will go with no

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

In [97]:

```
x=new_df1.drop('Exited',axis=1)
y=new_df1['Exited']
```

In [98]:

```
print('shape of x -',x.shape)
print('shape of y -',y.shape)
```

```
shape of x - (9589, 10)
shape of y - (9589,)
```

In [99]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=45)
```

In [100]:

```
print('shape of x_train', x_train.shape)
print('shape of x_test', x_test.shape)
print('shape of y_train', y_train.shape)
print('shape of y_test', y_test.shape)
```

```
shape of x_train (7191, 10)
shape of x_test (2398, 10)
shape of y_train (7191,)
shape of y_test (2398,)
```

In [101]:

```
mms=MinMaxScaler()
```

In [102]:

```
mms.fit(x_train)
```

Out[102]:

```
MinMaxScaler()
```

In [103]:

```
x_train_mms=mms.transform(x_train)
x_test_mms= mms.transform(x_test)
```

In [104]:

```
x_train_mms
```

Out[104]:

```
array([[0.686     , 0.        , 1.        , ..., 1.        , 1.        ,
        0.50938889],
       [0.844     , 0.5       , 0.        , ..., 0.        , 1.        ,
        0.23303447],
       [0.634     , 0.        , 1.        , ..., 1.        , 1.        ,
        0.65904212],
       ...,
       [0.604     , 1.        , 0.        , ..., 1.        , 1.        ,
        0.09610285],
       [0.628     , 0.        , 1.        , ..., 1.        , 0.        ,
        0.97386113],
       [0.76      , 0.5       , 0.        , ..., 1.        , 0.        ,
        0.59279383]])
```

In [105]:

```
x_test_mms
```

Out[105]:

```
array([[0.682     , 0.        , 1.        , ..., 0.        , 0.        ,
        0.2229194 ],
       [0.888     , 0.        , 1.        , ..., 1.        , 1.        ,
        0.35860304],
       [0.73      , 0.5       , 1.        , ..., 1.        , 1.        ,
        0.10626009],
       ...,
       [0.684     , 1.        , 0.        , ..., 1.        , 1.        ,
        0.18912967],
       [0.49      , 0.        , 1.        , ..., 1.        , 0.        ,
        0.43861426],
       [0.534     , 0.        , 0.        , ..., 0.        , 1.        ,
        0.84413213]])
```

In [106]:

```
new_df1.columns
```

Out[106]:

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

NOW WE HAVE TO CONVERT THIS 2D ARRAY INTO DATAFRAME

In [107]:

```
x_train_mms=pd.DataFrame(x_train_mms,columns=['CreditScore','Geography','Gender','Age','

x_test_mms=pd.DataFrame(x_test_mms,columns=['CreditScore','Geography','Gender','Age','Te
```

In [108]:

```
x_train_mms
```

Out[108]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrC
0	0.686	0.0	1.0	0.511628	0.4	0.587859	0.000000	
1	0.844	0.5	0.0	0.767442	0.9	0.647557	0.000000	
2	0.634	0.0	1.0	0.348837	0.4	0.000000	0.333333	
3	0.512	0.5	1.0	0.697674	0.4	0.595694	0.000000	
4	0.454	1.0	1.0	0.302326	0.6	0.000000	0.000000	
...	...	...	...	...	...	...	...	
7186	0.288	0.0	0.0	0.395349	0.9	0.507168	0.333333	
7187	0.530	1.0	0.0	0.534884	0.9	0.490459	0.000000	
7188	0.604	1.0	0.0	0.418605	0.1	0.000000	0.333333	
7189	0.628	0.0	1.0	0.511628	0.9	0.000000	0.333333	
7190	0.760	0.5	0.0	0.441860	0.5	0.558125	0.000000	

7191 rows × 10 columns

In [109]:

```
datatoconcatenate=[x_train_mms,x_test_mms]
data=pd.concat(datatoconcatenate)
```

In [110]:

```
data.head(5)
```

Out[110]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	0.686	0.0	1.0	0.511628	0.4	0.587859	0.000000	1.0
1	0.844	0.5	0.0	0.767442	0.9	0.647557	0.000000	0.0
2	0.634	0.0	1.0	0.348837	0.4	0.000000	0.333333	1.0
3	0.512	0.5	1.0	0.697674	0.4	0.595694	0.000000	0.0
4	0.454	1.0	1.0	0.302326	0.6	0.000000	0.000000	1.0

In [111]:

```
data.shape
```

Out[111]:

```
(9589, 10)
```

In [112]:

```
y
```

Out[112]:

```
0       1
1       0
2       1
3       0
4       0
       ..
9995    0
9996    0
9997    1
9998    1
9999    0
Name: Exited, Length: 9589, dtype: int64
```

In [113]:

```
y=pd.DataFrame(y,columns=['Exited'])
```

In [114]:

```
y
```

Out[114]:

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

9589 rows × 1 columns

There was some problem with concatenation so we directly append exited column now will proceed with

In [115]:

```
data['Exited']=y
```

In [116]:

```
data.head(60)
```

Out[116]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCar
0	0.686	0.0	1.0	0.511628	0.4	0.587859	0.000000	1.
1	0.844	0.5	0.0	0.767442	0.9	0.647557	0.000000	0.
2	0.634	0.0	1.0	0.348837	0.4	0.000000	0.333333	1.
3	0.512	0.5	1.0	0.697674	0.4	0.595694	0.000000	0.
4	0.454	1.0	1.0	0.302326	0.6	0.000000	0.000000	1.
5	0.580	0.0	0.0	0.558140	0.5	0.792284	0.000000	1.
6	0.568	0.0	1.0	0.441860	0.5	0.518950	0.333333	0.
7	0.722	0.0	0.0	0.534884	0.3	0.655763	0.000000	1.
8	0.504	0.5	1.0	0.395349	0.7	0.407628	0.333333	1.
9	0.358	1.0	1.0	0.395349	0.5	0.000000	0.333333	1.
10	1.000	0.5	1.0	0.813953	0.2	0.423269	0.333333	1.
11	0.494	1.0	0.0	0.186047	0.7	0.000000	0.333333	1.
12	0.494	0.0	0.0	0.186047	0.8	0.674814	0.000000	1.
13	0.434	0.0	1.0	0.953488	0.3	0.000000	0.333333	1.
14	0.504	0.0	1.0	0.395349	0.8	0.000000	0.000000	1.
15	0.144	1.0	0.0	0.837209	0.4	0.000000	0.333333	1.
16	0.270	1.0	1.0	0.325581	0.6	0.459977	0.333333	1.
17	0.800	0.0	1.0	0.093023	0.5	0.000000	0.333333	0.
18	0.384	0.0	0.0	0.953488	0.2	0.309954	0.333333	1.
19	0.598	0.0	1.0	0.488372	0.3	0.508830	0.000000	1.
20	0.740	0.5	1.0	0.651163	0.3	0.436602	0.000000	1.
21	0.712	1.0	0.0	0.558140	0.8	0.429153	0.000000	1.
22	0.608	0.0	0.0	0.441860	0.5	0.000000	0.000000	0.
23	0.968	1.0	1.0	0.372093	0.5	0.000000	0.333333	0.
24	0.648	0.5	1.0	0.372093	0.2	0.687450	0.000000	1.
25	0.688	0.0	0.0	0.465116	0.5	0.881489	0.000000	1.
26	0.500	0.0	1.0	0.720930	0.7	0.405902	0.000000	1.
27	0.540	0.0	1.0	0.232558	0.9	0.323495	0.000000	0.
28	0.630	0.0	0.0	0.860465	0.8	0.613468	0.000000	1.
29	0.706	0.5	0.0	0.255814	0.3	0.549269	0.000000	0.
30	0.626	0.0	1.0	0.372093	0.7	0.000000	0.333333	1.
31	0.850	1.0	1.0	0.255814	1.0	0.000000	0.333333	1.
32	0.612	1.0	0.0	0.511628	1.0	0.755299	0.000000	0.
33	0.724	1.0	1.0	0.813953	0.2	0.499672	0.333333	0.
34	0.434	0.5	0.0	0.651163	0.1	0.307011	0.333333	1.
35	0.852	0.5	1.0	0.465116	0.5	0.505164	0.000000	0.
36	0.592	0.0	1.0	0.465116	0.7	0.000000	0.333333	1.

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCar
37	0.722	0.0	1.0	0.348837	0.1	0.000000	0.000000	0.
38	0.642	1.0	0.0	0.372093	1.0	0.000000	0.000000	1.
39	0.310	1.0	1.0	0.395349	0.8	0.526089	0.000000	1.
40	0.512	0.5	1.0	0.209302	0.2	0.586114	0.333333	1.
41	0.402	0.0	0.0	0.953488	0.2	0.751204	0.000000	1.
42	0.796	0.0	1.0	0.465116	0.4	0.518390	0.000000	0.
43	0.422	0.0	0.0	0.093023	0.6	0.840379	0.333333	1.
44	0.400	1.0	1.0	0.372093	0.3	0.000000	0.333333	0.
45	0.536	0.5	0.0	0.511628	0.1	0.599482	0.333333	1.
46	0.362	0.0	0.0	0.441860	0.1	0.000000	0.000000	1.
47	0.540	1.0	1.0	0.604651	0.8	0.000000	0.333333	1.
48	0.542	0.0	1.0	0.488372	0.6	0.000000	0.333333	1.
49	0.736	0.0	1.0	0.697674	0.9	0.000000	0.333333	1.
50	0.668	1.0	0.0	0.418605	0.4	0.000000	0.000000	1.
51	0.370	0.5	1.0	0.465116	0.8	0.573521	0.000000	0.
52	0.590	1.0	1.0	0.604651	0.8	0.511796	0.333333	1.
53	0.222	1.0	0.0	0.162791	0.6	0.000000	0.333333	1.
54	0.658	0.0	0.0	0.581395	0.4	0.000000	0.666667	1.
55	0.656	0.0	0.0	0.744186	0.6	0.000000	0.000000	1.
56	0.228	0.0	1.0	0.651163	0.6	0.727945	0.000000	1.
57	0.454	1.0	1.0	0.534884	0.4	0.400488	0.000000	0.
58	0.710	0.0	0.0	0.139535	0.7	0.450671	0.000000	1.
59	0.414	1.0	0.0	0.511628	0.4	0.000000	0.333333	0.

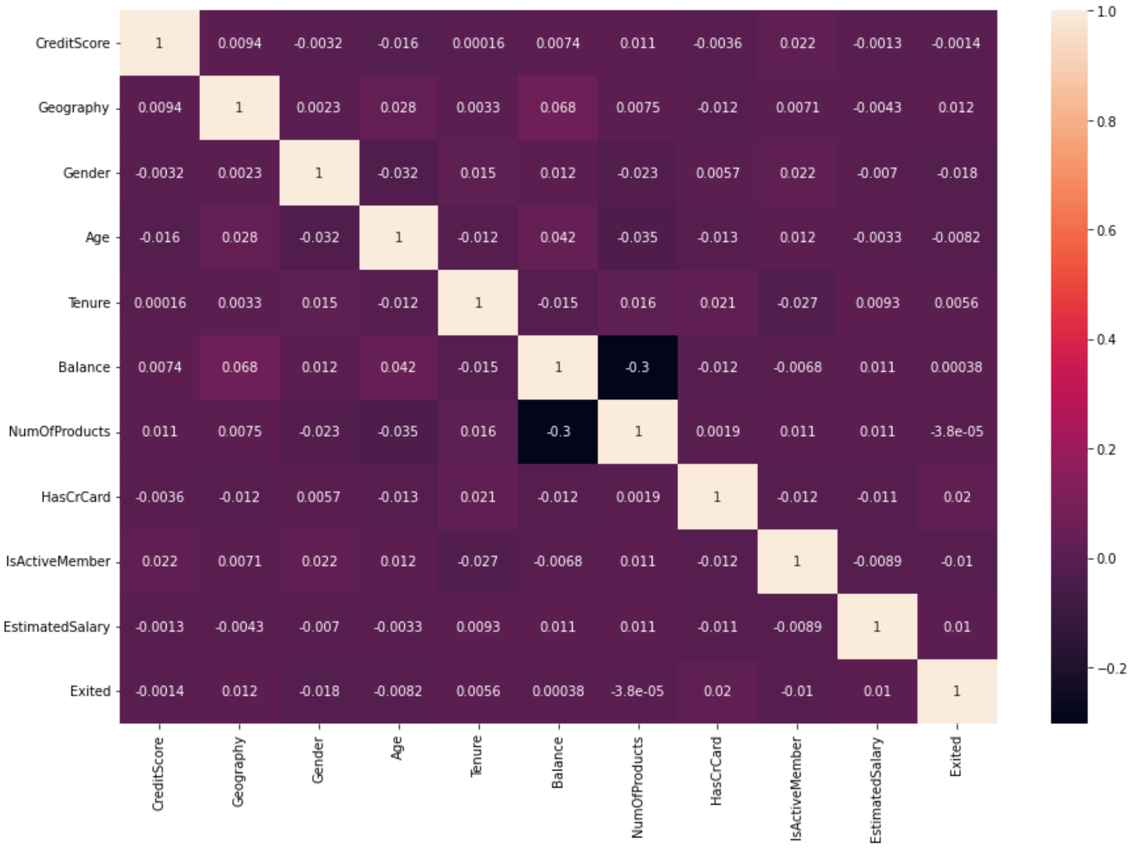
Now We Will Proceed With Feature Selection

In [117]:

```
# using HeatMap
corr=data.corr()
plt.figure(figsize=(15,10))
sns.heatmap(corr,annot=True)
```

Out[117]:

```
<AxesSubplot:>
```



FROM HEATMAP WE ARE NOT GETTING MUCH IDEA SO WE WILL PROCEED WITH OTHER FUNCTIONS

In [118]:

```
from sklearn.feature_selection import VarianceThreshold #Understand from sklearn documen
```

In [119]:

```
var=VarianceThreshold(threshold=0.1) # we will go with 0.08
var.fit(data)
var.get_support()
```

Out[119]:

```
array([False,  True,  True, False, False, False, False,  True,  True,
       False,  True])
```

Observation

From variance threshold we will select high variance values and avoid having low variance values So, selected features must be

1. Geography
2. Gender
3. Tenure
4. HasCrCreditCard
5. IsActivemember
6. estimated salary

ANNOVA F-TEST AND CHI-SQUARE TEST

In [120]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import chi2 # chi-2 test basically works with categorical
from sklearn.feature_selection import f_classif # The higher the f-value and lower the p
```

In [121]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9589 entries, 0 to 2397
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CreditScore      9589 non-null   float64
 1   Geography        9589 non-null   float64
 2   Gender           9589 non-null   float64
 3   Age              9589 non-null   float64
 4   Tenure           9589 non-null   float64
 5   Balance          9589 non-null   float64
 6   NumOfProducts    9589 non-null   float64
 7   HasCrCard        9589 non-null   float64
 8   IsActiveMember   9589 non-null   float64
 9   EstimatedSalary  9589 non-null   float64
 10  Exited           9210 non-null   float64
dtypes: float64(11)
memory usage: 899.0 KB
```

In [122]:

```
x=data[['Geography','Gender','HasCrCard','IsActiveMember']]
```

In [123]:

```
y=data['Exited']
```

In [124]:

```
np.isnan(y).sum() # to avoid issue for now we will add values in alternate of nan values
```

Out[124]:

```
379
```

In [125]:

```
np.isnan(x).sum()
```

Out[125]:

```
Geography         0
Gender            0
HasCrCard         0
IsActiveMember    0
dtype: int64
```

In [126]:

```
y_new=y.fillna(0.0)
```

In [127]:

```
# checking again
np.isnan(y_new).sum()
```

Out[127]:

```
0
```

In [128]:

```
# now we will divide the dataset in to train test split to reduce the dataset size
from sklearn.model_selection import train_test_split
```

In [129]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y_new,test_size=0.3,random_state=98)
```

In [130]:

```
# checking the shape of test and train size
print('x_train - ',x_train.shape)
print('x_test - ',x_test.shape)
print('y_train - ',y_train.shape)
print('y_test - ',y_test.shape)
```

```
x_train -  (6712, 4)
x_test -  (2877, 4)
y_train -  (6712,)
y_test -  (2877,)
```

In [131]:

```
f_p_value=chi2(x_train,y_train)
```

In [132]:

```
# First value would be f-score and second value p-value
f_p_value
```

Out[132]:

```
(array([0.2138678 , 1.39043629, 0.44675144, 2.43250454]),
 array([0.64375261, 0.23833143, 0.50388169, 0.11884289]))
```

In [133]:

```
# applying in full dataset
f_pvalue2=chi2(x,y_new)
```

In [134]:

```
f_pvalue2# use specific tests only
```

Out[134]:

```
(array([0.77965014, 1.25579242, 1.00125753, 0.58031065]),
 array([0.37724816, 0.26244903, 0.31700641, 0.44619058]))
```

Observation

At first we got error to avoid nan values from our input dataset Also we can directly go with full dataset and will select the features accordingly

So, according to the values we get we will choose higher f-value and less p-value

selected parameters will be ----- Geography,Gender,HasCrCard

NOW WE WILL PROCEED WITH ONE MORE TECHNIQUE

1. INFORMATION GAIN condition ------ we basically put all the continuous variables in independent feature and categorical will be the target. the more the value its importance will be more.

In [135]:

```
data.columns # we will not taking the columns like tenure and numof products as they hav
```

Out[135]:

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

In [136]:

```
from sklearn.feature_selection import mutual_info_classif
features=data[['CreditScore','Age','Balance','EstimatedSalary']]
target=y_new # here we have used y_new because it has no nan values ---otherwise it will
feature_scores=mutual_info_classif(features,target,random_state=0)
feature_scores
```

Out[136]:

```
array([0.00100946, 0.        , 0.        , 0.        ])
```

In [137]:

```
feature_scores=mutual_info_classif(features,target,random_state=100)
feature_scores
```

Out[137]:

```
array([0.        , 0.00052534, 0.        , 0.        ])
```

In [138]:

```
feature_scores=mutual_info_classif(features,target,random_state=52)
feature_scores
```

Out[138]:

```
array([0.00289708, 0.00644824, 0.0026638 , 0.        ])
```

observations

So from information gain we are not getting much idea 1-2 parameters only we identified as age and balance

Based on above operations we will be creating 2 different dataframes according to parameters and will proceed with other operations.

1. dataset1=[Age, balance, geography, gender, hascrcard, tenure, isactive member, estimated salary]
2. dataset2=[geography,gender,tenure, hascrcard, isactivemember, estimated_salary]

In [139]:

```
# first we will convert our trget feature into int
data.dtypes
```

Out[139]:

```
CreditScore        float64
Geography          float64
Gender             float64
Age                float64
Tenure             float64
Balance            float64
NumOfProducts      float64
HasCrCard          float64
IsActiveMember     float64
EstimatedSalary    float64
Exited             float64
dtype: object
```

In [140]:

```
data['Exited']=data['Exited'].fillna(0)
```

In [141]:

```
data['Exited'].value_counts()
```

Out[141]:

```
0.0    7693
1.0    1896
Name: Exited, dtype: int64
```

In [142]:

```
data['Exited']=data['Exited'].astype(int)
data.dtypes
```

Out[142]:

```
CreditScore        float64
Geography          float64
Gender             float64
Age                float64
Tenure             float64
Balance            float64
NumOfProducts      float64
HasCrCard          float64
IsActiveMember     float64
EstimatedSalary    float64
Exited               int32
dtype: object
```

In [143]:

```
# changes has been done!
data.head(10)
```

Out[143]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	0.686	0.0	1.0	0.511628	0.4	0.587859	0.000000	1.0
1	0.844	0.5	0.0	0.767442	0.9	0.647557	0.000000	0.0
2	0.634	0.0	1.0	0.348837	0.4	0.000000	0.333333	1.0
3	0.512	0.5	1.0	0.697674	0.4	0.595694	0.000000	0.0
4	0.454	1.0	1.0	0.302326	0.6	0.000000	0.000000	1.0
5	0.580	0.0	0.0	0.558140	0.5	0.792284	0.000000	1.0
6	0.568	0.0	1.0	0.441860	0.5	0.518950	0.333333	0.0
7	0.722	0.0	0.0	0.534884	0.3	0.655763	0.000000	1.0
8	0.504	0.5	1.0	0.395349	0.7	0.407628	0.333333	1.0
9	0.358	1.0	1.0	0.395349	0.5	0.000000	0.333333	1.0

In [144]:

```
# now we will handle our imbalanced target variable " Exited " using smotetomek

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

In [145]:

```
print(x.shape)
print(y.shape)
y.value_counts()
```

```
(9589, 10)
(9589,)
```

Out[145]:

```
0    7693
1    1896
Name: Exited, dtype: int64
```

In [146]:

```
from imblearn.combine import SMOTETomek
```

In [147]:

```
# Implementing Oversampling for Handling Imbalanced
smk = SMOTETomek(random_state=42)
x_res,y_res=smk.fit_resample(x,y)
```

In [148]:

```
print(x_res.shape)
print(y_res.shape)
```

```
(14938, 10)
(14938,)
```

In [149]:

```
y_res.value_counts()
```

Out[149]:

```
1    7469
0    7469
Name: Exited, dtype: int64
```

In [150]:

```
# applying random oversampling
from imblearn.over_sampling import RandomOverSampler
os =  RandomOverSampler(sampling_strategy=0.6)
X_train_res, y_train_res = os.fit_resample(x, y)
```

In [151]:

```
X_train_res.shape,y_train_res.shape
```

Out[151]:

```
((12308, 10), (12308,))
```

In [152]:

```
y_train_res.value_counts() # pca has to be done
```

Out[152]:

```
0    7693
1    4615
Name: Exited, dtype: int64
```

observations

1. randomoversampling looks well good beacause it is maintaining the data hygene.
2. smotetomek is exceptionally creating similar ratio data which is not true

we will proceed with random over sampling

In [153]:

```
X_train_res.head(5)
```

Out[153]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	0.686	0.0	1.0	0.511628	0.4	0.587859	0.000000	1.0
1	0.844	0.5	0.0	0.767442	0.9	0.647557	0.000000	0.0
2	0.634	0.0	1.0	0.348837	0.4	0.000000	0.333333	1.0
3	0.512	0.5	1.0	0.697674	0.4	0.595694	0.000000	0.0
4	0.454	1.0	1.0	0.302326	0.6	0.000000	0.000000	1.0

In [154]:

```
y_train_res.head(5)
```

Out[154]:

```
0    1
1    0
2    1
3    0
4    0
Name: Exited, dtype: int32
```

In [155]:

```
X_train_res.insert(10,'Exited',y_train_res)
```

In [156]:

```
X_train_res.head(5)
```

Out[156]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	0.686	0.0	1.0	0.511628	0.4	0.587859	0.000000	1.0
1	0.844	0.5	0.0	0.767442	0.9	0.647557	0.000000	0.0
2	0.634	0.0	1.0	0.348837	0.4	0.000000	0.333333	1.0
3	0.512	0.5	1.0	0.697674	0.4	0.595694	0.000000	0.0
4	0.454	1.0	1.0	0.302326	0.6	0.000000	0.000000	1.0

In [157]:

```
X_train_res.head(5)
```

Out[157]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	0.686	0.0	1.0	0.511628	0.4	0.587859	0.000000	1.0
1	0.844	0.5	0.0	0.767442	0.9	0.647557	0.000000	0.0
2	0.634	0.0	1.0	0.348837	0.4	0.000000	0.333333	1.0
3	0.512	0.5	1.0	0.697674	0.4	0.595694	0.000000	0.0
4	0.454	1.0	1.0	0.302326	0.6	0.000000	0.000000	1.0

In [158]:

```
data_final=X_train_res
```

In [159]:

```
data_final.sample()
```

Out[159]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCa
7510	0.684	1.0	1.0	0.651163	0.2	0.0	0.333333	

In [160]:

```
# Now we will prepare two different dataset for Model evaluation
dataone=data_final.drop(['CreditScore','NumOfProducts'],axis=1)
```

In [161]:

```
datatwo=data_final.drop(['CreditScore','Balance','NumOfProducts','Age'],axis=1)
```

In [162]:

```
print(dataone.shape)
print(datatwo.shape)
```

```
(12308, 9)
(12308, 7)
```

NOW WE WILL PROCEED WITH MODEL BUILDING PROCESS

1. DATA SPLITTING
2. CROSS VALIDATION
3. HYPER-PARAMETER TUNING
4. MODEL BUILDING
5. MODEL EVALUATION AND SELECTION

In [163]:

```
# first we will go with dataone dataset
X=dataone.drop('Exited',axis=1)
Y=dataone['Exited']
```

In [164]:

```
print(X.shape)
print(Y.shape)
```

```
(12308, 8)
(12308,)
```

In [165]:

```
# forst we will go with train test split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=69)
```

In [166]:

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(8615, 8)
(3693, 8)
(8615,)
(3693,)
```

In [167]:

```
# logistics regression
lg=LogisticRegression()
```

In [168]:

```
parameter={
    'penalty':['l1','l2','elasticnet'],
    'C':[1,2,3,4,5,6,10,20],
    'max_iter':[100,200,300,400,500]
}
```

In [169]:

```
aftercv=GridSearchCV(lg,param_grid=parameter,scoring='accuracy',cv=10)
```

In [170]:

```
aftercv.fit(x_train,y_train)
```

Out[170]:

```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [1, 2, 3, 4, 5, 6, 10, 20],
                         'max_iter': [100, 200, 300, 400, 500],
                         'penalty': ['l1', 'l2', 'elasticnet']},
             scoring='accuracy')
```

In [171]:

```
print(aftercv.best_params_)
print(aftercv.best_score_)
```

```
{'C': 1, 'max_iter': 100, 'penalty': 'l2'}
0.6275100986011518
```

In [172]:

```
y_pred=aftercv.predict(x_test)
```

In [173]:

```
from sklearn.metrics import accuracy_score,classification_report
```

In [174]:

```
score1=accuracy_score(y_pred,y_test)
score1
```

Out[174]:

```
0.6192797183861359
```

In [175]:

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

```
              precision    recall  f1-score   support

           0       1.00      0.62      0.76      3693
           1       0.00      0.00      0.00         0

    accuracy                           0.62      3693
   macro avg       0.50      0.31      0.38      3693
weighted avg       1.00      0.62      0.76      3693
```

NOW WE WILL WORK WITH SVC

In [176]:

```
sv=svm.SVC(kernel='poly',C=5,degree=2)
```

In [177]:

```
sv.fit(x_train,y_train)
```

Out[177]:

```
SVC(C=5, degree=2, kernel='poly')
```

In [178]:

```
y_pred=sv.predict(x_test)
```

In [179]:

```
sc=accuracy_score(y_pred,y_test)
```

In [180]:

```
sc
```

Out[180]:

```
0.6192797183861359
```

In [181]:

```
# decision tree

from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
```

In [182]:

```
parameterdt={
    'criterion':['gini','entropy'],
    'max_depth':[1,2,3,4,5,6,None]
}
```

In [183]:

```
aftercv_dt=GridSearchCV(dt,param_grid=parameterdt,cv=10,scoring='accuracy')
```

In [184]:

```
aftercv_dt.fit(x_train,y_train)
```

Out[184]:

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 2, 3, 4, 5, 6, None]},
             scoring='accuracy')
```

In [185]:

```
aftercv_dt.best_params_
```

Out[185]:

```
{'criterion': 'entropy', 'max_depth': None}
```

In [186]:

```
aftercv_dt.best_score_
```

Out[186]:

```
0.770637929780027
```

Applying Random Forest Algorithm

In [187]:

```
# so without using any hyper parameters score is increasing now on next we will apply hy
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
y_pred=rf.predict(x_test)
scorerf=accuracy_score(y_pred,y_test)
scorerf
```

Out[187]:

```
0.8773354995938262
```

In [188]:

```
rf_new=RandomForestClassifier()
```

In [189]:

```
parameter={
    'n_estimators':[10,40,60,100,120],
    'max_features':[0.2,0.4,0.6,1.0,1.2],
    'max_depth':[2,6,8,10,12,14],
    'max_samples':[0.5,0.75,1.0]
}
```

In [190]:

```
aftercv_rf=GridSearchCV(rf_new,param_grid=parameter,cv=5,scoring='accuracy',n_jobs=-1,ve
```

In [191]:

```
aftercv_rf.fit(x_train,y_train)
```

Out[191]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [2, 6, 8, 10, 12, 14],
                         'max_features': [0.2, 0.4, 0.6, 1.0, 1.2],
                         'max_samples': [0.5, 0.75, 1.0],
                         'n_estimators': [10, 40, 60, 100, 120]},
             scoring='accuracy')
```

In [192]:

```
print(aftercv_rf.best_params_)
```

```
{'max_depth': 14, 'max_features': 0.2, 'max_samples': 1.0, 'n_estimators':
120}
```

In [193]:

```
print(aftercv_rf.best_score_)
```

```
0.8160185722576901
```

In [194]:

```
y_pred=aftercv_rf.predict(x_test)
sco=accuracy_score(y_pred,y_test)
sco
```

Out[194]:

```
0.8323855943677227
```

In [195]:

```
# now well move with clustering,bagging and boosting techniques
# 1. We have checked with XGBOOST but because of long time we are deleting that for now
```

OBSERVATIONS:

- Till now we can understand that random forest performing very good with 88% of accuracy
- So, we will check now ada boost and clustering otherwise will go with random forest algorithm

In [ ]:

```

```