(1)Import Lib and dataset

In [264]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set()

import warnings
warnings.filterwarnings('ignore')


from sklearn.model_selection import cross_val_score
```

In [265]:

```
df=pd.read_csv(r"C:\Users\bhavi\Desktop\Data Science\dataset\Churn_Modelling - Larry
```

In [266]:

```
df.head()
```

Out[266]:

wNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Nu
1	15634602	Hargrave	619	France	Female	42	2.0	0.00	
2	15647311	Hill	608	Spain	Female	41	1.0	83807.86	
3	15619304	Onio	502	France	Female	42	8.0	159660.80	
4	15701354	Boni	699	France	Female	39	1.0	0.00	
5	15737888	Mitchell	850	NaN	Female	43	2.0	125510.82	

In [267]:

```
df.info()
```

```
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        9998 non-null   object 
 5   Gender           9999 non-null   object 
 6   Age              10000 non-null  int64  
 7   Tenure           9999 non-null   float64
 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(3), int64(8), object(3)
memory usage: 1.1+ MB
```

In [268]:

```
#To verify null values by another coding

df.isnull().sum()/len(df)*100
```

Out[268]:

```
RowNumber          0.00
CustomerId         0.00
Surname            0.00
CreditScore        0.00
Geography          0.02
Gender             0.01
Age                0.00
Tenure             0.01
Balance            0.00
NumOfProducts      0.00
HasCrCard          0.00
IsActiveMember     0.00
EstimatedSalary    0.00
Exited             0.00
dtype: float64
```

In [269]:

```
df.describe()
```

Out[269]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
count	10000.00000	1.000000e+04	10000.000000	10000.000000	9999.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012301	76485.889288
std	2886.89568	7.193619e+04	96.653299	10.487806	2.891889	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

(2) Cleansing the dataset (Handled Null values)

In [270]:

```
# check duplicate rows


def drop_dup(df):
    if df.duplicated().any() == True:
        df.drop_duplicates( inplace = True, Keep = "Last",reset_index = True)

        print("data after removig duplicate rows",df.duplicated().sum())
    else:
        return "No action required(No duplicate rows)"



drop_dup(df)
```

Out[270]:

```
'No action required(No duplicate rows)'
```

Data entry review : To see the data entry in all column

In [271]:

```
for i in df.columns:
    print(i)
    print()
    print(set(df[i].tolist()))
    print()
```

```
RowNumber

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 10
8, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 12
2, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 13
6, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 15
0, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 16
4, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 17
8, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 19
2, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 20
6, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 22
0, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 23
4, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 24
8, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 26
```

(3) Visulization the data through Summarytool, histogram

#Below through Summarytools, we can find unique values in each column.

In [272]:

```
from summarytools import dfSummary
dfSummary(df)
```

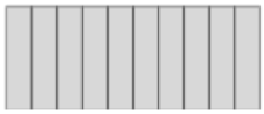
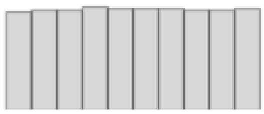
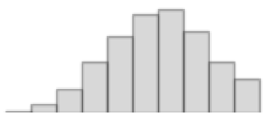
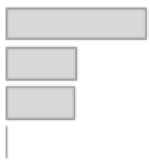
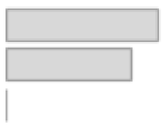
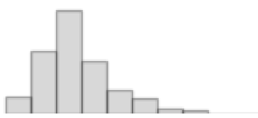
Out[272]:

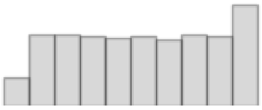
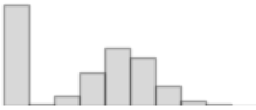
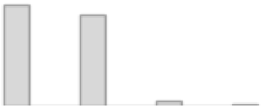
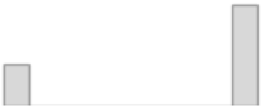
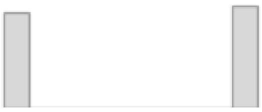
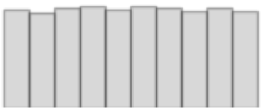
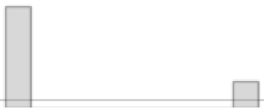
Data Frame Summary

df

Dimensions: 10,000 x 14

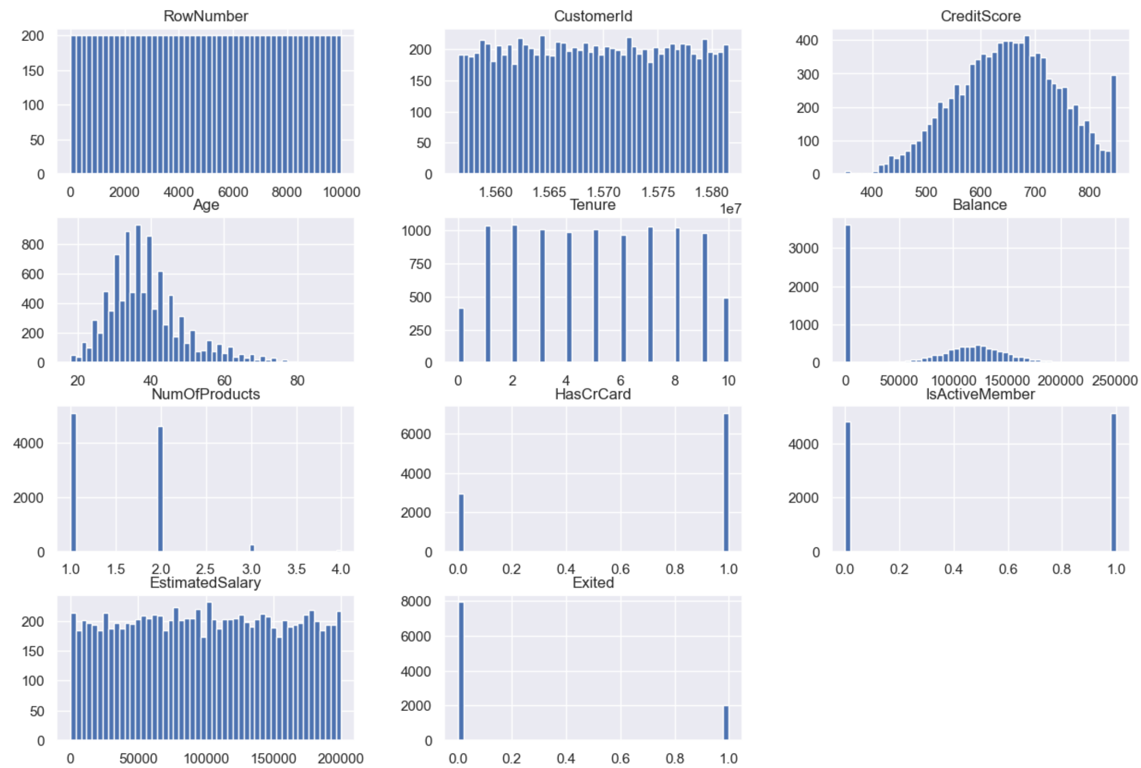
Duplicates: 0

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**RowNumber** [int64]	Mean (sd) : 5000.5 (2886.9) min < med < max: 1.0 < 5000.5 < 10000.0 IQR (CV) : 4999.5 (1.7)	10,000 distinct values		0 (0.0%)
2	**CustomerId** [int64]	Mean (sd) : 15690940.6 (71936.2) min < med < max: 15565701.0 < 15690738.0 < 15815690.0 IQR (CV) : 124705.5 (218.1)	10,000 distinct values		0 (0.0%)
3	**Surname** [object]	1. Smith 2. Scott 3. Martin 4. Walker 5. Brown 6. Yeh 7. Shih 8. Genovese 9. Maclean 10. Wright 11. other	32 (0.3%) 29 (0.3%) 29 (0.3%) 28 (0.3%) 26 (0.3%) 25 (0.2%) 25 (0.2%) 25 (0.2%) 24 (0.2%) 24 (0.2%) 9,733 (97.3%)		0 (0.0%)
4	**CreditScore** [int64]	Mean (sd) : 650.5 (96.7) min < med < max: 350.0 < 652.0 < 850.0 IQR (CV) : 134.0 (6.7)	460 distinct values		0 (0.0%)
5	**Geography** [object]	1. France 2. Germany 3. Spain 4. nan	5,014 (50.1%) 2,509 (25.1%) 2,475 (24.8%) 2 (0.0%)		2 (0.0%)
6	**Gender** [object]	1. Male 2. Female 3. nan	5,456 (54.6%) 4,543 (45.4%) 1 (0.0%)		1 (0.0%)
7	**Age** [int64]	Mean (sd) : 38.9 (10.5) min < med < max: 18.0 < 37.0 < 92.0 IQR (CV) : 12.0 (3.7)	70 distinct values		0 (0.0%)

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
8	**Tenure** [float64]	Mean (sd) : 5.0 (2.9) min < med < max: 0.0 < 5.0 < 10.0 IQR (CV) : 4.0 (1.7)	11 distinct values		1 (0.0%)
9	**Balance** [float64]	Mean (sd) : 76485.9 (62397.4) min < med < max: 0.0 < 97198.5 < 250898.1 IQR (CV) : 127644.2 (1.2)	6,382 distinct values		0 (0.0%)
10	**NumOfProducts** [int64]	Mean (sd) : 1.5 (0.6) min < med < max: 1.0 < 1.0 < 4.0 IQR (CV) : 1.0 (2.6)	4 distinct values		0 (0.0%)
11	**HasCrCard** [int64]	Mean (sd) : 0.7 (0.5) min < med < max: 0.0 < 1.0 < 1.0 IQR (CV) : 1.0 (1.5)	2 distinct values		0 (0.0%)
12	**IsActiveMember** [int64]	Mean (sd) : 0.5 (0.5) min < med < max: 0.0 < 1.0 < 1.0 IQR (CV) : 1.0 (1.0)	2 distinct values		0 (0.0%)
13	**EstimatedSalary** [float64]	Mean (sd) : 100090.2 (57510.5) min < med < max: 11.6 < 100193.9 < 199992.5 IQR (CV) : 98386.1 (1.7)	9,999 distinct values		0 (0.0%)
14	**Exited** [int64]	Mean (sd) : 0.2 (0.4) min < med < max: 0.0 < 0.0 < 1.0 IQR (CV) : 0.0 (0.5)	2 distinct values		0 (0.0%)

In [275]:

```
df.hist(bins=50,figsize=(15,10))
plt.show()
```



(4) Looking for corelations

In [274]:

```
corr_matrix=df.corr()
```

In [275]:

```
corr_matrix["Exited"].sort_values(ascending = False)
```

Out[275]:

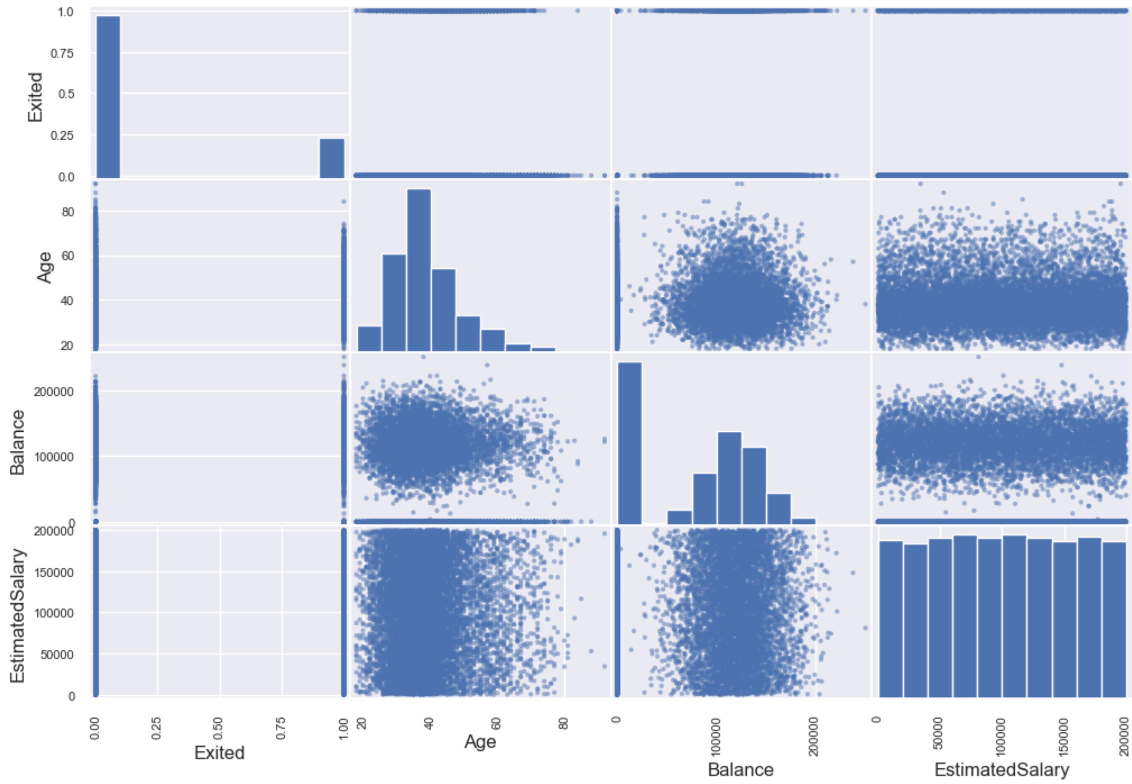
```
Exited             1.000000
Age                0.285323
Balance            0.118533
EstimatedSalary    0.012097
CustomerId        -0.006248
HasCrCard         -0.007138
Tenure            -0.013916
RowNumber         -0.016571
CreditScore       -0.027094
NumOfProducts     -0.047820
IsActiveMember    -0.156128
Name: Exited, dtype: float64
```

In [276]:

```python
from pandas.plotting import scatter_matrix
attributes = ["Exited","Age","Balance","EstimatedSalary"]
scatter_matrix(df[attributes],figsize = (12,8))
```

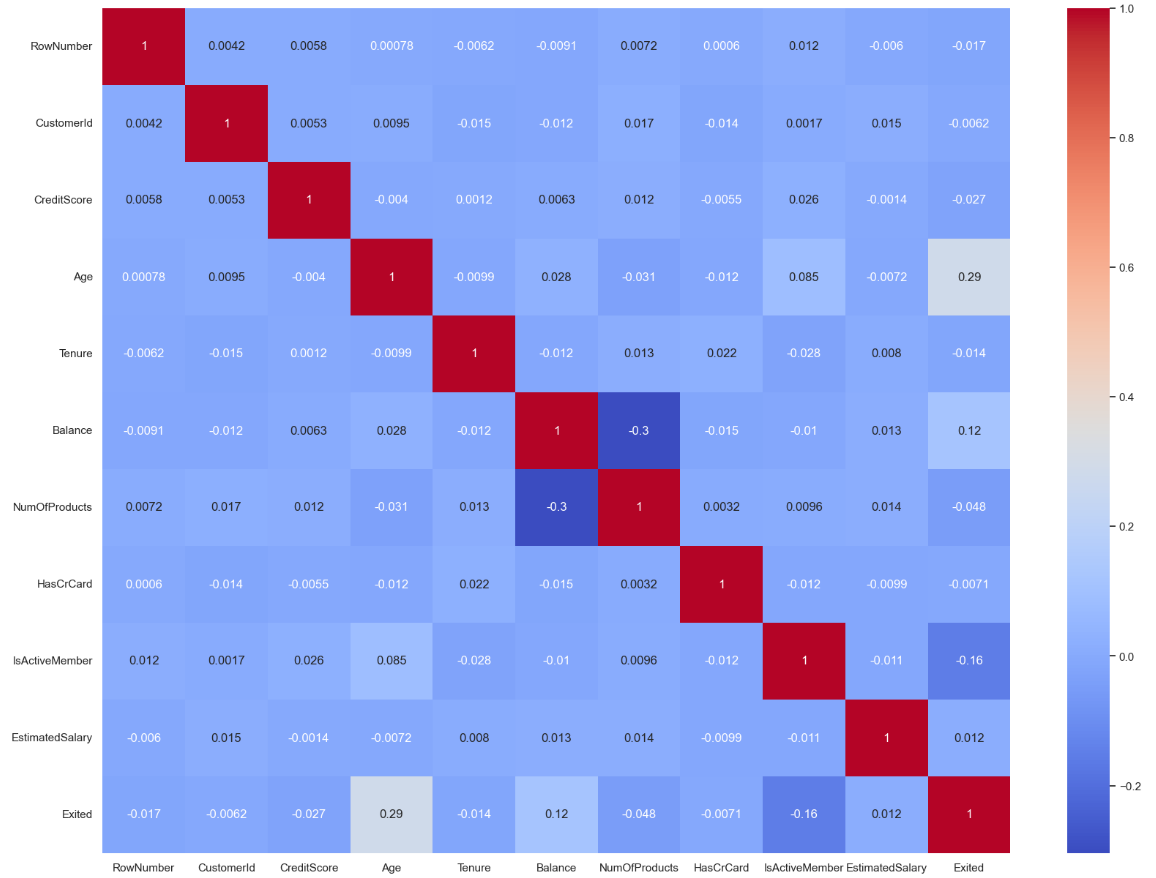
Out[276]:

```
array([[<Axes: xlabel='Exited', ylabel='Exited'>,
        <Axes: xlabel='Age', ylabel='Exited'>,
        <Axes: xlabel='Balance', ylabel='Exited'>,
        <Axes: xlabel='EstimatedSalary', ylabel='Exited'>],
       [<Axes: xlabel='Exited', ylabel='Age'>,
        <Axes: xlabel='Age', ylabel='Age'>,
        <Axes: xlabel='Balance', ylabel='Age'>,
        <Axes: xlabel='EstimatedSalary', ylabel='Age'>],
       [<Axes: xlabel='Exited', ylabel='Balance'>,
        <Axes: xlabel='Age', ylabel='Balance'>,
        <Axes: xlabel='Balance', ylabel='Balance'>,
        <Axes: xlabel='EstimatedSalary', ylabel='Balance'>],
       [<Axes: xlabel='Exited', ylabel='EstimatedSalary'>,
        <Axes: xlabel='Age', ylabel='EstimatedSalary'>,
        <Axes: xlabel='Balance', ylabel='EstimatedSalary'>,
        <Axes: xlabel='EstimatedSalary', ylabel='EstimatedSalary'>]],
      dtype=object)
```



In [277]:

```
#Heatmap to Find correlation
plt.figure(figsize=(20,15))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```



(5)Missing data

In [278]:

```
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        9998 non-null   object 
 5   Gender           9999 non-null   object 
 6   Age              10000 non-null  int64  
 7   Tenure           9999 non-null   float64
 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(3), int64(8), object(3)
```

In [279]:

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

In [280]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CreditScore      10000 non-null  int64  
 1   Geography        9998 non-null   object 
 2   Gender           9999 non-null   object 
 3   Age              10000 non-null  int64  
 4   Tenure           9999 non-null   float64
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64  
 7   HasCrCard        10000 non-null  int64  
 8   IsActiveMember   10000 non-null  int64  
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64  
dtypes: float64(3), int64(6), object(2)
memory usage: 859.5+ KB
```

In [281]:

```
df["Tenure"].median()
```

Out[281]:

```
5.0
```

In [282]:

```
df["Tenure"]= df["Tenure"].fillna(df["Tenure"].median())
```

In [283]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CreditScore      10000 non-null  int64  
 1   Geography        9998 non-null   object 
 2   Gender           9999 non-null   object 
 3   Age              10000 non-null  int64  
 4   Tenure           10000 non-null  float64
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64  
 7   HasCrCard        10000 non-null  int64  
 8   IsActiveMember   10000 non-null  int64  
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64  
dtypes: float64(3), int64(6), object(2)
memory usage: 859.5+ KB
```

```
Object column imputation is left only
```

In [284]:

```python
# Imputing null value
# pls treat numerical value first and then try below one - most_frequent
from sklearn.impute import SimpleImputer
imp_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
df_imputed = pd.DataFrame(imp_mode.fit_transform(df))
df_imputed.columns = df.columns
df_imputed
```

Out[284]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	619	France	Female	42	2.0	0.0	1	1
1	608	Spain	Female	41	1.0	83807.86	1	0
2	502	France	Female	42	8.0	159660.8	3	1
3	699	France	Female	39	1.0	0.0	2	0
4	850	France	Female	43	2.0	125510.82	1	1
...	...	...	...	...	...	...	...	...
9995	771	France	Male	39	5.0	0.0	2	1
9996	516	France	Male	35	10.0	57369.61	1	1
9997	709	France	Female	36	7.0	0.0	1	0
9998	772	Germany	Male	42	3.0	75075.31	2	1
9999	792	France	Female	28	4.0	130142.79	1	1

10000 rows × 11 columns

In [285]:

```python
df_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   CreditScore      10000 non-null  object
 1   Geography        10000 non-null  object
 2   Gender           10000 non-null  object
 3   Age              10000 non-null  object
 4   Tenure           10000 non-null  object
 5   Balance          10000 non-null  object
 6   NumOfProducts    10000 non-null  object
 7   HasCrCard        10000 non-null  object
 8   IsActiveMember   10000 non-null  object
 9   EstimatedSalary  10000 non-null  object
 10  Exited           10000 non-null  object
dtypes: object(11)
memory usage: 859.5+ KB
```

In [286]:

```
for i in df.select_dtypes(exclude=['object']).columns:
    df_imputed[i] = df_imputed[i].apply(lambda x :float(x))
```

In [287]:

```
df_new = df_imputed.copy()
```

In [288]:

```
df_new.head()
```

Out[288]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is
0	619.0	France	Female	42.0	2.0	0.00	1.0	1.0	
1	608.0	Spain	Female	41.0	1.0	83807.86	1.0	0.0	
2	502.0	France	Female	42.0	8.0	159660.80	3.0	1.0	
3	699.0	France	Female	39.0	1.0	0.00	2.0	0.0	
4	850.0	France	Female	43.0	2.0	125510.82	1.0	1.0	

In [289]:

```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CreditScore      10000 non-null  float64
 1   Geography        10000 non-null  object 
 2   Gender           10000 non-null  object 
 3   Age              10000 non-null  float64
 4   Tenure           10000 non-null  float64
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  float64
 7   HasCrCard        10000 non-null  float64
 8   IsActiveMember   10000 non-null  float64
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  float64
dtypes: float64(9), object(2)
memory usage: 859.5+ KB
```

(6) Encoding concept

Two column : for Gender, we will go with OHE and for Geography, we will go with dummy.

In [290]:

```
#Gender column : OHE
df_new['Gender'].value_counts()
df_new['Gender'] = df_new['Gender'].astype('category')
df_new['Gender'] = df_new['Gender'].cat.codes

#for Geography column : Dummy
df_new = pd.get_dummies(df_new, columns=['Geography'])
df_new = df_new.drop(['Geography_France'], axis=1)
```

In [291]:

```
df_new.head(10)
```

Out[291]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
0	619.0	0	42.0	2.0	0.00	1.0	1.0	1.
1	608.0	0	41.0	1.0	83807.86	1.0	0.0	1.
2	502.0	0	42.0	8.0	159660.80	3.0	1.0	0.
3	699.0	0	39.0	1.0	0.00	2.0	0.0	0.
4	850.0	0	43.0	2.0	125510.82	1.0	1.0	1.
5	645.0	1	44.0	8.0	113755.78	2.0	1.0	0.
6	822.0	1	50.0	7.0	0.00	2.0	1.0	1.
7	376.0	0	29.0	4.0	115046.74	4.0	1.0	0.
8	501.0	1	44.0	4.0	142051.07	2.0	0.0	1.
9	684.0	1	27.0	2.0	134603.88	1.0	1.0	1.

(7)Outlier handling

In [292]:

```
df_new_Ncat = df.select_dtypes(exclude='object')
df_new_Ncat.dtypes
```

Out[292]:

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

In [293]:

```
# outlier check on all  independent feature except dependent variable
outlier_list = list(df_new_Ncat.columns)

DepV_remove=['Exited']

for i in DepV_remove:
    outlier_list.remove(i)
    outlier_list
```

In [294]:

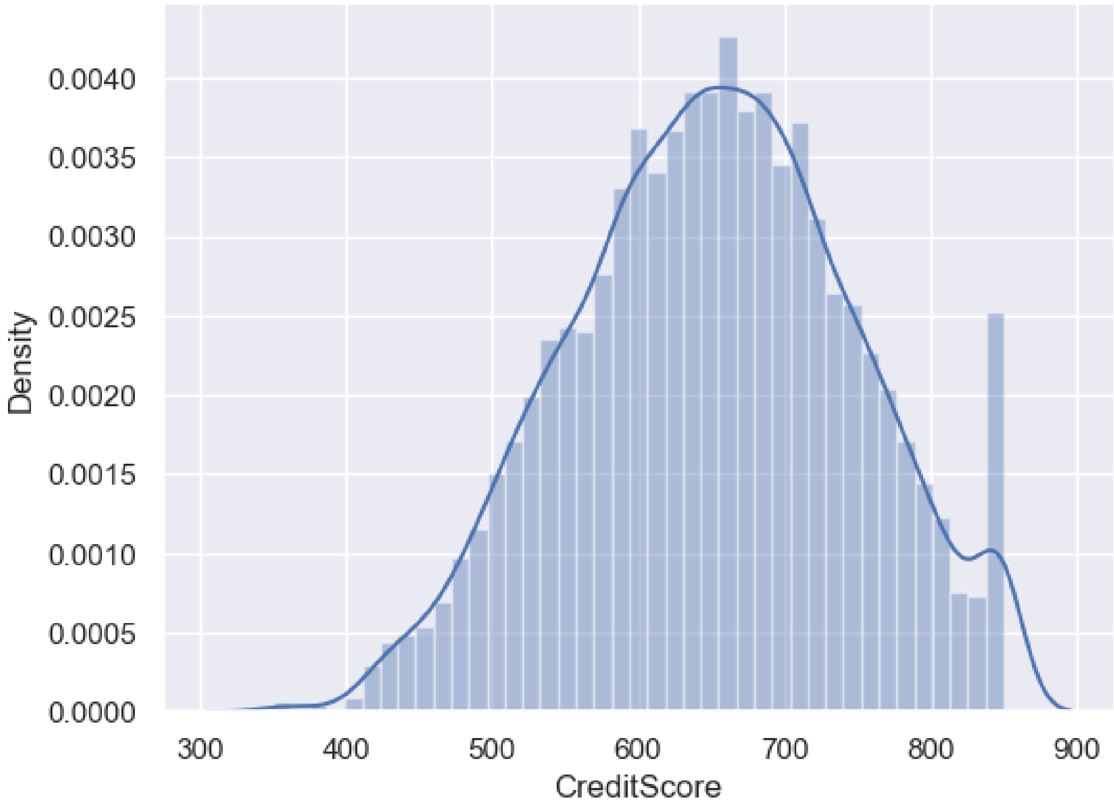
```
def boxplots(df,col):
    sns.boxplot(y = col, data = df_new_Ncat, palette ='Set2' )
    plt.show()
for col in outlier_list:
        boxplots(df,col)
```

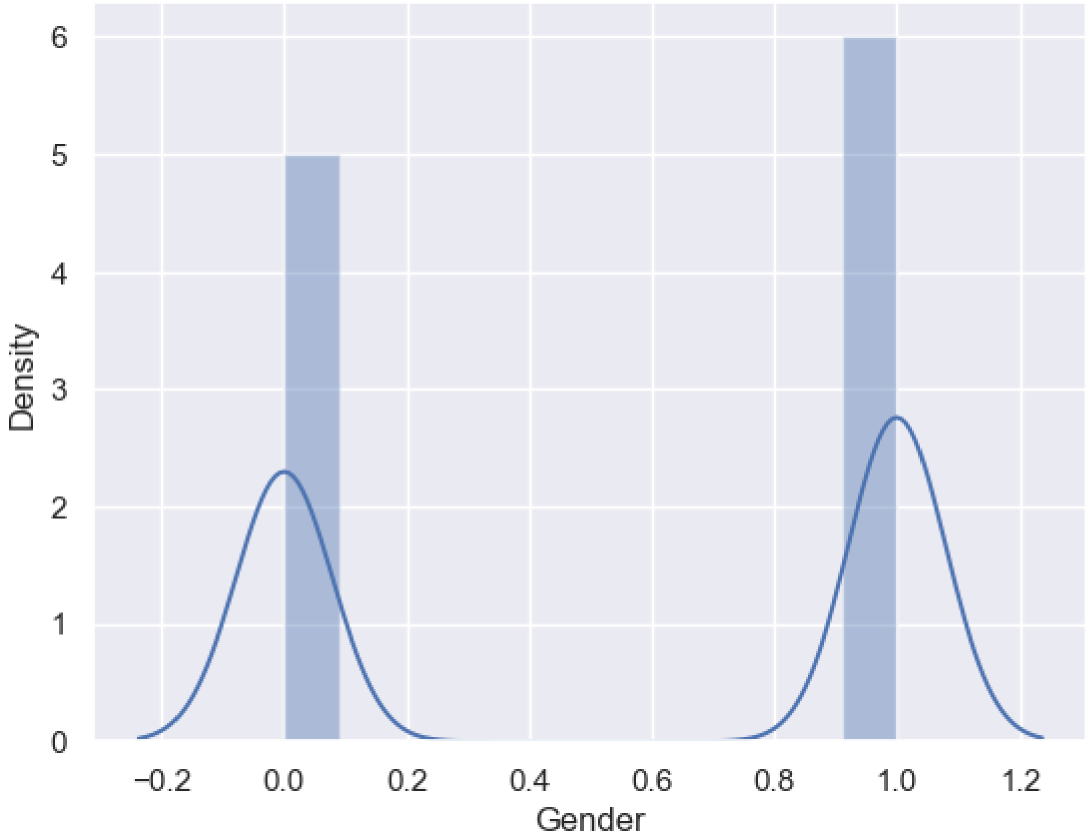
In [295]:

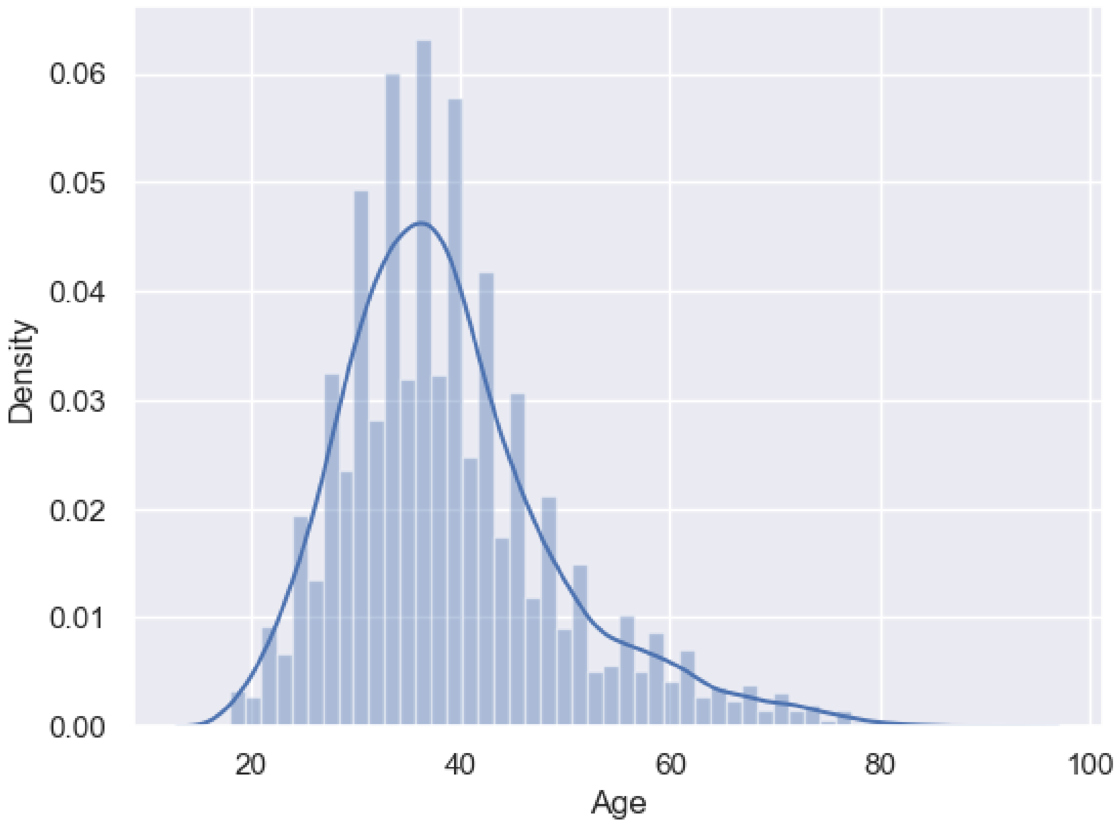
```
# Find the distribution of the dataset
```

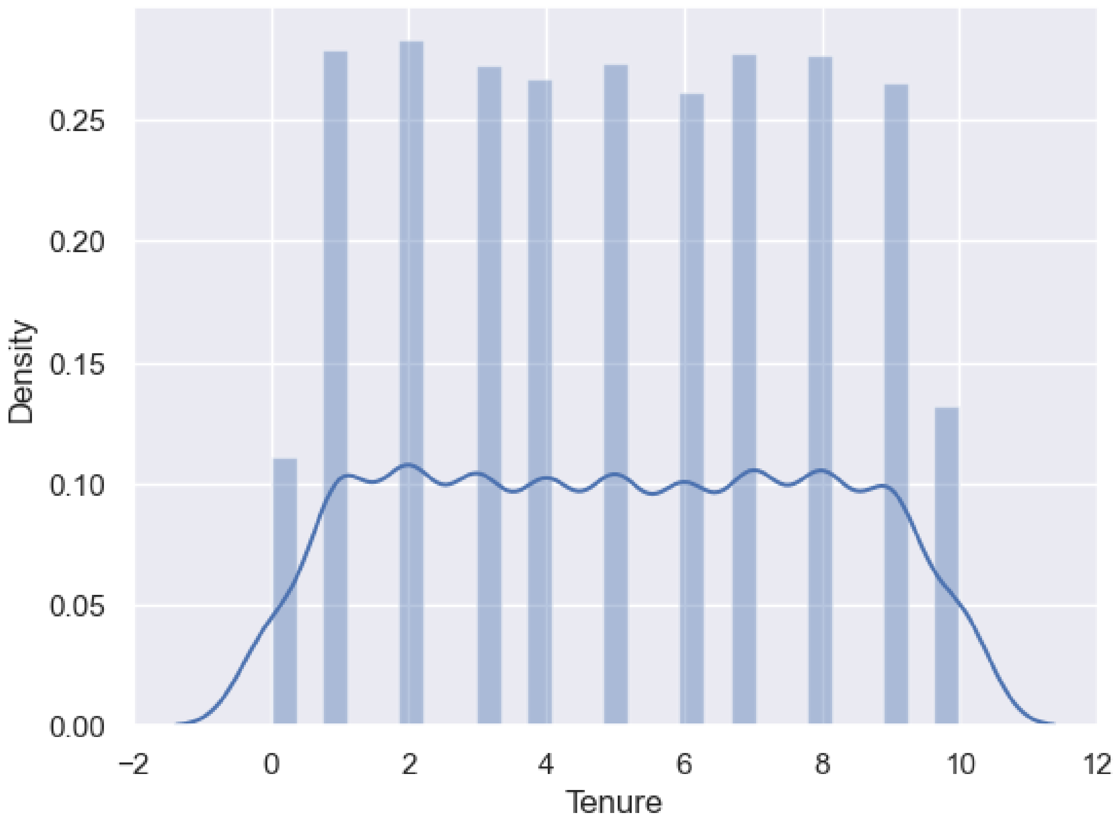
In [296]:

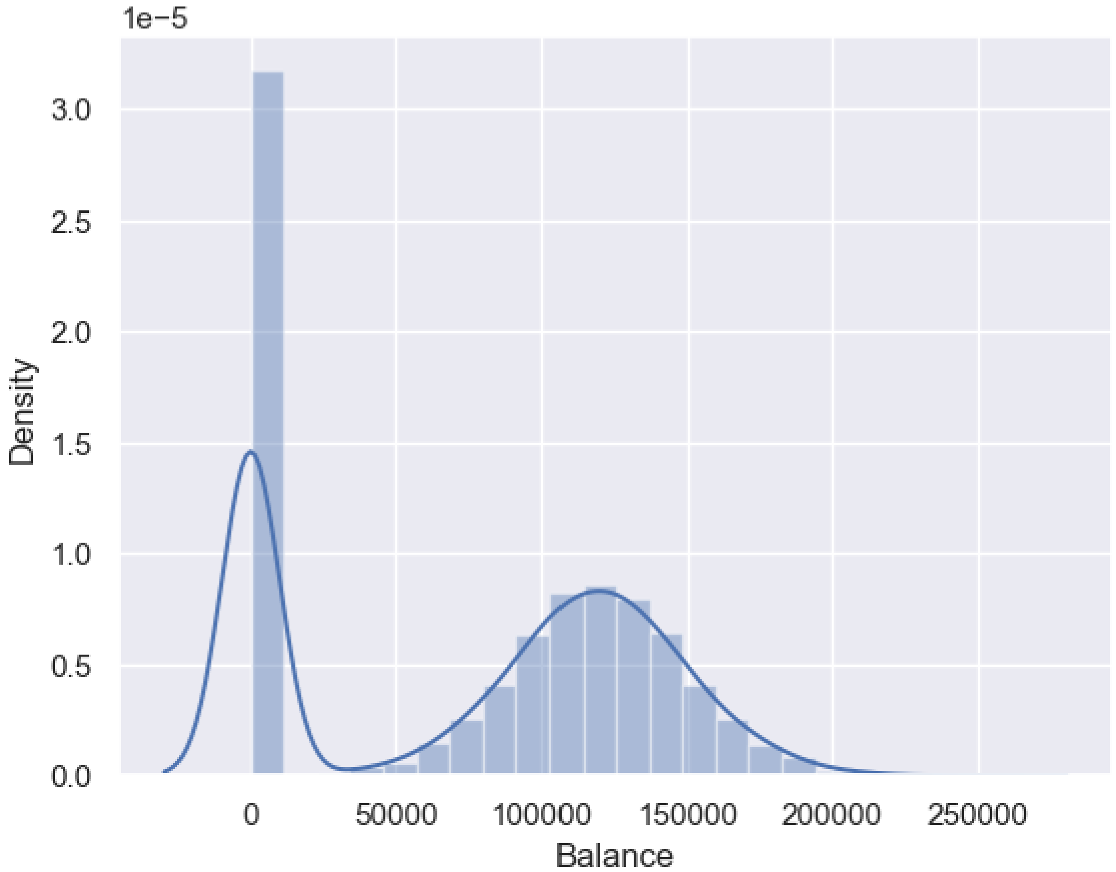
```python
# Find the distribution of the dataset
def distplots(col):
    sns.distplot(df_new[col])
    plt.show()

for i in list(df_new.select_dtypes(exclude=['object']).columns)[0:]:
    distplots(i)
```

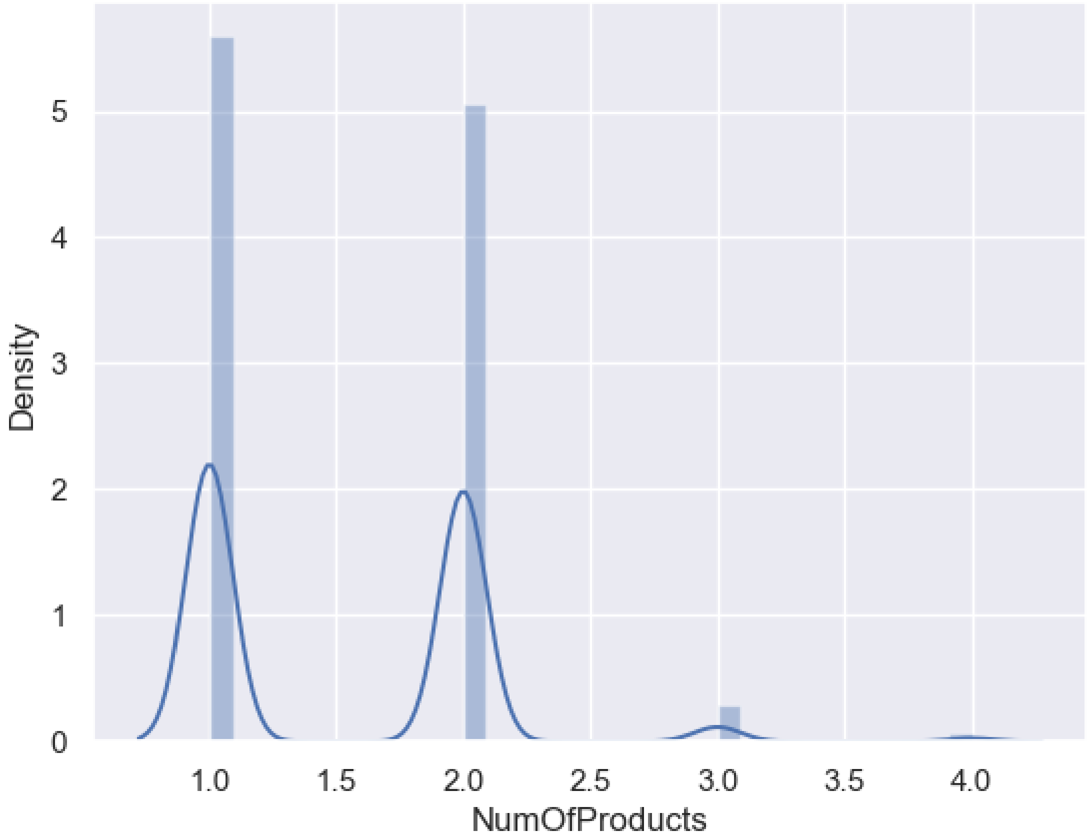



6
5
4
3
2
1
0
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Gender

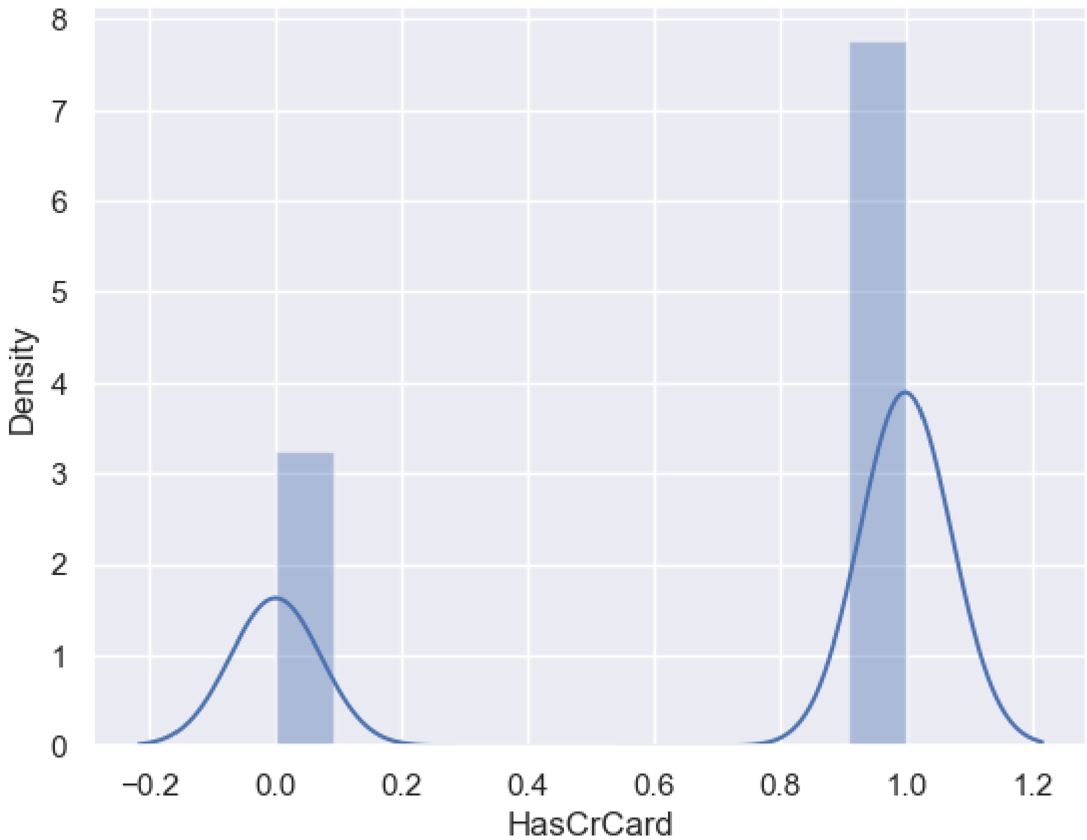

0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
20
40
60
80
100
Age


0.25
0.20
0.15
0.10
0.05
0.00
Density
−2
0
2
4
6
8
10
12
Tenure

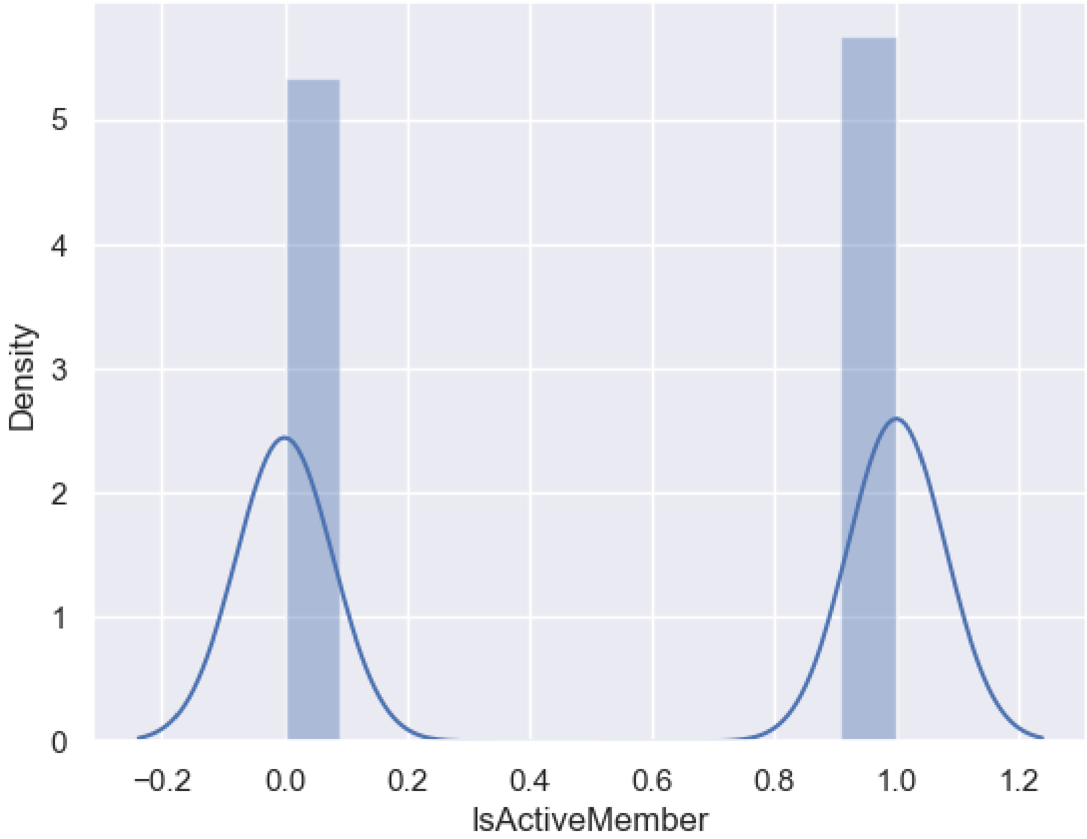

1e−5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Density
0
50000
100000
150000
200000
250000
Balance



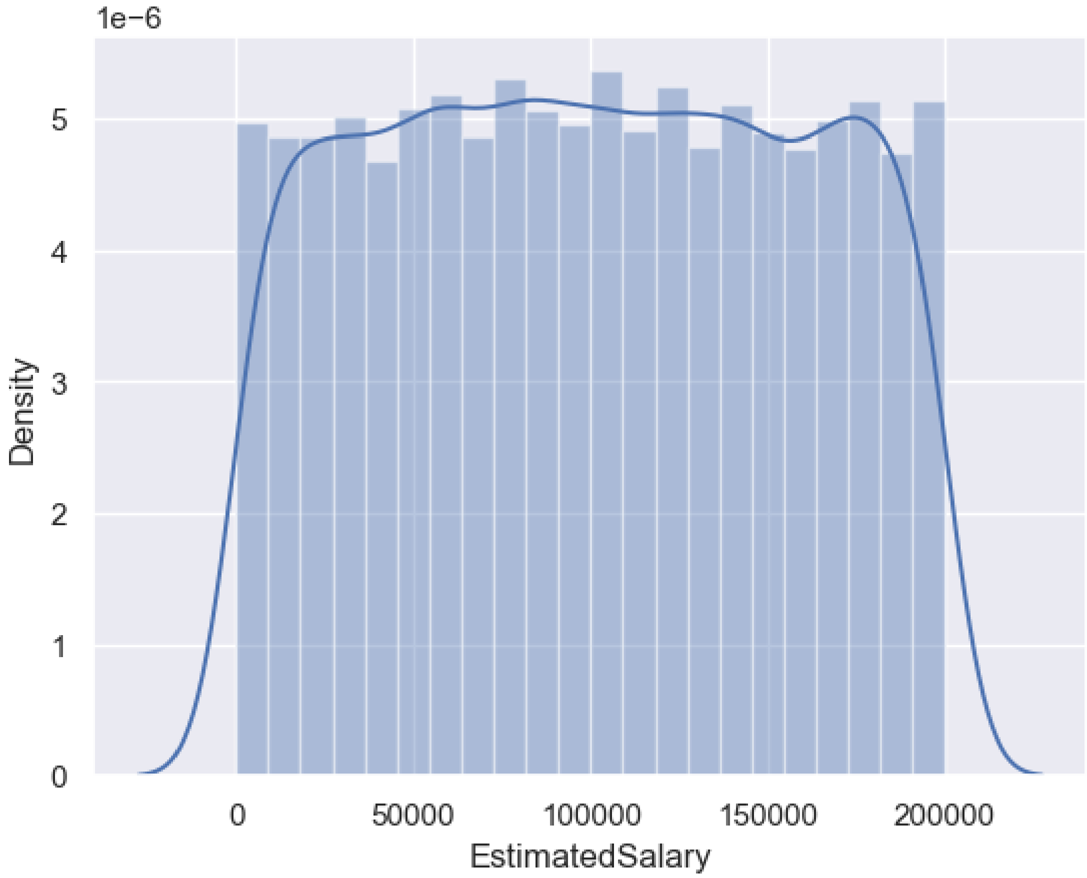
Density
5
4
3
2
1
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
NumOfProducts



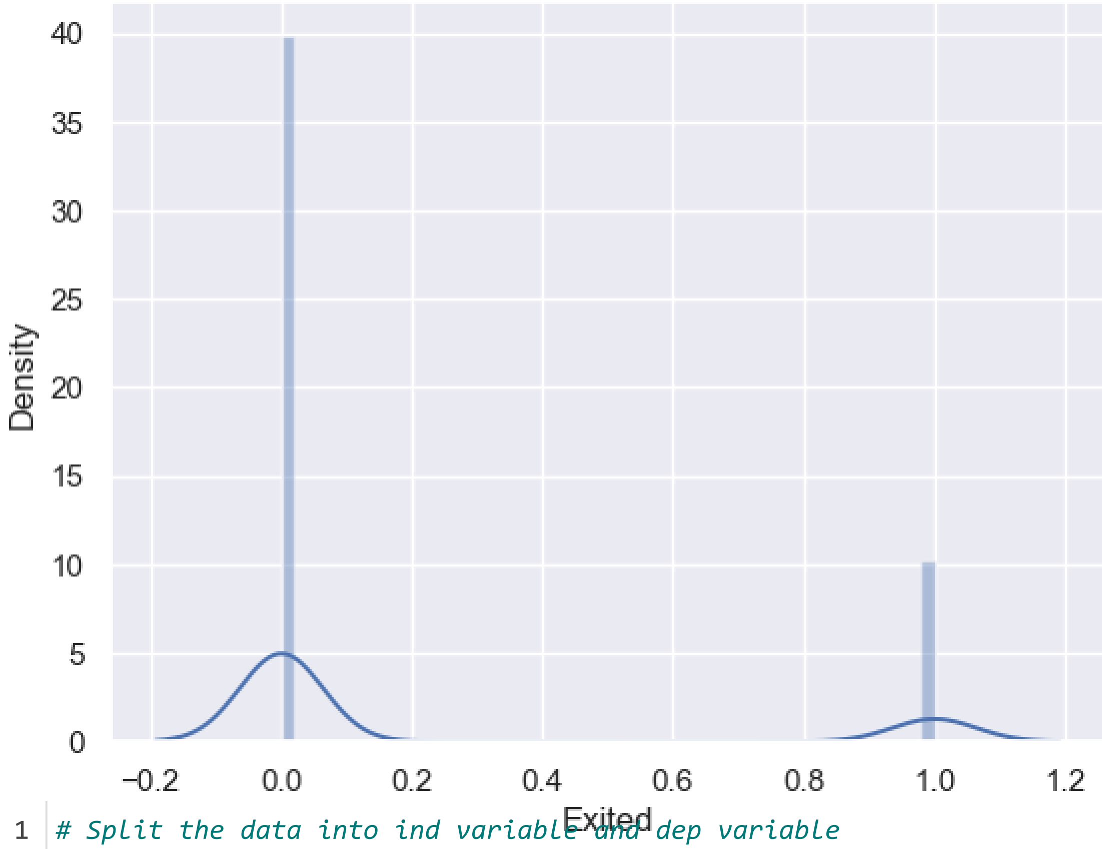
Density
8
7
6
5
4
3
2
1
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
HasCrCard



Density
0
1
2
3
4
5
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
IsActiveMember



1e−6
Density
0
1
2
3
4
5
0
50000
100000
150000
200000
EstimatedSalary

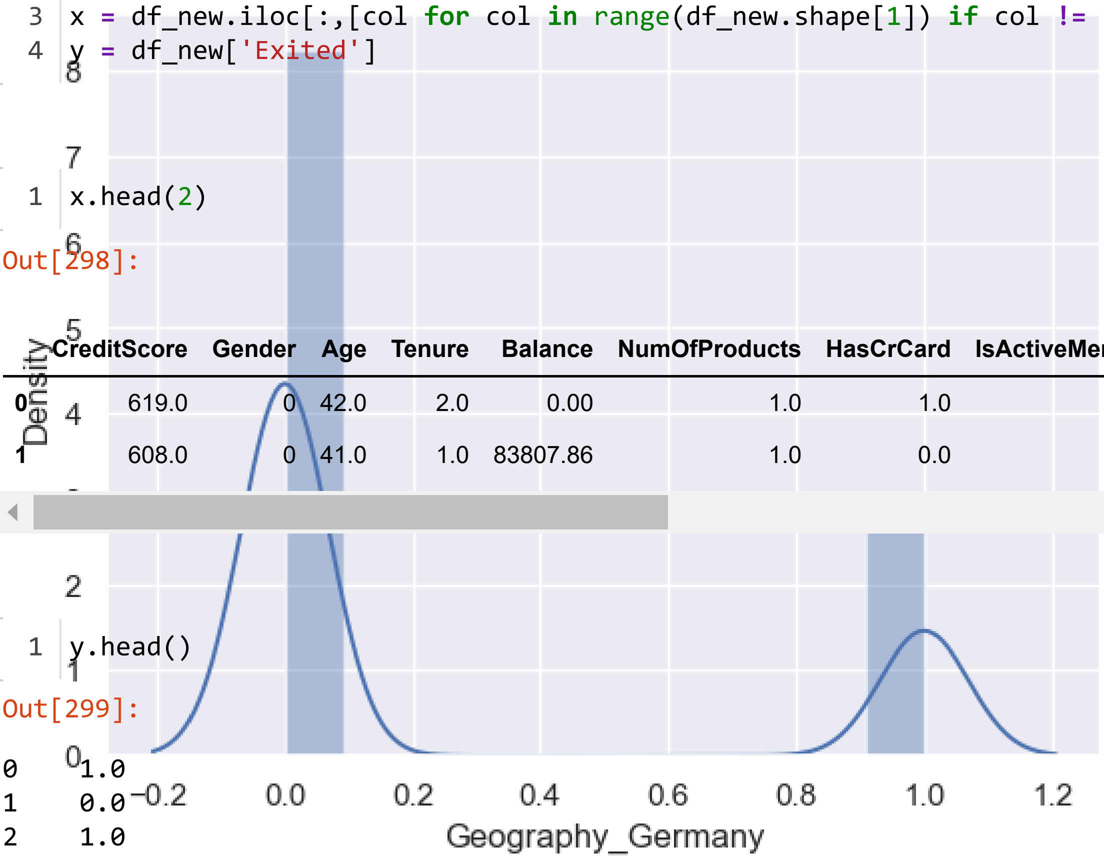



```
# Split the data into ind variable and dep variable
i=12
x = df_new.iloc[:,[col for col in range(df_new.shape[1]) if col != i]]
y = df_new['Exited']
```

```
x.head(2)
```

Out[298]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	619.0	0	42.0	2.0	0.00	1.0	1.0	1.0
1	608.0	0	41.0	1.0	83807.86	1.0	0.0	1.0

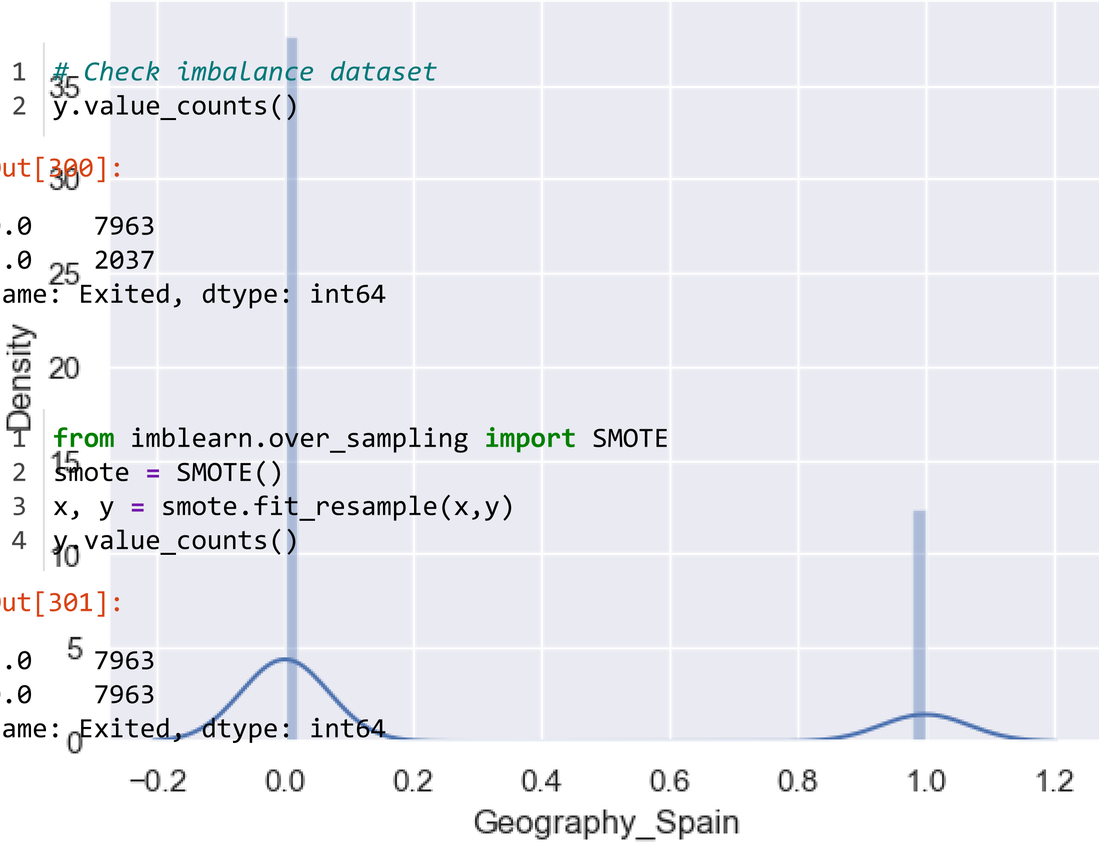



```
y.head()
```

Out[299]:

```
0    1.0
1    0.0
2    1.0
3    0.0
4    0.0
Name: Exited, dtype: float64
```

(11)Imbalance data




```
# Check imbalance dataset
y.value_counts()
```

Out[300]:

```
0.0    7963
1.0    2037
Name: Exited, dtype: int64
```

```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
x, y = smote.fit_resample(x,y)
y.value_counts()
```

Out[301]:

```
1.0    7963
0.0    7963
Name: Exited, dtype: int64
```

(10) Feature Scaling before spliting train test model(for independent variable)

In [302]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)
```

Out[302]:

	0	1	2	3	4	5	6	7	8	
0	-0.314408	-0.841684	0.080526	-1.098881	-1.334220	-0.834590	0.696388	1.158537	0.013958	1
1	-0.433927	-0.841684	-0.018536	-1.466585	0.033102	-0.834590	-1.641097	1.158537	0.208235	-1
2	-1.585649	-0.841684	0.080526	1.107344	1.270640	2.447289	0.696388	-0.975611	0.232342	1
3	0.554816	-0.841684	-0.216659	-1.466585	-1.334220	0.806349	-1.641097	-0.975611	-0.116598	-1
4	2.195476	-0.841684	0.179588	-1.098881	0.713484	-0.834590	0.696388	1.158537	-0.372469	-1
...	...	...	...	...	...	...	...	...	...	
15921	1.067609	1.188095	-1.124944	1.031784	-1.334220	-0.834590	0.696388	-0.118045	-1.397701	1
15922	0.852784	-0.841684	0.204974	-1.051766	-1.334220	-0.774516	0.696388	1.158537	1.205969	1
15923	-0.640367	-0.841684	0.524926	-1.427554	-1.334220	-0.660406	-1.392976	-0.749073	0.441154	1

In [303]:

```
x.head()
```

Out[303]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
0	619.0	0	42.0	2.0	0.00	1.0	1.0	1.
1	608.0	0	41.0	1.0	83807.86	1.0	0.0	1.
2	502.0	0	42.0	8.0	159660.80	3.0	1.0	0.
3	699.0	0	39.0	1.0	0.00	2.0	0.0	0.
4	850.0	0	43.0	2.0	125510.82	1.0	1.0	1.

In [304]:

```
x.shape
```

Out[304]:

```
(15926, 12)
```

In [305]:

```
# (9)Used PCA method to reduce variables( Unnecesary Variables)
```

(12) spliting in train and test

In [306]:

```
# split the training data into train and test
from sklearn.model_selection import train_test_split
x_train,x_test,y_train, y_test = train_test_split(x, y ,test_size=0.2, random_state=
```

(13) Machine Learning Model

In [307]:

```
#Machine Learning classification models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB

#voting method
from sklearn.ensemble import VotingClassifier

#to see the result
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score



```

(i) Accuracy check in Test data

In [308]:

```
# Applying all the model together

# Applying all the model together

# LogisticRegression
logistic = LogisticRegression()
logistic.fit(x_train, y_train)
y_train_pred_logistic = logistic.predict(x_train)
accuracy_logistic = accuracy_score(y_train, y_train_pred_logistic)

# DecisionTree
dtree = DecisionTreeClassifier()
dtree.fit(x_train, y_train)
y_train_pred_dtree = dtree.predict(x_train)
accuracy_dtree = accuracy_score(y_train, y_train_pred_dtree)

# RandomForest
rfmodel = RandomForestClassifier()
rfmodel.fit(x_train, y_train)
y_train_pred_rfmodel = rfmodel.predict(x_train)
accuracy_rfmodel = accuracy_score(y_train, y_train_pred_rfmodel)

# BaggingClassifier
bagg = BaggingClassifier()
bagg.fit(x_train, y_train)
y_train_pred_bagg = bagg.predict(x_train)
accuracy_bagg = accuracy_score(y_train, y_train_pred_bagg)

# AdaBoostClassifier
ada = AdaBoostClassifier()
ada.fit(x_train, y_train)
y_train_pred_ada = ada.predict(x_train)
accuracy_ada = accuracy_score(y_train, y_train_pred_ada)

# GradientBoostingClassifier
gdb = GradientBoostingClassifier()
gdb.fit(x_train, y_train)
y_train_pred_gdb = gdb.predict(x_train)
accuracy_gdb = accuracy_score(y_train, y_train_pred_gdb)

# XGBClassifier = RF + GDBoosting - lambda - regularisation, gamma - autoprunning, e
xgb = XGBClassifier()
xgb.fit(x_train, y_train)
y_train_pred_xgb = xgb.predict(x_train)
accuracy_xgb = accuracy_score(y_train, y_train_pred_xgb)


# SVM
svc = SVC()
svc.fit(x_train, y_train)
y_train_pred_svc = svc.predict(x_train)
accuracy_svc = accuracy_score(y_train, y_train_pred_svc)

# KNN
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
y_train_pred_knn = knn.predict(x_train)
accuracy_knn = accuracy_score(y_train, y_train_pred_knn)

```

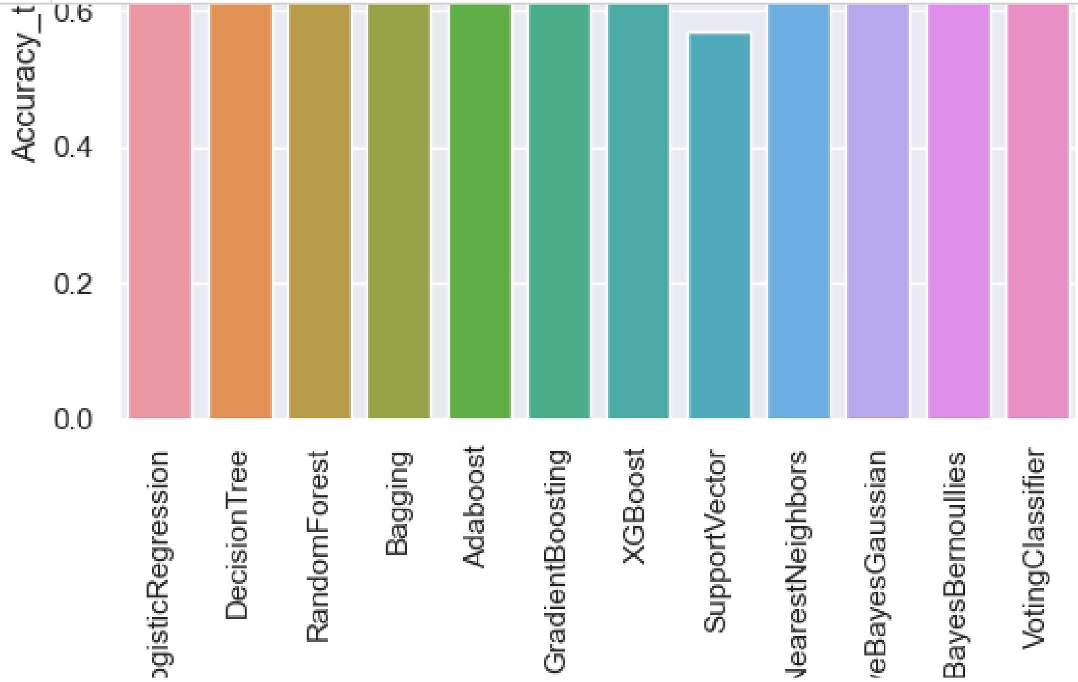
```
# GaussianNB
Gnb = GaussianNB()
Gnb.fit(x_train, y_train)
y_train_pred_Gnb = naive_gb.predict(x_train)
accuracy_Gnb = accuracy_score(y_train, y_train_pred_Gnb)

# BernoulliNB
Bnb = BernoulliNB()
Bnb.fit(x_train, y_train)
y_train_pred_Bnb = naive_bn.predict(x_train)
accuracy_Bnb = accuracy_score(y_train, y_train_pred_Bnb)
```

In [309]:

```
evc = VotingClassifier(estimators=[('logistic',logistic),('dtree',dtree),('rfmodel',
                                   ('gdb', gdb),('xgb', xgb),('svc', svc),('knn', knr
                                   ('Gnb', Gnb),('Bnb', Bnb)], voting='hard')

model_evc = evc.fit(x_train, y_train)
y_train_pred_evc = evc.predict(x_train)
accuracy_evc = accuracy_score(y_train, y_train_pred_evc)
```

In [331]:

```
Method_Used = ['LogisticRegression','DecisionTree','RandomForest','Bagging','Adaboos
         'GradientBoosting', 'XGBoost','SupportVector','KNearestNeighbors',
         'NaiveBayesGaussian','NaiveBayesBernoullies','VotingClassifier']

Accuracy_test = [accuracy_logistic, accuracy_dtree, accuracy_rfmodel, accuracy_bagg,
         accuracy_xgb, accuracy_svc, accuracy_knn, accuracy_Gnb, accuracy_Bnb, accur



final_accuracy_test = pd.DataFrame({'Method Used': Method_Used, "Accuracy_test": lis
print(final_accuracy_test)

charts = sns.barplot(x="Method Used", y = 'Accuracy_test', data=final_accuracy_test)
charts.set_xticklabels(charts.get_xticklabels(), rotation=90)
print(charts)
```



(ii)Accuracy check in Train data

In [332]:

```
# Applying all the model together

# LogisticRegression
logistic = LogisticRegression()
logistic.fit(x_train, y_train)
y_train_pred_logistic = logistic.predict(x_train)
accuracy_logistic = accuracy_score(y_train, y_train_pred_logistic)

# DecisionTree
dtree = DecisionTreeClassifier()
dtree.fit(x_train, y_train)
y_train_pred_dtree = dtree.predict(x_train)
accuracy_dtree = accuracy_score(y_train, y_train_pred_dtree)

# RandomForest
rfmodel = RandomForestClassifier()
rfmodel.fit(x_train, y_train)
y_train_pred_rfmodel = rfmodel.predict(x_train)
accuracy_rfmodel = accuracy_score(y_train, y_train_pred_rfmodel)

# BaggingClassifier
bagg = BaggingClassifier()
bagg.fit(x_train, y_train)
y_train_pred_bagg = bagg.predict(x_train)
accuracy_bagg = accuracy_score(y_train, y_train_pred_bagg)

# AdaBoostClassifier
ada = AdaBoostClassifier()
ada.fit(x_train, y_train)
y_train_pred_ada = ada.predict(x_train)
accuracy_ada = accuracy_score(y_train, y_train_pred_ada)

# GradientBoostingClassifier
gdb = GradientBoostingClassifier()
gdb.fit(x_train, y_train)
y_train_pred_gdb = gdb.predict(x_train)
accuracy_gdb = accuracy_score(y_train, y_train_pred_gdb)

# XGBClassifier = RF + GDBoosting - lambda - regularisation, gamma - autoprunning, e
xgb = XGBClassifier()
xgb.fit(x_train, y_train)
y_train_pred_xgb = xgb.predict(x_train)
accuracy_xgb = accuracy_score(y_train, y_train_pred_xgb)


# SVM
svc = SVC()
svc.fit(x_train, y_train)
y_train_pred_svc = svc.predict(x_train)
accuracy_svc = accuracy_score(y_train, y_train_pred_svc)

# KNN
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
y_train_pred_knn = knn.predict(x_train)
accuracy_knn = accuracy_score(y_train, y_train_pred_knn)

# GaussianNB
Gnb = GaussianNB()
```

```
Gnb.fit(x_train, y_train)
y_train_pred_Gnb = naive_gb.predict(x_train)
accuracy_Gnb = accuracy_score(y_train, y_train_pred_Gnb)

# BernoulliNB
Bnb = BernoulliNB()
Bnb.fit(x_train, y_train)
y_train_pred_Bnb = naive_bn.predict(x_train)
accuracy_Bnb = accuracy_score(y_train, y_train_pred_Bnb)
```

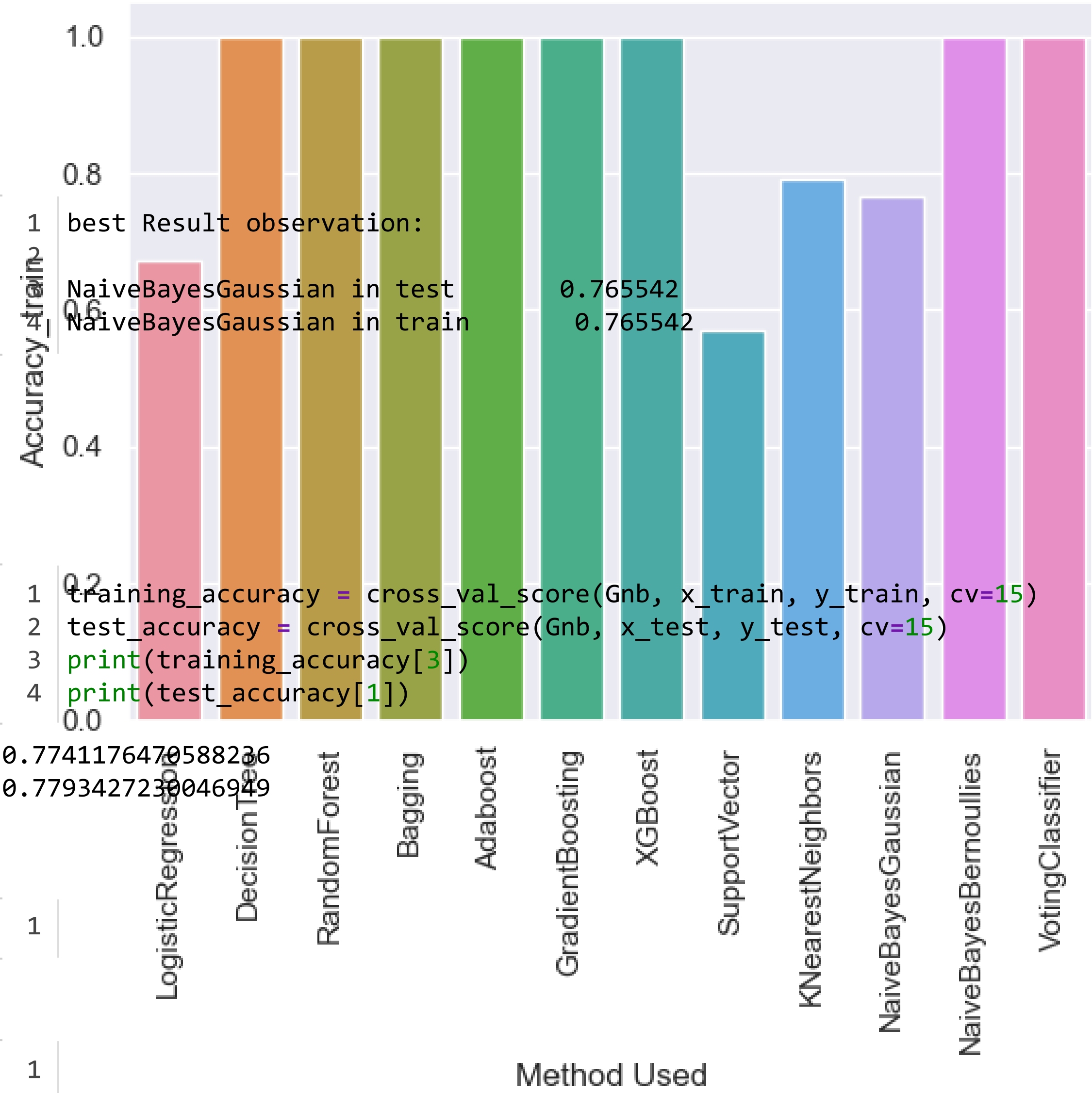
In [333]:

```
evc = VotingClassifier(estimators=[('logistic',logistic),('dtree',dtree),('rfmodel',
                                   ('gdb', gdb),('xgb', xgb),('svc', svc),('knn', knr
                                   ('Gnb', Gnb),('Bnb', Bnb)], voting='hard')

model_evc = evc.fit(x_train, y_train)
y_train_pred_evc = evc.predict(x_train)
accuracy_evc = accuracy_score(y_train, y_train_pred_evc)
```

In [334]:

```
Method_Used = ['LogisticRegression','DecisionTree','RandomForest','Bagging','Adaboos
         'GradientBoosting', 'XGBoost','SupportVector','KNearestNeighbors',
         'NaiveBayesGaussian','NaiveBayesBernoullies','VotingClassifier']

Accuracy_train = [accuracy_logistic, accuracy_dtree, accuracy_rfmodel, accuracy_bagg
         accuracy_xgb, accuracy_svc, accuracy_knn, accuracy_Gnb, accuracy_Bnb, accur



final_accuracy_train = pd.DataFrame({'Method Used': Method_Used, "Accuracy_train": l
print(final_accuracy_train)

charts = sns.barplot(x="Method Used", y = 'Accuracy_train', data=final_accuracy_trai
charts.set_xticklabels(charts.get_xticklabels(), rotation=90)
print(charts)
```

```
              Method Used  Accuracy_train
0      LogisticRegression        0.671193
1            DecisionTree        1.000000
2            RandomForest        1.000000
3                 Bagging        1.000000
4                Adaboost        1.000000
5        GradientBoosting        1.000000
6                 XGBoost        1.000000
7           SupportVector        0.568995
8       KNearestNeighbors        0.790188
9      NaiveBayesGaussian        0.765542
10  NaiveBayesBernoullies        1.000000
11       VotingClassifier        1.000000
Axes(0.125,0.11;0.775x0.77)
```




```
best Result observation:

NaiveBayesGaussian in test       0.765542
NaiveBayesGaussian in train       0.765542
```

```
training_accuracy = cross_val_score(Gnb, x_train, y_train, cv=15)
test_accuracy = cross_val_score(Gnb, x_test, y_test, cv=15)
print(training_accuracy[3])
print(test_accuracy[1])
```

```
0.7741176470588236
0.7793427230046949
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: