

# Predicting Churn Rate

## (1) Import Lib and dataset

In [264]:

```
1 import numpy as np
2 import pandas as pd
3
4 import matplotlib.pyplot as plt
5 %matplotlib inline
6
7 import seaborn as sns
8 sns.set()
9
10 import warnings
11 warnings.filterwarnings('ignore')
12
13
14 from sklearn.model_selection import cross_val_score
```

In [265]:

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

In [266]:

1 df.head()

Out[266]:

RowIndex	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts
1	15634602	Hargrave	619	France	Female	42	2.0	0.00	0
2	15647311	Hill	608	Spain	Female	41	1.0	83807.86	0
3	15619304	Onio	502	France	Female	42	8.0	159660.80	0
4	15701354	Boni	699	France	Female	39	1.0	0.00	0
5	15737888	Mitchell	850	NaN	Female	43	2.0	125510.82	0

In [267]:

1 df.info()

RangeIndex: 10000 entries, 0 to 9999

Data columns (total 14 columns):

#	Column	Non-Null Count	Dtype
0	RowIndex	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]:

```

1 #To verify null values by another coding
2
3 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]:

```
1 df.describe()
```

Out[269]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
<b>count</b>	10000.00000	1.000000e+04	10000.000000	10000.000000	9999.000000	10000.000000
<b>mean</b>	5000.50000	1.569094e+07	650.528800	38.921800	5.012301	76485.889288
<b>std</b>	2886.89568	7.193619e+04	96.653299	10.487806	2.891889	62397.405202
<b>min</b>	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000
<b>25%</b>	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000
<b>50%</b>	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000
<b>75%</b>	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000
<b>max</b>	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000

## (2) Cleansing the dataset (Handled Null values)

In [270]:

```

1 # check duplicate rows
2
3
4 def drop_dup(df):
5     if df.duplicated().any() == True:
6         df.drop_duplicates( inplace = True, Keep = "Last",reset_index = True)
7
8         print("data after removig duplicate rows",df.duplicated().sum())
9     else:
10         return "No action required(No duplicate rows)"
11
12
13
14 drop_dup(df)

```

Out[270]:

'No action required(No duplicate rows)'

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

In [271]:

```

1 for i in df.columns:
2     print(i)
3     print()
4     print(set(df[i].tolist()))
5     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, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000}

## (3) Visulization the data through Summarytool, histogram

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

In [272]:

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

Out[272]:

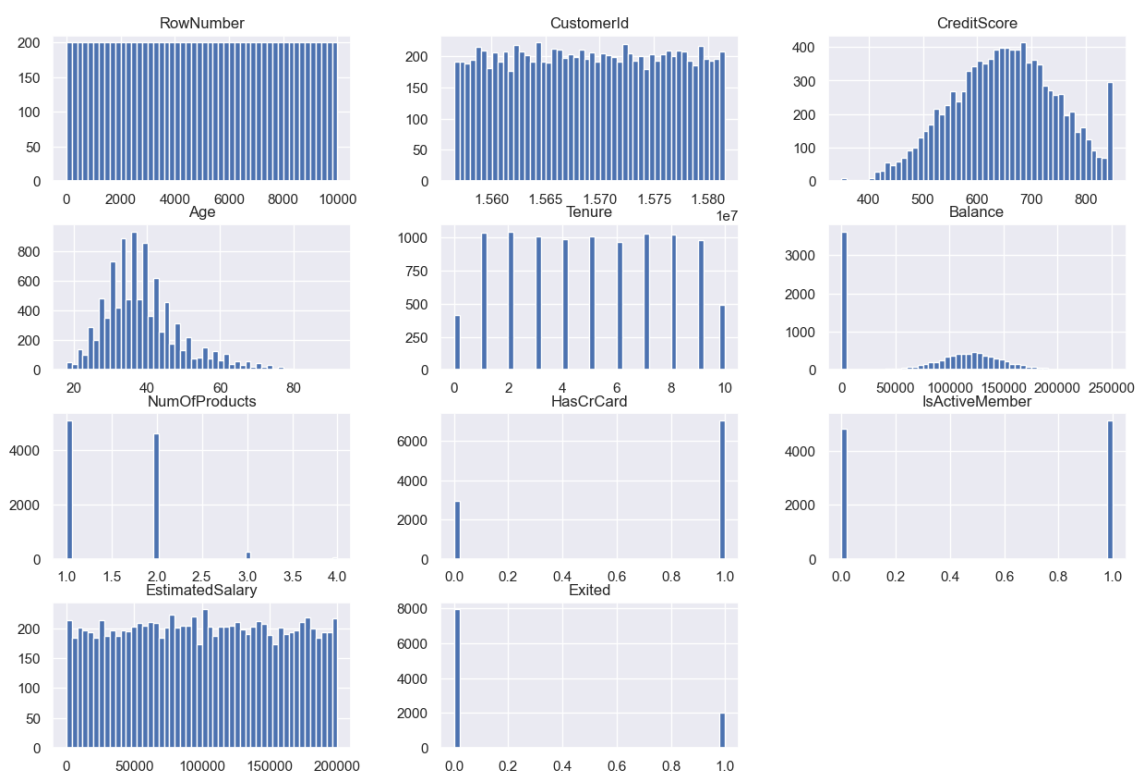
Data Frame Summary

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	<b>Tenure</b> [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	<b>Balance</b> [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	<b>NumOfProducts</b> [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	<b>HasCrCard</b> [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	<b>IsActiveMember</b> [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	<b>EstimatedSalary</b> [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	<b>Exited</b> [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%)

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





## (4) Looking for corelations

In [274]:

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

In [275]:

```
1 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]:

```

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

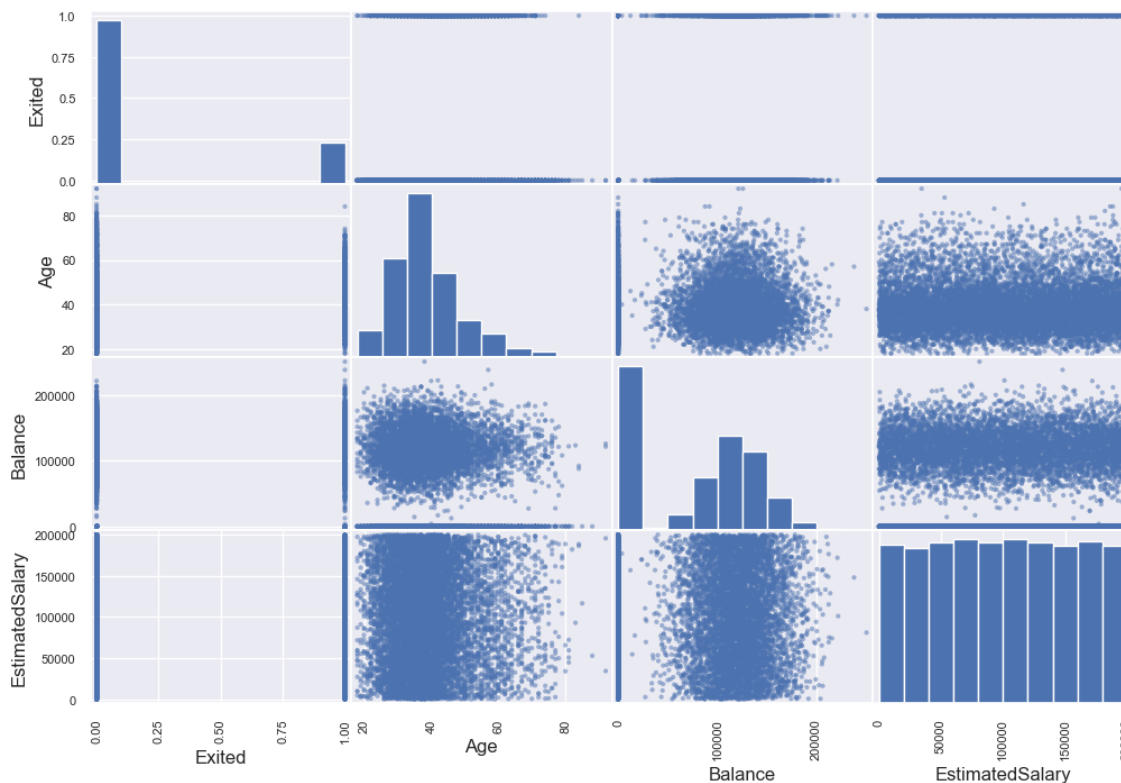
```

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)

```

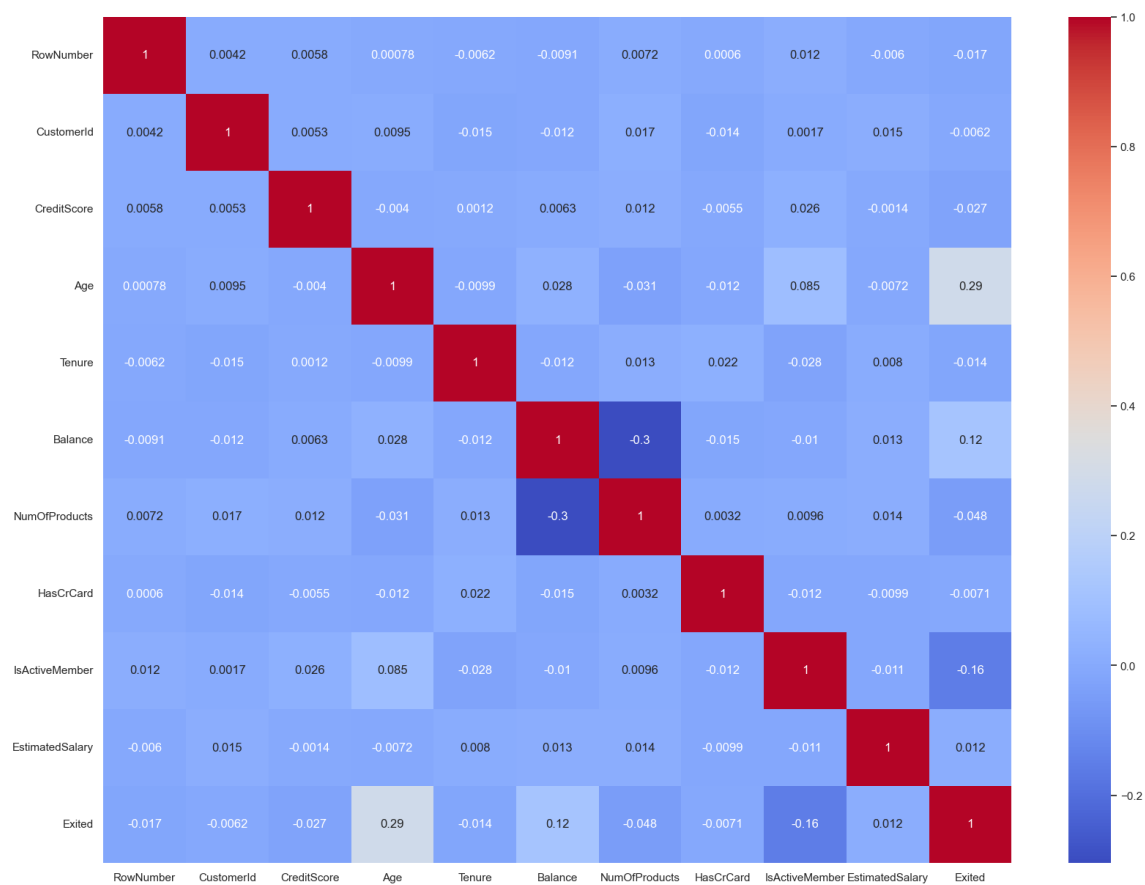


In [277]:

```

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

```



## (5)Missing data

In [278]:

1 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
```

In [279]:

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

In [280]:

1 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]:

1 df["Tenure"].median()

Out[281]:

5.0

In [282]:

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

In [283]:

```
1 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
```

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

In [284]:

```

1 # Imputing null value
2 # pls treat numerical value first and then try below one - most_frequent
3 from sklearn.impute import SimpleImputer
4 imp_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
5 df_imputed = pd.DataFrame(imp_mode.fit_transform(df))
6 df_imputed.columns = df.columns
7 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]:

```
1 df_imputed.info()
```

&lt;class 'pandas.core.frame.DataFrame'&gt;

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]:

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

In [287]:

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

In [288]:

```
1 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]:

```
1 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]:

```

1 #Gender column : OHE
2 df_new['Gender'].value_counts()
3 df_new['Gender'] = df_new['Gender'].astype('category')
4 df_new['Gender'] = df_new['Gender'].cat.codes
5
6 #for Geography column : Dummy
7 df_new = pd.get_dummies(df_new, columns=['Geography'])
8 df_new = df_new.drop(['Geography_France'], axis=1)

```

In [291]:

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

Out[291]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
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]:

```

1 df_new_Ncat = df.select_dtypes(exclude='object')
2 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

```

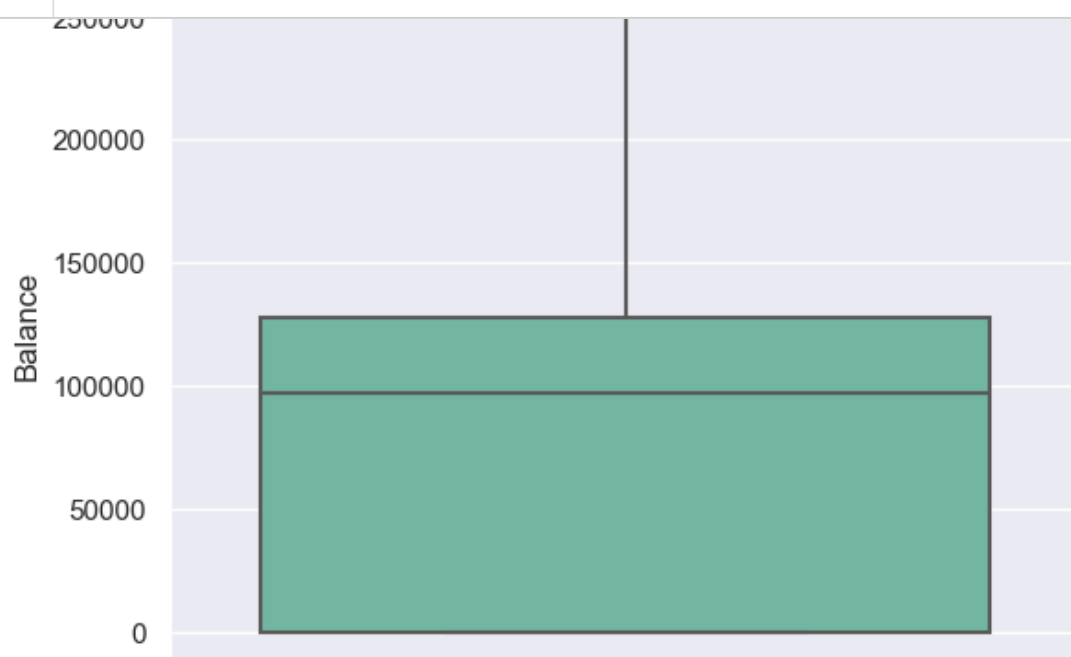


In [293]:

```
1 # outlier check on all independent feature except dependent variable
2 outlier_list = list(df_new_Ncat.columns)
3
4 DepV_remove=['Exited']
5
6 for i in DepV_remove:
7     outlier_list.remove(i)
8     outlier_list
```

In [294]:

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

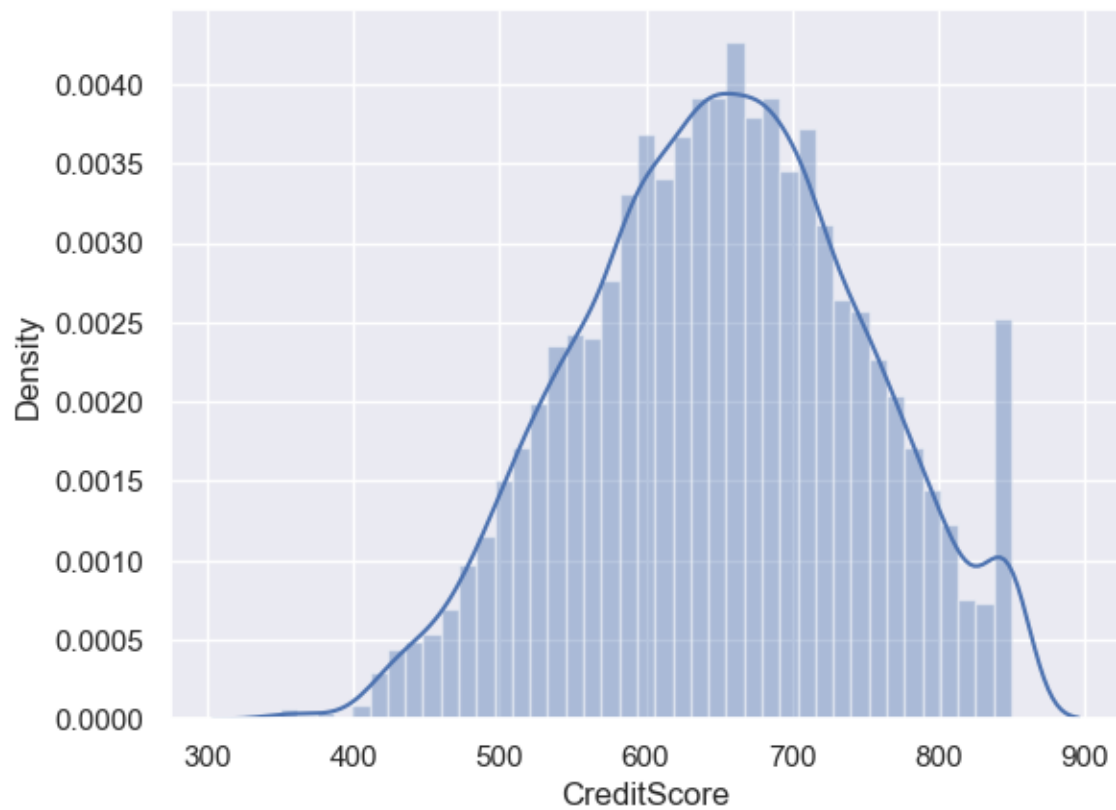


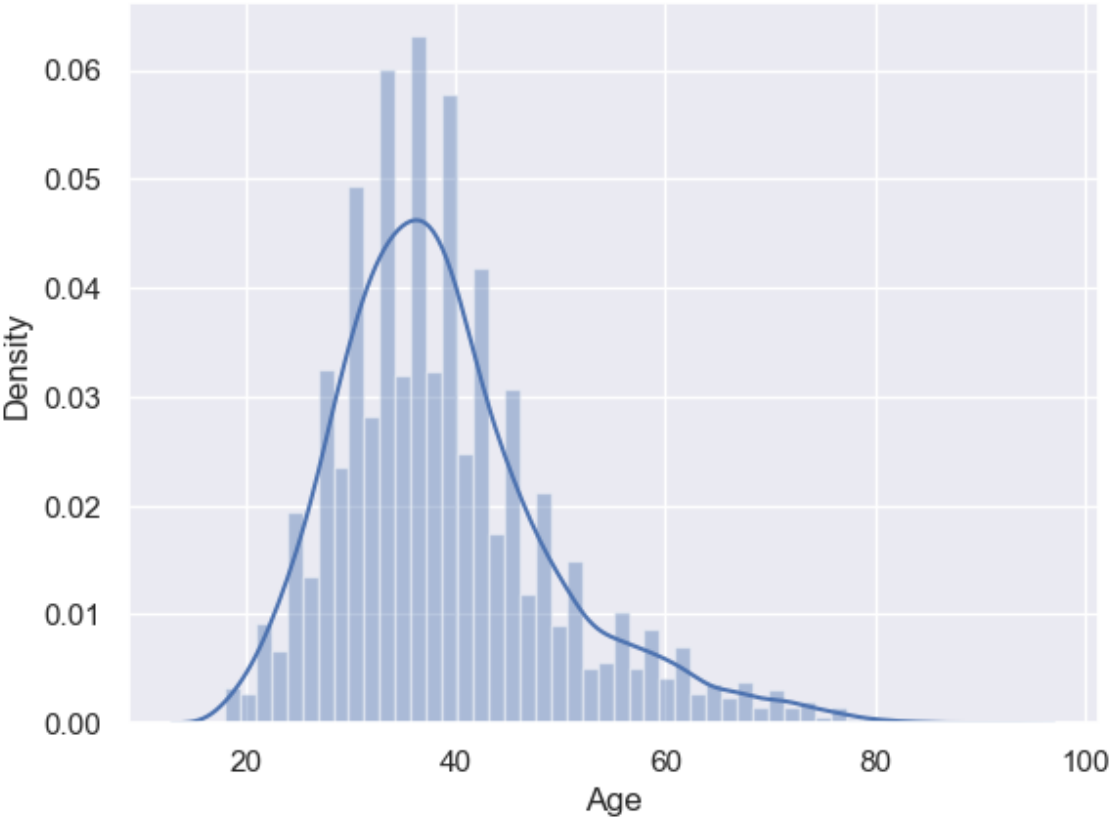
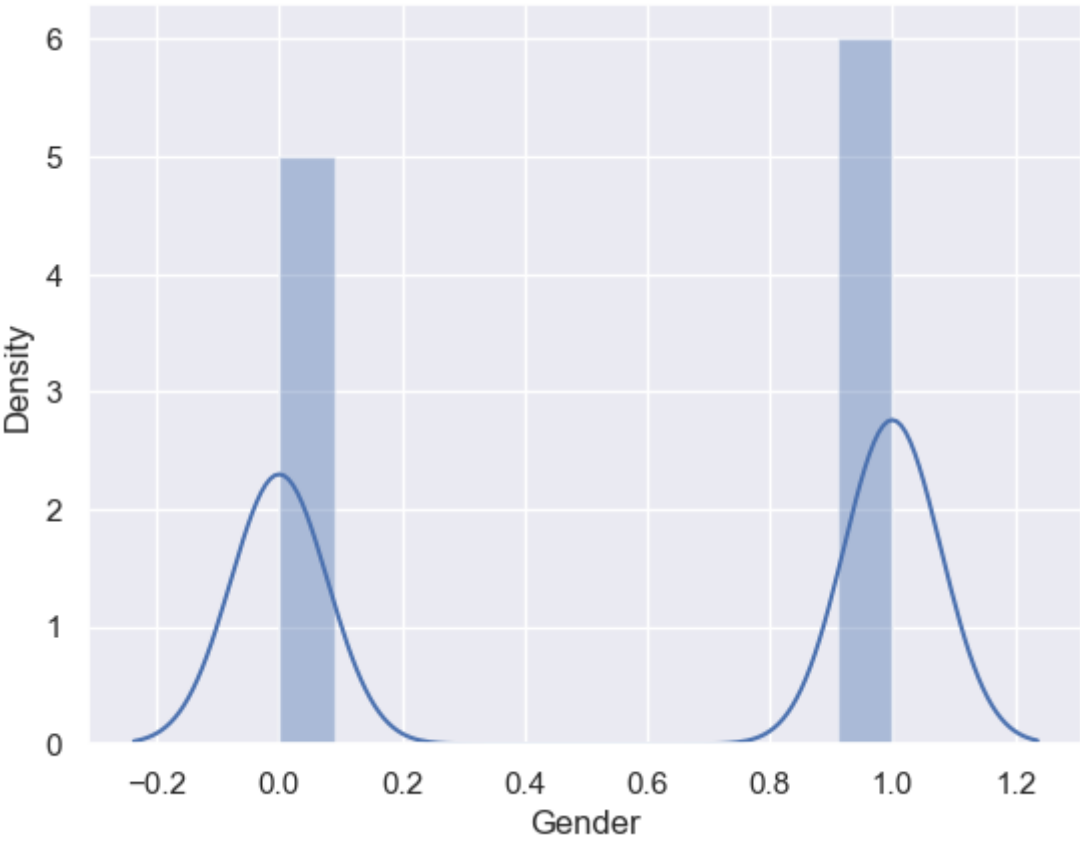
In [295]:

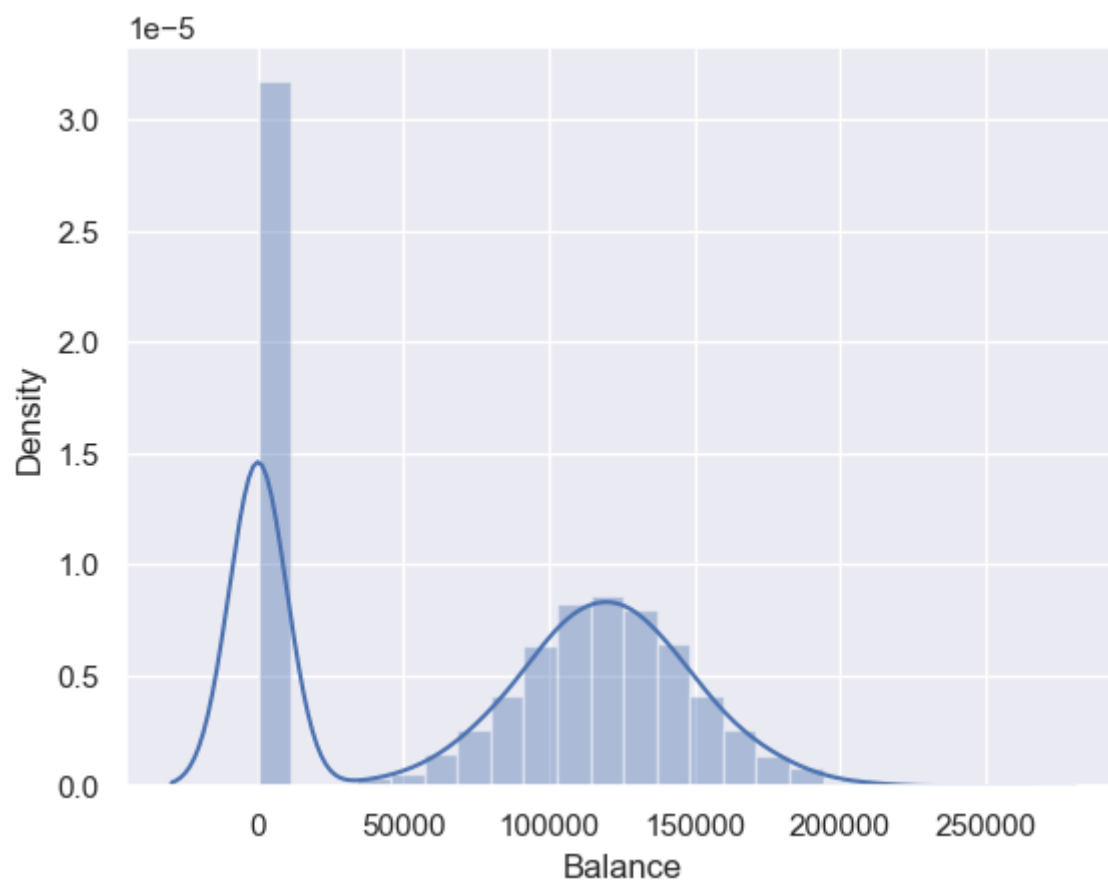
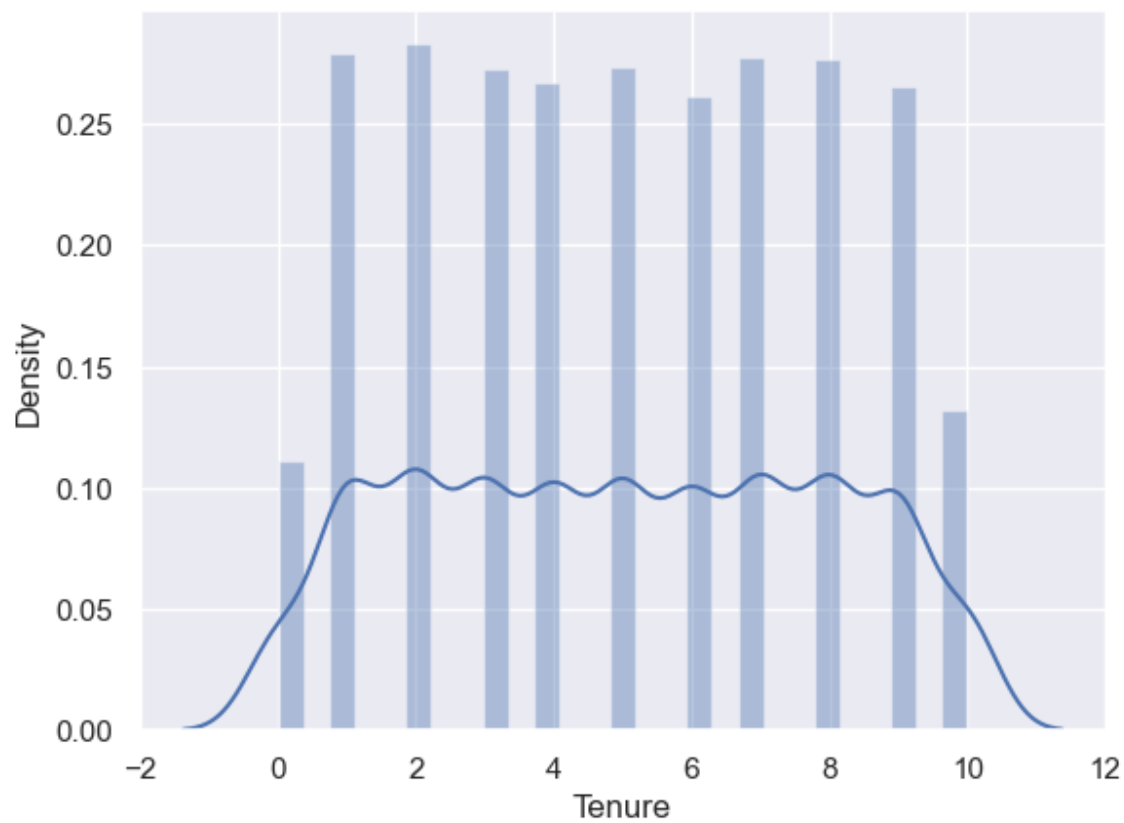
```
1 # Find the distribution of the dataset
```

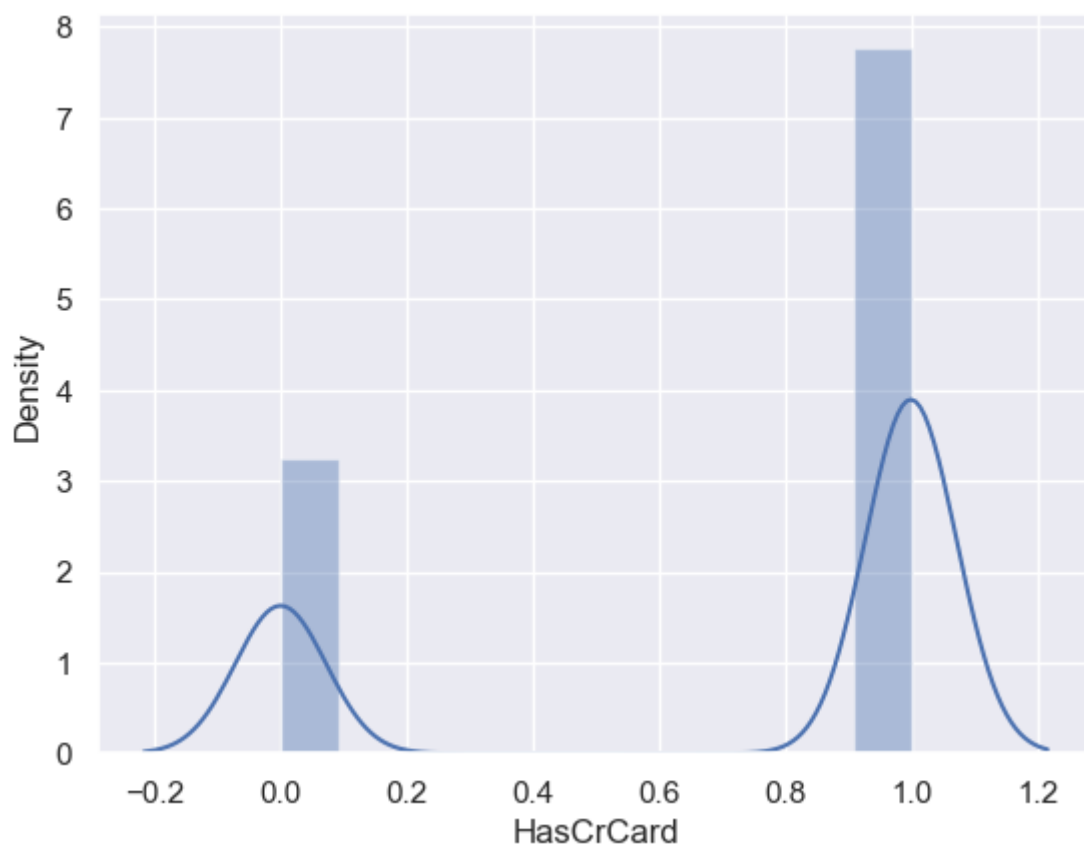
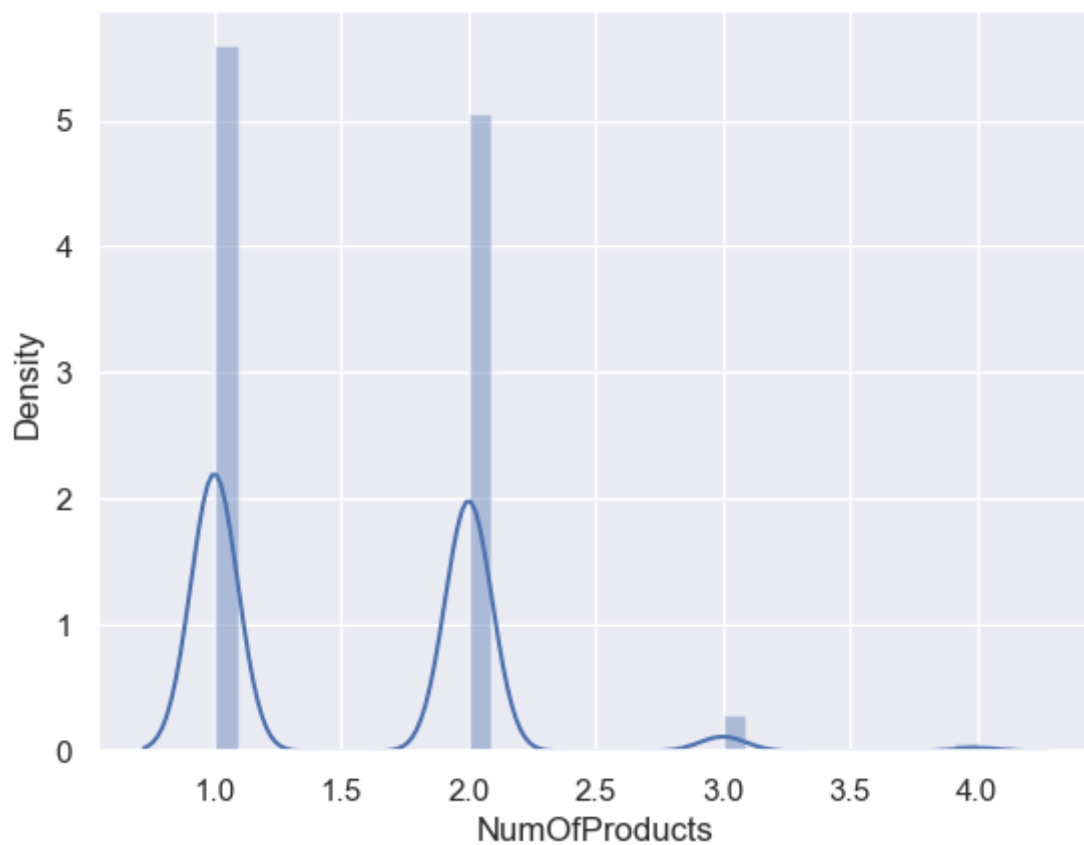
In [296]:

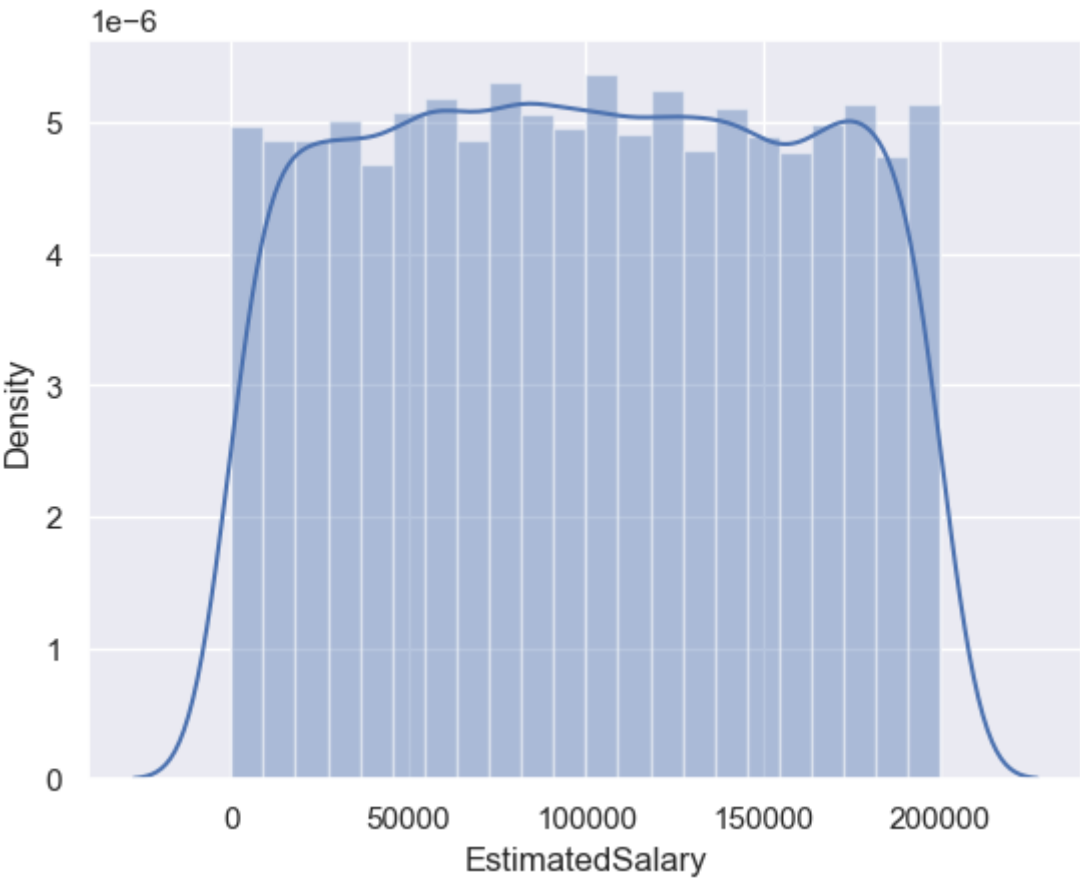
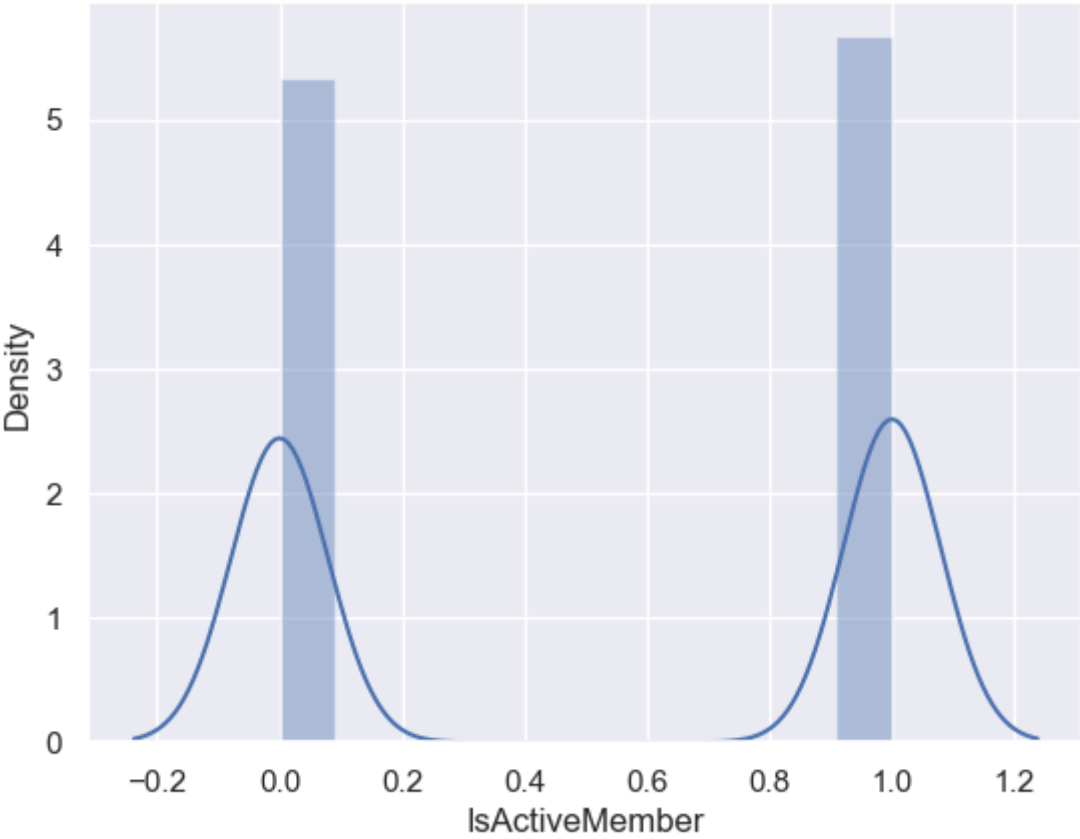
```
1 # Find the distribution of the dataset
2 def distplots(col):
3     sns.distplot(df_new[col])
4     plt.show()
5
6 for i in list(df_new.select_dtypes(exclude=['object']).columns)[0:]:
7     distplots(i)
```

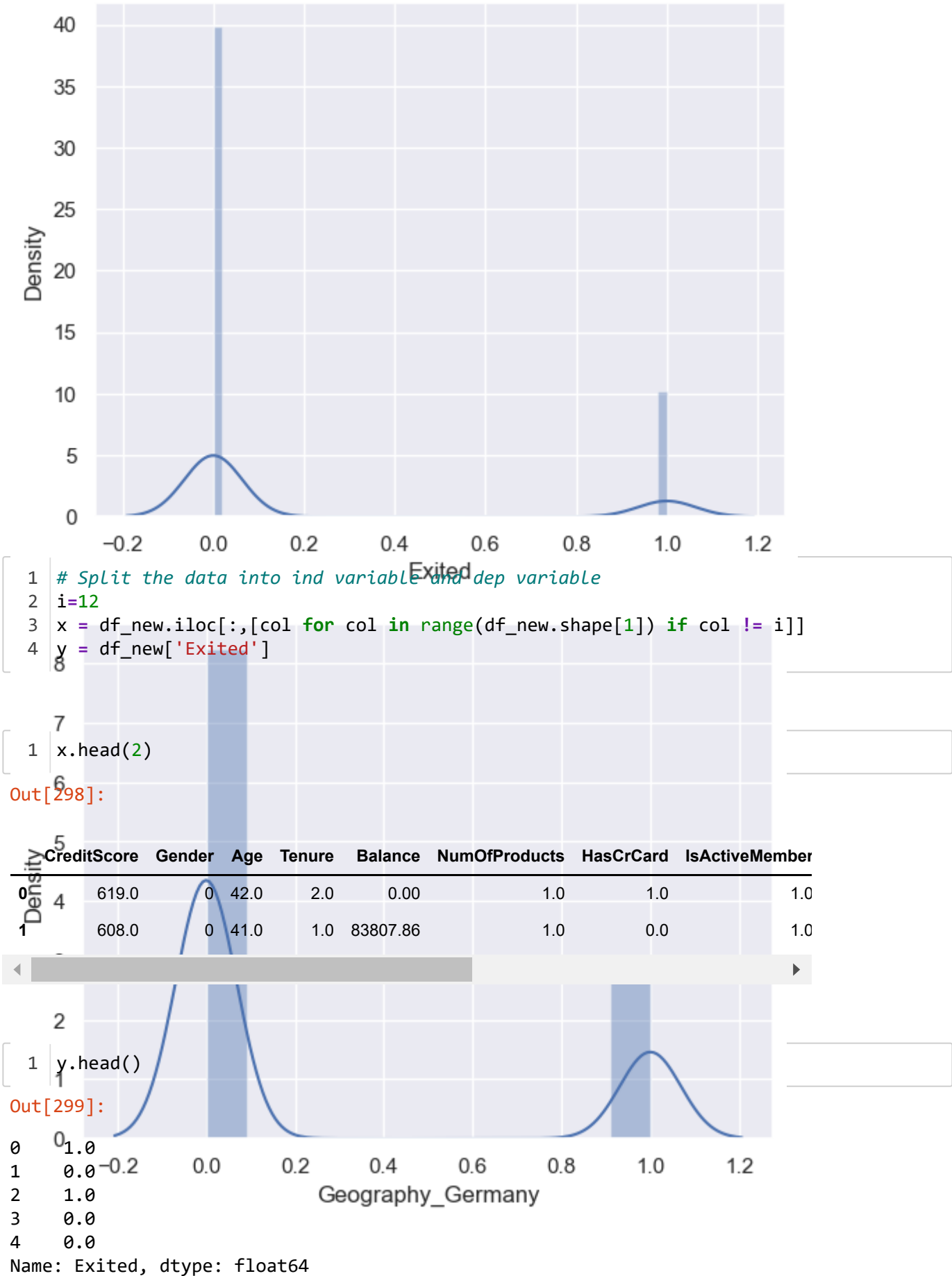












(11)Imbalance data

```

1 # Check imbalance dataset
2 y.value_counts()

```

Out[300]:

```

0.0    7963
1.0    2037
Name:Exited, dtype: int64

```

```

1 from imblearn.over_sampling import SMOTE
2 smote = SMOTE()
3 x, y = smote.fit_resample(x,y)
4 y.value_counts()

```

Out[301]:

```

1.0    7963
0.0    7963
Name:Exited, dtype: int64

```



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

In [302]:

```

1 from sklearn.preprocessing import StandardScaler
2 sc = StandardScaler()
3 sc_x = sc.fit_transform(x)
4 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]:

```
1 x.head()
```

Out[303]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
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]:

```
1 x.shape
```

Out[304]:

(15926, 12)

In [305]:

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

## (12) splitting in train and test

In [306]:

```
1 # split the training data into train and test
2 from sklearn.model_selection import train_test_split
3 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]:

```
1  #Machine Learning classification models
2  from sklearn.linear_model import LogisticRegression
3  from sklearn.tree import DecisionTreeClassifier
4  from sklearn.ensemble import RandomForestClassifier
5  from sklearn.ensemble import BaggingClassifier
6  from sklearn.ensemble import AdaBoostClassifier
7  from sklearn.ensemble import GradientBoostingClassifier
8  from xgboost import XGBClassifier
9  from sklearn.svm import SVC
10 from sklearn.neighbors import KNeighborsClassifier
11 from sklearn.naive_bayes import GaussianNB
12 from sklearn.naive_bayes import BernoulliNB
13
14 #voting method
15 from sklearn.ensemble import VotingClassifier
16
17 #to see the result
18 from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
19
20
21
```

**(i) Accuracy check in Test data**



In [308]:

```
1 # Applying all the model together
2
3 # Applying all the model together
4
5 # LogisticRegression
6 logistic = LogisticRegression()
7 logistic.fit(x_train, y_train)
8 y_train_pred_logistic = logistic.predict(x_train)
9 accuracy_logistic = accuracy_score(y_train, y_train_pred_logistic)
10
11 # DecisionTree
12 dtree = DecisionTreeClassifier()
13 dtree.fit(x_train, y_train)
14 y_train_pred_dtree = dtree.predict(x_train)
15 accuracy_dtree = accuracy_score(y_train, y_train_pred_dtree)
16
17 # RandomForest
18 rfmodel = RandomForestClassifier()
19 rfmodel.fit(x_train, y_train)
20 y_train_pred_rfmodel = rfmodel.predict(x_train)
21 accuracy_rfmodel = accuracy_score(y_train, y_train_pred_rfmodel)
22
23 # BaggingClassifier
24 bagg = BaggingClassifier()
25 bagg.fit(x_train, y_train)
26 y_train_pred_bagg = bagg.predict(x_train)
27 accuracy_bagg = accuracy_score(y_train, y_train_pred_bagg)
28
29 # AdaBoostClassifier
30 ada = AdaBoostClassifier()
31 ada.fit(x_train, y_train)
32 y_train_pred_ada = ada.predict(x_train)
33 accuracy_ada = accuracy_score(y_train, y_train_pred_ada)
34
35 # GradientBoostingClassifier
36 gdb = GradientBoostingClassifier()
37 gdb.fit(x_train, y_train)
38 y_train_pred_gdb = gdb.predict(x_train)
39 accuracy_gdb = accuracy_score(y_train, y_train_pred_gdb)
40
41 # XGBClassifier = RF + GDBosting - Lambda - regularisation, gamma - autopruning, e
42 xgb = XGBClassifier()
43 xgb.fit(x_train, y_train)
44 y_train_pred_xgb = xgb.predict(x_train)
45 accuracy_xgb = accuracy_score(y_train, y_train_pred_xgb)
46
47
48 # SVM
49 svc = SVC()
50 svc.fit(x_train, y_train)
51 y_train_pred_svc = svc.predict(x_train)
52 accuracy_svc = accuracy_score(y_train, y_train_pred_svc)
53
54 # KNN
55 knn = KNeighborsClassifier()
56 knn.fit(x_train, y_train)
57 y_train_pred_knn = knn.predict(x_train)
58 accuracy_knn = accuracy_score(y_train, y_train_pred_knn)
59
```

```

60 # GaussianNB
61 Gnb = GaussianNB()
62 Gnb.fit(x_train, y_train)
63 y_train_pred_Gnb = naive_gb.predict(x_train)
64 accuracy_Gnb = accuracy_score(y_train, y_train_pred_Gnb)
65
66 # BernoulliNB
67 Bnb = BernoulliNB()
68 Bnb.fit(x_train, y_train)
69 y_train_pred_Bnb = naive_bn.predict(x_train)
70 accuracy_Bnb = accuracy_score(y_train, y_train_pred_Bnb)

```

In [309]:

```

1 evc = VotingClassifier(estimators=[('logistic', logistic), ('dtree', dtree), ('rfmodel',
2                                     ('gdb', gdb), ('xgb', xgb), ('svc', svc), ('knn', knn),
3                                     ('Gnb', Gnb), ('Bnb', Bnb)], voting='hard')
4
5 model_evc = evc.fit(x_train, y_train)
6 y_train_pred_evc = evc.predict(x_train)
7 accuracy_evc = accuracy_score(y_train, y_train_pred_evc)

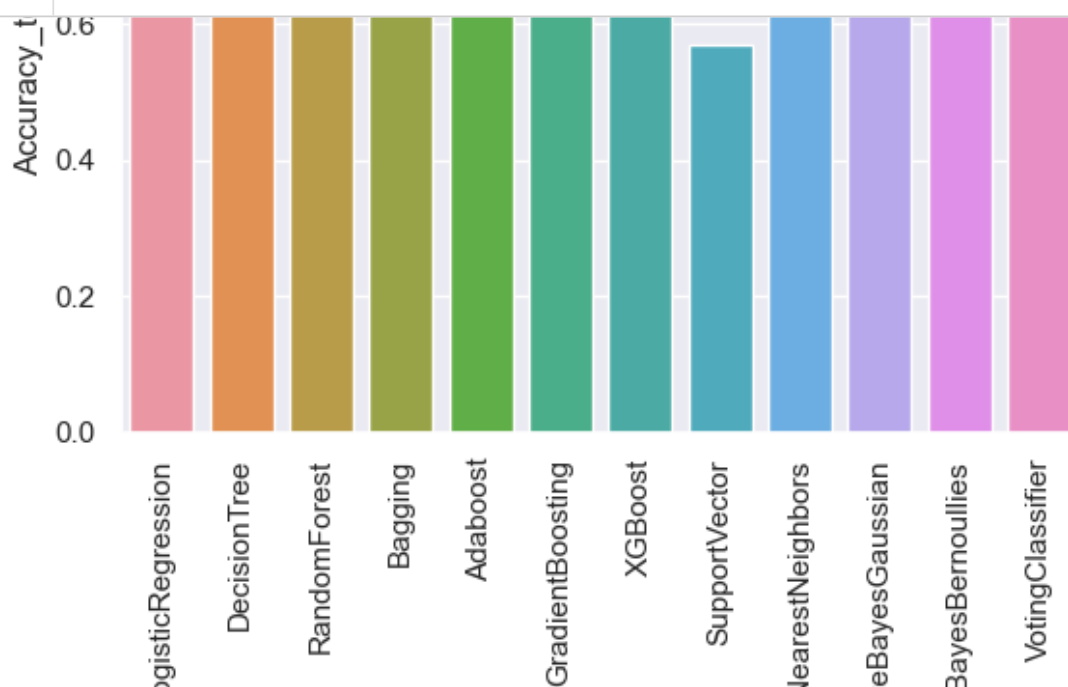
```

In [331]:

```

1 Method_Used = ['LogisticRegression', 'DecisionTree', 'RandomForest', 'Bagging', 'Adaboost',
2               'GradientBoosting', 'XGBoost', 'SupportVector', 'KNearestNeighbors',
3               'NaiveBayesGaussian', 'NaiveBayesBernoullies', 'VotingClassifier']
4
5 Accuracy_test = [accuracy_logistic, accuracy_dtree, accuracy_rfmodel, accuracy_bagging,
6                  accuracy_xgb, accuracy_svc, accuracy_knn, accuracy_Gnb, accuracy_Bnb, accuracy_evc]
7
8
9
10 final_accuracy_test = pd.DataFrame({'Method Used': Method_Used, "Accuracy_test": Accuracy_test})
11 print(final_accuracy_test)
12
13 charts = sns.barplot(x="Method Used", y = 'Accuracy_test', data=final_accuracy_test)
14 charts.set_xticklabels(charts.get_xticklabels(), rotation=90)
15 print(charts)

```



**(ii)Accuracy check in Train data**



In [332]:

```
1 # Applying all the model together
2
3 # LogisticRegression
4 logistic = LogisticRegression()
5 logistic.fit(x_train, y_train)
6 y_train_pred_logistic = logistic.predict(x_train)
7 accuracy_logistic = accuracy_score(y_train, y_train_pred_logistic)
8
9 # DecisionTree
10 dtree = DecisionTreeClassifier()
11 dtree.fit(x_train, y_train)
12 y_train_pred_dtree = dtree.predict(x_train)
13 accuracy_dtree = accuracy_score(y_train, y_train_pred_dtree)
14
15 # RandomForest
16 rfmodel = RandomForestClassifier()
17 rfmodel.fit(x_train, y_train)
18 y_train_pred_rfmodel = rfmodel.predict(x_train)
19 accuracy_rfmodel = accuracy_score(y_train, y_train_pred_rfmodel)
20
21 # BaggingClassifier
22 bagg = BaggingClassifier()
23 bagg.fit(x_train, y_train)
24 y_train_pred_bagg = bagg.predict(x_train)
25 accuracy_bagg = accuracy_score(y_train, y_train_pred_bagg)
26
27 # AdaBoostClassifier
28 ada = AdaBoostClassifier()
29 ada.fit(x_train, y_train)
30 y_train_pred_ada = ada.predict(x_train)
31 accuracy_ada = accuracy_score(y_train, y_train_pred_ada)
32
33 # GradientBoostingClassifier
34 gdb = GradientBoostingClassifier()
35 gdb.fit(x_train, y_train)
36 y_train_pred_gdb = gdb.predict(x_train)
37 accuracy_gdb = accuracy_score(y_train, y_train_pred_gdb)
38
39 # XGBClassifier = RF + GDBosting - Lambda - regularisation, gamma - autoprunning, e
40 xgb = XGBClassifier()
41 xgb.fit(x_train, y_train)
42 y_train_pred_xgb = xgb.predict(x_train)
43 accuracy_xgb = accuracy_score(y_train, y_train_pred_xgb)
44
45
46 # SVM
47 svc = SVC()
48 svc.fit(x_train, y_train)
49 y_train_pred_svc = svc.predict(x_train)
50 accuracy_svc = accuracy_score(y_train, y_train_pred_svc)
51
52 # KNN
53 knn = KNeighborsClassifier()
54 knn.fit(x_train, y_train)
55 y_train_pred_knn = knn.predict(x_train)
56 accuracy_knn = accuracy_score(y_train, y_train_pred_knn)
57
58 # GaussianNB
59 Gnb = GaussianNB()
```



```
60 Gnb.fit(x_train, y_train)
61 y_train_pred_Gnb = naive_gb.predict(x_train)
62 accuracy_Gnb = accuracy_score(y_train, y_train_pred_Gnb)
63
64 # BernoulliNB
65 Bnb = BernoulliNB()
66 Bnb.fit(x_train, y_train)
67 y_train_pred_Bnb = naive_bn.predict(x_train)
68 accuracy_Bnb = accuracy_score(y_train, y_train_pred_Bnb)
```

In [333]:

```
1 evc = VotingClassifier(estimators=[('logistic', logistic), ('dtree', dtree), ('rfmodel',
2                                     ('gdb', gdb), ('xgb', xgb), ('svc', svc), ('knn', knn),
3                                     ('Gnb', Gnb), ('Bnb', Bnb)], voting='hard')
4
5 model_evc = evc.fit(x_train, y_train)
6 y_train_pred_evc = evc.predict(x_train)
7 accuracy_evc = accuracy_score(y_train, y_train_pred_evc)
```

In [334]:

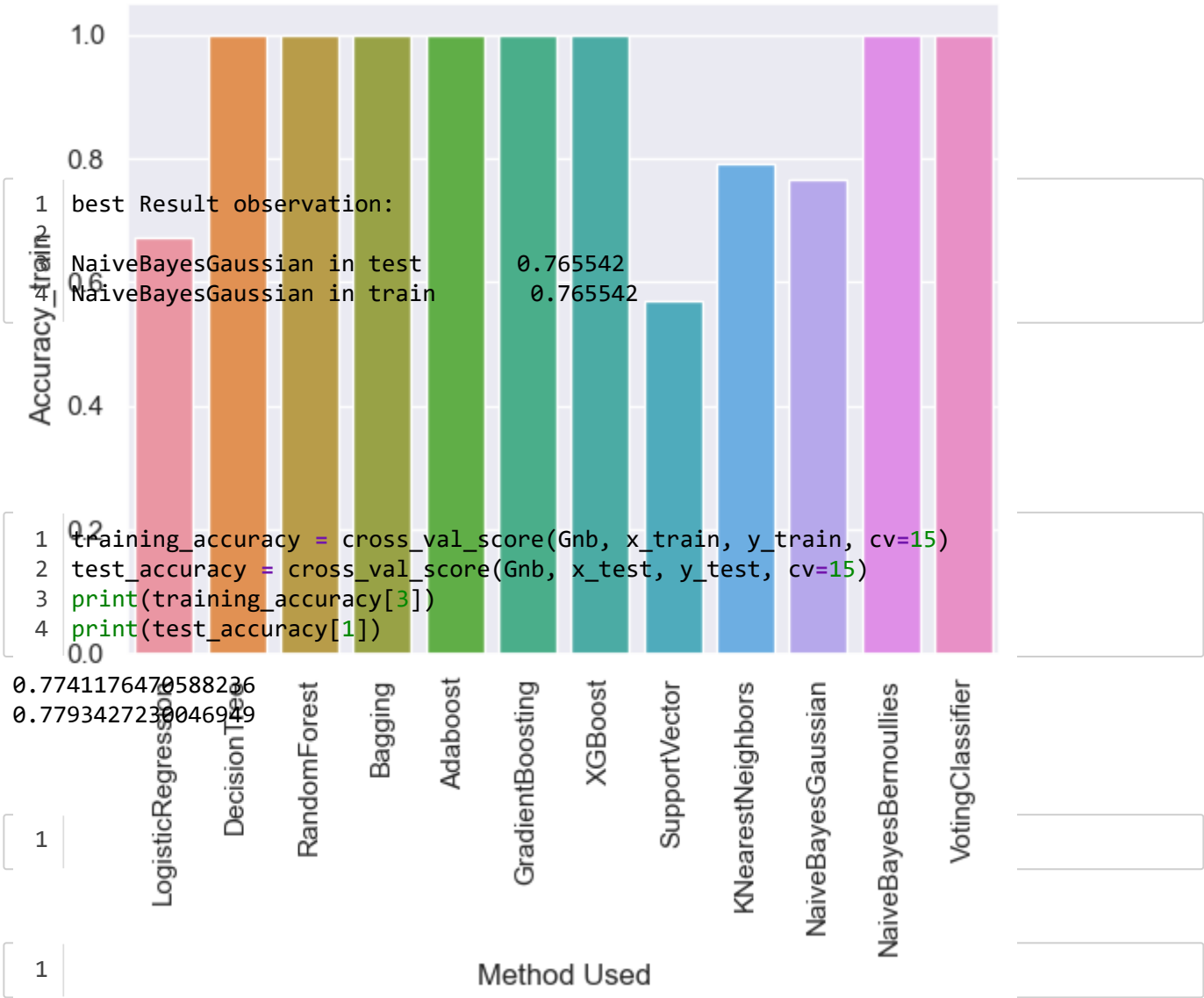
```

1 Method_Used = ['LogisticRegression', 'DecisionTree', 'RandomForest', 'Bagging', 'Adaboos
2               'GradientBoosting', 'XGBoost', 'SupportVector', 'KNearestNeighbors',
3               'NaiveBayesGaussian', 'NaiveBayesBernoullies', 'VotingClassifier']
4
5 Accuracy_train = [accuracy_logistic, accuracy_dtree, accuracy_rfmodel, accuracy_bagg
6                   accuracy_xgb, accuracy_svc, accuracy_knn, accuracy_Gnb, accuracy_Bnb, accur
7
8
9
10 final_accuracy_train = pd.DataFrame({'Method Used': Method_Used, "Accuracy_train": 1
11 print(final_accuracy_train)
12
13 charts = sns.barplot(x="Method Used", y = 'Accuracy_train', data=final_accuracy_train)
14 charts.set_xticklabels(charts.get_xticklabels(), rotation=90)
15 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)



In [ ]:

1

In [ ]:

1

In [ ]:

1

In [ ]:

1

In [ ]:

1

