1.Importing Libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
```

2. Load Dataset

```
from google.colab import drive
drive.mount('/content/drive')
cd /content/drive/MyDrive/Colab Notebooks
```

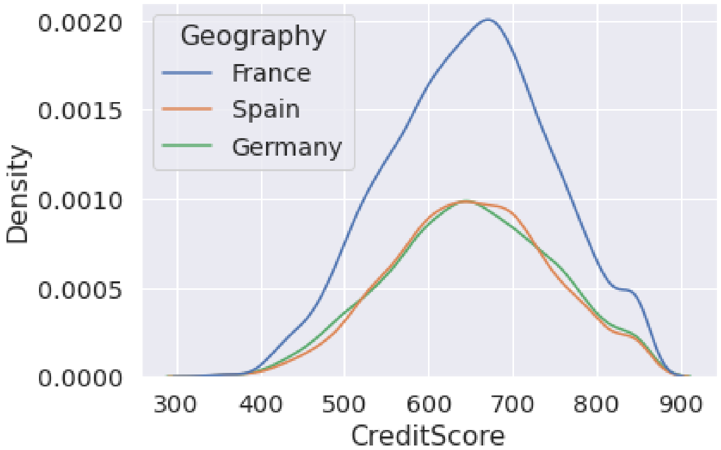
3.visualization

```
 sns.set_style('darkgrid')

sns.set(font_scale=1.3)
ds = pd.read_excel('Churn_Modelling.xlsx')
```

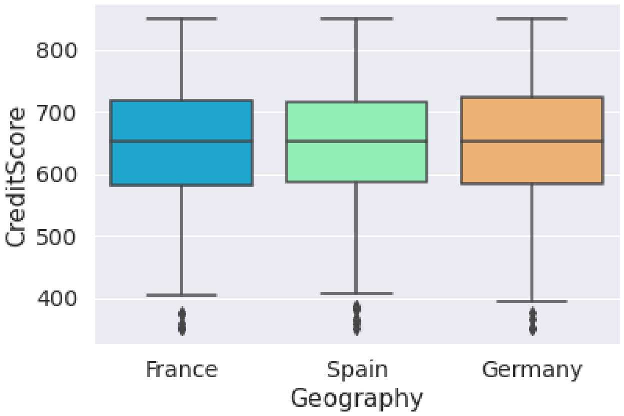
1.univarient

```
   sns.kdeplot(x='CreditScore',data=ds,hue='Geography')
```

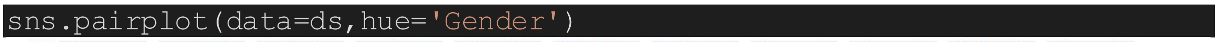


2.Bivarient

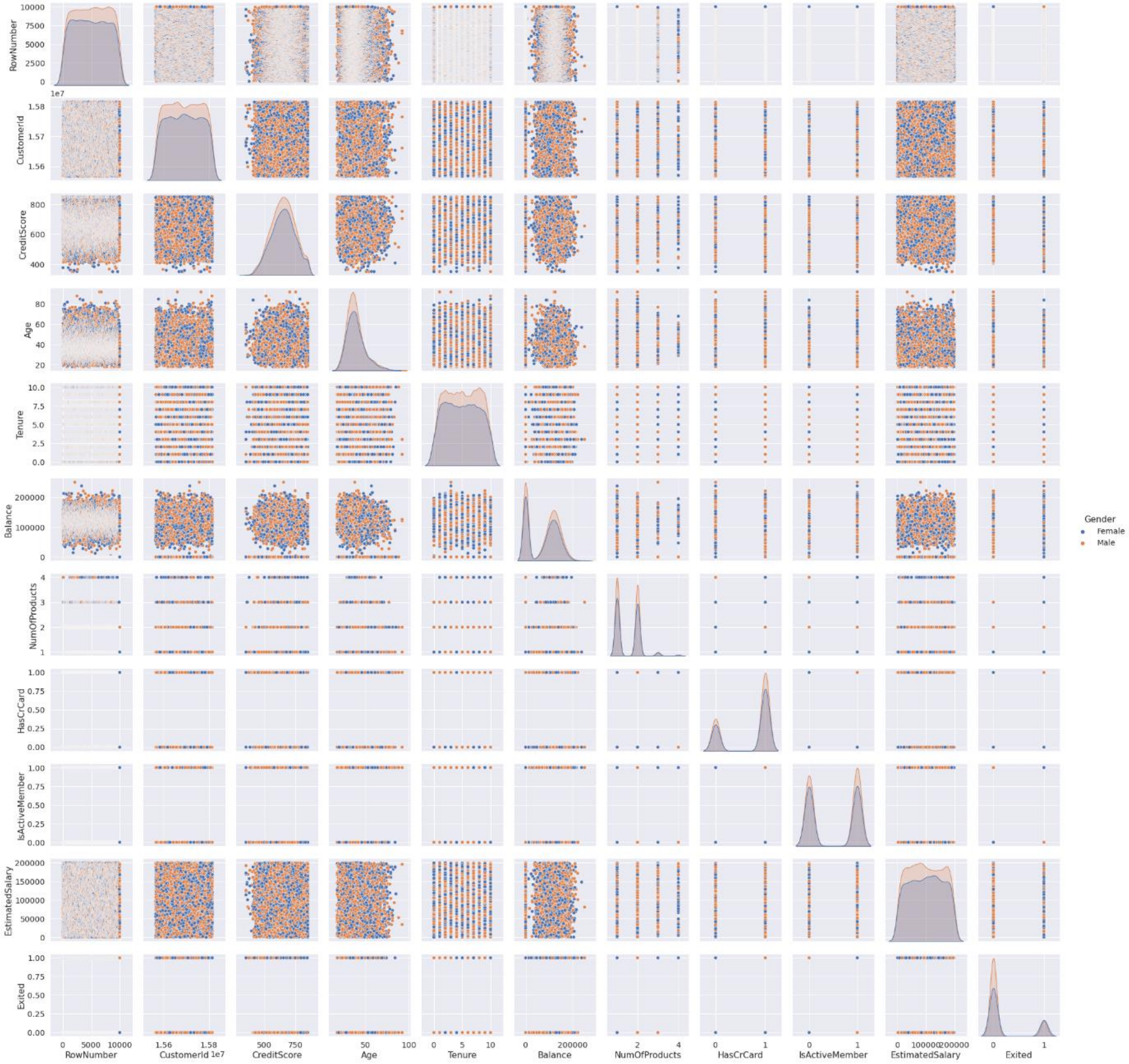
```
 sns.boxplot(x="Geography", y="CreditScore", data=ds,palette='rainbow')
```



3.Multivarient



```
sns.pairplot(data=ds,hue='Gender')
```



4.Discriptive Statistics

```
ds.head()
```

```
          RowNumber     CustomerId     Surname        CreditScore     Geography       Gender  Age
          Tenure   Balance NumOfProducts HasCrCard    IsActiveMember           EstimatedSalary
          Exited

0         1.0      15634602.0      Hargrave          619.0   France   Female  42.0     2.0      0.00      1.0
          1.0      1.0       101348.88          1.0

1         2.0      15647311.0      Hill       608.0   Spain    Female  41.0     1.0      83807.86         1.0
          0.0      1.0       112542.58          0.0

2         3.0      15619304.0      Onio       502.0   France   Female  42.0     8.0      159660.80        3.0
          1.0      0.0       113931.57          1.0
```

```
3       4.0     15701354.0      Boni    699.0   France  Female  39.0    1.0     0.00    2.0     0.0
        0.0     93826.63        0.0

4       5.0     15737888.0      Mitchell        850.0   Spain   Female  43.0    2.0     125510.82
        1.0     1.0     1.0     79084.10        0.0
```

```
ds.describe()
```

```
        RowNumber       CustomerId      CreditScore     Age     Tenure  Balance NumOfProducts
        HasCrCard       IsActiveMember          EstimatedSalary Exited

count   10000.00000     1.000000e+04    10000.000000    10000.000000    10000.000000
        10000.000000    10000.000000    10000.00000     10000.000000    10000.000000
        10000.000000

mean    5000.50000      1.569094e+07    650.528800      38.921800       5.012800
        76485.889288    1.530200        0.70550         0.515100        100090.239881  0.203700

std     2886.89568      7.193619e+04    96.653299       10.487806       2.892174
        62397.405202    0.581654        0.45584         0.499797        57510.492818    0.402769

min     1.00000         1.556570e+07    350.000000      18.000000       0.000000        0.000000
        1.000000        0.00000         0.000000        11.580000       0.000000

25%     2500.75000      1.562853e+07    584.000000      32.000000       3.000000        0.000000
        1.000000        0.00000         0.000000        51002.110000    0.000000

50%     5000.50000      1.569074e+07    652.000000      37.000000       5.000000
        97198.540000    1.000000        1.00000         1.000000        100193.915000  0.000000

75%     7500.25000      1.575323e+07    718.000000      44.000000       7.000000
        127644.240000   2.000000        1.00000         1.000000        149388.247500  0.000000

max     10000.00000     1.581569e+07    850.000000      92.000000       10.000000
        250898.090000   4.000000        1.00000         1.000000        199992.480000  1.000000
```

```
ds.dtypes
```

```
RowNumber         float64

CustomerId        float64

Surname           object

CreditScore       float64

Geography         object

Gender            object

Age               float64

Tenure            float64

Balance           float64

NumOfProducts     float64
```

```
HasCrCard        float64
IsActiveMember    float64
EstimatedSalary   float64
Exited           float64
dtype: object
```

```
ds.skew()
```

```
RowNumber         0.000000
CustomerId        0.001149
CreditScore      -0.071607
Age              1.011320
Tenure           0.010991
Balance         -0.141109
NumOfProducts     0.745568
HasCrCard       -0.901812
IsActiveMember   -0.060437
EstimatedSalary   0.002085
Exited           1.471611
dtype: float64
```

5.Handle missing value

```
ds.isnull().any()
```

```
RowNumber         False
CustomerId        False
Surname           False
CreditScore       False
Geography         False
Gender            False
Age               False
Tenure            False
Balance           False
NumOfProducts     False
```

```
HasCrCard        False
IsActiveMember    False
EstimatedSalary   False
Exited           False
dtype: bool
```

6.Find outliers and replace it

```
ds['CreditScore'].describe()
```

```
count    10000.000000
mean       650.528800
std         96.653299
min        350.000000
25%        584.000000
50%        652.000000
75%        718.000000
max        850.000000
Name: CreditScore, dtype: float64
```

```
ds['Age'].describe()
```

```
count    10000.000000
mean        38.921800
std         10.487806
min         18.000000
25%         32.000000
50%         37.000000
75%         44.000000
max         92.000000
Name: Age, dtype: float64
```

```
ds['Balance'].describe()
```

```
count     10000.000000
mean      76485.889288
```

```
std      62397.405202
min          0.000000
25%          0.000000
50%      97198.540000
75%     127644.240000
max     250898.090000
Name: Balance, dtype: float64
```

```
l=['Balance','Age','CreditScore']
for i in l:
  a=ds[i].quantile(0.1)
  b=ds[i].quantile(0.9)
ds=ds[(ds[i]<b)& (ds[i]>a)]
ds['CreditScore'].describe()
```

```
count   7995.000000
mean     650.995497
std       66.328034
min      522.000000
25%      599.000000
50%      652.000000
75%      704.000000
max      777.000000
Name: CreditScore, dtype: float64
```

```
ds['Balance'].describe()
```

```
count     7995.000000
mean     76183.940614
std      62412.914155
min          0.000000
25%          0.000000
50%      96858.350000
75%     127530.095000
max     250898.090000
Name: Balance, dtype: float64
```

```
ds['Age'].describe()
```

```
count    7995.000000
mean       38.881301
std        10.465870
min        18.000000
25%        32.000000
50%        37.000000
75%        44.000000
max        92.000000
Name: Age, dtype: float64
```

7.Check categorical columns and do encoding

```
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
for i in ds:
 if ds[i].dtype=='object':
  ds[i]=encoder.fit_transform(ds[i])
ds.head()
```

```
     RowNumber  CustomerId   Surname     CreditScore  Geography   Gender  Age
     Tenure  Balance NumOfProducts HasCrCard     IsActiveMember       EstimatedSalary
     Exited

0    1.0     15634602.0    1011    619.0   0     0      42.0    2.0    0.00    1.0    1.0
     1.0     101348.88     1.0

1    2.0     15647311.0    1060    608.0   2     0      41.0    1.0    83807.86       1.0
     0.0     1.0     112542.58     0.0

3    4.0     15701354.0    264     699.0   0     0      39.0    1.0    0.00    2.0    0.0
     0.0     93826.63      0.0

5    6.0     15574012.0    492     645.0   2     1      44.0    8.0    113755.78      2.0
     1.0     0.0     149756.71     1.0

9    10.0    15592389.0    978     684.0   0     1      27.0    2.0    134603.88      1.0
     1.0     1.0     71725.73      0.0
```

8.Dependent and Independent

```
ds.shape
(3354, 14)
x = ds.iloc[:,:13]
y = ds.iloc[:,13]
```

```
x.head()
```

```
      RowNumber   CustomerId   Surname      CreditScore   Geography    Gender  Age
      Tenure  Balance NumOfProducts HasCrCard   IsActiveMember         EstimatedSalary

0     1.0     15634602.0    1011   619.0   0     0      42.0    2.0    0.00    1.0    1.0
      1.0     101348.88

1     2.0     15647311.0    1060   608.0   2     0      41.0    1.0    83807.86       1.0
      0.0     1.0     112542.58

3     4.0     15701354.0    264    699.0   0     0      39.0    1.0    0.00    2.0    0.0
      0.0     93826.63

5     6.0     15574012.0    492    645.0   2     1      44.0    8.0    113755.78      2.0
      1.0     0.0     149756.71

9     10.0    15592389.0    978    684.0   0     1      27.0    2.0    134603.88      1.0
      1.0     1.0     71725.73
```

```
y.head()
```

```
0   1.0

1   0.0

3   0.0

5   1.0

9   0.0

Name: Exited, dtype: float64
```

9.Scale independent values

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

10.split training and testing data

```
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, ran
dom_state=0)
x_train.shape
```

```
(6396, 13)
```

```
y_train.shape
```

```
(6396,)
```

```
x_test.shape
```

```
(1599, 13)
```

```
y_test.shape
```

```
(1599,)
```