```
import pandas as pd
import numpy as np
import seaborn as sns
```

```
ds = pd.read_csv('Churn_Modelling.csv')
```

```
ds.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
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

Next steps: Generate code with ds View recommended plots

```
ds.isna().sum()
```

```
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
```

```
ds.shape
```

```
(10000, 14)
```

```
ds.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
```

```
ds.describe(include='all')
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	A
count	10000.00000	1.000000e+04	10000	10000.000000	10000	10000	10000.0000
unique	NaN	NaN	2932	NaN	3	2	N
top	NaN	NaN	Smith	NaN	France	Male	N
freq	NaN	NaN	32	NaN	5014	5457	N
mean	5000.50000	1.569094e+07	NaN	650.528800	NaN	NaN	38.9218
std	2886.89568	7.193619e+04	NaN	96.653299	NaN	NaN	10.4878
min	1.00000	1.556570e+07	NaN	350.000000	NaN	NaN	18.0000
25%	2500.75000	1.562853e+07	NaN	584.000000	NaN	NaN	32.0000
50%	5000.50000	1.569074e+07	NaN	652.000000	NaN	NaN	37.0000
75%	7500.25000	1.575323e+07	NaN	718.000000	NaN	NaN	44.0000
max	10000.00000	1.581569e+07	NaN	850.000000	NaN	NaN	92.0000

```
ds.columns
```

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

```
ds= ds.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
```

```
ds.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	I
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	

Next steps: Generate code with ds View recommended plots

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

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

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

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

```
ds = pd.get_dummies(ds,columns=["Geography","Gender"],drop_first=True)
```

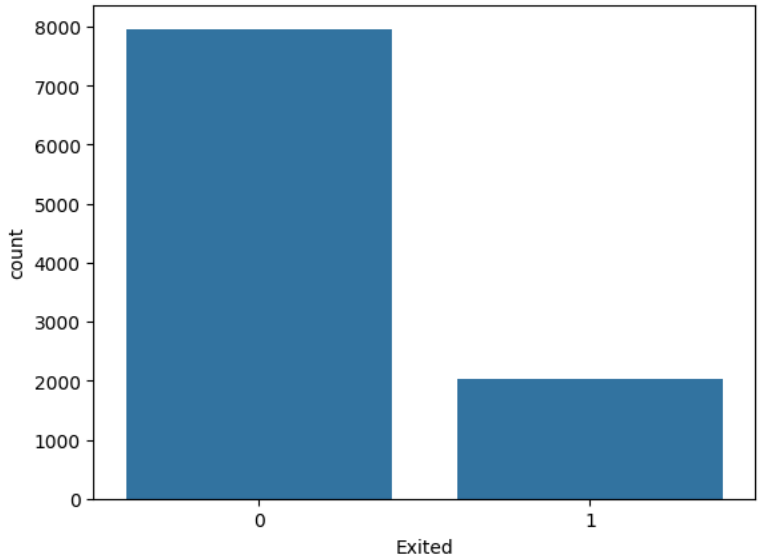
```
ds.head()
```

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

Next steps: Generate code with ds View recommended plots

```
sns.countplot(x=ds['Exited'])  #dataset is imbalanced
```

<Axes: xlabel='Exited', ylabel='count'>




```
X_in = ds.drop('Exited',axis=1)
y_in =ds['Exited']
```

```
y_in.value_counts()
```

```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

```
from imblearn.over_sampling import SMOTEN
X,y = SMOTEN().fit_resample(X_in,y_in)
```

```
y.value_counts()
```

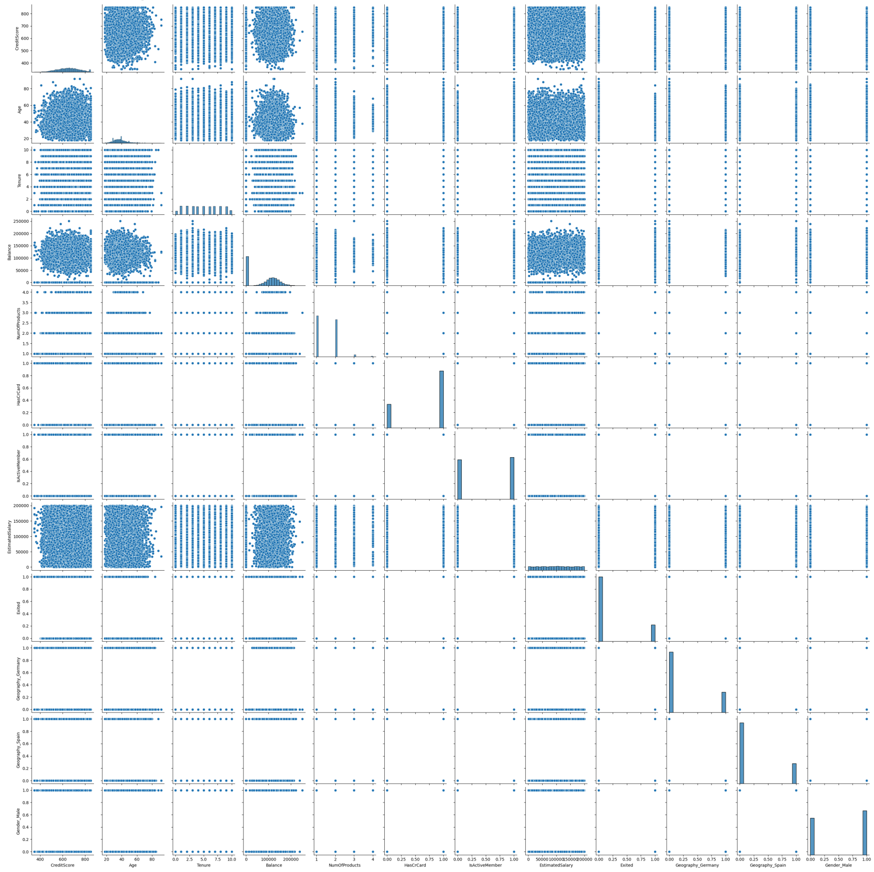
```
Exited
1    7963
0    7963
Name: count, dtype: int64
```

```
ds.corr()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCar
CreditScore	1.000000	-0.003965	0.000842	0.006268	0.012238	-0.0054
Age	-0.003965	1.000000	-0.009997	0.028308	-0.030680	-0.0117
Tenure	0.000842	-0.009997	1.000000	-0.012254	0.013444	0.0225
Balance	0.006268	0.028308	-0.012254	1.000000	-0.304180	-0.0148
NumOfProducts	0.012238	-0.030680	0.013444	-0.304180	1.000000	0.0031
HasCrCard	-0.005458	-0.011721	0.022583	-0.014858	0.003183	1.0000
IsActiveMember	0.025651	0.085472	-0.028362	-0.010084	0.009612	-0.0118
EstimatedSalary	-0.001384	-0.007201	0.007784	0.012797	0.014204	-0.0099
Exited	-0.027094	0.285323	-0.014001	0.118533	-0.047820	-0.0071
Geography_Germany	0.005538	0.046897	-0.000567	0.401110	-0.010419	0.0105
Geography_Spain	0.004780	-0.001685	0.003868	-0.134892	0.009039	-0.0134
Gender_Male	-0.002857	-0.027544	0.014733	0.012087	-0.021859	0.0057

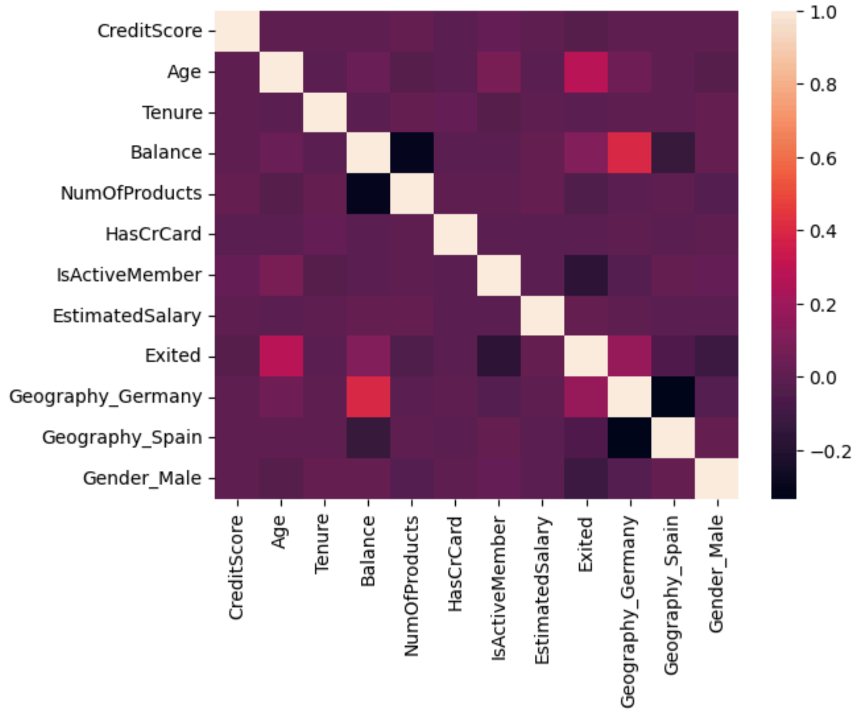
```
sns.pairplot(ds)
```

<seaborn.axisgrid.PairGrid at 0x7e68da3f70d0>




```
sns.heatmap(ds.corr())
```

```
<Axes: >
```




```
#splitting
from sklearn.model_selection import train_test_split
```

```
X_test, X_train, y_test, y_train = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
X_train
```

```
array([[-1.27858987,  0.32900953,  1.15825177, ...,  1.50588589,
        -0.49892083,  0.99187233],
       [-0.41459149, -1.15278912,  0.13091599, ...,  1.50588589,
        -0.49892083, -1.00819428],
       [ 0.49794612,  0.82294242,  0.81580651, ...,  1.50588589,
        -0.49892083,  0.99187233],
       ...,
       [-1.10384862, -1.05400255, -1.23886506, ..., -0.66406094,
        -0.49892083, -1.00819428],
       [-1.12326432,  0.13143638, -1.58131032, ..., -0.66406094,
        -0.49892083, -1.00819428],
       [ 0.97363062, -1.2515757 , -0.21152928, ...,  1.50588589,
        -0.49892083,  0.99187233]])
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
lo = LogisticRegression()
lo.fit(X_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
yp1 = lo.predict(X_test)
```

```
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
print("LOGISTIC REGRESSION")
print("Accuracy= ",accuracy_score(y_test,yp1))
print("Precision= ",precision_score(y_test,yp1))
print("Recall= ",recall_score(y_test,yp1))
print("F1_Score= ",f1_score(y_test,yp1))
print("MSE= ",mean_squared_error(y_test,yp1))
print("MAE= ",mean_absolute_error(y_test,yp1))
```

```
LOGISTIC REGRESSION
Accuracy=  0.7827315541601256
```

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
yp2 = rf.predict(X_test)
```

```
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
print("RANDOM FOREST CLASSIFIER")
print("Accuracy= ",accuracy_score(y_test,yp2))
print("Precision= ",precision_score(y_test,yp2))
print("Recall= ",recall_score(y_test,yp2))
print("F1_Score= ",f1_score(y_test,yp2))
print("MSE= ",mean_squared_error(y_test,yp2))
print("MAE= ",mean_absolute_error(y_test,yp2))
```

```
RANDOM FOREST CLASSIFIER
Accuracy=  0.8793563579277865
Precision=  0.8978583196046128
Recall=  0.8559761269043505
F1_Score=  0.8764171423976843
MSE=  0.1206436420722135
MAE=  0.1206436420722135
```

```
#Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb.fit(X_train,y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
yp3 = gb.predict(X_test)
```

```
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
print("GRADIENT BOOSTING")
print("Accuracy= ",accuracy_score(y_test,yp3))
print("Precision= ",precision_score(y_test,yp3))
print("Recall= ",recall_score(y_test,yp3))
print("F1_Score= ",f1_score(y_test,yp3))
print("MSE= ",mean_squared_error(y_test,yp3))
print("MAE= ",mean_absolute_error(y_test,yp3))
```

```
GRADIENT BOOSTING
```