```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import io
```

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

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age
0	1	15634602	Hargrave	619	France	Female	42
1	2	15647311	Hill	608	Spain	Female	41
2	3	15619304	Onio	502	France	Female	42
3	4	15701354	Boni	699	France	Female	39
4	5	15737888	Mitchell	850	Spain	Female	43
...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39
9996	9997	15569892	Johnstone	516	France	Male	35
9997	9998	15584532	Liu	709	France	Female	36
9998	9999	15682355	Sabbatini	772	Germany	Male	42
9999	10000	15628319	Walker	792	France	Female	28

10000 rows × 14 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.shape
```

(10000, 14)

```
df=df.drop(['RowNumber','CustomerId','Surname'],axis=1)
df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	Ha
0	619	France	Female	42	2	0.00	1	
1	608	Spain	Female	41	1	83807.86	1	
2	502	France	Female	42	8	159660.80	3	
3	699	France	Female	39	1	0.00	2	

4	850	Spain	Female	43	2	125510.82	1

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.isna().any()
df.isna().sum()
```

	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

```
print(df.shape)
df.info()
```

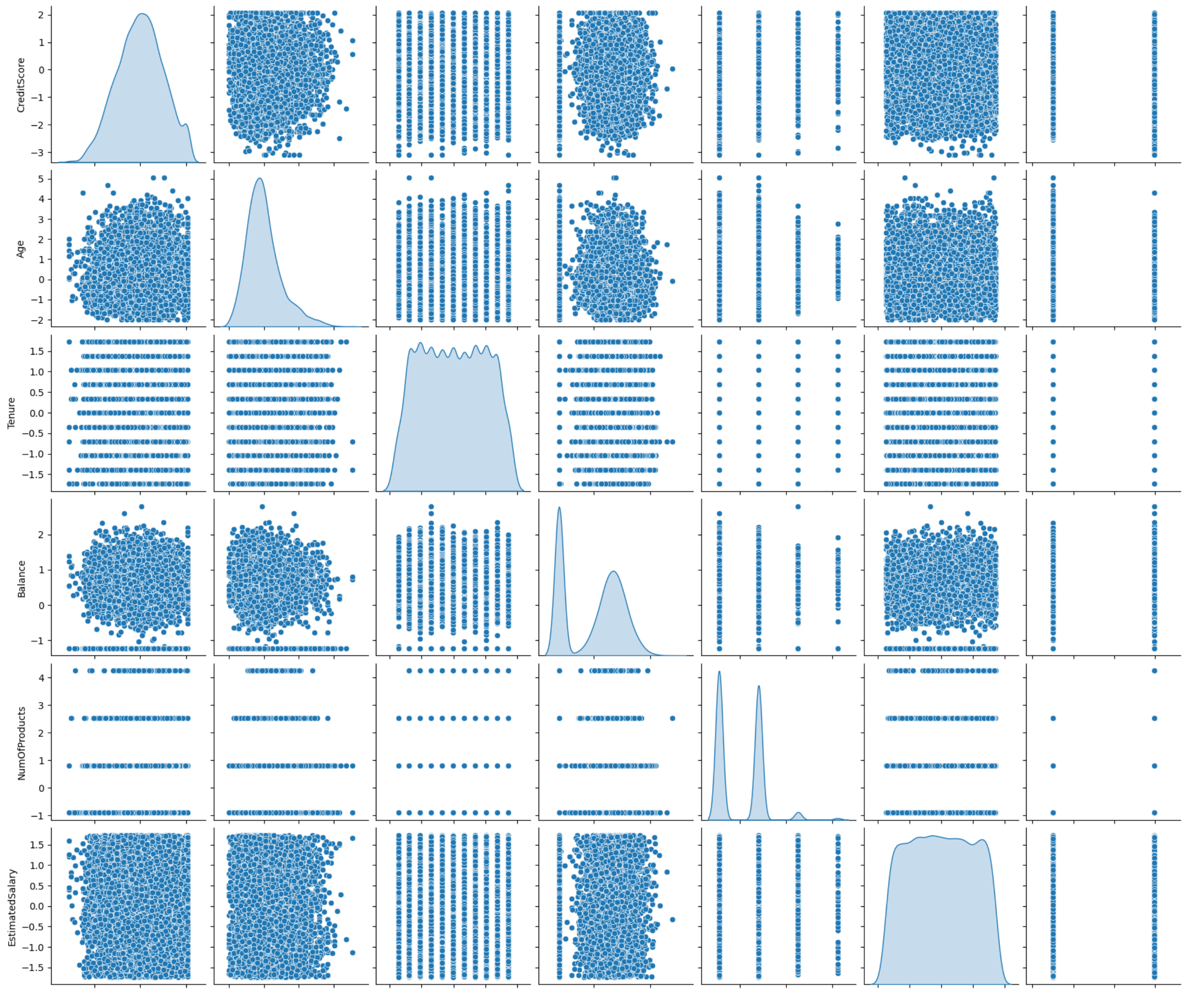
```
(10000, 11)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CreditScore      10000 non-null  int64
 1   Geography        10000 non-null  object
 2   Gender           10000 non-null  object
 3   Age              10000 non-null  int64
 4   Tenure           10000 non-null  int64
 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(2), int64(7), object(2)
memory usage: 859.5+ KB
```

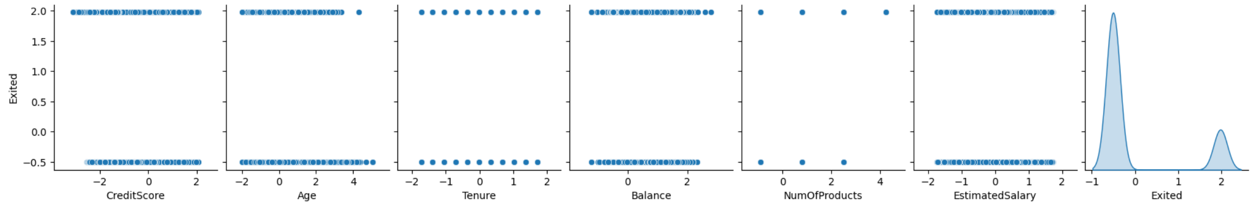
```
df.describe()
```

	CreditScore	Age	Tenure	Balance	NumOfProdu
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000
mean	650.528800	38.921800	5.012800	76485.889288	1.5302
std	96.653299	10.487806	2.892174	62397.405202	0.5816
min	350.000000	18.000000	0.000000	0.000000	1.0000
25%	584.000000	32.000000	3.000000	0.000000	1.0000
50%	652.000000	37.000000	5.000000	97198.540000	1.0000
75%	718.000000	44.000000	7.000000	127644.240000	2.0000
max	850.000000	92.000000	10.000000	250898.090000	4.0000

```
scaler=StandardScaler()
subset=df.drop(['Geography','Gender','HasCrCard','IsActiveMember'],axis=1)
scaled=scaler.fit_transform(subset)
scaled_df=pd.DataFrame(scaled,columns=subset.columns)
sns.pairplot(scaled_df,diag_kind='kde')
```

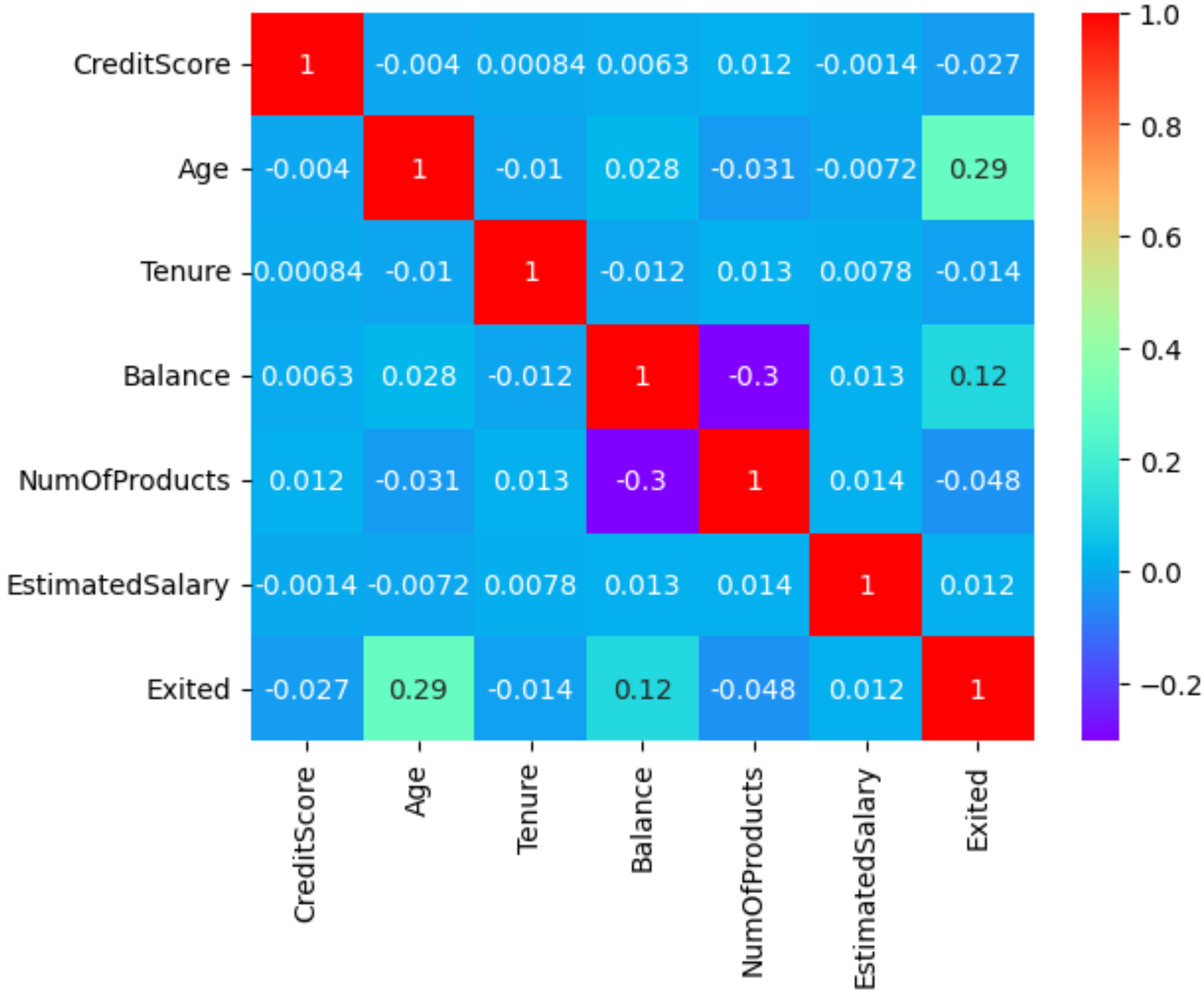
<seaborn.axisgrid.PairGrid at 0x7b66931f94b0>



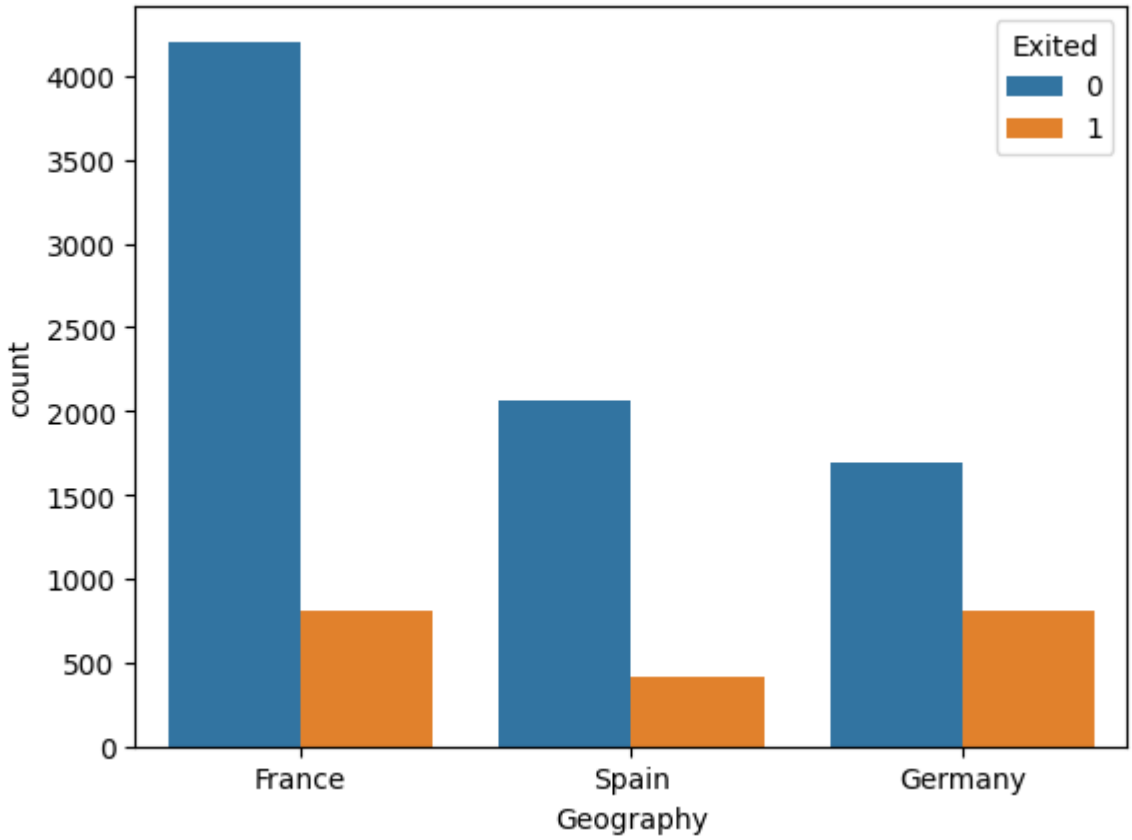


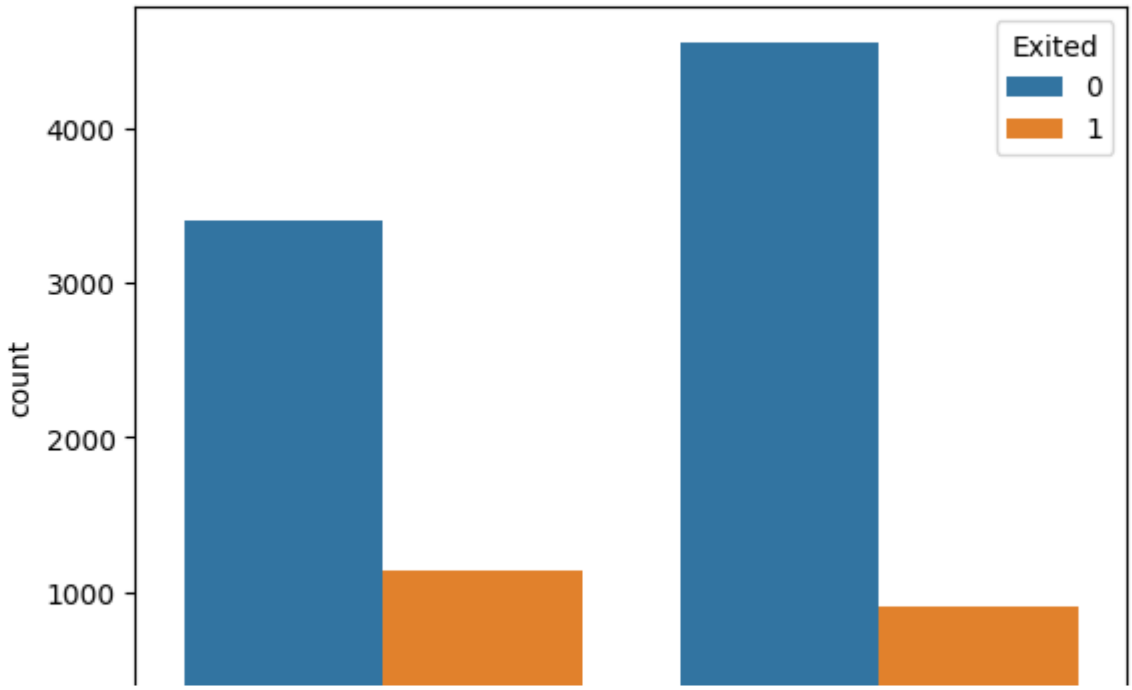

```
sns.heatmap(scaled_df.corr(),annot=True,cmap='rainbow')
```

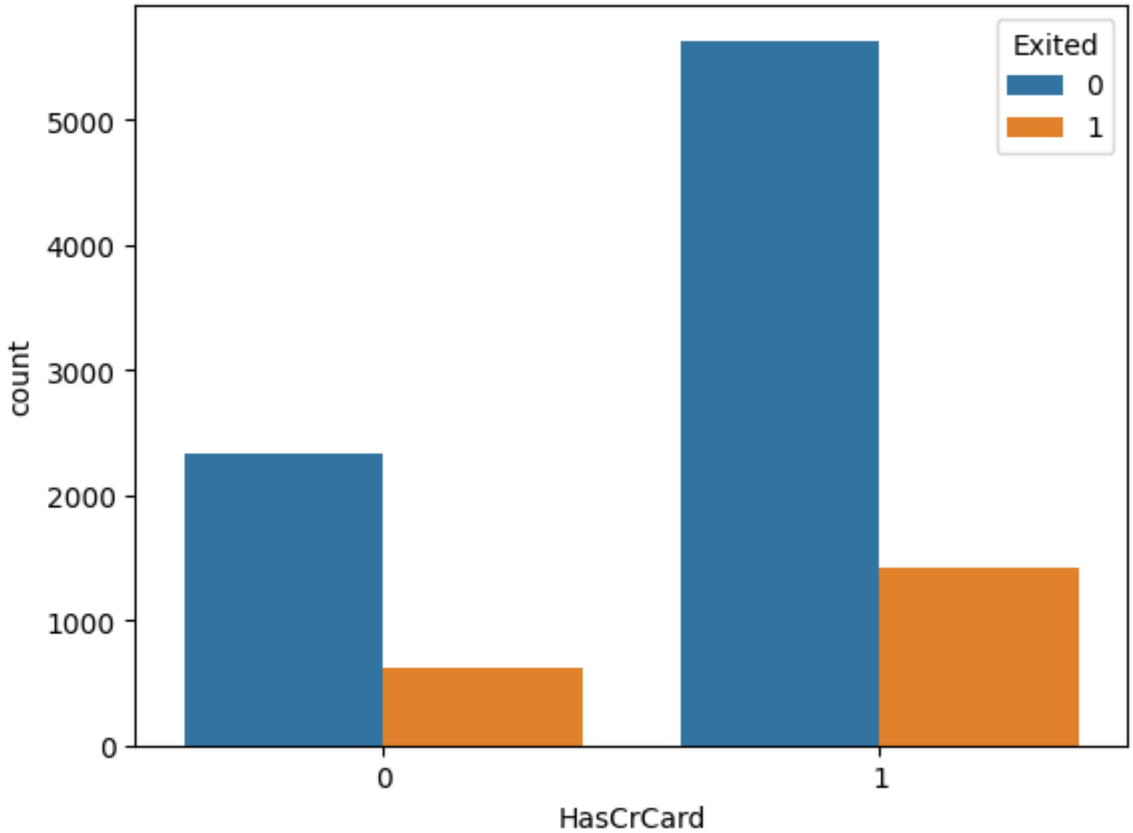
<Axes: >

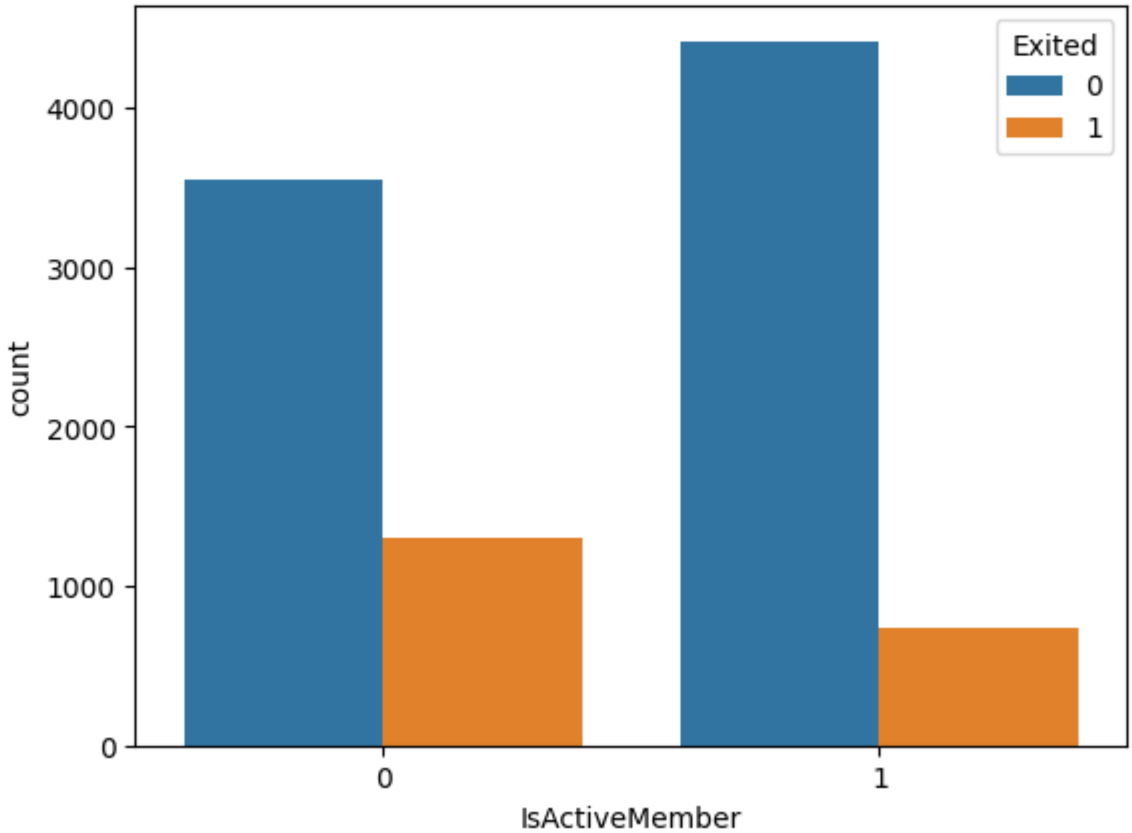


```
sns.countplot(x='Geography',data=df,hue='Exited')
plt.show()
sns.countplot(x='Gender',data=df,hue='Exited')
plt.show()
sns.countplot(x='HasCrCard',data=df,hue='Exited')
plt.show()
sns.countplot(x='IsActiveMember',data=df,hue='Exited')
plt.show()
```



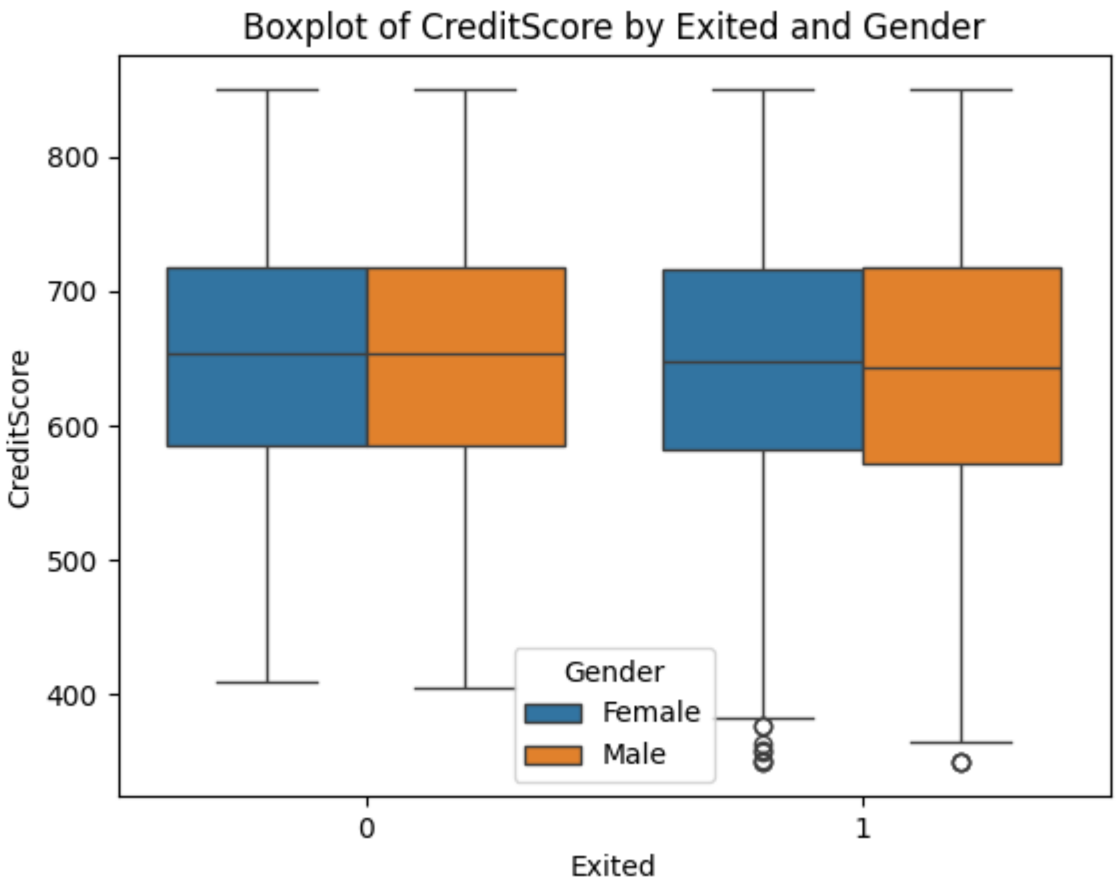


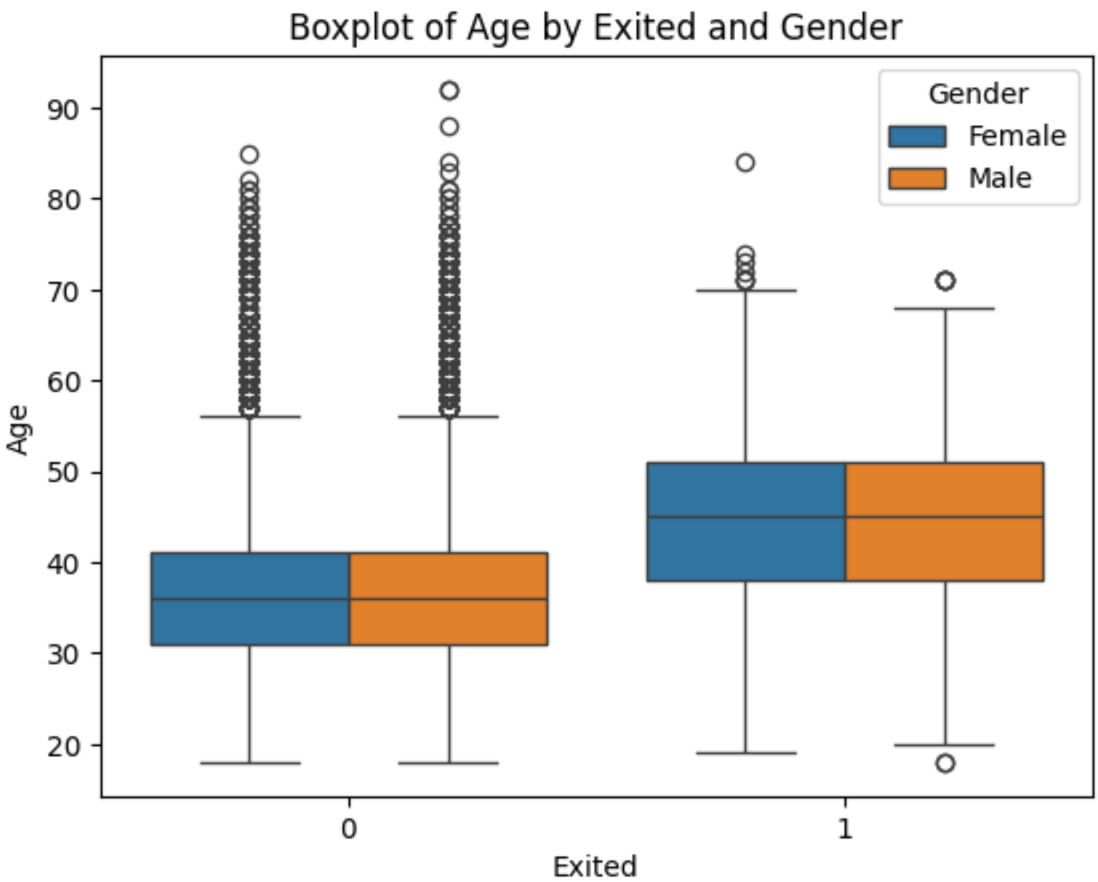




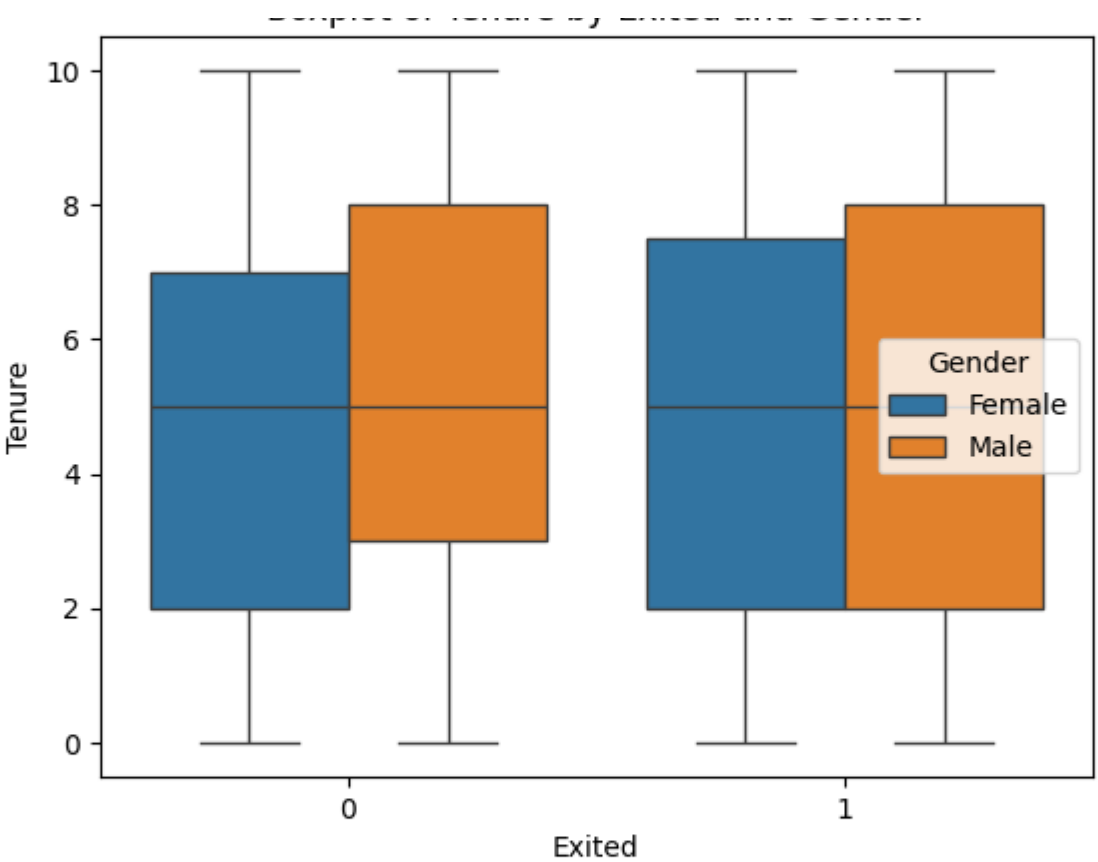

```
for i in subset.columns:
    sns.boxplot(x=df['Exited'], y=df[i], hue=df['Gender'])
    plt.title(f'Boxplot of {i} by Exited and Gender')
```

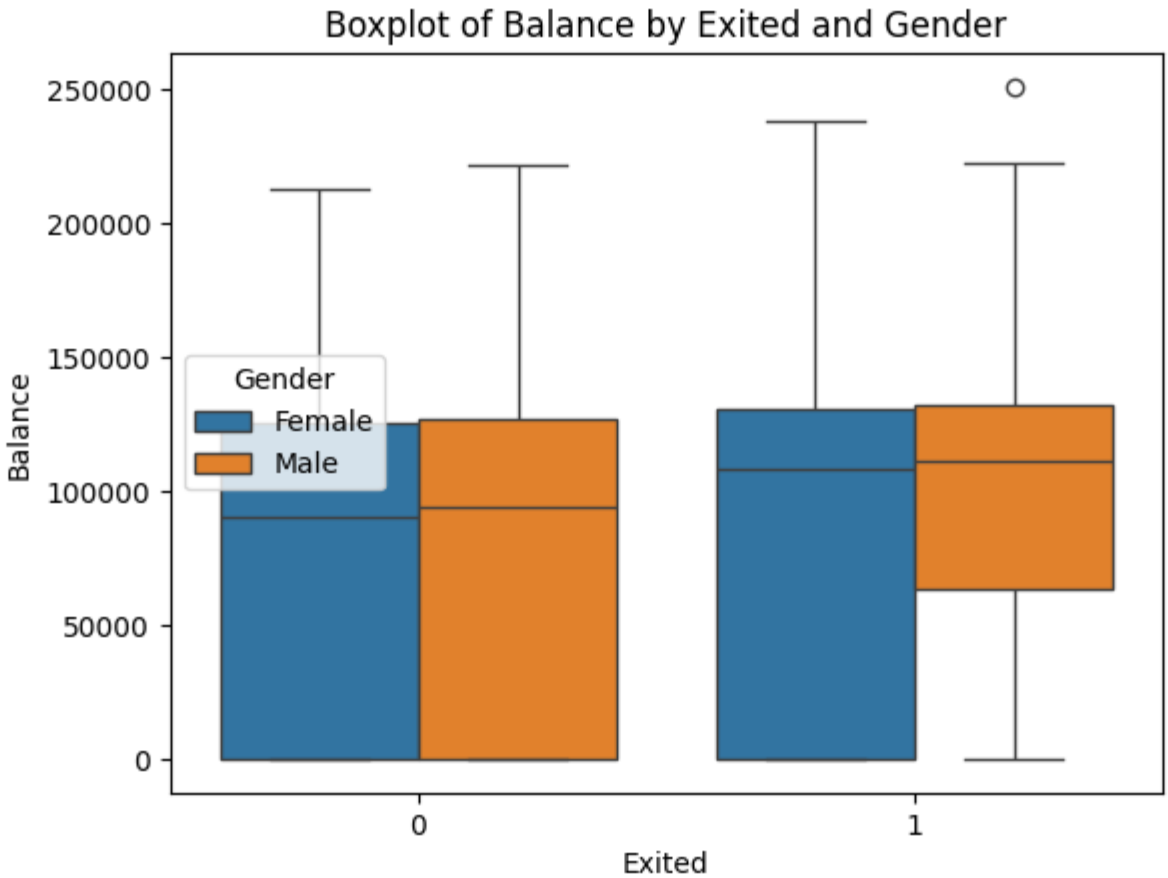
```
    plt.title(f'Boxplot of {i} by Exited and Gender')
    plt.show()
```

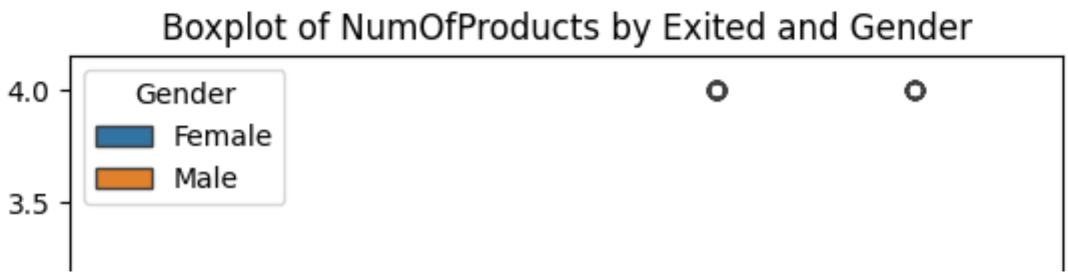


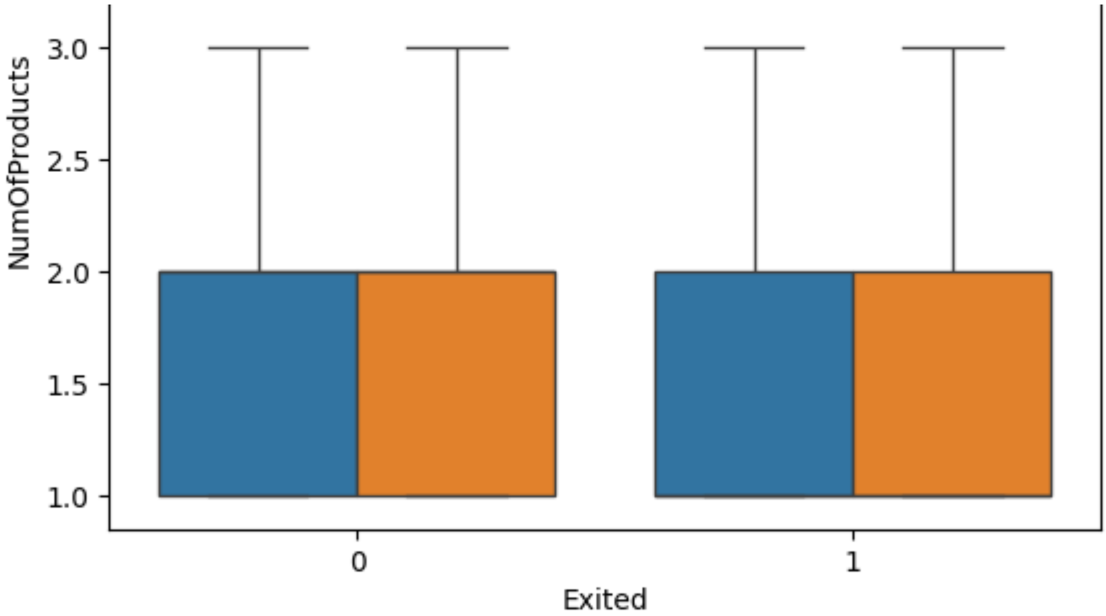


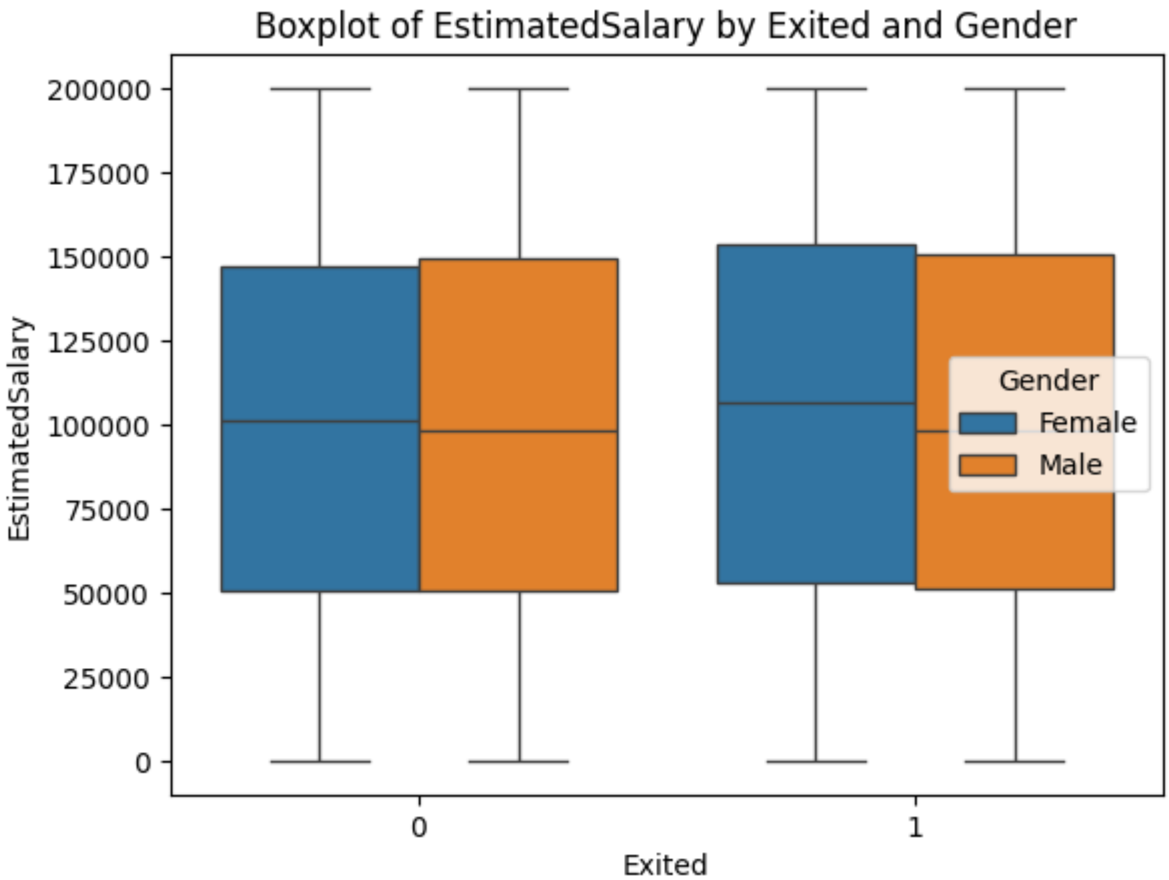
Boxplot of Tenure by Exited and Gender


Tenure
10
8
6
4
2
0
0
1
Exited
Gender
Female
Male


Boxplot of Balance by Exited and Gender
Balance
250000
200000
150000
100000
50000
0
0
1
Exited
Gender
Female
Male


Boxplot of NumOfProducts by Exited and Gender
4.0
3.5
Gender
Female
Male






```
X=df.drop('Exited',axis=1)
y=df.pop('Exited')
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.10,random_state
X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,test_size=0.10,ra
print("X_train size is {}".format(X_train.shape[0]))
print("X_val size is {}".format(X_val.shape[0]))
print("X_test size is {}".format(X_test.shape[0]))
```

```
X_train size is 8100
```

```
X_val size is 900
X_test size is 1000
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
num_cols=['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSal
num_subset=scaler.fit_transform(X_train[num_cols])
X_train_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_train_num_df['Geography']=list(X_train['Geography'])
X_train_num_df['Gender']=list(X_train['Gender'])
X_train_num_df['HasCrCard']=list(X_train['HasCrCard'])
X_train_num_df['IsActiveMember']=list(X_train['IsActiveMember'])
X_train_num_df.head()
num_subset=scaler.fit_transform(X_val[num_cols])
X_val_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_val_num_df['Geography']=list(X_val['Geography'])
X_val_num_df['Gender']=list(X_val['Gender'])
X_val_num_df['HasCrCard']=list(X_val['HasCrCard'])
X_val_num_df['IsActiveMember']=list(X_val['IsActiveMember'])
num_subset=scaler.fit_transform(X_test[num_cols])
X_test_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_test_num_df['Geography']=list(X_test['Geography'])
X_test_num_df['Gender']=list(X_test['Gender'])
X_test_num_df['HasCrCard']=list(X_test['HasCrCard'])
X_test_num_df['IsActiveMember']=list(X_test['IsActiveMember'])
```

```
X_train_num_df=pd.get_dummies(X_train_num_df,columns=['Geography','Gender'])
X_test_num_df=pd.get_dummies(X_test_num_df,columns=['Geography','Gender'])
X_val_num_df=pd.get_dummies(X_val_num_df,columns=['Geography','Gender'])
X_train_num_df.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSala
0	-1.178587	-1.041960	-1.732257	0.198686	0.820905	1.5603
1	-0.380169	-1.326982	1.730718	-0.022020	-0.907991	-0.7135
2	-0.349062	1.808258	-0.693364	0.681178	0.820905	-1.1265
3	0.625629	2.378302	-0.347067	-1.229191	0.820905	-1.6827
4	-0.203895	-1.136967	1.730718	0.924256	-0.907991	1.3325

Next steps: Generate code with X_train_num_df | View recommended plots | New interactive sheet

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model=Sequential()
```

```
model.add(Dense(7,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

```
import tensorflow as tf
optimizer=tf.keras.optimizers.Adam(0.01)
model.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'
```

```
model.fit(X_train_num_df,y_train,epochs=100,batch_size=10,verbose=1)
```

```
Epoch 1/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8094 - loss: 0.4333
Epoch 2/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8595 - loss: 0.3542
Epoch 3/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8634 - loss: 0.3406
Epoch 4/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8575 - loss: 0.3501
Epoch 5/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8562 - loss: 0.3511
Epoch 6/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8608 - loss: 0.3407
Epoch 7/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8603 - loss: 0.3399
Epoch 8/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8606 - loss: 0.3385
Epoch 9/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8590 - loss: 0.3385
Epoch 10/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8661 - loss: 0.3265
Epoch 11/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8639 - loss: 0.3331
Epoch 12/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8619 - loss: 0.3372
Epoch 13/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8683 - loss: 0.3298
Epoch 14/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8675 - loss: 0.3364
Epoch 15/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8654 - loss: 0.3223
Epoch 16/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8682 - loss: 0.3188
Epoch 17/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8646 - loss: 0.3332
Epoch 18/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8651 - loss: 0.3289
Epoch 19/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8636 - loss: 0.3340
Epoch 20/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8698 - loss: 0.3255
Epoch 21/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8751 - loss: 0.3104
Epoch 22/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8629 - loss: 0.3377
Epoch 23/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8677 - loss: 0.3269
```

```
810/810 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8677 - loss: 0.3269
Epoch 24/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8684 - loss: 0.3285
Epoch 25/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8667 - loss: 0.3321
Epoch 26/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8658 - loss: 0.3301
Epoch 27/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8589 - loss: 0.3379
Epoch 28/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8667 - loss: 0.3233
Epoch 29/100
810/810 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8646 - loss: 0.3410
```

```
y_pred_val=model.predict(X_val_num_df)
y_pred_val[y_pred_val>0.5]=1
y_pred_val[y_pred_val <0.5]=0
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

```
y_pred_val=y_pred_val.tolist()
X_compare_val=X_val.copy()
X_compare_val['y_actual']=y_val
X_compare_val['y_pred']=y_pred_val
X_compare_val.head(10)
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts
340	642	Germany	Female	40	6	129502.49	2
8622	706	Germany	Male	36	9	58571.18	2
8401	535	Spain	Male	58	1	0.00	2
4338	714	Spain	Male	25	2	0.00	1
8915	606	France	Male	36	1	155655.46	1
2624	605	Spain	Female	29	3	116805.82	1
2234	720	France	Female	38	10	0.00	2
349	582	France	Male	39	5	0.00	2
3719	850	France	Female	62	1	124678.35	1
2171	526	Germany	Male	58	9	190298.89	2

Next steps: Generate code with X_compare_val | View recommended plots | New interactive sheet

```
from sklearn.metrics import confusion_matrix
cm_val=confusion_matrix(y_val,y_pred_val)
cm_val
```

```
array([[672,  44],
       [ 89,  95]])
```

```
Accuracy=782/900
print("Accuracy of the Model on the Validation Data set is 86.89%")
```

```
Accuracy of the Model on the Validation Data set is 86.89%
```

```
loss1,accuracy1=model.evaluate(X_train_num_df,y_train,verbose=False)
loss2,accuracy2=model.evaluate(X_val_num_df,y_val,verbose=False)
print("Train Loss {}".format(loss1))
print("Train Accuracy {}".format(accuracy1))
print("Val Loss {}".format(loss2))
print("Val Accuracy {}".format(accuracy2))
```

```
Train Loss 0.3174085319042206
Train Accuracy 0.87086421251297
Val Loss 0.35060858726501465
Val Accuracy 0.852222204208374
```

```
from sklearn import metrics
y_pred_test=model.predict(X_test_num_df)
y_pred_test[y_pred_test>0.5]=1
y_pred_test[y_pred_test <0.5]=0
cm_test=metrics.confusion_matrix(y_test,y_pred_test)
cm_test
print("Test Confusion Matrix")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Test Confusion Matrix
```

```
cm_test
```

```
array([[742,  52],
       [114,  92]])
```

```
loss3,accuracy3=model.evaluate(X_test_num_df,y_test,verbose=False)
print("Test Accuracy is {}".format(accuracy3))
print("Test loss is {}".format(loss3))
```

```
Test Accuracy is 0.8339999914169312
Test loss is 0.3814462721347809
```

Start coding or generate with AI.