BankChurn Data Project / Yeoman Yoon

In [1]:
```
#!pip install tensorflow
```

In [2]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,
recall_score, f1_score, precision_recall_curve, auc
from sklearn.decomposition import PCA

#model
from sklearn.model_selection import train_test_split
#CV
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_classification

#Tensorflow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import optimizers

#Keras
import keras
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
from keras import layers
from keras.wrappers.scikit_learn import KerasClassifier
```

```
Using TensorFlow backend.
```

Import Data

In [3]:
```
bank = pd.read_csv("bank.csv") # use copy to store the original data
data = bank.copy()
```

In [4]:
```
data.head()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance
0	1	15634602	Hargrave	619	France	Female	42	2	0.00
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86
2	3	15619304	Onio	502	France	Female	42	8	159660.80
3	4	15701354	Boni	699	France	Female	39	1	0.00
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82

In [5]:
```
print(f"The data has {data.shape[0]} rows and {data.shape[1]} columns")
```

```
The data has 10000 rows and 14 columns
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
RowNumber          10000 non-null int64
CustomerId         10000 non-null int64
Surname            10000 non-null object
CreditScore        10000 non-null int64
Geography          10000 non-null object
Gender             10000 non-null object
Age                10000 non-null int64
Tenure             10000 non-null int64
Balance            10000 non-null float64
NumOfProducts      10000 non-null int64
HasCrCard          10000 non-null int64
IsActiveMember     10000 non-null int64
EstimatedSalary    10000 non-null float64
Exited             10000 non-null int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

Data does not have any missing values

Data Dictionary

Data Dictionary:

1. CustomerId: Unique ID which is assigned to each customer
2. Surname: Last name of the customer
3. CreditScore: It defines the credit history of the customer.
4. Geography: A customer's location
5. Gender: It defines the Gender of the customer
6. Age: Age of the customer
7. Tenure: Number of years for which the customer has been with the bank
8. NumOfProducts: It refers to the number of products that a customer has purchased through the bank.
9. Balance: Account balance
10. HasCrCard: It is a categorical variable that decides whether the customer has a credit card or not.
11. EstimatedSalary: Estimated salary
12. isActiveMember: It is a categorical variable that decides whether the customer is an active member of the bank or not (Active member in the sense, using bank products regularly, making transactions, etc)
13. Excited: It is a categorical variable that decides whether the customer left the bank within six months or not. It can take two values (Predicted value)

In [7]:
```
# Drop customer unique info, these are meaningless data in machine learning
data.drop(['RowNumber','CustomerId',"Surname"], axis=1, inplace = True)
```

In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
CreditScore        10000 non-null int64
Geography          10000 non-null object
Gender             10000 non-null object
Age                10000 non-null int64
Tenure             10000 non-null int64
Balance            10000 non-null float64
NumOfProducts      10000 non-null int64
HasCrCard          10000 non-null int64
IsActiveMember     10000 non-null int64
EstimatedSalary    10000 non-null float64
Exited             10000 non-null int64
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

In [9]:
```
data.nunique().sort_values(ascending=False)
```

Out[9]:
```
EstimatedSalary    9999
Balance            6382
CreditScore         460
Age                  70
Tenure               11
NumOfProducts         4
Geography             3
Exited                2
IsActiveMember        2
HasCrCard             2
Gender                2
dtype: int64
```

Data Binning may be needed for Salary, Balance, CreditScore, and Age

Datatype Conversion

For the memory purpose, Convert object and unique values to category.
Skip this line to see correlation scores with integer (data.describe())

In [10]:
```
to_category = ['Geography', 'Gender']

for col in to_category:
    data[col]=data[col].astype('category')
```

In [11]:
```
# to_category1 = ['NumOfProducts', 'HasCrCard', 'IsActiveMember', 'Exited']
# for col in to_category1:
#     data[col]=data[col].astype('category')
```

EDA

Univariate EDA:

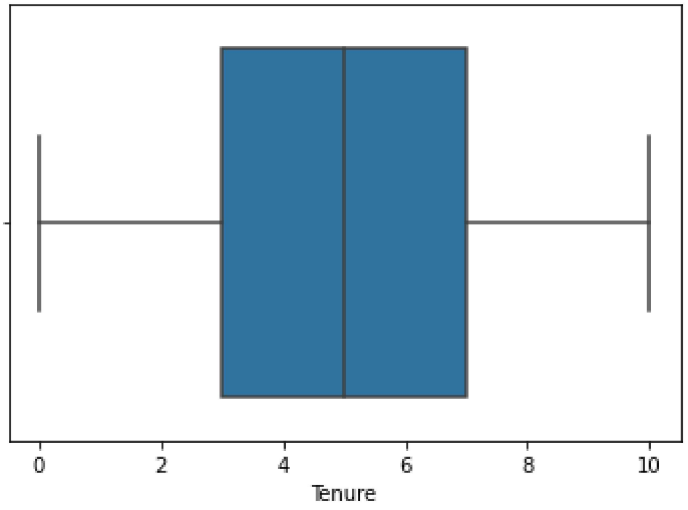
In [12]:
```
data.describe().T
```

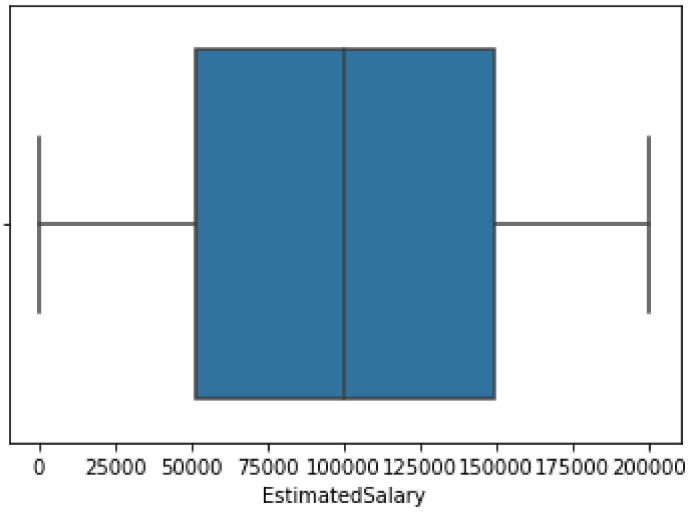
Out[12]:

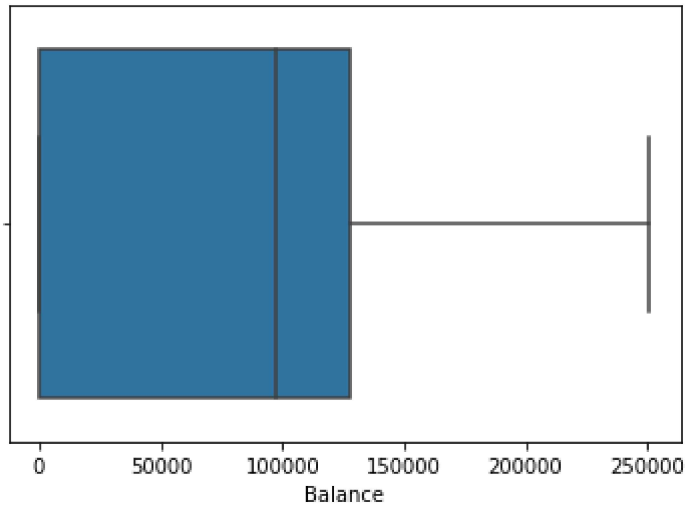
	count	mean	std	min	25%	50%	75%
CreditScore	10000.0	650.528800	96.653299	350.00	584.00	652.000	718.0000
Age	10000.0	38.921800	10.487806	18.00	32.00	37.000	44.0000
Tenure	10000.0	5.012800	2.892174	0.00	3.00	5.000	7.0000
Balance	10000.0	76485.889288	62397.405202	0.00	0.00	97198.540	127644.2400
NumOfProducts	10000.0	1.530200	0.581654	1.00	1.00	1.000	2.0000
HasCrCard	10000.0	0.705500	0.455840	0.00	0.00	1.000	1.0000
IsActiveMember	10000.0	0.515100	0.499797	0.00	0.00	1.000	1.0000
EstimatedSalary	10000.0	100090.239881	57510.492818	11.58	51002.11	100193.915	149388.2475
Exited	10000.0	0.203700	0.402769	0.00	0.00	0.000	0.0000

In [13]:
```
boxplot_col = ['Tenure','EstimatedSalary','Balance']
distplot_col = ['Age','CreditScore']
cat_col = ['NumOfProducts','Geography','Exited','IsActiveMember','HasCrCar
d','Gender']
```

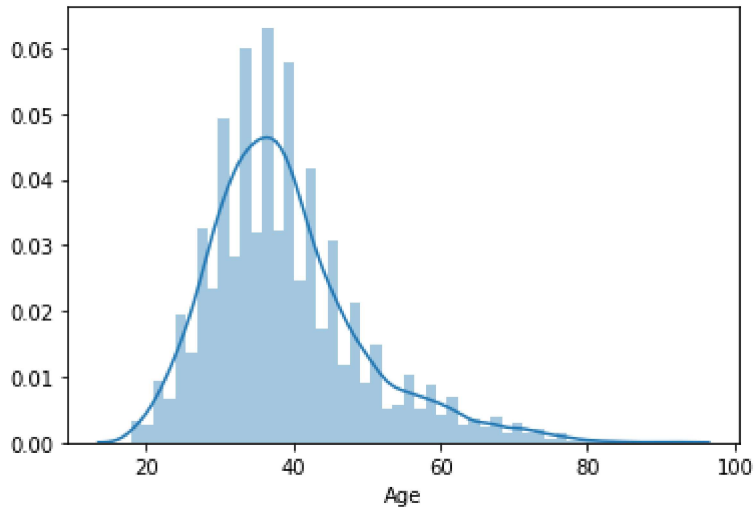
In [14]:
```
for i in boxplot_col:
    sns.boxplot(data[i])
    plt.show()
```

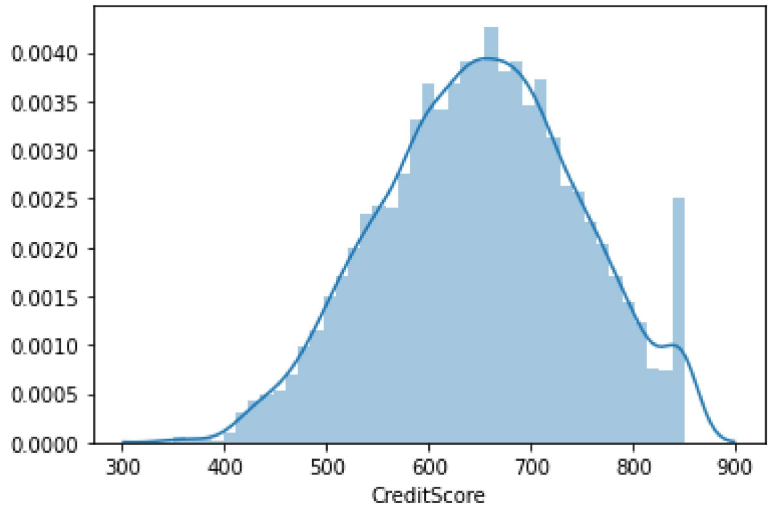






In [15]:
```
for i in distplot_col:
    sns.distplot(data[i])
    plt.show()
```





In [16]:
```
data[data['Age']>85] # quick Outlier treatment. No missing value treatment.
```

Out[16]:

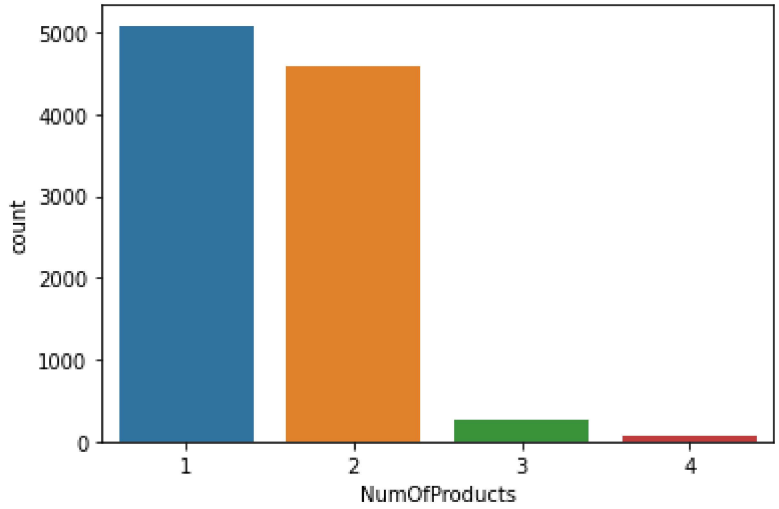
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsA
2458	513	Spain	Male	88	10	0.00	2	1	
6443	753	France	Male	92	3	121513.31	1	0	
6759	705	France	Male	92	1	126076.24	2	1	

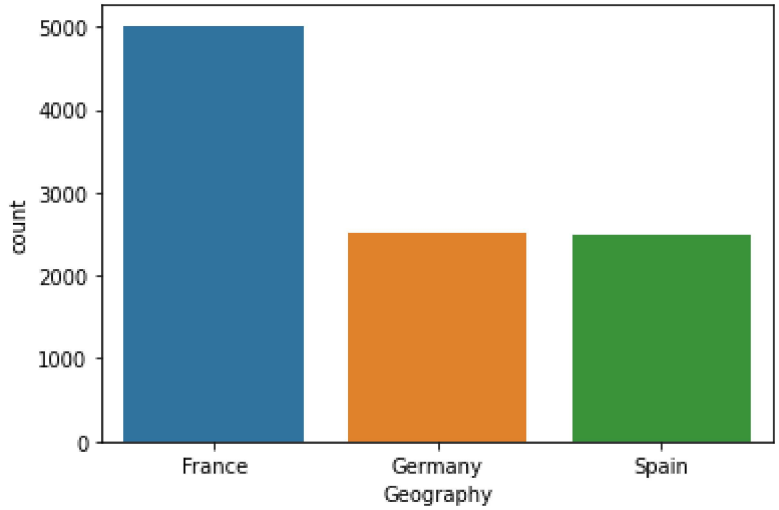
In [17]:
```
data.drop([2458,6443,6759], axis=0, inplace = True)
```

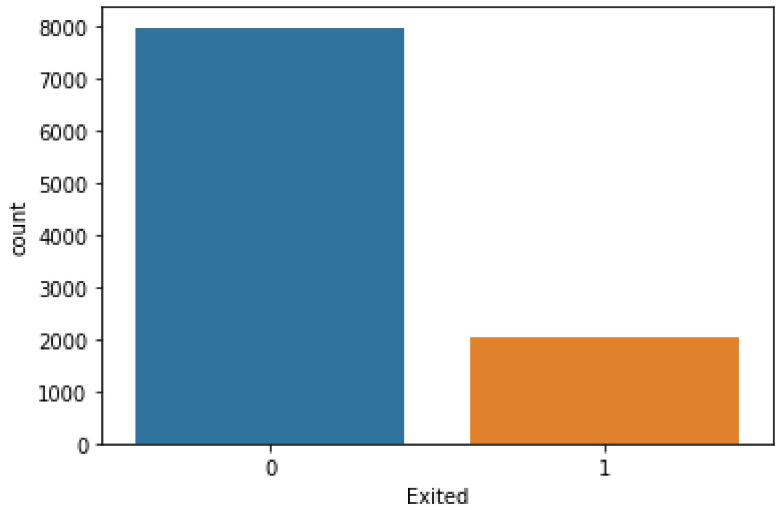
In [18]:
```
# data[data['CreditScore']>850]
data[data['CreditScore']==850]['Exited'].value_counts()
```

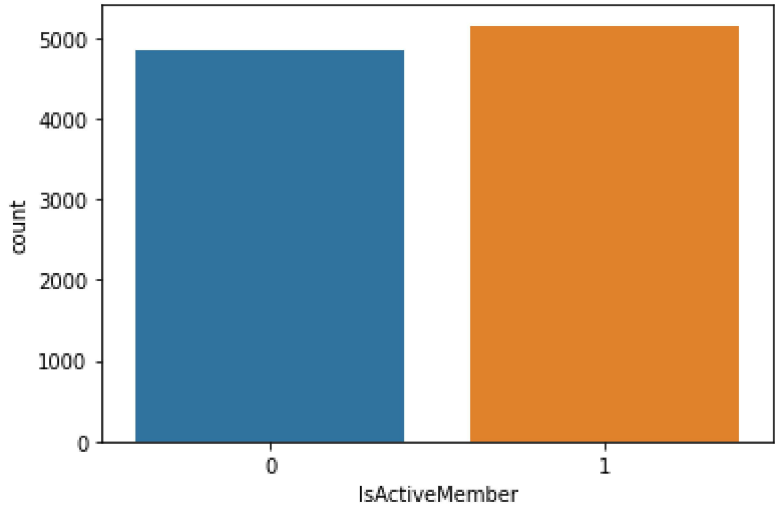
Out[18]:
```
0    190
1     43
Name: Exited, dtype: int64
```

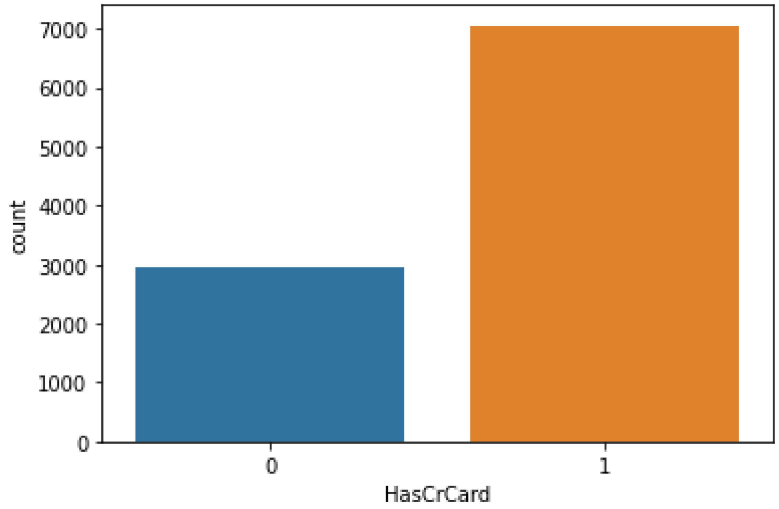
In [19]:
```
for i in cat_col:
    sns.countplot(data[i])
    plt.show()
```

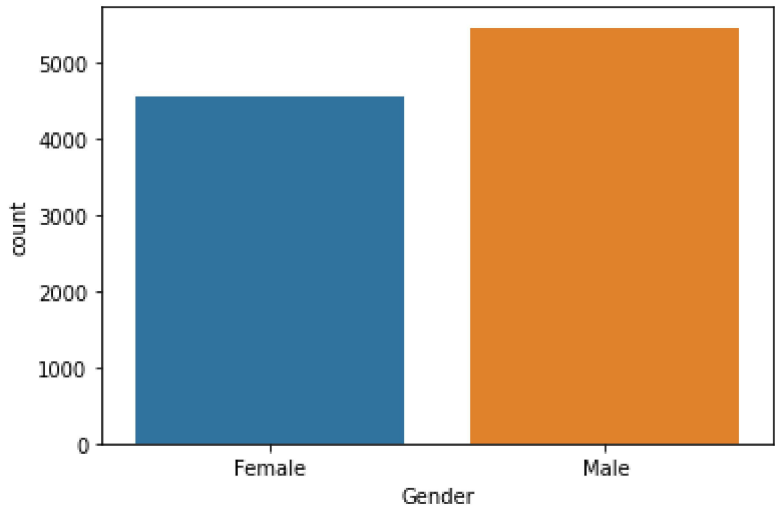

5000
4000
3000
2000
1000
0
count
1
2
3
4
NumOfProducts


5000
4000
3000
2000
1000
0
count
France
Germany
Spain
Geography


8000
7000
6000
5000
4000
3000
2000
1000
0
count
0
1
Exited


5000
4000
3000
2000
1000
0
count
0
1
IsActiveMember


7000
6000
5000
4000
3000
2000
1000
0
count
0
1
HasCrCard


5000
4000
3000
2000
1000
0
count
Female
Male
Gender

Overall data is balanced. Predicted value is unbalanced and couple columns are unbalanced.

1. More French customers.
2. More customers have credit card.
3. Number of Products are mostly 1 or 2.

Action: Consider Age > 85 as more extreme Outliers.
Since 850 is the highest credit score you can get, leave them as it is.

Bivariate EDA:

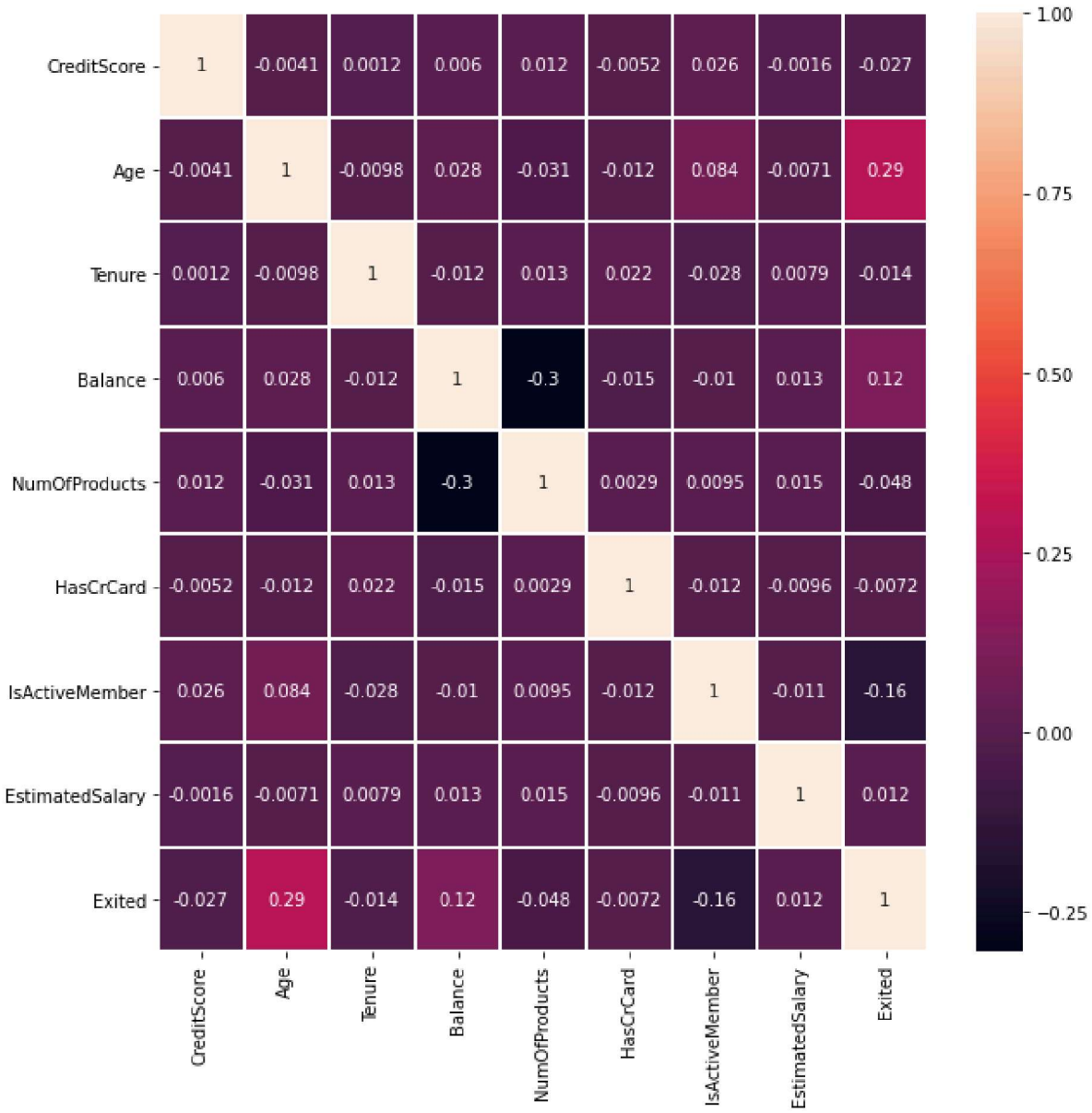
In [20]:
```
data.corr()
```

Out[20]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActive
CreditScore	1.000000	-0.004137	0.001240	0.005974	0.012407	-0.005241	
Age	-0.004137	1.000000	-0.009788	0.028222	-0.031123	-0.011611	
Tenure	0.001240	-0.009788	1.000000	-0.011887	0.013358	0.022463	-
Balance	0.005974	0.028222	-0.011887	1.000000	-0.304152	-0.014723	-
NumOfProducts	0.012407	-0.031123	0.013358	-0.304152	1.000000	0.002938	
HasCrCard	-0.005241	-0.011611	0.022463	-0.014723	0.002938	1.000000	-
IsActiveMember	0.025640	0.084357	-0.028339	-0.010115	0.009546	-0.011844	
EstimatedSalary	-0.001613	-0.007108	0.007886	0.012673	0.014519	-0.009554	-
Exited	-0.027089	0.287135	-0.014023	0.118568	-0.047791	-0.007152	-

In [21]:
```
plt.subplots(figsize=(10,10))
sns.heatmap(data.corr(), annot =True, linewidth=1)
```

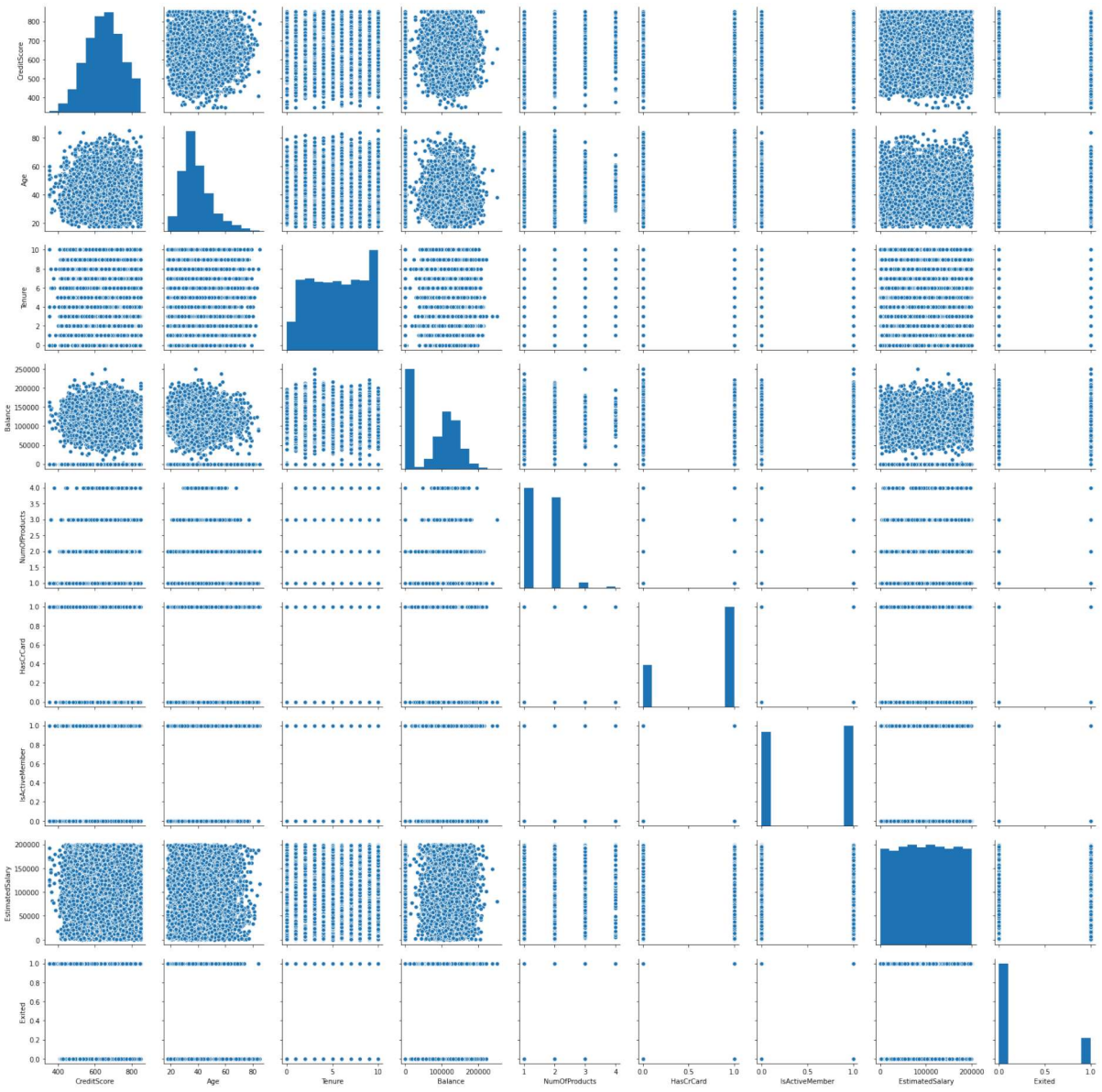
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1c6206bc548>



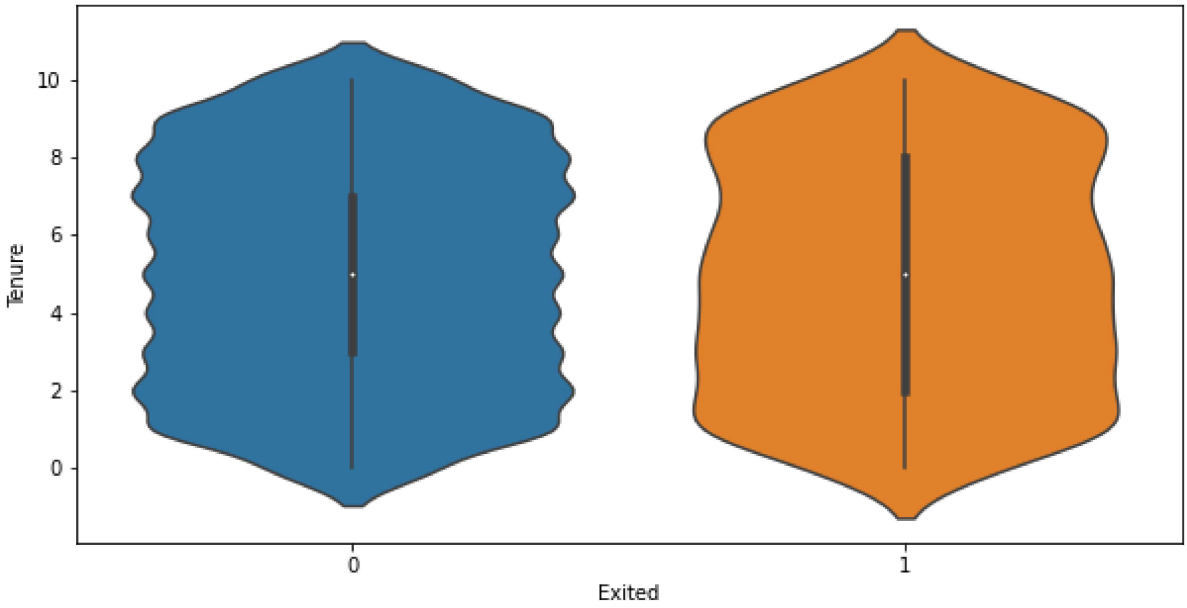
There is no Linear Correlation between variables. No correlation in variables is good when approached with machine learning.

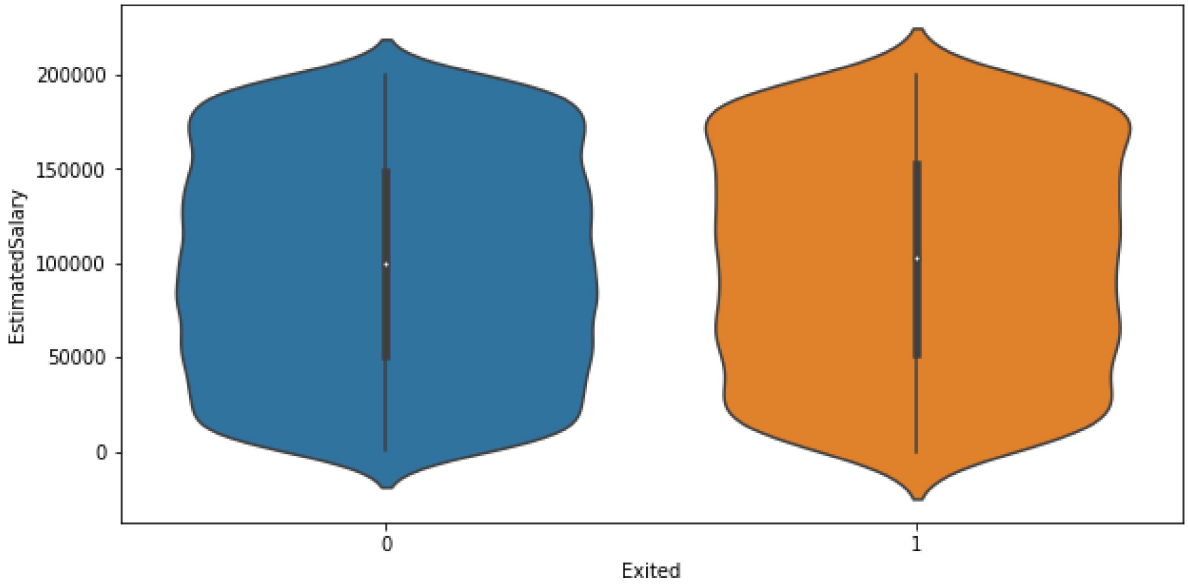
In [22]:
```
sns.pairplot(data)
```

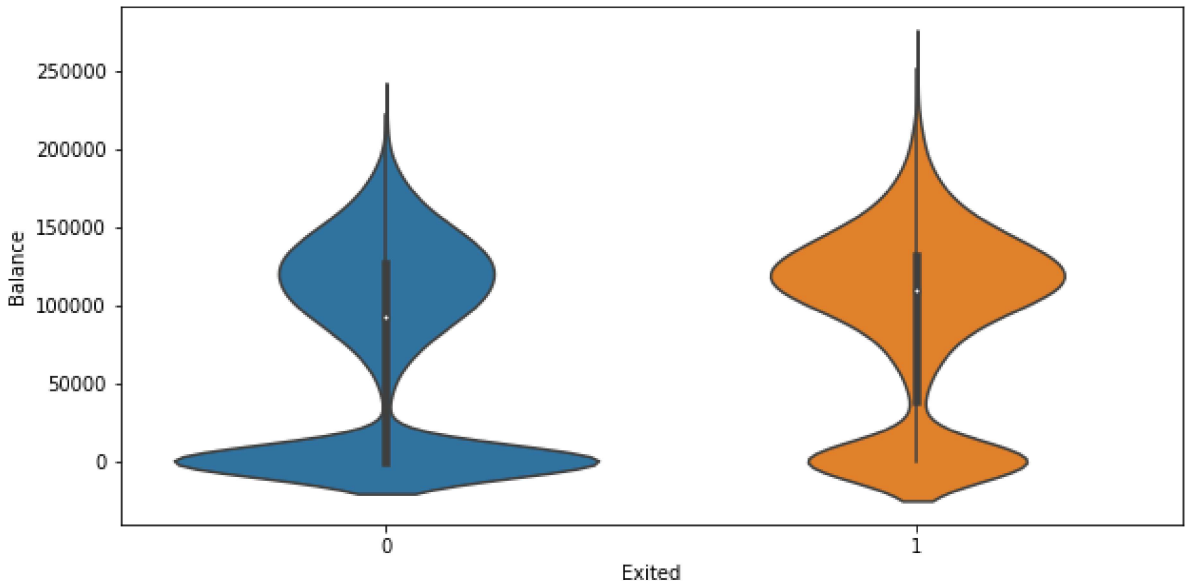
Out[22]: <seaborn.axisgrid.PairGrid at 0x1c620eb8608>



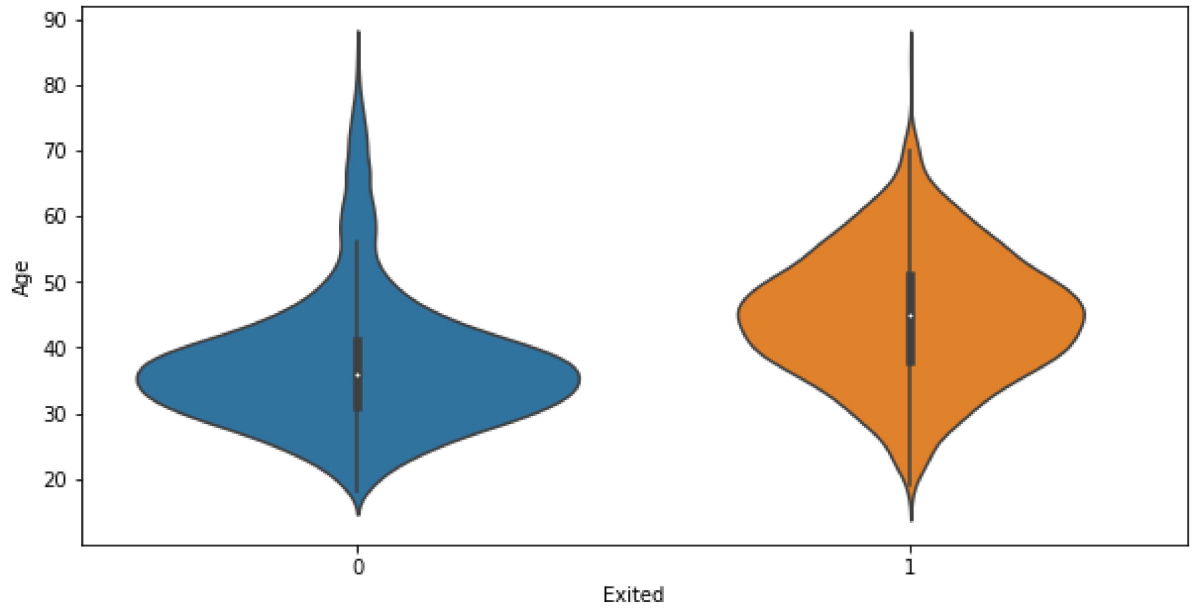
In [23]:
```
for i in boxplot_col:
    plt.figure(figsize=(10,5))
    sns.violinplot(x=data['Exited'], y=data[i])
    plt.show()
```

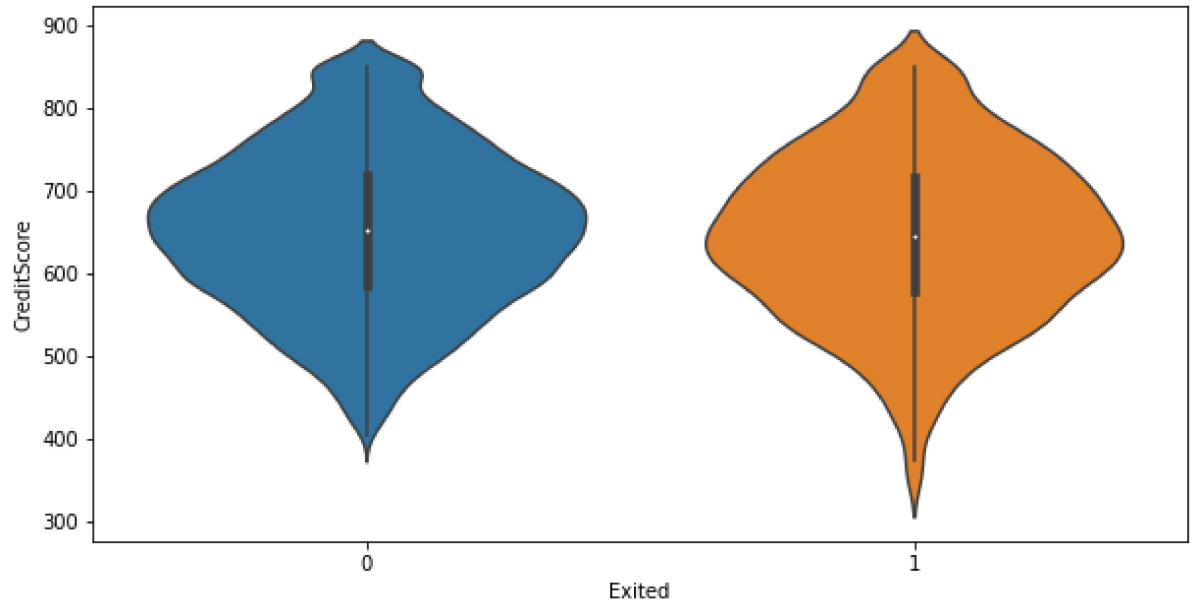

Tenure
10
8
6
4
2
0
0
1
Exited


EstimatedSalary
200000
150000
100000
50000
0
0
1
Exited

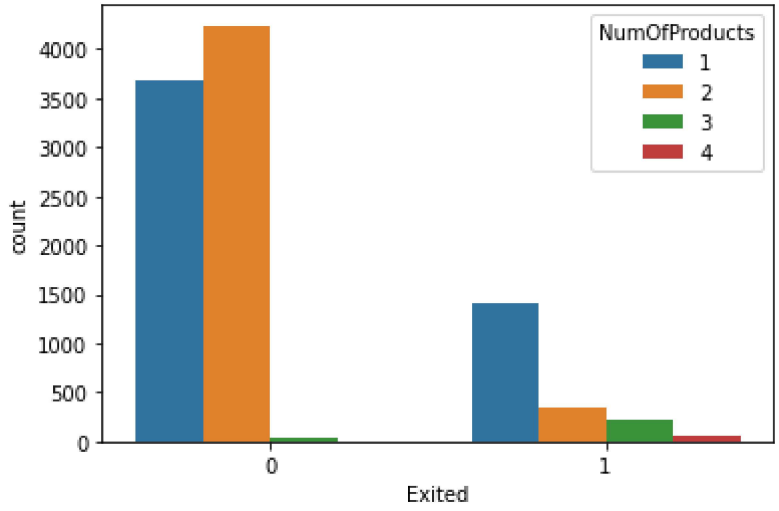

Balance
250000
200000
150000
100000
50000
0
0
1
Exited

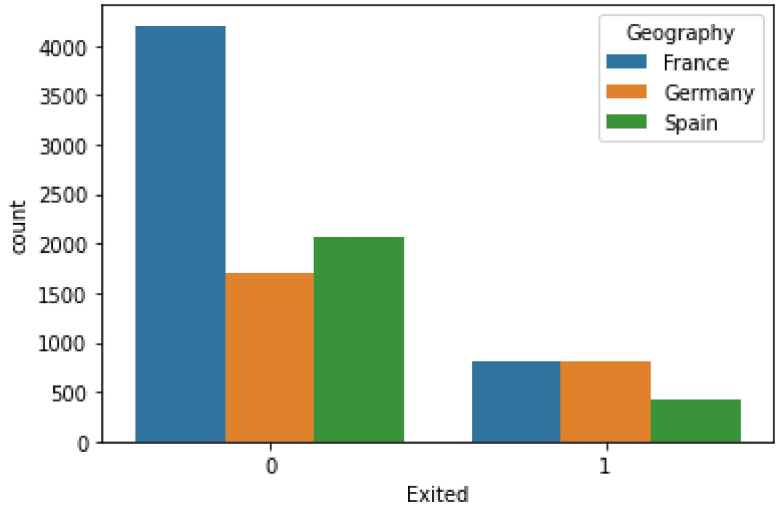
In [24]:
```
for i in distplot_col:
    plt.figure(figsize=(10,5))
    sns.violinplot(x=data['Exited'], y=data[i])
    plt.show()
```

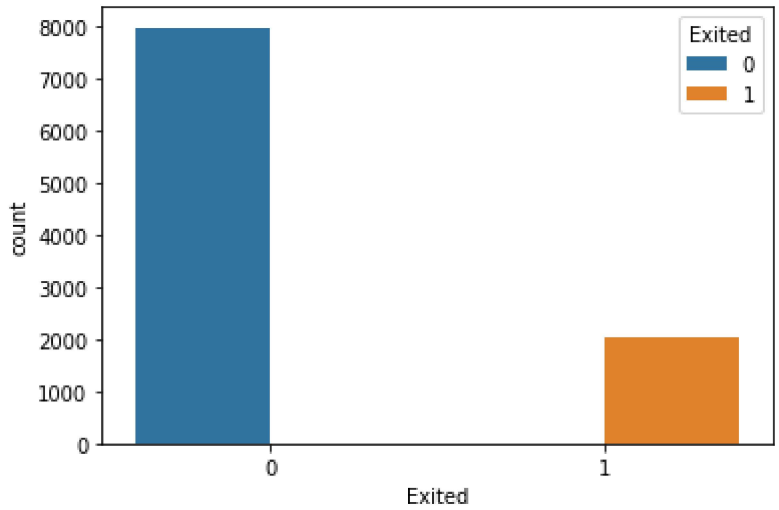


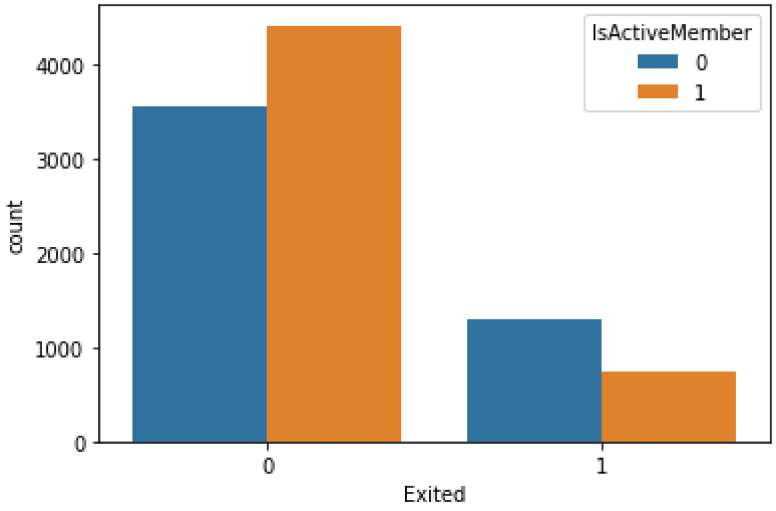


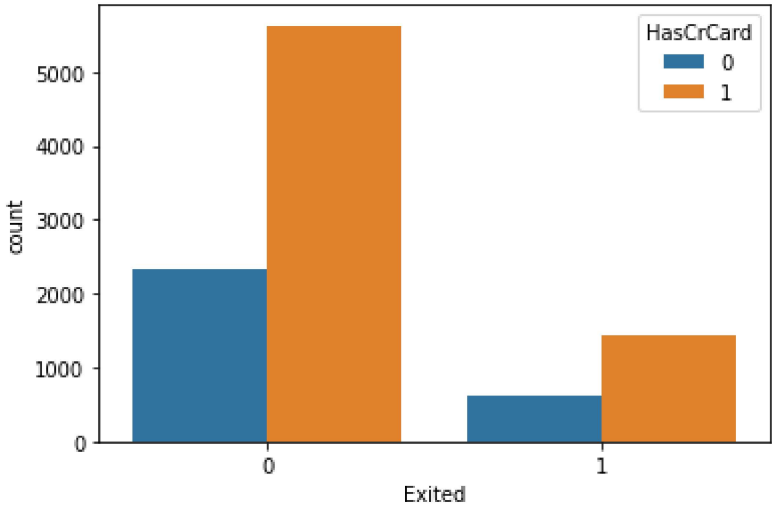
In [25]:
```
for i in cat_col:
    sns.countplot(data['Exited'], hue = data[i])
    plt.show()
```

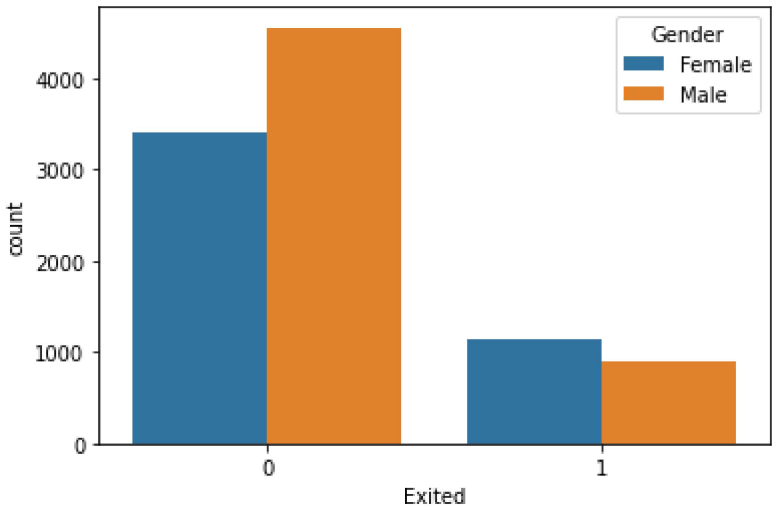

NumOfProducts
1
2
3
4
count
Exited
0
1
0
500
1000
1500
2000
2500
3000
3500
4000


Geography
France
Germany
Spain
count
Exited
0
1
0
500
1000
1500
2000
2500
3000
3500
4000


Exited
0
1
count
Exited
0
1
0
1000
2000
3000
4000
5000
6000
7000
8000


IsActiveMember
0
1
count
4000
3000
2000
1000
0
0
1
Exited


HasCrCard
0
1
count
5000
4000
3000
2000
1000
0
0
1
Exited


Gender
Female
Male
count
4000
3000
2000
1000
0
0
1
Exited

Bivariate Insights:

1. Females are more likely to leave than males
2. Non-active members are more likely to leave.
3. German are most likely to leave compared to French and Spanish
4. Customers with higher balnace are slightly more likely to leave.
5. Customers start to leave around age of 35. 40 and above leave a lot more

Data Pre-Processing

Dummy variables:

In [26]:
```
data = pd.get_dummies(data, columns=['Geography','Gender'],drop_first=True)
```

Data Rescale:

In [27]:
```
all_col = ['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCar
d','IsActiveMember','EstimatedSalary',
           'Geography_Germany','Geography_Spain','Gender_Male']
scaler = StandardScaler()
copy_data = data[all_col].copy() # we might need to use data again later, so d
on't change it
copy_data_scaled = scaler.fit_transform(copy_data)
scaled_data = pd.DataFrame(copy_data_scaled, columns=copy_data.columns)
scaled_data.head()
```

Out[27]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Esti
0	-0.326209	0.296039	-1.041920	-1.225797	-0.911474	0.646074	0.970525	
1	-0.440026	0.200349	-1.387739	0.117377	-0.911474	-1.547811	0.970525	
2	-1.536810	0.296039	1.032995	1.333059	2.527018	0.646074	-1.030370	
3	0.501552	0.008969	-1.387739	-1.225797	0.807772	-1.547811	-1.030370	
4	2.063952	0.391729	-1.041920	0.785743	-0.911474	0.646074	0.970525	

Data Split:

In [28]:
```
X = scaled_data
Y = data[['Exited']]

#Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.30, rando
m_state = 1)
```

In [29]:
```
X.head()
```

Out[29]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Esti
0	-0.326209	0.296039	-1.041920	-1.225797	-0.911474	0.646074	0.970525	
1	-0.440026	0.200349	-1.387739	0.117377	-0.911474	-1.547811	0.970525	
2	-1.536810	0.296039	1.032995	1.333059	2.527018	0.646074	-1.030370	
3	0.501552	0.008969	-1.387739	-1.225797	0.807772	-1.547811	-1.030370	
4	2.063952	0.391729	-1.041920	0.785743	-0.911474	0.646074	0.970525	

Neural Network

Model 1:

General Model with easy 5 layers.

In [30]:
```
model = Sequential()
```

Adding layers:

In [31]:
```
model.add(Dense(units=25, input_dim = 11,activation='relu'))   # input of 11 c
olumns as shown above
# hidden layer
model.add(Dense(units=25,activation='relu'))
model.add(Dropout(0.5)) #dropout for reducing noise
model.add(Dense(units=25,activation='relu'))
model.add(Dense(units=25,activation='relu'))

# Output of 1 node, (Exited or not) 0 or 1
# Sigmoid because we want probability outcomes
# Adding the output layer
model.add(Dense(1,activation='sigmoid'))# binary classification Exited or not
```

In [32]:
```
# Create optimizer with default learning rate (0.01)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accurac
y'])
```

In [33]:
```
model.summary() # quick visualization on how NN will be like
```

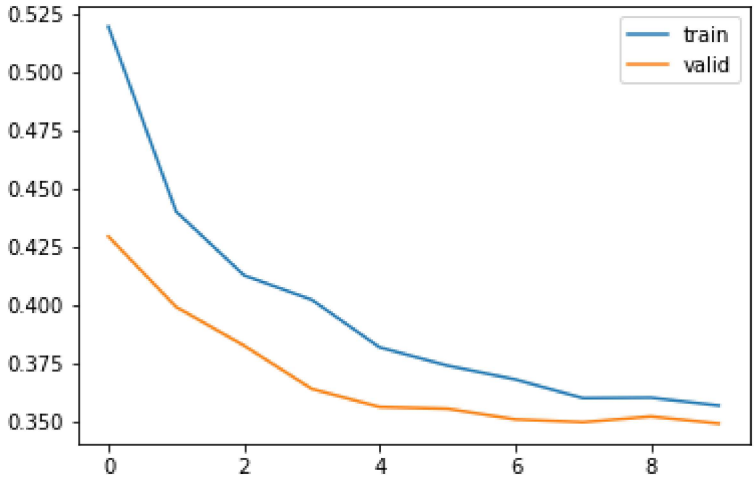
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 25)                300
_________________________________________________________________
dense_2 (Dense)              (None, 25)                650
_________________________________________________________________
dropout_1 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_3 (Dense)              (None, 25)                650
_________________________________________________________________
dense_4 (Dense)              (None, 25)                650
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 26
=================================================================
Total params: 2,276
Trainable params: 2,276
Non-trainable params: 0
_________________________________________________________________
```

In [34]:
```
#fit the model
history=model.fit(X_train,y_train,batch_size=15,epochs=10,validation_split=0.
2)
```

```
Train on 5597 samples, validate on 1400 samples
Epoch 1/10
5597/5597 [==============================] - 0s 83us/step - loss: 0.5195 - ac
curacy: 0.7604 - val_loss: 0.4295 - val_accuracy: 0.8071
Epoch 2/10
5597/5597 [==============================] - 0s 60us/step - loss: 0.4402 - ac
curacy: 0.8042 - val_loss: 0.3992 - val_accuracy: 0.8379
Epoch 3/10
5597/5597 [==============================] - 0s 58us/step - loss: 0.4128 - ac
curacy: 0.8233 - val_loss: 0.3827 - val_accuracy: 0.8436
Epoch 4/10
5597/5597 [==============================] - 0s 65us/step - loss: 0.4023 - ac
curacy: 0.8279 - val_loss: 0.3642 - val_accuracy: 0.8471
Epoch 5/10
5597/5597 [==============================] - 0s 57us/step - loss: 0.3820 - ac
curacy: 0.8378 - val_loss: 0.3564 - val_accuracy: 0.8600
Epoch 6/10
5597/5597 [==============================] - 0s 58us/step - loss: 0.3741 - ac
curacy: 0.8496 - val_loss: 0.3556 - val_accuracy: 0.8571
Epoch 7/10
5597/5597 [==============================] - 0s 56us/step - loss: 0.3682 - ac
curacy: 0.8465 - val_loss: 0.3511 - val_accuracy: 0.8607
Epoch 8/10
5597/5597 [==============================] - 0s 57us/step - loss: 0.3602 - ac
curacy: 0.8508 - val_loss: 0.3499 - val_accuracy: 0.8600
Epoch 9/10
5597/5597 [==============================] - 0s 67us/step - loss: 0.3604 - ac
curacy: 0.8531 - val_loss: 0.3523 - val_accuracy: 0.8579
Epoch 10/10
5597/5597 [==============================] - 0s 57us/step - loss: 0.3570 - ac
curacy: 0.8540 - val_loss: 0.3493 - val_accuracy: 0.8579
```

In [35]:
```
# Learning history per epoch using graph
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

# Accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)
```

Out[35]: <matplotlib.legend.Legend at 0x1c6303c8d08>



In [36]:
```
score = model.evaluate(X_test, y_test)
```

```
3000/3000 [==============================] - 0s 11us/step
```

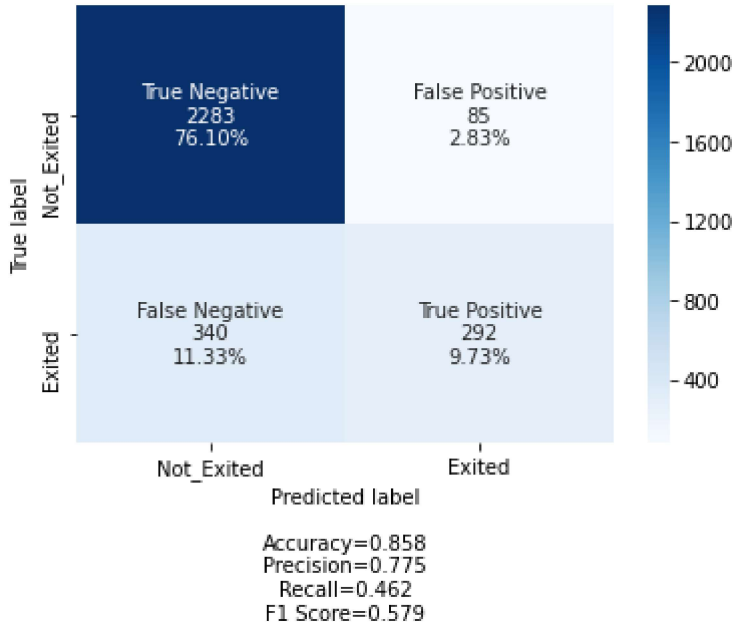
In [37]:
```
print(score)
```

```
[0.354510751247406, 0.8583333492279053]
```

In [38]:
```
# Used the function cm pretty plot of an sklearn Confusion Matrix
def make_cm(cf,
                            group_names=None,
                            categories='auto',
                            count=True,
                            percent=True,
                            cbar=True,
                            xyticks=True,
                            xyplotlabels=True,
                            sum_stats=True,
                            figsize=None,
                            cmap='Blues',
                            title=None):
    '''
    This function will make a pretty plot of an sklearn Confusion Matrix cm us
ing a Seaborn heatmap visualization.
    Arguments
    '''


    # CODE TO GENERATE TEXT INSIDE EACH SQUARE
    blanks = ['' for i in range(cf.size)]

    if group_names and len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
    else:
        group_labels = blanks

    if count:
        group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
    else:
        group_counts = blanks

    if percent:
        group_percentages = ["{0:.2%}".format(value) for value in cf.flatten
()/np.sum(cf)]
    else:
        group_percentages = blanks

    box_labels = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in zip(group_labels,g
roup_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])


    # CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
    if sum_stats:
        #Accuracy is sum of diagonal divided by total observations
        accuracy  = np.trace(cf) / float(np.sum(cf))

        #if it is a binary confusion matrix, show some more stats
        if len(cf)==2:
            #Metrics for Binary Confusion Matrices
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.
```

```
3f}\nF1 Score={:0.3f}".format(
                accuracy,precision,recall,f1_score)
        else:
            stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    else:
        stats_text = ""


    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)
    sns.heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=cat
egories,yticklabels=categories)

    if xyplotlabels:
        plt.ylabel('True label')
        plt.xlabel('Predicted label' + stats_text)
    else:
        plt.xlabel(stats_text)

    if title:
        plt.title(title)
```

In [39]:
```
## Confusion Matrix on unsee test set
y_pred1 = model.predict(X_test)
for i in range(len(y_test)):
    if y_pred1[i]>0.5:
        y_pred1[i]=1
    else:
        y_pred1[i]=0



cm2=confusion_matrix(y_test, y_pred1)
labels = ['True Negative','False Positive','False Negative','True Positive']
categories = [ 'Not_Exited','Exited']
make_cm(cm2,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



Model Improvement:

Model 2:

Little bit of parameter change. Increase in Neurons and iterations

In [40]:
```
#Training Multi-layer perceptron with 2 hidden layers

#adding earlystopping callback
es= keras.callbacks.EarlyStopping(monitor='val_loss',
                                  min_delta=0,
                                  patience=15,

                                 verbose=0, mode='min', restore_best_weights= True)

model2 = Sequential()
#Initializing the weights uisng hue_normal

model2.add(Dense(65, input_shape=(11, ), kernel_initializer='he_normal', activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(65, kernel_initializer='he_normal', activation='relu'))
model2.add(Dropout(0.5))

model2.add(Dense(1, kernel_initializer='he_normal', activation='sigmoid')) # output

model2.compile(Adam(lr=0.001), loss='binary_crossentropy', metrics=['accuracy']) # learning rate of 0.001

his_mod2= model2.fit(X_train, y_train, validation_split=0.2, batch_size=1000, epochs=60,
                     callbacks=[es], shuffle=True, verbose=1)
```

```
Train on 5597 samples, validate on 1400 samples
Epoch 1/60
5597/5597 [==============================] - 0s 27us/step - loss: 1.1824 - ac
curacy: 0.5062 - val_loss: 0.6229 - val_accuracy: 0.6707
Epoch 2/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.9057 - acc
uracy: 0.6252 - val_loss: 0.5653 - val_accuracy: 0.7993
Epoch 3/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.8236 - acc
uracy: 0.7002 - val_loss: 0.5765 - val_accuracy: 0.8071
Epoch 4/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.7862 - acc
uracy: 0.7341 - val_loss: 0.5521 - val_accuracy: 0.8071
Epoch 5/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.7385 - acc
uracy: 0.7409 - val_loss: 0.5087 - val_accuracy: 0.8064
Epoch 6/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.7002 - acc
uracy: 0.7413 - val_loss: 0.4731 - val_accuracy: 0.8086
Epoch 7/60
5597/5597 [==============================] - 0s 6us/step - loss: 0.6651 - acc
uracy: 0.7343 - val_loss: 0.4536 - val_accuracy: 0.8093
Epoch 8/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.6532 - acc
uracy: 0.7291 - val_loss: 0.4439 - val_accuracy: 0.8129
Epoch 9/60
5597/5597 [==============================] - 0s 5us/step - loss: 0.6414 - acc
uracy: 0.7363 - val_loss: 0.4386 - val_accuracy: 0.8150
Epoch 10/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.6120 - acc
uracy: 0.7404 - val_loss: 0.4355 - val_accuracy: 0.8171
Epoch 11/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.6191 - acc
uracy: 0.7445 - val_loss: 0.4328 - val_accuracy: 0.8164
Epoch 12/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5987 - acc
uracy: 0.7425 - val_loss: 0.4299 - val_accuracy: 0.8150
Epoch 13/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5784 - acc
uracy: 0.7522 - val_loss: 0.4267 - val_accuracy: 0.8179
Epoch 14/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5696 - acc
uracy: 0.7588 - val_loss: 0.4240 - val_accuracy: 0.8200
Epoch 15/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5748 - acc
uracy: 0.7570 - val_loss: 0.4222 - val_accuracy: 0.8207
Epoch 16/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5621 - acc
uracy: 0.7676 - val_loss: 0.4206 - val_accuracy: 0.8214
Epoch 17/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5447 - acc
uracy: 0.7652 - val_loss: 0.4192 - val_accuracy: 0.8221
Epoch 18/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5392 - acc
uracy: 0.7668 - val_loss: 0.4182 - val_accuracy: 0.8243
Epoch 19/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5405 - acc
```

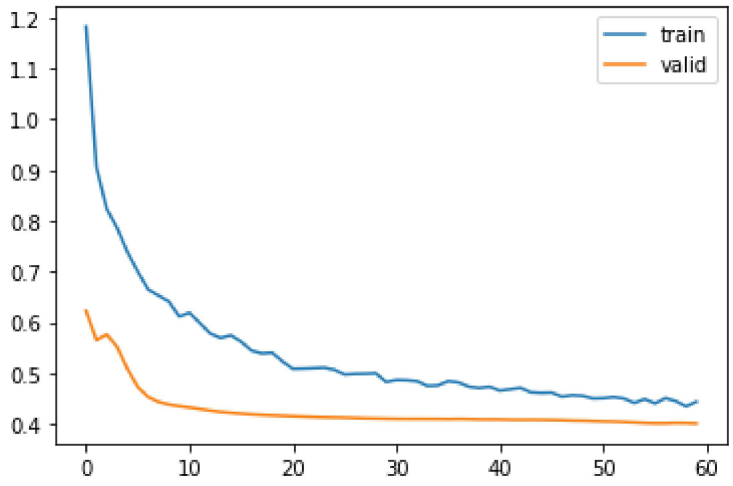
```
uracy: 0.7611 - val_loss: 0.4173 - val_accuracy: 0.8250
Epoch 20/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5232 - acc
uracy: 0.7765 - val_loss: 0.4164 - val_accuracy: 0.8264
Epoch 21/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5089 - acc
uracy: 0.7797 - val_loss: 0.4156 - val_accuracy: 0.8264
Epoch 22/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5094 - acc
uracy: 0.7856 - val_loss: 0.4149 - val_accuracy: 0.8271
Epoch 23/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5102 - acc
uracy: 0.7836 - val_loss: 0.4141 - val_accuracy: 0.8271
Epoch 24/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5112 - acc
uracy: 0.7804 - val_loss: 0.4136 - val_accuracy: 0.8271
Epoch 25/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5069 - acc
uracy: 0.7845 - val_loss: 0.4132 - val_accuracy: 0.8271
Epoch 26/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4977 - acc
uracy: 0.7877 - val_loss: 0.4128 - val_accuracy: 0.8286
Epoch 27/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4989 - acc
uracy: 0.7827 - val_loss: 0.4121 - val_accuracy: 0.8286
Epoch 28/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4992 - acc
uracy: 0.7843 - val_loss: 0.4115 - val_accuracy: 0.8279
Epoch 29/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.5001 - acc
uracy: 0.7836 - val_loss: 0.4112 - val_accuracy: 0.8286
Epoch 30/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4830 - acc
uracy: 0.7935 - val_loss: 0.4108 - val_accuracy: 0.8300
Epoch 31/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4870 - acc
uracy: 0.7942 - val_loss: 0.4104 - val_accuracy: 0.8300
Epoch 32/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4864 - acc
uracy: 0.7885 - val_loss: 0.4103 - val_accuracy: 0.8300
Epoch 33/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4845 - acc
uracy: 0.7901 - val_loss: 0.4103 - val_accuracy: 0.8300
Epoch 34/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4752 - acc
uracy: 0.7961 - val_loss: 0.4102 - val_accuracy: 0.8300
Epoch 35/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4761 - acc
uracy: 0.7954 - val_loss: 0.4101 - val_accuracy: 0.8307
Epoch 36/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4847 - acc
uracy: 0.7899 - val_loss: 0.4100 - val_accuracy: 0.8307
Epoch 37/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4821 - acc
uracy: 0.7931 - val_loss: 0.4103 - val_accuracy: 0.8286
Epoch 38/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4737 - acc
```

```
uracy: 0.7926 - val_loss: 0.4099 - val_accuracy: 0.8300
Epoch 39/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4711 - acc
uracy: 0.7926 - val_loss: 0.4093 - val_accuracy: 0.8307
Epoch 40/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4736 - acc
uracy: 0.7952 - val_loss: 0.4093 - val_accuracy: 0.8321
Epoch 41/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4660 - acc
uracy: 0.8028 - val_loss: 0.4092 - val_accuracy: 0.8336
Epoch 42/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4686 - acc
uracy: 0.7981 - val_loss: 0.4087 - val_accuracy: 0.8343
Epoch 43/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4714 - acc
uracy: 0.7992 - val_loss: 0.4085 - val_accuracy: 0.8336
Epoch 44/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4626 - acc
uracy: 0.8061 - val_loss: 0.4086 - val_accuracy: 0.8336
Epoch 45/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4615 - acc
uracy: 0.7954 - val_loss: 0.4084 - val_accuracy: 0.8329
Epoch 46/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4621 - acc
uracy: 0.8065 - val_loss: 0.4081 - val_accuracy: 0.8336
Epoch 47/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4542 - acc
uracy: 0.8026 - val_loss: 0.4077 - val_accuracy: 0.8343
Epoch 48/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4568 - acc
uracy: 0.8086 - val_loss: 0.4070 - val_accuracy: 0.8343
Epoch 49/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4557 - acc
uracy: 0.8065 - val_loss: 0.4067 - val_accuracy: 0.8343
Epoch 50/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4505 - acc
uracy: 0.8108 - val_loss: 0.4060 - val_accuracy: 0.8336
Epoch 51/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4511 - acc
uracy: 0.8060 - val_loss: 0.4053 - val_accuracy: 0.8336
Epoch 52/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4533 - acc
uracy: 0.8111 - val_loss: 0.4049 - val_accuracy: 0.8329
Epoch 53/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4504 - acc
uracy: 0.8029 - val_loss: 0.4042 - val_accuracy: 0.8321
Epoch 54/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4418 - acc
uracy: 0.8153 - val_loss: 0.4033 - val_accuracy: 0.8343
Epoch 55/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4493 - acc
uracy: 0.8078 - val_loss: 0.4023 - val_accuracy: 0.8350
Epoch 56/60
5597/5597 [==============================] - 0s 4us/step - loss: 0.4408 - acc
uracy: 0.8072 - val_loss: 0.4016 - val_accuracy: 0.8343
Epoch 57/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4512 - acc
```

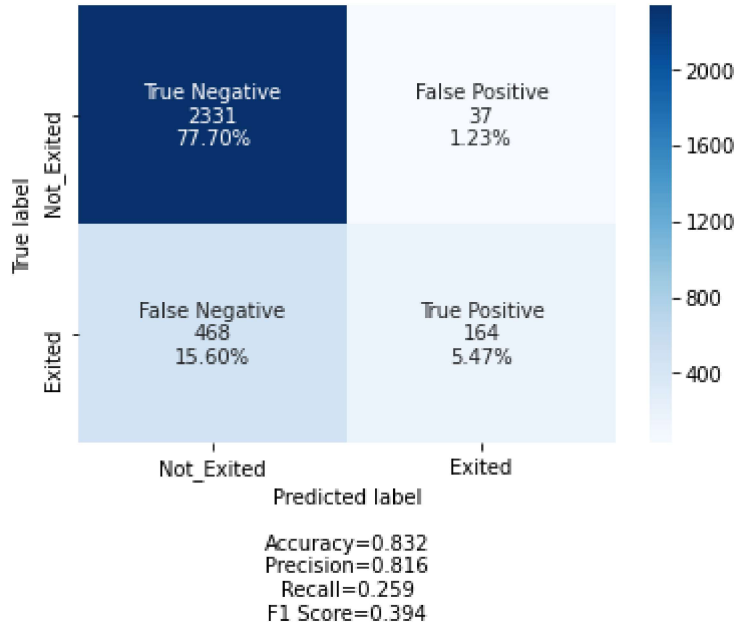
```
uracy: 0.8133 - val_loss: 0.4017 - val_accuracy: 0.8336
Epoch 58/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4454 - acc
uracy: 0.8147 - val_loss: 0.4023 - val_accuracy: 0.8321
Epoch 59/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4348 - acc
uracy: 0.8162 - val_loss: 0.4021 - val_accuracy: 0.8336
Epoch 60/60
5597/5597 [==============================] - 0s 3us/step - loss: 0.4441 - acc
uracy: 0.8081 - val_loss: 0.4012 - val_accuracy: 0.8336
```

In [41]:
```
# Learning history per epoch
hist  = pd.DataFrame(his_mod2.history)
hist['epoch'] = his_mod2.epoch

# Accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)
```

Out[41]:
```
<matplotlib.legend.Legend at 0x1c630401908>
```



In [42]:
```
y_pred1 = model2.predict(X_test)
for i in range(len(y_test)):
    if y_pred1[i]>0.5:
        y_pred1[i]=1
    else:
        y_pred1[i]=0



cm2=confusion_matrix(y_test, y_pred1)
labels = ['True Negative','False Positive','False Negative','True Positive']
categories = [ 'Not_Exited','Exited']
make_cm(cm2,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



Model 3:

Recall weighted model because of unbalanced data

In [43]:
```
from warnings import simplefilter # prevents error cell below
simplefilter(action='ignore', category=FutureWarning)
```

In [44]:
```
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight('balanced', np.unique(y_trai
n['Exited']), y_train['Exited'])
class_weights = dict(enumerate(class_weights))
class_weights
```

Out[44]:
```
{0: 0.6256258941344778, 1: 2.490035587188612}
```

In [45]:
```
from sklearn.utils import class_weight
model.fit(X_train,y_train,batch_size=15,epochs=10, class_weight=class_weights, shuffle=True)
```

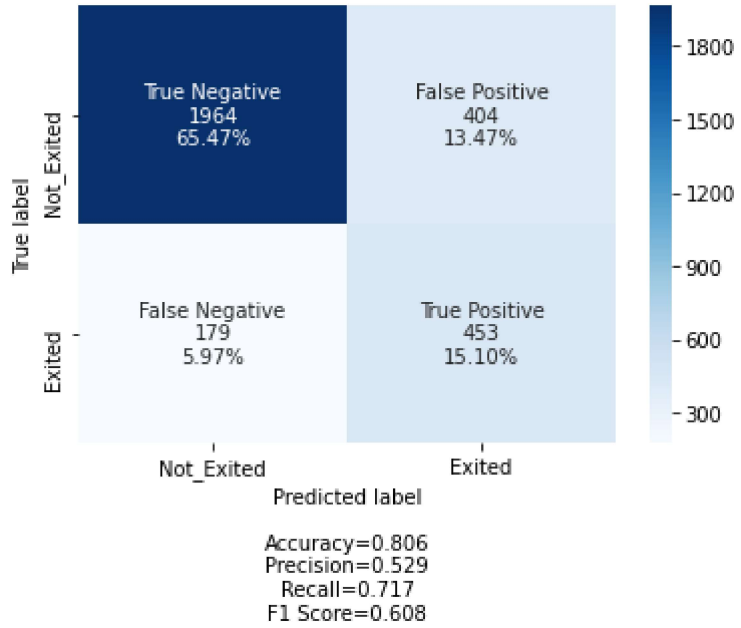
```
Epoch 1/10
6997/6997 [==============================] - 0s 50us/step - loss: 0.4868 - ac
curacy: 0.7953
Epoch 2/10
6997/6997 [==============================] - 0s 49us/step - loss: 0.4838 - ac
curacy: 0.7838
Epoch 3/10
6997/6997 [==============================] - 0s 49us/step - loss: 0.4797 - ac
curacy: 0.7889
Epoch 4/10
6997/6997 [==============================] - 0s 51us/step - loss: 0.4745 - ac
curacy: 0.7929
Epoch 5/10
6997/6997 [==============================] - 0s 50us/step - loss: 0.4705 - ac
curacy: 0.7932
Epoch 6/10
6997/6997 [==============================] - 0s 51us/step - loss: 0.4761 - ac
curacy: 0.7936
Epoch 7/10
6997/6997 [==============================] - 0s 48us/step - loss: 0.4688 - ac
curacy: 0.7925
Epoch 8/10
6997/6997 [==============================] - 0s 49us/step - loss: 0.4711 - ac
curacy: 0.7879
Epoch 9/10
6997/6997 [==============================] - 0s 49us/step - loss: 0.4659 - ac
curacy: 0.7925
Epoch 10/10
6997/6997 [==============================] - 0s 47us/step - loss: 0.4688 - ac
curacy: 0.7933
```

Out[45]:
```
<keras.callbacks.callbacks.History at 0x1c6317f88c8>
```

In [46]:
```
score_weighted = model.evaluate(X_test, y_test)
```

```
3000/3000 [==============================] - 0s 12us/step
```

In [47]:
```
y_pred1 = model.predict(X_test)
for i in range(len(y_test)):
    if y_pred1[i]>0.5:
        y_pred1[i]=1
    else:
        y_pred1[i]=0



cm2=confusion_matrix(y_test, y_pred1)
labels = ['True Negative','False Positive','False Negative','True Positive']
categories = [ 'Not_Exited','Exited']
make_cm(cm2,
                        group_names=labels,
                        categories=categories,
                        cmap='Blues')
```



Overall:

1. Protect Age target of 35 to 55.
2. Non-active members are more likely to leave.
3. German are most likely to leave compared to French and Spanish
4. Customers with higher balnace are slightly more likely to leave.
5. Depending on the preference, Recall could be most significant.
6. Although data interpretation may not be smooth, Neural Network provides flexible models

In [ ]: