In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn import preprocessing
```

Loading the Dataset

First we load the dataset and find out the number of columns, rows, NULL values, etc.

In [2]:
```
df = pd.read_csv('churn_modelling.csv')
```

In [3]:
```
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        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
```

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

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProdu
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	

Cleaning

In [5]:
```
df.drop(columns=['RowNumber', 'CustomerId', 'Surname'], inplace=True)
```

In [6]:
```
df.isna().sum()
```

Out[6]:
```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [7]:
```
df.describe()
```

Out[7]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.00000(
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.51510(
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.49979
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.00000(
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.00000(
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.00000(
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.00000(
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.00000(

Separating the features and the labels

In [8]:
```
X=df.iloc[:, :df.shape[1]-1].values        #Independent Variables
y=df.iloc[:, -1].values                    #Dependent Variable
X.shape, y.shape
```

Out[8]:
```
((10000, 10), (10000,))
```

Encoding categorical (string based) data.

In [9]:
```
print(X[:8,1], '... will now become: ')

label_X_country_encoder = LabelEncoder()
X[:,1] = label_X_country_encoder.fit_transform(X[:,1])
print(X[:8,1])
```

```
['France' 'Spain' 'France' 'France' 'Spain' 'Spain' 'France' 'Germany'] ... will now become:
[0 2 0 0 2 2 0 1]
```

In [10]:
```
print(X[:6,2], '... will now become: ')

label_X_gender_encoder = LabelEncoder()
X[:,2] = label_X_gender_encoder.fit_transform(X[:,2])
print(X[:6,2])
```

```
['Female' 'Female' 'Female' 'Female' 'Female' 'Male'] ... will now become:
[0 0 0 0 0 1]
```

Split the countries into respective dimensions. Converting the string features into their own dimensions.

In [11]:
```
transform = ColumnTransformer([("countries", OneHotEncoder(), [1])], remainder="passthrough") #
X = transform.fit_transform(X)
X
```

Out[11]:
```
array([[1.0, 0.0, 0.0, ..., 1, 1, 101348.88],
       [0.0, 0.0, 1.0, ..., 0, 1, 112542.58],
       [1.0, 0.0, 0.0, ..., 1, 0, 113931.57],
       ...,
       [1.0, 0.0, 0.0, ..., 0, 1, 42085.58],
       [0.0, 1.0, 0.0, ..., 1, 0, 92888.52],
       [1.0, 0.0, 0.0, ..., 1, 0, 38190.78]], dtype=object)
```

Dimensionality reduction. A 0 on two countries means that the country has to be the one variable which wasn't included

In [12]:
```
X = X[:,1:]
X.shape
```

Out[12]:
```
(10000, 11)
```

Splitting the Dataset

Training and Test Set

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Normalize the train and test data

`['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary']`

In [14]:
```
sc=StandardScaler()
X_train[:,np.array([2,4,5,6,7,10])] = sc.fit_transform(X_train[:,np.array([2,4,5,6,7,10])])
X_test[:,np.array([2,4,5,6,7,10])] = sc.transform(X_test[:,np.array([2,4,5,6,7,10])])
```

In [15]:
```
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

Out[15]:
```
array([[-0.5698444 ,  1.74309049,  0.16958176, ...,  0.64259497,
        -1.03227043,  1.10643166],
       [ 1.75486502, -0.57369368, -2.30455945, ...,  0.64259497,
         0.9687384 , -0.74866447],
       [-0.5698444 , -0.57369368, -1.19119591, ...,  0.64259497,
        -1.03227043,  1.48533467],
       ...,
       [-0.5698444 , -0.57369368,  0.9015152 , ...,  0.64259497,
        -1.03227043,  1.41231994],
       [-0.5698444 ,  1.74309049, -0.62420521, ...,  0.64259497,
         0.9687384 ,  0.84432121],
       [ 1.75486502, -0.57369368, -0.28401079, ...,  0.64259497,
        -1.03227043,  0.32472465]])
```

Initialize & build the model

INPUT = Number columns (Independet) HIDDEN - AF HIDDEN -AF . . . N OUTPUT (1,2) -Sigmoid

In [16]:
```
from tensorflow.keras.models import Sequential

# Initializing the ANN
classifier = Sequential()
```

In [17]:
```
from tensorflow.keras.layers import Dense

# The amount of nodes (dimensions) in hidden layer should be the average of input and output la
# This adds the input layer (by specifying input dimension) AND the first hidden layer (units)
classifier.add(Dense(activation = 'relu', input_dim = 11, units=256, kernel_initializer='unifor
```

In [18]:
```
# Adding the hidden layer
classifier.add(Dense(activation = 'relu', units=512, kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=256, kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=128, kernel_initializer='uniform'))
```

In [19]:
```
# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output (stay with the
# We use the sigmoid because we want probability outcomes
classifier.add(Dense(activation = 'sigmoid', units=1, kernel_initializer='uniform'))
```

In [20]:
```
# Create optimizer with default learning rate
# sgd_optimizer = tf.keras.optimizers.SGD()
# Compile the model
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [21]:
```
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 256)               3072      
                                                                 
 dense_1 (Dense)             (None, 512)               131584    
                                                                 
 dense_2 (Dense)             (None, 256)               131328    
                                                                 
 dense_3 (Dense)             (None, 128)               32896     
                                                                 
 dense_4 (Dense)             (None, 1)                 129       
                                                                 
=================================================================
Total params: 299,009
Trainable params: 299,009
Non-trainable params: 0
_________________________________________________________________
```

In [22]:
```
classifier.fit(
    X_train, y_train,
    validation_data=(X_test,y_test),
    epochs=20,
    batch_size=32
)
```

```
Epoch 1/20
250/250 [==============================] - 1s 3ms/step - loss: 0.4378 - accuracy: 0.8163 - val
_loss: 0.3689 - val_accuracy: 0.8480
Epoch 2/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3638 - accuracy: 0.8510 - val
_loss: 0.3509 - val_accuracy: 0.8590
Epoch 3/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3485 - accuracy: 0.8575 - val
_loss: 0.3412 - val_accuracy: 0.8585
Epoch 4/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3435 - accuracy: 0.8631 - val
_loss: 0.3468 - val_accuracy: 0.8645
Epoch 5/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3407 - accuracy: 0.8600 - val
_loss: 0.3440 - val_accuracy: 0.8620
Epoch 6/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3394 - accuracy: 0.8621 - val
_loss: 0.3385 - val_accuracy: 0.8645
Epoch 7/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3332 - accuracy: 0.8629 - val
_loss: 0.3372 - val_accuracy: 0.8625
Epoch 8/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3264 - accuracy: 0.8695 - val
_loss: 0.3410 - val_accuracy: 0.8605
Epoch 9/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3258 - accuracy: 0.8680 - val
_loss: 0.3419 - val_accuracy: 0.8640
Epoch 10/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3226 - accuracy: 0.8700 - val
_loss: 0.3418 - val_accuracy: 0.8630
Epoch 11/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3198 - accuracy: 0.8700 - val
_loss: 0.3416 - val_accuracy: 0.8650
Epoch 12/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3154 - accuracy: 0.8725 - val
_loss: 0.3461 - val_accuracy: 0.8580
Epoch 13/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3122 - accuracy: 0.8749 - val
_loss: 0.3400 - val_accuracy: 0.8615
Epoch 14/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3094 - accuracy: 0.8739 - val
_loss: 0.3443 - val_accuracy: 0.8620
Epoch 15/20
250/250 [==============================] - 1s 3ms/step - loss: 0.3051 - accuracy: 0.8761 - val
_loss: 0.3529 - val_accuracy: 0.8570
Epoch 16/20
250/250 [==============================] - 1s 3ms/step - loss: 0.2997 - accuracy: 0.8790 - val
_loss: 0.3541 - val_accuracy: 0.8595
Epoch 17/20
250/250 [==============================] - 1s 3ms/step - loss: 0.2939 - accuracy: 0.8774 - val
_loss: 0.3487 - val_accuracy: 0.8635
Epoch 18/20
250/250 [==============================] - 1s 3ms/step - loss: 0.2882 - accuracy: 0.8836 - val
_loss: 0.3876 - val_accuracy: 0.8530
Epoch 19/20
250/250 [==============================] - 1s 3ms/step - loss: 0.2871 - accuracy: 0.8796 - val
_loss: 0.3724 - val_accuracy: 0.8515
Epoch 20/20
```

```
250/250 [==============================] - 1s 3ms/step - loss: 0.2782 - accuracy: 0.8854 - val
_loss: 0.3754 - val_accuracy: 0.8615
```

Out[22]:
```
<keras.callbacks.History at 0x7f1a35f238b0>
```

Predict the results using 0.5 as a threshold

In [23]:
```
y_pred = classifier.predict(X_test)
y_pred
```

```
63/63 [==============================] - 0s 977us/step
```

Out[23]:
```
array([[0.2910964 ],
       [0.17213912],
       [0.14321707],
       ...,
       [0.00948479],
       [0.1535894 ],
       [0.11742345]], dtype=float32)
```

In [24]:
```
# To use the confusion Matrix, we need to convert the probabilities that a customer will leave
# So we will use the cutoff value 0.5 to indicate whether they are likely to exit or not.
y_pred = (y_pred > 0.5)
y_pred
```

Out[24]:
```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

Print the Accuracy score and confusion matrix

In [25]:
```
from sklearn.metrics import confusion_matrix,classification_report

cm1 = confusion_matrix(y_test, y_pred)
cm1
```

Out[25]:
```
array([[1526,   69],
       [ 208,  197]])
```

In [26]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1595
           1       0.74      0.49      0.59       405

    accuracy                           0.86      2000
   macro avg       0.81      0.72      0.75      2000
weighted avg       0.85      0.86      0.85      2000
```

In [27]:
```
accuracy_model1 = ((cm1[0][0]+cm1[1][1])*100)/(cm1[0][0]+cm1[1][1]+cm1[0][1]+cm1[1][0])
print (accuracy_model1, '% of testing data was classified correctly')
```

```
86.15 % of testing data was classified correctly
```