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
```

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

In [5]:
```
df.isna().sum()
```

Out[5]:
```
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 [6]:
```
X=df.iloc[:, :df.shape[1]-1].values #Independent Variables
y=df.iloc[:, -1].values #Dependent Variable
X.shape, y.shape
```

Out[6]:
```
((10000, 10), (10000,))
```

In [7]:
```
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 [8]:
```
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]
```

In [9]:
```
transform = ColumnTransformer([("countries", OneHotEncoder(), [1])], remainder="passthrough") #
X = transform.fit_transform(X)
X
```

Out[9]:
```
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)
```

In [10]:
```
X = X[:,1:]
X.shape
```

Out[10]:
```
(10000, 11)
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [12]:
```
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 [14]:
```
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

Out[14]:
```
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]])
```

In [15]:
```
from tensorflow.keras.models import Sequential
# Initializing the ANN
classifier = Sequential()
```

In [16]:
```
from tensorflow.keras.layers import Dense
```

In [18]:
```
classifier.add(Dense(activation = 'relu', input_dim = 11, units=256, kernel_initializer='unifo
```

In [19]:
```
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 [20]:
```
classifier.add(Dense(activation = 'sigmoid', units=1, kernel_initializer='uniform'))
```

In [21]:
```
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [22]:
```
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 256)               3072

 dense_2 (Dense)             (None, 512)               131584

 dense_3 (Dense)             (None, 256)               131328

 dense_4 (Dense)             (None, 128)               32896

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 299009 (1.14 MB)
Trainable params: 299009 (1.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [23]:
```
classifier.fit(
 X_train, y_train,
 validation_data=(X_test,y_test),
 epochs=20,
 batch_size=32
)
```

```
Epoch 1/20
250/250 [==============================] - 6s 12ms/step - loss: 0.4291 - accuracy: 0.8200 - v
al_loss: 0.3551 - val_accuracy: 0.8615
Epoch 2/20
250/250 [==============================] - 2s 10ms/step - loss: 0.3606 - accuracy: 0.8553 - v
al_loss: 0.3629 - val_accuracy: 0.8700
Epoch 3/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3450 - accuracy: 0.8584 - v
al_loss: 0.3508 - val_accuracy: 0.8575
Epoch 4/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3416 - accuracy: 0.8629 - v
al_loss: 0.3407 - val_accuracy: 0.8625
Epoch 5/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3387 - accuracy: 0.8611 - v
al_loss: 0.3357 - val_accuracy: 0.8675
Epoch 6/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3356 - accuracy: 0.8656 - v
al_loss: 0.3392 - val_accuracy: 0.8650
Epoch 7/20
250/250 [==============================] - 3s 10ms/step - loss: 0.3309 - accuracy: 0.8649 - v
al_loss: 0.3372 - val_accuracy: 0.8580
Epoch 8/20
250/250 [==============================] - 3s 10ms/step - loss: 0.3292 - accuracy: 0.8664 - v
al_loss: 0.3379 - val_accuracy: 0.8625
Epoch 9/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3262 - accuracy: 0.8670 - v
al_loss: 0.3729 - val_accuracy: 0.8485
Epoch 10/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3236 - accuracy: 0.8706 - v
al_loss: 0.3482 - val_accuracy: 0.8515
Epoch 11/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3182 - accuracy: 0.8710 - v
al_loss: 0.3340 - val_accuracy: 0.8675
Epoch 12/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3155 - accuracy: 0.8696 - v
al_loss: 0.3402 - val_accuracy: 0.8595
Epoch 13/20
250/250 [==============================] - 3s 10ms/step - loss: 0.3102 - accuracy: 0.8742 - v
al_loss: 0.3418 - val_accuracy: 0.8620
Epoch 14/20
250/250 [==============================] - 3s 10ms/step - loss: 0.3068 - accuracy: 0.8730 - v
al_loss: 0.3674 - val_accuracy: 0.8580
Epoch 15/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3036 - accuracy: 0.8756 - v
al_loss: 0.3448 - val_accuracy: 0.8580
Epoch 16/20
250/250 [==============================] - 3s 11ms/step - loss: 0.2988 - accuracy: 0.8795 - v
al_loss: 0.3455 - val_accuracy: 0.8595
Epoch 17/20
250/250 [==============================] - 3s 11ms/step - loss: 0.2915 - accuracy: 0.8838 - v
al_loss: 0.3696 - val_accuracy: 0.8515
Epoch 18/20
250/250 [==============================] - 3s 11ms/step - loss: 0.2837 - accuracy: 0.8855 - v
al_loss: 0.3598 - val_accuracy: 0.8590
Epoch 19/20
250/250 [==============================] - 3s 10ms/step - loss: 0.2813 - accuracy: 0.8845 - v
al_loss: 0.3747 - val_accuracy: 0.8565
Epoch 20/20
250/250 [==============================] - 3s 10ms/step - loss: 0.2740 - accuracy: 0.8882 - v
al_loss: 0.3831 - val_accuracy: 0.8600
```

Out[23]: <keras.src.callbacks.History at 0x19347b54150>

In [32]:
```
y_pred = classifier.predict(X_test)
y_pred
```

```
63/63 [==============================] - 0s 5ms/step
```

Out[32]:
```
array([[0.40783623],
       [0.2887754 ],
       [0.10280854],
       ...,
       [0.09997811],
       [0.19579199],
       [0.2798526 ]], dtype=float32)
```

In [33]:
```
y_pred = (y_pred > 0.5)
y_pred
```

Out[33]:
```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [34]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [38]:
```
cm1 = confusion_matrix(y_test,y_pred)
```

In [39]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      1595
           1       0.69      0.56      0.62       405

    accuracy                           0.86      2000
   macro avg       0.79      0.75      0.77      2000
weighted avg       0.85      0.86      0.85      2000
```

In [40]:
```
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')
```

```
86.0 % of testing data was classified correctly
```

In [ ]:
```

```