In [1]:
```
import numpy as np
import pandas as pd

import tensorflow as tf
import keras
```

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

Out[2]:

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

In [3]:
```
df["Exited"].value_counts()
```

Out[3]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

In [4]:
```
1 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 [5]:
```
1 df.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
df.drop(columns=["RowNumber","CustomerId","Surname"],inplace=True)
df
```

Out[6]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...
9995	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 11 columns

In [7]:
```
df['Geography'].value_counts()
```

Out[7]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [8]:
```
df['Gender'].value_counts()
```

Out[8]:
```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

In [9]:
```
X = pd.get_dummies(df.drop('Exited',axis=1),drop_first=True)
y = df['Exited']
```

In [10]:
```
1 X
```

Out[10]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_Germany	Geograpl
0	619	42	2	0.00	1	1	1	101348.88	0	
1	608	41	1	83807.86	1	0	1	112542.58	0	
2	502	42	8	159660.80	3	1	0	113931.57	0	
3	699	39	1	0.00	2	0	0	93826.63	0	
4	850	43	2	125510.82	1	1	1	79084.10	0	
...	...	...	...	...	...	...	...	...	...	
9995	771	39	5	0.00	2	1	0	96270.64	0	
9996	516	35	10	57369.61	1	1	1	101699.77	0	
9997	709	36	7	0.00	1	0	1	42085.58	0	
9998	772	42	3	75075.31	2	1	0	92888.52	1	
9999	792	28	4	130142.79	1	1	0	38190.78	0	

10000 rows × 11 columns

In [11]:
```
1 ### Splitting the dataset into the Training set and Test set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

In [12]:
```
1 X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[12]:
```
((8000, 11), (2000, 11), (8000,), (2000,))
```

In [13]:
```
1 from sklearn.preprocessing import StandardScaler
2 sc = StandardScaler()
3 X_train = sc.fit_transform(X_train)
4 X_test = sc.transform(X_test)
```

Modelling

Initializing the Artificial Neural Network

In [14]:
```
from keras.models import Sequential
ann = Sequential()
```

Adding the input layer and the first hidden layer

In [15]:
```
from keras.layers import Dense
ann.add(Dense(input_dim = 11,units = 6, kernel_initializer = 'uniform', activation = 'relu'))

#ann.add(Dropout(rate = 0.1))
```

Adding the second hidden layer

In [16]:
```
ann.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

#ann.add(Dropout(rate = 0.1))
```

Adding the output layer

In [17]:
```
ann.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

Compiling the ANN

In [18]:
```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the ANN on the Training set

In [19]:
```
ann.fit(X_train, y_train, epochs = 100,batch_size=32)
```

```
250/250 [==============================] - 0s 1ms/step - loss: 0.3399 - accuracy: 0.8615
Epoch 60/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3401 - accuracy: 0.8614
Epoch 61/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3400 - accuracy: 0.8601
Epoch 62/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3405 - accuracy: 0.8610
Epoch 63/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3396 - accuracy: 0.8627
Epoch 64/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3394 - accuracy: 0.8594
Epoch 65/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3402 - accuracy: 0.8606
Epoch 66/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3388 - accuracy: 0.8612
Epoch 67/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3397 - accuracy: 0.8633
Epoch 68/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3397 - accuracy: 0.8627
```

Predictions & Evaluating the model

Predictions

In [20]:
```
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
```

```
63/63 [==============================] - 0s 2ms/step
```

Evaluation

In [21]:
```
from sklearn.metrics import confusion_matrix, accuracy_score

print("Test Accuracy:",accuracy_score(y_test, y_pred))
confusion_matrix(y_test, y_pred)
```

```
Test Accuracy: 0.8555
```

Out[21]:
```
array([[1501,   94],
       [ 195,  210]], dtype=int64)
```

Cross validate the model

In [22]:
```
def build_cross_classifier():
    classifier = Sequential()
    classifier.add(Dense(input_dim = 11, units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = "adam", loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier
```

In [23]:
```
from scikeras.wrappers import KerasClassifier
classifier = KerasClassifier(build_cross_classifier,batch_size=32,epochs=100)
```

In [24]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(classifier, X, y, cv = 5)
```

```
Epoch 1/100
250/250 [==============================] - 1s 1ms/step - loss: 0.5371 - accuracy: 0.7937
Epoch 2/100
250/250 [==============================] - 0s 1ms/step - loss: 0.5236 - accuracy: 0.7964
Epoch 3/100
250/250 [==============================] - 0s 2ms/step - loss: 0.5169 - accuracy: 0.7964
Epoch 4/100
250/250 [==============================] - 0s 1ms/step - loss: 0.5104 - accuracy: 0.7964
Epoch 5/100
250/250 [==============================] - 0s 1ms/step - loss: 0.5076 - accuracy: 0.7964
Epoch 6/100
250/250 [==============================] - 0s 2ms/step - loss: 0.5083 - accuracy: 0.7964
Epoch 7/100
250/250 [==============================] - 0s 1ms/step - loss: 0.5044 - accuracy: 0.7964
Epoch 8/100
250/250 [==============================] - 0s 1ms/step - loss: 0.5062 - accuracy: 0.7964
Epoch 9/100
250/250 [==============================] - 0s 1ms/step - loss: 0.5055 - accuracy: 0.7964
Epoch 10/100
```

In [25]:
```
print(accuracies)
accuracies.mean()
```

```
[0.796  0.796  0.7965 0.7965 0.7965]
```

Out[25]:
```
0.7963
```


Improving and Tuning the ANN

This can be done by using 3 options

- 1.Hyperparameter Tuning
- 2.Regularization (L1 & L2) to reduce overfitting if needed (for Regression problems)
- 3.Dropout

Hyperparameter tuning can be done for identifying no.of hidden layers & no.of neurons in each hidden layer

- layers = [[20],[10],[30],[40]] --> 1 hidden layer with different options of no.of neurons in that hidden layer
- layers = [[20],[40,20],[45,30,15]] --> Multiple hidden layers with different options of no.of neurons

Hyperparameter tuning can be done for identifying best activation function for hidden layers

- activations=['relu','sigmoid']

Hyperparameter tuning can be done for identifying best optimizers

- optimizer =['adam','rmsprop']

Hyperparameter tuning can be done for identifying best batchsize for building/training model

- batchsize =[10,16,32,64,128,256]

In [27]:
```
estimator = KerasClassifier(build_cross_classifier())

param_grid = {'batch_size': [10,32], 'epochs': [50,100],"optimizer" :['adam','rmsprop']}
```

In [28]:
```
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(estimator, param_grid, scoring = 'accuracy', cv = 5)

grid_result = grid.fit(X_train, y_train)
```

...

In [29]:
```
#best parameters
grid_result.best_params_
```

Out[29]:
```
{'batch_size': 10, 'epochs': 100, 'optimizer': 'rmsprop'}
```

In [30]:
```
#best accuracy
grid_result.best_score_
```

Out[30]:
```
0.8428749999999999
```