In [1]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

```
2024-11-10 19:01:37.682772: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already been registered
2024-11-10 19:01:37.682819: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT
factory: Attempting to register factory for plugin cuFFT when one has already been registered
2024-11-10 19:01:37.684241: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBL
AS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2024-11-10 19:01:37.691945: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized
to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compile
r flags.
2024-11-10 19:01:40.514835: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find T
ensorRT
```

In [6]:

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

In [7]:

```
dataset.head()
```

Out[7]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActive
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 [8]:

```
# Step 2: Preprocessing the data
# Drop irrelevant columns
dataset = dataset.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
```

In [9]:

```
dataset.head()
```

Out[9]:

	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

In [10]:

```
# Encoding categorical variables
dataset = pd.get_dummies(dataset, drop_first=True)
```

In [11]:

```
# Step 3: Distinguishing features and target
X = dataset.drop('Exited', axis=1)  # Features
y = dataset['Exited']  # Target
```

In [12]:

```
# Step 4: Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [13]:

```
# Step 5: Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [14]:

```
# Step 6: Building the Neural Network model
model = Sequential()

# Adding input layer and the first hidden layer
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))

# Adding a second hidden layer
```

```
model.add(Dense(units=32, activation='relu'))

# Adding the output layer
model.add(Dense(units=1, activation='sigmoid'))  # Binary classification
```

In [15]:

```
# Step 7: Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 8: Training the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Step 9: Predicting the results on the test set
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)  # Converting probabilities to binary values (0 or 1)
```

```
Epoch 1/10
250/250 [==============================] - 2s 2ms/step - loss: 0.4768 - accuracy: 0.7824
Epoch 2/10
250/250 [==============================] - 0s 2ms/step - loss: 0.3891 - accuracy: 0.8376
Epoch 3/10
250/250 [==============================] - 0s 2ms/step - loss: 0.3564 - accuracy: 0.8551
Epoch 4/10
250/250 [==============================] - 0s 2ms/step - loss: 0.3457 - accuracy: 0.8577
Epoch 5/10
250/250 [==============================] - 0s 1ms/step - loss: 0.3406 - accuracy: 0.8594
Epoch 6/10
250/250 [==============================] - 0s 2ms/step - loss: 0.3377 - accuracy: 0.8621
Epoch 7/10
250/250 [==============================] - 0s 2ms/step - loss: 0.3336 - accuracy: 0.8625
Epoch 8/10
250/250 [==============================] - 0s 2ms/step - loss: 0.3301 - accuracy: 0.8644
Epoch 9/10
250/250 [==============================] - 0s 2ms/step - loss: 0.3277 - accuracy: 0.8656
Epoch 10/10
250/250 [==============================] - 0s 1ms/step - loss: 0.3259 - accuracy: 0.8654
63/63 [==============================] - 0s 1ms/step
```

In [16]:

```
# Step 10: Evaluating the model
# Accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Accuracy: 86.25%
Confusion Matrix:
[[1542   65]
 [ 210  183]]
```