In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [2]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

In [3]:
```
df = pd.read_csv("C:/Users/Adamin/OneDrive/Desktop/Churn_Modelling.csv")
df.head()
```

Out[3]:

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

In [4]:
```
# 2. Distinguish the feature and target set
X = df.iloc[:, 3:13]   # Features (from CreditScore to EstimatedSalary)
y = df.iloc[:, 13]     # Target (Exited column)
# Encoding categorical variables (Geography and Gender)
labelencoder_gender = LabelEncoder()
X['Gender'] = labelencoder_gender.fit_transform(X['Gender'])


# One-hot encode Geography
X = pd.get_dummies(X, columns=['Geography'], drop_first=True)
# 3. Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
# Normalize the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train_df = pd.DataFrame(X_train, columns=X.columns)

print("Sample of normalized training data:")
display(X_train_df.head())

print("\nMean of features after scaling:\n", X_train_df.mean())
print("\nStandard deviation of features after scaling:\n", X_train_df.std())
```

```
Sample of normalized training data:
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is
0	0.356500	0.913248	-0.655786	0.345680	-1.218471	0.808436	0.649203	
1	-0.203898	0.913248	0.294938	-0.348369	0.696838	0.808436	0.649203	
2	-0.961472	0.913248	-1.416365	-0.695393	0.618629	-0.916688	0.649203	
3	-0.940717	-1.094993	-1.131148	1.386753	0.953212	-0.916688	0.649203	
4	-1.397337	0.913248	1.625953	1.386753	1.057449	-0.916688	-1.540351	

```
Mean of features after scaling:
 CreditScore          5.435652e-16
Gender               2.842171e-17
Age                 -1.896261e-16
Tenure               6.661338e-17
Balance             -3.099743e-16
NumOfProducts        1.065814e-16
HasCrCard           -3.197442e-17
IsActiveMember      -1.776357e-17
EstimatedSalary      1.776357e-17
Geography_Germany   -4.263256e-17
Geography_Spain     -7.194245e-17
dtype: float64

Standard deviation of features after scaling:
 CreditScore          1.000063
Gender               1.000063
Age                  1.000063
Tenure               1.000063
Balance              1.000063
NumOfProducts        1.000063
HasCrCard            1.000063
IsActiveMember       1.000063
EstimatedSalary      1.000063
Geography_Germany    1.000063
Geography_Spain      1.000063
dtype: float64
```

In [5]:
```
# 4. Initialize and build the Neural Network model
model = Sequential()
model.add(Dense(units=6, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.3))
model.add(Dense(units=6, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
```

```
C:\Users\Adamin\AppData\Roaming\Python\Python313\site-packages\keras\src\layers\core\dense.py:95: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [6]:
```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']
```

In [7]:
```
# Train for 20 epochs
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=
```

```
Epoch 1/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7384 - loss: 0.5717 - val_
accuracy: 0.8035 - val_loss: 0.4834
Epoch 2/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7944 - loss: 0.4942 - val_
accuracy: 0.8035 - val_loss: 0.4521
Epoch 3/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7937 - loss: 0.4722 - val_
accuracy: 0.8035 - val_loss: 0.4367
Epoch 4/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7934 - loss: 0.4621 - val_
accuracy: 0.8030 - val_loss: 0.4263
Epoch 5/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7994 - loss: 0.4548 - val_
accuracy: 0.8075 - val_loss: 0.4190
Epoch 6/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8048 - loss: 0.4517 - val_
accuracy: 0.8150 - val_loss: 0.4136
Epoch 7/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8121 - loss: 0.4429 - val_
accuracy: 0.8245 - val_loss: 0.4066
Epoch 8/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8115 - loss: 0.4379 - val_
accuracy: 0.8280 - val_loss: 0.4012
Epoch 9/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8169 - loss: 0.4332 - val_
accuracy: 0.8320 - val_loss: 0.3960
Epoch 10/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8190 - loss: 0.4272 - val_
accuracy: 0.8355 - val_loss: 0.3907
Epoch 11/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8229 - loss: 0.4261 - val_
accuracy: 0.8390 - val_loss: 0.3872
Epoch 12/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8210 - loss: 0.4275 - val_
accuracy: 0.8390 - val_loss: 0.3870
Epoch 13/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8303 - loss: 0.4157 - val_
accuracy: 0.8415 - val_loss: 0.3809
Epoch 14/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8265 - loss: 0.4201 - val_
accuracy: 0.8380 - val_loss: 0.3791
Epoch 15/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8278 - loss: 0.4184 - val_
accuracy: 0.8425 - val_loss: 0.3763
Epoch 16/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8241 - loss: 0.4178 - val_
accuracy: 0.8435 - val_loss: 0.3766
Epoch 17/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8299 - loss: 0.4111 - val_
accuracy: 0.8425 - val_loss: 0.3749
Epoch 18/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8300 - loss: 0.4049 - val_
accuracy: 0.8425 - val_loss: 0.3722
Epoch 19/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8286 - loss: 0.4097 - val_
accuracy: 0.8465 - val_loss: 0.3703
Epoch 20/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8314 - loss: 0.4059 - val_
accuracy: 0.8420 - val_loss: 0.3703
```

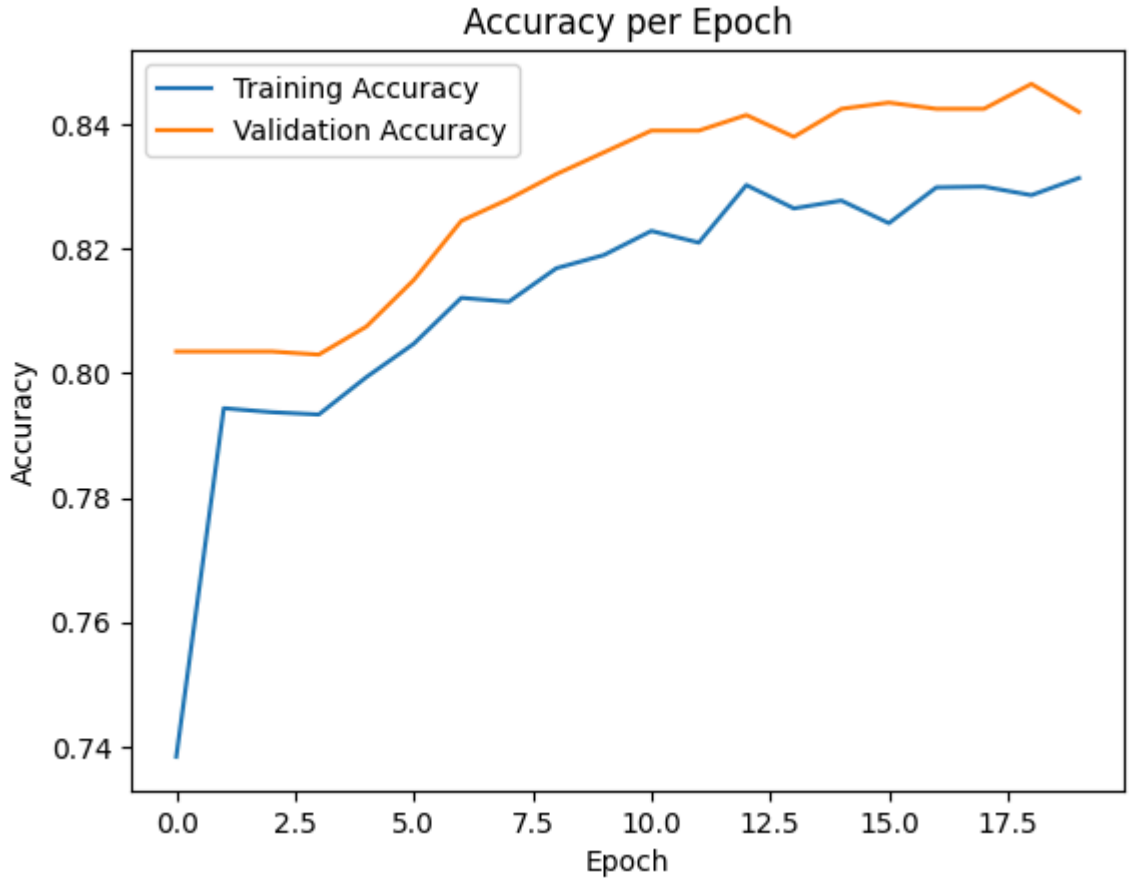
In [8]:
```
# Print accuracy per epoch
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

print("Accuracy per Epoch:\n")
for i in range(len(train_acc)):
    print(f"Epoch {i+1}: Training Accuracy = {train_acc[i]*100:.2f}%, Validation
```

```
Accuracy per Epoch:

Epoch 1: Training Accuracy = 73.84%, Validation Accuracy = 80.35%
Epoch 2: Training Accuracy = 79.44%, Validation Accuracy = 80.35%
Epoch 3: Training Accuracy = 79.37%, Validation Accuracy = 80.35%
Epoch 4: Training Accuracy = 79.34%, Validation Accuracy = 80.30%
Epoch 5: Training Accuracy = 79.94%, Validation Accuracy = 80.75%
Epoch 6: Training Accuracy = 80.48%, Validation Accuracy = 81.50%
Epoch 7: Training Accuracy = 81.21%, Validation Accuracy = 82.45%
Epoch 8: Training Accuracy = 81.15%, Validation Accuracy = 82.80%
Epoch 9: Training Accuracy = 81.69%, Validation Accuracy = 83.20%
Epoch 10: Training Accuracy = 81.90%, Validation Accuracy = 83.55%
Epoch 11: Training Accuracy = 82.29%, Validation Accuracy = 83.90%
Epoch 12: Training Accuracy = 82.10%, Validation Accuracy = 83.90%
Epoch 13: Training Accuracy = 83.03%, Validation Accuracy = 84.15%
Epoch 14: Training Accuracy = 82.65%, Validation Accuracy = 83.80%
Epoch 15: Training Accuracy = 82.78%, Validation Accuracy = 84.25%
Epoch 16: Training Accuracy = 82.41%, Validation Accuracy = 84.35%
Epoch 17: Training Accuracy = 82.99%, Validation Accuracy = 84.25%
Epoch 18: Training Accuracy = 83.00%, Validation Accuracy = 84.25%
Epoch 19: Training Accuracy = 82.86%, Validation Accuracy = 84.65%
Epoch 20: Training Accuracy = 83.14%, Validation Accuracy = 84.20%
```

In [9]:
```
import matplotlib.pyplot as plt
# Accuracy per epoch
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [15]:
```
y_pred = (model.predict(X_test) > 0.5).astype(int)

acc = accuracy_score(y_test, y_pred)
print("Final Accuracy: {:.2f}%".format(acc * 100))
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Final Accuracy: 84.20%
```

In [14]:
```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
 [[1601    6]
 [ 310   83]]
```