In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [2]:
```
df=pd.read_csv("C:/Users/daksh/OneDrive/Documents/Desktop/datasets/Churn_Modelling.csv")
```

In [3]:
```
df.head()
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]:
```
# 2. Preprocessing
# Drop unnecessary columns if present
df = df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1, errors='ignore')

# Encode categorical variables
label_encoder_geo = LabelEncoder()
label_encoder_gender = LabelEncoder()

df['Geography'] = label_encoder_geo.fit_transform(df['Geography'])
df['Gender'] = label_encoder_gender.fit_transform(df['Gender'])
```

In [5]:
```
from sklearn.preprocessing import StandardScaler

# Define X and y
X = df.drop('Exited', axis=1)
y = df['Exited']

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [6]:
```
# 3. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [7]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.20.0
```

In [8]:
```
# 4. Build Neural Network
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Binary classification

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

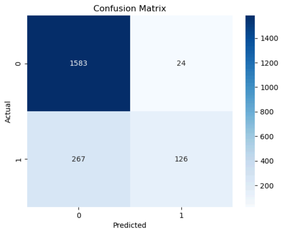
```
C:\Users\daksh\.conda\envs\tflib\site-packages\keras\src\layers\core\dense.py:95: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, pre
fer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,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

In [9]:
```
# 5. Train Model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_size=32, verbose=1)
```

```
Epoch 1/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7903 - loss: 0.5135 - val_accuracy: 0.8050 - val_loss: 0.4469
Epoch 2/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7966 - loss: 0.4661 - val_accuracy: 0.8181 - val_loss: 0.4238
Epoch 3/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8031 - loss: 0.4494 - val_accuracy: 0.8238 - val_loss: 0.4130
Epoch 4/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8098 - loss: 0.4378 - val_accuracy: 0.8331 - val_loss: 0.4047
Epoch 5/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8122 - loss: 0.4310 - val_accuracy: 0.8406 - val_loss: 0.3984
Epoch 6/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8213 - loss: 0.4244 - val_accuracy: 0.8431 - val_loss: 0.3914
Epoch 7/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8250 - loss: 0.4187 - val_accuracy: 0.8438 - val_loss: 0.3878
Epoch 8/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8273 - loss: 0.4109 - val_accuracy: 0.8438 - val_loss: 0.3829
Epoch 9/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8309 - loss: 0.4057 - val_accuracy: 0.8456 - val_loss: 0.3809
Epoch 10/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8336 - loss: 0.4025 - val_accuracy: 0.8481 - val_loss: 0.3786
Epoch 11/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8345 - loss: 0.3946 - val_accuracy: 0.8469 - val_loss: 0.3755
Epoch 12/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8369 - loss: 0.3939 - val_accuracy: 0.8500 - val_loss: 0.3728
Epoch 13/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8309 - loss: 0.3950 - val_accuracy: 0.8487 - val_loss: 0.3721
Epoch 14/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8366 - loss: 0.3905 - val_accuracy: 0.8494 - val_loss: 0.3710
Epoch 15/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8330 - loss: 0.3886 - val_accuracy: 0.8469 - val_loss: 0.3693
Epoch 16/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8327 - loss: 0.3930 - val_accuracy: 0.8481 - val_loss: 0.3698
Epoch 17/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8369 - loss: 0.3894 - val_accuracy: 0.8475 - val_loss: 0.3702
Epoch 18/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8373 - loss: 0.3886 - val_accuracy: 0.8462 - val_loss: 0.3694
Epoch 19/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8416 - loss: 0.3849 - val_accuracy: 0.8487 - val_loss: 0.3690
Epoch 20/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8430 - loss: 0.3866 - val_accuracy: 0.8487 - val_loss: 0.3676
Epoch 21/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8395 - loss: 0.3861 - val_accuracy: 0.8475 - val_loss: 0.3675
Epoch 22/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8402 - loss: 0.3856 - val_accuracy: 0.8481 - val_loss: 0.3674
Epoch 23/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8433 - loss: 0.3843 - val_accuracy: 0.8506 - val_loss: 0.3658
Epoch 24/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8445 - loss: 0.3788 - val_accuracy: 0.8506 - val_loss: 0.3663
Epoch 25/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8497 - loss: 0.3748 - val_accuracy: 0.8481 - val_loss: 0.3656
Epoch 26/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8448 - loss: 0.3769 - val_accuracy: 0.8487 - val_loss: 0.3643
Epoch 27/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8427 - loss: 0.3789 - val_accuracy: 0.8512 - val_loss: 0.3663
Epoch 28/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8411 - loss: 0.3829 - val_accuracy: 0.8519 - val_loss: 0.3653
Epoch 29/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8478 - loss: 0.3735 - val_accuracy: 0.8512 - val_loss: 0.3661
Epoch 30/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8466 - loss: 0.3778 - val_accuracy: 0.8519 - val_loss: 0.3661
Epoch 31/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8442 - loss: 0.3786 - val_accuracy: 0.8512 - val_loss: 0.3646
Epoch 32/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8489 - loss: 0.3711 - val_accuracy: 0.8512 - val_loss: 0.3646
Epoch 33/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8441 - loss: 0.3752 - val_accuracy: 0.8512 - val_loss: 0.3610
Epoch 34/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8487 - loss: 0.3682 - val_accuracy: 0.8512 - val_loss: 0.3623
Epoch 35/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8431 - loss: 0.3731 - val_accuracy: 0.8544 - val_loss: 0.3623
Epoch 36/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8489 - loss: 0.3686 - val_accuracy: 0.8531 - val_loss: 0.3627
Epoch 37/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8472 - loss: 0.3747 - val_accuracy: 0.8519 - val_loss: 0.3626
Epoch 38/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8505 - loss: 0.3732 - val_accuracy: 0.8519 - val_loss: 0.3623
Epoch 39/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8477 - loss: 0.3687 - val_accuracy: 0.8525 - val_loss: 0.3616
Epoch 40/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8497 - loss: 0.3688 - val_accuracy: 0.8512 - val_loss: 0.3623
Epoch 41/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8472 - loss: 0.3670 - val_accuracy: 0.8519 - val_loss: 0.3603
Epoch 42/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8477 - loss: 0.3683 - val_accuracy: 0.8519 - val_loss: 0.3603
Epoch 43/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8486 - loss: 0.3684 - val_accuracy: 0.8512 - val_loss: 0.3605
Epoch 44/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8508 - loss: 0.3637 - val_accuracy: 0.8525 - val_loss: 0.3589
Epoch 45/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8497 - loss: 0.3633 - val_accuracy: 0.8550 - val_loss: 0.3590
Epoch 46/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8450 - loss: 0.3696 - val_accuracy: 0.8525 - val_loss: 0.3582
Epoch 47/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8481 - loss: 0.3666 - val_accuracy: 0.8537 - val_loss: 0.3605
Epoch 48/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8494 - loss: 0.3680 - val_accuracy: 0.8531 - val_loss: 0.3606
Epoch 49/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8498 - loss: 0.3647 - val_accuracy: 0.8550 - val_loss: 0.3583
Epoch 50/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8472 - loss: 0.3694 - val_accuracy: 0.8500 - val_loss: 0.3601
```

In [11]:
```
# 6. Evaluate Model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

# Predictions
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8545 - loss: 0.3474
Test Accuracy: 0.8545
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1607
           1       0.84      0.32      0.46       393

    accuracy                           0.85      2000
   macro avg       0.85      0.65      0.69      2000
weighted avg       0.85      0.85      0.83      2000
```



In [12]:
```
# 7. Plot Training History
plt.figure(figsize=(8,5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training History')
plt.show()
```


Training History
Accuracy
Epochs
Train Accuracy
Validation Accuracy
0.85
0.84
0.83
0.82
0.81
0.80
0.79
0
10
20
30
40
50

In [ ]: