```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("/content/Churn_Modelling.csv")  # Change path if needed
print("✅ Dataset Loaded Successfully!")
print(df.head())
```

```
✅ Dataset Loaded Successfully!
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

```
# Step 2: Preprocessing
# ------------------------------------------------------
# Drop unnecessary columns
df = df.drop(["RowNumber", "CustomerId", "Surname"], axis=1)

# Encode categorical variables
le_gender = LabelEncoder()
df["Gender"] = le_gender.fit_transform(df["Gender"])  # Male=1, Female=0

# One-hot encode Geography column
df = pd.get_dummies(df, columns=["Geography"], drop_first=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-117718843.py in <cell line: 0>()
      2 # ------------------------------------------------------
      3 # Drop unnecessary columns
----> 4 df = df.drop(["RowNumber", "CustomerId", "Surname"], axis=1)
      5
      6 # Encode categorical variables

NameError: name 'df' is not defined
```

```
# Separate features and target
X = df.drop("Exited", axis=1)
y = df["Exited"]

# ------------------------------------------------------
# Step 3: Train-Test Split
# ------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
# Step 4: Feature Scaling
# ------------------------------------------------------
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Step 5: Build the ANN Model
# ------------------------------------------------------
model = Sequential()
```

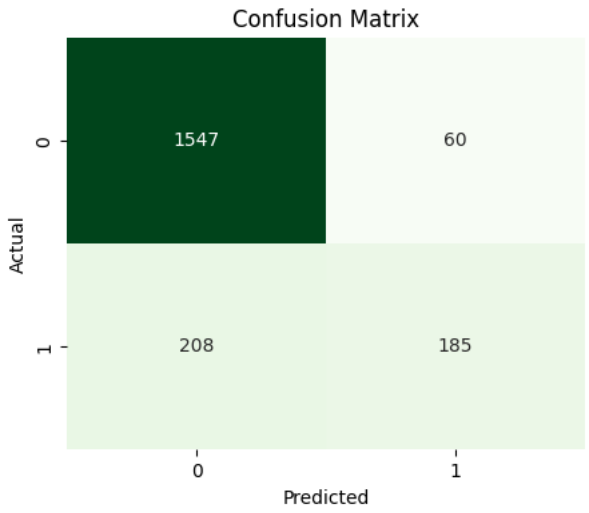
```
model.add(Dense(16, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.2))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

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

# ------------------------------------------------------
# Step 6: Train the Model
# ------------------------------------------------------
history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1, validation_split=0.2)

# ------------------------------------------------------
# Step 7: Evaluate Model Performance
# ------------------------------------------------------
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype("int32")

print("\n✅ Model Evaluation Results:")
print("Accuracy :", round(accuracy_score(y_test, y_pred), 3))
print("Classification Report:\n", classification_report(y_test, y_pred))

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

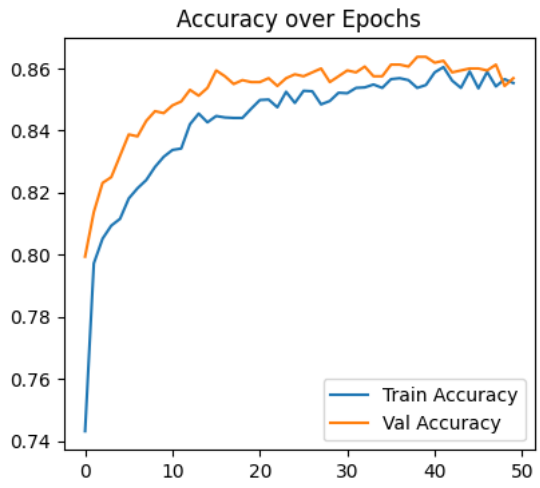
```
Epoch 1/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.6699 - loss: 0.6204 - val_accuracy: 0.7994 - val_loss: 0.4695
Epoch 2/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7976 - loss: 0.4772 - val_accuracy: 0.8138 - val_loss: 0.4385
Epoch 3/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8080 - loss: 0.4494 - val_accuracy: 0.8231 - val_loss: 0.4258
Epoch 4/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8151 - loss: 0.4403 - val_accuracy: 0.8250 - val_loss: 0.4185
Epoch 5/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8104 - loss: 0.4332 - val_accuracy: 0.8319 - val_loss: 0.4137
Epoch 6/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8207 - loss: 0.4154 - val_accuracy: 0.8388 - val_loss: 0.4071
Epoch 7/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8258 - loss: 0.4165 - val_accuracy: 0.8381 - val_loss: 0.4020
Epoch 8/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8292 - loss: 0.4052 - val_accuracy: 0.8431 - val_loss: 0.3968
Epoch 9/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8322 - loss: 0.4062 - val_accuracy: 0.8462 - val_loss: 0.3896
Epoch 10/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8314 - loss: 0.4049 - val_accuracy: 0.8456 - val_loss: 0.3834
Epoch 11/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8383 - loss: 0.3964 - val_accuracy: 0.8481 - val_loss: 0.3772
Epoch 12/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8330 - loss: 0.4025 - val_accuracy: 0.8494 - val_loss: 0.3711
Epoch 13/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8399 - loss: 0.3888 - val_accuracy: 0.8531 - val_loss: 0.3663
Epoch 14/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8397 - loss: 0.3909 - val_accuracy: 0.8512 - val_loss: 0.3628
Epoch 15/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8360 - loss: 0.3917 - val_accuracy: 0.8537 - val_loss: 0.3605
Epoch 16/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8461 - loss: 0.3869 - val_accuracy: 0.8594 - val_loss: 0.3569
Epoch 17/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8516 - loss: 0.3635 - val_accuracy: 0.8575 - val_loss: 0.3553
Epoch 18/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8481 - loss: 0.3620 - val_accuracy: 0.8550 - val_loss: 0.3532
Epoch 19/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8454 - loss: 0.3723 - val_accuracy: 0.8562 - val_loss: 0.3522
Epoch 20/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8499 - loss: 0.3602 - val_accuracy: 0.8556 - val_loss: 0.3514
Epoch 21/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8448 - loss: 0.3713 - val_accuracy: 0.8556 - val_loss: 0.3508
Epoch 22/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8484 - loss: 0.3605 - val_accuracy: 0.8569 - val_loss: 0.3509
Epoch 23/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8399 - loss: 0.3742 - val_accuracy: 0.8544 - val_loss: 0.3509
Epoch 24/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8563 - loss: 0.3516 - val_accuracy: 0.8569 - val_loss: 0.3480
Epoch 25/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8516 - loss: 0.3595 - val_accuracy: 0.8581 - val_loss: 0.3476
Epoch 26/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8500 - loss: 0.3623 - val_accuracy: 0.8575 - val_loss: 0.3465
Epoch 27/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8438 - loss: 0.3717 - val_accuracy: 0.8587 - val_loss: 0.3463
Epoch 28/50
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8483 - loss: 0.3574 - val_accuracy: 0.8600 - val_loss: 0.3467
Epoch 29/50
```

```
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

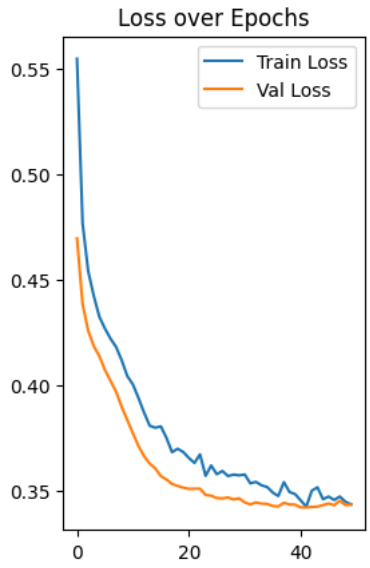



```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Accuracy over Epochs')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b380c267500>
```




```
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Loss over Epochs')
plt.legend()
plt.show()
```



Start coding or generate with AI.