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

# Preprocessing & Model Selection
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc

# Classifier
from sklearn.linear_model import LogisticRegression
# You can easily swap this for DecisionTreeClassifier or RandomForestClassifier

# --- Step 1: Load the Data ---
try:
    df = pd.read_csv("D:/Dalmia/Tri-5 2025/ML & DL/ML Internship/Churn_Modelling
    print("Data loaded successfully.")
except FileNotFoundError:
    print("Error: Make sure 'Churn_Modelling.csv' is in the same directory.")
    # Exit or handle error

print("\nInitial Data Structure:")
print(df.head())
print(df.info())
```

```
Data loaded successfully.

Initial Data Structure:
   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
<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
None
```

In [3]:

```python
# --- Step 2.1: Drop Irrelevant Columns ---
# RowNumber, CustomerId, and Surname do not contribute to churn prediction
df = df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)

# --- Step 2.2: Define Features (X) and Target (y) ---
# The target variable is 'Exited' (1 = Churned, 0 = Not Churned)
X = df.drop('Exited', axis=1)
y = df['Exited']

# --- Step 2.3: Identify Feature Types ---
# 'Geography' and 'Gender' need one-hot encoding.
# All other numerical columns need scaling.
categorical_features = ['Geography', 'Gender']
numerical_features = X.select_dtypes(include=np.number).columns.tolist()
```

```
# Remove features we dropped or the target
for col in ['Exited', 'RowNumber', 'CustomerId', 'Surname']:
    if col in numerical_features:
        numerical_features.remove(col)
```

In [5]:
```
# --- Step 3.1: Create Preprocessing Steps ---
# 1. Standard Scaler for numerical columns
numerical_transformer = StandardScaler()

# 2. One-Hot Encoder for categorical columns
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

# Combine transformers using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ],
    remainder='passthrough' # Keep any other columns if they existed
)

# --- Step 3.2: Create the Full Machine Learning Pipeline ---
# Define the model to use
model = LogisticRegression(random_state=42, solver='liblinear')

# Create the pipeline: Preprocessor -> Model
churn_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                 ('classifier', model)])

# --- Step 3.3: Split Data ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y # Stratify ensures equal ch
)

print("\nStarting Model Training...")
# Fit the pipeline to the training data
churn_pipeline.fit(X_train, y_train)
print("Training complete!")
```

```
Starting Model Training...
Training complete!
```

In [7]:
```
# --- Step 4: Make Predictions ---
y_pred = churn_pipeline.predict(X_test)
y_proba = churn_pipeline.predict_proba(X_test)[:, 1]

# --- Step 4.1: Model Performance Metrics ---
print("\n--- Model Evaluation on Test Set ---")

# 1. Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy:.4f}")

# 2. ROC-AUC Score (Area Under the Curve - crucial for imbalance)
roc_auc = roc_auc_score(y_test, y_proba)
print(f"ROC-AUC Score: {roc_auc:.4f}")

# 3. Classification Report (Precision, Recall, F1-Score)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

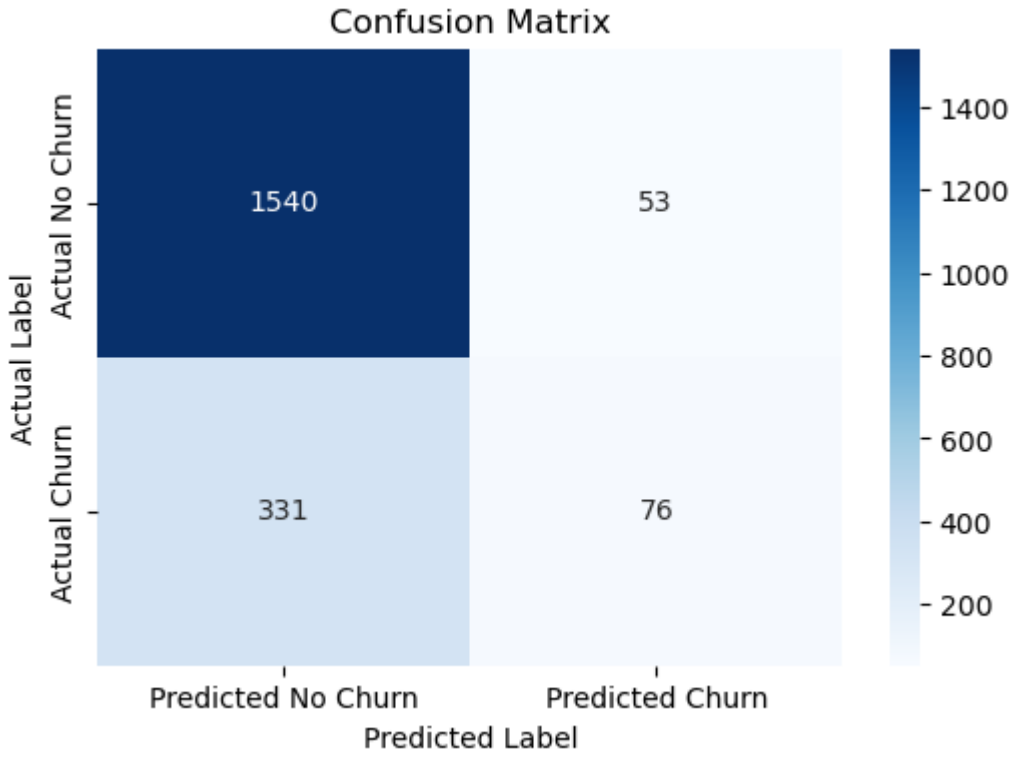
```
# --- Step 4.2: Confusion Matrix Visualization ---
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted No Churn', 'Predicted Churn'],
            yticklabels=['Actual No Churn', 'Actual Churn'])
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
--- Model Evaluation on Test Set ---
Accuracy Score: 0.8080
ROC-AUC Score: 0.7748

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      1593
           1       0.59      0.19      0.28       407

    accuracy                           0.81      2000
   macro avg       0.71      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000
```



In [ ]: