Churn Model Task

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
import joblib
```

In [3]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
dataset = pd.read_csv(r"C:\Users\JANHAVI\Desktop\Churn_Modelling.csv")


# List of columns to drop
columns_to_drop = ['RowNumber', 'CustomerId', 'Surname']

# Drop the specified columns
dataset = dataset.drop(columns=columns_to_drop)




X = dataset.drop('Exited', axis=1)
y = dataset['Exited']
```

In [6]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CreditScore      10000 non-null  int64  
 1   Geography        10000 non-null  object 
 2   Gender           10000 non-null  object 
 3   Age              10000 non-null  int64  
 4   Tenure           10000 non-null  int64  
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64  
 7   HasCrCard        10000 non-null  int64  
 8   IsActiveMember   10000 non-null  int64  
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64  
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

Data Preprocessing

In [7]:
```
# Numerical features
numerical_features = ['CreditScore', 'Age', 'Tenure', 'Balance',
                      'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSal
numerical_data = dataset[numerical_features]

# Apply scaling to numerical features
scaler = StandardScaler()
numerical_data_scaled = scaler.fit_transform(numerical_data)
```

In [8]:
```
# Categorical features
categorical_features = ['Geography', 'Gender']
categorical_data = dataset[categorical_features]

# Apply one-hot encoding to categorical features
encoder = OneHotEncoder(handle_unknown='ignore')
categorical_data_encoded = encoder.fit_transform(categorical_data).toarray()
```

In [9]:
```
# Combine the preprocessed numerical and categorical data
processed_data = np.hstack([numerical_data_scaled, categorical_data_encoded])

# Get the column names for encoded categorical features
encoded_feature_names = encoder.get_feature_names_out(categorical_features)

# Create a final dataframe
final_df = pd.DataFrame(processed_data, columns=numerical_features + list(encoded_f

# Display the final dataframe
final_df
```

Out[9]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	-0.326221	0.293517	-1.041760	-1.225848	-0.911583	0.646092	0.970243
1	-0.440036	0.198164	-1.387538	0.117350	-0.911583	-1.547768	0.970243
2	-1.536794	0.293517	1.032908	1.333053	2.527057	0.646092	-1.030670
3	0.501521	0.007457	-1.387538	-1.225848	0.807737	-1.547768	-1.030670
4	2.063884	0.388871	-1.041760	0.785728	-0.911583	0.646092	0.970243
...	...	...	...	...	...	...	...
9995	1.246488	0.007457	-0.004426	-1.225848	0.807737	0.646092	-1.030670
9996	-1.391939	-0.373958	1.724464	-0.306379	-0.911583	0.646092	0.970243
9997	0.604988	-0.278604	0.687130	-1.225848	-0.911583	-1.547768	0.970243
9998	1.256835	0.293517	-0.695982	-0.022608	0.807737	0.646092	-1.030670
9999	1.463771	-1.041433	-0.350204	0.859965	-0.911583	0.646092	-1.030670

10000 rows × 13 columns

Classification Models

In [10]:
```
models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'SVC': SVC(),
```

```
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
}
```

Evaluate Models

In [22]:
```
# Evaluate models
results = {}
for name, model in models.items():
    scores = cross_val_score(model, processed_data, y, cv=5, scoring='accuracy')
    results[name] = scores
    print(f'{name}: {scores.mean():.4f} (+/- {scores.std():.4f})')
```

```
Logistic Regression: 0.8097 (+/- 0.0050)
Decision Tree: 0.7901 (+/- 0.0073)
SVC: 0.8561 (+/- 0.0062)
Random Forest: 0.8611 (+/- 0.0052)
Gradient Boosting: 0.8642 (+/- 0.0069)
XGBoost: 0.8576 (+/- 0.0026)
```

In [24]:
```
# Identify the best model
best_model_name = max(results, key=lambda name: results[name].mean())
print(f"\nBest Model: {best_model_name}")
```

```
Best Model: Gradient Boosting
```

In [25]:
```
# Train the best model on the entire dataset
best_model = models[best_model_name]
best_model.fit(processed_data, y)
```

Out[25]:

▾ GradientBoostingClassifier

▶ Parameters

In [26]:
```
# Save the trained model (Optional)
import joblib
joblib.dump(best_model, 'best_model.pkl')

print(f"The best model ({best_model_name}) has been trained and saved as 'best_mode
```

```
The best model (Gradient Boosting) has been trained and saved as 'best_model.pkl'.
```

In [27]:
```
# Load the saved model (to verify or use later)
loaded_model = joblib.load('best_model.pkl')
print(f"Model loaded successfully: {loaded_model}")

# Make predictions using the loaded model
sample_data = processed_data[:5]  # Replace with new or unseen preprocessed data
predictions = loaded_model.predict(sample_data)
print(f"Predictions: {predictions}")
```

```
Model loaded successfully: GradientBoostingClassifier()
Predictions: [0 0 1 0 0]
```

In [ ]: