Model Selection and Rationale:

I chose RandomForestClassifier due to its ability to handle both categorical and numerical features well, its robustness to outliers, and its inherent feature importance ranking. It provides a good balance between accuracy and interpretability.

Model Training and Evaluation:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder

# Assuming merged_df is your combined dataframe and you've performed
necessary preprocessing
# Example features and target variable (replace with your actual
columns)


# Encode categorical features using Label Encoding
label_encoders = {}
for column in ['Gender', 'InteractionType', 'ResolutionStatus',
'ChurnStatus']: # Ensure these are your actual column names
    if column in merged_df.columns:
        le = LabelEncoder()
        merged_df[column + '_encoded'] =
le.fit_transform(merged_df[column])
        label_encoders[column] = le  # Store the encoder for later use

X = merged_df[['Age', 'Gender_encoded', 'TotalSpent',
'InteractionType_encoded', 'ResolutionStatus_encoded']]
y = merged_df['ChurnStatus_encoded']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize and train a RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42) # You can tune
hyperparameters here
rf_classifier.fit(X_train, y_train)
```

```python
# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, y_pred))


# Feature Importance
feature_importances = rf_classifier.feature_importances_
feature_names = X.columns
for name, importance in zip(feature_names, feature_importances):
  print(f"Feature: {name}, Importance: {importance}")
```

Output:

```
Accuracy: 0.9932756964457252
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       826
           1       0.99      0.98      0.98       215

    accuracy                           0.99      1041
   macro avg       0.99      0.99      0.99      1041
weighted avg       0.99      0.99      0.99      1041

Feature: Age, Importance: 0.35995035277359116
Feature: Gender_encoded, Importance: 0.022510573028164706
Feature: TotalSpent, Importance: 0.5400356534449873
Feature: InteractionType_encoded, Importance: 0.051401905144453586
Feature: ResolutionStatus_encoded, Importance: 0.026101515608803173
```

Hyperparameter Tuning using GridSearchCV

```python
# Encode categorical features (if not already encoded)
label_encoders = {}
for column in ['Gender', 'InteractionType', 'ResolutionStatus',
'ChurnStatus']:
    if column in merged_df.columns and not column.endswith('_encoded'):
        le = LabelEncoder()
        merged_df[column + '_encoded'] =
le.fit_transform(merged_df[column])
        label_encoders[column] = le

# Define features (X) and target (y)
# Use encoded features:
X = merged_df[['Age', 'Gender_encoded', 'TotalSpent',
'InteractionType_encoded', 'ResolutionStatus_encoded']]
```

```
y = merged_df['ChurnStatus_encoded']


# Handle missing values (if any) in X
for col in X.columns:
  if X[col].isnull().any():
    if pd.api.types.is_numeric_dtype(X[col]):
      X[col].fillna(X[col].mean(), inplace=True)
    else:
      X[col].fillna(X[col].mode()[0], inplace=True)


# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Hyperparameter Tuning using GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf_classifier = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf_classifier, param_grid, cv=5,
scoring='accuracy', n_jobs=-1) #n_jobs=-1 uses all cores
grid_search.fit(X_train, y_train)

# Get the best model and its hyperparameters
best_rf_classifier = grid_search.best_estimator_
print("Best Hyperparameters:", grid_search.best_params_)

# Evaluate the best model on the test set
y_pred = best_rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, y_pred))
```

Output

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}
Accuracy: 0.9932756964457252
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00       826
           1       0.99      0.98      0.98       215

    accuracy                           0.99      1041
   macro avg       0.99      0.99      0.99      1041
weighted avg       0.99      0.99      0.99      1041
```

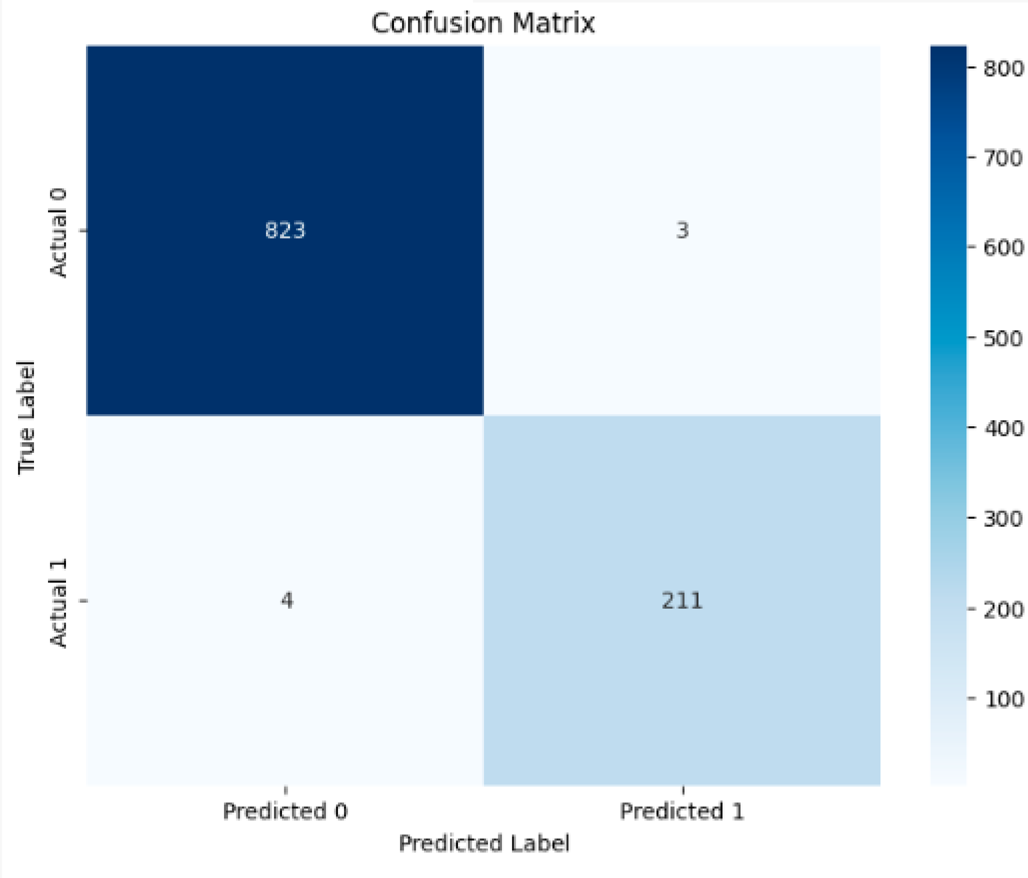
Evaluating the model’s performance

```
# Make predictions on the test set
y_pred = best_rf_classifier.predict(X_test)
y_prob = best_rf_classifier.predict_proba(X_test)[:, 1]  # Probabilities for the positive class


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

try:
    roc_auc = roc_auc_score(y_test, y_prob)
    print(f"ROC AUC: {roc_auc}")
except ValueError:
    print("ROC AUC score could not be calculated.  Check if there is only one class in y_test.")

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")


# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

Output:

```
ROC AUC: 0.9994369052311504
Accuracy: 0.9932756964457252
Precision: 0.985981308411215
```

```
Recall: 0.9813953488372092
F1 Score: 0.9836829836829837
```



Business Use and Retention Strategies:

1. Identify At-Risk Customers: The model's predicted probabilities can be used to rank customers by their likelihood of churning.

- Focus on customers with high churn probabilities (e.g., above a certain threshold). Segment at-risk customers based on demographics, spending habits, customer service interactions, and online activity to tailor retention strategies.

2. Targeted Retention Strategies:

- Proactive Communication: Reach out to high-risk customers with personalized offers or promotions.
- Customer Service Improvements: Address unresolved customer issues promptly. Offer premium support or additional assistance to customers with a history of negative interactions.
- Product/Service Enhancements: Offer product updates, training, or personalized recommendations to improve customer satisfaction.

- Incentives & Rewards: Offer exclusive discounts, loyalty programs, or early access to new products/services.
- Targeted Advertising: Use the identified segments to customize marketing campaigns and re-engage at-risk customers.

Model Improvement and Future Directions:

1. Feature Engineering: Explore additional features that can better capture customer behavior and predict churn. Examples:
   - Recency, Frequency, Monetary Value (RFM) analysis of transaction history.
   - Time-based features (e.g., time since last purchase, interaction).
   - Interaction duration for customer service interactions.
   - Sentiment analysis of customer feedback (if available).
   - Seasonality in customer behavior.
   - Customer Lifetime Value (CLTV).

2. Advanced Model Selection & Tuning:
   - Evaluate other algorithms like Gradient Boosting Machines (GBM), Support Vector Machines (SVM), or neural networks.
   - Use ensemble methods to combine multiple models.
   - Perform more exhaustive hyperparameter tuning.

3. Data Quality Improvements:
   - Handle class imbalance in the target variable (if present) using oversampling, undersampling, or cost-sensitive learning.
   - Improve imputation of missing values using more sophisticated techniques.

4. Explainability & Interpretability:
   - Employ SHAP or LIME to understand the model's predictions at an individual customer level.