In [2]:
```python
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import lightgbm as lgb
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_extraction.text import TfidfVectorizer
from textblob import TextBlob
from sklearn.impute import SimpleImputer

# Custom transformer to extract sentiment polarity from feedback text
class FeedbackSentimentExtractor(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return np.array([TextBlob(str(text)).sentiment.polarity if pd.notna(text) e

# Example dataset load
df = pd.read_csv('modify_service_df.csv')

# Target: Whether customer needs urgent reminder
# Ideally based on actual historical labels (response to reminder or actual service
# For demo, simulate using criteria as in your code
df['service_urgent'] = np.where(
    (df['next_service_due_days'] <= 120) |
    (df['feedback_score'] <= 2) |
    (df['customer_feedback'].isin(['Poor Service', 'Unresponsive', 'Delayed Pickup'
    1, 0
)

# Define features: structured + raw feedback text
numeric_features = [
    'feedback_score',
    'last_service_cost',
    'days_since_last_service',
    'next_service_due_days',
    'age_of_vehicle',
    'odometer_reading'
]
categorical_features = [
    'customer_type',
    'AMC_status'
]
text_feature = 'customer_feedback'  # raw text

# Pipelines
numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
text_transformer = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=100)),  # convert text into TF-IDF featu
])
```

```
# Combine text sentiment as extra feature
class TextFeatureAdder(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return np.array([TextBlob(str(text)).sentiment.polarity if pd.notna(text) e

# Final preprocessing
preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features),
    ('text_tfidf', TfidfVectorizer(max_features=100), text_feature)
])

# Prepare X and y
X = df[numeric_features + categorical_features + [text_feature]]
y = df['service_urgent']

# Split train-test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Build pipeline with feature processor + classifier
model = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42, n_estimators=100))
])

# Train
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

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

# Predict urgency for all
df['predicted_urgency'] = model.predict(X)

# Segment customers dynamically (could improve with clustering or learned threshold
def segment(row):
    if row['feedback_score'] <= 2 or row['customer_feedback'] in ['Poor Service', '
        return 'Critical'
    elif row['predicted_urgency'] == 1 and row['next_service_due_days'] <= 60:
        return 'High Priority'
    elif row['predicted_urgency'] == 1:
        return 'Medium Priority'
    else:
        return 'Low Priority'

df['customer_segment'] = df.apply(segment, axis=1)

# Generate personalized messages (templates or integrate with e.g. GPT for dynamic
def gen_message(row):
    base = f"Dear {row['customer_type']} Customer,\n"
    if row['customer_segment'] == 'Critical':
        base += ("We are sorry for any inconvenience caused and will personally mor
                 "Enjoy a 15% discount as our apology.\n")
    elif row['customer_segment'] == 'High Priority':
        base += "Your vehicle requires servicing soon. Book now for priority schedu
    elif row['customer_segment'] == 'Medium Priority':
        base += "Keep your vehicle in top shape by servicing it soon.\n"
    else:
        base += "Thank you for being a valued customer.\n"
    base += f"Service due in {row['next_service_due_days']} days.\n"
    return base
```

```
df['personalized_message'] = df.apply(gen_message, axis=1)

# Communication channel prediction (you can train a separate model on historical ch
# For demo, simple rule-based:
def comms_protocol(row):
    if row['customer_segment'] == 'Critical':
        return ['Phone', 'WhatsApp', 'Email']
    elif row['customer_segment'] == 'High Priority':
        return ['WhatsApp', 'Email']
    else:
        return ['Email', 'SMS']

df['preferred_channels'] = df.apply(comms_protocol, axis=1)

# Save or export result
output_cols = ['location', 'customer_type', 'make', 'model', 'year_of_purchase',
               'customer_feedback', 'feedback_score', 'next_service_due_days',
               'customer_segment', 'personalized_message', 'preferred_channels']

reminder_list = df[df['predicted_urgency'] == 1][output_cols]
reminder_list.to_csv('ai_based_service_reminder.csv', index=False)
print(f"Saved {len(reminder_list)} reminders.")

# Sample stats
print("\nCustomer segments distribution:")
print(reminder_list['customer_segment'].value_counts())

# Sample communication channel usage
channels = reminder_list['preferred_channels'].explode()
print("\nPreferred communication channels distribution:")
print(channels.value_counts())
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        54
           1       1.00      1.00      1.00       146

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200

Saved 676 reminders.

Customer segments distribution:
Critical           622
High Priority       41
Medium Priority     13
Name: customer_segment, dtype: int64

Preferred communication channels distribution:
Email       676
WhatsApp    663
Phone       622
SMS          13
Name: preferred_channels, dtype: int64
```

In [3]:

```
import joblib

# Save model
joblib.dump(model, 'Class_service_reminder_model5.pkl')
print("Model saved as 'Class_service_reminder_model5.pkl'")
```

```
# Later, you can load it back as:
# loaded_model = joblib.load('Class_service_reminder_model.pkl')
```

```
Model saved as 'Class_service_reminder_model5.pkl'
```

In [ ]: