In [6]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

class ServicePredictor:
    def __init__(self):
        self.model = self._build_model()
        self.customer_data = None
        self.predictions = None

    def _build_model(self):
        """Build the classification model pipeline"""
        numeric_features = [
            'age_of_vehicle',
            'odometer_reading',
            'last_service_kms',
            'avg_kms_per_month',
            'last_service_cost',
            'days_since_last_service',
            'number_of_services'
        ]

        categorical_features = [
            'warranty_status',
            'insurance_status',
            'fuel_type',
            'transmission',
            'customer_type',
            'customer_feedback',
            'AMC_status'
        ]

        preprocessor = ColumnTransformer(
            transformers=[
                ('num', StandardScaler(), numeric_features),
                ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_feature
            ])

        return Pipeline(steps=[
            ('preprocessor', preprocessor),
            ('classifier', RandomForestClassifier(random_state=42))
        ])

    def load_data(self, filepath):
        """Load and prepare the customer data"""
        df = pd.read_csv(filepath)

        # Create target variable (service needed within 120 days)
        df['service_needed_soon'] = df['next_service_due_days'].apply(
            lambda x: 1 if x <= 120 else 0)

        self.customer_data = df
        return df

    def train_model(self, test_size=0.2):
        """Train the model on the loaded data"""
        if self.customer_data is None:
```

```
            raise ValueError("No data loaded. Call load_data() first.")

        features = [
            'age_of_vehicle',
            'odometer_reading',
            'last_service_kms',
            'avg_kms_per_month',
            'last_service_cost',
            'days_since_last_service',
            'number_of_services',
            'warranty_status',
            'insurance_status',
            'fuel_type',
            'transmission',
            'customer_type',
            'customer_feedback',
            'AMC_status'
        ]

        target = 'service_needed_soon'

        X = self.customer_data[features]
        y = self.customer_data[target]

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=42)

        self.model.fit(X_train, y_train)

        # Evaluate model
        y_pred = self.model.predict(X_test)
        print("Model Evaluation:")
        print(classification_report(y_test, y_pred))

        return self.model

    def predict_service_needs(self):
        """Make predictions for all customers"""
        if self.model is None or not hasattr(self.model, 'fit'):
            raise ValueError("Model not trained. Call train_model() first.")

        features = [
            'age_of_vehicle',
            'odometer_reading',
            'last_service_kms',
            'avg_kms_per_month',
            'last_service_cost',
            'days_since_last_service',
            'number_of_services',
            'warranty_status',
            'insurance_status',
            'fuel_type',
            'transmission',
            'customer_type',
            'customer_feedback',
            'AMC_status'
        ]

        X = self.customer_data[features]
        self.predictions = self.model.predict(X)

        # Add predictions to customer data
        self.customer_data['predicted_service_need'] = self.predictions
        self.customer_data['confidence'] = np.max(
```

```
            self.model.predict_proba(X), axis=1)

        return self.customer_data

    def generate_reminder_list(self, min_confidence=0.7):
        """Generate list of customers needing reminders"""
        if self.predictions is None:
            self.predict_service_needs()

        reminder_list = self.customer_data[
            (self.customer_data['predicted_service_need'] == 1) &
            (self.customer_data['confidence'] >= min_confidence)
        ].copy()

        # Add reminder details
        reminder_list['reminder_message'] = reminder_list.apply(
            self._create_message, axis=1)
        reminder_list['channels'] = reminder_list.apply(
            self._determine_channels, axis=1)

        return reminder_list[
            ['location', 'customer_type', 'make', 'model',
             'year_of_purchase', 'next_service_due_days',
             'customer_feedback', 'feedback_score',
             'reminder_message', 'channels', 'telecaller_name']
        ]

    def _create_message(self, row):
        """Create personalized reminder message"""
        urgency = "soon" if row['next_service_due_days'] > 30 else "urgently"

        message = (
            f"Dear {row['customer_type']} Customer,\n\n"
            f"Our records indicate your {row['make']} {row['model']} "
            f"(purchased in {row['year_of_purchase']}) needs service {urgency}. "
            f"Recommended service in {row['next_service_due_days']} days.\n\n"
        )

        if row['feedback_score'] < 3:
            message += (
                "We noticed your previous feedback and want to ensure "
                "a better experience this time.\n\n"
            )

        if pd.notna(row['eligible_offer_code']):
            message += (
                f"Special offer: {row['offer_description']} "
                f"(Code: {row['eligible_offer_code']}).\n\n"
            )

        message += (
            "Please contact your service advisor to schedule an appointment.\n\n"
            "Best regards,\nYour Service Team"
        )

        return message

    def _determine_channels(self, row):
        """Determine which channels to use for each customer"""
        channels = []

        # Check customer's preferred language for channel selection
        if row['preferred_language'] in ['English', 'Hindi']:
            if row['sent_Whats'] == 'Yes':
```

```
                channels.append('WhatsApp')
            if row['sent_email'] == 'Yes':
                channels.append('Email')

        # Always include SMS as fallback
        channels.append('SMS')

        return list(set(channels))  # Remove duplicates

    def save_reminders(self, filename='service_reminders.csv'):
        """Save reminders to CSV for manual review and sending"""
        reminder_list = self.generate_reminder_list()
        reminder_list.to_csv(filename, index=False)
        print(f"Reminder list saved to {filename}")
        return reminder_list


# Example Usage
if __name__ == "__main__":
    # Initialize the predictor
    predictor = ServicePredictor()

    # Load your customer data
    predictor.load_data('modify_service_df.csv')

    # Train the model (AI learns patterns)
    predictor.train_model()

    # Generate predictions (AI identifies who needs service)
    predictions = predictor.predict_service_needs()

    # Create and save reminder list (human reviews before sending)
    reminders = predictor.save_reminders()

    print("\nSample reminder:")
    print(reminders.iloc[0]['reminder_message'])
    print(f"\nChannels: {', '.join(reminders.iloc[0]['channels'])}")
```

```
Model Evaluation:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       178
           1       1.00      0.59      0.74        22

    accuracy                           0.95       200
   macro avg       0.98      0.80      0.86       200
weighted avg       0.96      0.95      0.95       200

Reminder list saved to service_reminders.csv

Sample reminder:
Dear Retail Customer,

Our records indicate your Ford Figo (purchased in 2016) needs service urgently. Re
commended service in -86 days.

We noticed your previous feedback and want to ensure a better experience this tim
e.

Special offer: Free Wash (Code: OFF992).

Please contact your service advisor to schedule an appointment.

Best regards,
Your Service Team

Channels: Email, SMS, WhatsApp
```

In [7]:
```
import joblib

# Save trained model pipeline to file
joblib.dump(ServicePredictor, 'Class_service_reminder_model2.pkl')
```

Out[7]:
```
['Class_service_reminder_model2.pkl']
```

In [ ]:
```

```