Designed to predict whether a customer's vehicle **needs a service soon**, specifically within the next 120 days. This is a **binary classification problem with Service center recommendation** where the target variable (`service_needed_soon`) is:

- `1` if the vehicle is due for service within 120 days
- `0` otherwise

Prediction Target:

The key predicted characteristic is **"service needed within 120 days"**.

Features Used for Prediction:

The model uses a combination of **vehicle usage, maintenance history, and customer details** to make this prediction.

Specifically, it uses the following features as input:

Numeric Features (continuous variables):

- `age_of_vehicle` — How old the vehicle is (in years)
- `odometer_reading` — Total kilometers/miles driven
- `last_service_kms` — Kilometers driven since last service
- `avg_kms_per_month` — Average kilometers driven per month
- `last_service_cost` — Cost incurred during last service
- `days_since_last_service` — Number of days passed since last service
- `number_of_services` — How many times the vehicle has been serviced so far

Categorical Features (qualitative attributes):

- `warranty_status` — Whether the vehicle is under warranty or expired
- `insurance_status` — Status of the vehicle’s insurance (active/expired)
- `fuel_type` — Petrol, Diesel, etc.
- `transmission` — Manual or Automatic
- `customer_type` — Retail or Fleet customer
- `customer_feedback` — Feedback category from customer like 'Good', 'Poor Service', etc.
- `AMC_status` — Whether the customer has an Annual Maintenance Contract or not

Summary:

The model predicts **service urgency** based on:

- **Vehicle age and usage metrics**
- **Service history and costs**
- **Customer and vehicle attributes**

By training on these features with historical data labeled by whether service was needed or not, the system learns to predict the likelihood of an upcoming service need, so it can remind customers proactively.

In [1]:
```
import pandas as pd
import numpy as np
from geopy.distance import geodesic
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report


class EnhancedServicePredictor:
    def __init__(self):
        self.model = self._build_model()
        self.customer_data = None
        self.predictions = None
        self.saas_name = "AutoMoto AI"
        self.dealership_data = self.create_premium_dealerships()
        self.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'
        ]

    def create_premium_dealerships(self):
        dealership_patterns = {
            'Toyota': {
                'Anna Nagar': 'Lanson Toyota - Anna Nagar',
                'T Nagar': 'Harsha Toyota - T Nagar',
                'OMR': 'Harsha Toyota - OMR',
                'Velachery': 'Lanson Toyota - Velachery',
                'Adyar': 'Harsha Toyota - Adyar',
                'Porur': 'Harsha Toyota - Porur',
                'Ambattur': 'Lanson Toyota - Ambattur'
            },
            'Hyundai': {
                'Anna Nagar': 'Kun Hyundai - Anna Nagar',
                'T Nagar': 'Kun Hyundai - T Nagar',
                'OMR': 'V3 Hyundai - OMR',
                'Velachery': 'V3 Hyundai - Velachery',
                'Adyar': 'Kun Hyundai - Adyar',
                'Porur': 'V3 Hyundai - Porur',
                'Ambattur': 'V3 Hyundai - Ambattur'
            },
            'Honda': {
                'Anna Nagar': 'Olympia Honda - Anna Nagar',
                'T Nagar': 'Sundaram Honda - T Nagar',
                'OMR': 'Olympia Honda - OMR',
                'Velachery': 'Capital Honda - Velachery',
                'Adyar': 'Sundaram Honda - Adyar',
                'Porur': 'Olympia Honda - Porur',
                'Ambattur': 'Capital Honda - Ambattur'
            },
            'Ford': {
                'Anna Nagar': 'MPL Ford - Anna Nagar (Service)',
                'T Nagar': 'Chennai Ford - T Nagar (Service)',
                'OMR': 'MPL Ford - OMR (Service)',
                'Velachery': 'Chennai Ford - Velachery (Service)',
                'Adyar': 'MPL Ford - Adyar (Service)',
                'Porur': 'Chennai Ford - Porur (Service)',
                'Ambattur': 'MPL Ford - Ambattur (Service)'
            },
            'Maruti': {
                'Anna Nagar': 'CARS India - Anna Nagar',
                'T Nagar': 'Popular Maruti - T Nagar',
                'OMR': 'Popular Maruti - OMR',
                'Velachery': 'ABT Maruti - Velachery',
                'Adyar': 'Popular Maruti - Adyar',
                'Porur': 'ABT Maruti - Porur',
                'Ambattur': 'ABT Maruti - Ambattur'
            },
            'Tata': {
                'Anna Nagar': 'TAFE Reach Tata - Anna Nagar',
                'T Nagar': 'Sree Gokulam Motors - T Nagar',
                'OMR': 'TAFE Reach Tata - OMR',
                'Velachery': 'TAFE Reach Tata - Velachery',
                'Adyar': 'Sree Gokulam Motors - Adyar',
                'Porur': 'Sree Gokulam Motors - Porur',
                'Ambattur': 'TAFE Reach Tata - Ambattur'
            },
            'Volkswagen': {
                'Anna Nagar': 'KUN Volkswagen - Anna Nagar',
                'T Nagar': 'KUN Volkswagen - T Nagar',
                'OMR': 'Volkswagen Mount Road - OMR',
                'Velachery': 'KUN Volkswagen - Velachery',
                'Adyar': 'KUN Volkswagen - Adyar',
                'Porur': 'KUN Volkswagen - Porur',
                'Ambattur': 'KUN Volkswagen - Ambattur'
            }
        }

        locations = {
            'Anna Nagar': (13.0878, 80.2119),
            'T Nagar': (13.0478, 80.2427),
            'OMR': (12.9716, 80.2497),
            'Velachery': (12.9850, 80.2165),
            'Adyar': (13.0067, 80.2566),
            'Porur': (13.0390, 80.1619),
            'Ambattur': (13.1147, 80.1548)
        }

        dealerships = []
        for brand, area_patterns in dealership_patterns.items():
            for area, coords in locations.items():
                is_saas_partner = np.random.choice([True, False], p=[0.7, 0.3])
                dealerships.append({
                    'brand': brand,
                    'dealership_name': area_patterns.get(area, f"{brand} Premium {a
                    'location': area,
                    'latitude': coords[0],
                    'longitude': coords[1],
                    'contact_number': self._generate_contact_number(area),
                    'service_advisor': self._generate_service_advisor(),
                    'is_premium': True,
                    'working_hours': self._generate_working_hours(brand, area),
                    'is_saas_partner': is_saas_partner
                })

        return pd.DataFrame(dealerships)

    def _generate_contact_number(self, area):
        area_prefixes = {
            'Anna Nagar': '2620',
            'T Nagar': '2434',
            'OMR': '2498',
            'Velachery': '2246',
            'Adyar': '2442',
            'Porur': '2479',
            'Ambattur': '2657'
        }
        prefix = area_prefixes.get(area, '2445')
        return f"+91 44 {prefix} {np.random.randint(1000, 9999):04d}"

    def _generate_service_advisor(self):
        first_names = ['Kumar', 'Rajesh', 'Arun', 'Vijay', 'Senthil',
                       'Prakash', 'Manoj', 'Ramesh', 'Ganesan', 'Dinesh']
        last_names = ['Iyer', 'Pillai', 'Nair', 'Menon', 'Reddy',
                      'Naidu', 'Gowda', 'Shetty', 'Patel', 'Sharma']
        return f"Mr. {np.random.choice(first_names)} {np.random.choice(last_names)}

    def _generate_working_hours(self, brand, area):
        base_hours = "9:00 AM - 6:00 PM"
        if brand == 'Maruti' and area in ['OMR', 'Porur']:
            return "8:30 AM - 7:00 PM (Extended hours)"
        elif brand == 'Hyundai' and area == 'Velachery':
            return "8:00 AM - 8:00 PM (24/7 Service Available)"
        elif brand == 'Ford':
            return "10:00 AM - 5:00 PM (Service Only)"
        elif area in ['T Nagar', 'Anna Nagar']:
            return "9:30 AM - 7:30 PM (Weekend Special)"
        return base_hours

    def _build_model(self):
        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):
        df = pd.read_csv(filepath)
        df['service_needed_soon'] = df['next_service_due_days'].apply(
            lambda x: 1 if x <= 120 else 0)

        chennai_locations = {
            'Anna Nagar': (13.0878, 80.2119),
            'T Nagar': (13.0478, 80.2427),
            'OMR': (12.9716, 80.2497),
            'Velachery': (12.9850, 80.2165),
            'Adyar': (13.0067, 80.2566),
            'Porur': (13.0390, 80.1619),
            'Ambattur': (13.1147, 80.1548)
        }
        df['latitude'] = df['location'].map(lambda x: chennai_locations.get(x, (13.
        df['longitude'] = df['location'].map(lambda x: chennai_locations.get(x, (13

        self.customer_data = df
        return df

    def assign_offers_based_on_feedback(self):
        """Assign offer descriptions and codes dynamically based on customer feedba
        def offer_info(feedback):
            if feedback == 'Poor Service':
                return ("We value your feedback. Enjoy 15% OFF on your next service
            elif feedback == 'Good':
                return ("Thank you for your loyalty! Get a complimentary vehicle ch
            elif feedback == 'Excellent':
                return ("Exclusive offer! 20% discount on parts replacement on your
            else:
                return ("Get 10% OFF on your next service for being a valued custom

        offers = self.customer_data['customer_feedback'].apply(offer_info)
        self.customer_data['offer_description'] = offers.apply(lambda x: x[0])
        self.customer_data['eligible_offer_code'] = offers.apply(lambda x: x[1])

    def train_model(self, test_size=0.2):
        if self.customer_data is None:
            raise ValueError("No data loaded. Call load_data() first.")
        X = self.customer_data[self.features]
        y = self.customer_data['service_needed_soon']
        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)
        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):
        if not hasattr(self.model, 'predict'):
            raise ValueError("Model not trained. Call train_model() first.")
        X = self.customer_data[self.features]
        self.predictions = self.model.predict(X)
        self.customer_data['predicted_service_need'] = self.predictions
        self.customer_data['confidence'] = np.max(self.model.predict_proba(X), axis
        return self.customer_data

    def find_nearest_dealership(self, customer_row, only_saas=True):
        customer_coords = (customer_row['latitude'], customer_row['longitude'])
        brand = customer_row['make']
        brand_dealers = self.dealership_data[
            self.dealership_data['brand'].str.lower() == brand.lower()
        ]
        if only_saas:
            brand_dealers = brand_dealers[brand_dealers['is_saas_partner'] == True]
        if brand_dealers.empty:
            if only_saas:
                return self.find_nearest_dealership(customer_row, only_saas=False)
            else:
                return {}
        brand_dealers = brand_dealers.copy()
        brand_dealers['distance_km'] = brand_dealers.apply(
            lambda dealer: geodesic(customer_coords, (dealer['latitude'], dealer['l
            axis=1)
        nearest = brand_dealers.nsmallest(1, 'distance_km').iloc[0]
        return nearest.to_dict()

    def generate_reminder_list(self, min_confidence=0.7):
        if self.predictions is None:
            self.predict_service_needs()
        self.assign_offers_based_on_feedback()

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

        if reminder_list.empty:
            print("No customers meet the criteria for reminders.")
            return reminder_list

        dealership_info = reminder_list.apply(lambda row: self.find_nearest_dealers
        dealership_info = dealership_info.apply(lambda x: x if x is not None else {
        dealer_df = pd.DataFrame(dealership_info.tolist()).add_prefix('dealer_')
        reminder_list = pd.concat([reminder_list.reset_index(drop=True), dealer_df.

        reminder_list['saas_message'] = reminder_list.apply(self._create_saas_messa
        reminder_list['dealership_message'] = reminder_list.apply(self._create_deal
        return reminder_list

    def _create_saas_message(self, row):
        if not row.get('dealer_is_saas_partner', False):
            return f"No {self.saas_name} subscribed dealership found for this vehic

        message = (
            f"Dear {row['customer_type']} Customer,\n\n"
            f"Our {self.saas_name} Smart Service System recommends service for your
            f"within {row['next_service_due_days']} days.\n\n"
            f"Your nearest premium {self.saas_name}-partnered service center:\n"
            f"{row['dealer_dealership_name']}\n"
            f"Location: {row['dealer_location']}\n"
            f"Contact: {row['dealer_contact_number']}\n"
            f"Service Advisor: {row['dealer_service_advisor']}\n"
            f"Working Hours: {row['dealer_working_hours']}\n\n"
            f"Special Offer Just for You:\n{row['offer_description']}\n"
            f"Use Code: {row['eligible_offer_code']}\n\n"
            f"Book through our {self.saas_name} portal for priority service and tra
            f"Best regards,\n{self.saas_name} Team"
        )
        return message

    def _create_dealership_message(self, row):
        if not row.get('dealer_is_saas_partner', False):
            message = (
                f"Dear Valued {row['make']} Owner,\n\n"
                f"Your vehicle is due for service. Nearest available service center
                f"{row['dealer_dealership_name']}\n"
                f"Location: {row['dealer_location']}\n"
                f"Contact: {row['dealer_contact_number']}\n\n"
                f"Special Offer Just for You:\n{row['offer_description']}\n"
                f"Use Code: {row['eligible_offer_code']}\n\n"
                "Please contact the dealership directly for assistance.\n\n"
                "Thank you for your trust."
            )
            return message

        message = (
            f"Dear Valued {row['make']} Owner,\n\n"
            f"Your vehicle is due for service at {row['dealer_dealership_name']}.\r
            f"We've assigned {row['dealer_service_advisor']} as your personal servi
            f"Contact directly at {row['dealer_contact_number']} for immediate assi
        )

        if row['customer_feedback'] == 'Poor Service':
            message += "We apologize for your last experience and guarantee better

        message += (
            f"Special Offer Just for You:\n{row['offer_description']}\n"
            f"Use Code: {row['eligible_offer_code']}\n\n"
            f"Working Hours: {row['dealer_working_hours']}\n"
            "Walk-ins welcome, but appointments recommended.\n\n"
            f"Your {row['dealer_dealership_name']} Team"
        )
        return message

    def save_reminders(self, filename='enhanced_service_reminders.csv'):
        reminders = self.generate_reminder_list()
        if not reminders.empty:
            reminders.to_csv(filename, index=False)
            print(f"Enhanced reminder list saved to {filename}")
        else:
            print("No reminders to save.")
        return reminders


if __name__ == "__main__":
    predictor = EnhancedServicePredictor()

    try:
        predictor.load_data('sample_service_data.csv')
    except FileNotFoundError:
        print("Sample data file not found. Using demo data.")

        demo_data = {
            'make': ['Toyota', 'Hyundai', 'Honda', 'Maruti', 'Ford'],
            'model': ['Innova', 'Creta', 'City', 'Swift', 'Fiesta'],
            'location': ['Anna Nagar', 'Velachery', 'OMR', 'T Nagar', 'Adyar'],
            'age_of_vehicle': [3, 2, 4, 1, 5],
            'odometer_reading': [45000, 22000, 60000, 10000, 80000],
            'next_service_due_days': [90, 30, 120, 60, 45],
            'last_service_kms': [5000, 2500, 10000, 1200, 7000],
            'avg_kms_per_month': [1250, 1100, 1500, 800, 1300],
            'last_service_cost': [5000, 4000, 6000, 3500, 4500],
            'days_since_last_service': [120, 30, 200, 45, 180],
            'number_of_services': [3, 2, 5, 1, 4],
            'warranty_status': ['Expired', 'Active', 'Active', 'Expired', 'Expired'
            'insurance_status': ['Active', 'Active', 'Expired', 'Active', 'Active']
            'fuel_type': ['Petrol', 'Diesel', 'Petrol', 'Petrol', 'Diesel'],
            'transmission': ['Manual', 'Automatic', 'Manual', 'Automatic', 'Manual'
            'customer_type': ['Retail', 'Fleet', 'Retail', 'Retail', 'Fleet'],
            'customer_feedback': ['Good', 'Poor Service', 'Excellent', 'Good', 'Poo
            'AMC_status': ['Active', 'Not Subscribed', 'Active', 'Not Subscribed',
        }
        df_demo = pd.DataFrame(demo_data)
        df_demo['service_needed_soon'] = df_demo['next_service_due_days'].apply(lam

        chennai_locations = {
            'Anna Nagar': (13.0878, 80.2119),
            'T Nagar': (13.0478, 80.2427),
            'OMR': (12.9716, 80.2497),
            'Velachery': (12.9850, 80.2165),
            'Adyar': (13.0067, 80.2566),
            'Porur': (13.0390, 80.1619),
            'Ambattur': (13.1147, 80.1548)
        }
        df_demo['latitude'] = df_demo['location'].map(lambda x: chennai_locations.g
        df_demo['longitude'] = df_demo['location'].map(lambda x: chennai_locations.

        predictor.customer_data = df_demo

    predictor.train_model()
    predictor.predict_service_needs()
    enhanced_reminders = predictor.save_reminders()

    if not enhanced_reminders.empty:
        print("\nSample SaaS Message:")
        print(enhanced_reminders.iloc[0]['saas_message'])
        print("\nSample Dealership Message:")
        print(enhanced_reminders.iloc[0]['dealership_message'])
    else:
        print("\nNo service reminders needed based on current predictions.")
```

```
Sample data file not found. Using demo data.
Model Evaluation:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1

Enhanced reminder list saved to enhanced_service_reminders.csv

Sample SaaS Message:
Dear Retail Customer,

Our AutoMoto AI Smart Service System recommends service for your Toyota Innova wit
hin 90 days.

Your nearest premium AutoMoto AI-partnered service center:
Lanson Toyota - Anna Nagar
Location: Anna Nagar
Contact: +91 44 2620 9727
Service Advisor: Mr. Rajesh Naidu
Working Hours: 9:30 AM - 7:30 PM (Weekend Special)

Special Offer Just for You:
Thank you for your loyalty! Get a complimentary vehicle checkup with your next ser
vice.
Use Code: AMAI-CHECKUP

Book through our AutoMoto AI portal for priority service and tracking.

Best regards,
AutoMoto AI Team

Sample Dealership Message:
Dear Valued Toyota Owner,

Your vehicle is due for service at Lanson Toyota - Anna Nagar.

We've assigned Mr. Rajesh Naidu as your personal service advisor.
Contact directly at +91 44 2620 9727 for immediate assistance.

Special Offer Just for You:
Thank you for your loyalty! Get a complimentary vehicle checkup with your next ser
vice.
Use Code: AMAI-CHECKUP

Working Hours: 9:30 AM - 7:30 PM (Weekend Special)
Walk-ins welcome, but appointments recommended.

Your Lanson Toyota - Anna Nagar Team
```

In [2]:

```python
import joblib

# Save model
joblib.dump(EnhancedServicePredictor, 'Class_service_reminder_model_03.pkl')
print("Model saved as 'Class_service_reminder_model_03.pkl'")

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

```
Model saved as 'Class_service_reminder_model_03.pkl'
```

In [ ]:

```
```

In [ ]:

```
```