# Reminder System based on customer type and AMC Status

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.impute import SimpleImputer
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Load the dataset
df = pd.read_csv('modify_service_df.csv')
```

In [3]:
```
df
```

Out[3]:

| | location | customer_type | preferred_language | make | model | year_of_purchase | age_of_vehi |
|---|---|---|---|---|---|---|---|
| **0** | OMR | Retail | Tamil | Ford | Aspire | 2019 | |
| **1** | T Nagar | Corporate | Tamil | Toyota | Yaris | 2019 | |
| **2** | Anna Nagar | Retail | English | Ford | Figo | 2020 | |
| **3** | OMR | Corporate | English | Honda | City | 2019 | |
| **4** | T Nagar | Fleet | Hindi | Honda | City | 2015 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **995** | Anna Nagar | Retail | Hindi | Hyundai | i20 | 2015 | |
| **996** | Velachery | Corporate | Tamil | Hyundai | Creta | 2016 | |
| **997** | T Nagar | Retail | Tamil | Toyota | Innova | 2021 | |
| **998** | OMR | Fleet | Tamil | Hyundai | i10 | 2015 | |
| **999** | OMR | Retail | Hindi | Toyota | Innova | 2016 | |

1000 rows × 49 columns

In [4]:
```
df.isnull().sum()
```

Out[4]:

```
location                  0
customer_type             0
preferred_language        0
make                      0
model                     0
year_of_purchase          0
age_of_vehicle            0
fuel_type                 0
transmission              0
odometer_reading          0
warranty_status           0
insurance_status          0
last_service_type         0
service_center            0
number_of_services        0
last_service_kms          0
avg_kms_per_month         0
next_service_due_kms      0
next_service_due_days     0
AMC_status                0
pending_service           0
response                  0
follow_up_required        0
telecaller_name           0
service_booked            0
call_duration_sec         0
remark                    0
eligible_offer_code       0
offer_description         0
offer_valid_till          0
sent_sms                  0
sms_delivered             0
sms_clicked               0
sent_Whats                0
Whats_delivered           0
Whats_clicked             0
sent_email                0
clicked_email             0
email_opened              0
last_service_cost         0
feedback_score            0
pickup_drop_required      0
customer_feedback         0
days_since_last_service   0
days_until_next_service   0
days_since_follow_up      0
days_since_feedback       0
days_since_last_call      0
alert_due                 0
dtype: int64
```

In [6]:

```
#df['reminder_message'].fillna(0, inplace=True)
```

In [7]:

```
df.isnull().sum()
```

Out[7]:
```
location                  0
customer_type             0
preferred_language        0
make                      0
model                     0
year_of_purchase          0
age_of_vehicle            0
fuel_type                 0
transmission              0
odometer_reading          0
warranty_status           0
insurance_status          0
last_service_type         0
service_center            0
number_of_services        0
last_service_kms          0
avg_kms_per_month         0
next_service_due_kms      0
next_service_due_days     0
AMC_status                0
pending_service           0
response                  0
follow_up_required        0
telecaller_name           0
service_booked            0
call_duration_sec         0
remark                    0
eligible_offer_code       0
offer_description         0
offer_valid_till          0
sent_sms                  0
sms_delivered             0
sms_clicked               0
sent_Whats                0
Whats_delivered           0
Whats_clicked             0
sent_email                0
clicked_email             0
email_opened              0
last_service_cost         0
feedback_score            0
pickup_drop_required      0
customer_feedback         0
days_since_last_service   0
days_until_next_service   0
days_since_follow_up      0
days_since_feedback       0
days_since_last_call      0
alert_due                 0
dtype: int64
```

In [8]:
```
# Data Preparation
# Create target variable - whether service is due within 120 days
df['service_due_soon'] = df['next_service_due_days'].apply(lambda x: 1 if x <= 120
```

In [9]:
```
# Feature selection - choose relevant columns for prediction
features = [
    'age_of_vehicle',
    'odometer_reading',
    'last_service_kms',
    'avg_kms_per_month',
    'next_service_due_kms',
    'last_service_cost',
    'days_since_last_service',
```

```
    'warranty_status',
    'insurance_status',
    'fuel_type',
    'transmission',
    'customer_type',
    'feedback_score',
    'customer_feedback',
    'AMC_status',
    'number_of_services'
]

target = 'service_due_soon'
```

In [10]:
```
# Preprocessing pipeline
numeric_features = [
    'age_of_vehicle',
    'odometer_reading',
    'last_service_kms',
    'avg_kms_per_month',
    'next_service_due_kms',
    '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'
]
```

In [11]:
```
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

In [12]:
```
df
```

Out[12]:

| | location | customer_type | preferred_language | make | model | year_of_purchase | age_of_vehi |
|---|---|---|---|---|---|---|---|
| **0** | OMR | Retail | Tamil | Ford | Aspire | 2019 | |
| **1** | T Nagar | Corporate | Tamil | Toyota | Yaris | 2019 | |
| **2** | Anna Nagar | Retail | English | Ford | Figo | 2020 | |
| **3** | OMR | Corporate | English | Honda | City | 2019 | |
| **4** | T Nagar | Fleet | Hindi | Honda | City | 2015 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **995** | Anna Nagar | Retail | Hindi | Hyundai | i20 | 2015 | |
| **996** | Velachery | Corporate | Tamil | Hyundai | Creta | 2016 | |
| **997** | T Nagar | Retail | Tamil | Toyota | Innova | 2021 | |
| **998** | OMR | Fleet | Tamil | Hyundai | i10 | 2015 | |
| **999** | OMR | Retail | Hindi | Toyota | Innova | 2016 | |

1000 rows × 50 columns

In [13]:
```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

In [14]:
```
df
```

Out[14]:

| | location | customer_type | preferred_language | make | model | year_of_purchase | age_of_vehi |
|---|---|---|---|---|---|---|---|
| **0** | OMR | Retail | Tamil | Ford | Aspire | 2019 | |
| **1** | T Nagar | Corporate | Tamil | Toyota | Yaris | 2019 | |
| **2** | Anna Nagar | Retail | English | Ford | Figo | 2020 | |
| **3** | OMR | Corporate | English | Honda | City | 2019 | |
| **4** | T Nagar | Fleet | Hindi | Honda | City | 2015 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **995** | Anna Nagar | Retail | Hindi | Hyundai | i20 | 2015 | |
| **996** | Velachery | Corporate | Tamil | Hyundai | Creta | 2016 | |
| **997** | T Nagar | Retail | Tamil | Toyota | Innova | 2021 | |
| **998** | OMR | Fleet | Tamil | Hyundai | i10 | 2015 | |
| **999** | OMR | Retail | Hindi | Toyota | Innova | 2016 | |

1000 rows × 50 columns

In [15]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

In [16]:
```
df.head()
```

Out[16]:

| | location | customer_type | preferred_language | make | model | year_of_purchase | age_of_vehicle |
|---|---|---|---|---|---|---|---|
| **0** | OMR | Retail | Tamil | Ford | Aspire | 2019 | 6 |
| **1** | T Nagar | Corporate | Tamil | Toyota | Yaris | 2019 | 6 |
| **2** | Anna Nagar | Retail | English | Ford | Figo | 2020 | 5 |
| **3** | OMR | Corporate | English | Honda | City | 2019 | 6 |
| **4** | T Nagar | Fleet | Hindi | Honda | City | 2015 | 10 |

5 rows × 50 columns

In [17]:
```
# Split data
X = df[features]#indep
y = df[target]#dep
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [18]:
```
# Model training
models = {
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'SVM': SVC(random_state=42),
    'Logistic Regression': LogisticRegression(random_state=42)
}

results = {}
for name, model in models.items():
    clf = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    results[name] = {
        'accuracy': accuracy_score(y_test, y_pred),
        'report': classification_report(y_test, y_pred)
    }
```

In [19]:
```
# Print results
for model_name, metrics in results.items():
    print(f"Model: {model_name}")
    print(f"Accuracy: {metrics['accuracy']:.2f}")
    print("Classification Report:")
    print(metrics['report'])
    print("\n")
```

```
Model: Random Forest
Accuracy: 0.95
Classification Report:
              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



Model: Gradient Boosting
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       178
           1       1.00      1.00      1.00        22

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



Model: SVM
Accuracy: 0.89
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       178
           1       0.00      0.00      0.00        22

    accuracy                           0.89       200
   macro avg       0.45      0.50      0.47       200
weighted avg       0.79      0.89      0.84       200



Model: Logistic Regression
Accuracy: 0.89
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       178
           1       0.00      0.00      0.00        22

    accuracy                           0.89       200
   macro avg       0.45      0.50      0.47       200
weighted avg       0.79      0.89      0.84       200
```

In [20]:

```
# Select the best model (Random Forest in this case)
best_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', GradientBoostingClassifier(random_state=42))
])
best_model.fit(X, y)
```

Out[20]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImputer(s
trategy='median')),
                                                                  ('scaler',
                                                                   StandardScaler
())]),
                                                  ['age_of_vehicle',
                                                   'odometer_reading',
                                                   'last_service_kms',
                                                   'avg_kms_per_month',
                                                   'next_service_due_kms',
                                                   'last_service_cost',
                                                   'days_since_last_service',
                                                   'number_of_services']),
                                                 ('cat',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImputer(s
trategy='most_frequent')),
                                                                  ('onehot',
                                                                   OneHotEncoder(h
andle_unknown='ignore'))]),
                                                  ['warranty_status',
                                                   'insurance_status',
                                                   'fuel_type', 'transmission',
                                                   'customer_type',
                                                   'customer_feedback',
                                                   'AMC_status'])])),
                ('classifier', GradientBoostingClassifier(random_state=42))])
```

In [21]:
```
# Predict service due soon for all customers
df['predicted_service_due'] = best_model.predict(X)
```

In [22]:
```
# Reminder System based on customer type and AMC Status
def generate_reminder_message(row):
    """Generate personalized reminder messages based on customer data"""
    base_msg = f"Dear Valued Customer,\n\n"

    if row['customer_type'] == 'Corporate':
        base_msg += f"We hope this message finds you well. "
    else:
        base_msg += f"Hope you're doing well. "

    base_msg += f"Your {row['make']} {row['model']} (purchased in {row['year_of_pur
    base_msg += f"is due for service in {row['next_service_due_days']} days.\n\n"

    if row['AMC_status'] == 'Yes':
        base_msg += "As an AMC holder, you're eligible for special benefits. "
    else:
        base_msg += "Regular maintenance ensures optimal performance. "

    base_msg += "Please schedule your service at your earliest convenience.\n\n"

    # Add offer if available
    if pd.notna(row['eligible_offer_code']):
        base_msg += f"Special offer for you: {row['offer_description']} (Code: {row
        base_msg += f"Valid until {row['offer_valid_till']}.\n\n"

    base_msg += "Best regards,\nAutoMoto AI Service Team"

    return base_msg
```

In [23]:
```
# Generate messages for customers predicted to need service
df['reminder_message'] = df.apply(
    lambda row: generate_reminder_message(row) if row['predicted_service_due'] == 1
    axis=1
)
```

In [24]:
```
# Determine communication channels
def determine_channels(row):
    """Determine which channels to use based on customer preferences"""
    channels = []

    # Check which channels have been successful in the past
    if row['sent_Whats'] == 'Yes' and row['Whats_delivered'] == 'Yes':
        channels.append('WhatsApp')
    if row['sent_email'] == 'Yes' and row['email_opened'] == 'Yes':
        channels.append('Email')
    if row['sent_sms'] == 'Yes' and row['sms_delivered'] == 'Yes':
        channels.append('SMS')

    # Default channels if no history
    if not channels:
        channels = ['WhatsApp', 'Email', 'SMS']

    return channels

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

In [25]:
```
# Create reminder schedule based on urgency
def create_reminder_schedule(days_until_service):
    """Create a reminder schedule based on how soon service is needed"""
    if days_until_service <= 30:
        return [7, 3, 1]  # Days before service to send reminders
    elif days_until_service <= 60:
        return [30, 15, 7, 3]
    else:
        return [60, 30, 15, 7]

df['reminder_schedule'] = df['next_service_due_days'].apply(create_reminder_schedul
```

In [26]:
```
# Save results for the service team
service_due_customers = df[df['predicted_service_due'] == 1][[
    'location', 'customer_type', 'make', 'model', 'year_of_purchase',
    'next_service_due_days', 'preferred_channels', 'reminder_message',
    'reminder_schedule', 'telecaller_name'
]]
```

In [27]:
```
# Add priority based on days until service
service_due_customers['priority'] = pd.cut(
    service_due_customers['next_service_due_days'],
    bins=[0, 30, 60, 90, 120],
    labels=['High', 'Medium-High', 'Medium', 'Low']
)
```

In [28]:
```
# Save to CSV for the service team
service_due_customers.to_csv('service_reminder_list.csv', index=False)

print(f"Identified {len(service_due_customers)} customers needing service reminders
print("Reminder list saved to 'service_reminder_list.csv'")
```

```
Identified 127 customers needing service reminders.
Reminder list saved to 'service_reminder_list.csv'
```

In [29]:
```python
# Example of how to implement the actual reminder sending (pseudo-code)
def send_reminders(customer_data):
    """Function to actually send reminders (implementation would depend on your sys
    for _, customer in customer_data.iterrows():
        message = customer['reminder_message']
        channels = customer['preferred_channels']
        schedule = customer['reminder_schedule']

        # In a real implementation, you would:
        # 1. Schedule reminders based on the days in 'schedule'
        # 2. Send through each channel in 'channels'
        # 3. Log the communication for tracking

        print(f"\nReminder for {customer['make']} {customer['model']}:")
        print(f"Priority: {customer['priority']}")
        print(f"Channels: {', '.join(channels)}")
        print(f"Message:\n{message}")

# Uncomment to see example reminders
# send_reminders(service_due_customers.head())
```

In [30]:
```python
send_reminders(service_due_customers.head())
```

Reminder for Ford EcoSport:
Priority: Medium-High
Channels: WhatsApp, SMS
Message:
Dear Valued Customer,

Hope you're doing well. Your Ford EcoSport (purchased in 2022) is due for service in 32 days.

Regular maintenance ensures optimal performance. Please schedule your service at y our earliest convenience.

Special offer for you: Engine Oil Discount (Code: OFF697). Valid until 04-08-2025.

Best regards,
AutoMoto AI Service Team

Reminder for Ford Figo:
Priority: nan
Channels: WhatsApp, SMS
Message:
Dear Valued Customer,

Hope you're doing well. Your Ford Figo (purchased in 2016) is due for service in -86 days.

Regular maintenance ensures optimal performance. Please schedule your service at y our earliest convenience.

Special offer for you: Free Wash (Code: OFF992). Valid until 04-08-2025.

Best regards,
AutoMoto AI Service Team

Reminder for Toyota Innova:
Priority: Medium-High
Channels: WhatsApp, Email, SMS
Message:
Dear Valued Customer,

We hope this message finds you well. Your Toyota Innova (purchased in 2018) is due for service in 34 days.

As an AMC holder, you're eligible for special benefits. Please schedule your servi ce at your earliest convenience.

Special offer for you: Free Wash (Code: OFF787). Valid until 16-08-2025.

Best regards,
AutoMoto AI Service Team

Reminder for Toyota Etios:
Priority: Low
Channels: WhatsApp, Email, SMS
Message:
Dear Valued Customer,

We hope this message finds you well. Your Toyota Etios (purchased in 2017) is due for service in 91 days.

As an AMC holder, you're eligible for special benefits. Please schedule your servi ce at your earliest convenience.

Special offer for you: Engine Oil Discount (Code: OFF100). Valid until 31-07-2025.

```
Best regards,
AutoMoto AI Service Team

Reminder for Maruti Swift:
Priority: High
Channels: WhatsApp, SMS
Message:
Dear Valued Customer,

Hope you're doing well. Your Maruti Swift (purchased in 2018) is due for service i
n 2 days.

Regular maintenance ensures optimal performance. Please schedule your service at y
our earliest convenience.

Special offer for you: 15% OFF AMC (Code: OFF297). Valid until 01-08-2025.

Best regards,
AutoMoto AI Service Team
```

In [37]:
```
import joblib

# Save trained model pipeline to file
joblib.dump(best_model, 'Class_service_reminder_model1.pkl')
```

Out[37]:
```
['Class_service_reminder_model1.pkl']
```

In [38]:
```
# To load it later:
loaded_model = joblib.load('Class_service_reminder_model1.pkl')
```

In [ ]:
```

```