Regression Model based on :

1. Service Cost Prediction (last_service_cost)

2. Days Until Next Service (next_service_due_days)

3. Customer Lifetime Value (Potential future revenue)

4. Odometer Reading Prediction (future odometer_reading)

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

In [2]:

```
# Load the data
df = pd.read_csv('modify_service_df.csv')  # Replace with your CSV path
```

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]:

```
# --- Common Preprocessing ---
# Identify categorical columns you need to encode for your problems
categorical_cols = ['location', 'customer_type', 'preferred_language', 'make', 'mod
                    'fuel_type', 'transmission', 'warranty_status', 'insurance_stat
                    'last_service_type', 'service_center', 'AMC_status']
```

In [5]:

```
# Encode categorical columns with LabelEncoder
for col in categorical_cols:
    if col in df.columns:
        df[col] = LabelEncoder().fit_transform(df[col].astype(str))
```

In [6]:

```
# Fill missing numeric values with median
num_cols = df.select_dtypes(include=['number']).columns.tolist()
```

In [7]:

```
# Exclude target columns per problem to avoid pre-fill mistakes
for col in num_cols:
    df[col] = df[col].fillna(df[col].median())
```

In [9]:

```
# To keep examples focused, define train/test split helper
def train_and_evaluate(X, y, problem_name):
    print(f"\n===== {problem_name} =====")
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

    model = RandomForestRegressor(random_state=42, n_jobs=-1)
    # Optional: You can tune hyperparams here with GridSearchCV as well

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

    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    print(f"RMSE: {rmse:.2f}")
    print(f"R² Score: {r2:.3f}")
    return model

# 1. Service Cost Prediction (last_service_cost)
print("Building Service Cost Prediction model...")
features_sc = ['make', 'model', 'age_of_vehicle', 'odometer_reading', 'last_service
               'customer_type', 'number_of_services']
target_sc = 'last_service_cost'

# Filter features and target
df_sc = df.dropna(subset=features_sc + [target_sc])
X_sc = df_sc[features_sc]
y_sc = df_sc[target_sc]

model_sc = train_and_evaluate(X_sc, y_sc, "Service Cost Prediction")

# 2. Days Until Next Service (next_service_due_days)
print("Building Next Service Due Days Prediction model...")
features_sd = ['odometer_reading', 'avg_kms_per_month', 'last_service_type', 'age_o
target_sd = 'next_service_due_days'

df_sd = df.dropna(subset=features_sd + [target_sd])
X_sd = df_sd[features_sd]
y_sd = df_sd[target_sd]

model_sd = train_and_evaluate(X_sd, y_sd, "Next Service Due Days Prediction")

# 3. Customer Lifetime Value (Potential future revenue)
print("Building Customer Lifetime Value Prediction model...")
features_clv = ['number_of_services', 'last_service_cost', 'service_center', 'age_o
                'feedback_score', 'odometer_reading', 'customer_type']
# Assuming you have a column "customer_lifetime_value" or calculate proxy; here we
if 'customer_lifetime_value' in df.columns:
    target_clv = 'customer_lifetime_value'
else:
    # Create a proxy target for demo; in production, replace with actual CLV data
    df['customer_lifetime_value_proxy'] = df['last_service_cost'] * df['number_of_s
    target_clv = 'customer_lifetime_value_proxy'

df_clv = df.dropna(subset=features_clv + [target_clv])
X_clv = df_clv[features_clv]
y_clv = df_clv[target_clv]

model_clv = train_and_evaluate(X_clv, y_clv, "Customer Lifetime Value Prediction")

# 4. Odometer Reading Prediction (future odometer_reading)
print("Building Future Odometer Reading Prediction model...")
features_od = ['odometer_reading', 'avg_kms_per_month', 'age_of_vehicle', 'customer
target_od = 'next_service_due_kms'  # Proxy for future odometer reading

df_od = df.dropna(subset=features_od + [target_od])
X_od = df_od[features_od]
y_od = df_od[target_od]

model_od = train_and_evaluate(X_od, y_od, "Future Odometer Reading Prediction")

print("\nAll models trained and evaluated successfully.")
```

```
Building Service Cost Prediction model...

===== Service Cost Prediction =====
RMSE: 3958.84
R² Score: -0.287
Building Next Service Due Days Prediction model...

===== Next Service Due Days Prediction =====
RMSE: 54.97
R² Score: 0.703
Building Customer Lifetime Value Prediction model...

===== Customer Lifetime Value Prediction =====
RMSE: 1354.53
R² Score: 0.998
Building Future Odometer Reading Prediction model...

===== Future Odometer Reading Prediction =====
RMSE: 769.18
R² Score: 0.999

All models trained and evaluated successfully.
```

In [11]:

```
import joblib

# Save model
joblib.dump(train_and_evaluate, 'Reg_service_reminder_model_03.pkl')
print("Model saved as 'Reg_service_reminder_model_03.pkl'")

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

```
Model saved as 'Reg_service_reminder_model_03.pkl'
```

In [ ]: