Version 9: Fine-tuning with Lower Lasso Alpha

Introduction: Changes & Purpose

In Version 9, we implemented fine-tuning by reducing the Lasso regularization parameter (alpha).

- **Change:** Lowered the Lasso alpha value to reduce the extent of coefficient shrinkage.
- **Purpose:** This adjustment aimed to retain more informative features while still controlling for overfitting, thereby improving the model's generalization on new data.

In [8]:
```
import mlflow
import mlflow.sklearn
import pandas as pd
import numpy as np
import datetime
import json
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.metrics import root_mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
import pickle
import matplotlib.pyplot as plt

# Set MLflow experiment for Version 9
mlflow.set_experiment("Car Price Prediction - Version 9")

with mlflow.start_run():
    # 1. Data Loading and Preprocessing
    df = pd.read_csv("car_price_dataset.csv")

    # Map categorical features to numerical values
    brand_mapping = {
        "Audi": 0, "BMW": 1, "Mercedes": 2, "Volkswagen": 3,
        "Toyota": 4, "Ford": 5, "Honda": 6, "Chevrolet": 7,
        "Kia": 8, "Hyundai": 9
    }
    fuel_mapping = {
        "Petrol": 0, "Diesel": 1, "Electric": 2, "Hybrid": 3
    }
    transmission_mapping = {
        "Manual": 0, "Automatic": 1, "Semi-Automatic": 2
    }

    df["Brand"] = df["Brand"].map(brand_mapping)
    df["Fuel_Type"] = df["Fuel_Type"].map(fuel_mapping)
    df["Transmission"] = df["Transmission"].map(transmission_mapping)

    # Feature Engineering:
    # Calculate car age and apply a square root transformation to Mileage
    current_year = datetime.datetime.now().year
    df["Car_Age"] = current_year - df["Year"]
    df["Mileage_sqrt"] = np.sqrt(df["Mileage"])

    # Log-transform the target variable to reduce skewness
    df["Log_Price"] = np.log1p(df["Price"])

    # Select features and target
    features = ["Brand", "Engine_Size", "Mileage_sqrt", "Car_Age", "Fuel_Type", "Transmission", "Doo
    X = df[features]
    y = df["Log_Price"]

    # Drop any rows with missing values in our selected columns
    df = df.dropna(subset=features + ["Log_Price"])

    # 2. Data Splitting
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # 3. Model Training using a Pipeline with Polynomial Expansion and Lasso Regularization
```

```
    # In Version 9, we fine-tune by lowering the Lasso alpha value
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('poly', PolynomialFeatures(degree=3, include_bias=False)),  # Use polynomial features to ca
        ('lasso', Lasso(alpha=0.01, max_iter=10000))  # Fine-tuned lower alpha for less aggressive s
    ])

    pipeline.fit(X_train, y_train)

    # 4. Evaluation: Predictions & Metrics
    y_train_pred = pipeline.predict(X_train)
    y_test_pred = pipeline.predict(X_test)

    train_rmse_log = root_mean_squared_error(y_train, y_train_pred)
    test_rmse_log = root_mean_squared_error(y_test, y_test_pred)
    r2 = r2_score(y_test, y_test_pred)

    print(f"Train RMSE (log scale): {train_rmse_log:.2f}")
    print(f"Test RMSE (log scale): {test_rmse_log:.2f}")
    print(f"R² Score: {r2:.4f}")

    # Optionally, convert predictions back to the original price scale for additional evaluation
    y_test_pred_actual = np.expm1(y_test_pred)
    y_test_actual = np.expm1(y_test)
    test_rmse_actual = np.sqrt(root_mean_squared_error(y_test_actual, y_test_pred_actual))
    print(f"Test RMSE (Original Scale): {test_rmse_actual:.2f}")

    # 5. MLflow Logging: Parameters, Metrics, and Artifacts
    mlflow.log_param("model_type", "Lasso Regression with Polynomial Expansion")
    mlflow.log_param("lasso_alpha", 0.01)
    mlflow.log_param("poly_degree", 3)
    mlflow.log_metric("train_rmse_log", train_rmse_log)
    mlflow.log_metric("test_rmse_log", test_rmse_log)
    mlflow.log_metric("r2", r2)
    mlflow.log_metric("test_rmse_original", test_rmse_actual)

    # Save model parameters artifact (optional)
    final_model_info = {
        "lasso_intercept": float(pipeline.named_steps['lasso'].intercept_),
        "lasso_coefficients": pipeline.named_steps['lasso'].coef_.tolist(),
        "scaler_mean": pipeline.named_steps['scaler'].mean_.tolist(),
        "scaler_scale": pipeline.named_steps['scaler'].scale_.tolist()
    }
    with open("final_model_v9.json", "w") as f:
        json.dump(final_model_info, f)
    mlflow.log_artifact("final_model_v9.json", artifact_path="model_artifacts")

    # Log the entire model using MLflow's scikit-learn integration
    mlflow.sklearn.log_model(pipeline, "model")

    # 6. Register the Model in the MLflow Model Registry
    # Note: This registration creates a new model version under the name 'CarPriceRidgeModel'
    model_uri = f"runs:/{mlflow.active_run().info.run_id}/model"
    registered_model_name = "CarPriceRidgeModel"
    mlflow.register_model(model_uri, registered_model_name)

    print("Final model from Version 9 saved and registered successfully!")
```

```
Train RMSE (log scale): 0.08
Test RMSE (log scale): 0.08
R² Score: 0.9605
Test RMSE (Original Scale): 564.16
2025/03/16 01:17:34 WARNING mlflow.models.model: Model logged without a signature and input example.
Please set `input_example` parameter when logging the model to auto infer the model signature.
Final model from Version 9 saved and registered successfully!
Registered model 'CarPriceRidgeModel' already exists. Creating a new version of this model...
Created version '6' of model 'CarPriceRidgeModel'.
```

Results Discussion for Version 9

The final model in Version 9, which was fine-tuned by lowering the Lasso regularization parameter (alpha = 0.01), achieved the following performance metrics:

- **Train RMSE (log scale):** 0.08
- **Test RMSE (log scale):** 0.08
- **R^2 Score:** 0.9605
- **Test RMSE (Original Scale):** 564.16

Interpretation of Results

- **Low RMSE on Log Scale:**
  The extremely low RMSE values on the log-transformed scale (0.08 for both train and test sets) indicate that the model fits the transformed data very well.

- **R^2 Score of 0.9605:**
  An R^2 score of 0.9605 suggests that approximately 96% of the variance in the log-transformed car prices is explained by the model. This is a strong indicator of model performance.

- **Test RMSE on Original Scale:**
  The test RMSE of 564.16 on the original price scale implies that, on average, the predicted prices differ from the actual prices by about €564. This difference, when considered in the context of car prices, demonstrates that the model maintains a good balance between precision and generalization.

Model Registration

After logging the run in MLflow, the model was automatically registered under the name **"CarPriceRidgeModel"**. Since this model already existed, a new version (Version 3) was created. This ensures that the most recent and best performing model is available for deployment in Phase 3.

Overall Conclusion

The results indicate that the fine-tuning in Version 9 has led to a model that:

- **Generalizes well** to new data, as evidenced by the similar training and test RMSE values.
- **Explains a high percentage** of variance in the transformed target variable.
- **Produces practical predictions** on the original scale, with an average error of around €564.

This comprehensive performance analysis confirms that the model is well-prepared for the next phase, where it will be deployed in a Streamlit app for real-time predictions.