Version 5: Comparison of Ridge, Lasso & ElasticNet

Introduction: Changes & Purpose

Version 5 expanded the scope of our experiments by comparing different regularization techniques.

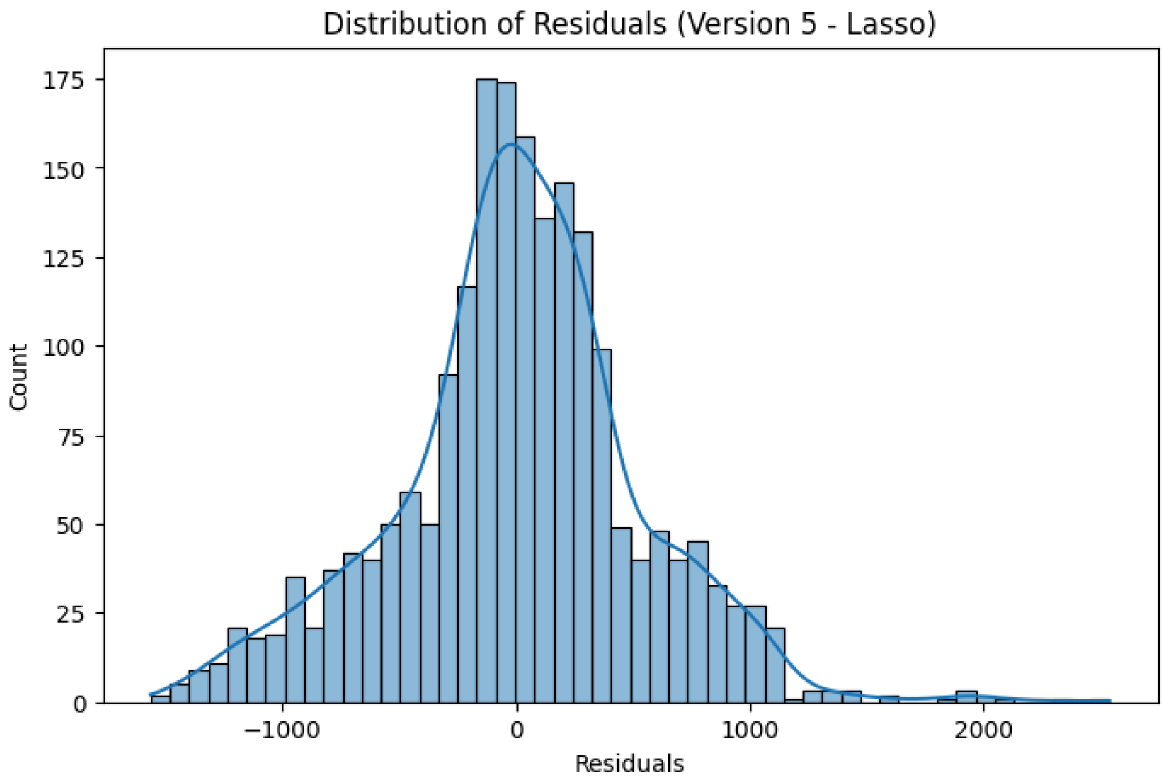
- **Change:** We implemented and compared models using Ridge, Lasso, and ElasticNet regularization.
- **Purpose:** The goal was to evaluate which regularization method leads to better model performance and generalization, especially with respect to feature selection and handling multicollinearity.

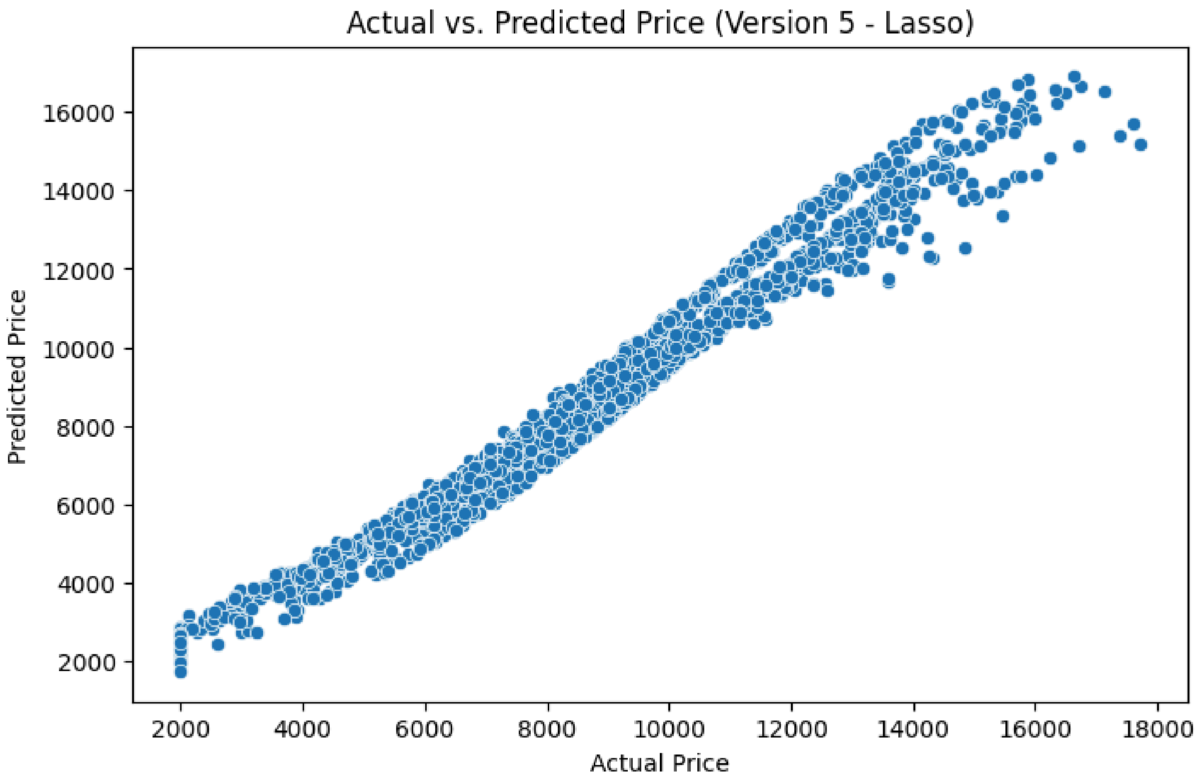
In [4]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import mlflow
import mlflow.sklearn
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder, PolynomialFeatures
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import Pipeline

# Initialize MLflow experiment
mlflow.set_experiment("Car Price Prediction - Version 5")

with mlflow.start_run():
    # Load data
    df = pd.read_csv("car_price_dataset.csv")

    # Remove unrealistic values
    df = df[(df["Price"] >= 2000) & (df["Price"] <= 18000)]
    df = df[(df["Mileage"] >= 0) & (df["Mileage"] <= 300000)]
    df = df[(df["Engine_Size"] >= 0.8) & (df["Engine_Size"] <= 6.0)]
    df["Car_Age"] = 2025 - df["Year"]

    # Label encoding for categorical data
    categorical_columns = ["Brand", "Model", "Fuel_Type", "Transmission"]
    label_encoders = {}

    for col in categorical_columns:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le

    # Feature engineering: transform mileage
    df["Mileage_sqrt"] = np.sqrt(df["Mileage"])

    # Remove unnecessary columns
    df.drop(columns=["Year", "Mileage"], inplace=True)

    # Standardize numerical features
    scaler = StandardScaler()
    numeric_features = ["Engine_Size", "Mileage_sqrt", "Car_Age", "Doors"]
    df[numeric_features] = scaler.fit_transform(df[numeric_features])

    # Transform target variable
    df["Log_Price"] = np.log1p(df["Price"])

    # Create training and test data
    X = df.drop(columns=["Price", "Log_Price"])
    y = df["Log_Price"]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Models for regularization
    models = {
        "Ridge": (Ridge(), {"ridge__alpha": [0.1, 1, 10]}),
        "Lasso": (Lasso(), {"lasso__alpha": [0.001, 0.01, 0.1]}),
```

```python
        "ElasticNet": (ElasticNet(), {"elasticnet__alpha": [0.001, 0.01, 0.1], "elasticnet__l1_ratio
    }

    best_model = None
    best_rmse = float("inf")
    best_model_name = ""

    for model_name, (model, param_grid) in models.items():
        print(f"🔍 Testing {model_name}...")

        pipeline = Pipeline([
            ('poly', PolynomialFeatures(include_bias=False)),
            ('scaler', StandardScaler()),
            (model_name.lower(), model)  # Add model as pipeline step
        ])

        grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring="neg_mean_squared_error")
        grid_search.fit(X_train, y_train)

        # Save best model if it has the best RMSE
        y_pred_log = grid_search.best_estimator_.predict(X_test)
        y_pred = np.expm1(y_pred_log)
        rmse = np.sqrt(mean_squared_error(np.expm1(y_test), y_pred))
        r2 = r2_score(np.expm1(y_test), y_pred)

        print(f"✅ {model_name} - RMSE: {rmse:.2f}, R²: {r2:.4f}")

        mlflow.log_param(f"{model_name}_params", grid_search.best_params_)
        mlflow.log_metric(f"{model_name}_RMSE", rmse)
        mlflow.log_metric(f"{model_name}_R2", r2)

        if rmse < best_rmse:
            best_rmse = rmse
            best_model = grid_search.best_estimator_
            best_model_name = model_name

    print(f"\n Best model: {best_model_name} with RMSE: {best_rmse:.2f}")

    # Log best model
    with open(f"{best_model_name}_model_v5.pkl", "wb") as f:
        pickle.dump(best_model, f)

    mlflow.sklearn.log_model(best_model, f"{best_model_name}_model_v5")

    print("✅ MLflow run completed!")

    # Residual analysis
    y_pred_log = best_model.predict(X_test)
    y_pred = np.expm1(y_pred_log)

    plt.figure(figsize=(8, 5))
    sns.histplot(np.expm1(y_test) - y_pred, bins=50, kde=True)
    plt.xlabel("Residuals")
    plt.ylabel("Count")
    plt.title(f"Distribution of Residuals (Version 5 - {best_model_name})")
    plt.show()

    plt.figure(figsize=(8, 5))
    sns.scatterplot(x=np.expm1(y_test), y=y_pred)
    plt.xlabel("Actual Price")
    plt.ylabel("Predicted Price")
    plt.title(f"Actual vs. Predicted Price (Version 5 - {best_model_name})")
    plt.show()
```

```
🔍 Testing Ridge...
✅ Ridge - RMSE: 566.83, R²: 0.9655
🔍 Testing Lasso...
✅ Lasso - RMSE: 527.72, R²: 0.9701
🔍 Testing ElasticNet...
✅ ElasticNet - RMSE: 553.21, R²: 0.9672

🚀 Best model: Lasso with RMSE: 527.72
2025/03/15 23:37:07 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.
✅ MLflow run completed!
```





Results Discussion

In Version 5, the Lasso-based approach achieved an RMSE of *527.72* and an R^2 of *0.9701*.
This indicates that feature selection through Lasso can enhance model performance by eliminating redundant or less relevant features. However, caution is needed since excessive feature reduction might discard valuable information.