Version 4: Hyperparameter Tuning for Ridge Regression

Introduction: Changes & Purpose

In Version 4, hyperparameter tuning was applied to the Ridge Regression model.

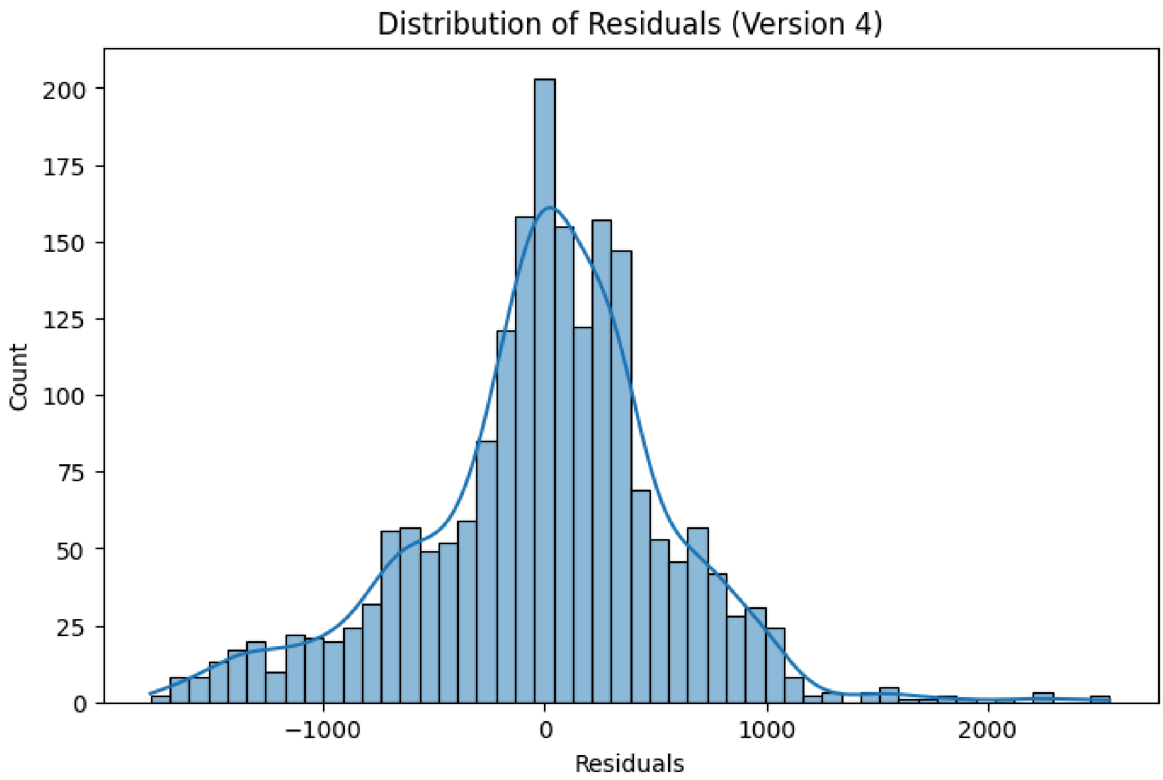
- **Change:** We experimented with various values of the regularization parameter (`alpha`) to optimize the trade-off between bias and variance.
- **Purpose:** The aim was to identify the optimal configuration that minimizes error without causing overfitting, thereby enhancing the model's generalization capability.

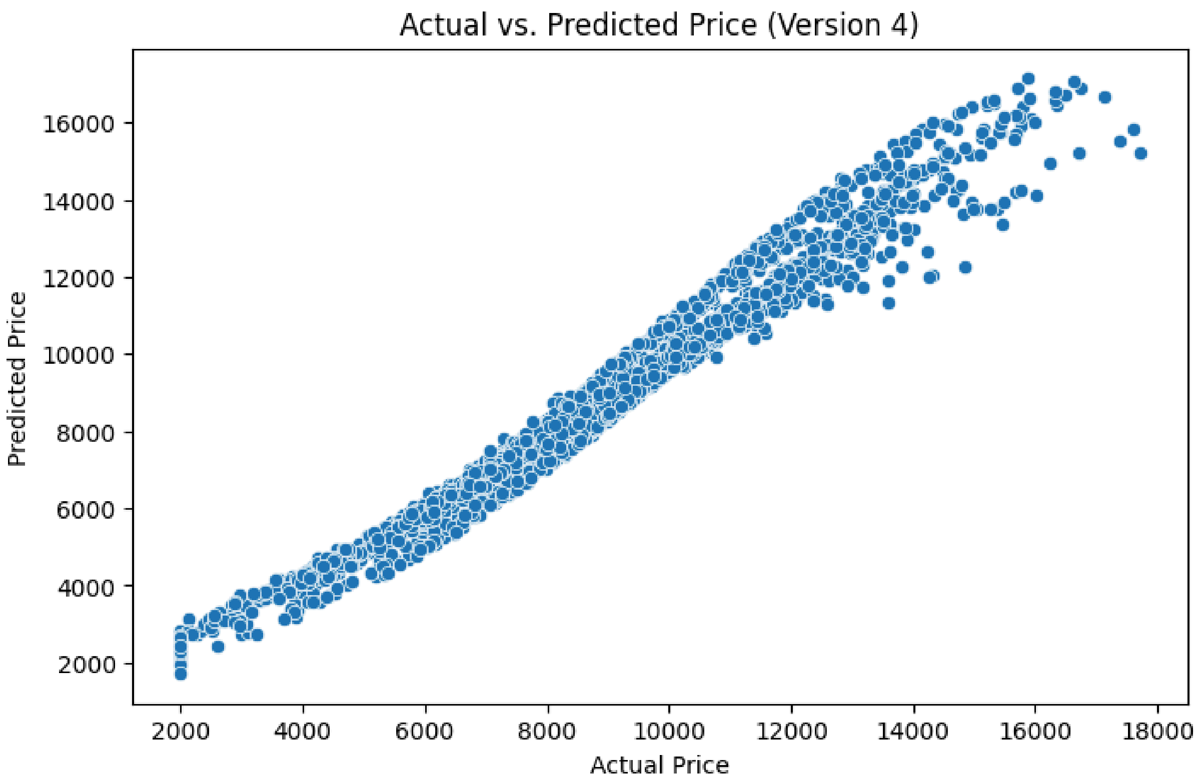
In [1]:

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

# Initialize MLflow experiment
mlflow.set_experiment("Car Price Prediction - Version 4")

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

    # Remove unrealistic values based on Version 3
    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"])
    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)

    # Machine learning pipeline with Ridge Regression
    pipeline = Pipeline([
        ('poly', PolynomialFeatures(degree=2, include_bias=False)),
```

```python
        ('scaler', StandardScaler()),
        ('ridge', Ridge(alpha=1.0))
    ])

    # Train the model
    pipeline.fit(X_train, y_train)

    # Evaluate model (train & test data)
    y_test_pred_log = pipeline.predict(X_test)
    y_test_pred = np.expm1(y_test_pred_log)

    test_rmse = np.sqrt(mean_squared_error(np.expm1(y_test), y_test_pred))
    r2 = r2_score(np.expm1(y_test), y_test_pred)

    print(f" Version 4 - Test RMSE: {test_rmse:.2f}, R²: {r2:.4f}")

    # Log metrics in MLflow
    mlflow.log_metric("Test_RMSE", test_rmse)
    mlflow.log_metric("R2_Score", r2)

    # Statsmodels OLS Regression for confidence intervals
    ols_model = sm.OLS(y_train, sm.add_constant(X_train)).fit()
    conf_interval = ols_model.conf_int(alpha=0.05)  # 95% confidence intervals

    # Save confidence intervals in MLflow
    conf_interval.to_csv("confidence_intervals.csv")
    mlflow.log_artifact("confidence_intervals.csv")

    # Save model
    with open("ridge_model_v4.pkl", "wb") as f:
        pickle.dump(pipeline, f)

    mlflow.sklearn.log_model(pipeline, "ridge_model_v4")

    print("✅ MLflow run completed!")

    # Generate plots
    plt.figure(figsize=(8, 5))
    sns.histplot(np.expm1(y_test) - y_test_pred, bins=50, kde=True)
    plt.xlabel("Residuals")
    plt.ylabel("Count")
    plt.title("Distribution of Residuals (Version 4)")
    plt.show()

    plt.figure(figsize=(8, 5))
    sns.scatterplot(x=np.expm1(y_test), y=y_test_pred)
    plt.xlabel("Actual Price")
    plt.ylabel("Predicted Price")
    plt.title("Actual vs. Predicted Price (Version 4)")
    plt.show()
```

```
🚀 Version 4 - Test RMSE: 565.58, R²: 0.9657
2025/03/15 23:37:27 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

Version 4 achieved an RMSE of *565.58* and an R^2 of *0.9657*.
The hyperparameter tuning led to marginal improvements. This suggests that while adjusting `alpha` is important, further enhancements in feature engineering or model selection might be necessary to achieve a significant performance boost.