Car Price Estimation Model using Ridge and Lasso Regression Techniques: A Comprehensive Walkthrough

---

In this project, we build a car price estimation model using Ridge and Lasso regression techniques. This comprehensive walkthrough will cover every part of the code, including data preprocessing, feature engineering, model training, evaluation, and visualization of results. The goal is to develop a predictive model for car prices based on various features such as age, fuel type, present price, and other car characteristics.

1. Importing Necessary Libraries

```
python
Copy code
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import Ridge, Lasso
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")
```

This section imports the essential libraries used in this project. Each library has its unique role:

- **Pandas** and **NumPy**: Used for data manipulation and mathematical operations.
- **Seaborn** and **Matplotlib**: Help visualize the data through graphs and plots.
- **Scikit-learn**: Offers various tools for machine learning, including data splitting, preprocessing, and regression models (Ridge and Lasso).
- **Warnings**: Suppresses unnecessary warnings that might clutter the output.

2. Loading the Dataset & Displaying an Overview

```
python
Copy code
data_path = r"D:\# DATA SCIENCE\# PROJECTS\- PROJECTS INTERNSHIPS\CODEALPHA
- DATA SCIENCE\Car Price Prediction with Machine Learning Project\car
data.csv"
df = pd.read_csv(data_path)
print(df.head())
```

In this section, the dataset is loaded from the specified path using **Pandas'** `read_csv` function, which reads the data into a DataFrame. The `head()` method displays the first few rows of the dataset to give a glimpse of the data structure, ensuring it is loaded correctly.

---

3. Data Preprocessing

Handling Missing Values & Removing Duplicates

```
python
Copy code
print("Missing Values:\n", df.isnull().sum())
df = df.dropna()
df = df.drop_duplicates()
```

Here, the dataset is first checked for missing values. Missing values can lead to inaccurate model predictions. If there are any, they are handled using the `dropna()` method, which removes rows with missing data. Duplicates are removed to ensure the uniqueness of records.

Outlier Detection and Removal using IQR for `Driven_kms`

```
python
Copy code
Q1 = df['Driven_kms'].quantile(0.25)
Q3 = df['Driven_kms'].quantile(0.75)
IQR = Q3 - Q1
df = df[~((df['Driven_kms'] < (Q1 - 1.5 * IQR)) | (df['Driven_kms'] > (Q3 +
1.5 * IQR)))]
```

Outliers can distort the model's performance, especially in regression models. Here, the **Interquartile Range (IQR)** method is used to identify and remove outliers in the `Driven_kms` column. Any data points beyond 1.5 times the IQR are considered outliers and removed.

Logarithmic Transformation for `Selling_Price` and `Present_Price`

```
python
Copy code
df['Selling_Price'] = np.log(df['Selling_Price'])
df['Present_Price'] = np.log(df['Present_Price'])
```

Both `Selling_Price` and `Present_Price` features undergo logarithmic transformation to stabilize variance and normalize the skewness. This transformation is often useful when the data contains wide-ranging values, ensuring that the model handles them better.

Encoding Categorical Variables

```
python
Copy code
label_encoder = LabelEncoder()
df['Fuel_Type'] = label_encoder.fit_transform(df['Fuel_Type'])
df['Selling_type'] = label_encoder.fit_transform(df['Selling_type'])
df['Transmission'] = label_encoder.fit_transform(df['Transmission'])
```

Since machine learning models only understand numerical values, we use **Label Encoding** to convert categorical features (`Fuel_Type`, `Selling_type`, and `Transmission`) into numerical format.

Feature Engineering: Age of the Car

```
python
Copy code
df['Age'] = 2024 - df['Year']
df = df.drop(['Car_Name', 'Year'], axis=1)
```

Here, a new feature, `Age`, is created by subtracting the `Year` of the car from the current year (2024). This is a crucial feature as the age of the car heavily influences its selling price. The irrelevant columns (`Car_Name` and `Year`) are dropped.

Dataset Overview

```
python
Copy code
print("Data Description:\n", df.describe())
print("Data Types:\n", df.dtypes)
```

The `describe()` method provides descriptive statistics, such as mean, standard deviation, and percentiles, for each numerical feature. The `dtypes` method checks the data types of each column to ensure they are appropriate for modeling.

---

4. Exploratory Data Analysis (EDA)

Distribution of `Selling_Price` and `Present_Price`

```
python
Copy code
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(df['Selling_Price'], kde=True)
plt.title('Distribution of Selling Price')
plt.subplot(1, 2, 2)
sns.histplot(df['Present_Price'], kde=True)
plt.title('Distribution of Present Price')
plt.tight_layout()
plt.show()
```

These histograms display the distribution of the log-transformed `Selling_Price` and `Present_Price` features, allowing us to observe the spread and skewness of the data. The KDE (Kernel Density Estimate) overlay provides insight into the data's probability density.

Correlation Heatmap

```
python
Copy code
plt.figure(figsize=(12, 8))
numeric_cols = df.select_dtypes(include=[np.number]).columns
sns.heatmap(df[numeric_cols].corr(), annot=True, fmt=".2f",
cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

A correlation heatmap helps visualize the linear relationships between numeric features. Features with high correlation values (close to +1 or -1) are strongly related. This heatmap can guide feature selection for modeling.

Pair Plot of Key Features

```
python
Copy code
sns.pairplot(df, vars=['Present_Price', 'Driven_kms', 'Age',
'Selling_Price'], diag_kind='kde')
plt.suptitle('Pair Plot of Features vs. Selling Price', y=1.02)
plt.show()
```

The pair plot provides a comprehensive view of relationships between multiple features (`Present_Price`, `Driven_kms`, `Age`) and the target (`Selling_Price`). The diagonal plots show the KDE for each feature, while scatter plots reveal potential patterns between features.

---

5. Defining Features and Target

```
python
Copy code
X = df.drop('Selling_Price', axis=1)
y = df['Selling_Price']
```

In this step, we define `X` as the set of independent features (everything except `Selling_Price`) and `y` as the dependent variable (target), which is the `Selling_Price` itself. This separation is crucial for supervised learning.

---

6. Splitting Data into Training and Testing Sets

```
python
Copy code
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

The dataset is split into training and testing sets using an 80/20 ratio. The **`train_test_split`** function ensures that the model is trained on one set and tested on another to evaluate its generalization performance.

---

7. Feature Scaling

```
python
Copy code
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Since regression models are sensitive to feature scales, we standardize the features using **StandardScaler**. This ensures that all features have a mean of 0 and a standard deviation of 1, improving the model's performance.

---

8. Model Training with Cross-Validation

```
python
Copy code
ridge_model = Ridge(alpha=1.0)
lasso_model = Lasso(alpha=0.1)
```

We use two regularized regression models:

- **Ridge Regression**: Adds a penalty proportional to the square of the magnitude of coefficients (L2 regularization).
- **Lasso Regression**: Adds a penalty proportional to the absolute value of the coefficients (L1 regularization).

Regularization helps prevent overfitting by shrinking large coefficients.

```
python
Copy code
ridge_cv_scores = cross_val_score(ridge_model, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')
lasso_cv_scores = cross_val_score(lasso_model, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')

print(f"Ridge RMSE: {np.sqrt(-ridge_cv_scores.mean())}")
print(f"Lasso RMSE: {np.sqrt(-lasso_cv_scores.mean())}")
```

Here, **cross-validation** is applied to evaluate model performance. The root mean square error (RMSE) is calculated for both Ridge and Lasso models. RMSE measures the difference between predicted and actual values, with lower values indicating better performance.

9. Model Evaluation and Predictions

```
python
Copy code
ridge_model.fit(X_train, y_train)
y_pred = ridge_model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print(f"Test RMSE: {rmse}")
print(f"Test R^2 Score: {r2}")
```

After training the **Ridge regression model** on the training data (`X_train` and `y_train`), we use it to make predictions on the test set (`X_test`). We then calculate the **Root Mean Squared Error (RMSE)** to evaluate the prediction accuracy of the model, with a lower RMSE indicating better performance. Additionally, we compute the **R-squared (R^2) score**, which measures how well the model explains the variance in the target variable. An R^2 score close to 1 means the model fits the data well.

---

10. Feature Importance and Visualization of Predictions

Feature Coefficients (Ridge)

```
python
Copy code
coefficients = ridge_model.coef_
plt.figure(figsize=(10, 6))
sns.barplot(x=coefficients, y=X.columns)
plt.title('Feature Coefficients (Ridge)')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.show()
```

One of the main benefits of linear models like Ridge regression is that they provide insight into feature importance via their coefficients. Larger positive or negative values indicate features that have a more significant impact on the prediction. In this step, we visualize the feature importance by plotting the Ridge regression coefficients, helping to identify which features are driving the model's predictions.

Actual vs. Predicted Prices

```
python
Copy code
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linestyle='--')
plt.title('Actual vs. Predicted Prices')
plt.xlabel('Actual Prices (Log)')
plt.ylabel('Predicted Prices (Log)')
plt.show()
```

This scatter plot compares the actual selling prices (from the test set) with the predicted prices from the Ridge model. Ideally, the points should lie along the red diagonal line, indicating perfect predictions. Deviations from the line reveal prediction errors, and this plot provides a quick visual check of the model's performance.

---

11. Further Insights: Binning and Residual Analysis

Binning the `Selling_Price`

```
python
Copy code
bins = [0, 1, 2, 3, 4, 5]
labels = ['0-1', '1-2', '2-3', '3-4', '4-5']
df['Selling_Price_Binned'] = pd.cut(df['Selling_Price'], bins=bins,
labels=labels)

price_counts = df['Selling_Price_Binned'].value_counts().sort_index()
plt.figure(figsize=(8, 6))
```

```
sns.barplot(x=price_counts.index, y=price_counts.values, palette="pastel")
plt.title('Distribution of Binned Selling Price')
plt.xlabel('Selling Price Bins')
plt.ylabel('Count')
plt.show()
```

To gain further insight into the distribution of the selling prices, we bin them into categories and create a bar plot that shows how many cars fall into each price range. Binning can be useful for identifying clusters or segments within the dataset that may require different modeling approaches.

Residual Plot

```
python
Copy code
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(0, color='red', linestyle='--')
plt.title('Residual Plot')
plt.xlabel('Predicted Prices (Log)')
plt.ylabel('Residuals')
plt.show()
```

Residual analysis is an essential step in regression modeling. Residuals are the differences between actual and predicted values. Ideally, the residuals should be randomly distributed with no discernible pattern. This plot helps in diagnosing issues like heteroscedasticity (where the variance of residuals changes with the fitted values), which could indicate problems in the model. A horizontal red line at zero helps visualize if residuals are centered around zero.

---

12. Lasso Regression: A Comparison

In this project, we also used **Lasso regression** as an alternative to Ridge. While Ridge regression penalizes the squared magnitude of coefficients, Lasso regression penalizes their absolute magnitude, often leading to sparse models where some coefficients become exactly zero. This can be useful for feature selection.

Cross-validation for Lasso

```
python
Copy code
lasso_cv_scores = cross_val_score(lasso_model, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')
print(f"Lasso RMSE: {np.sqrt(-lasso_cv_scores.mean())}")
```

We apply cross-validation for Lasso in the same way as for Ridge. The RMSE is calculated to compare the performance of the two models.

Lasso Model Training and Evaluation

```
python
```

```
Copy code
lasso_model.fit(X_train, y_train)
y_pred_lasso = lasso_model.predict(X_test)
lasso_rmse = np.sqrt(mean_squared_error(y_test, y_pred_lasso))
lasso_r2 = r2_score(y_test, y_pred_lasso)
print(f"Lasso Test RMSE: {lasso_rmse}")
print(f"Lasso Test R^2 Score: {lasso_r2}")
```

After fitting the Lasso model on the training data, we evaluate its performance on the test set. This involves calculating the RMSE and R^2 scores again to see how well the Lasso model generalizes. In some cases, Lasso may perform better than Ridge if certain features are irrelevant to the prediction, as Lasso effectively shrinks their coefficients to zero.

---

13. Discussion of Results

After evaluating both Ridge and Lasso models, we can draw conclusions based on their performance metrics (RMSE and R^2). The choice of the best model depends on several factors:

- **Ridge regression** tends to be better when all the features are important, and none can be discarded.
- **Lasso regression** may outperform Ridge when the dataset contains many irrelevant features, as it inherently performs feature selection.

In this project, we applied both methods, compared their results, and visualized key metrics, including residuals and actual vs. predicted values.

---

14. Conclusion and Future Work

This project successfully demonstrated how to build a car price estimation model using Ridge and Lasso regression techniques. We covered the entire machine learning workflow:

- **Data preprocessing**: Handling missing values, outliers, and categorical variables.
- **Feature engineering**: Transforming features to create a more informative dataset.
- **Model training and evaluation**: Using regularized regression techniques (Ridge and Lasso) with cross-validation for better generalization.
- **Visualization**: Through histograms, correlation heatmaps, pair plots, and residual analysis, we gained deep insights into the data and model performance.