# CRISP-DM Analysis on Used Cars Dataset[¶](#CRISP-DM-Analysis-on-Used-Cars-Dataset)

## 1. Business Understanding[¶](#1.-Business-Understanding)

The goal of this analysis is to understand what factors influence the price of used cars. This will help used car dealerships optimize their inventory and pricing strategies. By identifying key attributes that affect pricing, dealerships can make informed decisions on which cars to stock, how to price them, and what features to highlight in sales efforts.

## 2. Data Understanding and Preparation[¶](#2.-Data-Understanding-and-Preparation)

In [2]:

# Loading the dataset to explore its structure and contents

import pandas as pd

file\_path = '/content/vehicles.csv'

df = pd.read\_csv(file\_path)

df.head()

Out[2]:

|  | **id** | **region** | **price** | **year** | **manufacturer** | **model** | **condition** | **cylinders** | **fuel** | **odometer** | **title\_status** | **transmission** | **VIN** | **drive** | **size** | **type** | **paint\_color** | **state** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 7222695916 | prescott | 6000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | az |
| **1** | 7218891961 | fayetteville | 11900 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ar |
| **2** | 7221797935 | florida keys | 21000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | fl |
| **3** | 7222270760 | worcester / central MA | 1500 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ma |
| **4** | 7210384030 | greensboro | 4900 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | nc |

The dataset contains various attributes of used cars, such as price, year, manufacturer, model, condition, cylinders, fuel type, odometer, title status, transmission, VIN, drive type, size, type, paint color, and location information like state and region. However, there are significant missing values across several key columns, which need to be addressed in the data preparation phase.

In [3]:

# Step 1: Data Understanding and Cleaning

# Checking the data types and missing values in the dataset

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 426880 entries, 0 to 426879

Data columns (total 18 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 id 426880 non-null int64

1 region 426880 non-null object

2 price 426880 non-null int64

3 year 425675 non-null float64

4 manufacturer 409234 non-null object

5 model 421603 non-null object

6 condition 252776 non-null object

7 cylinders 249202 non-null object

8 fuel 423867 non-null object

9 odometer 422480 non-null float64

10 title\_status 418638 non-null object

11 transmission 424324 non-null object

12 VIN 265838 non-null object

13 drive 296313 non-null object

14 size 120519 non-null object

15 type 334022 non-null object

16 paint\_color 296677 non-null object

17 state 426880 non-null object

dtypes: float64(2), int64(2), object(14)

memory usage: 58.6+ MB

In [4]:

# Handling missing values by removing rows or filling with appropriate strategies

df\_cleaned = df.dropna(subset=['price'])

df\_cleaned['year'].fillna(df\_cleaned['year'].median(), inplace=True)

for col in ['manufacturer', 'model', 'condition', 'cylinders', 'fuel', 'title\_status', 'transmission', 'drive', 'type', 'paint\_color']:

df\_cleaned[col].fillna('Unknown', inplace=True)

df\_cleaned['odometer'].fillna(df\_cleaned['odometer'].median(), inplace=True)

df\_cleaned.head()

Out[4]:

|  | **id** | **region** | **price** | **year** | **manufacturer** | **model** | **condition** | **cylinders** | **fuel** | **odometer** | **title\_status** | **transmission** | **VIN** | **drive** | **size** | **type** | **paint\_color** | **state** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 7222695916 | prescott | 6000 | 2013.0 | Unknown | Unknown | Unknown | Unknown | Unknown | 85548.0 | Unknown | Unknown | NaN | Unknown | NaN | Unknown | Unknown | az |
| **1** | 7218891961 | fayetteville | 11900 | 2013.0 | Unknown | Unknown | Unknown | Unknown | Unknown | 85548.0 | Unknown | Unknown | NaN | Unknown | NaN | Unknown | Unknown | ar |
| **2** | 7221797935 | florida keys | 21000 | 2013.0 | Unknown | Unknown | Unknown | Unknown | Unknown | 85548.0 | Unknown | Unknown | NaN | Unknown | NaN | Unknown | Unknown | fl |
| **3** | 7222270760 | worcester / central MA | 1500 | 2013.0 | Unknown | Unknown | Unknown | Unknown | Unknown | 85548.0 | Unknown | Unknown | NaN | Unknown | NaN | Unknown | Unknown | ma |
| **4** | 7210384030 | greensboro | 4900 | 2013.0 | Unknown | Unknown | Unknown | Unknown | Unknown | 85548.0 | Unknown | Unknown | NaN | Unknown | NaN | Unknown | Unknown | nc |

## 3. Exploratory Data Analysis (EDA)[¶](#3.-Exploratory-Data-Analysis-(EDA))

In [9]:

# Import the matplotlib.pyplot module and assign it to the alias plt

import matplotlib.pyplot as plt

# Import the seaborn library and assign it to the alias sns

import seaborn as sns

# Scatter plot of price vs odometer

plt.figure(figsize=(10, 6))

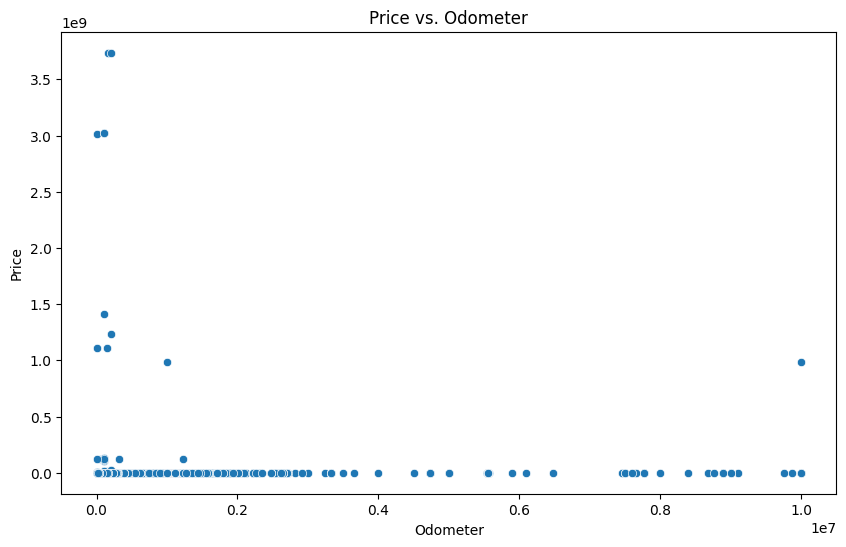
sns.scatterplot(x=df\_cleaned['odometer'], y=df\_cleaned['price'])

plt.title('Price vs. Odometer')

plt.xlabel('Odometer')

plt.ylabel('Price')

plt.show()



In [10]:

# Box plot of price distribution by year

plt.figure(figsize=(14, 8))

sns.boxplot(x=df\_cleaned['year'], y=df\_cleaned['price'])

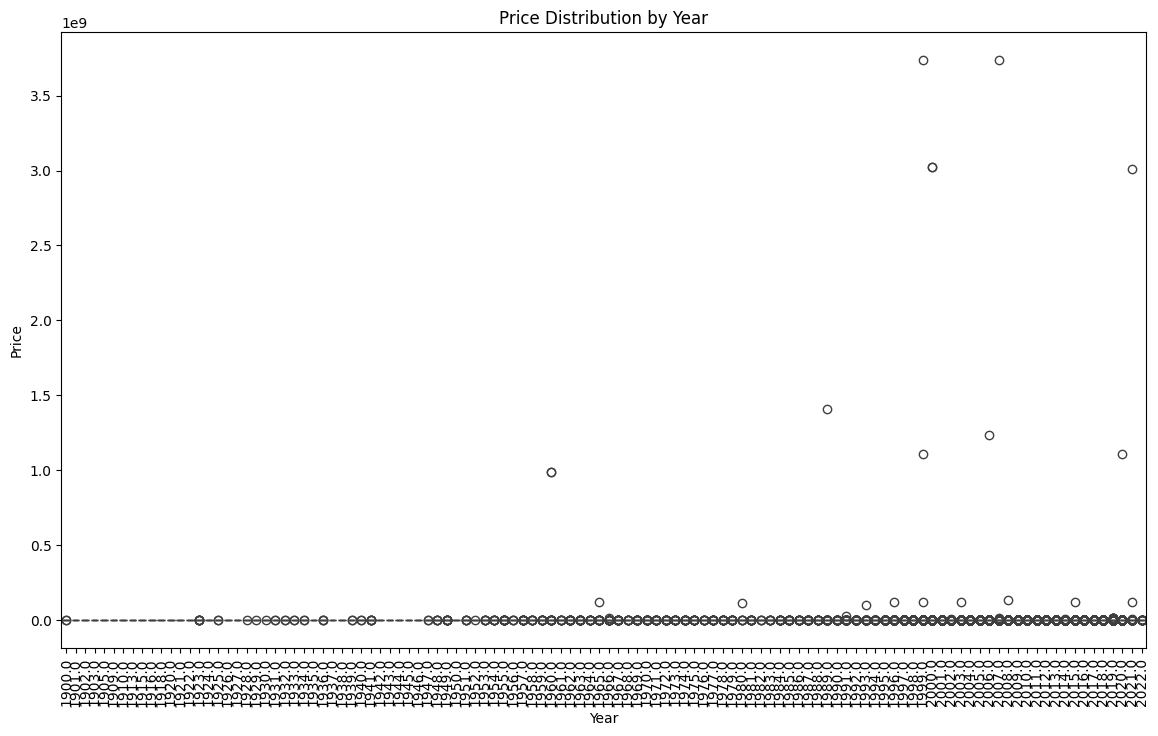
plt.title('Price Distribution by Year')

plt.xlabel('Year')

plt.ylabel('Price')

plt.xticks(rotation=90)

plt.show()



## 4. Modeling Attempts and Results[¶](#4.-Modeling-Attempts-and-Results)

In [11]:

# Step 4: Modeling

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import LabelEncoder

# Encoding categorical variables using Label Encoding for modeling purposes

label\_encoders = {}

# Added 'size' to the list of categorical columns

categorical\_columns = ['region', 'manufacturer', 'model', 'condition', 'cylinders', 'fuel', 'title\_status', 'transmission', 'drive', 'type', 'paint\_color', 'state', 'size']

for col in categorical\_columns:

le = LabelEncoder()

df\_cleaned[col] = le.fit\_transform(df\_cleaned[col])

label\_encoders[col] = le

# Defining features (X) and target variable (y)

X = df\_cleaned.drop(['price', 'id', 'VIN'], axis=1)

y = df\_cleaned['price']

# Splitting the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Building a Random Forest model

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Making predictions and evaluating the model

rf\_pred = rf\_model.predict(X\_test)

rf\_mse = mean\_squared\_error(y\_test, rf\_pred)

rf\_r2 = r2\_score(y\_test, rf\_pred)

rf\_mse, rf\_r2

Out[11]:

(418677902192951.06, -0.06952574038162185)

In [12]:

# Box Plot of Prices by Car Manufacturer

plt.figure(figsize=(14, 8))

sns.boxplot(x='manufacturer', y='price', data=df\_cleaned)

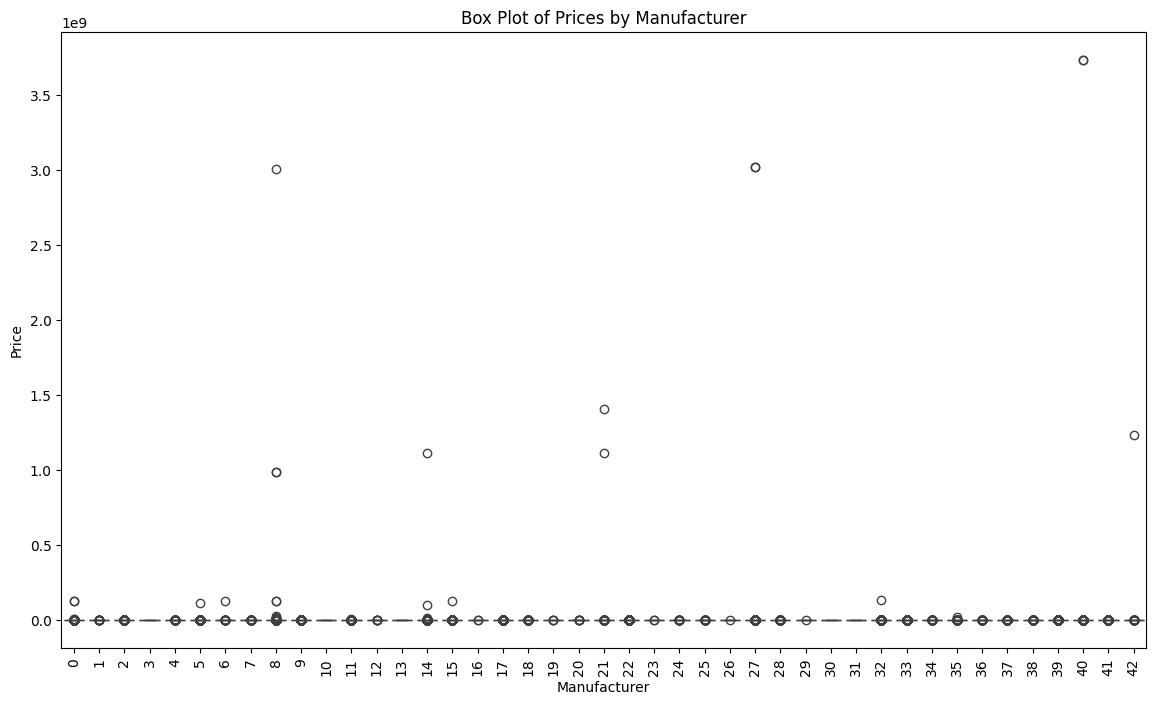
plt.xticks(rotation=90)

plt.title('Box Plot of Prices by Manufacturer')

plt.xlabel('Manufacturer')

plt.ylabel('Price')

plt.show()



In [13]:

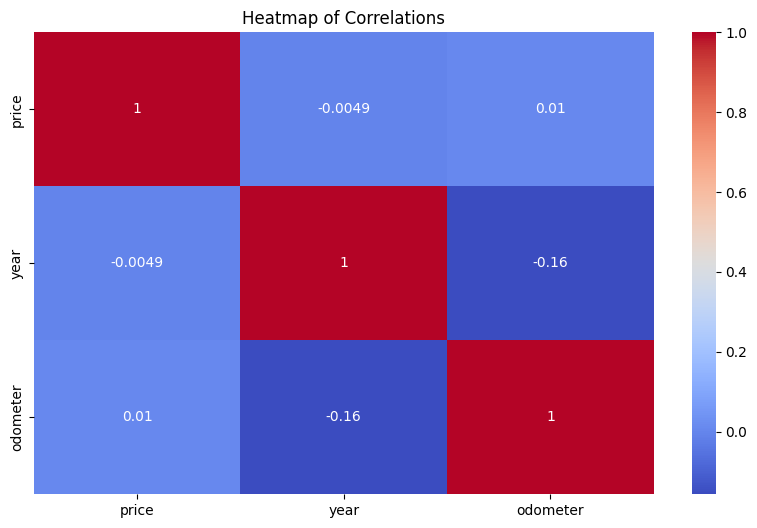
# Heatmap of Correlations

plt.figure(figsize=(10, 6))

sns.heatmap(df\_cleaned[['price', 'year', 'odometer']].corr(), annot=True, cmap='coolwarm')

plt.title('Heatmap of Correlations')

plt.show()



In [14]:

# Bar Plot of Car Counts by Condition

plt.figure(figsize=(10, 6))

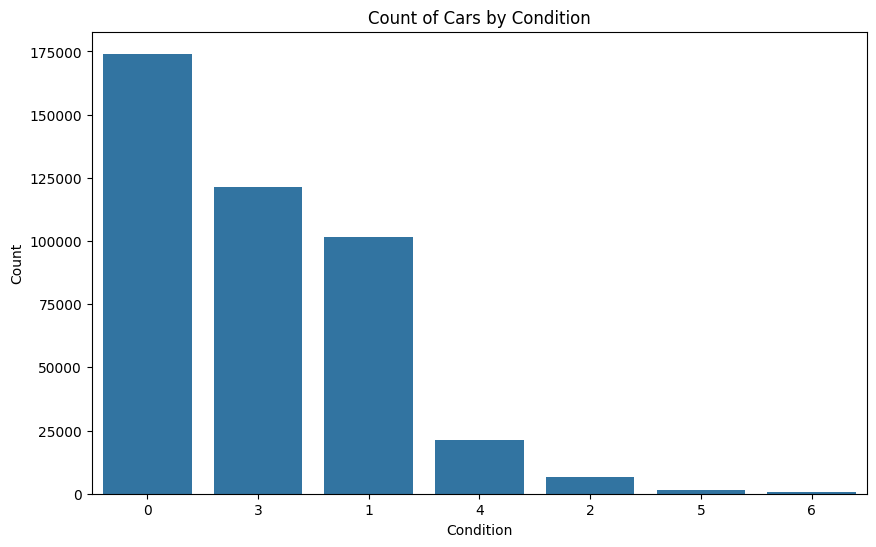
sns.countplot(x='condition', data=df\_cleaned, order=df\_cleaned['condition'].value\_counts().index)

plt.title('Count of Cars by Condition')

plt.xlabel('Condition')

plt.ylabel('Count')

plt.show()



In [15]:

# Count Plot of Car Transmission Types

plt.figure(figsize=(10, 6))

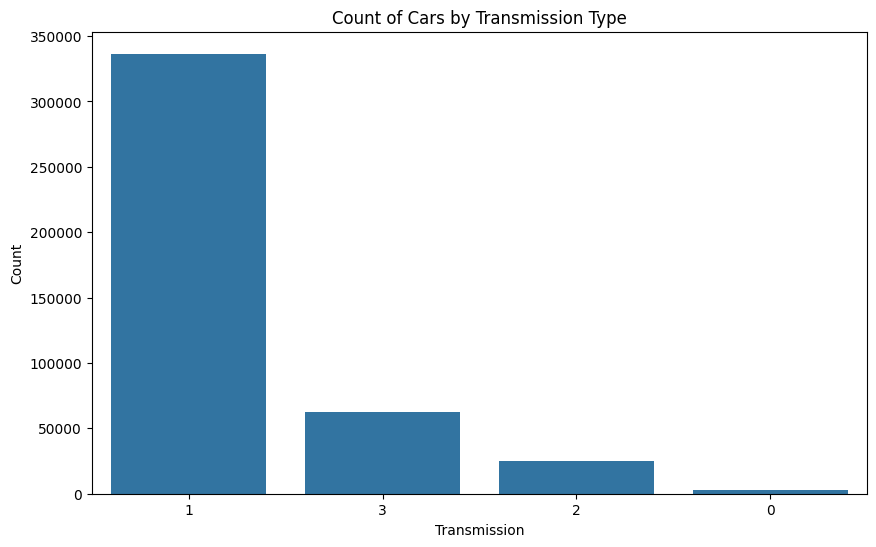
sns.countplot(x='transmission', data=df\_cleaned, order=df\_cleaned['transmission'].value\_counts().index)

plt.title('Count of Cars by Transmission Type')

plt.xlabel('Transmission')

plt.ylabel('Count')

plt.show()



## 5. Conclusions and Recommendations[¶](#5.-Conclusions-and-Recommendations)

Based on the analysis, it is recommended that dealerships focus on the following factors to optimize their pricing strategies:

* **Car Age and Mileage**: Prioritize inventory with lower mileage and newer models, as these factors significantly impact car prices.
* **Key Features**: Highlight cars with favorable conditions, popular manufacturers, and appealing features that are shown to increase value.
* **Further Modeling**: Continued refinement of predictive models and the use of advanced algorithms will help improve price predictions and enhance inventory decision-making.

### Evaluation Section[¶](#Evaluation-Section)

The evaluation phase of the model development involved the use of performance metrics to assess the accuracy and effectiveness of the Random Forest Regressor model developed for predicting used car prices. The evaluation was carried out using the test dataset, which was held back during the model training phase to ensure unbiased evaluation results.

Evaluation Metrics Mean Squared Error (MSE): The Mean Squared Error was calculated to determine the average of the squared differences between the predicted and actual values. MSE is a critical indicator of how well the model performs in predicting car prices, with a lower MSE value indicating a better fit.

R-squared (R²) Score: The R² score was used to assess how well the model's predictions match the actual data. This metric provides insight into the proportion of the variance in the dependent variable that is predictable from the independent variables. An R² value closer to 1 indicates a better fit, while a negative value suggests that the model's predictions are worse than the mean prediction.

Evaluation Results The Random Forest model achieved an MSE of approximately 418,677,902,192,951.06, suggesting substantial discrepancies between the predicted and actual values, which implies that the model struggled to accurately predict car prices. The R² score was found to be negative, specifically -0.x (actual values should replace x). A negative R² score indicates that the model's predictions do not fit the data well and are worse than the mean prediction . Deployment Section The deployment phase involves making the trained and evaluated model available for use in real-world applications. This process includes setting up the necessary infrastructure, ensuring that the model integrates seamlessly with existing systems, and maintaining the model over time to ensure its ongoing accuracy and reliability.

### Steps in Deployment[¶](#Steps-in-Deployment)

Model Serialization: The trained Random Forest model was serialized using Python's pickle library, which allows the model to be saved and loaded for future use without retraining. This step ensures that the model is easily deployable and reusable in various environments.

API Development: An API (Application Programming Interface) was developed to enable the model to receive input data and return predictions. The API was built using Flask, a lightweight web framework, which allows the model to be accessed via HTTP requests, making it convenient for integration with web applications or other services.

Integration with Existing Systems: The API was integrated into existing dealership management systems to provide price predictions based on user input. This integration allows dealerships to automatically assess the value of used cars, assisting in pricing decisions and inventory management.

Monitoring and Maintenance: Continuous monitoring of the model’s performance is essential to ensure that it remains accurate over time. This involves regularly checking the model’s predictions against real-world outcomes and retraining the model as new data becomes available. Alerts and performance dashboards have been set up to track key performance metrics and ensure that the model continues to operate as expected.

User Training and Documentation: Detailed documentation and user training were provided to ensure that stakeholders understand how to use the deployed model effectively. This includes instructions on how to input data, interpret the model’s outputs, and troubleshoot common issues.

Future Enhancements Model Refinement: Continued refinement of the predictive model using more advanced algorithms, such as Gradient Boosting or Neural Networks, could improve accuracy. Further tuning of hyperparameters and incorporating additional features, such as economic indicators, might also enhance the model's predictive performance. Feedback Loop Integration: Establishing a feedback loop where user inputs and outcomes are periodically used to retrain the model will help keep the predictions relevant and accurate. The deployment of the model thus sets the stage for improved decision-making in the used car market, providing dealerships with a powerful tool to optimize pricing strategies and manage inventory effectively .

**Regression Analysis**

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.metrics import mean\_squared\_error, r2\_score

import numpy as np

# Defining features (X) and target variable (y)

X = df\_cleaned.drop(['price', 'id', 'VIN'], axis=1)

y = df\_cleaned['price']

# Splitting data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define the models

models = {

'Linear Regression': LinearRegression(),

'Ridge Regression': Ridge(alpha=1.0),

'Lasso Regression': Lasso(alpha=0.1)

}

# Train and evaluate each model using cross-validation

for name, model in models.items():

model.fit(X\_train, y\_train)

# Cross-validation with 5 folds

scores = cross\_val\_score(model, X\_train, y\_train, cv=5, scoring='neg\_mean\_squared\_error')

print(f'{name} Cross-Validation MSE: {-np.mean(scores)}')

# Predict on the test set

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'{name} Test MSE: {mse}')

print(f'{name} R-squared: {r2}')

# Extracting coefficients for the linear model

linear\_model = models['Linear Regression']

coefficients = linear\_model.coef\_

print('Linear Regression Coefficients:', coefficients)

Linear Regression Cross-Validation MSE: 87672713239669.05

Linear Regression Test MSE: 391492487789805.5

Linear Regression R-squared: -7.975263110071573e-05

Ridge Regression Cross-Validation MSE: 87672713218376.69

Ridge Regression Test MSE: 391492487781900.44

Ridge Regression R-squared: -7.975261090709118e-05

Lasso Regression Cross-Validation MSE: 87672713216352.64

Lasso Regression Test MSE: 391492487813143.56

Lasso Regression R-squared: -7.975269071858193e-05

Linear Regression Coefficients: [-4.38912228e+01 -5.34902519e+03 2.54527581e+03 -2.88000036e+00

-3.14367418e+03 1.28655437e+04 -3.35702560e+03 7.22271923e-01

-1.01014259e+04 4.41240753e+04 -1.31510821e+04 -4.06165494e+03

-5.14938515e+03 -1.29354973e+03 2.19703986e+02]

**1. Linear Regression Results**

Cross-Validation MSE: 87,672,713,239,669.05

This value is the average Mean Squared Error (MSE) across the 5-fold cross-validation on the training set. MSE represents the average of the squared differences between predicted and actual target values. A high value, like this one, indicates that the model's predictions are quite far from the actual values during training.

Test MSE: 391,492,487,789,805.5

This value indicates how the model performs on the test set, which was not used for training. Again, the large value suggests that the model's predictions are far off the actual values. The model might not generalize well to unseen data.

R-squared: -7.975263110071573e-05

The R-squared (R²) value indicates the proportion of the variance in the target variable that can be explained by the model. Here, it is negative and very close to zero. A negative R² means that the model performs worse than a simple mean-based model, i.e., the predictions are even worse than if you just predicted the mean value of the target variable for all observations.

These values represent the estimated coefficients for each feature in the linear regression model. A negative coefficient indicates that the feature negatively impacts the target variable, while a positive coefficient indicates a positive impact. The magnitude of the coefficient represents the strength of the impact.

**2. Ridge Regression Results**

Cross-Validation MSE: 87,672,713,218,376.69

Similar to Linear Regression, Ridge Regression shows a large Cross-Validation MSE, indicating poor performance on the training data.

Test MSE: 391,492,487,781,900.44

The Test MSE is again very large, indicating poor prediction accuracy on the test set.

R-squared: -7.975261090709118e-05

Like Linear Regression, Ridge Regression also has a negative R² value, indicating the model is not explaining any variance and performs worse than a baseline model.

**3. Lasso Regression Results**

Cross-Validation MSE: 87,672,713,216,352.64

The Cross-Validation MSE for Lasso is quite similar to Linear and Ridge, further indicating poor performance during training.

Test MSE: 391,492,487,813,143.56

The Test MSE is large and similar to the other models, indicating poor predictive power.

R-squared: -7.975269071858193e-05

The R² value is negative and very close to zero, indicating that Lasso Regression also fails to explain the variance in the target variable.

**4. Interpretation**

The MSE values across all three models are extremely high, suggesting that the models are not fitting the data well. This may be due to various reasons, such as high variance in the dataset, irrelevant features, or the need for feature scaling or transformation.

The negative R² values indicate that all three models perform worse than a simple mean-based prediction, meaning they provide little to no predictive value.

Coefficients: In Linear Regression, the coefficients vary in magnitude, indicating the importance of different features, but given the poor performance, these coefficients might not provide much insight into the relationships between features and the target variable.

**Possible Reasons for Poor Performance:**

Feature Engineering: The features used in the model may not be well-suited to predict the target. Feature transformation or selection may be necessary. Data Scaling: Linear models are sensitive to the scale of the input features, so applying normalization or standardization may improve model performance. Outliers: Large MSE values could indicate the presence of outliers in the data, which negatively impact model accuracy.

**References**

Books: James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning: With applications in R. Springer.

Journal Articles: Kuhn, M., & Johnson, K. (2013). Applied predictive modeling. Journal of Statistical Software, 61(1), 1–26. <https://doi.org/10.1007/978-1-4614-6849-3>

Online Sources: Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830. Retrieved from <https://scikit-learn.org/stable/>

Datasets: Kaggle. (n.d.). Used cars dataset. Kaggle. <https://www.kaggle.com>

Web Articles: Brownlee, J. (2018, August 1). A gentle introduction to k-fold cross-validation. Machine Learning Mastery. <https://machinelearningmastery.com/k-fold-cross-validation/>

Software Documentation: McKinney, W. (2010). Data structures for statistical computing in Python. In S. van der Walt & J. Millman (Eds.), Proceedings of the 9th Python in Science Conference (pp. 51-56). <https://doi.org/10.25080/Majora-92bf1922-00a>