

# 1 Project Introduction

This project aims to predict the price of used vehicles based on their specifications using different regression algorithms. The dataset includes details such as name, make, model, year, mileage, engine, fuel, transmission, body and color. I cleaned the data, handled missing values and outliers, handled categorical columns with encoding, and scaled features. Then I trained and compared four models — Linear, Polynomial, Ridge, and Lasso Regression — to find which one gives the most accurate price predictions. Finally, I selected the best model based on the  $R^2$  score and error values (MSE and RMSE).

## 2 Importing Libraries and Loading Dataset

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder, PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score, root_mean_squared_error

df = pd.read_csv('VehiclePrice.csv')
df.head()
```

Out[1]:

	name	description	make	model	year	price	engine	cylinders	fuel	mileage	transmission	trim	bod
0	2024 Jeep Wagoneer Series II	\n \n Heated Leather Seats, Nav Sy...	Jeep	Wagoneer	2024	74600.0	24V GDI DOHC Twin Turbo	6.0	Gasoline	10.0	8-Speed Automatic	Series II	SU
1	2024 Jeep Grand Cherokee Laredo	Al West is committed to offering every custome...	Jeep	Grand Cherokee	2024	50170.0	OHV	6.0	Gasoline	1.0	8-Speed Automatic	Laredo	SU
2	2024 GMC Yukon XL Denali	NaN	GMC	Yukon XL	2024	96410.0	6.2L V-8 gasoline direct injection, variable V...	8.0	Gasoline	0.0	Automatic	Denali	SU
3	2023 Dodge Durango Pursuit	White Knuckle Clearcoat 2023 Dodge Durango Pur...	Dodge	Durango	2023	46835.0	16V MPFI OHV	8.0	Gasoline	32.0	8-Speed Automatic	Pursuit	SU
4	2024 RAM 3500 Laramie	\n \n 2024 Ram 3500 Laramie Billet...	RAM	3500	2024	81663.0	24V DDI OHV Turbo Diesel	6.0	Diesel	10.0	6-Speed Automatic	Laramie	Picku Truc

### 3 Intial Data Inspection

```
In [2]: print(df.info())
print(df.shape)
```

```
print(df.describe())  
print(df.isna().sum())
```

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

```
RangeIndex: 1002 entries, 0 to 1001
```

```
Data columns (total 17 columns):
```

#	Column	Non-Null Count	Dtype
0	name	1002 non-null	object
1	description	946 non-null	object
2	make	1002 non-null	object
3	model	1002 non-null	object
4	year	1002 non-null	int64
5	price	979 non-null	float64
6	engine	1000 non-null	object
7	cylinders	897 non-null	float64
8	fuel	995 non-null	object
9	mileage	968 non-null	float64
10	transmission	1000 non-null	object
11	trim	1001 non-null	object
12	body	999 non-null	object
13	doors	995 non-null	float64
14	exterior_color	997 non-null	object
15	interior_color	964 non-null	object
16	drivetrain	1002 non-null	object

```
dtypes: float64(4), int64(1), object(12)
```

```
memory usage: 133.2+ KB
```

```
None
```

```
(1002, 17)
```

	year	price	cylinders	mileage	doors
count	1002.000000	979.000000	897.000000	968.000000	995.000000
mean	2023.916168	50202.985700	4.975474	69.033058	3.943719
std	0.298109	18700.392062	1.392526	507.435745	0.274409
min	2023.000000	0.000000	0.000000	0.000000	2.000000
25%	2024.000000	36600.000000	4.000000	4.000000	4.000000
50%	2024.000000	47165.000000	4.000000	8.000000	4.000000
75%	2024.000000	58919.500000	6.000000	13.000000	4.000000
max	2025.000000	195895.000000	8.000000	9711.000000	5.000000
name	0				
description	56				
make	0				
model	0				
year	0				
price	23				
engine	2				

```
cylinders      105
fuel           7
mileage        34
transmission    2
trim           1
body           3
doors          7
exterior_color  5
interior_color 38
drivetrain      0
dtype: int64
```

## 4 Exploratory Data Analysis

### Univariate Analysis

```
In [3]: #Distribution of Price

#sns.set_palette('pastel')
#sns.set_style('whitegrid')
sns.set_style('whitegrid')
sns.set_palette('pastel')
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
axes[0].hist(df['price'], bins=30, color='green', edgecolor='orange')
axes[0].set_title("Distribution of Price")
axes[0].set_xlabel("Price")
axes[0].set_ylabel("Frequency")
axes[0].tick_params(axis='x', labelrotation=45)

sns.boxplot(df['price'], ax=axes[1])
axes[1].set_title('Boxplot of Price')

plt.tight_layout()
plt.show()

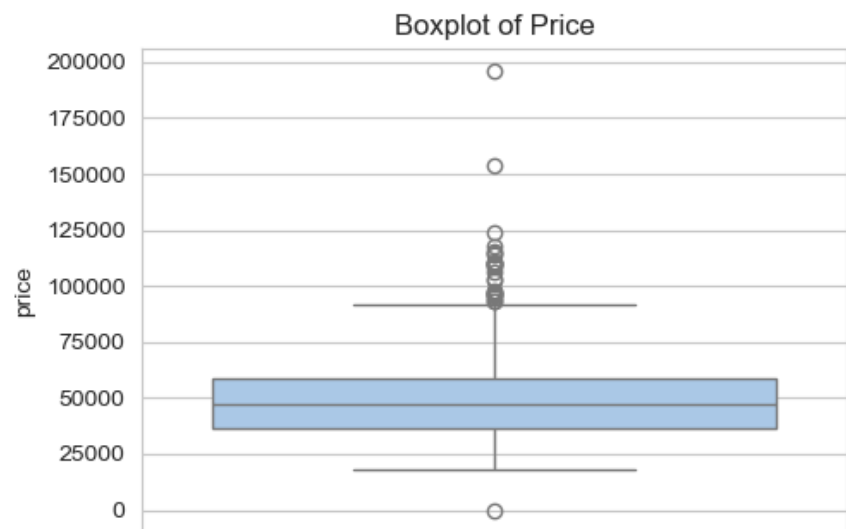
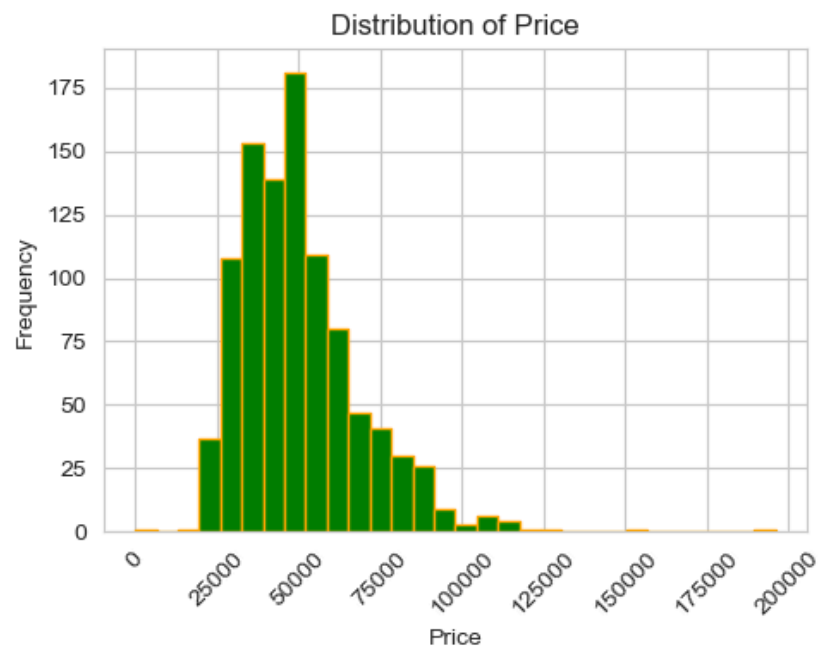
#Distibution of Fuel Type
plt.figure(figsize=(6, 4))
sns.countplot(x='fuel', data=df, palette='pastel', hue='fuel', legend=False)
```

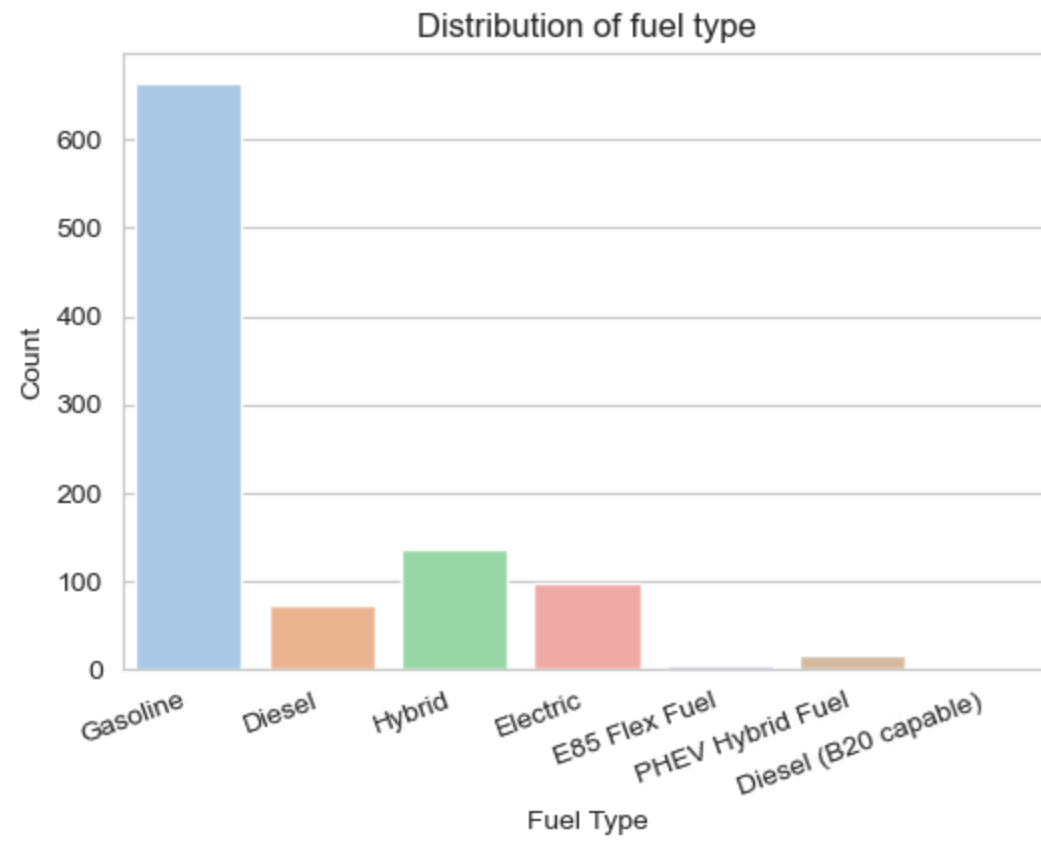
```

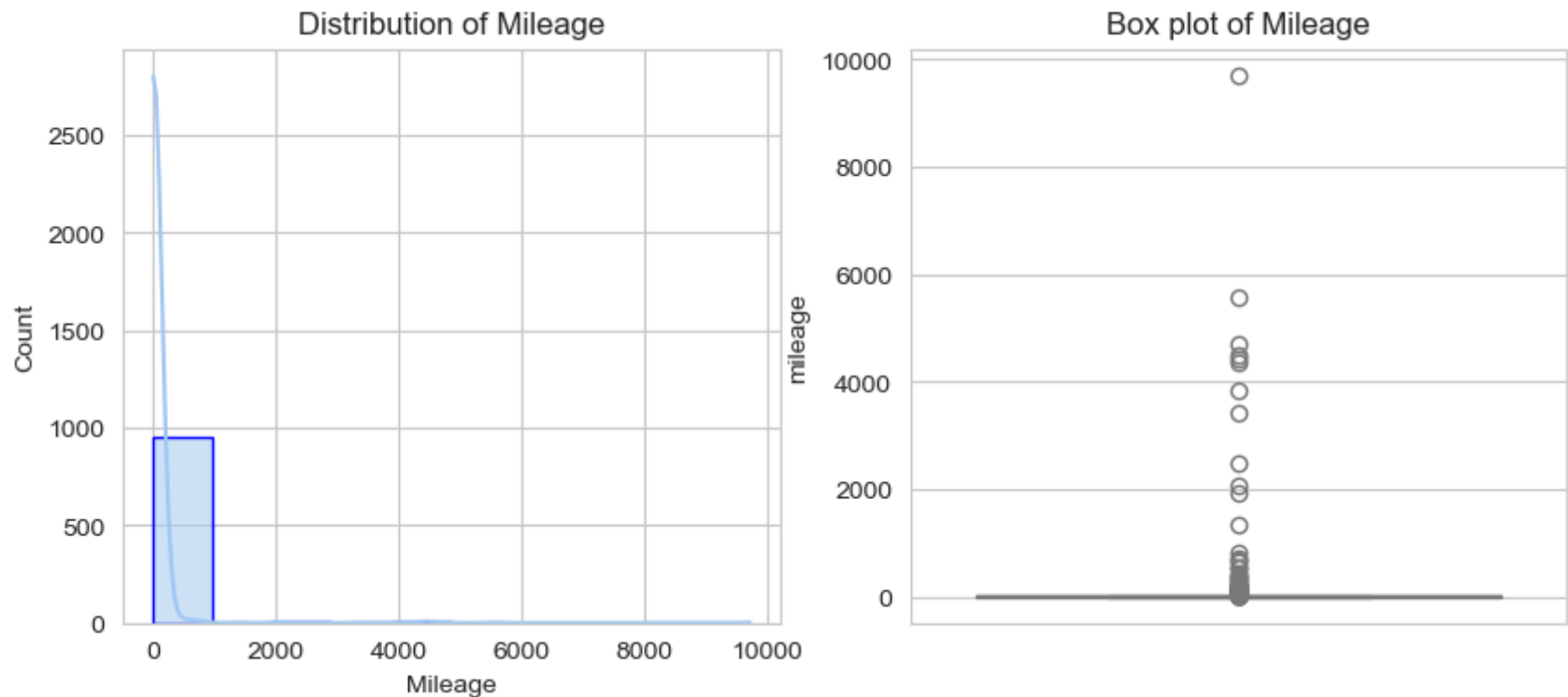
plt.title("Distribution of fuel type")
plt.xlabel("Fuel Type")
plt.ylabel("Count")
plt.xticks(rotation=20,ha='right')
plt.show()
#Distribution of Mileage
fig,axes=plt.subplots(1,2,figsize=(10,4))
sns.histplot(x=df['mileage'],bins=10,ax=axes[0],kde=True,edgecolor='blue')
sns.boxplot(df['mileage'],ax=axes[1])
axes[0].set_title("Distribution of Mileage")
axes[0].set_xlabel("Mileage")
axes[1].set_title("Box plot of Mileage")

plt.show()

```







Price Distribution is right skewed and showing many outliers. So I used Median Imputation. Most Vehicles showed low mileage while very few showed high mileage. Most of the cars are of 'Gasoline' fuel type.

## Bivariate Analysis

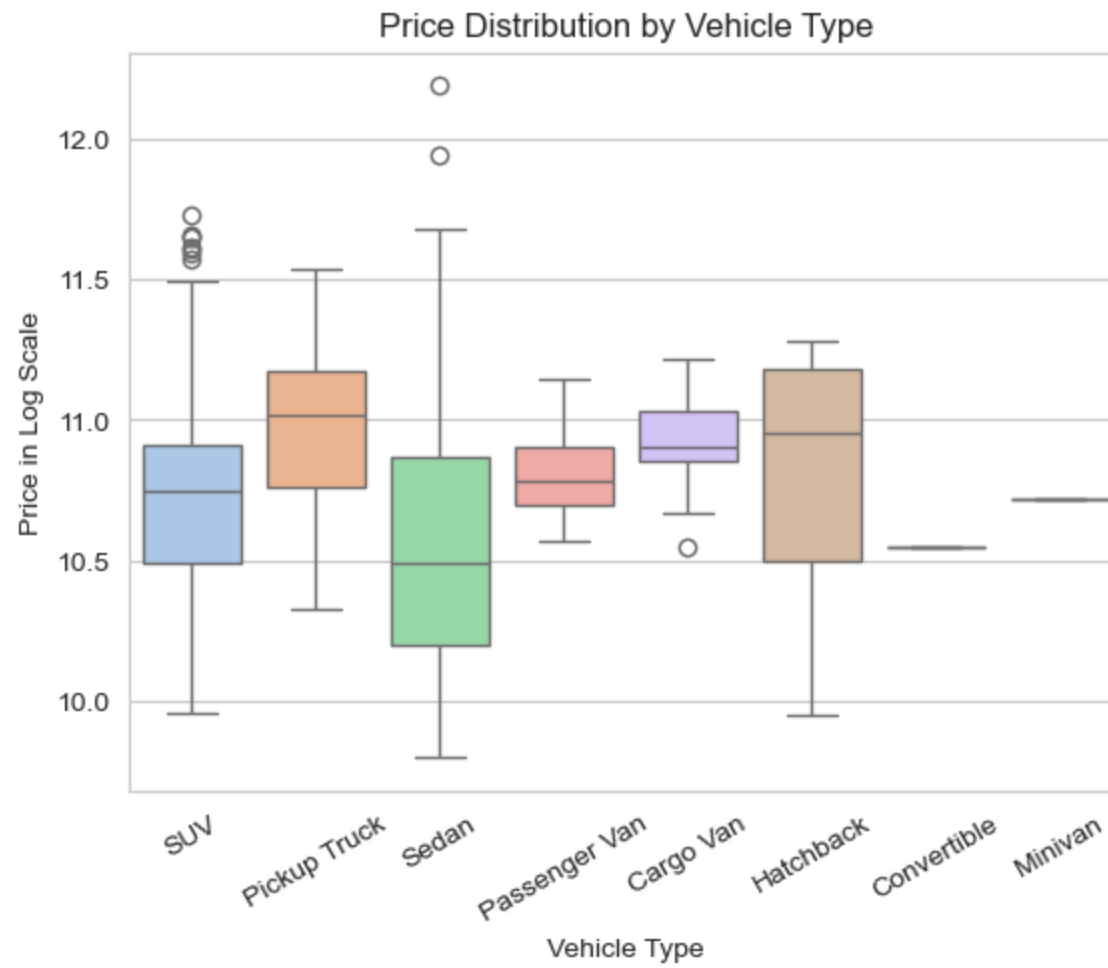
In [7]: *#Relationship Between Price and Vehicle Type*

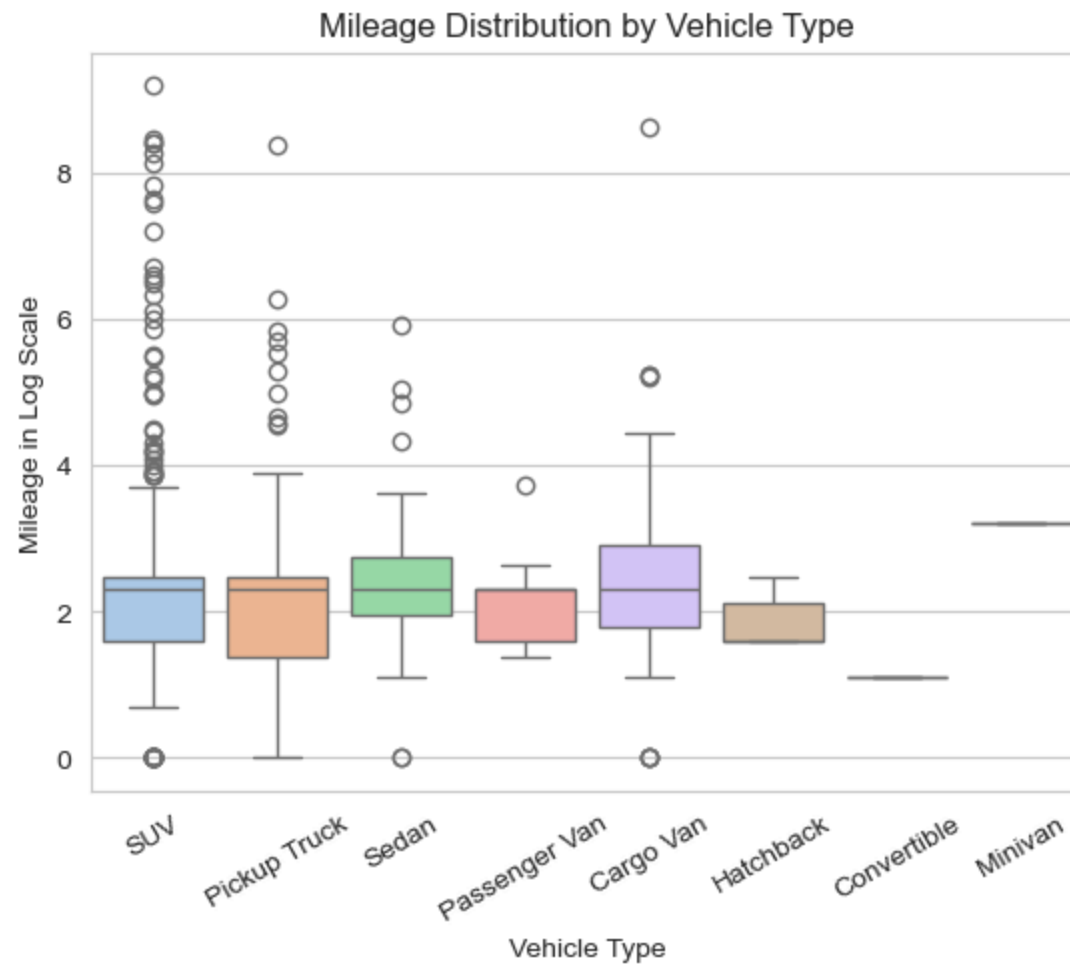
```
sns.boxplot(x='body', y='price', data=df, palette='pastel', hue='body', legend=False)
plt.title("Price Distribution by Vehicle Type")
plt.xlabel("Vehicle Type")
plt.ylabel("Price in Log Scale")
plt.xticks(rotation=30)
plt.show()
```

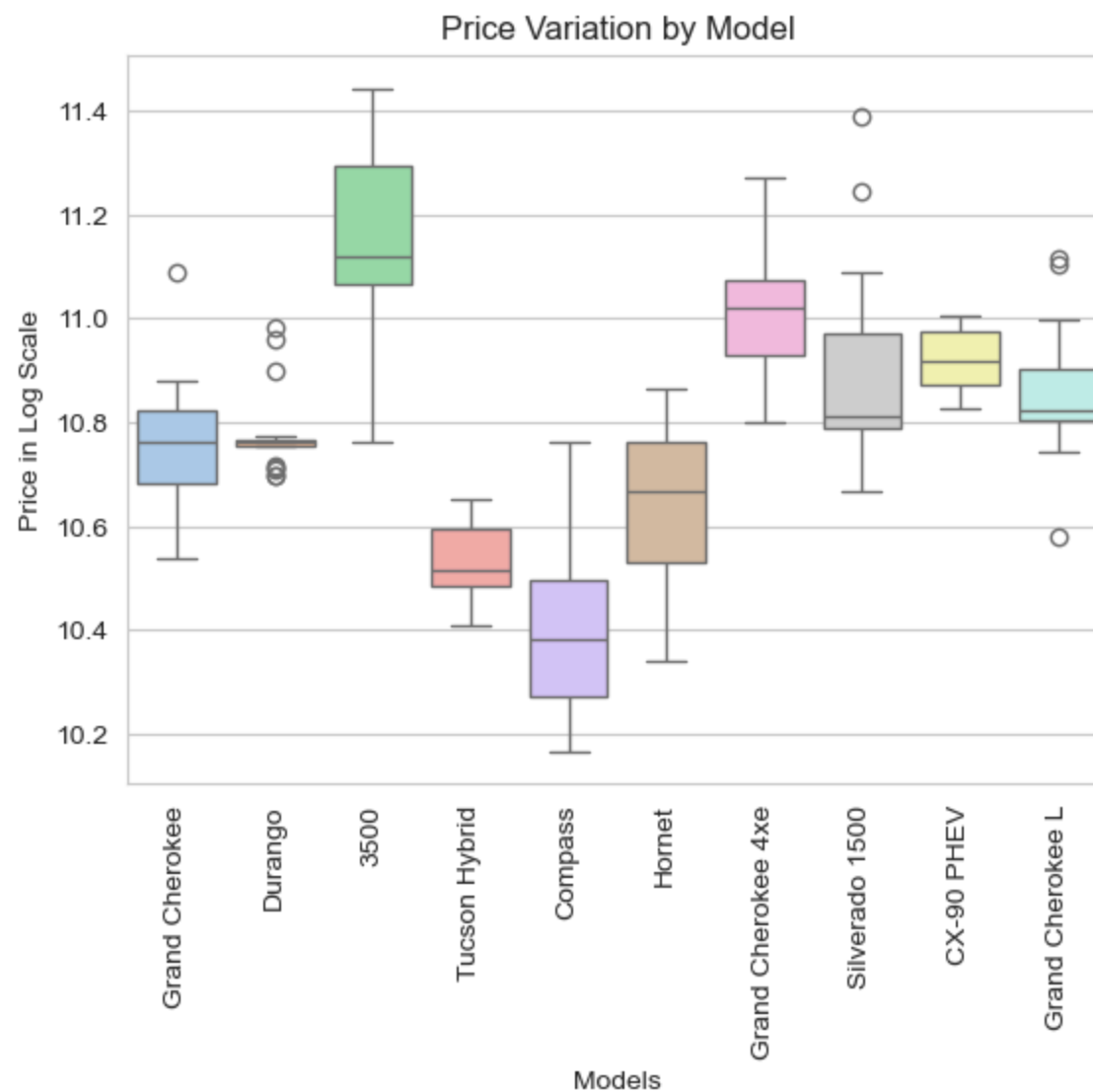
*#Mileage Vs Vehicle Type*

```
sns.boxplot(x='body',y='mileage',data=df,palette='pastel',hue='body',legend=False)
plt.title("Mileage Distribution by Vehicle Type")
plt.xlabel("Vehicle Type")
plt.ylabel("Mileage in Log Scale")
plt.xticks(rotation=30)
plt.show()

#Model Vs Price
top_models=df['model'].value_counts().head(10).index
df_top_models=df[df['model'].isin(top_models)]
sns.boxplot(x='model',y='price',data=df_top_models,hue='model',palette='pastel',legend=False)
plt.title("Price Variation by Model")
plt.xticks(rotation=90)
plt.xlabel("Models")
plt.ylabel("Price in Log Scale")
plt.show()
```







Both price and mileage vary across different vehicle types.

## 5 Handling Missing Values

```
In [4]: missing_pct=df.isna().mean()*100
print("Missing Percentage in each column")
print(round(missing_pct,2))
```

```
Missing Percentage in each column
name                0.00
description          5.59
make                0.00
model               0.00
year               0.00
price               2.30
engine              0.20
cylinders           10.48
fuel                0.70
mileage             3.39
transmission        0.20
trim                0.10
body                0.30
doors               0.70
exterior_color      0.50
interior_color      3.79
drivetrain          0.00
dtype: float64
```

Since missing percentage is less than 15% in most of the columns, I can do simple imputation.

Description column had some newline character which I replaced with spaces and trimmed afterwards then I filled the missing values in this column with a default string. Price Distribution showed outliers, so I used median imputation. Fuel type countplot showed most cars are gasoline fuel type. So imputed the missing values with 'Gasoline' which is the Frequent one.

I found cylinders missing mostly for Electric Fuel type, filled those rows with 0 for Cylinders (since electric car don't have cylinders). Rows with other fuel type is filled with mode value(4).

Mileage is highly right skewed with few extreme values. So I used Median Imputation. Mileage values with 0 replaced with nan and then done median imputation (since used cars can't have mileage 0).

```
In [5]: df['description']=df['description'].str.replace('\n',' ')
df['description']=df['description'].str.strip()
df['description']=df['description'].fillna('No Description Available')
```

```
df['price']=df['price'].fillna(df['price'].median())
missing_cyls=df[df['cylinders'].isna()]
display(missing_cyls.head(10))
print(missing_cyls.fuel.value_counts())
mask_electric=(df['fuel']=='Electric') & (df['cylinders'].isna())
df.loc[mask_electric,'cylinders']=0
cyl_mode=df['cylinders'].mode()[0]
df['cylinders']=df['cylinders'].fillna(cyl_mode)

df['fuel']=df['fuel'].fillna(df['fuel'].mode()[0])
df['mileage']=df['mileage'].replace(0,np.nan)
df['mileage']=df['mileage'].fillna(df['mileage'].median())
cols_to_fill=['transmission','trim','body','doors','exterior_color','interior_color','engine']
for col in cols_to_fill:
    df[col]=df[col].fillna(df[col].mode()[0])

print(df.isna().sum())
```

	name	description	make	model	year	price	engine	cylinders	fuel	mileage	transmission	ti
14	2024 Chevrolet Blazer EV 2LT	Sterling Gray Metallic 2024 Chevrolet Blazer E...	Chevrolet	Blazer EV	2024	51695.0	c	NaN	Electric	4.0	1-Speed Automatic	
28	2024 Chevrolet Blazer EV 2LT	Radiant Red Tintcoat 2024 Chevrolet Blazer EV ...	Chevrolet	Blazer EV	2024	52190.0	c	NaN	Electric	6.0	1-Speed Automatic	
33	2024 Kia EV6 GT	Yacht Blue 2024 Kia EV6 GT AWD 1-Speed Automat...	Kia	EV6	2024	49820.0	c	NaN	Electric	13.0	Automatic	
35	2024 Ford Mustang Mach-E Premium	2024 Ford Mustang Mach-E Premium 300A 99/86 Ci...	Ford	Mustang Mach-E	2024	47790.0	c	NaN	Electric	5.0	1-Speed Automatic	Premi
49	2024 Hyundai IONIQ 5 SE Standard Range	Vehicle pricing includes all offers and incent...	Hyundai	IONIQ 5	2024	44195.0	c	NaN	Electric	14.0	1-Speed Automatic	Stand Rai
50	2024 Audi Q8 e-tron S line Premium	Glacier White Metallic 2024 Audi Q8 e-tron Pre...	Audi	Q8 e-tron	2024	86670.0	c	NaN	Electric	5.0	Automatic	S Premi
53	2024 Kia EV9 Light Long Range	VISION KIA CANANDAIGUA.Aurora Black Pearl 2024...	Kia	EV9	2024	52388.0	c Motor	NaN	Electric	1.0	1-Speed Automatic	Li Lc Rai
62	2024 Hyundai IONIQ 5 Limited	Vehicle pricing includes all offers and incent...	Hyundai	IONIQ 5	2024	55365.0	c	NaN	Electric	10.0	1-Speed Automatic	Limi

	name	description	make	model	year	price	engine	cylinders	fuel	mileage	transmission	ti
68	2024 Hyundai IONIQ 5 SE	Country Hyundai is part of the TommyCar Auto G...	Hyundai	IONIQ 5	2024	42015.0	c	NaN	Electric	7.0	1-Speed Automatic	
86	2024 Mercedes- Benz EQE 350+ Base	No Description Available	Mercedes- Benz	EQE 350+	2024	76535.0	c	NaN	Electric	375.0	1-Speed Automatic	B

```

fuel
Electric    97
Gasoline     1
Name: count, dtype: int64
name        0
description  0
make        0
model       0
year        0
price       0
engine      0
cylinders   0
fuel        0
mileage     0
transmission 0
trim        0
body        0
doors       0
exterior_color 0
interior_color 0
drivetrain  0
dtype: int64

```

## 6 Detecting and Handling Outliers

```

In [6]: cols=['price','mileage']
        for col in cols:

```

```

q1=df[col].quantile(0.25)
q3=df[col].quantile(0.75)
iqr=q3-q1
upper_fence=q3+1.5*iqr
lower_fence=q1-1.5*iqr
mask=(df[col]>upper_fence) | (df[col]<lower_fence)
print(f"Number of outliers in {col}:{mask.sum()}")
for col in cols:
    df[col]=df[col].replace(0,np.nan)
    df[col]=df[col].fillna(df[col].median())
    df[col]=np.log(df[col])
df=df.drop(columns=['description','name'])

```

Number of outliers in price:27

Number of outliers in mileage:108

I observed that outliers in price represent luxury or high-end cars, while outliers in mileage likely correspond to older vehicles. Therefore, I decided not to cap or remove these outliers, as they reflect valid real-world cases. Instead, to reduce the right skewness, I applied a logarithmic transformation to price and mileage.

I removed description column since it contains long text which is not useful for my regression model. I also removed name column since it contains many unique categories and its information is already captured in other columns(make,model,year,trim).

```

In [8]: print(df['engine'].value_counts())
df=df.drop(columns=['engine'])

```

```

engine
16V GDI DOHC Turbo      132
c                        95
16V GDI DOHC Turbo Hybrid  94
24V MPFI DOHC           56
16V MPFI OHV            48
...
ne 3L I-6 gasoline direct injection, DOHC, variable valve
ream 2.5L I-4 port/direct injection, DOHC, CVT variable
6 DOHC, VVT variable valve control, engine with cylinder
24V DOHC                1
8 gasoline direct injection, variable valve control, regu
Name: count, Length: 100, dtype: int64

```

I found that engine column had inconsistent data-some rows had full text while others had single letters like 'c'.So I dropped it

## 7 Encoding Categorical Variables

```
In [9]: print("No of Unique categories in each of the categorical columns:")
print(df.select_dtypes(include=object).nunique())
print("Average Price Variation Across different Exterior Colors:")
print(df.groupby('exterior_color')['price'].mean().sort_values(ascending=False).head(10))
print("Average Price Variation Across different Interior Colors:")
print(df.groupby('interior_color')['price'].mean().sort_values(ascending=False).head(10))
top_models=df['model'].value_counts().head(15)
df['model']=df['model'].apply(lambda x:x if x in top_models else 'Other')
cols_to_drop=['trim','exterior_color','interior_color']
df=df.drop(columns=cols_to_drop)
print(df.select_dtypes(include=object).nunique())
cols_to_encode=['make','model','fuel','transmission','body','drivetrain']
for col in cols_to_encode:
    ohe=OneHotEncoder(sparse_output=False,drop='first')
    encoded_cols=ohe.fit_transform(df[[col]])
    new_col_names=ohe.get_feature_names_out([col])
    encoded_cols_df=pd.DataFrame(encoded_cols,columns=new_col_names)
    df=pd.concat([df,encoded_cols_df],axis=1)
    df.drop(columns=[col],inplace=True)

print(df.select_dtypes(include=object))
```

No of Unique categories in each of the categorical columns:

make	28
model	153
fuel	7
transmission	38
trim	197
body	8
exterior_color	263
interior_color	91
drivetrain	4

dtype: int64

Average Price Variation Across different Exterior Colors:

exterior\_color

Tactical Green Metallic	11.942693
Alpine Gray	11.645356
Gray Metallic	11.485409
Black Sapphire	11.462667
Tanzanite Blue II Metallic	11.460526
Mineral White	11.427640
Aventurin Red Metallic	11.418010
Sparkling	11.392959
Sterling Metallic	11.374525
Black Sapphire Metallic	11.365853

Name: price, dtype: float64

Average Price Variation Across different Interior Colors:

interior\_color

Caramel	12.185334
Teak/Light Shale	11.476365
Silverstone	11.460526
Alpine Umber	11.412202
Dark Walnut / Slate	11.374525
Tupelo	11.369809
Java	11.367784
Sea Salt	11.363230
Pearl Beige	11.352010
Lt Mountain Brown/Brown	11.331092

Name: price, dtype: float64

make	28
model	16
fuel	7
transmission	38
body	8

```
drivetrain      4
dtype: int64
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, ...]

[1002 rows x 0 columns]
```

I checked the number of unique categories in each of the categorical columns and found that only make,fuel,transmission,body,drivetrain is having low cardinality.Hence I decided to apply One Hot Encoding to these columns. Columns such as model,trim,exterior\_color,interior\_color have very high cardinality, so I avoided One Encoding for these columns. From The boxplot showing variation in price w.r.t models, I observed that price varies significantly with different models.So I retained the model column.However I filtered top 15 models based on frequency and replaced all other model names with 'Other'.This resulted in a total of 16 unique categories in the model column, which I can easily handle for One Hot Encoding. Trim column represents variant of same model and contains 197 unique characters which is difficult to handle and seems redundant with 'model'.So I dropped it. I grouped the rows based on different exterior and interior colors and compared their average prices.Average prices across different exterior and interior colors are almost similar,indicating that color does not have significant impact on price.So I decided to drop both of these columns. While performing One Hot Encoding, I set the parameter drop='first' to avoid multicollinearity among the encoded dummy variables.

## 8 Feature Scaling

```
In [10]: y=df['price']
x=df.drop(columns=['price'])
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
```

The dataset was split into 80% training and 20% testing sets. All numerical features were scaled using StandardScaler.

# 9 Model Building

## Linear Regression

```
In [11]: lin_reg_mdl=LinearRegression()  
lin_reg_mdl.fit(x_train,y_train)  
y_pred_lin=lin_reg_mdl.predict(x_test)  
mse_lr=mean_squared_error(y_test,y_pred_lin)  
rmse_lr=root_mean_squared_error(y_test,y_pred_lin)  
r2_lr=r2_score(y_test,y_pred_lin)
```

## Polynomial Regression

```
In [12]: poly=PolynomialFeatures(degree=2,include_bias=False)  
x_train_poly=poly.fit_transform(x_train)  
x_test_poly=poly.transform(x_test)  
poly_reg=LinearRegression()  
poly_reg.fit(x_train_poly,y_train)  
y_pred_poly=poly_reg.predict(x_test_poly)  
mse_poly=mean_squared_error(y_test,y_pred_poly)  
rmse_poly=root_mean_squared_error(y_test,y_pred_poly)  
r2_poly=r2_score(y_test,y_pred_poly)
```

## Ridge Regression

```
In [13]: ridge_mdl=Ridge()  
param_grid={'alpha':np.logspace(-4,3,20)}  
gcv=GridSearchCV(estimator=ridge_mdl,  
                  cv=5,  
                  param_grid=param_grid,  
                  n_jobs=-1,  
                  refit=True,  
                  scoring='neg_mean_squared_error')
```

```

gcv.fit(x_train,y_train)
best_alpha=gcv.best_params_['alpha']
best_ridge=gcv.best_estimator_
best_score=gcv.best_score_
y_pred_ridge=best_ridge.predict(x_test)
mse_ridge=mean_squared_error(y_test,y_pred_ridge)
rmse_ridge=root_mean_squared_error(y_test,y_pred_ridge)
r2_ridge=r2_score(y_test,y_pred_ridge)

```

## Lasso Regression

```

In [14]: lasso_md1=Lasso(max_iter=1000)
param_grid={'alpha':np.logspace(-4,3,20)}
gcv=GridSearchCV(estimator=lasso_md1,
                  param_grid=param_grid,
                  cv=5,
                  n_jobs=-1,
                  scoring='neg_mean_squared_error',
                  refit=True)
gcv.fit(x_train,y_train)
best_alpha=gcv.best_params_['alpha']
best_lasso=gcv.best_estimator_
best_score=gcv.best_score_
y_pred_lasso=best_lasso.predict(x_test)
mse_lasso=mean_squared_error(y_test,y_pred_lasso)
rmse_lasso=root_mean_squared_error(y_test,y_pred_lasso)
r2_lasso=r2_score(y_test,y_pred_lasso)

kept=[(name,coef) for name,coef in zip(x.columns,best_lasso.coef_) if coef!=0]
removed=[(name,coef) for name,coef in zip(x.columns,best_lasso.coef_) if coef==0]

print("Features kept by Lasso:")
for name,coef in kept:
    print(f"{name}:{coef:.3f}")
print("\n\nFeatures Removed:")
for name,coef in removed:
    print(f"{name}:{coef:.3f}")

```

Features kept by Lasso:

year:0.010  
cylinders:0.223  
mileage:0.013  
doors:0.027  
make\_Audi:0.063  
make\_BMW:0.089  
make\_Cadillac:0.025  
make\_Chevrolet:-0.026  
make\_Chrysler:0.003  
make\_Dodge:-0.024  
make\_Ford:-0.002  
make\_GMC:0.004  
make\_Genesis:0.012  
make\_Honda:-0.003  
make\_Hyundai:-0.017  
make\_INFiniti:0.029  
make\_Jaguar:0.019  
make\_Jeep:0.089  
make\_Kia:-0.029  
make\_Land Rover:0.019  
make\_Lexus:0.015  
make\_Lincoln:0.017  
make\_Mazda:0.004  
make\_Mercedes-Benz:0.081  
make\_Nissan:-0.027  
make\_RAM:0.059  
make\_Subaru:0.005  
make\_Toyota:0.006  
make\_Volkswagen:-0.012  
make\_Volvo:0.020  
model\_CX-90 PHEV:0.046  
model\_Compass:-0.119  
model\_Durango:-0.032  
model\_Grand Cherokee:-0.070  
model\_Grand Cherokee 4xe:-0.006  
model\_Grand Cherokee L:-0.049  
model\_Hornet:0.009  
model\_IONIQ 5:-0.015  
model\_Other:0.007  
model\_Santa Cruz:0.015  
model\_Silverado 1500:0.018

model\_Sportage:0.006  
model\_Taos:-0.007  
model\_Tucson Hybrid:-0.024  
model\_Wrangler 4xe:-0.010  
fuel\_Diesel (B20 capable):0.005  
fuel\_E85 Flex Fuel:-0.016  
fuel\_Electric:0.150  
fuel\_Gasoline:-0.097  
fuel\_Hybrid:-0.028  
fuel\_PHEV Hybrid Fuel:-0.017  
transmission\_1-Speed Automatic:0.024  
transmission\_1-Speed CVT with Overdrive:0.001  
transmission\_10-Speed Automatic:0.043  
transmission\_10-Speed Automatic with Overdrive:0.001  
transmission\_10-Speed Shifttable Automatic:0.000  
transmission\_6-Spd Aisin F21-250 PHEV Auto Trans:0.020  
transmission\_6-Speed Automatic Electronic with Overdrive:-0.003  
transmission\_62 kWh battery:-0.004  
transmission\_7-Speed Automatic S tronic:0.004  
transmission\_7-Speed Automatic with Auto-Shift:-0.005  
transmission\_7-Speed DSG Automatic with Tiptronic:-0.004  
transmission\_7-Speed DSGA? Automatic w/ 4M0:-0.026  
transmission\_8 Speed Dual Clutch:-0.007  
transmission\_8-Speed A/T:0.002  
transmission\_8-Speed Automatic:0.003  
transmission\_8-Speed Automatic Sport:-0.013  
transmission\_8-Speed Automatic with Auto-Shift:0.021  
transmission\_8-Speed Automatic with Tiptronic:-0.013  
transmission\_8-speed automatic:0.005  
transmission\_9 Spd Automatic:-0.009  
transmission\_9-Speed 948TE Automatic:-0.003  
transmission\_9-Speed A/T:-0.001  
transmission\_9-Speed Automatic:-0.002  
transmission\_A/T:0.001  
transmission\_Aisin 6-Speed Automatic:0.003  
transmission\_Automatic:-0.001  
transmission\_Automatic CVT:-0.024  
transmission\_CVT:-0.015  
transmission\_CVT with Xtronic:-0.013  
transmission\_Variable:-0.003  
transmission\_automatic w/paddle shifters:0.004  
body\_Hatchback:-0.036

```
body_Passenger Van:-0.013
body_Pickup Truck:-0.106
body_SUV:-0.044
body_Sedan:-0.055
drivetrain_Four-wheel Drive:0.053
drivetrain_Front-wheel Drive:-0.057
drivetrain_Rear-wheel Drive:-0.011
```

Features Removed:

```
make_Buick:-0.000
transmission_6-SPEED AUTOMATIC:0.000
transmission_6-Speed A/T:0.000
transmission_6-Speed Automatic:0.000
transmission_6-Speed DCT Automatic:0.000
transmission_8-Speed Shiftable Automatic:-0.000
transmission_9-speed automatic:0.000
body_Convertible:0.000
body_Minivan:-0.000
```

## 10 Model Evaluation

```
In [17]: model_comparison=pd.DataFrame({'Model':['Linear Regression','Polynomial Regression','Ridge Regression','Lasso Regression'],
'MSE':[mse_lr,mse_poly,mse_ridge,mse_lasso],
'RMSE':[rmse_lr,rmse_poly,rmse_ridge,rmse_lasso],
'R2-Score':[r2_lr,r2_poly,r2_ridge,r2_lasso]})

model_comparison
```

```
Out[17]:
```

	Model	MSE	RMSE	R2-Score
0	Linear Regression	2.363595e+19	4.861682e+09	-2.258841e+20
1	Polynomial Regression	2.709145e+22	1.645948e+11	-2.589076e+23
2	Ridge Regression	1.879664e-02	1.371008e-01	8.203642e-01
3	Lasso Regression	1.841635e-02	1.357069e-01	8.239986e-01

```
In [18]: sns.barplot(x='Model',y='R2-Score',data=model_comparison,hue='Model')
plt.ylim(-1,1)
plt.title("R2 score comparison of Models")
plt.xticks(rotation=20)
plt.show()
```



I evaluated the performance of all four regression models using metrics such as MSE, RMSE, R2 score. Linear and Polynomial Regression showed very large RMSE (indicating large prediction error) and negative R2 values, which shows these models are not suitable for this dataset. Ridge and Lasso Regression performed significantly better. Both models achieved very low RMSE and

higher R2 scores. Among them Lasso Regression achieved the best performance with highest R2 score and lowest RMSE making it the most accurate and reliable model for this vehicle prediction task.

## Challenges Faced and Solutions implemented

### 1. Selecting the Right Dataset

I downloaded 2–3 vehicle price datasets from Kaggle. Some were too clean and did not allow me to demonstrate proper data cleaning steps.

Solution: I selected a dataset that required moderate cleaning, so I could show essential preprocessing steps such as handling missing values, removing messy text, and encoding categorical variables.

### 2. Time-Consuming Data Cleaning

The chosen dataset contained many missing values, messy descriptive text fields, and inconsistent formatting. Cleaning took longer than expected.

Solution: I performed step-by-step cleaning:

Imputed missing values appropriately

Removed unnecessary long text columns

### 3. Handling Outliers in Price and Mileage

Significant outliers were detected in the price and mileage columns. Initially, I considered capping/flooring, but doing so would artificially limit high-priced or high-mileage cars (which may represent luxury or older vehicles in real life).

Solution: To preserve the natural distribution, I applied a log transformation, which reduced skewness while keeping the true variation in prices and mileage.

### 4. Removing Irrelevant Long Text Columns

Columns like description and name contained long, messy text that did not contribute meaningful information for predicting price.

Solution: I dropped these columns after confirming they did not add value and would unnecessarily increase noise in the dataset.

## 5. Managing High-Cardinality Categorical Columns

Columns such as model, trim, exterior\_color, interior\_color had very high cardinality (many unique values). Applying One-Hot Encoding on them would lead to feature explosion and increase model complexity.

Solution:

I first checked how these columns relate to price.

From the boxplot showing price variation across different models, I observed that the vehicle model has a strong influence on price. Therefore, I decided to retain the model column. However, since it originally contained more than 150 unique values, I filtered the top 15 most frequent models and grouped all remaining models under the category "Other." This reduced the model column to 16 unique categories, making it easier and more efficient to apply One Hot Encoding.

Trim was found redundant with model, so I dropped it.

I grouped and analyzed prices by exterior and interior color and found the average prices were almost the same. So color had no significant impact, and I dropped these columns too.

For the remaining categorical columns (make, fuel, transmission, body, drivetrain), I applied One-Hot Encoding.

While encoding, I used drop='first' to avoid multicollinearity among dummy variables.

## Conclusion

In this project, various regression models were built and compared to predict the price of used vehicles based on multiple features. After performing data preprocessing steps including handling outliers, encoding categorical variables, and feature scaling, four regression models — Linear, Polynomial, Ridge, and Lasso were trained and evaluated.

Base on the evaluation metrics (MSE, RMSE, and  $R^2$  Score), Lasso Regression achieved the best performance, followed closely by Ridge Regression. In contrast, Linear and Polynomial Regression showed high error values and negative  $R^2$  scores, indicating that they were not suitable for this dataset. Overall, the regularized models (Ridge and Lasso) effectively reduced overfitting and improved prediction stability. Therefore, Lasso Regression was selected as the final model for predicting vehicle prices.

