Car Price Prediction Multiple Linear Regression

Data Understanding

This step is necessary for us to understand the data that we are going to deal with our model. Understanding the types of features and samples are necessary for data processing for it to be effective in the later stages of developing our model.

Import Libraries

In [123…

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

Load Dataset

In [123…

```
pd.set_option('display.max_columns', None)
pd.set_option('future.no_silent_downcasting', True) # future downcasting behavior g
cars = pd.read_csv('CarPrice_Assignment.csv')
cars.head()
```

Out[123…

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewhe
0	1	3	alfa-romero giulia	gas	std	two	convertible	rv
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rv
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rv
3	4	2	audi 100 ls	gas	std	four	sedan	fv
4	5	2	audi 100ls	gas	std	four	sedan	4v

Reviewing the Dataset

1. Determine the dimension of the dataset

In [123…

```
rows, columns = cars.shape
print(f"Number of rows: {rows}")
print(f"Number of columns: {columns}")
```

```
Number of rows: 205
Number of columns: 26
```

2. Analyze the dataset information

In [124…

```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

3. Convert "symboling" as catergorical data

After referring to the data dictionary, symbolling is a categorical data so we need to convert it.

In [124…
```
cars['symboling'] = cars['symboling'].astype('object')
```

3. Determine continuous and categorical features of your dataset

Categorical: object

Numerical: float64, int64

In [124…
```
categorical = cars.select_dtypes(include=['object']).columns
continuous = cars.select_dtypes(include=['float64', 'int64']).columns

print(f"Number of categorical features: {len(categorical)}")
print(f"Number of continuous features: {len(continuous)}")
```

```
Number of categorical features: 11
Number of continuous features: 15
```

In [124…
```
print("\nCategorical Features:")
print(categorical)
print("\nContinuous Features:")
print(continuous)
```

```
Categorical Features:
Index(['symboling', 'CarName', 'fueltype', 'aspiration', 'doornumber',
       'carbody', 'drivewheel', 'enginelocation', 'enginetype',
       'cylindernumber', 'fuelsystem'],
      dtype='object')

Continuous Features:
Index(['car_ID', 'wheelbase', 'carlength', 'carwidth', 'carheight',
       'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio',
       'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price'],
      dtype='object')
```

4. Categorical Data Descriptions

In [124…
```
cars.describe(include='object')
```

Out[124…

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enc
count	205	205	205	205	205	205	205	
unique	6	147	2	2	2	5	3	
top	0	peugeot 504	gas	std	four	sedan	fwd	
freq	67	6	185	168	115	96	120	

5. Continuous Data Descriptions

In [124…
```
cars.describe(include=['float64', 'int64']).round(2)
```

Out[124…

	car_ID	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	borerati
count	205.00	205.00	205.00	205.00	205.00	205.00	205.00	205.0
mean	103.00	98.76	174.05	65.91	53.72	2555.57	126.91	3.3
std	59.32	6.02	12.34	2.15	2.44	520.68	41.64	0.2
min	1.00	86.60	141.10	60.30	47.80	1488.00	61.00	2.5
25%	52.00	94.50	166.30	64.10	52.00	2145.00	97.00	3.1
50%	103.00	97.00	173.20	65.50	54.10	2414.00	120.00	3.3
75%	154.00	102.40	183.10	66.90	55.50	2935.00	141.00	3.5
max	205.00	120.90	208.10	72.30	59.80	4066.00	326.00	3.9

6. Checking if there are NaN Values in the Dataset

In [124…
```
cars.isnull().sum()
```

Out[124…
```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

7. Checking for Duplicate Values

In [124…
```
print(f"Duplicate entries in the dataset are {(cars.duplicated().sum())}")
```

```
Duplicate entries in the dataset are 0
```

8. Show Dataset with Categroical Data Only

In [124…
```
cars.select_dtypes(include='object').head()
```

Out[124…

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	engi
0	3	alfa-romero giulia	gas	std	two	convertible	rwd	
1	3	alfa-romero stelvio	gas	std	two	convertible	rwd	
2	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	
3	2	audi 100 ls	gas	std	four	sedan	fwd	
4	2	audi 100ls	gas	std	four	sedan	4wd	

9. Show Dataset with Continuous Data Only

In [124…
```
cars.select_dtypes(include=['float64', 'int64']).head()
```

Out[124…

	car_ID	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	s
0	1	88.6	168.8	64.1	48.8	2548	130	3.47	
1	2	88.6	168.8	64.1	48.8	2548	130	3.47	
2	3	94.5	171.2	65.5	52.4	2823	152	2.68	
3	4	99.8	176.6	66.2	54.3	2337	109	3.19	
4	5	99.4	176.6	66.4	54.3	2824	136	3.19	

Data Cleaning

This step is necessary to be done for us to make corrections on the errors in our datasets. This is for us to prepare our dataset for the data visualization/exploratory data analysis (EDA).

1. Cleaning CarName Feature

Separating the Company Name and the Car Model from the CarName Feature

In [125…
```
cars['CompanyName'] = cars['CarName'].apply(lambda x: x.split(" ")[0])
```

2. Dropping CarName column inplace of the CompanyName

In [125…
```
cars.drop(columns='CarName', inplace=True)
cars.head()
```

Out[125…

	car_ID	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	engineloc
0	1	3	gas	std	two	convertible	rwd	
1	2	3	gas	std	two	convertible	rwd	
2	3	1	gas	std	two	hatchback	rwd	
3	4	2	gas	std	four	sedan	fwd	
4	5	2	gas	std	four	sedan	4wd	

In [125…

```
cars.insert(0, 'CompanyName', cars.pop('CompanyName'))
cars.head()
```

Out[125…

	CompanyName	car_ID	symboling	fueltype	aspiration	doornumber	carbody	drive
0	alfa-romero	1	3	gas	std	two	convertible	
1	alfa-romero	2	3	gas	std	two	convertible	
2	alfa-romero	3	1	gas	std	two	hatchback	
3	audi	4	2	gas	std	four	sedan	
4	audi	5	2	gas	std	four	sedan	

3. Inspecting CopmanyName Feature

In [125…

```
cars['CompanyName'].unique()
```

Out[125…

```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche',
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

4. Fixing Mispelled Company Names

There are some names that need to be fixing and the data entries that needed fixing is as follows:

- maxda -> mazda
- Nissan -> nissan
- porcshce -> porsche
- toyouta -> toyota
- vokswagen & vw -> volkswagen

In [125…

```
cars['CompanyName'].replace({
    'maxda': 'mazda',
    'Nissan': 'nissan',
    'porcshce': 'porsche',
    'toyouta': 'toyota',
    'vokswagen': 'volkswagen',
    'vw': 'volkswagen'
}, inplace=True)
cars['CompanyName'].unique()
```

```
C:\Users\saloj\AppData\Local\Temp\ipykernel_32728\2246581534.py:1: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  cars['CompanyName'].replace({
```

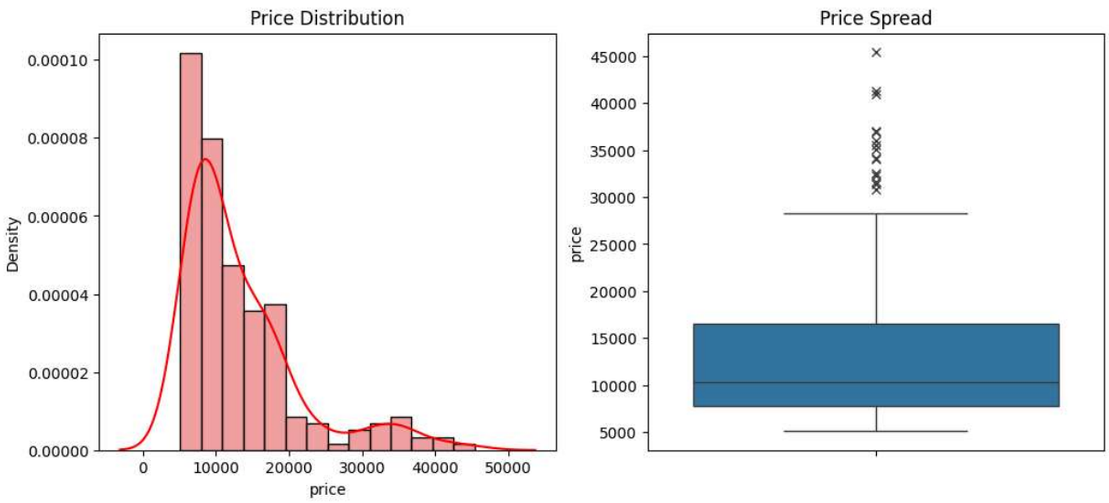
Out[125…
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'buick', 'mercury', 'mitsubishi',
       'nissan', 'peugeot', 'plymouth', 'porsche', 'renault', 'saab',
       'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

Data Visualzation / Exploratory Data Analysis (EDA)

This is a necessary step due to the data having some entries that needs to be processed. Also, we need to analyze the given dataset so we can evaluate what specific variables *correlates* to our *dependent variable* (price).

1. Visualizing the Independent Variable (Price)

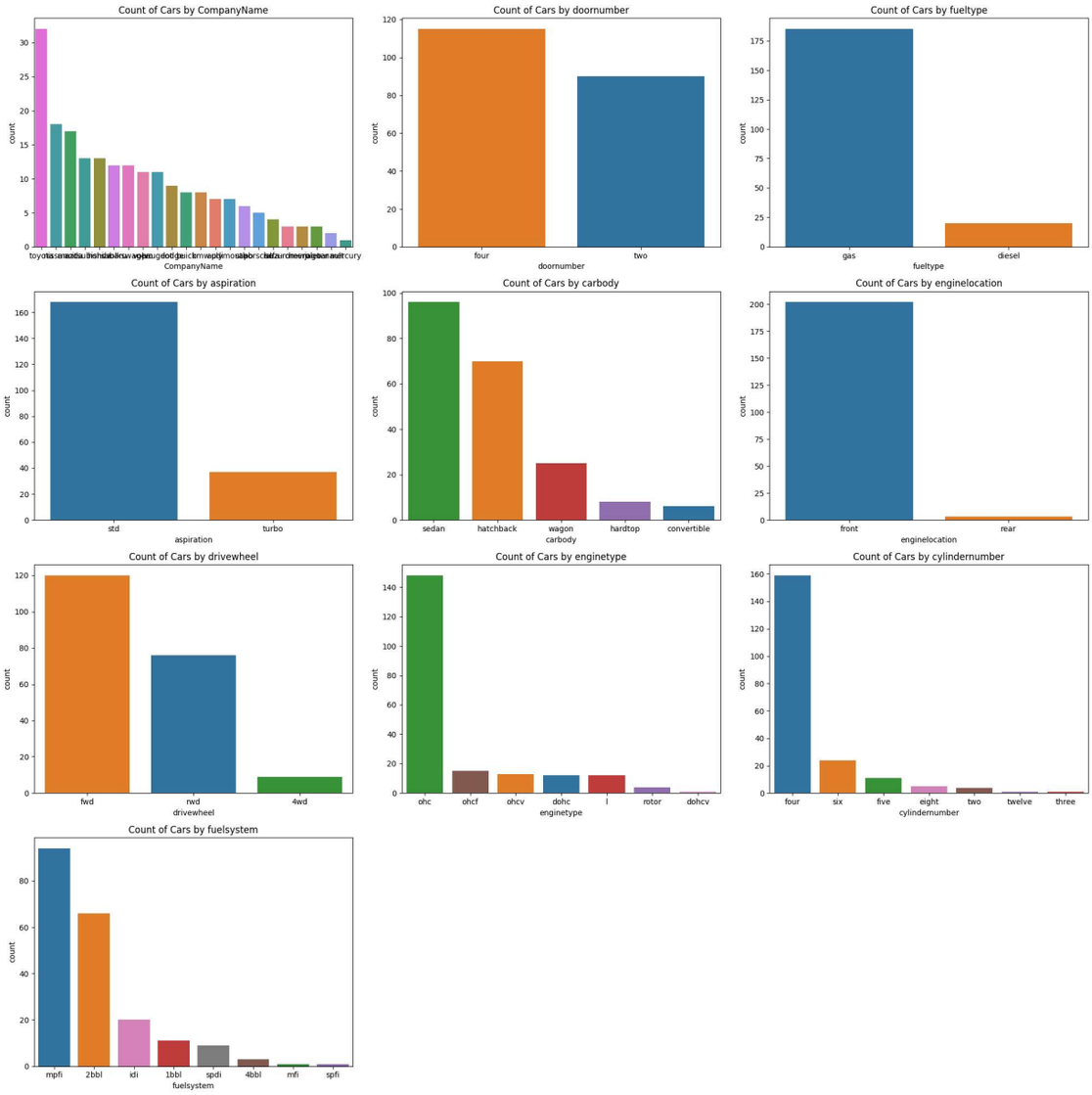
In [125…
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.histplot(cars['price'], stat="density", color="lightcoral")
sns.kdeplot(cars['price'], color="red", bw_adjust=1, label="KDE Curve")
plt.title('Price Distribution')

plt.subplot(1,2,2)
plt.title('Price Spread')
sns.boxplot(y=cars['price'], flierprops={"marker": "x"})

plt.show()
```



2. Visualizing Categorical Data

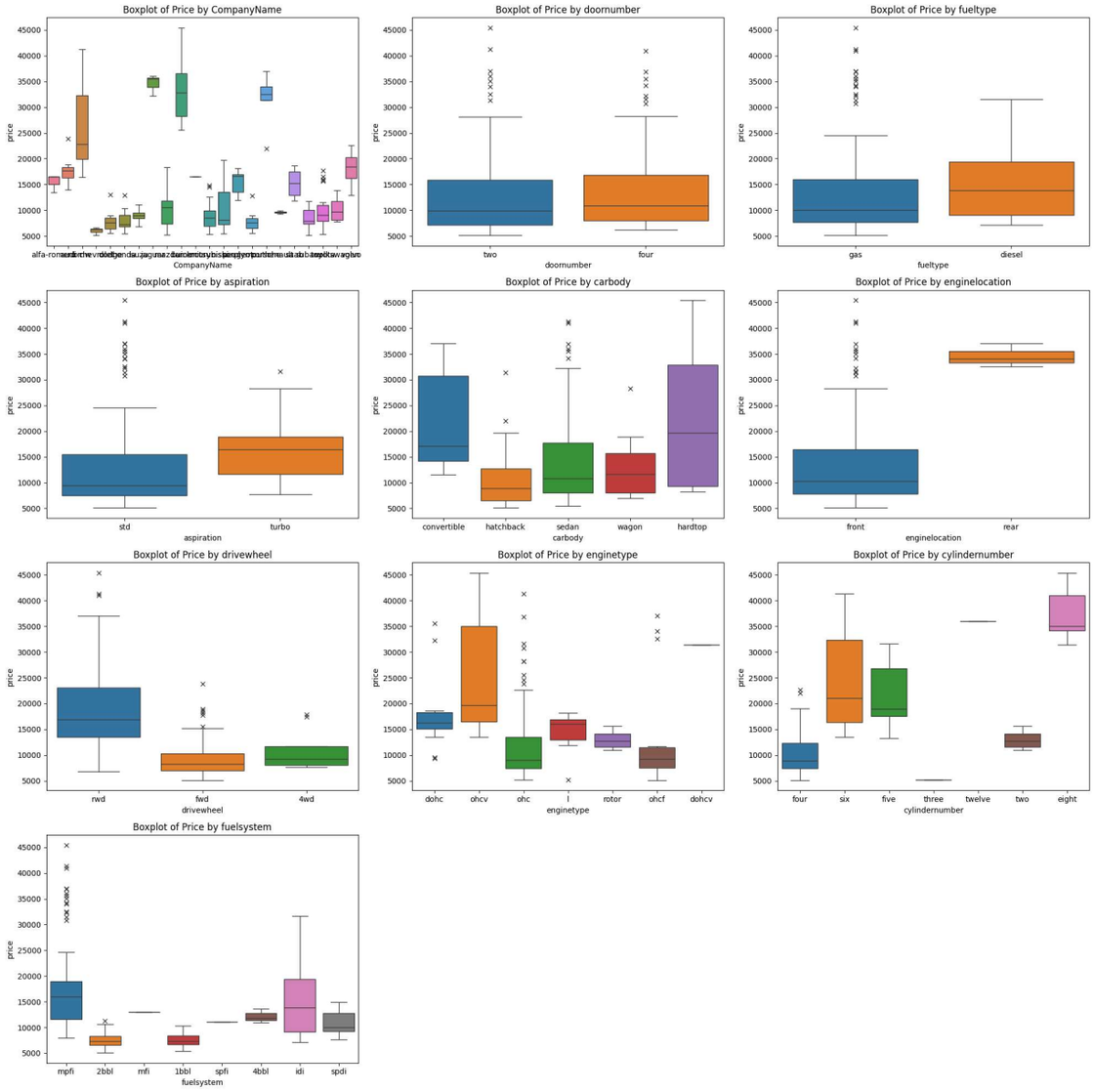
In [125…
```
categorical_features = ['CompanyName', 'doornumber', 'fueltype', 'aspiration',
                        'carbody', 'enginelocation', 'drivewheel', 'enginetype',
                        'cylindernumber', 'fuelsystem']

plt.figure(figsize=(20, 20))

# Create count plots
for i, feature in enumerate(categorical_features):
    plt.subplot(4, 3, i + 1)
    order = cars[feature].value_counts().index  # Get descending order for count pl
    sns.countplot(x=feature, hue=feature, data=cars, order=order)
    plt.title(f'Count of Cars by {feature}')
```

```
plt.tight_layout()
plt.show()
```



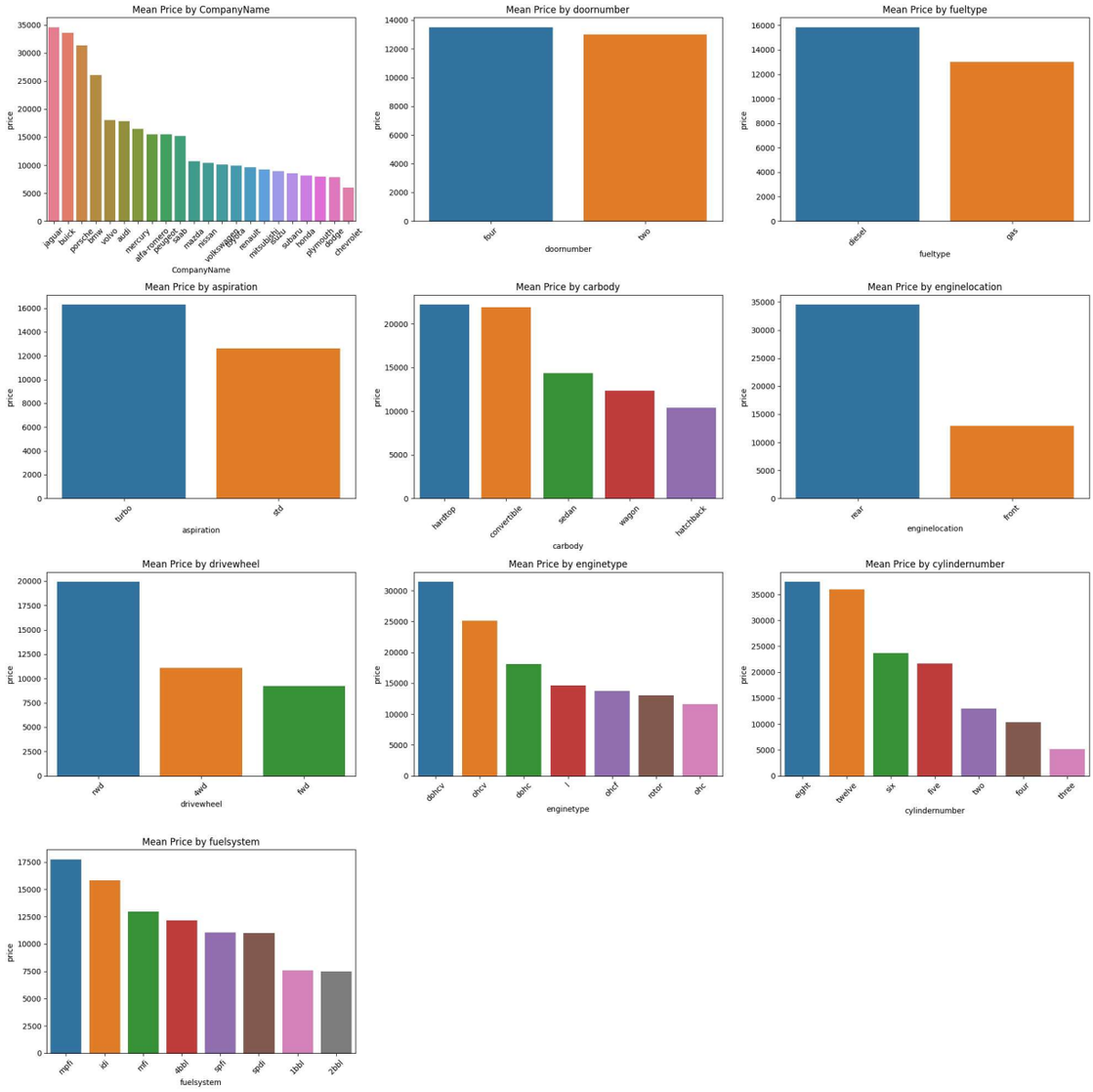
In [125…

```
# Set up the figure for box plots
plt.figure(figsize=(20, 20))

# Create box plots
for i, feature in enumerate(categorical_features):
    plt.subplot(4, 3, i + 1)
    sns.boxplot(y='price', hue=feature, x=feature, data=cars,
                legend=False, flierprops={'marker': 'x'})
    plt.title(f'Boxplot of Price by {feature}')

plt.tight_layout()
plt.show()
```



In [125…

```
plt.figure(figsize=(20, 20))

# Create bar plots for each categorical feature
for i, feature in enumerate(categorical_features):
    plt.subplot(4, 3, i + 1)

    # Calculate the mean price for each category and sort it in descending order
    mean_price = cars.groupby(feature)['price'].mean().sort_values(ascending=False)

    # Create a bar plot
    sns.barplot(x=feature, y='price', data=mean_price, estimator=lambda x: x.mean()
    plt.title(f'Mean Price by {feature}')
    plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility

plt.tight_layout()
plt.show()
```



In [125…
```
categorical_filtered = ['CompanyName', 'fueltype', 'aspiration', 'carbody', 'engine
categorical_filtered
```

Out[125…
```
['CompanyName',
 'fueltype',
 'aspiration',
 'carbody',
 'enginelocation',
 'drivewheel',
 'enginetype',
 'cylindernumber']
```

After visualization of the categorical features here are some of the insights:

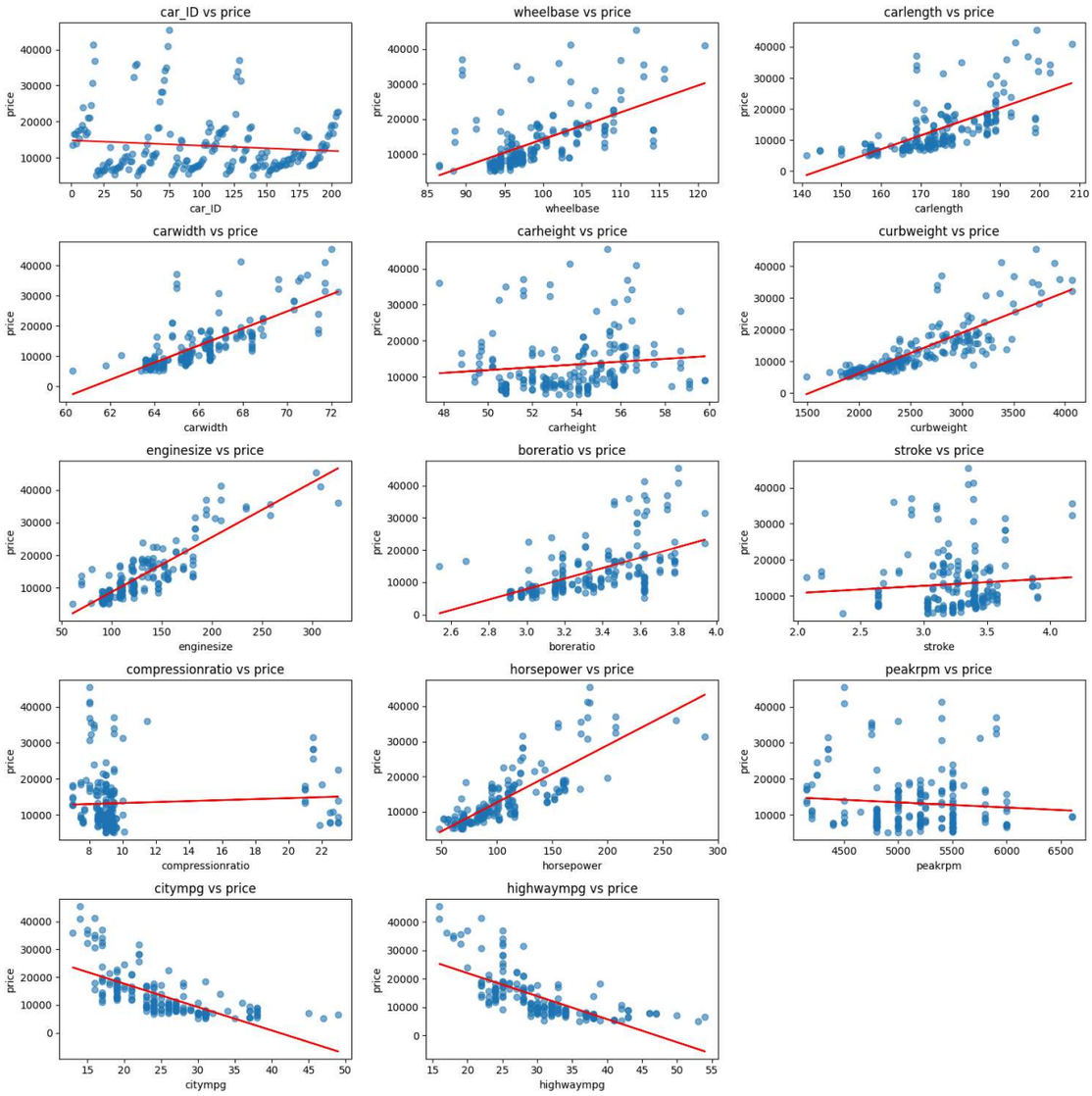
- **CompanyName**: There is a noticeable difference in mean prices across different car manufacturers, which suggests that CompanyName could be a useful feature, although it may need to be encoded appropriately.
- **doornumber**: The difference in mean price between cars with two and four doors is relatively small. This feature may have low predictive power for price.
- **fueltype**: There is a noticeable price difference between diesel and gas cars. fueltype could be useful.
- **aspiration**: There is a difference in mean price between turbo and standard aspiration, so aspiration could be a useful feature.
- **carbody**: Different body styles show distinct mean prices, indicating that carbody may contribute to price prediction.
- **enginelocation**: A significant price difference exists between front and rear engine locations. enginelocation might be an important feature.

- **drivewheel**: The mean prices vary by drive type (e.g., rwd, fwd, 4wd), so drivewheel could be a useful feature.
- **enginetype**: There are noticeable price variations across different engine types, so enginetype might also be a useful predictor.
- **cylindernumber**: The mean price varies with the number of cylinders, suggesting cylindernumber could be valuable.
- **fuelsystem**: While there is some variation in mean price across different fuel systems, it is relatively minor, indicating fuelsystem might be less significant.

3. Visualizing Continuous Data

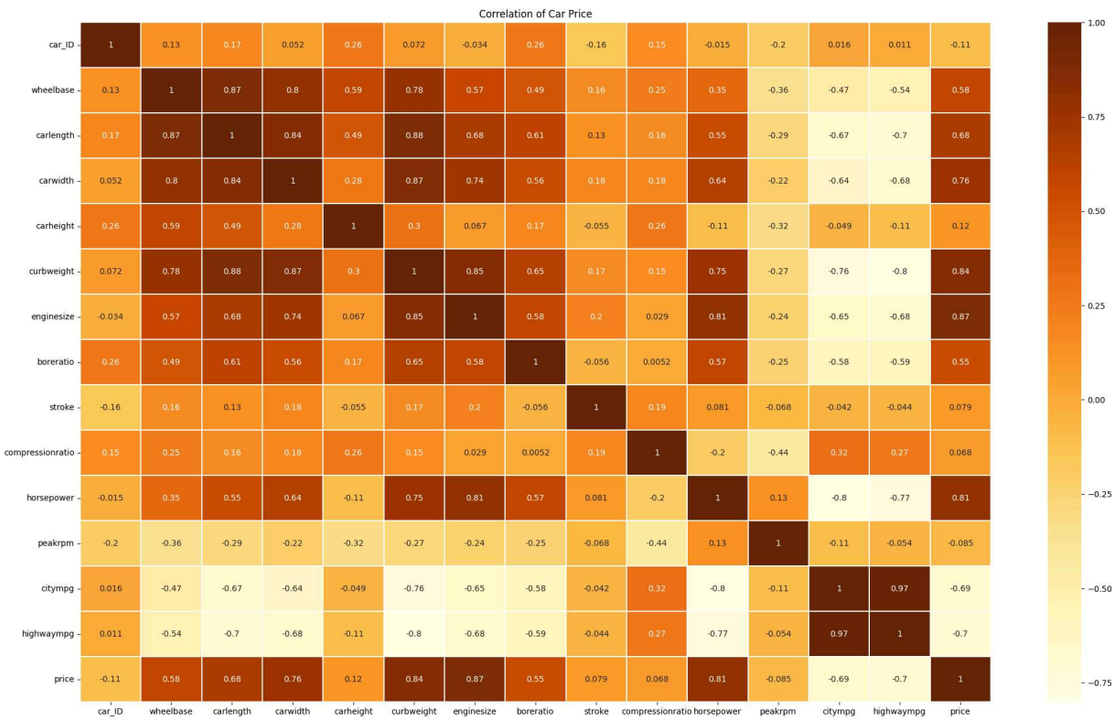
In [126…

```
# Using 'cars' as the dataset
target_variable = 'price'
features = [
    'car_ID', 'wheelbase', 'carlength', 'carwidth', 'carheight',
    'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio',
    'horsepower', 'peakrpm', 'citympg', 'highwaympg'
]

# Define the grid dimensions
n_cols = 3
n_rows = int(np.ceil(len(features) / n_cols))

# Create a grid of subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 3 * n_rows))
axes = axes.flatten()

for idx, column in enumerate(features):
    # Scatter plot
    axes[idx].scatter(cars[column], cars[target_variable], alpha=0.6)

    # Regression line
    m, b = np.polyfit(cars[column], cars[target_variable], 1)
    axes[idx].plot(cars[column], m * cars[column] + b, color='red')

    # Set titles and labels
    axes[idx].set_title(f'{column} vs {target_variable}')
    axes[idx].set_xlabel(column)
    axes[idx].set_ylabel(target_variable)

# Remove empty subplots if there are any
for i in range(len(features), len(axes)):
    fig.delaxes(axes[i])

# Layout adjustment
plt.tight_layout()
plt.show()
```



In [126…

```
data_corr=cars.select_dtypes(include=['int64','float64']).corr()
plt.figure(figsize=(25,15))
sns.heatmap(data_corr,annot=True,cmap='YlOrBr', linewidths=0.01)
plt.title('Correlation of Car Price')
plt.show()
```



In [126…

```
data_corr['price'][(data_corr['price'] > 0.5) | (data_corr['price'] < -0.5)].sort_v
```

Out[126…
```
price          1.000000
enginesize     0.874145
curbweight     0.835305
horsepower     0.808139
carwidth       0.759325
carlength      0.682920
wheelbase      0.577816
boreratio      0.553173
citympg       -0.685751
highwaympg    -0.697599
Name: price, dtype: float64
```

In [126…
```
continuous_filtered = data_corr['price'][(data_corr['price'] > 0.5) | (data_corr['p
continuous_filtered
```

Out[126…
```
Index(['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize',
       'boreratio', 'horsepower', 'citympg', 'highwaympg', 'price'],
      dtype='object')
```

After visualizing the continuous features, here are some insights:

- **curbweight** (correlation ~0.84): This has a strong positive correlation with price, indicating that heavier cars tend to be more expensive.
- **enginesize** (correlation ~0.87): This feature shows a very strong positive correlation with price, making it an essential predictor.
- **boreratio** (correlation ~0.55): This feature has a moderate positive correlation with price, suggesting that cars with a higher bore ratio
- **horsepower** (correlation ~0.81): Higher horsepower is associated with a higher price, suggesting it is an influential predictor.
- **carlength** (correlation ~0.68): This feature also has a moderate positive correlation with price, suggesting that longer cars are generally more expensive.
- **carwidth** (correlation ~0.76): This feature also has a strong positive correlation with price, suggesting that wider cars are generally more expensive.
- **wheelbase** (correlation ~0.58): Moderate positive correlation, which could still contribute useful information to the model.
- **citympg** and **highwaympg** (correlations ~-0.7): Both features have a strong negative correlation with price, indicating that cars with higher fuel efficiency tend to be less expensive.

After visualization here are the features to be retained for building the model:

Categorical Features

1. CompanyName
2. fueltype
3. aspiration
4. carbody
5. enginelocation
6. drivewheel
7. enginetype
8. cylindernumber

Continuous Features

1. curbweight
2. enginesize
3. boreratio
4. horsepower
5. carlength
6. carwidth
7. wheelbase
8. citympg

9. highwaympg

Feature Engineering

This section is for converting categorical data to numerical using One Hot Encoding and Label Encoding. Also this is where we wil drop the unnecessary features based on the insight provided from the Exploratory Data Analysis.

In [126…
```
cat_sig = ['CompanyName', 'fueltype', 'aspiration', 'carbody', 'enginelocation', 'd
con_sig = ['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize','borera
```

In [126…
```
cars[con_sig].head()
```

Out[126…

	wheelbase	carlength	carwidth	curbweight	enginesize	boreratio	horsepower	citymp
0	88.6	168.8	64.1	2548	130	3.47	111	2
1	88.6	168.8	64.1	2548	130	3.47	111	2
2	94.5	171.2	65.5	2823	152	2.68	154	1
3	99.8	176.6	66.2	2337	109	3.19	102	2
4	99.4	176.6	66.4	2824	136	3.19	115	1

1. Dropping unncessary features

In [126…
```
cars.head()
```

Out[126…

	CompanyName	car_ID	symboling	fueltype	aspiration	doornumber	carbody	drive
0	alfa-romero	1	3	gas	std	two	convertible	
1	alfa-romero	2	3	gas	std	two	convertible	
2	alfa-romero	3	1	gas	std	two	hatchback	
3	audi	4	2	gas	std	four	sedan	
4	audi	5	2	gas	std	four	sedan	

In [126…
```
cars = cars.loc[:, cat_sig + con_sig]
cars.head()
```

Out[126…

	CompanyName	fueltype	aspiration	carbody	enginelocation	drivewheel	enginetype
0	alfa-romero	gas	std	convertible	front	rwd	doh
1	alfa-romero	gas	std	convertible	front	rwd	doh
2	alfa-romero	gas	std	hatchback	front	rwd	ohcv
3	audi	gas	std	sedan	front	fwd	oh
4	audi	gas	std	sedan	front	4wd	oh

2. Categorical Feature Engineering

Converting cylindernumber from object datatype to numerical.

In [126…
```
cars['cylindernumber'] = cars['cylindernumber'].replace({'four': 4, 'six': 6, 'five
cars[cat_sig].head()
```

Out[126...

	CompanyName	fueltype	aspiration	carbody	enginelocation	drivewheel	enginetype
0	alfa-romero	gas	std	convertible	front	rwd	doh
1	alfa-romero	gas	std	convertible	front	rwd	doh
2	alfa-romero	gas	std	hatchback	front	rwd	ohcv
3	audi	gas	std	sedan	front	fwd	oh
4	audi	gas	std	sedan	front	4wd	oh

Converting other categorical features to numerical using One Hot Encoding

In [126...

```
# Exclude cylindernumber for One Hot Encoding

cat_ohe = ['CompanyName', 'fueltype', 'aspiration', 'carbody', 'enginelocation', 'd

cars_final = pd.get_dummies(cars, columns=cat_ohe, dtype='int64')

cars_final.head()
```

Out[126...

	cylindernumber	wheelbase	carlength	carwidth	curbweight	enginesize	boreratio	ho
0	4	88.6	168.8	64.1	2548	130	3.47	
1	4	88.6	168.8	64.1	2548	130	3.47	
2	6	94.5	171.2	65.5	2823	152	2.68	
3	4	99.8	176.6	66.2	2337	109	3.19	
4	5	99.4	176.6	66.4	2824	136	3.19	

Moving price to the rightmost column

In [127...

```
# Move 'price' column to the end
cols = list(cars_final.columns)
# Remove 'price' from its current position
cols.remove('price')
# Append 'price' to the end of the list
cols.append('price')

cars_final = cars_final[cols]

cars_final.head()
```

Out[127...

	cylindernumber	wheelbase	carlength	carwidth	curbweight	enginesize	boreratio	ho
0	4	88.6	168.8	64.1	2548	130	3.47	
1	4	88.6	168.8	64.1	2548	130	3.47	
2	6	94.5	171.2	65.5	2823	152	2.68	
3	4	99.8	176.6	66.2	2337	109	3.19	
4	5	99.4	176.6	66.4	2824	136	3.19	

In [127...

```
cars_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 54 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   cylindernumber           205 non-null    int64  
 1   wheelbase                205 non-null    float64
 2   carlength                205 non-null    float64
 3   carwidth                 205 non-null    float64
 4   curbweight               205 non-null    int64  
 5   enginesize               205 non-null    int64  
 6   boreratio                205 non-null    float64
 7   horsepower               205 non-null    int64  
 8   citympg                  205 non-null    int64  
 9   highwaympg               205 non-null    int64  
 10  CompanyName_alfa-romero  205 non-null    int64  
 11  CompanyName_audi         205 non-null    int64  
 12  CompanyName_bmw          205 non-null    int64  
 13  CompanyName_buick        205 non-null    int64  
 14  CompanyName_chevrolet    205 non-null    int64  
 15  CompanyName_dodge        205 non-null    int64  
 16  CompanyName_honda        205 non-null    int64  
 17  CompanyName_isuzu        205 non-null    int64  
 18  CompanyName_jaguar       205 non-null    int64  
 19  CompanyName_mazda        205 non-null    int64  
 20  CompanyName_mercury      205 non-null    int64  
 21  CompanyName_mitsubishi   205 non-null    int64  
 22  CompanyName_nissan       205 non-null    int64  
 23  CompanyName_peugeot      205 non-null    int64  
 24  CompanyName_plymouth     205 non-null    int64  
 25  CompanyName_porsche      205 non-null    int64  
 26  CompanyName_renault      205 non-null    int64  
 27  CompanyName_saab         205 non-null    int64  
 28  CompanyName_subaru       205 non-null    int64  
 29  CompanyName_toyota       205 non-null    int64  
 30  CompanyName_volkswagen   205 non-null    int64  
 31  CompanyName_volvo        205 non-null    int64  
 32  fueltype_diesel          205 non-null    int64  
 33  fueltype_gas             205 non-null    int64  
 34  aspiration_std           205 non-null    int64  
 35  aspiration_turbo         205 non-null    int64  
 36  carbody_convertible      205 non-null    int64  
 37  carbody_hardtop          205 non-null    int64  
 38  carbody_hatchback        205 non-null    int64  
 39  carbody_sedan            205 non-null    int64  
 40  carbody_wagon            205 non-null    int64  
 41  enginelocation_front     205 non-null    int64  
 42  enginelocation_rear      205 non-null    int64  
 43  drivewheel_4wd           205 non-null    int64  
 44  drivewheel_fwd           205 non-null    int64  
 45  drivewheel_rwd           205 non-null    int64  
 46  enginetype_dohc          205 non-null    int64  
 47  enginetype_dohcv         205 non-null    int64  
 48  enginetype_l             205 non-null    int64  
 49  enginetype_ohc           205 non-null    int64  
 50  enginetype_ohcf          205 non-null    int64  
 51  enginetype_ohcv          205 non-null    int64  
 52  enginetype_rotor         205 non-null    int64  
 53  price                    205 non-null    float64
dtypes: float64(5), int64(49)
memory usage: 86.6 KB
```

Building the Model

This is the section where we will build our Linear Regression Model.

1. Getting X and y variables

In [127…

```
X = cars_final.iloc[:,:-1].values
pd.DataFrame(X)
```

Out[127…

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	4.0	88.6	168.8	64.1	2548.0	130.0	3.47	111.0	21.0	27.0	1.0	0.0	0.0	0.0	0.0	0
1	4.0	88.6	168.8	64.1	2548.0	130.0	3.47	111.0	21.0	27.0	1.0	0.0	0.0	0.0	0.0	0
2	6.0	94.5	171.2	65.5	2823.0	152.0	2.68	154.0	19.0	26.0	1.0	0.0	0.0	0.0	0.0	0
3	4.0	99.8	176.6	66.2	2337.0	109.0	3.19	102.0	24.0	30.0	0.0	1.0	0.0	0.0	0.0	0
4	5.0	99.4	176.6	66.4	2824.0	136.0	3.19	115.0	18.0	22.0	0.0	1.0	0.0	0.0	0.0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	
200	4.0	109.1	188.8	68.9	2952.0	141.0	3.78	114.0	23.0	28.0	0.0	0.0	0.0	0.0	0.0	0
201	4.0	109.1	188.8	68.8	3049.0	141.0	3.78	160.0	19.0	25.0	0.0	0.0	0.0	0.0	0.0	0
202	6.0	109.1	188.8	68.9	3012.0	173.0	3.58	134.0	18.0	23.0	0.0	0.0	0.0	0.0	0.0	0
203	6.0	109.1	188.8	68.9	3217.0	145.0	3.01	106.0	26.0	27.0	0.0	0.0	0.0	0.0	0.0	0
204	4.0	109.1	188.8	68.9	3062.0	141.0	3.78	114.0	19.0	25.0	0.0	0.0	0.0	0.0	0.0	0

205 rows × 53 columns

In [127…

```
y = cars_final.iloc[:,-1].values
pd.DataFrame(y)
```

Out[127…

	0
0	13495.0
1	16500.0
2	16500.0
3	13950.0
4	17450.0
…	…
200	16845.0
201	19045.0
202	21485.0
203	22470.0
204	22625.0

205 rows × 1 columns

Train test split

In [151…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (164, 53)
X_test shape: (41, 53)
y_train shape: (164,)
y_test shape: (41,)
```

2. Training Linear Regression Model

In [151…

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

Out[151…

```
LinearRegression()
```

3. Inference

In [151…

```
data_sample = np.array([[4, 88.6, 168.8, 64.1, 2548, 130, 3.47, 111, 21, 27,
                         1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                         0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
                         1, 1, 0, 0, 0, 0, 0, 0]])
predicted_price = lr.predict(data_sample)
print(f"Predicted Price: {round(predicted_price[0],2)}")
price_row_1 = cars_final.iloc[0]['price']
print(f"Actual Price: {price_row_1}")
print(f"Difference: {price_row_1 - predicted_price}")
```

```
Predicted Price: 13647.55
Actual Price: 13495.0
Difference: [-152.55121761]
```

In [152…

```
y_pred = lr.predict(X_test)
y_pred
```

Out[152…

```
array([35007.55477017, 30557.75712889, 17657.04984859, 11416.73946059,
        6381.91592547, 13412.782285  , 33829.37496117,  6359.9832737 ,
        5992.81256663,  6464.90087023,  6843.14642654,  9153.46719542,
        6341.39279118, 18055.05890396,  7670.52285254, 21129.72525036,
       25908.02285801, 31372.81899926, 11585.88142271, 12983.28286777,
       14747.28290421, 17526.73547125,  5424.02657892,  9390.30549478,
       10282.64301764,  7875.51114401, 11898.04764469,  6254.82654555,
       16913.07711897,  6591.16960676, 19436.43853643, 19273.42487174,
        6037.19352755,  7794.37100871,  8303.71191817, 10826.73810855,
        5130.30503973,  8766.5633608 , 13647.55121761,  8979.55151067,
        5327.79855687])
```

In [152…

```
pred_df = pd.DataFrame({'Actual Value': y_test, 'Predicted Value': y_pred, 'Differe
pred_df = pred_df.round(2)

# Reset index to start at 1
pred_df.index = range(1, len(pred_df) + 1)
pred_df
```

Out[152...

	Actual Value	Predicted Value	Difference
1	32250.0	35007.55	-2757.55
2	28248.0	30557.76	-2309.76
3	15750.0	17657.05	-1907.05
4	13645.0	11416.74	2228.26
5	5499.0	6381.92	-882.92
6	15645.0	13412.78	2232.22
7	32528.0	33829.37	-1301.37
8	5399.0	6359.98	-960.98
9	7799.0	5992.81	1806.19
10	9258.0	6464.90	2793.10
11	6855.0	6843.15	11.85
12	8189.0	9153.47	-964.47
13	6918.0	6341.39	576.61
14	16558.0	18055.06	-1497.06
15	7957.0	7670.52	286.48
16	23875.0	21129.73	2745.27
17	21105.0	25908.02	-4803.02
18	30760.0	31372.82	-612.82
19	11850.0	11585.88	264.12
20	15510.0	12983.28	2526.72
21	18150.0	14747.28	3402.72
22	18950.0	17526.74	1423.26
23	8358.0	5424.03	2933.97
24	8949.0	9390.31	-441.31
25	8778.0	10282.64	-1504.64
26	7738.0	7875.51	-137.51
27	10245.0	11898.05	-1653.05
28	9538.0	6254.83	3283.17
29	14399.0	16913.08	-2514.08
30	7609.0	6591.17	1017.83
31	21485.0	19436.44	2048.56
32	17710.0	19273.42	-1563.42
33	7053.0	6037.19	1015.81
34	8499.0	7794.37	704.63
35	7295.0	8303.71	-1008.71
36	8845.0	10826.74	-1981.74
37	5572.0	5130.31	441.69
38	9960.0	8766.56	1193.44
39	16500.0	13647.55	2852.45
40	6989.0	8979.55	-1990.55

	Actual Value	Predicted Value	Difference
41	7898.0	5327.80	2570.20

4. Evaluation of Linear Regression Model

R^2 Score

In [152…
```
r2 = r2_score(y_test, y_pred)*100
print(f"R² Score: {r2:.2f}")
```

```
R² Score: 93.08
```

Mean Absolute Error (MAE)

In [152…
```
# Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

```
Mean Absolute Error (MAE): 1686.60
```

Mean Squared Error (MSE)

In [152…
```
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
```

```
Mean Squared Error (MSE): 3926875.33
```

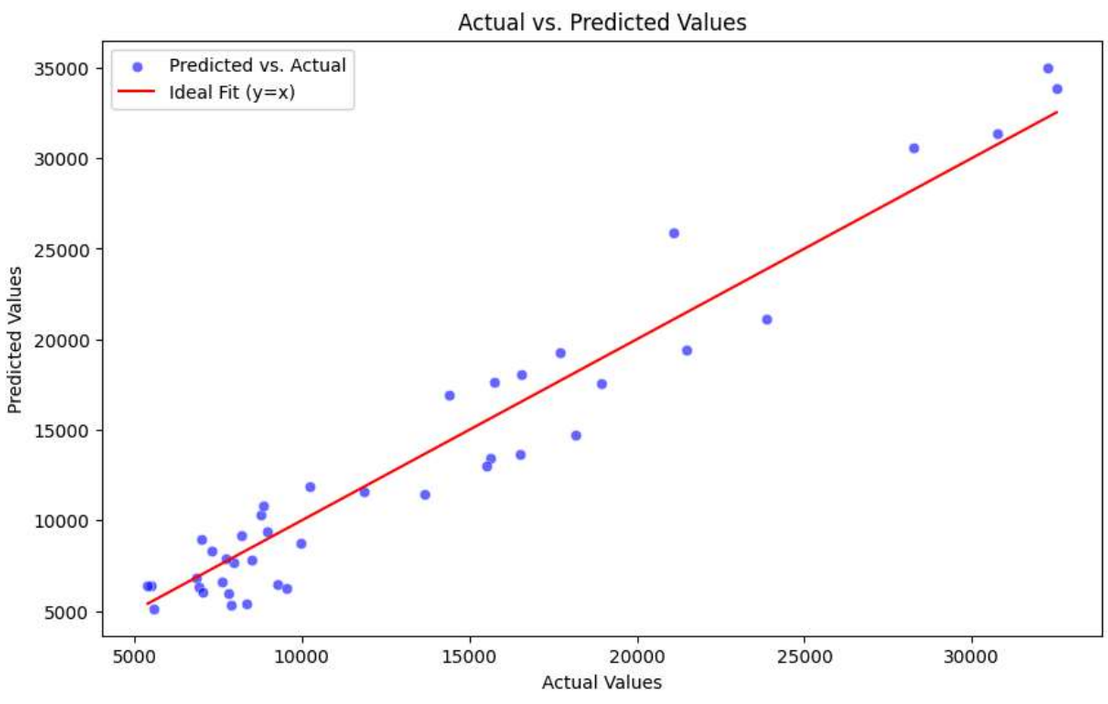
Root Mean Squared Error (RMSE)

In [152…
```
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

```
Root Mean Squared Error (RMSE): 1981.63
```

Plotting of the Predicted vs. Actual Price

In [152…
```
# Create a scatter plot for actual vs. predicted values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred, color="blue", label="Predicted vs. Actual", alp
sns.lineplot(x=y_test, y=y_test, color="red", label="Ideal Fit (y=x)")

# Set plot labels and title
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.legend()
plt.show()
```



Summary of Evaluation

In [152…

```python
# Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae:.2f}")

# Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")

# Root Mean Squared Error
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

# R² Score
r2 = r2_score(y_test, y_pred)*100
print(f"R² Score: {r2:.2f}")
```

```
Mean Absolute Error (MAE): 1686.60
Mean Squared Error (MSE): 3926875.33
Root Mean Squared Error (RMSE): 1981.63
R² Score: 93.08
```