Linear Regression Analysis: Automobile Datasets

Marquez, Keith Leigh Zhen R.

Data Wrangling:

```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.6)
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
automobile = fetch_ucirepo(id=10)

# data (as pandas dataframes)
X = automobile.data.features
y = automobile.data.targets

# metadata
print(automobile.metadata)

# variable information
print(automobile.variables)
```

```
{'uci_id': 10, 'name': 'Automobile', 'repository_url': 'https://archive.ics.uci.edu/dataset/10/automobile', 'data_url': 'https://arch
                 name     role         type demographic  \
0               price  Feature   Continuous        None
1         highway-mpg  Feature   Continuous        None
2            city-mpg  Feature   Continuous        None
3            peak-rpm  Feature   Continuous        None
4          horsepower  Feature   Continuous        None
5   compression-ratio  Feature   Continuous        None
6              stroke  Feature   Continuous        None
7                bore  Feature   Continuous        None
8         fuel-system  Feature  Categorical        None
9         engine-size  Feature   Continuous        None
10   num-of-cylinders  Feature      Integer        None
11        engine-type  Feature  Categorical        None
12        curb-weight  Feature   Continuous        None
13             height  Feature   Continuous        None
14              width  Feature   Continuous        None
15             length  Feature   Continuous        None
16         wheel-base  Feature   Continuous        None
17    engine-location  Feature       Binary        None
18       drive-wheels  Feature  Categorical        None
19         body-style  Feature  Categorical        None
20        num-of-doors  Feature      Integer        None
21         aspiration  Feature       Binary        None
22          fuel-type  Feature       Binary        None
23               make  Feature  Categorical        None
24  normalized-losses  Feature   Continuous        None
25          symboling   Target      Integer        None

                                  description units missing_values
0                  continuous from 5118 to 45400  None            yes
1                       continuous from 16 to 54  None             no
2                       continuous from 13 to 49  None             no
3                   continuous from 4150 to 6600  None            yes
4                      continuous from 48 to 288  None            yes
5                        continuous from 7 to 23  None             no
6                   continuous from 2.07 to 4.17  None            yes
7                   continuous from 2.54 to 3.94  None            yes
8       1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi  None             no
9                      continuous from 61 to 326  None             no
10        eight, five, four, six, three, twelve, two  None             no
11            dohc, dohcv, l, ohc, ohcf, ohcv, rotor  None             no
12                  continuous from 1488 to 4066  None             no
13                  continuous from 47.8 to 59.8  None             no
14                  continuous from 60.3 to 72.3  None             no
15                continuous from 141.1 to 208.1  None             no
16                     continuous from 86.6 120.9  None             no
17                                   front, rear  None             no
18                                 4wd, fwd, rwd  None             no
```

```
19      hardtop, wagon, sedan, hatchback, convertible  None            no
20                                          four, two  None           yes
21                                         std, turbo  None            no
22                                        diesel, gas  None            no
23  alfa-romero, audi, bmw, chevrolet, dodge, hond...  None            no
24                           continuous from 65 to 256  None           yes
25                               -3, -2, -1, 0, 1, 2, 3  None            no
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
auto = pd.concat([X,y], axis=1)
auto
```

	price	highway-mpg	city-mpg	peak-rpm	horsepower	compression-ratio	stroke	bore	fuel-system	engine-size	...	wheel-base	engine-location	drive-wheels	body-style
0	13495.0	27	21	5000.0	111.0	9.0	2.68	3.47	mpfi	130	...	88.6	front	rwd	convertible
1	16500.0	27	21	5000.0	111.0	9.0	2.68	3.47	mpfi	130	...	88.6	front	rwd	convertible
2	16500.0	26	19	5000.0	154.0	9.0	3.47	2.68	mpfi	152	...	94.5	front	rwd	hatchback
3	13950.0	30	24	5500.0	102.0	10.0	3.40	3.19	mpfi	109	...	99.8	front	fwd	sedan
4	17450.0	22	18	5500.0	115.0	8.0	3.40	3.19	mpfi	136	...	99.4	front	4wd	sedan
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
200	16845.0	28	23	5400.0	114.0	9.5	3.15	3.78	mpfi	141	...	109.1	front	rwd	sedan
201	19045.0	25	19	5300.0	160.0	8.7	3.15	3.78	mpfi	141	...	109.1	front	rwd	sedan
202	21485.0	23	18	5500.0	134.0	8.8	2.87	3.58	mpfi	173	...	109.1	front	rwd	sedan
203	22470.0	27	26	4800.0	106.0	23.0	3.40	3.01	idi	145	...	109.1	front	rwd	sedan
204	22625.0	25	19	5400.0	114.0	9.5	3.15	3.78	mpfi	141	...	109.1	front	rwd	sedan

205 rows × 26 columns

```
auto.dtypes
```

```
price                float64
highway-mpg            int64
city-mpg               int64
peak-rpm             float64
horsepower           float64
compression-ratio    float64
stroke               float64
bore                 float64
fuel-system           object
engine-size            int64
num-of-cylinders       int64
engine-type           object
curb-weight            int64
height               float64
width                float64
length               float64
wheel-base           float64
engine-location       object
drive-wheels          object
body-style            object
num-of-doors         float64
aspiration            object
fuel-type             object
make                  object
normalized-losses    float64
symboling              int64
dtype: object
```

Identify missing value

```
auto_null=auto.isnull().sum()
auto_null
```

```
price                 4
highway-mpg           0
city-mpg              0
peak-rpm              2
horsepower            2
compression-ratio     0
stroke                4
bore                  4
fuel-system           0
engine-size           0
num-of-cylinders      0
engine-type           0
curb-weight           0
height                0
width                 0
length                0
wheel-base            0
engine-location       0
drive-wheels          0
body-style            0
num-of-doors          2
aspiration            0
fuel-type             0
make                  0
normalized-losses    41
symboling             0
dtype: int64
```

Checking for duplicates

```
# Check for duplicates
duplicate_rows = auto.duplicated()

# Count the number of duplicate rows
num_duplicates = duplicate_rows.sum()
print("Number of duplicate rows:", num_duplicates)
```

```
Number of duplicate rows: 0
```

Summary of the data

```
auto.describe()
```

	price	highway-mpg	city-mpg	peak-rpm	horsepower	compression-ratio	stroke	bore	engine-size	num-of-cylinders	cu wei
count	201.000000	205.000000	205.000000	203.000000	203.000000	205.000000	201.000000	201.000000	205.000000	205.000000	205.000
mean	13207.129353	30.751220	25.219512	5125.369458	104.256158	10.142537	3.255423	3.329751	126.907317	4.380488	2555.565
std	7947.066342	6.886443	6.542142	479.334560	39.714369	3.972040	0.316717	0.273539	41.642693	1.080854	520.680
min	5118.000000	16.000000	13.000000	4150.000000	48.000000	7.000000	2.070000	2.540000	61.000000	2.000000	1488.000
25%	7775.000000	25.000000	19.000000	4800.000000	70.000000	8.600000	3.110000	3.150000	97.000000	4.000000	2145.000
50%	10295.000000	30.000000	24.000000	5200.000000	95.000000	9.000000	3.290000	3.310000	120.000000	4.000000	2414.000
75%	16500.000000	34.000000	30.000000	5500.000000	116.000000	9.400000	3.410000	3.590000	141.000000	4.000000	2935.000
max	45400.000000	54.000000	49.000000	6600.000000	288.000000	23.000000	4.170000	3.940000	326.000000	12.000000	4066.000

Fill in mssing data

```
auto['price'].fillna(auto['price'].median(), inplace=True)
auto['peak-rpm'].fillna(auto['peak-rpm'].median(), inplace=True)
auto['horsepower'].fillna(auto['horsepower'].median(), inplace=True)
auto['stroke'].fillna(auto['stroke'].median(), inplace=True)
auto['bore'].fillna(auto['bore'].median(), inplace=True)
auto['num-of-doors'].fillna(auto['num-of-doors'].mode()[0], inplace=True)
auto['normalized-losses'].fillna(auto['normalized-losses'].median(), inplace=True)
auto
```

	price	highway-mpg	city-mpg	peak-rpm	horsepower	compression-ratio	stroke	bore	fuel-system	engine-size	...	wheel-base	engine-location	drive-wheels	body-style
0	13495.0	27	21	5000.0	111.0	9.0	2.68	3.47	mpfi	130	...	88.6	front	rwd	convertible
1	16500.0	27	21	5000.0	111.0	9.0	2.68	3.47	mpfi	130	...	88.6	front	rwd	convertible
2	16500.0	26	19	5000.0	154.0	9.0	3.47	2.68	mpfi	152	...	94.5	front	rwd	hatchback
3	13950.0	30	24	5500.0	102.0	10.0	3.40	3.19	mpfi	109	...	99.8	front	fwd	sedan
4	17450.0	22	18	5500.0	115.0	8.0	3.40	3.19	mpfi	136	...	99.4	front	4wd	sedan
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
200	16845.0	28	23	5400.0	114.0	9.5	3.15	3.78	mpfi	141	...	109.1	front	rwd	sedan
201	19045.0	25	19	5300.0	160.0	8.7	3.15	3.78	mpfi	141	...	109.1	front	rwd	sedan
202	21485.0	23	18	5500.0	134.0	8.8	2.87	3.58	mpfi	173	...	109.1	front	rwd	sedan
203	22470.0	27	26	4800.0	106.0	23.0	3.40	3.01	idi	145	...	109.1	front	rwd	sedan
204	22625.0	25	19	5400.0	114.0	9.5	3.15	3.78	mpfi	141	...	109.1	front	rwd	sedan

205 rows × 26 columns

Checking for null

```
auto.isnull().sum()
```

```
price                0
highway-mpg          0
city-mpg             0
peak-rpm             0
horsepower           0
compression-ratio    0
stroke               0
bore                 0
fuel-system          0
engine-size          0
num-of-cylinders     0
engine-type          0
curb-weight          0
height               0
width                0
length               0
wheel-base           0
engine-location      0
drive-wheels         0
body-style           0
num-of-doors         0
aspiration           0
fuel-type            0
make                 0
normalized-losses    0
symboling            0
dtype: int64
```

Convert/change data types

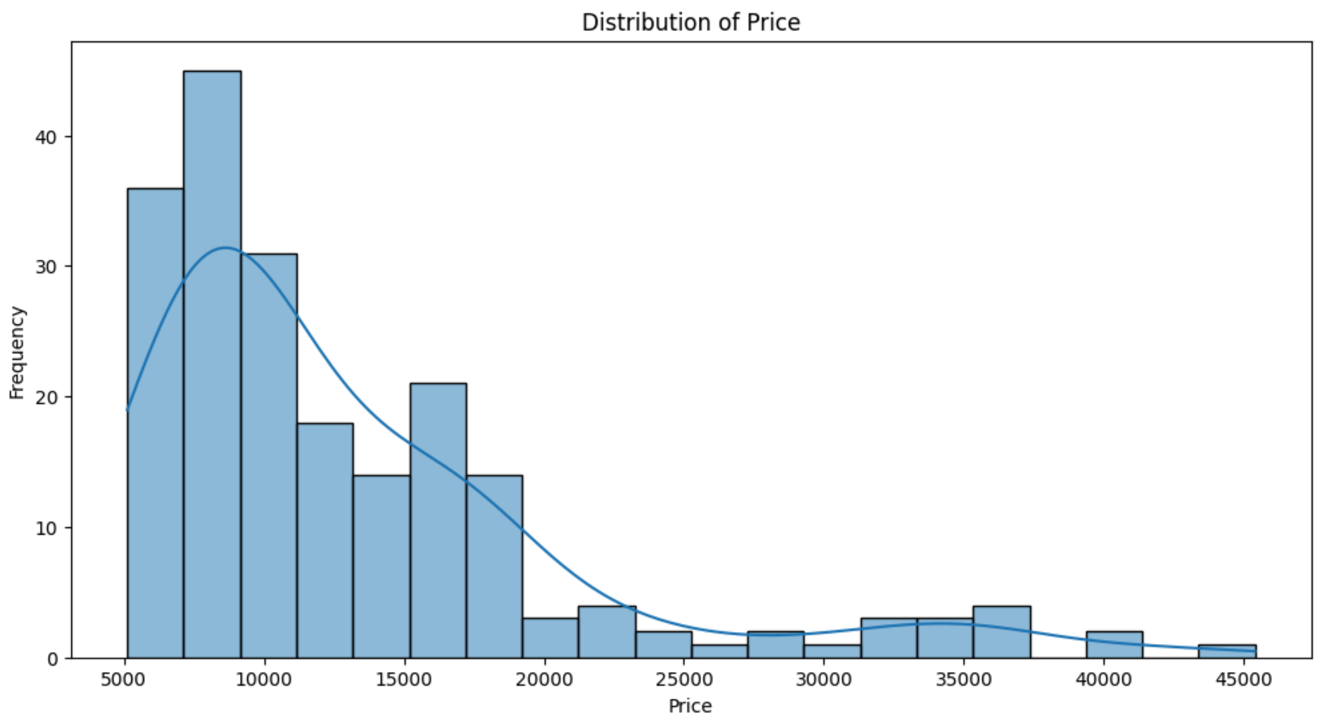
```
auto['num-of-doors'] = auto['num-of-doors'].astype(int)
auto.dtypes
```

```
price                float64
highway-mpg            int64
city-mpg               int64
peak-rpm             float64
horsepower           float64
compression-ratio    float64
stroke               float64
bore                 float64
fuel-system           object
engine-size            int64
num-of-cylinders       int64
engine-type           object
```

```
curb-weight            int64
height               float64
width                float64
length               float64
wheel-base           float64
engine-location       object
drive-wheels          object
body-style            object
num-of-doors           int64
aspiration            object
fuel-type             object
make                  object
normalized-losses    float64
symboling              int64
dtype: object
```

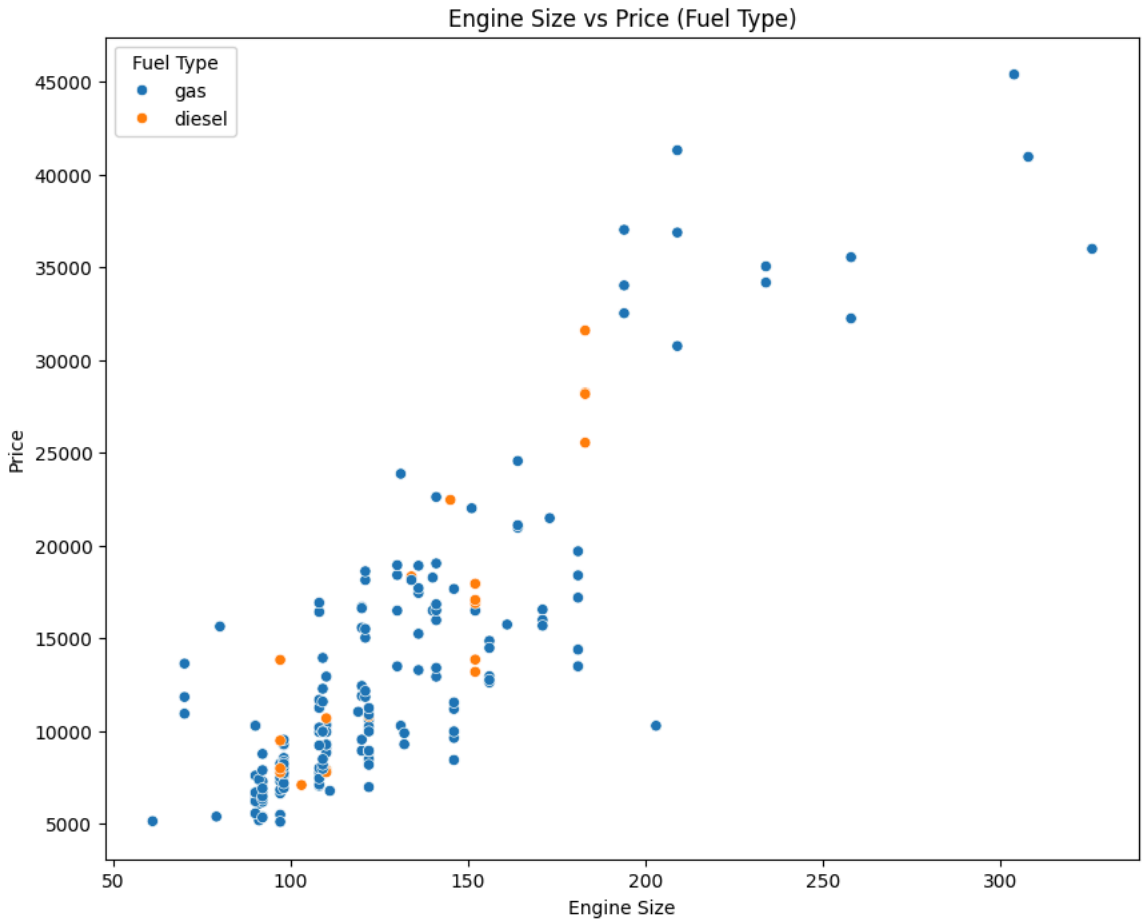
EDA (exploratory data analysis):

```
plt.figure(figsize=(12, 6))
sns.histplot(auto['price'], bins=20, kde=True)
plt.title('Distribution of Price')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



The graph shows a Distribution of Price with an overlaid line graph. It reveals that the majority of data points cluster around the 10,000 mark, indicating it's the most common price. Beyond this point, there's a decrease in frequency, suggesting higher-priced items are less common. The downward trend of the line graph reinforces this, indicating fewer items as prices rise. This implies a market where lower-priced items dominate and consumers may be sensitive to price increases.

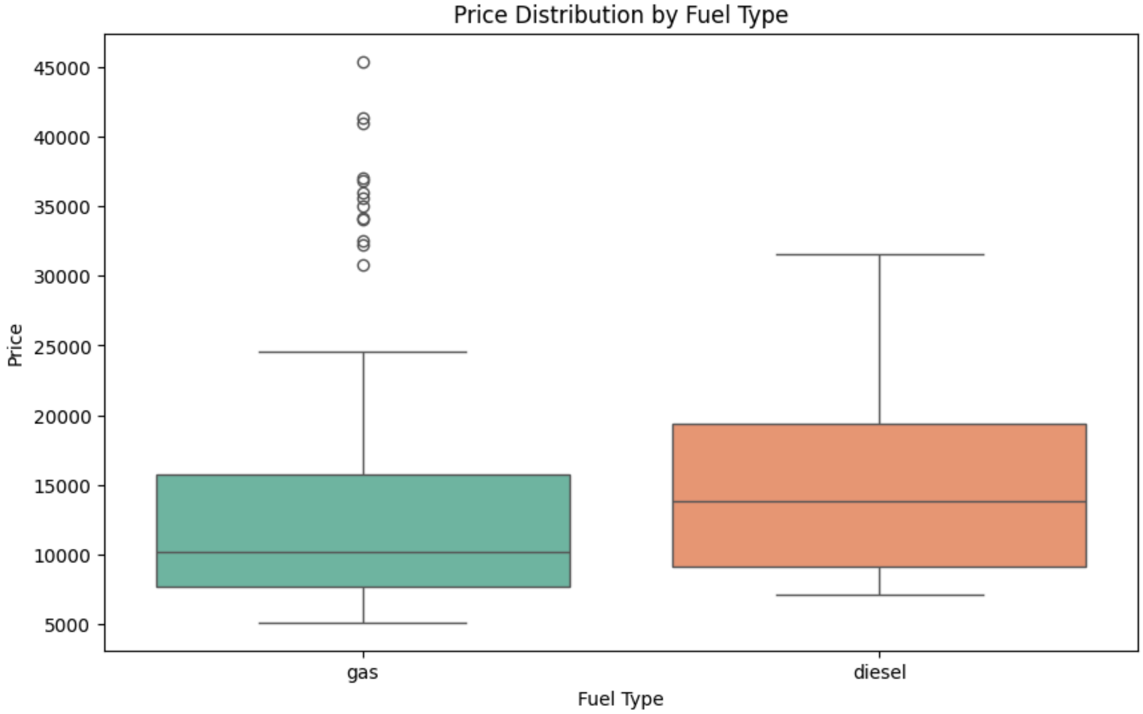
```
plt.figure(figsize=(10, 8))
sns.scatterplot(data=auto, x='engine-size', y='price', hue='fuel-type')
plt.title('Engine Size vs Price (Fuel Type)')
plt.xlabel('Engine Size')
plt.ylabel('Price')
plt.legend(title='Fuel Type')
plt.show()
```



The graph displays a scatter plot comparing engine size to price, with distinctions made for gas and diesel fuel types. Gas vehicles (depicted in blue) outnumber diesel vehicles (depicted in orange) and generally offer a wider range of engine sizes and prices, making them more affordable, especially for smaller engines. Diesel vehicles tend to have mid-range engine sizes and higher prices. This suggests that consumers seeking budget-friendly options may prefer gas vehicles with smaller engines, while diesel vehicles, despite their higher cost, may offer benefits justifying the expense, particularly for mid-range engine sizes. Overall, the trend indicates that larger engine sizes typically correlate with higher prices for both fuel types, though there are exceptions, especially among gas vehicles. This implies that factors beyond engine size, such as brand, features, or market demand, may also influence pricing.

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='fuel-type', y='price', data=auto, palette='Set2')
plt.title('Price Distribution by Fuel Type')
plt.xlabel('Fuel Type')
plt.ylabel('Price')
plt.show()
```

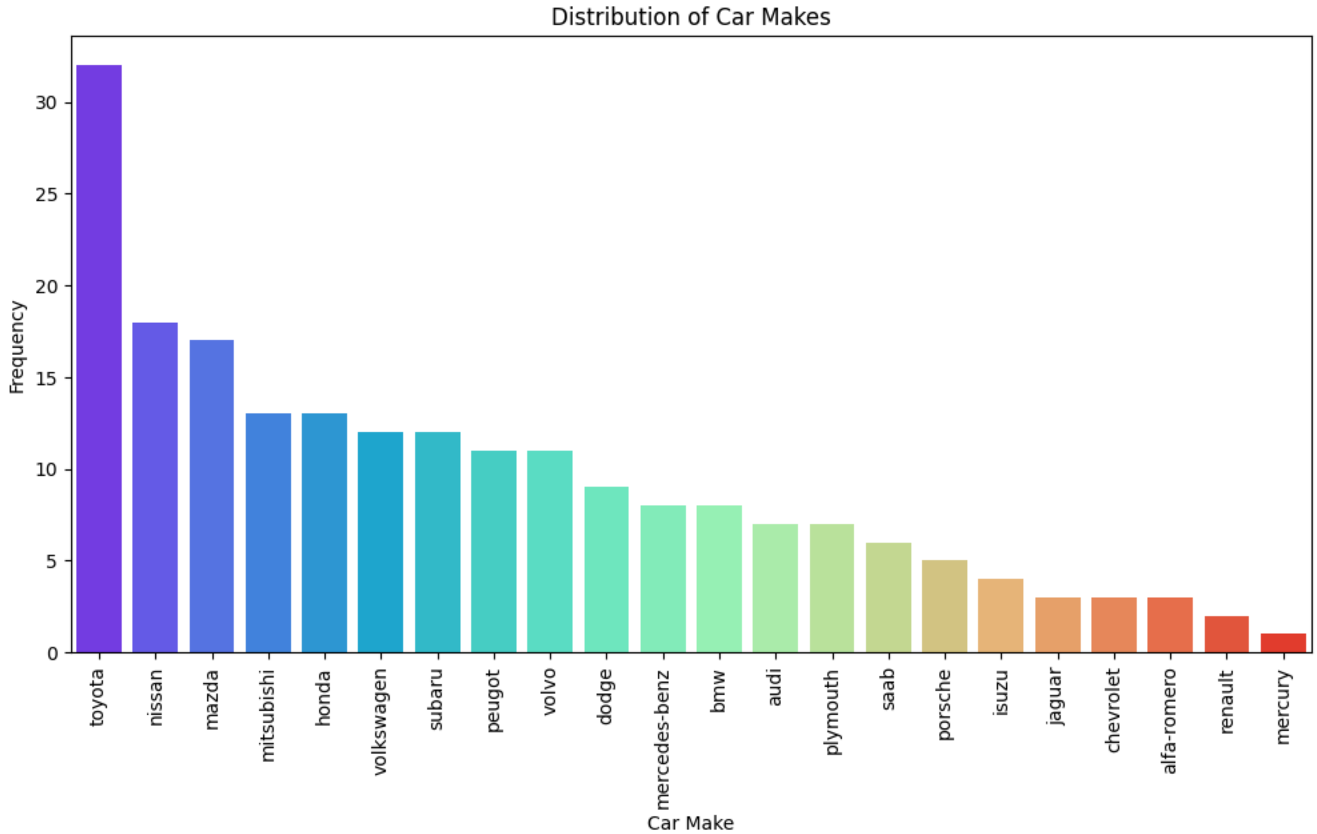
```
<ipython-input-20-5742f86d257d>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(x='fuel-type', y='price', data=auto, palette='Set2')
```



The graph compares vehicle prices by fuel type, showing gas vehicles have a median price around 10,000, with an interquartile range (IQR) of 7,500 to 12,500 and outliers reaching 45,000. Diesel vehicles have a higher median price of approximately 17,500, with a broader IQR (15,000 to 20,000) indicating greater price variance. Overall, diesel cars are generally pricier upfront, while gas cars exhibit more variability, occasionally surpassing diesel prices due to specific models or features. Gas vehicles include many high-priced outliers, while diesel prices are more consistently higher.

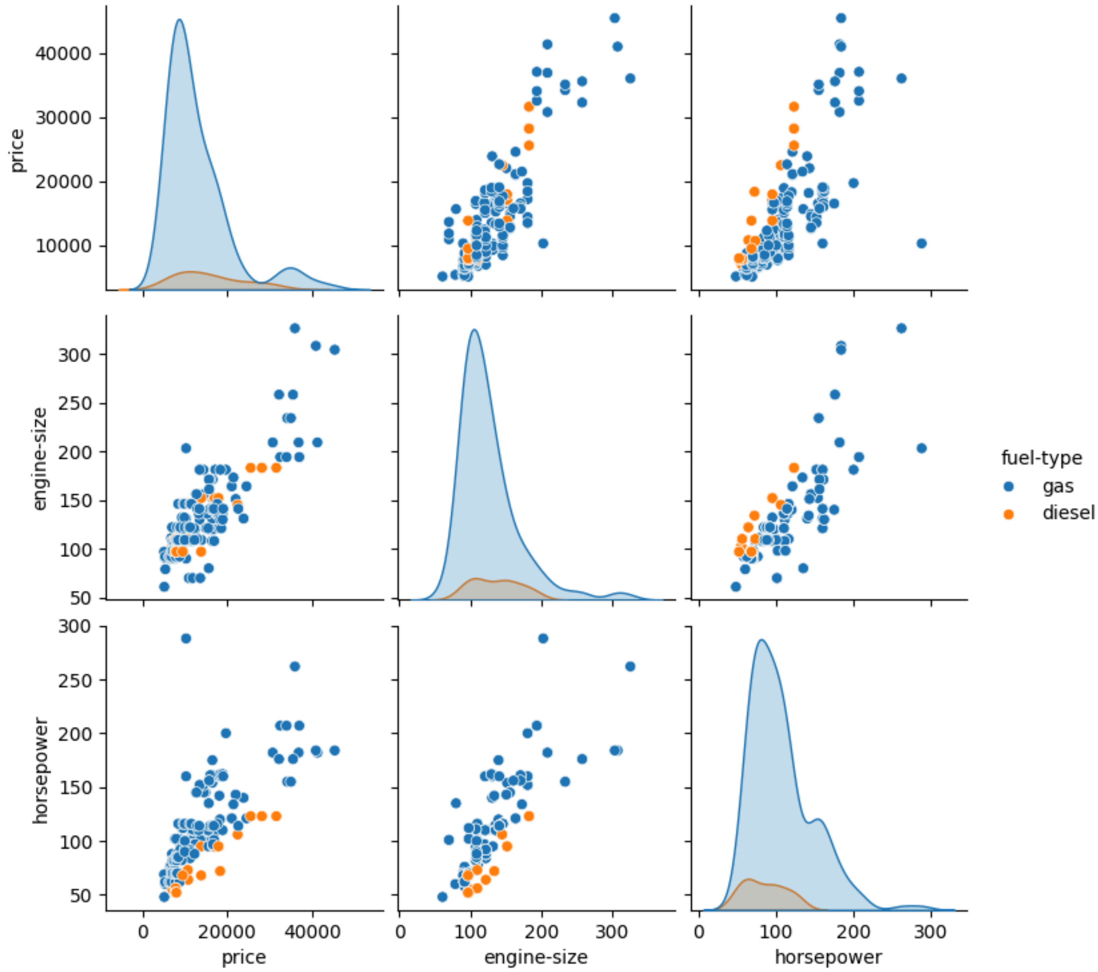
```
plt.figure(figsize=(12, 6))
sns.countplot(x='make', data=auto, order=auto['make'].value_counts().index, palette='rainbow')
plt.xticks(rotation=90)
plt.title('Distribution of Car Makes')
plt.xlabel('Car Make')
plt.ylabel('Frequency')
plt.show()
```

```
<ipython-input-19-6b7cb4b082c2>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.countplot(x='make', data=auto, order=auto['make'].value_counts().index, palette='rainbow')
```



The graph illustrates the distribution of car makes, revealing Toyota's dominance with over 30% frequency, followed by Nissan and Mazda. Luxury brands like Mercedes-Benz, BMW, and Audi hold moderate frequencies, while less common makes such as Jaguar, Alfa Romeo, Renault, and Mercury have limited popularity. This indicates Toyota's significant lead in the market, with other brands competing in a diverse but competitive landscape.

```
sns.pairplot(auto[['price', 'engine-size', 'horsepower', 'fuel-type']], hue='fuel-type')
plt.show()
```



This shows that cars with larger engines and higher horsepower tend to command higher prices, particularly evident in diesel cars. While there's a clear positive correlation between horsepower and price, the link between engine size and horsepower is less distinct. Overall, the visual representations highlight the trend of higher specifications correlating with higher prices, emphasizing the importance of considering fuel type when analyzing these relationships.

Linear Regression Analysis

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

Selecting predictor variables and target variable

```
predictors = ['engine-size', 'horsepower', 'curb-weight', 'highway-mpg']
target = 'price'
```

Splitting the dataset into training and testing sets

```
X = auto[predictors]
y = auto[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Train the linear regression model

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

```
▾ LinearRegression
LinearRegression()
```

Make predictions on the test set

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

Evaluate the model

```
mse = metrics.mean_squared_error(y_test, y_pred)
```