Importing Libraries

```
import pandas as pd
import matplotlib.pylab as plt
%matplotlib inline
import numpy as np
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from scipy.stats import boxcox
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder , PowerTransformer , StandardScaler, MinM
from sklearn.compose import ColumnTransformer
```

Loading Dataset

```
df = pd.read_csv('/content/CarPrice_Assignment.csv')
```

EDA

```
df.sample(5)
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drive
138	139	2	subaru	gas	std	two	hatchback	
19	20	1	chevrolet monte carlo	gas	std	two	hatchback	
3	4	2	audi 100 ls	gas	std	four	sedan	
183	184	2	volkswagen 1131 deluxe sedan	gas	std	two	sedan	
108	109	0	peugeot 304	diesel	turbo	four	sedan	

5 rows × 26 columns

```
df.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
```

```
df.describe()
```

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweig
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.0000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.5658
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.6802
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.0000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.0000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.0000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.0000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.0000

```
df.shape
```

```
(205, 26)
```

Data Cleaning and Wrangling

```
df.isnull().sum()
```

```
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
```

```
sum(df.duplicated(subset = 'car_ID')) == 0
```

```
True
```

Car Name

```
df["CarName"].unique()
```

```
array(['alfa-romero giulia', 'alfa-romero stelvio',
       'alfa-romero Quadrifoglio', 'audi 100 ls', 'audi 100ls',
       'audi fox', 'audi 5000', 'audi 4000', 'audi 5000s (diesel)',
       'bmw 320i', 'bmw x1', 'bmw x3', 'bmw z4', 'bmw x4', 'bmw x5',
       'chevrolet impala', 'chevrolet monte carlo', 'chevrolet vega 2300',
       'dodge rampage', 'dodge challenger se', 'dodge d200',
       'dodge monaco (sw)', 'dodge colt hardtop', 'dodge colt (sw)',
       'dodge coronet custom', 'dodge dart custom',
       'dodge coronet custom (sw)', 'honda civic', 'honda civic cvcc',
       'honda accord cvcc', 'honda accord lx', 'honda civic 1500 gl',
       'honda accord', 'honda civic 1300', 'honda prelude',
       'honda civic (auto)', 'isuzu MU-X', 'isuzu D-Max ',
       'isuzu D-Max V-Cross', 'jaguar xj', 'jaguar xf', 'jaguar xk',
       'maxda rx3', 'maxda glc deluxe', 'mazda rx2 coupe', 'mazda rx-4',
```

```
       'mazda glc deluxe', 'mazda 626', 'mazda glc', 'mazda rx-7 gs',
       'mazda glc 4', 'mazda glc custom l', 'mazda glc custom',
       'buick electra 225 custom', 'buick century luxus (sw)',
       'buick century', 'buick skyhawk', 'buick opel isuzu deluxe',
       'buick skylark', 'buick century special',
       'buick regal sport coupe (turbo)', 'mercury cougar',
       'mitsubishi mirage', 'mitsubishi lancer', 'mitsubishi outlander',
       'mitsubishi g4', 'mitsubishi mirage g4', 'mitsubishi montero',
       'mitsubishi pajero', 'Nissan versa', 'nissan gt-r', 'nissan rogue',
       'nissan latio', 'nissan titan', 'nissan leaf', 'nissan juke',
       'nissan note', 'nissan clipper', 'nissan nv200', 'nissan dayz',
       'nissan fuga', 'nissan otti', 'nissan teana', 'nissan kicks',
       'peugeot 504', 'peugeot 304', 'peugeot 504 (sw)', 'peugeot 604sl',
       'peugeot 505s turbo diesel', 'plymouth fury iii',
       'plymouth cricket', 'plymouth satellite custom (sw)',
       'plymouth fury gran sedan', 'plymouth valiant', 'plymouth duster',
       'porsche macan', 'porcshce panamera', 'porsche cayenne',
       'porsche boxter', 'renault 12tl', 'renault 5 gtl', 'saab 99e',
       'saab 99le', 'saab 99gle', 'subaru', 'subaru dl', 'subaru brz',
       'subaru baja', 'subaru r1', 'subaru r2', 'subaru trezia',
       'subaru tribeca', 'toyota corona mark ii', 'toyota corona',
       'toyota corolla 1200', 'toyota corona hardtop',
       'toyota corolla 1600 (sw)', 'toyota carina', 'toyota mark ii',
       'toyota corolla', 'toyota corolla liftback',
       'toyota celica gt liftback', 'toyota corolla tercel',
       'toyota corona liftback', 'toyota starlet', 'toyota tercel',
       'toyota cressida', 'toyota celica gt', 'toyouta tercel',
       'vokswagen rabbit', 'volkswagen 1131 deluxe sedan',
       'volkswagen model 111', 'volkswagen type 3', 'volkswagen 411 (sw)',
       'volkswagen super beetle', 'volkswagen dasher', 'vw dasher',
       'vw rabbit', 'volkswagen rabbit', 'volkswagen rabbit custom',
       'volvo 145e (sw)', 'volvo 144ea', 'volvo 244dl', 'volvo 245',
       'volvo 264gl', 'volvo diesel', 'volvo 246'], dtype=object)
```

```
df['brand'] = df.CarName.str.split(' ').str.get(0).str.lower()
```

```
df.brand.unique()
```

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

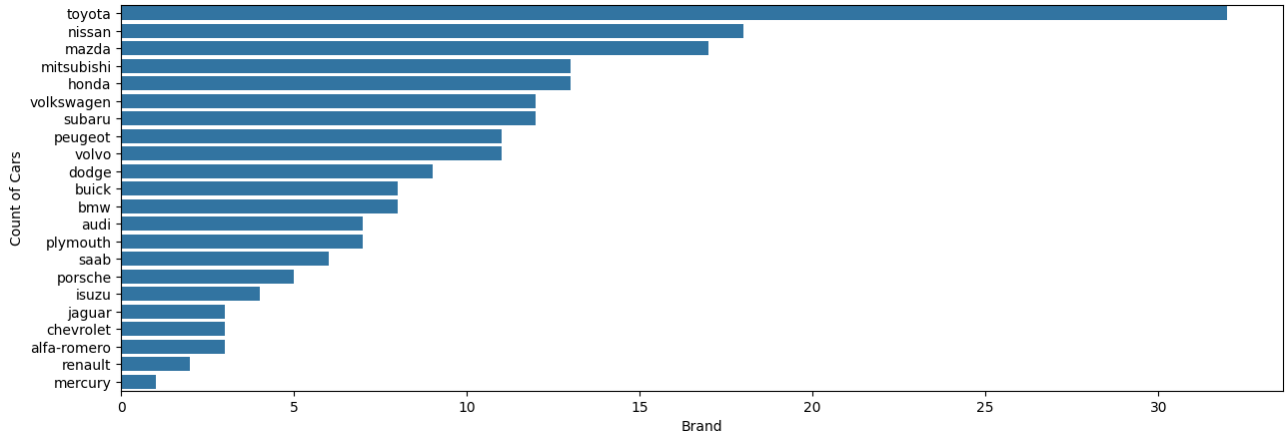
```
df['brand'] = df['brand'].replace(['vw', 'vokswagen'], 'volkswagen')
df['brand'] = df['brand'].replace(['maxda'], 'mazda')
df['brand'] = df['brand'].replace(['porcshce'], 'porsche')
df['brand'] = df['brand'].replace(['toyouta'], 'toyota')
```

```
df.brand.unique()
```

```
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)
```

plot and sort the total number of Brands

```
fig, ax = plt.subplots(figsize = (15,5))
plt1 = sns.countplot(df['brand'], order=pd.value_counts(df['brand']).index,)
plt1.set(xlabel = 'Brand', ylabel= 'Count of Cars')
plt.show()
plt.tight_layout()
```




```
<Figure size 640x480 with 0 Axes>
```

```
df.drop(['car_ID', 'symboling', 'CarName'],axis = 1, inplace = True)
```

```
df.info()
```

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

```
 23  brand             205 non-null    object
dtypes: float64(8), int64(6), object(10)
memory usage: 38.6+ KB
```

```
df_comp_avg_price = df[['brand','price']].groupby('brand', as_index = False).mean().renam
#df = df.merge(df_comp_avg_price, on = 'brand')
#df.brand_avg_price.describe()
#df['brand_category'] = df['brand_avg_price'].apply(lambda x : "Budget" if x < 10000
                                          #else  ("Mid_Range"  if  10000  <=  x  <  2
                                           #               else "Luxury"))
```

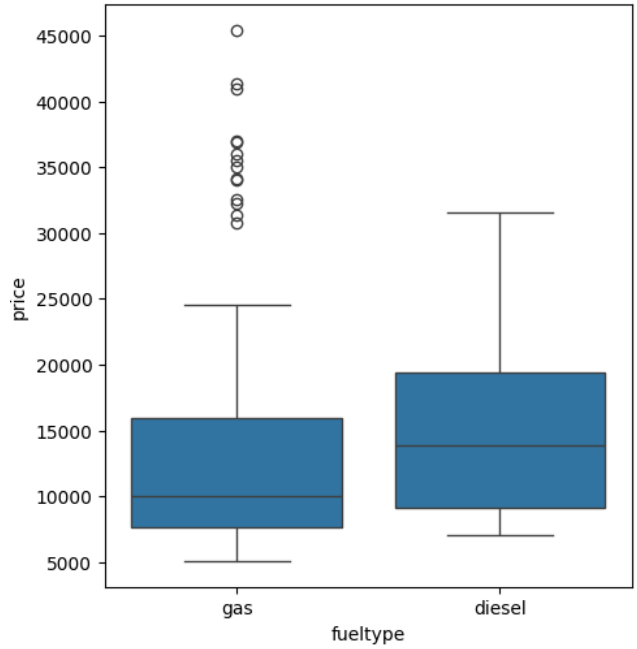
```
df = df.merge(df_comp_avg_price, on = 'brand')
```

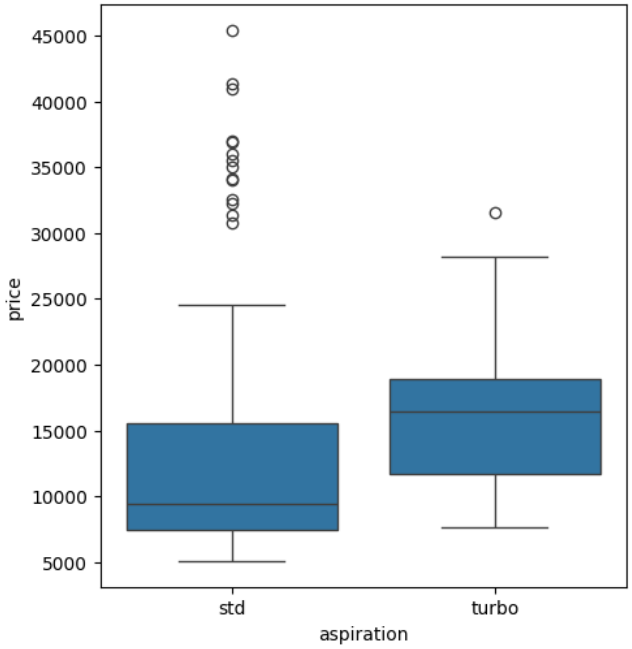
```
df.brand_avg_price.describe()
```

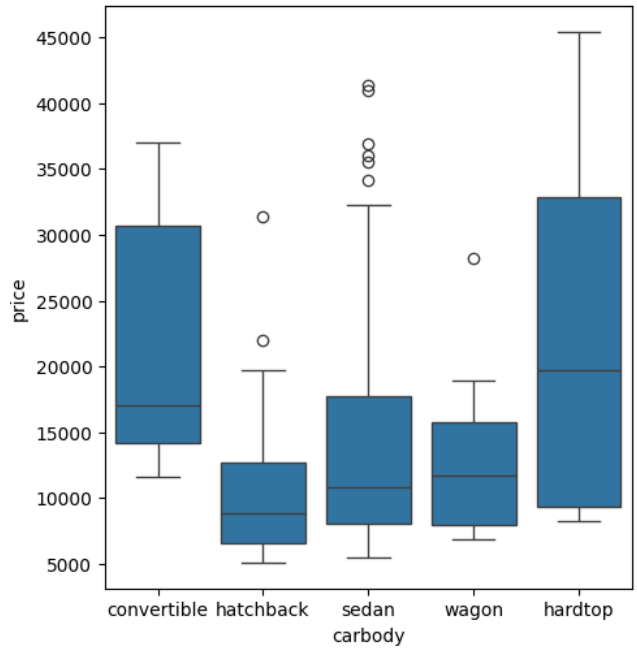
```
count      205.000000
mean     13276.710571
std       7154.179185
min       6007.000000
25%       9239.769231
50%      10077.500000
75%      15489.090909
max      34600.000000
Name: brand_avg_price, dtype: float64
```

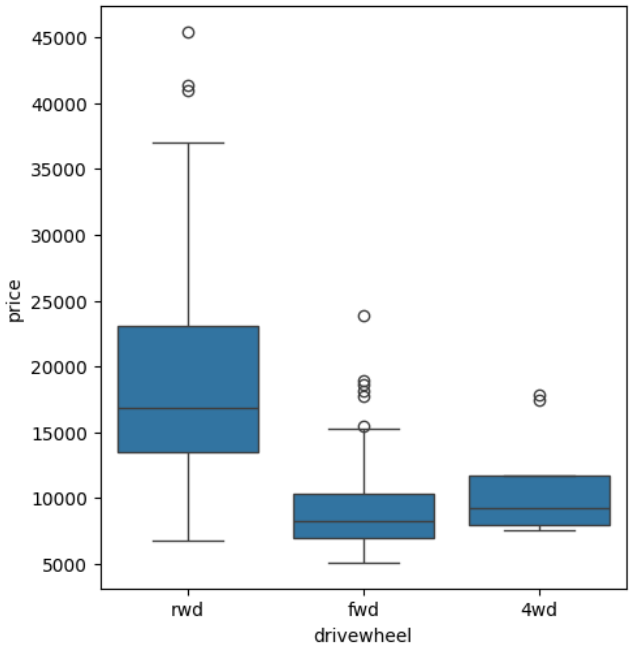
```
df['brand_category'] = df['brand_avg_price'].apply(lambda x : "Budget" if x < 10000
                                                 else ("Mid_Range" if 10000 <= x < 200
                                                      else "Luxury"))
```

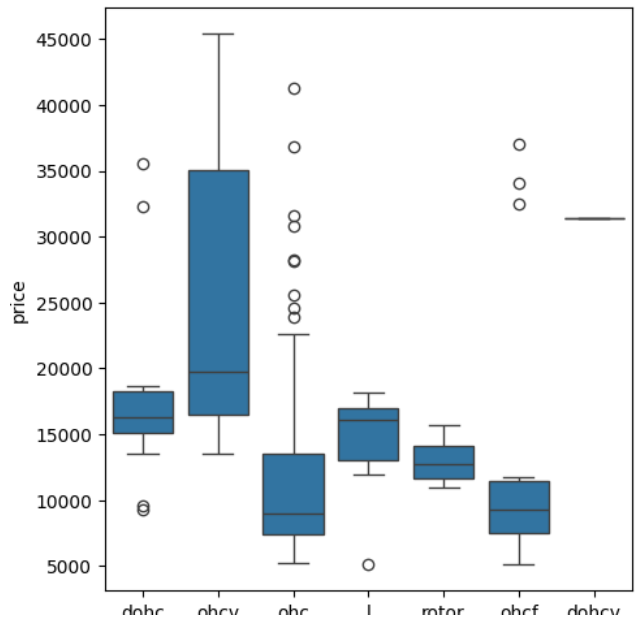
```
plt.figure(figsize=(10, 20))
plt.subplot(4,2,1)
sns.boxplot(x = 'fueltype', y = 'price', data = df)
plt.subplot(4,2,2)
sns.boxplot(x = 'aspiration', y = 'price', data = df)
plt.subplot(4,2,3)
sns.boxplot(x = 'carbody', y = 'price', data = df)
plt.subplot(4,2,4)
sns.boxplot(x = 'drivewheel', y = 'price', data = df)
plt.subplot(4,2,5)
sns.boxplot(x = 'enginetype', y = 'price', data = df)
plt.subplot(4,2,6)
sns.boxplot(x = 'brand_category', y = 'price', data = df)
plt.tight_layout()
plt.show()
```

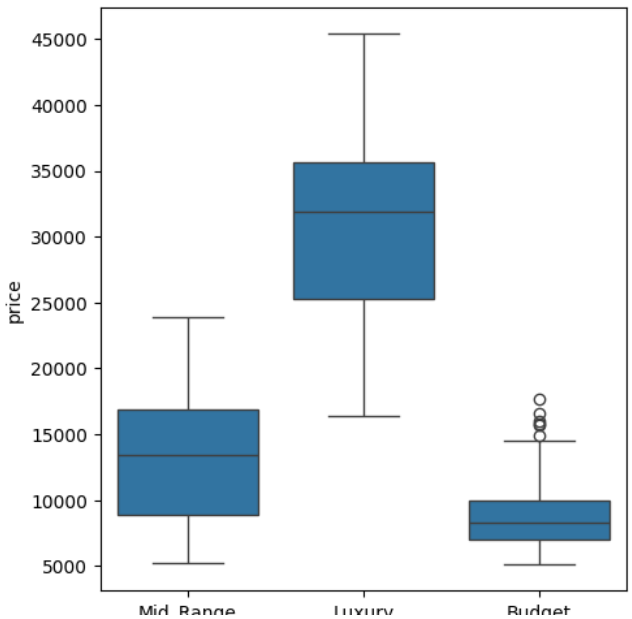

price
fueltype
gas
diesel


price
aspiration
std
turbo


price
carbody
convertible
hatchback
sedan
wagon
hardtop

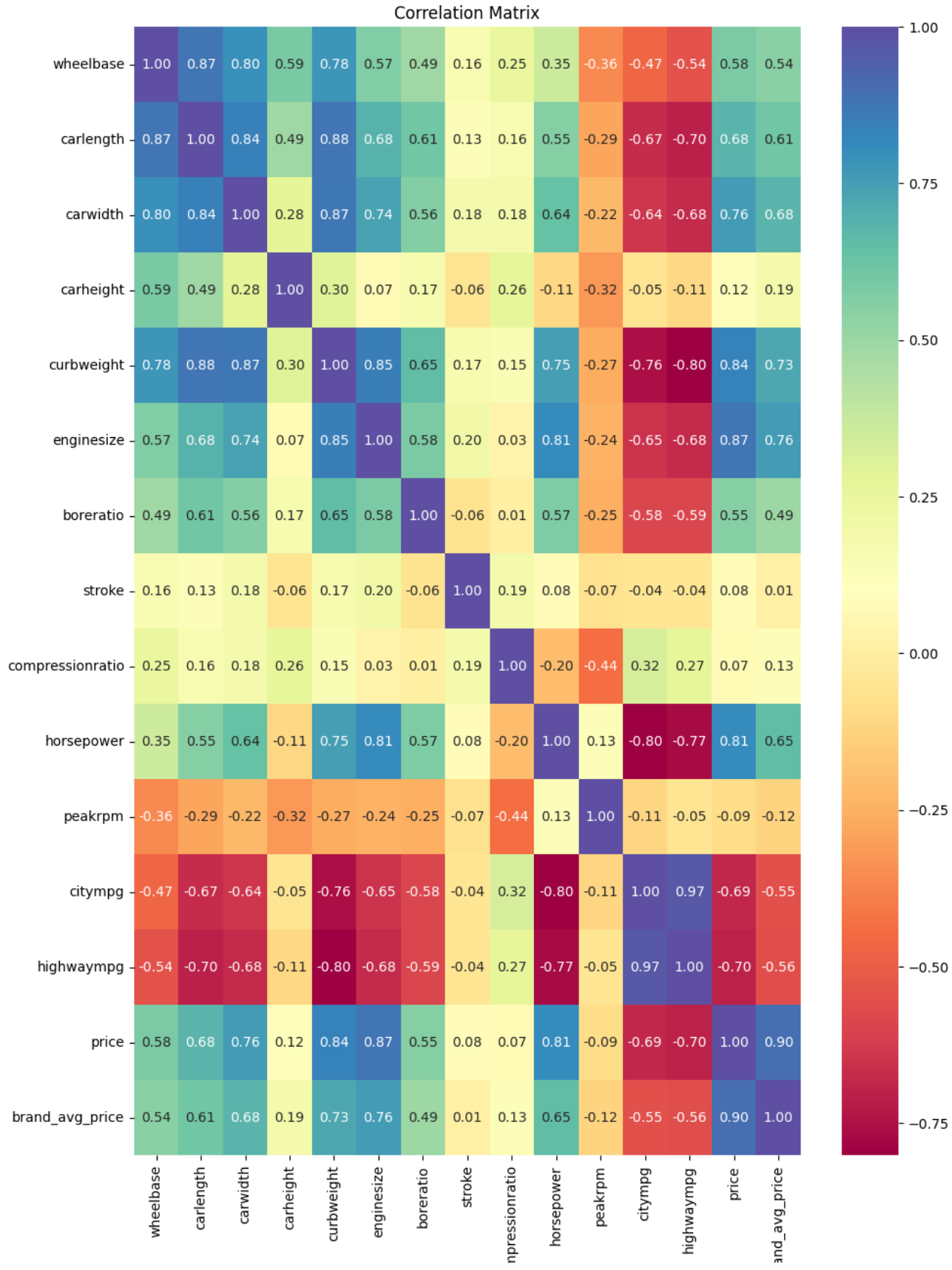

price
drivewheel
rwd
fwd
4wd


price
dohc
ohcv
ohc
l
rotor
ohcf
dohcv

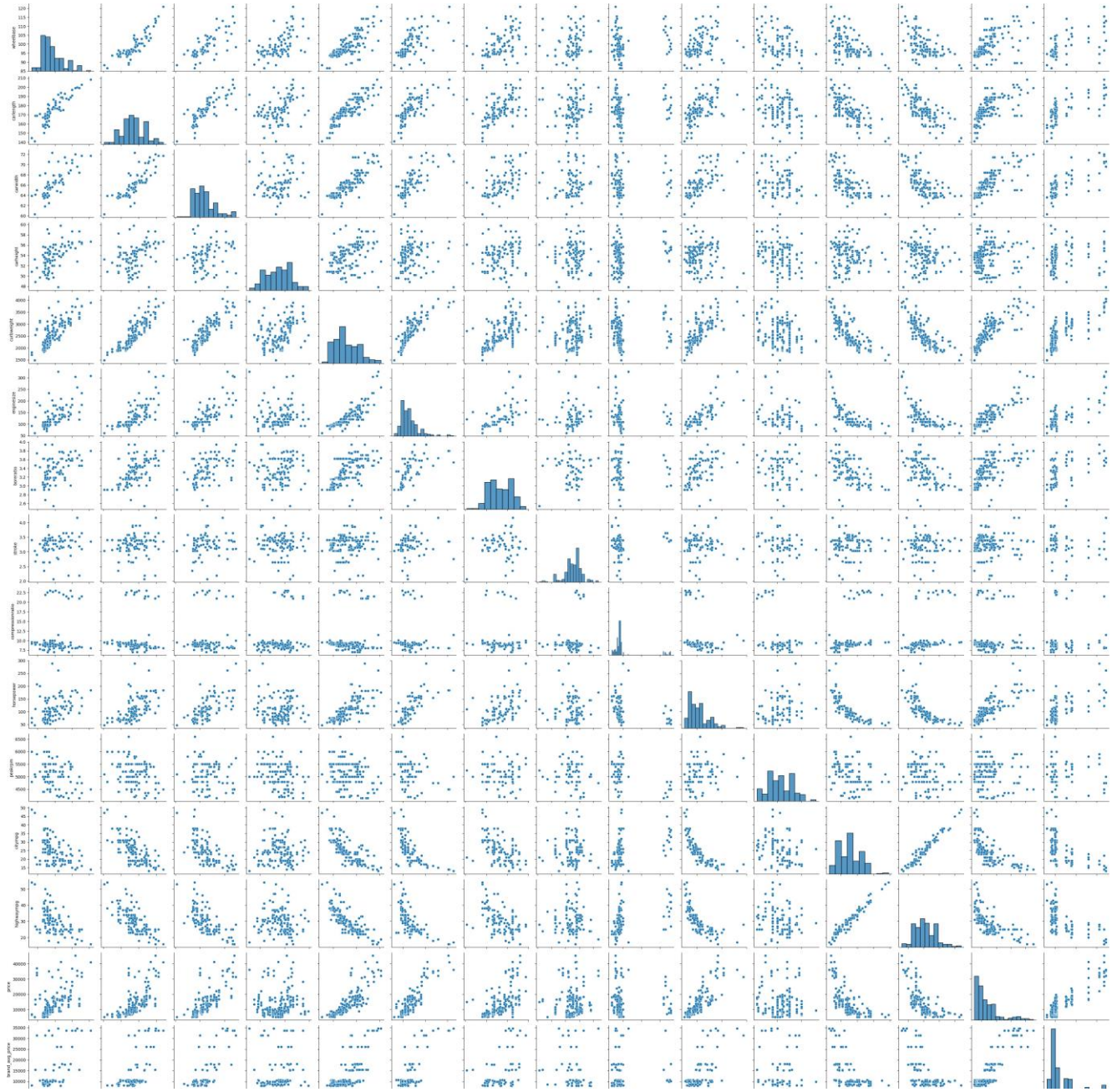

price
Mid_Range
Luxury
Budget

```
corr_matrix = df.corr(numeric_only=True)

corr_matrix['price'].sort_values(ascending=False)
plt.figure(figsize=(11,15))
sns.heatmap(corr_matrix, annot=True, cmap='Spectral', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



```
sns.pairplot(df)
plt.show()
```



Spliting data into training and testing set

```
x=df.drop('price', axis=1)
y=df['price']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
# Encoding categorical variables
from sklearn.preprocessing import OneHotEncoder

# Identify categorical columns
categorical_columns = ['fueltype', 'aspiration', 'carbody', 'drivewheel', 'enginetype', '

# Apply one-hot encoding to categorical variables
encoder = OneHotEncoder(drop='first', sparse=False)
encoded_categorical = encoder.fit_transform(df[categorical_columns])
encoded_categorical_df = pd.DataFrame(encoded_categorical, columns=encoder.get_feature_na

# Drop original categorical columns from the DataFrame
df_encoded = df.drop(columns=categorical_columns)

# Concatenate the encoded categorical columns with the DataFrame
df_encoded = pd.concat([df_encoded, encoded_categorical_df], axis=1)

# Now, you can proceed to split the data and fit the model
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: Futur
  warnings.warn(
```

```
# Check the DataFrame after encoding
print(df_encoded.head())

# Check for any remaining non-numeric values in the DataFrame
print(df_encoded.info())
```

```
0              0.0              0.0              0.0                  0.0
1              0.0              0.0              0.0                  0.0
2              0.0              1.0              0.0                  0.0
3              0.0              0.0              0.0                  0.0
4              0.0              0.0              0.0                  0.0
```

```
 8   enginesize                205 non-null    int64
 9   fuelsystem                205 non-null    object
 10  boreratio                 205 non-null    float64
 11  stroke                    205 non-null    float64
 12  compressionratio          205 non-null    float64
 13  horsepower                205 non-null    int64
 14  peakrpm                   205 non-null    int64
 15  citympg                   205 non-null    int64
 16  highwaympg                205 non-null    int64
 17  price                     205 non-null    float64
 18  brand                     205 non-null    object
 19  brand_avg_price           205 non-null    float64
 20  fueltype_gas              205 non-null    float64
 21  aspiration_turbo          205 non-null    float64
 22  carbody_hardtop           205 non-null    float64
 23  carbody_hatchback         205 non-null    float64
 24  carbody_sedan             205 non-null    float64
 25  carbody_wagon             205 non-null    float64
 26  drivewheel_fwd            205 non-null    float64
 27  drivewheel_rwd            205 non-null    float64
 28  enginetype_dohcv          205 non-null    float64
 29  enginetype_l              205 non-null    float64
 30  enginetype_ohc            205 non-null    float64
 31  enginetype_ohcf           205 non-null    float64
 32  enginetype_ohcv           205 non-null    float64
 33  enginetype_rotor          205 non-null    float64
 34  brand_category_Luxury     205 non-null    float64
 35  brand_category_Mid_Range  205 non-null    float64
dtypes: float64(25), int64(6), object(5)
memory usage: 57.8+ KB
None
```

```
X = df.drop(['price', 'brand_avg_price', 'brand'], axis=1)
y = df['price']

# Encoding categorical variables
categorical_columns = ['fueltype', 'aspiration', 'carbody', 'drivewheel', 'engine
column_transformer = ColumnTransformer([('encoder', OneHotEncoder(), categorical_
X_encoded = column_transformer.fit_transform(X)

# Normalizing price
scaler = MinMaxScaler()
y_normalized = scaler.fit_transform(y.values.reshape(-1, 1)).flatten()

# Splitting into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_normalized, test
```

```
# Check the data type of the 'price' column
print(df['price'].dtype)

# Inspect unique values in the 'price' column
print(df['price'].unique())

# Convert 'price' column to numeric format
df['price'] = pd.to_numeric(df['price'], errors='coerce')
```

```
# Drop rows with NaN values in the 'price' column (if necessary)
df.dropna(subset=['price'], inplace=True)

# Now, proceed with splitting the data and fitting the model
```

```
float64
[13495.    16500.    13950.    17450.    15250.    17710.    18920.
 23875.    17859.167 16430.    16925.    20970.    21105.    24565.
 30760.    41315.    36880.     5151.     6295.     6575.     5572.
  6377.     7957.     6229.     6692.     7609.     8558.     8921.
 12964.     6479.     6855.     5399.     6529.     7129.     7295.
  7895.     9095.     8845.    10295.    12945.    10345.     6785.
  8916.5   11048.    32250.    35550.    36000.     5195.     6095.
  6795.     6695.     7395.    10945.    11845.    13645.    15645.
  8495.    10595.    10245.    10795.    11245.    18280.    18344.
 25552.    28248.    28176.    31600.    34184.    35056.    40960.
 45400.    16503.     5389.     6189.     6669.     7689.     9959.
  8499.    12629.    14869.    14489.     6989.     8189.     9279.
  5499.     7099.     6649.     6849.     7349.     7299.     7799.
  7499.     7999.     8249.     8949.     9549.    13499.    14399.
 17199.    19699.    18399.    11900.    13200.    12440.    13860.
 15580.    16900.    16695.    17075.    16630.    17950.    18150.
 12764.    22018.    32528.    34028.    37028.    31400.5    9295.
  9895.    11850.    12170.    15040.    15510.    18620.     5118.
  7053.     7603.     7126.     7775.     9960.     9233.    11259.
  7463.    10198.     8013.    11694.     5348.     6338.     6488.
  6918.     7898.     8778.     6938.     7198.     7788.     7738.
  8358.     9258.     8058.     8238.     9298.     9538.     8449.
  9639.     9989.    11199.    11549.    17669.     8948.    10698.
  9988.    10898.    11248.    16558.    15998.    15690.    15750.
  7975.     7995.     8195.     9495.     9995.    11595.     9980.
 13295.    13845.    12290.    12940.    13415.    15985.    16515.
 18420.    18950.    16845.    19045.    21485.    22470.    22625.   ]
```

Linear Regression Model

```
# Selecting features and target variable
features = [ 'aspiration', 'carbody', 'drivewheel', 'enginetype', 'brand_category']
X = df[features]
y = df['price']

# Encoding categorical variables if needed
X = pd.get_dummies(X)

# Splitting 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)

# Initializing and fitting the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions on the testing set
y_pred = model.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

```
Mean Squared Error: 14765145.473809367
R-squared Score: 0.8129668933849041
```