

# HW 3

This assignment covers several aspects of Linear Regression. **DO NOT ERASE MARKDOWN CELLS AND INSTRUCTIONS IN YOUR HW submission**

- **Q** - QUESTION
- **A** - Where to input your answer

## Instructions

Keep the following in mind for all notebooks you develop:

- Structure your notebook.
- Use headings with meaningful levels in Markdown cells, and explain the questions each piece of code is to answer or the reason it is there.
- Make sure your notebook can always be rerun from top to bottom (Kernel Tab -> Restart and Run All)
- Start working on this assignment as soon as possible. If you are a beginner in Python this might take a long time. One of the objectives of this assignment is to help you learn python and scikit-learn package.
- Follow [README.md](#) for homework submission instructions
- In this notebook we assume '../data/' location of all data files to be read and written

## Related sklearn material and online tutorials

[sklearn User Guide](#)

### sklearn data pre-processing

- [train\\_test\\_split](#)
- [common\\_pitfalls](#)
- [train test split tutorial](#)

### sklearn multiple linear regression

- [tutorial](#)
- [API documentation](#)

- [Linear Regression](#)
- [multiple linear regression tutorial](#)

## sklearn polynomial regression

- [generate polynomial features](#)
- [polynomial regression tutorial](#)

## correlation

- [correlation](#)

# Linear Regression

In jupyter notebook environment, commands starting with the symbol % are magic commands or magic functions. `%%timeit` is one of such function. It basically gives you the speed of execution of certain statement or blocks of codes.

```
In [47]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

**Q1** Read the `car_data.csv` data (we assume `../data/` location of all data files to be read and written) from **data** folder using pandas. Replace the ??? in the code cell below to accomplish this task.

**A1** Replace ??? with code in the code cell below

```
In [48]: # Replace ??? with code in the code cell below

df = pd.read_csv("../data/Car_data.csv")
```

```
In [49]: # View head of the data to confirm the correctness of your answer
df.head()
```

Out [49]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd
3	4	2	audi 100 ls	gas	std	four	sedan	fwd
4	5	2	audi 100ls	gas	std	four	sedan	4wd

5 rows x 26 columns

## Data cleaning and manipulation

**Q2** Here, you will practice the usage of common data cleaning and manipulation functions in 3 steps.

1. Use `isnull()` to figure out the number of NaN values per column
2. Remove the column with majority NaN values (if any)
3. Check if there are still NaN values in the dataframe using `isna()` method

**A2** Replace ??? with code in the code cell below

```
In [50]: # There is no missing data here on this dataset :
df.isnull().sum()
df.isna().sum()
```

```
Out[50]: 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
```

```
In [51]: # lets get some statistical information :
          df.describe()
```

```
Out[51]:
```

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight
<b>count</b>	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
<b>mean</b>	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565000
<b>std</b>	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680000
<b>min</b>	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000
<b>25%</b>	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000
<b>50%</b>	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000
<b>75%</b>	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000
<b>max</b>	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000

**Q3:** In this task, out of all categorical columns, we focus only on the `fueltype` column processing in 2 steps.

1. Use label encoder from sklearn and convert the `fueltype` categorical values to numerical values.
2. Create a new dataframe that contains only the numerical columns.

**A3** Replace ??? with code in the code cell below.

```
In [52]: # Label Encoding for 2-class columns:
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['fueltype'] = le.fit_transform(df['fueltype'])
```

```
In [53]: # Create new dataframe with selected columns
df=df.select_dtypes(include=[np.number])
```

```
In [54]: df.head()
```

```
Out[54]:
```

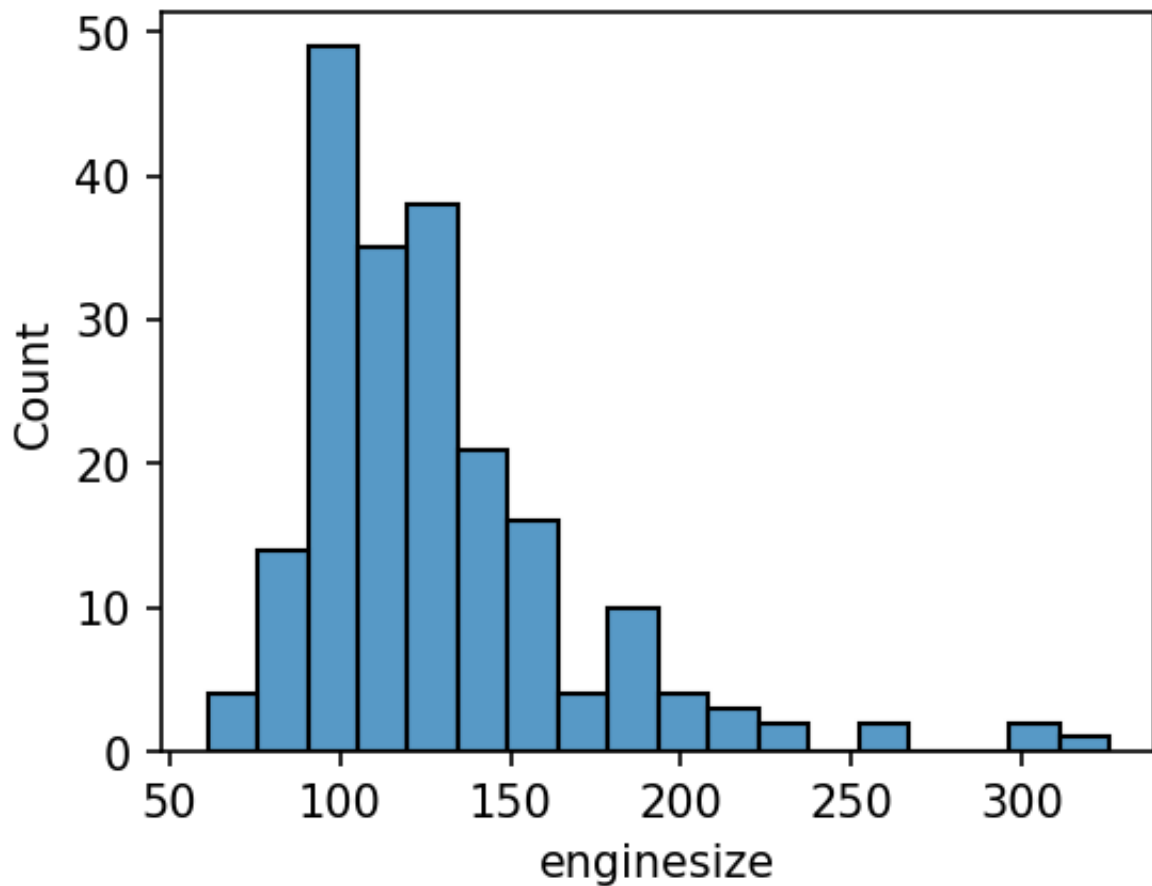
	car_ID	symboling	fueltype	wheelbase	carlength	carwidth	carheight	curbweight	eng
0	1	3	1	88.6	168.8	64.1	48.8	2548	
1	2	3	1	88.6	168.8	64.1	48.8	2548	
2	3	1	1	94.5	171.2	65.5	52.4	2823	
3	4	2	1	99.8	176.6	66.2	54.3	2337	
4	5	2	1	99.4	176.6	66.4	54.3	2824	

**Q4:** Use seaborn histplot to plot a distribution graph for the engine sizes

**A4** Replace ??? with code in the code cell below

```
In [55]: plt.figure(figsize=(4,3),dpi=150)
sns.histplot(df['engine size'])
```

```
Out[55]: <Axes: xlabel='engine size', ylabel='Count'>
```

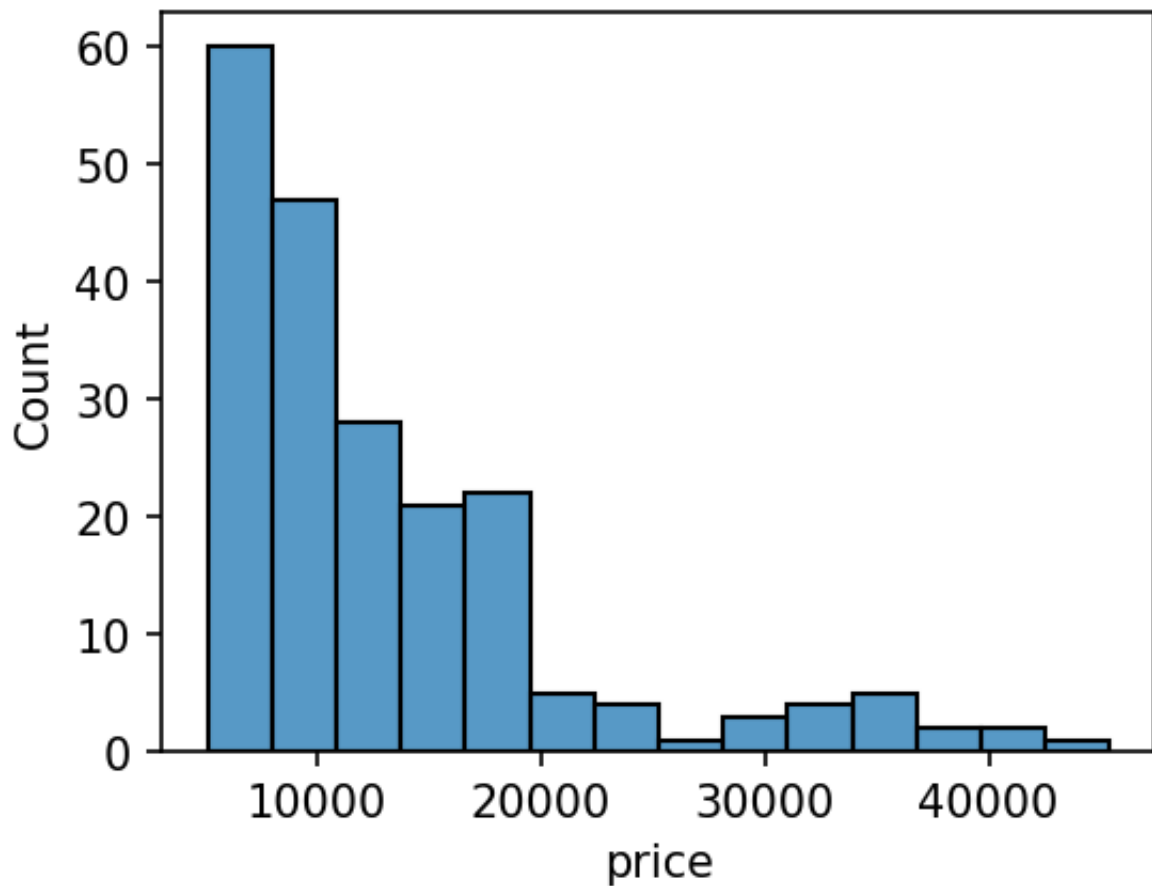


**Q5:** Use seaborn histplot to plot a distribution graph for the car prices

**A5** Replace ??? with code in the code cell below

```
In [56]: plt.figure(figsize=(4,3),dpi=150)
sns.histplot(df['price'])
```

```
Out[56]: <Axes: xlabel='price', ylabel='Count'>
```

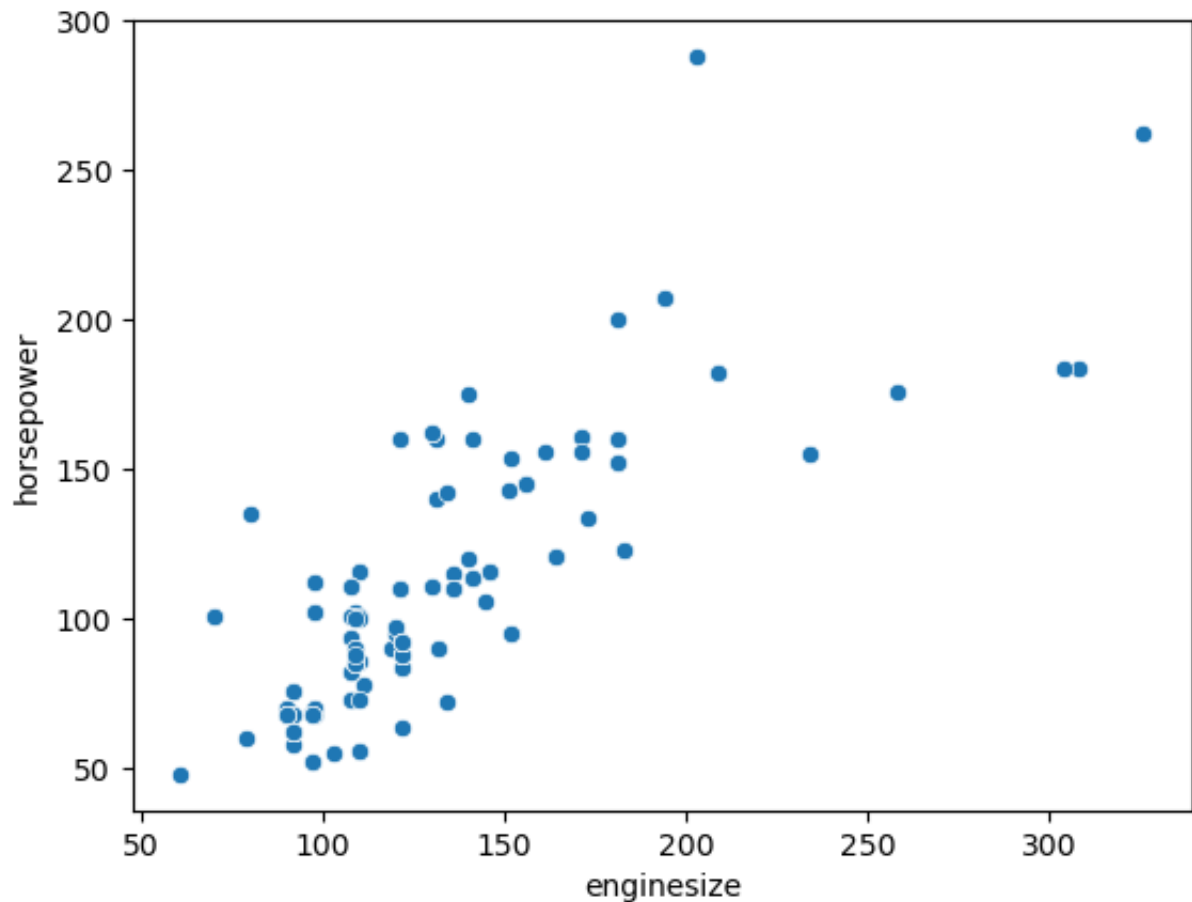


**Q6:** Use seaborn scatterplot to present the relation between enginesize and the horsepower of a car

**A6** Replace ??? with code in the code cell below

```
In [57]: sns.scatterplot(x='enginesize', y='horsepower', data=df)
```

```
Out[57]: <Axes: xlabel='enginesize', ylabel='horsepower'>
```



**Q7:** There is a correlation between the car price and the horsepower of a car. If horsepower of a car increase, the price of the car also increases most of the time, and in this question you will use the seaborn scatterplot to present the relation between price and horsepower.

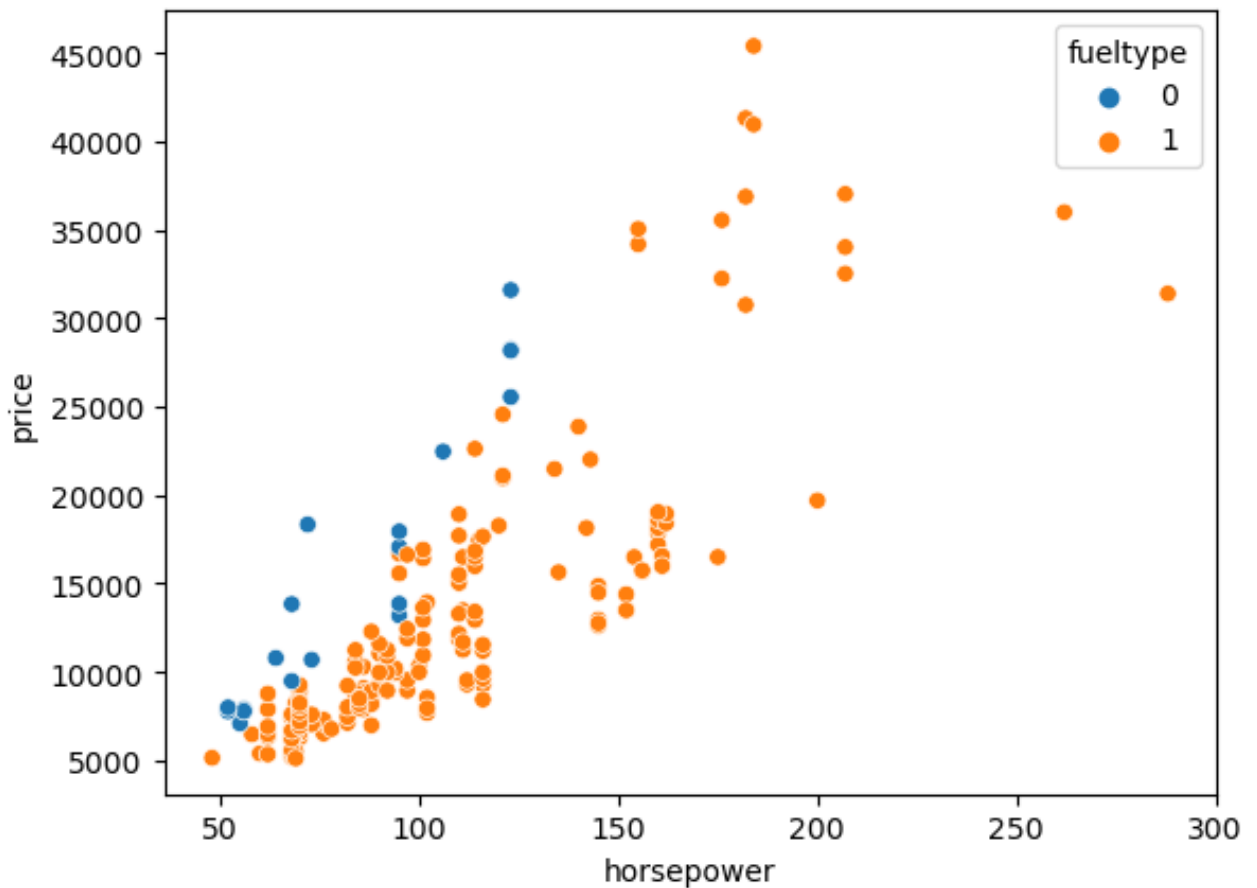
Next, use `hue` parameter of scatterplot function to illustrate datapoints that relate to specific fueltype category.

**A7** Replace ??? with code in the code cell below

```
In [58]: sns.scatterplot(x='horsepower', y='price', hue='fueltype', data=df)
```

```
Out[58]: <Axes: xlabel='horsepower', ylabel='price'>
```





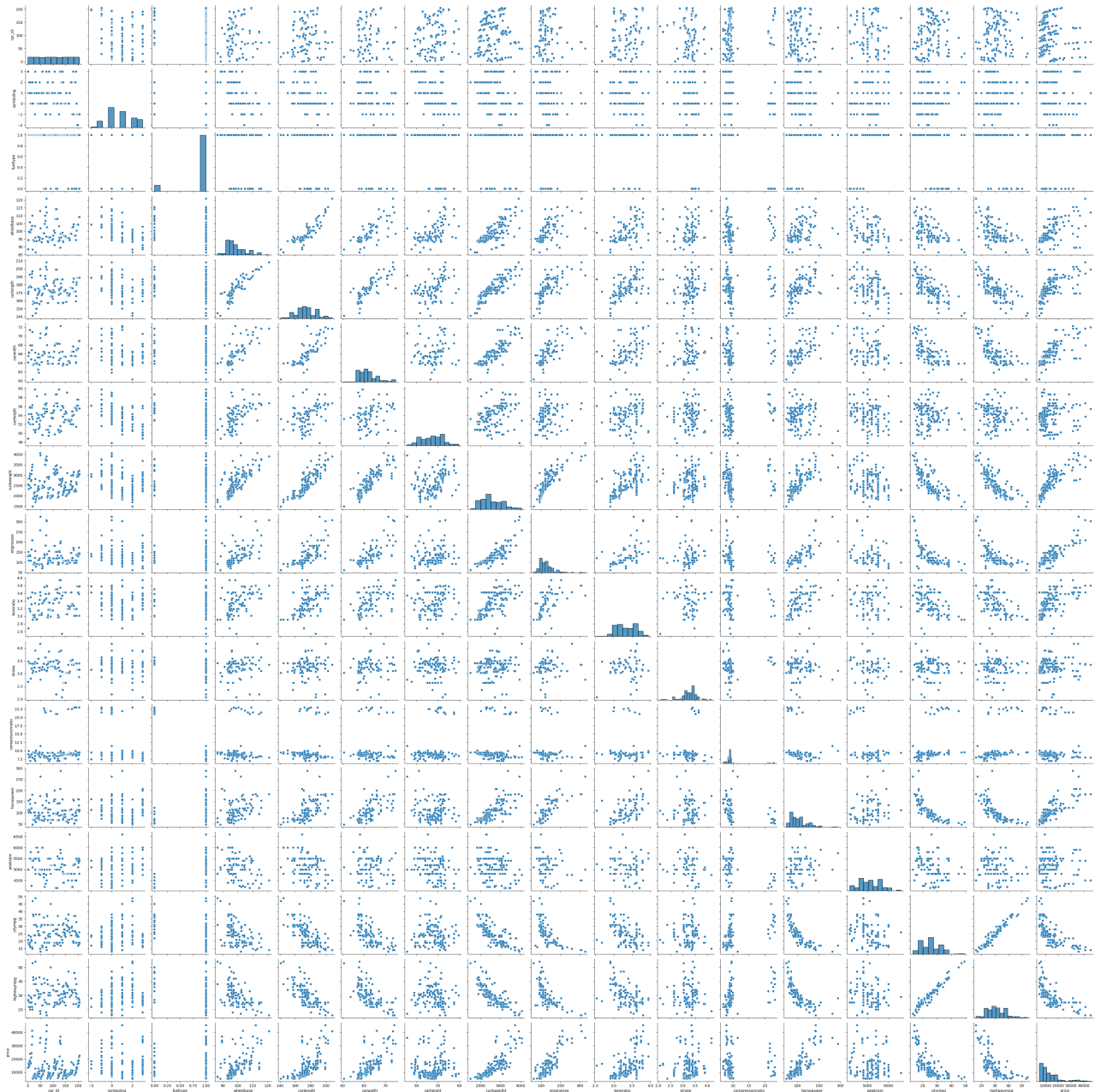
**Q8:** Use pairplot from sns to plot the data frame `df` and justify your feature selection.

**A8:** replace ??? with code in the code cell below.

```
In [59]: # 2. Use pairplot from sns to plot our data frame df
sns.pairplot(df)
```

```
/Users/javier/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:11
8: UserWarning: The figure layout has changed to tight
    self._figure.tight_layout(*args, **kwargs)
```

```
Out[59]: <seaborn.axisgrid.PairGrid at 0x2ad0573d0>
```



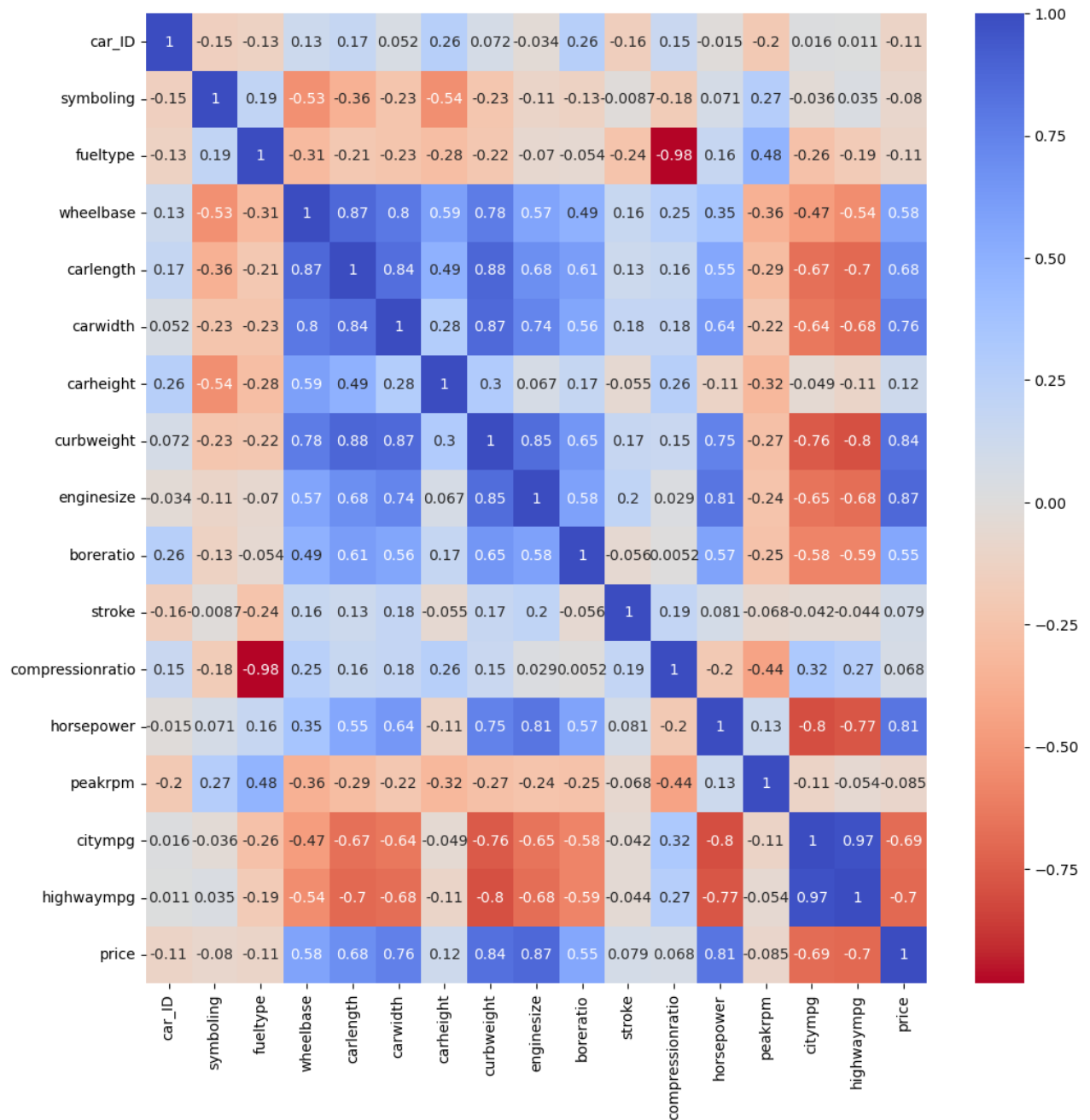
### Q9 Data Visualization:

1. Use heatmap chart from seaborn library to find out the correlation between the columns in our dataset.
2. Update data frame 'df' to contain 5 columns from existing 'df' with the highest correlation to column "price". Also include price column in the updated data frame.

**A9** Replace ??? with code in the code cell below

```
In [60]: corr_matrix = df.corr()
plt.figure(figsize=(12,12))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm_r')
```

Out[60]: <Axes: >



```
In [61]: # Task 2: Update data frame 'df' to contain 5 columns from existing 'df' with
df=df.corr()['price'].abs().sort_values(ascending=False).head(6).index
```

## Data Preparation

**Q10** Pre-processing

1. Assign 'price' column value to y and rest of the columns to x

**A10** Replace ??? with code in the code cell below

```
In [81]: df = pd.read_csv("../data/Car_data.csv")
df = pd.get_dummies(df)
y = df['price']
X = df.drop(columns='price')
```

**Q11** Use train\_test\_split to split the data set as train:test=(80%:20%) ratio.**A11** Replace ??? with code in the code cell below

```
In [82]: from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# View the shape of your data set
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
Out[82]: ((164, 200), (41, 200), (164,), (41,))
```

## Regression Task

### Multiple Linear Regression

**Q12** Fit multiple linear regression model on training data using all predictors, see (i) [Linear Regression Example](#); (ii) [scikit-learn linear model](#)

$$Y = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \dots + \beta_p * x_p$$

**A12:** Replace ??? with code in the code cell below

```
In [83]: from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
```

```
Out[83]: ▼ LinearRegression
LinearRegression()
```

**Q13: Model Scoring**

1. Calculate the test MSE
2. Print the score from the model using test data

**A13** Replace ??? with code in the code cell below

```
In [85]: # Calculate the score on train and test sets
# Your code goes below
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
y_pred=linear_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred) # Calculate the test MSE
print("Test mean squared error (MSE): {:.2f}".format(mse))
```

Test mean squared error (MSE): 178507387.43

## Polynomial Regression

**Q14:** Polynomial extension of the feature set captures the non-linear dependencies in the data

1. Create a polynomial feature transformer with degree **TWO** using sklearn library [PolynomialFeatures](#)
2. Transform the training dataset using the polynomial feature transformer

**A14** Replace ??? with code in the code cell below

```
In [86]: from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
poly_features = poly.fit_transform(X_train)
```

**Q15:** Train the new model

1. Create a LinearRegression model using sklearn
2. Train the model using the transformed Train data(X\_train)/ or Polynomial train data
3. Print the score for the Polynomial Regression for the Train data.

See (i) [Linear Regression Example](#); (ii) Use the transformed X\_train features inside the score() function for the correct model scores.

**A15** Replace ??? with code in the code cell below

```
In [87]: poly_reg_model = LinearRegression()
poly_reg_model.fit(poly_features, y_train)

print("Polynomial Regression train score: {:.2f}".format(poly_reg_model.score))
Polynomial Regression train score: 1.00
```

```
In [ ]:
```