Prepare the dataset

Before you start building the dataset must be prepared. First, execute the code in the cell below to load the packages required to run the rest of this notebook.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
```

Let’s check the first entries of dataset

In [2]:
```
# Using the Csv file
df = pd.read_csv("Automobile price data _Raw_.csv")

# Checking the entries of dataset
df.head()
```

Out[2]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...

5 rows × 26 columns

To analyze we need to find the info about data

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   aspiration         205 non-null    object 
 5   num-of-doors       205 non-null    object 
 6   body-style         205 non-null    object 
 7   drive-wheels       205 non-null    object 
 8   engine-location    205 non-null    object 
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64  
 14  engine-type        205 non-null    object 
 15  num-of-cylinders   205 non-null    object 
 16  engine-size        205 non-null    int64  
 17  fuel-system        205 non-null    object 
 18  bore               205 non-null    object 
 19  stroke             205 non-null    object 
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object 
 22  peak-rpm           205 non-null    object 
 23  city-mpg           205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price              205 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

Finding the missing value

In [4]:
```
data = df

# Finding the missing values
data.isna().any()

# Finding if missing values
data.isnull().any()
```

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

Here, price is of object type(string), it should be int or float, so we need to change it

In [6]:
```
# Here it contains '?', so we Drop it
data = data[data.price != '?']
data['price']=data['price'].astype(int)

# checking it again
data.dtypes
```

Out[6]:
```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                  int32
dtype: object
```

Then came to find that there were lot of ? symbols so in order to convert we need to first change the ? to NAN

In [16]:
```
# replace "?" to NaN
df.replace("?", np.nan, inplace = True)
df.head()
```

Out[16]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	.
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	.
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	.
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	.
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	.
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	.

5 rows × 26 columns

Next Step To find is to check data types

In [20]:
```
df.dtypes
```

Out[20]:
```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

Count missing values in each column

using a for loop in Python, we can quickly figure out the number of missing values in each column.

In the body of the for loop the method ".value_counts()" counts the number of "True" values.

In [22]:
```
missing_data = df.isnull().sum()
missing_data.sort_values(inplace=True, ascending=False)
missing_data.head()
```

Out[22]:
```
normalized-losses    41
price                 4
stroke                4
bore                  4
peak-rpm              2
dtype: int64
```

In General Whole columns should be dropped only if most entries in the column are empty. In our dataset, none of the columns are empty enough to drop entirely. so we need to deal with mean

enough to drop entirely. So we need to deal with mean

Replace by mean:

"normalized-losses": 41 missing data, replace them with mean "stroke": 4 missing data, replace them with mean "bore": 4 missing data, replace them with mean "horsepower": 2 missing data, replace them with mean "peak-rpm": 2 missing data, replace them with mean

Replace by frequency:

"num-of-doors": 2 missing data, replace them with "four". Reason: most of the sedans is four doors. Since four doors is most frequent, it is most likely to occur

Drop the whole row:

"price": 4 missing data, simply delete the whole row Reason: price is what we want to predict. Any data entry without price data cannot be used for prediction; therefore any row now without price data is not useful

Calculate the average of the columns

In [23]:

```
avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0)
print("Average of normalized-losses: ", avg_norm_loss)

avg_bore = df['bore'].astype('float').mean(axis=0)
print("Average of bore: ", avg_bore)

avg_stroke = df["stroke"].astype("float").mean(axis = 0)
print("Average of stroke:", avg_stroke)

avg_horsepower = df['horsepower'].astype('float').mean(axis=0)
print("Average horsepower:", avg_horsepower)

avg_peakrpm = df['peak-rpm'].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peakrpm)
```

```
Average of normalized-losses:  122.0
Average of bore:  3.3297512437810957
Average of stroke: 3.2554228855721337
Average horsepower: 104.25615763546799
Average peak rpm: 5125.369458128079
```

Replace "NaN" by mean value in columns

In [24]:

```
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
df["stroke"].replace(np.nan, avg_stroke, inplace = True)
df["bore"].replace(np.nan, avg_bore, inplace=True)
df['horsepower'].replace(np.nan, avg_horsepower, inplace=True)
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
```

To see which values are present in a particular column, we can use the "value_counts() method

In [25]:
```
df['num-of-doors'].value_counts()
```

Out[25]:
```
four    114
two      89
Name: num-of-doors, dtype: int64
```

replace the missing 'num-of-doors' values by the most frequent

In [26]:
```
df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

In [27]:
```
# simply drop whole row with NaN in "price" column
df.dropna(subset=["price"], axis=0, inplace=True)

# reset index, because we droped two rows
df.reset_index(drop=True, inplace=True)
df.head()
```

Out[27]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	.
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	.
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	.
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	.
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	.
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	.

5 rows × 26 columns

finally we attain the correct data set

The last step is data cleaning to check the whether the data types are correct or not

In [28]:
```
df.dtypes
```

Out[28]:
```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

As we can see above, some columns are not of the correct data type. Numerical variables should have type 'float' or 'int', and variables with strings such as categories should have type 'object'. For example, 'bore' and 'stroke' variables are numerical values that describe the engines, so we should expect them to be of the type 'float' or 'int'; however, they are shown as type 'object'. We have to convert data types into a proper format for each column using the "astype()" method.

In [29]:
```
df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
df[["price"]] = df[["price"]].astype("float")
df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
```

In [30]:
```
df.dtypes
```

Out[30]:
```
symboling              int64
normalized-losses      int32
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower            object
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
dtype: object
```

finally we obtain the cleaned data set with no missing values

In [37]:
```
df.head()
```

Out[37]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	.
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	.
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	.
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	.
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	.
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	.

5 rows × 26 columns

In [38]:
```
# Convert mpg to L/100km by mathematical operation (235 divided by mpg)
df['city-L/100km'] = 235/df["city-mpg"]

# check your transformed data
df.head()
```

Out[38]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...

5 rows × 27 columns

In [ ]: