Importing Dataset Practice Notebook

Data Acquisition

There are various formats for a dataset, .csv, .json, .xlsx etc. The dataset can be stored in different places, on your local machine or sometimes online.
In this section, you will learn how to load a dataset into our Jupyter Notebook.
In our case, the Automobile Dataset is an online source, and it is in CSV (comma separated value) format. Let's use this dataset as an example to practice data reading.

- dataset name: dataset_1.data

The Pandas Library is a useful tool that enables us to read various datasets into a data frame; our Jupyter notebook platforms have a built-in **Pandas Library** so that all we need to do is import Pandas without installing.

Importing Libraries

In [1]:
```
import pandas as pd
```

Importing datasets

We use `pandas.read_csv()` function to read the csv file. In the bracket, we put the file path along with a quotation mark, so that pandas will read the file into a data frame from that address. The file path can be either an URL or your local file address.

Name of Dataset is "dataset_1.data"

In [3]:
```
headers_data = ["symboling","normalized-losses","make","fuel-type","aspiration",
         "drive-wheels","engine-location","wheel-base", "length","width","height"
         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compres
         "peak-rpm","city-mpg","highway-mpg","price"]
print("headers\n", type(headers_data))
```

```
headers
 <class 'list'>
```

In [5]:
```
#Read the dataset in "df" variable
df=pd.read_csv('dataset_1.data',names=headers_data)
```

After reading the dataset, we can use the `dataframe.head(n)` method to check the top n rows

of the dataframe; where n is an integer. Contrary to `dataframe.head(n)`, `dataframe.tail(n)` will show you the bottom n rows of the dataframe.

In [10]:
```
# show the first 5 rows using dataframe.head() method
print('The First 5 Rows of the Dataframe')
df.head(5)
```

```
The First 5 Rows of the Dataframe
```

Out[10]:

	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

In [ ]:
```

```

In [11]:
```
#show the last 10 rows of the dataframe.
print('The First 10 rows of the DataFrame')
df.tail(10)
```

```
The First 10 rows of the DataFrame
```

Out[11]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base
195	-1	74	volvo	gas	std	four	wagon	rwd	front	104.3
196	-2	103	volvo	gas	std	four	sedan	rwd	front	104.3
197	-1	74	volvo	gas	std	four	wagon	rwd	front	104.3
198	-2	103	volvo	gas	turbo	four	sedan	rwd	front	104.3
199	-1	74	volvo	gas	turbo	four	wagon	rwd	front	104.3
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1

10 rows × 26 columns

Basic Insight of Dataset

After reading data into Pandas dataframe, it is time for us to explore the dataset.
There are several ways to obtain essential insights of the data to help us better understand our dataset.

Data Types

Data has a variety of types.
The main types stored in Pandas dataframes are **object**, **float**, **int**, **bool** and **datetime64**. In order to better learn about each attribute, it is always good for us to know the data type of each column.
In Pandas:

Syntax : dataframe.dtypes

returns a Series with the data type of each column.

In [12]:
```
# check the data type of data frame "df" by .dtypes
df.dtypes
```

Out[12]:
```
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
```

As a result, as shown above, it is clear to see that the data type of "symboling" and "curb-weight" are `int64`, "normalized-losses" is `object`, and "wheel-base" is `float64`, etc.

These data types can be changed; we will learn how to accomplish this in a later module.

Describe

If we would like to get a statistical summary of each column, such as count, column mean value, column standard deviation, etc. We use the describe method:

```
Syntax :
    dataframe.describe()
```

This method will provide various summary statistics, excluding `NaN` (Not a Number) values.

In [13]:
```
df.describe()
```

Out[13]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	comp
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	20
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	1
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	2

This shows the statistical summary of all numeric-typed (int, float) columns.
For example, the attribute "symboling" has 205 counts, the mean value of this column is 0.83, the standard deviation is 1.25, the minimum value is -2, 25th percentile is 0, 50th percentile is 1, 75th percentile is 2, and the maximum value is 3.
However, what if we would also like to check all the columns including those that are of type object.

You can add an argument `include = "all"` inside the bracket. Let's try it again.

In [14]:
```
# describe all the columns in "df"
df.describe(include='all')
```

Out[14]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe ba
count	205.000000	205	205	205	205	205	205	205	205	205.0000
unique	NaN	52	22	2	2	3	5	3	2	N
top	NaN	?	toyota	gas	std	four	sedan	fwd	front	N
freq	NaN	41	32	185	168	114	96	120	202	N
mean	0.834146	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	98.7565
std	1.245307	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.0217
min	-2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	86.6000
25%	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	94.5000
50%	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	97.0000
75%	2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	102.4000
max	3.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	120.9000

11 rows × 26 columns

Now, it provides the statistical summary of all the columns, including object-typed attributes.
We can now see how many unique values, which is the top value and the frequency of top value in the object-typed columns.
Some values in the table above show as "NaN", this is because those numbers are not available regarding a particular column type.

In [7]:
```
#Replacing "?" with np.nan so that pandas can recognize the null values.
```

Info

Another method you can use to check your dataset is:

```
Syntax :
    dataframe.info()
```

It provide a concise summary of your DataFrame.

This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

In [15]:
```
# look at the info of "df"
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
```

Save Dataset

Correspondingly, Pandas enables us to save the dataset to csv by using the `dataframe.to_csv()` method, you can add the file path and name along with quotation marks in the brackets.

For example, if you would save the dataframe **df** as **automobile.csv** to your local machine, you may use the syntax below:

```
df.to_csv("AE.csv",index=True)
```

We can also read and save other file formats, we can use similar functions to **`pd.read_csv()`** and **`df.to_csv()`** for other data formats, the functions are listed in the following table:

Read/Save Other Data Formats

Data Formate	Read	Save
csv	`pd.read_csv()`	`df.to_csv()`
json	`pd.read_json()`	`df.to_json()`
excel	`pd.read_excel()`	`df.to_excel()`
hdf	`pd.read_hdf()`	`df.to_hdf()`
sql	`pd.read_sql()`	`df.to_sql()`
...	...	...