IBM Developer
SKILLS NETWORK

Introduction Notebook

Estimated time needed: **10** minutes

Objectives

After completing this lab you will be able to:

- Acquire data in various ways
- Obtain insights from data with Pandas library

Table of Contents

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 a CSV (comma separated value) format. Let's use this dataset as an example to practice data reading.

- Data source: https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data (https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100 SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDA0101ENSkillsNetwork20235326-2022-01-01)
- Data type: csv

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

In [ ]:

```
#install specific version of libraries used in  lab
#! mamba install pandas==1.3.3  -y
#! mamba install numpy=1.21.2 -y
```

In [1]:

```
# import pandas library
import pandas as pd
import numpy as np
```

Read Data

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

Because the data does not include headers, we can add an argument `headers = None` inside the `read_csv()` method so that pandas will not automatically set the first row as a header.

You can also assign the dataset to any variable you create.

This dataset was hosted on IBM Cloud object. Click HERE (https://cocl.us/DA101EN_object_storage?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=1000655 SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDA0101ENSkillsNetwork20235326-2022-01-01) for free storage.

In [2]:

```
# Import pandas library
import pandas as pd

# Read the online file by the URL provides above, and assign it to variable "df"
other_path = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
df = pd.read_csv(other_path, header=None)
```

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 [3]: # 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[3]:

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115

5 rows × 26 columns

df.tail(10)

Question #1:

Check the bottom 10 rows of data frame "df".

In [4]:
```
# Write your code below and press Shift+Enter to execute
df.tail(10)
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	
195	-1	74	volvo	gas	std	four	wagon	rwd	front	104.3	...	141	mpfi	3.78	3.15	9.5	1
196	-2	103	volvo	gas	std	four	sedan	rwd	front	104.3	...	141	mpfi	3.78	3.15	9.5	1
197	-1	74	volvo	gas	std	four	wagon	rwd	front	104.3	...	141	mpfi	3.78	3.15	9.5	1
198	-2	103	volvo	gas	turbo	four	sedan	rwd	front	104.3	...	130	mpfi	3.62	3.15	7.5	1
199	-1	74	volvo	gas	turbo	four	wagon	rwd	front	104.3	...	130	mpfi	3.62	3.15	7.5	1
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	1
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	1
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	1
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	1
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	1

10 rows × 26 columns

Click here for the solution

Add Headers

Take a look at our dataset. Pandas automatically set the header with an integer starting from 0.

To better describe our data, we can introduce a header. This information is available at: https://archive.ics.uci.edu/ml/datasets/Automobile (https://archive.ics.uci.edu/ml/datasets/Automobile?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100065 SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDA0101ENSkillsNetwork20235326-2022-01-01).

Thus, we have to add headers manually.

First, we create a list "headers" that include all column names in order. Then, we use `dataframe.columns = headers` to replace the headers with the list we created.

In [5]:
```
# create headers list
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-
         "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", headers)
```

```
headers
 ['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-
doors', 'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'lengt
h', 'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders', 'engin
e-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio', 'horsepower', 'p
eak-rpm', 'city-mpg', 'highway-mpg', 'price']
```

We replace headers and recheck our dataframe:

In [6]:
```
df.columns = headers
df.head(10)
```

Out[6]:

	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	2	?	audi	gas	std	two	sedan	fwd	front	99.8	.
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	.
7	1	?	audi	gas	std	four	wagon	fwd	front	105.8	.
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	.
9	0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	.

10 rows × 26 columns

We need to replace the "?" symbol with NaN so the dropna() can remove the missing values:

In [9]:

```
df1=df.replace('?',np.NaN)
df1
```

Out[9]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel bas
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.
3	2	164	audi	gas	std	four	sedan	fwd	front	99.
4	2	164	audi	gas	std	four	sedan	4wd	front	99.
...	...	...	...	...	...	...	...	...	...	..
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.

201 rows × 26 columns

We can drop missing values along the column "price" as follows:

In [8]:
```
df=df1.dropna(subset=["price"], axis=0)
df.head(20)
```

Out[8]:

	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	2	NaN	audi	gas	std	two	sedan	fwd	front	99.8
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8
7	1	NaN	audi	gas	std	four	wagon	fwd	front	105.8
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8
10	2	192	bmw	gas	std	two	sedan	rwd	front	101.2
11	0	192	bmw	gas	std	four	sedan	rwd	front	101.2
12	0	188	bmw	gas	std	two	sedan	rwd	front	101.2
13	0	188	bmw	gas	std	four	sedan	rwd	front	101.2
14	1	NaN	bmw	gas	std	four	sedan	rwd	front	103.5
15	0	NaN	bmw	gas	std	four	sedan	rwd	front	103.5
16	0	NaN	bmw	gas	std	two	sedan	rwd	front	103.5
17	0	NaN	bmw	gas	std	four	sedan	rwd	front	110.0
18	2	121	chevrolet	gas	std	two	hatchback	fwd	front	88.4
19	1	98	chevrolet	gas	std	two	hatchback	fwd	front	94.5
20	0	81	chevrolet	gas	std	four	sedan	fwd	front	94.5

20 rows × 26 columns

Now, we have successfully read the raw dataset and added the correct headers into the dataframe.

Question #2:

Find the name of the columns of the dataframe.

In [10]:
```
# Write your code below and press Shift+Enter to execute
df.columns
```

Out[10]:
```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

Click here for the solution

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, where `index = False` means the row names will not be written.

```
df.to_csv("automobile.csv", index=False)
```

We can also read and save other file formats. We can use similar functions like **`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()`
...	...	...

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:

In [11]:
```
df.dtypes
```

Out[11]:
```
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
```

A series with the data type of each column is returned.

In [14]:

```
# check the data type of data frame "df" by .dtypes
print(df.dtypes)
```

```
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 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 e.g. count, column mean value, column standard deviation, etc., we use the describe method:

```
dataframe.describe()
```

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

In [15]:
```
df.describe()
```

Out[15]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	comp
count	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	20
mean	0.840796	98.797015	174.200995	65.889055	53.766667	2555.666667	126.875622	1
std	1.254802	6.066366	12.322175	2.101471	2.447822	517.296727	41.546834	
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	
25%	0.000000	94.500000	166.800000	64.100000	52.000000	2169.000000	98.000000	
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	
75%	2.000000	102.400000	183.500000	66.600000	55.500000	2926.000000	141.000000	
max	3.000000	120.900000	208.100000	72.000000	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 [16]:
```
# describe all the columns in "df"
df.describe(include = "all")
```

Out[16]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe ba
count	201.000000	164	201	201	201	199	201	201	201	201.0000
unique	NaN	51	22	2	2	2	5	3	2	N
top	NaN	161	toyota	gas	std	four	sedan	fwd	front	N
freq	NaN	11	32	181	165	113	94	118	198	N
mean	0.840796	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	98.7970
std	1.254802	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.0663
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 there, which one 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.

Question #3:

You can select the columns of a dataframe by indicating the name of each column. For example, you can select the three columns as follows:

```
dataframe[[' column 1 ',column 2', 'column 3']]
```

Where "column" is the name of the column, you can apply the method ".describe()" to get the statistics of those columns as follows:

```
dataframe[[' column 1 ',column 2', 'column 3'] ].describe()
```

Apply the method to ".describe()" to the columns 'length' and 'compression-ratio'.

In [17]:
```
# Write your code below and press Shift+Enter to execute
df[['make','price']]
```

Out[17]:
```
            make  price
  0  alfa-romero  13495
  1  alfa-romero  16500
  2  alfa-romero  16500
  3         audi  13950
  4         audi  17450
...          ...    ...
200        volvo  16845
201        volvo  19045
202        volvo  21485
203        volvo  22470
204        volvo  22625

201 rows × 2 columns
```

Click here for the solution

Info

Another method you can use to check your dataset is:

```
dataframe.info()
```

It provides 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 [18]: # look at the info of "df"
         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 201 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          201 non-null    int64  
 1   normalized-losses  164 non-null    object 
 2   make               201 non-null    object 
 3   fuel-type          201 non-null    object 
 4   aspiration         201 non-null    object 
 5   num-of-doors       199 non-null    object 
 6   body-style         201 non-null    object 
 7   drive-wheels       201 non-null    object 
 8   engine-location    201 non-null    object 
 9   wheel-base         201 non-null    float64
 10  length             201 non-null    float64
 11  width              201 non-null    float64
 12  height             201 non-null    float64
 13  curb-weight        201 non-null    int64  
 14  engine-type        201 non-null    object 
 15  num-of-cylinders   201 non-null    object 
 16  engine-size        201 non-null    int64  
 17  fuel-system        201 non-null    object 
 18  bore               197 non-null    object 
 19  stroke             197 non-null    object 
 20  compression-ratio  201 non-null    float64
 21  horsepower         199 non-null    object 
 22  peak-rpm           199 non-null    object 
 23  city-mpg           201 non-null    int64  
 24  highway-mpg        201 non-null    int64  
 25  price              201 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 42.4+ KB
```

Excellent! You have just completed the Introduction Notebook!

Thank you for completing this lab!

Author

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=100065 SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDA0101ENSkillsNetwork20235326-2022-01-01)

Other Contributors

Mahdi Noorian PhD (https://www.linkedin.com/in/mahdi-noorian-58219234/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=1000655
SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDA0101ENSkillsNetwork20235326-2022-01-01)

Bahare Talayian

Eric Xiao

Steven Dong

Parizad

Hima Vasudevan

Fiorella Wenver (https://www.linkedin.com/in/fiorellawever/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=1000655
SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDA0101ENSkillsNetwork20235326-2022-01-01)

Yi Yao (https:// https://www.linkedin.com/in/yi-leng-yao-84451275/).

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-10-30	2.3	Lakshmi	Changed URL of the csv
2020-09-22	2.2	Nayef	Added replace() method to remove '?'
2020-09-09	2.1	Lakshmi	Made changes in info method of dataframe
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab