HASAN FIRAT - Data Analyst & Engineer

[LinkedIn](https://www.linkedin.com/in/hasan-firat-1952a0224/)

[GitHub](https://github.com/hasan-firat-data-and-business-analyst)

[Dataset](https://github.com/hasan-firat-data-and-business-analyst/Data-Science-Python/blob/main/auto.csv)

Pandas: Zero to Hero

1 - Introduction

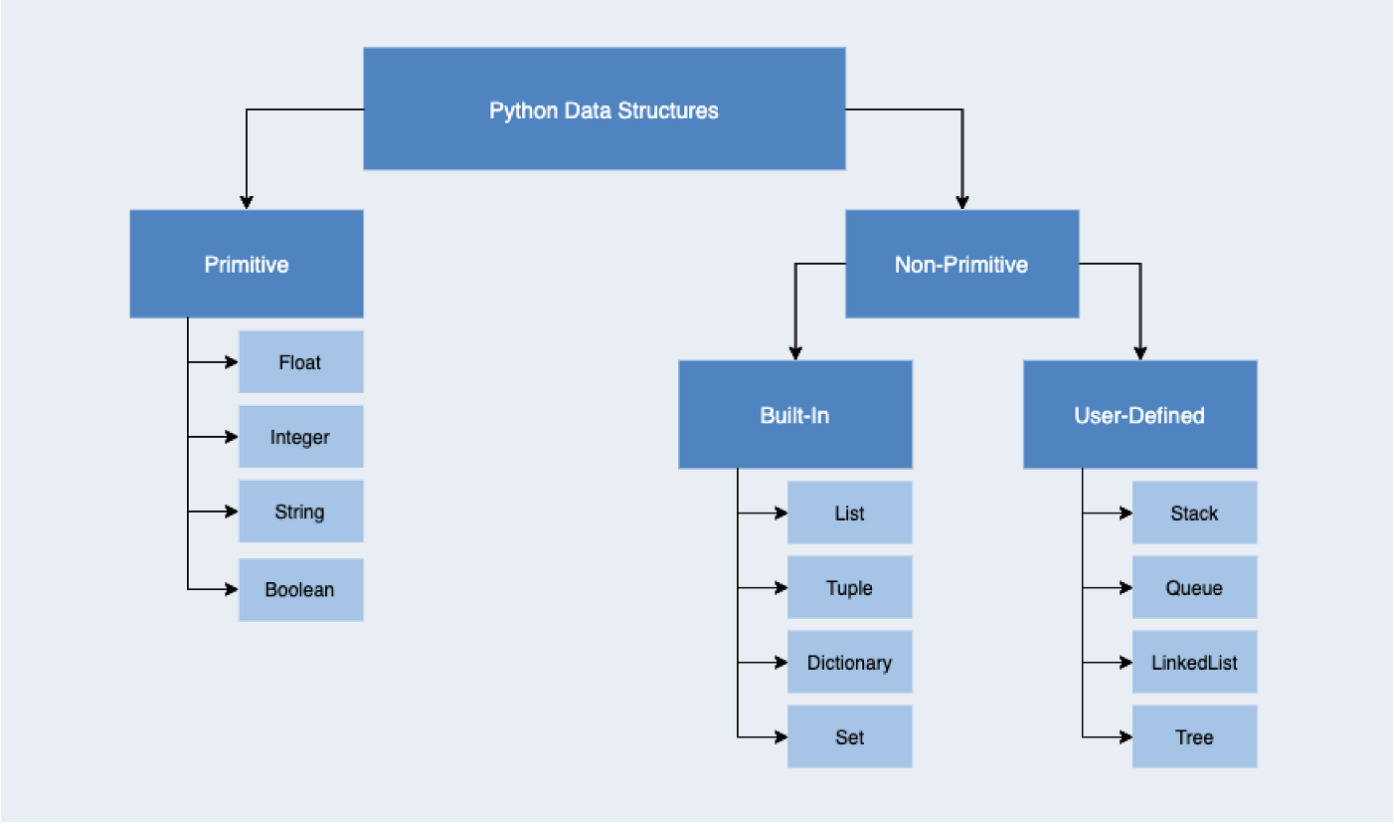
Pandas

Pandas is fast, powerful, flexible and easy to use open source library for data manipulation and analysis. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users. [Reference](https://pandas.pydata.org/) [Reference](https://www.geeksforgeeks.org/pandas-tutorial/?ref=lbp)

History

Developer Wes McKinney started working on pandas in 2008 while at AQR Capital Management out of the need for a high performance, flexible tool to perform quantitative analysis on financial data. Before leaving AQR he was able to convince management to allow him to open source the library. [Reference](https://en.wikipedia.org/wiki/Pandas_(software))

A. Data Structures in Python



1. Primitive Data Structures

a. Float or Floating Point

A floating point (known as a float) number has decimal points even if that decimal point value is 0. Such as, 8.85 , 458.001 or 521.10002 for example.

In [1]:
```
floating_point = 11.0
print(floating_point)
print(type(floating_point))
```

```
11.0
<class 'float'>
```

b. Integer

An integer does not have a decimal point. If we are willing to use 45.99 as an integer, it would be stored as 45. Such as, in integer types 9999.999 would be stored as 9999 or 10.0000000001 would be stored as 10, for example. Note: You will often see the data type **Int64 in Python which stands for 64 bit integer. The 64 refers to the memory allocated to store data in each cell which effectively relates to how many digits it can store in each "cell". Allocating space ahead of time allows computers to optimize storage and processing efficiency. [Reference](https://datacarpentry.org/python-ecology-lesson/04-data-types-and-format/#numeric-data-types) [For more knowledge](https://jakevdp.github.io/PythonDataScienceHandbook/02.01-understanding-data-types.html)**

In [2]:
```
integer = 11
print(integer)
print(type(integer))
```

```
11
<class 'int'>
```

c. Boolean

In computer science, the Boolean (sometimes shortened to Bool) is a data type that has one of two possible values (usually denoted true and false) which is intended to represent the two truth values of logic and Boolean algebra. [Reference](https://en.wikipedia.org/wiki/Boolean_data_type)

In [3]:
```
print(2 > 1)
print(15 == 16)
print(100 < 19)
```

```
True
False
False
```

d. String (Text) Data Types

Text data type is known as Strings in Python, or Objects in Pandas. Strings can contain numbers and / or characters. For example, a string might be a word, a sentence, or several sentences. Note: Strings that contain numbers can not be used for mathematical operations!!! [Reference](https://datacarpentry.org/python-ecology-lesson/04-data-types-and-format/#numeric-data-types)

In [4]:
```
text_data_types = "1,2,"'DATA'""
print(text_data_types)
print(type(text_data_types))
```

```
1,2,DATA
<class 'str'>
```

2. Non - Primitive Data Structures

2.1 Built-In Data Structures

a. List

A list in Python is a collection of items which can contain elements of multiple data types, which may be either numeric, character logical values, etc. [Reference](https://www.geeksforgeeks.org/difference-between-list-and-array-in-python/) Lists have a number of important characteristics: 1. List items are enclosed in square brackets, like this [item1, item2, item3]. 2. Lists are ordered – i.e. the items in the list appear in a specific order. This enables us to use an index to access to any item. 3. Lists are mutable, which means you can add or remove items after a list's creation. 4. List

elements do not need to be unique. Item duplication is possible, as each element has its own distinct place and can be accessed separately through the index. 5. Elements can be of different data types: you can combine strings, integers, and objects in the same list. [Reference] (https://learnpython.com/blog/python-array-vs-list/)

In [5]:
```
List_example_1 = [1,2.72,3,'Zurich','Toronto','San Francisco',True]

print(List_example_1)
print(type(List_example_1))
```

```
[1, 2.72, 3, 'Zurich', 'Toronto', 'San Francisco', True]
<class 'list'>
```

b. Tuple

Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and unchangeable/immutable. A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.). Tuples are written with round brackets. [Reference](https://www.w3schools.com/python/python_tuples.asp) [Reference] (https://www.programiz.com/python-programming/tuple)

In [6]:
```
Tuple_example_1 = (1,2,3,'Toronto','Zurich','Paris')
print(Tuple_example_1)
print(type(Tuple_example_1))
```

```
(1, 2, 3, 'Toronto', 'Zurich', 'Paris')
<class 'tuple'>
```

c. Dictionary

Dictionaries are used to store data values in key:value pairs. A dictionary, which is ordered, changeable and do not allow duplicates, is a collection.[Reference](https://www.w3schools.com /python/python_dictionaries.asp) Creating a dictionary is as simple as placing items inside curly braces {} separated by commas. An item has a key and a corresponding value that is expressed as a pair (key: value).[Reference](https://www.programiz.com/python-programming/dictionary) Note: As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered. [Reference](https://www.w3schools.com/python/python_dictionaries.asp)

In [7]:
```
Dictionary_example = {'City':['Zurich','Toronto'],
                      'Country':['Swiss','Canada'],
                      'Continental':['Europa','North America']}
print(Dictionary_example)
print(type(Dictionary_example))
```

```
{'City': ['Zurich', 'Toronto'], 'Country': ['Swiss', 'Canada'], 'Continental': ['Europa', 'North A
merica']}
<class 'dict'>
```

d. Set

A set is an unordered collection of items. Every set element is unique (no duplicates) and must be

immutable (cannot be changed) and unindexed. However, a set itself is mutable. We can add or remove items from it. Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc. It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

In [8]:
```
Set_example = {1,2,3,'Basel',1.1}
print(Set_example)
print(type(Set_example))
```

```
{1, 2, 3, 1.1, 'Basel'}
<class 'set'>
```

The Summary of Data Structures' Features

| LIST | TUPLE | DICTIONARY | SET | | :- | -: | :- | -: | | Mutable | Immutable | Mutable | Immutable | | Ordered | Ordered | Ordered | Unordered | | Int., Boole, String, Float, etc. | Int., Boole, String, Float, etc. | Int., Boole, String, Float, etc. | Int., Boole, String, Float, etc. | |Square Brackets | Round Brackets | Curvy Braces | Curvy Braces | | List_1 = [1, 2.22, 'exp'] | Tuple_1 = (1, 2.22, 'exp') | Dictionary_1 = {'City':[1, 2.2, 'exp']} | Set_1 = {1, 2.2, 'exp'} |

B. Data Structures in Pandas

A data structure is a collection of data values and defines the relationship between the data, and the operations that can be performed on the data. There are three main data structures in pandas: - 1. Series - 1D - 2. DataFrame - 2D - 3. Panel - 3D [Reference](https://medium.com/data-science-365/pandas-for-data-science-part-1-89bc231b3478)

1. Series - 1D

One-dimensional ndarray with axis labels (including time series). [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.Series.html)

In [9]:
```
import pandas as pd
import numpy as np
```

In [10]:
```
series = {'a':[1,11], 'b':[2,22], 'c':[3,33]}

pandas_series = pd.Series(series)
print(pandas_series)

print(type(pandas_series))
```

```
a    [1, 11]
b    [2, 22]
c    [3, 33]
dtype: object
```

2. DataFrame - 2D

Two-dimensional, size-mutable, potentially heterogeneous tabular data. [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)

In [11]:
```
pandas_dataframe = pd.DataFrame(series)
pandas_dataframe
```

Out[11]:

	a	b	c
0	1	2	3
1	11	22	33

In [12]:
```
type(pandas_dataframe)
```

Out[12]:
```
pandas.core.frame.DataFrame
```

3. Panel - 3D

Represents wide format panel data, stored as 3-dimensional array. [Reference](https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.Panel.html) Panel is deprecated. Hence, the recommended way to represent these types of 3-dimensional data is to use multi-indexing in DataFrames instead of Panels. A multi-indexed DataFrame can be directly converted to a Panel via DataFrame.to_panel() method. [Reference](https://medium.com/data-science-365/pandas-for-data-science-part-1-89bc231b3478) [Reference](https://pandas.pydata.org/docs/reference/api/pandas.MultiIndex.html) [Reference](https://pandas.pydata.org/docs/user_guide/advanced.html) Note: The panel has been removed from Pandas module 0.25.0 onwards. [Reference](https://www.geeksforgeeks.org/python-pandas-panel-shape/)

From dict of DataFrame Objects

```
#creating an empty panel
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p
```

Its **output** is as follows –

```
Dimensions: 2 (items) x 4 (major_axis) x 3 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 2
```

For more knowledge

2 - Reading CSV Files

Import Pandas Library and label it as 'pd'

In [13]:
```
import pandas as pd
```

In this part, you need to download the dataset. Keep in mind where the file is on your computer because as we need to specify the location of the file in Jupyter notebook in order to load the data. [dataset](https://github.com/hasan-firat-data-and-business-analyst/Data-Science-Python/blob/main/auto.csv).

In [14]:
```
reading_csv_files_example = pd.read_csv(r'auto.csv')
```

'head()' shows the first five rows of the dataframe by default but you can specify the number of rows in the parenthesis.

In [15]:
```
reading_csv_files_example.head()
```

Out[15]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	sy
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	...	136	

5 rows × 26 columns

'tail()' shows the bottom five rows by default.

In [16]:
```
reading_csv_files_example.tail()
```

Out[16]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel syster
196	-1	95.0	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpf
197	-1	95.0	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpf
198	-1	95.0	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpf
199	-1	95.0	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	id
200	-1	95.0	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpf

5 rows × 26 columns

we are good to use in the function of 'head()' and 'tail()' in any numbers.

In [17]:
```
reading_csv_files_example.head(3)
```

Out[17]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	sy
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	

3 rows × 26 columns

In [18]:
```
reading_csv_files_example.tail(7)
```

Out[18]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fue syste
194	-2	103.0	volvo	gas	turbo	four	sedan	rwd	front	104.3	...	130	mp
195	-1	74.0	volvo	gas	turbo	four	wagon	rwd	front	104.3	...	130	mp
196	-1	95.0	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mp
197	-1	95.0	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mp
198	-1	95.0	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mp
199	-1	95.0	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	i
200	-1	95.0	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mp

7 rows × 26 columns

'shape' function tells us how many rows and columns exist in a dataframe.

In [19]:
```
reading_csv_files_example.shape
```

Out[19]:
```
(201, 26)
```

3 - Pandas DataFrame

3.1 - Data Structure Transformation in Pandas

3.1.1 - Dictionary to DataFrame

Checking the Pandas Version out

In [20]:
```
print(pd.__version__)
```

```
1.2.4
```

Creating dictionary for using in the example and checking the data structure's type of example out

In [21]:
```
dict_example = { 'Country': ["Canada", "Germany", "Japan"],
                 'Continent': ["North America", "Europe", "Asia"]}
```

In [22]:
```
print(dict_example)
print(type(dict_example))
```

```
{'Country': ['Canada', 'Germany', 'Japan'], 'Continent': ['North America', 'Europe', 'Asia']}
<class 'dict'>
```

Changing types of dictionary to Pandas DataFrame and checking out the last data structure

In [23]:
```
dictionary_to_df = pd.DataFrame(dict_example)
dictionary_to_df
```

Out[23]:

	Country	Continent
0	Canada	North America
1	Germany	Europe
2	Japan	Asia

In [24]:
```
type(dictionary_to_df)
```

Out[24]:
```
pandas.core.frame.DataFrame
```

3.1.2 - Dictionary to Pandas Series

Creating the new dictionary for new example

In [25]:
```
dict_example_for_Pandas_Series = {'Name': ["John","Sebastian","Rayn"],
                                  'Place': ["London", "Vienna","Oslo"]}

print(dict_example_for_Pandas_Series)
print(type(dict_example_for_Pandas_Series))
```

```
{'Name': ['John', 'Sebastian', 'Rayn'], 'Place': ['London', 'Vienna', 'Oslo']}
<class 'dict'>
```

Changing the dictionary data structure to Pandas Series

In [26]:
```
dictionary_to_series = pd.Series(dict_example_for_Pandas_Series)

print(dictionary_to_series)
print(type(dictionary_to_series))
```

```
Name     [John, Sebastian, Rayn]
Place     [London, Vienna, Oslo]
dtype: object
<class 'pandas.core.series.Series'>
```

3.1.3 - List to DataFrame

Creating the list with only number elements

In [27]:
```
list_example = [1,2,3,4,5,6,7,8,9]

print(list_example)
print(type(list_example))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
<class 'list'>
```

Transformation of the list to Pandas DataFrame

In [28]:
```
list_example_df = pd.DataFrame(list_example)
```

In [29]:
```
list_example_df
```

Out[29]:

	0
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9

In [30]:
```
type(list_example_df)
```

Out[30]:
```
pandas.core.frame.DataFrame
```

Creating the other list with both numbers and string elements

In [31]:
```
list_example2 = [1,2,3,4,"Zurich"]

list_example2
```

Out[31]:
```
[1, 2, 3, 4, 'Zurich']
```

The transformation of the new list to Pandas DataFrame

In [32]:
```
list_example_df2 = pd.DataFrame(list_example2)
list_example_df2
```

Out[32]:

	0
0	1

```
   0
1  2
2  3
3  4
```

In [33]:
```
type(list_example_df2)
```

Out[33]:
```
pandas.core.frame.DataFrame
```

3.1.4 - List to Pandas Series

Creating the new list with only numbers

In [34]:
```
list_example_for_series = [11,22,33,44,55,66,77,88,99]

print(list_example_for_series)
print(type(list_example_for_series))
```

```
[11, 22, 33, 44, 55, 66, 77, 88, 99]
<class 'list'>
```

The list to Pandas Series

In [35]:
```
list_to_series = pd.Series(list_example_for_series)

print(list_to_series)
print(type(list_to_series))
```

```
0    11
1    22
2    33
3    44
4    55
5    66
6    77
7    88
8    99
dtype: int64
<class 'pandas.core.series.Series'>
```

Creating the other list with numbers and strings

In [36]:
```
list_example_for_series2 = [11,22,33,"Asia","Africa"]

print(list_example_for_series2)
print(type(list_example_for_series2))
```

```
[11, 22, 33, 'Asia', 'Africa']
<class 'list'>
```

List to Pandas Series

In [37]:
```
list_to_series2 = pd.Series(list_example_for_series2)

print(list_to_series2)
print(type(list_to_series2))
```

```
0        11
1        22
2        33
3      Asia
4    Africa
dtype: object
<class 'pandas.core.series.Series'>
```

3.2 Creating the Pandas DataFrame

3.2.1 - Text Dataframe

In [38]:
```
dict_for_dataframe_text = {'Name':["Carlos","David","Emma"],
                           'City':["New York","Tokyo","Berlin"]}

pandas_dataframe_example_text = pd.DataFrame(dict_for_dataframe_text, index = ['User 1','User 2','

pandas_dataframe_example_text
```

Out[38]:

	Name	City
User 1	Carlos	New York
User 2	David	Tokyo
User 3	Emma	Berlin

3.2.2 - The First Way

In [39]:
```
pandas_dataframe_example = pd.DataFrame({'ID':  [100,101,102,103],
                                         'Math':[99,76,88,98],
                                         'Geo': [78,98,90,89],
                                         'Literature':[98,87,76,77]
                                        })

pandas_dataframe_example
```

Out[39]:

	ID	Math	Geo	Literature
0	100	99	78	98
1	101	76	98	87
2	102	88	90	76
3	103	98	89	77

3.2.3 - The Second Way

In [40]:
```
dict_for_dataframe = {'ID':[100,101,102,103],'Math':[99,76,88,98],'Geo': [78,98,90,89],'Literature

pandas_dataframe_example2 = pd.DataFrame(dict_for_dataframe)
pandas_dataframe_example2
```

Out[40]:

	ID	Math	Geo	Literature
0	100	99	78	98
1	101	76	98	87
2	102	88	90	76
3	103	98	89	77

3.2.4 - The Third Way

In [41]:
```
import numpy as np
pandas_dataframe_example3 = pd.DataFrame([[100,99,78,98],[101,76,98,87],[102,88,90,76],[103,98,89,
                                          columns =['ID','Math','Geo','Literature'],index = np.rand

pandas_dataframe_example3
```

Out[41]:

	ID	Math	Geo	Literature
0.333086	100	99	78	98
0.788754	101	76	98	87
0.432543	102	88	90	76
0.333622	103	98	89	77

3.2.5 - Using Orient Index and Columns

In [42]:
```
dict_for_dataframe2 = {'ID':[100,101,102,103],'Math':[99,76,88,98],'Geo': [78,98,90,89],'Literatur
pandas_dataframe_example4 = pd.DataFrame.from_dict(dict_for_dataframe2,orient = 'columns')
pandas_dataframe_example4
```

Out[42]:

	ID	Math	Geo	Literature
0	100	99	78	98
1	101	76	98	87
2	102	88	90	76
3	103	98	89	77

In [43]:
```
dict_for_dataframe2 = {'ID':[100,101,102,103],'Math':[99,76,88,98],'Geo': [78,98,90,89],'Literatur
pandas_dataframe_example4 = pd.DataFrame.from_dict(dict_for_dataframe2,orient = 'index')
pandas_dataframe_example4
```

Out[43]:

	0	1	2	3

	0	1	2	3
ID	100	101	102	103
Math	99	76	88	98
Geo	78	98	90	89

3.3 The Columns in Pandas

3.3.1 - Creating The DataFrame Example

In [44]:
```
dict_for_dataframe = {'ID':[100,101,102,103],'Math':[99,76,88,98],'Geo': [78,98,90,89],'Literature

dataframe_column_example = pd.DataFrame(dict_for_dataframe)
dataframe_column_example
```

Out[44]:

	ID	Math	Geo	Literature
0	100	99	78	98
1	101	76	98	87
2	102	88	90	76
3	103	98	89	77

3.3.2 - Renaming The Columns

In [45]:
```
renanaming_column_example = dataframe_column_example.rename(columns = {'Geo': 'Geography'}  )

renanaming_column_example
```

Out[45]:

	ID	Math	Geography	Literature
0	100	99	78	98
1	101	76	98	87
2	102	88	90	76
3	103	98	89	77

3.3.3 - Dropping The Columns

In [46]:
```
dropping_example = dataframe_column_example.drop(columns = 'Literature')

dropping_example
```

Out[46]:

	ID	Math	Geo
0	100	99	78

	ID	Math	Geo
1	101	76	98
2	102	88	90

3.3.4 - Adding The Columns

In [47]:
```
dataframe_column_example['Data Science'] = [99,96,94,100]

dataframe_column_example
```

Out[47]:

	ID	Math	Geo	Literature	Data Science
0	100	99	78	98	99
1	101	76	98	87	96
2	102	88	90	76	94
3	103	98	89	77	100

3.4 - Index in Pandas and Indexing

3.4.1 - Creating the New DataFrame

In [48]:
```
dataframe_index_example = pd.DataFrame({

    'Name':["Emma","Maddy","Cassy","Tony","Montana","Carl"],
    'Location':["Toronto","Toronto","Zurich","Zurich","New York","Zurich"],
    'Sex':["Female","Female","Female","Male","Male","Male"],
    'Education':['PhD','Bachelor','Master','PhD','Master','Postgrad'],
    'Marital Status':['Single','Married','Divorced','Divorced','Single','Married'],
    'Profession':['Teacher','Lawyer','Engineer','Medical Doctor','Nurse','Business Owner'],
    'Annual Revenue - USD':[100000, 456000,768000,560000,240000,7600000],
    'Children Status':["yes","yes","no","yes","yes","yes"],
    'The Number of Children':[1,1,2,4,2,3],
    'Far from home - km':[124,36,85,777,25,12]

                                        })

dataframe_index_example
```

Out[48]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124
1	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85
3	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km

New

3.4.2 - Set Index

To set a column as index for a DataFrame, use DataFrame.set_index() function, with the column name passed as argument. Pandas DataFrame - Set Column as Index You can also setup MultiIndex with multiple columns in the index. In this case, pass the array of column names required for index, to set_index() method. [Reference](https://pythonexamples.org/pandas-set-column-as-index/)

Set Index Example

In [49]:
```
dataframe_index_example2 = dataframe_index_example.set_index('The Number of Children' )
dataframe_index_example2
```

Out[49]:

The Number of Children	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	Far from home - km
1	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	124
1	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	36
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	85
4	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	777
2	Montana	New York	Male	Master	Single	Nurse	240000	yes	25
3	Carl	Zurich	Male	Postgrad	Married	Business Owner	7600000	yes	12

Set Index Example

In [50]:
```
dataframe_index_example3 = dataframe_index_example.set_index((dataframe_index_example['The Number
dataframe_index_example3
```

Out[50]:

The Number of Children	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km

The Number of Children	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
False	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124
False	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36
True	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85

Set Index Example

In [51]:
```
dataframe_index_example4 = dataframe_index_example.set_index((dataframe_index_example['Children St
dataframe_index_example4
```

Out[51]:

Children Status	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
True	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124
True	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36
False	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85
True	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777
True	Montana	New York	Male	Master	Single	Nurse	240000	yes	2	25
True	Carl	Zurich	Male	Postgrad	Married	Business Owner	7600000	yes	3	12

Set Index Example

In [52]:
```
dataframe_set_index_example5 = dataframe_index_example.set_index(['Education','Location'])
dataframe_set_index_example5
```

Out[52]:

Education	Location	Name	Sex	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
PhD	Toronto	Emma	Female	Single	Teacher	100000	yes	1	124
Bachelor	Toronto	Maddy	Female	Married	Lawyer	456000	yes	1	36
Master	Zurich	Cassy	Female	Divorced	Engineer	768000	no	2	85

		Name	Sex	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
Education	Location								
PhD	Zurich				Medical				

Set Index Example

In [53]:
```
dataframe_set_index_example6 = dataframe_index_example.set_index(
    ['Marital Status',
    dataframe_index_example['Annual Revenue - USD'] > 300000])

dataframe_set_index_example6
```

Out[53]:

		Name	Location	Sex	Education	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
Marital Status	Annual Revenue - USD									
Single	False	Emma	Toronto	Female	PhD	Teacher	100000	yes	1	124
Married	True	Maddy	Toronto	Female	Bachelor	Lawyer	456000	yes	1	36
Divorced	True	Cassy	Zurich	Female	Master	Engineer	768000	no	2	85
	True	Tony	Zurich	Male	PhD	Medical Doctor	560000	yes	4	777
Single	False	Montana	New York	Male	Master	Nurse	240000	yes	2	25
Married	True	Carl	Zurich	Male	Postgrad	Business Owner	7600000	yes	3	12

Set Index Example

In [54]:
```
dataframe_set_index_example7 = dataframe_index_example.set_index(
    ['Marital Status',
    dataframe_index_example['Annual Revenue - USD'] > 300000,'Location'])

dataframe_set_index_example7
```

Out[54]:

			Name	Sex	Education	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
Marital Status	Annual Revenue - USD	Location								
Single	False	Toronto	Emma	Female	PhD	Teacher	100000	yes	1	124
Married	True	Toronto	Maddy	Female	Bachelor	Lawyer	456000	yes	1	36
Divorced	True	Zurich	Cassy	Female	Master	Engineer	768000	no	2	85

	Name	Sex	Education	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km

Marital Status	Annual Revenue - USD	Location

In [55]:
```
dataframe_set_index_example8 = dataframe_index_example.set_index(
    ['Marital Status',
    dataframe_index_example['Annual Revenue - USD'] > 300000,
     'Location',
    dataframe_index_example['Far from home - km'] > 75
    ])

dataframe_set_index_example8
```

Out[55]:

				Name	Sex	Education	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
Marital Status	Annual Revenue - USD	Location	Far from home - km								
Single	False	Toronto	True	Emma	Female	PhD	Teacher	100000	yes	1	124
Married	True	Toronto	False	Maddy	Female	Bachelor	Lawyer	456000	yes	1	36
Divorced	True	Zurich	True	Cassy	Female	Master	Engineer	768000	no	2	85
			True	Tony	Male	PhD	Medical Doctor	560000	yes	4	777
Single	False	New York	False	Montana	Male	Master	Nurse	240000	yes	2	25
Married	True	Zurich	False	Carl	Male	Postgrad	Business Owner	7600000	yes	3	12

Set Index Example

NOTE: If we do not use ' [ ] ' (square brackets) with multiple column selection, as you are able to see on the below example, we can get only the first columns name, which is written in the set_index function.

In [56]:
```
dataframe_set_index_example9 = dataframe_index_example.set_index('Sex','Location')
dataframe_set_index_example9
```

Out[56]:

	Name	Location	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
Sex									
Female	Emma	Toronto	PhD	Single	Teacher	100000	yes	1	124

Sex	Name	Location	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
Female	Maddy	Toronto	Bachelor	Married	Lawyer	456000	yes	1	36
Female	Cassy	Zurich	Master	Divorced	Engineer	768000	no	2	85
Male	Tony	Zurich	PhD	Divorced	Medical Doctor	560000	yes	4	777

3.4.3 - Dropping the Rows

The drop() function is used to drop specified labels from rows or columns. Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level. [Reference](https://www.w3resource.com/pandas/dataframe/dataframe-drop.php)
The drop() method removes the specified row or column. By specifying the column axis (axis='columns'), the drop() method removes the specified column. By specifying the row axis (axis='index'), the drop() method removes the specified row. [Reference](https://www.w3schools.com/python/pandas/ref_df_drop.asp)

In [57]:
```
dataframe_index_example
```

Out[57]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124
1	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85
3	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777
4	Montana	New York	Male	Master	Single	Nurse	240000	yes	2	25
5	Carl	Zurich	Male	Postgrad	Married	Business Owner	7600000	yes	3	12

Dropping the row

In [58]:
```
dataframe_index_dropping_example6 = dataframe_index_example.drop(0)
dataframe_index_dropping_example6
```

Out[58]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
1	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85
3	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777
4	Montana	New	Male	Master	Single	Nurse	240000	yes	2	25

Above example, we has dropped the first row, the number of index is 0 (zero). Please, check " dataframe_index_example " example out. NOTE: This method is used for only " ONE ROW ". If you want to add more row or index number for dropping, it needs to be written in " [ ] " square brackets.

In [59]:
```
dataframe_index_dropping_example6_more_rows = dataframe_index_example.drop([0,2,3])
dataframe_index_dropping_example6_more_rows
```

Out[59]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
1	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36
4	Montana	New York	Male	Master	Single	Nurse	240000	yes	2	25
5	Carl	Zurich	Male	Postgrad	Married	Business Owner	7600000	yes	3	12

NOTE: This method is used for only " ONE ROW ". If you want to add more row or index number, it needs to be written in " [] " square brackets.

Dropping the row

In [60]:
```
dataframe_index_dropping_example7 = dataframe_index_example.drop(5, axis='index')
dataframe_index_dropping_example7
```

Out[60]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124
1	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85
3	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777
4	Montana	New York	Male	Master	Single	Nurse	240000	yes	2	25

In above example, we has dropped the fifth row by using the other way of dropping function, the number of index is 5 (five). Please, check " dataframe_index_example " example out.

In [61]:
```
dataframe_index_dropping_example8 = dataframe_index_example.drop(index=([1,4,5]))
dataframe_index_dropping_example8
```

Out[61]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85
3	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777

In the last example, we has dropped the first, fourth and fifth rows by using the dropping function with index, the number of index is 1,4, and 5. Please, check " dataframe_index_example " example out.

Dropping the column

In [62]:
```
dataframe_index_dropping_example9 = dataframe_index_example.drop("Profession" , axis='columns')
dataframe_index_dropping_example9
```

Out[62]:

	Name	Location	Sex	Education	Marital Status	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	Female	PhD	Single	100000	yes	1	124
1	Maddy	Toronto	Female	Bachelor	Married	456000	yes	1	36
2	Cassy	Zurich	Female	Master	Divorced	768000	no	2	85
3	Tony	Zurich	Male	PhD	Divorced	560000	yes	4	777
4	Montana	New York	Male	Master	Single	240000	yes	2	25
5	Carl	Zurich	Male	Postgrad	Married	7600000	yes	3	12

In the first dropping column example, we used " axis = 'columns' " in " drop() method " to remove the specified column.

In [63]:
```
dataframe_index_dropping_example10 = dataframe_index_example.drop(["Children Status", "Education"]
dataframe_index_dropping_example10
```

Out[63]:

	Name	Location	Sex	Marital Status	Profession	Annual Revenue - USD	The Number of Children	Far from home - km
0	Emma	Toronto	Female	Single	Teacher	100000	1	124
1	Maddy	Toronto	Female	Married	Lawyer	456000	1	36

	Name	Location	Sex	Marital Status	Profession	Annual Revenue - USD	The Number of Children	Far from home - km
2	Cassy	Zurich	Female	Divorced	Engineer	768000	2	85
3	Tony	Zurich	Male	Divorced	Medical Doctor	560000	4	777
4	Montana	New York	Male	Single	Nurse	240000	2	25

In the second example to drop the column, we used " drop() method " to remove the "Children Status", and "Education" columns.

In [64]:
```
dataframe_index_dropping_example11 = dataframe_index_example.drop(columns = ["Sex","Location"])
dataframe_index_dropping_example11
```

Out[64]:

	Name	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	PhD	Single	Teacher	100000	yes	1	124
1	Maddy	Bachelor	Married	Lawyer	456000	yes	1	36
2	Cassy	Master	Divorced	Engineer	768000	no	2	85
3	Tony	PhD	Divorced	Medical Doctor	560000	yes	4	777
4	Montana	Master	Single	Nurse	240000	yes	2	25
5	Carl	Postgrad	Married	Business Owner	7600000	yes	3	12

In the last example, we used " columns = ["_","_"] " in " drop() method " to remove the "Sex", and "Location" columns.

In [65]:
```
dataframe_index_dropping_example11 = dataframe_index_example.drop(1, axis='index')
dataframe_index_dropping_example11
```

Out[65]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85
3	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777
4	Montana	New York	Male	Master	Single	Nurse	240000	yes	2	25
5	Carl	Zurich	Male	Postgrad	Married	Business Owner	7600000	yes	3	12

Dropping the row and column at the same time

In [66]:
```
dataframe_index_dropping_example12 = dataframe_index_example.drop(
    index = [1,3,5],
    columns=["Sex","Marital Status", "Annual Revenue - USD"])

dataframe_index_dropping_example12
```

Out[66]:

	Name	Location	Education	Profession	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	PhD	Teacher	yes	1	124
2	Cassy	Zurich	Master	Engineer	no	2	85
4	Montana	New York	Master	Nurse	yes	2	25

In this example, we have learned how to drop both the rows and columns at the same time in the same code block. This examples can be extended with the other kind of dropping syntax.

3.4.4 - Adding the Rows

Adding the Rows

In [67]:
```
df_for_adding_example = pd.DataFrame({
    'Name':["Albert"],
    'Location':["London"],
    'Sex':["Male"],
    'Education':["Bachelor"],
    'Marital Status':["Single"],
    'Profession':["Lawyer"],
    'Annual Revenue - USD': [158000],
    'Children Status': ["no"],
    'The Number of Children': [0]
})

df_for_adding_example
```

Out[67]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children
0	Albert	London	Male	Bachelor	Single	Lawyer	158000	no	0

In [68]:
```
dataframe_index_adding_example = dataframe_index_example.append(df_for_adding_example,ignore_index
dataframe_index_adding_example
```

Out[68]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124.0
1	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36.0

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85.0
3	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777.0
4	Montana	New York	Male	Master	Single	Nurse	240000	yes	2	25.0

In the above example, if we use " ignore_index " as " False ", the new row, that has added, is shown as " zero (0) " on the last row in the DataFrame.

In [69]:
```
dataframe_index_adding_example2 = dataframe_index_example.append(df_for_adding_example,ignore_inde
dataframe_index_adding_example2
```

Out[69]:

	Name	Location	Sex	Education	Marital Status	Profession	Annual Revenue - USD	Children Status	The Number of Children	Far from home - km
0	Emma	Toronto	Female	PhD	Single	Teacher	100000	yes	1	124.0
1	Maddy	Toronto	Female	Bachelor	Married	Lawyer	456000	yes	1	36.0
2	Cassy	Zurich	Female	Master	Divorced	Engineer	768000	no	2	85.0
3	Tony	Zurich	Male	PhD	Divorced	Medical Doctor	560000	yes	4	777.0
4	Montana	New York	Male	Master	Single	Nurse	240000	yes	2	25.0
5	Carl	Zurich	Male	Postgrad	Married	Business Owner	7600000	yes	3	12.0
6	Albert	London	Male	Bachelor	Single	Lawyer	158000	no	0	NaN

In the above example, if we use " ignore_index " as " True ", the new row, that has added, is shown as " 6 " on the last row in the DataFrame.

Renaming the Index

In [70]:
```
giving_a_name_for_index_example = pd.DataFrame({'Name':['Carlos','David','Emma'],
                                                'City':['New York','London','Berlin'],
                                                'Sex':['Male','Male','Female']},
                                               index = ['Status 1','Status 2', 'Status3']
                                               )

giving_a_name_for_index_example
```

Out[70]:

	Name	City	Sex
Status 1	Carlos	New York	Male
Status 2	David	London	Male

	Name	City	Sex

The Second Example for Renaming the Index

In [71]:
```
giving_a_name_for_index_example2 = pd.DataFrame({'Name':['Carlos','David','Emma'],
                                                 'City':['New York','London','Berlin'],
                                                 'Sex':['Male','Male','Female']},
                                                index = ['The first row','the 2nd Row',3]
                                                )

giving_a_name_for_index_example2
```

Out[71]:

	Name	City	Sex
The first row	Carlos	New York	Male
the 2nd Row	David	London	Male
3	Emma	Berlin	Female

The Third Example for Renaming the Index

In [72]:
```
example_label = [99,88,77,66,55,44,33,22,11]
example_label
```

Out[72]:
```
[99, 88, 77, 66, 55, 44, 33, 22, 11]
```

In [73]:
```
type(example_label)
```

Out[73]:
```
list
```

In [74]:
```
labels = pd.Series(example_label, index = ["a","b","c",4,5,6,"x","y","z"])
labels
```

Out[74]:
```
a    99
b    88
c    77
4    66
5    55
6    44
x    33
y    22
z    11
dtype: int64
```

In [75]:
```
type(labels)
```

Out[75]:
```
pandas.core.series.Series
```

In [76]:
```
labels2 = pd.DataFrame(example_label, index = ["a","b","c",4,5,6,"x","y","z"])
labels2
```

Out[76]:

	0

	0
a	99
b	88
c	77
4	66
5	55
6	44
x	33
y	22

In [77]:
```
type(labels2)
```

Out[77]:
```
pandas.core.frame.DataFrame
```

Creating the example for reset index function

In [78]:
```
reset_index_example = pd.DataFrame({'Name':['Carlos','David','Emma'],
                                    'City':['New York','London','Berlin'],
                                    'Sex':['Male','Male','Female']},
                                   index = ['The first row','the 2nd Row',3]
                                   )

reset_index_example
```

Out[78]:

	Name	City	Sex
The first row	Carlos	New York	Male
the 2nd Row	David	London	Male
3	Emma	Berlin	Female

Using the reset index function

In [79]:
```
reset_index_example.reset_index(drop=True,inplace=True)

reset_index_example
```

Out[79]:

	Name	City	Sex
0	Carlos	New York	Male
1	David	London	Male
2	Emma	Berlin	Female

3.5 - Selecting and Filtering Columns and Rows in a Data Frame

3.5.1 - Index-Based Selection with iloc

ILOC : Purely integer-location based indexing for selection by position. .iloc[] is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array. [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html)

Using the existence dataframe

In [80]:
```
reading_csv_files_example.head()
```

Out[80]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	sy
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	...	136	

5 rows × 26 columns

The Row Selection In this below example, we are going to select the second and the third row.

In [81]:
```
reading_csv_files_example.iloc[2:4]
```

Out[81]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fu
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	m
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	m

2 rows × 26 columns

The Column Selection In this below example, we are going to select the columns, that are in between second and the sixth columns with all rows. NOTE: Be careful, " num-of-doors " is the 5th column, not the 6th. When we apply this code block as you are going to see that, the column selection will stop the fifth column, even if we have written " 6 (six) " in the code block. It is a Pandas feature.

In [82]:
```
reading_csv_files_example.iloc[:,2:6]
```

Out[82]:

	make	fuel-type	aspiration	num-of-doors
0	alfa-romero	gas	std	two
1	alfa-romero	gas	std	two
2	alfa-romero	gas	std	two
3	audi	gas	std	four
4	audi	gas	std	four
...	...	...	...	...
196	volvo	gas	std	four
197	volvo	gas	turbo	four
198	volvo	gas	std	four
199	volvo	diesel	turbo	four
200	volvo	gas	turbo	four

201 rows × 4 columns

The Column and The Row Selection In this below example, we are going to select the rows, that are in between the sixth and the eleventh rows, and the columns, like " wheel-base, lenght, and width". NOTE: Be careful, the last column " width " is the 11th column, not the 12th. More information about it has been given in the previous example.

In [83]:
```
reading_csv_files_example.iloc[6:12,9:12]
```

Out[83]:

	wheel-base	length	width
6	105.8	192.7	71.4
7	105.8	192.7	71.4
8	105.8	192.7	71.4
9	101.2	176.8	64.8
10	101.2	176.8	64.8
11	101.2	176.8	64.8

Reverse Row Selection with iloc

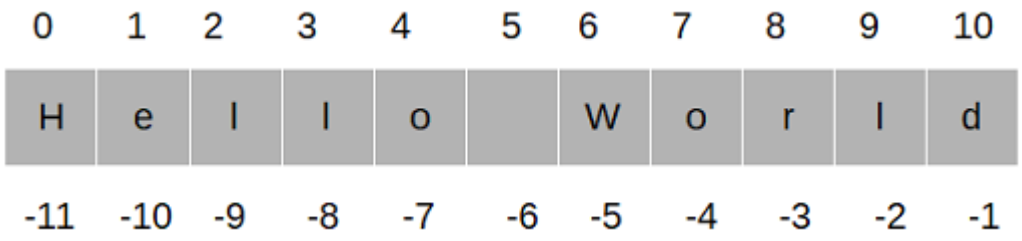
In [84]:
```
reading_csv_files_example.iloc[-7:]
```

Out[84]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fue syste
194	-2	103.0	volvo	gas	turbo	four	sedan	rwd	front	104.3	...	130	mp

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fue syste
195	-1	74.0	volvo	gas	turbo	four	wagon	rwd	front	104.3	...	130	mp
196	-1	95.0	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mp
197	-1	95.0	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mp
198	-1	95.0	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mp
199	-1	95.0	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	i

When it comes to selection of the column reversely, you do not need to start counting at the zero, you good to start at one (1). In above example, the seventh rows is 194 and do not forget to use " - (minus) ". On the other hand, when we apply " reverse selection ", as you can see that numbers start at minus one, but when it comes to, normal selection, it starts at zero.



Reverse Column Selection with iloc

In [85]:
```
reading_csv_files_example.iloc[77:81,-3:]
```

Out[85]:

	city-mpg	highway-mpg	price
77	23	30	9959
78	25	32	8499
79	19	24	12629
80	19	24	14869

In this above example, we have selected the rows, being between 77th and 81st rows, and the columns starting at the last one to the third one. The minus in this code block provides the REVERSE selection. That's why the DataFrame is being shown as reversely.

The Second Example for Reverse Column Selection with iloc

In [86]:
```
reading_csv_files_example.iloc[-2:,-2:]
```

Out[86]:

	highway-mpg	price
199	27	22470
200	25	22625

In this second example, we have selected the last two columns and rows in the DataFrame. If I need to draw an analogy, we selected the right bottom edge of the DataFrame.

The Third Example with iloc

In [87]:
```
reading_csv_files_example.iloc[-44:-40,-5:-2]
```

Out[87]:

	horsepower	peak-rpm	city-mpg
157	70.0	4800.0	28
158	70.0	4800.0	28
159	70.0	4800.0	29
160	70.0	4800.0	29

In this third example, we have started to select at 44th, the name of rows is 160, and it has stopped at 40, namely the name of row is 157. When it comes to columns, it has started at 5, that is " city-mpg " and it has stopped at 5 or " horsepower ". Do not forget that when you apply the reverse selection, you need to think the process reversely.

3.5.2 - Label-based Selection with loc function

** **

Access a group of rows and columns by label(s) or a boolean array. .loc[] is primarily label based, but may also be used with a boolean array. [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html)

In [88]:
```
loc_function_example = pd.DataFrame({
    'Name':['Carlos','David','Emma','Chaplin','Gogol','Gabriel'],
     'City':['New York','London','Berlin','Tokyo','Moscow','Aracataca'],
      'Sex':['Male','Male','Female','Male','Male','Male']},
       index = ['The first row','the 2nd Row',3,'okay','The Overcoat','One Hundred Years of solitu
                                              )

loc_function_example
```

Out[88]:

	Name	City	Sex
The first row	Carlos	New York	Male
the 2nd Row	David	London	Male
3	Emma	Berlin	Female
okay	Chaplin	Tokyo	Male
The Overcoat	Gogol	Moscow	Male
One Hundred Years of solitude	Gabriel	Aracataca	Male

The Row Selection with loc

In [89]:
```
loc_function_example.loc[[3,'okay','The Overcoat']]
```

Out[89]:

	Name	City	Sex
3	Emma	Berlin	Female
okay	Chaplin	Tokyo	Male
The Overcoat	Gogol	Moscow	Male

In this above example, when we apply the loc function in Pandas, we need to write the exact column name inside quotes down, othervise we fail to get the DataFrame. On the other hand, as you can see that, we are able to get the column name in order to in code block.

The Column Selection

In [90]:
```
loc_function_example.loc[:,['Name']]
```

Out[90]:

	Name
The first row	Carlos
the 2nd Row	David
3	Emma
okay	Chaplin
The Overcoat	Gogol
One Hundred Years of solitude	Gabriel

In [91]:
```
loc_function_example.loc[:,['Name','Sex']]
```

Out[91]:

	Name	Sex
The first row	Carlos	Male
the 2nd Row	David	Male
3	Emma	Female
okay	Chaplin	Male
The Overcoat	Gogol	Male
One Hundred Years of solitude	Gabriel	Male

The Row and Column Selection

In [92]:
```
loc_function_example.loc[['One Hundred Years of solitude'],['Name','City']]
```

Out[92]:

	Name	City
One Hundred Years of solitude	Gabriel	Aracataca

Conditional Selection

In [93]:
```
reading_csv_files_example.head()
```

Out[93]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	sy
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	...	136	

5 rows × 26 columns

The First Example

In [94]:
```
reading_csv_files_example[reading_csv_files_example['horsepower'] == 111]
```

Out[94]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
141	0	102.0	subaru	gas	turbo	four	sedan	4wd	front	97.0	...	108
145	0	85.0	subaru	gas	turbo	four	wagon	4wd	front	96.9	...	108

4 rows × 26 columns

The Second Example

In [95]:
```
reading_csv_files_example[

        (reading_csv_files_example.bore == 3.47) &
        (reading_csv_files_example.price == 16500)

    ]
```

Out[95]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	sy

						doors						
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130

1 rows × 26 columns

Multiple Conditional Selection

In [96]:
```
reading_csv_files_example[
    (reading_csv_files_example['normalized-losses'] > 170) &
    ((reading_csv_files_example['wheel-base'] > 90) & (reading_csv_files_example['wheel-base'] < 1
    ((reading_csv_files_example.stroke > 3.11) | (reading_csv_files_example.stroke<= 3.40)))
]
```

Out[96]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engin siz
101	3	194.0	nissan	gas	std	two	hatchback	rwd	front	91.3	...	18
102	3	194.0	nissan	gas	turbo	two	hatchback	rwd	front	91.3	...	18
103	1	231.0	nissan	gas	std	two	hatchback	rwd	front	99.2	...	18
122	3	186.0	porsche	gas	std	two	hatchback	rwd	front	94.5	...	15
186	3	256.0	volkswagen	gas	std	two	hatchback	fwd	front	94.5	...	10

5 rows × 26 columns

Multiple Conditional Selection - 2

In [97]:
```
reading_csv_files_example[
    ((reading_csv_files_example['aspiration'] == 'std') & (reading_csv_files_example['body-style']
    ((reading_csv_files_example['drive-wheels'] == 'rwd') & (reading_csv_files_example['fuel-syste
```

Out[97]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engi s
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	1
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	1
69	3	142.0	mercedes-benz	gas	std	two	convertible	rwd	front	96.6	...	2
121	3	122.0	plymouth	gas	turbo	two	hatchback	rwd	front	95.9	...	1
125	3	122.0	porsche	gas	std	two	convertible	rwd	rear	89.5	...	1
168	2	134.0	toyota	gas	std	two	convertible	rwd	front	98.4	...	1
185	3	122.0	volkswagen	gas	std	two	convertible	fwd	front	94.5	...	1

7 rows × 26 columns

isin function

isin is used to filter data frames. isin() method helps in selecting rows with having a particular(or Multiple) value in a particular column. [Reference](https://www.geeksforgeeks.org/python-pandas-dataframe-isin/)

In [98]:
```
isin_function_example = pd.DataFrame({'Legs': [2, 4,2], 'Wings': [2, 0,2]},
                  index=['Eagle', 'Dog','Ostrich'])
isin_function_example
```

Out[98]:

	Legs	Wings
Eagle	2	2
Dog	4	0
Ostrich	2	2

In [99]:
```
isin_function_example.isin([1])
```

Out[99]:

	Legs	Wings
Eagle	False	False
Dog	False	False
Ostrich	False	False

In [100…
```
isin_function_example.isin([1,2,3])
```

Out[100…

	Legs	Wings
Eagle	True	True
Dog	False	False
Ostrich	True	True

In [101…
```
isin_function_example.isin({'Wings':[0,3,5]})
```

Out[101…

	Legs	Wings
Eagle	False	False
Dog	False	True
Ostrich	False	False

In [102…
```
isin_function_example.isin({'Wings':[1],'Legs':[4,2]})
```

Out[102…

	Legs	Wings
Eagle	True	False

	Legs	Wings
Dog	True	False

In above five examples, we have applied many kind of different legs and wings' numbers for filtering in the DataFrame. If the number, has been written down by ourself is correct, we can see that, it is being written as " True ". If not, we are able to get " False " as output.

In [103…

```
isin_function_example2 = reading_csv_files_example["make"].isin(['isuzu'])

reading_csv_files_example[isin_function_example2]
```

Out[103…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fu syste
42	0	122.0	isuzu	gas	std	four	sedan	rwd	front	94.3	...	111	2b
43	2	122.0	isuzu	gas	std	two	hatchback	rwd	front	96.0	...	119	sp

2 rows × 26 columns

In above example, we have selected " isuzu ", which is being in the row of " make ".

In [104…

```
isin_function_example3 = reading_csv_files_example["make"].isin(['renault','saab'])

reading_csv_files_example[isin_function_example3]
```

Out[104…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	s
126	0	122.0	renault	gas	std	four	wagon	fwd	front	96.1	...	132	
127	2	122.0	renault	gas	std	two	hatchback	fwd	front	96.1	...	132	
128	3	150.0	saab	gas	std	two	hatchback	fwd	front	99.1	...	121	
129	2	104.0	saab	gas	std	four	sedan	fwd	front	99.1	...	121	
130	3	150.0	saab	gas	std	two	hatchback	fwd	front	99.1	...	121	
131	2	104.0	saab	gas	std	four	sedan	fwd	front	99.1	...	121	
132	3	150.0	saab	gas	turbo	two	hatchback	fwd	front	99.1	...	121	
133	2	104.0	saab	gas	turbo	four	sedan	fwd	front	99.1	...	121	

8 rows × 26 columns

In above example, we have selected " renault " and " saab " , which is being in the row of " make ".

Str Accessor

Pandas is a highly efficient library on textual data as well. The functions and methods under the

str accessor provide flexible ways to filter rows based on strings. For instance, we can select the names that starts with the words "do" in below example.

In [105…
```
reading_csv_files_example[reading_csv_files_example['make'].str.startswith('do')]
```

Out[105…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engine-size	f sys
20	1	118.0	dodge	gas	std	two	hatchback	fwd	front	93.7	…	90	2
21	1	118.0	dodge	gas	std	two	hatchback	fwd	front	93.7	…	90	2
22	1	118.0	dodge	gas	turbo	two	hatchback	fwd	front	93.7	…	98	n
23	1	148.0	dodge	gas	std	four	hatchback	fwd	front	93.7	…	90	2
24	1	148.0	dodge	gas	std	four	sedan	fwd	front	93.7	…	90	2
25	1	148.0	dodge	gas	std	four	sedan	fwd	front	93.7	…	90	2
26	1	148.0	dodge	gas	turbo	four	sedan	fwd	front	93.7	…	98	n
27	-1	110.0	dodge	gas	std	four	wagon	fwd	front	103.3	…	122	2
28	3	145.0	dodge	gas	turbo	two	hatchback	fwd	front	95.9	…	156	

9 rows × 26 columns

In above example, we have filtered the cars starting with two letters, like " do ", in make column.

In [106…
```
reading_csv_files_example[reading_csv_files_example['make'].str.endswith('ge')]
```

Out[106…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engine-size	f sys
20	1	118.0	dodge	gas	std	two	hatchback	fwd	front	93.7	…	90	2
21	1	118.0	dodge	gas	std	two	hatchback	fwd	front	93.7	…	90	2
22	1	118.0	dodge	gas	turbo	two	hatchback	fwd	front	93.7	…	98	n
23	1	148.0	dodge	gas	std	four	hatchback	fwd	front	93.7	…	90	2
24	1	148.0	dodge	gas	std	four	sedan	fwd	front	93.7	…	90	2
25	1	148.0	dodge	gas	std	four	sedan	fwd	front	93.7	…	90	2
26	1	148.0	dodge	gas	turbo	four	sedan	fwd	front	93.7	…	98	n
27	-1	110.0	dodge	gas	std	four	wagon	fwd	front	103.3	…	122	2
28	3	145.0	dodge	gas	turbo	two	hatchback	fwd	front	95.9	…	156	

9 rows × 26 columns

In above example, we have selected the cars, ending with " ge " in make column by using " endswith() ".

In [107…

```
reading_csv_files_example[reading_csv_files_example['body-style'].str.startswith('c')]
```

Out[107…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engi s
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	…	1
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	…	1
69	3	142.0	mercedes-benz	gas	std	two	convertible	rwd	front	96.6	…	2
125	3	122.0	porsche	gas	std	two	convertible	rwd	rear	89.5	…	1
168	2	134.0	toyota	gas	std	two	convertible	rwd	front	98.4	…	1
185	3	122.0	volkswagen	gas	std	two	convertible	fwd	front	94.5	…	1

6 rows × 26 columns

In [108…

```
reading_csv_files_example[reading_csv_files_example['make'].str.endswith('u')]
```

Out[108…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engine-size	sy
42	0	122.0	isuzu	gas	std	four	sedan	rwd	front	94.3	…	111	
43	2	122.0	isuzu	gas	std	two	hatchback	rwd	front	96.0	…	119	
134	2	83.0	subaru	gas	std	two	hatchback	fwd	front	93.7	…	97	
135	2	83.0	subaru	gas	std	two	hatchback	fwd	front	93.7	…	108	
136	2	83.0	subaru	gas	std	two	hatchback	4wd	front	93.3	…	108	
137	0	102.0	subaru	gas	std	four	sedan	fwd	front	97.2	…	108	
138	0	102.0	subaru	gas	std	four	sedan	fwd	front	97.2	…	108	
139	0	102.0	subaru	gas	std	four	sedan	fwd	front	97.2	…	108	
140	0	102.0	subaru	gas	std	four	sedan	4wd	front	97.0	…	108	
141	0	102.0	subaru	gas	turbo	four	sedan	4wd	front	97.0	…	108	
142	0	89.0	subaru	gas	std	four	wagon	fwd	front	97.0	…	108	
143	0	89.0	subaru	gas	std	four	wagon	fwd	front	97.0	…	108	
144	0	85.0	subaru	gas	std	four	wagon	4wd	front	96.9	…	108	
145	0	85.0	subaru	gas	turbo	four	wagon	4wd	front	96.9	…	108	

14 rows × 26 columns

In [208…

```
reading_csv_files_example[reading_csv_files_example['engine-location'].str.startswith('r')]
```

Out[208…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
123	3	122.0	porsche	gas	std	two	hardtop	rwd	rear	89.5	...	194
124	3	122.0	porsche	gas	std	two	hardtop	rwd	rear	89.5	...	194
125	3	122.0	porsche	gas	std	two	convertible	rwd	rear	89.5	...	194

In [221…

```
reading_csv_files_example[reading_csv_files_example['body-style'].str.startswith('c')]
```

Out[221…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engi s
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	1
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	1
69	3	142.0	mercedes-benz	gas	std	two	convertible	rwd	front	96.6	...	2
125	3	122.0	porsche	gas	std	two	convertible	rwd	rear	89.5	...	1
168	2	134.0	toyota	gas	std	two	convertible	rwd	front	98.4	...	1
185	3	122.0	volkswagen	gas	std	two	convertible	fwd	front	94.5	...	1

6 rows × 26 columns

Query

The query() method allows you to query the DataFrame and takes a query expression as a string parameter, which has to evaluate to either True of False. [Reference] (https://www.w3schools.com/python/pandas/ref_df_query.asp) The query function offers a little more flexibility at writing the conditions for filtering. We can pass the conditions as a string.

The below example is a basic example for query method.

In [111…

```
reading_csv_files_example.query('horsepower > 200')
```

Out[111…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
46	0	122.0	jaguar	gas	std	two	sedan	rwd	front	102.0	...	326
123	3	122.0	porsche	gas	std	two	hardtop	rwd	rear	89.5	...	194
124	3	122.0	porsche	gas	std	two	hardtop	rwd	rear	89.5	...	194
125	3	122.0	porsche	gas	std	two	convertible	rwd	rear	89.5	...	194

4 rows × 26 columns

In the second example, we are going to learn how to apply " Multiple Condtion Filtering " .

In [112…
```
reading_csv_files_example.query('stroke > 3.5 and price < 15000 and aspiration == "turbo" ')
```

Out[112…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engine siz
28	3	145.0	dodge	gas	turbo	two	hatchback	fwd	front	95.9	…	15
79	3	122.0	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	…	15
80	3	122.0	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	…	15
81	3	122.0	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	…	15
105	0	161.0	peugot	diesel	turbo	four	sedan	rwd	front	107.9	…	15
107	0	122.0	peugot	diesel	turbo	four	wagon	rwd	front	114.2	…	15
121	3	122.0	plymouth	gas	turbo	two	hatchback	rwd	front	95.9	…	15

7 rows × 26 columns

In the third example, we are going to combine the filtering with using query method and filtering without using query method.

Combining the filtering with using query method

In [113…
```
reading_csv_files_example.query('bore == stroke')
```

Out[113…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engine-size	sy
56	1	129.0	mazda	gas	std	two	hatchback	fwd	front	98.8	…	122	
57	0	115.0	mazda	gas	std	four	sedan	fwd	front	98.8	…	122	
58	1	129.0	mazda	gas	std	two	hatchback	fwd	front	98.8	…	122	
59	0	115.0	mazda	gas	std	four	sedan	fwd	front	98.8	…	122	
60	0	122.0	mazda	diesel	std	four	sedan	fwd	front	98.8	…	122	
61	0	115.0	mazda	gas	std	four	hatchback	fwd	front	98.8	…	122	

6 rows × 26 columns

Combining the filtering without using query method

In [114…
```
reading_csv_files_example[reading_csv_files_example['bore'] == reading_csv_files_example['stroke']]
```

Out[114…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engine-size	sy

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	sy
56	1	129.0	mazda	gas	std	two	hatchback	fwd	front	98.8	...	122	
57	0	115.0	mazda	gas	std	four	sedan	fwd	front	98.8	...	122	
58	1	129.0	mazda	gas	std	two	hatchback	fwd	front	98.8	...	122	
59	0	115.0	mazda	gas	std	four	sedan	fwd	front	98.8	...	122	
60	0	122.0	mazda	diesel	std	four	sedan	fwd	front	98.8	...	122	

In the above last two examples, as we can see that, we good to filter DataFrame with both using or not using query method.

In [115...

```
reading_csv_files_example[
    (reading_csv_files_example['stroke'] > 3.5) &
    (reading_csv_files_example['price'] < 15000)&
    (reading_csv_files_example['aspiration'] == 'turbo')]
```

Out[115...

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-siz
28	3	145.0	dodge	gas	turbo	two	hatchback	fwd	front	95.9	...	15
79	3	122.0	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	...	15
80	3	122.0	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	...	15
81	3	122.0	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	...	15
105	0	161.0	peugot	diesel	turbo	four	sedan	rwd	front	107.9	...	15
107	0	122.0	peugot	diesel	turbo	four	wagon	rwd	front	114.2	...	15
121	3	122.0	plymouth	gas	turbo	two	hatchback	rwd	front	95.9	...	15

7 rows × 26 columns

In the above last two examples, we have operate the Multiple Condition Filtering without using query method. NOTE: More knowledge and examples will be given in the next chapter.

Nlargest or nsmallest

In some cases, we do not have a specific range for filtering but just need the largest or smallest values. The nlargest and nsmallest functions allow for selecting rows that have the largest or smallest values in a column,respectively.

In [116...

```
reading_csv_files_example.nlargest(4, 'wheel-base')
```

Out[116...

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
70	0	122.0	mercedes-benz	gas	std	four	sedan	rwd	front	120.9	...	308
67	-1	93.0	mercedes-benz	diesel	turbo	four	sedan	rwd	front	115.6	...	183
68	-1	122.0	mercedes-benz	gas	std	four	sedan	rwd	front	115.6	...	234

In [117...

```
reading_csv_files_example.nsmallest(3, 'normalized-losses')
```

Out[117...

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
169	-1	65.0	toyota	gas	std	four	sedan	fwd	front	102.4	...	122
170	-1	65.0	toyota	diesel	turbo	four	sedan	fwd	front	102.4	...	110
171	-1	65.0	toyota	gas	std	four	hatchback	fwd	front	102.4	...	122

3 rows × 26 columns

In [118...

```
reading_csv_files_example.nsmallest(7,'engine-size')
```

Out[118...

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
17	2	121.0	chevrolet	gas	std	two	hatchback	fwd	front	88.4	...	61
52	3	150.0	mazda	gas	std	two	hatchback	rwd	front	95.3	...	70
53	3	150.0	mazda	gas	std	two	hatchback	rwd	front	95.3	...	70
54	3	150.0	mazda	gas	std	two	hatchback	rwd	front	95.3	...	70
31	1	101.0	honda	gas	std	two	hatchback	fwd	front	93.7	...	79
55	3	150.0	mazda	gas	std	two	hatchback	rwd	front	95.3	...	80
18	1	98.0	chevrolet	gas	std	two	hatchback	fwd	front	94.5	...	90

7 rows × 26 columns

select_dtypes() Method

This function return a subset of the DataFrame's columns based on the column dtypes. The parameters of this function can be set to include all the columns having some specific data type or it could be set to exclude all those columns which has some specific data types. [Reference] (https://www.geeksforgeeks.org/python-pandas-dataframe-select_dtypes/)

In [119...

```
reading_csv_files_example.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          201 non-null    int64
 1   normalized-losses  201 non-null    float64
 2   make               201 non-null    object
 3   fuel-type          201 non-null    object
 4   aspiration         201 non-null    object
 5   num-of-doors       201 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               201 non-null    float64
 19  stroke             201 non-null    float64
 20  compression-ratio  201 non-null    float64
 21  horsepower         201 non-null    float64
 22  peak-rpm           201 non-null    float64
 23  city-mpg           201 non-null    int64
 24  highway-mpg        201 non-null    int64
 25  price              201 non-null    int64
dtypes: float64(10), int64(6), object(10)
memory usage: 41.0+ KB
```

In [120…

```
select_dtypes_example = reading_csv_files_example.select_dtypes(include='float64')

select_dtypes_example
```

Out[120…

	normalized-losses	wheel-base	length	width	height	bore	stroke	compression-ratio	horsepower	peak-rpm
0	122.0	88.6	168.8	64.1	48.8	3.47	2.68	9.0	111.0	5000.0
1	122.0	88.6	168.8	64.1	48.8	3.47	2.68	9.0	111.0	5000.0
2	122.0	94.5	171.2	65.5	52.4	2.68	3.47	9.0	154.0	5000.0
3	164.0	99.8	176.6	66.2	54.3	3.19	3.40	10.0	102.0	5500.0
4	164.0	99.4	176.6	66.4	54.3	3.19	3.40	8.0	115.0	5500.0
…	…	…	…	…	…	…	…	…	…	…
196	95.0	109.1	188.8	68.9	55.5	3.78	3.15	9.5	114.0	5400.0
197	95.0	109.1	188.8	68.8	55.5	3.78	3.15	8.7	160.0	5300.0
198	95.0	109.1	188.8	68.9	55.5	3.58	2.87	8.8	134.0	5500.0
199	95.0	109.1	188.8	68.9	55.5	3.01	3.40	23.0	106.0	4800.0
200	95.0	109.1	188.8	68.9	55.5	3.78	3.15	9.5	114.0	5400.0

201 rows × 10 columns

In [121…

```
select_dtypes_example2 = reading_csv_files_example.select_dtypes(include='bool')

select_dtypes_example2
```

Out[121…

0
1
2
3
4
…
196
197
198
199
200

201 rows × 0 columns

In [122…

```
select_dtypes_example3 = reading_csv_files_example.select_dtypes(include='int64')

select_dtypes_example3
```

Out[122…

	symboling	curb-weight	engine-size	city-mpg	highway-mpg	price
0	3	2548	130	21	27	13495
1	3	2548	130	21	27	16500
2	1	2823	152	19	26	16500
3	2	2337	109	24	30	13950
4	2	2824	136	18	22	17450
…	…	…	…	…	…	…
196	-1	2952	141	23	28	16845
197	-1	3049	141	19	25	19045
198	-1	3012	173	18	23	21485
199	-1	3217	145	26	27	22470
200	-1	3062	141	19	25	22625

201 rows × 6 columns

In [123…

```
select_dtypes_example4 = reading_csv_files_example.select_dtypes(include='object')

select_dtypes_example4
```

Out[123…

	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system
0	alfa-romero	gas	std	two	convertible	rwd	front	dohc	four	mpfi
1	alfa-romero	gas	std	two	convertible	rwd	front	dohc	four	mpfi
2	alfa-romero	gas	std	two	hatchback	rwd	front	ohcv	six	mpfi
3	audi	gas	std	four	sedan	fwd	front	ohc	four	mpfi
4	audi	gas	std	four	sedan	4wd	front	ohc	five	mpfi
...	...	...	...	...	...	...	...	...	...	...
196	volvo	gas	std	four	sedan	rwd	front	ohc	four	mpfi
197	volvo	gas	turbo	four	sedan	rwd	front	ohc	four	mpfi
198	volvo	gas	std	four	sedan	rwd	front	ohcv	six	mpfi
199	volvo	diesel	turbo	four	sedan	rwd	front	ohc	six	idi
200	volvo	gas	turbo	four	sedan	rwd	front	ohc	four	mpfi

201 rows × 10 columns

4 - Math and Descriptive Statistics

Import the Dataset

In [124...

```
reading_csv_files_example.head()
```

Out[124...

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	sy
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...	109	
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	...	136	

5 rows × 26 columns

Shape Method: Return a tuple representing the dimensionality of the DataFrame. [Reference] (https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.shape.html)

In [125…
```
reading_csv_files_example.shape
```

Out[125…
```
(201, 26)
```

Describe Method: It returns description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains these information for each column: count - The number of not-empty values. mean - The average (mean) value. std - The standard deviation. min - the minimum value. 25% - The 25% percentile*. 50% - The 50% percentile*. 75% - The 75% percentile*. max - the maximum value. *Percentile meaning: how many of the values are less than the given percentile. Read more about percentiles in our Machine Learning Percentile chapter. [Reference](https://www.w3schools.com/python/pandas/ref_df_describe.asp)

In [126…
```
reading_csv_files_example.describe()
```

Out[126…

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	
count	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	20
mean	0.840796	122.00000	98.797015	174.200995	65.889055	53.766667	2555.666667	126.875622	
std	1.254802	31.99625	6.066366	12.322175	2.101471	2.447822	517.296727	41.546834	
min	-2.000000	65.00000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	
25%	0.000000	101.00000	94.500000	166.800000	64.100000	52.000000	2169.000000	98.000000	
50%	1.000000	122.00000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	
75%	2.000000	137.00000	102.400000	183.500000	66.600000	55.500000	2926.000000	141.000000	
max	3.000000	256.00000	120.900000	208.100000	72.000000	59.800000	4066.000000	326.000000	

In question columns with " describe()" method

In [127…
```
reading_csv_files_example[['bore','stroke','price']].describe()
```

Out[127…

	bore	stroke	price
count	201.000000	201.000000	201.000000
mean	3.330692	3.256874	13207.129353
std	0.268072	0.316048	7947.066342
min	2.540000	2.070000	5118.000000
25%	3.150000	3.110000	7775.000000
50%	3.310000	3.290000	10295.000000
75%	3.580000	3.410000	16500.000000
max	3.940000	4.170000	45400.000000

Info method: It prints information about the DataFrame. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells

in each column (non-null values). [Reference](https://www.w3schools.com/python/pandas/ref_df_info.asp)

In [128…
```
reading_csv_files_example.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          201 non-null    int64  
 1   normalized-losses  201 non-null    float64
 2   make               201 non-null    object 
 3   fuel-type          201 non-null    object 
 4   aspiration         201 non-null    object 
 5   num-of-doors       201 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               201 non-null    float64
 19  stroke             201 non-null    float64
 20  compression-ratio  201 non-null    float64
 21  horsepower         201 non-null    float64
 22  peak-rpm           201 non-null    float64
 23  city-mpg           201 non-null    int64  
 24  highway-mpg        201 non-null    int64  
 25  price              201 non-null    int64  
dtypes: float64(10), int64(6), object(10)
memory usage: 41.0+ KB
```

Unique method: Return unique values based on a hash table.

In [129…
```
reading_csv_files_example["make"].unique()
```

Out[129…
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'mercedes-benz', 'mercury',
       'mitsubishi', 'nissan', 'peugot', 'plymouth', 'porsche', 'renault',
       'saab', 'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

Nunique method: The nunique() method returns the number of unique values for each column. By specifying the column axis (axis='columns'), the nunique() method searches column-wise and returns the number of unique values for each row. [Reference](https://www.w3schools.com/python/pandas/ref_df_nunique.asp)

In [130…
```
reading_csv_files_example["make"].nunique()
```

Out[130…
```
22
```

value_counts() method: Return a Series containing counts of unique values. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html)

In [131…
```
reading_csv_files_example["make"].value_counts()
```

Out[131…
```
toyota           32
nissan           18
mazda            17
mitsubishi       13
honda            13
volkswagen       12
subaru           12
peugot           11
volvo            11
dodge             9
bmw               8
mercedes-benz     8
plymouth          7
saab              6
audi              6
porsche           4
jaguar            3
alfa-romero       3
chevrolet         3
isuzu             2
renault           2
mercury           1
Name: make, dtype: int64
```

Sum() method: Return the sum of the values over the requested axis. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sum.html)

In [132…
```
reading_csv_files_example["horsepower"].sum()
```

Out[132…
```
20784.512315270935
```

count() method: Count non-NA cells for each column or row. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.count.html)

In [133…
```
reading_csv_files_example["horsepower"].count()
```

Out[133…
```
201
```

max() method: Return the maximum of the values over the requested axis. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.max.html)

In [134…
```
reading_csv_files_example[['horsepower']].max()
```

Out[134…
```
horsepower    262.0
dtype: float64
```

min() method: Return the minimum of the values over the requested axis. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.min.html)

In [135…
```
reading_csv_files_example[['horsepower']].min()
```

Out[135…
```
horsepower    48.0
dtype: float64
```

mean() method: Count non-NA cells for each column or row. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.count.html)

In [136…
```
reading_csv_files_example[['horsepower']].mean()
```

Out[136…
```
horsepower    103.405534
dtype: float64
```

median() method: Return the median of the values over the requested axis. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.median.html)

In [137…
```
reading_csv_files_example[['horsepower']].median()
```

Out[137…
```
horsepower    95.0
dtype: float64
```

mode() method: Get the mode(s) of each element along the selected axis. The mode of a set of values is the value that appears most often. It can be multiple values. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mode.html)

In [138…
```
reading_csv_files_example[['horsepower']].mode()
```

Out[138…

	horsepower
0	68.0

std() method: Return sample standard deviation over requested axis. Normalized by N-1 by default. This can be changed using the ddof argument. [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.std.html)

In [139…
```
reading_csv_files_example[['horsepower']].std()
```

Out[139…
```
horsepower    37.3657
dtype: float64
```

std() method: Aggregate

Aggregate function: That is used to apply some aggregation across one or more column. Aggregate using callable, string, dict, or list of string/callables. Most frequently used aggregations are: sum: Return the sum of the values for the requested axis min: Return the minimum of the values for the requested axis max: Return the maximum of the values for the requested axis [Reference](https://www.geeksforgeeks.org/python-pandas-dataframe-aggregate/)

In [140…
```
reading_csv_files_example[["horsepower"]].aggregate(['min'])
```

Out[140…

	horsepower
min	48.0

In [141…
```
reading_csv_files_example[["horsepower"]].aggregate(['max'])
```

Out[141…

	horsepower
max	262.0

In [142…
```
reading_csv_files_example[["horsepower"]].aggregate(['sum'])
```

Out[142…

	horsepower
sum	20784.512315

In [143…
```
reading_csv_files_example[["horsepower"]].aggregate(['count'])
```

Out[143…

	horsepower
count	201

In [144…
```
reading_csv_files_example[["horsepower"]].aggregate(['std'])
```

Out[144…

	horsepower
std	37.3657

In [145…
```
reading_csv_files_example[["horsepower"]].aggregate(['median'])
```

Out[145…

	horsepower
median	95.0

In [146…
```
reading_csv_files_example[["horsepower"]].aggregate(['mode'])
```

Out[146…

	horsepower
	mode
0	68.0

Aggregate function with Groupby method: Grouping and aggregating will help to achieve data analysis easily using various functions. These methods will help us to the group and summarize our data and make complex analysis comparatively easy. [Reference] (https://www.geeksforgeeks.org/grouping-and-aggregating-with-pandas/)

In [147…

```
reading_csv_files_example.groupby(by = "make")[['horsepower']].aggregate(['min'])
```

Out[147…

make	horsepower min
alfa-romero	111.000000
audi	102.000000
bmw	101.000000
chevrolet	48.000000
dodge	68.000000
honda	58.000000
isuzu	78.000000
jaguar	176.000000
mazda	64.000000
mercedes-benz	123.000000
mercury	175.000000
mitsubishi	68.000000
nissan	55.000000
peugot	95.000000
plymouth	68.000000
porsche	143.000000
renault	104.256158
saab	110.000000
subaru	69.000000
toyota	56.000000
volkswagen	52.000000
volvo	106.000000

In [148…

```
reading_csv_files_example.groupby(by = "make")[['horsepower']].aggregate(['max'])
```

Out[148…

make	horsepower max
alfa-romero	154.000000
audi	140.000000
bmw	182.000000
chevrolet	70.000000

	horsepower
	max
make	
dodge	145.000000
honda	101.000000
isuzu	90.000000
jaguar	262.000000
mazda	135.000000
mercedes-benz	184.000000
mercury	175.000000
mitsubishi	145.000000
nissan	200.000000
peugot	142.000000
plymouth	145.000000
porsche	207.000000
renault	104.256158
saab	160.000000
subaru	111.000000

In [149…

```
reading_csv_files_example.groupby(by = "make")[['horsepower']].aggregate(['median'])
```

Out[149…

	horsepower
	median
make	
alfa-romero	111.000000
audi	110.000000
bmw	121.000000
chevrolet	70.000000
dodge	68.000000
honda	76.000000
isuzu	84.000000
jaguar	176.000000
mazda	84.000000
mercedes-benz	139.000000
mercury	175.000000
mitsubishi	102.000000
nissan	69.000000
peugot	95.000000

	horsepower
	median
make	
plymouth	68.000000
porsche	207.000000
renault	104.256158
saab	110.000000
subaru	82.000000
toyota	82.500000

In [150…

```
reading_csv_files_example.groupby(by = "make")[['horsepower']].aggregate(['mean'])
```

Out[150…

	horsepower
	mean
make	
alfa-romero	125.333333
audi	114.500000
bmw	138.875000
chevrolet	62.666667
dodge	86.333333
honda	80.230769
isuzu	84.000000
jaguar	204.666667
mazda	85.529412
mercedes-benz	146.250000
mercury	175.000000
mitsubishi	104.076923
nissan	102.555556
peugot	99.818182
plymouth	86.714286
porsche	191.000000
renault	104.256158
saab	126.666667
subaru	86.250000
toyota	92.781250
volkswagen	81.083333
volvo	128.000000

corr() method: That is used to find the pairwise correlation of all columns in the dataframe. Any na values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored. [Reference](https://www.geeksforgeeks.org/python-pandas-dataframe-corr/)

In [151…

```
reading_csv_files_example.corr()
```

Out[151…

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581	-0.140
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360	-0.029
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027	0.493
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025	0.608
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436	0.544
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694	0.180
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849072	0.644
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000	0.572
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572609	1.000
stroke	-0.008153	0.055045	0.158018	0.123952	0.188822	-0.060663	0.167438	0.205928	-0.055
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028889	0.001
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822676	0.566
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733	-0.267
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546	-0.582
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.679571	-0.591
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.872335	0.543

5 - Date Time

Convert argument to datetime

In [152…

```
import datetime
```

" now() " method provides the current date and time

In [153…

```
date_time_example = pd.DataFrame({'Date':[datetime.datetime.now()]})

date_time_example
```

Out[153...

	Date
0	2022-08-29 10:23:58.177598

" now().year " method provides the current year

In [154...

```
date_time_example2 = pd.DataFrame({'Date':[datetime.datetime.now().year]})

date_time_example2
```

Out[154...

	Date
0	2022

" strftime("%A") " method provides the current day

In [155...

```
date_time_example3 = pd.DataFrame({'Date':[datetime.datetime.now().strftime("%A")]})

date_time_example3
```

Out[155...

	Date
0	Monday

" strftime("%B") " method provides the current month

In [156...

```
date_time_example4 = pd.DataFrame({'Date':[datetime.datetime.now().strftime("%B")]})

date_time_example4
```

Out[156...

	Date
0	August

" strftime("%C") " method provides the current year

In [157...

```
date_time_example5 = pd.DataFrame({'Date':[datetime.datetime.now().strftime("%C")]})

date_time_example5
```

Out[157...

	Date
0	20

" strftime("%A") " method provides the current date

In [158...

```
date_time_example6 = pd.DataFrame({'Date':[datetime.datetime.now().strftime("%D")]})

date_time_example6
```

Out[158…

	Date
0	08/29/22

The other example with " strftime("%A") " method for specific date

In [159…

```
date_time_example_current_time = pd.DataFrame({'Date':[datetime.datetime(2022,5,10)]})

date_time_example_current_time
```

Out[159…

	Date
0	2022-05-10

In [160…

```
date_time_example7 = pd.DataFrame({'Date':[datetime.datetime(2022,5,10).strftime("%A")]})

date_time_example7
```

Out[160…

	Date
0	Tuesday

In [161…

```
date_time_example8 = pd.DataFrame({'Date':[datetime.datetime(2022,5,10).strftime("%B")]})

date_time_example8
```

Out[161…

	Date
0	May

In [162…

```
date_time_example9 = pd.DataFrame({'Date':[datetime.datetime(2022,5,10).strftime("%C")]})

date_time_example9
```

Out[162…

	Date
0	20

In [163…

```
date_time_example10 = pd.DataFrame({'Date':[datetime.datetime(2022,5,10).strftime("%D")]})

date_time_example10
```

Out[163…

	Date
0	05/10/22

In [164…

```
date_time_example11 = pd.DataFrame({'Date':[datetime.datetime(2022,9,2),
                                            datetime.datetime(2021,8,3),
                                            datetime.datetime(2020,7,4),
                                            datetime.datetime(2019,6,5),
                                            datetime.datetime(2018,5,6)],
                                    'City':['London','Berlin','Toronto','New York','Zurich']
                                   })

date_time_example11
```

Out[164…

	Date	City
0	2022-09-02	London
1	2021-08-03	Berlin
2	2020-07-04	Toronto
3	2019-06-05	New York
4	2018-05-06	Zurich

The determination of the exact year, month, day and day of week in datetime method() example

In [165…

```
date_time_example11['Year'] = date_time_example11['Date'].dt.year
date_time_example11['Month'] = date_time_example11['Date'].dt.month
date_time_example11['Day'] = date_time_example11['Date'].dt.day
date_time_example11['Day of week'] = date_time_example11['Date'].dt.dayofweek

date_time_example11
```

Out[165…

	Date	City	Year	Month	Day	Day of week
0	2022-09-02	London	2022	9	2	4
1	2021-08-03	Berlin	2021	8	3	1
2	2020-07-04	Toronto	2020	7	4	5
3	2019-06-05	New York	2019	6	5	2
4	2018-05-06	Zurich	2018	5	6	6

6 - Grouping and Sorting

In [166…

```
reading_csv_files_example.head(2)
```

Out[166…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engine-size	sy
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	…	130	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	…	130	

2 rows × 26 columns

6.1 - Groupby

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups. [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html)

In [167…

```
reading_csv_files_example.groupby('make')[['horsepower']].mean()
```

Out[167…

make	horsepower
alfa-romero	125.333333
audi	114.500000
bmw	138.875000
chevrolet	62.666667
dodge	86.333333
honda	80.230769
isuzu	84.000000
jaguar	204.666667
mazda	85.529412
mercedes-benz	146.250000
mercury	175.000000
mitsubishi	104.076923
nissan	102.555556
peugot	99.818182
plymouth	86.714286
porsche	191.000000
renault	104.256158
saab	126.666667
subaru	86.250000
toyota	92.781250
volkswagen	81.083333
volvo	128.000000

6.2 - Sort_values Function

Sort by the values along either axis [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html)

In [168…

```
reading_csv_files_example.sort_values("normalized-losses",
                                      ascending = False, inplace = False,
                                      na_position ='last')
```

Out[168…

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	…	engi…
186	3	256.0	volkswagen	gas	std	two	hatchback	fwd	front	94.5	…	1
103	1	231.0	nissan	gas	std	two	hatchback	rwd	front	99.2	…	1
175	3	197.0	toyota	gas	std	two	hatchback	rwd	front	102.9	…	1
174	3	197.0	toyota	gas	std	two	hatchback	rwd	front	102.9	…	1
102	3	194.0	nissan	gas	turbo	two	hatchback	rwd	front	91.3	…	1
…	…	…	…	…	…	…	…	…	…	…	…	
173	-1	65.0	toyota	gas	std	four	hatchback	fwd	front	102.4	…	1
169	-1	65.0	toyota	gas	std	four	sedan	fwd	front	102.4	…	1
170	-1	65.0	toyota	diesel	turbo	four	sedan	fwd	front	102.4	…	1
171	-1	65.0	toyota	gas	std	four	hatchback	fwd	front	102.4	…	1
172	-1	65.0	toyota	gas	std	four	sedan	fwd	front	102.4	…	1

201 rows × 26 columns

In [169…

```
reading_csv_files_example.groupby('make')[
    ["horsepower","price"]].mean().sort_values(
    "horsepower",ascending=False,inplace=False)
```

Out[169…

make	horsepower	price
jaguar	204.666667	34600.000000
porsche	191.000000	31400.500000
mercury	175.000000	16503.000000
mercedes-benz	146.250000	33647.000000
bmw	138.875000	26118.750000
volvo	128.000000	18063.181818
saab	126.666667	15223.333333
alfa-romero	125.333333	15498.333333
audi	114.500000	17859.166667
renault	104.256158	9595.000000
mitsubishi	104.076923	9239.769231

make	horsepower	price
nissan	102.555556	10415.666667
peugot	99.818182	15489.090909
toyota	92.781250	9885.812500
plymouth	86.714286	7963.428571
dodge	86.333333	7875.444444
subaru	86.250000	8541.250000
mazda	85.529412	10652.882353
isuzu	84.000000	8916.500000
volkswagen	81.083333	10077.500000

In [170…

```
reading_csv_files_example.groupby('make')[
    ["horsepower","price"]].mean().sort_values(
    "price",ascending=False,inplace=False)
```

Out[170…

make	horsepower	price
jaguar	204.666667	34600.000000
mercedes-benz	146.250000	33647.000000
porsche	191.000000	31400.500000
bmw	138.875000	26118.750000
volvo	128.000000	18063.181818
audi	114.500000	17859.166667
mercury	175.000000	16503.000000
alfa-romero	125.333333	15498.333333
peugot	99.818182	15489.090909
saab	126.666667	15223.333333
mazda	85.529412	10652.882353
nissan	102.555556	10415.666667
volkswagen	81.083333	10077.500000
toyota	92.781250	9885.812500
renault	104.256158	9595.000000
mitsubishi	104.076923	9239.769231
isuzu	84.000000	8916.500000
subaru	86.250000	8541.250000
honda	80.230769	8184.692308
plymouth	86.714286	7963.428571

horsepower price

make

7 - Data Types

What are the most common data types that you will see in pandas? 1. int64 (integer) 2. float64 (floating point number) 3. object (string) 4. datetime (datetime) 5. bool (true or false) We can convert a column of one type into another using the astype function.

In below example, we are able to find the data types of columns out with the help of " .dtypes " function.

In [171…

```
reading_csv_files_example.dtypes
```

Out[171…

```
symboling              int64
normalized-losses    float64
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           float64
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                  int64
dtype: object
```

Moreover, in the other example, we are able to find the data types of the exact column out with the help of " .dtypes " function.

In [172…

```
reading_csv_files_example['symboling'].dtypes
```

Out[172…

```
dtype('int64')
```

Converting the Columns' Data Types

Astype Function: This method is used to cast a pandas object to a specified dtype. astype()

function also provides the capability to convert any suitable existing column to categorical type.

In [173…
```
reading_csv_files_example['symboling'].astype('float64')
```

Out[173…
```
0      3.0
1      3.0
2      1.0
3      2.0
4      2.0
      ...
196   -1.0
197   -1.0
198   -1.0
199   -1.0
200   -1.0
Name: symboling, Length: 201, dtype: float64
```

In [174…
```
reading_csv_files_example['height'].astype('int64')
```

Out[174…
```
0      48
1      48
2      52
3      54
4      54
       ..
196    55
197    55
198    55
199    55
200    55
Name: height, Length: 201, dtype: int64
```

We good to combine the data types after the changing them by using " astype " function with the help of the below table.

In [175…
```
reading_csv_files_example.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          201 non-null    int64  
 1   normalized-losses  201 non-null    float64
 2   make               201 non-null    object 
 3   fuel-type          201 non-null    object 
 4   aspiration         201 non-null    object 
 5   num-of-doors       201 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               201 non-null    float64
 19  stroke             201 non-null    float64
 20  compression-ratio  201 non-null    float64
 21  horsepower         201 non-null    float64
 22  peak-rpm           201 non-null    float64
 23  city-mpg           201 non-null    int64
 24  highway-mpg        201 non-null    int64
 25  price              201 non-null    int64
dtypes: float64(10), int64(6), object(10)
```

8 - Combining DataFrame

** **

8.1 - Concat

Concatenate pandas objects along a particular axis with optional set logic along the other axes. Can also add a layer of hierarchical indexing on the concatenation axis, which may be useful if the labels are the same (or overlapping) on the passed axis number. [Reference] (https://pandas.pydata.org/docs/reference/api/pandas.concat.html)

In [176…

```
concat_example1 = {'Name': ['Rebekka','Katie'],'City': ['Toronto','Zurich']}

concat_example2 = {'Name': ['Carl','Gustav'],'City': ['New York','Basel']}
```

In [177…

```
df_concat1 = pd.DataFrame(concat_example1)
df_concat1
```

Out[177…

	Name	City
0	Rebekka	Toronto
1	Katie	Zurich

In [178…

```
df_concat2 = pd.DataFrame(concat_example2)
df_concat2
```

Out[178…

	Name	City
0	Carl	New York
1	Gustav	Basel

In below example, we have used " concate " method to concatenate two DataFrames for column-based with the help pf " axis = 0 ".

In [179…

```
pd.concat([df_concat1,df_concat2],axis=0,ignore_index=False)
```

Out[179…

	Name	City

	Name	City
0	Rebekka	Toronto
1	Katie	Zurich
0	Carl	New York

In [180…
```
pd.concat([df_concat1,df_concat2],axis=0,ignore_index=True)
```

Out[180…

	Name	City
0	Rebekka	Toronto
1	Katie	Zurich
2	Carl	New York
3	Gustav	Basel

In below example, we have used " concate " method to concatenate two DataFrames for row-based with the help pf " axis = 1 ".

In [181…
```
pd.concat([df_concat1,df_concat2],axis=1,ignore_index=False)
```

Out[181…

	Name	City	Name	City
0	Rebekka	Toronto	Carl	New York
1	Katie	Zurich	Gustav	Basel

In [182…
```
pd.concat([df_concat1,df_concat2],axis=1,ignore_index=True)
```

Out[182…

	0	1	2	3
0	Rebekka	Toronto	Carl	New York
1	Katie	Zurich	Gustav	Basel

Creating Random Variables in DataFrame with " concat " method

In [183…
```
pd.concat([pd.DataFrame([i], columns = ['A']) for i in range(3)], ignore_index = True)
```

Out[183…

	A
0	0
1	1
2	2

In [184…
```
pd.concat( [pd.DataFrame([i], columns = ["A"]) for i in np.random.random(3)], ignore_index = True)
```

Out[184…

	A

	A
0	0.845695
1	0.028184

Concatenating 3 created variables with random method in numpy by using " Concat " method

In [185…

```
data_concat1 = np.random.random((4,4))
data_concat1_df = pd.DataFrame(data_concat1)

data_concat2 = np.random.randint(111,222,size=(4,4))
data_concat2_df2 = pd.DataFrame(data_concat2)

data_concat3 = np.random.randint(3,4,size=(4,4))
data_concat3_df3 = pd.DataFrame(data_concat3)

concat_example_concat = pd.concat([data_concat1_df,data_concat2_df2,data_concat3_df3],axis=0,ignor
concat_example_concat
```

Out[185…

	0	1	2	3
0	0.558141	0.447123	0.957587	0.310080
1	0.959080	0.336362	0.537394	0.802518
2	0.825477	0.489251	0.393771	0.724862
3	0.991508	0.956542	0.471254	0.168881
4	139.000000	216.000000	154.000000	114.000000
5	214.000000	176.000000	150.000000	189.000000
6	212.000000	170.000000	185.000000	125.000000
7	169.000000	174.000000	210.000000	139.000000
8	3.000000	3.000000	3.000000	3.000000
9	3.000000	3.000000	3.000000	3.000000
10	3.000000	3.000000	3.000000	3.000000
11	3.000000	3.000000	3.000000	3.000000

8.2 - Merge

The merge() method updates the content of two DataFrame by merging them together, using the specified method(s). [Reference](https://www.w3schools.com/python/pandas/ref_df_merge.asp) Merge DataFrame or named Series objects with a database-style join. [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html)

In [186…

```
merge_df1 = pd.DataFrame({'Users': ['foo', 'bar', 'baz', 'foo'],
                          'Value': [1, 2, 3, 5]})

merge_df2 = pd.DataFrame({'Users_2': ['foo', 'bar', 'baz', 'foo'],
                          'Value': [5, 6, 7, 8]})
```

1. left_on : label or list, or array-like Column or index level names to join on in the left DataFrame. Can also be an array or list of arrays of the length of the left DataFrame. These arrays are treated as if they are columns. 2. right_on : label or list, or array-like Column or index level names to join on in the right DataFrame. Can also be an array or list of arrays of the length of the right DataFrame. These arrays are treated as if they are columns. 3. suffixes : list-like, default is ("_x", "_y") A length-2 sequence where each element is optionally a string indicating the suffix to add to overlapping column names in left and right respectively. Pass a value of None instead of a string to indicate that the column name from left or right should be left as-is, with no suffix. At least one of the values must not be None. [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html)

In [187…

```
merge_df1.merge(merge_df2, left_on='Users', right_on='Users_2')
```

Out[187…

	Users	Value_x	Users_2	Value_y
0	foo	1	foo	5
1	foo	1	foo	8
2	foo	5	foo	5
3	foo	5	foo	8
4	bar	2	bar	6
5	baz	3	baz	7

In [188…

```
merge_df1.merge(merge_df2, left_on='Users', right_on='Users_2', suffixes=('of User', 'of Users2'))
```

Out[188…

	Users	Valueof User	Users_2	Valueof Users2
0	foo	1	foo	5
1	foo	1	foo	8
2	foo	5	foo	5
3	foo	5	foo	8
4	bar	2	bar	6
5	baz	3	baz	7

In [189…

```
merge_df2.merge(merge_df1, left_on='Users_2', right_on='Users', suffixes=('of User_2', 'of Users')
```

Out[189…

Users_2	Valueof User_2	Users	Valueof Users

	Users_2	Valueof User_2	Users	Valueof Users
0	foo	5	foo	1
1	foo	5	foo	5
2	foo	8	foo	1
3	foo	8	foo	5
4	bar	6	bar	2

how : {‘left’, ‘right’, ‘outer’, ‘inner’, ‘cross’}, default ‘inner’ Type of merge to be performed. 1. left: use only keys from left frame, similar to a SQL left outer join; preserve key order. 2. right: use only keys from right frame, similar to a SQL right outer join; preserve key order. 3. outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically. 4. inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys. 5. cross: creates the cartesian product from both frames, preserves the order of the left keys. [Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html)

In [190…
```
merge_df3 = pd.DataFrame({'Users': ['A','B'], 'Values1':[1,2], 'Location':['Pink','Floyd']})
merge_df4 = pd.DataFrame({'Users': ['A','B'], 'Values2':[33,44], 'Location':['The','Wall']})
```

In [191…
```
merge_df3.merge(merge_df4, how = 'inner', on = 'Users')
```

Out[191…

	Users	Values1	Location_x	Values2	Location_y
0	A	1	Pink	33	The
1	B	2	Floyd	44	Wall

In [192…
```
merge_df4.merge(merge_df3, how = 'inner', on = 'Users')
```

Out[192…

	Users	Values2	Location_x	Values1	Location_y
0	A	33	The	1	Pink
1	B	44	Wall	2	Floyd

In [193…
```
merge_df3.merge(merge_df4, how = 'left', on = 'Users')
```

Out[193…

	Users	Values1	Location_x	Values2	Location_y
0	A	1	Pink	33	The
1	B	2	Floyd	44	Wall

In [194…
```
merge_df4.merge(merge_df3, how = 'right', on = 'Users')
```

Out[194…

	Users	Values2	Location_x	Values1	Location_y
0	A	33	The	1	Pink

	Users	Values2	Location_x	Values1	Location_y

In [195…
```
merge_df3.merge(merge_df4, how = 'cross')
```

Out[195…

	Users_x	Values1	Location_x	Users_y	Values2	Location_y
0	A	1	Pink	A	33	The
1	A	1	Pink	B	44	Wall
2	B	2	Floyd	A	33	The
3	B	2	Floyd	B	44	Wall

8.3 - Append

Append function is used to append rows of other dataframe to the end of the given dataframe, returning a new dataframe object. Columns not in the original dataframes are added as new columns and the new cells are populated with NaN value. [Reference] (https://www.geeksforgeeks.org/python-pandas-dataframe-append/)

In [196…
```
df_append1 = pd.DataFrame([[1, 2, 99], [3, 4, 88]], columns=list('ABC'), index=['x', 'y'])

df_append1
```

Out[196…

	A	B	C
x	1	2	99
y	3	4	88

In [197…
```
df_append2 = pd.DataFrame([[5, 6, 11], [7, 8, 22]], columns=list('ABC'), index=['x', 'y'])

df_append2
```

Out[197…

	A	B	C
x	5	6	11
y	7	8	22

In [198…
```
df_append1.append(df_append2)
```

Out[198…

	A	B	C
x	1	2	99
y	3	4	88
x	5	6	11
y	7	8	22

In [199…

```
df_append1.append(df_append2,ignore_index= True)
```

Out[199…

	A	B	C
0	1	2	99
1	3	4	88
2	5	6	11
3	7	8	22

In [200…

```
df_append3 = pd.DataFrame(columns = ['A'])

for i in range(5):
    df_append3 = df_append3.append({'A': i}, ignore_index = True)

df_append3
```

Out[200…

	A
0	0
1	1
2	2
3	3
4	4

In [201…

```
import numpy as np

df_append4 = pd.DataFrame(columns = ['A','B','X'])

for i in range(5):
    df_append4 = df_append4.append({'A': i,
                                    'B':np.random.rand(),
                                    'X': np.random.randint(140,150)},
                                    ignore_index = True)

df_append4
```

Out[201…

	A	B	X
0	0.0	0.900046	143.0
1	1.0	0.831271	141.0
2	2.0	0.368875	140.0
3	3.0	0.085680	145.0
4	4.0	0.870988	140.0

8.4 - Join

Join columns of another DataFrame. Join columns with other DataFrame either on index or on a

key column. Efficiently join multiple DataFrame objects by index at once by passing a list.
[Reference](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.join.html)

In [202…
```
join_df = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'], 'A': ['A0', 'A1', 'A2', 'A3',

join_df
```

Out[202…

	key	A
0	K0	A0
1	K1	A1
2	K2	A2
3	K3	A3
4	K4	A4
5	K5	A5

In [203…
```
join_df2 = pd.DataFrame({'key': ['K0', 'K1', 'K2'], 'B': ['B0', 'B1', 'B2']})

join_df2
```

Out[203…

	key	B
0	K0	B0
1	K1	B1
2	K2	B2

In [204…
```
join_df.join(join_df2,lsuffix = '_caller', rsuffix = '_other')
```

Out[204…

	key_caller	A	key_other	B
0	K0	A0	K0	B0
1	K1	A1	K1	B1
2	K2	A2	K2	B2
3	K3	A3	NaN	NaN
4	K4	A4	NaN	NaN
5	K5	A5	NaN	NaN

In [205…
```
join_df.set_index('key').join(join_df2.set_index('key'))
```

Out[205…

	A	B
key		
K0	A0	B0
K1	A1	B1
K2	A2	B2

	A	B
key		
K3	A3	NaN

In [206…

```
join_df.join(join_df2.set_index('key'), on = 'key')
```

Out[206…

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	NaN
4	K4	A4	NaN
5	K5	A5	NaN

In [207…

```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```