Module 2: Pandas Series

In the previous module, you are introduced to Pandas which is a library written for the Python programming language and is a great tool for data manipulation and analysis. In this module, you will start learning how to use it.

We first need to import the Pandas library via the Python's import command:

In [1]:

```
import pandas as pd
```

as keyword is used to shorten the library name by giving it an alias. **pd** is the widely used alias for pandas among Python users.

Now, let’s create our first pandas Series object from a dataset. Even though Pandas Series can read all kinds of data, we have to make sure to maintain data consistency.

Here, we have a list of ice cream flavors which we pass to pandas library to create Pandas Series.

In [2]:

```
ice_cream = ["Chocolate", "Banana", "Vanilla", "Strawberry"]

pd.Series(ice_cream)
```

Out[2]:

```
0     Chocolate
1        Banana
2       Vanilla
3    Strawberry
dtype: object
```

Dtype means the **data type for the Series**. **Object** indicates that the Series is **String** type. Notice the numbers generated on the left side? These are the **indexes of each element in the Series**. The difference between the indexes in Pandas Series and the ones in Python list is that they do not have to be numeric. We will learn how to change the index and access it using .loc function in the next few lessons.

In [3]:

```
lottery = [34,74,12,98,19]

pd.Series(lottery)
```

Out[3]:

```
0    34
1    74
2    12
3    98
4    19
dtype: int64
```

Referring to the above example, we have created a new list with different data type and then subsequently created a Pandas Series from that list.

We notice that the dtype is different from the previous one. It is shown as **int64** indicating our data type is now **Integer**.

Hint : Keep in mind that the index for list and Pandas Series starts with 0. Hence, the last number will always be less than the total length of the list.

What would happen if we combine different data types into one list?

In [4]:

```
combine = ice_cream + lottery
pd.Series(combine)
```

Out[4]:

```
0     Chocolate
1        Banana
2       Vanilla
3    Strawberry
4            34
5            74
6            12
7            98
8            19
dtype: object
```

The Series will automatically be converted to **object or String** type if we combine different data types in it. Bear in mind that this will cause problem if we plan to do mathematical operations on it later.

Hint : "+" operator also works if we want to combine two or more list.

If we create a Pandas Series from a python dictionary, we can see the **difference in its index**. For this case, the **keys in dictionary** are taken to be the **indexes in the Pandas Series** as shown in the example below.

In [5]:

```
student_grade = {
    "Amin" : "80",
    "Senoi" : "90",
    "Danial" : "89",
    "Aqiff" : "100"
}

pd.Series(student_grade)
```

Out[5]:

```
Amin       80
Aqiff     100
Danial     89
Senoi      90
dtype: object
```

Combining dictionary and list

In [6]:

```
student_grade = {
    "Amin" : [88,79,99,87],
    "Senoi" : [99,76,97,84],
    "Danial" : [82,49,59,87],
    "Aqiff" : [78,79,69,37]
}

pdStudent = pd.Series(student_grade)
pdStudent
```

Out[6]:

```
Amin      [88, 79, 99, 87]
Aqiff     [78, 79, 69, 37]
Danial    [82, 49, 59, 87]
Senoi     [99, 76, 97, 84]
dtype: object
```

In [7]:

```
pdStudent["Amin"]
```

Out[7]:

```
[88, 79, 99, 87]
```

Attributes and Methods in Pandas Series

Attribute does not modify or manipulate the object in any way. Its purpose is to display and present information.

Method performs some kind of operation, manipulation or calculation on the objects.

In [8]:

```
lottery = [34,74,12,98,19]

s = pd.Series(lottery)
s
```

Out[8]:

```
0    34
1    74
2    12
3    98
4    19
dtype: int64
```

.values attribute returns an array of values.

In [9]:

```
s.values
```

Out[9]:

```
array([34, 74, 12, 98, 19], dtype=int64)
```

.index attribute shows us information on the index of the Series

- **start** : the starting index of the Series
- **stop** : the last index of the Series

More : https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.RangeIndex.html#pandas.RangeIndex (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.RangeIndex.html#pandas.RangeIndex)

In [10]:

```
s.index
```

Out[10]:

```
RangeIndex(start=0, stop=5, step=1)
```

In [11]:

```
s.dtypes
```

Out[11]:

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

From here, you can understand the usage of attribute and what it means by only display and present information

Pandas Series Methods

The main difference between attributes and methods is, **methods have parentheses, ()** at the end while attribute does not.

.sum() method returns summation of all the values in the Series. Hence, we do not need to use any loop to do the calculation.

In [12]:

```
s.sum()
```

Out[12]:

```
237
```

.count() returns the total number of elements in the Series. NaN values will not be counted.

In [13]:

```
s.count()
```

Out[13]:

```
5
```

.mean() returns the average value of the Series.

In [14]:

```
s.mean()
```

Out[14]:

```
47.4
```

We can also do Mathematical operations on the methods.

In [15]:

```
s.sum() / s.count()
```

Out[15]:

```
47.399999999999999
```

.product() method will multiple all the values together in the Series

In [16]:

```
s.product()
```

Out[16]:

```
56217504
```

Parameters and Arguments

Parameter and argument are almost the same thing.

When we create a method, we need to specify what parameter(s) we need. Then, when we are calling the method, we need to give argument(s) according to its parameter.

Some parameters are set to None, hence there is a default value to the parameter.

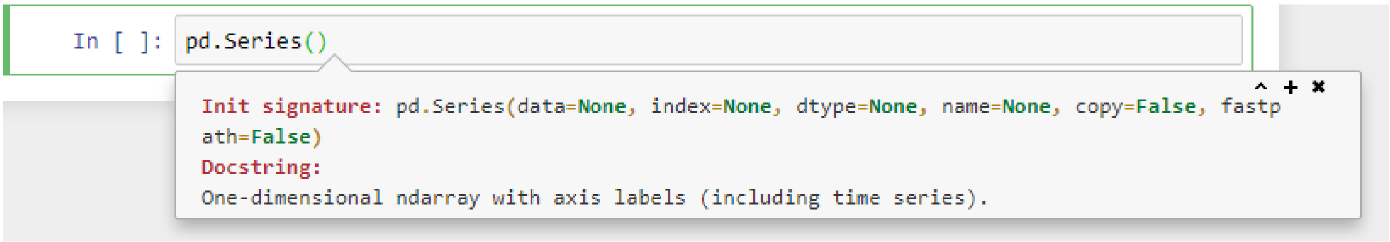
The method below is not run. I simply write the method and click the Shift key + Tab key to show the details of the method.

In [17]:

```
pd.Series()
```

Out[17]:

```
Series([], dtype: float64)
```



```
In [ ]: pd.Series()
```

```
Init signature: pd.Series(data=None, index=None, dtype=None, name=None, copy=False, fastp
ath=False)
Docstring:
One-dimensional ndarray with axis labels (including time series).
```

Here, we create a Pandas Series on students' grades and set the index to their names.

As you can see, **each name** (acting as index) has a **number** mapped to it.

In [18]:

```
student_name = ["Amin", "Senoi" ,"Danial", "Aqiff",]
grade = [88,79,99,87]

pd.Series(grade, student_name)
```

Out[18]:

```
Amin      88
Senoi     79
Danial    99
Aqiff     87
dtype: int64
```

Another way of inserting the arguments is by specifying which is the data, and which is the index. This way, we do not have to insert the arguments in a particular order(data first, then index)

In [19]:

```
pd.Series(data = grade, index = student_name)
```

Out[19]:

```
Amin      88
Senoi     79
Danial    99
Aqiff     87
dtype: int64
```

What will happen if the index is not unique?

To give you an example, let's add one additional data on both lists. We will create two indexes with the same name but holding different values.

In [20]:

```
student_name = ["Amin", "Senoi" ,"Danial", "Aqiff","Amin"]
grade = [88,79,99,87,50]

s= pd.Series(grade, student_name)
s
```

Out[20]:

```
Amin      88
Senoi     79
Danial    99
Aqiff     87
Amin      50
dtype: int64
```

In [21]:

```
s["Amin"]
```

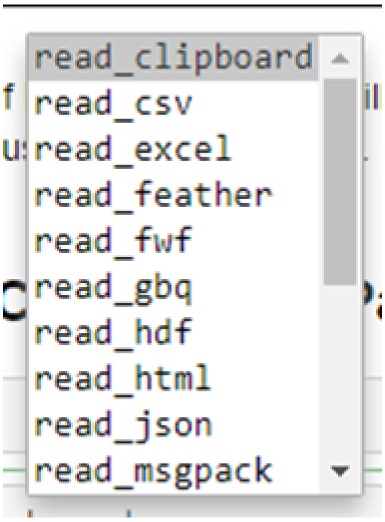
Out[21]:

```
Amin    88
Amin    50
dtype: int64
```

When we call the function, it will show two values, which is not practical if we want to access each row. Hence, it is advisable to keep the index name unique or use the default one.

CSV file into Pandas Series

Pandas can read many types of files; for example, JSON, Excel, CSV, and HTML.



In the following slides, we will learn how to read CSV file and convert it into Pandas Series.

Explanation of each parameters and arguments:

- **"data/pokemon.csv"** is the location of the file.
- **usecols** indicates that we only want to import Pokemon column.
- **squeeze** is set to **True** to **change Pandas DataFrame into Pandas Series**.

.head() is a method to display the first 5 rows. Without this method, **Pandas will show the first 30 rows and last 30 rows** of the dataset.

In [22]:

```
pd.read_csv("data/pokemon.csv").head()
```

Out[22]:

	Pokemon	Type
0	Bulbasaur	Grass
1	Ivysaur	Grass
2	Venusaur	Grass
3	Charmander	Fire
4	Charmeleon	Fire

In [23]:

```
pd.read_csv("data/pokemon.csv", usecols=["Pokemon"]).head()
```

Out[23]:

	Pokemon
0	Bulbasaur
1	Ivysaur
2	Venusaur
3	Charmander
4	Charmeleon

In [24]:

```
pd.read_csv("data/pokemon.csv", usecols=["Pokemon"], squeeze= True).head()
```

Out[24]:

```
0     Bulbasaur
1       Ivysaur
2      Venusaur
3    Charmander
4    Charmeleon
Name: Pokemon, dtype: object
```

In [25]:

```
pokemon = pd.read_csv("data/pokemon.csv", usecols=["Pokemon"], squeeze= True)
pokemon
```

Out[25]:

```
0       Bulbasaur
1         Ivysaur
2        Venusaur
3      Charmander
4      Charmeleon
5       Charizard
6        Squirtle
7       Wartortle
8       Blastoise
9        Caterpie
10        Metapod
11     Butterfree
12         Weedle
13         Kakuna
14       Beedrill
15         Pidgey
16      Pidgeotto
17        Pidgeot
18        Rattata
19       Raticate
20        Spearow
21         Fearow
22          Ekans
23          Arbok
24        Pikachu
25         Raichu
26      Sandshrew
27      Sandslash
28        Nidoran
29       Nidorina
          ...
691     Clauncher
692     Clawitzer
693    Helioptile
694     Heliolisk
695        Tyrunt
696     Tyrantrum
697        Amaura
698       Aurorus
699       Sylveon
700      Hawlucha
701       Dedenne
702       Carbink
703         Goomy
704       Sliggoo
705        Goodra
706        Klefki
707      Phantump
708     Trevenant
709     Pumpkaboo
710     Gourgeist
711      Bergmite
712       Avalugg
713        Noibat
```

```
714       Noivern
715       Xerneas
716       Yveltal
717       Zygarde
718       Diancie
719         Hoopa
720     Volcanion
Name: Pokemon, Length: 721, dtype: object
```

Notice how the each code returns different output?

Try to import google_stock_price.csv file into Pandas. You will get the result as below:

In [26]:

```
google = pd.read_csv("data/google_stock_price.csv", squeeze = True)
google
```

Out[26]:

```
0        50.12
1        54.10
2        54.65
3        52.38
4        52.95
5        53.90
6        53.02
7        50.95
8        51.13
9        50.07
10       50.70
11       49.95
12       50.74
13       51.10
14       51.10
15       52.61
16       53.70
17       55.69
```

.head() and .tail() methods

Both of these methods returns a copy of objects at certain rows in the Pandas Series.

.head() method will take the first few objects while **.tail()** method will take the last few objects in the Series. By default, the methods will take the first 5 or last 5 unless an argument is specified.

In [27]:

```
pokemon.head()
```

Out[27]:

```
0     Bulbasaur
1       Ivysaur
2      Venusaur
3    Charmander
4    Charmeleon
Name: Pokemon, dtype: object
```

In [28]:

```
pokemon.tail()
```

Out[28]:

```
716      Yveltal
717      Zygarde
718      Diancie
719        Hoopa
720    Volcanion
Name: Pokemon, dtype: object
```

In [29]:

```
pokemon.head(10)
```

Out[29]:

```
0     Bulbasaur
1       Ivysaur
2      Venusaur
3    Charmander
4    Charmeleon
5     Charizard
6      Squirtle
7     Wartortle
8     Blastoise
9      Caterpie
Name: Pokemon, dtype: object
```

Python Built-in Function

- **len()** : returns the total elements in a list
- **type()** : returns the type of list of elements
- **dir()** : returns all available attributes and methods within the object
- **sorted()** : returns a sorted list in alphabetical or ascending order
- **dict()** : returns Python dictionary data type
- **list()** : returns Python list data type
- **min()** : returns the minimum value from the list
- **max()** : returns the maximum value from the list

In [30]:

```
len(pokemon) , len(google)
```

Out[30]:

```
(721, 3012)
```

In [31]:

```
type(pokemon)
```

Out[31]:

```
pandas.core.series.Series
```

In [32]:

```
dir(pokemon)
```

```
 '__array_prepare__',
 '__array_priority__',
 '__array_wrap__',
 '__bool__',
 '__bytes__',
 '__class__',
 '__contains__',
 '__copy__',
 '__deepcopy__',
 '__delattr__',
 '__delitem__',
 '__dict__',
 '__dir__',
 '__div__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__finalize__',
 '__float__',
 '__floordiv__',
```

In [33]:

```
sorted(pokemon)
```

Out[33]:

```
['Abomasnow',
 'Abra',
 'Absol',
 'Accelgor',
 'Aegislash',
 'Aerodactyl',
 'Aggron',
 'Aipom',
 'Alakazam',
 'Alomomola',
 'Altaria',
 'Amaura',
 'Ambipom',
 'Amoonguss',
 'Ampharos',
 'Anorith',
 'Arbok',
 'Arcanine'.
```

In [34]:

```
sorted(google)
```

Out[34]:

```
[49.950000000000003,
 50.07,
 50.119999999999997,
 50.700000000000003,
 50.740000000000002,
 50.950000000000003,
 51.100000000000001,
 51.100000000000001,
 51.130000000000003,
 52.380000000000003,
 52.609999999999999,
 52.950000000000003,
 53.020000000000003,
 53.700000000000003,
 53.899999999999999,
 54.100000000000001,
 54.649999999999999,
 55.689999999999998.
```

In [35]:

```
dict(google)
```

Out[35]:

```
{0: 50.119999999999997,
 1: 54.100000000000001,
 2: 54.649999999999999,
 3: 52.380000000000003,
 4: 52.950000000000003,
 5: 53.899999999999999,
 6: 53.020000000000003,
 7: 50.950000000000003,
 8: 51.130000000000003,
 9: 50.07,
 10: 50.700000000000003,
 11: 49.950000000000003,
 12: 50.740000000000002,
 13: 51.100000000000001,
 14: 51.100000000000001,
 15: 52.609999999999999,
 16: 53.700000000000003,
 17: 55.689999999999998.
```

In [36]:

```
max(google)
```

Out[36]:

```
782.22000000000003
```

In [37]:

```
min(pokemon)
```

Out[37]:

```
'Abomasnow'
```

Pandas Series Attributes on CVS file

.is_unique attribute returns a Boolean value. True if there are no duplicates and False if there are duplicate values in the Series.

For Pokemon Series, is_unique attribute returns True meaning every single value in the Series is unique. There is no pokemon with the same name.

In [38]:

```
pokemon.is_unique
```

Out[38]:

```
True
```

Google Series has duplicates because there are some days that have the same stock value.

In [39]:

```
google.is_unique
```

Out[39]:

```
False
```

.ndim attribute returns the dimension of the Series. In some cases, we need to create multidimensional Series.

In [40]:

```
google.ndim
```

Out[40]:

```
1
```

.shape attribute returns the size of the Series in tuple data type.

Google have 3012 rows and 1 columns

In [41]:

```
google.shape
```

Out[41]:

```
(3012,)
```

.size attribute gives information about the total number of cells in the Series. (Keep in mind that it will also count the null values).

In [42]:

```
google.size
```

Out[42]:

```
3012
```

You can modify the Series name using **.name** attribute.

In [43]:

```
pokemon.name = "Pocket Monsters"
```

In [44]:

```
pokemon.head()
```

Out[44]:

```
0     Bulbasaur
1       Ivysaur
2      Venusaur
3    Charmander
4    Charmeleon
Name: Pocket Monsters, dtype: object
```

Pandas Series Methods on CSV file

.sort_values() returns sorted Pandas Series objects.

Hint : Methods Chaining is a style of invoking multiple method calls sequentially. For instance, after calling .sort_values method, we can call .head() method.

In [45]:

```
pokemon.sort_values().head()
```

Out[45]:

```
459    Abomasnow
62          Abra
358        Absol
616     Accelgor
680    Aegislash
Name: Pocket Monsters, dtype: object
```

In [46]:

```
pokemon.sort_values(ascending=False).head()
```

Out[46]:

```
717     Zygarde
633    Zweilous
40        Zubat
569       Zorua
570     Zoroark
Name: Pocket Monsters, dtype: object
```

If we want to get the highest stock price in Google Series, we can use either of the methods below.

In [47]:

```
google.max()
```

Out[47]:

```
782.22000000000003
```

```
In [48]:
```

```
google.sort_values(ascending=False).head(1)
```

```
Out[48]:
```

```
3011    782.22
Name: Stock Price, dtype: float64
```

inplace parameter : overwrites the original variable with the new result.

```
In [49]:
```

```
google.head(3)
```

```
Out[49]:
```

```
0    50.12
1    54.10
2    54.65
Name: Stock Price, dtype: float64
```

```
In [50]:
```

```
google.sort_values(ascending=False, inplace=True)
```

```
In [51]:
```

```
google.head(3)
```

```
Out[51]:
```

```
3011    782.22
2859    776.60
3009    773.18
Name: Stock Price, dtype: float64
```

.sort_index() method : sort the list based on the index.

If we sort the Pokemon Series according to its value, we can see that the order of the index number has changed.

```
In [52]:
```

```
pokemon.sort_values(ascending=False, inplace=True)
pokemon.head()
```

```
Out[52]:
```

```
717     Zygarde
633    Zweilous
40        Zubat
569       Zorua
570     Zoroark
Name: Pocket Monsters, dtype: object
```

To sort the series again based on the index number, we can use **.sort_index()** method.

```
In [53]:
```

```
pokemon.sort_index(inplace=True)
pokemon.head()
```

```
Out[53]:
```

```
0     Bulbasaur
1       Ivysaur
2      Venusaur
3    Charmander
4    Charmeleon
Name: Pocket Monsters, dtype: object
```

Pandas *in* keyword

returns a Boolean value after checking the values in the list. It will return **True** if the **element exists** in the list, and False if it does not.

```
In [54]:
```

```
3 in [1,2,3,4,5]
```

```
Out[54]:
```

```
True
```

```
In [55]:
```

```
pokemon.head()
```

```
Out[55]:
```

```
0     Bulbasaur
1       Ivysaur
2      Venusaur
3    Charmander
4    Charmeleon
Name: Pocket Monsters, dtype: object
```

For Pandas Series, by default, **in** keyword will check the Series' index. If we want to check the values, then we need to specify it.

```
In [56]:
```

```
"Bulbasaur" in pokemon
```

```
Out[56]:
```

```
False
```

```
In [57]:
```

```
"Bulbasaur" in pokemon.values
```

```
Out[57]:
```

```
True
```

Extract Values by Index Number Position

Series works like a list. We can access specific data using square bracket, [ ] notation. Let’s access the first and last data of Pokemon.

In [58]:

```
pokemon.head()
```

Out[58]:

```
0     Bulbasaur
1       Ivysaur
2      Venusaur
3    Charmander
4    Charmeleon
Name: Pocket Monsters, dtype: object
```

In [59]:

```
pokemon[0]
```

Out[59]:

```
'Bulbasaur'
```

In [60]:

```
pokemon.tail()
```

Out[60]:

```
716      Yveltal
717      Zygarde
718      Diancie
719        Hoopa
720    Volcanion
Name: Pocket Monsters, dtype: object
```

In [61]:

```
pokemon[720]
```

Out[61]:

```
'Volcanion'
```

We can also access a list of specific data.

In [62]:

```
lst = [100,200,300,400]

pokemon[lst]
```

Out[62]:

```
100    Electrode
200        Unown
300     Delcatty
400    Kricketot
Name: Pocket Monsters, dtype: object
```

Access data in a certain range by using colon (:). For example, let’s display the pokemon name between number 10 to 20.

Hint: Always add 1 to the end number. Like in this example, we would like to end at number 20. So we have to type 21.

In [63]:

```
pokemon[10:21]
```

Out[63]:

```
10       Metapod
11    Butterfree
12        Weedle
13        Kakuna
14      Beedrill
15        Pidgey
16     Pidgeotto
17       Pidgeot
18       Rattata
19      Raticate
20       Spearow
Name: Pocket Monsters, dtype: object
```

We can also access the Series using negative number. Negative number means the counting starts backward.

Here, we are displaying the last 10 values in the Series.

In [64]:

```
pokemon[-10:]
```

Out[64]:

```
711     Bergmite
712      Avalugg
713       Noibat
714      Noivern
715      Xerneas
716      Yveltal
717      Zygarde
718      Diancie
719        Hoopa
720    Volcanion
Name: Pocket Monsters, dtype: object
```

Extract Series Values by Index Label

To do this, we need to change the index from number to the Pokemon name using **index_col** parameter.

In [65]:

```
pokemon = pd.read_csv("data/pokemon.csv", index_col = "Pokemon", squeeze=True)
pokemon.head(3)
```

Out[65]:

```
Pokemon
Bulbasaur    Grass
Ivysaur      Grass
Venusaur     Grass
Name: Type, dtype: object
```

In [66]:

```
pokemon[["Bulbasaur" ,"Ditto", "Meowth"]]
```

Out[66]:

```
Pokemon
Bulbasaur     Grass
Ditto        Normal
Meowth       Normal
Name: Type, dtype: object
```

If the index label does not exist, it will prompt error.

In [67]:

```
pokemon["Digimon"]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-
packages\pandas\core\indexes\base.py in get_value(self, series, key)
   2482             try:
-> 2483                 return libts.get_value_box(s, key)
   2484             except IndexError:

pandas/_libs/tslib.pyx in pandas._libs.tslib.get_value_box (pandas\_libs\tsl
ib.c:18843)()

pandas/_libs/tslib.pyx in pandas._libs.tslib.get_value_box (pandas\_libs\tsl
ib.c:18477)()

TypeError: 'str' object cannot be interpreted as an integer

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-67-c753fd87bd34> in <module>()
----> 1 pokemon["Digimon"]

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-
packages\pandas\core\series.py in __getitem__(self, key)
    599         key = com._apply_if_callable(key, self)
    600         try:
--> 601             result = self.index.get_value(self, key)
    602 
    603             if not is_scalar(result):

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-
packages\pandas\core\indexes\base.py in get_value(self, series, key)
   2489                     raise InvalidIndexError(key)
   2490                 else:
-> 2491                     raise e1
   2492             except Exception:  # pragma: no cover
   2493                 raise e1

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-
packages\pandas\core\indexes\base.py in get_value(self, series, key)
   2475         try:
   2476             return self._engine.get_value(s, k,
-> 2477                                           tz=getattr(series.dtype,
 'tz', None))
   2478         except KeyError as e1:
   2479             if len(self) > 0 and self.inferred_type in ['integer',
'boolean']:

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHa
shTable.get_item()
```

```
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHa
shTable.get_item()

KeyError: 'Digimon'
```

The situation is different when we try to extract using a list of index labels and only a few of them does not exist. For example, in the list specified below, only "Digimon" does not exist as an index label in the Pokemon Series but the code will not prompt any error. However, it will state the value for that particular label as **NaN** which stands for **Not Available or Not a Number**.

In [68]:

```
pokemon[["Meowth", "Digimon", "Charizard"]]
```

Out[68]:

```
Pokemon
Meowth       Normal
Digimon         NaN
Charizard      Fire
Name: Type, dtype: object
```

We can extract a range of values using index labels. Notice that the last value is included too.

In [69]:

```
pokemon["Metapod" : "Spearow"]
```

Out[69]:

```
Pokemon
Metapod          Bug
Butterfree       Bug
Weedle           Bug
Kakuna           Bug
Beedrill         Bug
Pidgey        Normal
Pidgeotto     Normal
Pidgeot       Normal
Rattata       Normal
Raticate      Normal
Spearow       Normal
Name: Type, dtype: object
```

.get() method on Series

This method enables us to extract values from the Series too. The only different is that, if the value is not available, it will not return error but it will return the default value instead.

```
In [70]:
```

```
pokemon.head(3)
```

```
Out[70]:

Pokemon
Bulbasaur    Grass
Ivysaur      Grass
Venusaur     Grass
Name: Type, dtype: object
```

```
In [71]:
```

```
pokemon.get("Bulbasaur")
```

```
Out[71]:

'Grass'
```

```
In [72]:
```

```
pokemon.get("Digimon")
```

```
In [73]:
```

```
pokemon.get(["Bulbasaur", "Meowth"])
```

```
Out[73]:

Pokemon
Bulbasaur     Grass
Meowth       Normal
Name: Type, dtype: object
```

```
In [74]:
```

```
pokemon.get(["Bulbasaur", "Meowth", "Digimon"])
```

```
Out[74]:

Pokemon
Bulbasaur     Grass
Meowth       Normal
Digimon         NaN
Name: Type, dtype: object
```

```
In [75]:
```

```
pokemon.get("Digimon", default="The Pokemon is not available")
```

```
Out[75]:

'The Pokemon is not available'
```

In **if else** statement, .get() method is very useful.

In [76]:

```
pet = "Charizard"
if pokemon.get(pet):
    print("Charizard")
else:
    print("Not Available")
```

```
Charizard
```

Math Methods on Series Object

There are many mathematical methods that we can use to help ease our works.

In [77]:

```
google = pd.read_csv("data/google_stock_price.csv", squeeze = True)
google.head()
```

Out[77]:

```
0    50.12
1    54.10
2    54.65
3    52.38
4    52.95
Name: Stock Price, dtype: float64
```

In [78]:

```
google.median()
```

Out[78]:

```
283.315
```

.describe() method gives a brief information on the Series.

- count: total number of elements
- mean : the average number of the Series
- std : Standard Deviation
- min : smallest value in the Series
- max : highest value in the Series
- 25% : 1st quartile.
- 50% : 2nd quartile/ median.
- 75% : 3rd quartile

In [79]:

```
google.describe()
```

Out[79]:

```
count    3012.000000
mean      334.310093
std       173.187205
min        49.950000
25%       218.045000
50%       283.315000
75%       443.000000
max       782.220000
Name: Stock Price, dtype: float64
```

Finding IQR, Lower Fence, Upper Fence.

In [80]:

```
IQR = google.describe()["75%"] - google.describe()["25%"]
IQR
```

Out[80]:

```
224.95499999999998
```

In [81]:

```
lowerFence = google.describe()["25%"] - 1.5* IQR
lowerFence
```

Out[81]:

```
-119.38749999999999
```

In [82]:

```
upperFence = google.describe()["75%"] + 1.5* IQR
upperFence
```

Out[82]:

```
780.4325
```

In [83]:

```
google.quantile()
```

Out[83]:

```
283.315
```

In [84]:

```
google.plot.box()
```

Out[84]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16cfef0e9e8>
```

In [85]:

```
g = pd.read_csv("data/google_stock_price.csv")
```

In [86]:

```
g.boxplot(return_type = "axes" , figsize=(10,10) )
```

Out[86]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16cfef0e9e8>
```

.idxmax() and .idxmin() Methods

Returns the position index of the max/min value.

In [87]:

```
google.min()
```

Out[87]:

```
49.950000000000003
```

In [88]:

```
minIndex = google.idxmin()
google[minIndex]
```

Out[88]:

```
49.950000000000003
```

.value_counts() Method

Returns a new Series on unique counts on the Series. For example, we want to know how many Fire and Water Pokemon.

In [89]:

```
pokemon = pd.read_csv("data/pokemon.csv", squeeze = True, index_col= "Pokemon" )
pokemon.head()
```

Out[89]:

```
Pokemon
Bulbasaur     Grass
Ivysaur       Grass
Venusaur      Grass
Charmander     Fire
Charmeleon     Fire
Name: Type, dtype: object
```

In [90]:

```
pokemon.value_counts()
```

Out[90]:

```
Water       105
Normal       93
Grass        66
Bug          63
Fire         47
Psychic      47
Rock         41
Electric     36
Ground       30
Dark         28
Poison       28
Fighting     25
Dragon       24
Ice          23
Ghost        23
Steel        22
Fairy        17
Flying        3
Name: Type, dtype: int64
```

There are 105 Water Pokemon and 47 Fire Pokemon.

In [91]:

```
pokemon.value_counts().sum() == pokemon.count()
```

Out[91]:

```
True
```

.apply() method

Apply changes on every value in the Series based on the passed method.

For instance, we want to set a threshold on the google stock performance so we create a method as shown.

In [92]:

```
def performace_indicator(number):
    if number < 300:
        return "OK"
    elif number >= 300 and number <= 650:
        return "Quite good"
    else: return "Incredible!"
```

Then, we invoke that method on the Series’ elements using the apply() method.

In [93]:

```
google.apply(performace_indicator).head()
```

Out[93]:

```
0    OK
1    OK
2    OK
3    OK
4    OK
Name: Stock Price, dtype: object
```

In [94]:

```
google.apply(performace_indicator).tail()
```

Out[94]:

```
3007    Incredible!
3008    Incredible!
3009    Incredible!
3010    Incredible!
3011    Incredible!
Name: Stock Price, dtype: object
```

The .map() method

Map values of Series according to input correspondence.

In [95]:

```
pokemon_names = pd.read_csv("data/pokemon.csv", usecols=["Pokemon"], squeeze=True)
pokemon_names.head(3)
```

Out[95]:

```
0    Bulbasaur
1      Ivysaur
2     Venusaur
Name: Pokemon, dtype: object
```

In [96]:

```
pokemon_types = pd.read_csv("data/pokemon.csv", index_col="Pokemon" , squeeze=True)
pokemon_types.head(3)
```

Out[96]:

```
Pokemon
Bulbasaur    Grass
Ivysaur      Grass
Venusaur     Grass
Name: Type, dtype: object
```

In [97]:

```
pokemon_names.map(pokemon_types)
```

Out[97]:

```
0         Grass
1         Grass
2         Grass
3          Fire
4          Fire
5          Fire
6         Water
7         Water
8         Water
9           Bug
10          Bug
11          Bug
12          Bug
13          Bug
14          Bug
15       Normal
16       Normal
17       Normal
18       Normal
19       Normal
20       Normal
21       Normal
22       Poison
23       Poison
24     Electric
25     Electric
26       Ground
27       Ground
28       Poison
29       Poison
         ...
691       Water
692       Water
693    Electric
694    Electric
695        Rock
696        Rock
697        Rock
698        Rock
699       Fairy
700    Fighting
701    Electric
702        Rock
703      Dragon
704      Dragon
705      Dragon
706       Steel
707       Ghost
708       Ghost
709       Ghost
710       Ghost
711         Ice
712         Ice
713      Flying
714      Flying
```

```
715       Fairy
716        Dark
717      Dragon
718        Rock
719     Psychic
720        Fire
Name: Pokemon, Length: 721, dtype: object
```