2. Pandas Series

Pandas is a software library written for the Python programming language for data manipulation and analysis. First, we need to import the library before we can use it. Simple command on how we can import them.

In [1]:

```
import pandas as pd
```

as keyword is us to shorten the library name. Meanwhile, **pd** short form widely uses among Python users.

Let's create our first pandas Series object from a dataset. Pandas Series can read all kinds of data, but we should maintain the consistency of data.

Here I have a list of ice cream flavors. Next, I'll pass the list to the 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 Series**. **Object** indicates the Series is **String**. Notice that numbers are generated on the left side. It indicates the **index of particular element.**

The difference of index in Pandas Series and Python list is it does not have to be numeric. In the next few lessons, we will learn how to change the index and access it using .loc function.

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
```

Here I create a new list with different data types, which is **Integer,** and then create a Pandas Series.

Notice that the dtype is different from the previous one. It is now **int64** which indicate our data type is **Integer**

Hint: Keep in mind that index for list and Pandas Series start with 0. Hence, the last number will always 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 **become an object or String.** This can affect mathematical operation if the Series is in the wrong data type

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

In [5]:

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

pd.Series(student_grade)
```

Out[5]:

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

If we convert a dictionary into Pandas Series, we can see the **difference in the index.** The **key in the dictionary** has become the **index in Pandas Series.**

In

Combine dictionary and list

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

In [7]:

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

Out[7]:

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

Attributes and Methods in Pandas Series

Attribute do not modify the object or manipulate it. It view and gives us information On the hand, **Methods** do perform some kind of operation, manipulation or calculation.

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 return an array of values.

In [9]:

```
s.values
```

Out[9]: array([34, 74, 12, 98, 19],

dtype=int64)

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

- **start** : the starting index of the Series **stop**
- : the end index for the Series

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

(https://pandas.pydata.org/pandasdocs/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 give information

Pandas Series Methods

Main different with attributes and methods is, **methods have parentheses / ()** at the end while attribute does not.

.sum() method return summation of the all the values in the Series. Hence, we do not need to use for loop to calculate all the sum.

```

```

In [12]:

```
s
.sum()
```

Out[12]: 237

.count() return the total number in the Series. It will not count the NaN value.

[13]:

```
s.count()
```

Out[13]:

5

.mean() return average value for the Series In

[14]:

In

```
s.mean()
```

Out[14]:

```
47.4
```

We can also do Mathematic operations In

[15]:

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

Out[15]:

```
47.4
```

.product() method will multiple all the value inside the Series In

[16]:

```
s.product()
```

Out[16]:

```
56217504
```

Parameters and Arguments

Parameter and arguement is almost the same thing.

When we want to create a method, we need to specify what parameter(s) we need. Then, when we want to call the method, we need to give an argument according to the parameter.

Some parameter is set to **None**. Hence there is a default value to the parameter.

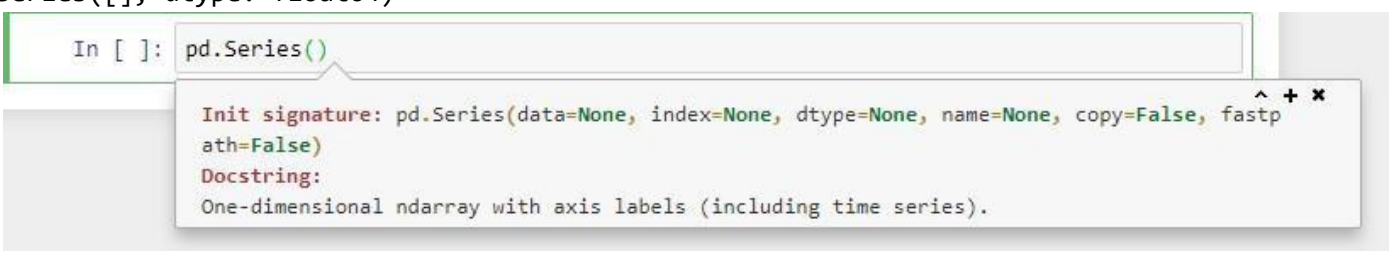
The method below is not run. I write the method and **click the Shift key + Tab key** to show the details of the method.

In [17]:

```
pd.Series()
```

Out[17]:

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



Here I want to create a Pandas Series about the grade, and I want to change the index into student names.

As you can see, **each name** (acting as index) **has a number** map on 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 using the argument is, we can specify the parameter name. If we are doing this way, we do not have to put the argument in the order.

In [19]:

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

Out[19]:

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

Both results are the same. Either way works
Next, what will happen if the index is not unique?

- For example, I have added one additional data on both lists.
- I have two indexes that have the same name but different values.

[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
```

In

If I call the function, it will show two values, which is not practical when 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 instance, JSON, Excel, CSV, and the list goes on.

I will show how to read CSV file and convert it into Pandas Series

There are a few arguments that I use.

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

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

r **3**

Charmand

er **4**

Charmeleo

n

```
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
  [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           ...
```

```
In
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 output change with changes of arguments

Try to import google_stock_price.csv file into Pandas. The answer is as below

[111]:

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

Out[111]:

```
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
18       55.94
19       56.93
20       58.69
21       59.62
22       58.86
23       59.13
24       60.35
25       59.86
26       59.07
27       63.37
28       65.47
29       64.74
          ...
2982    675.22
2983    668.26
2984    680.04
2985    684.11
2986    692.10
2987    699.21
2988    694.49
2989    697.77
2990    695.36
2991    705.63
2992    715.09
2993    720.64
2994    716.98
2995    720.95
2996    719.85
2997    733.78
```

```
In
2998    736.96
2999    741.19
3000    738.63
3001    742.74
3002    739.77
3003    738.42
3004    741.77 3005    745.91
3006    768.79
3007    772.88
3008    771.07
3009    773.18
3010    771.61
3011    782.22
Name: Stock Price, Length: 3012, dtype: float64
```

.head() and .tail() methods

Both of this methods create a copy of Pandas object for a certain row. For instance, the **default argument is 5**. Hence, it will **show first 5 for .head() method and last 5 for .tail() method of the Series.** If we want to change it, we can specify how much do we want `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
```

In

Python Built-in Function

- **len()** : return total elements in a list
-
- **dir()** : return all available attributes and methods within the object
- **sorted()** : return a sorted list in alphabetical order or ascending order
-
- **dict()** : return Python dictionary data type **list()** : return Python list
- data type **min()** : return minimum value from th e list **max()** : return a
- maximum value from the list In [30]:

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

type() : return type of list of elements

Out[30]:

```
(721, 3012)
```

In [31]:

```
type(pokemon)
```

Out[31]:

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

In [130]:

```
dir(pokemon)[:20]
```

Out[130]:

```
['Abra',
 'Alakazam',
'Arbok',
 'Arcanine',
 'Beedrill',
 'Bellsprout',
 'Blastoise',
 'Bulbasaur',
 'Butterfree',
 'Caterpie',
 'Charizard',
 'Charmander',
 'Charmeleon',
 'Clefable',
 'Clefairy',
 'Cloyster',
 'Dewgong',
 'Diglett',
 'Dodrio',
 'Doduo']
```

[129]:

```
sorted(pokemon)[:20]
```

Out[129]:

```
['Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug',
 'Bug']
```

In [128]:

```
sorted(google)[:20]
```

Out[128]:

```
[49.95,
 50.07,
 50.12,
50.7,
 50.74,
 50.95,
 51.1,
 51.1,
 51.13,
52.38,
52.61,
52.95,
 53.02,
 53.7,
53.9,
 54.1,
 54.65,
55.69,
 55.94,
 56.93]
```

In [120]:

```
dict(google)
```

Out[120]:

```
{0: 50.12,
 1: 54.1,
 2: 54.65,
 3: 52.38,
 4: 52.95,
 5: 53.9,
 6: 53.02,
 7: 50.95,
 8: 51.13,
 9: 50.07,
 10: 50.7,
 11: 49.95,
 12: 50.74,
 13: 51.1,
 14: 51.1,
 15: 52.61,
 16: 53.7,
 17: 55.69,
```

In [36]:

```
max(google)
```

Out[36]:

```
782.22
```

In [37]:

```
min(pokemon)
```

Out[37]:

```
'Abomasnow'
```

Pandas Series Attributes on CVS file

.is_unique attribute return a boolean value. True if there is no duplicates, False if there is duplicates value in the Series

In **Pokemon Series**, is_unique attribute return **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 do have duplicates because there are stocks with the same values.

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 attributes give information about total number of cells in the Series. This attribute also count the null value

In [42]:

```
google.size
```

Out[42]: 3012 modify the Series name using

.name attribute

In [43]:

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

[44]:

```
pokemon.head()
```

Out[44]:

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

In

Pandas Series Methods on CSV file

.sort_values() return a new sorted Pandas Series object .

Hint: **Methods Chaining, which** creates a sequence of methods. For instance, after calling .sort_values method, we then 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 do either method.

In [47]:

```
google.max()
```

Out[47]:

```
782.22
```

[48]:

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

Out[48]:

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

inplace parameter : overwrite the original variable with the new result In

[49]:

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

Out[49]:

```
0    50.12
1    54.10
```

In
```
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 base on the index.

Let's take change the Pokemon Series based on Pokemon column. Then, we can see 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
```

Hence, if we want to sort the series again based on the index number, we can use **.sort_index()** method
[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

Return a boolean value, which compares the value provided from the list. It will return **True** if the **element exists** in the list and False if it does not.

In
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
```

By default, *in* keyword will match with Series index. Hence, if we need to specify the Series values In

[56]:

```
"Bulbasaur" in pokemon
```

Out[56]:

```
False
```

In [57]:

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

Out[57]:

```
True
```

Extract Values by Index Number Position

y

The series works like a list. We can access specific data using 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'
```

Here is how we can 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 a data in range by using colon (:). For instances, let show the pokemon name between number 10 until 20.

Hint: Always add 1 to the end number. For example, the last number is 20. Hence, we need to specify the number 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 a negetive number. A negative number means the counting start from backward

Here, I want to show the last ten values.

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

Firstly, we read **pokemon.csv** file and change the index from number to the Pokemon name using the **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 value is not exist, it will prompt error.

```
[127]: pokemon["Digimon"]
```

In

```
---------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_value(sel
f, series, key)
   3123             try:
-> 3124                 return libindex.get_value_box(s, key)   3125
 except IndexError:

pandas\_libs\index.pyx in pandas._libs.index.get_value_box()

pandas\_libs\index.pyx in pandas._libs.index.get_value_box() TypeError:

'str' object cannot be interpreted as an integer During handling of the

above exception, another exception occurred:

KeyError                                  Traceback (most recent call las
t)
<ipython-input-127-ca2af26142e2> in <module>
----> 1 pokemon["Digimon"]

~\Anaconda3\lib\site-packages\pandas\core\series.py in __getitem__(self, k
ey)
  765  key = com._apply_if_callable(key, self)
  766  try: --> 767             result = self.index.get_value(self, key)
    768
    769             if not is_scalar(result):

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_value(sel
f, series, key)
 3130                     raise InvalidIndexError(key)
 3131                     else:
-> 3132                     raise e1
 3133                 except Exception:  # pragma: no cover
 3134                 raise e1

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_value(sel
f, series, key)
 3116             try:
 3117             return self._engine.get_value(s, k,
-> 3118                                           tz=getattr(series.dtype,
'tz', None))
  3119 except KeyError as e1:
  3120 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.PyObject
HashTable.get_item()
```

```
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObject
HashTable.get_item()

KeyError: 'Digimon'
```

However, it is different case when we extract many values and only a few do not exist. For example, "Digimon" is exist in Pokemon Series. Since we are extracting more than one values, then it will not promt error. However, it will state the value is **NaN** which stands for **Not Available or Not a Number** In [123]:

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

Out[123]:

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

Can we use Range to extract values using labels? **Of Course!!**

And, the last value is included too.

In [77]:

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

Out[77]:

```
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 enable us to extract values from the Series too. The different is, if the value is not available, it will not return error. However, it will return default value.

[78]:

pokemon.head(3) Out[78]:

In

```
Pokemon
Bulbasaur    Grass
Ivysaur      Grass
Venusaur     Grass Name:
Type, dtype: object In [79]:
```

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

```
Out[79]:
'Grass'

In [80]:
```

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

```
In [81]:
```

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

```
Out[81]:
Pokemon
Bulbasaur     Grass
Meowth       Normal Name:
Type, dtype: object In [82]:
```

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

```
Out[82]:
Pokemon
Bulbasaur     Grass
Meowth       Normal
Digimon         NaN Name:
Type, dtype: object

In [83]: pokemon.get("Digimon", default="The Pokemon is not
```

```
available") Out[83]:
'The Pokemon is not available'
```

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

In

[84]:

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

```
Charizard
```

Math Methods on Series Object

Methods will always ease our job to gain information. Here we will show some Mathematics methods that can be a help.

In [85]:

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

Out[85]:

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

In [86]:

```
google.median()
```

Out[86]:

```
283.315
```

.describe() method give 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
- [87]:

```
google.describe()
```

Out[87]:

```
count    3012.000000 mean
334.310093           std
```

In
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 [88]:

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

Out[88]:

224.95499999999998

In [89]:

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

Out[89]:

-119.38749999999999

In [90]:

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

Out[90]:

780.4325

In [91]:

```
google.quantile()
```

Out[91]:

283.315

In [92]:

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

Out[92]:

<matplotlib.axes._subplots.AxesSubplot at 0x25d59125780> [93]:

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

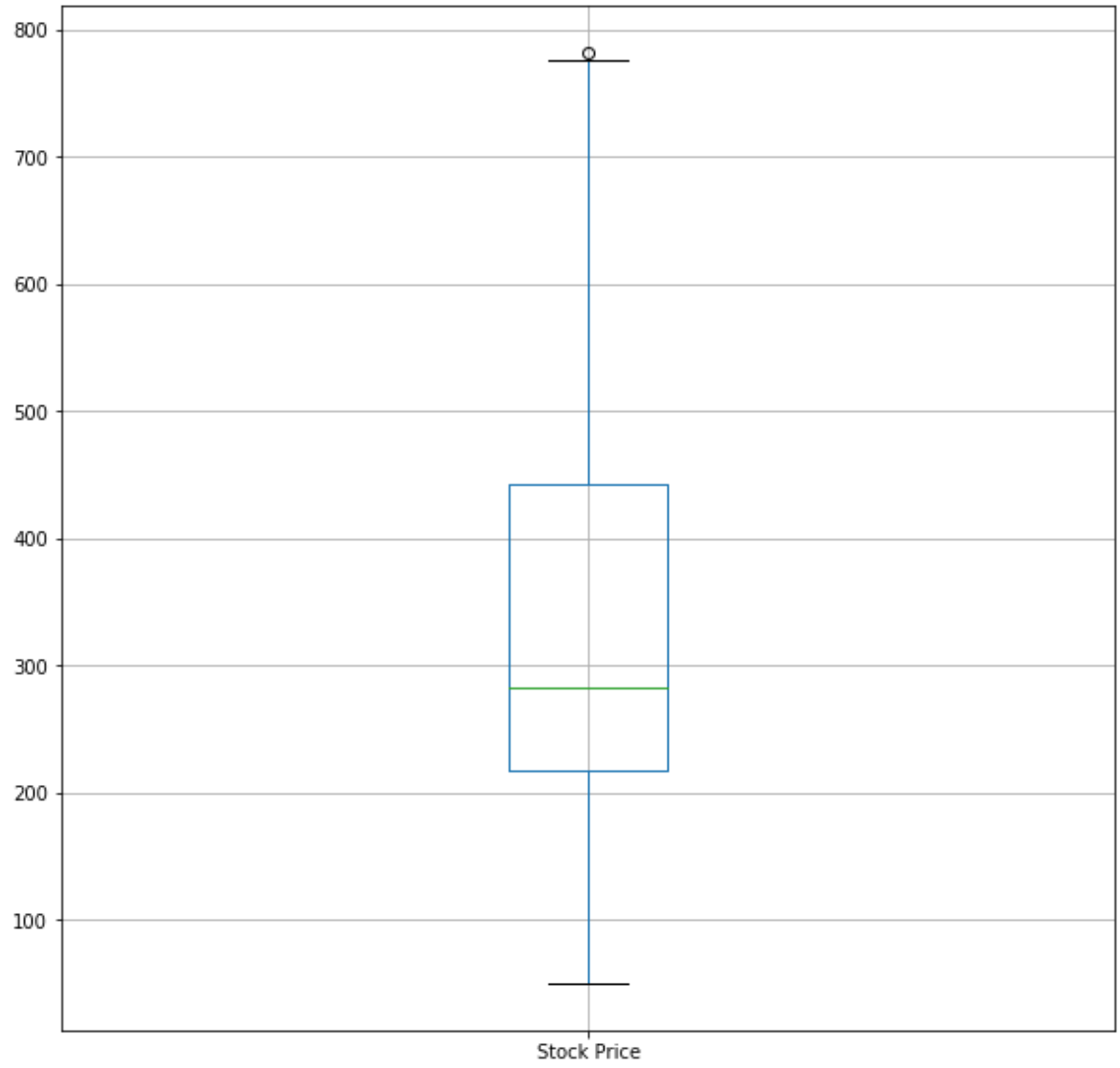
In [94]:

In

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

Out[94]: <matplotlib.axes._subplots.AxesSubplot at

0x25d5976b828>



.idxmax() and .idxmin() Methods

return the position index of the max/min value In

[95]:

```
google.min()
```

Out[95]:

49.95

In [96]:

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

Out[96]:

```
49.95
```

.value_counts() Method

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

In [97]:

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

Out[97]:

```
Pokemon
Bulbasaur     Grass
Ivysaur       Grass
Venusaur      Grass
Charmander     Fire
Charmeleon     Fire Name:
```

```
Type, dtype: object In [98]:
```

```
pokemon.value_counts()
```

Out[98]:

```
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 Pokemons and 47 Fire Pokemons

In [99]:

In

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

Out[99]:

```
True
```

.apply() method

apply changes on every value in the Series using method.

For example, i want to set a threshold on the google stock performace, i create method as follows.

In [100]:

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

In [101]:

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

Out[101]:

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

In [102]:

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

Out[102]:

```
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.

[103]:

```
pokemon_names = pd.read_csv("data/pokemon.csv", usecols=["Pokemon"],
squeeze=True) pokemon_names.head(3) Out[103]:
```

In

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

In [104]:

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

Out[104]:

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

In [105]:

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

Out[105]:

```
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
```

In

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
```

In [126]:

```
import warnings
warnings.filterwarnings('always')
```