Pandas

Pandas is one of the most important libary for the data analysis. This library consists of severeal methods and property that helps user to explore, modify and help them to draw conclusion. Generally, the data are available in tabular form (most of the time) and with the help of pandas, one can easily play with dataset and analyse it further.

In [2]:
```
# To work with pandas, we have to first import it with the command.
import pandas
```

In [3]:
```
# We use alias 'pd' for pandas library
import pandas as pd
```

Alias helps us to shorten the name of any module or library temporarily.

In pandas, we deal with three form of dataset.

- Series: 1-Dimensional
- Dataframe: 2-Dimensional (Most Important)
- Panel: Multi-Dimensional

Series

pandas.Series(data, index, dtype, copy)

In [9]:
```
l=['First', 'Second', 'Third', 'Fourth', 'Fifth']
s=pd.Series(data=l)   # Series method is use to create the 1-D series object.
print(s)
```

```
0     First
1    Second
2     Third
3    Fourth
4     Fifth
dtype: object
```

Note: Whenever we create Series, Dataframe, Panel, pandas automatically create label for each row which helps us to retreive the data at a faster rate. This is not a tabular data, its a single dimension series object. if nothing is pass for label, pandas consider index number as label. Here, you can see we didn't pass label, so the label of the Series are considered as the index numbers.

In [7]:
```
print(type(s))
```

```
<class 'pandas.core.series.Series'>
```

In [10]:
```
# Here, we are ourself passing the index value which means the label of the series
# the index that we passed.
s=pd.Series(data=l, index =['a','b','c','d','e'])
print(s)
```

```
a     First
b    Second
c     Third
d    Fourth
e     Fifth
dtype: object
```

In [11]:
```
s=pd.Series(data=[1,2,3,4,5], index =['a','b','c','d','e'], dtype='float64')
print(s)
```

```
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
dtype: float64
```

Here, you can see we pass the integer type element in data argument but in the output, it displayed as float type because we set the dtype argument as 'float64'.

In [12]:
```
import numpy as np
```

In [15]:
```
narr = np.array([1,2,3,4,5])
s=pd.Series(data=narr)  # Here, we are passing a numpy array in data argument.
print(s)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int32
```

In [16]:
```
s=pd.Series(2, index=['a','b','c','d','e'])
print(s)
```

```
a    2
b    2
c    2
d    2
e    2
dtype: int64
```

Here, if you noticed that we passed a constant value 2 as a data and passes some index. The constant 2 display multiple time when we print the series. It depends upon the number of index that we passed. Since we 5 index value that is why the constant 2 comes 5 times.

Note: In data argument, we can pass list, array, dictionary, constant.

In [19]:
```
s=pd.Series(data=[1,2,3,4,'a', 'b'])  # can store Heteregeneous data
s
```

Out[19]:
```
0    1
1    2
2    3
3    4
4    a
5    b
dtype: object
```

In [22]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product)
print(s1)
```

```
0    pencil
1    eraser
2       pen
3    marker
4     color
dtype: object
```

NOTE: The first letter of 'Series' word must be capital else it will raise an error.

In [58]:
```
# we can also change the index later after creating the series with the help of .i
s1.index=['a','b','c','d','e']
print(s1)
```

```
a    pencil
b    eraser
c       pen
d    marker
e     color
dtype: object
```

In [29]:
```
print(s1.index)  #This will also give the index or label of the series.
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

Accessing value from pandas Series

There are various method to access the elements from the series such as indexing, slicing, .loc and .iloc.

Indexing

In [51]:
```
# Consider the series.
print(s1)
```

```
105    pencil
106    eraser
107       pen
108    marker
109     color
dtype: object
```

In [33]:
```
# Now if we want to access the product 'marker' from the series we can use it inde
# list and tuple.
print(s1[3])  # The product marker is at 3rd index from vertical top. Indexing sta
```

```
marker
```

Since the index or label of the series is not numerical which means we can access the element from the label also. The corresponding label for the prodcut 'marker' is 'd'.

In [35]:
```
print(s1['d'])
```

```
marker
```

Note: If index or label of the series is not numerical then we have two option to access the elements, one is indexing and another is using label. But if the index or label of the series is numerical then we can have to use only label instead of index to access the element from the series.

In [54]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product, index=[105,106,107,108,109])
print(s1)
```

```
105    pencil
106    eraser
107       pen
108    marker
109     color
dtype: object
```

Here you can see the label of the series is numeric which means we cannot access the element from the series using the index. This means if you use s1[0] (0 is the first index correspond to product pencil), this will given error (try at your end) but if you use s[105] (105 is the label not the index) then it will give the corresponding product which is pencil.

In [39]:
```
s1[105]
```

Out[39]:
```
'pencil'
```

In [60]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product, index=['a','b','c','d','e'])
print(s1)
```

```
a    pencil
b    eraser
c       pen
d    marker
e     color
dtype: object
```

In [61]:
```
s1[-2]   # works with negative indexing as well
```

Out[61]:
```
'marker'
```

Note: The index or label must be in series else will raise an error.

In [67]:
```
s1[8]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[67], line 1
----> 1 s1[8]

File ~\anaconda3\lib\site-packages\pandas\core\series.py:978, in Series.__getitem__(self, key)
    975     key = unpack_1tuple(key)
    977 if is_integer(key) and self.index._should_fallback_to_positional:
--> 978     return self._values[key]
    980 elif key_is_scalar:
    981     return self._get_value(key)

IndexError: index 8 is out of bounds for axis 0 with size 5
```

Slicing

We use this method to access the range of the element from the series. This is similar to list slicing, you can refer that one.

In [63]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product, index=[105,106,107,108,109])
print(s1)
```

```
105    pencil
106    eraser
107       pen
108    marker
109     color
dtype: object
```

In [42]:
```
s1[1:3]
# while using slicing, index works which means it does not matter whether the label
# or not. this will give all the elements from the index 1 till the index 2 as inde
```

Out[42]:
```
106    eraser
107       pen
dtype: object
```

In [43]:
```
s1[:4]
```

Out[43]:
```
105    pencil
106    eraser
107       pen
108    marker
dtype: object
```

In [44]:
```
s1[2:]
```

Out[44]:
```
107       pen
108    marker
109     color
dtype: object
```

In [57]:
```
s1[-2:] #works on negative slicing as well like list
```

Out[57]:
```
108    marker
109     color
dtype: object
```

In [68]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product, index=['a','b','c','d','e'])
s1[-4:-1]
```

Out[68]:
```
b    eraser
c       pen
d    marker
dtype: object
```

In [65]:
```
s1[-4:-1:2]
```

Out[65]:
```
b    eraser
d    marker
dtype: object
```

In [66]:
```
s1[::2]
```

Out[66]:
```
a    pencil
c       pen
e     color
dtype: object
```

.loc method to access the element

This method takes the label value not the **index** to access the element.

In [72]:
```
s1.loc['b']
```

Out[72]:
```
'eraser'
```

In [73]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product, index=[105,106,107,108,109])
print(s1)
```

```
105    pencil
106    eraser
107       pen
108    marker
109     color
dtype: object
```

In [74]:
```
s1.loc[105]
```

Out[74]:
```
'pencil'
```

.iloc method to access the element.

This method takes the index not the **label** value to access the element.

In [75]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product, index=['a','b','c','d','e'])
s1.iloc[2]
```

Out[75]:
```
'pen'
```

In [81]:
```
s1.iloc[4]
```

Out[81]:
```
'color'
```

In [82]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product, index=[105,106,107,108,109])
s1.iloc[3]
```

Out[82]:
```
'marker'
```

Update value in the series.

In [84]:
```
product = ['pencil', 'eraser', 'pen', 'marker', 'color']
s1=pd.Series(data = product, index=['a','b','c','d','e'])
print(s1)
```

```
a    pencil
b    eraser
c       pen
d    marker
e     color
dtype: object
```

In [85]:
```
# Now, if we want to change the product 'pen' to 'inkpen' we can do it simply by ar
# listed above.
s1[2]='inkpen'
```

In [86]:
```
print(s1)
# See the product change from pen to inkpen.
```

```
a    pencil
b    eraser
c    inkpen
d    marker
e     color
dtype: object
```

In [88]:
```
s1.loc['d']='black marker'
```

In [89]:
```
print(s1)
# See the product change from marker to black marker.
```

```
a          pencil
b          eraser
c          inkpen
d    black marker
e           color
dtype: object
```

In [90]:
```
s1[2:4]=2
```

In [91]:
```
print(s1)
# See the product change from pen to inkpen.
```

```
a    pencil
b    eraser
c         2
d         2
e     color
dtype: object
```

In [ ]: