Pandas_Cheat_Sheet_Part_1_Series_and_Data_Frames

October 31, 2022

What is Pandas?

Pandas is an open source library in python for **data analysis**.

When/why to use Pandas?

1. It has built in tools for reading and writing data between many formats. It not only read csv or excel files but it can also connect to a larger scale database and write data to that database. It can also read HTML tables directly off a website.
2. It can intelligently grab data based on things like indexing, logic, conditional filtering and more.
3. It is only limited that how much RAM your computer has.

0.1 1. Series:

A series is a data structure / data type in Pandas that holds an array of information along with **named index.**

The named index differentiates Pandas from the simple Numpy array.

Formal definition: 1D ndimensional array with axis labels.

0.1.1 i. Generating Pandas Series object:

[2]:
```
import pandas as pd

mydata = ['USA', 'CANADA', 'MEXICO']    # Note that the data and index is given␣
↪in the form of list
myindex = [1776, 1778, 1779]
myseries = pd.Series(data = mydata, index = myindex)    # Note that Series has␣
↪capital S
```

[12]:
```
type(myseries)
```

[12]:
```
pandas.core.series.Series
```

[14]:
```
myseries
```

[14]:
```
1776       USA
1778    CANADA
1779    MEXICO
dtype: object
```

[15]:
```
myseries[1776]
```

[15]:
```
'USA'
```

[22]:
```
myindex = ['USA', 'CANADA', 'MEXICO']    # Note that the data and index is␣
↪given in the form of list
mydata = [1776, 1778, 1779]
myseries = pd.Series(data = mydata, index = myindex)    # Note that Series has␣
↪capital S
myseries['USA']
```

[22]:
```
1776
```

Named-index is of 2 types: 1. String 2. Integer

- If we have Named-index of type Integer, then we can access the data with that Integer-typed indices and with the normal integers as well e.g. if we set

  index = [1900, 2000, 2100]

  data = ['ABC', 'FGH', 'XYZ']

  newseries = pd.Series(data, index)

then we can access the data 'ABC' in two ways: newseries[0] (means with normal index like lists/np arrays) and newseries[1900] (means with named index).

- If we have Named-index of type String, then we can only access the data with that string-typed indices e.g. in above example, we can only access it as myseries['USA']. But if you want to access it with normal integer also, then you need to create the series from the dictionary. See in the following example:

[25]:
```
ages = {'Same': 5, 'Frank': 2, 'Spike': 6}
new_series = pd.Series(ages)
print(new_series)
```

```
Same     5
Frank    2
Spike    6
dtype: int64
```

[26]:
```
new_series[0]
```

[26]:
```
5
```

[27]:
```
new_series['Same']
```

[27]:
```
5
```

[28]:
```
new_series.keys()
```

[28]:
```
Index(['Same', 'Frank', 'Spike'], dtype='object')
```

[30]:
```
# TO GET ALL THE INDICES IN THE SERIES

new_series.keys()
```

[30]:
```
Index(['Same', 'Frank', 'Spike'], dtype='object')
```

[31]:
```
# As pandas is built on Numpy, as we can also apply broadcasting on Pandas␣
↪Series

new_series*2
```

[31]:
```
Same     10
Frank     4
Spike    12
dtype: int64
```

0.2 ii. Adding two series with some different values

[13]:
```
import numpy as np
series1_feed = {'Same': 5, 'Frank': 2, 'Spike': 6}
series2_feed = {'John': 4, 'Frank': 10, 'Ali': 3}

series1 = pd.Series(series1_feed, dtype = np.float64)
series2 = pd.Series(series2_feed, dtype = float)
print(series1)
print(series2)
```

```
Same     5.0
Frank    2.0
Spike    6.0
dtype: float64
John      4.0
Frank    10.0
Ali       3.0
dtype: float64
```

[14]:
```
series1 + series2
```

[14]:
```
Ali       NaN
Frank    12.0
John      NaN
Same      NaN
Spike     NaN
dtype: float64
```

[6]:
```
# to handle Nan

myser = series1.add(series2, fill_value = 0)
myser
```

[6]:
```
Ali       3.0
Frank    12.0
John      4.0
Same      5.0
Spike     6.0
dtype: float64
```

We can change the data type of a Series at two points:

1. While making the Series using **dtype =**: for example

- series1 = pd.Series(series1_feed, **dtype = np.float64**) (use np. if you want to use float64 or int64)
- series2 = pd.Series(series2_feed, **dtype = float**) (no need to use np if you want to use float or int simple)

2. After making the Series **seriesname.astype(typename) =**': for example

- myser = myser.astype('int')

[15]:
```
# to change data type  of a series: w
myser = myser.astype('int')
myser
```

[15]:
```
Ali       3
Frank    12
John      4
Same      5
Spike     6
dtype: int64
```

0.3 2. Dataframe:

A Dataframe is a table of columns and rows in Pandas that we can easily restructure and filter.

Formal Definition: **A group of pandas Series that share the same index.**

Each field of a dataframe is called a series (not column)

[18]:
```
import pandas as pd
import numpy as np

mydata = np.random.randint(0,101,(4,3))
mydata
```

[18]:
```
array([[88, 90,  7],
       [88, 65, 26],
       [81, 34,  5],
       [98,  0, 39]])
```

[25]:
```
myindex = ['USA', 'CANADA', 'UK', 'CHINA']
mydataframe = pd.DataFrame(data = mydata, index = myindex)
mydataframe
```

[25]:

	0	1	2
USA	88	90	7
CANADA	88	65	26
UK	81	34	5
CHINA	98	0	39

[26]:
```
# to add column names

mycolumns = ['Jan', 'Feb', 'Mar']
mydataframe = pd.DataFrame(data = mydata, index = myindex, columns = mycolumns)
mydataframe
```

[26]:

	Jan	Feb	Mar
USA	88	90	7
CANADA	88	65	26
UK	81	34	5
CHINA	98	0	39

[27]:
```
# to get the details of the data frame

mydataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, USA to CHINA
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Jan     4 non-null      int64
 1   Feb     4 non-null      int64
 2   Mar     4 non-null      int64
dtypes: int64(3)
memory usage: 300.0+ bytes
```