Introducing Pandas Objects

Pandas ၏ fundamental data structure (၃)မျိုးဖြစ်သည့် (၁) `Series` , (၂) `DataFrame` နှင့် (၃) `Index` တို့ အကြောင်းကို ရှင်းပြထားသည်။ Dictionary များ ၊ list များ ၊ NumPy array များ နှင့် `DataFrame` များ၏ indexing သဘောတရားကို နားလည် သဘောပေါက်ရန် အရေးကြီးသည်။

ပထမဦးစွာ NumPy နှင့် Pandas ကို import လုပ်သည်။

In [1]:
```
import numpy as np
import pandas as pd
```

Pandas တွင် `series` ဆိုသည်မှာ one-dimensional array တစ်ခု ဖြစ်သည်။ တစ်နည်းအားဖြင့် data များကို index လုပ်ထားသည့် one-dimensional array တစ်ခု ဖြစ်သည်။ Data များကို index လုပ်ထားသည် ဆိုသည်မှာ ဒေတာများ သိမ်းဆည်းထားသည့်နေရာကို အလွယ်တကူ ရှာဖွေနိုင်ရန် အမှတ်အသားပြု(index လုပ်ထား)လုပ်ထားခြင်း ဖြစ်သည်။

list တစ်ခု သို့မဟုတ် array တစ်ခုမှ series တစ်ခု ပြုလုပ်နိုင်သည်။ `pd.Series()` ဖြင့် series တစ်ခု ပြုလုပ်ပုံကို အောက်တွင် ဖော်ပြထားသည်။

In [2]:
```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
data
```

Out[2]:
```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

Pandas series တွင် `values` နှင့် `index` ဟုသည့် attribute နှစ်မျိုးဖြစ်သည်။ `values` သည် ဒေတာများ၏ တန်ဖိုးများဖြစ်သည်။ `index` သည် ထိုဒေတာတန်ဖိုးများ သိမ်းဆည်းထားသည့်နေရာကို အမှတ်အသားပြုထားသည့်နေရာ ဖြစ်သည်။

ထို့ကြောင့် `Series` တွင် sequence of values တစ်ခု နှင့် sequence of indices တစ်ခု ရှိသည်။ `values` များသာပါဝင်သည့် ကိန်းအစဉ်(sequence of values တစ်ခု)သည် NumPy array ဖြစ်သည်။

data.values ဖြင့် Pandas series အတွင်းမှ ဒေတာများကို access လုပ်နိုင်သည်။ ကြည့်ရှုနိုင်သည်။

In [3]:
```
data.values
```

Out[3]:
```
array([0.25, 0.5 , 0.75, 1.  ])
```

`index` သည် array-like object of type တစ်ခုဖြစ်သည်။ `pd.Index`

In [4]:
```
data.index
```

Out[4]:
```
RangeIndex(start=0, stop=4, step=1)
```

NumPy array များကဲ့သို့ပင် Pandas series အတွင်းမှ ဒေတာများကို `[ ]` Python square-bracket notation ဖြင့် access လုပ်နိုင်သည်။

`data[1]` ဖြင့် data ဟုခေါ်သည့် Pandas series အတွင်းမှ ဒုတိယမြောက်ဒေတာ ကို access လုပ်သည်။

ပထမ မြောက် ဒေတာသည် index [0] ဖြစ်သည်။

In [5]:
```
data[1]
```

Out[5]:
```
0.5
```

In [6]:
```
data[1:3]
```

Out[6]:
```
1    0.50
2    0.75
dtype: float64
```

Pandas `Series` သည် one-dimensional NumPy array ထက် ပို၍ ယေဘုယျဆန်သည်။ ပို၍ ပြင်လွယ်ပြောင်းလွယ်ဖြစ်သည်။ (more general and flexible than the one-dimensional NumPy array)

`Series` as generalized NumPy array

Series object မှ one-dimensional NumPy array အဖြစ်သို့ လည်းကောင်း one-dimensional NumPy array အဖြစ်မှ Series object အဖြစ်သို့ အပြန်အလှန် ပြောင်းလဲ နိုင်သည်။ (basically interchangeable)

ထူးခြားသည့် ကွာခြားချက်မှာ index များ ရှိခြင်းဖြစ်သည်။ ဒေတာများ၏ တန်ဖိုးများ(values)ကို access လုပ်သည့်အခါ Numpy Array ၏ integer index များကို *implicitly defined* ပြုလုပ်ထားပြီး `Series` ၏ integer index များကို *explicitly defined* ပြုလုပ်ထားသည်။

Explicit indexing ဖြစ်ခြင်းကြောင့် `Series` object များတွင် အားသာချက်များ (additional capabilities) ရှိသည်။ ဥပမာ - `Series` object များတွင် index များသည် integer များ ဖြစ်ရန် မလိုပါ။ မည်သည့် အမျိုးအစားမဆို (values of any desired type) index များ အဖြစ် သတ်မှတ်နိုင်သည်။

အောက်တွင် ဥပမာအဖြစ် string များကို index အဖြစ် သုံးနိုင်သည်။

In [7]:
```
data = pd.Series([0.25, 0.5, 0.75, 1.0],
                 index=['a', 'b', 'c', 'd'])
data
```

Out[7]:
```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

Series အတွင်းမှ item များကို အောက်တွင် ပြထားသည့် အတိုင်း access လုပ်နိုင်သည်။

In [8]:
```
data['b']
```

Out[8]: 0.5

We can even use non-contiguous or non-sequential indices:

In [9]:
```
data = pd.Series([0.25, 0.5, 0.75, 1.0],
                 index=[2, 5, 3, 7])
data
```

Out[9]:
```
2    0.25
5    0.50
3    0.75
7    1.00
dtype: float64
```

In [10]:
```
data[5]
```

Out[10]: 0.5

Series as specialized dictionary

Pandas `Series` များကို Python dictionary တစ်မျိုး အဖြစ်လည်း ယူဆနိုင်သည်။

Python dictionary ဆိုသည်မှာ arbitrary keys များ နှင့် a set of arbitrary values များ ယှဉ်တွဲထားသည့်(map လုပ်ထားသည့်) data structure တစ်မျိုးဖြစ်သည်။

`Series` သည် structure which maps typed keys နှင့် set of typed values ယှဉ်တွဲထားသည့်(map လုပ်ထားသည့်) data structure တစ်မျိုးဖြစ်သည်။

`NumPy array` တွင် `Python list` ထက် တချို့သော လုပ်ဆောင်မှုများ (certain operations)တွင် ပိုကောင်းသည့်အချက်(more efficient)များ ရှိသည်။

`Series` တွင် `Python list` ထက် တချို့သော လုပ်ဆောင်မှုများ (certain operations)တွင် ပိုကောင်းသည့်အချက်(more efficient)များ ရှိသည်။

`Python dictionary` မှ `Series` object အဖြစ် တိုက်ရိုက်ပြောင်းလဲနိုင်သည်။

In [11]:
```
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}
population = pd.Series(population_dict)
population
```

Out[11]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

ပုံမှန်အားဖြင့်(by default) sorted keys များကို index အဖြစ် သတ်မှတ်၍ `Series` များကို တည်ဆောက်နိုင်ပါသည်။ အောက်တွင် typical dictionary-style item ကို access လုပ်သည့် နည်းကို ဖော်ပြထားသည်။

In [12]:
```
population['California']
```

Out[12]:
```
38332521
```

Python dictionary နှင့် မတူသည့်အချက်မှာ `Series` များကို slicing စသည့် array-style operations များဖြင့် access လုပ်နိုင်သည်။

In [13]:
```
population['California':'Illinois']
```

Out[13]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

Constructing Series objects

အထက်တွင် Pandas `Series` များ ပြုလုပ်နည်းကို ဖော်ပြပြီး ဖြစ်သည်။ ကူးယူခြင်း ဒေတာဖိုင်မှ ဖတ်ယူခြင်း မပြုလုပ်ဘဲ(from scratch)

ထိုနည်းများ အာလုံး၏ ပုံစံမှာ အောက်ပါအတိုင်းဖြစ်သည်။

```
>>> pd.Series(data, index=index)
```

`index` သည် optional argument တစ်ခုဖြစ်သည်။ မဖြစ် မနေ ထည့်ပေးရမည့် argument တစ်ခု မဟုတ်ပါ။

`data` သည် သိမ်းဆည်းလိုသည့်အချက်များ အနက်မှ တစ်ခုခု ဖြစ်နိုင်သည်။ (one of many entities)

ဥပမာ `data` သည် list သို့မဟုတ် NumPy array တစ်ခု ဖြစ်နိုင်သည်။ `index` သည် ပုံမှန်အားဖြင့်(by default) integer sequence တစ်ဖြစ်နိုင်သည်။

In [14]:
```
pd.Series([2, 4, 6])
```

Out[14]:
```
0    2
1    4
2    6
dtype: int64
```

`data` သည် scalar ဖြစ်နိုင်သည်။

`data` can be a scalar, which is repeated to fill the specified index:

In [15]:
```
pd.Series(5, index=[100, 200, 300])
```

Out[15]:
```
100    5
200    5
300    5
dtype: int64
```

`data` သည် dictionary တစ်ခုလည်း ဖြစ်နိုင်သည်။ ထိုအခါ index သည် ပုံမှန်အားဖြင့်(by default) sorted dictionary keys သည်။

`data` can be a dictionary, in which `index` defaults to the sorted dictionary keys:

In [16]:
```
pd.Series({2:'a', 1:'b', 3:'c'})
```

Out[16]:
```
2    a
1    b
3    c
dtype: object
```

အထက်ပါ ဥပမာ (၂)မျိုးလုံးတွင် index များကို မိမိလိုသလို တိတိကျကျ(explicitly) သတ်မှတ်ပေးနိုင်သည်။

In each case, the index can be explicitly set if a different result is preferred:

In [17]:
```
pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2])
```

Out[17]:
```
3    c
2    a
dtype: object
```

Notice that in this case, the `Series` is populated only with the explicitly identified keys.

The Pandas DataFrame Object

Pandas ၏ fundamental structure နောက်တစ်မျိုးမှာ `DataFrame` ဖြစ်သည်။ အထက်တွင် ရှင်းပြခဲ့ပြီးသည့် `Series` ကဲ့သို့ပင် `DataFrame` ကို generalize လုပ်ထားသည့် NumPy array တစ်ခု သို့မဟုတ် specialize လုပ်ထားသည့် Python dictionary အဖြစ် သတ်မှတ်ယူဆနိုင်သည်။

DataFrame as a generalized NumPy array (Generalize လုပ်ထားသည့် NumPy array)

`Series` တစ်ခု သည် analog of a one-dimensional array with flexible indices ဖြစ်သည်။ `DataFrame` တစ်ခု သည် analog of a two-dimensional array တစ်ခု ဖြစ်ပြီး flexible row indices နှင့် flexible column names ဟု၍ နှစ်မျိုးစလုံး ပါရှိသည်။

Just as you might think of a two-dimensional array as an ordered sequence of aligned one-dimensional columns, you can think of a `DataFrame` as a sequence of aligned `Series` objects.

"aligned" ဆိုသည်မှာ တူညီသည့် index များကို ဆိုလိုသည်။ (they share the same index.) အမေရိကမှ ပြည်နယ်(၅)ခုကို `Series` အသစ်တစ်ခု တည်ဆောက်ပြီး ဥပမာ အဖြစ် လေ့လာကြရအောင်

In [18]:
```
area_dict = {'California': 423967, 'Texas': 695662, 'New York': 1412
97,
             'Florida': 170312, 'Illinois': 149995}
area = pd.Series(area_dict)
area
```

Out[18]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
dtype: int64
```

ထိုပြည်နယ်များတွင် နေထိုင်ကြသည့် လူဦးရေကို `population` Series အဖြစ် တည်ဆောက်သည်။ `area` နှင့် `population` နှစ်မျိုးပါသည့် dictionary တစ်ခု(single two-dimensional object dictionary) မှ DataFrame တည်ဆောက်သည်။

`{'population': population, 'area': area}` သည် single two-dimensional object dictionary ဖြစ်သည်။ `pd.DataFrame()` ၏ () ထဲတွင် dictionary ထည့်၍ DataFrame တည်ဆောက်သည်။

In [19]:
```
states = pd.DataFrame({'population': population,
                       'area': area})
states
```

Out[19]:

	population	area
California	38332521	423967
Texas	26448193	695662
New York	19651127	141297
Florida	19552860	170312
Illinois	12882135	149995

`Series` object ကဲ့သို့ပင် `DataFrame` တွင် index labels များကို access လုပ်ရန် `index` attribute ရှိသည်။

အောက်တွင် `states.index` ဖြင့် index labels များကို access လုပ်ပုံကို ဖော်ပြထားသည်။

In [20]:
```
states.index
```

Out[20]:
```
Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'],
dtype='object')
```

Additionally, the `DataFrame` has a `columns` attribute, which is an `Index` object holding the column labels:

In [21]:
```
states.columns
```

Out[21]:
```
Index(['population', 'area'], dtype='object')
```

`DataFrame` သည် generalization လုပ်ထားသည့် two-dimensional NumPy array တစ်ခုဖြစ်သည်။ Row နှင့် column နှစ်ခုစလုံးတွင် data များကို access လုပ်ရန် generalized index များရှိသည်။

DataFrame as specialized dictionary (specialize လုပ်ထားသည့် Python dictionary)

`DataFrame` သည် specialize လုပ်ထားသည့် Python dictionary တစ်ခုဖြစ်သည်။ dictionary ဆိုသည်မှာ key များနှင့် value များကို ယှဉ်တွဲ (maps a key to a value)ထားသည့် ဒေတာ သိမ်းဆည်းပုံ တစ်မျိုးဖြစ်သည်။ `DataFrame` တွင် column name နှင့် `Series` of column data များ ယှဉ်တွဲ(map)ထားသည်။

ဥပမာ `'area'` attribute ကို မေးလျှင် area ကော်လံအောက်ရှိ ဒေတာများအားလုံးကို `Series` တစ်ခု အဖြစ် ဖော်ပြပေးလိမ့်မည်။

In [22]:
```
states['area']
```

Out[22]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: area, dtype: int64
```

two-dimesnional NumPy array တွင် `data[0]` သည် ပထမဆုံးအတန်း(first *row*) ဖြစ်သည်။ `DataFrame` တွင် `data['col0']` သည် ပထမဆုံးကော်လံ ဖြစ်သည်။ ထို့ကြောင့် `DataFrame` ကို generalized array ဟု သတ်မှတ်မည့် အစား generalized dictionary အဖြစ် သတ်မှတ်သည်။

Constructing DataFrame objects

Pandas တွင် `DataFrame` တည်ဆောက်သည့် နည်းများစွာ ရှိသည်။ ပထမဦးစွာ Series object တစ်ခုဖြင့် `DataFrame` တည်ဆောက်ပုံကို ဖော်ပြမည်။

From a single Series object

`DataFrame` သည် `Series` objects စုဝေးနေသည့် အရာတစ်ခုလည်း ဖြစ်သည်။ `Series` တစ်ခုဖြင့် ကော်လံ တစ်ခုသာ ပါသည့် `DataFrame` တစ်ခုတည်ဆောက်နိုင်ပြီး `Series` များစွာဖြင့် ကော်လံ များစွာ ပါဝင်သည့် `DataFrame` တစ်ခုတည်ဆောက်နိုင်သည်။

In [23]:
```
pd.DataFrame(population, columns=['population'])
```

Out[23]:

	population
California	38332521
Texas	26448193
New York	19651127
Florida	19552860
Illinois	12882135

From a list of dicts

dictionary များကိုလည်း `DataFrame` အဖြစ် ပြုလုပ်နိုင်သည်။

In [24]:
```
data = [{'a': i, 'b': 2 * i}
        for i in range(3)]
pd.DataFrame(data)
```

Out[24]:

	a	b
0	0	0
1	1	2
2	2	4

`DataFrame` ပြုလုပ်နေချိန်တွင် dictionary တစ်ခုတွင် တချို့သော key များ ပျောက်နေလျှင်သော်လည်းကောင်း၊ မရှိလျှင်သော်လည်းကောင်း(missing)၊ ဖတ်မရလျှင်သော်လည်းကောင်း Pandas သည် `NaN` (i.e., "not a number") values အဖြစ် ဖြည့်စွက်ထည့်သွင်းပေးသည်။

In [25]:
```
pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}])
```

Out[25]:

	a	b	c
0	1.0	2	NaN
1	NaN	3	4.0

From a dictionary of Series objects

Dictionary of `Series` objects တစ်ခုမှလည်း `DataFrame` တည်ဆောက်နိုင်သည်။ a `DataFrame` can be constructed from a dictionary of `Series` objects as well:

In [26]:
```
pd.DataFrame({'population': population,
              'area': area})
```

Out[26]:

	population	area
California	38332521	423967
Texas	26448193	695662
New York	19651127	141297
Florida	19552860	170312
Illinois	12882135	149995

From a two-dimensional NumPy array

two-dimensional NumPy array တစ်ခုမှလည်း မိမိအလိုရှိသည့် ကော်လံနာမည်(column name)နှင့် index နာမည်ပေး၍ `DataFrame` တည်ဆောက်နိုင်သည်။ အကယ်၍ `DataFrame` တည်ဆောက်နေချိန်တွင် မိမိအလိုရှိသည့် ကော်လံနာမည်(column name)နှင့် index နာမည် မပေးပါက integer index အဖြစ် ထည့်ပေးလိမ့်မည်။

In [27]:
```
pd.DataFrame(np.random.rand(3, 2),
             columns=['foo', 'bar'],
             index=['a', 'b', 'c'])
```

Out[27]:

	foo	bar
a	0.304631	0.055655
b	0.610860	0.208457
c	0.309992	0.902059

From a NumPy structured array

structured array တစ်ခုခုကို Pandas `DataFrame` အဖြစ်လည်း တည်ဆောက်နိုင်သည်။ [Structured Data: NumPy's Structured Arrays] များအကြောင်းကို တခြား အခန်းတွင် ရှင်းပြထားပြီး ဖြစ်သည်။

np.zeros() ဖြင့် numpy.ndarray တစ်ခုတည်ဆောက်သည်။

In [28]:
```
A = np.zeros(3, dtype=[('A', 'i8'), ('B', 'f8')])
A
```

Out[28]:
```
array([(0, 0.), (0, 0.), (0, 0.)], dtype=[('A', '<i8'), ('B', '<f8
')])
```

numpy.ndarray ကို `pd.DataFrame(A)` ဖြင့် DataFrame တစ်ခုတည်ဆောက်သည်။

In [29]:
```
pd.DataFrame(A)
```

Out[29]:

	A	B
0	0	0.0
1	0	0.0
2	0	0.0

The Pandas Index Object

`Series` and `DataFrame` objects နှစ်မျိုးလုံးတွင် ရှိနေသည့် ဒေတာများကို reference လုပ်ရန်နှင့် လိုသလို ပြောင်းလဲနိုင်ရန်(modify လုပ်ရန်) explicit *index* များ ရှိသည်။ `Index` object များသည် ပြင်၍ ပြောင်း၍ မရနိုင်သည့် *immutable array* ဖြစ်သည်။ *ordered set* (technically a multi-set) လည်းဖြစ်သည် ။ `Index` objects များတွင် ကိန်းတစ်ခု သို့မဟုတ် နာမည်တစ်ခုသည် ထပ်ခါထပ်ခါ ပါဝင်နေနိုင်သည်။ (repeated values)

ထိုသို့ `Index` objects ကိန်းတစ်ခု ထပ်ခါထပ်ခါ ပါရှိနေနိုင်ခြင်းကြောင့် ပိုစိတ်ဝင်စားစရာကောင်းလှသည်။ အောက်တွင် ဥပမာအဖြစ် list of integers မှ `Index` အဖြစ် ပြုလုပ်ပုံကို လေ့လာကြပါစို့

In [30]:
```
ind = pd.Index([2, 3, 5, 7, 11])
ind
```

Out[30]:
```
Int64Index([2, 3, 5, 7, 11], dtype='int64')
```

Index as immutable array (Index သည် ပြင်၍ ပြောင်း၍ မရနိုင်သည့် *immutable array* ဖြစ်သည်)

`Index` တွင် array တစ်ခုကဲ့သို့ ဆောင်ရွက်ပေးနိုင်သည့် ကိစ္စများစွာရှိသည်။ ဥပမာ value များကို retrieve လုပ်ရန် သို့မဟုတ် slices လုပ်ရန် standard Python indexing notation ကို အသုံးပြုနိုင်သည်။

In [31]:
```
ind[1]
```

Out[31]:
```
3
```

In [32]:
```
ind[::2]
```

Out[32]:
```
Int64Index([2, 5, 11], dtype='int64')
```

`Index` object တွင် NumPy array အလုပ်လုပ်ပုံနှင့် တူညီသည့် attribute များစွာ ရှိသည်။

In [33]:
```
print(ind.size, ind.shape, ind.ndim, ind.dtype)
```

```
5 (5,) 1 int64
```

`Index` objects နှင့် NumPy array တို့၏ အဓိက ကွာခြားချက်မှာ index များသည် ပြင်၍ ပြောင်း၍ မရနိုင်သည့် immutable array ဖြစ်သည်။ ပြင်၍ ပြောင်း၍ မရဟုဆိုရာတွင် လုပ်နေကြနည်းများနှင့် ပြုလုပ်၍ မရနိုင်ခြင်းကို ဆိုလိုသည်။ (they cannot be modified via the normal means)

အောက်တွင် `Index` အတွင်းမှ တစ်ခုကို ပြောင်းလဲသည့် အခါ `TypeError: Index does not support mutable operations` ဖြစ်ပေါ်ပုံကို ဖော်ပြထားသည်။

In [34]:
```
ind[1] = 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-906a9fa1424c> in <module>
----> 1 ind[1] = 0

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\indexes\base.py in __setitem__(self, key, value)
   3936 
   3937     def __setitem__(self, key, value):
-> 3938         raise TypeError("Index does not support mutable operations")
   3939 
   3940     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

ဤကဲ့သို့ ပြင်၍ ပြောင်း၍ မရခြင်း(immutability) ကြောင့် `DataFrame` များ နှင့် array များ အကြားတွင် index များကို အတူတကွ ရောနှော သုံးစွဲသည့် လုံခြုံစိတ်ချရသည်။ အမှုမဲ့ အမှတ်မဲ့ index များကို မရည်ရွယ် ဘဲ ပြောင်းလဲ မိခြင်းမျိုး မဖြစ်နိုင်တော့ပေ။

Index as ordered set

Pandas object များသည် datasetများ ပေါင်းစပ်ခြင်း(join)၊ Python's built-in `set` data structure များဖြစ်သည့် unions လုပ်ခြင်း၊ intersections လုပ်ခြင်း differences လုပ်ခြင်း စသည့် operation များ ပြုလုပ်နိုင်အောင် ဒီဇိုင်းလုပ်ထားသည်။

In [35]:
```
indA = pd.Index([1, 3, 5, 7, 9])
indB = pd.Index([2, 3, 5, 7, 11])
```

indA နှင့် indB ကို intersection လုပ်သည်။

In [36]:
```
indA & indB  # intersection
```

Out[36]:
```
Int64Index([3, 5, 7], dtype='int64')
```

indA နှင့် indB ကို union လုပ်သည်။

In [37]:
```
indA | indB  # union
```

Out[37]:
```
Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int64')
```

indA နှင့် indB ကို symmetric difference လုပ်သည်။

In [38]:
```
indA ^ indB  # symmetric difference
```

Out[38]:
```
Int64Index([1, 2, 9, 11], dtype='int64')
```

object methods ကို သုံး၍လည်း ပြုလုပ်နိုင်သည်။ ဥပမာ `indA.intersection(indB)` .

In [39]:
```
indA.intersection(indB)
```

Out[39]:
```
Int64Index([3, 5, 7], dtype='int64')
```

ကောင်းထက်ညွန့်