MyEduSolve

Basics of Data Manipulation With Python

By Sintaks Group

This project is created by **Sintaks Group**, with the following members:

- Galang Setia Nugroho
- Khalishah Fiddina
- Donny Tambunan
- Tifani Amalina
- Hasballah Askar
- Muhammad Ilham Hakiqi

Table of Contents

01. Series

02. loc and iloc

03. DataFrame

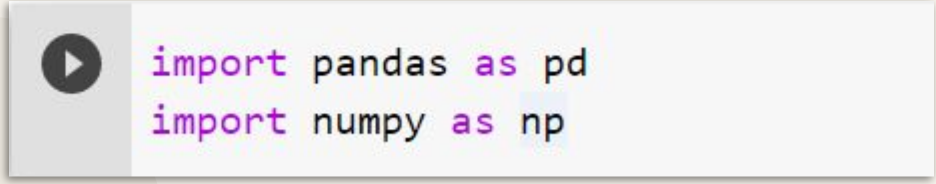
Data Manipulation

In this context, Data manipulation is not about engineering data or making data inconsistent with its original value.

Instead, Data Manipulation here is used to make it easier for machines to analyze data. It's the process of changing or altering data in order to make it more readable and structured or organized.

Before we get into it.

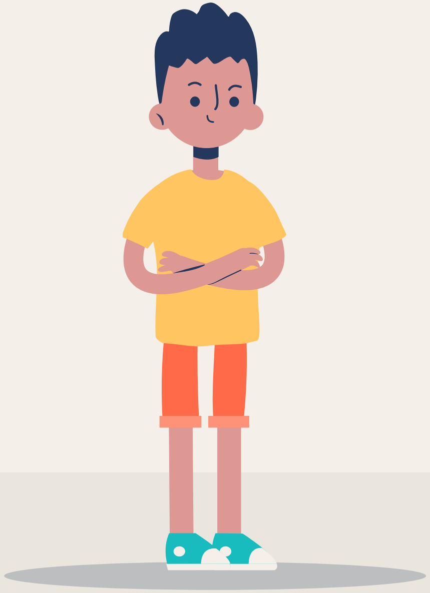
We need to import the required libraries beforehand



```
import pandas as pd
import numpy as np
```

Note:

- **pandas**, used for data manipulation and analysis
- **numpy**, used for working with arrays



01

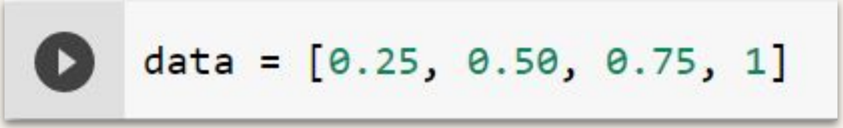
Series

Pandas Series

Series

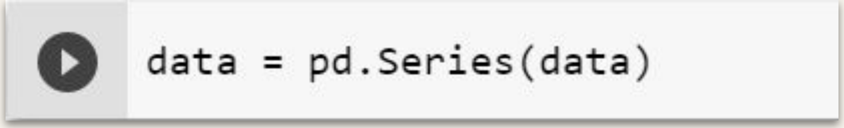
Pandas has two object, i.e. Series and DataFrame. Series is a one-dimensional labeled data structure. It's like a column but with no name. The axis labels are collectively called index.

We have a list called ‘data’



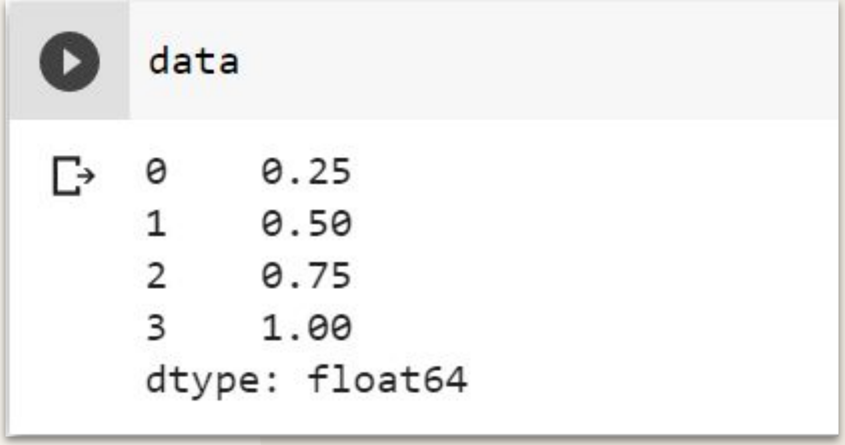
```
data = [0.25, 0.50, 0.75, 1]
```

Convert the list to pandas series



```
data = pd.Series(data)
```

Result of the conversion

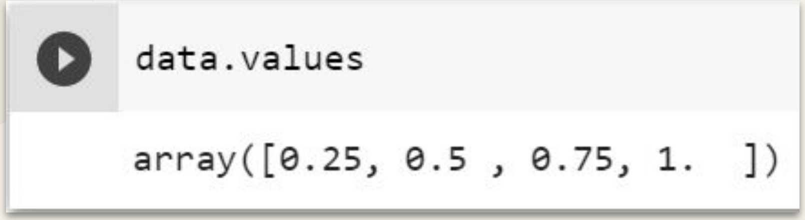


```
data
```

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

Series

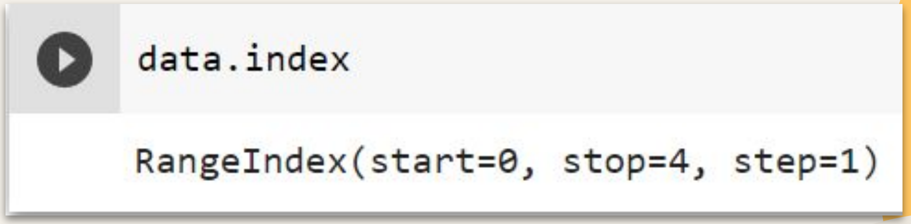
To return the series as ndarray



```
data.values
```

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

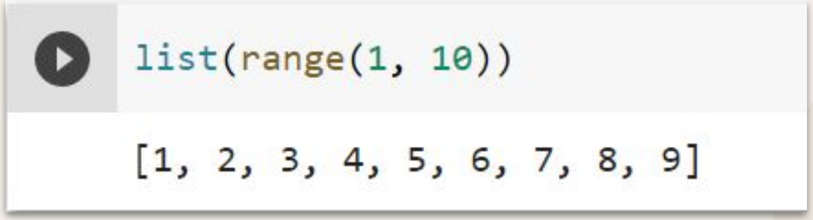
Display the range of pandas series index



```
data.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

Pandas series index is a range, where the starting point is inclusive and the stopping point is exclusive



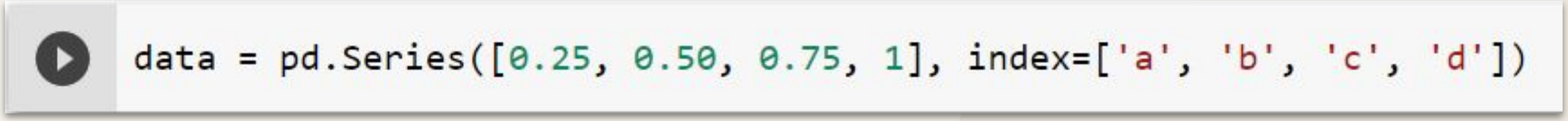
```
list(range(1, 10))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Series · Index

Implicit Index is the default index. But we can define a different index, it’s called **Explicit Index** i.e. the defined index. Defining an index, the number of index must be equal to the number of data.

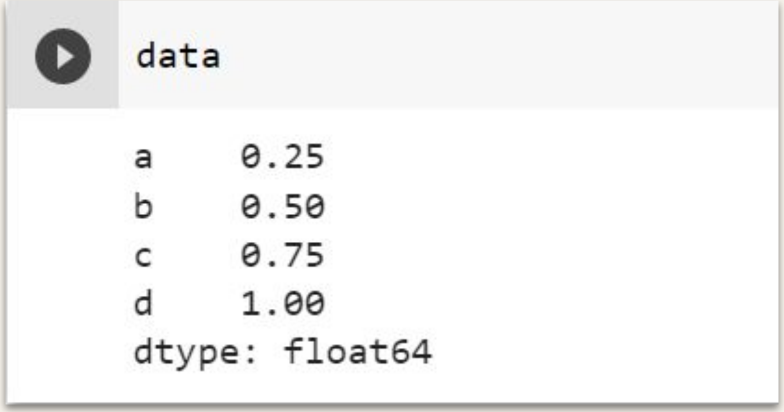
To define Explicit Index, take a look at the example below.



```
data = pd.Series([0.25, 0.50, 0.75, 1], index=['a', 'b', 'c', 'd'])
```

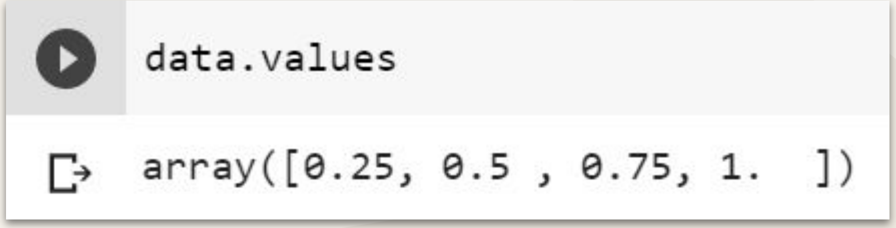
Series · Index

As you can see here, the indices have changed.



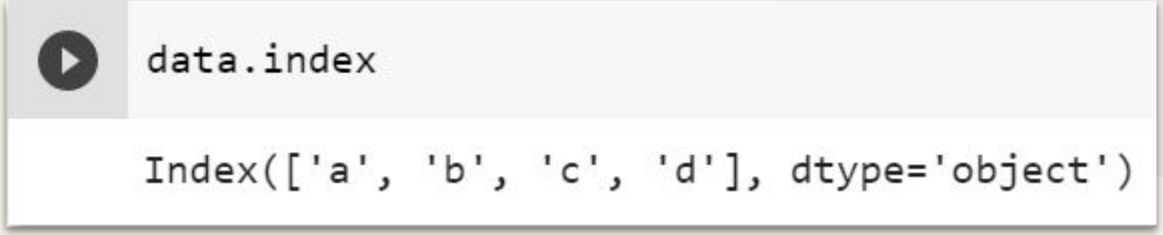
```
data
```

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```



```
data.values
```

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



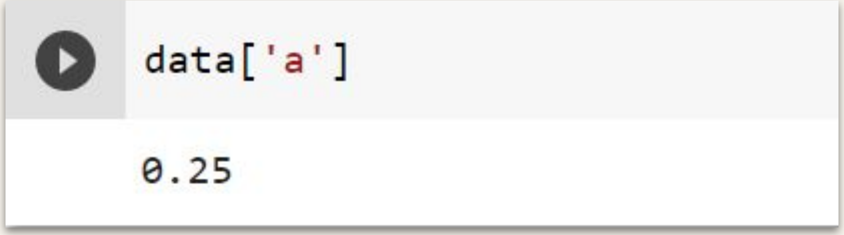
```
data.index
```

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

Series · Index

Calling a value at a specific index or also known as Data Selection

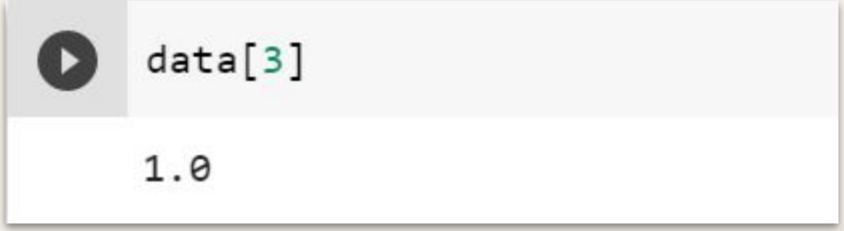
Using Explicit Index



```
data['a']
```

```
0.25
```

Using Implicit Index



```
data[3]
```

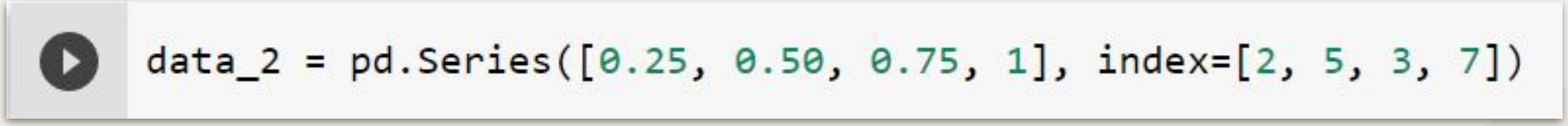
```
1.0
```

Although we have defined Explicit Index, we can still call it by using Implicit Index

Series · Index

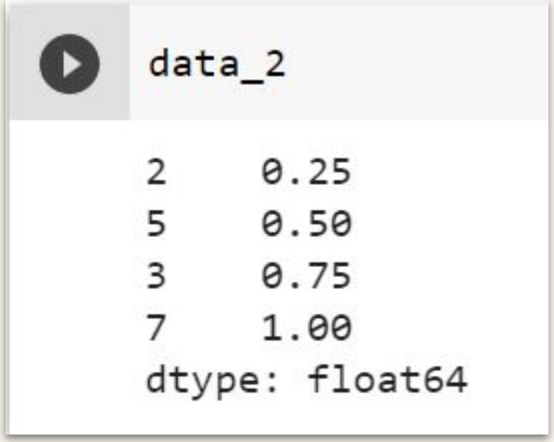
If there is the same implicit index and explicit index, it will depend **only on the explicit index** when it’s being called.

For example below



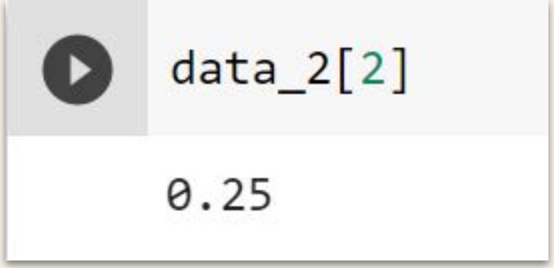
```
data_2 = pd.Series([0.25, 0.50, 0.75, 1], index=[2, 5, 3, 7])
```

Series · Index



```
data_2
```

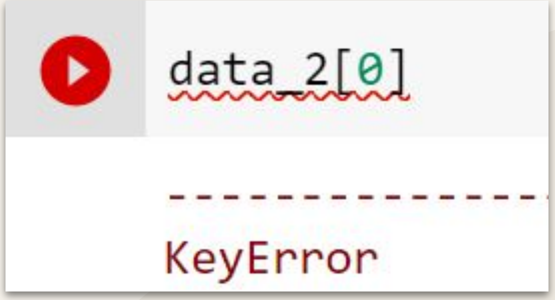
```
2    0.25
5    0.50
3    0.75
7    1.00
dtype: float64
```



```
data_2[2]
```

```
0.25
```

It returns the value of the specified Explicit Index



```
data_2[0]
```

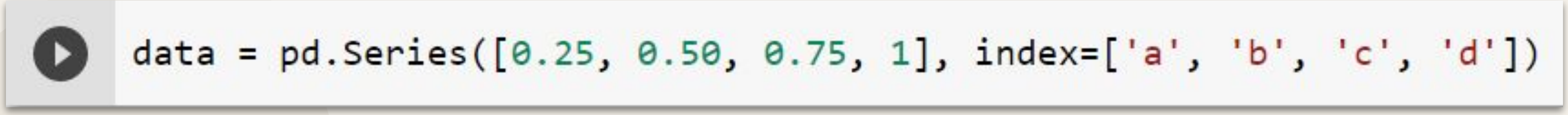
```
----------------
KeyError
```

It returns KeyError Exception because Explicit Index 0 doesn’t exist

Series · Index

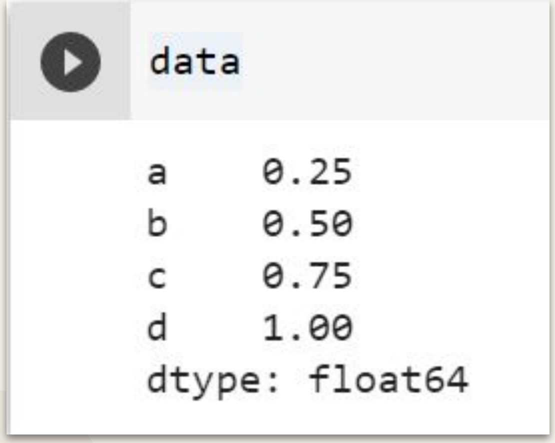
In this section we will try to perform Data Slicing.

See the example below



```
data = pd.Series([0.25, 0.50, 0.75, 1], index=['a', 'b', 'c', 'd'])
```

Series · Index

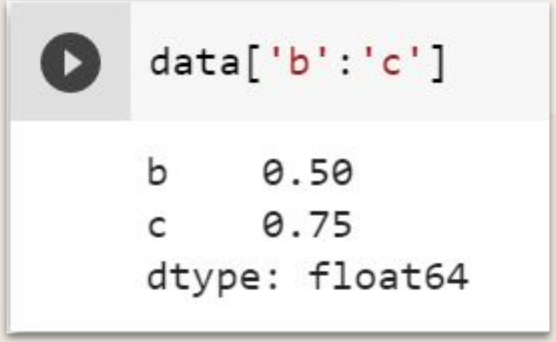


```
data
```

```
a      0.25
b      0.50
c      0.75
d      1.00
dtype: float64
```

Explicit Index

Call data from index b to c

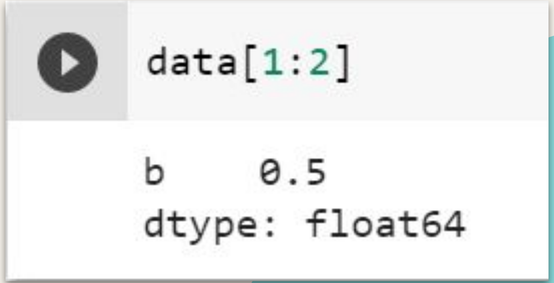


```
data['b':'c']
```

```
b      0.50
c      0.75
dtype: float64
```

Implicit Index

Call data from index 1 to 2



```
data[1:2]
```

```
b      0.5
dtype: float64
```

If we use implicit index, value at the stop point will not be returned because implicit index is a range.

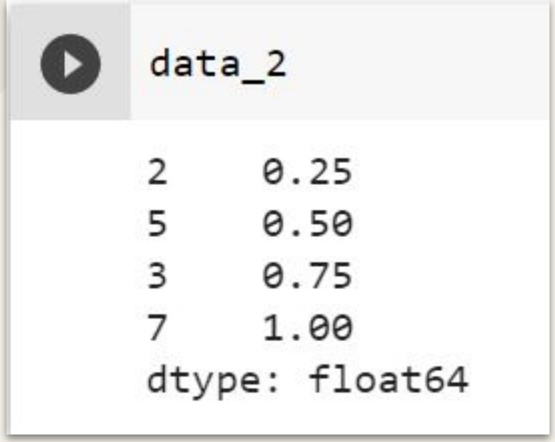
02

loc and iloc

Pandas Methods

If there is the same implicit index and explicit index, there will inconsistency.

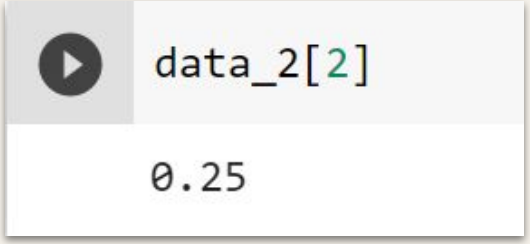
For example, we have data_2 as shown below



```
data_2
```

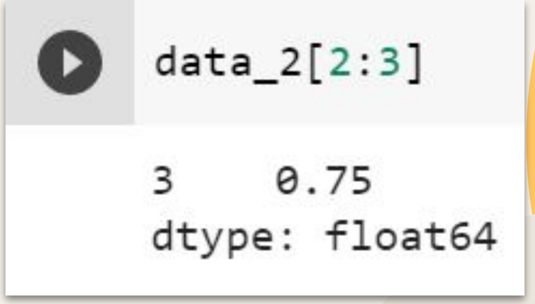
```
2    0.25
5    0.50
3    0.75
7    1.00
dtype: float64
```

When **selecting** data, **explicit index** is used



```
data_2[2]
```

```
0.25
```



```
data_2[2:3]
```

```
3    0.75
dtype: float64
```

When **slicing** data, **implicit index** is used instead

To overcome this inconsistency, we will use the **loc** and **iloc** methods.

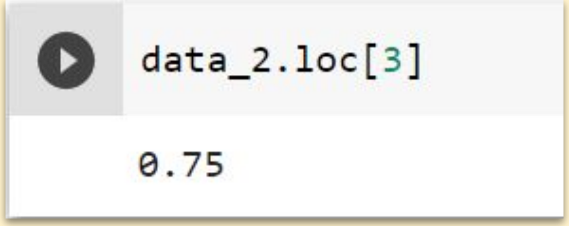
What are loc and iloc methods?

They’re methods in Pandas, **loc** is used to call **explicit index** meanwhile **implicit index** is called using **iloc**.

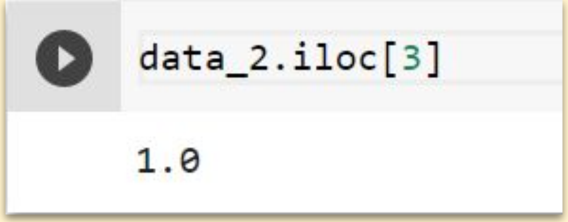
loc

iloc

selecting data

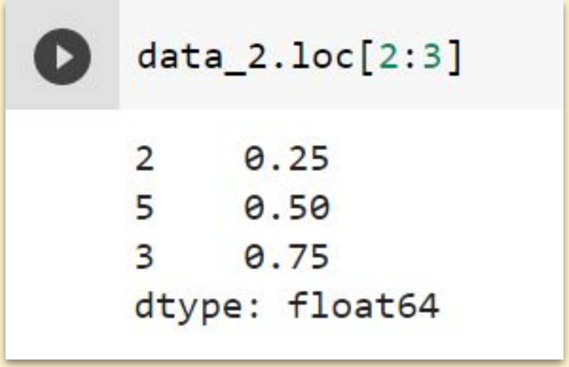


```
data_2.loc[3]

0.75
```

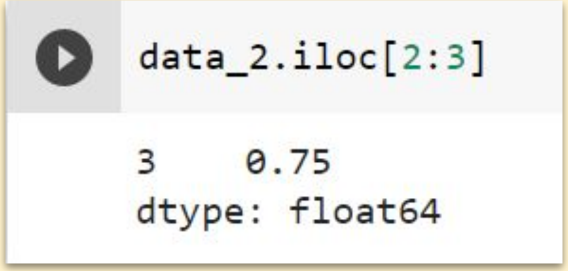


```
data_2.iloc[3]

1.0
```

slicing data



```
data_2.loc[2:3]

2     0.25
5     0.50
3     0.75
dtype: float64
```



```
data_2.iloc[2:3]

3     0.75
dtype: float64
```

03

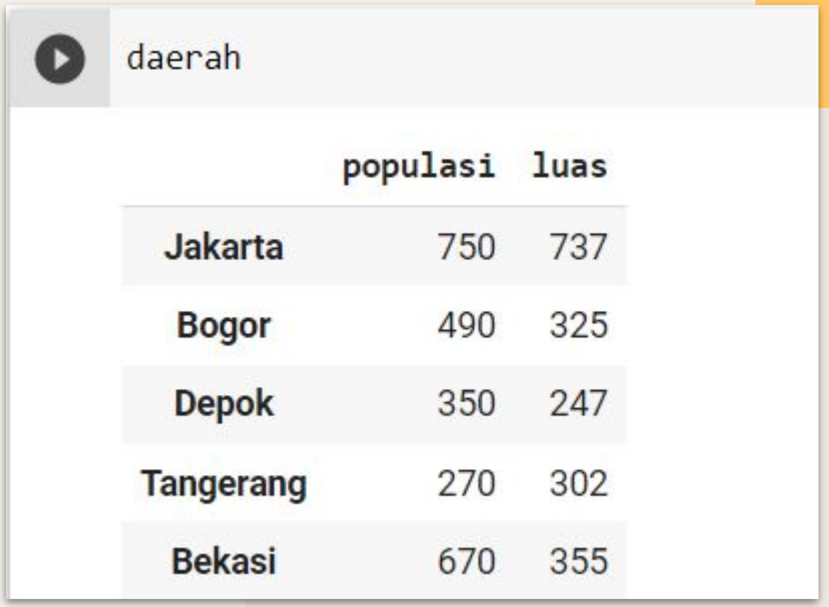
DataFrame

Pandas DataFrame

DataFrame

DataFrame is generally the most commonly used pandas object.
It’s a 2-dimensional labeled data structure with rows and columns.
It’s a collection of series with at least 1 series.

This is an example of DataFrame.



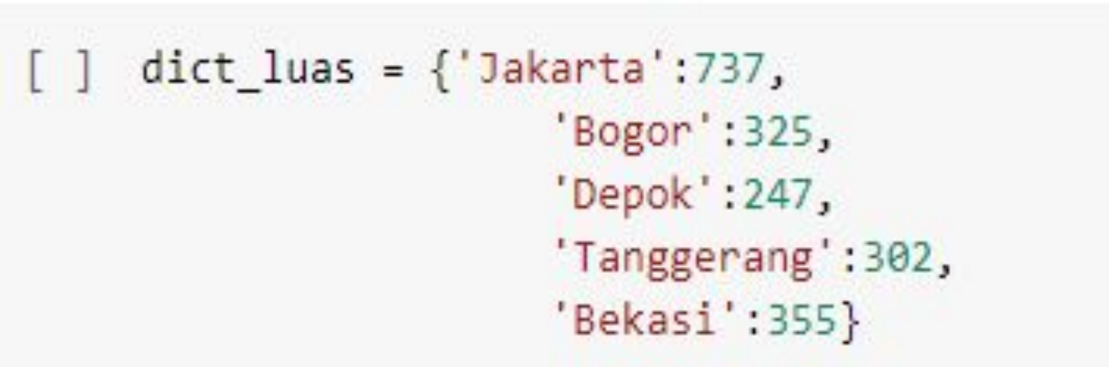
```
daerah
```

	populasi	luas
Jakarta	750	737
Bogor	490	325
Depok	350	247
Tangerang	270	302
Bekasi	670	355

In the following few steps we will walk you through the process of creating a DataFrame.

DataFrame

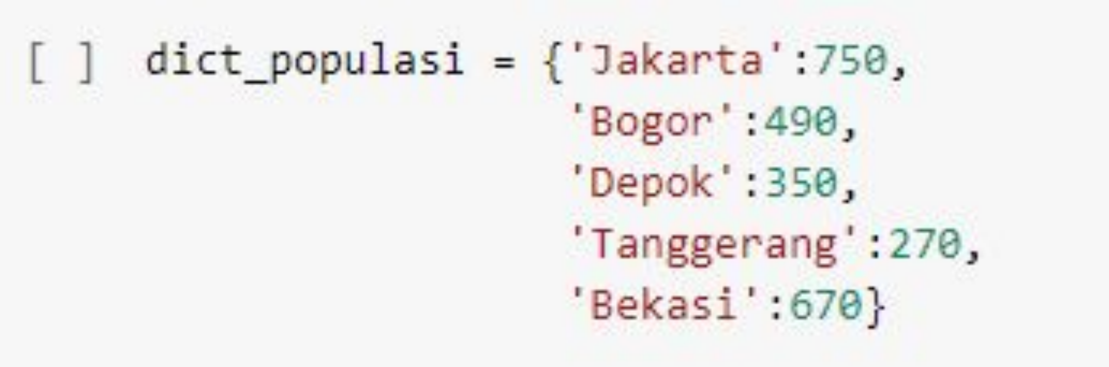
First, create a dictionary called ‘dict_luas’, with city names as the key and its area as the value



```
dict_luas = {'Jakarta':737,
                 'Bogor':325,
                 'Depok':247,
                 'Tanggerang':302,
                 'Bekasi':355}
```

DataFrame

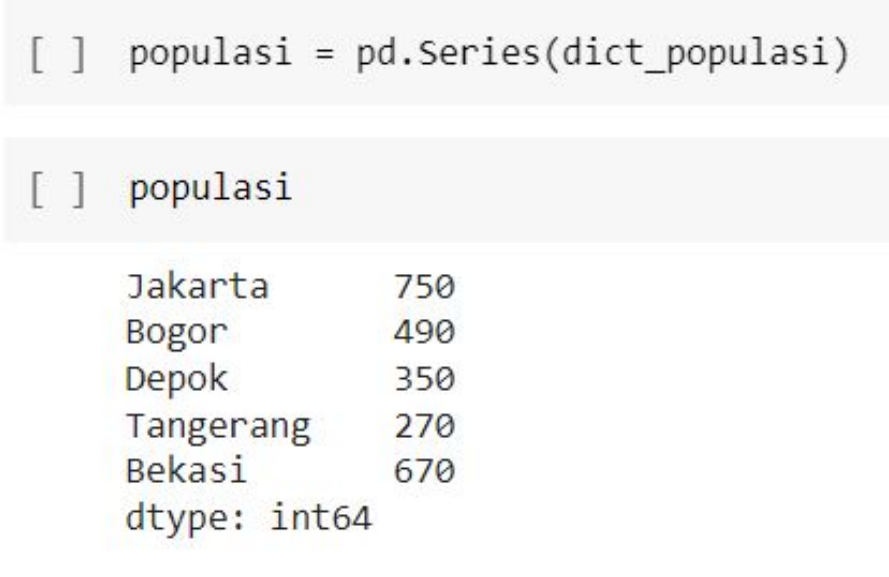
This time we create another dictionary called ‘dict_populasi’,
The keys are same as before, but with sample population as the value



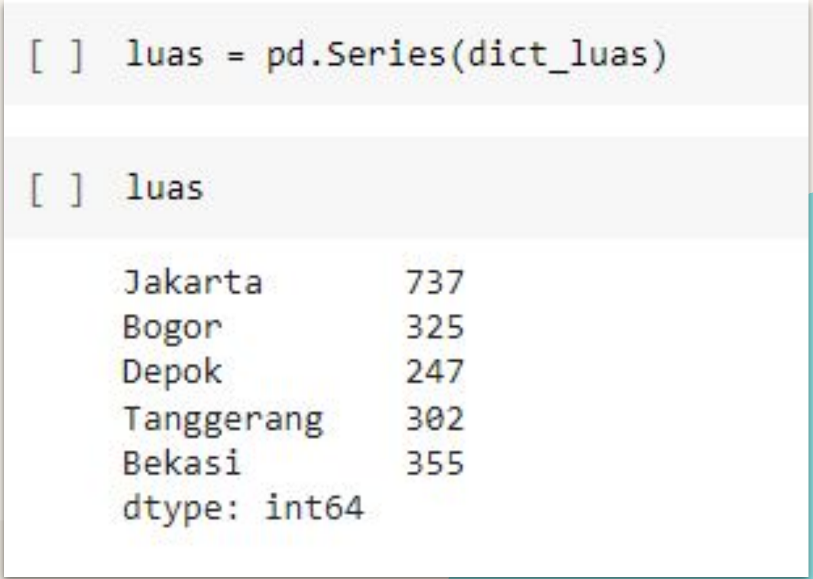
```
[ ]  dict_populasi = {'Jakarta':750,
                      'Bogor':490,
                      'Depok':350,
                      'Tanggerang':270,
                      'Bekasi':670}
```

DataFrame

Convert each of those dictionaries into series, as shown below



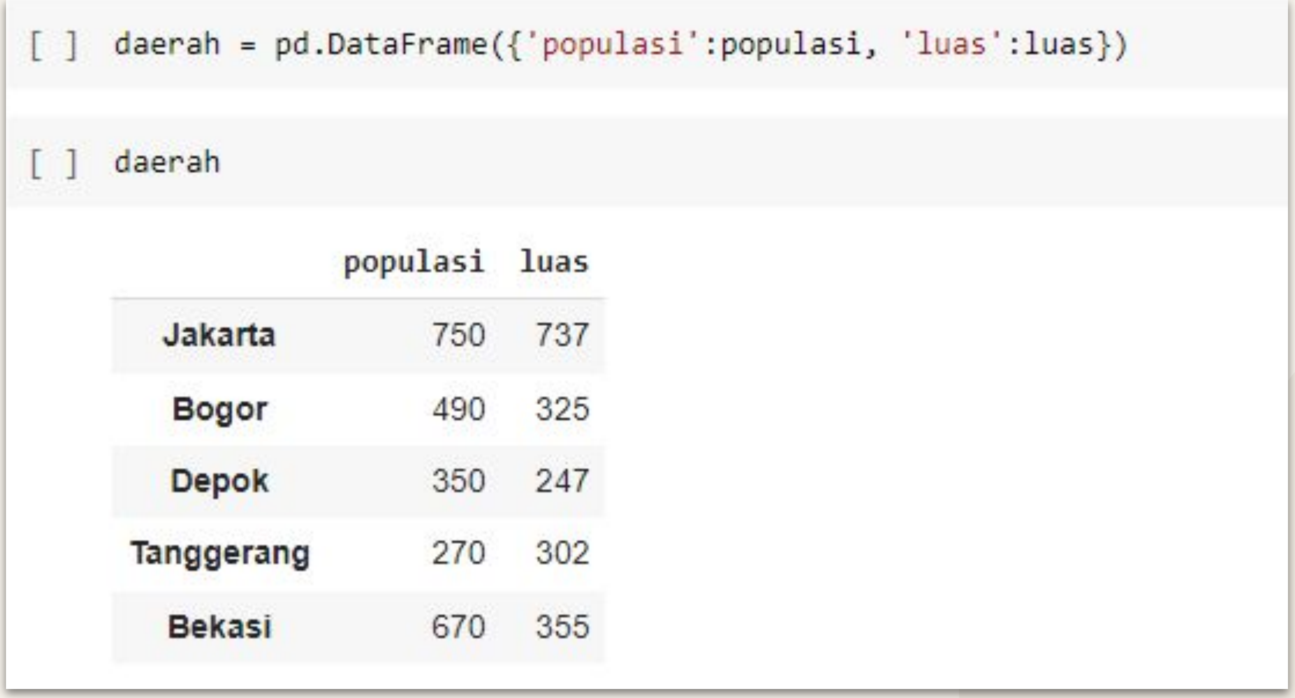
```
[ ]  populasi = pd.Series(dict_populasi)

[ ]  populasi

     Jakarta      750
     Bogor        490
     Depok        350
     Tangerang    270
     Bekasi       670
     dtype: int64
```



```
[ ]  luas = pd.Series(dict_luas)

[ ]  luas

     Jakarta       737
     Bogor         325
     Depok         247
     Tanggerang    302
     Bekasi        355
     dtype: int64
```

DataFrame

After that, we can create a DataFrame by combining the 2 series



```
[ ]  daerah = pd.DataFrame({'populasi':populasi, 'luas':luas})
```

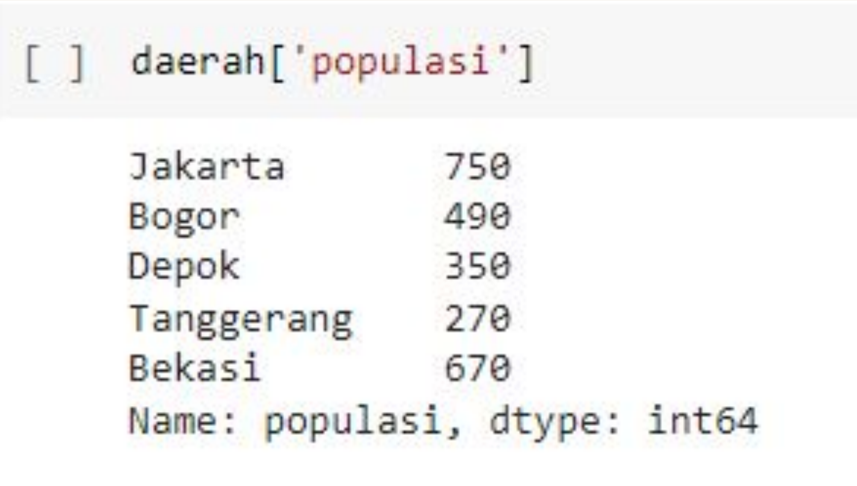
```
[ ]  daerah
```

	populasi	luas
Jakarta	750	737
Bogor	490	325
Depok	350	247
Tanggerang	270	302
Bekasi	670	355

DataFrame

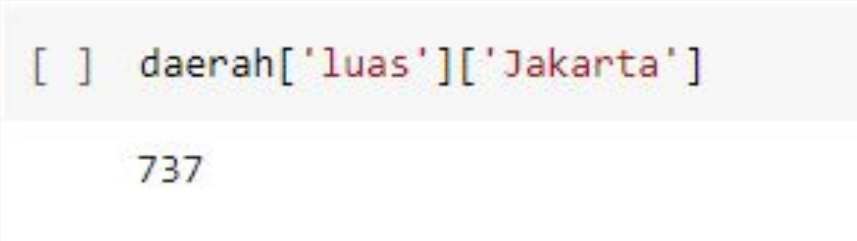
Data Selection using explicit index

selecting data at a specific column



```
[ ]  daerah['populasi']

     Jakarta      750
     Bogor        490
     Depok        350
     Tanggerang   270
     Bekasi       670
     Name: populasi, dtype: int64
```

selecting data at a specific column and row

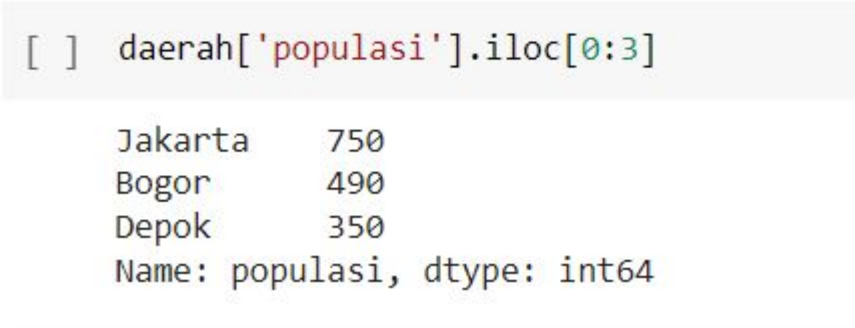


```
[ ]  daerah['luas']['Jakarta']

     737
```

DataFrame

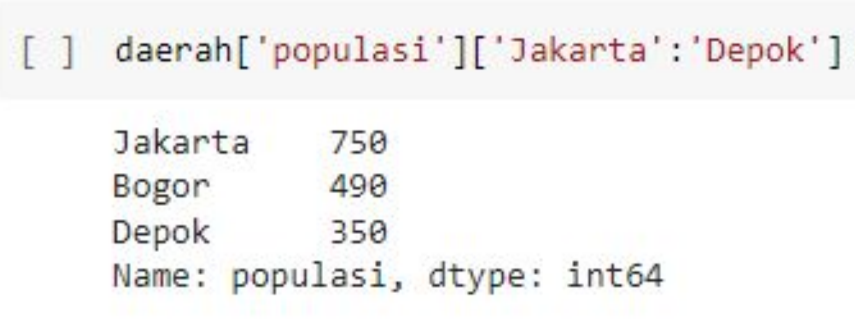
Data Slicing

Data slicing using implicit index



```
[ ]  daerah['populasi'].iloc[0:3]

     Jakarta     750
     Bogor       490
     Depok       350
     Name: populasi, dtype: int64
```

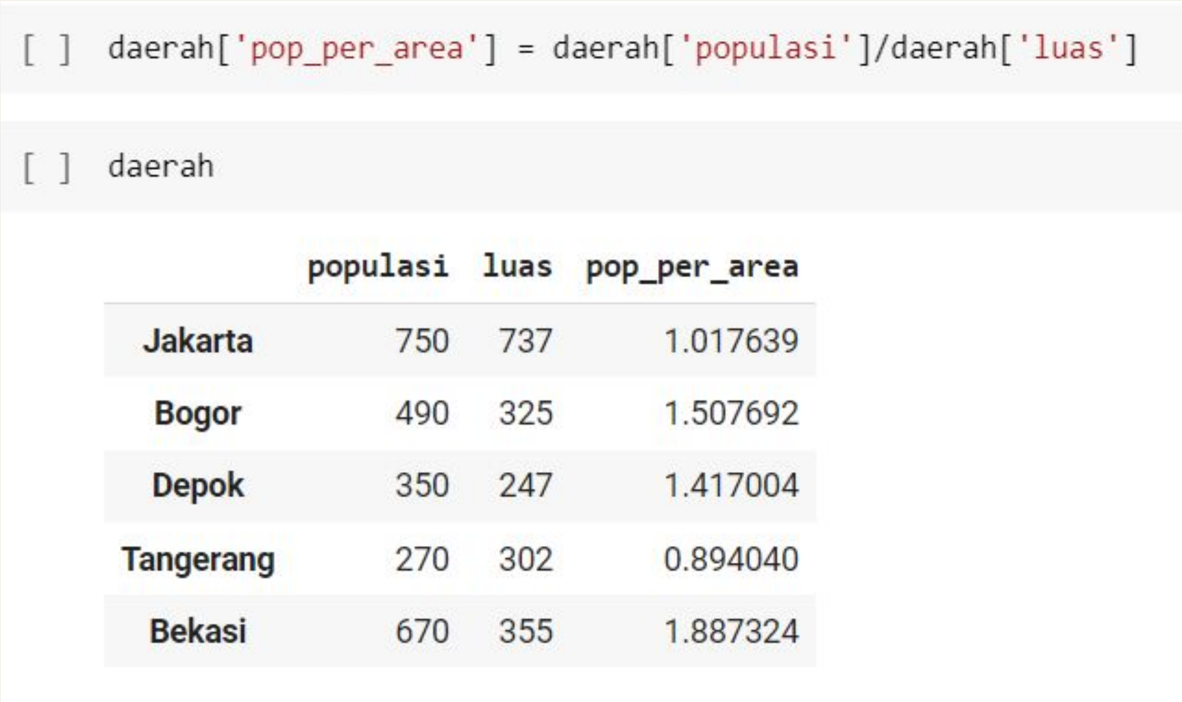
Data slicing using explicit index



```
[ ]  daerah['populasi']['Jakarta':'Depok']

     Jakarta     750
     Bogor       490
     Depok       350
     Name: populasi, dtype: int64
```

DataFrame · Add Column

We can add a new column to the DataFrame, as shown below



```
[ ] daerah['pop_per_area'] = daerah['populasi']/daerah['luas']
```

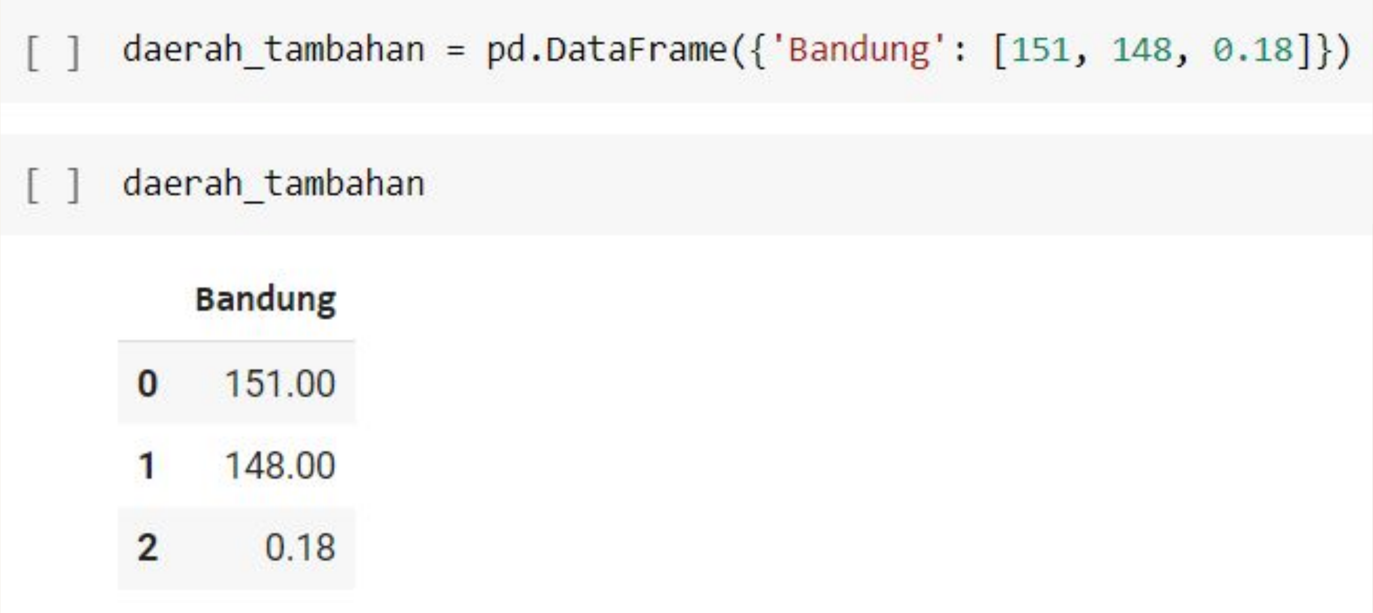
```
[ ] daerah
```

```
           populasi  luas  pop_per_area
Jakarta         750   737      1.017639
Bogor           490   325      1.507692
Depok           350   247      1.417004
Tangerang       270   302      0.894040
Bekasi          670   355      1.887324
```

DataFrame · Add Row

Not only column, we can also add a new row (though it’s more complex).

First, we need to create a new dataframe as shown below.



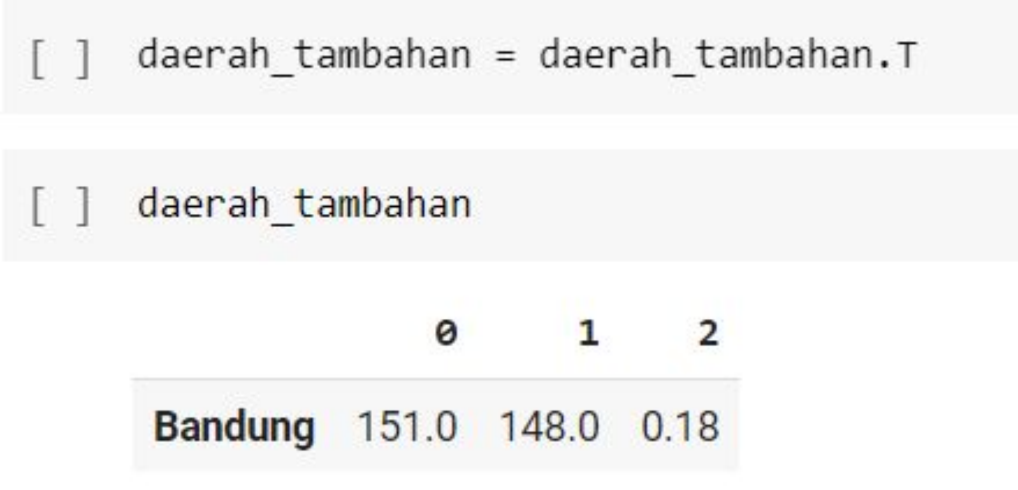
```
[ ]  daerah_tambahan = pd.DataFrame({'Bandung': [151, 148, 0.18]})
```

```
[ ]  daerah_tambahan
```

	Bandung
0	151.00
1	148.00
2	0.18

DataFrame • Add Row

Because data in ‘daerah_tambahan’ is vertical, we have to transpose it to make it horizontal



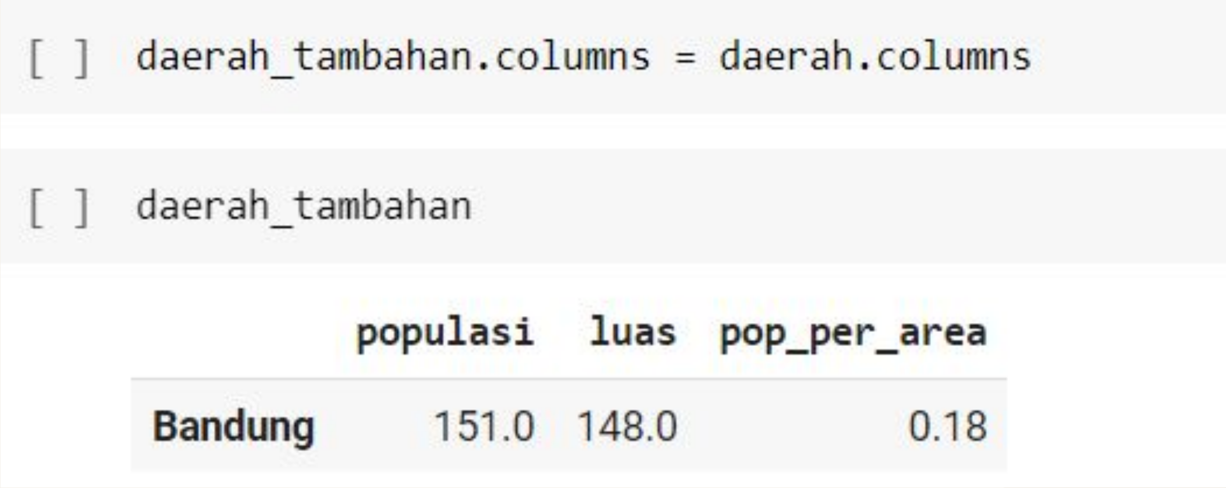
```
[ ]  daerah_tambahan = daerah_tambahan.T
```

```
[ ]  daerah_tambahan
```

	0	1	2
Bandung	151.0	148.0	0.18

DataFrame • Add Row

Define ‘daerah_tambahan’ column names same as dataframe ‘daerah’



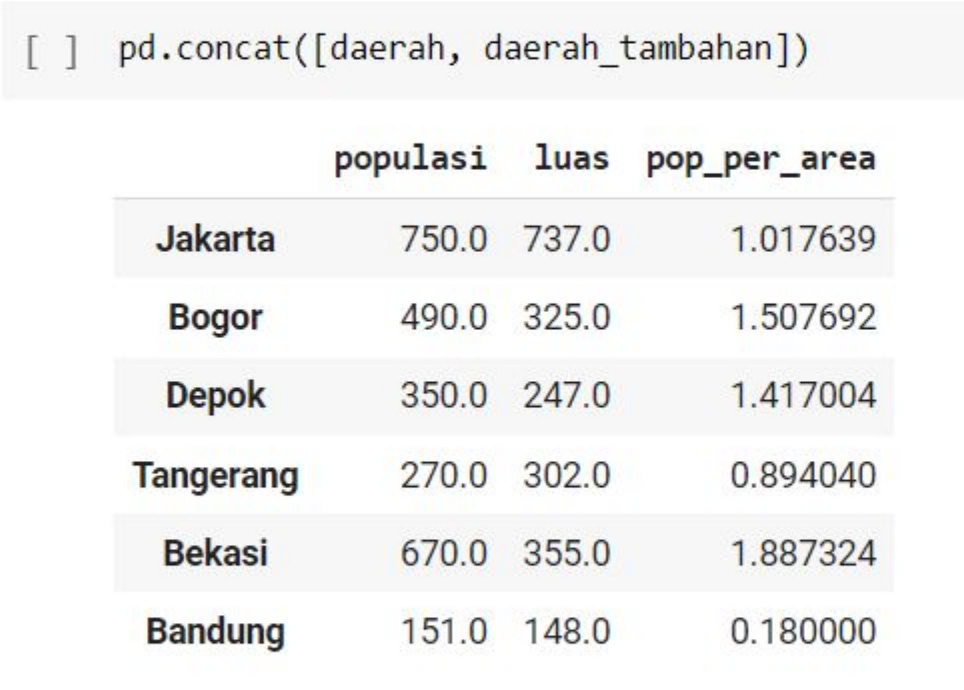
```
[ ]  daerah_tambahan.columns = daerah.columns
```

```
[ ]  daerah_tambahan
```

```
         populasi   luas  pop_per_area
Bandung     151.0  148.0          0.18
```

DataFrame · Add Row

After that, we can combine both dataframe using concat method



```
[ ]  pd.concat([daerah, daerah_tambahan])
```

	populasi	luas	pop_per_area
Jakarta	750.0	737.0	1.017639
Bogor	490.0	325.0	1.507692
Depok	350.0	247.0	1.417004
Tangerang	270.0	302.0	0.894040
Bekasi	670.0	355.0	1.887324
Bandung	151.0	148.0	0.180000

Thanks!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.

Please keep this slide for attribution.