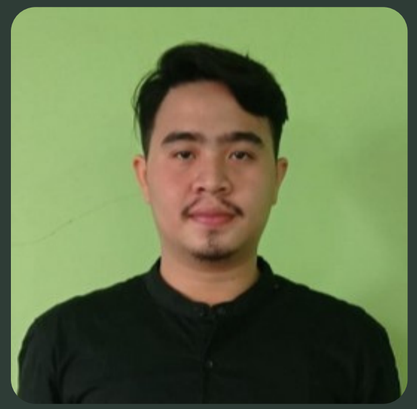
Data Manipulation

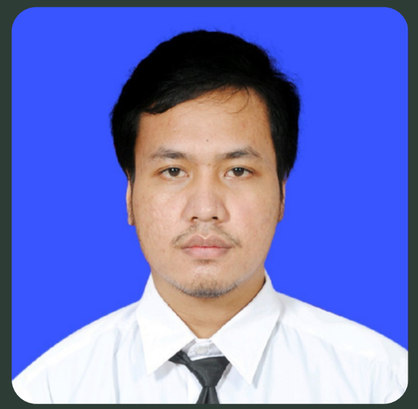
Seabron Kusto

OFFICIAL TEAM

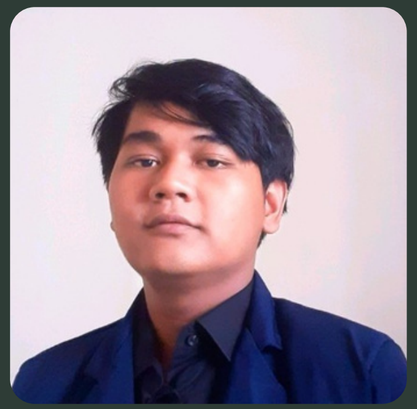


MALKA
RUSYD ABDUSSALAM

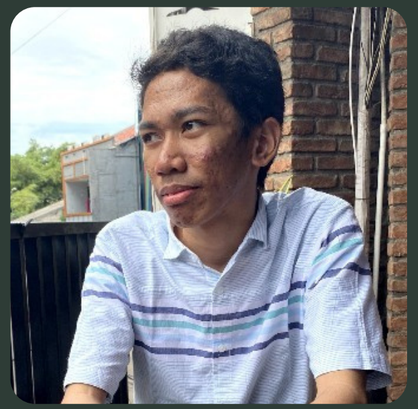
TRIANTO
HARYO NUGROHO



MAULANA
ISHAQ SIREGAR



FAUZI
WARDAH ALI



RIZKI
AFRINAL

DATA MANIPULATION WITH PYTHON

Data manipulation is not changing the data value, but it makes a machine easy to analyze the data.

Importing Libraries

- Pandas is a Python library for data analysis
- Numpy is a Pyhon library for mathematical operations

```
[1] #Import the required Libraries

    import pandas as pd
    import numpy as np
```

Pandas has two objects, namely series and data frames

Object Series

Object series is a data dimension. It has no column's name because it only has one column and index.

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

Convert data into index

Display data in the form of an index

```
[3]  data = pd.Series(data)
```

```
data
```

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

Convert from series into array

```
[5]  data.values

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

Display index

The Index is in the range from, where the start point is inclusive in the range and the stop point is the executive of the range

```
[6]  data.index

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

[7]  list(range(1,10))

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

How to call the data

```
[ ] data
```

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

To display the second index on the data

```
[8] data[2]
```

```
0.75
```

0.75 is the second index on the data

Implicit and Explicit Index

- Implicit index is the default index
- We can define the index, it's called explicit, a difined index
- When defining the index, the amount of index has to be equal with the data power

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

```
[ ] 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')
```

Call the data

```
[ ] #Index eksplisit

    data['a']
```

```
    0.25
```

This is data selection,

Even we have made an explicict index, we can still call the implicit index

```
[ ] #Index implisit

    data[3]
```

```
    1.0
```

When implicit and explicit index having the same value, it will rely on the explicict once we call the data

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

[ ] data_2

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

Replace the implicit index with a new index worth 2, 5, 3, 7

After changing the index on the data, index 2 is worth 0.25

```
[ ] data_2[2]
```

```
0.25
```

Index 0 is not available

```
[ ] data_2[0]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

                                5 frames

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTable.get_item()

KeyError: 0

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364
   3365         if is_scalar(key) and isna(key) and not self.hasnans:

KeyError: 0

SEARCH STACK OVERFLOW
```

INDEX 0

Remove an output Error because there is no index 0

Data Slicing

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

```
[ ] data
```

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

For example, we will call from data b to data c

```
data['b':'c']#indeks eksplisit
```

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

But when we slice the implicit index, it will only display the start point because the implicit index is a range.

```
[ ] data[1:2]#indeks implisit
```

```
b    0.5
dtype: float64
```

loc and iloc

When selecting rows and columns of a pandas DataFrame, loc and iloc are two commonly used functions. Here is the subtle difference between the two functions:

- **loc** selects rows and columns with spesific **labels**
  loc access a group of rows and columns by label(s) or a boolean array. loc[] is primarily label based, but may also be used with a boolean array.

- **iloc** selects rows and columns at specific **integer positions**
  iloc purely integer-location based indexing for selection by position. iloc[] is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array.

Calling and slicing data without using loc and iloc

Assign seriess and calling the values

```
[ ]  # Assign series and explicit index to the variable data_2

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

```
[ ]  # Calling the values, index and data type of the data_2 series

     data_2
```

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

When we access an index, then what appears is the explicit index

```
[ ]  # Calling the value of the explicit index 2 of the data_2 series(data selection)

     data_2[2]
```

```
     0.25
```

When we call the explicit index from index 2 to index 3, the value that appears is precisely from the implicit index

```
[ ]  # Calling the implicit index from index 2 to index 3 (slicing)

     data_2[2:3]
```

```
     5    0.75
     dtype: float64
```

Calling and slicing data using loc and iloc

When the explicit index and the implicit index are the same, there will be inconsistencies as in the case above. To overcome this inconsistency, we will use the loc and iloc rules. loc is to call its explicit index. iloc is to call its implicit index.

loc

```
[ ]  # Calling explicit index 3 using loc (data selection)

     data_2.loc[3]
```

```
    0.5
```

```
[ ]  # Calling implisit index 2 to index 3 using loc (data slicing)

     data_2.loc[2:3]
```

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

iloc

```
[ ]  # Calling implicit index 3 using iloc (data selection)

     data_2.iloc[3]
```

```
     1.0
```

```
[ ]  # Calling implicit index 2 to 3 using iloc (data slicing)

     data_2.iloc[2:3]
```

```
     5    0.75
     dtype: float64
```

Data Frame

Data Frame is a collection of series, with at least one series.

Assign a dictionary and calling the data

```
[ ]  # Assign keys and values of dictionary dict_population

     dict_population ={'Jakarta':750,
                       'Bogor':490,
                       'Depok':350,
                       'Tangerang':270,
                       'Bekasi':670}

     # This is just an example, not a real population figure
```

```
[ ]  # Calling the keys and values of dict_population

     dict_population
```

```
{'Bekasi': 670, 'Bogor': 490, 'Depok': 350, 'Jakarta': 750, 'Tangerang': 270}
```

Convert dictionary to series using pd.Series() and calling the data

```
[ ]  # Convert dictionary to series using pd.Series() and assign it to variable population

     population = pd.Series(dict_population)
```

```
[ ]  # Calling values of the population series

     population
```

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

Calling the number of popolution in Depok using explicit index

```
[ ]  # Calling the number of popolution in Depok using explicit index 'Depok'

     population.loc['Depok']
```

```
     350
```

Calling the number of popolution in Depok using implicit index

```
[ ]  # Calling the number of population in Depok using implicit index 2

     population.iloc[2]
```

```
     350
```

Assign second dictionary called dict_area

```
[ ]  # Assign Keys and Values to the variable dict_area

     dict_area = {'Jakarta':737,
                  'Bogor':325,
                  'Depok':247,
                  'Tangerang':302,
                  'Bekasi':355}

     # This is just an example, not a real area number
```

Convert dictionary to series

```
[ ]  # Convert the dictionary dict_area to series using pd.Series() and assign it to the variable area

     area = pd.Series(dict_area)
```

Calling the data of area series

```
[ ]  # Calling the values, index, and data types of the area series

     area
```

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

Convert both series to a data frame

```
[ ]  # Convert two series to Data Frame using pd.DataFrame(). It called concantenate two series and give the column name and assign as pop_area

     pop_area = pd.DataFrame({'pop':population, 'area':area})
```

Calling the data of the pop_area data frame

```
[ ]  # Calling the values of the pop_area Data Frame

     pop_area
```

	pop	area
Jakarta	750	737
Bogor	490	325
Depok	350	247
Tangerang	270	302
Bekasi	670	355

Calling the data of area series

```
[ ]  # Calling the values, index, and data types of the area series

     area
```

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

Calling the data of a spesific column and bar

```
[ ]  # Calling the data in the column 'area' and the explicit index 'Jakarta'

     pop_area['area']['Jakarta']
```

```
     737
```

When calling data with pop_area.pop syntax it will appear as below because pop is the same as the name of the function in the data frame.

```
[ ]  # Calling the data of the pop_area using pop_area.pop

     pop_area.pop
```

```
     <bound method DataFrame.pop of            population  area   density
     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>
```

```
[ ]  # Calling the data of the pop_area in column 'pop'

     pop_area['pop']
```

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

Rename the 'pop' column name to 'population'

```
[ ]  # Rename the 'pop' column with population

     pop_area = pd.DataFrame({'population':population,'area':area})
```

Calling the data of the pop_area data frame

```
[ ]  # Calling the data of the column 'population'

     pop_area['population']
```

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

Calling the spesific data in 'population' column and 'Jakarta' to 'Depok' bar using explicit index

```
[ ]  # Calling the data in the column 'population' using the explicit index 'Jakarta' to 'Depok'

     pop_area['population']['Jakarta':'Depok']
```

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

Calling the spesific data in 'population' column and 'Jakarta' to 'Depok' bar using implicit index

```
[ ]  # Calling the data in the column 'population' using the implicit index 0 to 3

     pop_area['population'].iloc[0:3]
```

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

Add a new column

```
[ ]  # Add a new column called 'density' whoose contents are the results of division calculations from the previous two columns

     pop_area['density']=pop_area['population']/pop_area['area']
```

Calling the updated data of the pop_area data frame

```
[ ]  # Calling the data of the pop_area data frame

     pop_area
```

	population	area	density
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

Add a new bar

```
[ ]  # Add new bar called 'Bandung' with the values of each column are 151, 148. 0.18

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

Transpose the new bar

```
[ ]  # Transpose the new_area data frame

     new_area=new_area.T
```

Calling the data of the new bar

```
[ ]  # Calling the data of the new_area data frame

     new_area
```

	0	1	2
Bandung	151.0	148.0	0.18

Change the column name

```
[ ]  # Change the column name the same as the column names of pop_area data frame

     new_area.columns=pop_area.columns
```

Calling the updated data of the new bar

```
[ ]  # Calling the data of the new_area series

     new_area
```

	population	area	density
Bandung	151.0	148.0	0.18

Concantenate the new bar to the existing data frame

```
[ ]  # Concantenate the data of pop_area and new_area using pd.concat()

     pd.concat([pop_area, new_area])

     # The 'Bandung' bar now exist in the last bars
```

	population	area	density
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

Thank You !

Seabron Kusto