Manipulasi Data dengan Pandas

Statistics descriptive, handling missing value and finding outliers

Topik

- M13 - Manipulasi Data Pandas - Bagian 1
  - **Introduction to Pandas**
    - Memanggil Library Pandas
    - DataFrame & Series
    - Atribut DataFrame & Series
      - .info()
      - .shape
      - .dtypes
      - .astype(*nama_tipe_data*)
      - .copy()
      - .to_list()
      - .unique()
      - .index
      - .columns
      - .loc
      - .iloc
    - Creating Series & Dataframe from List
    - Creating Series & Dataframe from Dictionary
    - Creating Series & Dataframe from Numpy Array

- M13 - Manipulasi Data dengan Pandas - Bagian 1
  - **Dataset I/O**
    - Read Dataset - CSV dan TSV
    - Read Dataset - Excel
    - Read Dataset - JSON
    - Read Dataset – SQL
    - Read Dataset - Google BigQuery
    - Write Dataset
    - Head & Tail
  - **Indexing, Slicing, dan Transforming**
    - Indexing
    - Slicing
    - Transforming
  - **Handling Missing Values**
    - Inspeksi Missing Value
    - Treatment untuk Missing Value

Pandas

(Python Data Analysis Library)

Pendahuluan

- Pandas adalah library python open source yang biasanya digunakan untuk kebutuhan data analisis.
- Pandas berguna untuk melakukan analisis dan pengolahan (manipulasi dan pembersihan) data dari menengah sampai besar.
- Pandas mendukung pembacaan dan penulisan data dengan media berupa *excel spreadsheet*, CSV, dan SQL yang kemudian akan dijadikan sebagai objek python dengan *rows* dan *columns* yang disebut *data frame* seperti halnya pada tabel statistik.
- Dengan Pandas, Python dapat bekerja dengan data yang berbentuk tabular seperti spreadsheet dengan cara :
  - pemuatan data yang cepat,
  - manipulasi data,
  - menggabungkan data,
  - berbagai fungsi yang lain.

Pendahuluan

- Instalasi Pandas
  - Dengan menggunakan pip:

    ```
    pip install pandas
    ```
  - Dengan menggunakan Anaconda:

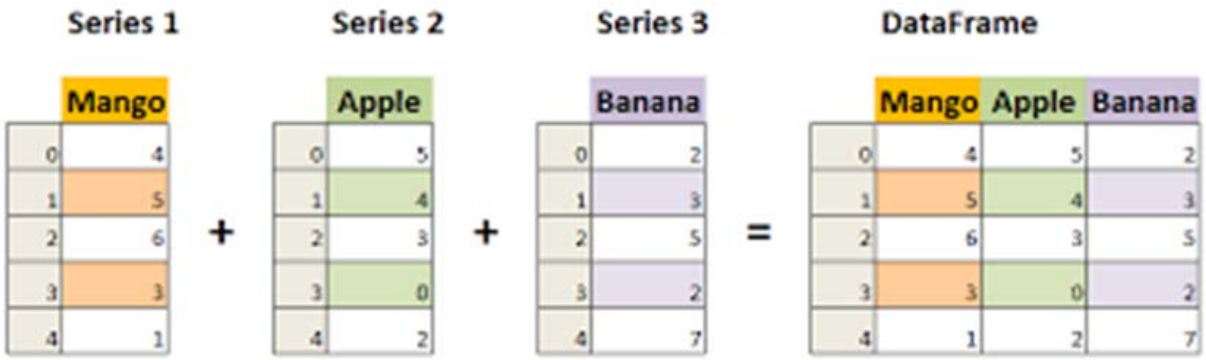
    ```
    conda install pandas
    ```
- Memanggil Library Pandas:

  ```
  import pandas as pd
  ```
- Biasanya ketika menggunakan library Pandas, library Numpy juga diimport:

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

Struktur Data Pandas

- Pandas memiliki dua tipe struktur data untuk versi terbaru dan satu *deprecated* struktur data:
  - Series
  - Data Frame
  - Panel (deprecated)



Series

- Series merupakan struktur data dasar dalam Pandas.
- Series bisa juga diibaratkan sebagai array satu dimensi seperti halnya yang ada pada **numpy array**, hanya bedanya mempunyai index dan kita dapat mengontrol index dari setiap elemen tersebut.
- Struktur data yang bisa ditampung berupa *integer*, *float,* dan juga *string.*
- Series juga mendukung operasi vektor.
- Secara definisi, Series tidak dapat mempunyai kolom ganda, untuk masalah ini bisa menggunakan struktur data *frame*.

Data Frame

- Data frame merupakan array dua dimensi dengan baris dan kolom.
- Struktur data ini merupakan cara paling standar untuk menyimpan data.
- Secara sederhana, data frame merupakan tabel/data tabular.
- Setiap kolom pada Data Frame merupakan objek dari Series, dan baris terdiri dari elemen yang ada pada Series.
- Baris berguna untuk menyimpan informasi dan kolom untuk menyimpan nama dari informasi tersebut.

DataFrame & Series

- Pandas memiliki 2 kelas data yang digunakan sebagai struktur dari spreadsheet:
  1. **Series**: satu kolom bagian dari tabel dataframe yang merupakan 1 dimensional numpy array sebagai basis data nya, terdiri dari 1 tipe data (integer, string, float, dll).

```
1 import pandas as pd
2 # Series
3 number_list = pd.Series([1,2,3,4,5,6])
4 print("Series:")
5 print(number_list)
```

  2. **DataFrame**: gabungan dari Series, berbentuk *rectangular data* yang merupakan tabel spreadsheet itu sendiri (karena dibentuk dari banyak Series, tiap Series biasanya punya 1 tipe data, yang artinya 1 dataframe bisa memiliki banyak tipe data).

```
6 # DataFrame
7 matrix = [[1,2,3],
8          ['a','b','c'],
9          [3,4,5],
10         ['d',4,6]]
11 matrix_list = pd.DataFrame(matrix)
12 print("DataFrame:")
13 print(matrix_list)
```

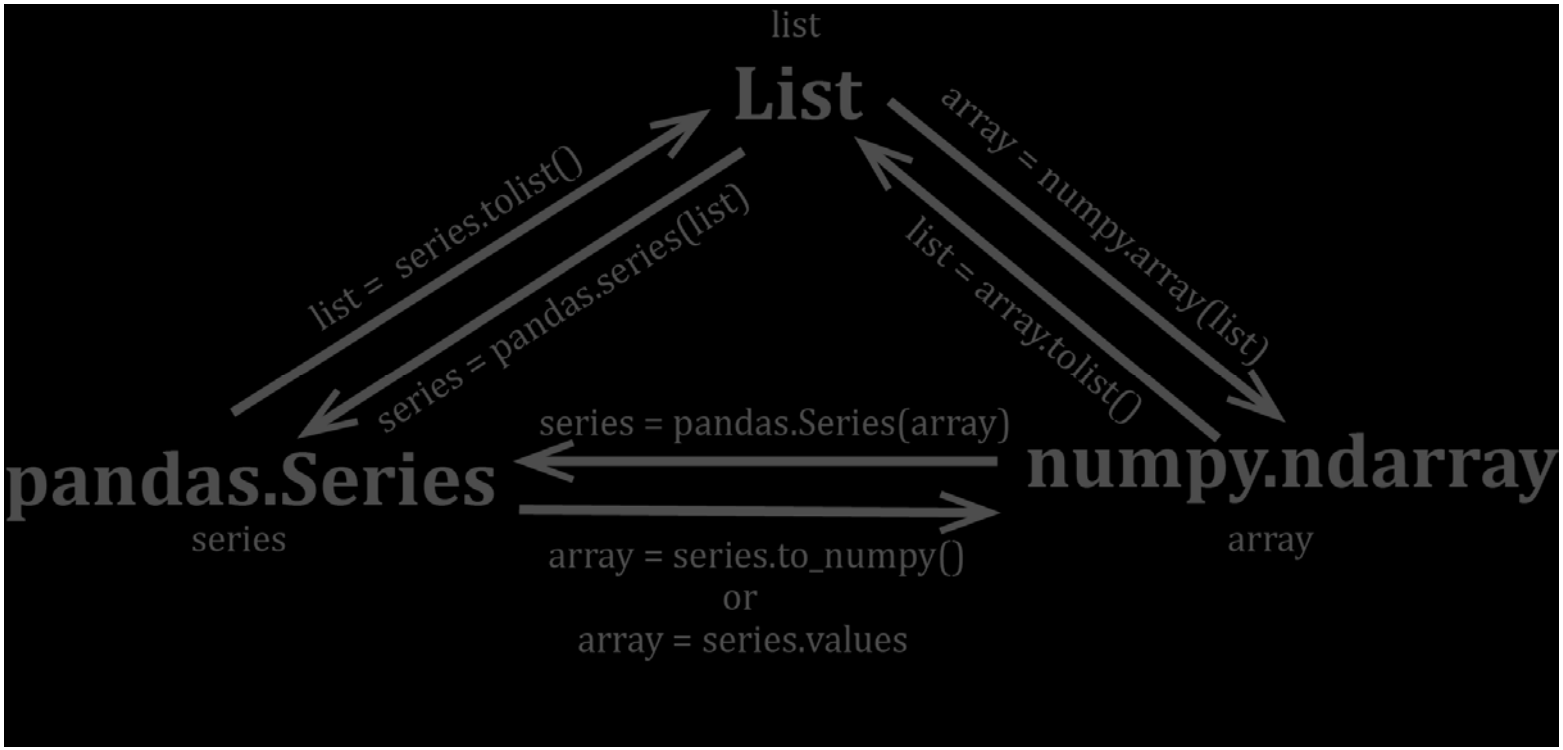
```
list_var = [1,2,3,4]
series_var = pd.Series(list_var)
```

```
print list_var*2
```

```
[1, 2, 3, 4, 1, 2, 3, 4]
```

```
print series_var*2
```

```
0    2
1    4
2    6
3    8
dtype: int64
```



Membuat Series di Pandas

- Series pada Pandas bisa dibuat dengan menggunakan python list ataupun array dari NumPy.
- Untuk membuatnya, kita bisa memanggil *pd.Series()* method dengan isi array.
- Berbeda dengan python list, Series hanya dapat berisi data dengan tipe yang sama.
- Jadi, bisa juga menggunakan array dari NumPy untuk mengisi data pada Series.
- Untuk membuat Series bisa dengan mendefinisikan

```
In [1]: import pandas as pd
   ...:
   ...: series = pd.Series()
   ...: print(series)
Series([], dtype: float64)
```

- Pada contoh di atas, kita membuat Series kosong tanpa data.

Membuat Series di Pandas

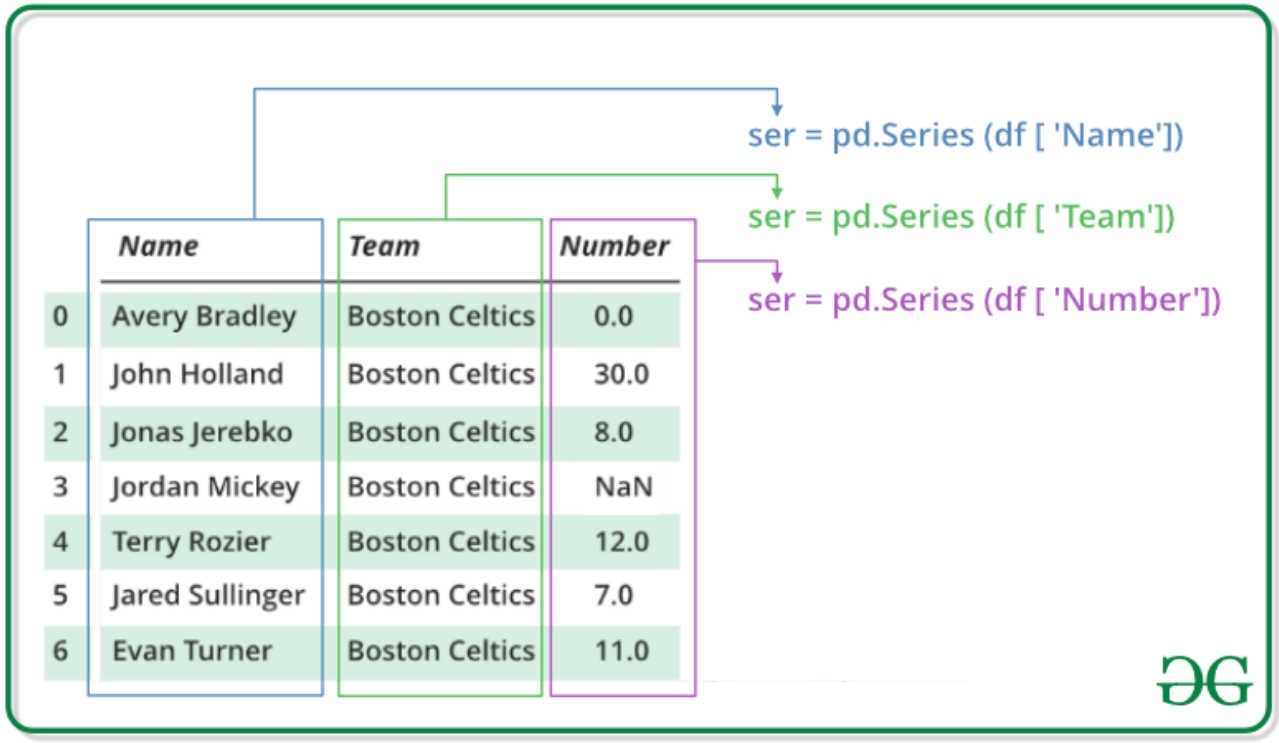
Menggunakan python list ataupun numpy sebagai contoh data untuk membuat series.

```
In [1]: import pandas as pd
   ...: import numpy as np
   ...: data = pd.Series(['a','b','c','d'])
   ...: np1 = np.array(['a','b','c','d'])
   ...: data_np = pd.Series(np1)

In [2]: print(data)
0   a
1   b
2   c
3   d
dtype: object

In [3]: print(data_np)
0   a
1   b
2   c
3   d
dtype: object
```

- Kita tidak mendefinisikan indeks pada data tersebut.
- Secara default indeks data akan diberikan seperti halnya indeks array yang bermula dari 0-N dengan RangeIndex(start, stop, step)


ser = pd.Series (df ['Name'])
ser = pd.Series (df ['Team'])
ser = pd.Series (df ['Number'])
Name
Team
Number
0 Avery Bradley Boston Celtics 0.0
1 John Holland Boston Celtics 30.0
2 Jonas Jerebko Boston Celtics 8.0
3 Jordan Mickey Boston Celtics NaN
4 Terry Rozier Boston Celtics 12.0
5 Jared Sullinger Boston Celtics 7.0
6 Evan Turner Boston Celtics 11.0

Tugas 1. Membuat series dan dataframe

- Mengganti tanda _ _ _ dengan yang sesuai seperti yang diberikan pada contoh di bagian teori.
- Import pandas terlebih dahulu sebagai aliasnya.
- Membuat list yang berisi nilai integer dari 1 s/d 6 dan kemudian mencetaknya ke console.
- Membuat data frame dengan diawali membuat list of list yang masing-masing list terdalam berisi 3 elemen, sementara list terluar berisi 4 buah list terdalam. Selanjutnya, cetaklah ke console.

Lengkapi kode berikut:

```
import ___ as ___

# Series
number_list = ___
print("Series:")
print(___)

# DataFrame
matrix = [[___],
          [___],
          [___],
          [___]]
matrix_list = ___
print("DataFrame:")
print(___)
```

Hasil yang diharapkan:

```
Series:
0   1
1   2
2   3
3   4
4   5
5   6
dtype: int64

DataFrame:
  0 1 2
0 1 2 3
1 a b c
2 3 4 5
3 d 4 6
```

Atribut DataFrame & Series

Dataframe dan Series memiliki sangat banyak atribut yang digunakan untuk transformasi data, tetapi ada beberapa attribute yang sering dipakai:

1. Attribute .info()
2. Attribute .shape
3. Attribute .dtypes
4. Attribute .astype(*nama_tipe_data*)
5. Attribute .copy()
6. Attribute .to_list()
7. Attribute .unique()
8. Attribute .index
9. Attribute .columns
10. Attribute .loc
11. Attribute .iloc

```
1 import pandas as pd
2 # Series
3 number_list = pd.Series([1,2,3,4,5,6])
4 # Dataframe
5 matrix_list = pd.DataFrame([[1,2,3],
6                             ['a','b','c'],
7                             [3,4,5],
8                             ['d',4,6]])
```

series **number_list** dan data frame **matrix_list** pada bagian sebelumnya digunakan kembali

Atribut DataFrame & Series:
1. Attribute .info()

- Attribute .info() digunakan untuk mengecek kolom apa yang membentuk dataframe itu, data types, berapa yang non null, dll.
- Attribute ini tidak dapat digunakan pada series, hanya pada data frame saja.

```
# [1] attribute .info()
print("[1] attribute .info()")
print(matrix_list.info())
```

- Output di console untuk penggunaan attribute .info() ini adalah

```
[1] attribute .info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
0    4 non-null object
1    4 non-null object
2    4 non-null object
dtypes: object(3)
memory usage: 176.0+ bytes
None
```

Atribut DataFrame & Series:
2. Attribute .shape

- Attribute .shape digunakan untuk mengetahui berapa baris dan kolom, hasilnya dalam format tuple (baris, kolom).

```
12 # [2] attribute .shape
13 print("\n[2] attribute .shape")
14 print("    Shape dari number_list:", number_list.shape)
15 print("    Shape dari matrix_list:", matrix_list.shape)
```

- Output di console untuk penggunaan attribute .shape ini adalah

```
[2] attribute .shape
    Shape dari number_list: (6,)
    Shape dari matrix_list: (4, 3)
```

Atribut DataFrame & Series:
3. Attribute .dtypes

- Attribute .dtypes digunakan untuk mengetahui tipe data di tiap kolom.
- ***Tipe data object***: kombinasi untuk berbagai tipe data (number & text, etc).

```
16 # [3] attribute .dtypes
17 print("\n[3] attribute .dtypes")
18 print("    Tipe data number_list:", number_list.dtypes)
19 print("    Tipe data matrix_list:", matrix_list.dtypes)
```

- Output di console untuk penggunaan attribute .dtypes ini adalah

```
[3] attribute .dtypes
    Tipe data number_list: int64
    Tipe data matrix_list: 0    object
1    object
2    object
dtype: object
```

Atribut DataFrame & Series:
4. Attribute .astype(*nama_tipe_data*)

- Attribute .astype(*nama_tipe_data*) untuk convert tipe data berdasarkan tipe data seperti: **float, int, str, numpy.float**, **numpy.int** ataupun **numpy.datetime**.

```
20 # [4] attribute .astype()
21 print("\n[4] attribute .astype()")
22 print("    Konversi number_list ke str:", number_list.astype("str"))
23 print("    Konversi matrix_list ke str:", matrix_list.astype("str"))
```

- Output di console untuk penggunaan attribute .astype() ini adalah

```
[4] attribute .astype()
    Konversi number_list ke str: 0    1
1    2
2    3
3    4
4    5
5    6
dtype: object
    Konversi matrix_list ke str:    0  1  2
0  1  2  3
1  a  b  c
2  3  4  5
3  d  4  6
```

Tugas 2. Atribut series dan dataframe

- Ganti tanda _ _ _ dengan yang sesuai seperti yang diberikan pada contoh di teori.
- Gunakan keempat attribute yang dicontohnya sesuaii urutan, yaitu .info(), .shape, .dtypes, dan .astype().
- Attribute .info() hanya dapat digunakan untuk dataframe saja.

```
import pandas as pd
# Series
number_list = pd.Series([1,2,3,4,5,6])
# DataFrame
matrix_list = pd.DataFrame([[1,2,3], ['a','b','c'],
                            [3,4,5], ['d',4,6]])
# [1] attribute .info()
print("[1] attribute .info()")
print(___)
# [2] attribute .shape
print("\n[2] attribute .shape")
print("    Shape dari number_list:", ___)
print("    Shape dari matrix_list:", ___)
# [3] attribute .dtypes
print("\n[3] attribute .dtypes")
print("    Tipe data number_list:", ___)
print("    Tipe data matrix_list:", ___)
# [4] attribute .astype()
print("\n[4] attribute .astype()")
print("    Konversi number_list ke str:", ___)
print("    Konversi matrix_list ke str:", ___)
```

Tugas 2. Membuat series dan dataframe

Hasil yang diharapkan

```
[1] attribute .info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
0    4 non-null object
1    4 non-null object
2    4 non-null object
dtypes: object(3)
memory usage: 176.0+ bytes
None

[2] attribute .shape
   Shape dari number_list: (6,)
   Shape dari matrix_list: (4, 3)
```

```
[3] attribute .dtypes
   Tipe data number_list: int64
   Tipe data matrix_list: 0    object
1    object
2    object
dtype: object

[4] attribute .astype()
   Konversi number_list ke str: 0    1
1    2
2    3
3    4
4    5
5    6
dtype: object
   Konversi matrix_list ke str:    0  1  2
0  1  2  3
1  a  b  c
2  3  4  5
3  d  4  6
```

Atribut DataFrame & Series:
5. Attribute .copy ()

- Attribute .copy() digunakan melakukan duplikat, untuk disimpan di variable yang berbeda mungkin supaya tidak loading data lagi.

```
# [5] attribute .copy()
print("[5] attribute .copy()")
num_list = number_list.copy()
print("    Copy number_list ke num_list:", num_list)
mtr_list = matrix_list.copy()
print("    Copy matrix_list ke mtr_list:", mtr_list)
```

- Output di console untuk penggunaan attribute .copy() ini adalah

```
[5] attribute .copy()
    Copy number_list ke num_list: 0    1
1    2
2    3
3    4
4    5
5    6
dtype: int64
    Copy matrix_list ke mtr_list:    0  1  2
0  1  2  3
1  a  b  c
2  3  4  5
3  d  4  6
```

Atribut DataFrame & Series:
6. Attribute .to_list()

- Attribute .to_list() digunakan untuk mengubah series menjadi list dan tidak dapat digunakan untuk dataframe.

```
15 # [6] attribute .to_list()
16 print("[6] attribute .to_list()")
17 print(number_list.to_list())
```

- Output di console untuk penggunaan attribute .to_list() ini adalah

```
[6] attribute .to_list()
[1, 2, 3, 4, 5, 6]
```

Atribut DataFrame & Series:
7. Attribute .unique()

- Attribute .unique() digunakan menghasilkan nilai unik dari suatu kolom, hasilnya dalam bentuk numpy array.
- Attribute ini hanya digunakan pada series saja.

```
18 # [7] attribute .unique()
19 print("[7] attribute .unique()")
20 print(number_list.unique())
```

- Output di console untuk penggunaan attribute . unique() ini adalah

```
[7] attribute .unique()
[1 2 3 4 5 6]
```

Tugas 3. Atribut series dan dataframe

- Ganti tanda _ _ _ dengan yang sesuai seperti yang diberikan pada contoh di teori.
- Gunakan attribute .copy(), .to_list(), dan .unique().
- Attribute .to_list() dan .unique() hanya dapat digunakan untuk pandas Series.

```
import pandas as pd
# Series
number_list = pd.Series([1,2,3,4,5,6])
# DataFrame
matrix_list = pd.DataFrame([[1,2,3], ['a','b','c'],
                            [3,4,5], ['d',4,6]])
# [5] attribute .copy()
print("[5] attribute .copy()")
num_list = number_list.copy()
print("    Copy number_list ke num_list:", num_list)
mtr_list = matrix_list.copy()
print("    Copy matrix_list ke mtr_list:", mtr_list)

# [6] attribute .to_list()
print("[6] attribute .to_list()")
print(number_list.to_list())
# [7] attribute .unique()
print("[7] attribute .unique()")
print(number_list.unique())
```

Tugas 3. Atribut series dan dataframe

Hasil yang diharapkan

```
[5] attribute .copy()
    Copy number_list ke num_list: 0    1
1    2
2    3
3    4
4    5
5    6
dtype: int64
    Copy matrix_list ke mtr_list:    0  1  2
0  1  2  3
1  a  b  c
2  3  4  5
3  d  4  6
[6] attribute .to_list()
[1, 2, 3, 4, 5, 6]
[7] attribute .unique()
[1 2 3 4 5 6]
```

Atribut DataFrame & Series:
8. Attribute .index

- Attribute .index digunakan untuk mencari index/key dari Series atau Dataframe.

```
9  # [8] attribute .index
10 print("[8] attribute .index")
11 print("    Index number_list:", number_list.index)
12 print("    Index matrix_list:", matrix_list.index)
```

- Output di console untuk penggunaan attribute .index ini adalah

```
[8] attribute .index
    Index number_list: RangeIndex(start=0, stop=6, step=1)
    Index matrix_list: RangeIndex(start=0, stop=4, step=1)
```

Data awal:

Series:

```
0  1
1  2
2  3
3  4
4  5
5  6
dtype: int64
```

DataFrame:

```
   0  1  2
0  1  2  3
1  a  b  c
2  3  4  5
3  d  4  6
```

Atribut DataFrame & Series:
9. Attribute .columns

- Attribute .columns digunakan untuk mengetahui apa saja kolom yang tersedia di dataframe tersebut (hanya digunakan untuk dataframe saja).

```
13 # [9] attribute .columns
14 print("[9] attribute .columns")
15 print("    Column matrix_list:", matrix_list.columns)
```

- Output di console untuk penggunaan attribute . columns ini adalah

```
[9] attribute .columns
    Column matrix_list: RangeIndex(start=0, stop=3, step=1)
```

Atribut DataFrame & Series:
10. Attribute .loc[]

- Attribute .loc digunakan slice dataframe atau series berdasarkan nama kolom dan/atau nama index.
- loc mendapat baris (dan / atau kolom) dengan tertentu label .

```
16 # [10] attribute .loc
17 print("[10] attribute .loc")
18 print("    .loc[0:1] pada number_list:", number_list.loc[0:1])
19 print("    .loc[0:1] pada matrix_list:", matrix_list.loc[0:1])
```

- Output di console untuk penggunaan attribute .loc[] ini adalah

```
[10] attribute .loc
    .loc[0:1] pada number_list: 0    1
1    2
dtype: int64
    .loc[0:1] pada matrix_list:    0  1  2
0  1  2  3
1  a  b  c
```

Data awal

Series:

```
0  1
1  2
2  3
3  4
4  5
5  6
dtype: int64
```

DataFrame:

```
   0 1 2
0 1 2 3
1 a b c
2 3 4 5
3 d 4 6
```

Atribut DataFrame & Series:
11. Attribute .iloc[]

- Attribute .iloc digunakan slice dataframe atau series berdasarkan nama kolom dan/atau nama index.
- iloc mendapatkan baris (dan / atau kolom) pada integer lokasi.

```
[8] attribute .index
    Index number_list: RangeIndex(start=0, stop=6, step=1)
    Index matrix_list: RangeIndex(start=0, stop=4, step=1)
[9] attribute .columns
    Column matrix_list: RangeIndex(start=0, stop=3, step=1)
[10] attribute .loc
    .loc[0:1] pada number_list: 0    1
1    2
dtype: int64
    .loc[0:1] pada matrix_list:    0  1  2
0  1  2  3
1  a  b  c
[11] attribute .iloc
    iloc[0:1] pada number_list: 0    1
dtype: int64
    iloc[0:1] pada matrix_list:    0  1  2
0  1  2  3
```

```
import pandas as pd
# Series
number_list = pd.Series([1,2,3,4,5,6])
# DataFrame
matrix_list = pd.DataFrame([[1,2,3],
                            ['a','b','c'],
                            [3,4,5],
                            ['d',4,6]])
# [8] attribute .index
print("[8] attribute .index")
print("    Index number_list:", number_list.index)
print("    Index matrix_list:", matrix_list.index)
# [9] attribute .columns
print("[9] attribute .columns")
print("    Column matrix_list:", matrix_list.columns)
# [10] attribute .loc
print("[10] attribute .loc")
print("    .loc[0:1] pada number_list:",
number_list.loc[0:1])
print("    .loc[0:1] pada matrix_list:",
matrix_list.loc[0:1])
# [11] attribute .iloc
print("[11] attribute .iloc")
print("    iloc[0:1] pada number_list:",
number_list.iloc[0:1])
print("    iloc[0:1] pada matrix_list:",
matrix_list.iloc[0:1])
```

Data awal

Series:

```
0  1
1  2
2  3
3  4
4  5
5  6
dtype: int64
```

DataFrame:

```
   0 1 2
0 1 2 3
1 a b c
2 3 4 5
3 d 4 6
```

```
>>> s = pd.Series(list("abcdef"), index=[49, 48, 47, 0, 1, 2])
49   a
48   b
47   c
0    d
1    e
2    f

>>> s.loc[0]    # value at index label 0
'd'

>>> s.iloc[0]   # value at index location 0
'a'

>>> s.loc[0:1]  # rows at index labels between 0 and 1
(inclusive)
0   d
1   e

>>> s.iloc[0:1] # rows at index location between 0 and 1
(exclusive)
49   a
```

Tugas 4. Atribut series dan dataframe

- Ganti tanda _ _ _ dengan yang sesuai seperti yang diberikan pada contoh di teori.
- Pastikan bahwa urutan penggunaan attribute sesuai dengan yang dicontohkan, yaitu .index, .columns, .loc[], dan .iloc[].
- Attribute .columns hanya dapat digunakan pada pandas Dataframe.

```
import pandas as pd
# Series
number_list = pd.Series([1,2,3,4,5,6])
# DataFrame
matrix_list = pd.DataFrame([[1,2,3], ['a','b','c'],
                            [3,4,5], ['d',4,6]])
# [8] attribute .index
print("[8] attribute .index")
print("    Index number_list:", ___)
print("    Index matrix_list:", ___)
# [9] attribute .columns
print("[9] attribute .columns")
print("    Column matrix_list:", ___)
# [10] attribute .loc
print("[10] attribute .loc")
print("    .loc[0:1] pada number_list:", ___)
print("    .loc[0:1] pada matrix_list:", ___)
# [11] attribute .iloc
print("[11] attribute .iloc")
```

Tugas 4. Atribut series dan dataframe

Hasil yang diharapkan

```
[8] attribute .index
    Index number_list: RangeIndex(start=0, stop=6, step=1)
    Index matrix_list: RangeIndex(start=0, stop=4, step=1)
[9] attribute .columns
    Column matrix_list: RangeIndex(start=0, stop=3, step=1)
[10] attribute .loc
    .loc[0:1] pada number_list: 0    1
                                1    2
dtype: int64
    .loc[0:1] pada matrix_list:    0  1  2
                                   0  1  2  3
                                   1  a  b  c
[11] attribute .iloc
    iloc[0:1] pada number_list: 0    1
dtype: int64
    iloc[0:1] pada matrix_list:    0  1  2
                                   0  1  2  3
```

Tugas 5. Atribut series dan dataframe

- Diberikan dataframe

```
matrix = [[1,2,3],
          ['a','b','c'],
          [3,4,5],
          ['d',4,6]]
matrix_list = pd.DataFrame(matrix)
```

- Apakah yang akan dihasilkan untuk command berikut

```
matrix_list.iloc[0:2,2].to_list()
```

Membuat Series dan Dataframe dari List

- Untuk membuat Series atau Dataframe bisa dari berbagai macam tipe data container/mapping di python, seperti:
  - list
  - dictionary
  - numpy array

Creating Series from List

- Membuat Series yang bersumber dari list.
- List merupakan sebuah kumpulan data berbagai macam tipe data, yang mutable (dapat diganti).
- Contoh membuat series dari **list**

```
1 import pandas as pd
2 # Creating series from list
3 ex_list = ['a',1,3,5,'c','d']
4 ex_series = pd.Series(ex_list)
5 print(ex_series)
```

Output:

```
0    a
1    1
2    3
3    5
4    c
5    d
dtype: object
```

Creating Dataframe from List

- Contoh membuat dataframe dari **list of list**:
- List merupakan sebuah kumpulan data berbagai macam tipe data, yang mutable (dapat diganti).

```
# Creating dataframe from list of list
ex_list_of_list = [[1   ,'a','b' ,'c'],
                   [2.5,'d','e' ,'f'],
                   [5   ,'g','h' ,'i'],
                   [7.5,'j',10.5,'l']]
index = ['dq','lab','kar','lan']
cols = ['float','char','obj','char']
ex_df = pd.DataFrame(ex_list_of_list, index=index, columns=cols)
print(ex_df)
```

- Output

```
     float char   obj char
dq     1.0    a     b    c
lab    2.5    d     e    f
kar    5.0    g     h    i
lan    7.5    j  10.5    l
```

Tugas 6. Membuat Series & Dataframe dari List

- Ganti tanda _ _ _ dengan yang sesuai seperti yang diberikan pada contoh di teori.
- Untuk membuat series dapat dilakukan dengan menggunakan pd.Series yang diisikan dengan sebuah list, serta data frame dapat dibuat dengan pd.DataFrame.
- Membuat dataframe dengan pandas dapat dilakukan dengan menerapkan pd.DataFrame, dengan argumen berupa:
  - data yang dapat dibentuk dari list of list
  - index merupakan nama dari masing-masing baris pada dataframe nantinya
  - columns merupakan nama dari masing-masing kolom dataframe nantinya.
- Perlu diingat bahwa panjang index dan kolom harus sesuai dengan ukuran data yang diinputkan ke dalam pd.DataFrame.

```
import pandas as pd
# Creating series from list
ex_list = ['a',1,3,5,'c','d']
ex_series = pd.___(___)
print(___)
# Creating dataframe from list of list
ex_list_of_list = [[___],
                   [___],
                    [___],
                    [___]]
index = [___]
cols = [___]
ex_df = pd.___(___, index=___, columns=___)
print(___)
```

Membuat Series dari Dictionary

- Membuat Series yang bersumber dari **dictionary**.
- Dictionary merupakan kumpulan data yang strukturnya terdiri dari key dan value.
- Contoh membuat series dari **dictionary:**

```
1 import pandas as pd
2 # Creating series from dictionary
3 dict_series = {'1':'a',
4                '2':'b',
5                '3':'c'}
6 ex_series = pd.Series(dict_series)
7 print(ex_series)
```

- Output :

```
1   a
2   b
3   c
dtype: object
```

Membuat Dataframe dari Dictionary

- Membuat dataframe yang bersumber dari **dictionary** dengan setiap pasangan key dan value-nya berisi list yang sama panjangnya.
- Dictionary merupakan kumpulan data yang strukturnya terdiri dari key dan value.
- Contoh membuat series dari **dictionary:**

```
# Creating dataframe from dictionary
df_series = {'1':['a','b','c'],
             '2':['b','c','d'],
             '4':[2,3,'z']}
ex_df = pd.DataFrame(df_series)
print(ex_df)
```

- Output :

```
   1  2  4
0  a  b  2
1  b  c  3
2  c  d  z
```

Tugas 7. Membuat Series & Dataframe dari Dictionary

- Ganti tanda _ _ _ dengan yang sesuai seperti yang diberikan pada contoh di teori.
- Pastikan mengentrikan pasangan key: value dictionary sesuai dengan yang dicontohkan baik untuk series dan dataframe.
- Penting diingat dalam menggunakan dictionary untuk:
  - series: key-nya akan digunakan sebagai index, dan value-nya harus berupa nilai tunggal sebagai isi dari series.
  - dataframe: key-nya akan secara otomatis jadi columns, serta value-nya akan menjadi isi dari dataframe yang dapat dibuat dari tipe data container atau array numpy (jika panjang dari masing-masing value tidak sama maka kekosongan akan diisi oleh np.nan. Untuk index dari data frame akan dibuat secara otomatis dari 0 (nol) hingga panjang value terpanjang - 1

```
import pandas as pd
# Creating series from dictionary
dict_series = {___,
               ___,
               ___}
ex_series = pd.___(___)
print(___)
# Creating dataframe from dictionary
df_series = {___,
             ___,
             ___}
ex_df = pd.___(___)
print(___)
```

Membuat Series dari Numpy Array

- Membuat Series yang bersumber dari **numpy array.**
- Numpy array kumpulan data yang terdiri atas berbagai macam tipe data,
- Numpy array bersifat mutable.
- Numpy array dibungkus dalam array oleh library Numpy.
- Contoh membuat series dari **numpy array 1D:**

```
1 import pandas as pd
2 import numpy as np
3 # Creating series from numpy array (1D)
4 arr_series = np.array([1,2,3,4,5,6,6,7])
5 ex_series = pd.Series(arr_series)
```

- Output:

```
0   1
1   2
2   3
3   4
4   5
5   6
6   6
7   7
```

Membuat Dataframe dari Numpy Array

- Contoh membuat dataframe dari **numpy array 2D**:

```
7  # Creating dataframe from numpy array (2D)
8  arr_df = np.array([[1  ,2  ,3  ,5],
9                     [5  ,6  ,7  ,8],
10                    ['a','b','c',10]])
11 ex_df = pd.DataFrame(arr_df)
12 print(ex_df)
```

- Output:

```
   0  1  2   3
0  1  2  3   5
1  5  6  7   8
2  a  b  c  10
```

Tugas 8. Membuat Series & Dataframe dari Numpy Array

- Ganti tanda _ _ _ dengan yang sesuai seperti yang diberikan pada contoh di teori.
- Pastikan mengentrikan pasangan key: value dictionary sesuai dengan yang dicontohkan baik untuk series dan dataframe.
- Pastikan bahwa untuk membuat pandas series harus menggunakan numpy array 1D.
- Pastikan bahwa untuk membuat pandas dataframe harus menggunakan numpy array 2D.

```
import pandas as pd
import numpy as ___

# Creating series from numpy array (1D)
arr_series = np.___(___)
ex_series = pd.___(___)
print(___)
# Creating dataframe from numpy array (2D)
arr_df = np.array([[___],
                   [___],
                   [___]])
ex_df = pd.___(___)
print(___)
```

Tugas 9. Mengubah data

- Diberikan dataframe

```
arr_df = np.array([[1   ,2   ,3,5],
                   [5   ,6   ,7,8],
                   ['a','b',9,10]])
df = pd.DataFrame(arr_df)
```

- Bagaimana sintaks mengubah data yang berupa string menjadi angka misal 'a' menjadi 11 dan 'b' menjadi 12?

Dataset I/O

Mempelajari bagaimana membaca dan menyimpan dataset dari dan ke beberapa tipe file dengan menggunakan pandas.

Pendahuluan

- Pandas menyediakan berbagai method untuk membaca file tersebut hanya dengan dipanggil method tersebut, sehingga kode menjadi simple.
- Output pembacaan file dapat berupa Series atau Dataframe.
- Terdapat banyak file yang dapat dibaca/disimpan oleh Pandas, tapi ada beberapa file yang paling umum dan sering digunakan oleh praktisi data seperti berikut ini:
  - CSV (Comma Separated Values), antar data dalam satu baris dipisahkan oleh comma, ",".
  - TSV (Tab Separated Values), antar data dalam satu baris dipisahkan oleh "Tab".
  - Excel
  - Google BigQuery
  - SQL Query
  - JSON (Java Script Object Notation)

Read Dataset - CSV dan TSV

- CSV dan TSV adalah tipe data text dengan perbedaan terletak pada pemisah antar data dalam satu baris.
  - Pada file CSV, antar data dalam satu baris dipisahkan oleh comma, ",".
  - Pada file TSV antar data dalam satu baris dipisahkan oleh "Tab".
- Fungsi **.read_csv()** digunakan untuk membaca file yang value nya dipisahkan oleh comma (default), terkadang pemisah value nya bisa di set ‘\t’ untuk file tsv (tab separated values).

Membaca file CSV dan TSV

- Kode:

```
import pandas as pd

# Membaca file CSV
df_csv =  pd.read_csv(“/path/sample_csv.csv”)
# Membaca file TSV
df_tsv =  pd.read_csv(“/path/sample_tsv.tsv”)
# Menampilkan 3 data teratas
print(df_tsv.head(3))
```

- Hasil:

```
   order_id  order_date  customer_id  ...    brand quantity item_price
0   1612339  2019-01-01        18055  ...  BRAND_C        4    1934000
1   1612339  2019-01-01        18055  ...  BRAND_V        8     604000
2   1612339  2019-01-01        18055  ...  BRAND_G       12     747000

[3 rows x 9 columns]
```

Tugas 10. Membaca file CSV dan TSV

- Ganti tanda _ _ _ dengan yang sesuai seperti yang diberikan pada contoh di teori.
- Pastikan telah mengimport pandas, dan telah membaca dan menampilkan beberapa baris teratas dari file csv dan tsv.
- Secara default .head() akan menampilkan 5 baris teratas. Dengan mengisikan bilangan bulat positif pada .head() seperti .head(7) maka akan menampilkan 7 baris teratas.

```
import ___ as ___

# File CSV
df_csv = pd.___(___)
# Menampilkan 3 data teratas
print(df_csv.___)

# File TSV
df_tsv = pd.___(___, sep=___)
# Menampilkan 3 data teratas tsv
print(df_tsv.___)
```

Read Dataset - Excel

- File Excel dengan ekstensi *.xls atau *.xlsx cukup banyak digunakan dalam menyimpan data.
- Pandas juga memiliki fitur untuk membaca file excel.
- Fungsi .read_excel() digunakan untuk membaca file excel menjadi dataframe pandas.

Membaca file Excel

- Kode:

```
import pandas as pd

# Membaca file xlsx dengan data di sheet “test”
df_excel =  pd.read_excel(“/path/sample_excel.xlsx”)

# Menampilkan 4 data teratas
print(df_excel.head(4))
```

- Hasil:

```
   order_id  order_date  customer_id  ...    brand quantity item_price
0   1612339  2019-01-01        18055  ...  BRAND_C        4    1934000
1   1612339  2019-01-01        18055  ...  BRAND_V        8     604000
2   1612339  2019-01-01        18055  ...  BRAND_G       12     747000
3   1612339  2019-01-01        18055  ...  BRAND_B       12     450000
4   1612339  2019-01-01        18055  ...  BRAND_G        3    1500000

[5 rows x 9 columns]
```

Read Dataset - JSON

- Method .**read_json()** digunakan untuk membaca URL API yang formatnya JSON dan merubahnya menjadi dataframe pandas.
- Kode:

```
import pandas as pd

# Membaca file JSON
url = “https://storage.googleapis.com/dqlab-dataset/covid2019-api-herokuapp-v2.json”
df_json =  pd.read_json(url)

# Menampilkan 10 data teratas
print(df_json.head(10))
```

	data	dt	ts
0	{'confirmed': 3363056, 'active': 2253244, 'loc...	07-14-2020	1594684800
1	{'confirmed': 1884967, 'active': 520883, 'loca...	07-14-2020	1594684800
10	{'confirmed': 255953, 'active': 77172, 'locati...	07-14-2020	1594684800
100	{'confirmed': 3072, 'active': 1636, 'location'...	07-14-2020	1594684800
101	{'confirmed': 3071, 'active': 2178, 'location'...	07-14-2020	1594684800
102	{'confirmed': 2980, 'active': 1662, 'location'...	07-14-2020	1594684800
103	{'confirmed': 2846, 'active': 762, 'location':...	07-14-2020	1594684800
104	{'confirmed': 2762, 'active': 459, 'location':...	07-14-2020	1594684800
105	{'confirmed': 2646, 'active': 654, 'location':...	07-14-2020	1594684800
106	{'confirmed': 2430, 'active': 1644, 'location'...	07-14-2020	1594684800

Read Dataset - SQL

- Fungsi .**read_sql()** atau **.read_sql_query()** digunakan untuk membaca query dari database dan translate menjadi pandas dataframe.
- Contoh case ini database sqlite.

```
import mysql.connector
import pandas as pd
# Membuat koneksi ke database financial di https://relational.fit.cvut.cz/dataset/Financial
my_conn = mysql.connector.connect(host="relational.fit.cvut.cz",
                                  port=3306,
                                  user="guest",
                                  passwd="relational",
                                  database="financial",
                                  use_pure=True)
# Buatlah query sql untuk membaca tabel loan
my_query = """
SELECT *
FROM loan;
"""
```

Menggunakan .read_sql_query

- Kode

```
15  # Gunakanlah .read_sql_query untuk membaca tabel load tersebut
16  df_loan = pd.read_sql_query(my_query, my_conn)
17  # Tampilkan 5 data teratas
18  df_loan.head()
```

- Output

	loan_id	account_id	date	amount	duration	payments	status
0	4959	2	1994-01-05	80952	24	3373.0	A
1	4961	19	1996-04-29	30276	12	2523.0	B
2	4962	25	1997-12-08	30276	12	2523.0	A
3	4967	37	1998-10-14	318480	60	5308.0	D
4	4968	38	1998-04-19	110736	48	2307.0	C

Menggunakan .read_sql

- Kode

```
19  # Jika menggunakan .read_sql
20  df_loan_ = pd.read_sql(my_query, my_conn)
21  # Tampilkan 5 data teratas
22  df_loan_.head()
```

- Output

	loan_id	account_id	date	amount	duration	payments	status
0	4959	2	1994-01-05	80952	24	3373.0	A
1	4961	19	1996-04-29	30276	12	2523.0	B
2	4962	25	1997-12-08	30276	12	2523.0	A
3	4967	37	1998-10-14	318480	60	5308.0	D
4	4968	38	1998-04-19	110736	48	2307.0	C

Read Dataset - Google BigQuery

- Untuk data yang besar (big data), umumnya digunakan Google BigQuery.
- Layanan ini dapat digunakan jika telah memiliki Google BigQuery account.
- Fungsi .**read_gbq()** digunakan untuk membaca Google BigQuery table menjadi dataframe pandas.

```
import pandas as pd
# Buat query
query = """
SELECT *
FROM `bigquery-public-data.covid19_jhu_csse_eu.summary`
LIMIT 1000;
"""
# Baca data dari Google Big Query
df_covid19_eu_summary = pd.read_gbq(query, project_id="XXXXXXXX")
# Tampilkan 5 data teratas
df_covid19_eu_summary
```

- project_id="XXXXXXXX" adalah ID dari Google BigQuery account.

	province_state	country_region	date	latitude	longitude	location_geom	cc
0	Hubei	Mainland China	2020-02-25	NaN	NaN	None	
1	Guangdong	Mainland China	2020-02-25	NaN	NaN	None	
2	Henan	Mainland China	2020-02-25	NaN	NaN	None	
3	Zhejiang	Mainland China	2020-02-25	NaN	NaN	None	
4	Hunan	Mainland China	2020-02-25	NaN	NaN	None	
...	...	...	...	...	...	...	
995	Shanxi	Mainland China	2020-02-17	NaN	NaN	None	
996	Tianjin	Mainland China	2020-02-17	NaN	NaN	None	
997	Liaoning	Mainland China	2020-02-17	NaN	NaN	None	
998	Gansu	Mainland China	2020-02-17	NaN	NaN	None	
999	Jilin	Mainland China	2020-02-17	NaN	NaN	None	

1000 rows × 13 columns

Write Dataset

- Dalam bekerja sebagai data scientist/analis setelah melakukan data cleaning, dataset menjadi rapi.
- Tahap berikut adalah method untuk menyimpan dataframe/series ke dalam media penyimpanan.

Method	Code

.to_csv()
→ digunakan untuk export dataframe kembali ke csv atau tsv
	CSV
	`df.to_csv("csv1.csv", index=False)`
	TSV
	`df.to_csv("tsv1.tsv", index=False, sep='\t')`

.to_clipboard() → export dataframe menjadi bahan copy jadi nanti bisa tinggal klik paste di excel atau google sheets	`df.to_clipboard()`
.to_excel() → export dataframe menjadi file excel	`df_excel.to_excel("xlsx1.xlsx", index=False)`
.to_gbq() → export dataframe menjadi table di Google BigQuery	`df.to_gbq("temp.test", project_id="XXXXXX", if_exists="fail")` **temp:** nama dataset, **test:** nama table **if_exists:** ketika tabel dengan dataset.table_name yang sama sudah ada, apa action yang ingin dilakukan (**"fail":** tidak melakukan apa-apa, **"replace':** membuang tabel yang sudah ada dan mengganti yang baru, **"append":** menambah baris di tabel tersebut dengan data yang baru)

Head & Tail

- Seperti yang telah dipelajari sebelumnya bahwa ada method **.head** yang diterapkan pada suatu variabel bertipe pandas dataframe/series.
- Method **.head** ditujukan untuk membatasi tampilan jumlah baris teratas dari dataset.
- Sementara itu, method **.tail** ditujukan untuk membatasi jumlah baris terbawah dari dataset.
- Secara umum kedua method ini memiliki bentuk

```
[nama_dataframe].head(n)
```

dan

```
[nama_dataframe].tail(n)
```

- dengan n merupakan jumlah baris yang akan ditampilkan, jika tidak disebutkan n = 5 (sebagai nilai default dari n).

Tugas 11. Head & Tail

- Berdasarkan file sample_csv.csv cetaklah 3 data teratas dan 3 data terbawah.
- Jika berhasil maka tampilan berikut yang akan kamu peroleh di console.

```
import ___

# Baca file sample_csv.csv
df = ___

# Tampilkan 3 data teratas
print("Tiga data teratas:\n", ___)

# Tampilkan 3 data terbawah
print("Tiga data terbawah:\n", ___)
```

```
Tiga data teratas:
    order_id  order_date  customer_id  ...    brand quantity item_price
0   1612339  2019-01-01        18055  ...  BRAND_C        4    1934000
1   1612339  2019-01-01        18055  ...  BRAND_V        8     604000
2   1612339  2019-01-01        18055  ...  BRAND_G       12     747000

[3 rows x 9 columns]
Tiga data terbawah:
     order_id  order_date  customer_id  ...    brand quantity item_price
98    1612390  2019-01-01        12681  ...  BRAND_S       24     450000
99    1612390  2019-01-01        12681  ...  BRAND_S       24     450000
100   1612390  2019-01-01        12681  ...  BRAND_B        4    1325000

[3 rows x 9 columns]
```

Indexing, Slicing, dan Transforming

Mempelajari bagaimana menerapkan indexing, slicing, dan transforming suatu dataframe.

Kegiatan ini sering digunakan dalam bidang data terutama untuk proses filter dan merubah tipe data yang ada.

Indexing (1)

- Index merupakan key identifier dari tiap row/column untuk Series atau Dataframe (sifatnya tidak mutable untuk masing-masing value tapi bisa diganti untuk semua value sekaligus).
- Jika tidak disediakan, pandas akan membuat kolom index default secara otomatis sebagai bilangan bulat (integer) dari 0 sampai range jumlah baris data tersebut.
- Kolom index dapat terdiri dari
  1. satu kolom (*single index*), atau
  2. multiple kolom (disebut dengan *hierarchical indexing*).
- Index dengan multiple kolom ini terjadi karena unique identifier tidak dapat dicapai hanya dengan set index di 1 kolom saja sehingga membutuhkan beberapa kolom yang menjadikan tiap row menjadi unique.

Indexing (2)

- Secara default setelah suatu data frame dibaca dari file dengan format tertentu, index-nya merupakan single index.
- Cetak index dan kolom yang dimiliki oleh file “/path/sample_csv.csv".
- Untuk menentukan index dan kolom yang dimiliki oleh dataset yang telah dinyatakan sebagai sebuah dataframe pandas dapat dilakukan dengan menggunakan attribut .index dan .columns.

Menampilkan **index** dan **kolom** data teratas dari file CSV

```
import pandas as pd

# Baca file CSV sample_csv.csv
df = pd.read_csv(“/path/sample_csv.csv", sep="\t")

# Index dari df
print("Index:", df.index)

# Column dari df
print("Columns:", df.columns)
```

```
Index   : RangeIndex(start=0, stop=101, step=1)
Columns: Index(['order_id', 'order_date', 'customer_id', 'city', 'province',
       'product_id', 'brand', 'quantity', 'item_price'],
      dtype='object')
```

Tugas 12. Tampilkanlah index dan kolom data teratas dari file TSV

- Tampilkanlah **index** dan **kolom** data teratas dari file TSV (sample_tsv.tsv)
- Hasil yang diharapkan

```
Index   : RangeIndex(start=0, stop=101, step=1)
Columns: Index(['order_id', 'order_date', 'customer_id', 'city', 'province',
       'product_id', 'brand', 'quantity', 'item_price'],
      dtype='object')
```

Indexing (3): Hierarchical indexing (Multi index)

- Sebelumnya telah dibahas terkait single index, kali ini akan bahas multi index atau disebut juga dengan ***hierarchical indexing*.**
- Untuk membuat multi index (*hierarchical indexing*) dengan pandas diperlukan kolom-kolom mana saja yang perlu disusun agar index dari data frame menjadi sebuah hirarki yang kemudian dapat dikenali.
- Sebelumnya telah diberikan nama-nama kolom dari dataframe yang telah dibaca, yaitu:

```
import pandas as pd
# Baca file CSV sample_csv.csv
df = pd.read_csv("/path/sample_tsv.tsv", sep="\t")

# Index dari df
print("Index:", df.index)

# Column dari df
print("Columns:", df.columns)
```

```
Index: RangeIndex(start=0, stop=101, step=1)
Columns: Index(['order_id', 'order_date', 'customer_id', 'city', 'province',
       'product_id', 'brand', 'quantity', 'item_price'],
      dtype='object')
```

Indexing (3): Hierarchical indexing (Multi index)

- Selanjutnya akan dibuat multi index dengan menggunakan kolom 'order_id', 'customer_id', 'product_id', dan 'order_date' dengan menggunakan method **.set_index()**.
- Kode

```
import pandas as pd

# Baca file TSV sample_tsv.tsv
df = pd.read_csv("/path/sample_tsv.tsv", sep="\t")

# Set multi index df
df_x = df.set_index(['order_id', 'customer_id', 'product_id', 'order_date'])

# Print nama dan level dari multi index
for name, level in zip(df_x.index.names, df_x.index.levels):
    print(name,':',level)
```

Berikut hasil tampilan nama dan level dari multi index
(multi index dengan menggunakan kolom 'order_id', 'customer_id', 'product_id', dan 'order_date')

```
order_id : Int64Index([1612339, 1612342, 1612345, 1612369, 1612372, 1612375, 1612378,
            1612381, 1612384, 1612387, 1612390],
           dtype='int64', name='order_id')
customer_id : Int64Index([12681, 13963, 15649, 17091, 17228, 17450, 17470, 17511, 17616,
            18055],
           dtype='int64', name='customer_id')
product_id : Index(['P0029', 'P0040', 'P0041', 'P0116', 'P0117', 'P0219', 'P0255', 'P0327',
       'P0422', 'P0449', 'P0491', 'P0515', 'P0517', 'P0520', 'P0525', 'P0535',
       'P0648', 'P0704', 'P0708', 'P0709', 'P0784', 'P0791', 'P0792', 'P0931',
       'P0968', 'P1118', 'P1251', 'P1255', 'P1305', 'P1306', 'P1307', 'P1329',
       'P1342', 'P1469', 'P1508', 'P1513', 'P1597', 'P1600', 'P1679', 'P1680',
       'P1683', 'P1684', 'P1685', 'P1700', 'P1741', 'P1742', 'P1780', 'P1800',
       'P1813', 'P1867', 'P1945', 'P2086', 'P2089', 'P2103', 'P2154', 'P2159',
       'P2325', 'P2494', 'P2556', 'P2575', 'P2594', 'P2641', 'P2660', 'P2707',
       'P2760', 'P2783', 'P2806', 'P2866', 'P2881', 'P2928', 'P2935', 'P2946',
       'P2964', 'P3006', 'P3082', 'P3122', 'P3132', 'P3354', 'P3357', 'P3362',
       'P3388', 'P3409', 'P3412', 'P3484', 'P3599', 'P3665', 'P3826', 'P3837',
       'P3911', 'P4009', 'P4025', 'P4057', 'P4059', 'P4060', 'P4075'],
      dtype='object', name='product_id')
order_date : Index(['2019-01-01'], dtype='object', name='order_date')
```

- Dari multi index tersebut terdapat atribut levels yang menunjukkan urutan index, dalam case ini 'order_id' > 'customer_id' > 'product_id' > 'order_date'.

```
# Print nama dan level dari multi index
for name, level in zip(df_x.index.names, df_x.index.levels):
    print(name,':',level)
```

- Output

```
order_id : Int64Index([1612339, 1612342, 1612345, 1612369, 1612372, 1612375, 1612378,
            1612381, 1612384, 1612387, 1612390],
           dtype='int64', name='order_id')
customer_id : Int64Index([12681, 13963, 15649, 17091, 17228, 17450, 17470, 17511, 17616,
            18055],
           dtype='int64', name='customer_id')
product_id : Index(['P0029', 'P0040', 'P0041', 'P0116', 'P0117', 'P0219', 'P0255', 'P0327',
       'P0422', 'P0449', 'P0491', 'P0515', 'P0517', 'P0520', 'P0525', 'P0535',
       'P0648', 'P0704', 'P0708', 'P0709', 'P0784', 'P0791', 'P0792', 'P0931',
       'P0968', 'P1118', 'P1251', 'P1255', 'P1305', 'P1306', 'P1307', 'P1329',
       'P1342', 'P1469', 'P1508', 'P1513', 'P1597', 'P1600', 'P1679', 'P1680',
       'P1683', 'P1684', 'P1685', 'P1700', 'P1741', 'P1742', 'P1780', 'P1800',
       'P1813', 'P1867', 'P1945', 'P2086', 'P2089', 'P2103', 'P2154', 'P2159',
       'P2325', 'P2494', 'P2556', 'P2575', 'P2594', 'P2641', 'P2660', 'P2707',
       'P2760', 'P2783', 'P2806', 'P2866', 'P2881', 'P2928', 'P2935', 'P2946',
       'P2964', 'P3006', 'P3082', 'P3122', 'P3132', 'P3354', 'P3357', 'P3362',
       'P3388', 'P3409', 'P3412', 'P3484', 'P3599', 'P3665', 'P3826', 'P3837',
       'P3911', 'P4009', 'P4025', 'P4057', 'P4059', 'P4060', 'P4075'],
      dtype='object', name='product_id')
order_date : Index(['2019-01-01'], dtype='object', name='order_date')
```

- Untuk melihat dataframe **df_x**-nya (multi index yang telah diset) dapat dilakukan dengan

```
5 # Index dari df_x
6 print("Index df_x:", df_x.index)
```

- Kumpulan index dari multi index adalah list dari banyak tuples, tuples nya merupakan kombinasi yang ada dari gabungan index-index tersebut.

```
Index df_x:  MultiIndex([(1612339, 18055, 'P0648', '2019-01-01'),
            (1612339, 18055, 'P3826', '2019-01-01'),
            (1612339, 18055, 'P1508', '2019-01-01'),
            (1612339, 18055, 'P0520', '2019-01-01'),
            (1612339, 18055, 'P1513', '2019-01-01'),
            (1612339, 18055, 'P3911', '2019-01-01'),
            (1612339, 18055, 'P1780', '2019-01-01'),
            (1612339, 18055, 'P3132', '2019-01-01'),
            (1612339, 18055, 'P1342', '2019-01-01'),
            (1612339, 18055, 'P2556', '2019-01-01'),
            ...
            (1612387, 17228, 'P0535', '2019-01-01'),
            (1612387, 17228, 'P0029', '2019-01-01'),
            (1612387, 17228, 'P3362', '2019-01-01'),
            (1612387, 17228, 'P3409', '2019-01-01'),
            (1612390, 12681, 'P1867', '2019-01-01'),
            (1612390, 12681, 'P3388', '2019-01-01'),
            (1612390, 12681, 'P3082', '2019-01-01'),
            (1612390, 12681, 'P3354', '2019-01-01'),
            (1612390, 12681, 'P3357', '2019-01-01'),
            (1612390, 12681, 'P0422', '2019-01-01')],
           names=['order_id', 'customer_id', 'product_id', 'order_date'], length=101)
```

Tugas 12. **Multi index** dari file TSV

- Tampilkan **multi index** dari file TSV "sample_tsv.tsv" yang telah dibaca berupa nama dan level indexnya.
- Kolom yang menjadi indexnya yaitu 'order_date', 'city', dan 'customer_id'!
- Pastikan menuliskan urutan dari kolom-kolom yang menjadi index baru dataframe dan kemudian menempatkannya ke dalam looping untuk diinspeksi kembali nama dan level multi index yang telah diset.

```
___
# Baca file TSV sample_tsv.tsv
df = pd.___(___, ___)

# Set multi index df
df_x = ___([___, ___, ___])

# Print nama dan level dari multi index
for ___, ___ in zip(df_x.___.___,
df_x.___.___):
    print(___,':',___)
```

```
order_date : Index(['2019-01-01'], dtype='object', name='order_date')
city : Index(['Bogor', 'Jakarta Pusat', 'Jakarta Selatan', 'Jakarta Utara',
       'Makassar', 'Malang', 'Surabaya', 'Tangerang'],
      dtype='object', name='city')
customer_id : Int64Index([12681, 13963, 15649, 17091, 17228, 17450, 17470, 17511, 17616,
            18055],
           dtype='int64', name='customer_id')
```

Indexing (4)

- Terdapat beberapa cara untuk membuat index, salah satunya adalah seperti yang telah dilakukan pada sub bab sebelumnya dengan menggunakan method **.set_index().**
- Di sub bab ini akan menggunakan assignment untuk menset index dari suatu data frame.

```
1 import pandas as pd
2 # Buat data frame
3 df_week = pd.DataFrame({
4    "day_number":[1,2,3,4,5,6,7],
5    "week_type":["weekday" for i in range(5)] + ["weekend" for i in range(2)]})
6 # Definisikan indexnya dan assign
7 df_week.index = ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
8 print(df_week)
```

	day_number	week_type
Mon	1	weekday
Tue	2	weekday
Wed	3	weekday
Thu	4	weekday
Fri	5	weekday
Sat	6	weekend
Sun	7	weekend

- Cara yang ditunjukkan oleh baris ketujuh (ke 7) pada kode di atas hanya berlaku jika index yang diassign tersebut memiliki panjang yang sama dengan jumlah baris dari dataframe.
- Jika ingin kembalikan dataframe ke index defaultnya yaitu dari 0 s/d jumlah baris data - 1, maka dapat menggunakan method **.reset_index(drop=True)**, argument drop=True bertujuan untuk menghapus index lama.

Tugas 13. method .set_index().

- Baca file TSV "sample_tsv.tsv" hanya untuk 10 baris pertama.
- Set lah indeknya dengan menggunakan nama "Pesanan ke-i" i adalah bilangan bulat dari 1 sampai dengan jumlah baris (10 baris data).

```
import ___

# Baca file sample_tsv.tsv untuk 10 baris pertama saja
df = pd.read_csv(___, ___, nrows=___)

# Cetak data frame awal
print("Dataframe awal:\n", ___)

# Set index baru
df.___ = ["___" + str(___) for i in range(___, ___)]

# Cetak data frame dengan index baru
print("Dataframe dengan index baru:\n", ___)
```

```
Dataframe awal:
    order_id  order_date  customer_id  ...    brand quantity item_price
0   1612339  2019-01-01        18055  ...  BRAND_C        4    1934000
1   1612339  2019-01-01        18055  ...  BRAND_V        8     604000
2   1612339  2019-01-01        18055  ...  BRAND_G       12     747000
3   1612339  2019-01-01        18055  ...  BRAND_B       12     450000
4   1612339  2019-01-01        18055  ...  BRAND_G        3    1500000
5   1612339  2019-01-01        18055  ...  BRAND_V        3    2095000
6   1612339  2019-01-01        18055  ...  BRAND_H        3    2095000
7   1612339  2019-01-01        18055  ...  BRAND_S        3    1745000
8   1612339  2019-01-01        18055  ...  BRAND_F        6    1045000
9   1612339  2019-01-01        18055  ...  BRAND_P        6    1045000

[10 rows x 9 columns]
Dataframe denga index baru:
             order_id  order_date  customer_id  ...    brand quantity item_price
Pesanan ke-1   1612339  2019-01-01        18055  ...  BRAND_C        4    1934000
Pesanan ke-2   1612339  2019-01-01        18055  ...  BRAND_V        8     604000
Pesanan ke-3   1612339  2019-01-01        18055  ...  BRAND_G       12     747000
Pesanan ke-4   1612339  2019-01-01        18055  ...  BRAND_B       12     450000
Pesanan ke-5   1612339  2019-01-01        18055  ...  BRAND_G        3    1500000
Pesanan ke-6   1612339  2019-01-01        18055  ...  BRAND_V        3    2095000
Pesanan ke-7   1612339  2019-01-01        18055  ...  BRAND_H        3    2095000
Pesanan ke-8   1612339  2019-01-01        18055  ...  BRAND_S        3    1745000
Pesanan ke-9   1612339  2019-01-01        18055  ...  BRAND_F        6    1045000
Pesanan ke-10  1612339  2019-01-01        18055  ...  BRAND_P        6    1045000

[10 rows x 9 columns]
```

Indexing (5)

- Jika file yang akan dibaca melalu penggunaan library pandas dapat dipreview terlebih dahulu struktur datanya maka melalui fungsi yang ditujukan untuk membaca file dapat diset mana kolom yang akan dijadikan index.
- Fitur ini telah dimiliki oleh setiap fungsi yang digunakan dalam membaca data dengan pandas, yaitu penggunaan argumen **index_col** pada fungsi yang dimaksud.
- Perhatikan pada kode berikut:

```
import pandas as pd

# Baca file sample_tsv.tsv dan set lah index_col sesuai
instruksi
df = pd.read_csv(“/path/sample_tsv.tsv", sep="\t",
index_col=["order_date","order_id"])

# Cetak data frame untuk 8 data teratas
print("Dataframe:\n", df.head(8))
```

- Dari dataset sample_csv.csv, sample_tsv.tsv, atau sample_excel.xlsx sudah tahu bahwa kolom dataset adalah:
  1. 'order_id';
  2. 'order_date';
  3. 'customer_id';
  4. 'city';
  5. 'province';
  6. 'product_id';
  7. 'brand';
  8. 'quantity'; and
  9. 'item_price'.

```
Dataframe:
                     customer_id            city  ... quantity item_price
order_date order_id                                ...
2019-01-01 1612339         18055  Jakarta Selatan  ...        4    1934000
           1612339         18055  Jakarta Selatan  ...        8     604000
           1612339         18055  Jakarta Selatan  ...       12     747000
           1612339         18055  Jakarta Selatan  ...       12     450000
           1612339         18055  Jakarta Selatan  ...        3    1500000
           1612339         18055  Jakarta Selatan  ...        3    2095000
           1612339         18055  Jakarta Selatan  ...        3    2095000
           1612339         18055  Jakarta Selatan  ...        3    1745000

[8 rows x 7 columns]
```

- Sehingga kode di atas digunakan langsung kolom 'order_date' pada saat membaca filenya.
- Terlihat bahwa kolom order_date dan order_id sudah jadi index, dan tentunya jumlah kolom dataframe berkurang dua, yaitu menjadi tujuh kolom.

Tugas 14. method .set_index().

- Baca file TSV "sample_tsv.tsv" dan set lah kolom "order_date" dan "order_id" sebagai index_col-nya dan cetaklah dataframe untuk delapan baris pertama.
- Pastikan untuk kolom indeksnya adalah kolom "order_date" dan "order_id" yang ditulis secara berurut dan dicasting di dalam sebuah list.

```
import ___
# Baca file sample_tsv.tsv dan set lah
index_col sesuai instruksi
df = ___(___, ___, index_col=___)

# Cetak data frame untuk 8 data teratas
print("Dataframe:\n", ___)
```

```
Dataframe:
                     customer_id            city  ... quantity item_price
order_date order_id                               ...
2019-01-01 1612339         18055  Jakarta Selatan  ...        4    1934000
           1612339         18055  Jakarta Selatan  ...        8     604000
           1612339         18055  Jakarta Selatan  ...       12     747000
           1612339         18055  Jakarta Selatan  ...       12     450000
           1612339         18055  Jakarta Selatan  ...        3    1500000
           1612339         18055  Jakarta Selatan  ...        3    2095000
           1612339         18055  Jakarta Selatan  ...        3    2095000
           1612339         18055  Jakarta Selatan  ...        3    1745000

[8 rows x 7 columns]
```

Slicing (1)

- Seperti artinya slicing adalah cara untuk melakukan filter ke dataframe/series berdasarkan kriteria tertentu dari nilai kolomnya ataupun kriteria index-nya.
- Terdapat 2 cara paling terkenal untuk slicing dataframe, yaitu dengan menggunakan method **.loc** dan **.iloc** pada variabel bertipe pandas DataFrame/Series.
- Method **.iloc** ditujukan untuk proses slicing berdasarkan index berupa nilai integer tertentu.
- Akan tetapi akan lebih sering menggunakan dengan method **.loc** karena lebih fleksibel.

Misal:

Ikuti ilustrasi berikut ini:

- Dataset belum dilakukan indexing, jadi slicing berdasarkan nilai kolomnya.
- Untuk itu "sample_csv.csv" dibaca kembali dan dipraktikkan metode .loc[] dengan mengambil tanggal 1 Januari 2019 dari kolom **order_date** dan **product_id** nya adalah P2154 dan P2556.

```
import pandas as pd
# Baca file sample_csv.csv
df = pd.read_csv(“/path/sample_csv.csv")
# Slice langsung berdasarkan kolom
df_slice = df.loc[(df["order_date"] == "2019-01-01") &
                  (df["product_id"].isin(["P2154", "2556"]))
             ]
print("Slice langsung berdasarkan kolom:\n", df_slice)
```

```
Slice langsung berdasarkan kolom:
     order_id  order_date  customer_id  ...    brand quantity item_price
10   1612339  2019-01-01        18055  ...  BRAND_M        4    1745000

[1 rows x 9 columns]
```

Tugas 15. Slicing (1)

- Baca file TSV "sample_csv.csv" dan slice/filter-lah dataset jika customer_id adalah 18055 dan product_id-nya yaitu P0029, P0040, P0041, P0116, dan P0117.
- Pastikan telah melakukan slicing langsung dengan menggunakan kolom customer_id sesuai dengan nilai id-nya dan kolom product_id sejumlah 5 id.

```
import ___

# Baca file sample_csv.csv
df = ___

# Slice langsung berdasarkan kolom
df_slice = df.loc[(___ == ___) &
                  (___.___([___]))
                ]
print("Slice langsung berdasarkan kolom:\n", ___)
```

```
Slice langsung berdasarkan kolom:
 Empty DataFrame
Columns: [order_id, order_date, customer_id, city, province, product_id, brand, quantity, item_price]
Index: []
```

Slicing (2)

- Dalam bagian sebelumnya telah mempelajari bagaimana menslicing/filtering dataset dengan menggunakan method **.loc** pada kolom dataset.
- Sekarang, menerapkan berdasarkan index.
- Syaratnya adalah dataset sudah dilakukan indexing terlebih dahulu melalui penerapan method **.set_index**.
- **CARA 1:** Gunakan method .loc seperti yang dicontohkan berikut

```
import pandas as pd

# Baca file sample_csv.csv
df = pd.read_csv("/path/sample_csv.csv")

# Set index dari df sesuai instruksi
df = df.set_index(["order_date","product_id"])

# Cara 1: Gunakan .loc
df_slice1 = df.loc[("2019-01-01",["P2154","P2156"]),:]
print("Cara 1:\n", df_slice1)
```

```
Cara 1:
                        order_id  customer_id  ... quantity item_price
order_date product_id                          ...
2019-01-01 P2154         1612339        18055  ...        4    1745000

[1 rows x 7 columns]
```

- **Cara 2**: Gunakan pd.IndexSlice sebagai varibel untuk melakukan slicing index

```
# Cara 2: Gunakan pd.Indexslice dan terapkan dengan .loc
idx = pd.IndexSlice
df_slice2 = df.sort_index().loc[idx["2019-01-01", "P2154":"P2556"], :]
print("Cara 2:\n", df_slice2)
```

```
Cara 2:
                     order_id  customer_id  ... quantity item_price
order_date product_id                       ...
2019-01-01 P2154       1612339        18055  ...        4    1745000

[1 rows x 7 columns]
```

Tugas 15. Slicing (2)

- Baca file TSV "sample_csv.csv" dan set terlebih dahulu indexnya yaitu order_date, order_id, dan product_id.
- Kemudian slice/filter-lah dataset jika order_date adalah 2019-01-01, order_id adalah 1612339 dan product_id-nya yaitu P2154 dan P2159. Gunakanlah cara pertama.
- Pastikan telah melakukan set_index terlebih dahulu sesuai dengan urutan kolom yang diinstruksikan dan kemudian menslice berdasarkan kondisi untuk setiap kolom yang disebutkan.
- Jangan lupa untuk mengimport pandas sebagai alias, membaca file csv-nya dan dalam menerapkan .loc perhatikan tipe data yang akan digunakan sebagai kondisi untuk setiap kolom.

```
import ___
# Baca file sample_csv.csv
df = ___
# Set index dari df sesuai instruksi
df = df.___
# Slice sesuai intruksi
df_slice = df.loc[(___,___,___),:]
print("Slice df:\n", df_slice)
```

```
Slice df:
                                    customer_id  ... item_price
order_date order_id product_id                   ...
2019-01-01 1612339  P2154             18055  ...    1745000
                    P2159             18055  ...     310000

[2 rows x 6 columns]
```

Transforming (1)

- Transform adalah mengubah dataset yang ada menjadi entitas baru, dapat dilakukan dengan
  - konversi dari satu data type ke data type yang lain,
  - transpose dataframe
  - atau yang lainnya.
- Setelah data dibaca, cek tipe data di setiap kolom apakah sesuai dengan representasinya menggunakan attribut .dtypes pada dataframe yang telah dibaca:

```
[nama_dataframe].dtypes
```

- Untuk konversi tipe data, secara default system akan mendeteksi data yang tidak bisa di render as date type or numeric type sebagai object yang basically string.
- Alasan tidak bisa di render oleh system karena berbagai hal, mungkin karena formatnya asing dan tidak dikenali oleh python secara umum (misal: date type data → '2019Jan01').
- Data contoh tersebut tidak bisa di render karena bulannya Jan tidak bisa di translate menjadi in form of number (00-12) dan tidak ada '-' di antara tahun, bulan dan harinya.
- Jika seluruh data pada kolom di order_date sudah tertulis dalam bentuk 'YYYY-MM-DD' maka ketika dibaca, kolom order_date sudah langsung dinyatakan bertipe data datetime.

Transforming (1)

- Untuk merubah kolom date_order yang sebelumnya bertipe object menjadi kolom bertipe datetime, cara pertama yang dapat dilakukan adalah menggunakan

```
pd.to_datetime(argumen)
```

- dengan argumen adalah isi kolom dari dataframe yang akan dirubah tipe datanya, misal dalam format umum

```
nama_dataframe["nama_kolom"]
```

- Sehingga lengkapnya dapat ditulis sebagai

```
nama_dataframe["nama_kolom"] = pd.to_datetime(nama_dataframe["nama_kolom"])
```

Tugas 16. Transforming (1)

- Ubahlah tipe data di kolom order_date yang semula bertipe objek menjadi bertipe datetime.
- Pastikan telah mengecek tipe datanya dengan menggunakan attribut .dtypes dan kemudian mengubah tipe data kolom order_date ke datetime melalui pd.datetime().
- Isi argumen pada pd.datetime(argumen) adalah content pada kolom yang akan diubah.

```
import ___
# Baca file sample_csv.csv
df = ___
# Tampilkan tipe data
print("Tipe data df:\n", ___)
# Ubah tipe data kolom order_date menjadi datetime
df[___] = pd.___(___)
# Tampilkan tipe data df setelah transformasi
print("\nTipe data df setelah transformasi:\n", ___)
```

```
Tipe data df:
 order_id         int64
order_date      object
customer_id      int64
city            object
province        object
product_id      object
brand           object
quantity         int64
item_price       int64
dtype: object

Tipe data df setelah transformasi:
 order_id                  int64
order_date     datetime64[ns]
customer_id               int64
city                     object
province                 object
product_id               object
brand                    object
quantity                  int64
item_price                int64
dtype: object
```

Transforming (2)

- Bagian ini akan mengubah tipe data pada kolom dataframe yang telah dibaca menjadi tipe data float (kolom quantity) dan tipe categori (kolom city).
- Secara umum, untuk merubah ke numerik dapat menggunakan pd.to_numeric(), yaitu

```
nama_dataframe["nama_kolom"] = pd.to_numeric(nama_dataframe["nama_kolom"],
                                             downcast="tipe_data_baru")
```

- Sedangkan untuk menjadi suatu kolom yang dapat dinyatakan sebagai kategory dapat menggunakan method .astype() pada dataframe, yaitu

```
nama_dataframe["nama_kolom"] = nama_dataframe["nama_kolom"].astype("category")
```

Tugas 17. Transforming (2)

- Ubahlah tipe data di kolom
  - quantity yang semula bertipe int64 menjadi bertipe float32, dan
  - city yang semula bertipe object menjadi bertipe category
- Pastikan bahwa kolom quantity diubah menjadi kolom bertipe numerik float serta kolom city menjadi kolom bertipe kategori dengan masing-masingnya menggunakan pd.to_numeric dan .astype().
- Pastikan untuk menbaca data dan mengecek tipe data awal sebelum dan setelah transformasi.

```
import pandas as pd
# Baca file sample_csv.csv
df = pd.read_csv("/path/sample_csv.csv")

# Tampilkan tipe data
print("Tipe data df:\n", df.dtypes)

# Ubah tipe data kolom quantity menjadi tipe data numerik float
df["quantity"] = pd.to_numeric(df["quantity"], downcast="float")

# Ubah tipe data kolom city menjadi tipe data category
df["city"] = df["city"].astype("category")

# Tampilkan tipe data df setelah transformasi
print("\nTipe data df setelah transformasi:\n", df.dtypes)
```

```
Tipe data df:
 order_id        int64
order_date      object
customer_id      int64
city            object
province        object
product_id      object
brand           object
quantity         int64
item_price       int64
dtype: object

Tipe data df setelah transformasi:
 order_id          int64
order_date       object
customer_id       int64
city           category
province         object
product_id       object
brand            object
quantity        float32
item_price        int64
dtype: object
```

Transforming (3)

- Teknik berikutnya dalam proses transformasi suatu dataframe adalah menggunakan method .apply() dan .map() pada suatu dataframe.
- **Method .apply()** digunakan untuk menerapkan suatu fungsi python (yang dibuat dengan def atau anonymous dengan lambda) pada dataframe/series atau hanya kolom tertentu dari dataframe.
- Contoh mengubah setiap baris pada kolom brand menjadi lowercase.

```
1. import pandas as pd

2. # Baca file sample_csv.csv
3. df = pd.read_csv("/path/sample_csv.csv")

4. # Cetak 5 baris teratas kolom brand
5. print("Kolom brand awal:\n", df["brand"].head())

6. # Gunakan method apply untuk merubah isi kolom menjadi lower case
7. df["brand"] = df["brand"].apply(lambda x: x.lower())

8. # Cetak 5 baris teratas kolom brand
9. print("Kolom brand setelah diubah:\n", df["brand"].head())
```

```
Kolom brand awal:
 0    BRAND_C
1    BRAND_V
2    BRAND_G
3    BRAND_B
4    BRAND_G
Name: brand, dtype: object

Kolom brand setelah diubah:
 0    brand_c
1    brand_v
2    brand_g
3    brand_b
4    brand_g
Name: brand, dtype: object
```

Transforming (3)

- **Method .map()** hanya dapat diterapkan pada series atau dataframe yang diakses satu kolom saja.
- Method ini digunakan untuk mensubstitusikan suatu nilai ke dalam tiap baris datanya.
- Contoh yang diberikan berikut ini akan mengambil huruf terakhir dari brand

```
1. # Gunakan method map untuk mengambil kode brand yaitu karakter terakhirnya
2. df["brand"] = df["brand"].map(lambda x: x[-1])
3. # Cetak 5 baris teratas kolom brand
4. print("Kolom brand setelah map:\n", df["brand"].head())
```

```
Kolom brand setelah map:
 0    c
1    v
2    g
3    b
4    g
Name: brand, dtype: object
```

Tugas 17. Transforming (3)

- Lakukan transforming seperti diinstruksikan pada teori Transforming
- Pastikan telah menerapkan method apply dan map yang masing-masingnya untuk mengubah seluruh isi pada kolom brand menjadi lowercase dan kemudian ambil karakter terakhirnya. Penggunaan anonymous function dengan lambda sangat bermanfaat.
- Baca data sample_csv.csv, dan cetak 5 baris teratas kolom brand untuk masing-masing langkah yang telah diinstruksikan.

```
import ___
# Baca file sample_csv.csv
df = ___
# Cetak 5 baris teratas kolom brand
print("Kolom brand awal:\n", ___)
# Gunakan method apply untuk merubah isi kolom menjadi lower case
df["brand"] = ___.___(___ x: ___)
# Cetak 5 baris teratas kolom brand
print("Kolom brand setelah apply:\n", ___)
# Gunakan method map untuk mengambil kode brand yaitu karakter terakhirnya
___ = ___.___(___: ___)
# Cetak 5 baris teratas kolom brand
print("Kolom brand setelah map:\n", ___.head())
```

Transforming (4)

- Di bahasan sebelumnya diketahui bahwa map hanya dapat digunakan untuk pandas series.
- Pada bagian ini akan digunakan method .applymap pada dataframe.

```
1 import numpy as np
2 import pandas as pd
3 # number generator, set angka seed menjadi suatu angka, bisa semua angka,
  supaya hasil random nya selalu sama ketika kita run
4 np.random.seed(100)
5 # create dataframe 3 baris dan 5 kolom dengan angka random
6 df_tr = pd.DataFrame(np.random.rand(3,5))
7 # Cetak dataframe
8 print("Dataframe:\n", df_tr)
```

- **Cara 1** dengan tanpa define function awalnya, langsung pake fungsi anonymous lambda x

```
9  # Cara 1 dengan tanpa define function awalnya, langsung pake fungsi
   anonymous lambda x
10 df_tr1 = df_tr.applymap(lambda x: x*100)
11 print("\nDataframe - cara 1:\n", df_tr1)
```

Transforming (4)

- **Cara 2** dengan define function

```
12 # Cara 2 dengan define function
13 def times_100(x):
14     return x*100
15 df_tr2 = df_tr.applymap(times_100)
16 print("\nDataframe - cara 2:\n", df_tr2)
```

- Cara 1 dan cara 2 menunjukkan bahwa keduanya menghasilkan dataframe yang sama.

```
Dataframe:
          0         1         2         3         4
0  0.543405  0.278369  0.424518  0.844776  0.004719
1  0.121569  0.670749  0.825853  0.136707  0.575093
2  0.891322  0.209202  0.185328  0.108377  0.219697

Dataframe - cara 1:
           0          1          2          3          4
0  54.340494  27.836939  42.451759  84.477613   0.471886
1  12.156912  67.074908  82.585276  13.670659  57.509333
2  89.132195  20.920212  18.532822  10.837689  21.969749

Dataframe - cara 2:
           0          1          2          3          4
0  54.340494  27.836939  42.451759  84.477613   0.471886
1  12.156912  67.074908  82.585276  13.670659  57.509333
2  89.132195  20.920212  18.532822  10.837689  21.969749
```

Tugas 18. Transforming (4)

- Dengan cara yang sama seperti teori, buatlah matriks random ukuran 3 x 4 dengan seed randomnya 1234. Kemudian gunakan kedua cara seperti di atas untuk merubah seluruh isi dengan fungsi kuadrat x**2 + 3*x + 2.
- Pastikan telah menerapkan method applymap dengan menggunakan anonymous function dan function dengan def untuk persamaan kuadrat yang diminta.
- Jangan lupa menggunakan random seed generator 1234 dan kemudian mencetak setiap langkah dari perubahan dataframe seperti yang telah diinstruksikan.

```
import ___ np
import ___
# number generator, set angka seed menjadi suatu angka,
bisa semua angka, supaya hasil random nya selalu sama
ketika kita run
np.random.seed(___)
# create dataframe 3 baris dan 4 kolom dengan angka random
df_tr = ___
# Cetak dataframe
print("Dataframe:\n", ___)
# Cara 1 dengan tanpa define function awalnya, langsung
pake fungsi anonymous lambda x
df_tr1 = ___.___(___ x: ___)
print("\nDataframe - cara 1:\n", ___)
# Cara 2 dengan define function
def qudratic_fun(x):
        return ___
df_tr2 = ___.___(___)
print("\nDataframe - cara 2:\n", ___)
```

Tugas 18. Transforming (4)

```
Dataframe:
          0         1         2         3
0  0.191519  0.622109  0.437728  0.785359
1  0.779976  0.272593  0.276464  0.801872
2  0.958139  0.875933  0.357817  0.500995

Dataframe - cara 1:
          0         1         2         3
0  2.611238  4.253346  3.504789  4.972864
1  4.948290  2.892085  2.905825  5.048616
2  5.792449  5.395056  3.201485  3.753981

Dataframe - cara 2:
          0         1         2         3
0  2.611238  4.253346  3.504789  4.972864
1  4.948290  2.892085  2.905825  5.048616
2  5.792449  5.395056  3.201485  3.753981
```

Missing Value

Handling missing values menggunakan Pandas

https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html

Inspeksi Missing Value

- Value yang hilang/tidak lengkap dari dataframe akan membuat analisis atau model prediksi yang dibuat menjadi tidak akurat dan mengakibatkan keputusan salah yang diambil.
- Terdapat beberapa cara untuk mengatasi data yang hilang/tidak lengkap tersebut.
- Di pandas data yang hilang umumnya direpresentasikan dengan NaN.
- Digunakan data COVID-19 yang diambil dari google big query, tetapi sudah disediakan datasetnya dalam format csv dengan nama "public data covid19 jhu csse eu.csv".
- Data set ini dapat digunakan sebagai contoh studi kasus untuk meng-handle missing value.

Inspeksi Missing Value

Langkah-langkah:

1. Mengetahui kolom mana yang terdapat data hilang dan berapa jumlahnya dengan
   - **Cara 1**: menerapkan method **.info()** pada dataframe yang dapat diikuti dari kode berikut ini
   - **Cara 2**: mengetahui berapa banyak nilai hilang dari tiap kolom di dataset tersebut dengan menerapkan chaining method pada dataframe yaitu **.isna().sum()**.

   Method .isna() digunakan untuk mengecek berapa data yang bernilai NaN dan

   Method .sum() menjumlahkannya secara default untuk masing-masing kolom dataframe.

Inspeksi Missing Value

```
1. import pandas as pd
2. # Baca file "public data covid19 jhu csse eu.csv"
3. df = pd.read_csv(“/path/CHAPTER+4+-+missing+value+-+public+data+covid19+.csv")
4. # Cetak info dari df
5. print(df.info())
6. # Cetak jumlah missing value di setiap kolom
7. mv= df.isna().sum()
8. print("\nJumlah missing value per kolom:\n", mv)
```

Output baris kode kelima:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 13 columns):
province_state    960 non-null object
country_region    1000 non-null object
date              1000 non-null object
latitude          874 non-null float64
longitude         874 non-null float64
location_geom     874 non-null object
confirmed         1000 non-null int64
deaths            999 non-null float64
recovered         999 non-null float64
active            949 non-null float64
fips              949 non-null float64
admin2            842 non-null object
combined_key      0 non-null float64
dtypes: float64(7), int64(1), object(5)
memory usage: 101.6+ KB
```

Output untuk baris kode ketujuh dan kedelapan:

```
Jumlah missing value per kolom:
 province_state      40
country_region       0
date                 0
latitude           126
longitude          126
location_geom      126
confirmed            0
deaths               1
recovered            1
active              51
fips                51
admin2             158
combined_key      1000
dtype: int64
```

artinya ada beberapa kolom yang ada null sebagian dan ada yang nilainya null semua untuk kolomnya.

Tugas 19. Inspeksi Missing Value (1)

Lakukan inspeksi missing value pada data covid19.

```
import ___
# Baca file "public data covid19 jhu csse eu.csv"
___
# Cetak info dari df
___
# Cetak jumlah missing value di setiap kolom
___
```

Treatment untuk Missing Value (1)

- Terdapat beberapa cara untuk mengatasi missing value, antara lain:
  - **dibiarkan saja,**
  - **hapus value itu, atau**
  - **isi value tersebut dengan value yang lain (biasanya interpolasi, mean, median, etc)**
- Sebelum melakukan action ke missing value pada data covid diatas, sebaiknya tampilkan beberapa row teratas dari dataset itu agar dapat ditelaah terlebih dahulu.

	province_state	country_region	date	latitude	longitude	location_geom	confirmed	deaths	recovered	active	fips	admin2	combined_key
0	NaN	UK	01-02-20	NaN	NaN	NaN	2	0.0	0.0	NaN	NaN	NaN	NaN
1	NaN	UK	18-02-20	NaN	NaN	NaN	9	0.0	8.0	NaN	NaN	NaN	NaN
2	NaN	UK	17-02-20	NaN	NaN	NaN	9	0.0	8.0	NaN	NaN	NaN	NaN
3	NaN	UK	31-01-20	NaN	NaN	NaN	2	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	UK	19-02-20	NaN	NaN	NaN	9	0.0	8.0	NaN	NaN	NaN	NaN
5	NaN	UK	22-02-20	NaN	NaN	NaN	9	0.0	8.0	NaN	NaN	NaN	NaN
6	NaN	UK	25-02-20	NaN	NaN	NaN	13	0.0	8.0	NaN	NaN	NaN	NaN
7	NaN	UK	16-02-20	NaN	NaN	NaN	9	0.0	8.0	NaN	NaN	NaN	NaN
8	NaN	UK	27-02-20	NaN	NaN	NaN	15	0.0	8.0	NaN	NaN	NaN	NaN
9	NaN	UK	03-02-20	NaN	NaN	NaN	2	0.0	0.0	NaN	NaN	NaN	NaN

```
Jumlah missing value per kolom:
 province_state      40
country_region        0
date                  0
latitude            126
longitude           126
location_geom       126
confirmed             0
deaths                1
recovered             1
active               51
fips                 51
admin2              158
combined_key       1000
dtype: int64
```

Treatment untuk Missing Value (1)

```
Jumlah missing value per kolom:
 province_state      40
country_region        0
date                  0
latitude            126
longitude           126
location_geom       126
confirmed             0
deaths                1
recovered             1
active               51
fips                 51
admin2              158
combined_key       1000
dtype: int64
```

- Hanya kolom **combine_key** yang keseluruhan barisnya adalah missing value (1000 buah)
- Kolom **country_region, date**, dan **confirmed** tidak memiliki missing value.
- Untuk kolom lainnya terdapat beragam jumlah missing value. Apa yang dapat dilakukan?
- Untuk memahami mana kolom yang akan ditreatment dengan tiga perlakukan di atas lihat nature dari data terlebih dahulu. Contohnya pada kolom death dan recovered jika ada yang missing value maka kemungkinan terbesarnya adalah tidak ada meninggal atau sembuh pada hari tersebut.
- Untuk kolom yang seluruhnya missing yaitu combined_key dapat dibuang saja satu kolom itu karena tidak ada data yang dapat diketahui dari kolom tersebut.
- Sementara, kolom yang lainnya bagaimana? Misal ambil kolom province_state, missing valuenya dapat terjadi bahwa tidak dilaporkan itu berasal dari daerah mana di negara itu. Dapat mengisi misal dengan string 'unknown' karena tahu kolom tersebut bertipe data string.

Treatment untuk Missing Value (2)

Selanjutnya dapat menerapkan dua aksi yaitu:

1. Membiarkannya saja
2. Mengahapus kolom

- Treatment pertama (membiarkannya saja) seperti pada kolom confirmed, death, dan recovered. Akan tetapi jika tidak ada yang terkonfirmasi, meninggal dan sembuh sebenarnya dapat menukar value ini dengan angka nol. Meskipun ini lebih make sense dalam representasi datanya, tetapi untuk sub bab ini ketiga kolom tersebut diasumsikan dibiarkan memiliki nilai missing value.

- Treatment kedua yaitu dengan menghapus kolom, yang mana ini digunakan jika seluruh kolom dari dataset yang dipunyai semua barisnya adalah missing value. Untuk itu dapat menerapkan method .dropna() pada dataframe, bagaimana caranya?

```
nama_dataframe.dropna(axis=1, how="all")
```

Treatment untuk Missing Value (2)

- Pada method .dropna() ada dua keyword argumen yang harus diisikan yaitu axis dan how.
- Keyword axis digunakan untuk menentukan arah dataframe yang akan dibuang angka :
  - 1 untuk menyatakan kolom (column-based) atau dapat ditulis dalam string "column".
  - 0 berarti itu dalam searah index (row-based) atau dapat ditulis dalam string "index".
- Sementara, keyword how digunakan untuk bagaimana cara membuangnya. Opsi yang dapat diterimanya (dalam string) adalah
  - "all" artinya jika seluruh data di satu/beberapa kolom atau di satu/beberapa baris adalah missing value.
  - "any" artinya jika memiliki 1 saja data yang hilang maka buanglah baris/kolom tersebut.

Tugas 19. Inspeksi Missing Value (2)

- Cetaklah ukuran awal dari dataframe dengan attribut .shape pada dataframe yang telah dibaca.
- Buanglah kolom jika memiliki seluruh data adalah missing value, kemudian cetaklah ukurannya.
- Dari dataframe hasil langkah kedua buanglah baris-baris yang setidaknya memilki satu saja missing value, dan cetak kembali ukurannya.
- Pastikan axis untuk membuang kolom adalah 1 dan untuk membuang baris adalah 0, serta bagaimana cara membuangnya yaitu "all" seluruh data adalah missing value dan "any" jika setidaknya hanya satu saja yang missing value maka buang kolom/barisnya.

Tugas 19. Inspeksi Missing Value (2)

```
import ___

# Baca file "public data covid19 jhu csse eu.csv"
df = ___

# Cetak ukuran awal dataframe
print("Ukuran awal df: %d baris, %d kolom." % df.___)

# Drop kolom yang seluruhnya missing value dan cetak ukurannya
df = df.___(axis=___, how=___)
print("Ukuran df setelah buang kolom dengan seluruh data missing: %d baris, %d kolom." %
___)

# Drop baris jika ada satu saja data yang missing dan cetak ukurannya
df = ___(___, ___)
print("Ukuran df setelah dibuang baris yang memiliki sekurangnya 1 missing value: %d baris,
%d kolom." % ___)
```

```
Ukuran awal df: 1000 baris, 13 kolom.
Ukuran df setelah buang kolom dengan seluruh data missing: 1000 baris, 12 kolom.
Ukuran df setelah dibuang baris yang memiliki sekurangnya 1 missing value: 746 baris, 12 kolom.
```

Treatment untuk Missing Value (3)

- Sekarang, akan melakukan treatment ketiga untuk menghandle missing value pada dataframe. Treatment ini dilakukan dengan cara mengisi missing value dengan nilai lain, yang dapat berupa :
  - nilai statistik seperti mean atau median
  - interpolasi data
  - text tertentu
- Akan mulai pada kolom yang missing yang tipe datanya adalah berupa object. Kolom tersebut adalah province_state, karena tidak tahu secara persis province_state mana yang dimaksud, bisa menempatkan string "unknown" sebagai substitusi missing value.
- Meskipun keduanya berarti sama-sama tidak tahu tetapi berbeda dalam representasi datanya.
- Untuk melakukan hal demikian dapat menggunakan method **.fillna()** pada kolom dataframe yang dimaksud.

```
import pandas as pd

# Baca file "public data covid19 jhu csse eu.csv"
df = pd.read_csv(“/path/CHAPTER+4+-+missing+value+-+public+data+covid19+.csv")

# Cetak unique value pada kolom province_state
print("Unique value awal:\n", df["province_state"].unique())

# Ganti missing value dengan string "unknown_province_state"
df["province_state"] = df["province_state"].fillna("unknown_province_state")

# Cetak kembali unique value pada kolom province_state
print("Unique value setelah fillna:\n",  df["province_state"].unique())
```

Terlihat bahwa unique value di kolom "province_state" yang semula ada nan telah berubah menjadi "unknown".

```
Unique value awal:
 [nan 'US' 'Guam' 'Iowa']
Unique value setelah fillna:
 ['unknown' 'US' 'Guam' 'Iowa']
```

Tugas 20. Inspeksi Missing Value (3)

- Lakukan inspeksi missing value, isilah missing value dengan string "unknown_province_state".
- Ganti kolom province_state dengan kolom province_state yang sudah diterapkan method .fillna. Isi dari argument fillna adalah string/text yang diinstruksikan.
- Import pandas sebagai aliasnya, membaca dataset, mencetak unique value sebelum dan setelah diterapkan method fillna.

```
import ___
# Baca file "public data covid19 jhu csse eu.csv"
df = ___
# Cetak unique value pada kolom province_state
print("Unique value awal:\n", ___.___)
# Ganti missing value dengan string "unknown_province_state"
df[___] = ___.___(___)
# Cetak kembali unique value pada kolom province_state
print("Unique value setelah fillna:\n", ___)
```

```
Unique value awal:
 [nan 'US' 'Guam' 'Iowa']
Unique value setelah fillna:
 ['unknown_province_state' 'US' 'Guam' 'Iowa']
```

Treatment untuk Missing Value (4)

- Masih melanjutkan bagaimana menghandle missing value tentunya dengan jalan mengganti missing value dengan nilai lainnya.
- Pada bagian sebelumnya telah mengganti kolom bertipe objek dengan sesuatu string/teks.
- Dalam sub bab ini akan mengganti missing value dengan nilai statistik kolom bersangkutan, baik median atau mean (nilai rata-rata).
- Misalnya akan menggunakan kolom active.
- Dengan mengabaikan terlebih dahulu sebaran berdasarkan negara (univariate), jika mengisi dengan nilai rata-rata maka harus melihat terlebih dahulu data apakah memiliki ouliers atau tidak. Jika ada outliers dari data maka menggunakan nilai tengah (median) data adalah cara yang lebih safe.
- Untuk itu diputuskan dengan mengecek nilai median dan nilai mean kolom active juga nilai min dan max-nya. Jika data pada kolom active terdistribusi normal maka nilai mean dan median akan hampir sama.

Treatment untuk Missing Value (4)

```
Min    : -6.0
Mean   : 192.57112750263434
Median: 41.0
Max    : 2243.0
```

- Terlihat data memiliki distribusi yang skewness, karena nilai mean dan median yang cukup jauh serta range data yang cukup lebar.
- Di sini pada kolom active data memiliki outliers.
- Jadi akan mengisi missing value dengan median.

Treatment untuk Missing Value (4)

```
import pandas as pd
# Baca file "https:/path/CHAPTER+4+-+missing+value+-+public+data+covid19+.csv"
df = pd.read_csv("https:/path/CHAPTER+4+-+missing+value+-+public+data+covid19+.csv")
```

```
# Cek missing value di kolom active
print("Jumlah missing value:", df["active"].isna().sum())
# Isi missing value kolom active dengan median
df["active"] = df["active"].fillna(df["active"].median())
# Cek kembali kolom active
print("Jumlah missing value setelah diisi median:",
df["active"].isna().sum())
```

```
Jumlah missing value: 51
Jumlah missing value setelah diisi median: 0
```

Tugas 20. Inspeksi Missing Value (4)

- Carilah perbedaan nilai mean dan median kolom active untuk kondisi sebelum dan setelah missing valuenya diisi masing-masingnya dengan median dan mean.
- Gunakan method fillna dengan argumennya adalah nilai median dan mean kolom yang akan diisi nilai missing valuenya.
- Pastikan mengisi nilai missing value pada kolom active dengan nilai median terlebih dahulu baru kemudian diikuti denga mengisi nya dengan mean.
- Cetaklah nilai mean dan median kolom tersebut sebelum dan setelah diisi missing valuenya.

```
Awal: mean = 192.571128, median = 41.000000.
Fillna median: mean = 184.841000, median = 41.000000.
Fillna mean: mean = 192.571128, median = 49.000000.
```

```
import ___

# Baca file "/path/CHAPTER%204%20-%20missing%20value%20-
%20public%20data%20covid19%20.csv"

df = ___
# Cetak nilai mean dan median awal
print("Awal: mean = %f, median = %f." % (df["active"].___, ___))

# Isi missing value kolom active dengan median
df_median = df["active"].___(___)

# Cetak nilai mean dan median awal setelah diisi dengan median
print("Fillna median: mean = %f, median = %f." % (___, ___))

# Isi missing value kolom active dengan mean
df_mean = df["active"].___(___)

# Cetak nilai mean dan median awal setelah diisi dengan mean
print("Fillna mean: mean = %f, median = %f." % (___, ___))
```

Treatment untuk Missing Value (5)

- Menggunakan teknik interpolasi dalam mengisi nilai missing value pada suatu dataset.
- Data yang menggunakan interpolasi untuk mengisi data yang hilang adalah time series data, yang secara default akan diisi dengan interpolasi linear.

```python
import numpy as np
import pandas as pd
# Data
ts = pd.Series({
    "2020-01-01":15,
    "2020-01-02":np.nan,
    "2020-01-05":np.nan,
    "2020-01-07":20,
    "2020-01-10":np.nan,
    "2020-01-12":22,
    "2020-01-15":25,
    "2020-01-17":np.nan,
    "2020-01-16":40,
    "2020-01-20":45,
    "2020-01-22":np.nan,
    "2020-01-25":60,
    "2020-01-28":np.nan,
    "2020-01-30":80
})
# Cetak time series
print("Awal:\n", ts)
# Isi missing value menggunakan interpolasi linier
ts = ts.interpolate()
# Cetak time series setelah interpolasi linier
print("Setelah diisi missing valuenya:\n", ts)
```

```
Awal:
 2020-01-01    15.0
2020-01-02     NaN
2020-01-05     NaN
2020-01-07    20.0
2020-01-10     NaN
2020-01-12    22.0
2020-01-15    25.0
2020-01-16    40.0
2020-01-17     NaN
2020-01-20    45.0
2020-01-22     NaN
2020-01-25    60.0
2020-01-28     NaN
2020-01-30    80.0
dtype: float64
Setelah diisi missing valuenya:
 2020-01-01    15.000000
2020-01-02    16.666667
2020-01-05    18.333333
2020-01-07    20.000000
2020-01-10    21.000000
2020-01-12    22.000000
2020-01-15    25.000000
2020-01-16    40.000000
2020-01-17    42.500000
2020-01-20    45.000000
2020-01-22    52.500000
2020-01-25    60.000000
2020-01-28    70.000000
2020-01-30    80.000000
dtype: float64
```

Tugas 21. Treatment untuk Missing Value (5)

```
import ___
import ___ pd
# Data
ts = pd.___({
    "2020-01-01":9,
    "2020-01-02":np.nan,
    "2020-01-05":np.nan,
    "2020-01-07":24,
    "2020-01-10":np.nan,
    "2020-01-12":np.nan,
    "2020-01-15":33,
    "2020-01-17":np.nan,
    "2020-01-16":40,
    "2020-01-20":45,
    "2020-01-22":52,
    "2020-01-25":75,
    "2020-01-28":np.nan,
    "2020-01-30":np.nan
})
# Isi missing value menggunakan interpolasi linier
ts = ___
# Cetak time series setelah interpolasi linier
print("Setelah diisi missing valuenya:\n", ___)
```

- Pastikan telah mengcreate pandas series serta menerapkan interpolasi melalui penggunaan .interpolate() pada series yang telah kamu buat.
- Perlu mengimport library numpy dan pandas sebagai alias masing-masingnya.

```
Setelah diisi missing valuenya:
 2020-01-01     9.0
2020-01-02    14.0
2020-01-05    19.0
2020-01-07    24.0
2020-01-10    27.0
2020-01-12    30.0
2020-01-15    33.0
2020-01-16    40.0
2020-01-17    42.5
2020-01-20    45.0
2020-01-22    52.0
2020-01-25    75.0
2020-01-28    75.0
2020-01-30    75.0
dtype: float64
```

Referensi

- Data Manipulation with Pandas - DQLab.
- https://towardsdatascience.com/data-manipulation-for-machine-learning-with-pandas-ab23e79ba5de