

# Modul 6

## Pengenalan NumPy, Pandas, Matplotlib

Pada modul kali ini, akan dibahas 3 library yang sering digunakan pada saat mengimplementasikan Machine Learning pada Python, yaitu **NumPy**, **Pandas**, dan **Matplotlib**. Langsung kita mulai dengan **NumPy**.

### NumPy

**NumPy** (*Numerical Python*) adalah library Python yang fokus pada *scientific computing*. NumPy memiliki kemampuan untuk membentuk objek N-dimensional *array*, yang mirip dengan *list* pada Python. Keunggulan NumPy array dibandingkan dengan *list* pada Python adalah konsumsi *memory* yang lebih kecil serta *runtime* yang lebih cepat. NumPy juga memudahkan kita pada Aljabar Linear, terutama operasi pada Vector (1-d *array*) dan Matrix (2-d *array*).

*List* pada Python tidak mendukung penuh kemudahan *scientific computing*, sebagai contoh lakukan operasi penjumlahan pada 2 *list*.

```
In [1]: 1 # list pada python
        2 a = [1, 2, 3]
        3 b = [4, 5, 6]
        4 a + b
```

```
Out[1]: [1, 2, 3, 4, 5, 6]
```

```
In [2]: 1 # penjumlahan dilakukan iterasi
        2 result = []
        3 for first, second in zip(a, b):
        4     result.append(first + second)
        5 result
```

```
Out[2]: [5, 7, 9]
```

Ketika ingin menjumlahkan tiap elemen pada *list a* dan *list b*, hasilnya dengan operator + adalah penggabungan (*concat*) keduanya. Tentu tidak sesuai yang diharapkan, maka harus menggunakan perulangan *for* untuk menambahkan tiap elemen pada *list a* dan *list b*. Proses penjumlahan *list* yang menggunakan perulangan *for* membutuhkan waktu yang lama dan tidak efisien dari sisi penulisan *code*.

## Pengenalan NumPy Arrays

### Membuat Array

Lakukan *import* terlebih dahulu library **numpy as np**. Penggunaan **as** disini, artinya menggantikan pemanggilan **numpy** dengan *prefix np* untuk proses berikutnya.

```
In [3]: 1 import numpy as np
```

```
In [4]: 1 # membuat array
        2 a = np.array([1, 2, 3])
        3 a
```

```
Out[4]: array([1, 2, 3])
```

Untuk membuat sebuah *array*, menggunakan fungsi **array()** yang terdapat pada **NumPy**. Pada NumPy, terdapat **upcasting**, yaitu ketika tipe data element array tidak sama, dilakukan penyamaan tipe data pada yang lebih tinggi. Misalkan kita membuat array *numeric* dengan semua element bertipe *integer*, kecuali 1 element bertipe *float*, maka otomatis akan dilakukan *upcasting* menjadi tipe *float* pada semua element array.

```
In [23]: 1 b = np.array([1, 2, 3.0])
        2 b
```

```
Out[23]: array([1., 2., 3.])
```

### Cek Tipe

Untuk melakukan pengecekan tipe pada array menggunakan fungsi **type()**.

```
In [6]: 1 # mengecek tipe data array
        2 type(a)
```

```
Out[6]: numpy.ndarray
```

NumPy array merupakan sebuah objek **ndarray**, yang merupakan singkatan dari **n-dimensional array**.

### Tipe Data untuk Element

Pengecekan tipe data element pada array menggunakan fungsi **dtype**.

```
In [25]: 1 a = np.array([1, 2, 3])
          2 a.dtype
```

```
Out[25]: dtype('int32')
```

Dalam membuat sebuah array, kita dapat menetapkan tipe data dengan menambahkan parameter **dtype**.

```
In [26]: 1 a = np.array([1, 2, 3], dtype='int64')
          2 a.dtype
```

```
Out[26]: dtype('int64')
```

Array *a* memiliki tipe data **int32** dan **int64** yang keduanya sama-sama bertipekan *integer*. Perbedaan keduanya pada kapasitas penyimpanan data. Pada **int32** mampu menampung hingga `(-2,147,483,648 to +2,147,483,647)` sedangkan **int64** mampu menampung hingga `(-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807)`. Penting bagi kita memperhatikan tipe data beserta kapasitas penyimpanannya agar dapat mengalokasikan *memory* penyimpanan dengan baik. Beberapa tipe data standar yang terdapat pada NumPy adalah sebagai berikut:

Data type	Description
<code>bool_</code>	Boolean (True or False) stored as a byte
<code>int_</code>	Default integer type (same as C <code>long</code> ; normally either <code>int64</code> or <code>int32</code> )
<code>intc</code>	Identical to C <code>int</code> (normally <code>int32</code> or <code>int64</code> )
<code>intp</code>	Integer used for indexing (same as C <code>ssize_t</code> ; normally either <code>int32</code> or <code>int64</code> )
<code>int8</code>	Byte (−128 to 127)
<code>int16</code>	Integer (−32768 to 32767)
<code>int32</code>	Integer (−2147483648 to 2147483647)
<code>int64</code>	Integer (−9223372036854775808 to 9223372036854775807)
<code>uint8</code>	Unsigned integer (0 to 255)
<code>uint16</code>	Unsigned integer (0 to 65535)
<code>uint32</code>	Unsigned integer (0 to 4294967295)
<code>uint64</code>	Unsigned integer (0 to 18446744073709551615)
<code>float_</code>	Shorthand for <code>float64</code>
<code>float16</code>	Half-precision float: sign bit, 5 bits exponent, 10 bits mantissa
<code>float32</code>	Single-precision float: sign bit, 8 bits exponent, 23 bits mantissa
<code>float64</code>	Double-precision float: sign bit, 11 bits exponent, 52 bits mantissa
<code>complex_</code>	Shorthand for <code>complex128</code>
<code>complex64</code>	Complex number, represented by two 32-bit floats
<code>complex128</code>	Complex number, represented by two 64-bit floats

## Jumlah Dimensi

NumPy array memiliki keunggulan mendukung operasi pada data dimensional seperti Vektor dan Matriks. Untuk mengetahui jumlah dimensi pada data menggunakan fungsi **ndim**.

```
In [34]: 1 # cek jumlah dimensi pada array
          2 a = np.array([1, 2, 3])
          3 a.ndim
```

Out[34]: 1

Array *a* memiliki jumlah dimensi 1. Jika kita membentuk array dengan 2-dimensi, maka jumlah dimensinya adalah 2, begitu juga dengan dimensi yang lebih besar.

## Array Shape

Pada fungsi **shape** menghasilkan sebuah *tuple* yang berisikan panjang sebuah array pada tiap dimensi.

```
In [38]: 1 # cek shape dari array
          2 a = np.array([1, 2, 3])
          3 a.shape
```

Out[38]: (3,)

Array *a* memiliki **shape** 3, yaitu panjang array pada 1-dimensi array.

## Operasi pada Array

NumPy memudahkan kita untuk operasi *elementwise* pada Vektor dan Matriks seperti penjumlahan, perkalian, pangkat, dan operasi lainnya.

```
In [39]: 1 a = np.array([1, 2, 3])
          2 f = np.array([1.1, 2.2, 3.3])
          3 a + f
```

Out[39]: array([2.1, 4.2, 6.3])

```
In [40]: 1 a * f
```

Out[40]: array([1.1, 4.4, 9.9])

```
In [41]: 1 a ** f
```

Out[41]: array([ 1. , 4.59479342, 37.5405076 ])

```
In [42]: 1 # universal function (ufuncs) pada numpy seperti sin
          2 np.sin(a)
```

Out[42]: array([0.84147098, 0.90929743, 0.14112001])

Pada NumPy telah disediakan *universal functions* (**ufunc**) yaitu fungsi yang dapat melakukan operasi pada NumPy array dengan eksekusi elemen-demi-elemen. **ufunc** merupakan sebuah '*vectorized*' *wrapper* untuk sebuah fungsi yang memiliki input dan output yang spesifik. Contoh dari **ufunc** misalkan fungsi *sin* dan *cos*.

## Element pada Array

### Array Indexing

Indeks pada suatu array dimulai pada indeks ke-0. Untuk mengakses element pada array, kita menggunakan indeks sebagai alamat elemen pada array.

```
In [32]: 1 # mengakses elemen pada indeks ke-0
          2 a = np.array([1, 2, 3])
          3 a[0]
```

Out[32]: 1

```
In [33]: 1 # melakukan assign pada element array dengan element baru
          2 a[0] = 10
          3 a
```

Out[33]: array([10, 2, 3])

Kita dapat melakukan *assign* nilai baru pada suatu element berdasarkan alamat indeks. Maka, setelah dilakukan *assign* nilai element pada indeks ke-0 berganti menjadi 10.

Perlu kita perhatikan jika **dtype** pada array adalah *integer*, misalkan **int32**. Kemudian kita *assign* nilai baru pada salah satu elemen dengan tipe data *float*. Maka, nilai baru tersebut akan dipaksa menjadi tipe **int32** sesuai dengan tipe data pada array.

```
In [29]: 1 a = np.array([1, 2, 3])
          2 a.dtype
```

Out[29]: dtype('int32')

```
In [30]: 1 # assign nilai baru pada indeks ke-0
          2 a[0] = 11.6
          3 a
```

Out[30]: array([11, 2, 3])

## Multi-Dimensional Arrays

NumPy array memudahkan kita untuk membuat array multi-dimensi.

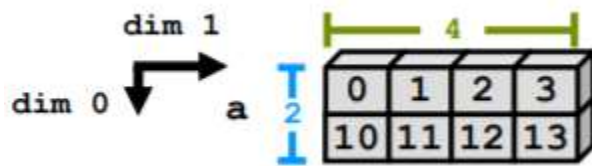
**2D Array**

a	0	1	2	3
	10	11	12	13

```
In [45]: 1 # membentuk array 2-dimensi
          2 a = np.array([[ 0, 1, 2, 3],
          3                  [10, 11, 12, 13]])
          4 a
```

Out[45]: array([[ 0, 1, 2, 3],  
 [10, 11, 12, 13]])

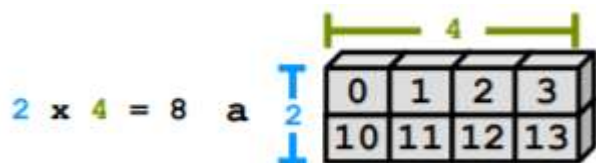
Fungsi **shape** pada array multi-dimensi menghasilkan *tuple* berisikan (**jumlah baris, jumlah kolom**).



```
In [46]: 1 a.shape
```

```
Out[46]: (2, 4)
```

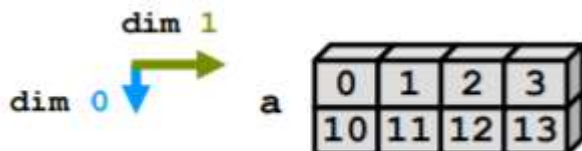
Untuk mengetahui jumlah elemen pada array multi-dimensi menggunakan fungsi **size**.



```
In [47]: 1 a.size
```

```
Out[47]: 8
```

Untuk mengetahui jumlah dimensi dari array multi-dimensi menggunakan fungsi **ndim**. Jumlah dimensi pada array **a** adalah 2, karena merupakan array 2-dimensi.



```
In [48]: 1 a.ndim
```

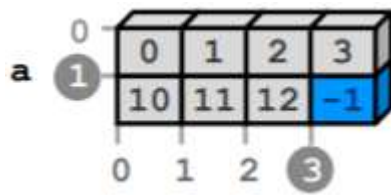
```
Out[48]: 2
```

Pada array multi-dimensi kita dapat mengakses nilai elemen berdasarkan indeks dengan pemisah tanda koma [ , ]. Misalkan kita ingin mendapatkan data pada baris ke-1 dan kolom ke-3 dari array **a**, maka dilakukan perintah **a[1, 3]**.

```
In [49]: 1 a[1, 3]
```

```
Out[49]: 13
```

Misalkan, kita akan mengganti data pada baris ke-1 dan kolom ke-3 pada array **a** dengan nilai -1, kita dapat langsung assign nilai baru tersebut pada indeks.



```
In [50]: 1 a[1, 3] = -1
          2 a
```

```
Out[50]: array([[ 0,  1,  2,  3],
                [10, 11, 12, -1]])
```

## Slicing

Selain kita dapat mengambil sebuah element array dengan cara indexing seperti cara di atas, kita dapat melakukan slicing, yaitu melakukan ekstraksi elemen atau beberapa elemen pada array, dengan menggunakan tanda `[:]`.

### `var[lower:upper:step]`

Perlu kita perhatikan, elemen pada **lower** akan dimasukkan hasil *slicing*, sedangkan element pada **upper** TIDAK dimasukkan hasil slicing, dan step adalah jarak antar elemen.

```
In [53]: 1 a = np.array([1, 2, 3])
          2 # slicing dari indeks ke-1, dan batas indeks ke-2
          3 a[1:2]
```

```
Out[53]: array([2])
```

```
In [56]: 1 # slicing dari indeks ke-1, dan batas indeks ke-1 dari belakang
          2 a[1:-1]
```

```
Out[56]: array([2])
```

```
In [62]: 1 a[::2]
```

```
Out[62]: array([1, 3])
```

Indeks dapat didefinisikan dengan nilai negatif. Misalkan indeksnya adalah `[-1]`, maka indeks terdapat pada elemen ke-1 yang dari lokasi akhir elemen suatu array.

NumPy masih menyediakan banyak sekali fungsi yang sangat memudahkan kita untuk scientific computing.

## Pandas

**Pandas (Python for Data Analysis)** adalah library Python yang fokus untuk proses analisis data seperti manipulasi data, persiapan data, dan pembersihan data. Pandas



menyediakan struktur data dan fungsi high-level untuk membuat pekerjaan dengan data terstruktur/tabular lebih cepat, mudah, dan ekspresif.

Dalam **pandas** terdapat dua objek yang akan dibahas pada tutorial ini, yaitu **DataFrame** dan **Series**. **DataFrame** adalah objek yang memiliki struktur data tabular, berorientasi pada kolom dengan label baris dan kolom. Sedangkan **Series** adalah objek array 1-dimensi yang memiliki label.

**Pandas** memadukan library **NumPy** yang memiliki kemampuan manipulasi data yang fleksibel dengan database relasional (seperti SQL). Sehingga memudahkan kita untuk melakukan *reshape*, *slice* dan *dice*, *agregasi data*, dan mengakses subset dari data.

Untuk memulai, lakukan import terlebih dahulu library **pandas as pd**. Penggunaan **as** disini, artinya kita menggantikan pemanggilan **pandas** dengan **prefix pd** untuk proses berikutnya.

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

## Series

**Series** adalah objek 1-dimensi yang berisi sequence nilai dan berasosiasi dengan label data, yang disebut indeks. Untuk membuat sebuah Series, kita dapat membentuknya dari sebuah array dengan memanggil fungsi **Series** pada **pandas**.

```
In [2]: 1 # membuat series
        2 obj = pd.Series([1, 2, 3])
        3 obj
```

```
Out[2]: 0    1
        1    2
        2    3
        dtype: int64
```

```
In [3]: 1 # cek tipe
        2 type(obj)
```

```
Out[3]: pandas.core.series.Series
```

Dapat dilihat, pada Series **obj** ditampilkan dua buah kolom, bagian kiri adalah *index* dan bagian kanan adalah *values*. Pada *index*, karena kita tidak mendefinisikan index pada data, maka secara default dimulai dari integer 0 hingga  $N-1$  (dimana  $N$  adalah panjang data). Kita dapat mendefinisikan index dengan menambahkan parameter index saat membuat objek Series.

```
In [5]: 1 # membuat series dengan index
        2 obj2 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
        3 obj2
```

```
Out[5]: a    1
        b    2
        c    3
        dtype: int64
```

```
In [6]: 1 # mendapatkan index obj2
        2 obj2.index
```

```
Out[6]: Index(['a', 'b', 'c'], dtype='object')
```

```
In [7]: 1 # mendapatkan values obj2
        2 obj2.values
```

```
Out[7]: array([1, 2, 3], dtype=int64)
```

Kita dapat mengambil **index** dari objek **Series** dengan menggunakan fungsi **index** dan mengambil **values** dari objek **Series** dengan menggunakan fungsi **values**.

Untuk mengakses suatu **value** pada objek **Series**, kita dapat menggunakan **index** sebagai alamat **value** tersebut. Hal ini memungkinkan untuk melakukan **assign** nilai baru pada objek **Series**.

```
In [8]: 1 # menampilkan value pada indeks a
        2 obj2['a']
```

```
Out[8]: 1
```

```
In [9]: 1 # assign nilai baru pada indeks a
        2 obj2['a'] = 4
        3 obj2
```

```
Out[9]: a    4
        b    2
        c    3
        dtype: int64
```

```
In [11]: 1 # akses nilai pada indeks a dan c
         2 obj2[['a', 'c']]
```

```
Out[11]: a    4
        c    3
        dtype: int64
```

Series juga memungkinkan untuk melakukan operasi aritmatika pada dua objek **Series** dengan indeks yang sama.

```
In [12]: 1 # membuat objek series baru
          2 obj3 = pd.Series([4, 5, 6], index=['a', 'd', 'c'])
          3 obj3
```

```
Out[12]: a    4
          d    5
          c    6
          dtype: int64
```

```
In [13]: 1 obj2 + obj3
```

```
Out[13]: a    8.0
          b    NaN
          c    9.0
          d    NaN
          dtype: float64
```

Hasil operasi arimatika tambah pada 2 objek Series mirip dengan operasi **join** pada pengolahan **database**. Indeks yang tidak memiliki kesamaan pada 2 objek **Series** akan memiliki value **NaN**.

## DataFrame

**DataFrame** merupakan tabel data yang terdapat kolom dan baris, dimana nilai-nilai yang terdapat di dalamnya dapat berupa tipe berbeda seperti *numeric*, *string*, *boolean*, dll. **DataFrame** mirip dengan *data 2-dimensi* dengan adanya baris dan kolom. Selain itu, *DataFrame* bisa dikatakan gabungan dari *dictionary* objek Series yang memiliki indeks yang sama.

Terdapat berbagai macam cara untuk membentuk objek *DataFrame*. Salah satu cara yang biasa dilakukan untuk membentuk objek **DataFrame** dengan menggunakan data masukan berupa *dictionary*.

```
In [15]: 1 # membuat dataframe
2 data = {'kota': ['semarang', 'semarang', 'semarang',
3               'bandung', 'bandung', 'bandung'],
4         'tahun': [2016, 2017, 2018, 2016, 2017, 2018],
5         'populasi': [1.5, 2.1, 3.2, 2.3, 3.2, 4.5]}
6 frame = pd.DataFrame(data)
7 frame
```

```
Out[15]:
```

	kota	tahun	populasi
0	semarang	2016	1.5
1	semarang	2017	2.1
2	semarang	2018	3.2
3	bandung	2016	2.3
4	bandung	2017	3.2
5	bandung	2018	4.5

```
In [16]: 1 # cek tipe
2         type(frame)
```

```
Out[16]: pandas.core.frame.DataFrame
```

Kita dapat menggunakan fungsi **shape** untuk mengetahui jumlah baris dan kolom dari DataFrame. Fungsi **shape** pada **DataFrame** memiliki hasil keluaran yang sama dengan fungsi shape pada NumPy, dimana menghasilkan keluaran sebuah **tuple** dari jumlah baris dan jumlah kolom.

```
In [17]: 1 # cek shape dari frame
2         frame.shape
```

```
Out[17]: (6, 3)
```

```
In [18]: 1 # cek info dari frame
2         frame.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 3 columns):
kota      6 non-null object
tahun     6 non-null int64
populasi  6 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 224.0+ bytes
```

Fungsi **info()** sangat berguna untuk mengetahui keterangan dari objek *DataFrame* yang kita buat seperti *index* dari *DataFrame* lengkap dengan *range* dari *index*, jumlah kolom beserta informasi tiap kolom untuk null data dan tipe data, dan jumlah total penggunaan memory pada tiap kolom dalam satuan bytes. Saat kita input data dictionary pada *DataFrame*, kita tidak perlu mendefinisikan tipe data untuk masing-

masing kolom, karena secara otomatis **pandas** akan memberikan tipe data sesuai dengan values untuk tiap kolom, meskipun kita juga bisa mendefinisikan tipe data secara manual.

Misalkan kita memiliki objek DataFrame yang memiliki baris hingga jutaan, kita tidak ingin menampilkan data secara keseluruhan karena akan menghabiskan memory. Kita dapat menggunakan fungsi **head()** dan **tail()** untuk menampilkan data secara default untuk 5 data teratas dan 5 data terbawah.

```
In [19]: 1 # menampilkan data 5 teratas
         2 frame.head()
```

Out[19]:

	kota	tahun	populasi
0	semarang	2016	1.5
1	semarang	2017	2.1
2	semarang	2018	3.2
3	bandung	2016	2.3
4	bandung	2017	3.2

```
In [20]: 1 # menampilkan data 5 terbawah
         2 frame.tail()
```

Out[20]:

	kota	tahun	populasi
1	semarang	2017	2.1
2	semarang	2018	3.2
3	bandung	2016	2.3
4	bandung	2017	3.2
5	bandung	2018	4.5

Seperti pada Series, kita juga dapat mengakses kolom pada DataFrame menggunakan fungsi **columns** dan untuk mengakses **indeks** dari DataFrame menggunakan fungsi **index**. Jika ingin mendapatkan data pada DataFrame secara keseluruhan menggunakan fungsi **values** yang akan menghasilkan output berupa array 2-dimensi sesuai dengan jumlah baris dan kolom.

```
In [22]: 1 # column pada dataframe
         2 frame.columns
```

```
Out[22]: Index(['kota', 'tahun', 'populasi'], dtype='object')
```

```
In [23]: 1 # index pada dataframe
         2 frame.index
```

```
Out[23]: RangeIndex(start=0, stop=6, step=1)
```

```
In [29]: 1 # mendapatkan keseluruhan data
         2 frame.values
```

```
Out[29]: array([[ 'semarang', 2016, 1.5],
                 [ 'semarang', 2017, 2.1],
                 [ 'semarang', 2018, 3.2],
                 [ 'bandung',  2016, 2.3],
                 [ 'bandung',  2017, 3.2],
                 [ 'bandung',  2018, 4.5]], dtype=object)
```

DataFrame menyediakan fungsi **describe()** untuk mengetahui statistika data untuk data numeric seperti count, mean, standard deviation, maximum, minimum, dan quartile. Untuk data string, misalkan data tersebut adalah kategori, kita dapat menggunakan fungsi **value\_counts()** untuk mengetahui jumlah tiap kategori pada data.

```
In [30]: 1 # statistika pada data numeric
         2 frame.describe()
```

```
Out[30]:
```

	tahun	populasi
count	6.000000	6.000000
mean	2017.000000	2.800000
std	0.894427	1.062073
min	2016.000000	1.500000
25%	2016.250000	2.150000
50%	2017.000000	2.750000
75%	2017.750000	3.200000
max	2018.000000	4.500000

```
In [31]: 1 # statistik untuk data string
         2 frame['kota'].value_counts()
```

```
Out[31]: semarang    3
         bandung     3
         Name: kota, dtype: int64
```

## Mengakses Data pada DataFrame

Terkadang kita butuh mengakses kolom tertentu atau lebih spesifik elemen tertentu pada DataFrame. Untuk mengakses suatu data, alamat data tersebut adalah pada nama

kolom sebagai petunjuk lokasi kolom dan indeks sebagai petunjuk lokasi baris. Misalkan untuk mengakses semua data pada kolom **populasi**, maka kita menggunakan perintah **frame['populasi']**. Perlu diingat karena nama kolom adalah tipe data string, maka kita menggunakan petik.

```
In [33]: 1 # akses data pada kolom a
          2 frame['populasi']

Out[33]: 0    1.5
          1    2.1
          2    3.2
          3    2.3
          4    3.2
          5    4.5
          Name: populasi, dtype: float64
```

Sedangkan mengakses data pada baris tertentu, kita menggunakan fungsi **loc[indeks]**. Kita juga bisa mendapatkan lebih dari 1 baris dengan menggunakan titik dua ':', misalkan kita ingin mengakses indeks 2–3, maka menggunakan perintah **loc[2:3]**.

```
In [38]: 1 # akses data pada baris ke-3, berarti indeks ke-2
          2 frame.loc[2]

Out[38]: kota      semarang
          tahun      2018
          populasi    3.2
          Name: 2, dtype: object
```

```
In [39]: 1 # akses data pada indeks 2-3
          2 frame.loc[2:3]

Out[39]:
```

	kota	tahun	populasi
2	semarang	2018	3.2
3	bandung	2016	2.3

Elemen pada DataFrame dapat diakses dengan mendefinisikan nama kolom dan indeks baris secara bersamaan. Misalkan kita ingin mendapatkan data populasi pada indeks ke-2, maka digunakan perintah **frame['populasi'][2]**.

```
In [40]: 1 # akses elemen pada kolom populasi indeks ke-2
          2 frame['populasi'][2]

Out[40]: 3.2
```

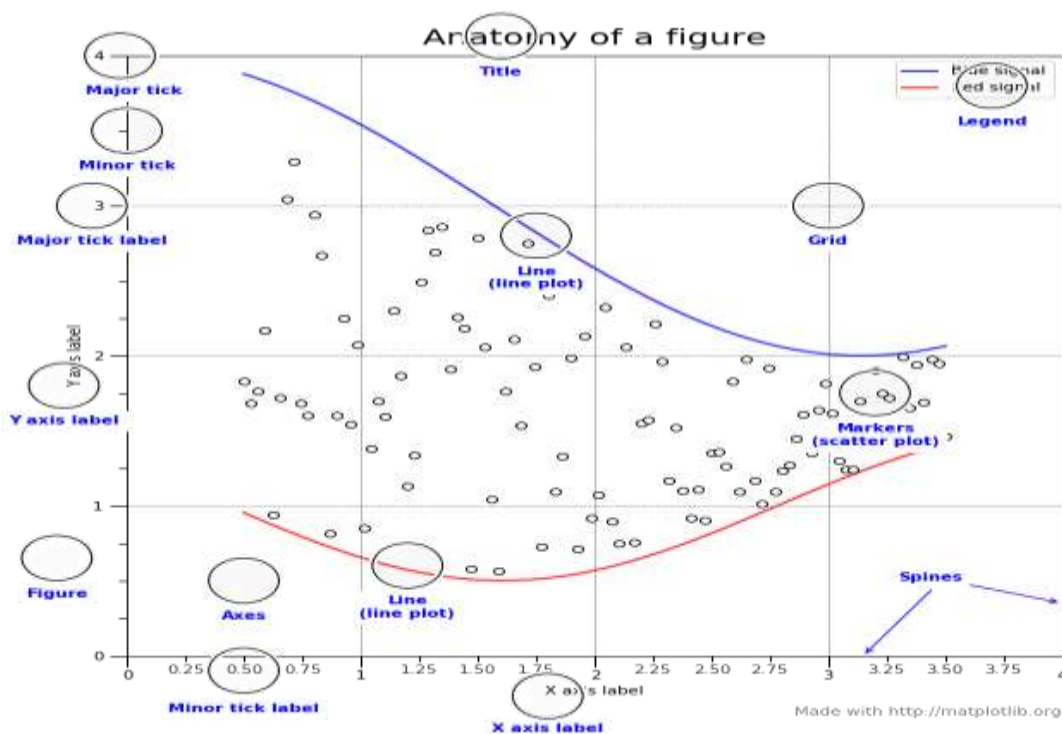
Pandas masih menyediakan banyak sekali fungsi yang sangat memudahkan kita untuk analisis data seperti mengisi data kosong dengan fungsi **fillna()**, menerapkan suatu fungsi pada DataFrame menggunakan **apply()**, mengubah format DataFrame dari wide ke long menggunakan **melt()**, dan masih banyak lagi.

## Matplotlib

**Matplotlib** adalah library Python yang fokus pada visualisasi data seperti membuat *plot* grafik. Matplotlib pertama kali diciptakan oleh John D. Hunter dan sekarang telah dikelola oleh tim developer yang besar. Awalnya matplotlib dirancang untuk menghasilkan *plot* grafik yang sesuai pada publikasi jurnal atau artikel ilmiah. Matplotlib dapat digunakan dalam skrip Python, Python dan IPython *shell*, server aplikasi web, dan beberapa toolkit *graphical user interface* (GUI) lainnya.

Visualisasi dari matplotlib adalah sebuah gambar grafik yang terdapat satu sumbu atau lebih. Setiap sumbu memiliki sumbu horizontal (x) dan sumbu vertikal (y), dan data yang direpresentasikan menjadi warna dan *glyphs* seperti *marker* (contohnya bentuk lingkaran) atau *lines* (garis) atau poligon.

Gambar di bawah menunjukkan bagian-bagian dari visualisasi matplotlib dibuat oleh Nicolas P. Rougier.



Hal yang penting dalam visualisasi data adalah penentuan warna, tekstur, dan *style* yang menarik untuk dilihat dan representatif terhadap data. Seorang Cartographer yaitu Jacques Bertin mengembangkan rekomendasi berikut untuk pemilihan informasi visual yang cocok, dan kita dapat menerapkannya menggunakan matplotlib.



	<i>Points</i>	<i>Lines</i>	<i>Areas</i>	<i>Best to show</i>
<i>Shape</i>		<i>possible, but too weird to show</i>	<i>cartogram</i>	<i>qualitative differences</i>
<i>Size</i>			<i>cartogram</i>	<i>quantitative differences</i>
<i>Color Hue</i>				<i>qualitative differences</i>
<i>Color Value</i>				<i>quantitative differences</i>
<i>Color Intensity</i>				<i>qualitative differences</i>
<i>Texture</i>				<i>qualitative &amp; quantitative differences</i>

Untuk memulai menggunakan matplotlib, lakukan *import* terlebih dahulu library **matplotlib.pyplot** as **plt**. Penggunaan *as* disini, artinya kita menggantikan pemanggilan fungsi **pyplot** pada **matplotlib** dengan *prefix* **plt** untuk proses berikutnya. Disini terdapat *magic command* **%matplotlib inline**, untuk pengaturan pada *backend* matplotlib agar setiap grafik ditampilkan secara '**inline**', yaitu akan ditampilkan langsung pada *cell notebook*.

```
In [1]: 1 # import library
        2 import matplotlib.pyplot as plt
        3 %matplotlib inline
```

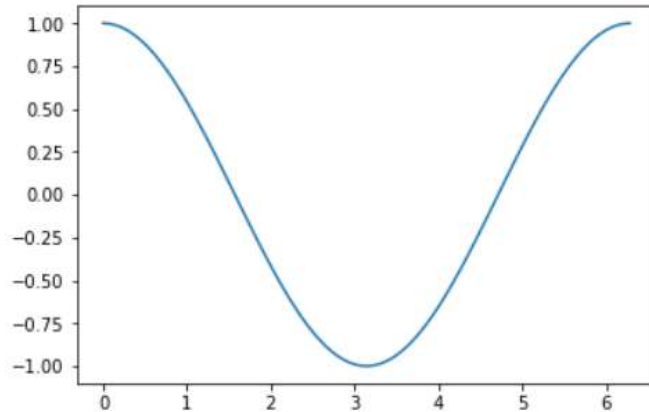
## Membuat Line Plot

**Line plot** berguna untuk melacak perubahan pada periode waktu pendek dan panjang. Ketika terdapat perubahan kecil, line plot lebih baik dalam melakukan visualisasi dibandingkan grafik *bar*.

Tutorial kali ini akan membuat plot grafik *line* menggunakan gelombang *cos*. Kita akan menggunakan **numpy** untuk *generate* data gelombang *cos* dengan jumlah data 100 yang berjarak dari 0 sampai  $2\pi$ .

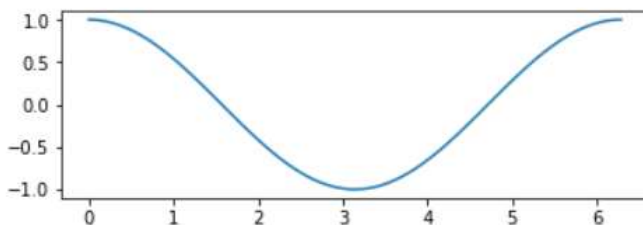
```
In [2]: 1 import numpy as np
2
3 x = np.linspace(0, 2*np.pi, 100)
4 cos_x = np.cos(x)
```

```
In [3]: 1 # membuat line plot
2 fig, ax = plt.subplots()
3 _ = ax.plot(x, cos_x)
```



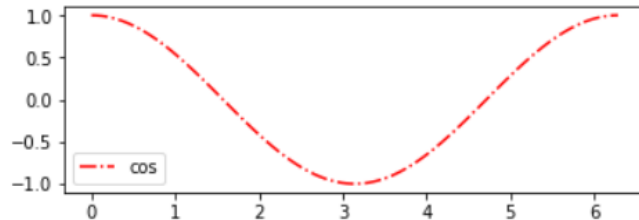
Sumbu x dan y pada kurva  $\cos$  seharusnya memiliki rasio yang sama karena keduanya merupakan satuan unit yang sama. Kita dapat mengaturnya menggunakan fungsi **.set\_aspect**.

```
In [4]: 1 # membuat sumbu x dan y memiliki rasio yang sama
2 fig, ax = plt.subplots()
3 _ = ax.plot(x, cos_x)
4 _ = ax.set_aspect('equal')
```



Kita dapat mengatur bentuk *marker* menggunakan parameter *linestyle*, ukuran *marker* menggunakan parameter *markersize*, warna menggunakan parameter *color*, dan memberikan *legend* menggunakan parameter *legend*.

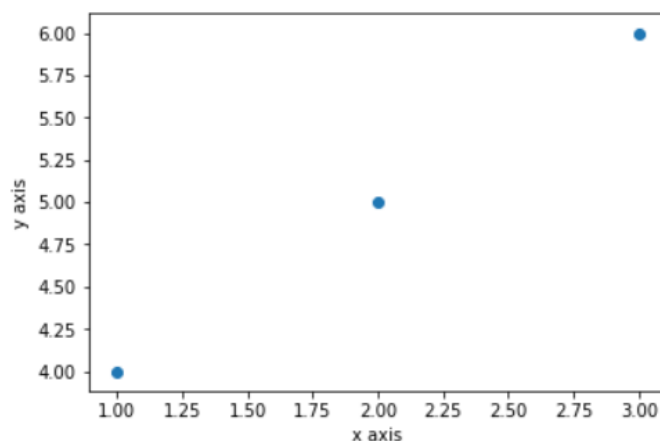
```
In [8]: 1 fig, ax = plt.subplots()
2         _ = ax.plot(x, cos_x, markersize=20, linestyle='-.',
3                     color='red', label='cos')
4         _ = ax.set_aspect('equal')
5         _ = ax.legend()
```



## Membuat Scatter Plot

**Scatter plot** biasanya digunakan untuk melakukan observasi dan menunjukkan hubungan relasi antara dua variabel *numeric*. Titik-titik pada scatter plot juga dapat menggambarkan pola dari data secara keseluruhan. Matplotlib menyediakan fungsi **scatter()** untuk mempermudah dalam visualisasi scatter plot.

```
In [13]: 1 # membuat scatter plot
2 x = np.array([1, 2, 3])
3 y = np.array([4, 5, 6])
4
5 fig, ax = plt.subplots()
6 _ = ax.scatter(x, y)
7 _ = ax.set_xlabel('x axis')
8 _ = ax.set_ylabel('y axis')
```

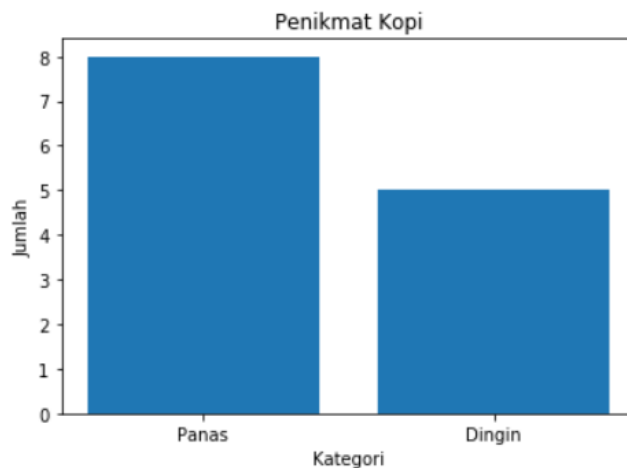


## Membuat Bar Plot

**Bar plot** digunakan untuk membandingkan perubahan tiap waktu pada beberapa kelompok data. Bar plot sangat bagus digunakan dalam visualisasi ketika perubahan data sangat besar dibandingkan dengan line plot. Bar plot biasanya memiliki dua sumbu yaitu sumbu x untuk jenis kelompok dan sumbu y untuk proporsi kelompok.

Matplotlib menyediakan fungsi **bar()** untuk mempermudah dalam visualisasi bar plot.

```
In [19]: 1 # membuat bar plot
          2 kategori = ['Panas', 'Dingin']
          3 jumlah = [8, 5]
          4
          5 fig, ax = plt.subplots()
          6 _ = ax.bar(kategori, jumlah)
          7 _ = ax.set_xlabel('Kategori')
          8 _ = ax.set_ylabel('Jumlah')
          9 _ = ax.set_title('Penikmat Kopi')
```



Matplotlib masih menyediakan banyak sekali fungsi untuk melakukan visualisasi dengan berbagai jenis grafik seperti *pie*, *histogram*, grafik 3-dimensi, dll.

## Latihan:

1. Diketahui dua buah array sebagai berikut :

$$\text{Matriks A} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$\times$

$$\text{Matriks B} = \begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix}$$

Buatlah program dengan numpy untuk melakukan operasi pada kedua matrik tersebut :

- a. Matrik A \* Matrik B
- b. Matrik A + Matrik B