Pandas

Pandas adalah library pada python yang memungkinkan kita untuk menganalisis data besar dan membuat kesimpulan berdasarkan teori statistik. Pandas dapat membersihkan kumpulan data yang berantakan, dan membuatnya dapat dibaca dan relevan. Data yang relevan sangat penting dalam ilmu data. Jika kamu punya tujuan untuk menjadi data engineer, mempelajari modul ini adalah tepat untuk kamu. Pandas wajib kamu pelajari.

Pandas adalah library open source. Kita bisa mengecek repository kodingnya di https://github.com/pandas-dev/pandas.

INSTALASI

Untuk menginstall pandas, kita dapat melakukannya dengan sangat mudah menggunakan PIP.

```
D:\MPP>pip install pandas
```

Setelah pandas kita install, untuk menggunakannya kita dapat mengimportnya di kodingan kita dengan perintah import pandas.

Contoh

```
import pandas as pd
```

Pada contoh diatas, kita gunakan alias as pd. Ini untuk mempermudah kita membuat kodingan saja. Jadi saat mau mengguakan pandas, kita cukup menggunakan aliasnya saja yaitu pd.

SERIES

Series, di pandas adalah array 1D yang nantinya akan kita oleh. Perhatikan contoh berikut. Pada contoh ini, kita akan mengubah List menjadi series pandas.

Contoh: https://s.id/mpp_ex253

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Output

```
0    1
1    7
2    2
dtype: int64
```

Pada contoh diatas, pada kolom pertama ada angka 0 1 2, itu adalah **label** indeks penomeran baris saja.

Kita dapat mengakses series dengan memanggil indeksnya dalam kurung siku. Perhatikan contoh berikut. Pada contoh ini kita akan memanggil indeks 0 pada series myvar.

Contoh: https://s.id/mpp_ex254

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar[0])
```

Output

```
1
```

Sebenarnya kita bisa menentukan sendiri label untuk indeks pada series. Berikut contohnya.

Contoh: https://s.id/mpp_ex255

```
import pandas as pd

a = [1, 7, 2]
```

```
myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

Output

```
x    1
y    7
z    2
dtype: int64
```

Pada contoh diatas, lihat pada kode pd.Series(a,index=[“x”, “y”, “z”]), disitu kita menentukan penamaan label indeks kita sendiri. Namun yang perlu dicatat adalah jumlah indeks yang kita tentukan harus tepat sama dengan jumlah komponen pada list yang akan kita jadikan series.

Dengan label indeks yang kita buat sendiri, untuk mengakses series tersebut, kita panggil menggunakan label tersebut juga. Perhatikan contoh berikut.

Contoh: https://s.id/mpp_ex256

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar["y"])
```

Output

```
7
```

Pada contoh diatas kita memanggil item di series yang memiliki indeks y.

Jika memang tujuan kita adalah kita ingin membuat label indeks series susuai yang kita mau, kita sebenarnya bisa menggunakan dictionary. Perhatikan contoh berikut.

Contoh: https://s.id/mpp_ex257

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

Output

```
day1    420
day2    380
day3    390
dtype: int64
```

Pada contoh diatas, kita bisa melihat dictionary calories. Saat diubah menjadi series, maka secara otomatis kunci pada calories akan menjadi label indeks.

DATAFRAME

Kumpulan data di Pandas biasanya berupa tabel multidimensi, yang disebut DataFrames. Jika series adalah tabel 1D, maka untuk tabel 2D adalah dataframe. Perhatikan contoh berikut.

Contoh: https://s.id/mpp_ex258

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```

Output

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Pada contoh diatas kita membuat dataframe pandas dengan menconvert directory yang masing-masing itemnya bernilai list atau array 1D.

Untuk mengakses data dataframe untuk baris tertentu, kita bisa menggunakan fungsi **loc[indeksnya]**. Perhatikan contoh berikut.

Contoh: https://s.id/mpp_ex259

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df.loc[0])
```

Output

```
calories    420
duration     50
Name: 0, dtype: int64
```

Pada contoh diatas, kode df.loc[0] artinya adalah kita mencoba mengakses dataframe df baris ke 1 (karena baris ke 1 adalah indeks ke 0). Kita bisa lihat hasilnya adalah calories 420, dan duration 50.

Dengan fungsi loc ini, kita juga bisa mengambil beberapa baris sekaligus. Perhatikan contoh berikut.

Contoh: https://s.id/mpp_ex260

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df.loc[[0, 1]])
```

Output

```
   calories  duration
0       420        50
1       380        40
```

Pada contoh diatas, kode df.loc[[0,1]] berarti adalah kita mengambil dataframe df pada baris ke 1 dan 2.

BERINTERAKSI DENGAN FILE CSV

Salah satu keunggulan pandas, adalah kemudahannya berinteraksi dengan file csv. Kita dapat membuka file csv dan menjadikannya sebagai dataframe menggunakan fungsi **pd.read_csv('nama_file.csv').** Perhatikan contoh berikut.

Contoh

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

Output

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

Jika Anda memiliki DataFrame besar dengan banyak baris, Pandas hanya akan memprintout 5 baris pertama, dan 5 baris terakhir saja di command line. Namun kita bisa mengubah jumlah maksimum ini dengan fungsi

pd.options.display.max_rows = 9999

Perhatikan contoh berikut

Contoh

```
import pandas as pd

pd.options.display.max_rows = 9999

df = pd.read_csv('data.csv')

print(df)
```

Pada contoh ini, outputnya tidak kita tampilkan di buku ini karena jumlah barisnya ada 169 sehingga cukup panjang.

MEMBACA DATA JSON

Kita juga bisa mengubah data JSON menjadi dataframe. Data JSON ini sangat sering sekali digunakan untuk berinteraksi antar sistem, biasaya digenerate dari aplikasi web. File ini sebenarnya adalah string yang bentuknya seperti dictionary. Perhatikan contoh berikut.

Contoh

```
import pandas as pd

data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
```

```
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409.1,
    "1":479.0,
    "2":340.0,
    "3":282.4,
    "4":406.0,
    "5":300.5
  }
```

```
}

df = pd.DataFrame(data)

print(df)
```

Output

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
5        60    102       127     300.5
```

FUNGSI HEAD DAN TAIL

Metode head() adalah fungsi yang mengembalikan header dan sejumlah baris tertentu, mulai dari atas dari sebuah dataframe. Perhatikan contoh berikut.

Contoh

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head(10))
```

Output

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
```

```
3         45    109         175      282.4
4         45    117         148      406.0
5         60    102         127      300.5
6         60    110         136      374.0
7         45    104         134      253.3
8         30    109         133      195.1
9         60     98         124      269.0
```

Pada contoh diatas, kode df.head(10) artinya adalah kita akan menampilkan header dataframe dan data 10 baris pertama. Ubah angka 10 menjadi angka yang kalian inginkan lalu perhatikan hasilnya. Jika angka 10 ini dikosongkan (tidak diisi angka), maka python akan menetapkan secara default yaitu 5.

Jika fungsi head adalah menampilkan data paling atas, maka fungsi tail(), adalah sebaliknya, ia akan menampilkan data paling terbawah.

Contoh

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.tail(10))
```

Output

```
     Duration  Pulse  Maxpulse  Calories
159        30     80       120     240.9
160        30     85       120     250.4
161        45     90       130     260.4
162        45     95       130     270.0
163        45    100       140     280.9
164        60    105       140     290.8
```

```
165         60    110         145      300.4
166         60    115         145      310.2
167         75    120         150      320.4
168         75    125         150      330.4
```

MENGHILANGKAN BARIS TIDAK LENGKAP

Terkadang, di suatu dataset ada beberapa baris yang kolomya tidak lengkap. Kita dapat menghapus baris-baris yang tidak lengkap ini dengan fungsi **dropna()**.

Sebelum kita bereksperimen dengan beberapa fungsi, kita tetapkan dataset kita yang akan kita pakai adalah data.csv yang memiliki data sebagaimana berikut.

data.csv

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3

```
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60    2020/12/26    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

Sekarang coba kita perhatikan pada baris ke 18 22 dan 28, terdapat kolom yang kosong. Kita dapat menghapusnya secara otomatis menggunakan fungsi dropna(). Perhatikan contoh berikut.

Contoh

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

Output

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	2020/12/26	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	243.0

Pada contoh diatas, kita bisa lihat indeks 18, 22, dan 28 sudah hilang secara otomatis. Perhatikan pada kode newdf.to_string(), kenapa diberi fungsi to_string()?. Fungsi to_string() ini memberikan isyarat ke python agar menampilkan semua data dataframe ke command line tanpa batasan.

MEREPLACE DATA PADA BARIS TIDAK LENGKAP

Ada kasus dimana kadang kita tidak ingin menghapus baris yang memiliki kolom tidak lengkap. Misalkan ada kolom kosong (NaN), maka kita ingin mengisinya secara otomatis dengan angka 0. Kita dapat menggunakan fungsi **fillna()**.

Contoh

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)

print(df.to_string())
```

Output

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112     130.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
```

```
21        60  '2020/12/21'    108       131     364.2
22        45           130    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60    2020/12/26    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132     130.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

Pada contoh diatas kode df.fillna(130, inplace = True), ini artinya adalah pada dataframe df, untuk kolom yang NaN (kosong) akan kita isi dengan angka 130. Anda bisa ganti 130 dengan apapun yang kamu suka.

Sebenarnya kita juga bisa definisikan fungsi fillna() pada kolom tertentu saja, misalkan pada kolom calories. Perhatikan contoh berikut.

Contoh

```
import pandas as pd

df = pd.read_csv('data.csv')

df["Calories"].fillna(130, inplace = True)

print(df.to_string())
```

Output

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
```

```
3         45  '2020/12/04'    109        175     282.4
4         45  '2020/12/05'    117        148     406.0
5         60  '2020/12/06'    102        127     300.0
6         60  '2020/12/07'    110        136     374.0
7        450  '2020/12/08'    104        134     253.3
8         30  '2020/12/09'    109        133     195.1
9         60  '2020/12/10'     98        124     269.0
10        60  '2020/12/11'    103        147     329.3
11        60  '2020/12/12'    100        120     250.7
12        60  '2020/12/12'    100        120     250.7
13        60  '2020/12/13'    106        128     345.3
14        60  '2020/12/14'    104        132     379.3
15        60  '2020/12/15'     98        123     275.0
16        60  '2020/12/16'     98        120     215.2
17        60  '2020/12/17'    100        120     300.0
18        45  '2020/12/18'     90        112     130.0
19        60  '2020/12/19'    103        123     323.0
20        45  '2020/12/20'     97        125     243.0
21        60  '2020/12/21'    108        131     364.2
22        45           NaN    100        119     282.0
23        60  '2020/12/23'    130        101     300.0
24        45  '2020/12/24'    105        132     246.0
25        60  '2020/12/25'    102        126     334.5
26        60    2020/12/26    100        120     250.0
27        60  '2020/12/27'     92        118     241.0
28        60  '2020/12/28'    103        132     130.0
29        60  '2020/12/29'    100        132     280.0
30        60  '2020/12/30'    102        129     380.3
31        60  '2020/12/31'     92        115     243.0
```

Pada contoh diatas, kode df["Calories"].fillna(130, inplace = True), artinya kita hanya mengeksekusi fungsi fillna hanya pada kolom Calories.

FUNGSI MEAN MEDIAN

Fungsi mean() adalah fungsi untuk menghitung nilai rata-rata dari dataframe. Perhatikan contoh berikut.

Contoh

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mean()

print(x)
```

Output

```
304.68
```

Pada contoh diatas, kode df[“Calories”].meand() artinya adalah kita menghitung nilai rata-rata untuk kolom Calories. Anda bisa juga lakukan ini untuk kolom yang lain.

Selain fungsi mean(), terdapat fungsi **median()**, untuk menghitung nilai tengah. Cara menggunakannya sama seperti mean.

Matplotlib

Matplotlib adalah package atau modul yang sangat familiar digunakan di python untuk visualisasi grafik. Saat belajar python, ini adalah termasuk modul yang wajib dipelajari.

Pada bab ini, kita tidak sediakan QR code dan link untuk online compiler karena matplotlib belum kompatibel dengan online compiler onecompiler yang kita gunakan. Kita berharap pembaca bisa mempraktekkannya langsung dari komputer atau laptopnya masing-masing.

INSTALASI

Untuk menginstall matplotlib kita dapat menggunakan PIP dengan sangat mudah. Kamu bisa buka kembali bab PIP jika lupa-lupa ingat apa itu PIP.

```
D:\MPP>pip install matplotlib
```

Setelah berhasil diinstall, maka untuk menggunakan matlplotlib kita tinggal import saja, dengan sintaks **import matplotib**.
Di dalam matplotlib terdapat sub modul yaitu pyplot. Kita akan menggunakan sub modul ini, karena kebanyakan fungsi yang

banyak digunakan ada di sub modul ini. Untuk menggunakannya, perhatikan contoh berikut:

Contoh

```
import matplotlib.pyplot as plt
```

Pada contoh diatas kita mengimport sub modul pyplot dari modul matplotlib lalu kita singkat namanya menjadi plt untuk digunakan di kodingan kita.

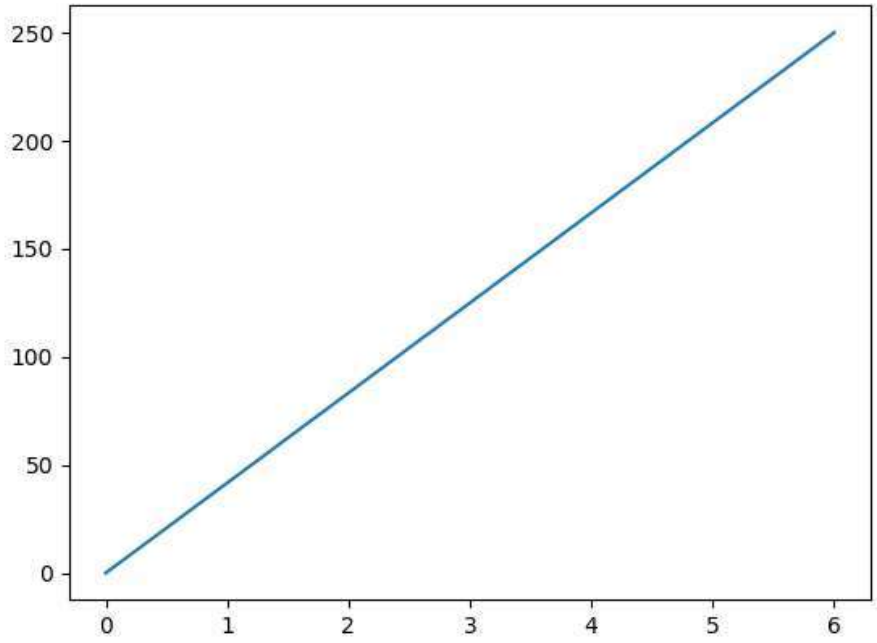
MENGEPLOT TITIK X DAN Y

Secara default, fungsi **plot(parameter1, parameter2)** menarik garis dari titik ke titik. Fungsi mengambil parameter untuk menentukan titik dalam diagram. Parameter 1 adalah array yang berisi titik-titik pada sumbu x . Parameter 2 adalah array yang berisi titik-titik pada sumbu y . Jika kita perlu memplot garis dari (0, 0) ke (6, 250), kita harus melewatkan dua array [0, 6] dan [0, 250] ke fungsi plot.

Contoh: https://s.id/mpp_ex261

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

Output



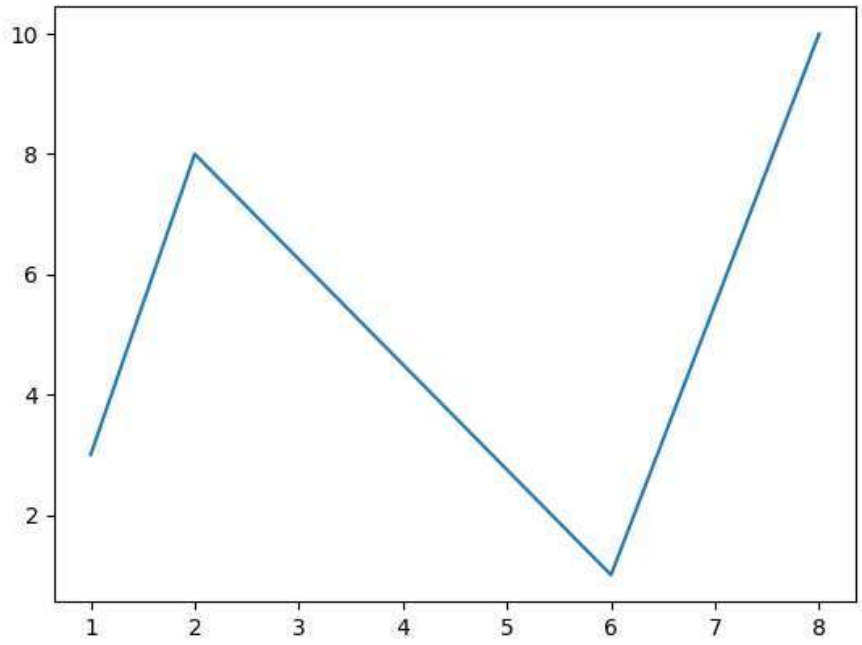
Pada contoh diatas, itu adalah contoh yang paling sederhana dari penggunaan pyplot matplotlob. Ada beberapa hal yang harus diperhatikan. Pertama kode *plt.plot(xpoints, ypoints)*, kode ini artinya adalah kita membuat plot dengan data sumbu-x-nya adalah data pada variabel xpoints dan data sumbu-y-nya adalah data pada variable ypoints. Sudah kita ketahui xpoints dan ypoints adalah variabel numpy array 1D. Namun dengan fungsi plt.plot(), hasil plot nya belum ditampilkan ke jendela. Untuk menampilkan hasisl plot ke layar, kita gunakan kode **plt.show()**. Contoh yang sederhana ini harap untuk diingat-ingat dan menjadi rujukan. Karena setelah ini akan banyak kita bahas lagi dan sebenarnya dasarnya adalah contoh ini.

Berikut contoh lain untuk titik yang lebih banyak.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Output

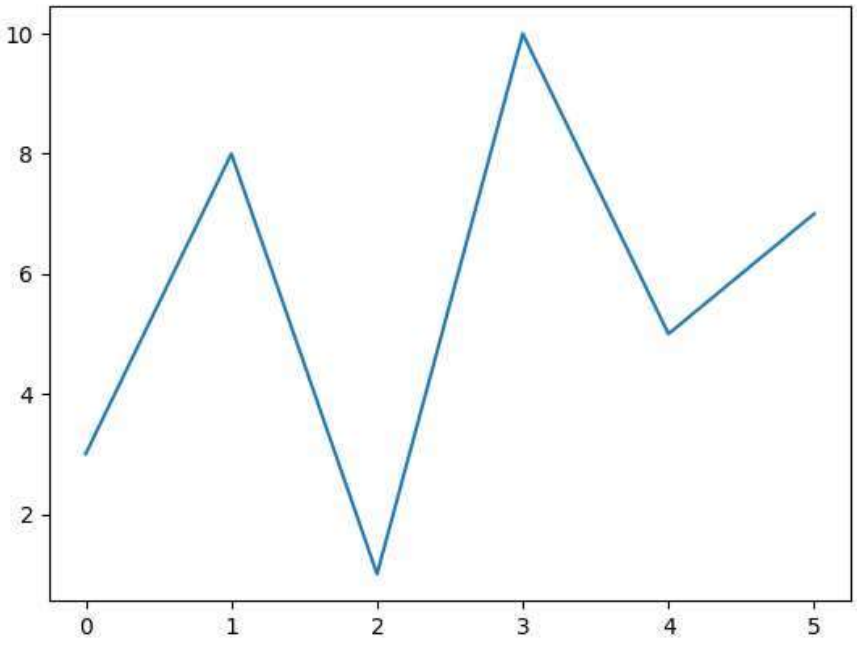


Sebenarnya, matplotlib sudah memiliki nilai sumbu bawaan yaitu dimulai dari 0,1,2, dst. Jadi jika kita menggunakan fungsi plt.plot() dengan hanya data sumbu y saja, itu tetap bisa dijalankan. Perhatikan contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])

plt.plot(ypoints)
plt.show()
```

Output



Perhatikan pada contoh diatas, terutama pada gambar sumbu x. Sumbu x dimulai dari angka 0 sampai 5 mengikuti jumlah datanya.

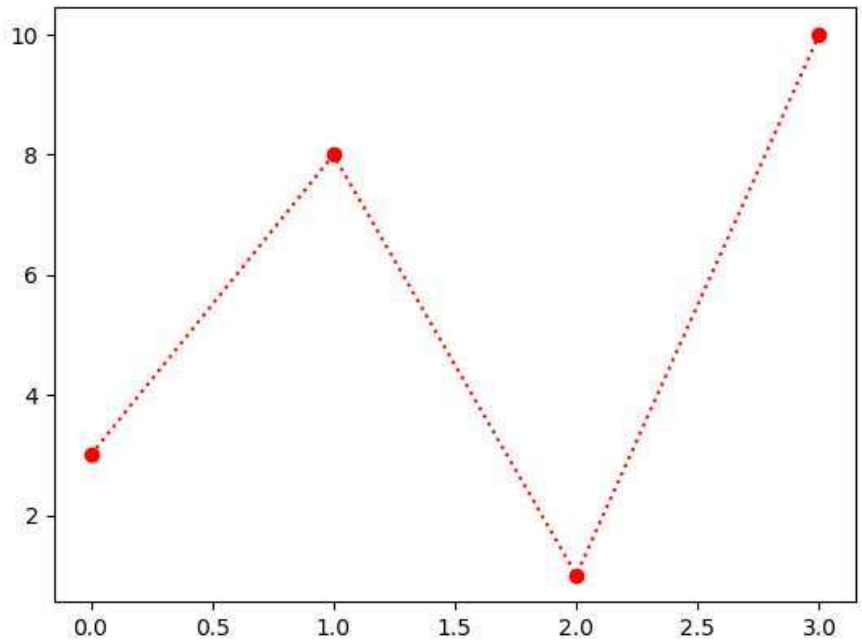
MARKER

Marker adalah penanda di setiap titik kordinat. Perhatikan contoh berikut untuk lebih jelasnya, terutama pada kode ang dicetak tebal.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'o:r')
plt.show()
```

Output



Pada contoh diatas, pada kode yang dicetak tebal, ’o:r’. Kode tersebut terdapat 3 karakter. Karakter pertama mendefinisikan tipe marker, karakter kedua mendefinisikan tipe garis, dan

karakter ketiga adalah mendefinisikan tipe warna. Sehingga ‘o:r’ artinya adalah kita mendefinisikan tipe marker o yairu sebuah simbol lingkaran. Sedangkan tipe garis adalah ‘:’ yaitu titik-titik putus-putus. Sedangkan tipe warnanya adalah ‘r’ yakni warna merah. Bisa kita lihat pada gambar hasil outputnya. Untuk simbol-simbol ini bisa kita lihat pada tiga tabel dibawah ini.

Marker	**Description**
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)
'p'	Pentagon
'H'	Hexagon
'h'	Hexagon
'v'	Triangle Down
'^'	Triangle Up
'<'	Triangle Left
'>'	Triangle Right
'1'	Tri Down
'2'	Tri Up
'3'	Tri Left
'4'	Tri Right
'|'	Vline
'_'	Hline

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

GARIS LEBIH DARI SATU

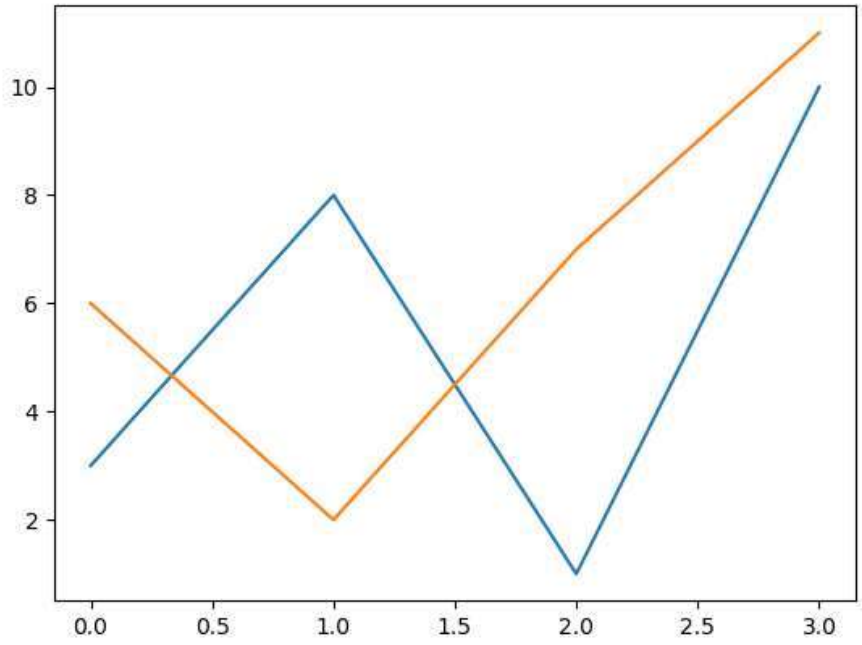
Kita dapat memplot baris sebanyak yang Anda suka hanya dengan menambahkan lebih banyak plt.plot(). Perhatikan contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)
```

```
plt.show()
```

Output



Pada contoh diatas coba perhatikan, sebelum kode plt.show(), kita menuliskan dua kali plt.plot() yaitu yang pertama plt.plot(y1) lalu plt.plot(y2), ini menjadikan 2 grafik dalam satu kanvas.

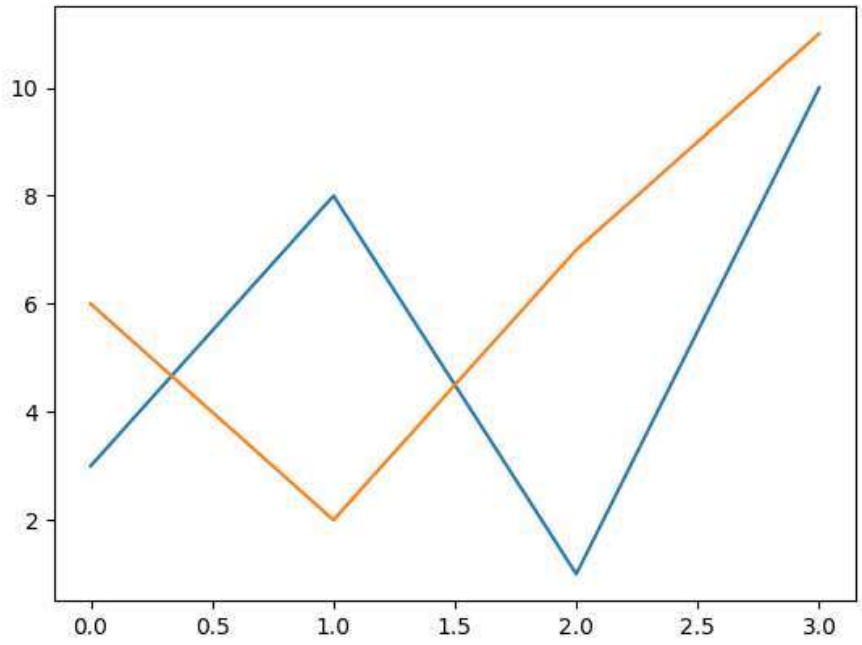
Jika contoh diatas adalah dua grafik dengan data sumbu x-nya dikosoongkan, lihat juga contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.array([0, 1, 2, 3])
```

```
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)

plt.show()
```

Output



Pada contoh diatas, perhatikan kode *plt.plot(x1,y1,x2,y2)*. Kode plt.plot() apabila memiliki parameter data array yang genap (berpasangan) maka akan dikenali sebagai pasangan data x dan y secara berurutan.

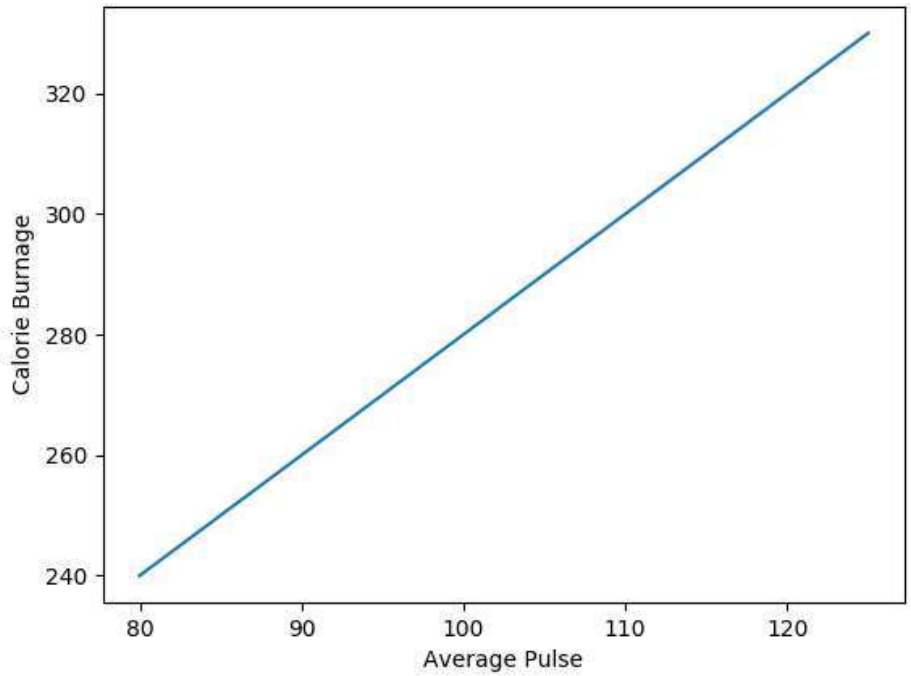
MENAMBAHKAN LABEL

Untuk menambahkan label pada sumbu x maupun sumbu y, kita dapat menggunakan fungsi xlabel() dan ylabel(). Perhatikan contoh berikut, terutama pada kodeyang dicetak tebal.

Contoh

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```

Output



Perhatikan pada kode yang tercetak tebal yaitu plt.xlabel("Average Pulse") dan plt.ylabel("Calorie Burnage").

Kode ini mengisyaratkan matplotlib untuk mencetak label pada sumbu x “Average Pulse” dan “Calorie Burnage” pada sumbu y.

Penempatan kode plt.xlabel() maupun plt.ylabel() harus sebelum plot.show(), namun tidak harus setelah plt.plot().

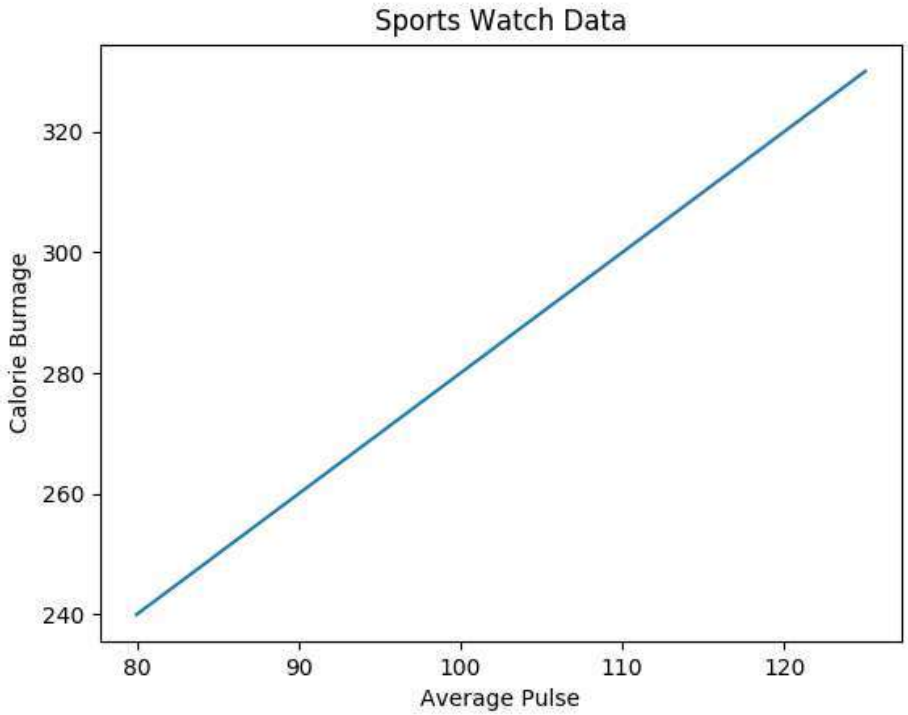
MENAMBAHKAN JUDUL

Untuk menambahkan judul pada plot kita dapat menggunakan fungsi title(). Perhatikan contoh berikut.

Contoh

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```

Output



GRID LINE

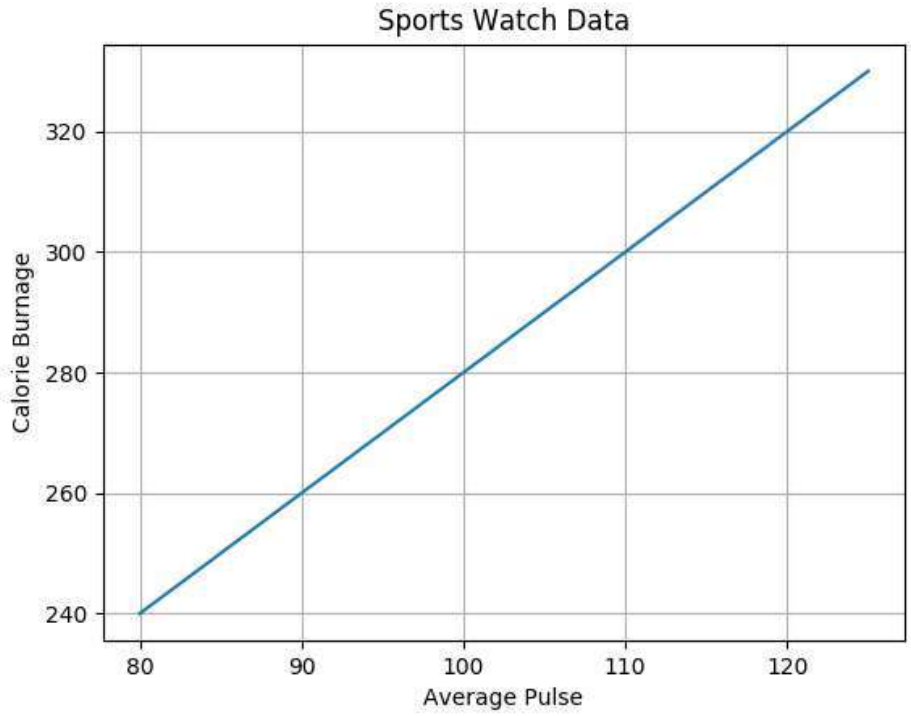
Jika kita lebih suka di kanvas plot kita terdapat garis grid line (kisi), maka kita bisa gunakan kode plt.grid() sebelum plot.show(). Perhatikan contoh berikut.

Contoh

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()
```

```
plt.show()
```

Output



Jika kita hanya ingin menampilkan hanya garis grid pada sumbu x saja, maka kode plt.grid() kita ubah menjadi plt.grid(axis = 'x'). Sedangkan jika kita ingin menampilkan hanya sumbu y, kita harus ubah kode plt.grid() menjadi plt.grid(axis = 'y'). Anda bisa mencobanya langsung dan melihat perubahannya.

SUBPLOT

Dengan subplot, kita dapat menunjukkan beberapa kanvas dalam satu jendela. Fungsi ini memiliki sintaks dengan 3 parameter:

plt.subplot(jumlah_baris, jumlah_kolom, nomor_cell)

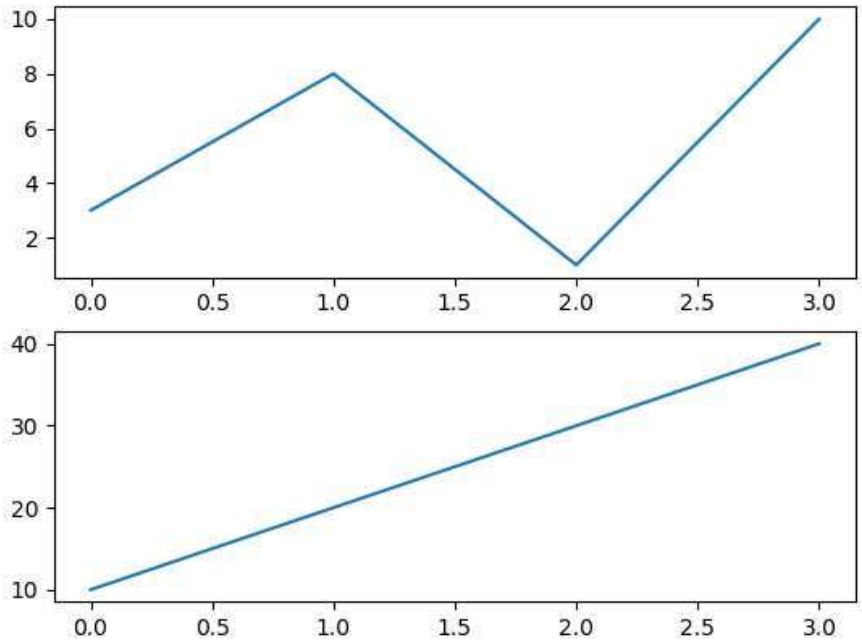
Pada parameter pertama adalah menentukan jumlah barisnya, parameter kedua adalah jumlah kolom, dan parameter ketiga adalah nomor cell yang saat ini aktif. Nomor cell aktif ini artinya adalah jika kita mengeksekusi kode plt.plot(), maka pada cell tersebut kanvas kita ditampilkan. Ingat, perhitungan nomor cell adalah dimulai dari baris 1 kolom 1, lalu beris 1 kolom 2, baris 1 kolom 3, …. , baris 1 kolom terakhir, lanjut baris 2 kolom 1, baris 2 kolom 2, dan seterusnya. Perhatikan contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)
```

```
plt.show()
```

Output



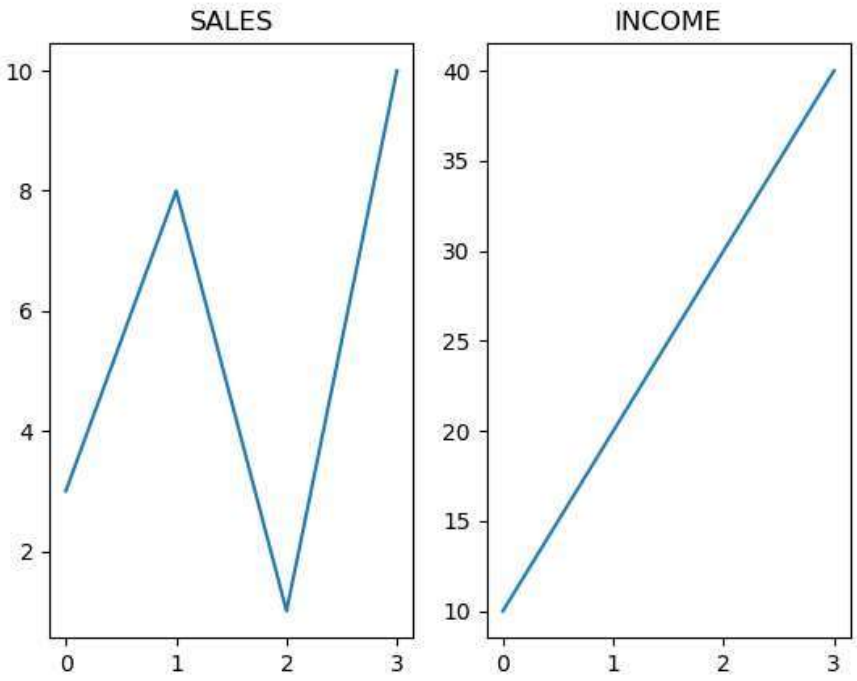
Pada contoh diatas, perhatikan pada kode tercetak tebal pertama plt.subplot(2,1,1). Kode ini artinya adalah kita mendefinisikan kita membagi kanvas menjadi beberapa bagian yaitu 2 baris dan 1 kolom. Lalu kita menetapkan cell yang aktif sekarag adalah cell nomor 1. Sehingga saat kode plt.plot(x,y) dibawahnya tepat, grafik tersebut akan ditampilkan di cell 1 yaitu baris 1 kolom 1. Lalu perhatikan pada kode tercetak tebal di bagian bawah yaitu plt.subplot(2,1,2). Kode ini mengisyaratkan cell aktifnya pindah ke cell nomor 2. Yang perlu kita perhatikan adalah nilai parameter jumlah baris dan kolom harus sama dengan yang kode plt.subplot yang lain.

Setiap kanvas pada setiap cell di dalam subplot, kita dapat mengatur title-nya masing-masing dengan fungsi title(). Perhatikan contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.show()
```

Output



Kita juga bisa membuat judul pada kanvas utama dengan fungsi **suptitle()**. Perhatikan contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")
```

```
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```

Output

SCATTER PLOT

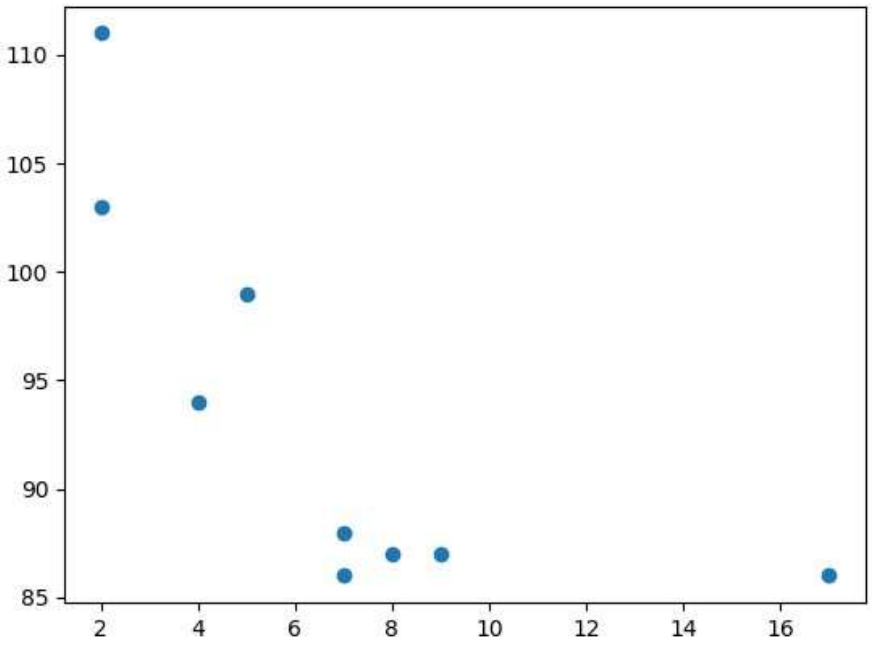
Sebenarnya, matplotlib menyediakan banyak cara mengeplot grafik. Fungsi plt.plot() adalah salah satunya. Salah satu lainnya

adalah **plt.scatter()**. Fungsi ini biasanya digunakan untuk melihat penyebaran data. Perhatikan contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4])
y = np.array([99,86,87,88,111,86,103,87,94])

plt.scatter(x, y)
plt.show()
```

Output

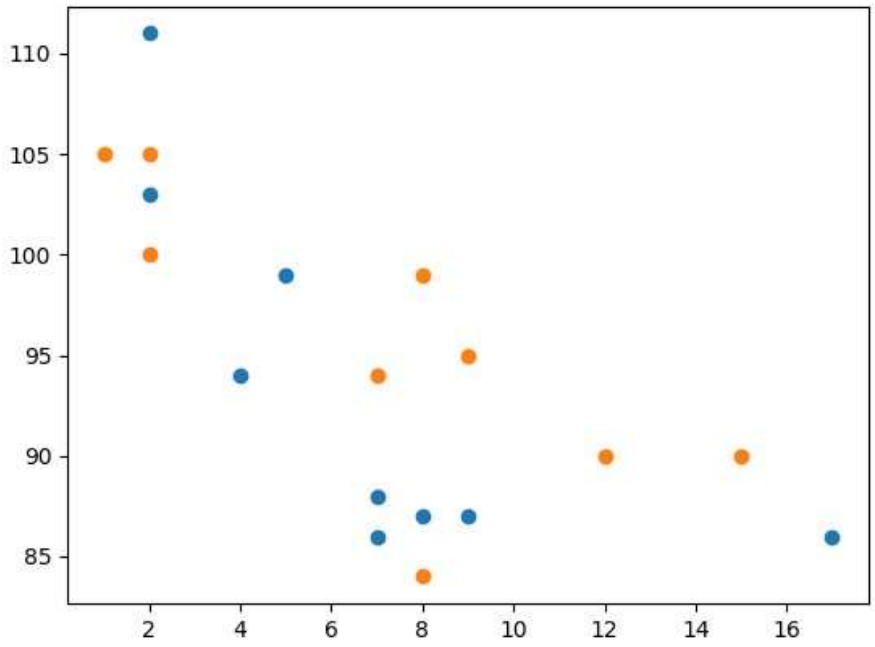


Kita juga bisa menampilkan 2 grafik dalam satu kanvas dengan scatter ini.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4])
y                                                =
np.array([99,86,87,88,111,86,103,87,94])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7])
y                                                =
np.array([100,105,84,105,90,99,90,95,94])
plt.scatter(x, y)

plt.show()
```

Output



Kita juga bisa mendefinisikan warna untuk setiap scatter plot kita dengan sintaks **plt.scatter(x,y,color="Blues").** Kode Blues adalah kode warna yang sudah ada di python. Selain dengan kode warna kita juga bisa menggunakan kode warna hexadecimal misal **plt.scatter(x,y,color="#88c999")**. Kode warna yang ada di python cukup banyak, kita bisa cek di https://matplotlib.org/2.0.1/examples/color/named_colors.html atau lihat pada gambar dibawah ini.


black
gray
silver
whitesmoke
rosybrown
firebrick
red
darksalmon
sienna
sandybrown
bisque
tan
moccasin
floralwhite
gold
darkkhaki
lightgoldenrodyellow
olivedrab
chartreuse
palegreen
darkgreen
seagreen
mediumspringgreen
lightseagreen
paleturquoise
darkcyan
darkturquoise
deepskyblue
aliceblue
slategray
royalblue
navy
blue
mediumpurple
darkorchid
plum
m
mediumvioletred
palevioletred
k
grey
lightgray
w
lightcoral
maroon
mistyrose
coral
seashell
peachpuff
darkorange
navajowhite
orange
darkgoldenrod
lemonchiffon
ivory
olive
yellowgreen
lawngreen
lightgreen
g
mediumseagreen
mediumaquamarine
mediumturquoise
darkslategray
c
cadetblue
skyblue
dodgerblue
slategrey
ghostwhite
darkblue
slateblue
rebeccapurple
darkviolet
violet
fuchsia
deeppink
crimson

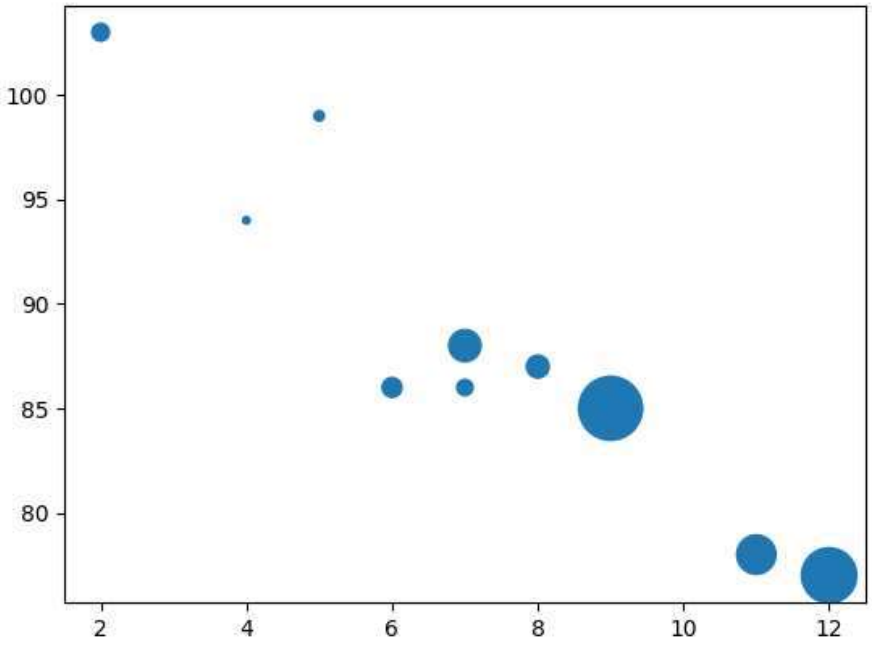
Dengan fungsi plt.scatter() kita juga dapat menentukan besar dari setiap titik yang akan ditampilkan di canvas. Perhatikan contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,103,86,94,78,77,85,86])
sizes = np.array([20,50,100,200,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes)
```

```
plt.show()
```

Output



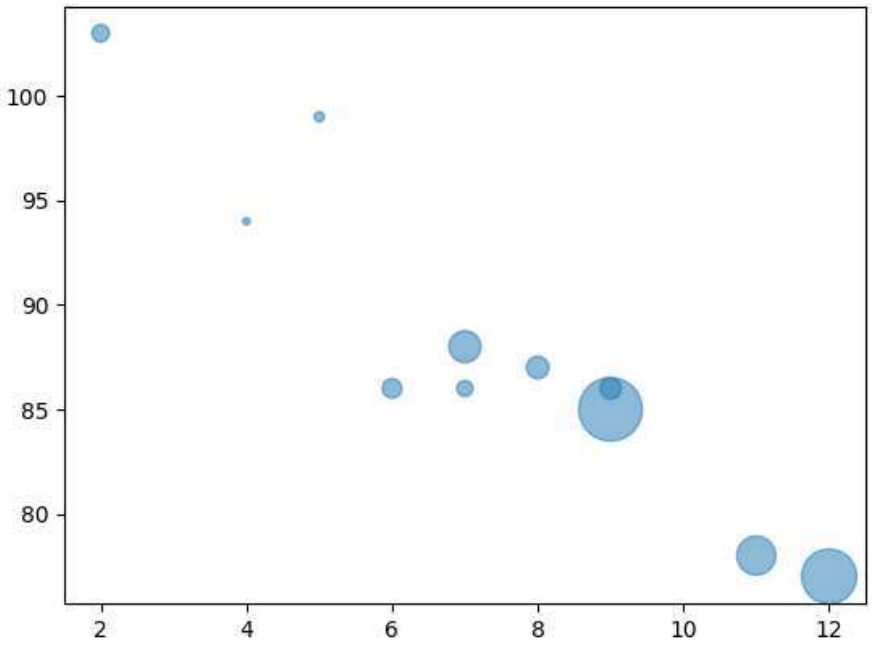
Saat kita bermain dengan size yang besar untuk data di scatter, terkadang antar data tumpang tindih. Kita bisa menggunakan transparansi dengan menambahkan parameter alpha pada sintaks plt.scatter(). Perhatikan contoh berikut.

Contoh

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,103,86,94,78,77,85,86])
sizes = np.array([20,50,100,200,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)
```

```
plt.show()
```

Output



FUNGSI LAIN PYPLOT

Pyplot memiliki fungsi yang cukup banyak. Sampai disini, kita yakin, jika anda mengikuti dari awal, anda sudah punya cukup skill untuk bisa membaca dokumentasi langsung dari pyplot dan mempraktekkannya sendiri. Untuk melihat fungsi apa saja yang disediakan oleh pyplot, anda bisa cek disini. Dibawah ini kita tampilkan dalam tabel berikut fungsi-fungsi yang disediakan oleh pyplot untuk menunjang plot grafik.

Fungsi dan sintaksnya	**Deskripsi**
acorr(x, *[, data])	Plot the autocorrelation of *x*.

angle_spectrum(x[, Fs, Fc, window, pad_to, ...])	Plot the angle spectrum.
annotate(text, xy, *args, **kwargs)	Annotate the point *xy* with text *text*.
arrow(x, y, dx, dy, **kwargs)	Add an arrow to the Axes.
autoscale([enable, axis, tight])	Autoscale the axis view to the data (toggle).
autumn()	Set the colormap to 'autumn'.
axes([arg])	Add an axes to the current figure and make it the current axes.
axhline([y, xmin, xmax])	Add a horizontal line across the axis.
axhspan(ymin, ymax[, xmin, xmax])	Add a horizontal span (rectangle) across the Axes.
axis(*args[, emit])	Convenience method to get or set some axis properties.
axline(xy1[, xy2, slope])	Add an infinitely long straight line.
axvline([x, ymin, ymax])	Add a vertical line across the Axes.
axvspan(xmin, xmax[, ymin, ymax])	Add a vertical span (rectangle) across the Axes.
bar(x, height[, width, bottom, align, data])	Make a bar plot.
bar_label(container[, labels, fmt, ...])	Label a bar plot.
barbs(*args[, data])	Plot a 2D field of barbs.
barh(y, width[, height, left, align])	Make a horizontal bar plot.
bone()	Set the colormap to 'bone'.
box([on])	Turn the axes box on or off on the current axes.
boxplot(x[, notch, sym, vert, whis, ...])	Draw a box and whisker plot.

broken_barh(xranges, yrange, *[, data])	Plot a horizontal sequence of rectangles.
cla()	Clear the current axes.
clabel(CS[, levels])	Label a contour plot.
clf()	Clear the current figure.
clim([vmin, vmax])	Set the color limits of the current image.
close([fig])	Close a figure window.
cohere(x, y[, NFFT, Fs, Fc, detrend, ...])	Plot the coherence between *x* and *y*.
colorbar([mappable, cax, ax])	Add a colorbar to a plot.
connect(s, func)	Bind function *func* to event *s*.
contour(*args[, data])	Plot contour lines.
contourf(*args[, data])	Plot filled contours.
cool()	Set the colormap to 'cool'.
copper()	Set the colormap to 'copper'.
csd(x, y[, NFFT, Fs, Fc, detrend, window, ...])	Plot the cross-spectral density.
delaxes([ax])	Remove an **Axes** (defaulting to the current axes) from its figure.
disconnect(cid)	Disconnect the callback with id *cid*.
draw()	Redraw the current figure.
draw_if_interactive()	Redraw the current figure if in interactive mode.
errorbar(x, y[, yerr, xerr, fmt, ecolor, ...])	Plot y versus x as lines and/or markers with attached errorbars.
eventplot(positions[, orientation, ...])	Plot identical parallel lines at the given positions.
figimage(X[, xo, yo, alpha, norm, cmap, ...])	Add a non-resampled image to the figure.

figlegend(*args, **kwargs)	Place a legend on the figure.
fignum_exists(num)	Return whether the figure with the given id exists.
figtext(x, y, s[, fontdict])	Add text to figure.
figure([num, figsize, dpi, facecolor, ...])	Create a new figure, or activate an existing figure.
fill(*args[, data])	Plot filled polygons.
fill_between(x, y1[, y2, where, ...])	Fill the area between two horizontal curves.
fill_betweenx(y, x1[, x2, where, step, ...])	Fill the area between two vertical curves.
findobj([o, match, include_self])	Find artist objects.
flag()	Set the colormap to 'flag'.
gca(**kwargs)	Get the current Axes.
gcf()	Get the current figure.
gci()	Get the current colorable artist.
get(obj, *args, **kwargs)	Return the value of an **Artist**'s *property*, or print all of them.
get_current_fig_manager()	Return the figure manager of the current figure.
get_figlabels()	Return a list of existing figure labels.
get_fignums()	Return a list of existing figure numbers.
get_plot_commands()	Get a sorted list of all of the plotting commands.
getp(obj, *args, **kwargs)	Return the value of an **Artist**'s *property*, or print all of them.
ginput([n, timeout, show_clicks, mouse_add, ...])	Blocking call to interact with a figure.
gray()	Set the colormap to 'gray'.

grid([visible, which, axis])	Configure the grid lines.
hexbin(x, y[, C, gridsize, bins, xscale, ...])	Make a 2D hexagonal binning plot of points *x*, *y*.
hist(x[, bins, range, density, weights, ...])	Plot a histogram.
hist2d(x, y[, bins, range, density, ...])	Make a 2D histogram plot.
hlines(y, xmin, xmax[, colors, linestyles, ...])	Plot horizontal lines at each *y* from *xmin* to *xma x*.
hot()	Set the colormap to 'hot'.
hsv()	Set the colormap to 'hsv'.
imread(fname[, format])	Read an image from a file into an array.
imsave(fname, arr, **kwargs)	Save an array as an image file.
imshow(X[, cmap, norm, aspect, ...])	Display data as an image, i.e., on a 2D regular raster.
inferno()	Set the colormap to 'inferno'.
install_repl_displayhook()	Install a repl display hook so that any stale figure are automatically redrawn when control is returned to the repl.
ioff()	Disable interactive mode.
ion()	Enable interactive mode.
isinteractive()	Return whether plots are updated after every plotting command.
jet()	Set the colormap to 'jet'.
legend(*args, **kwargs)	Place a legend on the Axes.
locator_params([axis, tight])	Control behavior of major tick locators.
loglog(*args, **kwargs)	Make a plot with log scaling on both the x and y axis.

magma()	Set the colormap to 'magma'.
magnitude_spectrum(x[, Fs, Fc, window, ...])	Plot the magnitude spectrum.
margins(*margins[, x, y, tight])	Set or retrieve autoscaling margins.
matshow(A[, fignum])	Display an array as a matrix in a new figure window.
minorticks_off()	Remove minor ticks from the Axes.
minorticks_on()	Display minor ticks on the Axes.
new_figure_manager(num, *args, **kwargs)	Create a new figure manager instance.
nipy_spectral()	Set the colormap to 'nipy_spectral'.
pause(interval)	Run the GUI event loop for *interval* seconds.
pcolor(*args[, shading, alpha, norm, cmap, ...])	Create a pseudocolor plot with a non-regular rectangular grid.
pcolormesh(*args[, alpha, norm, cmap, vmin, ...])	Create a pseudocolor plot with a non-regular rectangular grid.
phase_spectrum(x[, Fs, Fc, window, pad_to, ...])	Plot the phase spectrum.
pie(x[, explode, labels, colors, autopct, ...])	Plot a pie chart.
pink()	Set the colormap to 'pink'.
plasma()	Set the colormap to 'plasma'.
plot(*args[, scalex, scaley, data])	Plot y versus x as lines and/or markers.
plot_date(x, y[, fmt, tz, xdate, ydate, data])	Plot coercing the axis to treat floats as dates.
polar(*args, **kwargs)	Make a polar plot.
prism()	Set the colormap to 'prism'.

psd(x[, NFFT, Fs, Fc, detrend, window, ...])	Plot the power spectral density.
quiver(*args[, data])	Plot a 2D field of arrows.
quiverkey(Q, X, Y, U, label, **kwargs)	Add a key to a quiver plot.
rc(group, **kwargs)	Set the current **rcParams**. *group* is the grouping for the rc, e.g., for lines.linewidth the group is lines, for axes.facecolor, the group is axes, and so on. Group may also be a list or tuple of group names, e.g., (*xtick*, *ytick*). *kwargs* is a dictionary attribute name/value pairs, e.g.,::.
rc_context([rc, fname])	Return a context manager for temporarily changing rcParams.
rcdefaults()	Restore the **rcParams** from Matplotlib's internal default style.
rgrids([radii, labels, angle, fmt])	Get or set the radial gridlines on the current polar plot.
savefig(*args, **kwargs)	Save the current figure.
sca(ax)	Set the current Axes to *ax* and the current Figure to the parent of *ax*.
scatter(x, y[, s, c, marker, cmap, norm, ...])	A scatter plot of *y* vs.
sci(im)	Set the current image.
semilogx(*args, **kwargs)	Make a plot with log scaling on the x axis.
semilogy(*args, **kwargs)	Make a plot with log scaling on the y axis.

set_cmap(cmap)	Set the default colormap, and applies it to the current image if any.
set_loglevel(*args, **kwargs)	Set Matplotlib's root logger and root logger handler level, creating the handler if it does not exist yet.
setp(obj, *args, **kwargs)	Set one or more properties on an **Artist**, or list allowed values.
show(*[, block])	Display all open figures.
specgram(x[, NFFT, Fs, Fc, detrend, window, ...])	Plot a spectrogram.
spring()	Set the colormap to 'spring'.
spy(Z[, precision, marker, markersize, ...])	Plot the sparsity pattern of a 2D array.
stackplot(x, *args[, labels, colors, ...])	Draw a stacked area plot.
stairs(values[, edges, orientation, ...])	A stepwise constant function as a line with bounding edges or a filled plot.
stem(*args[, linefmt, markerfmt, basefmt, ...])	Create a stem plot.
step(x, y, *args[, where, data])	Make a step plot.
streamplot(x, y, u, v[, density, linewidth, ...])	Draw streamlines of a vector flow.
subplot(*args, **kwargs)	Add an Axes to the current figure or retrieve an existing Axes.
subplot2grid(shape, loc[, rowspan, colspan, fig])	Create a subplot at a specific location inside a regular grid.
subplot_mosaic(mosaic, *[, sharex, sharey, ...])	Build a layout of Axes based on ASCII art or nested lists.
subplot_tool([targetfig])	Launch a subplot tool window for a figure.

subplots([nrows, ncols, sharex, sharey, ...])	Create a figure and a set of subplots.
subplots_adjust([left, bottom, right, top, ...])	Adjust the subplot layout parameters.
summer()	Set the colormap to 'summer'.
suptitle(t, **kwargs)	Add a centered suptitle to the figure.
switch_backend(newbackend)	Close all open figures and set the Matplotlib backend.
table([cellText, cellColours, cellLoc, ...])	Add a table to an **Axes**.
text(x, y, s[, fontdict])	Add text to the Axes.
thetagrids([angles, labels, fmt])	Get or set the theta gridlines on the current polar plot.
tick_params([axis])	Change the appearance of ticks, tick labels, and gridlines.
ticklabel_format(*[, axis, style, ...])	Configure the **ScalarFormatter** used by default for linear axes.
tight_layout(*[, pad, h_pad, w_pad, rect])	Adjust the padding between and around subplots.
title(label[, fontdict, loc, pad, y])	Set a title for the Axes.
tricontour(*args, **kwargs)	Draw contour lines on an unstructured triangular grid.
tricontourf(*args, **kwargs)	Draw contour regions on an unstructured triangular grid.
tripcolor(*args[, alpha, norm, cmap, vmin, ...])	Create a pseudocolor plot of an unstructured triangular grid.
triplot(*args, **kwargs)	Draw a unstructured triangular grid as lines and/or markers.

twinx([ax])	Make and return a second axes that shares the *x*-axis.
twiny([ax])	Make and return a second axes that shares the *y*-axis.
uninstall_repl_displayhook()	Uninstall the Matplotlib display hook.
violinplot(dataset[, positions, vert, ...])	Make a violin plot.
viridis()	Set the colormap to 'viridis'.
vlines(x, ymin, ymax[, colors, linestyles, ...])	Plot vertical lines at each *x* from *ymin* to *yma x*.
waitforbuttonpress([timeout])	Blocking call to interact with the figure.
winter()	Set the colormap to 'winter'.
xcorr(x, y[, normed, detrend, usevlines, ...])	Plot the cross correlation between *x* and *y*.
xkcd([scale, length, randomness])	Turn on xkcd sketch-style drawing mode.
xlabel(xlabel[, fontdict, labelpad, loc])	Set the label for the x-axis.
xlim(*args, **kwargs)	Get or set the x limits of the current axes.
xscale(value, **kwargs)	Set the x-axis scale.
xticks([ticks, labels])	Get or set the current tick locations and labels of the x-axis.
ylabel(ylabel[, fontdict, labelpad, loc])	Set the label for the y-axis.
ylim(*args, **kwargs)	Get or set the y-limits of the current axes.
yscale(value, **kwargs)	Set the y-axis scale.
yticks([ticks, labels])	Get or set the current tick locations and labels of the y-axis.