# 12장 데이터 시각화

- 설치 후 '런타임 다시 시작'

```
!apt-get update
!apt-get install -y fonts-nanum
!fc-cache -fv
!rm ~/.cache/matplotlib -rf
```

```
Ign:1 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InRelease
Ign:2 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  InRelease
Get:3 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Release [697 B]
Hit:4 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Release
Get:5 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease [3,626 B]
Get:6 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Release.gpg [836 B]
Get:7 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease [15.9 kB]
Get:8 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:9 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:11 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:12 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ Packages [62.9 kB]
Hit:13 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
Ign:14 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Packages
Get:14 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Packages [695 kB]
Hit:15 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease
Get:16 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:17 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease [21.3 kB]
Get:18 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [510 kB]
Get:19 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main Sources [1,786 kB]
Get:20 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [2,263 kB]
Get:21 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [1,420 kB]
Get:22 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [26.7 kB]
Get:23 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [2,698 kB]
Get:24 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main amd64 Packages [914 kB]
Get:25 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [2,195 kB]
Get:26 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic/main amd64 Packages [44.1 kB]
Get:27 http://archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages [544 kB]
Get:28 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [39.4 kB]
Fetched 13.5 MB in 4s (3,036 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  fonts-nanum
0 upgraded, 1 newly installed, 0 to remove and 98 not upgraded.
Need to get 9,604 kB of archives.
After this operation, 29.5 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 fonts-nanum all 20170925-1 [9,604 kB]
Fetched 9,604 kB in 2s (5,382 kB/s)
Selecting previously unselected package fonts-nanum.
(Reading database ... 160837 files and directories currently installed.)
Preparing to unpack .../fonts-nanum_20170925-1_all.deb ...
Unpacking fonts-nanum (20170925-1) ...
Setting up fonts-nanum (20170925-1) ...
Processing triggers for fontconfig (2.12.6-0ubuntu2) ...
/usr/share/fonts: caching, new cache contents: 0 fonts, 1 dirs
/usr/share/fonts/truetype: caching, new cache contents: 0 fonts, 3 dirs
/usr/share/fonts/truetype/humor-sans: caching, new cache contents: 1 fonts, 0 dirs
/usr/share/fonts/truetype/liberation: caching, new cache contents: 16 fonts, 0 dirs
/usr/share/fonts/truetype/nanum: caching, new cache contents: 10 fonts, 0 dirs
/usr/local/share/fonts: caching, new cache contents: 0 fonts, 0 dirs
/root/.local/share/fonts: skipping, no such directory
/root/.fonts: skipping, no such directory
/var/cache/fontconfig: cleaning cache directory
/root/.cache/fontconfig: not cleaning non-existent cache directory
/root/.fontconfig: not cleaning non-existent cache directory
fc-cache: succeeded
```

- 숫자 데이터를 그래프로 시각화하면 한눈에 데이터를 파악할 수 있어서 편리

- 시각화 라이브러리인 matplotlib을 이용해 파이썬에서 숫자로 이뤄진 데이터를 다양한 그래프로 보기 좋게 표현하는 시각화 방법

## 12.1 matplotlib로 그래프 그리기

- matplotlib는 파이썬에서 데이터를 효과적으로 시각화하기 위해 만든 라이브러리

- matplotlib 는 MATLAB(과학 및 공학 연산을 위한 소프트웨어)의 시각화 기능을 모델링
- matplotlib# 이용하면 몇 줄의 코드로 간단하게 2차원 선 그래프(plot), 산점도(scatter plot), 막대 그래프(bar chart), 히스토그램(histogram), 파이 그래프(pie chart) 등을 그릴 수 있음

- matplotlib 홈페이지(https://matplotlib.org/)

- matplotUb의 그래프 기능을 이용하려면 matplotlib의 서브모듈(submodule)을 불러와야 함
- matplotlib의 서브모둘도 NumPy나 pandas처럼 다음과 같이 'import ~ as' 형식으로 불러옴

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

- matplotlib의 서브모듈을 불러오면 matplotlib.pyplot 대신 pit를 이용
- matplotlib에서 제공하는 그래프 기능을 'pit.그래프_함수()'형식으로 사용

- 별도의 팝업창에 그래프를 출력하려면 '그래프_함수()' 를 실행하기 전에 다음을 실행
- %matplotlib qt

- 그래프를 다시 코드 결과 출력 부분에 출력하려면 '그래프_함수()'를 실행하기 전에 다음을 실행
- %matplotlib inline

## 선 그래프

### 기본적인 선 그래프 그리기

- 선 그래프는 그래프 중 가장 기본이 되는 그래프
- 순서가 있는 숫자 데이터를 시각화하거나 시간 에 따라 변하는 숫자 데이터를 시각화하는 데 많이 사용

- matplotlib에서는 다음과 같은 형식으 로 2차원 선 그래프를 그림
- plt.plot([x,] y [,fmt])

- plt.plot(y)
- plt.plot(y, fmt)
- plt.plot(x, y)
- plt.plot(x, y, fmt)

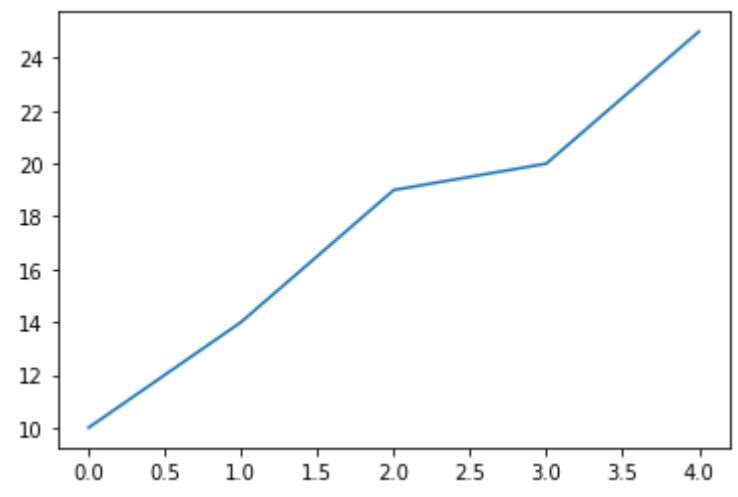
- 선 그래프를 그리기 위해 먼저 다음과 같이 숫자로 이뤄진 리스트 데이터를 생성

```
data1 = [10, 14, 19, 20, 25]
```

- 데이터 data1을 이용해 선 그래프를 생성

```
plt.plot(data1)
```

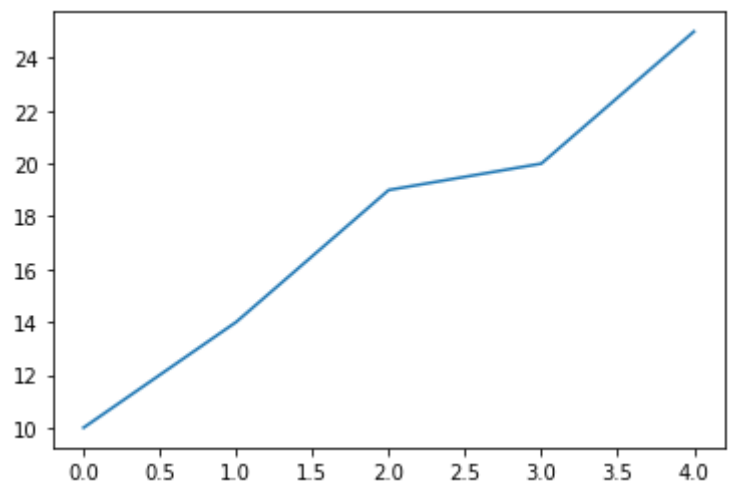
```
[<matplotlib.lines.Line2D at 0x7fa104b8cf90>]
```




- 그래프 객체의 정보 없이 그래프만 출력하려면

- '그래프_함수()'를 실행한 이후에 show()를 실행

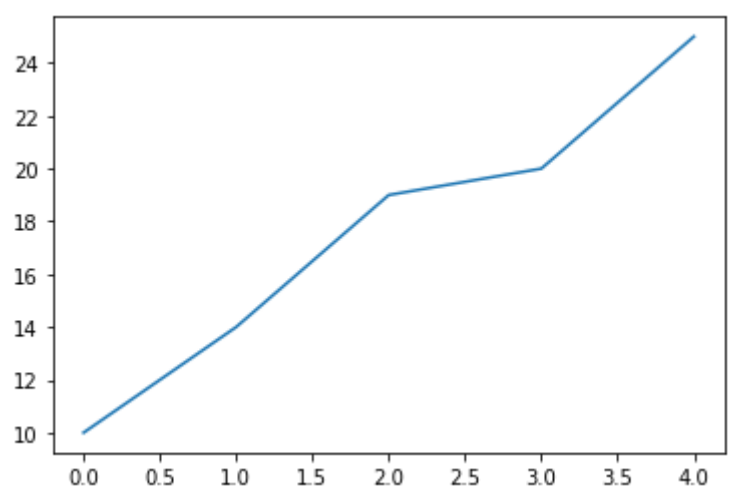
```
plt.plot(data1)
plt.show()
```




- 그래프가 출력되지 않는다면 '%natplotlib inline'을 실행한 후에 위의 코드를 다시 실행

```
plt.plot(data1)
```

```
[<matplotlib.lines.Line2D at 0x7fa1045fbd90>]
```




- 다시 결과 창에 그래프를 출력하게 하려면 다음과 같이 실행

```
%matplotlib inline
```

- X값과 y값 이 모두 있는 데이터를 2차원 선 그래프

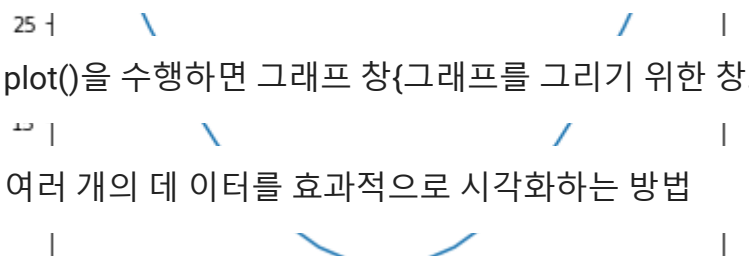
```
import numpy as np

x = np.arange(-4.5, 5, 0.5) # 배열 x 생성. 범위: [-4.5, 5), 0.5씩 증가
y = 2*x**2 # 수식을 이용해 배열 x에 대응하는 배열 y 생성
[x,y]
```

```
[array([-4.5, -4. , -3.5, -3. , -2.5, -2. , -1.5, -1. , -0.5,  0. ,  0.5,
         1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5]),
 array([40.5, 32. , 24.5, 18. , 12.5,  8. ,  4.5,  2. ,  0.5,  0. ,  0.5,
         2. ,  4.5,  8. , 12.5, 18. , 24.5, 32. , 40.5])]
```

- 생성된 x값과 y값으로 plot()을 이용해 그래프

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


- 여러 그래프 그리기




- plot()을 수행하면 그래프 창{그래프를 그리기 위한 창으로 Figure라고 함)이 하나 생성되고 그 안에 하나의 선 그래프 그렸음
- 여러 개의 데 이터를 효과적으로 시각화하는 방법
- 그래프를 위한 수식 생성

```
import numpy as np

x = np.arange(-4.5, 5, 0.5)
y1 = 2*x**2
y2 = 5*x + 30
y3 = 4*x**2 + 10
```

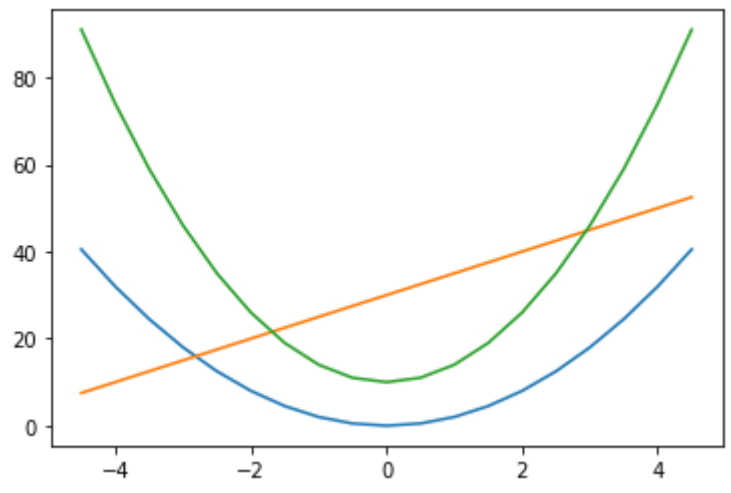
- 그래프 창에 여러 개의 데이터를 선 그래프로 표시하는 방법
- plt.plot([x2,] y2 [, fmt2])
- plt.plot([xn,] yn [, fmtn])

"

- 하나의 plot()에 표시하고자 하는 데이터를 모두 입력하는 방법으로도 하나의 그래프 창 에 여러 개의 선 그래프를 그릴 수 있음
- plt.plot(x1, y1 [, fmtl], x2, y2 [, fmt2], • • • , xn, yn [, fmtn])

- 생성한 데이터 x, y1, y2, y3를 이용해 그래프
- 수식으로 생성한 y1, y2, y3는 모두 공통으로 x에 대응
- y1, y2, y3 데이터를 하나의 그래프 창에 그리려면 다음과 같이 코드를 작성

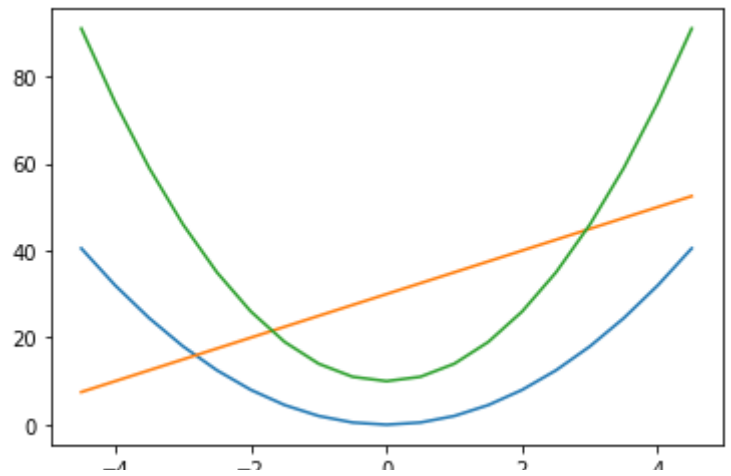
```
import matplotlib.pyplot as plt

plt.plot(x, y1)
plt.plot(x, y2)
plt.plot(x, y3)
plt.show()
```




- x에 대한 데이터 y1, y2, y3이 각각 다른 색의 선 그래프로 그려진 것을 볼 수 있음

- 하나의 plot()에 여러 개의 x, y 데이터 쌍을 입력해서 하나의 그래프 창에 여러 개의 선 그래프를 그릴 수도 있음

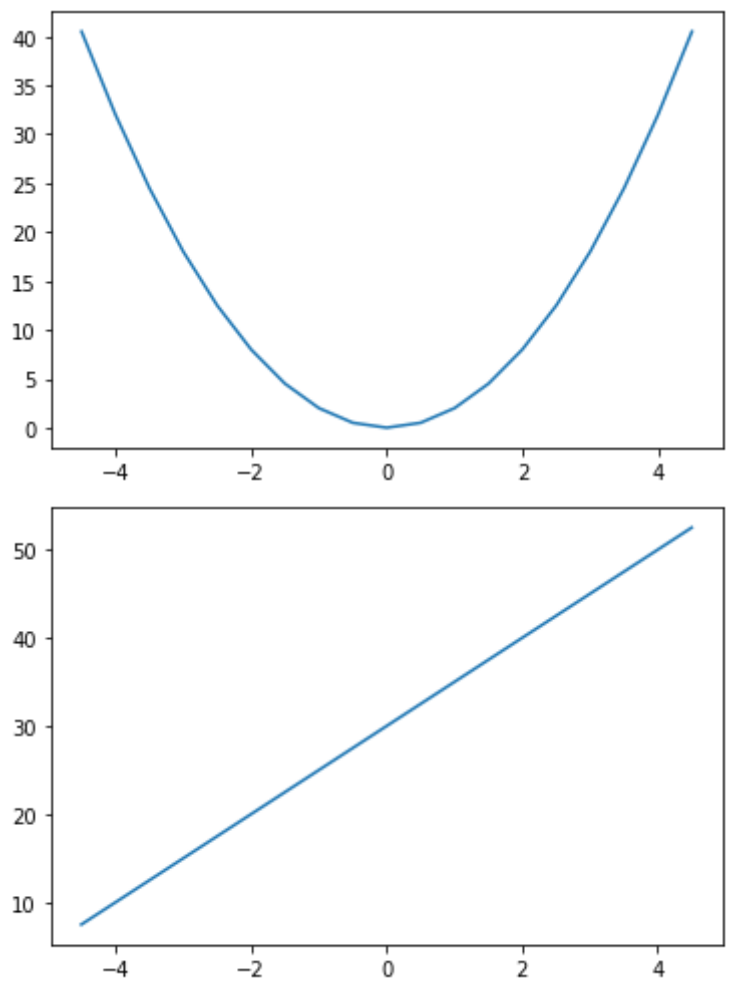
```
plt.plot(x, y1, x, y2, x, y3)
plt.show()
```




- 하나의 그래프 창에 여러 개의 그래프를 그리는 대신 새로운 그래프 창에 그래프를 그리려면
- 그래프_함수()'를 실행하기 전에 먼저 그래프 창을 생성
- plt.figure()

- figure()로 새로운 그래프 창을 생성해서 그래프를 그리는 예
- plot()과 같은 '그래프-함수 ()'를 처음 수행하는 경우에는 새로운 그래프 창이 자동으로 생성되므로 figure()를 실행할 필요가 없음

```
plt.plot(x, y1) #처음 그리기 함수를 수행하면 그래프 창이 자동으로 생성됨
plt.figure() # 새로운 그래프 창을 생성함
plt.plot(x, y2) # 새롭게 생성된 그래프 창에 그래프를 그림

plt.show()
```




- '그래프_함수()'를 실행하기 전에 figure()를 실행하면 새로운 그래프 창이 생성되고 그곳에 그래프를 그림

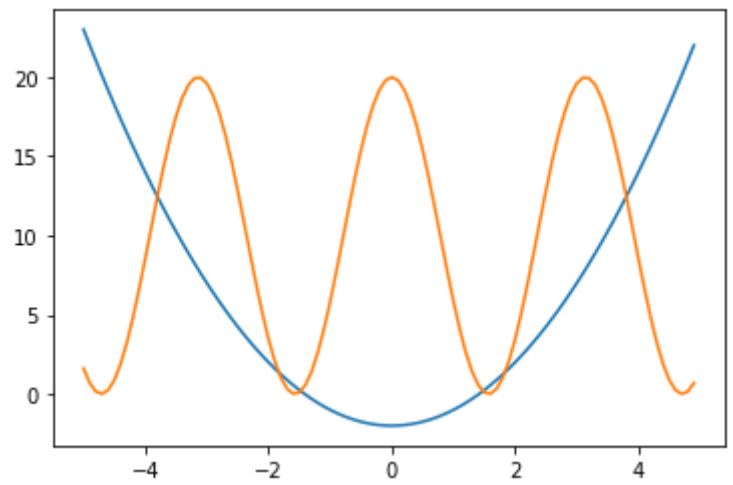
- 그래프 창의 번호를 명시적으로 지정한 후 해당 창에 그래프를 그릴 수도 있음
- plt.figure(n)
- 인자 n(정수)을 지정하면 지정된 번호로 그래프 창이 지정
- '그래프-함수()'를 실행하면 지정된 번호의 그래프 창에 그래프가 그려짐
- 지정된 번호의 그래프 창이 없다면 새로 그래 프 창을 생성한 후에 그래프가 그려짐

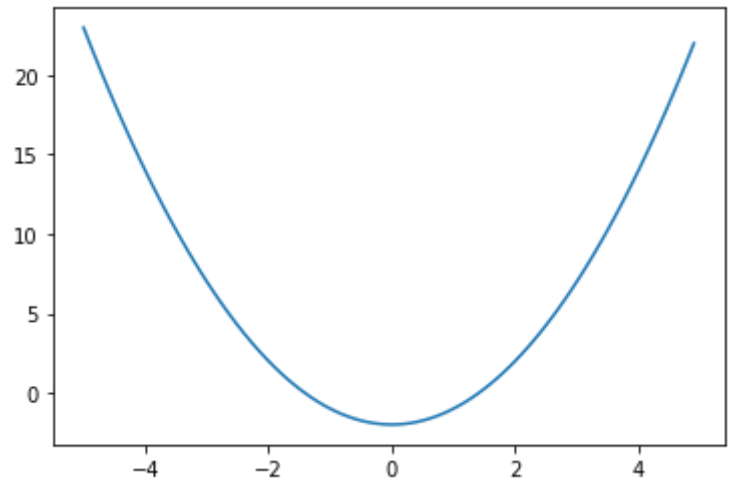
- '%matplotlib inline'이 실행된 경우는 그래프 창의 번호가 명시적으로 나타나지 않지만 '%matplotlib qt'를 실행해 별도의 팝업창에 그래프를 생성하면 제목에 'Figure 번호'가 표시됨.

- figure(n)로 그래프 창을 지정한 후에 '그래프_함수()'를 실행하기 전에 현재 그래프 창의 그래프를 모두 지우려면 clf()를 이용
- 현재의 그래프 창을 닫으려면 close()를 이용

- 두 개의 데이터 쌍 (x, y1), (x, y2)를 지정된 번호의 그래프 창에 그래프로 그리는 예

```
import numpy as np

# 데이터 생성
x = np.arange(-5, 5, 0.1)
y1 = x**2 -2
y2 = 20*np.cos(x)**2 # NumPy에서 cos()는 np.cos()으로 입력

plt.figure(1) # 1번 그래프 창을 생성함
plt.plot(x, y1) # 지정된 그래프 창에 그래프를 그림

plt.figure(2) # 2번 그래프 창을 생성함
plt.plot(x, y2) # 지정된 그래프 창에 그래프를 그림

plt.figure(1) # 이미 생성된 1번 그래프 창을 지정함
plt.plot(x, y2) # 지정된 그래프 창에 그래프를 그림

plt.figure(2) # 이미 생성된 2번 그래프 창을 지정함
plt.clf() # 2번 그래프 창에 그려진 모든 그래프를 지움
plt.plot(x, y1) # 지정된 그래프 창에 그래프를 그림

plt.show()
```




- 하나의 그래프 창에 여러 개의 그래프를 그리는 방법과 새로운 그래프 창을 생성해서 그래프를 그리는 방법

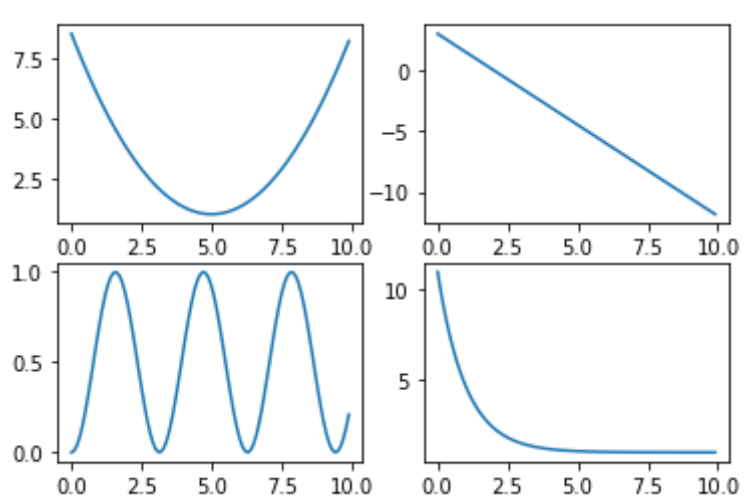
- 하나의 그래프 창을 하위 그래프 영역으로 나눈 후에 각 영 역에 그래프를 그리는 방법
- 하나의 그래프 창을 하위 그래프 영역으로 나누기 위해 서는 subplot()을 이용
- plt.subplot(m, n, p)
- m x n 행렬로 이뤄진 하위 그래프 중에서 p 번 위치에 그래프가 그려지도록 지정
- p는 왼쪽에서 오른쪽으로, 그리고 위에서 아래로 행렬의 위치를 지정

- subplot()을 이용해 하위 그래프의 위치를 지정한 후에 '그래프_함수()'로 지정한 위치에 그래프를 그림

- NumPy를 이용해 4개의 배열 데이터를 생성하고 각 데이터를 4개의 하위 그래프에 나눠서 그림

```
import numpy as np

# 데이터 생성
x = np.arange(0, 10, 0.1)
y1 = 0.3*(x-5)**2 + 1
y2 = -1.5*x + 3
y3 = np.sin(x)**2 # NumPy에서 sin()은 np.sin()으로 입력
```

```
y4 = 10*np.exp(-x) + 1 # NumPy에서 exp()는 np.exp()로 입력

# 2 × 2 행렬로 이뤄진 하위 그래프에서 p에 따라 위치를 지정
plt.subplot(2,2,1) # p는 1
plt.plot(x,y1)

plt.subplot(2,2,2) # p는 2
plt.plot(x,y2)

plt.subplot(2,2,3) # p는 3
plt.plot(x,y3)

plt.subplot(2,2,4) # p는 4
plt.plot(x,y4)

plt.show()
```




## 그래프의 출력 범위 지정하기

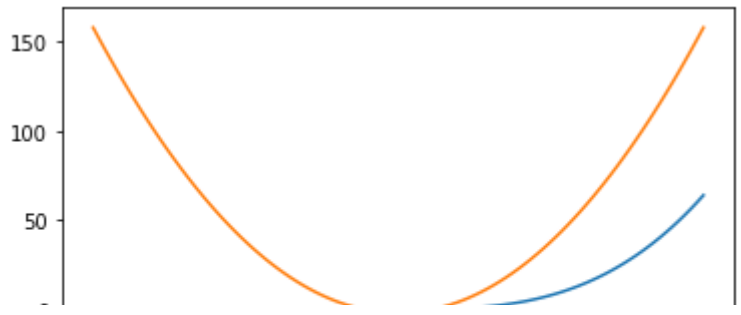
- 그래프로 데이터를 분석할 때 전체 그래프 중 관심 영역만 확대

- matplotlib에서는 출력되는 그래프의 x축과 y축의 좌표 범위를 지정해 전체 그래프 중 관심 영역만 그래프로 그릴 수 있음
- plt.xlim(xmin, xmax) # x축의 좌표 범위 지정(xmin ~ xmax)
- plt.ylim(ymin, ymax) # y축의 좌표 범위 지정(xmin ~ xmax)
- xmin과 xmax는 각각 그래프에서 표시하고자 하는 범위의 X축의 최솟값과 최댓값이고
- ymin과 ymax는 각각 y축의 최솟값과 최댓값
- xlim()과 ylim()은 독립적으로 동작하므로 둘 중 하나만 이용

- 그래프의 x축과 y축의 범위를 가져오려면 다음과 같은 방법을 이용
- [xmin, xmax] = plt.xlim() # x축의 좌표 범위 가져오기
- [ymin, yraax] = plt.ylim() # y축의 좌표 범위 가져오기
- xlim()과 ylim()은 모두 '그래프_함수()'로 그래프를 출력한 후에 실행해야
- 현재 출력한 그래프의 x축과 y축의 범위를 지정하거나 가져올 수 있음
-
-

- 그래프의 출력 범위 지정하는 예

```
import numpy as np

x = np.linspace(-4, 4,100) # [-4, 4] 범위에서 100개의 값 생성
y1 = x**3
y2 = 10*x**2 - 2

plt.plot(x, y1, x, y2)
plt.show()
```




- 다음 코드에서 그래프의 출력 범위를 지정해 관심 영역을 확대

```
plt.plot(x, y1, x, y2)
plt.xlim(-1, 1)
plt.ylim(-3, 3)
plt.show()
```

## 그래프 꾸미기

- plot()에서 사용할 수 있는 fmt 옵션을 활용해 그래프에서 선의 출력 형식을 지정하고 그래프에 축의 라벨, 제목, 격자, 문자 열을 추가하는 방법

### 출력 형식 지정

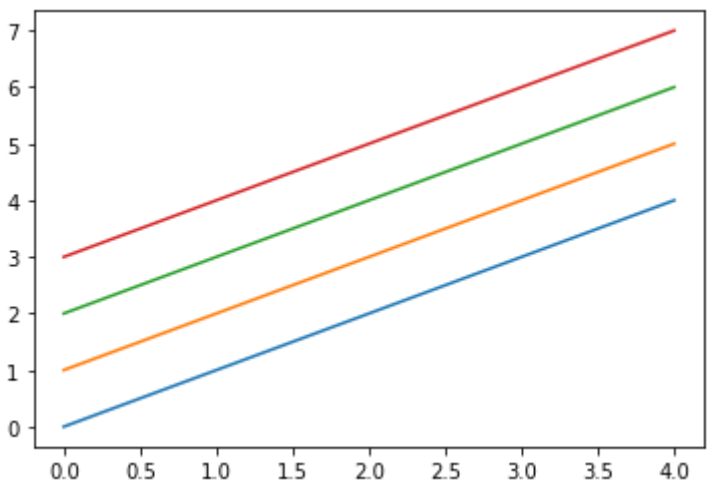
- plotO에서 fmt 옵션을 이용하면 그래프의 컬러, 선의 스타일, 마커를 지정
- fmt 옵션을 지정하는 형식
- fmt = '[color][line_style][marker]'
- color, line_style, marker는 각각 컬러, 선의 스타일, 마커 지정을 위한 약어(문자)
- 지정할 수도 있고 조합해서 지정할 수도 있음

- fmt 옵션의 지정 예를 살펴보기 위해 데이터를 생성

```
import numpy as np
x = np.arange(0, 5, 1)
y1 = x
y2 = x + 1
y3 = x + 2
y4 = x + 3
```

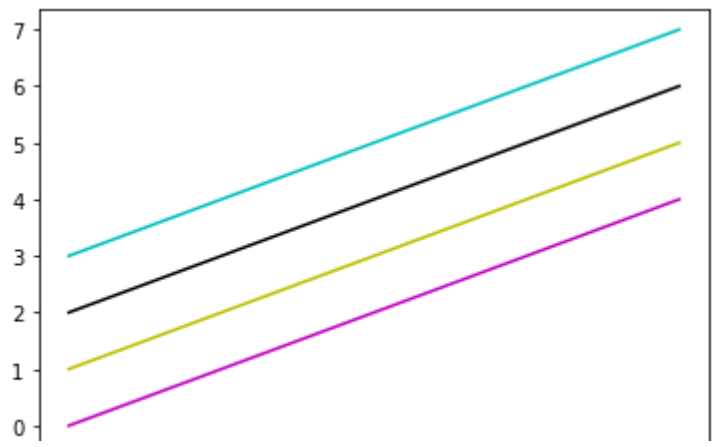
- fmt 옵션을 지정하지 않고 여러 개의 선 그래프를 plot()으로 그림
- fmt 옵션을 지정하지 않아도 데이터에 따라 그래프 선의 칼라가 다르게 그려짐

```
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.show()
```



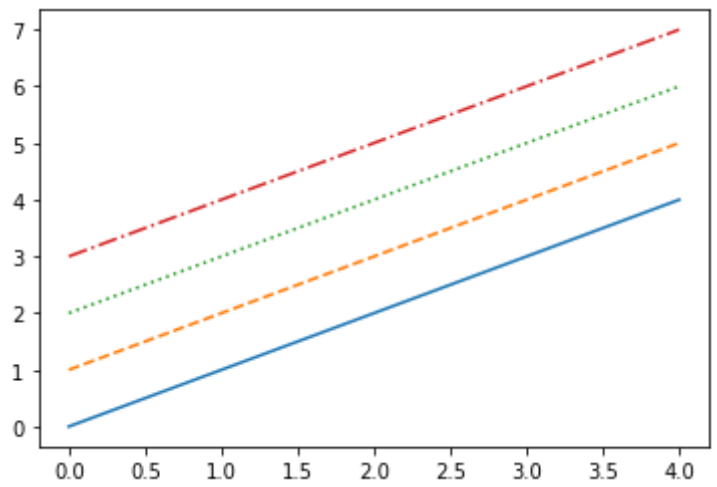

- fmt옵션을 이용해 선 그래프에서 데이터별로 선의 컬러를 지정하는 예

```
plt.plot(x, y1, 'm', x, y2,'y', x, y3, 'k', x, y4, 'c')
plt.show()
```



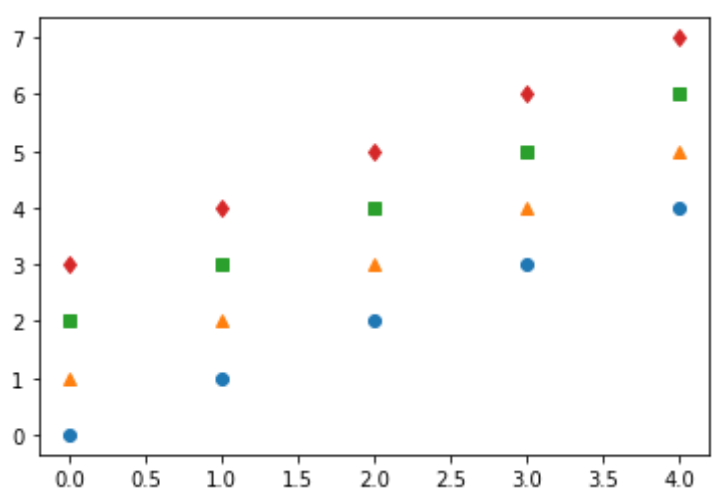
- 선의 스타일을 지정해 그래프를 그림
- 선의 스타일만 지정하면 선 컬러는 자동으로 다르게 지정되어 그러짐

```
plt.plot(x, y1, '-', x, y2, '--', x, y3, ':',  x, y4, '-.')
plt.show()
```



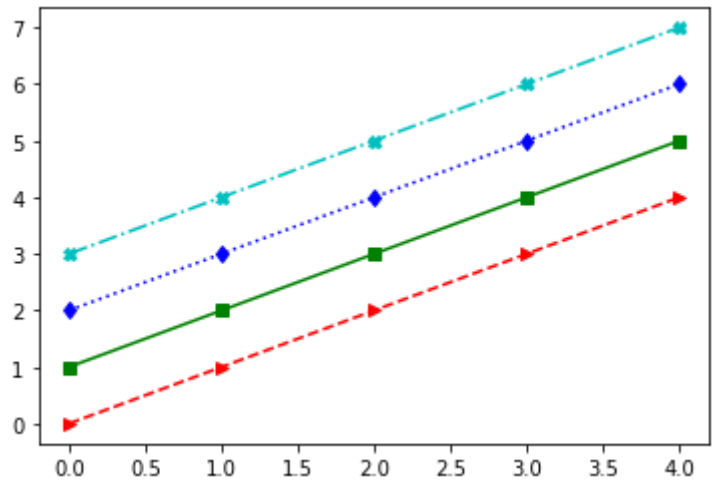
- (x,y) 데이터의 각 점을 선으로 연결하는 것이 아니라, 각 데이터가 표시하는 지점을 다양한 모양의 마커로 표시하는 예

```
plt.plot(x, y1, 'o', x, y2, '^',x, y3, 's', x, y4, 'd')
plt.show()
```



- plot()의 fmt 옵션에서 컬러 , 선의 스타일, 마커는 혼합해서 지정할 수 있음

```
plt.plot(x, y1, '>--r', x, y2, 's-g', x, y3, 'd:b', x, y4, '-.Xc')
plt.show()
```

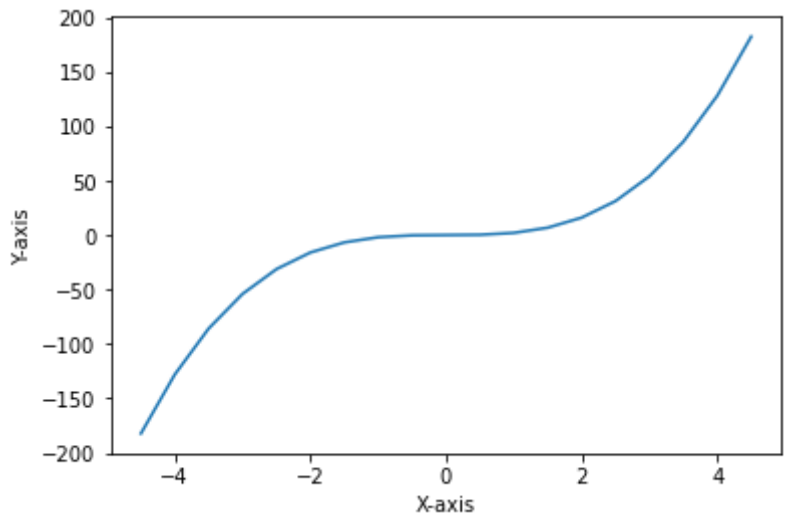


▾ 라벨, 제목, 격자, 범례, 문자열 표시

- 그래프의 x축과y축의 라벨, 그래프 제목, 격자, 범례, 문자열을 표시하는 방법

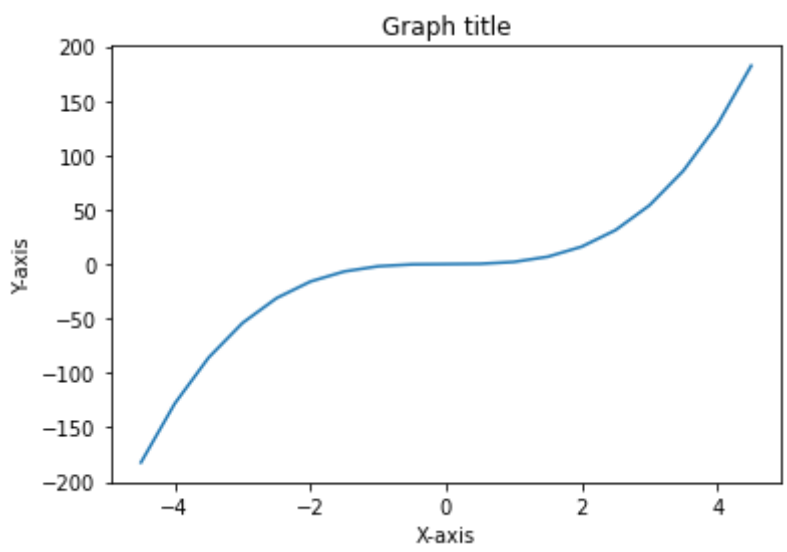
- X축과 y축 라벨은 각각 Xlabel( '문자열')과 ylabel('문자열')로 추가할 수 있음

```
import numpy as np

x = np.arange(-4.5, 5, 0.5)
y = 2*x**3

plt.plot(x,y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```




- 그래프 제목은 title('문자열')로 추가할 수 있음

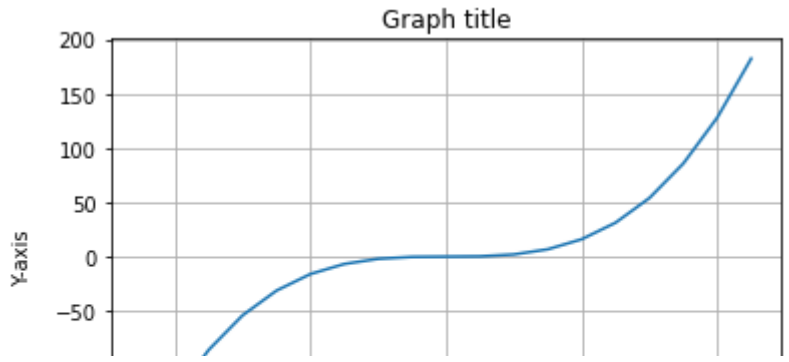
```
plt.plot(x,y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph title')
plt.show()
```




- 그래프에 격자를 추가하려면 grid(True) 혹은 grid()를 이용
- 이미 격자가 있는 그래프에서 격자를 제거하고 싶으면 grid(False)를 이용

- grid()를 수행하면 show()를 수행하지 않아도 Out[]에 그래프 객체의 정보를 출력하지 않고 그래프만 출력

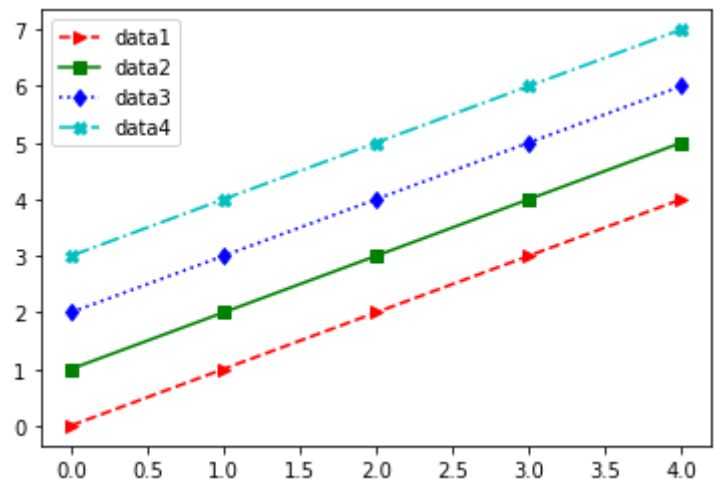
```
plt.plot(x,y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph title')
plt.grid(True) # 'plt.grid()'도 가능
```




- 사용할 수 있는 것이 범례(legend) 를 표시
- matplotlib에서는 여러 개의 선이 있는 그래프에서 legend(['str1'str2', 'str3', • • • l)를 이용해 범례를 표시

−4 −2 0 2 4

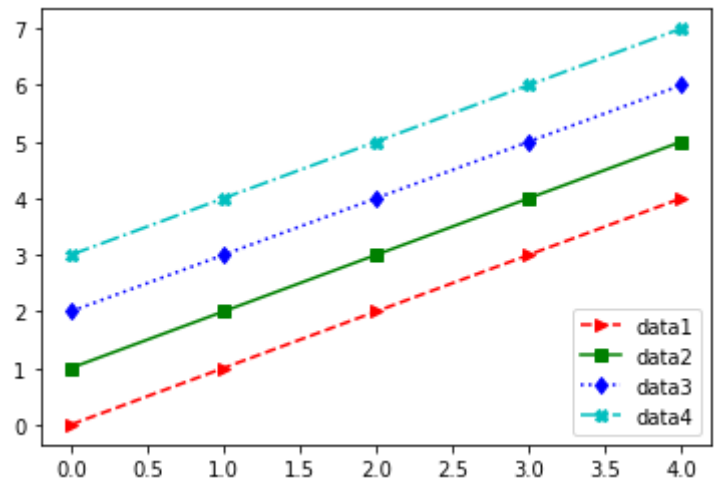
```
import numpy as np
x = np.arange(0, 5, 1)
y1 = x
y2 = x + 1
y3 = x + 2
y4 = x + 3

plt.plot(x, y1, '>--r', x, y2, 's-g', x, y3, 'd:b', x, y4, '-.Xc')
plt.legend(['data1', 'data2', 'data3', 'data4'])
plt.show()
```




- legend()에서 범례의 위치를 지정하지 않으면 자동으로 지정되지만, loc 옵션으로 범례의 위치를 지정할 수도 있음

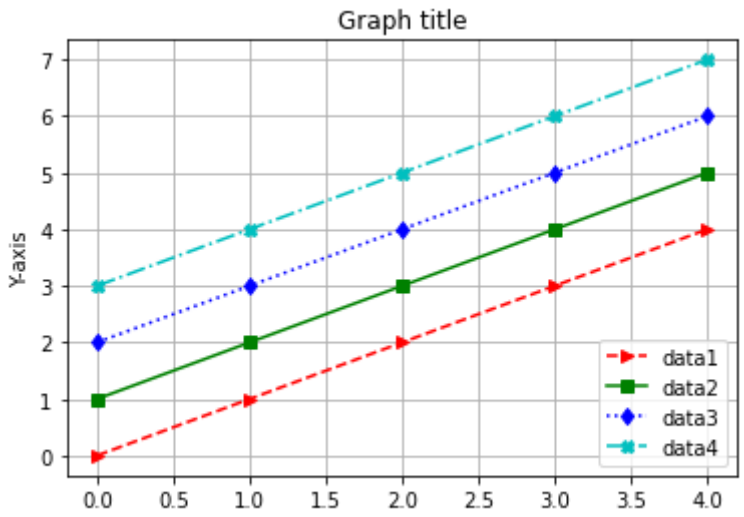
- 범례가 '하단 우측'에 표시되도록 범례의 위치를 지정한 예제

```
plt.plot(x, y1, '>--r', x, y2, 's-g', x, y3, 'd:b', x, y4, '-.Xc')
plt.legend(['data1', 'data2', 'data3', 'data4'], loc = 'lower right')
plt.show()
```




- 그래프에 x축과 y축 라벨 , 그래프 제목, 격자 , 범례를 모두 추가

```
plt.plot(x, y1, '>--r', x, y2, 's-g', x, y3, 'd:b', x, y4, '-.Xc')
plt.legend(['data1', 'data2', 'data3', 'data4'], loc = 4)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph title')
plt.grid(True)
```




- 한글을 표시하고 싶다면 matplotlib에서 사용하는 폰트를 한글 폰트로 지정
- 현재 사용하고 있는 기본 폰트는 다음과 같은 방법으로 알 수 있음
- import matplotlib
- matplotlib.rcParams['font.family'

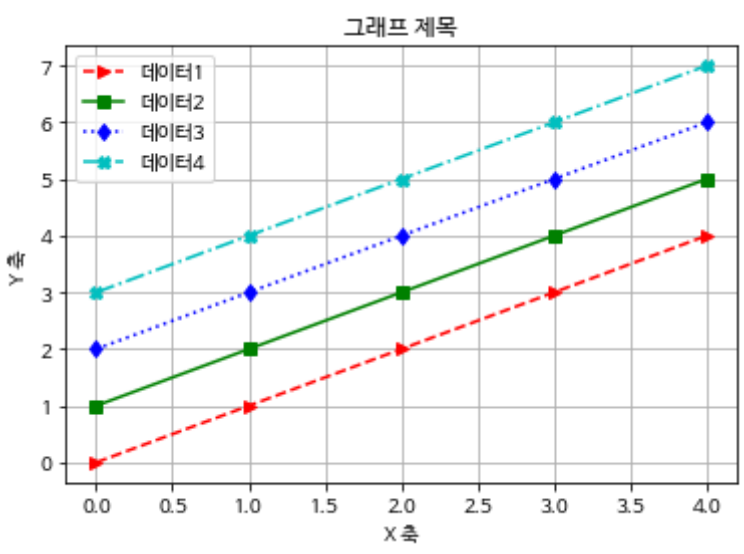
- 폰트를 변경하지 않았다면 기본 폰트는 'sans-serif'
- 폰트를 변경
- matplotlib.rcParams['font.family' ] = '폰트 이름'
- matplotlib.rcParams['axes.unicode_minus'] = False

- matplotlib의 그래프에서 사용할 한글 폰트로 윈도우에 기본으로 설치된 '맑은 고딕'을 지정

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

mpl.rcParams['axes.unicode_minus'] = False

plt.rc('font', family='NanumBarunGothic')
```

- matplotlib에 사용할 폰트를 한글 폰트로 지정한 후에는 x축과 y축 라벨 , 그래프 제목, 범 례를 한글로 입력할수 있음

```
plt.plot(x, y1, '>--r', x, y2, 's-g', x, y3, 'd:b', x, y4, '-.Xc')
plt.legend(['데이터1', '데이터2', '데이터3', '데이터4'], loc = 'best')
plt.xlabel('X 축')
plt.ylabel('Y 축')
plt.title('그래프 제목')
plt.grid(True)
```

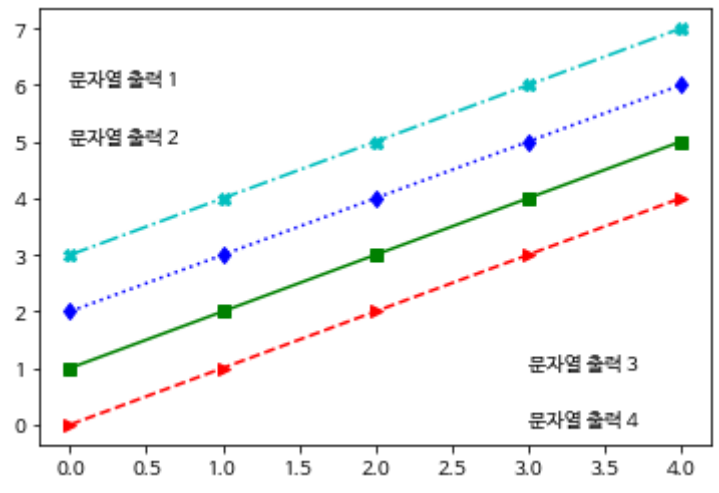



- 그래프 창에 좌표(x, y)를 지정해 문자열(str)을 표시할 수도 있음
- plt.text(x,y,str)

- 그래프 창에 문자열을 표시한 예

```
plt.plot(x, y1, '>--r', x, y2, 's-g', x, y3, 'd:b', x, y4, '-.Xc')
plt.text(0, 6, "문자열 출력 1")
plt.text(0, 5, "문자열 출력 2")
plt.text(3, 1, "문자열 출력 3")
plt.text(3, 0, "문자열 출력 4")
```

```
plt.show()
```

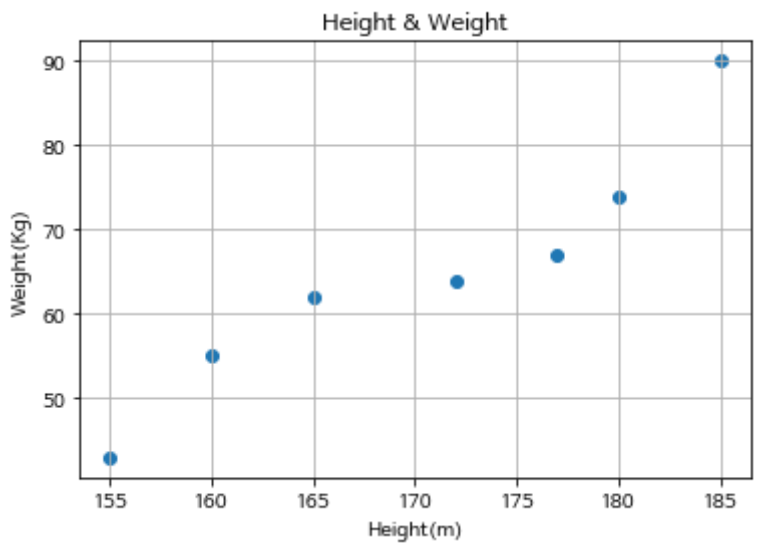



## 산점도

- 산점도(scatter 그래프)는 두 개의 요소로 이뤄진 데이터 집합의 관계(예를 들면, 키와 몸무게와의 관 계, 기온과 아이스크림 판매량과의 관계, 공부 시간과 시험 점수와의 관계)를 시각화하는 데 유용함
- 산점도는 다음과 같은 형식으로 그림
- plt.scatter(x, y [,s=size_n, c=colors, marker='marker_string', alpha=alpha_f])

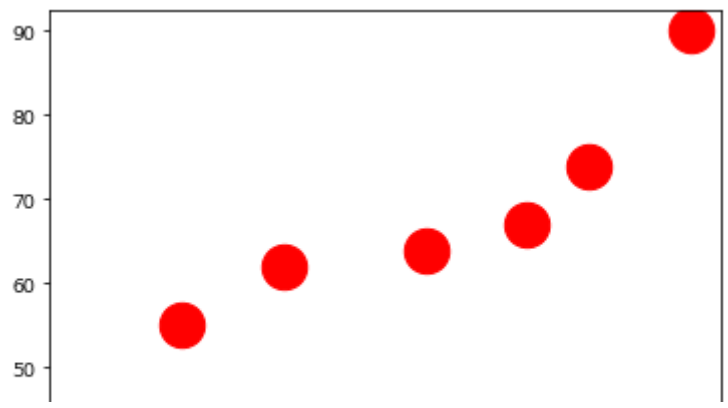
- 키와 몸무게의 데이터 쌍을 생성해 산점도를 그리기
- xlabel(), ylabel(), title(), grid()를 이용

```
import matplotlib.pyplot as plt

height = [165, 177, 160, 180, 185, 155, 172]   # 키 데이터
weight = [62, 67, 55, 74, 90, 43, 64]   # 몸무게 데이터

plt.scatter(height, weight)
plt.xlabel('Height(m)')
plt.ylabel('Weight(Kg)')
plt.title('Height & Weight')
plt.grid(True)
```



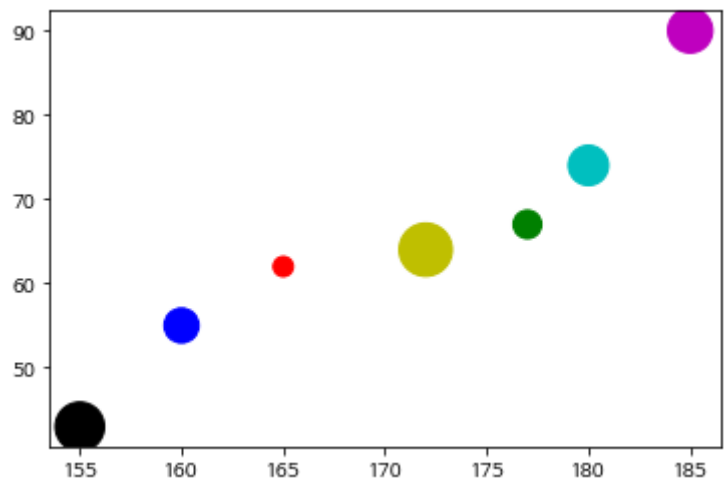

- 산점도를 그릴 때 옵션인 마커 크기(S)와 컬러(C)는 전체적으로 혹은 데이터 쌍마다 지정
- 데이터 전체에 마커 크기와 컬러를 동일하게 지정하는 예

```
plt.scatter(height, weight, s=500, c='r') # 마커 크기는 500, 컬러는 붉은색(red)
plt.show()
```




- 데이터마다 마커의 크기와 컬러를 다르게 지정하는 예제

```
size = 100 * np.arange(1,8) # 데이터별로 마커의 크기 지정
colors = ['r', 'g', 'b', 'c', 'm', 'k', 'y'] # 데이터별로 마커의 컬러 지정

plt.scatter(height, weight, s=size, c=colors)
plt.show()
```

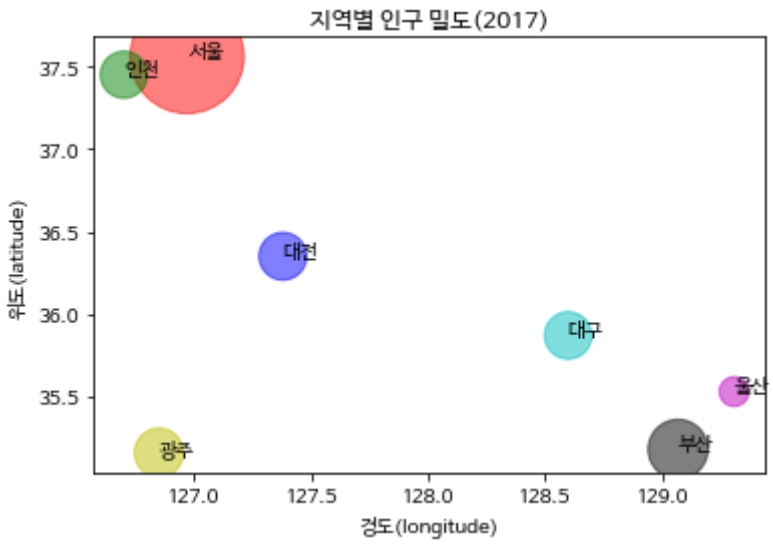



- 데이터마다 마커의 크기와 색깔을 다르게 지정하는 기능을 이용하면
- 지역별 인구 밀 도, 질병 발생률 등을 직관적으로 그래프에 나타낼 수 있음

- 산점도를 이용해 우리나라 주요 도시의 인구 밀도를 시각화한 예

- 주요 도시의 이 름, 위도, 경도, 인구 밀도를 순서에 맞게 리스트 데이터로 입력
- 데이터로 산점도를 그릴 때, 각 도시의 경도와 위도를 (x, y) 좌표로 지정하고
- 도시별로 마커의 컬러를 다르게 지정
- 한 마커의 크기는 인구 밀도에 비례하도록 설정하고 마커의 투명도를 중간으로 설정
- 마커가 위치한 곳에 도시의 이름을 표시

```
import numpy as np

city = ['서울', '인천', '대전', '대구', '울산', '부산', '광주']

# 위도(latitude)와 경도(longitude)
lat  = [37.56, 37.45, 36.35, 35.87, 35.53, 35.18, 35.16]
lon = [126.97, 126.70, 127.38, 128.60, 129.31, 129.07, 126.85]

# 인구 밀도(명/km^2): 2017년 통계청 자료
pop_den = [16154, 2751, 2839, 2790, 1099, 4454, 2995]

size = np.array(pop_den) * 0.2 # 마커의 크기 지정
colors = ['r', 'g', 'b', 'c', 'm', 'k', 'y'] # 마커의 컬러 지정

plt.scatter(lon, lat, s=size, c=colors, alpha=0.5)
plt.xlabel('경도(longitude)')
plt.ylabel('위도(latitude)')
plt.title('지역별 인구 밀도(2017)')

for x, y, name in zip(lon, lat, city):
    plt.text(x, y, name) # 위도 경도에 맞게 도시 이름 출력

plt.show()
```




## 막대 그래프

- 막대 그래프(bar 그래프)는 값을 막대의 높이로 나타내므로 여러 항목의 수량이 많고 적음을 한눈에 알아 볼수 있음
- 여러 항목의 데이터를 서로 비교할 때 주로 이용

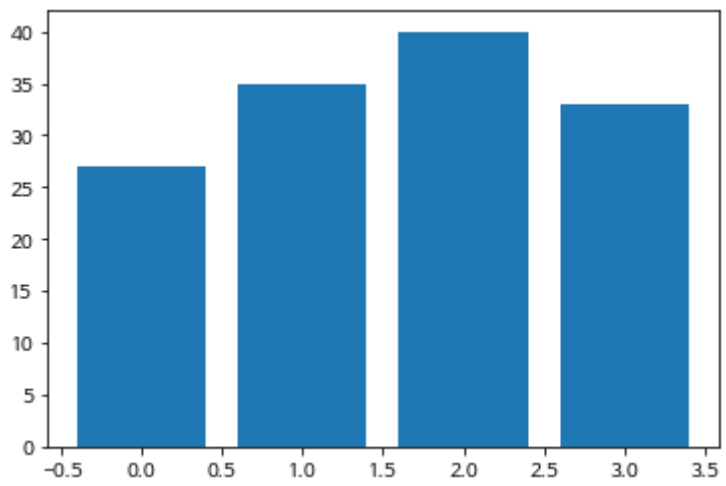
- 막대 그래프는 다음과 같은 형식으로 그림
- plt.bar(x, height [,width=width_f, color=colors, tick_label=tick_labels, align='center'(기본) 혹은 'edge', label=labels])
- height는 시각화하고자 하는 막대 그래프의 데이터
- x는 height와 길이가 일치하는 데이 터로 x축에 표시될 위치를 지정
- 순서만 지정하므로 0부터 시작해서 height의 길이만 큼 1씩 증가하는 값을 갖음
- width 옵션으로 [0, 1 ] 사이의 실수를 지정 해 막대의 폭을 조절할 수 있음
- width 옵션을 입력하지 않으면 기본값인 0.8이 입력
- color 옵션으로는 fmt 옵션의 컬러 지정 약어를 이용해 막대 그래프의 색을 지정할 수 있음
- tick_label 옵션에 문자열 혹은 문자열 리스트를 입력해 막대 그래프 각각의 이름을 지정
- tick_label 옵션을 지정하지 않으면 기본적으로 숫자로 라벨이 지정
- align은 막대 그래프의 위치를 가운데로 할지 (center) 한쪽으로 치우치게 할지 (edge)를 설정
- align 옵션의 기본은 center
- label에는 범례에 사용될 문자열을 지정할 수 있음

- 데이터로는 헬스클럽 회원 네 명이 운동을 시작하기 전에 측정한 윗몸 일으키기 횟수와 한 달 후에 다시 측정한 윗몸 일으키기 횟수

```
member_IDs = ['m_01', 'm_02', 'm_03', 'm_04'] # 회원 ID
before_ex = [27, 35, 40, 33] # 운동 시작 전
after_ex = [30, 38, 42, 37] # 운동 한 달 후
```

- 이 데이터를 이용해 막대 그래프를 그림

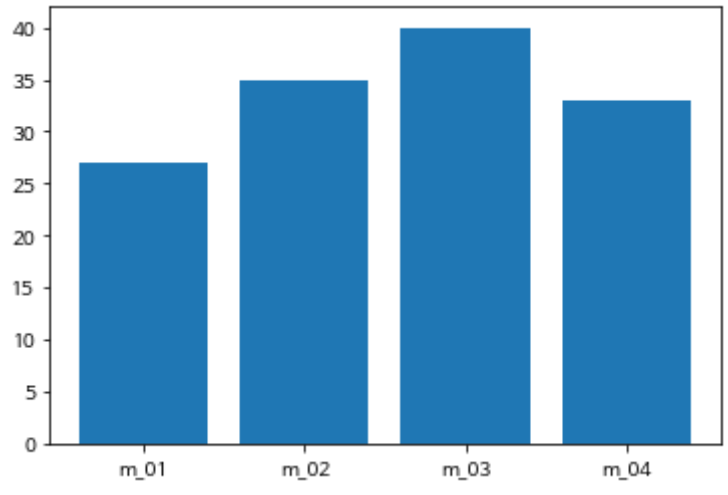
```
import matplotlib.pyplot as plt
import numpy as np

n_data = len(member_IDs)     # 회원이 네 명이므로 전체 데이터 수는 4
index = np.arange(n_data)    # NumPy를 이용해 배열 생성 (0, 1, 2, 3)
plt.bar(index, before_ex)    # bar(x,y)에서 x=index, height=before_ex 로 지정
plt.show()
```




- 생성된 막대 그래프의 네 개의 막대는 {0, 1, 2, 3}을 중심으로 0.8의 두께를 갖음

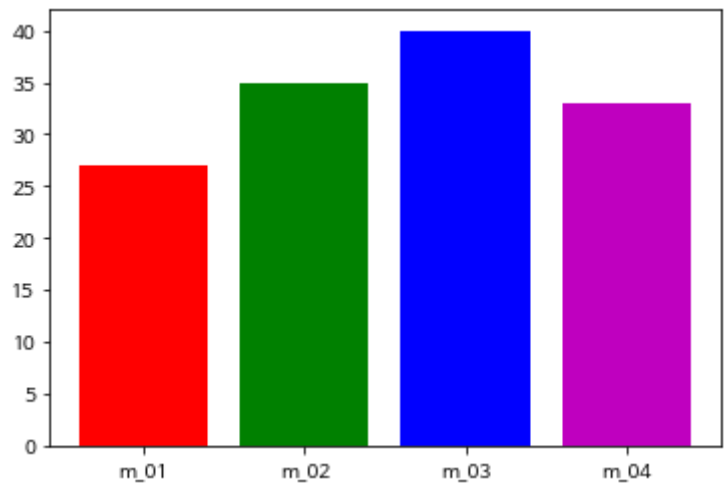
- tick_label 옵션을 이용하면 x축의 tick 라벨을 원하는 데이터로 지정할 수 있음
- x축의 tick 라벨을 앞에서 생성한 리스트 변수(member_IDs)로 지정

```
plt.bar(index, before_ex, tick_label = member_IDs)
plt.show()
```



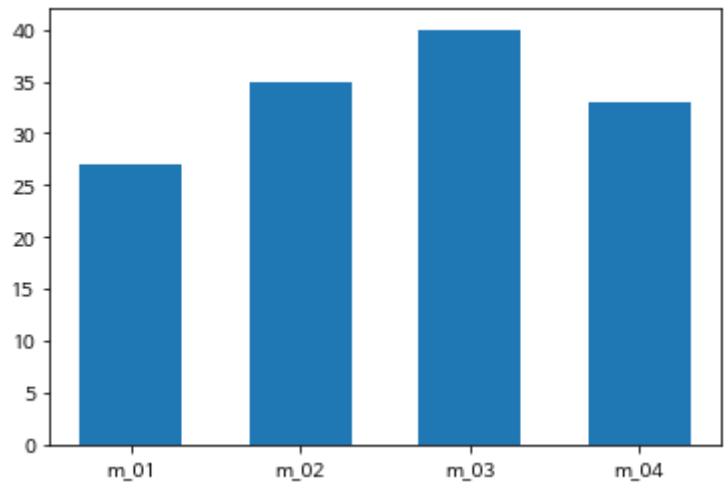

- X축이 숫자 대신 각각 막대 그래프가 의미하는 라벨로 변경
- 막대 그래프의 색깔 도 변경하고 싶다면 막대 그래프별로 컬러를 지정

```
colors=['r', 'g', 'b', 'm']
plt.bar(index, before_ex, color = colors, tick_label = member_IDs)
plt.show()
```



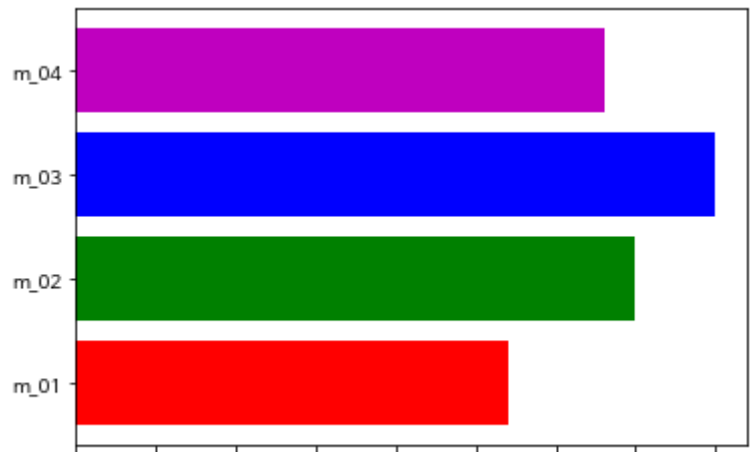

- 막대 그래프의 폭을 변경하려면 width에 값을 지정

```
plt.bar(index, before_ex, tick_label = member_IDs, width = 0.6)
plt.show()
```



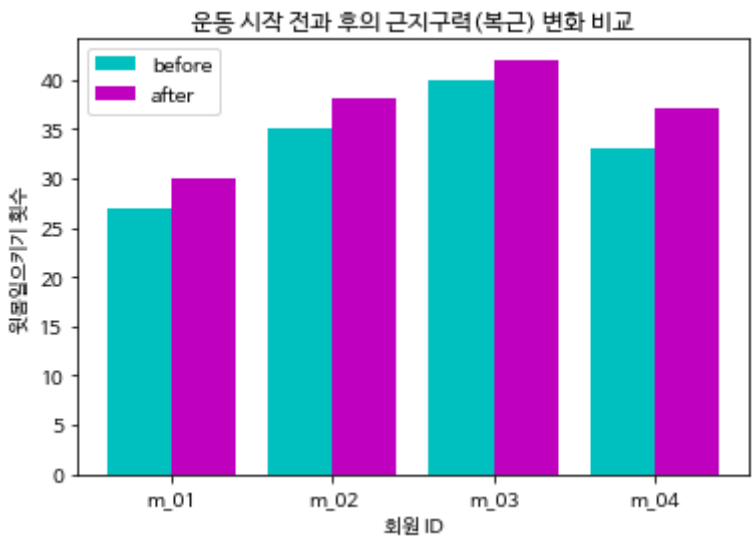

- 가로 막대 그래프를 그리려면 barh()를 이용
- barh()는 bar() 그래프와 사용법이 같음. 단, barh()에서는 width 옵션을 이용할 수 없음

```
colors=['r', 'g', 'b', 'm']
plt.barh(index, before_ex, color = colors, tick_label = member_IDs)
plt.show()
```




- 운동 시작 전(before_ex)과 한 달 후(after_ex)의 데이터를 하나의 그래프 창에 막대 그래프로 그려서 비교

```
barWidth = 0.4
plt.bar(index, before_ex, color='c', align='edge', width = barWidth, label='before')
plt.bar(index + barWidth, after_ex , color='m', align='edge', width = barWidth, label='after')

plt.xticks(index + barWidth, member_IDs)
plt.legend()
plt.xlabel('회원 ID')
plt.ylabel('윗몸일으키기 횟수')
plt.title('운동 시작 전과 후의 근지구력(복근) 변화 비교')
plt.show()
```




## 히스토그램

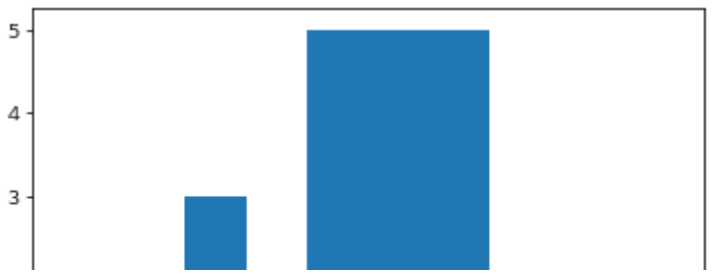
- 히스토그램(histogram)은 데이터를 정해진 간격으로 나눈 후 그 간격 안에 들어간 데이터 개수를 막대로 표시한 그래프
- 데이터가 어떤 분포를 갖는지를 볼 때 주로 이용
- 히스토그램은 도수 분 포표를 막대 그래프로 시각화한 것
- 히스토그램은 주로 통계 분야에서 데이터가 어떻게 분포하 는지 알아볼 때 많이 이용

- 히스토그램은 다음과 같은 형식을 이용해 그림
- plt.hist(x, [,bins = bins_n 혹은 'auto'])
- x는 변량 데이터
- 옵션 bins는 계급의 개수로 이 개수만큼 자동으로 계급이 생성되어 히 스토그램을 그림
- 옵션 bins를 입력하지 않으면 기본적으로 bins는 10
- bin =’auto’가 입력되면, x에 맞게 자동으로 bins에 값이 들어감

- 수학 점수를 다음과 같이 변량 데이터로 입력하고 히스토그램을 그림

```
import matplotlib.pyplot as plt

math = [76, 82, 84, 83, 90, 86, 85, 92, 72, 71, 100, 87, 81, 76, 94, 78, 81, 60, 79, 69, 74, 87, 82, 68, 79]
plt.hist(math)
```

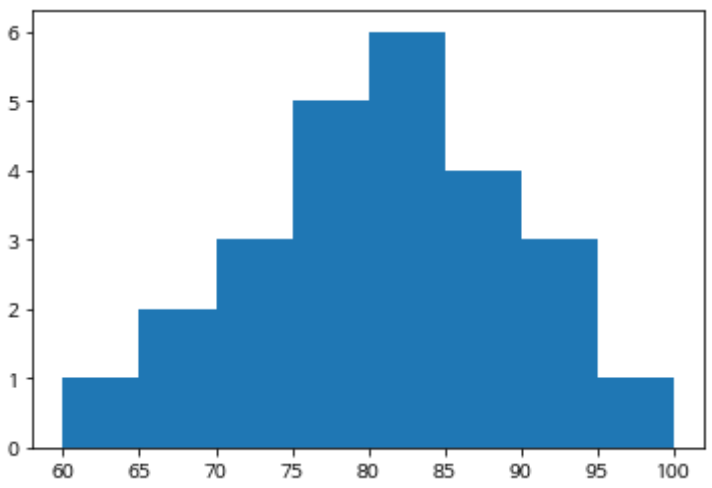
```
(array([1., 0., 3., 2., 5., 5., 5., 1., 2., 1.]),
 array([ 60.,  64.,  68.,  72.,  76.,  80.,  84.,  88.,  92.,  96., 100.]),
 <a list of 10 Patch objects>)
```




- 출력 결과에서 볼 수 있듯이 hist()는 기본적으로 변량을 10개의 계급으로 나눠서 표시

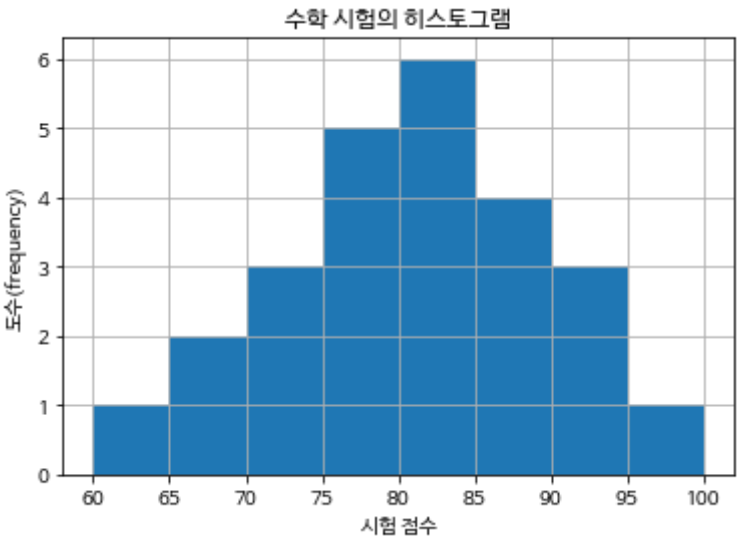

도수 분포표를 만들 때 60에서 100까지 5 간격으로 8개의 계급으로 나눴으니 bins 옵션에 8을 입력해서 히스토그램을 그림

```
plt.hist(math, bins= 8)
plt.show()
```




- hist()를 이용한 히스토그램에서도 x 라벨, y 라벨, 그래프 제목 , 격자를 추가

```
plt.hist(math, bins= 8)
plt.xlabel('시험 점수')
plt.ylabel('도수(frequency)')
plt.title('수학 시험의 히스토그램')
plt.grid()
plt.show()
```




## 파이 그래프

- 파이(pie) 그래프는 원 안에 데이터의 각 항목이 차지하는 비율만큼 부채꼴의 크기를 갖는 영역으로 이뤄진 그래프
- 각 부채꼴 부분이 파이 조각처럼 생겨서 파이 그래프
- 파이 그래프에서 부채꼴 부분의 크기는 각 항목 크기에 비례
- 파이 그래프는 전체 데이터에서 각 항목이 차지한 비율을 비교할 때 많이 이용

- 파이 그래프는 다음과 같은 형식으로 그림
- plt.pie(x, [,labels = label_sep, autopct='비율 표시 형식(ex: %0.1f)', shadow = False(기본)
- 혹은 True, explode = explode_seq, counterclock = True(기본) 혹은 False, startangle = 각도 (기본은 0) ])
- x는 배열 혹은 시퀀스 형태의 데이터
- Pie()는 x를 입력하면 x의 각 요소가 전체에서 차지하는 비율을 계산하고

- 그 비율에 맞게 부채꼴 부분의 크기를 결정해서 파이 그래프를 그림

- 다른 그래프와 달리 파이 그래프는 가로와 세로 비율이 1 대 1로 같아야 그래프가 제대로 보임
- 파이 그래프를 그리기 전에 다음처럼 그래프 크기(너비와 높이)를 지정해서 비율을 조절할 필요가 있음
- plt.figure(figsize = (w,h))
- w와 h는 그래프의 너비(width)와 높이(height)를 의미하며 단위는 인치(inch)임
- figure(figsize = (*, h))를 이용해 w와 h 값을 지정하지 않으면 (w, h)의 기본값은 (6, 4)가 됨

- 어떤 학급에서 20명의 학생에게 5개의 과일 (사과，바나나, 딸기, 오렌지，포도) 중
- 제일 좋아하는 과일을 선택하라고 했을 때 각각 (7, 6, 3, 2, 2) 의 결과를 얻었다고 가정
- 과일 이름과 이를 선택한 학생 수를 할당한 데이터를 다음과 같이 생성

```
fruit = ['사과', '바나나', '딸기', '오렌지', '포도']
result = [7, 6, 3, 2, 2]
```

- 생성한 데이터를 이용해 파이 그래프

```
import matplotlib.pyplot as plt

plt.pie(result)
plt.show()
```

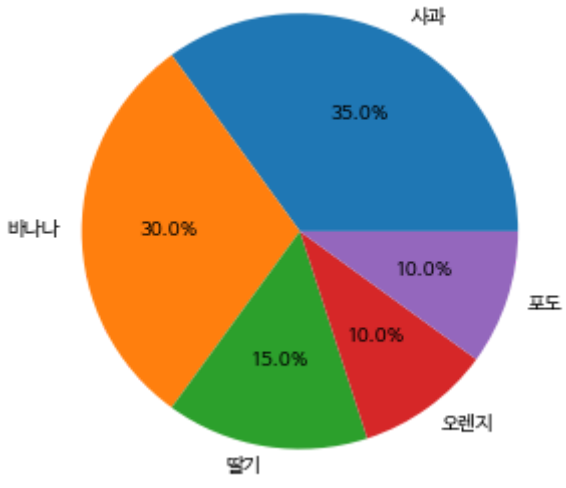
- 파이 그래프는 원이 아니라 타원처럼 보임
- 그래프의 너비와 높이의 비율이 1 대 1이 아니기 때문

- 비율을 맞추기 위해 파이 그래프를 생성 전에 그래프의 너비와 높이가 같아지도록 지정

```
plt.figure(figsize=(5,5))
plt.pie(result)
plt.show()
```

- 각 부채꼴 부분에 속하는 데이터의 라벨과 비율을 추가

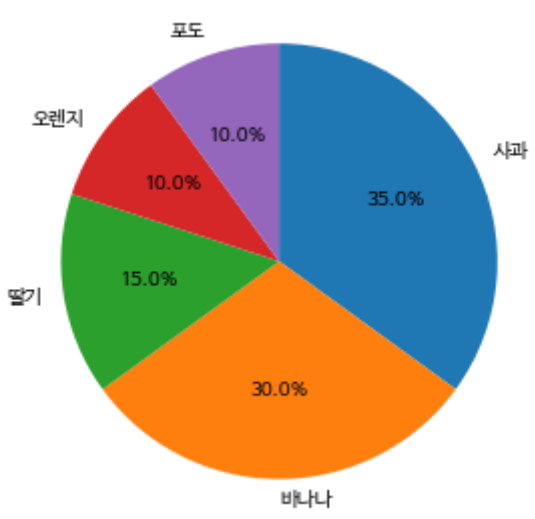
```
plt.figure(figsize=(5,5))
plt.pie(result, labels= fruit, autopct='%.1f%%')
plt.show()
```




- 파이 그래프에서 각 부채 꼴 부분은 X축 기준 각도 0도를 시작으로 반시계방향으로 각 부채꼴 부분이 그려짐

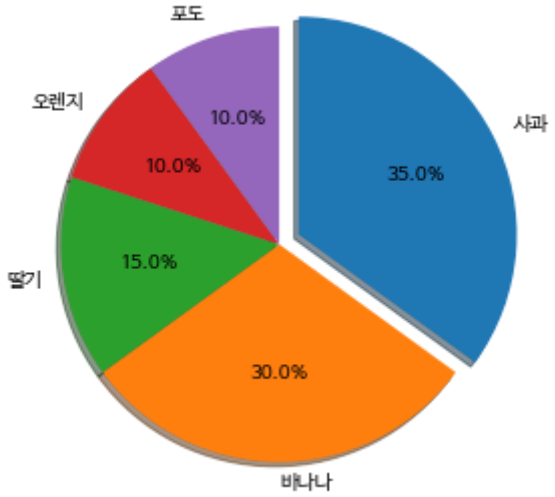
- X축 기준 각도 90도에서 시작해서 시계방향으로 설정해서 각 부채꼴 부분을 표시하려면 파이 그래 프의 옵션을 지정

```
plt.figure(figsize=(5,5))
plt.pie(result, labels= fruit, autopct='%.1f%%', startangle=90, counterclock = False)
plt.show()
```




- 그림자 를 추가하고 특정 요소(여기서는 사과)를 표시한 부채꼴 부분만 강조하려면 옵션을 추가

```
explode_value = (0.1, 0, 0, 0, 0)

plt.figure(figsize=(5,5))
plt.pie(result, labels= fruit, autopct='%.1f%%', startangle=90, counterclock = False, explode=explode_value, shadc
plt.show()
```




## 그래프 저장하기

- matplotlib를 이용해 생성한 그래프는 화면으로 출력할 수 있을 뿐만 아니라 이미지 파일로 저장할 수도 있음
- 그래프를 파일로 저장하려면 다음 방법을 이용
- plt.savefig(file_name, [,dpi = dpi_n(기본은 72)])
- file_name은 저장하고자 하는 이미지 파일 이름
- 옵션 dpi에는 숫자가 들어감

- 옵션 dpi에 대입되는 숫자가 클수록 해상도 가 높아져서 세밀한 그림이 그려지지만,
- 파일의 크기도 커지므로 적당한 숫자를 설정해야 함
- dpi를 지정하지 않으면 기본적으로 72가 지정

- matplotlib에서 현재 설정된 그래프의 크기와 dpi 값이 무엇인지 알 수 있음
- 그래프의 크기는 figure(figsize = (*,h))를 이용해 변경할 수 있음

```
import matplotlib as mpl
mpl.rcParams['figure.figsize']
```

```
[6.0, 4.0]
```

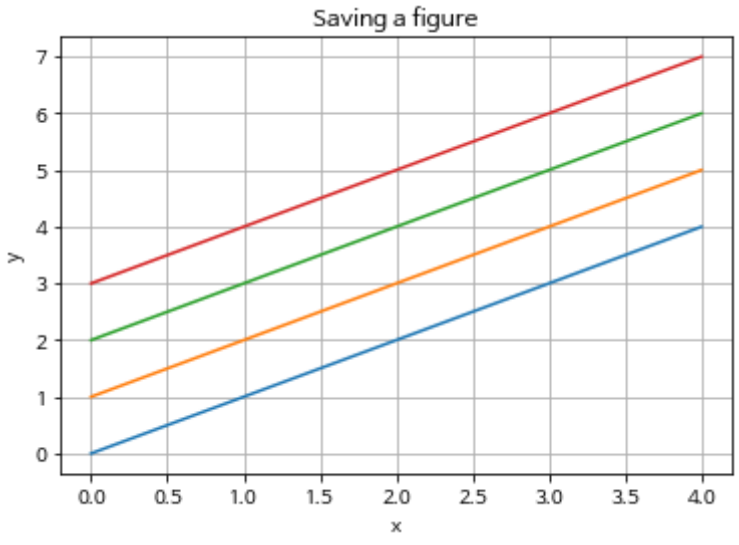
- 현재 설정된 dpi 값을 확인하는 방법

```
mpl.rcParams['figure.dpi']
```

```
72.0
```

- savefig()에서 이미지 파일 이름을 지정 했고, dpi 값은 100으로 지정

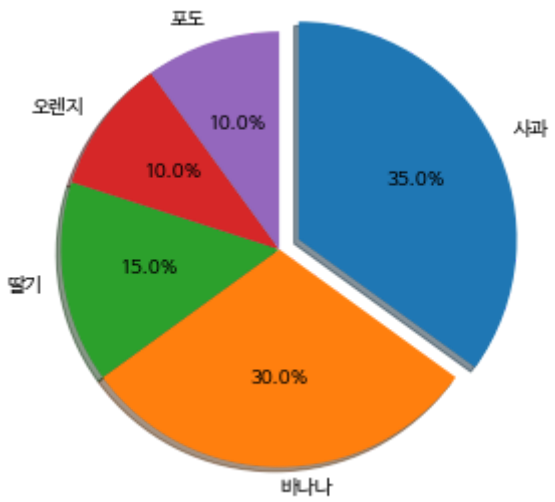
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 5, 1)
y1 = x
y2 = x + 1
y3 = x + 2
y4 = x + 3

plt.plot(x, y1, x, y2, x, y3, x, y4)

plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Saving a figure')

# 그래프를 이미지 파일로 저장. dpi는 100으로 설정
plt.savefig('saveFigTest1.png', dpi = 100)
plt.show()
```




- plt.figure(figsize=(w,h))로 생성되는 그래프의 크기를 지정해 이미지 파일로 저장하는 예

```
import matplotlib.pyplot as plt

fruit = ['사과', '바나나', '딸기', '오렌지', '포도']
result = [7, 6, 3, 2, 2]
explode_value = (0.1, 0, 0, 0, 0)

plt.figure(figsize=(5,5)) # 그래프의 크기를 지정
plt.pie(result, labels= fruit, autopct='%.1f%%', startangle=90, counterclock = False, explode=explode_value, shado

# 그래프를 이미지 파일로 저장. dpi는 200으로 설정
```

```
plt.savefig('saveFigTest2.png', dpi = 200)
plt.show()
```




## 12.2 pandas로 그래프 그리기

- pandas로 생성한 데이터와 pandas의 그래프 그리기 기능을 이용해
- 선 그래프, 산점도, 막대 그래프 , 히스토그램 , 파이 그래프를 그리는 방법

## pandas의 그래프 구조

- pandas의 Series나 DateFrame으로 생성 한 데이터가 있을 때 다음과 같은 형식으로 그래프
- Series_data.plot([kind='graph_kind' ][,option])
- DataFrame_data.plot([x=label 혹은 position, y=label 혹은 position,] [kind='graph_kind'][,option]
- plot()에 kind 옵션을 선택해 그래프의 종류를 선택할 수도 있지만 plot.graph_kind()처럼 사용할 수도 있음

## pandas의 선 그래프

- pandas로 생성한 데이터를 plot()을 이용해 그래프로 그림
- pandas의 Series 데이터를 생성

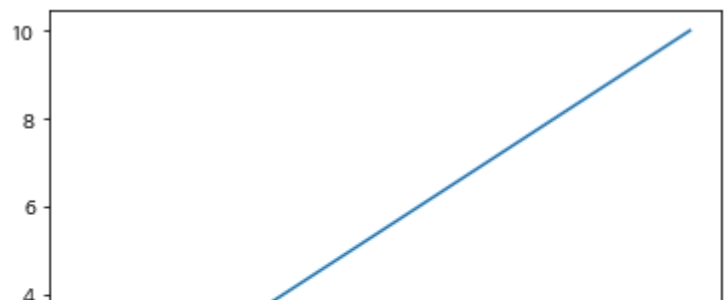
```
import pandas as pd
import matplotlib.pyplot as plt

s1 = pd.Series([1,2,3,4,5,6,7,8,9,10])
s1
```

```
0     1
1     2
2     3
3     4
4     5
5     6
6     7
7     8
8     9
9    10
dtype: int64
```

- 첫 번째 줄은 index이고 두 번째 줄은 values

- 데이터로 선 그래프

```
s1.plot()
plt.show()
```




- 인자 없이 s1.plot()'을 수행하면 이의 index와 values가 각각 x축과 y축 좌푯 값으로 입력되어 그래프를 그림

- pandas의 Series 데이터의 index 값을 변경하면 plot()을 수행할 때 x 좌푯값이 변경된 index 값으로 그려짐

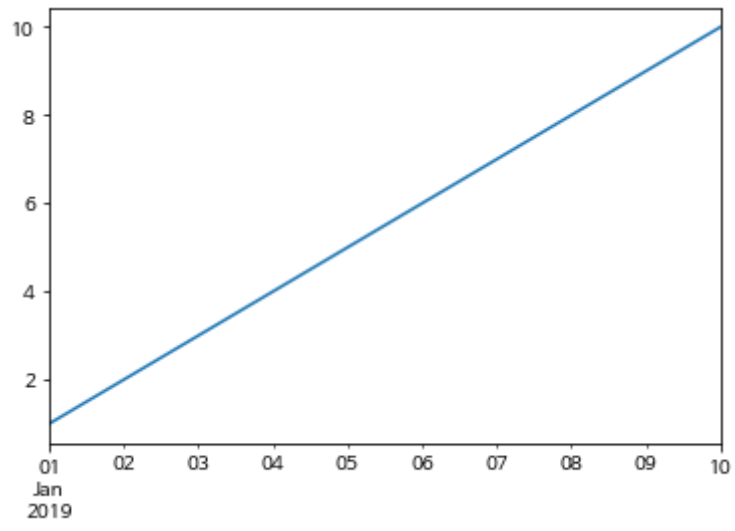
- index를 날짜로 지정한 Series 데이터

```
s2 = pd.Series([1,2,3,4,5,6,7,8,9,10], index = pd.date_range('2019-01-01', periods=10))
s2
```

```
2019-01-01     1
2019-01-02     2
2019-01-03     3
2019-01-04     4
2019-01-05     5
2019-01-06     6
2019-01-07     7
2019-01-08     8
2019-01-09     9
2019-01-10    10
Freq: D, dtype: int64
```

- pandas의 Series 데이터 s2를 plot()을 이용해 그래프로 그림

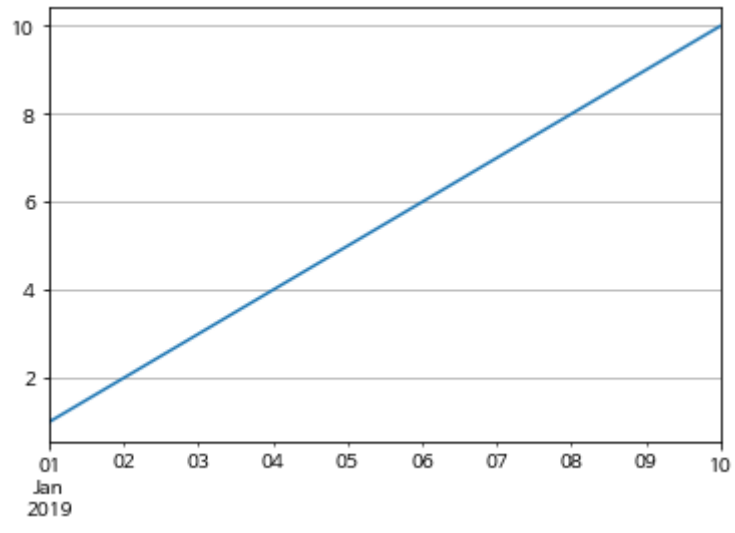
```
s2.plot()
plt.show()
```




- index가 날짜로 지정됐으므로 x축 값이 날짜로 변경

- 격자를 추가하려 면 plot()의 인자로 grid=True를 입력

```
s2.plot(grid=True)
plt.show()
```



- pandas의 DataFrame 데이터를 이용해 그래프를 그리는 방법

- pandas의 read_csv()를 이용해 앞 장에서 만든 csv 데이터 파일을 읽어와서 DataFrame 데이터를 생성

```
df_rain = pd.read_csv('sea_rain1.csv', index_col="연도" )
df_rain
```

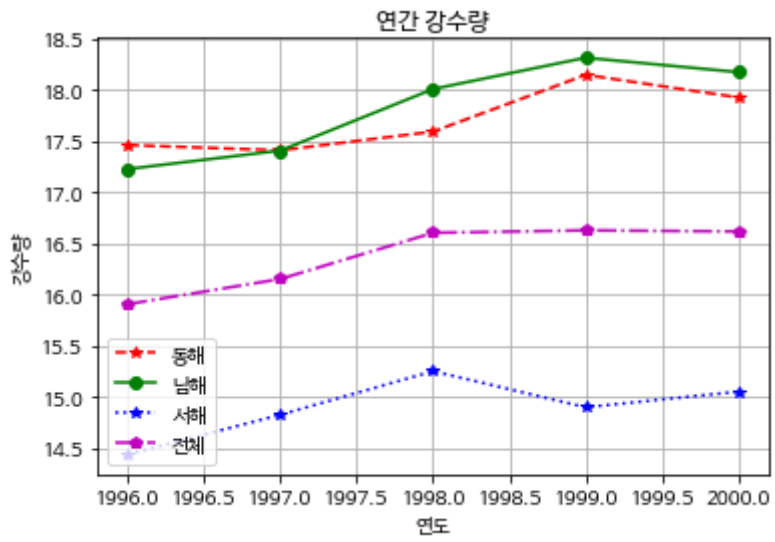
| 연도 | 동해 | 남해 | 서해 | 전체 |
|---|---|---|---|---|
| **1996** | 17.4629 | 17.2288 | 14.4360 | 15.9067 |
| **1997** | 17.4116 | 17.4092 | 14.8248 | 16.1526 |
| **1998** | 17.5944 | 18.0110 | 15.2512 | 16.6044 |
| **1999** | 18.1495 | 18.3175 | 14.8979 | 16.6284 |
| **2000** | 17.9288 | 18.1766 | 15.0504 | 16.6178 |

- pandas의 DataFrame 데이터에 대해 인자 없이 plot() 실행하면 index는 그래프의 x축 데이터가 되고
- 모든 열 데이터(values)는 각각 그래프의 y축 데이터가 됨
- columns의 경우는 범례로 표시

- plot()의 옵션으로 'grid = True'를 지정해 격자를 추가하고
- style에는 데이터마다 선의 컬러, 선의 모양 , 마커를 지정하는 예

- pandas 데이터로 그래프를 그릴 때 x축과 y축 라벨과 제목을 지정하고 싶다면
- plot()의 반환 값에 set_xlabel(), set_ylabel(), set_title() 메서드 를 이용해
- 각각 x축 라벨, y축 라벨, 제목을 추가할 수 있음

```
rain_plot = df_rain.plot(grid = True, style = ['r--*', 'g-o', 'b:*', 'm-.p'])
rain_plot.set_xlabel("연도")
rain_plot.set_ylabel("강수량")
rain_plot.set_title("연간 강수량")
plt.show()
```



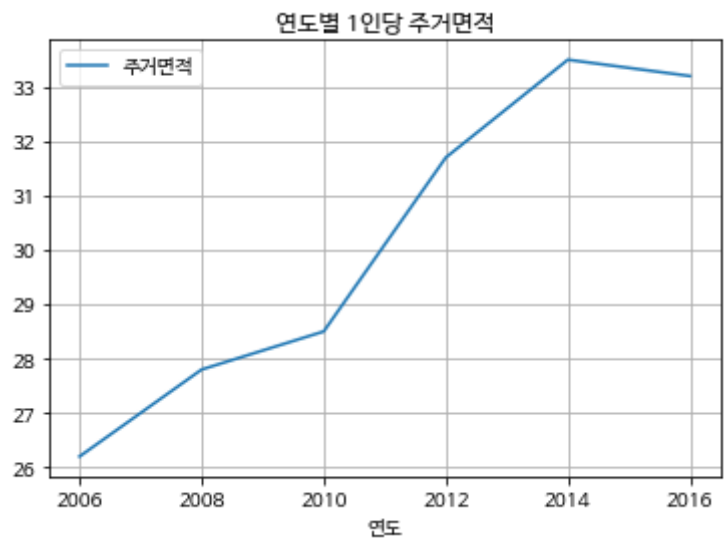

- 연도별 1인당 주거면적 데이터(전국)를 이용해 DataFrame 데이터를 생성

```
year = [2006, 2008, 2010, 2012, 2014, 2016] # 연도
area = [26.2, 27.8, 28.5, 31.7, 33.5, 33.2] # 1인당 주거면적
table = {'연도':year, '주거면적':area}
df_area = pd.DataFrame(table, columns=['연도', '주거면적'])
df_area
```

| | 연도 | 주거면적 |
|---|---|---|
| **0** | 2006 | 26.2 |

- pandas의 DataFrame 변수 df_area에 plot()을 적용해 그래프를 그림
- plot()의 인자 x와 y에 각각 열 이름을 지정하면
- x에 지정한 열 데이터는 x축 데이터로 , y에 지정한 열 데이터는 y축 데이터로 지정돼 선 그래프를 그림
- 'grid = True'를 입력하면 격자를 넣을 수도 있음
- 'title = 문자열'옵션을 추가하면 그래프의 제목을 추가

```
df_area.plot(x='연도', y='주거면적', grid = True, title = '연도별 1인당 주거면적')
plt.show()
```




## pandas의 산점도

- 일정 기간(10일) 동안 기록한 일일 최고 기온과 아이스크림 판매량 데이터를 이용해 pandas의 DataFrame 데이터를 생성

```
import matplotlib.pyplot as plt
import pandas as pd

temperature = [25.2, 27.4, 22.9, 26.2, 29.5, 33.1, 30.4, 36.1, 34.4, 29.1]
Ice_cream_sales = [236500, 357500, 203500, 365200, 446600, 574200, 453200, 675400, 598400, 463100]

dict_data = {'기온':temperature, '아이스크림 판매량':Ice_cream_sales}
df_ice_cream = pd.DataFrame(dict_data, columns=['기온', '아이스크림 판매량'])

df_ice_cream
```

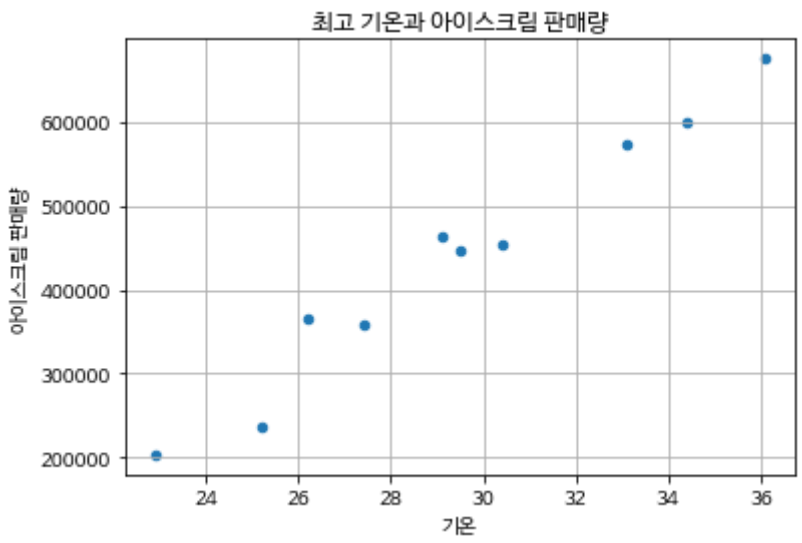
| | 기온 | 아이스크림 판매량 |
|---|---|---|
| **0** | 25.2 | 236500 |
| **1** | 27.4 | 357500 |
| **2** | 22.9 | 203500 |
| **3** | 26.2 | 365200 |
| **4** | 29.5 | 446600 |
| **5** | 33.1 | 574200 |
| **6** | 30.4 | 453200 |
| **7** | 36.1 | 675400 |
| **8** | 34.4 | 598400 |
| **9** | 29.1 | 463100 |

- pandas에 서 산점도 그래프를 그리려면 DataFrame_data.plot(kind='scatter') 혹은 DataFrame_data.plot.scatter() 를 이용

- 인자 때 y에 각각 열 이름을 지정하면 x에 지정한 열 데이터는 x축 데이터로, y에 지 정한 열 데이터는 y축 데이터로 지정돼 산점도 그래프를 그림

```
df_ice_cream.plot.scatter(x='기온', y='아이스크림 판매량', grid=True, title='최고 기온과 아이스크림 판매량')
```

```
plt.show()
```




## pandas의 막대 그래프

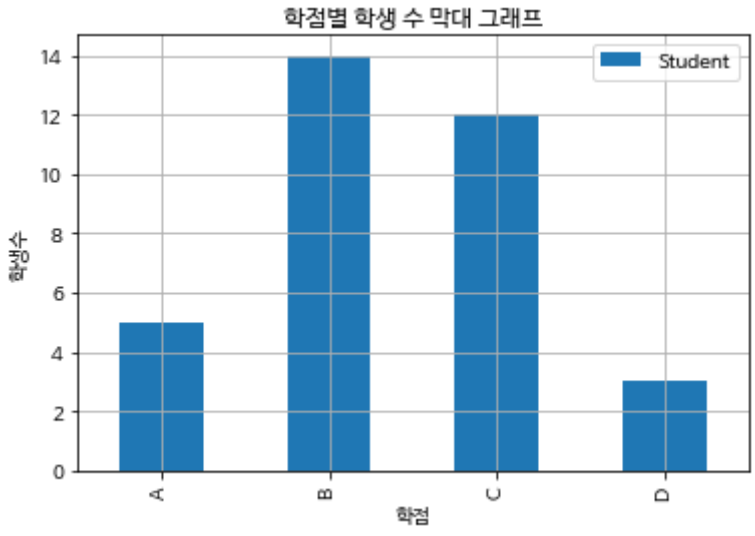
- 한 학급에서 학점이 (A, B, C, D)인 학생이 각각 (5, 14, 12, 3)명이라고 가정할 때 이를 이용해 pandas의 DataFrame 형식으로 데이터를 생성

```
import matplotlib.pyplot as plt
import pandas as pd

grade_num = [5, 14, 12, 3]
students = ['A', 'B', 'C', 'D']

df_grade = pd.DataFrame(grade_num, index=students, columns = ['Student'])
df_grade
```

| | Student |
|---|---|
| **A** | 5 |
| **B** | 14 |
| **C** | 12 |
| **D** | 3 |

- pandas에서 막대 그래프를 그리려면 plot(kind='bar') 혹은 plot.bar()를 이용

```
grade_bar = df_grade.plot.bar(grid = True)
grade_bar.set_xlabel("학점")
grade_bar.set_ylabel("학생수")
grade_bar.set_title("학점별 학생 수 막대 그래프")
plt.show()
```

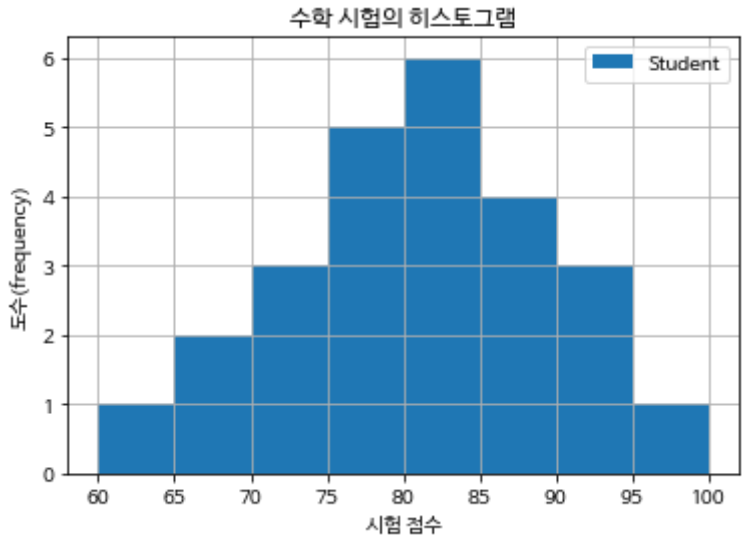



## pandas의 히스토그램

- pandas에서 히스토그램을 그리려면 plot(kind='hist'[, bin=num]) 혹은 plot.hist([bin=num])를 이용
- 옵션 bin은 계급의 개수
- bin 옵션이 없으면 기본적으로 bin은 10

- 'hist()의 인자로 _bin=8'을 넣어서 계급의 개수를 8로 조정

```
import matplotlib.pyplot as plt
import pandas as pd

math = [76,82,84,83,90,86,85,92,72,71,100,87,81,76,94,78,81,60,79,69,74,87,82,68,79]

df_math = pd.DataFrame(math, columns = ['Student'])

math_hist = df_math.plot.hist(bins=8, grid = True)
math_hist.set_xlabel("시험 점수")
math_hist.set_ylabel("도수(frequency)")
math_hist.set_title("수학 시험의 히스토그램")

plt.show()
```




## pandas의 파이 그래프

- pandas에서 파이 그래프를 그리려면 plot(kind= , 'pie') 혹은 plot.pie()를 이용

- 한 학급에서 20명의 학생이 5개의 과일(사과, 바나나, 딸기 , 오랜 지 , 포도) 중 제일 좋아하는 과일을 선택하라고 했을 때
- 과일별로 선택한 학생 수(7, 6, 3, 2, 2)

```
import matplotlib.pyplot as plt
import pandas as pd

fruit = ['사과', '바나나', '딸기', '오렌지', '포도']
result = [7, 6, 3, 2, 2]

df_fruit = pd.Series(result, index = fruit, name = '선택한 학생수')
df_fruit
```

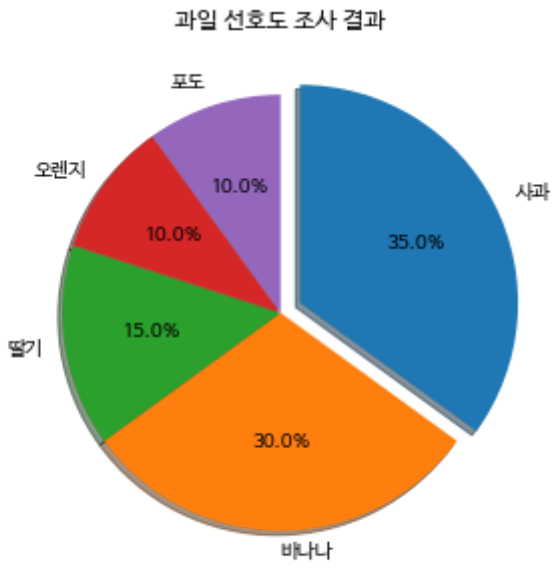
```
사과     7
바나나    6
딸기     3
오렌지    2
포도     2
Name: 선택한 학생수, dtype: int64
```

- pandas의 Series 데이터를 이용해 파이 그래프를 그림

```
df_fruit.plot.pie()
plt.show()
```


- pandas의 그래프도 plt.savefig()를 이용해 이미지 파일로 저장 할수 있음

```
explode_value = (0.1, 0, 0, 0, 0)
fruit_pie = df_fruit.plot.pie(figsize=(5, 5), autopct='%.1f%%', startangle=90,
    counterclock = False, explode=explode_value, shadow=True, table=True)
fruit_pie.set_ylabel("") # 불필요한 y축 라벨 제거
fruit_pie.set_title("과일 선호도 조사 결과")

# 그래프를 이미지 파일로 저장. dpi는 200으로 설정
plt.savefig('saveFigTest3.png', dpi = 200)
plt.show()
```




| | 사과 | 바나나 | 딸기 | 오렌지 | 포도 |
|---|---|---|---|---|---|
| 선택한 학생수 | 7 | 6 | 3 | 2 | 2 |

## 12.3 정리

- matplodib을 이용해 숫자 데이터를 그래프로 시각화하는 방법
- 기본인 선 그래프를 그리는 것에서 시작해 컬러 , 선 스타일, 마커를 지정하는 방법
- 그래프 창 하나에 여 러 그래프를 그리는 방법과 여러 개의 창을 나눠 그래프를 그리는 방법
- 라벨, 제목, 격자, 문자를 그래프에 추가하는 방법
- 선 그래프뿐만 아니라 산점도, 막 대 그래프, 히스토그램 , 파이 그래프를 그리는 방법
- 그려진 그래프를 다른 자 료에 사용할 수 있도록 그림 파일로 저장하는 방법
- pandas 형식의 데이 터를 이용해 시각화하는 방법