제주코딩베이스캠프

# 제주 하간디 이신 데이터들 Python으로 몬딱 분석해불게!

이호준 김유진 김혜원 김대현 이숭신 이현창 장하림 차경림

# 이호준

(現) 바울랩아이씨티기술연구원 대표이사
(現) 바울랩아이씨티컴퓨터학원 대표이사
(現) 사도출판 대표이사
(現) 바울랩미디어 대표이사
(現) 주식회사 위니브 대표이사
(現) GDG(Google Developers Group) JEJU Organizer
(現) 제주스타트업협회 부회장
(現) 제주대학교 풀스택 수업 강사
(現) 제주코딩베이스캠프 운영진

직책과 감투가 많지만, 사회에 공헌하며 미래를 꿈꾸는 아이들 가르치며 소박하게 살고 싶은 제주 도민입니다.

# 김대현

(前) 제주대학교 전산통계학과 전공
(現) 빅데이터 전략 마에스트로 교육생
(現) 바울랩아이씨티기술연구원 연구원

# 김유진

(前) 제주대학교 컴퓨터공학 전공
(現) 바울랩아이씨티기술연구원 연구원
제주코딩베이스캠프 Code Festival: Python 100제 집필

## 김혜원

(前) 제주대학교 컴퓨터공학 전공
(現) 바울랩아이씨티기술연구원 연구원
카카오와 함께하는 제주 코딩 베이스캠프 11기 스탭
코딩도장 튜토리얼로 배우는 Python 문제풀이 외 4권 집필

## 이승신

(前) 제주대학교 언론홍보학 전공
(現) 더큰내일센터 탐나는 인재 1기

## 이현창

(前) 제주대학교 전산통계학과 전공
(現) 빅데이터 전략 마에스트로 교육생
(現) 바울랩아이씨티기술연구원 연구원

## 장하림

(前) Liberty University mathematics major
(現) 빅데이터 전략 마에스트로 교육생
(現) 바울랩아이씨티기술연구원 연구원

## 차경림

(前) 제주대학교 산업디자인학부 문화조형디자인 전공
(現) 바울랩아이씨티기술연구원 연구원

**이 책은 전자책(PDF), PDF, Notion페이지를 제공하고 있습니다. 실습을 하실 때에는 노션에서 Code를 복사해 사용하세요.**

http://bitly.kr/GmGrTc2br

## Notion

데이터분석 1권 Part -1
http://bitly.kr/Vosbdg3xO

데이터분석 1권 Part -2
http://bitly.kr/jviJ6JeX9

데이터분석 2권
http://bitly.kr/W4LqEC3P5

데이터분석 30분 요약강좌(무료 동영상 강좌 제공)
http://bitly.kr/LvLUkRfbP

## PDF

데이터분석 1권
http://bitly.kr/OnZqDJiQM

데이터분석 2권
http://bitly.kr/j2bKWyz80N

데이터분석 30분 요약강좌(무료 동영상 강좌 제공)
http://bitly.kr/FuUspmPZv

# 이 책에 있는 데이터분석 강의는 인프런에서 유료로

# 30분 요약 강좌는 무료로 만나보실 수 있습니다.

# 인프런에서 제주코딩베이스캠프를 검색해주세요.

# Road Map


제주코딩베이스캠프 로드맵
junior basic
basic
advanced
total advanced
ect
초등학생을 위한 코딩교육
입문자를 위한 코딩교육
중급자를 위한 코딩교육
입문자~중급자 코딩교육
기타 강의들
- playbot(javascript)
- 러플(python)
- python
- html, css, bootstrap
- js+jquery,ajax
- Algorithm
- Django
- Data Analytics
- Node.js, Express
- Playbot 강의
- Rurple 강의
- html, css, js, python 30분 요약강좌
- 30분 요약강좌 시즌 2 : 파이썬 활용편
- 웹 개발의 시작 : html, css, bootstrap
- 웹 개발의 시작 : js, jquery, ajax
- Python basic
- Python 100제
- JS 100제
- Django Mini project BEST 3
- 인스타그램 클론 강의
- 페이스북 클론 강의
- Node.js, Express
- 제주 하간디 이신 데이터들 Python으로 몬딱 분석해불게
- 코알못에서 웹서비스 런칭까지 : 제주 코딩 베이스캠프
- python all-in-one
- 직접 만드는 노션 템플릿 BEST 7선
- 깔끔한 협업, 슬기로운 슬랙생활 시작하기
- 프로그래머의 콘텐츠 생산 가이드
- 나도 할 수 있다! sw강사!
- 온라인 개학에 대한 모든 것

# 대상 독자

이 책은 Python을 배우고 싶은 코딩 초보자 분들에게 추천합니다. 함께 Python 기초부터 실제 데이터 분석을 해보면서 데이터 분석을 처음부터 끝까지 경험해 보세요. 코딩으로 데이터 분석 시각화를 하고자 하는 분, 실제 데이터를 수집하여 분석하는 기존의 마케팅 및 영업 관련 업무를 담당하고 계신 분들에게 추천합니다.

코딩을 많이 해본 적 없는 초보자를 위해 기초 이론부터 실전 프로젝트까지 하나하나 탄탄하게 배울 수 있습니다.

# 이 책의 내용

튜토리얼 별로 Jupyter Notebook 코드와 Python, 데이터 분석을 위한 이론적인 학습 내용이 있습니다. 이론을 통해 배우고 실전 코드를 치면서 쉽고 재미있게 Python을 배울 수 있습니다. Python의 기본 문법부터 자료 구조 및 데이터 시각화를 배울 수 있고, 기본 분석부터 실무자를 위한 실전 프로젝트까지 진행할 수 있습니다. 데이터 분석을 위한 수학적 이론을 쉽게 배울 수 있습니다. 기초 이론부터 실전 프로젝트까지 빠르고 깊게 배울 수 있는 단기 집중 코스로 구성된 책입니다.

# Contents

## 1일차. Numpy



## 2일차. Pandas

## 3일차. 기본 실습



## 4일차. 데이터 시각화

## 5일차. 데이터 시각화 2



## 6일차. 코로나 데이터 분석

# 7일차. 데이터 분석을 위한 수학적 이론

# 1일차 Numpy

이 장에서 다루는 내용

Numpy란
Numpy 차원
Numpy 배열
Numpy 함수
Boolean Indexing
Broadcast
Random

## NumPy란

- 행렬 연산이나 대규모 다차원 배열을 편리하게 처리할 수 있도록 지원하는 파이썬 라이브러리
- NumPy는 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공

출처 : 위키백과

## NumPy 특징

- N 차원 배열 객체
- 기본적으로 array 단위로 데이터 관리
- 큰 규모의 데이터 연산을 빠르게 수행 (반복문 없이 배열에 대한 처리 지원)
- 정교한 브로드캐스팅(Broadcast) 기능

## ndarray 클래스

- Numpy의 핵심인 다차원 행렬 자료 구조를 지원하는 클래스

```python
# Vector(1차원 행렬)
import numpy as np

a = np.array([1,2,3,4,5,6,7])
print(type(a))
```

Python

```
Out[-]
<class 'numpy.ndarray'>
```

Python

## NumPy 산술 연산

```python
import numpy as np

data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
dataTwo = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]

# 행렬과 스칼라의 곱
for i in range(len(data)):
    for j in range(len(data[0])):
        data[i][j] *= 2

print(data)

# 행렬끼리 덧셈
for i in range(len(data)):
    for j in range(len(data[0])):
        data[i][j] += dataTwo[i][j]

print(data)
```

Python

```
Out[-]
[[2, 4, 6], [8, 10, 12], [14, 16, 18]]
[[3, 5, 7], [10, 12, 14], [17, 19, 21]]
```

Python

```python
import numpy as np

data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
dataTwo = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]

a = np.array(data)
b = np.array(dataTwo)

print(a * 2)
print(a + a)
print(a + b)
print(a * b)
print(np.dot(a, b)) # 행렬곱
```

Python

```
Out[-]
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]

[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]

[[ 2  3  4]
 [ 6  7  8]
 [ 10 11 12]]

[[ 1  2  3]
 [ 8 10 12]
 [21 24 27]]

[[ 14 14 14]
 [ 32 32 32]
 [ 50 50 50]]
```

Python

## 데이터 타입 종류

1. int(8bit, 16bit, 32bit, 64bit) i1, i2, i4, i8

- 부호가 있음
- 비트수 만큼 크기를 가지는 정수형
- 저장할 수 있는 값의 범위
  - int8 : 8 비트 부호있는 정수
  - int16 : 16 비트 부호있는 정수
  - int32 : 32 비트 부호있는 정수
  - int64 : 64 비트 부호있는 정수

2. uint(8bit, 16bit, 32bit, 64bit) u1, u2, u4, u8

- 부호가 없음
- 비트수 만큼 크기를 가지는 정수형
- 저장할 수 있는 값의 범위
  - uint8 : 8비트 부호없는 정수
  - uint16 : 16비트 부호없는 정수
  - uint32 : 32비트 부호없는 정수
  - uint64 : 64비트 부호없는 정수

3. float(16bit, 32bit, 64bit, 128bit) f2, f4, f8, f16

- 부호가 있음
- 비트수 만큼 크기를 가지는 실수형

4. 복소수형

- complex64 : 두개의 32비트 부동 소수점으로 표시되는 복소수 c8
- complex128 : 두개의 64비트 부동소수점으로 표시되는 복소수 c16

5. unicode

- 고정 길이 문자열 unicode

6. bool

- True, False

```python
data = [1.1, 2, 3]

a = np.array(data)
a.dtype
```

Python

```
Out[-]
dtype('float64')
```

Python

```python
data = [1.1, 2, 3]

a = np.array(data, dtype=np.float32)
a.dtype
```

Python

```
Out[-]
dtype('float32')
```

Python

```python
data = [1.1, 2, 3]

a = np.array(data, dtype=np.int32)
print(a.dtype)
print(a)
```

Python

```
Out[-]
int32
[1 2 3]
```

Python

```
data = [1.1, 2, 3]

a = np.array(data, dtype='i4')
a.dtype
```

Python

```
Out[-]
dtype('int32')
```

Python

## 데이터 형 변환

방법. 1

- variable(변수) = np.astype(data)
- astype : 변환하고자 하는 데이터 타입 지정
- data : 변환하고자 하는 array 지정

방법. 2

- variable(변수) = ndarray.astype(dtype)
- ndarray : 변환하고자 하는 array 지정
- dtype : 변환하고자 하는 데이터 타입 지정

```python
data = [1.1, 2, 3]

a = np.float64(data)
print(a.dtype)
print(a)
a = np.int32(a)
print(a.dtype)
print(a)
```

Python

```
Out[-]
float64
[1.1 2.  3. ]
int32
[1 2 3]
```

Python

```python
data = [1.1, 2, 3]

a = np.float64(data)
print(a.dtype)
print(a)
a = a.astype(np.int64)
print(a.dtype)
print(a)
```

Python

```
Out[-]
float64
[1.1 2.  3. ]
int64
[1 2 3]
```

Python

```python
data = [1.1, 2, 3]

a = np.float64(data)
print(a.dtype)
a = a.astype(np.string_)
print(a.dtype)
print(a)
```

Python

```
Out[-]
float64
|S32
[b'1.1' b'2.0' b'3.0']
```

Python

```python
a = np.uint16(0)
print(a.dtype)
a = a - 1
print(a.dtype)
```

Python

```
Out[-]
uint16
int32
```

Python

```python
a = np.uint16(-1)
a
```

Python

```
Out[-]
65535
```

Python

## NumPy 차원

- Scalar : 하나의 데이터 값으로만 존재하는 것
- Vector : 숫자들의 배열 (1D array)
- Matrix : 숫자들의 2D array (rows: 행, columns: 열)

```python
# 0 차원 (Scalar)
a = np.array(1)
print(a)
print(a.shape, a.ndim)
```

Python

```
Out[-]
1
() 0
```

Python

```python
# 1 차원 (Vector)
a = np.array([1, 2, 3])
print(a)
print(a.shape, a.ndim)
```

Python

```
Out[-]
[1 2 3]
(3,) 1
```

Python

```python
# 1 차원
a = np.array([1])
print(a)
print(a.shape, a.ndim)
```

Python

```
Out[-]
[1]
(1,) 1
```

Python

```python
# 2 차원
a = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
print(a)
print(a.shape, a.ndim)
```

Python

```
Out[-]
[[1 2 3]
 [1 2 3]
 [1 2 3]]

 (3, 3) 2
```

Python

```python
# 2 차원 (Matrix)
a = np.array([[1]])
print(a)
print(a.shape, a.ndim)
```

Python

```
Out[-]
[[1]]
(1, 1) 2
```

Python

```python
# 3 차원(3차원 이상의 다차원의 행렬을 Tensor)
a = np.array([[[1, 2], [1, 2], [1, 2]], [[1, 2], [1, 2], [1, 2]]])
print(a)
print(a.shape, a.ndim)
```

Python

```
Out[-]
[[[1 2]
  [1 2]
  [1 2]]

 [[1 2]
  [1 2]
  [1 2]]]
(2, 3, 2) 3
```

Python

## Ndarray(다차원 array)

- 서로 다른 타입의 데이터를 담을 수 없다
- 내부 반복문을 사용해서 속도가 빠르다
- 배열 간에 산술 연산이 가능하다
- NumPy는 다차원 배열을 지원하고 배열의 구조는 'shape'로 표현된다
- 다차원 배열은 입체적인 데이터 구조를 가진다

```python
a = np.array([i for i in range(2, 11, 2)]) # range(start, stop, step)
a
```

Python

```
Out[-]
array([ 2, 4, 6, 8, 10])
```

Python

```python
a = np.array(range(2, 11, 2))
a
```

Python

```
Out[-]
array([ 2, 4, 6, 8, 10])
```

Python

Copy to clipboard

```python
print(np.arange(10))
print(np.arange(2, 11, 2))   # np.arange(start, stop, step)
print(np.arange(10, 15, .5))
print(np.arange(10, 2, -1))
print(np.arange(10, 2, -1.5))
```

Python

```
Out[-]
[0 1 2 3 4 5 6 7 8 9]
[ 2  4  6  8 10]
[10.  10.5 11.  11.5 12.  12.5 13.  13.5 14.  14.5]
[10  9  8  7  6  5  4  3]
[10.   8.5  7.   5.5  4.   2.5]
```

Python

```python
a = np.arange(10)
print(a.ndim)
print(a.shape)
print(a.size)
```

Python

```
Out[-]
1
(10,)
10
```

Python

## indexing

- 특정 위치를 가리키는 데이터를 가져오는 방법

- 1차원 indexing

```python
a = np.arange(10)
a
```

Python

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Python

```python
a[2]
```

Python

```
Out[-]
2
```

Python

- 2차원 indexing

```python
a = np.arange(12).reshape(3, 4)
a
```

Python

```
Out[-]
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Python

```python
a[1][3]
```

Python

```
Out[-]
7
```

Python

```python
a[-2][-1]
```

Python

```
Out[-]
7
```

Python

```python
a[1, 3]
```

Python

```
Out[-]
7
```

Python

- 3차원 indexing

```python
a = np.arange(40).reshape(2, 5, 4)
a
```

Python

```
Out[-]
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15],
        [16, 17, 18, 19]],

       [[20, 21, 22, 23],
        [24, 25, 26, 27],
        [28, 29, 30, 31],
        [32, 33, 34, 35],
        [36, 37, 38, 39]]])
```

Python

```python
a[0][1][1]
```

Python

```
Out[-]
5
```

Python

```python
a[-1][-3][-3]
```

Python

```
Out[-]
29
```

Python

```python
a[1][2][1]
```

Python

```
Out[-]
29
```

Python

```python
a[-1, -3, -3]
```

Python

```
Out[-]
29
```

Python

```python
a[1, 2, 1]
```

Python

```
Out[-]
29
```

Python

```python
a[a > 30]
```

Python

```
Out[-]
array([31, 32, 33, 34, 35, 36, 37, 38, 39])
```

Python

## slicing

- 연속된 객체에서 범위를 지정하고 선택한 객체를 가져옴
- 1차원 → [start : end : step]
- 2차원 배열은 행과 열을 콤마로 구분 → [ : , :]
- 3차원 배열 → [ :, :, :]

- 1차원 slicing

```python
a = np.arange(30)
a
```

Python

```
Out[-]
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

Python

```python
a[::-1] # 역순
```

Python

```
Out[-]
array([29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13,
       12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0])
```

Python

```python
a[5:20:2] # a[start:stop:step]
```

Python

```
Out[-]
array([ 5,  7,  9, 11, 13, 15, 17, 19])
```

Python

```python
a = a.reshape(5, 6)
a
```

Python

- 2차원 slicing

```
Out[-]
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29]])
```

Copy to clipboard ••• Python

```
print(a[:, 1])
print(a[1, :]) # a[1]
```

Python

```
Out[-]
[ 1  7 13 19 25]

[ 6  7  8  9 10 11]
```

Python

```
a[:3, :3]
```

Python

```
Out[-]
array([[ 0,  1,  2],
       [ 6,  7,  8],
       [12, 13, 14]])
```

Python

```
a[::2, ::2]  # 처음부터 끝까지 2개 간격으로
```

Python

Copy to clipboard •••

```
Out[-]
array([[ 0,  2,  4],
       [12, 14, 16],
       [24, 26, 28]])
```

Python

```
b = a[::2, ::2]
b
```

Python

```
Out[-]
array([[ 0,  2,  4],
       [12, 14, 16],
       [24, 26, 28]])
```

Python

```
b[0] = 100
b
```

Python

```
Out[-]
array([[100, 100, 100],
       [ 12,  14,  16],
       [ 24,  26,  28]])
```

Python

```
a
```

Python

```
Out[-]
array([[100,   1, 100,   3, 100,   5],
       [  6,   7,   8,   9,  10,  11],
       [ 12,  13,  14,  15,  16,  17],
       [ 18,  19,  20,  21,  22,  23],
       [ 24,  25,  26,  27,  28,  29]])
```

Python

Copy to clipboard

```python
b = a[::2, ::2].copy() # 원래의 배열과 독립된 배열로 처리
b
```

Python

```
Out[-]
array([[100, 100, 100],
       [ 12,  14,  16],
       [ 24,  26,  28]])
```

Python

```python
b[0] = 1000
b
```

Python

```
Out[-]
array([[1000, 1000, 1000],
       [  12,   14,   16],
       [  24,   26,   28]])
```

Python

```python
a
```

Python

```
Out[-]
array([[100,   1, 100,   3, 100,   5],
       [  6,   7,   8,   9,  10,  11],
       [ 12,  13,  14,  15,  16,  17],
       [ 18,  19,  20,  21,  22,  23],
       [ 24,  25,  26,  27,  28,  29]])
```

Python

- 3차원 slicing

```python
a = np.arange(30).reshape(2, 5, 3) # 2x5x3
a
```

Python

```
Out[-]
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11],
        [12, 13, 14]],

       [[15, 16, 17],
        [18, 19, 20],
        [21, 22, 23],
        [24, 25, 26],
        [27, 28, 29]]])
```

Python

```python
a[:,:,1] # 첫 번째 열만 출력
```

Python

```
Out[-]
array([[ 1,  4,  7, 10, 13],
       [16, 19, 22, 25, 28]])
```

Python

```python
a[:,1,:] # 첫 번째 행만 출력
```

Python

Copy to clipboard

```
Out[-]
array([[ 3,  4,  5],
       [18, 19, 20]])
```

Python

```python
a[1,:,:] # 첫 번째 배열 출력
```

Python

```
Out[-]
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])
```

Python

```python
a[1,:,::-1]
```

Python

```
Out[-]
array([[17, 16, 15],
       [20, 19, 18],
       [23, 22, 21],
       [26, 25, 24],
       [29, 28, 27]])
```

Python

# NumPy 배열

## 배열의 연결과 분할

1. 연결
   - hStack : 열 추가
   - vstack : 행 추가
   - concatenate ([a, b], axis = ) : axis=0 → 행 추가, axis=1 → 열추가
2. 분할
   - vsplit : 수직축으로 분할, 행 분할
   - hsplit : 수평축으로 분할, 열 분할

```python
x = [1, 2, 3]
y = [4, 5, 6]

# np.array(x) + np.array(y) # x와 y의 각 자리에 맞는것 끼리 더함=> array([5, 7, 9])
a = np.array(x)
b = np.array(y)

np.concatenate([a, b])
```

Python

```
Out[-]
array([1, 2, 3, 4, 5, 6])
```

Python

```python
a = np.arange(10).reshape(2, 5)
b = np.arange(10, 20).reshape(2, 5)

np.concatenate([a, b])
```

Python

```
Out[-]
array([[[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9]],

       [[10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19]],

       [[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9]],

       [[10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19]]])
```

Python

Copy to clipboard

```
np.concatenate([a, b], axis=1)
```

Python

```
Out[-]
array([[ 0,  1,  2,  3,  4, 10, 11, 12, 13, 14],
       [ 5,  6,  7,  8,  9, 15, 16, 17, 18, 19]])
```

Python

```
a = np.arange(12).reshape(2, 6)
b = np.arange(10, 20).reshape(2, 5)

np.hstack([a, b])
```

Python

```
Out[-]
array([[ 0,  1,  2,  3,  4,  5, 10, 11, 12, 13, 14],
       [ 6,  7,  8,  9, 10, 11, 15, 16, 17, 18, 19]])
```

Python

```python
a = np.arange(15).reshape(3, 5)
b = np.arange(10, 20).reshape(2, 5)

np.vstack([a, b]) # vertical stack
```

Python

```
Out[-]
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

Python

```python
a = np.arange(12)
# np.split(a, 5)
# np.split(a, 2)
np.split(a, 3) # a를 3개의 배열로 분할
```

Python

```
Out[-]
[array([0, 1, 2, 3]), array([4, 5, 6, 7]), array([ 8, 9, 10, 11])]
```

Python

```python
a = np.arange(10, 121, 10)
a
```

Python

```
Out[-]
array([ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120])
```

Python

Copy to clipboard

```python
np.split(a, [3, 5, 9, 10]) # a를 [3,5,9,10]위치까지 나누기
                           # a[0:3], a[3:5], a[5:9], a[9:10], a[10:]
```

Python

```
Out[-]
[array([10, 20, 30]),
 array([40, 50]),
 array([60, 70, 80, 90]),
 array([100]),
 array([110, 120])]
```

Python

```python
a = np.arange(10, 121, 10).reshape(3, 4)
a
```

Python

```
Out[-]
array([[ 10,  20,  30,  40],
       [ 50,  60,  70,  80],
       [ 90, 100, 110, 120]])
```

Python

```python
np.vsplit(a,3)
```

Python

```
Out[-]
[array([[10, 20, 30, 40]]),
 array([[50, 60, 70, 80]]),
 array([[ 90, 100, 110, 120]])]
```

Python

```python
np.hsplit(a, 4)
```

Python

```
Out[-]
[array([[10],
        [50],
        [90]]),
 array([[ 20],
        [ 60],
        [100]]),
 array([[ 30],
        [ 70],
        [110]]),
 array([[ 40],
        [ 80],
        [120]])]
```

Python

```
np.hsplit(a,4)[0]
```

Python

```
Out[-]
array([[10],
       [50],
       [90]])
```

Python

## 다양한 Matrix 만들기

- zeros : 0으로 초기화된 배열 생성
- ones : 1로 초기화 된 배열 생성
- eye : 주대각선의 원소가 모두 1이고 나머지 원소는 0인 정사각행렬 (단위행렬)
- empty : 초기화 하지 않고 배열만 생성, 기존에 메모리에 저장되어 있는 값으로 나타남
- linespace(시작, 끝, 개수, endpoint = ) : 지정한 구간에서 개수만큼 분할
  - endpoint = True or False : 마지막 값을 포함시킬지 시키지 않을지 선택
- logspace(시작, 끝, 개수, endpoint = ) : 지정한 구간에서 개수만큼 로그를 이용하여 분할
- ravel : 다차원 배열을 1차원 배열로 변환

```python
a = np.arange(12).reshape(3, 4)
a
```

Python

```
Out[-]
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Python

```python
b = np.ones(12)
b
```

Python

```
Out[-]
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Python

```python
c = np.ones([3, 4])
c
```

Python

```
Out[-]
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

Python

```python
c = np.ones([3, 4, 5])
c
```

Python

```
Out[-]
array([[[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]]])
```

Python

Copy to clipboard

```
c = np.zeros([3, 4, 5])
c
```

Python

```
Out[-]
array([[[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]])
```

Python

```
d = np.eye(3)
d
```

Python

```
Out[-]
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Python

```python
e = np.arange(9).reshape(3, 3)
e
```

Python

```
Out[-]
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Python

```python
np.dot(d, e)
```

Python

```
Out[-]
array([[0., 1., 2.],
       [3., 4., 5.],
       [6., 7., 8.]])
```

Python

```python
np.empty(10)
```

Python

```
Out[-]
array([8.90070286e-308, 2.11393900e-307, 1.27945906e-307, 1.11258192e-307,
       9.79098366e-307, 1.00133162e-307, 1.11261027e-306, 1.29061821e-306,
       8.90103559e-307, 1.24611470e-306])
```

Python

```python
np.full((3, 4), 100)  # 모든값을 100으로 초기화
```

```
Out[-]
array([[100, 100, 100, 100],
       [100, 100, 100, 100],
       [100, 100, 100, 100]])
```

```python
np.linspace(2, 10, 6) # 2부터 10까지 6개로 균등하게 분할
```

```
Out[-]
array([ 2. , 3.6, 5.2, 6.8, 8.4, 10. ])
```

```python
np.linspace(2, 100, 10)
```

```
Out[-]
array([  2.        ,  12.88888889,  23.77777778,  34.66666667,
        45.55555556,  56.44444444,  67.33333333,  78.22222222,
        89.11111111, 100.        ])
```

```python
np.logspace(2, 100, 10)
```

Python

```
Out[-]
array([1.00000000e+002, 7.74263683e+012, 5.99484250e+023, 4.64158883e+034,
       3.59381366e+045, 2.78255940e+056, 2.15443469e+067, 1.66810054e+078,
       1.29154967e+089, 1.00000000e+100])
```

Python

```python
a = np.arange(12).reshape(3, 4)
a
```

Python

```
Out[-]
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Python

Copy to clipboard

```python
a.ravel()
```

Python

```
Out[-]
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
```

Python

```python
a.ravel(order='C') # 행기준
```

Python

```
Out[-]
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
```

Python

```python
a.ravel(order='F') # 열기준
```

Python

```
Out[-]
array([ 0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11])
```

Python

# NumPy 함수

## 범용함수 (universal function)

- 배열의 값에 반복된 연산을 빠르게 수행하는 함수

```python
max(range(100))
```

Python

```
Out[-]
99
```

Python

```python
min(range(100))
```

Python

```
Out[-]
0
```

Python

```python
%%timeit                # 셀 코드의 실행시간을 측정하는 IPython 매직 명령

for i in range(10):
    max(range(1000000))
```

Python

```
Out[-]
425 ms ± 23.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Python

Copy to clipboard

```
%%timeit

for i in range(10):
    np.max(np.arange(1000000))
```

Python

```
Out[-]
43.8 ms ± 2.52 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Python

```
a = np.arange(12).reshape(3, 4)
a
```

Python

```
Out[-]
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Python

```python
np.max(a)
```

Python

```
Out[-]
11
```

Python

```python
np.max(a, axis=1)
```

Python

```
Out[-]
array([ 3, 7, 11])
```

Python

```python
a[0][3] = 1
np.max(a, axis = 1)
```

Python

```
Out[-]
array([ 2, 7, 11])
```

Python

## 집계 함수

- mean : 평균 구하는 함수
- median : 중앙값 구하는 함수
- std : 표준편차 구하는 함수
- var : 분산 구하는 함수
- sum : 합계 구하는 함수
- cusum : 누적값 구하는 함수
- argmin : 최소값에 해당하는 index 값 찾는 함수
- any : 배열에서 1개 이상의 원소가 참인지 평가하는 함수
- all : 배열의 모든 원소가 참인지 평가하는 함수
- nansum : NaN을 0으로 간주하고 더는 함수
- where(조건, 조건에 맞을 때 값, 조건과 다를 때 값)

```python
a = np.arange(12).reshape(3, 4)
a
```

Python

```
Out[-]
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Python

```python
a[0][0] = 100
np.mean(a)
```

Python

```
Out[-]
13.833333333333334
```

Python

```python
np.median(a)
```

Python

```
Out[-]
6.5
```

Python

```python
np.std(a)
```

Python

```
Out[-]
26.15604880116431
```

Python

```python
np.var(a)
```

Python

```
Out[-]
684.138888888889
```

Python

Copy to clipboard •••

```python
np.sum(a)
```

Python

```
Out[-]
166
```

Python

```python
np.sum(a, axis=1)
```

Python

```
Out[-]
array([106, 22, 38])
```

Python

Copy to clipboard

```python
a
```

Python

```
Out[-]
array([[100,   1,   2,   3],
       [  4,   5,   6,   7],
       [  8,   9,  10,  11]])
```

Python

```python
np.cumsum(a)
```

Python

```
Out[-]
array([100, 101, 103, 106, 110, 115, 121, 128, 136, 145, 155, 166],
      dtype=int32)
```

Python

```python
np.min(a)
```

Python

```
Out[-]
1
```

Python

```python
np.argmin(a)
```

Python

```
Out[-]
1
```

Python

```python
np.argmax(a)
```

Python

```
Out[-]
0
```

Python

```python
np.any(a)
```

Python

```
Out[-]
True
```

Python

```python
np.all(a)
```

Python

```
Out[-]
True
```

Python

```python
np.nansum(a)
```

Python

```
Out[-]
166
```

Python

```python
a
```

Python

```
Out[-]
array([[100,   1,   2,   3],
       [  4,   5,   6,   7],
       [  8,   9,  10,  11]])
```

Python

```python
a > 5
```

Python

```
Out[-]
array([[ True, False, False, False],
       [False, False,  True,  True],
       [ True,  True,  True,  True]])
```

Python

```python
np.all(a > 5)
```

Python

```
Out[-]
False
```

Python

Copy to clipboard

```python
np.all(a > -5)
```

Python

```
Out[-]
True
```

Python

```python
np.where(a > 5)
```

Python

```
Out[-]
(array([0, 1, 1, 2, 2, 2, 2], dtype=int64),
 array([0, 2, 3, 0, 1, 2, 3], dtype=int64))
```

Python

```python
a
```

Python

```
Out[-]
array([[100,   1,   2,   3],
       [  4,   5,   6,   7],
       [  8,   9,  10,  11]])
```

Python

```python
np.where(a > 5, a, -100)
```

Python

```
Out[-]
array([[ 100, -100, -100, -100],
       [-100, -100,    6,    7],
       [   8,    9,   10,   11]])
```

Python

```python
a = np.arange(36).reshape(3, 4, 3)
a
```

Python

```
Out[-]
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11]],

       [[12, 13, 14],
        [15, 16, 17],
        [18, 19, 20],
        [21, 22, 23]],

       [[24, 25, 26],
        [27, 28, 29],
        [30, 31, 32],
        [33, 34, 35]]])
```

Python

```python
print(np.sum(a))
print(np.sum(a, axis=1))
print(np.sum(a, axis=2))
```

Python

```
Out[-]
630

[[ 18  22  26]
 [ 66  70  74]
 [114 118 122]]

[[  3  12  21  30]
 [ 39  48  57  66]
 [ 75  84  93 102]]
```

Python

## Boolean Indexing

- 조건에 만족 여부를 따져 논리값(True, False)을 출력
- 단, 행 값을 활용한 조건만 선택이 가능

```python
a = np.arange(12).reshape(3, 4)
a
```

Python

```
Out[-]
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Python

```python
a > 5
```

Python

```
Out[-]
array([[False, False, False, False],
       [False, False,  True,  True],
       [ True,  True,  True,  True]])
```

Python

```python
a[a > 5]
```

Python

```
Out[-]
array([ 6, 7, 8, 9, 10, 11])
```

Python

```python
a[a > 5] = 1
a
```

Python

```
Out[-]
array([[0, 1, 2, 3],
       [4, 5, 1, 1],
       [1, 1, 1, 1]])
```

Python

```python
a[~(a == 1)]
```

Python

```
Out[-]
array([0, 2, 3, 4, 5])
```

Python

```python
a[~(a == 1) & (a < 4)]
```

Python

```
Out[-]
array([0, 2, 3])
```

Python

```python
a[~(a == 1) | (a < 4)]
```

Python

```
Out[-]
array([0, 1, 2, 3, 4, 5, 1, 1, 1, 1, 1, 1])
```

Python

## Broadcast

- 크기가 다른 배열 간의 연산 함수를 적용한는 규칙 집합

Copy to clipboard

```python
np.array([1, 2, 3]) + 1
```

Python

```
Out[-]
array([2, 3, 4])
```

Python

```python
np.arange(9).reshape(3, 3) + np.array([1, 2, 3])
```

Python

```
Out[-]
array([[ 1,  3,  5],
       [ 4,  6,  8],
       [ 7,  9, 11]])
```

Python

```python
np.arange(27).reshape(3, 3, 3) + np.array([1, 2, 3])
```

Python

```
Out[-]
array([[[ 1,  3,  5],
        [ 4,  6,  8],
        [ 7,  9, 11]],

       [[10, 12, 14],
        [13, 15, 17],
        [16, 18, 20]],

       [[19, 21, 23],
        [22, 24, 26],
        [25, 27, 29]]])
```

Python

## random

- seed : 시작 숫자를 정해 정해진 난수 알고리즘으로 난수 생성
- shuffle : 데이터의 순서를 바꿔줌
- choice : 데이터에서 일부를 무작위로 선택
- rand : 0부터 1까지 균일 분포로 난수 생성
- randn : 가우시안 표준 정규 분포로 난수 생성

Copy to clipboard •••

```python
import random as r

print(r.randint(1, 10)) # 1부터 10까지의 임의의 정수를 리턴
print(r.random()) # 0부터 1까지의 부동소수점 숫자를 리턴
print(r.uniform(1, 10)) # 1부터 10까지의 부동소수점 숫자를 리턴
print(r.randrange(1, 10, 2)) # 1부터 9까지 2 간격으로 임의의 정수를 리턴
```

Python

```
Out[-]
2
0.36568891691258554
1.5219903229723613
9
```

Python

```python
l = [i for i in range(20)]
r.shuffle(l)
l
```

Python

```
Out[-]
[3, 9, 4, 19, 17, 11, 0, 14, 5, 7, 12, 15, 13, 8, 1, 10, 2, 6, 18, 16]
```

Python

```python
r.choice(l)
```

Python

```
Out[-]
17
```

Python

```python
r.sample(l, 5)
```

Python

```
Out[-]
[9, 15, 12, 17, 19]
```

Python

```python
r.seed(7)
r.randint(1, 10)
```

Python

```
Out[-]
6
```

Python

```python
print(np.random.randint(1, 10))
print(np.random.normal(1, 10, size=(10)))
print(np.random.uniform(1, 10, size=(3, 4)))
```

Python

```
Out[-]
2

 [ 22.54439604   0.06003082  -2.38677184   1.83293534  -5.10144747
  -1.16943924  -0.9915091    2.10254182 -13.11856085   3.66203174]

 [[5.55433615 5.66114991 5.92052974 6.67300498]
 [8.70958655 5.38328336 3.88877589 6.53031795]
 [5.80594633 3.99338487 8.55861939 7.99957725]]
```

Python

```python
np.random.seed(1)
np.random.randint(1, 10)
```

Python

```
Out[-]
6
```

Python

# 2일차 Pandas

이 장에서 다루는 내용

Pandas

Series

DataFrame

## pandas

- Python에서 사용하는 데이터를 분석하는 라이브러리
- 행과 열을 쉽게 처리할 수 있는 함수를 제공하는 도구
- numpy보다 유연하게 수치연산 가능
- numpy는 데이터 누락을 허락하지 않지만, pandas는 데이터 누락을 허락

```python
import pandas as pd
import numpy as np
# 버전 확인
print(pd.__version__)
print(np.__version__)

pd?                    # tab누르면 사용가능한 여러가지 method를 볼 수 있다.
np?
```

Python

```
Out[-]
1.0.1
1.18.1
```

Python

## Series

- 인덱스와 values로 이루어진 1차원 배열
- 모든 유형의 데이터를 보유할 수 있음
- 인덱스를 지정해 줄 수 있음
- 명시적 인덱스와 암묵적 인덱스를 가짐

### Series 형태

- RangeIndex : 인덱스 자동 생성

```python
data = [100, 200, 300, 400, 500]
pd.Series(data)
```

Python

```
Out[-]
0    100
1    200
2    300
3    400
4    500
dtype: int64
```

Python

```python
print(pd.Series(data)[0])
print(pd.Series(data)[0:3])
print(pd.Series(data)[::-1]) # 역순
```

Python

Copy to clipboard

```
Out[-]
100

0    100
1    200
2    300
dtype: int64

4    500
3    400
2    300
1    200
0    100
dtype: int64
```

Python

```python
d = pd.Series(data)
print(d.values)
print(d.index)
```

Python

```
Out[-]
[100 200 300 400 500]
RangeIndex(start=0, stop=5, step=1)
```

Python

```python
d = pd.Series(data, index=['a', 'b', 'c', 'd', 'e']) # 인덱스 지정
print(d)
print(d.values)
print(d.index)
```

Python

## Series 산술 연산

```
Out[-]
a    100
b    200
c    300
d    400
e    500
dtype: int64

[100 200 300 400 500]

Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

Python

```
print(d + 100)
print(d * 2)
print(d // 2)
print(d ** 2)
```

Python

```
Out[-]
a    200
b    300
c    400
d    500
e    600
dtype: int64

a     200
b     400
c     600
d     800
e    1000
dtype: int64

a     50
b    100
c    150
d    200
e    250
dtype: int64

a     10000
b     40000
c     90000
d    160000
e    250000
dtype: int64
```

Python

## Series indexing , slicing

```python
print(d['a'])
print(d[::-1])
print(d[::2]) # 전체에서 두 칸씩 건너 뛰면서 출력
print(d.a)
print(d.b)
print(d.c)
```

Python

Copy to clipboard

```
Out[-]
100

e    500
d    400
c    300
b    200
a    100
dtype: int64

a    100
c    300
e    500
dtype: int64

100

200

300
```

Python

```python
d[d>300]
```

Python

```
Out[-]
d    400
e    500
dtype: int64
```

Python

```python
d > 300
```

Python

```
Out[-]
a    False
b    False
c    False
d     True
e     True
dtype: bool
```

Python

```python
for i in d:
    print(i)
```

Python

```
Out[-]
100
200
300
400
500
```

Python

```python
# 딕셔너리 형태도 Series로 만들 수 있음
dic = {
    '2015년':1000000,
    '2016년':2000000,
    '2017년':3000000,
    '2018년':4000000,
    '2019년':11000000,
    '2020년':30000000,
}
pd.Series(dic)['2018년':]
```

Python

```
Out[-]
2018년     4000000
2019년    11000000
2020년    30000000
dtype: int64
```

Python

```python
pd.Series(dic)[-3:] # 뒤에서 3번째까지 출력
```

Python

Copy to clipboard •••

```
Out[-]
2018년     4000000
2019년    11000000
2020년    30000000
dtype: int64
```

Python

```python
dic = {
    '2015년':1000000,
    '2016년':2000000,
    '2017년':3000000,
    '2018년':4000000,
    '2019년':11000000,
    '2020년':30000000,
}
pd.Series(dic, index=['2017년', '2019년', '2020년'])
```

Python

```
Out[-]
2017년     3000000
2019년    11000000
2020년    30000000
dtype: int64
```

Python

## Series에 key, value, index

- index
  - Series, DataFrame의 레코드를 식별
  - 집합 연산이 가능
- loc : 인덱스를 기반으로 행 데이터를 읽음
- iloc : 행 번호를 기반으로 행 데이터를 읽음
- items() : key와 value를 튜플로 묶어서 리턴
- 팬시 인덱싱 : 스칼라 대신 인덱스 배열을 리턴

```python
s = pd.Series(dic)
print('2015년' in s)
print(1000000 in s)
```

Python

```
Out[-]
True
False
```

Python

```python
print('2015년' in dic)
print(1000000 in dic)
```

Python

```
Out[-]
True
False
```

Python

```
s.keys()
```

Python

```
Out[-]
Index(['2015년', '2016년', '2017년', '2018년', '2019년', '2020년'], dtype='object')
```

Python

```
list(s.items()) # Series에서는 values를 허락하지 않음
                # items를 통해서 values 확인
```

Python

```
Out[-]
[('2015년', 1000000),
 ('2016년', 2000000),
 ('2017년', 3000000),
 ('2018년', 4000000),
 ('2019년', 11000000),
 ('2020년', 30000000)]
```

Python

```
# 딕셔너리에서는 key, value, item을 다 허락
print(dic.keys())
print(dic.values())
print(dic.items())
```

Python

```
Out[-]
dict_keys(['2015년', '2016년', '2017년', '2018년', '2019년', '2020년'])
dict_values([1000000, 2000000, 3000000, 4000000, 11000000, 30000000])
dict_items([('2015년', 1000000), ('2016년', 2000000), ('2017년', 3000000), ('2018년', 4000
000), ('2019년', 11000000), ('2020년', 30000000)])
```

Python

```
s
```

Python

```
Out[-]
2015년      1000000
2016년      2000000
2017년      3000000
2018년      4000000
2019년     11000000
2020년     30000000
dtype: int64
```

Python

```
s[['2017년', '2020년']] # 팬시 인덱싱
```

Python

```
Out[-]
2017년      3000000
2020년     30000000
dtype: int64
```

Python

```
data = ['a', 'b', 'c', 'd', 'e']
pd.Series(data)[::2]    # 두 칸씩 슬라이싱
```

Python

```
Out[-]
0    a
2    c
4    e
dtype: object
```

Python

```python
pd.Series(data, index=[1, 3, 5, 7, 9])
```

Python

```
Out[-]
1    a
3    b
5    c
7    d
9    e
dtype: object
```

Python

```python
pd.Series(data, index=[1, 3, 5, 7, 9])[1:4]  # 묵시적인 인덱스만 따름
```

Python

```
Out[-]
3    b
5    c
7    d
dtype: object
```

Python

```python
pd.Series(data, index=[1, 3, 5, 7, 9]).loc[1:4] # 명시적인 인덱스만 따름
```

Copy to clipboard ••• Python

```
Out[-]
1    a
3    b
dtype: object
```

Python

```python
pd.Series(data, index=[1, 3, 5, 7, 9]).iloc[1:4] # 묵시적인 인덱스만 따름
```

Python

```
Out[-]
3    b
5    c
7    d
dtype: object
```

Python

## 결측값(NaN, None) 처리

1. NaN
   - 자료형이 Float
   - 배열에서 연산할 경우 오류가 발생하지 않지만 결과값이 NaN이 됨
2. None
   - 자료형이 None
   - 배열 연산을 할 경우 오류가 발생
3. 처리방법
   - isnull() : 결측값 확인 (결측 이면 True , 결측이 아니면 False )
   - notnull() : 결측값 확인 (결측 이면 False , 결측이 아니면 True )
   - dropna() : 결측값을 삭제
   - fillna(Num) : 결측을 Num 으로 채워 넣음

```python
data = [1, 2, 3, None]

print(np.array(data))  # None - 수치연산 시 error
print(pd.Series(data)) # NaN - 수치연산 처리가 용이
```

Python

```
Out[-]
[1 2 3 None]

0    1.0
1    2.0
2    3.0
3    NaN
dtype: float64
```

Python

```python
print(pd.Series(data)[3] + 100)
print(pd.Series(data)[3] * 100)
print(pd.Series(data)[3] * 0)
```

Python

```
Out[-]
nan
nan
nan
```

Python

```python
s = pd.Series(data)
print(s)
print(s.sum())
print(s.max())
print(s.min())
```

Python

```
Out[-]
0    1.0
1    2.0
2    3.0
3    NaN
dtype: float64

6.0

3.0

1.0
```

Python

```python
s.isnull()
```

Python

```
Out[-]
0    False
1    False
2    False
3     True
dtype: bool
```

Python

```python
data = [1, 2, 3, None, None, None, None]

s = pd.Series(data)
print(s.isnull())
print(s.isnull().sum())
```

Python

```
Out[-]
0    False
1    False
2    False
3     True
4     True
5     True
6     True
dtype: bool

4
```

Python

```python
print(s.notnull())
print(s.notnull().sum())
```

Python

```
Out[-]
0     True
1     True
2     True
3    False
4    False
5    False
6    False
dtype: bool

3
```

Python

```python
s.dropna()
```

Python

```
Out[-]
0    1.0
1    2.0
2    3.0
dtype: float64
```

Python

```python
s.fillna(0)
```

Python

```
Out[-]
0    1.0
1    2.0
2    3.0
3    0.0
4    0.0
5    0.0
6    0.0
dtype: float64
```

Python

## multiIndex

```python
매출 = {
    '2015년':1000000,
    '2016년':2000000,
    '2017년':3000000,
    '2018년':4000000,
    '2019년':11000000,
    '2020년':30000000,
}
순익 = {
    '2015년':100001,
    '2016년':200001,
    '2017년':300001,
    '2018년':400001,
    '2019년':1100001,
    '2020년':3000001,
}
```

Python

```python
indexOne = list(zip(['매출' for i in range(len(매출.keys()))], 매출.keys()))
indexTwo = list(zip(['순익' for i in range(len(순익.keys()))], 순익.keys()))
index = indexOne + indexTwo
index
```

Python

```
Out[-]
[('매출', '2015년'),
 ('매출', '2016년'),
 ('매출', '2017년'),
 ('매출', '2018년'),
 ('매출', '2019년'),
 ('매출', '2020년'),
 ('순익', '2015년'),
 ('순익', '2016년'),
 ('순익', '2017년'),
 ('순익', '2018년'),
 ('순익', '2019년'),
 ('순익', '2020년')]
```

Python

```python
index = pd.MultiIndex.from_tuples(index)
index
```

Python

```
Out[-]
MultiIndex([('매출', '2015년'),
            ('매출', '2016년'),
            ('매출', '2017년'),
            ('매출', '2018년'),
            ('매출', '2019년'),
            ('매출', '2020년'),
            ('순익', '2015년'),
            ('순익', '2016년'),
            ('순익', '2017년'),
            ('순익', '2018년'),
            ('순익', '2019년'),
            ('순익', '2020년')],
           )
```

Python

```python
값 = list(매출.values()) + list(순익.values()) # 더하기를 하려면 list로 변환해야 한다.

print(매출.values())
print(순익.values())
print(값)
```

Python

```
Out[-]
dict_values([1000000, 2000000, 3000000, 4000000, 11000000, 30000000])
dict_values([100001, 200001, 300001, 400001, 1100001, 3000001])
[1000000, 2000000, 3000000, 4000000, 11000000, 30000000, 100001, 200001, 300001, 400001,
1100001, 3000001]
```

Python

```python
result = pd.Series(값, index=index)
result
```

Python

Copy to clipboard

```
Out[-]
매출  2015년     1000000
    2016년     2000000
    2017년     3000000
    2018년     4000000
    2019년    11000000
    2020년    30000000
순익  2015년      100001
    2016년      200001
    2017년      300001
    2018년      400001
    2019년     1100001
    2020년     3000001
dtype: int64
```

Python

```python
print(result['매출'].sum())
print(result['순익'][-3:].sum())
```

Python

```
Out[-]
51000000
4500003
```

Python

```python
result['순익'][-3:]
```

Python

```
Out[-]
2018년     400001
2019년    1100001
2020년    3000001
dtype: int64
```

Python

## 연산 함수

- add : 더하기 연산 함수
- sub : 빼기 연산 함수
- mul : 곱하기 연산 함수
- floordiv : 나누었을 때 몫을 구하는 함수
- div : 나누기 연산 함수
- mod : 나머지 구하는 연산 함수
- pow : 거듭제곱 연산 함수

```python
s = pd.Series([100, 200, 300, 400, 500])
ss = pd.Series([10, 20, 30, 40, 50])
```

Python

```python
s + 100
```

Python

```
Out[-]
0    200
1    300
2    400
3    500
4    600
dtype: int64
```

Python

```python
s.add(100)
```

Python

```
Out[-]
0    200
1    300
2    400
3    500
4    600
dtype: int64
```

Python

```python
s + ss
```

Python

```
Out[-]
0    110
1    220
2    330
3    440
4    550
dtype: int64
```

Python

```
s.add(ss)
```

Python

```
Out[-]
0    110
1    220
2    330
3    440
4    550
dtype: int64
```

Python

```
s - ss
```

Python

Copy to clipboard

```
Out[-]
0     90
1    180
2    270
3    360
4    450
dtype: int64
```

Python

```
s.sub(ss)
```

Python

```
Out[-]
0     90
1    180
2    270
3    360
4    450
dtype: int64
```

Python

```python
s * ss
```

Python

```
Out[-]
0     1000
1     4000
2     9000
3    16000
4    25000
dtype: int64
```

Python

```python
s.mul(ss)
```

Python

```
Out[-]
0     1000
1     4000
2     9000
3    16000
4    25000
dtype: int64
```

Python

```python
s // ss  # integer형
```

Python

```
Out[-]
0    10
1    10
2    10
3    10
4    10
dtype: int64
```

Python

```python
s / ss # float형
```

Python

```
Out[-]
0    10.0
1    10.0
2    10.0
3    10.0
4    10.0
dtype: float64
```

Python

```python
s.floordiv(ss)
```

Python

```
Out[-]
0    10
1    10
2    10
3    10
4    10
dtype: int64
```

Python

```python
s.div(ss)
```

Python

```
Out[-]
0    10.0
1    10.0
2    10.0
3    10.0
4    10.0
dtype: float64
```

Python

```python
s % ss
```

```
Out[-]
0    0
1    0
2    0
3    0
4    0
dtype: int64
```

```python
s.mod(ss)
```

```
Out[-]
0    0
1    0
2    0
3    0
4    0
dtype: int64
```

```python
s.mod(3)
```

```
Out[-]
0    1
1    2
2    0
3    1
4    2
dtype: int64
```

```
s
```

Python

```
Out[-]
0    100
1    200
2    300
3    400
4    500
dtype: int64
```

Python

```
s ** 3
```

Python

```
Out[-]
0      1000000
1      8000000
2     27000000
3     64000000
4    125000000
dtype: int64
```

Python

```
s.pow(3)
```

Python

```
Out[-]
0      1000000
1      8000000
2     27000000
3     64000000
4    125000000
dtype: int64
```

Python

## 집계 함수

- count : 데이터 개수 구하는 함수
- min : 최소값 구하는 함수
- max : 최대값 구하는 함수
- mean : 평균 구하는 함수
- median : 중앙값 구하는 함수
- std : 표준편차 구하는 함수
- var : 분산 구하는 함수
- mad : 절대 표준편차 구하는 함수
- describe : 기초 통계를 한 번에 볼 수 있는 함수

```python
s.count()
```

Python

```
Out[-]
5
```

Python

```python
print(s.min())
print(s.max())
print(s.mean())
print(s.median())
print(s.sum())
print(s.std())
print(s.var())
print(s.mad())
```

Python

```
Out[-]
100
500
300.0
300.0
1500
158.11388300841898
25000.0
120.0
```

Python

```python
s.describe()
```

Python

```
Out[-]
count      5.000000
mean     300.000000
std      158.113883
min      100.000000
25%      200.000000
50%      300.000000
75%      400.000000
max      500.000000
dtype: float64
```

Python

```python
s.head(3)
```

Python

```
Out[-]
0    100
1    200
2    300
dtype: int64
```

Python

```python
s.tail(3)
```

Python

```
Out[-]
2    300
3    400
4    500
dtype: int64
```

Python

## 데이터 결합

- concat : 데이터 프레임끼리 결합
  - verify_integrity=True일 때, 인덱스의 중복이 존재하면 error 출력
  - ignore_index : 기존의 인덱스를 무시하고 차례대로 인덱스 출력
  - join='inner' : 결합하는 데이터들의 공통 부분만 출력
  - join='outer' : 결합하는 데이터들의 모든 값 출력

※ concatenate : 배열끼리 결합

```python
import numpy as np

a = np.arange(10).reshape(2, 5)
b = np.arange(10).reshape(2, 5)
c = np.arange(10).reshape(2, 5)

a
```

Python

```
Out[-]
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

Plain Text

```python
np.concatenate([a, b, c])
```

Python

```
Out[-]
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

Python

```python
np.concatenate([a, b, c], axis=1)
```

Python

```
Out[-]
array([[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9, 5, 6, 7, 8, 9, 5, 6, 7, 8, 9]])
```

Python

```python
import pandas as pd

a = pd.Series(['A', 'B', 'C', 'D', 'E'], index=range(1, 6))
b = pd.Series(['A', 'B', 'C', 'D', 'E'], index=range(1, 6))
c = pd.Series(['A', 'B', 'C', 'D', 'E'], index=range(1, 6))

a
```

Python

```
Out[-]
1    A
2    B
3    C
4    D
5    E
dtype: object
```

Python

```python
np.concatenate([a, b, c])
```

Python

```
Out[-]
array(['A', 'B', 'C', 'D', 'E', 'A', 'B', 'C', 'D', 'E', 'A', 'B', 'C',
       'D', 'E'], dtype=object)
```

Python

```python
pd.concat([a, b, c])
```

Python

Copy to clipboard

```
Out[-]
1    A
2    B
3    C
4    D
5    E
1    A
2    B
3    C
4    D
5    E
1    A
2    B
3    C
4    D
5    E
dtype: object
```

Python

```python
pd.concat([a, b, c])[5]
```

Python

```
Out[-]
5    E
5    E
5    E
dtype: object
```

Python

```python
pd.concat([a, b, c], verify_integrity=False)
```

Python

```
Out[-]
1    A
2    B
3    C
4    D
5    E
1    A
2    B
3    C
4    D
5    E
1    A
2    B
3    C
4    D
5    E
dtype: object
```

Python

```python
d = pd.concat([a, b, c], verify_integrity=False, ignore_index=True, copy=False, axis=1)
print(type(d))
print(d)
```

Python

```
Out[-]
<class 'pandas.core.frame.DataFrame'>

  0 1 2
1 A A A
2 B B B
3 C C C
4 D D D
5 E E E
```

Python

```python
a.append(b)
```

Python

```
Out[-]
1    A
2    B
3    C
4    D
5    E
1    A
2    B
3    C
4    D
5    E
dtype: object
```

Python

```python
a = pd.Series(['A', 'B', 'C', 'D', 'E', 'F'], index=range(1, 7))
b = pd.Series(['A', 'B', 'C', 'D', 'E'], index=range(1, 6))
c = pd.Series(['A', 'B', 'C', 'D', 'E'], index=range(1, 6))

d = pd.concat([a, b, c], axis=1)
print(type(d))
print(d)
```

Python

```
Out[-]
<class 'pandas.core.frame.DataFrame'>
   0    1    2
1  A    A    A
2  B    B    B
3  C    C    C
4  D    D    D
5  E    E    E
6  F  NaN  NaN
```

Python

```python
pd.concat([a, b, c], axis=1, join='inner')
```

Python

Copy to clipboard

```
Out[-]
  0 1 2
1 A A A
2 B B B
3 C C C
4 D D D
5 E E E
```

Python

```python
pd.concat([a, b, c], axis=1, join='outer')
```

Python

```
Out[-]
   0    1    2
1  A    A    A
2  B    B    B
3  C    C    C
4  D    D    D
5  E    E    E
6  F  NaN  NaN
```

Python

## DataFrame

- 다차원 배열(Series의 특성을 가지고 있는 2차원 배열)
- 가장 기본적인 데이터 구조

```
연차  연도      매출     순익    직원수
1     2015   1000000  100001        1
2     2016   2000000  200001        2
3     2017   3000000  300001        4
4     2018   4000000  400001        8
5     2019   8000000  800001       16
6     2020  16000000 1600001       32
```

Python

```
rawData = {
    '연차':[1, 2, 3, 4, 5, 6],
    '연도':[2015, 2016, 2017, 2018, 2019, 2020],
    '매출':[1000000, 2000000, 3000000, 4000000, 8000000, 16000000],
    '순익':[100001, 200001, 300001, 400001, 800001, 1600001],
    '직원수':[1, 2, 4, 8, 16, 32]
}
pd.DataFrame(rawData)
```

Python

```
Out[-]
   연차   연도      매출      순익    직원수
0  1    2015   1000000   100001    1
1  2    2016   2000000   200001    2
2  3    2017   3000000   300001    4
3  4    2018   4000000   400001    8
4  5    2019   8000000   800001   16
5  6    2020  16000000  1600001   32
```

Python

```python
pd.DataFrame(rawData)['연도']
```

Python

```
Out[-]
0    2015
1    2016
2    2017
3    2018
4    2019
5    2020
Name: 연도, dtype: int64
```

Python

```python
pd.DataFrame(rawData).iloc[0:3]
```

Python

```
Out[-]
  연차   연도      매출     순익   직원수
0  1    2015   1000000  100001   1
1  2    2016   2000000  200001   2
2  3    2017   3000000  300001   4
```

Python

```python
pd.DataFrame(rawData).iloc[-3:]
```

Python

```
Out[-]
  연차   연도      매출      순익    직원수
3  4    2018   4000000   400001   8
4  5    2019   8000000   800001  16
5  6    2020  16000000  1600001  32
```

Python

```python
pd.DataFrame(rawData, columns=['연차','매출','순익','직원수'], index=rawData['연도'])
```

Python

```
Out[-]
     연차  매출       순익    직원수
2015  1  1000000   100001     1
2016  2  2000000   200001     2
2017  3  3000000   300001     4
2018  4  4000000   400001     8
2019  5  8000000   800001    16
2020  6 16000000  1600001    32
```

Python

```python
%%writefile rawData.csv
1, 2, 3, 4, 5, 6, 7
연차,1, 2, 3, 4, 5, 6
연도,2015, 2016, 2017, 2018, 2019, 2020
매출,1000000, 2000000, 3000000, 4000000, 8000000, 16000000
순익,100001, 200001, 300001, 400001, 800001, 1600001
직원수,1, 2, 4, 8, 16, 32
```

Python

```python
pd.read_csv('rawData.csv')
```

Python

```
Out[-]
       1        2        3        4        5        6         7
0   연차        1        2        3        4        5         6
1   연도     2015     2016     2017     2018     2019      2020
2   매출  1000000  2000000  3000000  4000000  8000000  16000000
3   순익   100001   200001   300001   400001   800001   1600001
4  직원수        1        2        4        8       16        32
```

Python

```python
pd.read_csv('rawData.csv').columns
```

Python

Copy to clipboard

```
Out[-]
Index(['1', ' 2', ' 3', ' 4', ' 5', ' 6', ' 7'], dtype='object')
```

Python

```python
pd.read_csv('rawData.csv').index
```

Python

```
Out[-]
RangeIndex(start=0, stop=5, step=1)
```

Python

## DataFrame에 데이터 조작

- np.nan : NaN으로 값을 채움
- drop : 컬럼 삭제
  - inplace = True : drop후 원본에 반영
- pd.to_numeric() : 문자형을 숫자형으로 변환

```python
import pandas as pd

rawData = {
    '연차':[1, 2, 3, 4, 5, 6],
    '연도':[2015, 2016, 2017, 2018, 2019, 2020],
    '매출':[1000000, 2000000, 3000000, 4000000, 8000000, 16000000],
    '순익':[100001, 200001, 300001, 400001, 800001, 1600001],
    '직원수':[1, 2, 4, 8, 16, 32]
}

df = pd.DataFrame(rawData)
df
```

Python

```
Out[-]
   연차   연도      매출       순익   직원수
0   1   2015   1000000   100001    1
1   2   2016   2000000   200001    2
2   3   2017   3000000   300001    4
3   4   2018   4000000   400001    8
4   5   2019   8000000   800001   16
5   6   2020  16000000  1600001   32
```

Python

```python
df['매출']
df.매출
```

Python

```
Out[-]
0     1000000
1     2000000
2     3000000
3     4000000
4     8000000
5    16000000
Name: 매출, dtype: int64
```

Python

```python
df['순이익율'] = (df['순익'] / df['매출']) * 100
df
```

Python

```
Out[-]
   연차    연도      매출        순익    직원수     순이익율
0   1   2015   1000000    100001     1    10.000100
1   2   2016   2000000    200001     2    10.000050
2   3   2017   3000000    300001     4    10.000033
3   4   2018   4000000    400001     8    10.000025
4   5   2019   8000000    800001    16    10.000013
5   6   2020  16000000   1600001    32    10.000006
```

Python

```python
df['test'] = 100
df
```

Python

Copy to clipboard

```
Out[-]
   연차   연도      매출        순익    직원수     순이익율    test
0    1   2015   1000000    100001     1    10.000100    100
1    2   2016   2000000    200001     2    10.000050    100
2    3   2017   3000000    300001     4    10.000033    100
3    4   2018   4000000    400001     8    10.000025    100
4    5   2019   8000000    800001    16    10.000013    100
5    6   2020  16000000   1600001    32    10.000006    100
```

Python

```python
import numpy as np

df['testTwo'] = np.nan
df
```

Python

```
Out[-]
   연차   연도      매출        순익    직원수     순이익율    test  testTwo
0    1   2015   1000000    100001     1    10.000100    100     NaN
1    2   2016   2000000    200001     2    10.000050    100     NaN
2    3   2017   3000000    300001     4    10.000033    100     NaN
3    4   2018   4000000    400001     8    10.000025    100     NaN
4    5   2019   8000000    800001    16    10.000013    100     NaN
5    6   2020  16000000   1600001    32    10.000006    100     NaN
```

Python

```python
df['testThree'] = None
df
```

Python

```
Out[-]
   연차   연도      매출        순익   직원수     순이익율   test  testTwo  testThree
0   1   2015   1000000    100001    1    10.000100   100    NaN       None
1   2   2016   2000000    200001    2    10.000050   100    NaN       None
2   3   2017   3000000    300001    4    10.000033   100    NaN       None
3   4   2018   4000000    400001    8    10.000025   100    NaN       None
4   5   2019   8000000    800001   16    10.000013   100    NaN       None
5   6   2020  16000000   1600001   32    10.000006   100    NaN       None
```

Python

```python
df[['test', 'testTwo', 'testThree']] = 1000
df
```

Python

```
Out[-]
   연차   연도      매출        순익   직원수     순이익율   test  testTwo  testThree
0   1   2015   1000000    100001    1    10.000100   1000   1000      1000
1   2   2016   2000000    200001    2    10.000050   1000   1000      1000
2   3   2017   3000000    300001    4    10.000033   1000   1000      1000
3   4   2018   4000000    400001    8    10.000025   1000   1000      1000
4   5   2019   8000000    800001   16    10.000013   1000   1000      1000
5   6   2020  16000000   1600001   32    10.000006   1000   1000      1000
```

Python

```python
del df['test']
df
```

Python

```
Out[-]
   연차   연도      매출        순익    직원수     순이익율    testTwo  testThree
0    1   2015   1000000    100001     1    10.000100     1000       1000
1    2   2016   2000000    200001     2    10.000050     1000       1000
2    3   2017   3000000    300001     4    10.000033     1000       1000
3    4   2018   4000000    400001     8    10.000025     1000       1000
4    5   2019   8000000    800001    16    10.000013     1000       1000
5    6   2020  16000000   1600001    32    10.000006     1000       1000
```

Python

```python
df.drop(['testTwo'], axis='columns', inplace=True)
df
```

Python

```
Out[-]
   연차   연도      매출        순익    직원수     순이익율    testThree
0    1   2015   1000000    100001     1    10.000100       1000
1    2   2016   2000000    200001     2    10.000050       1000
2    3   2017   3000000    300001     4    10.000033       1000
3    4   2018   4000000    400001     8    10.000025       1000
4    5   2019   8000000    800001    16    10.000013       1000
5    6   2020  16000000   1600001    32    10.000006       1000
```

Python

```python
df.drop(['testThree'], axis='columns', inplace=True)
df
```

Python

```
Out[-]
   연차   연도      매출        순익    직원수     순이익율
0    1   2015   1000000    100001     1    10.000100
1    2   2016   2000000    200001     2    10.000050
2    3   2017   3000000    300001     4    10.000033
3    4   2018   4000000    400001     8    10.000025
4    5   2019   8000000    800001    16    10.000013
5    6   2020  16000000   1600001    32    10.000006
```

Python

```python
df.drop(df.columns[[0, 2]], axis='columns')
```

Python

```
Out[-]
    연도     순익   직원수 순이익율
0  2015   100001     1   10.000100
1  2016   200001     2   10.000050
2  2017   300001     4   10.000033
3  2018   400001     8   10.000025
4  2019   800001    16   10.000013
5  2020  1600001    32   10.000006
```

Python

```
df
```

Python

```
Out[-]
   연차   연도      매출         순익    직원수     순이익율
0    1   2015   1000000     100001     1    10.000100
1    2   2016   2000000     200001     2    10.000050
2    3   2017   3000000     300001     4    10.000033
3    4   2018   4000000     400001     8    10.000025
4    5   2019   8000000     800001    16    10.000013
5    6   2020  16000000    1600001    32    10.000006
```

Python

```
dfTwo = pd.DataFrame(np.array([[7, 2021, 160000000, 16000001, 60]]),
        columns=['연차','연도','매출','순익','직원수']).append(df, ignore_index=True)
# dfTwo.ndim
dfTwo
```

Python

```
Out[-]
   연차   연도      매출         순익    직원수      순이익율
0    7   2021  160000000   16000001    60          NaN
1    1   2015    1000000     100001     1    10.000100
2    2   2016    2000000     200001     2    10.000050
3    3   2017    3000000     300001     4    10.000033
4    4   2018    4000000     400001     8    10.000025
5    5   2019    8000000     800001    16    10.000013
6    6   2020   16000000    1600001    32    10.000006
```

Python

```python
dfTwo.drop([0], inplace=True)
dfTwo
```

Python

```
Out[-]
   연차   연도      매출      순익    직원수 순이익율
1   1   2015   1000000   100001   1    10.000100
2   2   2016   2000000   200001   2    10.000050
3   3   2017   3000000   300001   4    10.000033
4   4   2018   4000000   400001   8    10.000025
5   5   2019   8000000   800001  16    10.000013
6   6   2020  16000000  1600001  32    10.000006
```

Python

```python
df
```

Python

```
Out[-]
   연차   연도      매출      순익    직원수 순이익율
0   1   2015   1000000   100001   1    10.000100
1   2   2016   2000000   200001   2    10.000050
2   3   2017   3000000   300001   4    10.000033
3   4   2018   4000000   400001   8    10.000025
4   5   2019   8000000   800001  16    10.000013
5   6   2020  16000000  1600001  32    10.000006
```

Python

```python
df[df.매출 > 5000000]
```

Python

```
Out[-]
  연차  연도     매출       순익   직원수  순이익율
4  5   2019   8000000   800001   16    10.000013
5  6   2020  16000000  1600001   32    10.000006
```

Python

Copy to clipboard

```python
# df[df.매출 > 5000000, ['순익', '직원수']] Error
df.loc[df.매출 > 5000000, ['순익', '직원수']]
```

Python

```
Out[-]
    순익   직원수
4  800001   16
5 1600001   32
```

Python

```python
df[df.직원수 > 10]['순익'] - 10000
```

Python

```
Out[-]
4     790001
5    1590001
Name: 순익, dtype: int64
```

Python

```python
df['순익'] = df[df.직원수 > 10]['순익'] - 10000
df
```

Python

Copy to clipboard

```
Out[-]
  연차   연도     매출      순익      직원수 순이익율
0   1   2015   1000000     NaN       1    10.000100
1   2   2016   2000000     NaN       2    10.000050
2   3   2017   3000000     NaN       4    10.000033
3   4   2018   4000000     NaN       8    10.000025
4   5   2019   8000000  790001.0    16    10.000013
5   6   2020  16000000 1590001.0    32    10.000006
```

Python

```python
rawData = {
    '연차':[1, 2, 3, 4, 5, 6],
    '연도':[2015, 2016, 2017, 2018, 2019, 2020],
    '매출':[1000000, 2000000, 3000000, 4000000, 8000000, 16000000],
    '순익':[100001, 200001, 300001, 400001, 800001, 1600001],
    '직원수':[1, 2, 4, 8, 16, 32]
}

df = pd.DataFrame(rawData)

df['순익'] = np.where(df['직원수'] > 10, df['순익'] - 10000, df['순익'])
df
```

Python

```
Out[-]
   연차  연도     매출       순익    직원수
0   1   2015   1000000   100001    1
1   2   2016   2000000   200001    2
2   3   2017   3000000   300001    4
3   4   2018   4000000   400001    8
4   5   2019   8000000   790001   16
5   6   2020  16000000  1590001   32
```

Python

```python
df.loc[6] = df.loc[5] * 2
df
```

Python

```
Out[-]
   연차   연도      매출       순익     직원수
0    1   2015   1000000   100001    1
1    2   2016   2000000   200001    2
2    3   2017   3000000   300001    4
3    4   2018   4000000   400001    8
4    5   2019   8000000   790001   16
5    6   2020  16000000  1590001   32
6   12   4040  32000000  3180002   64
```

Python

```python
df['순이익율'] = (df['순익'] / df['매출'])*100
df.loc[6] = df.loc[5] * 2
df
```

Python

```
Out[-]
    연차    연도       매출         순익      직원수     순이익율
0   1.0   2015.0   1000000.0   100001.0   1.0    10.000100
1   2.0   2016.0   2000000.0   200001.0   2.0    10.000050
2   3.0   2017.0   3000000.0   300001.0   4.0    10.000033
3   4.0   2018.0   4000000.0   400001.0   8.0    10.000025
4   5.0   2019.0   8000000.0   790001.0  16.0     9.875012
5   6.0   2020.0  16000000.0  1590001.0  32.0     9.937506
6  12.0   4040.0  32000000.0  3180002.0  64.0    19.875013
```

Python

```python
df['연도'][6] = 2021
df['연차'][6] = 7
df
```

Python

```
Out[-]
   연차    연도       매출         순익      직원수 순이익율
0  1.0  2015.0   1000000.0   100001.0    1.0    10.000100
1  2.0  2016.0   2000000.0   200001.0    2.0    10.000050
2  3.0  2017.0   3000000.0   300001.0    4.0    10.000033
3  4.0  2018.0   4000000.0   400001.0    8.0    10.000025
4  5.0  2019.0   8000000.0   790001.0   16.0     9.875012
5  6.0  2020.0  16000000.0  1590001.0   32.0     9.937506
6  7.0  2021.0  32000000.0  3180002.0   64.0    19.875013
```

Python

```python
df.dtypes
```

Python

```
Out[-]
연차       float64
연도       float64
매출       float64
순익       float64
직원수      float64
순이익율     float64
dtype: object
```

Python

```python
df['연차'] = df['연차'].astype('int')
df['연도'] = df['연도'].astype('int')
df
```

Python

```
Out[-]
   연차  연도       매출         순익       직원수 순이익율
0   1   2015   1000000.0   100001.0    1.0   10.000100
1   2   2016   2000000.0   200001.0    2.0   10.000050
2   3   2017   3000000.0   300001.0    4.0   10.000033
3   4   2018   4000000.0   400001.0    8.0   10.000025
4   5   2019   8000000.0   790001.0   16.0    9.875012
5   6   2020  16000000.0  1590001.0   32.0    9.937506
6   7   2021  32000000.0  3180002.0   64.0   19.875013
```

Python

```python
pd.Series([1, '2', '3', 'hojun', True, 10.1])
```

Python

```
Out[-]
0        1
1        2
2        3
3    hojun
4     True
5     10.1
dtype: object
```

Python

```python
pd.to_numeric(pd.Series([1, '2', '3', 'hojun', True, 10.1]), errors='ignore')
```

Python

```
Out[-]
0        1
1        2
2        3
3    hojun
4     True
5     10.1
dtype: object
```

Python

```python
pd.to_numeric(pd.Series([1, '2', '3', 'hojun', True, 10.1]), errors='coerce') # errors='coerce' : error를 결측치로 만듬
```

Python

```
Out[-]
0     1.0
1     2.0
2     3.0
3     NaN
4     1.0
5    10.1
dtype: float64
```

Python

## MultiIndex

```python
import numpy as np

print(np.random.rand(4, 2)) # 0부터 1사이, 균일 분포, Matrix 생성
print(np.random.randint(10)) # 0부터 9사이, 숫자 1개 생성
print(np.random.randint(10, 20, size=10))
print(np.random.randint(10, 20, size=(3, 5)))
print(np.random.randn(4, 2)) # 가우시안 표준 정규분포, Matrix 생성
print(np.unique([1, 1, 1, 2, 2, 3])) # 중복된 값 제거
print(np.random.choice(10, 5, replace=False))  # 5개만 선택, replace는 중복허락함
```

Python

```
Out[-]
[[0.84378258 0.74200548]
 [0.55837495 0.69908872]
 [0.85818263 0.56343036]
 [0.90647635 0.77013871]]

5

[16 16 16 13 18 14 11 16 10 19]

[[11 14 10 17 13]
 [10 13 14 19 15]
 [14 15 12 12 17]]

[[ 0.67612663 -0.52565421]
 [ 0.70822562  0.52063027]
 [ 0.23706617  0.74834107]
 [-0.16540672 -0.62646096]]

[1 2 3]

[4 1 6 3 9]
```

Python

```python
import pandas as pd

df = pd.DataFrame(np.random.randint(50, 100, size=(4, 3)),
            index=[['1학년', '1학년', '2학년', '2학년'], ['1반', '2반', '1반', '2반']],
            columns=['국', '영', '수'])

df
```

Python

```
Out[-]
          국  영  수
1학년 1반 77  83  60
      2반 94  54  92
2학년 1반 81  70  75
      2반 65  99  52
```

Python

```python
df = pd.DataFrame(np.random.randint(50, 100, size=(4, 3)))

df
```

Python

```
Out[-]
   0   1   2
0 63  66  82
1 65  65  78
2 52  52  88
3 50  73  74
```

Python

```python
df.index
```

Python

```
Out[-]
RangeIndex(start=0, stop=4, step=1)
```

Python

```python
df.index = ['1반', '2반', '1반', '2반']

df
```

Python

```
Out[-]
     0   1   2
1반  63  66  82
2반  65  65  78
1반  52  52  88
2반  50  73  74
```

Python

```python
df.columns
```

Python

```
Out[-]
RangeIndex(start=0, stop=3, step=1)
```

Python

```python
df.columns = ['국', '영', '수']

df
```

Python

```
Out[-]
    국  영  수
1반 63  66  82
2반 65  65  78
1반 52  52  88
2반 50  73  74
```

Python

```python
df.index = [['1학년', '1학년', '2학년', '2학년'], ['1반', '2반', '1반', '2반']]

df
```

Python

```
Out[-]
          국  영  수
1학년 1반 63  66  82
      2반 65  65  78
2학년 1반 52  52  88
      2반 50  73  74
```

Python

```python
df.columns = [['언어', '언어', '수리'],['국', '영', '수']]

df
```

Python

```
Out[-]
          언어     수리
          국  영   수
1학년 1반 63  66  82
      2반 65  65  78
2학년 1반 52  52  88
      2반 50  73  74
```

Python

```python
df['언어']
```

Python

```
Out[-]
          국  영
1학년 1반 63  66
      2반 65  65
2학년 1반 52  52
      2반 50  73
```

Python

```python
df['수리']
```

Python

```
Out[-]
          수
1학년 1반 82
      2반 78
2학년 1반 88
      2반 74
```

Python

```python
df['언어']['국']
```

Python

```
Out[-]
1학년  1반    63
       2반    65
2학년  1반    52
       2반    50
Name: 국, dtype: int32
```

Python

```python
df.loc['1학년']
```

Python

```
Out[-]
    언어     수리
    국  영   수
1반 63  66  82
2반 65  65  78
```

Python

```python
df.loc['1학년', '1반']
```

Python

```
Out[-]
언어  국    63
      영    66
수리  수    82
Name: (1학년, 1반), dtype: int32
```

Python

```python
df.index = [['제주고', '제주고', '제주고', '제주고'], ['1학년', '1학년', '2학년', '2학년'], [
'1반', '2반', '1반', '2반']]

df
```

Python

```
Out[-]
                 언어     수리
                 국  영   수
제주고 1학년  1반  63  66  82
             2반  65  65  78
       2학년  1반  52  52  88
             2반  50  73   7
```

Python

# 3일차 기본 실습

이 장에서 다루는 내용

Data Information
Indexing, Slicing
Assignment
통계치와 결측치 처리, 값의 변경
공분산
상관계수
데이터 출력

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

Python

```python
df = pd.DataFrame(np.random.randint(50, 100, size=(10, 5)),
     index = [['1학년' for i in range(5)]+
              ['2학년' for i in range(5)],
              [str(i)+'반' for i in range(1, 6)] +
              [str(i)+'반' for i in range(1, 6)]],
     columns = ['국', '영', '수', '사', '과'])
df
```

Python

Out[-]

| | | 국 | 영 | 수 | 사 | 과 |
|---|---|---|---|---|---|---|
| 1학년 | 1반 | 57 | 79 | 66 | 96 | 85 |
| | 2반 | 93 | 69 | 90 | 87 | 67 |
| | 3반 | 87 | 81 | 83 | 66 | 79 |
| | 4반 | 80 | 67 | 59 | 69 | 59 |
| | 5반 | 51 | 68 | 88 | 94 | 80 |
| 2학년 | 1반 | 60 | 84 | 89 | 59 | 50 |
| | 2반 | 62 | 93 | 59 | 60 | 73 |
| | 3반 | 84 | 99 | 55 | 90 | 63 |
| | 4반 | 95 | 57 | 68 | 85 | 69 |
| | 5반 | 68 | 61 | 86 | 67 | 66 |

Python

## Data Information

```
df.info()
```

Python

Copy to clipboard •••

```
Out[-]
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 10 entries, ('1학년', '1반') to ('2학년', '5반')
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   국       10 non-null     int32
 1   영       10 non-null     int32
 2   수       10 non-null     int32
 3   사       10 non-null     int32
 4   과       10 non-null     int32
dtypes: int32(5)
memory usage: 308.0+ bytes
```

Python

```
df.dtypes
```

Python

```
Out[-]
국    int32
영    int32
수    int32
사    int32
과    int32
dtype: object
```

Python

```python
df.head()
```

Python

```
Out[-]
          국  영  수  사  과
1학년 1반 57  79  66  96  85
      2반 93  69  90  87  67
      3반 87  81  83  66  79
      4반 80  67  59  69  59
      5반 51  68  88  94  80
```

Python

```python
df.tail(3)
```

Python

```
Out[-]
          국  영  수  사  과
2학년 3반 84  99  55  90  63
      4반 95  57  68  85  69
      5반 68  61  86  67  66
```

Python

```python
df.index
```

```
Out[-]
MultiIndex([('1학년', '1반'),
            ('1학년', '2반'),
            ('1학년', '3반'),
            ('1학년', '4반'),
            ('1학년', '5반'),
            ('2학년', '1반'),
            ('2학년', '2반'),
            ('2학년', '3반'),
            ('2학년', '4반'),
            ('2학년', '5반')],
           )
```

```python
df.values
```

```
Out[-]
array([[57, 79, 66, 96, 85],
       [93, 69, 90, 87, 67],
       [87, 81, 83, 66, 79],
       [80, 67, 59, 69, 59],
       [51, 68, 88, 94, 80],
       [60, 84, 89, 59, 50],
       [62, 93, 59, 60, 73],
       [84, 99, 55, 90, 63],
       [95, 57, 68, 85, 69],
       [68, 61, 86, 67, 66]])
```

```python
df.describe() # 가장 기초적인 통계 분석
```

Python

```
Out[-]
              국         영         수         사         과
count 10.000000 10.000000 10.000000 10.000000 10.000000
 mean 73.700000 75.800000 74.300000 77.300000 69.100000
  std 15.986453 13.725888 14.189589 14.453373 10.556199
  min 51.000000 57.000000 55.000000 59.000000 50.000000
  25% 60.500000 67.250000 60.750000 66.250000 63.750000
  50% 74.000000 74.000000 75.500000 77.000000 68.000000
  75% 86.250000 83.250000 87.500000 89.250000 77.500000
  max 95.000000 99.000000 90.000000 96.000000 85.000000
```

Python

## indexing, slicing

```python
df
```

Python

```
Out[-]
          국  영  수  사  과
1학년 1반 57  79  66  96  85
      2반 93  69  90  87  67
      3반 87  81  83  66  79
      4반 80  67  59  69  59
      5반 51  68  88  94  80
2학년 1반 60  84  89  59  50
      2반 62  93  59  60  73
      3반 84  99  55  90  63
      4반 95  57  68  85  69
      5반 68  61  86  67  66
```

Python

```python
print(df.국)
print(df['국'])  # 같은 결과
```

Python

```
Out[-]
1학년  1반    57
     2반    93
     3반    87
     4반    80
     5반    51
2학년  1반    60
     2반    62
     3반    84
     4반    95
     5반    68
Name: 국, dtype: int32


1학년  1반    57
     2반    93
     3반    87
     4반    80
     5반    51
2학년  1반    60
     2반    62
     3반    84
     4반    95
     5반    68
Name: 국, dtype: int32
```

Python

```python
df[2:4] # 묵시적으로 선언된 인덱스로 slicing
```

Python

```
Out[-]
          국  영  수  사  과
1학년 3반 87  81  83  66  79
      4반 80  67  59  69  59
```

Python

```python
df[['국', '영', '수']]
```

Python

Copy to clipboard

```
Out[-]
          국  영  수
1학년 1반 57  79  66
      2반 93  69  90
      3반 87  81  83
      4반 80  67  59
      5반 51  68  88
2학년 1반 60  84  89
      2반 62  93  59
      3반 84  99  55
      4반 95  57  68
      5반 68  61  86
```

Python

```python
df.loc[:'1학년', ['국', '영', '수']]
```

Python

```
Out[-]
          국  영  수
1학년 1반 57  79  66
      2반 93  69  90
      3반 87  81  83
      4반 80  67  59
      5반 51  68  88
```

Python

```python
df.loc[:'1학년', ['국']]
```

Python

```
Out[-]
            국
1학년 1반 57
      2반 93
      3반 87
      4반 80
      5반 51
```

Python

```python
df.loc[:'1학년', ['국']][:3]
```

Python

```
Out[-]
            국
1학년 1반 57
      2반 93
      3반 87
```

Python

```python
df.loc[:'1학년', ['국']][::-1]
```

Python

```
Out[-]
            국
1학년 5반 51
      4반 80
      3반 87
      2반 93
      1반 57
```

Python

```python
df.iloc[:5, [0, 3]]
```

Python

```
Out[-]
          국  사
1학년 1반 57  96
      2반 93  87
      3반 87  66
      4반 80  69
      5반 51  94
```

Python

```python
df.iloc[-3:, [0, 3]]
```

Python

```
Out[-]
          국  사
2학년 3반 84  90
      4반 95  85
      5반 68  67
```

Python

```python
df.iloc[::2, [0, 3]]
```

Python

```
Out[-]
          국  사
1학년 1반 57  96
      3반 87  66
      5반 51  94
2학년 2반 62  60
      4반 95  85
```

Python

```python
df.iloc[::-1, [0, 3]]
```

Python

```
Out[-]
          국  사
2학년 5반 68  67
      4반 95  85
      3반 84  90
      2반 62  60
      1반 60  59
1학년 5반 51  94
      4반 80  69
      3반 87  66
      2반 93  87
      1반 57  96
```

Python

## Assignment

```python
df
```

Python

```
Out[-]
          국  영  수  사  과
1학년 1반 57  79  66  96  85
      2반 93  69  90  87  67
      3반 87  81  83  66  79
      4반 80  67  59  69  59
      5반 51  68  88  94  80
2학년 1반 60  84  89  59  50
      2반 62  93  59  60  73
      3반 84  99  55  90  63
      4반 95  57  68  85  69
      5반 68  61  86  67  66
```

Python

```python
df_3 = pd.DataFrame(np.random.randint(50, 100, size=(5, 5)),
     index = [['3학년' for i in range(5)],
              [str(i)+'반' for i in range(1, 6)]],
     columns = ['국', '영', '수', '사', '과'])
df_3
```

Python

```
Out[-]
          국  영  수  사  과
3학년 1반 99  74  74  62  94
      2반 55  60  64  84  60
      3반 61  89  61  64  65
      4반 60  51  50  82  56
      5반 62  80  99  76  96
```

Python

```python
df.append(df_3)
```

Python

```
Out[-]
          국  영  수  사  과
1학년 1반 57  79  66  96  85
      2반 93  69  90  87  67
      3반 87  81  83  66  79
      4반 80  67  59  69  59
      5반 51  68  88  94  80
2학년 1반 60  84  89  59  50
      2반 62  93  59  60  73
      3반 84  99  55  90  63
      4반 95  57  68  85  69
      5반 68  61  86  67  66
3학년 1반 99  74  74  62  94
      2반 55  60  64  84  60
      3반 61  89  61  64  65
      4반 60  51  50  82  56
      5반 62  80  99  76  96
```

Python

```python
# for i in df:
#     print(i, df[i])

df['국']
```

Python

```
Out[-]
1학년  1반    57
     2반    93
     3반    87
     4반    80
     5반    51
2학년  1반    60
     2반    62
     3반    84
     4반    95
     5반    68
3학년  1반    99
     2반    55
     3반    61
     4반    60
     5반    62
Name: 국, dtype: int32
```

Python

```python
df['평균'] = (df['국'] + df['영'] + df['수'] + df['사'] + df['과']) / 5
df
```

Python

Out[-]

| | | 국 | 영 | 수 | 사 | 과 | 평균 |
|---|---|---|---|---|---|---|---|
| 1학년 | 1반 | 57 | 79 | 66 | 96 | 85 | 76.6 |
| | 2반 | 93 | 69 | 90 | 87 | 67 | 81.2 |
| | 3반 | 87 | 81 | 83 | 66 | 79 | 79.2 |
| | 4반 | 80 | 67 | 59 | 69 | 59 | 66.8 |
| | 5반 | 51 | 68 | 88 | 94 | 80 | 76.2 |
| 2학년 | 1반 | 60 | 84 | 89 | 59 | 50 | 68.4 |
| | 2반 | 62 | 93 | 59 | 60 | 73 | 69.4 |
| | 3반 | 84 | 99 | 55 | 90 | 63 | 78.2 |
| | 4반 | 95 | 57 | 68 | 85 | 69 | 74.8 |
| | 5반 | 68 | 61 | 86 | 67 | 66 | 69.6 |
| 3학년 | 1반 | 99 | 74 | 74 | 62 | 94 | 80.6 |
| | 2반 | 55 | 60 | 64 | 84 | 60 | 64.6 |
| | 3반 | 61 | 89 | 61 | 64 | 65 | 68.0 |
| | 4반 | 60 | 51 | 50 | 82 | 56 | 59.8 |
| | 5반 | 62 | 80 | 99 | 76 | 96 | 82.6 |

Python

## 통계치와 결측치 처리, 값의 변경

```python
len(df) # 1학년 1반부터 3학년 5반까지 총 15개의 반
```

Python

```
Out[-]
15
```

Python

```python
for i in df:
    print(i)
```

Python

```
Out[-]
국
영
수
사
과
평균
```

Python

```python
df.size
```

Python

```
Out[-]
90
```

Python

```python
df.values # 이중 배열이기 때문에 len(df.values)는 15가 나옴
```

Python

```
Out[-]
array([[57. , 79. , 66. , 96. , 85. , 76.6],
       [93. , 69. , 90. , 87. , 67. , 81.2],
       [87. , 81. , 83. , 66. , 79. , 79.2],
       [80. , 67. , 59. , 69. , 59. , 66.8],
       [51. , 68. , 88. , 94. , 80. , 76.2],
       [60. , 84. , 89. , 59. , 50. , 68.4],
       [62. , 93. , 59. , 60. , 73. , 69.4],
       [84. , 99. , 55. , 90. , 63. , 78.2],
       [95. , 57. , 68. , 85. , 69. , 74.8],
       [68. , 61. , 86. , 67. , 66. , 69.6],
       [99. , 74. , 74. , 62. , 94. , 80.6],
       [55. , 60. , 64. , 84. , 60. , 64.6],
       [61. , 89. , 61. , 64. , 65. , 68. ],
       [60. , 51. , 50. , 82. , 56. , 59.8],
       [62. , 80. , 99. , 76. , 96. , 82.6]])
```

Python

```python
df.count()
```

Python

```
Out[-]
국      15
영      15
수      15
사      15
과      15
평균     15
dtype: int64
```

Python

```python
df.shape
```

Python

```
Out[-]
(15, 6)
```

Python

```python
df.ndim
```

Python

```
Out[-]
2
```

Python

```python
df.mean()
```

Python

```
Out[-]
국      71.600000
영      74.133333
수      72.733333
사      76.066667
과      70.800000
평균     73.066667
dtype: float64
```

Python

```
df
```

Python

Out[-]

| | | 국 | 영 | 수 | 사 | 과 | 평균 |
|---|---|---|---|---|---|---|---|
| 1학년 | 1반 | 57 | 79 | 66 | 96 | 85 | 76.6 |
| | 2반 | 93 | 69 | 90 | 87 | 67 | 81.2 |
| | 3반 | 87 | 81 | 83 | 66 | 79 | 79.2 |
| | 4반 | 80 | 67 | 59 | 69 | 59 | 66.8 |
| | 5반 | 51 | 68 | 88 | 94 | 80 | 76.2 |
| 2학년 | 1반 | 60 | 84 | 89 | 59 | 50 | 68.4 |
| | 2반 | 62 | 93 | 59 | 60 | 73 | 69.4 |
| | 3반 | 84 | 99 | 55 | 90 | 63 | 78.2 |
| | 4반 | 95 | 57 | 68 | 85 | 69 | 74.8 |
| | 5반 | 68 | 61 | 86 | 67 | 66 | 69.6 |
| 3학년 | 1반 | 99 | 74 | 74 | 62 | 94 | 80.6 |
| | 2반 | 55 | 60 | 64 | 84 | 60 | 64.6 |
| | 3반 | 61 | 89 | 61 | 64 | 65 | 68.0 |
| | 4반 | 60 | 51 | 50 | 82 | 56 | 59.8 |
| | 5반 | 62 | 80 | 99 | 76 | 96 | 82.6 |

Python

```python
df.loc['1학년']['수'] = df.loc['1학년']['수'] + 10
df
```

Python

```
Out[-]
          국  영   수  사  과  평균
1학년 1반 57  79  76  96  85  76.6
      2반 93  69  100 87  67  81.2
      3반 87  81  93  66  79  79.2
      4반 80  67  69  69  59  66.8
      5반 51  68  98  94  80  76.2
2학년 1반 60  84  89  59  50  68.4
      2반 62  93  59  60  73  69.4
      3반 84  99  55  90  63  78.2
      4반 95  57  68  85  69  74.8
      5반 68  61  86  67  66  69.6
3학년 1반 99  74  74  62  94  80.6
      2반 55  60  64  84  60  64.6
      3반 61  89  61  64  65  68.0
      4반 60  51  50  82  56  59.8
      5반 62  80  99  76  96  82.6
```

Python

```python
df[df['수'] > 80]
```

Python

```
Out[-]
          국  영   수  사  과  평균
1학년 2반 93  69  100 87  67  81.2
      3반 87  81  93  66  79  79.2
      5반 51  68  98  94  80  76.2
2학년 1반 60  84  89  59  50  68.4
      5반 68  61  86  67  66  69.6
3학년 5반 62  80  99  76  96  82.6
```

Python

```python
df['수'] > 80
```

Python

```
Out[-]
1학년  1반    False
     2반     True
     3반     True
     4반    False
     5반     True
2학년  1반     True
     2반    False
     3반    False
     4반    False
     5반     True
3학년  1반    False
     2반    False
     3반    False
     4반    False
     5반     True
Name: 수, dtype: bool
```

Python

## 공분산 : 2개의 확률변수 상관정도

- 하나의 값이 상승하는 경향을 보일 때, 다른 값도 상승하는 상관관계라면 공분산의 값은 양수
- 하나의 값이 상승하는 경향을 보일 때, 다른 값이 하강하는 경향의 상관관계라면 공분산의 값은 음수

```python
df.cov()
```

Python

```
Out[-]
```

| | 국 | 영 | 수 | 사 | 과 | 평균 |
|---|---|---|---|---|---|---|
| 국 | 264.971429 | -4.585714 | 12.600000 | -23.685714 | 32.414286 | 54.914286 |
| 영 | -4.585714 | 193.838095 | -8.366667 | -48.152381 | 33.028571 | 34.104762 |
| 수 | 12.600000 | -8.366667 | 288.495238 | 6.423810 | 92.157143 | 70.309524 |
| 사 | -23.685714 | -48.152381 | 6.423810 | 166.923810 | 15.300000 | 18.838095 |
| 과 | 32.414286 | 33.028571 | 92.157143 | 15.300000 | 183.885714 | 69.071429 |
| 평균 | 54.914286 | 34.104762 | 70.309524 | 18.838095 | 69.071429 | 47.352381 |

Python

## 상관계수(피어슨 상관계수)

- 두 변수간에 원인과 결과의 인과관계가 있는지에 대한 것은 회귀분석을 통해 인과관계의 방향, 정도와 수학적 모델을 확인

- r이 -1.0과 -0.7 사이이면, 강한 음적 선형관계,
- r이 -0.7과 -0.3 사이이면, 뚜렷한 음적 선형관계,
- r이 -0.3과 -0.1 사이이면, 약한 음적 선형관계,
- r이 -0.1과 +0.1 사이이면, 거의 무시될 수 있는 선형관계,
- r이 +0.1과 +0.3 사이이면, 약한 양적 선형관계,
- r이 +0.3과 +0.7 사이이면, 뚜렷한 양적 선형관계,
- r이 +0.7과 +1.0 사이이면, 강한 양적 선형관계

```python
df['국'].corr(df['사'])
```

Python

```
Out[-]
-0.11262318855527205
```

Python

```python
df['수'].corr(df['과'])
```

Python

```
Out[-]
0.4001157819565991
```

Python

```
df
```

Python

Out[-]

| | | 국 | 영 | 수 | 사 | 과 | 평균 |
|---|---|---|---|---|---|---|---|
| 1학년 | 1반 | 57 | 79 | 76 | 96 | 85 | 76.6 |
| | 2반 | 93 | 69 | 100 | 87 | 67 | 81.2 |
| | 3반 | 87 | 81 | 93 | 66 | 79 | 79.2 |
| | 4반 | 80 | 67 | 69 | 69 | 59 | 66.8 |
| | 5반 | 51 | 68 | 98 | 94 | 80 | 76.2 |
| 2학년 | 1반 | 60 | 84 | 89 | 59 | 50 | 68.4 |
| | 2반 | 62 | 93 | 59 | 60 | 73 | 69.4 |
| | 3반 | 84 | 99 | 55 | 90 | 63 | 78.2 |
| | 4반 | 95 | 57 | 68 | 85 | 69 | 74.8 |
| | 5반 | 68 | 61 | 86 | 67 | 66 | 69.6 |
| 3학년 | 1반 | 99 | 74 | 74 | 62 | 94 | 80.6 |
| | 2반 | 55 | 60 | 64 | 84 | 60 | 64.6 |
| | 3반 | 61 | 89 | 61 | 64 | 65 | 68.0 |
| | 4반 | 60 | 51 | 50 | 82 | 56 | 59.8 |
| | 5반 | 62 | 80 | 99 | 76 | 96 | 82.6 |

Python

```
# df['수']['1학년'] = np.nan
print(df.info()) # int32
2**32//2   # 32비트 int형에는 nan을 입력해도 nan이 들어가지 못함

print(df['수'].dtypes)
df['수'].astype('float')
df['수'] = np.nan
print(df)
```

Python

Copy to clipboard

```
Out[-]
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 15 entries, ('1학년', '1반') to ('3학년', '5반')
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   국       15 non-null     int32
 1   영       15 non-null     int32
 2   수       0 non-null      float64
 3   사       15 non-null     int32
 4   과       15 non-null     int32
 5   평균      15 non-null     float64
dtypes: float64(2), int32(4)
memory usage: 626.0+ bytes

2147483648

int32

          국  영  수  사  과  평균
1학년 1반 57  79  NaN 96  85  76.6
      2반 93  69  NaN 87  67  81.2
      3반 87  81  NaN 66  79  79.2
      4반 80  67  NaN 69  59  66.8
      5반 51  68  NaN 94  80  76.2
2학년 1반 60  84  NaN 59  50  68.4
      2반 62  93  NaN 60  73  69.4
      3반 84  99  NaN 90  63  78.2
      4반 95  57  NaN 85  69  74.8
      5반 68  61  NaN 67  66  69.6
3학년 1반 99  74  NaN 62  94  80.6
      2반 55  60  NaN 84  60  64.6
      3반 61  89  NaN 64  65  68.0
      4반 60  51  NaN 82  56  59.8
      5반 62  80  NaN 76  96  82.6
```

Python

```python
df.isna()
```

Python

Copy to clipboard

Out[-]

| | | 국 | 영 | 수 | 사 | 과 | 평균 |
|---|---|---|---|---|---|---|---|
| 1학년 | 1반 | False | False | True | False | False | False |
| | 2반 | False | False | True | False | False | False |
| | 3반 | False | False | True | False | False | False |
| | 4반 | False | False | True | False | False | False |
| | 5반 | False | False | True | False | False | False |
| 2학년 | 1반 | False | False | True | False | False | False |
| | 2반 | False | False | True | False | False | False |
| | 3반 | False | False | True | False | False | False |
| | 4반 | False | False | True | False | False | False |
| | 5반 | False | False | True | False | False | False |
| 3학년 | 1반 | False | False | True | False | False | False |
| | 2반 | False | False | True | False | False | False |
| | 3반 | False | False | True | False | False | False |
| | 4반 | False | False | True | False | False | False |
| | 5반 | False | False | True | False | False | False |

Python

```python
# max(df.mean())
df.fillna(50)    # 결측치를 다른 값으로 채움
```

Python

```
Out[-]
          국  영    수  사  과  평균
1학년 1반 57  79  50.0  96  85  76.6
      2반 93  69  50.0  87  67  81.2
      3반 87  81  50.0  66  79  79.2
      4반 80  67  50.0  69  59  66.8
      5반 51  68  50.0  94  80  76.2
2학년 1반 60  84  50.0  59  50  68.4
      2반 62  93  50.0  60  73  69.4
      3반 84  99  50.0  90  63  78.2
      4반 95  57  50.0  85  69  74.8
      5반 68  61  50.0  67  66  69.6
3학년 1반 99  74  50.0  62  94  80.6
      2반 55  60  50.0  84  60  64.6
      3반 61  89  50.0  64  65  68.0
      4반 60  51  50.0  82  56  59.8
      5반 62  80  50.0  76  96  82.6
```

Python

## 데이터 출력

- .T : 행과 열의 위치 치환
- apply : 호출 함수

```
df.T
```

Python

```
Out[-]
      1학년                          2학년                          3학년
      1반   2반   3반   4반   5반   1반   2반   3반   4반   5반   1반   2반   3반   4반
5반
  국  57.0  93.0  87.0  80.0  51.0  60.0  62.0  84.0  95.0  68.0  99.0  55.0  61.0  60.0
62.0
  영  79.0  69.0  81.0  67.0  68.0  84.0  93.0  99.0  57.0  61.0  74.0  60.0  89.0  51.0
80.0
  수   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
NaN
  사  96.0  87.0  66.0  69.0  94.0  59.0  60.0  90.0  85.0  67.0  62.0  84.0  64.0  82.0
76.0
  과  85.0  67.0  79.0  59.0  80.0  50.0  73.0  63.0  69.0  66.0  94.0  60.0  65.0  56.0
96.0
평균  76.6  81.2  79.2  66.8  76.2  68.4  69.4  78.2  74.8  69.6  80.6  64.6  68.0  59.8
82.6
```

Python

```
df.apply(max)
```

Python

```
Out[-]
국     99.0
영     99.0
수      NaN
사     96.0
과     96.0
평균    82.6
dtype: float64
```

Python

```python
df.apply(min)
```

Python

```
Out[-]
국      51.0
영      51.0
수       NaN
사      59.0
과      50.0
평균     59.8
dtype: float64
```

Python

```python
df.apply(lambda x: x.max() - x.min())  # 메소드가 들어가야 하므로 lambda 사용
```

Python

```
Out[-]
국      48.0
영      48.0
수       NaN
사      37.0
과      46.0
평균     22.8
dtype: float64
```

Python

```python
df.sort_values('국', ascending=True)
```

Python

```
Out[-]
          국  영   수 사  과  평균
1학년 5반 51  68  NaN 94  80  76.2
3학년 2반 55  60  NaN 84  60  64.6
1학년 1반 57  79  NaN 96  85  76.6
2학년 1반 60  84  NaN 59  50  68.4
3학년 4반 60  51  NaN 82  56  59.8
      3반 61  89  NaN 64  65  68.0
2학년 2반 62  93  NaN 60  73  69.4
3학년 5반 62  80  NaN 76  96  82.6
2학년 5반 68  61  NaN 67  66  69.6
1학년 4반 80  67  NaN 69  59  66.8
2학년 3반 84  99  NaN 90  63  78.2
1학년 3반 87  81  NaN 66  79  79.2
      2반 93  69  NaN 87  67  81.2
2학년 4반 95  57  NaN 85  69  74.8
3학년 1반 99  74  NaN 62  94  80.6
```

Python

```python
#df.to_csv('학교점수.csv')
df.to_csv('학교점수.csv', encoding='utf-8-sig')  # 한글 인코딩
```

Python

# 4일차 데이터 시각화

이 장에서 다루는 내용

EDA

Graph Visualization

Scatter

히스토그램

Basic Attributes

Pie Chart

Bar Chart

Subplot

기타 시각화 그래프

# EDA

## 탐색적 자료 분석

- 데이터를 분석하기 전에 그래프나 통계적인 방법으로 데이터를 직관적으로 바라보는 과정
- 데이터를 있는 그대로 바라보는데 중점을 맞추어 데이터가 가지고 있는 의미를 다양한 각도로 바라보고 이해

## EDA 목적

- 데이터 수집 의사를 결정
- 데이터 유형에 맞는 모델을 선택
- 변수들 사이의 관계를 파악

# Graph Visualization

## Matplotlib를 이용한 시각화

- plotly와 같은 최신 시각화 패키지에 비하면 투박하다고 생각될 수 있으나, 거의 모든 운영체제와 출력형식을 지원하고 있어 아직도 유용한 패키지 중 하나
- 2003년 0.1 출시, 17년이된 패키지
- https://github.com/matplotlib/matplotlib
- https://matplotlib.org/

- % matplotlib inline : jupyter notebook 내에서 output을 보여줌
- ! : 콘솔에서 사용가능한 명령어를 사용가능하게 해줌
- matplotlib 기본 구성 : 그림(figure), 축(axes)
- fig.savefig() : figure에 있는 이미지를 저장

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
```

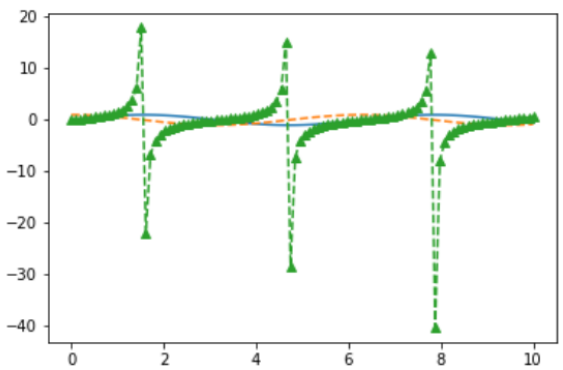
Python

```python
x = np.linspace(0, 10, 100)
fig = plt.figure()

plt.plot(x, np.sin(x), '-')   # 실선으로 sin그래프 그리기
plt.plot(x, np.cos(x), '--')  # 파선으로 cos그래프 그리기
plt.plot(x, np.tan(x), '--^') # 파선에다가 삼각형선으로 tan그래프 그리기
```

Python

Out[-]

[<matplotlib.lines.Line2D at 0x186c8e58f98>]




```python
fig.savefig('test.png')
```

Python

```
# 버전 확인
!python --version
```

Python

```
Out[-]
Python 3.7.3
```

Python

```
!dir
```

Python

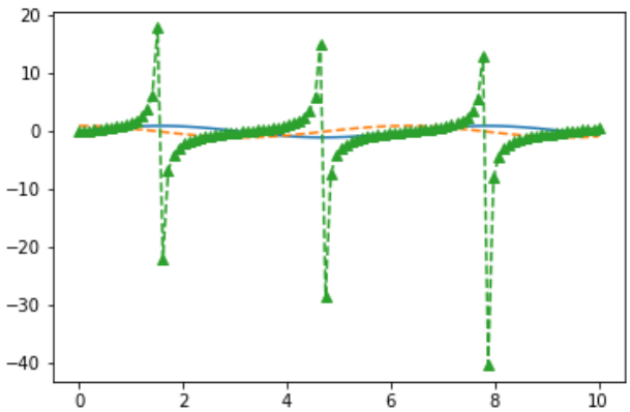
```
Out[-]
C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: CC5E-6766

 C:\Users\leehojun\Google 드라이브\11_1. 콘텐츠 동영상 결과물\006. 데이터분석 강좌\04. 4일차
\최종강의자료 디렉터리

2020-03-30  01:34    <DIR>          .
2020-03-30  01:34    <DIR>          ..
2020-03-30  01:25    <DIR>          .ipynb_checkpoints
2020-03-30  01:34            34,368 004일차_Graph_Visualization.ipynb
2020-03-30  01:34            16,234 test.png
               2개 파일              50,602 바이트
               3개 디렉터리  17,973,571,584 바이트 남음
```

Python

```python
from IPython.display import Image

Image('test.png') # 저장한 'test.png'를 불러오기
```

Python

Out[-]




```python
fig.canvas.get_supported_filetypes() # figure가 지원하는 타입확인
```

Python

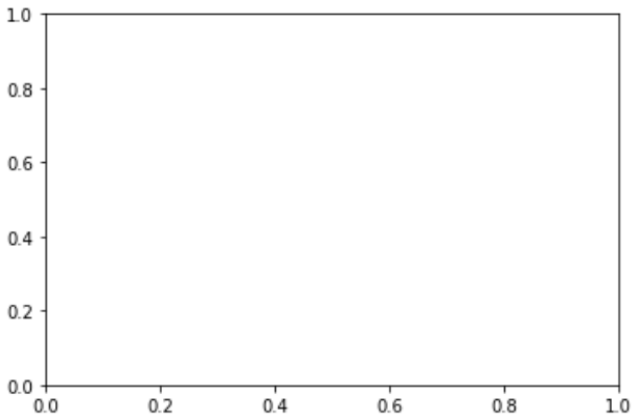
```
Out[-]
{'ps': 'Postscript',
 'eps': 'Encapsulated Postscript',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

Python

```python
fig = plt.figure()
ax = plt.axes()
# 위에 두 코드는 사용하지 않아도 작동됨
# 단, 저장할 때는 필요
plt.show()
```

Python

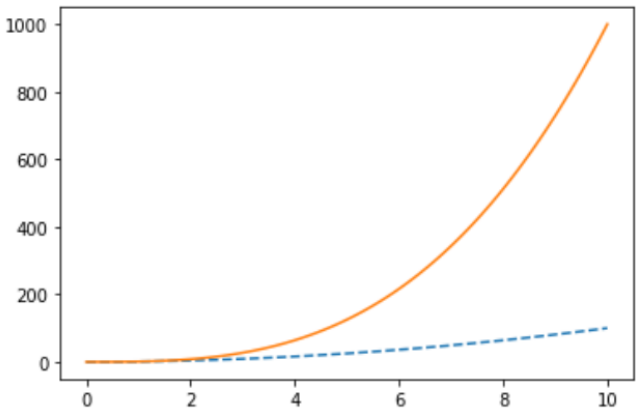
Out[-]




```python
x = np.linspace(0, 10, 100)
y = x ** 2
y_ = x ** 3

plt.plot(x, y, '--')
plt.plot(x, y_) # 여러 개의 그래프를 동시에 출력하려면 plot을 하나 더 사용하면 됨

plt.show()
```

Python

Out[-]



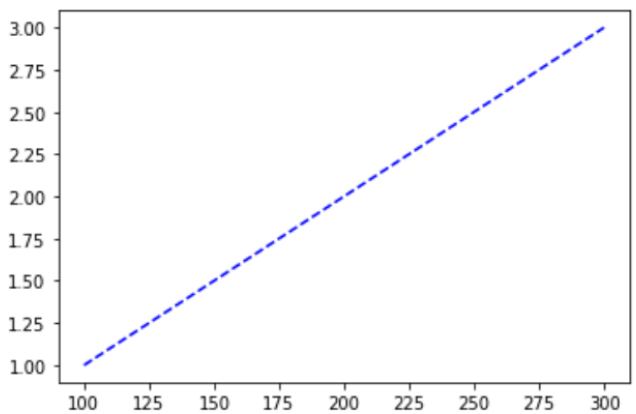
## 선 색상과 스타일

- plt.plot(x, y, color='red') -> red, green, blue 등 색상 이름
- plt.plot(x, y, color='r') -> r, g, b, y, m(자홍), k(검정) 등 색상 이름
- plt.plot(x, y, color='0.2') -> 회색조(0-1사이 값)
- plt.plot(x, y, color='#ff0000') -> 16진수 색상 값
- plt.plot(x, y, linestyle='solid') -> 실선('-')
- plt.plot(x, y, linestyle='dashed') -> 파선('--')
- plt.plot(x, y, linestyle='dashdot') -> 1점 쇄선('-.')
- plt.plot(x, y, linestyle='dotted') -> 점선(':')
- plt.plot(x, y, '--r') -> 빨간색 파선

```python
value = pd.Series([1, 2, 3], [100, 200, 300])

# plt.plot(value, linestyle='dashed', color='blue') # 파선, 파란색
plt.plot(value, '--b')
```

Python

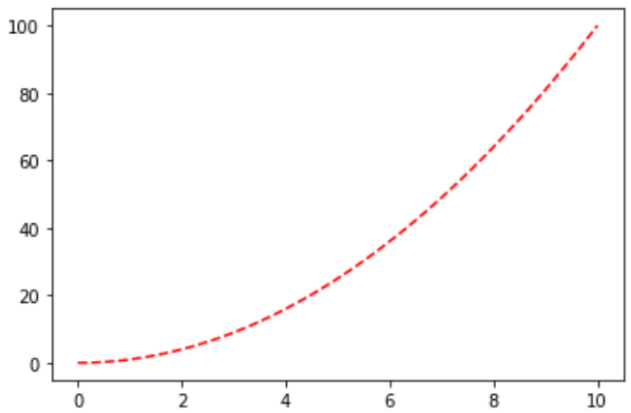
Out[-]



```python
x = np.linspace(0, 10, 100)
y = x ** 2

plt.plot(x, y, '--r')

plt.show()
```

Python

Out[-]



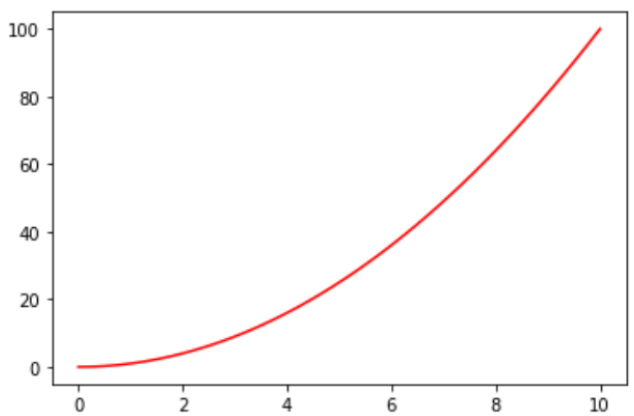
## 경계 표현

- 자동으로 표현
- plt.xlim, plt.ylim으로 상세 조정 가능
- plt.axis([x축 최솟값, x축 최댓값, y축 최솟값, y축 최댓값])로 한 번에 조정 가능 - axes와 다름! 비슷한 키워드 주의!
- plt.axis('keyword') -> tight, equal(균등)

```python
x = np.linspace(0, 10, 100)
y = x ** 2

plt.plot(x, y, 'r')
# plt.xlim(-1, 11)                # x축 -1 ~ 11
# plt.ylim(-10, 110)              # y축 -10 ~ 110
# plt.axis([-1, 11, -10, 110])    # x축 -1 ~ 11, y축 -10 ~ 110
# plt.axis('tight')
# plt.axis('equal')
plt.show()
```

Python

Out[-]



## label과 legend

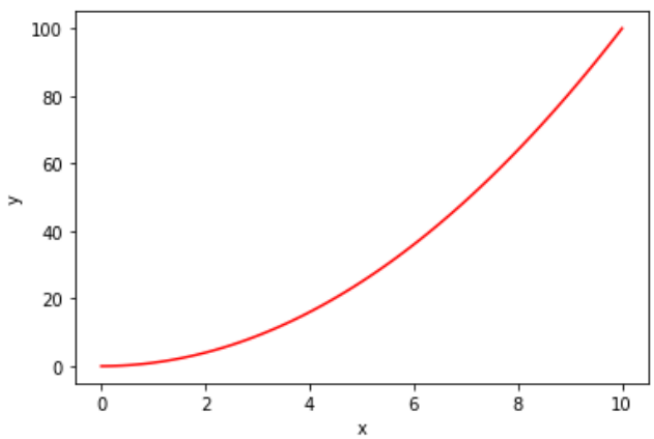
label : x축, y축의 값이 무슨 데이터인지 표시해 주는 방법

legend : 좌표축에 범례를 추가함

```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(0, 10, 100)
y = x ** 2

plt.plot(x, y, 'r')

plt.xlabel('x')
plt.ylabel('y')

plt.show()
```

Python

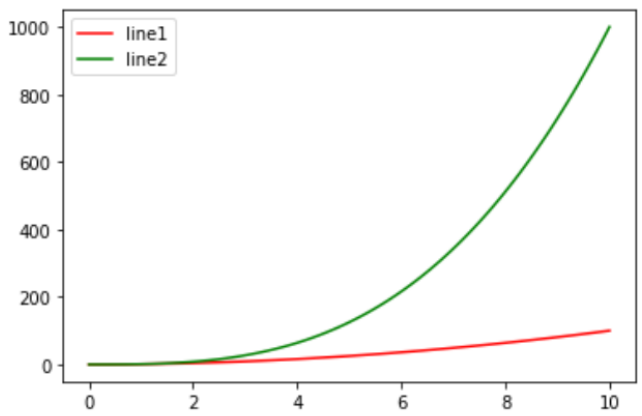
Out[-]



```python
x = np.linspace(0, 10, 100)
y = x ** 2
y_ = x ** 3

plt.plot(x, y, 'r', label='line1')
plt.plot(x, y_, 'g', label='line2')

# plt.xlabel('x')
# plt.ylabel('y')

plt.legend()

plt.show()
```

Python

Out[-]



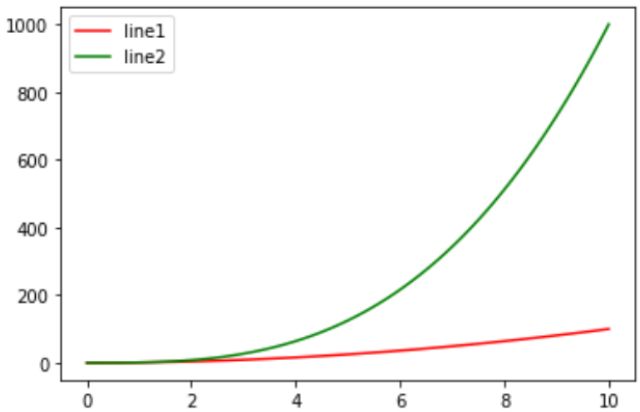
https://matplotlib.org/tutorials/intermediate/legend_guide.html

- 'best' 0
- 'upper right' 1
- 'upper left' 2
- 'lower left' 3
- 'lower right' 4
- 'right' 5
- 'center left' 6
- 'center right' 7
- 'lower center' 8
- 'upper center' 9
- 'center' 10
- loc : 레전드의 위치

```python
x = np.linspace(0, 10, 100)
y = x ** 2
y_ = x ** 3

plt.plot(x, y, 'r', label='line1')
plt.plot(x, y_, 'g', label='line2')

# plt.xlabel('x')
# plt.ylabel('y')

plt.legend(loc=2)

'''
2  9  1
6     5,7
3  8  4
'''

plt.show()
```

Python

Out[-]

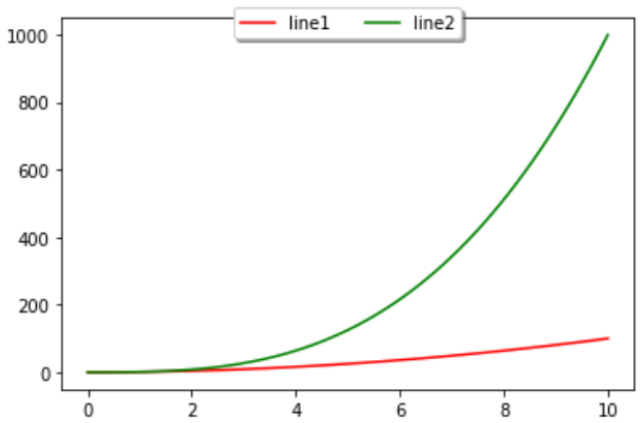


- bbox : figure 밖에 legend위치하게 해주는 것
- fancybox : 모서리를 깎아주는 것

```python
x = np.linspace(0, 10, 100)
y = x ** 2
y_ = x ** 3

plt.plot(x, y, 'r', label='line1')
plt.plot(x, y_, 'g', label='line2')

plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05),
           ncol=2, fancybox=True, shadow=True)
'''
2  9  1
6     5,7
3  8  4
'''

plt.show()
```

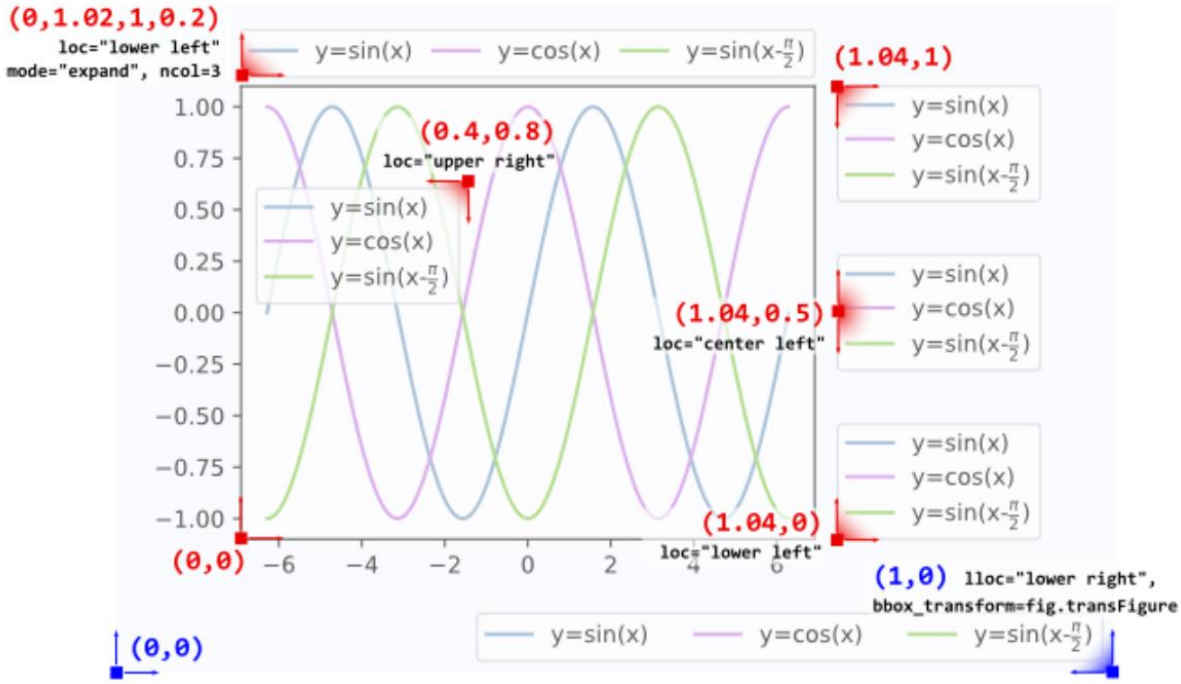
Python

Out[-]



## 바운딩 박스 밖에서 위치 다루기

- 왼쪽 하단이 0, 0
- 오른쪽 하단이 1, 0
- 왼쪽 상단이 0, 1
- 오른쪽 상단이 1, 1




출처 : https://stackoverflow.com/questions/4700614/how-to-put-the-legend-out-of-the-plot

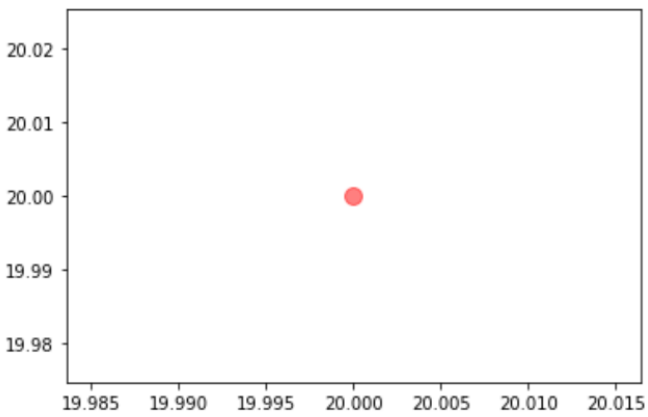
## Scatter(산점도)

- 변수들의 관계, 밀집 위치를 점으로 표현하는 방법
- 양의 상관관계, 음의 상관관계
- 군집 : 점들의 모임

```
plt.scatter(20, 20,
            s=100, c='r',
            alpha=0.5) # (20,20), size가 100, 빨간색, 투명도 0.5인 점  생성

plt.show()
```

Python

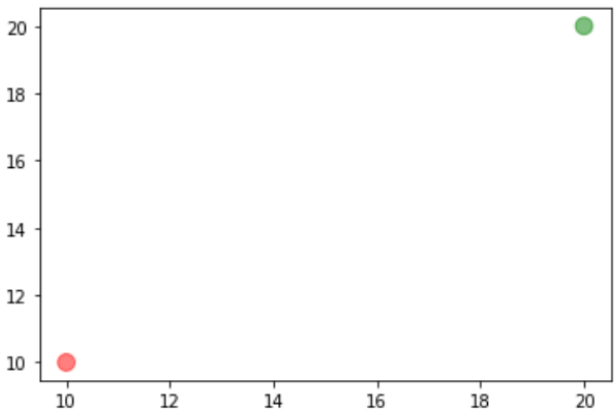
Out[-]



```python
plt.scatter([10, 20], [10, 20],
            s=100, c=['r', 'g'],
            alpha=0.5)

plt.show()
```

Python

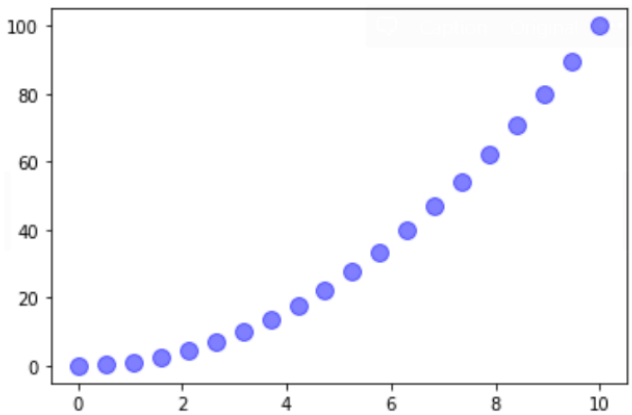
Out[-]



```python
x = np.linspace(0, 10, 20)
y = x ** 2

plt.scatter(x, y,
            s=100, c='b',
            alpha=0.5)

plt.show()
```

Python

Out[-]



```
plt.scatter(x, y,
            s=100, c='b',
            alpha=0.5)

plt.show()
```

Python

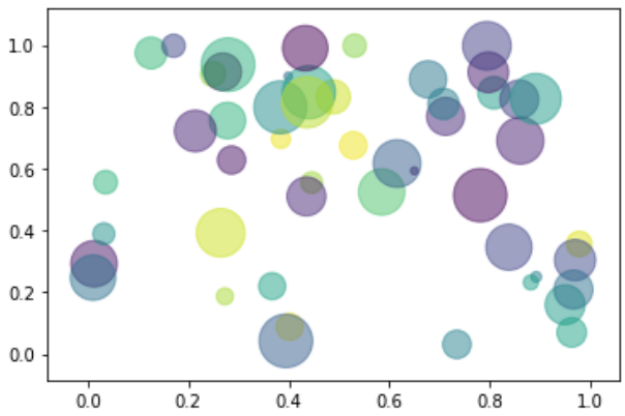
```
Out[-]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-2c8a900ea56a> in <module>
      1 x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
----> 2 y = x ** 2
      3
      4 plt.scatter(x, y, s=100, c='b', alpha=0.5)
      5 plt.show()

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Python

```python
x = np.random.rand(50)
y = np.random.rand(50)
colors = np.random.rand(50)

size = 1000 * np.random.rand(50)

plt.scatter(x, y,
            s=size, c=colors,
            alpha=0.5)

plt.show()
```

Python

Out[-]



# 히스토그램

- 히스토그램(histogram)은 표로 되어 있는 도수 분포를 정보 그림으로 나타낸 것. 더 간단하게 말하면, 도수분포표를 그래프로 나타낸 것
- 막대그래프는 계급 즉 가로를 생각하지 않고 세로의 높이로만 나타냄(출처 : wikipedia)

```python
np.random.rand(4, 2) # 0부터 1사이, 균일 분포, Matrix 생성
np.random.randint(10) # 0부터 9사이, 숫자 1개 생성
np.random.randint(10, 20, size=10)
np.random.randint(10, 20, size=(3, 5))
np.random.randn(4, 2) # 가우시안 표준 정규분포, Matrix 생성
np.unique([1, 1, 1, 2, 2, 3]) # 중복된 값 제거
np.random.choice(10, 5, replace=False)  # 3개만 선택, replace는 중복허락함
```

Python

```python
arr = [np.random.randint(10) for i in range(10)]
arr
```

Python

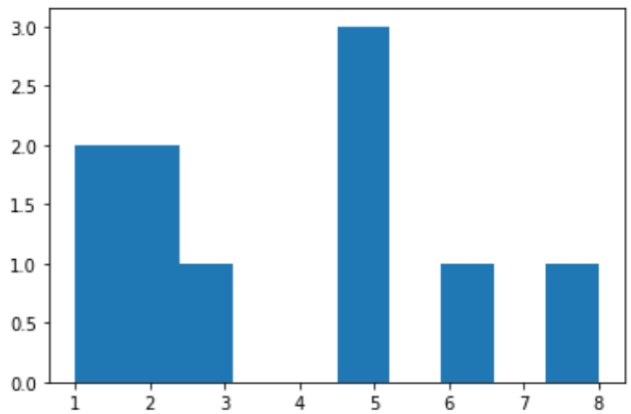
```
Out[-]
[3, 1, 2, 5, 5, 5, 1, 2, 6, 8]
```

Python

```
# plt.grid(True)
plt.hist(arr)

plt.show()
```

Python

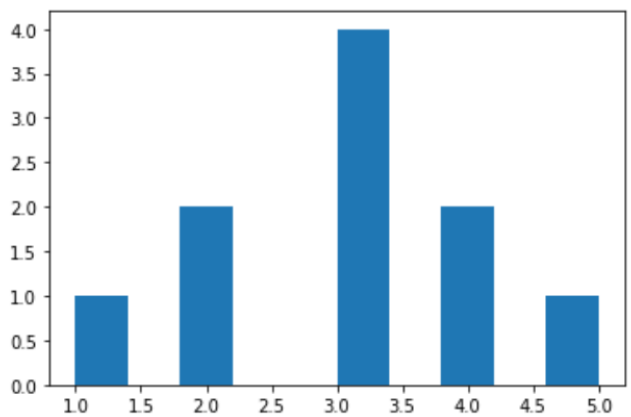
Out[-]




```
arr = [np.random.randint(1, 7) for i in range(10)]

plt.hist(arr)

plt.show()
```

Python

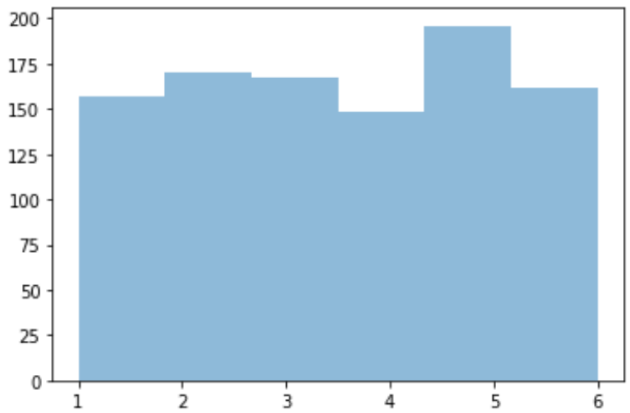
Out[-]



```python
arr = [np.random.randint(1, 7) for i in range(1000)]

plt.hist(arr, bins=6, alpha=0.5)

plt.show()
```

Python

Out[-]



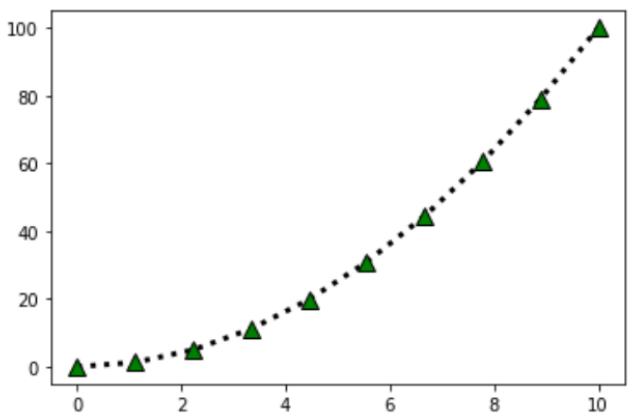
## Basic Attributes

- alpha : 투명도
- logy : Y축 Log scaling
- kind : line, bar, barh, kde
- subplots : 여러개의 plot 그리기
- legend : subplot의 범례 지정
- grid : 그리드 표현(True, False)
- title : 그래프의 제목 지정
- linewidth : 라인 넓이
- color : 색
- linestyle : 실선, 점선, 1점 쇄선 등
- marker : 마커 지정
- markerfacecolor : 마커 색
- markersize : 마커 크기
- xlim, ylim : X축과 Y축의 경계(axis로 한번에 그릴 수 있음)
- 그 외 Attribute : https://matplotlib.org/3.1.1/tutorials/introductory/pyplot.html

```python
x = np.linspace(0, 10, 10)
y = x ** 2

plt.plot(x, y, 'k:',
         linewidth=3,
         marker='^',
         markersize=10,
         markerfacecolor='green')

plt.show()
x
```

Python

Out[-]



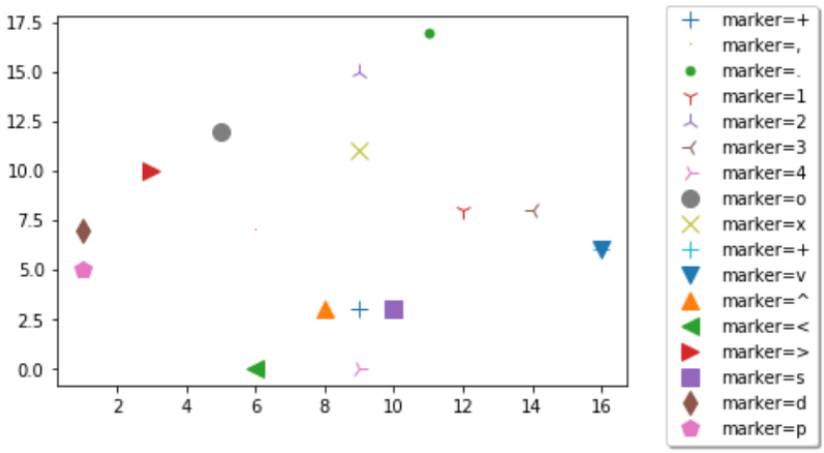
```
array([ 0.        ,  1.11111111,  2.22222222,  3.33333333,  4.44444444,
        5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.        ])
```

Python

```
for m in [ '+', ',', '.', '1', '2', '3', '4', 'o', 'x', '+',
           'v', '^', '<', '>', 's', 'd', 'p']:

    plt.plot(np.random.randint(20),
             np.random.randint(20),
             m,
             markersize=10,
             label=f'marker={m}')

plt.legend(loc='upper center',
           bbox_to_anchor=(1.2, 1.05),
           fancybox=True, shadow=True)

plt.show()
```

Python

Out[-]

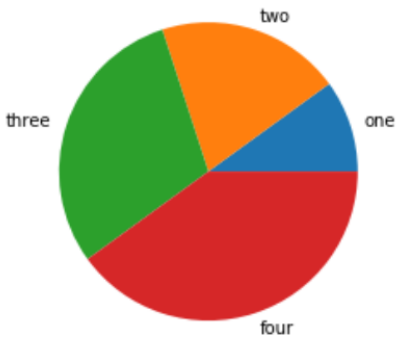


## Pie Chart

```python
labels = ['one', 'two', 'three', 'four']
sizes = [10, 20, 30, 40]

plt.pie(sizes, labels=labels)

plt.show()   # 반시계방향으로 출력
```

Python

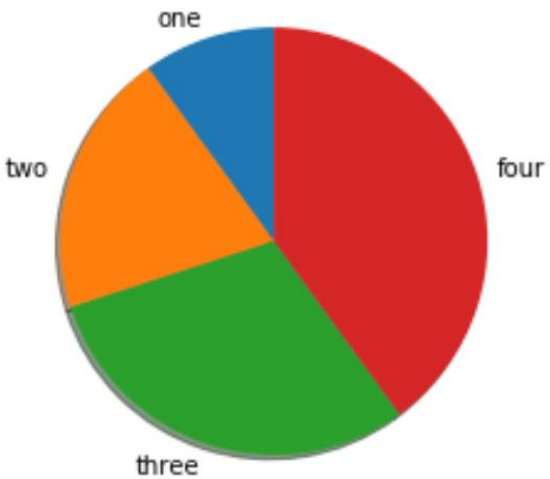
Out[-]



```python
labels = ['one', 'two', 'three', 'four']
sizes = [10, 20, 30, 40]

plt.pie(sizes, labels=labels,
        shadow=True,
        startangle=90) # startangle 90: 12시 방향에서 시작

plt.show()
```

Python

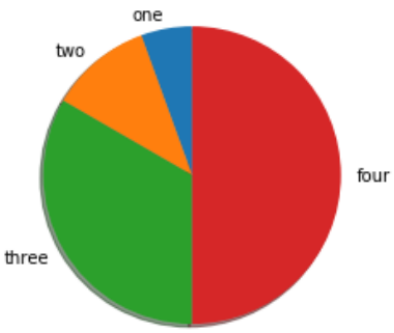
Out[-]



```python
labels = ['one', 'two', 'three', 'four']
sizes = [10, 20, 60, 90]

plt.pie(sizes,
        labels=labels,
        shadow=True, startangle=90)

plt.show()
```

Python

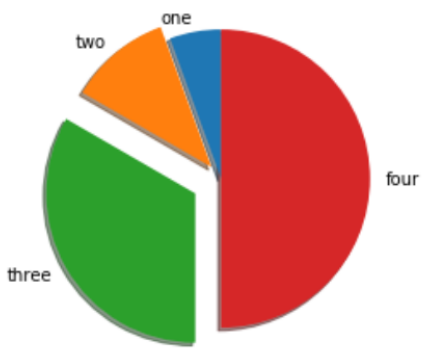
Out[-]



```
labels = ['one', 'two', 'three', 'four']

sizes = [10, 20, 60, 90]
explode = (0, 0.1, 0.2, 0) # 어떤 특정값을 pie chart에서 띄어서 보여주고 싶을 때 사용

plt.pie(sizes,
        labels=labels,
        shadow=True,
        startangle=90, explode=explode)

plt.show()
```

Python

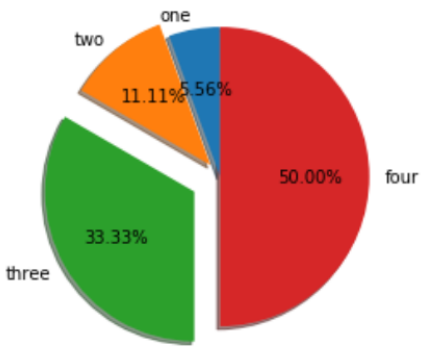
Out[-]



```
labels = ['one', 'two', 'three', 'four']
sizes = [10, 20, 60, 90]
explode = (0, 0.1, 0.2, 0)

plt.pie(sizes, labels=labels,
        shadow=True, startangle=90,
        explode=explode, autopct='%1.2f%%')
                                                    # atuopct : %
 출력
plt.show()
```

Python

Out[-]

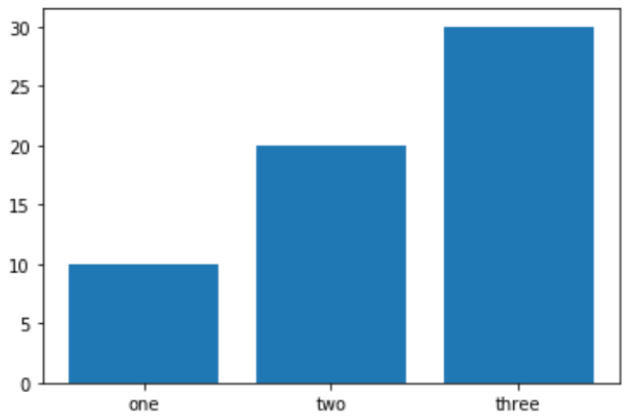


## Bar Chart

```
plt.bar(['one', 'two', 'three'], [10, 20, 30])
plt.show()
```

Python

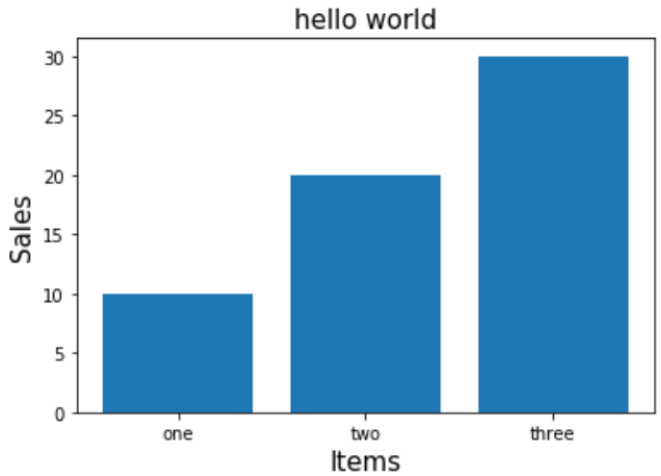
Out[-]




```
plt.bar(['one', 'two', 'three'], [10, 20, 30])

plt.title('hello world', fontsize=15)
plt.xlabel('Items', fontsize=15)
plt.ylabel('Sales', fontsize=15)

plt.show()
```

Python

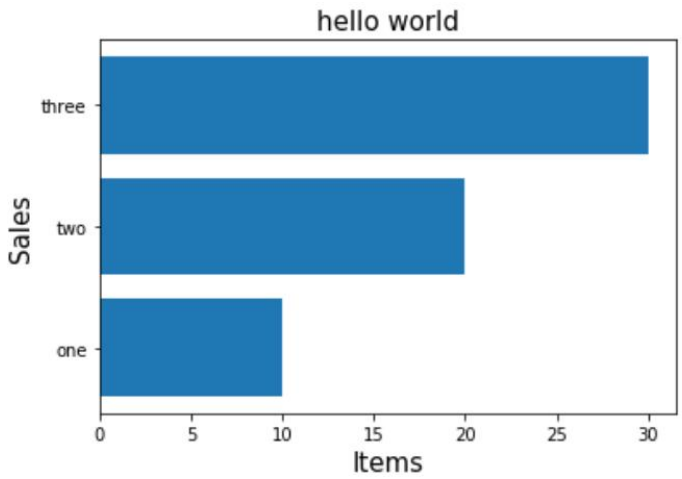
Out[-]



```python
# barh : 가로 막대 그래프 그리기
plt.barh(['one', 'two', 'three'], [10, 20, 30])

plt.title('hello world', fontsize=15)
plt.xlabel('Items', fontsize=15)
plt.ylabel('Sales', fontsize=15)

plt.show()
```

Python

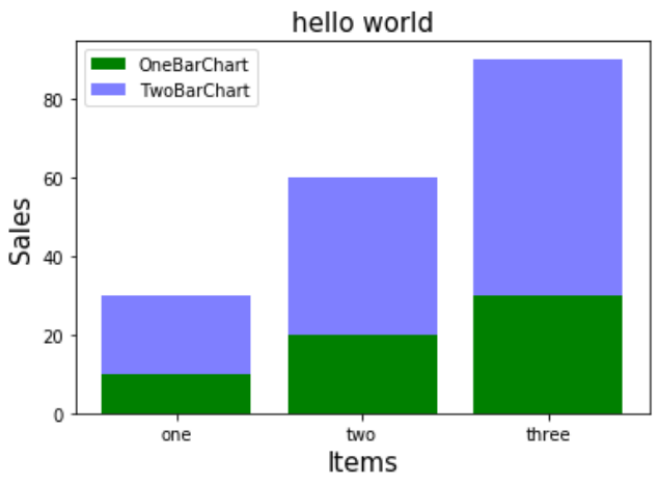
Out[-]



```python
plt.bar(['one', 'two', 'three'],
        [10, 20, 30],
        color='g')

plt.bar(['one', 'two', 'three'],
        [20, 40, 60],
        color='b',
        alpha=0.5,
        bottom=[10, 20, 30])

plt.title('hello world', fontsize=15)
plt.xlabel('Items', fontsize=15)
plt.ylabel('Sales', fontsize=15)

plt.legend(['OneBarChart', 'TwoBarChart'])

plt.show()
```

Python

Out[-]



## subplot

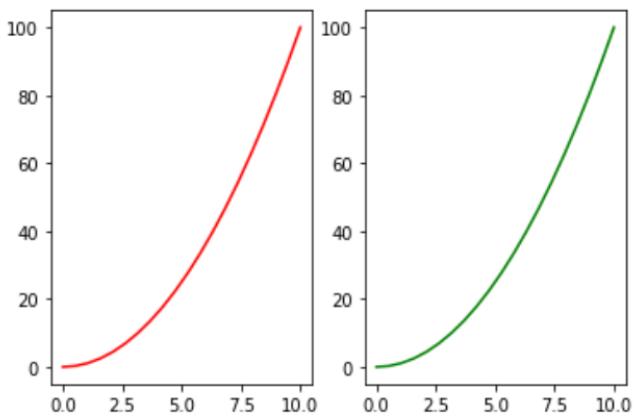
```python
x = np.linspace(0, 10, 20)
y = x ** 2
```

Python

```python
plt.subplot(121)  # row 1 col 2 중 첫번째에 출력
plt.plot(x, y, 'r')

plt.subplot(122)  # row 1 col 2 중 두번째에 출력
plt.plot(x, y, 'g')

plt.show()
```

Python

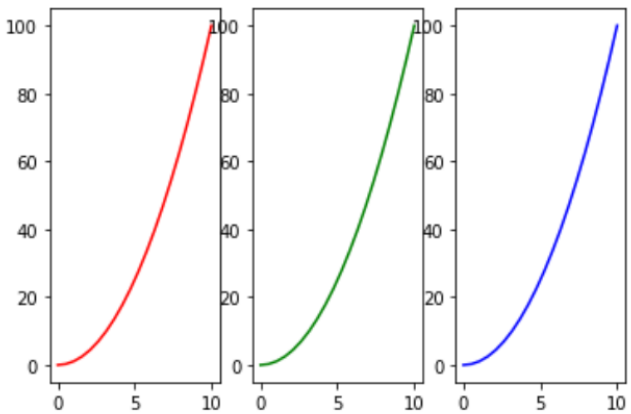
Out[-]



```python
plt.subplot(131)
plt.plot(x, y, 'r')

plt.subplot(132)
plt.plot(x, y, 'g')

plt.subplot(133)
plt.plot(x, y, 'b')

plt.show()
```

Python

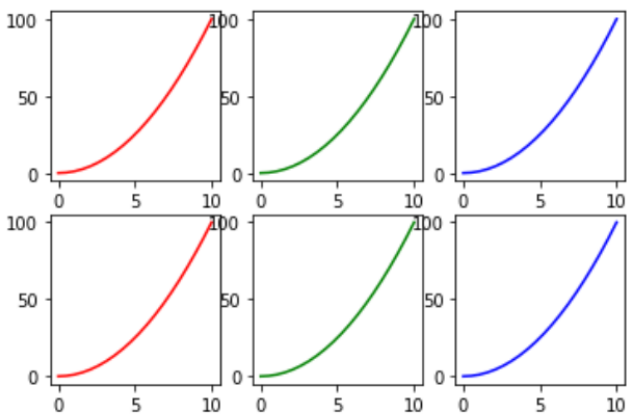
Out[-]



```python
plt.subplot(231)
plt.plot(x, y, 'r')

plt.subplot(232)
plt.plot(x, y, 'g')

plt.subplot(233)
plt.plot(x, y, 'b')

plt.subplot(234)
plt.plot(x, y, 'r')

plt.subplot(235)
plt.plot(x, y, 'g')

plt.subplot(236)
plt.plot(x, y, 'b')

plt.show()
```

Python

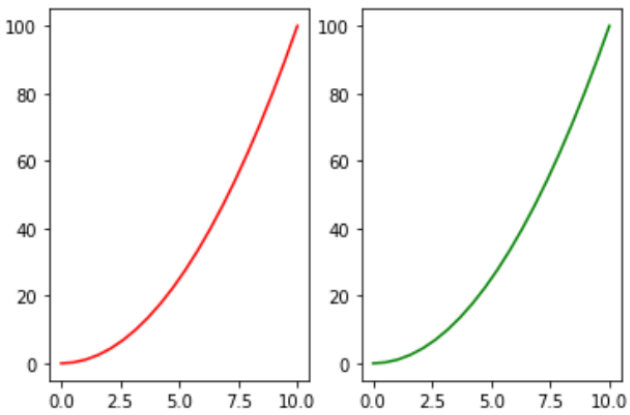
Out[-]



```python
plt.subplot(1,2,1) # , 안쓴거랑 같은 효과
plt.plot(x, y, 'r')

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

plt.show()
```

Python

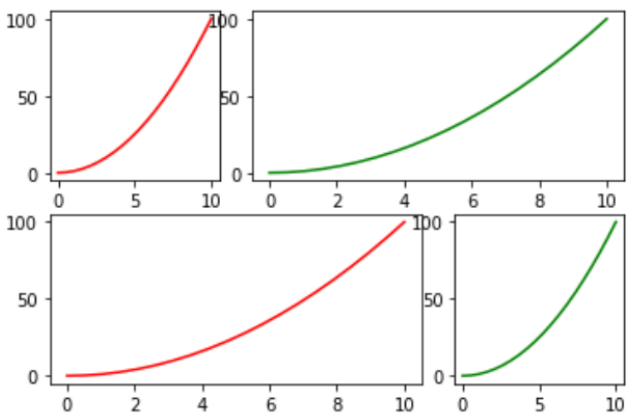
Out[-]



```python
grid = plt.GridSpec(2, 3)   # grid를 이용하여 전체 figure을 나눔

plt.subplot(grid[0, 0])     # gird[행, 열]
plt.plot(x, y, 'r')

plt.subplot(grid[0, 1:])
plt.plot(x, y, 'g')

plt.subplot(grid[1, :2])
plt.plot(x, y, 'r')

plt.subplot(grid[1, 2])
plt.plot(x, y, 'g')

plt.show()
```

Python

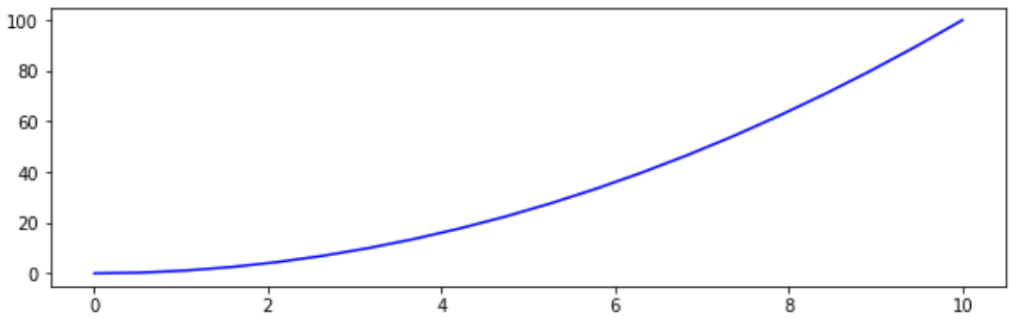
Out[-]



```python
plt.figure(figsize=(10, 3))   # size 조절

plt.plot(x, y, 'b')

plt.show()
```

Python

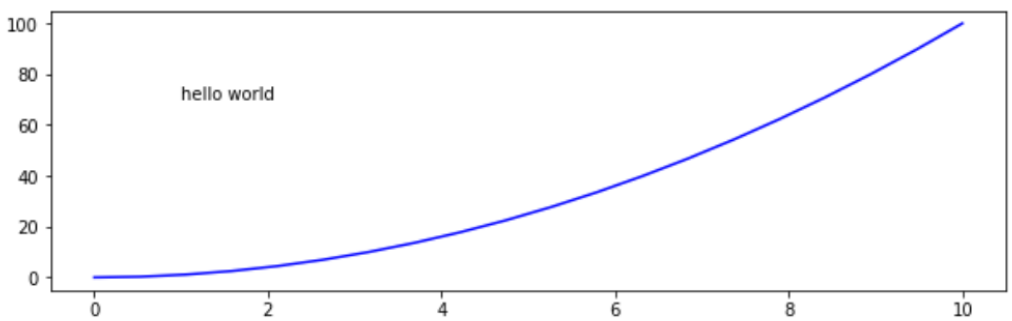
Out[-]




```python
plt.figure(figsize=(10, 3))
plt.plot(x, y, 'b')

plt.text(1, 70, 'hello world')  # (1, 70) 지점에 글씨 출력

plt.show()
```

Python

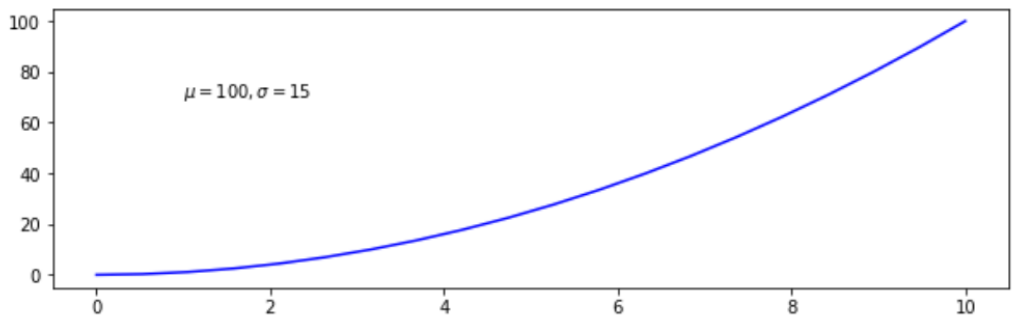
Out[-]



```python
plt.figure(figsize=(10, 3))
plt.plot(x, y, 'b')

plt.text(1, 70, f'$\mu=100, \sigma=15$')

plt.show()
```

Python

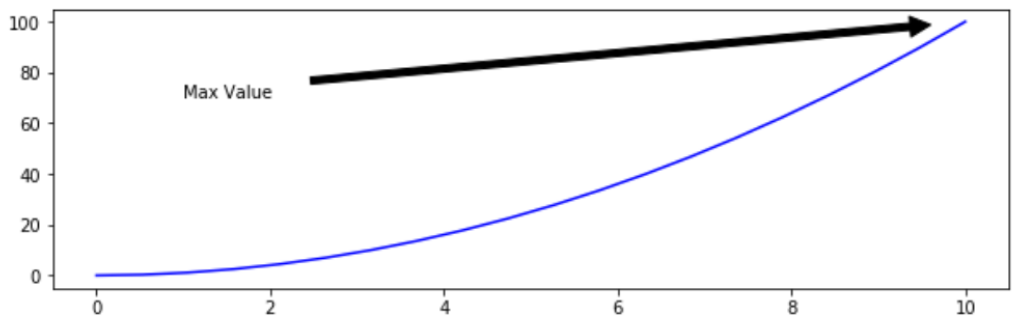
Out[-]




```python
plt.figure(figsize=(10, 3))
plt.plot(x, y, 'b')

plt.annotate('Max Value', xy=(10, 100), xytext=(1, 70),
             arrowprops=dict(facecolor='black', shrink=0.05))
plt.show()
```

Python

Out[-]

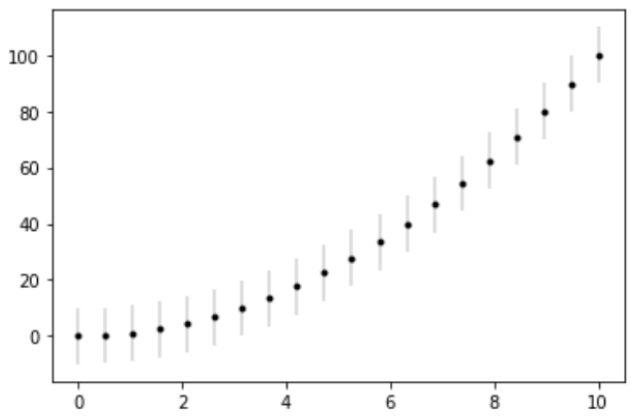


## 기타 시각화 그래프(공식 홈페이지 튜토리얼 소개)

```python
# 오차 막대 : 표준편차의 범위를 나타냄

x = np.linspace(0, 10, 20)
y = x ** 2
dy = 10

plt.errorbar(x, y, yerr=dy, fmt='.k', ecolor='lightgray')

plt.show()
```

Python

Out[-]

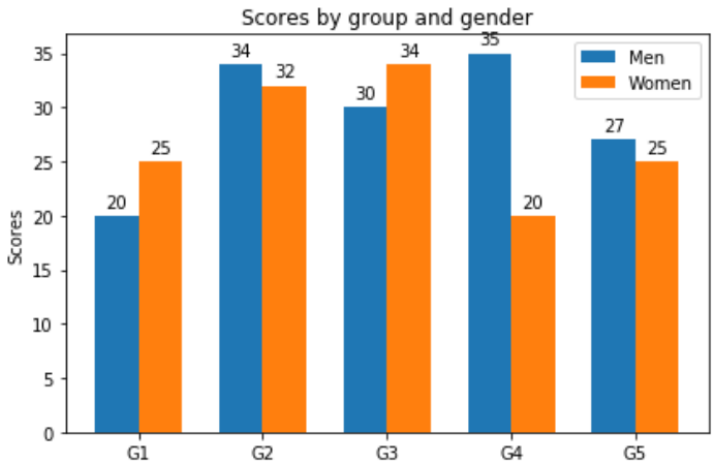



```python
# labels 있는 그룹화된 bar 차트

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


labels = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [20, 34, 30, 35, 27]
women_means = [25, 32, 34, 20, 25]
x = np.arange(len(labels))  # label 위치
width = 0.35  # bar의 넓이
```

```python
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2,
                men_means, width, label='Men')

rects2 = ax.bar(x + width/2,
                women_means, width, label='Women')

# label, 제목 및 사용자 정의 x축 눈금 label 등에 대학 텍스트 추가
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks(x)
ax.set_xticklabels(labels)

ax.legend()

def autolabel(rects): # 그래프에 수치를 나타내주는 코드
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 텍스트를 3칸 위로 띄우기
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)
fig.tight_layout()

plt.show()
```

Python

Out[-]

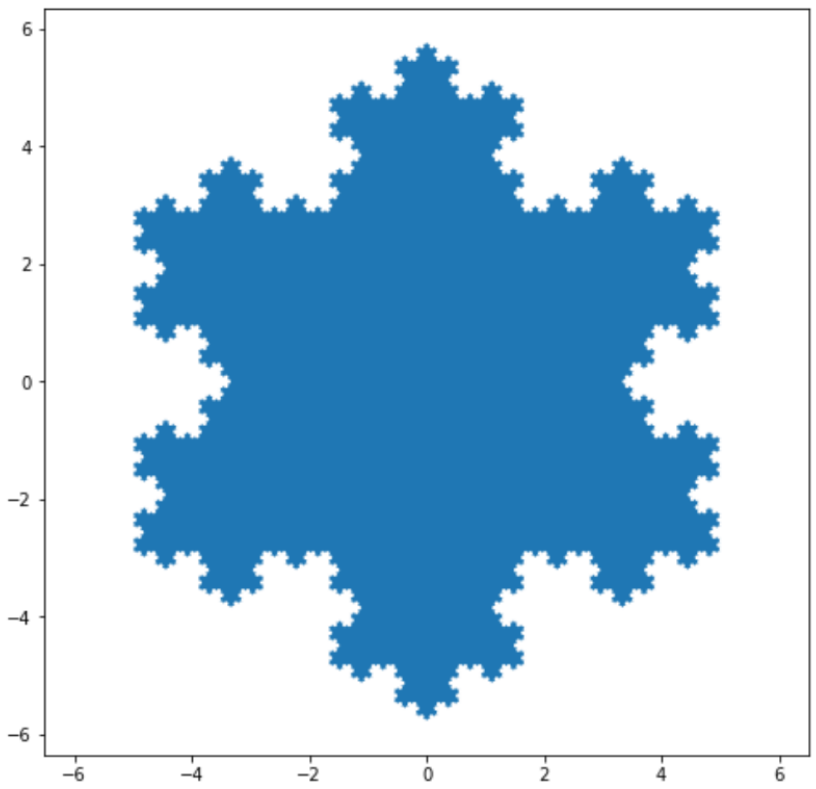


```python
# Filled polygon
import numpy as np
import matplotlib.pyplot as plt

def koch_snowflake(order, scale=10):
    """
    Koch 곡선의 점 좌표 x, y의 두개 list를 return

    Arguments
    ---------
    order : int
        The recursion depth.
    scale : float
       Koch 곡선의 범위(기본 삼각형의 가장자리 길)
    """
    def _koch_snowflake_complex(order):
        if order == 0:
            # 초기 삼각형
            angles = np.array([0, 120, 240]) + 90
            return scale / np.sqrt(3) * np.exp(np.deg2rad(angles) * 1j)
        else:
            ZR = 0.5 - 0.5j * np.sqrt(3) / 3

            p1 = _koch_snowflake_complex(order - 1)  # 시작점
            p2 = np.roll(p1, shift=-1)  # 마침점
            dp = p2 - p1  # 연결 벡터

            new_points = np.empty(len(p1) * 4, dtype=np.complex128)
            new_points[::4] = p1
            new_points[1::4] = p1 + dp / 3
            new_points[2::4] = p1 + dp * ZR
            new_points[3::4] = p1 + dp / 3 * 2
            return new_points

    points = _koch_snowflake_complex(order)
    x, y = points.real, points.imag
    return x, y

x, y = koch_snowflake(order=5)
```

```python
plt.figure(figsize=(8, 8))
plt.axis('equal')
plt.fill(x, y)

plt.show()
```

Python

Out[-]

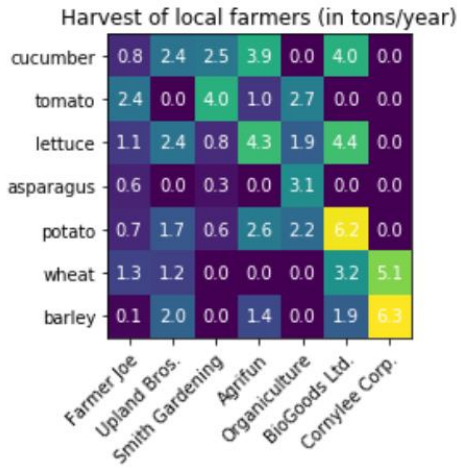


```
# 히트맵(heatmaps) 만들기
# 상관도 분석할 때 가장 많이 쓰임

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
# sphinx_gallery_thumbnail_number = 2

vegetables = ["cucumber", "tomato", "lettuce", "asparagus",
              "potato", "wheat", "barley"]
farmers = ["Farmer Joe", "Upland Bros.", "Smith Gardening",
           "Agrifun", "Organiculture", "BioGoods Ltd.", "Cornylee Corp."]

harvest = np.array([[0.8, 2.4, 2.5, 3.9, 0.0, 4.0, 0.0],
                    [2.4, 0.0, 4.0, 1.0, 2.7, 0.0, 0.0],
                    [1.1, 2.4, 0.8, 4.3, 1.9, 4.4, 0.0],
                    [0.6, 0.0, 0.3, 0.0, 3.1, 0.0, 0.0],
                    [0.7, 1.7, 0.6, 2.6, 2.2, 6.2, 0.0],
                    [1.3, 1.2, 0.0, 0.0, 0.0, 3.2, 5.1],
                    [0.1, 2.0, 0.0, 1.4, 0.0, 1.9, 6.3]])


fig, ax = plt.subplots()
im = ax.imshow(harvest)

# 축에 표시되는 값을 수치로 넣기 (문자열을 정수 리스트로)
ax.set_xticks(np.arange(len(farmers)))
ax.set_yticks(np.arange(len(vegetables)))

# 다시 이름으로 넣어주기
ax.set_xticklabels(farmers)
ax.set_yticklabels(vegetables)

# tick 라벨 각도를 돌리고 조정하기
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")
```

```python
# 수치 표시하기
for i in range(len(vegetables)):
    for j in range(len(farmers)):
        text = ax.text(j, i, harvest[i, j],
                       ha="center", va="center", color="w")

ax.set_title("Harvest of local farmers (in tons/year)")
fig.tight_layout()
plt.show()
```

Python

Out[-]

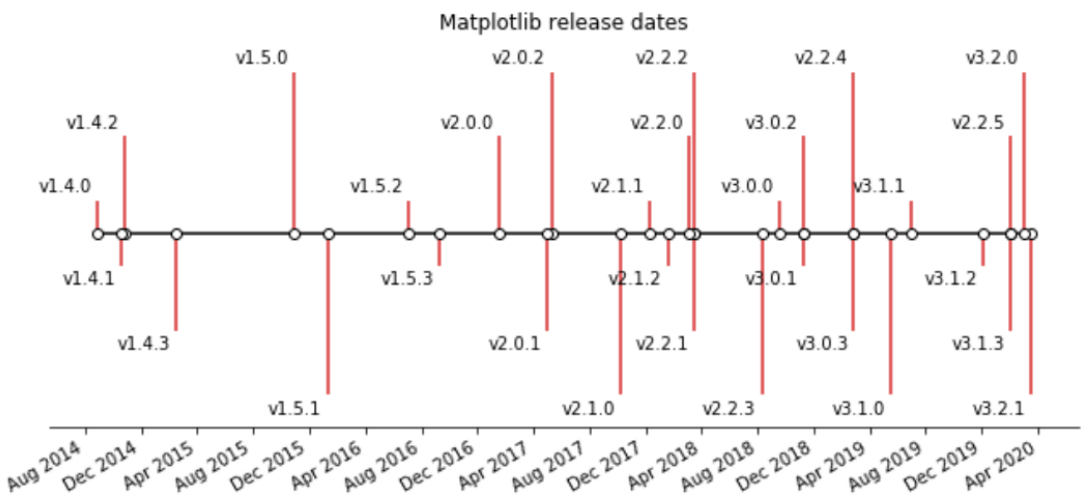


```
# 라인, 날짜 그리고 텍스트가 있는 타임라인 생성
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.dates as mdates
from datetime import datetime

try:
    # Matplotlib 목록과 해당 날짜 가져오기
    # 출처 : https://api.github.com/repos/matplotlib/matplotlib/releases
    import urllib.request
    import json

    url = 'https://api.github.com/repos/matplotlib/matplotlib/releases'
    url += '?per_page=100'
    data = json.loads(urllib.request.urlopen(url, timeout=.4).read().decode())

    dates = []
    names = []
    for item in data:
        if 'rc' not in item['tag_name'] and 'b' not in item['tag_name']:
            dates.append(item['published_at'].split("T")[0])
            names.append(item['tag_name'])
    # Convert date strings (e.g. 2014-10-18) to datetime
    dates = [datetime.strptime(d, "%Y-%m-%d") for d in dates]

except Exception:
    # 위에 것이 실패할 경우, 예를 들어 인터넷 연결이 되지 않은 경우
    # 대비책으로 다음 리스트를 사용해라
    names = ['v2.2.4', 'v3.0.3', 'v3.0.2', 'v3.0.1', 'v3.0.0', 'v2.2.3',
             'v2.2.2', 'v2.2.1', 'v2.2.0', 'v2.1.2', 'v2.1.1', 'v2.1.0',
             'v2.0.2', 'v2.0.1', 'v2.0.0', 'v1.5.3', 'v1.5.2', 'v1.5.1',
             'v1.5.0', 'v1.4.3', 'v1.4.2', 'v1.4.1', 'v1.4.0']

    dates = ['2019-02-26', '2019-02-26', '2018-11-10', '2018-11-10',
             '2018-09-18', '2018-08-10', '2018-03-17', '2018-03-16',
             '2018-03-06', '2018-01-18', '2017-12-10', '2017-10-07',
             '2017-05-10', '2017-05-02', '2017-01-17', '2016-09-09',
             '2016-07-03', '2016-01-10', '2015-10-29', '2015-02-16',
             '2014-10-26', '2014-10-18', '2014-08-26']

    # 날짜 문자열(ex. 2014-10-18)을 datetime으로 변환
    dates = [datetime.strptime(d, "%Y-%m-%d") for d in dates]

# 최적의 levels 선택
levels = np.tile([-5, 5, -3, 3, -1, 1],
                 int(np.ceil(len(dates)/6)))[:len(dates)]
```

```python
# figure, plot, 날짜에 따라서 줄기 그림 그리기
fig, ax = plt.subplots(figsize=(8.8, 4), constrained_layout=True)
ax.set(title="Matplotlib release dates")

markerline, stemline, baseline = ax.stem(dates, levels,
                                         linefmt="C3-", basefmt="k-",
                                         use_line_collection=True)

plt.setp(markerline, mec="k", mfc="w", zorder=3)

# y 데이터를 0으로 교체하여 마커를 기준선으로 이동
markerline.set_ydata(np.zeros(len(dates)))

# annotate lines
vert = np.array(['top', 'bottom'])[(levels > 0).astype(int)]
for d, l, r, va in zip(dates, levels, names, vert):
    ax.annotate(r, xy=(d, l), xytext=(-3, np.sign(l)*3),
                textcoords="offset points", va=va, ha="right")

# 4개월 간격으로 x축 형식 지정
ax.get_xaxis().set_major_locator(mdates.MonthLocator(interval=4))
ax.get_xaxis().set_major_formatter(mdates.DateFormatter("%b %Y"))
plt.setp(ax.get_xticklabels(), rotation=30, ha="right")

# y축과 spine(좌표의 테두리를 굵게 표시하는 방법)를 제
ax.get_yaxis().set_visible(False)
for spine in ["left", "top", "right"]:
    ax.spines[spine].set_visible(False)

ax.margins(y=0.1)
plt.show()
```

Python

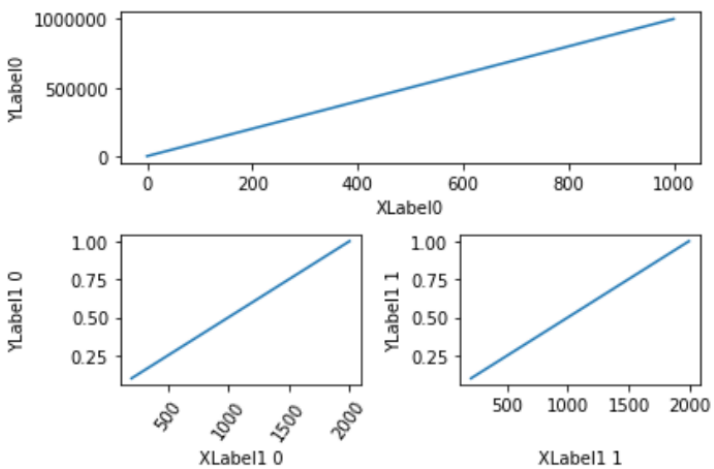
Out[-]



```python
# plot 나누기
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.gridspec as gridspec

fig = plt.figure(tight_layout=True)
gs = gridspec.GridSpec(2, 2)

ax = fig.add_subplot(gs[0, :])
ax.plot(np.arange(0, 1e6, 1000))
ax.set_ylabel('YLabel0')
ax.set_xlabel('XLabel0')

for i in range(2):
    ax = fig.add_subplot(gs[1, i])
    ax.plot(np.arange(1., 0., -0.1) * 2000., np.arange(1., 0., -0.1))
    ax.set_ylabel('YLabel1 %d' % i)
    ax.set_xlabel('XLabel1 %d' % i)
    if i == 0:
        for tick in ax.get_xticklabels():
            tick.set_rotation(55)
fig.align_labels()  # fig.align_xlabels(); fig.align_ylabels() 와 같이 하기

plt.show()
```

Python

Out[-]

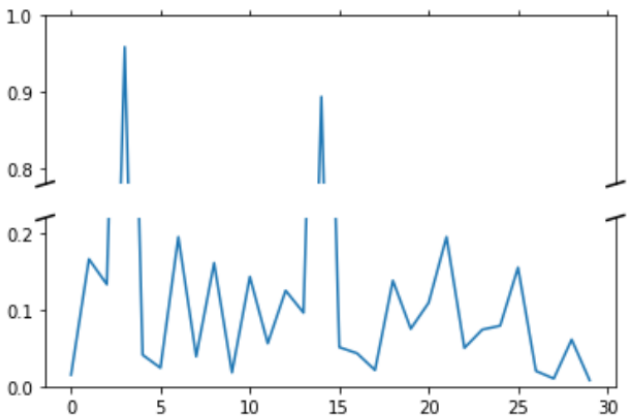


```
# 깨진 축

import matplotlib.pyplot as plt
import numpy as np


# [0, 0.2)사이에 np.random.rand(30)*.2 사용하여 만든 30개 데이터
pts = np.array([
    0.015, 0.166, 0.133, 0.159, 0.041, 0.024, 0.195, 0.039, 0.161, 0.018,
    0.143, 0.056, 0.125, 0.096, 0.094, 0.051, 0.043, 0.021, 0.138, 0.075,
    0.109, 0.195, 0.050, 0.074, 0.079, 0.155, 0.020, 0.010, 0.061, 0.008])

pts[[3, 14]] += .8

# 우리가 단순히 pts를 구성한다면, 우리는 이상치 때문에 대부분의 세부사항들을 잃게 됨.
# 따라서 Y 축을 두 부분으로 'break'하거나 'cut-out'함
# 이상치에는 상단(ax)을, 대다수의 데이터에는 하단(ax2)을 사용
f, (ax, ax2) = plt.subplots(2, 1, sharex=True)

# 두 축을 동일한 데이터로 plot
ax.plot(pts)
ax2.plot(pts)

ax.set_ylim(.78, 1.)  # 이상치만
ax2.set_ylim(0, .22)  # 대부분의 데이터

# ax와 ax2 사이에 spine 숨기기
ax.spines['bottom'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax.xaxis.tick_top()
ax.tick_params(labeltop=False)  # 맨 위에 눈금 label 표시 안 함
ax2.xaxis.tick_bottom()
```

```python
d = .015  # 축 좌표에서 사선을 만드는 방법
kwargs = dict(transform=ax.transAxes, color='k', clip_on=False)
ax.plot((-d, +d), (-d, +d), **kwargs)        # 왼쪽 상단 사선
ax.plot((1 - d, 1 + d), (-d, +d), **kwargs)  # 오른쪽 상단 사선

kwargs.update(transform=ax2.transAxes)  # 밑의 축으로 전환
ax2.plot((-d, +d), (1 - d, 1 + d), **kwargs)  # 왼쪽 하단 사선
ax2.plot((1 - d, 1 + d), (1 - d, 1 + d), **kwargs)  # 오른쪽 하단 사선

plt.show()
```

Python

Out[-]

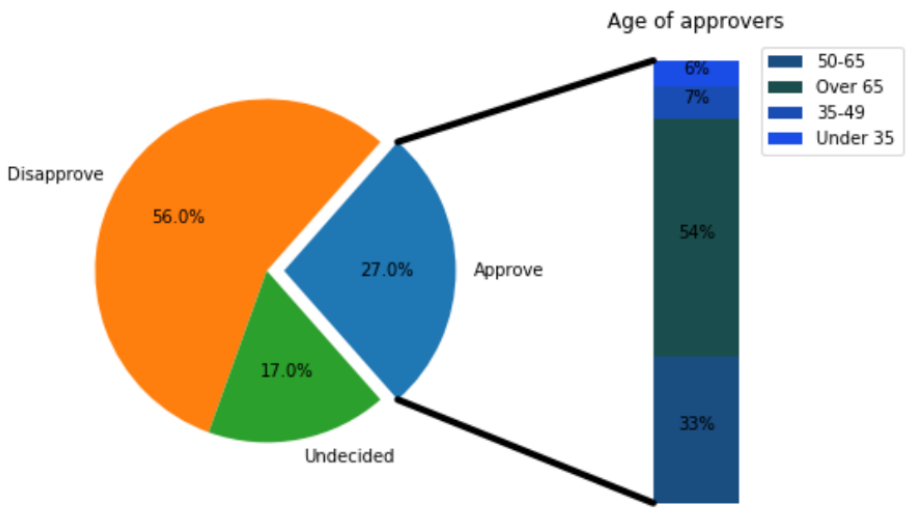


```
# Bar of pie

import matplotlib.pyplot as plt
from matplotlib.patches import ConnectionPatch
import numpy as np

fig = plt.figure(figsize=(9, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
fig.subplots_adjust(wspace=0)

# pie chart
ratios = [.27, .56, .17]
labels = ['Approve', 'Disapprove', 'Undecided']
explode = [0.1, 0, 0]
angle = -180 * ratios[0]
ax1.pie(ratios, autopct='%1.1f%%', startangle=angle,
        labels=labels, explode=explode)

# bar chart
xpos = 0
bottom = 0
ratios = [.33, .54, .07, .06]
width = .2
colors = [[.1, .3, .5], [.1, .3, .3], [.1, .3, .7], [.1, .3, .9]]

for j in range(len(ratios)):
    height = ratios[j]
    ax2.bar(xpos, height, width, bottom=bottom, color=colors[j])
    ypos = bottom + ax2.patches[j].get_height() / 2
    bottom += height
    ax2.text(xpos, ypos, "%d%%" % (ax2.patches[j].get_height() * 100),
             ha='center')

ax2.set_title('Age of approvers')
ax2.legend(('50-65', 'Over 65', '35-49', 'Under 35'))
ax2.axis('off')
ax2.set_xlim(- 2.5 * width, 2.5 * width)

# 두 plot 사이에 ConnectionPatch 사용해서 선 그리기
theta1, theta2 = ax1.patches[0].theta1, ax1.patches[0].theta2
center, r = ax1.patches[0].center, ax1.patches[0].r
bar_height = sum([item.get_height() for item in ax2.patches])
```

```python
# 상단 연결선 그리기
x = r * np.cos(np.pi / 180 * theta2) + center[0]
y = np.sin(np.pi / 180 * theta2) + center[1]
con = ConnectionPatch(xyA=(-width / 2, bar_height), coordsA=ax2.transData,
                      xyB=(x, y), coordsB=ax1.transData)
con.set_color([0, 0, 0])
con.set_linewidth(4)
ax2.add_artist(con)

# 하단 연결선 그리기
x = r * np.cos(np.pi / 180 * theta1) + center[0]
y = np.sin(np.pi / 180 * theta1) + center[1]
con = ConnectionPatch(xyA=(-width / 2, 0), coordsA=ax2.transData,
                      xyB=(x, y), coordsB=ax1.transData)
con.set_color([0, 0, 0])
ax2.add_artist(con)
con.set_linewidth(4)

plt.show()
```

Python

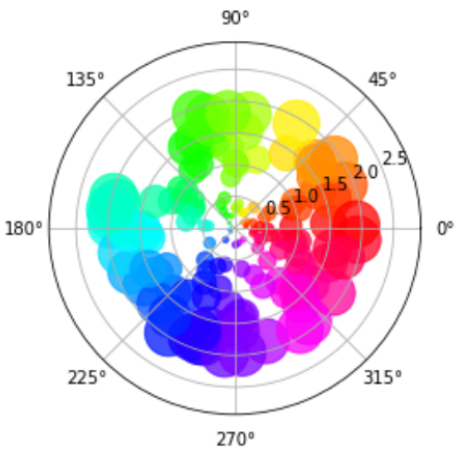
Out[-]



```python
## Scatter plot on polar axis

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(19680801)

# 영역 및 색상 계산
N = 150
r = 2 * np.random.rand(N)
theta = 2 * np.pi * np.random.rand(N)
area = 200 * r**2
colors = theta

fig = plt.figure()
ax = fig.add_subplot(111, projection='polar')
c = ax.scatter(theta, r, c=colors, s=area, cmap='hsv', alpha=0.75)
```

Python

Out[-]

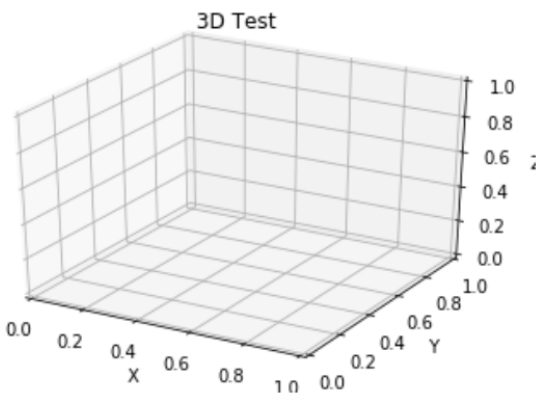


```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
# 주의!!
from mpl_toolkits import mplot3d

np.random.seed(19680801)


def gen_rand_line(length, dims=2):
    """
    random walk algorithm을 사용하여 선 만들기.

    Parameters
    ----------
    length : int
        The number of points of the line.
    dims : int
        The number of dimensions of the line.
    """
    line_data = np.empty((dims, length))
    line_data[:, 0] = np.random.rand(dims)
    for index in range(1, length):
        # 무작위 숫자를 0.1만큼 스케일링하므로 위치에 비해 움직임이 작음
        # 0.5로 빼는것은 범위를 [-0.5, 0.5]로 변경하여 선이 뒤로 움직일 수 있게함
        step = (np.random.rand(dims) - 0.5) * 0.1
        line_data[:, index] = line_data[:, index - 1] + step
    return line_data


def update_lines(num, dataLines, lines):
    for line, data in zip(lines, dataLines):
        line.set_data(data[0:2, :num])
        line.set_3d_properties(data[2, :num])
    return lines


# figure에 3D축 붙이기
fig = plt.figure()
ax = fig.add_subplot(projection="3d")

# 무작위 3D 선 50개
data = [gen_rand_line(25, 3) for index in range(50)]
```

```python
# 50개 선 생성.
# 빈 배열을 3D 버전의 plot으로 전달할 수 없음
lines = [ax.plot(dat[0, 0:1], dat[1, 0:1], dat[2, 0:1])[0] for dat in data]

# 축 속성 설정
ax.set_xlim3d([0.0, 1.0])
ax.set_xlabel('X')

ax.set_ylim3d([0.0, 1.0])
ax.set_ylabel('Y')

ax.set_zlim3d([0.0, 1.0])
ax.set_zlabel('Z')

ax.set_title('3D Test')

# 애니메이션 개체 만들기
line_ani = animation.FuncAnimation(
    fig, update_lines, 25, fargs=(data, lines), interval=50)

plt.show()
```

Python

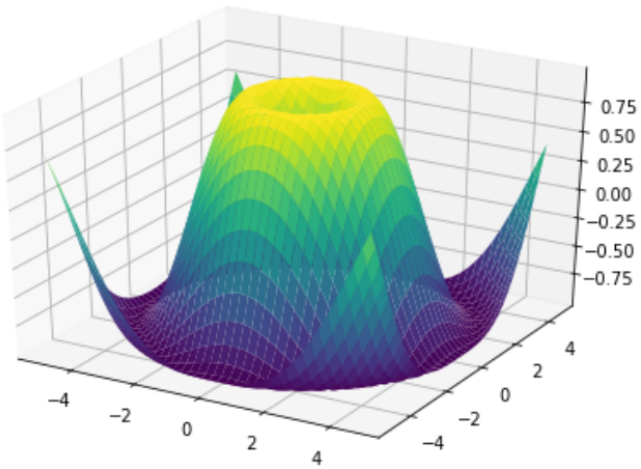
Out[-]



```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.viridis)

plt.show()
```

Python

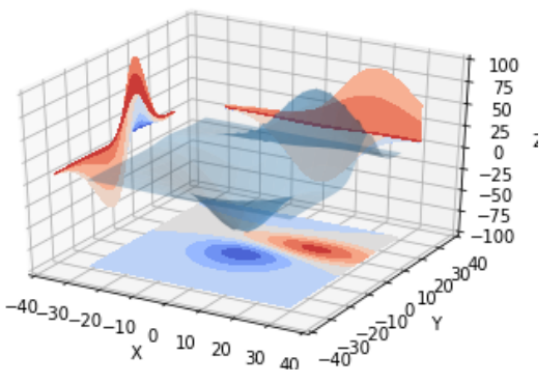
Out[-]



```python
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm

fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)

# Plot the 3D surface
ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.3)

cset = ax.contourf(X, Y, Z, zdir='z', offset=-100, cmap=cm.coolwarm)
cset = ax.contourf(X, Y, Z, zdir='x', offset=-40, cmap=cm.coolwarm)
cset = ax.contourf(X, Y, Z, zdir='y', offset=40, cmap=cm.coolwarm)

ax.set_xlim(-40, 40)
ax.set_ylim(-40, 40)
ax.set_zlim(-100, 100)

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

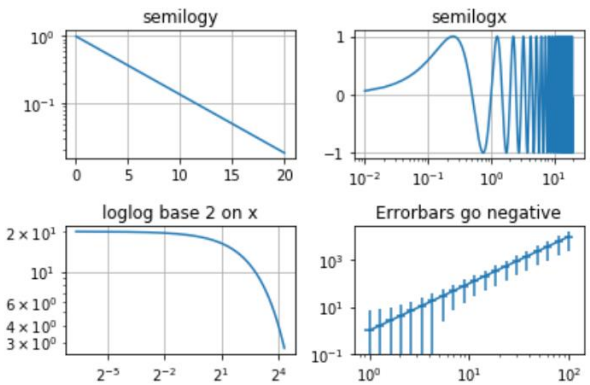
Python

Out[-]



```
import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0.01, 20.0, 0.01)

# figure 생성
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
# log y axis
ax1.semilogy(t, np.exp(-t / 5.0))
ax1.set(title='semilogy')
ax1.grid()

# log x axis
ax2.semilogx(t, np.sin(2 * np.pi * t))
ax2.set(title='semilogx')
ax2.grid()
# log x and y axis
ax3.loglog(t, 20 * np.exp(-t / 10.0), basex=2)
ax3.set(title='loglog base 2 on x')
ax3.grid()

x = 10.0**np.linspace(0.0, 2.0, 20)
y = x**2.0
ax4.set_xscale("log", nonposx='clip')
ax4.set_yscale("log", nonposy='clip')
ax4.set(title='Errorbars go negative')
ax4.errorbar(x, y, xerr=0.1 * x, yerr=5.0 + 0.75 * y)

ax4.set_ylim(bottom=0.1)
fig.tight_layout()
plt.show()
```

Out[-]



# 5일차 데이터 시각화 2

이 장에서 다루는 내용

Seaborn

Plotly Python Open Source Graphing Library

Plotly

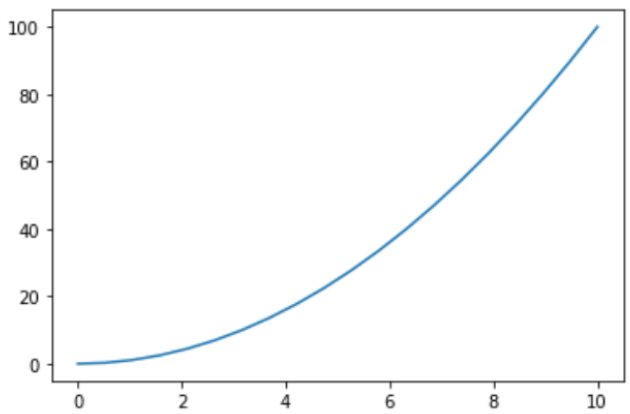
## Seaborn

- Matplotlib에 API에 대한 불만족
- Pandas DataFrame과의 호완성 부족(pandas가 개발된 것은 2008년, matplotlib은 2003년)
  - 이 부분은 많은 부분이 해결되었음
- https://seaborn.pydata.org/

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

x = np.linspace(0, 10, 20)
y = x ** 2

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

Python

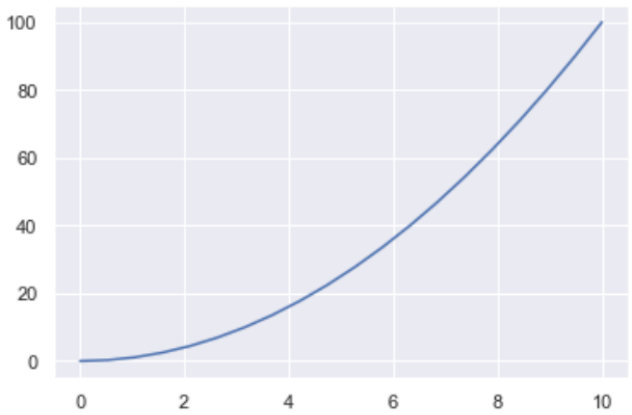
Out[-]



```python
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

x = np.linspace(0, 10, 20)
y = x ** 2

sns.set()

plt.plot(x, y)

plt.show()
```

Python

Out[-]



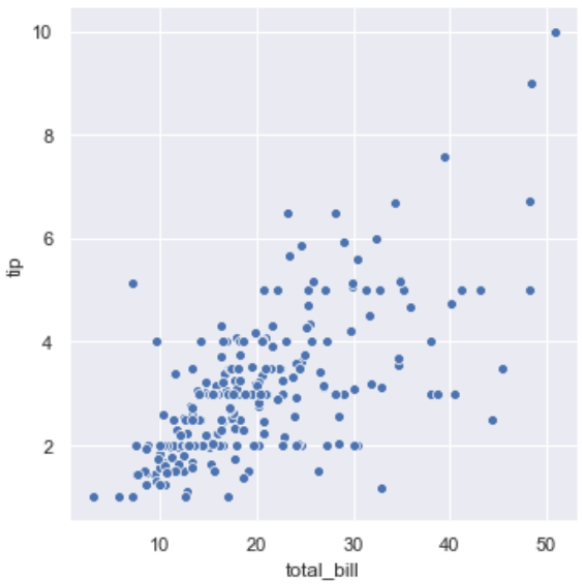
## relplot

- 기본 설정은 산점도 그리기
  - hue : 종류에 따라서 색상을 다르게 분류
  - style : 종류에 따라서 마커를 다르게 분류
  - sizes : 크기를 다르게 해서 분류
  - col_wrap : n개의 컬럼으로 나눔
  - height : 각각의 그래프 높이
  - kind : 산점도 형식에서 다른 형식으로 변경

```
tips = sns.load_dataset("tips") # sns.load_dataset : seaborn에서 제공하는 데이터 불러오기
    sns.relplot(x="total_bill", y="tip", data=tips)

    plt.show()
sns.relplot(x="total_bill", y="tip", data=tips)

plt.show()
```

Python

Out[-]



```python
tips.head(10)
```

Python

```
Out[-]
  total_bill   tip     sex smoker  day  time   size
0      16.99  1.01  Female  No    Sun  Dinner  2
1      10.34  1.66    Male  No    Sun  Dinner  3
2      21.01  3.50    Male  No    Sun  Dinner  3
3      23.68  3.31    Male  No    Sun  Dinner  2
4      24.59  3.61  Female  No    Sun  Dinner  4
5      25.29  4.71    Male  No    Sun  Dinner  4
6       8.77  2.00     Male No    Sun  Dinner  2
7      26.88  3.12    Male  No    Sun  Dinner  4
8      15.04  1.96    Male  No    Sun  Dinner  2
9      14.78  3.23    Male  No    Sun  Dinner  2
```

Python

```python
type(tips)
```

Python

```
Out[-]
pandas.core.frame.DataFrame
```

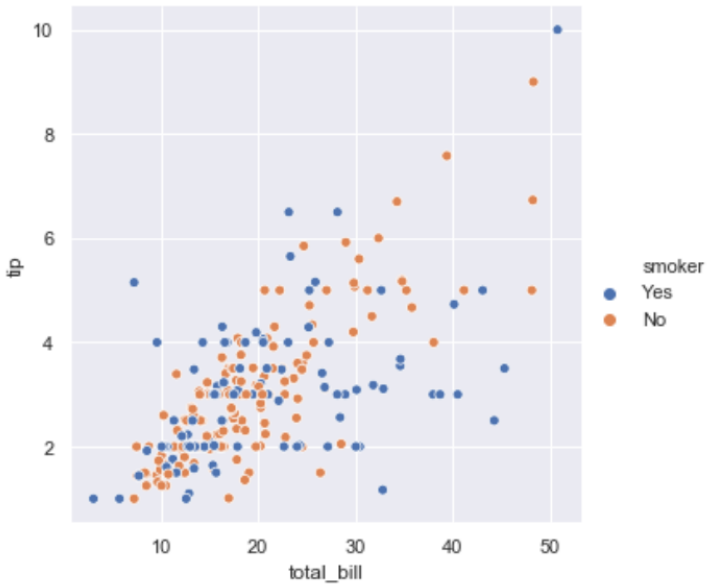
Python

```
sns.relplot(x="total_bill", y="tip", hue="smoker", data=tips)
```

Python

Out[-]

<seaborn.axisgrid.FacetGrid at 0x1b613781860>

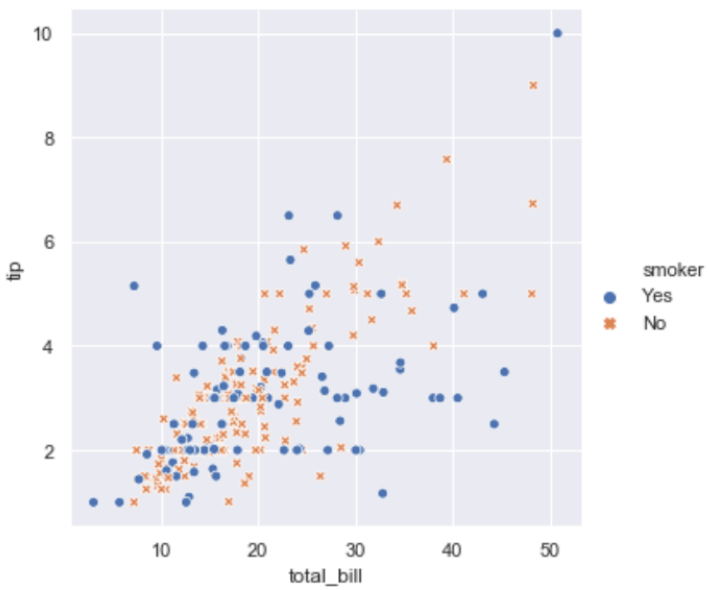



```
sns.relplot(x="total_bill", y="tip", hue="smoker", style='smoker', data=tips)
```

Python

Out[-]

<seaborn.axisgrid.FacetGrid at 0x1b613833710>

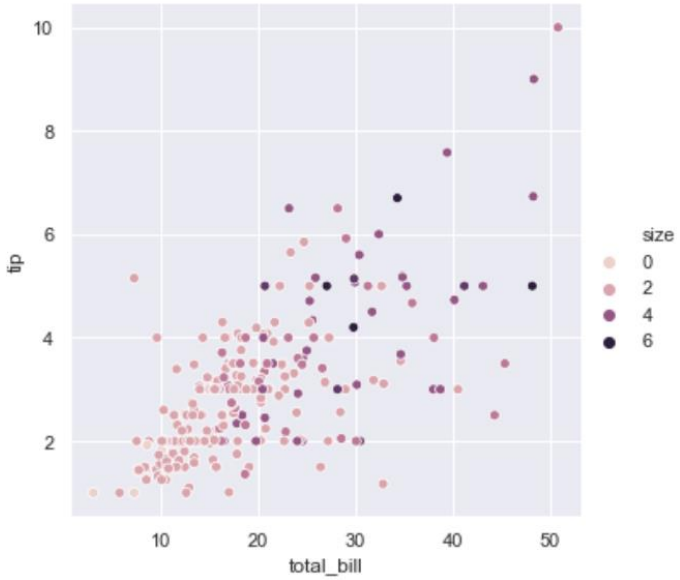


```python
sns.relplot(x="total_bill", y="tip", hue="size", data=tips)
```

Python

Out[-]

<seaborn.axisgrid.FacetGrid at 0x1b6115a1e10>

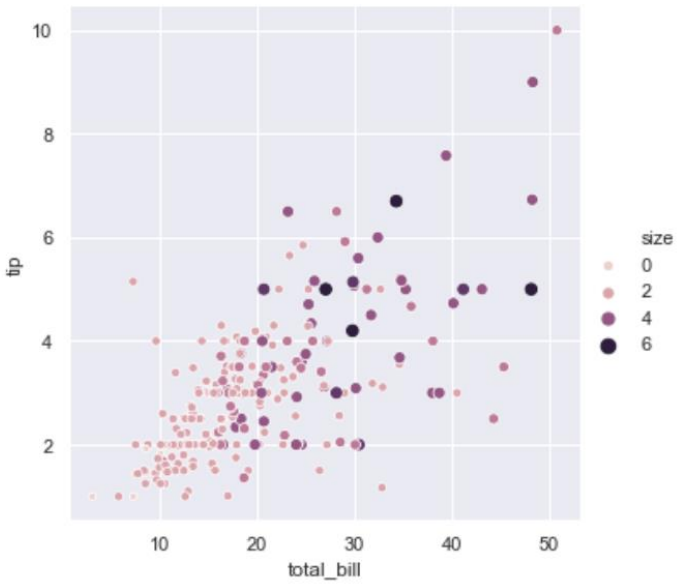



```python
sns.relplot(x="total_bill", y="tip", hue="size", size='size', data=tips)
```

Python

Out[-]

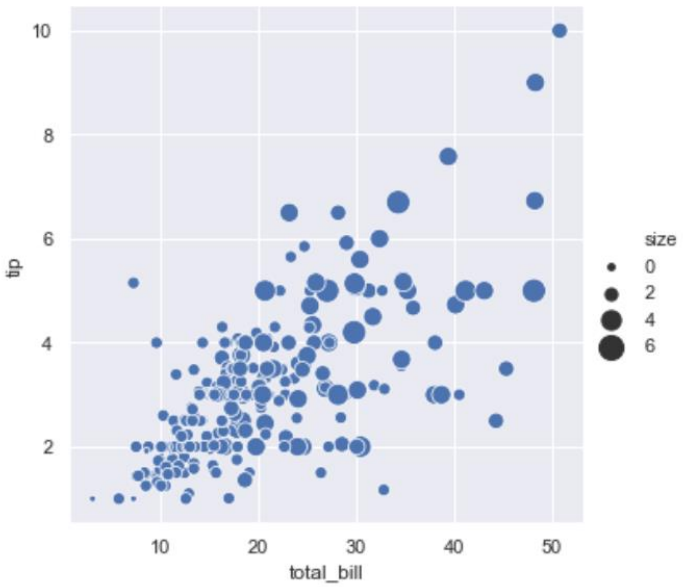
<seaborn.axisgrid.FacetGrid at 0x1b6133d9ef0>



```
sns.relplot(x="total_bill", y="tip", size="size", sizes=(15, 200), data=tips)
```

Python

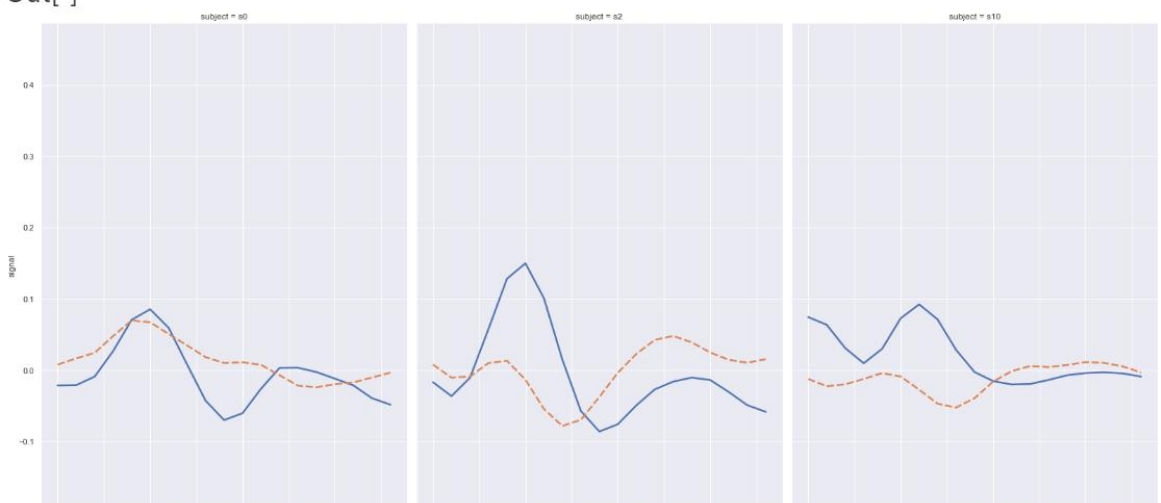
Out[-]
<seaborn.axisgrid.FacetGrid at 0x1b6139fbba8>

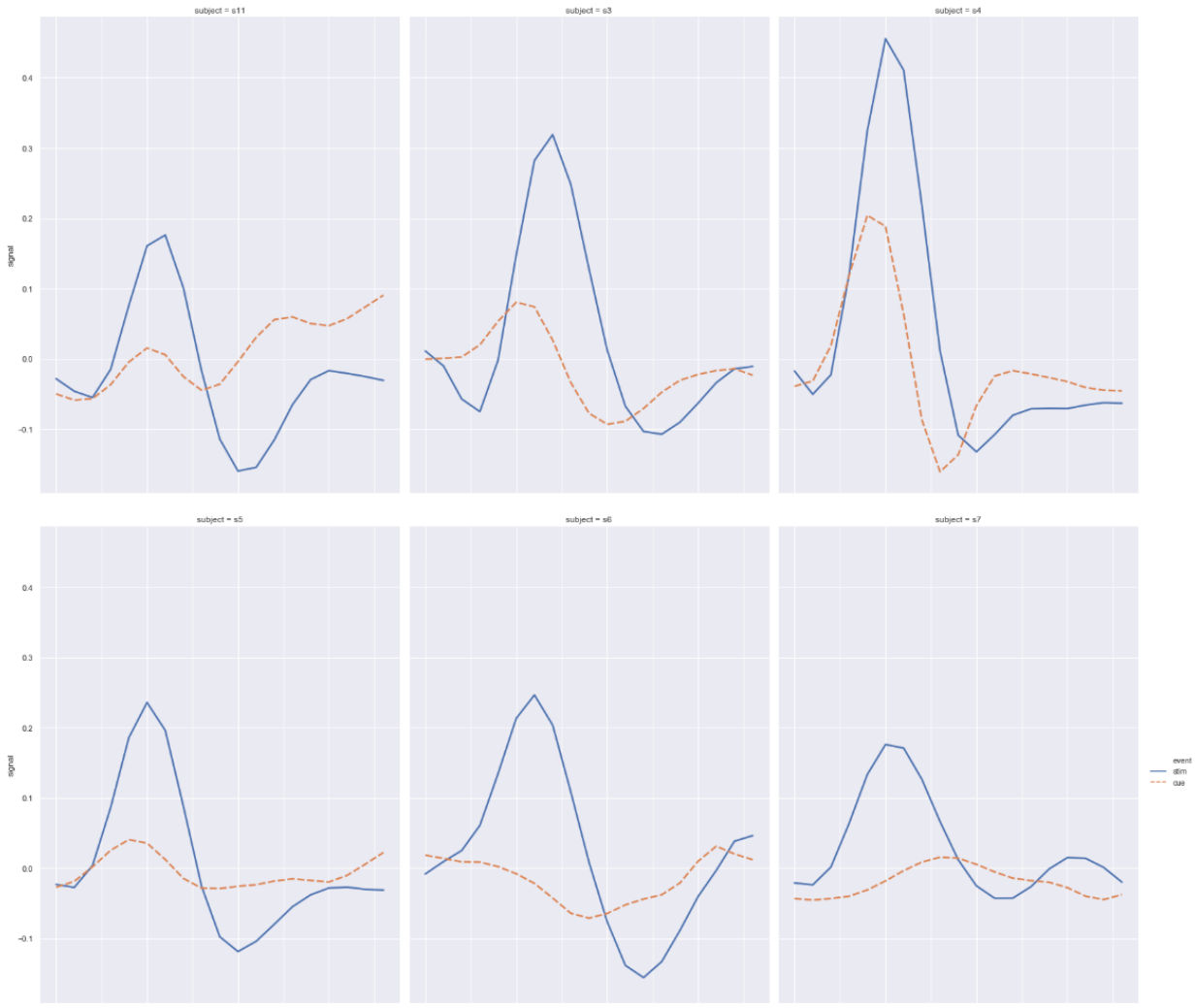


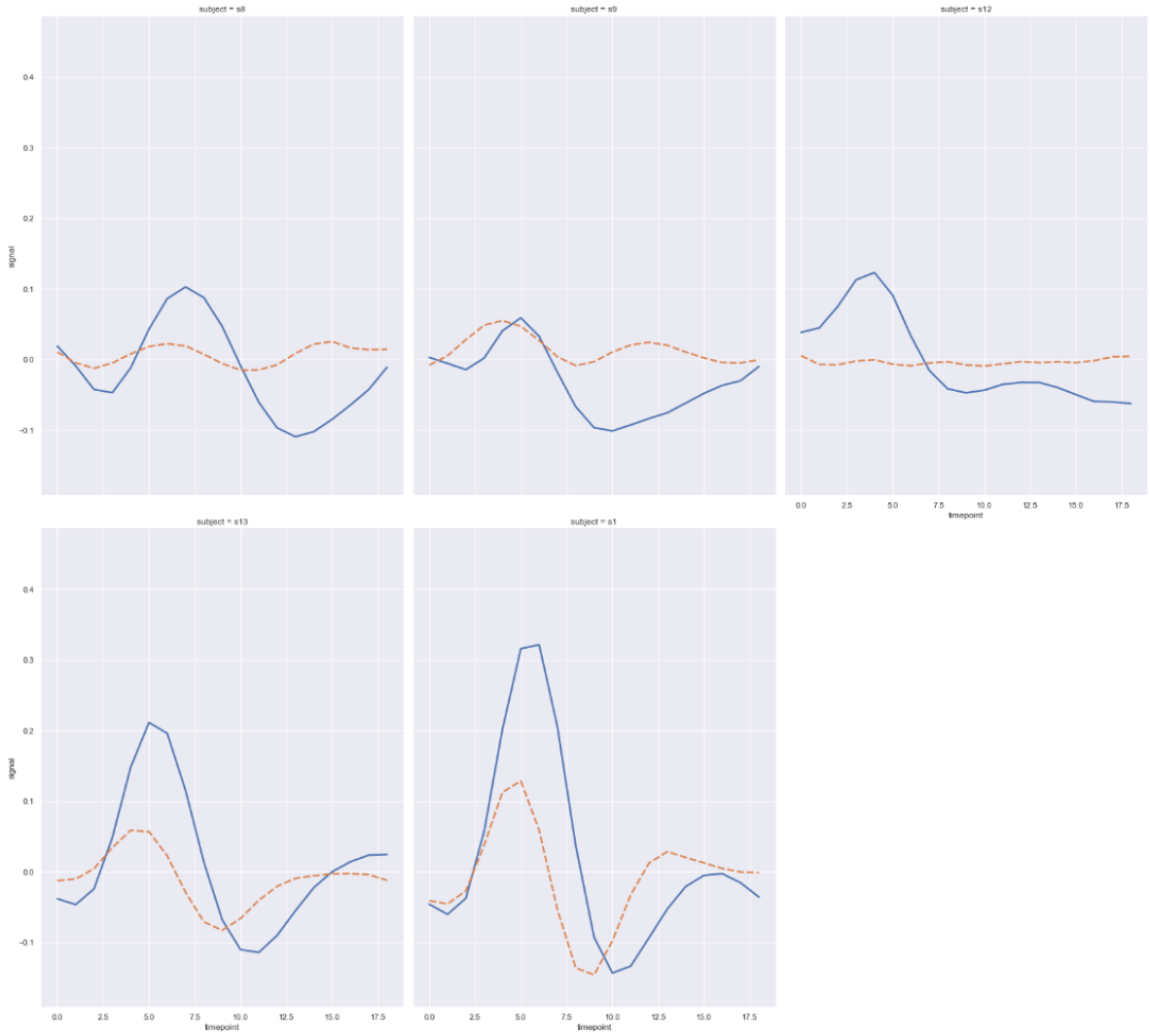

```
fmri = sns.load_dataset("fmri")
sns.relplot(x="timepoint", y="signal", hue="event", style="event",
            col="subject", col_wrap=3,
            height=10, aspect=.75, linewidth=2.5,
            kind="line", data=fmri.query("region == 'frontal'"));
```

Python

Out[-]




subject = s11
subject = s3
subject = s4
subject = s5
subject = s6
subject = s7
signal
0.4
0.3
0.2
0.1
0.0
-0.1
event
stim
cue


subject = s8
subject = s9
subject = s12
subject = s13
subject = s1
signal
timepoint
0.4
0.3
0.2
0.1
0.0
-0.1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

## countplot()

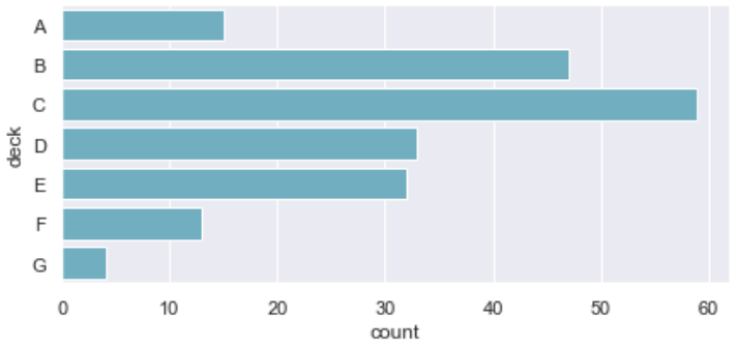
- 지정한 컬럼의 데이터가 얼마나 있는지 그리기

```python
titanic = sns.load_dataset("titanic")
f, ax = plt.subplots(figsize=(7, 3))
sns.countplot(y="deck", data=titanic, color="c") # 가로 그래프 생성
                                                 # y 대신 x를 사용하면 세로 그래프
```

Python

Out[-]

<matplotlib.axes._subplots.AxesSubplot at 0x1b614314c88>



## pairplot()

- 3차원 이상의 데이터에서 각 데이터끼리의 그래프를 그리기

```
titanic.head(10)
```

Python

```
Out[-]
  survived  pclass     sex   age  sibsp parch     fare  embarked  class     who adult_mal
e deck  embark_town alive alone
0        0       3    male  22.0      1     0   7.2500         S   Third    man      True
NaN Southampton   no  False
1        1       1  female  38.0      1     0  71.2833         C   First  woman     False
C   Cherbourg  yes  False
2        1       3  female  26.0      0     0   7.9250         S   Third  woman     False
NaN Southampton  yes   True
3        1       1  female  35.0      1     0  53.1000         S   First  woman     False
C Southampton  yes  False
4        0       3    male  35.0      0     0   8.0500         S   Third    man      True
NaN Southampton   no   True
5        0       3    male   NaN      0     0   8.4583         Q   Third    man      True
NaN  Queenstown   no   True
6        0       1    male  54.0      0     0  51.8625         S   First    man      True
E Southampton   no   True
7        0       3    male   2.0      3     1  21.0750         S   Third  child     False
NaN Southampton   no  False
8        1       3  female  27.0      0     2  11.1333         S   Third  woman     False
NaN Southampton  yes  False
9        1       2  female  14.0      1     0  30.0708         C  Second  child     False
NaN   Cherbourg  yes  False
```

Python

```
sns.pairplot(titanic, hue='class')

plt.show()
```

Python

Out[-]


survived
pclass
age
sibsp
parch
fare
adult_male
alone
class
First
Second
Third

## Plotly Python Open Source Graphing Library

- 공식 홈페이지(https://plotly.com/python/) 튜토리얼

## Plotly

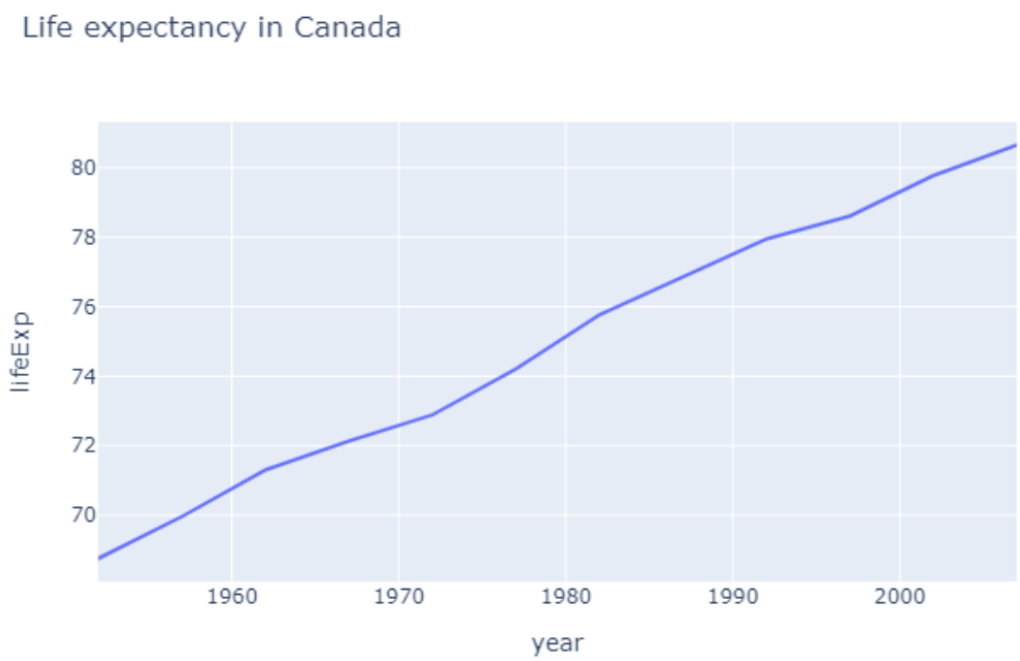
- 인터랙티브한 그래프를 그리기에 적합한 패키지
- 웹 시각화인 자바스크립트의 라이브러리 D3를 이용해 그래프가 웹에서 빠르게 그려진다.

### Line

```python
import plotly.express as px

df = px.data.gapminder().query("country=='Canada'")

fig = px.line(df,
              x="year", y="lifeExp",
              title='Life expectancy in Canada')

fig.show()
```

Python

Out[-]

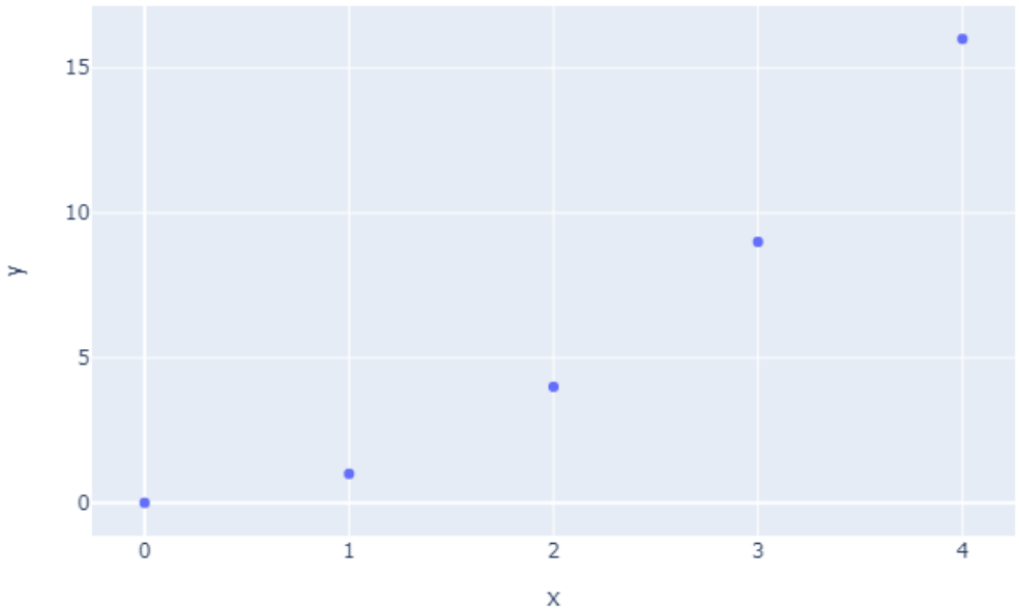


## Scatter

```
fig = px.scatter(x=[0, 1, 2, 3, 4], y=[0, 1, 4, 9, 16])
fig.show()
```

Python

Out[-]

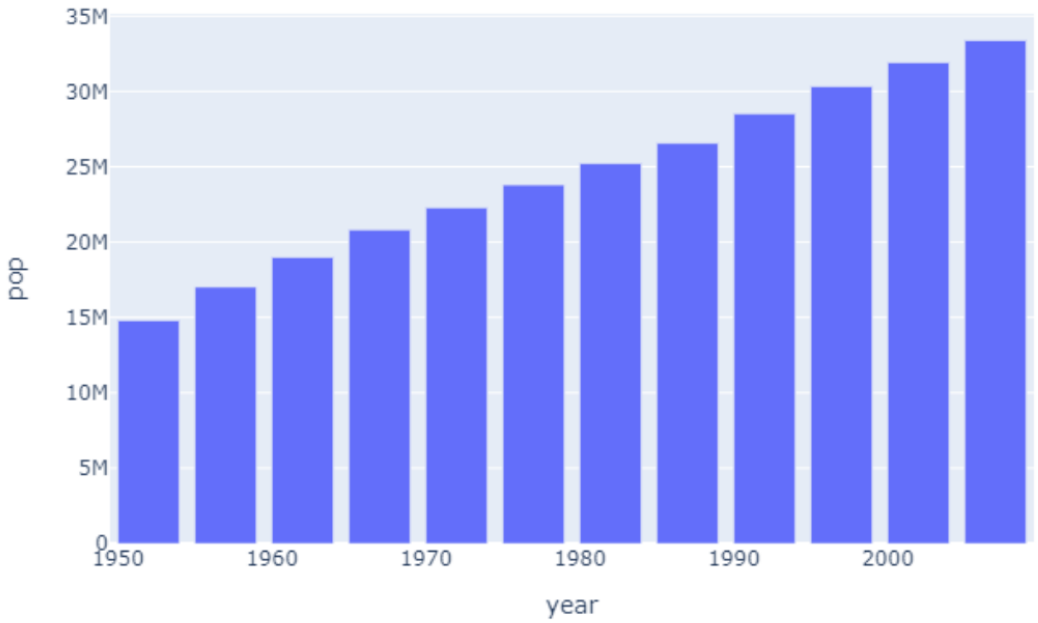


## Bar

```
data_canada = px.data.gapminder().query("country == 'Canada'")

fig = px.bar(data_canada,
             x='year', y='pop')

fig.show()
```

Python

Out[-]



```
data_canada
```

Python

Out[-]

| | country | continent | year | lifeExp | pop | gdpPercap | iso_alpha | iso_num |
|---|---|---|---|---|---|---|---|---|
| 240 | Canada | Americas | 1952 | 68.750 | 14785584 | 11367.16112 | CAN | 124 |
| 241 | Canada | Americas | 1957 | 69.960 | 17010154 | 12489.95006 | CAN | 124 |
| 242 | Canada | Americas | 1962 | 71.300 | 18985849 | 13462.48555 | CAN | 124 |
| 243 | Canada | Americas | 1967 | 72.130 | 20819767 | 16076.58803 | CAN | 124 |
| 244 | Canada | Americas | 1972 | 72.880 | 22284500 | 18970.57086 | CAN | 124 |
| 245 | Canada | Americas | 1977 | 74.210 | 23796400 | 22090.88306 | CAN | 124 |
| 246 | Canada | Americas | 1982 | 75.760 | 25201900 | 22898.79214 | CAN | 124 |
| 247 | Canada | Americas | 1987 | 76.860 | 26549700 | 26626.51503 | CAN | 124 |
| 248 | Canada | Americas | 1992 | 77.950 | 28523502 | 26342.88426 | CAN | 124 |
| 249 | Canada | Americas | 1997 | 78.610 | 30305843 | 28954.92589 | CAN | 124 |
| 250 | Canada | Americas | 2002 | 79.770 | 31902268 | 33328.96507 | CAN | 124 |
| 251 | Canada | Americas | 2007 | 80.653 | 33390141 | 36319.23501 | CAN | 124 |

Python

## Pie

- pull : Pie 차트의 조각을 분리함

```python
import plotly
print(plotly.__version__) #버전확인
```

Python

```
Out[-]
4.5.4
```

Python

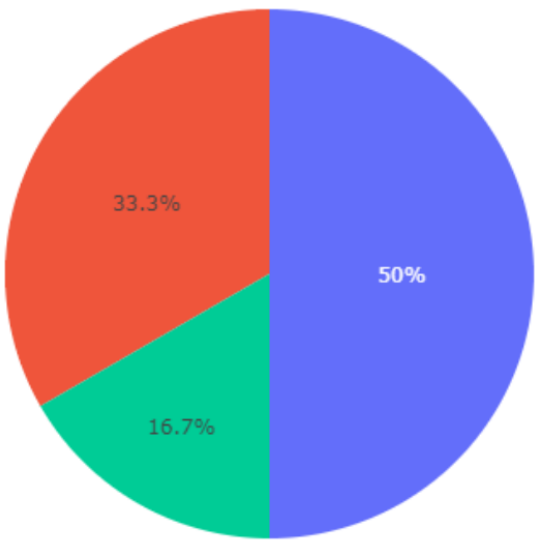
```python
# 버전이 낮으면 파이차트 불가능
# !pip3 install plotly
```

Python

```python
import plotly.express as px
fig = px.pie(values=[1, 2, 3])
fig.show()
```

Python

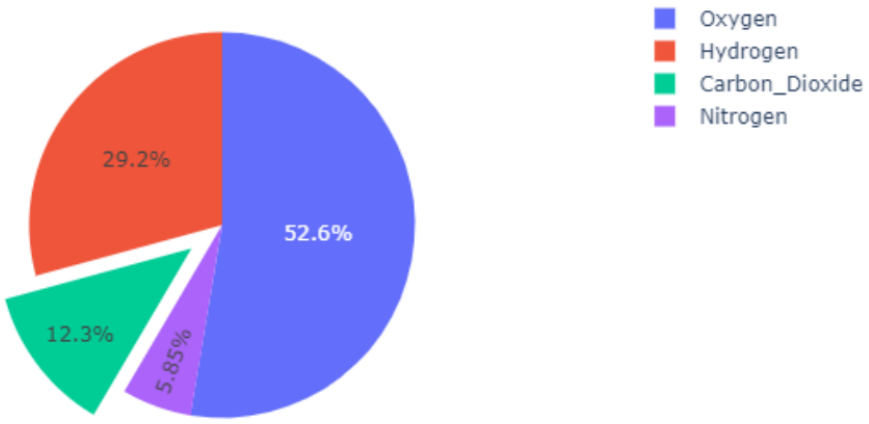
Out[-]



```python
import plotly.graph_objects as go

labels = ['Oxygen','Hydrogen','Carbon_Dioxide','Nitrogen']
values = [4500, 2500, 1053, 500]

fig = go.Figure(data=[go.Pie(labels=labels, values=values, pull=[0, 0, 0.2, 0])])

fig.show()
```

Python

Out[-]



## Sunburst

- 데이터의 계층 구조를 보기 쉽게 나타냄

```python
import plotly.graph_objects as go

fig =go.Figure(go.Sunburst(
    labels=["Eve", "Cain", "Seth", "Enos", "Noam", "Abel", "Awan", "Enoch", "Azura"],
    parents=["", "Eve", "Eve", "Seth", "Seth", "Eve", "Eve", "Awan", "Eve" ],
    values=[10, 14, 12, 10, 2, 6, 6, 4, 4],
))

fig.update_layout(margin = dict(t=0, l=0, r=0, b=0))

fig.show()
```

Python

Out[-]

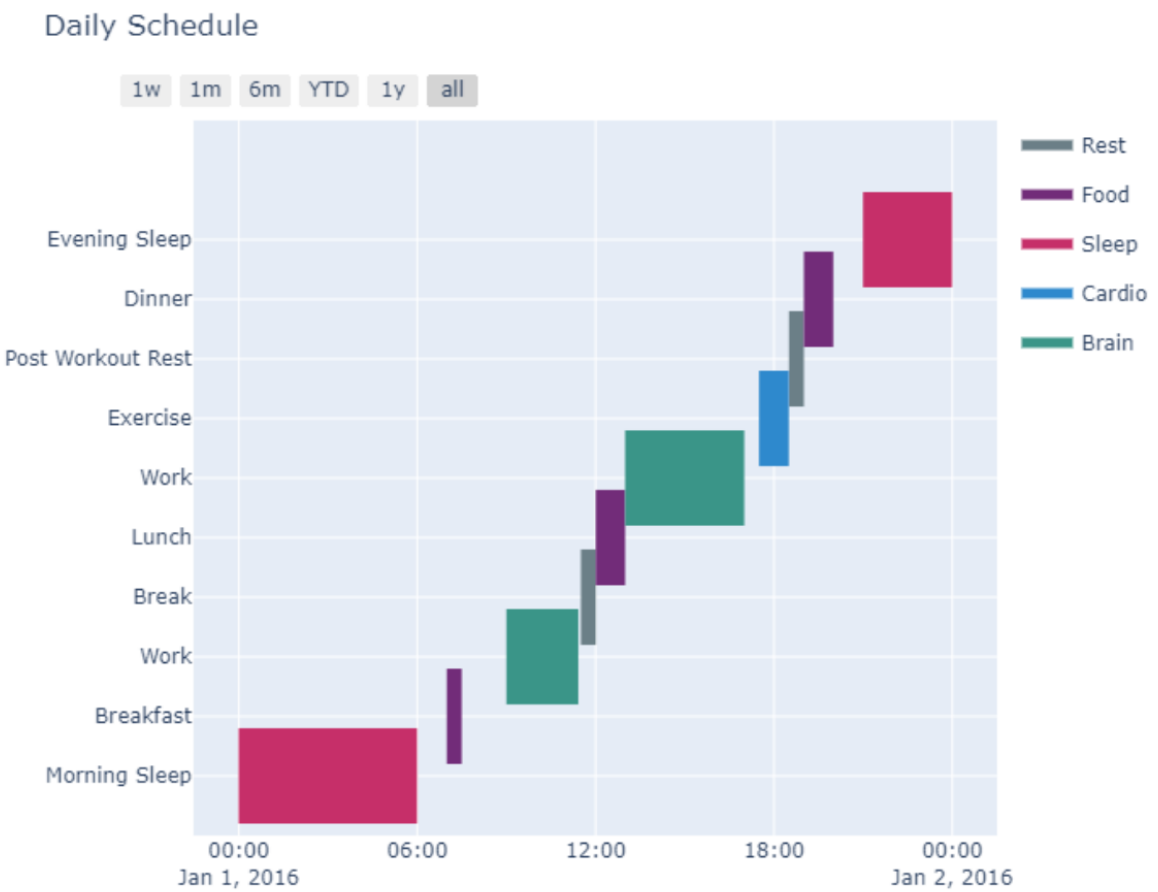
## gantt

- 일정관리를 위해서 bar형태로 만든 차트

```python
import plotly.figure_factory as ff

df = [
    dict(Task='Morning Sleep', Start='2016-01-01',
         Finish='2016-01-01 6:00:00', Resource='Sleep'),
    dict(Task='Breakfast', Start='2016-01-01 7:00:00',
         Finish='2016-01-01 7:30:00', Resource='Food'),
    dict(Task='Work', Start='2016-01-01 9:00:00',
         Finish='2016-01-01 11:25:00', Resource='Brain'),
    dict(Task='Break', Start='2016-01-01 11:30:00',
         Finish='2016-01-01 12:00:00', Resource='Rest'),
    dict(Task='Lunch', Start='2016-01-01 12:00:00',
         Finish='2016-01-01 13:00:00', Resource='Food'),
    dict(Task='Work', Start='2016-01-01 13:00:00',
         Finish='2016-01-01 17:00:00', Resource='Brain'),
    dict(Task='Exercise', Start='2016-01-01 17:30:00',
         Finish='2016-01-01 18:30:00', Resource='Cardio'),
    dict(Task='Post Workout Rest', Start='2016-01-01 18:30:00',
         Finish='2016-01-01 19:00:00', Resource='Rest'),
    dict(Task='Dinner', Start='2016-01-01 19:00:00',
         Finish='2016-01-01 20:00:00', Resource='Food'),
    dict(Task='Evening Sleep', Start='2016-01-01 21:00:00',
         Finish='2016-01-01 23:59:00', Resource='Sleep')
]

colors = dict(Cardio = 'rgb(46, 137, 205)',
              Food = 'rgb(114, 44, 121)',
              Sleep = 'rgb(198, 47, 105)',
              Brain = 'rgb(58, 149, 136)',
              Rest = 'rgb(107, 127, 135)')

fig = ff.create_gantt(df, colors=colors, index_col='Resource',
                      title='Daily Schedule', show_colorbar=True,
                      bar_width=0.8, showgrid_x=True, showgrid_y=True)
fig.show()
```

Python

Out[-]

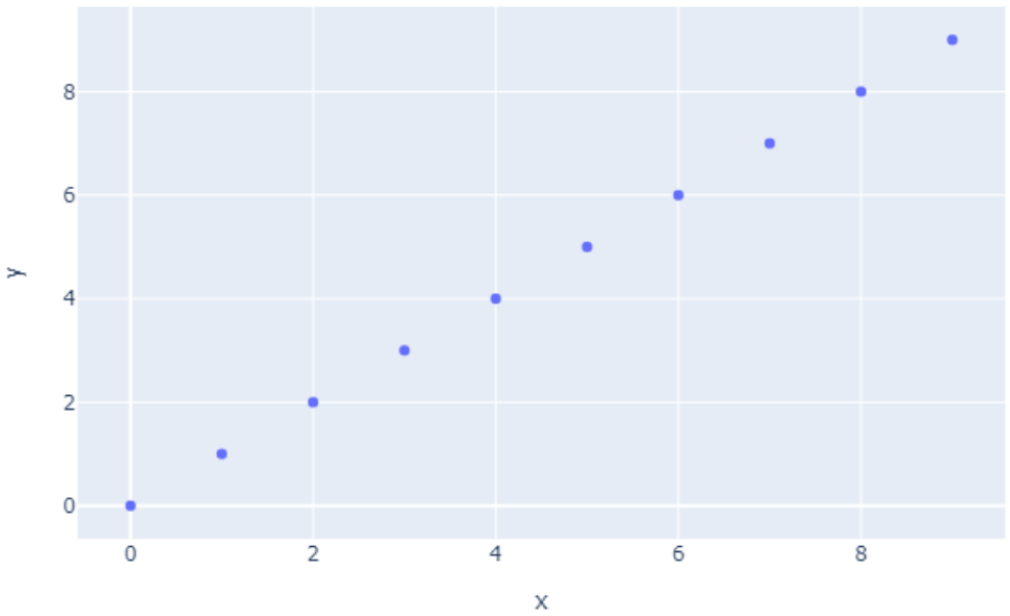

Daily Schedule
1w
1m
6m
YTD
1y
all
Evening Sleep
Dinner
Post Workout Rest
Exercise
Work
Lunch
Break
Work
Breakfast
Morning Sleep
00:00
Jan 1, 2016
06:00
12:00
18:00
00:00
Jan 2, 2016
Rest
Food
Sleep
Cardio
Brain

- https://plotly.com/python/interactive-html-export/
- https://plotly.com/python/v3/static-image-export/

```python
import plotly.express as px

fig =px.scatter(x=range(10), y=range(10))
fig.show()
# fig.write_html("path/to/file.html")
```

Python

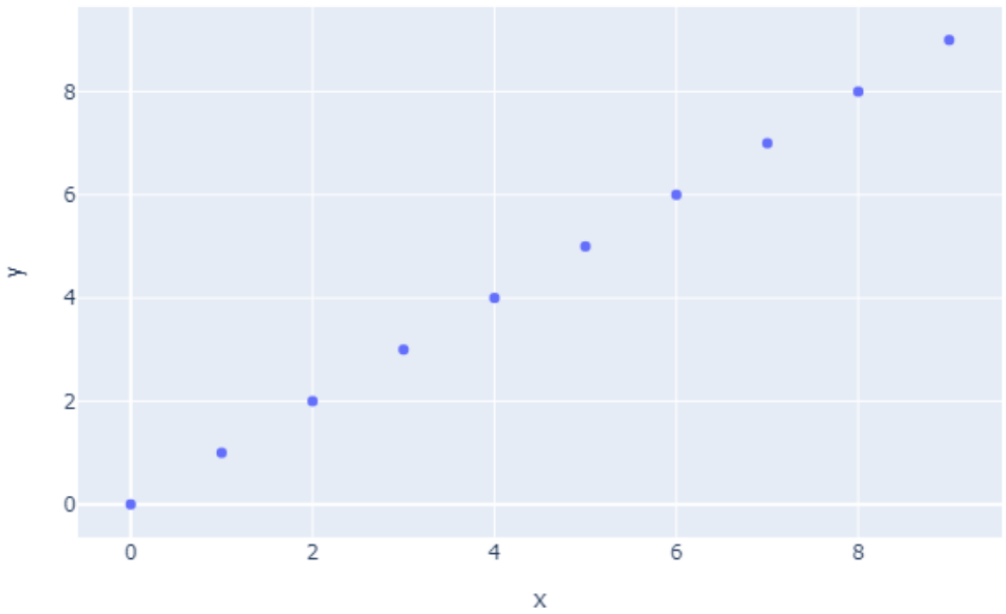
Out[-]



```python
import plotly.express as px

fig =px.scatter(x=range(10), y=range(10))

fig.show()
# html로 저장
fig.write_html("file.html")
```

Python

Out[-]



# 6일차 코로나 데이터 분석

이 장에서 다루는 내용

코로나19 성별, 나이별 분석
코로나19 지역별 분석
코로나19 한국 데이터 분석
세계 코로나19 현황

```python
# 그래프에 한글 출력하기 위한 코드
from matplotlib import font_manager, rc

font_name = font_manager.FontProperties(
    fname = 'c:/Windows/Fonts/malgun.ttf').get_name()
rc('font', family=font_name)

# import matplotlib.pyplot as plt
# plt.rc('font', family='AppleGothic') # mac
# plt.rc('font', family='Malgun Gothic') # window
```

Python

# 1. 성별, 나이별

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

# 출처 : 질병관리본부
confirmed = pd.read_csv('./data/daily_Confirmed.csv', index_col = ['Date'])
```

Python

```python
#pd.read_csv('./data.csv', encoding='utf-8') # Default(기본)
#pd.read_csv('./data.csv', encoding='cp949') # 11172자
#pd.read_csv('./data.csv', encoding='euc-kr') # 2350자
```

Python

```python
confirmed.info() # confirmed의 정보 확인
```

Python

```
<class 'pandas.core.frame.DataFrame'>
Index: 32 entries, 2020-03-01 to 2020-04-01
Data columns (total 11 columns):
Female    32 non-null int64
Male      32 non-null int64
0-9       32 non-null int64
10-       32 non-null int64
20-29     32 non-null int64
30-39     32 non-null int64
40-49     32 non-null int64
50-59     32 non-null int64
60-69     32 non-null int64
70-79     32 non-null int64
80-       32 non-null int64
dtypes: int64(11)
memory usage: 3.0+ KB
```

Python

```
confirmed.head() # defult 값은 5
```

Python

```
Out[-]
            Female  Male  0-9  10- 20-29  30-39 40-49 50-59 60-69 70-79 80-
      Date
2020-03-01    2197  1329  27  137   1054   426   521   687   453   158  63
2020-03-02    2621  1591  32  169   1235   506   633   834   530   192  81
2020-03-03    3002  1810  34  204   1417   578   713   952   597   224  93
2020-04-04    3332  1996  34  233   1575   631   790  1051   646   260  108
2020-03-05    3617  2149  38  257   1727  6 59   847  1127   699   288  124
```

Python

```python
confirmed.tail()
```

Python

```
Out[-]
            Female  Male  0-9  10- 20-29  30-39 40-49 50-59 60-69 70-79 80-
      Date
2020-03-28    5742  3736  109  501  2567   978  1278  1780  1201   632  432
2020-03-29    5784  3799  111  508  2602   991  1292  1798  1210   635  434
2020-03-30    5827  3836  112  513  2630  1002  1297  1812  1218   640  437
2020-03-31    5881  3905  112  515  2656  1012  1312  1851  1235   651  442
2020-04-01    5941  3946  116  519  2682  1027  1323  1865  1245   658  452
```

Python

```python
confirmed.isnull().sum()  # 결측치 개수 구하기
```

Python

```
Out[-]
Female    0
Male      0
0-9       0
10-       0
20-29     0
30-39     0
40-49     0
50-59     0
60-69     0
70-79     0
80-       0
dtype: int64
```

Python

```python
# 혹시 결측치가 있다면!
# pd.to_numeric(df['컬럼이름'], errors='coerce') -> nan
```

Python

```python
confirmed['Female'].describe()
```

Python

```
Out[-]
count      32.000000
mean     4882.531250
std       999.970515
min      2197.000000
25%      4547.250000
50%      5146.500000
75%      5601.000000
max      5986.000000
Name: Female, dtype: float64
```

Python

## 시각화

- plt.pie
  - rcParams : 차트 크기, 선의 색, 두께 등 설정
  - explode : Pie 차트 조각 추출되는 크기
  - autopct : Pie 차트 조각의 전체 대비 백분율

```python
# 마지막 날인 2020년 4월 1일 데이터
# 성별 데이터 가져오기
sex = confirmed.iloc[-1, :2]
sex
```

Python

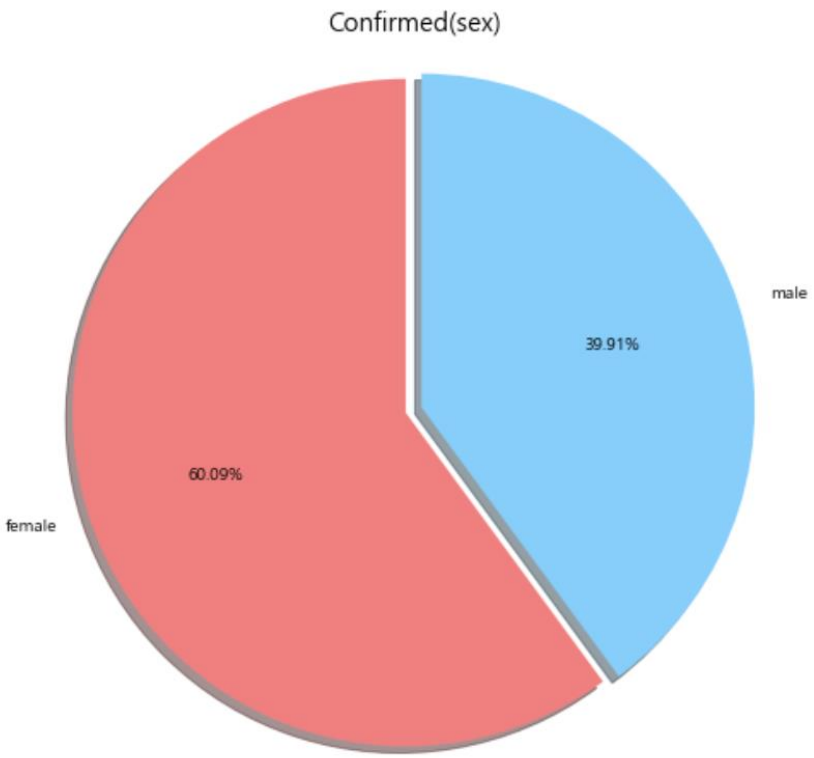
```
Out[-]
Female    5941
Male      3946
Name: 2020-04-01, dtype: int64
```

Python

```python
# 성별에 따른 확진자 수 pie 그래프
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = 12,8
group_explodes = (0.05, 0)

plt.pie(sex, labels=['female', 'male'],
        explode = group_explodes,
        shadow = True,
        startangle=90,
        colors = ['lightcoral', 'lightskyblue'],
        autopct='%1.2f%%')

plt.title('Confirmed(sex)', size=15)
plt.axis('equal')
plt.show()
```

Python

Out[-]



```python
# 2020년 3월 1일 ~ 4월 1일 시계열 데이터
sex = confirmed.iloc[:, :2]
sex
```

Python

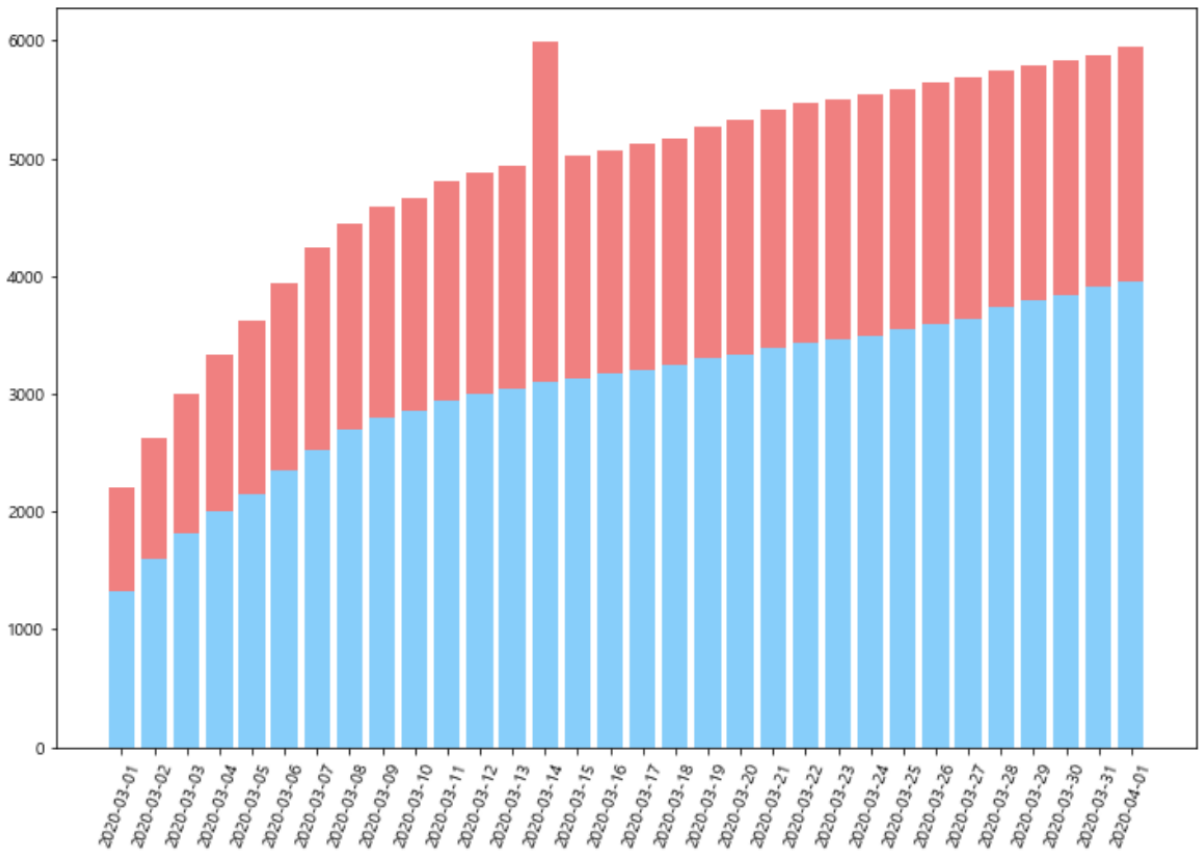
Out[-]

| Date | Female | Male |
|---|---|---|
| 2020-03-01 | 2197 | 1329 |
| 2020-03-02 | 2621 | 1591 |
| 2020-03-03 | 3002 | 1810 |
| 2020-03-04 | 3332 | 1996 |
| 2020-03-05 | 3617 | 2149 |
| 2020-03-06 | 3939 | 2345 |
| 2020-03-07 | 4245 | 2522 |
| 2020-03-08 | 4440 | 2694 |
| 2020-03-09 | 4583 | 2799 |
| 2020-03-10 | 4661 | 2852 |
| 2020-03-11 | 4808 | 2947 |
| 2020-03-12 | 4875 | 2994 |
| 2020-03-13 | 4936 | 3043 |
| 2020-03-14 | 5986 | 3100 |
| 2020-03-15 | 5026 | 3136 |
| 2020-03-16 | 5067 | 3169 |
| 2020-03-17 | 5120 | 3200 |
| 2020-03-18 | 5173 | 3240 |
| 2020-03-19 | 5269 | 3296 |
| 2020-03-20 | 5322 | 3330 |
| 2020-03-21 | 5412 | 3387 |
| 2020-03-22 | 5467 | 3430 |
| 2020-03-23 | 5504 | 3457 |
| 2020-03-24 | 5540 | 3497 |
| 2020-03-25 | 5587 | 3550 |
| 2020-03-26 | 5643 | 3598 |
| 2020-03-27 | 5694 | 3638 |
| 2020-03-28 | 5742 | 3736 |
| 2020-03-29 | 5784 | 3799 |
| 2020-03-30 | 5827 | 3836 |
| 2020-03-31 | 5881 | 3905 |
| 2020-04-01 | 5941 | 3946 |

```python
plt.xticks(rotation=70) # x 축 방향 기울이기(기울이지 않으면 겹침)

plt.bar(sex.index,
        sex.loc[:,'Female'],
        color='lightcoral', label='Female')

plt.bar(sex.index,
        sex.loc[:,'Male'],
        color='lightskyblue', label='Male')

plt.show()
# 3월 14일의 데이터의 확인이 필요
```

Python

Out[-]



```python
confirmed['Female']['2020-03-14'] = 5000
```

Python

```python
# 2020년 3월 1일 ~ 4월 1일 시계열 데이터(수정)
sex = confirmed.iloc[:, :2]
sex
```

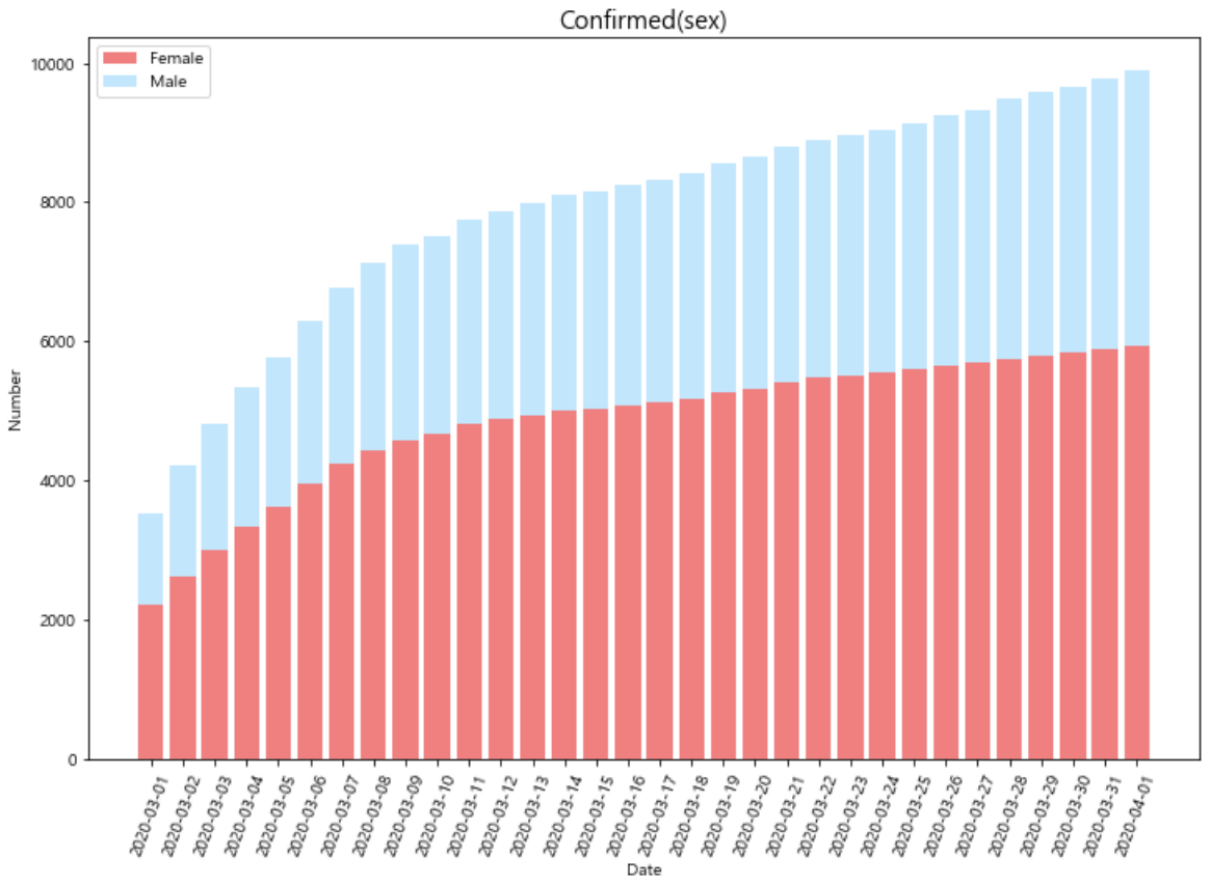
Python

```
Out[-]
           Female  Male
     Date
2020-03-01  2197  1329
2020-03-02  2621  1591
2020-03-03  3002  1810
2020-03-04  3332  1996
2020-03-05  3617  2149
2020-03-06  3939  2345
2020-03-07  4245  2522
2020-03-08  4440  2694
2020-03-09  4583  2799
2020-03-10  4661  2852
2020-03-11  4808  2947
2020-03-12  4875  2994
2020-03-13  4936  3043
2020-03-14  5000  3100 # 변경 후 (5986 -> 5000)
2020-03-15  5026  3136
2020-03-16  5067  3169
2020-03-17  5120  3200
2020-03-18  5173  3240
2020-03-19  5269  3296
2020-03-20  5322  3330
2020-03-21  5412  3387
2020-03-22  5467  3430
2020-03-23  5504  3457
2020-03-24  5540  3497
2020-03-25  5587  3550
2020-03-26  5643  3598
2020-03-27  5694  3638
2020-03-28  5742  3736
2020-03-29  5784  3799
2020-03-30  5827  3836
2020-03-31  5881  3905
2020-04-01  5941  3946
```

```
plt.xticks(rotation=70)
plt.bar(sex.index,
        sex.loc[:,'Female'],
        color='lightcoral', label='Female')
plt.bar(sex.index,
        sex.loc[:,'Male'],
        color='lightskyblue',
        label='Male',
        bottom = sex.loc[:,'Female'],
        alpha = 0.5)

plt.xlabel('Date')
plt.ylabel('Number')
plt.title('Confirmed(sex)', size=15)
plt.legend()

plt.show()
```

Python

Out[-]



## Plotly

```
sex = confirmed.iloc[-1, :2]
print(sex)
print(type(sex))
```

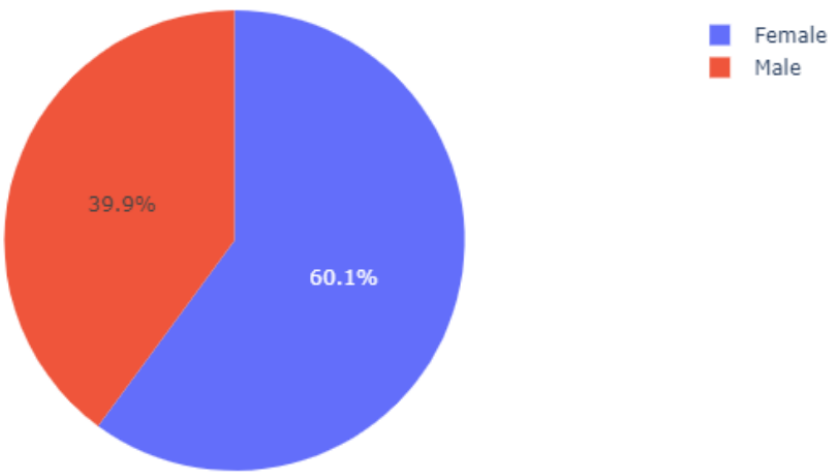
Python

```
Out[-]
Female    5941
Male      3946
Name: 2020-04-01, dtype: int64
<class 'pandas.core.series.Series'>
```

```
import plotly.graph_objects as go

labels = sex.index
values = sex.values

fig = go.Figure(data=[go.Pie(labels=labels,
                             values=values)])

fig.show()
```

Python

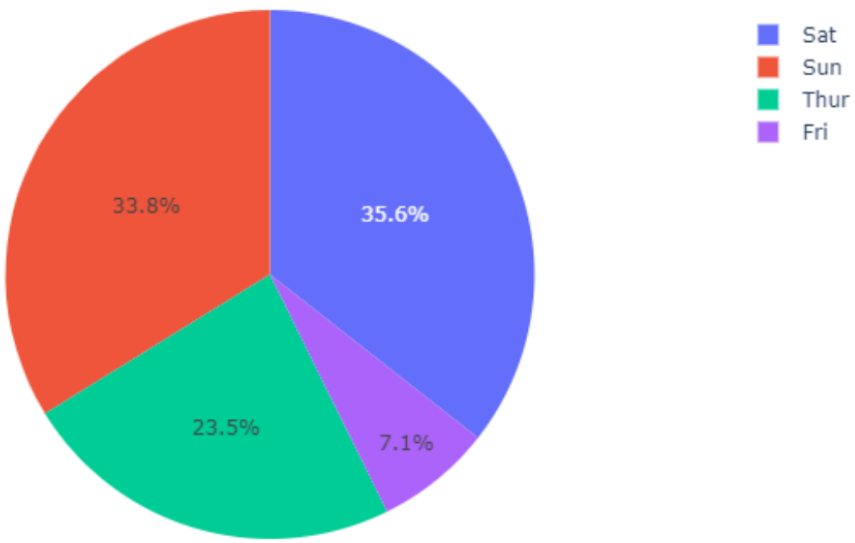
Out[-]



```python
# 공식홈페이지 튜토리얼, 보통은 df으로 그립니다.
import plotly.express as px
# tips데이터는 244 x 7 로 이루어져 있고 day는 4개의 고유값이 있다.
df = px.data.tips()

fig = px.pie(df,
             values='tip', names='day')

fig.show()
```

Python

Out[-]



```
sex = confirmed.iloc[:, :2]
sex
```

Python

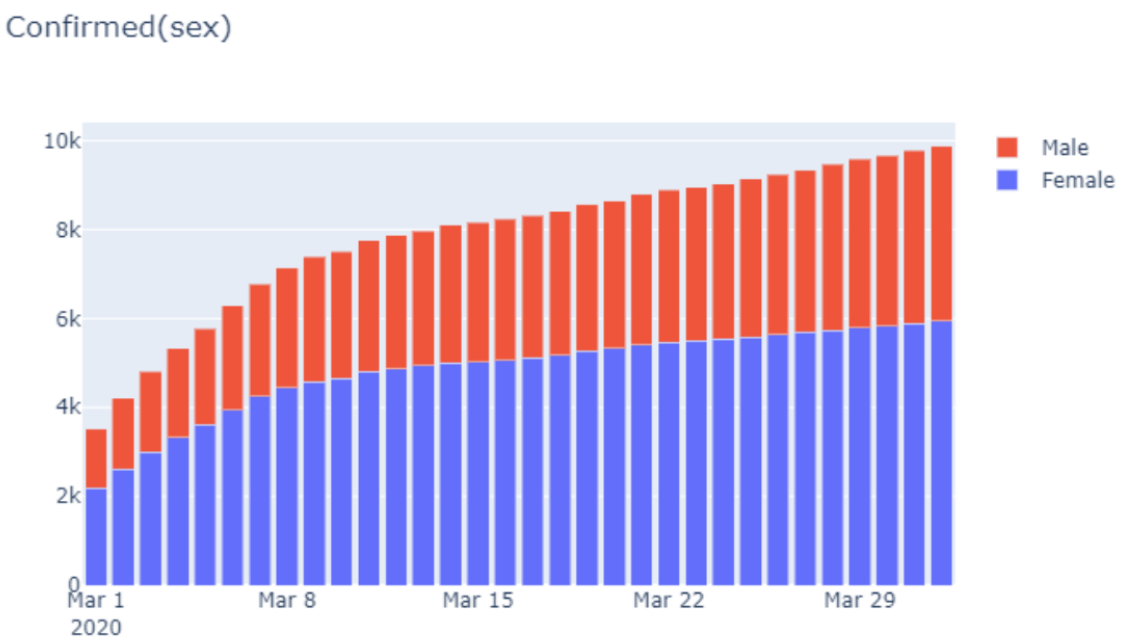
```
Out[-]
            Female  Male
     Date
2020-03-01  2197  1329
2020-03-02  2621  1591
2020-03-03  3002  1810
2020-03-04  3332  1996
2020-03-05  3617  2149
2020-03-06  3939  2345
2020-03-07  4245  2522
2020-03-08  4440  2694
2020-03-09  4583  2799
2020-03-10  4661  2852
2020-03-11  4808  2947
2020-03-12  4875  2994
2020-03-13  4936  3043
2020-03-14  5000  3100
2020-03-15  5026  3136
2020-03-16  5067  3169
2020-03-17  5120  3200
2020-03-18  5173  3240
2020-03-19  5269  3296
2020-03-20  5322  3330
2020-03-21  5412  3387
2020-03-22  5467  3430
2020-03-23  5504  3457
2020-03-24  5540  3497
2020-03-25  5587  3550
2020-03-26  5643  3598
2020-03-27  5694  3638
2020-03-28  5742  3736
2020-03-29  5784  3799
2020-03-30  5827  3836
2020-03-31  5881  3905
2020-04-01  5941  3946
```

Python

```
female = go.Bar(x=sex.index,
                y=sex.iloc[:,0], name='Female')

male = go.Bar(x=sex.index,
              y=sex.iloc[:,1], name='Male')

data = female, male
# data
layout = go.Layout(title='Confirmed(sex)', barmode='stack')
fig = go.Figure(data=data, layout=layout)

fig.show()
```

Python

Out[-]



```
age = confirmed.iloc[-1, 2:]
age
```

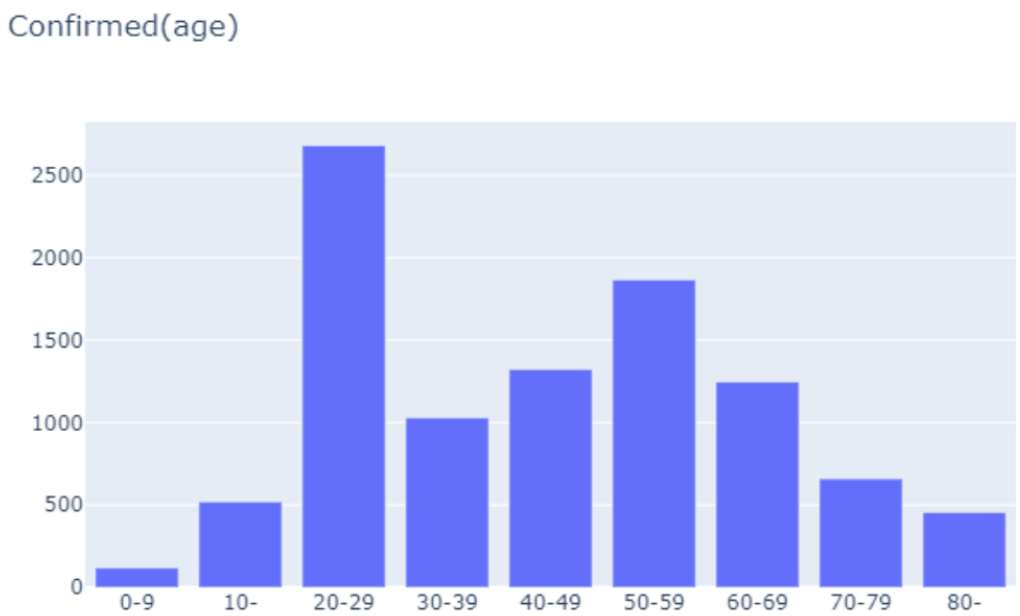
Python

```
Out[-]
0-9       116
10-       519
20-29    2682
30-39    1027
40-49    1323
50-59    1865
60-69    1245
70-79     658
80-       452
Name: 2020-04-01, dtype: int64
```

```
# 4월 1일 나이별 확진자
# bar 그래프

data = [go.Bar(x=age.index, y=age.values)]
layout = go.Layout(title='Confirmed(age)')
fig = go.Figure(data=data, layout=layout)

fig.show()
```

Python

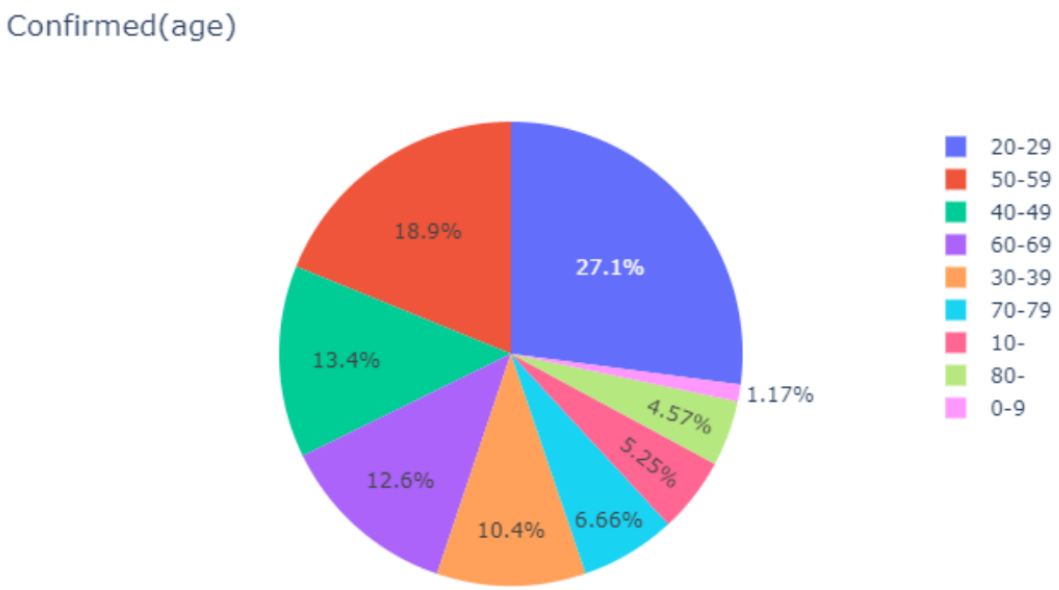
Out[-]



```python
# 4월 1일 나이별 확진자
# bar 그래프

data = [go.Pie(labels=age.index, values=age.values)]
layout = go.Layout(title='Confirmed(age)')
fig = go.Figure(data=data, layout=layout)

fig.show()
```

Python

Out[-]



```python
# 치명률
# 4월 1일 기준
# 치명률 = 사망자수/확진자수 * 100

death = pd.read_csv('./data/daily_Deceased.csv', index_col = ['Date']) # 사망자
death.tail()
```

Python

Out[-]

| Date | Female | Male | 0-9 | 10- | 20-29 | 30-39 | 40-49 | 50-59 | 60-69 | 70-79 | 80- |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2020-03-28 | 67 | 77 | 0 | 0 | 0 | 1 | 1 | 10 | 21 | 41 | 70 |
| 2020-03-29 | 74 | 78 | 0 | 0 | 0 | 1 | 1 | 10 | 21 | 43 | 76 |
| 2020-03-30 | 78 | 80 | 0 | 0 | 0 | 1 | 1 | 10 | 21 | 45 | 80 |
| 2020-03-31 | 80 | 82 | 0 | 0 | 0 | 1 | 1 | 10 | 22 | 46 | 82 |
| 2020-04-01 | 81 | 84 | 0 | 0 | 0 | 1 | 1 | 10 | 23 | 46 | 84 |

Python

```python
confirmed_temp = confirmed.iloc[-1, :]
death_temp = death.iloc[-1, :]
```

Python

```python
confirmed_temp
```

Python

Out[-]

```
Female    5941
Male      3946
0-9        116
10-        519
20-29     2682
30-39     1027
40-49     1323
50-59     1865
60-69     1245
70-79      658
80-        452
Name: 2020-04-01, dtype: int64
```

Python

```
death_temp
```

Python

```
Out[-]
Female    81
Male      84
0-9        0
10-        0
20-29      0
30-39      1
40-49      1
50-59     10
60-69     23
70-79     46
80-       84
Name: 2020-04-01, dtype: int64
```

Python

```
death_rate = round((death_temp/confirmed_temp)*100, 2) # round 소수점 2번째자리까지 반올림
death_rate
```

Python

```
Out[-]
Female     1.36
Male       2.13
0-9        0.00
10-        0.00
20-29      0.00
30-39      0.10
40-49      0.08
50-59      0.54
60-69      1.85
70-79      6.99
80-       18.58
Name: 2020-04-01, dtype: float64
```

```python
sex = confirmed_temp[:2]
sex
```

Python

```
Out[-]
Female    5941
Male      3946
Name: 2020-04-01, dtype: int64
```

Python

```python
sex_death = death_temp[:2]
sex_death
```

Python

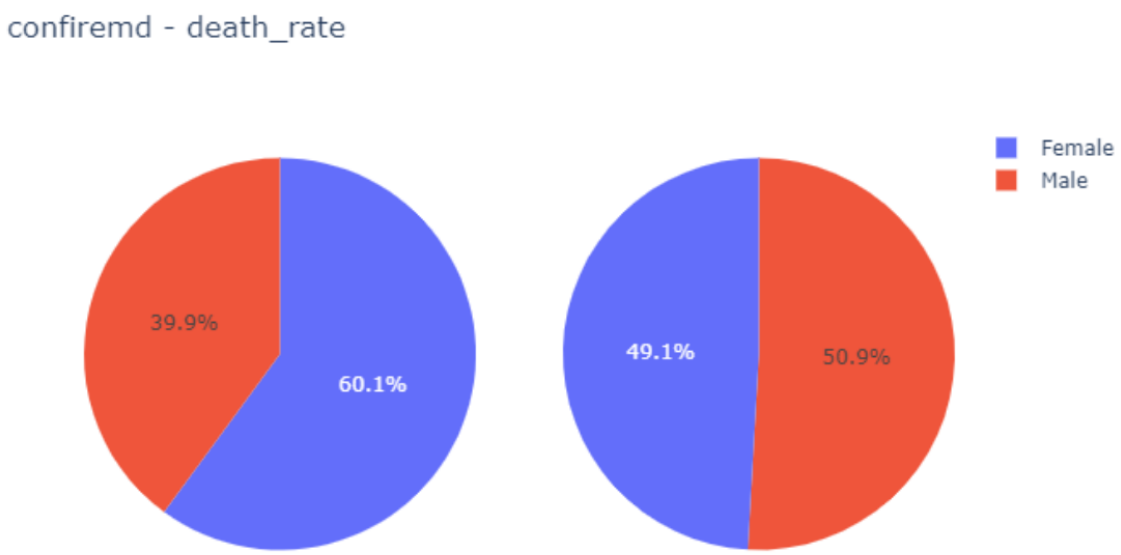
```
Out[-]
Female    81
Male      84
Name: 2020-04-01, dtype: int64
```

Python

```
from plotly.subplots import make_subplots

fig = make_subplots(rows=1,cols=2,specs=[[{'type':'domain'}, {'type':'domain'}]])

labels = sex.index
values = sex.values
labels2 = sex_death.index
values2 = sex_death.values

fig.add_trace(go.Pie(labels=labels,
                     values=values), row=1, col=1)

fig.add_trace(go.Pie(labels=labels2,
                     values=values2), row=1, col=2)
fig.update_layout(title='confiremd - death_rate')

fig.show()
```

Python

Out[-]



```
age_confirmed = confirmed.iloc[-1, 2:]
age_confirmed
```

Python

```
Out[-]
0-9       116
10-       519
20-29    2682
30-39    1027
40-49    1323
50-59    1865
60-69    1245
70-79     658
80-       452
Name: 2020-04-01, dtype: int64
```

Python

```
age_death = death_rate.iloc[2:]
age_death
```

Python

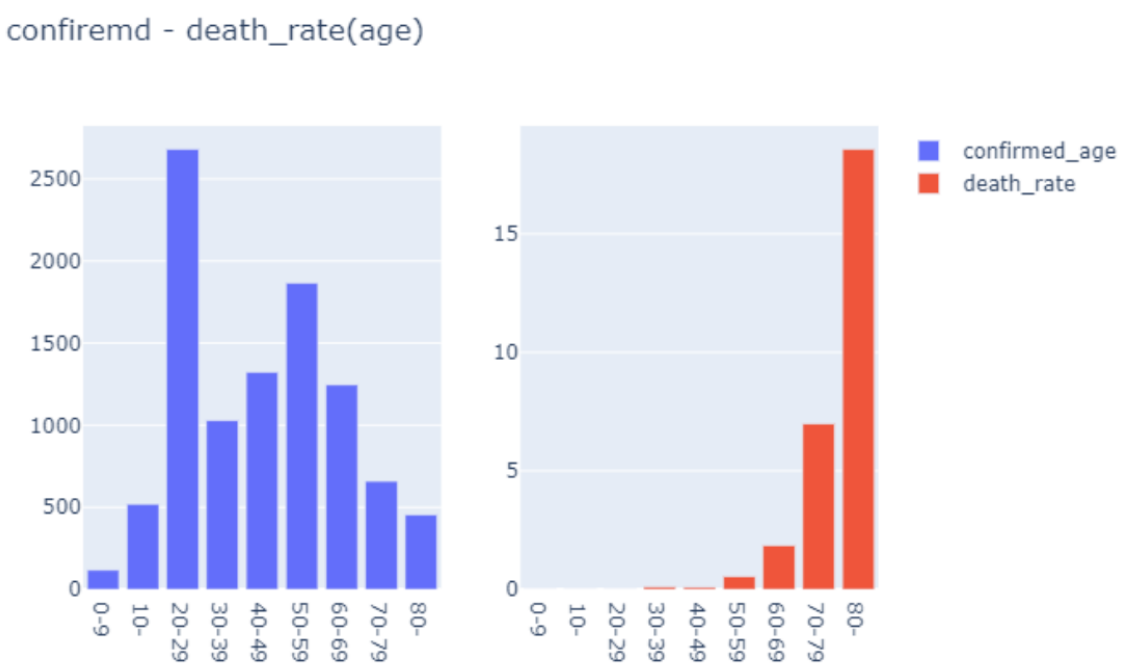
```
Out[-]
0-9       0.00
10-       0.00
20-29     0.00
30-39     0.10
40-49     0.08
50-59     0.54
60-69     1.85
70-79     6.99
80-      18.58
Name: 2020-04-01, dtype: float64
```

Python

```
fig = make_subplots(rows=1,cols=2,specs=[[{'type':'bar'}, {'type':'bar'}]])

labels = age_confirmed.index
values = age_confirmed.values
labels2 = age_death.index
values2 = age_death.values

fig.add_trace(go.Bar(x=labels,
                     y=values, name='confirmed_age'), row=1, col=1)

fig.add_trace(go.Bar(x=labels2,
                     y=values2, name='death_rate'), row=1, col=2)

fig.update_layout(title='confiremd - death_rate(age)')

fig.show()
```

Python

Out[-]



## 2. 코로나 지역별 분석

```
# 데이터 불러오기(출처 : 질병관리본부)

import pandas as pd
from matplotlib import pyplot as plt

region = pd.read_csv("./data/region.csv", index_col=['day'])
```

Python

```
region.head()
```

Python

Out[-]

| day | 서울 | 부산 | 대구 | 인천 | 광주 | 대전 | 울산 | 세종 | 경기 | 강원 | 충북 | 충남 | 전북 | 전남 | 경북 | 경남 | 제주 | 검역 | 총확진자수 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2020-03-01 | 82 | 81 | 2569 | 6 | 9 | 13 | 17 | 1 | 84 | 7 | 11 | 60 | 5 | 3 | 514 | 62 | 2 | 0 | 3526 |
| 2020-03-02 | 91 | 88 | 3081 | 7 | 9 | 14 | 20 | 1 | 92 | 19 | 11 | 78 | 6 | 5 | 624 | 64 | 2 | 0 | 4212 |
| 2020-03-03 | 98 | 90 | 3601 | 7 | 11 | 14 | 20 | 1 | 94 | 20 | 11 | 81 | 7 | 5 | 685 | 64 | 3 | 0 | 4812 |
| 2020-03-04 | 99 | 93 | 4006 | 9 | 13 | 15 | 23 | 1 | 101 | 21 | 11 | 82 | 7 | 5 | 774 | 65 | 3 | 0 | 5328 |
| 2020-03-05 | 103 | 92 | 4326 | 9 | 14 | 16 | 23 | 1 | 110 | 23 | 12 | 86 | 7 | 5 | 861 | 74 | 4 | 0 | 5766 |

Python

```
region.tail()
```

Python

```
Out[-]
            서울  부산  대구  인천  광주  대전  울산  세종  경기  강원  충북  충남  전북
전남  경북  경남  제주  검역  총확진자수
       day
2020-03-28  390   114   6587   51    20    31    39    44   433    32    41   126   10
8   1285   91     8   168     9478
2020-03-29  410   117   6610   58    20    34    39    46   448    34    41   127   12
9   1287   94     8   189     9583
2020-03-30  426   118   6624   58    20    34    39    46   463    36    44   127   13
9   1298   95     9   202     9661
2020-03-31  450   119   6684   64    20    36    39    46   476    36    44   128   13
9   1300   96     9   217     9786
2020-04-01  474   122   6704   69    24    36    39    46   499    38    44   131   14
12    1302  100     9   224     9887
```

Python

```python
region.info()
```

Python

```
Out[-]
<class 'pandas.core.frame.DataFrame'>
Index: 32 entries, 2020-03-01 to 2020-04-01
Data columns (total 19 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   서울      32 non-null     int64
 1   부산      32 non-null     int64
 2   대구      32 non-null     int64
 3   인천      32 non-null     int64
 4   광주      32 non-null     int64
 5   대전      32 non-null     int64
 6   울산      32 non-null     int64
 7   세종      32 non-null     int64
 8   경기      32 non-null     int64
 9   강원      32 non-null     int64
 10  충북      32 non-null     int64
 11  충남      32 non-null     int64
 12  전북      32 non-null     int64
 13  전남      32 non-null     int64
 14  경북      32 non-null     int64
 15  경남      32 non-null     int64
 16  제주      32 non-null     int64
 17  검역      32 non-null     int64
 18  총확진자수   32 non-null     int64
dtypes: int64(19)
memory usage: 5.0+ KB
```

Python

```python
region.isnull().sum()
```

Python

```
Out[-]
서울        0
부산        0
대구        0
인천        0
광주        0
대전        0
울산        0
세종        0
경기        0
강원        0
충북        0
충남        0
전북        0
전남        0
경북        0
경남        0
제주        0
검역        0
총확진자수     0
dtype: int64
```

Python

```python
pd.options.display.max_rows = 1000
pd.options.display.max_columns = 100
```

Python

```
region.tail()
```

Python

Out[-]

| | 서울 | 부산 | 대구 | 인천 | 광주 | 대전 | 울산 | 세종 | 경기 | 강원 | 충북 | 충남 | 전북 | 전남 | 경북 | 경남 | 제주 | 검역 | 총확진자수 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| day | | | | | | | | | | | | | | | | | | | |
| 2020-03-28 | 390 | 114 | 6587 | 51 | 20 | 31 | 39 | 44 | 433 | 32 | 41 | 126 | 108 | 1285 | 91 | 8 | 168 | | 9478 |
| 2020-03-29 | 410 | 117 | 6610 | 58 | 20 | 34 | 39 | 46 | 448 | 34 | 41 | 127 | 129 | 1287 | 94 | 8 | 189 | | 9583 |
| 2020-03-30 | 426 | 118 | 6624 | 58 | 20 | 34 | 39 | 46 | 463 | 36 | 44 | 127 | 139 | 1298 | 95 | 9 | 202 | | 9661 |
| 2020-03-31 | 450 | 119 | 6684 | 64 | 20 | 36 | 39 | 46 | 476 | 36 | 44 | 128 | 139 | 1300 | 96 | 9 | 217 | | 9786 |
| 2020-04-01 | 474 | 122 | 6704 | 69 | 24 | 36 | 39 | 46 | 499 | 38 | 44 | 131 | 14 | 12 | 1302 | 100 | 9 | 224 | 9887 |

Python

```python
last = region.iloc[-1,:]
last
```

Python

```
Out[-]
서울         474
부산         122
대구        6704
인천          69
광주          24
대전          36
울산          39
세종          46
경기         499
강원          38
충북          44
충남         131
전북          14
전남          12
경북        1302
경남         100
제주           9
검역         224
총확진자수     9887
Name: 2020-04-01, dtype: int64
```

Python

```python
last = region.iloc[-1,:-2]
last
```

Python

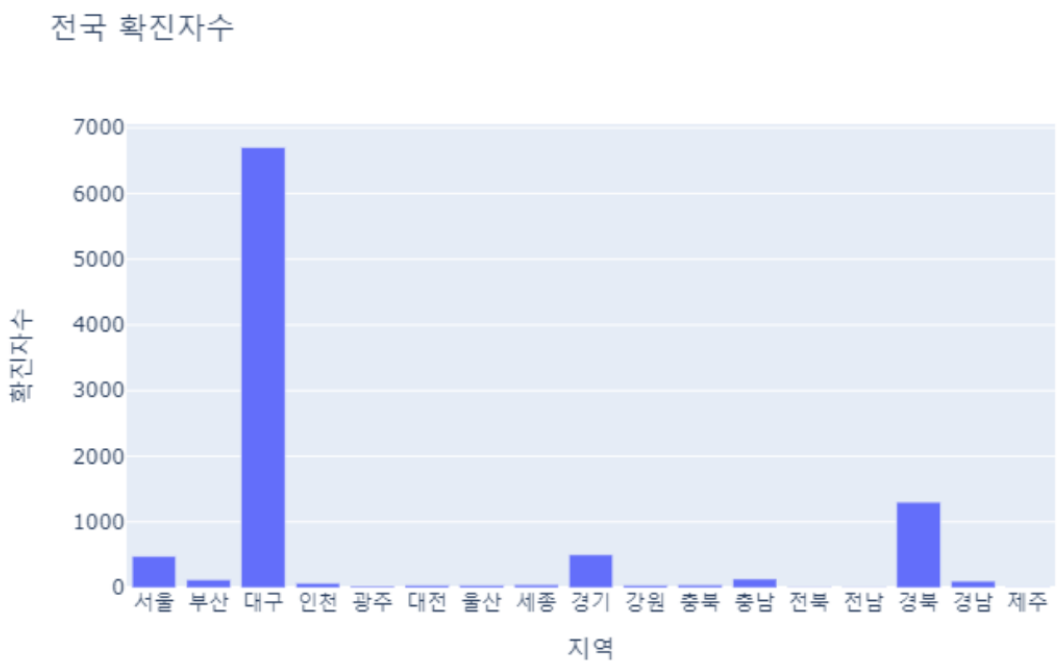
```
Out[-]
서울     474
부산     122
대구    6704
인천      69
광주      24
대전      36
울산      39
세종      46
경기     499
강원      38
충북      44
충남     131
전북      14
전남      12
경북    1302
경남     100
제주       9
Name: 2020-04-01, dtype: int64
```

Python

```
location_data = go.Bar(x=last.index, y=last.values)
layout = go.Layout(title='전국 확진자수', xaxis_title='지역', yaxis_title='확진자수')
fig = go.Figure(data=location_data, layout=layout)
fig.show()
```

Python

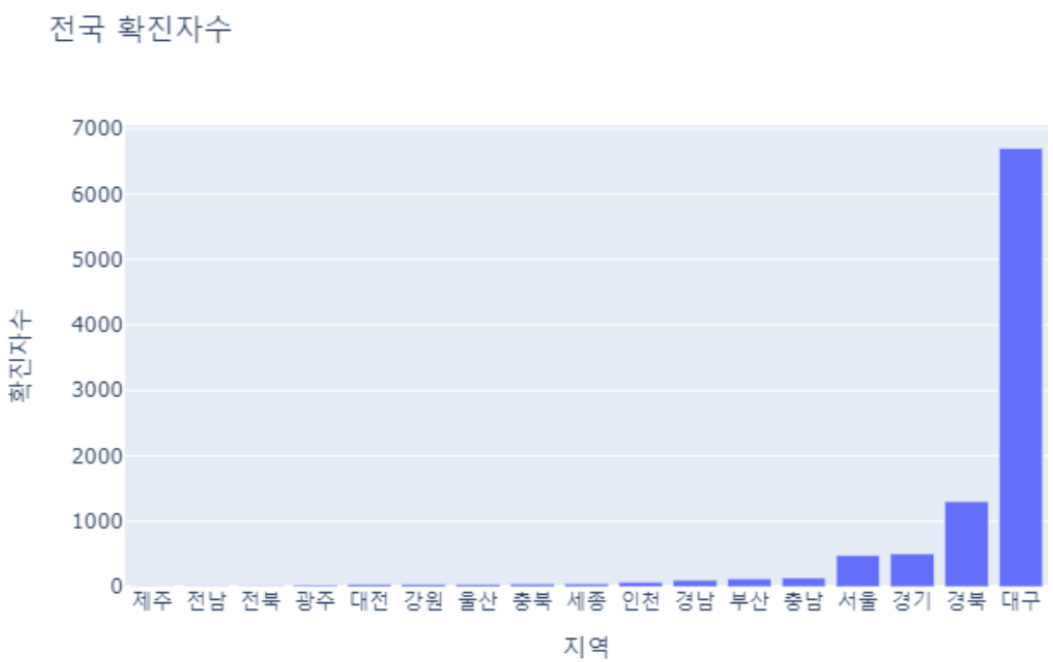
Out[-]



```python
last = last.sort_values(ascending=True)

# ascending 오름차순
# 낮은 것부터 차례로 배열

# descending 내림차순
# 높은 것부터 차례로 배열

location_data = go.Bar(x=last.index, y=last.values)
layout = go.Layout(title='전국 확진자수', xaxis_title='지역', yaxis_title='확진자수')
fig = go.Figure(data=location_data, layout=layout)
fig.show()
```

Python

Out[-]



```python
region.index
```

Python

```
Out[-]
Index(['2020-03-01', '2020-03-02', '2020-03-03', '2020-03-04', '2020-03-05',
       '2020-03-06', '2020-03-07', '2020-03-08', '2020-03-09', '2020-03-10',
       '2020-03-11', '2020-03-12', '2020-03-13', '2020-03-14', '2020-03-15',
       '2020-03-16', '2020-03-17', '2020-03-18', '2020-03-19', '2020-03-20',
       '2020-03-21', '2020-03-22', '2020-03-23', '2020-03-24', '2020-03-25',
       '2020-03-26', '2020-03-27', '2020-03-28', '2020-03-29', '2020-03-30',
       '2020-03-31', '2020-04-01'],
      dtype='object', name='day')
```

Python

```python
region.iloc[:,2]
```

Python

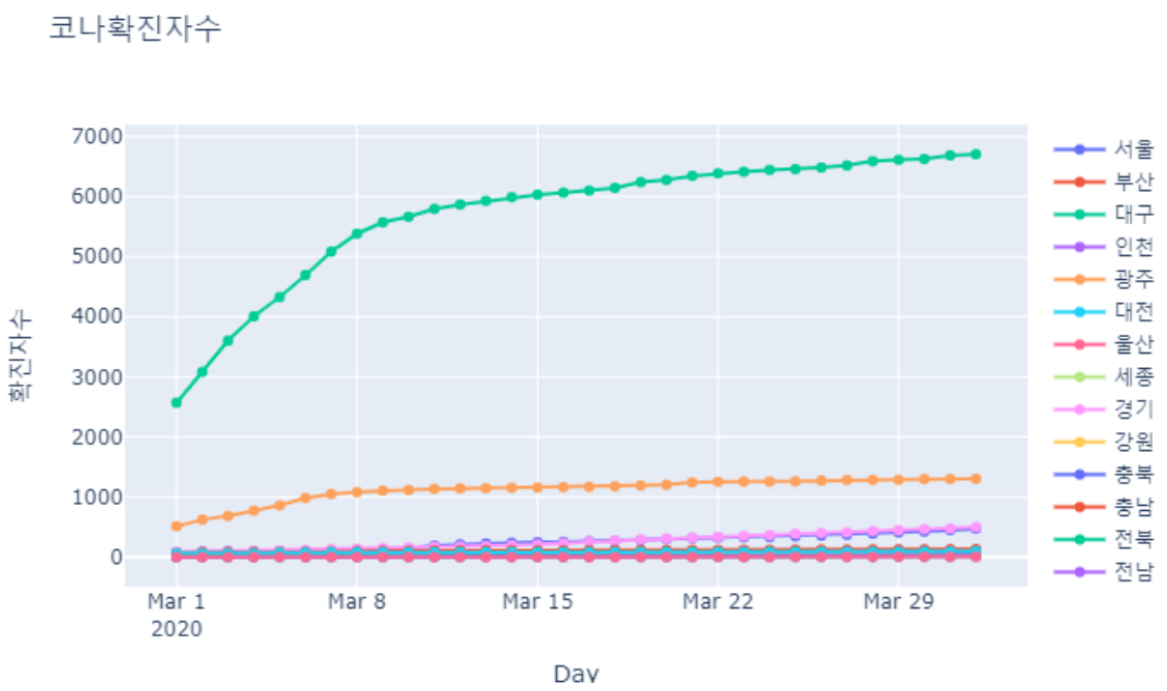
```
Out[-]
day
2020-03-01    2569
2020-03-02    3081
2020-03-03    3601
2020-03-04    4006
2020-03-05    4326
2020-03-06    4693
2020-03-07    5084
2020-03-08    5381
2020-03-09    5571
2020-03-10    5663
2020-03-11    5794
2020-03-12    5867
2020-03-13    5928
2020-03-14    5990
2020-03-15    6031
2020-03-16    6066
2020-03-17    6098
2020-03-18    6144
2020-03-19    6241
2020-03-20    6275
2020-03-21    6344
2020-03-22    6387
2020-03-23    6411
2020-03-24    6442
2020-03-25    6456
2020-03-26    6482
2020-03-27    6516
2020-03-28    6587
2020-03-29    6610
2020-03-30    6624
2020-03-31    6684
2020-04-01    6704
Name: 대구, dtype: int64
```

Python

```python
# 전체 지역 그래프 그리기
import plotly.graph_objects as go
fig = go.Figure()
for i in range(len(last)) :
    fig.add_trace(go.Scatter(x=region.index, y=region.iloc[:,i], mode='lines+markers', na
me = region.columns[i]))
fig.update_layout(title='코나확진자수', xaxis_title='Day', yaxis_title='확진자수')
fig.show()
```

Python

Out[-]



```python
last = region.iloc[-1, :18]
last
```

Python

```
Out[-]
서울     474
부산     122
대구    6704
인천      69
광주      24
대전      36
울산      39
세종      46
경기     499
강원      38
충북      44
충남     131
전북      14
전남      12
경북    1302
경남     100
제주       9
검역     224
Name: 2020-04-01, dtype: int64
```

Python

```python
last.index
```

Python

```
Out[-]
Index(['서울', '부산', '대구', '인천', '광주', '대전', '울산', '세종', '경기', '강원', '충북',
'충남',
       '전북', '전남', '경북', '경남', '제주', '검역'],
      dtype='object')
```

Python

```
last.values
```

Python

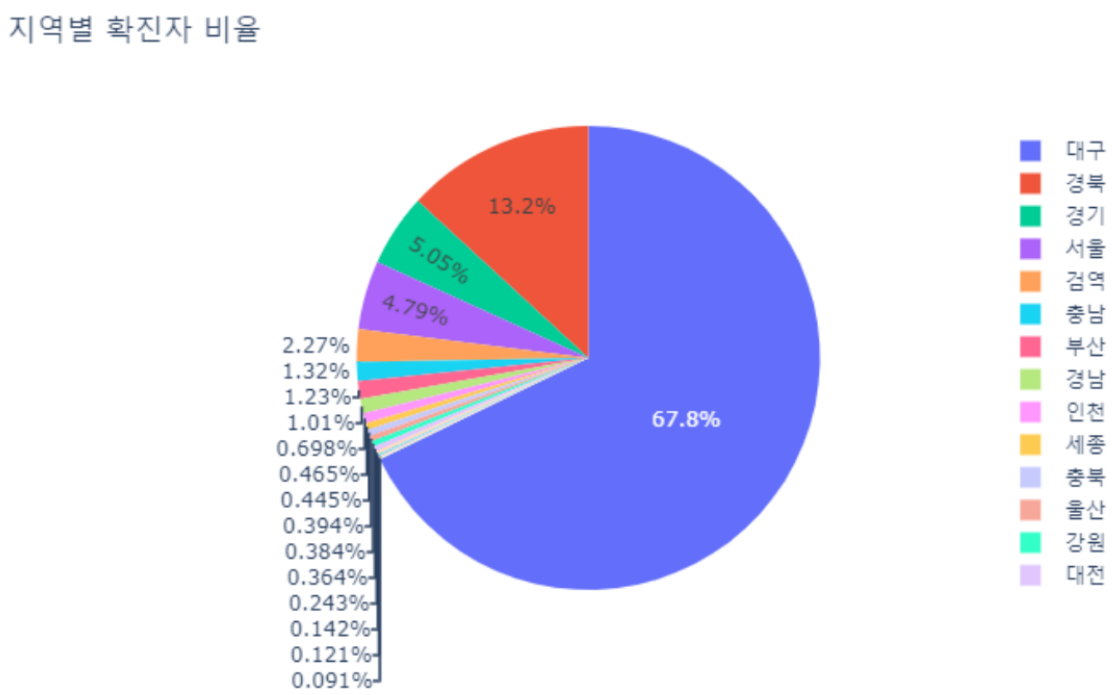
```
Out[-]
array([ 474,  122, 6704,   69,   24,   36,   39,   46,  499,   38,   44,
        131,   14,   12, 1302,  100,    9,  224], dtype=int64)
```

Python

```
labels = last.index
values = last.values

fig = go.Figure(data=[go.Pie(labels=labels, values=values)])
fig.update_layout(title="지역별 확진자 비율")

fig.show()
```

Python

Out[-]



```python
region.iloc[:, 16]
```

Python

```
Out[-]
day
2020-03-01    2
2020-03-02    2
2020-03-03    3
2020-03-04    3
2020-03-05    4
2020-03-06    4
2020-03-07    4
2020-03-08    4
2020-03-09    4
2020-03-10    4
2020-03-11    4
2020-03-12    4
2020-03-13    4
2020-03-14    4
2020-03-15    4
2020-03-16    4
2020-03-17    4
2020-03-18    4
2020-03-19    4
2020-03-20    4
2020-03-21    4
2020-03-22    4
2020-03-23    4
2020-03-24    4
2020-03-25    6
2020-03-26    6
2020-03-27    7
2020-03-28    8
2020-03-29    8
2020-03-30    9
2020-03-31    9
2020-04-01    9
Name: 제주, dtype: int64
```

Python

```python
# region.index
index = region.columns
list(index).index('제주')
```

Python

```
Out[-]
16
```

```python
jeju = region.iloc[:, 16]
jeju
```

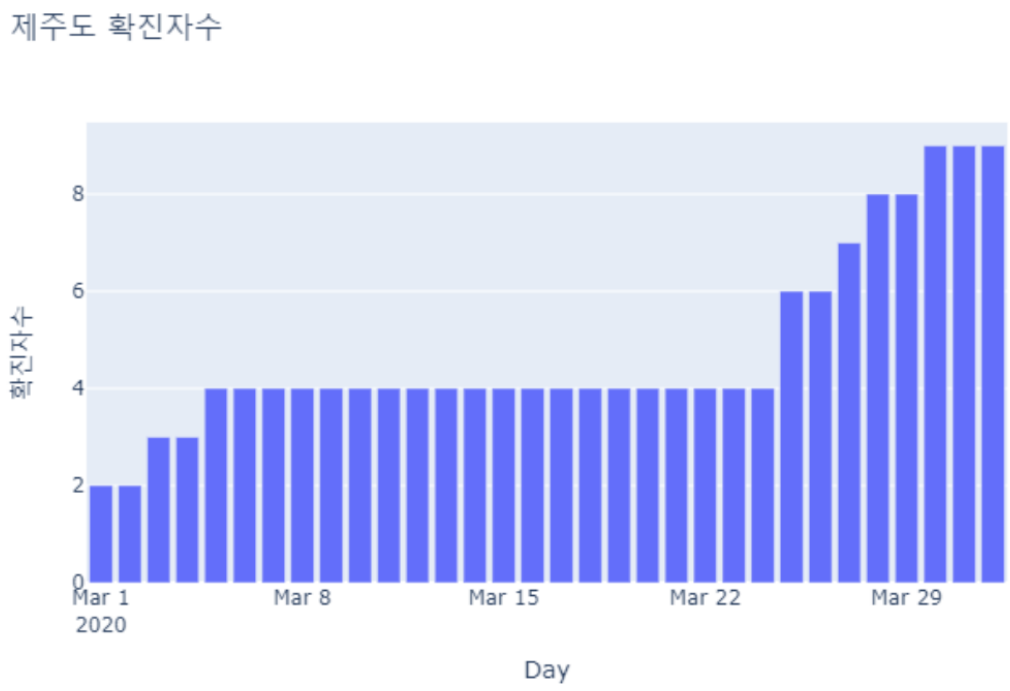
Python

```
Out[-]
day
2020-03-01    2
2020-03-02    2
2020-03-03    3
2020-03-04    3
2020-03-05    4
2020-03-06    4
2020-03-07    4
2020-03-08    4
2020-03-09    4
2020-03-10    4
2020-03-11    4
2020-03-12    4
2020-03-13    4
2020-03-14    4
2020-03-15    4
2020-03-16    4
2020-03-17    4
2020-03-18    4
2020-03-19    4
2020-03-20    4
2020-03-21    4
2020-03-22    4
2020-03-23    4
2020-03-24    4
2020-03-25    6
2020-03-26    6
2020-03-27    7
2020-03-28    8
2020-03-29    8
2020-03-30    9
2020-03-31    9
2020-04-01    9
Name: 제주, dtype: int64
```

```python
# plotly
data = go.Bar(x=jeju.index, y=jeju.values,)
layout = go.Layout(title='제주도 확진자수',xaxis_title='Day', yaxis_title='확진자수')
fig = go.Figure(data=data, layout=layout)
fig.show()
```

Python

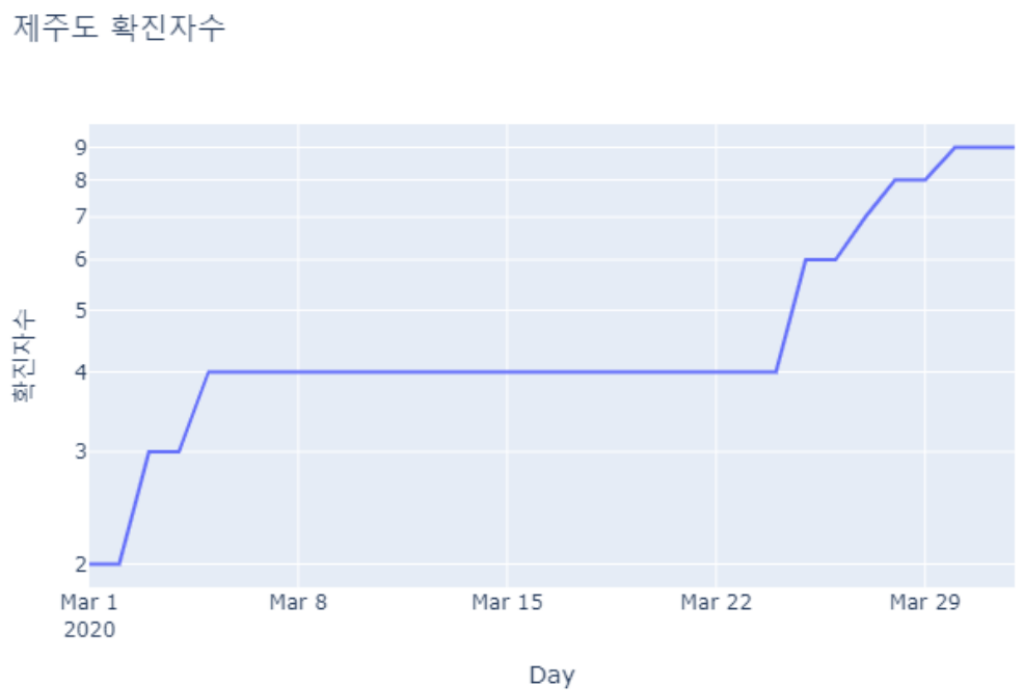
Out[-]



```python
# plotly semi log 그래프
data = go.Line(x=jeju.index, y=jeju.values,)

layout = go.Layout(title='제주도 확진자수',
                   xaxis_title='Day',
                   yaxis_title='확진자수',
                   yaxis_type='log',
                  )

fig = go.Figure(data=data, layout=layout)
fig.show()
```

Python

Out[-]



## Map

```python
import pandas as pd

data = pd.read_csv("./data/region.csv", index_col=['day'])
data.tail()
```

Python

Out[-]

| day | 서울 | 부산 | 대구 | 인천 | 광주 | 대전 | 울산 | 세종 | 경기 | 강원 | 충북 | 충남 | 전북 | 전남 | 경북 | 경남 | 제주 | 검역 | 총확진자수 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2020-03-28 | 390 | 114 | 6587 | 51 | 20 | 31 | 39 | 44 | 433 | 32 | 41 | 126 | 108 | 1285 | 91 | 8 | 168 | 9478 | |
| 2020-03-29 | 410 | 117 | 6610 | 58 | 20 | 34 | 39 | 46 | 448 | 34 | 41 | 127 | 129 | 1287 | 94 | 8 | 189 | 9583 | |
| 2020-03-30 | 426 | 118 | 6624 | 58 | 20 | 34 | 39 | 46 | 463 | 36 | 44 | 127 | 139 | 1298 | 95 | 9 | 202 | 9661 | |
| 2020-03-31 | 450 | 119 | 6684 | 64 | 20 | 36 | 39 | 46 | 476 | 36 | 44 | 128 | 139 | 1300 | 96 | 9 | 217 | 9786 | |
| 2020-04-01 | 474 | 122 | 6704 | 69 | 24 | 36 | 39 | 46 | 499 | 38 | 44 | 131 | 1412 | 1302 | 100 | 9 | 224 | 9887 | |

Python

```python
map_data = data.iloc[-1, :17]
map_data = pd.DataFrame(map_data)
map_data
```

Python

```
Out[-]
    2020-04-01
서울         474
부산         122
대구        6704
인천          69
광주          24
대전          36
울산          39
세종          46
경기         499
강원          38
충북          44
충남         131
전북          14
전남          12
경북        1302
경남         100
제주           9
```

Python

```python
loc = {
  '서울' : [37.566418, 126.977950],#서울시청
  '부산' : [35.180152, 129.074980],#부산시청
  '대구' : [35.871468, 128.601757],#대구시청
  '인천' : [37.456445, 126.705873],#인천시청
  '광주' : [35.160068, 126.851426],#광주광역시청
  '대전' : [36.350664, 127.384819],#대전시청
  '울산' : [35.539772, 129.311486],#울산시청
  '세종' : [36.480838, 127.289181],#세종시청
  '경기' : [37.275221, 127.009382],#경기도청
  '강원' : [37.885300, 127.729835],#강원(강원도청)
  '충북' : [36.635947, 127.491345],#충북도청
  '충남' : [36.658826, 126.672849],#충남도청
  '전북' : [35.820599, 127.108759],#전북도청
  '전남' : [34.816351, 126.462924],#전남도청
  '경북' : [36.574108, 128.509303],#경북도청
  '경남' : [35.238398, 128.692371],#경남도청
  '제주' : [33.3617007, 126.511657]#제주
    }
type(loc)
```

Python

```
Out[-]
dict
```

Python

```python
# 위도(latitude)와 경도(longitude)

loc = pd.DataFrame(loc).T
loc.columns = ['lat', 'lon']
loc
```

Python

```
Out[-]
            lat        lon
서울   37.566418 126.977950
부산   35.180152 129.074980
대구   35.871468 128.601757
인천   37.456445 126.705873
광주   35.160068 126.851426
대전   36.350664 127.384819
울산   35.539772 129.311486
세종   36.480838 127.289181
경기   37.275221 127.009382
강원   37.885300 127.729835
충북   36.635947 127.491345
충남   36.658826 126.672849
전북   35.820599 127.108759
전남   34.816351 126.462924
경북   36.574108 128.509303
경남   35.238398 128.692371
제주   33.361701 126.511657
```

Python

```
!pip install folium
```

Python

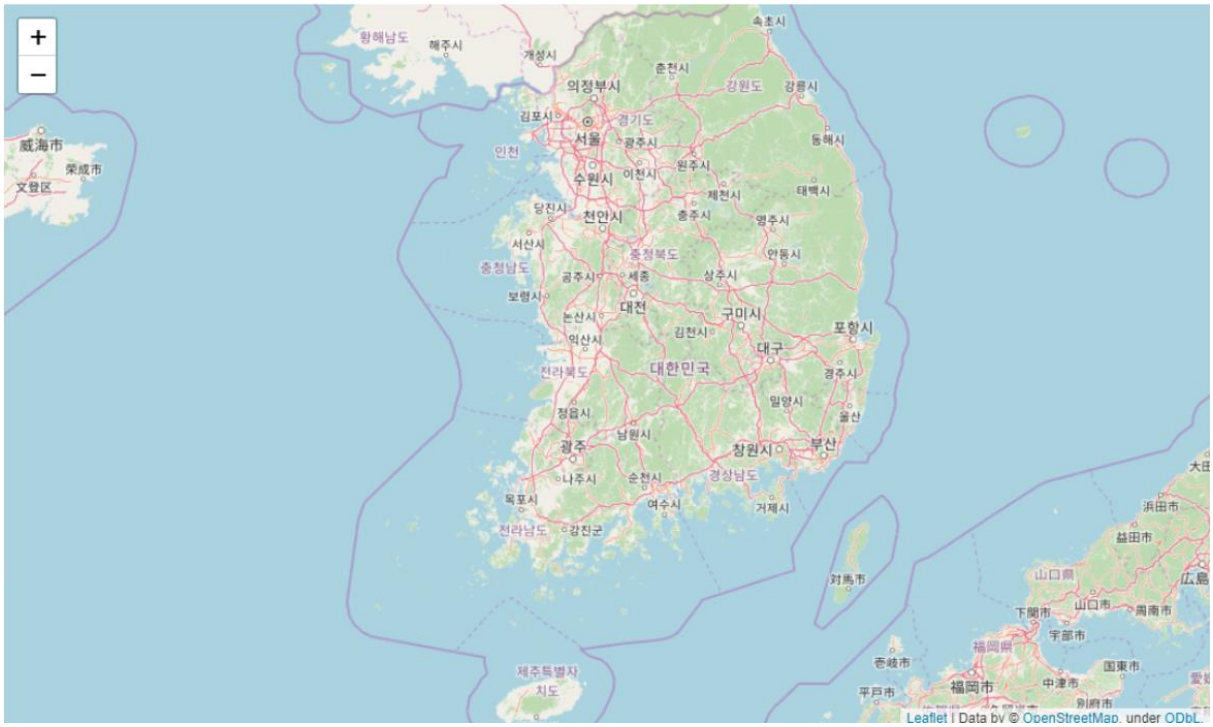
```
Out[-]
Collecting folium
  Downloading folium-0.10.1-py2.py3-none-any.whl (91 kB)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from
 folium) (1.16.4)
Collecting branca>=0.3.0
  Downloading branca-0.4.0-py3-none-any.whl (25 kB)
Requirement already satisfied: requests in c:\programdata\anaconda3\lib\site-packages (fr
om folium) (2.22.0)
Requirement already satisfied: jinja2>=2.9 in c:\programdata\anaconda3\lib\site-packages
(from folium) (2.10.1)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from br
anca>=0.3.0->folium) (1.12.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\programdata
\anaconda3\lib\site-packages (from requests->folium) (1.24.2)
Requirement already satisfied: idna<2.9,>=2.5 in c:\programdata\anaconda3\lib\site-packag
es (from requests->folium) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\programdata\anaconda3\lib\site
-packages (from requests->folium) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\site-pa
ckages (from requests->folium) (2019.6.16)
Requirement already satisfied: MarkupSafe>=0.23 in c:\programdata\anaconda3\lib\site-pack
ages (from jinja2>=2.9->folium) (1.1.1)
Installing collected packages: branca, folium
Successfully installed branca-0.4.0 folium-0.10.1
```

Python

```
import matplotlib.pyplot as plt
import folium

map_osm = folium.Map(location=[35.824,127.147], zoom_start=7)
map_osm
```

Python

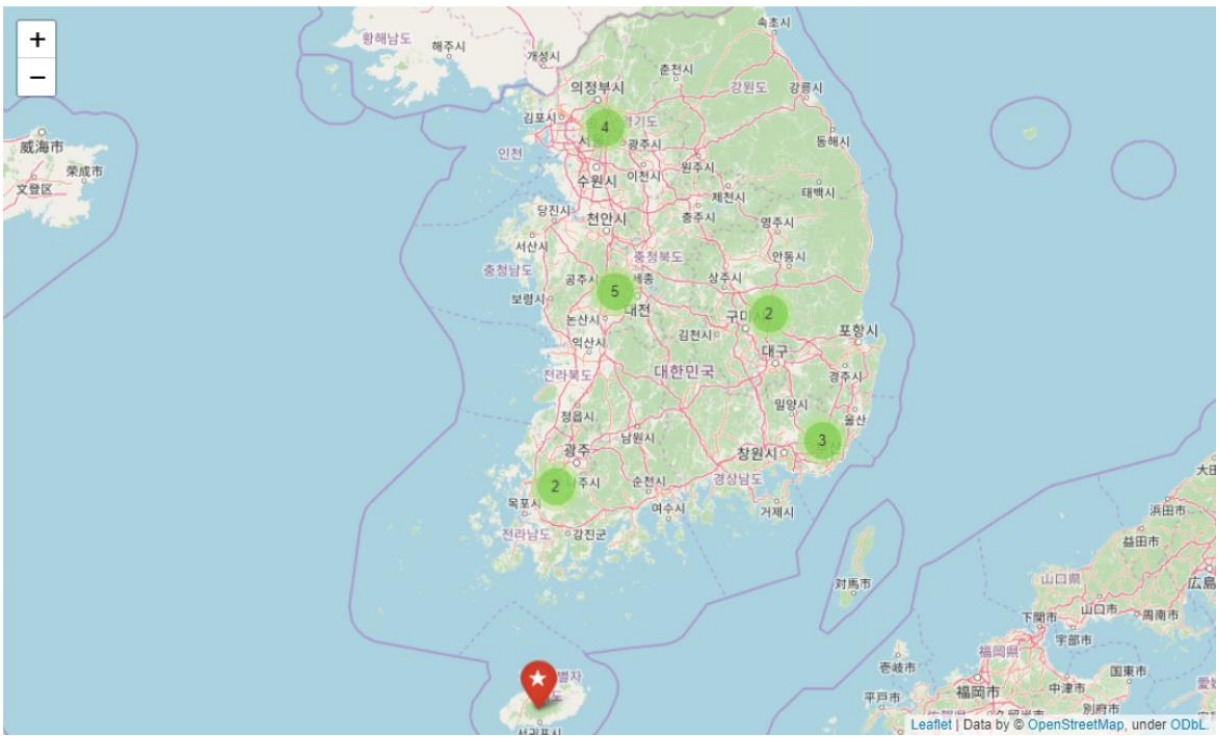
Out[-]



```python
from folium.plugins import MarkerCluster
marker_cluster = MarkerCluster().add_to(map_osm)

for i in range(17):
    folium.Marker(
        [loc.iloc[i:i+1,0], loc.iloc[i:i+1,1]],
        popup = map_data[i:i+1],
        icon=folium.Icon(color='red',icon='star'),
    ).add_to(marker_cluster)
map_osm.save("map.html") # jupyter notebook에 한글 깨져서 html로 저장
map_osm
```

Python

Out[-]



```
map_data.iloc[1]
```

Python

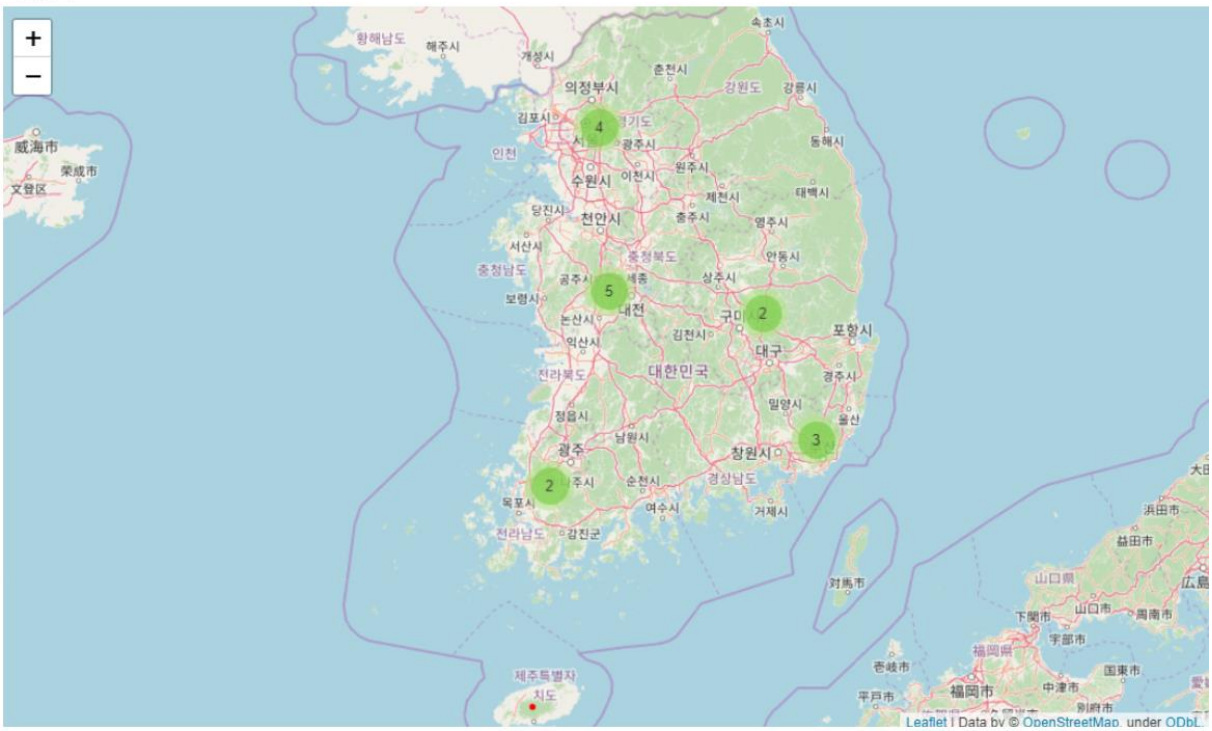
```
Out[-]
2020-04-01    122
Name: 부산, dtype: int64
```

Python

```
from folium.plugins import MarkerCluster

map_osm = folium.Map(location=[35.824,127.147], zoom_start=7)
marker_cluster = MarkerCluster().add_to(map_osm)

for i in range(17):
    folium.Circle(
        [loc.iloc[i:i+1,0], loc.iloc[i:i+1,1]],
        tooltip = map_data[i:i+1],
        radius = int(map_data.iloc[i])*10, ## 확진자 수 * 20배
        fill_color='red',
        color='red'
    ).add_to(marker_cluster)
map_osm.save("map.html")
map_osm
```

Python

Out[-]



## 옵션값

공식홈페이지(https://python-visualization.github.io/folium/)

- stroke (Bool, True) :
  Whether to draw stroke along the path. Set it to false to disable borders on polygons or circles.
- color (str, '#3388ff') :
  Stroke color.
- weight (int, 3) :
  Stroke width in pixels.
- opacity (float, 1.0) :
  Stroke opacity.
- line_cap (str, 'round' (lineCap)) :
  A string that defines shape to be used at the end of the stroke.
- line_join (str, 'round' (lineJoin)) :
  A string that defines shape to be used at the corners of the stroke.
- dash_array (str, None (dashArray)) :
  A string that defines the stroke dash pattern. Doesn’t work on Canvas-powered layers in some old browsers.
- dash_offset (str, None (dashOffset)) :
  A string that defines the distance into the dash pattern to start the dash. Doesn’t work on Canvas-powered layers in some old browsers.
- fill (Bool, False) :
  Whether to fill the path with color. Set it to false to disable filling on polygons or circles.
- fill_color (str, default to color (fillColor)) :
  Fill color. Defaults to the value of the color option.
- fill_opacity (float, 0.2 (fillOpacity)) :
  Fill opacity.
- fill_rule (str, 'evenodd' (fillRule)) :
  A string that defines how the inside of a shape is determined.
- bubbling_mouse_events (Bool, True (bubblingMouseEvents)) :
  When true a mouse event on this path will trigger the same event on the map (unless L.DomEvent.stopPropagation is used).

## 3. 코로나19 한국 데이터 분석

```python
# 현재 코로나 검사 현황 데이터(출처 : 질병관리본부)
import pandas as pd
df = pd.read_csv("./data/total.csv", index_col = ["date"])
df.head()
```

Python

Out[-]

| date | 총계 | 확진자 | 격리해제 | 격리중 | 사망 | 검사중 | 결과 음성 |
|---|---|---|---|---|---|---|---|
| 2020-03-01 | 96985 | 3526 | 30 | 3479 | 17 | 32422 | 61037 |
| 2020-03-02 | 109591 | 4212 | 31 | 4159 | 22 | 33799 | 71580 |
| 2020-03-03 | 125851 | 4812 | 34 | 4750 | 28 | 35555 | 85484 |
| 2020-03-04 | 136707 | 5328 | 41 | 5255 | 32 | 28414 | 102965 |
| 2020-03-05 | 146541 | 5766 | 88 | 5643 | 35 | 21810 | 118965 |

Python

```python
df.tail()
```

Python

Out[-]

| date | 총계 | 확진자 | 격리해제 | 격리중 | 사망 | 검사중 | 결과 음성 |
|---|---|---|---|---|---|---|---|
| 2020-03-28 | 387925 | 9478 | 4811 | 4523 | 144 | 16564 | 361883 |
| 2020-03-29 | 394141 | 9583 | 5033 | 4398 | 152 | 15028 | 369530 |
| 2020-03-30 | 395194 | 9661 | 5228 | 4275 | 158 | 13531 | 372002 |
| 2020-03-31 | 410564 | 9786 | 5408 | 4216 | 162 | 16892 | 383886 |
| 2020-04-01 | 421547 | 9887 | 5567 | 4155 | 165 | 16585 | 395075 |

Python

```python
df.info()
```

Python

```
Out[-]
<class 'pandas.core.frame.DataFrame'>
Index: 32 entries, 2020-03-01 to 2020-04-01
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   총계      32 non-null     int64
 1   확진자     32 non-null     int64
 2   격리해제    32 non-null     int64
 3   격리중     32 non-null     int64
 4   사망      32 non-null     int64
 5   검사중     32 non-null     int64
 6   결과 음성   32 non-null     int64
dtypes: int64(7)
memory usage: 2.0+ KB
```

Python

```python
df.isnull().sum()
```

Python

```
Out[-]
총계        0
확진자       0
격리해제      0
격리중       0
사망        0
검사중       0
결과 음성     0
dtype: int64
```

Python

```python
df.index
```

Python

```
Out[-]
Index(['2020-03-01', '2020-03-02', '2020-03-03', '2020-03-04', '2020-03-05',
       '2020-03-06', '2020-03-07', '2020-03-08', '2020-03-09', '2020-03-10',
       '2020-03-11', '2020-03-12', '2020-03-13', '2020-03-14', '2020-03-15',
       '2020-03-16', '2020-03-17', '2020-03-18', '2020-03-19', '2020-03-20',
       '2020-03-21', '2020-03-22', '2020-03-23', '2020-03-24', '2020-03-25',
       '2020-03-26', '2020-03-27', '2020-03-28', '2020-03-29', '2020-03-30',
       '2020-03-31', '2020-04-01'],
      dtype='object', name='date')
```

Python

```python
df.iloc[:,6]
```

Python

```
Out[-]
date
2020-03-01     61037
2020-03-02     71580
2020-03-03     85484
2020-03-04    102965
2020-03-05    118965
2020-03-06    136624
2020-03-07    151802
2020-03-08    162008
2020-03-09    171778
2020-03-10    184179
2020-03-11    196100
2020-03-12    209402
2020-03-13    222728
2020-03-14    235615
2020-03-15    243778
2020-03-16    251297
2020-03-17    261105
2020-03-18    270888
2020-03-19    282555
2020-03-20    292487
2020-03-21    303006
2020-03-22    308343
```

```
2020-03-23    315447
2020-03-24    324105
2020-03-25    334481
2020-03-26    341332
2020-03-27    352410
2020-03-28    361883
2020-03-29    369530
2020-03-30    372002
2020-03-31    383886
2020-04-01    395075
Name: 결과 음성, dtype: int64
```

Python

```
df.columns[6]
```

Python

```
Out[-]
'결과 음성'
```

Python

```
df.columns[1]
```

Python

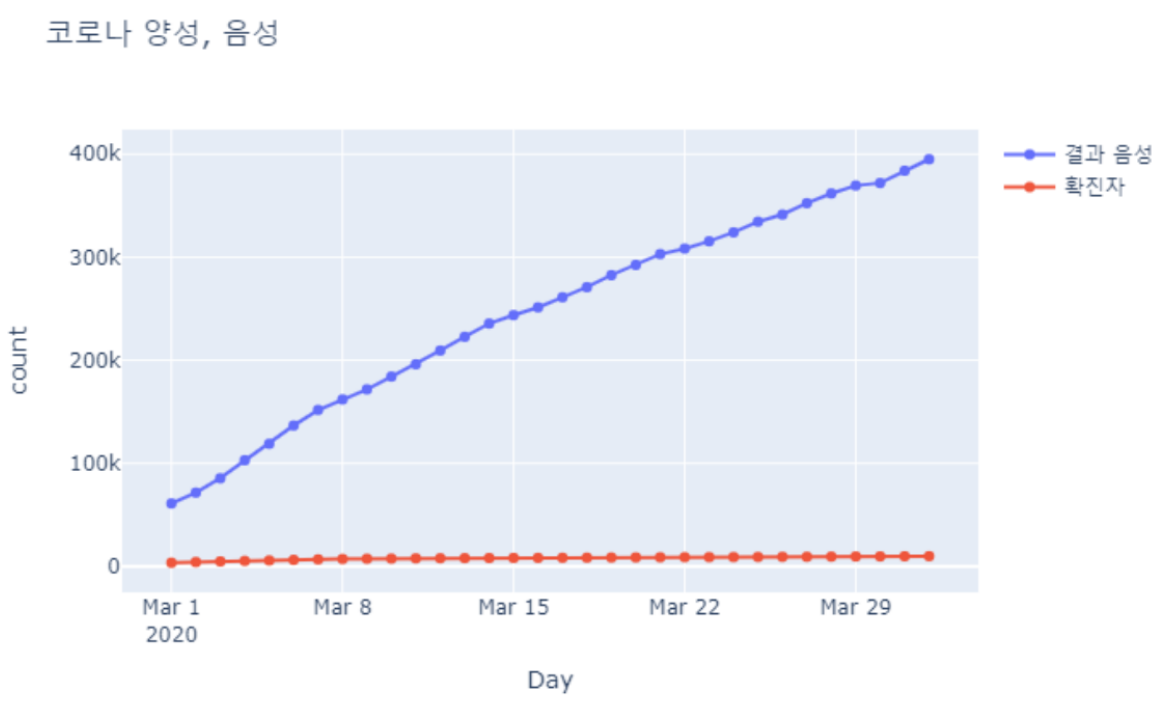
```
Out[-]
'확진자'
```

Python

```python
# 코로나 확진자-음성 그래프
import plotly.graph_objects as go
fig = go.Figure()

fig.add_trace(go.Scatter(x=df.index,
                         y=df.iloc[:,6],
                         mode = 'lines+markers',
                         name='결과 음성'))
fig.add_trace(go.Scatter(x=df.index,
                         y=df.
                         iloc[:,1],
                         mode = 'lines+markers',
                         name='확진자'))

fig.update_layout(title='코로나 양성, 음성', xaxis_title='Day', yaxis_title='count')
fig.show()
```

Python

Out[-]



```python
# 코로나 확진자-음성 그래프(semi-log)
import plotly.graph_objects as go
fig = go.Figure()

fig.add_trace(go.Scatter(x=df.index,
                         y=df.iloc[:,6],
                         mode = 'lines+markers',
                         name='결과 음성'))
fig.add_trace(go.Scatter(x=df.index,
                         y=df.
                         iloc[:,1],
                         mode = 'lines+markers',
                         name='확진자'))

fig.update_layout(title='코로나 양성, 음성',
                  xaxis_title='Day',
                  yaxis_title='count',
                  yaxis_type='log'
                 )
fig.show()
```

Python

Out[-]

```python
d = df.copy()
d.tail()
```

Python

```
Out[-]
            총계  확진자 격리해제 격리중 사망  검사중 결과  음성
      date
2020-03-28  387925  9478  4811  4523  144  16564  361883
2020-03-29  394141  9583  5033  4398  152  15028  369530
2020-03-30  395194  9661  5228  4275  158  13531  372002
2020-03-31  410564  9786  5408  4216  162  16892  383886
2020-04-01  421547  9887  5567  4155  165  16585  395075
```

Python

```python
# 치명률 = 사망자수/확진자수 *100
# 완치율 = 격리해제/확진자수 *100
d['치명률'] = round(d.iloc[:,4]/d.iloc[:,1]*100, 2)
d['완치율'] = round(d.iloc[:,2]/d.iloc[:,1]*100, 2)
d.tail()
```

Python

```
Out[-]
            총계  확진자 격리해제 격리중 사망  검사중 결과  음성  치명률  완치율
      date
2020-03-28  387925  9478  4811  4523  144  16564  361883  1.52  50.76
2020-03-29  394141  9583  5033  4398  152  15028  369530  1.59  52.52
2020-03-30  395194  9661  5228  4275  158  13531  372002  1.64  54.11
2020-03-31  410564  9786  5408  4216  162  16892  383886  1.66  55.26
2020-04-01  421547  9887  5567  4155  165  16585  395075  1.67  56.31
```

Python

```python
d.index
```

Python

```
Out[-]
Index(['2020-03-01', '2020-03-02', '2020-03-03', '2020-03-04', '2020-03-05',
       '2020-03-06', '2020-03-07', '2020-03-08', '2020-03-09', '2020-03-10',
       '2020-03-11', '2020-03-12', '2020-03-13', '2020-03-14', '2020-03-15',
       '2020-03-16', '2020-03-17', '2020-03-18', '2020-03-19', '2020-03-20',
       '2020-03-21', '2020-03-22', '2020-03-23', '2020-03-24', '2020-03-25',
       '2020-03-26', '2020-03-27', '2020-03-28', '2020-03-29', '2020-03-30',
       '2020-03-31', '2020-04-01'],
      dtype='object', name='date')
```

Python

```
d.iloc[:,7]
```

Python

```
Out[-]
date
2020-03-01    0.48
2020-03-02    0.52
2020-03-03    0.58
2020-03-04    0.60
2020-03-05    0.61
2020-03-06    0.67
2020-03-07    0.65
2020-03-08    0.70
2020-03-09    0.69
2020-03-10    0.72
2020-03-11    0.77
2020-03-12    0.84
2020-03-13    0.84
2020-03-14    0.89
2020-03-15    0.92
2020-03-16    0.91
2020-03-17    0.97
2020-03-18    1.00
2020-03-19    1.06
2020-03-20    1.09
2020-03-21    1.16
2020-03-22    1.17
2020-03-23    1.24
2020-03-24    1.33
2020-03-25    1.38
2020-03-26    1.42
2020-03-27    1.49
2020-03-28    1.52
2020-03-29    1.59
2020-03-30    1.64
2020-03-31    1.66
2020-04-01    1.67
Name: 치명률, dtype: float64
```

Python

```
d_g = go.Bar(x=d.index, y=d.iloc[:,7])
layout = go.Layout(title='치명율(사망/확진자수)')
fig = go.Figure(data=d_g, layout=layout)
fig.show()
```

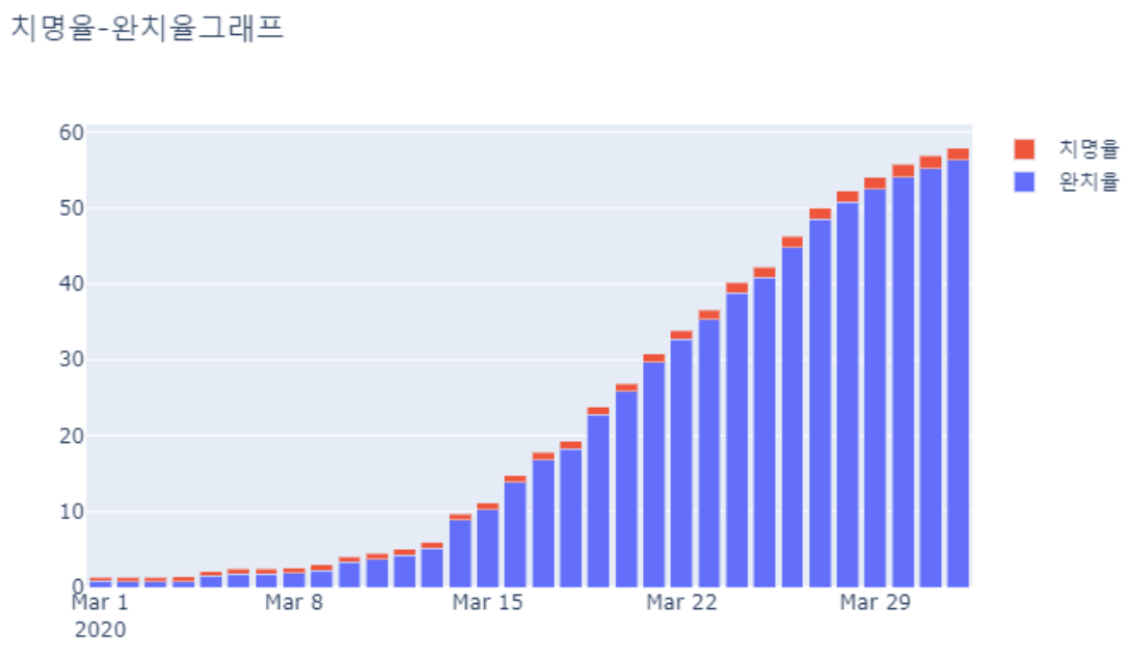
Python

Out[-]

```
death = go.Bar(x=d.index, y=d.iloc[:,7], name = "치명율")
cure = go.Bar(x=d.index, y=d.iloc[:,8], name = "완치율")
data = cure, death
layout = go.Layout(title="치명율-완치율그래프", barmode='stack')
fig = go.Figure(data=data, layout = layout)
fig.show()
```

Python

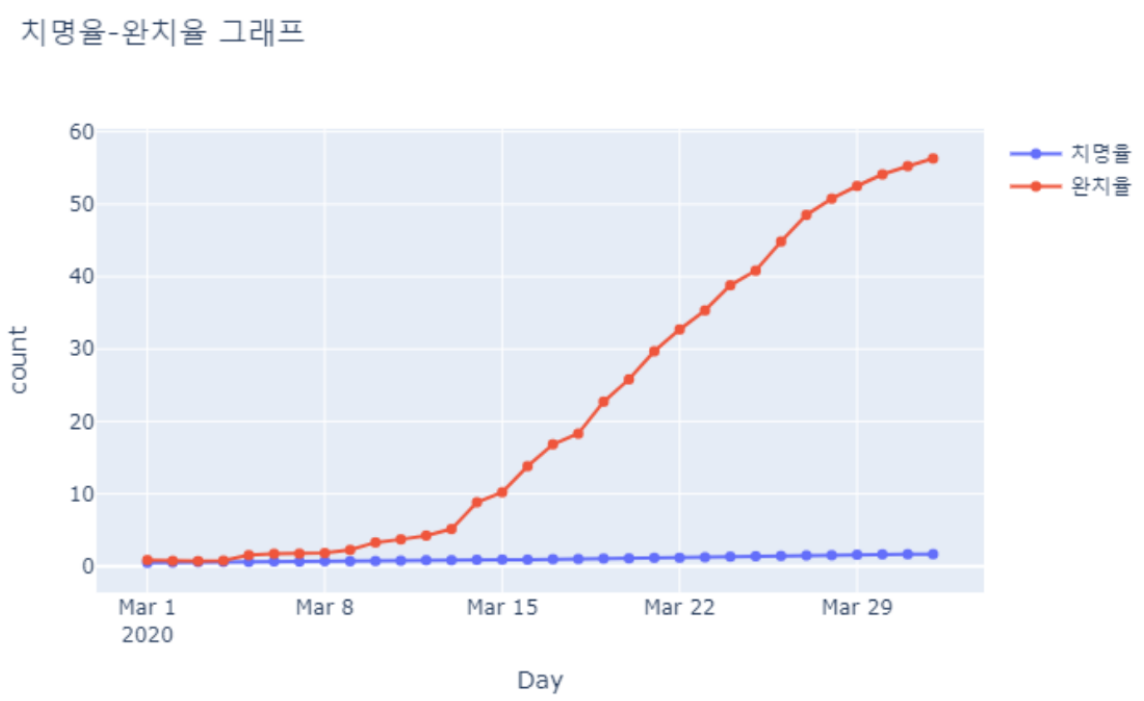
Out[-]



```
fig = go.Figure()

fig.add_trace(go.Scatter(x=d.index,
                         y=d.iloc[:,7],
                         mode = 'lines+markers',
                         name='치명율'))
fig.add_trace(go.Scatter(x=d.index,
                         y=d.iloc[:,8],
                         mode = 'lines+markers',
                         name='완치율'))

fig.update_layout(title='치명율-완치율 그래프',
                  xaxis_title='Day',
                  yaxis_title='count')
fig.show()
```

Python

Out[-]



```
today = df.copy()
today.tail()
```

Python

```
Out[-]
              총계   확진자 격리해제 격리중 사망   검사중 결과  음성
      date
2020-03-28  387925   9478     4811     4523    144  16564     361883
2020-03-29  394141   9583     5033     4398    152  15028     369530
2020-03-30  395194   9661     5228     4275    158  13531     372002
2020-03-31  410564   9786     5408     4216    162  16892     383886
2020-04-01  421547   9887     5567     4155    165  16585     395075
```

Python

```
print(today.columns[1])
print(today.columns[2])
print(today.columns[4])
```

Python

```
Out[-]
확진자
격리해제
사망
```

Python

```
일일확진자 = []
일일격리해제 = []
일일사망자 = []


일일확진자.append(595)
일일격리해제.append(3)
일일사망자.append(1)

for i in range(len(today)-1):
    일일확진자.append(today.iloc[:, 1][i+1]-today.iloc[:, 1][i])
    일일격리해제.append(today.iloc[:, 2][i+1]-today.iloc[:, 2][i])
    일일사망자.append(today.iloc[:, 4][i+1]-today.iloc[:, 4][i])

일일확진자 = pd.DataFrame(일일확진자, index=today.index)
일일격리해제 = pd.DataFrame(일일격리해제, index=today.index)
일일사망자 = pd.DataFrame(일일사망자, index=today.index)

일일확진자 = 일일확진자.rename(columns={0:'일일확진자'})
일일격리해제 = 일일격리해제.rename(columns={0:'일일격리해제'})
일일사망자 = 일일사망자.rename(columns={0:'일일사망자'})

today = pd.concat([today, 일일확진자, 일일격리해제, 일일사망자], axis=1)
today
```

Python

Out[-]

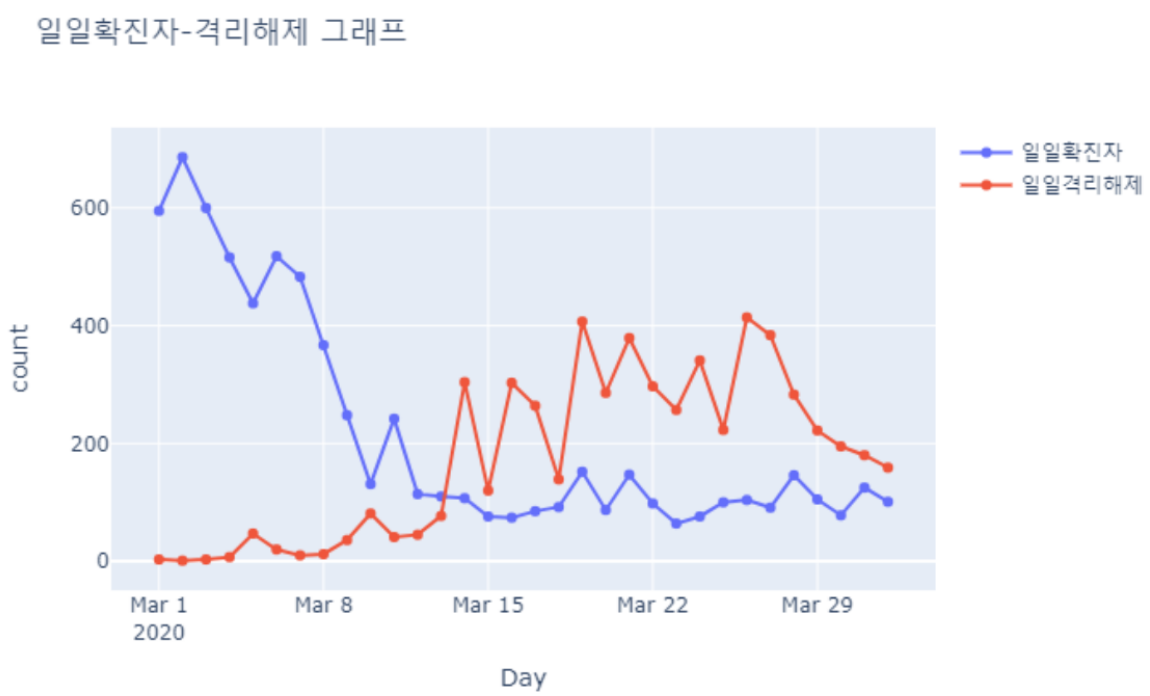
| | 총계 | 확진자 | 격리해제 | 격리중 | 사망 | 검사중 | 결과 음성 | 일일확진자 | 일일격리해제 | 일일사망자 |
|---|---|---|---|---|---|---|---|---|---|---|
| date | | | | | | | | | | |
| 2020-03-28 | 387925 | 9478 | 4811 | 4523 | 144 | 16564 | 361883 | 146 | 283 | 5 |
| 2020-03-29 | 394141 | 9583 | 5033 | 4398 | 152 | 15028 | 369530 | 105 | 222 | 8 |
| 2020-03-30 | 395194 | 9661 | 5228 | 4275 | 158 | 13531 | 372002 | 78 | 195 | 6 |
| 2020-03-31 | 410564 | 9786 | 5408 | 4216 | 162 | 16892 | 383886 | 125 | 180 | 4 |
| 2020-04-01 | 421547 | 9887 | 5567 | 4155 | 165 | 16585 | 395075 | 101 | 159 | 3 |

Python

```python
# 일일확진자-격리해제 그래프
fig=go.Figure()

fig.add_trace(go.Scatter(x=today.index,
                         y=today.iloc[:,7],
                         mode = 'lines+markers',
                         name='일일확진자'))

fig.add_trace(go.Scatter(x=today.index,
                         y=today.iloc[:,8],
                         mode = 'lines+markers',
                         name='일일격리해제'))

fig.update_layout(title='일일확진자-격리해제 그래프',
                  xaxis_title='Day',
                  yaxis_title='count')
fig.show()
```

Python

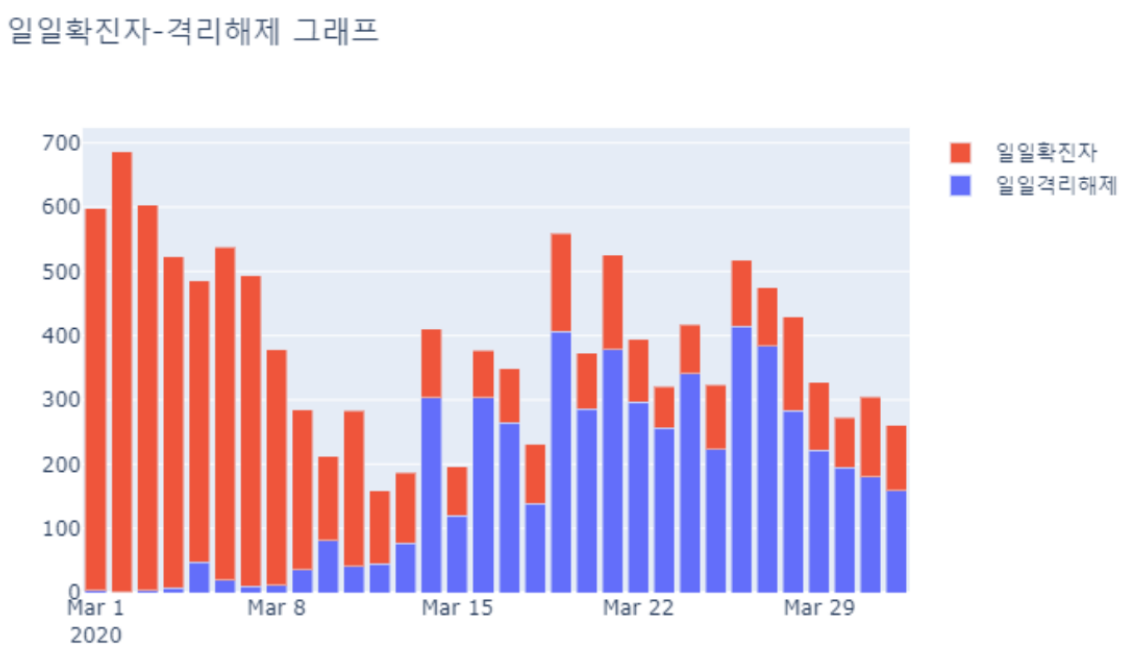
Out[-]



```
fig=go.Figure()

t1 = go.Bar(x=today.index,
            y=today.iloc[:,7],
            name='일일확진자')

t2 = go.Bar(x=today.index,
            y=today.iloc[:,8],
            name='일일격리해제')

data = t2, t1
layout = go.Layout(title='일일확진자-격리해제 그래프', barmode='stack')
fig = go.Figure(data=data, layout=layout)
fig.show()
```

Python

Out[-]



```python
일일사망자 = today.iloc[:,-1] # 일일사망자
치명률 = d.iloc[ : , -2] # 치명률
치명률
```

Python

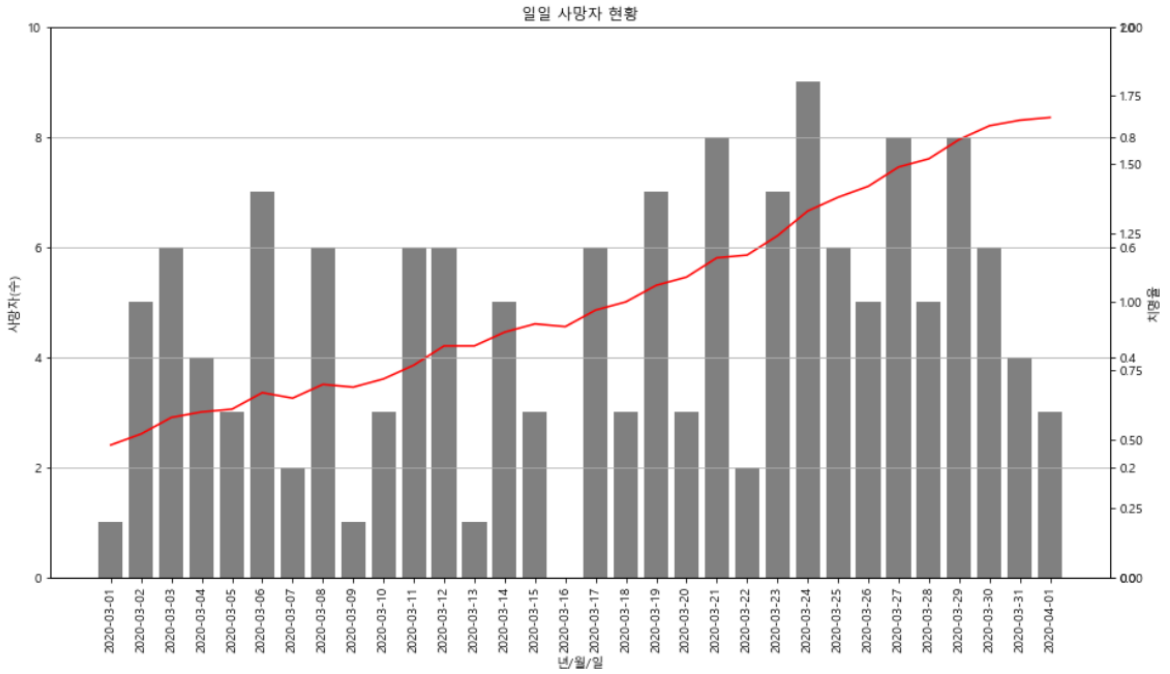
```
Out[-]
date
2020-03-01    0.48
2020-03-02    0.52
2020-03-03    0.58
2020-03-04    0.60
2020-03-05    0.61
2020-03-06    0.67
2020-03-07    0.65
2020-03-08    0.70
2020-03-09    0.69
2020-03-10    0.72
2020-03-11    0.77
2020-03-12    0.84
2020-03-13    0.84
2020-03-14    0.89
2020-03-15    0.92
2020-03-16    0.91
2020-03-17    0.97
2020-03-18    1.00
2020-03-19    1.06
2020-03-20    1.09
2020-03-21    1.16
2020-03-22    1.17
2020-03-23    1.24
2020-03-24    1.33
2020-03-25    1.38
2020-03-26    1.42
2020-03-27    1.49
2020-03-28    1.52
2020-03-29    1.59
2020-03-30    1.64
2020-03-31    1.66
2020-04-01    1.67
Name: 치명률, dtype: float64
```

Python

```python
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize']=15,8 # 그래프 크기
fig , ax = plt.subplots()
plt.xticks(rotation=90)

# ax 막대그래프
ax.bar(death_1.index ,death_1.values,
         color = 'gray')


ax.set_xlabel("년/월/일")
ax.set_ylabel('사망자(수)')

ax.set_ylim(0 ,10)
# ax1 선그래프
ax1 = ax.twinx()

ax1.plot(death ,
         color = 'r',
         label = '치명율')

ax1.set_ylim(0.0 , 2)
ax1.set_ylabel('치명율')

plt.title('일일 사망자 현황')

plt.grid(True)

plt.show()
```

Python

Out[-]


일일 사망자 현황
사망자(수)
치명율
년/월/일
2020-03-01
2020-03-02
2020-03-03
2020-03-04
2020-03-05
2020-03-06
2020-03-07
2020-03-08
2020-03-09
2020-03-10
2020-03-11
2020-03-12
2020-03-13
2020-03-14
2020-03-15
2020-03-16
2020-03-17
2020-03-18
2020-03-19
2020-03-20
2020-03-21
2020-03-22
2020-03-23
2020-03-24
2020-03-25
2020-03-26
2020-03-27
2020-03-28
2020-03-29
2020-03-30
2020-03-31
2020-04-01

# 4. 세계 코로나 현황

- 자료 출처 : kaggle - https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset#time_series_covid_19_confirmed.csv
- 참고자료 : https://github.com/CSSEGISandData/COVID-19

```python
import pandas as pd

confirmed = pd.read_csv('./data/time_series_covid_19_confirmed.csv')
deaths = pd.read_csv('./data/time_series_covid_19_deaths.csv')
recovered = pd.read_csv('./data/time_series_covid_19_recovered.csv')
```

Python

```python
confirmed.info()
```

Python

```
Out[-]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 258 entries, 0 to 257
Data columns (total 76 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Province/State  80 non-null     object
 1   Country/Region  258 non-null    object
 2   Lat             258 non-null    float64
 3   Long            258 non-null    float64
 4   1/22/20         258 non-null    int64
 5   1/23/20         258 non-null    int64
 6   1/24/20         258 non-null    int64
 7   1/25/20         258 non-null    int64
 8   1/26/20         258 non-null    int64
 9   1/27/20         258 non-null    int64
 10  1/28/20         258 non-null    int64
 11  1/29/20         258 non-null    int64
 12  1/30/20         258 non-null    int64
 13  1/31/20         258 non-null    int64
 14  2/1/20          258 non-null    int64
 15  2/2/20          258 non-null    int64
 16  2/3/20          258 non-null    int64
 17  2/4/20          258 non-null    int64
 18  2/5/20          258 non-null    int64
```

```
 19  2/6/20          258 non-null    int64
 20  2/7/20          258 non-null    int64
 21  2/8/20          258 non-null    int64
 22  2/9/20          258 non-null    int64
 23  2/10/20         258 non-null    int64
 24  2/11/20         258 non-null    int64
 25  2/12/20         258 non-null    int64
 26  2/13/20         258 non-null    int64
 27  2/14/20         258 non-null    int64
 28  2/15/20         258 non-null    int64
 29  2/16/20         258 non-null    int64
 30  2/17/20         258 non-null    int64
 31  2/18/20         258 non-null    int64
 32  2/19/20         258 non-null    int64
 33  2/20/20         258 non-null    int64
 34  2/21/20         258 non-null    int64
 35  2/22/20         258 non-null    int64
 36  2/23/20         258 non-null    int64
 37  2/24/20         258 non-null    int64
 38  2/25/20         258 non-null    int64
 39  2/26/20         258 non-null    int64
 40  2/27/20         258 non-null    int64
 41  2/28/20         258 non-null    int64
 42  2/29/20         258 non-null    int64
 43  3/1/20          258 non-null    int64
 44  3/2/20          258 non-null    int64
 45  3/3/20          258 non-null    int64
 46  3/4/20          258 non-null    int64
 47  3/5/20          258 non-null    int64
 48  3/6/20          258 non-null    int64
 49  3/7/20          258 non-null    int64
 50  3/8/20          258 non-null    int64
 51  3/9/20          258 non-null    int64
 52  3/10/20         258 non-null    int64
 53  3/11/20         258 non-null    int64
 54  3/12/20         258 non-null    int64
 55  3/13/20         258 non-null    int64
 56  3/14/20         258 non-null    int64
 57  3/15/20         258 non-null    int64
 58  3/16/20         258 non-null    int64
 59  3/17/20         258 non-null    int64
 60  3/18/20         258 non-null    int64
 61  3/19/20         258 non-null    int64
 62  3/20/20         258 non-null    int64
 63  3/21/20         258 non-null    int64
```

```
 64  3/22/20         258 non-null    int64
 65  3/23/20         258 non-null    int64
 66  3/24/20         258 non-null    int64
 67  3/25/20         258 non-null    int64
 68  3/26/20         258 non-null    int64
 69  3/27/20         258 non-null    int64
 70  3/28/20         258 non-null    int64
 71  3/29/20         258 non-null    int64
 72  3/30/20         258 non-null    int64
 73  3/31/20         258 non-null    int64
 74  4/1/20          258 non-null    int64
 75  4/2/20          258 non-null    int64
dtypes: float64(2), int64(72), object(2)
memory usage: 153.3+ KB
```

Python

```
confirmed.head()
```

Python

```
Out[-]
  Province/State  Country/Region      Lat      Long 1/22/20 1/23/20 1/24/20 1/25/20 1/26/
20  1/27/20 ... 3/28/20 3/29/20 3/30/20 3/31/20 4/1/20  4/2/20
0           NaN      Afghanistan   33.0000  65.0000       0       0       0       0
0       0 ...     110     120     170     174     237     273
1           NaN          Albania   41.1533  20.1683       0       0       0       0
0       0 ...     197     212     223     243     259     277
2           NaN          Algeria   28.0339   1.6596       0       0       0       0
0       0 ...     454     511     584     716     847     986
3           NaN          Andorra   42.5063   1.5218       0       0       0       0
0       0 ...     308     334     370     376     390     428
4           NaN           Angola  -11.2027  17.8739       0       0       0       0
0       0 ...       5       7       7       7       8       8

5 rows × 76 columns
```

Python

```
confirmed.tail()
```

Python

```
Out[-]
    Province/State  Country/Region         Lat        Long  1/22/20 1/23/20 1/24/20 1/25/
20  1/26/20 1/27/20 ... 3/28/20 3/29/20 3/30/20 3/31/20 4/1/20  4/2/20
253            NaN        Botswana  -22.328500   24.684900        0       0       0
0       0       0 ...        0       0       3       4      4       4
254            NaN         Burundi   -3.373100   29.918900        0       0       0
0       0       0 ...        0       0       0       2      2       3
255            NaN    Sierra Leone    8.460555  -11.779889        0       0       0
0       0       0 ...        0       0       0       1      2       2
     Bonaire, Sint
256  Eustatius and     Netherlands   12.178400  -68.238500        0       0       0
0       0       0 ...        0       0       0       0      0       2
              Saba
257
               NaN          Malawi  -13.254308   34.301525        0       0       0
0       0       0 ...        0       0       0       0      0       3
5 rows × 76 columns
```

Python

## 데이터 전처리

```
# 데이터 전처리 (Province/State) 제거
confirmed_1 = confirmed.drop(['Province/State'], axis = 1)
deaths_1 = deaths.drop(['Province/State'], axis = 1)
recovered_1 = recovered.drop(['Province/State'], axis = 1)
```

Python

```
confirmed_1.head()
```

Python

```
Out[-]
  Country/Region  1/22/20 1/23/20 1/24/20 1/25/20 1/26/20 1/27/20 ... 3/28/20 3/29/20 3/3
0/20  3/31/20 4/1/20  4/2/20
0    Afghanistan        0       0       0       0       0       0 ...     110     120
170     174    237     273
1        Albania        0       0       0       0       0       0 ...     197     212
223     243    259     277
2        Algeria        0       0       0       0       0       0 ...     454     511
584     716    847     986
3        Andorra        0       0       0       0       0       0 ...     308     334
370     376    390     428
4         Angola        0       0       0       0       0       0 ...       5       7
7       7      8       8

5 rows × 75 columns
```

Python

## 중복된 Country를 찾고 그룹화 하기

```python
# 중복된 나라 찾기
country = confirmed_2.iloc[:,0]
df = country.value_counts()
df.index[1]
df[0]
# 중복된 나라 1개 초과일 경우
for i in range(len(df)):
    temp = df[i]
    if temp > 1:
        print(df.index[i])
```

Python

```
Out[-]
China
Canada
France
United Kingdom
Australia
Netherlands
Denmark
```

Python

```python
# 중복된 나라 출력하기
confirmed_2.iloc[:,0]
confirmed_2.iloc[:,0].value_counts()
```

Python

```
Out[-]
China                  33
Canada                 15
France                 10
United Kingdom         10
Australia               8
                       ..
Congo (Brazzaville)     1
Holy See                1
Armenia                 1
Sweden                  1
Malawi                  1
Name: Country/Region, Length: 181, dtype: int64
```

Python

- 중복된 나라 : China, Canada, United Kingdom, France, Australia, Netherlands, Denmark

```python
# 중복된 나라 그룹화 해서 값을 합함
confirmed_3 = confirmed_2.groupby('Country/Region').sum()
deaths_3= deaths_2.groupby('Country/Region').sum()
recovered_3= recovered_2.groupby('Country/Region').sum()
```

Python

```
confirmed_3.loc[['China']].T
confirmed_3.loc[['United Kingdom']].T
```

Python

```
Out[-]
Country/Region  United Kingdom
       1/22/20               0
       1/23/20               0
       1/24/20               0
       1/25/20               0
       1/26/20               0
           ...             ...
       3/28/20           17312
       3/29/20           19780
       3/30/20           22453
       3/31/20           25481
        4/1/20           29865
        4/2/20           34173
72 rows × 1 columns
```

Python

## 시각화

```
confirmed_3.iloc[:, -1]
confirmed_3.iloc[:, -1]['China']
```

Python

```
Out[-]
82432
```

Python

```
# 4월 2일 기준으로 누적 확진자 데이터
confirmed_4 = confirmed_3.iloc[:, -1 ]
deaths_4 = deaths_3.iloc[:, -1 ]
recovered_4 = recovered_3.iloc[:, -1 ]
```

Python

```
df = confirmed_4.sort_values(ascending=False)
df
```

Python

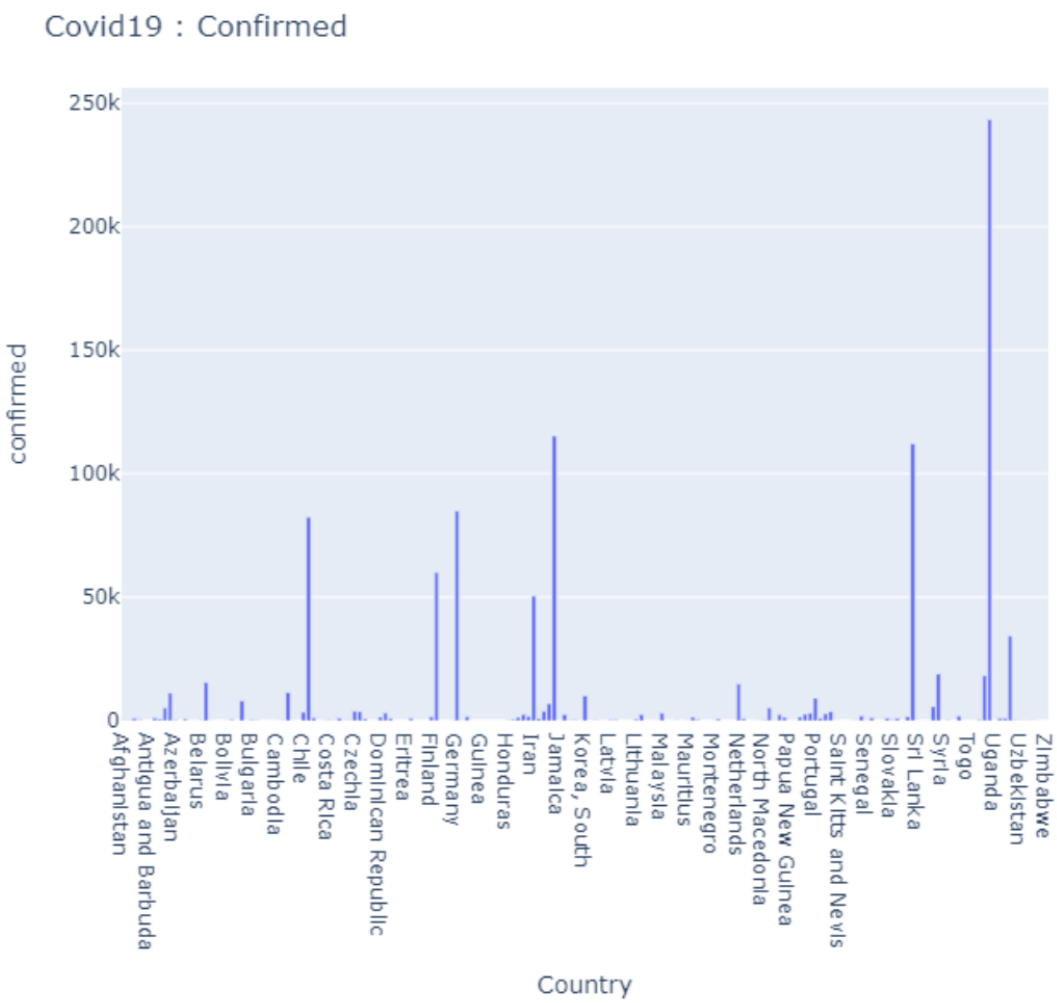
```
Out[-]
Country/Region
US                                  243453
Italy                               115242
Spain                               112065
Germany                              84794
China                                82432
                                     ...
Burundi                                  3
Sierra Leone                             2
Saint Vincent and the Grenadines         2
Papua New Guinea                         1
Timor-Leste                              1
Name: 4/2/20, Length: 181, dtype: int64
```

Python

```python
import plotly.graph_objects as go
import plotly.express as px

fig = px.bar(confirmed_4,
             x = confirmed_4.index,
             y = confirmed_4.values,
             height = 600)

fig.update_layout(title_text='Covid19 : Confirmed',
                  xaxis_title='Country',
                  yaxis_title='confirmed')

fig.show()
```

Python

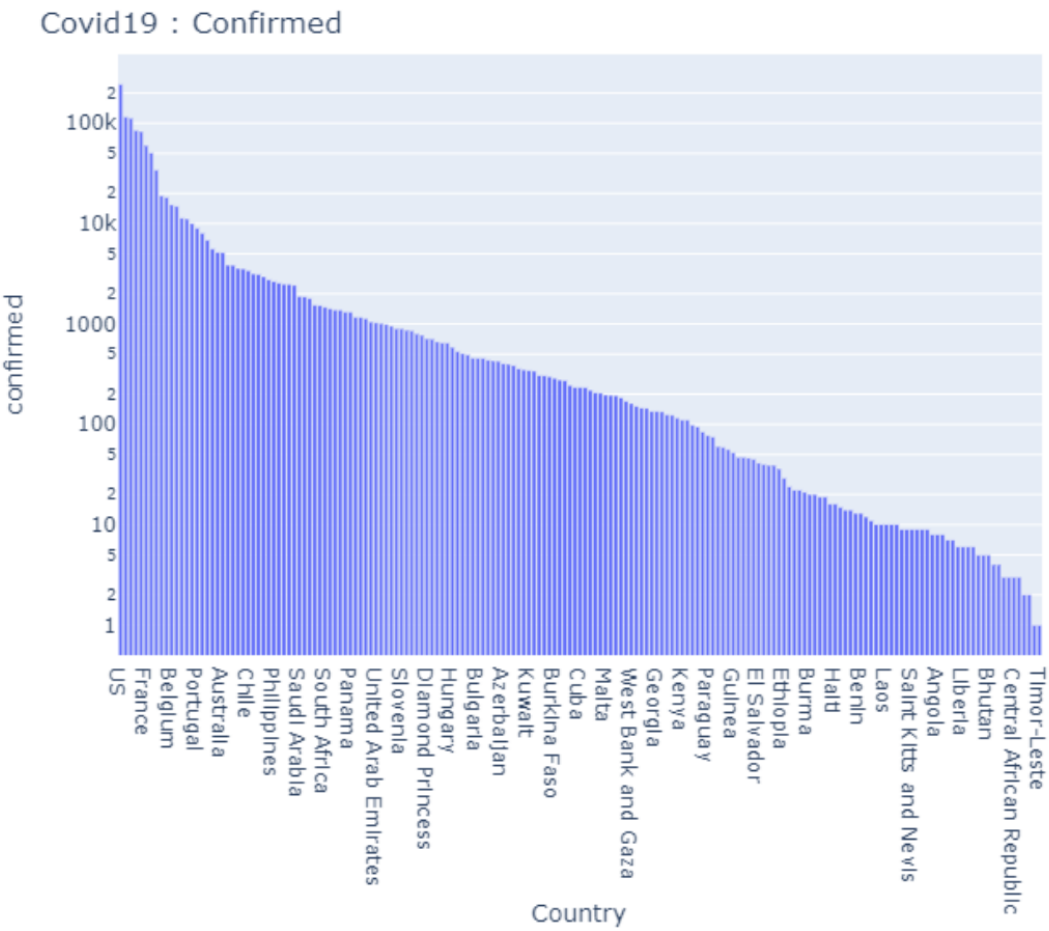
Out[-]



```python
import plotly.graph_objects as go
import plotly.express as px

fig = px.bar(df,
             x = df.index,
             y = df.values,
             height = 600)

fig.update_layout(title_text='Covid19 : Confirmed',
                  xaxis_title='Country',
                  yaxis_title='confirmed',
                  yaxis_type='log',
                 )

fig.show()
```

Python

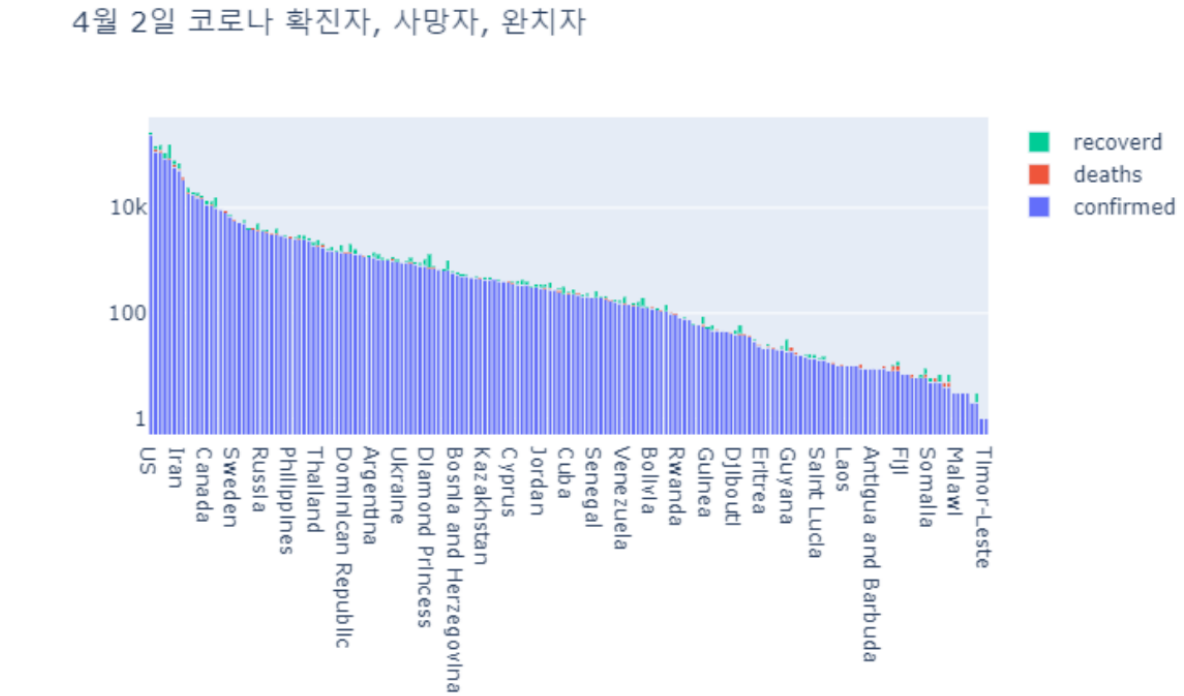
Out[-]



```
import plotly.graph_objects as go

df1 = confirmed_4.sort_values(ascending=False)
df2 = deaths_4.sort_values(ascending=False)
df3 = recovered_4.sort_values(ascending=False)

fig = go.Figure(
    data = [go.Bar(name = 'confirmed',
                   x = df1.index,
                   y = df1.values),
            go.Bar(name = 'deaths',
                   x = df2.index,
                   y = df2.values ),
            go.Bar(name = 'recoverd',
                   x = df3.index,
                   y = df3.values)])

fig.update_layout(barmode = 'stack',
                  title_text = '4월 2일 코로나 확진자, 사망자, 완치자',
                  yaxis_type='log',
                 )
fig.show()
```

Python

Out[-]



## Top 15

```python
# 누적 확진자 기준으로 나라 top15를 구한다
# 사망자 데이터, 완치자 데이터 에서 위에 나라의 변수 값을 추출한다.
confirmed_5 = confirmed_4.sort_values(ascending = False).head(15)
# confirmed_5
confirmed_list15 = []
for i in confirmed_5.index:
    confirmed_list15.append(i)
confirmed_list15
```

Python

```
Out[-]
['US',
 'Italy',
 'Spain',
 'Germany',
 'China',
 'France',
 'Iran',
 'United Kingdom',
 'Switzerland',
 'Turkey',
 'Belgium',
 'Netherlands',
 'Canada',
 'Austria',
 'Korea, South']
```

Python

```
confirmed_5
```

Python

```
Out[-]
Country/Region
US                243453
Italy             115242
Spain             112065
Germany            84794
China              82432
France             59929
Iran               50468
United Kingdom     34173
Switzerland        18827
Turkey             18135
Belgium            15348
Netherlands        14788
Canada             11284
Austria            11129
Korea, South        9976
Name: 4/2/20, dtype: int64
```

Python

```
confirmed_5 = pd.DataFrame(confirmed_5)
confirmed_5
```

Python

```
Out[-]
                4/2/20
Country/Region
            US  243453
         Italy  115242
         Spain  112065
       Germany   84794
         China   82432
        France   59929
          Iran   50468
United Kingdom   34173
   Switzerland   18827
        Turkey   18135
       Belgium   15348
   Netherlands   14788
        Canada   11284
       Austria   11129
  Korea, South    9976
```

```python
confirmed_list15

# top 15 누적사망자
deaths_5 = []
for i in confirmed_list15:
    for index, values in enumerate(deaths_4.index):
        if i == values:
            deaths_5.append(deaths_4[index])

deaths_5 = pd.DataFrame(deaths_5,
                        index=confirmed_5.index,
                        columns=confirmed_5.columns)
deaths_5
```

Python

```
Out[-]
                4/2/20
Country/Region
            US    7087
         Italy   14681
         Spain   11198
       Germany    1275
         China    3326
        France    6520
          Iran    3294
United Kingdom    3611
   Switzerland     591
        Turkey     425
       Belgium    1143
   Netherlands    1490
        Canada     179
       Austria     168
  Korea, South     174
```

Python

```python
deaths_4
# deaths_4.index
deaths_4[2]
```

Python

```
Out[-]
105
```

Python

```python
for index, values in enumerate(confirmed_list15):
    print(index, values)
```

Python

```
Out[-]
0 US
1 Italy
2 Spain
3 Germany
4 China
5 France
6 Iran
7 United Kingdom
8 Switzerland
9 Turkey
10 Belgium
11 Netherlands
12 Canada
13 Austria
14 Korea, South
```

Python

```python
# top 15 누적완치자
recovered_5  = []
for i in confirmed_list15:
    for index, values in enumerate(recovered_4.index):
        if i == values :
            recovered_5.append(recovered_4[index])

recovered_5 = pd.DataFrame(recovered_5,
                           index = confirmed_5.index,
                           columns = confirmed_5.columns)
recovered_5
```

Python

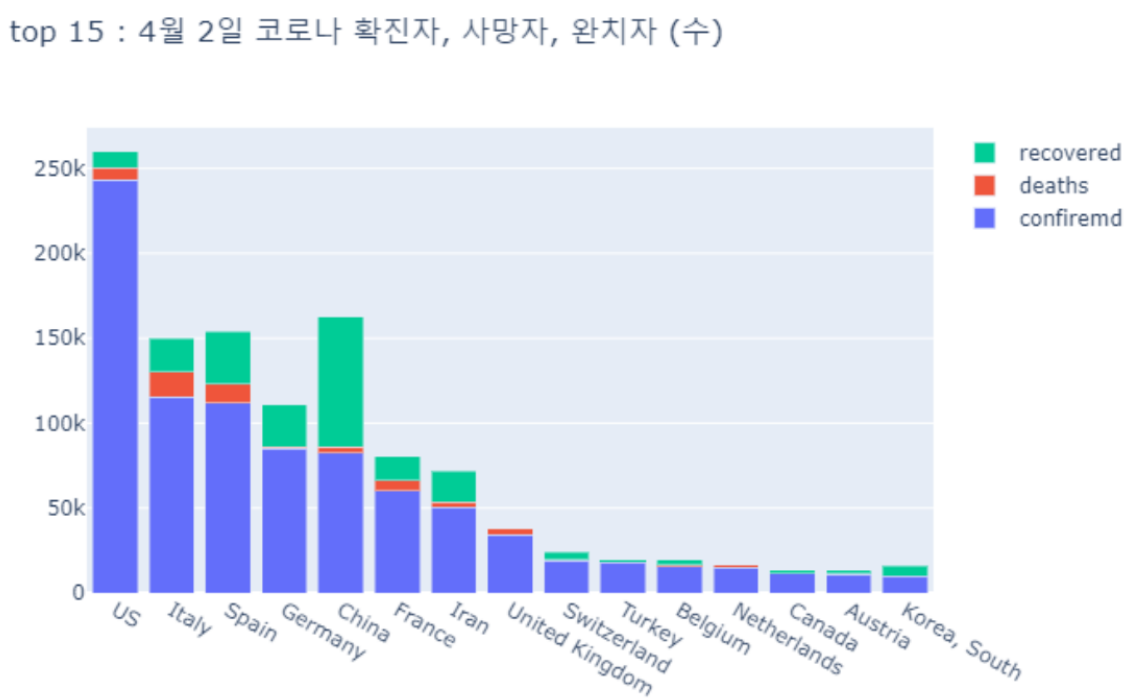
```
Out[-]
                4/2/20
Country/Region
            US   9707
         Italy  19758
         Spain  30513
       Germany  24575
         China  76760
        France  14135
          Iran  17935
United Kingdom    208
   Switzerland   4846
        Turkey    484
       Belgium   2872
   Netherlands    260
        Canada   2175
       Austria   2022
  Korea, South   6021
```

Python

```
import plotly.graph_objects as go

fig = go.Figure(

data = [go.Bar(name = 'confiremd',
               x = confirmed_5.index,
               y = confirmed_5['4/2/20']),

       go.Bar(name = 'deaths',
              x = deaths_5.index,
              y = deaths_5['4/2/20'] ),

       go.Bar(name = 'recovered',
              x = recovered_5.index,
              y = recovered_5['4/2/20'])])

fig.update_layout(barmode = 'stack',
                  title_text = ' top 15 : 4월 2일 코로나 확진자, 사망자, 완치자 (수)')
fig.show()
```

Python

Out[-]



## US covid - 19

```
confirmed_3.loc["US", ]
```

Python

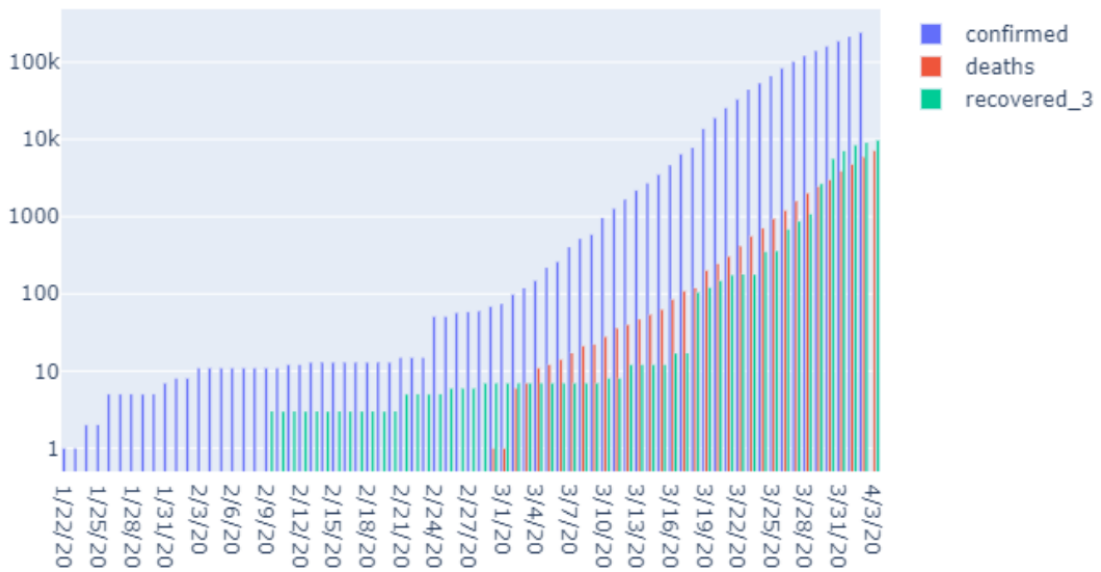
```
Out[-]
1/22/20         1
1/23/20         1
1/24/20         2
1/25/20         2
1/26/20         5
              ...
3/29/20    140886
3/30/20    161807
3/31/20    188172
4/1/20     213372
4/2/20     243453
Name: US, Length: 72, dtype: int64
```

Python

```python
# 확진자 , 사망자 , 완치자 시계열 막대그래프
fig = go.Figure()

fig.add_trace(go.Bar(x = confirmed_3.loc["US",].index,
                     y = confirmed_3.loc["US",],
                     name = 'confirmed'))

fig.add_trace(go.Bar(x = deaths_3.loc["US",].index,
                     y = deaths_3.loc["US",],
                     name = 'deaths'))

fig.add_trace(go.Bar(x = recovered_3.loc["US",].index,
                     y = recovered_3.loc["US",],
                     name = 'recovered_3'))

fig.update_layout(yaxis_type='log')

fig.show()
```

Python

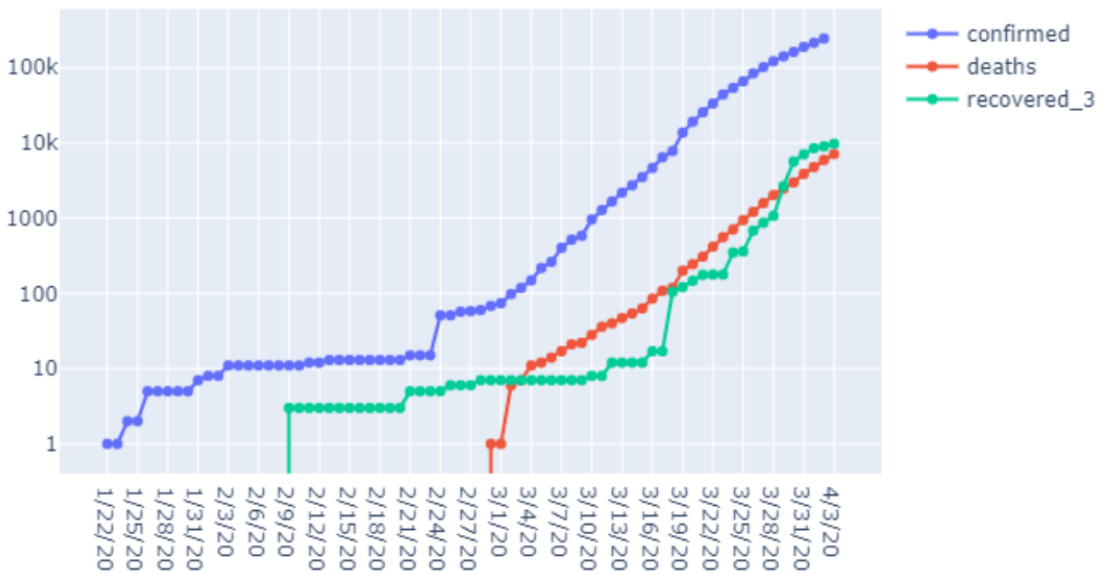
Out[-]



```python
# 확진자 , 사망자 , 완치자 시계열 선그래프
fig = go.Figure()

fig.add_trace(go.Scatter(x = confirmed_3.loc["US",].index,
                         y = confirmed_3.loc["US",],
                         mode = 'lines + markers',
                         name = 'confirmed'))

fig.add_trace(go.Scatter(x = deaths_3.loc["US",].index,
                         y = deaths_3.loc["US",],
                         mode = 'lines + markers',
                         name = 'deaths'))

fig.add_trace(go.Scatter(x = recovered_3.loc["US",].index,
                         y = recovered_3.loc["US",],
                         mode = 'lines + markers',
                         name = 'recovered_3'))

fig.update_layout(yaxis_type='log')

fig.show()
```

Python

Out[-]



# 7일차 데이터 분석을 위한 수학적 이론

이 장에서 다루는 내용

통계 이론
확률 이론
미분
선형 회귀
행렬

## 통계이론

- 모수(parameter) : 모집단의 특성을 수치로 표현한 수
- 통계량/추정량(statistic) : 표본을 통해서 계산되어진 양
  - 표본평균 : 표본의 평균

$$\overline{X} = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{k=1}^{n} X_k$$

  - 표본 분산 : 표분의 분산

$$s^2 = \frac{1}{n-1}\left[\left(X_1 - \overline{X}\right)^2 + \left(X_2 - \overline{X}\right)^2 + \cdots + \left(X_n - \overline{X}\right)^2\right]$$

$$= \frac{1}{n-1}\sum_{k=1}^{n}\left(X_k - \overline{X}\right)^2$$

## 평균

```python
반_1 = [100, 0]
반_2 = [75, 25]
반_3 = [50, 50]
```

Python

```python
반_1_평균 = sum(반_1)/len(반_1)
반_1_평균
```

Python

```
Out[-]
50.0
```

Python

```python
반_2_평균 = sum(반_2)/len(반_2)
반_2_평균
```

Python

```
Out[-]
50.0
```

Python

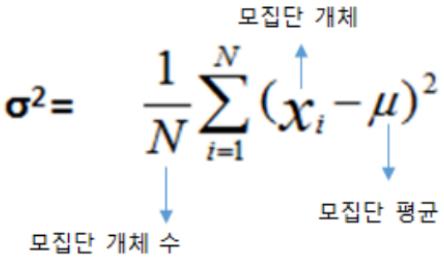
```python
반_3_평균 = sum(반_3)/len(반_3)
반_3_평균
```

Python

```
Out[-]
50.0
```

Python

## 분산




- 평균으로부터 각 숫자들의 거리 편차를 제곱한 값의 평균 ⇒ 편차의 제곱의 평균

```python
# 반_1 = [50, -50]
# 반_2 = [25, -25]
# 반_3 = [0, 0]
반_1_분산 = [50**2, (-50)**2]
반_2_분산 = [25**2, (-25)**2]
반_3_분산 = [0, 0]
```

Python

```python
반_1_분산 = sum(반_1_분산)/len(반_1_분산)
반_1_분산
```

Python

```
Out[-]
2500.0
```

Python

```python
반_2_분산 = sum(반_2_분산)/len(반_2_분산)
반_2_분산
```

Python

```
Out[-]
625.0
```

Python

```python
반_3_분산 = sum(반_3_분산)/len(반_3_분산)
반_3_분산
```

Python

```
Out[-]
0.0
```

Python

```python
반_1_분산 ** 0.5
```

Python

```
Out[-]
50.0
```

Python

## 표준편차

$$\sigma = \sqrt{\sigma^2}$$

- 데이터의 산포도(퍼진 정도)를 나타내는 값
- 데이터가 밀집되지 않고 넓게 분포되어지면 표준편차의 값은 커짐

```python
import math

반_1_표준편차 = math.sqrt(반_1_분산)
반_2_표준편차 = math.sqrt(반_2_분산)
반_3_표준편차 = math.sqrt(반_3_분산)
```

Python

```python
print(반_1_표준편차, 반_2_표준편차, 반_3_표준편차)
```

Python

```
Out[-]
50.0 25.0 0.0
```

Python

# 확률 이론

- 확률 실험/확률 시행(Random Experiment) : 이론적으로 동일한 조건에서 여러 번 반복할 수 있고 그 결과는 우연에 의해서 결정되는 실험
- 표본 공간(Sample space) : 나올 수 있는 모든 경우의 결과들의 모임, 시행의 결과들의 집합
- 근원 사건(Sample outcome) : 표본 공간의 원소
- 사건(Event) : 표본 공간의 부분집합이자 근원 사건의 집합
- 곱 사건 : 두 사건의 교집합으로 표현
- 배반 사건 : 두 집합의 교집합이 공집합인 사건
- 확률 변수 : 사건의 확률을 수치적 변수로 표현

## 동전 2개를 던졌을 때 앞면의 개수

```
x       0     1     2
P(x)   1/4   1/2   1/4
```

Plain Text

```
import numpy as np

평균 = 1
확률 = [1/4, 1/2, 1/4]
편차 = np.array([-1, 0, 1])
편차의제곱 = 편차**2
편차의제곱
```

Python

```
Out[-]
array([1, 0, 1], dtype=int32)
```

Python

```
sum(편차의제곱*확률)
```

Python

```
Out[-]
0.5
```

## 공분산

$$Cov(X,Y) = E(X-\mu_X)(Y-\mu_Y) = \sum_{i=1}^{n}(x_i-\mu_X)\,(y_i-\mu_Y)f(x_i,y_i)$$

- 두 변수(x, y)의 어떤 관계를 가지고 변화하는 측도를 나타냄
- 공분산이 양수이면 두 변수가 같은 방향이고, 음수일 경우 서로 다른 방향
- 공분산이 0이면 서로가 독립을 의미

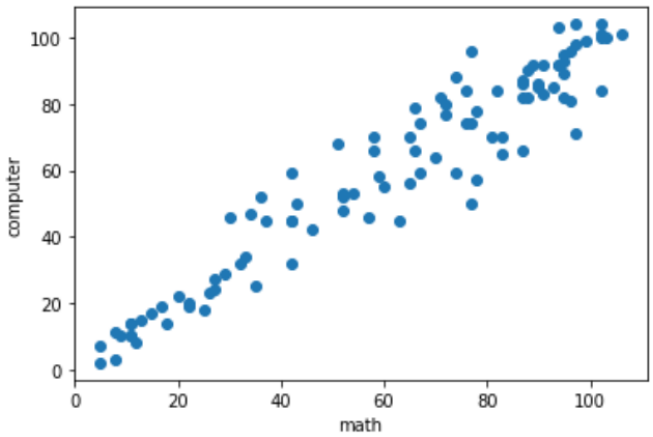
```python
수학점수 = []
컴퓨터점수 = []
for i in range(100):
    if i > 20 and i < 80:
        수학점수.append(i + np.random.randint(1, 30))
        컴퓨터점수.append(i + np.random.randint(1, 30))
    else:
        수학점수.append(i + np.random.randint(1, 10))
        컴퓨터점수.append(i + np.random.randint(1, 10))
```

Python

```python
import matplotlib.pyplot as plt
%matplotlib inline

plt.scatter(수학점수, 컴퓨터점수)
plt.xlabel('math')
plt.ylabel('computer')
plt.show()
```

Python

Out[-]




```
# 평균(mean): np.mean(x)
# 표준편차(standard deviation) : np.std(x)
# 분산(variance) : np.var(x)

# 1사분면 = (x - np.mean(x)(y - np.mean(y))
#           (90 - 50)(90 - 50) = 1600
# 2사분면 = (x - np.mean(x)(y - np.mean(y))
#           (40 - 50)(60 - 50) = -100
# 3사분면 = (x - np.mean(x)(y - np.mean(y))
#           (20 - 50)(20 - 50) = 900
# 4사분면 = (x - np.mean(x)(y - np.mean(y))
#           (70 - 50)(40 - 50) = -200
```

Python

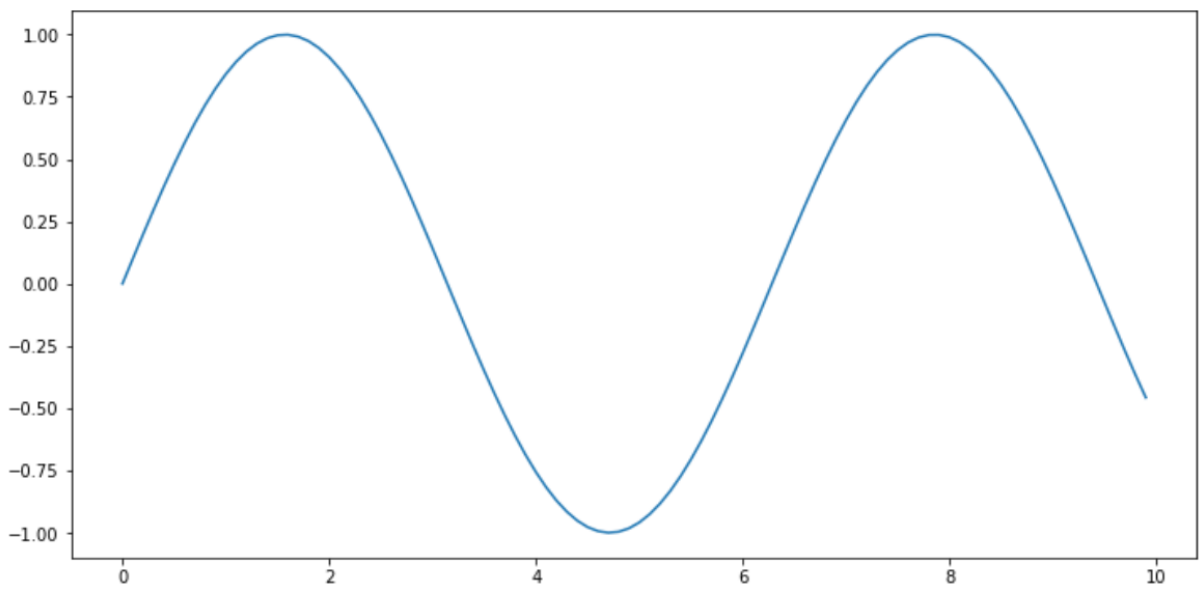
# 미분

- 함수가 주어졌을 때 도함수를 구하는 과정
  - 도함수 : x가 바뀔 때마다 생기는 순간 변화율을 구하는 함수
- 함수의 극대값, 극소값을 구할 수 있음
- 평균 변화율 : x가 변할 때 y의 변화량을 말함
- 순간 변화율 : 평균 변화율의 극한값으로 접선의 기울기

```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.arange(0, 10, 0.1)
y = np.sin(x)

plt.figure(figsize=(12,6))
plt.plot(x, y);
```

Python

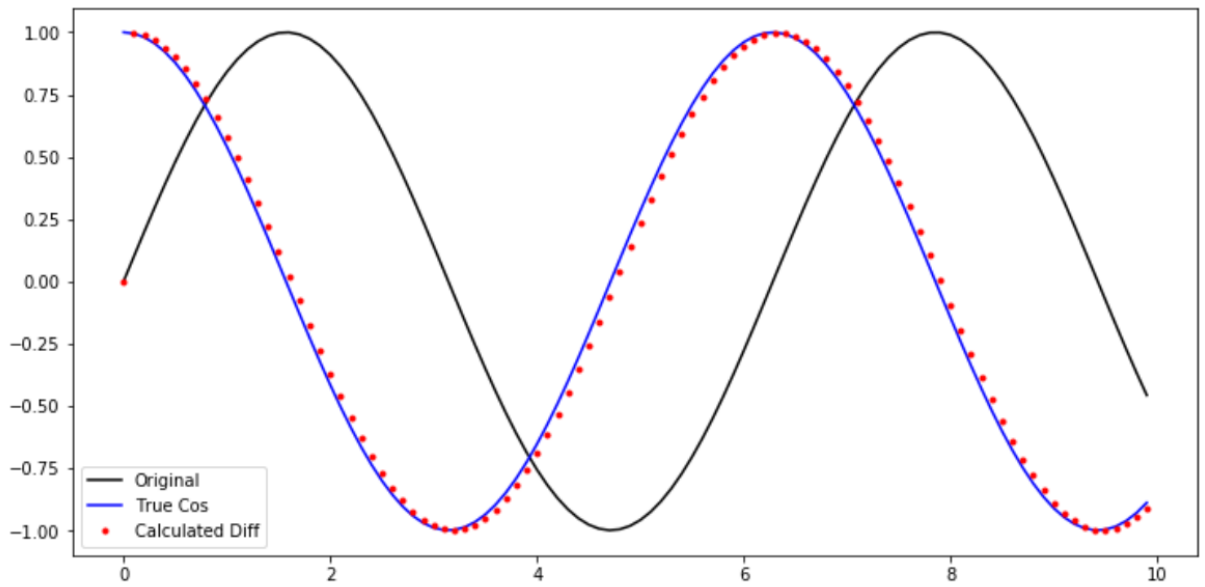
Out[-]



```python
plt.figure(figsize=(12,6))

plt.plot(x, y, 'k', label='Original')
plt.plot(x, np.cos(x), 'b', label='True Cos')
plt.plot(x, np.r_[0, np.diff(y)]/0.1, 'r.', label='Calculated Diff')

plt.legend(loc='best')
plt.show()
```

Python

Out[-]



# 선형 회귀

- Modeling : 모델을 만들어 내는 과정
- 지도 학습 : 정답을 제공하고 학습을 시키는 것
  - 회귀 분석 : 입력 변수 x에 대해서 연속형 출력 변수 Y를 예측하는 분석
- 기본 가정
  1. 선형성 : 독립 변수 x와 종속변수 y가 선형적이어야 함
  2. 독립성 : 종속변수는 다른 종속변수의 값에 영향 받지 않음
  3. 정규성 : 잔차는 기대값이 0인 정규분포를 따라야 함
  4. 등분산성 : 잔차의 분산은 일정해야 함

## 정규분포

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

```python
import numpy as np

# x정의
x = np.linspace(-5,5,101)
x
```

Python

```
Out[-]
array([-5. , -4.9, -4.8, -4.7, -4.6, -4.5, -4.4, -4.3, -4.2, -4.1, -4. ,
       -3.9, -3.8, -3.7, -3.6, -3.5, -3.4, -3.3, -3.2, -3.1, -3. , -2.9,
       -2.8, -2.7, -2.6, -2.5, -2.4, -2.3, -2.2, -2.1, -2. , -1.9, -1.8,
       -1.7, -1.6, -1.5, -1.4, -1.3, -1.2, -1.1, -1. , -0.9, -0.8, -0.7,
       -0.6, -0.5, -0.4, -0.3, -0.2, -0.1,  0. ,  0.1,  0.2,  0.3,  0.4,
        0.5,  0.6,  0.7,  0.8,  0.9,  1. ,  1.1,  1.2,  1.3,  1.4,  1.5,
        1.6,  1.7,  1.8,  1.9,  2. ,  2.1,  2.2,  2.3,  2.4,  2.5,  2.6,
        2.7,  2.8,  2.9,  3. ,  3.1,  3.2,  3.3,  3.4,  3.5,  3.6,  3.7,
        3.8,  3.9,  4. ,  4.1,  4.2,  4.3,  4.4,  4.5,  4.6,  4.7,  4.8,
        4.9,  5. ])
```

```python
# 정규분포식
y = (1/np.sqrt(2*np.pi))*np.exp(-x**2 / 2)
y
```

Python

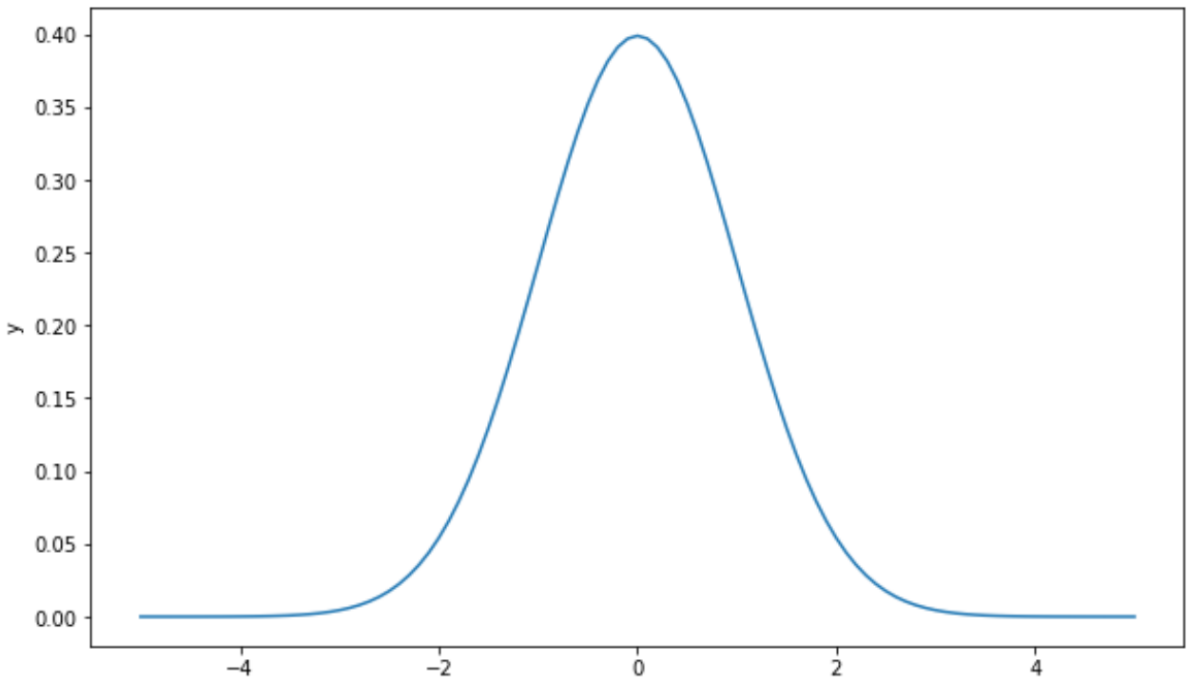
```
Out[-]
array([1.48671951e-06, 2.43896075e-06, 3.96129909e-06, 6.36982518e-06,
       1.01408521e-05, 1.59837411e-05, 2.49424713e-05, 3.85351967e-05,
       5.89430678e-05, 8.92616572e-05, 1.33830226e-04, 1.98655471e-04,
       2.91946926e-04, 4.24780271e-04, 6.11901930e-04, 8.72682695e-04,
       1.23221917e-03, 1.72256894e-03, 2.38408820e-03, 3.26681906e-03,
       4.43184841e-03, 5.95253242e-03, 7.91545158e-03, 1.04209348e-02,
       1.35829692e-02, 1.75283005e-02, 2.23945303e-02, 2.83270377e-02,
       3.54745928e-02, 4.39835960e-02, 5.39909665e-02, 6.56158148e-02,
       7.89501583e-02, 9.40490774e-02, 1.10920835e-01, 1.29517596e-01,
       1.49727466e-01, 1.71368592e-01, 1.94186055e-01, 2.17852177e-01,
       2.41970725e-01, 2.66085250e-01, 2.89691553e-01, 3.12253933e-01,
       3.33224603e-01, 3.52065327e-01, 3.68270140e-01, 3.81387815e-01,
       3.91042694e-01, 3.96952547e-01, 3.98942280e-01, 3.96952547e-01,
       3.91042694e-01, 3.81387815e-01, 3.68270140e-01, 3.52065327e-01,
       3.33224603e-01, 3.12253933e-01, 2.89691553e-01, 2.66085250e-01,
       2.41970725e-01, 2.17852177e-01, 1.94186055e-01, 1.71368592e-01,
       1.49727466e-01, 1.29517596e-01, 1.10920835e-01, 9.40490774e-02,
       7.89501583e-02, 6.56158148e-02, 5.39909665e-02, 4.39835960e-02,
       3.54745928e-02, 2.83270377e-02, 2.23945303e-02, 1.75283005e-02,
       1.35829692e-02, 1.04209348e-02, 7.91545158e-03, 5.95253242e-03,
       4.43184841e-03, 3.26681906e-03, 2.38408820e-03, 1.72256894e-03,
       1.23221917e-03, 8.72682695e-04, 6.11901930e-04, 4.24780271e-04,
       2.91946926e-04, 1.98655471e-04, 1.33830226e-04, 8.92616572e-05,
       5.89430678e-05, 3.85351967e-05, 2.49424713e-05, 1.59837411e-05,
       1.01408521e-05, 6.36982518e-06, 3.96129909e-06, 2.43896075e-06,
       1.48671951e-06])
```

Python

```python
# 정규분포 그리기
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize = (10,6))
plt.plot(x,y)
plt.ylabel("y")
plt.show()
```

Python

Out[-]



## 선형 회귀 분석

- 결정 계수
  - 선형 회귀 분석에서 모델이 얼마나 적합한지 평가
  - 범위는 [0,1]
  - 1에 가까울 수록 독립변수와 종속변수를 잘 설명하고 있음

$$R^2 := \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

    - SST : 실제값과 종속변수의 표본평균의 차의 제곱합
    - SSR : 예측값과 종속변수의 표본평균의 차의 제곱합
    - SSE : 예측값과 실제값의 차의 제곱합
    - SST = SSR + SSE

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

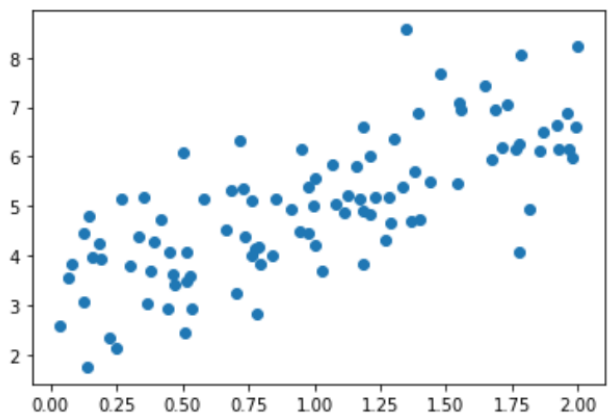
Python

```python
# 랜덤한 임의의 X,Y 값 생성
np.random.seed(1200)
X = 2*np.random.rand(100,1)
print('X shape : ', X.shape)

Y = 3 + 2*X + np.random.randn(100,1)
print('Y shape : ', Y.shape)

plt.scatter(X,Y)
plt.show()
```

Python

Out[-]

X shape : (100, 1)
Y shape : (100, 1)



```python
lin_reg = LinearRegression()
lin_reg.fit(X,Y)
print(f'y절편: {lin_reg.intercept_}, 기울기: {lin_reg.coef_}')
```

Python

```
Out[-]
y절편: [3.18198669], 기울기: [[1.81375875]]
```

Python

```python
# X_test 생성
X_test = np.linspace(0,2,100).reshape(100,1)
print('X_test')
print(X_test[:6])
print(X_test[-6:])

# 예측
predictions = lin_reg.predict(X_test)
print('prediction : ', predictions[:6])
```

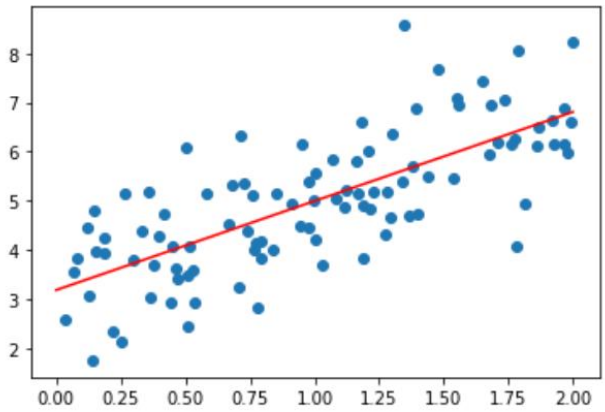
Python

```
Out[-]
X_test
[[0.        ]
 [0.02020202]
 [0.04040404]
 [0.06060606]
 [0.08080808]
 [0.1010101 ]]
[[1.8989899 ]
 [1.91919192]
 [1.93939394]
 [1.95959596]
 [1.97979798]
 [2.        ]]
prediction :  [[3.18198669]
 [3.21862828]
 [3.25526987]
 [3.29191146]
 [3.32855306]
 [3.36519465]]
```

```python
# 그래프 그리기
plt.scatter(X,Y)
plt.plot(X_test, predictions,'r')
plt.show()
```

Python

Out[-]



## 비선형 회귀 분석

```python
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

Python

```python
# training Set 만들기
# range : -3 <= x < 3 의 x값 100개
np.random.seed(1216)
X = 6*np.random.rand(100,1)-3
print('X = ', X[:6])


# y값 랜덤 생성
Y = 0.3 + 2 *X + X**2 + np.random.randn(100,1)
print('Y = ', Y[:6])
```

Python

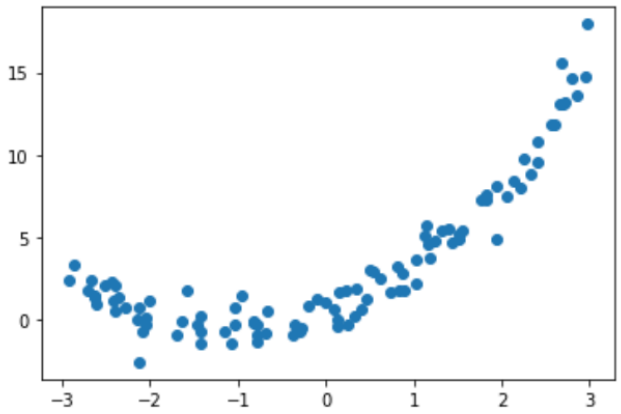
```
Out[-]
X =  [[ 2.9735891 ]
 [ 0.74046546]
 [ 2.25656212]
 [ 2.33626717]
 [-0.67942513]
 [ 1.13951938]]
Y =  [[17.94142067]
 [ 1.59013283]
 [ 9.77487261]
 [ 8.76874343]
 [-0.83438259]
 [ 5.68952924]]
```

Python

```python
# 그래프 구현
plt.scatter(X,Y)
plt.show()
```

Python

Out[-]



```python
# poly_feature 생성하기
poly_feature = PolynomialFeatures(degree=2, include_bias = False)

# X에 poly_featrue 적용
X_poly = poly_feature.fit_transform(X)
print('X poly = ', X_poly[:6])
```

Python

```
Out[-]
X poly =  [[ 2.9735891   8.84223212]
 [ 0.74046546  0.5482891 ]
 [ 2.25656212  5.09207262]
 [ 2.33626717  5.45814428]
 [-0.67942513  0.46161851]
 [ 1.13951938  1.29850441]]
```

Python

```python
# LinearRegression 객체 생성
lin_reg = LinearRegression()

# 회귀분석 실행 매소드
lin_reg.fit(X_poly, Y)

print('intercept : ', lin_reg.intercept_)
print('coefficients : ', lin_reg.coef_)

# y = b + a1 * x + a2 * x^2
```

Python

```
Out[-]
intercept :  [0.33611387]
coefficients :  [[2.02456163 0.97973891]]
```

Python

```python
# X_test 만들기
# linspace 함수 : -3~3사이에서 100개의 균일한 간격으로 점 생성
# reshape 함수 : 데이터 타입을 100,1 의 행렬로 변환

X_test = np.linspace(-3,3,100).reshape(100,1)
print(X_test[:6])
print(X_test[-6:])
```

Python

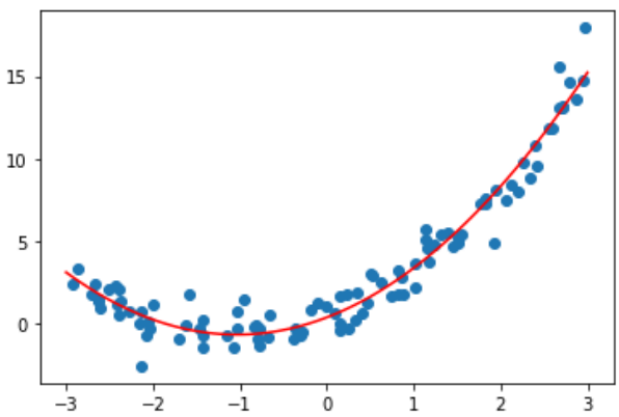
```
Out[-]
[[-3.        ]
 [-2.93939394]
 [-2.87878788]
 [-2.81818182]
 [-2.75757576]
 [-2.6969697 ]]
[[2.6969697 ]
 [2.75757576]
 [2.81818182]
 [2.87878788]
 [2.93939394]
 [3.        ]]
```

Python

```python
# X_test를 회귀분석에 맞게 핏하기
X_test_poly = poly_feature.fit_transform(X_test)

# X_test로 예측해보기
y_pred = lin_reg.predict(X_test_poly)

#예측 그래프 그리기
plt.scatter(X,Y)
plt.plot(X_test, y_pred, 'r')
plt.show()
```

Python

Out[-]



## 선형 회귀 분석 실습

```python
from sklearn.datasets import make_regression
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```

Python

```python
# 데이터 생성하기
x,y = make_regression(n_samples = 500, n_features=1, n_informative = 1, noise = 25,
                      random_state = 10)

# train data 와 test data 분류 (7:3)
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3,
                                                    random_state = 10)
```

Python

```python
# 선형회귀에 적용하기
line = LinearRegression().fit(x_train, y_train)

# 예측하기
pred = line.predict(x_test)
```

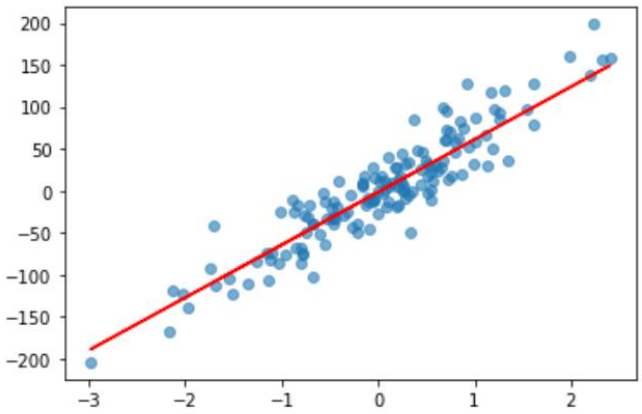
Python

```python
# 그래프 그리기
fig, ax = plt.subplots()
ax.scatter(x_test, y_test, alpha = 0.6)
ax.plot(x_test, pred, color = 'red')
```

Python

Out[-]

[<matplotlib.lines.Line2D at 0x1633d205c08>]




```python
# 평가하기
print('R2 : ', r2_score(y_test, pred))
```

Python

```
Out[-]
R2 :  0.8535702276418713
```

Python

## 행렬

- 수 또는 문자를 직사각형 모양으로 배열하여 괄호로 묶은 것
- 차원 : 행렬의 크기

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

l[1][2]
```

Python

```
Out[-]
6
```

Python

```python
import numpy as np

matrix = np.array(l)
print(matrix.ndim)
print(matrix.shape)
# test = 3  # 스칼라
test= [[[[3, 2, 1, 3, 5, 4, 2]]]] # 괄호 개수 = 차원 수
test_matrix = np.array(test)
print(test_matrix.ndim)
print(test_matrix.shape)
```

Python

```
Out[-]
2
(3, 3)
4
(1, 1, 1, 7)
```

Python

```python
matrix = np.arange(30).reshape(2, 3, 5)
print(matrix)
print(matrix[0][1][3])
```

Python

```
Out[-]
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]
  [10 11 12 13 14]]

 [[15 16 17 18 19]
  [20 21 22 23 24]
  [25 26 27 28 29]]]

8
```

Python

```python
matrix + matrix
```

Python

```
Out[-]
array([[[ 0,  2,  4,  6,  8],
        [10, 12, 14, 16, 18],
        [20, 22, 24, 26, 28]],

       [[30, 32, 34, 36, 38],
        [40, 42, 44, 46, 48],
        [50, 52, 54, 56, 58]]])
```

Python

## 행렬의 연산

```python
matrix = np.arange(15).reshape(3, 5)
print(matrix)
```

Python

```
Out[-]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

Python

```python
print(matrix * 2)
print(matrix / 2)
print(matrix + 2)
print(matrix - 2)
```

Python

```
Out[-]
[[ 0  2  4  6  8]
 [10 12 14 16 18]
 [20 22 24 26 28]]

[[0.  0.5 1.  1.5 2. ]
 [2.5 3.  3.5 4.  4.5]
 [5.  5.5 6.  6.5 7. ]]

[[ 2  3  4  5  6]
 [ 7  8  9 10 11]
 [12 13 14 15 16]]

[[-2 -1  0  1  2]
 [ 3  4  5  6  7]
 [ 8  9 10 11 12]]
```

Python

```python
test = np.zeros((3,4))
test
```

Python

```
Out[-]
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

Python

## 행렬의 곱셈

```
  키보드   마우스
S  5000     8000
L  7000     3000

        바울랩
키보드    31
마우스    40

5000*31 + 8000*40
7000*31 + 3000*40
```

Plain Text

```python
x = np.array([[5000, 8000],
              [7000, 3000]])
y = np.array([[31],
              [40]])
print(x @ y)
print(5000*31 + 8000*40)
print(7000*31 + 3000*40)
```

Python

```
Out[-]
[[475000]
 [337000]]
475000
337000
```

## 행렬의 유형

1. 단위 행렬 : 주 대각원소가 모두 1이고 나머지 원소가 0으로 이루어진 정사각행렬
2. 정방 행렬 : 행과 열의 수가 같은 행렬
3. 영행렬
    - 모든 원소가 0으로 이루어진 행렬
    - 반드시 정방행렬일 필요는 없음
    - 대수법칙의 덧셈의 항등원에 해당
4. 전치 행렬 : 원래의 행렬의 행과 열을 바꾼 행렬
5. 역행렬
    - 단위 행렬이 나오도록 특정 행렬에 곱하여진 정방 행렬
    - 행렬 A X B가 단위 행렬이 되면 A의 역행렬은 B
    - A와 B는 정방 행렬이어야 함. 단, 모든 정방 행렬이 역행렬을 가지는 것은 아님

오리와 양의 머리수 7
오리와 양의 다리수 22

이 문제를 행렬로 풀어보세요.

1x + 1y = 7
2x + 4y = 22

1 1 x 7
2 4 y 22

```python
x = np.random.randint(1, 10, size=[3, 3])
print(x)
y = np.linalg.inv(x)
print(x @ y)
```

Python

```
Out[-]
[[5 6 5]
 [8 4 6]
 [5 1 4]]
[[ 1.00000000e+00 -4.44089210e-16  2.22044605e-16]
 [ 0.00000000e+00  1.00000000e+00 -4.44089210e-16]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

Python

```python
x = np.array([[1, 1], [2, 4]])
y = np.linalg.inv(x) # np.linalg : 선형 대수 함수, inv : 역행렬
z = np.array([[7], [22]])

print(y @ z)
```

Python

```
Out[-]
[[3.]
 [4.]]
```

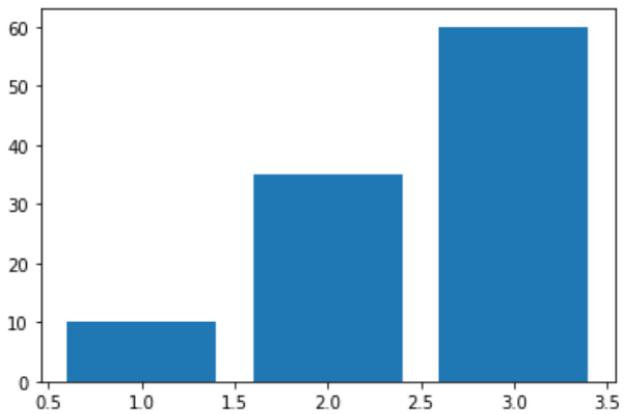
Python

```python
%matplotlib inline
import matplotlib.pyplot as plt

print(sum(matrix))
print(np.sum(matrix))
print(np.sum(matrix, axis=0))
print(np.sum(matrix, axis=1))

student = np.sum(matrix, axis=1)
plt.bar([1,2,3],student)
plt.show()
```

Python

```
Out[-]
[15 18 21 24 27]
105
[15 18 21 24 27]
[10 35 60]
```

Python



```python
print(matrix[matrix > 5])
print(matrix[matrix % 2 == 0])
```

Python

```
Out[-]
[ 6  7  8  9 10 11 12 13 14]
[ 0  2  4  6  8 10 12 14]
```

Python

```python
print(matrix[matrix > 5])
print(matrix[matrix % 2 == 0])
print(matrix[1][2])
print(matrix[1, 2])
mask = matrix % 2 == 0
print(mask)
print(matrix[mask])
```

Python

```
Out[-]
[ 6  7  8  9 10 11 12 13 14]
[ 0  2  4  6  8 10 12 14]
7
7
[[ True False  True False  True]
 [False  True False  True False]
 [ True False  True False  True]]
[ 0  2  4  6  8 10 12 14]
```

Python

초판 1쇄 발행 | 2020년 4월 7일
지은이 | 이호준 김유진 김혜원 이현창 장하림 차경림 이숭신 김대현
편 집 | 이호준
총 괄 | 이호준
펴낸곳 | 사도출판
주 소 | 제주특별자치도 제주시 동광로 137 대동빌딩 2층
표지디자인 | 차경림
홈페이지 | http://www.paullab.co.kr
E - mail | paul-lab@naver.com
ISBN | 979-11-88786-31-2

---

# 제주 하간디 이신 데이터들 Python으로 몬딱 분석해불게!

비매품/무료
15560
9 791188 786312
ISBN 979-11-88786-31-2 (PDF)