Introduction to NumPy

NumPy

- 고성능의 과학 계산 컴퓨팅과 데이터 분석에 필요한 기본 패키지로 Numerical Python의 줄임말

- 지원 기능
  - » 빠르고 메모리를 효율적으로 사용하여 벡터 산술연산과 세련된 브로드케스팅 기능을 제공하는 다차원 배열 ndarray
  - » 반복문을 작성할 필요 없이 전체 데이터 배열에 대해 빠른 연산을 제공하는 표준 함수
  - » 배열 데이터를 디스크에 쓰거나 읽을 수 있는 도구
  - » 메모리에 올려진 파일을 사용하는 도구
  - » 선형대수, 난수 발생기, 푸리에 변환 기능
  - » C, C++, 포트란으로 쓰여진 코드를 통합하는 도구

NumPy 사용 환경

- 설치
  - » 표준 파이썬 배포판에 NumPy 모듈이 포함되지 않아서 별도 설치 필요

```
> conda install numpy
```

- 사용
  - » NumPy를 사용하기 전에 임포트

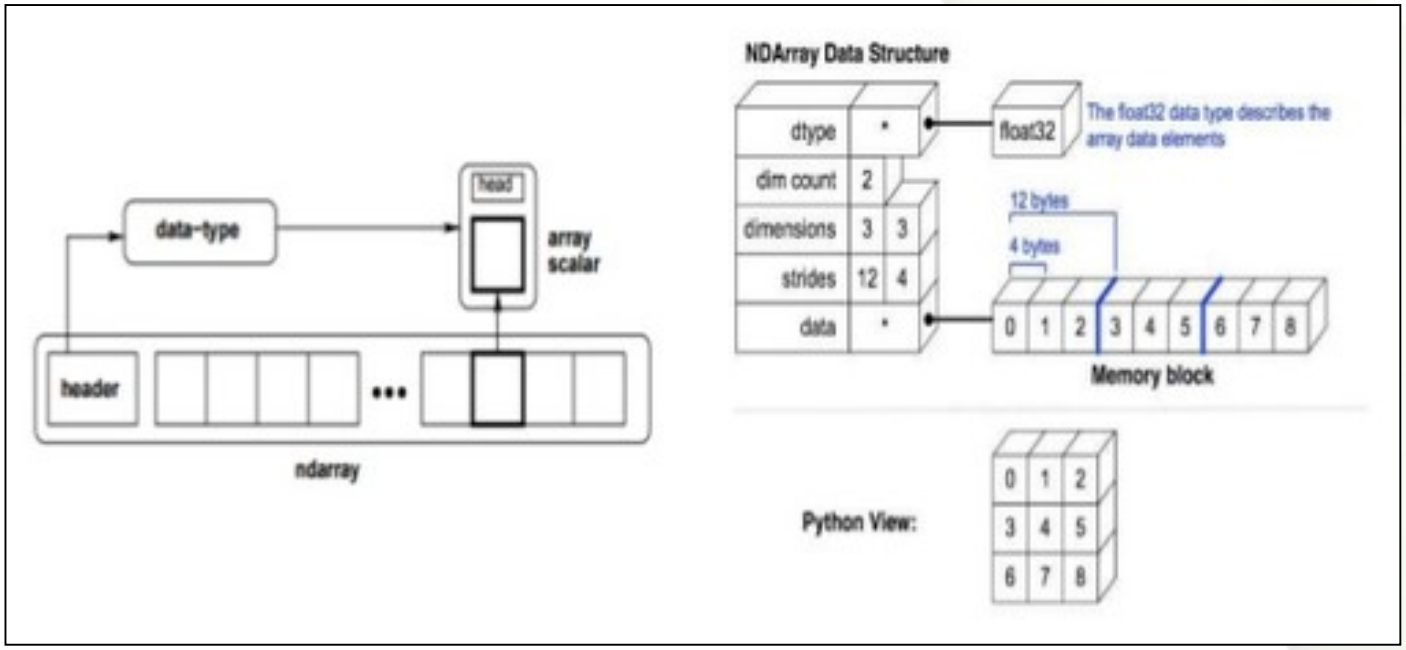
```
import numpy
```

```
import numpy as np
```

ndarray 객체

- NumPy에서 가장 중요한 객체는 다차원 배열 타입인 ndarray

- 같은 자료형을 갖는 다수의 데이터의 집합을 표현
  - » ndarray에 포함된 객체는 0에서 시작되는 순서 번호를 사용해서 접근
  - » ndarray에 포함된 객체는 같은 크기의 메모리 블록에 저장됨
  - » ndarray에 포함된 객체는 연속된 메모리 공간에 할당됨



NumPy 자료형

- NumPy는 파이썬이 제공하는 기본 자료형 보다 많은 자료형 제공

자료형	설명
bool_	Boolean (True or False)
int_	기본 정수 타입 (C언어의 long과 같은 형식, 8 byte 또는 4 byte)
intc	C언어의 int와 같은 형식(int32 또는 int64)
intp	순서번호로 사용되는 형식(C언어의 C ssize_t와 같은 형식 4byte of 8byte)
int8	정수 (Byte, -128 to 127) .
int16	정수 (-32768 to 32767)
int32	정수 (-2147483648 to 2147483647)
uint64	정수 (-9223372036854775808 to 9223372036854775807)
uint8	부호 없는 정수 (0 to 255)
uint16	부호 없는 정수 (0 to 65535)

NumPy 자료형

- NumPy는 파이썬이 제공하는 기본 자료형 보다 많은 자료형 제공 (계속)

자료형	설명
uint32	부호 없는 정수 (0 to 4294967295)
uint64	부호 없는 정수 (0 to 18446744073709551615)
float_	float64의 축약형
float16	반 정밀도 부동 소수점 (2 byte)
float32	단 정밀도 부동 소수점 (4 byte)
float64	배 정밀도 부동 소수점 (8 byte)
complex_	complex128의 축약형
complex64	복소수 (실수부, 허수부 각각 4byte)
complex128	복소수 (실수부, 호수부 각각 8byte)

NumPy 자료형

- 기본 타입 자료형 사용 예제

```
x = np.array([1, 2, 3], dtype='f')
x.dtype
```

```
dtype('float32')
```

```
x = np.array([1, 2, 3], dtype='i2')
x.dtype
```

문자열 i1, i2, i4, i8로 정수형 표현

```
dtype('int16')
```

- 자료형 표시 문자열 종류

```
'b' – (signed) byte
'i' – (signed) integer
'u' – unsigned integer
'f' – floating-point
```

```
'?' – boolean
'c' – complex floating-point
'O’ – (Python) Object
'U' – Unicode string
```

ndarray 생성

- 다양한 방법으로 ndarray 생성 가능
  - » 그 중 기본이 되는 방법은 numpy.array를 사용

```
numpy.array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)
```

전달인자	설명
object	배열을 반환하는 함수 또는 데이터 목록
dtype	배열 요소의 데이터 타입 (선택적)
copy	객체의 복사 여부 (선택적, 기본값은 True)
order	배열의 메모리 저장 구조 지정 C (row major) or F (column major)
subok	상속받은 클래스일 경우 True → 상속 클래스 형식으로, False → 기본 배열 클래스 형식으로 생성
ndmin	결과 배열의 최소 차원(dimensions) 수

ndarray 생성

- 리스트 전달인자로 생성

```
a=np.array([1,2,3])
print (a)
```

```
[1 2 3]
```

- 다차원 배열

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

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

- 최소 차원을 설정해서 생성

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

```
[[1 2 3 4 5]]
```

- 자료형을 설정해서 생성

```
a = np.array([1, 2, 3], dtype=complex)
print (a)
```

```
[ 1.+0.j  2.+0.j  3.+0.j]
```

ndarray 속성

- ndarray.shape
  - » 배열의 차수(행/열) 정보 : 읽기/쓰기 가능

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

```
(2, 3)
```

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

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

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

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

- ndarray.ndim
  - » 배열의 차원의 개수 정보

```
a = np.arange(24)
print(a.ndim)
b = a.reshape(2,4,3)
print(b.ndim)
```

```
1
3
```

ndarray 생성 함수

함수	설명
array	입력 데이터(리스트,튜플,배열 또는 순차형 데이터)를 ndarray로 변환. dtype이 명시되지 않으면 자료형을 추정하고 입력데이터는 복사해서 생성
asarray	입력 데이터를 ndarray로 변환하지만 이미 ndarray인 데이터일 경우 복사되지 않음
arange	내장 range함수와 유사하나 리스트 대신 ndarray 반환
linspace	주어진 데이터 범위를 일정한 간격으로 분할해서 배열을 만든 후 튜플로 반환
ones	주어진 dtype과 모양으로 배열을 생성하고 내용을 모두 1로 초기화
zeros	ones와 같지만 내용을 0으로 채움
empty	메모리를 할당하여 새로운 배열을 생성하지만 값을 초기화하지 않음

ndarray 생성 함수

- ndarray 생성 함수 사용 사례
  - » numpy.empty

```
x = np.empty([3,2], dtype=int)
print(x)
```

```
[[0 0]
 [0 0]
 [0 0]]
```

  - » numpy.zeros

```
x = np.zeros((5,), dtype="int")
print(x)
```

```
[0 0 0 0 0]
```

  - » numpy.ones / 자료형 지정

```
x = np.ones([2,2], dtype=int)
print(x)
```

```
[[1 1]
 [1 1]]
```

ndarray 생성 함수

- ndarray 생성 함수 사용 사례 (계속)
  - » numpy.asarray

```
x = [1,2,3]
a = np.asarray(x)
print (a)
```

```
[1  2  3]
```

```
x = [1,2,3]
a = np.asarray(x, dtype = float)
print (a)
```

```
[ 1.  2.  3.]
```

```
x = (1,2,3)
a = np.asarray(x)
print (a)
```

```
[1  2  3]
```

```
x = [(1,2,3),(4,5)]
a = np.asarray(x)
print (a)
```

```
[(1, 2, 3) (4, 5)]
```

ndarray 생성 함수

- ndarray 생성 함수 사용 사례 (계속)
  - » numpy.arange

```
x = np.arange(5)
print (x)
```

```
[0  1  2  3  4]
```

```
x = np.arange(5, dtype = float)
print (x)
```

```
[0.  1.  2.  3.  4.]
```

```
x = np.arange(10,20,2)
print (x)
```

```
[10  12  14  16  18]
```

ndarray 생성 함수

- ndarray 생성 함수 사용 사례 (계속)
  - » numpy.linspace

```
a = np.linspace(1.0, 2.0, num = 5)
print (a)
```

```
[ 1.    1.25  1.5   1.75  2.  ]
```

```
x = np.linspace(10,20, 5, endpoint = False)
print (x)
```

```
[10.   12.   14.   16.   18.]
```

```
x = np.linspace(1,2,5, retstep = True)
print (x)
```

```
(array([ 1.  ,  1.25,  1.5 ,  1.75,  2.  ]), 0.25)
```

색인과 슬라이싱

- ndarray 객체의 내용은 인덱싱, 슬라이싱 기법을 사용하여 접근하거나 수정할 수 있음

```
a = np.arange(10)
s = slice(2,7,2)
print (a[s])
```

```
[2  4  6]
```

```
a = np.arange(10)
b = a[2:7:2]
print (b)
```

```
[2  4  6]
```

```
a = np.arange(10)
b = a[5]
print (b)
```

```
5
```

```
a = np.arange(10)
print (a[2:])
```

```
[2  3  4  5  6  7  8  9]
```

색인과 슬라이싱

- ndarray 객체의 내용은 인덱싱, 슬라이싱 기법을 사용하여 접근하거나 수정할 수 있음

```
a = np.arange(10)
print (a[2:5])
```

```
[2  3  4]
```

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

print (a[1:])
```

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]

[[3 4 5]
 [4 5 6]]
```

색인과 슬라이싱

- ndarray 객체의 내용은 인덱싱, 슬라이싱 기법을 사용하여 접근하거나 수정할 수 있음

```
x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2], [0,1,0]]
print (y)
```

```
[1  4  5]
```

```
x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])

rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]

print(y)
```

```
[[ 0  2]
 [ 9 11]]
```

```
x = np.array([[ 0,  1,  2],[ 3,  4,  5],
              [ 6,  7,  8],[ 9, 10, 11]])
z = x[1:4,1:3]
print (z)
y = x[1:4,[1,2]]
print (y)
```

```
[[ 4  5]
 [ 7  8]
 [10 11]]

[[ 4  5]
 [ 7  8]
 [10 11]]
```

색인과 슬라이싱

- ndarray 객체의 내용은 인덱싱, 슬라이싱 기법을 사용하여 접근하거나 수정할 수 있음 (계속)

```
x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])

print(x[x > 5])
```

```
[ 6  7  8  9 10 11]
```

```
a = np.array([np.nan, 1,2,np.nan,3,4,5])

print(a[~np.isnan(a)])
```

```
[ 1.  2.  3.  4.  5.]
```

브로드캐스팅 (Broadcasting)

- 연산 과정에서 형태가 다른 배열을 처리하는 기능
  - » 형태가 같은 배열 연산 처리

```
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print(c)
```

```
[10   40   90   160]
```

  - » 형태가 다른 배열 연산 처리

```
a = np.array([[0.0,0.0,0.0],[10.0,10.0,10.0],[20.0,20.0,20.0],[30.0,30.0,30.0]])
b = np.array([1.0,2.0,3.0])

print (a + b)
```

```
[[ 1. 2. 3.]
 [ 11. 12. 13.]
 [ 21. 22. 23.]
 [ 31. 32. 33.]]
```

배열 반복자

- 배열을 순회하기 위해 numpy.nditer 객체 제공
  - » 다차원 배열을 효과적으로 순회할 수 있는 기능 제공

```
a = np.arange(0,60,5)
a = a.reshape(3,4)

for x in np.nditer(a):
   print(x, end=' ')
```

```
0 5 10 15 20 25 30 35 40 45 50 55
```

```
a = np.arange(0,60,5)
a = a.reshape(3,4)
b = a.T

for x in np.nditer(b):
   print (x, end=' ')
```

```
0 5 10 15 20 25 30 35 40 45 50 55
```

출력 순서는 메모리 저장 순서를 따름

배열 반복자

- 배열을 순회하기 위해 numpy.nditer 객체 제공 (계속)
  - » 다차원 배열을 효과적으로 순회할 수 있는 기능 제공

```
a = np.arange(0,60,5)
a = a.reshape(3,4)

for x in np.nditer(a, order = 'C'):
   print (x, end=' ')
print ('\n')

for x in np.nditer(a, order = 'F'):
   print (x, end=' ')
```

```
0 5 10 15 20 25 30 35 40 45 50 55

0 20 40 5 25 45 10 30 50 15 35 55
```

order는 데이터 읽기 순서

Pandas

Pandas

- 강력한 데이터 구조를 사용해서 고성능 데이터 조작 및 분석 도구를 제공하는 오픈 소스 라이브러리 (Panel Data를 줄여서 Panda로 명명)

- 파이썬이 기본 제공 기능 보완 및 확장
  - » 데이터 원본의 종류에 영향 받지 않고 데이터 로딩, 준비, 처리, 모델링, 분석 등의 작업 수행 가능

- 설치

```
> conda install pandas
```

- 사용

```
from pandas import Series, DataFrame
import pandas as pd
```

데이터 구조

- Pandas는 세 종류의 데이터 구조 제공

데이터 구조	차원	설명
Series	1	▪ 1차원 명명된 단일 데이터 배열 ▪ 크기 변경 불가
Data Frames	2	▪ 축(행과 열)에 이름을 붙일 수 있음 ▪ 행, 열의 크기 변경 가능 ▪ 컬럼별로 다른 자료형 사용 (한 컬럼의 데이터는 동일 자료형) ▪ 행과 열을 대상으로 산술 연산 수행 가능
Panel	3	▪ 이름 붙인 크기 변경 가능한 DataFrame 배열 ▪ deprecated

데이터 구조

- Series
  - » 같은 자료형의 데이터만 저장하는 1차원 배열 구조

10	23	56	17	52	61	73	90	26	72

- DataFrame
  - » 열 별로 다른 데이터를 저장할 수 있는 2차원 배열 구조

Column

Row

Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78

Series 생성

- pandas.Series 함수
  - » 형식

```
pandas.Series(data, index, dtype, copy)
```

전달인자	설명
data	▪ Series를 만들기 위해 사용하는 데이터 ▪ ndarray, 리스트 등 다양한 형식의 데이터 사용
index	▪ 각 데이터의 인덱스로 사용할 중복되지 않는 데이터 목록 ▪ 인덱스의 개수는 데이터 개수와 같아야 함 ▪ 생략되면 np.arange(n) 함수 사용
dtype	▪ 데이터 형식 ▪ 생략되면 데이터를 통해 유추함
copy	▪ 데이터 복사 여부 (기본 값은 False)

Series 생성

- 빈 Series 생성

```
s = pd.Series()
print(s)
```

```
Series([], dtype: float64)
```

- ndarray를 사용해서 생성

```
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print(s)
```

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

Series 생성

- dict를 사용해서 생성

```
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print(s)
```

```
a    0.0
b    1.0
c    2.0
dtype: float64
```

```
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=['b','c','d','a'])
print(s)
```

```
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

Series 생성

- 단일 값으로 생성

```
s = pd.Series(5, index=[0, 1, 2, 3])
print(s)
```

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

Series 데이터 사용

- 위치 값을 사용해서 데이터 접근

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[0])
```

```
1
```

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[:3])
```

```
a    1
b    2
c    3
dtype: int64
```

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[-3:])
```

```
c    3
d    4
e    5
dtype: int64
```

Series 데이터 사용

- 이름을 사용해서 데이터 접근

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s['a'])
```

```
1
```

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s[['a','c','d']])
```

```
a    1
c    3
d    4
dtype: int64
```

```
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s['f'])
```

```
...
KeyError   ...
...
```

Data Frame 생성

- pandas.DataFrame 함수 사용
  - » 형식

```
pandas.DataFrame(data, index, columns, dtype, copy)
```

전달인자	설명
data	▪ 새 데이터프레임을 만드는데 사용할 데이터 ▪ ndarray, series, list, map, dict 및 다른 DataFrame 이 될 수 있음
index	▪ 생성된 데이터프레임의 행 식별자에 사용될 데이터 목록 ▪ 지정되지 않으면 np.arange(n) 함수 사용
columns	▪ 생성된 데이터프레임의 열 식별자에 사용할 데이터 목록 ▪ 지정되지 않으면 np.arange(n) 함수 사용
dtype	▪ 컬럼별 자료형
copy	▪ 데이터 복사 여부(기본 값은 False)

DataFrame 생성

- 빈 데이터프레임 생성

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

```
Empty DataFrame
Columns: []
Index: []
```

- 리스트를 사용해서 데이터프레임 생성

```
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print(df)
```

```
   0
0  1
1  2
2  3
3  4
4  5
```

DataFrame 생성

- 리스트를 사용해서 데이터프레임 생성 (계속)

```
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print(df)
```

```
 Name  Age
0    Alex   10
1     Bob   12
2  Clarke   13
```

```
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)
print(df)
```

```
   Name    Age
0    Alex  10.0
1     Bob  12.0
2  Clarke  13.0
```

DataFrame 생성

- ndarray / 리스트의 딕셔너리를 사용해서 데이터프레임 생성

```
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print(df)
```

```
 Age   Name
0   28    Tom
1   34   Jack
2   29  Steve
3   42  Ricky
```

```
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
print(df)
```

```
 Age   Name
rank1   28    Tom
rank2   34   Jack
rank3   29  Steve
rank4   42  Ricky
```

DataFrame 생성

- 딕셔너리의 리스트를 사용해서 데이터프레임 생성

```
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print(df)
```

```
 a   b     c
0  1   2   NaN
1  5  10  20.0
```

```
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second'])
print(df)
```

```
 a   b     c
first   1   2   NaN
second  5  10  20.0
```

DataFrame 생성

- 딕셔너리의 리스트를 사용해서 데이터프레임 생성 (계속)

```
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b'])
df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b1'])

print(df1)
print(df2)
```

```
        a   b
first   1   2
second  5  10

        a  b1
first   1 NaN
second  5 NaN
```

DataFrame 생성

- Series의 딕셔너리를 사용해서 데이터프레임 생성

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df)
```

```
 one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

DataFrame 데이터 사용

- 컬럼 식별자로 데이터 선택

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df['one'])
```

```
a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype: float64
```

DataFrame 데이터 사용

- 컬럼 추가

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)

df['three'] = pd.Series([10,20,30], index=['a','b','c'])
print(df)

df['four'] = df['one'] + df['three']
print(df)
```

```
 one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN
   one  two  three  four
a  1.0    1   10.0  11.0
b  2.0    2   20.0  22.0
c  3.0    3   30.0  33.0
d  NaN    4    NaN   NaN
```

DataFrame 데이터 사용

- 컬럼 삭제

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
     'three' : pd.Series([10,20,30], index=['a','b','c'])}
df = pd.DataFrame(d)
print(df)

del df['one']
print(df)

df.pop('two')
print(df)
```

```
 one  three  two
a  1.0   10.0    1
b  2.0   20.0    2
c  3.0   30.0    3
d  NaN    NaN    4
   three  two
a   10.0    1
b   20.0    2
c   30.0    3
d    NaN    4
   three
a   10.0
b   20.0
c   30.0
d    NaN
```

DataFrame 데이터 사용

- 행 선택 → loc[index-name], iloc[index-order]

```
df = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print(df.loc['b'])
```

```
one       2.0
three    20.0
two       2.0
Name: b, dtype: float64
```

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print(df.iloc[2])
```

```
one    3.0
two    3.0
Name: c, dtype: float64
```

DataFrame 데이터 사용

- 행 슬라이싱

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print(df[2:4])
```

```
 one  two
c  3.0    3
d  NaN    4
```

- 행 추가

```
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df = df.append(df2)
print(df)
```

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

DataFrame 데이터 사용

- 행 삭제

```
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])

df = df.append(df2)
df = df.drop(0)

print(df)
```

```
 a  b
1  3  4
1  7  8
```

Series 기본 함수

- Series에 사용할 수 있는 기본 함수 목록

속성/함수	설명
axes	행 식별자 리스트 반환
dtype	객체의 데이터 타입(dtype 형식) 반환
empty	Series에 데이터가 없으면 True 반환
ndim	데이터의 차수 반환
size	데이터의 개수 반환
values	ndarray 형식으로 변환된 데이터 반환
head()	처음 n개의 행 반환
tail()	마지막 n개의 행 반환

Series 기본 함수

- axes 속성

```
s = pd.Series(np.random.randn(4))
print(s.axes)
```

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

- empty 속성

```
s = pd.Series(np.random.randn(4))
print(s.empty)
```

```
False
```

- ndim 속성

```
s = pd.Series(np.random.randn(4))
print(s.ndim)
```

```
1
```

Series 기본 함수

- size 속성

```
s = pd.Series(np.random.randn(2))
print(s.size)
```

```
2
```

- values 속성

```
s = pd.Series(np.random.randn(4))
print(s.values)
```

```
[ 0.36731893  0.46945044  0.90285222  0.96697052]
```

- head(), tail() 함수

```
s = pd.Series(np.random.randn(4))
print(s.head(2))
print(s.tail(2))
```

```
0   -0.948594
1    0.624306
dtype: float64
2    1.230468
3   -0.318933
dtype: float64
```

데이터프레임 기본 함수

- 데이터프레임에 사용할 수 있는 기본 함수 목록

속성/함수	설명
T	전치 행렬 반환 (행과 열의 위치를 교환)
axes	행과 열 인덱스 식별자 리스트 반환
dtypes	객체의 데이터 타입 반환 (dtypes 형식)
empty	데이터프레임에 데이터가 없으면 True 반환
ndim	축의 개수를 반환
shape	데이터프레임의 행/열 정보를 튜플 형식으로 반환
size	데이터프레임에 저장된 데이터 개수 반환
values	데이터프레임의 데이터를 numpy 형식으로 반환
head()	처음 n개의 행 데이터 반환
tail()	마지막 n개의 행 데이터 반환

데이터프레임 기본 함수

- 예제에 사용할 데이터 생성

```
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),
 'Age':pd.Series([25,26,25,23,30,29,23]),
 'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

df = pd.DataFrame(d)
print(df)
```

```
 Age   Name  Rating
0   25    Tom    4.23
1   26  James    3.24
2   25  Ricky    3.98
3   23    Vin    2.56
4   30  Steve    3.20
5   29  Smith    4.60
6   23   Jack    3.80
```

데이터프레임 기본 함수

- T 속성

```
print(df.T)
```

```
  0      1      2      3      4      5      6
Age      25     26     25     23     30     29     23
Name    Tom  James  Ricky    Vin  Steve  Smith  Jack
Rating 4.23   3.24   3.98   2.56    3.2    4.6    3.8
```

- axes 속성

```
print(df.axes)
```

```
[RangeIndex(start=0, stop=7, step=1), Index(['Age', 'Name', 'Rating'],
dtype='object')]
```

- dtypes 속성

```
print(df.dtypes)
```

```
Age         int64
Name       object
Rating    float64
dtype: object
```

데이터프레임 기본 함수

- empty 속성

```
print(df.empty)
```

```
False
```

- ndim 속성

```
print(df.ndim)
```

```
2
```

- shape 속성

```
print(df.shape)
```

```
(7, 3)
```

데이터프레임 기본 함수

- size 속성

```
print(df.size)
```

```
21
```

- values 속성

```
print(df.values)
```

```
[[25 'Tom' 4.23]
 [26 'James' 3.24]
 [25 'Ricky' 3.98]
 [23 'Vin' 2.56]
 [30 'Steve' 3.2]
 [29 'Smith' 4.6]
 [23 'Jack' 3.8]]
```

- head(), tail() 함수

```
print(df.head(2))
print(df.tail(2))
```

```
  Age   Name  Rating
0   25    Tom    4.23
1   26  James    3.24
   Age   Name  Rating
5   29  Smith     4.6
6   23   Jack     3.8
```

matplotlib

matplotlib

- 데이터를 시각화하는 패키지
  - » Line Plot, Scatter Plot, Bar Plot, Box Plot 등 다양한 플롯 생성 가능
  - » 예제 확인 → https://matplotlib.org/gallery/index.html

- pylab 서브패키지
  - » pyplot + numpy
  - » matlab 수치해석 소프트웨어 시각화 명령을 거의 그대로 사용할 수 있는 명령어 집합 제공

- 설치 및 사용

```
> activate pyenv3
> conda install matplotlib
```

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

플롯의 구조

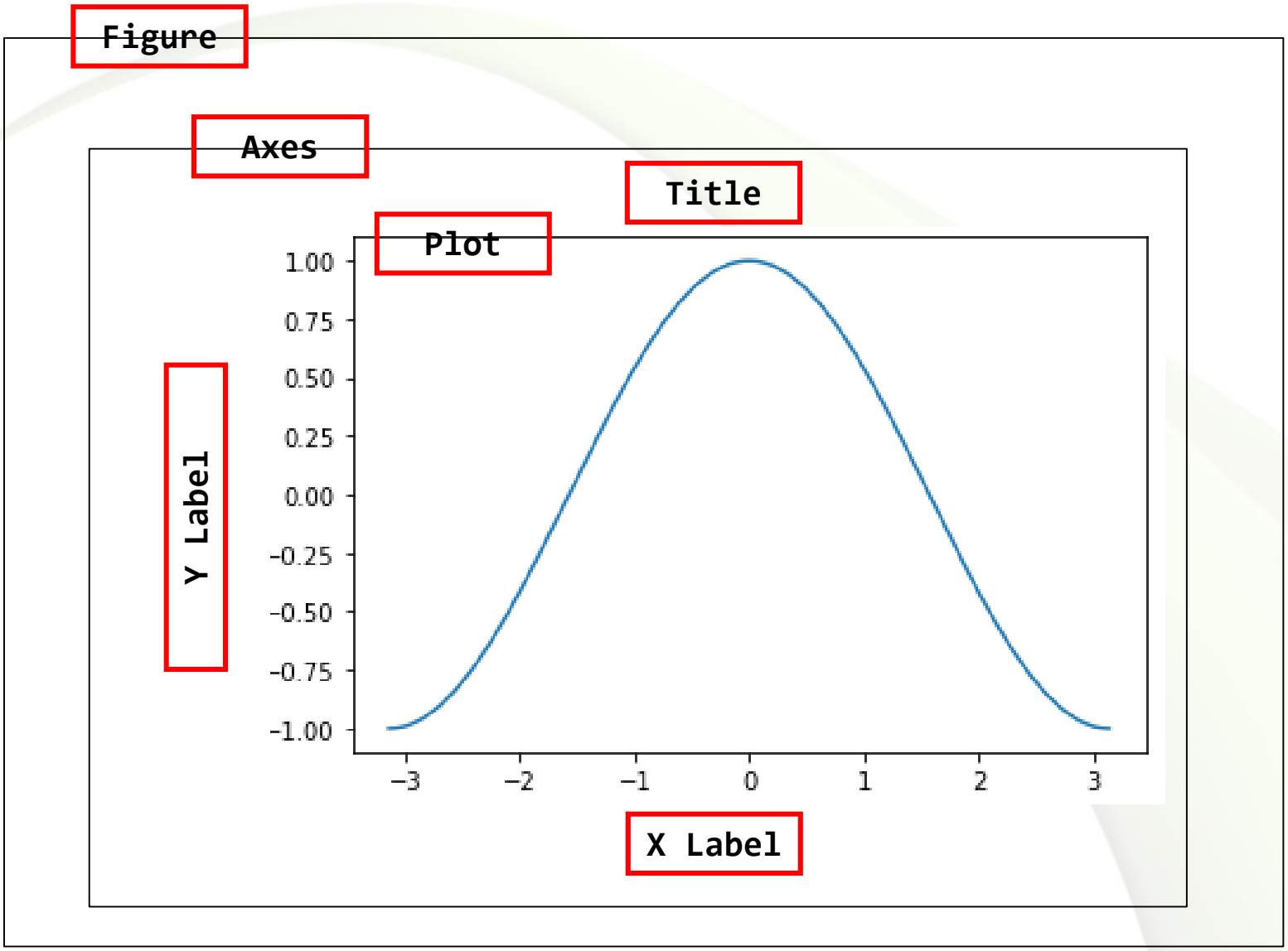
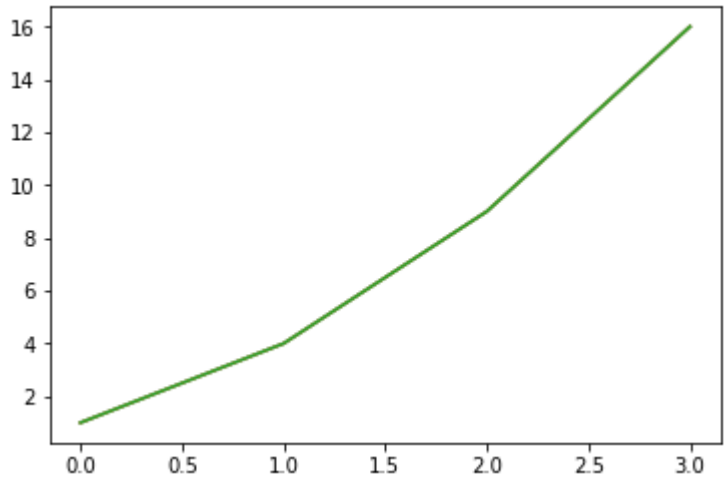


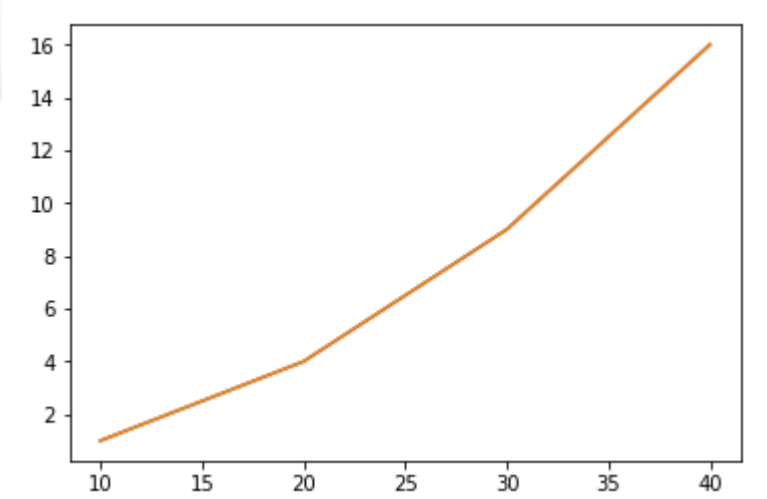
Figure
Axes
Title
Plot
Y Label
X Label
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
−3
−2
−1
0
1
2
3

선 그래프

- 주로 데이터가 시간, 순서 등에 따라 어떻게 변화하는지 보여주기 위해 사용

```
plt.plot([1, 4, 9, 16])
plt.show()
```






```
plt.plot([10, 20, 30, 40], [1, 4, 9, 16])
plt.show()
```

스타일 지정

- plot 명령어의 추가 전달인자를 사용해서 플롯의 모양 변경

```
plt.plot(data, '속성문자열')
```

- 색상 스타일
  - » 색 이름 또는 #문자로 시작하는 16진수 RGB 값 사용
  - » 주요 색상 문자열

문자열	약자	문자열	약자
blue	b	magenta	m
green	g	yellow	y
red	r	black	k
cyan	c	white	w

- 선 스타일 문자열

문자열	약자	문자열	약자
-	직선 스타일	-.	대시-점 스타일
--	대시 스타일	:	점 스타일

스타일 지정

- 마커 스타일
  - » 데이터의 위치를 표시하는 기호인 마크의 표시 모양
  - » 주요 마커 문자열

마커 문자	의미	마커 문자	의미
.	point marker	4	tri_right marker
,	pixel marker	s	square marker
o	circle marker	p	pentagon marker
v	triangle_down marker	*	star marker
^	triangle_up marker	h	hexagon1 marker
<	triangle_left marker	H	hexagon2 marker
>	triangle_right marker	+	plus marker
1	tri_down marker	x	x marker
2	tri_up marker	D	diamond marker
3	tri_left marker	d	thin_diamond marker

스타일 지정

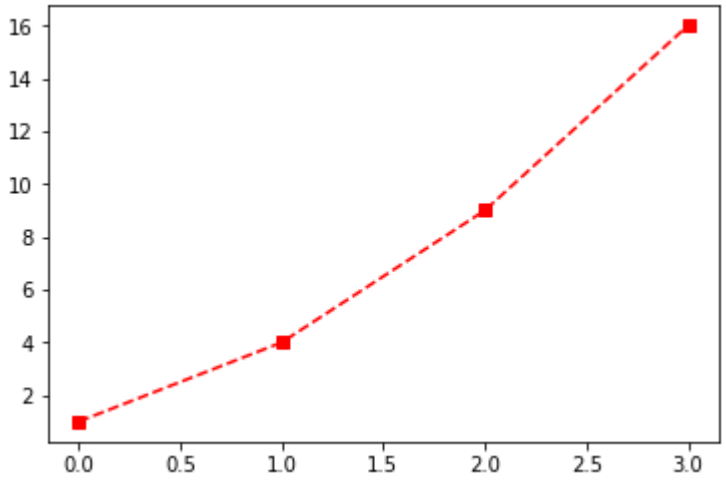
- 기타 스타일

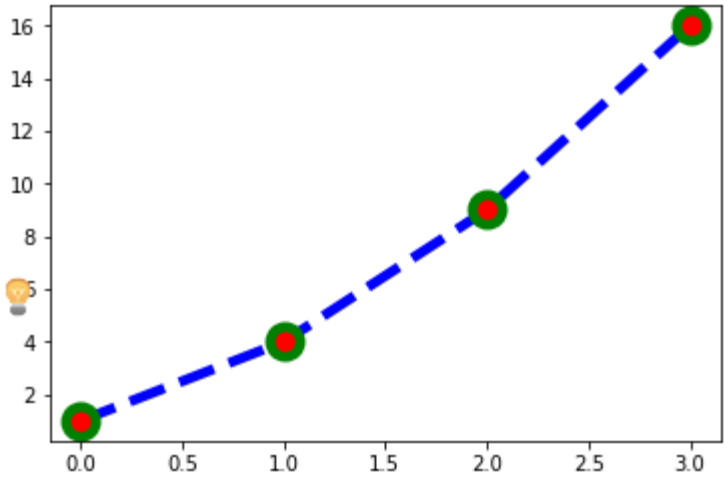
스타일 문자열	약자	의미
color	c	선 색깔
linewidth	lw	선 굵기
linestyle	ls	선 스타일
marker		마커 종류
markersize	ms	마커 크기
markeredgecolor	mec	마커 선 색깔
markeredgewidth	mew	마커 선 굵기
markerfacecolor	mfc	마커 내부 색깔

그래프 설정

- 스타일을 적용한 그래프

```
plt.plot([1, 4, 9, 16], 'rs--')
plt.show()
```



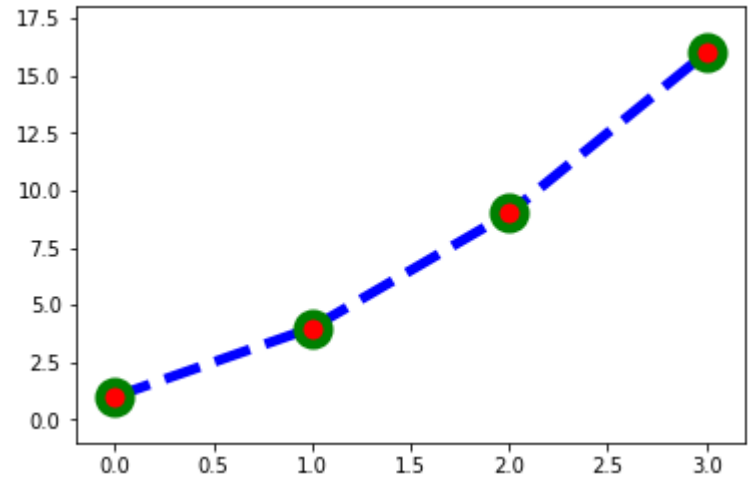



```
plt.plot([1, 4, 9, 16], c="b", lw=5, ls="--",
         marker="o", ms=15, mec="g", mew=5, mfc="r")
plt.show()
```

그래프 설정

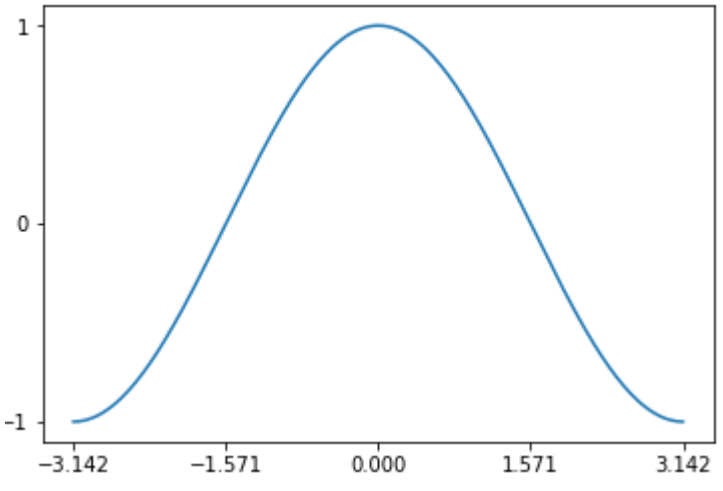
- 축의 범위 설정 (xlim, ylim)

```
plt.plot([1, 4, 9, 16], c="b",
         lw=5, ls="--", marker="o",
         ms=15, mec="g", mew=5, mfc="r")
plt.xlim(-0.2, 3.2)
plt.ylim(-1, 18)
plt.show()
```



- 틱 설정 (축의 마커 설정 / xticks, yticks)

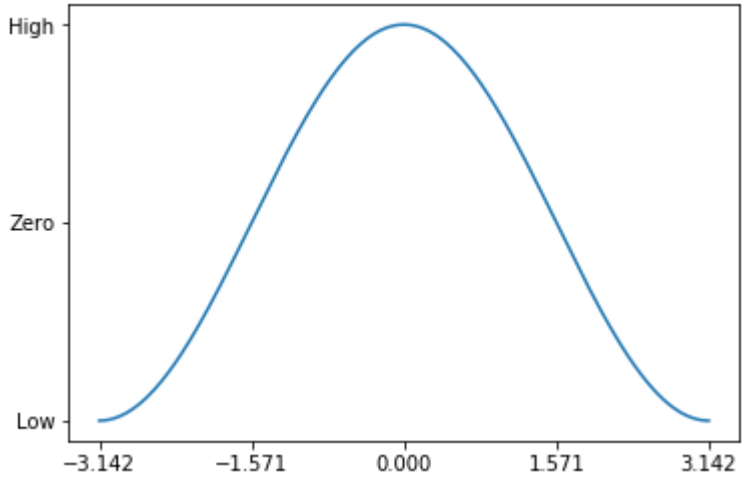
```
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.plot(X, C)
plt.xticks([-np.pi, -np.pi/2, 0,
            np.pi/2, np.pi])
plt.yticks([-1, 0, +1])
plt.show()
```



그래프 설정

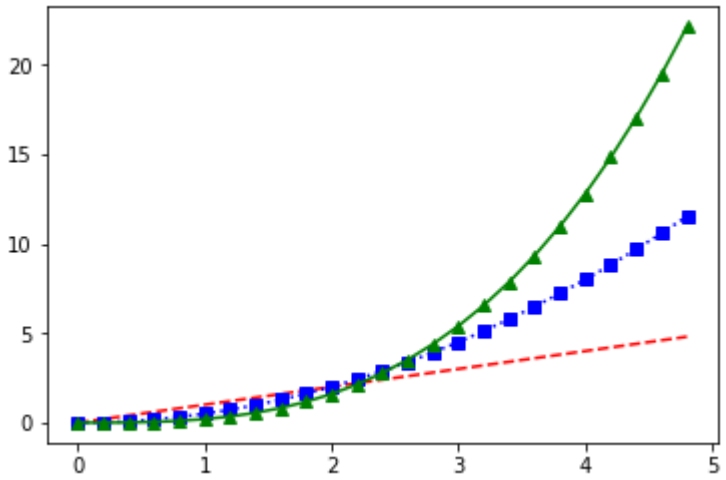
- 그리드 설정 (grid)

```
X = np.linspace(-np.pi, np.pi, 256)
C = np.cos(X)
plt.plot(X, C)
plt.xticks([-np.pi, -np.pi/2, 0,
            np.pi/2, np.pi])
plt.yticks([-1, 0, 1],
            ["Low", "Zero", "High"])
plt.grid(False)
plt.show()
```



- 여러 개의 그래프 1 – 전달인자 반복

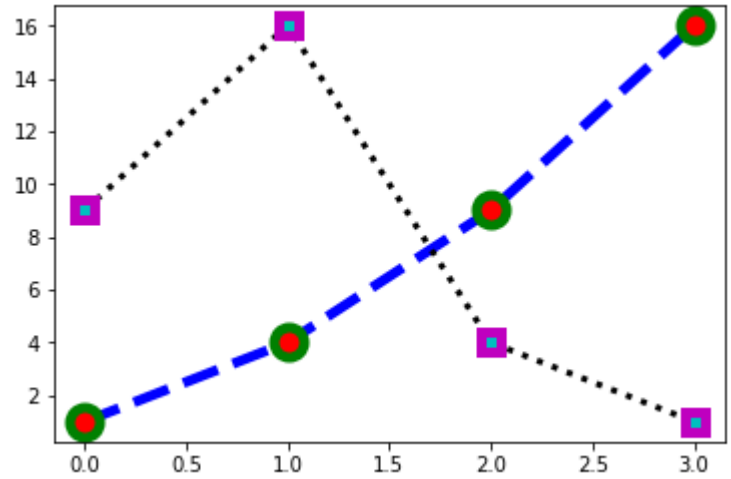
```
t = np.arange(0., 5., 0.2)
plt.plot(t, t, 'r--', t, 0.5*t**2, 'bs:',
         t, 0.2*t**3, 'g^-')
plt.show()
```



그래프 설정

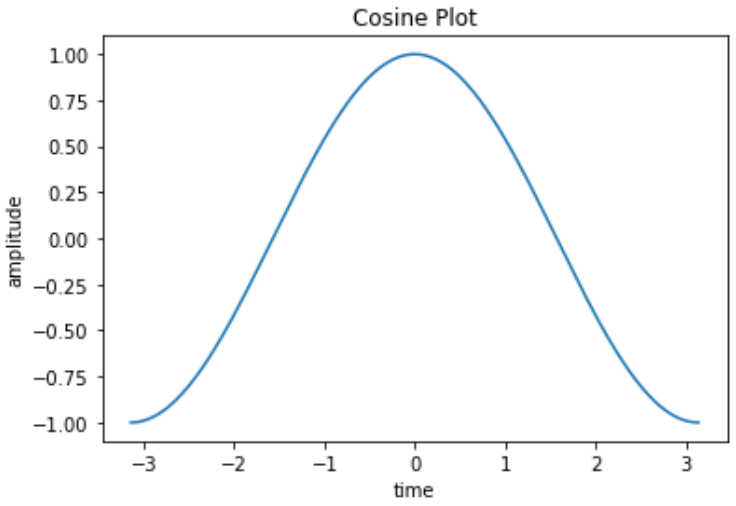
- 여러 개의 그래프 2 – hold 함수를 사용해서 그리기 명령 반복

```
plt.plot([1, 4, 9, 16], c="b", lw=5,
         ls="--", marker="o", ms=15,
         mec="g", mew=5, mfc="r")
# plt.hold(True)   # <- 1,5 버전
plt.plot([9, 16, 4, 1], c="k", lw=3,
         ls=":", marker="s", ms=10,
         mec="m", mew=5, mfc="c")
# plt.hold(False)  # <- 1,5 버전
plt.show()
```



- 축 제목 표시 (title, xlabel, ylabel)

```
X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, label="cosine")
plt.xlabel("time")
plt.ylabel("amplitude")
plt.title("Cosine Plot")
plt.show()
```

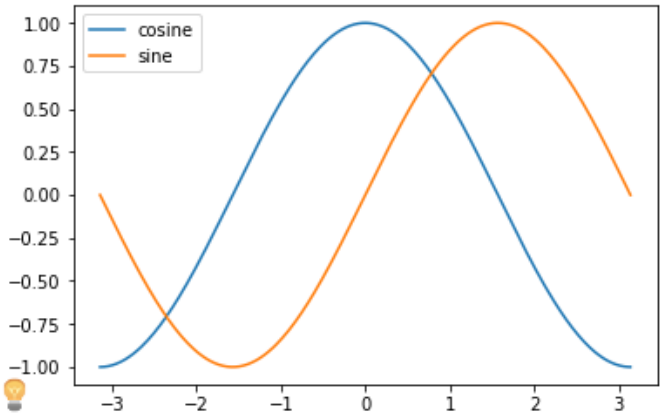


그래프 설정

- 범례
  - » 여러 개의 그래프가 표시될 경우 각 그래프를 설명하기 위한 도구
  - » legend 함수 사용 (loc 전달인자는 범례 표시 위치)

loc 문자열	숫자	loc 문자열	숫자	loc 문자열	숫자
best	0	lower right	4	lower center	8
upper right	1	right	5	upper center	9
upper left	2	center left	6	center	10
lower left	3	center right	7		

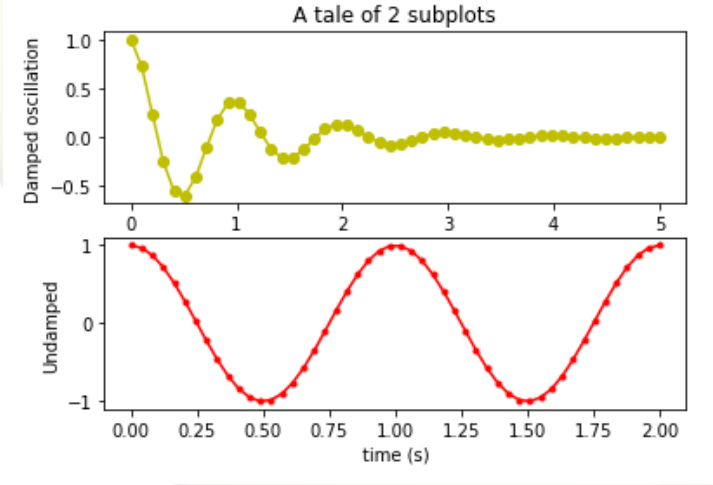
```
X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, label="cosine")
plt.plot(X, S, label="sine")
plt.legend(loc=2)
plt.show()
```



그래프 설정

- subplot을 사용해서 한 화면에 여러 개의 그래프 표시

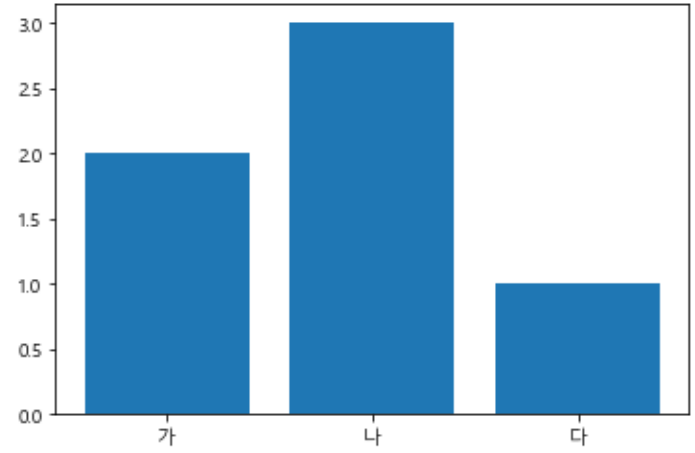
```
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)

ax1 = plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'yo-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')
print(ax1)

ax2 = plt.subplot(2, 1, 2)
plt.plot(x2, y2, 'r.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
print(ax2)

plt.show()
```



그래프 종류

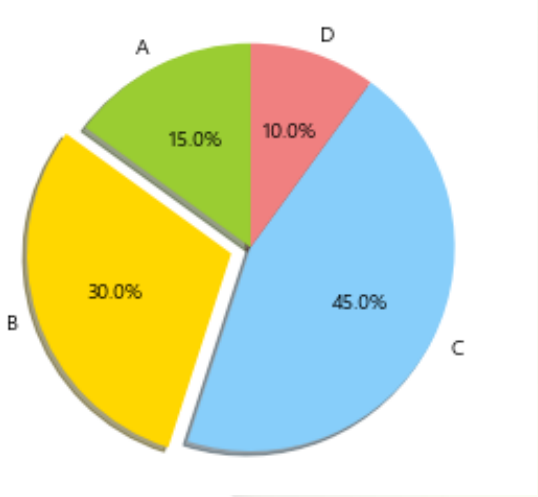
- bar 함수를 사용해서 막대 그래프 출력

```
mpl.rc('font', family='malgun gothic')
y = [2, 3, 1]
x = np.arange(len(y))
xlabel = [u'가', u'나', u'다']
plt.bar(x, y)
plt.xticks(x, xlabel)
plt.show()
```



- pie 함수를 사용해서 파이 그래프 출력

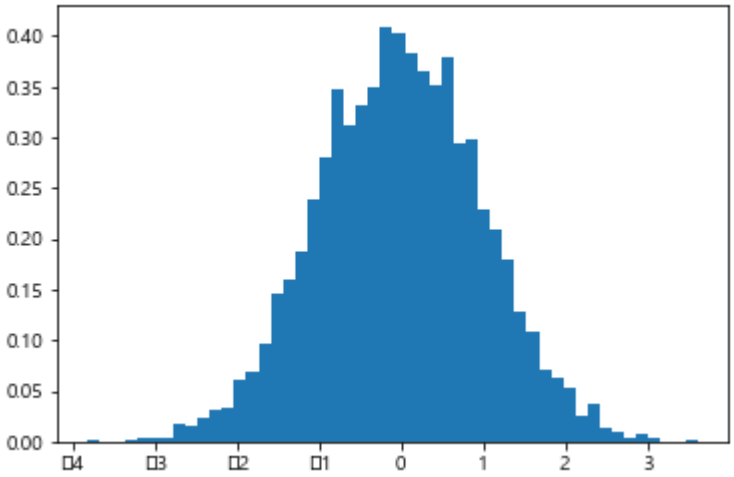
```
labels = 'A', 'B', 'C', 'D'
sizes = [15, 30, 45, 10]
colors = ['yellowgreen', 'gold',
'lightskyblue', 'lightcoral']
explode = (0, 0.1, 0, 0)
plt.pie(sizes, explode=explode, labels=labels,
        colors=colors, autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.axis('equal')
plt.show()
```



그래프 종류

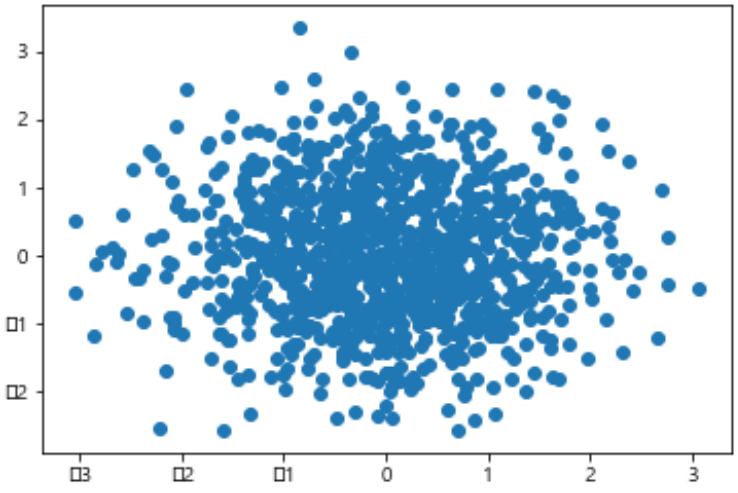
- hist 함수를 사용해서 히스토그램 출력

```
x = np.random.randn(5000)
arrays, bins, patches = plt.hist(x,
bins=50, normed=True)
plt.show()
```



- scatter 함수를 사용해서 산점도 그래프 출력

```
X = np.random.normal(0,1,1024)
Y = np.random.normal(0,1,1024)
plt.scatter(X,Y)
plt.show()
```



seaborn

seaborn

- matplotlib을 기반으로 다양한 색상 테마와 통계용 차트 등의 기능을 추가한 시각화 패키지.
  - » 기본적인 시각화 기능은 matplotlib 패키지에 의존
  - » 통계 기능은 statsmodels 패키지에 의존

- 설치
  - » 표준 파이썬 배포판에 seaborn 모듈이 포함되지 않아서 별도 설치 필요

```
> conda install seaborn
```

- 사용
  - » seaborn을 사용하기 전에 import

```
import seaborn
```

```
import seaborn as sns
```

-

색상 팔레트

- seaborn의 색상 스타일 지정을 위한 도구

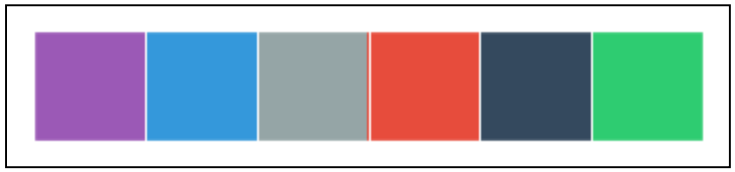
- matplotlib의 컬러맵(colormap)으로 사용 가능.

```
current_palette = sns.color_palette()
sns.palplot(current_palette)
```

```
sns.palplot(sns.color_palette("Blues"))
```

색상 팔레트

```
flatui = ["#9b59b6", "#3498db", "#95a5a6", "#e74c3c", "#34495e", "#2ecc71"]
sns.palplot(sns.color_palette(flatui))
```



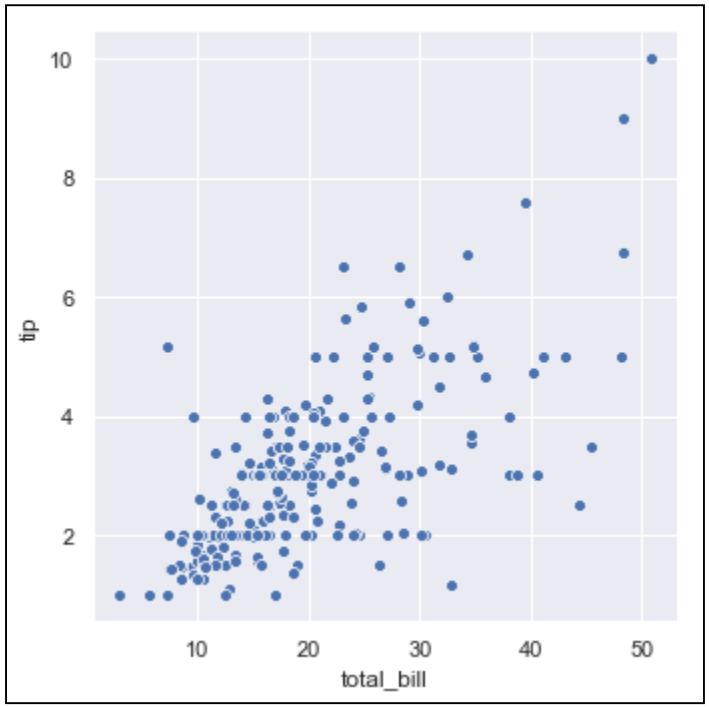
```
colors = ["windows blue", "amber", "greyish", "faded green", "dusty purple"]
sns.palplot(sns.xkcd_palette(colors))
```

» xkcd 색상명은 https://xkcd.com/color/rgb/에서 참고

통계적 관계 표현

- 산점도 그래프

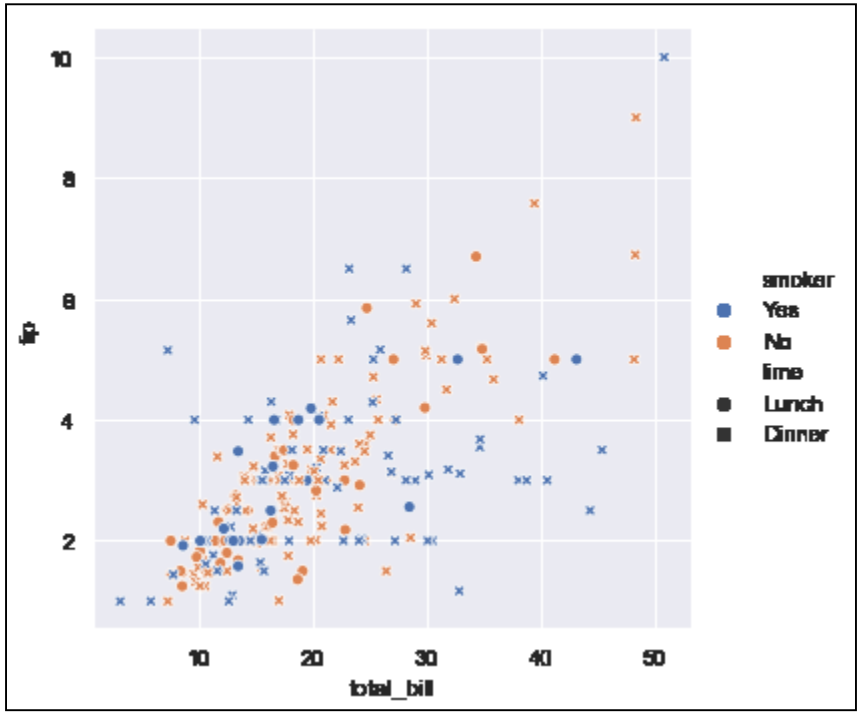
```
# 두 변수 사이의 관계 표현
tips = sns.load_dataset("tips")
sns.relplot(x="total_bill", y="tip", data=tips)
```



통계적 관계 표현

- 산점도 그래프

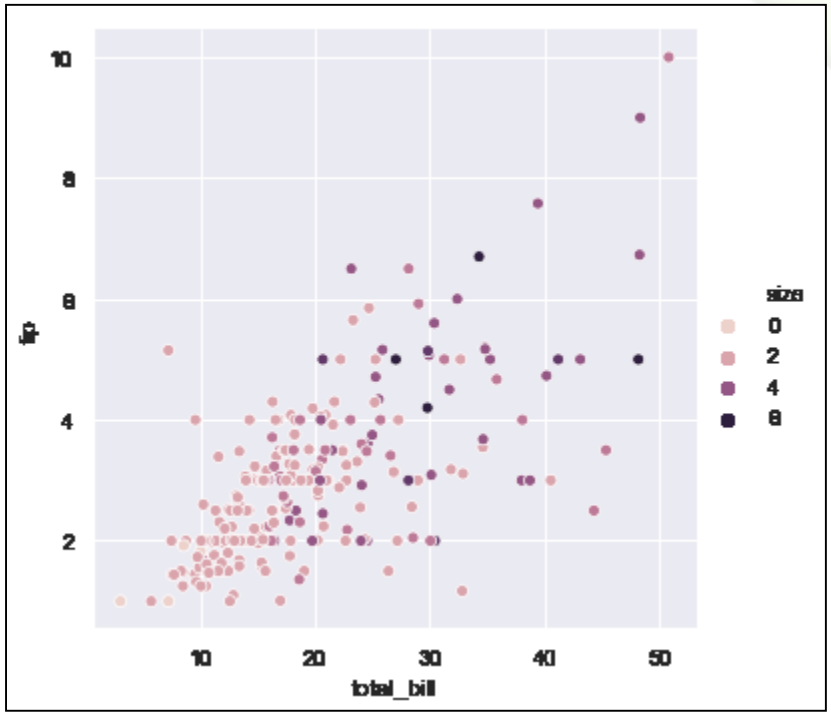
```
# 범주형 변수와 시각화 특성 바인딩
sns.relplot(x="total_bill", y="tip", hue="smoker", style="time", data=tips);
#sns.scatterplot(x="total_bill", y="tip", hue="smoker", style="time",
data=tips);
```

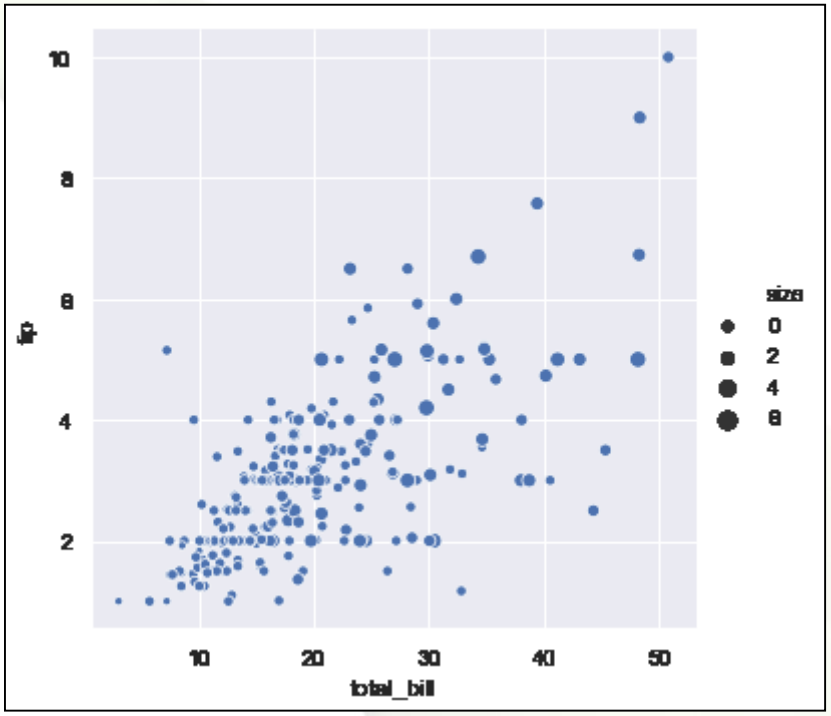


통계적 관계 표현

- 산점도 그래프

```
# 연속형 변수와 시각화 특성 바인딩
sns.relplot(x="total_bill", y="tip", hue="size", data=tips);
```



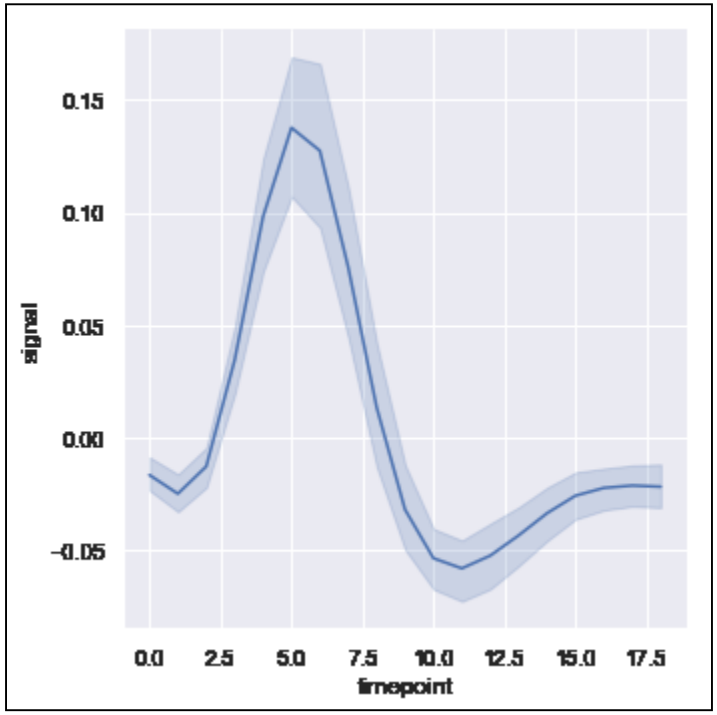



```
# 연속형 변수와 시각화 특성 바인딩
sns.relplot(x="total_bill", y="tip", size="size", data=tips);
```

통계적 관계 표현

- 선 그래프

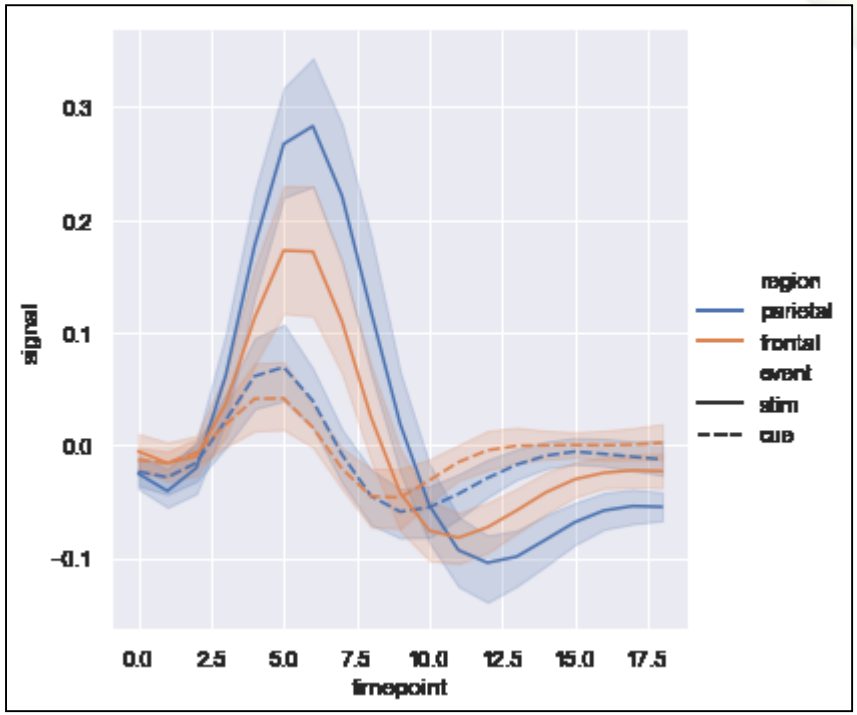
```
# 두 변수 사이의 관계
# 불확실성 표시
fmri = sns.load_dataset("fmri")
sns.relplot(x="timepoint", y="signal", kind="line", data=fmri);
```



통계적 관계 표현

- 선 그래프

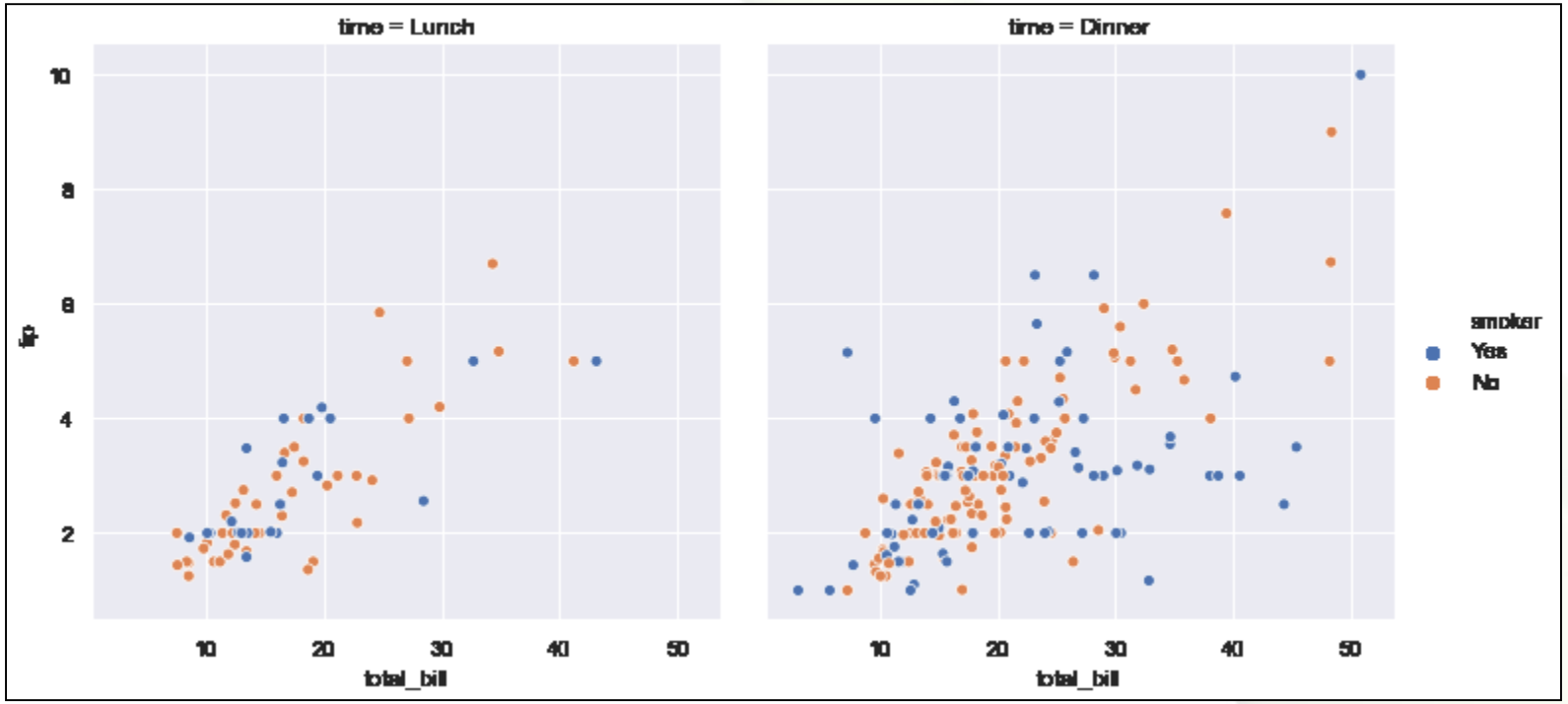
```
# 데이터와 시각화 특성 바인딩
sns.relplot(x="timepoint", y="signal", hue="region", style="event",
kind="line", data=fmri);
```



통계적 관계 표현

- 다중 관계 표현

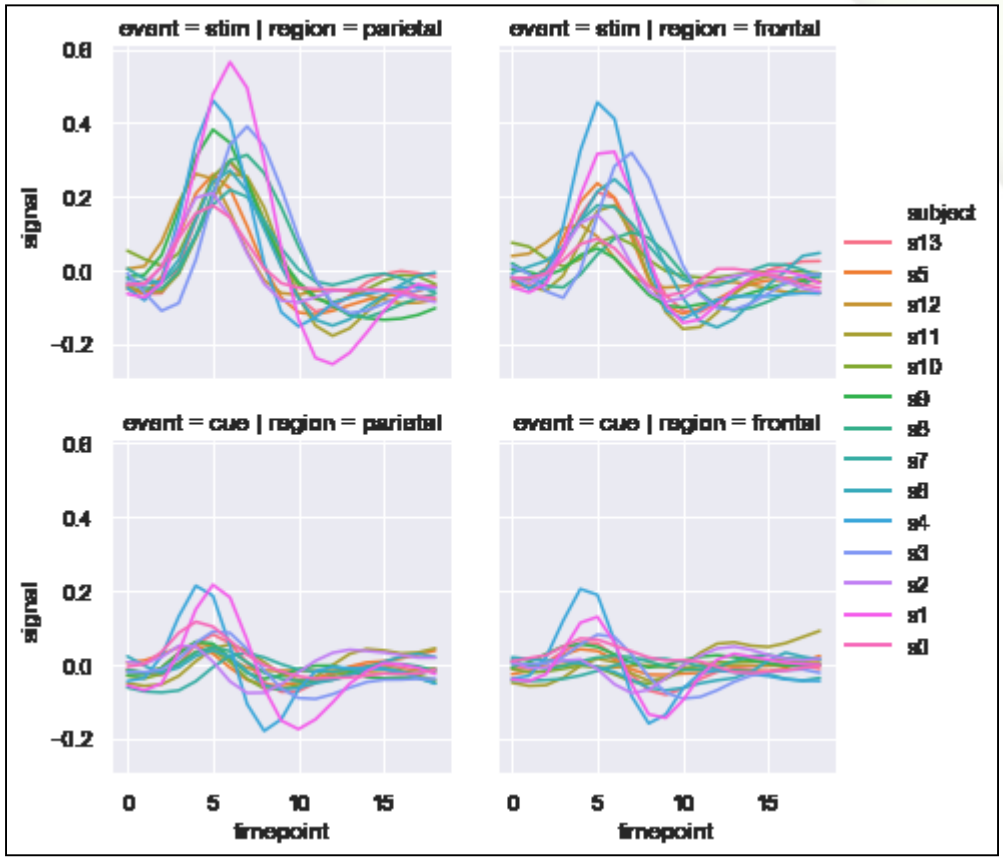
```
# 컬럼에 비교할 분류 바인딩
sns.relplot(x="total_bill", y="tip", hue="smoker", col="time", data=tips);
```



통계적 관계 표현

- 다중 관계 표현

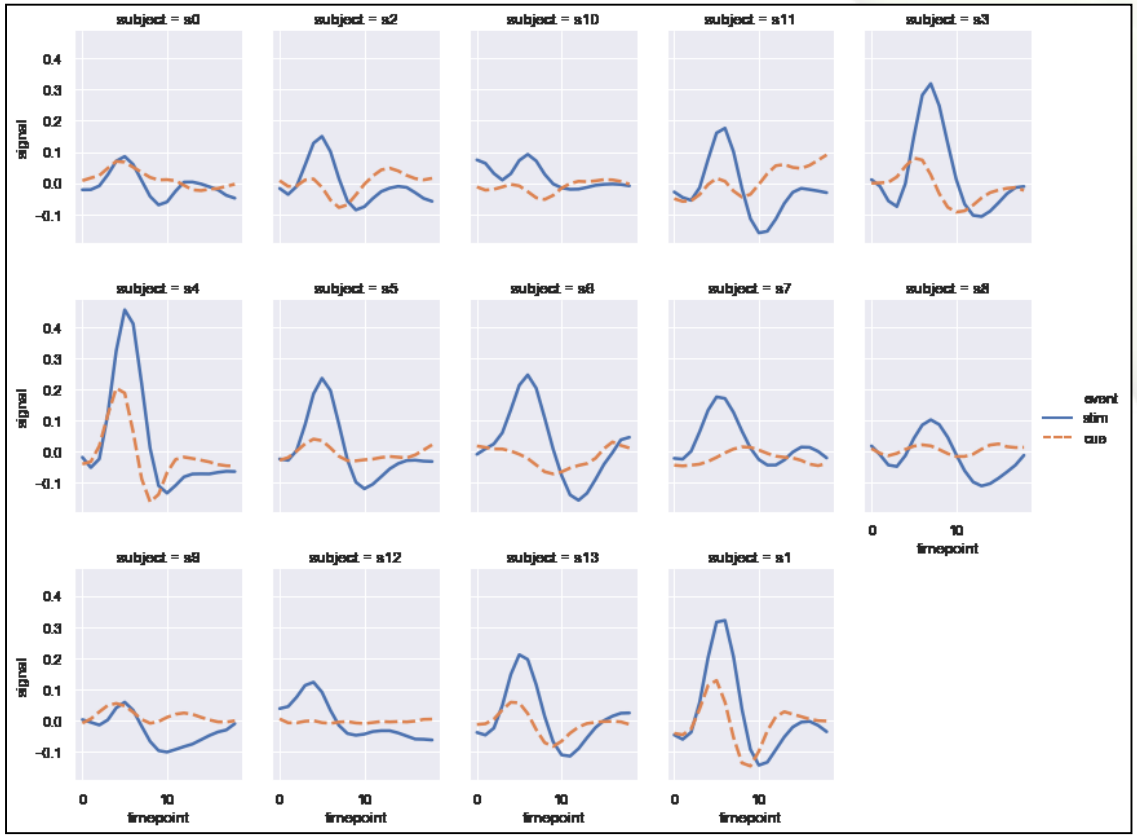
```
# 행과 열에 비교할 분류 바인딩
sns.relplot(x="timepoint", y="signal", hue="subject",
            col="region", row="event", height=3,
            kind="line", estimator=None, data=fmri);
```



통계적 관계 표현

- 다중 관계 표현

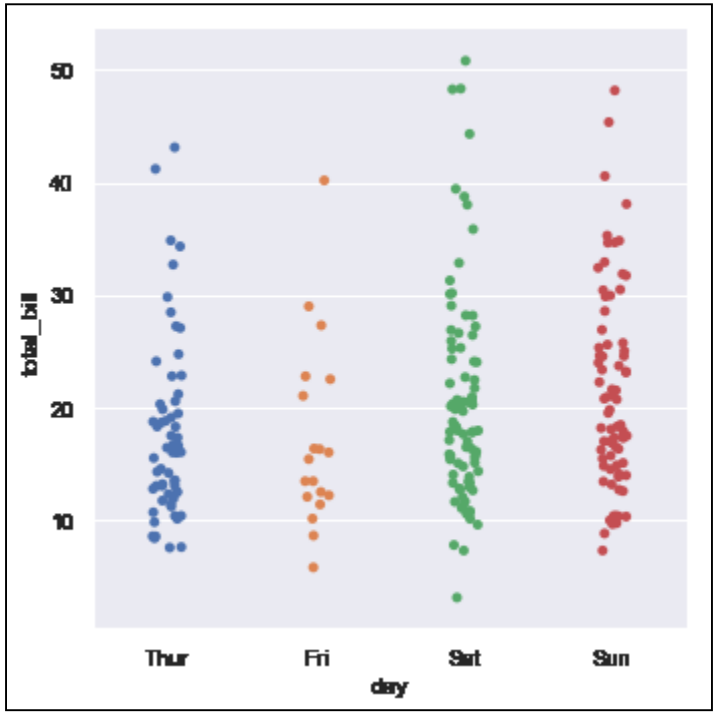
```
# 비교할 분류가 많은 경우 한 행에 표시할 최대 플롯 개수 지정 : wrap
sns.relplot(x="timepoint", y="signal", hue="event", style="event",
            col="subject", col_wrap=5, height=3, linewidth=2.5,
            kind="line", data=fmri.query("region == 'frontal'"));
```



범주형 데이터 시각화

- 범주 데이터 산점도 그래프

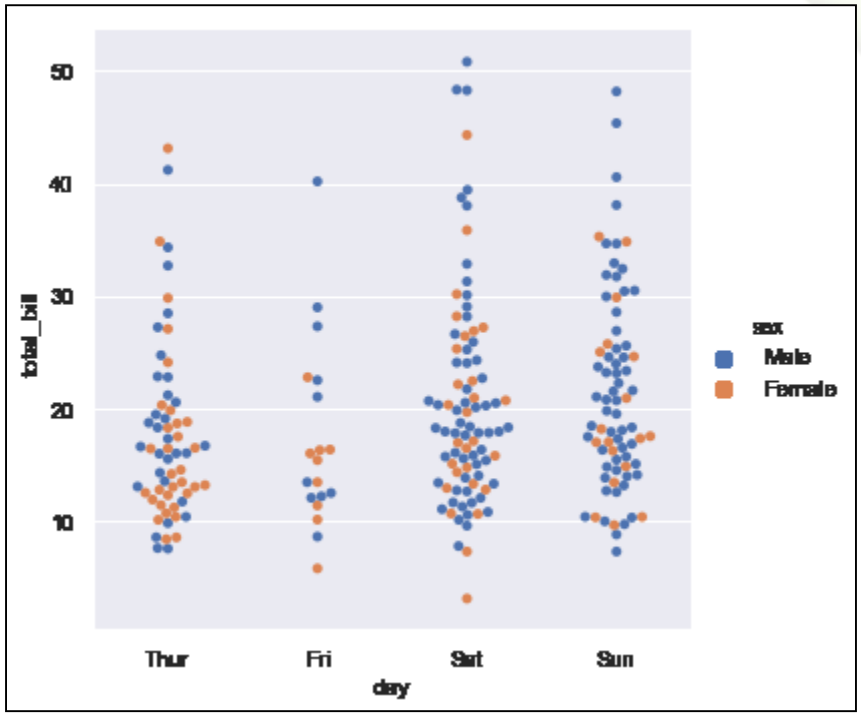
```
# 기본 범주 데이터 산점도 그래프
tips = sns.load_dataset("tips")
sns.catplot(x="day", y="total_bill", data=tips, jitter=True); # jitter :
easing overplotting
```



범주형 데이터 시각화

- 범주 데이터 산점도 그래프

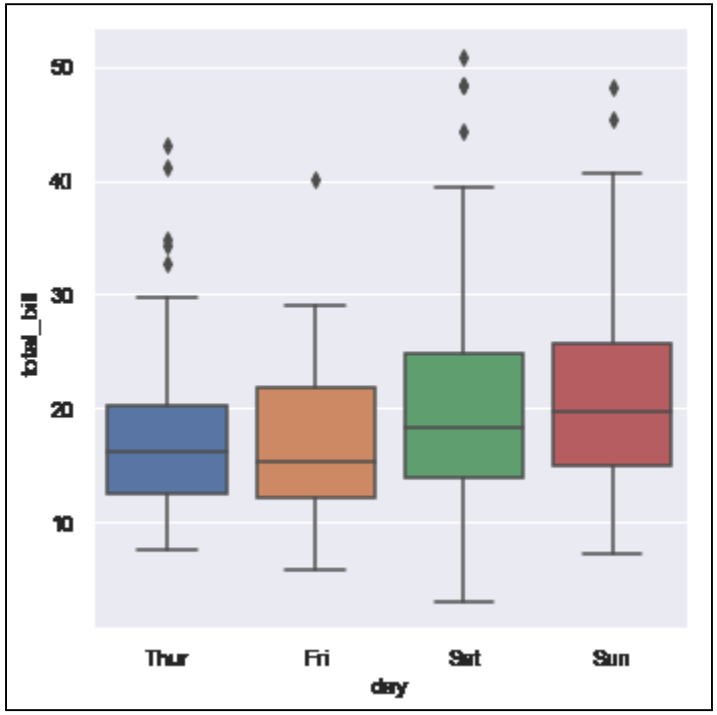
```
# beeswarm형 그래프
sns.catplot(x="day", y="total_bill", hue="sex", kind="swarm", data=tips); #
kind="swarm" : easing overplotting
```



범주형 데이터 시각화

- 박스 플롯

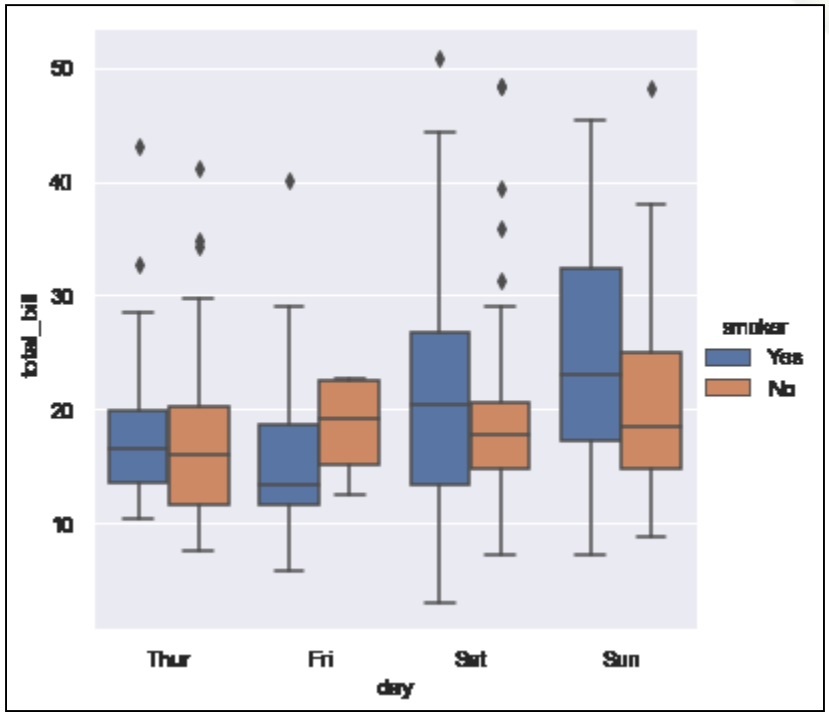
```
# 기본 박스 플롯
sns.catplot(x="day", y="total_bill", kind="box", data=tips);
```



범주형 데이터 시각화

- 박스 플롯

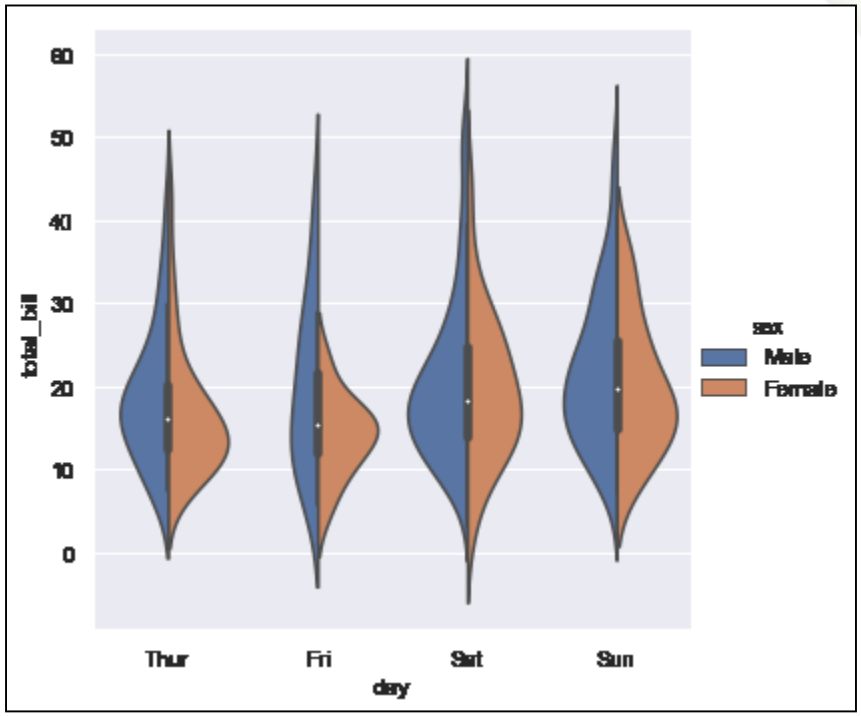
```
# 데이터와 시각화 요소 바인딩
sns.catplot(x="day", y="total_bill", hue="smoker", kind="box", data=tips);
```



범주형 데이터 시각화

- 바이올린 플롯

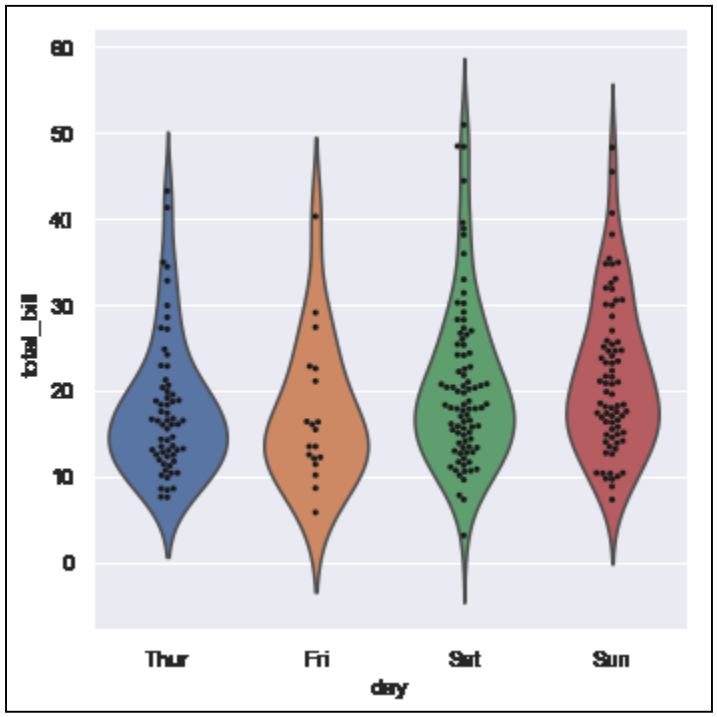
```
# 단일 바이올린 플롯
sns.catplot(x="day", y="total_bill", hue="sex", kind="violin", split=True,
data=tips);
```



범주형 데이터 시각화

- 바이올린 플롯

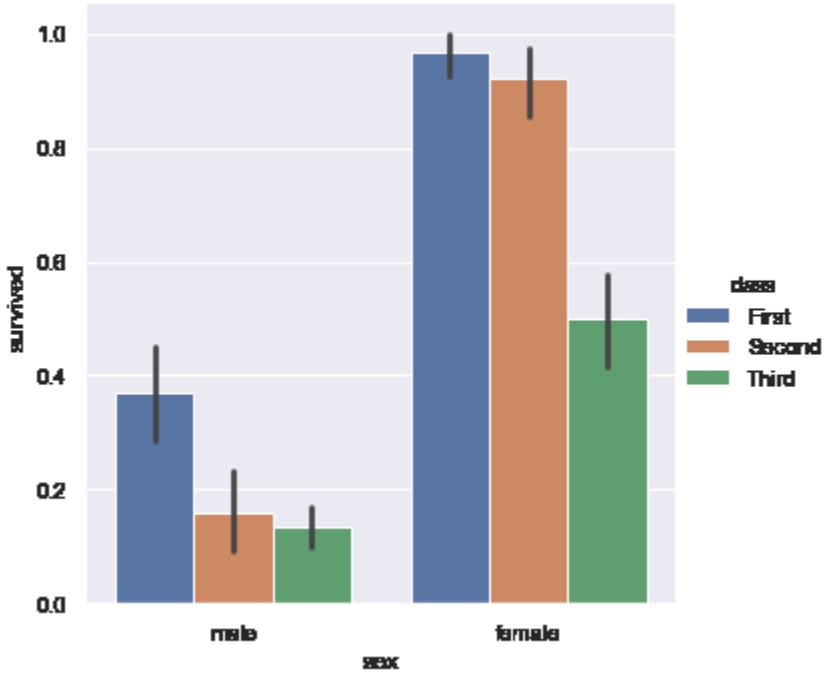
```
# 바이올린플롯과 범주데이터 산점도 그래프 병합
g = sns.catplot(x="day", y="total_bill", kind="violin", inner=None,
data=tips)
sns.swarmplot(x="day", y="total_bill", color="k", size=3, data=tips,
ax=g.ax);
```



범주형 데이터 시각화

- 범주별 통계적 추정

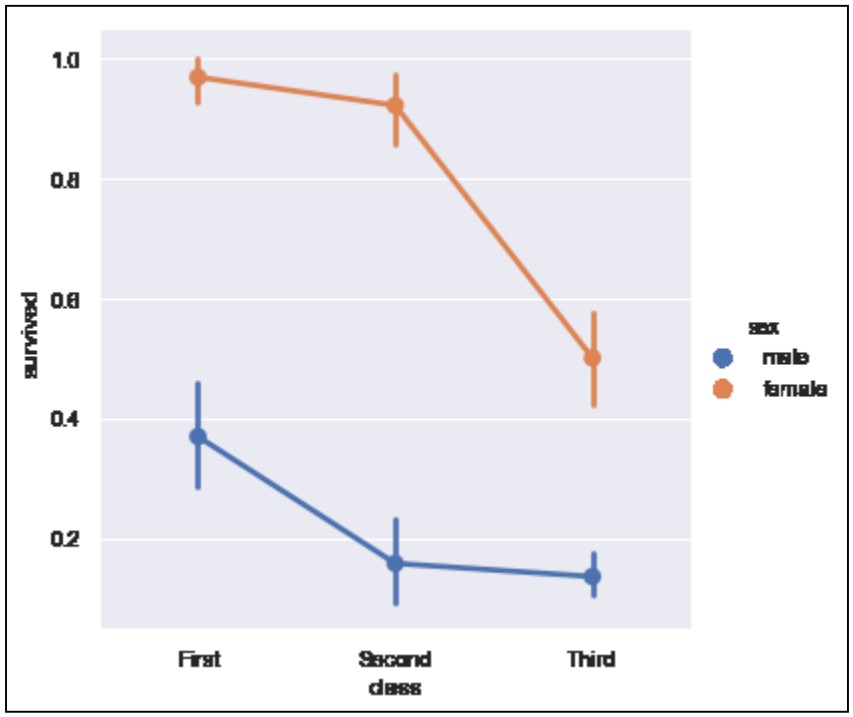
```
#Bar plot
titanic = sns.load_dataset("titanic")
sns.catplot(x="sex", y="survived", hue="class", kind="bar", data=titanic);
```



범주형 데이터 시각화

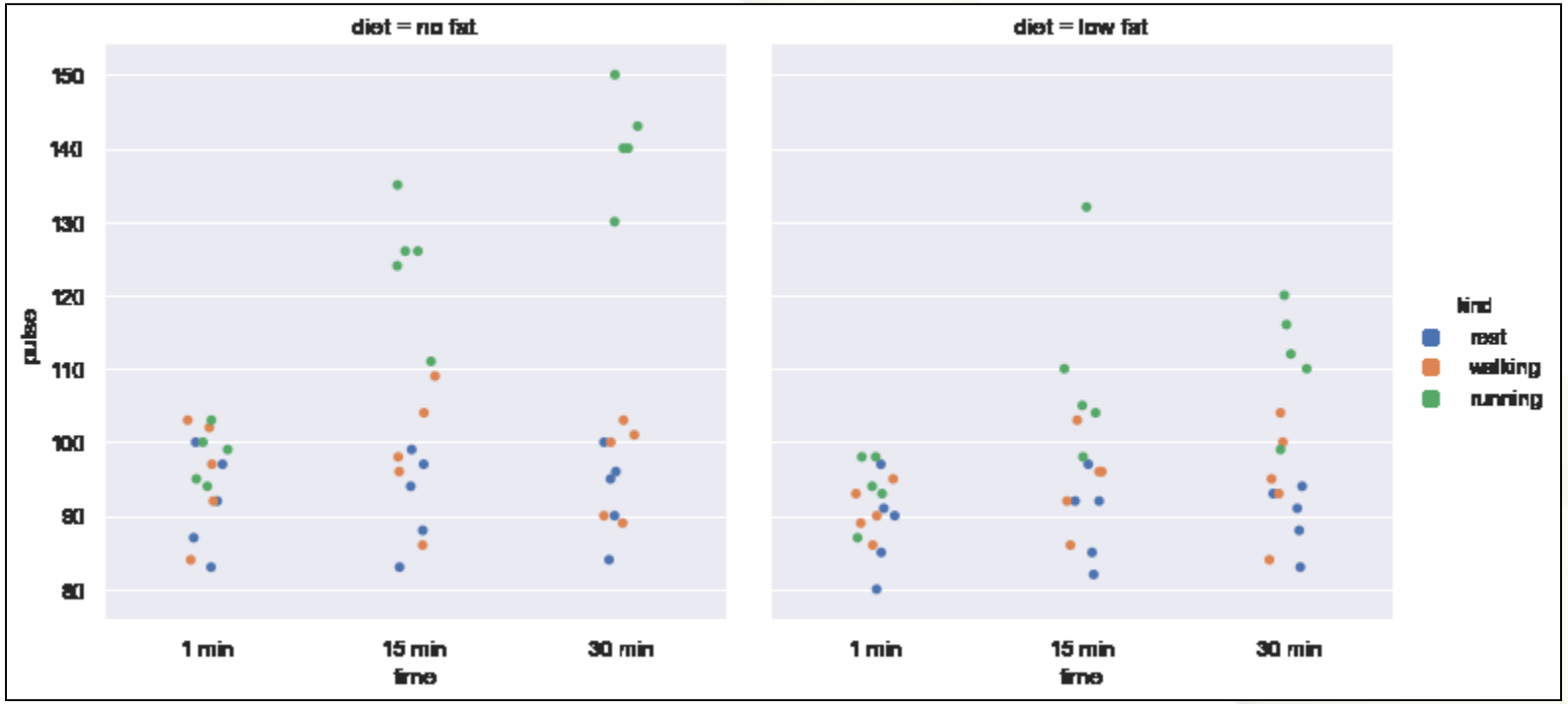
- 범주별 통계적 추정

```
# Point plots
sns.catplot(x="class", y="survived", hue="sex",
            #palette={"male": "g", "female": "m"},
            #markers=["^", "o"], linestyles=["-", "--"],
            kind="point", data=titanic);
```



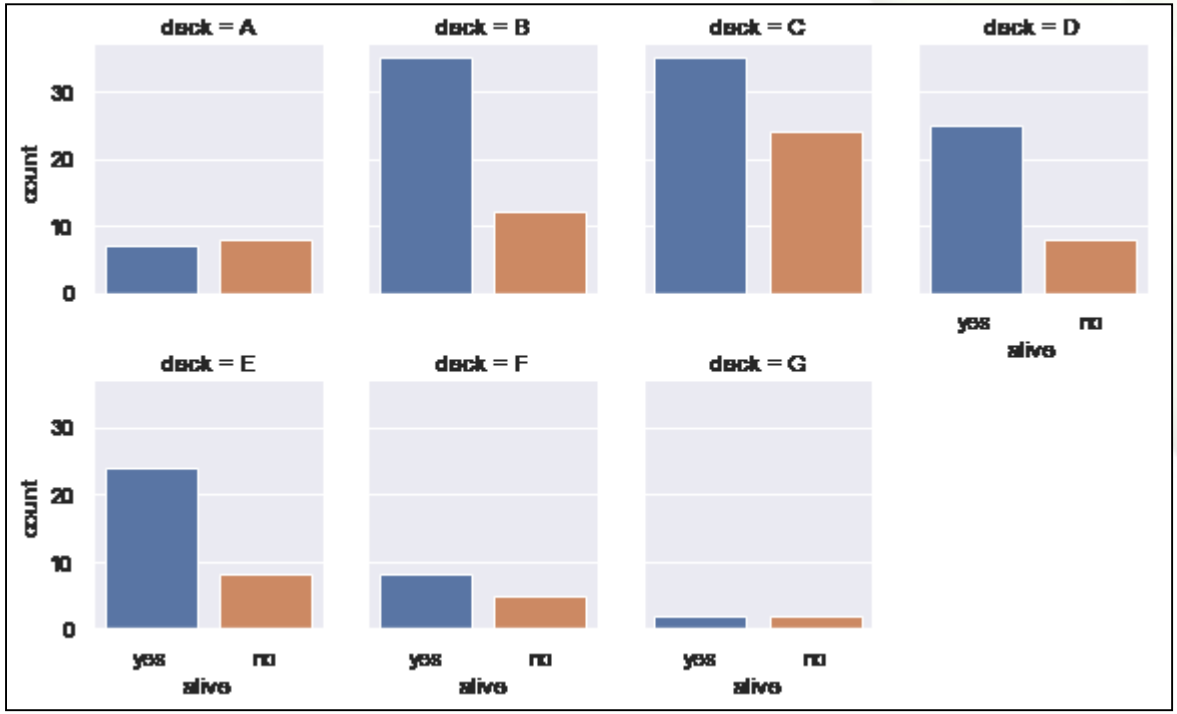
다중 관계 표현

```
exercise = sns.load_dataset("exercise")
sns.catplot(x="time", y="pulse", hue="kind", col="diet", data=exercise)
```



다중 관계 표현

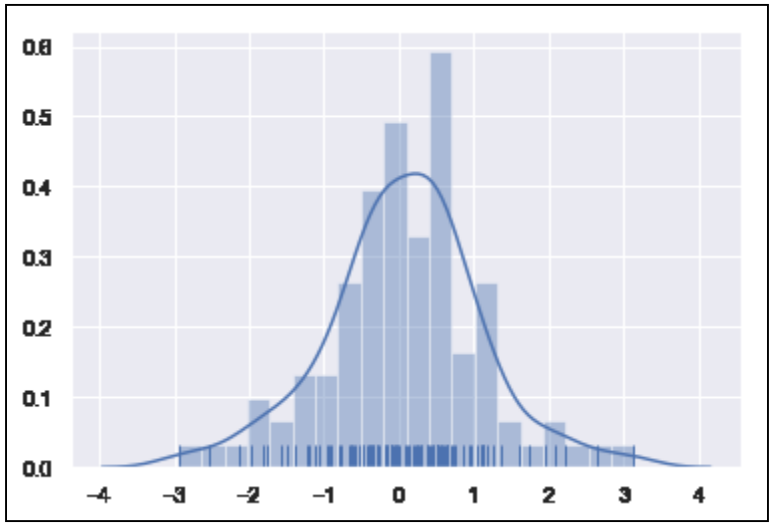
```
titanic = sns.load_dataset("titanic")
g = sns.catplot(x="alive", col="deck", col_wrap=4,
                data=titanic[titanic.deck.notnull()],
                kind="count", height=2.5, aspect=.8, )
```



데이터 분포 시각화

- 일변량 분포 시각화

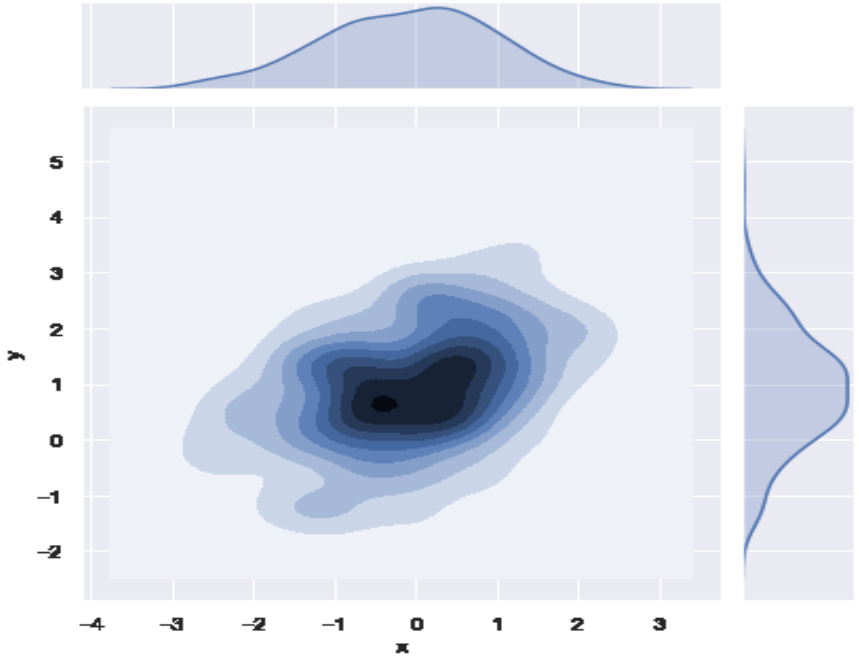
```
# 일변량 분포 시각화
x = np.random.normal(size=100)
sns.distplot(x, bins=20, kde=True, rug=True); # kde : kernel density
estimation
```



데이터 분포 시각화

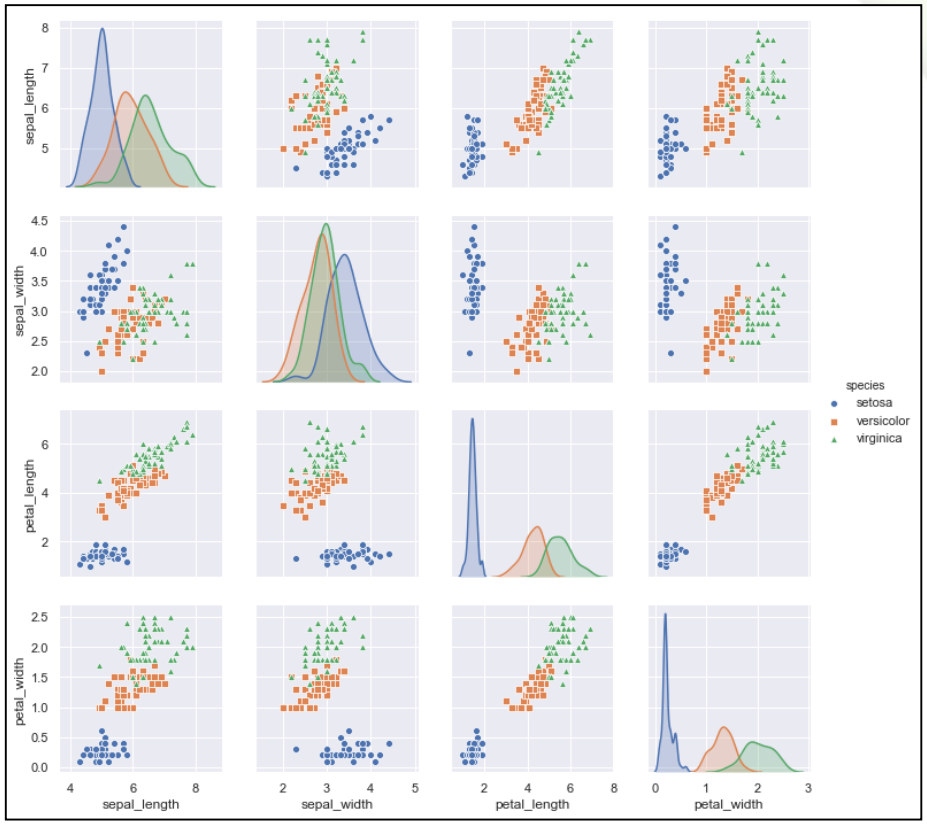
- 이변량 분포 시각화

```
# 이변량 분포 시각화
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
sns.jointplot(x="x", y="y", kind="kde", data=df); # kind : (scatter,hex,kde)
```



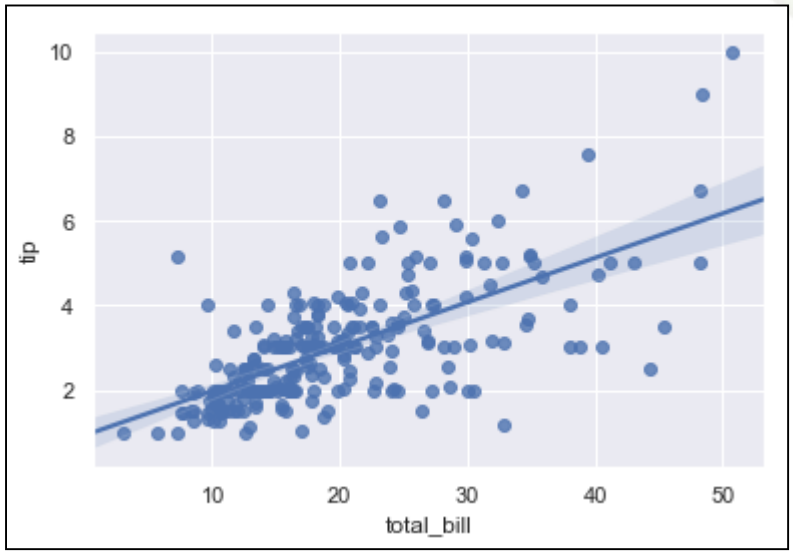
짝지은 관계 시각화

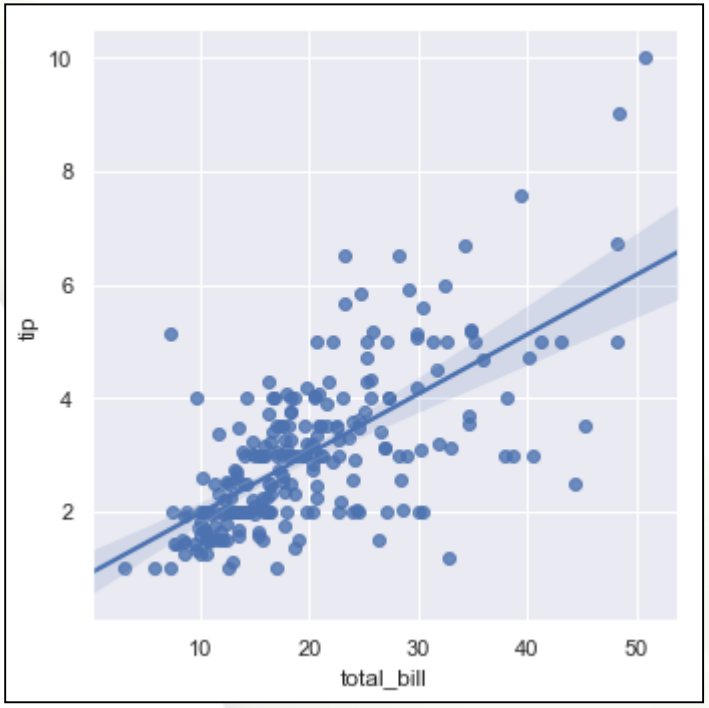
```
iris = sns.load_dataset("iris")
sns.pairplot(iris, hue="species", markers=["o", "s", "^"]) # markers는 hue를 지정했을 경우에만 사용 가능하며 hue에 지정된 컬럼의 데이터 범주의 개수와 일치
plt.show()
```



선형 관계 시각화

```
# 선형 관계 시각화
sns.regplot(x="total_bill", y="tip", data=tips);
```

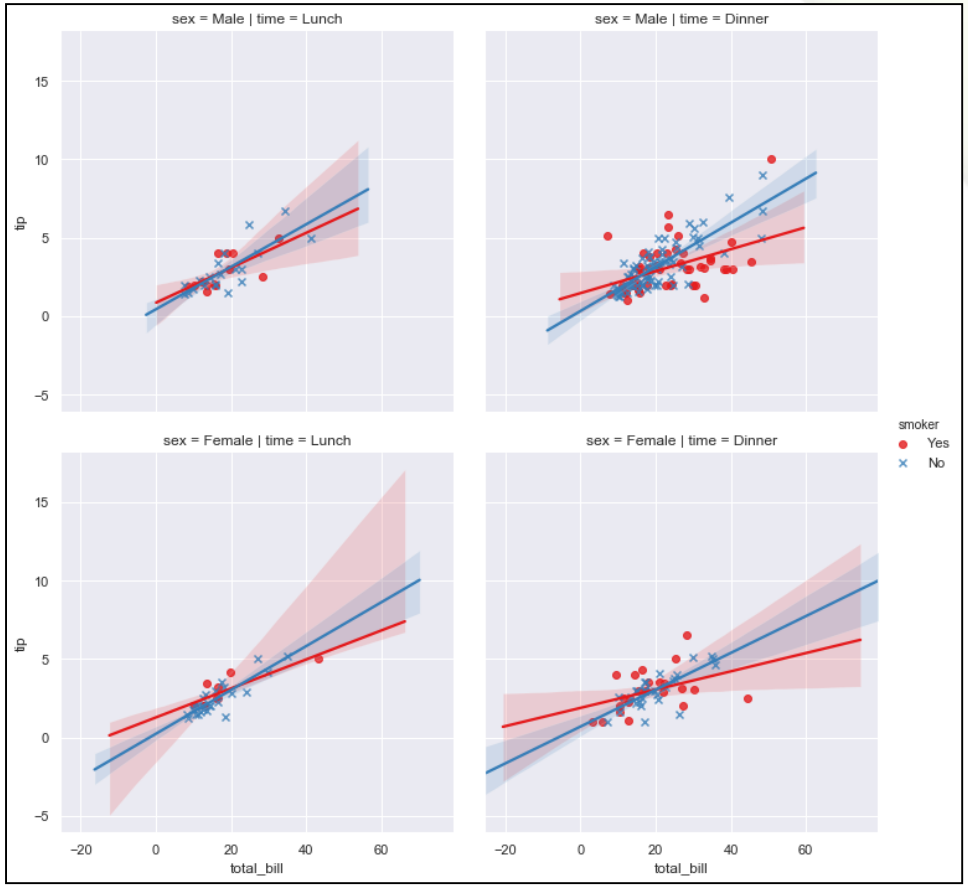





```
# 선형 관계 시각화
sns.lmplot(x="total_bill", y="tip", data=tips);
```

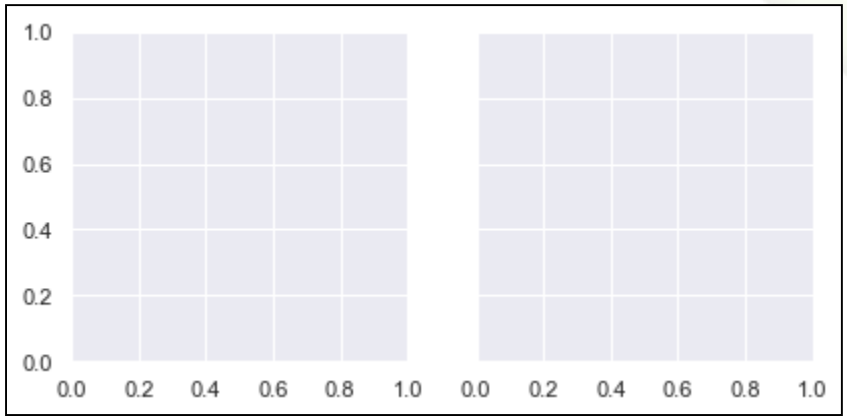
선형 관계 시각화

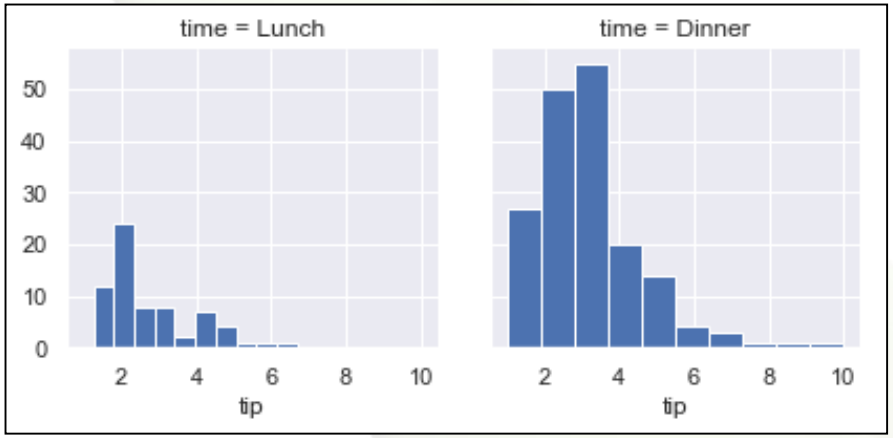
```
# 데이터와 시각화 속성 매핑
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips, markers=["o",
"x"], palette="Set1", col="time", row="sex");
```



다중 그래프 시각화

```
tips = sns.load_dataset("tips")
g = sns.FacetGrid(tips, col="time")
```

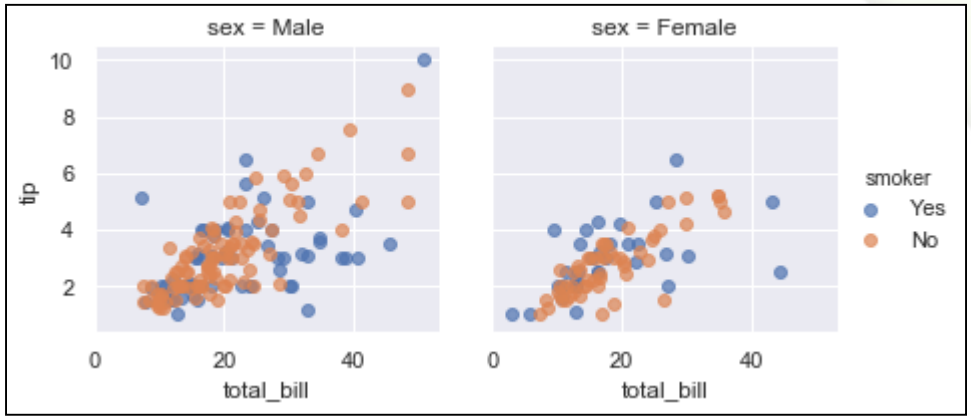





```
g = sns.FacetGrid(tips, col="time")
g.map(plt.hist, "tip");
```

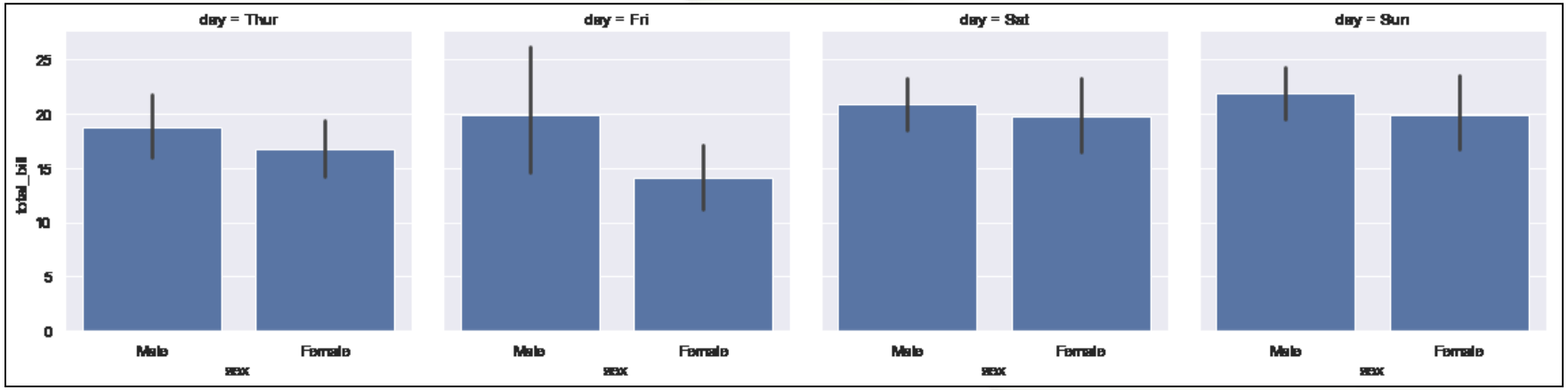
다중 그래프 시각화

```
# 데이터와 시각화 속성 매핑
g = sns.FacetGrid(tips, col="sex", hue="smoker")
g.map(plt.scatter, "total_bill", "tip", alpha=.7)
g.add_legend()
```



다중 그래프 시각화

```
g = sns.FacetGrid(tips, col="day", height=4)
g.map(sns.barplot, "sex", "total_bill");
```



다중 그래프 시각화

```
g = sns.PairGrid(iris, hue="species")
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
g.add_legend();
```