```
table.dataframe th {
  text-align: center;
  font-weight: bold;
  padding: 8px;
}

table.dataframe td {
  text-align: center;
  padding: 8px;
}

table.dataframe tr:hover {
  background: #b8d1f3;
}

.output_prompt {
  overflow: auto;
  font-size: 0.9rem;
  line-height: 1.45;
  border-radius: 0.3rem;
  -webkit-overflow-scrolling: touch;
  padding: 0.8rem;
  margin-top: 0;
  margin-bottom: 15px;
  font: 1rem Consolas, "Liberation Mono", Menlo, Courier, monospace;
  color: $code-text-color;
  border: solid 1px $border-color;
  border-radius: 0.3rem;
  word-break: normal;
  white-space: pre;
}
```

.dataframe tbody tr th:only-of-type { vertical-align: middle; }

.dataframe tbody tr th { vertical-align: top; }

.dataframe thead th { text-align: center !important; padding: 8px; }

.page__content p { margin: 0 0 0px !important; }

.page__content p > strong { font-size: 0.8rem !important; }

Part 1 | 파이썬 기초

1. 파이썬 첫 걸음

출력하기

기본 출력

- print(value1, value2, ...,)

```
# hello 출력하기
print('hello')
print("hello")
```

```
hello
hello
```

```
# hello, python 출력하기
print('hello','python')
```

```
hello python
```

```
# 계산식 출력하기 (1+1=2)
print('1+1=',1+1)
```

```
1+1= 2
```

구분자 지정

- print(value1, value2, ..., sep='구분자')

```
# ,로 구분 (hello,python)
print('hello','python',sep=',')
```

```
hello,python
```

```
# 공백없이 붙여서 출력 (1+1=2)
print('1+1=',1+1,sep='')
```

```
1+1=2
```

끝문자 지정

- print(value1, value2, ..., end='끝문자')

```
# 공백으로 끝내기 (안녕하세요 반갑습니다)
print('안녕하세요', end=' ')
print('반갑습니다')
```

```
안녕하세요 반갑습니다
```

```
# !!로 끝내기(안녕하세요!!반갑습니다!!)
print('안녕하세요', end='!!')
print('반갑습니다', end='!!')
```

```
안녕하세요!!반갑습니다!!
```

여러 줄 한번에 출력하기

- 홑따옴표 세개(''') 혹은 쌍따옴표 세개(""")로 묶는다.

```
print('자세히 보아야 예쁘다.')
print('오래 보아야 사랑스럽다.')
print('너도 그렇다.')
```

```
자세히 보아야 예쁘다.
오래 보아야 사랑스럽다.
너도 그렇다.
```

```
print('''
자세히 보아야 예쁘다.
오래 보아야 사랑스럽다.
너도 그렇다.
''')
```

```
자세히 보아야 예쁘다.
오래 보아야 사랑스럽다.
너도 그렇다.
```

```
print("""
자세히 보아야 예쁘다.
오래 보아야 사랑스럽다.
너도 그렇다.
""")
```

```
자세히 보아야 예쁘다.
오래 보아야 사랑스럽다.
너도 그렇다.
```

주석달기

- 프로그램에 대한 설명을 적을 때, 코드의 실행을 잠시 막아둘 때 주석을 사용한다.
- 파이썬에서는 샵('#') 을 주석 기호로 사용한다.

```
## 주석에 대한 코드입니다##
print(1) # 주석
#print(2)
print(3)
```

```
1
3
```

들여쓰기

- 파이썬에서는 들여쓰기 자체가 문법입니다.
- 파이썬 들여쓰기 방법은 공백2칸, 공백4칸 등 여러가지 방법이 있지만 보통 공백 4칸을 사용합니다.

```
print('hello1')
 print('hello2')
```

오류 해석하기

파이썬에서는 대소문자를 구분한다.

```
Print('hello')
```

따옴표의 짝이 맞아야 한다.

```
print('hello'')
```

괄호의 짝이 맞아야 한다.

```
print('hello'}
```

2. 기본 자료형 다루기

변수

변수에 값 할당하기

- 변수명 = 변수값
- 변수에 값이 할당될 때 변수값의 자료형에 따라 변수의 자료형이 결정된다.

```
name = 'james'
age = 20
height = 175.0
ischild = age<20
```

변수 값, 자료형 출력하기

- 변수에 접근할 때는 변수명을 사용한다.
- 변수값 출력 : print(변수명)
- 변수자료형 출력 : print(type(변수명))

```
print(name)
print(type(name))
```

```
james
```

```
print(age)
print(type(age))
```

```
20
```

```
print(height)
print(type(height))
```

```
175.0
```

```
print(ischild)
print(type(ischild))
```

```
False
```

기본 자료형 변환하기

- int(value) : int형으로 변환

```
age = int('20')
type(age)
```

```
int
```

- str(value) : str형으로 변환

```
age = str(20)
type(age)
```

```
str
```

- float(value) : float형으로 변환

```
age = float(20)
print(age, type(age))
```

```
20.0
```

- input으로 입력받은 값을 숫자형으로 변환하여 사용하기

```
a = int(input('나이:'))
```

```
나이:20
```

```
type(a)
```

```
int
```

Pandas 자료형 변환

자료형 확인

- **데이터프레임.dtyes**
- **시리즈.dtype**
- 한 시리즈에 문자열과 숫자, 문자열과 부울 등으로 데이터타입이 혼합되어 있으면 object형으로 결정된다.
- 한 시리즈에 정수와 실수가 혼합되어 있으면 float64으로 결정된다.

```
# 샘플데이터
df = pd.DataFrame({'float': [1.0, 2.0],
                   'int': [1,2],
                   'datetime':
[pd.Timestamp('20200101'),pd.Timestamp('20210101')],
                   'string': ['a','b'],
                   'bool':[True,False],
                   'object':[1,'-'],
                   'float2' : [1.0, 2]})
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	float	int	datetime	string	bool	object	float2
0	1.0	1	2020-01-01	a	True	1	1.0
1	2.0	2	2021-01-01	b	False	-	2.0

데이터프레임의 자료형 확인

```
# 데이터프레임의 자료형 확인
df.dtypes
```

```
float              float64
int                  int64
datetime    datetime64[ns]
string              object
bool                  bool
```

```
object               object
float2              float64
dtype: object
```

시리즈의 자료형 확인

```
# int 컬럼의 자료형 확인
df['int'].dtype
```

```
dtype('int64')
```

자료형이 혼합된 컬럼의 자료형 확인

```
# 숫자형과 문자형이 혼합되어 있는 경우 각 데이터의 자료형 확인
type(df.loc[0,'object'])
```

```
int
```

```
print(df.loc[1,'object'])
print(type(df.loc[1,'object']))
```

```

```

<class 'str'>

자료형 변환

- 데이터프레임.**astype**('자료형')
- 시리즈.**astype**('자료형')

```
# 샘플 데이터
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	col1	col2
0	1	3
1	2	4

```
df.dtypes
```

```
col1    int64
col2    int64
dtype: object
```

데이터프레임 전체 자료형 변환

```
# 실수형으로 변환
df = df.astype('float64')
df.dtypes
```

```
col1    float64
col2    float64
dtype: object
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	col1	col2
0	1.0	3.0
1	2.0	4.0

```
# 문자열로 변환
df = df.astype('str')
df.dtypes
```

```
col1    object
col2    object
dtype: object
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	col1	col2

	col1	col2
0	1.0	3.0
1	2.0	4.0

```
# 정수형으로 변환
df = df.astype('float').astype('int')
df.dtypes
```

```
col1    int32
col2    int32
dtype: object
```

컬럼의 자료형 변환

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	col1	col2
0	1	3
1	2	4

```
# col2의 자료형만 str로 변환
df['col2'] = df['col2'].astype('str')
df.dtypes
```

```
col1     int32
col2    object
dtype: object
```

```
# col2의 자료형만 float로변환
df['col2'] = df['col2'].astype('float')
df.dtypes
```

```
col1      int32
col2    float64
dtype: object
```

```
# col2의 자료형만 int로변환
df['col2'] = df['col2'].astype('int')
df.dtypes
```

```
col1    int32
col2    int32
dtype: object
```

자료형이 혼합된 컬럼의 자료형 변환

```
# 샘플 데이터
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4],'col3':[5,'-']})
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	col1	col2	col3
0	1	3	5
1	2	4	-

```
df.dtypes
```

```
col1     int64
col2     int64
col3    object
dtype: object
```

```
# col3 컬럼을 str형으로 변환
df['col3'] = df['col3'].astype('str')
type(df.loc[0,'col3'])
```

```
str
```

```
# 변경할 수 없는 자료가 섞여있으면 error
df['col3'] = df['col3'].astype('int')
```

자료형이 혼합된 컬럼을 숫자형으로 변경

- pd.to_numeric(컬럼, **errors='ignore'**) : 숫자로 변경할 수 없는 값이 있으면 작업하지 않음
- pd.to_numeric(컬럼, **errors='coerce'**) : 숫자로 변경할 수 없는 값이 NaN으로 설정
- pd.to_numeric(컬럼, **errors='raise'**) : 숫자로 변경할 수 없는 값이 있으면 에러발생(default)

astype으로 변환

```
# 모든 값을 숫자로 변경할 수 있음
s1 = pd.Series(['1.0', '2', -3])
s1.astype('float')
```

```
0    1.0
1    2.0
2   -3.0
dtype: float64
```

```
# 숫자로 변경할 수 없는 데이터가 섞여있음
s2 = pd.Series(['1.0', '2', -3, 'a'])
s2.astype('float')
```

to_numeric으로 변환

```
# ignore:숫자료 변경할 수 없는 데이터가 있으면 작업하지 않음
pd.to_numeric(s2, errors='ignore')
```

```
0    1.0
1      2
2     -3
3      a
dtype: object
```

```
# coerce:숫자로 변경할 수 없는 값이 NaN으로 설정
pd.to_numeric(s2, errors='coerce')
```

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

```
# raise : 숫자로 변경할 수 없는 값이 있으면 에러발생(default)
pd.to_numeric(s2, errors='raise')
```

시계열 데이터로 변경

- pd.to_datetime(컬럼)

```
df = pd.read_csv('data/birth_die.csv')
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	이름	주요경력	출생	사망
0	스티븐 호킹	이론 물리학자	1942-01-08	2018-03-14
1	마이클잭슨	가수	1958-08-29	2009-06-25
2	스티브잡스	CEO	1955-02-24	2011-10-05
3	로빈윌리엄스	배우	1951-07-21	2014-08-11
4	앨빈토플러	미래학자	1928-10-04	2016-06-27

```
df.dtypes
```

```
이름      object
주요경력    object
출생      object
사망      object
dtype: object
```

astype으로 변환

```
# 출생
df['출생'] = df['출생'].astype('datetime64')
```

```
df.dtypes
```

```
이름              object
주요경력            object
출생        datetime64[ns]
사망              object
dtype: object
```

to_datetime으로 변환

```
# 사망
df['사망'] = pd.to_datetime(df['사망'])
```

```
df.dtypes
```

```
이름              object
주요경력            object
출생        datetime64[ns]
사망        datetime64[ns]
dtype: object
```

할당 연산자

```
a = 1
```

```
# a,b,c에 모두 1할당
a = b = c = 1
print(a,b,c)
```

```
1 1 1
```

```
# a,b,c에 각각 1,2,3 할당
a,b,c = 1,2,3
print(a,b,c)
```

```
1 2 3
```

문자열 다루기

쌍따옴표나 홑따옴표가 포함되어 있는 문자열 사용하기

- 홑따옴표가 포함되어 있는 문자열 사용하기

'낮말'은 새가 듣고 '밤말'은 쥐가 듣는다.

```
a = "'낮말'은 새가 듣고 '밤말'은 쥐가 듣는다."
print(a)
```

```
'낮말'은 새가 듣고 '밤말'은 쥐가 듣는다.
```

- 쌍따옴표가 포함되어 있는 문자열 사용하기

"시간은 금"이라는 말이 있다.

```
b = '"시간은 금"이라는 말이 있다.'
print(b)
```

```
"시간은 금"이라는 말이 있다.
```

키보드로 변수에 값 입력받기

- input()
- input으로 입력받은 값은 무조건 str

```
a = input('나이:')
```

```
나이:20
```

```
type(a)
```

```
str
```

f스트링으로 출력하기

```
name = input('이름:')
print(name,'님, 안녕하세요?')
```

```
이름:james
james 님, 안녕하세요?
```

```
print(f'{name}님 안녕하세요')
```

```
james님 안녕하세요
```

```
age = int(input('나이:'))
print('나이는',age,'살이시군요. 내년이면',age+1,'살이 되시겠네요')
```

```
나이:20
나이는 20 살이시군요. 내년이면 21 살이 되시겠네요
```

```
print(f'나이는 {age}살이시군요. 내년이면 {age+1}살이 되시겠네요')
```

```
나이는 20살이시군요. 내년이면 21살이 되시겠네요
```

- 두 수를 입력받아 더하는 프로그램을 작성하기

```
a = int(input('첫번째숫자:'))
b = int(input('두번째숫자:'))

print(a,'+',b,'=',a+b)
```

```
첫번째숫자:1
두번째숫자:3
1 + 3 = 4
```

```
print(f'{a}+{b}={a+b}')
```

```
1+3=4
```

문자열 인덱싱

- 양수 인덱스 사용하기

```
a = 'Hello World'

# 'H' 출력하기
print(a[0])

# 'W' 출력하기
print(a[6])

# 'd' 출력하기
print(a[10])
```

```
H
W
d
```

- 음수 인덱스 사용하기

```
a = 'Hello World'

# 'H' 출력하기
print(a[-11])
```

```
# 'W' 출력하기
print(a[-5])

# 'd' 출력하기
print(a[-1])
```

```
H
W
d
```

- 범위 밖의 인덱스를 사용하면?

```
a = 'Hello World'
print(a[11])
```

```
a = 'Hello World'
print(a[-12])
```

문자열 슬라이싱

- 양수 인덱스로 슬라이싱

```
a = 'Hello World'
# 'Hello' 출력하기
print(a[0:5])

# 'World' 출력하기
print(a[6:11])
```

```
Hello
World
```

- 음수 인덱스로 슬라이싱

```
a = 'Hello World'

# 'Hello' 출력하기
print(a[-11:-6])
```

```
# 'World' 출력하기
print(a[-5:])
```

```
Hello
World
```

- 첫번째와 마지막 인덱스는 생략 가능

```
a = 'Hello World'

# 'Hello'
print(a[:5])

# 'World'
print(a[6:])

# 'Hello World'
print(a[:])
```

```
Hello
World
Hello World
```

- 한문자씩 건너뛰어 출력하기

```
a = 'Hello World'
print(a[::2])
```

```
HloWrd
```

- 역순으로 출력하기

:간격을 마이너스(-)로 하면 역순으로 출력된다.

```
a = 'Hello World'
print(a[::-1])
```

```
dlroW olleH
```

문자열 연산

```
# 문자열 더하기(문자열 연결하기)
a = 'good'
b = 'morning'
a+b
```

```
'goodmorning'
```

```
# 문자열 곱하기(문자열 반복하기)
s = 'ha'
s * 3
```

```
'hahaha'
```

```
# 문자열과 숫자형 더하기
english = 80
result = '영어점수'+ str(english)
print(result)
```

```
영어점수80
```

문자열 함수

문자열 교체하기

- 문자열.replace(찾을문자열, 교체할문자열)

```
a = '나는 초코우유 좋아. 초코우유 최고'

# '초코'-->'딸기'로 교체
print(a.replace('초코','딸기'))
```

```
나는 딸기우유 좋아. 딸기우유 최고
```

문자의 위치 찾기

- 문자열.find(문자)

```
a = 'Hello, Python!!!'

# 'e'의 위치 찾기
print(a.find('e'))

# 'l'의 위치 찾기
print(a.find('l'))
```

```
1
2
```

소문자로 변환하기

*문자열.lower()

```
a = 'Hello, Python!!!'
a.lower()
```

```
'hello, python!!!'
```

대문자로 변환하기

*문자열.upper()

```
a = 'Hello, Python!!!'
a.upper()
```

```
'HELLO, PYTHON!!!'
```

문자열 나누기

*문자열.split(구분자)

```

```

```
phone = '010-123-4567'
phone.split('-')
```

```
['010', '123', '4567']
```

```
email = 'abc@naver.com'
email.split('@')
```

```
['abc', 'naver.com']
```

연습문제

- 사용자의 영문 이름을 입력받아 성과 이름 순서를 바꾸어서 출력하는 프로그램을 작성하세요.

  성과 이름은 공백으로 구분합니다.

```
# 사용자의 영문이름 입력받기 (성과 이름은 공백으로 구분)
full_name = input('영문이름(성과 이름은 공백으로 구분하세요):')

# 공백의 위치 찾기
space = full_name.find(' ')
print(space)
# 성, 이름을 슬라이싱하여 각각 변수에 담기
first_name = full_name[:space]
last_name = full_name[space+1:]

print()

# 성, 이름의 순서를 바꾸어 출력하기
```

```
영문이름(성과 이름은 공백으로 구분하세요):py thon
2
```

수치형 변수 다루기

기본 연산자

```
a, b = 3,4
```

```
print('a+b=',a+b)
print('a-b=',a-b)
print('a*b=',a*b)
print('a/b=',a/b)
print('a//b=',a//b)
print('a%b=',a%b)
print('a**b=',a**b)
```

```
a+b= 7
a-b= -1
a*b= 12
a/b= 0.75
a//b= 0
a%b= 3
a**b= 81
```

복합 할당 연산자

```
x = 10
x += 20  # x = x+20 = 10 + 20 = 30
print(x)
```

```
30
```

```
x = 3
y = 5
x *= x+y   # x = x * (x+y) = 3 * (3 + 5) = 24
print(x)
```

```
24
```

연습문제

- 화씨 온도를 입력받아 섭씨 온도로 변환하는 프로그램을 작성해보세요.

$ C = (F - 32) * \frac{5}{9}$

```
# 화씨온도 입력
f = float(input('화씨온도:'))
```

```
# 섭씨온도로 계산
c = (f - 32) * (5/9)

# 결과 출력
print(f'화씨온도:{f} --> 섭씨온도:{c}')
```

```
화씨온도:100
화씨온도:100.0 --> 섭씨온도:37.77777777777778
```

리스트 다루기

리스트 만들기

```
# []를 사용하여 빈 리스트 만들기
l1 = []
l1
```

```
[]
```

```
# list() 함수를 사용하여 빈 리스트 만들기
l2 = list()
l2
```

```
[]
```

```
# []를 사용하여 초기값이 있는 리스트 만들기
l3 = [1,3,5,7,9]
l3
```

```
[1, 3, 5, 7, 9]
```

```
# list() 함수를 사용하여 초기값이 있는 리스트 만들기(규칙이 있는 정수값의 나열)
l4 = list(range(1,100,2))
l4
```

```
[1,
 3,
 5,
 7,
 9,
 11,
 13,
 15,
 17,
 19,
 21,
 23,
 25,
 27,
 29,
 31,
 33,
 35,
 37,
 39,
 41,
 43,
 45,
 47,
 49,
 51,
 53,
 55,
 57,
 59,
 61,
 63,
 65,
 67,
 69,
 71,
 73,
 75,
 77,
 79,
 81,
 83,
 85,
 87,
 89,
 91,
 93,
 95,
 97,
 99]
```

리스트의 자료형

- 모든 자료형이 혼합되어 들어갈 수 있다.

```
myinfo = ['amy',20,165.5, ['독서','코딩']]
myinfo
```

```
['amy', 20, 165.5, ['독서', '코딩']]
```

리스트 인덱싱

- 리스트명[인덱스]

```
l1 = ['축구','농구','배구','야구','족구','발야구','피구']
```

```
# l1 항목수
len(l1)
```

```
7
```

```
# '축구'
print(l1[0])
print(l1[-7])
```

```
축구
축구
```

```
# '배구'
print(l1[2])
print(l1[-5])
```

```
배구
배구
```

```
# '피구'
print(l1[6])
print(l1[-1])
```

```
피구
피구
```

중첩리스트 인덱싱

```
# 아래 리스트 l에서 '수박' 인덱싱하여 가져오기
l2 = ['사과','오렌지','포도',['수박','바나나']]
l2[3][0]
```

```
'수박'
```

```
# 아래 튜플 t에서 'Life' 인덱싱하여 가져오기
t2 = (1,2,('a','b',('Life','is')))
t2[2][2][0]
```

```
'Life'
```

리스트 슬라이싱

- 리스트명[시작인덱스:끝인덱스:간격]

```
l1 = ['축구','농구','배구','야구','족구','발야구','피구']
```

```
# ['농구','배구']
l1[1:3]
```

```
['농구', '배구']
```

```
# ['축구','농구','배구']
print(l1[0:3])
print(l1[:3])
```

```
['축구', '농구', '배구']
['축구', '농구', '배구']
```

```
# ['족구','발야구']
l1[-3:-1]
```

```
['족구', '발야구']
```

```
# ['족구','발야구','피구']
l1[-3:]
```

```
['족구', '발야구', '피구']
```

```
# 모든 항목
l1[:]
```

```
['축구', '농구', '배구', '야구', '족구', '발야구', '피구']
```

```
# 모든 항목 역순으로
l1[::-1]
```

```
['피구', '발야구', '족구', '야구', '배구', '농구', '축구']
```

```
# ['발야구','족구','야구']
l1[-2:-5:-1]
```

```
['발야구', '족구', '야구']
```

```
# 아래 리스트 l3에서 홀수만 슬라이싱하기
l3 = list(range(1,21))
l3[::2]
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

리스트 연결하기

```
l1 = ['사과','딸기','포도']
l2 = ['포도','수박']
l1 + l2
```

```
['사과', '딸기', '포도', '포도', '수박']
```

리스트 반복하기

```
l = ['먹고','자고']
l * 3
```

```
['먹고', '자고', '먹고', '자고', '먹고', '자고']
```

리스트 변경하기

리스트에 항목 추가하기

리스트 끝에 항목 추가하기

- 리스트명.append(항목)

```
todolist = ['양치','물마시기']
```

```
# '아침운동' 항목 추가하기
todolist.append('아침운동')
```

```
todolist
```

```
['양치', '물마시기', '아침운동']
```

리스트 중간에 항목 삽입하기

- 리스트명.insert(인덱스, 항목)

```
# '아침운동'하기 전에 '아침식사'
todolist.insert(2,'아침식사')
```

```
todolist
```

```
['양치', '물마시기', '아침식사', '아침운동']
```

리스트 끝에 여러 항목 추가하기

- 리스트명.extend(리스트)

```
# '아침운동' 후에 ['샤워','드라이','공부']
```

```
todolist.extend(['샤워','드라이','공부'])
```

```
todolist
```

```
['양치', '물마시기', '아침식사', '아침운동', '샤워', '드라이', '공부']
```

리스트 항목 수정하기

하나의 항목 수정하기

- 리스트명[인덱스] = 항목

```
# 아침운동-->산책으로 수정
todolist[3]='산책'
```

```
todolist
```

```
['양치', '물마시기', '아침식사', '산책', '샤워', '드라이', '공부']
```

여러 항목 수정하기

- 리스트명[시작인덱스:끝인덱스] = 리스트

```
# '아침식사','산책' --> '독서','산책','아침식사'
todolist[2:4] = ['독서','산책','아침식사']
```

```
todolist
```

```
['양치', '물마시기', '독서', '산책', '아침식사', '샤워', '드라이', '공부']
```

리스트 항목 삭제하기

인덱스를 이용하여 항목 삭제

- del 리스트명[인덱스]

```
# 마지막 인덱스 삭제
del todolist[-1]
```

```
todolist
```

```
['양치', '물마시기', '독서', '산책', '아침식사', '샤워', '드라이']
```

항목값을 이용하여 삭제

- 리스트.remove(항목)

항목이 여러개인 경우 앞의 인덱스만 삭제

```
# '양치' 삭제
todolist.remove('양치')
```

```
todolist
```

```
['물마시기', '독서', '산책', '아침식사', '샤워', '드라이']
```

마지막 항목을 반환하고 삭제

- 리스트.pop()

```
a = todolist.pop()
```

```
a
```

```
'드라이'
```

```
todolist
```

```
['물마시기', '독서', '산책', '아침식사', '샤워']
```

연습문제

- 아래 wishlist의 '시계','신발'을 myCart로 이동해봅시다.

```
wishlist = ['가방','시계','신발']
mycart = []

#wishlist의 시계, 신발을 추출해서 mycart에 담는다.
mycart.extend(wishlist[1:3])

#wishlist의 시계, 신발을 삭제한다.
del wishlist[1:3]
```

```
wishlist
```

```
['가방']
```

```
mycart
```

```
['시계', '신발']
```

학생별 총점,평균 구하기

- 다음은 학생 별 [국어,영어,수학]점수가 저장된 리스트이다.

score_list = [[96,84,80],[96,86,76],[76,95,83],[89,96,69],[90,76,91]]

각 학생의 세 과목의 성적의 [총점, 평균]을 구하여 리스트에 담으시오.

(평균은 반올림하여 소수점 1자리까지 표현한다.)

```
score_list = [[96,84,80],[96,86,76],[76,95,83],[89,96,69],[90,76,91]]
stu_scores = []
```

```
for i in score_list:
    total = sum(i)
    average = total/3
    #print([total,rount(average,1)])
    stu_scores.append([total,round(average,1)])
stu_scores
```

```
[[260, 86.7], [258, 86.0], [254, 84.7], [254, 84.7], [257, 85.7]]
```

과목별 평균 구하기

- 다음은 학생 별 [국어,영어,수학]점수가 저장된 리스트이다.

score_list = [[96,84,80],[96,86,76],[76,95,83],[89,96,69],[90,76,91]]

각 과목의 리스트를 분리하고 과목별 평균을 구해봅시다.

평균은 소수점 1자리까지 출력한다.

```
score_list = [[96,84,80],[96,86,76],[76,95,83],[89,96,69],[90,76,91]]
kor_list=[]
eng_list=[]
math_list=[]

kor_average=0
eng_average=0
math_average=0

#각 과목의 리스트 분리하기
for i in score_list:
    #print(i)
    kor_list.append(i[0])
    eng_list.append(i[1])
    math_list.append(i[2])

print('kor_list:',kor_list)
print('eng_list:',eng_list)
print('math_list:',math_list)

kor_average = sum(kor_list)/len(kor_list)
eng_average = sum(eng_list)/len(eng_list)
math_average = sum(math_list)/len(math_list)

print('kor_average:',round(kor_average,1))
print('eng_average:',round(eng_average,1))
print('math_average:',round(math_average,1))
```

```
kor_list: [96, 96, 76, 89, 90]
eng_list: [84, 86, 95, 96, 76]
math_list: [80, 76, 83, 69, 91]
kor_average: 89.4
eng_average: 87.4
math_average: 79.8
```

랜덤 항목 추출하기

- random.choice(리스트명)

```
menulist = ['한식','일식','중식','양식','분식','이탈리아식']
print('오늘은 뭘먹지?')
```

```
오늘은 뭘먹지?
```

```
import random
random.choice(menulist)
```

```
'양식'
```

- 메뉴를 입력받아 랜덤으로 메뉴를 정하는 프로그램을 작성해봅시다.

공백('')을 입력할 때까지 메뉴를 입력받습니다.

```
menulist = []
while True:
    menu = input('메뉴:')
    if menu=='':
        break
    menulist.append(menu)

print('오늘의 메뉴는:',random.choice(menulist))
```

```
메뉴:피자
메뉴:치킨
메뉴:떡볶이
메뉴:
오늘의 메뉴는: 치킨
```

리스트에 값 존재여부 확인

- 멤버연산자 : in, not in

```

```

```
l = [1,2,3,4,5]
print(5 in l)
print(10 in l)

print(5 not in l)
print(10 not in l)
```

```
True
False
False
True
```

리스트 원소의 인덱스 찾기

- 리스트.index(찾는 값)

```
l1 = ('스키','보드','스케이트','스케이트보드','수상스키','웨이크보드')
```

```
# '스케이트'의 인덱스 찾기
l1.index('스케이트')
```

```
2
```

```
# '오토바이'의 인덱스 찾기
l1.index('오토바이')
```

```
if '오토바이' in l1:
    print(l1.index('오토바이'))
else:
    print('존재하지 않음')
```

```
존재하지 않음
```

리스트 항목 개수 구하기 : len()

```
l = [1,3,2,4,3,5,7,8,77.7,88.8,'a','b','c',[10,20,30]]
len(l)
```

```
14
```

리스트의 통계값 구하기

- 항목수 : len(리스트명)
- 합계 : sum(리스트명)
- 최소값 : min(리스트명)
- 최대값 : max(리스트명)
- 항목의 갯수 : 리스트명.count(항목)

```
l = [1,2,3,4,5,3,4,5]
print('항목수:',len(l))
print('합계:',sum(l))
print('최소값:',min(l))
print('최대값:',max(l))
print('평균:',sum(l)/len(l))
print('항목5의 갯수:',l.count(5))
```

```
항목수: 8
합계: 27
최소값: 1
최대값: 5
평균: 3.375
항목5의 갯수: 2
```

리스트 정렬하기

원본리스트 정렬

- 리스트명.sort()
- 리스트명.sort(reverse=True) : 내림차순

```
friends = ['수지','지은','찬혁','수현','범준']
```

```
friends.sort()
```

```
friends
```

```
['범준', '수지', '수현', '지은', '찬혁']
```

```
friends.sort(reverse=True)
```

```
friends
```

```
['찬혁', '지은', '수현', '수지', '범준']
```

복사본 만들어서 정렬

- sorted(리스트명)
- sorted(리스트명, reverse=True) : 내림차순

```
friends = ['수지','지은','찬혁','수현','범준']
```

```
sorted_friends = sorted(friends)
```

```
sorted_friends
```

```
['범준', '수지', '수현', '지은', '찬혁']
```

```
friends
```

```
['수지', '지은', '찬혁', '수현', '범준']
```

리스트 순서 뒤집기

- 리스트명.reverse()

```
friends = ['수지','지은','찬혁','수현','범준']
```

```
friends.reverse()
```

```
friends
```

```
['범준', '수현', '찬혁', '지은', '수지']
```

2차원리스트 다루기

- 2차원리스트 만들기

```
# 리스트 만들기
l2 = [[10,20,30],[40,50,60]]

# 30 추출하기
print(l2[0][2])

# 1행 2열 추출하기
print(l2[1][2])
```

```
30
60
```

- 연습

파일리스트에서 파일명과 확장자를 분리하여 다음과 같은 형태로 저장하는 리스트를 만들어봅시다.

|파일명|확장자|

|--|--|

|file1|py|

|file2|txt|

|file3|pptx|

```
file_list = ['file1.py','file2.txt','file3.pptx']
name_extension = []

for i in file_list:
    #print(i.split('.'))
    name_extension.append(i.split('.'))

print(name_extension)
```

```
[['file1', 'py'], ['file2', 'txt'], ['file3', 'pptx']]
```

튜플 다루기

튜플 만들기

```
# ( )를 사용하여 빈 튜플 만들기
t1 = ()
t1
```

```
()
```

```
# tuple() 함수를 사용하여 빈 튜플 만들기
t2 = tuple()
t2
```

```
()
```

```
# ( )에 초기값 지정하여 튜플 만들기
t3 = (1,3,5,7,9)
t3
```

```
(1, 3, 5, 7, 9)
```

```
# tuple() 함수를 사용하여 초기값이 있는 튜플 만들기(규칙이 있는 정수값의 나열)
t4 = tuple(range(1,100,2))
t4
```

```
(1,
 3,
 5,
 7,
 9,
 11,
 13,
 15,
 17,
 19,
 21,
 23,
 25,
 27,
 29,
 31,
 33,
 35,
 37,
 39,
 41,
 43,
 45,
 47,
 49,
 51,
 53,
 55,
 57,
 59,
 61,
 63,
 65,
 67,
 69,
 71,
 73,
 75,
 77,
 79,
 81,
 83,
```

```
 85,
 87,
 89,
 91,
 93,
 95,
 97,
 99)
```

```
# 튜플을 만들 때 괄호()를 생략할 수 있다.
t5 = 1,3,5,7,9
t5
```

```
(1, 3, 5, 7, 9)
```

```
# 항목이 1개일 때는 ,를 붙인다.
t6 = (1,)
type(t6)
```

```
tuple
```

```
t7 = 1,
t7
```

```
(1,)
```

튜플의 자료형

- 모든 자료형이 혼합되어 들어갈 수 있다.

```
myinfo = ('amy',20,165.5, ['독서','코딩'])
myinfo
```

```
('amy', 20, 165.5, ['독서', '코딩'])
```

튜플 연결하기

```
t1 = ('빨','주')
t2 = ('노','초','파','남','보')
t1 + t2
```

```
('빨', '주', '노', '초', '파', '남', '보')
```

튜플 인덱싱

- 튜플명[인덱스]

```
t1 = ('스키','보드','스케이트','스케이트보드','수상스키','웨이크보드')
```

```
# t1 항목수
len(t1)
```

```
6
```

```
# '스키'
print(t1[0])
print(t1[-6])
```

```
스키
스키
```

```
# '웨이크보드'
print(t1[5])
print(t1[-1])
```

```
웨이크보드
웨이크보드
```

튜플 반복하기

```
t = ('놀고','먹고','자고')
t * 2
```

```
('놀고', '먹고', '자고', '놀고', '먹고', '자고')
```

튜플에 값 존재여부 확인

- 멤버연산자 : in, not in

```
l = [1,2,3,4,5]
print(5 in l)
print(10 in l)

print(5 not in l)
print(10 not in l)
```

```
True
False
False
True
```

```
t = (1,2,3,4,5)
print(5 in t)
print(10 in t)

print(5 not in t)
print(10 not in t)
```

```
True
False
False
True
```

튜플 원소 인덱스 찾기

- 튜플명.index(항목)

```
t1 = ('스키','보드','스케이트','스케이트보드','수상스키','웨이크보드')
```

```
# '스케이트'의 인덱스 찾기
t1.index('스케이트')
```

```
2
```

```
# '오토바이'의 인덱스 찾기
t1.index('오토바이')
```

```
if '오토바이' in t1:
    print(t1.index('오토바이'))
else:
    print('존재하지 않음')
```

```
존재하지 않음
```

튜플 항목 개수 구하기 : len()

```
t = ((1,2),[3,4])
len(t)
```

```
2
```

딕셔너리 다루기

딕셔너리 만들기

딕셔너리명 = {키1:값1, 키2:값2,...,}

- 중괄호 안에 키:값의 쌍으로 된 항목을 콤마(,)로 구분하여 적어준다.

|메뉴|가격|

|----|----|

|김밥|2000|

|떡볶이|2500|

|어묵|2000|

|튀김|3000|

```
menu = {'김밥':2000, '떡볶이':2500, '어묵':2000, '튀김':3000}
menu
```

```
{'김밥': 2000, '떡볶이': 2500, '어묵': 2000, '튀김': 3000}
```

```
menu = {'김밥':2000,
        '떡볶이':2500,
        '어묵':2000,
        '튀김':3000}
menu
```

```
{'김밥': 2000, '떡볶이': 2500, '어묵': 2000, '튀김': 3000}
```

dict로 딕셔너리 만들기

- 딕셔너리명 = dict(키1=값1, 키2=값2,...,)
- 키에 따옴표('')를 쓰지 않는다는 점에 주의한다.
- 키에 따옴표('')를 쓰지 않아도 딕셔너리가 생성되면서 자동으로 문자열형으로 지정된다.

```
menu1 = dict(김밥=2000, 떡볶이=2500, 어묵=2000, 튀김=3000)
menu1
```

```
{'김밥': 2000, '떡볶이': 2500, '어묵': 2000, '튀김': 3000}
```

- 딕셔너리명 = dict(zip(key리스트, value리스트))

```
key_list = ['김밥','떡볶이','어묵','튀김']
value_list = [2000,2500,2000,3000]
```

```
menu2 = dict(zip(key_list,value_list))
menu2
```

```
{'김밥': 2000, '떡볶이': 2500, '어묵': 2000, '튀김': 3000}
```

- 딕셔너리명 = dict([(키1,값1),(키1,값2),...,])

```
[('김밥',2000),('떡볶이',2500),('어묵',2000),('튀김',3000)]
menu3 = dict([('김밥',2000),('떡볶이',2500),('어묵',2000),('튀김',3000)])
menu3
```

```
{'김밥': 2000, '떡볶이': 2500, '어묵': 2000, '튀김': 3000}
```

- 딕셔너리명 = dict({키1:값1,키2:값2,...,})

```
menu4 = dict({'김밥':2000, '떡볶이':2500, '어묵':2000, '튀김':3000})
print(menu4)
```

```
{'김밥': 2000, '떡볶이': 2500, '어묵': 2000, '튀김': 3000}
```

딕셔너리에 사용할 수 있는 자료형

- 딕셔너리의 value에는 모든 자료형을 혼합하여 사용할 수 있다.

```
person = {'name':'james',
          '나이':25,
          '키':175.5,
          '시력':(1.0,1.0),
          '취미':['운동','독서']}
person
```

```
{'name': 'james', '나이': 25, '키': 175.5, '시력': (1.0, 1.0), '취미': ['운
동', '독서']}
```

- 딕셔너리의 key에는 숫자, 문자열, 부울형, 튜플을 사용할 수 있다.

```
number=7

dict1 = {
    100:'hundred',
    True:'참',
    False:'거짓',
    (1,3):'학년,반',
    number:'번호'
}
print(dict1)
```

```
{100: 'hundred', True: '참', False: '거짓', (1, 3): '학년,반', 7: '번호'}
```

중복된 key를 사용하여 딕셔너리 만들면?

```
person = {'name':'james',
          '나이':25,
          '나이':30,
          '키':175.5,
          '시력':(1.0,1.0),
          '취미':['운동','독서']}
person
```

```
{'name': 'james', '나이': 30, '키': 175.5, '시력': (1.0, 1.0), '취미': ['운
동', '독서']}
```

딕셔너리 값 추출하기

딕셔너리명[key]

```
person['나이']
```

```
30
```

존재하지 않는 key로 추출하면?

- KeyError가 발생한다.

```
person['몸무게']
```

딕셔너리에 키 존재여부 확인하기

- key in 딕셔너리명

```
if '몸무게' in person:
    print(person['몸무게'])
else:
    print('존재하지 않음')
```

```
존재하지 않음
```

딕셔너리명.get(key, msg)

- 존재하지 않는 key로 추출 시도해도 오류가 발생하지 않는다.
- 존재하지 않는 key로 추출 시도할 경우 출력할 메시지를 설정할 수 있다.

```
person.get('나이')
```

```
30
```

```
person.get('몸무게')
```

```
person.get('몸무게','존재하지않음')
```

```
'존재하지않음'
```

딕셔너리의 키, 값 호출

딕셔너리명.keys()

- 딕셔너리의 키만 리스트로 가져오기

- dict_keys객체로 받아온다. 리스트처럼 사용할 수 있지만 리스트는 아니다.

```
scores = {'kor':100, 'eng':90, 'math':80}
scores.keys()
```

```
dict_keys(['kor', 'eng', 'math'])
```

딕셔너리명.values()

- 딕셔너리의 value만 리스트로 가져오기
- dict_values객체로 받아온다.

```
scores = {'kor':100, 'eng':90, 'math':80}
scores.values()
```

```
dict_values([100, 90, 80])
```

딕셔너리명.items()

- 딕셔너리의 (key,value) 쌍을 리스트로 가져오기
- dict_items객체로 받아온다.

```
scores = {'kor':100, 'eng':90, 'math':80}
scores.items()
```

```
dict_items([('kor', 100), ('eng', 90), ('math', 80)])
```

for문으로 딕셔너리 출력하기

for문으로 딕셔너리의 키만 출력하기

```
scores = {'kor':100, 'eng':90, 'math':80}

for i in scores.keys():
    print(i)
```

```
kor
eng
math
```

```
# 딕셔너리로 for문 돌리기
for i in scores:
    print(i)
```

```
kor
eng
math
```

for문으로 딕셔너리의 value만 출력하기

```
scores = {'kor':100, 'eng':90, 'math':80}
for i in scores.values():
    print(i)
```

```
100
90
80
```

for문으로 딕셔너리의 key, value 출력하기

```
scores = {'kor':100, 'eng':90, 'math':80}
for i in scores.items():
    print(i)
```

```
('kor', 100)
('eng', 90)
('math', 80)
```

```
scores = {'kor':100, 'eng':90, 'math':80}
for k,v in scores.items():
    print(k,v)
```

```
kor 100
eng 90
math 80
```

딕셔너리 정렬하기

키 정렬하기

```
scores = {'kor':100, 'eng':90, 'math':80}
sorted(scores.keys())
```

```
['eng', 'kor', 'math']
```

값 정렬하기

```
scores = {'kor':100, 'eng':90, 'math':80}
sorted(scores.values())
```

```
[80, 90, 100]
```

(키,값) 정렬하기

```
scores = {'kor':100, 'eng':90, 'math':80}
sorted(scores.items())
```

```
[('eng', 90), ('kor', 100), ('math', 80)]
```

```
# .sort()는 지원하지 않음
scores.keys().sort()
```

딕셔너리 항목 추가/수정

딕셔너리명[키]=값

- 키가 존재하지 않으면 추가, 존재하면 수정된다.

```
scores = {'kor':100, 'eng':90, 'math':80}

# math점수 85점으로 수정하기
scores['math']=85

# music점수 95점 추가하기
scores[' music']=95

scores
```

```
{'kor': 100, 'eng': 90, 'math': 85, ' music': 95}
```

setdefault로 항목 추가하기

- 딕셔너리명.setdefault(키,값)
- 이미 들어있는 키의 값은 수정할 수 없다.

```
scores = {'kor':100, 'eng':90, 'math':80}

# music점수 넣기(key만 넣고 value를 생략하면? value에 빈값)
scores.setdefault('music')

scores
```

```
{'kor': 100, 'eng': 90, 'math': 80, 'music': None}
```

```
scores = {'kor':100, 'eng':90, 'math':80}

# music점수 90점 추가
scores.setdefault('music',90)

scores
```

```
{'kor': 100, 'eng': 90, 'math': 80, 'music': 90}
```

```
scores = {'kor': 100, 'eng': 90, 'math': 80, 'music': 95}

# music점수 85점으로 수정
scores.setdefault('music',85)

scores
```

```
{'kor': 100, 'eng': 90, 'math': 80, 'music': 95}
```

update로 여러 항목 수정하기

- 키가 존재하면 수정, 존재하지 않으면 추가된다.
- **딕셔너리명.update(키1=값1, 키1=값2,...)**
- 키에 따옴표를 하지 않지만, 딕셔너리에 들어갈 때는 따옴표가 붙어서 들어간다.

```
scores = {'kor':100, 'eng':90, 'math':80}

# math:90, music:90
scores.update(math=90, music=90)

scores
```

```
{'kor': 100, 'eng': 90, 'math': 90, 'music': 90}
```

- **딕셔너리명.update(zip(key리스트, value리스트)**

```
scores = {'kor':100, 'eng':90, 'math':80}

# math:90, music:90
scores.update(zip(['math','music'],[90,90]))

scores
```

```
{'kor': 100, 'eng': 90, 'math': 90, 'music': 90}
```

- **딕셔너리명.update([(키1,값1),(키2,값2),...])**

```
scores = {'kor':100, 'eng':90, 'math':80}

# math:90, music:90
scores.update([('math',90),('music',90)])
scores
```

```
{'kor': 100, 'eng': 90, 'math': 90, 'music': 90}
```

- **딕셔너리명.update({키1:값1,키2:값2,...,})**

```
scores = {'kor':100, 'eng':90, 'math':80}

# math:90, music:90
scores.update({'math': 90, 'music': 90})

scores
```

```
{'kor': 100, 'eng': 90, 'math': 90, 'music': 90}
```

딕셔너리에서 항목 삭제하기

del 딕셔너리명[키]

- 해당 키의 항목 삭제

```
scores = {'kor':100, 'eng':90, 'math':80}

# 키가 'kor'인 항목 삭제
del scores['kor']

scores
```

```
{'eng': 90, 'math': 80}
```

딕셔너리명.pop(키, 기본값)

- 해당 키의 항목(값) 반환하고 삭제

```
scores = {'kor':100, 'eng':90, 'math':80}

# 키가 'kor'인 항목의 값 받아온 후 삭제
kor = scores.pop('kor')

print(kor)
print(scores)
```

```
100
{'eng': 90, 'math': 80}
```

- 해당 키의 항목 반환하고 삭제(키가 존재하지 않을 때 기본값 반환)

```
scores = {'kor':100, 'eng':90, 'math':80}

# 키가 'music'인 항목의 값 받아온 후 삭제(삭제할 키가 존재하지 않는 경우)
music = scores.pop('music','x')


print(music)
print(scores)
```

```
x
{'kor': 100, 'eng': 90, 'math': 80}
```

딕셔너리명.clear()

- 딕셔너리의 모든 항목 삭제

```
scores = {'kor':100, 'eng':90, 'math':80}

scores.clear()
print(scores)
```

```
{}
```

연습문제

영어단어장 만들기

엔터를 입력할 때까지 영어단어,뜻을 입력받아 단어장을 만들고, 입력이 끝나면 단어 테스트를 실시하는 프로그램을 만들어봅시다.

1. 단어장을 만듭니다.
2. 단어테스트를 실시하고 맞은 갯수를 계산합니다.
3. 테스트가 끝나면 맞은갯수/전체단어수/점수 형태로 결과를 출력합니다

단어장 만들기

엔터('')를 입력할 때까지 영어단어,뜻을 입력받아 딕셔너리에 저장

```
dict_word={}
while True:
    input_word = input('영어단어,뜻:')
    if input_word=='':
        break
    eng = input_word.split(',')[0]
    kor = input_word.split(',')[1]
    dict_word[eng] = kor
dict_word
```

```
영어단어,뜻:apple,사과
영어단어,뜻:banana,바나나
영어단어,뜻:grapes,포도
영어단어,뜻:
```

```
{'apple': '사과', 'banana': '바나나', 'grapes': '포도'}
```

단어테스트

단어장의 단어들을 모두 테스트

맞은 갯수는 별도로 카운트

```
cnt = 0
for eng, kor in dict_word.items():
    answer = input(eng)
    if answer==kor:
        print('O')
        cnt+=1
    else:
        print('X')
```

```
apple사과
O
banana수박
X
grapes포도
O
```

테스트 결과 출력

맞은갯수/전체문제수/점수 출력

```
print('맞은갯수:',cnt)
print('전체문제수:',len(dict_word))
print('점수:',round(cnt/len(dict_word)*100,1))
```

```
맞은갯수: 2
전체문제수: 3
점수: 66.7
```

datetime형 다루기

컬럼을 datetime 자료형으로 변경하기

- pd.**to_datetime**(컬럼)

```
import pandas as pd
df = pd.read_csv('data/birth_die.csv')
df
```

```
df.dtypes
```

```
# 출생, 사망 컬럼을 datetime 자료형으로 변경하기
df['출생'] = pd.to_datetime(df['출생'])
```

```
df['사망'] = pd.to_datetime(df['사망'])
```

```
df.dtypes
```

연, 월, 일, 분기 추출하기

- 컬럼.dt.year
- 컬럼.dt.month
- 컬럼.dt.day
- 컬럼.dt.quarter

```
# 출생 컬럼 0번 인덱스의 연도
df['출생'][0].year
```

```
# 출생 컬럼의 연도
df['출생'].dt.year
```

```
df['출생'].dt.month
```

```
df['출생'].dt.day
```

```
# 분기 컬럼 만들기
df['분기']=df['출생'].dt.quarter
df
```

요일, 월이름 추출하기

- 컬럼.dt.strftime('%a') : 요약된 요일이름
- 컬럼.dt.strftime('%A') : 긴 요일이름
- 컬럼.dt.strftime('%w') : 숫자요일(0:일요일)
- 컬럼.dt.strgtime('%b') : 요약된 월이름
- 컬럼.dt.strftime('%B') : 긴 월이름

```
df['출생'].dt.strftime('%b')
```

```
# 출생요일 컬럼
df['출생요일'] = df['출생'].dt.strftime('%a')
df
```

```
# 출생월 컬럼 추가하기
df['출생월'] = df['출생'].dt.strftime('%b')
df
```

날짜 계산하기

```
# 생존일수 컬럼 만들기
df['생존일수'] = df['사망']-df['출생']
df
```

```
# 생존기간 컬럼 만들기
df['생존기간'] = df['사망'].dt.year-df['출생'].dt.year
df
```

datetime 자료형을 인덱스로 만들어 사용하기

```
# 출생 컬럼을 인덱스로 만들기
df.index = df['출생']
df
```

```
# 1955년 출생한 데이터 추출하기
df.loc['1955']
```

```
# 1955년 2월 출생한 데이터 추출하기
df.loc['1955-02']
```

카테고리형 다루기

데이터 준비/확인

- 평균점수에 따른 등급을 카테고리형 자료로 다루기 위해 평균점수에 따른 등급 컬럼 추가하기

```
import pandas as pd
df = pd.read_csv('data/scores.csv')
df.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0

결측치 확인/처리하기

- 결측치는 0으로 대치

```
# 결측치 확인하기 (info)
df.info()
```

```
RangeIndex: 30 entries, 0 to 29
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   name    30 non-null     object
 1   kor     27 non-null     float64
 2   eng     28 non-null     float64
 3   math    29 non-null     float64
dtypes: float64(3), object(1)
memory usage: 1.1+ KB
```

```
# 결측치가 있는 행 삭제하기
df.dropna(inplace=True)
```

```
# 결측치 확인(isnull)
df.isnull().sum()
```

```
name    0
kor     0
eng     0
math    0
dtype: int64
```

평균 점수 컬럼 추가하기

- 등급을 매기기 위한 평균점수 컬럼 추가

```
df['average'] = round((df.kor+df.eng+df.math)/3,1)
df.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math	average
0	Aiden	100.0	90.0	95.0	95.0

평균점수에 따른 등급 컬럼 추가

- 컬럼.apply(함수)

```
def get_grade(x):
    if x>=90:
        return 1
    elif x>=80:
        return 2
    elif x>=70:
        return 3
    elif x>=60:
        return 4
    else:
        return 5

df['grade'] = df['average'].apply(get_grade)
df.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math	average	grade
0	Aiden	100.0	90.0	95.0	95.0	1

카테고리형으로 변환하기

- 등급컬럼(grade)을 카테고리형 자료형으로 변환하기

```
# 자료형 확인하기
df.dtypes
```

```
name         object
kor         float64
eng         float64
math        float64
average     float64
grade         int64
dtype: object
```

```
# 자료형 변환하기
df['grade'] = df['grade'].astype('category')
```

```
df.dtypes
```

```
name          object
kor          float64
eng          float64
math         float64
average      float64
grade       category
dtype: object
```

```
df['grade'].dtype
```

```
CategoricalDtype(categories=[1, 2, 3, 4], ordered=False)
```

카테고리 이름 바꾸기

- 컬럼.**cat.categories** = 카테고리리스트

```
df['grade'].cat.categories = ['A','B','C','D']
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math	average	grade
0	Aiden	100.0	90.0	95.0	95.0	A
1	Charles	90.0	80.0	75.0	81.7	B
2	Danial	95.0	100.0	100.0	98.3	A
3	Evan	100.0	100.0	100.0	100.0	A
5	Ian	90.0	100.0	90.0	93.3	A
6	James	70.0	75.0	65.0	70.0	C
7	Julian	80.0	90.0	55.0	75.0	C

누락된 카테고리 추가

- 컬럼.**cat.set_categories**(카테고리리스트)

```
df['grade'] = df['grade'].cat.set_categories(['A','B','C','D','F'])
df['grade'].dtypes
```

```
CategoricalDtype(categories=['A', 'B', 'C', 'D', 'F'], ordered=False)
```

데이터 용량 확인하기

- titanic 데이터에서 카테고리형으로 관리할 수 있는 자료형을 카테고리형으로 변환하여 데이터 용량 비교하기

데이터 준비하고 확인하기

```
df_titanic = pd.read_csv('data/titanic.csv')
df_titanic.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2

```
df_titanic['Survived'].unique()
```

```
array([0, 1], dtype=int64)
```

```
df_titanic['Pclass'].unique()
```

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

```
df_titanic['Sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
df_titanic['Embarked'].unique()
```

```
array(['S', 'C', 'Q', nan], dtype=object)
```

```
# 데이터 타입
df_titanic.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
# 용량 확인하기
df_titanic.info()
```

```
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  1309 non-null   int64
 1   Survived     1309 non-null   int64
 2   Pclass       1309 non-null   int64
 3   Name         1309 non-null   object
 4   Sex          1309 non-null   object
 5   Age          1046 non-null   float64
 6   SibSp        1309 non-null   int64
 7   Parch        1309 non-null   int64
 8   Ticket       1309 non-null   object
 9   Fare         1308 non-null   float64
 10  Cabin        295 non-null    object
 11  Embarked     1307 non-null   object
dtypes: float64(2), int64(5), object(5)
memory usage: 122.8+ KB
```

카테고리형으로 바꾸기

- 컬럼.astype('category')

```
# 카테고리형으로 바꾸기 (Survived, Pclass, Sex, Embarked)
df_titanic['Survived'] = df_titanic['Survived'].astype('category')
df_titanic['Pclass'] = df_titanic['Pclass'].astype('category')
```

```
df_titanic['Sex'] = df_titanic['Sex'].astype('category')
df_titanic['Embarked'] = df_titanic['Embarked'].astype('category')
```

```
# 데이터타입
df_titanic.dtypes
```

```
PassengerId       int64
Survived       category
Pclass         category
Name             object
Sex            category
Age             float64
SibSp             int64
Parch             int64
Ticket           object
Fare            float64
Cabin            object
Embarked       category
dtype: object
```

```
# 용량
df_titanic.info()
```

```
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  1309 non-null   int64
 1   Survived     1309 non-null   category
 2   Pclass       1309 non-null   category
 3   Name         1309 non-null   object
 4   Sex          1309 non-null   category
 5   Age          1046 non-null   float64
 6   SibSp        1309 non-null   int64
 7   Parch        1309 non-null   int64
 8   Ticket       1309 non-null   object
 9   Fare         1308 non-null   float64
 10  Cabin        295 non-null    object
 11  Embarked     1307 non-null   category
dtypes: category(4), float64(2), int64(3), object(3)
memory usage: 87.6+ KB
```

3. 조건문과 반복문

조건문

if문

- 점수를 입력받아, 점수가 60점 이상이면 '합격'을 출력합니다.

```
score = int(input('점수:'))

if score>=60:
    print('합격')
    print('축하합니다.')
print('수고하셨습니다.')
```

```
점수:50
수고하셨습니다.
```

if-else문

- 점수를 입력받아, 점수가 60점 이상이면 '합격'을 출력하고, 아니면 '불합격'을 출력합니다.

```
score = int(input('점수:'))
if score>=60:
    print('합격')
else:
    print('불합격')
```

```
점수:50
불합격
```

elif문

- 점수를 입력받고 점수의 범위에 따라 등급을 출력하는 프로그램을 작성하세요.

|등급|A|B|C|D|F|

|---|--|--|--|--|--|

|점수|90이상|80~89|70~79|60~69|0~59|

```

```

```
score = int(input('점수:'))
if score>=90:
    grade='A'
elif score>=80:
    grade='B'
elif score>=70:
    grade='C'
elif score>=60:
    grade='D'
else:
    grade='F'

print('grade:',grade)
```

```
점수:50
grade: F
```

```
score = int(input('점수:'))

if score>=80:
    grade='B'
elif score>=90:
    grade='A'
elif score>=70:
    grade='C'
elif score>=60:
    grade='D'
else:
    grade='F'

print('grade:',grade)
```

```
점수:90
grade: B
```

연습문제

- 정수를 입력받아 짝수/홀수를 판별하세요.

```
num = int(input('정수:'))

if num%2 == 1:
```

```
    print('홀수')
elif num==0:
    pirnt(0)
else:
    print('짝수')
```

```
정수:0
짝수
```

반복문

for문

```
for i in [1,2,3,4,5]:
    print(i)
```

```
1
2
3
4
5
```

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

```
p
y
t
h
o
n
```

```
for i in range(5): #0,1,2,3,4
    print(i)
```

```
0
1
2
```

```
3
4
```

```

```

- 'Hello' 10번 출력하기

특정 횟수만큼 반복하려면 range(횟수)의 형태를 사용한다.

```
for i in range(10):
    print('hello')
```

```
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

중첩반복구조

- 내부 for문을 외부 for문의 시퀀스만큼 반복한다.
- 내부루프와 외부루프는 동일한 제어변수를 사용해서는 안된다.

```
for i in range(3):
    for j in range(4):
        print(i,j)
```

```
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
```

```
2 2
2 3
```

- 2단부터 9단까지 전체 구구단 출력하기

```
for i in range(2,10):
    for j in range(1,10):
        print(f'{i}*{j}={i*j}', end='\t')
    print()
```

```
2*1=2   2*2=4   2*3=6   2*4=8   2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
3*1=3   3*2=6   3*3=9   3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
4*1=4   4*2=8   4*3=12  4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
5*1=5   5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42  6*8=48  6*9=54
7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49  7*8=56  7*9=63
8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64  8*9=72
9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

```
for i in range(1,10):
    for j in range(2,10):
        print(f'{j}*{i}={j*i}', end='\t')
    print()
```

```
2*1=2   3*1=3   4*1=4   5*1=5   6*1=6   7*1=7   8*1=8   9*1=9
2*2=4   3*2=6   4*2=8   5*2=10  6*2=12  7*2=14  8*2=16  9*2=18
2*3=6   3*3=9   4*3=12  5*3=15  6*3=18  7*3=21  8*3=24  9*3=27
2*4=8   3*4=12  4*4=16  5*4=20  6*4=24  7*4=28  8*4=32  9*4=36
2*5=10  3*5=15  4*5=20  5*5=25  6*5=30  7*5=35  8*5=40  9*5=45
2*6=12  3*6=18  4*6=24  5*6=30  6*6=36  7*6=42  8*6=48  9*6=54
2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49  8*7=56  9*7=63
2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64  9*8=72
2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

연습문제

- 1부터 10까지 정수의 합 구하여 출력하기

```
total = 0
for i in range(1,11):
    total += i
print(total)
```

```
55
```

- 1부터 100까지 홀수의 합 구하여 출력하기

```
total = 0
for i in range(1,101):
    if i%2==1:
        total+=i
print(total)
```

```
2500
```

while문

for문과 while문

1부터 5까지 출력하기

- for문

```
for i in range(1,6):
    print(i)
```

```
1
2
3
4
5
```

- while문

```
n = 1
while n<=5:
    print(n)
    n+=1
```

```
1
2
3
4
5
```

1부터 10까지 더하기

- for문

```
total=0
for i in range(1,11):
    total += i
print(total)
```

```
55
```

- while문

```
n = 1
total = 0
while n<=10:
    total += n
    n+=1
print(total)
```

```
55
```

while문만 가능한 경우

- 'q'를 입력할 때까지 반복하여 이름 입력받기

```
name = input('이름:')
while name != 'q':
    name = input('이름:')
```

```
이름:a
이름:b
```

```
이름:c
이름:d
이름:q
```

- break로 반복문 빠져나오기

```
while True:
    name = input('이름:')
    if name=='q':
        break
```

```
이름:a
이름:b
이름:c
이름:d
이름:f
이름:q
```

연습문제

로그인 프로그램

- 올바른 아이디/비밀번호를 입력할 때까지 아이디와 비밀번호를 입력하는 프로그램 만들기

```
# 올바른 아이디와 비밀번호
id = 'id123'
pwd = 'pwd123'

while True:
    input_id = input('id:')
    input_pwd = input('pwd:')

    if id==input_id and pwd==input_pwd:
        break
```

```
id:id123
pwd:adfasdfasdf
id:id123
pwd:pwd123
```

- 확장하기

아이디가 잘못되었으면 '아이디를 확인하세요' 출력

비밀번호가 잘못되었으면 '비밀번호를 확인하세요' 출력

```
# 올바른 아이디와 비밀번호
id = 'id123'
pwd = 'pwd123'

while True:
    input_id = input('id:')
    input_pwd = input('pwd:')

    if id==input_id and pwd==input_pwd:
        break
    if id!=input_id:
        print('아이디를 확인하세요')
    if pwd!=input_pwd:
        print('비밀번호를 확인하세요.')
```

```
id:id123
pwd:asdfdasf
비밀번호를 확인하세요.
id:adfasdf
pwd:adfasdfasdf
아이디를 확인하세요
비밀번호를 확인하세요.
id:id123
pwd:pwd123
```

- 사용자가 0을 입력할 때까지 숫자을 입력받아 입력받은 숫자들의 합을 구하는 프로그램을 작성하세요

```
total = 0

while True:
    num = int(input('숫자:'))
    if num==0:
        break
    total+=num
```

```
숫자:1
숫자:2
숫자:3
숫자:0
```

```
print(total)
```

```
6
```

up&down 숫자맞추기 게임

- 1~100 사이의 정답 숫자를 랜덤으로 하나 정하고, 정답 숫자를 맞출 때까지 숫자를 입력하는 게임이다.

내가 입력한 숫자가 정답보다 작으면 'DOWN', 정답보다 크면 'UP'을 출력하고 숫자를 다시 입력받는다.

정답을 맞추면 '정답!'이라고 출력하고 게임을 끝낸다.

```
#정답숫자
import random
num = random.randrange(1,101)
print(num)

#정답을 맞출 때까지 반복하기(정답을 맞추면 반복에서 벗어나기)
while True:
    answer = int(input('예상숫자:'))
    if answer==num:
        print('정답~!')
        break
    if answer<num:
        print('up')
    else:
        print('down')
```

```
14
예상숫자:14
정답~!
```

- 확장

기회는 5번까지만 주어집니다.

5회가 넘으면 '횟수초과' 메시지와 함께 정답을 알려줍니다.

정답을 맞추면 몇번째에 맞추었는지 출력합니다.

```
# 횟수
cnt = 0

#정답숫자
import random
num = random.randrange(1,101)
```

```
print(num)

#정답을 맞출 때까지 반복하기(정답을 맞추면 반복에서 벗어나기)
while True:
    cnt += 1
    if cnt>5:
        print('횟수초과:정답은',num)
        break

    answer = int(input('예상숫자:'))
    if answer==num:
        print('정답~!')
        print(cnt,'번만에 맞추었습니다.')
        break
    if answer<num:
        print('up')
    else:
        print('down')
```

```
27
예상숫자:30
down
예상숫자:20
up
예상숫자:27
정답~!
3 번만에 맞추었습니다.
```

사칙연산 프로그램

- 두 수와 사칙연산기호(+,-,*,/)을 입력받아 연산 기호에 따라 연산 결과를 출력하는 프로그램을 작성하세요.
- 사칙연산기호(+,-,*,/)가 아닌 경우 '잘못입력하셨습니다' 출력

```
num1 = int(input('숫자1:'))
num2 = int(input('숫자2:'))
op = input('연산기호:')


if op=='+':
    print(f'{num1}+{num2}={num1+num2}')
elif op=='-':
    print(f'{num1}-{num2}={num1-num2}')
elif op=='*':
    print(f'{num1}*{num2}={num1*num2}')
elif op=='/':
    print(f'{num1}/{num2}={num1/num2}')
```

```
else:
    print('잘못 입력하셨습니다.')
```

```
숫자1:1
숫자2:2
연산기호:!
잘못 입력하셨습니다.
```

할인된 금액 계산

- 물건 구매가를 입력받고, 금액에 따른 할인율을 계산하여 구매가, 할인율, 할인금액, 지불금액을 출력하세요.

|금액|할인율|

|----|------|

|1만원이상 5만원미만|5%|

|5만원이상 10만원미만|7%|

|10만원이상|10%|

```
price = int(input('물건구매가:'))

if price>=100000:
    dc = 10
elif price>=50000:
    dc = 7
elif price>=10000:
    dc=5
else:
    dc=0

print(f'''

구매가:{price}
할인율:{dc}
할인금액:{price*(dc/100)}
지불금액:{price-price*(dc/100)}

''')
```

```
물건구매가:3000


구매가:3000
```

```
할인율:0
할인금액:0.0
지불금액:3000.0
```

4. 함수, 클래스, 모듈, 패키지

함수

함수 정의하고 호출하기

- 자기소개 함수 정의하고 호출하기

```
## 자기소개 ##
print('안녕하세요!')
print('저의 이름은 파이썬입니다.')
```

```
안녕하세요!
저의 이름은 파이썬입니다.
```

```
# 함수 정의
def introduce():
    print('안녕하세요!')
    print('저의 이름은 파이썬입니다.')

# 함수 호출
introduce()
```

```
안녕하세요!
저의 이름은 파이썬입니다.
```

- 함수 정의를 먼저 한 후 호출하도록 코드를 순차적으로 작성해야 한다.

```
# 함수 호출
introduce2()

# 함수 정의
def introduce2():
    print('안녕하세요!')
    print('저의 이름은 파이썬입니다.')
```

```
# 코드는 위에서부터 아래로 순차적으로 실행된다.
# 함수를 호출하는 시점에 함수가 정의되어 있지 않으면 에러 발생
```

매개변수 전달하기

```
## 자기소개 ##
name='홍길동'
print('안녕하세요!')
print('저의 이름은, '+name+'입니다.')
```

```
안녕하세요!
저의 이름은, 홍길동입니다.
```

- 매개변수 1개 전달하기

```
def introduce(name):
    print('안녕하세요!')
    print('저의 이름은, '+name+'입니다.')

introduce('파이썬')
introduce('홍길동')
```

```
안녕하세요!
저의 이름은, 파이썬입니다.
안녕하세요!
저의 이름은, 홍길동입니다.
```

- 매개변수 여러개 전달하기

```
## 자기소개 ##
name='파이썬'
age = 20
print('안녕하세요!')
print('저의 이름은, '+name+'입니다.')
print('나이는, '+str(age)+'살입니다.')
```

```
안녕하세요!
저의 이름은, 파이썬입니다.
```

```
나이는, 20살입니다.
```

```
def introduce(name, age):
    print('안녕하세요!')
    print('저의 이름은, '+name+'입니다.')
    print('나이는, '+str(age)+'살입니다.')

introduce('파이썬',20)
introduce('홍길동',21)
```

```
안녕하세요!
저의 이름은, 파이썬입니다.
나이는, 20살입니다.
안녕하세요!
저의 이름은, 홍길동입니다.
나이는, 21살입니다.
```

값 반환하기

값 1개 반환하기

```
# 두 수를 매개변수로 받아 더한 후 결과를 리턴하는 함수 만들기
def get_plus(n1,n2):
    return n1+n2

print(get_plus(1,2))

plus = get_plus(1,2)
print('plus:',plus)
```

```
3
plus: 3
```

값 여러개 반환하기

- 여러개의 값을 하나의 튜플로 묶어서 반환한다.

```
# 두 수를 매개변수로 받아 더한값과 뺀 값을 리턴하는 함수 만들기
def get_plus_minus(n1,n2):
    return n1+n2, n1-n2

result = get_plus_minus(1,2)
```

```
result

plus,minus = get_plus_minus(1,2)
print(plus,minus)
```

```
3 -1
```

함수에서 빠져나오기

- return을 만나면 함수를 빠져나온다.
- 반환할 값이 있다면 값을 반환하고 빠져나오고, 없다면 그냥 빠져나온다.

```
# 정수를 입력받아 0, 짝수, 홀수 여부를 리턴하는 함수 만들기
def is_odd_even(n):
    if n == 0:
        result=0
    elif n%2==0:
        result='짝수'
    else:
        result='홀수'

    return result

is_odd_even(21)
```

```
'홀수'
```

```
def is_odd_even(n):
    if n == 0:
        return 0
    if n%2 == 0:
        return '짝수'
    return '홀수'

is_odd_even(23)
```

```
'홀수'
```

인수 전달 방법

위치인수

- 기본적인 인수 전달방법이다.
- 함수에 정의된 매개변수와 순서에 맞게 짝을 맞추어 인수를 전달한다.

```
def greet(name, msg):
    print('안녕',name,msg)

greet('친구','오랜만이야')
```

```
안녕 친구 오랜만이야
```

디폴트인수

- 함수를 정의할 때 매개변수에 디폴트값을 지정하면 디폴트값이 지정된 인수를 생략할 수 있다.

```
def greet(name, msg='오랜만이야'):
    print('안녕',name,msg)

greet('친구')
greet('친구', '반가워')
```

```
안녕 친구 오랜만이야
안녕 친구 반가워
```

- 디폴트 인수는 기본 위치 인수를 다 적은 다음에 적어야 한다.

```
def greet(name='친구', msg):
    print('안녕',name,msg)
```

키워드인수

- 함수를 호출할 때 인수의 이름을 명시하면, 순서를 바꾸어 전달할 수 있다.

```
def get_minus(x,y,z):
    return x-y-z
```

```
print(get_minus(5,10,15))
print(get_minus(5,z=15,y=10))
```

```
-20
-20
```

- 키워드 인수는 기본 위치 인수를 다 적은 다음에 적어야 한다.

```
def get_minus(x,y,z):
    return x-y-z

print(get_minus(5,10,15))
print(get_minus(z=15,10,x=5))
```

가변인수

- 인수를 하나의 튜플이나 리스트로 전달한다.

```
# 가변적인 수를 하나의 리스트/튜플로 받아서 수의 평균을 리턴하는 함수
def average(args):
    return sum(args)/len(args)

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

```
3.0
```

- 매개변수에 '*'를 붙이면 여러개의 인수를 하나의 튜플로 받는다.
- 인수의 갯수는 가변적이다.

```
# 가변적인 수를 받아서 수의 평균을 리턴하는 함수
def average(*args):
    # print(args)
    return sum(args)/len(args)

average(1,2,3)
```

```
2.0
```

- print함수의 위치인수, 키워드인수

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
print(1,2,3,sep='@')
print(4,5)
```

```
1@2@3
4 5
```

변수 사용하기

변수의 종류

전역변수

- 함수 밖에서 생성된 변수
- 함수 내에서도 사용할 수 있다.

```
say1 = 'hello'

def sayhello():
    print(say1)

sayhello()
```

```
hello
```

지역변수

- 함수 내에서 생성된 변수
- 함수 내에서만 사용할 수 있다.

```
def saygoodbye():
    say2 = 'goodbye' #지역변수
    print(say2)


saygoodbye()
print(say2)
```

```
goodbye
```

함수 안에서 전역변수 값 변경하기

- 함수 내에서 전역변수 값을 변경하려면 'global' 키워드를 함께 사용하여야 한다.
- global을 사용하지 않으면 동일한 이름의 지역변수가 생성되어 사용된다.

```
n1 = 1
n2 = 10

def get_plus_minus():
    n1 = 2
    plus = n1+n2
    minus = n1-n2
    return plus, minus

get_plus_minus()
n1
```

```
1
```

```
n1 = 1
n2 = 10

def get_plus_minus():
    global n1
```

```
    n1 = 2
    plus = n1+n2
    minus = n1-n2
    return plus, minus

get_plus_minus()
n1
```

```
2
```

람다 함수

- lambda 매개변수1, 매개변수2, ... : 수식
- 식 형태로 되어있어서 람다표현식(lambda experssion)이라고 부른다.
- 람다표현식은 이름이 없는 익명함수이다.

```
# 두 수를 매개변수로 받아 더한 결과를 리턴하는 함수
def get_plus(n1, n2):
    return n1+n2

get_plus(1,2)
```

```
# 람다표현식으로 만들기
lambda n1,n2:n1+n2

# 람다표현식 사용하기
(lambda n1,n2:n1+n2)(1,2)
```

```
3
```

```
# 람다표현식을 프로그램 내에서 재사용하고 싶다면, 람다표현식을 변수에 담아서 사용한다.
lambda_plus = lambda n1,n2:n1+n2

# 변수로 람다표현식 담아 호출하기
lambda_plus(1,2)
lambda_plus(3,5)
```

```
8
```

람다표현식에 조건부표현식 사용하기

- lambda 매개변수들:식1 if 조건식 else 식2

```
# 아래 리스트에서 짝수는 float로 바꾸고, 홀수는 str로 바꾸기
l1 = [1,2,3,4,5,6,7,8,9,10]

list(map((lambda x:float(x) if x%2==0 else str(x)),l1))
```

```
['1', 2.0, '3', 4.0, '5', 6.0, '7', 8.0, '9', 10.0]
```

map함수

- map은 리스트나 튜플의 각 요소를 지정된 함수로 처리해주는 함수이다.
- 원본리스트를 변경하지 않고 새 리스트를 생성한다.
- list(map(함수, 리스트))
- tuple(map(함수, 튜플))

```
# map함수를 이용하여 리스트 a의 각 요소를 정수화 하여 새로운 리스트로 만들기
a = ['1', '2', '3', '4']
b = list(map(int,a))

print(a)
print(b)
```

```
['1', '2', '3', '4']
[1, 2, 3, 4]
```

map함수에 람다표현식 사용하기

```
# 아래 리스트 l1에 1을 더한 l2 리스트 만들기
l1 = [1,2,3,4,5]
l2 = list(map((lambda x:x+1),l1))
print(l1)
print(l2)
```

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

연습문제

생일 축하 메세지 출력 함수

- 이름과 나이를 입력받아 생일축하 메시지를 출력하는 함수를 만들고 호출하세요.
- 생일문구 : ooo님의 oo번째 생일을 축하합니다!!

```
name = input('이름:')
age = input('나이:')

def happybirthday(name, age):
    print(f'{name}님의 {age}번째 생일을 축하합니다.')

happybirthday(name,age)
```

```
이름:파이썬
나이:20
파이썬님의 20번째 생일을 축하합니다.
```

소수 여부 판단하기

매개변수로 전달받은 수가 소수인지 아닌지 판별하는 함수를 작성하고 호출하세요.

- 소수란 : 1과 자기 자신으로만 나누어 떨어지는 1보다 큰 양의 정수

```
def is_prime(n):
    if n<=1:
        return False

    for i in range(2,n):
        if n%i==0:
            return False

    return True

is_prime(15)
```

```
False
```

클래스

클래스 생성하기

자동차 클래스와 객체

- 3가지 속성과 3가지 기능이 있는 자동차 객체를 찍어내기 위한 틀을 만든다.
- 속성 : brand, model, color
- 기능 : turn_on, turn_off, drive

자동차클래스 만들기

```
class Car:
    def __init__(self, b,m,c):
        self.brand = b
        self.model = m
        self.color = c

        print(self.brand, self.model, self.color, '출고')

    def turn_on(self):
        print(self.brand, '시동을 겁니다.')

    def turn_off(self):
        print(self.brand, '시동을 끕니다.')

    def drive(self):
        print(self.brand, '주행중입니다.')
```

객체 생성하기

- 객체명 = 클래스명(매개변수1,매개변수2,...)

```
car1 = Car('현대자동차','소나타','화이트')
car2 = Car('르노삼성','SM7','블랙')
```

```
현대자동차 소나타 화이트 출고
르노삼성 SM7 블랙 출고
```

메소드 호출하기

- 객체명.메소드명(매개변수1,매개변수2,...)

```
car1.turn_on()
car1.turn_off()
car1.drive()

car2.turn_on()
car2.turn_off()
car2.drive()
```

```
현대자동차 시동을 겁니다.
현대자동차 시동을 끕니다.
현대자동차 주행중입니다.
르노삼성 시동을 겁니다.
르노삼성 시동을 끕니다.
르노삼성 주행중입니다.
```

객체의 메소드 목록 조회

- dir(객체)

```
dir(list)
```

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
```

```
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

함수, 메소드 사용법

- help(함수명)

```
a = [1,2,3]
help(a.append)
```

```
Help on built-in function append:

append(object, /) method of builtins.list instance
    Append object to the end of the list.
```

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
```

모듈

모듈 사용하기

- import 모듈명
- 모듈명.함수명( )

```
import myCalc
print(myCalc.get_plus(1,2))
print(myCalc.get_minus(1,2))
print(myCalc.get_multiply(1,2))
print(myCalc.get_division(1,2))
```

```
3
-1
2
0.5
```

모듈 별칭 사용하기

- import 모듈명 as 별칭
- 별칭.함수명()

```
import myCalc as calc
print(calc.get_plus(1,2))
print(calc.get_minus(1,2))
print(calc.get_multiply(1,2))
print(calc.get_division(1,2))
```

```
3
-1
2
0.5
```

모듈 이름 붙이지 않고 함수 사용하기

- from 모듈명 import 함수명1, 함수명2,...

```
from myCalc import get_plus, get_minus
print(get_plus(1,2))
print(get_minus(1,2))
print(myCalc.get_multiply(1,2))
```

```
3
-1
2
```

```
from myCalc import *
print(get_plus(1,2))
print(get_minus(1,2))
print(get_multiply(1,2))
print(get_division(1,2))
```

```
3
-1
2
0.5
```

파이썬 내장 모듈

math

- 수학적 연산과 관련된 함수들을 모아놓은 모듈
- ceil : 올림하여 정수로 만들기
- floor : 내림하여 정수로 만들기
- sqrt : 제곱근
- factorial : 팩토리얼

- pi : 원주율

```
import math
print(math.ceil(1.4))
print(math.floor(1.7))
print(math.sqrt(4))
print(math.factorial(3))
print(math.pi)
```

```
2
1
2.0
6
3.141592653589793
```

random

- 임의의 수를 발생시키거나 리스트의 요소 중 임의의 수를 선택하는 데 사용되는 모듈

```
import random
```

랜덤 정수 구하기

- random.randint(시작값, 끝값) : 시작값~끝값 사이의 랜덤 정수 구하기 (끝값 포함)
- random.randrange(시작값, 끝값) : 시작값~끝값 사이의 랜덤 정수 구하기 (끝값 미포함)
- random.randrange(끝값) : 0~끝값 사이의 랜덤 정수 구하기 (끝값 미포함)

```
# 1~10 사이의 랜덤 정수 구하기(10포함)
print(random.randint(1,10))

# 1~9 사이의 랜덤 정수 구하기
print(random.randrange(1,10))

# 0~9 사이의 랜덤 정수 구하기
print(random.randrange(10))
```

```
6
7
1
```

랜덤 실수 구하기

- random.random() : 0~1 사이의 랜덤 실수 구하기
- random.uniform(시작값,끝값) : 시작값 ~ 끝값 사이의 랜덤 실수 반환(끝값 미포함)

```
# 0~1 사이의 랜덤 실수 구하기
print(random.random())

# 1~10 사이의 랜덤 실수 구하기
print(random.uniform(1,10))
```

```
0.9558941639754068
2.5328923420483207
```

시퀀스 데이터에서 무작위 요소 추출

- random.choice(시퀀스)

```
print(random.choice([1,2,3]))
print(random.choice('python'))
print(random.choice(range(1,101)))
```

```
3
n
46
```

시퀀스 데이터에서 무작위로 n개 요소 추출

- random.sample(시퀀스,n)

```
print(random.sample([1,2,3],2))
print(random.sample('python',2))
print(random.sample(range(1,101),2))
```

```
[1, 2]
['y', 'n']
[89, 7]
```

시퀀스 데이터를 무작위로 랜덤하게 섞기

- random.shuffle(시퀀스) : 원본을 섞는다. 리턴값이 없다.

```
a = [1,2,3,4,5]
random.shuffle(a)
a
```

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

datetime

- 날짜, 시간과 관련된 모듈.
- 날짜 형식을 만들 때 주로 사용된다.

```
import datetime
```

현재 날짜와 시각 가져오기

- datetime.datetime.now()

```
now = datetime.datetime.now()
```

현재 날짜와 시각 출력하기

```
print(now.year,'년')
print(now.month,'월')
print(now.day,'일')
print(now.hour,'시')
print(now.minute,'분')
print(now.second,'초')
```

```
2021 년
9 월
24 일
14 시
16 분
46 초
```

시간을 포멧에 맞게 출력하기

- datetime.datetime.now().strftime(포멧)

```
now.strftime('%Y.%m.%d %H:%M:%S')
```

```
'2021.09.24 14:16:46'
```

특정 시간 이후의 날짜와 시간 구하기

- datetime.datetime.now()+datetime.timedelta(더할시간) : 특정 일, 시간, 분, 초 이후의 날짜와 시간 구하기
- timedelta에는 year로 계산하는 기능은 없음

```
now + datetime.timedelta(weeks=1, days=1, hours=1, minutes=1, seconds=1)
```

```
datetime.datetime(2021, 10, 2, 15, 17, 47, 299163)
```

```
# 현재로부터 100일 이후의 날짜와 시간 구하기
now + datetime.timedelta(days=100)
```

```
datetime.datetime(2022, 1, 2, 14, 16, 46, 299163)
```

```
# 현재로부터 100일 전의 날짜와 시간 구하기
now + datetime.timedelta(days=-100)
```

```
datetime.datetime(2021, 6, 16, 14, 16, 46, 299163)
```

time

- 시간 데이터를 다루기 위한 모듈

```
import time
```

현재 날짜와 시간 가져오기

- time.localtime()
- time.ctime()

```
tm = time.localtime()
tm
```

```
time.struct_time(tm_year=2021, tm_mon=9, tm_mday=24, tm_hour=14, tm_min=24,
tm_sec=37, tm_wday=4, tm_yday=267, tm_isdst=0)
```

```
time.localtime().tm_year
time.localtime().tm_mon
time.localtime().tm_mday
time.localtime().tm_hour
time.localtime().tm_min
```

```
25
```

```
time.ctime()
```

```
'Fri Sep 24 14:25:33 2021'
```

일시정지

- time.sleep(초)

```
# 카운트다운
print(3)
time.sleep(1)
print(2)
time.sleep(1)
print(1)
time.sleep(1)
print('srart')
```

```
3
2
1
srart
```

모듈 살펴보기

모듈 내 함수 확인

- dir(모듈명)

```
# 파이썬 내장함수 확인
dir(__builtins__)
```

```
['ArithmeticError',
 'AssertionError',
 'AttributeError',
 'BaseException',
 'BlockingIOError',
 'BrokenPipeError',
 'BufferError',
 'BytesWarning',
 'ChildProcessError',
 'ConnectionAbortedError',
 'ConnectionError',
 'ConnectionRefusedError',
 'ConnectionResetError',
 'DeprecationWarning',
 'EOFError',
 'Ellipsis',
 'EnvironmentError',
 'Exception',
 'False',
 'FileExistsError',
 'FileNotFoundError',
 'FloatingPointError',
 'FutureWarning',
 'GeneratorExit',
 'IOError',
 'ImportError',
 'ImportWarning',
 'IndentationError',
 'IndexError',
 'InterruptedError',
 'IsADirectoryError',
 'KeyError',
```

```
    'KeyboardInterrupt',
    'LookupError',
    'MemoryError',
    'ModuleNotFoundError',
    'NameError',
    'None',
    'NotADirectoryError',
    'NotImplemented',
    'NotImplementedError',
    'OSError',
    'OverflowError',
    'PendingDeprecationWarning',
    'PermissionError',
    'ProcessLookupError',
    'RecursionError',
    'ReferenceError',
    'ResourceWarning',
    'RuntimeError',
    'RuntimeWarning',
    'StopAsyncIteration',
    'StopIteration',
    'SyntaxError',
    'SyntaxWarning',
    'SystemError',
    'SystemExit',
    'TabError',
    'TimeoutError',
    'True',
    'TypeError',
    'UnboundLocalError',
    'UnicodeDecodeError',
    'UnicodeEncodeError',
    'UnicodeError',
    'UnicodeTranslateError',
    'UnicodeWarning',
    'UserWarning',
    'ValueError',
    'Warning',
    'WindowsError',
    'ZeroDivisionError',
    '__IPYTHON__',
    '__build_class__',
    '__debug__',
    '__doc__',
    '__import__',
    '__loader__',
    '__name__',
    '__package__',
    '__spec__',
    'abs',
    'all',
    'any',
    'ascii',
    'bin',
```

```
    'bool',
    'breakpoint',
    'bytearray',
    'bytes',
    'callable',
    'chr',
    'classmethod',
    'compile',
    'complex',
    'copyright',
    'credits',
    'delattr',
    'dict',
    'dir',
    'display',
    'divmod',
    'enumerate',
    'eval',
    'exec',
    'filter',
    'float',
    'format',
    'frozenset',
    'get_ipython',
    'getattr',
    'globals',
    'hasattr',
    'hash',
    'help',
    'hex',
    'id',
    'input',
    'int',
    'isinstance',
    'issubclass',
    'iter',
    'len',
    'license',
    'list',
    'locals',
    'map',
    'max',
    'memoryview',
    'min',
    'next',
    'object',
    'oct',
    'open',
    'ord',
    'pow',
    'print',
    'property',
    'range',
    'repr',
```

```
 'reversed',
 'round',
 'set',
 'setattr',
 'slice',
 'sorted',
 'staticmethod',
 'str',
 'sum',
 'super',
 'tuple',
 'type',
 'vars',
 'zip']
```

```
dir(datetime)
```

```
['MAXYEAR',
 'MINYEAR',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'date',
 'datetime',
 'datetime_CAPI',
 'sys',
 'time',
 'timedelta',
 'timezone',
 'tzinfo']
```

모듈 내 함수 사용법 확인

- import 모듈명

help(모듈명.함수명)

```
import random
help(random.randint)
```

```
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

외부모듈

- 외부모듈 설치하기 : pip install 모듈명
- 설치된 외부모듈 확인하기 : pip list

```
pip list
```

```
Package                           Version
--------------------------------- -------------------
alabaster                         0.7.12
anaconda-client                   1.7.2
anaconda-navigator                2.0.3
anaconda-project                  0.9.1
anyio                             2.2.0
appdirs                           1.4.4
argh                              0.26.2
argon2-cffi                       20.1.0
asn1crypto                        1.4.0
astroid                           2.4.2
astropy                           4.2.1
async-generator                   1.10
atomicwrites                      1.4.0
attrs                             20.3.0
autopep8                          1.5.6
Babel                             2.9.0
backcall                          0.2.0
backports.functools-lru-cache     1.6.4
backports.shutil-get-terminal-size 1.0.0
backports.tempfile                1.0
backports.weakref                 1.0.post1
bcrypt                            3.2.0
beautifulsoup4                    4.9.3
bitarray                          1.9.2
bkcharts                          0.2
black                             19.10b0
bleach                            3.3.0
bokeh                             2.3.2
boto                              2.49.0
Bottleneck                        1.3.2
brotlipy                          0.7.0
certifi                           2020.12.5
```

```
cffi                              1.14.5
chardet                           4.0.0
click                             7.1.2
cloudpickle                       1.6.0
clyent                            1.2.2
colorama                          0.4.4
comtypes                          1.1.9
conda                             4.10.1
conda-build                       3.21.4
conda-content-trust               0+unknown
conda-package-handling            1.7.3
Note: you may need to restart the kernel to use updated packages.
conda-repo-cli                    1.0.4
conda-token                       0.3.0
conda-verify                      3.4.2
contextlib2                       0.6.0.post1
cryptography                      3.4.7
cycler                            0.10.0
Cython                            0.29.23
cytoolz                           0.11.0
dask                              2021.4.0
decorator                         5.0.6
defusedxml                        0.7.1
diff-match-patch                  20200713
distributed                       2021.4.0
docutils                          0.17
entrypoints                       0.3
et-xmlfile                        1.0.1
fastcache                         1.1.0
filelock                          3.0.12
flake8                            3.9.0
Flask                             1.1.2
fsspec                            0.9.0
future                            0.18.2
gevent                            21.1.2
glob2                             0.7
greenlet                          1.0.0
h5py                              2.10.0
HeapDict                          1.0.1
html5lib                          1.1
idna                              2.10
imagecodecs                       2021.3.31
imageio                           2.9.0
imagesize                         1.2.0
importlib-metadata                3.10.0
iniconfig                         1.1.1
intervaltree                      3.1.0
ipykernel                         5.3.4
ipython                           7.22.0
ipython-genutils                  0.2.0
ipywidgets                        7.6.3
isort                             5.6.4
itsdangerous                      1.1.0
jdcal                             1.4.1
```

```
jedi                                0.17.2
Jinja2                              2.11.3
joblib                              1.0.1
json5                               0.9.5
jsonschema                          3.2.0
jupyter                             1.0.0
jupyter-client                      6.1.12
jupyter-console                     6.4.0
jupyter-contrib-core                0.3.3
jupyter-contrib-nbextensions        0.5.1
jupyter-core                        4.7.1
jupyter-highlight-selected-word     0.2.0
jupyter-latex-envs                  1.4.6
jupyter-nbextensions-configurator   0.4.1
jupyter-packaging                   0.7.12
jupyter-server                      1.4.1
jupyterlab                          3.0.14
jupyterlab-pygments                 0.1.2
jupyterlab-server                   2.4.0
jupyterlab-widgets                  1.0.0
keyring                             22.3.0
kiwisolver                          1.3.1
lazy-object-proxy                   1.4.3
libarchive-c                        2.9
llvmlite                            0.36.0
locket                              0.2.1
lxml                                4.6.3
MarkupSafe                          1.1.1
matplotlib                          3.3.4
mccabe                              0.6.1
menuinst                            1.4.16
mistune                             0.8.4
mkl-fft                             1.3.0
mkl-random                          1.2.1
mkl-service                         2.3.0
mock                                4.0.3
more-itertools                      8.7.0
mpmath                              1.2.1
msgpack                             1.0.2
multipledispatch                    0.6.0
mypy-extensions                     0.4.3
navigator-updater                   0.2.1
nbclassic                           0.2.6
nbclient                            0.5.3
nbconvert                           6.0.7
nbformat                            5.1.3
nest-asyncio                        1.5.1
networkx                            2.5
nltk                                3.6.1
nose                                1.3.7
notebook                            6.3.0
numba                               0.53.1
numexpr                             2.7.3
numpy                               1.20.1
```

```
numpydoc                        1.1.0
olefile                         0.46
openpyxl                        3.0.7
packaging                       20.9
pandas                          1.2.4
pandocfilters                   1.4.3
paramiko                        2.7.2
parso                           0.7.0
partd                           1.2.0
path                            15.1.2
pathlib2                        2.3.5
pathspec                        0.7.0
patsy                           0.5.1
pep8                            1.7.1
pexpect                         4.8.0
pickleshare                     0.7.5
Pillow                          8.2.0
pip                             21.0.1
pkginfo                         1.7.0
pluggy                          0.13.1
ply                             3.11
prometheus-client               0.10.1
prompt-toolkit                  3.0.17
psutil                          5.8.0
ptyprocess                      0.7.0
py                              1.10.0
pycodestyle                     2.6.0
pycosat                         0.6.3
pycparser                       2.20
pycurl                          7.43.0.6
pydocstyle                      6.0.0
pyerfa                          1.7.3
pyflakes                        2.2.0
Pygments                        2.8.1
pylint                          2.6.0
pyls-black                      0.4.6
pyls-spyder                     0.3.2
PyNaCl                          1.4.0
pyodbc                          4.0.0-unsupported
pyOpenSSL                       20.0.1
pyparsing                       2.4.7
pyreadline                      2.1
pyrsistent                      0.17.3
PySocks                         1.7.1
pytest                          6.2.3
python-dateutil                 2.8.1
python-jsonrpc-server           0.4.0
python-language-server          0.36.2
pytz                            2021.1
PyWavelets                      1.1.1
pywin32                         227
pywin32-ctypes                  0.2.0
pywinpty                        0.5.7
PyYAML                          5.4.1
```

```
pyzmq                             20.0.0
QDarkStyle                        2.8.1
QtAwesome                         1.0.2
qtconsole                         5.0.3
QtPy                              1.9.0
regex                             2021.4.4
requests                          2.25.1
rope                              0.18.0
Rtree                             0.9.7
ruamel-yaml-conda                 0.15.100
scikit-image                      0.18.1
scikit-learn                      0.24.1
scipy                             1.6.2
seaborn                           0.11.1
Send2Trash                        1.5.0
setuptools                        52.0.0.post20210125
simplegeneric                     0.8.1
singledispatch                    0.0.0
sip                               4.19.13
six                               1.15.0
sniffio                           1.2.0
snowballstemmer                   2.1.0
sortedcollections                 2.1.0
sortedcontainers                  2.3.0
soupsieve                         2.2.1
Sphinx                            4.0.1
sphinxcontrib-applehelp           1.0.2
sphinxcontrib-devhelp             1.0.2
sphinxcontrib-htmlhelp            1.0.3
sphinxcontrib-jsmath              1.0.1
sphinxcontrib-qthelp              1.0.3
sphinxcontrib-serializinghtml     1.1.4
sphinxcontrib-websupport          1.2.4
spyder                            4.2.5
spyder-kernels                    1.10.2
SQLAlchemy                        1.4.7
statsmodels                       0.12.2
sympy                             1.8
tables                            3.6.1
tblib                             1.7.0
terminado                         0.9.4
testpath                          0.4.4
textdistance                      4.2.1
threadpoolctl                     2.1.0
three-merge                       0.1.1
tifffile                          2021.4.8
toml                              0.10.1
toolz                             0.11.1
tornado                           6.1
tqdm                              4.59.0
traitlets                         5.0.5
typed-ast                         1.4.2
typing-extensions                 3.7.4.3
ujson                             4.0.2
```

```
unicodecsv                    0.14.1
urllib3                       1.26.4
watchdog                      1.0.2
wcwidth                       0.2.5
webencodings                  0.5.1
Werkzeug                      1.0.1
wheel                         0.36.2
widgetsnbextension            3.5.1
win-inet-pton                 1.1.0
win-unicode-console           0.5
wincertstore                  0.2
wrapt                         1.12.1
xlrd                          2.0.1
XlsxWriter                    1.3.8
xlwings                       0.23.0
xlwt                          1.3.0
xmltodict                     0.12.0
yapf                          0.31.0
zict                          2.0.0
zipp                          3.4.1
zope.event                    4.5.0
zope.interface                5.3.0
```

패키지 : 모듈이 모인 것.

Part 2 | Pandas 데이터 분석

5. Pandas

데이터 구조 파악 및 인덱싱

데이터 구조

시리즈

시리즈 만들기

- pd.Series(리스트)

*시리즈 : 엑셀시트의 열 1개를 의미한다.(1차원 리스트형태)

```
# 시리즈 만들기
s = pd.Series(['amy',170,240])
s.index
s
```

```
0    amy
1    170
2    240
dtype: object
```

```
# 타입 확인하기
type(s)
```

```
pandas.core.series.Series
```

시리즈의 index와 value 가져오기

- 시리즈에는 index와 value가 있다.
- 시리즈의 index 가져오기 : **시리즈.index**
- 시리즈의 value 가져오기 : **시리즈.values**
- 시리즈의 인덱스는 리스트의 인덱스와 다른 개념이다.

시리즌의 인덱스는 데이터의 이름이고, 행번호는 따로 있다.

```
# 시리즈 인덱스 가져오기
s.index
```

```
RangeIndex(start=0, stop=3, step=1)
```

```
# 시리즈 데이터 가져오기
s.values
```

```
array(['amy', 170, 240], dtype=object)
```

시리즈의 index 지정하기

- 시리즈.index = 인덱스리스트
- 시리즈의 인덱는 숫자, 문자열 모두 가능하다.

```
# 시리즈 인덱스 지정하기
s.index = ['name','height','footsize']
```

```
# 시리즈 출력하기
s
```

```
name         amy
height       170
footsize     240
dtype: object
```

```
# 인덱스 확인하기
s.index
```

```
Index(['name', 'height', 'footsize'], dtype='object')
```

```
# 데이터 확인하기
s.values
```

```
array(['amy', 170, 240], dtype=object)
```

시리즈의 통계값 사용하기

- 평균 : 시리즈.mean()
- 최소값 : 시리즈.min()
- 최대값 : 시리즈.max()
- 중간값 : 시리즈.median()
- 표준편차 : 시리즈.std()
- 시리즈의 통계값은 시리즈의 value가 모두 숫자형일 때 사용할 수 있다.

```
s2 = pd.Series([10,20,30,40,50])
```

```
print('평균:',s2.mean())
print('최소값:',s2.min())
print('최대값:',s2.max())
print('중간값:',s2.median())
print('표준편차:',s2.std())
```

```
평균: 30.0
최소값: 10
최대값: 50
중간값: 30.0
표준편차: 15.811388300841896
```

- 요약통계 : 시리즈.describe()

```
s2.describe()
```

```
count     5.000000
mean     30.000000
std      15.811388
min      10.000000
25%      20.000000
50%      30.000000
75%      40.000000
max      50.000000
dtype: float64
```

시리즈 주요 메서드

- 값 정렬 : 시리즈.**sort_values( )**
- 인덱스 정렬 : 시리즈.**sort_index( )**
- 인덱스 리셋 : 시리즈.**reset_index( )** --> 행번호로 인덱스 재지정
- 특정 값을 가진 시리즈 값을 교체 : **replace(찾을값, 교체할값)**
- 시리즈를 데이터프레임으로 변환 : **시리즈.to_frame( )**

```
s3 = pd.Series([1,3,2,4,10])
```

```
# s3의 value 중 10을 5로 교체
s3 = s3.replace(10,5)
s3
```

```
0    1
1    3
2    2
3    4
4    5
dtype: int64
```

```
# s3 정렬 (디폴트는 오름차순 : ascending=True)
s3.sort_values()
```

```
0    1
2    2
1    3
3    4
4    5
dtype: int64
```

```
s3.sort_values(ascending=False)
```

```
4    5
3    4
1    3
2    2
0    1
dtype: int64
```

```
# s3을 데이터프레임으로 만들기
s3.to_frame()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	0
0	1
1	3
2	2
3	4
4	5

```
# 시리즈 인덱스 새로 만들기
s.reset_index()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	index	0
0	name	amy
1	height	170
2	footsize	240

데이터프레임

데이터 프레임 만들기

- 데이터프레임 : 엑셀의 시트(sheet)와 동일한 개념이다. (2차원 표 형태)

- 리스트로 만들기
    - pd.DataFrame(2차원리스트, columns=컬럼리스트, index=인덱스리스트)
    - 행 단위로 데이터프레임이 생성된다.
    - 컬럼, 인덱스를 지정하지 않으면 디폴트로 0부터 시작하는 숫자가 지정된다.

```
df = pd.DataFrame(
    [['james',30,'programmer'],
    ['amy',20,'student'],
    ['david',25,'designer']],
    columns=['name','age','job'],
    index=['a','b','c']
)

df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	age	job
a	james	30	programmer
b	amy	20	student
c	david	25	designer

- 딕셔너리로 만들기
    - pd.DataFrame(딕셔너리, index=인덱스리스트)
    - 딕셔너리는 {컬럼명1:컬럼값리스트, 컬럼명2:컬럼값리스트...}
    - 컬럼 단위로 데이터프레임이 생성된다.
    - 딕셔너리의 키가 컬럼명이 된다.

    - 인덱스를 지정하지 않으면 디폴트로 0부터 시작하는 숫자가 지정된다.

```
{'name':['james','amy','david'],
 'age':[30,20,25],
 'job':['programmer','student','designer']
}

df = pd.DataFrame(
    {'name':['james','amy','david'],
     'age':[30,20,25],
     'job':['programmer','student','designer']
    },
    index=['a','b','c']
)

df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	age	job
a	james	30	programmer
b	amy	20	student
c	david	25	designer

- csv 파일에서 데이터를 읽어와 만들기
    - pd.read_csv(파일경로)
    - csv파일은 utf-8 형식이어야 한다.

```
df = pd.read_csv('data/scores.csv')
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	60.0
5	Ian	90.0	100.0	90.0
6	James	70.0	75.0	65.0

데이터 확인하기

데이터 미리보기

```
import pandas as pd
df = pd.read_csv('data/scores.csv')
```

가장 앞의 n행 보기

- 데이터프레임.head(n)
- 시리즈.head(n)
- n을 생략하면 5개의 행을 출력한다

```
df.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0

가장 뒤의 n행 보기

- 데이터프레임.tail(n)
- n을 생략하면 5개의 행을 출력한다

```
df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
25	Vanessa	95.0	70.0	95.0
26	Viviana	100.0	80.0	100.0
27	Vikkie	NaN	50.0	100.0
28	Winnie	70.0	100.0	70.0
29	Zuly	80.0	90.0	95.0

랜덤 보기

랜덤 n개 데이터 보기

- 데이터프레임.sample(n)
- n을 생략하면 1개의 샘플을 출력한다.

```
df.sample()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
13	Amy	90.0	75.0	90.0

랜덤 샘플 비율로 보기

- 데이터프레임.sample(frac=0.2)
- 지정한 비율의 샘플을 출력한다

```
df.sample(frac=0.2)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
19	Kate	50.0	NaN	50.0
12	Peter	100.0	95.0	100.0
15	Danna	100.0	100.0	100.0
23	Sofia	100.0	100.0	100.0
6	James	70.0	75.0	65.0
11	Oliver	70.0	75.0	65.0

높은 순/낮은순 보기

높은순 보기

- 데이터프레임.nlargest(갯수,컬럼명)
- 컬럼의 데이터가 숫자형일 때 사용할 수 있다.

```
# eng 높은 순으로 5개 보기
df.nlargest(5,'eng')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
5	Ian	90.0	100.0	90.0
9	Kevin	100.0	100.0	90.0
14	Chloe	95.0	100.0	95.0

낮은순 보기

- 데이터프레임.nsmallest(갯수,컬럼명)
- 컬럼의 데이터가 숫자형일 때 사용할 수 있다.

```
# eng 낮은 순으로 5개 보기
df.nsmallest(5,'eng')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
4	Henry	NaN	35.0	60.0
27	Vikkie	NaN	50.0	100.0
18	Jennifer	80.0	55.0	80.0
8	Justin	50.0	60.0	100.0
16	Ellen	NaN	60.0	NaN

데이터 요약 보기

(행,열)의 크기 보기

- 데이터프레임.shape

```
df.shape
```

```
(30, 4)
```

데이터의 갯수 보기

- len(데이터프레임)

```
len(df)
```

```
30
```

컬럼명 보기

- 데이터프레임.columns
- 데이터프레임의 열 이름을 확인한다.

```
df.columns
```

```
Index(['name', 'kor', 'eng', 'math'], dtype='object')
```

인덱스 보기

- 데이터프레임.index
- 데이터프레임의 인덱스를 확인한다.

```
df.index
```

```
RangeIndex(start=0, stop=30, step=1)
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	60.0
5	Ian	90.0	100.0	90.0

데이터의 자료형 보기

- 데이터프레임.dtypes
- 데이터프레임을 구성하는 값의 자료형을 확인한다.
- 판다스에서는 문자열의 데이터타입이 object이다.

```
df.dtypes
```

```
name     object
kor     float64
eng     float64
math    float64
dtype: object
```

데이터프레임 정보 보기

- 데이터프레임.info()
- 데이터프레임의 총 샘플 갯수, 컬럼 수, 컬럼 별 정보 등을 확인한다.

```
df.info()
```

```
RangeIndex: 30 entries, 0 to 29
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   name    30 non-null     object
 1   kor     27 non-null     float64
 2   eng     28 non-null     float64
 3   math    29 non-null     float64
```

```
dtypes: float64(3), object(1)
memory usage: 1.1+ KB
```

컬럼의 유니크한 데이터 뽑기

- 컬럼.unique()

```
df['kor'].unique()
```

```
array([100.,  90.,  95.,  nan,  70.,  80.,  50.])
```

컬럼의 유니크한 값의 갯수 보기

- 컬럼.**value_counts()**

```
df['kor'].value_counts()
```

```
100.0    8
90.0     6
70.0     5
80.0     3
95.0     3
50.0     2
Name: kor, dtype: int64
```

요약통계 보기

```
df.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	kor	eng	math
count	27.000000	28.000000	29.000000
mean	85.740741	80.892857	84.827586
std	15.045042	18.361270	15.950972
min	50.000000	35.000000	50.000000
25%	75.000000	68.750000	70.000000
50%	90.000000	85.000000	90.000000

```
df['kor'].mean()
```

```
85.74074074074075
```

데이터프레임 인덱싱/슬라이싱

loc( )

컬럼명으로 데이터 추출하기

```
import pandas as pd
df = pd.read_csv('data/scores.csv')
df.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0

시리즈 형태로 추출하기

- **데이터프레임명['컬럼명'] / 데이터프레임명["컬럼명"]**

: 컬럼명은 1개만 지정할 수 있으며 대괄호 1개를 사용한다.

- **데이터프레임명.컬럼명**

: 컬럼명에 공백이나 특수문자가 섞여있을 때는 사용할 수 없다.

```
# 'name'컬럼 추출하기
s_name = df['name']
s_name.head(3)
```

```
0      Aiden
1    Charles
2     Danial
Name: name, dtype: object
```

```
# type
type(s_name)
```

```
pandas.core.series.Series
```

```
# index
s_name.index
```

```
RangeIndex(start=0, stop=30, step=1)
```

```
# values
s_name.values
```

```
array(['Aiden', 'Charles', 'Danial', 'Evan', 'Henry', 'Ian', 'James',
       'Julian', 'Justin', 'Kevin', 'Leo', 'Oliver', 'Peter', 'Amy',
       'Chloe', 'Danna', 'Ellen', 'Emma', 'Jennifer', 'Kate', 'Linda',
       'Olivia', 'Rose', 'Sofia', 'Tiffany', 'Vanessa', 'Viviana',
       'Vikkie', 'Winnie', 'Zuly'], dtype=object)
```

```
# shape
s_name.shape
```

```
(30,)
```

```
# 'eng'컬럼 추출하기
df.eng
```

```
0      90.0
1      80.0
2     100.0
3     100.0
4      35.0
5     100.0
6      75.0
7      90.0
8      60.0
9     100.0
10     95.0
11     75.0
12     95.0
13     75.0
14    100.0
15    100.0
16     60.0
17     65.0
18     55.0
19      NaN
20     90.0
21     70.0
22     65.0
23    100.0
24      NaN
25     70.0
26     80.0
27     50.0
28    100.0
```

```
29     90.0
Name: eng, dtype: float64
```

```
# 'matn'컬럼 추출하기
df["math"].head(3)
```

```
0     95.0
1     75.0
2    100.0
Name: math, dtype: float64
```

데이터프레임 형태로 추출하기

- 데이터프레임명[**컬럼명리스트**]
- 대괄호안에 컬럼리스트가 들어간다.(**대괄호 2개**로 표현되어야한다.)

```
# 'name','kor' 컬럼 데이터 추출하기
df_name_kor = df[['name','kor']]
df_name_kor.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor
0	Aiden	100.0
1	Charles	90.0
2	Danial	95.0

```
# 'math' 컬럼을 데이터프레임 형태로 추출하기
df_math = df[['math']]
df_math
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	math
0	95.0
1	75.0
2	100.0
3	100.0
4	60.0
5	90.0
6	65.0

```
# type
type(df_math)
```

```
pandas.core.frame.DataFrame
```

조건에 따라 데이터 추출하기

- 조건의 결과에 따른 불린인덱스 추출
    - 조건식의 결과에 따른 결과가 불린인덱스로 만들어진다.

```
# kor 점수가 100점인 데이터 불린인덱스
df['kor']==100
```

```
0       True
1      False
```

```
2     False
3      True
4     False
5     False
6     False
7     False
8     False
9      True
10    False
11    False
12     True
13    False
14    False
15     True
16    False
17    False
18    False
19    False
20     True
21    False
22    False
23     True
24    False
25    False
26     True
27    False
28    False
29    False
Name: kor, dtype: bool
```

- 불린인덱스로 데이터 추출
    - 불린인덱스를 **데이터프레임명[]**으로 감싸주면 True인 데이터만 추출된다.

```
# kor 점수가 100점인 데이터 추출
df[df['kor']==100]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0

	name	kor	eng	math
3	Evan	100.0	100.0	100.0
9	Kevin	100.0	100.0	90.0
12	Peter	100.0	95.0	100.0
15	Danna	100.0	100.0	100.0
20	Linda	100.0	90.0	100.0

- 여러 조건
    - 논리연산자는 **'&' , '|' ,'~', '^'** 기호를 사용한다.
    - 논리연산자를 사용할 때에는 **각 조건을 ()로 감싼다.**

```
# 한 과목이라도 100을 받은 학생 추출
df[(df.kor==100)|(df.eng==100)|(df.math==100)]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
5	Ian	90.0	100.0	90.0
8	Justin	50.0	60.0	100.0
9	Kevin	100.0	100.0	90.0
12	Peter	100.0	95.0	100.0

```
# kor의 값이 60~90인 학생의 name, kor 추출

df[(df['kor']>=60)&(df['kor']<=90)][['name','kor']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor
1	Charles	90.0
5	Ian	90.0
6	James	70.0
7	Julian	80.0
10	Leo	90.0
11	Oliver	70.0
13	Amy	90.0

- 특정 값을 가진 데이터만 추출
    - 컬럼.isin(값리스트)

```
# 이름이 Amy인 데이터 추출
df[df['name'].isin(['Amy'])]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
13	Amy	90.0	75.0	90.0

```
# 이름이 Amy, Rose인 데이터 추출
df[df['name'].isin(['Amy','Rose'])]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
13	Amy	90.0	75.0	90.0
22	Rose	70.0	65.0	70.0

```
# kor이 50,100 데이터 추출
df[df['kor'].isin([50,100])]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
3	Evan	100.0	100.0	100.0

	name	kor	eng	math
8	Justin	50.0	60.0	100.0
9	Kevin	100.0	100.0	90.0
12	Peter	100.0	95.0	100.0

- null 여부에 따른 데이터 추출
    - 컬럼.isnull() --> 해당 컬럼의 값이 null인 데이터 추출
    - 컬럼.notnull() --> 해당 컬럼의 값이 null이 아닌 데이터 추출

```
# kor이 null인 데이터 추출
df[df.kor.isnull()]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
4	Henry	NaN	35.0	60.0
16	Ellen	NaN	60.0	NaN
27	Vikkie	NaN	50.0	100.0

```
# kor이 null이 아닌 데이터 추출
df[df.kor.notnull()]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
5	Ian	90.0	100.0	90.0
6	James	70.0	75.0	65.0
7	Julian	80.0	90.0	55.0

인덱스로 행 데이터 추출하기

- 데이터프레임명.**loc[인덱스]**

: 시리즈 형태로 추출한다. 하나의 인덱스만 사용 가능하다.

- 데이터프레임명.**loc[인덱스리스트]**

: 데이터프레임 형태로 추출한다. 한개 이상의 인덱스를 사용할 수 있다.

```
import pandas as pd
df = pd.read_csv('data/scores.csv')
df.index = 'i'+df.index.astype('str')
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i0	Aiden	100.0	90.0	95.0
i1	Charles	90.0	80.0	75.0
i2	Danial	95.0	100.0	100.0
i3	Evan	100.0	100.0	100.0

	name	kor	eng	math
i4	Henry	NaN	35.0	60.0

- 시리즈 형태로 추출하기

```
# 인덱스가 i3인 행 추출하여 s1에 담기
s1=df.loc['i3']
```

```
# s1 type
type(s1)
```

```
pandas.core.series.Series
```

```
# s1 index
s1.index
```

```
Index(['name', 'kor', 'eng', 'math'], dtype='object')
```

```
# s1 value
s1.values
```

```
array(['Evan', 100.0, 100.0, 100.0], dtype=object)
```

- 데이터프레임 형태로 추출하기

```
# 인덱스가 i1,i3,i5인 행 추출하기
df.loc[['i1','i3','i5']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	name	kor	eng	math
i1	Charles	90.0	80.0	75.0
i3	Evan	100.0	100.0	100.0
i5	Ian	90.0	100.0	90.0

```
# 인덱스가 i3인 행을 데이터프레임 형태로 추출하기
df.loc[['i3']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i3	Evan	100.0	100.0	100.0

```
# 인덱스에 없는 값을 사용하면 error
df.loc[-1]
```

인덱스로 행,열 추출하기

- 인덱스, 컬럼에 해당하는 한개의 데이터 뽑아오기
    - **loc[인덱스, 컬럼명]**

```
df.head(6)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i0	Aiden	100.0	90.0	95.0
i1	Charles	90.0	80.0	75.0
i2	Danial	95.0	100.0	100.0
i3	Evan	100.0	100.0	100.0
i4	Henry	NaN	35.0	60.0
i5	Ian	90.0	100.0	90.0

```
# 인덱스 i1의 kor점수
df.loc['i1','kor']
```

```
90.0
```

```
# type
type(df.loc['i1','kor'])
```

```
numpy.float64
```

- 특정 인덱스의 여러 컬럼 데이터 추출하기
    - **loc[인덱스, 컬럼명리스트]**

```
# 인덱스 i1 name, kor
df.loc['i1',['name','kor']]
```

```
name    Charles
kor        90.0
Name: i1, dtype: object
```

```
# type
type(df.loc['i1',['name','kor']])
```

```
pandas.core.series.Series
```

- 여러 인덱스의 특정 컬럼 데이터 추출하기
    - **loc[인덱스리스트,컬럼명]**

```
# 인덱스 i1,i3,i5의 name
df.loc[['i1','i3','i5'],'name']
```

```
i1    Charles
i3       Evan
i5        Ian
Name: name, dtype: object
```

```
# type
type(df.loc[['i1','i3','i5'],'name'])
```

```
pandas.core.series.Series
```

- 여러 인덱스의 여러 컬럼 가져오기
    - **loc[인덱스리스트, 컬럼명리스트]**

```
# 인덱스 i1,i3,i5의 name, kor
df.loc[['i1','i3','i5'],['name', 'kor']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor
i1	Charles	90.0
i3	Evan	100.0
i5	Ian	90.0

- 모든 행에서 특정 열 가져오기
    - loc[:,컬럼명]
    - loc[:,컬럼명리스트]

```
# 모든 행에서 'name' 가져오기
df.loc[:,'name']
```

```
i0        Aiden
i1      Charles
i2       Danial
i3         Evan
i4        Henry
i5          Ian
i6        James
i7       Julian
i8       Justin
i9        Kevin
i10         Leo
i11      Oliver
i12       Peter
i13         Amy
i14       Chloe
i15       Danna
```

```
i16       Ellen
i17        Emma
i18    Jennifer
i19        Kate
i20       Linda
i21      Olivia
i22        Rose
i23       Sofia
i24     Tiffany
i25     Vanessa
i26     Viviana
i27      Vikkie
i28      Winnie
i29        Zuly
Name: name, dtype: object
```

```
# 모든 행에서 'name','math' 가져오기
df.loc[:,['name','math']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	math
i0	Aiden	95.0
i1	Charles	75.0
i2	Danial	100.0
i3	Evan	100.0
i4	Henry	60.0
i5	Ian	90.0
i6	James	65.0

```
# 모든 행에서 'name' 가져오기(데이터프레임)
df.loc[:,['name']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name
i0	Aiden
i1	Charles
i2	Danial
i3	Evan
i4	Henry
i5	Ian
i6	James

iloc( )

- 데이터프레임명.**iloc[행번호]**
- 데이터프레임명.**iloc[행번호리스트]**
- 데이터프레임명.**iloc[행번호슬라이스]**
- **음수**를 사용하면 행번호를 뒤에서부터 센다.

```
import pandas as pd
df = pd.read_csv('data/scores.csv')
#df.index = range(10,310,10)
df.index = 'i'+df.index.astype('str')
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i0	Aiden	100.0	90.0	95.0
i1	Charles	90.0	80.0	75.0
i2	Danial	95.0	100.0	100.0
i3	Evan	100.0	100.0	100.0
i4	Henry	NaN	35.0	60.0

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i0	Aiden	100.0	90.0	95.0
i1	Charles	90.0	80.0	75.0
i2	Danial	95.0	100.0	100.0
i3	Evan	100.0	100.0	100.0
i4	Henry	NaN	35.0	60.0

시리즈 형태로 추출하기

```
# 첫번째 행 추출하기
df.iloc[0]
```

```
name     Aiden
kor      100.0
eng       90.0
```

```
math      95.0
Name: i0, dtype: object
```

데이터프레임 형태로 추출하기

```
# 1,3,5번 행 추출하기
df.iloc[[1,3,5]]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i1	Charles	90.0	80.0	75.0
i3	Evan	100.0	100.0	100.0
i5	Ian	90.0	100.0	90.0

```
# 첫번째 행 추출하기(데이터프레임)
df.iloc[[0]]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math

	name	kor	eng	math
i0	Aiden	100.0	90.0	95.0

- 행번호 슬라이스로 추출하기

```
# 1~3행 추출하기
df.iloc[1:4]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i1	Charles	90.0	80.0	75.0
i2	Danial	95.0	100.0	100.0
i3	Evan	100.0	100.0	100.0

```
# 1,3,5행 슬라이스
df.iloc[1:6:2]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	name	kor	eng	math
i1	Charles	90.0	80.0	75.0
i3	Evan	100.0	100.0	100.0
i5	Ian	90.0	100.0	90.0

```
# 1행 추출하기
df.iloc[1:2]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i1	Charles	90.0	80.0	75.0

```
# 짝수 행번호의 데이터 추출하기
df.iloc[::2]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i0	Aiden	100.0	90.0	95.0
i2	Danial	95.0	100.0	100.0
i4	Henry	NaN	35.0	60.0
i6	James	70.0	75.0	65.0
i8	Justin	50.0	60.0	100.0
i10	Leo	90.0	95.0	70.0
i12	Peter	100.0	95.0	100.0

```
# 홀수 행번호의 데이터 추출하기
df.iloc[1::2]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i1	Charles	90.0	80.0	75.0
i3	Evan	100.0	100.0	100.0
i5	Ian	90.0	100.0	90.0
i7	Julian	80.0	90.0	55.0
i9	Kevin	100.0	100.0	90.0
i11	Oliver	70.0	75.0	65.0
i13	Amy	90.0	75.0	90.0

- 음수 행번호로 추출하기

```
# 마지막행 추출하기
df.iloc[-1]
```

```
name    Zuly
kor     80.0
eng     90.0
math    95.0
Name: i29, dtype: object
```

```
# 마지막 2개 행 추출하기(리스트)
df.iloc[[-2,-1]]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i28	Winnie	70.0	100.0	70.0
i29	Zuly	80.0	90.0	95.0

```
# 마지막 2개 행 추출하기(슬라이스)
df.iloc[-2:]
```

```
.dataframe tbody tr th {
    vertical-align: top;
```

```
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i28	Winnie	70.0	100.0	70.0
i29	Zuly	80.0	90.0	95.0

행번호로 행,열 추출하기

- iloc[행번호,열번호]

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i0	Aiden	100.0	90.0	95.0
i1	Charles	90.0	80.0	75.0
i2	Danial	95.0	100.0	100.0
i3	Evan	100.0	100.0	100.0
i4	Henry	NaN	35.0	60.0

```
# 0번째 행, 0번째 열
df.iloc[0,0]
```

```
'Aiden'
```

- iloc[행번호,열번호리스트]

```
# 0번째 행 kor, eng
df.iloc[0,[1,2]]
```

```
kor    100.0
eng     90.0
Name: i0, dtype: object
```

- iloc[행번호리스트,열번호]

```
# 1,3,4번째 행 kor
df.iloc[[1,3,4],1]
```

```
i1     90.0
i3    100.0
i4      NaN
Name: kor, dtype: float64
```

- iloc[행번호리스트,열번호리스트]

```
# 1,3,4번째 행 name, eng
df.iloc[[1,3,4],[0,2]]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	eng
i1	Charles	80.0
i3	Evan	100.0
i4	Henry	35.0

- iloc[행번호슬라이싱, 열번호슬라이싱]
    - iloc에서는 슬라이싱을 사용할 수 있다.
    - iloc에서는 음수를 사용할 수 있다.

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i0	Aiden	100.0	90.0	95.0
i1	Charles	90.0	80.0	75.0
i2	Danial	95.0	100.0	100.0
i3	Evan	100.0	100.0	100.0
i4	Henry	NaN	35.0	60.0

```
# 0,1번째 행 0,1번째 열 슬라이싱
df.iloc[:2, :2]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor
i0	Aiden	100.0
i1	Charles	90.0

```
# 1,3,5번째 행 0,2번째 열 슬라이싱
df.iloc[1:6:2, :3:2]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	eng
i1	Charles	80.0
i3	Evan	100.0
i5	Ian	100.0

```
df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
i25	Vanessa	95.0	70.0	95.0
i26	Viviana	100.0	80.0	100.0
i27	Vikkie	NaN	50.0	100.0
i28	Winnie	70.0	100.0	70.0
i29	Zuly	80.0	90.0	95.0

```
# 마지막행 1,3열 슬라이싱
df.iloc[-1,1:4:2]
```

```
kor     80.0
math    95.0
Name: i29, dtype: object
```

```
# 모든행, 1열
df.iloc[:,1]
```

```
i0     100.0
i1      90.0
i2      95.0
i3     100.0
i4       NaN
i5      90.0
i6      70.0
i7      80.0
```

```
i8      50.0
i9     100.0
i10     90.0
i11     70.0
i12    100.0
i13     90.0
i14     95.0
i15    100.0
i16      NaN
i17     70.0
i18     80.0
i19     50.0
i20    100.0
i21     90.0
i22     70.0
i23    100.0
i24     90.0
i25     95.0
i26    100.0
i27      NaN
i28     70.0
i29     80.0
Name: kor, dtype: float64
```

```
# 모든행, 1,2열
df.iloc[:,[1,2]]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	kor	eng
i0	100.0	90.0
i1	90.0	80.0
i2	95.0	100.0
i3	100.0	100.0
i4	NaN	35.0
i5	90.0	100.0
i6	70.0	75.0

데이터 변경

열 단위 변경하기

열 추가/수정하기

- **데이터프레임[컬럼] = 추가할데이터**
- **데이터프레임[컬럼] = 수정할데이터**

컬럼이 존재하면 추가, 존재하지 않으면 수정된다.

```
import pandas as pd
df = pd.read_csv('data/scores.csv')
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	60.0

열 추가하기

```
len(df)
```

```
30
```

```
# 학생 번호 추가하기 (1부터 시작하여 1씩 증가)
df['no'] = range(1,len(df)+1)
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math	no
0	Aiden	100.0	90.0	95.0	1
1	Charles	90.0	80.0	75.0	2
2	Danial	95.0	100.0	100.0	3
3	Evan	100.0	100.0	100.0	4
4	Henry	NaN	35.0	60.0	5
5	Ian	90.0	100.0	90.0	6
6	James	70.0	75.0	65.0	7

```
# sum 추가
df['sum'] = df['kor']+df['eng']+df['math']
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math	no	sum
0	Aiden	100.0	90.0	95.0	1	285.0
1	Charles	90.0	80.0	75.0	2	245.0
2	Danial	95.0	100.0	100.0	3	295.0
3	Evan	100.0	100.0	100.0	4	300.0
4	Henry	NaN	35.0	60.0	5	NaN
5	Ian	90.0	100.0	90.0	6	280.0

열 수정하기

```
# 학생 번호 수정하기 (100부터 시작하여 1씩 증가)
df['no']  = df['no']+99
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math	no	sum
0	Aiden	100.0	90.0	95.0	100	285.0
1	Charles	90.0	80.0	75.0	101	245.0
2	Danial	95.0	100.0	100.0	102	295.0
3	Evan	100.0	100.0	100.0	103	300.0
4	Henry	NaN	35.0	60.0	104	NaN
5	Ian	90.0	100.0	90.0	105	280.0
6	James	70.0	75.0	65.0	106	210.0

열 삭제하기

- 데이터프레임.**drop**(columns=삭제할컬럼리스트, inplace=True)
- 존재하지 않는 열은 삭제할 수 없다.

```
# no, sum 컬럼 삭제하기
df.drop(columns=['no','sum'], inplace=True)
#df = df.drop(columns=['no','sum'])
```

```
df.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0

컬럼명 바꾸기

컬럼명 한번에 바꾸기

- 데이터프레임.**columns** = 컬럼명리스트

  컬럼명리스트의 항목 수는 컬럼 수와 동일해야한다.

```
df.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0

```
#'이름','국어','영어','수학'
df.columns=['이름','국어','영어','수학']
```

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	이름	국어	영어	수학
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	60.0

```

```

```
# 전체컬럼수와 컬럼명 리스트의 항목 수가 다르면 error
df.columns=['이름','국어','영어']
```

특정 컬럼명 바꾸기

- 데이터프레임.**rename**(columns={'현재컬럼명1':'바꿀컬럼명1','현재컬럼명2':'바꿀컬럼명2',...})

```
# 이름-->성명
df = df.rename(columns={'이름':'성명'})
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	성명	국어	영어	수학
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	60.0
5	Ian	90.0	100.0	90.0
6	James	70.0	75.0	65.0

행 단위 변경하기

마지막에 행 추가하고 인덱스 다시 지정하기

- 데이터프레임.**append**(추가할데이터, **ignore_index=True**)
- 추가할 데이터는 딕셔너리 형태로 전달 : **{컬럼1:값1, 컬럼2:값2,...}**
- 데이터프레임의 끝에 행 추가
- 기존 인덱스는 무시하고, 인덱스가 새롭게 생성된다.

```
import pandas as pd
df = pd.read_csv('data/scores.csv')
df.index = range(100,3100,100)
df.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
100	Aiden	100.0	90.0	95.0
200	Charles	90.0	80.0	75.0
300	Danial	95.0	100.0	100.0

```
new_value = {'name':'Python','kor':80,'eng':90,'math':100}
df = df.append(new_value,  ignore_index=True)
```

```
df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math

	name	kor	eng	math
26	Viviana	100.0	80.0	100.0
27	Vikkie	NaN	50.0	100.0
28	Winnie	70.0	100.0	70.0
29	Zuly	80.0	90.0	95.0
30	Python	80.0	90.0	100.0

인덱스 지정하여 추가/수정하기

- 데이터프레임.**loc[인덱스] = 추가할데이터**
- 데이터프레임.**loc[인덱스] = 수정할데이터**
- 인덱스가 존재하면 해당 인덱스의 데이터가 수정된다.
- 인덱스가 존재하지 않으면 데이터프레임의 끝에 데이터가 추가된다.

```
# 인덱스 35에 추가
df.loc[35] = ['aaa',70,80,90]
```

```
# 인덱스 34에 추가
df.loc[34] = ['bbb',80,90,100]
```

```
df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
28	Winnie	70.0	100.0	70.0
29	Zuly	80.0	90.0	95.0
30	Python	80.0	90.0	100.0

	name	kor	eng	math
35	aaa	70.0	80.0	90.0
34	bbb	80.0	90.0	100.0

```
# 인덱스 30
df.loc[30] = ['ccc',60,70,80]
```

```
df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
28	Winnie	70.0	100.0	70.0
29	Zuly	80.0	90.0	95.0
30	ccc	60.0	70.0	80.0
35	aaa	70.0	80.0	90.0
34	bbb	80.0	90.0	100.0

행 삭제하기

- 데이터프레임.**drop**(index=[삭제할인덱스리스트], inplace=True)

```
# 30,34,35 삭제
df.drop(index=[30,34,35], inplace=True)
```

```
df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
25	Vanessa	95.0	70.0	95.0
26	Viviana	100.0	80.0	100.0
27	Vikkie	NaN	50.0	100.0
28	Winnie	70.0	100.0	70.0
29	Zuly	80.0	90.0	95.0

인덱스 변경하기

전체 인덱스명 변경하기

- 데이터프레임.**index** = 인덱스명리스트

  인덱스명리스트의 항목 수는 인덱스 수와 동일해야한다.

```
# range(100,3100,100)
df.index = range(100,3100,100)
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
100	Aiden	100.0	90.0	95.0
200	Charles	90.0	80.0	75.0
300	Danial	95.0	100.0	100.0
400	Evan	100.0	100.0	100.0
500	Henry	NaN	35.0	60.0
600	Ian	90.0	100.0	90.0

특정 인덱스명 변경하기

- 데이터프레임.rename(index={'현재인덱스명1':'바꿀인덱스명1','현재인덱스명2':'바꿀인덱스명2',...})

```
# 100-->'a', 200-->'b'
df.rename(index={100:'a',200:'b'}, inplace=True)
```

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
a	Aiden	100.0	90.0	95.0
b	Charles	90.0	80.0	75.0
300	Danial	95.0	100.0	100.0
400	Evan	100.0	100.0	100.0
500	Henry	NaN	35.0	60.0

apply

데이터 준비하기

```
df = pd.read_csv('data/scores.csv')
df = df.head()
df_copy = df.copy()
```

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	60.0

```
df_copy.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0

	name	kor	eng	math
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	60.0

함수로 컬럼의 데이터 변경하기

- 컬럼.**apply(함수명)**
- 컬럼.**apply(함수명, 매개변수명=매개변수값)**

매개변수명을 명시해주어야 한다.

- 적용할 함수가 미리 정의되어 있어야 한다.

```
#df['math']의 모든점수에 5점 더하기.(--> 105점?)
df_copy.math = df.math+5
df_copy
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	100.0
1	Charles	90.0	80.0	80.0
2	Danial	95.0	100.0	105.0
3	Evan	100.0	100.0	105.0
4	Henry	NaN	35.0	65.0

```
#df['math']의 모든점수에 5점 더하기. 100점이 넘을 수 없다.
def plus5(x):
    score = x+5
    if score>=100:
```

```
        score=100
    return score

df_copy['math'] = df['math'].apply(plus5)
df_copy
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	100.0
1	Charles	90.0	80.0	80.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	65.0

```
# 모든 점수에 n점 더하기. 100점이 넘을 수 없다.
def plusn(x,n):
    score = x+n
    if score>=100:
        score=100
    return score

# df['eng']의 모든 점수에 1점 더하기. 100점이 넘을 수 없다.
df_copy['eng'] = df['eng'].apply(plusn, n=1)
```

```
df_copy
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	91.0	100.0
1	Charles	90.0	81.0	80.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	36.0	65.0

함수로 행/열의 데이터 집계하기

- 데이터프레임.apply(함수명, axis=0) : 열단위로 함수가 적용된다.
- 데이터프레임.apply(함수명, axis=1) : 행단위로 함수가 적용된다.

```
df = pd.read_csv('data/scores.csv')
df = df.head()
df.index = df.name
df.drop(columns=['name'], inplace=True)
df_copy = df.copy()
```

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	kor	eng	math
name			
Aiden	100.0	90.0	95.0
Charles	90.0	80.0	75.0

	kor	eng	math
name			
Danial	95.0	100.0	100.0

```
df_copy.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	kor	eng	math
name			
Aiden	100.0	90.0	95.0
Charles	90.0	80.0	75.0
Danial	95.0	100.0	100.0
Evan	100.0	100.0	100.0
Henry	NaN	35.0	60.0

```
# 행단위 데이터 출력
def print_me(x):
    print(x)

df.apply(print_me, axis=1)
```

```
kor     100.0
eng      90.0
math     95.0
Name: Aiden, dtype: float64
kor     90.0
eng     80.0
math    75.0
Name: Charles, dtype: float64
kor      95.0
eng     100.0
```

```
math    100.0
Name: Danial, dtype: float64
kor     100.0
eng     100.0
math    100.0
Name: Evan, dtype: float64
kor      NaN
eng     35.0
math    60.0
Name: Henry, dtype: float64
```

```
name
Aiden      None
Charles    None
Danial     None
Evan       None
Henry      None
dtype: object
```

```
# 합계 구하기
def get_sum(x):
    return x.sum()
```

```
# 학생 별 점수 합계
df_copy['sum'] = df.apply(get_sum, axis=1)
```

```
df_copy
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	kor	eng	math	sum
name				
Aiden	100.0	90.0	95.0	285.0

	kor	eng	math	sum
name				
Charles	90.0	80.0	75.0	245.0
Danial	95.0	100.0	100.0	295.0

```
# 과목 별 점수 합계
df_copy.loc['sum'] = df.apply(get_sum, axis=0)
```

```
df_copy
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	kor	eng	math	sum
name				
Aiden	100.0	90.0	95.0	285.0
Charles	90.0	80.0	75.0	245.0
Danial	95.0	100.0	100.0	295.0
Evan	100.0	100.0	100.0	300.0
Henry	NaN	35.0	60.0	95.0
sum	385.0	405.0	430.0	NaN

행/열 형태 변경

melt

행을 열로 보내기(melt)

- 데이터프레임.**melt()**
- pd.melt(데이터프레임)

```
import pandas as pd
df = pd.read_csv('data/scores.csv')
df = df.head(2)
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0

모든 열 melt

- df.melt( )
- pd.melt(df)

```
df.melt()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	variable	value

	variable	value
0	name	Aiden
1	name	Charles
2	kor	100.0
3	kor	90.0
4	eng	90.0

고정할 컬럼 지정하여 melt

- **id_vars=[열이름리스트]** --> 위치를 그대로 유지할 열 이름

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0

```
# name 고정
df.melt(id_vars='name')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	name	variable	value
0	Aiden	kor	100.0
1	Charles	kor	90.0
2	Aiden	eng	90.0
3	Charles	eng	80.0
4	Aiden	math	95.0
5	Charles	math	75.0

```
# name, kor 고정
df.melt(id_vars=['name','kor'])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	kor	variable	value
0	Aiden	100.0	eng	90.0
1	Charles	90.0	eng	80.0
2	Aiden	100.0	math	95.0
3	Charles	90.0	math	75.0

행으로 위치를 변경할 열 지정

- **value_vars=[열이름리스트]**

```
# kor
df.melt(id_vars='name', value_vars='kor')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	variable	value
0	Aiden	kor	100.0
1	Charles	kor	90.0

```
# kor, eng
df.melt(id_vars='name', value_vars=['kor','eng'])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	variable	value
0	Aiden	kor	100.0
1	Charles	kor	90.0
2	Aiden	eng	90.0
3	Charles	eng	80.0

컬럼명 변경하기

- **var_name=컬럼명** --> value_vars로 위치를 변경한 열 이름
- **value_name=var_name**으로 위치를 변경한 열의 데이터를 저장한 열 이름

```
# subject, score
df.melt(id_vars='name', value_vars=['kor','eng'], var_name='subject',
value_name='score')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	subject	score
0	Aiden	kor	100.0
1	Charles	kor	90.0
2	Aiden	eng	90.0
3	Charles	eng	80.0

pivot

열을 행으로 보내기(pivot)

```
# 샘플데이터
df = pd.read_csv('data/scores.csv')
df = df.head(2)
df = df.melt(id_vars = 'name', var_name='subject',value_name='score')

def get_grade(x):
    if x>=90: grade='A'
```

```
    elif x>=80: grade='B'
    elif x>=70: grade='C'
    elif x>=60: grade='D'
    else: grade='F'
    return grade

df['grade'] = df['score'].apply(get_grade)
df = df.sort_values('name')
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	subject	score	grade
0	Aiden	kor	100.0	A
2	Aiden	eng	90.0	A
4	Aiden	math	95.0	A
1	Charles	kor	90.0	A
3	Charles	eng	80.0	B
5	Charles	math	75.0	C

- 데이터프레임.pivot(index=인덱스로 사용할 컬럼, columns=컬럼으로 사용할 컬럼, values=값으로 사용할 컬럼)

```
# name, subject, score
df.pivot(index='name', columns='subject',values='score')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

subject	eng	kor	math
name			
Aiden	90.0	100.0	95.0
Charles	80.0	90.0	75.0

```
# name, subject, grade
df.pivot(index='name', columns='subject',values='grade')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

subject	eng	kor	math
name			
Aiden	A	A	A
Charles	B	A	C

```
# name, subject, [grade,score]
df.pivot(index='name', columns='subject',values=['score','grade'])
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead tr th {
    text-align: left;
}

.dataframe thead tr:last-of-type th {
    text-align: right;
}
```

	score			grade		
subject	eng	kor	math	eng	kor	math
name						
Aiden	90.0	100.0	95.0	A	A	A
Charles	80.0	90.0	75.0	B	A	C

```
# name, subject
df.pivot(index='name', columns='subject')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead tr th {
    text-align: left;
}

.dataframe thead tr:last-of-type th {
    text-align: right;
}
```

	score			grade		
subject	eng	kor	math	eng	kor	math
name						
Aiden	90.0	100.0	95.0	A	A	A
Charles	80.0	90.0	75.0	B	A	C

행과 열 바꾸기

- 데이터프레임.transpose()

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	subject	score	grade
0	Aiden	kor	100.0	A
2	Aiden	eng	90.0	A
4	Aiden	math	95.0	A
1	Charles	kor	90.0	A
3	Charles	eng	80.0	B

```
df.transpose()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	0	2	4	1	3	5
name	Aiden	Aiden	Aiden	Charles	Charles	Charles

	0	2	4	1	3	5
subject	kor	eng	math	kor	eng	math
score	100.0	90.0	95.0	90.0	80.0	75.0
grade	A	A	A	A	B	C

데이터프레임 연결

- pd.**concat(데이터프레임리스트)**

데이터프레임이 **동일한 컬럼명 기준으로** 행으로 연결된다.

- pd.concat(데이터프레임리스트, **axis=1**)

데이터프레임이 **동일한 인덱스 기준으로** 열로 연결된다.

행으로 연결하기

```
# 샘플 데이터
df1 = pd.DataFrame([['a', 1], ['b', 2]], columns=['letter', 'number'])
df2 = pd.DataFrame([['c', 3], ['d', 4]], columns=['letter', 'number'])
df3 = pd.DataFrame([['e', 5, '!'], ['f', 6, '@']], columns=['letter', 'number',
'etc'])
```

```
df1
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	letter	number
0	a	1
1	b	2

```
df2
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	letter	number
0	c	3
1	d	4

```
df3
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	letter	number	etc
0	e	5	!

	letter	number	etc
1	f	6	@

컬럼명 기준으로 연결하기

```
df_rowconcat = pd.concat([df1,df2,df3])
df_rowconcat
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	letter	number	etc
0	a	1	NaN
1	b	2	NaN
0	c	3	NaN
1	d	4	NaN
0	e	5	!
1	f	6	@

공통된 컬럼만 남기기

- **join='inner'**

```
df_rowconcat = pd.concat([df1,df2,df3], join='inner')
```

```
df_rowconcat.loc[0]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	letter	number
0	a	1
0	c	3
0	e	5

인덱스 재지정

- **ignore_index=True**

```
df_rowconcat = pd.concat([df1,df2,df3], join='inner',ignore_index=True)
df_rowconcat
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	letter	number
0	a	1
1	b	2
2	c	3
3	d	4
4	e	5

	letter	number
5	f	6

열로 연결하기

```
# 샘플 데이터
df4 = pd.DataFrame({'age':[20,21,22]},index = ['amy','james','david'])
df5 = pd.DataFrame({'phone':['010-111-1111','010-222-2222','010-333-3333']},index
= ['amy','james','david']
)
df6 = pd.DataFrame({'job':['student','programmer','ceo','designer']},index =
['amy','james','david','J']
)
```

```
df4
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age
amy	20
james	21
david	22

```
df5
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	phone
amy	010-111-1111
james	010-222-2222
david	010-333-3333

```
df6
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	job
amy	student
james	programmer
david	ceo
J	designer

인덱스를 기준으로 연결하기

```
df_column_concat = pd.concat([df4,df5,df6], axis=1)
df_column_concat
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age	phone	job
amy	20.0	010-111-1111	student
james	21.0	010-222-2222	programmer
david	22.0	010-333-3333	ceo
J	NaN	NaN	designer

공통된 인덱스만 남기기

```
df_column_concat = pd.concat([df4,df5,df6], axis=1, join='inner')
df_column_concat
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age	phone	job
amy	20	010-111-1111	student
james	21	010-222-2222	programmer
david	22	010-333-3333	ceo

공통된 열을 기준으로 연결하기(merge)

- pd.**merge**(left,right,on=기준컬럼,how=연결방법)
- 2개의 데이터프레임을 연결한다.

```
df = pd.read_csv('data/scores.csv')
df7 = df.loc[[1,2,3]][['name','eng']]
df8 = df.loc[[1,2,4]][['name','math']]
```

```
df7
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	eng
1	Charles	80.0
2	Danial	100.0
3	Evan	100.0

```
df8
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	name	math
1	Charles	75.0
2	Danial	100.0
4	Henry	60.0

공통 데이터만으로 연결

- how='inner'(default)

```
pd.merge(df7, df8, on='name')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	eng	math
0	Charles	80.0	75.0
1	Danial	100.0	100.0

```
pd.merge(df7, df8, on='name', how='inner')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	eng	math
0	Charles	80.0	75.0
1	Danial	100.0	100.0

모든 행 연결

- how='outer'

```
pd.merge(df7, df8, on='name', how='outer')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	eng	math
0	Charles	80.0	75.0
1	Danial	100.0	100.0
2	Evan	100.0	NaN
3	Henry	NaN	60.0

왼쪽 데이터프레임 기준으로 연결

```
pd.merge(df7, df8, on='name', how='left')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	eng	math
0	Charles	80.0	75.0
1	Danial	100.0	100.0
2	Evan	100.0	NaN

오른쪽 데이터프레임 기준으로 연결

```
pd.merge(df7, df8, on='name', how='right')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	eng	math

	name	eng	math
0	Charles	80.0	75.0
1	Danial	100.0	100.0
2	Henry	NaN	60.0

집계

pivot( )

피벗테이블로 데이터 집계하기

- 피벗테이블은 표의 데이터를 요약하는 통계표이다.
- pd.**pivot_table**(데이터프레임, index=인덱스, columns=컬럼, values=집계할데이터, aggfunc=통계함수)
- aggrunc의 디폴트는 mean

```
import pandas as pd
```

```
# 샘플데이터
df = pd.DataFrame({"item": ["shirts", "shirts", "shirts", "shirts", "shirts",
                            "pants", "pants", "pants", "pants"],
                   "color": ["white", "white", "white", "black", "black",
                             "white", "white", "black", "black"],
                   "size": ["small", "large", "large", "small",
                            "small", "large", "small", "small",
                           "large"],
                   "sale": [1, 2, 2, 3, 3, 4, 5, 6, 7],
                   "inventory": [2, 4, 5, 5, 6, 6, 8, 9, 9]})
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	item	color	size	sale	inventory
0	shirts	white	small	1	2
1	shirts	white	large	2	4
2	shirts	white	large	2	5
3	shirts	black	small	3	5
4	shirts	black	small	3	6
5	pants	white	large	4	6
6	pants	white	small	5	8

```
# item, size별 재고 합계
df.pivot_table(index='item', columns='size', values='inventory', aggfunc='sum')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

size	large	small
item		
pants	15	17
shirts	9	13

```
# [item,color], size별 재고 합계
df.pivot_table(index=['item','color'], columns='size', values='inventory',
aggfunc='sum')
```

```
.dataframe tbody tr th {
    vertical-align: top;
```

```
}

.dataframe thead th {
    text-align: right;
}
```

	size	large	small
item	color		
pants	black	9.0	9.0
	white	6.0	8.0
shirts	black	NaN	11.0
	white	9.0	2.0

```
# null값은 0으로 처리
df.pivot_table(index=['item','color'], columns='size', values='inventory',
aggfunc='sum', fill_value=0)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	size	large	small
item	color		
pants	black	9	9
	white	6	8
shirts	black	0	11
	white	9	2

```
# [item,color], size별 판매,재고 합계
df.pivot_table(index=['item','color'], columns='size', values=
```

```
['sale','inventory'], aggfunc='sum', fill_value=0)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead tr th {
    text-align: left;
}

.dataframe thead tr:last-of-type th {
    text-align: right;
}
```

		inventory		sale	
	size	large	small	large	small
item	color				
pants	black	9	9	7	6
	white	6	8	4	5
shirts	black	0	11	0	6
	white	9	2	4	1

타이타닉호 성별, 객실등급별 생존분석

```
import pandas as pd
df = pd.read_csv('data/titanic.csv')
df_titanic = df[['Survived','Pclass','Sex','Age','Embarked']]
df_titanic = df_titanic.dropna()
df_titanic.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived	Pclass	Sex	Age	Embarked

	Survived	Pclass	Sex	Age	Embarked
0	0	3	male	22.0	S
1	1	1	female	38.0	C
2	1	3	female	26.0	S
3	1	1	female	35.0	S
4	0	3	male	35.0	S

```
len(df_titanic)
```

```
1044
```

성별,객실등급별 승선자 수

- count

```
df_titanic.pivot_table(index='Sex',columns='Pclass',values='Survived',aggfunc='cou
nt', margins=True)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

Pclass	1	2	3	All
Sex				
female	131	103	152	386
male	151	158	349	658
All	282	261	501	1044

성별,객실등급별 생존자 수

- sum

```
df_titanic.pivot_table(index='Sex',columns='Pclass',values='Survived',aggfunc='sum', margins=True)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

Pclass	1	2	3	All
Sex				
female	128	97	97	322
male	40	15	38	93
All	168	112	135	415

성별, 객실등급별 생존율

- mean(default)

```
df_titanic.pivot_table(index='Sex',columns='Pclass',values='Survived',aggfunc='mean', margins=True)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

Pclass	1	2	3	All
Sex				
female	0.977099	0.941748	0.638158	0.834197
male	0.264901	0.094937	0.108883	0.141337
All	0.595745	0.429119	0.269461	0.397510

```
df_titanic.pivot_table(index='Sex',columns='Pclass',values='Survived',
margins=True)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

Pclass	1	2	3	All
Sex				
female	0.977099	0.941748	0.638158	0.834197
male	0.264901	0.094937	0.108883	0.141337
All	0.595745	0.429119	0.269461	0.397510

groupby( )

그룹의 통계값 계산하기

```
import pandas as pd
df = pd.read_csv('data/titanic.csv')
df = df[['Survived','Pclass','Sex','Age','Embarked']]
df = df.dropna()
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived	Pclass	Sex	Age	Embarked
0	0	3	male	22.0	S
1	1	1	female	38.0	C
2	1	3	female	26.0	S
3	1	1	female	35.0	S
4	0	3	male	35.0	S

```
len(df)
```

```
1044
```

- **df.groupby(그룹기준컬럼).통계적용컬럼.통계함수**
- count() : 누락값을 제외한 데이터 수
- size() : 누락값을 포함한 데이터 수
- mean() : 평균
- sum() : 합계
- std() : 표준편차
- min() : 최소값
- max() : 최대값
- sum() : 전체 합

객실등급별 생존 통계

```

```

```
# 객실등급(Pclass)별 승선자 수를 구한 결과를 데이터프레임 df1로 만들기
df1 = df.groupby('Pclass').Survived.count().to_frame()
df1
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived
Pclass	
1	282
2	261
3	501

```
# 객실등급(Pclass)별 생존자 수를 구한 결과를 데이터프레임 df2로 만들기
df2 = df.groupby('Pclass').Survived.sum().to_frame()
df2
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived
Pclass	
1	168
2	112

	Survived
Pclass	
3	135

```
# 객실등급(Pclass)별 생존율 구한 결과를 데이터프레임 df3으로 만들기
df3 = df.groupby('Pclass').Survived.mean().to_frame()
df3
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived
Pclass	
1	0.595745
2	0.429119
3	0.269461

```
# 객실등급(Pclass)별 탑승자수, 생존자수, 생존율 데이터프레임을 df4로 만들기
df4 = pd.concat([df1,df2,df3], axis=1)
df4.columns=['승선자수','생존자수','생존율']
df4
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	승선자수	생존자수	생존율
Pclass			
1	282	168	0.595745
2	261	112	0.429119
3	501	135	0.269461

성별 생존 통계

```
# 성별 승선자 수 데이터프레임을 df5로 만들기
df5 = df.groupby('Sex').Survived.count().to_frame()
df5
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived
Sex	
female	386
male	658

```
# 성별 생존자 수 데이터프레임을 df6으로 만들기
df6 = df.groupby('Sex').Survived.sum().to_frame()
df6
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived
Sex	
female	322
male	93

```
# 성별 생존율 데이터프레임을 df7로 만들기
df7 = df.groupby('Sex').Survived.mean().to_frame()
df7
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived
Sex	
female	0.834197
male	0.141337

```
# 성별 탑승자수, 생존자수, 생존율 데이터프레임을 df8로 만들기
df8 = pd.concat([df5,df6,df7], axis=1)
df8.columns = ['승선자수','생존자수','생존율']
df8
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	승선자수	생존자수	생존율
Sex			
female	386	322	0.834197
male	658	93	0.141337

성별, 객실등급별 생존 통계

```
# 성별, 객실등급별 생존율
df.groupby(['Sex','Pclass']).Survived.mean()
```

```
Sex     Pclass
female  1         0.977099
        2         0.941748
        3         0.638158
male    1         0.264901
        2         0.094937
        3         0.108883
Name: Survived, dtype: float64
```

그룹에 사용자 정의 함수 적용하기

- df.groupby(그룹기준컬럼).통계적용컬럼.**agg(사용자정의함수,매개변수들)**

```
def my_mean(values):
    return sum(values)/len(values)
```

```
df.groupby(['Sex','Pclass']).Survived.agg(my_mean)
```

```
Sex     Pclass
female  1         0.977099
        2         0.941748
        3         0.638158
male    1         0.264901
        2         0.094937
        3         0.108883
Name: Survived, dtype: float64
```

그룹 오브젝트 출력하기

- 데이터프레임.groupby(그룹기준컬럼).**groups** --> 그룹별 인덱스:[데이터리스트] 출력
- 데이터프레임.groupby(그룹기준컬럼).**get_group(그룹인덱스)** --> 그룹별 인덱스에 해당하는 데이터프레임 출력

```
df20 = df[:20]
df20
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived	Pclass	Sex	Age	Embarked
0	0	3	male	22.0	S
1	1	1	female	38.0	C
2	1	3	female	26.0	S
3	1	1	female	35.0	S
4	0	3	male	35.0	S

	Survived	Pclass	Sex	Age	Embarked

```
len(df20)
```

```
20
```

```
# Pclass 그룹별 인덱스
df20.groupby('Pclass').groups
```

```
{1: [1, 3, 6, 11], 2: [9, 15, 20, 21], 3: [0, 2, 4, 7, 8, 10, 12, 13, 14,
16, 18, 22]}
```

```
# Pclass 그룹 출력(1등석)
df20.groupby('Pclass').get_group(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived	Pclass	Sex	Age	Embarked
1	1	1	female	38.0	C
3	1	1	female	35.0	S
6	0	1	male	54.0	S
11	1	1	female	58.0	S

```
# Pclass 그룹 출력(2등석)
df20.groupby('Pclass').get_group(2)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived	Pclass	Sex	Age	Embarked
9	1	2	female	14.0	C
15	1	2	female	55.0	S
20	0	2	male	35.0	S
21	1	2	male	34.0	S

```
# Pclass 그룹 출력(3등석)
df20.groupby('Pclass').get_group(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived	Pclass	Sex	Age	Embarked
0	0	3	male	22.0	S
2	1	3	female	26.0	S
4	0	3	male	35.0	S
7	0	3	male	2.0	S

	Survived	Pclass	Sex	Age	Embarked
8	1	3	female	27.0	S

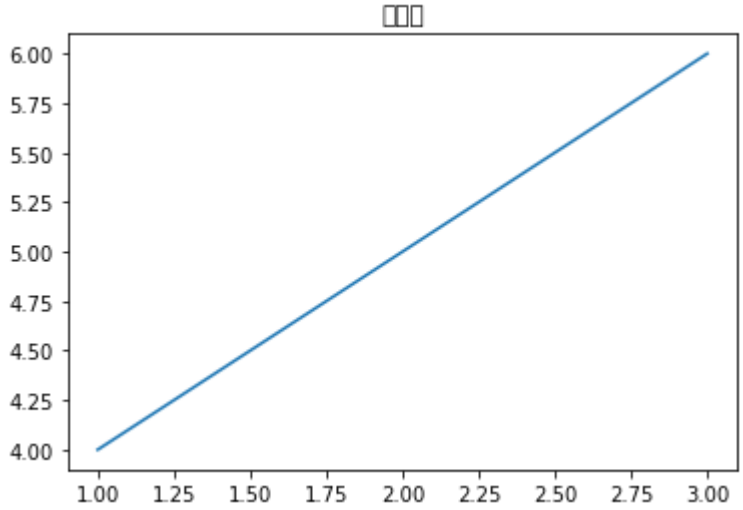
6.데이터 시각화

서울시 코로나19 현황 분석

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

```
plt.plot([1,2,3],[4,5,6])
plt.title('그래프')
plt.show()
```

```
C:\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:238:
RuntimeWarning: Glyph 44536 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:238:
RuntimeWarning: Glyph 47000 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:238:
RuntimeWarning: Glyph 54532 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:201:
RuntimeWarning: Glyph 44536 missing from current font.
  font.set_text(s, 0, flags=flags)
C:\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:201:
RuntimeWarning: Glyph 47000 missing from current font.
  font.set_text(s, 0, flags=flags)
C:\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py:201:
RuntimeWarning: Glyph 54532 missing from current font.
  font.set_text(s, 0, flags=flags)
```




```
# 그래프를 노트북 안에 그리기 위해 설정
%matplotlib inline

# 필요한 패키지와 라이브러리 가져온다.
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

# 그래프에서 마이너스 폰트 깨지는 문제에 대한 대처
mpl.rcParams['axes.unicode_minus'] = False

# 폰트 지정하기
plt.rcParams['font.family'] = 'NanumGothic'
```

```
# 폰트 확인하기
[f.name for f in fm.fontManager.ttflist if 'Nanum' in f.name]
```

```
['NanumSquare',
 'NanumGothic',
 'Nanum HaNaSonGeurSsi',
 'NanumSquare_ac',
 'NanumSquare',
 'NanumGothic',
 'Nanum Pen Script',
 'Nanum Brush Script OTF',
 'NanumSquareRoundOTF',
 'NanumGothic',
 'CUNanum',
 'NanumSquare',
 'NanumSquareRound',
 'NanumBarunGothic',
 'NanumMyeongjo',
 'NanumSquareRound',
```

```
    'NanumSquare_ac',
    'NanumBarunpen',
    'NanumGothic',
    'NanumSquareRound',
    'NanumBarunGothic',
    'NanumBarunpenOTF',
    'Nanum Pen Script',
    'NanumBarunGothic',
    'NanumBarunpen',
    'NanumSquare_ac',
    'NanumMyeongjo',
    'NanumMyeongjo',
    'NanumSquareRound',
    'NanumSquareRound',
    'NanumSquare',
    'Nanum Brush Script',
    'NanumGothicCoding',
    'NanumBarunGothic',
    'NanumSquareRound',
    'NanumMyeongjo',
    'NanumMyeongjo',
    'NanumBarunpenOTF',
    'NanumSquare',
    'NanumMyeongjo',
    'Nanum Pen Script',
    'CUNanum',
    'Nanum Brush Script',
    'NanumBarunGothic',
    'NanumBarunGothic',
    'NanumSquare',
    'NanumBarunGothic',
    'NanumBarunGothic',
    'NanumGothic',
    'Nanum Brush Script',
    'NanumBarunpen',
    'NanumBarunGothic',
    'NanumSquareRoundOTF',
    'NanumSquare_ac',
    'NanumGothicCoding',
    'NanumBarunpen',
    'NanumGothic',
    'NanumSquare',
    'NanumGothic',
    'NanumGothic',
    'NanumBarunpen',
    'NanumSquare',
    'NanumSquareRoundOTF',
    'NanumGothic',
    'NanumSquareRound',
    'Nanum Pen Script OTF',
    'NanumGothic',
    'NanumGothic',
    'NanumBarunGothic',
    'NanumSquareRound',
```

```
 'NanumBarunGothic',
 'NanumBarunpen',
 'NanumGothic',
 'NanumSquareRoundOTF',
 'NanumBarunGothic']
```

- 데이터 수집

https://data.seoul.go.kr/dataList/1/literacyView.do

파일을 utf-8로 변환하여 사용

데이터 불러오기

```
df = pd.read_csv('data/서울시 코로나19 확진자 현황.csv', low_memory=False)
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	연번	확진일	환자번호	국적	환자정보	지역	여행력	접촉력	조치사항	상태	이동경로	등록일	수정일
0	99550	2021-09-28	NaN	NaN	NaN	기타	NaN	감염경로조사중	NaN	-	NaN	2021-09-29 10:54	202 09- 10:

99550 rows × 14 columns

데이터 확인 및 전처리

컬럼별 데이터 확인

```
df['이동경로'].unique()
```

```
array([nan, '이동경로 공개기간 경과'], dtype=object)
```

불필요한 컬럼 삭제

```
df.drop(columns=['환자번호','국적','환자정보','조치사항','이동경로','등록일','수정일','노출여부'], inplace=True)
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	연번	확진일	지역	여행력	접촉력	상태
0	99550	2021-09-28	기타	NaN	감염경로 조사중	-
1	99549	2021-09-28	기타	NaN	감염경로 조사중	-
2	99548	2021-09-28	기타	NaN	감염경로 조사중	-
3	99547	2021-09-28	기타	NaN	감염경로 조사중	-
4	99546	2021-09-28	기타	NaN	감염경로 조사중	-
...	...	...	...	...	...	...
99545	5	2020-01-31	성북구	NaN	기타 확진자 접촉	퇴원

99550 rows × 6 columns

자료형 변환

```
# 자료형 확인
df.dtypes
```

```
연번       int64
확진일     object
지역       object
여행력     object
접촉력     object
상태       object
dtype: object
```

```
# 확진일 --> datetime
df['확진일'] = pd.to_datetime(df['확진일'])
```

```
df.dtypes
```

```
연번               int64
확진일    datetime64[ns]
지역               object
여행력              object
접촉력              object
상태               object
dtype: object
```

```
# 지역의 공백 제거
df['지역'].nunique()
```

```
29
```

```
df['지역'].unique()
```

```
array(['기타', '영등포구', '관악구', '서대문구', '송파구', '동대문구', '성동구',
'중랑구', '타시도',
       '종로구', '도봉구', '용산구', '마포구', '구로구', '동작구', '강동구', '중
구', '노원구',
       '양천구', '강서구', '은평구', '성북구', '광진구', '금천구', '강북구', '강남
구', '서초구',
       '타시도 ', ' 성북구'], dtype=object)
```

```
df['지역'] = df['지역'].str.strip()
```

```
df['지역'].nunique()
```

```
27
```

```
df['지역'].unique()
```

```
array(['기타', '영등포구', '관악구', '서대문구', '송파구', '동대문구', '성동구',
'중랑구', '타시도',
       '종로구', '도봉구', '용산구', '마포구', '구로구', '동작구', '강동구', '중
구', '노원구',
       '양천구', '강서구', '은평구', '성북구', '광진구', '금천구', '강북구', '강남
구', '서초구'],
      dtype=object)
```

```
# 지역, 상태 --> category (지역의 공백 제거)
df['지역'] = df['지역'].astype('category')
```

```
df.dtypes
```

```
연번              int64
확진일    datetime64[ns]
지역            category
여행력             object
접촉력             object
상태              object
dtype: object
```

```
# 정보
df.info()
```

```
RangeIndex: 99550 entries, 0 to 99549
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   연번      99550 non-null  int64
 1   확진일     99550 non-null  datetime64[ns]
 2   지역      99550 non-null  category
 3   여행력     1717 non-null   object
 4   접촉력     99550 non-null  object
 5   상태      99550 non-null  object
dtypes: category(1), datetime64[ns](1), int64(1), object(3)
memory usage: 3.9+ MB
```

결측치 분석

```
df.isnull().sum()
```

```
연번          0
확진일         0
지역          0
여행력     97833
접촉력         0
상태          0
dtype: int64
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	연번	확진일	지역	여행력	접촉력	상태
0	99550	2021-09-28	기타	NaN	감염경로 조사중	-
1	99549	2021-09-28	기타	NaN	감염경로 조사중	-

	연번	확진일	지역	여행력	접촉력	상태
2	99548	2021-09-28	기타	NaN	감염경로 조사중	-
3	99547	2021-09-28	기타	NaN	감염경로 조사중	-
4	99546	2021-09-28	기타	NaN	감염경로 조사중	-

99550 rows × 6 columns

구 별 확진자 동향

확진일-구별 확진자수 집계

피봇테이블 만들기

```
df_gu = pd.pivot_table(df, index='확진일', columns='지역', values='연번',
aggfunc='count', margins=True)
df_gu
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

지역	강남구	강동구	강북구	강서구	관악구	광진구	구로구	금천구	기타	노원구	...	송 구
확진일												
2020-01-24 00:00:00	0	0	0	1	0	0	0	0	0	0	...	(
2020-01-30 00:00:00	0	0	0	0	0	0	0	0	0	0	...	(
2020-01												

583 rows × 28 columns

서울시 일별 추가확진자 동향

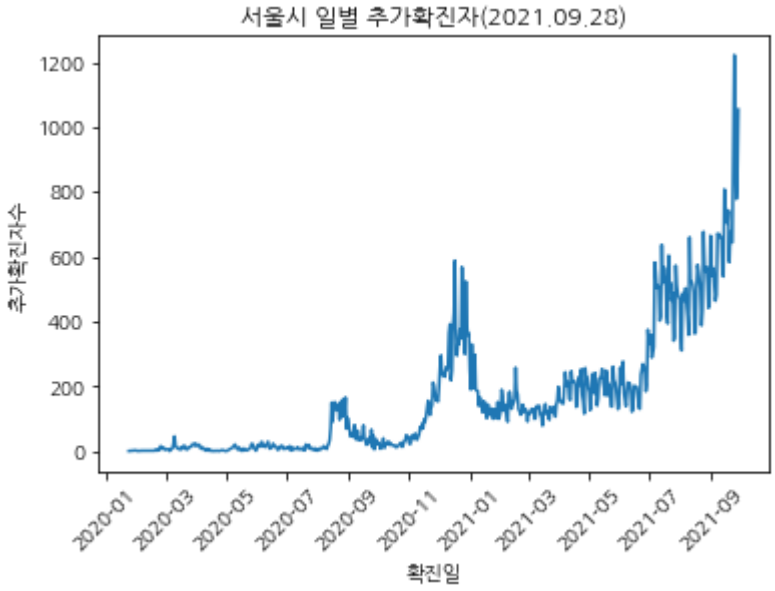
```
s_date = df_gu['All'][:-1]
s_date
```

```
확진일
2020-01-24       1
2020-01-30       3
2020-01-31       3
2020-02-02       1
2020-02-05       2
               ...
2021-09-24    1222
2021-09-25     928
2021-09-26     778
2021-09-27     842
2021-09-28    1054
Name: All, Length: 582, dtype: int64
```

```
# 서울시 일별 추가확진자가 많았던 순으로 보기
s_date.sort_values(ascending=False)
```

```
확진일
2021-09-24    1222
2021-09-28    1054
2021-09-25     928
2021-09-23     905
2021-09-27     842
               ...
2020-04-24       1
2020-04-30       1
2020-05-01       1
2020-05-17       1
2020-01-24       1
Name: All, Length: 582, dtype: int64
```

```
# 서울시 일별 추가확진자 시각화
x = s_date.index
y = s_date.values
plt.plot(x,y)
plt.title('서울시 일별 추가확진자(2021.09.28)')
plt.xlabel('확진일')
plt.ylabel('추가확진자수')
plt.xticks(rotation=45)
plt.show()
```



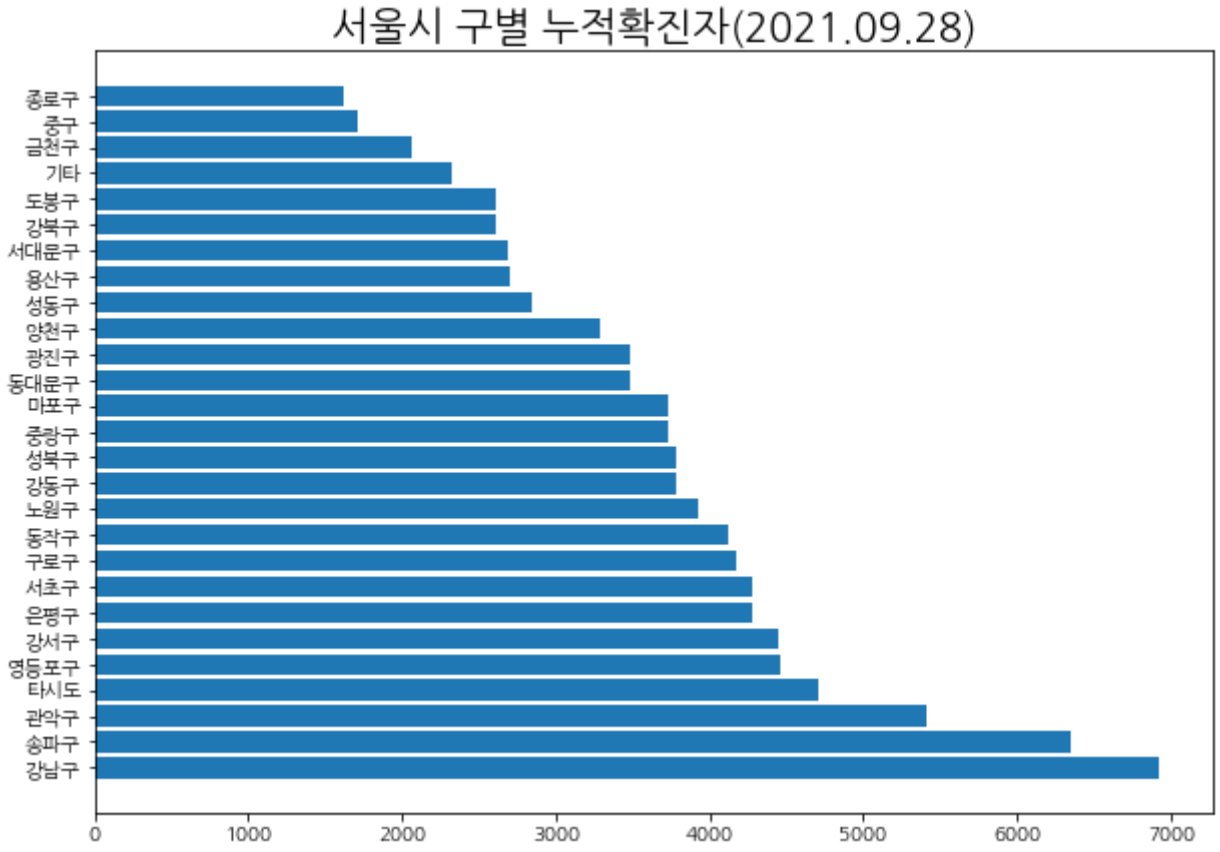
서울시 구별 누적확진자 비교

```
# 서울시 구별 누적확진자 많은 순으로 보기
s_gu = df_gu.loc['All'][:-1]
s_gu = s_gu.sort_values(ascending=False)
s_gu
```

```
지역
강남구     6926
송파구     6356
관악구     5418
타시도     4715
영등포구    4463
강서구     4447
은평구     4284
서초구     4284
구로구     4168
동작구     4120
노원구     3932
강동구     3782
성북구     3779
중랑구     3729
마포구     3726
동대문구    3487
광진구     3486
양천구     3291
성동구     2839
용산구     2705
서대문구    2692
강북구     2606
도봉구     2605
기타      2321
금천구     2064
```

```
중구       1708
종로구      1617
Name: All, dtype: int64
```

```
# 서울시 구별 누적확진자 많은 순으로 시각화
x = s_gu.index
y = s_gu.values
plt.figure(figsize=(10,7))
plt.title('서울시 구별 누적확진자(2021.09.28)', size=20)
plt.barh(x,y)
plt.show()
```



최근일 기준 지역별 추가확진자

```
df_gu
```

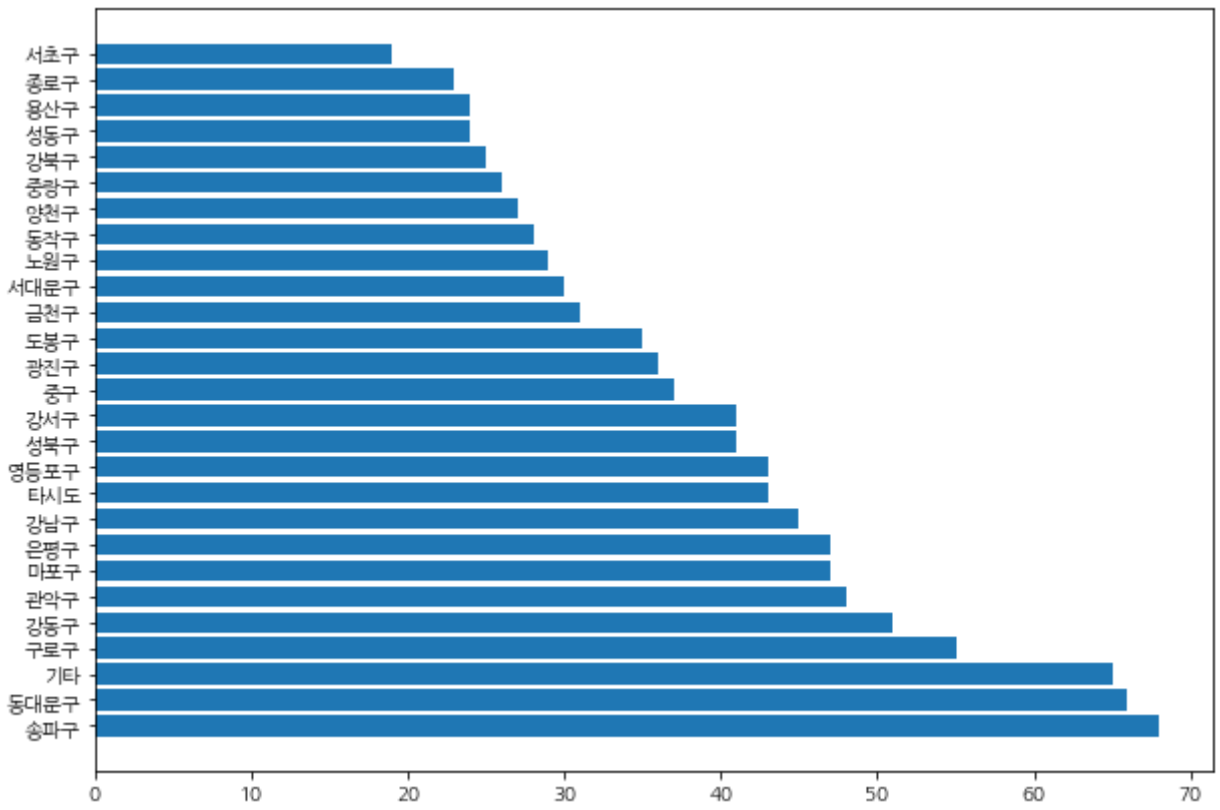
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

지역	강남구	강동구	강북구	강서구	관악구	광진구	구로구	금천구	기타	노원구	...	송구
확진일												
2020-01-24 00:00:00	0	0	0	1	0	0	0	0	0	0	...	0
2020-01-30 00:00:00	0	0	0	0	0	0	0	0	0	0	...	0
2020-01												

583 rows × 28 columns

```
s_gu = df_gu.iloc[-2][:-1]
s_gu = s_gu.sort_values(ascending=False)
```

```
x = s_gu.index
y = s_gu.values
plt.figure(figsize=(10,7))
plt.barh(x,y)
plt.show()
```



접촉력에 따른 확진 분석

접촉력에 따른 확진 건수 best10

```
df['접촉력'].value_counts()[:10].to_frame()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	접촉력
기타 확진자 접촉	37943
감염경로 조사중	31964
타시도 확진자 접촉	4244
해외유입	1729
동부구치소 관련	1175
병원 및 요양시설	988
송파구 소재 시장 관련(?21.9.)	659

최근월 접촉력에 따른 확진 건수 best10

- 2021-09

```
df[(df['확진일'].dt.year==2021)&(df['확진일'].dt.month==9)]['접촉
력'].value_counts()[:10].to_frame()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	접촉력
감염경로 조사중	8117
기타 확진자 접촉	8088
송파구 소재 시장 관련(?21.9.)	658
타시도 확진자 접촉	587
중구 소재 시장 관련(?21.9.)	277
병원 및 요양시설	186

서울시 공공 자전거

- 데이터 수집

https://data.seoul.go.kr/dataList/5/literacyView.do

```
import pandas as pd
```

```
# 그래프를 노트북 안에 그리기 위해 설정
%matplotlib inline

# 필요한 패키지와 라이브러리 가져온다.
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

# 그래프에서 마이너스 폰트 깨지는 문제에 대한 대처
mpl.rcParams['axes.unicode_minus'] = False

# 폰트 지정하기
plt.rcParams['font.family'] = 'gulim'
```

데이터 확인 및 전처리

데이터프레임 생성/확인

```
df1 = pd.read_csv('data/공공자전거 대여이력 정보_2021.01.csv', encoding='cp949',
low_memory=False)
df2 = pd.read_csv('data/공공자전거 대여이력 정보_2021.02.csv', encoding='cp949',
low_memory=False)
df3 = pd.read_csv('data/공공자전거 대여이력 정보_2021.03.csv', encoding='cp949',
low_memory=False)
df4 = pd.read_csv('data/공공자전거 대여이력 정보_2021.04.csv', encoding='cp949',
low_memory=False)
df5 = pd.read_csv('data/공공자전거 대여이력 정보_2021.05.csv', encoding='cp949',
low_memory=False)
```

```
df6 = pd.read_csv('data/공공자전거 대여이력 정보_2021.06.csv', encoding='cp949',
low_memory=False)
```

```
df6.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	자전거번호	대여일시	대여대여소번호	대여 대여소명	대여거치대	반납일시	반납대여소번호	반납대여소명	반납거치대	이용시간	이용거리
0	SPB-30385	2021-05-31 23:07:00	3571	화양 APT(횡단보도 옆)	0	2021-06-01 00:00:00	03538	서울숲 IT 캐	0	53.0	2502.8

데이터 연결/확인

- concat

```
# concat
df = pd.concat([df1,df2,df3,df4,df5,df6])
```

```
# head
df.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	자전거번호	대여일시	대여대여소번호	대여대여소명	대여거치대	반납일시	반납대여소번호	반납대여소명	반납거치대	이용시간	이용거리
0	SPB-53145	2021-01-02 20:50:36	3	중랑센터	0	2021-01-02 21:15:41	668	서울축산농협(장안지점)	0	25.0	0.0

```
#tail
df.tail(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	자전거번호	대여일시	대여대여소번호	대여대여소명	대여거치대	반납일시	반납대여소번호	반납대여소명	반납거치대	이용시간	이용거리
3445943	SPB-52274	2021-06-30	2220	반포본동주	0	2021-07-01	02526	반포경남	0	403.0	4541.22

```
# 데이터 크기
df.shape
```

```
(13613873, 11)
```

```
# 데이터 정보(사용메모리)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13613873 entries, 0 to 3445943
Data columns (total 11 columns):
 #   Column     Dtype  
---  ------     -----  
 0   자전거번호      object 
 1   대여일시       object 
 2   대여 대여소번호   int64  
 3   대여 대여소명    object 
 4   대여거치대      object 
 5   반납일시       object 
 6   반납대여소번호    object 
 7   반납대여소명     object 
 8   반납거치대      int64  
 9   이용시간       float64
 10  이용거리       float64
dtypes: float64(2), int64(2), object(7)
memory usage: 1.2+ GB
```

데이터 전처리

불필요한 컬럼 제거

```
# 자전거번호, 대여거치대, 반납거치대 제거
df.drop(columns=['자전거번호', '대여거치대', '반납거치대'], inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13613873 entries, 0 to 3445943
Data columns (total 8 columns):
 #   Column     Dtype  
---  ------     -----  
 0   대여일시       object 
 1   대여 대여소번호   int64  
 2   대여 대여소명    object 
```

```
 3   반납일시       object
 4   반납대여소번호    object
 5   반납대여소명     object
 6   이용시간       float64
 7   이용거리       float64
dtypes: float64(2), int64(1), object(5)
memory usage: 934.8+ MB
```

자료형 확인/변경

```
# 자료형 확인
df.dtypes
```

```
대여일시          object
대여 대여소번호      int64
대여 대여소명       object
반납일시          object
반납대여소번호       object
반납대여소명        object
이용시간         float64
이용거리         float64
dtype: object
```

```
# 카테고리형으로 변경  :    대여 대여소번호, 반납대여소번호
df['대여 대여소번호'] = df['대여 대여소번호'].astype('category')
df['반납대여소번호'] = df['반납대여소번호'].astype('category')
```

```
df.dtypes
```

```
대여일시          object
대여 대여소번호     category
대여 대여소명       object
반납일시          object
반납대여소번호      category
반납대여소명        object
이용시간         float64
이용거리         float64
dtype: object
```

```
# 메모리 용량 확인
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13613873 entries, 0 to 3445943
Data columns (total 8 columns):
 #   Column     Dtype   
---  ------     -----   
 0   대여일시       object  
 1   대여 대여소번호   category
 2   대여 대여소명    object  
 3   반납일시       object  
 4   반납대여소번호    category
 5   반납대여소명     object  
 6   이용시간       float64 
 7   이용거리       float64 
dtypes: category(2), float64(2), object(4)
memory usage: 779.2+ MB
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	대여일시	대여 대여소번호	대여 대여소명	반납일시	반납대여소번호	반납대여소명	이용시간	이용거리
0	2021-01-02 20:50:36	3	중랑센터	2021-01-02 21:15:41	668	서울축산농협(장안지점)	25.0	0.00
1	2021-01-04 16:02:12	3	중랑센터	2021-01-04 16:17:06	668	서울축산농협(장안지점)	14.0	0.00
	2021-01-		중랑센	2021-01-		서울축산		

13613873 rows × 8 columns

```
# datetime형으로 변경 : 대여일시,반납일시
df['대여일시'] = pd.to_datetime(df['대여일시'])
```

```
df['반납일시'] = pd.to_datetime(df['반납일시'], errors='coerce')
```

```
# 자료형 변경 확인
df.dtypes
```

```
대여일시          datetime64[ns]
대여 대여소번호           category
대여 대여소명             object
반납일시          datetime64[ns]
반납대여소번호           category
반납대여소명             object
이용시간                float64
이용거리                float64
dtype: object
```

결측치 확인/처리

```
# 결측치 확인
df.isnull().sum()
```

```
대여일시            0
대여 대여소번호        0
대여 대여소명         0
반납일시          123
반납대여소번호         0
반납대여소명          0
이용시간            0
이용거리          269
dtype: int64
```

```
# 결측치 제거
df.dropna(inplace=True)
```

```
# 결측치 확인
df.isnull().sum()
```

```
대여일시         0
대여 대여소번호     0
대여 대여소명      0
반납일시         0
반납대여소번호      0
반납대여소명       0
이용시간         0
이용거리         0
dtype: int64
```

일별 이용 현황

대여날짜 컬럼 추가

- date

```
df['대여날짜'] = df['대여일시'].dt.date
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	대여일시	대여 대여소번호	대여 대여소명	반납일시	반납대여소번호	반납대여소명	이용시간	이용거리	대여날짜
0	2021-01-02 20:50:36	3	중랑센터	2021-01-02 21:15:41	668	서울축산농협(장안지점)	25.0	0.00	2021-01-02

		대여			반납				

13613604 rows × 9 columns

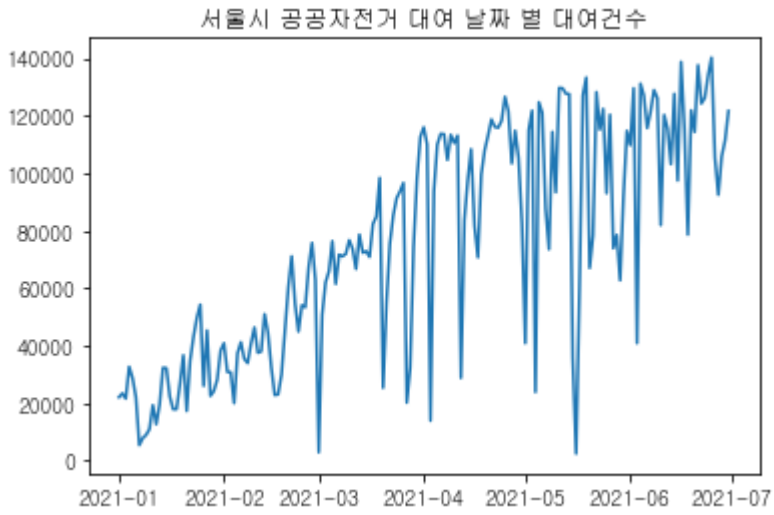
대여날짜 별 대여 건수

- groupby
- to_frame

```
# 대여날짜 별 대여건수 추출
df_count = df.groupby('대여날짜').대여일시.count().to_frame()
```

```
# 대여날짜 별 대여건수 시각화
df_count.columns=['대여건수']
```

```
plt.plot(df_count.index, df_count.values)
plt.title('서울시 공공자전거 대여 날짜 별 대여건수')
plt.show()
```

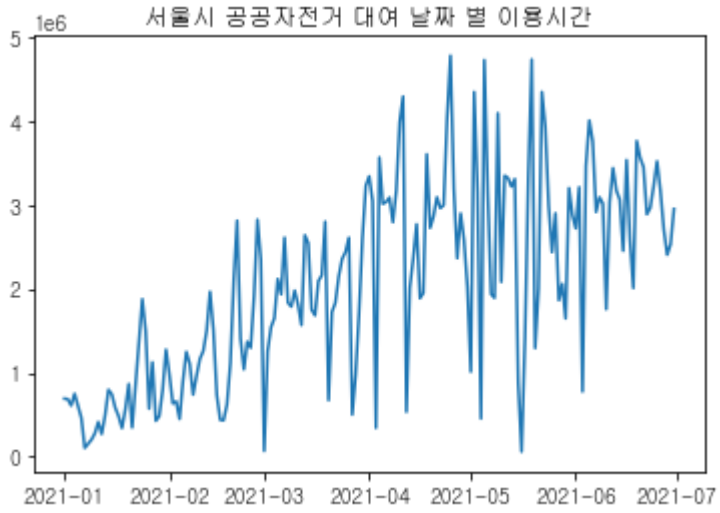


대여날짜 별 이용시간

- groupby
- to_frame

```
# 대여날짜 별 이용시간 추출
df_time = df.groupby('대여날짜')['이용시간'].sum().to_frame()
```

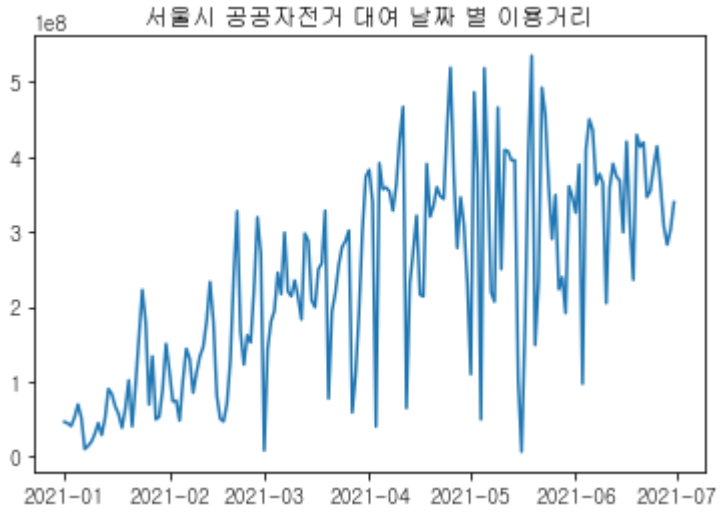
```
# 대여날짜 별 이용시간 시각화
plt.plot(df_time.index, df_time['이용시간'])
plt.title('서울시 공공자전거 대여 날짜 별 이용시간')
plt.show()
```



대여날짜 별 이용 거리

```
# 대여날짜 별 이용거리 추출
df_distance = df.groupby('대여날짜')['이용거리'].sum().to_frame()
```

```
# 대여날짜 별 이용거리 시각화
plt.plot(df_distance.index, df_distance['이용거리'])
plt.title('서울시 공공자전거 대여 날짜 별 이용거리')
plt.show()
```



데이터프레임 합치기

```
df_date = pd.concat([df_time,df_distance,df_count], axis=1)
df_date
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	이용시간	이용거리	대여건수
대여날짜			
2021-01-01	692309.0	4.613529e+07	22119
2021-01-02	686580.0	4.409618e+07	23463
2021-01-03	609571.0	4.059830e+07	21656
2021-01-04	750721.0	5.249645e+07	32732
2021-01-05	610227.0	6.944856e+07	28819
...	...	...	...

181 rows × 3 columns

시간대별 대여/반납 현황

대여시간, 반납시간 컬럼 추가

```
df['대여시간'] = df['대여일시'].dt.hour
```

```
df['반납시간'] = df['반납일시'].dt.hour
```

```
df.dtypes
```

```
대여일시          datetime64[ns]
대여 대여소번호           category
대여 대여소명             object
반납일시          datetime64[ns]
```

```
반납대여소번호          category
반납대여소명             object
이용시간               float64
이용거리               float64
대여날짜                object
대여시간                 int64
반납시간                 int64
dtype: object
```

시간대별 대여/반납 현황

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	대여일시	대여대여소번호	대여대여소명	반납일시	반납대여소번호	반납대여소명	이용시간	이용거리	대여날짜	대여시간	반
0	2021-01-02	3	중랑	2021-01-02	668	서울축산농	25.0	0.00	2021-	20	2

13613604 rows × 11 columns

```
# 시간대별 대여현황
s_rental = df['대여시간'].value_counts()
s_rental
```

```
18    1432779
17    1178279
19     984476
16     934211
```

```
20     825031
15     816040
8      808506
21     770378
14     725175
13     667237
22     643611
12     623065
11     537810
9      499995
7      486775
10     438084
23     341859
0      240025
6      188920
1      158905
2      103629
5       84713
3       68710
4       55391
Name: 대여시간, dtype: int64
```

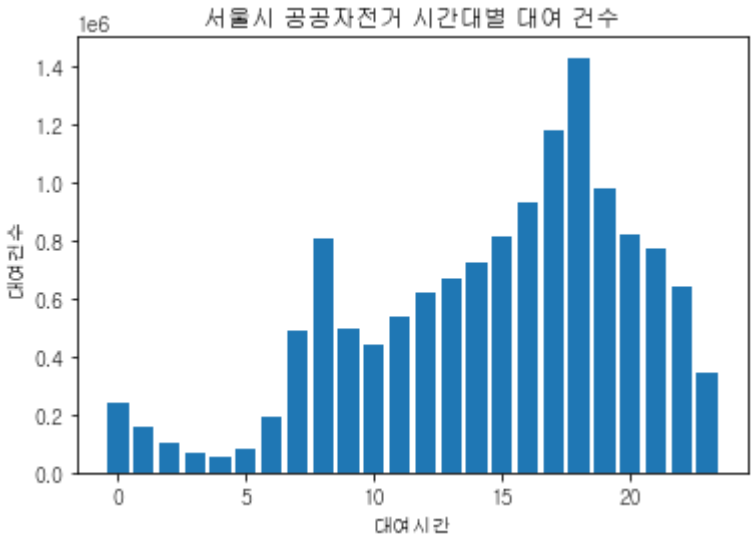
```
# 시간대별 반납현황
s_return = df['반납시간'].value_counts()
s_return
```

```
18    1437723
19    1125447
17    1097623
16     880629
20     864644
8      817076
21     812357
22     768302
15     749940
14     657826
13     628908
12     591577
9      499033
11     470053
23     447477
10     402350
7      389007
0      290169
1      190954
6      152003
2      125912
3       80243
5       73983
```

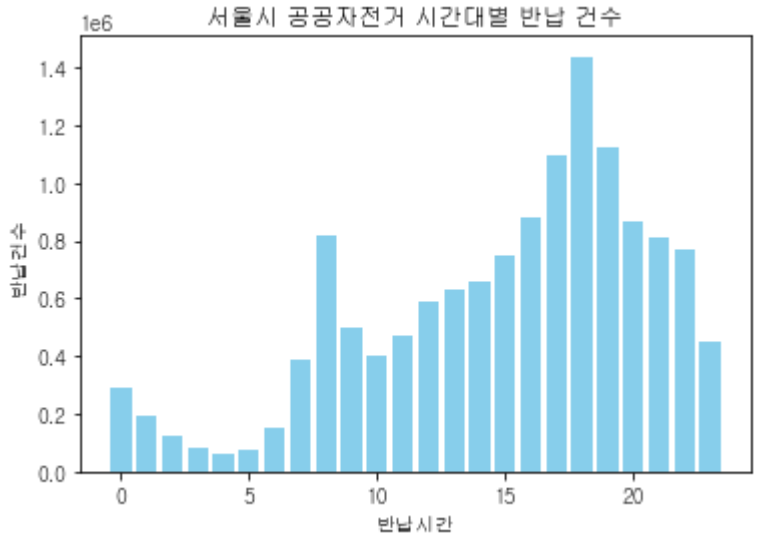
```
4        60368
Name: 반납시간, dtype: int64
```

시각화

```
# 시간대별 공공자전거 대여건수 시각화
s_rental = s_rental.sort_index()
s_rental
x = s_rental.index
y = s_rental.values
plt.bar(x,y)
plt.title('서울시 공공자전거 시간대별 대여 건수')
plt.xlabel('대여시간')
plt.ylabel('대여건수')
plt.show()
```




```
# 시간대별 공공자전거 반납건수 시각화
s_return = s_return.sort_index()
x = s_return.index
y = s_return.values
plt.bar(x,y,color='skyblue')
plt.title('서울시 공공자전거 시간대별 반납 건수')
plt.xlabel('반납시간')
plt.ylabel('반납건수')
plt.show()
```



대여소별 대여/반납 현황

대여소 현황

```
# 대여 대여소번호 갯수
df['대여 대여소번호']
```

```
0              3
1              3
2              3
3              3
4              3
            ...
3445939      152
3445940      152
3445941      152
3445942     2220
3445943     2220
Name: 대여 대여소번호, Length: 13613604, dtype: category
Categories (2493, int64): [3, 5, 10, 101, ..., 9999, 88888, 99997, 99999]
```

```
# 반납대여소번호 갯수
df['반납대여소번호']
```

```
0              668
1              668
2              668
3              668
4              540
            ...
```

```
3445939    00126
3445940    00437
3445941    00437
3445942    02526
3445943    02526
Name: 반납대여소번호, Length: 13613604, dtype: category
Categories (4811, object): [3, 10, 101, 102, ..., '화랑대역 2번출구 앞', '휘경
sk뷰아파트 앞', '휘경여중고삼거리', '흑석역 4번출구']
```

```
# 반납대여소 번호 처리 (str형으로 변환)
df['반납대여소번호'] = df['반납대여소번호'].astype('str')
```

```
# 반납대여소 번호 처리 (왼쪽의 '0' 제거)
df['반납대여소번호'] = df['반납대여소번호'].str.lstrip('0')
```

```
# 반납대여소 번호 처리 (int형으로 변환)
df['반납대여소번호'] = df['반납대여소번호'].astype('int')
```

```
# 반납대여소 번호 처리 (category형으로 변환)
df['반납대여소번호'] = df['반납대여소번호'].astype('category')
```

```
df['반납대여소번호']
```

```
0           668
1           668
2           668
3           668
4           540
           ...
3445939     126
3445940     437
3445941     437
3445942    2526
3445943    2526
Name: 반납대여소번호, Length: 13613604, dtype: category
Categories (2493, int64): [3, 5, 10, 101, ..., 9999, 88888, 99997, 99999]
```

대여건수가 가장 많은 대여소 best10

```
# value_counts
df[['대여 대여소번호','대여 대여소명']].value_counts()[:10].to_frame()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

		0
대여 대여소번호	대여 대여소명	
207	여의나루역 1번출구 앞	68994
502	뚝섬유원지역 1번출구 앞	68588
152	마포구민체육센터 앞	43535
2102	봉림교 교통섬	43368
1210	롯데월드타워(잠실역2번출구 쪽)	37149
2715	마곡나루역 2번 출구	36988

반납건수가 가장 많은 대여소 best10

```
# value_counts
df[['반납대여소번호','반납대여소명']].value_counts()[:10].to_frame()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

		0
반납대여소번호	반납대여소명	
502	뚝섬유원지역 1번출구 앞	78958
207	여의나루역 1번출구 앞	68966

		0
반납대여소번호	반납대여소명	
152	마포구민체육센터 앞	50607
2102	보린교 교통섬	44572

여의나루역 1번출구 앞 대여소 이용현황

서브셋 만들기

```
df_207 = df[df['대여 대여소번호']==207]
df_207.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	대여일시	대여 대여소번호	대여 대여소명	반납일시	반납 대여소번호	반납 대여소명	이용시간	이용거리	대여 날짜	대여 시간	반납 시간
45838	2021-01-01 01:05:49	207	여의나루역 1번출구 앞	2021-01-01 01:16:29	201	진미파라곤 앞	10.0	0.0	2021-01-01	1	1

반납 현황

```
# value_counts
df_207[['반납대여소번호','반납대여소명']].value_counts().to_frame()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

		0
반납대여소번호	**반납대여소명**	
207	**여의나루역 1번출구 앞**	18347
202	**국민일보 앞**	2006
222	**시범아파트버스정류장 옆**	1588
249	**여의도중학교 옆**	1525
272	**당산육갑문**	1269
...	**...**	...

1413 rows × 1 columns

요일별 대여현황

```
#요일컬럼 추가 : strftime('%a')
df_207['대여요일'] = df_207['대여일시'].dt.strftime('%a')
```

```
:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_207['대여요일'] = df_207['대여일시'].dt.strftime('%a')
```

```
# value_counts
df_207['대여요일'].value_counts()
```

```
Sun    14454
Sat    10932
Wed    10755
Fri     9191
Mon     8391
Thu     8037
Tue     7235
Name: 대여요일, dtype: int64
```

이용시간 통계

```
# 이용시간 평균
df_207['이용시간'].mean()
```

```
50.28551344300312
```

```
# 이용시간 최대
df_207['이용시간'].max()
```

```
1268.0
```

```
# 이용시간 최소
df_207['이용시간'].min()
```

```
1.0
```

```
# 전체데이터 이용시간 평균
df['이용시간'].mean()
```

```
27.27823322905529
```

서울시 물가 정보 분석

데이터 수집

http://data.seoul.go.kr/dataList/OA-1170/S/1/datasetView.do

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

```
# 그래프를 노트북 안에 그리기 위해 설정
%matplotlib inline

# 필요한 패키지와 라이브러리 가져온다.
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

# 그래프에서 마이너스 폰트 깨지는 문제에 대한 대처
mpl.rcParams['axes.unicode_minus'] = False

# 폰트 지정하기
plt.rcParams['font.family'] = 'NanumBarunGothic'
```

데이터프레임 생성

- csv 파일을 데이터프레임으로 만들기
- encoding='cp949'

```
df = pd.read_csv('data/생필품 농수축산물 가격 정보(2021년1월_6월).csv',
encoding='cp949')
```

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

42699 rows × 14 columns

결측치 확인

```
df.isnull().sum()
```

```
일련번호                        0
시장/마트 번호                    0
시장/마트 이름                    0
품목 번호                        0
품목 이름                        0
실판매규격                        0
가격(원)                        0
년도-월                         0
비고                          274
시장유형 구분(시장/마트) 코드           0
시장유형 구분(시장/마트) 이름           0
자치구 코드                       0
자치구 이름                       0
점검일자                         0
dtype: int64
```

자료형 확인

```
df.dtypes
```

```
일련번호                        int64
시장/마트 번호                    int64
시장/마트 이름                   object
품목 번호                        int64
품목 이름                       object
실판매규격                       object
가격(원)                        int64
년도-월                         object
비고                          object
시장유형 구분(시장/마트) 코드          int64
시장유형 구분(시장/마트) 이름         object
자치구 코드                       int64
자치구 이름                      object
점검일자                        object
dtype: object
```

데이터 확인

컬럼별 데이터 확인

```
df['시장/마트 번호'].nunique()
```

```
102
```

시장/마트 목록

```
# 시장/마트 목록
df['시장/마트 이름'].nunique()
```

```
102
```

```
df['시장/마트 이름'].unique()
```

```
array(['신세계백화점', '방배종합시장', '통인시장', '방학동도깨비시장', '이마트 용산점', '용문시장',
       'NC백화점 불광점', '관악신사시장(신림4동)', '영천시장', '고척근린시장', '이마트 여의도점',
       '현대백화점 미아점', '롯데백화점', '이마트 가양점', '홈플러스 동대문점', '송화시장', '영등포전통시장',
       '홈플러스 등촌점', '후암시장', '경동시장', '롯데마트 강변점', '롯데백화점 노원점', '롯데백화점 청량리점',
       '홈플러스 영등포점', '우림시장', '청량리종합시장', '농협 하나로마트 용산점', '이마트 자양점',
       '롯데백화점 미아점', '자양골목시장', '수유재래시장', '홈플러스 중계점', '금남시장', '인왕시장',
       '원당종합시장', '목3동시장', '신영시장', '이마트 왕십리점', '뉴코아아울렛 강남점', '망원시장',
       '마천중앙시장', '신세계백화점 강남점', '롯데백화점 영등포점', '남성시장', '마포농수산물시장',
       '이마트 창동점', '공릉동 도깨비시장', '홈플러스 면목점', '이마트 청계점', '이마트 성수점',
       '홈플러스 방학점', '이마트 역삼점', '남구로시장', '이마트 은평점', '뚝도시장', '롯데백화점 강남점',
       '노룬산골목시장', '현대백화점 신촌점', '홈플러스 목동점', '태평백화점', '대조시장', '남문시장',
       '광장시장', '대림중앙시장', '남대문시장', '하나로클럽 양재점', '현대시장', '상계중앙시장',
       '농협하나로마트 신촌점', '이마트 신도림점', '신창시장', '롯데백화점 잠실점', '이마트 목동점',
```

```
       '홈플러스 잠실점', '대림시장', '홈플러스 시흥점', '암사종합시장', '이마트 상
봉점', '숭인시장',
       '둔촌역전통시장', '홈플러스 월드컵점', '도곡시장', '신원시장(신림1동)', '돈암
제일시장', '청담삼익시장',
       '롯데백화점 관악점', '세이브 마트', '화곡본동시장', '롯데마트 서울역점', '방
이시장', '장위골목시장',
       '롯데슈퍼', '이마트 미아점', '이마트 명일점', '홈플러스 강동점', '서울중앙시
장', '홈플러스 독산점',
       '동원시장', '하나로클럽 미아점', 'NC백화점 신구로점', '이마트 에브리데이 창동
점', '롯데마트 구로점'],
      dtype=object)
```

```
# 시장/마트 목록
df_market = df[['시장/마트 번호','시장/마트 이름','자치구 이름','시장유형 구분(시장/마
트) 이름']].drop_duplicates()
```

```
df_market
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	시장/마트 번호	시장/마트 이름	자치구 이름	시장유형 구분(시장/마트) 이름
0	2	신세계백화점	중구	대형마트
1	228	방배종합시장	서초구	전통시장
2	1	통인시장	종로구	전통시장
3	25	방학동도깨비시장	도봉구	전통시장
4	6	이마트 용산점	용산구	대형마트
...	...	...	...	...

102 rows × 4 columns

```
# 자치구 별 시장/마트 갯수
df_market['자치구 이름'].value_counts()
```

```
중구        6
구로구       5
영등포구      5
도봉구       5
관악구       5
광진구       4
동대문구      4
강북구       4
강동구       4
용산구       4
양천구       4
서대문구      4
중랑구       4
노원구       4
강남구       4
성동구       4
은평구       4
서초구       4
금천구       4
성북구       4
송파구       4
마포구       4
강서구       4
동작구       2
종로구       2
Name: 자치구 이름, dtype: int64
```

```
# 자치구 이름으로 시장/마트 확인
df_market[df_market['자치구 이름']=='중구']
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	시장/마트 번호	시장/마트 이름	자치구 이름	시장유형 구분(시장/마트) 이름
0	2	신세계백화점	중구	대형마트
21	56	롯데백화점	중구	대형마트
257	60	이마트 청계점	중구	대형마트
576	11	남대문시장	중구	전통시장

	시장/마트 번호	시장/마트 이름	자치구 이름	시장유형 구분(시장/마트) 이름
3799	8	롯데마트 서울역점	중구	대형마트

품목 목록

```
# 품목 목록
df_items =df[['품목 번호','품목 이름']].drop_duplicates()
df_items = df_items.sort_values('품목 이름')
```

```
# 품목 이름 (30개씩 확인)
df_items[:30]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	품목 번호	품목 이름
222	13	고등어
135	304	고등어
15781	316	고등어(30cm,국산)
1550	318	고등어(30cm,수입산)
3	268	고등어(냉동,국산)
4834	269	고등어(냉동,수입산)
2	266	고등어(생물,국산)

```
df_items[30:60]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	품목 번호	품목 이름
2457	265	명태(생물,수입산)
1019	184	명태(일본산,냉동)
79	25	무
103	308	무(1kg)
0	133	무(세척무)
33	282	무(세척무)
2432	274	무(세척무, 중)

```
df_items[60:]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	품목 번호	품목 이름
10	309	양파(1.5kg망)
2191	272	양파(작은망)
5	22	오이
21	311	오이(다다기)
31	253	오징어
2620	54	오징어(냉동)
441	256	오징어(냉동,국산)

자치구 목록

```
df_gu = df[['자치구 코드','자치구 이름']].drop_duplicates()
df_gu.shape
```

```
(25, 2)
```

시장 유형

```
df_gubun = df[['시장유형 구분(시장/마트) 코드','시장유형 구분(시장/마트) 이
름']].drop_duplicates()
df_gubun
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	시장유형 구분(시장/마트) 코드	시장유형 구분(시장/마트) 이름
0	2	대형마트
1	1	전통시장

삼겹살 가격 분석

삼겹살 데이터

- 특정 문자열이 포함되어있는지 확인 : .str.contains(문자열)

```
# 2021-06 데이터 이용
df_sam = df[ (df['품목 이름'].str.contains('삼겹살')) & (df['년도-월']=='2021-06')
& (df['실판매규격'].str.contains('600g')) ]
df_sam
```

```
.dataframe tbody tr th {
    vertical-align: top;
```

```
}

.dataframe thead th {
    text-align: right;
}
```

	일련번호	시장/마트번호	시장/마트이름	품목번호	품목이름	실판매규격	가격(원)	년도-월	비고	시장유형구분(시장/마트)코드	시장유형구분(시장/마트)이름	자치구코드

224 rows × 14 columns

```
# 삼겹살 600g의 평균 가격은?
df_sam['가격(원)'].mean()
```

```
16842.723214285714
```

```
# 삼겹살 600g의 최고 가격은?
df_sam['가격(원)'].max()
```

```
35890
```

```
df_sam[df_sam['가격(원)']<5000]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	일련번호	시장/마트번호	시장/마트이름	품목번호	품목이름	실판매규격	가격(원)	년도-월	비고	시장유형구분(시장/마트)코드	시장유형구분(시장/마트)이름	자치구코드	자치구이름

```
# 삼겹살 600g의 최저 가격은?
df_sam['가격(원)'].min()
```

```
1690
```

우리동네 삼겹살 가격

```
gu = input('구이름:')
```

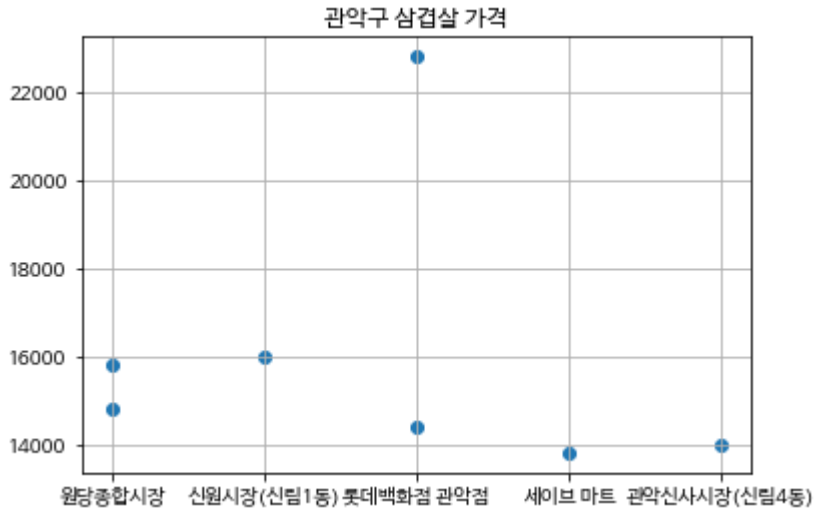
```
구이름:관악구
```

```
# 우리구 삼겹살 가격
df_sam_gu = df_sam[df_sam['자치구 이름']==gu][['시장/마트 이름','품목 이름','실판매규격','가격(원)']].drop_duplicates()
df_sam_gu
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	시장/마트 이름	품목 이름	실판매규격	가격(원)
2879	원당종합시장	돼지고기(생삼겹살)	600g	14800
2889	원당종합시장	돼지고기(생삼겹살)	600g	15800
2978	신원시장(신림1동)	돼지고기(생삼겹살)	600g	16000
3212	롯데백화점 관악점	돼지고기(생삼겹살)	600g	22800
3231	롯데백화점 관악점	돼지고기(생삼겹살)	600g	14400
3383	세이브 마트	돼지고기(생삼겹살)	600g	13800
5503	관악신사시장(신림4동)	돼지고기(생삼겹살)	600g	14000

```
# 시각화
x = df_sam_gu['시장/마트 이름']
y = df_sam_gu['가격(원)']
plt.scatter(x,y)
plt.title(gu+' 삼겹살 가격')
plt.grid(True)
plt.show()
```



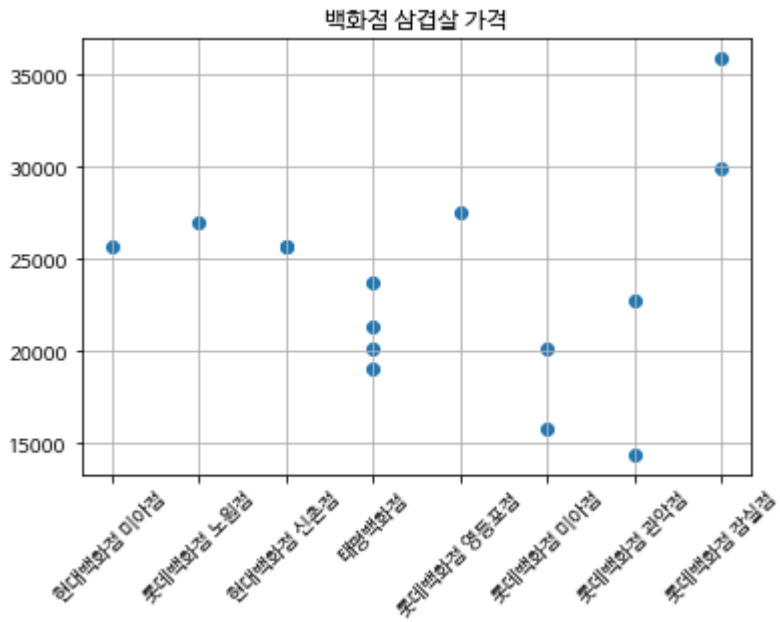
마트 지점별 삼겹살 가격

```
mart = input('시장/마트이름:')
```

시장/마트이름:백화점

```
# 마트 지점별 삼겹살 가격
df_sam_mart = df_sam[df_sam['시장/마트 이름'].str.contains(mart)][['시장/마트 이름','품목 이름','실판매규격','가격(원)']].drop_duplicates()
```

```
# 시각화
df_sam_mart
x = df_sam_mart['시장/마트 이름']
y = df_sam_mart['가격(원)']
plt.scatter(x,y)
plt.grid(True)
plt.title(mart+ ' 삼겹살 가격')
plt.xticks(rotation=45)
plt.show()
```



달걀 가격 분석

달걀 데이터

```
# 2021-06 데이터 이용
df_egg = df[(df['품목 이름'].str.contains('달걀')) & (df['년도-월']=='2021-06') & (df['실판매규격'].str.contains('30개')) & (df['가격(원)']>0)]
```

```
# 달걀 30구 평균 가격
df_egg['가격(원)'].mean()
```

```
9442.334426229509
```

```
# 달걀 최고 가격
df_egg['가격(원)'].max()
```

```
75000
```

```
df_egg[df_egg['가격(원)'] < 6000]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	일련번호	시장/마트번호	시장/마트이름	품목번호	품목이름	실판매규격	가격(원)	년도-월	비고	시장유형구분(시장/마트)코드	시장유형구분(시장/마트)이름	자치구코드	자치구이름

```
# 달걀 최저 가격
df_egg['가격(원)'].min()
```

```
5520
```

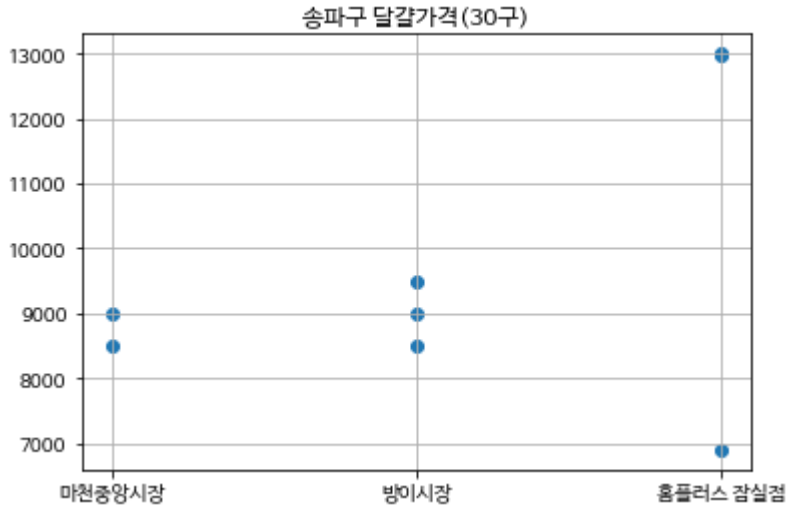
우리동네 달걀 가격

```
gu = input('구이름:')
```

```
구이름:송파구
```

```
# 우리구 달걀 가격
df_egg_gu = df_egg[df_egg['자치구 이름']==gu][['시장/마트 이름','품목 이름','실판매규격','가격(원)']].drop_duplicates()
```

```
# 시각화
df_egg_gu.sort_values('가격(원)')
x = df_egg_gu['시장/마트 이름']
y = df_egg_gu['가격(원)']
plt.scatter(x,y)
plt.grid(True)
plt.title(gu+' 달걀가격(30구)')
plt.show()
```



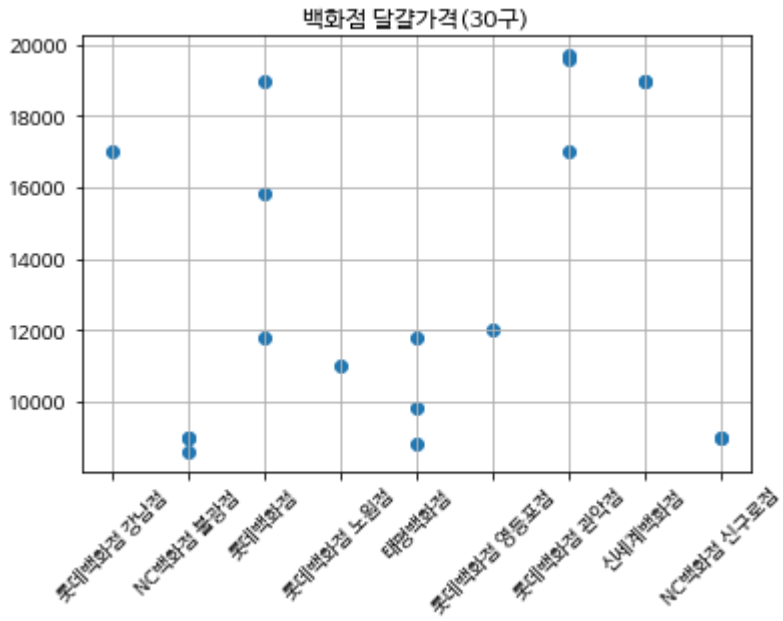
마트 지점별 달걀 가격

```
gu = input('마트이름:')
```

```
마트이름:백화점
```

```
# 마트 지점별 달걀 가격
df_egg_mart = df_egg[df_egg['시장/마트 이름'].str.contains(gu)][['시장/마트 이름','품목 이름','실판매규격','가격(원)']].drop_duplicates()
```

```
# 시각화
x = df_egg_mart['시장/마트 이름']
y = df_egg_mart['가격(원)']
plt.scatter(x,y)
plt.xticks(rotation=45)
plt.grid(True)
plt.title(gu+' 달걀가격(30구)')
plt.show()
```



지하철 승하차 현황 분석

- 데이터 수집

  http://data.seoul.go.kr/dataList/OA-12252/S/1/datasetView.do

```
import pandas as pd
```

```
# 그래프를 노트북 안에 그리기 위해 설정
%matplotlib inline

# 필요한 패키지와 라이브러리 가져온다.
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

# 그래프에서 마이너스 폰트 깨지는 문제에 대한 대처
mpl.rcParams['axes.unicode_minus'] = False
```

```
# 폰트 지정하기
plt.rcParams['font.family'] = 'gulim'
```

데이터 확인 및 전처리

- 월별, 시간대별, 역별 지하철 승하차정보 데이터이다.

```
df = pd.read_csv('data/서울시 지하철 호선별 역별 시간대별 승하차 인원 정보.csv',
encoding='cp949')
```

```
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04시-05시 승차인원	04시-05시 하차인원	05시-06시 승차인원	05시-06시 하차인원	06시-07시 승차인원	06시-07시 하차인원	07시-08시 승차인원	...	23시-24시 하차인원
0	202108	1호선	동대문	415	11	10380	1815	7880	5352	12037	...	5983
1	202108	1호선	동묘앞	84	3	2861	918	3286	4614	5128	...	1759

5 rows × 52 columns

```
df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04시-05시 승차인원	04시-05시 하차인원	05시-06시 승차인원	05시-06시 하차인원	06시-07시 승차인원	06시-07시 하차인원	07시-08시 승차인원	...	23시 시 차 원
46547	201501	중앙선	운길산	0	0	177	56	642	403	1292	...	6
46548	201501	중앙선	원덕	1	0	292	4	415	46	537	...	3

5 rows × 52 columns

데이터 크기

```
df.shape
```

```
(46552, 52)
```

결측치 확인

```
df.isnull().sum()
```

```
사용월                0
호선명                0
지하철역               0
04시-05시 승차인원     0
04시-05시 하차인원     0
05시-06시 승차인원     0
05시-06시 하차인원     0
06시-07시 승차인원     0
06시-07시 하차인원     0
07시-08시 승차인원     0
07시-08시 하차인원     0
08시-09시 승차인원     0
08시-09시 하차인원     0
09시-10시 승차인원     0
```

```
09시-10시 하차인원    0
10시-11시 승차인원    0
10시-11시 하차인원    0
11시-12시 승차인원    0
11시-12시 하차인원    0
12시-13시 승차인원    0
12시-13시 하차인원    0
13시-14시 승차인원    0
13시-14시 하차인원    0
14시-15시 승차인원    0
14시-15시 하차인원    0
15시-16시 승차인원    0
15시-16시 하차인원    0
16시-17시 승차인원    0
16시-17시 하차인원    0
17시-18시 승차인원    0
17시-18시 하차인원    0
18시-19시 승차인원    0
18시-19시 하차인원    0
19시-20시 승차인원    0
19시-20시 하차인원    0
20시-21시 승차인원    0
20시-21시 하차인원    0
21시-22시 승차인원    0
21시-22시 하차인원    0
22시-23시 승차인원    0
22시-23시 하차인원    0
23시-24시 승차인원    0
23시-24시 하차인원    0
00시-01시 승차인원    0
00시-01시 하차인원    0
01시-02시 승차인원    0
01시-02시 하차인원    0
02시-03시 승차인원    0
02시-03시 하차인원    0
03시-04시 승차인원    0
03시-04시 하차인원    0
작업일자              0
dtype: int64
```

컬럼별 데이터 확인

```
# 사용월
df['사용월'].unique()
```

```
array([202108, 202107, 202106, 202105, 202104, 202103, 202102, 202101,
       202012, 202011, 202010, 202009, 202008, 202007, 202006, 202005,
       202004, 202003, 202002, 202001, 201912, 201911, 201910, 201909,
       201908, 201907, 201906, 201905, 201904, 201903, 201902, 201901,
```

```
       201812, 201811, 201810, 201809, 201808, 201807, 201806, 201805,
       201804, 201803, 201802, 201801, 201712, 201711, 201710, 201709,
       201708, 201707, 201706, 201705, 201704, 201703, 201702, 201701,
       201612, 201611, 201610, 201609, 201608, 201607, 201606, 201605,
       201604, 201603, 201602, 201601, 201512, 201511, 201510, 201509,
       201508, 201507, 201506, 201505, 201504, 201503, 201502, 201501],
      dtype=int64)
```

```
# 호선명
df['호선명'].unique()
```

```
array(['1호선', '2호선', '3호선', '4호선', '5호선', '6호선', '7호선', '8호선',
'9호선',
       '9호선2~3단계', '경강선', '경부선', '경원선', '경의선', '경인선', '경춘선',
'공항철도 1호선',
       '과천선', '분당선', '수인선', '안산선', '우이신설선', '일산선', '장항선',
'중앙선', '9호선2단계'],
      dtype=object)
```

데이터 타입 확인 및 변경

```
df.dtypes
```

```
사용월                int64
호선명               object
지하철역              object
04시-05시 승차인원      int64
04시-05시 하차인원      int64
05시-06시 승차인원      int64
05시-06시 하차인원      int64
06시-07시 승차인원      int64
06시-07시 하차인원      int64
07시-08시 승차인원      int64
07시-08시 하차인원      int64
08시-09시 승차인원      int64
08시-09시 하차인원      int64
09시-10시 승차인원      int64
09시-10시 하차인원      int64
10시-11시 승차인원      int64
10시-11시 하차인원      int64
11시-12시 승차인원      int64
11시-12시 하차인원      int64
12시-13시 승차인원      int64
12시-13시 하차인원      int64
```

```
13시-14시 승차인원     int64
13시-14시 하차인원     int64
14시-15시 승차인원     int64
14시-15시 하차인원     int64
15시-16시 승차인원     int64
15시-16시 하차인원     int64
16시-17시 승차인원     int64
16시-17시 하차인원     int64
17시-18시 승차인원     int64
17시-18시 하차인원     int64
18시-19시 승차인원     int64
18시-19시 하차인원     int64
19시-20시 승차인원     int64
19시-20시 하차인원     int64
20시-21시 승차인원     int64
20시-21시 하차인원     int64
21시-22시 승차인원     int64
21시-22시 하차인원     int64
22시-23시 승차인원     int64
22시-23시 하차인원     int64
23시-24시 승차인원     int64
23시-24시 하차인원     int64
00시-01시 승차인원     int64
00시-01시 하차인원     int64
01시-02시 승차인원     int64
01시-02시 하차인원     int64
02시-03시 승차인원     int64
02시-03시 하차인원     int64
03시-04시 승차인원     int64
03시-04시 하차인원     int64
작업일자               int64
dtype: object
```

```
df['사용월'] = df['사용월'].astype('str')
```

```
df.dtypes
```

```
사용월               object
호선명               object
지하철역              object
04시-05시 승차인원     int64
04시-05시 하차인원     int64
05시-06시 승차인원     int64
05시-06시 하차인원     int64
06시-07시 승차인원     int64
06시-07시 하차인원     int64
07시-08시 승차인원     int64
```

```
07시-08시 하차인원     int64
08시-09시 승차인원     int64
08시-09시 하차인원     int64
09시-10시 승차인원     int64
09시-10시 하차인원     int64
10시-11시 승차인원     int64
10시-11시 하차인원     int64
11시-12시 승차인원     int64
11시-12시 하차인원     int64
12시-13시 승차인원     int64
12시-13시 하차인원     int64
13시-14시 승차인원     int64
13시-14시 하차인원     int64
14시-15시 승차인원     int64
14시-15시 하차인원     int64
15시-16시 승차인원     int64
15시-16시 하차인원     int64
16시-17시 승차인원     int64
16시-17시 하차인원     int64
17시-18시 승차인원     int64
17시-18시 하차인원     int64
18시-19시 승차인원     int64
18시-19시 하차인원     int64
19시-20시 승차인원     int64
19시-20시 하차인원     int64
20시-21시 승차인원     int64
20시-21시 하차인원     int64
21시-22시 승차인원     int64
21시-22시 하차인원     int64
22시-23시 승차인원     int64
22시-23시 하차인원     int64
23시-24시 승차인원     int64
23시-24시 하차인원     int64
00시-01시 승차인원     int64
00시-01시 하차인원     int64
01시-02시 승차인원     int64
01시-02시 하차인원     int64
02시-03시 승차인원     int64
02시-03시 하차인원     int64
03시-04시 승차인원     int64
03시-04시 하차인원     int64
작업일자              int64
dtype: object
```

불필요한 컬럼 삭제

```
df.drop(columns=['작업일자'], inplace=True)
```

승차/하차 테이블 분리

승차 테이블 만들기

```
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04 시-05 시 승차인원	04 시-05 시 하차인원	05 시-06 시 승차인원	05 시-06 시 하차인원	06 시-07 시 승차인원	06 시-07 시 하차인원	07 시-08 시 승차인원	...	23 시 시 차 원
0	202108	1호선	동대문	415	11	10380	1815	7880	5352	12037	...	1
1	202108	1호선	동묘앞	84	3	2861	918	3286	4614	5128	...	5

46552 rows × 51 columns

```
# 공통 컬럼
df1 = df.iloc[:,:3]

# 승차 컬럼만 가져오기
df2 = df.iloc[:,3::2]
df2.columns = df2.columns.str.split(' ').str[0]
```

```
# 공통컬럼과 승차컬럼 연결하기
df_in = pd.concat([df1,df2], axis=1)
df_in
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04시-05시	05시-06시	06시-07시	07시-08시	08시-09시	09시-10시	10시-11시	...
0	202108	1호선	동대문	415	10380	7880	12037	16637	15671	13922	...
1	202108	1호선	동묘앞	84	2861	3286	5128	8066	7541	9130	...

46552 rows × 27 columns

하차 테이블 만들기

```
# 공통 컬럼
df1 = df.iloc[:,:3]

# 하차 컬럼만 가져오기
df2 = df.iloc[:,4::2]
df2.columns = df2.columns.str.split(' ').str[0]
```

```
# 공통컬럼과 하차컬럼 연결하기
df_out = pd.concat([df1,df2], axis=1)
df_out
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04시-05시	05시-06시	06시-07시	07시-08시	08시-09시	09시-10시	10시-11시	...

	사용월	호선명	지하철역	04시-05시	05시-06시	06시-07시	07시-08시	08시-09시	09시-10시	10시-11시	...
0	202108	1호	동대	11	1815	5352	9885	19783	19192	17298	...

46552 rows × 27 columns

출퇴근시간 역별 승하차인원 분석

- 최근 월을 기준으로 한 승하차 데이터프레임 생성

```
df_in_202108 = df_in[df_in['사용월']=='202108']
df_out_202108 = df_out[df_out['사용월']=='202108']
```

```
df_out_202108
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04시-05시	05시-06시	06시-07시	07시-08시	08시-09시	09시-10시	10시-11시	...	18시
0	202108	1호선	동대문	11	1815	5352	9885	19783	19192	17298	...	16
1	202108	1호선	동묘앞	3	918	4614	8004	18779	15213	17602	...	9

607 rows × 27 columns

- 출근시간에 가장 많은 사람이 승차하는 역은 어디일까? (08시-09시)

```
df_in_202108.nlargest(10,'08시-09시')[['지하철역','08시-09시']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	지하철역	08시-09시
37	신림	267128
14	구로디지털단지	153411
28	서울대입구(관악구청)	150090
83	연신내	132615
133	까치산	130443
52	잠실(송파구청)	128005
175	화곡	121310

- 출근시간에 가장 많은 사람이 하차하는 역은 어디일까?(09시-10시)

```
df_out_202108.nlargest(10,'09시-10시')[['지하철역','09시-10시']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	지하철역	09시-10시
10	강남	274355
43	역삼	226437
215	가산디지털단지	188412
30	선릉	178766
26	삼성(무역센터)	172539
80	압구정	139283

	지하철역	09시-10시

- 퇴근시간에 가장 많은 사람이 승차하는 역은 어디일까?(18시-19시)

```
df_in_202108.nlargest(10,'18시-19시')[['지하철역','18시-19시']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	지하철역	18시-19시
10	강남	302398
215	가산디지털단지	283125
30	선릉	245318
43	역삼	239209
26	삼성(무역센터)	230299
50	을지로입구	207093
31	성수	202624

- 퇴근시간에 가장 많은 사람이 하차하는 역은 어디일까?(19시-20시)

```
df_out_202108.nlargest(10,'19시-20시')[['지하철역','19시-20시']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	지하철역	19시-20시
37	신림	182832
28	서울대입구(관악구청)	121426

	지하철역	19시-20시
52	잠실(송파구청)	116568
14	구로디지털단지	107520
83	연신내	103283

강남역의 최근 시간대별 승하차정보 분석

강남역의 최근 승차정보 분석

```
# 강남역의 최근 승차 데이터 불러오기
df_gangnam_in = df_in_202108[df_in_202108['지하철역']=='강남'].iloc[:,3:]
df_gangnam_in
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	04 시-05 시	05 시-06 시	06 시-07 시	07 시-08 시	08 시-09 시	09 시-10 시	10 시-11 시	11 시-12 시	12 시-13 시	13 시-14 시	...
10	64	4993	14177	31783	50239	40611	44108	59573	75942	88926	...

1 rows × 24 columns

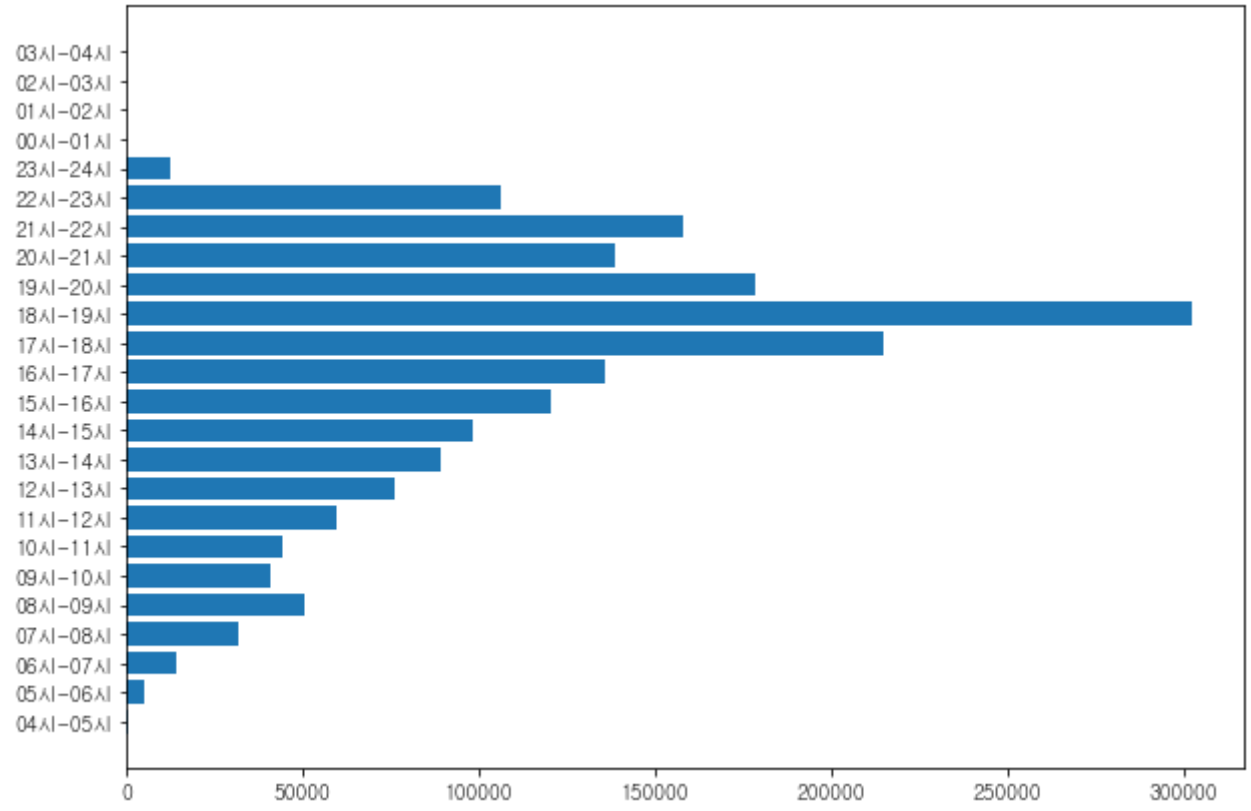
```
# melt
df_gangnam_in = df_gangnam_in.melt()
```

```
# 컬럼명 변경
df_gangnam_in.columns=['시간대','승차건수']
df_gangnam_in.sort_values('승차건수')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	시간대	승차건수
23	03시-04시	0
22	02시-03시	0
21	01시-02시	2
20	00시-01시	5
0	04시-05시	64
1	05시-06시	4993
19	23시-24시	12509

```
# 시간대별 승차인원 시각화하기
df_gangnam_in
plt.figure(figsize=(10,7))
plt.barh(df_gangnam_in['시간대'], df_gangnam_in['승차건수'])
plt.show()
```



강남역의 최근 하차 정보 분석

```
# 강남역의 최근 하차 데이터 불러오기
df_gangnam_out = df_out_202108[df_out_202108['지하철역']=='강남'].iloc[:,3:]
df_gangnam_out
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	04 시-05 시	05 시-06 시	06 시-07 시	07 시-08 시	08 시-09 시	09 시-10 시	10 시-11시	11 시-12 시	12 시-13 시	13 시-14 시
10	4	10411	45059	129595	269877	274355	129744	99120	96644	115153

1 rows × 24 columns

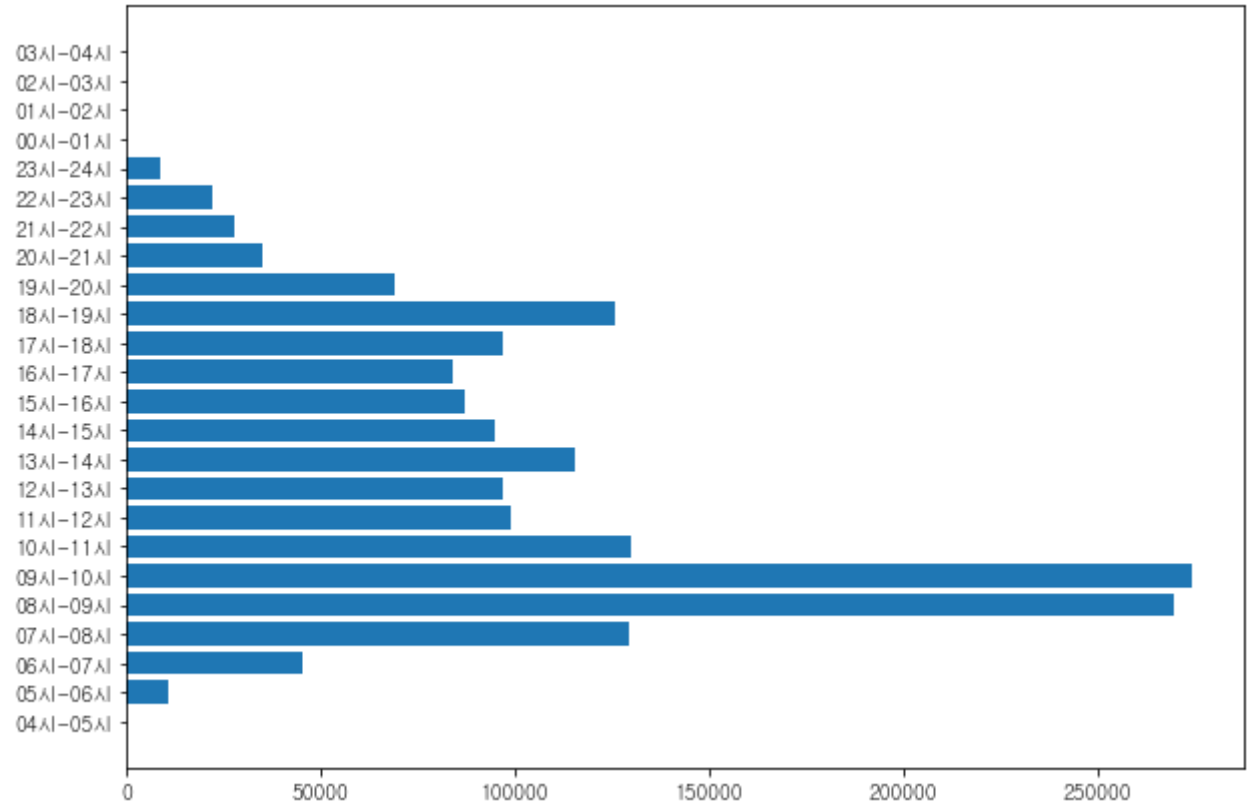
```
# melt
df_gangnam_out = df_gangnam_out.melt()
```

```
# 컬럼명 변경
df_gangnam_out.columns=['시간대','하차건수']
df_gangnam_out.sort_values('하차건수')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	시간대	하차건수
23	03시-04시	0
22	02시-03시	0
21	01시-02시	1
0	04시-05시	4
20	00시-01시	15
19	23시-24시	8377
1	05시-06시	10411

```
# 시간대별 승차인원 시각화하기
df_gangnam_out
plt.figure(figsize=(10,7))
plt.barh(df_gangnam_out['시간대'], df_gangnam_out['하차건수'])
plt.show()
```



지하철 시간대별, 역별 이용현황 분석

시간대별 승차 현황

승차정보 집계 데이터 만들기

```
# df_in_202108카피하여 사용하기
df_in_202108_agg = df_in_202108.copy()
```

```
df_in_202108_agg
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04 시-05 시	05 시-06 시	06 시-07 시	07 시-08 시	08 시-09 시	09 시-10 시	10 시-11 시	...	18 시- 시
0	202108	1호선	동대문	415	10380	7880	12037	16637	15671	13922	...	20

607 rows × 27 columns

```
# 인덱스 변경('지하철역')
df_in_202108_agg.index = df_in_202108_agg['지하철역']
df_in_202108_agg
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04 시-05 시	05 시-06 시	06 시-07 시	07 시-08 시	08 시-09 시	09 시-10 시	10 시-11 시	...	18 시-1 시
지하철역												
동대문	202108	1호선	동대문	415	10380	7880	12037	16637	15671	13922	...	203

607 rows × 27 columns

```
# 컬럼 삭제('사용월','호선명','지하철역')
df_in_202108_agg.drop(columns= ['사용월','호선명','지하철역'], inplace=True)
```

```
# 행,열 합계
df_in_202108_agg.loc['sum'] = df_in_202108_agg.apply('sum', axis=0)
```

```
df_in_202108_agg['sum'] = df_in_202108_agg.apply('sum',axis=1)
```

```
df_in_202108_agg
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

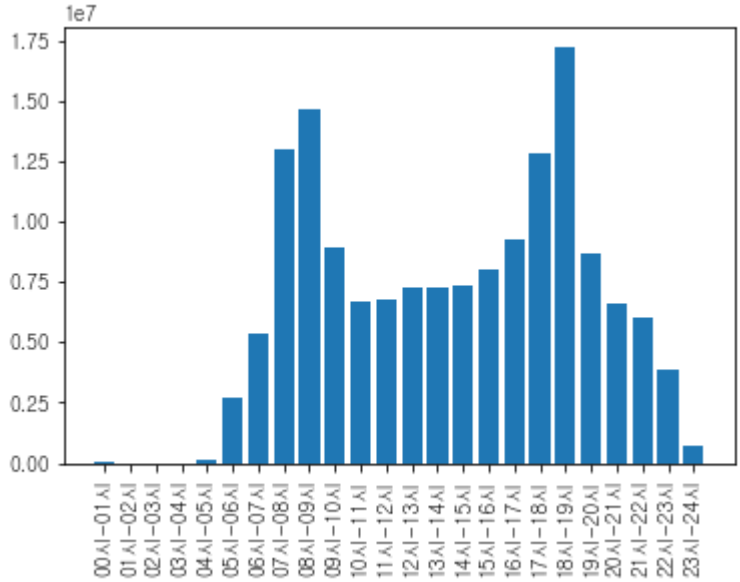
	04시-05시	05시-06시	06시-07시	07시-08시	08시-09시	09시-10시	10시-11시	11시-12시
지하철역								
동대문	415	10380	7880	12037	16637	15671	13922	15675
동묘앞	84	2861	3286	5128	8066	7541	9130	13369
서울								

608 rows × 25 columns

시간대별 승차건수

```
s_in = df_in_202108_agg.loc['sum'][:-1].sort_values()
```

```
s_in = s_in.sort_index()
s_in
x = s_in.index
y = s_in.values
plt.bar(x,y)
plt.xticks(rotation=90)
plt.show()
```



지하철역별 승차건수

```
df_in_202108_agg['sum'][:-1].sort_values(ascending=False).to_frame()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sum
지하철역	
강남	1874711
잠실(송파구청)	1539356
신림	1503604
구로디지털단지	1331304
홍대입구	1247932
...	...

607 rows × 1 columns

시간대별 하차 현황

하차정보 집계 테이블 만들기

```
# df_out_202108카피하여 사용하기
df_out_202108_agg = df_out_202108.copy()
```

```
# 인덱스 변경('지하철역')
df_out_202108_agg.index = df_out_202108_agg['지하철역']
df_out_202108_agg
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	사용월	호선명	지하철역	04 시-05 시	05 시-06 시	06 시-07 시	07 시-08 시	08 시-09 시	09 시-10 시	10 시-11 시	...	18 시-1 시
지하철역												
동대문	202108	1호선	동대문	11	1815	5352	9885	19783	19192	17298	...	163

607 rows × 27 columns

```
# 컬럼 삭제('사용월','호선명','지하철역')
df_out_202108_agg.drop(columns= ['사용월','호선명','지하철역'], inplace=True)
```

```
# 행,열 합계
df_out_202108_agg.loc['sum'] = df_out_202108_agg.apply('sum', axis=0)
```

```
df_out_202108_agg['sum'] = df_out_202108_agg.apply('sum', axis=1)
```

```
df_out_202108_agg
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	04시-05시	05시-06시	06시-07시	07시-08시	08시-09시	09시-10시	10시-11시	11시-12시	1
지하철역									
동대문	11	1815	5352	9885	19783	19192	17298	19346	
동묘앞	3	918	4614	8004	18779	15213	17602	21823	
서울									

608 rows × 25 columns

시간대별 하차 건수

```
df_out_202108_agg.loc['sum'].sort_values()
```

```
03시-04시            1
02시-03시           12
01시-02시           66
04시-05시         1644
00시-01시       214094
05시-06시       706189
23시-24시      2216218
06시-07시      3603606
22시-23시      5217987
21시-22시      6338999
11시-12시      6666215
12시-13시      7062738
20시-21시      7093483
10시-11시      7146549
14시-15시      7174578
13시-14시      7449305
15시-16시      7581218
07시-08시      8042590
```

```
16시-17시      8380236
17시-18시     10646691
09시-10시     11402630
19시-20시     12354319
18시-19시     16007443
08시-09시     17604264
sum         152911075
Name: sum, dtype: int64
```

지하철역별 하차건수

```
df_out_202108_agg['sum'][:-1].sort_values(ascending=False)[:10].to_frame()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sum
지하철역	
강남	1819150
잠실(송파구청)	1514327
신림	1467939
구로디지털단지	1339493
홍대입구	1273453
역삼	1154897

[학습목표]

멧플롯립으로 그래프를 그리는 기본 방법을 이해한다.

라이브러리 임포트

- matplotlib의 pyplot 모듈 사용
- 관용적으로 plt라는 별칭 사용

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

기본 그래프 그리기

- plt.plot(data)
- plot에 데이터를 전달하여 그린다.

y축 데이터로 그리기

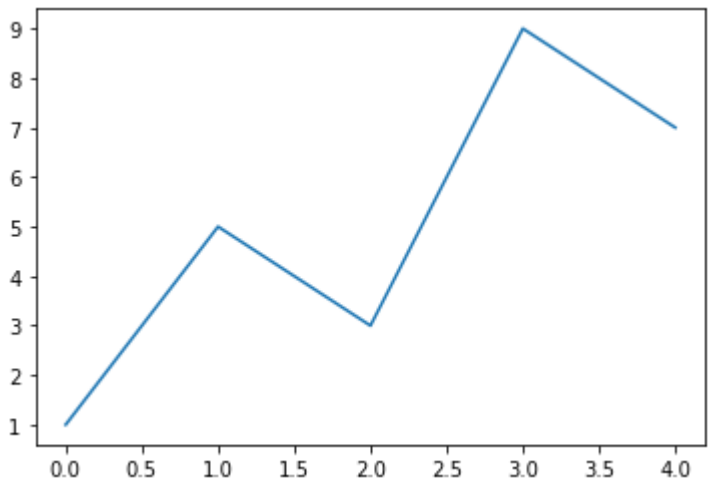
- 1차원 리스트, 튜플, 시리즈 데이터를 전달하여 그린다.
- x축은 데이터의 인덱스로 자동 지정된다.

리스트 데이터로 그리기

```
# y축 데이터 [1,5,3,9,7]
data = [1,5,3,9,7]
```

```
# 그래프 그리기
plt.plot(data)
```

```
[]
```



시리즈 데이터로 그리기

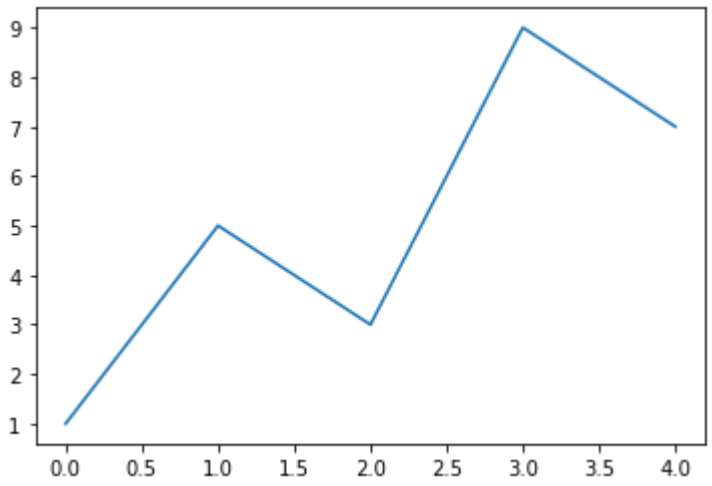
```
# pandas import
import pandas as pd
```

```
# 시리즈 데이터 만들기 ([1,5,3,9,7])
s = pd.Series([1,5,3,9,7])
s
```

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

```
# 그래프 그리기 (plot)
plt.plot(s)
```

```
[]
```

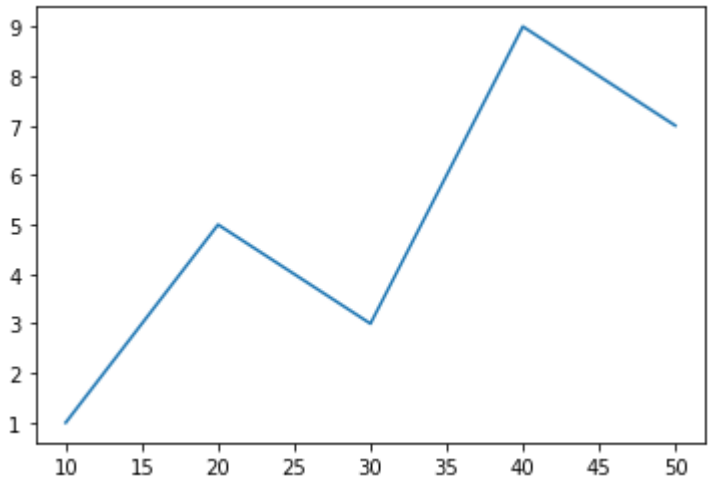



```
# 시리즈 데이터의 인덱스 변경하기 : index = [10,20,30,40,50]
ss = pd.Series([1,5,3,9,7], index=[10,20,30,40,50])
ss
```

```
10    1
20    5
30    3
40    9
50    7
dtype: int64
```

```
# 그래프 그리기
plt.plot(ss)
```

```
[]
```

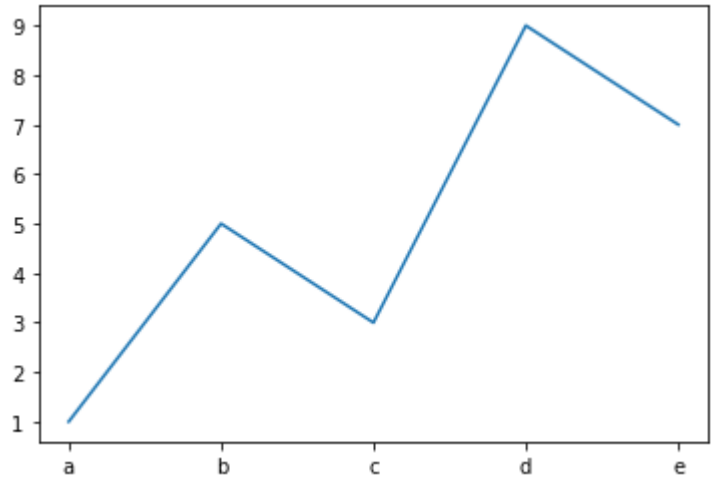



```
# 시리즈 데이터의 인덱스 변경하기 : index = ['a','b','c','d','e']
sss = pd.Series([1,5,3,9,7], index=['a','b','c','d','e'])
sss
```

```
a    1
b    5
c    3
d    9
e    7
dtype: int64
```

```
# 그래프 그리기
plt.plot(sss)
```

```
[]
```



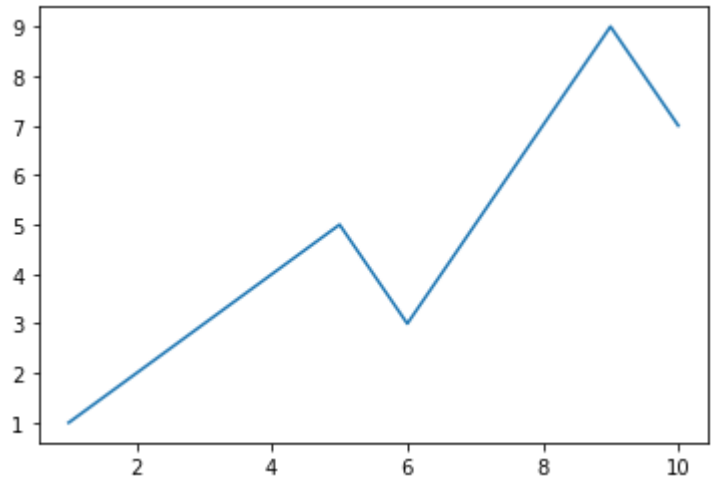
x축, y축 데이터로 그리기

- x축과 y축 데이터의 길이가 같아야 한다.

리스트데이터로 그리기

```
x = [1,5,6,9,10]
y = [1,5,3,9,7]

# 그래프 그리기
plt.plot(x,y)
```

```
[]
```



데이터프레임 데이터로 그리기

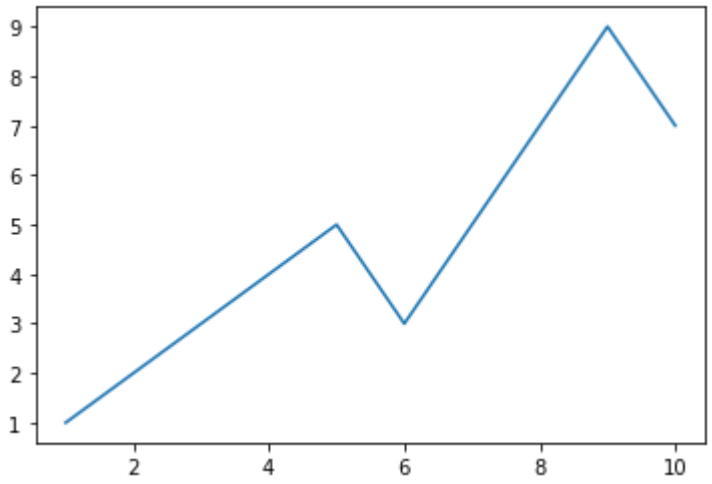
```
sss = pd.DataFrame({'x':[1,5,6,9,10], 'y':[1,5,3,9,7]})
sss
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	x	y
0	1	1
1	5	5
2	6	3
3	9	9
4	10	7

```
# 그래프 그리기
x = sss['x']
y = sss['y']
plt.plot(x,y)
```

```
[]
```



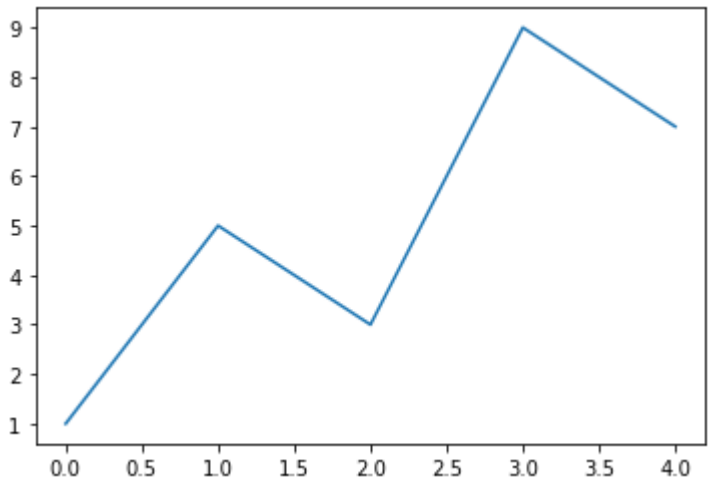
시리즈 데이터로 그리기

```

```

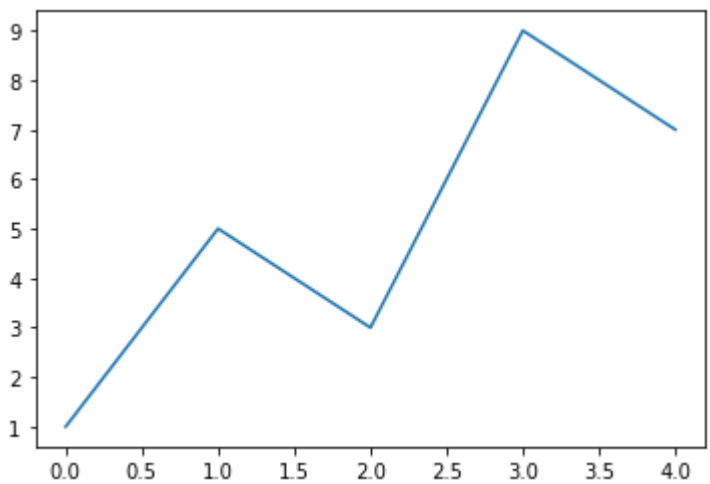
```
s = pd.Series([1,5,3,9,7])
s
```

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

```
# 시리즈만 전달하여 그리기
plt.plot(s)
plt.show()
```




```
# x, y 각각 전달하여 그리기
plt.plot(s.index, s)
plt.show()
```



[학습목표]

목적에 따른 여러가지 그래프를 그리는 기본 방법을 이해한다.

라이브러리 임포트

- matplotlib의 pyplot 모듈 사용
- 관용적으로 plt라는 별칭 사용

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

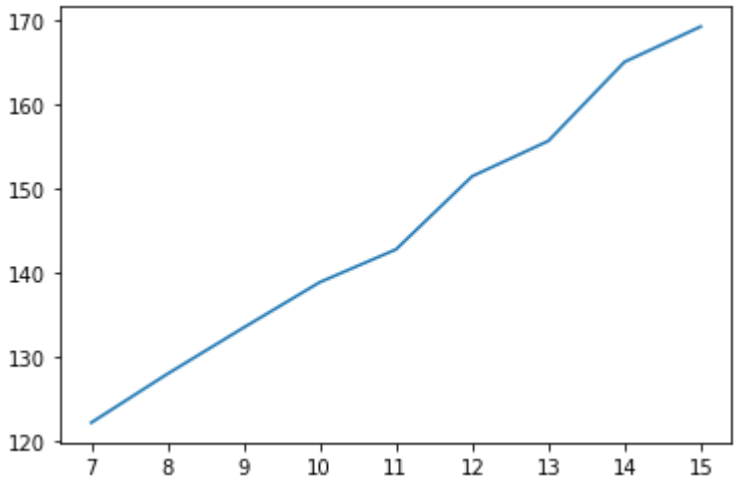
시간의 시각화

```
# 샘플데이터 : 어떤 아이의 나이별 키의 변화
age = [7,8,9,10,11,12,13,14,15]
height = [122.1, 127.9, 133.4, 138.8, 142.7, 151.4, 155.6, 165.0, 169.2]
```

선그래프

```
plt.plot(age, height)
```

```
[]
```



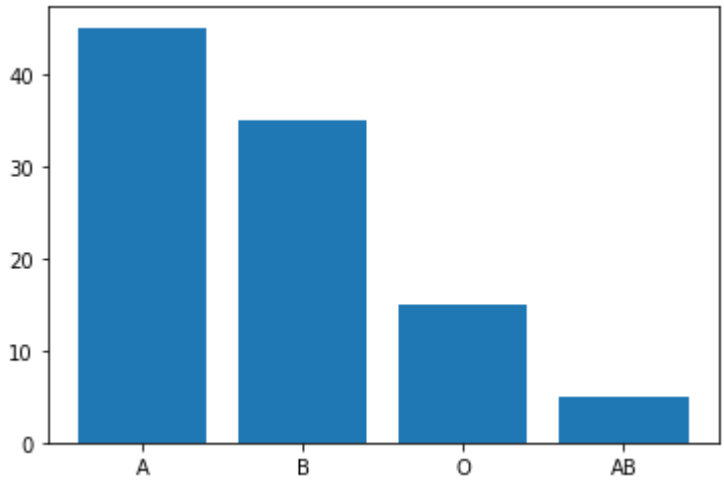
수량비교 시각화

```
# 샘플데이터 : 100명 중 혈액형 인원 비교
blood_type=['A','B','O','AB']
count=[45,35,15,5]
```

막대 그래프

```
plt.bar(blood_type,count)
```

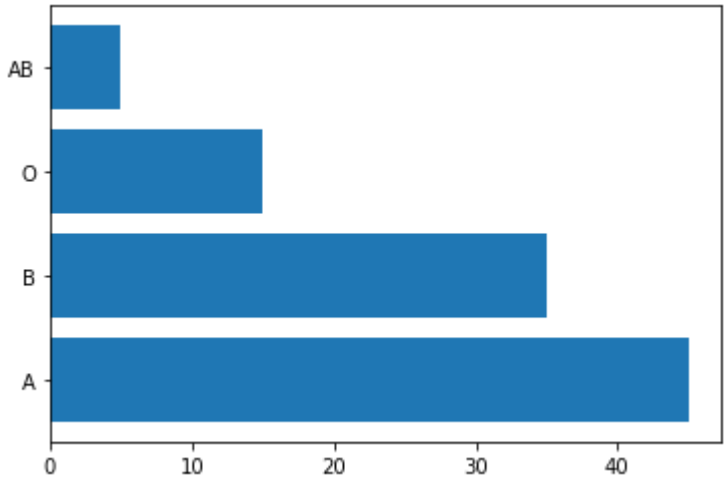
```

```



가로 막대 그래프

```
plt.barh(blood_type,count)
```

```

```



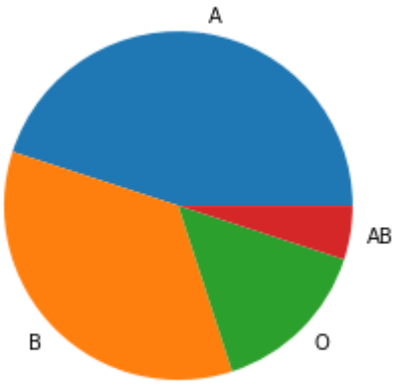
비율의 시각화

```
# 샘플데이터 : 100명 중 혈액형 인원의 비율
blood_type=['A','B','O','AB']
count=[45,35,15,5]
```

파이차트

- pie

```
plt.pie(count, labels=blood_type)
plt.show()
```



분포의 시각화

```
# 샘플데이터 (1~100사이의 랜덤 정수 1000개)
import numpy as np
```

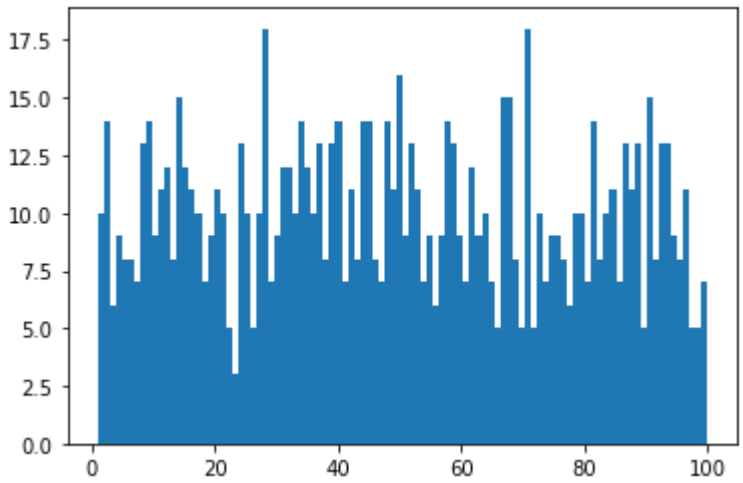
```
data = np.random.randint(1,101,1000)
data
```

```
array([ 43,  71,  44,  30,  56,  95,  42,  94,  99,  81,  83,  78,  16,
        27,  10,  71,  52,  12,   1,  18,  43,  71,  91,  13,  82,   6,
        50,  37,  28,  76,  94,  62,  70,  32,   3,  85,  99,  36,  67,
        66,  84,  82,  36,  14,  55,  27,  37,  46,  53,  37,  58,  28,
        73,  84,  24,  52,  62,  97,  20,  89,  33,  56,  50,  62,  64,
        59,   2,  68,  59,  26,  75,  37,  63,  24,  97,  59,  99,  89,
        82,  62,  82,  34,  88,  19,  45,  47,  91,  34,  14,  34,  16,
         2,  38,  36,  43,   8,  33,  79,  62,  78,  58,  60,  44,  77,
        48,   7, 100,  90,  73,  67,  39,  39,  85,  44,  14,  52,  79,
        14,  32,  83,  92,  69,  95,  60,  78,  71,  72,  43,  75,  26,
        71,   8,  39,   7,  91,  82,  46,  60,  55,  72, 100,  72,  71,
        62,   4,  87,  36,  14,  34,  93,  97,  85,  60,  70,  27,  17,
        67,   1,   7,  88,  91,  48,  21,  24,  95,  29,  30,  69,   9,
         1,  67,  27,  37,  53, 100,  50,  55,  11,   8,   4,  18,  21,
        22,  93,  45,  37,  86,  75,  25,  63,  91,  91,  47,  56,  52,
        59,  60,  93,  50,  30,  87,  66,   8,  59,  97,  36,  93,  76,
        91,  88,  27,  12,  16,  31,  28,  53,  64,  87,  42,  84,   2,
        89,  55,  40,  93,  43,  80,  46,  36,  17,  43,  91,  32,  20,
         2,  36,  89,  39,  98,  96,  31,  21,  87,  28,  77,   9,  90,
        96,  23,  45,  94,  15,  46,  37,  96,  19,   8,   7,  34,  13,
        71,  35,   9,   4,  10,  74,  82,  84,  85,  34,  41,   9,  73,
        44,  54,  51,  48,   2,  92,  25,  55,  44,  34,  34,  97,  63,
        26,  57,  40,  15,  68,  39,   4,  82,  77,  24,  90,  17,  85,
        11,  35,  15,  40,  36,  12,  49,  99,  44,  38,  18,   1,  55,
        67,  70,  50,  91,  57,  59,  72,  19,  24,   3,  88,  13,  16,
        86,  68,  61,  17,  97,  22,   2,  33,  12,  74,  63,  50,  40,
        33,  40,  41,  12,  74,  84,  52,  88,  26,  68,   9,  44,  37,
        40,  45,  68,   5,  14,  56,  20,  94,  10,  31,   8,  36,  21,
        67,  50,  59,  71,  81,  18,  23,  21,  58,  15,  28,  54,  32,
        36,  58,  35,  62,  76,  94,  15,  42,  85,  71,  48,  48,  55,
        48,   3,  28,  25,  70,  35,  12,  44,  61,  99,  66,  42,  62,
        13,   5,  68,  87,  76,  19,  84,   4,  67,  63,  32,  39,  51,
        56,  58,  19,  16,  32,  66,  81,  28,  33,  82,  12,  21,  80,
        88,  25,  46,  44,  40,  16,  48,  14,  95,   9,  67,  75,  48,
        30,  27,  74,  16,  37,  15,  14,  71,  81,  82,  59,  85,  61,
        33,  98,  94,  64,   2,  84,   8,  32,  49,  92,  87,  17,  64,
        31,  25,  88,  11,  24,  79,  92,  91,  58,  93,   7,  35,   2,
        90,   2,   6,  28,  59,   5,  38,  77,  80,  96,  17,  30,  65,
        37,  68,  93,  13,  34,  57,  14,  31,  84,  61,  89,  56,  52,
        30,  89,  57,  28,  73,  12,  32,  39,  24,  15,  54,  53,  67,
        86,  39,  42,  51,  48,  52,  83,  18,  83,  11,   8,  65,  29,
        11,  85,  54,  25,  89,   4,  58,  67,  71,  98,  39,  21,  59,
        64,  42,  35,  47,  54,  31,  11,   6,   1,   8,  43,  94,  82,
        79,  79,  16,  52,  50,  41,  87,   4,  82,  34, 100,  87,   2,
        65,  93,  53,  94,  39,  29,  35,  92,   9,  28,   2,   3,  51,
        92,  54,  71,  88,  58,  41,  25,  38,  75,  97,  22,  31,   8,
        89,  42,  49,  88,  61,  35,  88,  98,  67,   6,  83,  95,  49,
        28,  24,  69,   9,   9,  24,  76,  80,  27,  60,  40,  33,  52,
```

```
       97,  14,  40,  73,  50,  58,  50,  61,  12,  31,  44,  35,  16,
       44,  50,  33,  31,  87,  87,  12,  81,  29,  95,  28,  38,  68,
        6,  27,  78,  87,  50,  28,  24,  86,  44,  71,  65,  41,  12,
       64,  74,  16,  19,  45,  51,  55,   9,  77,  34,  10,  21,  83,
       11,  59,  89,  95,  58,  89,  19,  39,  37,  49,  45,  75,  49,
       31,  17,   1,  33,  80,  64,  45,  62,  85,  89,  84,  77,  64,
       29,  95,  47,  49,  15,  46,   1,  69,  17,  57,  39,  19,  94,
       11,  80,   5,  44,  17,  77,  74,  58,  67, 100,  88,  93,  22,
       96,  79,  20,  96,   5,  97,   8,  60,  82,  86,  28,  53,  68,
       13,  49,  93,   3,  59,  51,  37,  24,  71,  55,  35,  45,  46,
       80,  35,  19,  22,  82,  40,  85,  49,  49,  32,  23,  33,  68,
       53,  52, 100,  18,  14,  68,  94,  48,   1,  53,  94,  73,  75,
       79,  94,  84,  34, 100,  26,  80,  91,  28,  81,   2,  14,  77,
       87,  73,  28,  58,  20,  14,   8,  97,  20,  92,  97,  53,  20,
       45,  47,  14,  53,  14,  27,  89,  69,  37,  94,  20,  58,  32,
       57,  62,   9,  70,  11,  74,   4,  10,  54,  25,   3,  93,   1,
       45,  17,  34,  48,  28,  41,  80,  69,  30,  93,  11,  76,  51,
       48,  50,  24,  76,  79,   8,  13,  73,  68,  35,   9,   1,  73,
       96,  20,  47,  16,  21,  75,  71,  11,  69,  96,  68,  10,  63,
       82,  89,  91,  41,  86,   2,  15,  29,  78,  40,   9,  10,  59,
       50,  40,  72,  10,  43,  15,  38,  64,  31,   6,  83,  57,  79,
       38,  65,  32,  38,   4,  48,  73,  63,  30,  27,  53,  68,  28,
       50,  83,  62,  52,  71,  61,  45,  75,  95,  63,  65,  71,  67,
       79,  60,  91,  91,  13,  98,  71,  45,  90,  40,  76,  25,  51,
       45,  87,  58,  49,  47,  67,  20,  15,   5,  10,  40,  57,  69,
        7,  76,  46,  42,  52,   5,  51,  42,  57,  24,  91,  65,  45,
       32,   5,  52,  48,  81,  44,  60,  18,  68,  29,  63,  67,   9,
       15,  30,  78,   6,  62,  80,  12,  86,  42,  85,  20,   2,  66,
       50,  93,  34,   6,  25,  64,  42,   7,  31,  92,  21,  39])
```

히스토그램

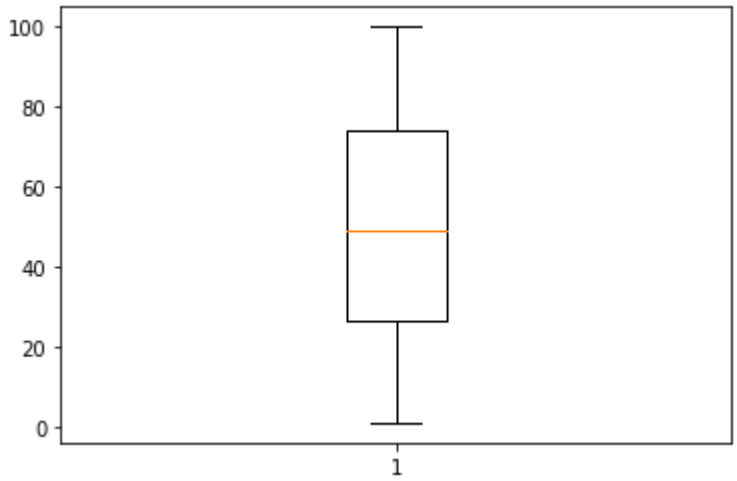
```
plt.hist(data, bins=100)
plt.show()
```



상자수염그래프

- boxplot

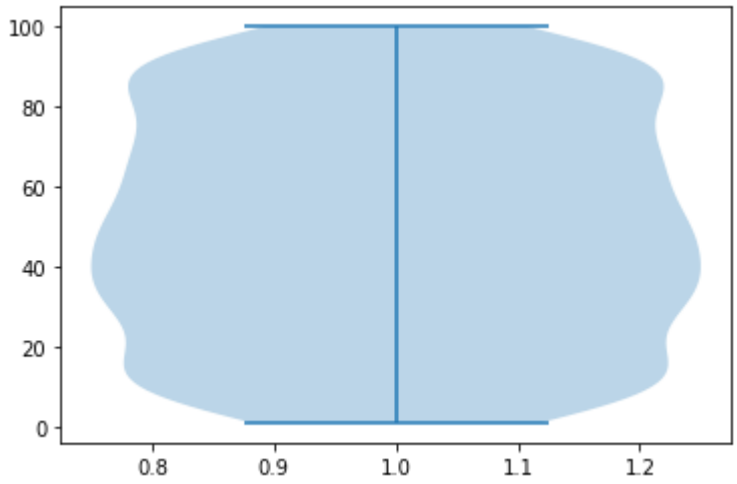
```
plt.boxplot(data)
plt.show()
```



바이올린그래프

- violinplot

```
plt.violinplot(data)
plt.show()
```



관계의 시각화

```
## 샘플데이터 (지불금액에 따른 팁)
import seaborn as sns
tips = sns.load_dataset('tips')[['total_bill','tip']]
tips
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

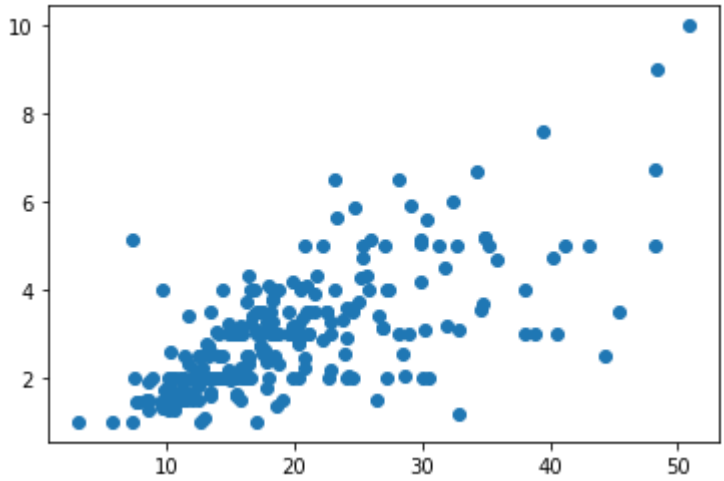
	total_bill	tip
0	16.99	1.01
1	10.34	1.66
2	21.01	3.50
3	23.68	3.31
4	24.59	3.61
...	...	...
239	29.03	5.92

244 rows × 2 columns

산점도

- scatter

```
plt.scatter(tips['total_bill'], tips['tip'])
```



[학습목표]

그래프의 가독성을 높이고 보기좋게 꾸미기 위해 색상, 마커, 선의 스타일을 설정할 수 있다

라이브러리 임포트

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

한글폰트 사용

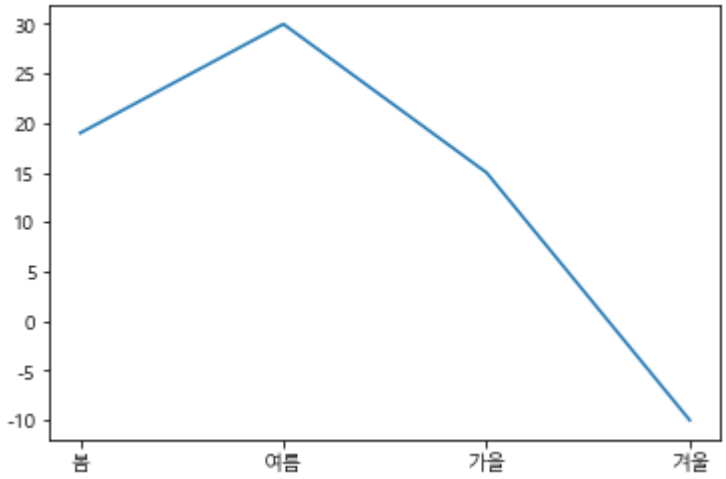
```
# 폰트 확인하기
import matplotlib.font_manager as fm
[f.name for f in fm.fontManager.ttflist if 'Malgun' in f.name]
```

```
['Malgun Gothic',
 'Malgun Gothic',
 'Malgun Gothic',
 'Malgun Gothic',
 'Malgun Gothic',
 'Malgun Gothic']
```

```
# 한글폰트 지정
plt.rcParams['font.family'] = 'Malgun Gothic'

# 한글폰트 사용 시 -기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus']=False
```

```
# 그래프로 확인
x = ['봄','여름','가을','겨울']
y = [19,30,15,-10]

# 그래프 그리기
plt.plot(x,y)
```

```
[]
```



색상, 마커, 선

색상

- color=색상
- 색상 이름을 사용한다.

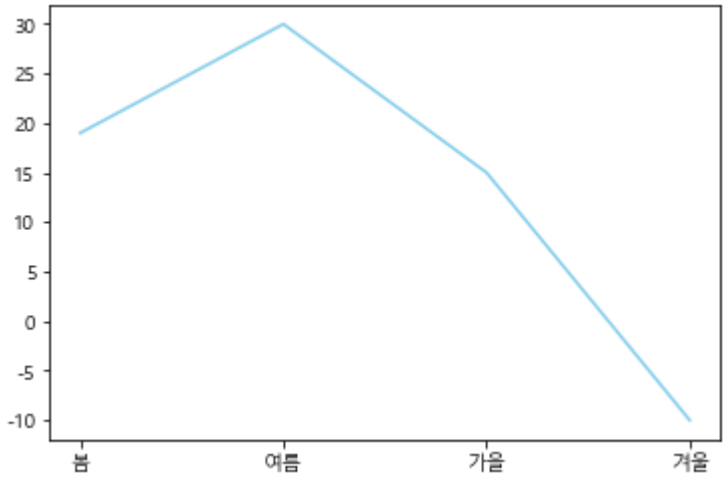
https://matplotlib.org/2.0.2/examples/color/named_colors.html

- 자주 사용되는 색깔은 약자를 사용할 수 있다.(blue:b, green:g, red:r, cyan:c, magenta:m, yellow:y, black:k, white:w)
- hex code를 사용할 수 있다.

```
x = ['봄','여름','가을','겨울']
y = [19,30,15,-10]
```

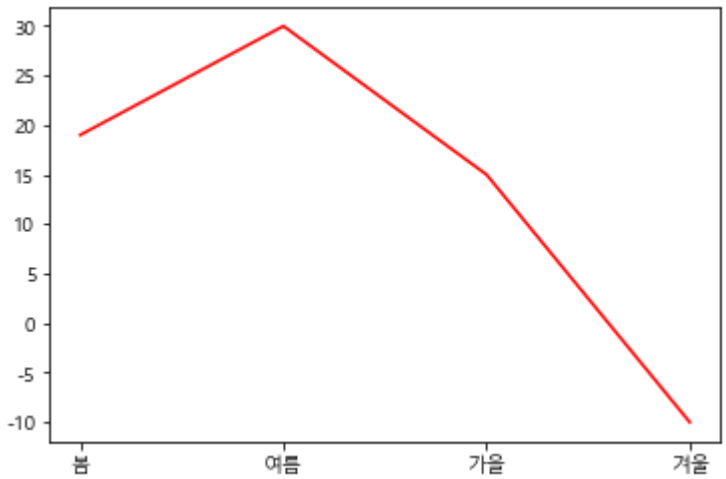
```
# 색상이름
plt.plot(x,y,color='skyblue')
```

```
[]
```



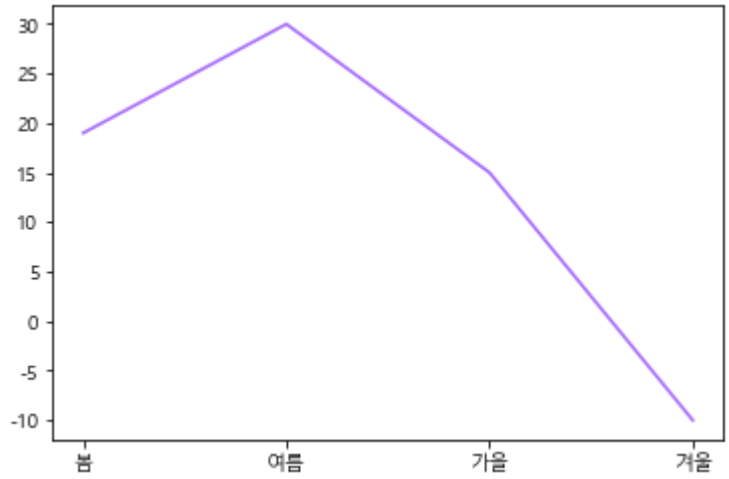

```
# 색상이름 약자(blue:b, green:g, red:r, cyan:c, magenta:m, yellow:y, black:k,
white:w)
plt.plot(x,y,color='r')
```

```
[]
```




```
# 색상코드
plt.plot(x,y,color='#A566FF')
```

```
[]
```

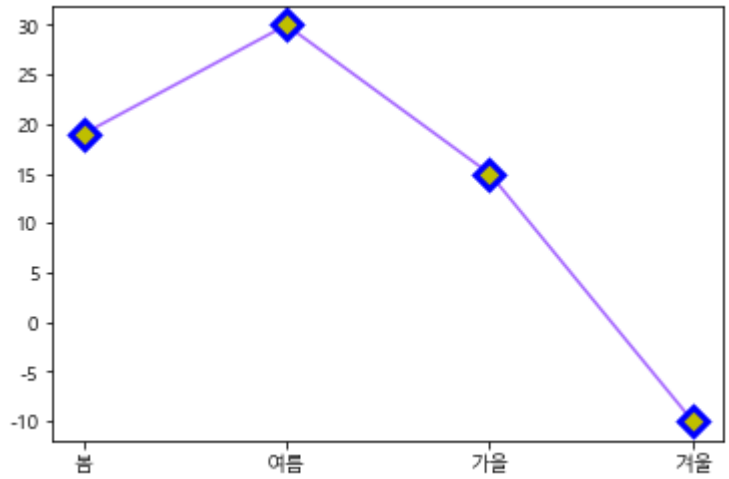


마커

- marker : 마커종류 (* . , o v ^ < > 1 2 3 4 s p * h H + x D d)
- markersize, ms : 마커사이즈
- markeredgecolor, mec : 마커 선 색깔
- markeredgewidth, mew : 마커 선 굵기
- markerfacecolor, mfc : 마커 내부 색깔

```
plt.plot(x,y,color='#A566FF', marker='D', ms='10', mec='b', mew='3', mfc='y')
```

```
[]
```



선

- linestyle, ls : 선스타일

'-' solid line style

'--' dashed line style

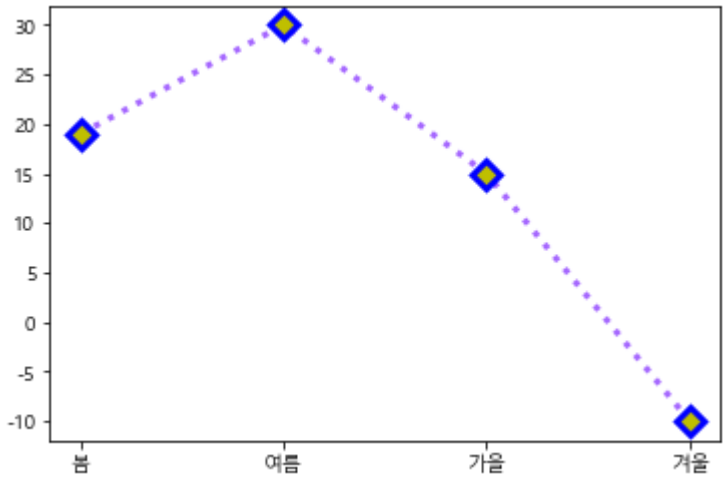
'-.' dash-dot line style

':' dotted line style

- linewidth, lw : 선 굵기

```
plt.plot(x,y,color='#A566FF', marker='D', ms='10', mec='b', mew='3', mfc='y'
        ,ls=':', lw=3)
```

```
[]
```



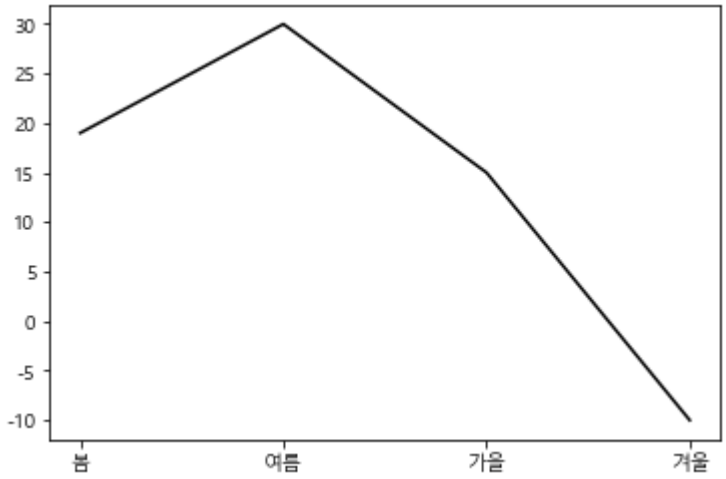
[색상][마커][선]

- [color][marker][line]

색상

```
plt.plot(x,y,'k')
```

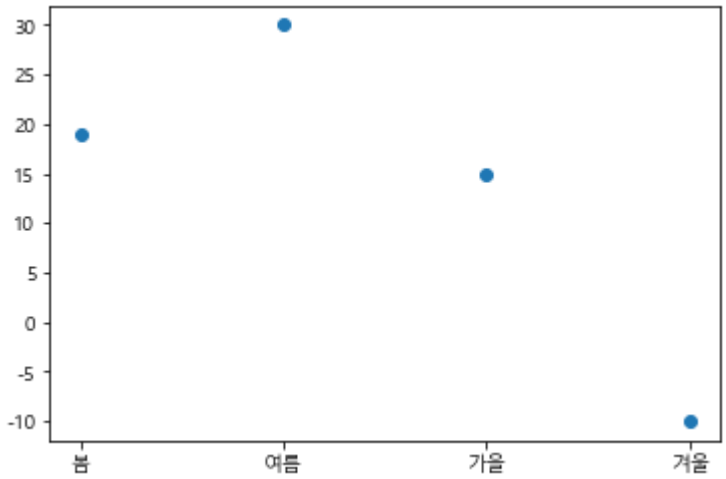
```
[]
```



마커

```
plt.plot(x,y,'o')
```

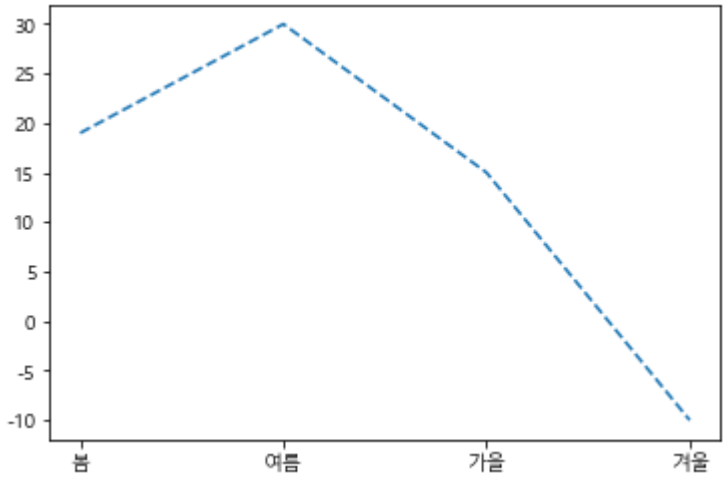
```
[]
```



선

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

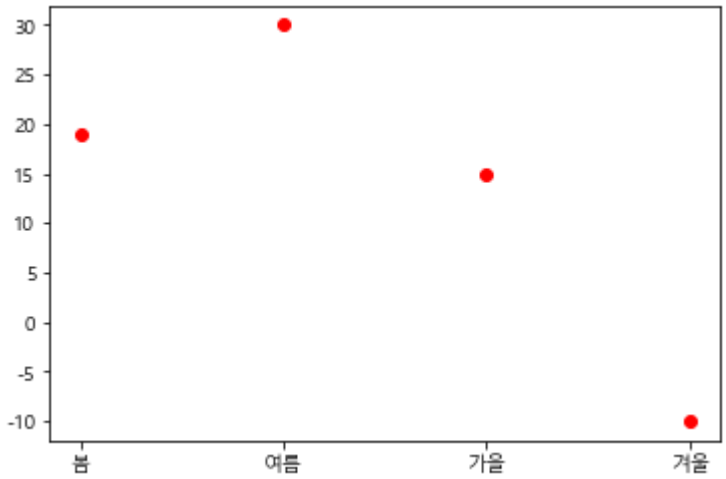
```
[]
```



색상,마커

```
plt.plot(x,y,'ro')
```

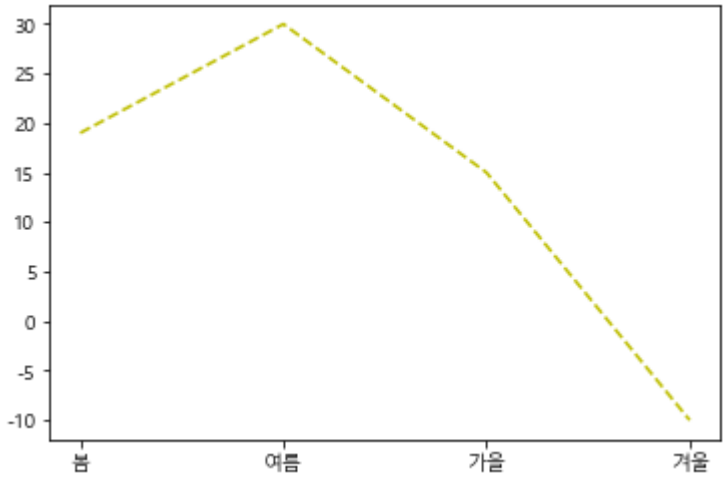
```
[]
```



색상,선

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

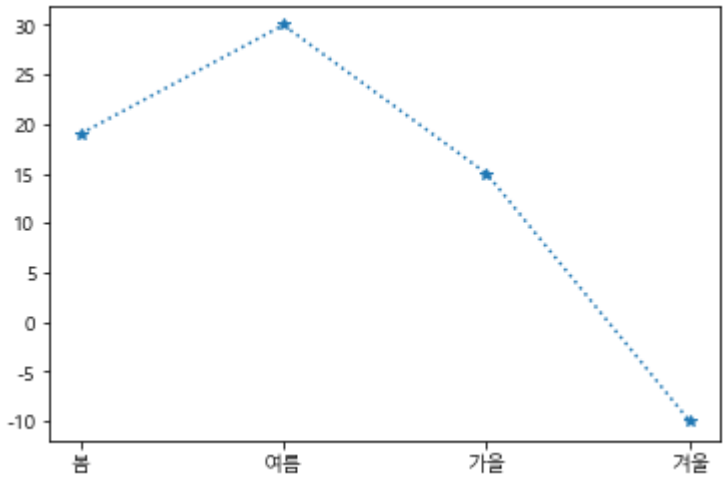
```
[]
```



마커, 선

```
plt.plot(x,y,'*:')
```

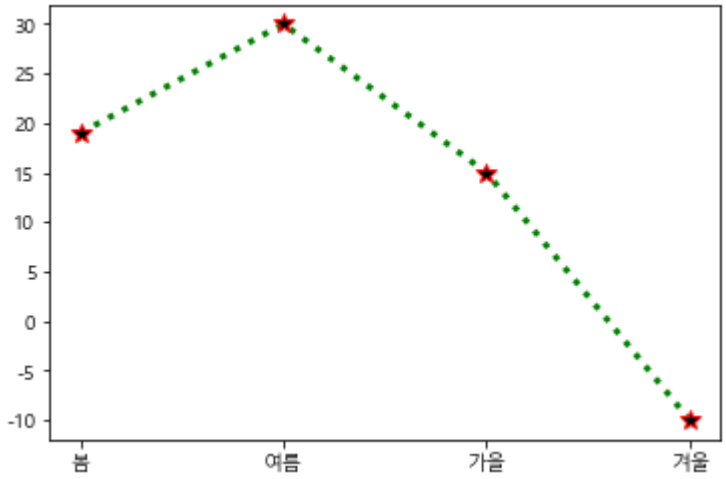
```
[]
```



색상,마커,선

```
plt.plot(x,y,'g*:', ms=10, mec='r', mfc='k', lw=3)
```

```
[]
```



라이브러리 임포트 및 그래프 설정

라이브러리 임포트

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

```
# 그래프에 한글폰트 설정
plt.rcParams['font.family'] = 'Malgun Gothic'

# 유니코드에서 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

```
# 폰트 확인하기
import matplotlib.font_manager as fm
[f.name for f in fm.fontManager.ttflist if 'Nanum' in f.name]
```

```
['NanumSquare',
 'NanumGothic',
 'Nanum HaNaSonGeurSsi',
 'NanumSquare_ac',
 'NanumSquare',
 'NanumGothic',
 'Nanum Pen Script',
 'Nanum Brush Script OTF',
 'NanumSquareRoundOTF',
 'NanumGothic',
 'CUNanum',
 'NanumSquare',
 'NanumSquareRound',
 'NanumBarunGothic',
```

```
    'NanumMyeongjo',
    'NanumSquareRound',
    'NanumSquare_ac',
    'NanumBarunpen',
    'NanumGothic',
    'NanumSquareRound',
    'NanumBarunGothic',
    'NanumBarunpenOTF',
    'Nanum Pen Script',
    'NanumBarunGothic',
    'NanumBarunpen',
    'NanumSquare_ac',
    'NanumMyeongjo',
    'NanumMyeongjo',
    'NanumSquareRound',
    'NanumSquareRound',
    'NanumSquare',
    'Nanum Brush Script',
    'NanumGothicCoding',
    'NanumBarunGothic',
    'NanumSquareRound',
    'NanumMyeongjo',
    'NanumMyeongjo',
    'NanumBarunpenOTF',
    'NanumSquare',
    'NanumMyeongjo',
    'Nanum Pen Script',
    'CUNanum',
    'Nanum Brush Script',
    'NanumBarunGothic',
    'NanumBarunGothic',
    'NanumSquare',
    'NanumBarunGothic',
    'NanumBarunGothic',
    'NanumGothic',
    'Nanum Brush Script',
    'NanumBarunpen',
    'NanumBarunGothic',
    'NanumSquareRoundOTF',
    'NanumSquare_ac',
    'NanumGothicCoding',
    'NanumBarunpen',
    'NanumGothic',
    'NanumSquare',
    'NanumGothic',
    'NanumGothic',
    'NanumBarunpen',
    'NanumSquare',
    'NanumSquareRoundOTF',
    'NanumGothic',
    'NanumSquareRound',
    'Nanum Pen Script OTF',
    'NanumGothic',
    'NanumGothic',
```

```
 'NanumBarunGothic',
 'NanumSquareRound',
 'NanumBarunGothic',
 'NanumBarunpen',
 'NanumGothic',
 'NanumSquareRoundOTF',
 'NanumBarunGothic']
```

샘플 데이터

- 월별 몸무게 변화

어떤 사람이 체중 관리를 위해 월별 몸무게를 기록하고 있음.

월별 몸무게 변화를 시각화하고자 함

- 1월:80kg, 2월:78kg, 3월:75kg, 4월:73kg, 5월:70kg

데이터프레임 만들기

```
import pandas as pd
```

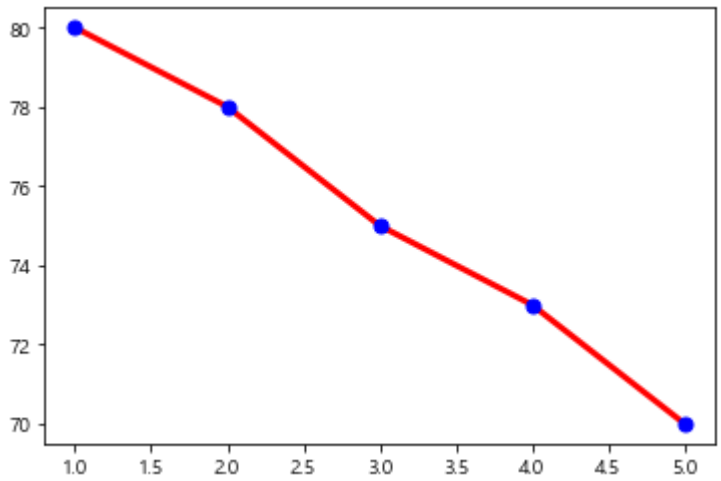
```
df = pd.DataFrame({'월':[1,2,3,4,5], '몸무게':[80,78,75,73,70]})
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	월	몸무게
0	1	80
1	2	78
2	3	75
3	4	73
4	5	70

그래프 그리기

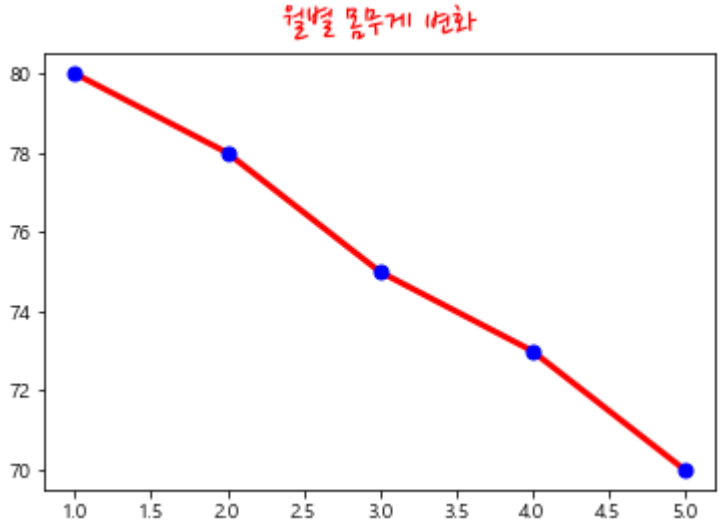
```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'ro-', mec='b', mfc='b', lw=3, ms=7)
plt.show()
```



제목

- plt.title('제목')
- loc : 제목 위치('left','center','right')
- pad : 타이틀과 그래프와의 간격
- color : 폰트색상
- fontsize : 폰트사이즈
- fontweight : 폰트굵기('normal','bold','heavy','light','ultrabold','ultralight')
- fontfamily : 폰트

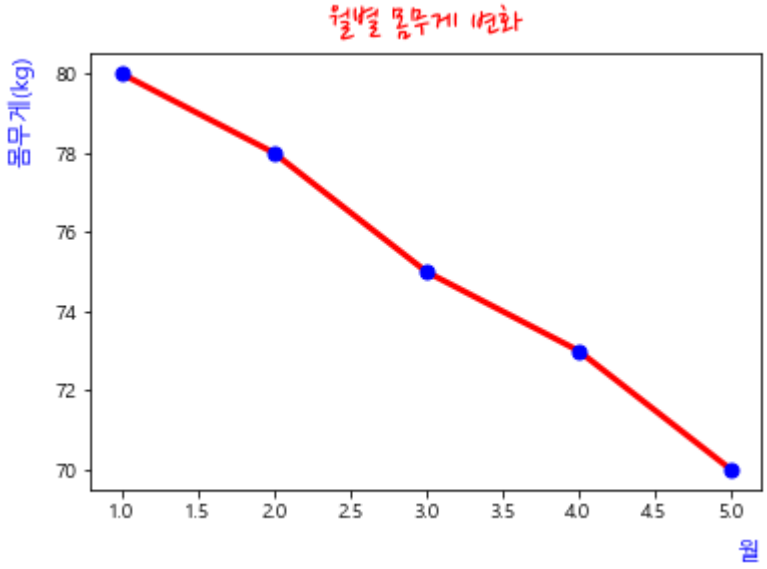
```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'ro-', mec='b', mfc='b', lw=3, ms=7)
plt.title('월별 몸무게 변화', loc='center', pad=10, color='r', fontsize=20,
fontweight='bold'
          , fontfamily='Nanum Pen Script')
plt.show()
```



x축, y축 레이블

- plt.xlabel('레이블')
- plt.ylabel('레이블')
- loc : 위치('left','center','right' / 'bottom', 'center','top')
- labelpad : 레이블과 그래프와의 간격
- color : 폰트색상
- fontsize : 폰트사이즈
- fontfamily : 폰트

```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'ro-', mec='b', mfc='b', lw=3, ms=7)
plt.title('월별 몸무게 변화', loc='center', pad=10, color='r', fontsize=20,
fontweight='bold'
          , fontfamily='Nanum Pen Script')

plt.xlabel('월',loc='right',labelpad=10, color='b',fontsize=12)
plt.ylabel('몸무게(kg)', loc='top',labelpad=10, color='b', fontsize=12)
plt.show()
```



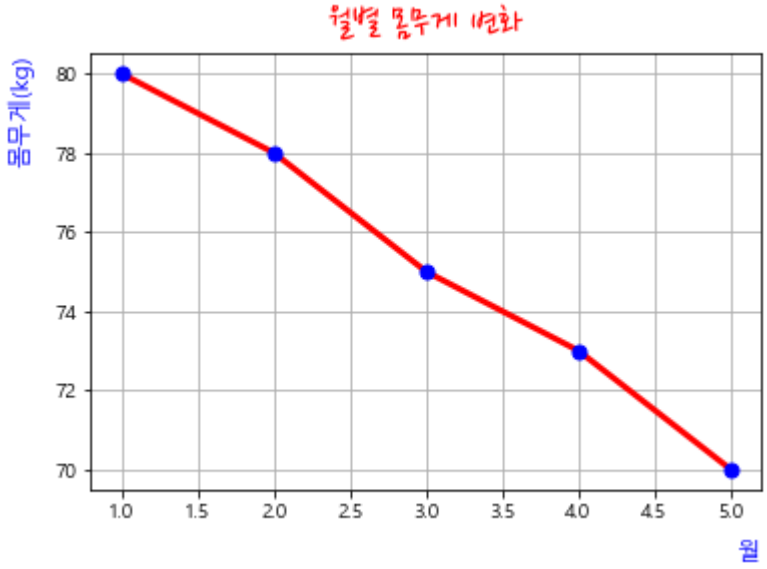
그리드

- plt.grid(True)
- axis : 그리드 방향
- ls, lw, color, alpha 등의 속성을 지정할 수 있음

기본 그리드

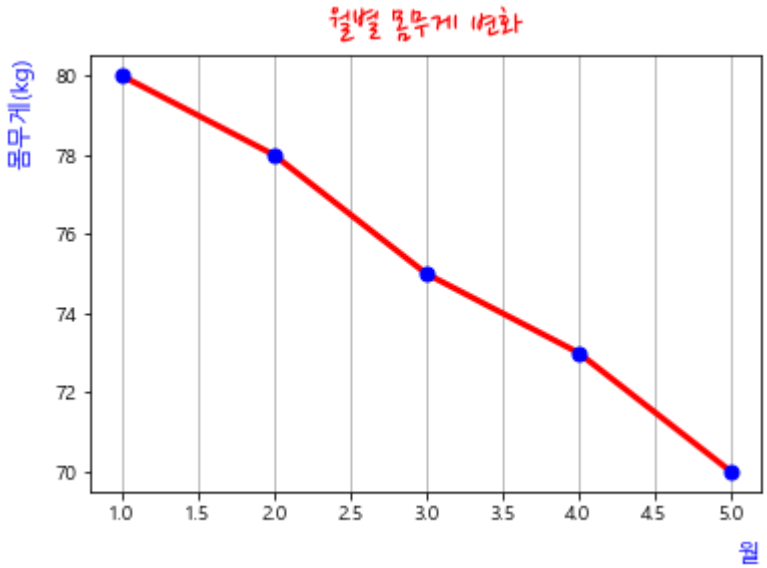
- plt.grid()

```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'ro-', mec='b', mfc='b', lw=3, ms=7)
plt.title('월별 몸무게 변화', loc='center', pad=10, color='r', fontsize=20,
fontweight='bold'
          , fontfamily='Nanum Pen Script')

plt.xlabel('월',loc='right',labelpad=10, color='b',fontsize=12)
plt.ylabel('몸무게(kg)', loc='top',labelpad=10, color='b', fontsize=12)

plt.grid()
plt.show()
```



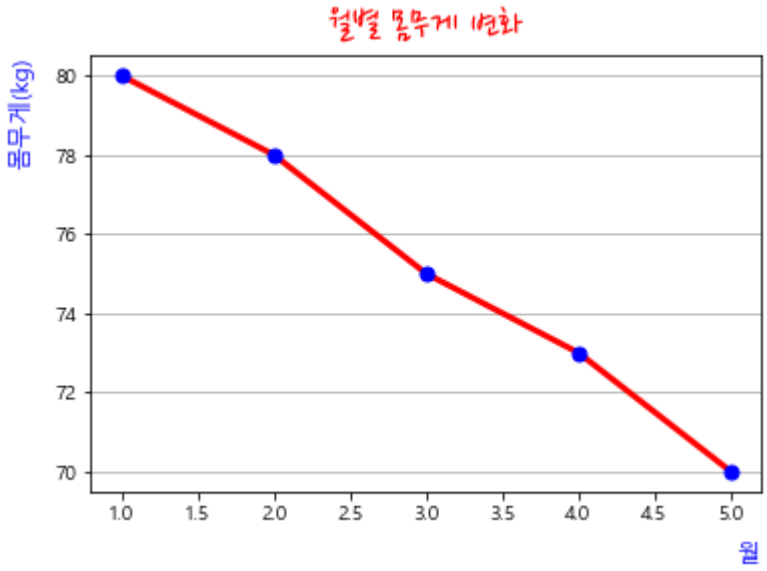
x축 그리드

```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'ro-', mec='b', mfc='b', lw=3, ms=7)
plt.title('월별 몸무게 변화', loc='center', pad=10, color='r', fontsize=20,
fontweight='bold'
          , fontfamily='Nanum Pen Script')

plt.xlabel('월',loc='right',labelpad=10, color='b',fontsize=12)
plt.ylabel('몸무게(kg)', loc='top',labelpad=10, color='b', fontsize=12)

plt.grid(axis='x')
plt.show()
```



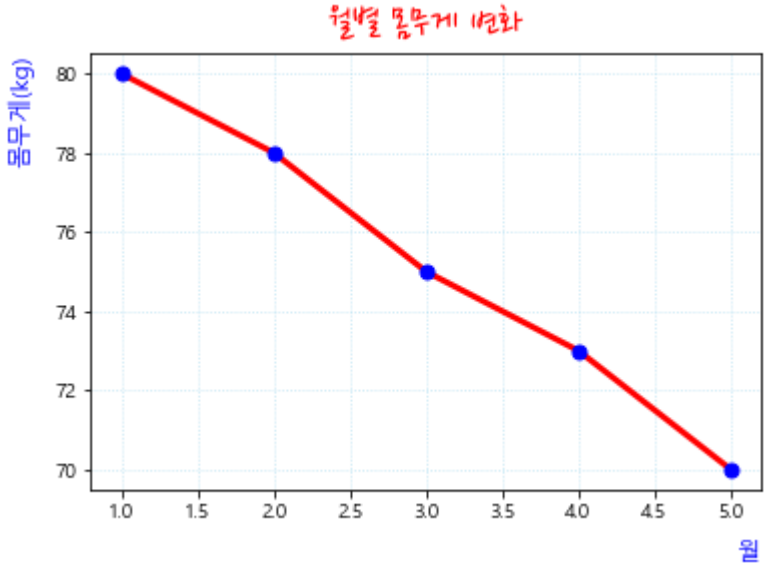
y축 그리드

```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'ro-', mec='b', mfc='b', lw=3, ms=7)
plt.title('월별 몸무게 변화', loc='center', pad=10, color='r', fontsize=20,
fontweight='bold'
          , fontfamily='Nanum Pen Script')

plt.xlabel('월',loc='right',labelpad=10, color='b',fontsize=12)
plt.ylabel('몸무게(kg)', loc='top',labelpad=10, color='b', fontsize=12)

plt.grid(axis='y')
plt.show()
```



그리드 스타일링

```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'ro-', mec='b', mfc='b', lw=3, ms=7)
plt.title('월별 몸무게 변화', loc='center', pad=10, color='r', fontsize=20,
fontweight='bold'
          , fontfamily='Nanum Pen Script')

plt.xlabel('월',loc='right',labelpad=10, color='b',fontsize=12)
plt.ylabel('몸무게(kg)', loc='top',labelpad=10, color='b', fontsize=12)

plt.grid(color='skyblue', alpha=0.5, ls=':')
plt.show()
```



[학습목표]

그래프의 가독성을 높이기 위해 축의 범위와 눈금을 설정할 수 있다.

라이브러리 임포트 및 그래프 설정

라이브러리 임포트

- matplotlib의 pyplot 모듈 사용
- 관용적으로 plt라는 별칭 사용

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

```
import matplotlib as mpl

# 그래프에 마이너스 기호 깨지는 문제 해결
mpl.rcParams['axes.unicode_minus'] = False

# 그래프에 한글 설정
mpl.rcParams['font.family'] = 'NanumGothic'
```

샘플 데이터

- 월별 몸무게 변화

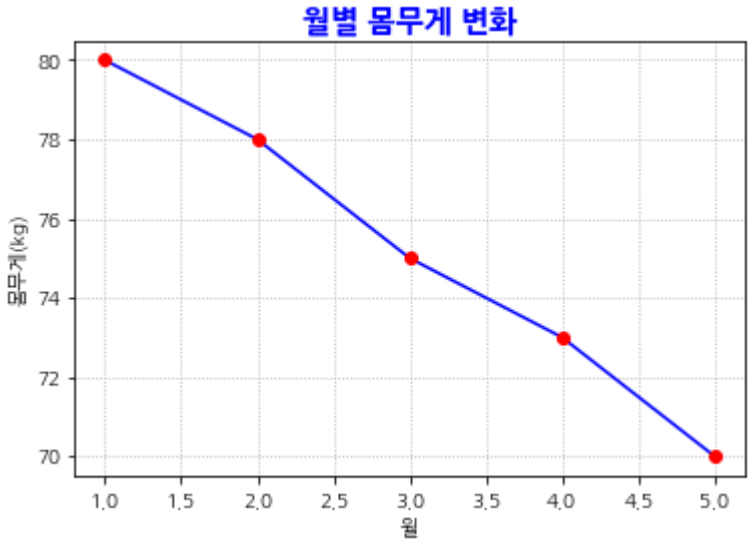
```
import pandas as pd
df = pd.DataFrame({'월':[1,2,3,4,5],'몸무게':[80,78,75,73,70]})
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	월	몸무게
0	1	80
1	2	78
2	3	75
3	4	73
4	5	70

그래프 그리기

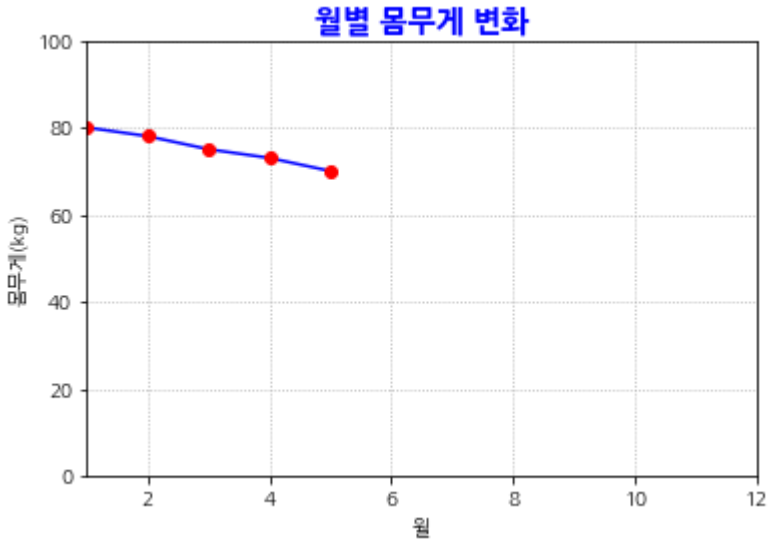
```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'bo-', mfc='r',mec='r')

# title
plt.title('월별 몸무게 변화', size=15, color='b', fontweight='bold')

# 축 레이블
plt.xlabel('월')
plt.ylabel('몸무게(kg)')

# 그리드
plt.grid(ls=':')
plt.show()
```



축의 범위 지정하기

- plt.xlim(min,max)
- plt.ylim(min,max)

```
x = df['월']
y = df['몸무게']
plt.plot(x,y,'bo-', mfc='r',mec='r')

# title
plt.title('월별 몸무게 변화', size=15, color='b', fontweight='bold')

# 축 레이블
plt.xlabel('월')
plt.ylabel('몸무게(kg)')

# 축의 범위
plt.xlim(1,12)
plt.ylim(0,100)

# 그리드
plt.grid(ls=':')
plt.show()
```



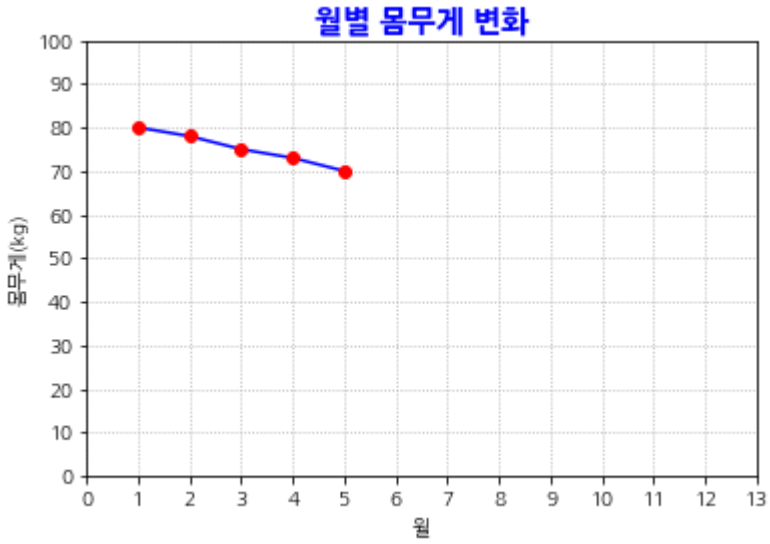
틱 설정

- x축, y축 눈금을 틱(tick)이라고 한다.
- 틱의 갯수, 레이블, 스타일을 변경할 수 있다.

틱 지정

- plt.xticks(틱리스트)
- plt.yticks(틱리스트)

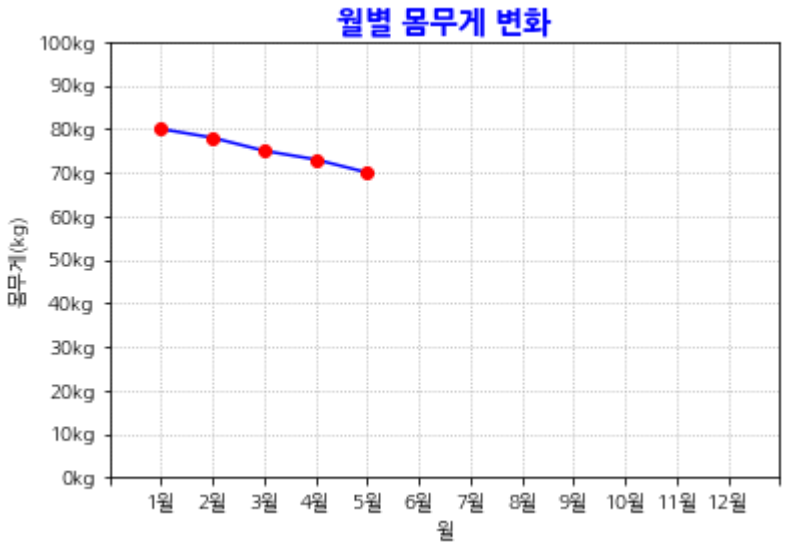
```
# x축 : 0,1,2,3,4,5,6,7,8,9,10,11,12,13
# y축 : 0,10,20,30,40,50,60,70,80,90,100

x = df['월']
y = df['몸무게']
plt.plot(x,y,'bo-', mfc='r',mec='r')

# title
plt.title('월별 몸무게 변화', size=15, color='b', fontweight='bold')

# 축 레이블
plt.xlabel('월')
plt.ylabel('몸무게(kg)')

# 틱
plt.xticks(range(0,14,1))
plt.yticks(range(0,110,10))

# 그리드
plt.grid(ls=':')
plt.show()
```



틱 레이블 지정

- plt.xticks(눈금리스트, label = 레이블리스트)
- plt.yticks(눈금리스트, label = 레이블리스트)
- 눈금의 개수와 동일한 개수의 레이블을 지정한다.

```
# x축 : 0,1,2,3,4,5,6,7,8,9,10,11,12,13
# y축 : 0,10,20,30,40,50,60,70,80,90,100

xticks_lable = ['','1월','2월','3월','4월','5월','6월','7월','8월','9월','10월','11
월','12월','']
yticks_lable =
['0kg','10kg','20kg','30kg','40kg','50kg','60kg','70kg','80kg','90kg','100kg']

# x축 : 0,1,2,3,4,5,6,7,8,9,10,11,12,13
# y축 : 0,10,20,30,40,50,60,70,80,90,100

x = df['월']
y = df['몸무게']
plt.plot(x,y,'bo-', mfc='r',mec='r')

# title
plt.title('월별 몸무게 변화', size=15, color='b', fontweight='bold')

# 축 레이블
plt.xlabel('월')
plt.ylabel('몸무게(kg)')

# 틱
plt.xticks(range(0,14,1), labels=xticks_lable)
plt.yticks(range(0,110,10), labels=yticks_lable)

# 그리드
plt.grid(ls=':')
plt.show()
```

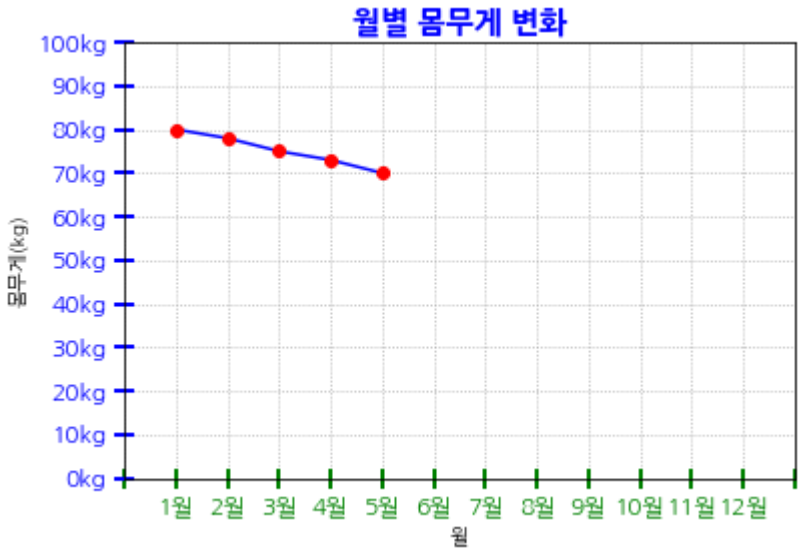


틱 스타일 지정

- plt.tick_params( )
- direction : 틱 위치 (in, out, inout)
- length : 틱의 길이
- width : 틱의 두께
- color : 틱 색상
- labelcolor : 틱 레이블 색상
- colors : 틱과 틱 레이블 색상
- pad : 틱과 레이블 사이의 거리
- labelsize : 틱 레이블 사이즈
- axis : 축 지정

```
# x축 : 0,1,2,3,4,5,6,7,8,9,10,11,12,13
# y축 : 0,10,20,30,40,50,60,70,80,90,100

xticks_lable = ['','1월','2월','3월','4월','5월','6월','7월','8월','9월','10월','11
월','12월','']
yticks_lable =
['0kg','10kg','20kg','30kg','40kg','50kg','60kg','70kg','80kg','90kg','100kg']

# x축 : 0,1,2,3,4,5,6,7,8,9,10,11,12,13
# y축 : 0,10,20,30,40,50,60,70,80,90,100

x = df['월']
y = df['몸무게']
plt.plot(x,y,'bo-', mfc='r',mec='r')
```

```
# title
plt.title('월별 몸무게 변화', size=15, color='b', fontweight='bold')

# 축 레이블
plt.xlabel('월')
plt.ylabel('몸무게(kg)')

# 틱
plt.xticks(range(0,14,1), labels=xticks_lable)
plt.yticks(range(0,110,10), labels=yticks_lable)

#plt.tick_params(direction='inout',length=10,width=2,color='g',labelcolor='m' )
plt.tick_params(axis='x',direction='inout',length=10,width=2,colors='g',labelsize=
12 )
plt.tick_params(axis='y',direction='inout',length=10,width=2,colors='b',labelsize=
12 )
# 그리드
plt.grid(ls=':')
plt.show()
```



[학습목표]

여러개의 그래프를 한번에 그리고 범례를 표시할 수 있다.

라이브러리 임포트 및 그래프 설정

라이브러리 임포트

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

그래프 설정

```
import matplotlib as mpl
# 그래프에 한글 설정
mpl.rcParams['font.family'] = 'NanumSquare'

# 그래프에 마이너스 기호 깨지는 문제 해결
mpl.rcParams['axes.unicode_minus'] = False
```

여러개의 그래프 한번에 그리기

```
import pandas as pd
df1 = pd.DataFrame({'월':[1,2,3,4,5],'몸무게':[80,78,75,73,70]})
df2 = pd.DataFrame({'월':[1,2,3,4,5],'몸무게':[60,62,59,55,54]})
```

```
df1
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	월	몸무게
0	1	80
1	2	78
2	3	75
3	4	73
4	5	70

```
df2
```

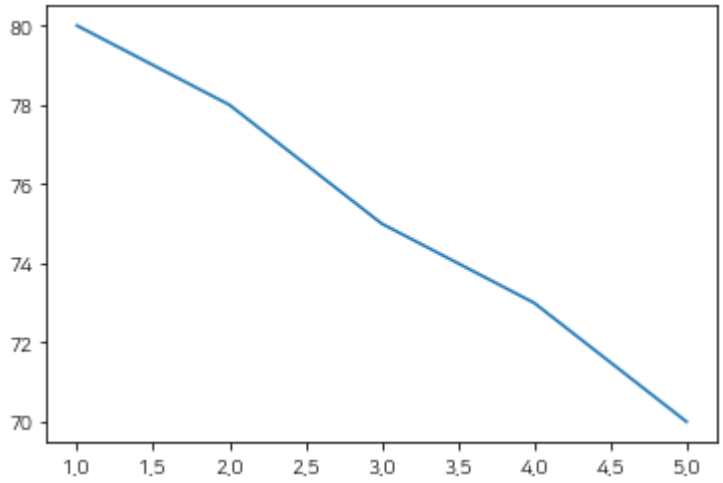
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

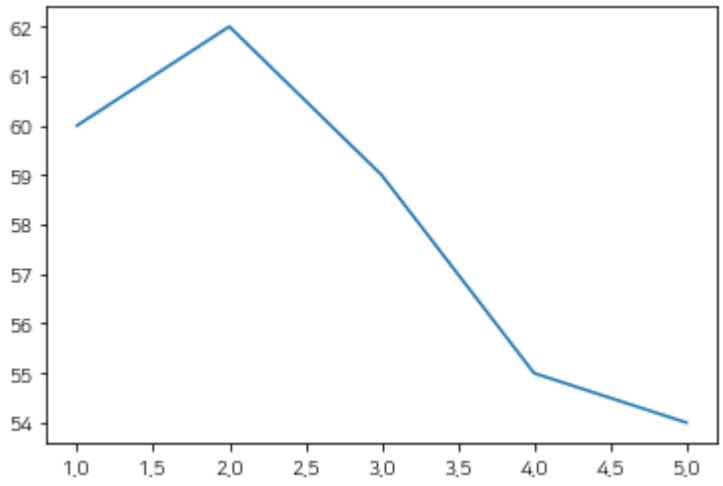
	월	몸무게
0	1	60
1	2	62
2	3	59
3	4	55
4	5	54

여러 개의 그래프 각각 그리기

- plt.show()로 그래프를 구분한다.

```
plt.plot(df1['월'],df1['몸무게'])
plt.show()

plt.plot(df2['월'],df2['몸무게'])
plt.show()
```





데이터가 다른 두 그래프 그리기

- plt.show()를 마지막에 한번만 사용한다.

```
#x축 틱 라벨
def fn_xtick(x):
    x = str(x)
    if x=='0' or x=='13':
        return ''
    else:
        return x+'월'

xtick_label = pd.Series(range(0,14,1))
xtick_label = xtick_label.apply(fn_xtick)
xtick_label
```

```
0
1      1월
2      2월
3      3월
4      4월
5      5월
6      6월
```

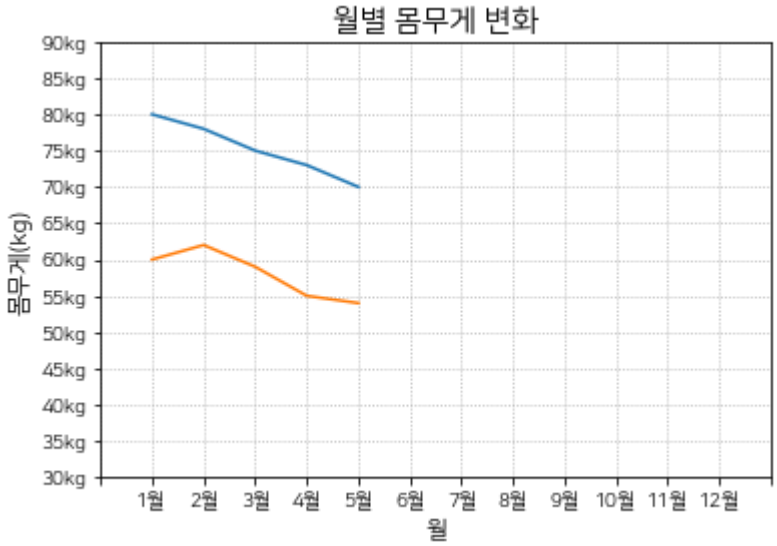
```
7      7월
8      8월
9      9월
10    10월
11    11월
12    12월
13
dtype: object
```

```
# y축 틱 라벨
def fn_ytick(x):
    x = str(x)
    return x+'kg'

ytick_label = pd.Series(range(30,95,5))
ytick_label = ytick_label.apply(fn_ytick)
ytick_label
```

```
0     30kg
1     35kg
2     40kg
3     45kg
4     50kg
5     55kg
6     60kg
7     65kg
8     70kg
9     75kg
10    80kg
11    85kg
12    90kg
dtype: object
```

```
# 두 그래프 그리기
plt.plot(df1['월'],df1['몸무게'])
plt.plot(df2['월'],df2['몸무게'])

# 그래프 제목
plt.title('월별 몸무게 변화', size=15)

# 축 라벨
plt.xlabel('월', fontsize=12)
plt.ylabel('몸무게(kg)', fontsize=12)

# x축 틱 : 1월, 2월, 3월,...,12월
plt.xticks(range(0,14,1), labels=xtick_label)
```

```
# y축 틱 : 30kg, 35kg, 40kg, ..., 90kg
plt.yticks(range(30,95,5), labels=ytick_label)

# 그리드
plt.grid(ls=':')

plt.show()
```



범례 표시하기

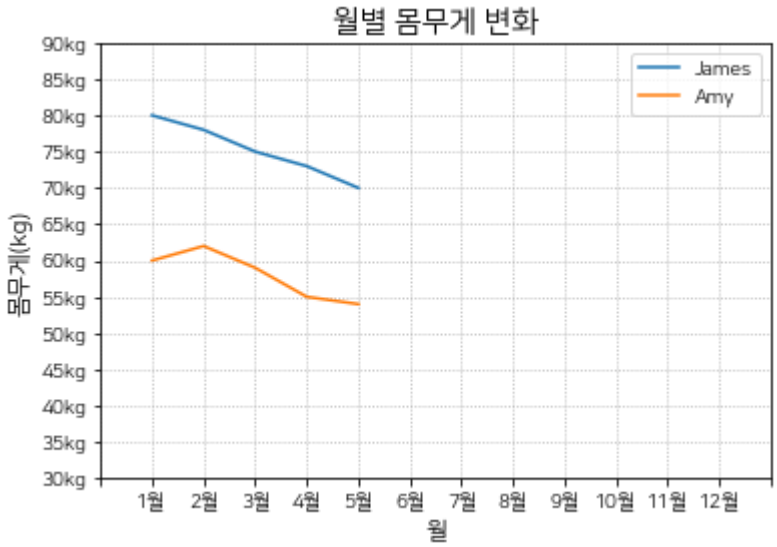
- 여러 개의 플롯을 동시에 그리는 경우 각 그래프가 무엇을 표시하는지 보여주기 위해 범례를 추가한다.

1. 그래프에 레이블을 지정한다.(label=그래프명)
2. 범례를 표시한다. --> plt.legend()

- 범례의 위치는 그래프에 따라 최적의 위치에 자동으로 표시된다.

```
# 두 그래프 그리기
plt.plot(df1['월'],df1['몸무게'], label='James')
plt.plot(df2['월'],df2['몸무게'], label='Amy')
plt.legend()

# 그래프 제목
plt.title('월별 몸무게 변화', size=15)

# 축 라벨
plt.xlabel('월', fontsize=12)
plt.ylabel('몸무게(kg)', fontsize=12)

# x축 틱 : 1월, 2월, 3월,...,12월
plt.xticks(range(0,14,1), labels=xtick_label)

# y축 틱 : 30kg, 35kg, 40kg, ..., 90kg
plt.yticks(range(30,95,5), labels=ytick_label)
```

```
# 그리드
plt.grid(ls=':')

plt.show()
```



범례 위치 지정

- plt.legend(loc=위치번호)
- plt.legend(loc=위치이름)

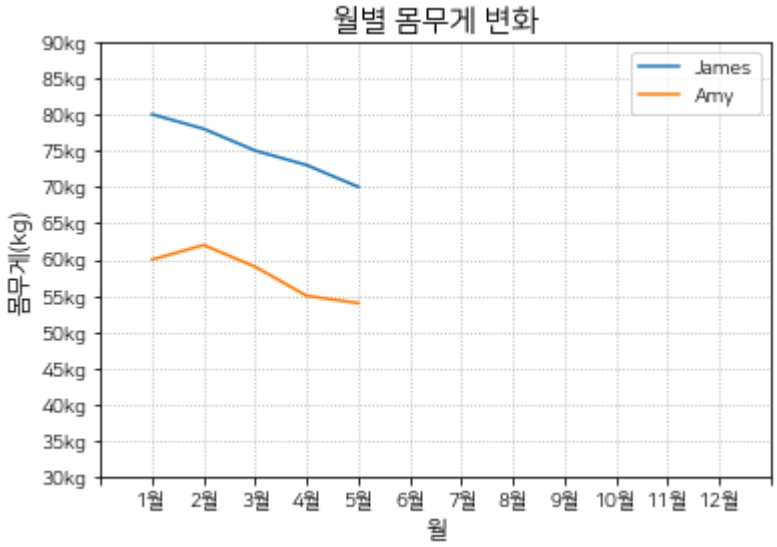
```
Location String   Location Code

===============   =============

'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

===============   =============
```

- plt.legend(loc=(x위치,y위치)) : 범례의 왼쪽 하단이 기준점.

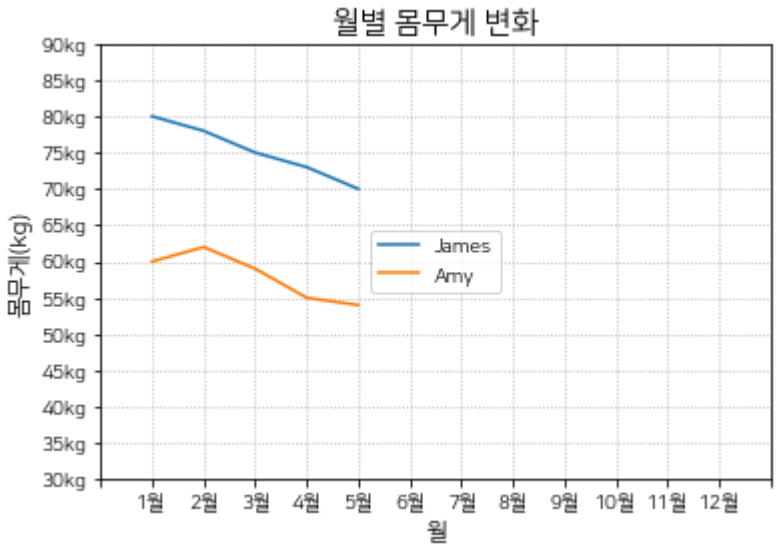
범례 위치 번호 사용

```
# 두 그래프 그리기
plt.plot(df1['월'],df1['몸무게'], label='James')
plt.plot(df2['월'],df2['몸무게'], label='Amy')
plt.legend(loc=1)

# 그래프 제목
plt.title('월별 몸무게 변화', size=15)

# 축 라벨
plt.xlabel('월', fontsize=12)
plt.ylabel('몸무게(kg)', fontsize=12)

# x축 틱 : 1월, 2월, 3월,...,12월
plt.xticks(range(0,14,1), labels=xtick_label)

# y축 틱 : 30kg, 35kg, 40kg, ..., 90kg
plt.yticks(range(30,95,5), labels=ytick_label)

# 그리드
plt.grid(ls=':')

plt.show()
```



범례 위치 이름 사용

```
# 두 그래프 그리기
plt.plot(df1['월'],df1['몸무게'], label='James')
plt.plot(df2['월'],df2['몸무게'], label='Amy')
plt.legend(loc='center')

# 그래프 제목
plt.title('월별 몸무게 변화', size=15)

# 축 라벨
plt.xlabel('월', fontsize=12)
plt.ylabel('몸무게(kg)', fontsize=12)

# x축 틱 : 1월, 2월, 3월,...,12월
plt.xticks(range(0,14,1), labels=xtick_label)

# y축 틱 : 30kg, 35kg, 40kg, ..., 90kg
plt.yticks(range(30,95,5), labels=ytick_label)

# 그리드
plt.grid(ls=':')

plt.show()
```



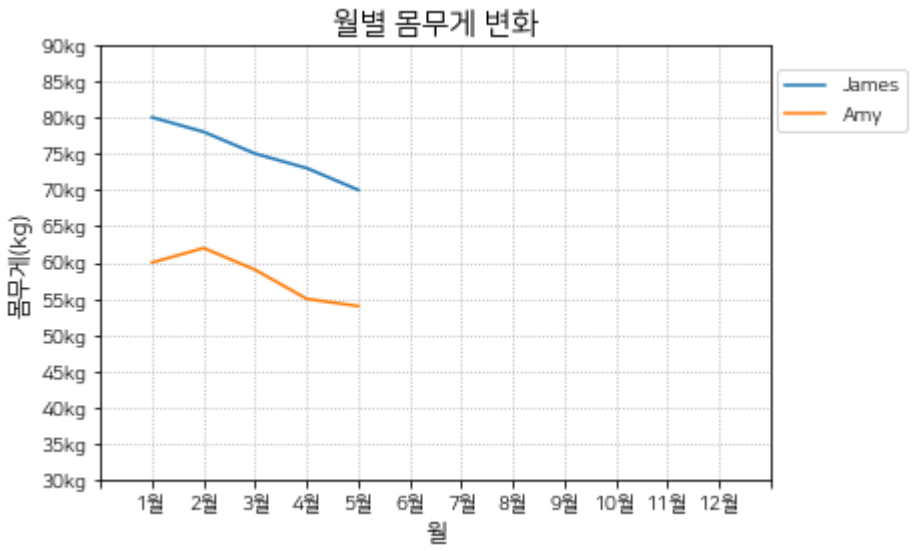
범례 x,y 위치 지정

- loc=(x,y)

왼쪽 아래 꼭지점 기준

```
# 두 그래프 그리기
plt.plot(df1['월'],df1['몸무게'], label='James')
plt.plot(df2['월'],df2['몸무게'], label='Amy')
plt.legend(loc=(1.01,0.8))

# 그래프 제목
```

```
plt.title('월별 몸무게 변화', size=15)

# 축 라벨
plt.xlabel('월', fontsize=12)
plt.ylabel('몸무게(kg)', fontsize=12)

# x축 틱 : 1월, 2월, 3월,...,12월
plt.xticks(range(0,14,1), labels=xtick_label)

# y축 틱 : 30kg, 35kg, 40kg, ..., 90kg
plt.yticks(range(30,95,5), labels=ytick_label)

# 그리드
plt.grid(ls=':')

plt.show()
```

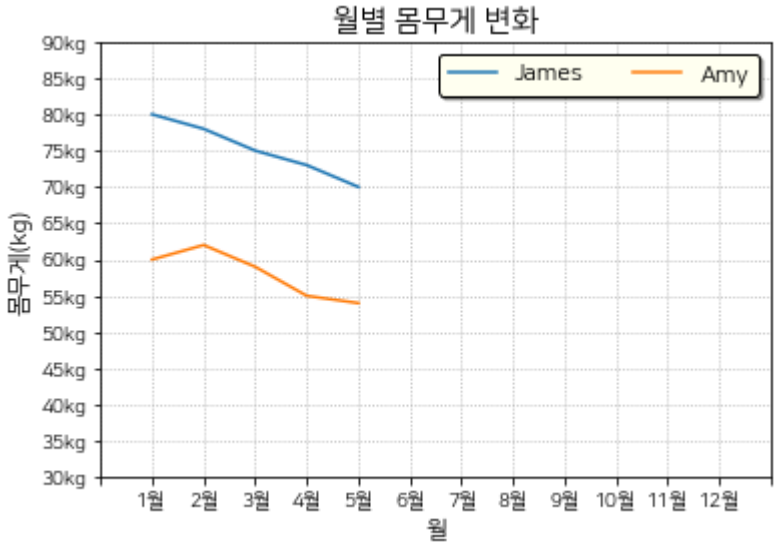


범례 속성 지정

- 열 개수 : plt.legend(ncol=열개수)
- 폰트 사이즈 : plt.legend(fontsize=폰트사이즈)
- 테두리 : plt.legend(frameon=True/False)
- 음영 : plt.legend(shadow=True/False)
- 바탕색 : plt.legend(facecolor=색상)
- 테두리색 : plt.legend(edgecolor=색상)

```
# 두 그래프 그리기
plt.plot(df1['월'],df1['몸무게'], label='James')
plt.plot(df2['월'],df2['몸무게'], label='Amy')
plt.legend(ncol=2,fontsize=12,shadow=True,facecolor='ivory',edgecolor='k')
```

```
# 그래프 제목
plt.title('월별 몸무게 변화', size=15)

# 축 라벨
plt.xlabel('월', fontsize=12)
plt.ylabel('몸무게(kg)', fontsize=12)

# x축 틱 : 1월, 2월, 3월,...,12월
plt.xticks(range(0,14,1), labels=xtick_label)

# y축 틱 : 30kg, 35kg, 40kg, ..., 90kg
plt.yticks(range(30,95,5), labels=ytick_label)

# 그리드
plt.grid(ls=':')

plt.show()
```



[학습목표]

pyplot 메소드로 하나의 실행창에 여러 그래프를 그려 비교할 수 있다.

- 앤스콤 4분할 그래프

영국의 프랭크 앤스콤(Frank Anscombe)이 데이터를 시각화하지 않고 수치만 확인할 때 발생할 수 있는 함정을 보여주기 위해 만든 그래프

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

데이터 불러오기

- seaborn 라이브러리에서 제공하는 anscombe 데이터 사용

```
import seaborn as sns
anscombe = sns.load_dataset('anscombe')
anscombe
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33
5	I	14.0	9.96
6	I	6.0	7.24

- 4가지 데이터를 각각의 데이터프레임으로 만들기

```
df1 = anscombe[anscombe['dataset']=='I']
df2 = anscombe[anscombe['dataset']=='II']
df3 = anscombe[anscombe['dataset']=='III']
df4 = anscombe[anscombe['dataset']=='IV']
```

- 데이터 확인하기

```
df4
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	dataset	x	y
33	IV	8.0	6.58
34	IV	8.0	5.76
35	IV	8.0	7.71
36	IV	8.0	8.84
37	IV	8.0	8.47
38	IV	8.0	7.04
39	IV	8.0	5.25

```
df4.shape
```

```
(11, 3)
```

데이터의 통계수치 확인

- 데이터프레임.describe()
- 4개의 데이터는 갯수, 평균, 표준편차가 모두 같다.
- 이러한 수치만 보고 4개의 데이터 그룹의 데이터가 모두 같을 것이라고 착각할 수 있다.

```
df1.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	x	y
count	11.000000	11.000000
mean	9.000000	7.500909
std	3.316625	2.031568

	x	y
min	4.000000	4.260000
25%	6.500000	6.315000
50%	9.000000	7.580000

```
df2.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	x	y
count	11.000000	11.000000
mean	9.000000	7.500909
std	3.316625	2.031657
min	4.000000	3.100000
25%	6.500000	6.695000
50%	9.000000	8.140000
75%	11.500000	8.950000

```
df3.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	x	y
count	11.000000	11.000000

	x	y
mean	9.000000	7.500000
std	3.316625	2.030424
min	4.000000	5.390000
25%	6.500000	6.250000

```
df4.describe()
```

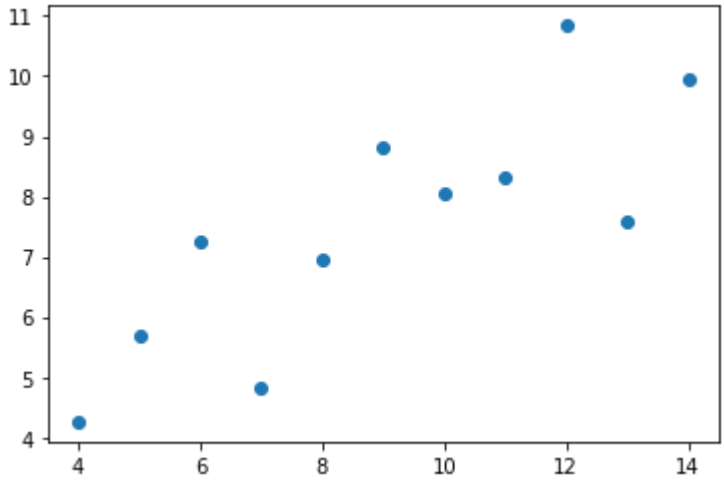
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	x	y
count	11.000000	11.000000
mean	9.000000	7.500909
std	3.316625	2.030579
min	8.000000	5.250000
25%	8.000000	6.170000
50%	8.000000	7.040000
75%	8.000000	8.190000

데이터 시각화

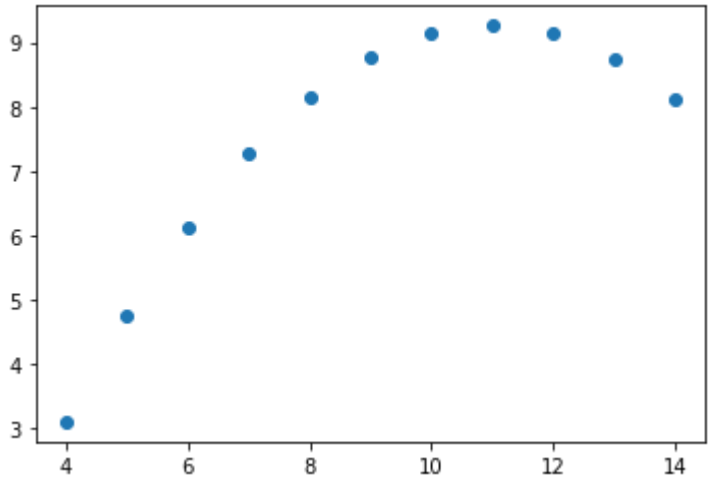
```
# 첫번째 데이터
plt.plot(df1['x'],df1['y'],'o')
```

```
[]
```



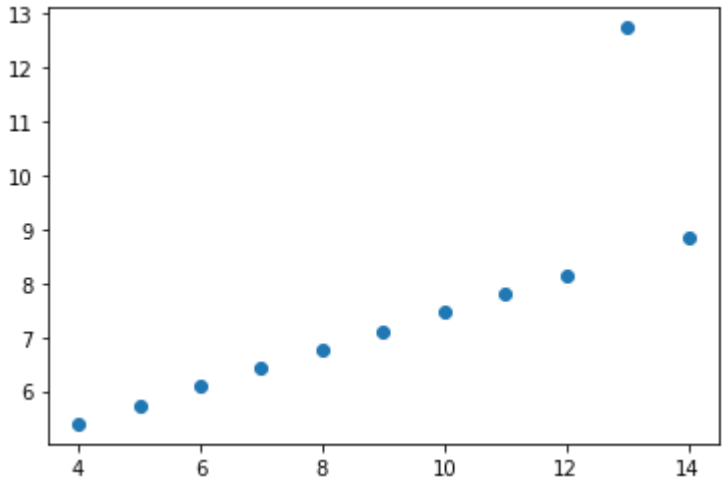

```
# 두번째 데이터
plt.plot(df2['x'],df2['y'],'o')
```

```
[]
```



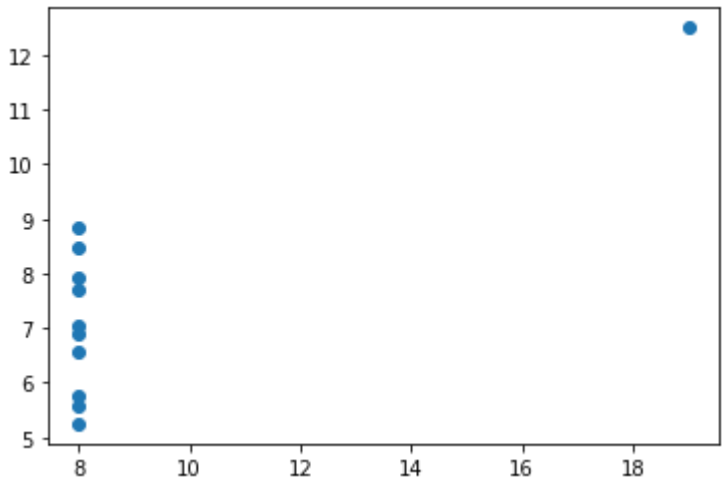

```
# 세번째 데이터
plt.plot(df3['x'],df3['y'],'o')
```

```
[]
```



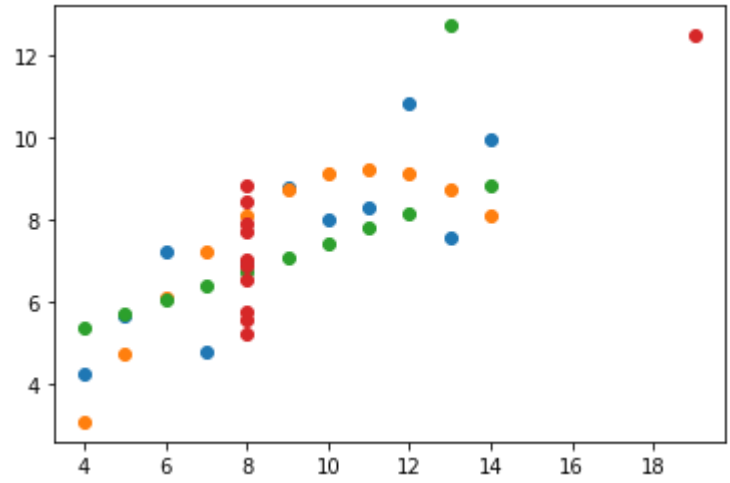

```
# 네번째 데이터
plt.plot(df4['x'],df4['y'],'o')
```

```
[]
```




```
# 4개의 그래프 한번에 그리기
plt.plot(df1['x'],df1['y'],'o')
plt.plot(df2['x'],df2['y'],'o')
plt.plot(df3['x'],df3['y'],'o')
plt.plot(df4['x'],df4['y'],'o')
```

```
[]
```



서브플롯 그리기

1. 전체 그래프의 크기를 정한다. (정하지 않으면 디폴트 크기로 지정된다.)

```
plt.figure(figsize=(x사이즈, y사이즈))
```

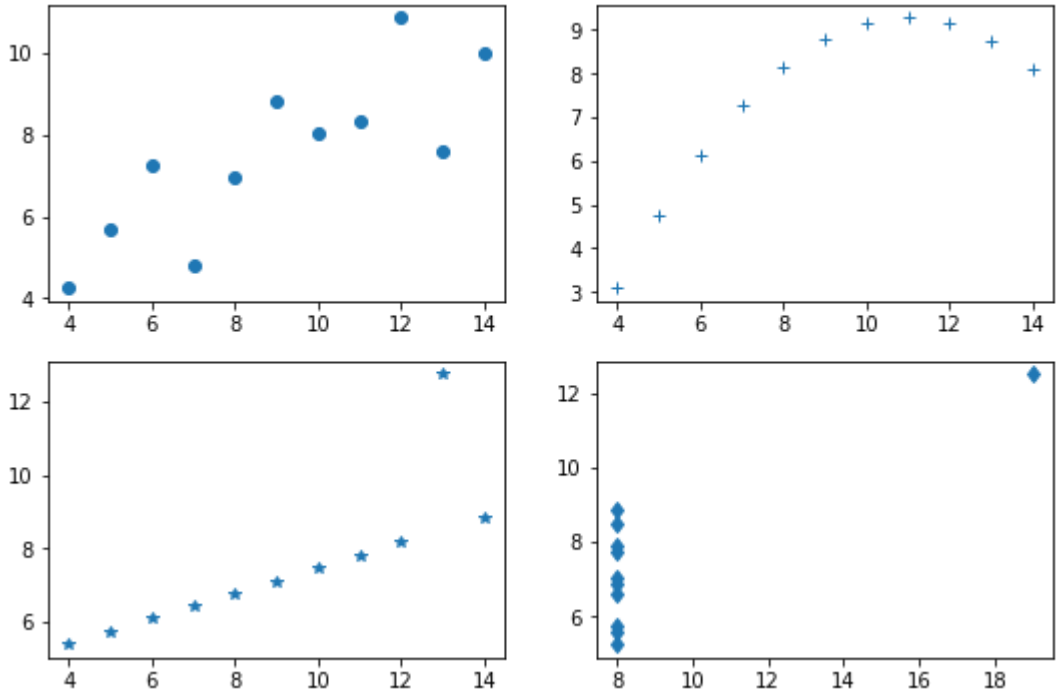
2. 그래프를 그려 넣을 격자를 지정한다.(전체행개수,전체열개수,그래프순서)

```
plt.subplot(전체행개수,전체열개수,그래프순서
```

3. 격자에 그래프를 하나씩 추가한다.

```
plt.figure(figsize=(9,6))
plt.subplot(221)
plt.plot(df1['x'],df1['y'],'o')

plt.subplot(222)
plt.plot(df2['x'],df2['y'],'+')

plt.subplot(223)
plt.plot(df3['x'],df3['y'],'*')

plt.subplot(224)
plt.plot(df4['x'],df4['y'],'d')
```

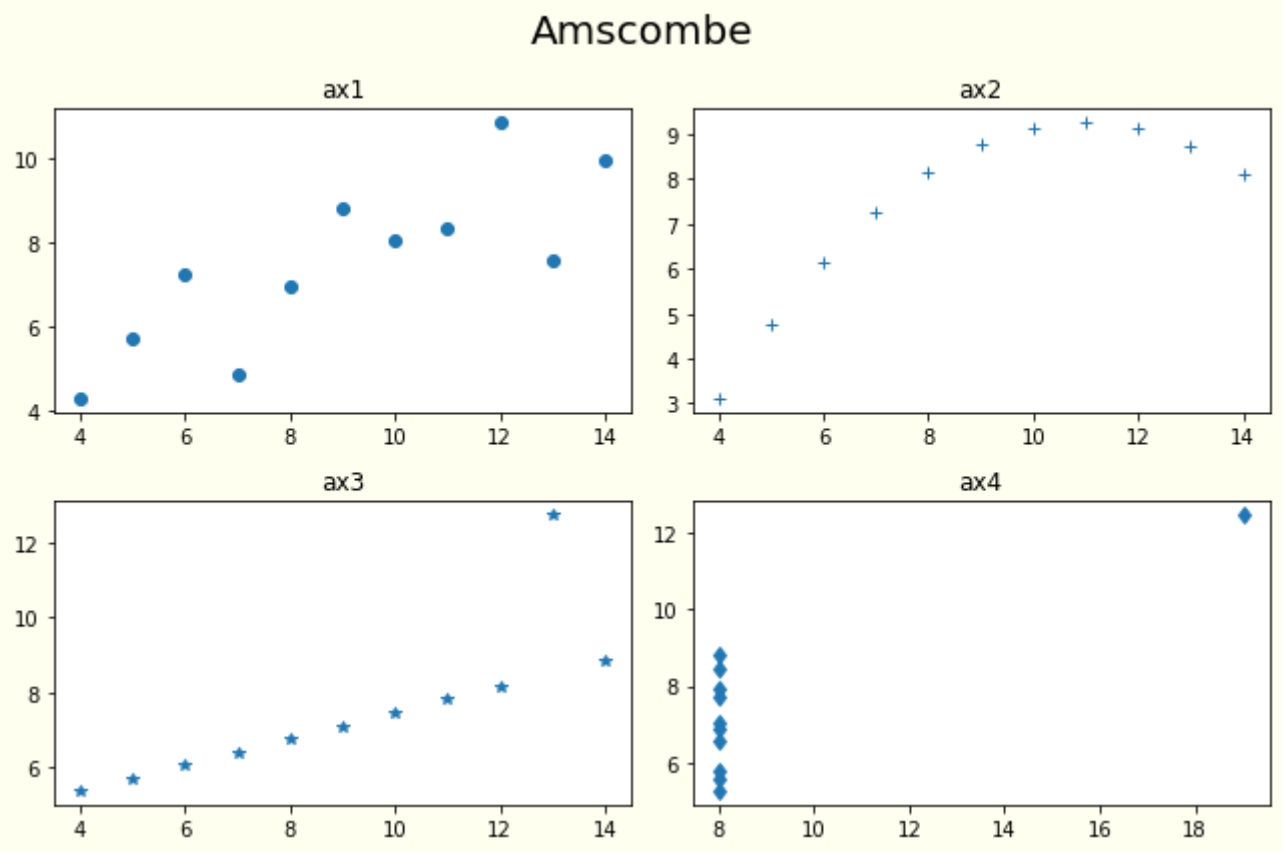
```
[]
```



전체 그래프의 속성 지정

- figure 객체를 변수에 받는다.
- figure객체의 suptitle(제목)메소드로 전체 그래프의 제목을 표시한다.
- figure객체의 tight_layout()메소드로 그래프의 간격, 너비를 최적화한다.

```
fig = plt.figure(figsize=(9,6), facecolor='ivory')

plt.subplot(221)
plt.plot(df1['x'],df1['y'],'o')
plt.title('ax1')

plt.subplot(222)
plt.plot(df2['x'],df2['y'],'+')
plt.title('ax2')

plt.subplot(223)
plt.plot(df3['x'],df3['y'],'*')
plt.title('ax3')

plt.subplot(224)
plt.plot(df4['x'],df4['y'],'d')
plt.title('ax4')

fig.suptitle('Amscombe', size=20)
fig.tight_layout()
```



[학습목표]

그래프의 위치, 크기를 지정하여 그래프를 그릴 수 있다.

라이브러리 임포트

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

위치, 크기 지정하여 그래프 그리기

figure, axes

- figure : 그림이 그려지는 캔버스
- axes : 하나의 그래프

위치, 크기 지정하여 그래프 그리기

1. figure 객체를 생성한다.

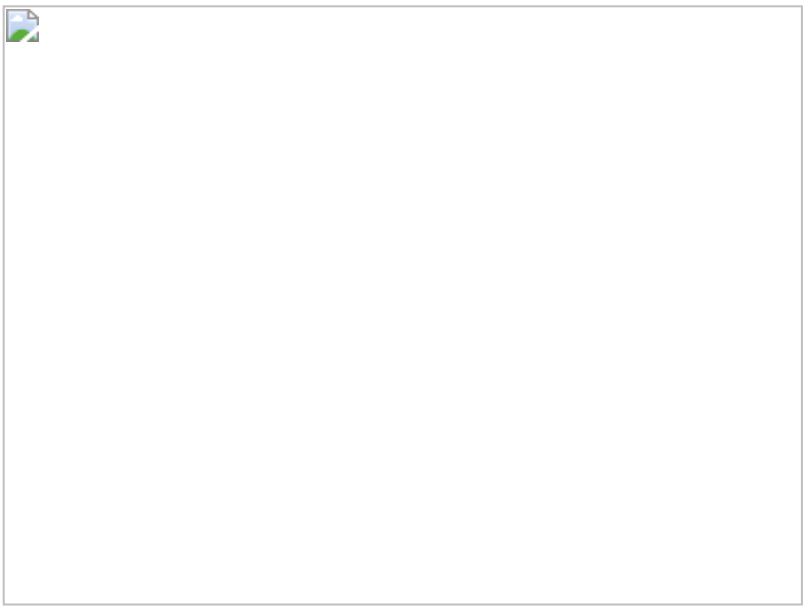
```
fig = plt.figure(figsize=(가로길이,세로길이))
```

2. figure객체의 add_axes 메소드로 위치와 크기를 지정하여 axes 객체를 생성한다.

```
ax1 = fig.add_axes([left, bottom, width, height])
```

left, bottom : 상대적인 시작 위치 (figsize의 크기를 1이라고 했을 때 상대적 위치)

width, height : 상대적인 크기(figsize의 크기를 1이라고 했을 때 상대적 크기)



3. axes에 그래프를 그린다.

```
ax1.plot(x,y)
```

4. axes에 제목 추가.

```
ax1.set_title(제목)
```

위치와 크기를 자유롭게 지정하여 axes 객체 만들기

- add_axes를 사용하면, 서브플롯의 크기와 위치를 자유롭게 지정할 수 있다.
- 그래프를 겹쳐그리거나, 크기가 각각 다른 그래프를 그릴 수 있다.

앤스콤 4분할 그래프 그리기

```
import seaborn as sns
anscombe = sns.load_dataset('anscombe')
df1 = anscombe[anscombe['dataset']=='I']
df2 = anscombe[anscombe['dataset']=='II']
df3 = anscombe[anscombe['dataset']=='III']
df4 = anscombe[anscombe['dataset']=='IV']
```

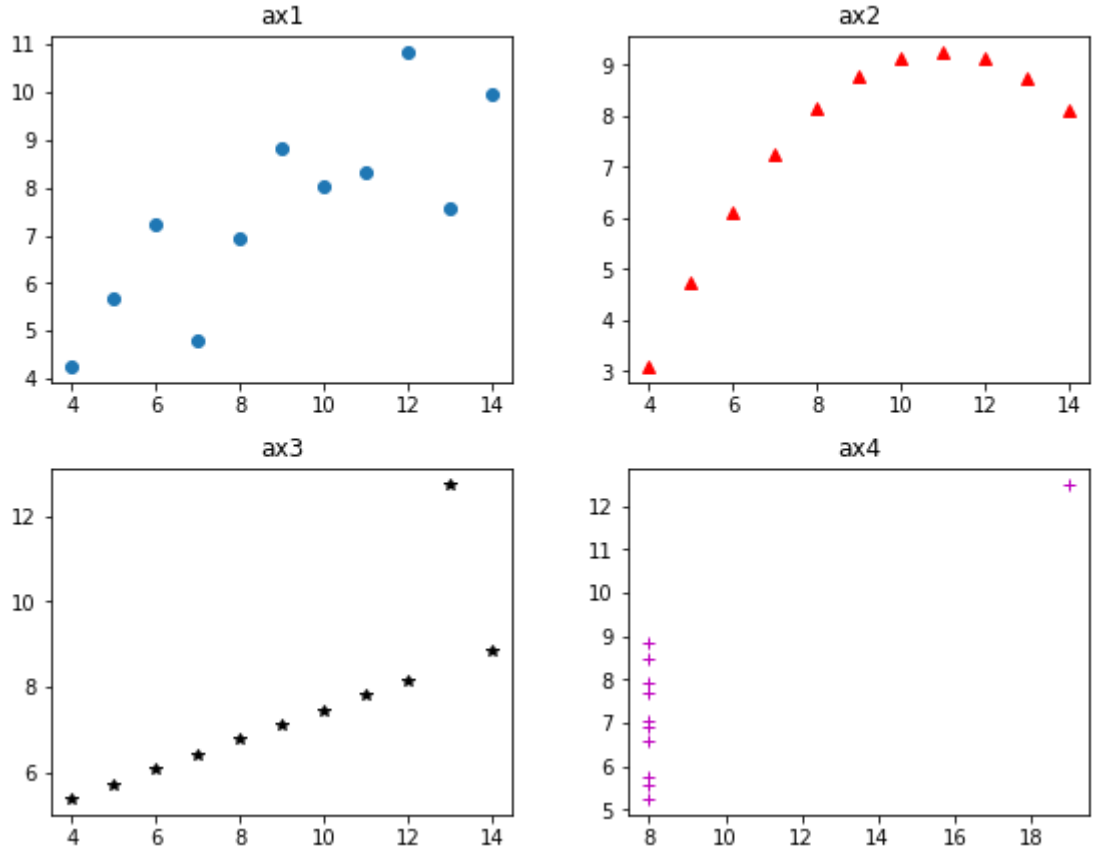
```
df4
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	dataset	x	y
33	IV	8.0	6.58
34	IV	8.0	5.76
35	IV	8.0	7.71
36	IV	8.0	8.84
37	IV	8.0	8.47
38	IV	8.0	7.04
39	IV	8.0	5.25

```
# 1)figure 객체를 생성한다.
fig = plt.figure(figsize=(8,6))

# 2) figure객체의 add_axes 메소드로 위치와 크기를 지정하여 axes 객체를 생성한다.
ax1 = fig.add_axes([0,0.5,0.4,0.4])
ax2 = fig.add_axes([0.5,0.5,0.4,0.4])
ax3 = fig.add_axes([0,0,0.4,0.4])
ax4 = fig.add_axes([0.5,0,0.4,0.4])

# 3) axes에 그래프를 그린다.
ax1.plot(df1['x'],df1['y'],'o')
ax2.plot(df2['x'],df2['y'],'r^')
ax3.plot(df3['x'],df3['y'],'k*')
ax4.plot(df4['x'],df4['y'],'m+')

# 4) axes에 제목 추가.
ax1.set_title('ax1')
ax2.set_title('ax2')
ax3.set_title('ax3')
ax4.set_title('ax4')
```

```
Text(0.5, 1.0, 'ax4')
```



[학습목표]

axes를 행,열로 쪼개어 서브플롯을 그릴 수 있다.

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

데이터 불러오기

```
import seaborn as sns
anscombe = sns.load_dataset('anscombe')
df1 = anscombe[anscombe['dataset']=='I']
df2 = anscombe[anscombe['dataset']=='II']
df3 = anscombe[anscombe['dataset']=='III']
df4 = anscombe[anscombe['dataset']=='IV']
```

```
df4
```

```
.dataframe tbody tr th {
    vertical-align: top;
```

```
}

.dataframe thead th {
    text-align: right;
}
```

	dataset	x	y
33	IV	8.0	6.58
34	IV	8.0	5.76
35	IV	8.0	7.71
36	IV	8.0	8.84
37	IV	8.0	8.47
38	IV	8.0	7.04
39	IV	8.0	5.25

axes를 행, 열로 쪼개어 서브플롯 그리기

- plt.subplots() 함수를 호출하면 figure, axes 객체를 생성하여 튜플 형태로 반환한다.

```
fig, ax = plt.subplots()
```

1. axes 객체를 행,열로 쪼개어 생성하기

```
fig, ax = plt.subplots(nrows=행개수, ncols=열개수,figsize=(가로사이즈,세로사이즈))
```

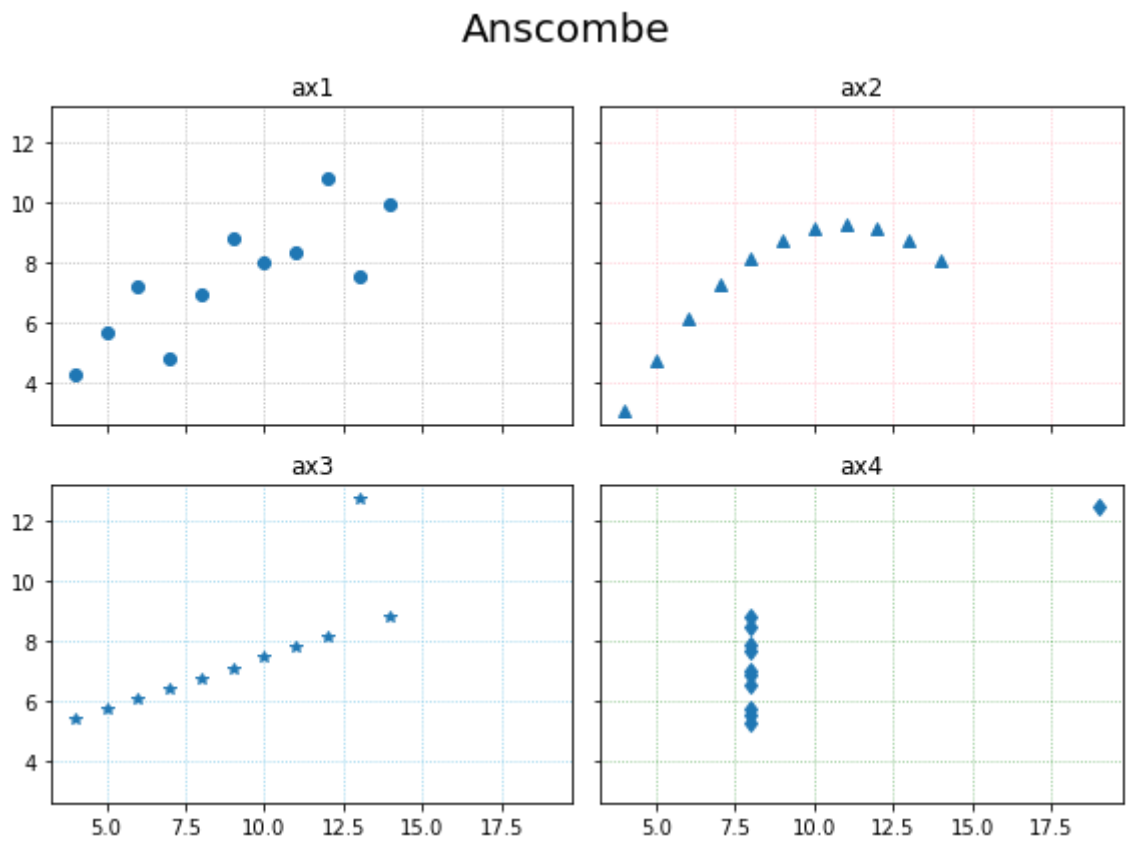
2. axes[행번호][열번호] 형태로 접근하여 그래프 그리기

3. 서브플롯간 축을 공유할 수 있다.

```
sharex=True, sharey=True
```

```
# 1) axes 객체를 행,열로 쪼개어 생성하기
# 3) 서브플롯간 축을 공유할 수 있다.
fig,ax = plt.subplots(nrows=2, ncols=2, figsize=(8,6), sharex=True, sharey=True)
```

```
# 2) axes[행번호][열번호] 형태로 접근하여 그래프 그리기
ax[0][0].plot(df1['x'],df1['y'],'o')
ax[0][1].plot(df2['x'],df2['y'],'^')
ax[1][0].plot(df3['x'],df3['y'],'*')
ax[1][1].plot(df4['x'],df4['y'],'d')

# 4) 각 그래프에 제목 추가
ax[0][0].set_title('ax1')
ax[0][1].set_title('ax2')
ax[1][0].set_title('ax3')
ax[1][1].set_title('ax4')

# 4) 각 그래프에 그리드 추가
ax[0][0].grid(ls=':')
ax[0][1].grid(ls=':', color='pink')
ax[1][0].grid(ls=':', color='skyblue')
ax[1][1].grid(ls=':', color='green', alpha=0.5)

# 6) 그래프 전체 제목
fig.suptitle('Anscombe', size=20)

# 7) 그래프 간격, 크기 최적화
fig.tight_layout()
```



[학습목표]

전체 행 열과 그래프 순서를 지정하여 서브플롯을 그릴 수 있다.

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

데이터 불러오기

```
import seaborn as sns
anscombe = sns.load_dataset('anscombe')
df1 = anscombe[anscombe['dataset']=='I']
df2 = anscombe[anscombe['dataset']=='II']
df3 = anscombe[anscombe['dataset']=='III']
df4 = anscombe[anscombe['dataset']=='IV']
```

```
df4
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	dataset	x	y
33	IV	8.0	6.58
34	IV	8.0	5.76
35	IV	8.0	7.71
36	IV	8.0	8.84
37	IV	8.0	8.47
38	IV	8.0	7.04
39	IV	8.0	5.25

전체 행 열과 그래프 순서에 따라 서브플롯 그리기

1. figure 객체를 생성한다.

```
fig=plt.figure()
```

2. 서브플롯을 그릴 axes 객체를 생성한다.

```
ax = fig.add_subplot(전체행개수,전체열개수,순서)
```

3. axes 객체에 그래프를 그린다.

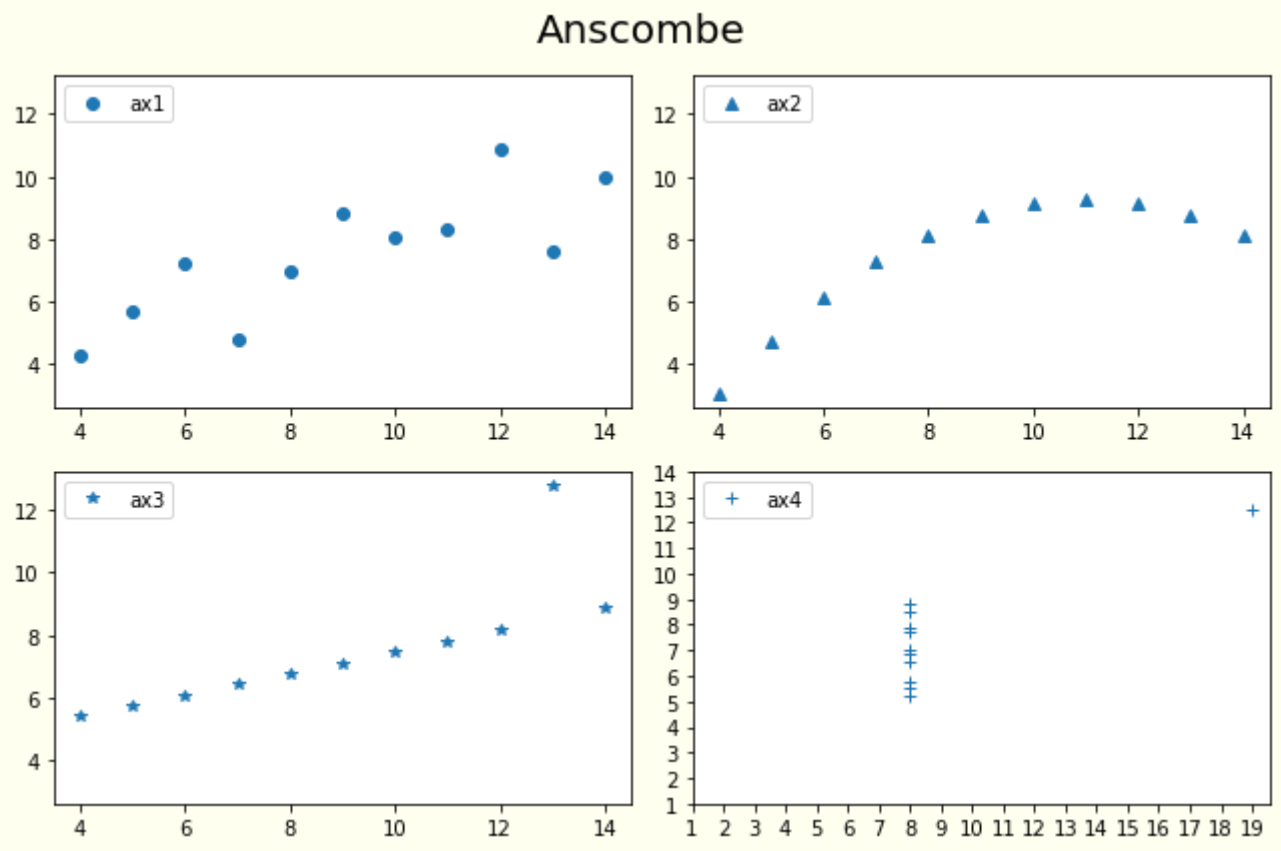
```
ax.plot(x,y)
```

4. 축 공유하기 : 어떤 axes의 축을 공유할 것인지 지정한다.

```
sharex=axes객체, sharey=axes객체
```

```
# 1) figure 객체를 생성한다.
fig = plt.figure(figsize=(9,6), facecolor='ivory')

# 2) 서브플롯을 그릴 axes 객체를 생성한다.
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2, sharex=ax1, sharey=ax1)
ax3 = fig.add_subplot(2,2,3, sharex=ax1, sharey=ax1)
ax4 = fig.add_subplot(2,2,4)


# 3) axes 객체에 그래프를 그린다.
# 4) 축 공유하기 : 어떤 axes의 축을 공유할 것인지 지정한다.
ax1.plot(df1['x'],df1['y'],'o', label='ax1')
ax2.plot(df2['x'],df2['y'],'^', label='ax2')
ax3.plot(df3['x'],df3['y'],'*', label='ax3')
ax4.plot(df4['x'],df4['y'],'+', label='ax4')

# 4) 틱 변경하기
ax4.set_xticks(range(1,20,1))
ax4.set_yticks(range(1,15,1))

# 5) 범례 표시하기
ax1.legend(loc=2)
ax2.legend(loc=2)
ax3.legend(loc=2)
ax4.legend(loc=2)

# 6) figure 제목 추가하기
fig.suptitle('Anscombe', size=20)

# 8) 그래프 크기,간격 최적화하기
```

```
fig.tight_layout()

plt.show()
```



이미지로 그래프 저장하기

- **fig.savefig(파일명, dpi=해상도)**
- **해상도 default : 100**

```
fig.savefig('img100.png')
```

```
fig.savefig('img150.png', dpi=150)
```

[학습목표]

막대그래프를 그리고, 막대그래프의 여러가지 옵션을 지정할 수 있다.

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

```
# 그래프에 한글 설정
plt.rc('font',family='Malgun Gothic')

# 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

샘플데이터

- **어느 두 매장의 요일별 평균 매출액**

```
df1 = pd.DataFrame({'요일':['월','화','수','목','금','토','일'],
                    '매출액':[10000,9000,11000,8000,13000,15000,14000]})
df1
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	요일	매출액
0	월	10000
1	화	9000
2	수	11000
3	목	8000
4	금	13000
5	토	15000
6	일	14000

```
df2 = pd.DataFrame({'요일':['월','화','수','목','금','토','일'],
                    '매출액':[9000,9500,13000,7000,12000,14000,11000]})
df2
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	요일	매출액
0	월	9000
1	화	9500
2	수	13000
3	목	7000
4	금	12000
5	토	14000
6	일	11000

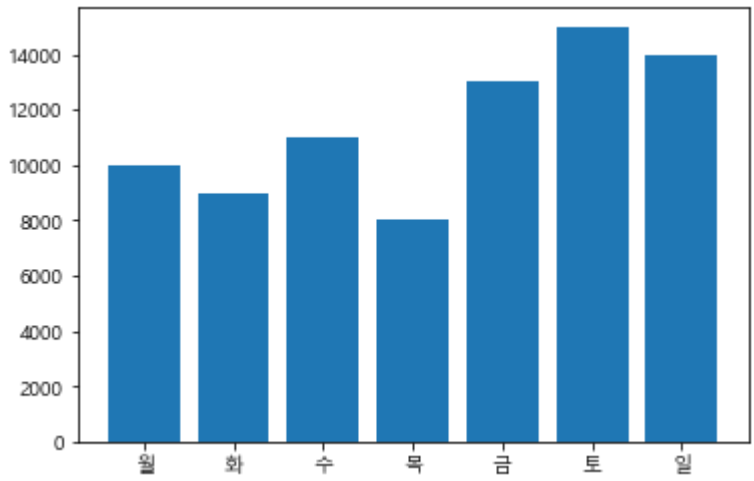
막대그래프

- **plt.bar(x축데이터,y축데이터)**
- **plt.barh(x축데이터,y축데이터)**

세로 막대 그래프

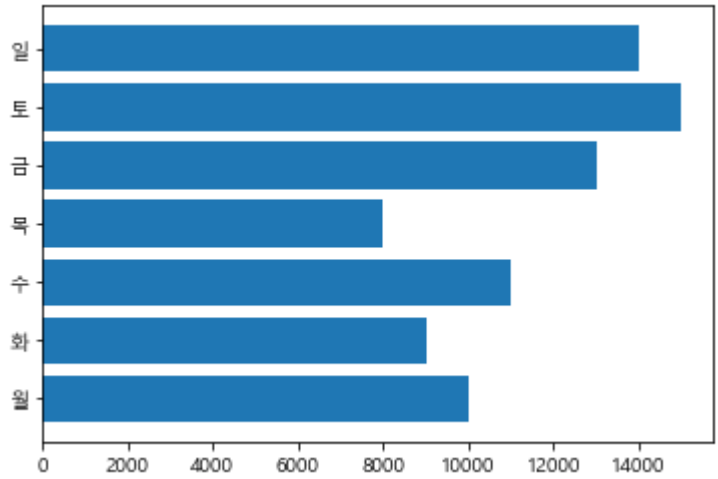
```
plt.bar(df1['요일'],df1['매출액'])
```

```

```



가로 막대 그래프

```
plt.barh(df1['요일'],df1['매출액'])
```

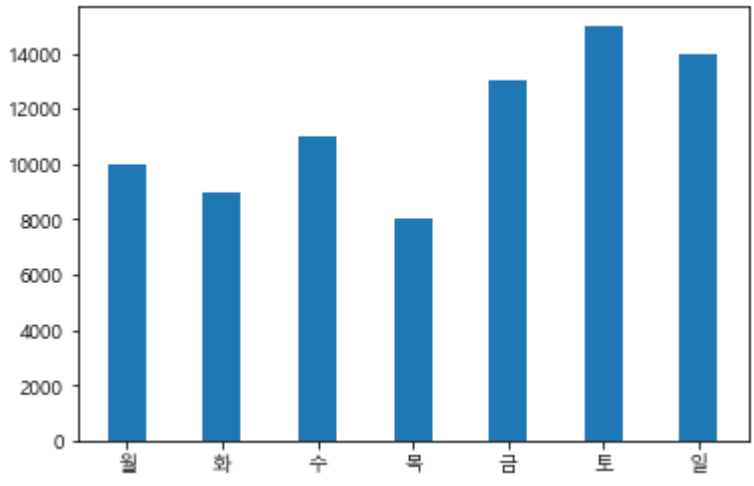


막대 폭 지정

세로 막대 그래프

- **width = 0~1사이의 실수(default:0.8)**

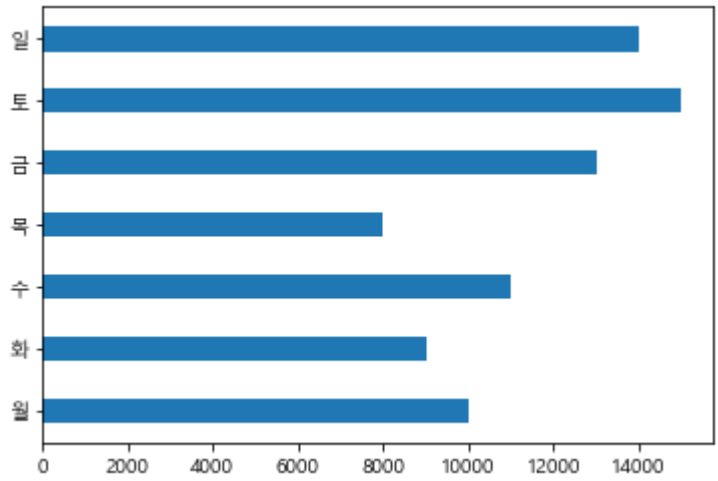
```
plt.bar(df1['요일'],df1['매출액'], width=0.4)
```



가로막대 그래프

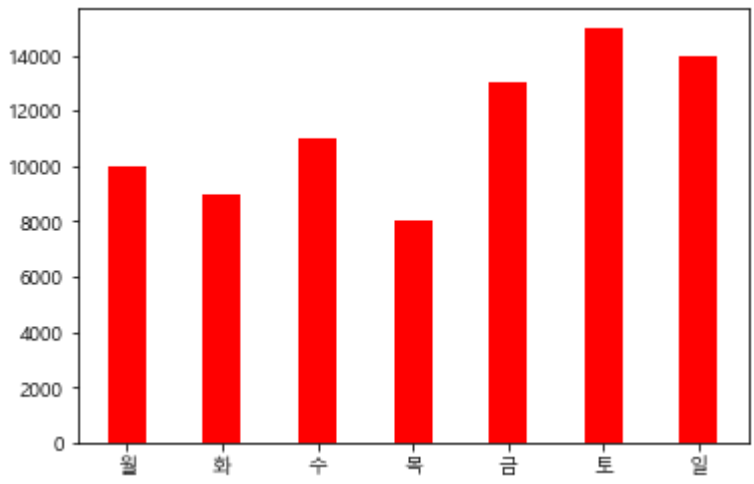
- **height = 0~1사이의 실수(default:0.8)**

```
plt.barh(df1['요일'],df1['매출액'],height=0.4)
```



막대 색상 지정

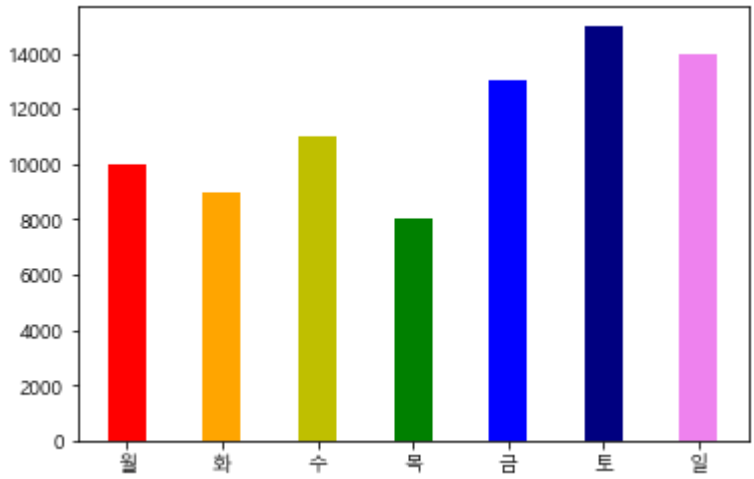
```
plt.bar(df1['요일'],df1['매출액'], width=0.4, color='r')
```



막대마다 다른 색 지정

- **['r','orange','y','g','b','navy','violet']**

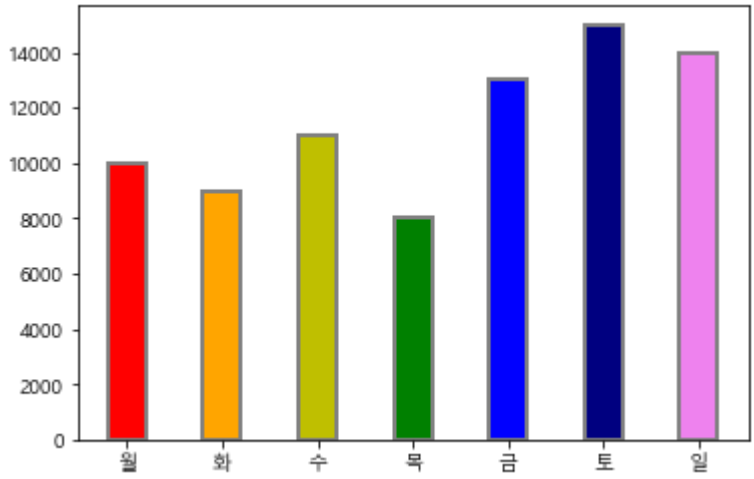
```
plt.bar(df1['요일'],df1['매출액'], width=0.4, color=
['r','orange','y','g','b','navy','violet'])
```



막대 테두리

- **edgecolor = 테두리 색상**
- **linewidth = 테두리 두께**

```
plt.bar(df1['요일'],df1['매출액'], width=0.4, color=
['r','orange','y','g','b','navy','violet']
       ,edgecolor='gray',linewidth=2)
```



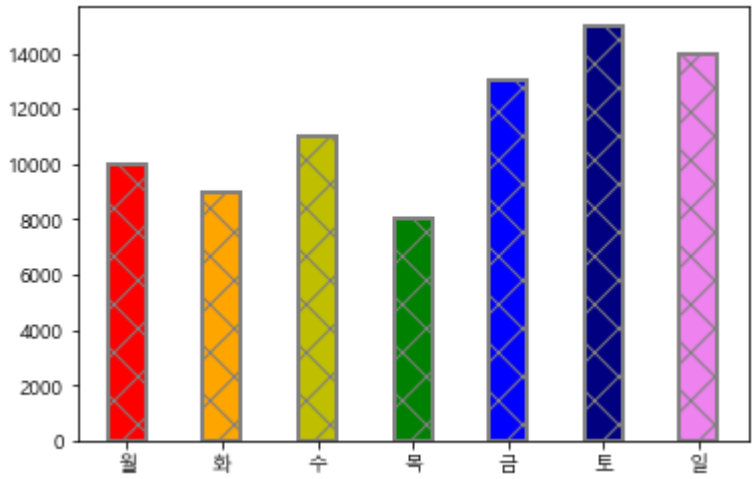
막대 패턴 지정하기

막대에 패턴 지정

- **hatch 파라미터에 기호 전달 : '/', '', '|', '-', '+', 'x', 'o', 'O', '.', '*'**

```
plt.bar(df1['요일'],df1['매출액'], width=0.4, color=
['r','orange','y','g','b','navy','violet']
       ,edgecolor='gray',linewidth=2,hatch='x')
```

```

```



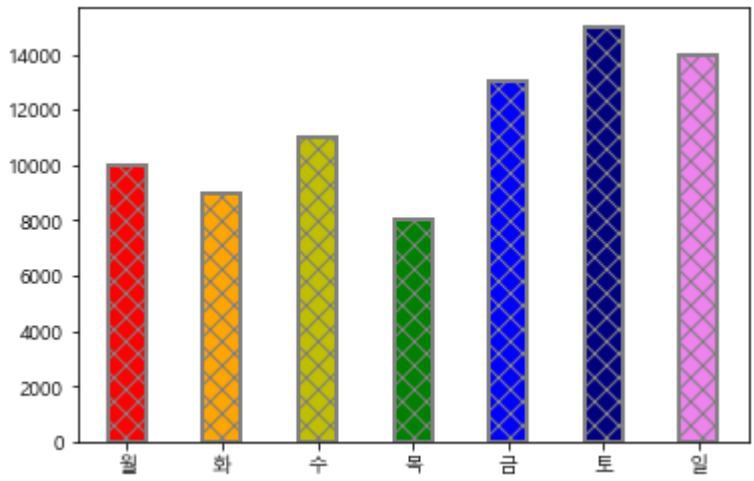
패턴의 밀도 지정

- **패턴기호의 개수로 밀도를 조정한다.**

```
plt.bar(df1['요일'],df1['매출액'], width=0.4, color=
['r','orange','y','g','b','navy','violet']
```

```
        ,edgecolor='gray',linewidth=2,hatch='xx')
```

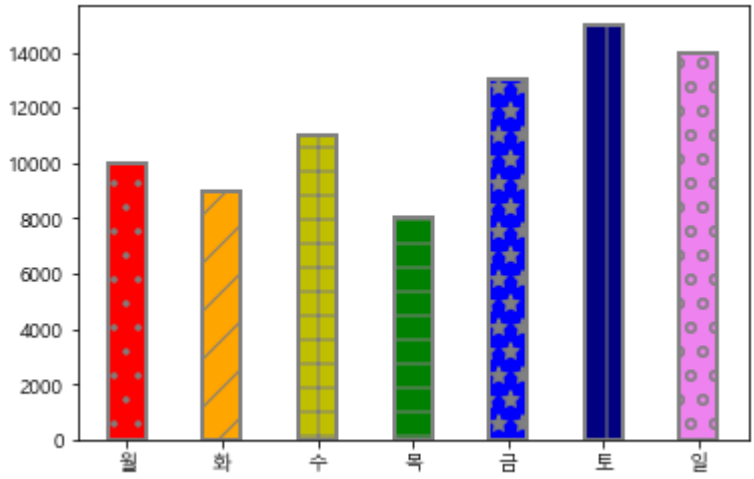
```

```



막대마다 다른 패턴 지정

- **막대그래프를 객체로 받는다.**
- **막대마다 다른 패턴을 지정한다.**
- **set_hatch(기호)**

```
bars = plt.bar(df1['요일'],df1['매출액'], width=0.4, color=
['r','orange','y','g','b','navy','violet']
       ,edgecolor='gray',linewidth=2)

bars[0].set_hatch('.')
bars[1].set_hatch('/')
bars[2].set_hatch('+')
bars[3].set_hatch('-')
bars[4].set_hatch('*')
bars[5].set_hatch('|')
bars[6].set_hatch('o')
```



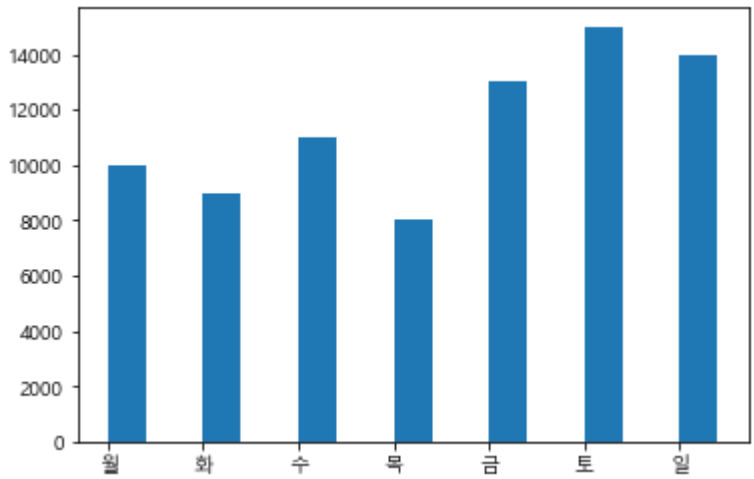
막대 위치 지정

`align = center/edge`

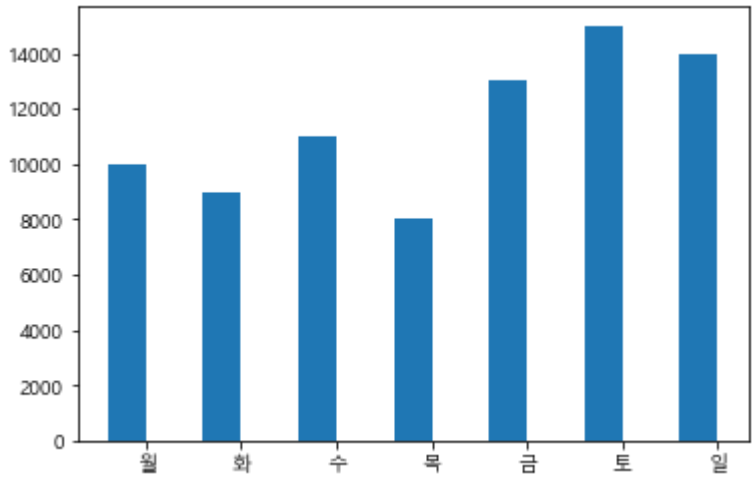
- **디폴트: center**
- **edge로 지정하면 막대의 왼쪽 끝과 틱을 맞춘다.**
- **막대의 오른쪽 끝과 틱을 맞추려면 width를 음수로 지정한다.**

```
plt.bar(df1['요일'],df1['매출액'], width=0.4, align='edge')
```

```

```




```
plt.bar(df1['요일'],df1['매출액'], width=-0.4, align='edge')
```



두 개의 막대그래프 비교하기

```
df1
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

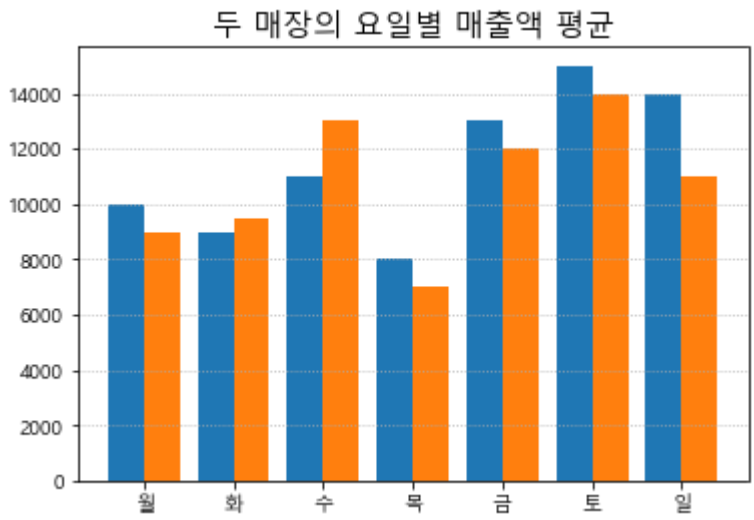
	요일	매출액
0	월	10000
1	화	9000
2	수	11000
3	목	8000
4	금	13000
5	토	15000
6	일	14000

```
df2
```

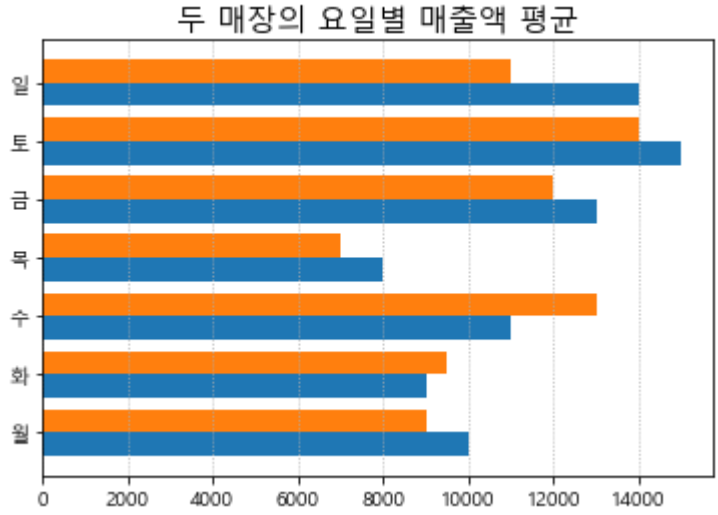
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	요일	매출액
0	월	9000
1	화	9500
2	수	13000
3	목	7000
4	금	12000
5	토	14000
6	일	11000

```
plt.bar(df1['요일'],df1['매출액'], width=-0.4, align='edge')
plt.bar(df2['요일'],df2['매출액'], width=0.4, align='edge')
plt.title('두 매장의 요일별 매출액 평균', size=15)
plt.grid(axis='y', ls=':')
```




```
plt.barh(df1['요일'],df1['매출액'], height=-0.4, align='edge')
plt.barh(df2['요일'],df2['매출액'], height=0.4, align='edge')
plt.title('두 매장의 요일별 매출액 평균', size=15)
plt.grid(axis='x', ls=':')
```



[학습목표]

히트맵을 이용하여 색으로 값의 크기를 표현하여 비교할 수 있다.

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

히트맵

- **plt.pcolor(2차원데이터)**
- **컬러맵의 종류**

https://matplotlib.org/3.3.1/tutorials/colors/colormaps.html

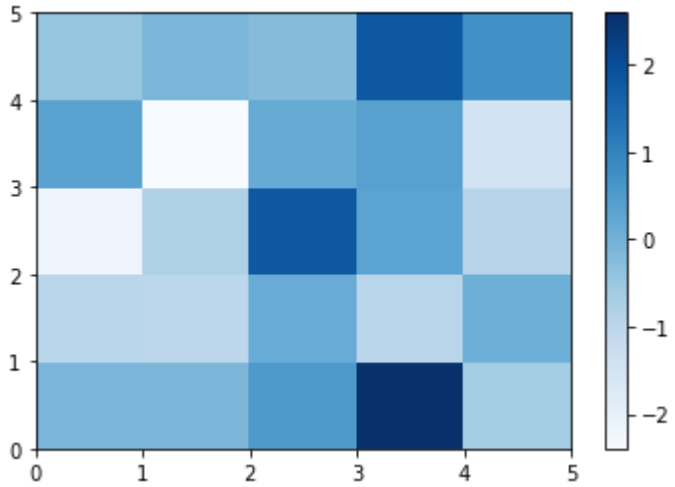
```
# 샘플데이터
import numpy as np
arr = np.random.standard_normal((5,5))
df = pd.DataFrame(arr)
df
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	0	1	2	3	4

	0	1	2	3	4
0	-0.089481	-0.111585	0.541304	2.589565	-0.639976
1	-0.962396	-0.982419	0.150846	-0.919216	0.060101
2	-2.124236	-0.795101	1.813740	0.344869	-0.907642
3	0.365848	-2.393687	0.162361	0.383582	-1.450279
4	-0.420488	-0.086968	-0.219094	1.819900	0.728049

```
# 히트맵 그리기
plt.pcolor(df, cmap='Blues')
plt.colorbar()
```



히트맵 예제

- **타이나틱호의 연령대_객실등급별 승선자수**

데이터 준비

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
titanic
```

```
.dataframe tbody tr th {
    vertical-align: top;
```

```
}

.dataframe thead th {
    text-align: right;
}
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	
0	0	3	male	22.0	1	0	7.2500	S	Third	
1	1	1	female	38.0	1	0	71.2833	C	First	
2	1	3	female	26.0	0	0	7.9250	S	Third	
3	1	1	female	35.0	1	0	53.1000	S	First	
4	0	3	male	35.0	0	0	8.0500	S	Third	
...	...	...	...	...	...	...	...	...	...	
886	0	2	male	27.0	0	0	13.0000	S	Second	

891 rows × 15 columns

데이터 전처리

결측치 처리

```
# 결측치 확인
titanic.isnull().sum()
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

```
# 컬럼 삭제
titanic = titanic.drop(columns=['deck'])
```

```
# 결측치 삭제
titanic = titanic.dropna()
```

```
titanic.isnull().sum()
```

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

```
titanic.shape
```

```
(712, 14)
```

연령대 컬럼 생성

```
titanic['agerange'] = (titanic['age']/10).astype('int')*10
```

피벗테이블 : 연령대-객실등급 별 승선자 수

```
titanic_pivot = titanic.pivot_table(index='class', columns='agerange',
values='survived', aggfunc='count')
```

```
titanic_pivot
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

agerange	0	10	20	30	40	50	60	70	80
class									
First	3	18	34	49	37	27	12	3	1
Second	17	18	53	48	18	15	3	1	0
Third	42	66	133	69	34	6	3	2	0

히트맵

matplotlib

```
range(0,len(titanic_pivot.columns),1)
```

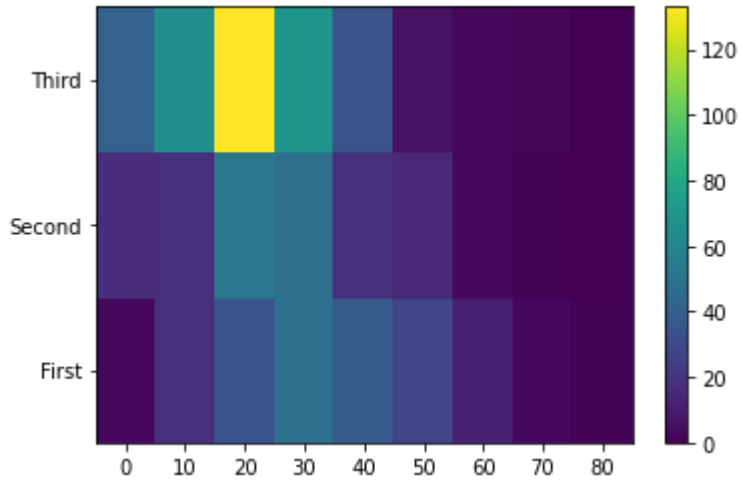
```
range(0, 9)
```

```
np.arange(0.5,len(titanic_pivot.columns),1)
```

```
array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5])
```

```
plt.pcolor(titanic_pivot)
plt.colorbar()
```

```
plt.xticks(np.arange(0.5,len(titanic_pivot.columns),1),
labels=titanic_pivot.columns)
plt.yticks(np.arange(0.5,len(titanic_pivot.index),1), labels=titanic_pivot.index)
plt.show()
```



seaborn

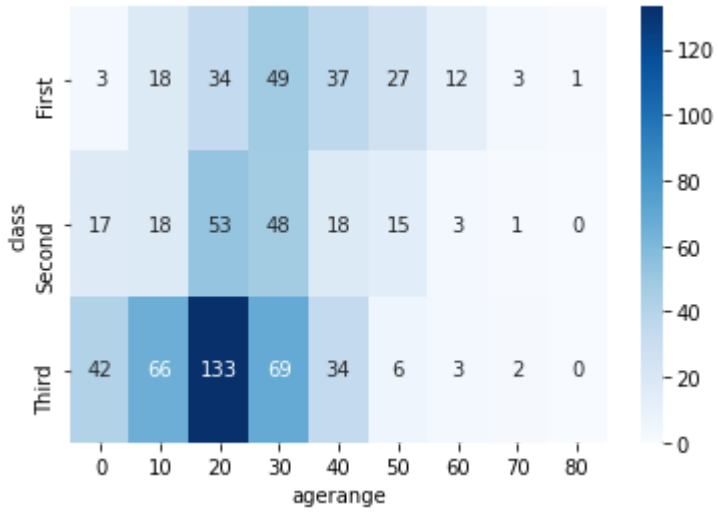
- **heatmap(data=2차원데이터)**

cmap=컬러맵 : 컬러맵 지정

annot=True : 수치 표시

fmt='d' : 정수로 표시

```
sns.heatmap(titanic_pivot, cmap='Blues', annot=True, fmt='d')
```



[학습목표]

산점도로 x축, y축 데이터의 관계를 파악하고, 점의 색과 크기로 정보를 표현할 수 있다.

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

```
import matplotlib as mpl

# 그래프에 한글 설정
mpl.rcParams['font.family'] = 'NanumSquare'

# 유니코드에서 그래프에 마이너스 기호 깨지는 문제 해결
mpl.rcParams['axes.unicode_minus'] = False
```

스캐터플롯

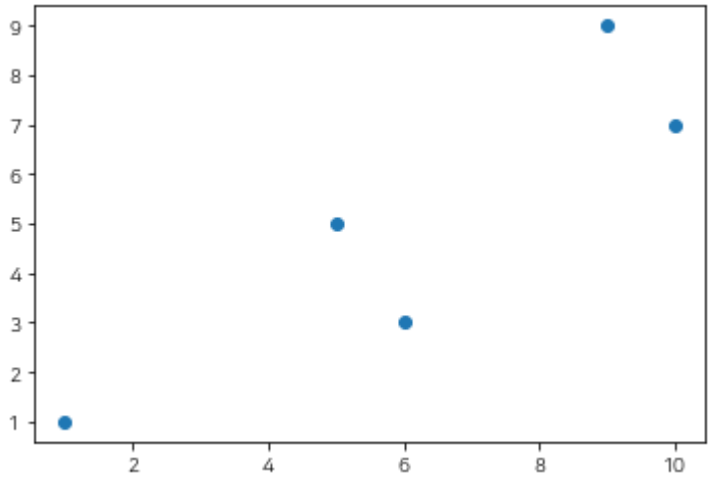
- **점의 크기, 색깔로 정보를 표현할 수 있다. (버블차트)**
- **크기는 s, 색깔은 c로 지정한다.**

```
# 데이터
x = [1,5,6,9,10]
y = [1,5,3,9,7]
```

기본

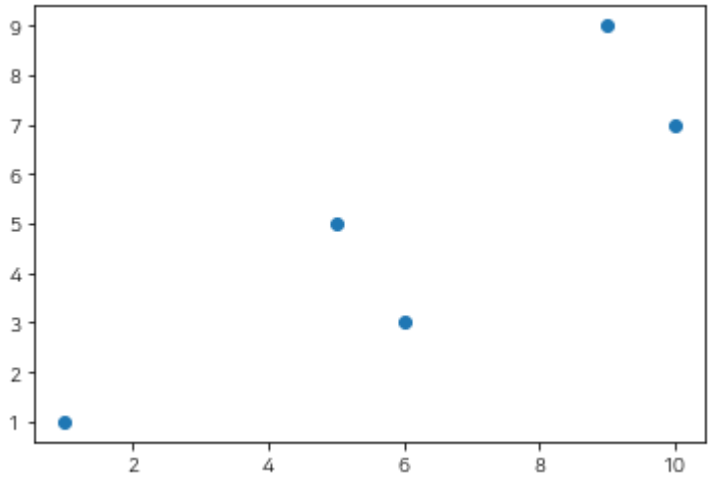
```
plt.scatter(x,y)
```

```

```




```
plt.plot(x,y,'o')
```

```
[]
```



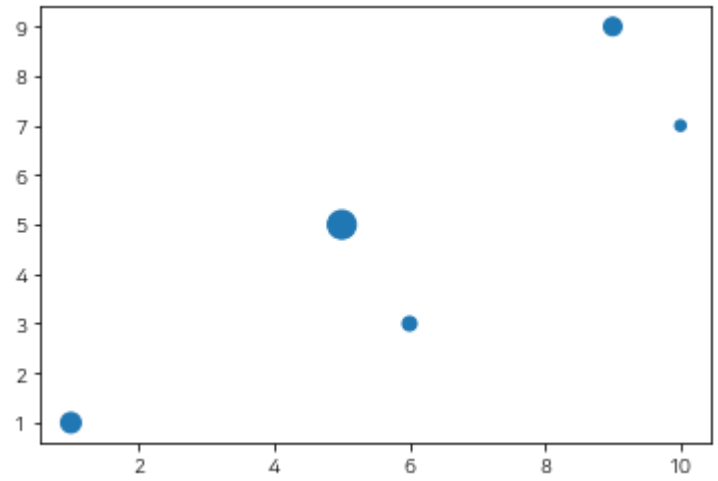
점의 크기

s=크기공통지정

s=크기목록

- **plt.scatter()의 파라미터 s로 점 크기를 지정한다.**
- **점 크기는 공통된 크기로 지정할 수도 있고, 점마다 다른 크기로 지정할 수도 있다.**

```
plt.scatter(x,y,s=[100,200,50,80,30])
```



점의 색상

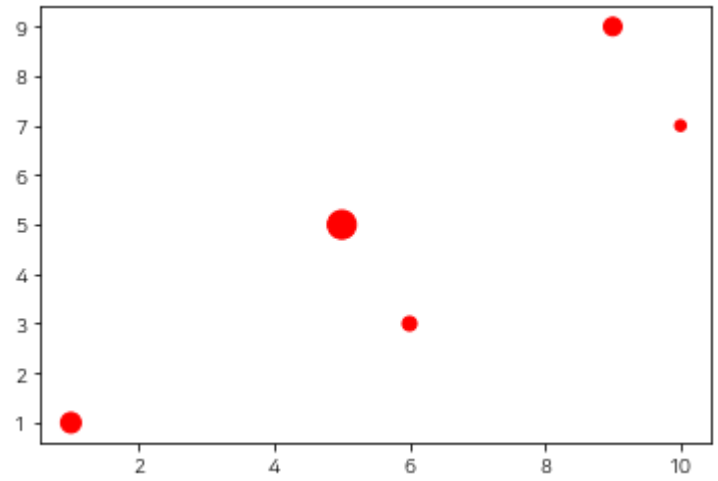
c=색상공통지정

c=색상목록

- **plt.scatter()의 파라미터 c로 점 색상을 지정한다.**
- **점 색상은 공통된 색상으로 지정할 수도 있고, 점마다 다른 색상으로 지정할 수도 있다.**

단일색 지정

```
plt.scatter(x,y,s=[100,200,50,80,30], c='r')
```

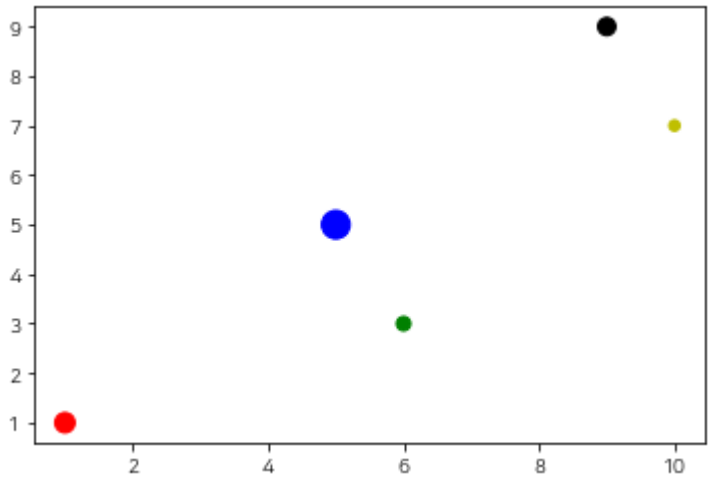


값에 따른 색상 지정

- **['r','b','g','k','y']**

```
plt.scatter(x,y,s=[100,200,50,80,30], c=['r','b','g','k','y'])
```

```

```

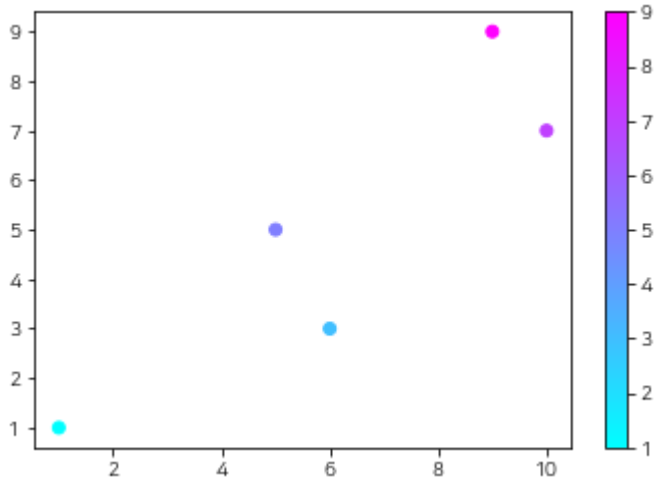


컬러맵으로 색상 지정

- **컬러맵 지정 : cmap=컬러맵**
- **컬러바 표시 : plt.colorbar()**
- **컬러맵 : https://matplotlib.org/3.3.1/tutorials/colors/colormaps.html**

```
plt.scatter(x,y,c=y, cmap='cool')
plt.colorbar()
```

```

```



```
# 컬러맵 종류
plt.colormaps()
```

```
['Accent',
 'Accent_r',
 'Blues',
 'Blues_r',
 'BrBG',
 'BrBG_r',
 'BuGn',
 'BuGn_r',
 'BuPu',
 'BuPu_r',
 'CMRmap',
 'CMRmap_r',
 'Dark2',
 'Dark2_r',
 'GnBu',
 'GnBu_r',
 'Greens',
 'Greens_r',
 'Greys',
 'Greys_r',
 'OrRd',
 'OrRd_r',
 'Oranges',
 'Oranges_r',
 'PRGn',
 'PRGn_r',
 'Paired',
 'Paired_r',
 'Pastel1',
 'Pastel1_r',
 'Pastel2',
 'Pastel2_r',
 'PiYG',
 'PiYG_r',
 'PuBu',
 'PuBuGn',
 'PuBuGn_r',
 'PuBu_r',
 'PuOr',
 'PuOr_r',
 'PuRd',
 'PuRd_r',
 'Purples',
 'Purples_r',
 'RdBu',
 'RdBu_r',
 'RdGy',
 'RdGy_r',
```

```
    'RdPu',
    'RdPu_r',
    'RdYlBu',
    'RdYlBu_r',
    'RdYlGn',
    'RdYlGn_r',
    'Reds',
    'Reds_r',
    'Set1',
    'Set1_r',
    'Set2',
    'Set2_r',
    'Set3',
    'Set3_r',
    'Spectral',
    'Spectral_r',
    'Wistia',
    'Wistia_r',
    'YlGn',
    'YlGnBu',
    'YlGnBu_r',
    'YlGn_r',
    'YlOrBr',
    'YlOrBr_r',
    'YlOrRd',
    'YlOrRd_r',
    'afmhot',
    'afmhot_r',
    'autumn',
    'autumn_r',
    'binary',
    'binary_r',
    'bone',
    'bone_r',
    'brg',
    'brg_r',
    'bwr',
    'bwr_r',
    'cividis',
    'cividis_r',
    'cool',
    'cool_r',
    'coolwarm',
    'coolwarm_r',
    'copper',
    'copper_r',
    'cubehelix',
    'cubehelix_r',
    'flag',
    'flag_r',
    'gist_earth',
    'gist_earth_r',
    'gist_gray',
    'gist_gray_r',
```

```
    'gist_heat',
    'gist_heat_r',
    'gist_ncar',
    'gist_ncar_r',
    'gist_rainbow',
    'gist_rainbow_r',
    'gist_stern',
    'gist_stern_r',
    'gist_yarg',
    'gist_yarg_r',
    'gnuplot',
    'gnuplot2',
    'gnuplot2_r',
    'gnuplot_r',
    'gray',
    'gray_r',
    'hot',
    'hot_r',
    'hsv',
    'hsv_r',
    'inferno',
    'inferno_r',
    'jet',
    'jet_r',
    'magma',
    'magma_r',
    'nipy_spectral',
    'nipy_spectral_r',
    'ocean',
    'ocean_r',
    'pink',
    'pink_r',
    'plasma',
    'plasma_r',
    'prism',
    'prism_r',
    'rainbow',
    'rainbow_r',
    'seismic',
    'seismic_r',
    'spring',
    'spring_r',
    'summer',
    'summer_r',
    'tab10',
    'tab10_r',
    'tab20',
    'tab20_r',
    'tab20b',
    'tab20b_r',
    'tab20c',
    'tab20c_r',
    'terrain',
    'terrain_r',
```

```
 'turbo',
 'turbo_r',
 'twilight',
 'twilight_r',
 'twilight_shifted',
 'twilight_shifted_r',
 'viridis',
 'viridis_r',
 'winter',
 'winter_r']
```

산점도 예제

데이터 준비

- 어느 레스토랑의 팁 데이터
- 지불금액, 팁, 성별, 흡연여부, 요일, 시간대, 테이블인원

```
import seaborn as sns
tips = sns.load_dataset('tips')
tips.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	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

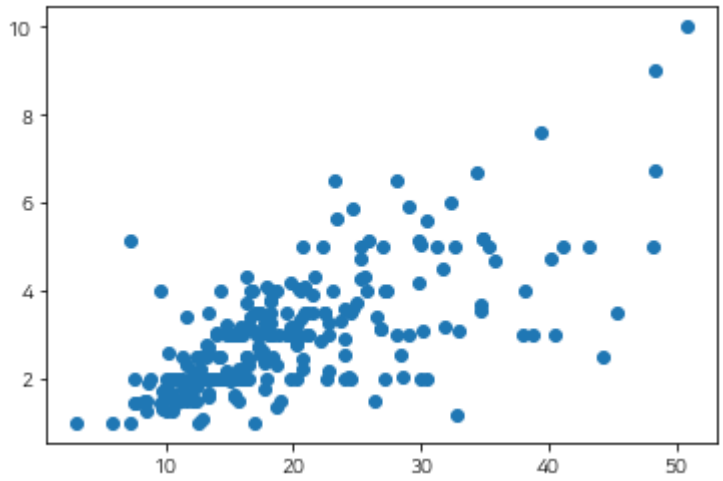
```
tips.shape
```

```
(244, 7)
```

지불금액과 팁의 관계

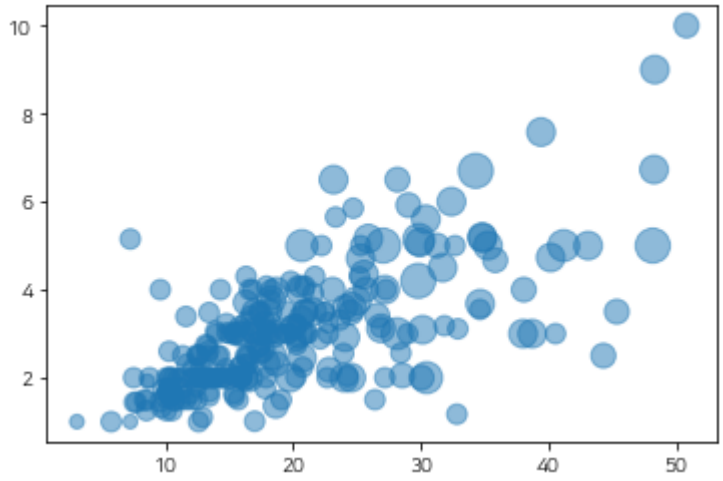
- **지불금액이 클수록 팁도 많이 줄까?**

```
plt.scatter(tips['total_bill'], tips['tip'])
```



점의 크기로 테이블 인원 표시

```
plt.scatter(tips['total_bill'], tips['tip'], s=tips['size']*50, alpha=0.5)
```



점의 색으로 성별 표시

```
# 성별에 따른 color 컬럼 추가
def set_color(x):
    if x=='Male':
        return 'blue'
    elif x=='Female':
        return 'red'

tips['color'] = tips['sex'].apply(set_color)
```

```
tips
```

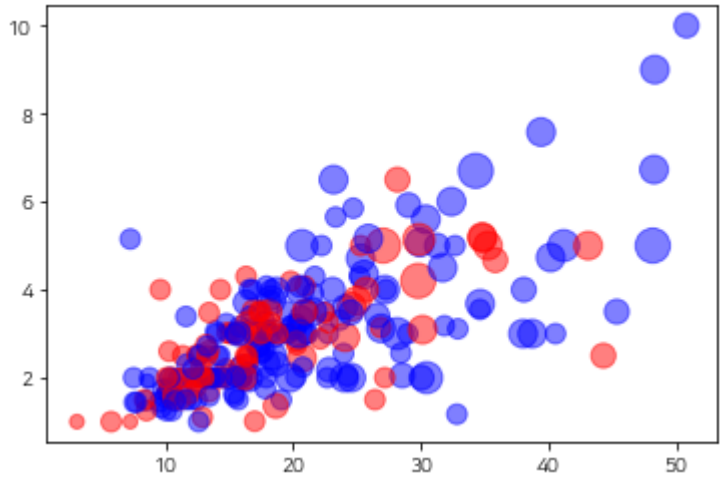
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	total_bill	tip	sex	smoker	day	time	size	color
0	16.99	1.01	Female	No	Sun	Dinner	2	red
1	10.34	1.66	Male	No	Sun	Dinner	3	blue
2	21.01	3.50	Male	No	Sun	Dinner	3	blue
3	23.68	3.31	Male	No	Sun	Dinner	2	blue
4	24.59	3.61	Female	No	Sun	Dinner	4	red
...	...	...	...	...	...	...	...	...

	total_bill	tip	sex	smoker	day	time	size	color

244 rows × 8 columns

```
plt.scatter(tips['total_bill'], tips['tip'], s=tips['size']*50, alpha=0.5,
c=tips['color'])
```



[학습목표]

히스토그램으로 데이터의 도수분포를 표현할 수 있다.

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

```
# 그래프에 한글 설정
plt.rcParams['font.family'] = 'Malgun Gothic'

# 유니코드에서 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

히스토그램

- **자료의 도수분포를 나타내는 그래프**

수치를 나타내는 자료를 일정 구간(계급)으로 나누어 각 구간별 값의 개수(도수)를 나타낸다.

- **x축은 계급(값의 구간), y축은 도수(구간별 값의 개수)를 나타냄**

샘플데이터

- **125명의 점수 데이터**

```
scores =
[0,10,15,15,15,16,19,20,21,25,25,26,26,29,30,35,36,37,39,40,41,41,44,45,45,45,45,4
7,

50,50,50,50,51,51,51,53,54,55,55,56,60,61,62,62,63,64,65,65,65,65,66,66,66,66,66,

67,68,68,69,70,70,70,70,70,70,70,70,71,71,71,71,71,72,72,72,72,73,74,74,74,75,75,

76,76,76,77,77,77,77,78,78,78,78,78,79,79,79,79,80,80,80,80,80,80,81,81,81,82,82,
          85,85,85,88,88,89,90,90,90,93,93,95,95,95,97,100]
```

```
len(scores)
```

```
125
```

히스토그램

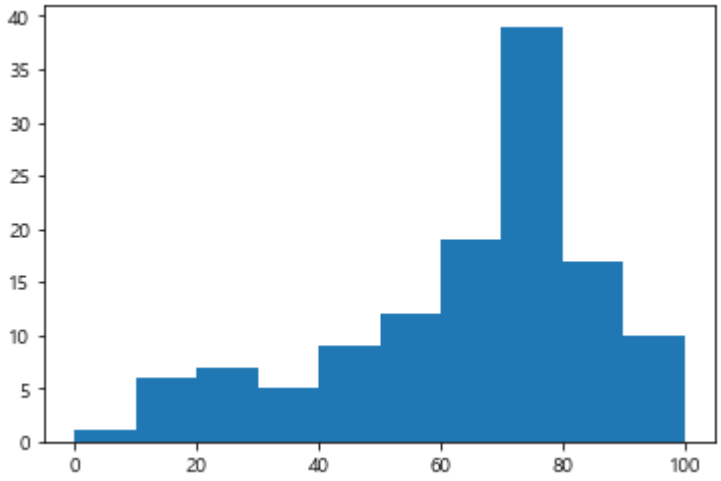
기본 히스토그램

- `plt.hist(data)`

default : 10개의 구간으로 나누어 도수분포를 보여준다.

```
plt.hist(scores)
```

```
(array([ 1.,  6.,  7.,  5.,  9., 12., 19., 39., 17., 10.]),
 array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.]),
 )
```

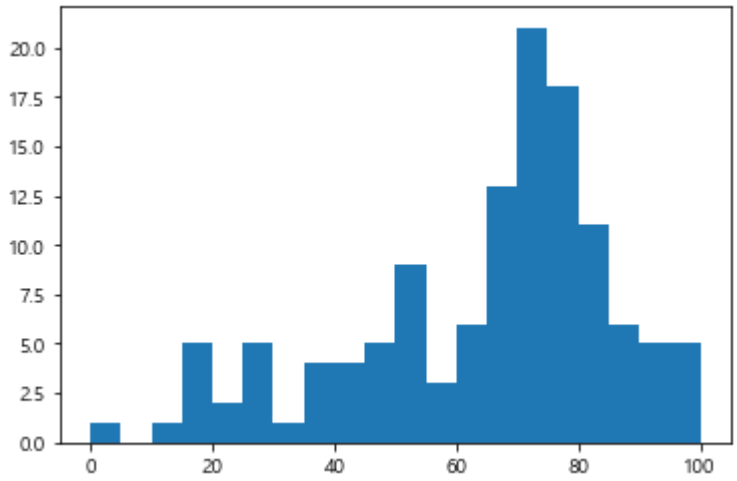


구간의 개수 변경

`bins=구간의 개수`

```
plt.hist(scores, bins=20)
```

```
(array([ 1.,  0.,  1.,  5.,  2.,  5.,  1.,  4.,  4.,  5.,  9.,  3.,  6.,
        13., 21., 18., 11.,  6.,  5.,  5.]),
 array([  0.,   5.,  10.,  15.,  20.,  25.,  30.,  35.,  40.,  45.,  50.,
         55.,  60.,  65.,  70.,  75.,  80.,  85.,  90.,  95., 100.]),
 )
```

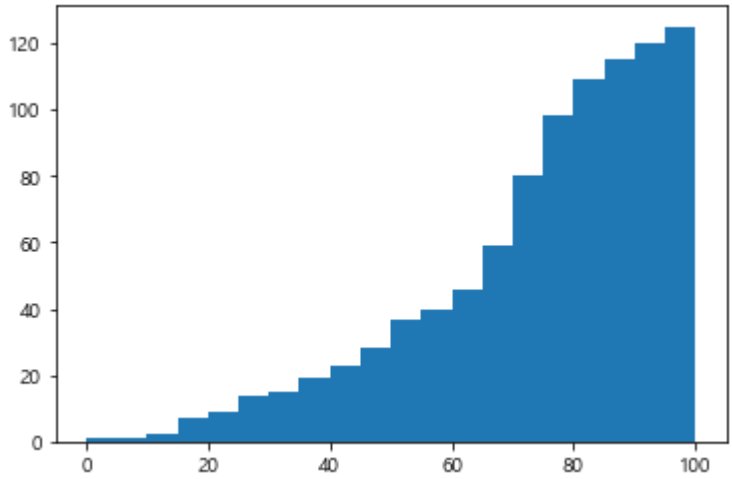


누적 히스토그램

`cumulative=True`

```
plt.hist(scores, bins=20, cumulative=True)
```

```
(array([  1.,   1.,   2.,   7.,   9.,  14.,  15.,  19.,  23.,  28.,  37.,
         40.,  46.,  59.,  80.,  98., 109., 115., 120., 125.]),
 array([  0.,   5.,  10.,  15.,  20.,  25.,  30.,  35.,  40.,  45.,  50.,
         55.,  60.,  65.,  70.,  75.,  80.,  85.,  90.,  95., 100.]),
 )
```

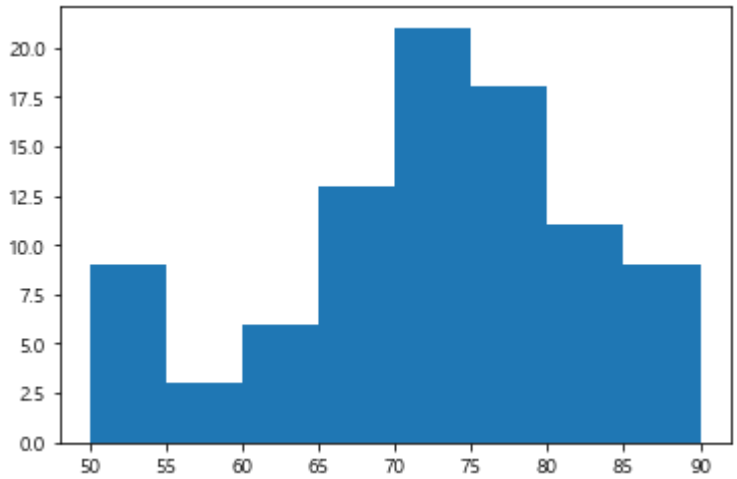


범위 지정

`range=(min,max)`

```
plt.hist(scores, bins=8, range=(50,90))
```

```
(array([ 9.,  3.,  6., 13., 21., 18., 11.,  9.]),
 array([50., 55., 60., 65., 70., 75., 80., 85., 90.]),
 )
```



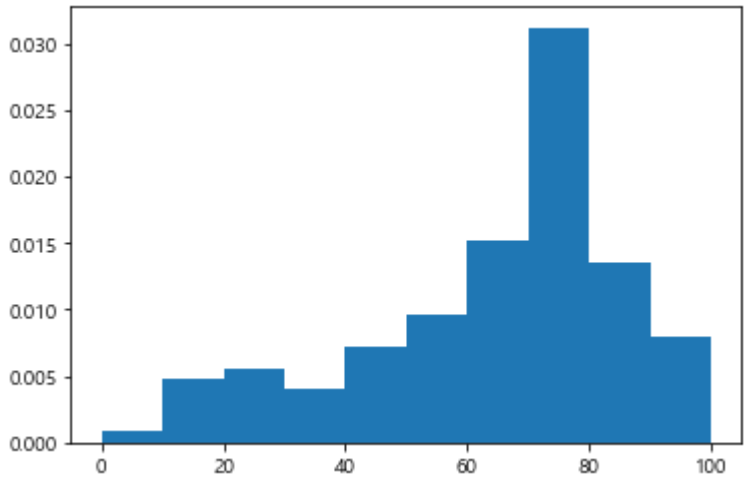
밀도 표시

density=True

도수를 총 개수로 나눈 수치를 y축에 표시

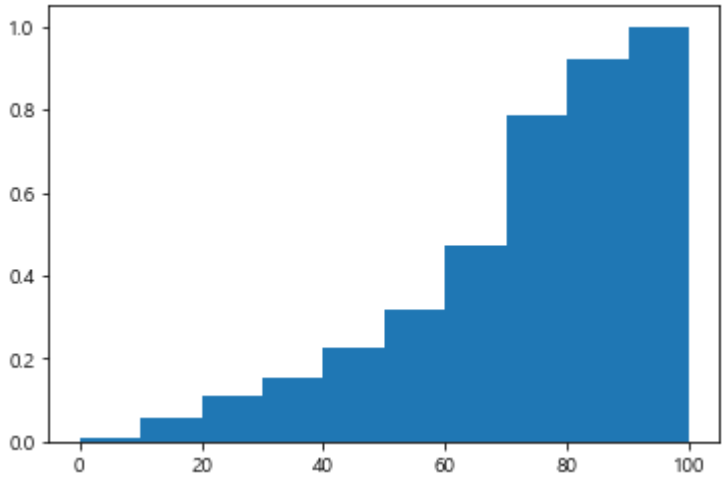
```
plt.hist(scores, density=True)
```

```
(array([0.0008, 0.0048, 0.0056, 0.004 , 0.0072, 0.0096, 0.0152, 0.0312,
        0.0136, 0.008 ]),
 array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.]),
 )
```




```
plt.hist(scores, density=True, cumulative=True)
```

```
(array([0.008, 0.056, 0.112, 0.152, 0.224, 0.32 , 0.472, 0.784, 0.92 ,
        1.   ]),
 array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.]),
 )
```

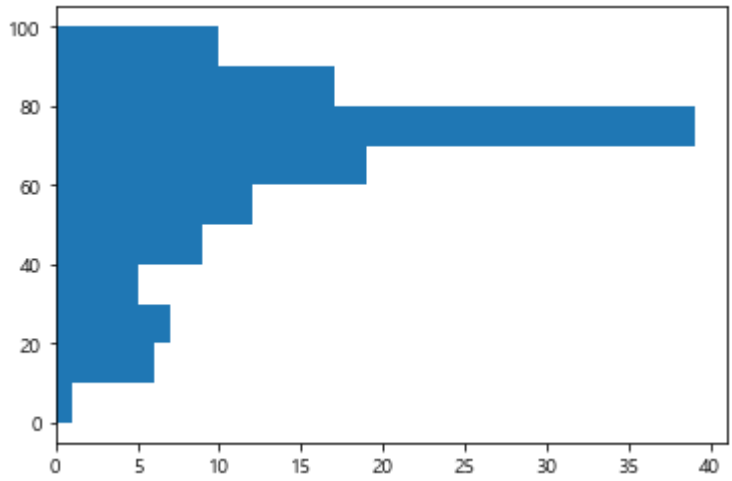


가로 히스토그램

`orientation='horizontal'`

```
plt.hist(scores,orientation='horizontal')
```

```
(array([ 1.,  6.,  7.,  5.,  9., 12., 19., 39., 17., 10.]),
 array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.]),
 )
```

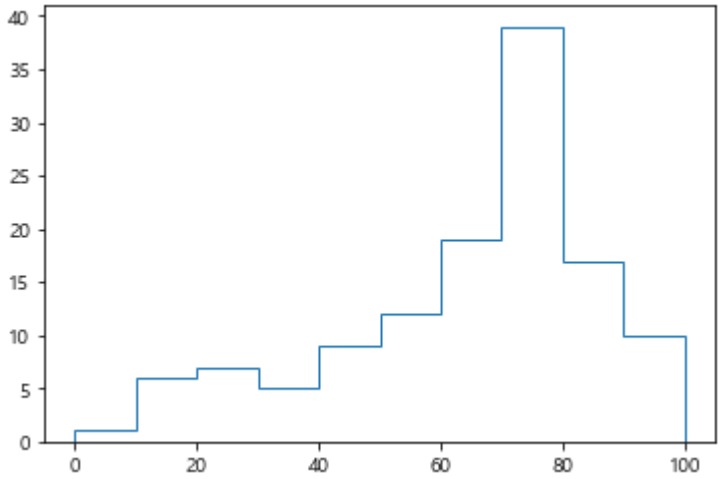


막대 스타일

- **선만 표시** `histtype='step'`

```
plt.hist(scores,histtype='step')
```

```
(array([ 1.,  6.,  7.,  5.,  9., 12., 19., 39., 17., 10.]),
 array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.]),
 [])
```

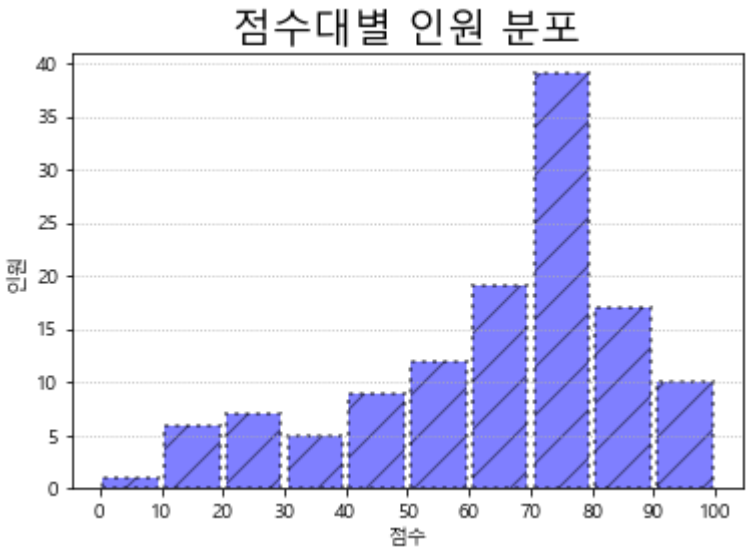


rwidth=막대폭(0~1) color=막대색 alpha=투명도

edgecolor(ec)=선색 linewidth(lw)=선두께 linestyle(ls)=선스타일

hatch=패턴

```
plt.hist(scores, rwidth=0.9, color='b', alpha=0.5, ec='k', lw=2, ls=':',hatch='/')
plt.xticks(range(0,101,10))
plt.grid(axis='y', ls=':')
plt.xlabel('점수')
plt.ylabel('인원')
plt.title('점수대별 인원 분포', size=20)
plt.show()
```



[학습목표]

박스플롯을 이용하여 데이터의 통계와 이상치를 표현할 수 있다.

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

```
# 그래프에 한글 설정
plt.rcParams['font.family'] = 'Malgun Gothic'

# 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

박스플롯

- **데이터로부터 얻어진 아래의 다섯 가지 요약 수치를 사용해서 그려진다.**

최소값

제 1사분위 수 (Q1) : 전체 데이터 중 하위 25%에 해당하는 값

제 2사분위 수 또는 중위수 (Q2)

제 3사분위 수 (Q3) : 전체 데이터 중 상위 25%에 해당하는 값

최대값

- **다른 값들과 동떨어진 값을 이상치로 표현한다.**

어떤 값이 (1/4 지점의 값 - 1.5 * 사분위수 범위) 보다 작거나,

어떤 값이 (3/4 지점의 값 + 1.5 * 사분위수 범위) 보다 크면 그 값을 이상치로 정한다.

- **사분위수 범위 = 3/4 지점의 값 - 1/4 지점의 값**

샘플 데이터

- **125명의 점수 데이터**

```
scores =
pd.Series([0,10,15,15,15,16,19,20,21,25,25,26,26,29,30,35,36,37,39,40,41,41,44,45,
45,45,45,47,

50,50,50,50,51,51,51,53,54,55,55,56,60,61,62,62,63,64,65,65,65,65,66,66,66,66,66,

67,68,68,69,70,70,70,70,70,70,70,70,71,71,71,71,71,72,72,72,72,73,74,74,74,75,75,
```

```
76,76,76,77,77,77,77,78,78,78,78,78,79,79,79,79,80,80,80,80,80,80,81,81,81,82,82,
          85,85,85,88,88,89,90,90,90,93,93,95,95,95,97,100])
```

샘플데이터의 통계값

시리즈.describe()

```
scores.describe()
```

```
count    125.000000
mean      63.416000
std       21.763829
min        0.000000
25%       50.000000
50%       70.000000
75%       78.000000
max      100.000000
dtype: float64
```

이상치 구하기

- **1/4 지점의 값 : 시리즈.quantile(.25)**
- **3/4 지점의 값 : 시리즈.quantile(.75)**
- **이상치**

1/4 지점의 값 - 1.5 * 사분위수 범위

3/4 지점의 값 + 1.5 * 사분위수 범위

```
Q1 = scores.quantile(.25)
print('1/4 지점의 값 : ',Q1)

Q3 = scores.quantile(.75)
print('3/4 지점의 값 : ',Q3)

# 이상치
print('이상치:',Q1 - 1.5*(Q3-Q1),'미만')

# 3/4 지점의 값 + 1.5 * 사분위수 범위
print('이상치:',Q3 + 1.5*(Q3-Q1),'이상')
```

```
1/4 지점의 값 :  50.0
3/4 지점의 값 :  78.0
```

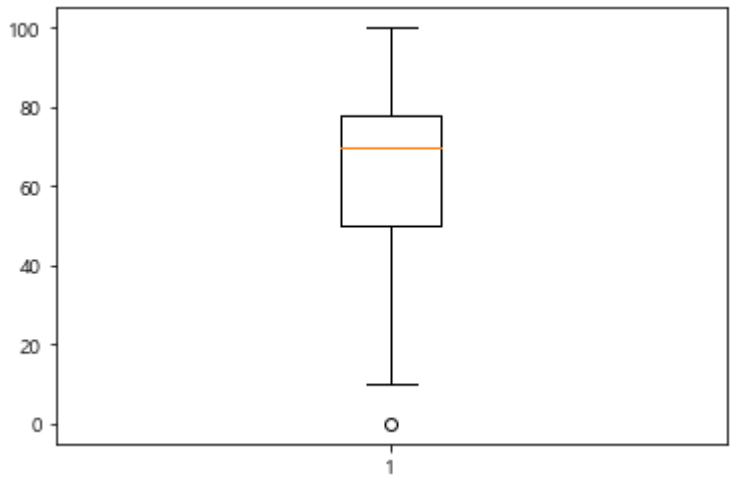
```
이상치: 8.0 미만
이상치: 120.0 이상
```

박스플롯

* plt.boxplot(data)

```
plt.boxplot(scores)
```

```
{'whiskers': [,
  ],
 'caps': [,
  ],
 'boxes': [],
 'medians': [],
 'fliers': [],
 'means': []}
```



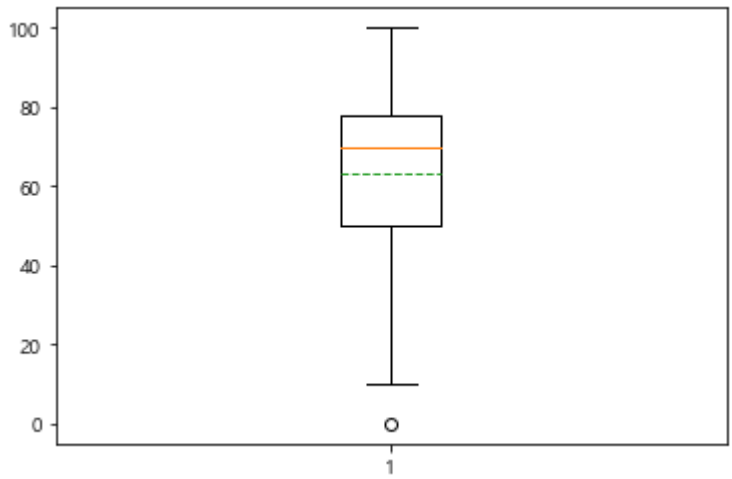
평균 표시하기

- **showmeans=True**
- **meanline=True**

```
plt.boxplot(scores,showmeans=True, meanline=True)
```

```
{'whiskers': [,
  ],
 'caps': [,
  ],
```

```
 'boxes': [],
 'medians': [],
 'fliers': [],
 'means': []}
```

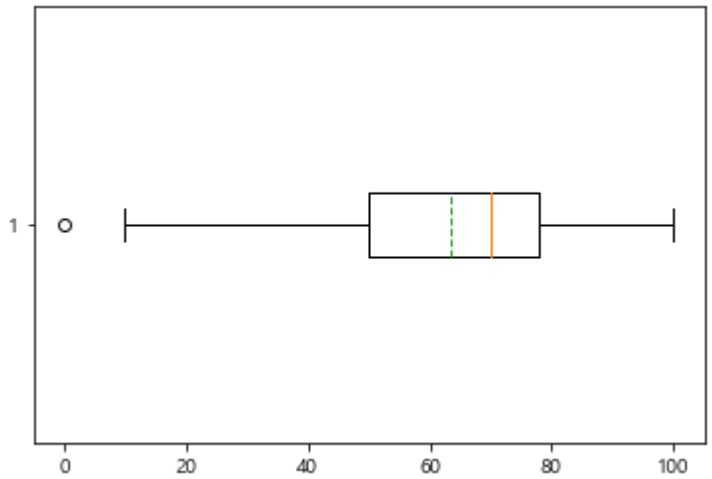


수평 박스플롯

- **vert=False**

```
plt.boxplot(scores,showmeans=True, meanline=True, vert=False)
```

```
{'whiskers': [,
  ],
 'caps': [,
  ],
 'boxes': [],
 'medians': [],
 'fliers': [],
 'means': []}
```



여러개의 데이터 비교하기

샘플 데이터

- **붓꽃 데이터**

```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

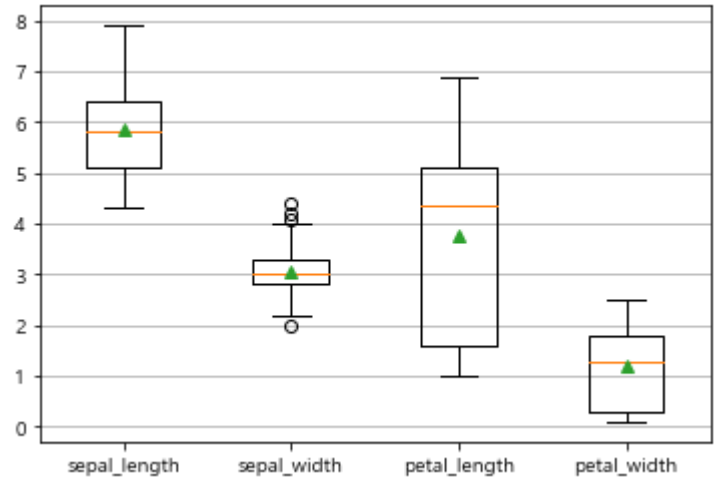
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

여러 개의 데이터 비교하기

- **plt.boxplot(데이터리스트, labels=레이블리스트)**

```
plt.boxplot([ iris['sepal_length'], iris['sepal_width'], iris['petal_length'],
iris['petal_width']    ]
            , labels=['sepal_length', 'sepal_width','petal_length','petal_width']
            , showmeans=True)
plt.grid(axis='y')
plt.show()
```



[학습목표]

바이올린플롯으로 데이터의 범위와 분포를 표현할 수 있다.

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

```
# 그래프에 한글 설정
plt.rcParams['font.family'] = 'Malgun Gothic'

# 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

바이올린플롯

샘플 데이터

- **125명의 점수 데이터**

```
scores =
pd.Series([0,10,15,15,15,16,19,20,21,25,25,26,26,29,30,35,36,37,39,40,41,41,44,45,
45,45,45,47,

50,50,50,50,51,51,51,53,54,55,55,56,60,61,62,62,63,64,65,65,65,65,66,66,66,66,66,

67,68,68,69,70,70,70,70,70,70,70,70,71,71,71,71,71,72,72,72,72,73,74,74,74,75,75,

76,76,76,77,77,77,77,78,78,78,78,78,79,79,79,79,80,80,80,80,80,80,81,81,81,82,82,
          85,85,85,88,88,89,90,90,90,93,93,95,95,95,97,100])
```

```
len(scores)
```

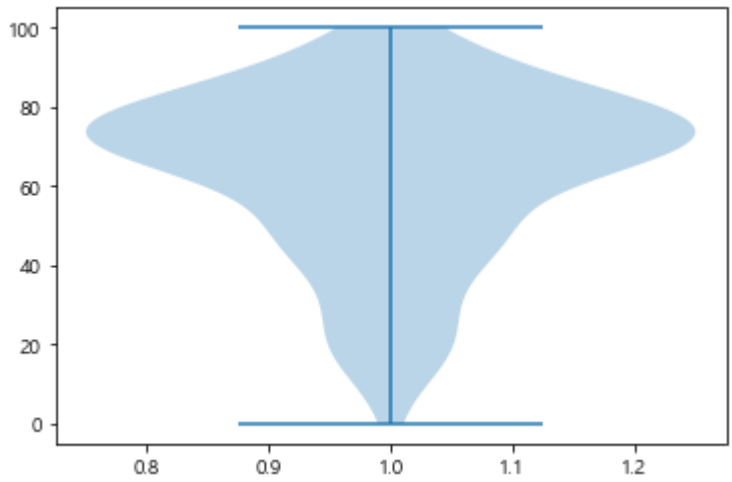
```
125
```

바이올린플롯

- **plt.violinplot(data)**

```
plt.violinplot(scores)
```

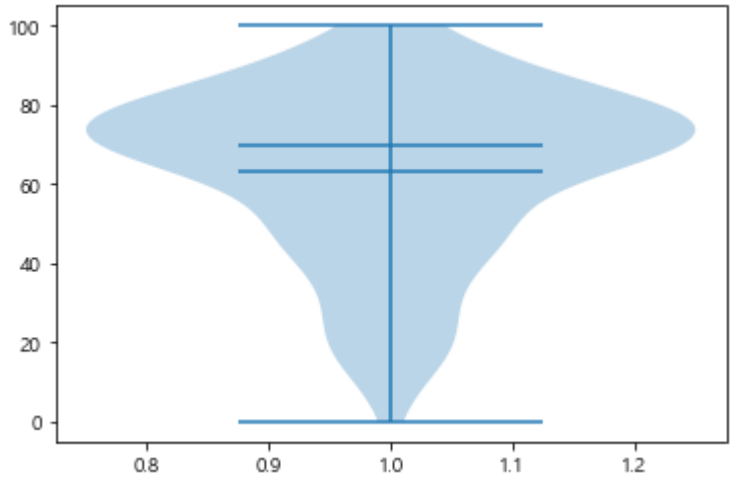
```
{'bodies': [],
 'cmaxes': ,
 'cmins': ,
 'cbars': }
```



최대값, 최소값, 평균값, 중간값 표시

- **showextrema=True/False : 최대값,최소값에 직선 표시(default:True)**
- **showmeans=True/False : 평균값에 직선 표시(default:False)**
- **showmedians=True/False : 중간값에 직선 표시(default:False)**

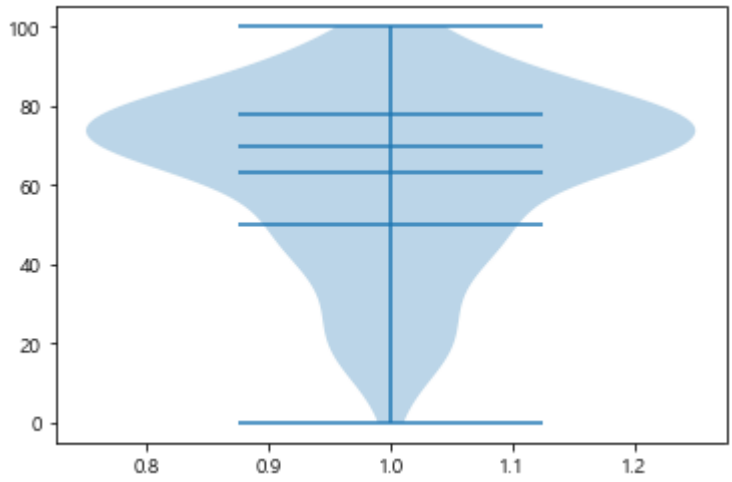
```
plt.violinplot(scores, showextrema=True, showmeans=True, showmedians=True)
plt.show()
```



분위수 지정하기

- **quantiles=0~1사이의 실수리스트**

```
plt.violinplot(scores, showextrema=True, showmeans=True, showmedians=True
               ,quantiles=[0.25, 0.75])
plt.show()
```

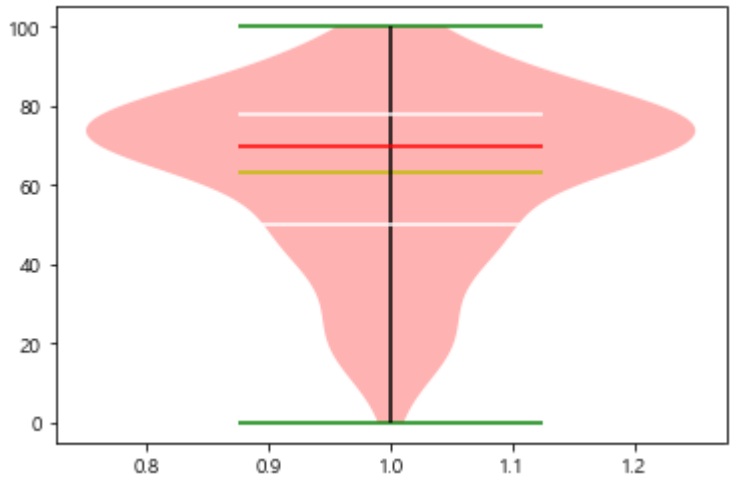


스타일 지정하기

- **바이올린플롯 객체를 받아서 스타일을 지정한다.**
- **플롯['bodies'][인덱스].set_facecolor(컬러)**
- **플롯['cmins'].set_edgecolor(컬러)**
- **플롯['cmaxes'].set_edgecolor(컬러)**
- **플롯['cbars'].set_edgecolor(컬러)**
- **플롯['cmedians'].set_edgecolor(컬러)**
- **플롯['cquantiles'].set_edgecolor(컬러)**

- **플롯['cmeans'].set_edgecolor(컬러)**

```
v1 = plt.violinplot(scores, showextrema=True, showmeans=True, showmedians=True
                ,quantiles=[0.25, 0.75])
v1['bodies'][0].set_facecolor('r')
v1['cmins'].set_edgecolor('g')
v1['cmaxes'].set_edgecolor('g')
v1['cbars'].set_edgecolor('k')
v1['cmedians'].set_edgecolor('r')
v1['cquantiles'].set_edgecolor('w')
v1['cmeans'].set_edgecolor('y')
plt.show()
```



여러 개의 데이터 비교하기

샘플 데이터

- **붓꽃 데이터**

```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

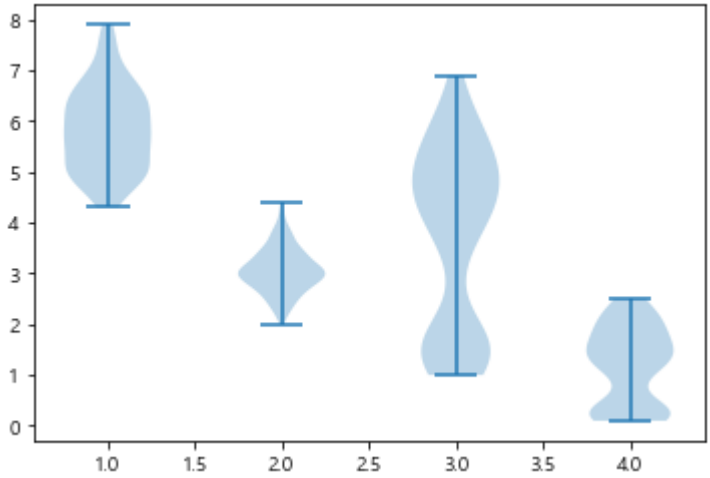
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

바이올린 플롯

기본 그리기

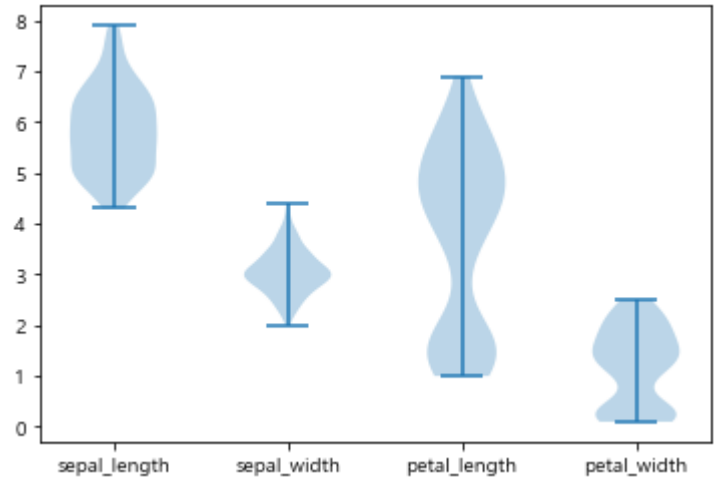
- **plt.violinplot(데이터리스트)**

```
plt.violinplot([iris['sepal_length'], iris['sepal_width'], iris['petal_length'],
iris['petal_width']])
plt.show()
```



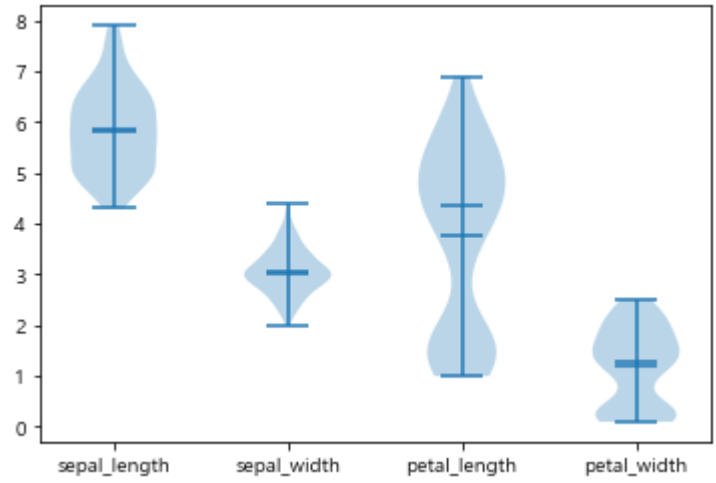
x틱

```
plt.violinplot([iris['sepal_length'], iris['sepal_width'], iris['petal_length'],
iris['petal_width']])
plt.xticks(range(1,5,1), labels=
['sepal_length','sepal_width','petal_length','petal_width'])
plt.show()
```



평균값, 중간값 표시

```
plt.violinplot([iris['sepal_length'], iris['sepal_width'], iris['petal_length'],
iris['petal_width']]
               , showmeans=True, showmedians=True)
plt.xticks(range(1,5,1), labels=
['sepal_length','sepal_width','petal_length','petal_width'])
plt.show()
```

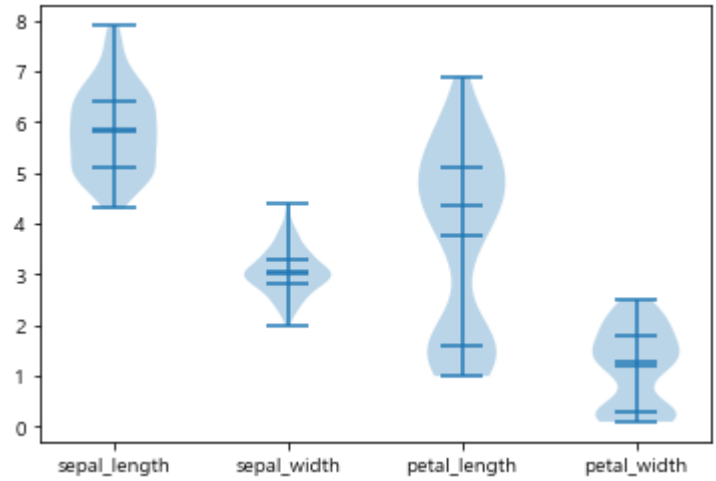


분위수 표시

- **quantiles=분위수리스트**

데이터별로 각각 지정한다.

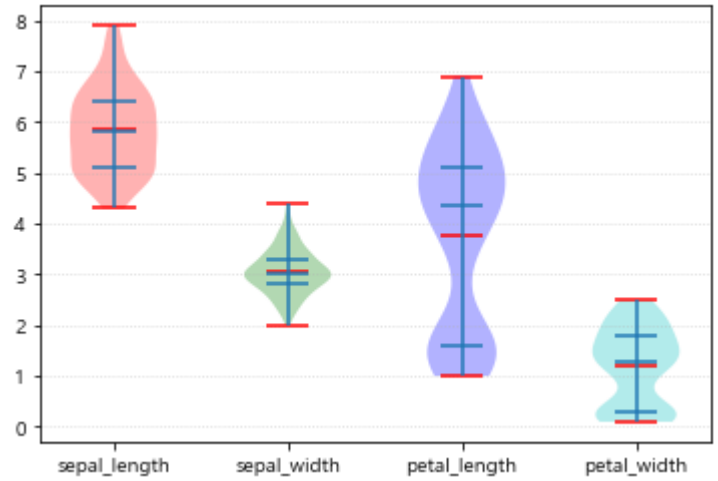
```
plt.violinplot([iris['sepal_length'], iris['sepal_width'], iris['petal_length'],
iris['petal_width']]
               , showmeans=True, showmedians=True
               , quantiles=[[0.25,0.75],[0.25,0.75],[0.25,0.75],[0.25,0.75]])
plt.xticks(range(1,5,1), labels=
['sepal_length','sepal_width','petal_length','petal_width'])
plt.show()
```



스타일 지정하기

- **바이올린플롯 객체를 받아서 스타일을 지정한다.**
- **플롯['bodies'][인덱스].set_facecolor(컬러)**
- **플롯['cmins'].set_edgecolor(컬러)**
- **플롯['cmaxes'].set_edgecolor(컬러)**
- **플롯['cbars'].set_edgecolor(컬러)**
- **플롯['cmedians'].set_edgecolor(컬러)**
- **플롯['cquantiles'].set_edgecolor(컬러)**
- **플롯['cmeans'].set_edgecolor(컬러)**

```
v2 = plt.violinplot([iris['sepal_length'], iris['sepal_width'],
iris['petal_length'], iris['petal_width']]
              , showmeans=True, showmedians=True
              , quantiles=[[0.25,0.75],[0.25,0.75],[0.25,0.75],[0.25,0.75]])

v2['bodies'][0].set_facecolor('r')
v2['bodies'][1].set_facecolor('g')
v2['bodies'][2].set_facecolor('b')
v2['bodies'][3].set_facecolor('c')
v2['cmins'].set_edgecolor('r')
v2['cmaxes'].set_edgecolor('r')
v2['cmeans'].set_edgecolor('r')

plt.xticks(range(1,5,1), labels=
['sepal_length','sepal_width','petal_length','petal_width'])

plt.grid(axis='y', ls=':', alpha=0.5)
plt.show()
```



[학습목표]

파이차트를 이용하여 데이터의 상대적 비율을 표현할 수 있다.

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

```
# 그래프에 한글 설정
plt.rcParams['font.family'] = 'Malgun Gothic'

# 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

파이차트

- **전체에 대한 각 부분의 비율을 부채꼴 모양으로 나타낸 그래프이다.**
- **각 부채꼴의 중심각이 전체에서 해당 데이터가 차지하는 비율을 나타낸다.**
- **1차원 리스트/배열/시리즈를 이용하여 그린다.**

샘플데이터

```
# 2019년 병역판정검사 - 혈액형 분포
blood_type = ['A','B','O','AB']
Personnel = [111901,87066,86804,36495]
```

기본 파이그래프 그리기

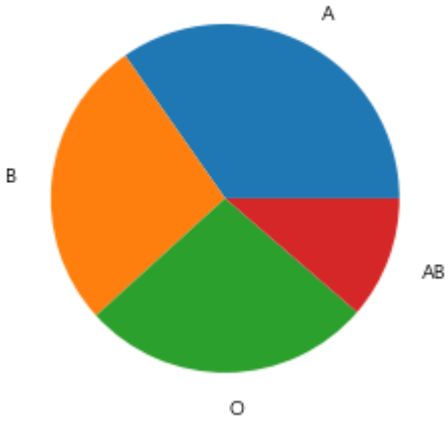
- **plt.pie(data)**

```
plt.pie(Personnel)
plt.show()
```

레이블 달기

- **labels=label목록**
- **labeldistance = 그래프로부터 레이블을 얼만큼 떨어뜨려서 표시할것인가 (default:1.1)**

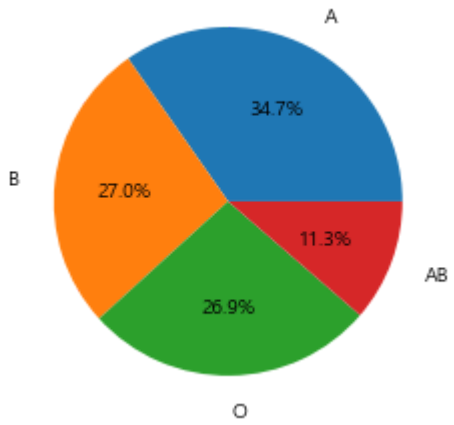
```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2)
plt.show()
```



비율 표시하기

- **autopct = '%소수점자리수%%'**
- **pctdistance = 중심에서의 거리(반지름을 1이라고 했을 때 반지름으로부터 얼만큼 떨어져서 비율을 표시할 것인지.., default:0.6)**

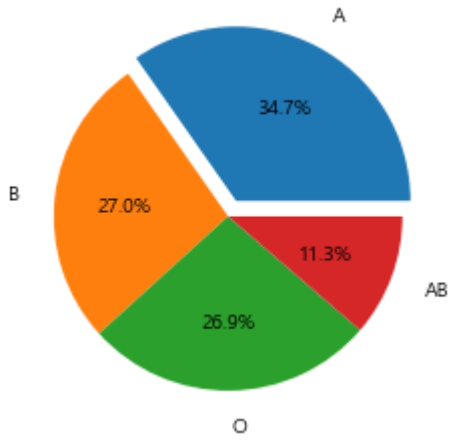
```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6)
plt.show()
```



돌출효과

- **explode=돌출정도리스트**
- **반지름의 길이를 1이라고 했을 때를 기준으로 하여 돌출 정도를 지정**

```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
       , explode=[0.1,0,0,0])
plt.show()
```

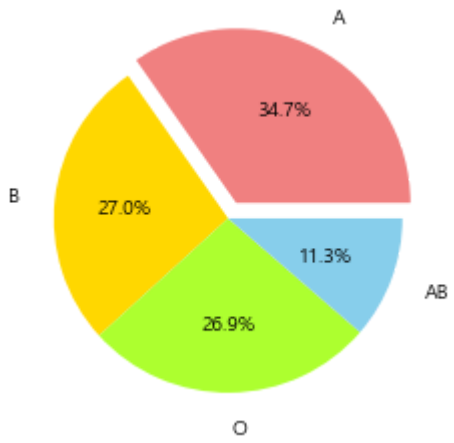


색상 바꾸기

- **colors = 색상리스트**

['lightcoral', 'gold', 'greenyellow', 'skyblue']

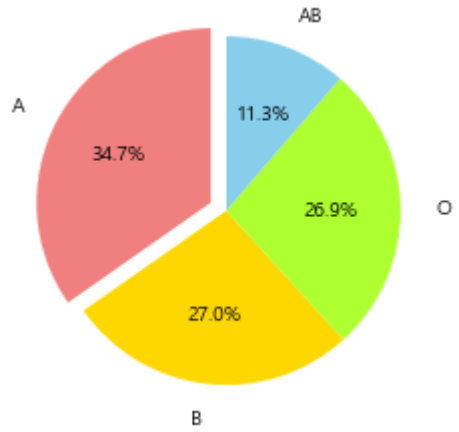
```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
       , explode=[0.1,0,0,0]
       , colors =['lightcoral', 'gold', 'greenyellow', 'skyblue'])
plt.show()
```



시작각도

- **startangle = 시작각도**
- **기본시작각도는 3시방향**
- **시작각도를 지정하면 3시방향으로부터 반 시계방향으로 각도만큼 이동하여 시작**

```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
       , explode=[0.1,0,0,0]
       , colors =['lightcoral', 'gold', 'greenyellow', 'skyblue']
       , startangle=90)
plt.show()
```

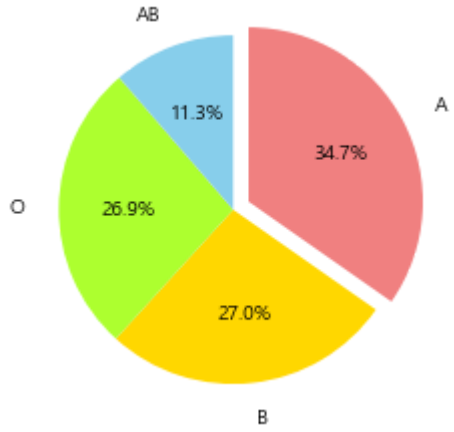


회전방향

- **counterclock=True/False (반시계/시계)**

```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
       , explode=[0.1,0,0,0]
```

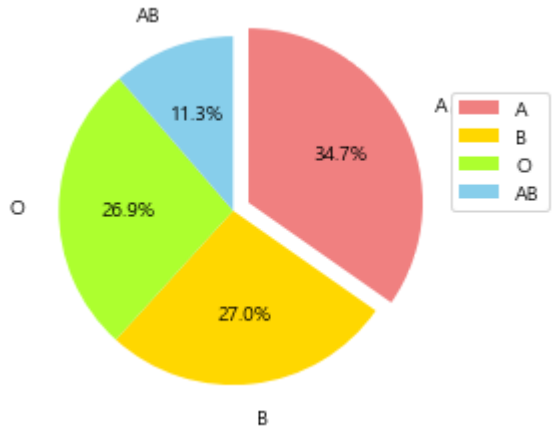
```
        , colors =['lightcoral', 'gold', 'greenyellow', 'skyblue']
        , startangle=90
        , counterclock = False)
plt.show()
```



범례

- **legend(레이블리스트)**

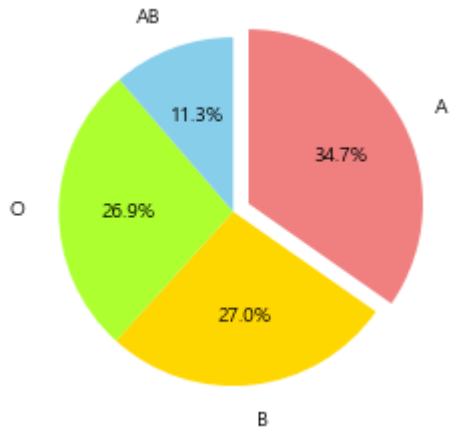
```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
        , explode=[0.1,0,0,0]
        , colors =['lightcoral', 'gold', 'greenyellow', 'skyblue']
        , startangle=90
        , counterclock = False)
plt.legend(loc=(1,0.5))
plt.show()
```



반지름

- **radius=반지름(Default:1)**

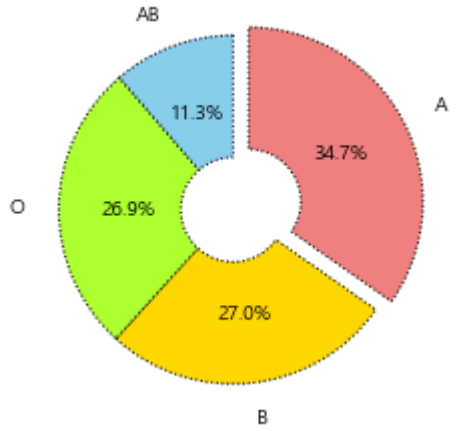
```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
       , explode=[0.1,0,0,0]
       , colors =['lightcoral', 'gold', 'greenyellow', 'skyblue']
       , startangle=90
       , counterclock = False
       , radius=1)
plt.show()
```



부채꼴 스타일

- **wedgeprops = {'ec':테두리컬러, 'lw':선두께, 'ls':선스타일, 'width':반지름에대한비율}**

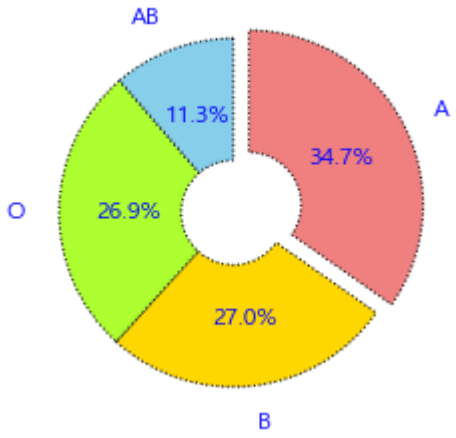
```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
       , explode=[0.1,0,0,0]
       , colors =['lightcoral', 'gold', 'greenyellow', 'skyblue']
       , startangle=90
       , counterclock = False
       , radius=1
       , wedgeprops = {'ec':'k', 'lw':1,'ls':':' ,'width':0.7})
plt.show()
```



폰트

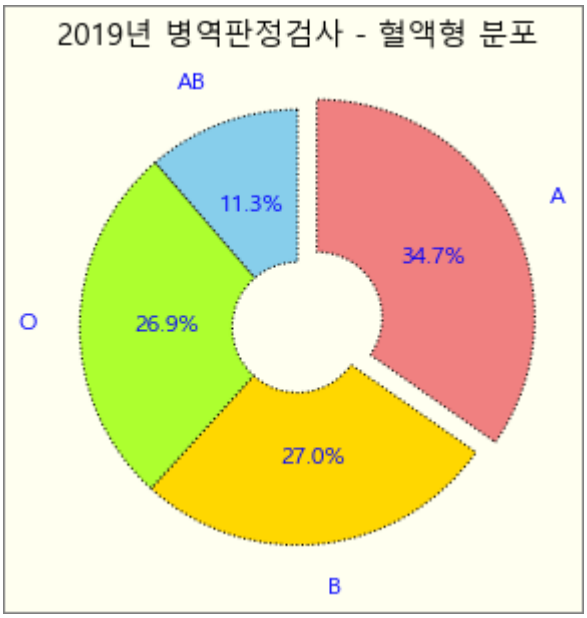
- **textprops = {'fontsize':폰트사이즈, 'color':폰트컬러, 'rotation':폰트회전각도}**

```
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
       , explode=[0.1,0,0,0]
       , colors =['lightcoral', 'gold', 'greenyellow', 'skyblue']
       , startangle=90
       , counterclock = False
       , radius=1
       , wedgeprops = {'ec':'k', 'lw':1,'ls':':' ,'width':0.7}
       , textprops = {'fontsize':12, 'color':'b', 'rotation':0})

plt.show()
```



그래프 완성

```
plt.figure(figsize=(5,5), facecolor='ivory', edgecolor='gray', linewidth=2)
plt.pie(Personnel, labels=blood_type, labeldistance = 1.2, autopct ='%.1f%%',
pctdistance=0.6
       , explode=[0.1,0,0,0]
       , colors =['lightcoral', 'gold', 'greenyellow', 'skyblue']
       , startangle=90
       , counterclock = False
       , radius=1
       , wedgeprops = {'ec':'k', 'lw':1,'ls':':' ,'width':0.7}
       , textprops = {'fontsize':12, 'color':'b', 'rotation':0})

plt.title('2019년 병역판정검사 - 혈액형 분포', size=15)
plt.show()
```



[학습목표]

Matplotlib의 rcParams를 사용해서 그래프의 공통 스타일을 변경 할 수 있다.

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

rcParams

- **rcParams : Runtime Configuration Parameters**
- **그래프를 구성하는 공통 속성을 지정한다.**

```
# 그래프에 한글폰트 설정
plt.rcParams['font.family']='Malgun Gothic'

# 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

rcParams의 종류

```
plt.rcParams
```

```
RcParams({'_internal.classic_mode': False,
          'agg.path.chunksize': 0,
          'animation.avconv_args': [],
          'animation.avconv_path': 'avconv',
```

```
           'animation.bitrate': -1,
           'animation.codec': 'h264',
           'animation.convert_args': [],
           'animation.convert_path': 'convert',
           'animation.embed_limit': 20.0,
           'animation.ffmpeg_args': [],
           'animation.ffmpeg_path': 'ffmpeg',
           'animation.frame_format': 'png',
           'animation.html': 'none',
           'animation.html_args': [],
           'animation.writer': 'ffmpeg',
           'axes.autolimit_mode': 'data',
           'axes.axisbelow': 'line',
           'axes.edgecolor': 'black',
           'axes.facecolor': 'white',
           'axes.formatter.limits': [-5, 6],
           'axes.formatter.min_exponent': 0,
           'axes.formatter.offset_threshold': 4,
           'axes.formatter.use_locale': False,
           'axes.formatter.use_mathtext': False,
           'axes.formatter.useoffset': True,
           'axes.grid': False,
           'axes.grid.axis': 'both',
           'axes.grid.which': 'major',
           'axes.labelcolor': 'black',
           'axes.labelpad': 4.0,
           'axes.labelsize': 'medium',
           'axes.labelweight': 'normal',
           'axes.linewidth': 0.8,
           'axes.prop_cycle': cycler('color', ['#1f77b4', '#ff7f0e',
'#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f',
'#bcbd22', '#17becf']),
           'axes.spines.bottom': True,
           'axes.spines.left': True,
           'axes.spines.right': True,
           'axes.spines.top': True,
           'axes.titlecolor': 'auto',
           'axes.titlelocation': 'center',
           'axes.titlepad': 6.0,
           'axes.titlesize': 'large',
           'axes.titleweight': 'normal',
           'axes.titley': None,
           'axes.unicode_minus': False,
           'axes.xmargin': 0.05,
           'axes.ymargin': 0.05,
           'axes3d.grid': True,
           'backend': 'module://ipykernel.pylab.backend_inline',
           'backend_fallback': True,
           'boxplot.bootstrap': None,
           'boxplot.boxprops.color': 'black',
           'boxplot.boxprops.linestyle': '-',
           'boxplot.boxprops.linewidth': 1.0,
           'boxplot.capprops.color': 'black',
           'boxplot.capprops.linestyle': '-',
```

```
 'boxplot.capprops.linewidth': 1.0,
 'boxplot.flierprops.color': 'black',
 'boxplot.flierprops.linestyle': 'none',
 'boxplot.flierprops.linewidth': 1.0,
 'boxplot.flierprops.marker': 'o',
 'boxplot.flierprops.markeredgecolor': 'black',
 'boxplot.flierprops.markeredgewidth': 1.0,
 'boxplot.flierprops.markerfacecolor': 'none',
 'boxplot.flierprops.markersize': 6.0,
 'boxplot.meanline': False,
 'boxplot.meanprops.color': 'C2',
 'boxplot.meanprops.linestyle': '--',
 'boxplot.meanprops.linewidth': 1.0,
 'boxplot.meanprops.marker': '^',
 'boxplot.meanprops.markeredgecolor': 'C2',
 'boxplot.meanprops.markerfacecolor': 'C2',
 'boxplot.meanprops.markersize': 6.0,
 'boxplot.medianprops.color': 'C1',
 'boxplot.medianprops.linestyle': '-',
 'boxplot.medianprops.linewidth': 1.0,
 'boxplot.notch': False,
 'boxplot.patchartist': False,
 'boxplot.showbox': True,
 'boxplot.showcaps': True,
 'boxplot.showfliers': True,
 'boxplot.showmeans': False,
 'boxplot.vertical': True,
 'boxplot.whiskerprops.color': 'black',
 'boxplot.whiskerprops.linestyle': '-',
 'boxplot.whiskerprops.linewidth': 1.0,
 'boxplot.whiskers': 1.5,
 'contour.corner_mask': True,
 'contour.linewidth': None,
 'contour.negative_linestyle': 'dashed',
 'date.autoformatter.day': '%Y-%m-%d',
 'date.autoformatter.hour': '%m-%d %H',
 'date.autoformatter.microsecond': '%M:%S.%f',
 'date.autoformatter.minute': '%d %H:%M',
 'date.autoformatter.month': '%Y-%m',
 'date.autoformatter.second': '%H:%M:%S',
 'date.autoformatter.year': '%Y',
 'date.epoch': '1970-01-01T00:00:00',
 'docstring.hardcopy': False,
 'errorbar.capsize': 0.0,
 'figure.autolayout': False,
 'figure.constrained_layout.h_pad': 0.04167,
 'figure.constrained_layout.hspace': 0.02,
 'figure.constrained_layout.use': False,
 'figure.constrained_layout.w_pad': 0.04167,
 'figure.constrained_layout.wspace': 0.02,
 'figure.dpi': 72.0,
 'figure.edgecolor': (1, 1, 1, 0),
 'figure.facecolor': (1, 1, 1, 0),
 'figure.figsize': [6.0, 4.0],
```

```
 'figure.frameon': True,
 'figure.max_open_warning': 20,
 'figure.raise_window': True,
 'figure.subplot.bottom': 0.125,
 'figure.subplot.hspace': 0.2,
 'figure.subplot.left': 0.125,
 'figure.subplot.right': 0.9,
 'figure.subplot.top': 0.88,
 'figure.subplot.wspace': 0.2,
 'figure.titlesize': 'large',
 'figure.titleweight': 'normal',
 'font.cursive': ['Apple Chancery',
                  'Textile',
                  'Zapf Chancery',
                  'Sand',
                  'Script MT',
                  'Felipa',
                  'cursive'],
 'font.family': ['Malgun Gothic'],
 'font.fantasy': ['Comic Neue',
                  'Comic Sans MS',
                  'Chicago',
                  'Charcoal',
                  'ImpactWestern',
                  'Humor Sans',
                  'xkcd',
                  'fantasy'],
 'font.monospace': ['DejaVu Sans Mono',
                    'Bitstream Vera Sans Mono',
                    'Computer Modern Typewriter',
                    'Andale Mono',
                    'Nimbus Mono L',
                    'Courier New',
                    'Courier',
                    'Fixed',
                    'Terminal',
                    'monospace'],
 'font.sans-serif': ['DejaVu Sans',
                     'Bitstream Vera Sans',
                     'Computer Modern Sans Serif',
                     'Lucida Grande',
                     'Verdana',
                     'Geneva',
                     'Lucid',
                     'Arial',
                     'Helvetica',
                     'Avant Garde',
                     'sans-serif'],
 'font.serif': ['DejaVu Serif',
                'Bitstream Vera Serif',
                'Computer Modern Roman',
                'New Century Schoolbook',
                'Century Schoolbook L',
                'Utopia',
```

```
                    'ITC Bookman',
                    'Bookman',
                    'Nimbus Roman No9 L',
                    'Times New Roman',
                    'Times',
                    'Palatino',
                    'Charter',
                    'serif'],
 'font.size': 10.0,
 'font.stretch': 'normal',
 'font.style': 'normal',
 'font.variant': 'normal',
 'font.weight': 'normal',
 'grid.alpha': 1.0,
 'grid.color': '#b0b0b0',
 'grid.linestyle': '-',
 'grid.linewidth': 0.8,
 'hatch.color': 'black',
 'hatch.linewidth': 1.0,
 'hist.bins': 10,
 'image.aspect': 'equal',
 'image.cmap': 'viridis',
 'image.composite_image': True,
 'image.interpolation': 'antialiased',
 'image.lut': 256,
 'image.origin': 'upper',
 'image.resample': True,
 'interactive': True,
 'keymap.all_axes': ['a'],
 'keymap.back': ['left', 'c', 'backspace', 'MouseButton.BACK'],
 'keymap.copy': ['ctrl+c', 'cmd+c'],
 'keymap.forward': ['right', 'v', 'MouseButton.FORWARD'],
 'keymap.fullscreen': ['f', 'ctrl+f'],
 'keymap.grid': ['g'],
 'keymap.grid_minor': ['G'],
 'keymap.help': ['f1'],
 'keymap.home': ['h', 'r', 'home'],
 'keymap.pan': ['p'],
 'keymap.quit': ['ctrl+w', 'cmd+w', 'q'],
 'keymap.quit_all': [],
 'keymap.save': ['s', 'ctrl+s'],
 'keymap.xscale': ['k', 'L'],
 'keymap.yscale': ['l'],
 'keymap.zoom': ['o'],
 'legend.borderaxespad': 0.5,
 'legend.borderpad': 0.4,
 'legend.columnspacing': 2.0,
 'legend.edgecolor': '0.8',
 'legend.facecolor': 'inherit',
 'legend.fancybox': True,
 'legend.fontsize': 'medium',
 'legend.framealpha': 0.8,
 'legend.frameon': True,
 'legend.handleheight': 0.7,
```

```
 'legend.handlelength': 2.0,
 'legend.handletextpad': 0.8,
 'legend.labelspacing': 0.5,
 'legend.loc': 'best',
 'legend.markerscale': 1.0,
 'legend.numpoints': 1,
 'legend.scatterpoints': 1,
 'legend.shadow': False,
 'legend.title_fontsize': None,
 'lines.antialiased': True,
 'lines.color': 'C0',
 'lines.dash_capstyle': 'butt',
 'lines.dash_joinstyle': 'round',
 'lines.dashdot_pattern': [6.4, 1.6, 1.0, 1.6],
 'lines.dashed_pattern': [3.7, 1.6],
 'lines.dotted_pattern': [1.0, 1.65],
 'lines.linestyle': '-',
 'lines.linewidth': 1.5,
 'lines.marker': 'None',
 'lines.markeredgecolor': 'auto',
 'lines.markeredgewidth': 1.0,
 'lines.markerfacecolor': 'auto',
 'lines.markersize': 6.0,
 'lines.scale_dashes': True,
 'lines.solid_capstyle': 'projecting',
 'lines.solid_joinstyle': 'round',
 'markers.fillstyle': 'full',
 'mathtext.bf': 'sans:bold',
 'mathtext.cal': 'cursive',
 'mathtext.default': 'it',
 'mathtext.fallback': 'cm',
 'mathtext.fallback_to_cm': None,
 'mathtext.fontset': 'dejavusans',
 'mathtext.it': 'sans:italic',
 'mathtext.rm': 'sans',
 'mathtext.sf': 'sans',
 'mathtext.tt': 'monospace',
 'mpl_toolkits.legacy_colorbar': True,
 'patch.antialiased': True,
 'patch.edgecolor': 'black',
 'patch.facecolor': 'C0',
 'patch.force_edgecolor': False,
 'patch.linewidth': 1.0,
 'path.effects': [],
 'path.simplify': True,
 'path.simplify_threshold': 0.111111111111,
 'path.sketch': None,
 'path.snap': True,
 'pcolor.shading': 'flat',
 'pdf.compression': 6,
 'pdf.fonttype': 3,
 'pdf.inheritcolor': False,
 'pdf.use14corefonts': False,
 'pgf.preamble': '',
```

```
 'pgf.rcfonts': True,
 'pgf.texsystem': 'xelatex',
 'polaraxes.grid': True,
 'ps.distiller.res': 6000,
 'ps.fonttype': 3,
 'ps.papersize': 'letter',
 'ps.useafm': False,
 'ps.usedistiller': None,
 'savefig.bbox': None,
 'savefig.directory': '~',
 'savefig.dpi': 'figure',
 'savefig.edgecolor': 'auto',
 'savefig.facecolor': 'auto',
 'savefig.format': 'png',
 'savefig.jpeg_quality': 95,
 'savefig.orientation': 'portrait',
 'savefig.pad_inches': 0.1,
 'savefig.transparent': False,
 'scatter.edgecolors': 'face',
 'scatter.marker': 'o',
 'svg.fonttype': 'path',
 'svg.hashsalt': None,
 'svg.image_inline': True,
 'text.antialiased': True,
 'text.color': 'black',
 'text.hinting': 'force_autohint',
 'text.hinting_factor': 8,
 'text.kerning_factor': 0,
 'text.latex.preamble': '',
 'text.latex.preview': False,
 'text.usetex': False,
 'timezone': 'UTC',
 'tk.window_focus': False,
 'toolbar': 'toolbar2',
 'webagg.address': '127.0.0.1',
 'webagg.open_in_browser': True,
 'webagg.port': 8988,
 'webagg.port_retries': 50,
 'xaxis.labellocation': 'center',
 'xtick.alignment': 'center',
 'xtick.bottom': True,
 'xtick.color': 'black',
 'xtick.direction': 'out',
 'xtick.labelbottom': True,
 'xtick.labelsize': 'medium',
 'xtick.labeltop': False,
 'xtick.major.bottom': True,
 'xtick.major.pad': 3.5,
 'xtick.major.size': 3.5,
 'xtick.major.top': True,
 'xtick.major.width': 0.8,
 'xtick.minor.bottom': True,
 'xtick.minor.pad': 3.4,
 'xtick.minor.size': 2.0,
```

```
            'xtick.minor.top': True,
            'xtick.minor.visible': False,
            'xtick.minor.width': 0.6,
            'xtick.top': False,
            'yaxis.labellocation': 'center',
            'ytick.alignment': 'center_baseline',
            'ytick.color': 'black',
            'ytick.direction': 'out',
            'ytick.labelleft': True,
            'ytick.labelright': False,
            'ytick.labelsize': 'medium',
            'ytick.left': True,
            'ytick.major.left': True,
            'ytick.major.pad': 3.5,
            'ytick.major.right': True,
            'ytick.major.size': 3.5,
            'ytick.major.width': 0.8,
            'ytick.minor.left': True,
            'ytick.minor.pad': 3.4,
            'ytick.minor.right': True,
            'ytick.minor.size': 2.0,
            'ytick.minor.visible': False,
            'ytick.minor.width': 0.6,
            'ytick.right': False})
```

rcParams 사용하기

figure

```
# figure 크기
plt.rcParams['figure.figsize']=(9,4)
```

axes

```
# 그래프 테두리 두께
plt.rcParams['axes.linewidth'] = 2

# 그래프 테두리 색
plt.rcParams['axes.edgecolor'] = 'navy'

# 그래프 바탕 색
plt.rcParams['axes.facecolor'] = 'ghostwhite'

# 그리드 표시
plt.rcParams['axes.grid'] = True
```

```
# 그래프 제목
plt.rcParams['axes.titlecolor'] = 'navy'
plt.rcParams['axes.titlesize'] = 15
plt.rcParams['axes.titleweight']='bold'
```

```
# 레이블
plt.rcParams['axes.labelcolor'] = 'gray'
plt.rcParams['axes.labelsize'] = 12
```

그리드

```
plt.rcParams['grid.alpha'] = 0.3
plt.rcParams['grid.color'] = 'skyblue'
plt.rcParams['grid.linestyle'] ='--'
```

폰트

```
plt.rcParams['font.size']=12
```

눈금

```
plt.rcParams['xtick.top'] = True
plt.rcParams['ytick.right'] = True
plt.rcParams['xtick.direction'] = 'in'
plt.rcParams['ytick.direction'] = 'in'
plt.rcParams['xtick.major.size'] = 10
plt.rcParams['ytick.major.size'] = 10
```

선

```
plt.rcParams['lines.linewidth']=3
plt.rcParams['lines.linestyle']='--'
plt.rcParams['lines.marker'] ='o'
```

패턴

```
plt.rcParams['hatch.linewidth']=3
plt.rcParams['hatch.color'] = 'w'
```

박스

```
# 박스
plt.rcParams['boxplot.boxprops.color'] = 'b'
plt.rcParams['boxplot.boxprops.linewidth'] = 2

# 최대값, 최소값
plt.rcParams['boxplot.capprops.color'] = 'r'
plt.rcParams['boxplot.capprops.linewidth'] = 2

# 평균
plt.rcParams['boxplot.showmeans'] = True
```

```
plt.rcParams
```

그래프

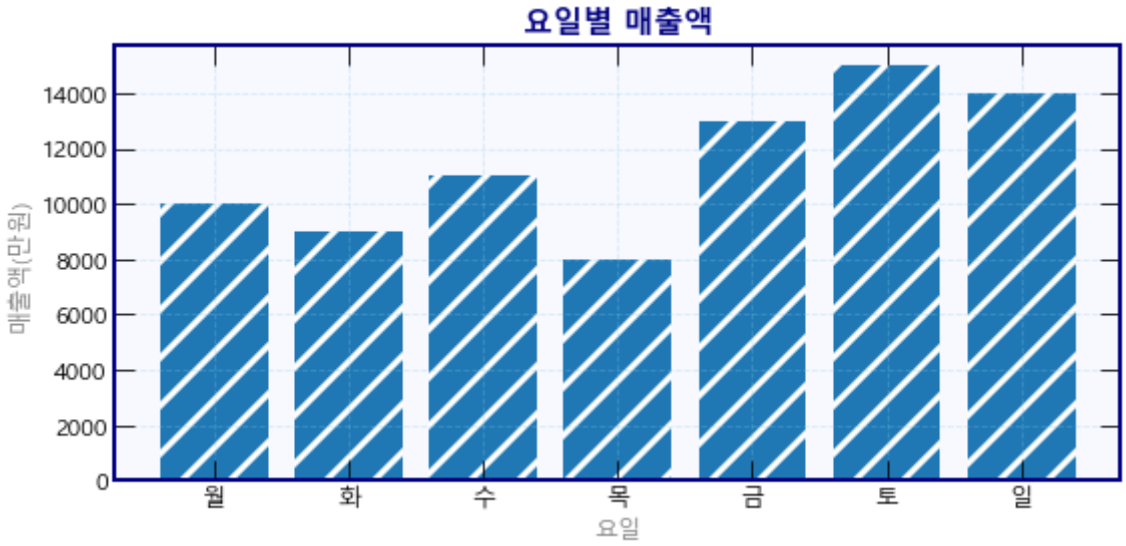
선그래프

```
df1 = pd.DataFrame({'요일':['월','화','수','목','금','토','일'],
                    '매출액':[10000,9000,11000,8000,13000,15000,14000]})
```

```
plt.plot(df1['요일'],df1['매출액'])
plt.title('요일별 매출액')
plt.xlabel('요일')
plt.ylabel('매출액(만원)')
plt.show()
```

막대그래프

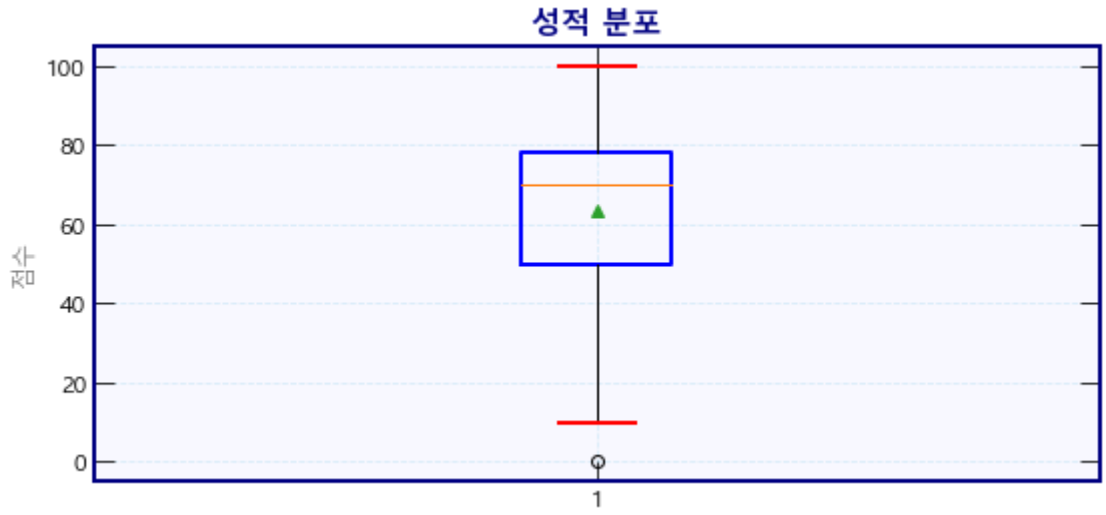
```
plt.bar(df1['요일'],df1['매출액'], hatch='/')
plt.title('요일별 매출액')
plt.xlabel('요일')
plt.ylabel('매출액(만원)')
plt.show()
```



박스플롯

```
scores =
pd.Series([0,10,15,15,15,16,19,20,21,25,25,26,26,29,30,35,36,37,39,40,41,41,44,45,
45,45,45,47,

50,50,50,50,51,51,51,53,54,55,55,56,60,61,62,62,63,64,65,65,65,65,66,66,66,66,66,

67,68,68,69,70,70,70,70,70,70,70,70,71,71,71,71,71,72,72,72,72,73,74,74,74,75,75,

76,76,76,77,77,77,77,78,78,78,78,78,79,79,79,79,80,80,80,80,80,80,81,81,81,82,82,
          85,85,85,88,88,89,90,90,90,93,93,95,95,95,97,100])
```

```
plt.boxplot(scores)
plt.title('성적 분포')
plt.ylabel('점수')
plt.show()
```



[학습목표]

그래프의 영역을 색으로 채워 그래프의 특정 부분을 강조할 수 있다.

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

```
# 그래프에 한글폰트 설정
plt.rcParams['font.family']='Malgun Gothic'

# 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

```
# 그리드 스타일 : 라인스타일'--', 투명도0.3
plt.rcParams['grid.linestyle'] = '--'
plt.rcParams['grid.alpha'] = 0.3

# 폰트 사이즈 12
plt.rcParams['font.size']=12
```

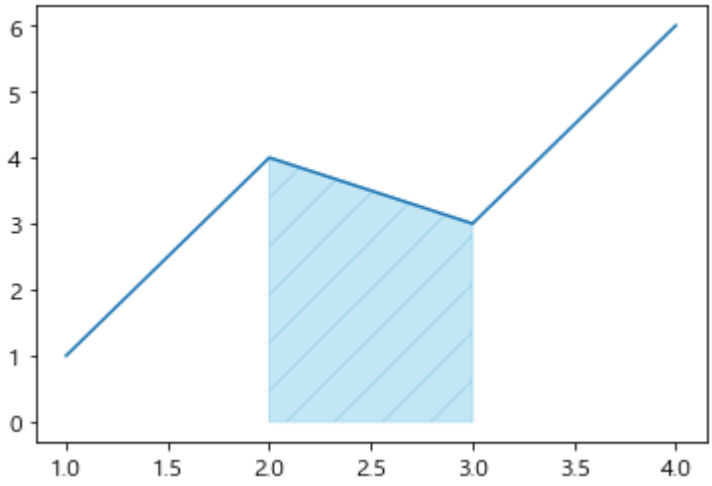
그래프 영역 채우기

가로방향으로 채우기

- `plt.fill_between(x슬라이싱, y슬라이싱)`
- **슬라이싱 범위가 같아야 한다.**

```

```

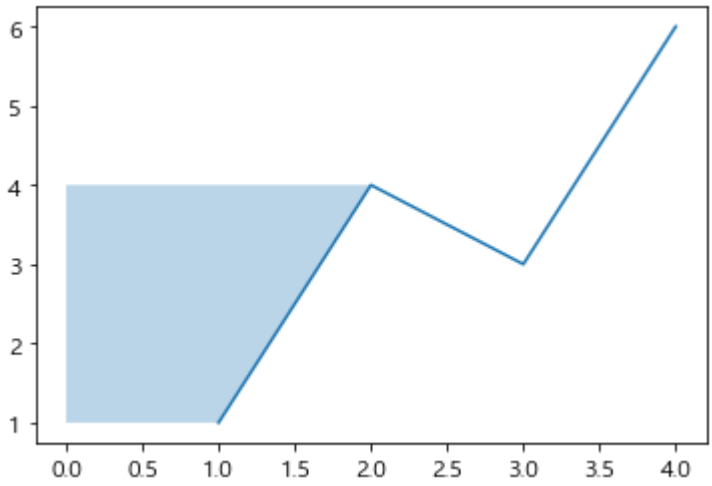
```
x = [1,2,3,4]
y = [1,4,3,6]
plt.plot(x,y)
plt.fill_between(x[1:3], y[1:3], color='skyblue', alpha=0.5, hatch='/')
plt.show()
```



세로방향으로 채우기

- `plt.fill_betweenx(y슬라이싱, x슬라이싱)`
- **슬라이싱 범위가 같아야 한다.**

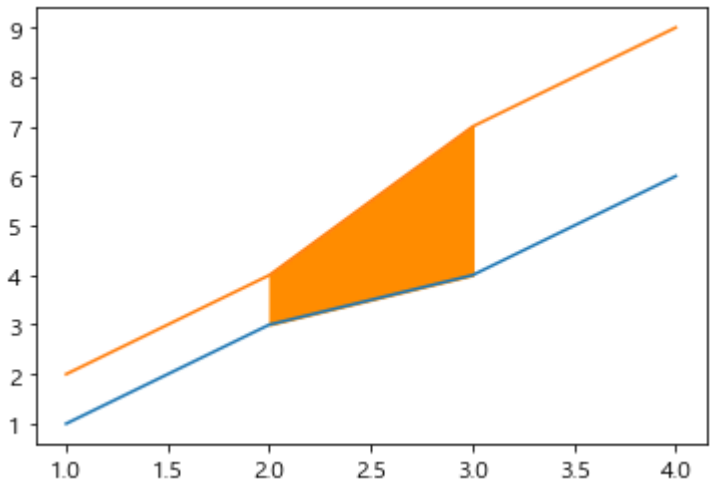
```
x = [1,2,3,4]
y = [1,4,3,6]
plt.plot(x,y)

plt.fill_betweenx(y[:2], x[:2], alpha=0.3)

plt.show()
```



두 그래프 사이의 영역 채우기

- `plt.fill_between(x슬라이싱, y1슬라이싱, y2슬라이싱)`

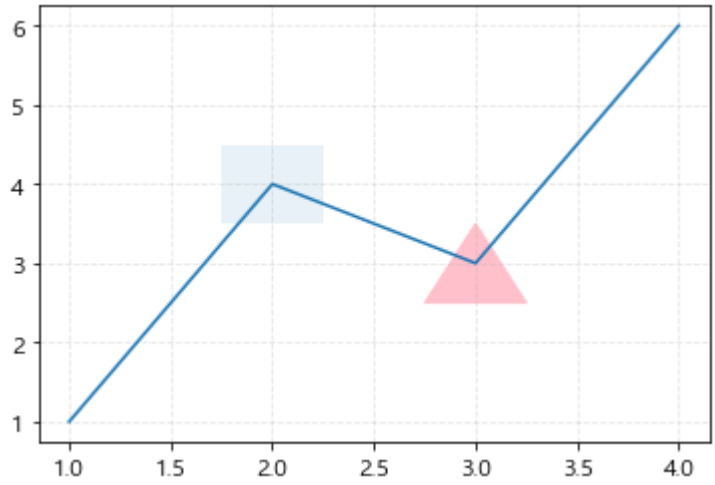
```
x = [1,2,3,4]
y1 = [1,3,4,6]
y2 = [2,4,7,9]

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

plt.fill_between(x[1:3], y1[1:3], y2[1:3], color='darkorange')
plt.show()
```



다각형 채우기

- `plt.fill([x축의 좌표들],[y축의 좌표들])`

```
x = [1,2,3,4]
y = [1,4,3,6]
plt.plot(x,y)
plt.grid()
plt.fill([1.75, 2.25, 2.25, 1.75],[3.5, 3.5, 4.5, 4.5], alpha=0.1)
plt.fill([2.75, 3.25, 3.0],[2.5, 2.5, 3.5], color='pink')
plt.show()
```



[학습목표]

그래프 내에 수직선이나 수평선을 표시하여 그래프의 특정 부분을 강조할 수 있다.

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

```
# 그래프에 한글폰트 설정
plt.rcParams['font.family']='Malgun Gothic'

# 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

```
# 그리드 스타일
plt.rcParams['grid.linestyle'] = '--'
plt.rcParams['grid.alpha'] = 0.3
```

수평선, 수직선 그리기

수평선 그리기

- `plt.axhline(y좌표, x축시작위치, x축끝위치)`

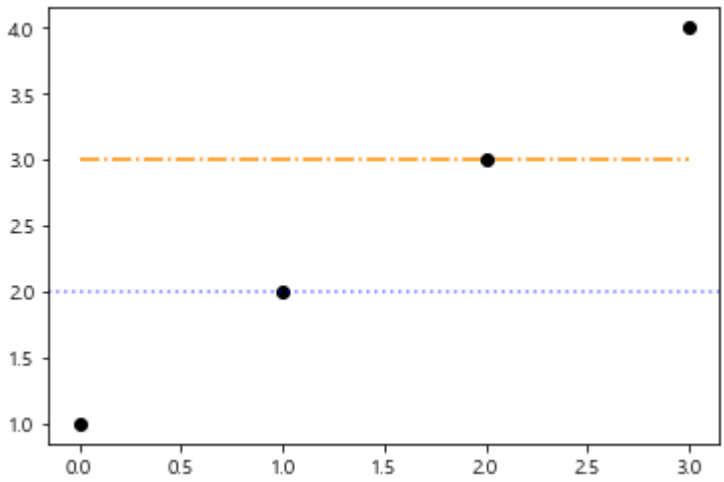
수평선의 길이가 1이라고 했을 때 x축시작위치, x축끝위치를 지정한다.

따로 지정하지 않으면 x축 전범위에 걸쳐 그려진다ㅏ.

- `plt.hlines(y, x축시작좌표, x축끝좌표)`

```

```

```
plt.plot([1,2,3,4], 'ko')

plt.axhline(2,0,1, color='b', alpha=0.5, ls=':')

plt.hlines(3, 0.0, 3.0, color='darkorange', ls='-.')

plt.show()
```



수직선 그리기

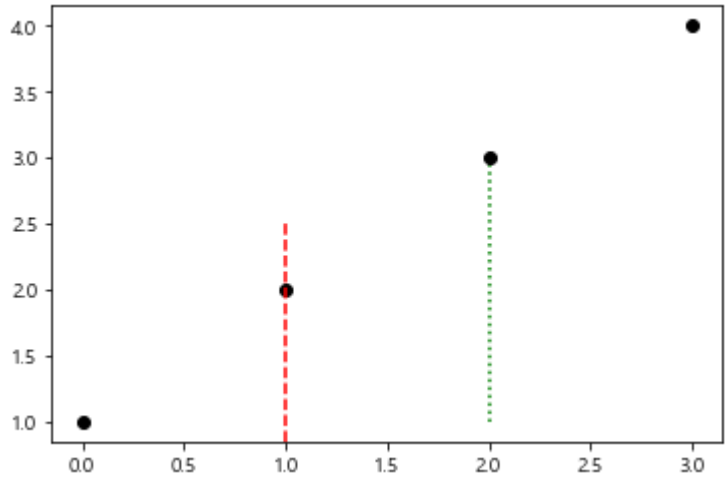
- `axvline(x좌표, y축시작위치, y축끝위치)`

수직선의 길이가 1이라고 했을 때 y축시작위치, y축끝위치를 지정한다.

따로 지정하지 않으면 y축 전범위에 걸쳐 그려진디ㅏ.

- `vlines(x, y축시작좌표, y축끝좌표)`

```
plt.plot([1,2,3,4], 'ko')
plt.axvline(1, 0, 0.5, ls='--', color='r')
plt.vlines(2, 1.0, 3.0, ls=':', color='g')
plt.show()
```



예제 : 요일별 판매 테이블 수

데이터

```
import seaborn as sns
tips = sns.load_dataset('tips')
tips.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	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

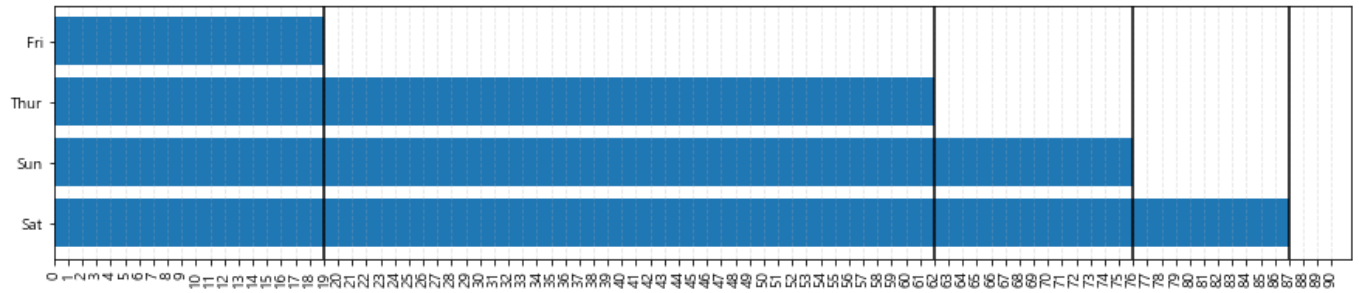
요일 별 테이블 수

- **요일별 데이터 수**

```
s = tips['day'].value_counts()
s
```

```
Sat     87
Sun     76
Thur    62
Fri     19
Name: day, dtype: int64
```

```
plt.figure(figsize=(15,3))
plt.barh(s.index, s.values)
plt.xticks(range(0,91,1), rotation=90)
plt.grid(axis='x')
plt.axvline(s['Fri'], color='k')
plt.axvline(s['Thur'], color='k')
plt.axvline(s['Sun'], color='k')
plt.axvline(s['Sat'], color='k')
plt.show()
```



[학습목표]

그래프의 특정 위치에 설명을 추가할 수 있다.

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

```
# 그래프에 한글폰트 설정
plt.rcParams['font.family']='Malgun Gothic'

# 그래프에 마이너스 기호 깨지는 문제 해결
plt.rcParams['axes.unicode_minus'] = False
```

텍스트 추가하기

- **plt.text(x좌표, y좌표, 텍스트)**
- **rotation=회전각도**
- **ha : horizontal alignment**
- **va : vertical alignment**
- **텍스트 상자**

`bbox = {'boxstyle':상자스타일, 'fc':facecolor,'ec':edgecolor,...}`

boxstyle : 'round'/'square'

```
plt.plot([1,2,3,4], 'ko')
plt.text(2.1, 3, '(x:2, y:3)', ha='left', va='bottom', fontsize=12, rotation=45
        , bbox={'boxstyle':'round', 'fc':'skyblue', 'ec':'b', 'alpha':0.3})
plt.axhline(3, ls=':', alpha=0.5, lw=0.5)
plt.axvline(2, ls=':', alpha=0.5, lw=0.5)
plt.plot(2,3,'ro')
plt.show()
```

화살표와 텍스트 추가하기

- `plt.annotate('텍스트',xy=(화살표x,화살표y), xytext=(텍스트x,텍스트y), arrowprops=화살표속성(딕셔너리))`

- **화살표 속성**

`width` **The width of the arrow in points**

`headwidth` **The width of the base of the arrow head in points**

`headlength` **The length of the arrow head in points**

`shrink` **Fraction of total length to shrink from both ends**

```
plt.plot([1,2,3,4], 'ko')
plt.axhline(2, color='orange', lw=0.5, alpha=0.5, ls='--')
plt.axvline(1, color='orange', lw=0.5, alpha=0.5, ls='--')
plt.plot(1,2,'ro')

plt.annotate('(x:1,y:2)', xy=(1,2), xytext=(1.5,2.5)
             , arrowprops={'width':1, 'headwidth':10, 'headlength':10,
'shrink':0.1, 'fc':'r'}
             , fontsize=12, color='r')
plt.show()
```

[학습목표]

seaborn으로 막대그래프를 그릴 수 있다.

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

```
plt.rcParams['font.family']='Malgun Gothic'
plt.rcParams['axes.unicode_minus']=False
```

샘플데이터

```
tips = sns.load_dataset('tips')
```

```
tips.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	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

```
tips.info()
```

```
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   total_bill  244 non-null    float64 
 1   tip         244 non-null    float64 
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64   
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

요일별 팁 평균

matplotlib으로 시각화

데이터 가공

- **요일별 팁 평균 계산**
- **그룹핑 : 데이터프레임.groupby(그룹기준컬럼)[통계적용컬럼].통계함수**

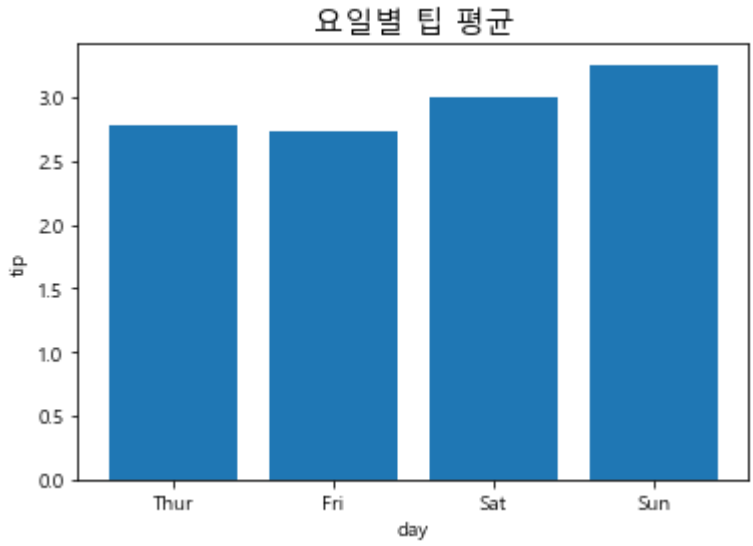
```
day_tip_mean = tips.groupby('day')['tip'].mean()
day_tip_mean
```

```
day
Thur    2.771452
Fri     2.734737
Sat     2.993103
Sun     3.255132
Name: tip, dtype: float64
```

데이터 시각화

- **plt.bar(x,y)**

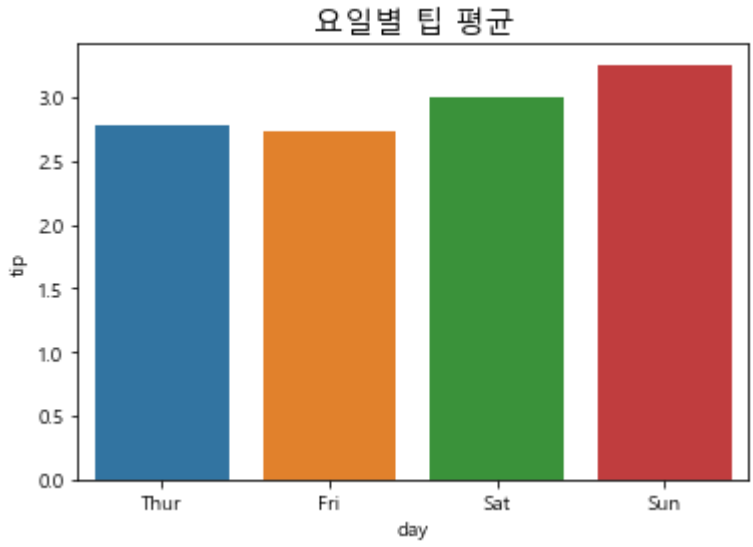
```
plt.bar(day_tip_mean.index, day_tip_mean)
plt.xlabel('day')
plt.ylabel('tip')
plt.title('요일별 팁 평균', size=15)
plt.show()
```



seaborn으로 그리기

- `sns.barplot(data=데이터프레임명, x=x축컬럼, y=y축컬럼)`
- **x축데이터로 그룹핑한 y축데이터의 평균값을 계산하여 그래프를 그려준다.**
- **신뢰구간(CI:Confidence Interval)을 함께 표시**

```
sns.barplot(data=tips, x='day', y='tip', ci=None)
plt.title('요일별 팁 평균', size=15)
plt.show()
```



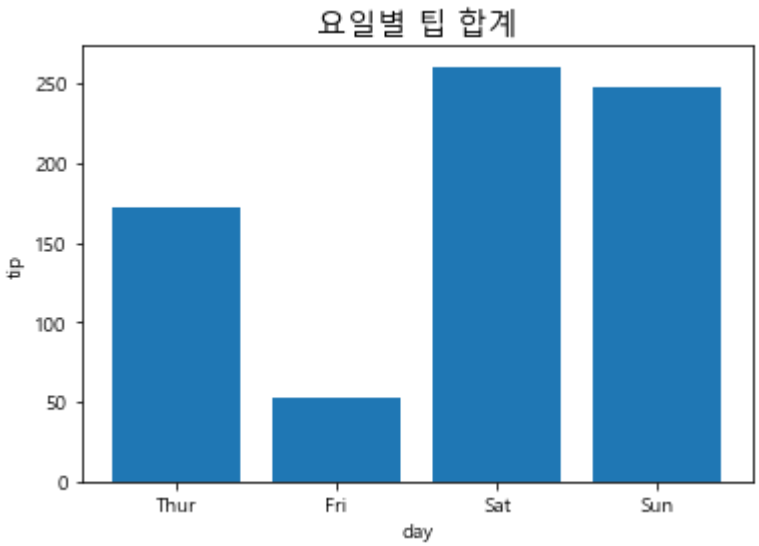
요일별 팁 합계

matplotlib으로 그리기

```
day_tip_sum = tips.groupby('day')['tip'].sum()
day_tip_sum
```

```
day
Thur    171.83
Fri      51.96
Sat     260.40
Sun     247.39
Name: tip, dtype: float64
```

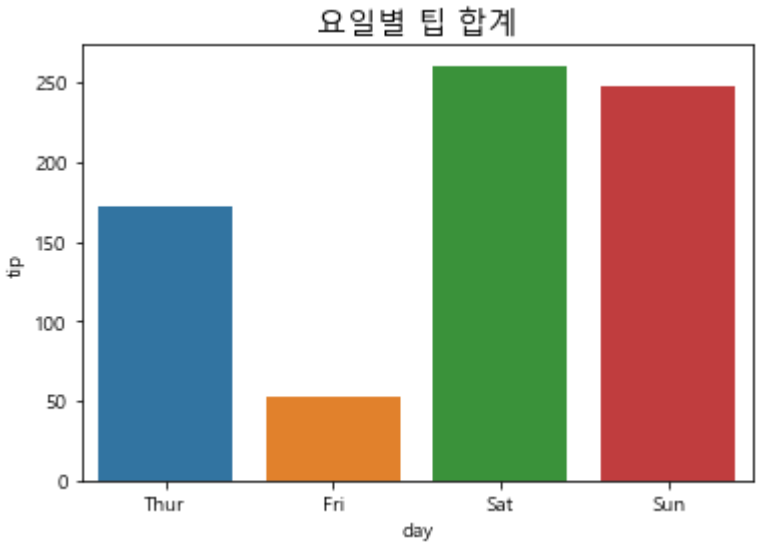
```
plt.bar(day_tip_sum.index, day_tip_sum)
plt.xlabel('day')
plt.ylabel('tip')
plt.title('요일별 팁 합계', size=15)
plt.show()
```



seaborn으로 그리기

- **estimator = 통계함수**

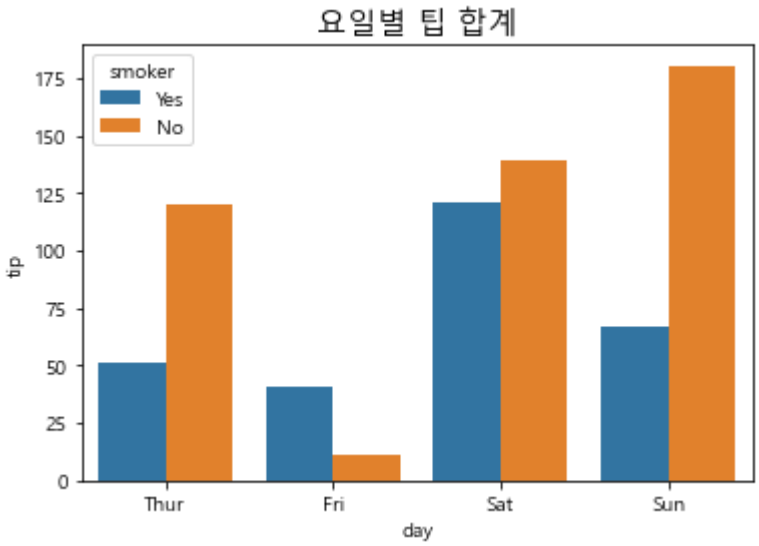
```
sns.barplot(data=tips, x='day', y='tip', ci=None, estimator=sum)
plt.title('요일별 팁 합계', size=15)
plt.show()
```



요일별 팁 합계를 흡연여부로 비교

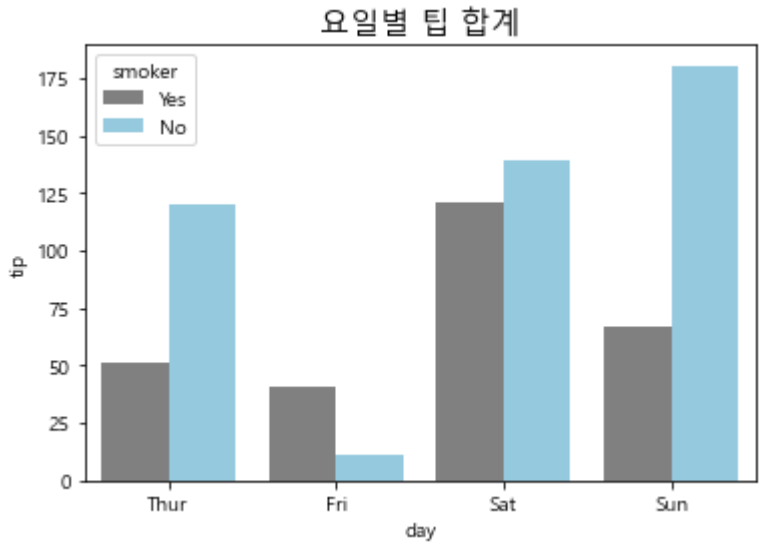
- **hue : y를 그룹핑할 컬럼**

```
sns.barplot(data=tips, x='day', y='tip', ci=None, estimator=sum, hue='smoker')
plt.title('요일별 팁 합계', size=15)
plt.show()
```



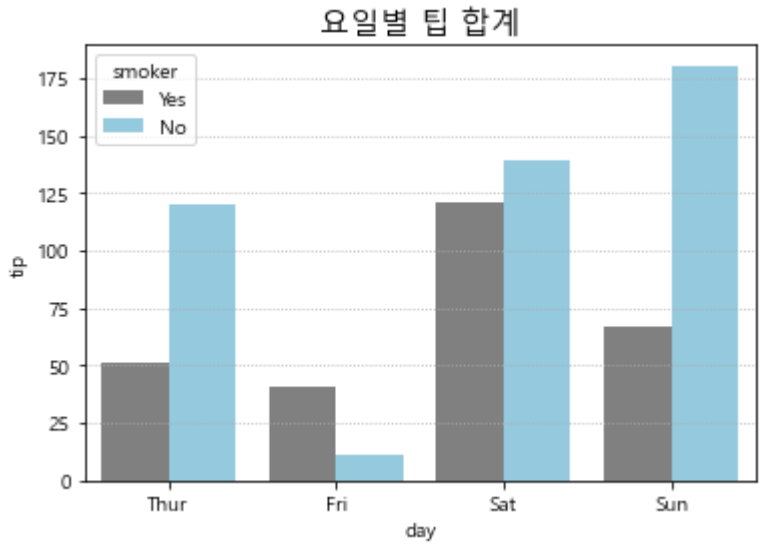
- **hue 색상 변경 : palette = 구분:색상 딕셔너리**

```
sns.barplot(data=tips, x='day', y='tip', ci=None, estimator=sum, hue='smoker'
            , palette ={'Yes':'gray', 'No':'skyblue'})
plt.title('요일별 팁 합계', size=15)
plt.show()
```



- **pyplot의 메소드로 그리드 추가**

```
sns.barplot(data=tips, x='day', y='tip', ci=None, estimator=sum, hue='smoker'
            , palette ={'Yes':'gray', 'No':'skyblue'})
plt.title('요일별 팁 합계', size=15)
plt.grid(axis='y', ls=':')
plt.show()
```



[학습목표]

seaborn으로 산점도를 그릴 수 있다.

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

```
plt.rcParams['font.family']='Malgun Gothic'
plt.rcParams['axes.unicode_minus']='False'
```

샘플데이터

```
tips = sns.load_dataset('tips')
```

```
tips.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

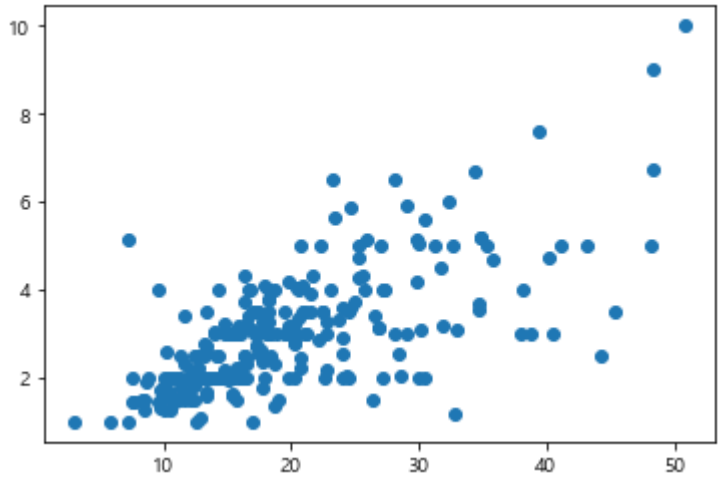
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

total_bill과 tip의 관계

멧플롯립으로 그리기

plt.scatter(x,y)

```
plt.scatter(tips['total_bill'],tips['tip'])
plt.show()
```



요일 구분

- **total_bill에 따른 tip의 분포 - 색상으로 요일 표시**

'Sun':'red'

'Sat':'blue'

'Thur':'green'

'Fri':'yello'

```
tips['day'].unique()
```

```
['Sun', 'Sat', 'Thur', 'Fri']
Categories (4, object): ['Sun', 'Sat', 'Thur', 'Fri']
```

요일별 서브셋 만들기

```
tips_Sun = tips[tips['day']=='Sun']
```

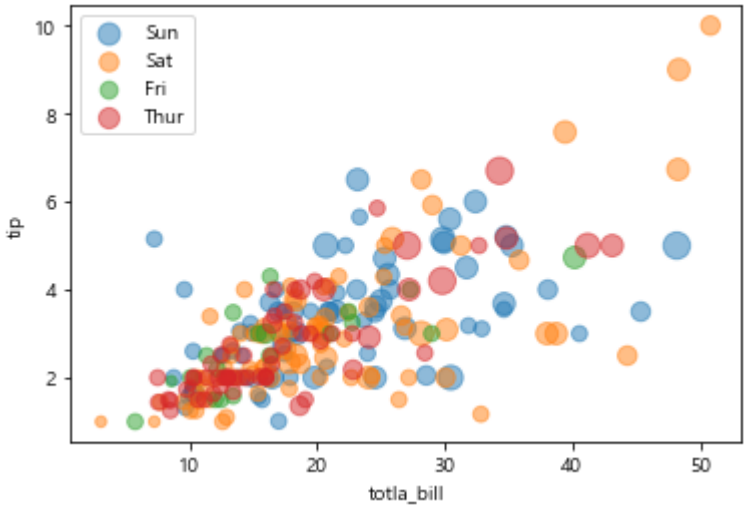
```
tips_Sat = tips[tips['day']=='Sat']
```

```
tips_Fri = tips[tips['day']=='Fri']
```

```
tips_Thur = tips[tips['day']=='Thur']
```

시각화하기

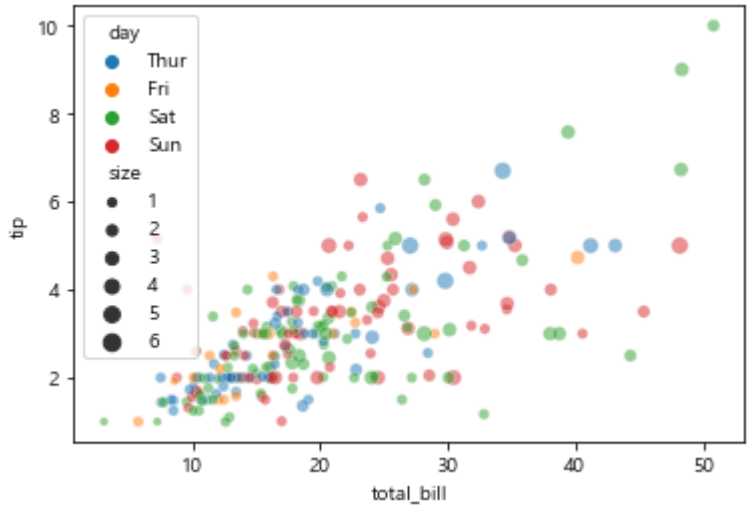
```
plt.scatter(tips_Sun['total_bill'], tips_Sun['tip'], label='Sun',
s=tips_Sun['size']*30, alpha=0.5)
plt.scatter(tips_Sat['total_bill'], tips_Sat['tip'], label='Sat',
s=tips_Sat['size']*30, alpha=0.5)
plt.scatter(tips_Fri['total_bill'], tips_Fri['tip'], label='Fri',
s=tips_Fri['size']*30, alpha=0.5)
plt.scatter(tips_Thur['total_bill'], tips_Thur['tip'], label='Thur',
s=tips_Thur['size']*30, alpha=0.5)
plt.legend()
plt.xlabel('totla_bill')
plt.ylabel('tip')

plt.show()
```



seaborn으로 그리기

- `sns.scatterplot(data=데이터프레임, x=x축컬럼, y=y축컬럼)`

```
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='day', size='size',
alpha=0.5)
plt.show()
```



[학습목표]

Seaborn으로 선그래프를 그릴 수 있다.

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

```
plt.rcParams['font.family']='Malgun Gothic'
plt.rcParams['axes.unicode_minus']=False
```

샘플데이터

```
flights = sns.load_dataset('flights')
flights.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	year	month	passengers
0	1949	Jan	112

	year	month	passengers
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121

```
flights.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	year	month	passengers
139	1960	Aug	606
140	1960	Sep	508
141	1960	Oct	461
142	1960	Nov	390
143	1960	Dec	432

```
flights.shape
```

```
(144, 3)
```

```
# 연도별 데이터 수
flights['year'].value_counts()
```

```
1949    12
1950    12
```

```
1951    12
1952    12
1953    12
1954    12
1955    12
1956    12
1957    12
1958    12
1959    12
1960    12
Name: year, dtype: int64
```

```
# 월별 데이터 수
flights['month'].value_counts()
```

```
Jan    12
Feb    12
Mar    12
Apr    12
May    12
Jun    12
Jul    12
Aug    12
Sep    12
Oct    12
Nov    12
Dec    12
Name: month, dtype: int64
```

연도별 승객수의 변화

matplotlib으로 시각화

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

데이터 가공

```
flights.head(1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

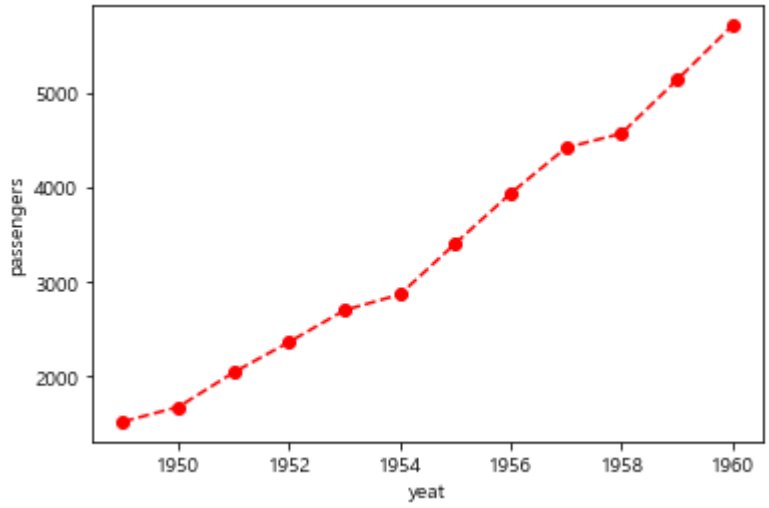
	year	month	passengers
0	1949	Jan	112

```
flights_year = flights.groupby('year')['passengers'].sum()
flights_year
```

```
year
1949    1520
1950    1676
1951    2042
1952    2364
1953    2700
1954    2867
1955    3408
1956    3939
1957    4421
1958    4572
1959    5140
1960    5714
Name: passengers, dtype: int64
```

시각화

```
plt.plot(flights_year, 'ro--')
plt.xlabel('yeat')
plt.ylabel('passengers')
plt.show()
```



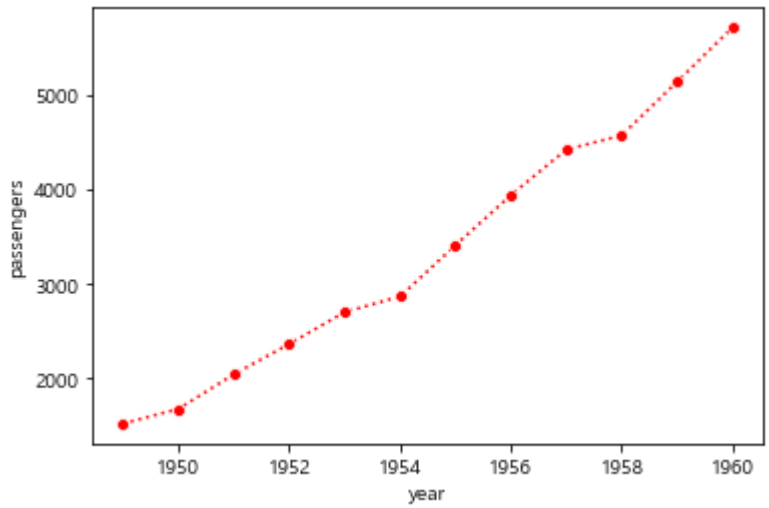
seaborn

- `sns.lineplot(data=데이터프레임, x=x축컬럼, y=y축컬럼, estimator=통계함수)`

estimator를 생략하면 평균으로 통계를 적용함

- **전체 데이터로 차트를 그리면 신뢰구간 표시**

```
sns.lineplot(data=flights, x='year', y='passengers', ci=None, estimator=sum,
color='r', marker='o', ls=':')
plt.show()
```



연도-월별 승객수의 변화

matplotlib으로 시각화

데이터 가공

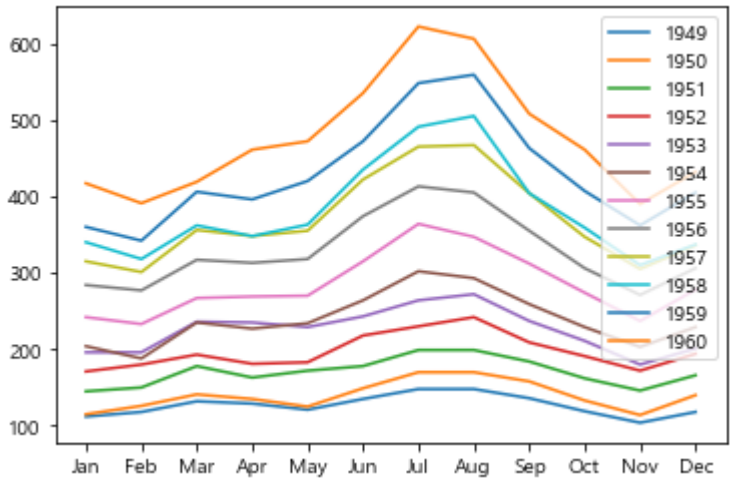
```
flights_pivot = flights.pivot(index='year', columns='month', values='passengers')
flights_pivot
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
year												
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
1951	145	150	178	163	172	178	199	199	184	162	146	166
1952	171	180	193	181	183	218	230	242	209	191	172	194
1953	196	196	236	235	229	243	264	272	237	211	180	201
1954	204	188	235	227	234	264	302	293	259	229	203	229

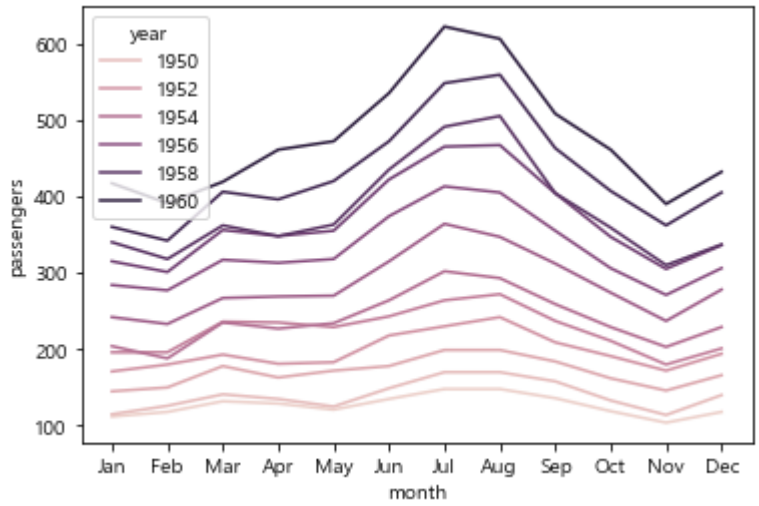
시각화

```
for i in range(12):
    plt.plot(flights_pivot.iloc[i], label=flights_pivot.index[i])
plt.legend()
plt.show()
```



seaborn으로 시각화

```
sns.lineplot(data=flights, x='month', y='passengers', ci=None, hue='year')
```



[학습목표]

seaborn으로 데이터의 분포를 나타내는 여러가지 그래프를 그릴 수 있다.

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

```
plt.rcParams['font.family']='Malgun Gothic'
plt.rcParams['axes.unicode_minus']=False
```

카운트플롯

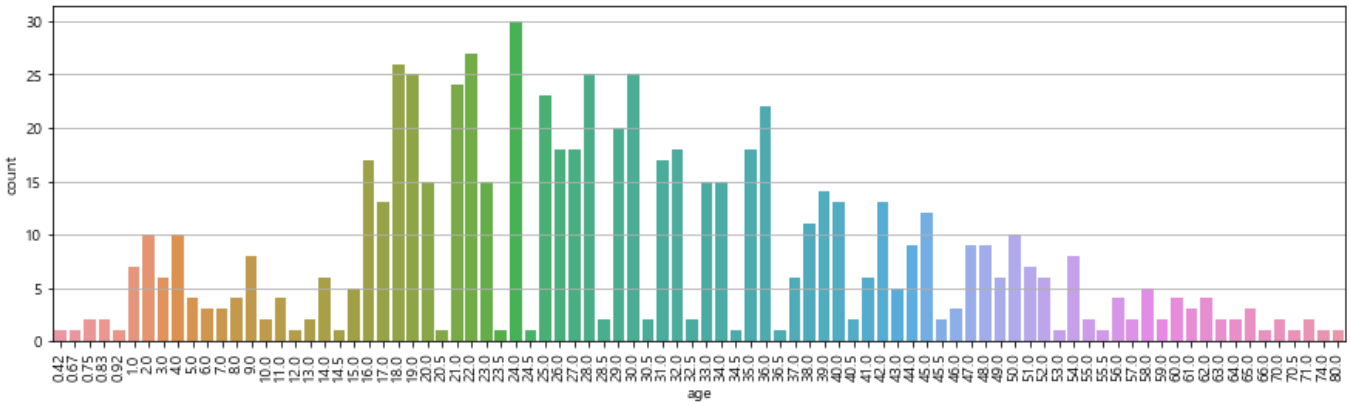
- **데이터의 갯수를 카운트하여 시각화**
- `sns.countplot(data=데이터프레임, x=컬럼)`

```
titanic = sns.load_dataset('titanic')
titanic.head()
```

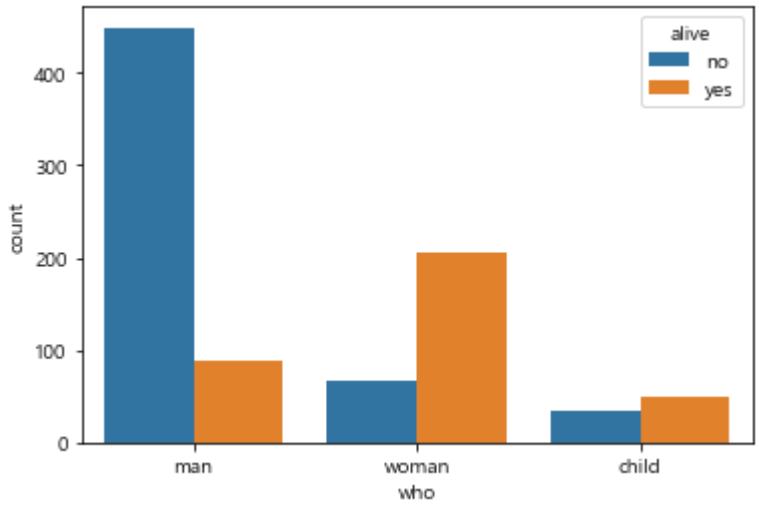
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who
0	0	3	male	22.0	1	0	7.2500	S	Third	mai
1	1	1	female	38.0	1	0	71.2833	C	First	wom
2	1	3	female	26.0	0	0	7.9250	S	Third	wom
3	1	1	female	35.0	1	0	53.1000	S	First	wom
4	0	3	male	35.0	0	0	8.0500	S	Third	mai

```
# 연령별 승선인원 카운트하여 시각화
plt.figure(figsize=(15,4))
sns.countplot(data=titanic, x='age')
plt.xticks(rotation=90)
plt.grid(axis='y')
plt.show()
```



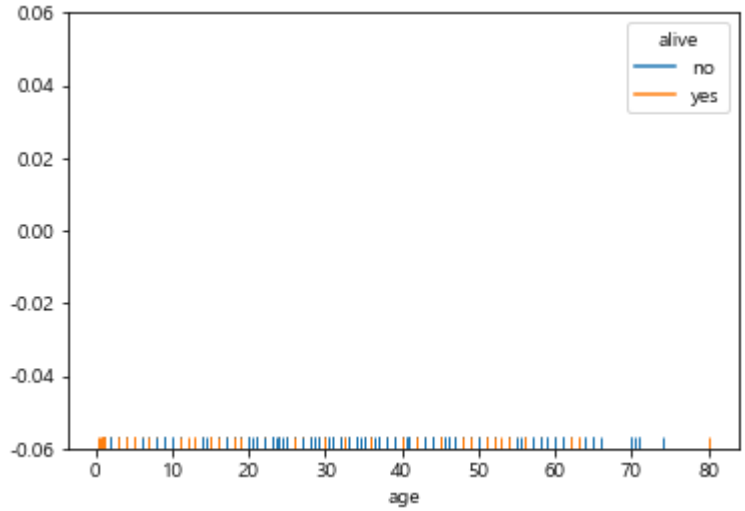

```
# 1. 남자/여자/어린이 승선인원 시각화
# 2. 남자/여자/어린이 별 생존여부

sns.countplot(data=titanic, x='who', hue='alive')
```



러그플롯

- `sns.rugplot(data=데이터프레임, x=컬럼)`

```
sns.rugplot(data=titanic,x='age', hue='alive')
```



히스토그램

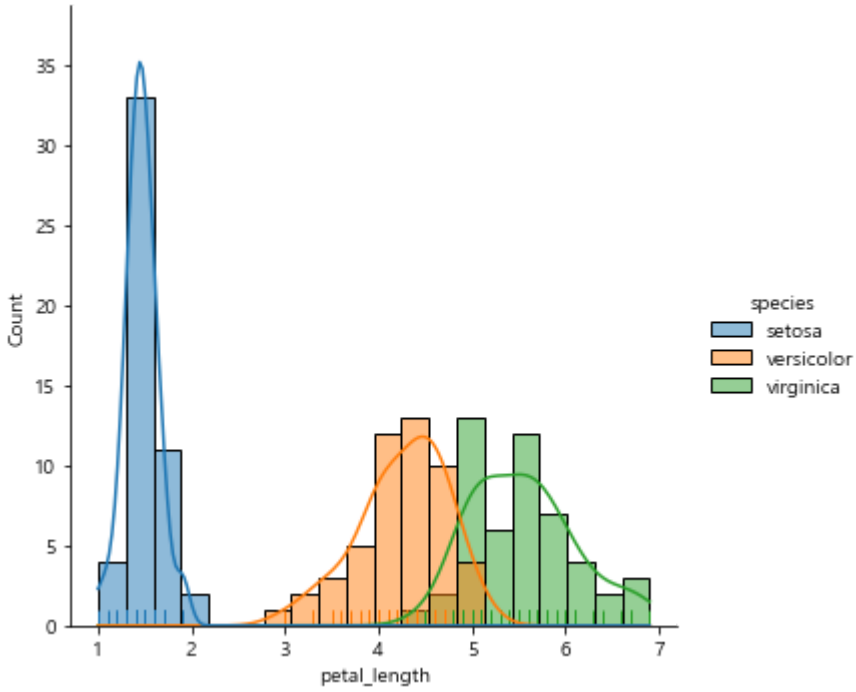
- `sns.displot(data=데이터프레임, x=컬럼)`

```
iris = sns.load_dataset('iris')
iris.head(1)
```

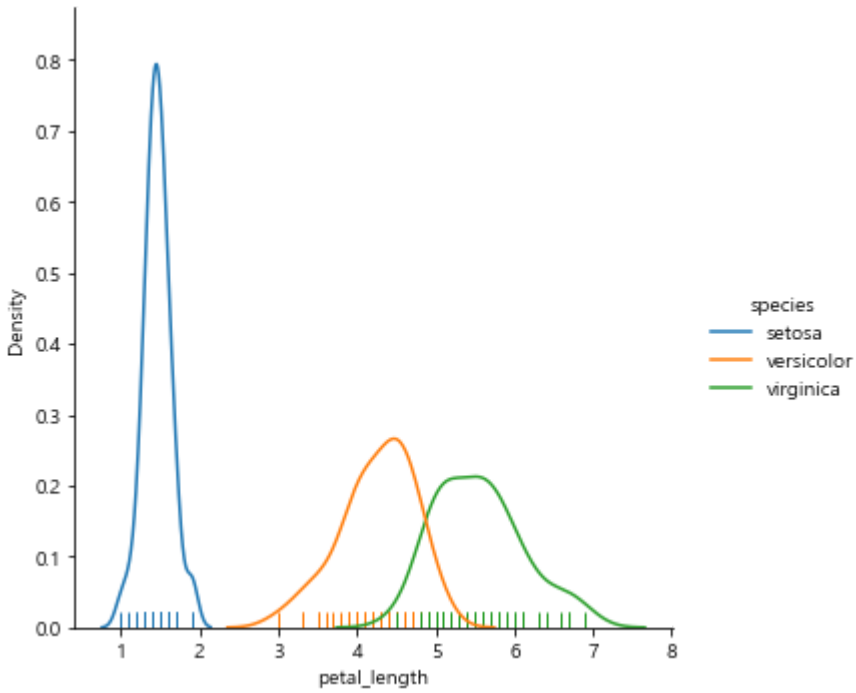
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa

```
sns.displot(data=iris, x='petal_length', bins=20, rug=True, hue='species',
kde=True)
```




```
sns.displot(data=iris, x='petal_length', rug=True, hue='species', kind='kde')
```

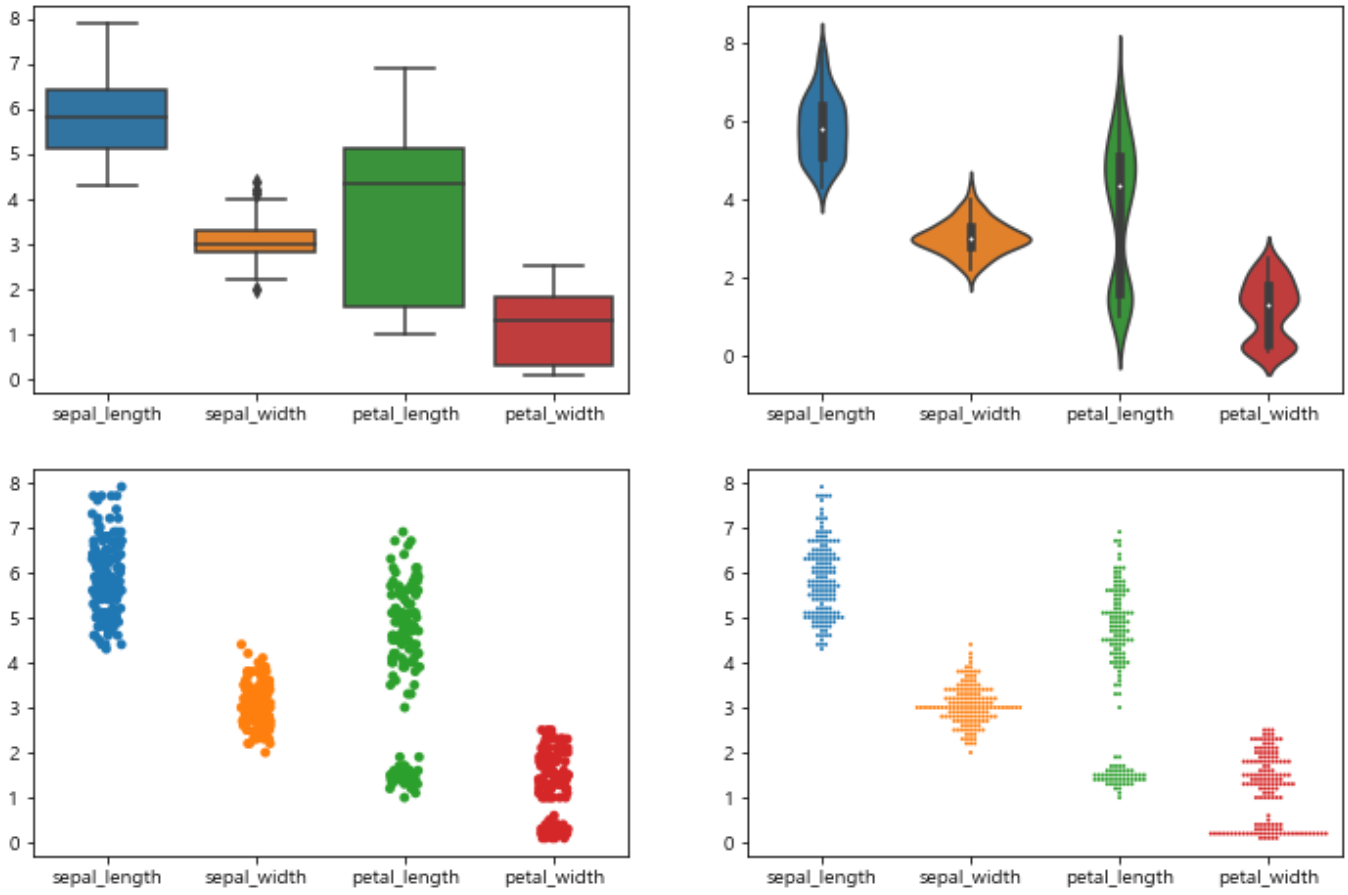


상자수염그래프, 바이올린플롯, 스트립플롯, 스웜플롯

- **sns.boxplot(data=데이터프레임)**

- **sns.violinplot(data=데이터프레임)**
- **sns.stripplot(data=데이터프레임)**
- **sns.swarmplot(data=데이터프레임)**

```
plt.figure(figsize=(12,8))
plt.subplot(221)
sns.boxplot(data=iris)

plt.subplot(222)
sns.violinplot(data=iris)

plt.subplot(223)
sns.stripplot(data=iris)

plt.subplot(224)
sns.swarmplot(data=iris, s=2)

plt.show()
```




```
plt.figure(figsize=(12,8))
plt.subplot(221)
# 박스플롯
```

```
sns.boxplot(data=iris, x='species', y='petal_length')


plt.subplot(222)
# 바이올린플롯
sns.violinplot(data=iris, x='species', y='petal_length')

plt.subplot(223)
# 스트립플롯
sns.stripplot(data=iris, x='species', y='petal_length')

plt.subplot(224)
# 스웜플롯
sns.swarmplot(data=iris, x='species', y='petal_length',s=2)

plt.show()
```