이산형 확률변수
이산형 확률분포

```
In [1]: 1 + 1
Out[1]: 2

In [2]: 5 - 2
Out[2]: 3

In [3]: 2 * 3
Out[3]: 6

In [4]: 2 ** 3
Out[4]: 8

In [5]: 6 / 3
Out[5]: 2.0

In [6]: 7 // 3
Out[6]: 2

In [7]: # 1 + 1

In [8]: "A"
Out[8]: 'A'

In [9]: 'A'
Out[9]: 'A'

In [10]: type("A")
Out[10]: str

In [11]: type('A')
Out[11]: str

In [12]: # 정수형
         type(1)
Out[12]: int

In [13]: # 부동소수점
         type(2.4)
Out[13]: float

In [14]: # 부울형
         type(True)
Out[14]: bool

In [15]: # 부울형
         type(False)
Out[15]: bool
```

```
In [16]: "A" + 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-8c1ba8ce860b> in <module>
----> 1 "A" + 1

TypeError: can only concatenate str (not "int") to str
```

```
In [17]: 1 > 0.89
Out[17]: True

In [18]: 3 >= 2
Out[18]: True

In [19]: 3 < 2
Out[19]: False

In [20]: 3 <= 2
Out[20]: False

In [21]: 3 == 2
Out[21]: False

In [22]: 3 != 2
Out[22]: True
```

```
In [23]: x = 2
         x + 1
Out[23]: 3

In [24]: (x + 2) * 4
Out[24]: 16

In [25]: def sample_function(data):
             return((data + 2) * 4)

In [26]: sample_function(x)
Out[26]: 16

In [27]: sample_function(3)
Out[27]: 20

In [28]: sample_function(x) + sample_function(3)
Out[28]: 36
```

In [29]:
```
class Sample_Class:
    def __init__(self, data1, data2):
        self.data1 = data1
        self.data2 = data2

    def method2(self):
        return(self.data1 + self.data2)
```

In [30]:
```
sample_instance = Sample_Class(data1 = 2, data2 = 3)
```

In [31]:
```
sample_instance.data1
```

Out[31]: 2

In [32]:
```
sample_instance.method2()
```

Out[32]: 5

In [33]:
```
data = 1
if(data < 2):
    print("2보다 작은 데이터입니다")
else:
    print("2 이상인 데이터입니다")
```

2보다 작은 데이터입니다

In [34]:
```
data = 3
if(data < 2):
    print("2보다 작은 데이터입니다")
else:
    print("2 이상인 데이터입니다")
```

2 이상인 데이터입니다

In [35]:
```
range(0, 3)
```

Out[35]: range(0, 3)

In [36]:
```
for i in range(0, 3):
    print(i)
```

```
0
1
2
```

In [37]:
```
for i in range(0, 3):
    print("hello")
```

```
hello
hello
hello
```

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
sample_list = [1,2,3,4,5]
sample_list
```

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

In [3]:
```
sample_array = np.array([1,2,3,4,5])
sample_array
```

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

In [4]:
```
sample_array + 2
```

Out[4]:
```
array([3, 4, 5, 6, 7])
```

In [5]:
```
sample_array * 2
```

Out[5]:
```
array([ 2,  4,  6,  8, 10])
```

In [6]:
```
np.array([1 ,2, "A"])
```

Out[6]:
```
array(['1', '2', 'A'], dtype='<U11')
```

In [7]:
```
# 행렬
sample_array_2 = np.array(
    [[1,2,3,4,5],
     [6,7,8,9,10]])
sample_array_2
```

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

In [8]:
```
# 행수·열수의 확인
sample_array_2.shape
```

Out[8]:
```
(2, 5)
```

In [9]:
```
np.arange(start = 1, stop = 6, step = 1)
```

Out[9]:
```
array([1, 2, 3, 4, 5])
```

In [10]:
```
np.arange(start = 0.1, stop = 0.8, step = 0.2)
```

Out[10]:
```
array([0.1, 0.3, 0.5, 0.7])
```

In [11]:
```
np.arange(0.1, 0.8, 0.2)
```

Out[11]:
```
array([0.1, 0.3, 0.5, 0.7])
```

In [12]:
```
# 동일한 값의 반복
np.tile("A", 5)
```

Out[12]:
```
array(['A', 'A', 'A', 'A', 'A'], dtype='<U1')
```

In [13]:
```
# 0을 4번 반복
np.tile(0, 4)
```

Out[13]:
```
array([0, 0, 0, 0])
```

In [14]:
```
# 00| 4개인 배열
np.zeros(4)
```

Out[14]:
```
array([0., 0., 0., 0.])
```

In [15]:
```
# 2차원 배열
np.zeros([2,3])
```

Out[15]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [16]:
```
# 1로만 된 배열
np.ones(3)
```

Out[16]:
```
array([1., 1., 1.])
```

In [17]:
```
# 1차원 배열
d1_array = np.array([1,2,3,4,5])
d1_array
```

Out[17]:
```
array([1, 2, 3, 4, 5])
```

In [18]:
```
# 첫 번째 요소를 취득
d1_array[0]
```

Out[18]:
```
1
```

In [19]:
```
# 2~3번 째 요소를 취득
d1_array[1:3]
```

Out[19]:
```
array([2, 3])
```

In [20]:
```
# 2차원 배열
d2_array = np.array(
    [[1,2,3,4,5],
    [6,7,8,9,10]])
d2_array
```

Out[20]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [21]:
```
d2_array[0, 3]
```

Out[21]:
```
4
```

In [22]:
```
d2_array[1, 2:4]
```

Out[22]:
```
array([8, 9])
```

In [23]:
```
sample_df = pd.DataFrame({
    'col1' : sample_array,
    'col2' : sample_array * 2,
    'col3' : ["A", "B", "C", "D", "E"]
})
print(sample_df)
```

```
   col1  col2 col3
0     1     2    A
1     2     4    B
2     3     6    C
3     4     8    D
4     5    10    E
```

In [24]:
```
sample_df
```

Out[24]:

	col1	col2	col3
0	1	2	A
1	2	4	B
2	3	6	C
3	4	8	D
4	5	10	E

In [25]:
```
file_data = pd.read_csv("2-4-1-sample_data.csv")
print(file_data)
```

```
   col1 col2
0     1    A
1     2    A
2     3    B
3     4    B
4     5    C
5     6    C
```

In [26]:
```
type(file_data)
```

Out[26]:
```
pandas.core.frame.DataFrame
```

In [27]:
```
df_1 = pd.DataFrame({
    'col1' : np.array([1, 2, 3]),
    'col2' : np.array(["A", "B", "C"])
})
df_2 = pd.DataFrame({
    'col1' : np.array([4, 5, 6]),
    'col2' : np.array(["D", "E", "F"])
})
```

In [28]:
```
# 세로 방향으로 결합
print(pd.concat([df_1, df_2]))
```

```
   col1 col2
0     1    A
1     2    B
2     3    C
0     4    D
1     5    E
2     6    F
```

In [29]:
```
# 가로 방향으로 결합
print(pd.concat([df_1, df_2], axis = 1))
```

```
   col1 col2  col1 col2
0     1    A     4    D
1     2    B     5    E
2     3    C     6    F
```

In [30]:
```
# 대상 데이터
print(sample_df)
```

```
   col1  col2 col3
0     1     2    A
1     2     4    B
2     3     6    C
3     4     8    D
4     5    10    E
```

In [31]:
```
# 열 이름을 지정해서 추출
print(sample_df.col2)
```

```
0     2
1     4
2     6
3     8
4    10
Name: col2, dtype: int32
```

In [32]:
```
print(sample_df["col2"])
```

```
0     2
1     4
2     6
3     8
4    10
Name: col2, dtype: int32
```

In [33]:
```
print(sample_df[["col2", "col3"]])
```

```
   col2 col3
0     2    A
1     4    B
2     6    C
3     8    D
4    10    E
```

In [34]:
```
# 열을 삭제
print(sample_df.drop("col1", axis = 1))
```

```
   col2 col3
0     2    A
1     4    B
2     6    C
3     8    D
4    10    E
```

```
In [35]: # 처음 3행만 추출
         print(sample_df.head(n = 3))
```

```
   col1  col2 col3
0     1     2    A
1     2     4    B
2     3     6    C
```

```
In [36]: # 첫 번째 행을 추출
         print(sample_df.query('index == 0'))
```

```
   col1  col2 col3
0     1     2    A
```

```
In [37]: # 다양한 조건으로 추출
         print(sample_df.query('col3 == "A"'))
```

```
   col1  col2 col3
0     1     2    A
```

```
In [38]: # OR 조건으로 추출
         print(sample_df.query('col3 == "A" | col3 == "D"'))
```

```
   col1  col2 col3
0     1     2    A
3     4     8    D
```

```
In [39]: # AND 조건으로 추출
         print(sample_df.query('col3 == "A" & col1 == 3'))
```

```
Empty DataFrame
Columns: [col1, col2, col3]
Index: []
```

```
In [40]: # 행과 열로 선택
         print(sample_df.query('col3 == "A"')[["col2", "col3"]])
```

```
   col2 col3
0     2    A
```

```
In [41]: # 시리즈
         type(sample_df)
```

```
Out[41]: pandas.core.frame.DataFrame
```

```
In [42]: type(sample_df.col1)
```

```
Out[42]: pandas.core.series.Series
```

```
In [43]: # 시리즈를 배열로 변환
         type(np.array(sample_df.col1))
```

```
Out[43]: numpy.ndarray
```

```
In [44]: type(sample_df.col1.values)
```

```
Out[44]: numpy.ndarray
```

```
In [45]: help(sample_df.query)
```

```
Help on method query in module pandas.core.frame:

query(expr, inplace=False, **kwargs) method of pandas.core.frame.DataFrame
    Query the columns of a DataFrame with a boolean expression.

    Parameters
    ----------
    expr : str
        The query string to evaluate.

        You can refer to variables
        in the environment by prefixing them with an '@' character like
        ``@a + b``.

        You can refer to column names that contain spaces or operators by
        surrounding them in backticks. This way you can also escape
        names that start with a digit, or those that  are a Python keyword
        Basically when it is not valid Python identifier. See notes down
        for more details.
```

추측통계

- 일부 데이터로부터 전체의 통계적 설질을 추측

어느 고등학교에서 전교생 400명이 수학 시험을 동일하게 치렀습니다. 3학년인 A 학생은 이 시험에서 80점을 받았지만, 학교에서 전교생의 평균 점수를 알려주지 않았기 때문에 A 학생은 자신이 전교생 중 어느 정도의 수준인지 알지 못합니다. 자신의 성적이 좋은지 나쁜지가 궁금한 A 학생은 스스로 전교생의 평균 점수를 구해보려 했지만, 400명 전원의 시험 결과를 수소문하는 것은 무리입니다. 그래서 A 학생은 학교 안에서 우연히 만난 20명에게 시험 점수를 물어보고, 그 결과로부터 전교생의 평균 점수를 추측하기로 했습니다.

20명의 시험 점수 평균은 70.4점이었습니다. A 학생은 전교생의 평균도 그 정도일 것으로 생각하고, 자신의 점수가 평균보다 위에 있다는 것에 만족했습니다.

데이터 준비

In [1] :

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

%precision 3
%matplotlib inline
```

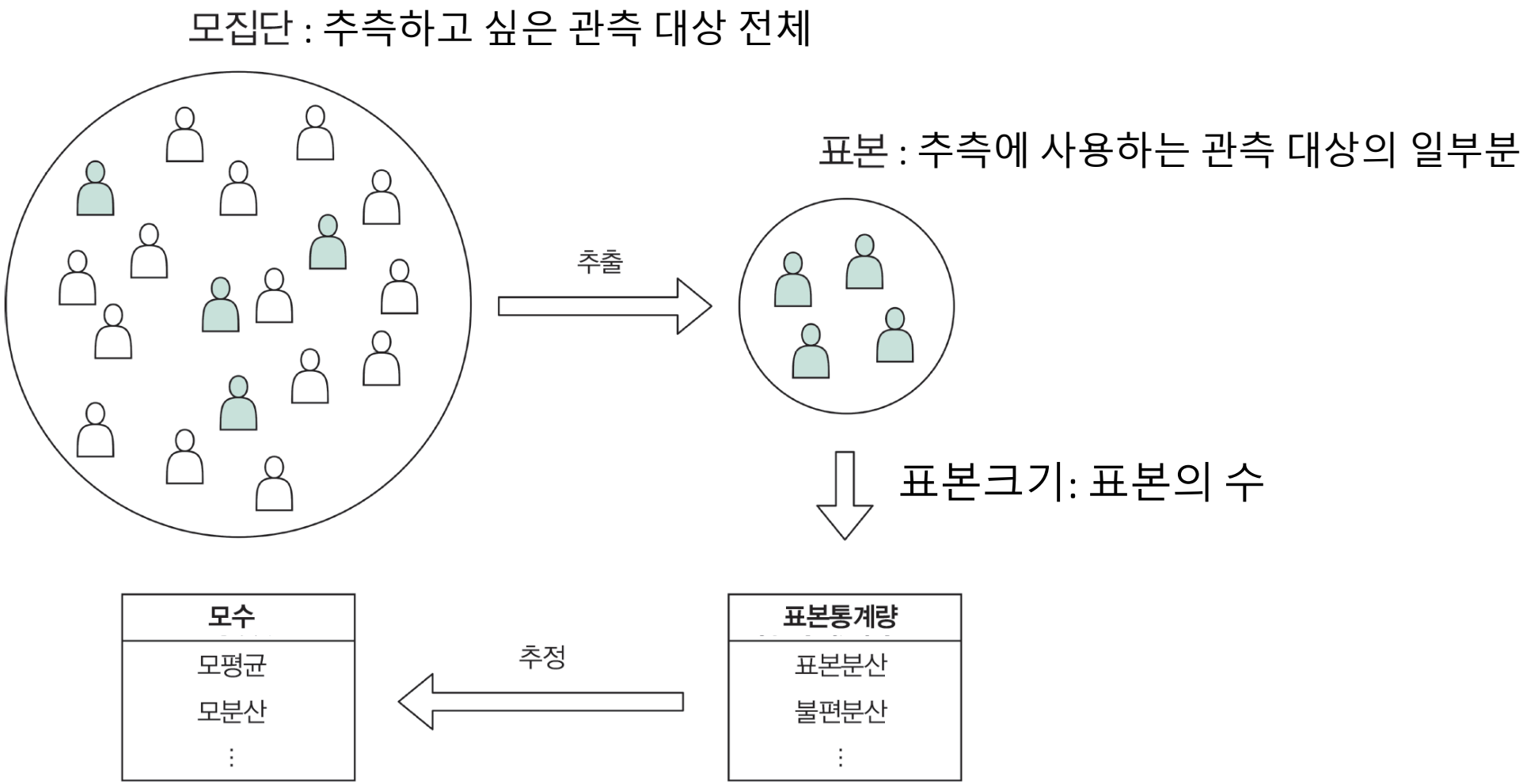
In [2] :

```
df = pd.read_csv( ' ../data/ch4_scores400.csv ' )
scores = np.array(df[ ' score ' ])
scores[:10]
```

Out [2] :

```
array([76, 55, 80, 80, 74, 61, 81, 76, 23, 80])
```

모집단과 표본



[그림 4-1] 모집단과 표본

표본추출 방법

- 무작위 추출(임의 추출): 임의로 표본을 추출하는 방법
- 복원추출: 여러 차례 동일한 표본을 선택하는 방법

In [3] :

```
np.random.choice([1, 2, 3], 3)
```

Out [3] :

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

- 비복원추출: 동일한 표본은 한 번만 선택하는 방법

In [4] :

```
np.random.choice([1, 2, 3], 3, replace=False)
```

Out [4] :

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

표본추출 방법

- 시드를 0으로 하는 무작위 추출(임의 추출)은 매번 동일한 결과

In [5] :

```
np.random.seed(0)
np.random.choice([1, 2, 3], 3)
```

- 표본크기 20으로 복원추출, 표본 평균 계산

In [6] :

```
np.random.seed(0)
sample = np.random.choice(scores, 20)
sample.mean()
```

Out [6] :

```
70.400
```

- 모평균은 69.530(score.mean())이므로 꽤 괜찮은 추측

표본추출 방법

- 무작위 추측은 실행할 때마다 결과가 달라지므로, 표본평균도 매번 달라짐

In [8] :

```
for i in range(5):
    sample = np.random.choice(scores, 20)
    print(f '{i+1}번째 무작위추출로 얻은 표본평균 ', sample.mean())
```

Out [8] :

```
1번째 무작위추출로 얻은 표본평균 72.45
2번째 무작위추출로 얻은 표본평균 63.7
3번째 무작위추출로 얻은 표본평균 66.05
4번째 무작위추출로 얻은 표본평균 71.7
5번째 무작위추출로 얻은 표본평균 74.15
```

확률의 기본

- 확률
  - 무작위 추출과 같은 불확정성을 수반한 현상을 해석
- 확률 모형
  - 무작위 추출 혹은 주사위를 모델링
- 확률변수
  - 결과를 알아맞힐 수는 없지만, 취하는 값과 그 값이 나올 확률이 결정되어 있는 것
- 시행
  - 확률변수의 결과를 관측하는 것
- 실현값
  - 시행에 의해 관측되는 값

확률의 기본

- 사건 : 시행 결과로 나타날 수 있는 값(눈이 1, 눈이 홀수)
  - 주사위의 눈은 확률 변수 X
  - 눈이 1이 되는 사건의 확률 $P(X=1)=\frac{1}{6}$
  - 눈이 홀수인 사건의 확률 $P((X=1)\cup(X=3)\cup(X=5)) = P(X=1)+P(X=3)+P(X=5)$
    $=\frac{1}{6}+\frac{1}{6}+\frac{1}{6}$
    $=\frac{1}{2}$
- 근원사건 : 세부적으로 더 분해될 수 없는 사건(눈이 1)
- 상호배반 : 동시에 일어날 수 없는 사건
  - ‘눈이 1 또는 2 또는 3’이라는 사건과 ’눈이 6’이라는 사건

확률변수

- 확률변수가 어떻게 움직이는지를 나타낸 것
- 공정한 주사위

[표 4-1] 주사위의 확률분포

눈	1	2	3	4	5	6
확률	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

- 불공정한 주사위

[표 4-2] 불공정한 주사위의 확률분포

눈	1	2	3	4	5	6
확률	$\frac{1}{21}$	$\frac{2}{21}$	$\frac{3}{21}$	$\frac{4}{21}$	$\frac{5}{21}$	$\frac{6}{21}$

확률변수

- 불공정한 주사위의 확률분포를 구하는 실험

In [9] :

```
dice = [1, 2, 3, 4, 5, 6]
prob = [1/21, 2/21, 3/21, 4/21, 5/21, 6/21]
```

In [11] :

```
num_trial = 100
sample = np.random.choice(dice, num_trial, p=prob)
sample
```

Out [11] :

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

- 도수분포표와 히스토그램

dice	frequency	relative frequency
1	5	0.05
2	9	0.09
3	13	0.13
4	27	0.27
5	19	0.19
6	27	0.27

확률변수

- 불공정한 주사위의 확률분포를 구하는 실험

In [9] :

```
dice = [1, 2, 3, 4, 5, 6]
prob = [1/21, 2/21, 3/21, 4/21, 5/21, 6/21]
```

In [11] :

```
num_trial = 100
sample = np.random.choice(dice, num_trial, p=prob)
sample
```

Out [11] :

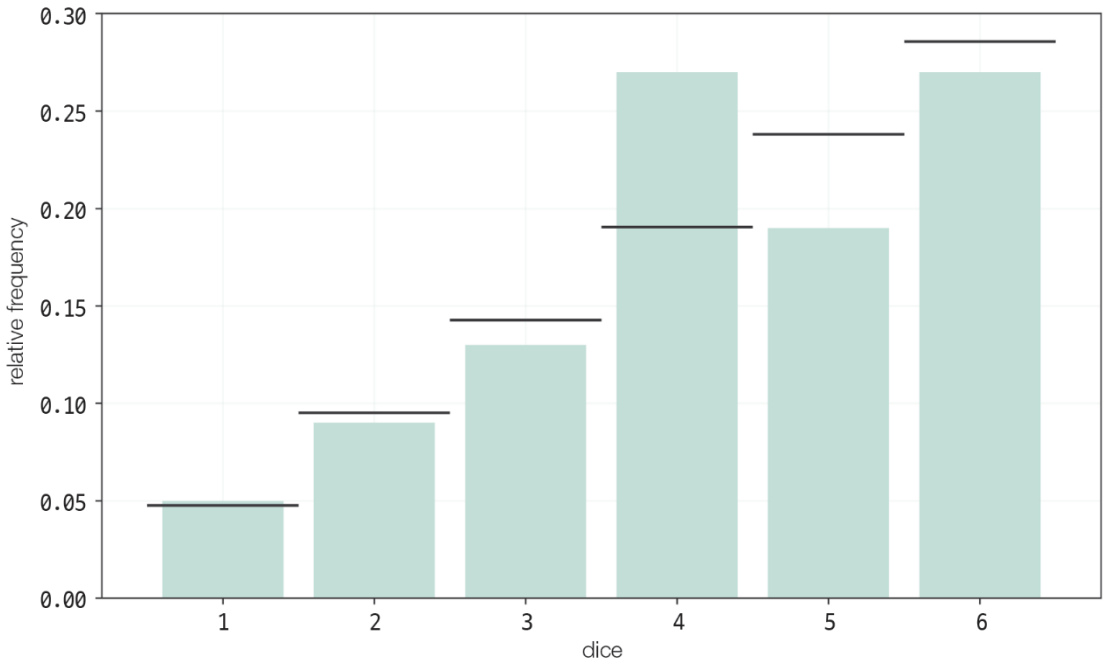
```
array([4, 6, 4, 5, 5, 6, 6, 3, 5, 6, 5, 6, 6, 2, 3, 1, 6, 5, 6, 3,
       4, 5, 3, 4, 3, 5, 5, 4, 4, 6, 4, 6, 5, 6, 5, 4, 6, 2, 6, 4,
       5, 3, 4, 6, 5, 5, 5, 3, 4, 5, 4, 4, 6, 4, 4, 6, 6, 2, 2, 4,
       5, 1, 6, 4, 3, 2, 2, 6, 3, 5, 4, 2, 4, 4, 6, 6, 1, 5, 3, 6,
       6, 4, 2, 1, 6, 4, 4, 2,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

- 도수분포표

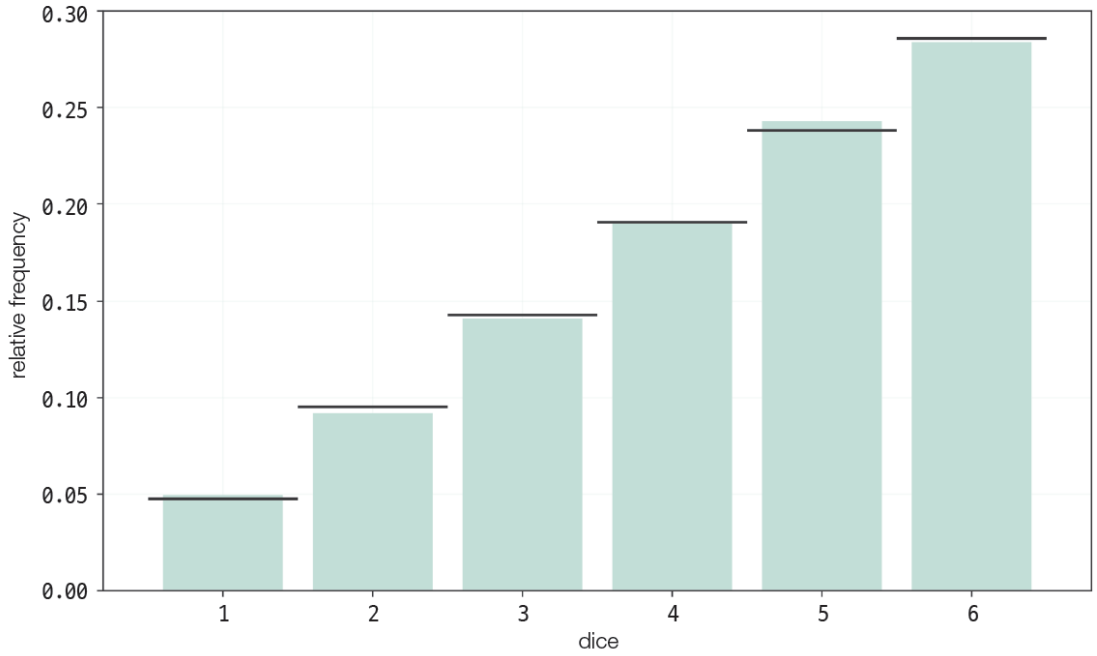
dice	frequency	relative frequency
1	5	0.05
2	9	0.09
3	13	0.13
4	27	0.27
5	19	0.19
6	27	0.27

확률변수

- 히스토그램
  - 10000번 시행했을 때의 히스토그램은 실제의 확률분포에 가까워짐



[그림 4-2] 100번 시행했을 때 주사위 눈에 대한 히스토그램



[그림 4-3] 10000번 시행했을 때 주사위 눈의 히스토그램 SAMPLE CODE

추측통계의 확률

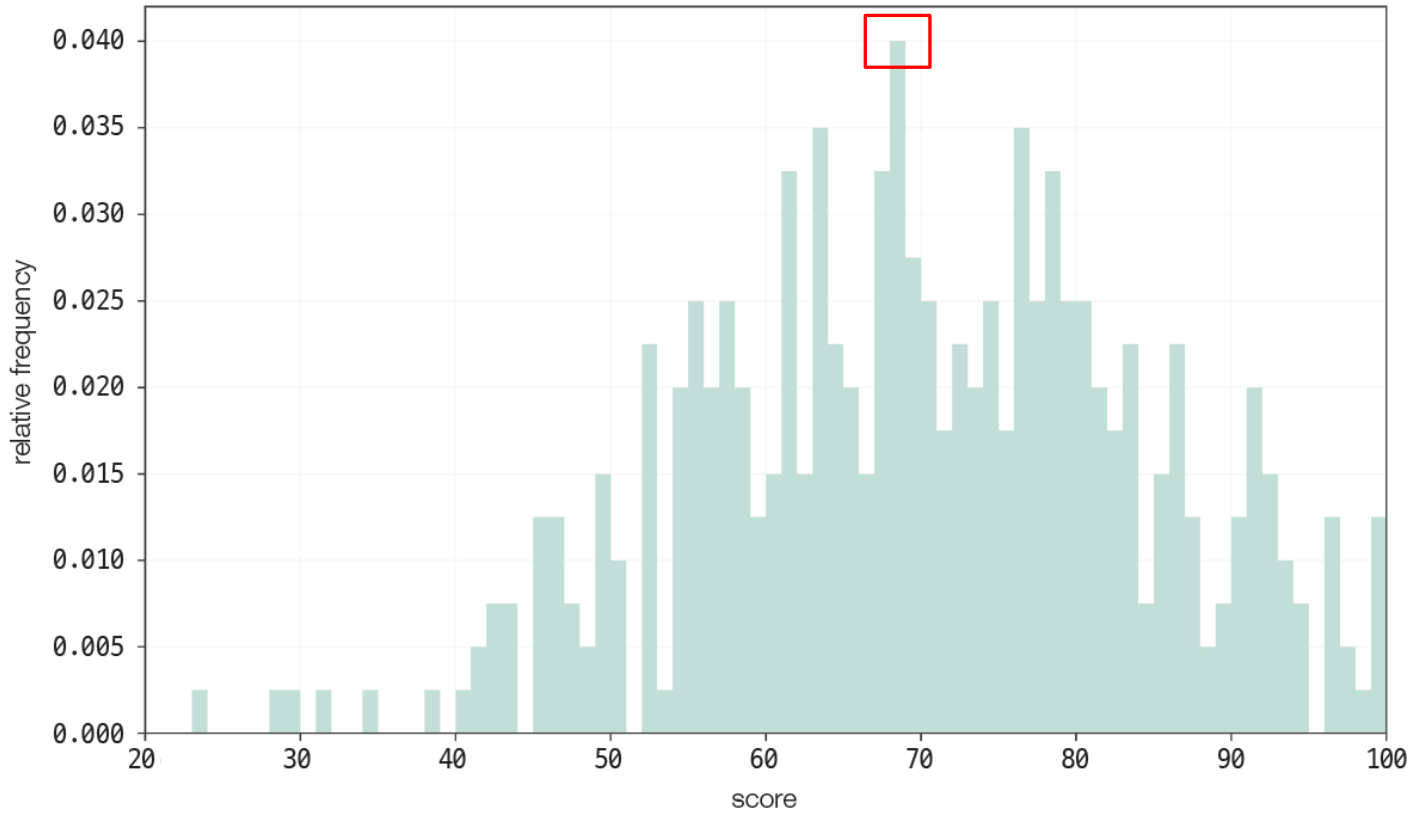
- 계급폭을 1점으로 하는 히스토그램

In [15] :

```
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
ax.hist(scores, bins=100, range=(0, 100), density=True)
ax.set_xlim(20, 100)
ax.set_ylim(0, 0.042)
ax.set_xlabel( ‘ score ’ )
ax.set_ylabel( ‘ relative frequency ’ )
plt.show()
```

추측통계의 확률

- 69점을 얻은 학생은 전교생의 0.04(4%)이므로 무작위추출을 수행하면 4%의 확률로 69점이라는 표본 데이터 획득



[그림 4-4] 전교생 시험 점수에 대한 히스토그램

추측통계의 확률

- 무작위추출은 확률분포를 따르는 확률변수의 시행

In [16] :

```
np.random.choice(scores)
```

Out [16] :

```
89
```

무작위추출로 얻은 표본 데이터가 89점

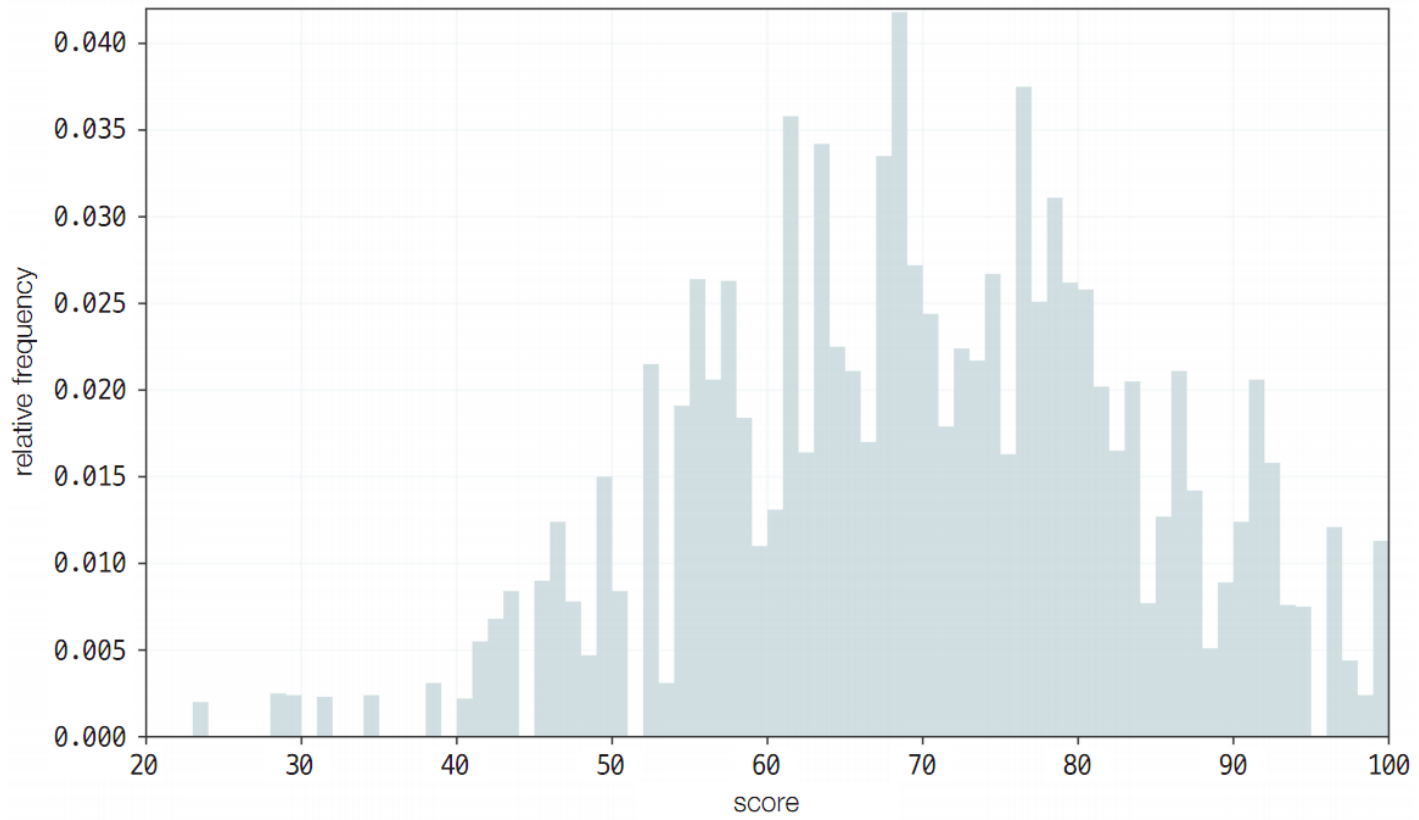
- 시행 횟수를 늘리면 주사위의 상대도수는 실제의 확률분포에 가까워짐
- 무작위추출에서도 표본의 크기가 커지면, 표본 데이터의 상대도수는 실제의 확률분포에 근사
- 무작위추출로 샘플 사이즈가 10000인 표본 추출

In [17] :

```
sample = np.random.choice(scores, 10000)

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
ax.hist(sample, bins=100, range=(0, 100), density=True)
ax.set_xlim(20, 100)
ax.set_ylim(0, 0.042)
ax.set_xlabel( ' score ' )
ax.set_ylabel( ' relative frequency ' )
plt.show()
```

추측통계의 확률

- 히스토그램이 실제의 점수 분포에 가까운 형태
- 표본 크기가 커지면 실제의 분포에 수렴



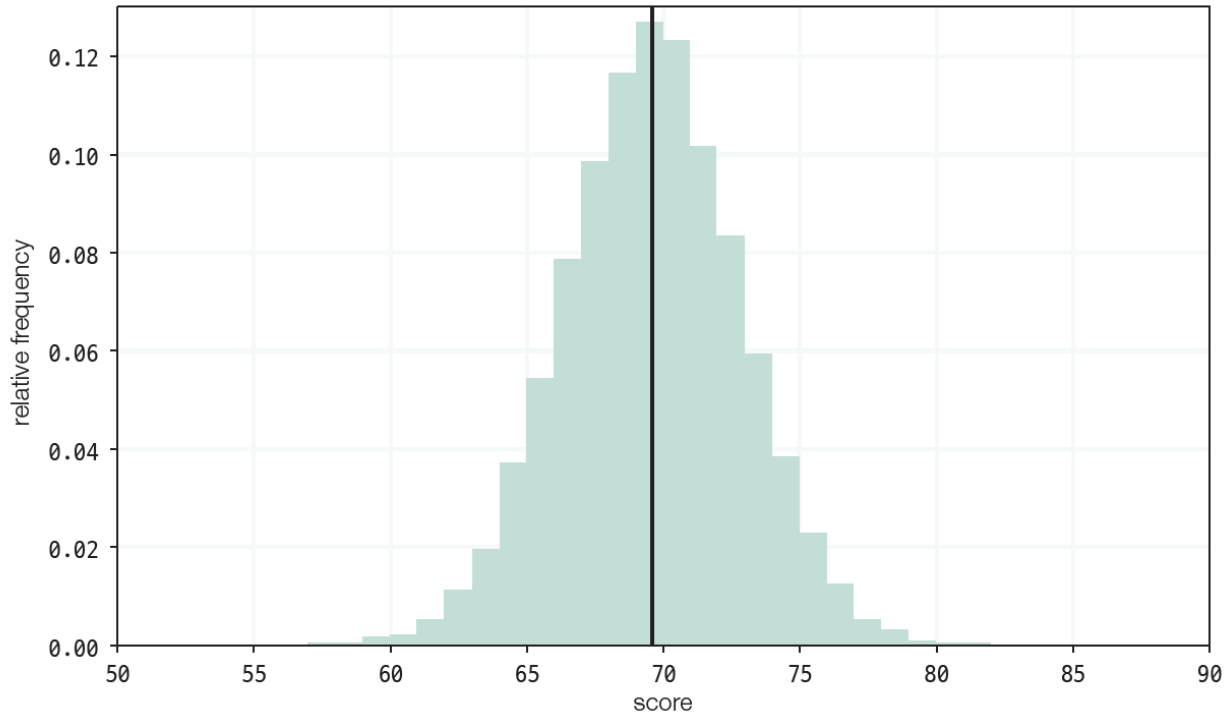
[그림 4-5] 무작위추출로 얻은 표본 데이터의 히스토그램 SAMPLE CODE

추측통계의 확률

- 표본크기가 20인 표본을 추출하여 표본평균을 계산하는 작업을 10000번 수행
- 표본평균은 모평균을 중심으로 분포
  - => **무작위추출에 의한 표본평균으로 모평균 추측 가능**

In [18] :

```
sample_means = [np.random.choice(scores, 20).mean()
                for _ in range(10000)]

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
ax.hist(sample_means, bins=100, range=(0, 100), density=True)
# 모평균을 세로선으로 표시
ax.vlines(np.mean(scores), 0, 1, 'gray')
ax.set_xlim(50, 90)
ax.set_ylim(0, 0.13)
ax.set_xlabel('score')
ax.set_ylabel('relative frequency')
plt.show()
```



[그림 4-6] 표본평균의 분포 SAMPLE CODE

1차형 이산형 확률변수

- 확률변수 X가 취할 수 있는 값의 집합 $\{x_1, x_2, \cdots\}$
- X가 x_k라는 값을 취하는 확률 $P(X = x_k) = p_k \quad (k = 1, 2, \cdots)$
- 확률질량함수(확률함수) $f(x) = P(X = x)$

1차형 이산형 확률변수

- 불공정한 주사위의 확률분포
  - 확률변수가 취할 수 있는 값의 집합 x_set
  - x_set에 대응하는 확률

In [2] :

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

[표 5-1] 불공정한 주사위의 확률분포

눈	1	2	3	4	5	6
확률	$\frac{1}{21}$	$\frac{2}{21}$	$\frac{3}{21}$	$\frac{4}{21}$	$\frac{5}{21}$	$\frac{6}{21}$

- 불공정한 주사위의 확률변수

$$f(x) = \begin{cases} \frac{x}{21} & (x \in \{1, 2, 3, 4, 5, 6\}) \\ 0 & (otherwise) \end{cases}$$

$$p_1 = P(X = 1) = \frac{1}{21}$$

$$p_2 = P(X = 2) = \frac{2}{21}$$

$$\vdots$$

1차형 이산형 확률변수

In [2] :

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

$$\frac{1}{21} = 0.048,\ \frac{2}{21} = 0.095,\ \frac{3}{21} = 0.143,\ \cdots$$

In [3] :

```
def f(x):
    if x in x_set:
        return x / 21
    else:
        return 0
```

In [4] :

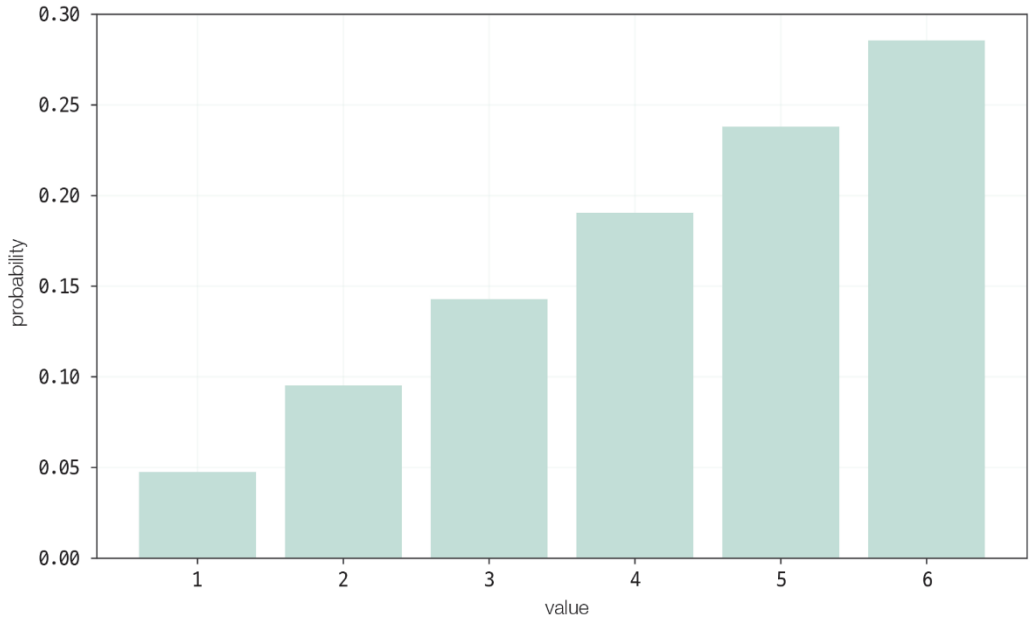
```
X = [x_set, f]
```

In [5] :

```
# 확률 p_k를 구한다
prob = np.array([f(x_k) for x_k in x_set])
# x_k와 p_k의 대응을 사전식으로 표시
dict(zip(x_set, prob))
```

Out [5] :

```
{1: 0.048, 2: 0.095, 3: 0.143, 4: 0.190, 5: 0.238, 6: 0.286}
```



[그림 5-1] 확률분포

1차형 이산형 확률변수

In [2] :

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

$$\frac{1}{21} = 0.048,\ \frac{2}{21} = 0.095,\ \frac{3}{21} = 0.143,\ \cdots$$

In [3] :

```
def f(x):
    if x in x_set:
        return x / 21
    else:
        return 0
```

In [4] :

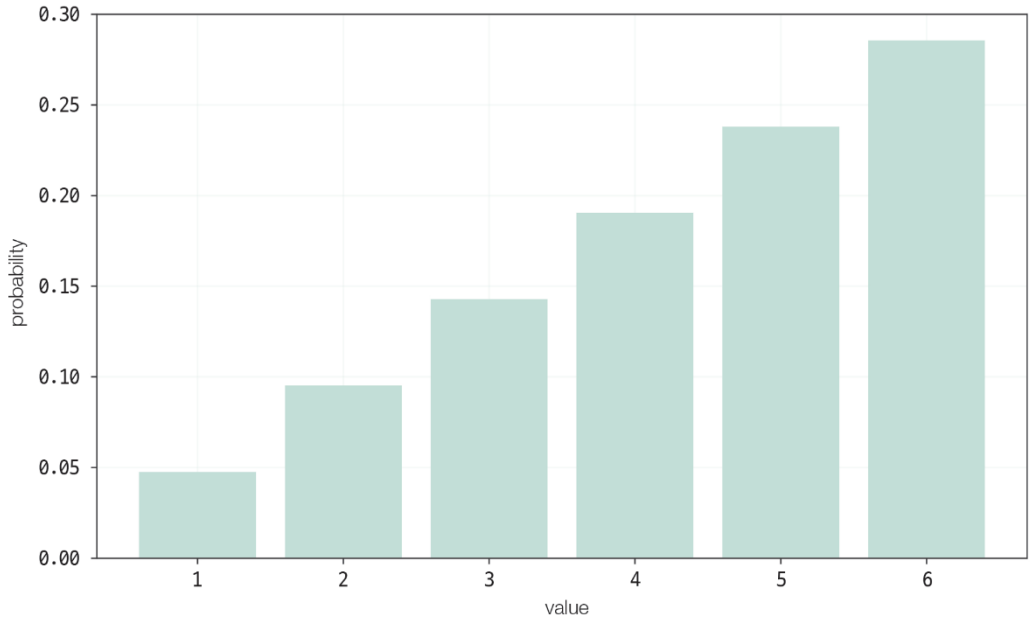
```
X = [x_set, f]
```

In [5] :

```
# 확률 p_k를 구한다
prob = np.array([f(x_k) for x_k in x_set])
# x_k와 p_k의 대응을 사전식으로 표시
dict(zip(x_set, prob))
```

Out [5] :

```
{1: 0.048, 2: 0.095, 3: 0.143, 4: 0.190, 5: 0.238, 6: 0.286}
```



[그림 5-1] 확률분포

1차형 이산형 확률변수

```
In [1]: a = {'사과':1, '딸기':5, '귤':10}

In [2]: a
Out[2]: {'사과': 1, '딸기': 5, '귤': 10}

In [3]: a ={('초콜릿', 200):20, ('마카롱', 500):15, ('쿠키', 300):30}
        a
Out[3]: {('초콜릿', 200): 20, ('마카롱', 500): 15, ('쿠키', 300): 30}

In [4]: a = {'사과':1, '딸기':5, '귤':10}
        v1 = a['딸기']
        v1
Out[4]: 5

In [5]: v2=a['레몬']
        v2
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-5-780f342b305b> in <module>
----> 1 v2=a['레몬']
      2 v2

KeyError: '레몬'
```

```
In [6]: f1 = '딸기' in a
        f1
Out[6]: True

In [7]: f2 = '레몬' not in a
        f2
Out[7]: True

In [8]: f3 = '레몬' in a
        f3
Out[8]: False
```

```
In [9]: v1 = a.get('딸기')
        v1
Out[9]: 5

In [10]: v2 = a.get('레몬')
         v2
```

1차형 이산형 확률변수

In [11]:
```
a = {'초콜릿':1, '마카롱':2, '쿠키':3}
a['초콜릿'] = 'One'
a['마카롱'] = 'Two'
a['쿠키'] = 'Three'
a
```

Out[11]:
```
{'초콜릿': 'One', '마카롱': 'Two', '쿠키': 'Three'}
```

In [13]:
```
d = dict(초콜릿 = 20, 마카롱 = 15, 쿠키 = 30)
d
```

Out[13]:
```
{'초콜릿': 20, '마카롱': 15, '쿠키': 30}
```

In [14]:
```
key = ['초콜릿', '마카롱', '쿠키']
value = [20, 15, 30]
d = dict(zip(key, value))
d
```

Out[14]:
```
{'초콜릿': 20, '마카롱': 15, '쿠키': 30}
```

In [15]:
```
d = dict([('초콜릿', 20), ('마카롱', 15), ('쿠키', 30)])
d
```

Out[15]:
```
{'초콜릿': 20, '마카롱': 15, '쿠키': 30}
```

1차형 이산형 확률변수

- 확률의 성질

$$f(x_k) \geq 0$$

$$\sum_k f(x_k) = 1$$

In [7] :

```
np.all(prob >= 0)
```

Out [7] :

```
True
```

  - np.all은 모든 요소가 참일 때만 참을 반환

- 확률의 총합은 1

$$\frac{1}{21}+\frac{2}{21}+\frac{3}{21}+\frac{4}{21}+\frac{5}{21}+\frac{6}{21}=1$$

In [8] :

```
np.sum(prob)
```

Out [8] :

```
1.000
```

1차형 이산형 확률변수

- 누적분포함수(분포함수) $F(x)$
  - X가 x 이하가 될 때의 확률을 반환하는 함수

$$F(x) = P(X \leq x) = \sum_{x_k \leq x} f(x_k)$$

In [9] :

```
def F(x):
    return np.sum([f(x_k) for x_k in x_set if x_k <= x])
```

- 눈이 3 이하가 되는 확률

$$F(3) = P(X \leq 3) = \sum_{x_k \leq 3} f(x_k)$$

$$\frac{1}{21} + \frac{2}{21} + \frac{3}{21} = 0.048 + 0.095 + 0.143 = 0.286$$

In [10] :

```
F(3)
```

Out [10] :

```
0.286
```

1차형 이산형 확률변수

- 확률변수의 변환
  - 확률변수 X에 2를 곱하고 3을 더한 $2X + 3$도 확률변수
  - $2X + 3$을 확률변수 Y라고 하면

- Y의 확률분포

In [11] :

```
y_set = np.array([2 * x_k + 3 for x_k in x_set])
prob = np.array([f(x_k) for x_k in x_set])
dict(zip(y_set, prob))
```

Out [11] :

```
{5: 0.048, 7: 0.095, 9: 0.143, 11: 0.190, 13: 0.238, 15: 0.286}
```

1차형 이산형 확률변수의 지표

- 기댓값 = 확률변수의 평균
  - 확률변수를 몇 번이나(무제한) 시행하여 얻어진 실현값의 평균
  - 무제한 시행할 수 없으므로 확률변수가 취할 수 있는 값과 확률의 곱의 총합 $E(X) = \sum_k x_k f(x_k)$
  - 불공정한 주사위의 기댓값

$$1 \times 0.048 + 2 \times 0.095 + 3 \times 0.143 + 4 \times 0.190 + 5 \times 0.238 + 6 \times 0.286 = 4.333$$

In [12] :

```
np.sum([x_k * f(x_k) for x_k in x_set])
```

  - 주사위를 100만(10^6)번 굴린 실현값의 평균

In [13] :

```
sample = np.random.choice(x_set, int(1e6), p=prob)
np.mean(sample)
```

Out [13] :

```
4.333
```

  - 확률변수 X를 $2X + 3$ 으로 변환한 Y의 기댓값 $E(Y) = E(2X + 3) = \sum_k (2x_k + 3) f(x_k)$

1차형 이산형 확률변수의 지표

이미 있는 데이터 집합에서 일부를 무작위로 선택하는 것을 샘플링(sampling)이라고 한다. 샘플링에는 `choice` 명령을 사용한다. `choice` 명령은 다음과 같은 인수를 가질 수 있다.

```
numpy.random.choice(a, size=None, replace=True, p=None)
```

- a : 배열이면 원래의 데이터, 정수이면 arange(a) 명령으로 데이터 생성
- size : 정수. 샘플 숫자
- replace : 불리언. True이면 한번 선택한 데이터를 다시 선택 가능
- p : 배열. 각 데이터가 선택될 수 있는 확률

In [11]:

```
np.random.choice(5, 5, replace=False)  # shuffle 명령과 같다.
```

Out:

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

1차형 이산형 확률변수의 지표

In [12]:

```
np.random.choice(5, 3, replace=False)  # 3개만 선택
```

Out:

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

In [13]:

```
np.random.choice(5, 10)  # 반복해서 10개 선택
```

Out:

```
array([0, 4, 1, 4, 1, 2, 2, 0, 1, 1])
```

In [14]:

```
np.random.choice(5, 10, p=[0.1, 0, 0.3, 0.6, 0])  # 선택 확률을 다르게 해서 10개 선택
```

Out:

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

1차형 이산형 확률변수의 지표

- 람다(lambda) 함수 - 익명함수
  - 값을 반환하는 단순한 한 문장으로 이루어진 함수
  - 코드를 적게 쓰고 더 간결해짐

```
def short_function(x):
    return x*2

equiv_anon = lambda x: x*2
```

```
def apply_to_list(some_list, f):
    return [(f(x) for x in some_list]

ints = [4, 0, 1, 5, 6]
apply_to_list(ints, lambda x: x*2)
```

```
In [1]: strings = ['hyeja', 'parkhyeja', 'youngtae', 'kimyoungtae', 'bbangtae']

In [3]: strings.sort(key=lambda x: len(set(list(x))))

In [4]: strings
Out[4]: ['hyeja', 'bbangtae', 'parkhyeja', 'youngtae', 'kimyoungtae']
```

[x*2 for x in ints]

1차형 이산형 확률변수의 지표

- 기댓값 = 확률변수의 평균
  - 수식을 기댓값의 함수로 구현
  - 인수 g가 확률변수에 대한 변환의 함수
  - g에 아무것도 지정하지 않으면
    확률변수 X의 기댓값이 구해짐

이산형 확률변수의 기댓값

$$E(g(X)) = \sum_k g(x_k) f(x_k)$$

In [14] :

```
def E(X, g=lambda x: x):
    x_set, f = X
    return np.sum([g(x_k) * f(x_k) for x_k in x_set])
```

  - 확률변수 $Y = 2X + 3$의 기댓값

In [16] :

```
E(X, g=lambda x: 2*x + 3)
```

Out [16] :

```
11.667
```

$$(2\times1+3)\times0.048+(2\times2+3)\times0.095+\ \cdots\ (2\times6+3)\times0.286$$

$$=11.667$$

1차형 이산형 확률변수의 지표

- 기댓값 = 확률변수의 평균

기댓값의 선형성

a, b를 실수, X를 확률변수로 했을 때

$$E(aX+b)=aE(X)+b$$

가 성립합니다.

- $\mathrm{E}(2X+3) \equiv 2\mathrm{E}(X)+3$

In [17] :

```
2 * E(X) + 3
```

Out [17] :

```
11.667
```

1차형 이산형 확률변수의 지표

- 분산

$$V(X) = \sum_k (x_k - \mu)^2 f(x_k)$$
$$= (1-4.333)^2 \times 0.048 + (2-4.333)^2 \times 0.095 + \cdots + (6-4.333)^2 \times 0.286$$
$$= 2.222$$

  - 불공정한 주사위의 분산

In [18] :
```
mean = E(X)
np.sum([(x_k-mean)**2 * f(x_k) for x_k in x_set])
```

Out [18] :
```
2.222
```

  - 확률변수 $Y = 2X + 3$의 분산 $V(2X+3) = \sum_k ((2x_k + 3) - \mu)^2 f(x_k)$

1차형 이산형 확률변수의 지표

- 분산
  - 이산형 확률변수의 분산식을 분산의 함수로 구현

이산형 확률변수의 분산

$$V(g(X)) = \sum_k (g(x_k)) - E(g(X)))^2 f(x_k)$$

  - 인수 g가 확률변수에 대한 변환의 함수

In [19] :

```
def V(X, g=lambda x: x):
    x_set, f = X
    mean = E(X, g)
    return np.sum([(g(x_k)-mean)**2 * f(x_k) for x_k in x_set])
```

In [20] :

```
V(X)
```

Out [20] :

```
2.222
```

  - 확률변수 $Y = 2X + 3$의 분산

In [21] :

```
V(X, lambda x: 2*x + 3)
```

Out [21] :

```
8.889
```

1차형 이산형 확률변수의 지표

- 분산

분산의 공식

a, b를 실수, X를 확률변수라고 하면

$$V(aX+b)=a^2V(X)$$

가 성립합니다.

- $V(2X+3)=2^2V(X)$

In [22] :

```
2**2 * V(X)
```

Out [22] :

```
8.889
```

1차형 이산형 확률변수의 지표

- 분산

분산의 공식

a, b를 실수, X를 확률변수라고 하면

$$V(aX+b)=a^2V(X)$$

가 성립합니다.

- $V(2X+3)=2^2V(X)$

In [22] :

```
2**2 * V(X)
```

Out [22] :

```
8.889
```

2차형 이산형 확률변수

- 결합확률분포

1차원 확률분포 2개를 동시에 다룹니다$(X,\ Y)$

확률은 X와 Y가 각각 취할 수 있는 값의 조합에 관해서 정의

  - 확률변수 X가 x_i, 확률변수 Y가 y_j를 취하는 확률 $P(X = x_i,\ Y = y_j) = p_{ij} \qquad (i = 1, 2, \cdots ;\ j = 1, 2, \cdots)$
  - 확률변수 (X, Y)의 움직임을 동시에 고려한 분포
  - 불공정한 주사위 A와 B
    - A와 B의 눈을 더한 것 X, A의 눈을 Y로 하는 2차원 확률분포
  - 결합확률함수
    - $P(X = x, Y = y) = f_{xy}(x, y)$ $\qquad f_{XY}(x,\ y) = \begin{cases} \dfrac{y(x-y)}{441} \\ 0 \end{cases}$

2차형 이산형 확률변수

[표 5-2] 불공정한 주사위의 결합확률분포

X \ Y	1	2	3	4	5	6
2	$\frac{1}{441}$	0	0	0	0	0
3	$\frac{2}{441}$	$\frac{2}{441}$	0	0	0	0
4	$\frac{3}{441}$	$\frac{4}{441}$	$\frac{3}{441}$	0	0	0
5	$\frac{4}{441}$	$\frac{6}{441}$	$\frac{6}{441}$	$\frac{4}{441}$	0	0
6	$\frac{5}{441}$	$\frac{8}{441}$	$\frac{9}{441}$	$\frac{8}{441}$	$\frac{5}{441}$	0
7	$\frac{6}{441}$	$\frac{10}{441}$	$\frac{12}{441}$	$\frac{12}{441}$	$\frac{10}{441}$	$\frac{6}{441}$
8	0	$\frac{12}{441}$	$\frac{15}{441}$	$\frac{16}{441}$	$\frac{15}{441}$	$\frac{12}{441}$
9	0	0	$\frac{18}{441}$	$\frac{20}{441}$	$\frac{20}{441}$	$\frac{18}{441}$
10	0	0	0	$\frac{24}{441}$	$\frac{25}{441}$	$\frac{24}{441}$
11	0	0	0	0	$\frac{30}{441}$	$\frac{30}{441}$
12	0	0	0	0	0	$\frac{36}{441}$

$$\frac{4}{21} \times \frac{5}{21} = \frac{20}{441}$$

2차형 이산형 확률변수

- 확률의 성질

$$f_{XY}(x_i, y_j) \geq 0$$

$$\sum_i \sum_j f_{XY}(x_i, y_j) = 1$$

$$f_{XY}(x_2, y_1) + f_{XY}(x_3, y_1) + f_{XY}(x_3 + y_2) + \cdots + f_{XY}(x_{12} + y_6)$$
$$= \frac{1}{441} + \frac{2}{441} + \frac{2}{441} + \cdots + \frac{36}{441}$$
$$= 1$$

  - X와 Y가 취할 수 있는 값의 집합

  - 결합확률함수

In [23] :

```
x_set = np.arange(2, 13)
y_set = np.arange(1, 7)
```

In [24] :

```
def f_XY(x, y):
    if 1 <= y <=6 and 1 <= x - y <= 6:
        return y * (x-y) / 441
    else:
        return 0
```

In [25] :

```
XY = [x_set, y_set, f_XY]
```

In [27] :

```
np.all(prob >= 0)
```

Out [27] :

```
True
```

In [28] :

```
np.sum(prob)
```

Out [28] :

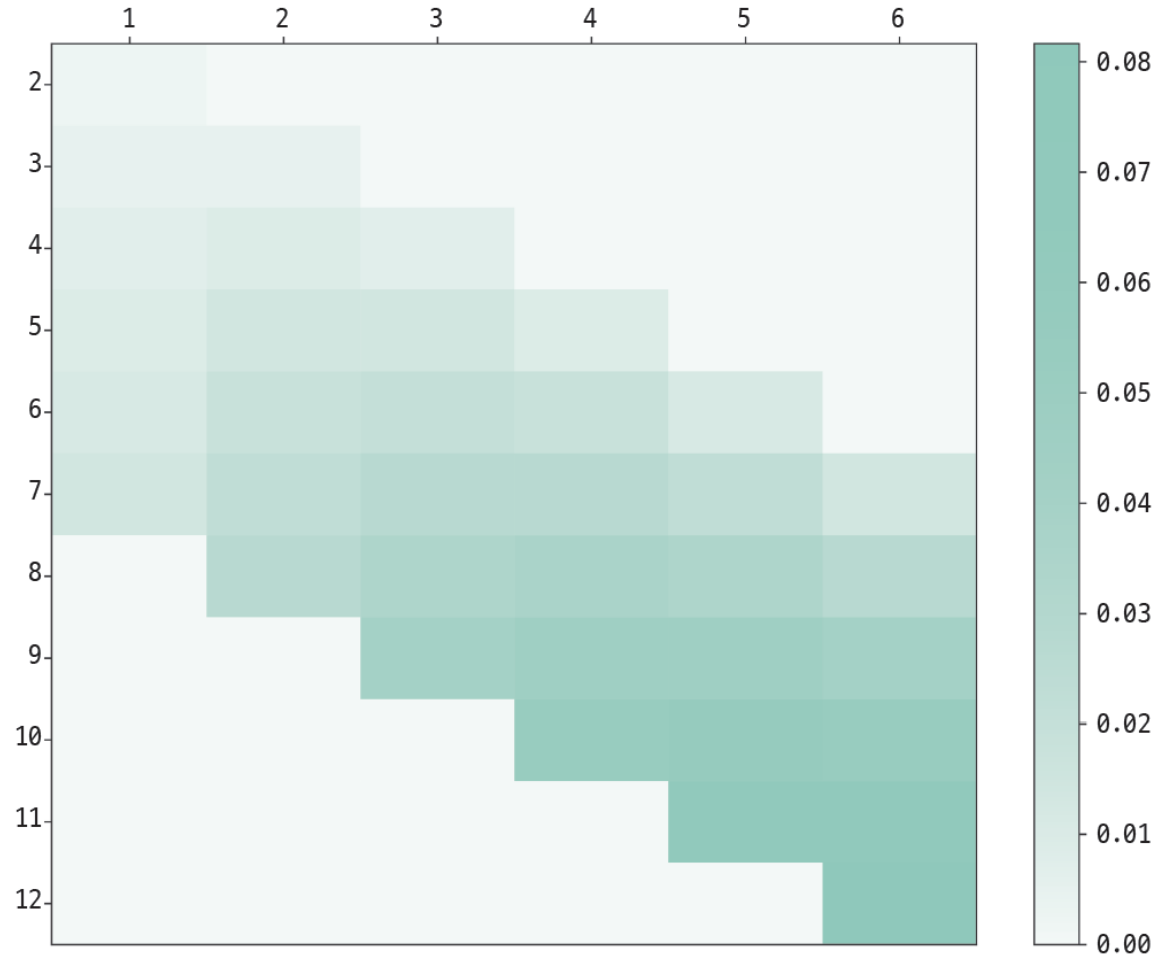
```
1.000
```

2차형 이산형 확률변수

- 확률의 성질
  - 확률분포의 히트맵

In [26] :

```
prob = np.array([[f_XY(x_i, y_j) for y_j in y_set]
                 for x_i in x_set])

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111)

c = ax.pcolor(prob)
ax.set_xticks(np.arange(prob.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(prob.shape[0]) + 0.5, minor=False)
ax.set_xticklabels(np.arange(1, 7), minor=False)
ax.set_yticklabels(np.arange(2, 13), minor=False)
# y축을 내림차순의 숫자가 되게 하여, 위 아래를 역전시킨다
ax.invert_yaxis()
# x축 눈금을 그래프 위쪽에 표시
ax.xaxis.tick_top()
fig.colorbar(c, ax=ax)
plt.show()
```



[그림 5-2] 2차원 확률분포의 히트맵

2차형 이산형 확률변수

- 주변확률분포

개별 확률변수에만 흥미

  - 확률변수 (X, Y)는 결합확률분포에 의해 동시에 정의되지만, 확률변수 X의 확률함수 $f_X(x)$를 알고 싶을 때
  - f_{XY}에서 Y가 취할 수 있는 값 모두를 대입한 다음 모두 더한

$$f_X(x) = \sum_k f_{XY}(x, y_k)$$

결합확률함수 f_{XY}에서 확률변수 Y의 영향을 제거

In [29] :

```
def f_X(x):
    return np.sum([f_XY(x, y_k) for y_k in y_set])
```

In [30] :

```
def f_Y(y):
    return np.sum([f_XY(x_k, y) for x_k in x_set])
```

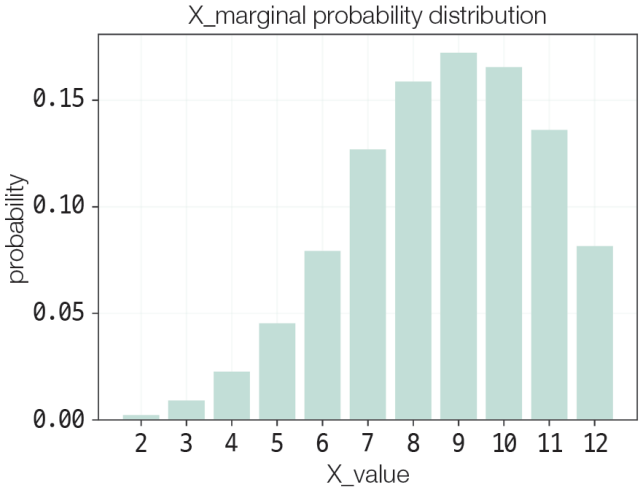
In [31] :

```
X = [x_set, f_X]
Y = [y_set, f_Y]
```

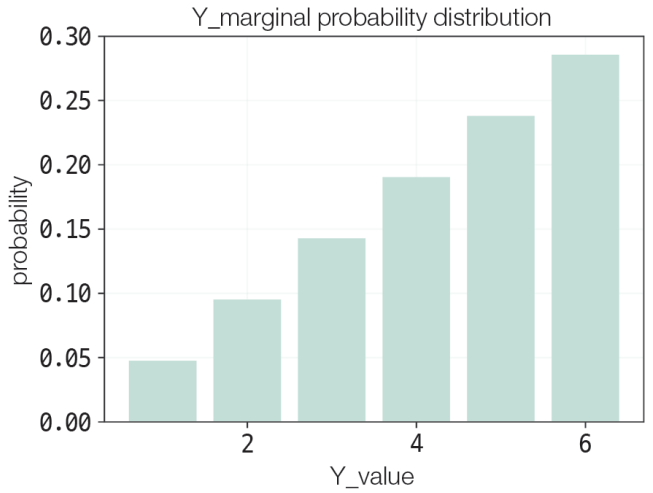
2차형 이산형 확률변수

In [32] :

```
prob_x = np.array([f_X(x_k) for x_k in x_set])
prob_y = np.array([f_Y(y_k) for y_k in y_set])

fig = plt.figure(figsize=(12, 4))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

ax1.bar(x_set, prob_x)
ax1.set_title( ' X_marginal probability distribution ' )
ax1.set_xlabel( ' X_value ' )
ax1.set_ylabel( ' probability ' )
ax1.set_xticks(x_set)

ax2.bar(y_set, prob_y)
ax2.set_title( ' Y_marginal probability distribution ' )
ax2.set_xlabel( ' Y_value ' )
ax2.set_ylabel( ' probability ' )

plt.show()
```



[그림 5-3] 주변분포



2차형 이산형 확률변수의 지표

- 기댓값

$$\mu_X = E(X) = \sum_i \sum_j x_i f_{XY}(x_i, y_j) \qquad E(g(X, Y)) = \sum_i \sum_j g(x_i, y_j) f_{XY}(x_i, y_j)$$

In [33] :

```
np.sum([x_i * f_XY(x_i, y_j) for x_i in x_set for y_j in y_set])
```

Out [33] :

```
8.667
```

  - 기댓값의 함수로 구현

In [34] :

```
def E(XY, g):
    x_set, y_set, f_XY = XY
    return np.sum([g(x_i, y_j) * f_XY(x_i, y_j)
                   for x_i in x_set for y_j in y_set])
```

$$2 \times \frac{1}{441} + 3 \times \left(\frac{2}{441} + \frac{2}{441} \right) + \cdots + 36 \times \frac{36}{441} = 8.667$$

2차형 이산형 확률변수의 지표

- X와 Y의 기댓값

In [35] :

```
mean_X = E(XY, lambda x, y: x)
mean_X
```

Out [35] :

```
8.667
```

In [36] :

```
mean_Y = E(XY, lambda x, y: y)
mean_Y
```

Out [36] :

```
4.333
```

기댓값의 선형성

a, b를 실수, X, Y를 확률변수로 했을 때

$$E(aX+bY)=aE(X)+bE(Y)$$

가 성립합니다.

In [37] :

```
a, b = 2, 3
```

In [38] :

```
E(XY, lambda x, y: a*x + b*y)
```

Out [38] :

```
30.333
```

In [39] :

```
a * mean_X + b * mean_Y
```

Out [39] :

```
30.333
```

$$2\times 8.667+3\times 4.333=30.333$$

2차형 이산형 확률변수의 지표

- 분산
  - X의 분산은 X에 관한 편차 제곱의 기댓값

$$\sigma_X^2 = V(X) = \sum_i \sum_j (x_i - \mu_X)^2 f_{XY}(x_i, y_j)$$

In [40] :

```
np.sum([(x_i-mean_X)**2 * f_XY(x_i, y_j)
        for x_i in x_set for y_j in y_set])
```

Out [40] :

```
4.444
```

  - X와 Y의 함수 g(X, Y)의 분산

$$V(g(X, Y)) = \sum_i \sum_j (g(x_i, y_j) - E(g(X, Y)))^2 f_{XY}(x_i, y_j)$$

In [41] :

```
def V(XY, g):
    x_set, y_set, f_XY = XY
    mean = E(XY, g)
    return np.sum([(g(x_i, y_j)-mean)**2 * f_XY(x_i, y_j)
                   for x_i in x_set for y_j in y_set])
```

2차형 이산형 확률변수의 지표

- 분산
  - X와 Y의 분산

In [42] :

```
var_X = V(XY, g=lambda x, y: x)
var_X
```

Out [42] :

```
4.444
```

In [42] :

```
var_X = V(XY, g=lambda x, y: x)
var_X
```

Out [42] :

```
4.444
```

- 공분산
  - 두 확률변수 X, Y 사이의 상관

$$\sigma_{XY} = Cov(X,\ Y) = \sum_i \sum_j (x_i - \mu_X)(y_j - \mu_Y) f_{XY}(x_i,\ y_j)$$

In [44] :

```
def Cov(XY):
    x_set, y_set, f_XY = XY
    mean_X = E(XY, lambda x, y: x)
    mean_Y = E(XY, lambda x, y: y)
    return np.sum([(x_i-mean_X) * (y_j-mean_Y) * f_XY(x_i, y_j)
                   for x_i in x_set for y_j in y_set])
```

In [45] :

```
cov_xy = Cov(XY)
cov_xy
```

Out [45] :

```
2.222
```

2차형 이산형 확률변수의 지표

분산과 공분산의 공식

a, b를 실수, X, Y를 확률변수로 했을 때

$$V(aX+bY)=a^2V(X)+b^2V(Y)+2abCov(X,Y)$$

가 성립합니다.

In [46] :

```
V(XY, lambda x, y: a*x + b*y)
```

Out [46] :

```
64.444
```

In [47] :

```
a**2 * var_X + b**2 * var_Y + 2*a*b * cov_xy
```

Out [47] :

```
64.444
```

$$V(2X+3Y)=4V(X)+9V(Y)+12Cov(X,Y)$$

2차형 이산형 확률변수의 지표

- 상관계수

$$\rho_{XY} = \rho(X,\ Y) = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

In [48] :

```
cov_xy / np.sqrt(var_X * var_Y)
```

Out [48] :

```
0.707
```