

In [8]:

```
largest_value = max(num_friends)
print(largest_value)
```

100

In [9]:

```
smallest_value = min(num_friends)
print(smallest_value)
```

1

In [10]:

```
sorted_vaule = sorted(num_friends)
print(sorted_vaule)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8,
8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
9, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 12, 12, 13,
13, 13, 13, 14, 14, 15, 15, 15, 15, 16, 18, 18, 19, 19, 21, 21, 25, 40, 41, 49, 10
0]
```

In [11]:

```
smallest_value = sorted_vaule[0]
print(smallest_value)
```

1

In [12]:

```
second_smallest_value = sorted_vaule[1]
print(second_smallest_value)
```

1

In [13]:

```
second_largest_value = sorted_vaule[-2]
print(second_largest_value)
```

49

중심 성향 - 평균

In [14]:

```
# 만약 Python2.x라면 from __future__ import division가 필요하나 Python3.x에서는 필요없음
from __future__ import division
def mean(x):
    return sum(x)/len(x)
```

In [15]:

```
mean(num_friends)    #7.333333
```

Out[15]:

7.333333333333333

- 중간값

중심 성향 - 중간값

In [16]:

```
def median(v):  
    """finds the 'middle-most' value of v"""  
    n = len(v)  
    sorted_v = sorted(v)  
    midpoint = n // 2  
  
    if n % 2 == 1:  
        # if odd, return the middle value  
        return sorted_v[midpoint]  
    else:  
        # if even, return the average of the middle values  
        lo = midpoint - 1  
        hi = midpoint  
        return (sorted_v[lo] + sorted_v[hi]) / 2
```

In [17]:

```
median(num_friends)    #6.0
```

Out[17]:

6.0

quantile

In [18]:

```
def quantile(x, p):  
    """returns the pth-percentile value in x"""  
    p_index = int(p * len(x))  
    return sorted(x)[p_index]
```

In [19]:

```
quantile(num_friends, 0.10)    # 1
```

Out[19]:

1

In [20]:

```
quantile(num_friends, 0.25)    # 3
```

Out[20]:

3

In [21]:

```
quantile(num_friends, 0.75)    # 9
```

Out[21]:

9

최빈값 – mode

In [23]:

```
def mode(x):  
    """returns a list, might be more than one mode"""  
    counts = Counter(x)  
    max_count = max(counts.values())  
    return [x_i for x_i, count in counts.items()  
            if count == max_count]
```

```
mode(num_friends)    # 1 and 6
```

Out[23]:

[6, 1]

범위 – range

In [24]:

```
def data_range(x):  
    return max(x) - min(x)
```

In [25]:

```
data_range(num_friends)    # 99
```

Out[25]:

99

분산 - variance

In [26]:

```
from linear_algebra import sum_of_squares, dot
def de_mean(x):
    """translate x by subtracting its mean (so the result has mean 0)"""
    x_bar = mean(x)
    return [x_i - x_bar for x_i in x]

def variance(x):
    """assumes x has at least two elements"""
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations) / (n - 1)

variance(num_friends)    # 81.54
```

Out[26]:

81.54351395730716

표준 편차

In [27]:

```
import math
```

In [28]:

```
def standard_deviation(x):
    return math.sqrt(variance(x))
```

In [29]:

```
standard_deviation(num_friends)    # 9.03
```

Out[29]:

9.03014473623248

사분범위 - Interquartile range

In [30]:

```
def interquartile_range(x):
    return quantile(x, 0.75) - quantile(x, 0.25)
```

In [31]:

```
interquartile_range(num_friends)    # 6
```

Out[31]:

6

공분산

In [32]:

```
daily_minutes = [1,68.77,51.25,52.08,38.36,44.54,57.13,51.4,41.42,31.22,34.76,54.01,38.79,47.59,
49.1,27.66,41.03,36.73,48.65,28.12,46.62,35.57,32.98,35.26,07,23.77,39.73,40.57,31.65,31.21,36.3
2,20.45,21.93,26.02,27.34,23.49,46.94,30.5,33.8,24.23,21.4,27.94,32.24,40.57,25.07,19.42,22.39,1
8.42,46.96,23.72,26.41,26.97,36.76,40.32,35.02,29.47,30.2,31,38.11,38.18,36.31,21.03,30.86,36.07
,28.66,29.08,37.28,15.28,24.17,22.31,30.17,25.53,19.85,35.37,44.6,17.23,13.47,26.33,35.02,32.09,
24.81,19.33,28.77,24.26,31.98,25.73,24.86,16.28,34.51,15.23,39.72,40.8,26.06,35.76,34.76,16.13,4
4.04,18.03,19.65,32.62,35.59,39.43,14.18,35.24,40.13,41.82,35.45,36.07,43.67,24.61,20.9,21.9,18.
79,27.61,27.21,26.61,29.77,20.59,27.53,13.82,33.2,25,33.1,36.65,18.63,14.87,22.2,36.81,25.53,24.
62,26.25,18.21,28.08,19.42,29.79,32.8,35.99,28.32,27.79,35.88,29.06,36.28,14.1,36.63,37.49,26.9,
18.58,38.48,24.48,18.95,33.55,14.24,29.04,32.51,25.63,22.22,19,32.73,15.16,13.9,27.2,32.01,29.27
,33,13.74,20.42,27.32,18.23,35.35,28.48,9.08,24.62,20.12,35.26,19.92,31.02,16.49,12.16,30.7,31.2
2,34.65,13.13,27.51,33.2,31.57,14.1,33.42,17.44,10.12,24.42,9.82,23.39,30.93,15.03,21.67,31.09,3
3.29,22.61,26.89,23.48,8.38,27.81,32.35,23.84]
```

- 공분산

In [33]:

```
def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n - 1)
```

In [34]:

```
covariance(num_friends, daily_minutes)    # 22.43
```

Out[34]:

22.425435139573064

상관관계

In [35]:

```
def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y) / stdev_x / stdev_y
    else:
        return 0    # if no variation, correlation is zero
```

In [36]:

```
correlation(num_friends, daily_minutes)    # 0.25
```

Out[36]:

0.24736957366478218

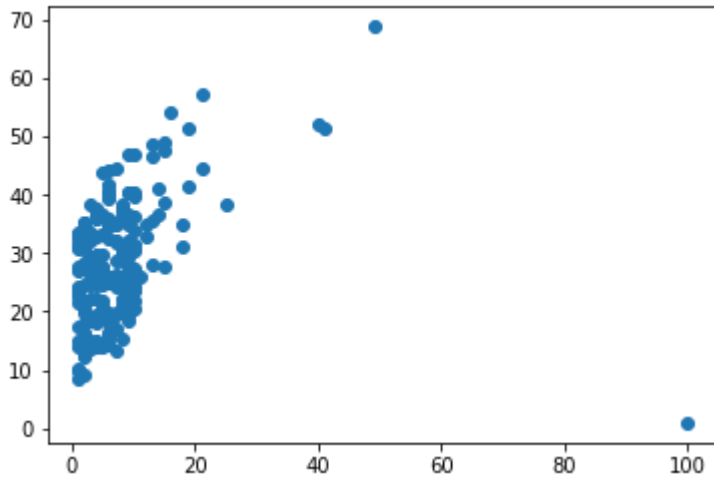
scatter plot으로 데이터 확인

In [37]:

```
import matplotlib.pyplot as plt
plt.scatter(num_friends, daily_minutes)
```

Out[37]:

<matplotlib.collections.PathCollection at 0x2a41ebe6a20>



Outlier

- 100명의 친구를 가진 사람은 outlier라고 간주해 보자.
 - 상관관계(correlation)는 outlier에 민감함.

In [38]:

```
outlier = num_friends.index(100)
num_friends_good = [x for i, x in enumerate(num_friends)
                    if i != outlier]
daily_minutes_good = [x for i, x in enumerate(daily_minutes)
                     if i != outlier]
correlation(num_friends_good, daily_minutes_good)  # 0.57
```

Out[38]:

0.5736792115665573

numpy를 이용한 평균

- mean을 이용하여 평균을 구함. function이나 method로 활용

In [40]:

```
import numpy as np
x = np.arange(10)
print(x.mean())          #4.5
print(np.mean(x))        #4.5
```

4.5

4.5

- 행렬 형식의 데이터의 평균

In [41]:

```
b = np.array([[0,1,2], [3,4,5]])
print(b.mean())
print(b.mean(0))
print(b.mean(1))
```

```
2.5
[1.5 2.5 3.5]
[1. 4.]
```

numpy를 이용한 중앙값

- np.median은 함수 형식으로 존재

In [42]:

```
x = np.random.randn(4, 5)
print(np.median(x))
print(np.median(x, 0))
print(np.median(x, 1))
```

```
-0.11775850799184819
[ 0.57075899 -0.18410954  0.76513869 -1.03706682 -0.37275286]
[-0.29482574 -0.16212368 -0.07339334  0.96782272]
```

- 여기서 np.random.randn(4, 5)는 표준정규분포를 따르는 난수로 이루어진 4×5 행렬 생성

numpy를 이용한 분산과 표준편차

- np.std와 np.var함수를 이용하여 분산과 표준편차를 계산

In [43]:

```
x = np.random.randn(4, 5)
print(np.std(x))
print(np.std(x, 0))
print(np.std(x, 1))
```

```
1.4264721406628709
[0.71179545 1.24401439 1.57108875 1.12221976 1.60395328]
[2.0911661  0.97664255 0.80911266 0.77155449]
```

- ddof 인자를 이용하여 자유도 설정 가능 (기본값 : ddof=0)

In [44]:

```
print(np.std(x, ddof=1))  #표준편차 계산시 분모를 N-1로
```

```
1.4635295361726743
```

numpy를 이용한 상관계수

- np.corrcoef(x)는 x가 2차원 행렬일 때, 각 행들간의 상관계수 행렬 계산

In [45]:

```
x = np.random.randn(3, 4)
print(np.corrcoef(x))
```

```
[[ 1.          0.20534744 -0.38129921]
 [ 0.20534744  1.          -0.24815979]
 [-0.38129921 -0.24815979  1.          ]]
```

- `np.corrcoef(x, y)`는 `x, y`가 각각 1차원 array일 때 `x`와 `y`간의 상관계수 행렬 계산

In [46]:

```
print(np.corrcoef(x[0], x[1]))
```

```
[[1.          0.20534744]
 [0.20534744  1.          ]]
```

- 공분산 행렬은 `np.cov()`를 이용하여 계산