ch.5 Statistics

단일 데이터 세트 (Describing a Single Set of Data)

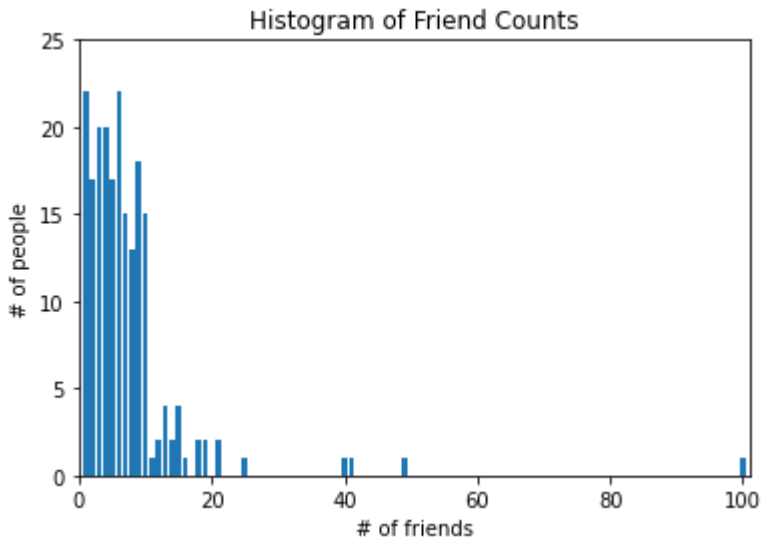
- 모금 활동의 부사장이 회원들이 친구를 얼마나 가지고 있는지에 대한 설명을 요청했습니다.

```
from collections import Counter
from linear_algebra import sum_of_squares, dot
import math
from operator import add
```

```
num_friends = [100,49,41,40,25,21,21,19,19,18,18,16,15,15,15,15,14,14,13,13,13,13,12,12,11,10,10
t = Counter(num_friends)
```

```
def make_friend_counts_histogram(plt):
    # Counter 객체를 사용하여 각 친구 수의 빈도를 계산합니다.
    friend_counts = Counter(num_friends)

    # x 축은 0부터 100까지의 정수로 설정합니다.
    xs = range(101)

    # 친구 수에 대한 빈도를 가져와서 ys 리스트에 저장합니다.
    ys = [friend_counts[x] for x in xs]

    # 막대 그래프를 그립니다.
    plt.bar(xs, ys)

    # x 축과 y 축의 범위를 설정합니다.
    plt.axis([0, 101, 0, 25])

    # 그래프의 제목을 설정합니다.
    plt.title("Histogram of Friend Counts")

    # x 축의 레이블을 설정합니다.
    plt.xlabel("# of friends")

    # y 축의 레이블을 설정합니다.
    plt.ylabel("# of people")

    # 그래프를 화면에 표시합니다.
    plt.show()
```

```
import matplotlib.pyplot as plt
make_friend_counts_histogram(plt)
```



- 아쉽게도, 이 차트는 여전히 대화에 쉽게 포함하기 어렵습니다.
- 그러므로 몇 가지 통계를 생성하기 시작합니다.
- 아마도 가장 간단한 통계는 데이터 포인트의 수입니다.

min, max, smallest, largest

```
num_points = len(num_friends)               # 204

largest_value = max(num_friends)            # 100
smallest_value = min(num_friends)           # 1

sorted_values = sorted(num_friends)
smallest_value = sorted_values[0]           # 1
second_smallest_value = sorted_values[1]    # 1
second_largest_value = sorted_values[-2]    # 49

print("num_points", len(num_friends))
print("largest value", max(num_friends))
print("smallest value", min(num_friends))
print("second_smallest_value", sorted_values[1])
print("second_largest_value", sorted_values[-2])
```

```
num_points 204
largest value 100
smallest value 1
second_smallest_value 1
second_largest_value 49
```

💡 위 코드는 데이터 포인트의 수를 계산하고, 최댓값과 최솟값을 찾으며, 정렬된 값에서 두 번째로 작은 값과 두 번째로 큰 값을 찾는 등의 통계를 생성합니다. 이를 출력하여 확인할 수 있습니다.

Numpy Version

```
import numpy as np

num_friends = np.array(num_friends)

num_points = num_friends.shape[0]               # 204

largest_value = np.max(num_friends)             # 100
```

```
smallest_value = np.min(num_friends)           # 1

sorted_values = np.sort(num_friends)
smallest_value = sorted_values[0]           # 1
second_smallest_value = sorted_values[1]    # 1
second_largest_value = sorted_values[-2]    # 49

print("num_points", len(num_friends))
print("largest value", max(num_friends))
print("smallest value", min(num_friends))
print("second_smallest_value", sorted_values[1])
print("second_largest_value", sorted_values[-2])

num_friends = list(num_friends)
```

```
num_points 204
largest value 100
smallest value 1
second_smallest_value 1
second_largest_value 49
```

💡 이 코드는 NumPy를 사용하여 데이터를 배열로 변환하고, 배열의 형태를 확인하며, 최댓값과 최솟값을 계산하고, 배열을 정렬하여 그 중 작은 값과 큰 값 등을 찾습니다. 그리고 출력하여 확인할 수 있습니다. 마지막으로, 배열을 다시 리스트로 변환합니다.

중심 경향성(Central Tendencies)

- **median은 데이터의 모든 값에 의존하지 않습니다.**
- **예를 들어, 가장 큰 점수를 더 크게 만들거나 (또는 가장 작은 점수를 더 작게 만들면), 중간 점수는 변경되지 않습니다.**
- **mean은 데이터의 이상치에 매우 민감합니다.**
- **예를 들어, 우리 친구가 100명이 아닌 200명의 친구를 가졌다면, mean은 7.82로 상승하지만, median은 그대로 유지됩니다.**
- **예를 들어, 1980년대 중반에 노스캐롤라이나 대학에서 가장 높은 평균 초봉을 가진 전공은 지리학이었다고 자주 언급됩니다. 이는 주로 NBA 스타이자 이상치인 마이클 조던 때문이었습니다.**
- **median의 일반화된 형태는 quantile로, 특정 백분위수의 데이터가 그 값보다 작다는 것을 나타냅니다. (median은 데이터의 50%가 그 값보다 작다는 것을 나타냅니다.)**
- **mode: 가장 흔한 값**

💡 중심경향성 → Outlier(결측치)에 있으면 누가 민감하게 반응? (Mean: 평균값이 강하게 반응)

mean(평균), median(중간), quantile(제일 작은값 ~ 제일 큰값), mode(최빈값)

```
def mean(x):
    # 리스트 x의 평균을 계산하여 반환합니다.
    return sum(x) / len(x)

def median(v):
    """v의 '중앙값'을 찾습니다."""
    n = len(v)
```

```
    sorted_v = sorted(v)  # v를 정렬합니다.
    midpoint = n // 2  # 중앙 지점을 계산합니다.

    if n % 2 == 1:
        # 만약 리스트의 길이가 홀수라면, 중앙의 값을 반환합니다.
        return sorted_v[midpoint]
    else:
        # 짝수라면, 중앙에 위치한 두 값의 평균을 반환합니다.
        lo = midpoint - 1
        hi = midpoint
        return (sorted_v[lo] + sorted_v[hi]) / 2

def quantile(x, p):
    """x의 p번째 백분위수 값을 반환합니다."""
    p_index = int(p * len(x))  # 백분위수에 해당하는 인덱스를 계산합니다.
    return sorted(x)[p_index]  # x를 정렬하고, 해당 인덱스의 값을 반환합니다.

def mode(x):
    """리스트에서 가장 많이 나타나는 값을 반환합니다. 여러 개일 경우 리스트로 반환됩니다."""
    from collections import Counter  # Counter를 사용하여 각 값의 빈도를 계산합니다.
    counts = Counter(x)
    max_count = max(counts.values())  # 가장 높은 빈도를 찾습니다.
    return [x_i for x_i, count in counts.items()
            if count == max_count]  # 가장 높은 빈도를 가진 값들을 리스트로 반환합니다.
```

💡 **이 코드는 데이터 집합에 대한 평균(mean), 중앙값(median), 백분위수(quantile), 최빈값(mode)을 계산하는 함수들을 정의합니다. 각 함수는 주어진 리스트의 통계적 특성을 계산하여 반환합니다.**

```
print("mean(num_friends)", mean(num_friends))
print("median(num_friends)", median(num_friends))
print("quantile(num_friends, 0.10)", quantile(num_friends, 0.10))
print("quantile(num_friends, 0.25)", quantile(num_friends, 0.25))
print("quantile(num_friends, 0.75)", quantile(num_friends, 0.75))
print("quantile(num_friends, 0.90)", quantile(num_friends, 0.90))
print("mode(num_friends)", mode(num_friends))
```

```
mean(num_friends) 7.333333333333333
median(num_friends) 6.0
quantile(num_friends, 0.10) 1
quantile(num_friends, 0.25) 3
quantile(num_friends, 0.75) 9
quantile(num_friends, 0.90) 13
mode(num_friends) [6, 1]
```

```
num_friends_2 = num_friends[:] #
```

💡 **이 `num_friends` 리스트의 모든 요소를 복사하여 `num_friends_2` 라는 새로운 리스트에 할당합니다. `[:]` 는 리스트의 시작부터 끝까지 모든 요소를 선택하는 슬라이싱(slicing) 연산자입니다.**

따라서, 이는 `num_friends` 리스트의 깊은 복사(deep copy)를 생성하는 방법 중 하나

```
mean(num_friends_2) # 평균

# Result: 7.333333333333333
```

```
median(num_friends_2) # 중간값

# Result: 6.0
```

```
num_friends_2[0] = 200
```

💡 `num_friends_2[0] = 200` 이 코드는 `num_friends_2` 리스트의 첫 번째 요소(인덱스 0에 위치한 요소)의 값을 200으로 변경합니다.

만약 `num_friends_2` 리스트가 `[10, 20, 30, 40]`와 같이 정의되었다면, 이 코드를 실행한 후 `num_friends_2`의 내용은 `[200, 20, 30, 40]`으로 변경됩니다.

이는 리스트 내 특정 위치에 있는 값을 새로운 값으로 업데이트하고 싶을 때 사용하는 방법입니다.

```
75900 * 1300 // 10000 # 75900 x 1300후 나누기 10000
# Result: 9867
```

Numpy Version

```
import numpy as np

# Numpy 라이브러리를 사용하여 num_friends 데이터셋에 대한 기초 통계를 계산하는 코드입니다.
# np.quantile 함수는 Numpy 버전 1.16부터 사용 가능합니다.
# np.percentile 함수는 비율 대신 퍼센트를 사용합니다.

# 데이터셋의 평균을 계산합니다.
print("mean(num_friends)", np.mean(num_friends))

# 데이터셋의 중앙값을 계산합니다. 데이터가 홀수라면 중앙의 값, 짝수라면 중앙에 있는 두 값의 평균을 반환합니다.
print("median(num_friends)", np.median(num_friends))

# 데이터셋의 10%에 해당하는 분위수를 계산합니다. 즉, 데이터를 오름차순으로 정렬했을 때 하위 10%를 차지하는 값입니다.
print("quantile(num_friends, 0.10)", np.percentile(num_friends, 10))

# 데이터셋의 25%에 해당하는 분위수를 계산합니다. 이를 1사분위수 또는 하위 25% 경계값이라고 합니다.
print("quantile(num_friends, 0.25)", np.percentile(num_friends, 25))

# 데이터셋의 75%에 해당하는 분위수를 계산합니다. 이를 3사분위수 또는 상위 25% 경계값이라고 합니다.
print("quantile(num_friends, 0.75)", np.percentile(num_friends, 75))

# 데이터셋의 90%에 해당하는 분위수를 계산합니다. 즉, 데이터를 오름차순으로 정렬했을 때 하위 90%를 차지하는 값입니다.
print("quantile(num_friends, 0.90)", np.percentile(num_friends, 90))

# numpy를 사용하여 데이터셋의 최빈값(가장 자주 나타나는 값)을 계산하는 것은 별도의 과제로 남겨둡니다.
```

```
mean(num_friends) 7.333333333333333
median(num_friends) 6.0
quantile(num_friends, 0.10) 1.0
```

```
quantile(num_friends, 0.25) 3.0
quantile(num_friends, 0.75) 9.0
quantile(num_friends, 0.90) 13.0
```

```
import numpy as np

np.percentile(np.array([0,1,2,3,4,5,100]), [75, 50, 25])

# Result: array([4.5, 3. , 1.5])
```

💡 **Numpy의 `np.percentile` 함수를 사용하여 주어진 배열 `[0, 1, 2, 3, 4, 5, 100]`의 75번째, 50번째(중앙값), 그리고 25번째 백분위수를 계산합니다. 이 함수는 데이터를 정렬한 후 지정된 백분위수에 해당하는 데이터 포인트 값을 반환**

```
import numpy as np
from scipy import stats

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

# stats.mode 함수를 사용하여 배열의 최빈값(mode)을 계산합니다.
# 이 함수는 최빈값과 그 빈도를 모두 반환합니다.
mode = stats.mode(array)

# 최빈값만 출력합니다. mode[0]은 최빈값을, mode[1]은 해당 최빈값의 빈도를 나타냅니다.
# 여기서는 최빈값만 필요하기 때문에 mode[0]만을 출력합니다.
print(mode[0])

# Result: [2]
```

💡 **이 코드는 `[1, 2, 2, 3, 4, 4, 5]` 배열에서 가장 자주 나타나는 값을 찾고 그 값을 출력합니다. `stats.mode` 함수는 최빈값과 해당 최빈값의 빈도를 반환하는데, 여기서는 최빈값인 `2`만을 출력합니다. 만약 배열에서 여러 개의 최빈값이 동일한 빈도로 나타난다면, `stats.mode` 함수는 가장 작은 값을 최빈값으로 반환**

```
import numpy as np

# 배열을 생성합니다.
array = np.array([1,2,2,3,4,4,5])

# np.unique를 사용하여 배열 내의 고유한 값들(vals)과 각 값의 등장 횟수(counts)를 반환받습니다.
vals, counts = np.unique(array, return_counts=True)

# counts 배열에서 가장 큰 값을 찾습니다. 이 값은 배열 내에서 가장 많이 등장하는 원소의 등장 횟수입니다.
max_val = np.max(counts)

# counts == max_val은 counts 배열에서 max_val과 같은 값을 가지는 원소의 위치를 불리언 배열로 반환합니다.
# 이 불리언 배열을 이용하여 vals 배열에서 조건을 만족하는 원소들만을 선택합니다.
# 결과적으로, 가장 많이 등장하는 원소들이 반환됩니다. 만약 최빈값이 여러 개라면, 그 모든 값을 반환합니다.
mode_vals = vals[counts == max_val]
```

```
# 최빈값(들)을 출력합니다.
print(mode_vals)

# Result: array([2, 4])
```

💡 **이 코드는 배열 `[1,2,2,3,4,4,5]`에서 `2`와 `4`가 각각 두 번씩 등장하여 가장 많이 등장하는 값임을 확인하고, 두 값을 모두 반환합니다. 이 방법은 여러 최빈값을 찾을 때 유용하며, SciPy 라이브러리를 사용하지 않고 순수 Numpy로만 구현**

Dispersion(분산)

- Dispersion(분산)은 데이터가 얼마나 퍼져 있는지 측정하는 것을 말합니다.
- Range(범위)는 가장 큰 요소와 가장 작은 요소의 차이일 뿐입니다 → 최소, 최대값의 차이
- Variance
- **Standard deviation(**표준편차)
- Range(범위)와 **Standard deviation(**표준편차) 모두 Outlier 문제가 동일합니다
- 사분위수_범위(**Interquartile_range)**: 75번째 백분위수 값과 25번째 백분위수 값의 차이

```
# "range"라는 이름은 Python에서 이미 다른 의미로 사용되므로 다른 이름을 사용합니다.
def data_range(x):
    # 데이터 집합 x의 최대값과 최소값의 차이를 반환합니다.
    return max(x) - min(x)

def de_mean(x):
    # 데이터 집합 x의 각 원소에서 x의 평균을 뺀 값을 반환합니다.
    # 그 결과로 반환된 데이터 집합의 평균은 0이 됩니다.
    x_bar = sum(x) / len(x)
    return [x_i - x_bar for x_i in x]

def variance(x):
    # 데이터 집합 x의 분산을 계산합니다.
    # 분산은 데이터가 평균으로부터 얼마나 퍼져 있는지를 측정합니다.
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations) / (n - 1)

def standard_deviation(x):
    # 데이터 집합 x의 표준편차를 계산합니다.
    return math.sqrt(variance(x))

def interquartile_range(x):
    # 데이터 집합 x의 사분위범위(IQR)를 계산합니다.
    return quantile(x, 0.75) - quantile(x, 0.25)
```

```
print("data_range(num_friends)", data_range(num_friends))
print("variance(num_friends)", variance(num_friends))
print("standard_deviation(num_friends)", standard_deviation(num_friends))
print("interquartile_range(num_friends)", interquartile_range(num_friends))
```

```
data_range(num_friends) 99
variance(num_friends) 81.54351395730716
standard_deviation(num_friends) 9.03014473623248
interquartile_range(num_friends) 6
```

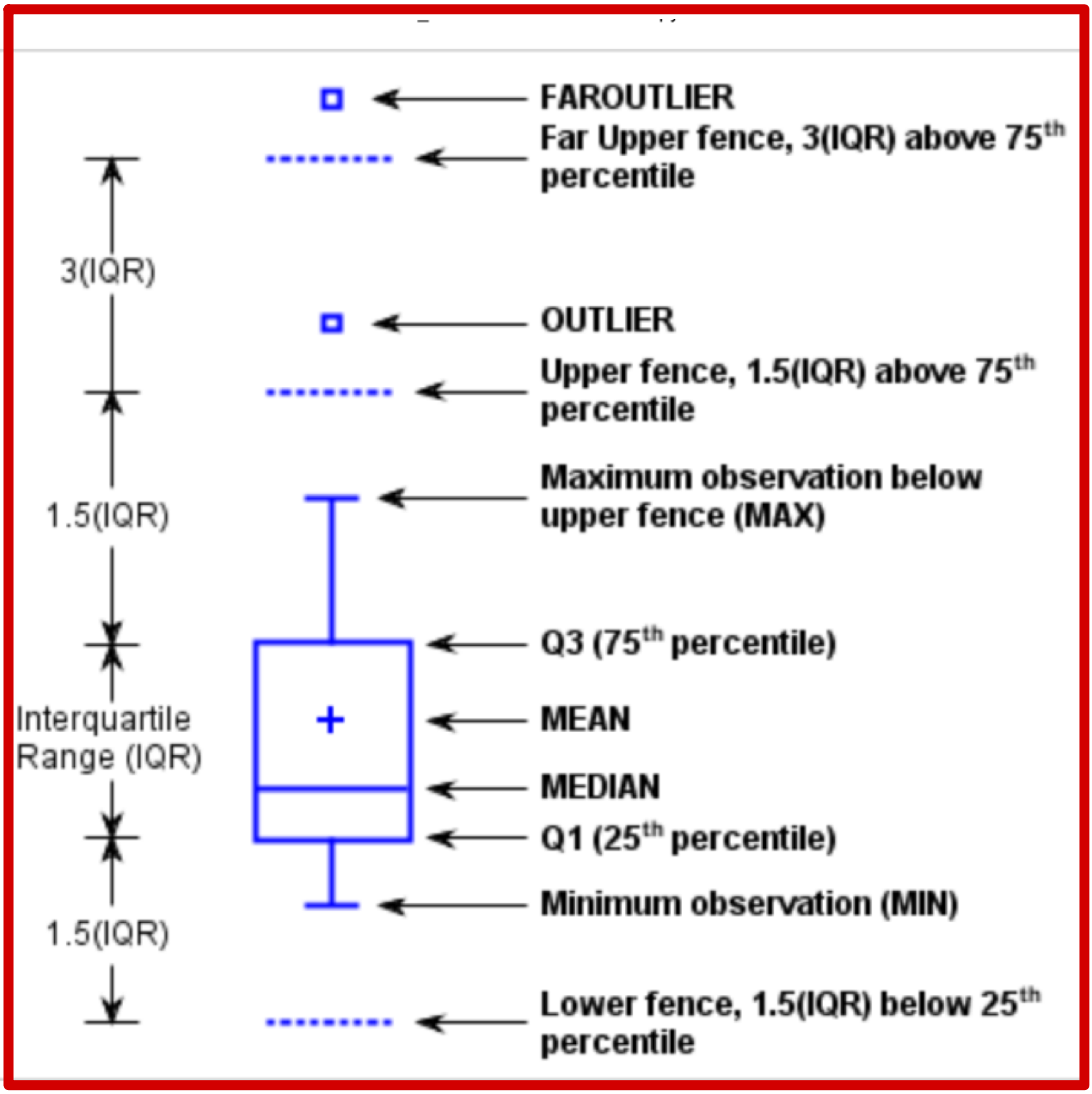
Numpy Version

```
num_friends = np.array(num_friends)  # num_friends 데이터를 NumPy 배열로 변환

# 데이터의 최댓값과 최솟값의 차이를 계산 (ptp - 분산)
print("data_range(num_friends)", np.ptp(num_friends))

# 직접 최대값과 최소값의 차이를 계산하여 데이터 범위를 구함
print("data_range(num_friends)", np.max(num_friends) - np.min(num_friends))

 # 데이터의 분산을 계산, ddof=1로 설정하여 표본 분산을 구함 (var - 분산)
print("variance(num_friends)", np.var(num_friends, ddof=1))

# 데이터의 표준편차를 계산, ddof=1로 설정하여 표본 표준편차를 구함
print("standard_deviation(num_friends)", np.std(num_friends, ddof=1))

# 데이터의 75번째 백분위수와 25번째 백분위수를 계산
q75, q25 = np.percentile(num_friends, [75, 25])

# 백분위수 범위, 즉 IQR을 계산
print("interquartile_range(num_friends)", q75 - q25)

# NumPy 배열을 다시 리스트로 변환
num_friends = list(num_friends)
```

```
data_range(num_friends) 99
data_range(num_friends) 99
variance(num_friends) 81.54351395730707
standard_deviation(num_friends) 9.030144736232474
interquartile_range(num_friends) 6.0
```

Boxplot

- 그것은 매우 설명력이 있습니다
- 데이터 요약을 보여줍니다




```
import numpy as np
p75, p50, p25 = np.percentile(np.array([0,1,2,3,4,5,100]), [75, 50, 25])
p75, p50, p25
```

💡 **이 코드는 주어진 배열 `[0,1,2,3,4,5,100]`에 대해 75번째 백분위수(p75), 50번째 백분위수(중앙값, p50), 25번째 백분위수(p25)를 계산합니다.**

`np.percentile` 함수는 지정된 백분위수에 해당하는 값을 반환합니다.

위 배열에서,

25번째 백분위수(p25)는 1과 2 사이의 값입니다.50번째 백분위수(p50), 즉 중앙값은 3입니다.75번째 백분위수(p75)는 4와 5 사이의 값입니다.

```
(4.5, 3.0, 1.5)
```

```
np.percentile(np.array([0,1,2,3]),90)
```

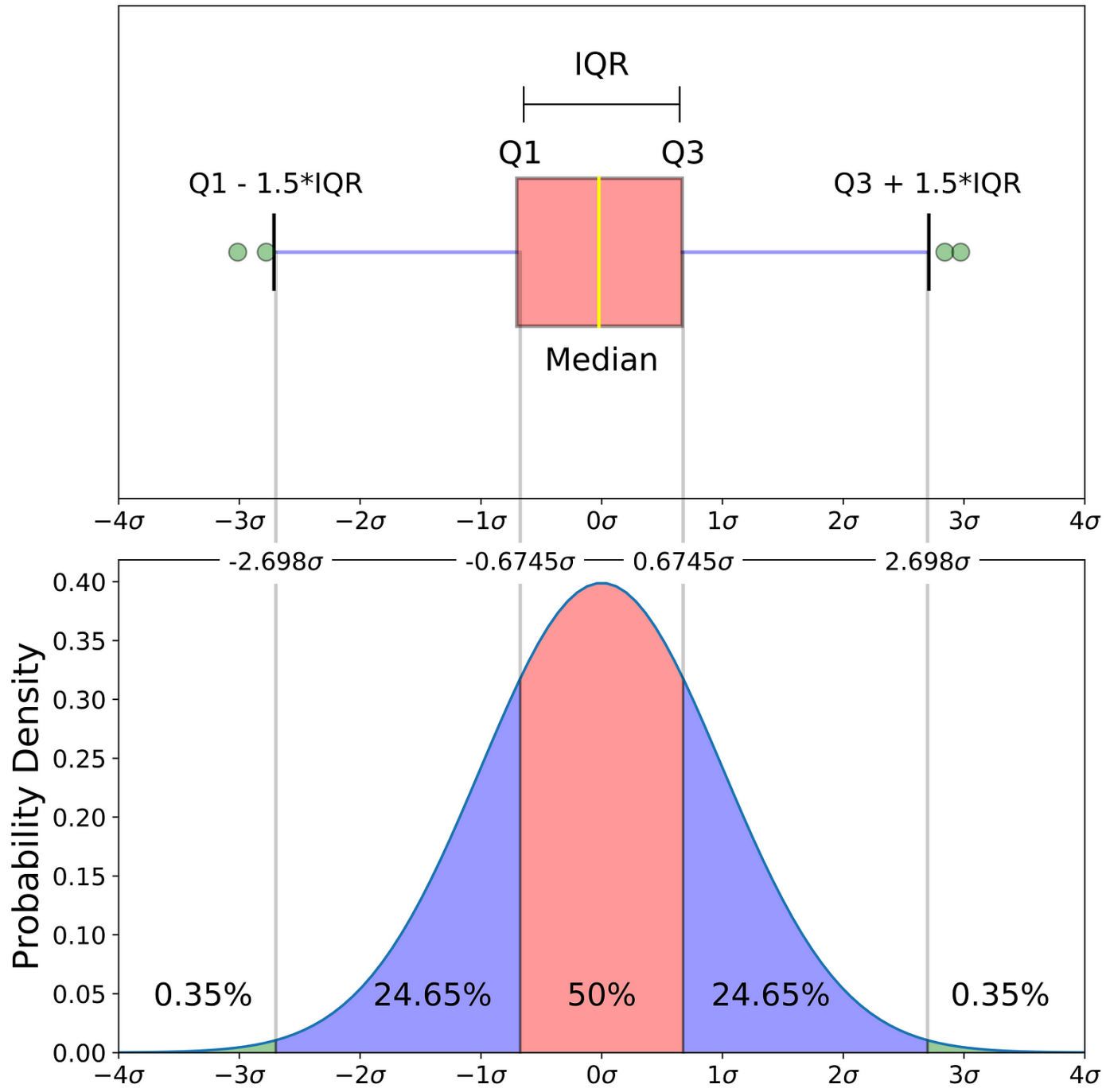
```
# Result: 2.7
```

💡 **NumPy의 `np.percentile` 함수를 사용하여 주어진 배열 [0,1,2,3]의 90번째 백분위수를 계산하면, 배열의 데이터 분포를 고려하여 해당 백분위수에 해당하는 값을 선형 보간(linear interpolation) 방식으로 찾습니다.**

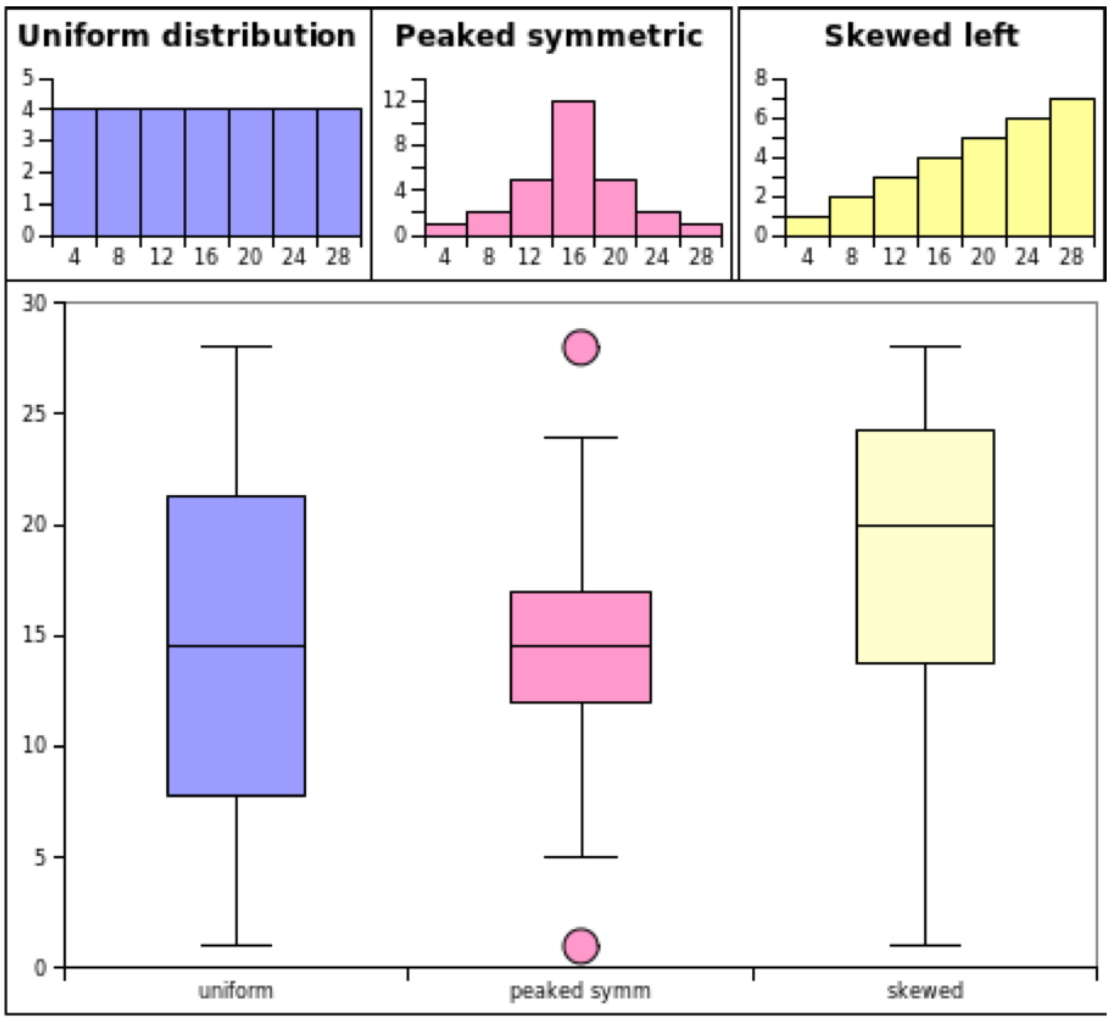
이 배열에서 90번째 백분위수는 2와 3 사이에 위치합니다. 구체적으로, 90%에 해당하는 위치는 배열의 끝에서 두 번째 값과 마지막 값 사이에 있으므로, 2와 3 사이의 값입니다.

선형 보간을 사용하여 계산하면, 90번째 백분위수는 대략적으로 2.7이 됩니다. 이는 2와 3 사이의 거리(1)의 90%에 해당하는 위치를 의미합니다.

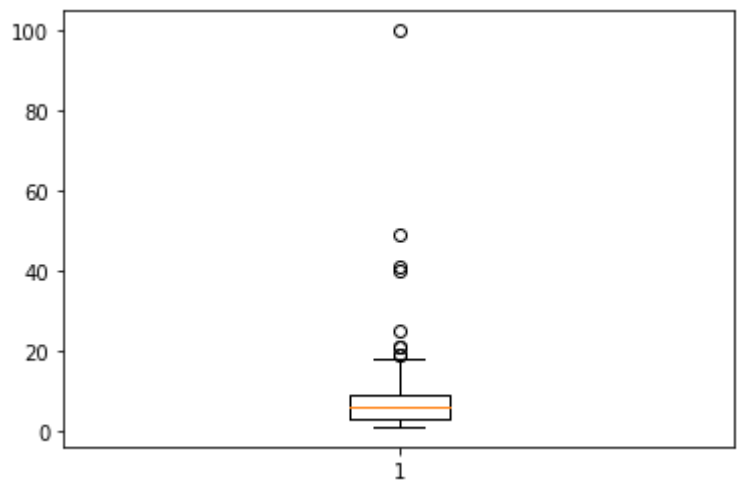
- In case of normal distribution



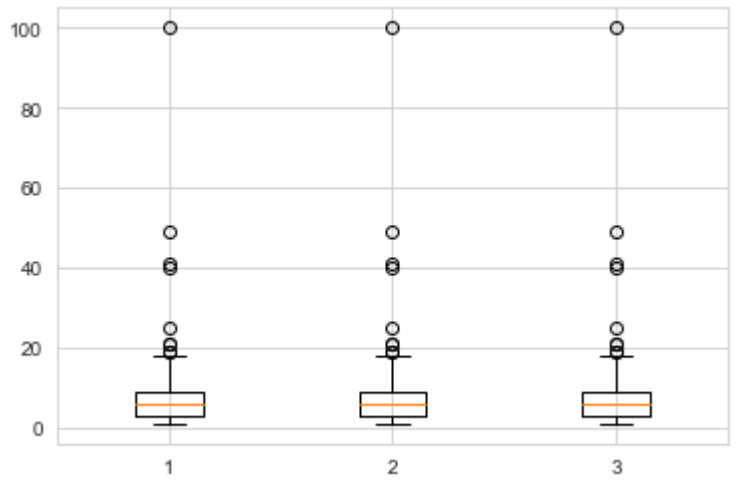
- uniform vs peaked vs skewed (균일 대 정점 대 비대칭)



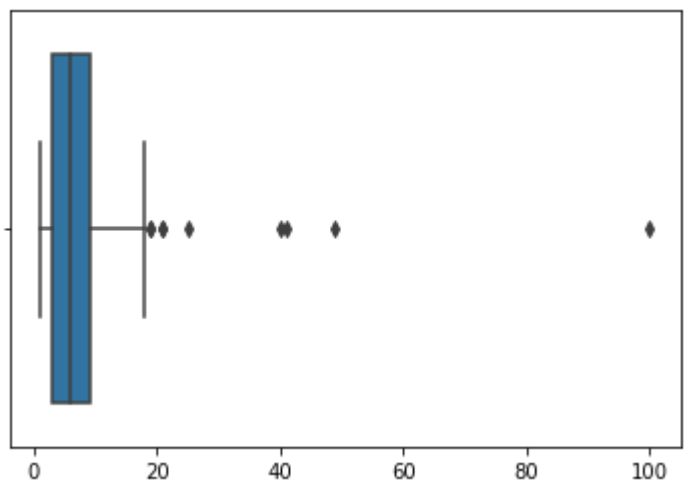

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

plt.boxplot(num_friends) # num_friends 데이터셋에 대한 박스 플롯을 생성
plt.show()
```



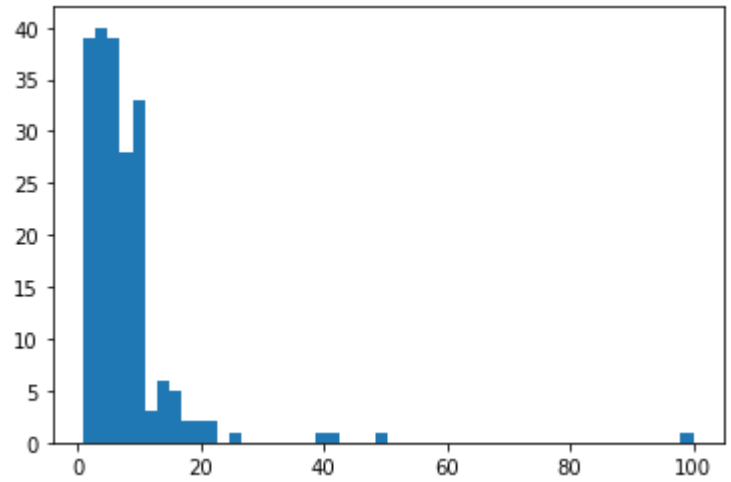
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

# 데이터셋을 포함하는 배열을 만들어 이 배열의 boxplot 그림을 생성
# 같은 데이터셋을 세 번 반복하여 배열로 만들었습니다. 따라서 세 개의 boxplot 그림이 생김
plt.boxplot([num_friends, num_friends, num_friends])
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.boxplot(num_friends) # Seaborn을 사용하여 박스플롯 생성
plt.show()
```

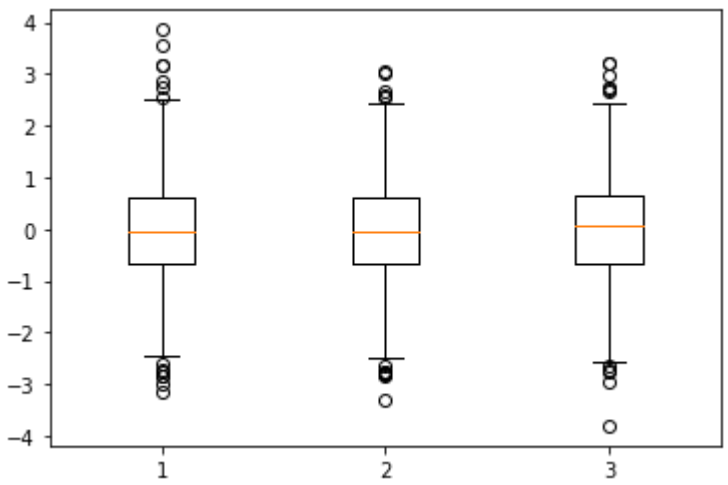



```
# bins=50 인자는 데이터를 50개의 구간으로 나누어 각 구간에 속하는 데이터의 개수를 막대로 표현
plt.hist(num_friends, bins=50)
plt.show()
```

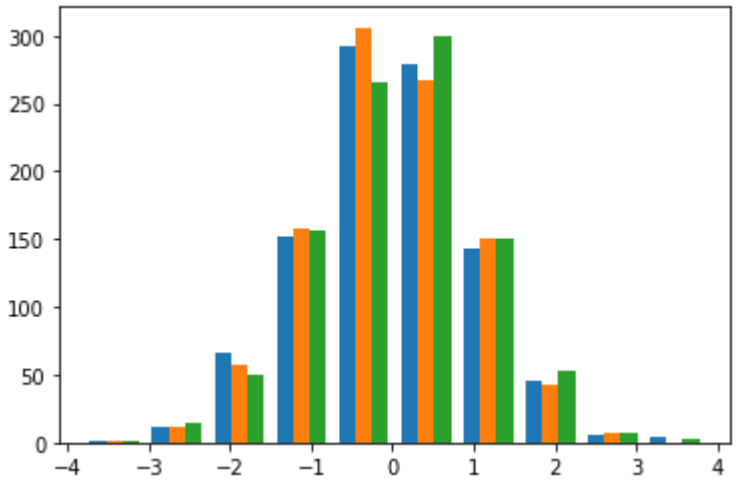



```
data = np.random.randn(1000,3)
plt.boxplot(data)
plt.show()
```

💡 **평균 0, 표준편차 1을 가진 정규 분포에서 1000x3 배열을 생성합니다. 이 배열은 1000개의 샘플을 각각 가진 3개의 독립적인 데이터 세트**




```
plt.hist(data, bins=10)
plt.show()
```



💡 **데이터의 분포를 10개의 구간으로 나누어 각 구간에 속하는 샘플의 수를 막대로 표현한 히스토그램을 생성합니다. 이 코드에서는 3개의 데이터 세트에 대해 히스토그램이 생성되며, 각각의 데이터 세트는 별도의 막대 그룹으로 표시**

💡 만약 모집단의 평균(μ)을 알고 있다면, 표준편차(σ)는 다음과 같이 계산됩니다.

$$\sigma = \sqrt{\frac{\sum (x-\mu)^2}{n}}$$

여기서 x 는 각 데이터 포인트를 나타내며, μ 는 모집단의 평균이고, n 은 데이터 포인트의 수입니다.

이 공식은 모든 데이터 포인트와 평균 간의 편차를 제곱한 값을 모두 더한 후, 이를 데이터 포인트의 수로 나누어 구합니다.

💡 모집단의 평균을 알지 못하는 경우, 표본 평균($\bar{x}$)을 사용하여 표준편차(σ)를 계산할 수 있습니다

$$\sigma = \sqrt{\frac{\sum (x-\bar{x})^2}{n-1}}$$

여기서 x는 각 데이터 포인트를 나타내며, $\bar{x}$는 표본의 평균이고, n은 데이터 포인트의 수입니다.

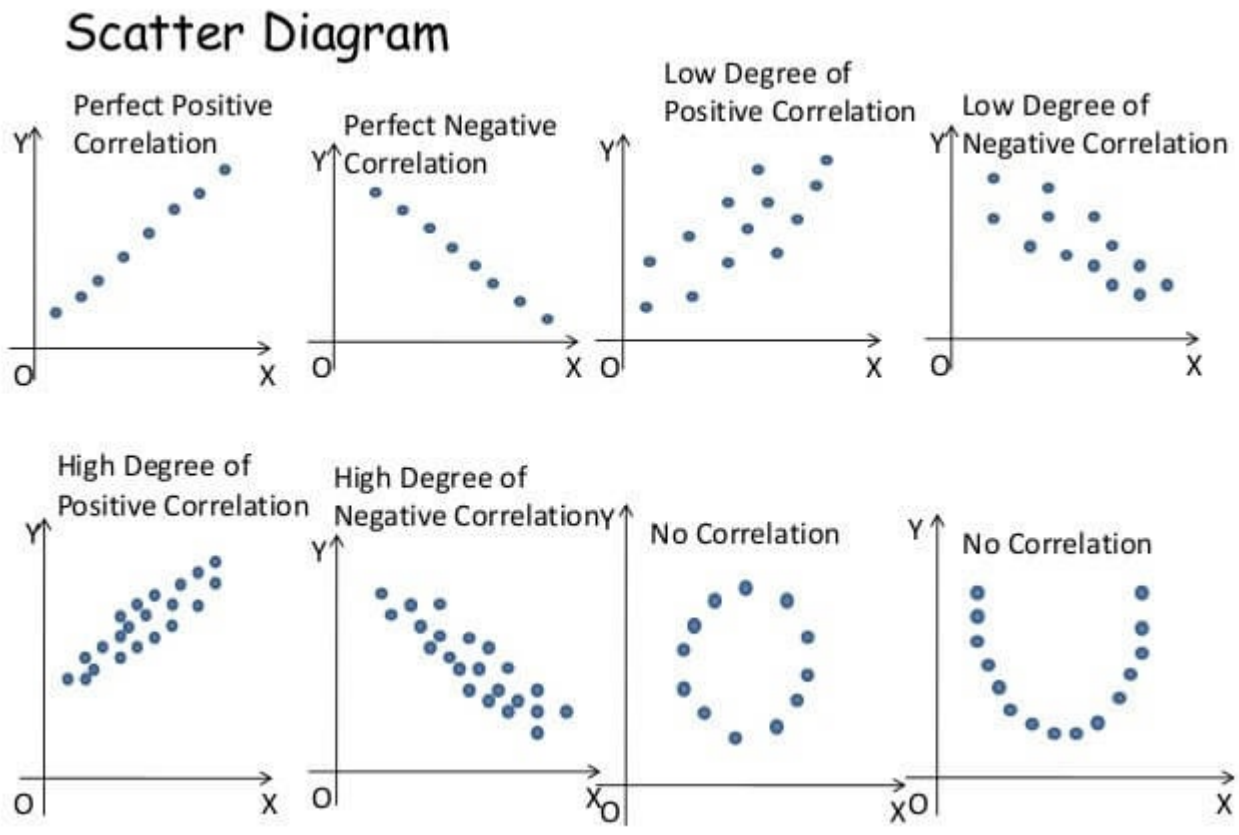
이 공식은 모든 데이터 포인트와 표본 평균 간의 편차를 제곱한 값을 모두 더한 후, 이를 데이터 포인트의 수에서 1을 뺀 값으로 나누어 구합니다.

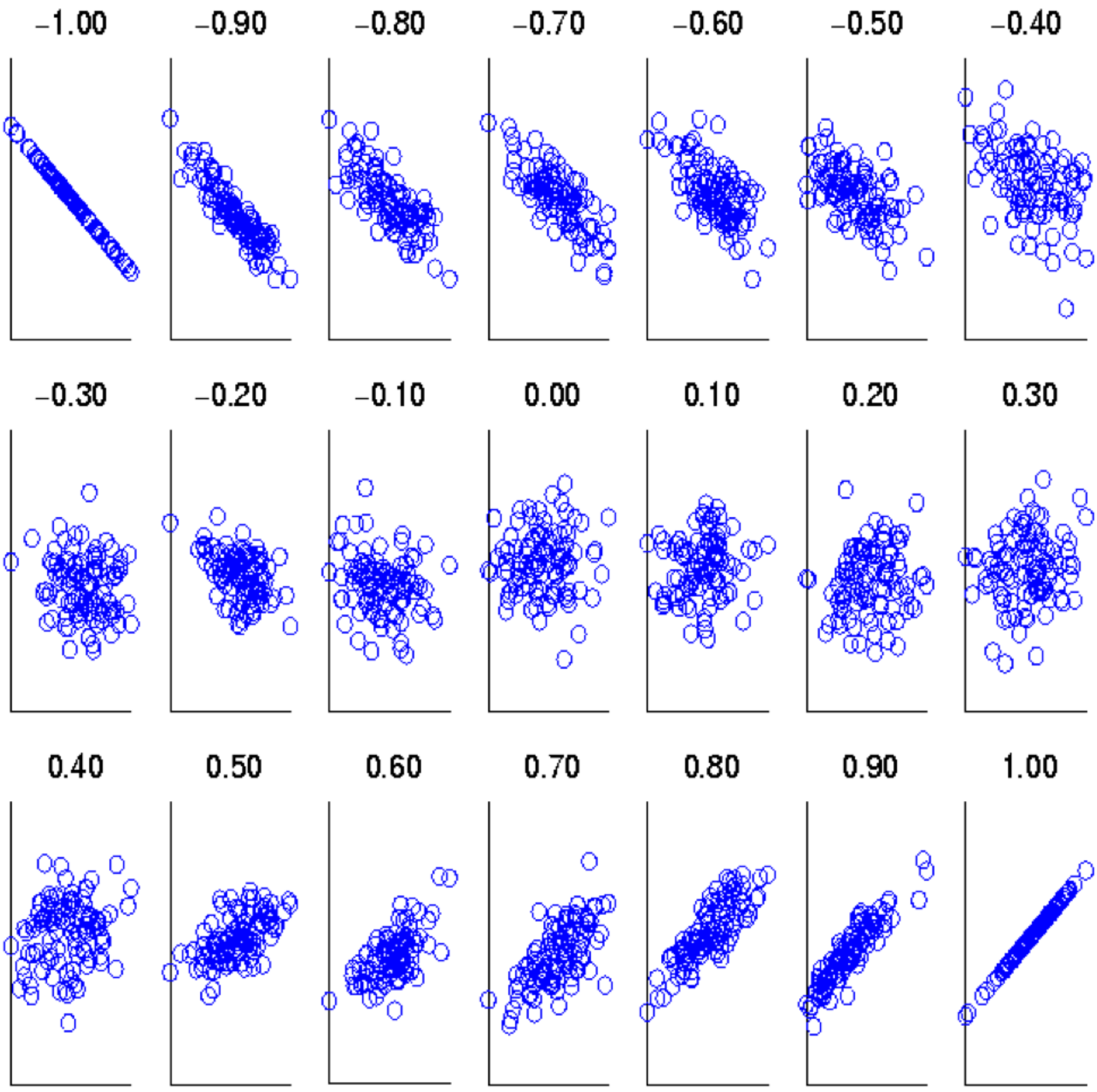
```
import numpy as np
print(np.std([1,2,3]))                # 모집단 표준편차
print(np.std([1,2,3], ddof=1))        # 표본 표준편차, ddof=1을 설정. 표본 표준편차를 계산
print(standard_deviation([1,2,3])) # 이 줄은 오류를 발생시킴
```

```
0.816496580927726
1.0
1.0
```

Correlation (상관관계)

- DataSciences의 성장 담당 부사장은 사람들이 사이트에서 보내는 시간이 사이트에 있는 친구의 수와 관련이 있다는 이론을 가지고 있으며, 그녀는 이를 확인해 줄 것을 요청했습니다.
- 이 두 지표(metrics) 간의 관계를 조사하고자 합니다.
- Variance(분산)은 단일 변수가 평균에서 벗어나는 방법을 측정하는 반면 covariance(공분산)은 두 변수가 평균에서 함께 변하는 방법을 측정합니다.
- 정규화되지 않는 한 "큰" covariance(공분산)이 무엇인지 해석하기가 어렵습니다.
- 상관관계: covariance(공분산)을 두 변수의 standard deviations(표준 편차)로 나눈 값
- correlation(상관 관계)는 단위가 없으며 항상 -1(perfect anti-correlation - 완벽한 반상관)과 1(perfect correlation - 완벽한 상관) 사이에 있습니다.






```
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,
# daily_minutes와 다른 데이터 세트 간의 공분산 및 상관관계 계산을 위한 코드

def covariance(x, y):
    n = len(x)  # x의 길이, 즉 데이터 포인트의 수
    return dot(de_mean(x), de_mean(y)) / (n - 1)  # 공분산 계산

def correlation(x, y):
    stdev_x = standard_deviation(x)  # x의 표준편차 계산
    stdev_y = standard_deviation(y)  # y의 표준편차 계산
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y) / stdev_x / stdev_y  # 상관관계 계산
    else:
        return 0  # 변동성이 없으면 상관관계는 0
```

💡 `de_mean(x)` : 데이터 집합 `x` 에서 각 데이터 포인트의 값에서 평균 값을 빼서, 평균이 0이 되도록 조정한 새로운 데이터 집합을 생성합니다. 이는 `x` 데이터의 중심을 원점으로 이동시키는 효과가 있습니다.

`dot(a, b)` : 두 데이터 집합 `a` 와 `b` 에서 각각 대응하는 데이터 포인트들의 곱셈 결과를 모두 더하는 연산입니다. 즉, `a[0]*b[0] + a[1]*b[1] + ... + a[n-1]*b[n-1]` 의 합을 계산합니다. 이는 두 데이터 집합 사이의 상호 작용을 수치화하는 과정입니다.

`n` : 데이터의 개수를 의미합니다. 여기서는 `x` 또는 `y` 데이터 집합의 길이입니다.

`/ (n - 1)` : 최종적으로 더해진 값을 `n - 1` 로 나누어 줍니다. 이는 표본 공분산을 계산할 때 사용되는 보정 공식의 일부로, 데이터 집합의 크기가 작을 때 발생할 수 있는 편향을 조정하기 위해 사용됩니다.

```
print(covariance(num_friends, daily_minutes))  # 22.43
print(correlation(num_friends, daily_minutes)) # 0.25

# we may conclude that two variables are less correlated
```

```
22.425435139573064
0.24736957366478218
```

```
plt.scatter(num_friends, daily_minutes)
plt.xlabel('num_friends')
plt.ylabel('daily_minutes')
plt.title('corr = {:.2}'.format(correlation(num_friends, daily_minutes)))
plt.show()
```

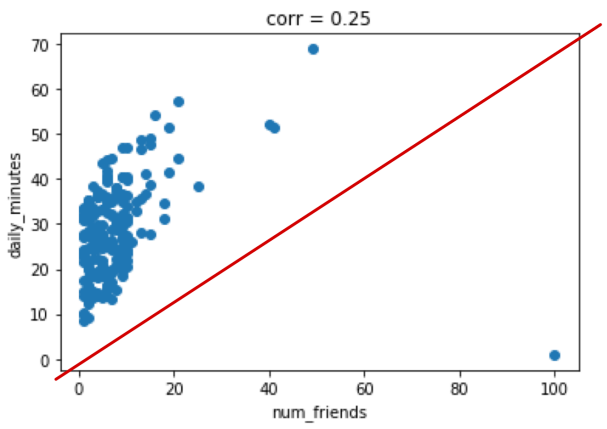
💡 `plt.scatter(num_friends, daily_minutes)` : `num_friends` 를 x축 값으로, `daily_minutes` 를 y축 값으로 하는 산점도를 그립니다.

`plt.xlabel('num_friends')` : x축에 'num_friends'라는 레이블을 붙입니다.

`plt.ylabel('daily_minutes')` : y축에 'daily_minutes'라는 레이블을 붙입니다.

`plt.title('corr = {:.2}'.format(correlation(num_friends, daily_minutes)))` : 상관관계 값을 계산하여 제목에 포맷을 적용한 후 'corr = ' 뒤에 표시합니다.

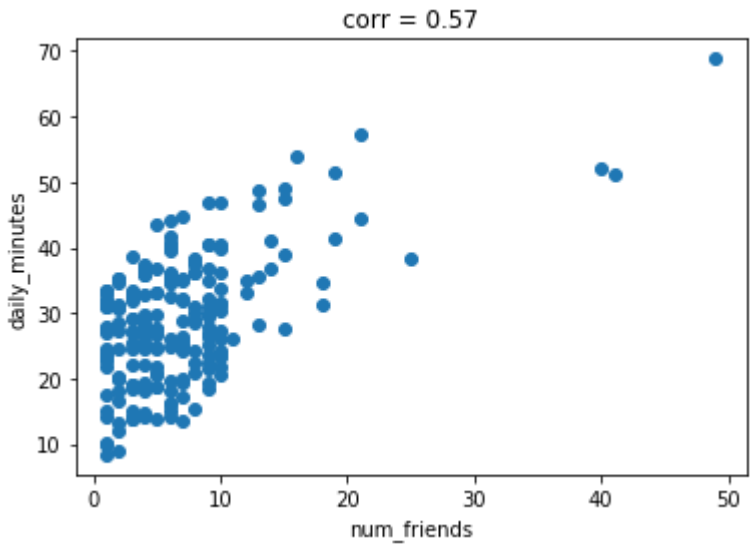
`{:.2}` 는 상관관계 값을 소수점 두 자리까지 표시하도록 합니다. `plt.show()` : 그래프를 화면에 표시합니다.



- Index(지수)가 0인 outliers(이상치-결측치) 제거
- outlier(이상치-결측치)_는 garbage인 데이터베이스의 테스트 데이터임이 밝혀졌습니다
- Recall "GIGO: Garbage in, garbage out"

```
import matplotlib.pyplot as plt

# 이상치를 제외한 num_friends와 daily_minutes 데이터로 산점도를 그립니다.
```

```
plt.scatter(num_friends[1:], daily_minutes[1:])

# x축에 'num_friends' 라벨을 추가합니다.
plt.xlabel('num_friends')
# y축에 'daily_minutes' 라벨을 추가합니다.
plt.ylabel('daily_minutes')

# 제목에 이상치를 제외한 데이터를 기반으로 계산된 상관계수를 포함시킵니다.
# {:.2}는 상관계수를 소수점 두 자리까지 표시하도록 합니다.
plt.title('corr = {:.2}'.format(correlation(num_friends[1:], daily_minutes[1:])))

plt.show()
```



Correlation (상관계수)

- 정확히 –1. 완벽한 내리막 (negative) 선형 관계
- 0.70. 강한 내리막 (negative) 선형 관계
- 0.50. 보통의 내리막 (negative) 관계
- 0.30. 약한 내리막 (negative) 선형 관계
- 0. 선형 관계 없음
- +0.30. 약한 오르막 (positive) 선형 관계
- +0.50. 보통의 오르막 (positive) 관계
- +0.70. 강한 오르막 (positive) 선형 관계
- 정확히 +1. 완벽한 오르막 (positive) 선형 관계

Correlation (상관계수) vs Cosine 유사성

💡 **corr(x, y) = cosine(x - x̄, y - ȳ) → 상관계수와 코사인 유사도 사이의 관계를 나타내는 수식**

- **x - x̄, y - ȳ는 각각** x와 y 데이터의 평균을 뺀 값, 즉 "평균 중심화(mean-centered)"된 데이터

1. **상관계수 (Correlation coefficient)**: 이는 두 변수 x와 y 간의 선형 관계의 강도와 방향을 측정하는 값입니다.
   - 상관계수는 -1과 1 사이의 값을 가지며, +1은 완벽한 양의 선형 관계, -1은 완벽한 음의 선형 관계, 0은 선형 관계가 전혀 없음을 나타냅니다.
2. **코사인 유사도 (Cosine similarity)**: 이는 두 벡터 간의 코사인 각도를 이용하여 계산합니다.
   - 코사인 유사도는 두 벡터의 방향이 얼마나 유사한지를 측정하며, 이는 벡터의 크기와는 독립적입니다. 코사인 유사도는 -1과 1 사이의 값을 가집니다.

💡 Example: x = (1, 2, 3), y = (2, 4, 6), **x - x̄ = (-1, 0, 1), y - ȳ = (-2, 0, 2)**

- 같은 방향: 1, 다른 방향: -1. 직각: 0 → Correlation

Numpy Version

- np.cov는 공분산 행렬을 반환합니다: 대각선은 변수의 분산입니다.
- np.corrcoef는 상관계수 행렬을 반환합니다: 대각선은 항상 1입니다.
- $n \times n$ 공분산 행렬을 얻을 수 있습니다.

```
# NumPy 배열로 변환
num_friends = np.array(num_friends)
daily_minutes = np.array(daily_minutes)

# 두 배열을 행으로 하는 2D 배열 생성
data = np.array([num_friends, daily_minutes])

# 데이터의 모양 출력 (2, N) 형태일 것임
print(data.shape)
print()

# 공분산 행렬 계산 및 출력
# 대각선 요소는 각각의 변수에 대한 분산을, 비대각선 요소는 두 변수 간의 공분산을 나타냄
print(np.cov(data))

# 상관계수 행렬 계산 및 출력
# 대각선 요소는 1(자기 자신과의 상관계수), 비대각선 요소는 두 변수 간의 상관계수를 나타냄
print(np.corrcoef(data))
print()

# 두 변수 간의 상관계수만 출력
print(np.corrcoef(data)[0,1])
print()

# 주석 처리된 부분: 각 변수의 표본 분산 계산
# ddof=1은 표본 분산을 계산할 때 사용됨. 기본값은 모분산 계산
# print(np.var(num_friends, ddof=1))
# print(np.var(daily_minutes, ddof=1))

# NumPy 배열을 다시 리스트로 변환
```

```
num_friends = list(num_friends)
daily_minutes = list(daily_minutes)
```

💡 **상관계수의 절대값이 1에 가까울수록 강한 선형 관계를 나타내며, 0에 가까울수록 선형 관계가 약하거나 없음을 의미**

```
(2, 204)

[[ 81.54351396  22.42543514]
 [ 22.42543514 100.78589895]]
[[1.         0.24736957]
 [0.24736957 1.        ]]

0.247369573664782
```

Simpson's Paradox (심슨의 역설)

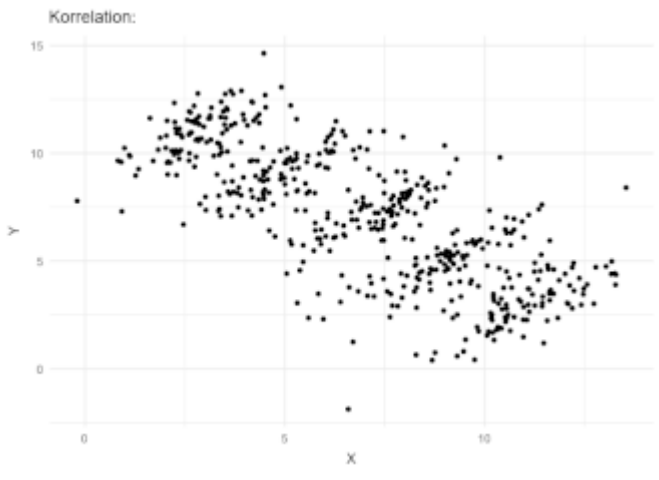
- 데이터를 분석할 때 하나의 놀라운 점은 심슨의 역설인데, 이 역설에서 혼동 변수를 무시하면 상관관계가 오해의 소지가 있을 수 있습니다.

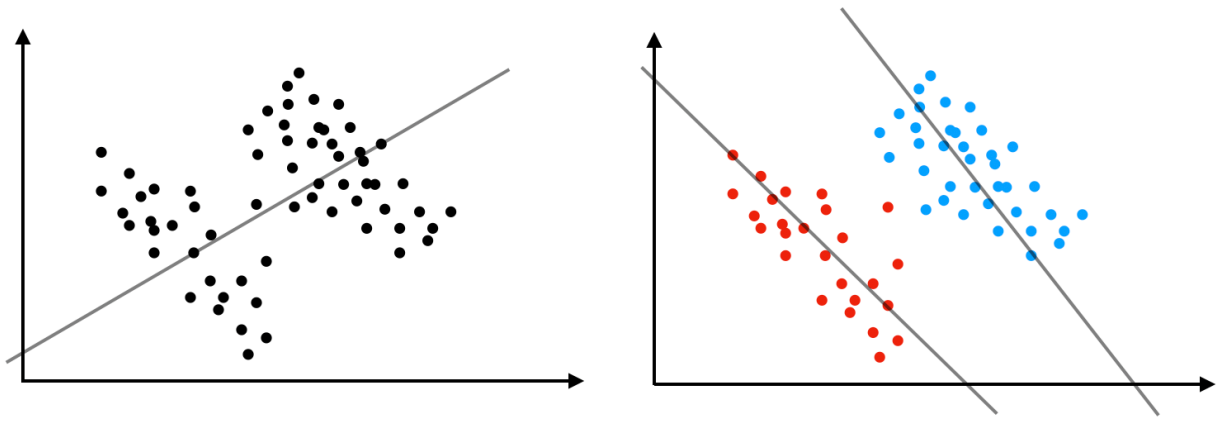
해안	회원 수	평균 친구 수
서부 해안	101	8.2
동부 해안	103	6.5

- 서부 해안의 데이터 과학자들은 동부 해안의 데이터 과학자들보다 더 친근하다고 볼 수 있습니다.
- 왜 그럴까요?
- 사실, 추가 변수를 도입하면 상관관계가 정반대 방향으로 나타납니다!

해안	학위	회원 수	평균 친구 수
서부 해안	박사	35	3.1
동부 해안	박사	70	3.2
서부 해안	비박사	66	10.9
동부 해안	비박사	33	13.4

- 이를 피하는 유일한 방법은 데이터를 잘 알고, 가능한 혼동 요인을 확인하려고 노력하는 것입니다.
- 물론, 이것이 항상 가능한 것은 아닙니다.
- **실제 세계의 변동성을 반영한 데이터에 대한 심슨의 역설의 시각화는 진정한 관계를 판단하는 것이 실제로 어려울 수 있음을 나타냅니다.**
- 전체적으로는 강한 부정적 상관관계: -0.74
- 개별적으로는 강한 긍정적 상관관계: +0.74, +0.82, +0.75, +0.72, +0.69





Some Other Correlational Caveats (기타 상관관계 주의사항)

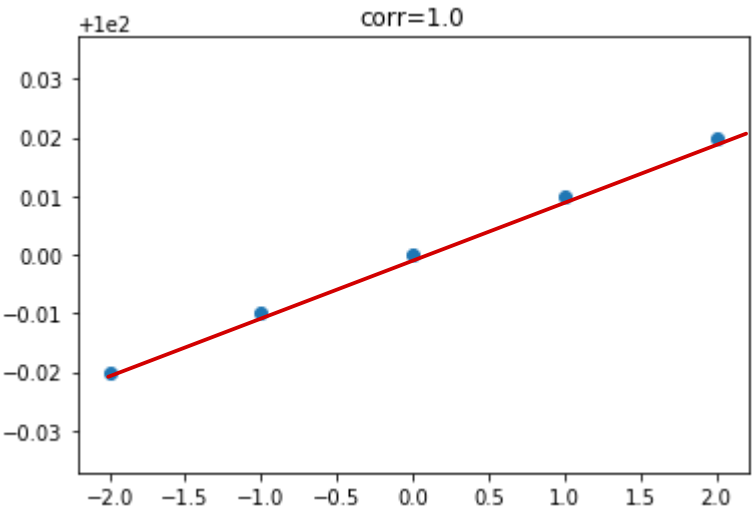
- 상관관계가 0이면 두 변수 사이에 선형 관계가 없음을 나타냅니다.
- 관련성이 있으나 포착되지 않음 (**Related but not captured)**

> 💡 Correlation(상관 관계) X → but, Correlation(상관 관계)로 환원 가능

```
# absolute value relationship
x = [-2, -1, 0, 1, 2]
y = [ 2, 1, 0, 1, 2]
plt.scatter(x, y)
plt.show()
```


- 상관관계가 얼마나 중요하거나 얼마나 흥미로운가요? (**How important or how interesting the correlation is?)**

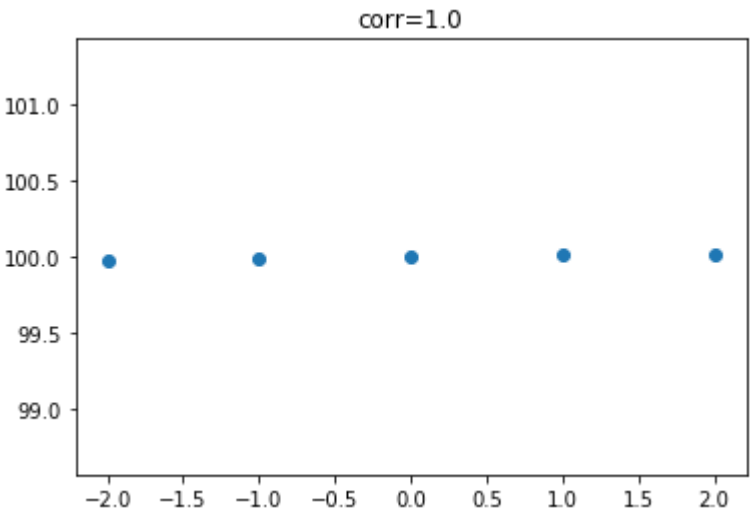
```
# x와 y 데이터를 정의합니다.
x = [-2, -1, 0, 1, 2]
y = [99.98, 99.99, 100, 100.01, 100.02]

# x와 y 데이터를 기반으로 산점도를 그립니다.
plt.scatter(x, y)

# 산점도의 제목을 설정합니다. 여기서 correlation(x,y)는 x와 y 사이의 상관관계를 계산하는 함수로,
# 이 함수의 결과값을 소수점 둘째 자리까지 반올림하여 제목에 포함시킵니다.
plt.title('corr={:.2}'.format(correlation(x, y)))

plt.show()
```



💡 **상관관계는 두 변수의 관계의 강도와 방향을 나타내는 지표로, -1부터 1까지의 값으로 표현됩니다.**

1에 가까울수록 강한 정(+)의 선형 관계를, -1에 가까울수록 강한 부(-)의 선형 관계를, 0에 가까울수록 선형 관계가 약하거나 없음을 의미

```
# x와 y 데이터를 정의합니다. 이 데이터들은 완벽한 선형 관계를 나타냅니다.
x = [-2, -1, 0, 1, 2]
y = [99.98, 99.99, 100, 100.01, 100.02]

# x와 y 데이터를 이용하여 산점도를 그립니다.
plt.scatter(x, y)

# 축을 동등한 비율로 설정합니다. 이는 데이터 간의 관계를 더 명확하게 보여주기 위함입니다.
plt.axis('equal')

# 그래프의 제목에 x와 y 데이터 간의 상관관계 계수를 표시합니다.
# 여기서 'correlation(x,y)'는 x와 y 데이터의 상관관계를 계산하는 함수를 호출하며,
plt.title('corr={:.2}'.format(correlation(x,y)))

plt.show()
```



💡 `plt.axis('equal')` **은 두 축의 스케일을 동일하게 설정하여, 데이터 포인트 간의 직선적 관계를 더욱 명확하게 보여주는 데 사용됩니다.**

상관관계 계수(`corr` **)는 -1에서 1 사이의 값을 가지며, 이 값이 1에 가까울수록 변수 간에 완벽한 양의 선형 관계가 있음을 의미합니다. 이 코드에서는** `correlation(x, y)` **함수를 통해 계산.**

Correlation and Causation (상관관계와 인과관계)

- "상관관계는 인과관계가 아니다(**Correlation is not causation)**"
- 만약 x와 y가 강한 상관관계를 가진다면,
  - 그것은 x가 y를 일으킨다거나,
  - y가 x를 일으킨다거나,
  - 서로가 서로를 일으킨다거나,
  - 어떤 제3의 요인이 둘 다를 일으킨다거나,
  - 또는 아무 의미가 없을 수도 있습니다.
- DataSciencester 사이트에서 더 많은 친구를 가지고 있는 것이 사용자들이 사이트에서 더 많은 시간을 보내는 원인이라고 할 수 있습니다.
- 데이터사이언스 포럼에서 논쟁하는 시간이 길수록, 유사한 사고방식을 가진 사람들을 더 많이 만나고 친구를 사귈 가능성이 있습니다.
- 데이터 과학에 열정적인 사용자들이 사이트에서 더 많은 시간을 보내는 것은 사이트를 더 흥미롭게 여겨서이며, 더 많은 데이터 과학 친구를 활동적으로 모으는 것은 다른 사람들과 교류하고 싶지 않기 때문일 수 있습니다.

Causation

- Experiment (실험을 통해)
- Or explore history data (또는 역사 데이터를 조사하여)
- 조건부 확률 관점에서 인과관계가 추론될 수 있습니다. (Causation may be inferred in terms of conditional probability)
- 연관 규칙은 인과관계를 추론하기 위한 데이터 마이닝 기술입니다. (Assocaition rule is a data mining technique to infer a causation)