MÔ TẢ DỮ LIỆU BẰNG THỐNG KÊ

Ở chương này chúng ta dùng Python để khám phá các số liệu thống kê, giúp ta hiểu hơn về dữ liệu mà ta đang có. Trong chương này trình bày các thước đo từ đơn giản như trung bình, trung vị, phân bố đến những phần nâng cao như phương sai, độ lệch chuẩn và phần cuối trình bày thuật toán tính hệ số tương quan (correlation coefficient).

I.Mean

Gía trị trung bình là một khái niệm tổng quát khi nói đến một tập số liệu, khi nói đến bộ số liệu nào đó, chúng ta hay nhắc đến trung bình của bộ số liệu đó. Python cung cấp hàm mean() để chúng ta tính trung bình nhanh hơn. Nhưng chúng ta đều hiểu bản chất của khái niệm trung bình là lấy tổng của dãy số cho trước sau đó chia cho độ dài của dãy số. Dưới đây, tôi sẽ trình bày 2 cách tính trên.

Ví dụ: Chúng ta có dãy số liệu dưới đây thể hiện số tiền quyên góp của một tổ chức từ thiện trong 12 ngày qua, đơn vị USD: *100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, and 1200*

Cách 1: Dùng hàm mean() của python.

In [1]:
```
#import packages time để tính thời gian hàm mean trả kết quả
import time
start= time.time()
# import packages numpy dùng để tính toán
import numpy as np
# Tạo mảng donations đại diện cho số tiền quyên góp trong 12 ngày
donations = np.array([100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000,120
0])
# Dùng hàm .mean() để tính trung bình
print('Trung bình số tiền quyên góp là : ', np.mean(donations))
print('Thời gian tính hàm mean bằng hàm của python', (time.time()-start)*1000)
```

```
Trung bình số tiền quyên góp là :  477.75
Thời gian tính hàm mean bằng hàm của python 112.69664764404297
```

Cách 2: Viết hàm để tính trung bình của một dãy số.

In [2]:
```
def calculate_mean(numbers): #1
    s = sum(numbers)   #2
    N = len(numbers)   #3
    mean = s/N        #4
    return mean       #5
```

1. Bước đầu tiên ta xác định hàm cần tính, đặt tên là *calculate_mean()*, hàm này sẽ nhận đối số *numbers* là chuỗi các số cần tính trung bình.***
2. Bước 2 ta sử dụng hàm sum() để tính tổng dãy số cho trước, đại diện bởi biến s.
3. Bước 3 ta sử dụng hàm len() để tính chiều dài của dãy số cần tính, đại diện bởi biến N.
4. Bước 4 ta tính trung bình của dãy số trên bằng cách lấy tổng chia cho chiều dài, nghĩa là lấy s chia N, đại điện bởi biến mean.
5. Cuối cùng ta cho hàm trả về giá trị mean tính được ở bước 4.

In [3]:
```
start= time.time()
donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000,1200]
print('Trung bình số tiền quyên góp là : ', calculate_mean(donations))
print('Thời gian tính hàm mean bằng hàm calculate_mean', (time.time()-start)*1
000)
```

```
Trung bình số tiền quyên góp là :  477.75
Thời gian tính hàm mean bằng hàm calculate_mean 0.0
```

Ta thấy cả hai cách đều cho giá trị đúng nhưng cách áp dụng hàm calculate_mean() có hiệu suất tốt hơn với thời gian tính toán là cực nhỏ.

II. Median

Số trung vị là một số đứng ở vị trí giữa trong dãy số đã được sắp xếp theo thứ tự tăng dần, median chia dãy số cho trước thành 2 nửa bằng nhau.Nếu độ dài của dãy số là số lẻ, thì số ở giữa là trung bị, nếu chiều dài của dãy số là số chẵn thì trung bị được xác định bằng cách lấy trung bình của hai số ở giữa.

Ví dụ: Chúng ta có dãy số liệu dưới đây thể hiện số tiền quyên góp của một tổ chức từ thiện trong 12 ngày qua, đơn vị USD: *100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, and 1200*

Cách 1: Dùng hàm median() của python.

In [4]:
```
#import packages time để tính thời gian hàm median trả kết quả
import time
start= time.time()
# import packages numpy dùng để tính toán
import numpy as np
# Tạo mảng donations đại diện cho số tiền quyên góp trong 12 ngày
donations = np.array([100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000,120
0])
# Dùng hàm np.median() để tính trung vị
print('Trung bình số tiền quyên góp là : ', np.median(donations))
print('Thời gian tính hàm mean bằng hàm của python', (time.time()-start)*1000)
```

```
Trung bình số tiền quyên góp là :  500.0
Thời gian tính hàm mean bằng hàm của python 0.9951591491699219
```

Cách 2: Dùng hàm tính median.

In [5]:
```
def calculate_median(numbers):
    N = len(numbers)
    numbers.sort()
    if N%2 == 0:
        m1 = N/2
        m2 = (N/2) + 1
        m1 = int(m1)-1
        m2 = int(m2)-1
        median = (numbers[m1] + numbers[m2])/2
    else:
        m = (N+1)/2
        m = int(m)-1
        median = numbers[m]
    return median
```

1. Bước 1 xác định hàm cần tính, đặt tên calculate_median, hàm này sẽ nhận đối số là chuối số numbers ta truyền vào.
2. Bước 2 tính chiều dài của chuỗi số numbers, đại diện bởi biến N
3. Bước 3 sắp xếp chuỗi số numbers từ nhỏ đến lớn.
4. Tính median: Nếu chiều dài của chuỗi số numbers là số lẻ, thì số trung vị là số ở vị trí chính giữa, nghĩa là (N + 1)/2. Nếu chiều dài của numbers là số chẵn thì lấy 2 phần tử ở giữa là N/2 và (N/2) + 1, đại diện bởi biến m1 và m2, sau đó ra lấy phần nguyên của phép chia rồi trừ một để lấy được vị trí của 2 phần tử. Cuối cùng ta lấy giá trị trung bình của 2 phần từ vừa tìm được.
5. Ta cho hàm trả về giá trị trung vị tính ở bước 4

In [6]:
```
start= time.time()
donations = np.array([100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000,120
0])
print('Trung vị số tiền quyên góp là : ', calculate_median(donations))
print('Thời gian tính hàm mean bằng hàm calculate_mean', (time.time()-start)*1
000)
```

```
Trung vị số tiền quyên góp là :  500.0
Thời gian tính hàm mean bằng hàm calculate_mean 0.9837150573730469
```

III.Mode

Mode trả về xuất hiện nhiều nhất trong dãy số cho trước. Ví dụ ta có điểm thi toán của 20 học sinh, trên thang 10 điểm: *7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10*. Mode sẽ giúp ta trả lời câu hỏi: *"Điểm số nào được học sinh đạt nhiều nhất?"*. Mode sẽ trả lời cho ta là điểm 9. Công việc mode thực hiện là với mỗi giá trị trong chuỗi số, đếm số lần xuất hiện của giá trị đó và trả ra giá trị có số lần xuất hiện nhiều nhất.

Ta lấy lại ví dụ điểm số 20 học sinh để tính mode chi tiết

In [7]:
```
points = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10]
```

Cách 1: Python cung cấp hàm most_common() để tính mode.

In [8]:
```
start= time.time()
# import packages Counter để đếm số lần xuất hiện của mỗi giá trị trong chuỗi
from collections import Counter
c = Counter(points) # đếm số xuất hiện
print(c)
print(c.most_common())  # Sắp xếp giá trị các phần tử có số luần xuất hiện từ
 cao tới thấp
print('Thời gian tính mode most_common(): ', (time.time()-start)*1000)
```

```
Counter({9: 5, 6: 3, 7: 2, 8: 2, 10: 2, 5: 2, 1: 2, 2: 1, 4: 1})
[(9, 5), (6, 3), (7, 2), (8, 2), (10, 2), (5, 2), (1, 2), (2, 1), (4, 1)]
Thời gian tính mode most_common():  0.9987354278564453
```

Nhưng ta chỉ cần giá trị xuất hiện nhiều nhất thôi, ví dụ trong trường hợp trên chỉ cần trả về 9, ta cần thêm vài thao tác để chỉ đúng vị trí của điểm cần tìm.

In [9]:
```
mode = c.most_common(1)
mode[0]
mode[0][0]
print('Số điểm xuất hiện nhiều nhất: ',mode[0][0])
```

```
Số điểm xuất hiện nhiều nhất:  9
```

Cách 2: Viết hàm tính mode.

In [10]:
```
def calculate_mode(numbers): #1
    c = Counter(numbers)     #2
    mode = c.most_common(1)  #3
    return mode[0][0]        #4
```

1. Xác đinh hàm cần tìm, đặt tên calculate_mode, hàm này nhận đối số numbers truyền vào.
2. Dùng hàm counter để đếm số lần xuất hiện của mỗi phần tử trong chuỗi, đại diện bởi biến c.
3. dùng most_common(), truyền vào 1, trả về giá trị và số lần xuất hiện của phần tử xuất hiện nhiều nhất, đại diện bởi biến mode.
4. Trả về phần giá trị của mode được tính ở bước 3.

In [11]:
```
print('Mode của chuỗi số đã cho : ', calculate_mode(points))
```

```
Mode của chuỗi số đã cho :  9
```

Chuyện gì xảy ra nếu dãy số truyền vào có nhiều phần tử có mật độ xuất hiện giống nhau? Ta vẫn tính và trả về toàn bộ các mode đó.

In [12]:
```
def calculate_mode(numbers):
    c = Counter(numbers)
    numbers_freq = c.most_common()
    max_count = numbers_freq[0][1]
    modes = []
    for num in numbers_freq:
        if num[1] == max_count:
             modes.append(num[0])
    return modes
```

1. Xác đinh hàm cần tìm, đặt tên calculate_mode, hàm này nhận đối số numbers truyền vào.
2. Dùng hàm counter để đếm số lần xuất hiện của mỗi phần tử trong chuỗi, đại diện bởi biến c.
3. dùng most_common(), trả về giá trị và số lần xuất hiện của mỗi phần tử, đại diện bởi biến number_freq.
4. Theo sắp xếp của hàm most_common, vị trí đầu tiên chứa giá trị và số lần xuất hiện nhiền nhất, nên ta cần lấy số lần xuất hiện của giá trị xuất hiện nhiều nhất, đại diện bởi biến max_count.
5. Tạo một chuỗi rỗng để chưa giá trị.
6. Với lần lượt từng cặp giá trị trong bước 3 tính được, so sánh với max_count của bước 4 trả về, nếu bằng max_count thì bỏ giá trị phần tử vào mảng rỗng tạo ở bước 5

In [13]:
```
points = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10, 6, 6]
print('Mode của dãy số points: ', calculate_mode(points))
```

```
Mode của dãy số points:  [9, 6]
```

Nếu ta muốn trình bày toàn bộ số lần xuất hiện cùng giá trị của chuỗi thành bảng?

In [14]:
```
def frequency_table(numbers):
    table = Counter(numbers)
    numbers_freq = table.most_common()
    numbers_freq.sort()
    print('Number\tFrequency')
    for number in numbers_freq:
            print('{0}\t{1}'.format(number[0], number[1]))
```

1. Xác đinh hàm cần tìm, đặt tên frequency_table, hàm này nhận đối số numbers truyền vào.
2. Dùng hàm counter để đếm số lần xuất hiện của mỗi phần tử trong chuỗi, đại diện bởi biến table.
3. dùng most_common(), trả về giá trị và số lần xuất hiện của mỗi phần tử, đại diện bởi biến number_freq.
4. Vì hàm most_common sắp xếp theo số lần xuất hiện nhưng ta đang cần sắp xếp theo giá trị phần tử, nên dùng sort()
5. Tạo bảng với cột trái là giá trị phần tử, cột phải là số lần xuất hiện (\t để tạo ra khoảng trống giữa 2 giá trị)
6. Với lần lượt từng cặp giá trị trong bước 4 tính được, in giá trị phần tử trước rồi khoảng trống rồi số lần xuất hiện.hàm format dùng để truyền giá trị vào các phần tử trước đó được bỏ trong dấu *"{}"*.

In [15]:
```
frequency_table(points)
```

```
Number  Frequency
1       2
2       1
4       1
5       2
6       5
7       2
8       2
9       5
10      2
```

IV. Range of data

Độ phân tán của dữ liệu cho biết dữ liệu chúng ta có giá trị trải rộng như thế nào. Ví dụ, nếu chỉ quan tâm đến trung bình thì có khi ta đánh giá sai dữ liệu.Độ phân tán dữ liệu đơn giản là độ lệch giữa giá trị lớn nhất và giá trị nhỏ nhất của tập dữ liệu.

In [16]:
```
def find_range(numbers):
    lowest = min(numbers)
    highest = max(numbers)
    r = highest-lowest
    return print('Lowest: {0}\tHighest: {1}\tRange: {2}'.format(lowest, highes
t, r))
```

1. Xác đinh hàm cần tìm, đặt đến là find_range, hàm nhận đối số numbers cho trước.
2. Dùng hàm min tính giá trị nhỏ nhất của chuỗi numbers, đại diện bởi biến lowest.
3. Dùng hàm max tính giá trị lớn nhất của chuỗi numbers, đại diện bởi biến highest.
4. Tính range của chuỗi numbers bằng cách lấy giá trị lớn nhất trừ giá trị nhỏ nhất, đại diện bởi biến r.
5. Trả kết quả range

```
In [17]: points = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10, 6, 6]
```

```
In [18]: find_range(points)
```

```
Lowest: 1       Highest: 10     Range: 9
```

V. Variance

Phương sai cho ta biết các giá trị trong tập dữ liệu có khác biệt nhiều với giá trị trung bình của cả tập hay không? Đánh giá mức độ phân tán của giữ liệu so với giá trị trung bình.Nếu muốn tính độ lệch chuẩn chỉ cần lấy căn bậc hai của phương sai.

Công thức của variance: Lấy từng giá trị của tập dữ liệu trừ cho giá trị trung bình của cả tập, bình phương và chia cho số lượng phần tử có trong chuỗi.

$$variance = \frac{\sum(x_i - x_{mean})^2}{n}$$

Ví dụ, điểm số của 20 học sinh được cho ở mảng points dưới đây

```
In [19]: points = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10, 6, 6]
```

In [20]:
```
def calculate_mean(numbers):
    s = sum(numbers)
    N = len(numbers)
    mean = s/N
    return mean

def caculate_variance(numbers):
    mean = calculate_mean(numbers)
    diff = []
    for num in numbers:
         diff.append(num-mean)
    squared_diff = []
    for d in diff:
        squared_diff.append(d**2)
        sum_squared_diff = sum(squared_diff)
        variance = sum_squared_diff/len(numbers)
    return variance
```

In [21]:
```
print('Phương sai của dãy số là : ', caculate_variance(points))
print('Độ lệch chuẩn của dãy số là : ', caculate_variance(points)**0.5)
```

```
Phương sai của dãy số là :  6.9772727272727275
Độ lệch chuẩn của dãy số là :  2.6414527683213884
```

1. Trong công thức cần phép tính trung bình nên ta dùng lại hàm calculate_mean() phía trên.
2. Xác đinh hàm cần tìm, đặt đến là caculate_variance, hàm nhận đối số numbers cho trước.
3. Tính giá trị trung bình của dãy numbers cho trước, đại diện bởi biến mean.
4. Tạo mảng diff rỗng, với mỗi giá trị của dãy numbers, đem trừ cho mean rồi bỏ kết quả vào mảng diff
5. Tạo mảng squared_diff rỗng, với mỗi giá trị của mảng diff ở bước 4, bình phương mỗi giá trị, sum toàn bộ giá trị của mảng và chia cho chiều dài của mảng.
6. Trả kết quả phương sai

VI. CORRELATION COEFFECIENT

Khi phân tích hồi quy tuyến tính thì yêu cầu hai biến độc lập phải có mối quan hệ tuyến tính với nhau. Khi đó ta đánh giá qua hệ số tương quan. Hệ số tương quan thể hiện mối quan hệ tương quan tuyến tính giữa hai biến, hệ số tương quan nằm trong khoảng [-1; 1],

- Khi hệ số tương quan bằng 0 thì ta kết luận hai biến không có tương quan tuyến tính với nhau (nhưng không chắc chúng độc lập),
- khi hệ số gần hoặc bằng 1 thì ta nói có mối quan hệ tuyến tính dương (cùng tăng hoặc cùng giảm),
- khi hệ số gần bằng -1 thì ta nói hai biến số có mối quan hệ tuyến tính âm (x giảm y tăng và ngược lại)

Lưu ý rằng tương quan tuyến tinh KHÁC với mối quan hệ nhân quả, giả sử khi hệ số tương quan là dương thì ta chỉ được kết luận hai biến có mối quan hệ tuyến tính dương thôi. Ví dụ vào mùa hè, doanh số bán đồ tắm tăng mạnh và số lượng kem bán ra cũng tăng mạnh, khi xét hệ số tuyến tính có kết quả dương gần 1 ta không thể kết luận do lượng kem bán ra nhiều làm tăng doanh số bán áo tắm.

Công thức tính hệ số tương quan của hai biến x,y:

$$Correlation = \frac{n\sum xy - \sum x \sum y}{((n\sum x^2 - (\sum x)^2)(n\sum y^2 - (\sum y)^2))^{0.5}}$$

In [22]:
```
import numpy as np
x = np.random.randint(0, 99, size=50)
y = np.random.randint(0, 99, size=50)
```

In [23]:
```
def find_corr_x_y(x,y):
    n = len(x)
    prod = []
    for xi,yi in zip(x,y):
         prod.append(xi*yi)
    sum_prod_x_y = sum(prod)
    sum_x = sum(x)
    sum_y = sum(y)
    squared_sum_x = sum_x**2
    squared_sum_y = sum_y**2
    x_square = []
    for xi in x:
         x_square.append(xi**2)

    x_square_sum = sum(x_square)

    y_square=[]
    for yi in y:
        y_square.append(yi**2)

    y_square_sum = sum(y_square)

    # Use formula to calculate correlation
    numerator = n*sum_prod_x_y - sum_x*sum_y
    denominator_term1 = n*x_square_sum - squared_sum_x
    denominator_term2 = n*y_square_sum - squared_sum_y
    denominator = (denominator_term1*denominator_term2)**0.5
    correlation = numerator/denominator
    return correlation
```

1. Trong công thức phép tính hệ số tương quan có n là chiều dài của dữ liệu.
2. Xác đinh hàm cần tìm, đặt đến là find_corr_x_y, hàm nhận đối số x và y cho trước.
3. Trong công thức chủ yếu là phép nhân x.y nên ta dùng hàm zip để ghép 2 mảng x, y thành tabel rồi dùng hàm in để chỉ tới phần tử cần tính.
4. Chuẩn bị dữ liệu như
$$\sum(x.y), \sum x, \sum y, \sum(x^2), \sum(y^2), (\sum x)^2, (\sum y)^2, \sum(xy)$$
5. Thay lần lượt vào công thức
6. Trả về kết quả hệ số tương quan của x,y

In [24]:
```
print('Hệ số tương quan tuyến tính giữa hai biến x,y: ',find_corr_x_y(x,y))
```

```
Hệ số tương quan tuyến tính giữa hai biến x,y:  4.152975074228733

C:\Users\a\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: RuntimeWarni
ng: overflow encountered in long_scalars
```

Programming Challenges

#1: Better Correlation Coefficient–Finding Program

In [25]:
```
import numpy as np
x = np.random.randint(0, 99, size=50)
y = np.random.randint(0, 99, size=50)
```

In [26]:
```
def find_corr_x_y(x,y):
    if len(x) != len(y):
        print('Hai bộ dữ liệu không bằng nhau')
        return None
    else:
        n = len(x)
        prod = []
        for xi,yi in zip(x,y):
             prod.append(xi*yi)
        sum_prod_x_y = sum(prod)
        sum_x = sum(x)
        sum_y = sum(y)
        squared_sum_x = sum_x**2
        squared_sum_y = sum_y**2
        x_square = []
        for xi in x:
             x_square.append(xi**2)

        x_square_sum = sum(x_square)

        y_square=[]
        for yi in y:
            y_square.append(yi**2)

        y_square_sum = sum(y_square)

        # Use formula to calculate correlation
        numerator = n*sum_prod_x_y - sum_x*sum_y
        denominator_term1 = n*x_square_sum - squared_sum_x
        denominator_term2 = n*y_square_sum - squared_sum_y
        denominator = (denominator_term1*denominator_term2)**0.5
        correlation = numerator/denominator
    return correlation
```

1. Như hàm find_corr_x_y tính tương quan giữa hai biến, ta thêm điều kiện kiểm tra chiều dài hai mảng cho trước.
2. Trong công thức phép tính hệ số tương quan có n là chiều dài của dữ liệu.
3. Xác đinh hàm cần tìm, đặt đến là find_corr_x_y, hàm nhận đối số x và y cho trước.
4. Trong công thức chủ yếu là phép nhân x.y nên ta dùng hàm zip để ghép 2 mảng x, y thành tabel rồi dùng hàm in để chỉ tới phần tử cần tính.
5. Chuẩn bị dữ liệu như
$$\sum(x.y), \sum x, \sum y, \sum(x^2), \sum(y^2), (\sum x)^2, (\sum y)^2, \sum(xy)$$
6. Thay lần lượt vào công thức
7. Trả về kết quả hệ số tương quan của x,y

In [27]:
```
print('Hệ số tương quan của hai biến x,y : ', find_corr_x_y(x,y))
```

```
Hệ số tương quan của hai biến x,y :  nan

C:\Users\a\Anaconda3\lib\site-packages\ipykernel_launcher.py:31: RuntimeWarni
ng: overflow encountered in long_scalars
C:\Users\a\Anaconda3\lib\site-packages\ipykernel_launcher.py:31: RuntimeWarni
ng: invalid value encountered in power
```

#2: Statistics Calculator

Đọc file dữ liệu cho trước và thực hiện các phép tính toán thống kê như mean, median, mode, variances, standard deviation.

In [28]:
```
from collections import Counter
def read_data(filename):
    numbers = []
    with open(filename) as f:
        for line in f:
            numbers.append(float(line))
    return numbers
def caculate_statistics(data):
    data = read_data(data)
    mean = calculate_mean(data)
    median = calculate_median(data)
    mode = calculate_mode(data)
    variances = caculate_variance(data)
    std = variances**0.5
    return print('Dữ liệu đã có:{0}\nTrung bình: {1}\nTrung vị: {2}\nMode:{3}
\nPhương sai: {4}\nĐộ lệch chuẩn:{5}'.format(data,mean,median,mode,variances,s
td))
```

In [29]:
```
caculate_statistics('C:/Users/a/Desktop/doingmath_python/sample/mydata.txt')
```

```
Dữ liệu đã có:[60.0, 70.0, 100.0, 100.0, 200.0, 500.0, 500.0, 503.0, 600.0, 9
00.0, 1000.0, 1200.0]
Trung bình: 477.75
Trung vị: 500.0
Mode:[100.0, 500.0]
Phương sai: 141047.35416666666
Độ lệch chuẩn:375.5627166887931
```

#4: Finding the Percentile

Viết hàm trả về kết quả là giá trị của phân vị thứ *q* của dãy số cho trước. Ví dụ dãy số

$$[5, 1, 9, 3, 14, 9, 7]$$

, phân vị thứ 50th của dãy số trên bằng 7, nghĩa là 50% số lượng dữ liệu sau khi sắp xếp nhỏ hơn 7.

In [30]:
```
def find_percentile_score(data, percentile):
    if percentile < 0 or percentile > 100:
        return None

    data.sort()
    if percentile == 0:
        return data[0]
    if percentile == 100:
        return data[-1]
    n = len(data)
    rank = (percentile/100)*(n-1) + 1
    k = int(rank)
    d = rank - k
    real_idx_1 = k-1
    real_idx_2 = k
    return print('Phân vị thứ {0} của dữ liệu đã cho là: {1}'. format(percenti
le,data[real_idx_1] + d*(data[real_idx_2]-data[real_idx_1])))
```

In [31]:
```
data = [ 5, 1, 9, 3, 14, 9, 7]
```

In [32]:
```
find_percentile_score(data, 50)
```

```
Phân vị thứ 50 của dữ liệu đã cho là: 7.0
```

#5: Creating a Grouped Frequency Table

Viết hàm gom gom nhóm chuỗi dữ liệu và thể hiện số lượng phần tử trong từng nhóm. Ví dụ: khi truyền vào dãy số 7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10. Với yêu cầu chia thành 3 nhóm.

- Bước 1Tạo nhóm: Với mảng đã cho chia thành n nhóm với range của mỗi nhóm là bằng nhau, nhóm 1 từ 1 đến dưới 5 thì nhóm 2 sẽ từ 6 đến dưới 10.
- Bước 2: Với mỗi phần tử của mảng so sánh lần lượt với giá trị đầu cuối của mỗi nhóm, nếu thuộc nhóm nào thì đếm 1 vào biến số lượng của nhóm ấy.

In [33]:
```
def create_classes(numbers, n):
    low = min(numbers)
    high = max(numbers)
    width = (high - low)/n
    classes = []
    a = low
    b = low + width
    classes = []
    while a < (high-width):
        classes.append((a, b))
        a = b
        b = a + width
    classes.append((a, high+1))
    return classes

def classify(numbers, n):
    classes = create_classes(numbers, n)
    count = [0]*len(classes)
    for n in numbers:
        for index, c in enumerate(classes):
            if n >= c[0] and n < c[1]:
                count[index] += 1
                break
    for c, cnt in zip(classes, count):
        print('{0:.2f} - {1:.2f} \t {2}'.format(c[0], c[1], cnt))
```

In [34]:
```
numbers = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10]
```

In [35]:
```
classify(numbers,3)
```

```
1.00 - 4.00      3
4.00 - 7.00      6
7.00 - 11.00     11
```