1- Create a sample having 11 observations. Obtain the quartiles using the four methods given in the link https://en.wikipedia.org/wiki/Quartile#Discrete_distributions (Links to an external site.). Note that the method we've discussed is Method 4.

Sample dataset,

73, 35, 76, 6, 72, 53, 45, 12, 33, 29, 60

Sorted values,

6, 12, 29, 33, 35, 45, 53, 60, 72, 73, 76

Method1,

Find the Median, and based on its position, separate the samples into two parts and find the median of each part as the first quartile and the third quartile.

The point of this method is, for an odd number of samples after finding the median **omit** it for finding quartiles.

N = 11, Median = 45, Q1 = 29, Q3 = 72, IQR = 50

Method2,

Find the Median, and based on its position, separate the samples into two parts and find the median of each part as the first quartile and the third quartile.

The point of this method is, for an odd number of samples after finding the median **include** it for finding quartiles.

N = 11, Median = 45, Q1 = (33+29)/2 = 31, Q3 = (60+72)/2 = 66, IQR = 35

Method3,

If there are even numbers of data points, then Method 3 starts the same as Method 1 or Method 2 above and you can choose to include or not include the median as a datapoint. If you choose to include the median as a new data point, proceed to step 2 or 3 of Method 3 because you now have an odd number of data points.

If there are odd numbers of data points, then

If n is an integer in (4n+1) = N then,
Q1 = (n)value *1/4 + (n+1)value *3/4
Q3 = (3n+1)value *3/4 + (3n+2)value *1/4

If n is an integer in (4n+3) = N then,
Q1 = (n+1)value *3/4 + (n+2)value *1/4
Q3 = (3n+2)value *1/4 + (3n+3)value *3/4

The point here is the value of n should be an integer and choose between the second or third part.

For example,

N = 11

Second part, 4n+1 = 11 => n =2.5 and Third part, 4n+3 = 11 => n = 2

So, for N equal to 11 we need to follow the third part of method 3 to find quartiles,

N = 11, Median = 45, Q1 = (29*3/4) + (33/4) = 30, Q3 = (60/4) + (72*3/4) = 69, IQR = 39

Method4,

Q1 = value at 0.25(n + 1) position

Q3 = value at 0.75(n + 1) position

N = 11, Median = 45, Q1 = 29, Q3 = 72, IQR = 43

Summerize,

	Q1	Median	Q3	IQR
Method1	29	45	72	50
Method2	31	45	66	35
Method3	30	45	69	39
Method4	29	45	72	43

2-

a) Define the same sample in **Python**.
b) Using np.percentiles or np.quantiles write a **Python** code that gives the quartiles of your sample. Indicate which method in the link https://en.wikipedia.org/wiki/Quartile#Discrete_distributions (Links to an external site.) is used.
c) Write **Python** codes to obtain the quartiles of your sample using each of the four methods.

```
import numpy as np
arr = [73, 35, 76, 6, 72, 53, 45, 12, 33, 29, 60]
arr.sort()

print('Q1 quantile of arr: {0}\nMedian of arr: {1}\nQ3 quantile of arr: {2}'
.format(np.quantile(arr, .25), np.quantile(arr, .50), np.quantile(arr, .75)))
```

Q1 quantile of arr: 31.0
Median of arr: 45.0
Q3 quantile of arr: 66.0

```
print('Q1 quantile of arr: {0}\nMedian of arr: {1}\nQ3 quantile of arr: {2}'
.format(np.percentile(arr, 25), np.percentile(arr, 50), np.percentile(arr, 75)))
```

Q1 quantile of arr: 31.0
Median of arr: 45.0
Q3 quantile of arr: 66.0

The output of the python code is the same as Method2.

```
import numpy as np
arr = [73, 35, 76, 6, 72, 53, 45, 12, 33, 29, 60]
def Median(arr):
    arr.sort()
    if len(arr)%2==0:
        pos = len(arr)//2
        med = (arr[pos-1]+arr[pos])/2
    else:
        pos = len(arr)//2
        med = arr[pos]
    return med

def method1(arr):
    MED = Median(arr)
    if len(arr)%2==0:
        pos = len(arr)/2
        arr1, arr2 = arr[:pos], arr[pos:]
        Q1, Q3 = Median(arr1), Median(arr2)
    else:
        pos = len(arr)//2
        arr1, arr2 = arr[:pos], arr[pos+1:]
        Q1, Q3 = Median(arr1), Median(arr2)
    return 'Q1: {0}, Median: {1}, Q3: {2}'.format(Q1, MED, Q3)

def method2(arr):
    MED = Median(arr)
    if len(arr)%2==0:
        pos = len(arr)/2
        arr1, arr2 = arr[:pos], arr[pos:]
        Q1, Q3 = Median(arr1), Median(arr2)
    else:
        pos = len(arr)//2
        arr1, arr2 = arr[:pos+1], arr[pos:]
        Q1, Q3 = Median(arr1), Median(arr2)
    return 'Q1: {0}, Median: {1}, Q3: {2}'.format(Q1, MED, Q3)

def method3(arr, include_median='N'):
    if len(arr)%2==0 and include_median=='N':
        MED = Median(arr)
        pos = len(arr)//2
        arr1, arr2 = arr[:pos], arr[pos:]
        Q1, Q3 = Median(arr1), Median(arr2)
    elif len(arr)%2==0 and include_median=='Y':
```

```
        MED1 = Median(arr)
        arr.append(MED1)
        arr.sort()
        Q1, MED, Q3 = method2(arr)
    elif len(arr)%2!=0:
        MED = Median(arr)
        n = (len(arr)-1)/4
        m = (len(arr)-3)/4
        if n.is_integer():
            pos = int(n - 1)
            pos3 = int(3*n -1)
            Q1 = arr[pos]/4 + arr[pos+1]*3/4
            Q3 = arr[pos3+1]*3/4 + arr[pos3+2]/4
        elif m.is_integer():
            pos = int(m - 1)
            pos3 = int(3*m - 1)
            Q1 = arr[pos+1]*3/4 + arr[pos+2]/4
            Q3 = arr[pos3+2]/4 + arr[pos3+3]*3/4
    return 'Q1: {0}, Median: {1}, Q3: {2}'.format(Q1, MED, Q3)

def method4(arr):
    n = (len(arr)+1) / 4
    MED = Median(arr)
    if n.is_integer():
        pos1, pos3 = int(n-1), int(3*n-1)
        Q1 = arr[pos1]
        Q3 = arr[pos3]
    else:
        m = (len(arr)+1) % 4
        n = (len(arr)+1) // 4
        pos = int(n)
        Q1 = ((arr[pos]*m)+(arr[pos+1]*(1-m)))/2
        Q3 = ((arr[3*pos]*m)+(arr[3*pos+1]*(1-m)))/2
    return 'Q1: {0}, Median: {1}, Q3: {2}'.format(Q1, MED, Q3)

print('Method1,\n   {0}\nMethod2,\n   {1}\nMethod3,\n   {2}\nMethod4,\n   {3}'
.format(method1(arr), method2(arr), method3(arr), method4(arr)))
```

Output,

Method1,
  Q1: 29, Median: 45, Q3: 72
Method2,
  Q1: 31.0, Median: 45, Q3: 66.0
Method3,
  Q1: 30.0, Median: 45, Q3: 69.0

Method4,
Q1: 29, Median: 45, Q3: 72

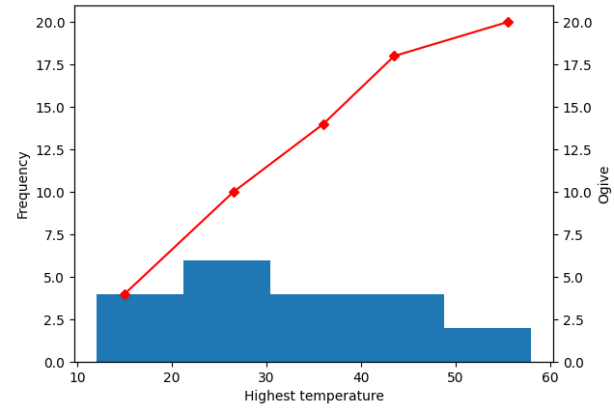
3-

In **Python**, try to sketch a Pareto chart for the dataset Highest temperatures (the one we used in the Example of our discussion on the topic Frequency distribution tables, Histogram, and Ogive).

Dataset = [24, 35, 17, 21, 24, 37, 26, 46, 58, 30, 32, 13, 12, 38, 41, 43, 44, 27, 53, 27]

```
# Pareto graph creation
import bisect

def find_range(dataset, low, high):
    l = bisect.bisect_left(dataset, low)
    r = bisect.bisect_right(dataset, high)
    return dataset[l:r]

def Median(arr):
    arr.sort()
    if len(arr)%2==0:
        pos = len(arr)//2
        med = (arr[pos-1]+arr[pos])/2
    else:
        pos = len(arr)//2
        med = arr[pos]
    return med

dataset = [24, 35, 17, 21, 24, 37, 26, 46, 58, 30, 32, 13, 12, 38, 41, 43, 44,
27, 53, 27]
dataset.sort()

# default values
co_ , medi, c1= [] , [], 0
vbin = 5
dataset_range = dataset[-1]-dataset[0]
range_c = dataset_range / vbin
a = dataset[0]
b = range_c + dataset[0]

# create points and ogive values
for i in range(0,vbin):
    if i == vbin-1:
        ran = find_range(dataset, a, b)
        c = len(ran)
        med = Median(ran)
    else:
```

```
        ran = find_range(dataset, a, b)
        while ran[-1] == b:
            ran.pop()
        c = len(ran)
        med = Median(ran)
    c1 += c
    co_.append(c1)
    medi.append(med)
    a = b
    b += range_c

# Pareto Chart
hist, bin_edge = np.histogram(dataset, bins=vbin)
fig, ax = plt.subplots()
ax.hist(dataset, bin_edge, cumulative=False)
ax.set_xlabel('Highest temperature')
ax.set_ylabel('Frequency')
ax.set_ylim(0,co_[-1]+1)
ax2 = ax.twinx()
ax2.plot(medi, co_, color='red', marker='D', ms=5)
ax2.set_ylabel('Ogive')
ax2.set_ylim(0,co_[-1]+1)
plt.show()
```



4-

One of the assignments for Day 2 of Week 1 was on covariance and correlation coefficient. Now it's time to discuss those concepts in Python. Construct a **Python document** on covariance and correlation coefficient using your own dataset (the one you used in the assignment for Day 2). This document should include the following:

a) Codes for covariance (pure Python codes and codes in which some libraries are used),

b) Codes for correlation coefficient (pure Python codes and codes in which some

libraries used),
c) The Heatmaps (the covariance and correlation coefficient matrices) for your data, and
d) your interpretations of what you've obtained.

```
from scipy import stats
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

dataset = pd.read_csv("input.csv")
dataset.describe()
```

	Height	Weight
count	200.000000	200.000000
mean	67.949800	127.221950
std	1.940363	11.960959
min	63.430000	97.900000
25%	66.522500	119.895000
50%	67.935000	127.875000
75%	69.202500	136.097500
max	73.900000	158.960000

```
arr = np.array(dataset)
c , n = arr.shape
print('counts of rows: {0}\ncounts of columns: {1}'.format(c,n))
```

counts of rows: 200
counts of columns: 2

```
# pure python calculation of covariance and
l1 , l2 = list(arr[:,0]), list(arr[:,1])
l1_mean, l2_mean = sum(l1)/c, sum(l2)/c

l1_var_s = sum((item-l1_mean)**2 for item in l1) / (c-1)
l2_var_s = sum((item-l2_mean)**2 for item in l2) / (c-1)

l1_std, l2_std = l1_var_s ** 0.5, l2_var_s ** 0.5

l1l2_cov_s = sum((l1[i]-l1_mean)*(l2[i]-l2_mean) for i in range(0,c))/c
l1l2_cor = l1l2_cov_s / (l1_std*l2_std)
print('''first column mean: {0}\nsecond column mean: {1}\nfirst column variance
of sample: {2}
second column variance of sample: {3}\nstd of first column: {4}\nstd of second
column: {5}\ncovariance_sample: {6}
correlative: {7}'''.format(l1_mean, l2_mean, l1_var_s, l2_var_s, l1_std, l2_std,
l1l2_cov_s, l1l2_cor))
```

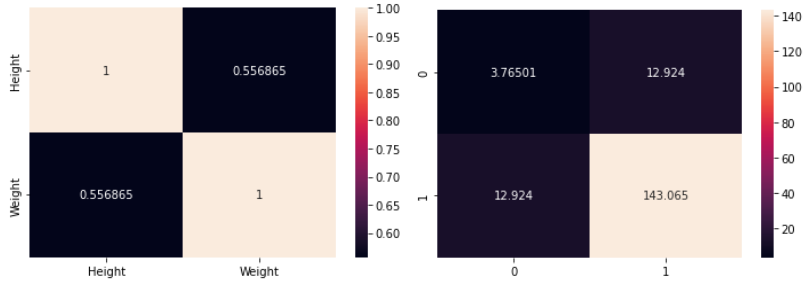
first column mean: 67.94979999999998
second column mean: 127.2219500000001
first column variance of sample: 3.765006994974874
second column variance of sample: 143.0645444195981
std of first column: 1.9403625936857456
std of second column: 11.960959176403792
covariance_sample: 12.859429389999994
correlative: 0.5540804109392374

```
# Numpy
np_cov_s = np.cov(l1,l2)[0][1]
np_cor = np.corrcoef(l1,l2)[0][1]
# Pandas
pandas_corr = dataset.corr()['Height']['Weight']
# Scipy
scipy_cor = stats.pearsonr(l1,l2)[0]
print('Numpy library output,\ncovariance: {0}\ncorrelative: {1}'.format(np_cov_s,
np_cor))
print('Pandas library output, correlative: {0}\nScipy library output,
correlative: {1}'.format(pandas_corr, scipy_cor))
```

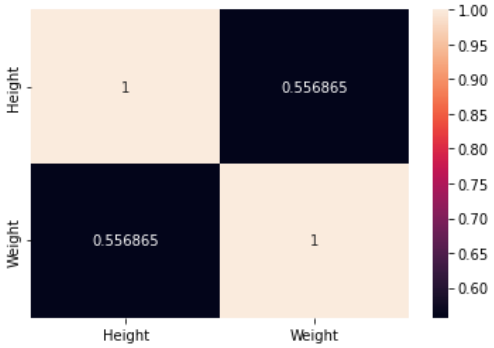
covariance: 12.92404963819095
correlative: 0.5568647346122992

Pandas library output, correlative: 0.5568647346123003
Scipy library output, correlative: 0.5568647346122995

```
# heatmap of Numpy
sn.heatmap(np.cov(l1,l2), annot=True, fmt='g')
sn.heatmap(np.corrcoef(l1,l2), annot=True, fmt='g')
```




```
# heatmap of pandas
sn.heatmap(dataset.corr(), annot=True, fmt='g')
```



According to my job, I think Pandas library creates better solutions for finding and showing the correlation between the different features in python, it can be shown heatmap graph of all features correlation at one time but in other libraries, I need to work with every two features separately to get correlation amount of each two features.

The amount of correlation is more understandable and useful than the amount of covariance.