Fitbit 데이터 분석 및 시각화

1. 걸음수 TOP10 구하기

1) 데이터 가공과정

```
import pandas as pd
import json
import os
import csv

dates = pd.date_range('20160401', '20160520')
dates05 = pd.date_range('20160501','20160520')
step = []
name = []

user_name = 'A0'
num = 1
```

=> 걸음수 Top10 을 구하기 위해서 유저의 이름과 총 걸음수를 저장하기 위해 step 과 name 의 list 를 만들고 user_name 을 A0 로 해서 num 을 하나씩 증가시켜 유저의 번호를 찾는다. 또한 fitbit 의 기간인 20160401 ~ 20160520 과 5 월부터 있는 user 때 문에 20160501 ~ 20160520 까지의 범위를 만든다.

```
for x in range(1,99):
    i = 0
    if os.path.exists('sokulee/A0'+str(x)) == True:
        if os.path.exists('sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + '20160401' + '_steps.json') == True:
            for date in dates:
                fname = 'sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + date.strftime('%Y%m%d') + '_steps.json'
                try: f = open(fname)
                except IOError as e:
                    print(str(e))
                else:
                    date_data = json.loads(f.read())
                    if (list(date_data.keys()) == ['errors', 'success'] or list(date_data.keys()) == ['success','errors']) == False :
                        i = i + int(date_data['activities-steps'][0]['value'])
        else :
            for date in dates05:
                fname = 'sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + date.strftime('%Y%m%d') + '_steps.json'
                try: f = open(fname)
                except IOError as e:
                    print(str(e))
                else:
                    date_data = json.loads(f.read())
                    if (list(date_data.keys()) == ['errors', 'success'] or list(date_data.keys()) == ['success','errors']) == False :
                        i = i + int(date_data['activities-steps'][0]['value'])
            name.append(user_name + str(x))
            step.append(i)
```

=> 유저의 번호가 총 98 번까지 있기에 for 문을 98 번 돌리면서 폴더 안에 존재하는 유저를 찾는다. 유저가 있다면 if 문에 들어가고 없다면 다음 번호로 가서 유저의 번호 를 확인한다. 그리고 그 유저의 steps 파일의 존재 여부를 확인하고 4 월달 user 정보 가 없다면 else 로 가서 5 월 부터의 정보를 받아서 총 걸음수를 계산한다. 4 월 부터의 정보가 있다면 4 월 1 일 부터의 fitbit 정보를 불러와서 open 후 그 파일이 error 인지의
존재 여부를 list(date_data.keys()) == [‘errors’,‘success’] 가 false 이면 error 파일 이 아니기에 date_data[‘activities-steps’][0][‘value’]인 하루 총 걸음수를 I 에 저장하 면서 모든 날짜를 저장해서 그 값을 list 에 저장하고 한명의 user 의 총 걸음수를 저 장한 후 I 의 값을 0 으로 만들어 다음 user 의 총 걸음수를 list 에 저장시켜 나간다.

```
with open('step.csv','w',newline='₩n') as csvfile:
    fieldnames = ['name','total_steps']
    writer = csv.DictWriter(csvfile,fieldnames=fieldnames)
    writer.writeheader()
    for x in range(0,68):
        writer.writerow({'name' : str(name[x]), 'total_steps' : str(step[x])})
```

=> list 에 저장한 user 의 name 과 총 걸음수를 csv 파일에 DictWriter 와 writerow 를 이용해서 한 줄 한 줄 저장 시킨다

가공결과

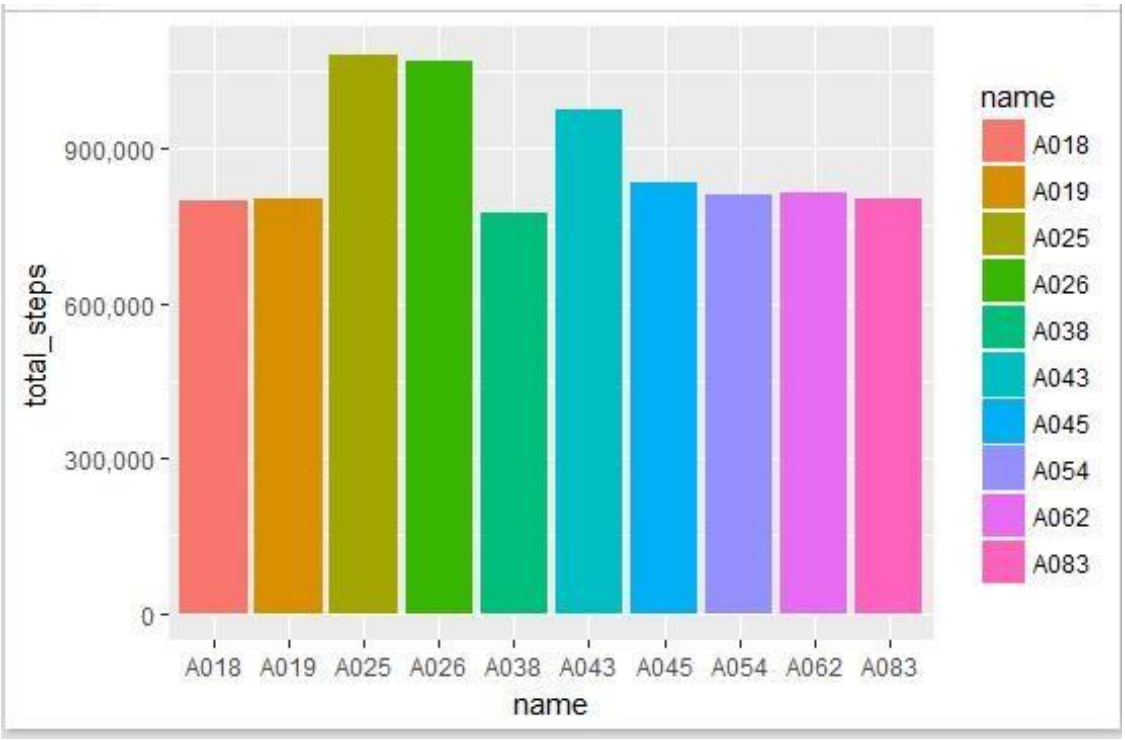
```
name,total_steps
A01,351518
A02,317873
A03,391302
A04,667043
A05,205815
A06,688864
A07,584462
A08,616758
A010,266725
A016,535526
A017,697669
A018,800509
A019,802587
A020,676258
```

2)시각화

```
1  library("ggplot2")
2  library("scales")
3
4  setwd('C:/Users/heebin/Documents/R/fitbit')
5  data <- read.csv('step.csv')
6
7  result <- data[order(-data$total_steps),]
8
9  top_user <- result[1:10,]
10
11 bar <- ggplot(top_user, aes(x=name, y=total_steps, fill=name)) + geom_bar(stat="identity") + scale_y_continuous(labels = comma)
12 print(bar)
13
```

=> 가공시킨 step.csv 파일을 읽은 후 총 발걸음 수를 중심으로 큰 순서로 sort 시킨 다. 그리고 top10 의 정보만 top_user 에 저장 시키고 geom_bar 형식으로 시각화를 시킨다.

3)시각화 그래프



2.가설 => 총 발걸음 수가 많을수록 운동을 많이 했을 것이다.

1)가공과정

```
import pandas as pd
import json
import os
import csv


dates = pd.date_range('20160401', '20160520')
dates05 = pd.date_range('20160501','20160520')
heart = []
name = []
step = []
user_name = 'A0'
num = 1
```

=> 유저의 name 과 총 발걸음 수 그리고 운동 시간을 저장하기 위한 list 를 만든다.
1 번 문제와 마찬가지로 5 월부터 있는 유저를 위해 date 범위를 둘로 만들었다.

```
for x in range(1,99):
    i = 0
    if os.path.exists('sokulee/A0'+str(x)) == True:
        if os.path.exists('sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + '20160401' + '_steps.json') == True:
            for date in dates:
                fname = 'sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + date.strftime('%Y%m%d') + '_steps.json'
                try: f = open(fname)
                except IOError as e:
                    print(str(e))
                else:
                    date_data = json.loads(f.read())
                    if (list(date_data.keys()) == ['errors', 'success'] or list(date_data.keys()) == ['success','errors']) == False :
                        i = i + int(date_data['activities-steps'][0]['value'])
        else :
            for date in dates05:
                fname = 'sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + date.strftime('%Y%m%d') + '_steps.json'
                try: f = open(fname)
                except IOError as e:
                    print(str(e))
                else:
                    date_data = json.loads(f.read())
                    if (list(date_data.keys()) == ['errors', 'success'] or list(date_data.keys()) == ['success','errors']) == False :
                        i = i + int(date_data['activities-steps'][0]['value'])
        name.append(user_name + str(x))
        step.append(i)
```

=> 1 번 문제와 마찬가지로 총 발걸음 수를 가져오기 위한 알고리즘. 유저의 파일 존 재 여부를 확인하고 4 월부터 있는지에 대한 존재 여부를 확인 하고 그 파일에 에러가 존재하는지를 확인한다. 그리고 그 유저의 하루 발걸음 수를 모두 더해서 list 에 저장 시킨다.

```
for x in range(1,99):
    i = 0
    if os.path.exists('sokulee/A0'+str(x)) == True:
        if os.path.exists('sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + '20160401' + '_heart.json') == True:
            for date in dates:
                fname = 'sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + date.strftime('%Y%m%d') + '_heart.json'
                try: f = open(fname)
                except IOError as e:
                    print(str(e))
                else:
                    date_data = json.loads(f.read())
                    if (list(date_data.keys()) == ['errors', 'success'] or list(date_data.keys()) == ['success','errors']) == False :
                        if(len(list(date_data['activities-heart'][0]['value']['heartRateZones'][2].keys())) == 3):
                            i = i + 0
                        else:
                            i = i + int(date_data['activities-heart'][0]['value']['heartRateZones'][2]['minutes'])
                        print(fname)
```

```
        else :
            for date in dates05:
                fname = 'sokulee/A0'+ str(x) + '/A0' + str(x) + '_' + date.strftime('%Y%m%d') + '_heart.json'
                try: f = open(fname)
                except IOError as e:
                    print(str(e))
                else:
                    date_data = json.loads(f.read())
                    if (list(date_data.keys()) == ['errors', 'success'] or list(date_data.keys()) == ['success','errors']) == False :
                        if(len(list(date_data['activities-heart'][0]['value']['heartRateZones'][2].keys())) == 3):
                            i = i + 0
                        else:
                            i = i + int(date_data['activities-heart'][0]['value']['heartRateZones'][2]['minutes'])
        name.append(user_name + str(x))
        heart.append(i)
```

=> 유저의 운동량을 체크하기 위한 알고리즘 heart 에 존재하는 heartRateZones 의 cario([2])의 시간을 통해서 운동을 한 시간을 체크 했다. 이를 찾기위해

date_data['activities-heart'][0]['value']['heartRateZones'][2]['minutes']을 이용해 서 그 유저의 하루하루 운동 시간을 저장해서 list 에 저장했다. 그런데 총 발걸음 수 와 마찬가지로 error 가 있는 파일과 5 월부터 있는 유저가 있어 if 문을 통해 존재 여 부를 확인하고 date_data['activities-heart'][0]['value']['heartRateZones'][2]에 존
재하는 key 값중 minutes 이 없고 max, min, name 만 존재하는 파일이 있었기에 그 킷값이 3 개이면 운동시간을 0 으로 저장하게 만들었다.

```
with open('heart.csv','w',newline='₩n') as csvfile:
    fieldnames = ['name','time','step']
    writer = csv.DictWriter(csvfile,fieldnames=fieldnames)
    writer.writeheader()
    for x in range(0,68):
        writer.writerow({'name' : str(name[x]), 'time' : str(heart[x]), 'step' : str(step[x])})
```

=> 마지막으로 csv 파일에 user 의 name 과 운동시간인 time, 총 발걸음 수인 step 을 DictWriter 와 writerow 를 이용해서 한 줄 한 줄 저장시킨다.

가공 결과

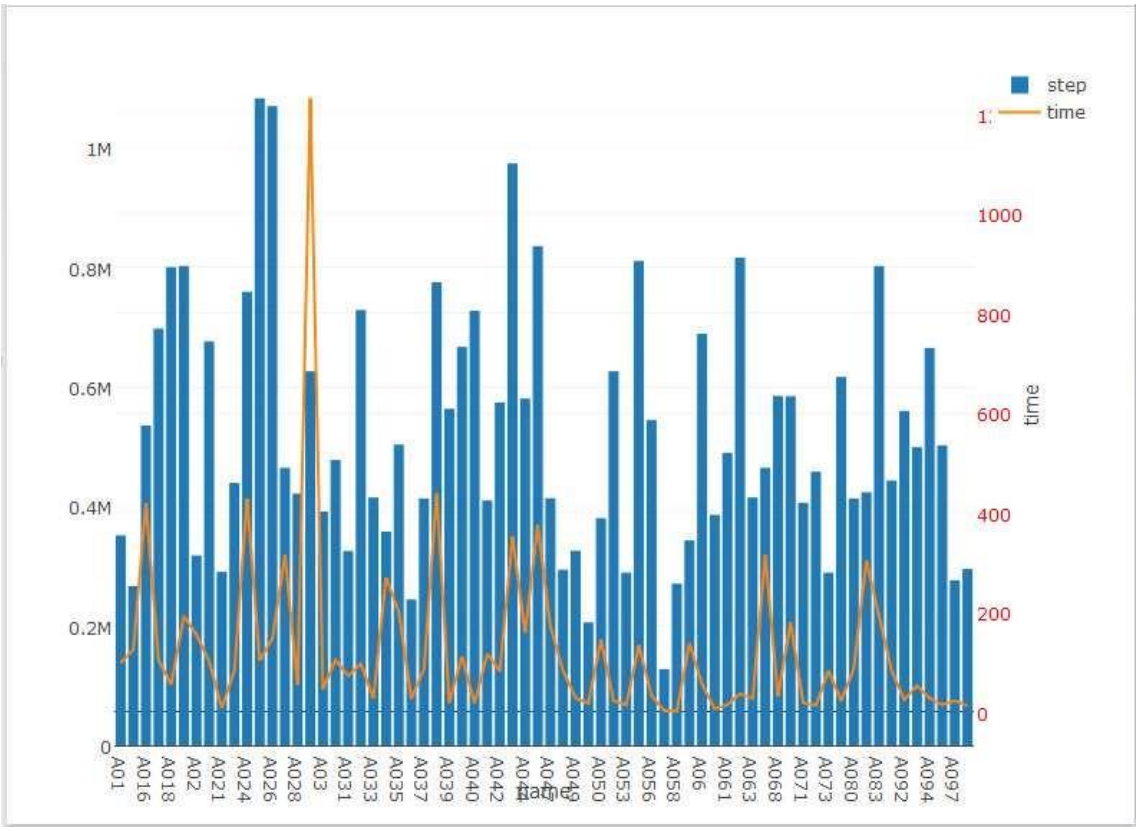
```
name,time,step
A01,99,351518
A02,155,317873
A03,46,391302
A04,112,667043
A05,18,205815
A06,56,688864
A07,180,584462
A08,23,616758
A010,125,266725
A016,419,535526
A017,104,697669
A018,56,800509
A019,193,802587
A020,102,676258
A021,9,290324
A022,83,439406
A024,428,759414
```

2) 시각화

```
library("ggplot2")
library("scales")
library("plotly")
setwd('C:/Users/heebin/Documents/R/fitbit')
result <- read.csv('heart.csv')
ay <- list(tickfont = list(color = "red"), overlaying = "y", side = "right", title = "time")
p <- plot_ly() %>%
  add_bars(x = result$name, y = result$step, name="step") %>%
  add_lines(x = result$name, y = result$time, name="time", yaxis="y2")%>%
  layout(yaxis2 = ay, xaxis=list(title="name"))
print(p)
```

=> plotly 를 이용해 총 발걸음수와 운동시간을 나타내는 그래프를 결합시켰다.
plot_ly()함수를 이용해서 add_bars()함수를 이용해 총 발걸음 수를 막대그래프로 나 타내고 add_lines()함수를 이용해 운동시간을 선 그래프로 한 그래프 안에 나타내게
하였는데 좌측 y 축이 총 발걸음 수 우측 y 축이 운동시간이며 x 축이 유저의 name 이 다

3)운동시간과 발걸음 수의 관계



=> 그래프를 보면 발걸음 수가 많지만 그에 비해 운동 시간이 더 작고 발걸음 수가 적지만 그에 비해서 운동 시간이 굉장히 많은 것을 확인 할 수 있다. 비교적 발걸음 이 많은 유저들의 운동 시간이 더 작고 발걸음 수는 많지 않지만 운동 시간이 탁월하 게 높은 사람들이 많이 있었다. 이를 통해 가설이었던 발걸음 수가 많으면 운동시간 도 많을 것이다 라는 가설은 거짓이라는 것을 알게 되었다.

Fitbit data manufacture

전체 사용자에 대해서 step, sleep, heart 데이터를 1일 평균으로 구함

```
import pandas as pd
import json
import os
import csv
```

```
def readFile(filename):
    f = open(filename, 'r')
    js = json.loads(f.read())
    f.close()
    return js
```

```
steps = []
day = []
sleep = []
hearts = []
```

```
dates = pd.date_range("20160401","20160520")
for date in dates :
    personal_steps = 0
    personal_sleep = 0
    personal_hearts = 0
    for person_number in range(1,98): #실제는 68명
        try:
            str_step = "sokulee/A0"+str(person_number)+"/A0"+str(person_num
            str_sleep = "sokulee/A0"+str(person_number)+"/A0"+str(person_nu
            str_hearts = "sokulee/A0"+str(person_number)+"/A0"+str(person_n
            day_steps = readFile(str_step)
            day_sleep = readFile(str_sleep)
            day_hearts = readFile(str_hearts)
            try:
                personal_steps = personal_steps + int(day_steps['activities
                personal_sleep = personal_sleep + int(day_sleep['summary'][
                personal_hearts = personal_hearts + int(day_hearts["activit
            except KeyError:
                continue
        except FileNotFoundError:
            continue
    str_tmp = "20"+str(date.strftime("%y%m%d"))
    day.append(str_tmp)
    steps.append(personal_steps/68)
    sleep.append(personal_sleep/68)
    hearts.append(personal_hearts/68)
```

```
total_day = pd.DataFrame({'Date' : day ,'step_count' : steps, 'sleep_time'
total_day
```

	Date	sleep_time	step_count	wear_time
0	20160401	336.382353	11254.750000	131.058824
1	20160402	319.073529	9983.808824	100.588235
2	20160403	218.132353	4636.411765	45.250000
3	20160404	321.191176	15981.073529	149.573529
4	20160405	321.235294	13701.308824	149.279412
5	20160406	331.176471	12669.882353	113.705882
6	20160407	323.235294	13513.985294	113.308824
7	20160408	366.073529	15802.529412	159.764706
8	20160409	361.750000	13777.676471	141.102941
9	20160410	405.029412	8412.647059	99.676471
10	20160411	337.044118	12302.764706	102.014706

```
filename = 'total_day.csv'
total_day.to_csv(filename, index=False, encoding='utf-8')
```

일, 시간 heatmap

```
total_step = [0] * 168
```

```
dates = pd.date_range("20160401","20160520")
for date in dates :
    for person_number in range(1,98): #실제는 68명
        try:
            str_step = "sokulee/A0"+str(person_number)+"/A0"+str(person_num
            print(str_step)
            day_steps = readFile(str_step)
            try:
                now_date = day_steps['activities-steps'][0]['dateTime']

                now_weekday = pd.to_datetime(now_date).weekday()

                intraday = day_steps['activities-steps-intraday']['dataset'
                for minute in intraday:
                    if int(minute['value']) != 0:
                        total_step[(now_weekday * 24) + int(minute['time'][
            except KeyError:
                continue
        except FileNotFoundError:
            continue
    str_tmp = "20"+str(date.strftime("%y%m%d"))
```

```
weekday = []
hour = []
for i in range(7):
    for j in range(24):
        weekday.append(i)
        hour.append(j)
```

```
data = pd.DataFrame({'weekday' : weekday ,'hour' : hour, 'step' : total_ste
```

```
filename = 'heatmap_day.csv'
data.to_csv(filename, index=False, encoding='utf-8')
```

Fitbit spark 유클리디안, K-means

top step 과 top sleep 의 유클리디안으로 5명 구하기

```
data = pd.DataFrame({"user" : user , "steps" : steps , "sleep" : sleep}).ro
```

```
data.sort_values('steps', ascending=False, inplace=True)
top_step_data = data[:1]
top_step_data
```

	sleep	steps	user
34	271.82	21666.4	A025

```
data.sort_values('sleep', ascending=False, inplace=True)
top_sleep_data = data[:1]
top_sleep_data
```

	sleep	steps	user
65	471.37	9293.2	A027

```
import math

def euclidean_distance(a, b):
    return math.sqrt( (a['sleep'] - b['sleep'])**2 +
                      (a['steps'] - b['steps'])**2 )

def norm_euclidean_distance(a, b):
    return 1 / (euclidean_distance(a, b) + 1)
```

```
top5_step = []
top5_sleep = []
```

```
for index, row in data.iterrows():
    dist = norm_euclidean_distance(top_sleep_data, row)
    top5_sleep.append(dist)
```

```
for index, row in data.iterrows():
    dist = norm_euclidean_distance(top_step_data, row)
    top5_step.append(dist)
```

```
data['top5_step'] = top5_step
```

```
data['top5_sleep'] = top5_sleep
```

```
data.sort_values('top5_step', ascending=False, inplace=True)
top_sleep_data = data[1:6]
top_sleep_data
```

	sleep	steps	user	top5_step	top5_sleep
20	338.86	21405.82	A026	0.003703	0.000083
30	360.76	19486.72	A043	0.000458	0.000098
44	372.76	16718.14	A045	0.000202	0.000135
29	213.18	16331.34	A062	0.000187	0.000142
42	342.38	16217.22	A054	0.000183	0.000144

```
data.sort_values('top5_sleep', ascending=False, inplace=True)
top_sleep_data = data[1:6]
top_sleep_data
```

	sleep	steps	user	top5_step	top5_sleep
41	471.37	9293.20	A065	0.000081	1.000000
62	421.60	9161.90	A072	0.000080	0.007071
55	253.49	9554.96	A030	0.000083	0.002928
60	249.44	8866.20	A084	0.000078	0.002074
58	426.96	8788.12	A022	0.000078	0.001968

k mean 클러스터링

```
data = pd.DataFrame({"user" : user , "steps" : steps , "sleep" : sleep, "he
```

```
k_data = pd.DataFrame({"steps" : steps , "sleep" : sleep, "hearts" : hearts
```

```
data
```

	sleep	steps	user	top5_step	top5_sleep
65	471.37	9293.20	A027	0.000081	1.000000
41	471.37	9293.20	A065	0.000081	1.000000
67	442.56	8274.74	A047	0.000075	0.000981
0	427.82	6516.66	A049	0.000066	0.000360
9	427.12	5413.86	A058	0.000062	0.000258
58	426.96	8788.12	A022	0.000078	0.001968
62	421.60	9161.90	A072	0.000080	0.007071
22	402.78	8267.06	A037	0.000075	0.000971
52	401.37	8428.42	A028	0.000076	0.001151
2	399.06	11698.64	A068	0.000100	0.000415
21	398.69	5806.48	A021	0.000063	0.000287

```
k_data['hearts'] = k_data['hearts'].map('1:{:.2f}'.format)
k_data['sleep'] = k_data['sleep'].map('2:{:.2f}'.format)
k_data['steps'] = k_data['steps'].map('3:{:.2f}'.format)
k_data
```

```
#k_mean를 하기위해 txt 파일로 저장하는 것
k_data.to_csv('avr.txt', sep=' ', header=False)
```

k_mean 하기

```
from pyspark.ml.clustering import KMeans
```

```
dataset = spark.read.format("libsvm").load('avr.txt')
```

```
# 뭉쳐져있는 각각의 data 중에서 중심인 4개를 뽑아냄
#setK 의 값에 따라 중심 값의 수를 뽑아낸다.
# k means 하기
kmeans = KMeans().setK(4).setSeed(1)
model = kmeans.fit(dataset)

# Evaluate clustering by computing Within Set Sum of Squared Errors.
wssse = model.computeCost(dataset)
print("Within Set Sum of Squared Errors = " + str(wssse))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Within Set Sum of Squared Errors = 99560628.37097663
Cluster Centers:
[ 1107.633    277.322   6225.785]
[  1291.73333333    323.81333333  20852.98      ]
[ 1203.31814815   314.28666667  9630.59851852]
[  1238.18611111    326.47611111  14611.09      ]
```

유클리디안 하기

```
me = pd.Series([900, 500, 5000, 'A000'], index = ['hearts' , 'sleep', 'step
```

```
import math

def euclidean_distance(a, b):
    return math.sqrt( (a['sleep'] - b['sleep'])**2 +
                      (a['steps'] - b['steps'])**2 +
                      (a['hearts'] - b['hearts'])**2  )

def norm_euclidean_distance(a, b):
    return 1 / (euclidean_distance(a, b) + 1)
```

```
normoalized_dist = -1
nearest_user = 'A00'
for index, row in data.iterrows():
    dist = norm_euclidean_distance(me, row)
    if normoalized_dist < dist:
        normoalized_dist = dist
        nearest_user = row['user']

print("Nearest user is %s, normalized_dist = %.10f" %(nearest_user, normoal
```

```
Nearest user is A010, normalized_dist = 0.0022634106
```

내가 속한 클러스터는 무엇인가?!?!?

```
import numpy as np
me = np.array([900, 500, 5000])
```

```
import math

def euclidean_distance(a, b):
    return math.sqrt( (a[0] - b[0])**2 +
                      (a[1] - b[1])**2 +
                      (a[2] - b[2])**2  )

def norm_euclidean_distance(a, b):
    return 1 / (euclidean_distance(a, b) + 1)
```

```
normoalized_dist = -1
nearest_center = []
for center in centers:
    dist = norm_euclidean_distance(me, center)
    if normoalized_dist < dist:
        normoalized_dist = dist
        nearest_center = center

print("my 클러스터 :", nearest_center)
```

```
my 클러스터 : [ 1107.633   277.322  6225.785]
```