Report

Class 데이터과학

Title | Fitness Trecker데이터 분석

학 과 명	컴퓨터공학과
교 수 명	이영석 교수님
학 번	201601989
이 름	김 진 섭
제 출 일	2019. 05.27

충남대학교
Chungnam National University

목 차

<과제 목표>

데이터
- 2016년 소쿠리 분석대회의 68명의 건강 데이터
- 4월 1일 ~ 5월 20일까지의 데이터
- 요일 별 심박수와 수면량 그리고 걸음수

과제목표 1
- 소쿠리 대회 데이터를 가공

과제목표 2
- 평균 걸음수 : 걸음수 10000보 목표를 달성 못한 사람은 얼마나 있는가? 그래프를 그려 확인
- 평균 수면량 : 성인 하루 권장 수면시간 6시간을 지키지 못한 사람이 얼마나 있는가? 그래프를 그려서 확인
- 하루 평균 심박수가 60 이하인 사람들이 얼마나 있는가? 그래프를 그려서 확인

과제목표 3
- 모든 학생들의 날짜별 평균 걸음수와 수면량의 날씨(최고온도, 총 강수량, 평균 습도, 평균 풍속) 및 요일의 상관관계 파악
- seaborn 라이브러리를 활용하여 요일별 STEP과 수면량 그래프로 표현
- 기상 데이터를 활용하여 좌 우 측에 온도와 Step을 표시하고 x축을 Date로 하여 그래프를 그리기
- 동일하게 좌 우측에 각각 강수량과 Step을 표시하고 x축을 Date로 하여 그래프로 표현하기

과제목표 4
- Fitbit 하루 이용자의 변화를 분석하기

<Preprocessing>

전처리를 수행하기 위하여 저는 os의 listdir로 해당 폴더에 있는 모든 파일 리스트를 가져오고 그 파일들을 json으로 열어서 데이터들을 전처리 수행하여 주었다.

```
import json,os
import pandas as pd
```

전처리를 위하여 필요한 모듈을 모두 import하여온다.

```
H_UserName = []
H_Date = []
H_Time = []
H_Value = []

S_UserName =[]
S_Date = []
S_Time = []
S_Hour = []
S_Minute = []
S_Steps = []

SL_UserName =[]
SL_Date = []
SL_StartTime = []
SL_MinutesASleep =[]
SL_MinutesAWake =[]
SL_TimeInBed = []
```

데이터프레임을 만들어주기 위하여 각각의 정제해야하는 데이터의 변수들을 생성합니다. H_가 붙은 데이터는 심박수 데이터, S_는 Step 데이터, SL_는 수면 데이터입니다.

```
FL = [x for x in os.listdir('sokulee/') if x.find("A") != -1 ]
```

os,listdir을 활용하여 sokulee에 있는 데이터들을 가져옵니다. 데이터들의 이름이 모두 A로 시작하는데, DB_~라는 파일이 있기 때문에, 위처럼 A가 들어 있는 파일명만 가져와서 FL에 넣어줍니다.

```
for Name in FL:
    DL = [x for x in os.listdir('sokulee/'+Name) if x.find("A") != -1 ]
```

반복문의 시작단계로 FL의 데이터들을 사용하여 반복문을 실행합니다. DL에 넣는 데이터는 FL 파일 내에 있는 리스트들로, json파일들임을 알 수 있습니다.

```
    for AllData in DL:
        with open('sokulee/'+Name+'/'+AllData) as json_file:
            json_data = json.load(json_file)
```

위 for문의 내부의 for문입니다. DL을 이용하여 돌려주는데, open을 활용하여 해당 json_file을 열어줍니다. 그리고, json.load를 이용해 json_file을 json 읽기를 수행합니다.

```
        if 'errors' in json_data.keys():
            continue
```

위 for문의 내부에 있는 것입니다. json파일을 읽었을 때, 만약 에러가 있는 데이터라면 error라는 key를 가지고 있는 경우가 있기 때문에 예외처리로 위와 같은경우에는 해당 json 파일을 건너뛰기 위하여 continue를 사용하였습니다.

```
        if AllData.split('_')[2] == 'heart.json' :
            HeartData = json_data['activities-heart-intraday']['dataset']
            i = 0
            while i < len(HeartData) :
                H_UserName.append(Name)
                H_Date.append(json_data['activities-heart'][0]['dateTime'])
                H_Time.append(HeartData[i]['time'])
                H_Value.append(HeartData[i]['value'])
                i = i + 1
```

심장수에 대한 데이터일 경우 처리하는 내용입니다. HeartData에 json파일의 데이터셋을 넣

어주고, 해당 데이터 셋의 길이만큼 반복하여줍니다. 데이터 셋에는 1분당 심장 박동수를 체크한 정보가 들어있습니다. 반복을 하며, 사용자 이름, 날짜, 몇 분인가에 대한 time, 그리고 심장 박동수(value)를 list에 append하여주는 과정을 수행합니다.

```
            if AllData.split('_')[2] == 'steps.json' :
                StepData = json_data['activities-steps-intraday']['dataset']
                i = 0
                while i < len(StepData) :
                    S_UserName.append(Name)
                    S_Date.append(json_data['activities-steps'][0]['dateTime'])
                    S_Time.append(StepData[i]['time'])
                    S_Hour.append(int(StepData[i]['time'].split(':')[0]))
                    S_Minute.append(int(StepData[i]['time'].split(':')[1]))
                    S_Steps.append(StepData[i]['value'])
                    i = i + 1
```

두 번째로 step 데이터에 대한 처리입니다. 우선 step 데이터의 dataset을 가져오고, 이 또한 1분당 걸음수를 측정한 데이터가 들어있기 때문에 while문으로 돌아가며 필요한 정보를 넣어줍니다.

```
            if AllData.split('_')[2] == 'sleep.json' :
                SleepData = json_data['sleep']
                if len(SleepData) > 0 :
                    SL_UserName.append(Name)
                    SL_Date.append(SleepData[0]['dateOfSleep'])
                    SL_StartTime.append(SleepData[0]['startTime'].split('T')[1][:8])
                    SL_MinutesASleep.append(SleepData[0]['minutesAsleep'])
                    SL_MinutesAWake.append(SleepData[0]['minutesAwake'])
                    SL_TimeInBed.append(SleepData[0]['timeInBed'])
```

마지막으로 sleep 데이터인데, 위와 동일하게 처리를 해주돼, sleep 데이터는 하루에 1회, 낮잠을 포함하면 2회 이상의 데이터가 들어갑니다. 과제에서 요구하는 sleep에 대한 데이터는 하루당 하나를 요구하고 있으므로, 이를 위해서, SleepData[0]으로 접근하여 데이터를 가져옵니다.

```
H_UserName = []
H_Date = []
H_Time = []
H_Value = []

S_UserName =[]
S_Date = []
S_Time = []
S_Hour = []
S_Minute = []
S_Steps = []

SL_UserName =[]
SL_Date = []
SL_StartTime = []
SL_MinutesASleep =[]
SL_MinutesAWake =[]
SL_TimeInBed = []

FL = [x for x in os.listdir('sokulee/') if x.find("A") != -1 ]

for Name in FL:
    DL = [x for x in os.listdir('sokulee/'+Name) if x.find("A") != -1 ]
    for AllData in DL:
        with open('sokulee/'+Name+'/'+AllData) as json_file:
            json_data = json.load(json_file)
            if 'errors' in json_data.keys():
                continue
            if AllData.split('_')[2] == 'heart.json' :
                HeartData = json_data['activities-heart-intraday']['dataset']
                i = 0
                while i < len(HeartData) :
                    H_UserName.append(Name)
                    H_Date.append(json_data['activities-heart'][0]['dateTime'])
                    H_Time.append(HeartData[i]['time'])
                    H_Value.append(HeartData[i]['value'])
                    i = i + 1
            if AllData.split('_')[2] == 'steps.json' :
                StepData = json_data['activities-steps-intraday']['dataset']
                i = 0
                while i < len(StepData) :
                    S_UserName.append(Name)
                    S_Date.append(json_data['activities-steps'][0]['dateTime'])
                    S_Time.append(StepData[i]['time'])
                    S_Hour.append(int(StepData[i]['time'].split(':')[0]))
                    S_Minute.append(int(StepData[i]['time'].split(':')[1]))
                    S_Steps.append(StepData[i]['value'])
                    i = i + 1
            if AllData.split('_')[2] == 'sleep.json' :
                SleepData = json_data['sleep']
                if len(SleepData) > 0 :
                    SL_UserName.append(Name)
                    SL_Date.append(SleepData[0]['dateOfSleep'])
                    SL_StartTime.append(SleepData[0]['startTime'].split('T')[1][:8])
                    SL_MinutesASleep.append(SleepData[0]['minutesAsleep'])
                    SL_MinutesAWake.append(SleepData[0]['minutesAwake'])
                    SL_TimeInBed.append(SleepData[0]['timeInBed'])
```

*위의 띄어쓰기 없이 나열되어 있는 설명은 해당코드를 분할하여 설명한 것입니다.

```
df_Heart = pd.DataFrame()
df_Heart['USERNAME'] = H_UserName
df_Heart['DATE'] = H_Date
df_Heart['TIME'] = H_Time
df_Heart['VALUE'] = H_Value
df_Heart.head(5)
```

	USERNAME	DATE	TIME	VALUE
0	A01	2016-04-01	00:00:00	79
1	A01	2016-04-01	00:01:00	80
2	A01	2016-04-01	00:02:00	78
3	A01	2016-04-01	00:03:00	73
4	A01	2016-04-01	00:04:00	77

Heart 데이터를 데이터프레임으로 생성하여줍니다.

```
df_Step = pd.DataFrame()
df_Step['USERNAME'] = S_UserName
df_Step['DATE'] = S_Date
df_Step['TIME'] = S_Time
df_Step['HOUR'] = S_Hour
df_Step['MINUTE'] = S_Minute
df_Step['STEPS'] = S_Steps
df_Step.head()
```

	USERNAME	DATE	TIME	HOUR	MINUTE	STEPS
0	A01	2016-04-01	00:00:00	0	0	0
1	A01	2016-04-01	00:01:00	0	1	0
2	A01	2016-04-01	00:02:00	0	2	0
3	A01	2016-04-01	00:03:00	0	3	0
4	A01	2016-04-01	00:04:00	0	4	0

Step 데이터를 데이터프레임으로 생성하여줍니다.

```
df_Sleep = pd.DataFrame()
df_Sleep['USERNAME'] = SL_UserName
df_Sleep['DATE'] = SL_Date
df_Sleep['STARTTIME'] = SL_StartTime
df_Sleep['MINUTESASLEEP'] = SL_MinutesASleep
df_Sleep['MINUTESAWAKE'] = SL_MinutesAWake
df_Sleep['TIMEINBED'] = SL_TimeInBed
df_Sleep.head()
```

	USERNAME	DATE	STARTTIME	MINUTESASLEEP	MINUTESAWAKE	TIMEINBED
0	A01	2016-04-01	02:39:00	485	26	511
1	A01	2016-04-02	02:09:00	512	37	549
2	A01	2016-04-04	01:52:30	335	11	346
3	A01	2016-04-05	02:04:00	311	24	335
4	A01	2016-04-06	01:49:00	491	59	551

Sleep 데이터를 데이터 프레임으로 생성하여줍니다.

```
df_Heart.to_csv("./data/all_user_hearts.csv", index = False)
df_Step.to_csv("./data/all_user_steps.csv", index = False)
df_Sleep.to_csv("./data/all_user_sleeps.csv", index = False)
```

위에서 만든 데이터 프레임을 csv파일로 저장합니다.

< HW 1-a >

*과제를 하나의 파일내에서 모두 수행하였기 때문에 중복되는 작업은 생략될 수 있습니다.

```
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import seaborn as sns
```

과제를 수행하기 위해 필요한 패키지들을 import하여준다.

[1-a-i] 평균 걸음수 분포

```
df_s = pd.read_csv('./data/all_user_steps.csv')
df_s = df_s.fillna(0)
df_s.head(3)
```

	USERNAME	DATE	TIME	HOUR	MINUTE	STEPS
0	A01	2016-04-01	00:00:00	0	0	0
1	A01	2016-04-01	00:01:00	0	1	0
2	A01	2016-04-01	00:02:00	0	2	0

걸음수 데이터를 read_csv로 읽어온다.

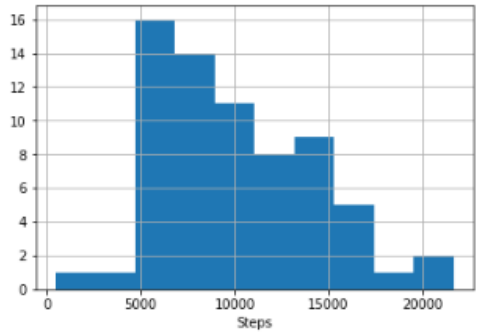
```
StepData = df_s.groupby(['USERNAME', 'DATE'])['STEPS'].sum().reset_index()
StepData = StepData.groupby(['USERNAME'])['STEPS'].mean().reset_index()
StepData.head()
```

	USERNAME	STEPS
0	A01	6771.02
1	A010	5254.30
2	A016	10700.10
3	A017	13946.88
4	A018	15999.28

걸음수 데이터를 groupby를 활용하여 사용자별 하루 걸음수로 만들어주고, 그 데이터들을 또 groupby하여 사용자별 평균 걸음수로 바꾸어준다.

```
StepData['STEPS'].hist()
plt.xlabel('Steps')
plt.show()
```

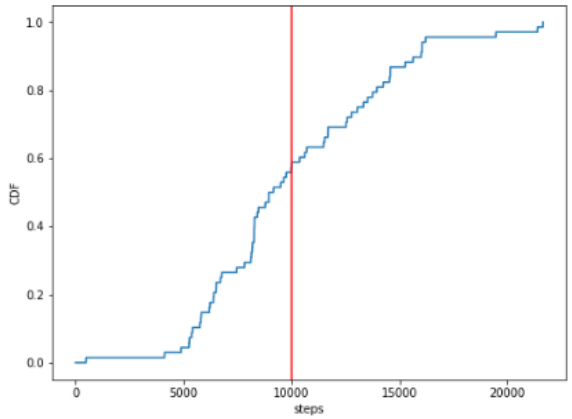
위에서 생성한 데이터로 그래프를 그려준다.



생성된 그래프 결과인데, x축은 사람들의 평균 걸음수 y축은 사람의 수이다.
5000~10000걸음의 사이의 사람들이 가장 많음을 확인할 수 있다.

```
steps_list = [int(steps) for steps in StepData['STEPS']]
steps_list_count = np.zeros(max(steps_list)+1)
for steps in steps_list:
    steps_list_count[int(steps)] +=1
steps_cumsum = steps_list_count.cumsum()
steps_cdf = [cumsum/steps_cumsum[-1] for cumsum in steps_cumsum]
print('평균 걸음수 10,000보를 달성하지 못한 경우 : {0}%'.format(round(steps_cdf[10000]*100,2)))
plt.figure(figsize=(8,6))
plt.plot(steps_cdf)
plt.axvline(x = 10000, color = 'r')
plt.xlabel('steps')
plt.ylabel('CDF')
plt.show()
```

```
평균 걸음수 10,000보를 달성하지 못한 경우 : 57.35%
```



누적분포함수를 사용하여 10,000보를 달성하지 못한 경우를 뽑아내어 보았다. 우선, 평균 걸음수가 만보를 달성하지 못한 경우를 위해 steps_cdf를 생성하여주었다. 그리고, 만보를 달성못한 사람들을 소수점 둘째자리 수 까지 계산하여주고, CDF그래프를 그려주었다.
그래프를 해석하면 y축은 %이고, x축은 평균 걸음수 이다. 걸음수가 10,000일 경우의 y값을 읽어주게 되면 걸음수가 10,000보 이하인 사람들의 수가 %로 나오게 된다.
따라서, 위 빨간색 선과 그래프가 만나는 지점의 y값이 10000보를 달성하지 못한 경우이다.
대략 57.35%가 나온다.

[1-a-ii] 평균 수면량 분포

```
df_SL = pd.read_csv('./data/all_user_sleeps.csv')
df_SL = df_SL.fillna(0)
df_SL.head(3)
```

	USERNAME	DATE	STARTTIME	MINUTESASLEEP	MINUTESAWAKE	TIMEINBED
0	A01	2016-04-01	02:39:00	485	26	511
1	A01	2016-04-02	02:09:00	512	37	549
2	A01	2016-04-04	01:52:30	335	11	346

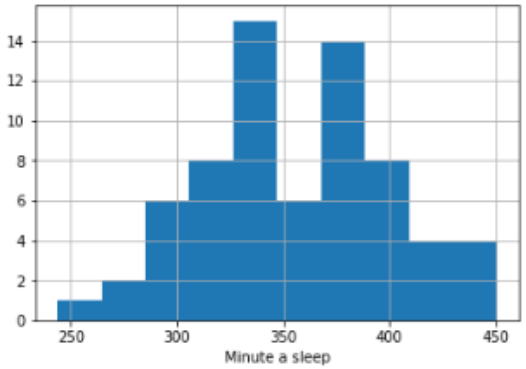
평균 수면량에 대한 분포를 그려주기 위하여 sleep 데이터를 가져온다.

```
SleepData = df_SL.groupby(['USERNAME', 'DATE'])['MINUTESASLEEP'].sum().reset_index()
SleepData = SleepData.groupby(['USERNAME'])['MINUTESASLEEP'].mean().reset_index()
SleepData.head()
```

	USERNAME	MINUTESASLEEP
0	A01	371.357143
1	A010	420.733333
2	A016	303.900000
3	A017	343.816327
4	A018	370.217391

수면량도 위 Step과 같이 계산하여 사람들 각각의 하루 평균 수면량을 계산하여준다.

```
SleepData['MINUTESASLEEP'].hist()
plt.xlabel('Minute a sleep')
plt.show()
```

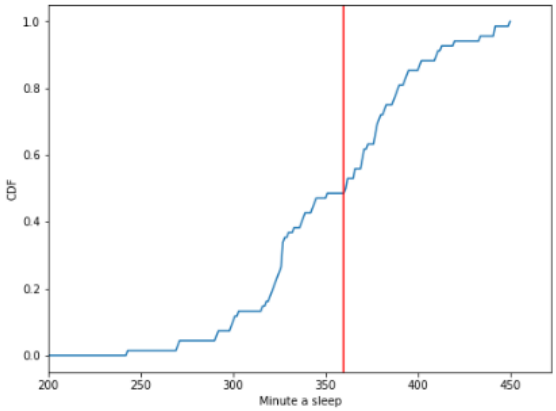


수면량에 대한 그래프를 그려준다.
x축은 수면을 분단위로 나타내어준 것이고, y축은 사람수를 의미한다. 약 340분정도의 평균 수면 시간을 갖는 사람들이 많은 것으로 확인할 수 있다.

```
sleep_list = [int(sleep) for sleep in SleepData['MINUTESASLEEP']]
sleep_list_count = np.zeros(max(sleep_list) + 1)
for sleep in sleep_list:
    sleep_list_count[int(sleep)] += 1
sleep_cumsum = sleep_list_count.cumsum()
sleep_cdf = [cumsum/sleep_cumsum[-1] for cumsum in sleep_cumsum]
print('평균 수면 시간이 6시간이 안되는 경우 : {0}%'.format(round(sleep_cdf[360]*100,2)))
plt.figure(figsize = (8,6))
plt.plot(sleep_cdf)
plt.axvline(x = 360, color = 'r')
plt.xlim(200)
plt.xlabel('Minute a sleep')
plt.ylabel('CDF')
plt.show()
```

위 Step과 동일하게 평균 수면이 6시간이 안되는 사람을 확인하기 위하여 sleep_cdf를 생성하여준다. 그리고 6시간이 안돼는 경우를 cdf 그래프에서 데이터를 가져와 %로 그리고 소수점 2째 자리까지 나타내어준다.
그리고 CDF 그래프를 그리고 해당 그래프의 360분이 되는 지점에 빨간색 선으로 표시하여 주었다.

```
평균 수면 시간이 6시간이 안되는 경우 : 48.53%
```



위에서 그린 그래프이다. 평균 수면 시간이 6시간이 안되는 경우는 약 48.53%이다.
위 step의 누적분포함수와 동일하게 x축은 수면량을 분단위로 나타내어준 것이고, y축은 %를 나타내어준 것이다. x축이 360이 되는 지점에 선을 긋고 그 지점과 만나는 지점의 값을 읽어주면 수면시간이 6시간이하인 사람들의 수가 %로 나타내어지게 되는데 그것이 약 48.53%임을 알 수 있다.

[1-a-iii] 평균 심박수 분포

```
df_H = pd.read_csv('./data/all_user_hearts.csv')
df_H = df_H.fillna(0)
df_H.head(3)
```

	USERNAME	DATE	TIME	VALUE
0	A01	2016-04-01	00:00:00	79
1	A01	2016-04-01	00:01:00	80
2	A01	2016-04-01	00:02:00	78

심박수 분포를 계산하기 위하여 데이터를 가져온다.

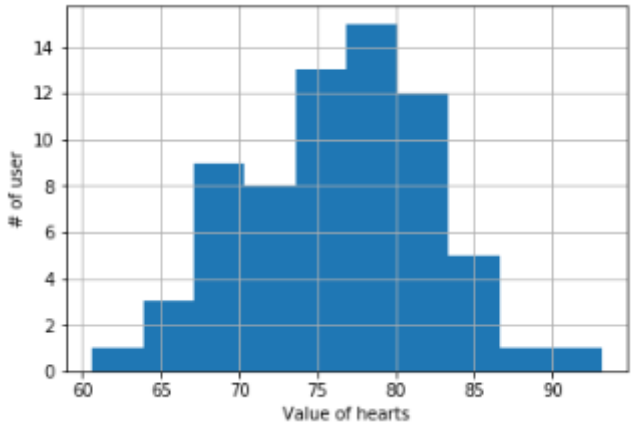
```
HeartData = df_H.groupby(['USERNAME'])['VALUE'].mean().reset_index()
HeartData.head()
```

	USERNAME	VALUE
0	A01	81.620110
1	A010	75.746871
2	A016	83.961060
3	A017	77.922819
4	A018	74.145573

심박수 데이터들을 가져와 사람들의 하루 평균 심박수를 측정하여준다.

```
HeartData['VALUE'].hist()
plt.xlabel('Value of hearts')
plt.ylabel('# of user')
plt.show()
```

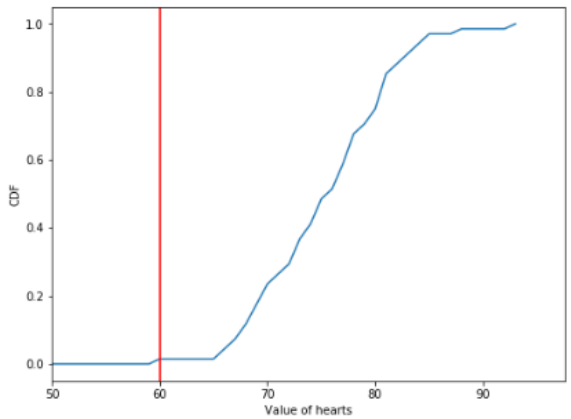
만들어진 사람들의 하루 평균 심박수 데이터를 그래프로 표현하여준다.



생성된 그래프이다. x축은 사람들의 심박수이고 y축은 사람의 수이다. 약 75~80정도의 평균 심박수를 가지고 있는 사람들이 가장 많음을 확인할 수 있다.

```
HeartValue_list = [int(Value) for Value in HeartData['VALUE']]
HeartValue_list_count = np.zeros(max(HeartValue_list)+1)
for Value in HeartValue_list:
    HeartValue_list_count[int(Value)] +=1
HeartValue_cumsum = HeartValue_list_count.cumsum()
HeartValue_cdf = [cumsum/HeartValue_cumsum[-1] for cumsum in HeartValue_cumsum]
print('하루 평균 심박수가 60이하인 사람들 : {0}%'.format(round(HeartValue_cdf[60]*100,2)))

plt.figure(figsize=(8,6))
plt.plot(HeartValue_cdf)
plt.axvline(x = 60, color = 'r')
plt.xlim(50)
plt.xlabel('Value of hearts')
plt.ylabel('CDF')
plt.show()
```

```
하루 평균 심박수가 60이하인 사람들 : 1.47%
```



위와 동일한 이유로, cdf 그래프를 생성하여주고, 심박수가 60이하인 사람들을 나타내어주었다. 위 그래프 역시 CDF이므로 60인 지점의 y값을 보게되면, 60이하인 사람들의 %를 알 수 있다.

그래프의 x축은 심박수, y축은 사람들의 누적 분포이다. 따라서, x = 60인 지점에 선을 긋고 그 선과 만나는 지점의 y값을 읽어주면 하루 평균 심박수가 60 이하인 사람을 확인할 수있고, 그 값은 약 1.47%가 된다.

< HW 1-b >

[1-b] 걸음, 수면 상관관계 요인

모든 학생들의 각 날짜별 평균 걸음수와 수면량의 날씨 및 요일의 상관관계를 파악하는 과제이므로 모든 학생들의 걸음수와 수면량에 대한 평균을 날짜별로 나타낸 데이터를 생성하여야 합니다.

```
df_Step = pd.read_csv('./data/all_user_steps.csv')
df_Step = df_Step.groupby(['USERNAME','DATE'])['STEPS'].sum().reset_index()
df_Step = df_Step.groupby(['DATE'])['STEPS'].mean().reset_index()
df_Step.head()
```

	DATE	STEPS
0	2016-04-01	11632.500000
1	2016-04-02	10662.938462
2	2016-04-03	7870.104478
3	2016-04-04	16206.477612
4	2016-04-05	13893.089552

걸음 수 데이터를 가져와서, 사람들의 하루 걸음 데이터를 만들어주고, 그 데이터를 날짜별로 평균을 내어줍니다.

```
df_Sleep = pd.read_csv('./data/all_user_sleeps.csv')
df_Sleep = df_Sleep.groupby(['DATE'])['MINUTESASLEEP'].mean().reset_index()
df_Sleep.head()
```

	DATE	MINUTESASLEEP
0	2016-04-01	343.771930
1	2016-04-02	336.525424
2	2016-04-03	422.937500
3	2016-04-04	359.189655
4	2016-04-05	335.931034

수면 데이터를 가져와서, 날짜별로 사람들의 수면평균에 대한 정보를 만들어 줍니다.

```
df = pd.merge(df_Step, df_Sleep, on = ['DATE'])
df.head()
```

	DATE	STEPS	MINUTESASLEEP
0	2016-04-01	11632.500000	343.771930
1	2016-04-02	10662.938462	336.525424
2	2016-04-03	7870.104478	422.937500
3	2016-04-04	16206.477612	359.189655
4	2016-04-05	13893.089552	335.931034

df에 두 개의 데이터를 날짜별로 병합하여 넣어줍니다.

```
df_Weather = pd.read_csv('./data/sokulee_weather.csv')
dt = df_Weather['DATETIME']
datetime = []
for i in dt :
    datetime.append(i.split(' ')[0])
df_Weather['DATETIME'] = datetime
df_Weather.head()
```

	STATION	DATETIME	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY
0	133	2016-04-01	10.8	NaN	0.0	38
1	133	2016-04-01	9.5	NaN	0.1	49
2	133	2016-04-01	8.6	NaN	0.3	49
3	133	2016-04-01	7.8	NaN	1.1	52
4	133	2016-04-01	6.5	NaN	0.1	60

날씨 정보를 가져옵니다. 이 때 Weather의 날씨 정보 데이터가 이상하기 때문에, 위처럼 for 문을 활용하여 split하여 새로운 DATETIME 정보를 생성해 넣어줍니다.

```
df_MaxTem = df_Weather.groupby(['DATETIME'])['TEMPERATURE'].max().reset_index()
df_MeanHumidity = df_Weather.groupby(['DATETIME'])['HUMIDITY'].mean().reset_index()
df_MeanWindSpeed = df_Weather.groupby(['DATETIME'])['WINDSPEED'].mean().reset_index()
df_Weather =df_Weather.fillna(0)
df_sumrainFall =  df_Weather.groupby(['DATETIME'])['RAINFALL'].sum().reset_index()

df_MaxTem['RAINFALL'] = df_sumrainFall['RAINFALL']
df_MaxTem['WINDSPEED'] = df_MeanWindSpeed['WINDSPEED']
df_MaxTem['HUMIDITY'] = df_MeanHumidity['HUMIDITY']
df_Weather = df_MaxTem
df_Weather = df_Weather.rename(columns = {'DATETIME' : 'DATE'})
df_Weather.head()
```

	DATE	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY
0	2016-04-01	24.9	0.0	0.979167	37.208333
1	2016-04-02	24.1	0.0	1.145833	40.958333
2	2016-04-03	16.3	2.5	1.379167	64.916667
3	2016-04-04	18.0	0.0	1.887500	72.375000
4	2016-04-05	20.5	0.0	1.712500	40.125000

과제에서 요구하던 날씨 정보를 groupby를 활용하여 생성하여줍니다. DATETIME별로 최고 온도, 평균 Humidity, 평균 풍속, 하루 강수량을 생성하여주고, df_Weather 데이터에 넣어줍니다.
merge를 수행하여주기 위해 columns의 이름을 변경하여줍니다.

```
df = pd.merge(df, df_Weather, on = ['DATE'] ,how ='left')
df.head()
```

	DATE	STEPS	MINUTESASLEEP	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY
0	2016-04-01	11632.500000	343.771930	24.9	0.0	0.979167	37.208333
1	2016-04-02	10662.938462	336.525424	24.1	0.0	1.145833	40.958333
2	2016-04-03	7870.104478	422.937500	16.3	2.5	1.379167	64.916667
3	2016-04-04	16206.477612	359.189655	18.0	0.0	1.887500	72.375000
4	2016-04-05	13893.089552	335.931034	20.5	0.0	1.712500	40.125000

merge로 생성한 데이터들을 merge하여 목표하는 작업을 수행할 데이터를 생성하여줍니다.

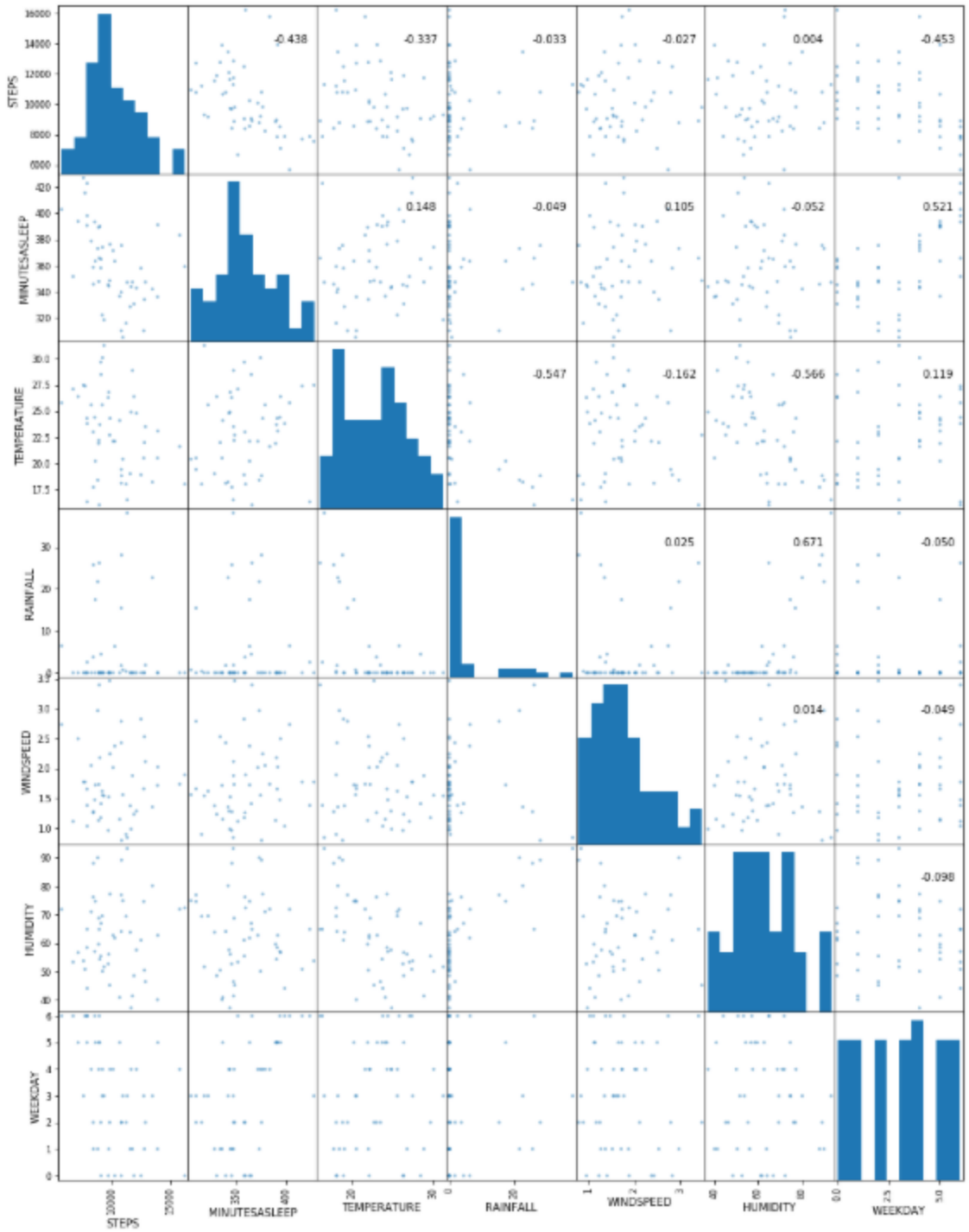
```
# 요일 정보 삽입
import datetime
weekday = []
for i in df['DATE'] :
    weekday.append(datetime.datetime(int(i.split('-')[0]),int(i.split('-')[1]),int(i.split('-')[2])).weekday())
df['WEEKDAY'] = weekday
del df['DATE']
df.head()
```

	STEPS	MINUTESASLEEP	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY	WEEKDAY
0	11632.500000	343.771930	24.9	0.0	0.979167	37.208333	4
1	10662.938462	336.525424	24.1	0.0	1.145833	40.958333	5
2	7870.104478	422.937500	16.3	2.5	1.379167	64.916667	6
3	16206.477612	359.189655	18.0	0.0	1.887500	72.375000	0
4	13893.089552	335.931034	20.5	0.0	1.712500	40.125000	1

weekday정보를 삽입하기 위하여 datetime을 import하여 수행하여줍니다.

```
axes = pd.plotting.scatter_matrix(df, figsize = (15,20), alpha = 0.5)

corr = df.corr().as_matrix()
for i,j in zip(*plt.np.triu_indices_from(axes, k=1)):
    axes[i,j].annotate("%.3f" %corr[i,j],(0.8, 0.8), xycoords = 'axes fraction', ha = 'center', va ='center')
plt.show()
```

위처럼 작업을 수행하여, 걸음, 수면, 날씨와의 상관관계 요인을 그래프로 나타내고 상관계수를 구할 수 있습니다.



구하여진 상관계수입니다.

보시는 것 과 같이 걸음과 온도는 -0.337의 상관관계를 가지고 있습니다.

걸음과 요일은 -0.453의 상관관계를 가지고 있고, 걸음과 강수량은 -0.033의 관계를 가지고 있습니다.

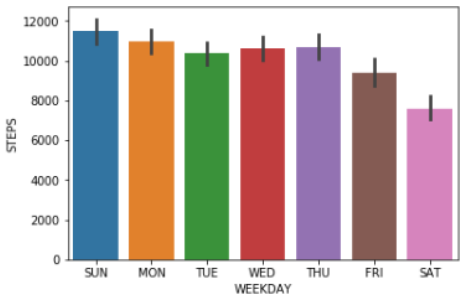
수면과 요일은 0.521의 상관관계를 가지고 있고, 수면과 온도는 0.148의 상관관계를 가지고 있습니다. 그리고 수면과 강수량은 -0.049의 상관관계를 가지고 있음을 확인할 수 있습니다.

[1-b-i] 요일

```
import datetime
weekday = []
df_Step = pd.read_csv('./data/all_user_steps.csv')
df_Step = df_Step.groupby(['USERNAME','DATE'])['STEPS'].sum().reset_index()

for i in df_Step['DATE'] :
    weekday.append(datetime.datetime(int(i.split('-')[0]),int(i.split('-')[1]),int(i.split('-')[2])).weekday())
df_Step['WEEKDAY'] = weekday
df_Step.head()
```

걸음수 데이터를 가져와서 각 학생의 날짜별 걸음수에 대한 데이터를 생성하여줍니다.
그리고 DATE 데이터를 활용하여 요일 정보를 생성하여줍니다.

```
sns.barplot(x="WEEKDAY", y="STEPS", data=df_Step)

plt.xticks(range(0,7),['SUN','MON','TUE','WED','THU','FRI','SAT'])
plt.show()
```



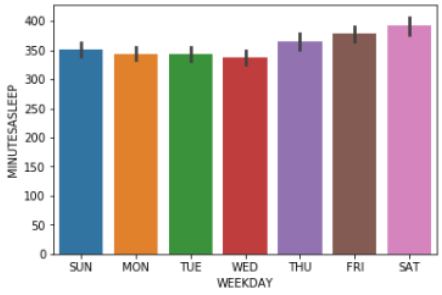
각 사람들의 걸음수 데이터를 가지고, 생성한 그래프입니다.
X축은 요일이고 Y축은 걸음수로, 학생들의 하루 걸음수 데이터 전부를 활용하여 그래프를 그려낸 결과입니다.
xticks를 활용하여 X축의 데이터 값을 요일로 변경하여주었습니다.
일요일에 평균적으로 걸음수가 가장 많고, 토요일에 걸음수가 가장 적음을 확인할 수 있습니다.

```
import datetime
weekday = []

df_Sleep = pd.read_csv('./data/all_user_sleeps.csv')
df_Sleep = df_Sleep.groupby(['USERNAME','DATE'])['MINUTESASLEEP'].mean().reset_index()
for i in df_Sleep['DATE'] :
    weekday.append(datetime.datetime(int(i.split('-')[0]),int(i.split('-')[1]),int(i.split('-')[2])).weekday())
df_Sleep['WEEKDAY'] = weekday
df_Sleep.head()
```

	USERNAME	DATE	MINUTESASLEEP	WEEKDAY
0	A01	2016-04-01	485	4
1	A01	2016-04-02	512	5
2	A01	2016-04-04	335	0
3	A01	2016-04-05	311	1
4	A01	2016-04-06	491	2

수면 데이터를 가져와서 사람들의 일별 수면 데이터를 생성하여줍니다.
그리고, 위 Step 데이터와 동일하게 요일 데이터를 생성하여 줍니다.

```
sns.barplot(x="WEEKDAY", y="MINUTESASLEEP", data=df_Sleep)
plt.xticks(range(0,7),['SUN','MON','TUE','WED','THU','FRI','SAT'])
plt.show()
```



위 데이터를 가지고 생성한 그래프입니다.
X축은 요일이고, Y축은 수면에 대한 양임을 확인할 수 있습니다.
요일별로 금요일과 토요일에 잠을 좀 더 많이 자는 것을 알 수 있습니다.

[1-b-ii] 최고기온

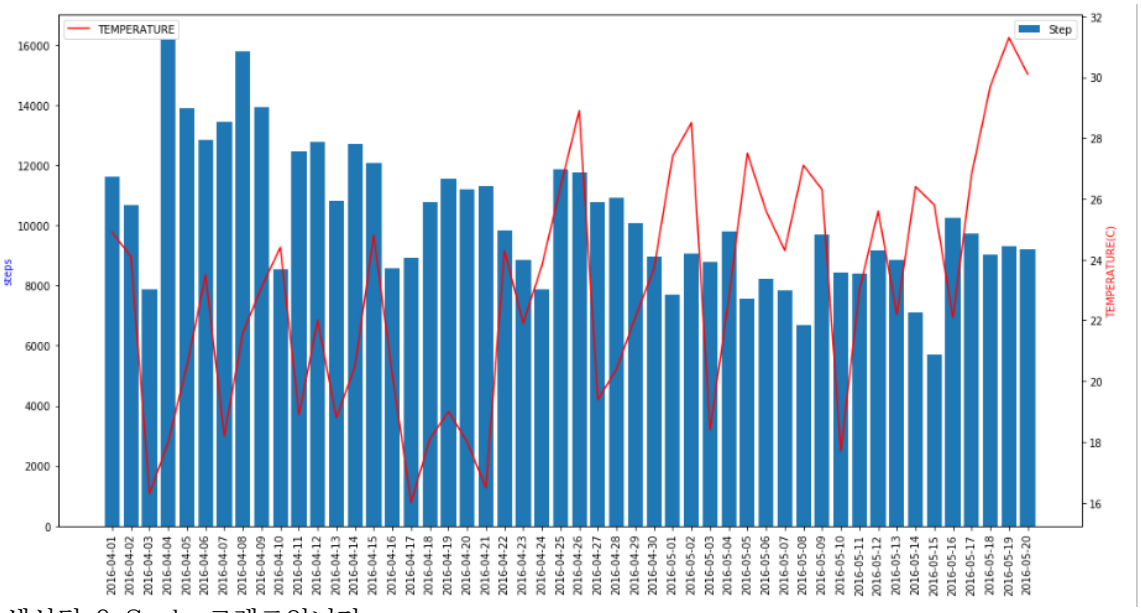
```
df_Step = pd.read_csv('./data/all_user_steps.csv')
df_Step = df_Step.groupby(['USERNAME','DATE'])['STEPS'].sum().reset_index()
df_Step = df_Step.groupby(['DATE'])['STEPS'].mean().reset_index()

UseData = pd.merge(df_Step, df_Weather, on = ['DATE'] ,how ='left')
UseData.head()
```

	DATE	STEPS	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY
0	2016-04-01	11632.500000	24.9	0.0	0.979167	37.208333
1	2016-04-02	10662.938462	24.1	0.0	1.145833	40.958333
2	2016-04-03	7870.104478	16.3	2.5	1.379167	64.916667
3	2016-04-04	16206.477612	18.0	0.0	1.887500	72.375000
4	2016-04-05	13893.089552	20.5	0.0	1.712500	40.125000

사용할 데이터를 만들어줍니다.
DATE별 STEP 데이터의 평균을 만들어주고, 위에서 생성한 날씨 데이터와 merge를 활용하여 합쳐줍니다.

```
fig, ax1 = plt.subplots(figsize=(15, 8))
t = UseData['DATE']
s2 = UseData['STEPS']

ax1.bar(t, s2)
ax1.set_ylabel('steps',color = 'b')
plt.xticks(rotation=90)
plt.legend(['Step'])

ax2 = ax1.twinx()
s1 = UseData['TEMPERATURE']
ax2.plot(t, s1, color = 'r')
ax2.set_xlabel('DATE')
ax2.set_ylabel('TEMPERATURE(C)',color = 'r')
fig.tight_layout()
plt.legend(['TEMPERATURE'])

plt.show()
```

그래프를 그려주기 위한 코드입니다.
원하는 그래프를 그리기 위해서 그래프를 그리는 figsize를 가로를 좀 더 크게 만들었습니다.
우선 t데이터는 X축의 데이터로 'DATE'의 데이터를 넣어줍니다. Y축의 데이터는 'STEP'데이터로 걸음수를 넣어줍니다. 걸음수 데이터는 bar로 생성하여 주었습니다.
두 번째로 온도 데이터를 생성하여주는데, 온도 데이터는 plot으로 생성하여 꺾은선 그래프와 같은 모양으로 표현하여줍니다.

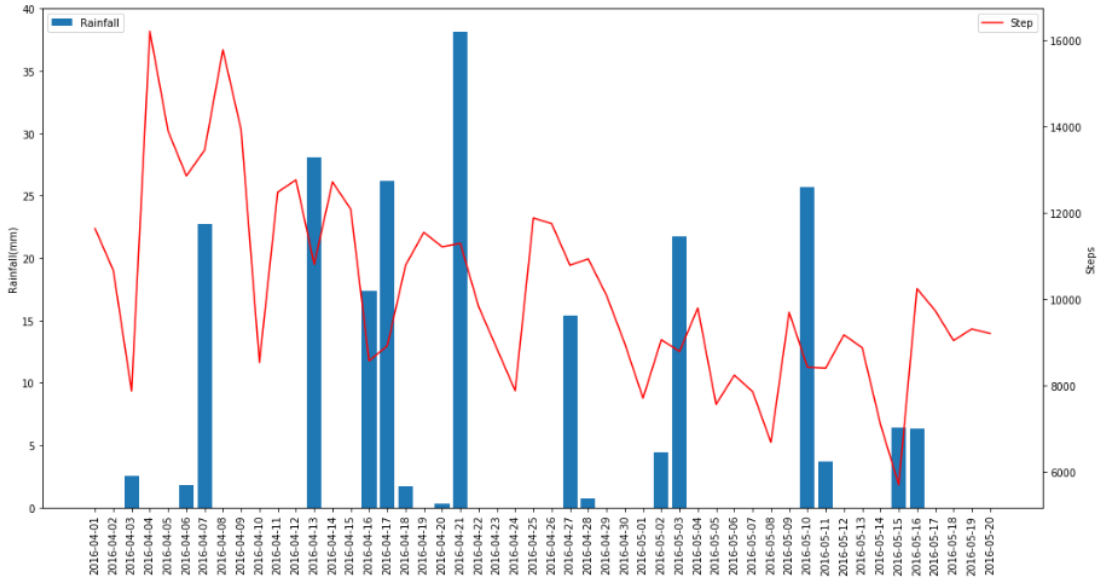


생성된 2-Scale 그래프입니다.
X축은 날짜임을 확인할 수 있습니다.
빨간색 꺾은선 그래프와 같은 plot은 온도를 의미합니다.
파란색 bar 그래프는 Step 걸음수를 의미합니다.
Y축의 좌측에는 Step 데이터의 값들이 들어가 있습니다.
Y축의 우측에는 온도 데이터의 값들이 들어가 있습니다.
두 개의 데이터(온도, 걸음수)를 한번에 날짜별로 확인할 수 있습니다.

[1-b-iii] 강수량

```
fig, ax1 = plt.subplots(figsize=(15, 8))
t = UseData['DATE']

s2 = UseData['RAINFALL']

ax1.bar(t, s2)
ax1.set_ylabel('Rainfall(mm)')
plt.xticks(rotation=90)
ax1.legend(['Rainfall'],loc='upper left')

ax2 = ax1.twinx()
s1 = UseData['STEPS']
ax2.plot(t, s1, color = 'r')
ax2.set_xlabel('DATE')
ax2.set_ylabel('Steps')
fig.tight_layout()
ax2.legend(['Step'])

plt.show()
```

데이터는 위에 1-b-ii에서 생성한 데이터를 활용하여 그래프를 그려주었습니다.
우선 동일한 크기의 그래프를 그려줍니다. 이 때 s2데이터를 RAINFALL로 그려줍니다.
강수량 데이터는 bar로 그려주고, 걸음 수 데이터는 꺾은선 그래프로 그려주었습니다.



최종적으로 생성한 그래프입니다.

보는 것과 같이, X축은 날짜에 대한 정보이고, 좌측 Y축은 강수량에 대한 값을 우측 Y축은 Step 걸음 수 데이터를 축으로 생성하였습니다.

빨간색 꺽은선 그래프는 Step에 대한 데이터입니다.

파란색 막대 그래프는 강수량에 대한 데이터입니다.

두 개의 데이터(걸음수, 강수량)을 한번에 확인할 수 있습니다.

< HW 1-c >

[1-c] Fitbit 하루 이용자 변화 분석

하루 이용자 변화를 분석하기 위해서는 심박수 데이터를 활용합니다. 심박수 데이터가 0인 경우에는 착용을 하지 않았다는 것을 의미하므로, 하루 심박수 평균이 0인 경우를 NoCount하여주어 하루 이용자의 변화를 분석할 수 있는 그래프를 만들어줍니다.

```
df_H = pd.read_csv('./data/all_user_hearts.csv')
df_H = df_H.fillna(0)
df_H.head(3)
```

	USERNAME	DATE	TIME	VALUE
0	A01	2016-04-01	00:00:00	79
1	A01	2016-04-01	00:01:00	80
2	A01	2016-04-01	00:02:00	78

우선 심박수 데이터를 활용해야하기 때문에 심박수 데이터를 가져옵니다.

```
HeartData = df_H.groupby(['USERNAME','DATE'])['VALUE'].mean().reset_index()
HeartData.head()
```

	USERNAME	DATE	VALUE
0	A01	2016-04-01	84.276132
1	A01	2016-04-02	83.300429
2	A01	2016-04-03	87.374734
3	A01	2016-04-04	83.575121
4	A01	2016-04-05	83.615627

groupby를 활용하여 모든 사용자마다 모든 날의 평균 심장 박동수 데이터를 만들어줍니다.

```
FL = [x for x in os.listdir('sokulee/') if x.find("A") != -1 ]
name = []
date = []
makeName =[]
makeDate =[]
for Name in FL:
    name.append(Name)

for i in range(20160401,20160431):
    date.append(i)
for i in range(20160501,20160521):
    date.append(i)

for A in name:
    for B in date :
        makeName.append(A)
        makeDate.append(B)
newData = pd.DataFrame()
newData['USERNAME'] = makeName
newData['DATE'] = makeDate
newData.head()
```

	USERNAME	DATE
0	A01	20160401
1	A01	20160402
2	A01	20160403
3	A01	20160404
4	A01	20160405

위 과정은 언제 학생이 사용을 하지 않았는지를 체크하기 위하여 데이터를 생성하는 과정이다. 우선, FL에 os를 활용하여 List를 만들어서 모든 name에 대한 정보를 넣어주고,
date에는 우리가 사용하는 데이터의 날짜 정보(2016.04.01.~2016.05.20.)을 넣어준다.
그리고 그 date와 name을 이중 for문을 활용하여, 각 학생별 날짜로 데이터를 만들어준다.
그렇게 되면 A01부터 모든 사용자의 2016.4.1.~2016.5.20의 데이터가 생성됩니다. 그리고 그 데이터를 dataframe으로 생성하여줍니다.

```
Date = []
for i in HeartData['DATE']:
    Date.append(i.split('-')[0] + i.split('-')[1]+ i.split('-')[2])
HeartData['DATE'] = Date
HeartData.head()
```

	USERNAME	DATE	VALUE
0	A01	20160401	84.276132
1	A01	20160402	83.300429
2	A01	20160403	87.374734
3	A01	20160404	83.575121
4	A01	20160405	83.615627

심박수데이터의 DATE가 yyyy-mm-dd의 형태를 가지고 있으므로 split을 활용하여 붙여준다.
merge를 수행하기 위하여 동일한 형태를 만들어주기 위해서이다.

```
HeartData['DATE'] = HeartData['DATE'].astype(int)
newData['DATE'] = newData['DATE'].astype(int)
UseData = pd.merge(newData, HeartData, on = ['DATE','USERNAME'] ,how ='left')
UseData = UseData.fillna(0)

UseData.head()
```

	USERNAME	DATE	VALUE
0	A01	20160401	84.276132
1	A01	20160402	83.300429
2	A01	20160403	87.374734
3	A01	20160404	83.575121
4	A01	20160405	83.615627

HeartData와 newData를 활용하여 merge를 수행하여줍니다. 이때 newData를 기반으로 join을 하여주어, VALUE값이 없는 데이터의 경우 NA가 들어가게 됩니다. 즉, NA의 데이터가 있는 위치는 사용을하지 않은 데이터가 되므로 심박수를 0이라 생각하고 fillna로 0을 넣어줍니다.

```
Count = []
FindData = UseData[UseData['VALUE'] == 0]
index = FindData.index
d = 0
for i in UseData['DATE']:
    if d in index:
        Count.append(0)
    else:
        Count.append(1)
    d = d + 1
UseData['Count'] = Count
UseData.head(5)
```

	USERNAME	DATE	VALUE	Count
0	A01	20160401	84.276132	1
1	A01	20160402	83.300429	1
2	A01	20160403	87.374734	1
3	A01	20160404	83.575121	1
4	A01	20160405	83.615627	1

Count정보를 생성하여줍니다. 우선 FindData를 만들어 위에서 생성한 데이터에 0인 부분을 찾아줍니다. 그리고, 그 찾은 데이터의 인덱스 위치를 index에 넣어줍니다.
for문을 돌려가며 심박수가 0인 위치의 index에 접근할 때는 0을 넣어주고 그 외에 경우에는 1을 넣어 Count 리스트를 생성하고, 그 데이터를 UseData에 넣어줍니다.

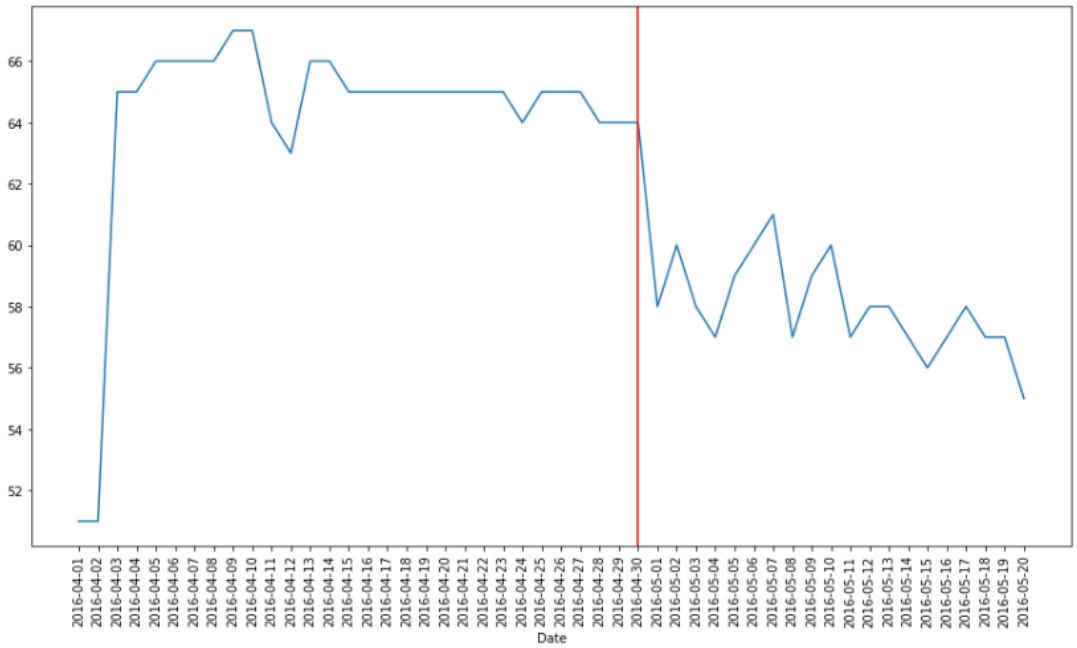
```
UseData = UseData.groupby(['DATE'])['Count'].sum().reset_index()

date = []
for da in UseData['DATE']:
    date.append(str(da)[0:4] +"-" + str(da)[4:6]+"-" + str(da)[6:8])
UseData['DATE'] = date
UseData.head()
```

	DATE	Count
0	2016-04-01	51
1	2016-04-02	51
2	2016-04-03	65
3	2016-04-04	65
4	2016-04-05	66

최종적으로 생성한 데이터를 groupby로 날짜별 카운트를 하여줍니다.
이 때, DATE를 다시 yyyy-mm-dd 형태로 변경하여주는 작업을 수행합니다.

```
plt.subplots(figsize=(15, 8))
plt.plot(UseData['DATE'], UseData['Count'])
plt.xticks(rotation=90)
plt.xlabel('Date')
plt.axvline(color = 'r' ,x = '2016-04-30')
plt.show()
```

그래프를 그려주는데, x축에는 DATE정보로 y축에는 Count로 그려줍니다.
rotation을 해주어 x축의 데이터가 잘 보일 수 있게 만들어줍니다.
x = 2016-04-30지점에 axvline을 활용하여 빨간색으로 선을 그어줍니다.



생성된 그래프입니다. 위에서 설명한 것처럼 X축에는 Date정보를 Y축에는 사용한 사람들에 대하여 Count한 값을 나타내어줍니다.
이 때, 4월 30일을 기점으로 가장 큰 감소가 이루어지는 것을 확인할 수 있습니다. 따라서 4월 30일에 빨간색 선으로 급격히 감소하는 부분을 표시하여주었습니다.