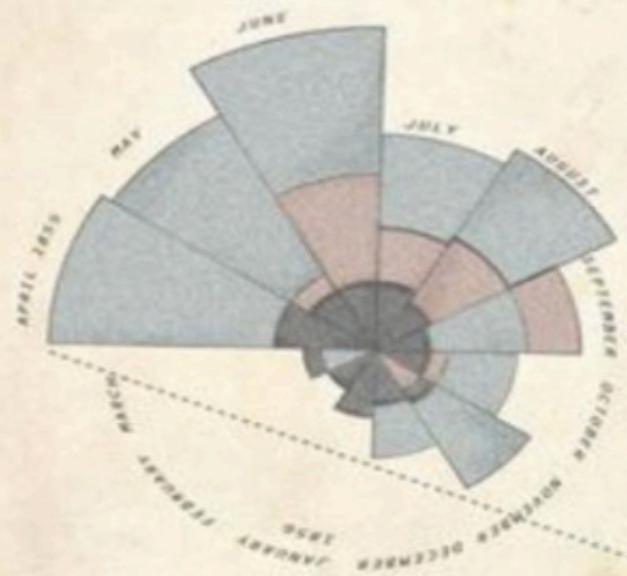


DIAGRAM of the CAUSES of MORTALITY IN THE ARMY IN THE EAST

APRIL 1855 to MARCH 1856



APRIL 1854 to MARCH 1855



THE AREAS OF THE BLUE, RED, & BLACK WEDGES ARE EACH MEASURED FROM THE CENTRE AS THE COMMON VERTEX.
THE BLUE WEDGES MEASURED FROM THE CENTRE THE DEATHS FROM PREVENTABLE OR MITIGABLE ZYNOTIC DISEASES.
THE RED WEDGES MEASURED FROM THE CENTRE THE DEATHS FROM WOUNDS, & THE
BLACK WEDGES MEASURED FROM THE CENTRE THE DEATHS FROM ALL OTHER CAUSES.
THE BLACK LINE ACROSS THE RED TRIANGLE IN NOV. 1854 MARKS THE BOUNDARY
OF THE DEATHS FROM ALL OTHER CAUSES DURING THE MONTH.
IN OCTOBER 1854, & APRIL 1855, THE BLACK AREA COINCIDES WITH THE RED.
IN JANUARY & FEBRUARY 1855, THE BLUE COINCIDES WITH THE BLACK.
THE ENTIRE AREAS MAY BE COMPARED BY FOLLOWING THE BLUE, THE RED & THE
BLACK LINE ENCLITICED.

나이팅게일의 로즈 다이어그램

데이터 과학의 목적 중 하나

- 가정(혹은 ‘인식’)을 검증하는 것

플로렌스 나이팅게일

위키백과, 우리 모두의 백과사전.

☞ 이 문서는 간호사에 관한 것입니다. 새에 대해서는 나이팅게일 문서를 참조하십시오.

플로伦스 나이팅게일(Florence Nightingale, OM, 1820년 5월 12일~1910년 8월 13일)은 영국의 간호사, 작가, 통계학자이다.

성공회의 성인이기도 하며, 성공회에서는 8월 13일을 나이팅게일의 축일로 지키고 있다.

목차 [숨기기]

1 생애

- 1.1 간호수업
- 1.2 간호활동
 - 1.2.1 크림전쟁에서의 활약
 - 1.2.2 이론가
 - 1.2.3 집중치료실
- 1.3 저술활동
- 1.4 통계학자
- 1.5 사망
- 1.6 나이팅게일 다시 읽기
 - 1.6.1 여성의 노동인권으로 읽기
 - 1.6.2 메리 시클 이야기

2 각주

3 같이 보기

4 바깥고리

플로伦스 나이팅게일
Florence Nightingale



출생 1820년 5월 12일

 이탈리아 피렌체

사망 1910년 8월 13일 (90세)

 잉글랜드 런던

사인 병사

생애 [편집]

간호수업 [편집]

1849년 이집트 여행 도중에 알렉산드리아 병원을 참관하고, 정규 간호 교육의 중요성을 절실히 느끼게 되었다. [인문주의자](#)들인 부모의 영향으로 어려서부터 가난한 이웃들에게 관심이 많았던 나이팅게일은 전쟁의 참상에 대한 기사를 타임스 신문에서 읽은 뒤 자극받아, [간호사](#)에 대한 편견 탓에 집안의 명예실추를 우려한 부모의 반대를 무릅쓰고, [개신교 목사가 운영하는 독일 카이저벨트의 프로테스탄트 학교](#)에서 간호학을 공부하고 1853년 런던 숙녀병원의 간호부장이 되었다.

간호활동 [편집]

크림전쟁에서의 활약 [편집]

이듬해 [크림 전쟁\(1854년~1856년\)](#) 당시 38명의 [성공회 수녀](#)들의 도움을 받으며 [스쿠타리](#)의 야전 병원에서 초인간적인 활약을 보였다. 흔히 나이팅게일 하면 고통 받는 부상병들을 돌본 봉사자를 연상하지만 이 시기의 나이팅게일은 유능한 행정가요 협상가였다. 그는 [관료주의](#)에 물든 군의 관리들을 설득했고, [병원](#)에서 쓰는 물건들을 세심하게 조사했으며^[1], 무질서한 [병원](#)에 규율을 세웠다. 환자의 사망률은 42퍼센트에서 2퍼센트로 뚝 떨어졌다는 사실은 나이팅게일이 뛰어난 행정가임을 말해준다.

이론가 [편집]

1860년 나이팅게일 간호 학교(현재 [킹스 칼리지 런던](#)의 일부)를 설립하고 간호전문서적을 쓰으로써 [조선](#)의 [의녀](#)처럼 천대받던 [직업](#)인 간호사를 전문직업으로 성숙시키는 업적을 남겼다. 이 밖에도 많은 병원 및 간호 시설의 창립, 개선에 힘쓰고 남북 전쟁과 [프로이센-프랑스 전쟁](#) 때는 외국 정부의 고문으로 활약하였다. 1907년 [영국 왕 에드워드 7세](#)로부터 여성 최초로 메리트 훈장(Order of Merit)을 받았으며, 만국 적십자사에서는 '나이팅게일 상'을 설정하여 매년 세계 각국의 우수한 간호사를 표창하고 있다.

국적  영국

별칭 램프를 든 천사(The Lady with the Lamp)

학력 독일에서 [간호사](#) 수업

직업 [간호사](#), 통계학자

종교 성공회

부모 부친 윌리엄 에드워드 나이팅게일
모친 프랜시스 나이팅게일

친척 언니 파시노프 나이팅게일

서명 

나라를 운용할 사람들은 통계활용법을 배워야 한다.

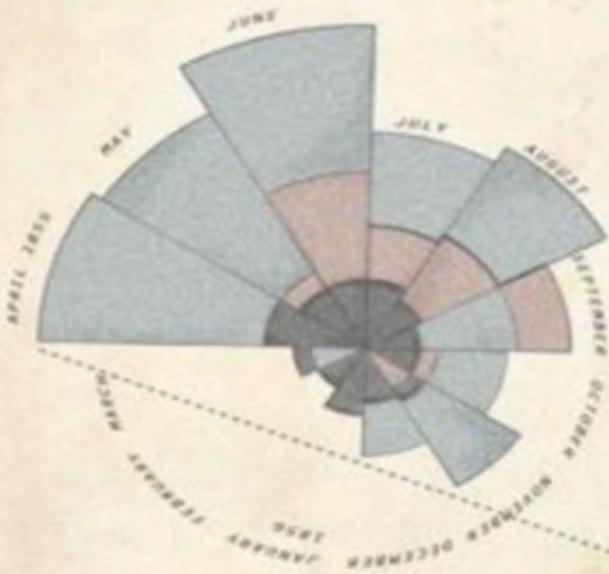
- 나이팅게일 -

- 영국왕립통계학회 회원 -



DIAGRAM of the CAUSES of MORTALITY IN THE ARMY IN THE EAST

APRIL 1855 to MARCH 1856



THE AREAS OF THE BLUE, RED, & BLACK HEDGES ARE EACH MEASURED FROM
THE CENTRE AS THE COMMON VERTEX.

THE BLUE HEDGES MEASURED FROM THE CENTRE OF THE CIRCLE REPRESENT AREA FOR AREA THE DEATHS FROM PREVENTABLE OR MITIGABLE ZYNOTIC DISEASES.

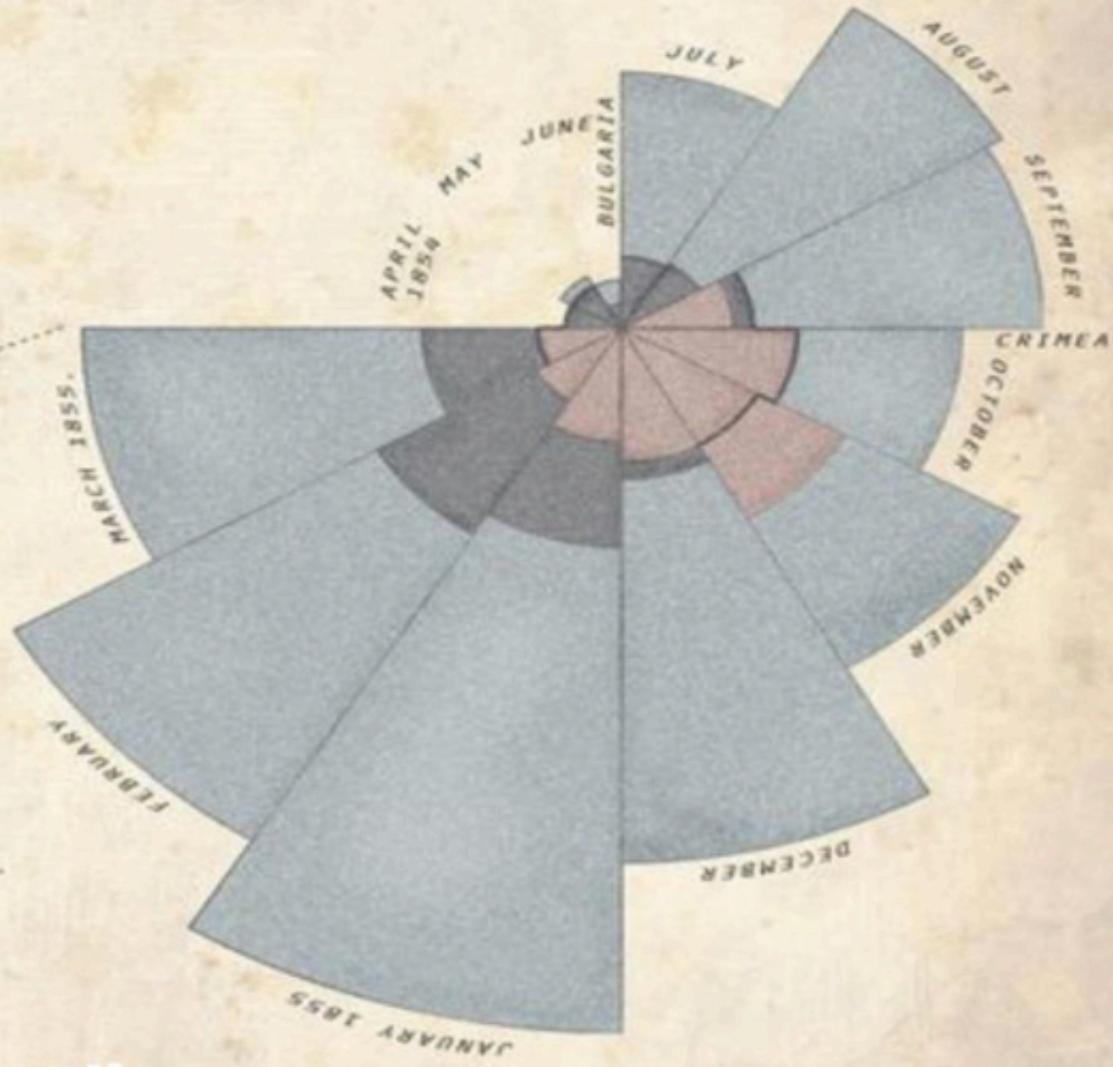
THE RED HEDGES MEASURED FROM THE CENTRE THE DEATHS FROM WOUNDS. & THE
BLACK HEDGES MEASURED FROM THE CENTRE THE DEATHS FROM ALL OTHER CAUSES.

THE BLACK LINE ACROSS THE RED TRIANGLE IN NOV. 1854 MARKS THE BOUNDARY OF THE DEATHS FROM ALL OTHER CAUSES DURING THE MONTH.

IN OCTOBER 1854, & APRIL 1855, THE BLACK AREA COINCIDES WITH THE RED,
IN JANUARY & FEBRUARY 1856, THE BLUE COINCIDES WITH THE BLACK.

THE ENTIRE AREA MAY BE COMPARED BY FOLLOWING THE BLUE-, THE RED & THE
BLACK LINES ON THIS MAP.

APRIL 1854 to MARCH 1855



나이팅게일의 로즈 다이어그램

나폴레옹의 March

Carte Figurative des étapes successives des hommes de l'Armée Française dans la Campagne de Russie 1812-1813.

Sur le plan - L. Minard, Inspecteur Général des Ponts et Chaussées en retraite.

Paris, le 20 Novembre 1869.

Les nombres d'hommes présents sont représentés par les larges des zones colorées à raison d'un millimètre pour dix mille hommes; ils sont de plus écrits en lettres des zones. Le rouge désigne les hommes qui ont été en Russie; le noir ceux qui en sortent. — Les renseignements qui ont servi à dresser la carte ont été puisés dans les ouvrages de M. M. Chiers, de Segur, de Fezensac, de Chambray et le journal médical de Jacob, pharmacien de l'Armée depuis le 28 Octobre.

Pour mieux faire juger à l'œil la diminution de l'armée, j'ai supposé que les corps du Prince Jérôme et du Maréchal Davout qui avaient été détachés sur Minsk et Mobilow et qui rejoignirent Ossatch et Wilekow, avaient toujours marché avec l'armée.

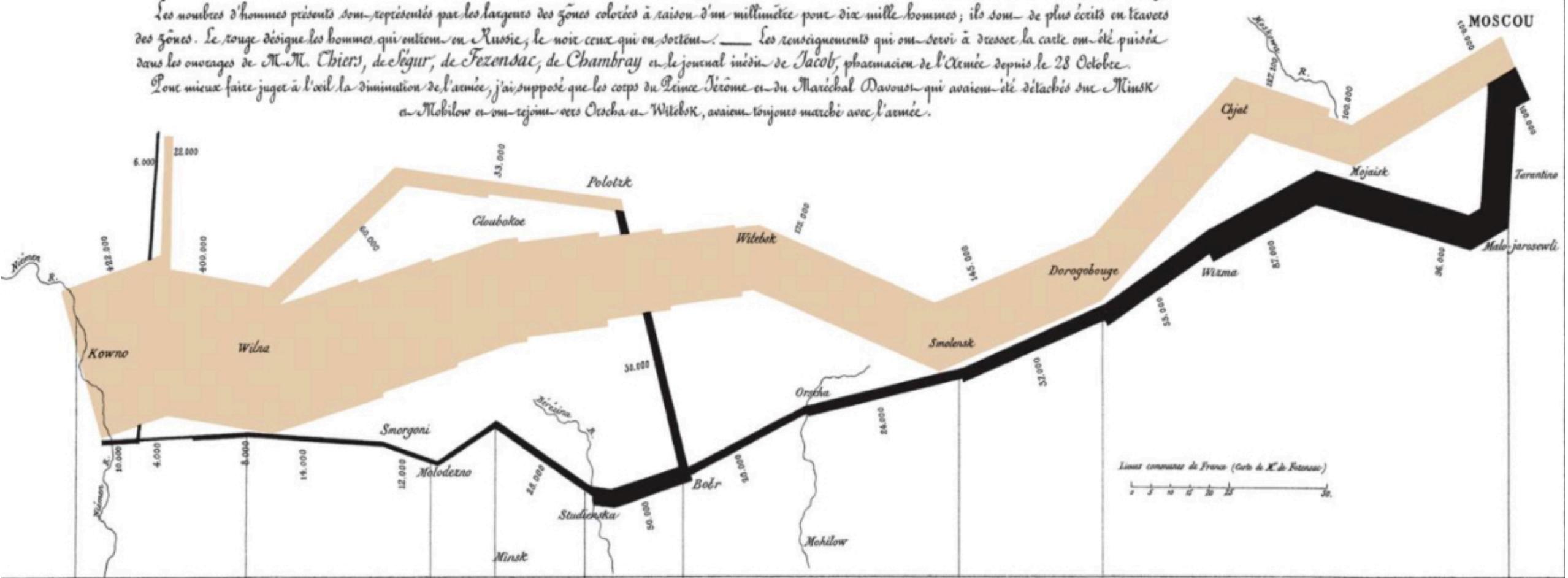
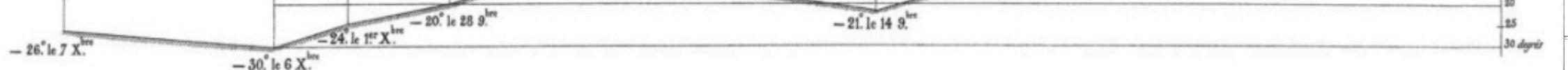


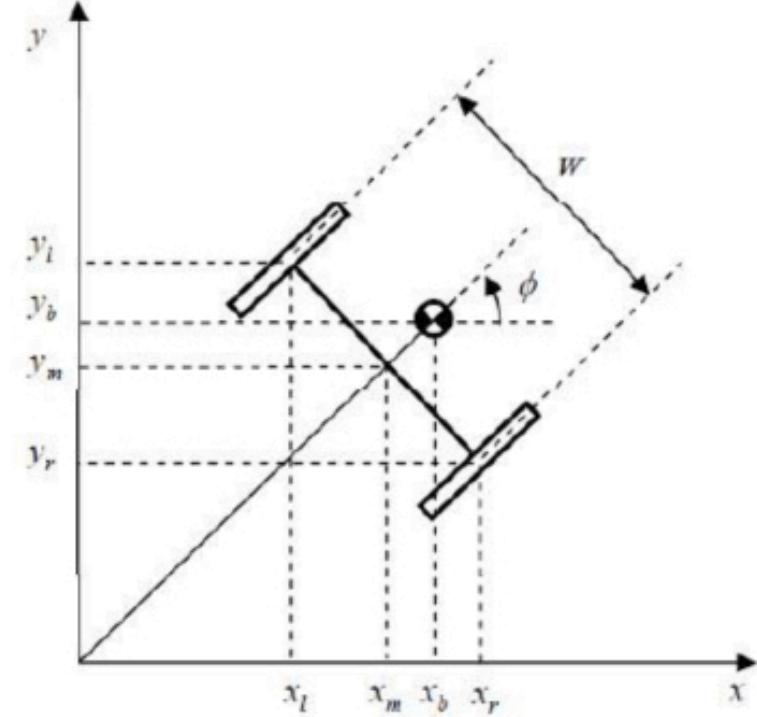
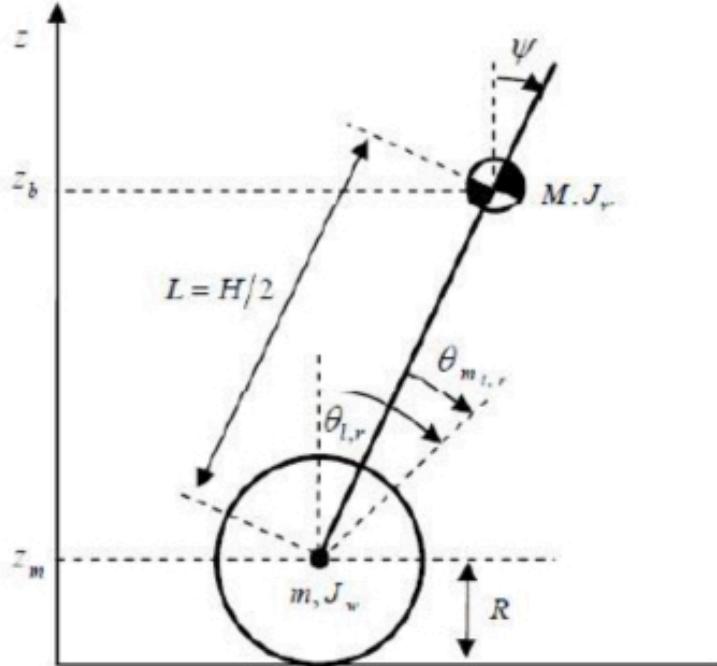
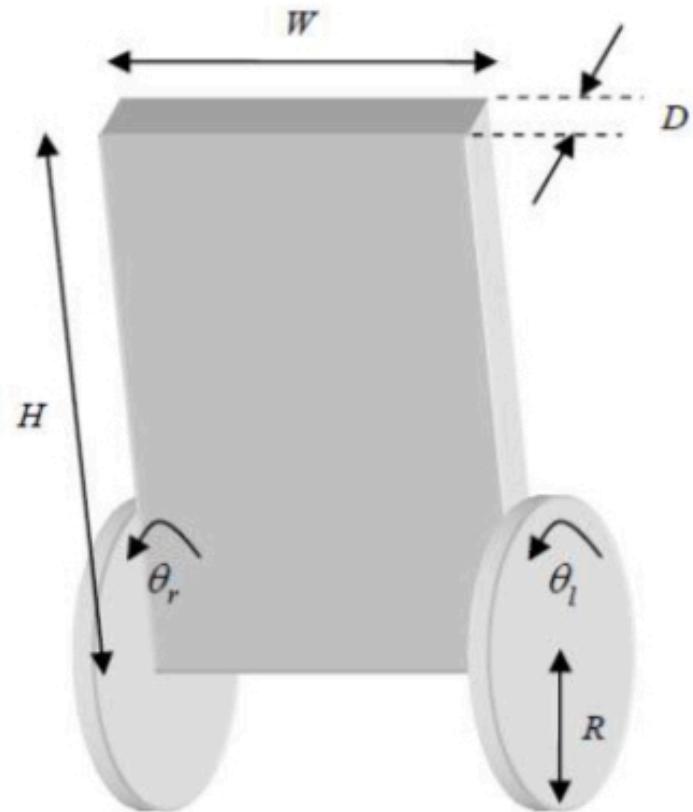
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.

Les cosaques passent au galop
le Niemen gelé.



Why? Python?

Actually My Favorite Language was MATLAB

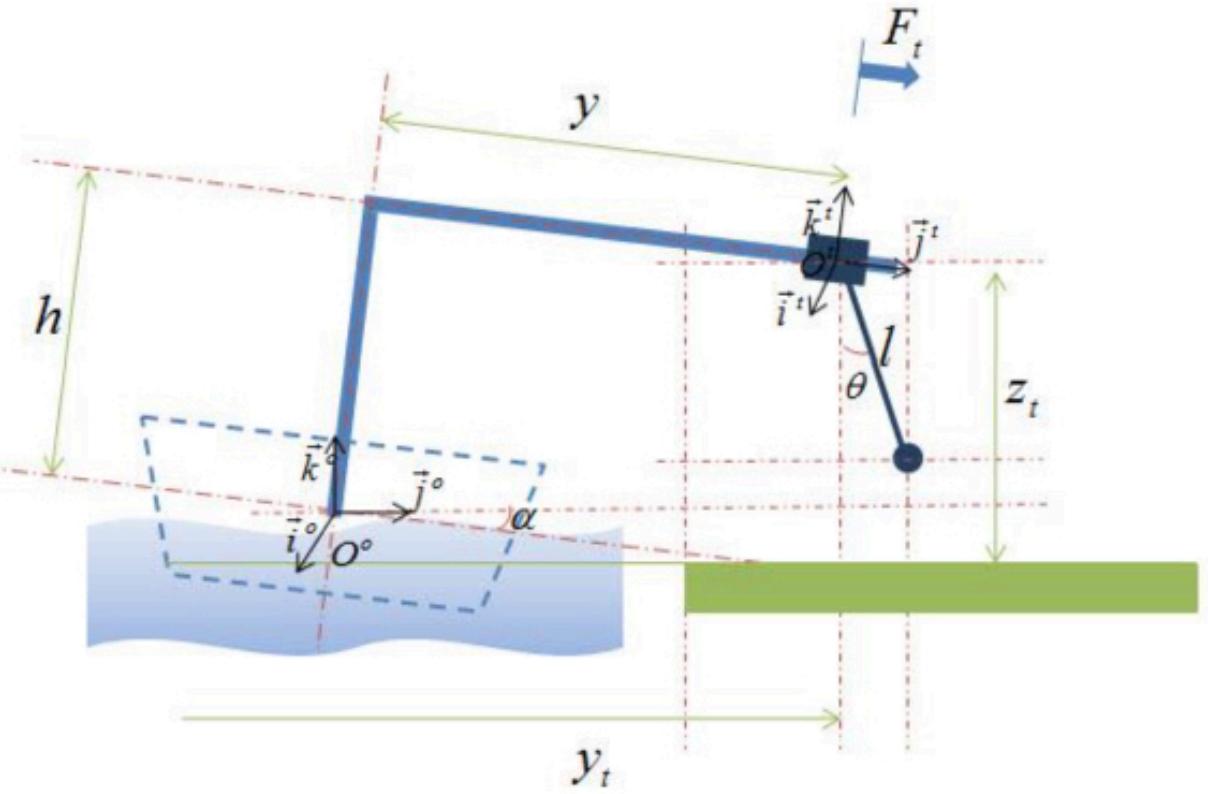


$$\ddot{\theta} = ((\cos \psi (LMR(\alpha(v_l + v_r) + 2\dot{\psi}\beta - 2\dot{\theta}\beta) - L^3 M^2 R \dot{\phi}^2 \sin \psi \cos \psi)) + \sin \psi (RL^3 M^2 \dot{\psi}^2 - Rg \cos \psi L^2 M^2 + 2J_m RLM \dot{\psi}^2 n^2 + J_\omega RLM \dot{\psi}^2 \\ + 2J_m g LM n^2) + J_\omega (\alpha(v_l + v_r) - 2\dot{\theta}(f_\omega + \beta) + 2\dot{\psi}\beta) + L^2 M(\alpha(v_l + v_r) - 2\dot{\theta}(f_u + \beta) + 2\dot{\psi}\beta) + 2J_m n^2 (\alpha(v_l + v_r) - 2\dot{\theta}(f_u + \beta) + 2\dot{\psi}\beta) \\ - 2J_m n^2 (\alpha(v_l + v_r) + 2\dot{\psi}\beta - 2\dot{\theta}\beta) + 2J_m L^2 M \dot{\phi}^2 n^2 \sin \psi \cos \psi) \\ / (2J_\omega J_\omega + L^2 M^2 R^2 + 2J_\omega L^2 M + J_\omega MR^2 + 2J_m J_\omega n^2 + 4J_m J_\omega n^2 + 2J_\omega R^2 m - L^2 M^2 R^2 \cos^2 \psi + 2J_m L^2 M n^2 + 2J_m MR^2 n^2 + 2L^2 M R^2 m \\ + 4J_m R^2 m n^2 + 4J_m LMR n^2 \cos \psi))$$

$$\ddot{\phi} = - \frac{1}{(2ML^2 R^2 \sin^2 \psi + mR^2 W^2 + 2J_\phi R^2 + J_m W^2 n^2 + J_\omega W^2)} (RW \alpha(v_l - v_r) + W^2 \dot{\phi}(f_\omega + \beta) + 2R^2 L^2 M \dot{\phi} \dot{\psi} \sin(2\psi))$$

$$\ddot{\psi} = -(4J_\omega (\alpha(v_l + v_r) + 2\dot{\psi}\beta - 2\dot{\theta}\beta) - \sin \psi (2LgM^2 R^2 + 4LgmMR^2 + 4J_m LMR \dot{\psi}^2 n^2 + 4J_m LgM n^2 + 4J_\omega LgM) + 4R^2 m(\alpha(v_l + v_r) + 2\dot{\psi}\beta - 2\dot{\theta}\beta) \\ - 4J_m n^2 (\alpha(v_l + v_r) - 2\dot{\theta}(f_\omega + \beta) + 2\dot{\psi}\beta) + 2MR^2 (\alpha(v_l + v_r) + 2\dot{\psi}\beta - 2\dot{\theta}\beta) + 4J_m n^2 (\alpha(v_l + v_r) + 2\dot{\psi}\beta - 2\dot{\theta}\beta) + 2LMR \cos \psi (\alpha(v_l + v_r) \\ - 2\dot{\theta}(f_\omega + \beta) + 2\dot{\psi}\beta) - L^2 M^2 R^2 \dot{\phi}^2 \sin(2\psi) + L^2 M^2 R^2 \dot{\psi}^2 \sin(2\psi) - 2J_\omega L^2 M \dot{\phi}^2 \sin(2\psi) - 2J_m L^2 M \dot{\phi}^2 n^2 \sin(2\psi) - 2L^2 M R^2 \dot{\phi}^2 m \sin(2\psi)) \\ / (4J_\omega J_\omega + 2L^2 M^2 R^2 + 4J_\omega L^2 M + 2J_\omega MR^2 + 4J_m J_\omega n^2 + 8J_m J_\omega n^2 + 4J_\omega R^2 m + 4J_m L^2 M n^2 + 4J_m MR^2 n^2 + 4L^2 M R^2 m + 8J_m R^2 m n^2 \\ - 2L^2 M^2 R^2 \cos^2 \psi + 8J_m LMR n^2 \cos \psi)$$

$$\alpha = \frac{nK_t}{R_m}, \quad \beta = \frac{nK_t K_b}{R_m} + f_m$$



$$\begin{aligned}\ddot{y}_t &= \frac{1}{M_t} (m_s l F_t - m_s l \cos(\theta - \alpha) F_b + m_s^2 l^2 \dot{\theta}^2 \sin(\theta - \alpha) \\ &\quad - m_s (m_t + m_s) l (\ddot{z}_t + g) \sin \alpha + m_s^2 l (\ddot{z}_t + g) \sin(\theta - \alpha)) \\ \ddot{\theta} &= \frac{1}{M_t} (-m_s \cos \theta F_t + (m_t + m_s) \cos \alpha F_b - m_s^2 l \dot{\theta} \sin(\theta - \alpha) \cos \theta \\ &\quad + m_s (m_t + m_s) (\ddot{z}_t + g) \sin(\theta - \alpha))\end{aligned}$$

그런데... 문제가

고가의 **MATLAB**을 구입해 줄 수 없는 회사...

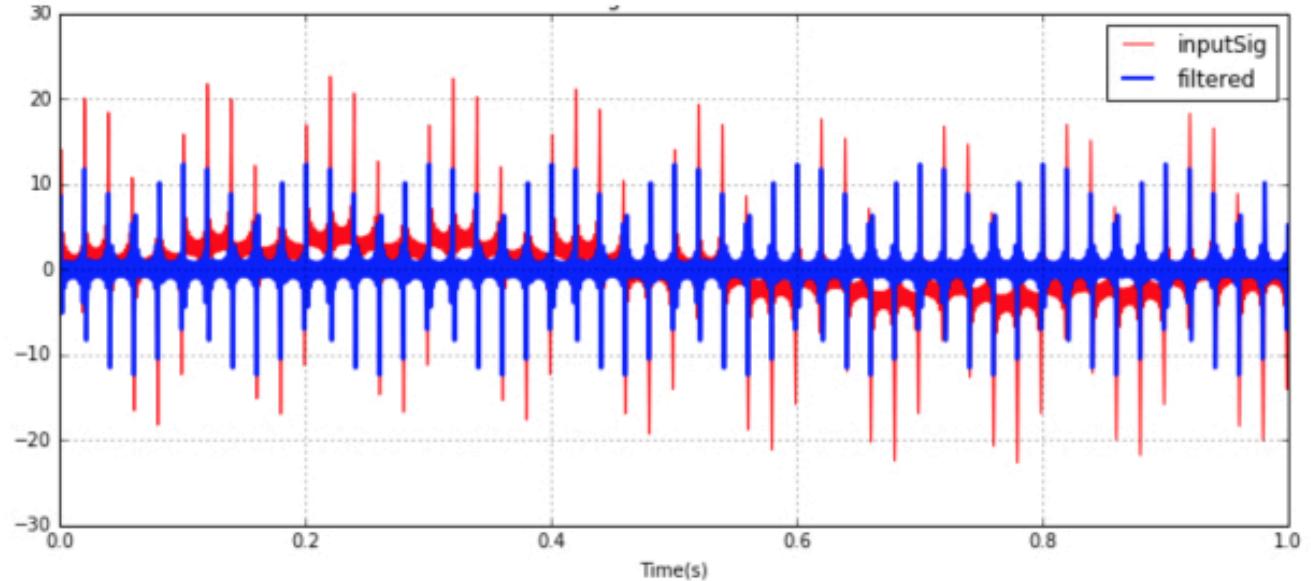
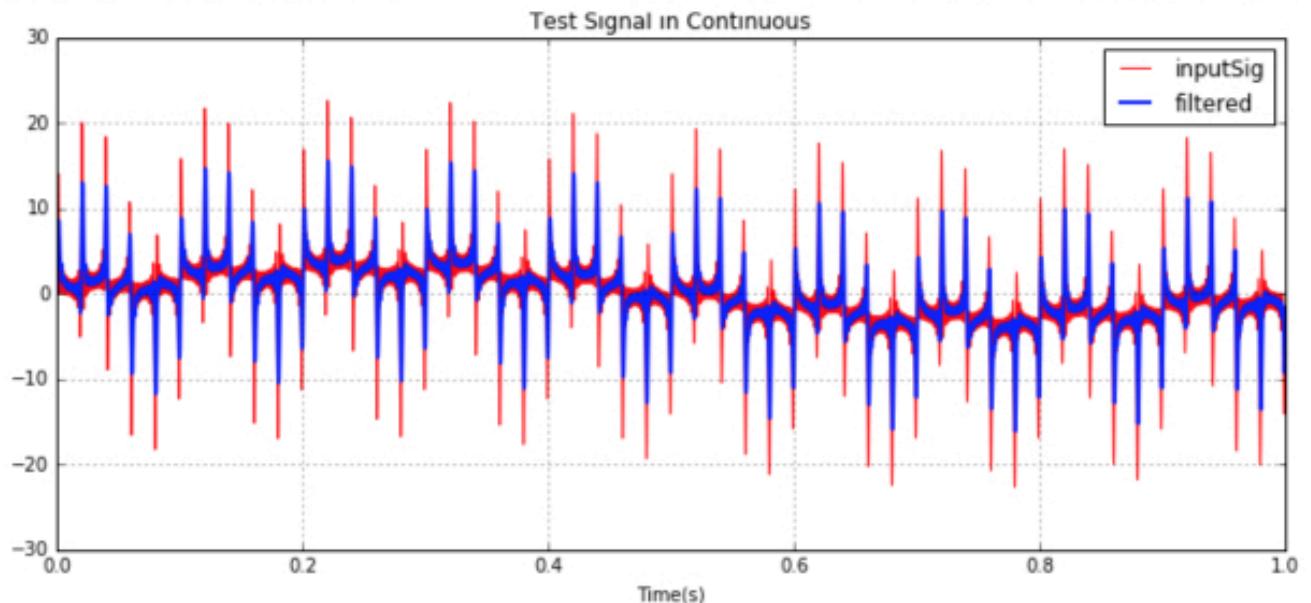
그래서...

엄청나게 대체 가능한 소프트웨어 도구를 찾던 중

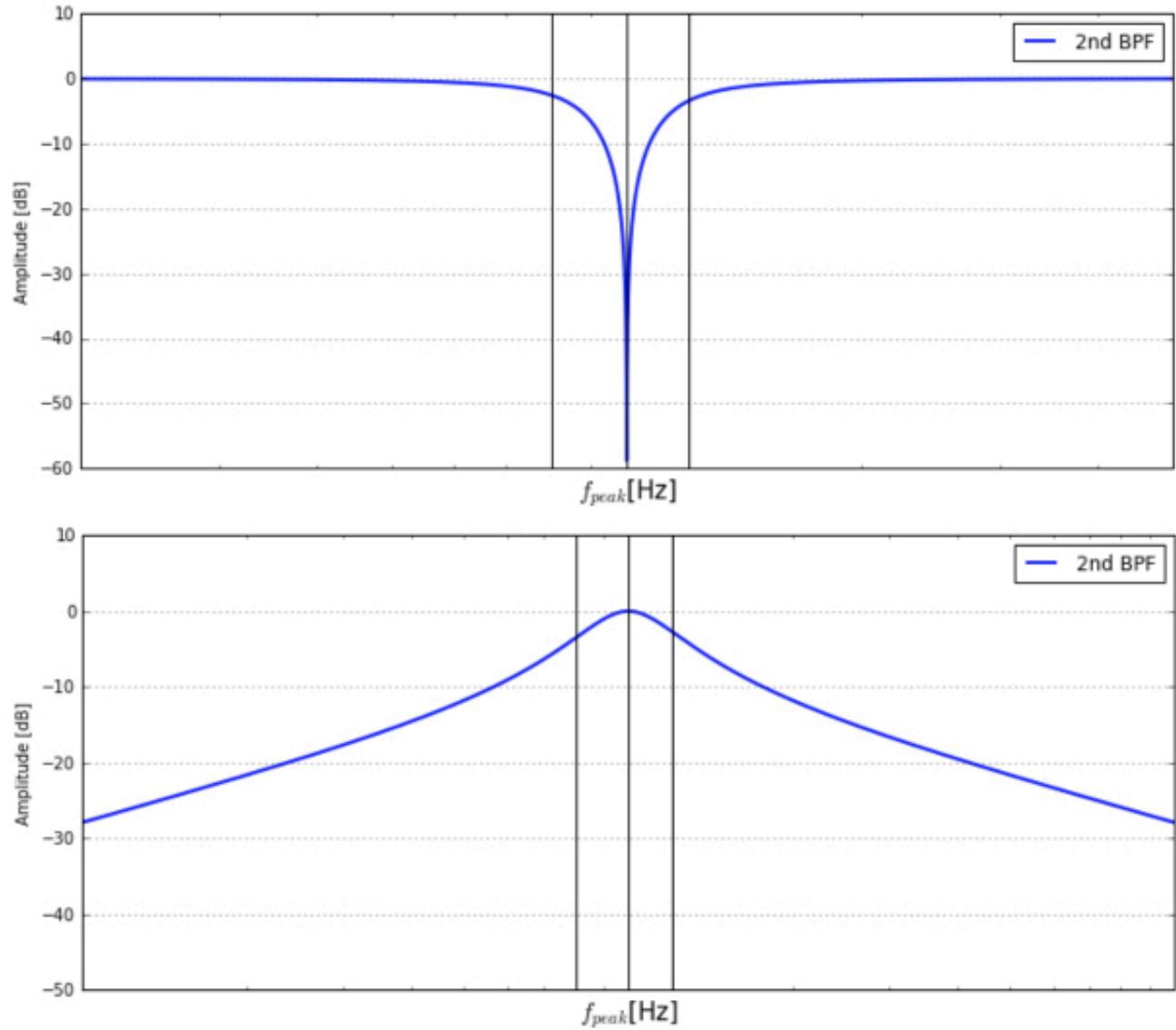
만나게 된 **Python**

그럼 Python으로 뭘 한 거지???

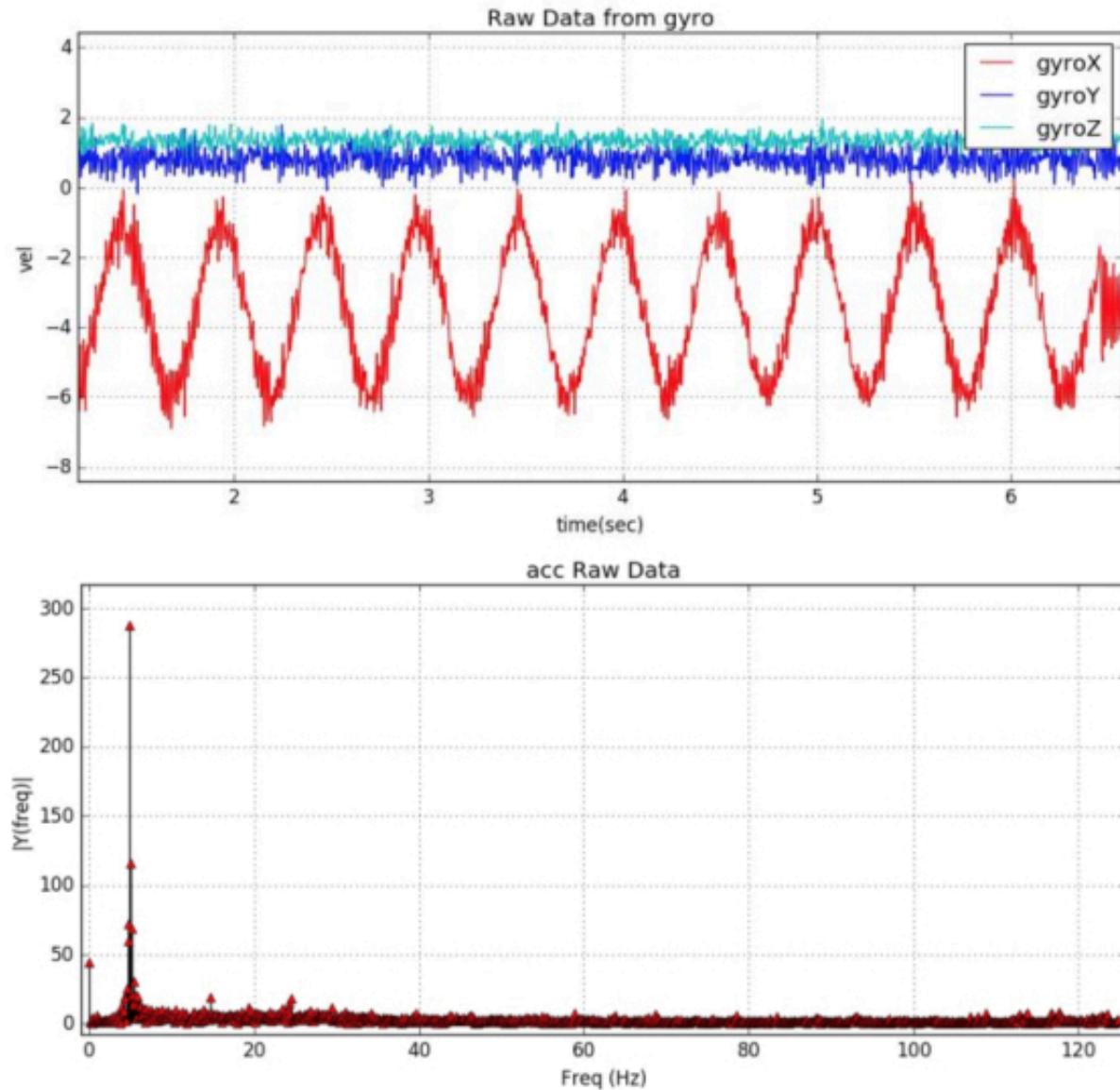
Digital Filter Design



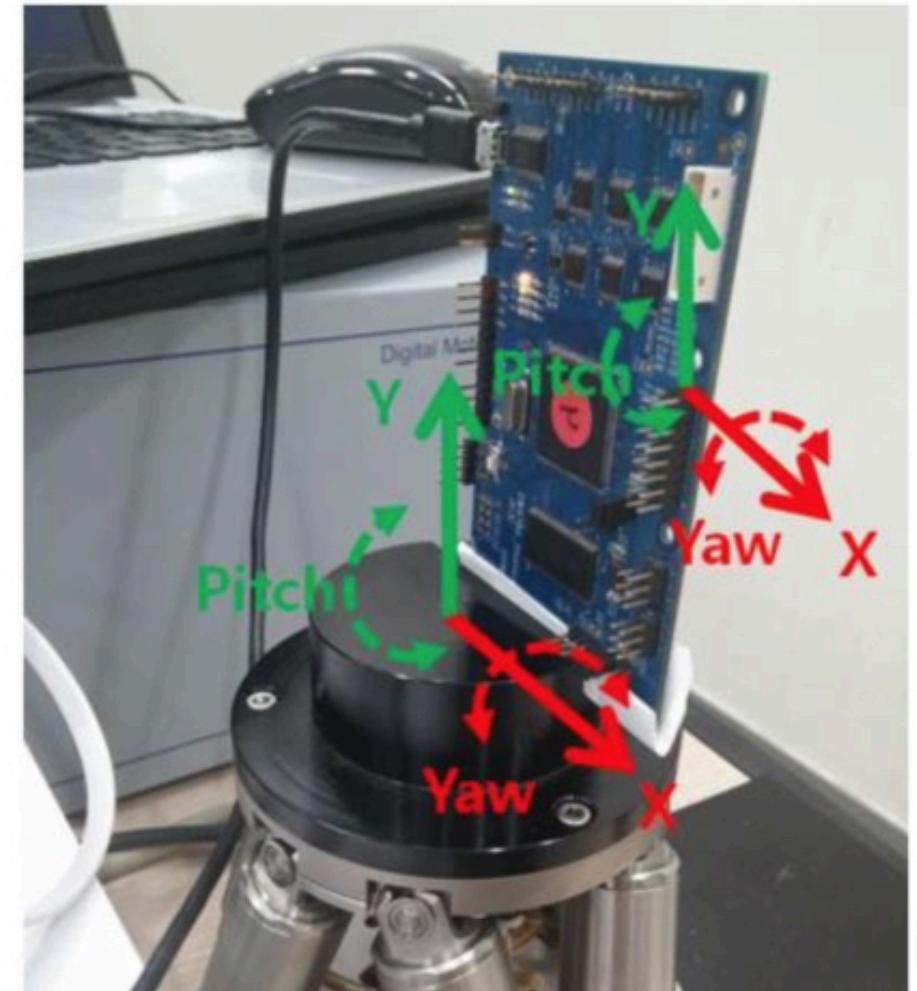
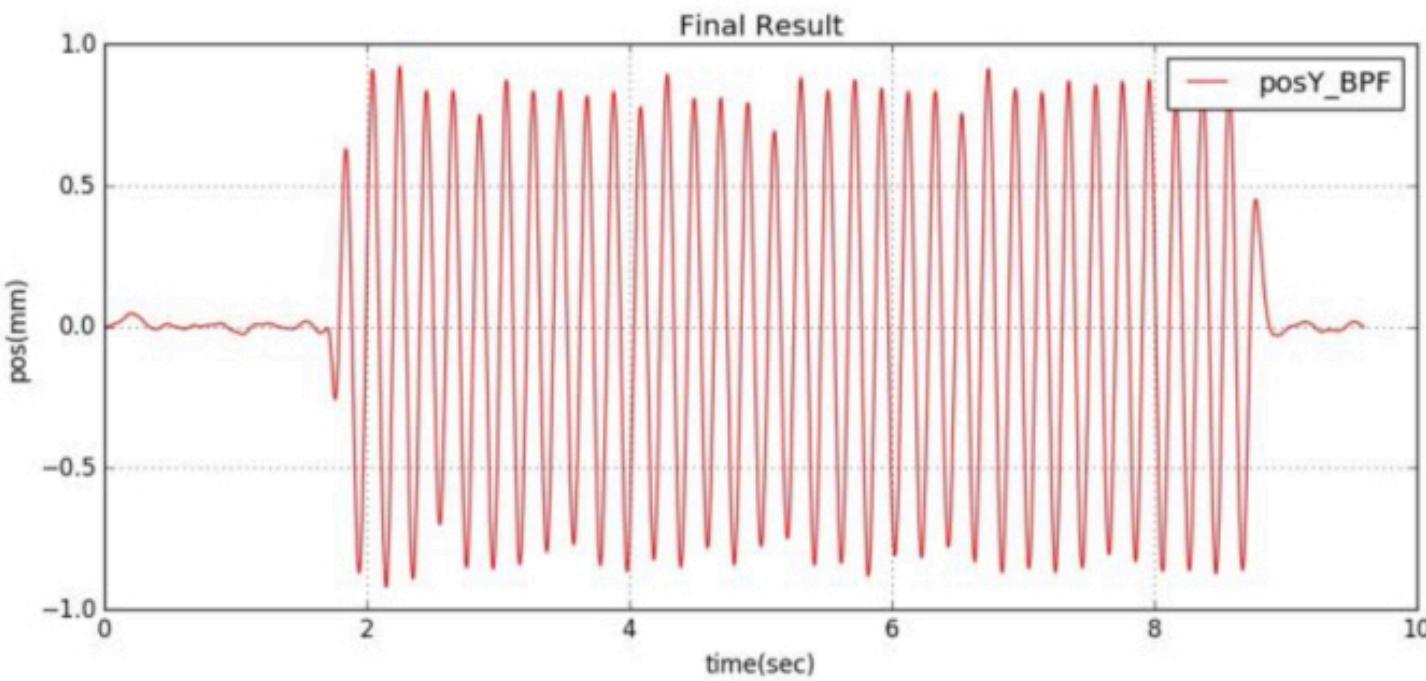
Digital Filter Design



Optical Image Stabilization



Optical Image Stabilization

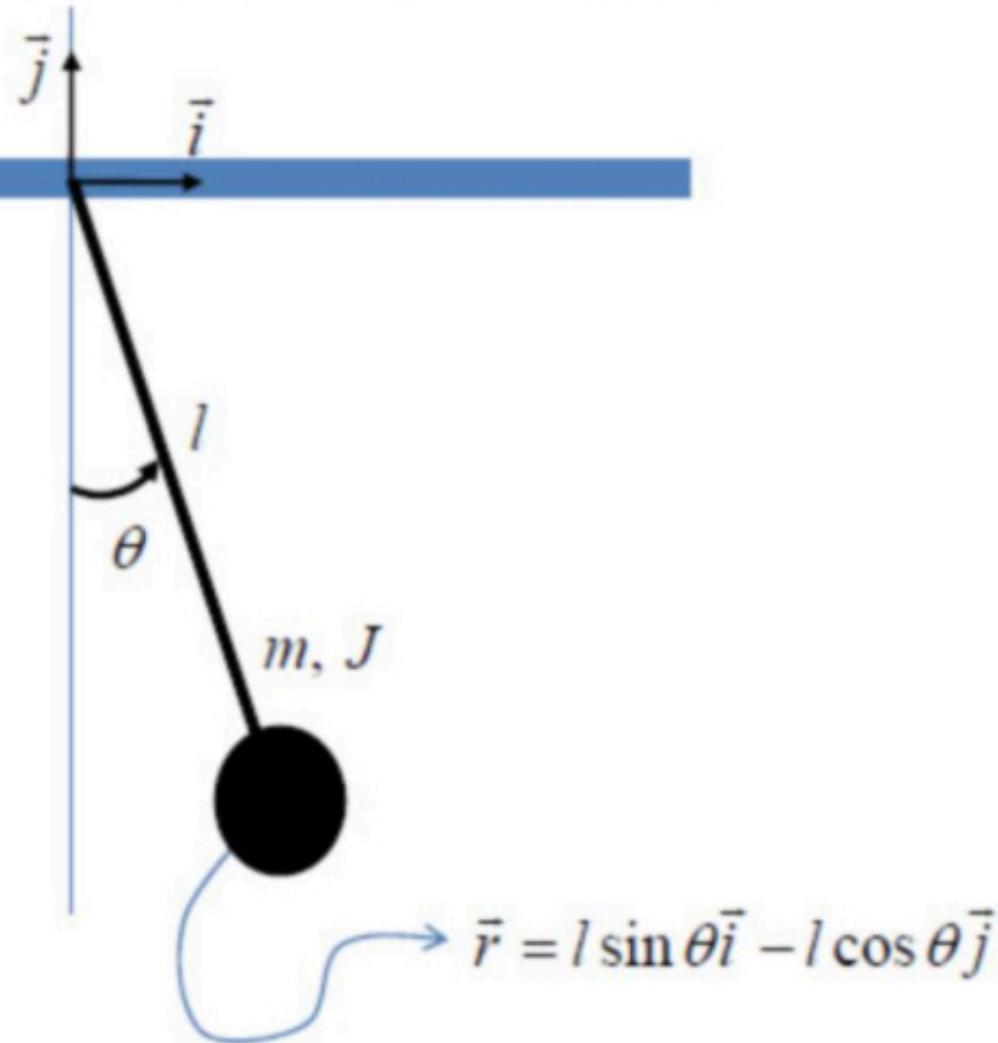


if

간단한 물리 현상을
확인해 보고 싶다면
어떻게 해야 할까?

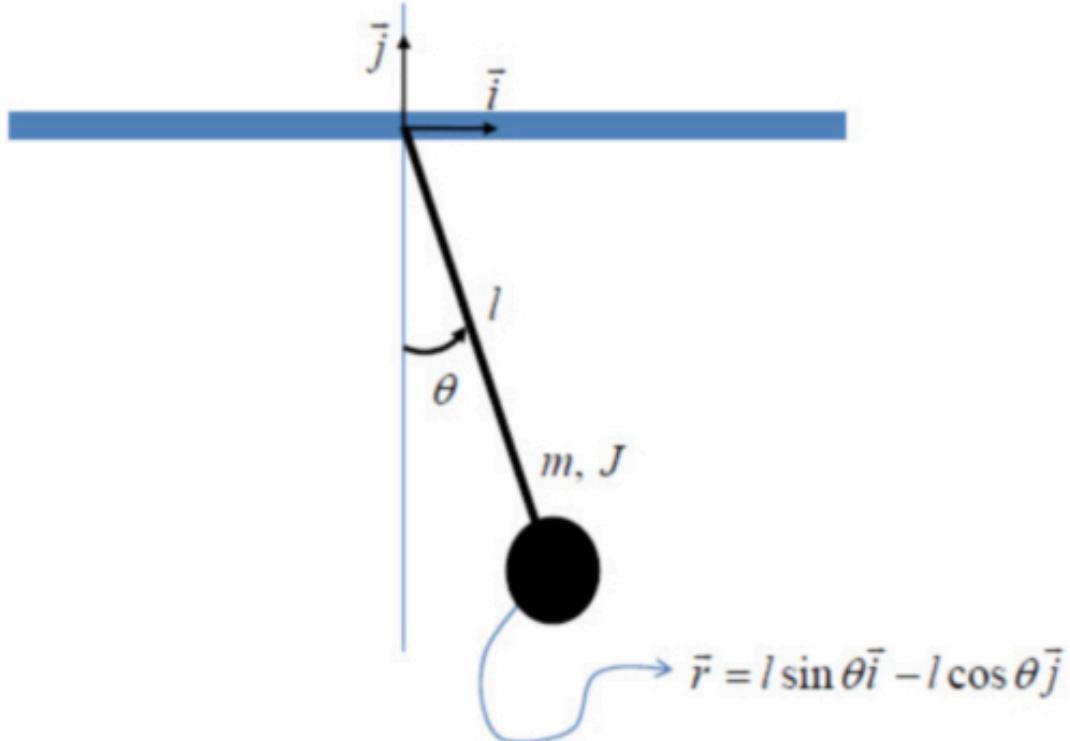
고전적 방법)

역학적 모델링을 위한 초기 작업



고전적 방법)

역학적 모델을 구현



$$\ddot{\theta} = -\frac{f_m}{ml^2 + J} \dot{\theta} - \frac{mgl}{ml^2 + J} \theta$$

고전적 방법) 미분방정식을 푸는 코드 작성

```
def calcODEFunc(tVal, xVal, vVal):
    return -pen_fm/(pen_m*pen_l*pen_l+pen_J)*vVal-pen_m*pen_g*pen_l/(pen_m*pen_l*pen_l+pen_J)*xVal

def solveODEusingRK4(t, x, v):
    kx1 = v
    kv1 = calcODEFunc( t, x, v )

    kx2 = v + h*kv1/2
    kv2 = calcODEFunc( t + h/2, x + h*kx1/2, v + h*kv1/2 )

    kx3 = v + h*kv2/2
    kv3 = calcODEFunc( t + h/2, x + h*kx2/2, v + h*kv2/2 )

    kx4 = v + h*kv3
    kv4 = calcODEFunc( t + h, x + h*kx3, v + h*kv3 )

    dx = h*(kx1 + 2*kx2 + 2*kx3 + kx4)/6
    dv = h*(kv1 + 2*kv2 + 2*kv3 + kv4)/6

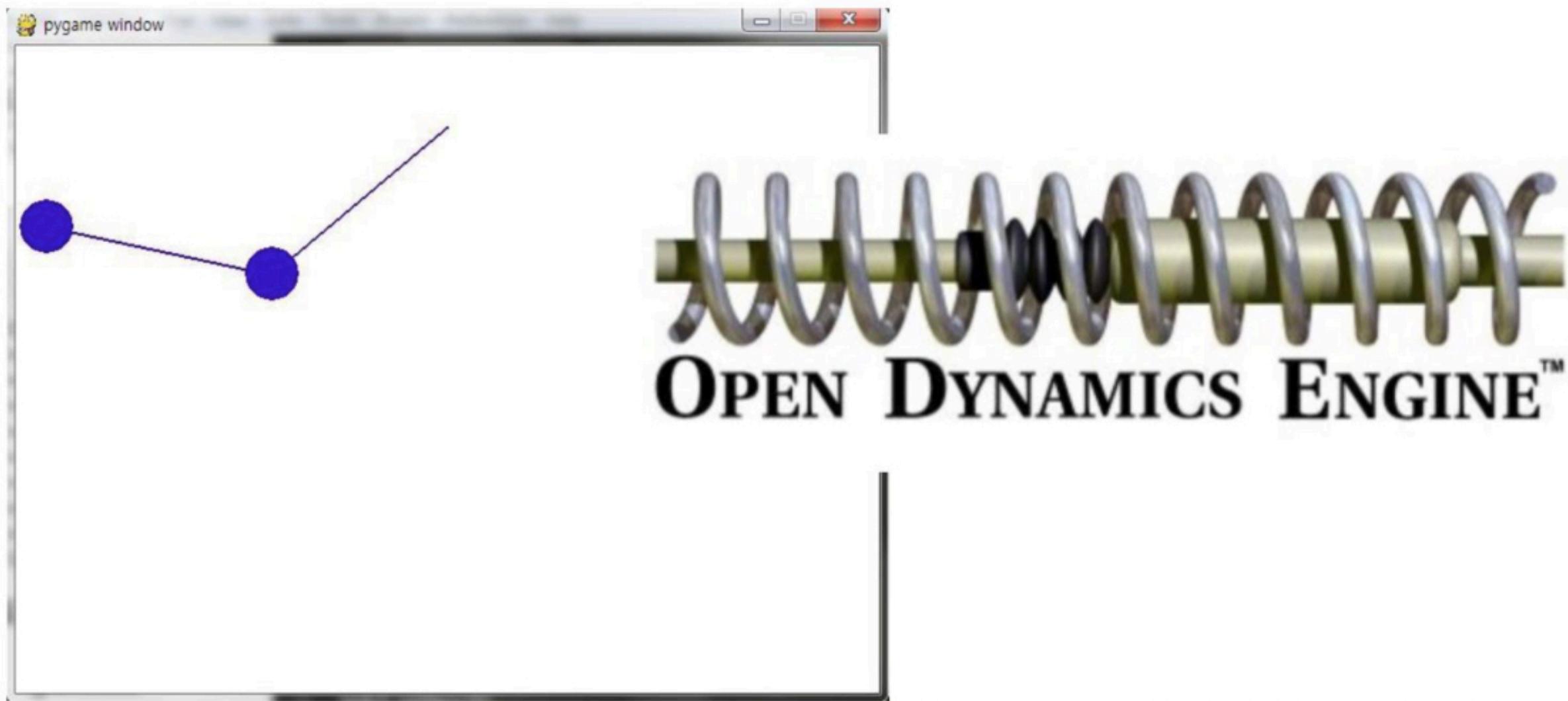
    return x+dx, v+dv
```

$$\ddot{\theta} = -\frac{f_m}{ml^2 + J} \dot{\theta} - \frac{mgl}{ml^2 + J} \theta$$

but, if

역학 모델과 학술적 난이도를 극복하는 것이 아니라
물리 법칙을 단순 적용할 필요가 있어서
간편히 접근하고 싶었다면…

Pythonic한 방법) 더 복잡한 시스템에 대해



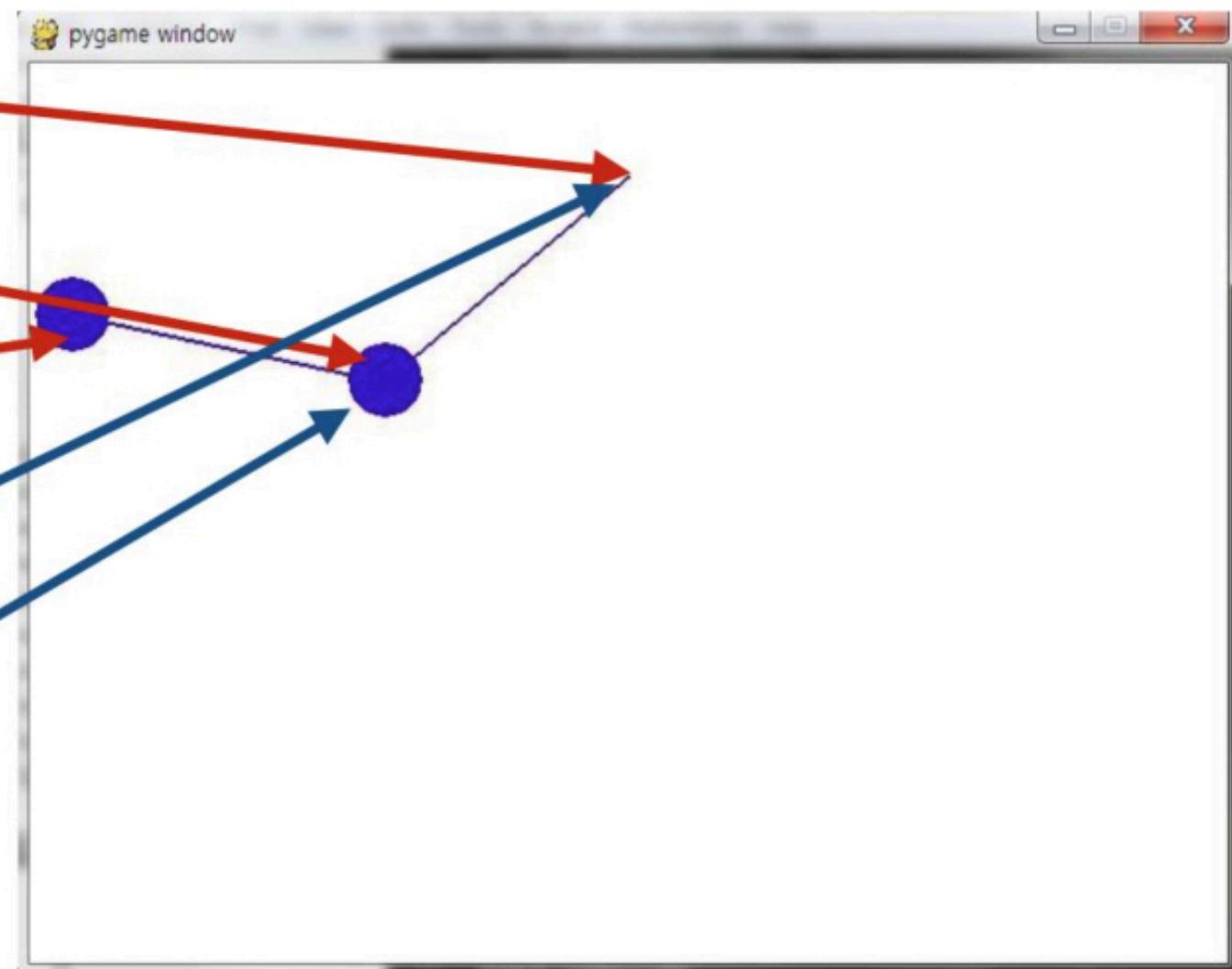
```
# Create a world object
world = ode.World()
world.setGravity((0,-9.81,0))

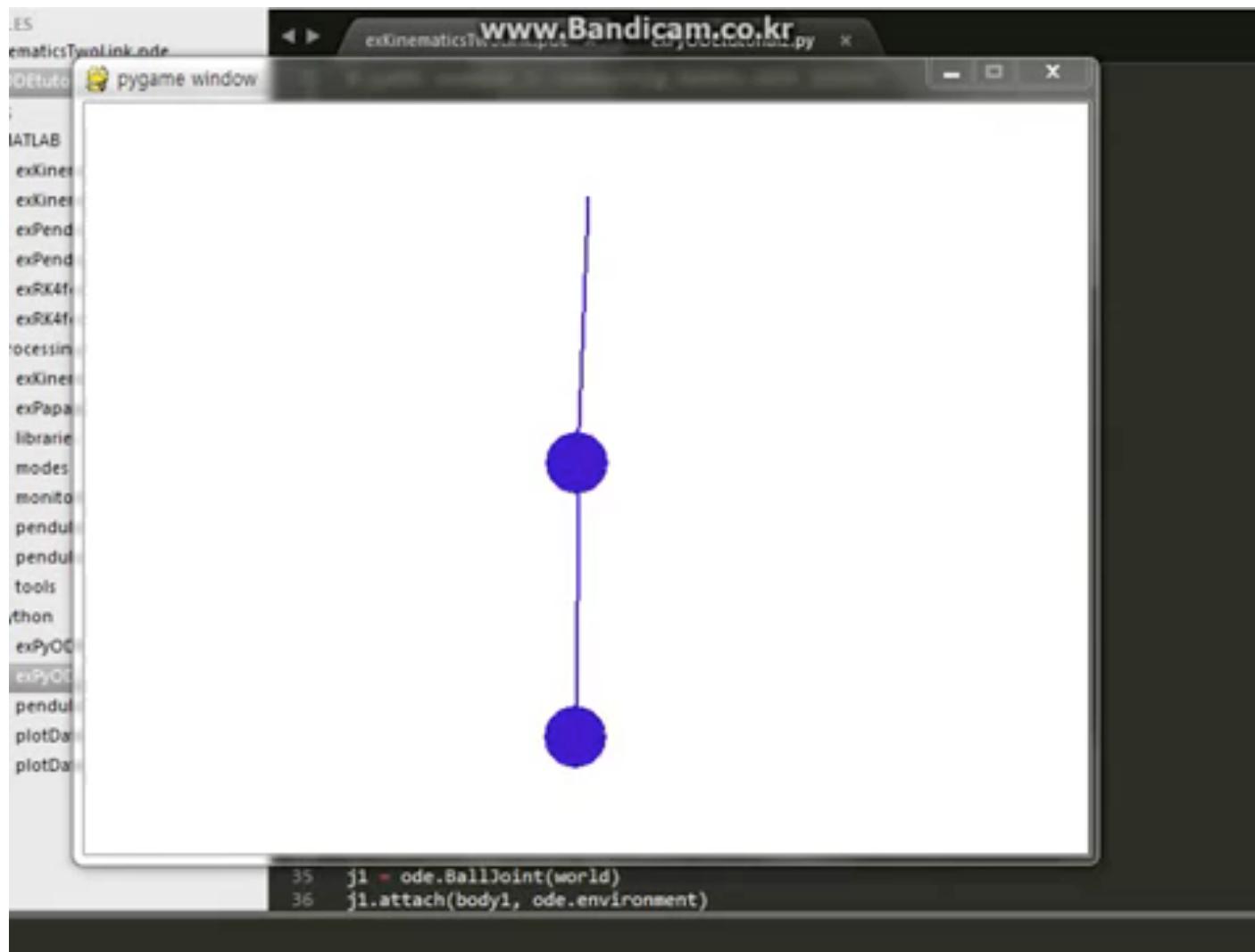
# Create two bodies
body1 = ode.Body(world)
M = ode.Mass()
M.setSphere(2500, 0.05)
body1.setMass(M)
body1.setPosition((1,2,0))

body2 = ode.Body(world)
M = ode.Mass()
M.setSphere(2500, 0.05)
body2.setMass(M)
body2.setPosition((2,2,0))

# Connect body1 with the static environment
j1 = ode.BallJoint(world)
j1.attach(body1, ode.environment)
j1.setAnchor( (0,2,0) )

# Connect body2 with body1
j2 = ode.BallJoint(world)
j2.attach(body1, body2)
j2.setAnchor( (1,2,0) )
```





이건 또 뭐지???

```
#This function will draw base plate
def Draw_Base_Plate():
    #Added two cubes for cutting sides of base plate

    bpy.ops.mesh.primitive_cube_add(radius=0.05, location=(0.175,0,0.09))
    bpy.ops.mesh.primitive_cube_add(radius=0.05, location=(-0.175,0,0.09))

    #Adding base plate
    bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.09))

    #Adding booleab difference modifier from first cube

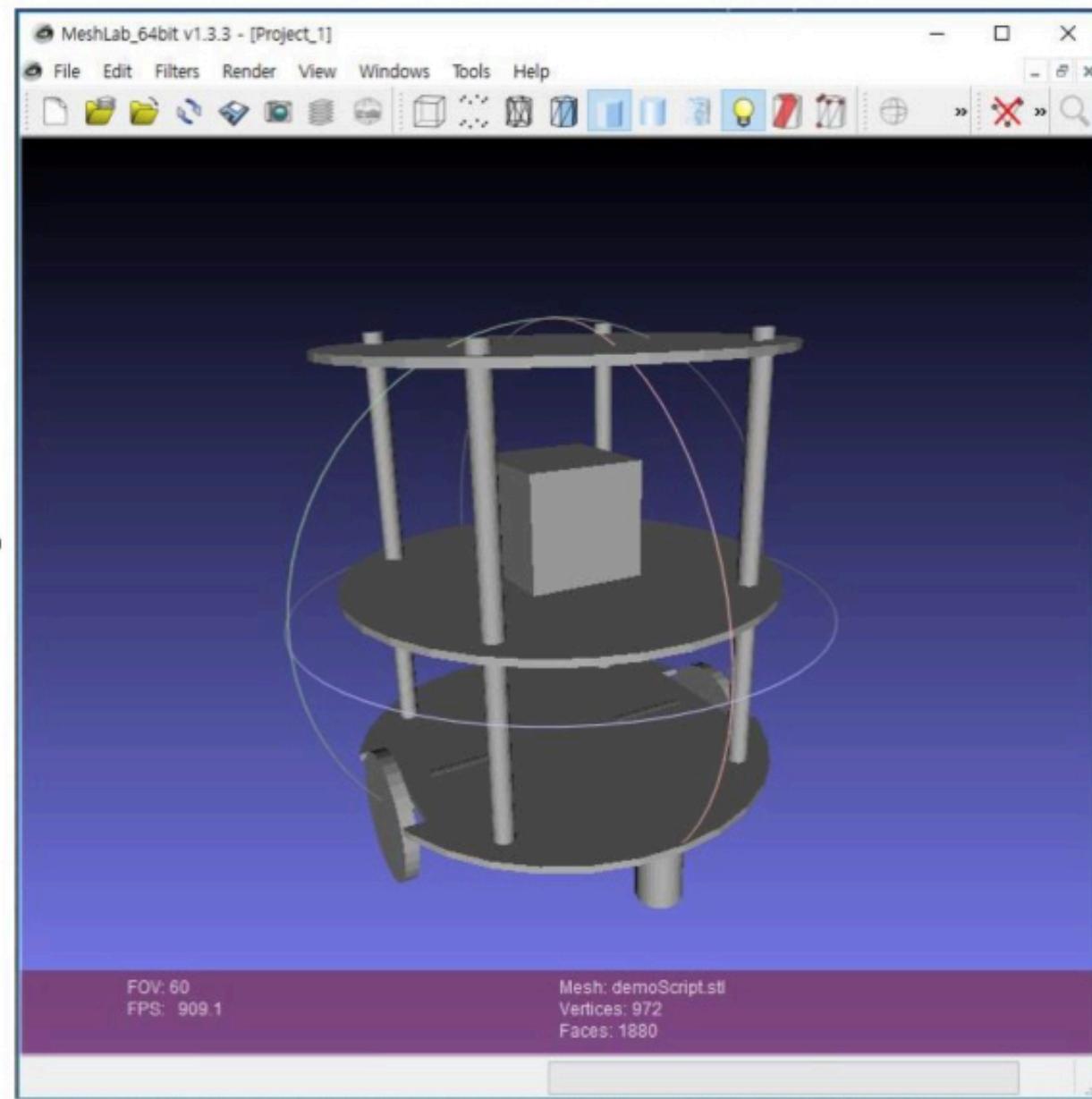
    bpy.ops.object.modifier_add(type='BOOLEAN')
    bpy.context.object.modifiers["Boolean"].operation = 'DIFFERENCE'
    bpy.context.object.modifiers["Boolean"].object = bpy.data.objects["Cube"]
    bpy.ops.object.modifier_apply(modifier="Boolean")

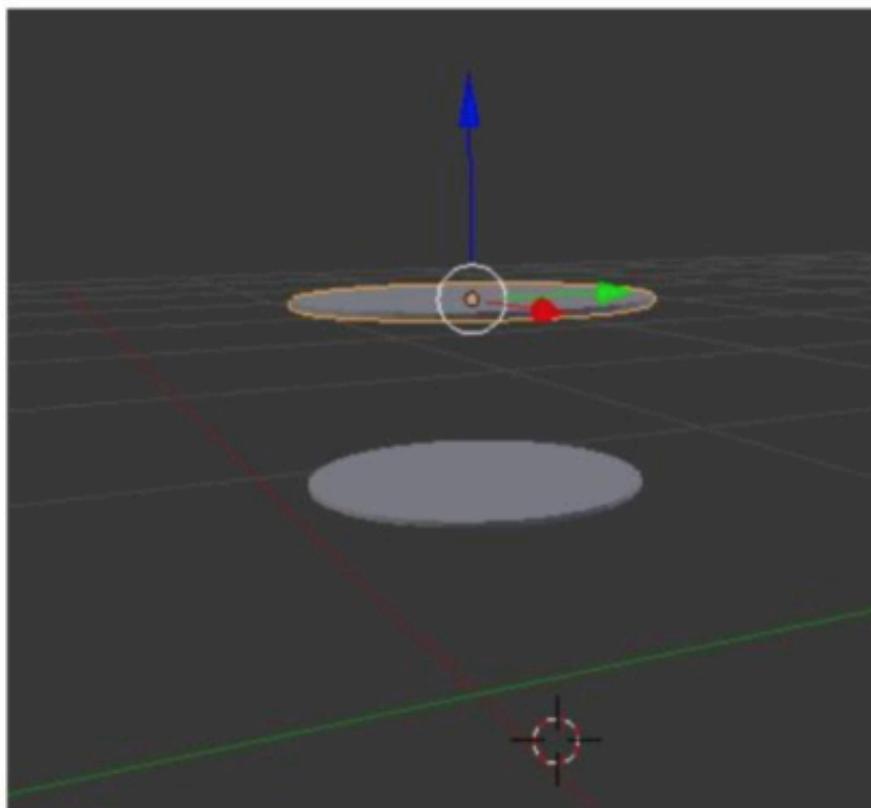
    #Adding booleab difference modifier from second cube

    bpy.ops.object.modifier_add(type='BOOLEAN')
    bpy.context.object.modifiers["Boolean"].operation = 'DIFFERENCE'
    bpy.context.object.modifiers["Boolean"].object = bpy.data.objects["Cube.001"]
    bpy.ops.object.modifier_apply(modifier="Boolean")

    #Deselect cylinder and delete cubes
    bpy.ops.object.select_pattern(pattern="Cube")
    bpy.ops.object.select_pattern(pattern="Cube.001")
    bpy.data.objects['Cylinder'].select = False
    bpy.ops.object.delete(use_global=False)
```

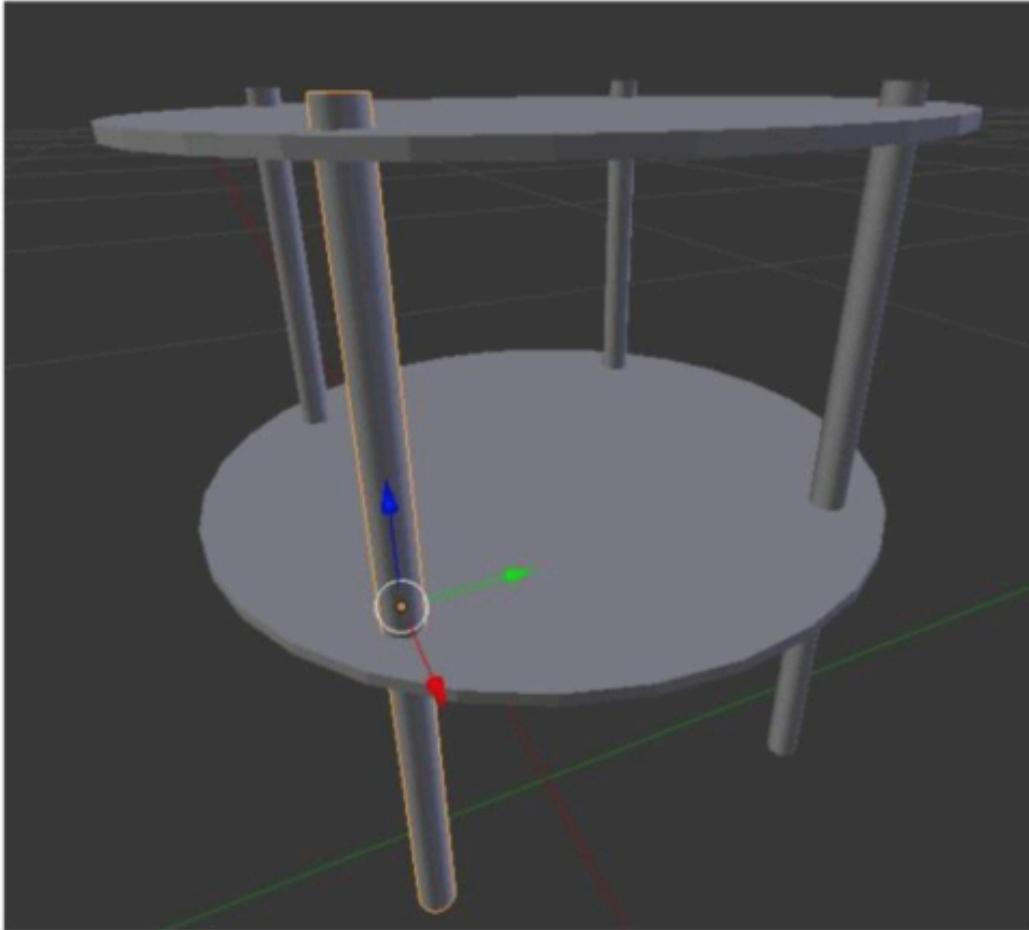
기계 설계가 아니라
동작과 운용 매커니즘을
확인하고 시뮬레이션하기 위한
대상이 필요하다면…





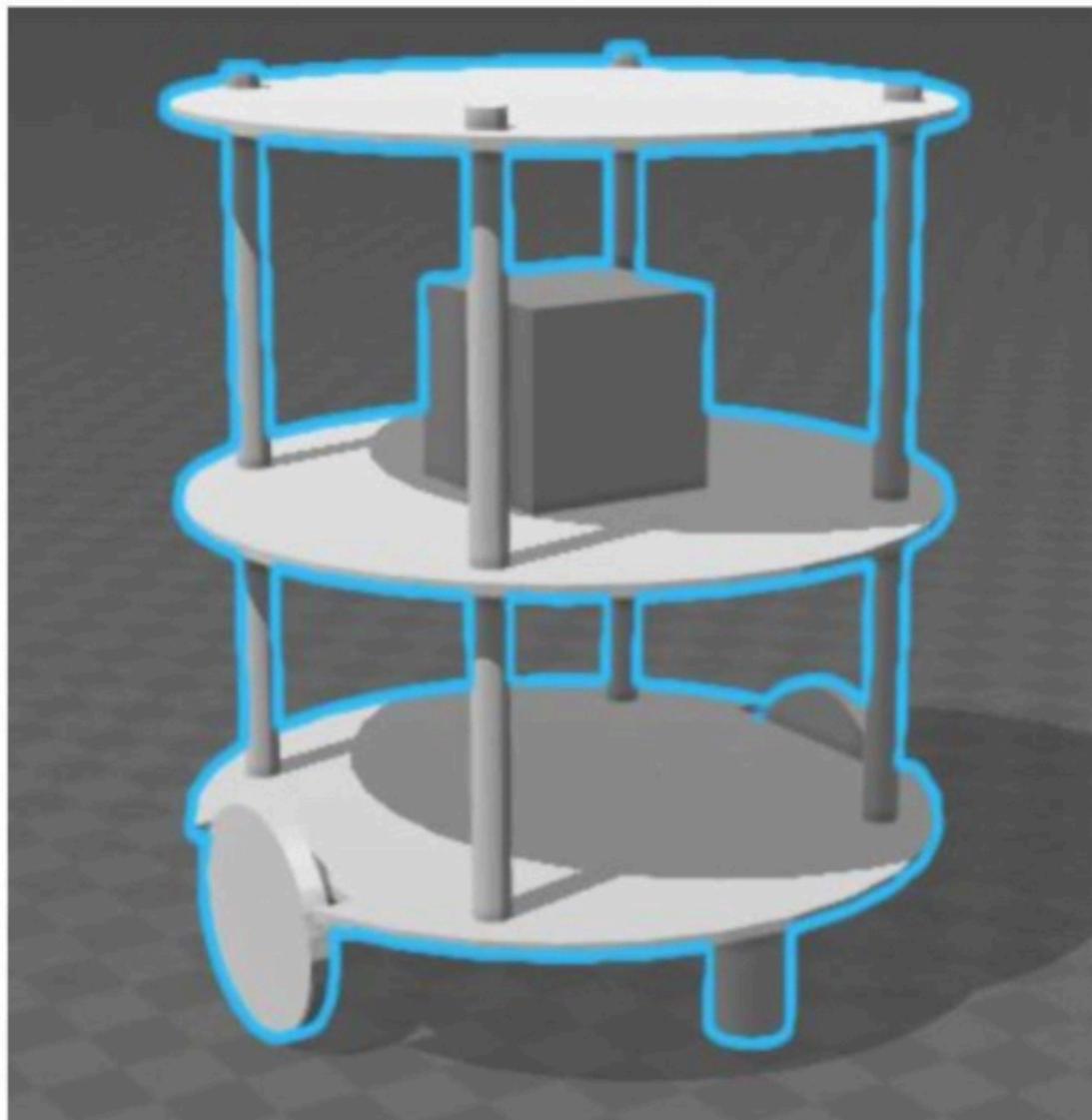
```
#Draw middle plate
def Draw_Middle_Plate():
    bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.22))

#Adding top plate
def Draw_Top_Plate():
    bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.37))
```



```
#Adding support tubes
def Draw_Support_Tubes():
    #Cylinders
    bpy.ops.mesh.primitive_cylinder_add(radius=0.007,depth=0.30, location=(0.09,0.09,0.23))
    bpy.ops.mesh.primitive_cylinder_add(radius=0.007,depth=0.30, location=(-0.09,0.09,0.23))
    bpy.ops.mesh.primitive_cylinder_add(radius=0.007,depth=0.30, location=(-0.09,-0.09,0.23))
    bpy.ops.mesh.primitive_cylinder_add(radius=0.007,depth=0.30, location=(0.09,-0.09,0.23))
```

BLENDER라는 툴로
Python으로 코드를 작성해서
3차원 기구를
Visualize 할 수 있음
이를 다시 GAZEBO와 같은
ROS 툴에서 동작 시뮬레이션



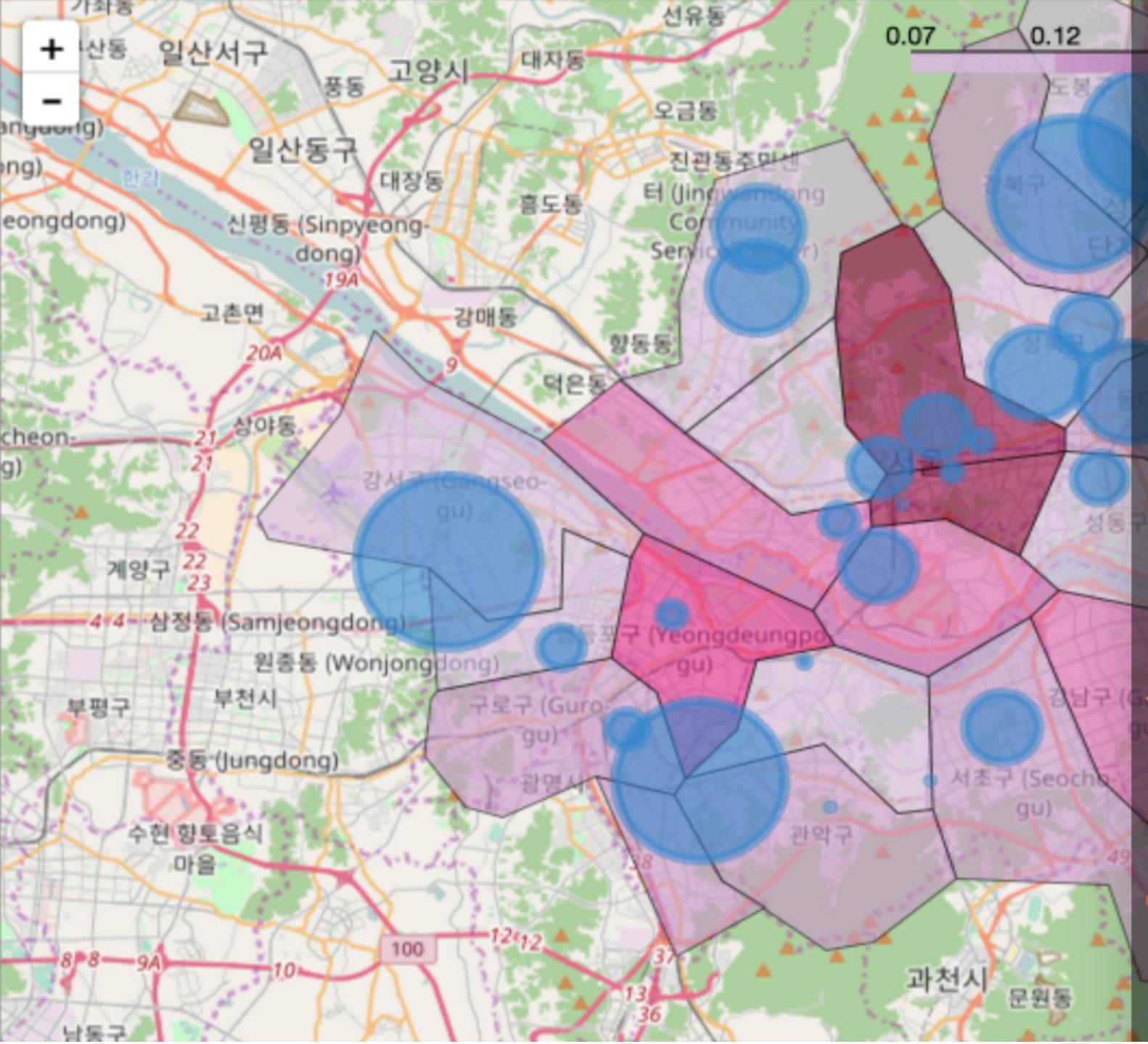
Open Source Platform + Python

= Simulation,

Verification,

AI,

Robot



데이터 과학?
현상,
인식,
가정을 확인하는
도구로 사용해보자

데이터 과학의 목적 중 하나 – 가정(혹은 ‘인식’)을 검증하고 표현하는 것

[국감브리핑] '부자 동네'…서울 강남 3구 체감안전도 높아

종로서 76.6점 1위…관악서 60.0점 최하위

(서울=뉴스1) 성도현 기자 | 2014-10-20 09:54 송고

기사보기

네티즌의견

좋아요

공유하기

0

트윗

blog

인쇄 | + 확대 | - 축소

부유층이 많이 살아 '부자 동네'로 불리는 강남 3구(강남·서초·송파) 지역의 체감안전도가 대체로 높은 것으로 나타났다.

국회 안전행정위원회 소속 강기윤 새누리당 의원이 서울경찰청으로부터 제출받아 20일 공개한 자료에 따르면 2014년 상반기 서울지역 31개 경찰서별 체감안전도 조사 결과 강남지역을 관할



하는 강남서와 수서서, 서초지역을 담당하는 서초서와 방배서, 송파지역을 관할하는 송파서 등 5개 경찰서가 10위권 안에 들었다.

체감안전도가 가장 높은 곳은 종로서(76.6점)였고 수서서 74.5점, 용산서 70.3점, 서초서 70.0 점, 강남서 69.8점, 송파서 69.6점, 방배서 69.3점 등 순으로 나타났다.

원대한 꿈에 비해 아직 우리는 미천(^^)하므로...

CSV 파일 Pandas로 읽기

In [3]:

```
import numpy as np  
import pandas as pd
```

In [2]:

```
crime_raw_data = pd.read_csv('../data/Crime_data.csv', encoding='euc-kr')  
crime_raw_data.head()
```

	구분	죄종	발생검거	건수
0	중부	살인	발생	2.0
1	중부	살인	검거	2.0
2	중부	강도	발생	3.0
3	중부	강도	검거	3.0
4	중부	강간	발생	141.0

- 배포된 데이터를 읽어 보자.
- **pandas**라는 아이는 데이터 파일을 그냥 읽을 수 있다.

데이터 확인하기

In [28]:

```
crime_raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65534 entries, 0 to 65533
Data columns (total 4 columns):
구분      310 non-null object
죄종      310 non-null object
발생검거   310 non-null object
건수      310 non-null float64
dtypes: float64(1), object(3)
memory usage: 2.0+ MB
```

- 이상하다.

```
In [29]: crime_raw_data['죄종'].unique()
```

```
array(['살인', '강도', '강간', '절도', '폭력', nan], dtype=object)
```

- 일단, 죄종은 “살인, 강도, 강간, 절도, 폭력”에 대한 데이터인데
- **nan**이 있다.

데이터 확인하기

```
In [32]: crime_raw_data['죄종'].isnull()
```

```
0      False
1      False
2      False
3      False
4      False
5      False
6      False
7      False
8      False
9      False
10     False
11     False
12     False
13     False
```

- pandas에서는 `isnull()` 함수를 이용해서 `nan`을 찾을 수 있다

데이터 확인하기

In [31]:

```
crime_raw_data[crime_raw_data['죄종'].isnull()]
```

	구분	죄종	발생검거	건수
310	NaN	NaN	NaN	NaN
311	NaN	NaN	NaN	NaN
312	NaN	NaN	NaN	NaN
313	NaN	NaN	NaN	NaN
314	NaN	NaN	NaN	NaN
315	NaN	NaN	NaN	NaN
316	NaN	NaN	NaN	NaN
317	NaN	NaN	NaN	NaN

- 그 결과로 **nan** 으로만 된 행을 조건문으로 가져다 볼 수 있다
- 310번째 행부터** 다 **nan**이다

데이터 확인하기

In [33]:

```
crime_raw_data = crime_raw_data[crime_raw_data['죄종'].notnull()]
crime_raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 310 entries, 0 to 309
Data columns (total 4 columns):
구분      310 non-null object
죄종      310 non-null object
발생검거   310 non-null object
건수      310 non-null float64
dtypes: float64(1), object(3)
memory usage: 12.1+ KB
```

- **nan**이 아닌 행만 다시 저장하자.
- 크기가 **12.1kB**로 줄었다.

데이터 확인하기

- 애초의 목적은?
 - 강남 3구가 안전한지 확인하는 것
- 현재 데이터는
 - 경찰서 이름, 죄종 등으로 그냥 나열된 데이터
- 우리는 구별로 정리된 데이터가 필요하다

	구분	죄종	발생검거	건수
0	중부	살인	발생	2.0
1	중부	살인	검거	2.0
2	중부	강도	발생	3.0
3	중부	강도	검거	3.0
4	중부	강간	발생	141.0
5	중부	강간	검거	96.0
6	중부	절도	발생	1204.0
7	중부	절도	검거	485.0
8	중부	폭력	발생	1335.0
9	중부	폭력	검거	1164.0
10	종로	살인	발생	4.0
11	종로	살인	검거	0.0
12	종로	강도	발생	6.0
13	종로	강도	검거	3.0
14	종로	강간	발생	137.0
15	종로	강간	검거	113.0
16	종로	절도	발생	952.0
17	종로	절도	검거	389.0
18	종로	폭력	발생	1291.0
19	종로	폭력	검거	1135.0

Pivot_table을 이용한 데이터 정리

Pivot_table로 데이터 정리

- 이게 우리 원 데이터(**raw data**)였다
- 경찰서별로 잘 정리된 데이터가 필요하다
- 방법은?
 - **pivot_table**

	구분	죄종	발생검거	건수
0	중부	살인	발생	2.0
1	중부	살인	검거	2.0
2	중부	강도	발생	3.0
3	중부	강도	검거	3.0
4	중부	강간	발생	141.0
5	중부	강간	검거	96.0
6	중부	절도	발생	1204.0
7	중부	절도	검거	485.0
8	중부	폭력	발생	1335.0
9	중부	폭력	검거	1164.0
10	종로	살인	발생	4.0
11	종로	살인	검거	0.0
12	종로	강도	발생	6.0
13	종로	강도	검거	3.0
14	종로	강간	발생	137.0
15	종로	강간	검거	113.0
16	종로	절도	발생	952.0
17	종로	절도	검거	389.0
18	종로	폭력	발생	1291.0
19	종로	폭력	검거	1135.0

Pivot_table로 데이터 정리

In [34]:

```
crime_station = crime_raw_data.pivot_table(  
    crime_raw_data, index=[ "구분" ], columns=[ "죄종" , "발생검거" ], aggfunc=[ np.sum ] )  
  
crime_station.head( )
```

sum											
건수											
죄종	강간		강도		살인		절도		폭력		
	발생검거	검거	발생	검거	발생	검거	발생	검거	발생	검거	발생
구분											
강남	269.0	339.0	26.0	24.0	3.0	3.0	1129.0	2438.0	2096.0	2336.0	
강동	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0	
강북	159.0	217.0	4.0	5.0	6.0	7.0	672.0	1222.0	2482.0	2778.0	
강서	239.0	275.0	10.0	10.0	10.0	9.0	1070.0	1952.0	2768.0	3204.0	
관악	264.0	322.0	10.0	12.0	7.0	6.0	937.0	2103.0	2707.0	3235.0	

- 이런 한 줄로 된 명령이 있었을 줄...

Pivot_table로 데이터 정리

죄종	sum									
	강간	강도	살인	절도	폭력	검거	발생	검거	발생	검거
발생검거	검거	발생	검거	발생	검거	발생	검거	발생	검거	발생
구분										
강남	269.0	339.0	26.0	24.0	3.0	3.0	1129.0	2438.0	2096.0	2336.0
강동	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0
강북	159.0	217.0	4.0	5.0	6.0	7.0	672.0	1222.0	2482.0	2778.0
강서	239.0	275.0	10.0	10.0	10.0	9.0	1070.0	1952.0	2768.0	3204.0
관악	264.0	322.0	10.0	12.0	7.0	6.0	937.0	2103.0	2707.0	3235.0

- 그런데 이 복잡한 컬럼들은 뭐지?

Pivot_table로 데이터 정리

In [35]:

```
crime_station.columns
```

```
MultiIndex(levels=[['sum'], ['건수'], ['강간', '강도', '살인', '절도', '폭력'], ['검거', '발생']],
           labels=[[0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0], [0,
           1, 1, 2, 2, 3, 3, 4, 4], [0, 1, 0, 1, 0, 1, 0, 1, 1]], names=[None, None, '죄종', '발생검거'])
```

구분											
죄종	강간		강도		살인		절도		폭력		구분
	발생검거	검거	발생	검거	발생	검거	발생	검거	발생	검거	발생
sum											
강남	269.0	339.0	26.0	24.0	3.0	3.0	1129.0	2438.0	2096.0	2336.0	
강동	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0	
강북	159.0	217.0	4.0	5.0	6.0	7.0	672.0	1222.0	2482.0	2778.0	
강서	239.0	275.0	10.0	10.0	10.0	9.0	1070.0	1952.0	2768.0	3204.0	
관악	264.0	322.0	10.0	12.0	7.0	6.0	937.0	2103.0	2707.0	3235.0	

Pivot_table로 데이터 정리

In [36]:

```
crime_station.columns = crime_station.columns.droplevel([0,1])
```

```
crime_station.columns
```

```
MultiIndex(levels=[[ '강간', '강도', '살인', '절도', '폭력'], [ '검거', '발생']],
           labels=[[0, 0, 1, 1, 2, 2, 3, 3, 4, 4], [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]],
           names=[ '죄종', '발생검거'])
```

- 일단 제일 필요없는 상위 두 개의 컬럼 이름은 제거하자

Pivot_table로 데이터 정리

```
In [37]: crime_station[ '강도' , '검거' ]
```

구분	
강남	26.0
강동	13.0
강북	4.0
강서	10.0
관악	10.0
광진	6.0
구로	13.0
금천	7.0

- 이 상태에서는 이렇게 접근할 수 있다. 강도의 검거 건수는?

Pivot_table로 데이터 정리

```
In [38]: crime_station[ '살인' , '발생' ]
```

구분	
강남	3.0
강동	4.0
강북	7.0
강서	9.0
관악	6.0
광진	4.0
구로	9.0
금천	6.0

- 살인이 발생된 건수는?

Pivot_table로 데이터 정리

In [39]:

```
crime_station.head()
```

죄종	강간		강도		살인		절도		폭력	
	발생	검거	검거	발생	검거	발생	검거	발생	검거	발생
구분										
강남	269.0	339.0	26.0	24.0	3.0	3.0	1129.0	2438.0	2096.0	2336.0
강동	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0
강북	159.0	217.0	4.0	5.0	6.0	7.0	672.0	1222.0	2482.0	2778.0
강서	239.0	275.0	10.0	10.0	10.0	9.0	1070.0	1952.0	2768.0	3204.0
관악	264.0	322.0	10.0	12.0	7.0	6.0	937.0	2103.0	2707.0	3235.0

- 그러나 정리가 되긴 했지만 현재 **2중** 컬럼 이름은 불필요해 보인다
- 오히려 컬럼을 강도검거, 강도발생 등으로 한 줄로 보는 것이 더 편해 보임

Pivot_table로 데이터 정리

In [40]:

```
tmp = crime_station.columns.get_level_values(0) + \
      crime_station.columns.get_level_values(1)
tmp
```

```
Index(['강간검거', '강간발생', '강도검거', '강도발생', '살인검거', '살인발생', '절도검거', '절도발생', '폭  
력검거',  
       '폭력발생'],  
      dtype='object')
```

- 이렇게 두 개의 컬럼을 더해보면 된다.

Pivot_table로 데이터 정리

```
In [41]: crime_station.columns = tmp  
crime_station.head( )
```

구분	강간검거	강간발생	강도검거	강도발생	살인검거	살인발생	절도검거	절도발생	폭력검거	폭력발생
강남	269.0	339.0	26.0	24.0	3.0	3.0	1129.0	2438.0	2096.0	2336.0
강동	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0
강북	159.0	217.0	4.0	5.0	6.0	7.0	672.0	1222.0	2482.0	2778.0
강서	239.0	275.0	10.0	10.0	10.0	9.0	1070.0	1952.0	2768.0	3204.0
관악	264.0	322.0	10.0	12.0	7.0	6.0	937.0	2103.0	2707.0	3235.0

- 그리고 그 두 컬럼을 합친 하나의 컬럼을 선택하면 된다

Pivot_table로 데이터 정리

- 문제는 이 데이터들이 구별로 정리된 것이 아니라 경찰서별로 정리된 것이라는 것
- 그렇다면 경찰서 이름으로 경찰서가 위치해 있는 구 이름을 알아야 한다는 것

구분	강간검거	강간발생	강도검거	강도발생	살인검거	살인발생	절도검거	절도발생	폭력검거	폭력발생
강남	269.0	339.0	26.0	24.0	3.0	3.0	1129.0	2438.0	2096.0	2336.0
강동	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0
강북	159.0	217.0	4.0	5.0	6.0	7.0	672.0	1222.0	2482.0	2778.0
강서	239.0	275.0	10.0	10.0	10.0	9.0	1070.0	1952.0	2768.0	3204.0
관악	264.0	322.0	10.0	12.0	7.0	6.0	937.0	2103.0	2707.0	3235.0
광진	234.0	279.0	6.0	11.0	4.0	4.0	1057.0	2636.0	2011.0	2392.0
구로	181.0	273.0	13.0	10.0	9.0	9.0	861.0	1910.0	2680.0	3164.0
금천	143.0	175.0	7.0	7.0	6.0	6.0	654.0	1264.0	1946.0	2193.0
남대문	52.0	57.0	4.0	5.0	1.0	1.0	429.0	946.0	832.0	890.0
노원	142.0	159.0	9.0	6.0	6.0	5.0	740.0	1857.0	2124.0	2516.0
도봉	121.0	123.0	5.0	5.0	3.0	3.0	492.0	866.0	1309.0	1441.0
동대문	158.0	190.0	12.0	10.0	5.0	5.0	1071.0	1940.0	2377.0	2642.0
동작	149.0	325.0	7.0	7.0	6.0	8.0	554.0	1226.0	1444.0	1661.0
마포	320.0	399.0	7.0	4.0	4.0	4.0	940.0	2527.0	2500.0	2939.0
방배	51.0	78.0	5.0	4.0	1.0	1.0	293.0	472.0	446.0	500.0
서대문	147.0	175.0	4.0	5.0	7.0	6.0	752.0	1503.0	1644.0	1976.0
서부	49.0	54.0	2.0	4.0	4.0	4.0	389.0	781.0	980.0	1128.0
서초	220.0	350.0	5.0	9.0	4.0	4.0	814.0	1733.0	1583.0	1838.0
성동	78.0	104.0	5.0	3.0	5.0	4.0	896.0	1296.0	1408.0	1619.0
성북	75.0	103.0	3.0	3.0	2.0	2.0	381.0	790.0	1004.0	1176.0
송파	184.0	234.0	12.0	15.0	8.0	9.0	1048.0	2543.0	2808.0	3289.0
수서	144.0	177.0	16.0	15.0	2.0	2.0	789.0	1149.0	1431.0	1666.0
양천	108.0	131.0	7.0	7.0	5.0	5.0	755.0	1719.0	1918.0	2250.0
영등포	225.0	356.0	14.0	19.0	13.0	13.0	940.0	2341.0	3007.0	3593.0

구 이름 추출

In [34]:

```
police_station_pos = {  
    '강남':'강남구', '강동':'강동구', '강서':'강서구', '관악':'관악구',  
    '구로':'구로구', '남대문':'중구', '금천':'금천구', '동작':'동작구',  
    '노원':'노원구', '도봉':'도봉구', '혜화':'종로구', '광진':'광진구',  
    '마포':'마포구', '방배':'서초구', '강북':'강북구', '서부':'은평구',  
    '서대문':'서대문구', '서초':'서초구', '성동':'성동구',  
    '성북':'성북구', '송파':'송파구', '수서':'강남구', '양천':'양천구',  
    '영등포':'영등포구', '용산':'용산구', '은평':'은평구', '종로':'종로구',  
    '종암':'성북구', '중랑':'중랑구', '중부':'중구', '동대문':'동대문구'  
}
```

In [36]:

```
police_station_pos.get('동대문')
```

'동대문구'

In [38]:

```
tmp = [police_station_pos.get(idx) for idx, row in crime_station.iterrows()]
tmp
```

```
[ '강남구',
  '강동구',
  '강북구',
  '강서구',
  '관악구',
  '광진구',
  '구로구',
  '금천구',
  '중구',
  '노원구',
  '도봉구',
  '동대문구',
  '동작구',
  '마포구',
```

In [39]:

```
crime_station['구'] = tmp  
crime_station
```

구분	강간검거	강간발생	강도검거	강도발생	살인검거	살인발생	절도검거	절도발생	폭력검거	폭력발생	구
강남	269.0	339.0	26.0	24.0	3.0	3.0	1129.0	2438.0	2096.0	2336.0	강남구
강동	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0	강동구
강북	159.0	217.0	4.0	5.0	6.0	7.0	672.0	1222.0	2482.0	2778.0	강북구
강서	239.0	275.0	10.0	10.0	10.0	9.0	1070.0	1952.0	2768.0	3204.0	강서구
관악	264.0	322.0	10.0	12.0	7.0	6.0	937.0	2103.0	2707.0	3235.0	관악구
광진	234.0	279.0	6.0	11.0	4.0	4.0	1057.0	2636.0	2011.0	2392.0	광진구

In [40]:

```
crime_station['구'].unique()
```

```
array(['강남구', '강동구', '강북구', '강서구', '관악구', '광진구', '구로구', '금천구', '중구', '노원구',
       '도봉구', '동대문구', '동작구', '마포구', '서초구', '서대문구', '은평구', '성동구', '성북구',
       '송파구', '양천구', '영등포구', '용산구', '종로구', '중랑구'], dtype=object)
```

In [41]:

```
len(crime_station['구'].unique())
```

25

In [42]:

```
crime_station.to_csv('../data/crime_station.csv', sep=',', encoding='UTF-8')
```

In [43]:

```
crime_gu = pd.pivot_table(crime_station, index='구', aggfunc=np.sum)  
crime_gu
```

구	강간검거	강간발생	강도검거	강도발생	살인검거	살인발생	절도검거	절도발생	폭력검거	폭력발생
	강남구	413.0	516.0	42.0	39.0	5.0	5.0	1918.0	3587.0	3527.0
강동구	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0
강북구	159.0	217.0	4.0	5.0	6.0	7.0	672.0	1222.0	2482.0	2778.0
강서구	239.0	275.0	10.0	10.0	10.0	9.0	1070.0	1952.0	2768.0	3204.0
관악구	264.0	322.0	10.0	12.0	7.0	6.0	937.0	2103.0	2707.0	3235.0

- **Pivot_table**로 다시 한번 구 이름으로 정리하면 된다.

In [9]:

```
crime_gu[ '강도검거' ]/crime_gu[ '강도발생' ]
```

구	
강남구	1.076923
강동구	0.928571
강북구	0.800000
강서구	1.000000
관악구	0.833333
광진구	0.545455
구로구	1.300000

- 이렇게. 해도 된다. 5대 범죄니까 5번하면 된다.
- 그러나...

In [10]:

```
crime_gu[ [ '강도검거' , '살인검거' ] ].div(crime_gu[ '강도발생' ], axis=0)
```

구	강도검거	살인검거
강남구	1.076923	0.128205
강동구	0.928571	0.357143
강북구	0.800000	1.200000
강서구	1.000000	1.000000
관악구	0.833333	0.583333
광진구	0.545455	0.363636

- 여러 컬럼을 한 컬럼으로 나눌 때는 이렇게 해도 된다.
- 우리가 원하는 건 아니지만

In [11]:

```
num = ['강간검거', '강도검거', '살인검거', '절도검거', '폭력검거']
den = ['강간발생', '강도발생', '살인발생', '절도발생', '폭력발생']

crime_gu[num].div(crime_gu[den].values)
```

구	강간검거	강도검거	살인검거	절도검거	폭력검거
강남구	0.800388	1.076923	1.000000	0.534709	0.881309
강동구	0.950000	0.928571	1.250000	0.514253	0.869960
강북구	0.732719	0.800000	0.857143	0.549918	0.893449
강서구	0.869091	1.000000	1.111111	0.548156	0.863920
관악구	0.819876	0.833333	1.166667	0.445554	0.836785

- 각각의 컬럼을 분모(den), 분자(num)로 나눠주는 것은 위의 방법

컬럼간 연산을 통한 데이터 정리

In [12]:

```
target = ['강간검거율', '강도검거율', '살인검거율', '절도검거율', '폭력검거율']
num = ['강간검거', '강도검거', '살인검거', '절도검거', '폭력검거']
den = ['강간발생', '강도발생', '살인발생', '절도발생', '폭력발생']

crime_gu[target] = crime_gu[num].div(crime_gu[den].values)*100
crime_gu.head()
```

구	강간 검거	강간 발생	강도 검거	강도 발생	살인 검거	살 인 발 생	절도검 거	절도발 생	폭력검 거	폭력발 생	강간검거율	강도검거율	살인검거율	절도검거 율	폭력검거 율
강 남 구	413.0	516.0	42.0	39.0	5.0	5.0	1918.0	3587.0	3527.0	4002.0	80.038760	107.692308	100.000000	53.470867	88.130935
강 동 구	152.0	160.0	13.0	14.0	5.0	4.0	902.0	1754.0	2201.0	2530.0	95.000000	92.857143	125.000000	51.425314	86.996047

- 여기에 만들어질 컬럼까지 지정해주면 OK

In [13]:

```
crime_gu = crime_gu.drop(columns=num)
crime_gu.head()
```

구	강간발생	강도발생	살인발생	절도발생	폭력발생	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율
강남구	516.0	39.0	5.0	3587.0	4002.0	80.038760	107.692308	100.000000	53.470867	88.130935
강동구	160.0	14.0	4.0	1754.0	2530.0	95.000000	92.857143	125.000000	51.425314	86.996047
강북구	217.0	5.0	7.0	1222.0	2778.0	73.271889	80.000000	85.714286	54.991817	89.344852
강서구	275.0	10.0	9.0	1952.0	3204.0	86.909091	100.000000	111.111111	54.815574	86.392010
관악구	322.0	12.0	6.0	2103.0	3235.0	81.987578	83.333333	116.666667	44.555397	83.678516

- 검거율을 사용하고, 검거 컬럼들(5개)은 삭제

구	강간발생	강도발생	살인발생	절도발생	폭력발생	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율
강남구	516.0	39.0	5.0	3587.0	4002.0	80.038760	107.692308	100.000000	53.470867	88.130935
강동구	160.0	14.0	4.0	1754.0	2530.0	95.000000	92.857143	125.000000	51.425314	86.996047
강북구	217.0	5.0	7.0	1222.0	2778.0	73.271889	80.000000	85.714286	54.991817	89.344852
강서구	275.0	10.0	9.0	1952.0	3204.0	86.909091	100.000000	111.111111	54.815574	86.392010
관악구	322.0	12.0	6.0	2103.0	3235.0	81.987578	83.333333	116.666667	44.555397	83.678516

- 검거 시점의 문제로 **100%**가 넘는 검거율이 존재한다.
- 그대로 둬도 되지만, 나중에 그래프에서의 일관성을 위해 손을 대자...

In [14]:

crime_gu[target] > 100

구	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율
강남구	False	True	False	False	False
강동구	False	False	True	False	False
강북구	False	False	False	False	False
강서구	False	False	True	False	False
관악구	False	False	True	False	False
광진구	False	False	False	False	False
구로구	False	True	False	False	False
금천구	False	False	False	False	False
노원구	False	True	True	False	False
도봉구	False	False	False	False	False

- 위와 같이 조건을 주면 100이상인 경우만 True를 받을 수 있다.

In [15]:

```
crime_gu[crime_gu[target] > 100] = 100
crime_gu
```

구	강간발생	강도발생	살인발생	절도발생	폭력발생	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율
강남구	516.0	39.0	5.0	3587.0	4002.0	80.038760	100.000000	100.000000	53.470867	88.130935
강동구	160.0	14.0	4.0	1754.0	2530.0	95.000000	92.857143	100.000000	51.425314	86.996047
강북구	217.0	5.0	7.0	1222.0	2778.0	73.271889	80.000000	85.714286	54.991817	89.344852
강서구	275.0	10.0	9.0	1952.0	3204.0	86.909091	100.000000	100.000000	54.815574	86.392010
관악구	322.0	12.0	6.0	2103.0	3235.0	81.987578	83.333333	100.000000	44.555397	83.678516
광진구	279.0	11.0	4.0	2636.0	2392.0	83.870968	54.545455	100.000000	40.098634	84.071906

- 그럴때 강제로 100으로 맞추게 한다.
- 단, 이 코드는 일부 **Pandas** 버전에서는 동작하지 않는다

```
In [16]: crime_gu.rename(columns = {'강간발생': '강간',
                               '강도발생': '강도',
                               '살인발생': '살인',
                               '절도발생': '절도',
                               '폭력발생': '폭력'},
                           inplace=True)
crime_gu.head()
```

구	강간	강도	살인	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율
강남구	516.0	39.0	5.0	3587.0	4002.0	80.038760	100.000000	100.000000	53.470867	88.130935

- 일부 컬럼의 이름을 변경한다

구	강간	강도	살인	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율
	516.0	39.0	5.0	3587.0	4002.0	80.038760	100.000000	100.000000	53.470867	88.130935
강동구	160.0	14.0	4.0	1754.0	2530.0	95.000000	92.857143	100.000000	51.425314	86.996047
강북구	217.0	5.0	7.0	1222.0	2778.0	73.271889	80.000000	85.714286	54.991817	89.344852
강서구	275.0	10.0	9.0	1952.0	3204.0	86.909091	100.000000	100.000000	54.815574	86.392010
관악구	322.0	12.0	6.0	2103.0	3235.0	81.987578	83.333333	100.000000	44.555397	83.678516

- 5대 범죄의 수치적 차이가 극심하다
- 문제는 5대 범죄를 한 번에 시각화할 때 살인같은 경우는 잘 들어나지 않는다
- 범죄간 경중을 따질 수 없지만, 각 컬럼을 정규화하여 최대값을 같게 만든다

In [17]:

```
col = ['살인', '강도', '강간', '절도', '폭력']
crime_gu_norm = crime_gu[col] / crime_gu[col].max()
crime_gu_norm.head()
```

	살인	강도	강간	절도	폭력
구					
강남구	0.384615	1.000000	1.000000	1.000000	1.000000
강동구	0.307692	0.358974	0.310078	0.488988	0.632184
강북구	0.538462	0.128205	0.420543	0.340675	0.694153
강서구	0.692308	0.256410	0.532946	0.544187	0.800600
관악구	0.461538	0.307692	0.624031	0.586284	0.808346

- 모든 요소가 양수이므로 정규화할 컬럼을 정하고,
- 각 컬럼의 최대값을 각각의 컬럼에 나눠주면 된다.
- 주의) 새로 **DataFrame**을 만들었다. (**crime_gu_norm**)

In [18]:

```
col2 = ['강간검거율', '강도검거율', '살인검거율', '절도검거율', '폭력검거율']
crime_gu_norm[col2] = crime_gu[col2]
crime_gu_norm.head()
```

구	살인	강도	강간	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율
강남구	0.384615	1.000000	1.000000	1.000000	1.000000	80.038760	100.000000	100.000000	53.470867	88.130935
강동구	0.307692	0.358974	0.310078	0.488988	0.632184	95.000000	92.857143	100.000000	51.425314	86.996047
강북구	0.538462	0.128205	0.420543	0.340675	0.694153	73.271889	80.000000	85.714286	54.991817	89.344852
강서구	0.692308	0.256410	0.532946	0.544187	0.800600	86.909091	100.000000	100.000000	54.815574	86.392010
관악구	0.461538	0.307692	0.624031	0.586284	0.808346	81.987578	83.333333	100.000000	44.555397	83.678516

- 그리고 나머지 컬럼을 또 가져와서 맞췄다
- 이제 발생건들은 최대값이 1
- 검거는 최대값이 100이다.

인구 현황 데이터 불러오기

In [64]:

```
POP_Seoul = pd.read_csv('../data/Seoul_pop.csv', index_col=1, encoding='utf-8')  
POP_Seoul.head()
```

구	Unnamed: 0	인구수	남성	여성	내국인	남성내국인	여성내국인	외국인	남성외국인	여성외국인	세대당인구	고령자	외국인비율	고령비율	여성비율
종로구	1	164348	79962	84386	154549	75749	78800	9799	4213	5586	2.09	26429	0.059623	0.160811	0.513459
중구	2	135139	66582	68557	126082	62376	63706	9057	4206	4851	2.07	21655	0.067020	0.160242	0.507307
용산구	3	245411	119985	125426	229909	111262	118647	15502	8723	6779	2.12	37238	0.063168	0.151737	0.511085

인구 현황 데이터와 합치기

	Unnamed: 0	인구수	남성	여성	내국인	남성내 국인	여성내 국인	외국 인	남성 외국 인	여성외 국인	세대 당인 구	고령 자	외국인비 율	고령비율	여성비율
구															
종로구	1	164640	80173	84467	155109	76155	78954	9531	4018	5513	2.11	26034	0.057890	0.158127	0.513041
중구	2	134174	66064	68110	125332	62011	63321	8842	4053	4789	2.08	21249	0.065900	0.158369	0.507624
용산구	3	2439	살인	강도	강간	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율			
성동구	4	3129	구												
광진구	5	3724	강남구	0.384615	1.000000	1.000000	1.000000	1.000000	80.038760	100.000000	100.000000	53.470867	88.130935		
			강동구	0.307692	0.358974	0.310078	0.488988	0.632184	95.000000	92.857143	100.000000	51.425314	86.996047		
			강북구	0.538462	0.128205	0.420543	0.340675	0.694153	73.271889	80.000000	85.714286	54.991817	89.344852		
			강서구	0.692308	0.256410	0.532946	0.544187	0.800600	86.909091	100.000000	100.000000	54.815574	86.392010		
			관악구	0.461538	0.307692	0.624031	0.586284	0.808346	81.987578	83.333333	100.000000	44.555397	83.678516		

- 그런데 순서도 다른 데이터를 어떻게 가져와서 합치지???

인구 현황 데이터와 합치기

	Unnamed: 0	인구수	남성	여성	내국인	남성내 국인	여성내 국인	외국 인	남성 외국 인	여성외 국인	세대 당인 구	고령 자	외국인비 율	고령비율	여성비율
구	종로구	164640	80173	84467	155109	76155	78954	9531	4018	5513	2.11	26034	0.057890	0.158127	0.513041
중구	중구	134174	66064	68110	125332	62011	63321	8842	4053	4789	2.08	21249	0.065900	0.158369	0.507624
용산구	용산구	살인	강도	강간	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율				
성동구	성동구	강남구	0.384615	1.000000	1.000000	1.000000	1.000000	80.038760	100.000000	100.000000	53.470867	88.130935			
광진구	광진구	강동구	0.307692	0.358974	0.310078	0.488988	0.632184	95.000000	92.857143	100.000000	51.425314	86.996047	3		
		강북구	0.538462	0.128205	0.420543	0.340675	0.694153	73.271889	80.000000	85.714286	54.991817	89.344852			
		강서구	0.692308	0.256410	0.532946	0.544187	0.800600	86.909091	100.000000	100.000000	54.815574	86.392010	3		
		관악구	0.461538	0.307692	0.624031	0.586284	0.808346	81.987578	83.333333	100.000000	44.555397	83.678516			

- 인덱스의 순서와 관계없이 인덱스의 내용이 같고 **unique**하다면

In [20]:

```
crime_gu_norm[['인구수']] = POP_Seoul[['인구수']]
crime_gu_norm.head()
```

구	살인	강도	강간	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율	인구수
강남구	0.384615	1.000000	1.000000	1.000000	1.000000	80.038760	100.000000	100.000000	53.470867	88.130935	565731
강동구	0.307692	0.358974	0.310078	0.488988	0.632184	95.000000	92.857143	100.000000	51.425314	86.996047	446760
강북구	0.538462	0.128205	0.420543	0.340675	0.694153	73.271889	80.000000	85.714286	54.991817	89.344852	329042
강서구	0.692308	0.256410	0.532946	0.544187	0.800600	86.909091	100.000000	100.000000	54.815574	86.392010	607877
관악구	0.461538	0.307692	0.624031	0.586284	0.808346	81.987578	83.333333	100.000000	44.555397	83.678516	522849

- 그냥 = 기호로 합쳐진다
- if 인덱스가 다른 경우는 key가 될 컬럼을 잡고 합치는 명령을 사용

merge / concat

In [21]:

```
col = ['강간', '강도', '살인', '절도', '폭력']
crime_gu_norm['범죄'] = np.mean(crime_gu_norm[col], axis=1)
crime_gu_norm.head()
```

	살인	강도	강간	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율	인구수	범죄
구												
강남구	0.384615	1.000000	1.000000	1.000000	1.000000	80.038760	100.000000	100.000000	53.470867	86.130035	56573	0.876923
강동구	0.307692	0.358974	0.310078	0.488988	0.632184	95.000000	92.857143	100.000000	51.425514	86.996047	44676	0.419583

- 범죄라는 컬럼으로 범죄 발생의 평균치를 저장
- 컬럼 방향 연산이 아니라 row 방향 연산이라 **axis=1** 옵션을 지정

인구 현황 데이터와 합치기

In [22]:

```
col = ['강간검거율', '강도검거율', '살인검거율', '절도검거율', '폭력검거율']
crime_gu_norm['검거'] = np.mean(crime_gu_norm[col], axis=1)
crime_gu_norm.head()
```

살인	강도	강간	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율	인구수	범죄	검거
84615	1.000000	1.000000	1.000000	1.000000	80.038760	100.000000	100.000000	53.470867	88.130935	565731	0.876923	84.328112
07692	0.358974	0.310078	0.488988	0.632184	95.000000	92.857143	100.000000	51.425314	86.996047	446760	0.419583	85.255701
38462	0.128205	0.420543	0.340675	0.694153	73.271889	80.000000	85.714286	54.991817	89.344852	329042	0.424407	76.664569

- 동일하게 검거 컬럼을 검거율의 평균으로 본다

인구 현황 데이터와 합치기

```
In [23]: crime_gu_norm['범죄'].sort_values(ascending=False).head()
```

구
강남구 0.876923
영등포구 0.745508
송파구 0.612240
강서구 0.565290
구로구 0.560174
Name: 범죄, dtype: float64

```
In [24]: crime_gu_norm['검거'].sort_values(ascending=False).head()
```

구
도봉구 89.205322
성동구 86.220613
동대문구 85.666760
강서구 85.623335
중랑구 85.377882
Name: 검거, dtype: float64

인구 현황 데이터와 합치기

In [24]:

```
crime_gu_norm['검거'].sort_values(ascending=False).head()
```

구

도봉구 89.205322

성동구 86.220613

동대문구 85.666760

강서구 85.623335

중랑구 85.377882

Name: 검거, dtype: float64

In [25]:

```
crime_gu_norm['검거'].sort_values().head()
```

구

종로구 61.930524

동작구 70.593867

영등포구 72.146149

광진구 72.517393

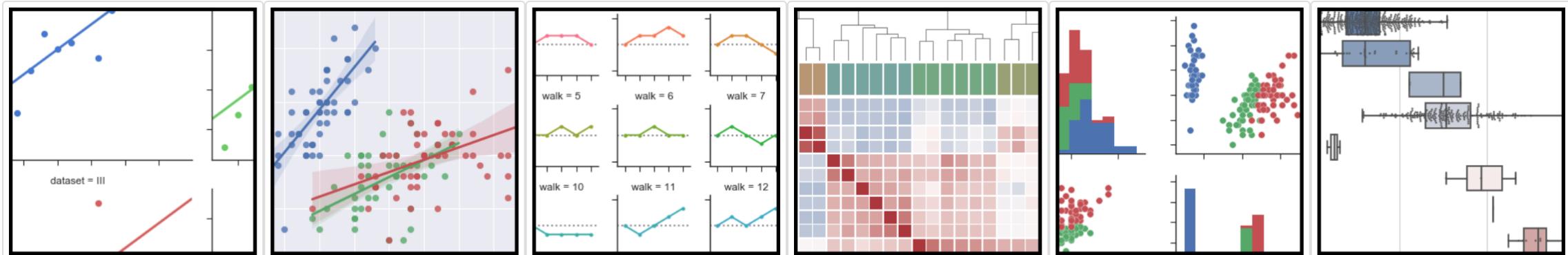
송파구 74.821606

Name: 검거, dtype: float64

-
- 데이터는 모두 정리된 듯 하지만
 - 여전히 시각화에 대한 아쉬움이 남는다

Seaborn – 통계를 위해 태어난 시각화 도구

seaborn: statistical data visualization



Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

For a brief introduction to the ideas behind the package, you can read the [introductory notes](#). More practical information is on the [installation page](#). You may also want to browse the [example gallery](#) to get a sense for what you can do with seaborn and then check out the [tutorial](#) and [API reference](#) to find out how.

To see the code or report a bug, please visit the [github repository](#). General support issues are most at home on [stackoverflow](#), where there is a seaborn tag.

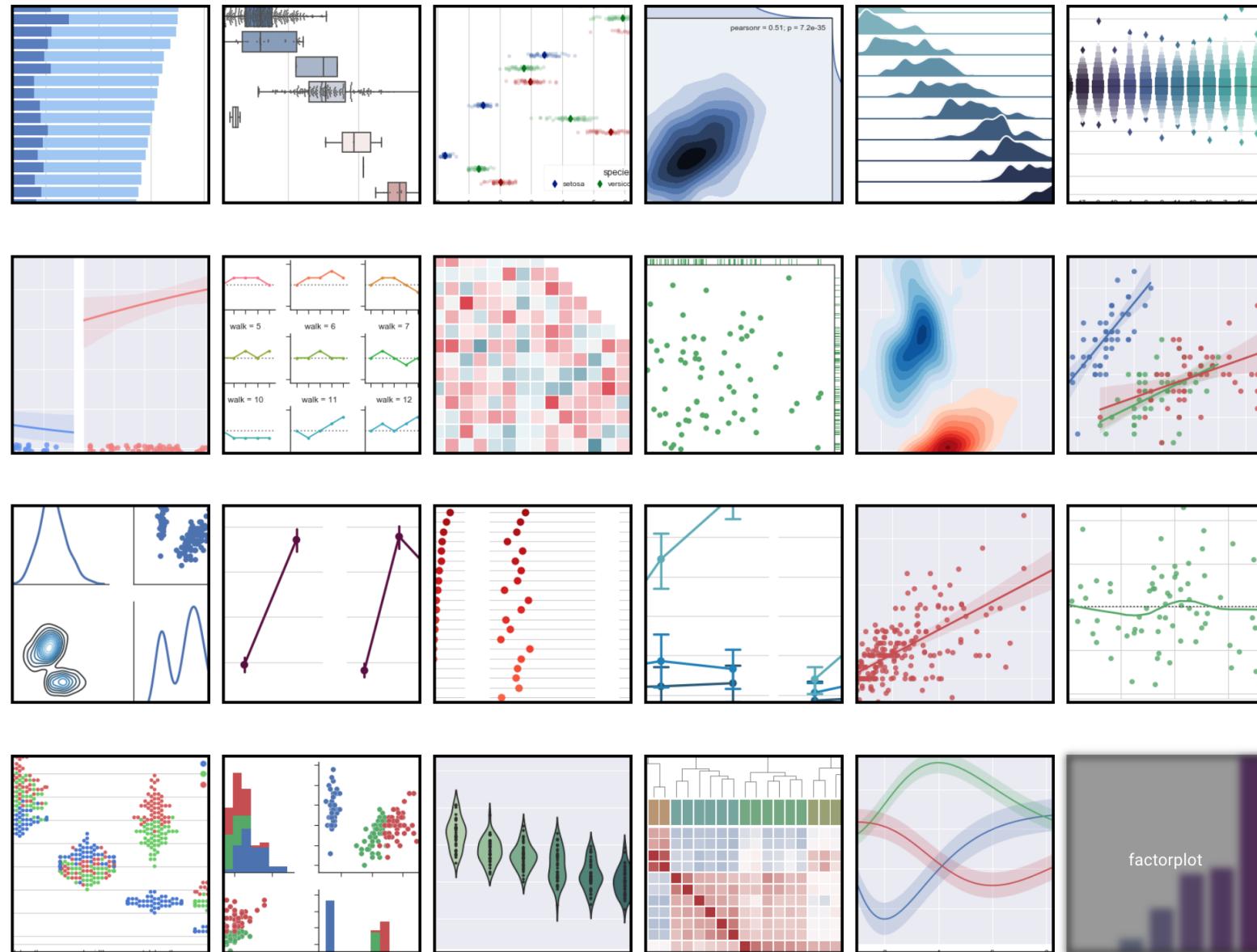
Documentation

- [An introduction to seaborn](#)
- [What's new in the package](#)
- [Installing and getting started](#)
- [Example gallery](#)
- [Seaborn tutorial](#)
- [API reference](#)

Features

- Style functions: [API](#) | [Tutorial](#)
- Color palettes: [API](#) | [Tutorial](#)
- Distribution plots: [API](#) | [Tutorial](#)
- Categorical plots: [API](#) | [Tutorial](#)
- Regression plots: [API](#) | [Tutorial](#)
- Axis grid objects: [API](#) | [Tutorial](#)

Seaborn - Basic



In [26]:

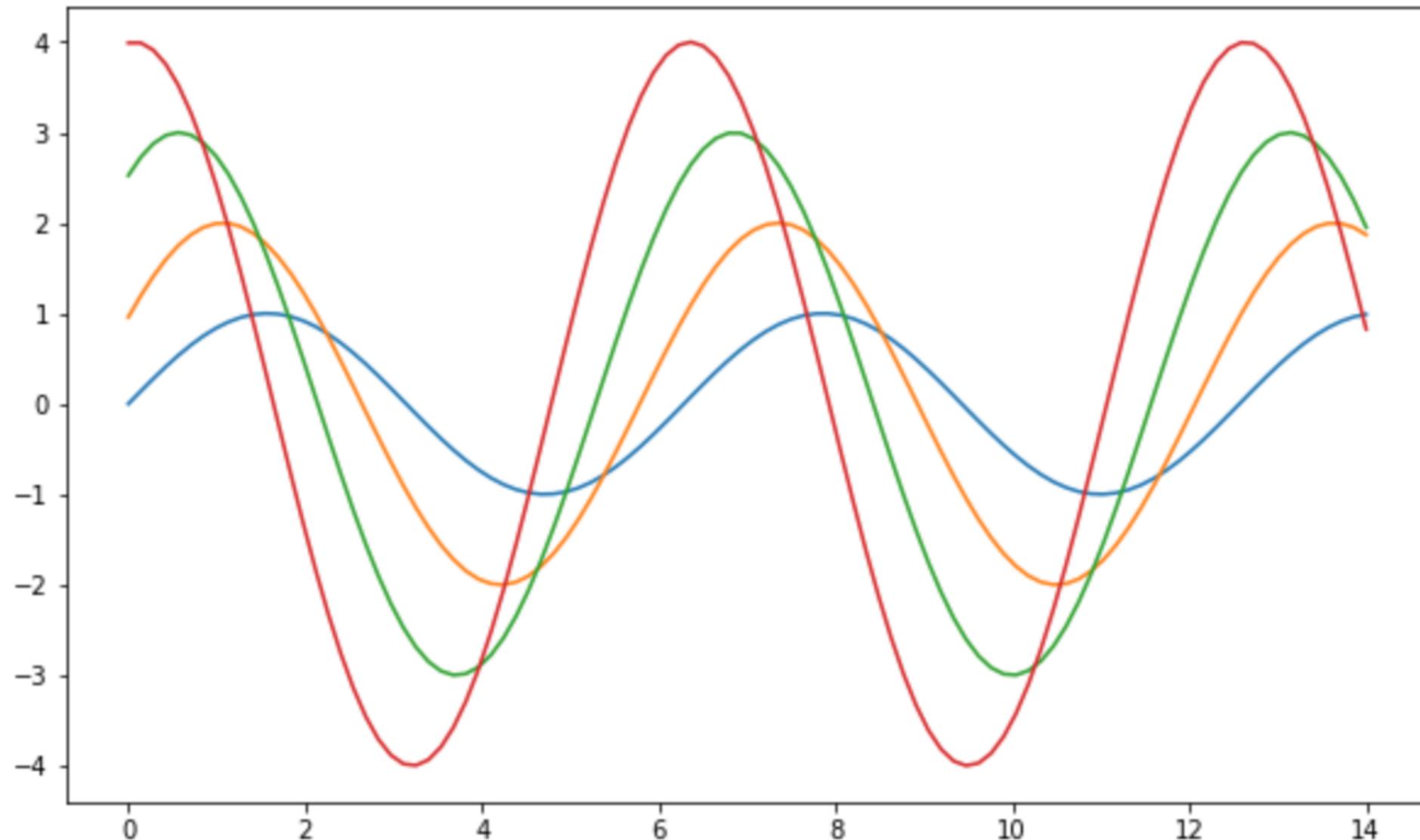
```
import matplotlib.pyplot as plt  
import seaborn as sns  
%matplotlib inline
```

- **seaborn**은 **matplotlib**을 백엔드로 동작한다
- 그저 **import**만 시켜도 그래프 스타일이 **seaborn**으로 변경된다

```
In [27]:  
x = np.linspace(0, 14, 100)  
y1 = np.sin(x)  
y2 = 2*np.sin(x+0.5)  
y3 = 3*np.sin(x+1.0)  
y4 = 4*np.sin(x+1.5)
```

```
In [28]:  
plt.figure(figsize=(10,6))  
plt.plot(x,y1, x,y2, x,y3, x,y4)  
plt.show()
```

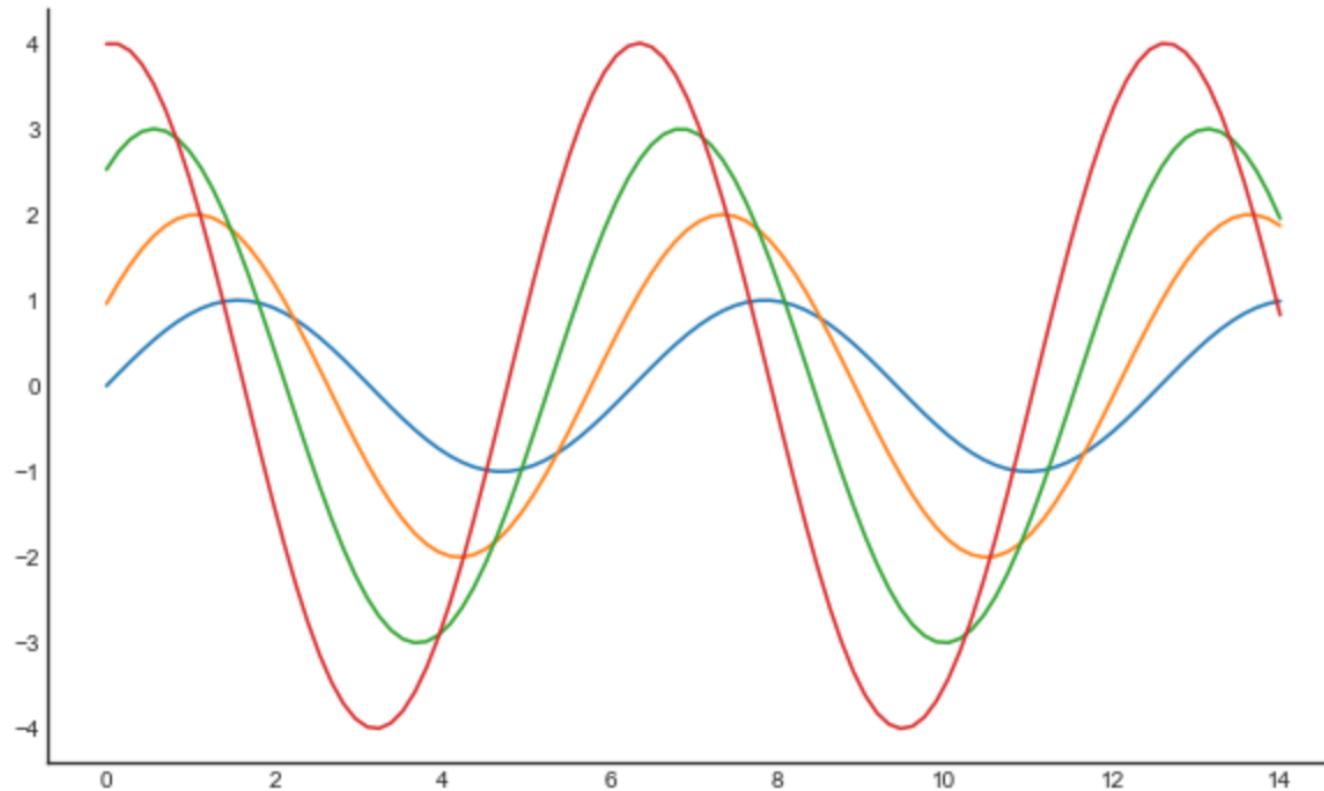
- 테스트용 그래프,
- 다수의 **plt.plot**을 두어도 되고,
- 하나의 **plt.plot**에 **(x,y1, x,y2, x,y3 ...)** 와 같은 스타일이어도 된다



Seaborn - Basic

In [29]:

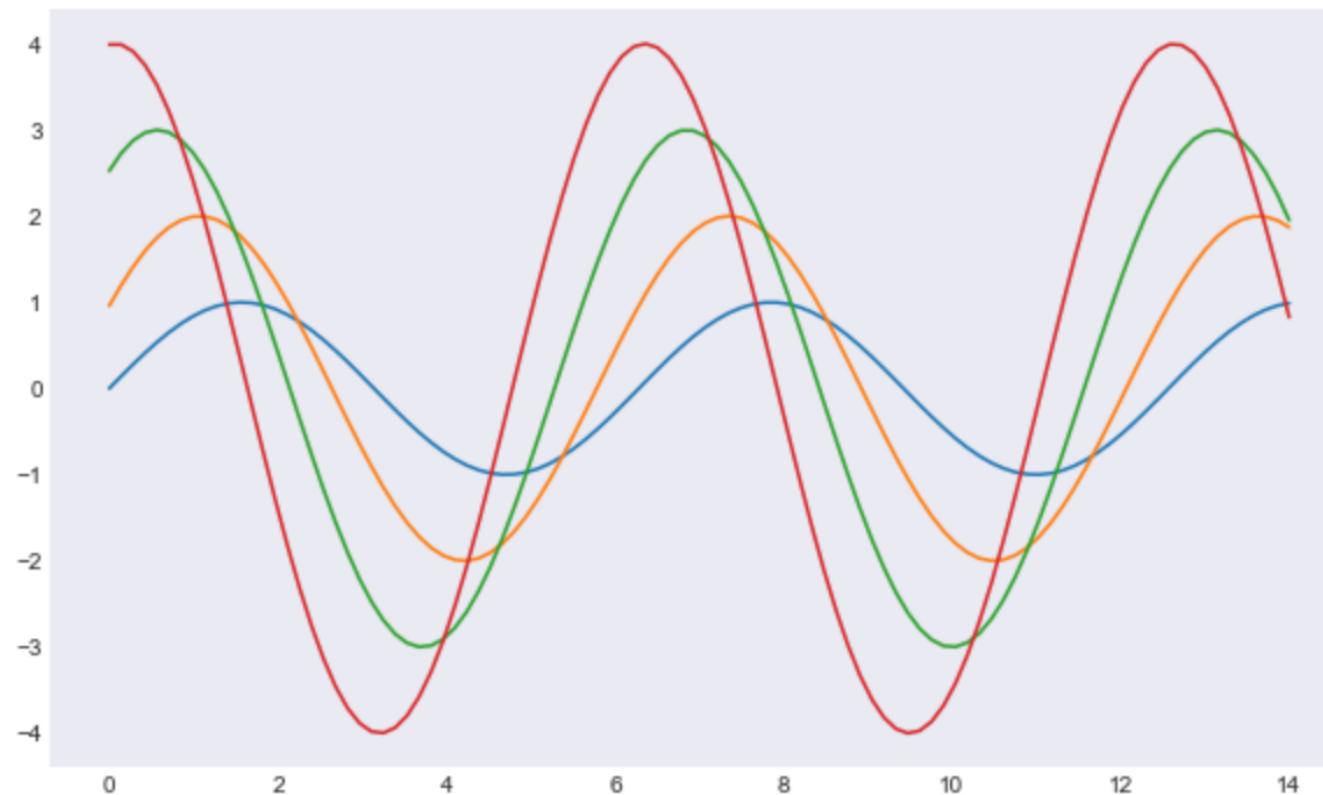
```
sns.set_style("white")
plt.figure(figsize=(10,6))
plt.plot(x,y1, x,y2, x,y3, x,y4)
sns.despine(); plt.show()
```



Seaborn - Basic

In [30]:

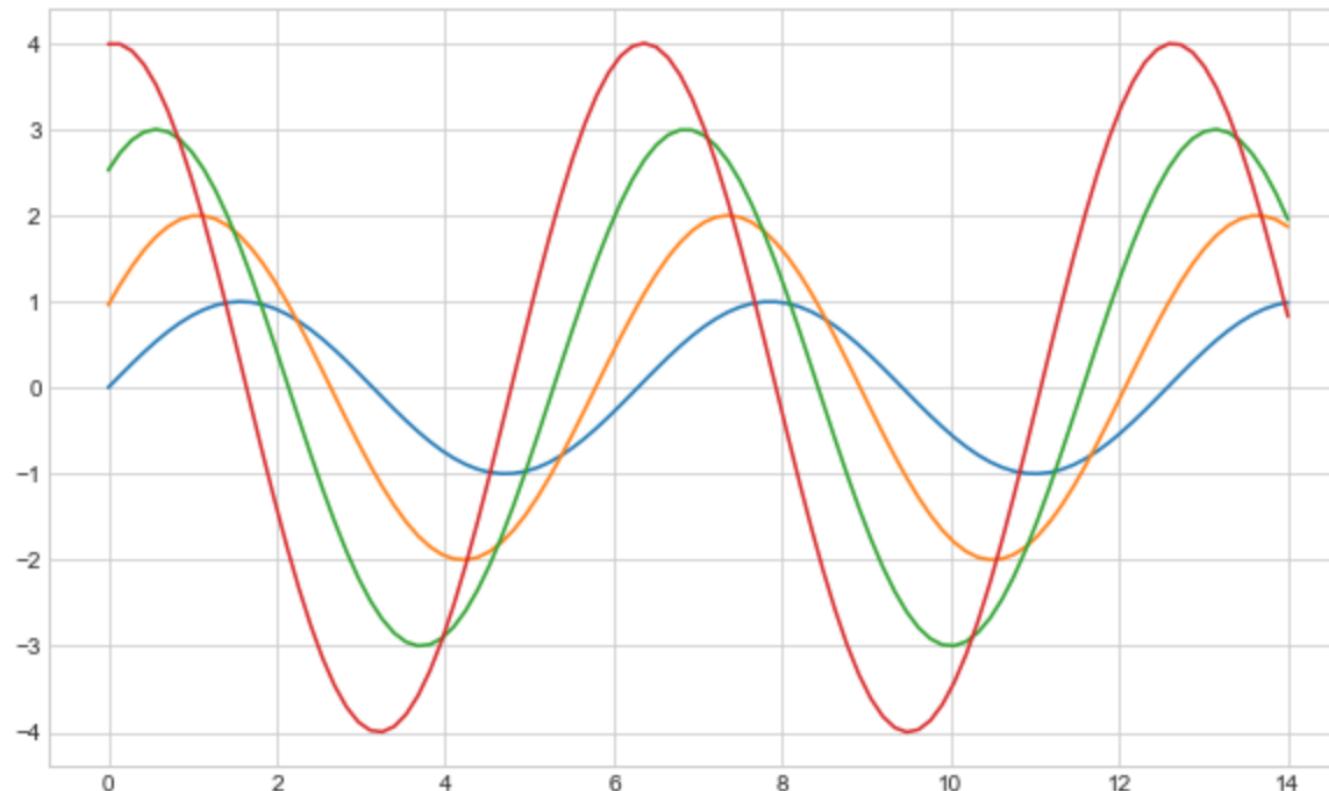
```
sns.set_style( "dark" )  
plt.figure(figsize=( 10 , 6 ))  
plt.plot(x,y1, x,y2, x,y3, x,y4)  
plt.show( )
```



Seaborn - Basic

In [31]:

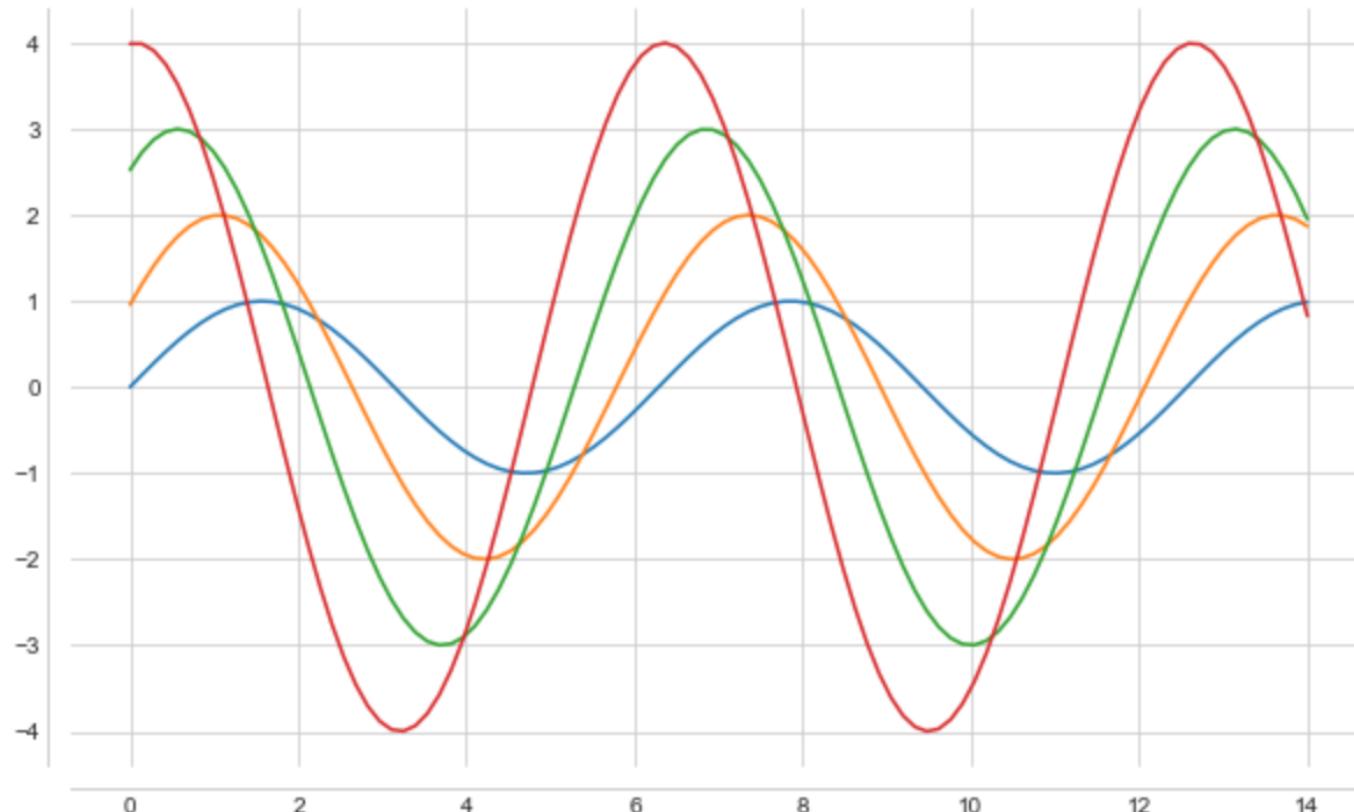
```
sns.set_style( "whitegrid" )  
plt.figure(figsize=( 10,6 ))  
plt.plot(x,y1, x,y2, x,y3, x,y4)  
plt.show( )
```



Seaborn - Basic

In [32]:

```
plt.figure(figsize=(10,6))  
plt.plot(x,y1, x,y2, x,y3, x,y4)  
sns.despine(offset=10)  
plt.show()
```



Seaborn – Data Visualization

In [33]:

```
tips = sns.load_dataset("tips")
tips.head(5)
```

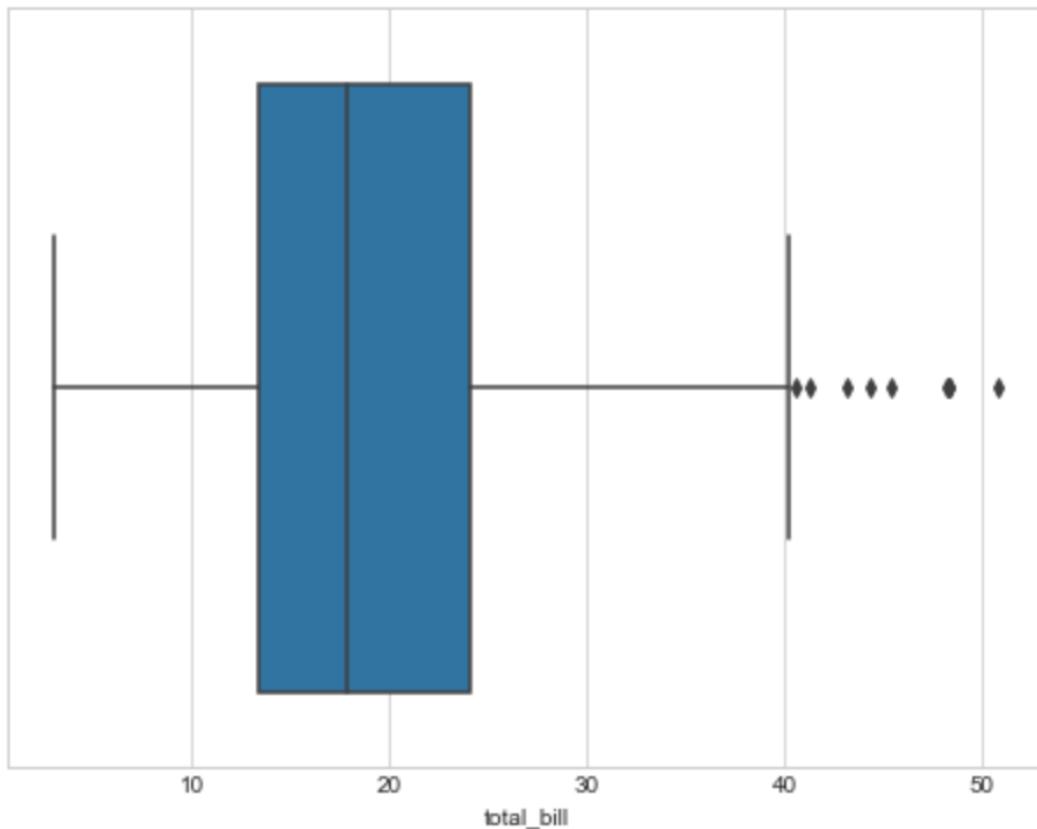
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

- Seaborn은 연습을 위한 데이터를 가지고 있다
- 그 중에 tips ~~

Seaborn – Data Visualization

In [34]:

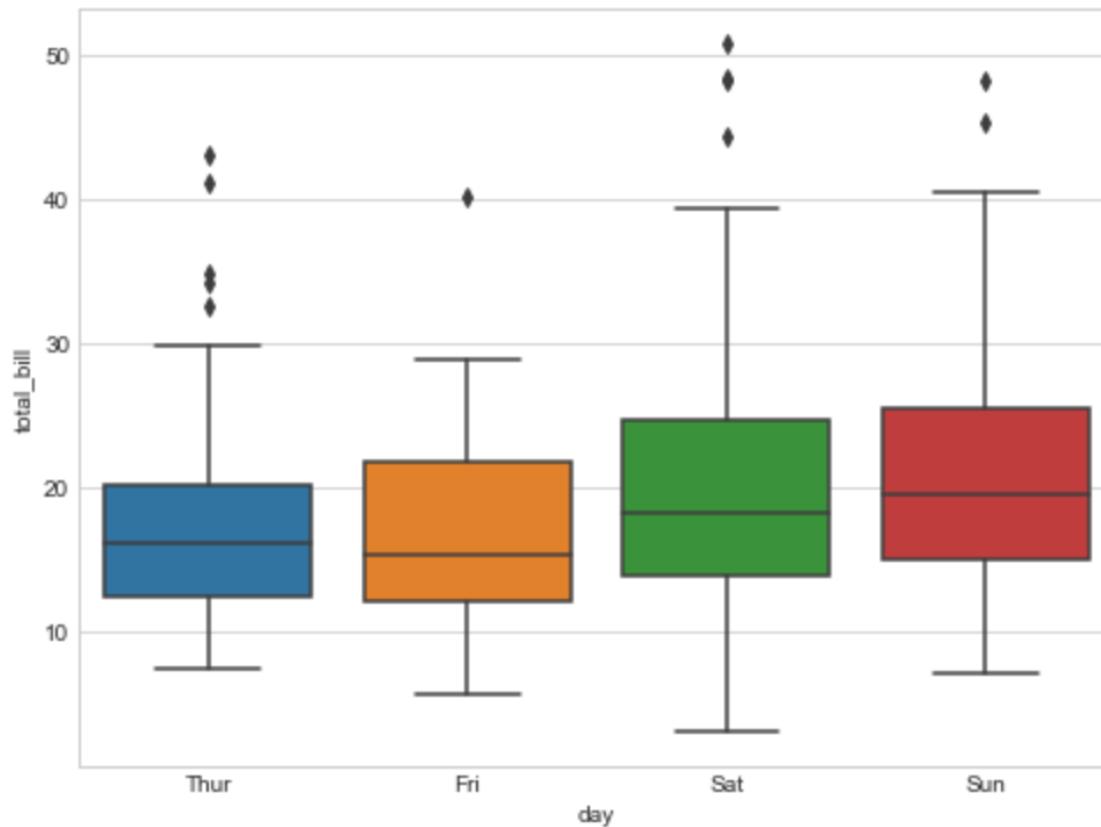
```
plt.figure(figsize=(8,6))
sns.boxplot(x=tips["total_bill"])
plt.show()
```



Seaborn – Data Visualization

In [35]:

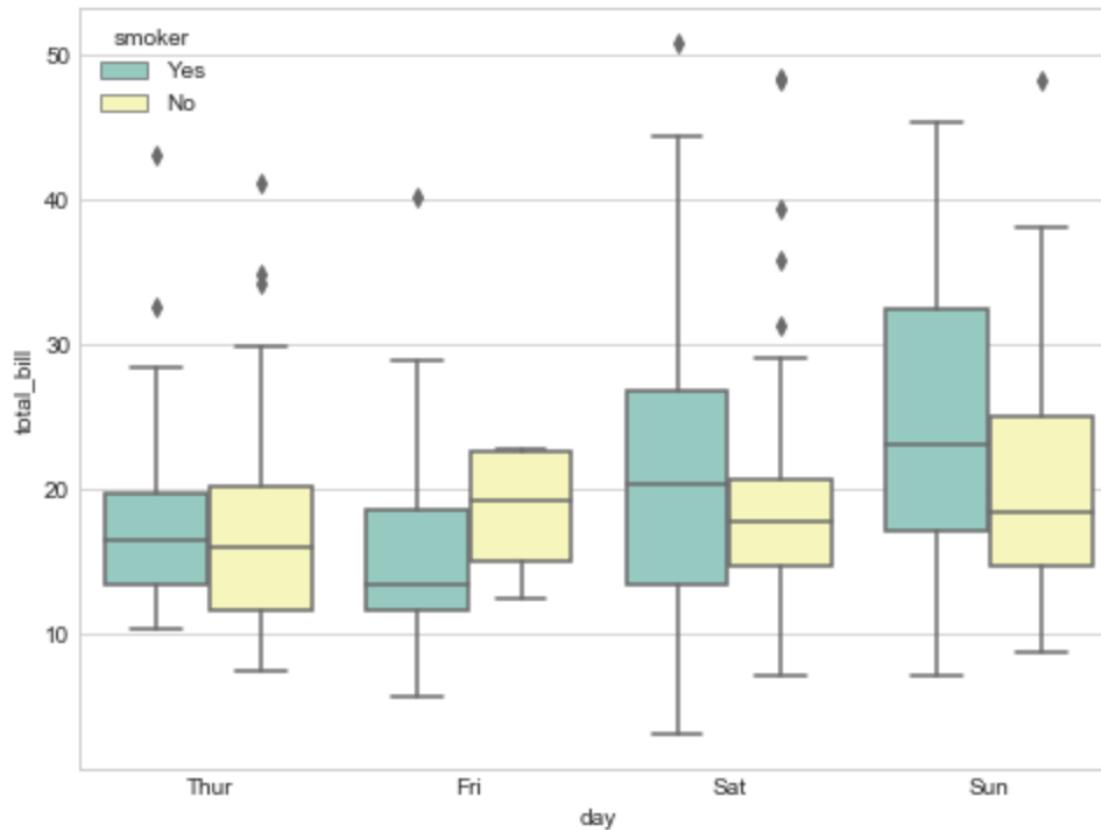
```
plt.figure(figsize=(8,6))  
sns.boxplot(x="day", y="total_bill", data=tips)  
plt.show()
```



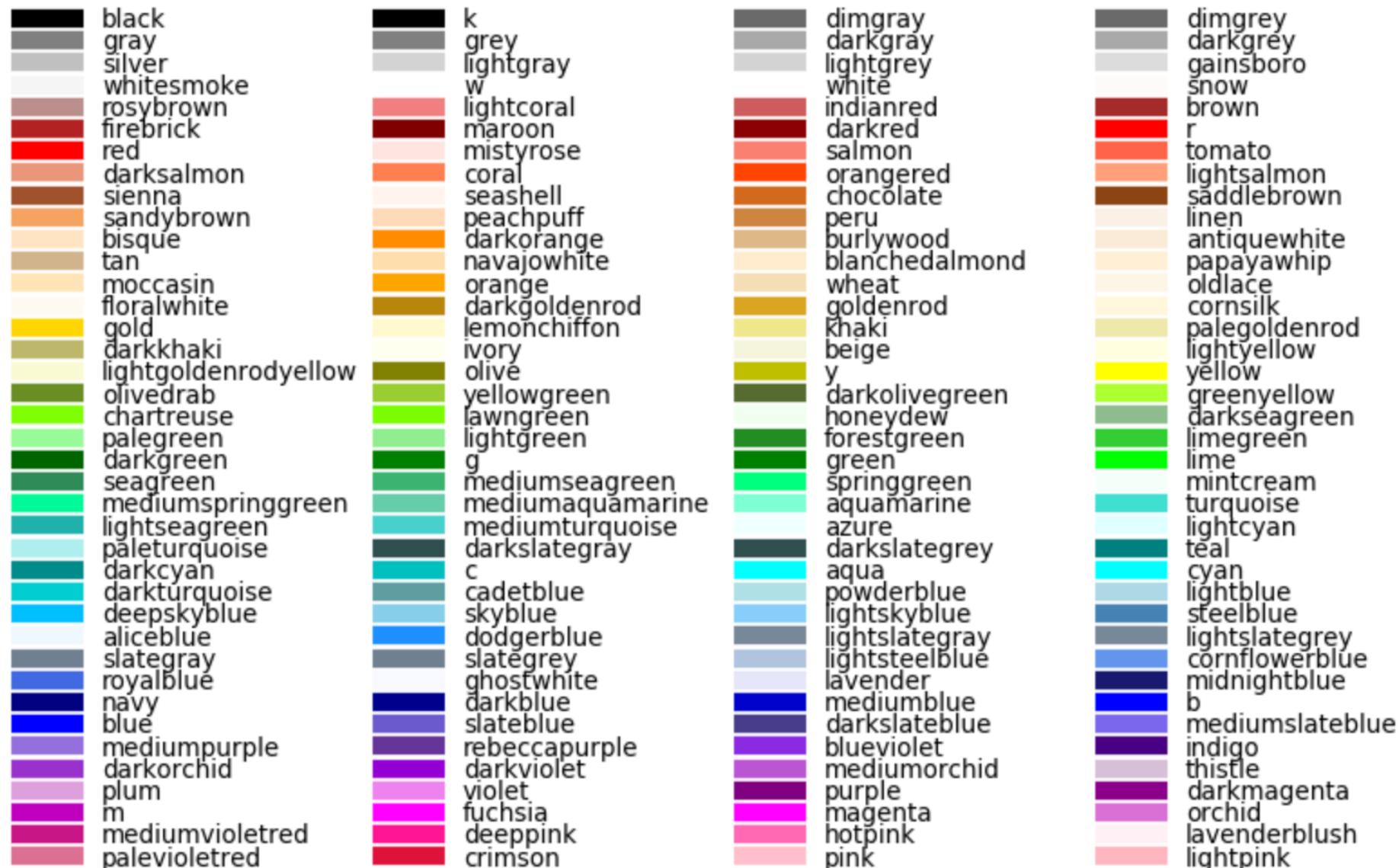
Seaborn – Data Visualization

In [36]:

```
plt.figure(figsize=(8,6))
sns.boxplot(x="day", y="total_bill", hue="smoker", data=tips, palette="Set3")
plt.show()
```



Seaborn – Data Visualization

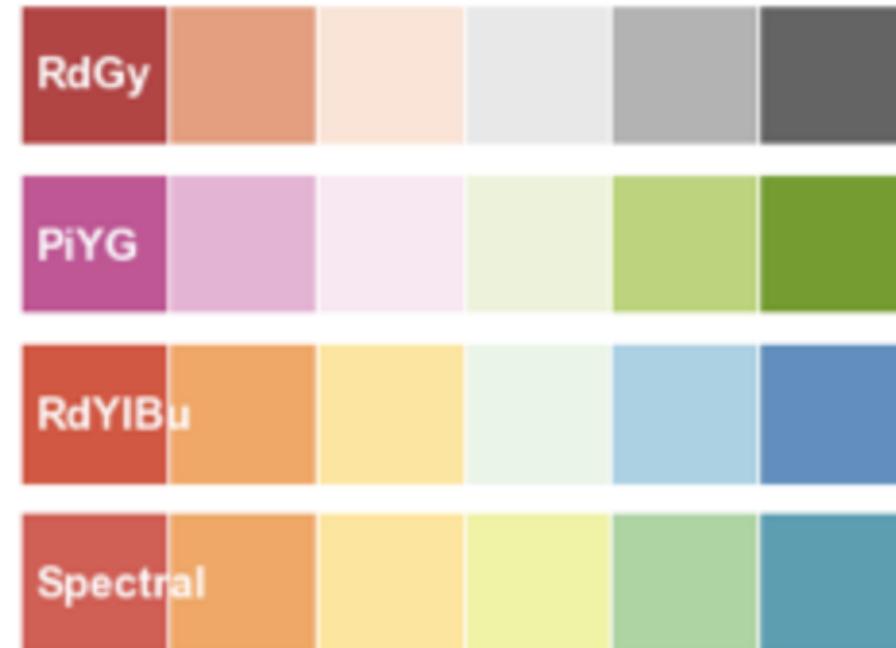
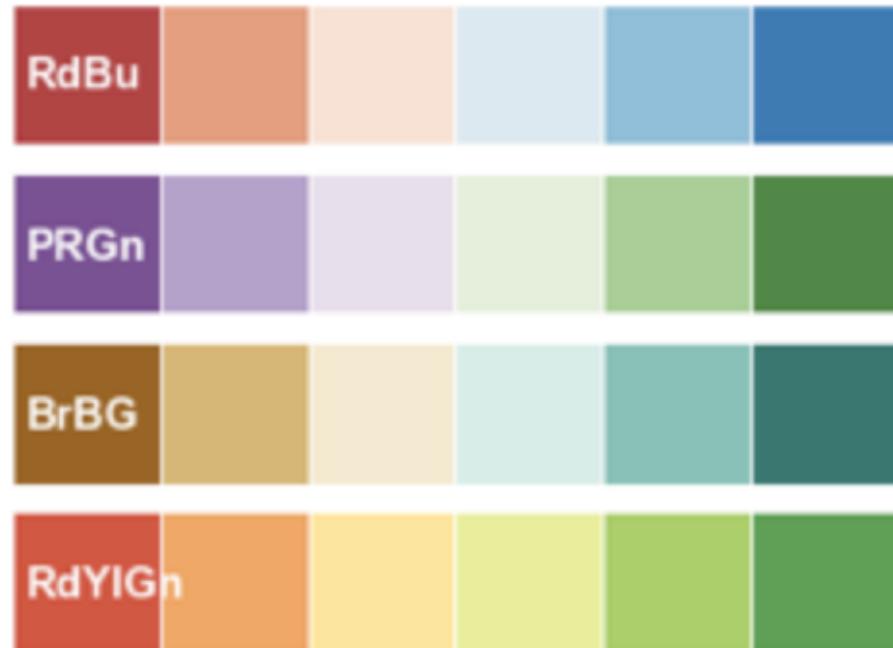


Seaborn – Data Visualization



Seaborn – Data Visualization

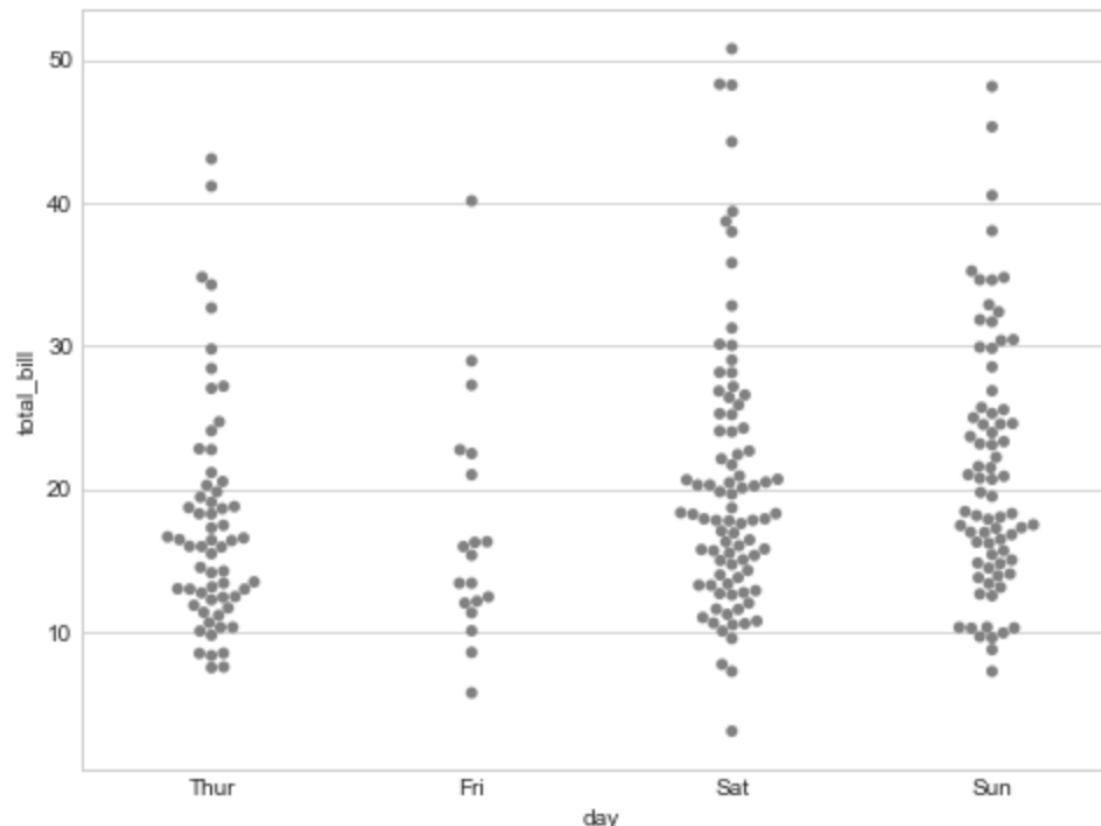




Seaborn – Data Visualization

In [37]:

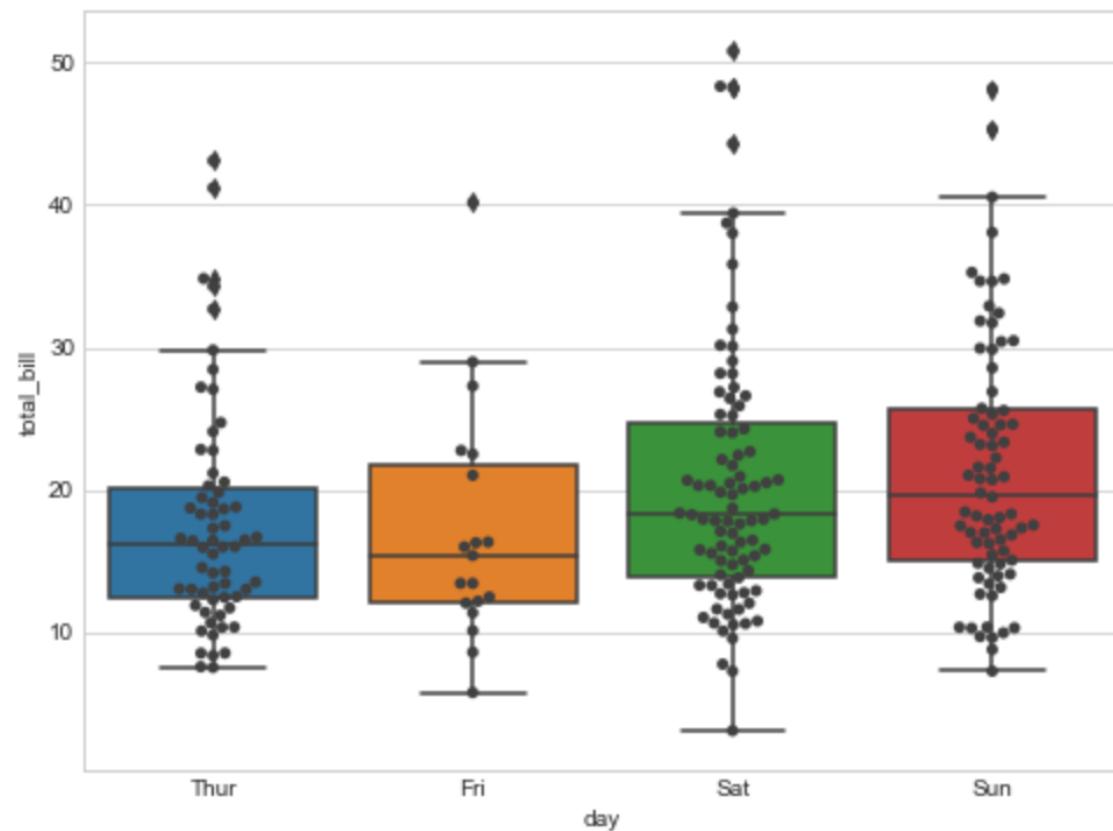
```
plt.figure(figsize=(8,6))  
sns.swarmplot(x="day", y="total_bill", data=tips, color=".5")  
plt.show()
```



Seaborn – Data Visualization

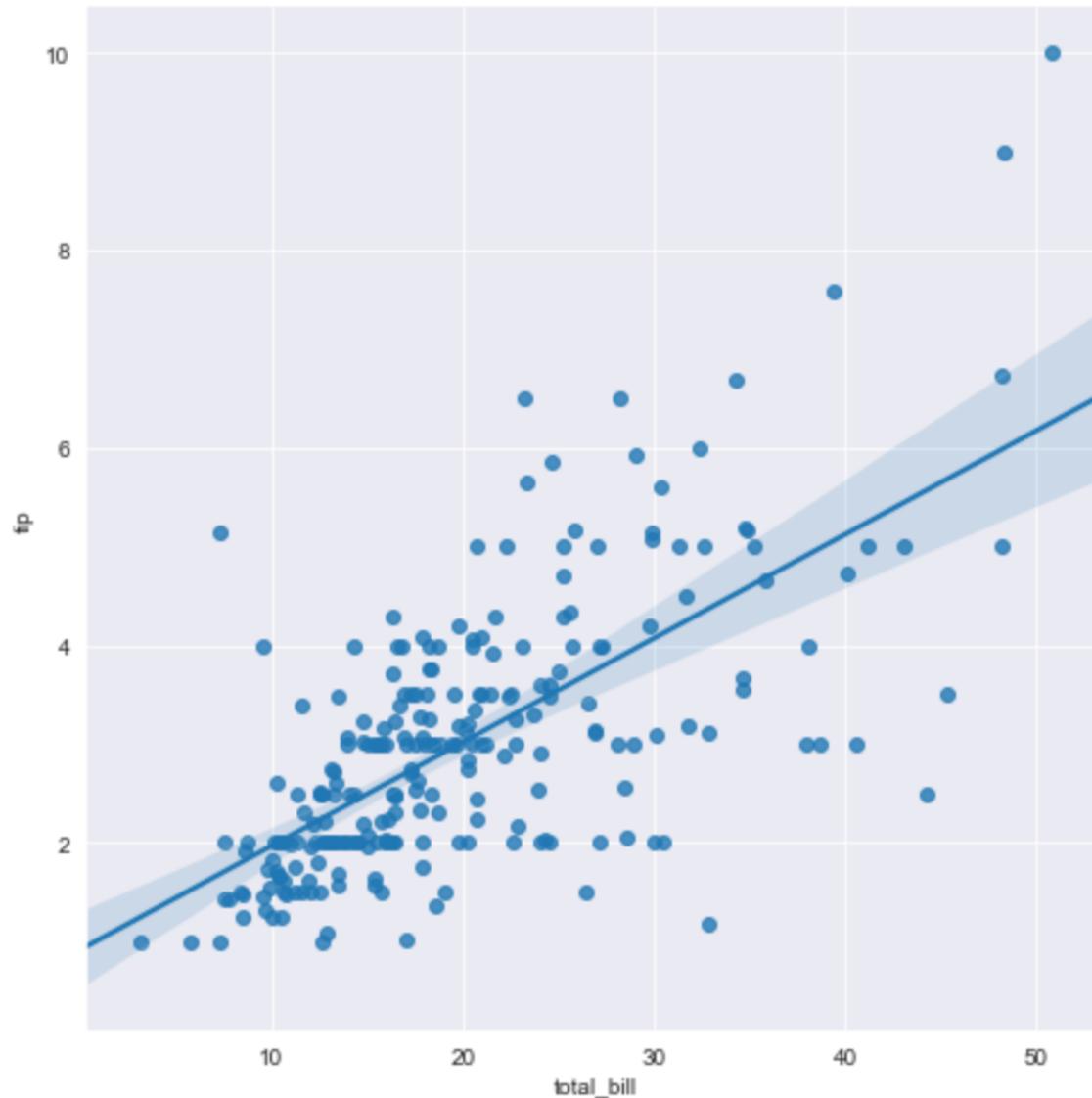
In [38]:

```
plt.figure(figsize=(8,6))
sns.boxplot(x="day", y="total_bill", data=tips)
sns.swarmplot(x="day", y="total_bill", data=tips, color=".25")
plt.show()
```



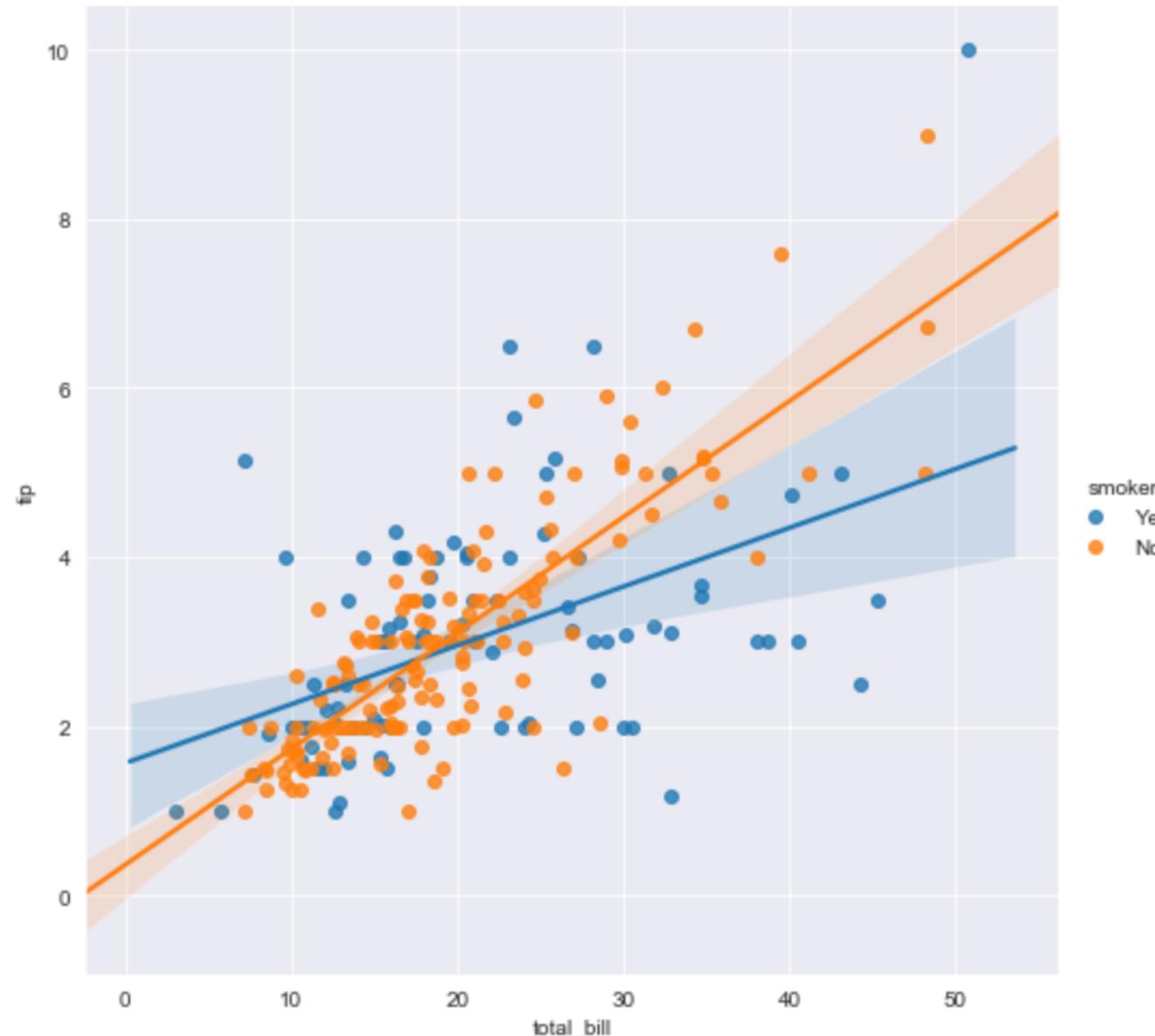
In [41]:

```
sns.set_style("darkgrid")
sns.lmplot(x="total_bill", y="tip", data=tips, height=7)
plt.show()
```



In [42]:

```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips, height=7)  
plt.show()
```



```
In [41]: flights = sns.load_dataset("flights")
flights.head(5)
```

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121

- 또 다른 데이터가 있다 **flights** ~

In [42]:

```
flights = flights.pivot("month", "year", "passengers")
flights.head(5)
```

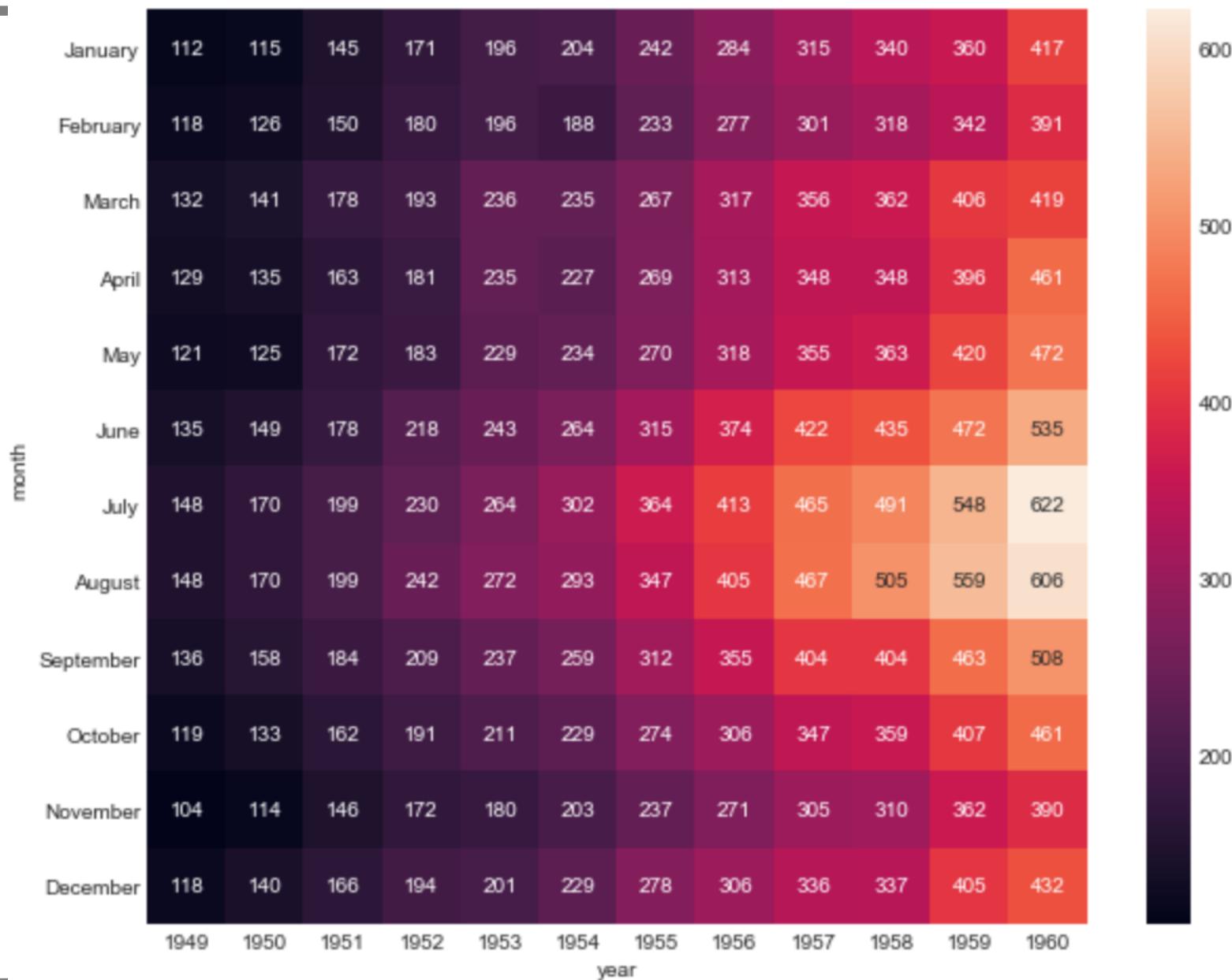
year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
January	112	115	145	171	196	204	242	284	315	340	360	417
February	118	126	150	180	196	188	233	277	301	318	342	391
March	132	141	178	193	236	235	267	317	356	362	406	419
April	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472

- 간단하게 **pivot** 옵션으로 **pivot_table**을 구현할 수도 있다.

In [43]:

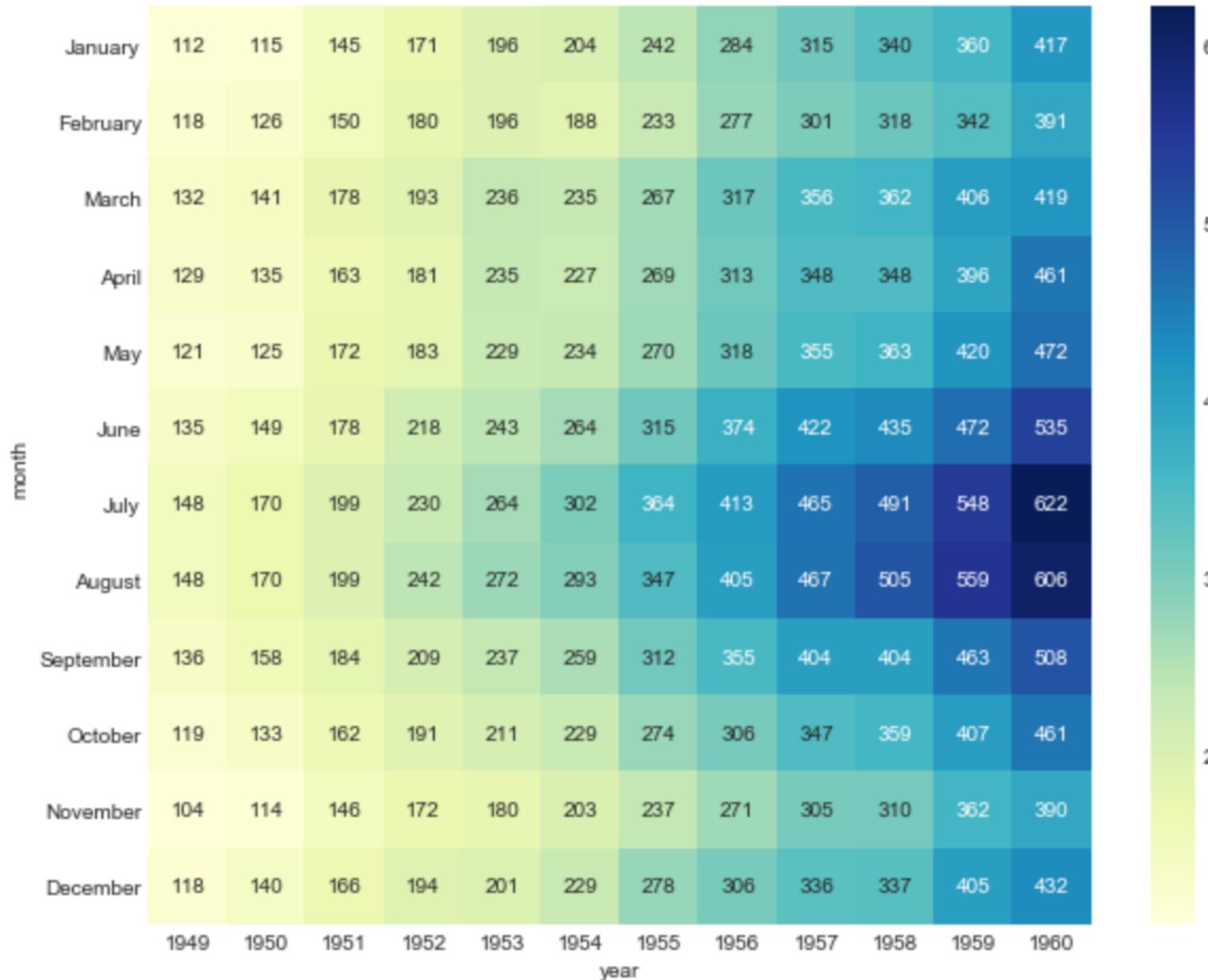
```
plt.figure(figsize=(10,8))
sns.heatmap(flights, annot=True, fmt="d")
plt.show()
```

Seaborn – Data Visualization



In [44]:

```
plt.figure(figsize=(10,8))
sns.heatmap(flights, annot=True, fmt="d", cmap="YlGnBu")
plt.show()
```



- **heatmap**이 적용 가능하다면 시간의 흐름 혹은
- 정렬된 데이터에 대해 경향성을 보여주는데 유리하다



Seaborn – Data Visualization

In [45]:

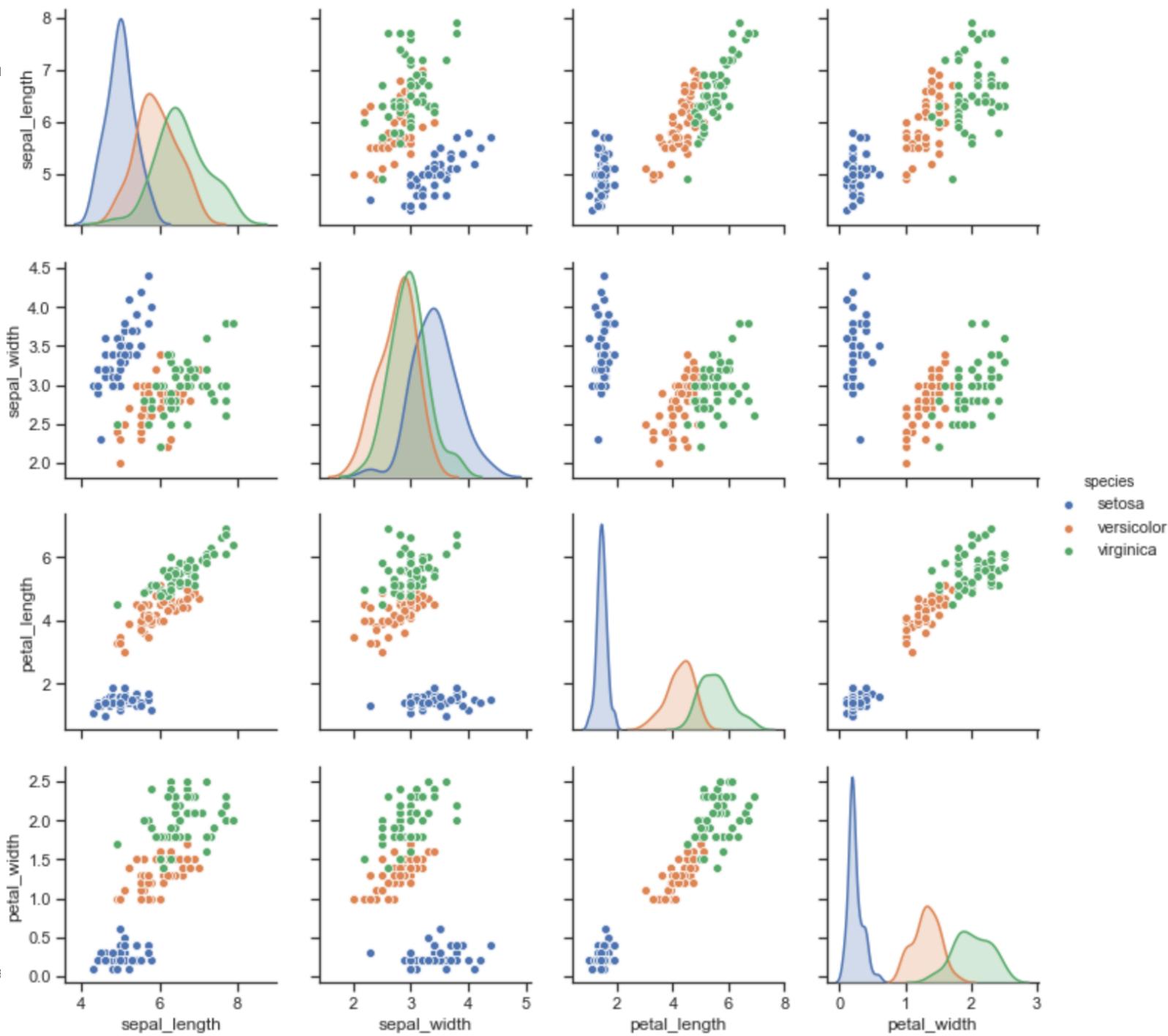
```
sns.set(style="ticks")
iris = sns.load_dataset("iris")
iris.head(10)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa

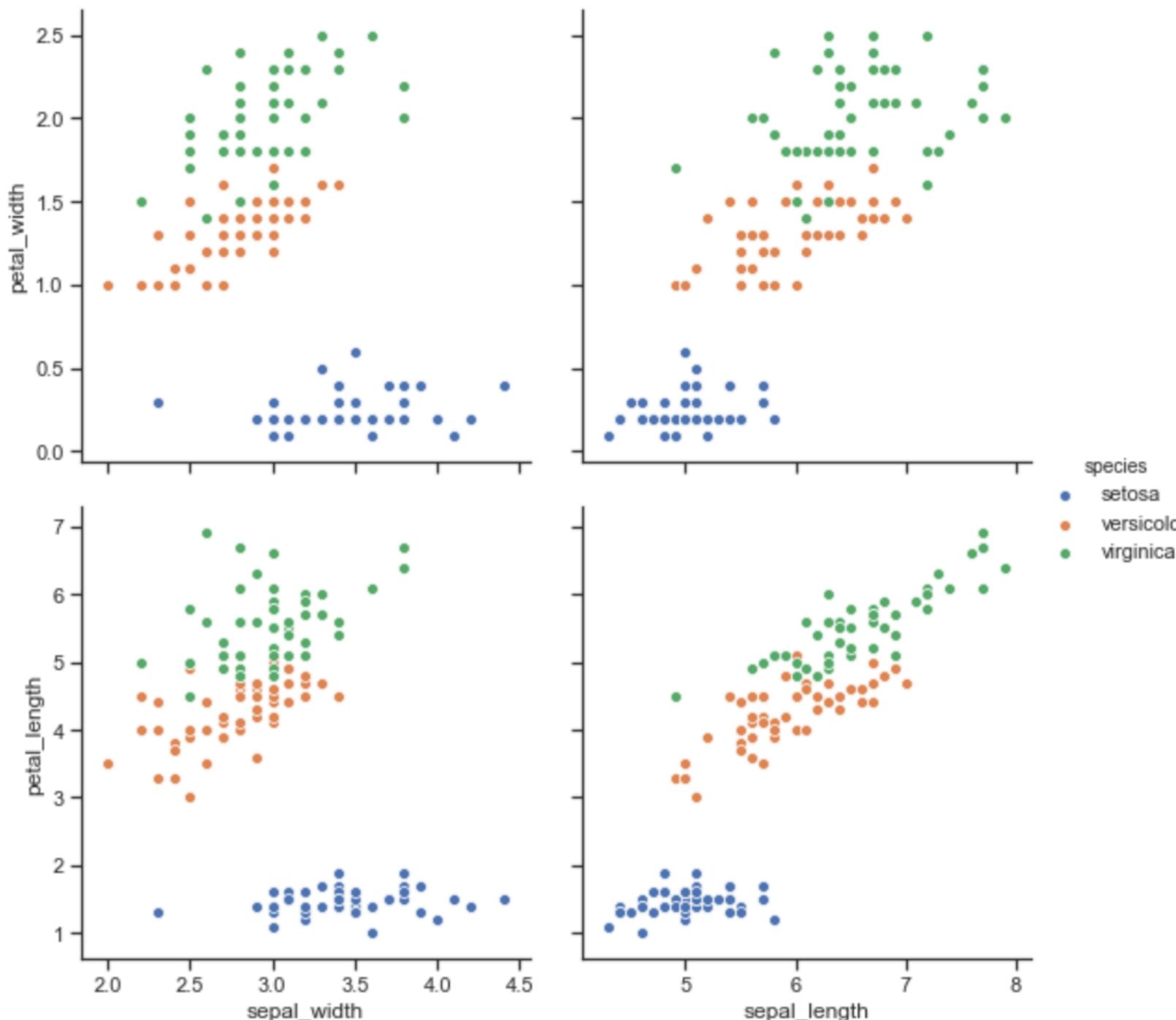
In [46]:

```
sns.pairplot(iris, hue="species")  
plt.show()
```

- **pairplot**: 컬럼간의 비교를 한 번에 할 수 있다



```
In [50]: sns.pairplot(iris, x_vars=["sepal_width", "sepal_length"],  
                  y_vars=["petal_width", "petal_length"],  
                  hue="species",  
                  height=4)  
plt.show()
```



우리 서울 범죄 데이터도 멋지게 도전하자

In [48]:

```
from matplotlib import font_manager, rc
plt.rcParams['axes.unicode_minus'] = False

f_path = "c:/Windows/Fonts/malgun.ttf"
# f_path = "/Users/pinkwink/Library/Fonts/D2Coding-Ver1.3-20171129.ttf"
# f_path = "/Library/Fonts/AppleGothic.ttf"
font_name = font_manager.FontProperties(fname=f_path).get_name()
rc('font', family=font_name)
```

- 한글 문제를 일단 해결하고...

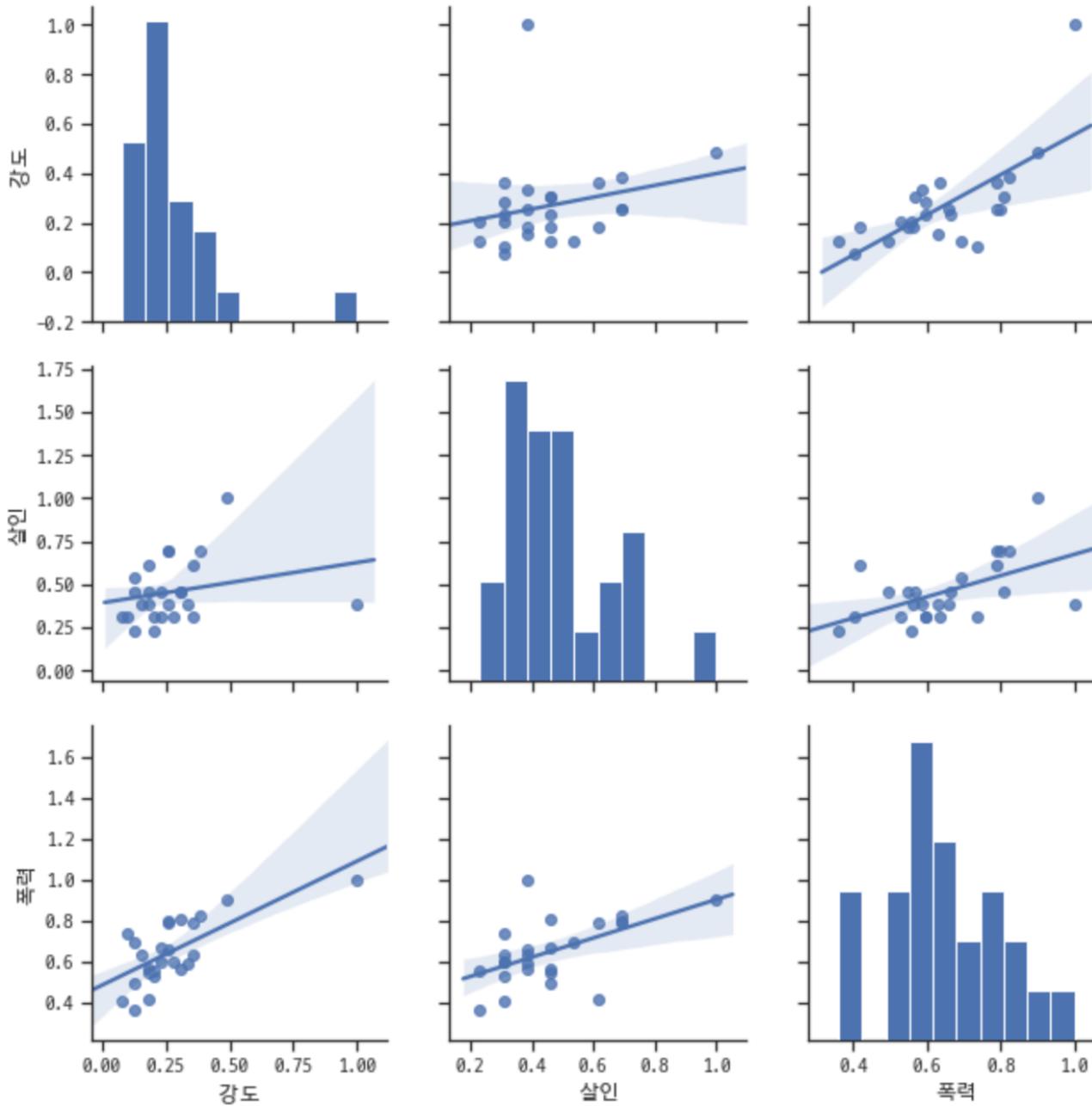
In [49]:

```
crime_gu_norm.head()
```

구	살인	강도	강간	절도	폭력	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율	인구수	범죄
강남구	0.384615	1.000000	1.000000	1.000000	1.000000	80.038760	100.000000	100.000000	53.470867	88.130935	565731	0.876923
강동구	0.307692	0.358974	0.310078	0.488988	0.632184	95.000000	92.857143	100.000000	51.425314	86.996047	446760	0.419583
강북구	0.538462	0.128205	0.420543	0.340675	0.694153	73.271889	80.000000	85.714286	54.991817	89.344852	329042	0.424407
강서구	0.692308	0.256410	0.532946	0.544187	0.800600	86.909091	100.000000	100.000000	54.815574	86.392010	607877	0.565290

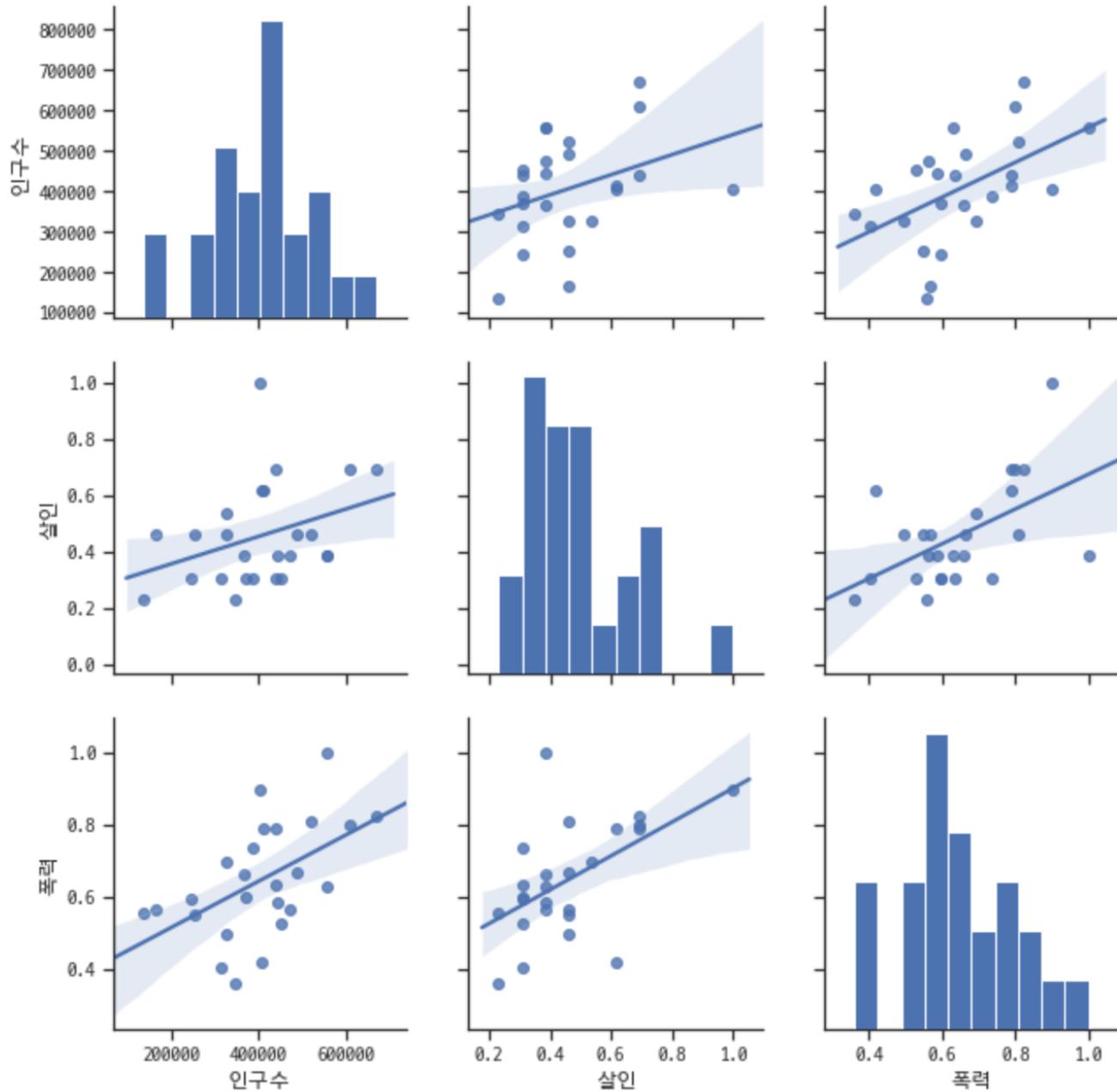
In [51]:

```
sns.pairplot(crime_gu_norm, vars=["강도", "살인", "폭력"], kind='reg', height=3);
```



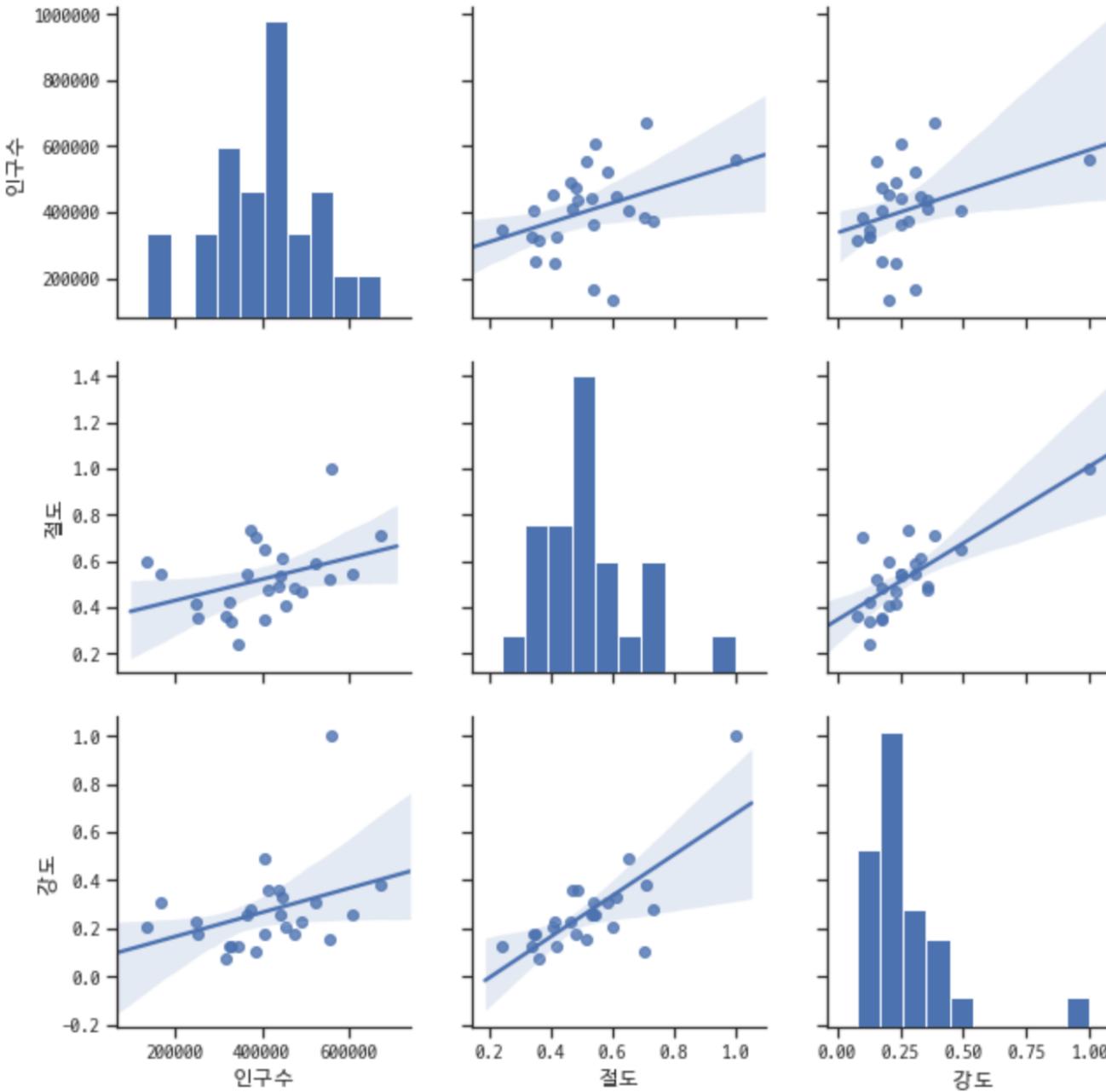
In [52]:

```
sns.pairplot(crime_gu_norm, vars=["인구수", "살인", "폭력"], kind='reg', height=3);
```



In [54]:

```
sns.pairplot(crime_gu_norm, vars=[ "인구수", "절도", "강도"], kind='reg', height=3);
```



In [53]:

```
target_col = ['강간검거율', '강도검거율', '살인검거율', '절도검거율', '폭력검거율', '검거']

crime_gu_norm_sort = crime_gu_norm.sort_values(by='검거', ascending=False)

plt.figure(figsize = (10,10))
sns.heatmap(crime_gu_norm_sort[target_col],
            annot=True, fmt='f', linewidths=.5, cmap='RdPu')
plt.title('범죄 검거 비율 (정규화된 검거의 합으로 정렬)')
plt.show()
```

- **heatmap**으로 검거 상황을 그려보자
- 검거의 종합점수로 정렬하고

범죄 검거 비율 (정규화된 검거의 합으로 정렬)

	강간검거율	강도검거율	살인검거율	절도검거율	폭력검거율	검거
도봉구	98.373984	100.000000	100.000000	56.812933	90.839695	89.205322
성동구	75.000000	100.000000	100.000000	69.135802	86.967264	86.220613
동대문구	83.157895	100.000000	100.000000	55.206186	89.969720	85.666760
강서구	86.909091	100.000000	100.000000	54.815574	86.392010	85.623335
중랑구	91.463415	100.000000	87.500000	62.211709	85.714286	85.377882
강동구	95.000000	92.857143	100.000000	51.425314	86.996047	85.255701
금천구	81.714286	100.000000	100.000000	51.740506	88.736890	84.438336
강남구	80.038760	100.000000	100.000000	53.470867	88.130935	84.328112
노원구	89.308176	100.000000	100.000000	39.849219	84.419714	82.715422
양천구	82.442748	100.000000	100.000000	43.920884	85.244444	82.321615
용산구	85.258964	100.000000	100.000000	40.228341	84.228188	81.943099
은평구	91.025641	77.777778	100.000000	53.421369	86.636637	81.772285
마포구	80.200501	100.000000	100.000000	37.198259	85.062947	80.492341
서대문구	84.000000	80.000000	100.000000	50.033267	83.198381	79.446329
구로구	66.300366	100.000000	100.000000	45.078534	84.702908	79.216362
중구	74.747475	87.500000	100.000000	42.511628	89.707865	78.893394
관악구	81.987578	83.333333	100.000000	44.555397	83.678516	78.710965
성북구	75.974026	100.000000	75.000000	49.319728	86.290323	77.316815
강북구	73.271889	80.000000	85.714286	54.991817	89.344852	76.664569
서초구	63.317757	76.923077	100.000000	50.204082	86.783576	75.445698
송파구	78.632479	80.000000	88.888889	41.211168	85.375494	74.821606
광진구	83.870968	54.545455	100.000000	40.098634	84.071906	72.517393
영등포구	63.202247	73.684211	100.000000	40.153780	83.690509	72.146149
동작구	45.846154	100.000000	75.000000	45.187602	86.935581	70.593867
종로구	74.369748	75.000000	33.333333	39.587629	87.361909	61.930524



In [54]:

```
target_col = target_col = ['강간', '강도', '살인', '절도', '폭력', '범죄']

crime_gu_norm_sort = crime_gu_norm.sort_values(by='범죄', ascending=False)

plt.figure(figsize = (10,10))
sns.heatmap(crime_gu_norm_sort[target_col],
            annot=True, fmt='f', linewidths=.5, cmap='RdPu')
plt.title('범죄비율 (정규화된 발생 건수로 정렬)')
plt.show()
```

- 범죄 발생으로도 정렬해서 그려보자

범죄비율 (정규화된 발생 건수로 정렬)

	강간	강도	살인	절도	폭력	범죄
강남구	1.000000	1.000000	0.384615	1.000000	1.000000	0.876923
영등포구	0.689922	0.487179	1.000000	0.652635	0.897801	0.745508
송파구	0.453488	0.384615	0.692308	0.708949	0.821839	0.612240
강서구	0.532946	0.256410	0.692308	0.544187	0.800600	0.565290
구로구	0.529070	0.256410	0.692308	0.532478	0.790605	0.560174
관악구	0.624031	0.307692	0.461538	0.586284	0.808346	0.557578
서초구	0.829457	0.333333	0.384615	0.614720	0.584208	0.549267
마포구	0.773256	0.102564	0.307692	0.704488	0.734383	0.524477
중랑구	0.317829	0.358974	0.615385	0.471425	0.790605	0.510844
광진구	0.540698	0.282051	0.307692	0.734876	0.597701	0.492604
종로구	0.461240	0.307692	0.461538	0.540842	0.565467	0.467356
동대문구	0.368217	0.256410	0.384615	0.540842	0.660170	0.442051
마포구	0.629845	0.179487	0.615385	0.341790	0.415042	0.436310
은평구	0.302326	0.230769	0.461538	0.464455	0.665667	0.424951
강북구	0.420543	0.128205	0.538462	0.340675	0.694153	0.424407
강동구	0.310078	0.358974	0.307692	0.488988	0.632184	0.419583
용산구	0.486434	0.230769	0.307692	0.415110	0.595702	0.407142
노원구	0.308140	0.153846	0.384615	0.517703	0.628686	0.398598
중구	0.383721	0.205128	0.230769	0.599387	0.555972	0.394995
금천구	0.339147	0.179487	0.461538	0.352384	0.547976	0.376107
양천구	0.253876	0.179487	0.384615	0.479231	0.562219	0.371886
서대문구	0.339147	0.128205	0.461538	0.419013	0.493753	0.368331
성북구	0.298450	0.205128	0.307692	0.409813	0.526737	0.349564
성동구	0.201550	0.076923	0.307692	0.361305	0.404548	0.270404
도봉구	0.238372	0.128205	0.230769	0.241427	0.360070	0.239769



- 비록 **2016**년 데이터를 기준으로 했지만
- 강남**3**구가 안전하다고 보기는 어렵다
 - **5**대 범죄의 발생 정도가 상위권이며
 - **5**대 범죄에 대한 검거율이 상위권이 아니다
- 조금더 효과적인 시각화를 위해 지도 시각화를 도전하자

언제나 그렇듯 저장하자

```
In [110]: crime_gu_norm.to_csv('..../data/crime_seoul_final.csv', sep=',', encoding='utf-8')
```

Folium 기초

Folium



Python data, leaflet.js maps

`folium` builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the `leaflet.js` library. Manipulate your data in Python, then visualize it in on a Leaflet map via `folium`.

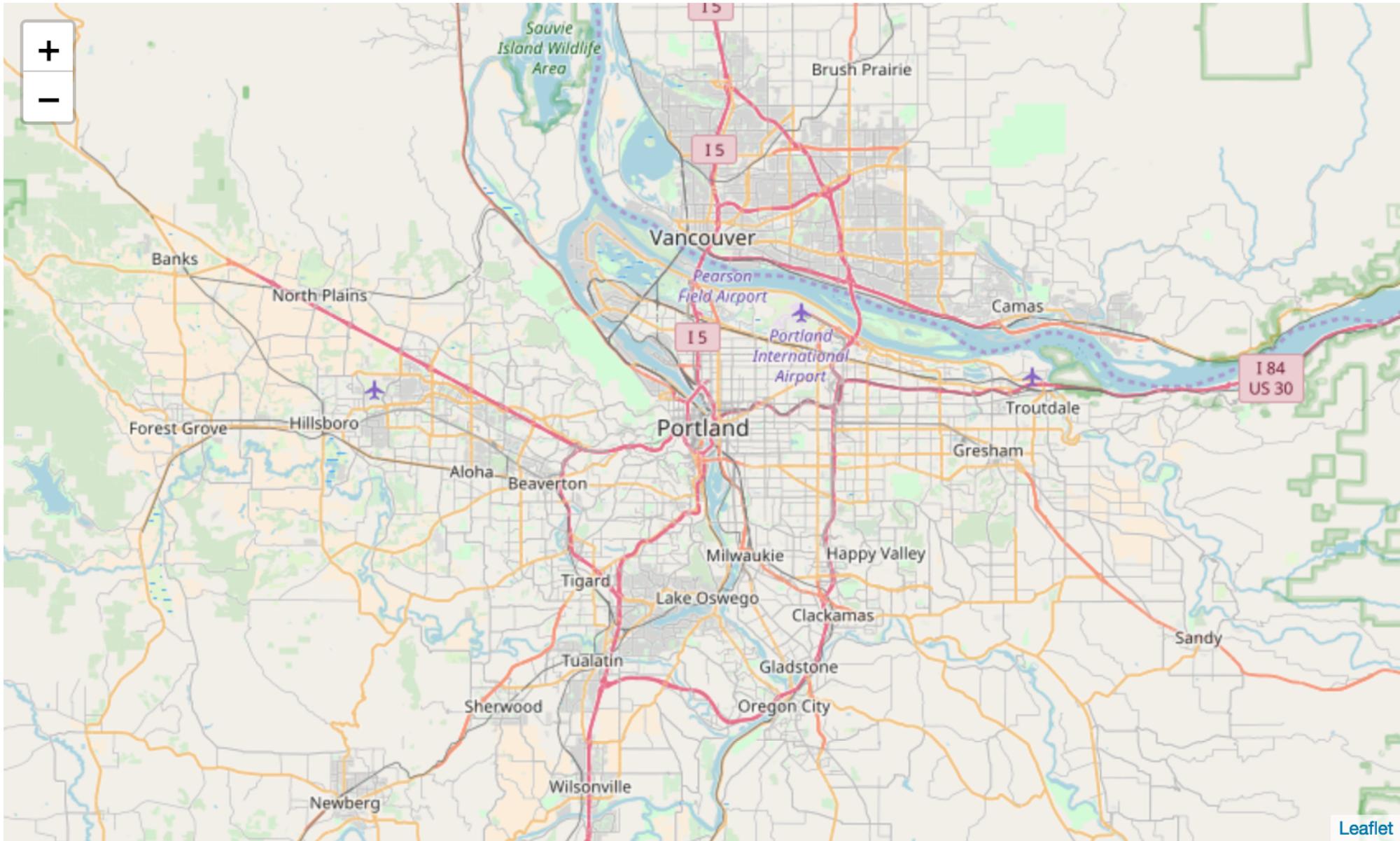
-
- 터미널을 열고

pip install folium

```
In [56]: import folium
```

```
In [57]: m = folium.Map(location=[45.5236, -122.6750])  
m
```

- **import**하고
- 위도, 경도 정보를 던지면 된다



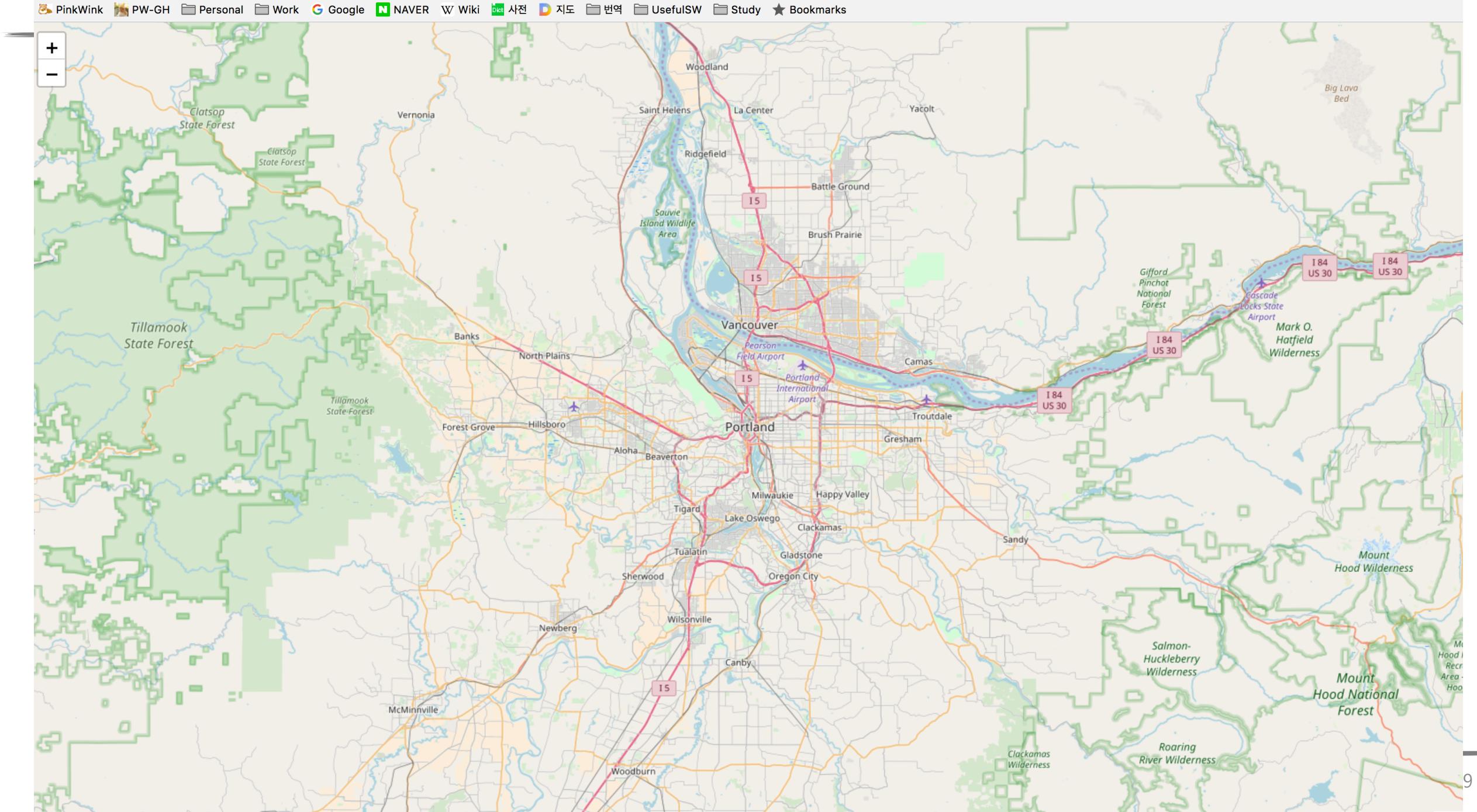
Leaflet

In [58]:

```
m.save('..../data/folium_map_save.html')
```

- 지도 화면을 파일로 저장도 할 수 있다

이름	수정일
01. POP_result.csv	2018년 5월 11일 오후 11:10
02_Crime_data_by_loc.csv	2017년 10월 19일 오후 8:34
02_Crime_data.csv	2017년 10월 20일 오전 10:52
02_crime_station.csv	2018년 5월 17일 오전 12:57
02_Seoul_POP_data.csv	2018년 5월 15일 오전 1:11
03_crime_seoul_final.csv	어제 오후 10:06
03_US_Unemployment_Oct2012.csv	2018년 5월 6일 오전 1:22
folium_map_save.html	어제 오후 10:06
03_skorea_municipalities_geo_simple.json	2018년 5월 6일 오전 1:22
03_us-states.json	2018년 5월 6일 오전 1:22
01_Population_Seoul.xls	2018년 5월 9일 오전 9:01
02. sales-funnel.xlsx	2018년 5월 6일 오전 1:22



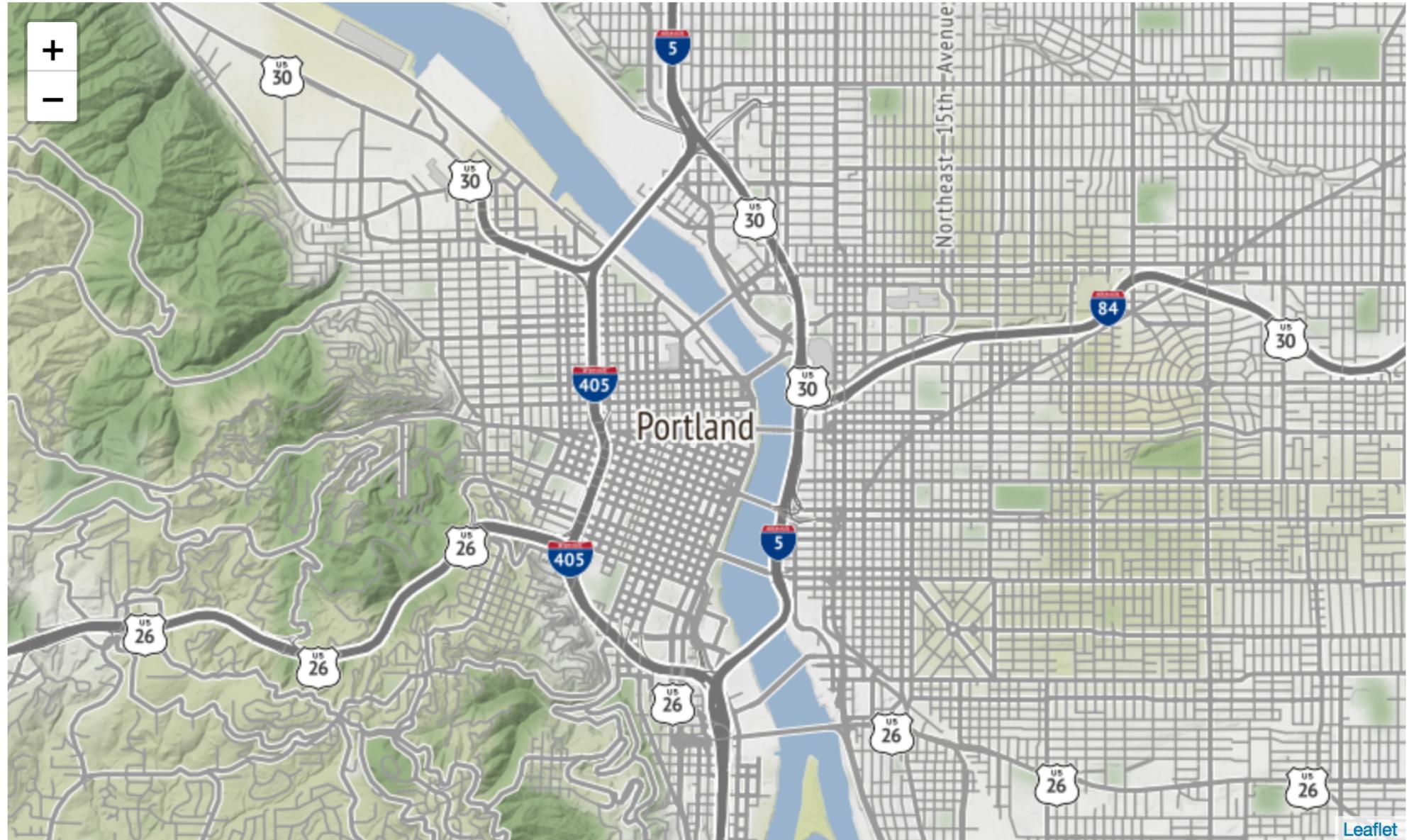
```
In [59]: folium.Map(location=[45.5236, -122.6750], tiles='Stamen Toner', zoom_start=13 )
```

- **tiles** 옵션으로 지도 스타일을 지정할 수 있다
- 처음 시작할때 확대 정도도 **zoom_start** 옵션으로 지정할 수 있다



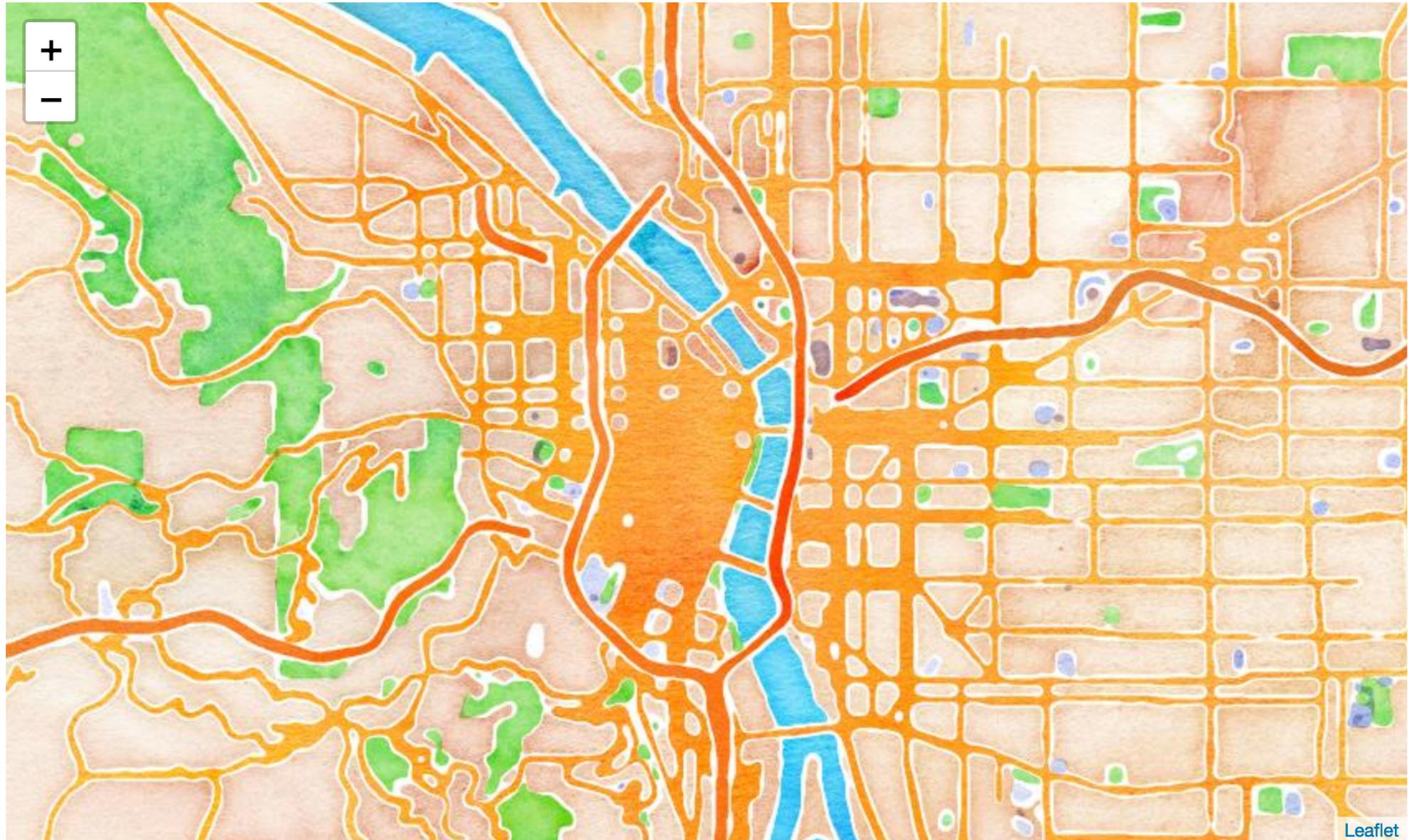
In [65]:

```
folium.Map(location=[45.5236, -122.6750], tiles='Stamen Terrain', zoom_start=13 )
```



In [66]:

```
folium.Map(location=[45.5236, -122.6750], tiles='Stamen Watercolor', zoom_start=13 )
```



```
In [67]: my_map = folium.Map(location=[45.372, -121.6972], zoom_start=12,  
                      tiles='Stamen Terrain')  
  
folium.Marker([45.3288, -121.6625], popup='<i>Mt. Hood Meadows</i>').add_to(my_map)  
folium.Marker([45.3311, -121.7113], popup='<b>Timberline Lodge</b>').add_to(my_map)  
  
my_map
```

- **Marker**를 추가할 수 있다
- **popup** 옵션은 클릭하면 글자가 나타나게 하는 것



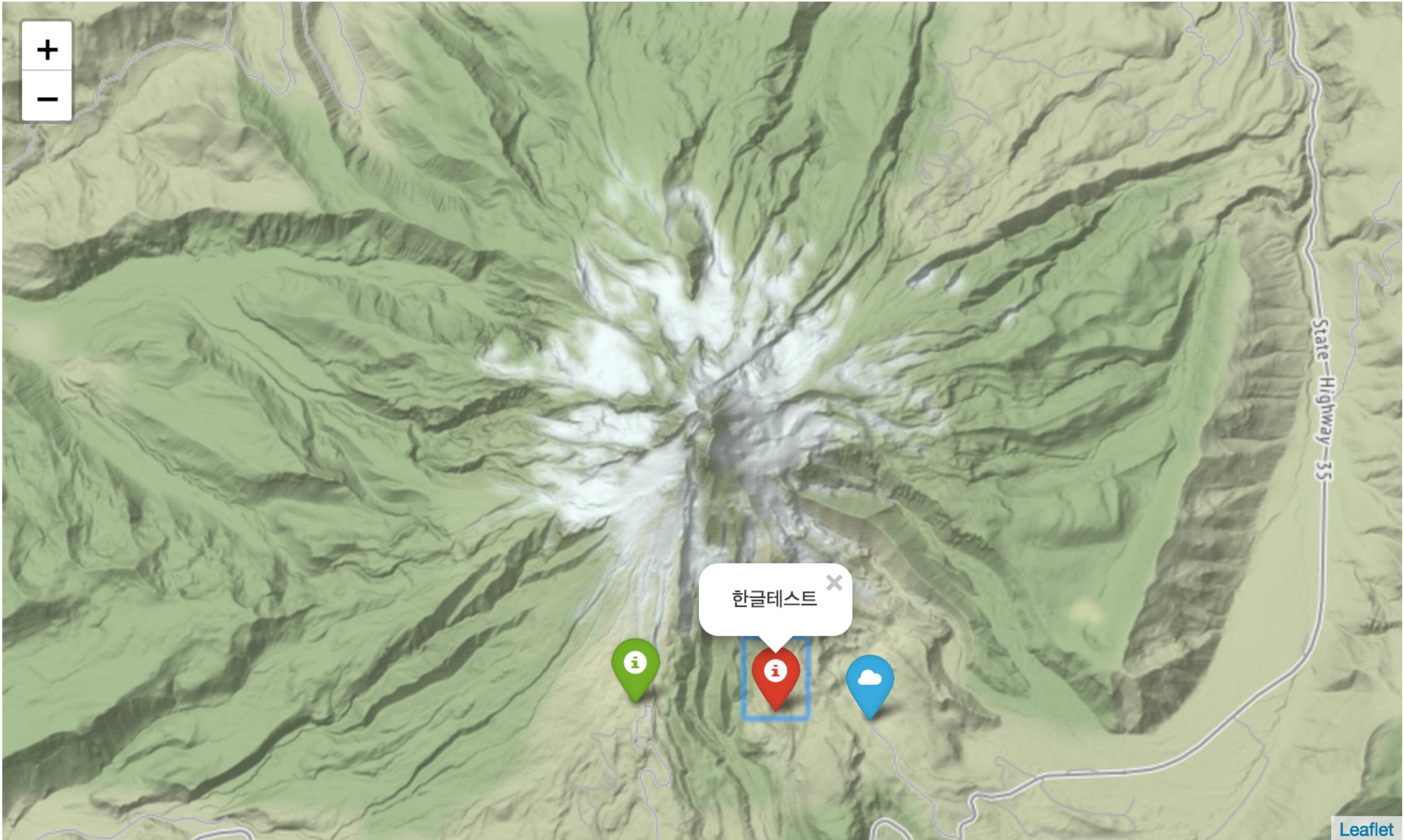
In [68]:

```
m = folium.Map(location=[45.372, -121.6972], zoom_start=12, tiles='Stamen Terrain')

folium.Marker(location=[45.3288, -121.6625], popup='Mt. Hood Meadows',
              icon=folium.Icon(icon='cloud')).add_to(m)
folium.Marker(location=[45.3311, -121.7113], popup='Timberline Lodge',
              icon=folium.Icon(color='green')).add_to(m)
folium.Marker(location=[45.3300, -121.6823], popup='한글테스트',
              icon=folium.Icon(color='red', icon='info-sign')).add_to(m)

m
```

- **Marker**의 색상이나 모양을 변경할 수 있다. 자세한 옵션은 공식 홈에서...
- 한글도 0.3.0부터 지원되기 시작했다



```
In [78]: m = folium.Map(location=[45.5236, -122.6750], tiles='Stamen Toner', zoom_start=13)

folium.CircleMarker(location=[45.5215, -122.6261], radius=50,
                     popup='Laurelhurst Park', color='#3186cc',
                     fill=True, fill_color='#3186cc').add_to(m)

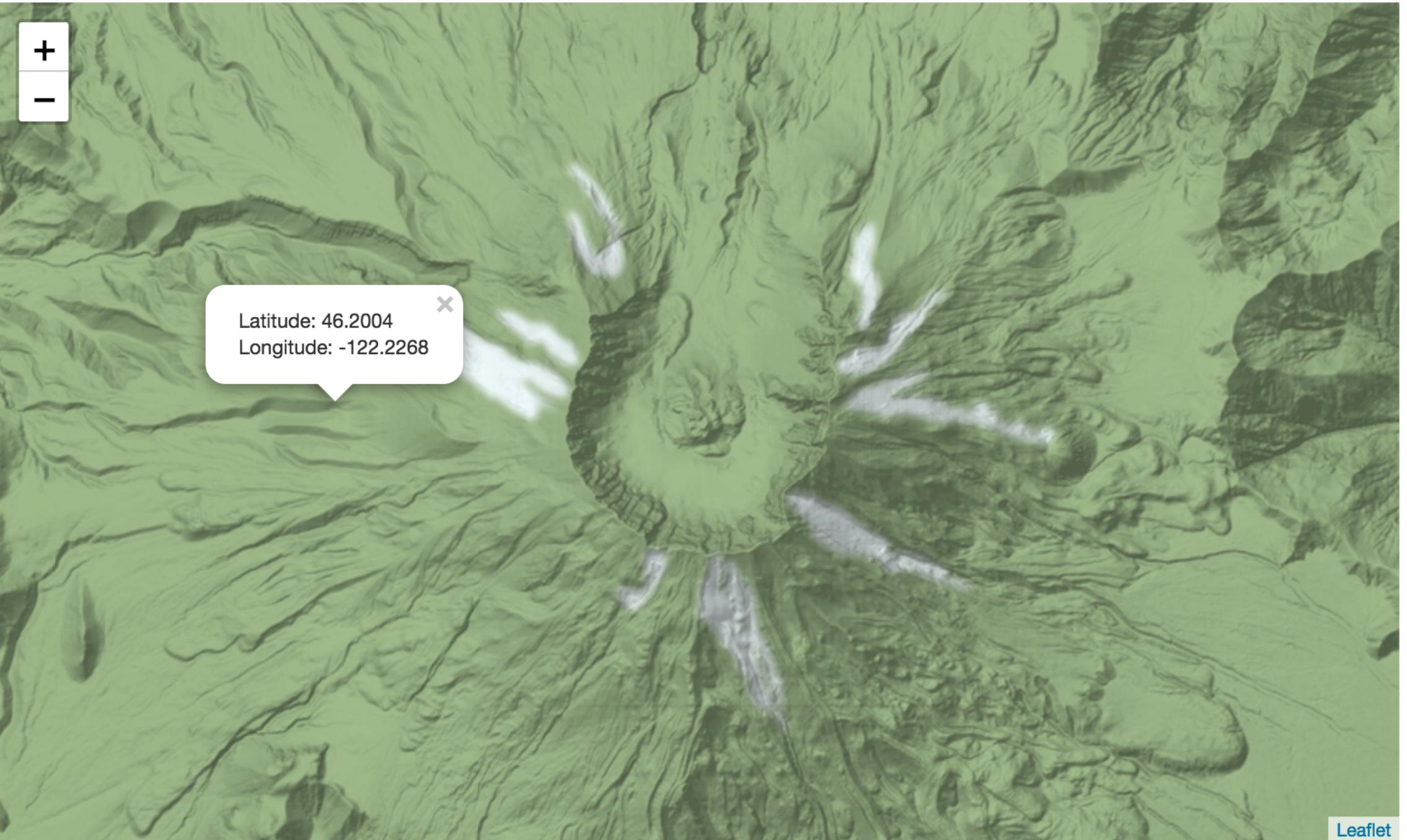
m
```

- **CircleMarker** 옵션으로 원 모양의 마커를 그릴 수 있다



```
In [81]: m = folium.Map(location=[46.1991, -122.1889], tiles='Stamen Terrain', zoom_start=13)
m.add_child(folium.LatLngPopup())
m
```

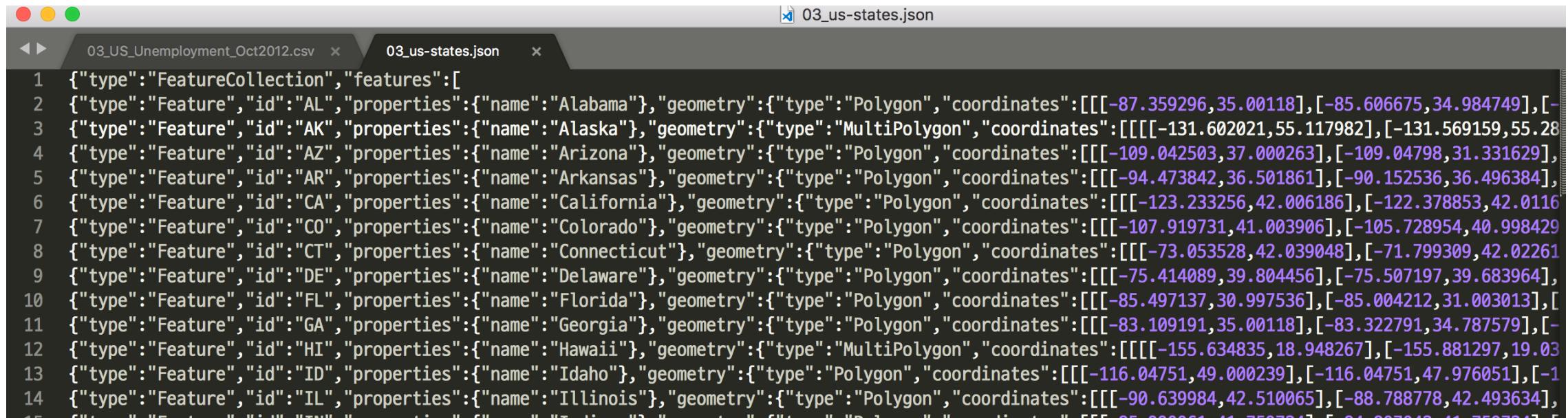
- 지도위의 어디를 클릭해도 위도 경도 정보를 준다



The screenshot shows a CSV file titled "03_US_Unemployment_Oct2012.csv" in a dark-themed code editor. The file contains 19 rows of data, each consisting of a row number and a state-unemployment rate pair. The data is as follows:

	State,Unemployment
1	State,Unemployment
2	AL,7.1
3	AK,6.8
4	AZ,8.1
5	AR,7.2
6	CA,10.1
7	CO,7.7
8	CT,8.4
9	DE,7.1
10	FL,8.2
11	GA,8.8
12	HI,5.4
13	ID,6.6
14	IL,8.8
15	IN,8.4
16	IA,5.1
17	KS,5.6
18	KY,8.1
19	LA,5.9

- 2012년 기준 미국의 주별 실업률 데이터



```
03_us-states.json
1 {"type": "FeatureCollection", "features": [
2 {"type": "Feature", "id": "AL", "properties": {"name": "Alabama"}, "geometry": {"type": "Polygon", "coordinates": [[[[-87.359296, 35.00118], [-85.606675, 34.984749], [-85.606675, 34.984749], [-87.359296, 35.00118], [-87.359296, 35.00118]]]}},
3 {"type": "Feature", "id": "AK", "properties": {"name": "Alaska"}, "geometry": {"type": "MultiPolygon", "coordinates": [[[[-131.602021, 55.117982], [-131.569159, 55.2828], [-131.569159, 55.2828], [-131.602021, 55.117982], [-131.602021, 55.117982]]]}},
4 {"type": "Feature", "id": "AZ", "properties": {"name": "Arizona"}, "geometry": {"type": "Polygon", "coordinates": [[[[-109.042503, 37.000263], [-109.04798, 31.331629], [-109.04798, 31.331629], [-109.042503, 37.000263], [-109.042503, 37.000263]]]}},
5 {"type": "Feature", "id": "AR", "properties": {"name": "Arkansas"}, "geometry": {"type": "Polygon", "coordinates": [[[[-94.473842, 36.501861], [-90.152536, 36.496384], [-90.152536, 36.496384], [-94.473842, 36.501861], [-94.473842, 36.501861]]]}},
6 {"type": "Feature", "id": "CA", "properties": {"name": "California"}, "geometry": {"type": "Polygon", "coordinates": [[[[-123.233256, 42.006186], [-122.378853, 42.0116], [-122.378853, 42.0116], [-123.233256, 42.006186], [-123.233256, 42.006186]]]}},
7 {"type": "Feature", "id": "CO", "properties": {"name": "Colorado"}, "geometry": {"type": "Polygon", "coordinates": [[[[-107.919731, 41.003906], [-105.728954, 40.998429], [-105.728954, 40.998429], [-107.919731, 41.003906], [-107.919731, 41.003906]]]}},
8 {"type": "Feature", "id": "CT", "properties": {"name": "Connecticut"}, "geometry": {"type": "Polygon", "coordinates": [[[[-73.053528, 42.039048], [-71.799309, 42.022611], [-71.799309, 42.022611], [-73.053528, 42.039048], [-73.053528, 42.039048]]]}},
9 {"type": "Feature", "id": "DE", "properties": {"name": "Delaware"}, "geometry": {"type": "Polygon", "coordinates": [[[[-75.414089, 39.804456], [-75.507197, 39.683964], [-75.507197, 39.683964], [-75.414089, 39.804456], [-75.414089, 39.804456]]]}},
10 {"type": "Feature", "id": "FL", "properties": {"name": "Florida"}, "geometry": {"type": "Polygon", "coordinates": [[[[-85.497137, 30.997536], [-85.004212, 31.003013], [-85.004212, 31.003013], [-85.497137, 30.997536], [-85.497137, 30.997536]]]}},
11 {"type": "Feature", "id": "GA", "properties": {"name": "Georgia"}, "geometry": {"type": "Polygon", "coordinates": [[[[-83.109191, 35.00118], [-83.322791, 34.787579], [-83.322791, 34.787579], [-83.109191, 35.00118], [-83.109191, 35.00118]]]}},
12 {"type": "Feature", "id": "HI", "properties": {"name": "Hawaii"}, "geometry": {"type": "MultiPolygon", "coordinates": [[[[-155.634835, 18.948267], [-155.881297, 19.0311], [-155.881297, 19.0311], [-155.634835, 18.948267], [-155.634835, 18.948267]]]}},
13 {"type": "Feature", "id": "ID", "properties": {"name": "Idaho"}, "geometry": {"type": "Polygon", "coordinates": [[[[-116.04751, 49.000239], [-116.04751, 47.976051], [-116.04751, 47.976051], [-116.04751, 49.000239], [-116.04751, 49.000239]]]}},
14 {"type": "Feature", "id": "IL", "properties": {"name": "Illinois"}, "geometry": {"type": "Polygon", "coordinates": [[[[-90.639984, 42.510065], [-88.788778, 42.493634], [-88.788778, 42.493634], [-90.639984, 42.510065], [-90.639984, 42.510065]]]}],
15 {"type": "Feature", "id": "IN", "properties": {"name": "Indiana"}, "geometry": {"type": "Polygon", "coordinates": [[[[-85.750724, 40.222261], [-85.750724, 40.222261], [-85.750724, 40.222261], [-85.750724, 40.222261], [-85.750724, 40.222261]]]}]
```

- 미국의 각 주 경계를 위도/경도 정보로 경계선을 만들어 둔 **JSON** 파일

In [124]:

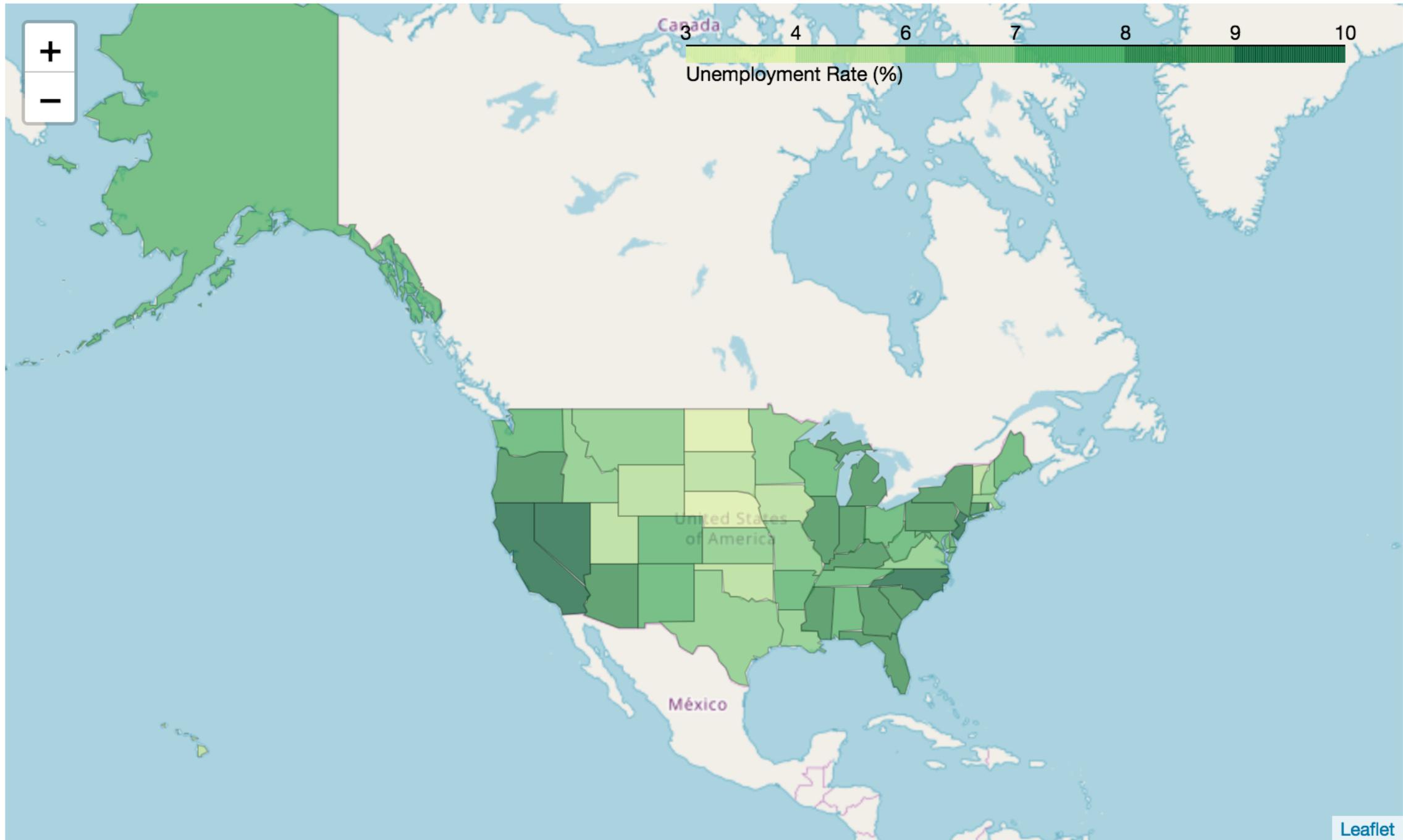
```
import json
import folium
import pandas as pd

state_data = pd.read_csv('../data/US_Unemployment_Oct2012.csv')

m = folium.Map(location=[48, -102], zoom_start=3)
m.choropleth(geo_data='../data/us-states.json', data=state_data,
             columns=['State', 'Unemployment'], key_on='feature.id',
             fill_color='YlGn', fill_opacity=0.7, line_opacity=0.2,
             legend_name='Unemployment Rate (%)')

m
```

- 경계선 정보(**json**)에 고유한 **ID**를 두고 이것과 데이터를 매칭 시켜서
- **choropleth** 옵션을 사용하면 지도에 **colormap**을 입힐 수 있다.



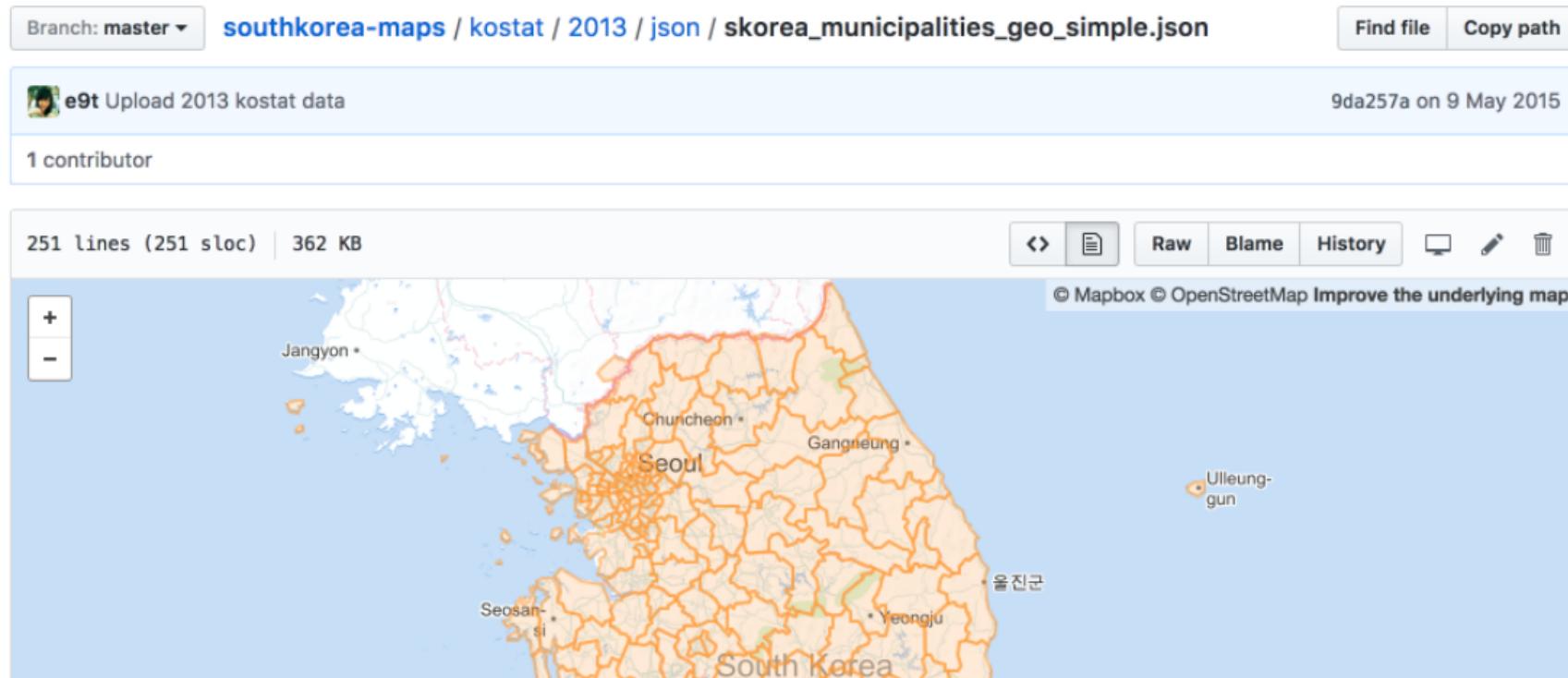
이제 우리 범죄 현황 데이터도 지도 시각화를 ...

The screenshot shows a GitHub repository page for 'southkorea / southkorea-maps'. The repository has 27 pull requests, 158 stars, and 49 forks. The 'Code' tab is selected. There are 4 issues and 0 pull requests. The repository has 0 projects, 0 wiki pages, and a pulse graph.

- 우리나라 지도에 대해서도 e9t라는 아이디로 git에서 활동하시는
- Lucy Park 님이 공개하고 있다
- Lucy Park 님은 텍스트 마이닝때 좀 더 자세히 소개하는 걸로~

Author	Commit Message	Date
gadm	Update gadm downloader	2 years ago
kostat	create geojson from topojson	a year ago
license	Merge branch 'master' of github.com:southkorea/southkorea-maps	2 years ago
popong	Remove temp files	a year ago
static	Add precinct map screenshot	a year ago

구별 범죄 현황 지도 시각화



- 전국 시군구 단위로 표현한 json파일이 있는데 2013년에 만들어져
- 현재 우리나라 시군구와는 딱 일치하지 않는다
- 물론 서울시는 큰 문제가 없다

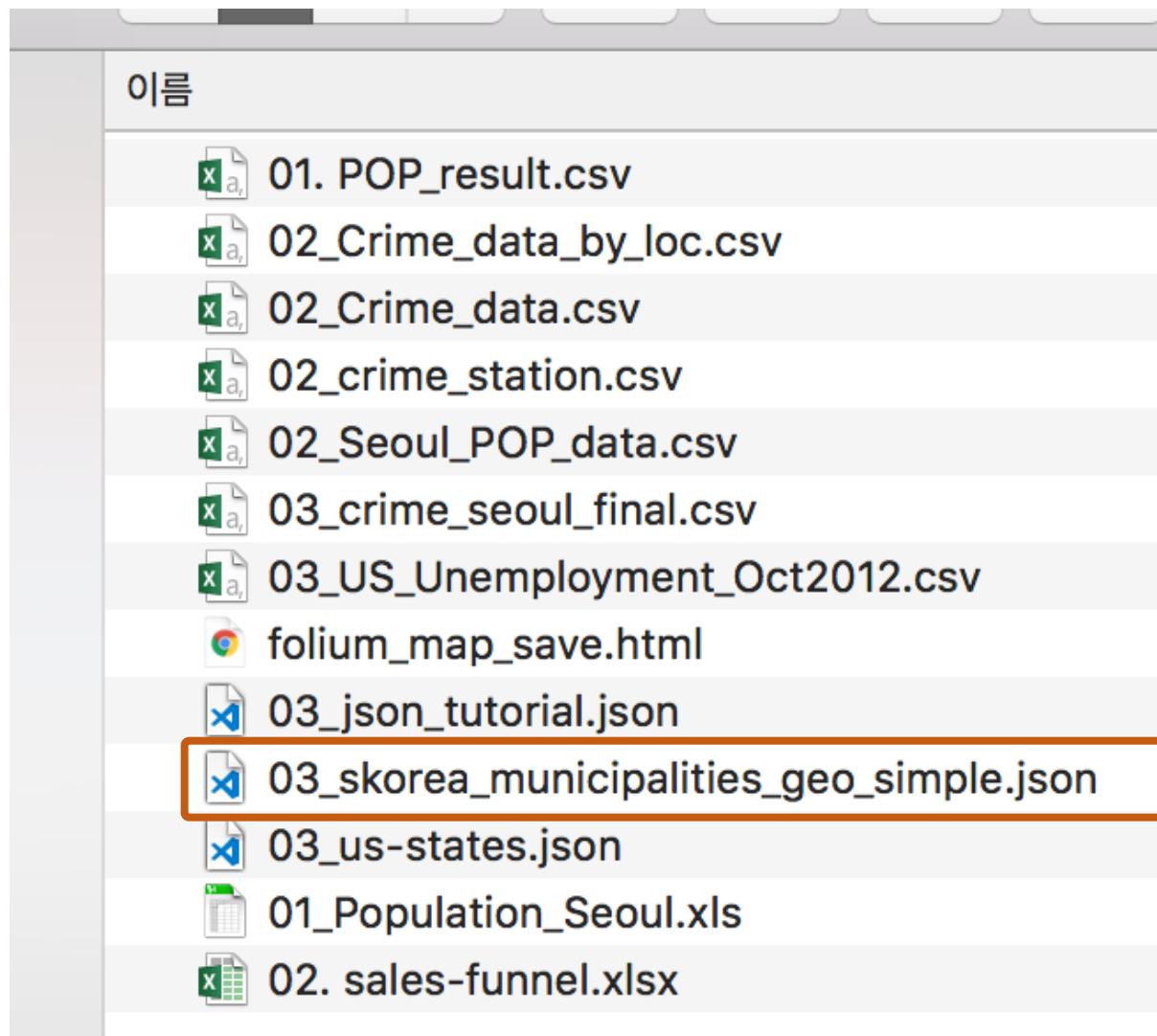


구별 범죄 현황 지도 시각화

```
{"type": "FeatureCollection", "features": [{"type": "Feature", "properties": {"code": "39020", "name": "서귀포시", "name_eng": "Seogwipo-si", "base_year": "2013"}, "geometry": {"type": "Polygon", "coordinates": [[[126.1701670531016, 33.27833920373795], [126.17796199822318, 33.28904450342792], [126.20366562455006, 33.292582069125935], [126.23227804627976, 33.28054651709448], [126.27081469983565, 33.29305651947374], [126.28692516789717, 33.3095121006344], [126.32582807936741, 33.32307849544494], [126.33863040850332, 33.3370002289742], [126.37692773779688, 33.34348978209229], [126.42008790072302, 33.33448285889923], [126.43967991919219, 33.34042205120408], [126.44742456956253, 33.35529267692266], [126.4918716225209, 33.351283204155564], [126.54001194569452, 33.35949354681064], [126.55310458705854, 33.368769331336765], [126.5830715764017, 33.368027779973275], [126.65655971322707, 33.39658294968576], [126.68911351796443, 33.39691133705054], [126.70784071345439, 33.41819831436605], [126.73472429940135, 33.422146338653775], [126.75951968887561, 33.41497179862364], [126.79074582138537, 33.43825351998022], [126.82385785054652, 33.44338028367965], [126.8898274152664, 33.47823603716075], [126.90529687529418, 33.480086403367224], [126.91451306461225, 33.46996423390579], [126.93834260748045, 33.47055272752707], [126.91017277435895, 33.402610687558585], [126.90919306806411, 33.3879255679562], [126.87535882585192, 33.36822409322237], [126.8705501162605, 33.34975159885652], [126.84671810776004, 33.33005904428782], [126.83805963301805, 33.30541883125935], [126.81058128765947, 33.29810905802241], [126.78058721090692, 33.30348117728641], [126.74744517061241, 33.27477917807315], [126.69397491052028, 33.26589785324775], [126.66139509103422, 33.26764883602634], [126.64370899012728, 33.26176379473827], [126.62124671799259, 33.238001461003755], [126.60023270919079, 33.23253794499945], [126.58793276547512, 33.24078903556731], [126.55000275643071, 33.233994617394124], [126.5228788872885, 33.237410624439875], [126.5112182224236, 33.226741039104716], [126.47233571972863, 33.219241198557235], [126.45245786615838, 33.23676200849374], [126.43326944303328, 33.22862021392413], [126.41456422346796, 33.24077765309614], [126.38663415314419, 33.2281960784446], [126.32186188505729, 33.2348134021202], [126.29961093191966, 33.219470994774156], [126.29462959454436, 33.20325737472914], [126.27360444082919, 33.19065258266024], [126.2344607950971, 33.231192587887506], [126.20182608174507, 33.242644947267415], [126.18421578146823, 33.2556336839387], [126.1701670531016, 33.27833920373795]]]}}, {"type": "Feature", "properties": {"code": "39010", "name": "제주시", "name_eng": "Jeju-si", "base_year": "2013"}, "geometry": {"type": "MultiPolygon", "coordinates": [[[[[[126.95749379080732, 33.52245641440839], [126.9731989684708, 33.498754077952555], [126.94954584507536, 33.4895447597362], [126.94279608187666, 33.50103054337929], [126.95749379080732, 33.52245641440839]], [[[[126.90529687529418, 33.480086403367224], [126.8898274152664, 33.47823603716075], [126.82385785054652, 33.44338028367965], [126.79074582138537, 33.43825351998022], [126.75951968887561, 33.41497179862364], [126.73472429940135, 33.422146338653775], [126.70784071345439, 33.41819831436605], [126.68911351796443, 33.39691133705054], [126.65655971322707, 33.39658294968576], [126.5830715764017, 33.368027779973275], [126.55310458705854, 33.368769331336765], [126.54001194569452, 33.35949354681064], [126.4918716225209, 33.351283204155564], [126.44742456956253, 33.35529267692266], [126.43967991919219, 33.34042205120408], [126.42008790072302, 33.33448285889923], [126.37692773779688, 33.34348978209229], [126.33863040850332, 33.3370002289742], [126.32582807936741, 33.32307849544494], [126.28692516789717, 33.3095121006344], [126.27081469983565, 33.29305651947374], [126.23227804627976, 33.28054651709448], [126.20366562455006, 33.292582069125935], [126.17796199822318, 33.28904450342792], [126.1701670531016, 33.27833920373795], [126.16319295942347, 33.32059642402829], [126.1692541537235, 33.34310558958249], [126.21461375807213, 33.37309156042842], [126.2619192214544, 33.413737972061774], [126.26197058380335, 33.4314254768364], [126.3070400569873, 33.44823091074929], [126.31005406486322, 33.463098336550836], [126.33749501698236, 33.46367421762971], [126.3876117613898, 33.4865145900699], [126.39670361974751, 33.4811364736424], [126.48208433034038, 33.50677622835366], [126.49529547568828, 33.5184121837681], [126.51667984271614, 33.51331402436857], [126.54176863235955, 33.527562476211756], [126.55087422195945, 33.51759024881304], [126.58362236141552, 33.523927713435334], [126.59940627536525, 33.53451980393117], [126.63484958960886, 33.53527429124482], [126.64568938881722, 33.553414822717535], [126.66008979414049, 33.548871831149306], [126.75800613842082, 33.55555170516014], [126.78655545465347, 33.562854180697364], [126.80185825451599, 33.553702356637885], [126.82997198815602, 33.55685083846837], [126.8331417008821, 33.543008206988866], [126.85956040111658, 33.5237459631211], [126.87389326759744, 33.52840985278047], [126.9044869524447, 33.52306487335703], [126.91799070617053, 33.498337310154], [126.90529687529418, 33.480086403367224]]]}}, {"type": "Feature", "properties": {"code": "38400", "name": "합천군", "name_eng": "Hapcheon-gun", "base_year": "2013"}, "geometry": {"type": "Polygon", "coordinates": [[[128.1664702916369, 35.75973538713466], [128.1894575867061, 35.751146695515], [128.20376038881219, 35.71979380237135], [128.20429067824932, 35.68136577618283], [128.16631648353624, 35.67023484769286], [128.1659217940842, 35.650471858798795], [128.18707873977326, 35.65477072180895], [128.20320925841966, 35.64077141381145], [128.26329705154626, 35.63966480980178], [128.2805996850208, 35.64872993618149], [128.30824855773733, 35.652120313603305], [128.35712252872025, 35.63944851830449], [128.36455141800863, 35.6086521477258], [128.35409802241534, 35.60371927292992], [128.36393478624836, 35.574975399652736], [128.3524935496341, 35.5500008227065], [128.35496578489835, 35.531894795852615], [128.3744422383229, 35.515918290232825], [128.37407817895624, 35.49890497146474], [128.34872680409157, 35.497321764508584], [128.31804497181153, 35.51223556949818], [128.27145497185825, 35.507959905286114], [128.25718580241585, 35.49069085259646], [128.21164046549285, 35.47987941280686], [128.20932135887128, 35.44514848238987], [128.18760031125066, 35.416098144676965], [128.19163963328742, 35.395600668867864], [128.15025100085697, 35.3829356736949], [128.13122380773524, 35.38470212549547], [128.0974979573329, 35.36509977790733], [128.08252050187886, 35.367071488869165], [128.0911919700177, 35.376295973942185], [128.06149064233767, 35.40249128057027], [128.0370495961085, 35.412354291679875], [127.98909413543431, 35.44239897076061], [127.9644438069297, 35.49708390818465], [127.96199110121547, 35.515544170593], [127.9572157884059, 35.54689130661954], [127.98168204634861, 35.5604840613817], [128.000000000005576, 35.62512172418849], [128.02274033148038, 35.648851589143554], [128.04669466251949, 35.65846299958867], [128.05252951106863, 35.67512928831512], [128.09341425381487, 35.679002127942525], [128.04635387023492, 35.7491552480925], [128.0530956577539, 35.77520364109215], [128.03781558941037, 35.78313812768474], [128.05675234917973, 35.80278899493612], [128.07162436990026, 35.803127367997455], [128.09285821427525, 35.82781190417947], [128.12631731840716, 35.81934758210269], [128.13099154092245, 35.7863677785382], [128.16624529480816, 35.77401846166936]]]}}, {"type": "Feature", "properties": {"code": "38390", "name": "거창군", "name_eng": "Geochang-gun", "base_year": "2013"}, "geometry": {"type": "Polygon", "coordinates": [[[128.0808946019517, 35.83592789823322],
```

- 전국 데이터에서 서울시의 구별 데이터만 따로 가져와서 만든다

```
{"type":"FeatureCollection","features": [ {"type":"Feature","id":"강동구","properties":{"code":"11250","name":"강동구","name_eng":"Gangdong-gu"}, {"type":"Feature","id":"송파구","properties":{"code":"11240","name":"송파구","name_eng":"Songpa-gu","base_y":100}, {"type":"Feature","id":"강남구","properties":{"code":"11230","name":"강남구","name_eng":"Gangnam-gu"}, {"type":"Feature","id":"서초구","properties":{"code":"11220","name":"서초구","name_eng":"Seocho-gu"}, {"type":"Feature","id":"관악구","properties":{"code":"11210","name":"관악구","name_eng":"Gwanak-gu"}, {"type":"Feature","id":"동작구","properties":{"code":"11200","name":"동작구","name_eng":"Dongjak-gu"}, {"type":"Feature","id":"영등포구","properties":{"code":"11190","name":"영등포구","name_eng":"Yeongdeungpo-gu"}, {"type":"Feature","id":"금천구","properties":{"code":"11180","name":"금천구","name_eng":"Geumcheon-gu"}, {"type":"Feature","id":"구로구","properties":{"code":"11170","name":"구로구","name_eng":"Guro-gu"}, {"type":"Feature","id":"강서구","properties":{"code":"11160","name":"강서구","name_eng":"Gangseo-gu"}, {"type":"Feature","id":"양천구","properties":{"code":"11150","name":"양천구","name_eng":"Yangcheon-gu"}, {"type":"Feature","id":"마포구","properties":{"code":"11140","name":"마포구","name_eng":"Mapo-gu"}, {"type":"Feature","id":"서대문구","properties":{"code":"11130","name":"서대문구","name_eng":"Seodaemun-gu"}, {"type":"Feature","id":"은평구","properties":{"code":"11120","name":"은평구","name_eng":"Eunpyeong-gu"}, {"type":"Feature","id":"노원구","properties":{"code":"11110","name":"노원구","name_eng":"Nowon-gu"}, {"type":"Feature","id":"도봉구","properties":{"code":"11100","name":"도봉구","name_eng":"Dobong-gu"}, {"type":"Feature","id":"강북구","properties":{"code":"11090","name":"강북구","name_eng":"Gangbuk-gu"}, {"type":"Feature","id":"성북구","properties":{"code":"11080","name":"성북구","name_eng":"Seongbuk-gu"}, {"type":"Feature","id":"중랑구","properties":{"code":"11070","name":"중랑구","name_eng":"Jungnang-gu"}, {"type":"Feature","id":"동대문구","properties":{"code":"11060","name":"동대문구","name_eng":"Dongdaemun-gu"}, {"type":"Feature","id":"광진구","properties":{"code":"11050","name":"광진구","name_eng":"Gwangjin-gu"}, {"type":"Feature","id":"성동구","properties":{"code":"11040","name":"성동구","name_eng":"Seongdong-gu"}, {"type":"Feature","id":"용산구","properties":{"code":"11030","name":"용산구","name_eng":"Yongsan-gu"}, {"type":"Feature","id":"중구","properties":{"code":"11020","name":"중구","name_eng":"Jung-gu"}, {"type":"Feature","id":"종로구","properties":{"code":"11010","name":"종로구","name_eng":"Jongno-gu"}, {"type":"Feature","id":"성북구","properties":{"code":"11000","name":"성북구","name_eng":"Seongbuk-gu"}]}]
```



In [139]:

```
import pandas as pd
import folium
import json

crime_gu_norm = pd.read_csv('../data/crime_seoul_final.csv',
                           index_col = 0, encoding='utf-8')
geo_path = '../data/skorea_municipalities_geo_simple.json'
geo_str = json.load(open(geo_path, encoding='utf-8'))

crime_gu_norm.head()
```

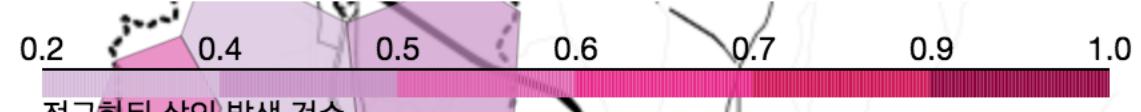
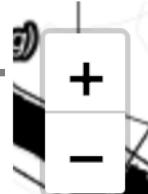
- 저장했던 **crime_gu_norm** 데이터도 다시 읽고
- json**을 **load** 명령과 **open** 명령으로 위와 같이 읽어도 된다

```
In [106]: my_map = folium.Map(location=[37.5502, 126.982], zoom_start=11, tiles='Stamen Toner')

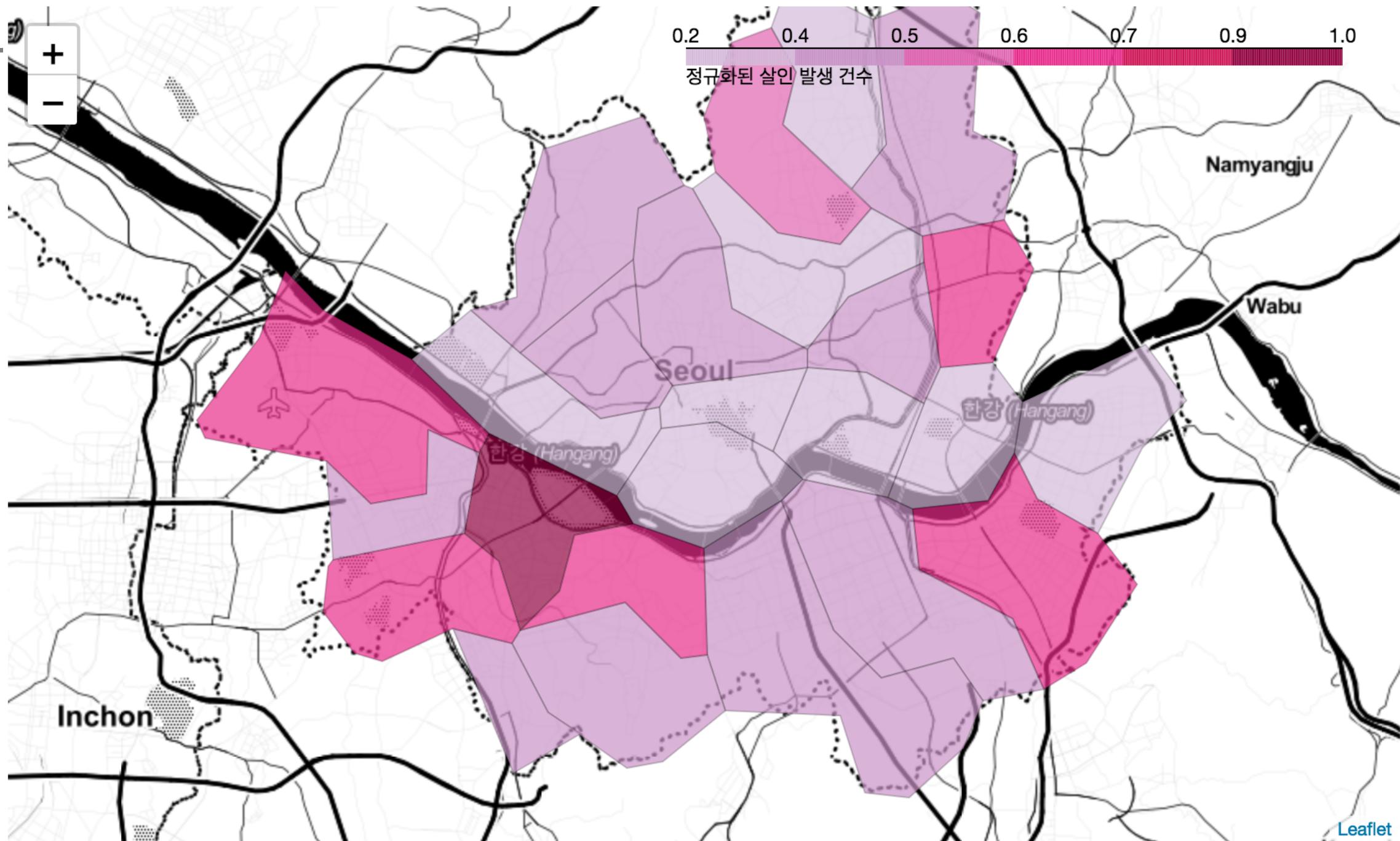
my_map.choropleth(geo_data = geo_str, data = crime_gu_norm[ '살인' ],
                  columns = [crime_gu_norm.index, crime_gu_norm[ '살인' ]],
                  fill_color = 'PuRd', key_on = 'feature.id',
                  fill_opacity=0.7, line_opacity=0.2,
                  legend_name = '정규화된 살인 발생 건수')

my_map
```

- 살인 발생 건수로 시각화를 먼저 해보자



정규화된 살인 발생 건수

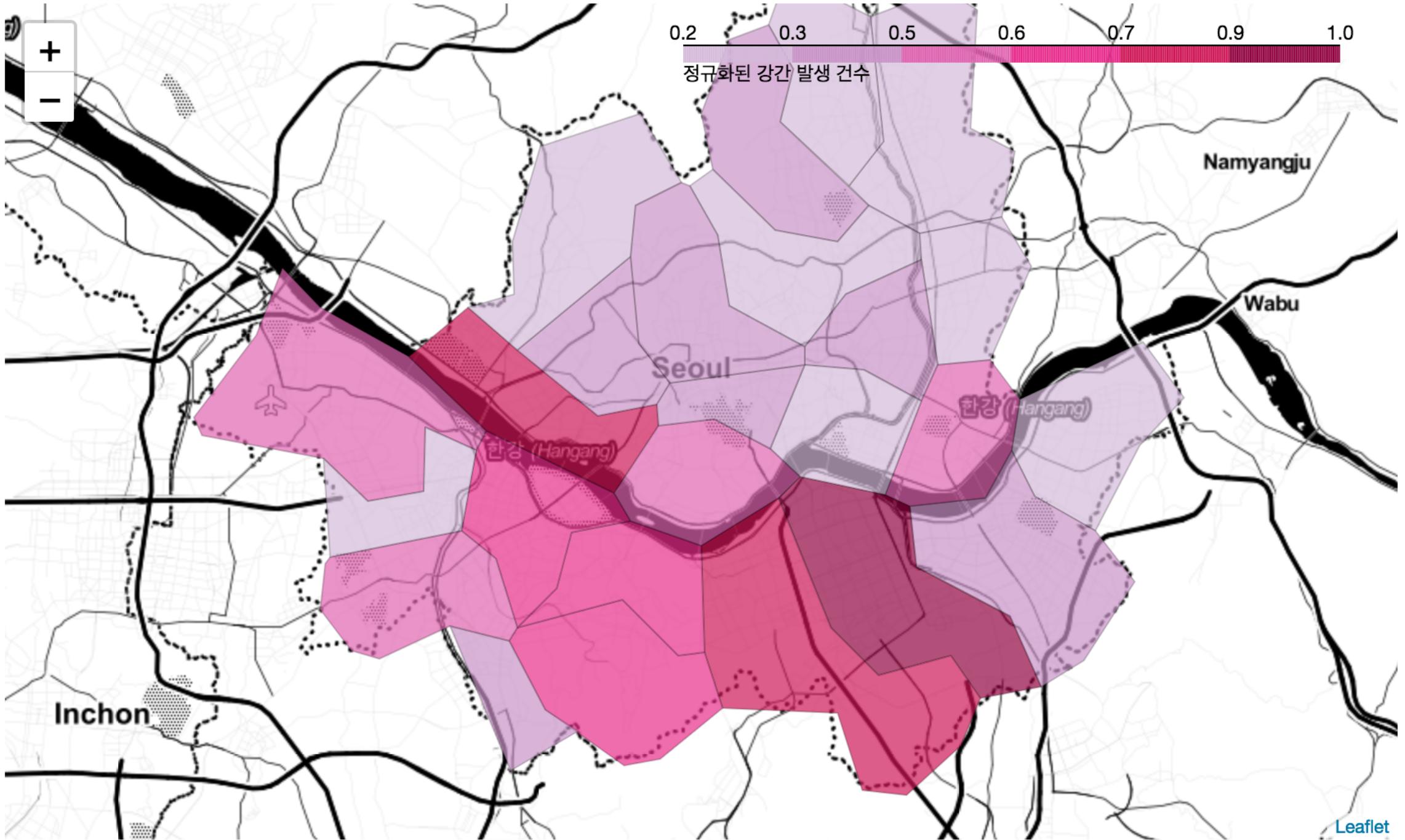


```
In [107]: my_map = folium.Map(location=[37.5502, 126.982], zoom_start=11, tiles='Stamen Toner')

my_map.choropleth(geo_data = geo_str, data = crime_gu_norm['강간'],
                  columns = [crime_gu_norm.index, crime_gu_norm['강간']],
                  fill_color = 'PuRd', key_on = 'feature.id',
                  fill_opacity=0.7, line_opacity=0.2,
                  legend_name = '정규화된 강간 발생 건수')

my_map
```

- 이번에는 강간 사건을 기준으로

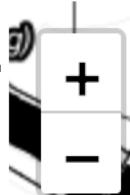


```
In [108]: my_map = folium.Map(location=[37.5502, 126.982], zoom_start=11, tiles='Stamen Toner')

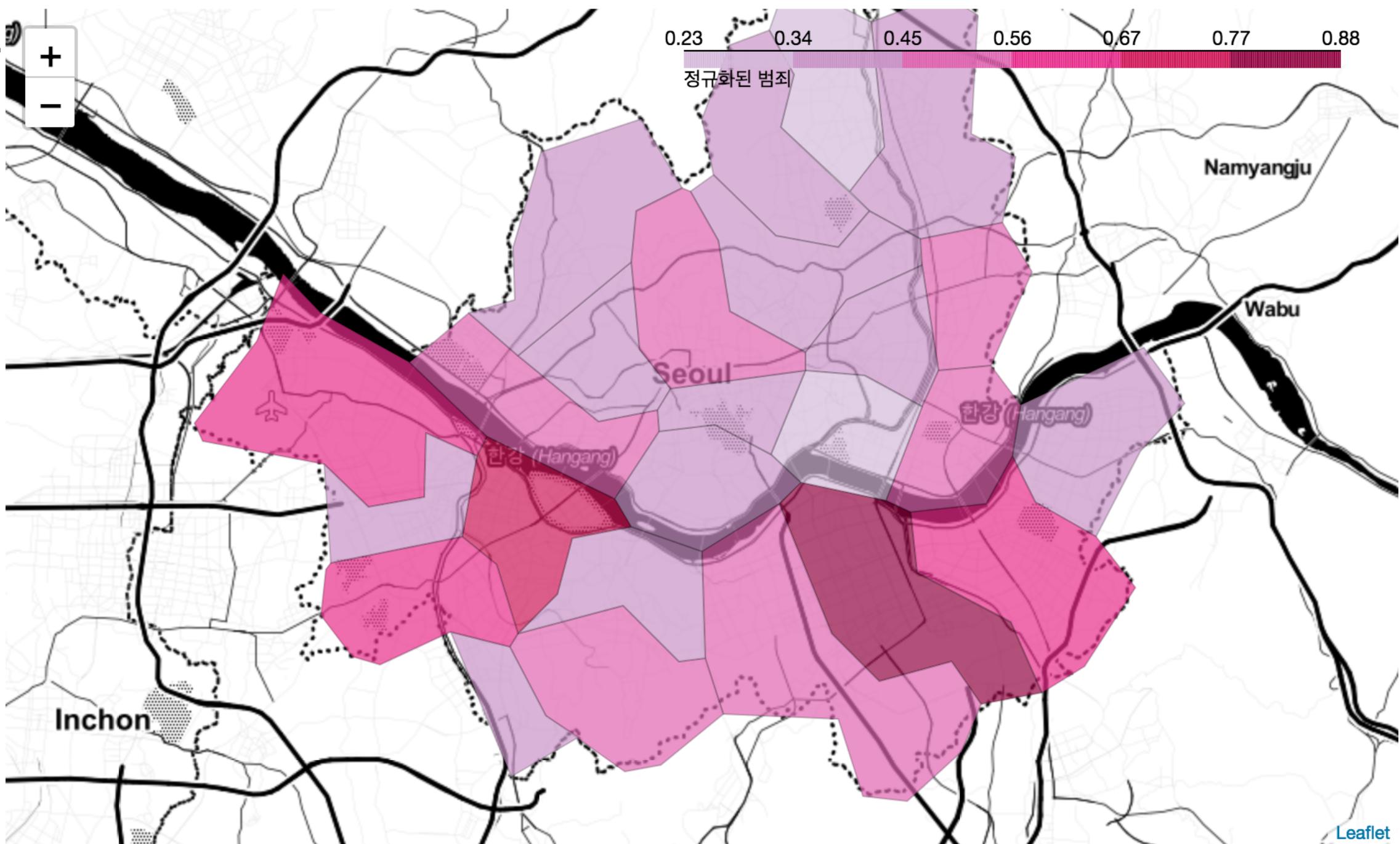
my_map.choropleth(geo_data = geo_str, data = crime_gu_norm[ '범죄' ],
                  columns = [crime_gu_norm.index, crime_gu_norm[ '범죄' ]],
                  fill_color = 'PuRd', key_on = 'feature.id',
                  fill_opacity=0.7, line_opacity=0.2,
                  legend_name = '정규화된 범죄')

my_map
```

- 범죄 전체를 평균한 데이터를 기준으로



정규화된 범죄



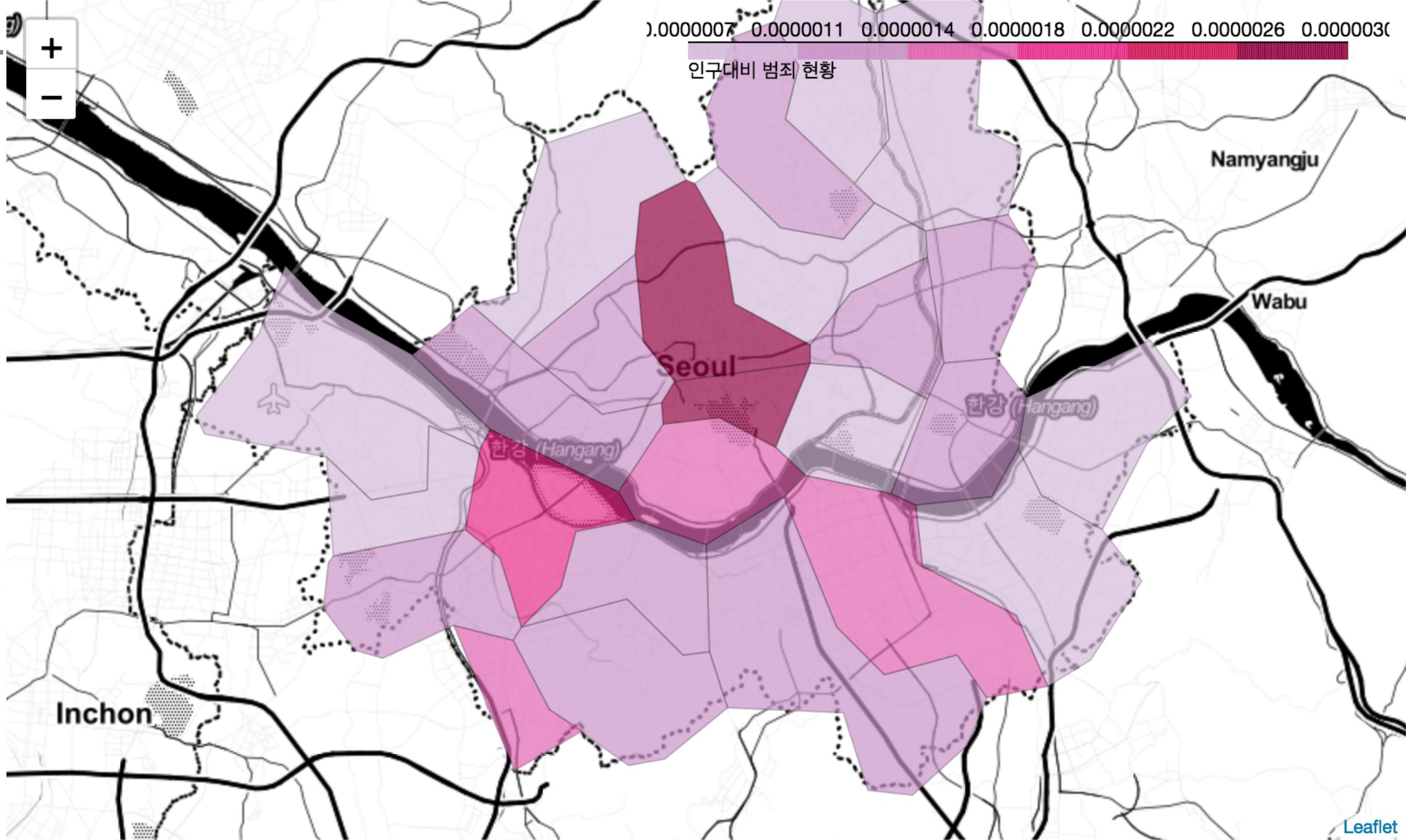
```
In [109]: tmp_crime_norm = crime_gu_norm['범죄'] / crime_gu_norm['인구수']

my_map = folium.Map(location=[37.5502, 126.982], zoom_start=11, tiles='Stamen Toner')

my_map.choropleth(geo_data = geo_str, data = tmp_crime_norm,
                  columns = [crime_gu_norm.index, tmp_crime_norm],
                  fill_color = 'PuRd', key_on = 'feature.id',
                  fill_opacity=0.7, line_opacity=0.2,
                  legend_name = '인구대비 범죄 현황')

my_map
```

- 이번에는 인구대비 범죄 데이터를 만들어서



In [110]:

```
col = ['살인검거', '강도검거', '강간검거', '절도검거', '폭력검거']

crime_station_simple = crime_station[col]
crime_station_simple.head( )
```

	살인검거	강도검거	강간검거	절도검거	폭력검거
구분					
강남	3.0	26.0	269.0	1129.0	2096.0
강동	5.0	13.0	152.0	902.0	2201.0
강북	6.0	4.0	159.0	672.0	2482.0
강서	10.0	10.0	239.0	1070.0	2768.0
관악	7.0	10.0	264.0	937.0	2707.0

- 경찰서별 데이터에서 검거 건수만 따로 저장하자

```
In [111]: crime_station_simple = crime_station_simple / crime_station_simple.max( )
crime_station_simple['검거'] = crime_station_simple.mean(axis=1)
crime_station_simple.head( )
```

구분	살인검거	강도검거	강간검거	절도검거	폭력검거	검거
강남	0.230769	1.000000	0.840625	1.000000	0.697040	0.753687
강동	0.384615	0.500000	0.475000	0.798937	0.731959	0.578102
강북	0.461538	0.153846	0.496875	0.595217	0.825407	0.506577
강서	0.769231	0.384615	0.746875	0.947741	0.920519	0.753796
관악	0.538462	0.384615	0.825000	0.829938	0.900233	0.695650

- normalize 하고...

```
In [112]:
```

```
col = ['lat', 'lng']

crime_station_simple[col] = crime_station[col]
crime_station_simple.head()
```

구분	살인검거	강도검거	강간검거	절도검거	폭력검거	검거	lat	lng
강남	0.230769	1.000000	0.840625	1.000000	0.697040	0.753687	37.509435	127.066958
강동	0.384615	0.500000	0.475000	0.798937	0.731959	0.578102	37.528511	127.126822
강북	0.461538	0.153846	0.496875	0.595217	0.825407	0.506577	37.637388	127.027324
강서	0.769231	0.384615	0.746875	0.947741	0.920519	0.753796	37.539783	126.829997
관악	0.538462	0.384615	0.825000	0.829938	0.900233	0.695650	37.474379	126.950975

- 추가로 위도/경도 정보도 붙이자

- 강남 3구가 안전할까?
 - 각종 범죄 발생건수가 절대 강남 3구가 안전하지 않음을 보여준다
 - 경찰서의 검거력 또한 최 상위권은 아니다
- 반론 :
 - 서울내에서 유흥가가 밀집해 있는 특성으로 인한 문제 일 수 있다.
 - 주민들의 주거 지역은 안전할 수 있다.

반론의 반론 : 범죄 발생 장소별 데이터를 보자

In [144]:

```
import pandas as pd

crime_loc_raw = pd.read_csv('..../data/Crime_data_by_loc.csv',
                           thousands=',', encoding='euc-kr')
crime_loc_raw.head()
```

	범죄명	장소	발생건수
0	살인	아파트, 연립 다세대	12
1	살인	단독주택	30
2	살인	노상	22
3	살인	상점	1
4	살인	숙박업소, 목욕탕	4

In [121]:

```
crime_loc_raw['범죄명'].unique()
```

```
array(['살인', '강도', '강간.추행', '절도', '폭력'], dtype=object)
```

In [122]:

```
crime_loc_raw['장소'].unique()
```

```
array(['아파트, 연립 다세대', '단독주택', '노상', '상점', '숙박업소, 목욕탕', '유흥 접객업소', '사무실',  
'역, 대합실', '교통수단', '유원지 ', '학교', '금융기관', '기타'], dtype=object)
```

In [123]:

```
import numpy as np

crime_loc = crime_loc_raw.pivot_table(crime_loc_raw,
                                         index=[ "장소"], columns=[ "범죄명"],
                                         aggfunc=[np.sum])
crime_loc.columns = crime_loc.columns.droplevel([0,1])
crime_loc.head()
```

범죄명	강간.추행	강도	살인	절도	폭력
장소					
교통수단	691	0	0	457	222
금융기관	2	1	1	1081	42
기타	2128	67	65	21734	26382
노상	986	87	22	9329	24535
단독주택	395	15	30	2241	3579

In [124]:

```
col = ['살인', '강도', '강간', '절도', '폭력']
crime_loc_norm = crime_loc / crime_loc.max()
crime_loc_norm.head()
```

범죄명 장소	강간.추행	강도	살인	절도	폭력
교통수단	0.324718	0.000000	0.000000	0.021027	0.008415
금융기관	0.000940	0.011494	0.015385	0.049738	0.001592
기타	1.000000	0.770115	1.000000	1.000000	1.000000
노상	0.463346	1.000000	0.338462	0.429235	0.929990
단독주택	0.185620	0.172414	0.461538	0.103110	0.135661

In [125]:

```
crime_loc_norm[ '종합' ] = np.mean(crime_loc_norm, axis=1)
crime_loc_norm.head( )
```

범죄명	강간.추행	강도	살인	절도	폭력	종합
장소						
교통수단	0.324718	0.000000	0.000000	0.021027	0.008415	0.070832
금융기관	0.000940	0.011494	0.015385	0.049738	0.001592	0.015830
기타	1.000000	0.770115	1.000000	1.000000	1.000000	0.954023
노상	0.463346	1.000000	0.338462	0.429235	0.929990	0.632207
단독주택	0.185620	0.172414	0.461538	0.103110	0.135661	0.211669

```
In [126]: crime_loc_norm_sort = crime_loc_norm.sort_values(by='종합', ascending=False)

plt.figure(figsize = (10,10))
sns.heatmap(crime_loc_norm_sort, annot=True, fmt='f', linewidths=.5, cmap='RdPu' )
plt.title('범죄와 발생 장소')
plt.show()
```

범죄와 발생 장소

