



## **SESSION # 06**

데이터 시각화, 데이터 다루기

By Team 3  
@ Mu, Fred, Justin, Chloe  
Date\_2017.08.05

# CONTENTS

---

01 Intro

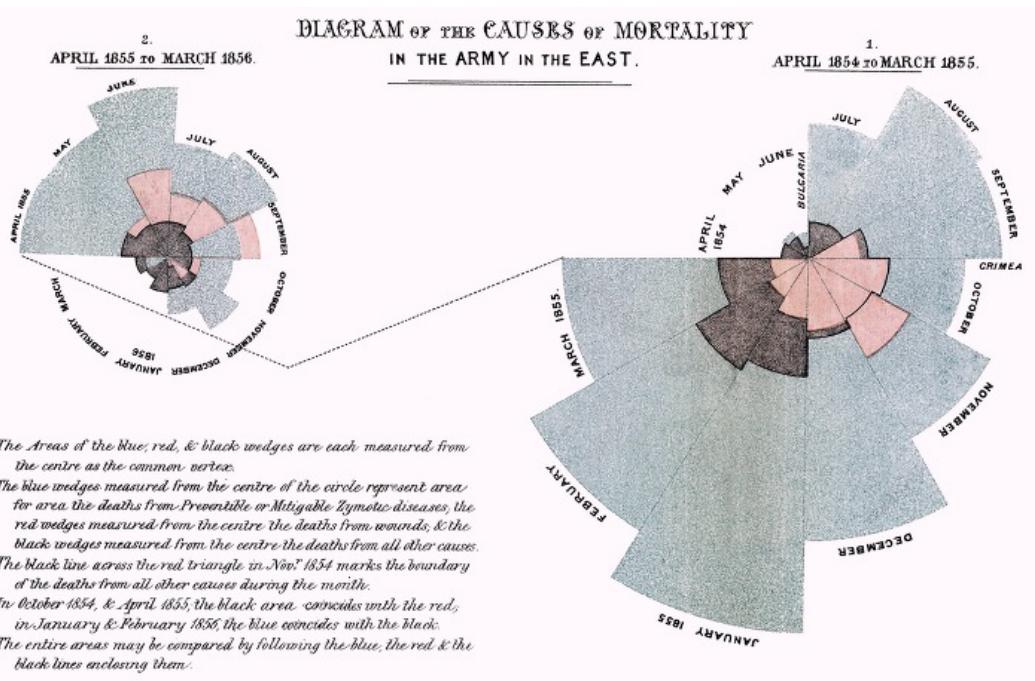
02 시작화 Basic Tutorial

03 데이터 다루기

04 데이터 시작화 고급기술

# 01 Intro \_ 데이터 시각화의 목적

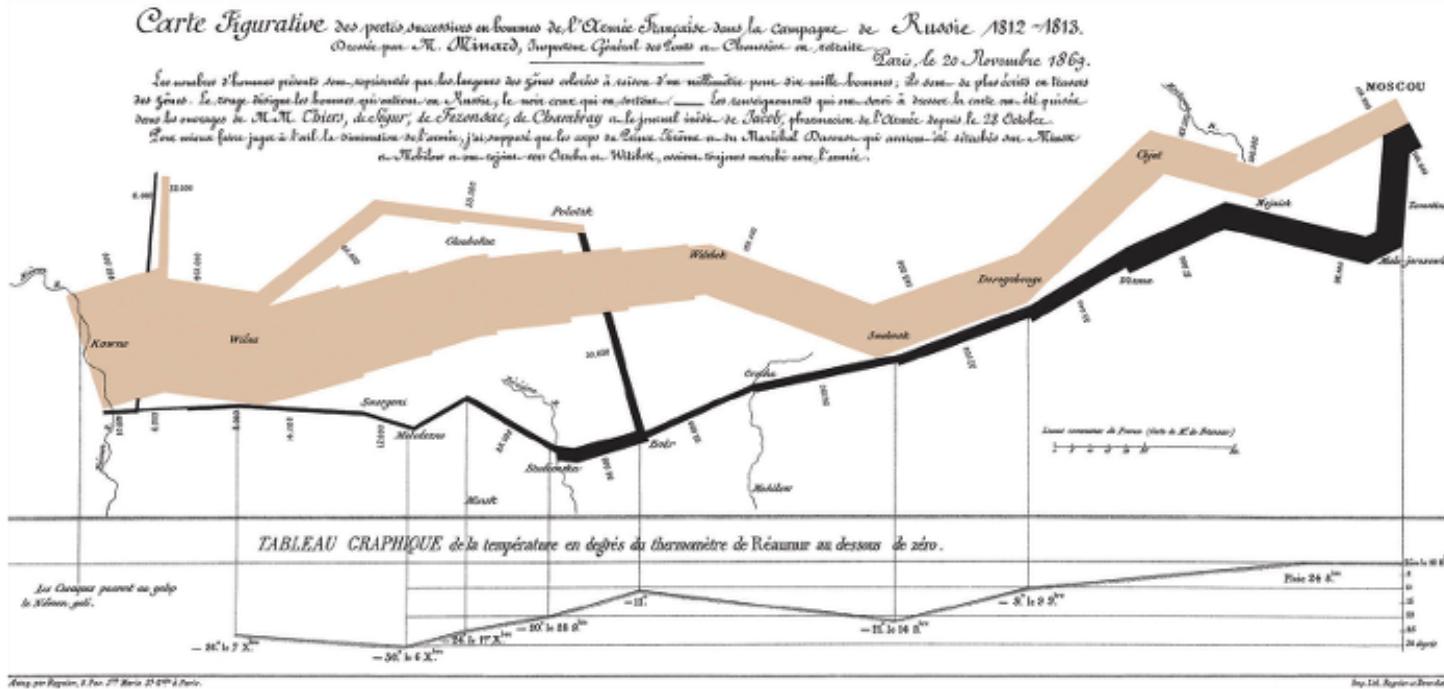
## 설득



나이팅게일 Florence Nightingale | 폴라그래프(1858년)

# 01 Intro \_ 데이터 시각화의 목적

## 설득

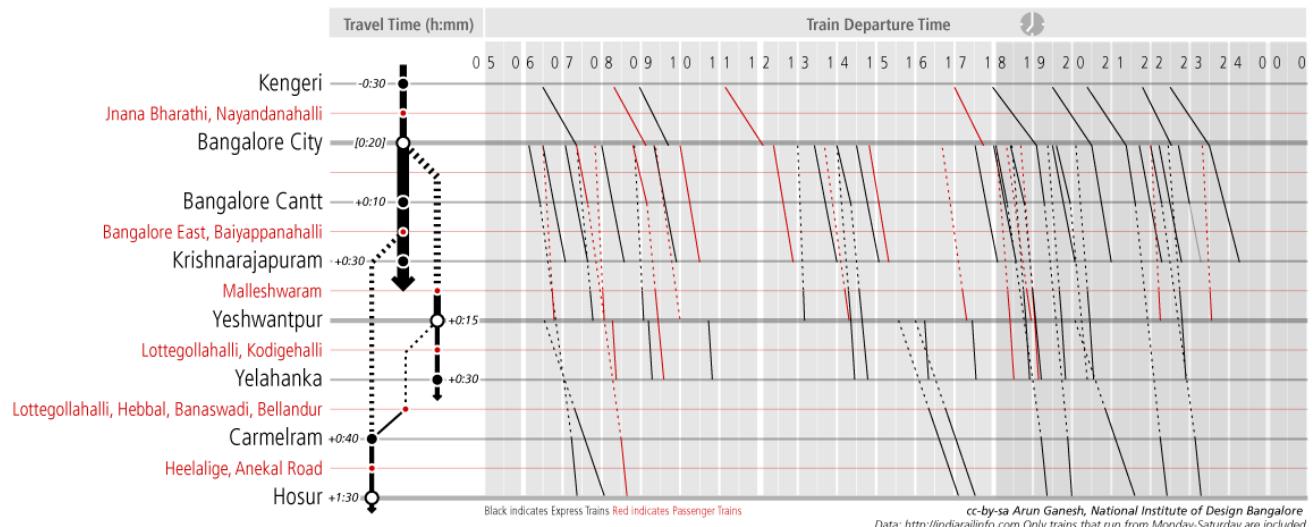


샤를 조셉 미나르  
 Charles Joseph Minard |  
 나폴레옹의 행군(1869  
 년)

# O1 Intro \_ 데이터 시각화의 목적

## 사실 확인

Frequency Chart of Outbound Trains from Bangalore Area



Bangalore Outbound Trains Frequency Chart

19 April 2010 (

[PlaneMad](#)

)

This is a work of a student or faculty from the [National Institute of Design](#)(India), 2017

# O1 Intro

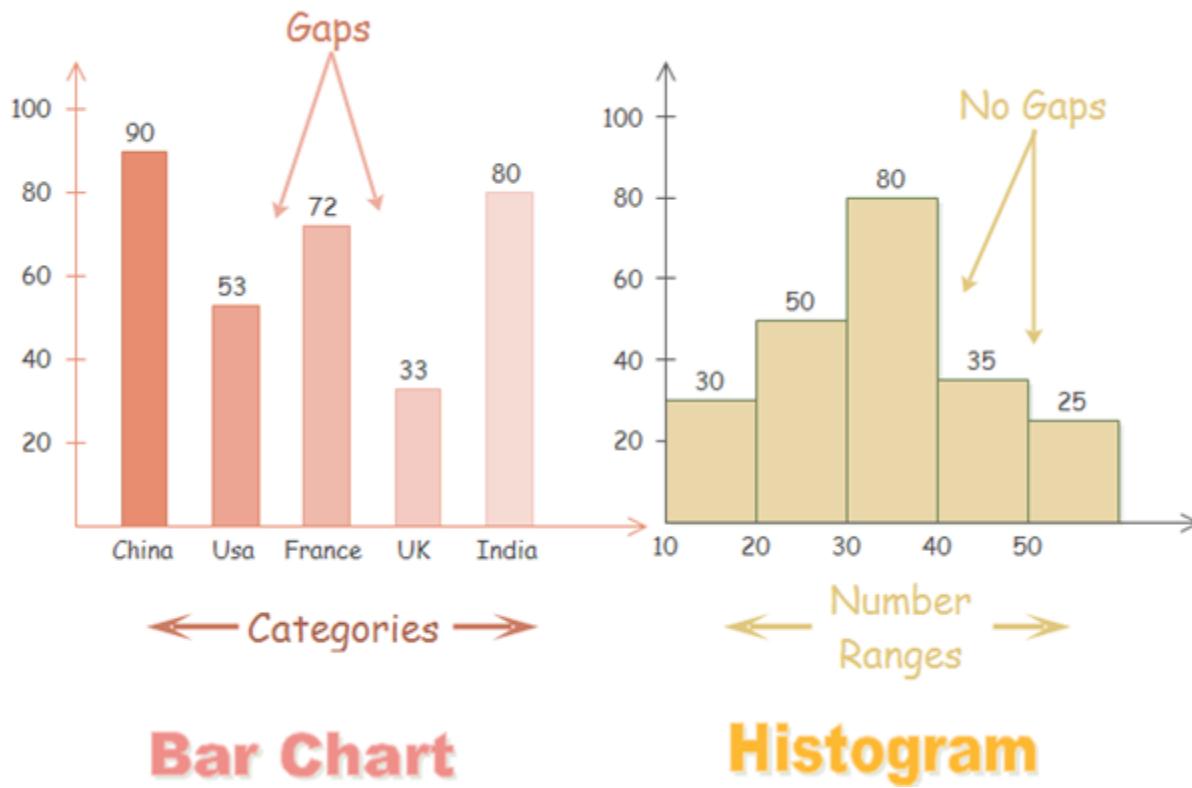
## 정보 시각화 종류

[https://www.slideshare.net/neofuture/sds-2?next\\_slideshow=1](https://www.slideshare.net/neofuture/sds-2?next_slideshow=1)

정보 시각화 방법				
시간 시각화	분포 시각화	관계 시각화	비교 시각화	공간 시각화
막대그래프 (Bar graph)  누적 막대그래프 (Stacked Bar graph)  점그래프 (Point graph)	파이차트 (Pie chart)  도넛 차트 (Donut chart)  트리맵 (Tree map)  누적 연속 그래프 (Cumulative continuous graph)	스캐터 플롯 (Scatter plot)  버블 차트 (Bubble chart)  히스토그램 (Histogram)	히트맵 (heat map)  체르노프페이스 (Chernoff face)  별그래프 (Star graph)  평행 좌표계 (Parallel coordinate system)  다차원 척도법 (Multi-dimensional scaling)	지도 매핑 (Dataviz on map)

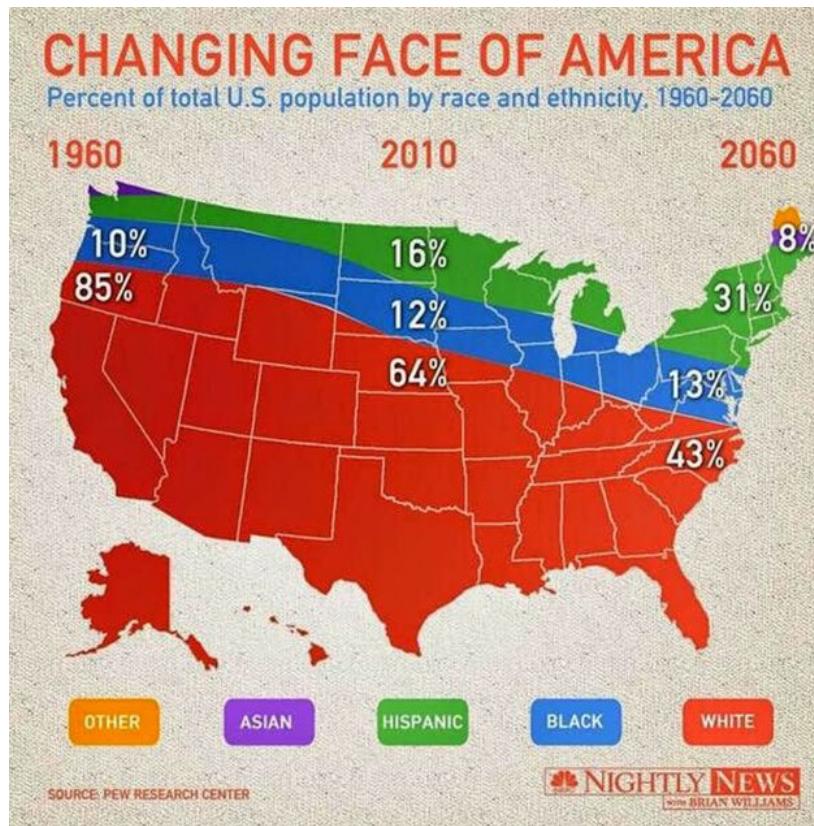
# O1 Intro

## 막대그래프와 히스토그램 차이



# O1 Intro

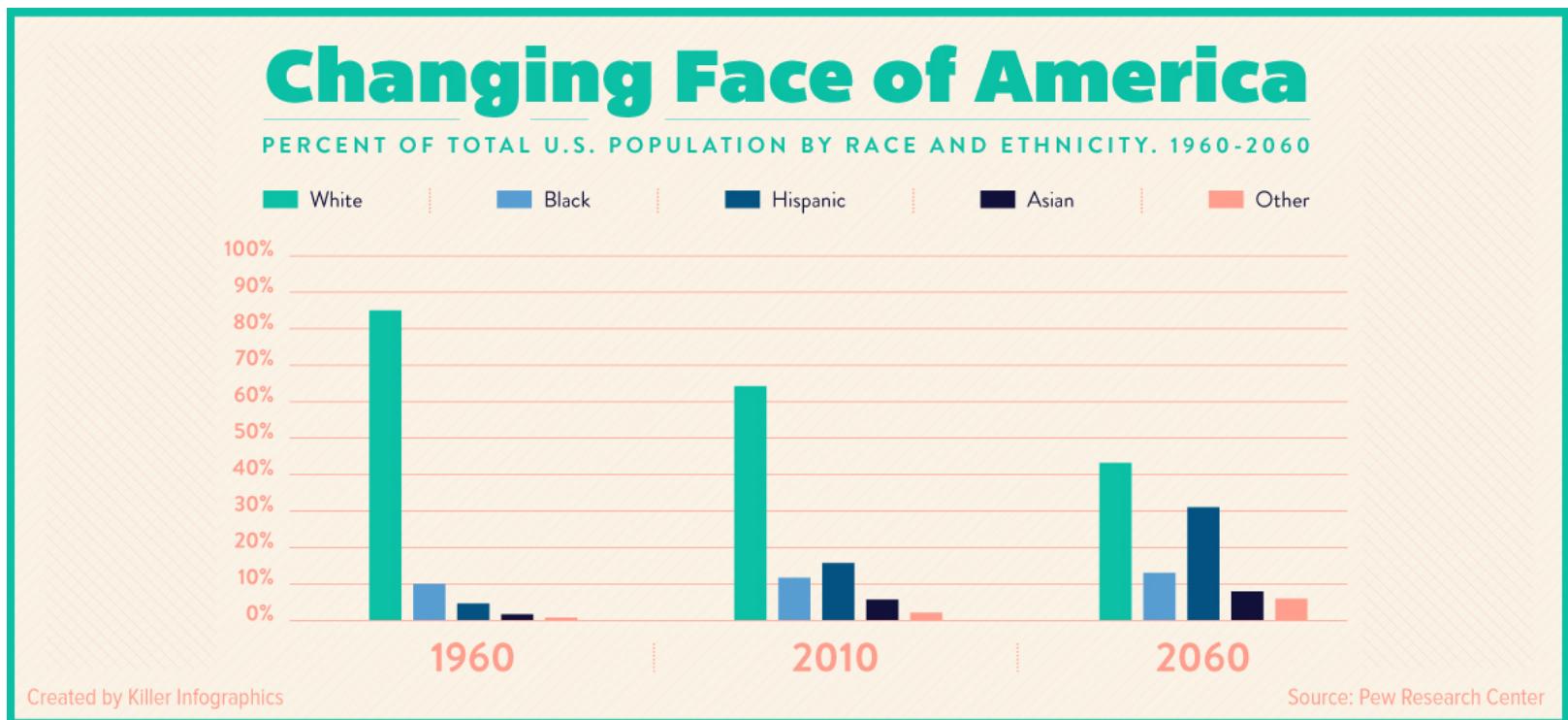
## 데이터시각화에서 고민할 것 (1)



하나의 디자인에 너무 많은 정보를 담으려하지 말아라

# O1 Intro

## 데이터시각화에서 고민할 것들 (1)



# O1 Intro

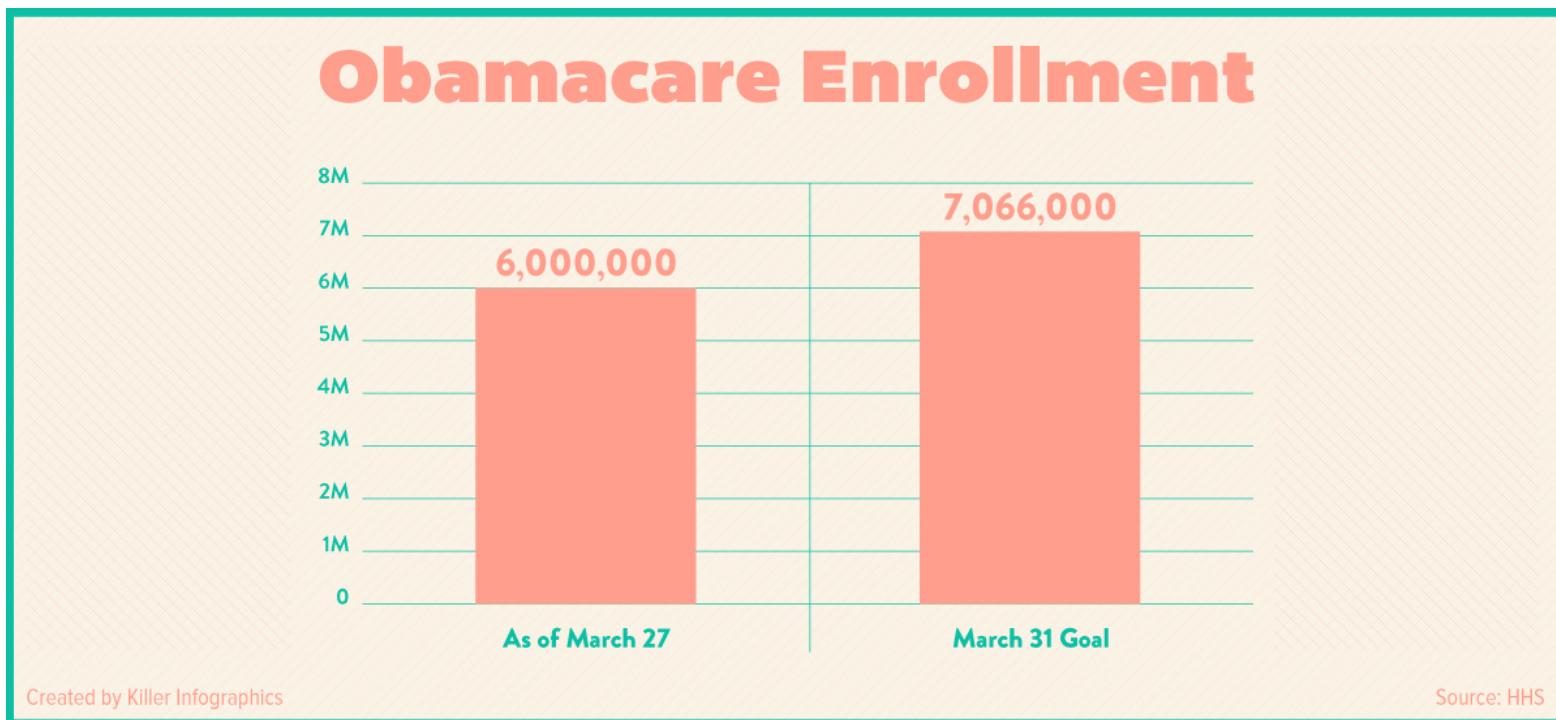
## 데이터시각화에서 고민할 것들 (2)



맥락 전달, 편향되지 않은 시각화

# O1 Intro

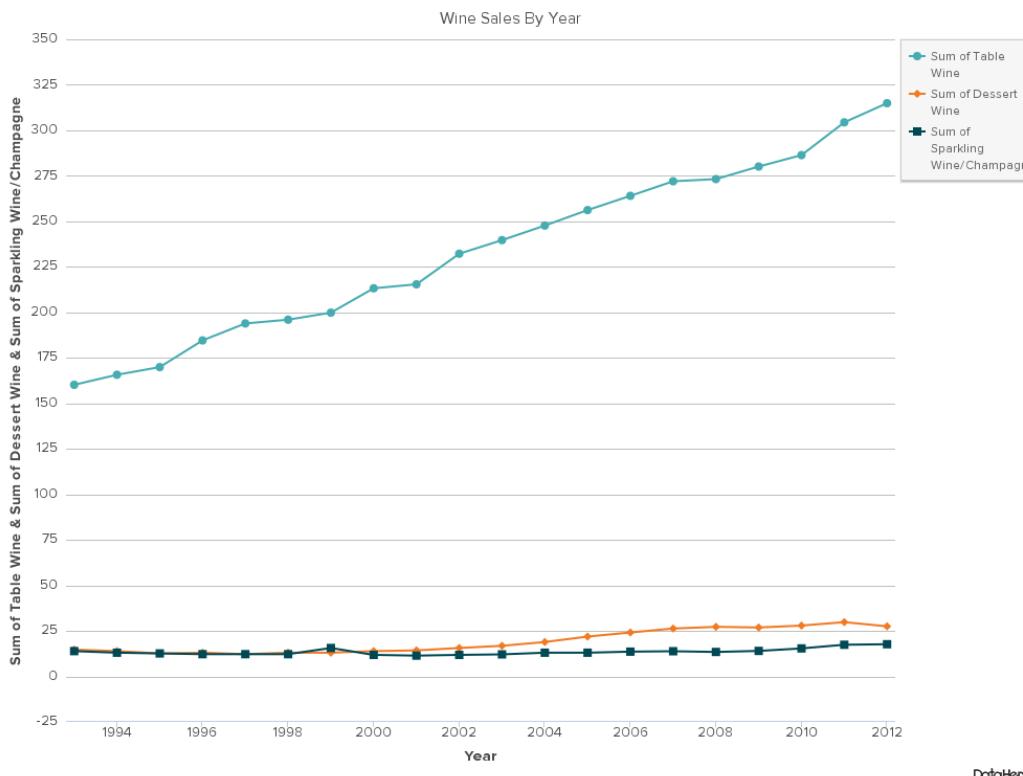
## 데이터시각화에서 고민할 것들 (2)



# O1 Intro

## 데이터시각화에서 고민할 것들 (3)

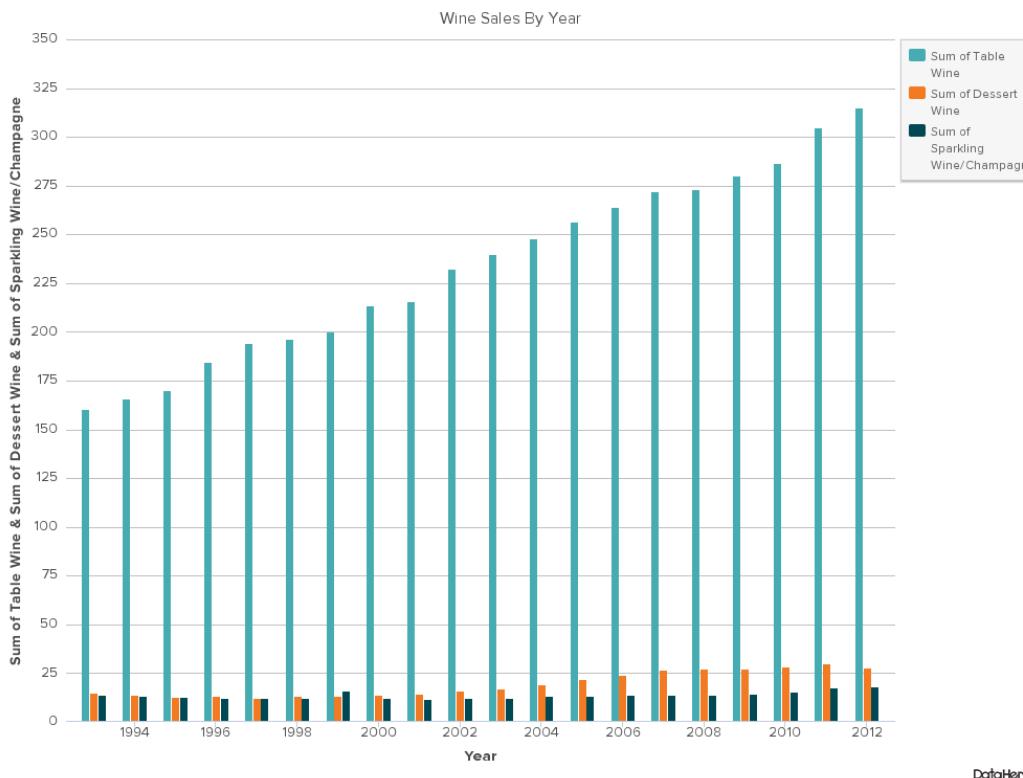
<https://datahero.com/blog/2013/08/06/line-or-bar-graph/>



## 선그래프 vs 막대그래프

# O1 Intro

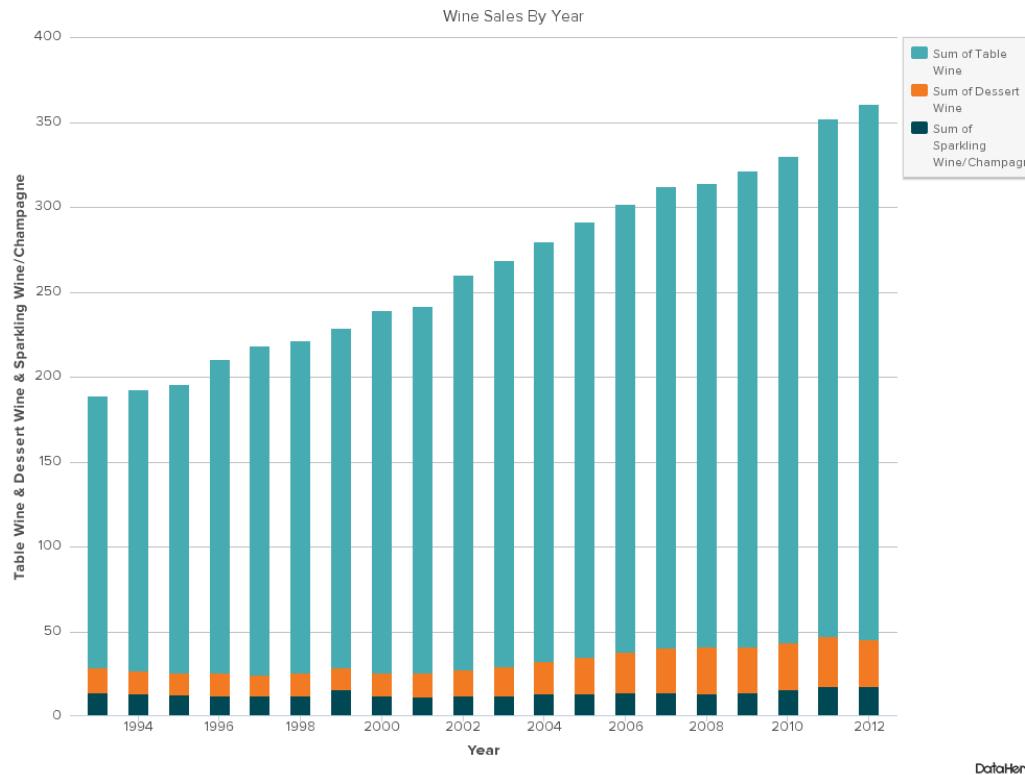
## 데이터시각화에서 고민할 것들 (3)



그룹핑된 막대그래프 (grouped)

# O1 Intro

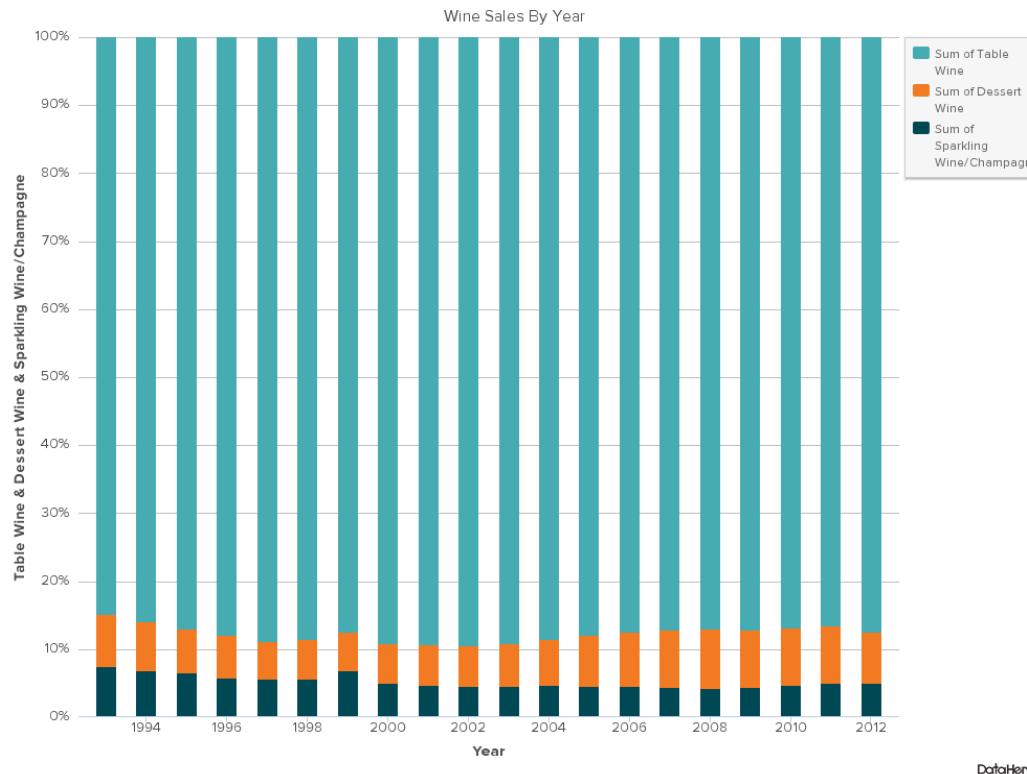
## 데이터시각화에서 고민할 것들 (3)



쌓인 막대그래프 (stacked)

# O1 Intro

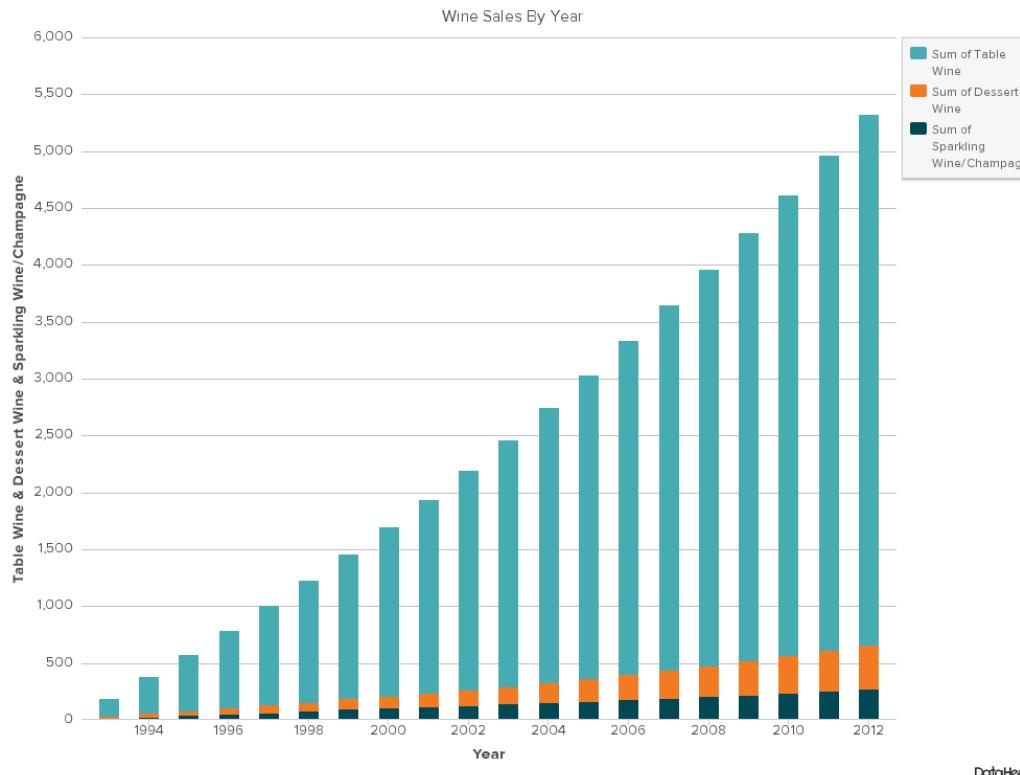
## 데이터시각화에서 고민할 것들 (3)



비율화된 막대그래프 (percentage)

# O1 Intro

## 데이터시각화에서 고민할 것들 (3)



누적 막대그래프 (cumulative)

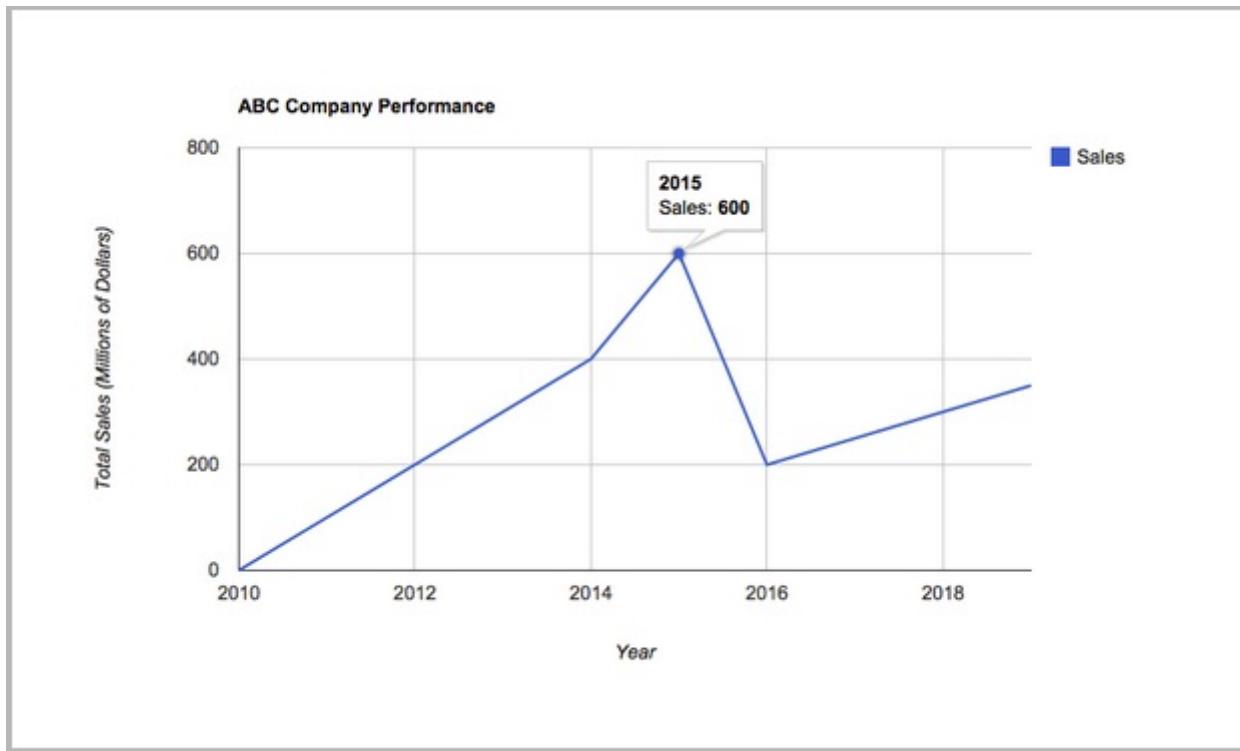
# O1 Intro \_ 퀴즈

## ■ 1차원 데이터일 때. 언제 어떤 선그래프/막대 그래프?

1. 다양한 카테고리가 시간이 변화함에 따라 변하는 1차원 데이터를 갖고 있을 때
2. 특정 카테고리에서 시간별로 변동량을 확인하고 싶을 때
3. 세 카테고리를 합친 전체 판매량의 변화를 보고 싶을 때
4. 전체에서 카테고리별로 비중의 변화를 확인하고 싶을 때
5. 예전부터 현재까지 세 카테고리를 합친 누적 판매량이 얼마나 되는지 확인하고 싶을 때

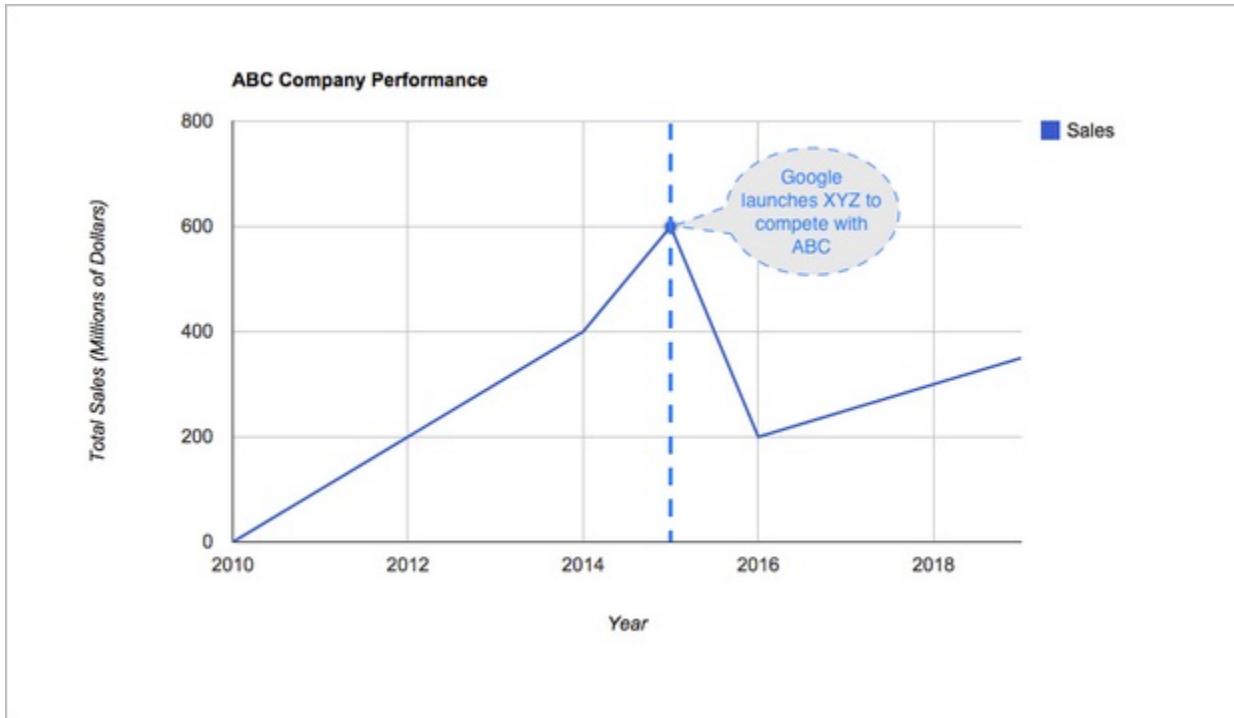
# O1 Intro

## 데이터시각화에서 고민할 것들 (4)



# O1 Intro

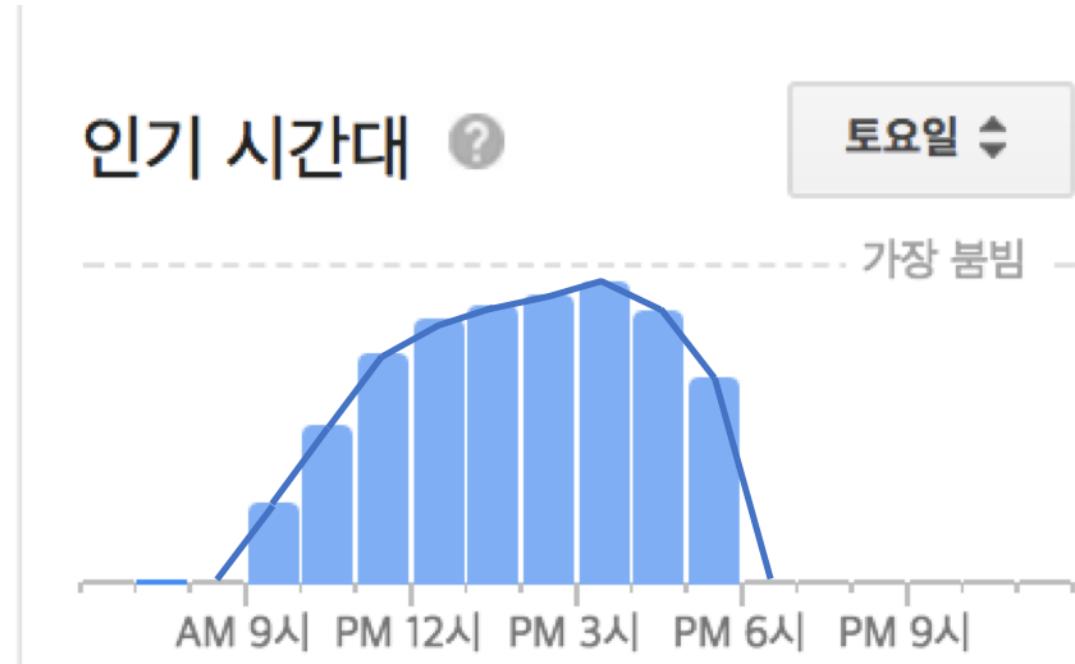
## 데이터시각화에서 고민할 것들 (4)



적절한 이유를 시각화 안에 삽입하여 이해를 돋는다.

# O1 Intro

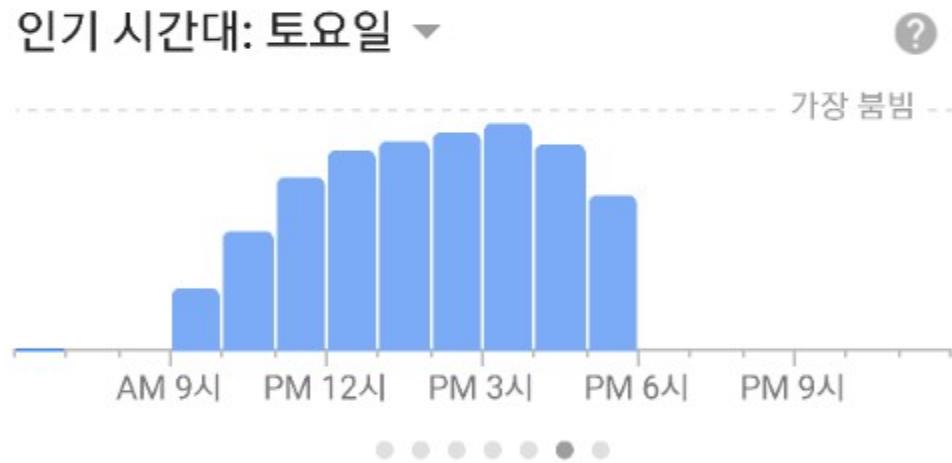
## ■ 데이터시각화에서 고민할 것들 (5)



꺾은선으로 표현한다면 주의할 점

# O1 Intro

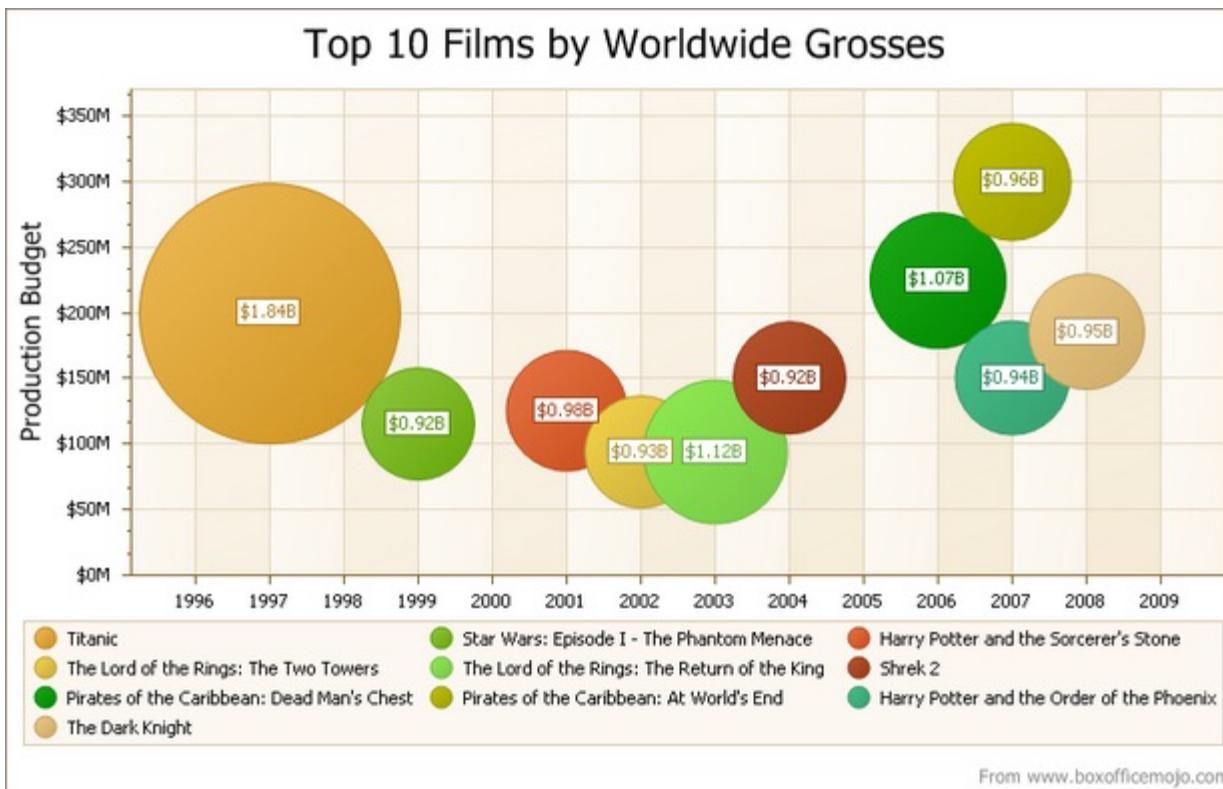
## ■ 데이터시각화에서 고민할 것들 (5)



막대그래프가 낫다

# O1 Intro

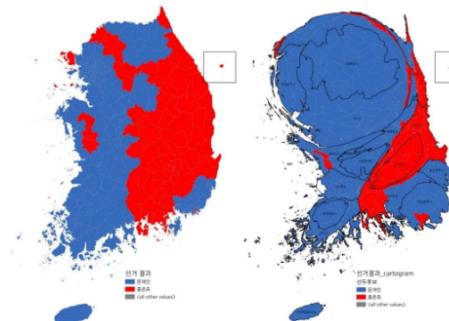
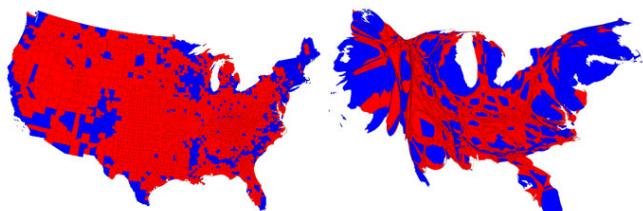
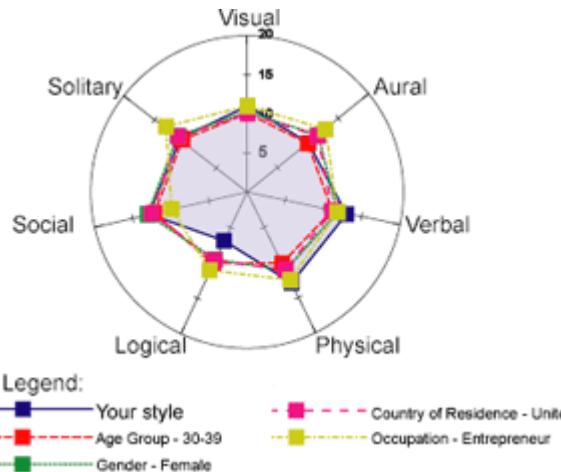
## 데이터시각화에서 고민할 것들 (6)



버블 차트 사용시 많이 저지르는 실수

# O1 Intro

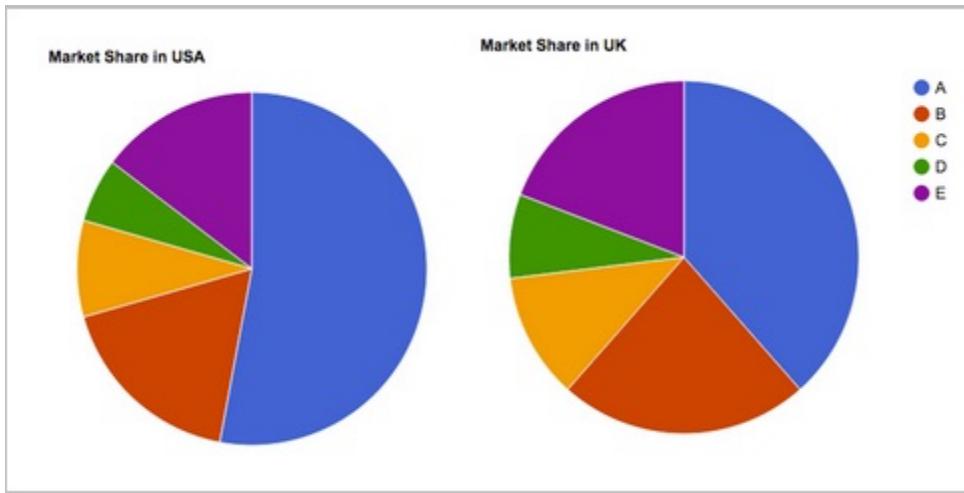
## 데이터시각화에서 고민할 것들 (7)



면을 색칠을 하면 양을 나타내는 듯한 착각을하게 한다.

# O1 Intro

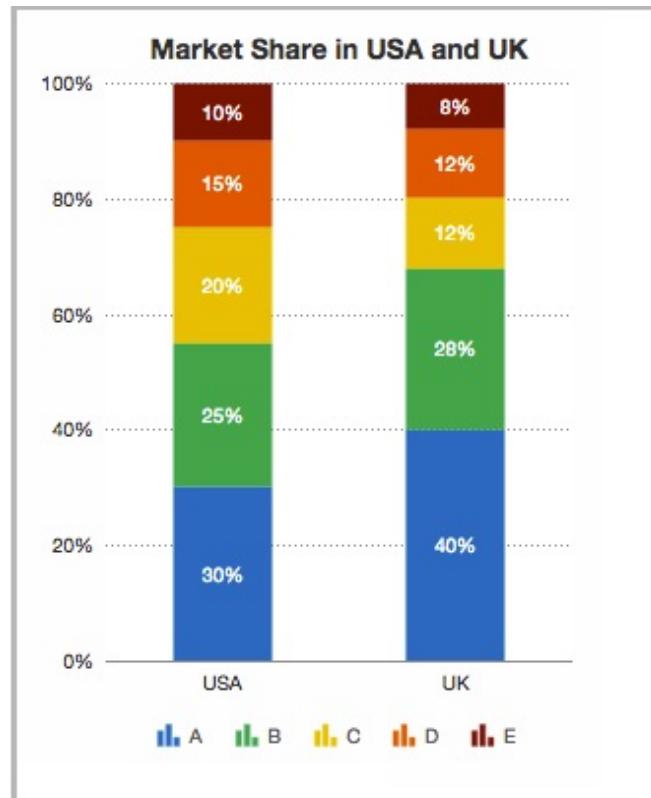
## 데이터시각화에서 고민할 것들 (8)



비교가 한눈에 들어오게 하라

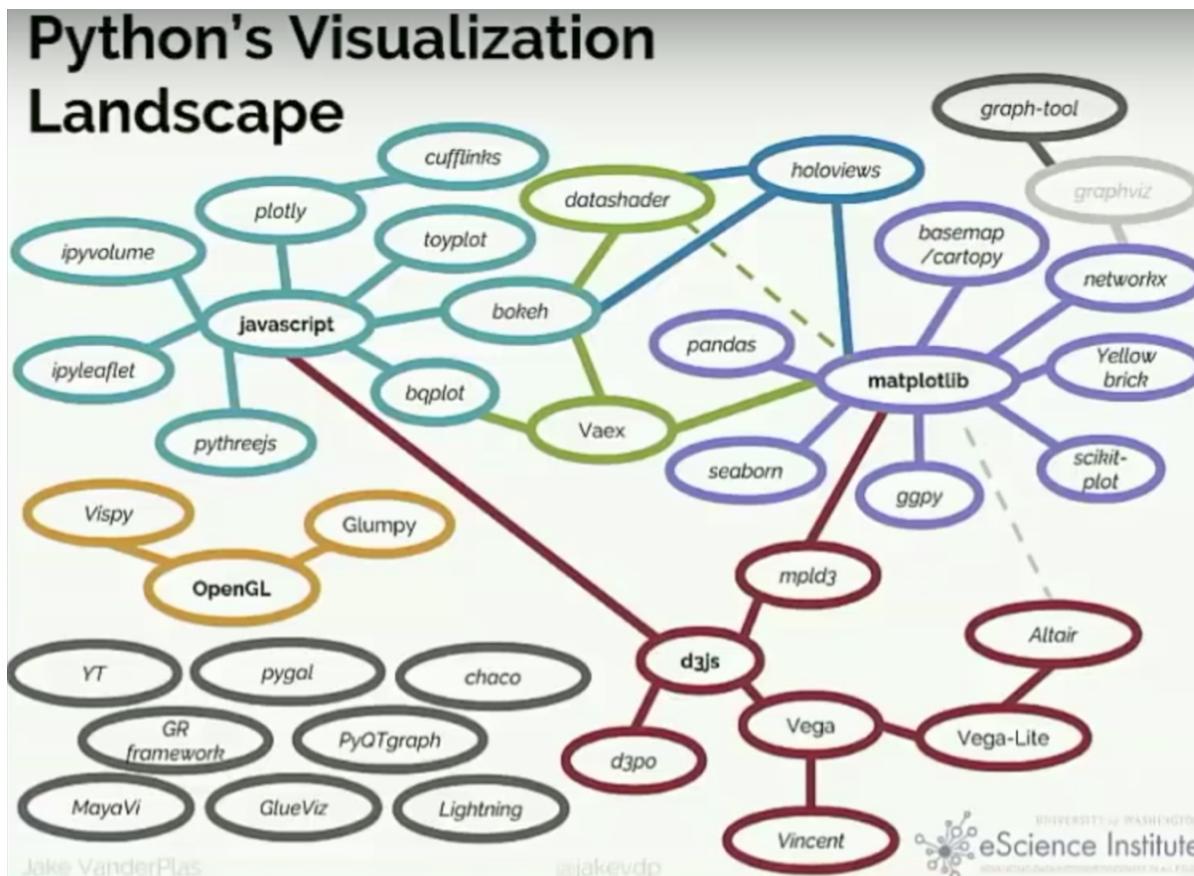
# O1 Intro \_ 퀴즈

## 전 페이지의 시각화와 차이점? (8)



# 01 Intro

## 파이썬 데이터 시각화 지형



## 02 시각화 Basic Tutorial

### ① matplotlib

# ① matplotlib

Matplotlib 소개가 끝난 뒤에는 바로 실습과 함께 진행됩니다.  
슬랙에 올린 HR\_comma.sep.csv 를 미리 다운로드 해주세요!

# 02 시각화 Basic Tutorial

## ① matplotlib

### matplotlib 소개

- : 파이썬에서 자료를 차트(chart)나 플롯(plot)으로 시각화(visualization)하는 패키지
- : 웹을 위한 복잡하고 interactive한 시각화에는 적절하지 X
- : 저수준의 api를 사용한 다양한 시각화 기능도 제공
- : line plot, bar chart, histogram, scatter plot, box plot, contour plot, surface plot etc.

## 02 시각화 Basic Tutorial

### ① matplotlib

#### matplotlib.pyplot 모듈

##### matplotlib.pyplot is

a collection of command style functions  
that make matplotlib work like MATLAB.  
Each pyplot function makes some change to a figure

: e.g., creates a figure, creates a plotting area in a figure,  
plots some lines in a plotting area, decorates the plot with labels, etc.

# 02 시각화 Basic Tutorial

## ① matplotlib

### matplotlib.pyplot 모듈

- 시각화를 단계별로 간편하게 만들 수 있게 함.
- from matplotlib import pyplot as plt (항상 상단에 입력!)
- (ex) >>> plt.plot()  
>>> plt.bar()
- 관련 링크  
: [http://matplotlib.org/api/pyplot\\_api.html#matplotlib.pyplot.plot](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot)

## 02 시각화 Basic Tutorial ① matplotlib

### matplotlib.pyplot 에서 그래프 속성 변경/추가하기

#### : (1) argument

```
>>> plt.plot(x, y, color='', marker='', linestyle='')
```

(plt.plot은 선그래프 입력 함수, 각 그래프 별로 argument 조금씩 다름)

#### : (2) commander

```
>>> plt.plot(x, y, color='', marker='', linestyle='')
```

```
>>> plt.xticks(x, objects)
```

```
>>> plt.grid(false)
```

```
>>> plt.show()
```

Commander	Function	입력 표현
plt.show()	그래프 나타내기	>>> plt.show()
plt.savefig()	저장하기	>>> plt.savefig()
plt.xlabel("x축 label") plt.ylabel("y축 label")	축 label	>>> plt.xlabel("# of students")
plt.xticks(인덱스, 변량명) plt.yticks(인덱스, 변량명)	막대별 label	>>> plt.xticks(x, object)
plt.axis()	축 조절	>>> plt.axis([xmin, xmax, ymin, ymax]) >>> plt.axis('off') >>> plt.axis('equal')
plt.grid(T/F)	격자눈금	>>> plt.grid(true)
plt.legend(위치)	범주	>>> plt.legend(loc='upper left') >>> plt.legend(loc='top center')
plt.annotate()	포인트 레이블	

## 02 시각화 Basic Tutorial ① matplotlib

### \* **HR\_comma\_sep.csv**

**Why are our best and most experienced employees leaving prematurely? Have fun with this database and try to predict which valuable employees will leave next.**

Fields in the dataset include:

Satisfaction Level / Last evaluation / Number of projects  
/ Average monthly hours / Time spent at the company  
/ Whether they have had a work accident  
/ Whether they have had a promotion in the last 5 years  
/ Departments / Salary/ Whether the employee has left

url \_ <https://www.kaggle.com/zhenjiaqin/analyse-hr-comma-sep-data/data>

# 02 시각화 Basic Tutorial

## ① matplotlib

### 실습

우리는 오늘  
외부데이터 'HR\_comma\_sep.csv'로  
막대그래프, 선그래프, 산점도 등을 그려봅니다!

# 02 시각화 Basic Tutorial

## ① matplotlib

### 실습

우리는 오늘  
외부데이터 'HR\_comma\_sep.csv'로  
막대그래프, 선그래프, 산점도 등을 그려봅니다!

-> 지금 해주세요!

- ① HR\_comma\_sep.csv 불러오기
- ② f.readlines를 활용하여  
'row'들로 이루어진 list 만들기
- ③ 하나의 column을 정해 list 만들기 (**이산형!**)
- ④ 첫 번째 인수 조심!

## 02 시각화 Basic Tutorial ① matplotlib

```
f=open("HR_comma_sep.csv", 'r', encoding = "UTF-8")
lines=f.readlines()
print(lines)
```

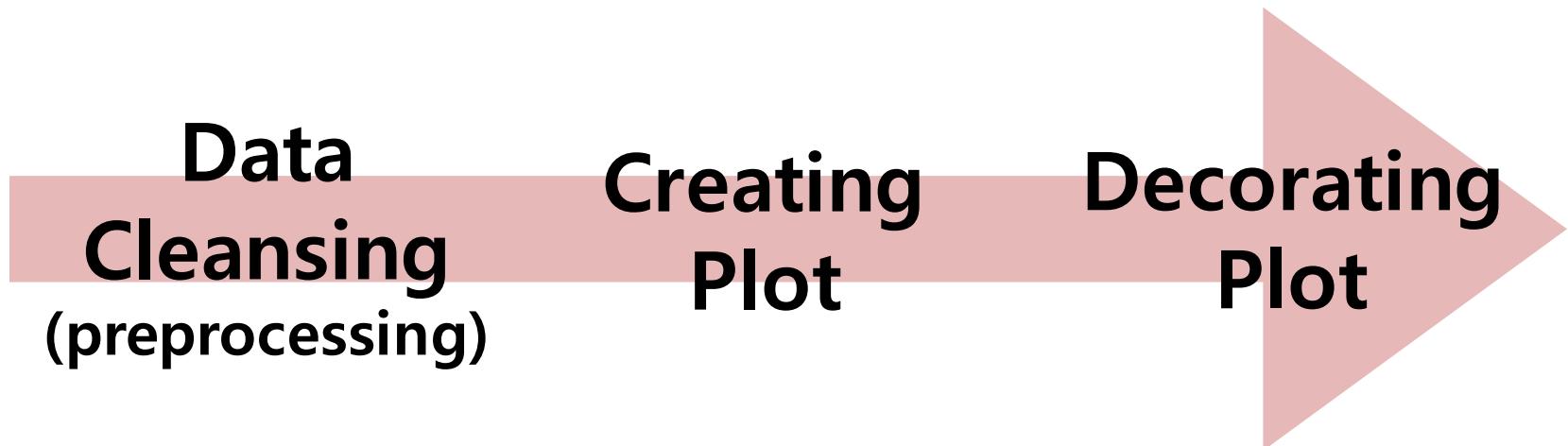
--- csv파일에서 필요한 자료 추출하기 ---

```
dept=[]
for line in lines: # lines는 line으로 이루어진 list
    new_element = line.split(',') [8] # 각 line을 ','로 쪼개고 그것 중 8번째(dept)
    dept.append(new_element) # append는 list의 함수
del dept[0] # dept의 첫번째 인수는 'sales'
print(dept)
```

# 02 시각화 Basic Tutorial

## ① matplotlib

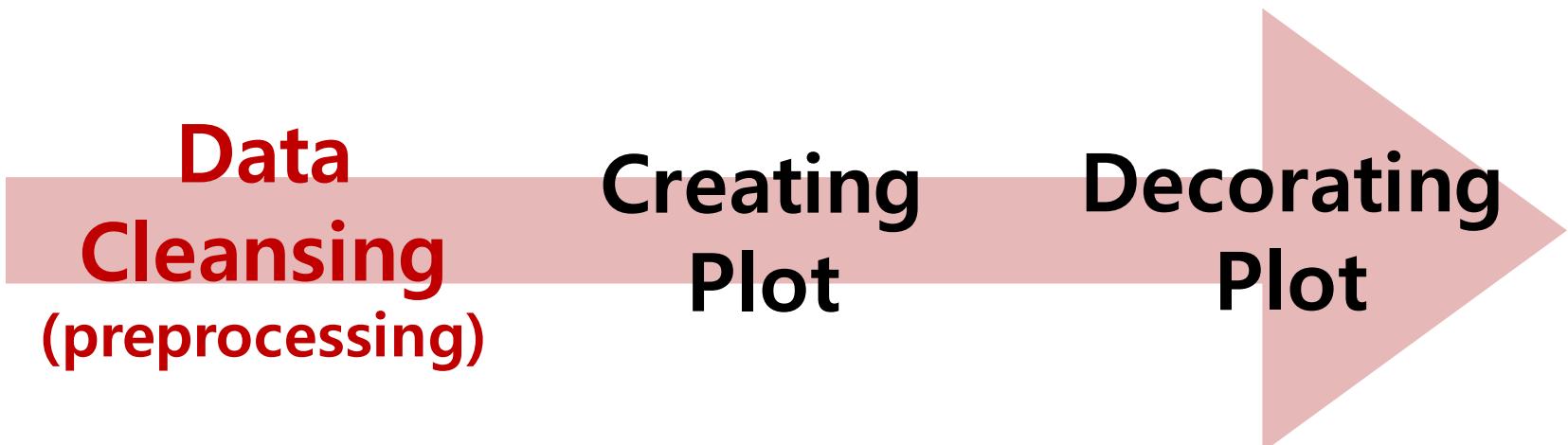
### 시각화 process



# 02 시각화 Basic Tutorial

## ① matplotlib

### 시각화 process



## 02 시각화 Basic Tutorial    ② Bar charts

# ② Bar charts

막대 그래프, 히스토그램

# 02 시각화 Basic Tutorial

## ② Bar charts

둘의 차이점?

막대 그래프, 히스토그램

# 02 시각화 Basic Tutorial

## ② Bar charts

Category  
이산적인 변량

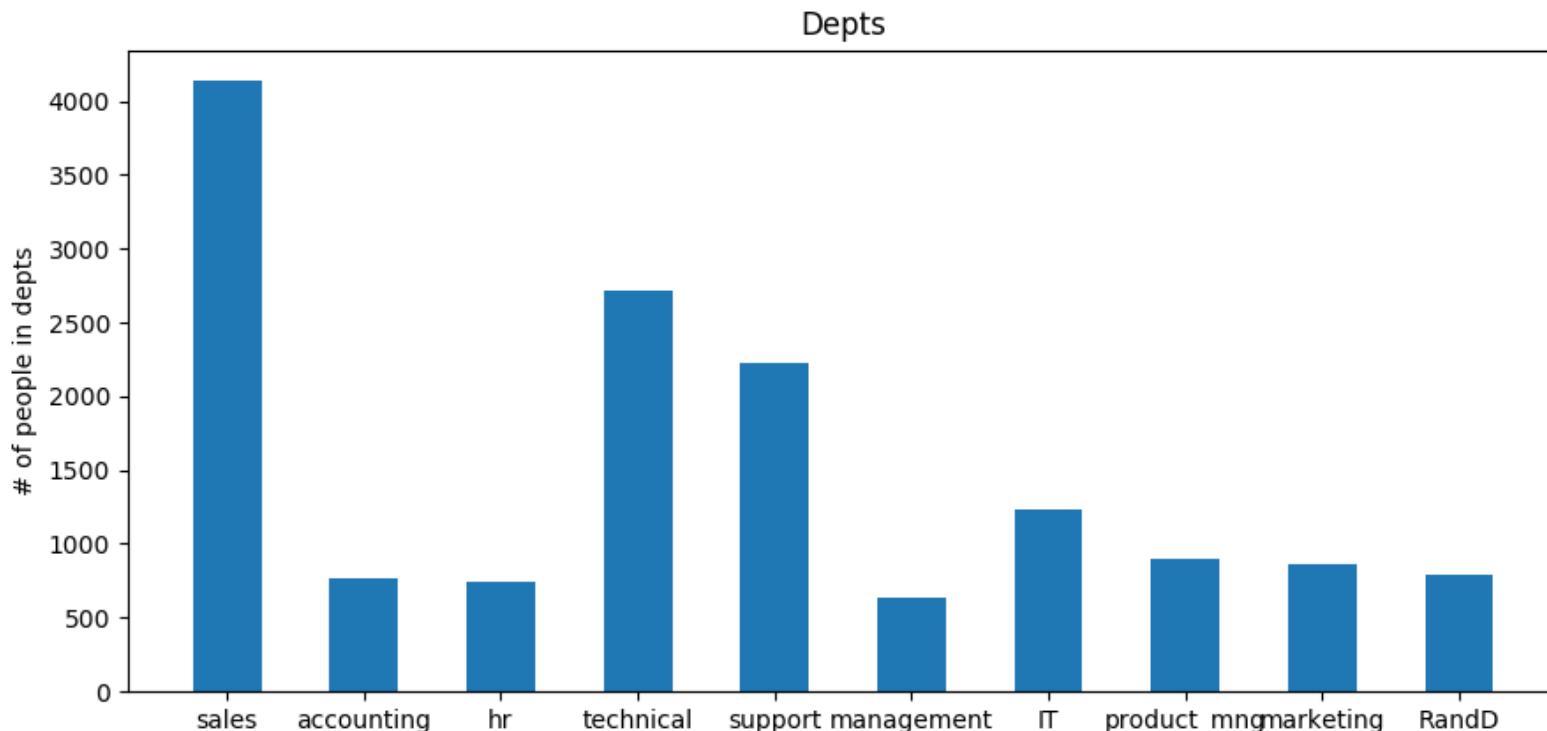
막대 그래프, 히스토그램

Class  
연속적인 변량

# 02 시각화 Basic Tutorial

## ② Bar charts

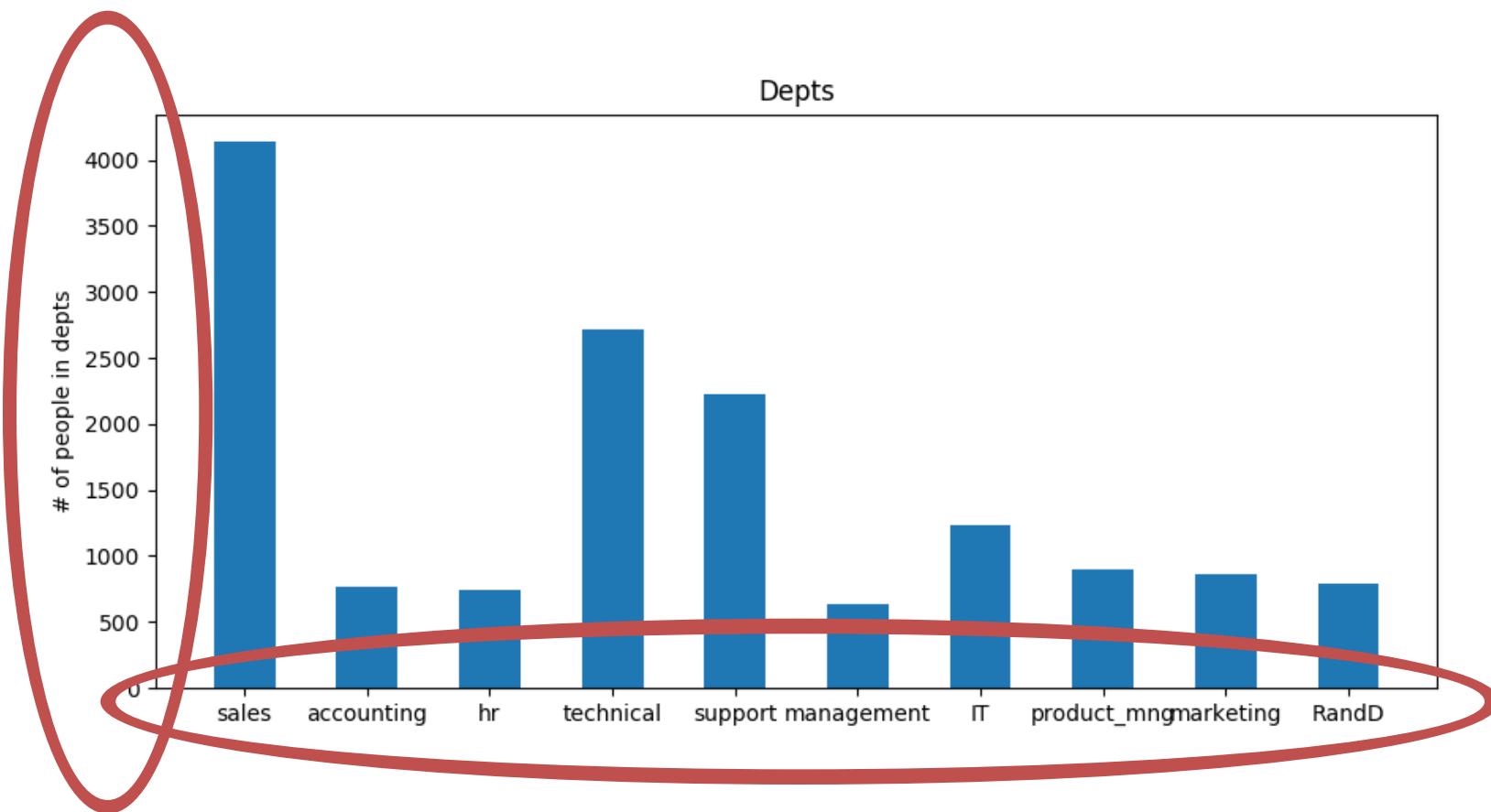
### 막대그래프 (1) data cleansing



# 02 시각화 Basic Tutorial

## ② Bar charts

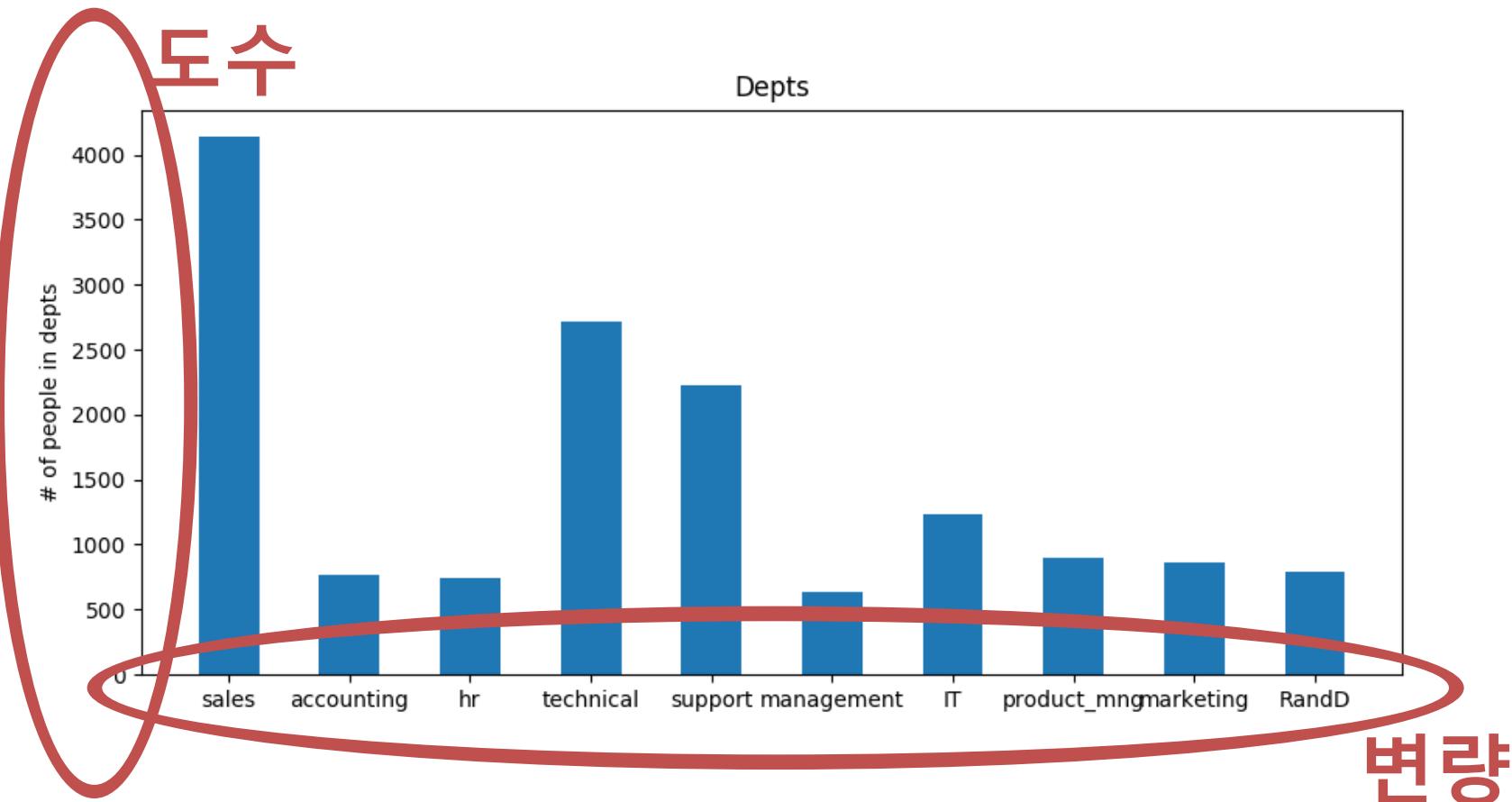
### 막대그래프 (1) data cleansing



# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (1) data cleansing



# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (1) data cleansing

한번 고민해봅시다!

Q. 어떻게 변량과 도수를 추출해낼 수 있을까?!

## 02 시각화 Basic Tutorial ② Bar charts

### 막대그래프 (1) data cleansing

한번 고민해봅시다!

Q. 어떻게 변량과 도수를 추출해낼 수 있을까?!

⇒ 추출한 데이터(column data)에서  
어떤 값들이 존재하는지,  
각 값의 frequency는 어떠한지를 확인!

## 02 시각화 Basic Tutorial ② Bar charts

### 막대그래프 (1) data cleansing

한번 고민해봅시다!

Q. 어떻게 변량과 도수를 추출해낼 수 있을까?!

⇒ 추출한 데이터(column data)에서  
어떤 값들이 존재하는지,  
각 값의 frequency는 어떠한지를 확인!

Counter()  
함수 이용!

# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (1) data cleansing Counter() – 변량, 도수 추출

```
1 from collections import Counter
2
3 my_list = ['a', 'b', 'c', 'b', 'c', 'd', 'a']
4 my_Counter = Counter(my_list)
5 print(my_Counter)
6 # Counter({'a': 2, 'b': 2, 'c': 2, 'd': 1})
7
8 print(my_Counter['a'])
9 #2
```

# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (1) data cleansing Counter() – 변량, 도수 추출

```
1 from collections import Counter  
2  
3 my_list = ['a', 'b', 'c', 'b', 'c', 'd', 'a']  
4 my_Counter = Counter(my_list)  
5 print(my_Counter)  
6 # Counter({'a': 2, 'b': 2, 'c': 2, 'd': 1})  
7  
8 print(my_Counter['a'])  
9 #2
```

⇒ Dic 자료형과 형태 유사!  
⇒ Key, Value 값 이용  
(단, type이 dic은 아님)

# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (1) data cleansing Counter() – 변량, 도수 추출

```
1 from collections import Counter  
2  
3 my_list = ['a', 'b', 'c', 'b', 'c', 'd', 'a']  
4 my_Counter = Counter(my_list)  
5 print(my_Counter)  
6 # Counter({'a': 2, 'b': 2, 'c': 2, 'd': 1})  
7  
8 print(my_Counter['a'])  
9 #2
```

변량

도수

⇒ Dic 자료형과 형태 유사!  
⇒ Key, Value 값 이용  
(단, type이 dic은 아님)

# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (1) data cleansing Counter() – 변량, 도수 추출

```

1  from collections import Counter
2
3  my_list = ['a', 'b', 'c', 'b', 'c', 'd', 'a']
4  my_Counter = Counter(my_list)
5  print(my_Counter)
6  # Counter({'a': 2, 'b': 2, 'c': 2, 'd': 1})
7
8  print(my_Counter['a'])
9  #2

```

실습

막대그래프를  
그리기 위해  
적절한 자료로  
다듬어 보세요!

#### Hint !

dic 자료형에서 key값들의 리스트는 dic.keys() (value는 dic.values())

Python3 -> list를 만들기 위해 list(dic.keys()) 로 한번 더 묶어주기!

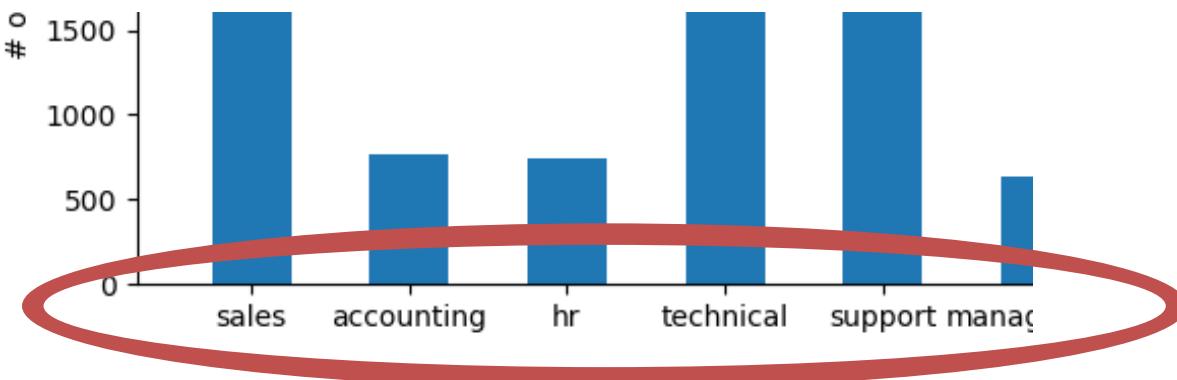
# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (2) Creating plot enumerate() – index와 변량

지금까지 x, y값을 구했음!

Plot을 그리기 위해서는 여기서 x에 좀 더 주목해야 함.



첫번째 막대의 label인  
'sales'는 나중에 따로  
지정해주는 것.

->  
즉, 입력 해야하는 x값은  
변량의 index  
Ex) sales의 index는 0

=> Sales의 y값이 아닌 첫 번째 변량의 y값이라고 인식

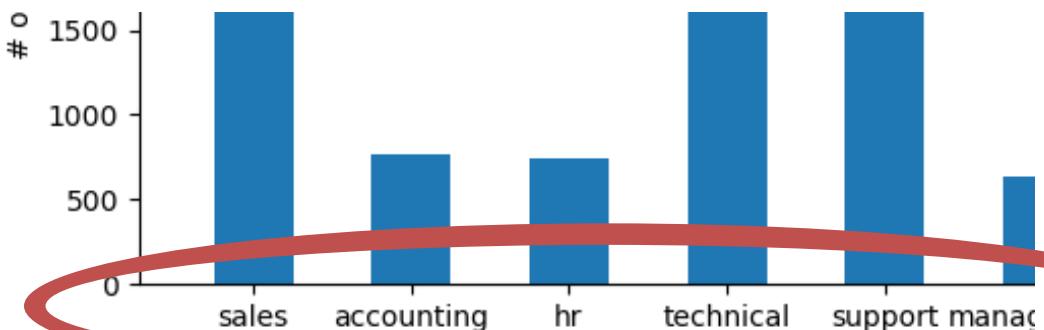
# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (2) Creating plot enumerate() – index와 변량

지금까지 x, y값을 구했음!

Plot을 그리기 위해서는 여기서 x에 좀 더 주목해야 함.



첫번째 막대의 label인  
'sales'는 나중에 따로  
지정해주는 것.

->  
즉, 앞과 같은  
변량 `enumerate()`  
함수 이용!

=> Sales의 y값이 아닌 첫 번째 변량의 y값이라고 인식

## 02 시각화 Basic Tutorial ② Bar charts

### 막대그래프 (2) Creating plot enumerate() – index와 변량

- **enumerate()**를 활용하여 x값 만들기!

*enumerate*는 "열거하다"라는 뜻이다.

이 함수는 순서가 있는 자료형(리스트, 튜플, 문자열)을  
입력으로 받아 인덱스 값을 포함하는 *enumerate* 객체를 리턴한다.

-----<점프 투 파이썬>

```
1 team_three= ["Mu", "Fred", "Justin", "Chloe"]
2 team_three_index=enumerate(team_three)
3
4 print(list(team_three_index))
5 # [(0, 'Mu'), (1, 'Fred'), (2, 'Justin'), (3, 'Chloe')]
6
```

# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (2) Creating plot enumerate() – index와 변량

*enumerate*는 "열거하다"라는 뜻이다.

이 함수는 순서가 있는 자료형(리스트, 튜플, 문자열)을  
입력으로 받아 인덱스 값을 포함하는 *enumerate* 객체를 리턴한다.

-----<점프 투 파이썬>

```

1 team_three= ["Mu", "Fred", "Justin", "Chloe"]
2 team_three_index=enumerate(team_three)
3
4 print(list(team_three_index))
5 # [(0, 'Mu'), (1, 'Fred'), (2, 'Justin'), (3, 'Chloe')]
6

```

실습

=> **enumerate()**를 활용하여 x값 만들기!

# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (2) Creating plot enumerate() – index와 변량

```
from collections import Counter # Counter는 리스트 내 각 elements 의 개수 알려줌  
num_dept=Counter(dept)  
print(num_dept) # Counter()는 dictionary 자료형과 형태 유사  
dept_keys=list(num_dept.keys())  
dept_values=list(num_dept.values())  
# python3는 a.keys() 또는 a.values()를 list로 만들 때 따로 list()로 묶어 줘야 함.  
print(dept_keys)  
print(dept_values)  
  
print(list(enumerate(dept_keys)))  
#[0, 'sales'), (1, 'accounting'), (2, 'hr'), (3, 'technical'), (4, 'support'), (5, 'management'), (6, 'IT'),  
# (7, 'product_mng'), (8, 'marketing'), (9, 'RandD')]
```

## 02 시각화 Basic Tutorial ② Bar charts

### 막대그래프 (2) Creating plot enumerate() – index와 변량

실습

이제 가공한 데이터로 막대그래프를 그려보세요!

Hint!

```
from matplotlib import pyplot as plt
```

```
>>> plt.bar(x, y, width)
>>> plt.show( )  <- 항상 마지막에 입력!
```

# 02 시각화 Basic Tutorial

## ② Bar charts

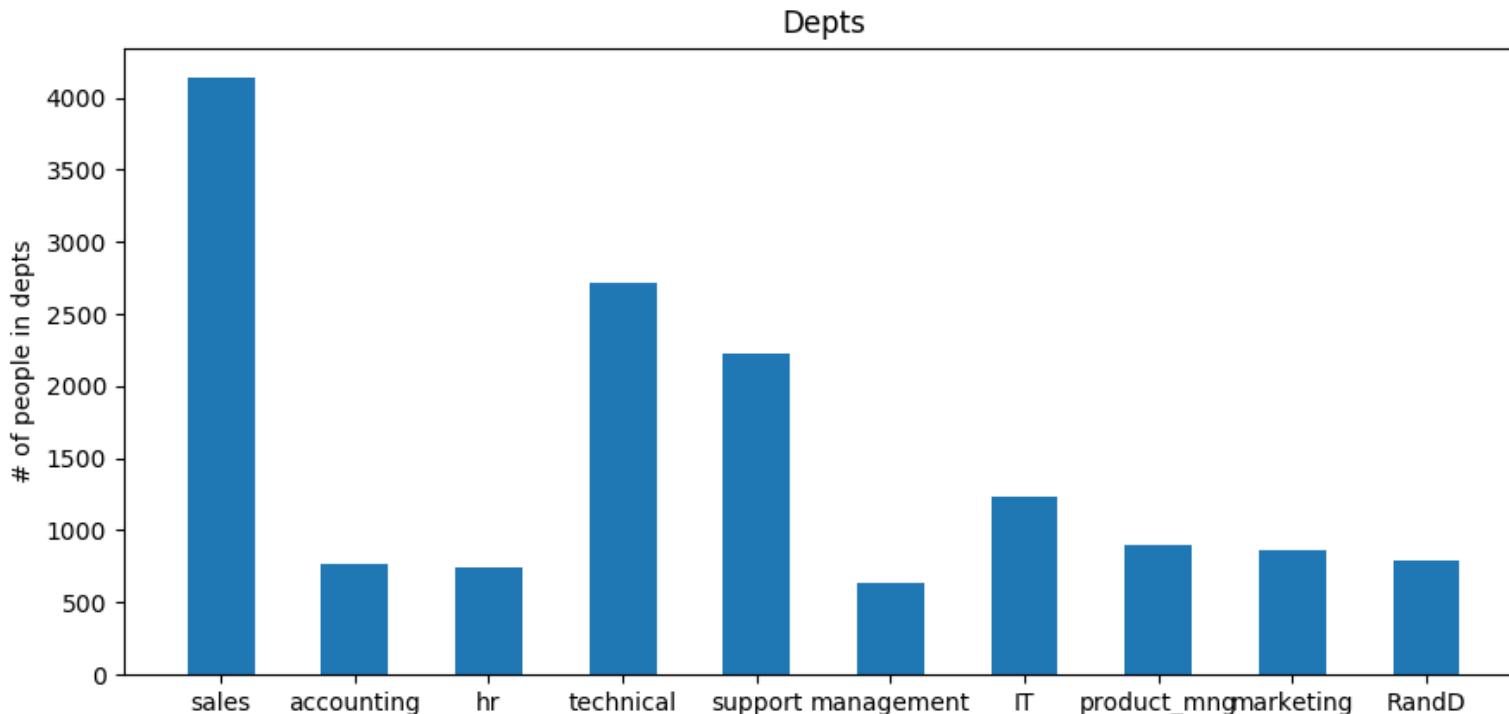
### 막대그래프 (2) Creating plot enumerate() – index와 변량

```
from matplotlib import pyplot as plt
objects = dept_keys
x = [i+0.3 for i, _ in enumerate(dept_keys)] #enumerate()는 index와 인수 쌍의 나열
freq = dept_values
plt.bar(x, freq, 0.5) # 막대 그래프 그리기! plt.bar(x값, y값, width)
```

# 02 시각화 Basic Tutorial

## ② Bar charts

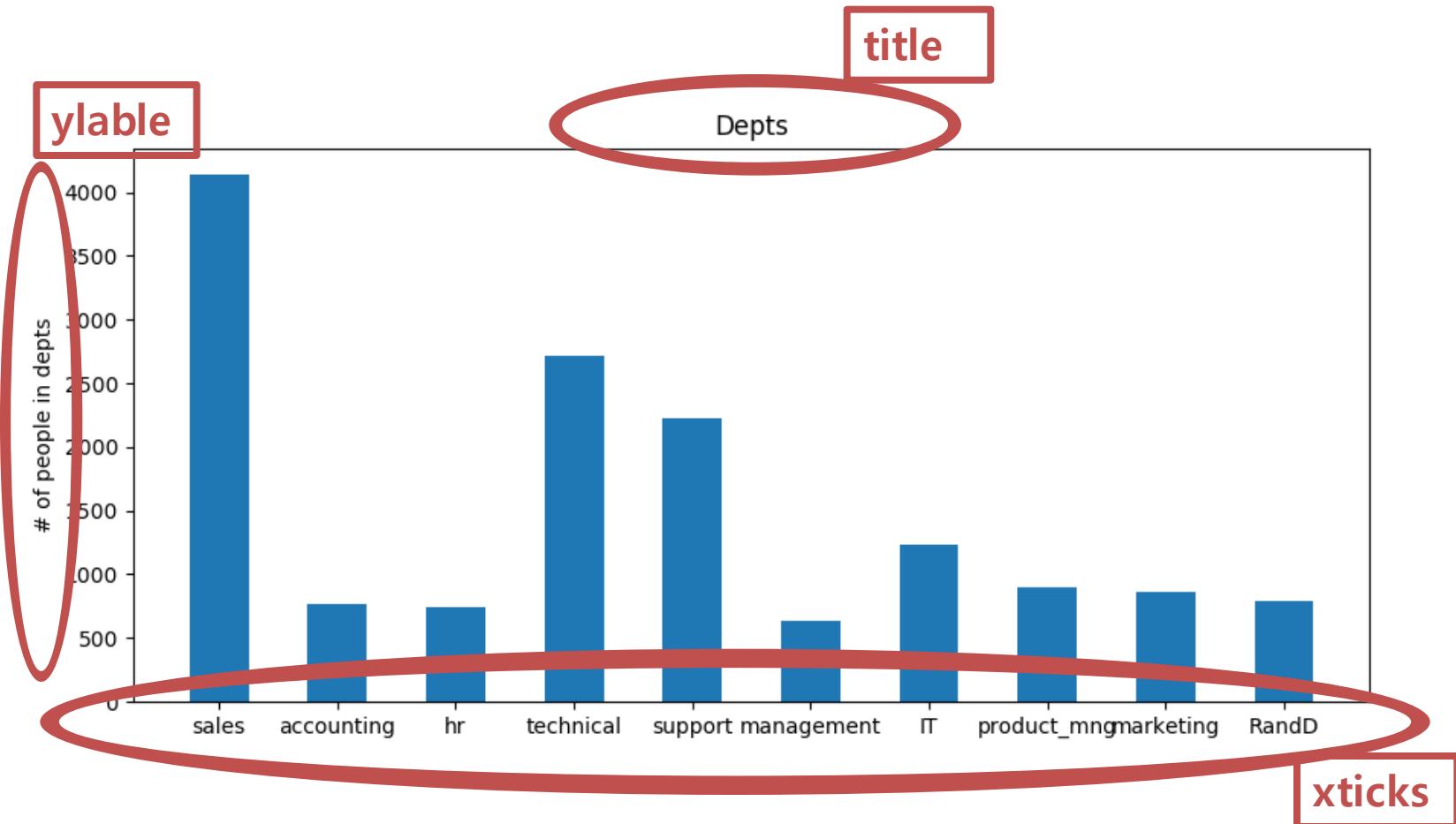
### 막대그래프 (3) Decorating plot



# 02 시각화 Basic Tutorial

## ② Bar charts

### 막대그래프 (3) Decorating plot



# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (1) data cleansing

#### Histogram

- > 정해진 구간에 해당되는 항목의 개수를 보여줌으로써 값의 분포를 관찰할 수 있는 그래프
- > 계급, 도수
- > 연속형 자료! (<-> 막대 그래프는 이산형 자료)

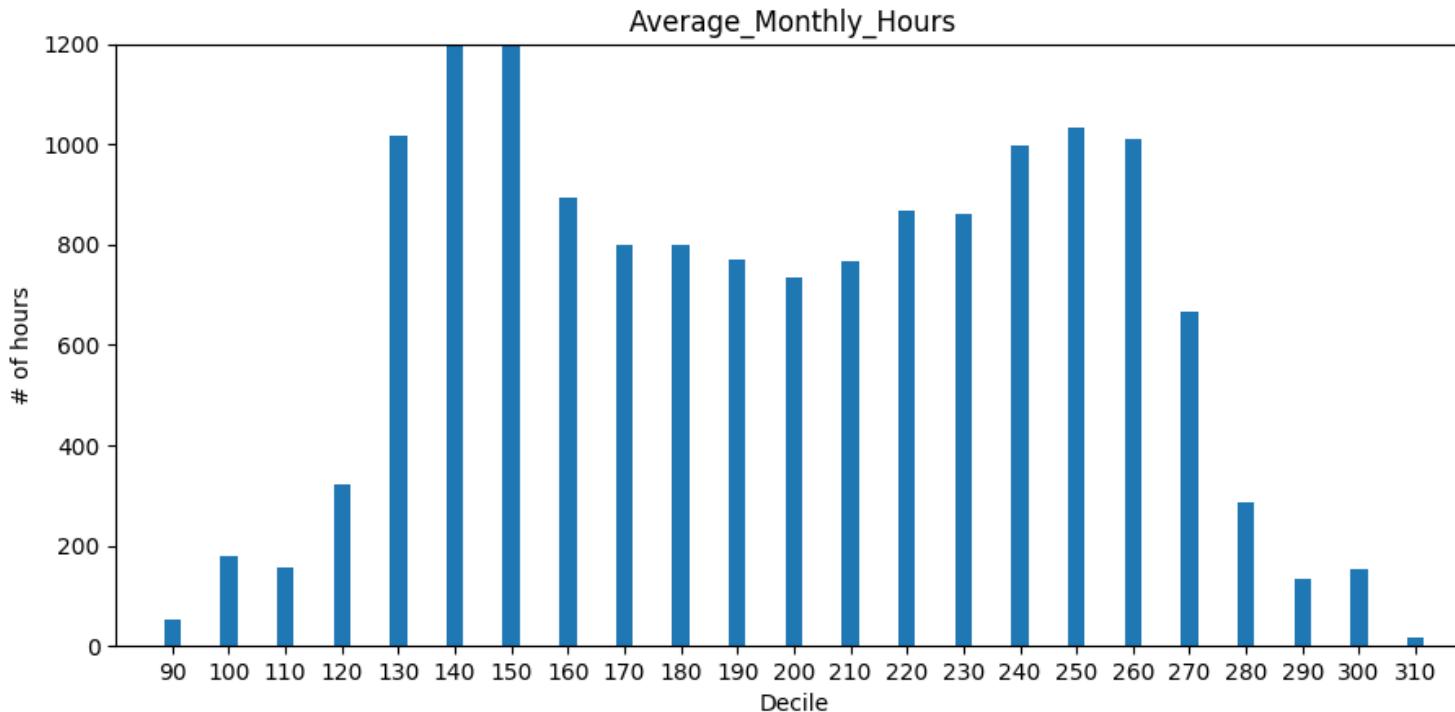
실습

연속형 column 하나를 정해 list로 만들어주세요!

# 02 시각화 Basic Tutorial

## ② Bar charts

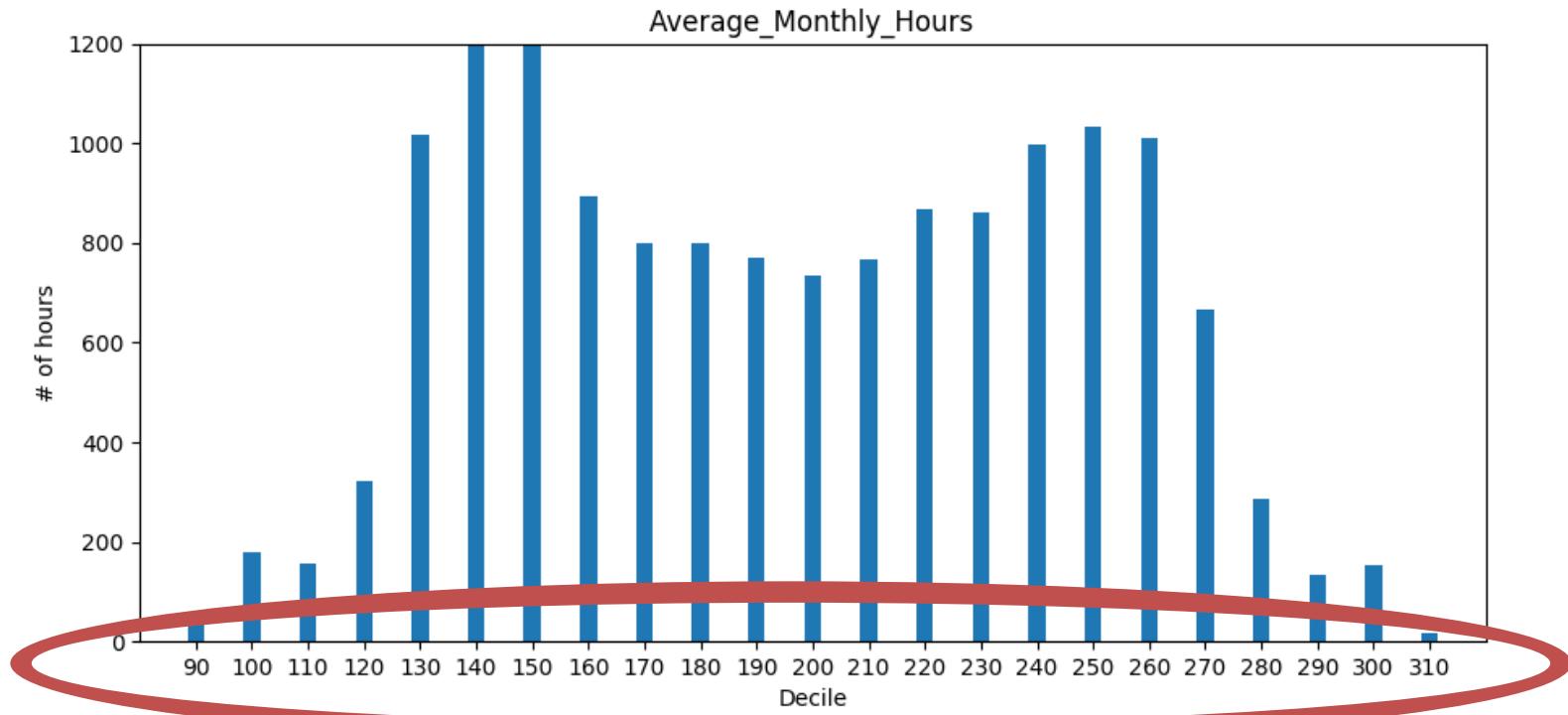
### Histogram (1) data cleansing



# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (1) data cleansing



# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (1) data cleansing

문자열->숫자열  
map(function, x)

```
#자료정제 (1) (문자열 리스트 -> 숫자 리스트)
amhs = [] # amh = average_monthly_hours
for line in lines : #lines는 line으로 이루어진 list
    new_element = line.split(',') [3] #각 line을 ','로 쪼개고 그것 중 3번째(dept)
    amhs.append(new_element) #append는 list의 인수 추가 할수
del amhs[0] # amhs의 첫번째 인수는 'average_monthly_hours'

print(amhs)
#[ '157', '262', '272', '223', '159', '153', '247', '259', '224', '142', '135', '305']
```

'숫자', 즉 문자(str)로 이루어진 list  
-> 계급으로, 도수로 처리할 수 X

# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (1) data cleansing

문자열->숫자열  
map(function, x)

```
#자료정제 (1) (문자열 리스트 -> 숫자 리스트)
amhs=[ ] # amh = average_monthly_hours
for line in lines : #lines는 line으로 이루어진 list
    new_element = line.split(',') [3] #각 line을 ','로 쪼개고 그것 중 3번째(dept)
    amhs.append(new_element) #append는 list의 인수 추가 할수
del amhs[0] # amhs의 첫번째 인수는 'average_monthly_hours'

print(amhs)
#[ '157', '262', '272', '223', '159', '153', '247', '259', '224', '142', '135', '305']
```

'숫자', 즉 문자(str)로 이루어진 list  
 -> 계급으로, 도수로 처리할 수 X  
 -> 그럼 어떻게..?

# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (1) data cleansing

문자열->숫자열  
map(function, x)

map()를 활용하여 x값 만들기!

>>> map(function to apply, list of inputs)

```
print(amhs)
#[157, 262, 272, 223, 159, 153, 247, 259, 224, 142, 135, 305]
```

```
amhs=list(map(int, amhs)) # map(function_to_apply, list_of_inputs)
print(amhs)
#[157, 262, 272, 223, 159, 153, 247, 259, 224, 142, 135, 305, 234, 148, 137, 143, 16]
```

## 02 시각화 Basic Tutorial ② Bar charts

### Histogram (1) data cleansing

급간 나누기  
Lambda( )

Q. 히스토그램의 계급을 어떻게 설정할 수 있을까?

실습

**Hint!**

- lambda로 계급의 크기만큼의 sep을 둔 list 만들기  
>>> decile = lambda( )
- Counter 활용!  
>>> from collections import Counter

# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (1) data cleansing

급간 나누기  
Lambda( )

```
#자료정제 (2) (급간 나눠주기)
decile = lambda amh : amh // 10 * 10
from collections import Counter
histogram = Counter(decile(amh) for amh in amhs)

print(histogram) #dic과 유사! {key : value} = {변량 : 도수}
histogram_keys = list(histogram.keys())
histogram_values = list(histogram.values())
```

## 02 시각화 Basic Tutorial ② Bar charts

### Histogram (2) Creating Plot

-> 이제 histogram을 그려봅시다!

```
#그래프 그리기
from matplotlib import pyplot as plt
plt.bar(histogram_keys, histogram_values, 3)
plt.show()
```

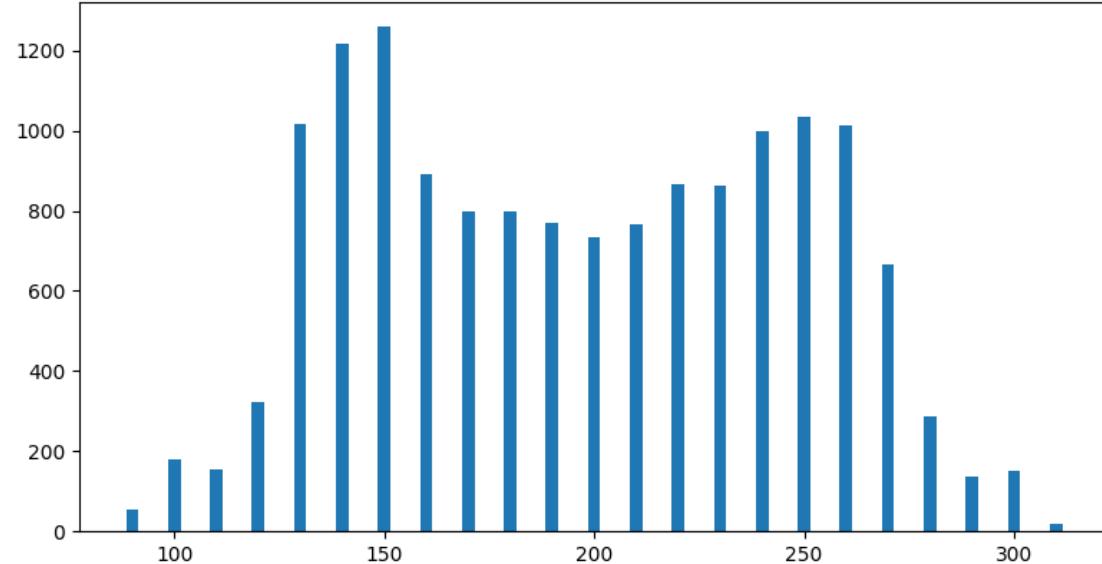
# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (2) Creating Plot

-> 이제 histogram을 그려봅시다!

```
#그래프 그리기
from matplotlib import pyplot as plt
plt.bar(histogram_keys, histogram_values, 3)
plt.show()
```



# 02 시각화 Basic Tutorial

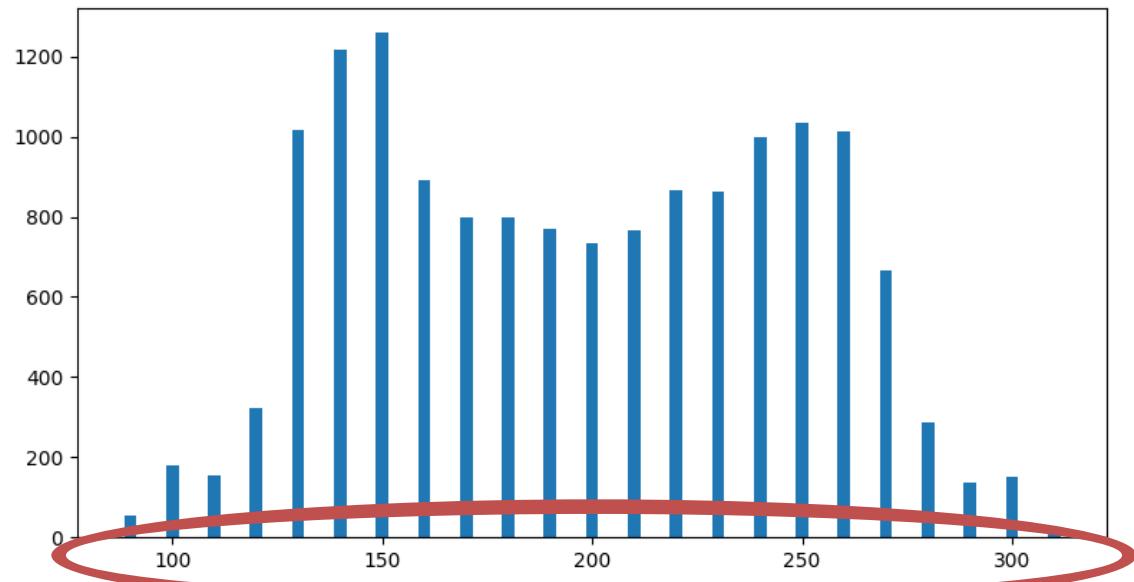
## ② Bar charts

### Histogram (2) Creating Plot

-> 이제 histogram을 그려봅시다!

```
#그래프 그리기  
from matplotlib import pyplot as plt  
plt.bar(histogram_keys, histogram_values, 3)  
plt.show()
```

계급별(x축)  
table 중요!! ->



# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (3) Decorating Plot

... x축 label 표시해주기 plt.xticks() ...  
histogram\_keys.sort() #차례대로 정렬 -> xmin, xmax 확인!

```
print(histogram_keys)  
#[90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250,
```

```
x_label = [i for i in range(histogram_keys[0],histogram_keys[-1]+10,10)]  
# range(0,10,5)는 0에서부터 10 미만의 수 5+씩 나열
```

```
plt.xticks(x_label)  
plt.show()
```

X축 막대별  
label 설정해주기

# 02 시각화 Basic Tutorial

## ② Bar charts

### Histogram (3) Decorating Plot

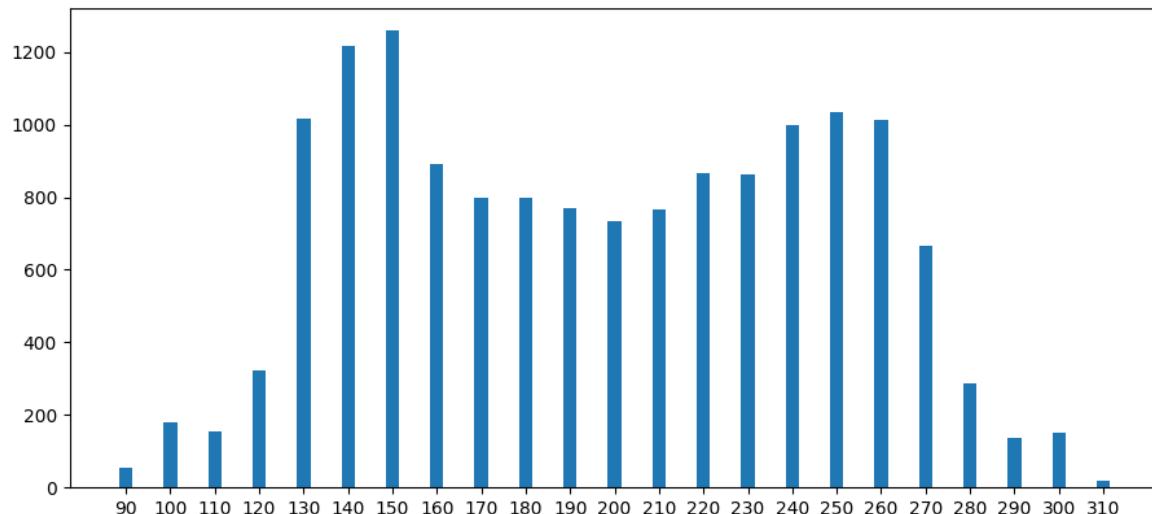
X축 막대별  
label 설정해주기

... x축 label 표시해주기 plt.xticks() ...  
histogram\_keys.sort() #차례대로 정렬 -> xmin, xmax 확인!

```
print(histogram_keys)
#[90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250,
```

```
x_label = [i for i in range(histogram_keys[0],histogram_keys[-1]+10,10)]
# range(0,10,5)는 0에서부터 10 미만의 수 5+씩 나열
```

```
plt.xticks(x_label)
plt.show()
```



# 02 시각화 Basic Tutorial

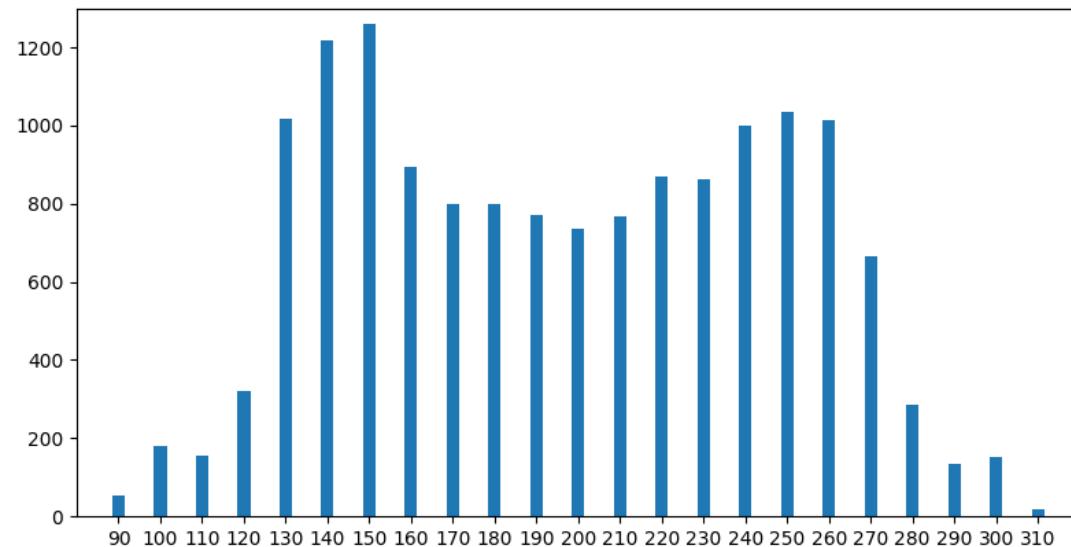
## ② Bar charts

### Histogram (3) Decorating Plot

X축 label  
범위 조정

--- x축 label size 조절하기 ---

```
# plt.axis([x min, x max, y min, y max])  
plt.axis([80, 320, 0, 1300])  
plt.show()
```



# 02 시각화 Basic Tutorial

## ② Bar charts

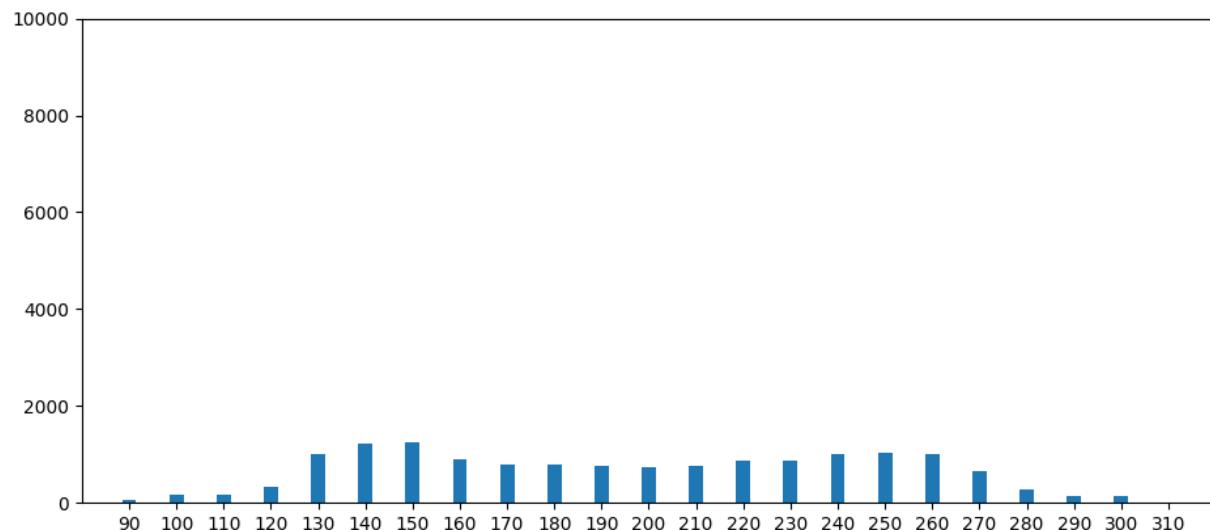
### Histogram (3) Decorating Plot

X축 label  
범위 조정

--- x축 label size 조절하기 ---

```
# plt.axis([x min, x max, y min, y max])  
plt.axis([80, 320, 0, 10000])  
plt.show()
```

-> 적절하지 않은 label size는  
오해를 불러일으킬 소지 존재



## 02 시각화 Basic Tutorial

## ② Bar charts

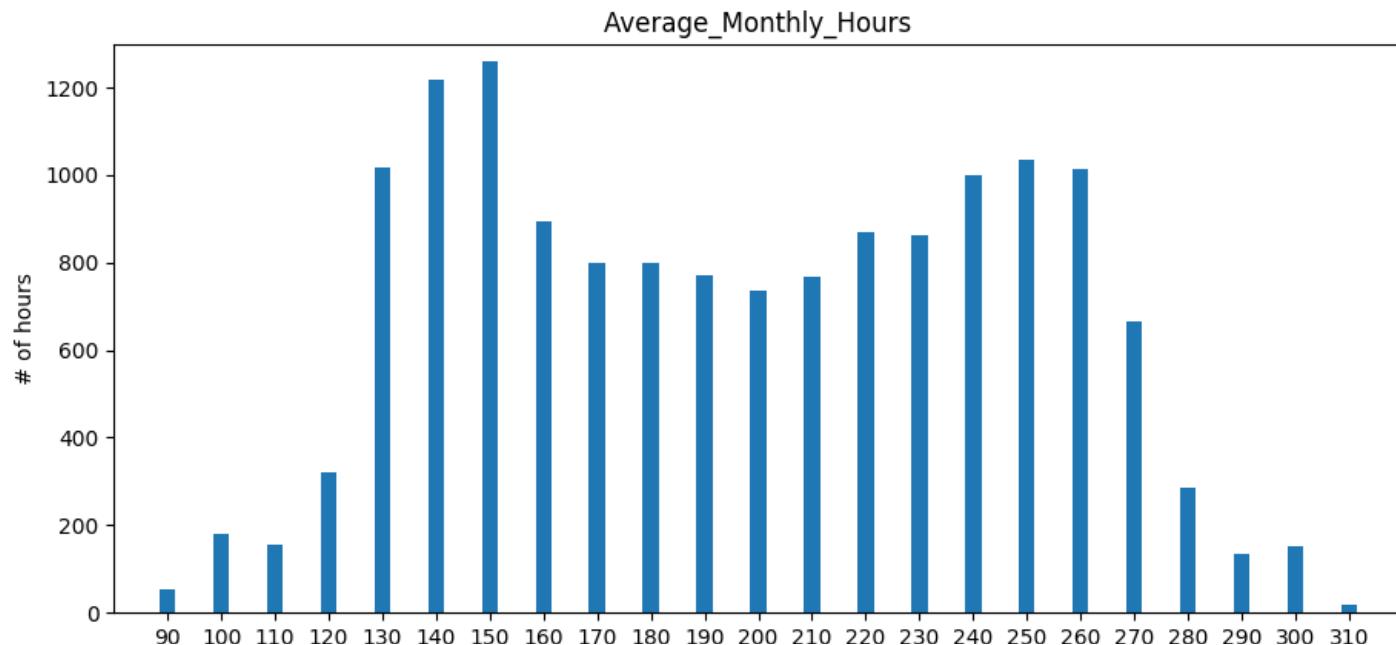
## Histogram (3) Decorating Plot

그 외

Commander

--- 그 외 속성 ---

```
plt.title("Average_Monthly_Hours")
plt.xlabel("Decile")
plt.ylabel("# of hours")
plt.show()
```



## 02 시각화 Basic Tutorial ③ Line Chart

# ③ Line Chart 선 그래프

-> 경향성

# 02 시각화 Basic Tutorial

## ③ Line Chart

### Line Chart (1) Decorating Plot

arguments

- **plt.plot( x, y, arguments )**

```
>>> plt.plot(x, y, color='', linestyle='', marker='')
```

- **arguments in plot**

linewidth  
linestyle  
color  
marker  
markersize  
markeredgewidth  
markeredgecolor  
markerfacecolor  
markerfacecoloralt

fillstyle  
antialiased  
dash\_capstyle  
solid\_capstyle  
dash\_joinstyle  
solid\_joinstyle  
pickradius  
drawstyle  
...

[더 자세한 정보](#)

[http://matplotlib.org/  
api/lines\\_api.html#ma  
tplotlib.lines.Line2D](http://matplotlib.org/api/lines_api.html#matplotlib.lines.Line2D)

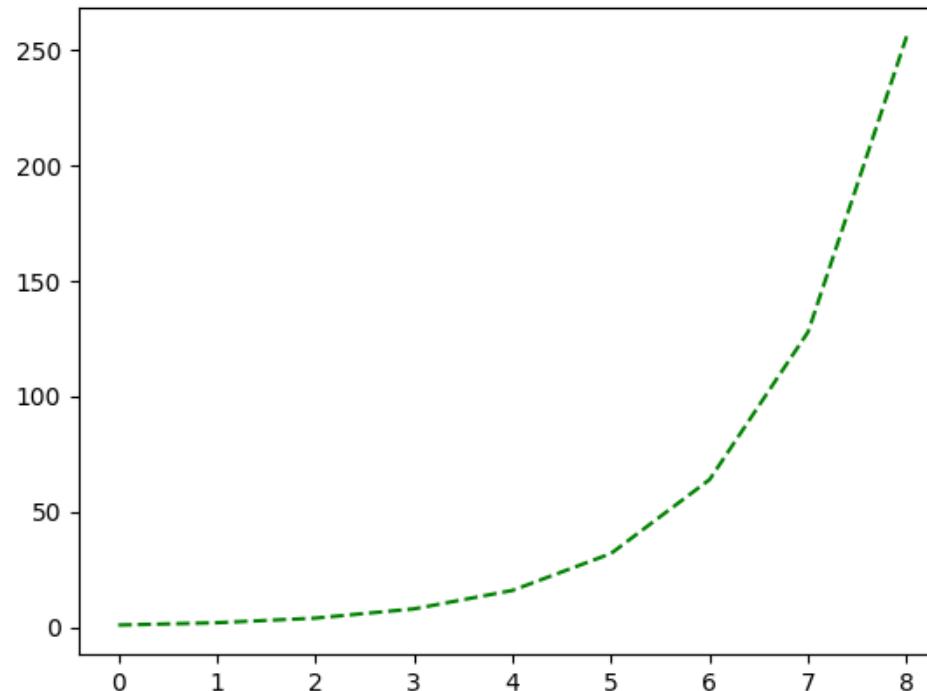
# 02 시각화 Basic Tutorial

## ③ Line Chart

### Line Chart (1) Decorating Plot

arguments

```
# 꾸미기 plot(x, y, color-linestyle-marker) 표현방식 다양, 추가 가능한 속성도 다양
plt.plot(xs, variance, color='green', linestyle='--')
plt.plot(xs, variance, 'g--', label='variance')
plt.show()
```



# 02 시각화 Basic Tutorial

## ③ Line Chart

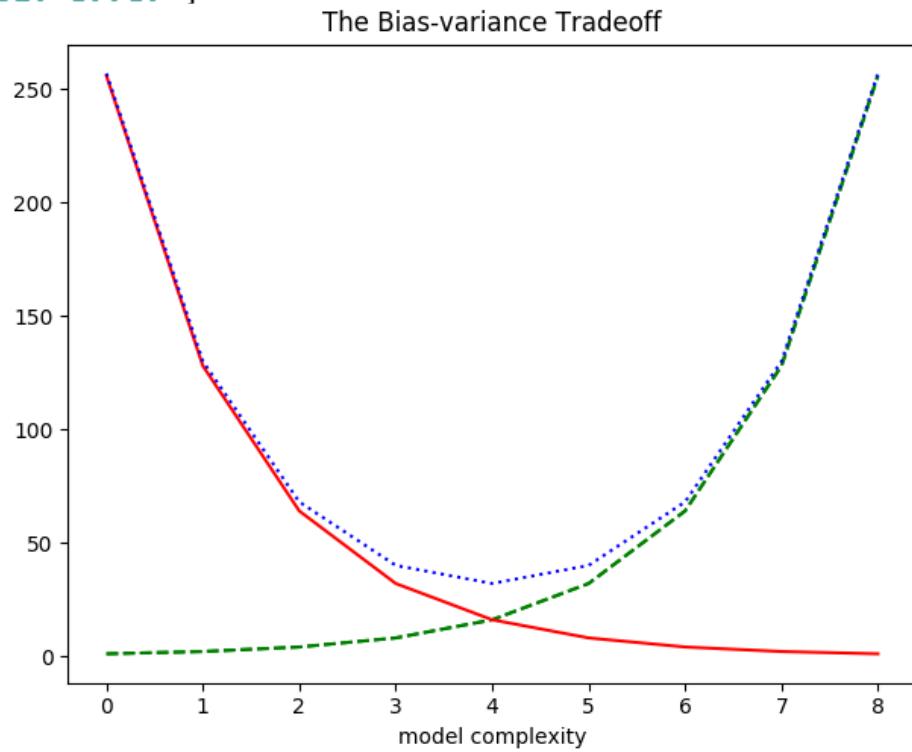
### Line Chart (1) Decorating Plot

```
plt.plot(xs, variance, color='green', linestyle='--')
plt.plot(xs, variance, 'g--', label = 'variance')
plt.plot(xs, bias_squared, 'r-', label = 'bias^2')
plt.plot(xs, total_error, 'b:', label = 'total error')

plt.title("The Bias-variance Tradeoff")
plt.xlabel("model complexity")

plt.show()
```

commander  
그래프 중첩



# 02 시각화 Basic Tutorial

## ③ Line Chart

### Line Chart (1) Decorating Plot

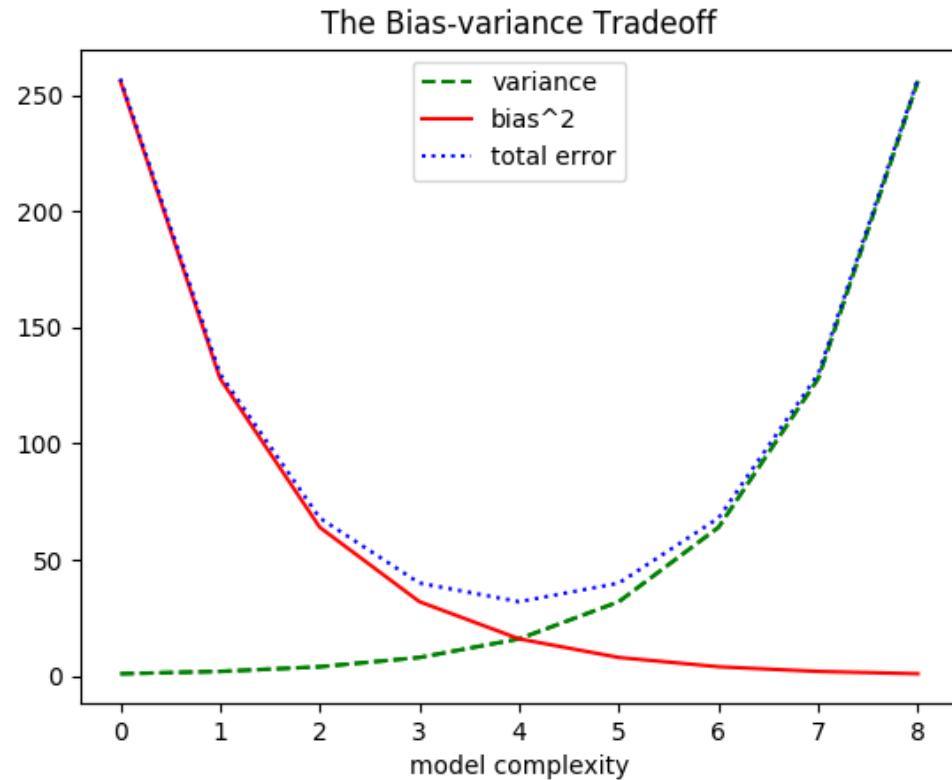
```
plt.plot(xs, variance, color='green', linestyle='--')
plt.plot(xs, variance, 'g--', label = 'variance')
plt.plot(xs, bias_squared, 'r-', label = 'bias^2')
plt.plot(xs, total_error, 'b:', label = 'total error')
```

`plt.legend(loc="top center")`

```
plt.title("The Bias-variance Tradeoff")
plt.xlabel("model complexity")
```

`plt.show()`

**commander**  
범례  
`plt.legend()`



# 02 시각화 Basic Tutorial

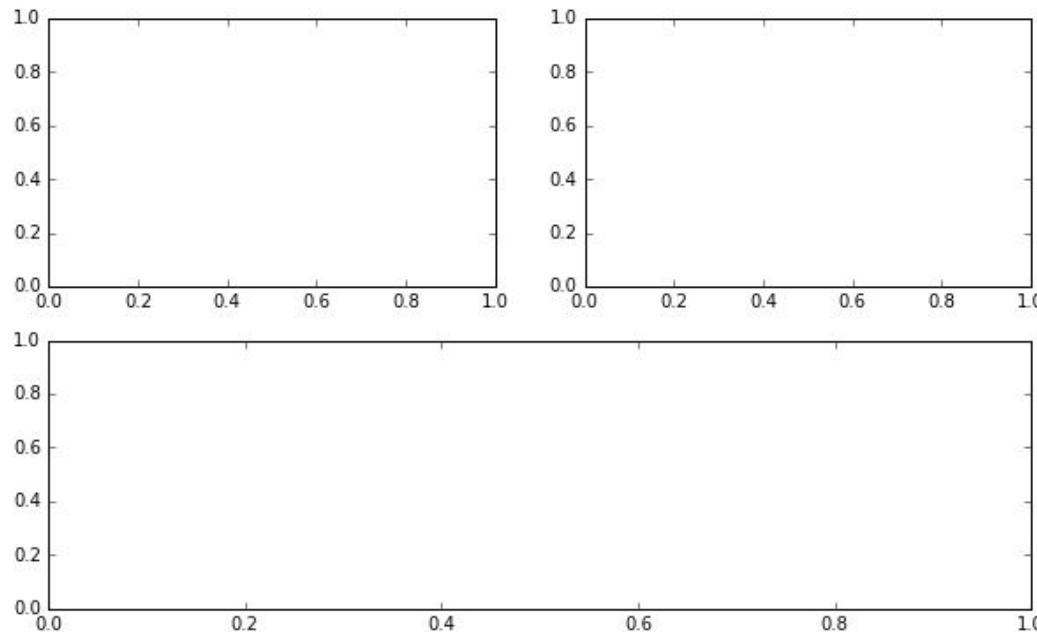
## ③ Line Chart

### Line Chart (1) Decorating Plot

```
In [5]: plt.figure(figsize=(10,6))

plt.subplot(221)
plt.subplot(222)
plt.subplot(212)

plt.show()
```



commander  
**Multiplot**  
plt.subplot()

출처 : <http://pinkwink.kr/972>

# 02 시각화 Basic Tutorial

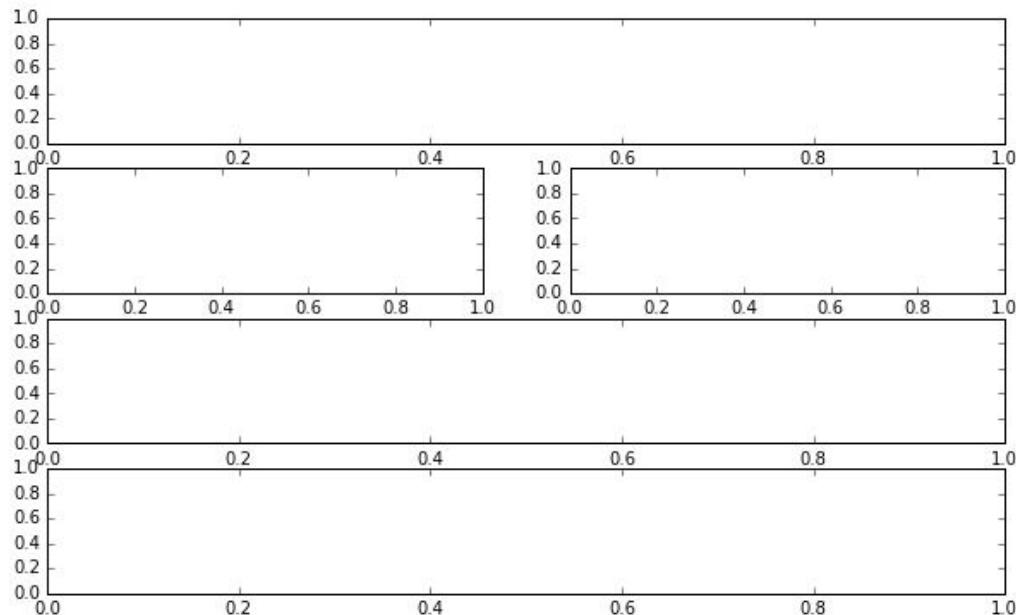
## ③ Line Chart

### Line Chart (1) Decorating Plot

```
In [7]: plt.figure(figsize=(10,6))

plt.subplot(411)
plt.subplot(423)
plt.subplot(424)
plt.subplot(413)
plt.subplot(414)

plt.show()
```



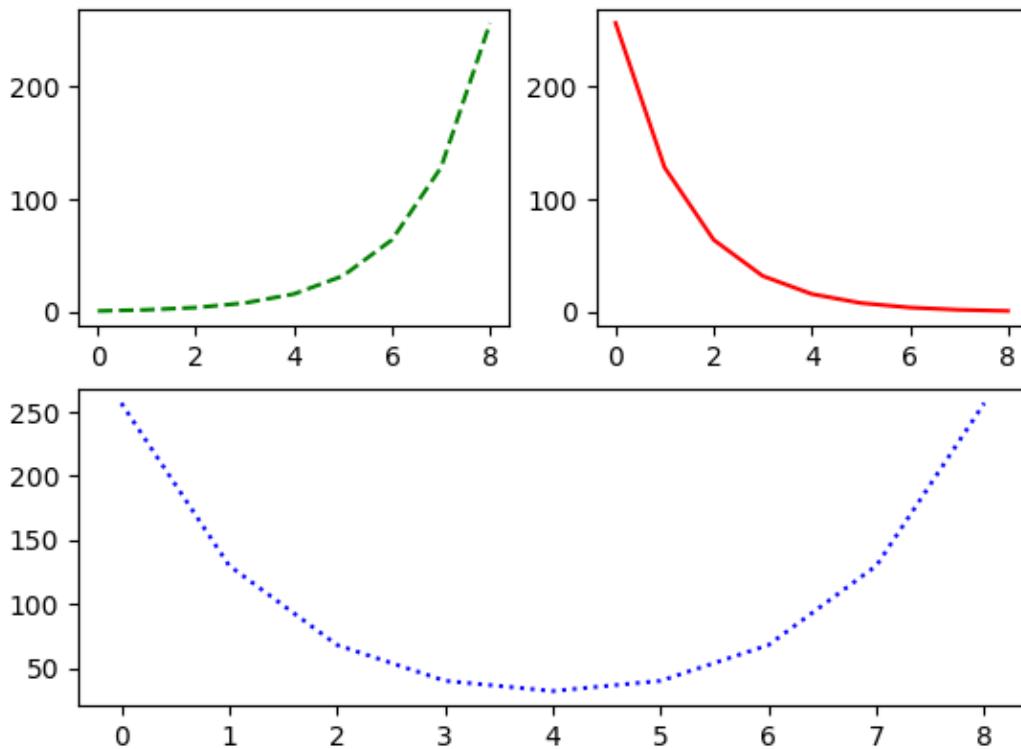
commander  
**Multiplot**  
plt.subplot()

출처 : <http://pinkwink.kr/972>

# 02 시각화 Basic Tutorial

## ③ Line Chart

### Line Chart (1) Decorating Plot



commander  
**Multiplot**  
plt.subplot()

실습

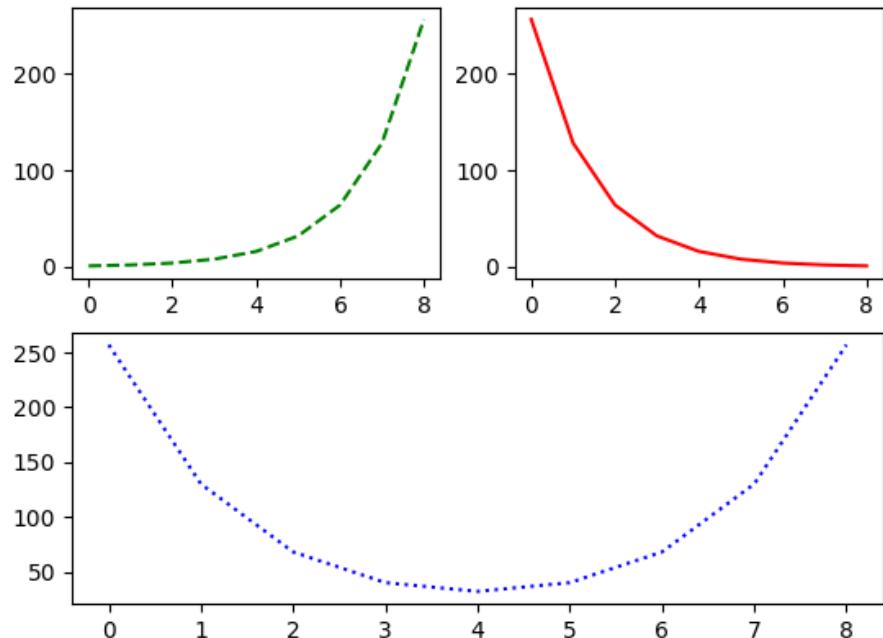
이러한 multiplot을  
표현하기 위해서  
어떻게 입력해야 할까?

# 02 시각화 Basic Tutorial

## ③ Line Chart

### Line Chart (1) Decorating Plot

commander  
**Multiplot**  
plt.subplot()



```
plt.subplot(221)
```

```
plt.plot(xs, variance, color='green', linestyle='--')
```

```
plt.subplot(222)
```

```
plt.plot(xs, bias_squared, 'r-', label = 'bias^2')
```

```
plt.subplot(212)
```

```
plt.plot(xs, total_error, 'b:', label = 'total error')
```

# ④ Scatter Plot

## 산점도

# 02 시각화 Basic Tutorial

## ④ Scatter Plot

### Scatter Plot

## Scatter Plot

-> 두 변수 간의 연관 관계 나타낼 때

-> 'HR\_comma\_sep.csv' data에서  
직장 만족도(satisfaction\_level) & 월 평균 근무시간  
(average\_monthly\_hours)으로 그려보자!

# 02 시각화 Basic Tutorial

## ④ Scatter Plot

### Scatter Plot (1) Cleansing data

---csv파일에서 필요한 자료 추출하기---

```
satisfaction=[]
tsc=[]
for line in lines : #lines는 line으로 이루어진 list
    ele_satisfaction = line.split(',') [0] #각 line을 ','로 쪼개고 그것 중 0번째
    satisfaction.append(ele_satisfaction) #append는 list의 할수
    ele_tsc = line.split(',') [3] #tsc=time_spend_company
    tsc.append(ele_tsc)

del satisfaction[0]
del tsc[0]

#문자 list를 숫자 list로! / python3는 list() 따로 둘어줘야 함!
satisfaction = list(map(float,satisfaction))
tsc = list(map(int, tsc))
```

# 02 시각화 Basic Tutorial

## ④ Scatter Plot

### Scatter Plot (1) Creating + Decorating plot

```
# 그래프 그리기 (+꾸미기)
from matplotlib import pyplot as plt
plt.scatter(satisfaction, tsc, s=2, color='red', marker='s')
plt.grid(True)
plt.title('Is there any Correlation?')
plt.xlabel("satisfaction_level")
plt.ylabel("time_spend_company")
plt.show()
```

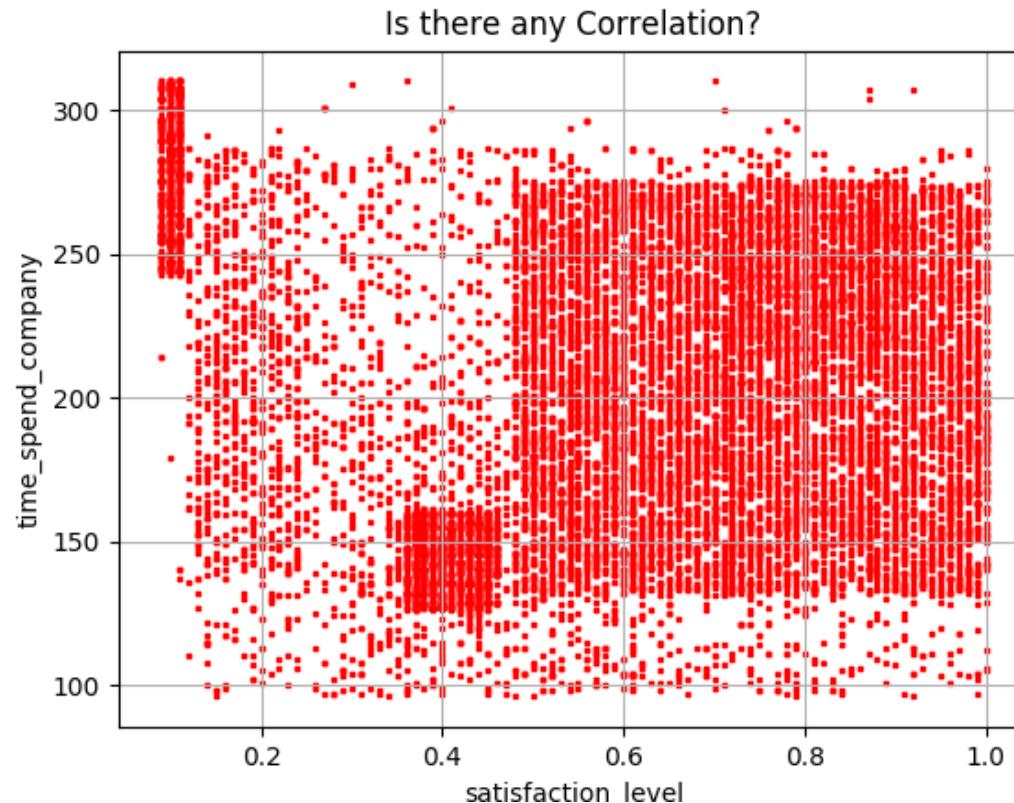
# 02 시각화 Basic Tutorial

## ④ Scatter Plot

### Scatter Plot (1) Creating + Decorating plot

```
# 그래프 그리기 (+꾸미기)
from matplotlib import pyplot as plt
plt.scatter(satisfaction, tsc, s=2, color='red', marker='s')
plt.grid(True)
plt.title('Is there any Correlation?')
plt.xlabel("satisfaction_level")
plt.ylabel("time_spend_company")
plt.show()
```

어떤 관계?



# 03 데이터 다루기

## ■ 사실 확인 (내부적인 목적)

앞에서 사용한 “HR\_comma\_sep.csv” 파일을 이용합니다.

1차원 데이터 vs 다차원 데이터

# 03 데이터 다루기

## 1차원 데이터

앞에서 사용한 "HR\_comma\_sep.csv" 파일을 이용합니다.

- 1) 요약 통계치 (summary statistics)
  - 데이터 개수, 최솟값, 최댓값, 평균, 표준편차
- 2) 히스토그램

# 03 데이터 다루기

## 사실 확인 (내부적인 목적)

```
f=open("HR_comma_sep.csv","r",encoding="UTF-8")
lines=f.readlines()
average_montly_hours=[]
for line in lines:
    worker_amh=line.split(',')[2]
    average_montly_hours.append(worker_amh)
print(average_montly_hours)
```

# 03 데이터 다루기

## ■ 가공을 위해 바꿀 점을 2가지 찾아보세요

현재 데이터 모양

```
['average_montly_hours', '157', '262', '272', '223', '159', '153', '247', '  
259', '224', '142', '135', '305', '234', '148', '137', '143', '160', '255', '1  
60', '262', '282', '147', '304', '139', '158', '242', '239', '135', '128', '13  
2', '294',...]
```

원하는 데이터 모양

```
[157, 262, 272, 223, 159, 153, 247, 259, 224, 142, 135, 305, 234, 148, 137, 143,  
160, 255, 160, 262, 282, 147, 304, 139, 158, 242, 239, 135, 128, 132, 294, ...]
```

# 03 데이터 다루기

## ■ 가공을 위해 바꿀 점을 2가지 찾아보세요

원하는 데이터 모양

[157, 262, 272, 223, 159, 153, 247, 259, 224, 142, 135, 305, 234, 148, 137, 143, 160, 255, 160, 262, 282, 147, 304, 139, 158, 242, 239, 135, 128, 132, 294, ...]

```
del average_montly_hours[0]
int_average_monthly_hours=list(map(int,average_montly_hours))

print(int_average_monthly_hours)
```

# 03 데이터 다루기

## 1차원 데이터

데이터 개수, 최솟값, 최댓값, 평균, 표준편차 (요약 통계치)

pandas 패키지를 사용하면 엑셀에서 하는 기본  
데이터 요약 통계치 기능을 쉽게 처리할 수 있다

pandas is a **Python** package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in **Python**

**import pandas as pd**

<https://pandas.pydata.org/pandas-docs/stable/>



# 03 데이터 다루기

## Pandas의 자료구조



우리가 python에서 쓰는 데이터타입을  
pandas에서 쓸 수 있는 데이터타입으로 바  
꿔줘야한다

**Series (하나의 열에 index를 가지는 Pandas의 자료 구조 - 1차원 데이터)**

**DataFrame (여러 열에 index를 가지는 Pandas의 자료 구조 - 다차원 데이터)**

# 03 데이터 다루기

## ■ Series

```
pd.Series(int_average_monthly_hours)
seriesform=pd.Series(int_average_monthly_hours)
print(seriesform)
```

```
[In [18]: seriesform=pd.Series(int_average_monthly_hours)
```

```
[In [19]: print(seriesform)
```

0	157
1	262
2	272
3	223
4	159
5	153
6	247
7	259
8	224
9	142
10	135
11	305
12	234
13	148
14	137
15	143
16	160
17	255

# 03 데이터 다루기

## 묘사해봐! `describe()`로 데이터 요약 통계치 구하기

Series로 바꾸었으니  
타입이 되었습니다



에서 데이터를 쉽게 처리하기 적합한 데이터

### `seriesform.describe()`

14999개의 데이터가 있고  
평균 201.050337 (시간)  
표준편차 49.943  
최솟값 96  
1/4분위수 (first quartile, 25%)  
156  
중앙값(median, 50%) 200  
3/4분위수 (third quartile, 75%)  
245  
최댓값 310

```
[In [20]: seriesform.describe()
Out[20]:
count      14999.000000
mean       201.050337
std        49.943099
min        96.000000
25%       156.000000
50%       200.000000
75%       245.000000
max        310.000000
dtype: float64
```

# 03 데이터 다루기

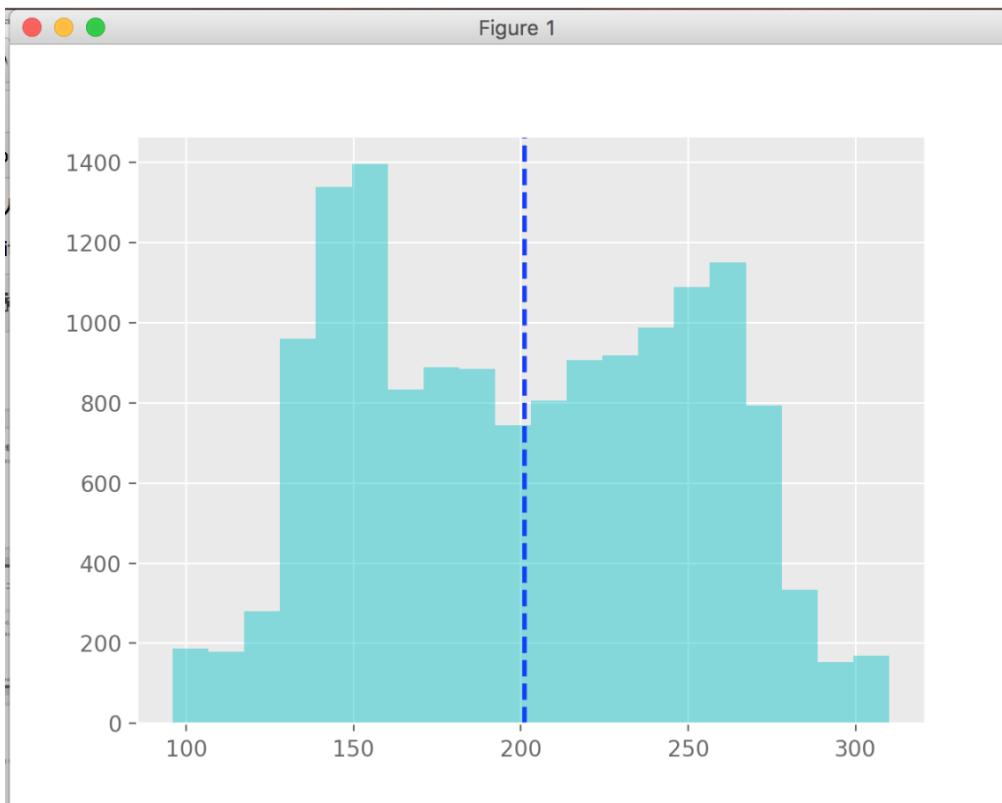
## matplotlib로 히스토그램 그리기

```
import matplotlib.pyplot as plt  
plt.style.use('ggplot') R의 plotting 스타일  
plt.hist(seriesform, bins=20, alpha=0.5,color='c')  
plt.axvline(x=seriesform.mean(), color='b', linestyle='dashed', linewidth=2)  
plt.show()
```

[http://matplotlib.org/api/\\_as\\_gen/matplotlib.axes.Axes.axvline.html](http://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.axvline.html)

# 03 데이터 다루기

## matplotlib로 히스토그램 그리기



# 03 데이터 다루기

## DataFrame

```
csv_file = pd.read_csv("HR_comma_sep.csv")
```

[14999 rows x 10 columns]

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	sales	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low
5	0.41	0.50	2	153	3	0	1	0	sales	low
6	0.10	0.77	6	247	4	0	1	0	sales	low
7	0.92	0.85	5	259	5	0	1	0	sales	low
8	0.89	1.00	5	224	5	0	1	0	sales	low
9	0.42	0.53	2	142	3	0	1	0	sales	low
10	0.45	0.54	2	135	3	0	1	0	sales	low
11	0.11	0.81	6	305	4	0	1	0	sales	low
12	0.84	0.92	4	234	5	0	1	0	sales	low
13	0.41	0.55	2	148	3	0	1	0	sales	low
14	0.36	0.56	2	137	3	0	1	0	sales	low
15	0.38	0.54	2	143	3	0	1	0	sales	low

type(csv\_file)

Out[13]: pandas.core.frame.DataFrame

맨 첫 줄을 각 열의 이름 행으로 인식함

DataFrame이라는 데이터 타입 확인 가능

# 03 데이터 다루기

## 모든 열의 요약 통계치를 구하는 방법 (feat.

`csv_file.describe()`



```
In [14]: csv_file.describe()
Out[14]:
      satisfaction_level  last_evaluation  number_project \
count          14999.000000    14999.000000    14999.000000
mean           0.612834     0.716102     3.803054
std            0.248631     0.171169     1.232592
min            0.090000     0.360000     2.000000
25%           0.440000     0.560000     3.000000
50%           0.640000     0.720000     4.000000
75%           0.820000     0.870000     5.000000
max            1.000000     1.000000     7.000000

      average_montly_hours  time_spend_company  Work_accident      left \
count          14999.000000    14999.000000    14999.000000    14999.000000
mean          201.050337     3.498233     0.144610     0.238083
std           49.943099     1.460136     0.351719     0.425924
min           96.000000     2.000000     0.000000     0.000000
25%          156.000000     3.000000     0.000000     0.000000
50%          200.000000     3.000000     0.000000     0.000000
75%          245.000000     4.000000     0.000000     0.000000
max          310.000000    10.000000     1.000000     1.000000

      promotion_last_5years
count          14999.000000
mean           0.021268
std            0.144281
min            0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max            1.000000
```

왜 column이 8개로 줄었을까요?

# 03 데이터 다루기

## DataFrame

```
[In [14]: csv_file.describe()
Out[14]:
      satisfaction_level  last_evaluation  number_project \
count          14999.000000     14999.000000    14999.000000
mean           0.612834      0.716102      3.803054
std            0.248631      0.171169     1.232592
min            0.090000      0.360000      2.000000
25%           0.440000      0.560000      3.000000
50%           0.640000      0.720000      4.000000
75%           0.820000      0.870000      5.000000
max            1.000000      1.000000      7.000000

      average_monthly_hours  time_spend_company  Work_accident  left \
count          14999.000000     14999.000000    14999.000000  14999.000000
mean           201.050337      3.498233      0.144610      0.238083
std            49.943099      1.460136      0.351719      0.425924
min            96.000000      2.000000      0.000000      0.000000
25%          156.000000      3.000000      0.000000      0.000000
50%          200.000000      3.000000      0.000000      0.000000
75%          245.000000      4.000000      0.000000      0.000000
max            310.000000     10.000000      1.000000      1.000000

      promotion_last_5years
count          14999.000000
mean           0.021268
std            0.144281
min            0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max            1.000000
```

Sales, salary 항목은 범주형 자료였다!

그리고 보니 이렇게 요약 통계치를 주면 안되는데 준 column도 보이지 않나요?

# 03 데이터 다루기

## DataFrame

### csv\_file.describe()

```
[In [14]: csv_file.describe()
Out[14]:
      satisfaction_level  last_evaluation  number_project \
count          14999.000000     14999.000000    14999.000000 \
mean           0.612834      0.716102      3.803054
std            0.248631      0.171169     1.232592
min            0.090000      0.360000      2.000000
25%           0.440000      0.560000      3.000000
50%           0.640000      0.720000      4.000000
75%           0.820000      0.870000      5.000000
max            1.000000      1.000000      7.000000

      average_monthly_hours  time_spend_company  Work_accident  \
count          14999.000000     14999.000000    14999.000000 \
mean           201.050337      3.498233      0.144610      0.238083
std            49.943099      1.460136      0.351719      0.425924
min            96.000000      2.000000      0.000000      0.000000
25%          156.000000      3.000000      0.000000      0.000000
50%          200.000000      3.000000      0.000000      0.000000
75%          245.000000      4.000000      0.000000      0.000000
max           310.000000     10.000000      1.000000      1.000000

      promotion_last_5years
count          14999.000000
mean           0.021268
std            0.144281
min            0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max            1.000000
```

Sales, salary 항목은 범주형 자료였다!  
그리고 보니 이렇게 요약 통계치를 주면 안  
되는데 준 column도  
보이지 않는가요?

max가 1이면 의심해봅시다 (boolean type  
인 것)

# 03 데이터 다루기

## 비슷한 열끼리 1차원 데이터 분석을 해봅시다

satisfaction\_level과 last\_evaluation을 해봅시다

```
[In [22]: csv_file.describe()
Out[22]:
satisfaction_level    last_evaluation    number_project \
count          14999.000000      14999.000000      14999.000000
mean           0.612834        0.716102       3.803054
std            0.248631        0.171169       1.232592
min           0.090000        0.360000       2.000000
25%          0.440000        0.560000       3.000000
50%          0.640000        0.720000       4.000000
75%          0.820000        0.870000       5.000000
max           1.000000        1.000000       7.000000

average_montly_hours   time_spend_company   Work_accident
count          14999.000000      14999.000000      14999.000000
mean           201.050337        3.498233       0.144610
std            49.943099        1.460136       0.351719
min           96.000000        2.000000       0.000000
25%          156.000000        3.000000       0.000000
50%          200.000000        3.000000       0.000000
75%          245.000000        4.000000       0.000000
max           310.000000       10.000000       1.000000
```



csv\_file['satisfaction\_level']

csv\_file['last\_evaluation']

이렇게 하면 열의 이름으로  
값을 가져올 수 있습니다.

# 03 데이터 다루기

## 비슷한 열끼리 1차원 데이터 분석을 해봅시다

csv\_file['satisfaction\_level']

csv\_file['last\_evaluation']



```
[In [23]: csv_file['satisfaction_level']
Out[23]:
0      0.38
1      0.80
2      0.11
3      0.72
4      0.37
5      0.41
6      0.10
7      0.92
8      0.89
9      0.42
10     0.45
11     0.11
12     0.84
13     0.41
14     0.36
15     0.38
16     0.45
17     0.78
18     0.45
19     0.76
20     0.11
21     0.38
22     0.09
23     0.46
24     0.40
25     0.89
26     0.82
27     0.40
28     0.41
29     0.38
...
14969  0.46
```

```
[In [25]: csv_file['last_evaluation']
Out[25]:
0      0.53
1      0.86
2      0.88
3      0.87
4      0.52
5      0.50
6      0.77
7      0.85
8      1.00
9      0.53
10     0.54
11     0.81
12     0.92
13     0.55
14     0.56
15     0.54
16     0.47
17     0.99
18     0.51
19     0.89
20     0.83
21     0.55
22     0.95
23     0.57
24     0.53
25     0.92
26     0.87
27     0.49
28     0.46
29     0.50
...
14969  0.46
```

# 03 데이터 다루기

## ■ 히스토그램으로 두 데이터 비교

```
import matplotlib.pyplot as plt
```

```
satisfaction_level=csv_file['satisfaction_level']
```

```
plt.style.use('ggplot')
```

```
plt.hist(satisfaction_level, bins=10,color='y')
```

```
plt.axvline(x=satisfaction_level.mean(), color='r',  
linestyle='dashed', linewidth=2)
```

```
plt.show()
```

```
last_evaluation=csv_file['last_evaluation']
```

```
plt.style.use('ggplot')
```

```
plt.hist(last_evaluation, bins=10,color='y')
```

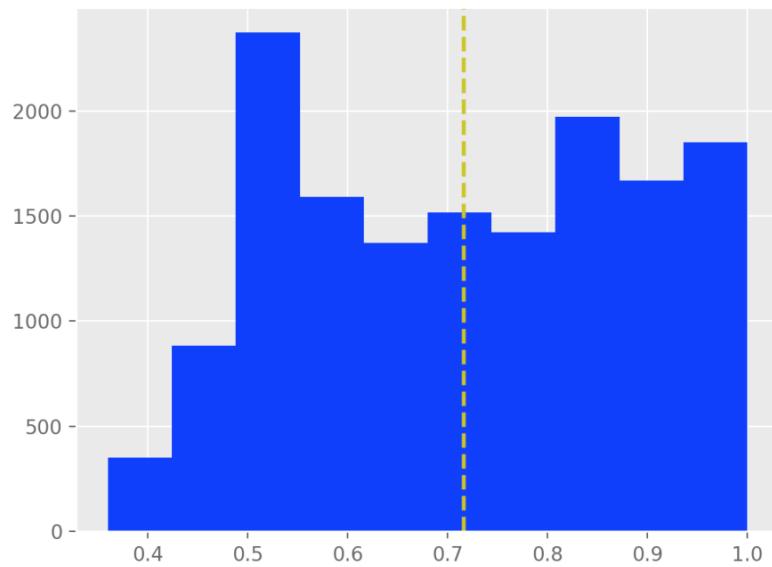
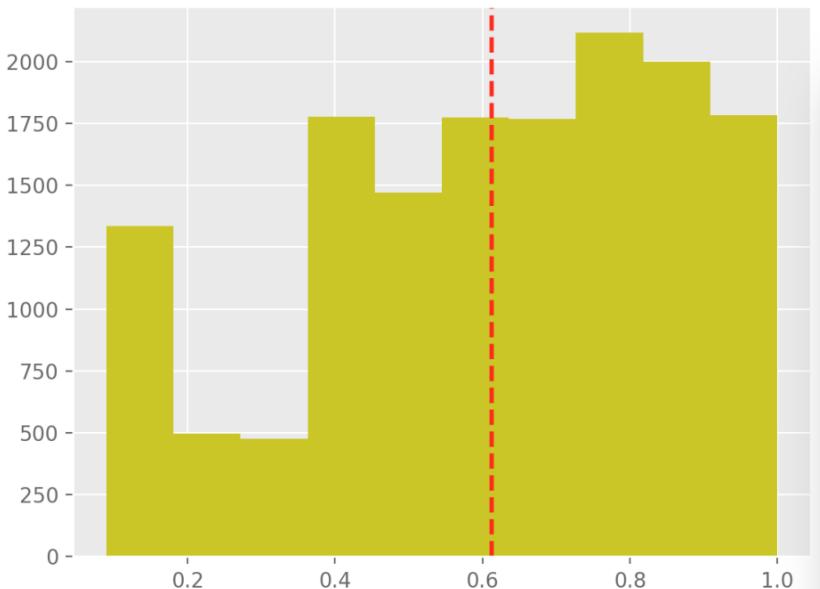
```
plt.axvline(x=last_evaluation.mean(), color='r',  
linestyle='dashed', linewidth=2)
```

```
plt.show()
```

# 03 데이터 다루기

## 히스토그램으로 두 데이터 비교

```
import matplotlib.pyplot as plt
```



# 03 데이터 다루기

## ■ 2차원 데이터

평균과 표준편차가 비슷한

'satisfaction\_level'과 'last\_evaluation'이 각각 다른 데이터와 결합될 때는 어떤 상관관계를 가지는지 확인해봅시다.

'average\_montly\_hours'와 'left'와 각각이 결합될 때의 상관관계를 확인해보겠습니다.

# 03 데이터 다루기

## Pandas DataFrame 열 / 행 가져오기

### 열 가져오기

```
csv_file['satisfaction_level'] #열의 이름으로 가져올 수 있다  
csv_file['last_evaluation'] #열의 이름으로 가져올 수 있다
```

### 행 가져오기

```
csv_file.iloc[[0]] #자동으로 생성되는 정수 인덱스로 가져올  
수 있다
```



# 03 데이터 다루기

## ■ 산포도로 두 데이터 비교 (결합확률분포)

```
satisfaction_level=csv_file['satisfaction_level']
last_evaluation=csv_file['last_evaluation']
left=csv_file['left']

plt.style.use('ggplot')
plt.scatter(left,satisfaction_level,marker='.',color='red',label='satisfaction_level', s=3)
plt.scatter(left,last_evaluation,marker='.',color='gray',label='last_evaluation', s=3)
plt.xlabel('left')
plt.ylabel('satisfaction_level & last_evaluation')
plt.title('Joint Distribution')
plt.show()

print(satisfaction_level.corr(left))
print(last_evaluation.corr(left))
```

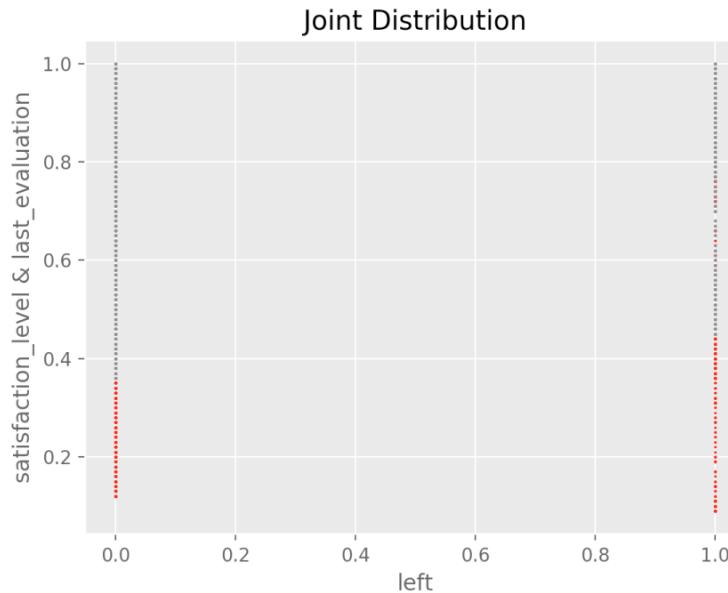
# 03 데이터 다루기

## 산포도로 left와 두 데이터 비교

Correlation 결과는?

-0.388374983424 (satisfaction\_level과 left)

0.00656712044753 (last\_evaluation과 left)



# 03 데이터 다루기

## 산포도로 left와 두 데이터 비교

```
satisfaction_level=csv_file['satisfaction_level']
last_evaluation=csv_file['last_evaluation']
average_montly_hours=csv_file['average_montly_hours']
plt.style.use('ggplot')
plt.scatter(average_montly_hours,satisfaction_level,marker='.',color='red',label='satisfaction_level', s=3)
plt.scatter(average_montly_hours,last_evaluation,marker='.',color='gray',label='last_evaluation', s=3)
plt.xlabel('average_montly_hours')
plt.ylabel('satisfaction_level & last_evaluation')
plt.title('Joint Distribution')
plt.show()

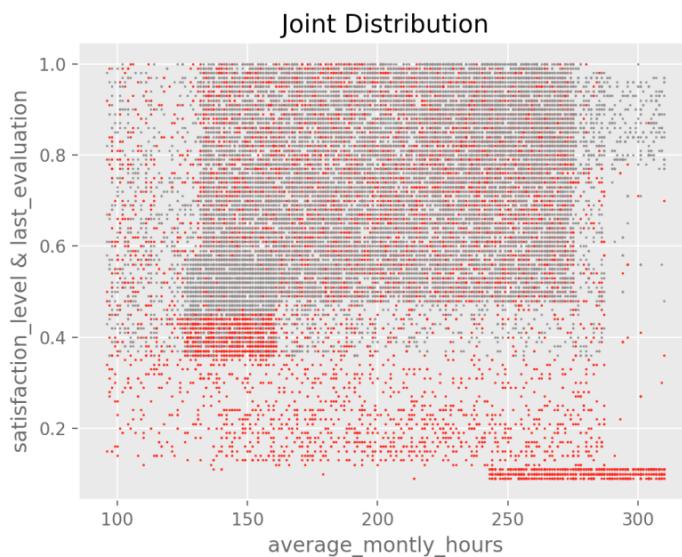
print(satisfaction_level.corr(average_montly_hours))
print(last_evaluation.corr(average_montly_hours))
```

# 03 데이터 다루기

## 산포도로 average\_monthly\_hours와 두 데이터 비교

Correlation 결과는?

-0.0200481132195 (satisfaction\_level과 average\_monthly\_hours)  
0.339741799838 (last\_evaluation과 average\_monthly\_hours)



# 03 데이터 다루기

## Seaborn



주로 sns라는 이름으로 import합니다.

**import seaborn as sns**

`sns.set()` #스타일을 변경한다.

`sns.set_color_codes()`

seaborn에 대해 기초적인 내용 소개하고 있는  
한국 웹사이트

<https://datascienceschool.net/view-notebook/4c2d5ff1caab4b21a708cc662137bc65/>

Seaborn 갤러리 (시각화 한번씩 보길 추천!)

<http://seaborn.pydata.org/examples/index.html>

# 03 데이터 다루기

## ■ 다차원 데이터

산포도행렬을 그리기

```
import seaborn as sns; sns.set()
import pandas as pd
import matplotlib.pyplot as plt
import os

os.path.abspath(os.path.curdir)
os.chdir(r'/Users/kangmina')
csv_file=pd.read_csv("HR_comma_sep.csv", dtype=
{'left':bool,'promotion_last':bool})

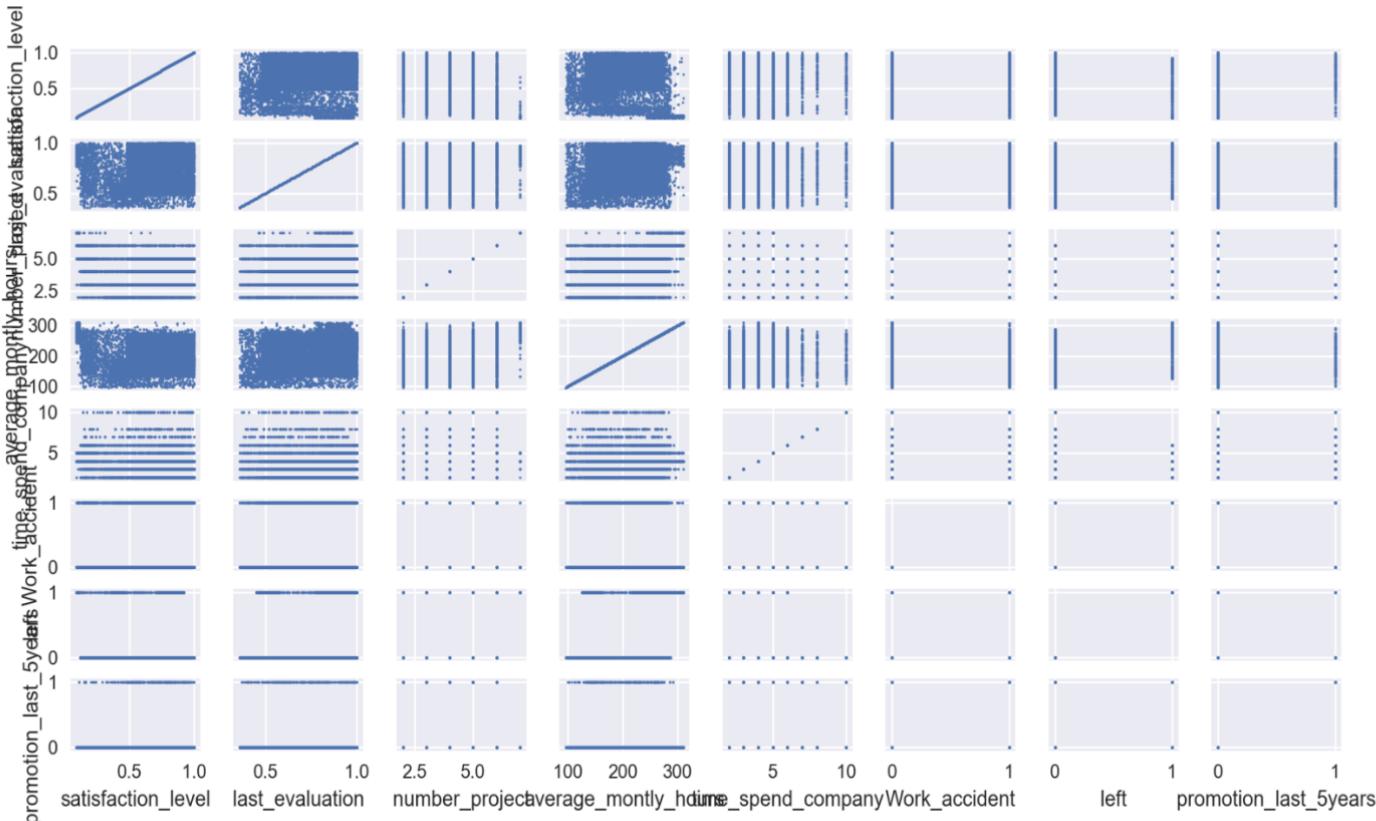
making_pair=sns.PairGrid(csv_file, size=2)
corr_matrix=making_pair.map(plt.scatter, s=1)
plt.show(corr_matrix)
```



# 03 데이터 다루기

## 어떤 데이터 관계가 보이시나요?

<http://seaborn.pydata.org/generated/seaborn.PairGrid.html>



# 03 데이터 다루기

## 10.2 정제하고 합치기

### PARSING (구문분석)

#### 1. 언어학

: 문장을 구성 성분으로 분해하고 분석하여 문장의 구조 결정

#### 2. 컴퓨터과학

: 일련의 문자열을 의미 있는 토큰(token)으로 분해하고 이들로 이루어진 파스 트리(parse tree)를 만드는 과정

- 토큰 : 어휘 분석의 단위
- 파스 트리 : 문장의 구조를 나무 그림으로 나타낸 것

# 03 데이터 다루기

## 10.2 정제하고 합치기 (1)

```
"""10.2 정제하고 합치기"""


```

```
""" 1. csv.reader를 치환하는 함수 만들기 """


```

```
def parse_row(input_row, parsers):
    """파서 List(None) 포함될 수도 있다. 각각 input_row의 항목에 적절한 파서를 적용"""
    return [parser(value) if parser is not None
            else value for value, parser in zip(input_row, parsers)]
```

Parsers : 한 row를 parsers 기준에 따라 나누어줌  
(parsing)

```
def parse_rows_with(reader, parsers):
    """각 열에 파서를 적용하기 위해 reader를 치환"""
    for row in reader:
        yield parse_row(row, parsers)
```

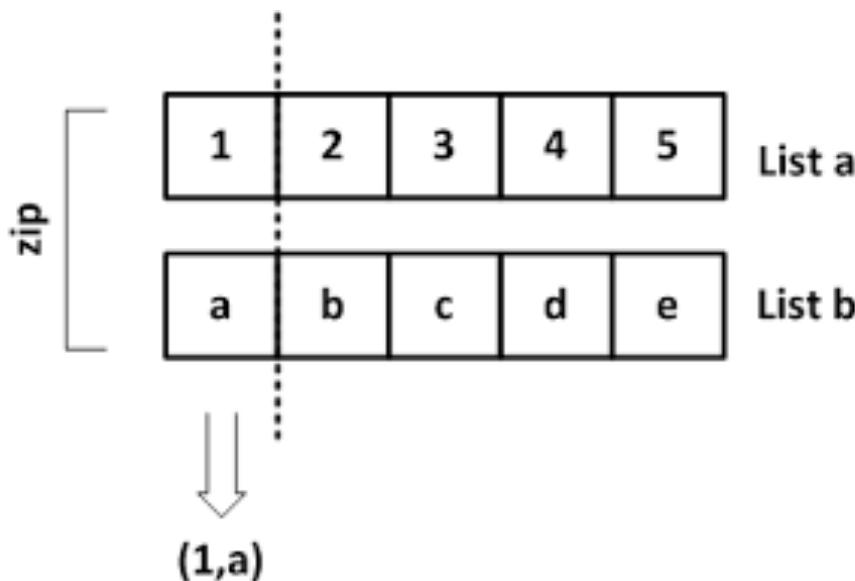
Parser가 not None이면? / None 이면?

앞에서 정의한 parse\_row 함수를 전체 row에 적용

# 03 데이터 다루기

## 10.2 정제하고 합치기 (1)

zip : 여러 리스트를 짹지어 주는 역할



# 03 데이터 다루기

## 10.2 정제하고 합치기 (2)

"" 2. 나쁜 데이터가 포함되어 있을 때 보완하기 """

```
def try_or_none(f):
    """f가 하나의 입력값을 받는다고 가정하고,
    오류가 발생하면 f는 None을 반환해주는 함수로 치환하자"""
    def f_or_none(x):
        try: return f(x)
        except: return None
    return f_or_none

def parse_row_new(input_row, parsers):
    return [try_or_none(parser)(value) if parser is not None
            else value for value, parser in zip(input_row, parsers)]

def parse_rows_with_new(reader, parsers):
    """각 열에 파서를 적용하기 위해 reader를 치환"""
    for row in reader:
        yield parse_row_new(row, parsers)
```

Try 블록 오류 발생 시 except 블록 실행

Parsers가 not None 이면? / None 이면?

# 03 데이터 다루기

## 10.2 정제하고 합치기 (3)

```
import csv
import codecs

data = []

with codecs.open("HR_comma_sep.csv", "rb", "utf-8") as f:
    reader = csv.reader(f)
    for line in parse_rows_with_new(reader, [None, None]):
        data.append(line)

print(data)
print(type(data[2][0]))
```

실제 파일 읽는 부분

'Data'라는 list 출력 시 어떤 모양?  
- parsers를 다른 모양으로 바꾸면?  
- Parsers부분에서 None의 의미?

# 03 데이터 다루기

## 10.3 데이터 처리 (0)

```
"""10.3 데이터 처리"""

''' 0. 파일 읽어오기'''

import a
import csv
import codecs

whole = []

with codecs.open("HR_comma_sep1.csv", "rb", "utf-8") as f:
    reader = csv.reader(f)
    for line in a.parse_rows_with_new(reader, [None, None, None, None, None, None, None, None, None, None]):
        whole.append(line)

data = []
keys = whole[0]
for contents in whole[1:]:
    values = contents
    dictionary = dict(zip(keys, values))
    data.append(dictionary)

print(data)
```

### 파일 읽는 부분

- 앞의 파일 함수를 사용하기 위해서 import
- “HR\_comma\_sep1.csv”라는 파일을 읽음

### ‘Data’라는 list 출력 시 어떤 모양?

- 각 element가 key, value 값을 가지는 dictionary

# 03 데이터 다루기

## 10.3 데이터 처리 (1)

```
""" 1. accounting 부서의 average_monthly_hours 최고치 찾기 """
```

```
max_avg_hours_company = max(row["average_monthly_hours"]
    for row in data
    if row["sales"] == "accounting")

print(max_avg_hours_company)
```

```
""" 2. 모든 부서의 최고 average_monthly_hours 찾기 """
```

```
from collections import defaultdict

# 부서(sales)을 기준으로 행을 그룹화
by_sales = defaultdict(list)
for row in data:
    by_sales[row["sales"]].append(row)

# list comprehension으로 각 그룹의 최고치 계산
max_avg_hours_by_sales = {sales: max(row["average_monthly_hours"]
    for row in grouped_rows)
    for sales, grouped_rows in by_sales.items()}

print(max_avg_hours_by_sales)
```

Data list의 각 성분(row) 중에서 row의 “sales”라는 키값이 accounting과 같은 row  
- 골라내고  
- 그 중 최고 값 구해냄

Defaultdict : dictionary에 기본값 정의 (키값)이 없더라도 에러를 출력하지 않고 기본값을 출력

# 03 데이터 다루기

## 10.3 데이터 처리 (2)

```
"""3. dict의 특정 필드를 갖고 오는 함수/ 여러 dict에서 동일한 필드를 갖고 오는 함수"""

def picker(field_name):
    """dict의 특정 필드를 선택해 주는 함수를 반환"""
    return lambda row: row[field_name]
```

```
def pluck(field_name, rows):
    """dict list를 필드 리스트로 변환"""
    return map(picker(field_name), rows)
```

```
"""4. grouper 함수 : 여러 행을 하나의 그룹으로 묶어주기"""

def group_by(grouper, rows, value_transform = None):
    # key는 grouper의 결과값이며 value는 각 그룹에 속하는 모든 행의 list
    grouped = defaultdict(list)
    for row in rows:
        grouped[grouper(row)].append(row)
    if value_transform is None:
        return grouped
    else:
        return {key: value_transform(rows)
                for key, rows in grouped.items()}
```

```
# 앞에서 max_avg_hours_by_sales를 나타내었던 것을 더욱 간단히 나타낼 수 있다
max_avg_hours_by_sales2 = group_by(picker("sales"),
                                     data,
                                     lambda rows: max(pluck("average_monthly_hours", rows)))
```

Picker :  
Pluck :

Group\_by :

# 03 데이터 다루기

## 10.4 척도 조절 (0)

- 사전 작업 :
  - 1) 필요한 모듈 불러오기
  - 2) 필요한 함수 정의하기
    - shape
    - make\_matrix

# 03 데이터 다루기

## 10.4 척도 조절 (0)

```
def get_column(A, j):
    return [A_i[j] # jth element of row A_i
            for A_i in A]

def shape(A): # 행 갯수 열 갯수 반환 / np.shape()
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0 # number of elements in first row
    return num_rows, num_cols

def make_matrix(num_rows, num_cols, entry_fn): # 행렬 만드는 함수
    """returns a num_rows x num_cols matrix
    whose (i,j)th entry is entry_fn(i, j)"""
    return [[entry_fn(i, j) # given i, create a list
            for j in range(num_cols)] # [entry_fn(i, 0), ...]
            for i in range(num_rows)] # create one list for each i
```

Shape (A)

: A라는 행렬의 행 개수, 열개수를 반환

Make\_matrix (행갯수, 열갯수, 특정 성분)

: 행과 열의 개수와 특정 성분을 받아 새로운 행렬 생성

# 03 데이터 다루기

## 10.4 척도 조절 (0)

```
# 파일 읽어오기

whole = []
f = open("HR_comma_sep2.csv", "r", encoding="utf-8")
lines = f.readlines()

for line in lines:
    new_element = line.split(',')[:5] # 우리가 분석할
    whole.append(new_element)

data = []
for contents in whole[1:]:
    contents_int = [float(i) for i in contents]
    data.append(contents_int)

print(data)
```

“HR\_comma\_sep2\_csv” 파일을  
읽음

satisfaction\_level/  
last\_evaluation/  
number\_project/  
average\_montly\_hours/  
time\_spend\_company 부분만 추출

첫 줄 (구분 항목)을 제외한 다른 줄을  
float type의 원소를 가진 리스트로 만듦  
그 후, data라는 리스트에 추가  
-> “data”라는 list는 행렬이 됨

# 03 데이터 다루기

## 10.4 척도 조절 (1)

""" 1. 각 열의 평균과 표준편차 계산 """

```
def scale(data_matrix):
    """각 열의 평균과 표준편차를 반환"""
    num_rows, num_cols = shape(data_matrix)
    means = [mean(get_column(data_matrix, j))
              for j in range(num_cols)]
    std devs = [standard_deviation(get_column(data_matrix, j))
                for j in range(num_cols)]
    results = means, std devs
    return list(results)

print(scale(data))
```

Get\_column (A, j)

: A라는 행렬의 j열을 반환

각 열의 평균을 구함

각 열의 표준 편차를 구함

Tuple로 나온 results를

List로 바꾸어 반환

# 03 데이터 다루기

## 10.4 척도 조절 (2)

```
""" 2. 계산된 평균과 표준편차를 이용해, 표준정규분포를 이용한 행렬 만들기 """
```

```
# 각 차원의 평균을 0, 표준편차를 1로 변환하여 척도를 조절
```

```
def rescale(data_matrix):
```

```
    """각 열의 평균을 0, 표준편차를 1로 변환하면서
```

```
    입력되는 데이터의 척도를 조절
```

```
    편차가 없는 열은 그대로 유지"""
means, stdevs = scale(data_matrix)
```

```
def rescaled(i, j):
```

```
    if stdevs[j] > 0:
```

```
        return (data_matrix[i][j] - means[j]) / stdevs[j]
```

```
    else:
```

```
        return data_matrix[i][j]
```

```
num_rows, num_cols = shape(data_matrix)
```

```
💡 return make_matrix(num_rows, num_cols, rescaled)
```

```
print(rescale(data))
```

평균과 표준편차는 앞에서 scale  
함수로 구한 값을 그대로 사용

주어진 행렬의 행 개수와 열 개수,  
rescaled 처리한 값을 원소로 받아  
행렬을 새로 만듦

# 03 데이터 다루기

## 10.5 차원 축소 (0)

- 사전 작업 :
  - 1) 필요한 모듈 불러오기
  - 2) 필요한 함수 정의하기
  - 3) data 정의하기
    - 교재에서 주어진 데이터 활용

# 03 데이터 다루기

## 10.5 차원 축소 (1)

```
""" 1. 각 차원의 평균이 0이 되도록 데이터를 바꿈 """
```

```
def de_mean_matrix(A):
    """A의 모든 값에서 각 열의 평균을 빼준 행렬을 반환
    반환된 행렬의 모든 열의 평균은 0"""
    nr, nc = shape(A)
    column_means, _ = scale(A)
    return make_matrix(nr, nc, lambda i, j: A[i][j] - column_means[j])
```

A 행렬의 각 열의 원소에 해당 열의 평균값을 뺀,  
새로운 행렬 만듦

# 03 데이터 다루기

## 10.5 차원 축소 (2)

```
""" 2. 방향 찾아내기 """
```

# 1) 길이가 1로 정규화된, 방향을 나타내는 벡터

```
def direction(w):
    mag = magnitude(w)
    return [w_i / mag for w_i in w]
```

# 예를 들어, direction([1,2])의 값은 [1/루트5, 2/루트5]

방향을 나타내며 길이를 1로 정규화시킨  
벡터 만들기

# 2) 길이가 1로 정규화된 방향을 나타내는 벡터

```
def directional_variance_i(x_i, w): # 각 데잍에 대한 편차 계산
    """ w가 나타내는 방향에서 x_i 행의 편차를 반환 """
    return dot(x_i, direction(w)) ** 2
```

위에서 정규화시킨 방향벡터가  
나타내는 방향으로 데이터의 편차 계산

```
def directional_variance(X, w): # 전체 데이터에 대해 계산
    """ w가 나타내는 방향에서 데이터 전체의 편차를 반환 """
    return sum(directional_variance_i(x_i, w) for x_i in X)
```

# 03 데이터 다루기

## 10.5 차원 축소 (2)

```
# 3) 편차를 최대화 시키는 방향을 찾기 위한 경사 하강법 사용
# 편차를 최대화 시키는 이유는, 모든 데이터를 최대한 포괄하는 방향을 구하기 위함
def directional_variance_gradient_i(x_i, w): # 한 데이터가 벡터 w의 방향에 기여하는 부분
    """방향의 경사(w의 기울기)에 x_i가 기여하는 부분"""
    projection_length = dot(x_i, direction(w))
    return [2 * projection_length * x_ij for x_ij
            in X]

def directional_variance_gradient(X, w): # 전체 데이터가 벡터 w의 방향에 기여하는 부분
    return vector_sum(directional_variance_gradient_i(x_i, w) for x_i in X)
```

데이터가  
정규화된 방향벡터의 방향에 기여하는 정도

```
# 4) 제1주성분 구하기
def first_principal_component(X):
    """제 1 주성분은 directional_variance를 최대화"""
    guess = [1 for _ in X[0]]
    unscaled_maximizer = maximize_batch(
        partial(directional_variance, X),
        partial(directional_variance_gradient, X),
        guess)
    return direction(unscaled_maximizer)
```

경사 하강법을 통해  
Directional\_variance를 최대화 시키는  
정규화된 방향벡터인 제1주성분 구하기  
- 왜 directional\_variance를 최대화?

```
print(first_principal_component(X)) # 1번째 주성분
```

# 03 데이터 다루기

## 10.5 차원 축소 (2)

```
""" 2. 방향 찾아내기 """
```

# 1) 길이가 1로 정규화된, 방향을 나타내는 벡터

```
def direction(w):
    mag = magnitude(w)
    return [w_i / mag for w_i in w]
```

# 예를 들어, direction([1,2])의 값은 [1/루트5, 2/루트5]

방향을 나타내며 길이를 1로 정규화시킨  
벡터 만들기

# 2) 길이가 1로 정규화된 방향을 나타내는 벡터

```
def directional_variance_i(x_i, w): # 각 데잍에 대한 편차 계산
    """ w가 나타내는 방향에서 x_i 행의 편차를 반환 """
    return dot(x_i, direction(w)) ** 2
```

위에서 정규화시킨 방향벡터가  
나타내는 방향으로 데이터의 편차 계산

```
def directional_variance(X, w): # 전체 데이터에 대해 계산
    """ w가 나타내는 방향에서 데이터 전체의 편차를 반환 """
    return sum(directional_variance_i(x_i, w) for x_i in X)
```

# 03 데이터 다루기

## 10.5 차원 축소 (3)

"" 3. 제 1 주성분에 해당하는 방향을 찾은 후, 데이터를 주성분에 투영시켜 해당 주성분위에서 어디에 위치하는가 찾기""

# 1) 한 데이터를 주성분의 방향에 투영시키기

```
def project(v, w):
```

""" $v$ 를  $w$ 방향으로 투영, 즉 해당 데이터가 해당 주 성분 위의 어디에 위치하는가를 구함"""

```
coefficient = dot(v, w)
```

```
return scalar_multiply(coefficient, w) # 이 결과 벡터가
```

벡터  $v$ 를 주성분의 방향으로 투영하여,  
해당 벡터가 나타내는 데이터가  
주성분의 방향에서 어디쯤에 위치하는지 구함

- 1) X의 가장 첫 원소벡터가 제1주성분의 방향에 투영되어 나온 새 벡터
- 2) 새로 나온 벡터의 길이 ( 다시 보게 될 것 )
- 3) 새로 나온 벡터의 방향 = 제1 주성분 벡터의 방향

```
print(project([20.9666776351559, -13.1138080189357], first_principal_component(X))) # 투영되어 나온 벡터
```

```
print(magnitude(project([20.9666776351559, -13.1138080189357], first_principal_component(X)))) # 투영되어 나온 벡터의 길이
```

```
print(direction(project([20.9666776351559, -13.1138080189357], first_principal_component(X)))) # 투영되어 나온 벡터의 방향
```

# 03 데이터 다루기

## 10.5 차원 축소 (3)

# 2) 다른 주성분을 구하기 위해 투영된 데이터 제거

```
def remove_projection_from_vector(v, w): # 하나의 데이터에 대해 제거
    """v에서 w로 투영시킨 결과를 빼줌"""
    return vector_subtract(v, project(v, w))
```

```
print(remove_projection_from_vector([20.9666776351559, -13.1138080189357], project([20.9666776351559, -13.1138080189357], first_principal_component(X))))
```

`def remove_projection(X, w): # 모든 데이터에 대해 제거`

"""X의 각 행을 w로 투영시키고 각 행에서 투영시킨 값을 빼줌""""

```
return [remove_projection_from_vector(x_i, w) for x_i in X]
```

```
print(remove_projection(X, first_principal_component(X))) # 이 결과 원래
```

# 3) 제2주성분 구하기

```
print(first_principal_component(remove_projection(X, first_principal_component(X)))) # 이 결과 2번째 주성분을 구할 수 있다
```

다른 주성분을 구하기 위해  
기준에 투영된 데이터 제거하여  
새 데이터 형성  
(책 그림 10.8 참고)

제2주성분 구하기  
: 위에서 구한 새 데이터 활용하여  
제2 주성분 구함

# 03 데이터 다루기

## 10.5 차원 축소 (4)

```
""" 4. 반복적인 과정을 통해 원하는 만큼 많은 주성분 구하기 """
```

```
# 1) num of components는 반환할 주성분의 개수를 뜻함
def principal_component_analysis(X, num_components):
    components = []
    for _ in range(num_components):
        component = first_principal_component(X)
        components.append(component)
        X = remove_projection(X, component)
    return components
```

Num\_components 만큼 주성분 구함  
- 1이면 주성분 1개 반환  
- 2이면 주성분 2개 반환

```
print(principal_component_analysis(X, 1)) # 숫자 2 이므로 주성분 2개를 반환
```

# 03 데이터 다루기

## 10.5 차원 축소 (4)

```
# 2) 주어진 데이터를 저차원 공간에서 생성 시키기
```

```
# 벡터 하나를 하나의 주성분 위에다 투영
```

```
def transform_vector(v, components):
    return [dot(v, w) for w in components]
```

```
print(transform_vector([20.9666776351559, -13.1138080189357], [[0.8483153298152323, -0.529491360836486]]))
```

# X의 가장 첫 원소를 제 1 주성분 위에다 투영했을 경우

# 이는 3에서 구한 magnitude(project([20.9666776351559, -13.1138

# 데이터 전체를 여러 주성분 위에다 투영

```
def transform(X, components):
    return [transform_vector(x_i, components) for x_i in X]
```

```
print(transform(X, [[0.8483153298152323, -0.529491360836486], [0.5294915482155131, 0.8483152128591935]]))
```

# 데이터 전체를 앞에서 구한 두 개의 주성분 위에다 투영했을 경우

# [X의 첫번째 원소를 제 1 주성분 위에 투영했을 경우, X의 첫번째

**벡터 하나를  
앞에서 구한 제1주성분 위에 투영**

X의 가장 첫 원소벡터를  
제1주성분 위에 투영한 결과는  
(3)에서 구한 ‘새로 나온 벡터의 길이’ 와 동일

**데이터 전체를  
앞에서 구한 전체 주성분들 위에 투영**

[X의 i번째 원소를 제 1 주성분 위에 투영했을 경우,  
X의 i번째 원소를 제 2 주성분 위에 투영했을 경우]  
와 같은 원소를 가진 list로 결과가 나온다.

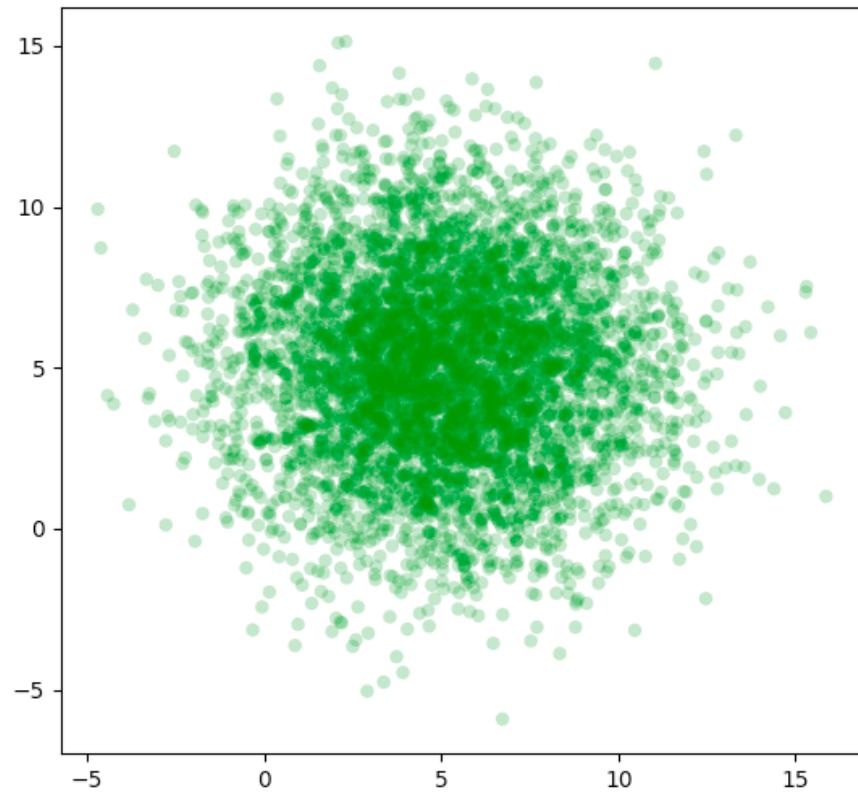
# 04 데이터 시각화 고급기술

더 멀리 가기

# 04 데이터 시각화 고급기술

## Scatter plot

- `matplotlib.pyplot.scatter( )`

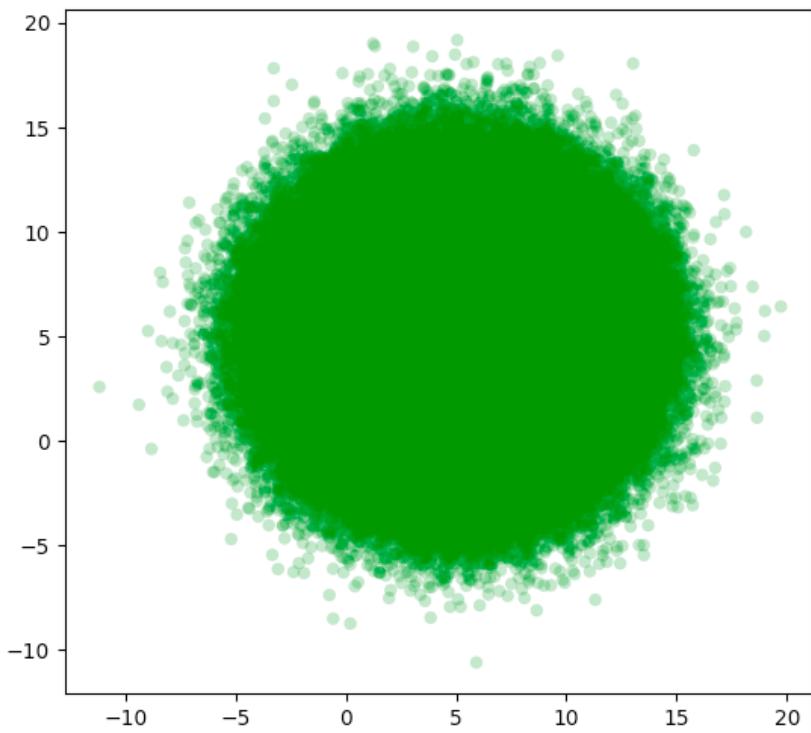


# 04 데이터 시각화 고급기술

## Scatter plot

$10^5$  data points.

어떤 분포인지 파악X



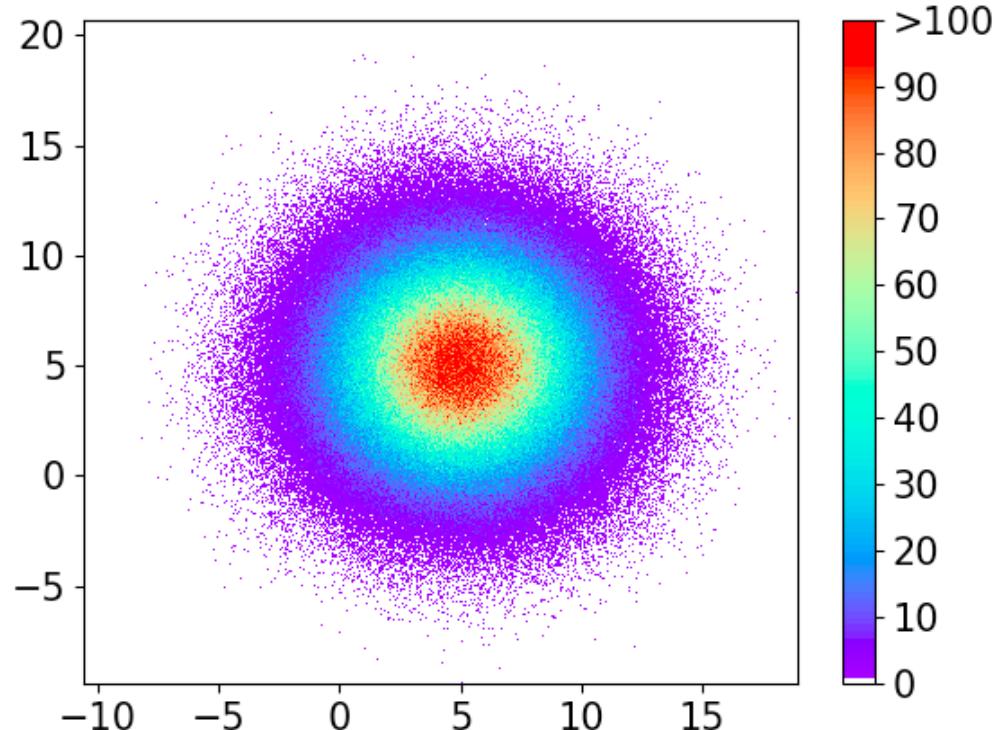
# 04 데이터 시각화 고급기술

## Scatter plot -> Heat Map

point들의 밀도를 Heatmap으로 만들기.

normal한 분포임을 확인할 수 있다!

numpy.histogram2d( )



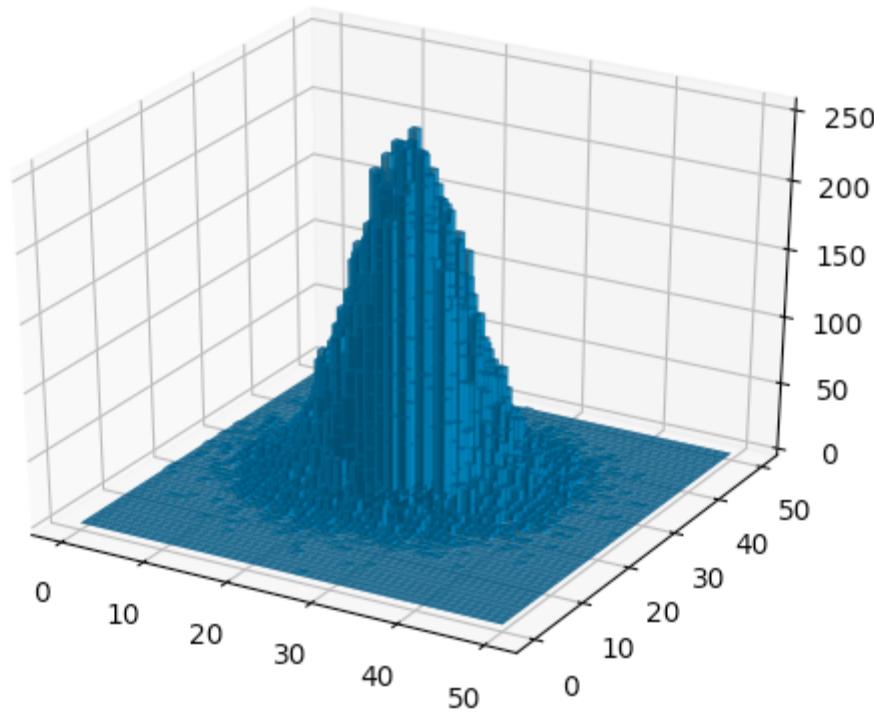
# 04 데이터 시각화 고급기술

## 2. Scatter plot -> Heat Map -> 3Dplot

물론 3D plot도 가능

Data

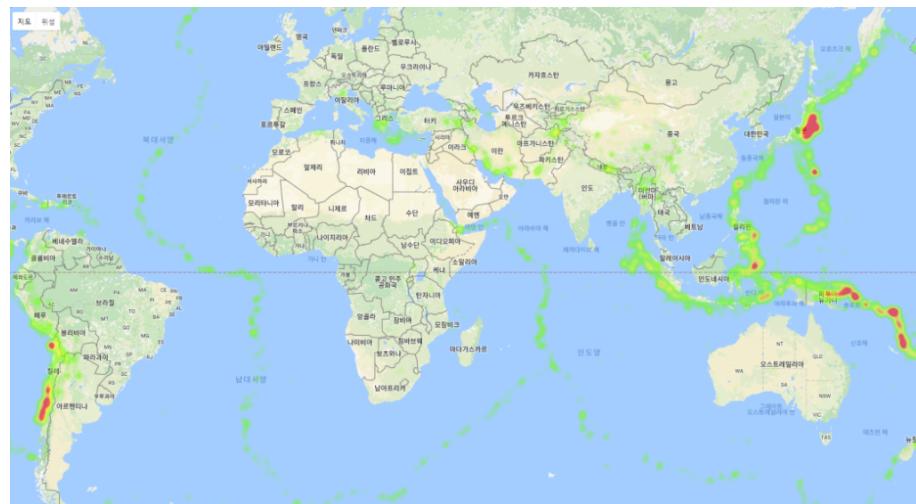
- ☞ np.histogram2d
- ☞ Mpl\_toolkits.mplot3d
- ☞ matplotlib.pyplot.add\_subplot  
(111, projection='3d')



# 04 데이터 시각화 고급기술

## 지도 데이터 시각화

gmplot.heatmap( )

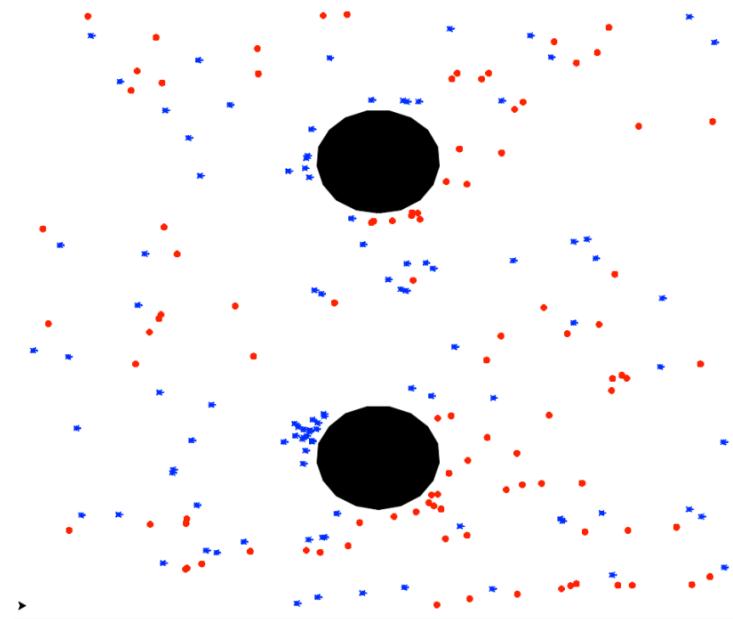


# 04 데이터 시각화 고급기술

## turtle

Agent Based Model 등을 시뮬레이션

Python3 X



# 04 데이터 시각화 고급기술

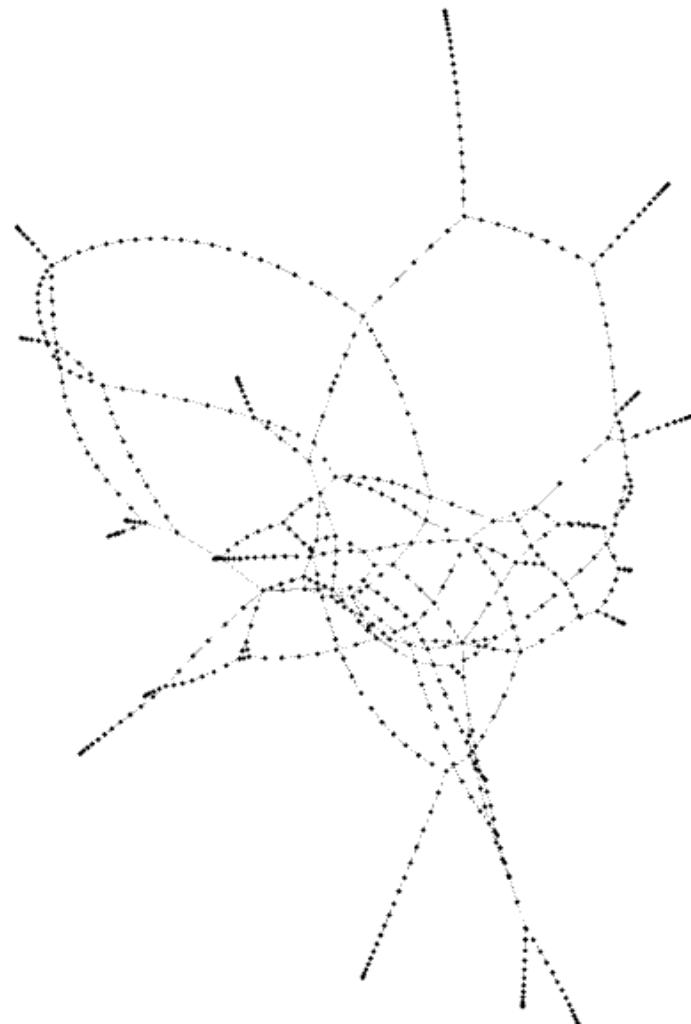
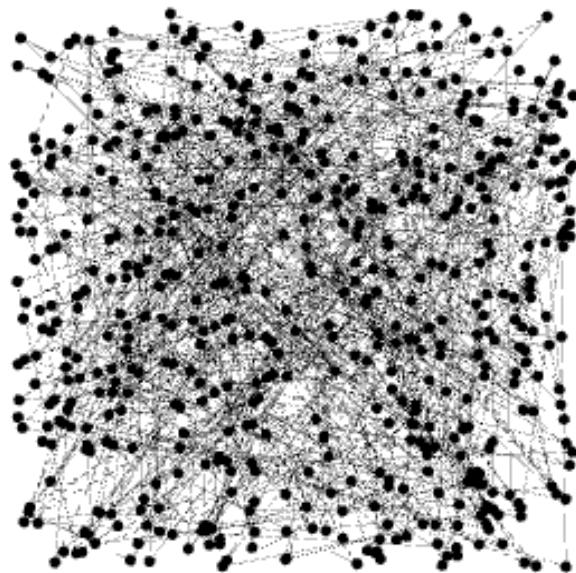
## Gephi

Network와 graph 시각화 프로그램

Python :

Import networkx &

nx.write\_gexf(Graph, 'filename.gexf')



# 04 데이터 시각화 고급기술

■ Quest : From my\_gps\_file plot something

1. Google 위치 데이터
2. Import json
3. Scatter plot

# 참고자료

2017 pycon Portland (데이터 시각화 지형)

<https://www.youtube.com/watch?v=OC-YdBz8Llw>

[https://www.slideshare.net/neofuture/sds-2?next\\_slideshow=1](https://www.slideshare.net/neofuture/sds-2?next_slideshow=1)

<https://datahero.com/blog/2013/08/06/line-or-bar-graph/>

[http://matplotlib.org/api/lines\\_api.html#matplotlib.lines.Line2D](http://matplotlib.org/api/lines_api.html#matplotlib.lines.Line2D)

<https://pandas.pydata.org/pandas-docs/stable/>

[http://matplotlib.org/api/\\_as\\_gen/matplotlib.axes.Axes.axvline.htm](http://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.axvline.htm)

<https://datascienceschool.net/view-notebook/4c2d5ff1caab4b21a708cc662137bc65/>

Seaborn 갤러리 (시각화 한번씩 보길 추천)

<http://seaborn.pydata.org/examples/index.html>

# THANK YOU !

# THANK YOU !