Visualization

Data Visualization

■ **정보를 명확하고 효율적으로 전달**

■ **데이터 분석을 통해 얻을 수 있는 척도는 데이터의 단일 측면을 나타낼 수는 있지만 모든 데이터 특성을 한 번에 표시하기는 어려움.**

○ 복잡한 데이터를 보다 쉽게 액세스하고 이해하며 사용

○ 비교 분석이나 인과 관계 이해와 같은 특정 분석 작업을 수행가능

■ **시각화 기법은 인간의 뇌에서 가장 높은 정보 처리 대역폭을 가짐.**

■ Presentation

○ Known facts about data

○ Task: Communicate results

■ Exploration

○ Data with hypothesis

○ Task: Generate hypothesis

■ Confirmation

○ Hypothesis is given

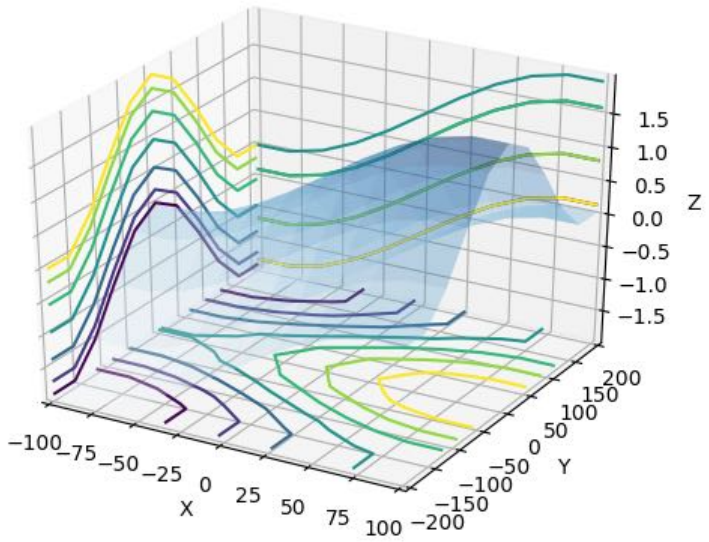
○ Task: Verify / falsify hypothesis

단계	설명
정보조직화 단계	•사용자의 정보 인지에 관여 •혼돈의 상태로 존재하는 데이터를 분류하고 배열하고 조직화하여 질서를 부여
정보시각화 단계	•사용자의 정보 지각에 관여 •보다 효율적으로 정보 전달을 위해 감각 기관에 최적의 자극을 제시하는 방법 제시
상호작용 단계	•정보와 사용자 간의 상호작용 측면의 사용자 경험을 디자인 •정보의 인지적 요인뿐만 아니라 지각적 요인을 함께 활용 •정보 시각화 단계와 밀접하게 연동되면서 동시에 입력 기술의 특성도 함께 고려

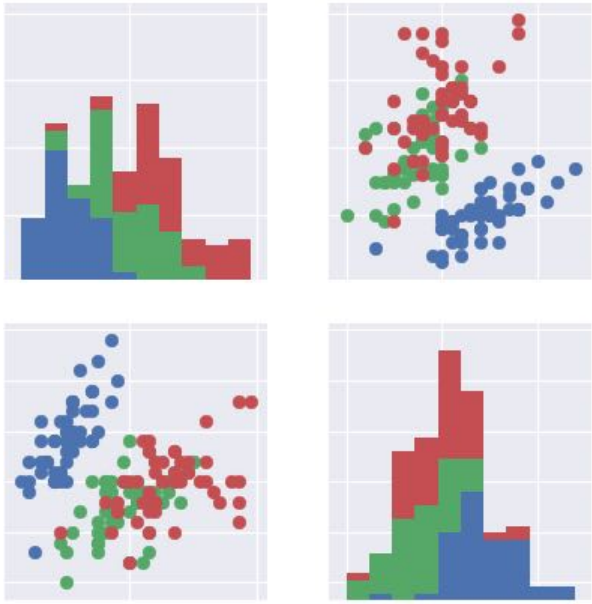
Rendering/ Grammar	특징	라이브러리
Python	• 데이터를 다루는 다양한 라이브러리가 있는 언어 • 다양한 범용 라이브러리	• Matplotlib • Pandas • Seaborn • ggplot
JavaScript	• 웹 기반의 interactive 시각화가 가능	• Bokeh • plot.ly
Vega	• Declarative language • Interactive한 시각화 • 복잡한 프로그래밍 없이 시각화	• Altair • Vincent • Pdvega (pandas vega)
SVG	• XML기반의 vector graphic format의 웹기반 • DOM, XSL과 같이, W3C standard에 포함됨	• pygal
OpenGL	• 현재 가장 많이 쓰이는 2D, 3D 그래픽 API	• vispy
QT	• GUI 어플리케이션 프로그래밍에 주로 사용	• pyqwt • pyqtgraph

이름	특징
Matplotlib	• 대표적 python 시각화 라이브러리 • Pandas와 Seaborn은 Matplotlib의 Wrappers • MATLAB의 기능과 문법을 가져옴 • Default 스타일이 1990년대 느낌이라는 비판을 받음
Pandas	• Labeled, relational data를 다루는 프레임워크 • Data Wrangling 기능을 주로 다루나, Visualization 기능도 포함
Seaborn	• Matplotlib 기반으로 구현 • 통계적 시각화에 특화 • 기본 스타일과 컬러 팔레트가 Matplotlib보다 비교적 현대적 • Numpy, DataFrame 자료구조 사용
ggplot	• R의 ggplot2 기반 • Grammar of graphics를 python으로 구현한 것 • Matplotlib 기반으로 구현 • Matplotlib과 다르게 축을 만들고, 점을 만들고, 선을 만드는 식으로 작동 • Pandas와 tightly integrate되어 DataFrame 자료구조 사용

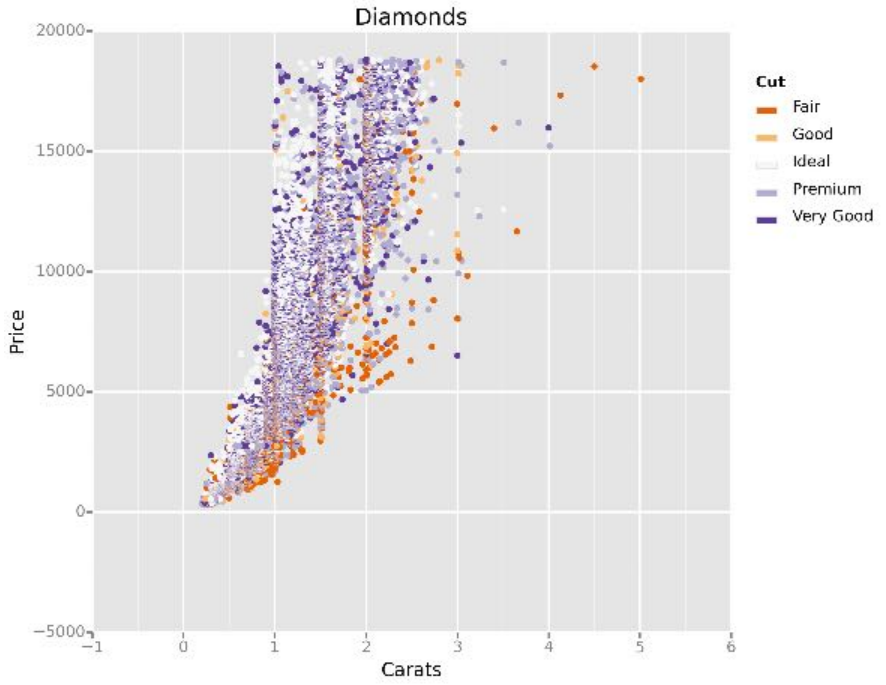
- Pylab interface + matplotlib API(multi backends)
- rich ecosystem of python tools built around it
- De Facto of Python
- The base of many libraries



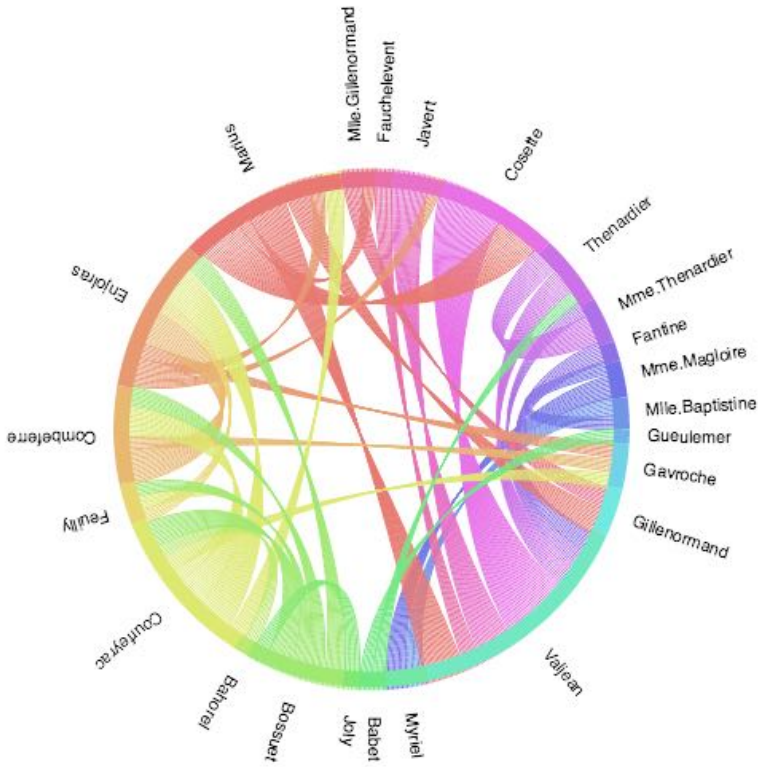
- Made with matplotlib
- Easy coding when using pandas
- Various color theme
- Various statistical graphs



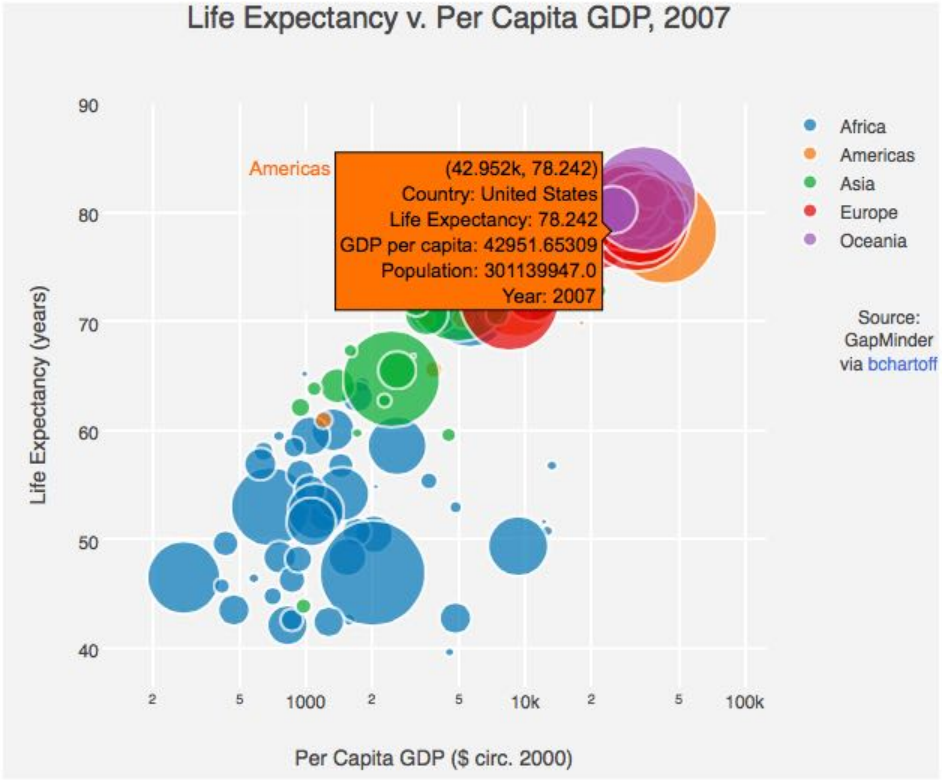
- R ggplot2 implemented in Python
- For users friendly of R
- Professional looking
- Minimal coding



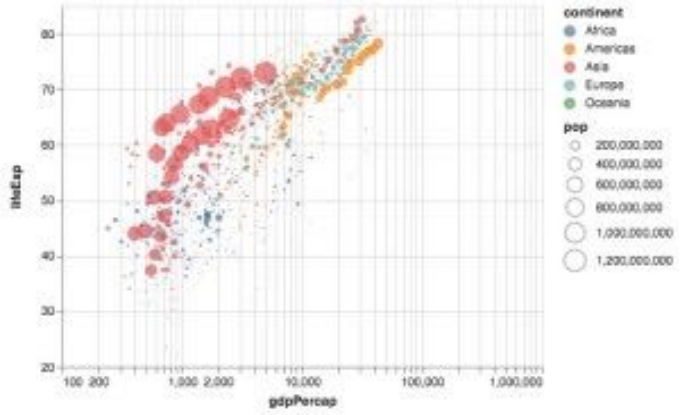
- Javascript based
- Interactive dashboard
- Graphs for various purposes



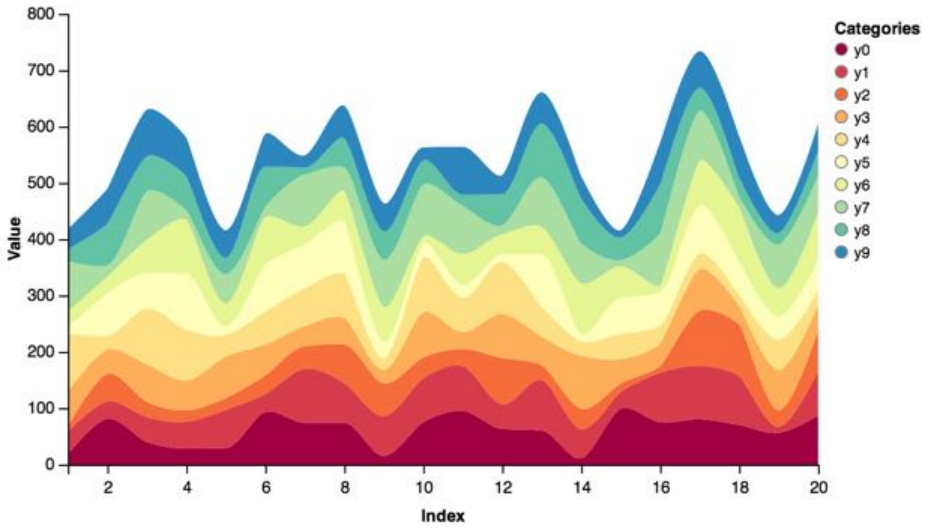
- Interactive(javascript)
- Live-streaming
- Integrations with Google



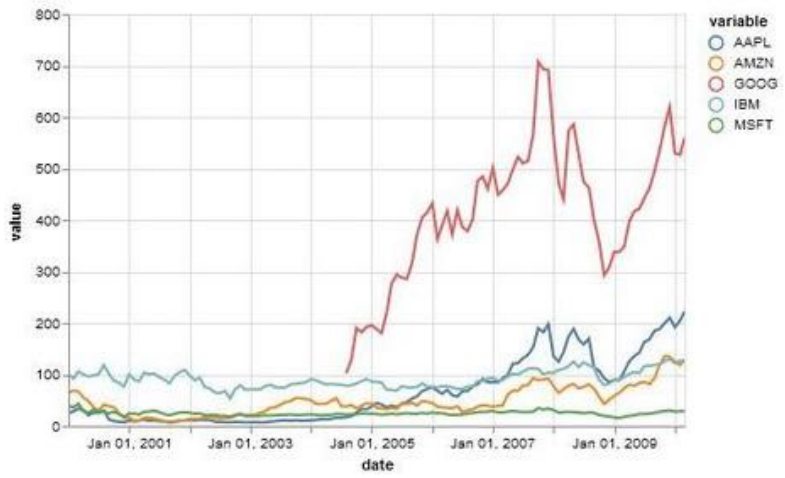
- Vega based
- Plot details are plotted automatically
- Wide range of statistical visualizations
- links between *data columns* and *visual encoding channels*,



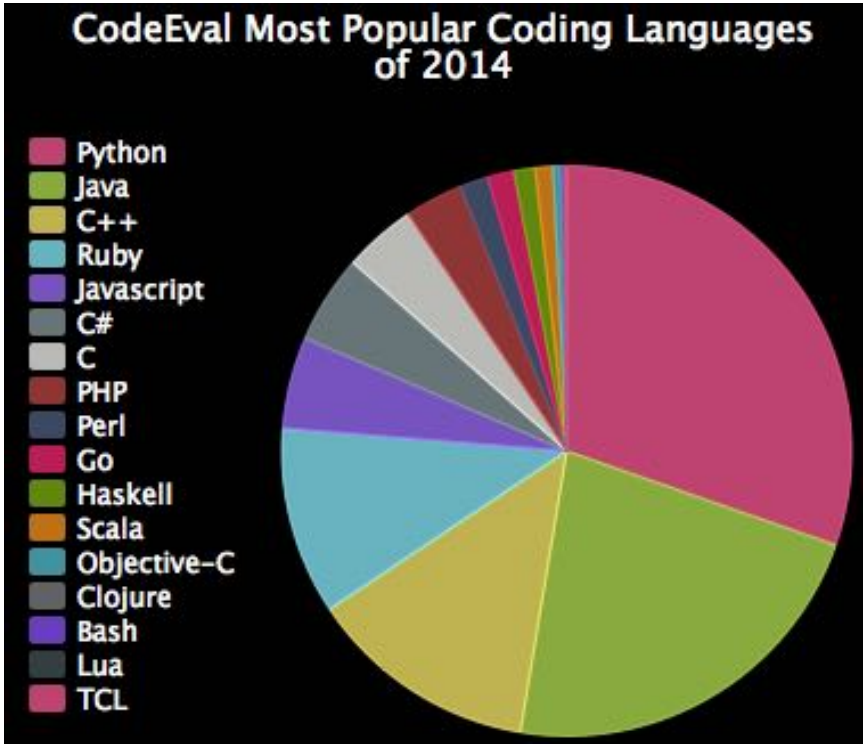
- Vega made
- convenience chart-building method
- quick plotting of DataFrames and Series



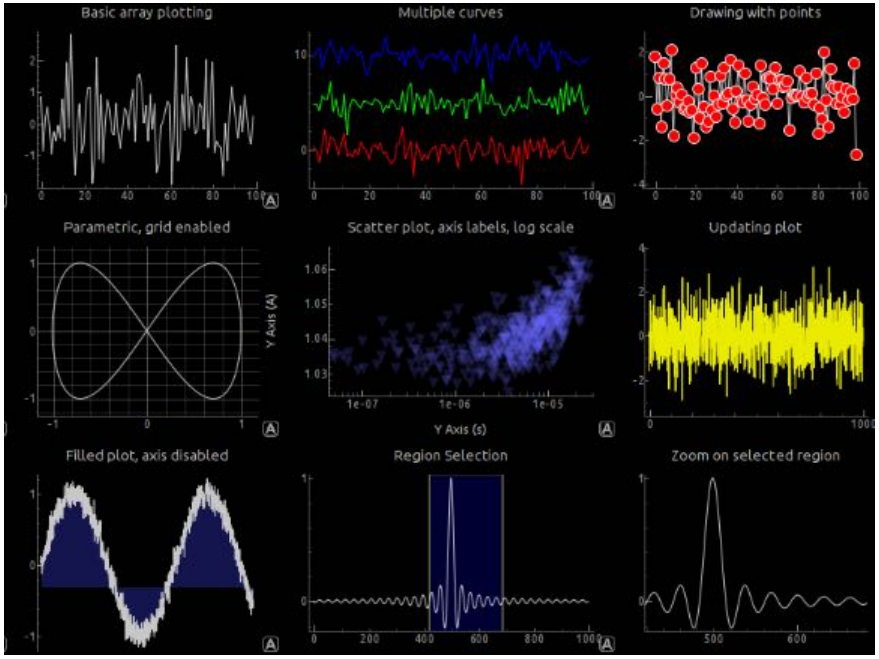
- Vega-lite made
- quickly create interactive plots
- easy use within the Jupyter notebook



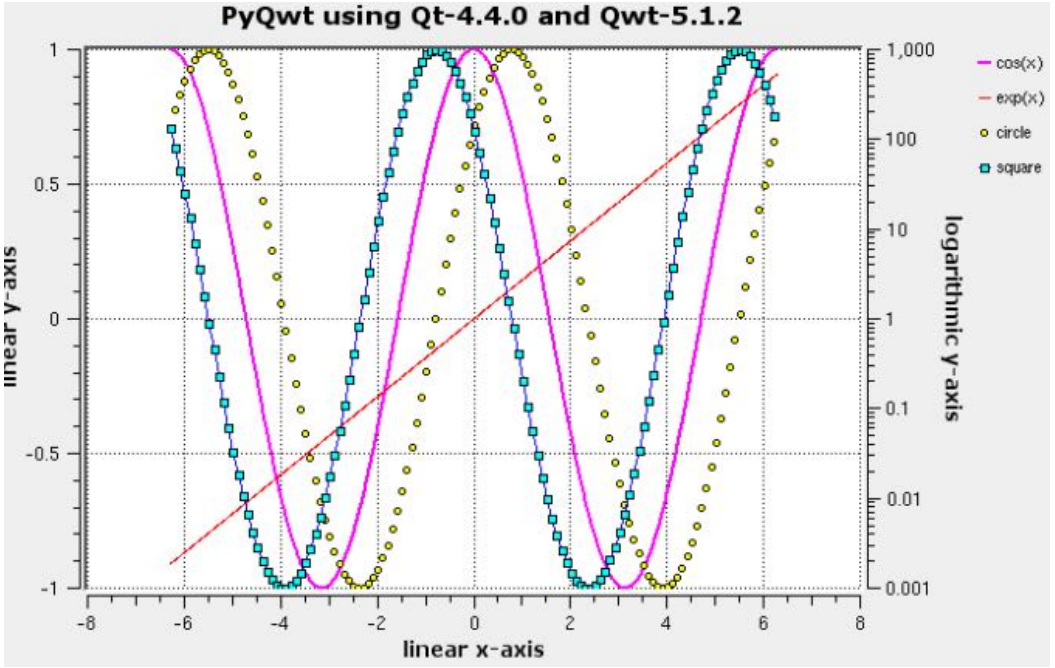
- SVG made
- 14 charts available
- Webpage imbedding



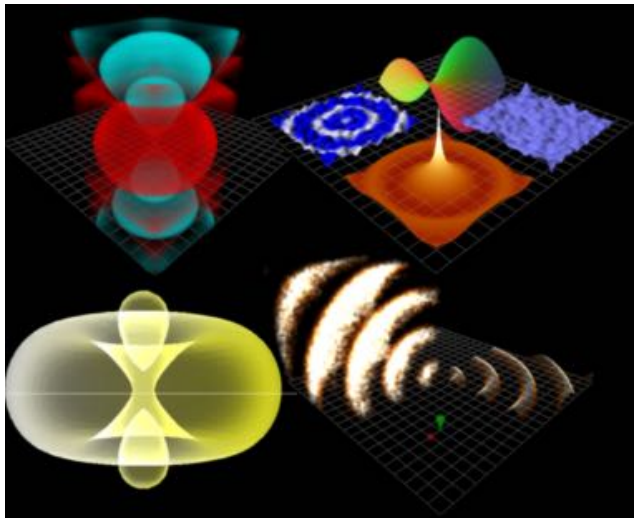
- Open GL made
- fast, scalable, and easy to use



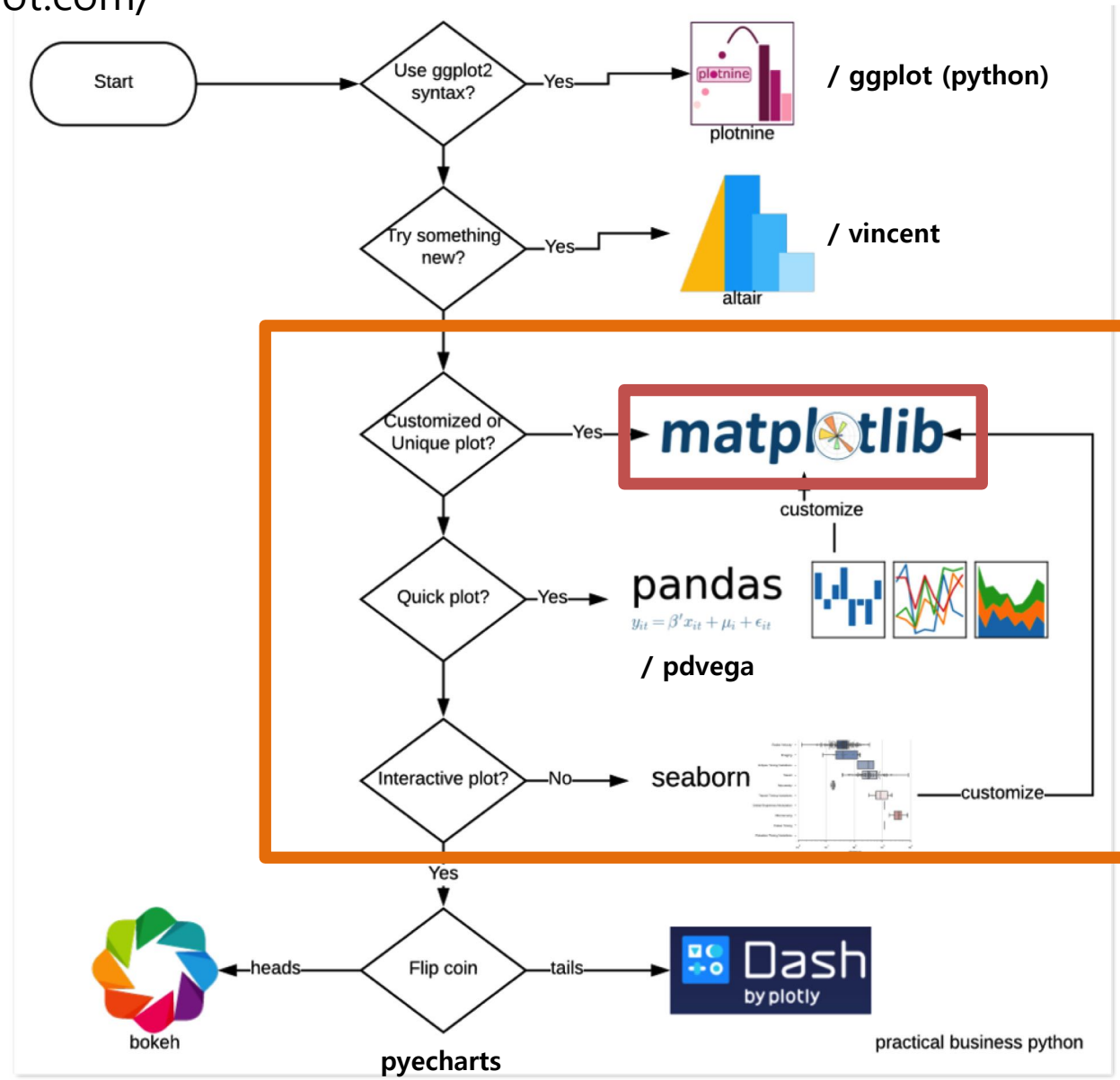
- Qt framework
- Qwt C++ class library
- PyQwt3D of 3D plots



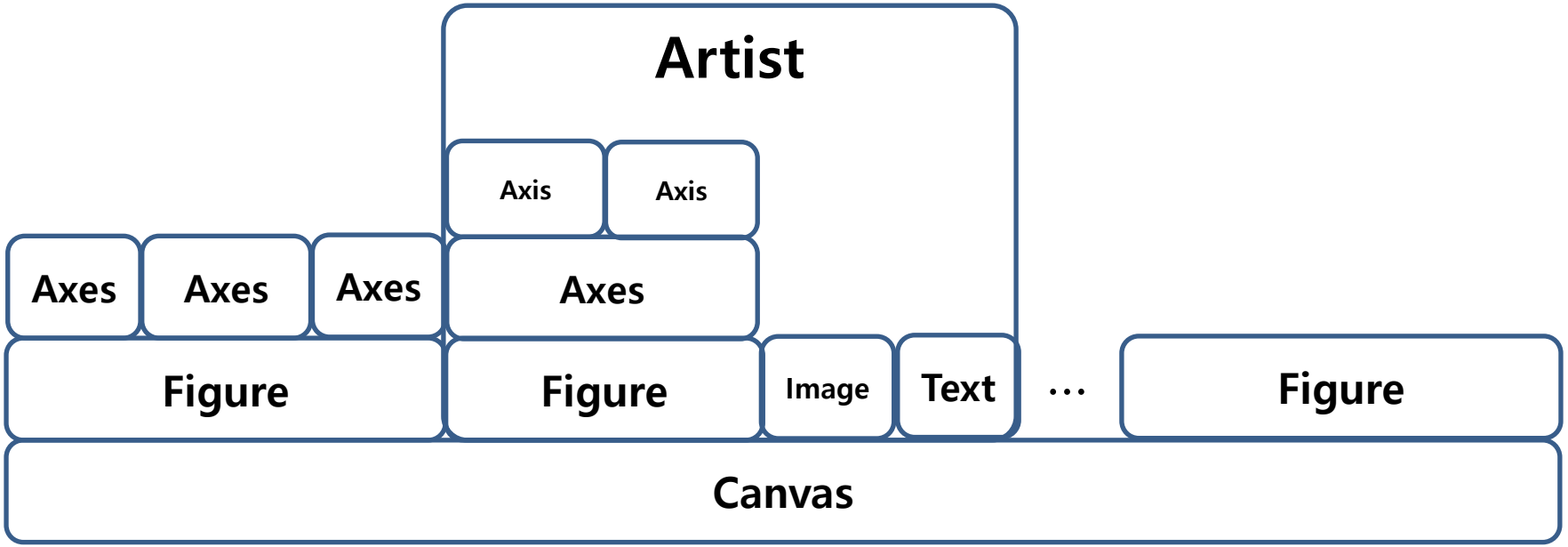
- Qt made
- For desktop programming
- Basic 2D plotting in interactive view boxes
- Easy to generate new graphics



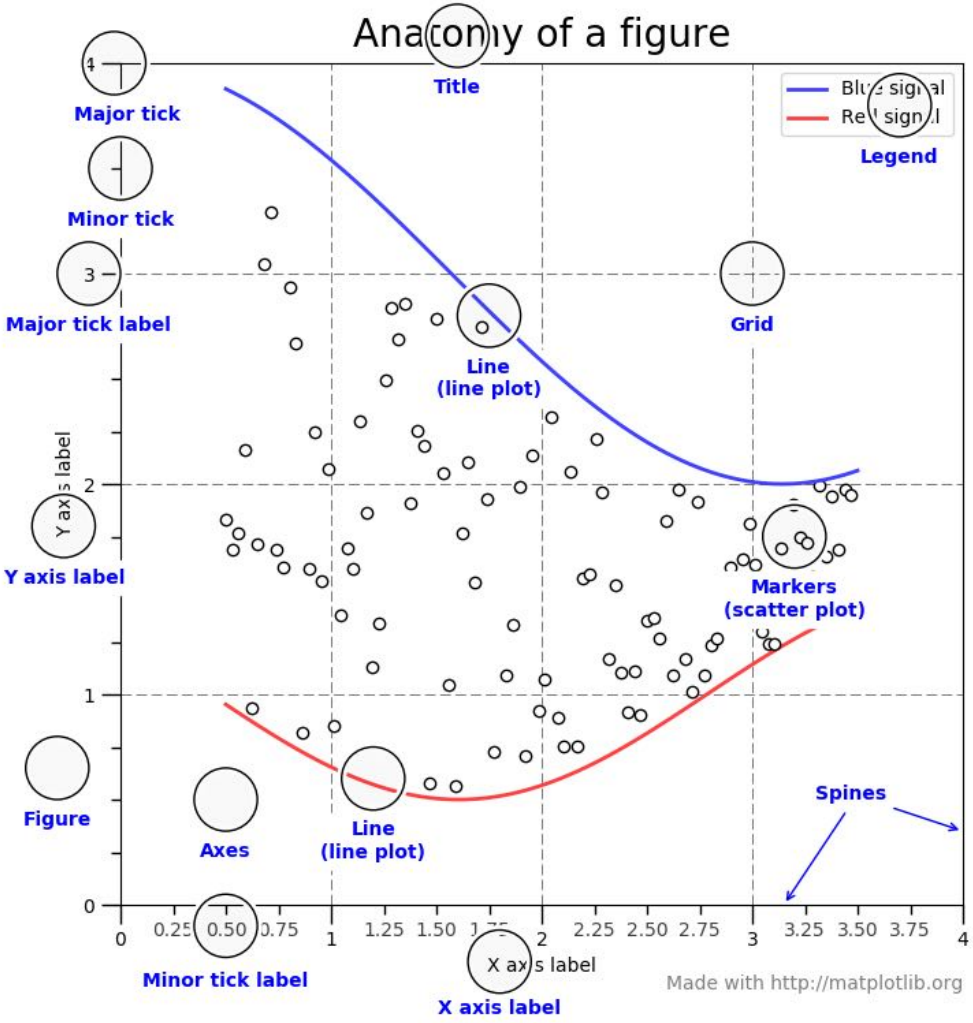
http://pythonplot.com/

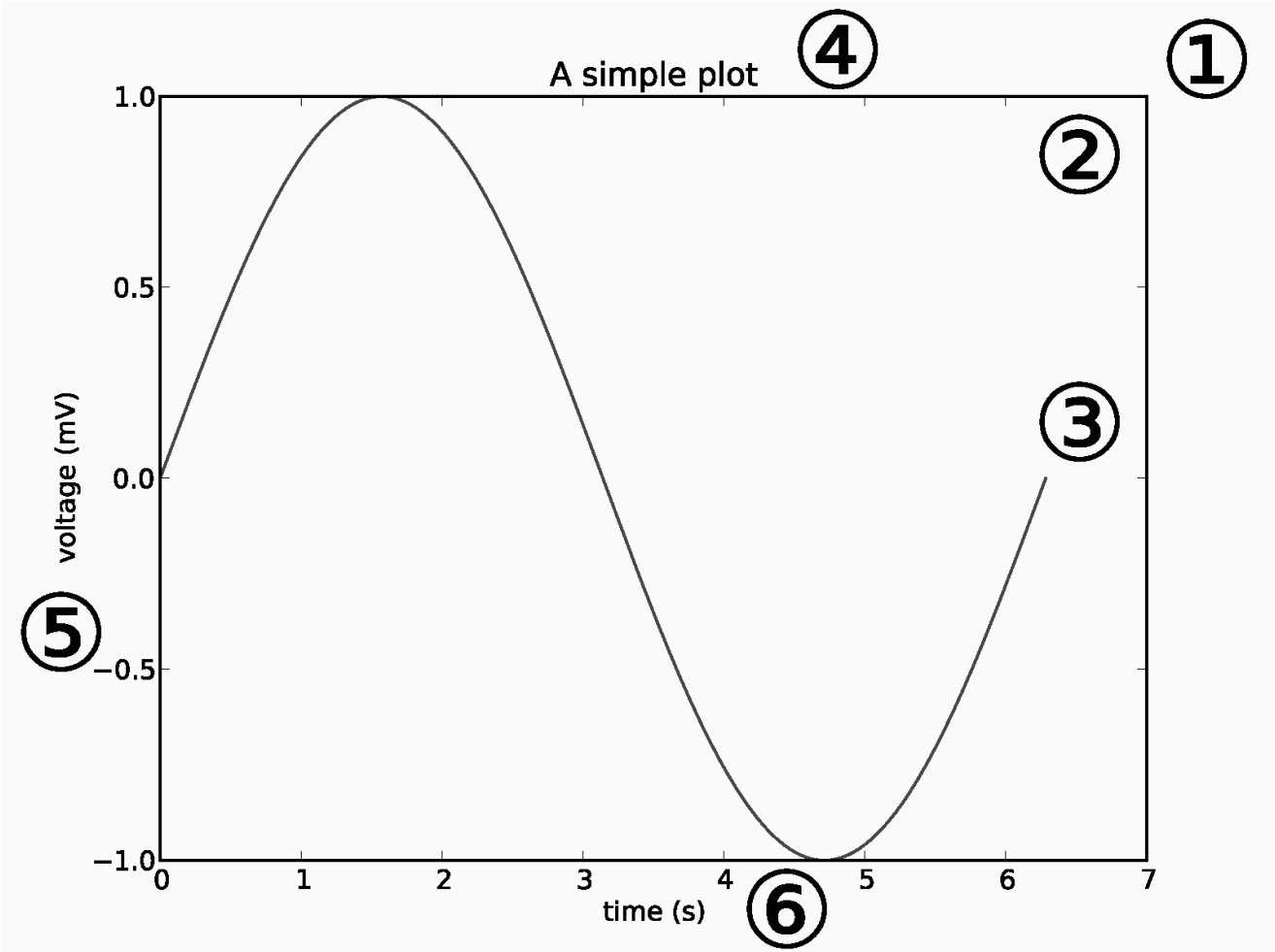


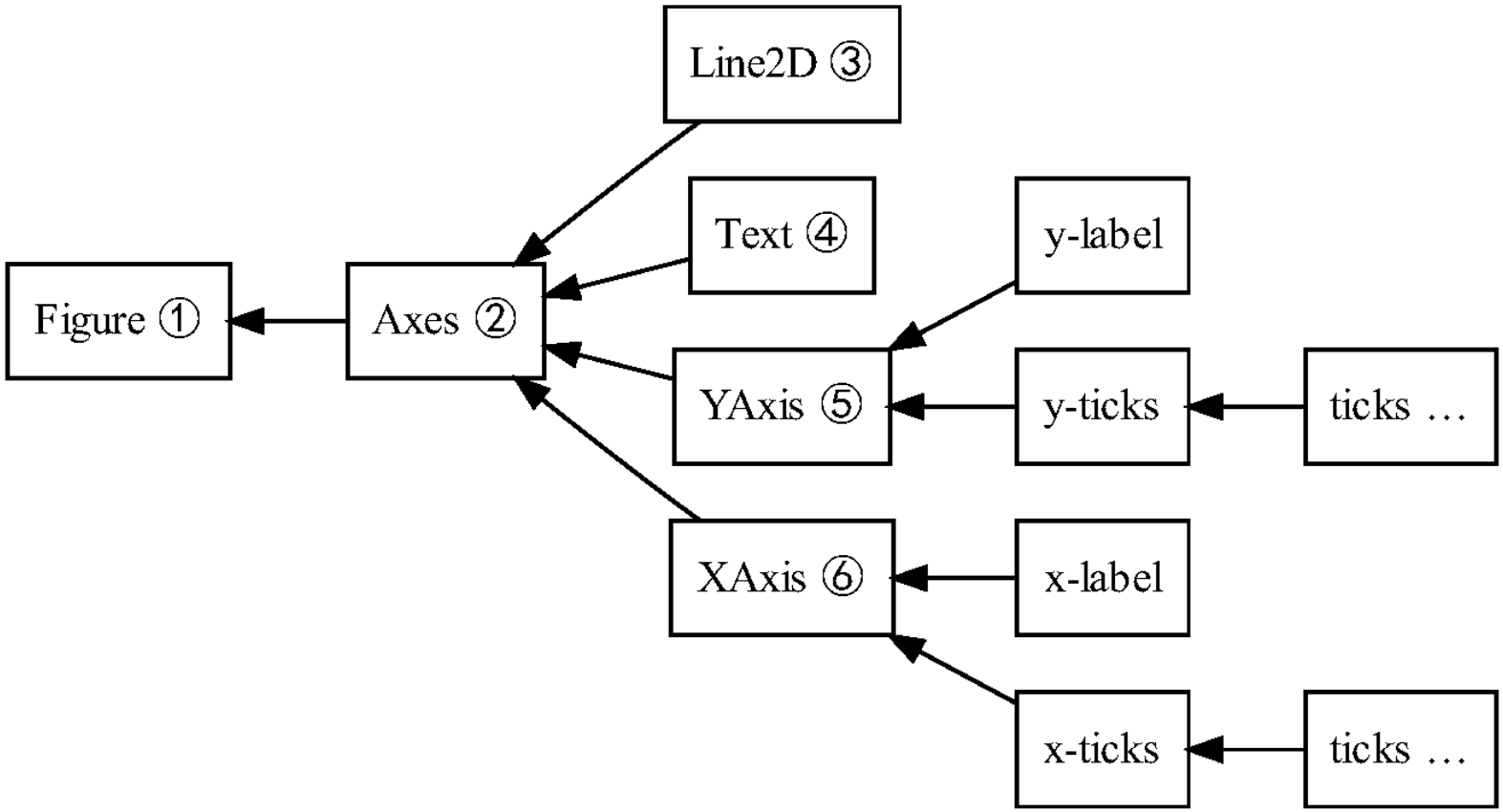
Matplotlib


Artist
Axis
Axis
Axes
Axes
Axes
Axes
Figure
Figure
Image
Text
…
Figure
Canvas

이름	특징
Canvas	• Graphic을 그려주는 객체 • Don't worry too much about the canvas, it is crucial as it is the object that actually does the drawing to get you your plot, but as the user it is more-or-less invisible to you • Figure 뿐만 아니라 Image, Text도 모두 Canvas에서 화면에 뿌려줌
Figure	• Axes을 자식으로 갖는 객체 • Axes를 0개부터 여러 개까지 가질 수 있음(subplot) • Rendering 되어야 함
Axes	• 일반적으로 Plot이라고 생각되는 객체 • The region of the image with the data space • Figure는 여러 개의 Axes를 가질 수 있으나, 하나의 Axes는 하나의 Figure 위에만 있을 수 있음 • Axes는 2~3개의 Axis를 가짐(Axes와 Axis는 다른 것이므로 유의)
Axis	• Graph limit(x_lim, y_lim)을 정하고, • Tick/ticklabel(ticks에 붙는 label string)을 정하는 역할을 함 • Locator 객체가 tick의 위치를 정하고, • Formatter가 tick label string을 formatted함
Artist	• Figure 위에 보이는 것들은 모두 Artist 객체 • Figure, Axes, Axis 객체 역시 Artist 객체 • Text, Line2D, collection, patch 등도 모두 artist 객체


Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Legend
Grid
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
X axis label
Made with http://matplotlib.org


A simple plot
voltage (mV)
time (s)
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6
7
①
②
③
④
⑤
⑥


Line2D ③
Text ④
y-label
Figure ①
Axes ②
YAxis ⑤
y-ticks
ticks …
XAxis ⑥
x-label
x-ticks
ticks …

■ 그래프를 구성하는 기본 객체(틀)

■ Figure를 별도로 선언하지 않을 경우, axes 등 하위 객체 생성시 자동적으로 생성

```
plt.figure()
plt.show()
```

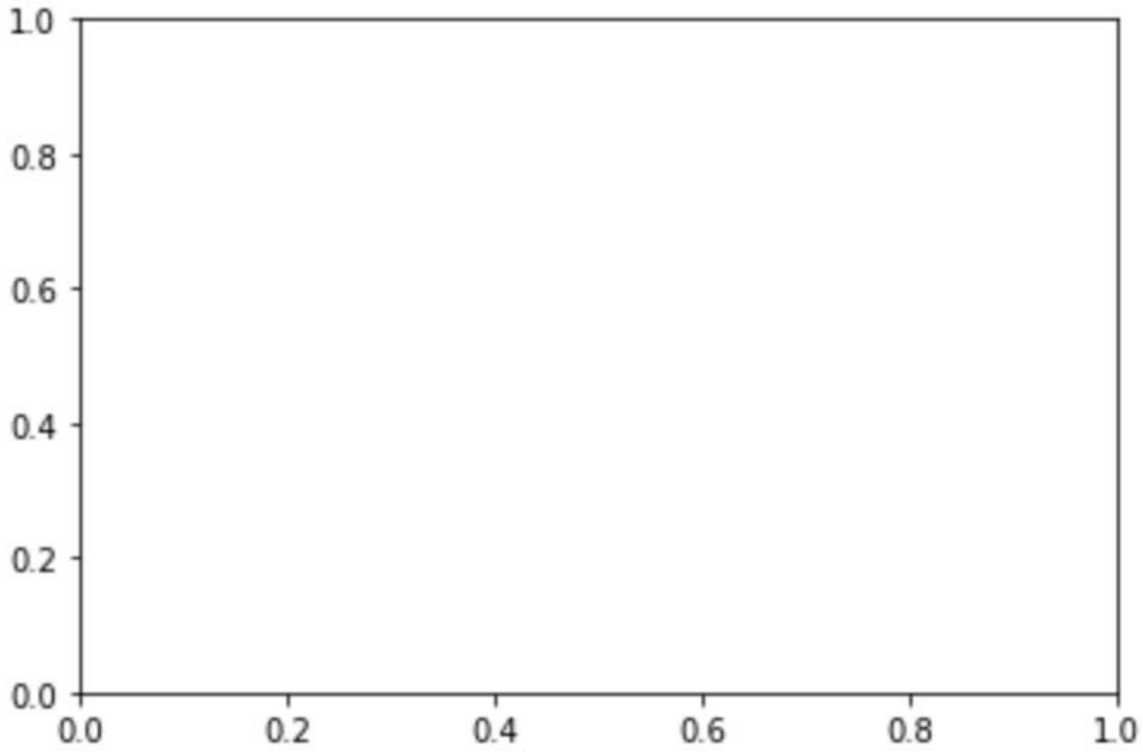
```
<Figure size 432x288 with 0 Axes>
```

```
plt.figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameo
n=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, **kwargs)
```

argument	description
num	Figure의 id값. 지정하지 않을 경우 순차적 증가
figsize	Figure의 크기. Width와 height의 tuple로 지정
dpi	Figure의 해상도
facecolor	기본 배경 색
edgecolor	기본 경계선 색
frameon	테투리를 생성할지 여부. 기본 True
clear	중복된 id의 figure가 있을 경우, 기존 figure 제거

■ Plot에 대한 정보를 관리하는 영역

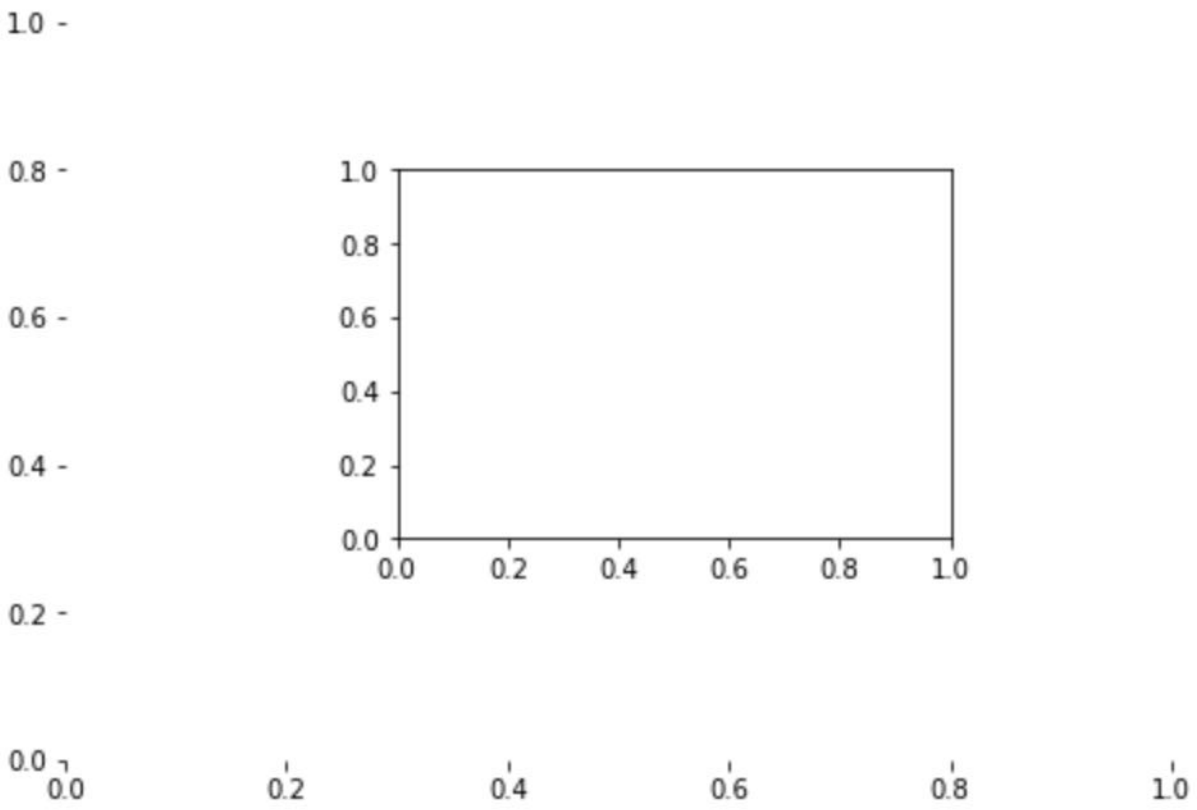
```
plt.figure()
plt.axes()
plt.show()
```



```
plt.axes(arg=None, **kwargs)
```

argument	description
arg	4-tuple : (left, right, width, height) 또는 Axes 객체
figsize	Figure의 크기. Width와 height의 tuple로 지정
facecolor	기본 배경 색
frameon	테투리를 생성할지 여부. 기본 True
polar	Plot을 polar plot으로 만들지 여부. 기본 False

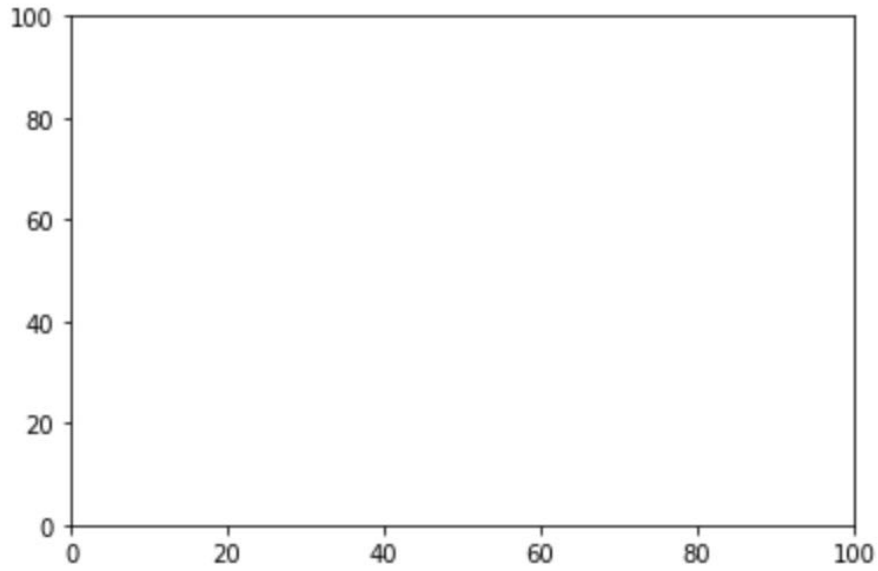
```
import matplotlib.pyplot as plt

plt.figure()
plt.axes([0, 0, 1, 1], frameon=False)
plt.axes([0.3, 0.3, 0.5, 0.5])
plt.show()
```



■ 각 데이터 축을 관리

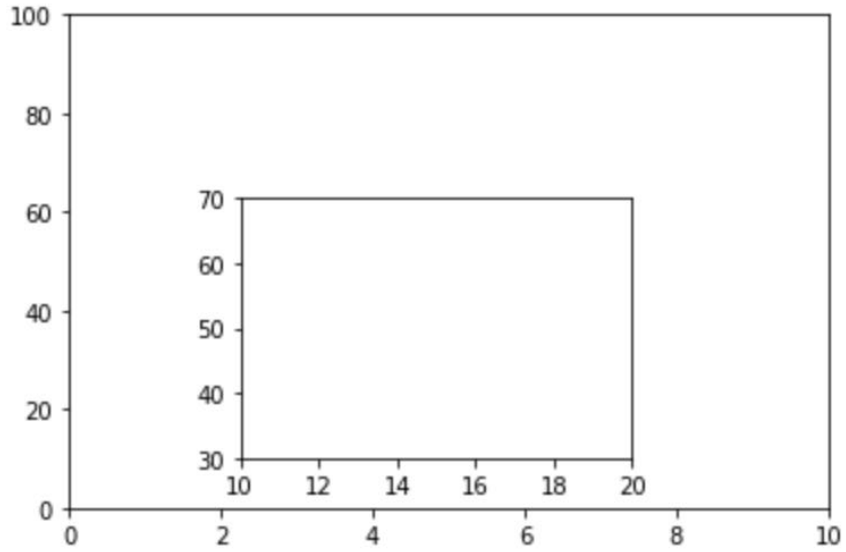
argument	description
v	4-tuple : (xmin, xmax, ymin, ymax)

```
import matplotlib.pyplot as plt

plt.figure()
plt.axes()
plt.axis([0,100,0,100])
plt.show()
```



■ Xlim : x축의 최대값 및 최솟값 관리
■ Ylim : y축의 최대값 및 최솟값 관리

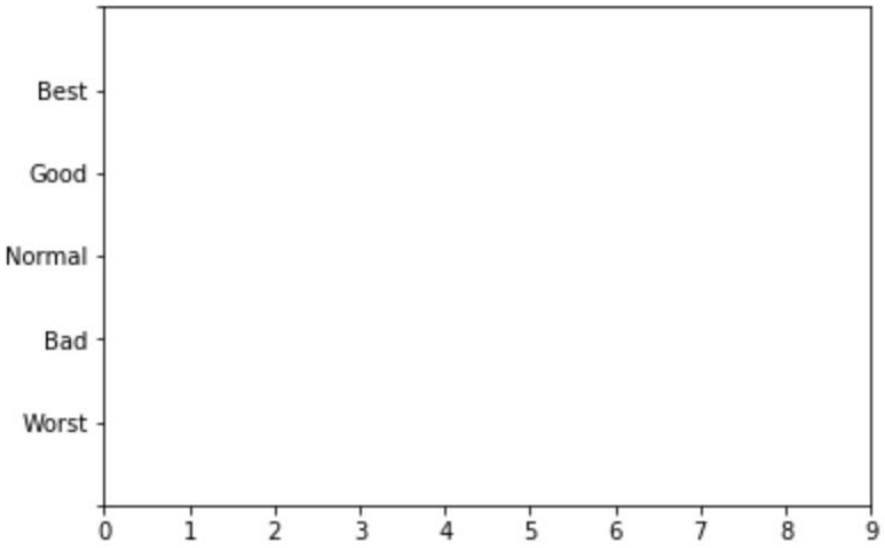
```
import matplotlib.pyplot as plt

plt.figure()
plt.axes()
plt.axis((0, 10, 0, 100))
plt.axes((0.3, 0.2, 0.4, 0.4))
xmin, xmax = plt.xlim(10, 20)
ymin, ymax = plt.ylim(30, 70)
plt.show()
print(xmin, xmax, ymin, ymax)
```




```
10 20 30 70
```

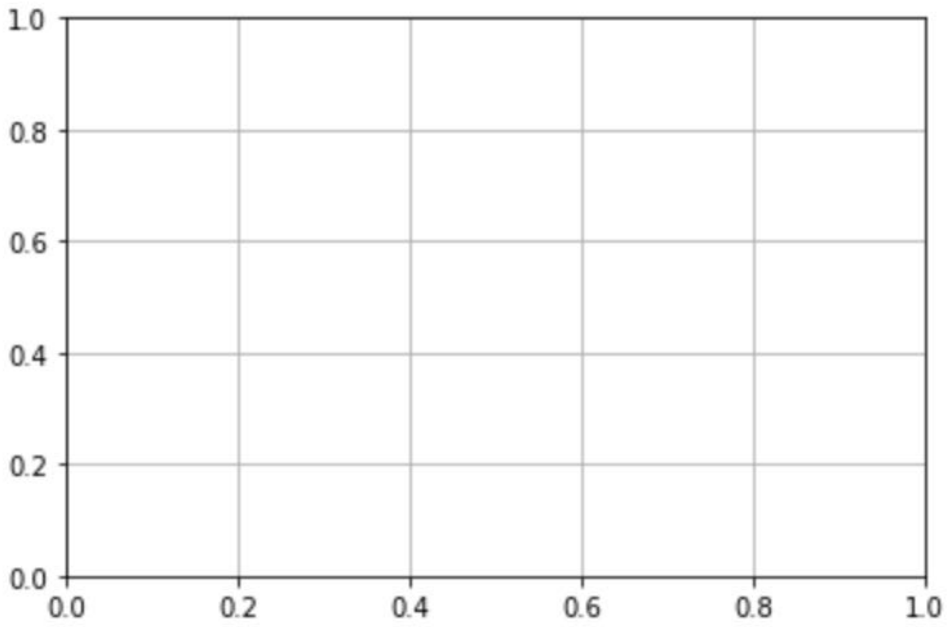
■ 각 축의 간격 및 값을 지정

```
import matplotlib.pyplot as plt
import numpy as np

plt.figure()
plt.axes()
plt.xticks(np.arange(0,10, step=1))
plt.yticks(np.arange(-1,6), ('', 'Worst', 'Bad', 'Normal', 'Good', 'Best', ''))
plt.show()
```



■ Plot에 격자를 출력

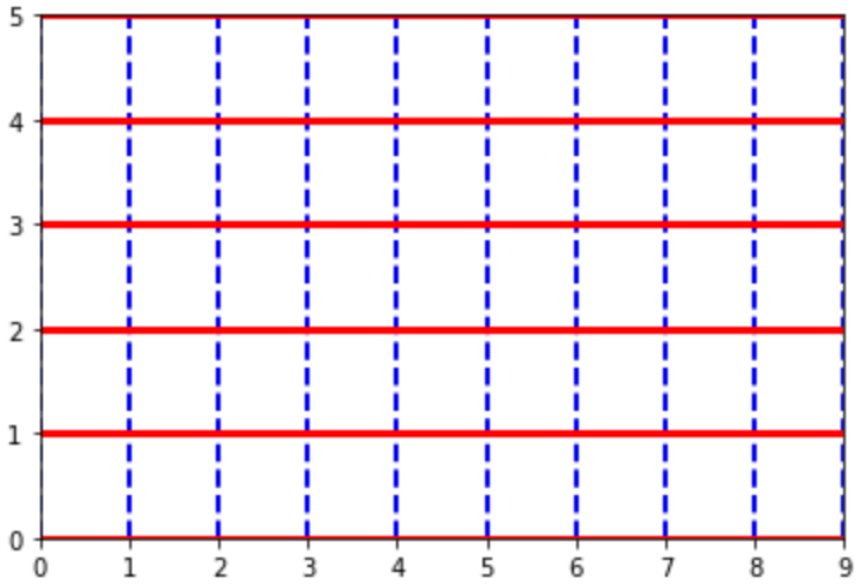
```
import matplotlib.pyplot as plt

plt.figure()
plt.axes()
plt.grid()
plt.show()
```



```
plt.grid(b=None, which='major', axis='both', **kwargs)
```

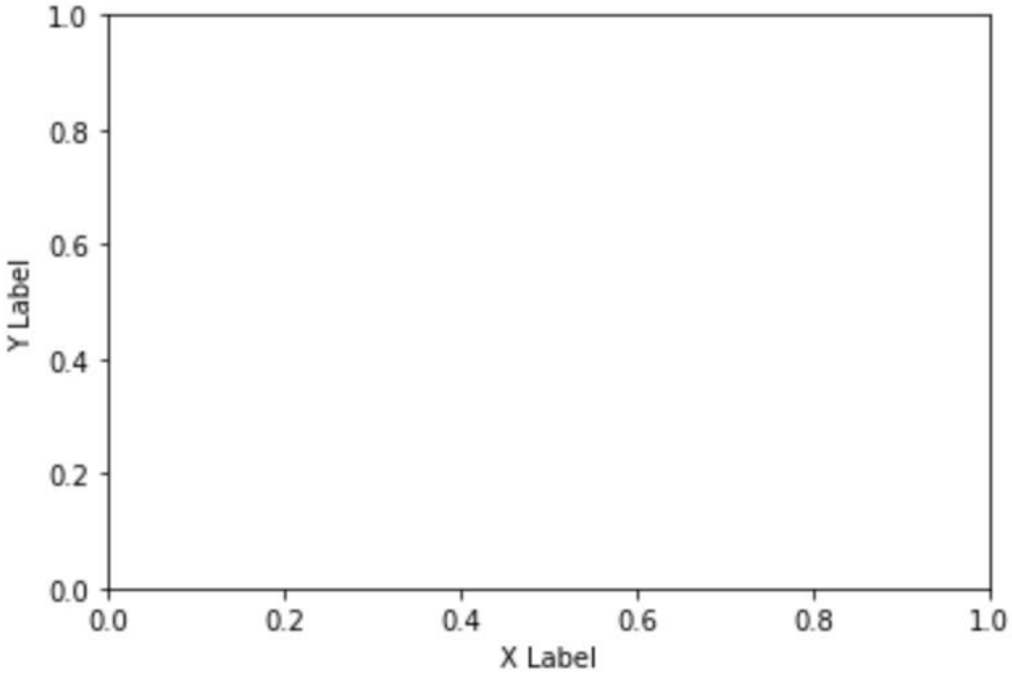
argument	description
b	Grid를 출력할지 말지 유무
which	어떤 레벨의 tick까지 grid를 표현할지 결정. 'major', 'minor', 'both'
axis	어떤 축의 grid를 표현할지 결정. 'x, 'y', 'both'
kwargs	기타 grid를 꾸미는 요소들

```
import matplotlib.pyplot as plt
import numpy as np

plt.figure()
plt.axes()
plt.xticks(np.arange(0,10,step=1))
plt.grid(True, 'major', 'x', color='b', linestyle='--', linewidth=2)
plt.ylim(0,5)
plt.grid(True, 'major', 'y', color='r', linewidth=3)
plt.show()
```



■ Axes의 각각의 축을 설명하는 문자열

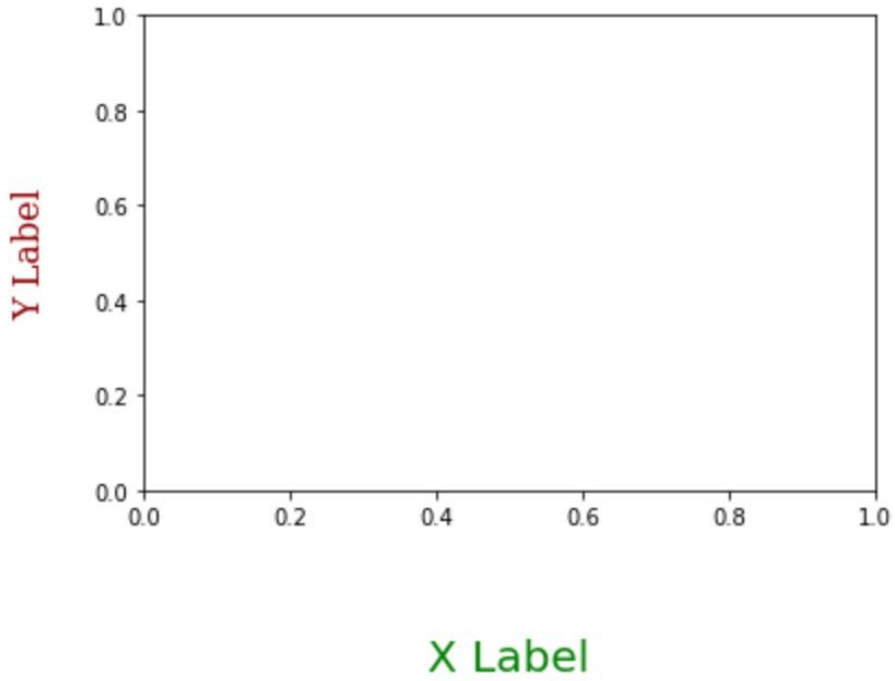
```
import matplotlib.pyplot as plt

plt.figure()
plt.axes()
plt.xlabel('X Label')
plt.ylabel('Y Label')
plt.show()
```



```
plt.xlabel(label, fontdict=None, labelpad=None, **kwargs)
```

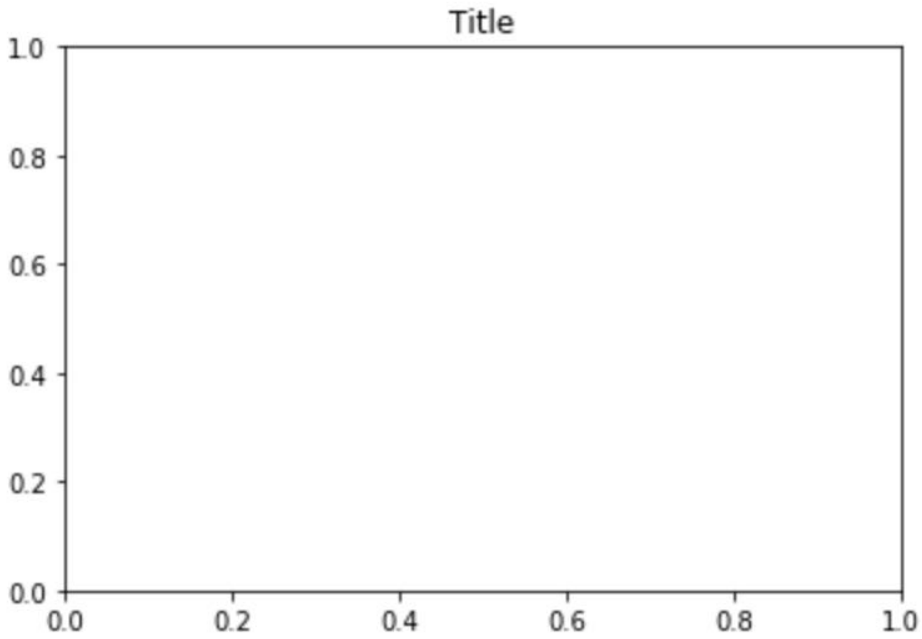
argument	description
label	Label의 문자열값
fontdict	Label에 사용할 폰트 속성 지정
labelpad	축과의 간격

```
import matplotlib.pyplot as plt

plt.figure()
plt.axes()
plt.xlabel('X Label', labelpad=50, color='g', fontsize=20)
font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'normal',
        'size': 16,
        }
plt.ylabel('Y Label', fontdict=font, labelpad=20)
plt.show()
```



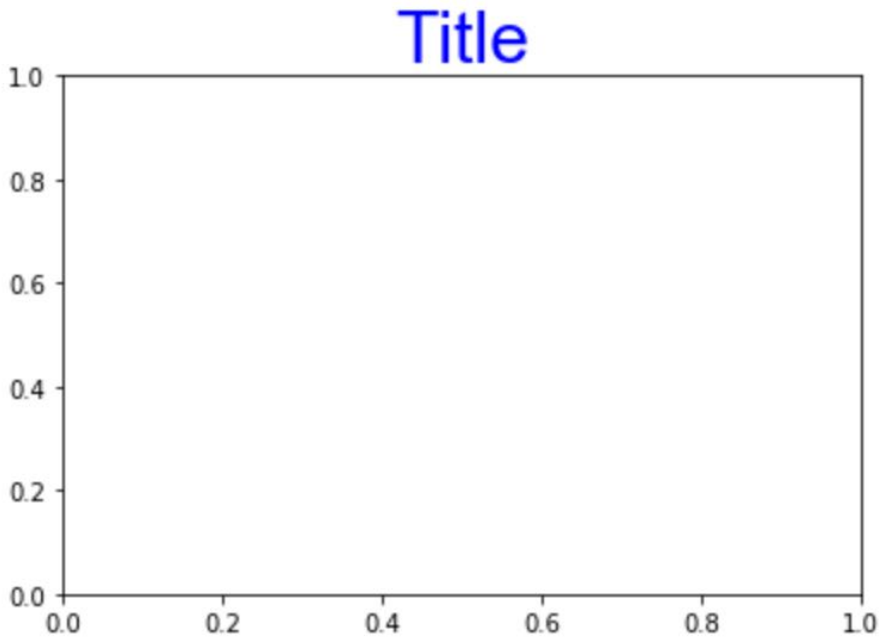
■ Axes의 plot을 설명하는 제목 문구

```
import matplotlib.pyplot as plt

plt.figure()
plt.axes()
plt.title('Title')
plt.show()
```



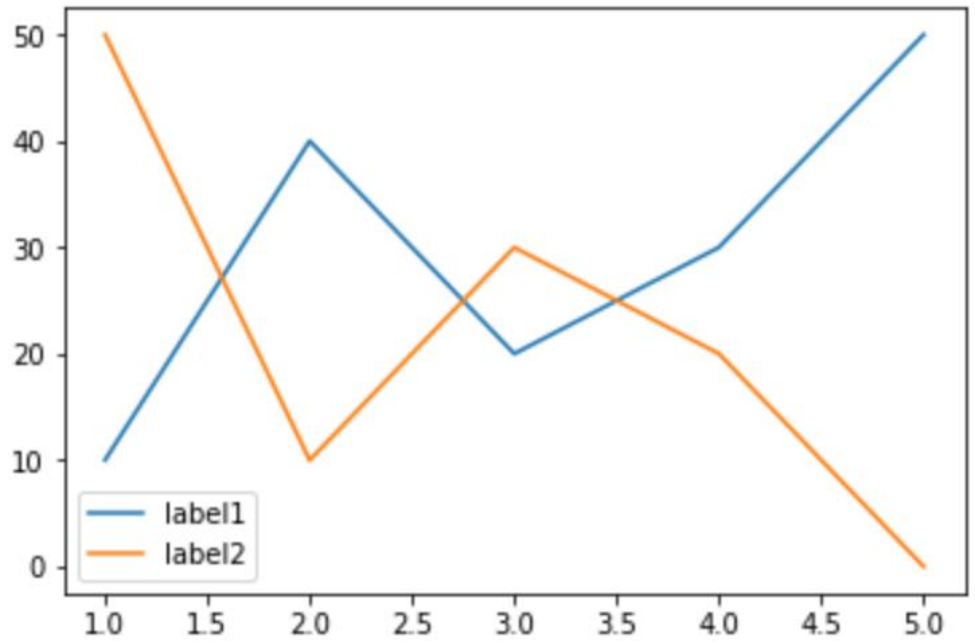
argument	description
s	Title의 문자열값
fontdict	Title에 사용할 폰트 속성 지정
loc	Title의 위치에 대한 정보

```
import matplotlib.pyplot as plt

plt.figure()
plt.axes()
font = {'family': 'Arial',
        'color': 'blue',
        'size': 30}
plt.title('Title', font)
plt.show()
```



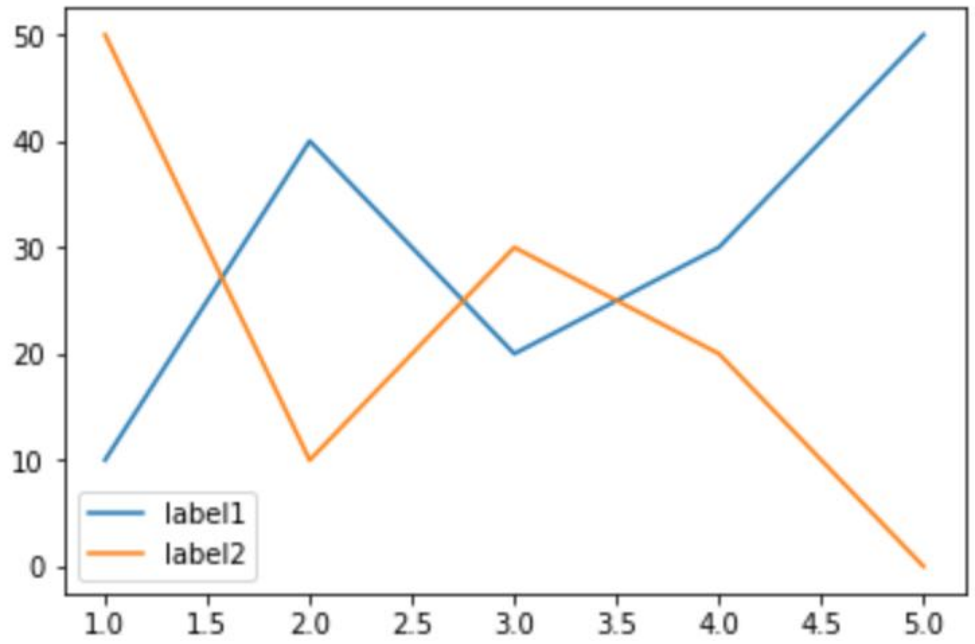
■ Plot 데이터의 범례

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot([1,2,3,4,5],[10,40,20,30,50], label='label1')
plt.plot([1,2,3,4,5],[50,10,30,20,0], label='label2')
plt.legend()
plt.show()
```



■ Plot 데이터의 범례

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot([1,2,3,4,5],[10,40,20,30,50], label='label1')
plt.plot([1,2,3,4,5],[50,10,30,20,0], label='label2')
plt.legend()
plt.show()
```



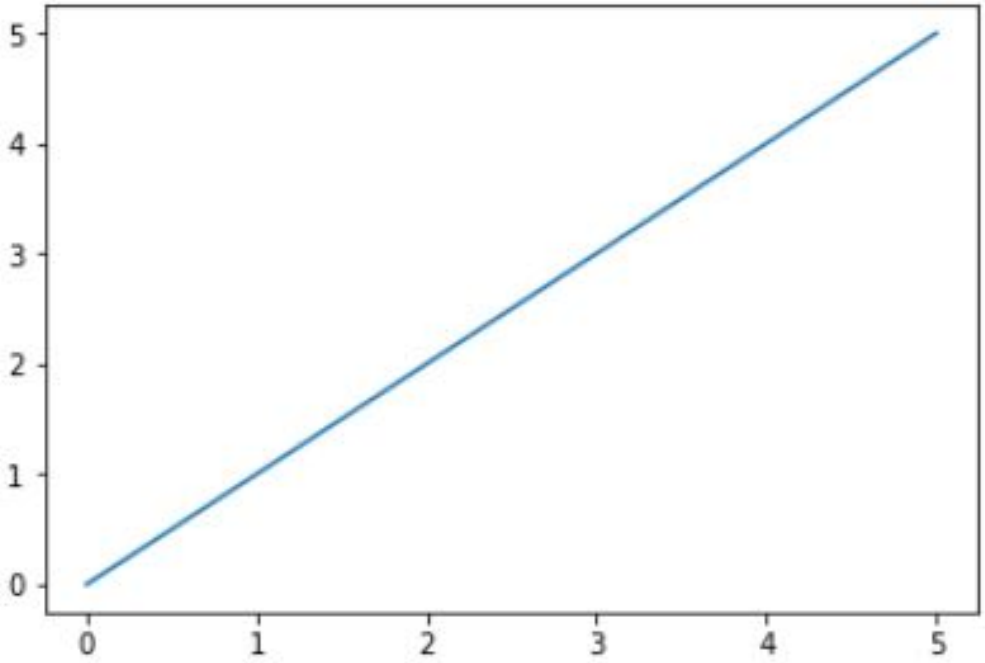
Basic Graph

■ Stephen Few의 8가지 정량적 표현 유형

종류	설명	예시
시계열 표현	일정 기간 동안의 단일 변수를 표현	Line Chart
순위 표현	범주 세분이 같은 데이터의 순위를, 오름차순 또는 내림차순으로 표현	Bar Chart, Bar Graph
부분과 전체	전체에 대한 범주형 세분 비율 표현	Pie Chart
편차	표준으로부터 벗어난 차이의 표현	Distribution Chart
빈도분포	주어진 변수의 발생 빈도를 표현	Histogram, Box Plot, Violin Plot
상관관계	두 변수의 연관성 값을 비교하여 표현	Scatter Plot
Nominal 비교	특정 순서없이 카테고리 하위 구분을 비교	Bar chart
지리정보	지리적인 데이터의 시각적 표현	Cartogram

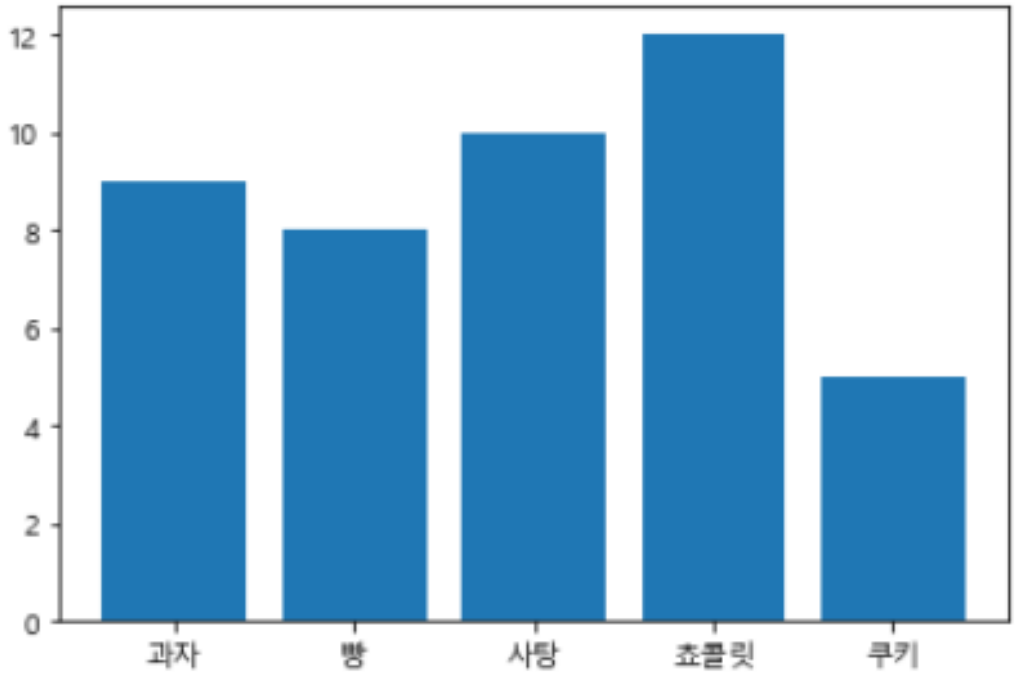
- 가장 기본적인 그래프
- 2개의 변수를 비교할 때 효과적
- 시계열 데이터의 변화를 확인
- 정규분포 확인 가능

```
x = [0,1,2,3,4,5]
y = [0,1,2,3,4,5]
plt.plot(x,y)
```



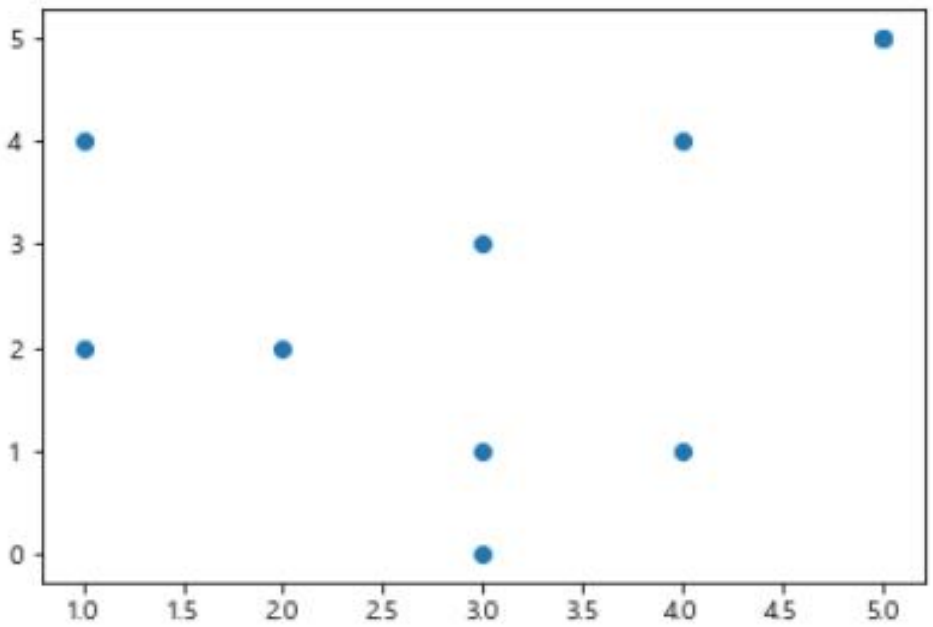
■ 여러 종류의 분류형 데이터를 비교
■ X축은 범주, Y축은 값을 주로 사용

```
x = ['사탕','쵸콜릿','빵','과자','쿠키']
y = [10,12,8,9,5]
plt.bar(x,y)
```



- 2개의 변수에서 데이터의 분포를 확인
- 주로 상관관계를 분석함

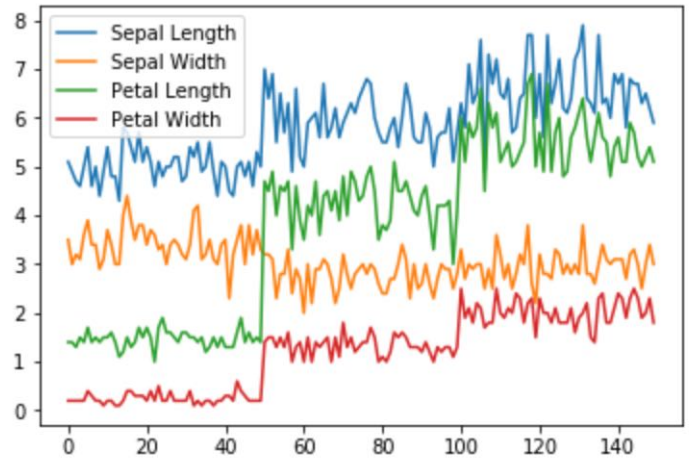
```
x = [3,2,2,3,5,5,1,3,4,5]
y = [3,1,2,3,4,5,2,2,4,5]
plt.scatter(x,y)
```



Multi Line plot

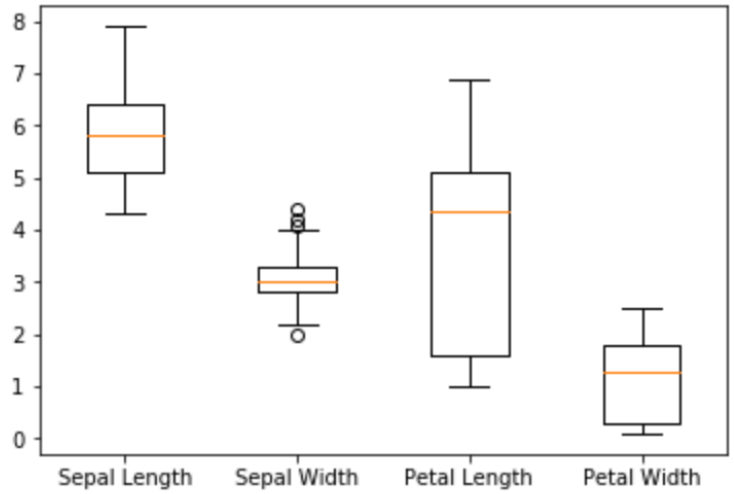
- Line 그래프를 여러 개 그림
- 각 변수들의 분포를 확인가능
- 시계열 또는 범주형 데이터 확인

```
import pandas as pd
import matplotlib.pyplot as plt

iris = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header=None)
iris.columns=["Sepal Length","Sepal Width","Petal Length","Petal Width","Class"]

plt.figure()
plt.plot(iris['Sepal Length'].values, label='Sepal Length')
plt.plot(iris['Sepal Width'].values, label='Sepal Width')
plt.plot(iris['Petal Length'].values, label='Petal Length')
plt.plot(iris['Petal Width'].values, label='Petal Width')
plt.legend()
plt.show()
```



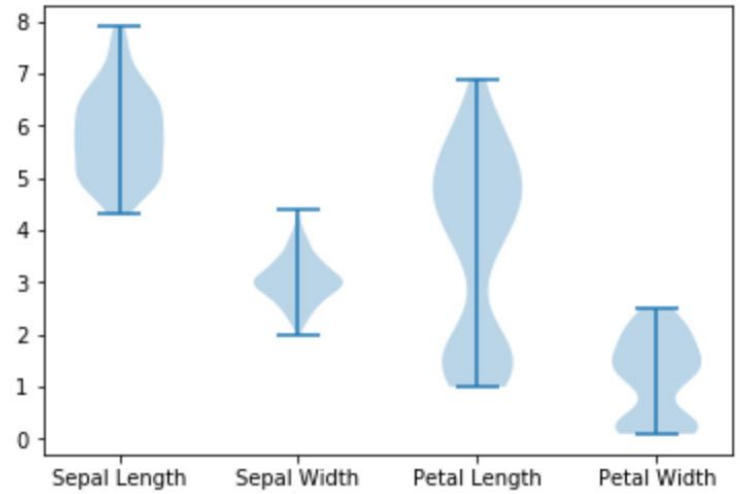
- 상자와 수염으로 이루어짐
- 최솟값, 최댓값, 1 2 3사분위 표현
- 비슷한 크기의 값들을 함께 나타냄

```
import pandas as pd
import matplotlib.pyplot as plt

iris = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header=None)
iris.columns=["Sepal Length","Sepal Width","Petal Length","Petal Width","Class"]
collection = iris[['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']].values

plt.figure()
plt.boxplot(collection)
plt.xticks([1,2,3,4], ('Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'))
plt.show()
```



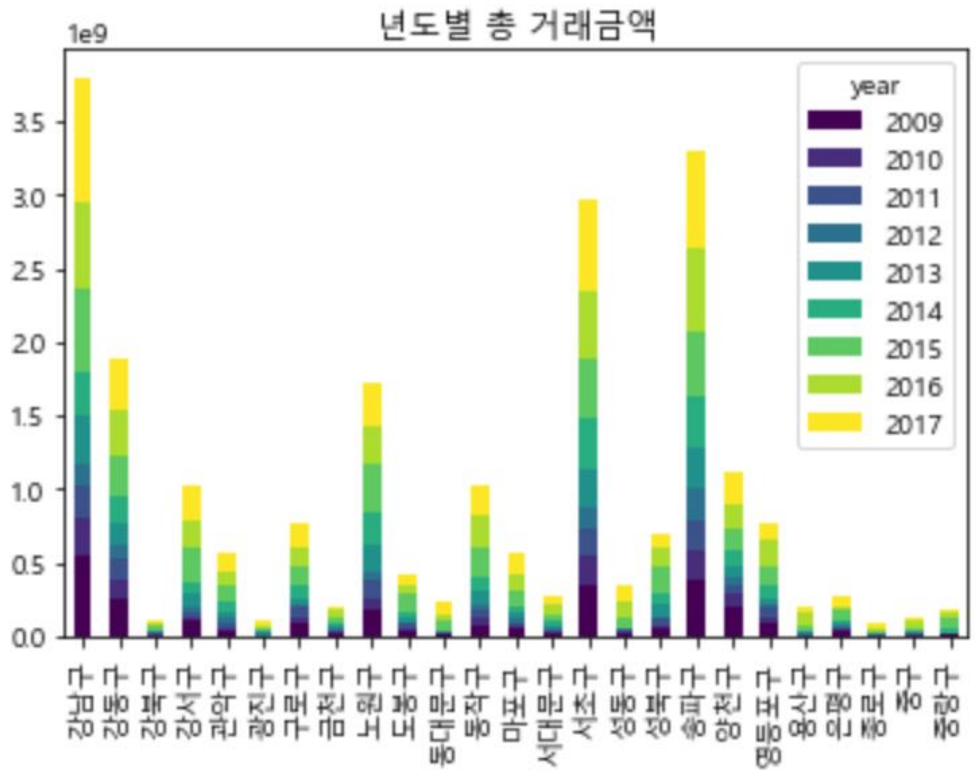
■ Box plot과 유사함
■ 분포를 두께로 표현

```
import pandas as pd
import matplotlib.pyplot as plt

iris = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header=None)
iris.columns=["Sepal Length","Sepal Width","Petal Length","Petal Width","Class"]
collection = iris[['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']].values

plt.figure()
plt.violinplot(collection)
plt.xticks([1,2,3,4], ('Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'))
plt.show()
```

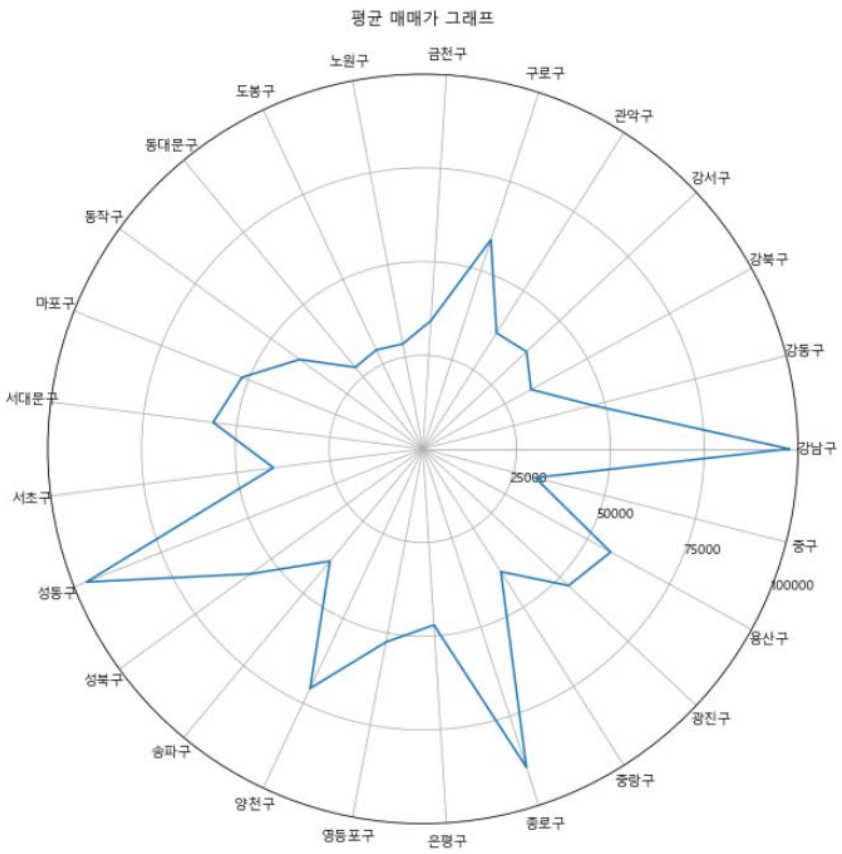


■ 원하는 3개의 변수를 한번에 나타냄
■ 시계열 데이터 표현에 효과적

```
apt = pd.read_csv("apt.csv")
gu = apt.groupby(['gu','year']).price.sum()
gu.unstack().plot(kind='bar',stacked=True,
                  colormap='viridis',grid=False)
plt.title('년도별 총 거래금액')
plt.show()
```



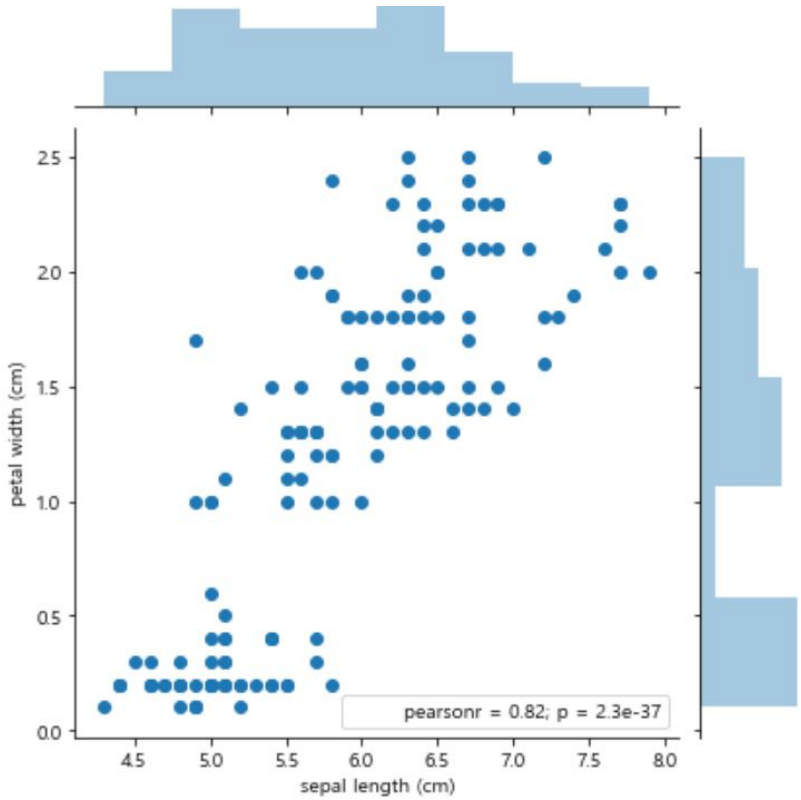
- X축을 각도로 표현
- 비슷한 값을 비교하기 편리함
- 세세한 파라미터 설정 필요




```
import numpy as np
m = pd.DataFrame(apt.groupby('gu').price.mean())
mvalue = [0]*26
for i in range(0,25):
    mvalue[i] = m.price[i]
mvalue[25] = mvalue[0]
xlabel = apt.gu.unique()
r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r * 4
plt.figure(figsize=(10,10))
ax = plt.subplot(111, projection='polar')
ax.plot(theta[0:26], mvalue,)
ax.set_rmax(2)
ax.set_rticks([25000, 50000, 75000, 100000])
ax.set_rlabel_position(-22.5)
xticks = np.arange(0,26,1)
xticks = xticks * np.pi * 2 / 25
ax.set_xticks(xticks)
ax.set_xticklabels(xlabel)
ax.grid(True)
ax.set_title("평균 매매가 그래프", va='bottom')
plt.show()
```

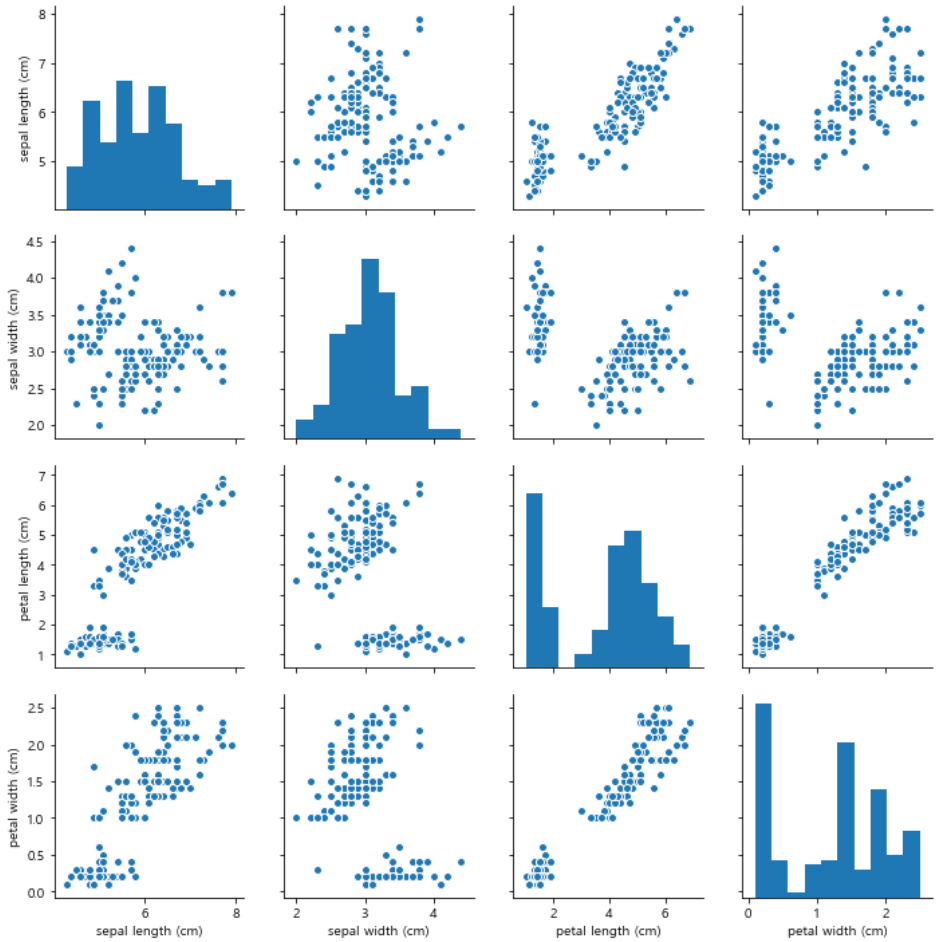
- Scatter와 box plot을 합친 그래프
- 2개 변수의 관계를 좀더 쉽게 확인

```
import seaborn as sns
sns.jointplot(iris['sepal length (cm)'],
              iris['petal width (cm)'])
```



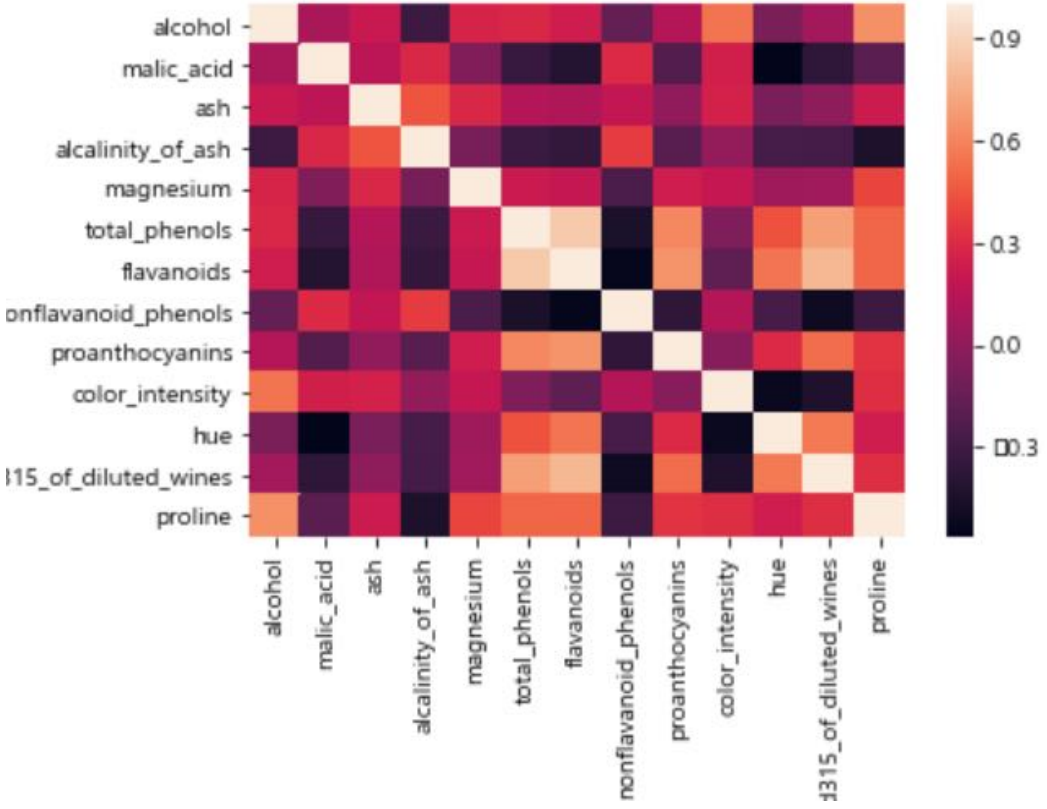
- 모든 변수간 scatter plot을 나타냄
- 대각선은 변수 하나의 box plot
- 적당한 변수들 상관관계 분석가능

```
sns.pairplot(iris)
```



- 변수들의 상관관계를 색으로 표현
- 밝을수록 양의 상관관계를 가짐

```
corr = wine.corr()
sns.heatmap(corr, xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```



Data Wrangling

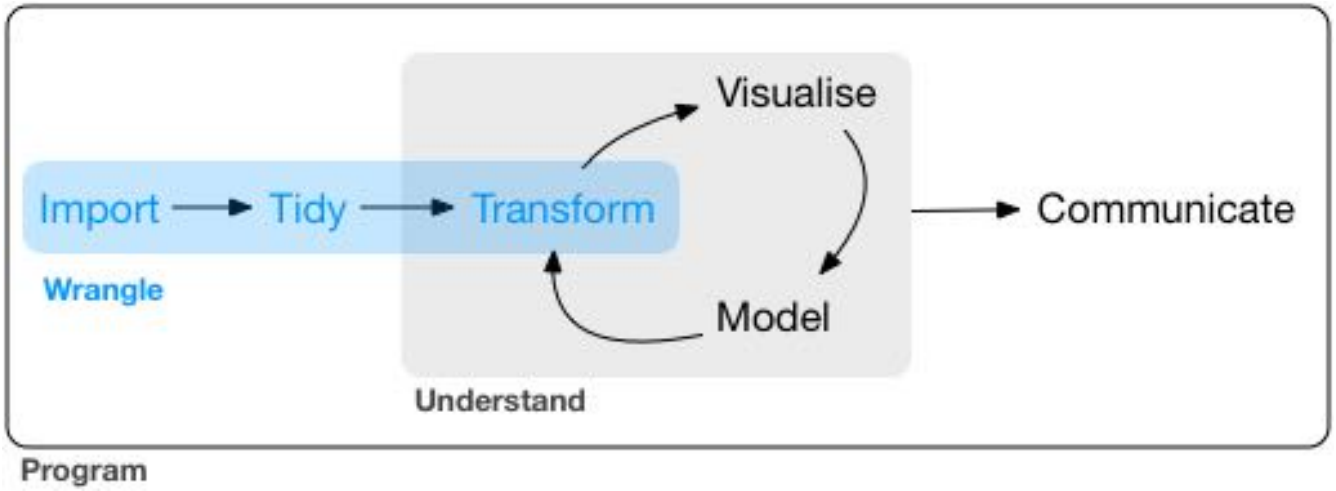
■ Real World Data는 정확하지 않음

○ Noisy data의 존재

○ 불연속적인 데이터

■ Random Forest같은 기계 학습 모델은 not null을 요구

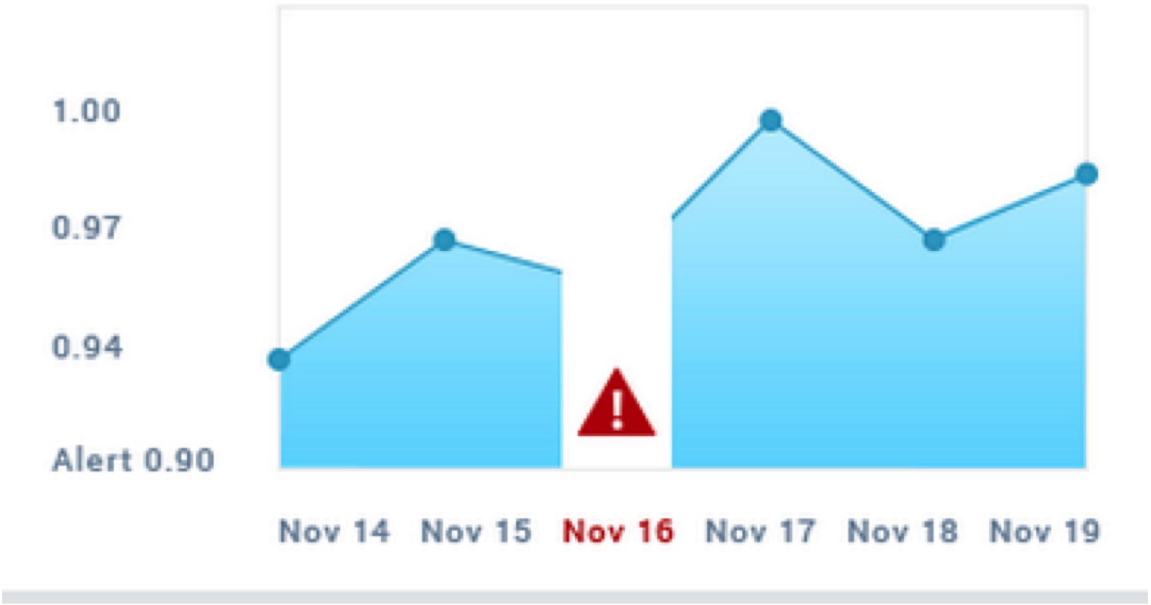
○ 분석에 맞지 않는 outlier 제거



■ NaN은 모델 학습의 걸림돌

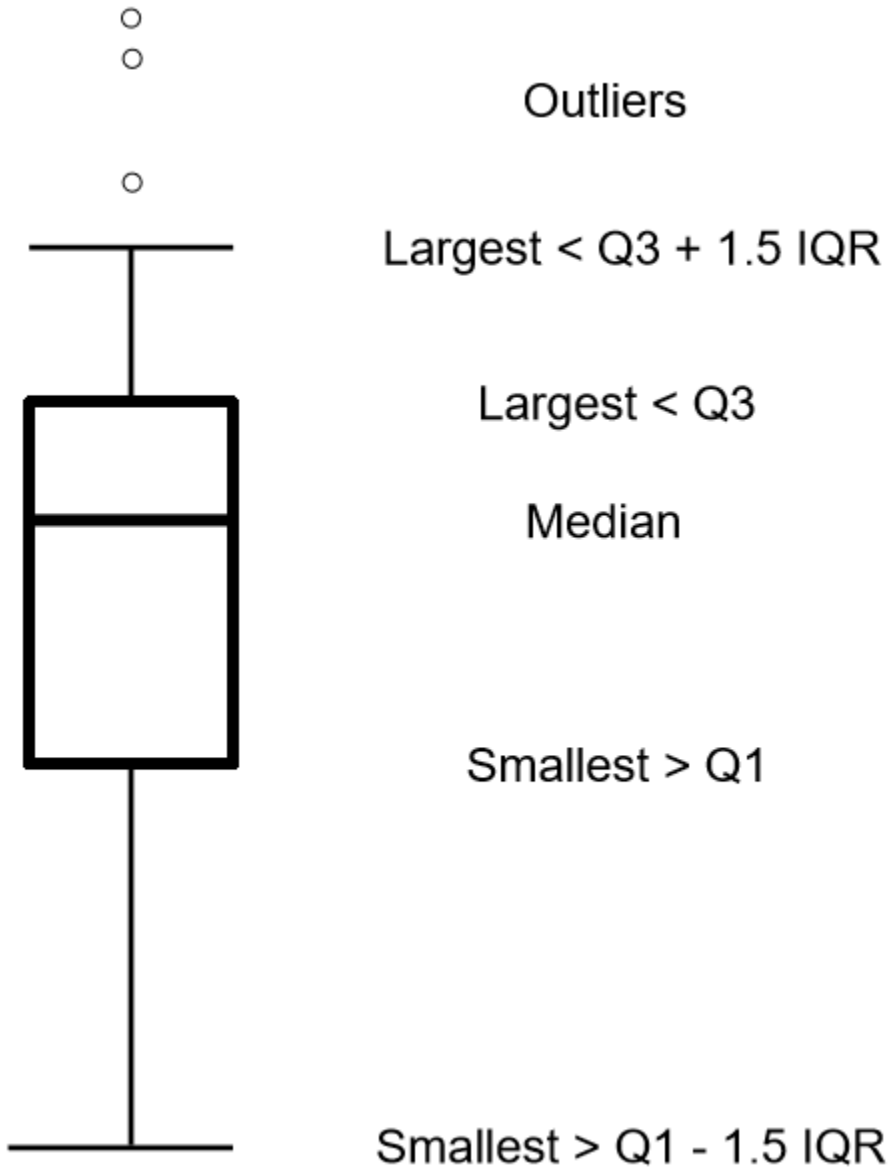
■ 데이터 항목 삭제
■ Smoothing으로 주변과 비슷한 값을 부여함

○ 연속적인 데이터일 경우 주로 Smoothing을 사용

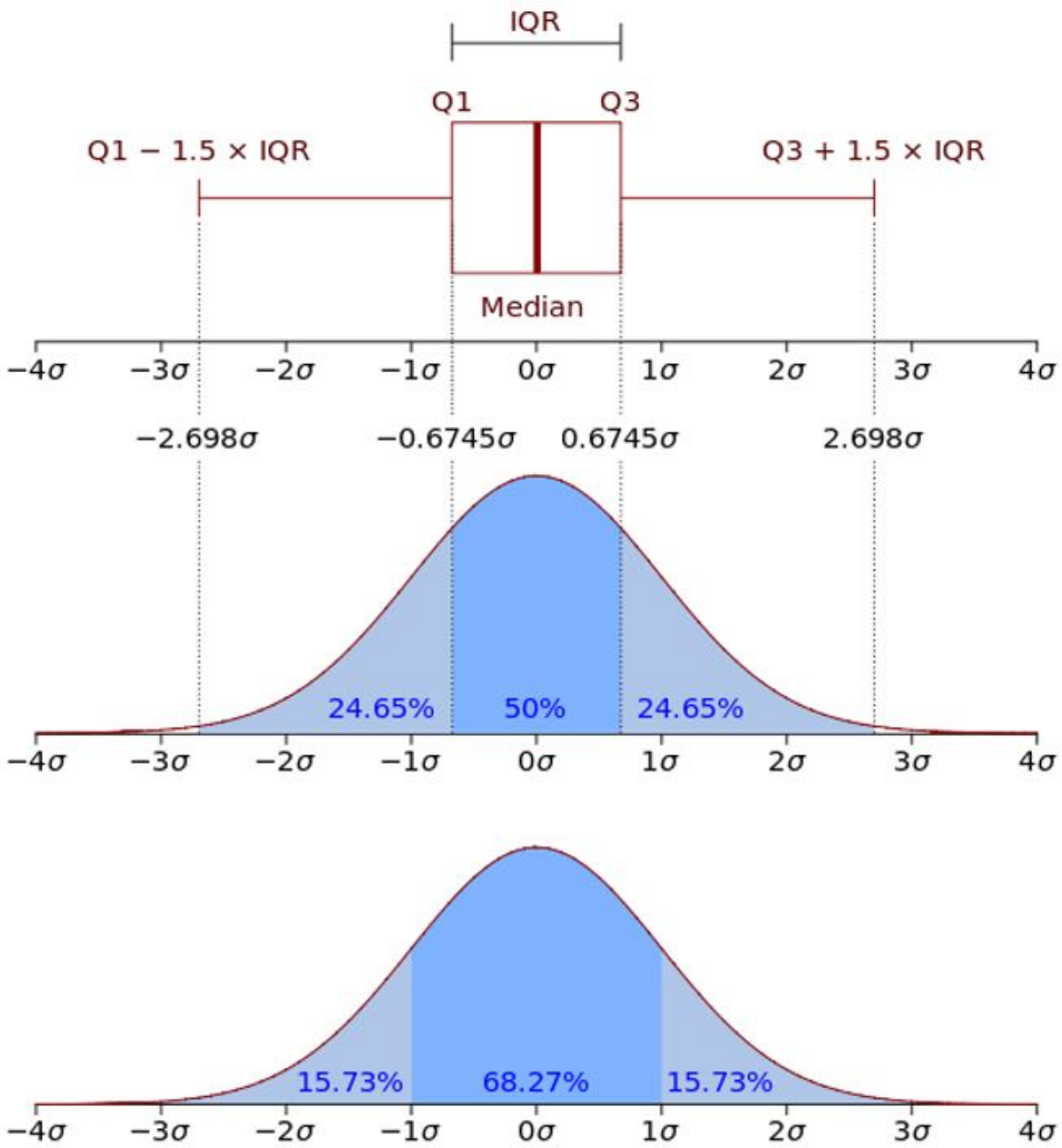


- 데이터 학습에 방해가 되는 instance

- Box plot으로 확인
- Scatter plot등으로 눈으로 확인



Outlier의 기준


IQR
Q1
Q3
Q1 − 1.5 × IQR
Q3 + 1.5 × IQR
Median
−4σ
−3σ
−2σ
−1σ
0σ
1σ
2σ
3σ
4σ
−2.698σ
−0.6745σ
0.6745σ
2.698σ
24.65%
50%
24.65%
15.73%
68.27%
15.73%

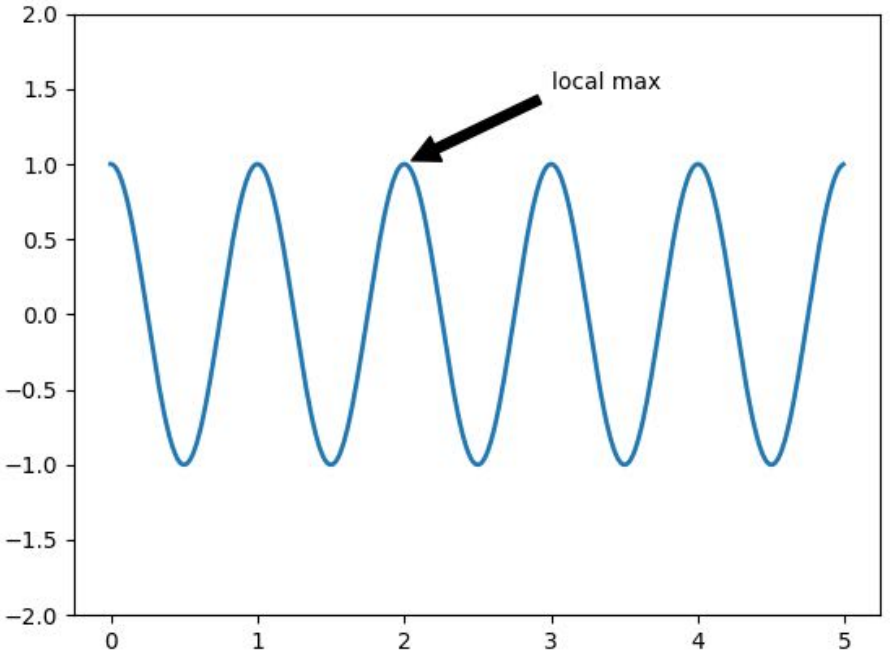
- 분산이 없는 데이터는 의미가 없음
- 데이터의 특징으로 정보량이 제일 많음
- PCA과 같은 축을 줄이는 방식도 분산을 보존함

Matplotlib
Advanced Feature

```
import numpy as np
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = ax.plot(t, s, lw=2)
ax.annotate('local max', xy=(2, 1), xytext=(3, 1.5), arrowprops=dict(facecolor='black', s
hrink=0.05), )
ax.set_ylim(-2, 2)

plt.show()
```

Default

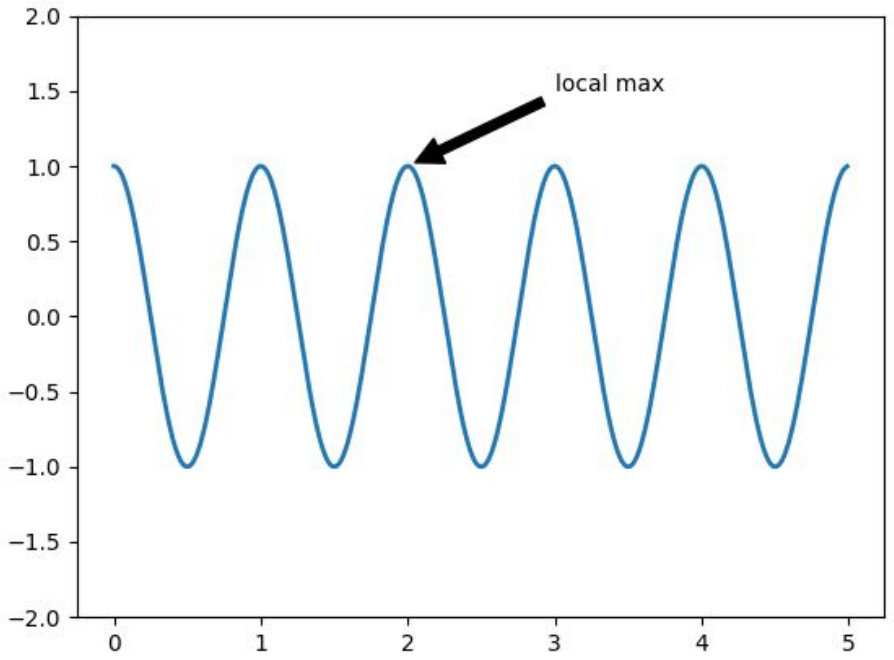
```
ax.annotate(xycoords='data', textcoords='data')
```



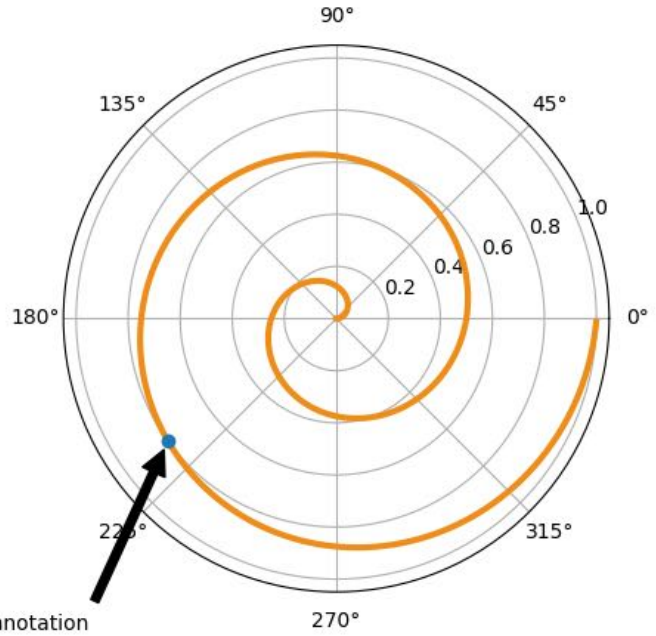
argument	coordinate system
'figure points'	Figure의 왼쪽 아래부터 Points로 좌표 지정
'figure pixels'	Figure의 왼쪽 아래부터 Pixel로 좌표 지정
'figure fraction'	Figure의 왼쪽 아래 (0,0) 오른쪽 위 (1,1)
'axes points'	Axes의 왼쪽 아래부터 Points로 좌표 지정
'axes pixels'	Axes의 왼쪽 아래부터 Pixel로 좌표 지정
'axes fraction'	Axes의 왼쪽 아래 (0,0) 오른쪽 위 (1,1)
'data' **(default)**	Data 값 기준 Coordinate system

arrowprops key	description
width	Arrow에 있는 point의 width
frac	Arrow의 head가 차지하는 비율
headwidth	Arrow의 head의 width
shrink	Annotate된 점이나 Text에서 Arrow의 Tip이 몇 % 떨어져 있는 지
**kwargs	matplotlib.patches.Polygon에 있는 모든 key(argument) e.g. facecolor

```
ax.annotate('local max', xy=(3, 1), xycoords='data',
            xytext=(0.8, 0.95), textcoords='axes fraction',
            arrowprops=dict(facecolor='black', shrink=0.05),
            horizontalalignment='right', verticalalignment='top',
            )
```



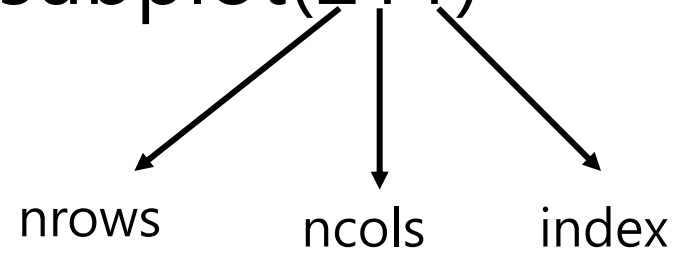
```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111, polar=True)
r = np.arange(0,1,0.001)
theta = 2 * 2*np.pi * r
line, = ax.plot(theta, r, color='#ee8d18', lw=3)
ind = 800
thisr, thistheta = r[ind], theta[ind]
ax.plot([thistheta], [thisr], 'o')
ax.annotate('a polar annotation',
                    xy=(thistheta, thisr), # theta, radius
                    xytext=(0.05, 0.05), # fraction, fraction
                    textcoords='figure fraction',
                    arrowprops=dict(facecolor='black', shrink=0.05),
                    horizontalalignment='left',
                    verticalalignment='bottom',
                    )
plt.show()
```

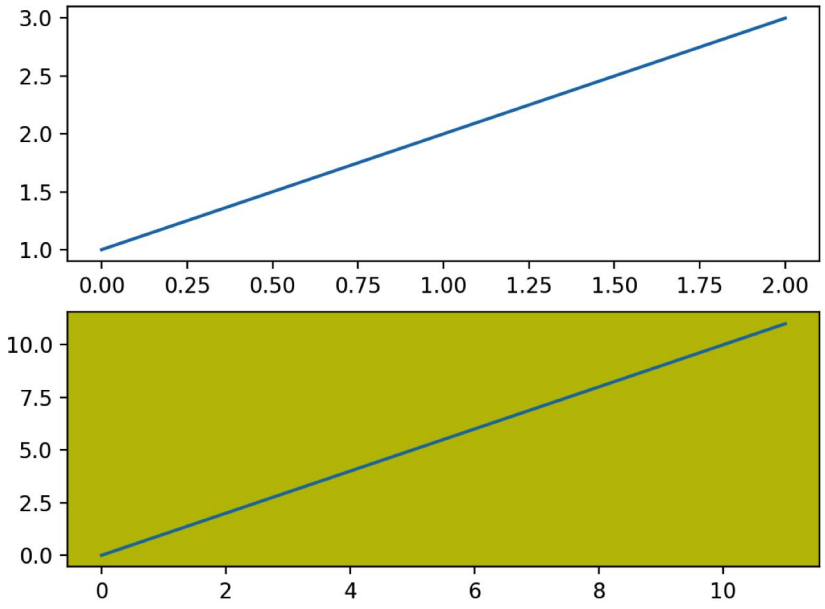


matplotlib.pyplot.subplot(*args, **kwargs)

subplot(nrows, ncols, index, **kwargs)

plt.subplot(211)

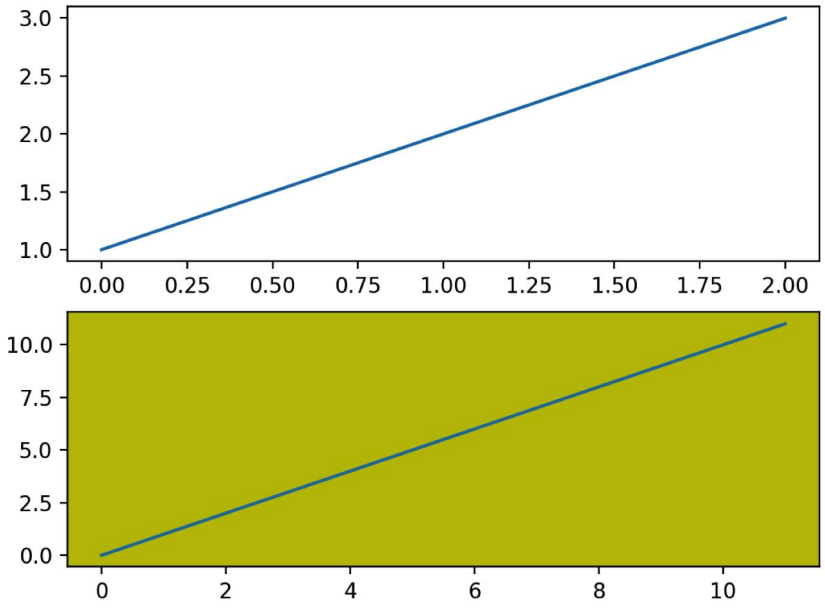




matplotlib.pyplot.subplot(*args, **kwargs)

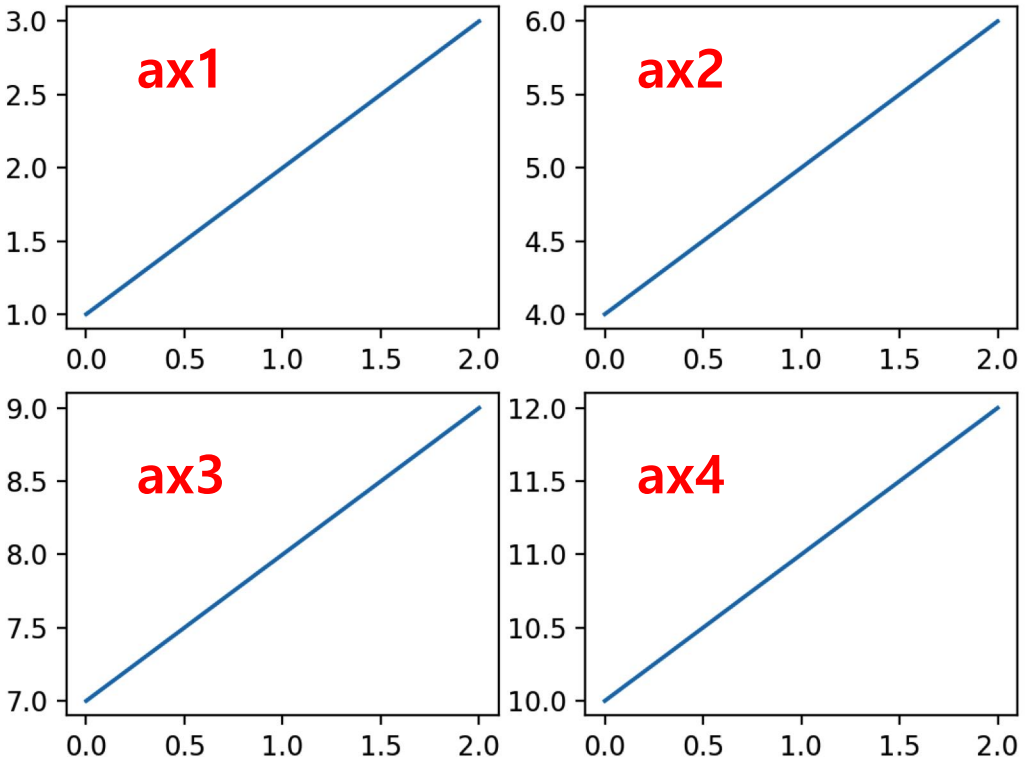
subplot(nrows, ncols, index, **kwargs)

```
import matplotlib.pyplot as plt
plt.subplot(211)
plt.plot([1,2,3])
plt.plot(range(12))
plt.subplot(212, facecolor='y')
```



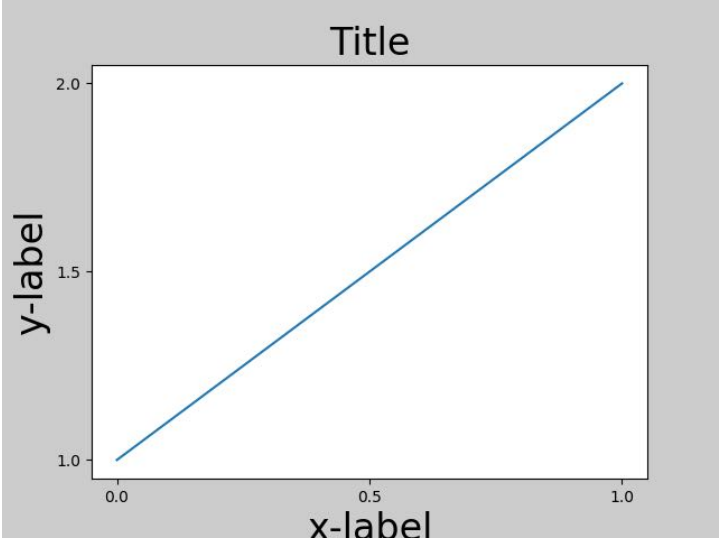
Axes

```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2)
ax1.plot([1,2,3])
ax2.plot([4,5,6])
ax3.plot([7,8,9])
ax4.plot([10,11,12])
```

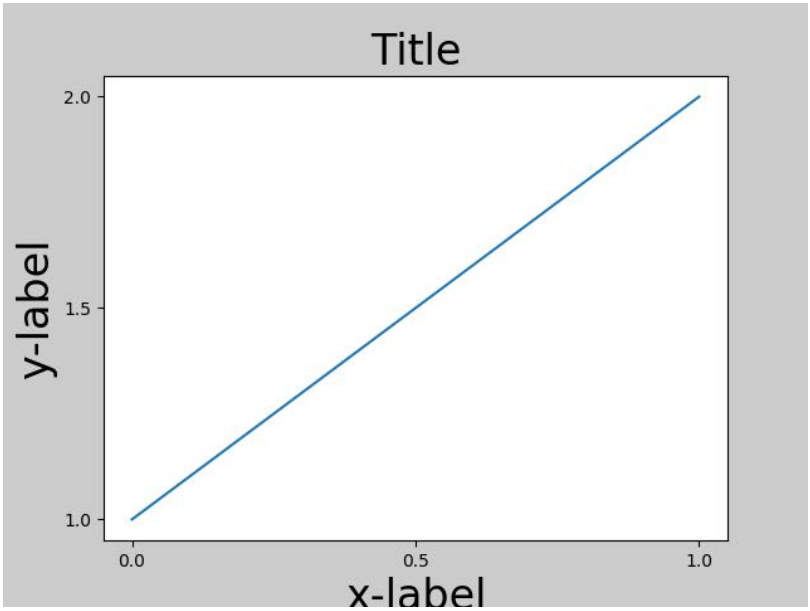


```
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams['savefig.facecolor'] = "0.8"

def example_plot(ax, fontsize=12):
    ax.plot([1, 2])

    ax.locator_params(nbins=3)
    ax.set_xlabel('x-label', fontsize=fontsize)
    ax.set_ylabel('y-label', fontsize=fontsize)
    ax.set_title('Title', fontsize=fontsize)
```

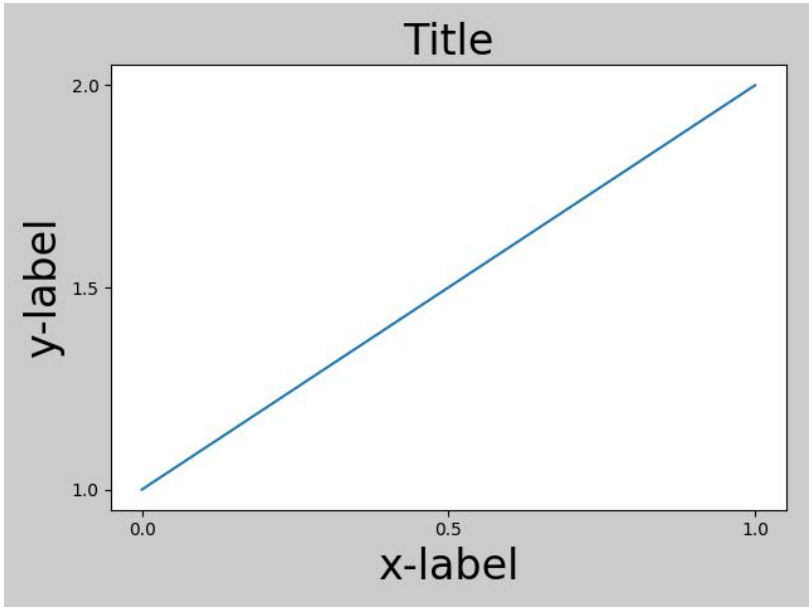
example_plot



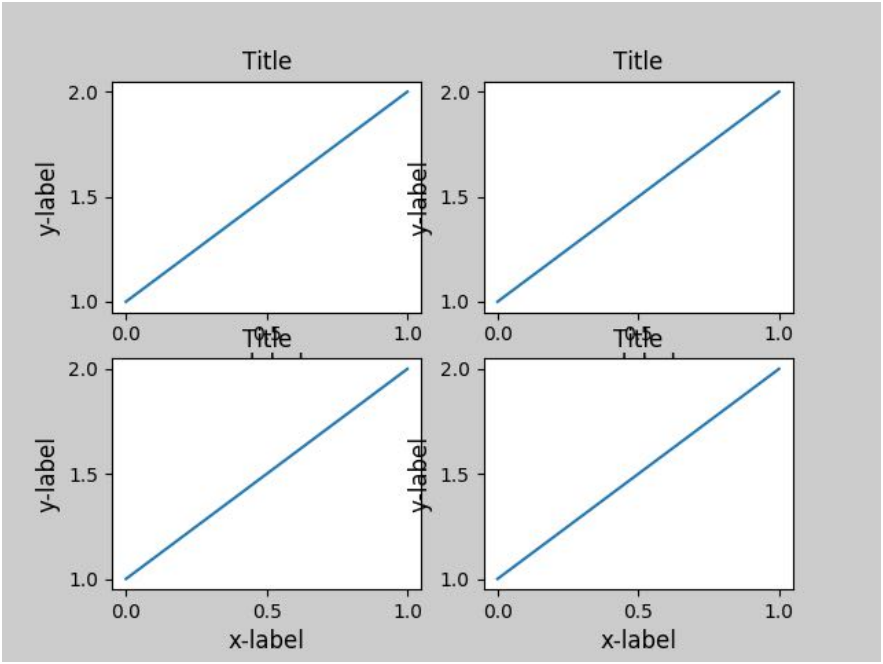
```
fig, ax = plt.subplots()
example_plot(ax, fontsize=24)
```



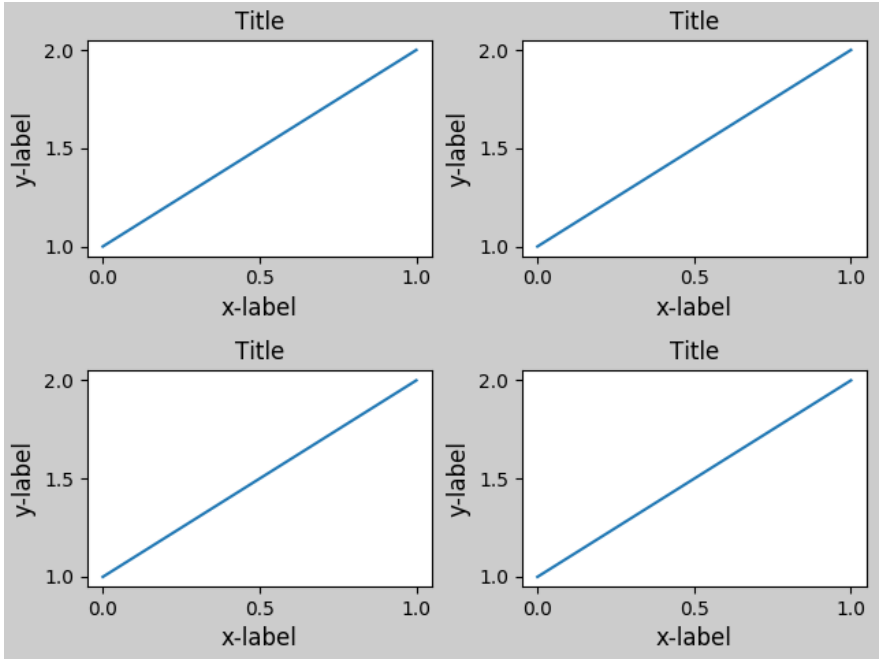

```
fig, ax = plt.subplots()
example_plot(ax, fontsize=24)
plt.tight_layout()
```



```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(
nrows=2, ncols=2)
example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)
```

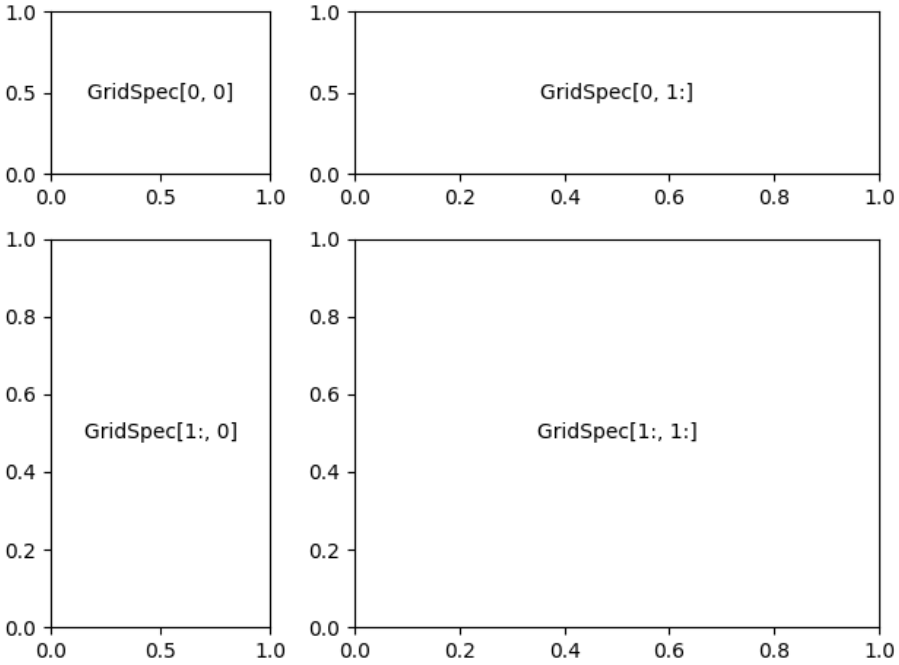



```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(
nrows=2, ncols=2)
example_plot(ax1)
example_plot(ax2)
example_plot(ax3)
example_plot(ax4)
plt.tight_layout(pad=0.4, w_pad=0.5,
h_pad=1.0)
```



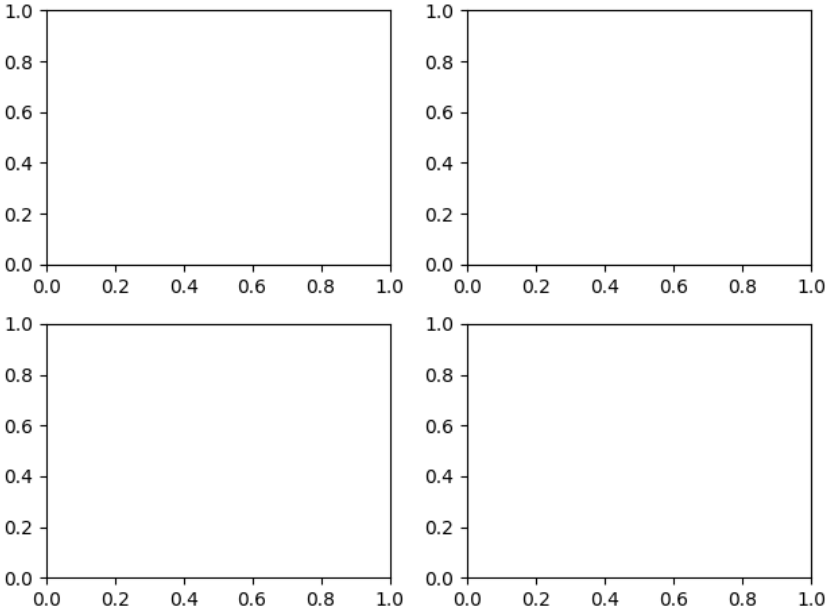
Create **grid-shaped** combinations of axes

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

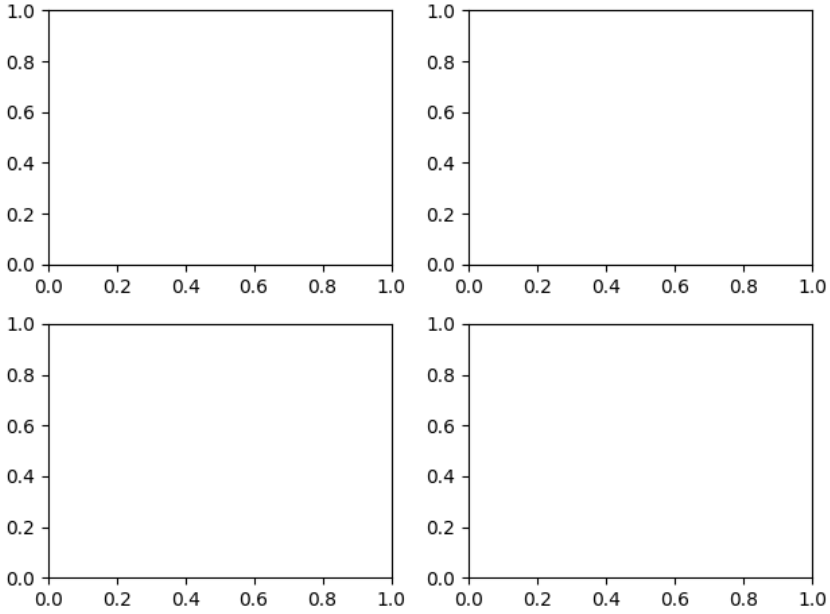


```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
```

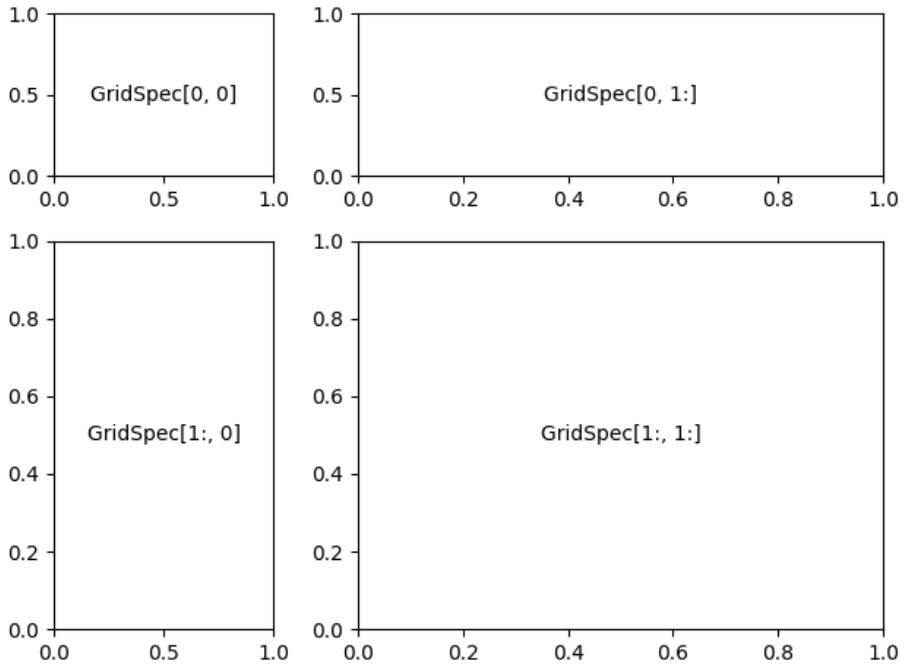
```
fig1, f1_axes = plt.subplots(ncols=2, nrows=2)
fig1.tight_layout()
```



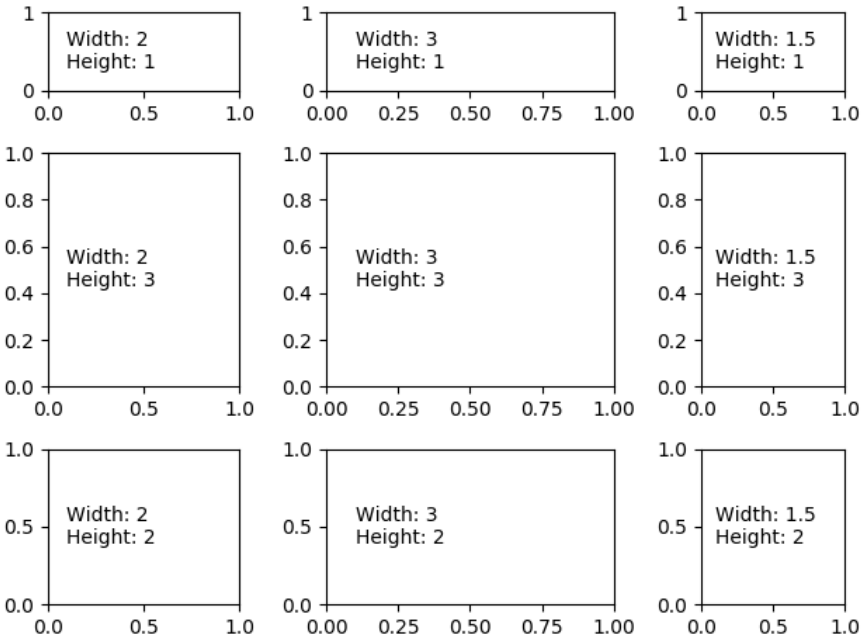

```
fig2 = plt.figure()
spec2 = gridspec.GridSpec(ncols=2, nrows=2)
f2_ax1 = fig2.add_subplot(spec2[0, 0])
f2_ax2 = fig2.add_subplot(spec2[0, 1])
f2_ax3 = fig2.add_subplot(spec2[1, 0])
f2_ax4 = fig2.add_subplot(spec2[1, 1])
fig2.tight_layout()
```



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

fig3 = plt.figure()
spec3 = gridspec.GridSpec(ncols=3, nrows=3)
anno_opts = dict(xy=(0.5, 0.5), xycoords='axes fraction', va='center', ha='center')
fig3.add_subplot(spec3[0, 0]).annotate('GridSpec[0, 0]', **anno_opts)
fig3.add_subplot(spec3[0, 1:]).annotate('GridSpec[0, 1:]', **anno_opts)
fig3.add_subplot(spec3[1:, 0]).annotate('GridSpec[1:, 0]', **anno_opts)
fig3.add_subplot(spec3[1:, 1:]).annotate('GridSpec[1:, 1:]', **anno_opts)

fig3.tight_layout()
```

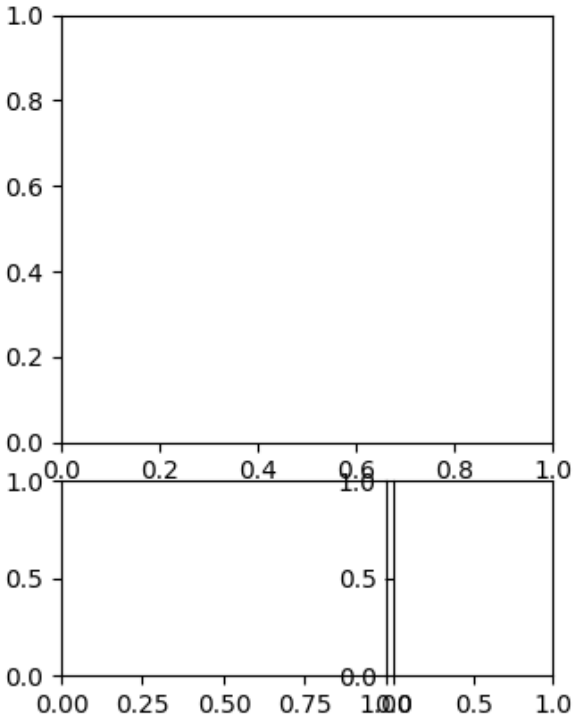


```
fig4 = plt.figure()
widths = [2, 3, 1.5]
heights = [1, 3, 2]
spec4 = gridspec.GridSpec(ncols=3, nrows=3, width_ratios=widths,
                          height_ratios=heights)
for row in range(3):
    for col in range(3):
        ax = fig4.add_subplot(spec4[row, col])
        label = 'Width: {}₩nHeight: {}'.format(widths[col], heights[row])
        ax.annotate(label, (0.1, 0.5), xycoords='axes fraction', va='center')

fig4.tight_layout()
```

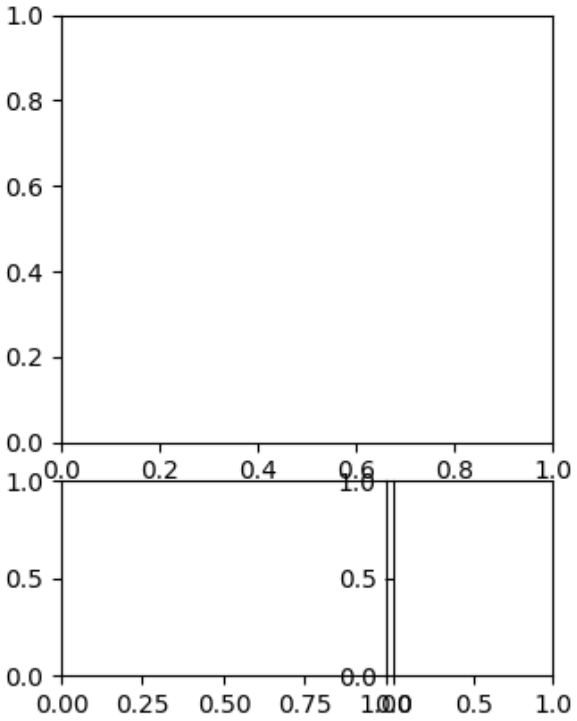


class matplotlib.gridspec.GridSpec(*nrows*, *ncols*, *figure=None*, *left=None*, *bottom=None*, *right=None*, *top=None*, *wspace=None*, *hspace=None*, *width_ratios=None*, *height_ratios=None*)

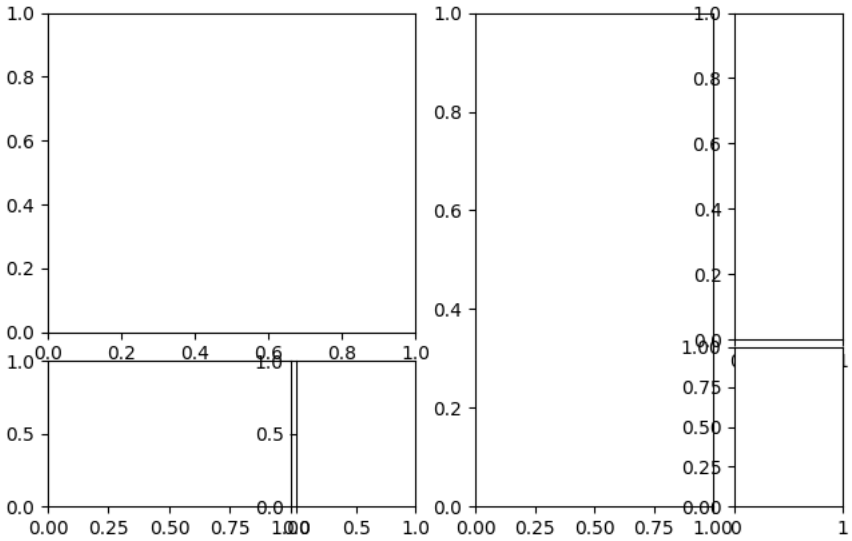
```
fig = plt.figure()
gs1 = gridspec.GridSpec(nrows=3, ncols=3,
left=0.05, right=0.48, wspace=0.05)
ax1 = fig.add_subplot(gs1[:-1, :])
ax2 = fig.add_subplot(gs1[-1, :-1])
ax3 = fig.add_subplot(gs1[-1, -1])
```



```
fig = plt.figure()
gs1 = gridspec.GridSpec(nrows=3, ncols=3,
left=0.05, right=0.48, wspace=0.05)
ax1 = fig.add_subplot(gs1[:-1, :])
ax2 = fig.add_subplot(gs1[-1, :-1])
ax3 = fig.add_subplot(gs1[-1, -1])
```



```
fig = plt.figure()
gs1 = gridspec.GridSpec(nrows=3, ncols=3,
left=0.05, right=0.48, wspace=0.05)
ax1 = fig.add_subplot(gs1[:-1, :])
ax2 = fig.add_subplot(gs1[-1, :-1])
ax3 = fig.add_subplot(gs1[-1, -1])

gs2 = gridspec.GridSpec(nrows=3, ncols=3,
 left=0.55, right=0.98, hspace=0.05)
ax4 = fig.add_subplot(gs2[:, :-1])
ax5 = fig.add_subplot(gs2[:-1, -1])
ax6 = fig.add_subplot(gs2[-1, -1])
```



class matplotlib.gridspec.SubplotSpec(*gridspec*, *num1*, *num2=None*)

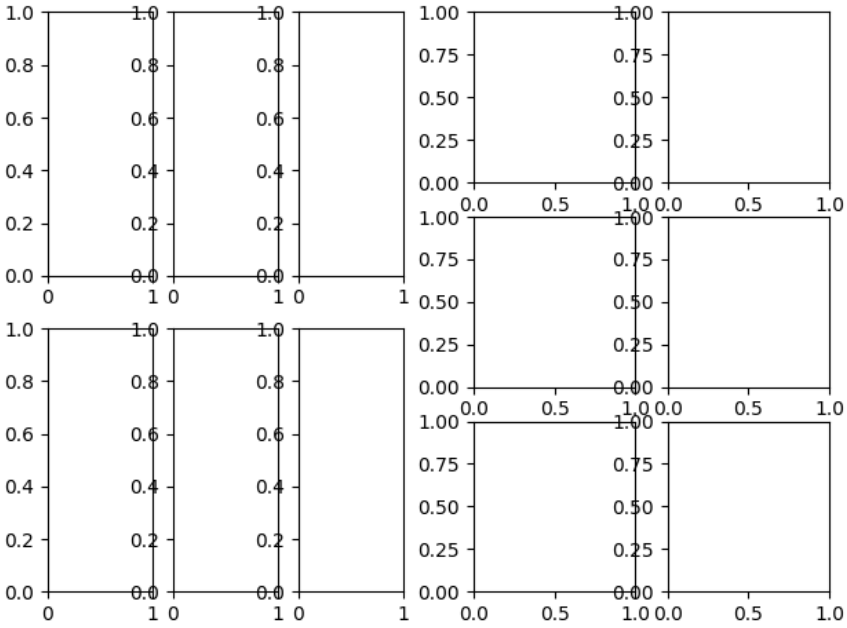
입력받은 **GridSpec**을 subplot의 **location** 값으로 반환해주는 Class

SubplotSpec Methods

- get_geometry()
  - Get the subplot geometry (n_rows, n_cols, start, stop).
- Get_gridspec()
- Get_position(figure, return_all=False)
- get_rows_columns()
- get_topmost_subplotspec()

class matplotlib.gridspec.GridSpecFromSubplotSpec(*nrows*, *ncols*, *subplot_spec*, *wspace=None*, *hspace=None*, *height_ratios=None*, *width_ratios=None*)

```
fig = plt.figure()
gs0 = gridspec.GridSpec(1, 2)
gs00 = gridspec.GridSpecFromSubplotSpec(2, 3,
subplot_spec=gs0[0])
gs01 = gridspec.GridSpecFromSubplotSpec(3, 2,
subplot_spec=gs0[1])

for a in range(2):
        for b in range(3):
                fig.add_subplot(gs00[a, b])
                fig.add_subplot(gs01[b, a])

fig.tight_layout()
```

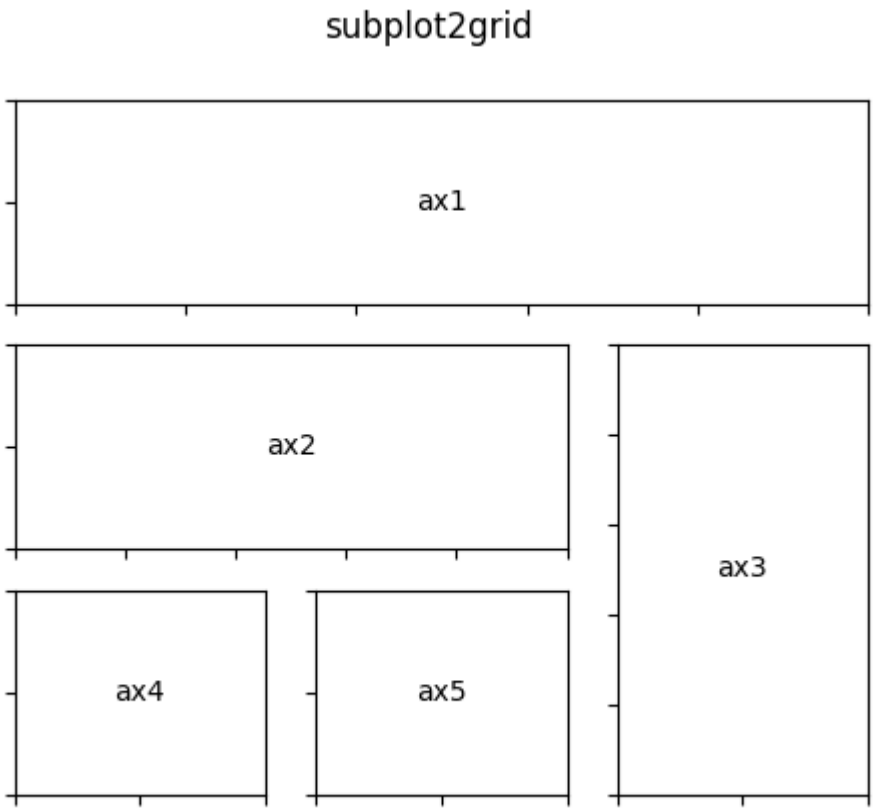


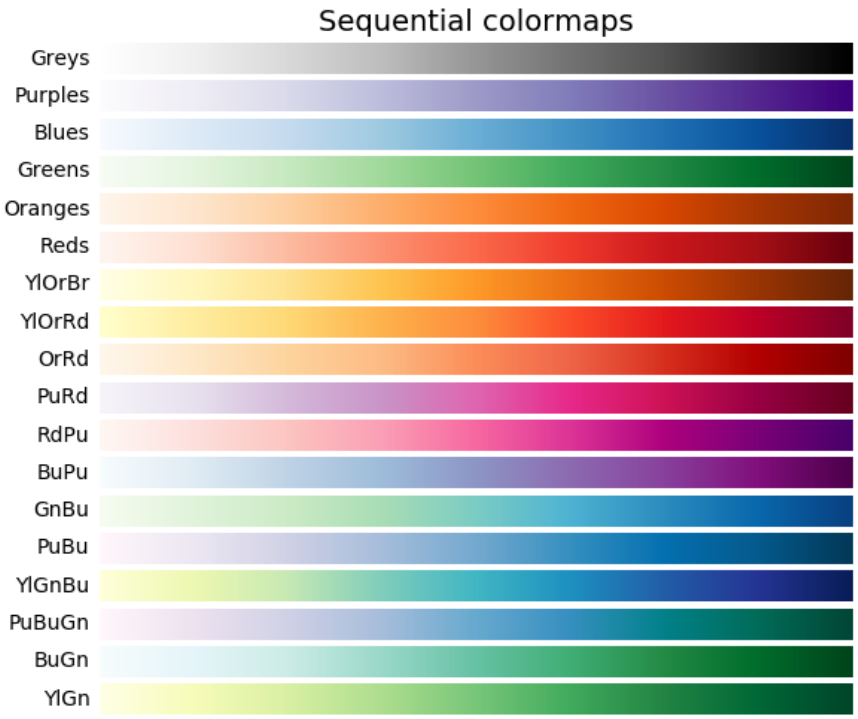
matplotlib.pyplot.subplot2grid(*shape*, *loc*, *rowspan=1*, *colspan=1*, *fig=None*, ***kwargs*)

subplot2grid(shape, loc, rowspan=1, colspan=1)

```
ax = plt.subplot2grid((2, 2), (0, 0))  ==  ax = plt.subplot(2, 2, 1)
```

```
ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
ax2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)
ax3 = plt.subplot2grid((3, 3), (1, 2), rowspan=2)
ax4 = plt.subplot2grid((3, 3), (2, 0))
ax5 = plt.subplot2grid((3, 3), (2, 1))
```





■ Colormap

○ Dataset을 직관적으로 잘 표현해 주기 위해서

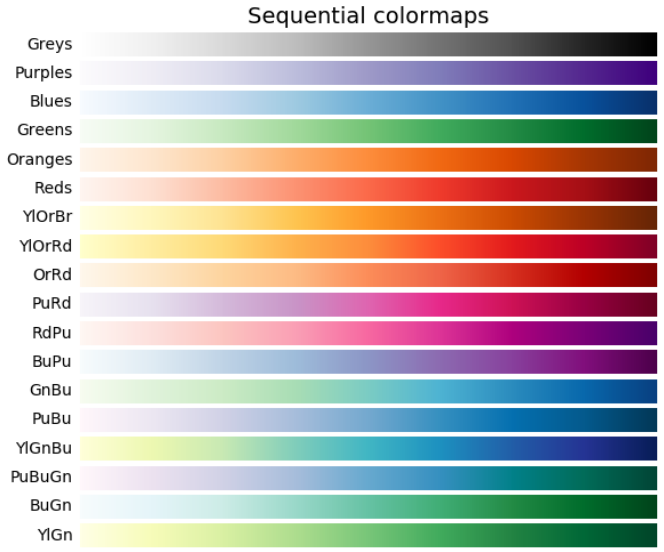
■ Classes of Colormap

○ Sequential

○ Diverging

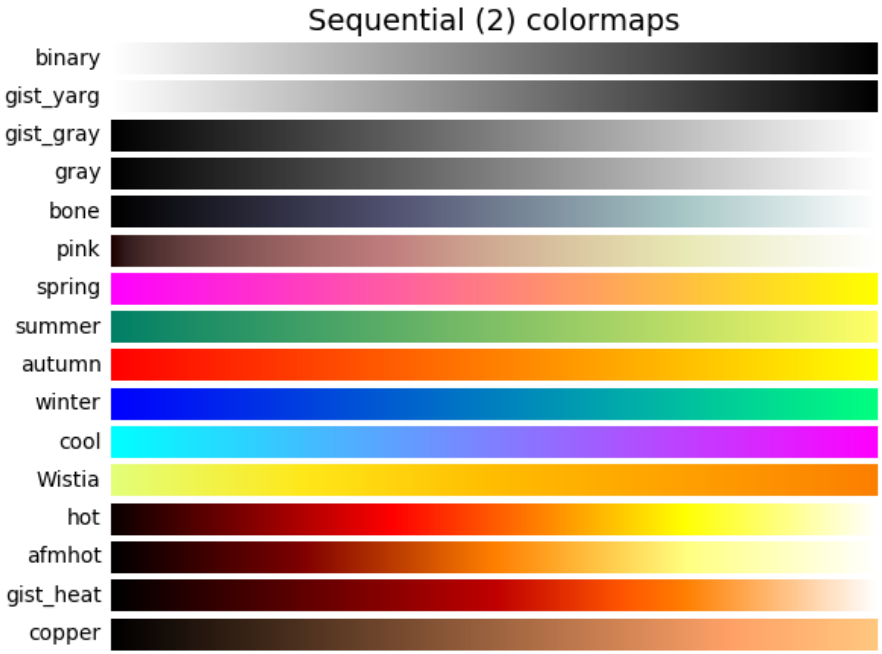
○ Qualitative

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt from matplot
import cm from colorspacious
import cspace_converter from collections
import OrderedDict

cmaps = OrderedDict()
```

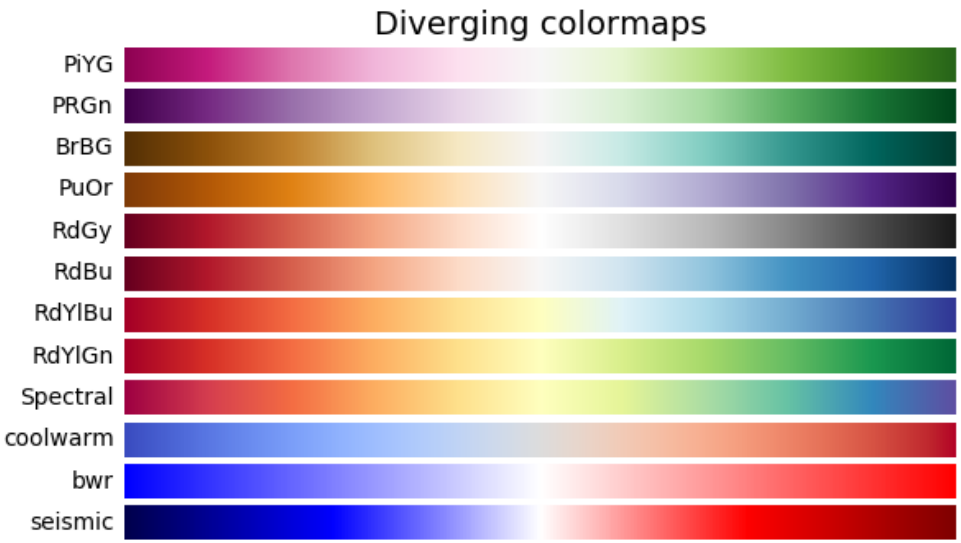



```
cmaps['Perceptually Uniform Sequential'] = [ 'viridis', 'plasma', 'i
nferno', 'magma', 'cividis']

cmaps['Sequential'] = [ 'Greys', 'Purples', 'Blues', 'Greens', 'Oran
ges', 'Reds', 'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu', 'GnB
u', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn']
```

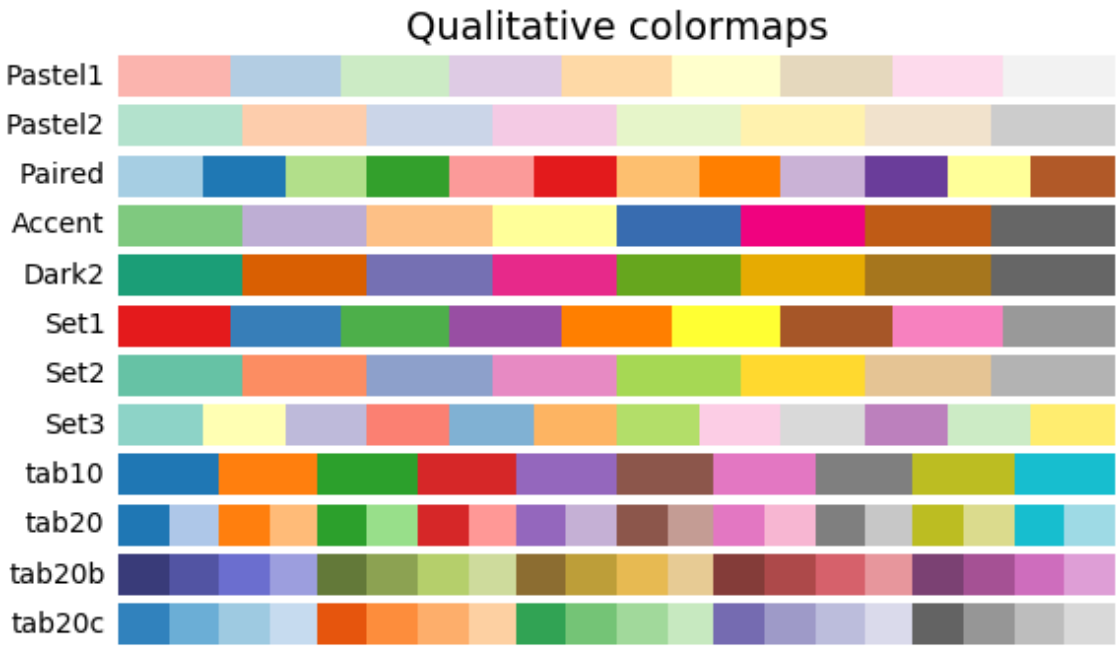
cmaps['Sequential (2)'] = ['binary', 'gist_yarg', 'gist_gray', 'gray', 'bone', 'pink', 'spring', 'summer', 'autumn', 'winter', 'cool', 'Wistia', 'hot', 'afmhot', 'gist_heat', 'copper']



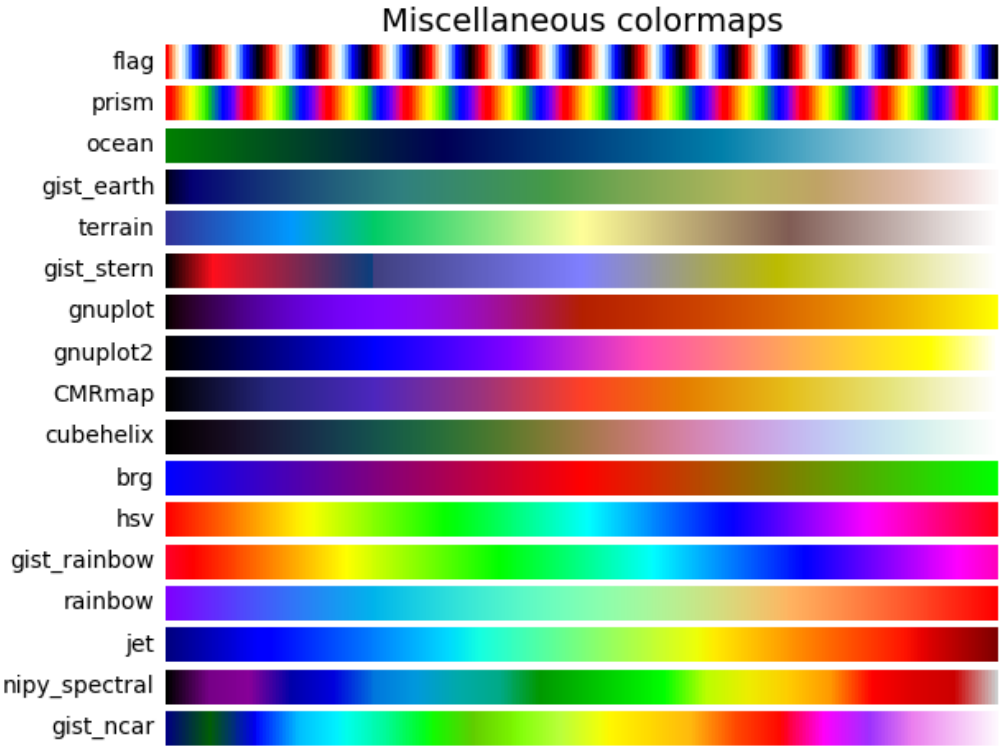
cmaps['Diverging'] = ['PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu', 'RdYlBu', '
RdYlGn', 'Spectral', 'coolwarm', 'bwr', 'seismic']



```
cmaps['Qualitative'] = ['Pastel1', 'Pastel2', 'Paired', 'Accent', 'Dark2', 'Set1', 'Set2', 'Set3', 'tab10', 'tab20', 'tab20b', 'tab20c']
```

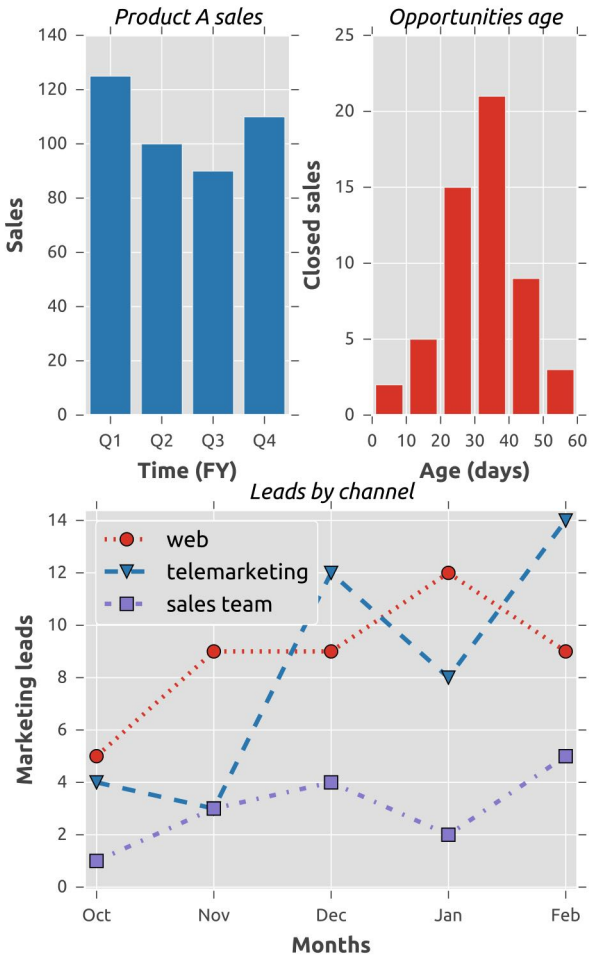


```
cmaps['Miscellaneous'] = [ 'flag', 'prism', 'ocean', 'gist_earth', 'terrain', 'gist_st
ern', 'gnuplot', 'gnuplot2', 'CMRmap', 'cubehelix', 'brg', 'hsv', 'gist_rainbow', 'r
ainbow', 'jet', 'nipy_spectral', 'gist_ncar']
```

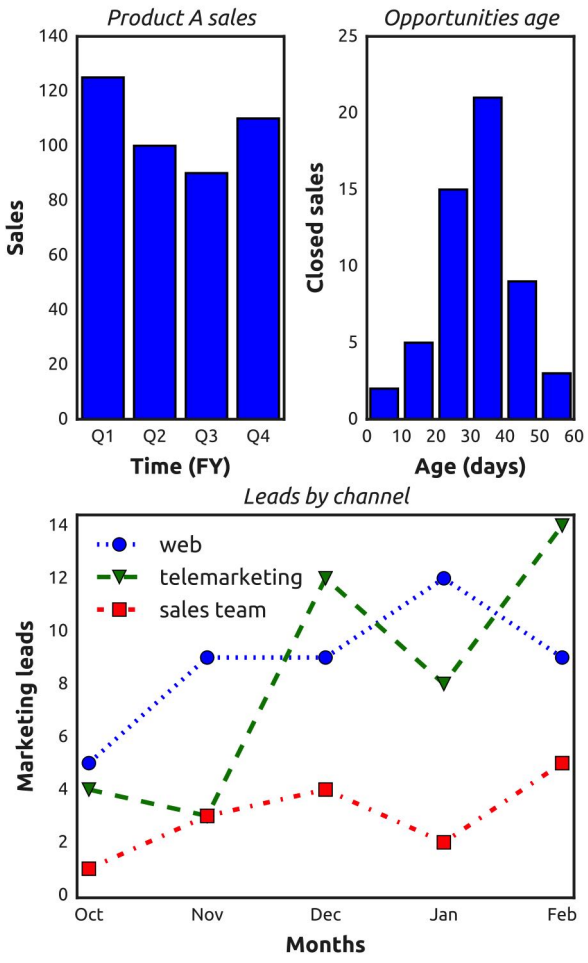


```
nrows = max(len(cmap_list) for cmap_category, cmap_list in cmaps.items())
gradient = np.linspace(0, 1, 256)
gradient = np.vstack((gradient, gradient))
def plot_color_gradients(cmap_category, cmap_list, nrows):
    fig, axes = plt.subplots(nrows=nrows)
    fig.subplots_adjust(top=0.95, bottom=0.01, left=0.2, right=0.99)
    axes[0].set_title(cmap_category + ' colormaps', fontsize=14)
    for ax, name in zip(axes, cmap_list):
        ax.imshow(gradient, aspect='auto', cmap=plt.get_cmap(name))
        pos = list(ax.get_position().bounds)
        x_text = pos[0] - 0.01
        y_text = pos[1] + pos[3]/2.
        fig.text(x_text, y_text, name, va='center', ha='right', fontsize=10)

    # Turn off *all* ticks & spines, not just the ones with colormaps.
    for ax in axes:
    ax.set_axis_off()
for cmap_category, cmap_list in cmaps.items():
    plot_color_gradients(cmap_category, cmap_list, nrows)

plt.show()
```

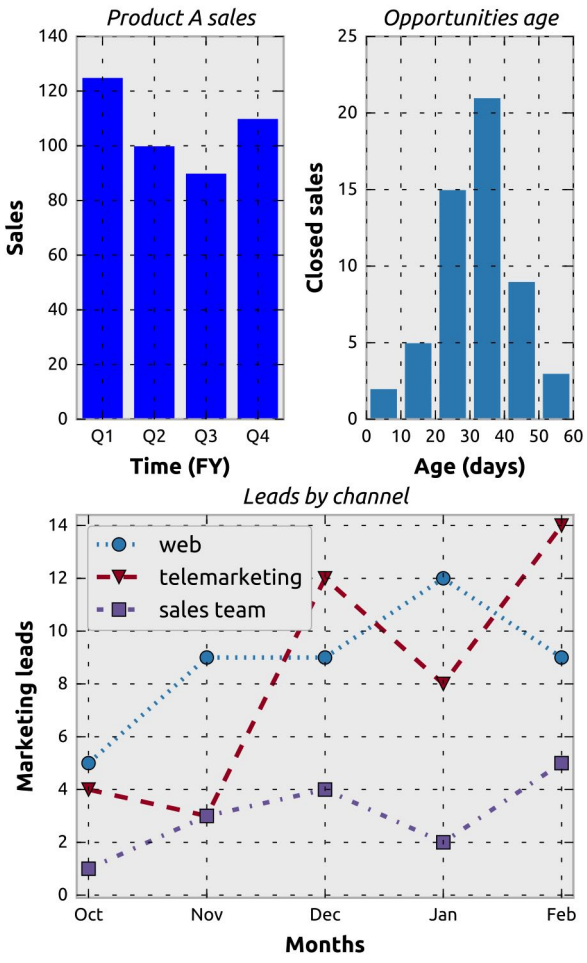
Ggplot style example



Seaborn-white style example



BMH style example



```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
plt.style.use('ggplot')
data = np.random.randn(50)

print(plt.style.available)
```

```
['seaborn-ticks', 'ggplot', 'dark_background', 'bmh', 'seaborn-poster', 'seaborn-notebook', 'fast', 'seaborn', 'classic', 'Solarize_Light2', 'seaborn-dark', 'seaborn-pastel', 'seaborn-muted', '_classic_test', 'seaborn-paper', 'seaborn-colorblind', 'seaborn-bright', 'seaborn-talk', 'seaborn-dark-palette', 'tableau-colorblind10', 'seaborn-darkgrid', 'seaborn-whitegrid', 'fivethirtyeight', 'grayscale', 'seaborn-white', 'seaborn-deep']
```

1. 아래의 함수로 Matplotlib configure 폴더를 확인

```
import matplotlib as mpl
mpl.get_configdir()
```

Default : ~/.config/matplotlib/mpl_configdir

2. mpl_configdir/stylelib/presentation.mpstyle을 만들어서 아래와 같이 작성

```
axes.titlesize : 24
axes.labelsize : 20
lines.linewidth : 3
lines.markersize : 10
xtick.labelsize : 16
ytick.labelsize : 16
```

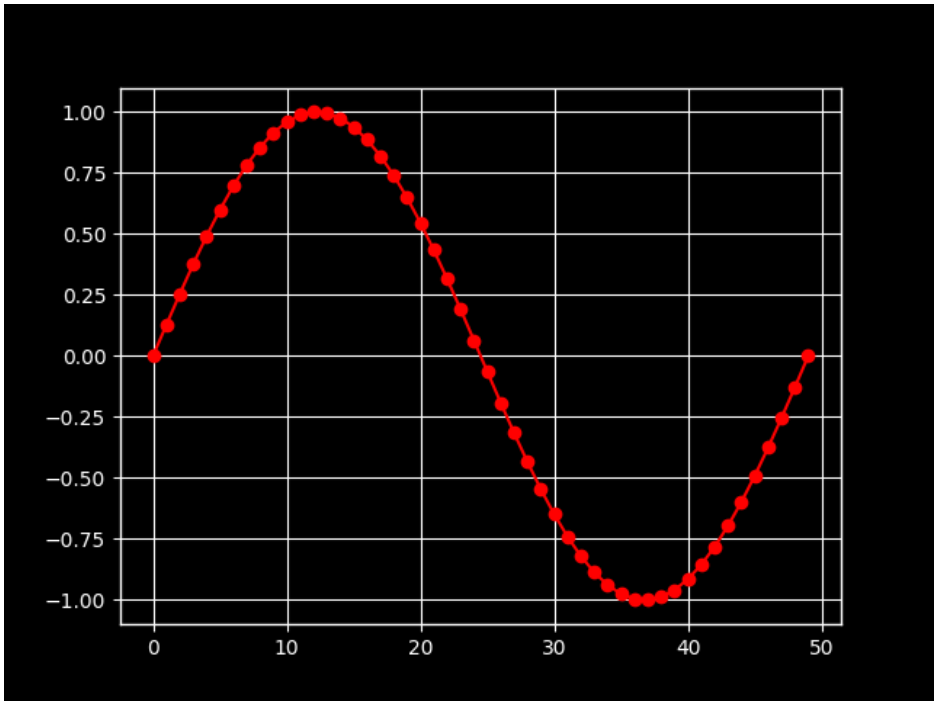
3. 아래 코드를 사용해서 custom style 사용 가능

```
import matplotlib.pyplot as plt
plt.style.use('presentation')
```

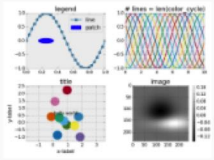
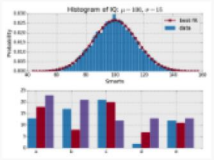
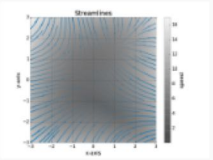
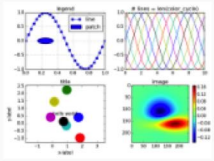
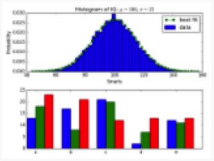
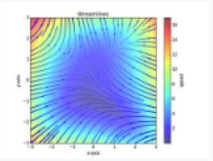
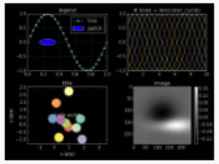
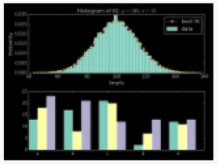
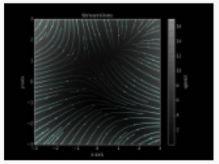
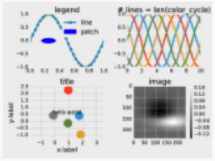
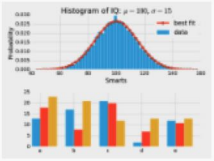
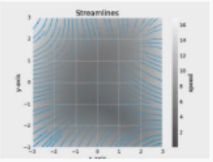
Style.use의 인자를 List를 사용하여, style을 Composing할 수 있음

```
import matplotlib.pyplot as plt
plt.style.use(['dark_background', 'presentation'])
```

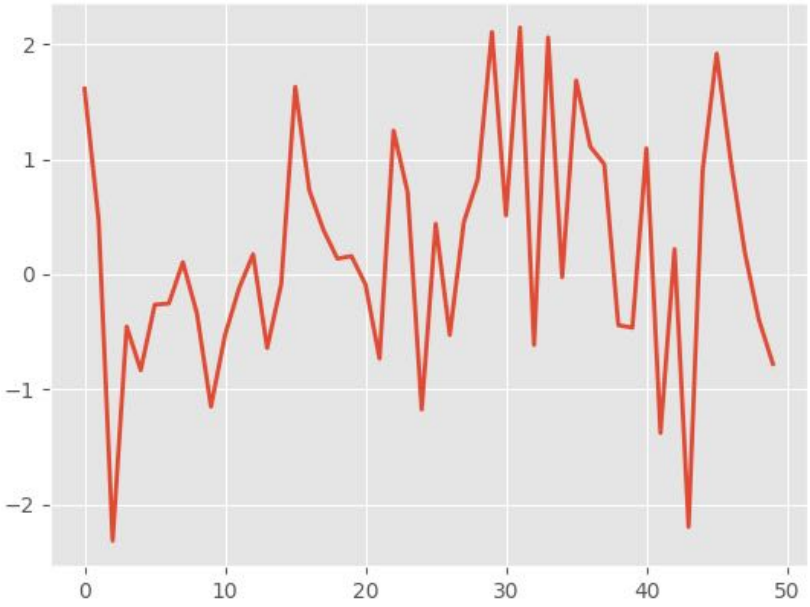
```
with plt.style.context(('dark_background')):
    plt.plot(np.sin(np.linspace(0, 2 * np.pi)), 'r-o')
plt.show()
```



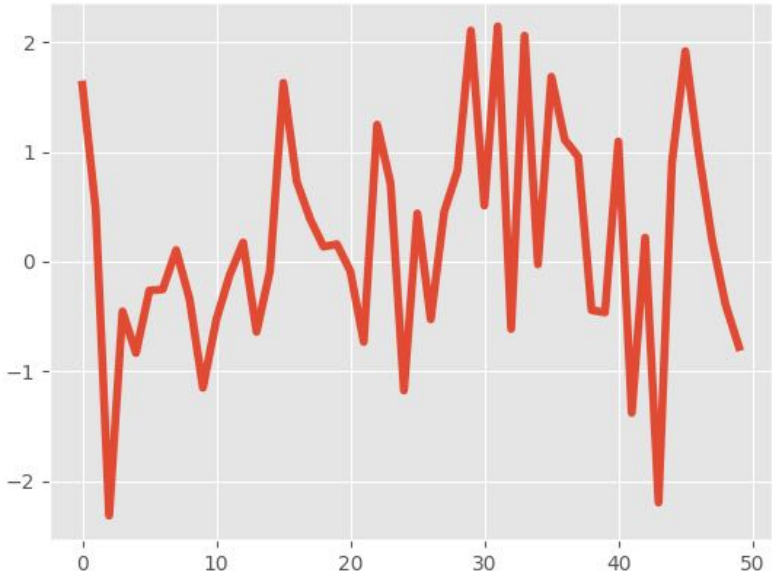
https://github.com/tonysyu/matplotlib-style-gallery

Style	artist-demo	bar-plots	streamplot
bmh			
classic			
dark_background			
fivethirtyeight			

```
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.color'] = 'r'
plt.plot(data)
```




```
mpl.rc('lines', linewidth=4, color='g')
plt.plot(data)
```

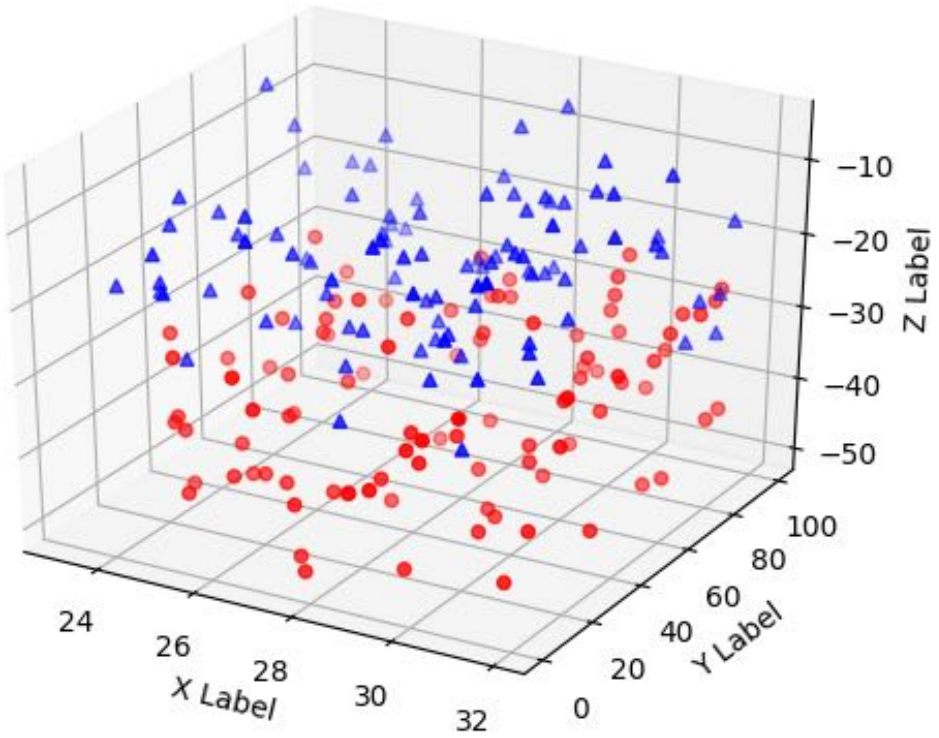


```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

Random data point 생성 함수

```
np.random.seed(19680801)

def randrange(n, vmin, vmax): ′
    return (vmax - vmin)*np.random.rand(n) + vmin

n = 100
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for c, m, zlow, zhigh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
xs = randrange(n, 23, 32)
ys = randrange(n, 0, 100)
zs = randrange(n, zlow, zhigh)
ax.scatter(xs, ys, zs, c=c, marker=m)
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```


−10
−20
−30
−40
−50
Z Label
100
80
60
40
20
0
Y Label
24
26
28
30
32
X Label

- Line plots
- Scatter plots
- Wireframe plots
- Surface plots
- Tri-Surface plots
- Contour plots
- Filled contour plots
- Polygon plots
- Bar plots
- Quiver
- 2D plots in 3D
- Text
- Subplotting

Visual Analytics

■ 과학적 시각화(Scientific visualization-SciVis)

○ 현실 세계를 시각적으로 표현

○ 물리 화학적 현상, 분자 및 생물학적 구조 등등...

■ 정보 시각화(Information visualization-InfoVis)

○ 데이터의 추상적인 정보의 시각화

○ 다차원 데이터, 그래프, 네트워크 데이터, 텍스트 데이터, 시공간 데이터 등등...

■ 시각적 분석(Visual Analytics)은 정보 시각화의 제한을 극복

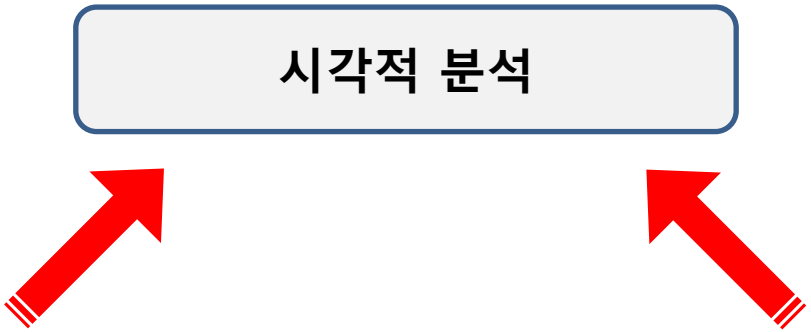
○ InfoVis는 주어진 데이터를 그대로 시각화하여 데이터의 특징을 확인하는 것에 중점

○ 그러나, Bid Data는 데이터 자체가 방대하고, 데이터의 차원이 높으며, 노이즈가 많기 때문에, 제한된 공간상에 시각화하기 여러움

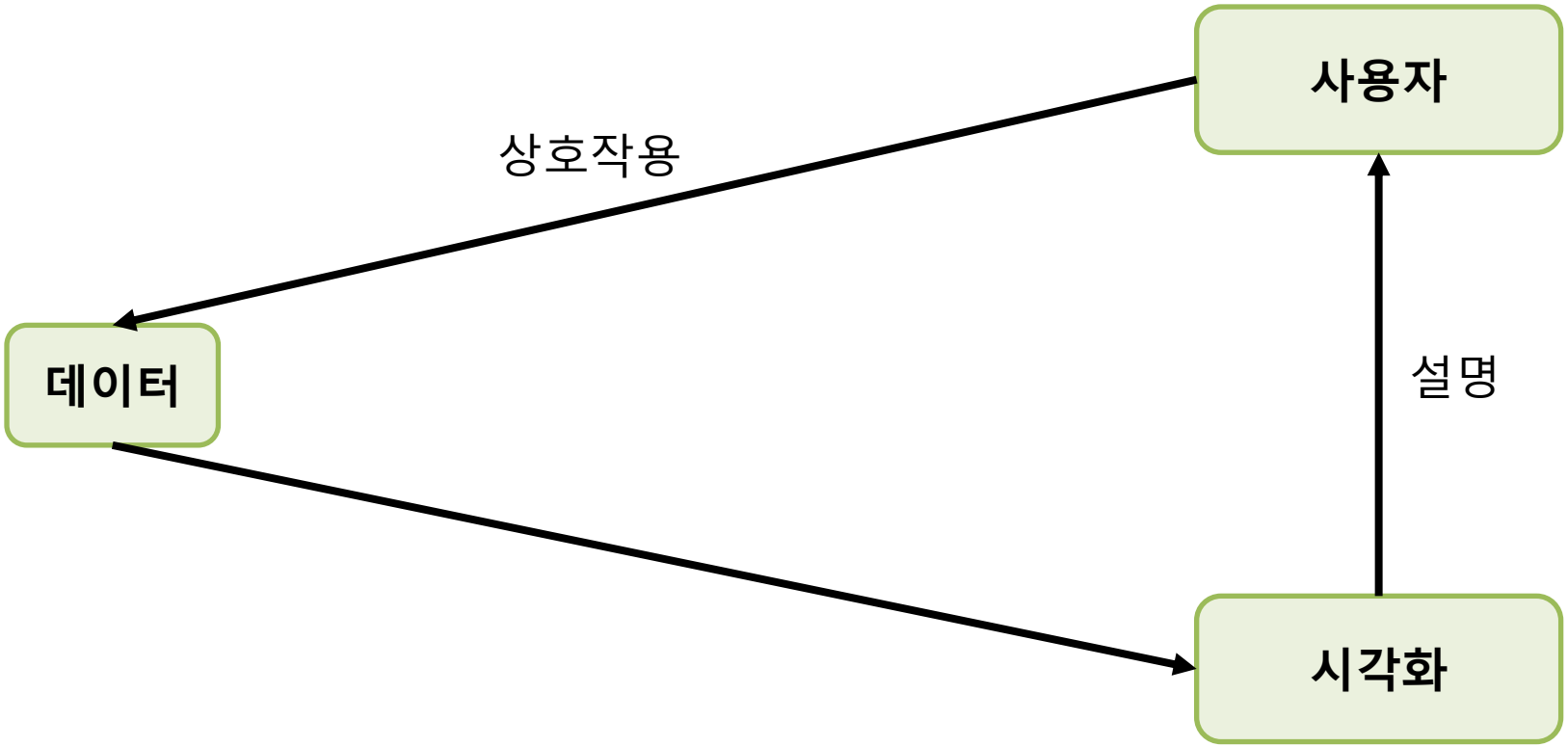
■ 정보 시각화와 시각적 분석의 주요 차이점

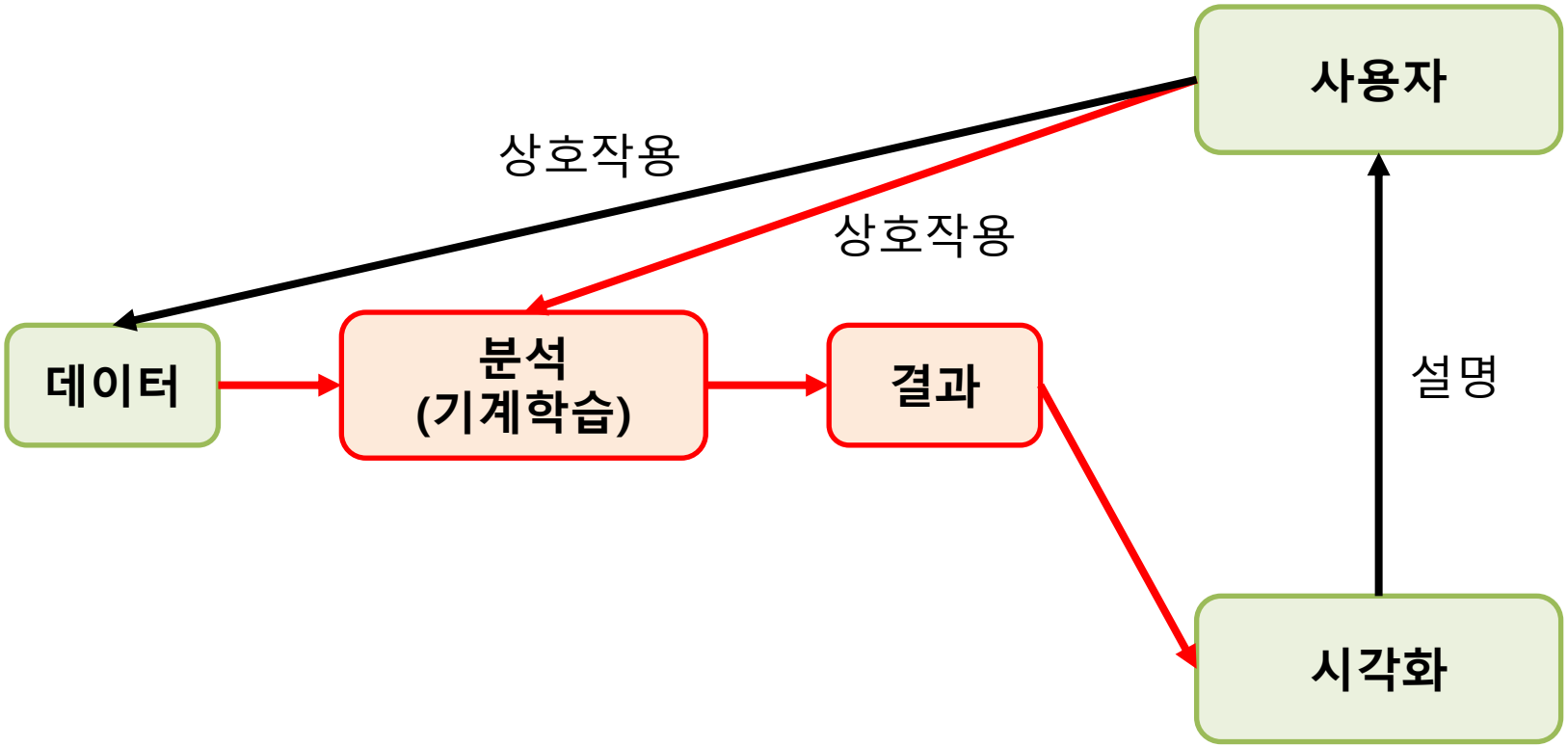
○ 기계 학습 및 기타 데이터 마이닝 기술을 사용하여 데이터에서 의미있는 정보 추출 후 시각화

○ 시각적 분석의 주된 목표는 실제 데이터 분석 작업을 지원하는 것



기계학습	정보 시각화
Automated	Interactive
Clealy defined tasks	Exploratory analysis
Fast Computation	Deeper understanding
Large data set	Limited data set


사용자
상호작용
데이터
설명
시각화


사용자
상호작용
상호작용
데이터
분석
(기계학습)
결과
설명
시각화

■ Tensorflow 데이터 임베딩