BIND SOFT

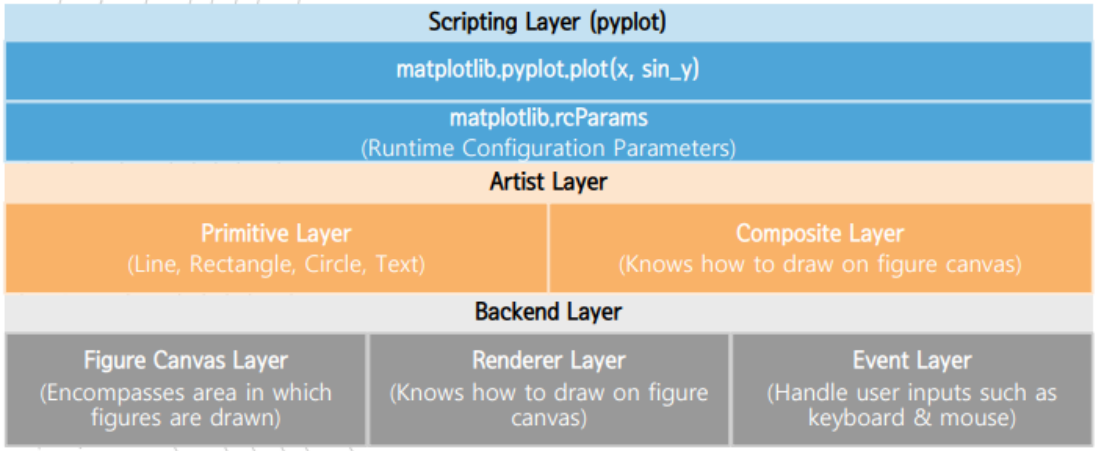
MATPLOTLIB 라이브러리

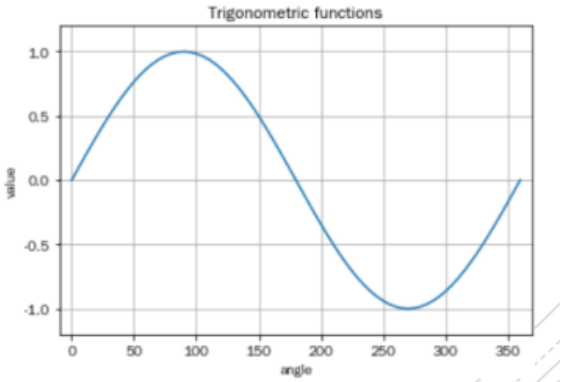
BIND SOFT ㈜바인드소프트

BIND SOFT

matplotlib

- 둘 이상의 데이터 사이 관계를 플롯으로 나타내는 가장 오래된 파이썬 플로팅 plotting 라이브러리
  - MATLAB과 유사한 오픈소스 과학 컴퓨팅 라이브러리인 SciPy의 일부로 2003년 개발되어 현재까지 업데이트가 이어짐
    - 파이썬으로 작성되었으며 NumPy 및 기타 확장 코드를 사용하여 대규모 배열에서도 우수한 성능 제공
  - 아키텍처는 세 가지 주요 계층으로 구성됨
    - 스크립팅 scripting , 아티스트 artist , 백엔드 backen



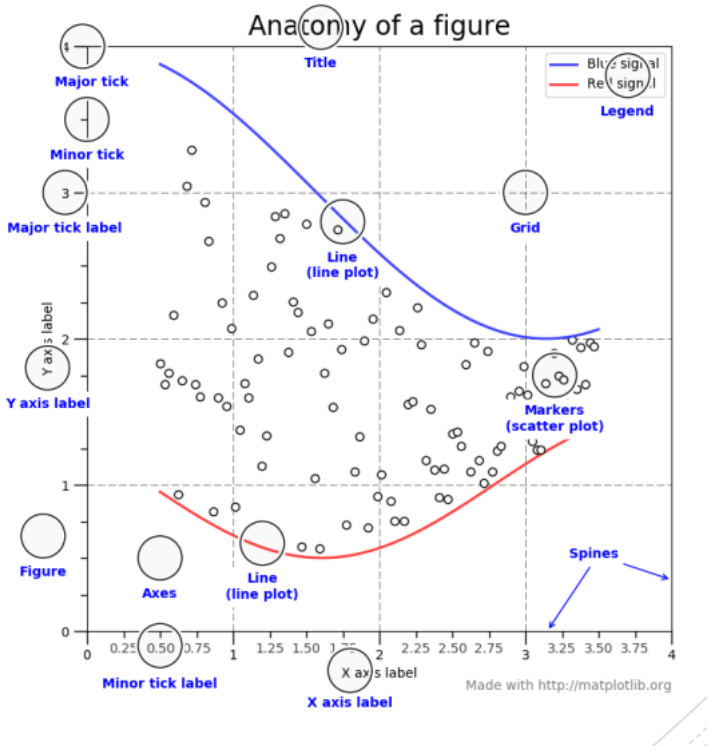


matplotlib

- 스크립팅 레이어
  - Matplotlib가 MATLAB 스크립트처럼 작동하도록 설계된 최상위 계층으로 절차적 플로팅 수행
- 아티스트 레이어
  - 스크립팅 레이어에 비해 더 많은 사용자 정의를 수행할 수 있으므로 고급 플롯에 사용
    - 모든 플롯 요소에 대한 최상위 컨테이너인 피겨 figure 를 완벽하게 제어하고 미세 조정
  - 렌더러를 사용하여 캔버스에 그리는 Artis 객체 기반 플롯팅 수행
    - 여러 그림/축을 처리할 때 모든 하위 플롯이 Artis 객체에 할당되기 때문에 현재 활성화된 그림/축을 혼동하지 않음
  - Matplotlib 그림에서 볼 수 있는 모든 것은 Artis 인스턴스
    - 제목, 선, 눈금 레이블, 이미지 등은 모두 개별 Artis
  - Artis 유형
    - 기본 유형: Line2D, Rectangle, Circle, Text
    - 복합 유형: Axis, Tick, Axes, Figure
- 백엔드 레이어
  - PyQt5, ipympl, GTK3, Cocoa, Tk, wxPython과 같은 툴킷이나 PostScript와 같은 그리기 언어와 통신하여 모든 실제 그리기 작업 처리
  - 대부분의 사용자는 이 계층을 직접 다룰 필요가 없음
    - FigureCanvas: Figure가 렌더링되는 캔버스
    - Renderer: 그리기/렌더링 작업을 처리하는 추상 기본 클래스
      - FigureCanvas에서 그리는 일을 담당
    - Event: 키보드 및 마우스 클릭과 같은 사용자 입력 처리

matplotlib

- 기본 용어 정리
  - Figure: 플롯 전체
  - Axes: 플롯이 그려지는 좌표축
    - Figure의 subplot으로 다중 subplot 허용
  - Spines: 테두리
  - X axis: X 축
  - Y axis: Y 축
  - Tick : 눈금
    - 주, 보조 눈금으로 나뉨
  - Line: 라인 플롯 line plot 의 선
  - Markers: 선이나 산포도 scatter 플롯의 점
  - Grid: 격자
  - Title: 제목
  - Label: 각 축이나 눈금 등에 붙이는 텍스트
  - Legend : 범례
    - 여러 플롯의 의미를 구분하기 위한 별도 표시



아티스트와 스크립트 레이어 플로팅 비교

BIND SOFT

* 아티스트

```
from matplotlib.backends.backend_agg import
FigureCanvasAgg
from matplotlib.figure import Figure
import numpy as np
from PIL import Image

x = np.random.standard_normal(20000)
fig = Figure()
canvas = FigureCanvasAgg(fig)

ax = fig.add_subplot(111)

ax.set_title("Normal Distribution")
ax.hist(x, 100)
canvas.draw()
image = np.frombuffer(canvas.tostring_rgb(),
dtype=np.uint8)
w, h = fig.canvas.get_width_height()
im = Image.frombytes("RGB", (w, h), image)
im.show()
```

* 스크립트

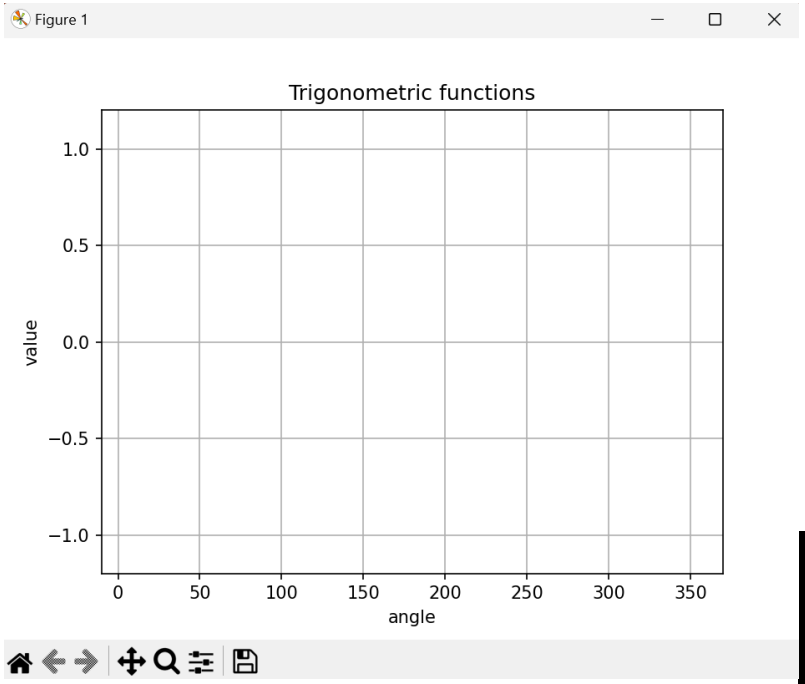
```
import matplotlib.pyplot as plt
import numpy as np

x =
np.random.standard_normal(20000)

plt.title("Normal Distribution")
plt.hist(x, 100)
plt.show()
```

BIND SOFT

플롯 기본

- 기본 피겨 figure 와 기본 좌표축 axes 에 그리기
  - pyplot.title(name,...): 타이틀
    - loc: 표시 위치. 'left', 'center', right' 중 하나
  - pyplot.xlim(*args, **kwargs), pyplot.ylim(...): x, y축 범위
    - xlime: left, right, ylim: bottom, top
  - pyplot.xlabel(xlabel, loc=None, **kwargs), pyplot.ylabel(...): x, y축 이름
    - loc: 표시 위치. 'center' 공통. xlabel: 'left', 'right'. ylabel: 'bottom', 'top'
  - pyplot.grid(visible=None, which='major', axis='both', **kwargs): 격자
    - visible: 불 타입 표시 유무
    - axis: 축 선택. 'x', 'y', 기본값은 None으로 양쪽
    - color: '#rrggbb' 또는 색상표 문자열(예: 'red', 'blue', ...) (기본값 자동)
    - linestyle: 선 스타일. '-'(기본값), '—', '-.', ':' 중 하나
    - alpha: 0.0 ~ 1.0 사이 실수 타입 투명도 (기본값 1)
    - linewidth: 정수 타입 선 굵기
  - pyplot.show(*, block=None): 모든 그림 표시 및 피겨를 비움
    - block: 불 타입으로 True이면 모든 그림이 닫힐 때까지 대기.

플롯 기본



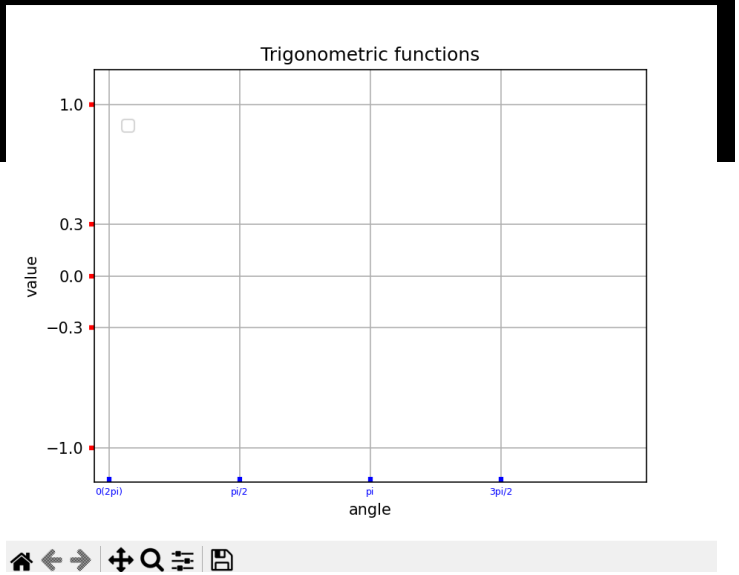

```python
import matplotlib.pyplot as plt

plt.title("Trigonometric functions")
plt.xlim(-10, 360+10)
plt.ylim(-1.2, 1.2)
plt.xlabel("angle")
plt.ylabel("value")
plt.grid(True)
plt.show()
```

플롯 기본

- pyplot.xticks(ticks=None, labels=None, **kwargs): x축 간격 표시 눈금
  - ticks: 리스트 타입 시퀀스. None은 표시 안함
  - labels: 리스트 타입 시퀀스. ticks 값 대신 사용할 이름
- pyplot.yticks(...): x축 간격 표시 눈금
- pyplot.tick_params(axis='both', **kwargs): 축 눈금 세부 설정
  - axis: 축 선택. 'x', 'y', 'both' (기본값)
  - direction: 축선 기준 눈금 방향으로 'in', 'out' (기본값), 'inout' 중 하나
  - length: 정수 타입 눈금 길이. (기본값 3)
  - width: 정수 타입 눈금 굵기. (기본값 1)
  - color: 눈금 색. (기본값 'black')
  - labelsize: 정수 타입 레이블 크기. (기본값은 10)
  - labelcolor: 레이블 색. (기본값 'black')
- pyplot.legend(...): 범례
  - label: 리스트 타입 시퀀스로 요소의 순서는 플롯팅 순
    - 플롯팅할 플롯이 없으면 표시 안되고 플롯에서 label을 사용하면 생략 가능
  - loc: 4방향에 대한 0.0 ~ 1.0 실수 타입 튜플(x, y)
    - 생략하면 적당한 곳 자동 선택

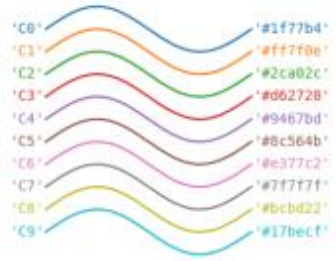
플롯 기본

```
import matplotlib.pyplot as plt

plt.title("Trigonometric functions")
plt.xlim(-10, 360+10)
plt.ylim(-1.2, 1.2)
plt.xlabel("angle")
plt.ylabel("value")
plt.grid(True)
plt.xticks([0, 90, 180, 270], ['0(2pi)', 'pi/2', 'pi', '3pi/2'])
plt.yticks([-1, -0.3, 0, 0.3, 1])
plt.tick_params(axis='x', width=3, direction='in',
color='blue', labelsize=6, labelcolor='blue')
plt.tick_params(axis='y', width=3, color='red')
plt.legend(['sine', 'cosine'], loc=(0.05, 0.85))
plt.show()
```



라인 플롯

- 독립변수 x에 대한 종속변수 y의 변화를 선으로 표현
  - pyplot.plot(*args, **kwargs)
    - 주요 인자
      - x: x축 실수 또는 배열 형태 데이터
        - n개의 x축 데이터는 n차원 배열을 사용
      - y: y축 실수 또는 배열 형태 데이터
        - 생략하면 x는 x의 인덱스, y는 x의 데이터
      - label: 플롯 이름
      - color: 색상으로 문자열(예: 'red', 'blue', ...) 또는 RGB 문자열('#rrggbb')
        - 색상을 지정하지 않으면 플롯마다 'C0' ~ 'C9'까지 돌아가며 사용
      - linestyle: 선 스타일. '-' (기본값), '—', '-.', ':' 중 하나
      - marker: 표시 스타일. '.,ov^<>12348,spP*hH+xDd|_' 중 하나



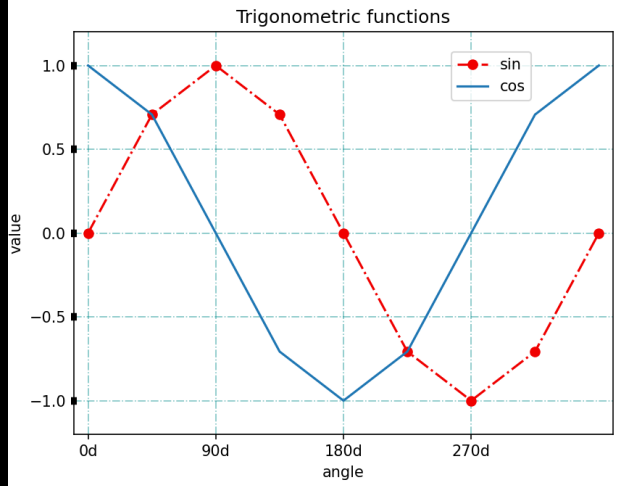
Cycler 색상

CSS 색상

라인 플롯

- sine, cosine 라인 플롯팅
  - 0 ~ 2π 범위 sine, cosine 배열 생성 후 마커(sine 만)와 함께 선 플로팅. step을 줄이면 좀더 부드러운 곡선을 얻을 수 있음

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

plt.title("Trigonometric functions")
plt.xlabel("angle")
plt.ylabel("value")
plt.grid(True, color='darkcyan', alpha=0.5, linestyle='-.')
plt.xlim(-10, 360+10)
plt.ylim(-1.2, 1.2)

x = np.arange(0, 360+1, 45)
sin_y = np.sin(x * np.pi / 180)
cos_y = np.cos(x * np.pi / 180)

plt.plot(x, sin_y, label='sin', color='#f00000', linestyle='-
.', marker='o')
plt.plot(x, cos_y, label='cos')
plt.legend(loc=(0.7, 0.83))
plt.xticks([0, 90, 180, 270], ['0d', '90d', '180d', '270d'])
plt.tick_params('y', direction='inout', width=5)
plt.show()
```



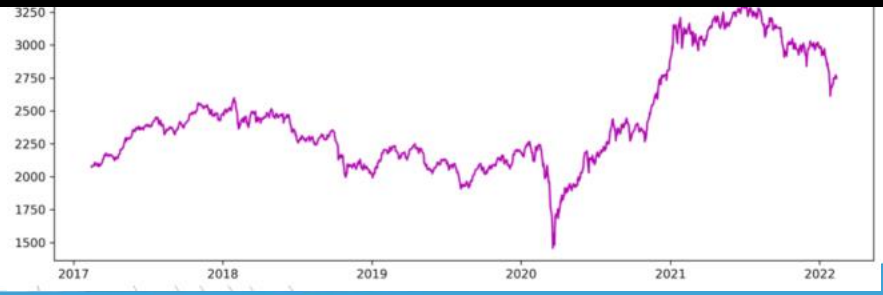
주가 데이터 얻기

- 전세계 주가는 pandas-datareader 패키지로 야후 파이낸스 사이트 (https://finance.yahoo.com/)를 통해 얻을 수 있음
  - 설치
    - sudo pip3 install pandas-datareader
  - 모듈
    - from pandas_datareader import data as pdr
  - 주가 데이터
    - 데이터 구조
      - Date(날짜), High(고가), Low(저가), Open(시가), Close(종가), Valume(거래량), Adj Close(수정종가) 필드로 구성된 테이블 구조
        - 데이터 분석에는 데이터의 연속성을 위해 수정 종가 사용
        - 수정 종가는 분할, 배당, 배분, 신주 발생을 고려해 주식 가격을 조정한 가격
        - 야후 파이낸스는 거래량 데이터의 경우 십자리 이하는 버림
    - 종목 코드로 데이터 얻기
      - kospi = pdr.DataReader('KS11', 'yahoo'): 2017년부터 현재까지 코드피 전체 데이터
      - sec = pdr.DataReader('005930.KS', 'yahoo', '20200101'): 삼성전자 2020.01.01 ~ 현재까지
      - lg = pdr.DataReader('066570.KS', 'yahoo', '20210101', '20211231': LG 전자 2021년 전체 (2021.01.01 ~ 2021.12.31)

- 야후 파이낸스에서 코스피(주식 코드 'KS11')의 주가 데이터를 읽어온 후 ['Adj Close'] 필드를 라인 플롯으로 플롯팅
  - DataReader()는 pandas.core.frame.DataFrame() 객체 반환
  - DataFrame()['High']은 pandas.core.series.Series() 객체 반환
    - pyplot.plot()는 Series() 객체의 첫 번째 열을 x (Data), 두 번째 열을 y (Adj Close)로 해석

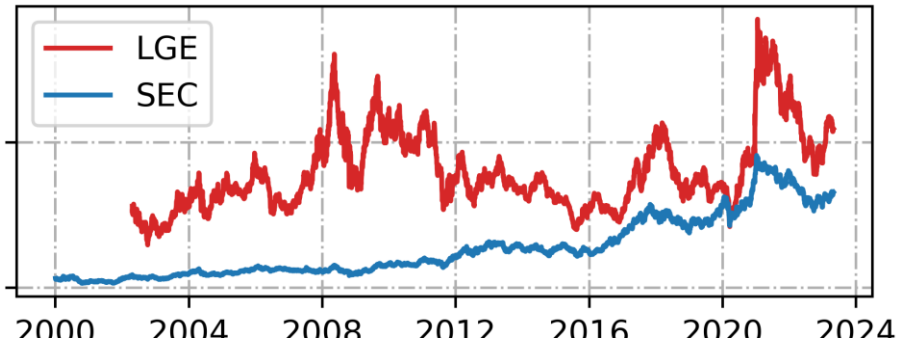
```
from pandas_datareader import data as pdr
import matplotlib.pyplot as plt
import yfinance as yfin
from datetime import datetime
yfin.pdr_override()

start_date = datetime(2007,1,1)
end_date = datetime(2020,3,3)

kospi = pdr.get_data_yahoo('AAPL', start_date, end_date)

fig = plt.figure(figsize=(12, 4), dpi=300)
plt.plot(kospi['High'], label="KOSPI", color='#b000b0')
plt.show()
```

Date						
2017-02-20	2085.590088	2077.129883	2084.159912	2084.389893	297200	2084.389893
2017-02-21	2108.479980	2085.000000	2085.969971	2102.929932	292500	2102.929932
2017-02-22	2108.979980	2101.560059	2106.419922	2106.610107	312200	2106.610107
2017-02-23	2108.989990	2103.110107	2106.149902	2107.629883	430900	2107.629883
2017-02-24	2107.830078	2090.050049	2106.429932	2094.120117	385400	2094.120117
...	...	...	...	...	...	...
2022-02-11	2766.699951	2735.080078	2739.139893	2747.709961	480300	2747.709961
2022-02-14	2724.719971	2688.239990	2715.100098	2704.479980	616000	2704.479980
2022-02-15	2716.449951	2665.469971	2712.449951	2676.540039	588700	2676.540039
2022-02-16	2730.429932	2711.340088	2719.610107	2729.679932	422100	2676.540039
2022-02-17	2770.659912	2711.989990	2735.110107	2744.090088	602261	2729.679932



LG전자와 삼성전자 주식 추이

```
from pandas_datareader import data as pdr
import matplotlib.pyplot as plt
import yfinance as yfin
yfin.pdr_override()

lge = pdr.get_data_yahoo('066570.KS')
sec = pdr.get_data_yahoo('005930.KS')

fig = plt.figure(figsize=(12, 4), dpi=300)
plt.plot(lge['Close'], label="LGE", color='C3')
plt.plot(sec['Close'], label="SEC", color='C0')
plt.grid(True, linestyle='-.')
plt.legend()
plt.show()
```

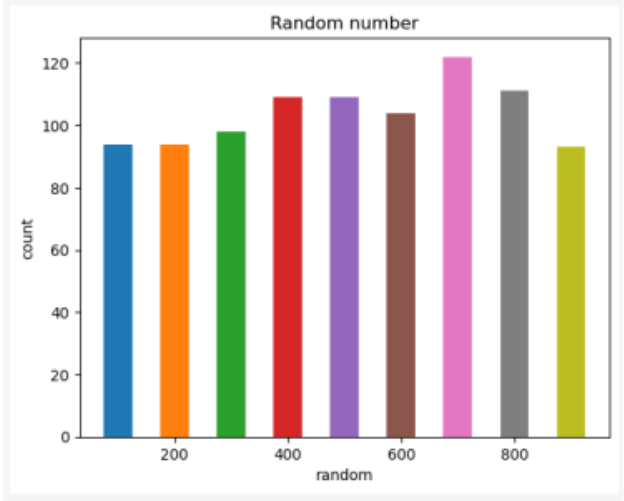


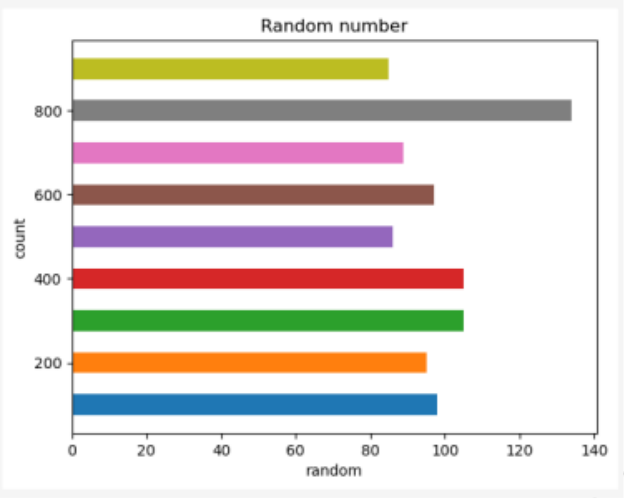
바 플롯

- 연속적이지 않은 x에 대해 y의 변화를 막대 타입으로 표현
  - pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', **kwargs): 세로 막대
    - x: 막대의 x 좌표 배열 형태 데이터
    - height: 막대 높이의 높이(y 좌표)를 나타내는 배열 형태 데이터
    - width: 막대 너비. 기본값은 0.8
    - Bottom: 스택 바 타입으로 아래 쪽으로 표시할 막대 높이 배열. (기본값 0)
    - Align: x 축 눈금 기준 막대 정렬 방법. 'center' (기본값), 'edge'
    - 기타: color, edgecolor, linewidth, tick_label 등
  - pyplot.barh(y, with, height=0.8, left=None, align='center', **kwargs): 가로 막대
    - y: 막대의 y 좌표 배열 형태 데이터
    - width: 막대의 폭(x 좌표)을 나타내는 배열 형태 데이터
    - height: 막대 높이. 기본값은 0.8
  - 난도 발생 빈도를 바 플롯으로 플롯팅
    - 0 <= n < 1000 구간 난수 생성을 100,000회 수행하면서 리스트를 이용해 각 수의 발생 빈도 카운트 (난수값이 리스트의 인덱스)
    - 결과에서 [100::100] 위치의 발생빈도를 바 플롯으로 표시

바 플롯

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

count = [0] * 1_000
for i in range(100_000):
    x = np.random.randint(1_000)
    count[x] += 1

x = np.array([i * 100 for i in range(1, 9+1)])
y = np.array(count[100::100])
print(y)
plt.title("Random number")
plt.xlabel("random")
plt.ylabel("count")
plt.bar(x, y, width=50, color=['C0', 'C1',
'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8'])
plt.show()

plt.title("Random number")
plt.xlabel("count")
plt.ylabel("random")
for i in range(len(x)):
    plt.barh(x[i], y[i], height=50)
plt.show()
```

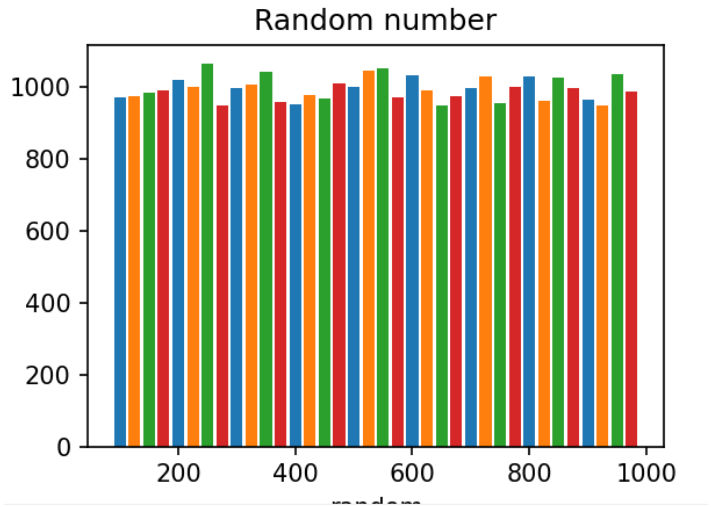




- x가 같고 y가 다른 n개의 바 플롯팅시 중첩 방지
  - 플롯팅 시 일정값 만큼 x 축을 이동해 중첩 방지

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

def make_rand(n, m):
    count = [0] * n
    for _ in range(m):
        count[np.random.randint(n)] += 1
    return count

def init_plot():
    plt.title("Random number")
    plt.xlabel("random")
    plt.ylabel("count")

def rand_plot(x_offset, n, m, s):
    r = make_rand(n, m)
    x = np.array([i * 100 for i in range(1,
9+1)])
    y = np.array(r[s::s])
    plt.bar(x+x_offset, y, width=20)

def show_plot():
    plt.show()
```

```
if __name__ == '__main__':
    init_plot()
    for x_offset in [0, 25, 50,
75]:
        rand_plot(x_offset,
1_000, 100_0000, 100)
    show_plot()
```



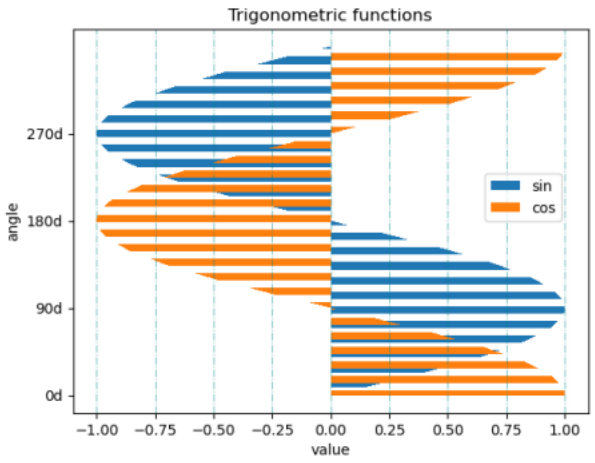
Sine, cosine 값을 가로 바 플롯으로 플로팅하시오.

- 절차를 함수로 구조화해 sine, cosine 값을 가로 바 플롯으로 플롯팅
  - data 생성: make_sincos(a, b)
    - 0 <= x < 360 사이 x 값 생성
      - x = arange(0, 360)
    - x를 라디안으로 바꾼 후 sin_y, cos_y 값 생성
      - sin_y, cos_y = sin(x*np.pi/180), cos(y*np.pi/180)
  - 플롯팅: init_plot(), sincos_plot(), show_plot()
    - 타이틀, x, y 레이블, 그리드 추가
    - 생성한 data를 가져와 barh()로 sin_y, cos_y 플롯팅
      - cos_y를 플롯팅할 때 정렬을 'edge'로 설정해 sin_y와 중첩 최소화
    - x 축 기준 0.8, y 축 기준 0.5 위치에 범례 추가
    - y축 눈금 0, 90, 180, 270에 대해 레이블 '0d', '90d', '180d', '270d' 추가

- 과연 sine, cosine 값을 바 플롯으로 플롯팅하는 것이 적합한가?

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

def make_sincos(a, b):
    x = np.arange(a, b)
    sin_y = np.sin(x * np.pi / 180)
    cos_y = np.cos(x * np.pi / 180)
    return x, sin_y, cos_y

def init_plot():
    plt.title("Trigonometric functions")
    plt.xlabel("value")
    plt.ylabel("angle")
    plt.grid(True, axis='x', color='darkcyan', alpha=0.5, linestyle='-.')

def sincos_plot(a, b):
    x, sin_y, cos_y = make_sincos(a, b)
    plt.barh(x, sin_y, label='sin', height=0.5)
    plt.barh(x, cos_y * -1, label='cos', align='edge', height=0.5)

    plt.legend(loc=(0.7, 0.83))
    plt.yticks([0, 90, 180, 270], ['0d', '90d', '180d', '270d'])

def show_plot():
    plt.show()
```

```python
if __name__ == '__main__':
    init_plot()
    sincos_plot(0, 360)
    show_plot()
```



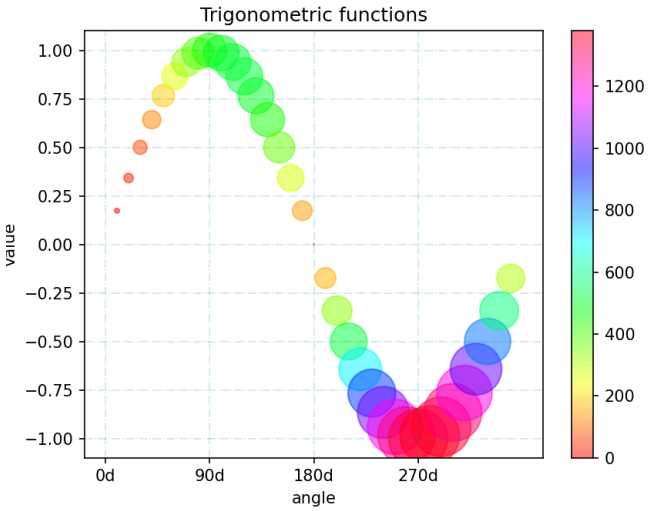
스캐터 플롯

- 서로다른 2개의 독립변수 x1와 x2의 관계를 산점도로 표현
  - pyplot.scatter(x, y, s=None, c=None, marker=None,..., **kwargs)
    - x, y; x, y 축 실수 또는 배열 형태 데이터
    - s: 실수 또는 배열 형태의 (점 ** 2)의 마커 크기. 기본값은 rcParams['lines.markersize'] ** 2
    - c: 배열 형태의 마커 색상
    - marker: 마커 스타일. 기본값은 rcParams['scatter.marker'] == 'o'
    - cmap: c가 배열인 경우만 사용하며, 컬러맵 인스턴스 또는 'hsv'와 같은 이름. 기본값은 'viridis'
    - linewidths: 실수 또는 배열 형태의 마커의 테두리 선 폭. 기본값은 1.5
    - edgecolors: 마커의 테두리 색. 'face'(기본값), 'none' 또는 색상 또는 색상 배열 형태
      - 'face': 면과 동일
      - 'none' : 테두리를 그리지 않음
  - pyplot.colorbar(...): 플롯에 색상바 추가

스캐터플롯-산포도

- 0 ~ 2π 범위 sine 값의 크기 변화를 산포도로 표시

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

plt.title("Trigonometric functions")
plt.xlabel("angle")
plt.ylabel("value")
plt.grid(True, color='darkcyan', alpha=0.2,
linestyle='-.')

x = np.arange(0, 360+1, 10)
sin_y = np.sin(x * np.pi / 180)
area = x * np.abs(sin_y) *
np.random.randint(10)
color = area
alpha = 0.5
plt.scatter(x, sin_y, s=area, c=color,
alpha=alpha, cmap='hsv')
plt.colorbar()
plt.xticks([0, 90, 180, 270], ['0d', '90d',
'180d', '270d'])

plt.show()
```



그 밖의 플롯

- 히스토그램은 값의 분포를 바 플롯으로 표현
  - 구간별 확률분포나 밀도를 비교하기 좋은 플롯
  - pyplot.hist(x,..., **kwargs)
    - x: 배열 형테의 데이터
- 파이는 각 값의 비율을 한눈에 비교하기 좋게 원 플롯으로 표현
  - 입력되는 배열 요소는 100개까지 표현 가능
  - pyplot.pie(x, explode=None, labels=None, colors=None,..., shadow=False,...)
    - x: 1차원 배열 형태로 파이 크기 데이터
    - explode: 배열 형태로 각 파이의 반경 비율 지정. 기본값은 None
    - labels: 배열 형태의 파이 레이블
    - colors: 배열 형태의 파이 색상
    - shadow: 불 타입의 그림자 표시 유무

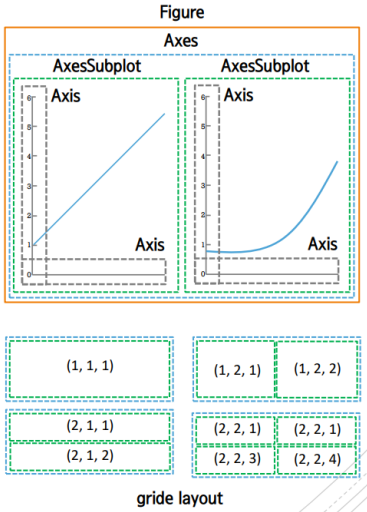
서브 플롯팅

- 여러 좌표축 비교
  - show()를 호출할 때마다 좌표축이 포함된 현재 피겨를 백엔드 레이어로 전달해 실제 이미지 출력
  - 피겨에 포함된 좌표축의 내용을 바꿔가며 show() 호출
    - 여러 좌표축 비교로는 적합하지 않음

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

x = np.linspace(0.1, 6.0, 1000)
y1 = np.sin(x)
y2 = np.exp(x)
y3 = np.log(x)
plt.plot(x, y1)
plt.show()
plt.plot(x, y2)
plt.show()
plt.plot(x, y3)
plt.show()
```

서브 플롯팅2

- Figure와 Axes 객체를 만들지 않고 플롯 함수를 호출하면 스크립팅 레이어는 사용자를 대신해 Figure 및 Axes 객체를 만듦
  - Figure: 모든 플롯 요소에 대한 최상위 컨테이너
    - 피겨 객체를 닫으면 담긴 모든 객체도 함께 제거됨
  - Axes: 데이터 공간에 대한 이미지 영역을 좌표축으로 관리하는 객체
    - 그리드 스타일로 AxesSubplot 배치
  - Axis: 그래프 한계를 설정하고 눈금과 눈금 레이블을 관리하는 숫자 라인과 같은 객체
    - 눈금 위치는 Locator 객체의해 결정되고 문자열은 포맷터에 의해 형식이 지정됨
- 피겨, 좌표축 객체 참조
  - pyplot.figure() 또는 pyplot.subplots()로 만든 피겨는 지속적으로 참조 유지
    - pyplot.gcf(): 현재 피겨 객체의 참조 반환
    - pyplot.gca(): 현재 좌표축 객체의 참조 반환
  - pyplot.close('all'): 모든 피겨 닫기



서브 플롯팅3

- 피겨에는 원하는 만큼 하위 좌표축을 추가할 수 있으며, 그리드 형태의 레이아웃을 가짐
  - Figure.add_subplot(*args, **kwargs): 지정한 피겨에 좌표축을 추가한 후 Axes의 하위 객체인 AxesSubplot 반환
    - 피겨 객체는 figure.Figure() 또는 pyplot.figure()로 만듦
    - *args: 좌표축. 기본값은 (1, 1, 1): 기존 좌표축과 다르면 새로 만들고, 같으면 검색
      - 세 개의 정수 (row, column, index): row x column 그리드에서 index로 위치 선택. index는 row 우선
      - 3자리 정수: 서브 플롯이 9개 이하일 때 rci (r=row, c=column, i=index)
    - projection: 서브플롯의 투영 유형으로 'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', 'rectilinear' 중 하나. 기본값은 'rectilinear'
    - sharex, sharey: x 또는 y 축을 sharex 또는 sharey와 공유
    - constrained_layout: 불 타입으로 True이면 자동으로 좌표축 사이 간격 조정
      - 세부 설정은 [Figure | pyplot].subplots_adjust(left=None, bottom=None, right=None, top=None, wpsace=None, hspace=None) 메소드나 함수 사용
        - wspace, hspace는실수 타입으로 좌표축 사이 패딩 폭과 높이. top, tottom, left, right은 실수 타입으로 좌표축의 위, 아래, 왼쪽, 오른쪽 가장자리 위치
  - pyplot.subplot(*args, **kwargs): 현재 피겨에서 좌표축을 추가하거나 검색한 후 현재 좌표축으로 지정 및 반환
    - 인자는 add_subplot()과 같음
  - pyplot.subplots(*args, **kwargs): 새 피겨를 만든 후 Axes 객체를 그리드로 채움
    - *args는 행과 열 개수이고, 피겨와 ndarray 타입 AxesSubplot 객체 시퀀스를 튜플로 반환
  - add_subplot(), subplot(), subplots() 비교
    - subplot()은 현재 좌표축 기준
    - subplots()와 add_subplot()는 반환된 좌표축 기준
-

서브 플롯팅 방법을 동일한 사례로 비교해 보라

- 0.1 ~ 6.0 구간 100 개의 x 값에 대해 sin_y, exp_y, log_y를 개산한 후 show(), subplots(), add_subplots()로 라인 플롯팅

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

x = np.linspace(0.1, 6.0, 1000)
y1 = np.sin(x) ; y2 =
np.exp(x) ; y3 = np.log(x)
plt.subplot(221)
plt.plot(x, y1)
plt.subplot(222)
plt.plot(x, y2)
plt.subplot(223)
plt.plot(x, y3)
plt.show()
```

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

x = np.linspace(0.1, 6.0, 1000)
y1 = np.sin(x) ; y2 =
np.exp(x) ; y3 = np.log(x)
fig, ((ax1, ax2), (ax3, _)) =
plt.subplots(2, 2)
ax1.plot(x, y1)
ax2.plot(x, y2)
ax3.plot(x, y3)
plt.show()
```

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

x = np.linspace(0.1, 6.0, 1000)
y1 = np.sin(x) ; y2 = np.exp(x) ;
y3 = np.log(x)
fig = plt.figure()
ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)
ax1.plot(x, y1)
ax2.plot(x, y2)
ax3.plot(x, y3)
plt.show()
```