Lecture 9. Matplotlib을 활용한 그래프 구성

기초 데이터 분석

Matplotlib 그래프 구성

- Grid 설정
- figure title 설정
- x축, y축 Label 설정
- x축, y축 범위 설정
- Tick 설정
- 여러개의 차트 그리기
- Legend
- Subplot
- etc…

복습 : 실습용 그래프 만들기

- **Line Plot , Scatter Plot 을 활용해서 실습용 차트를 만들어보자**
  1. 사인 곡선을 갖는 그래프
     - x 축 범위 : 0~6
     - 선 모양 : 대시
     - 선 색상 : Red

  2. **Scatter Plot으로 iris 데이터의 petal_length와 petal_width에 대해 나타내자**
     - 점 모양 : '.'

복습 : 실습용 그래프 만들기

- **Line Plot , Scatter Plot 을 활용해서 실습용 차트를 만들어보자**
  1. 사인 곡선을 갖는 그래프
     - x 축 범위 : 0~6
     - 선 모양 : 대시
     - 선 색상 : Red

In [3]:
```
x = np.arange(0,6,0.01) # x축 범위 지정
y = np.sin(x) # 사인 함수 y 지정
lines = plt.plot(x, y,'--',c='r')
```

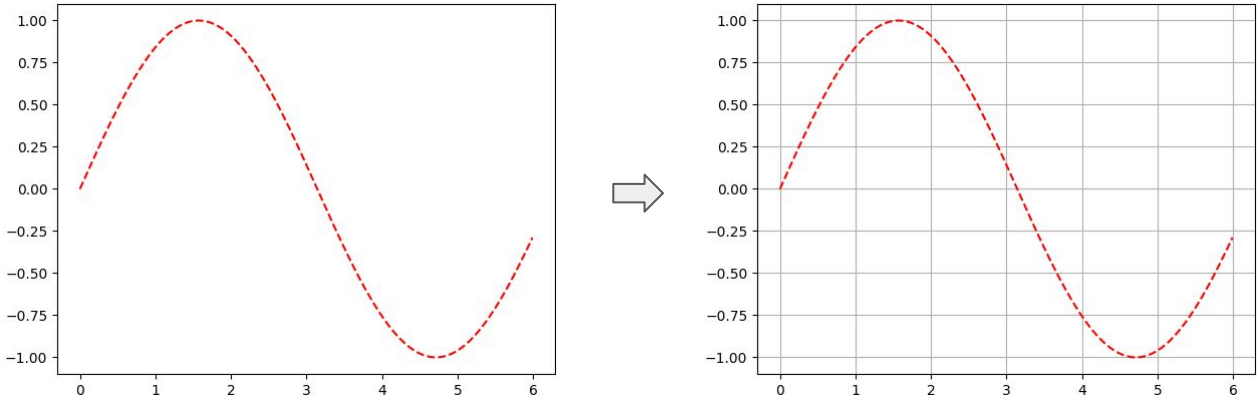
  2. **Scatter Plot으로 iris 데이터의 petal_length와 petal_width에 대해 나타내자**
     - 점 모양 : '.'

In [3]:
```
x = iris.petal_width
y = iris.petal_length
plt.scatter(x,y,marker='.') # marker = '.'
```

눈금선(Grid) 설정

- 보다 정확한 수치 전달을 위해 눈금선 활성화 가능
- **plt.grid()** 함수의 인자로 True/False 를 입력하여 활성화

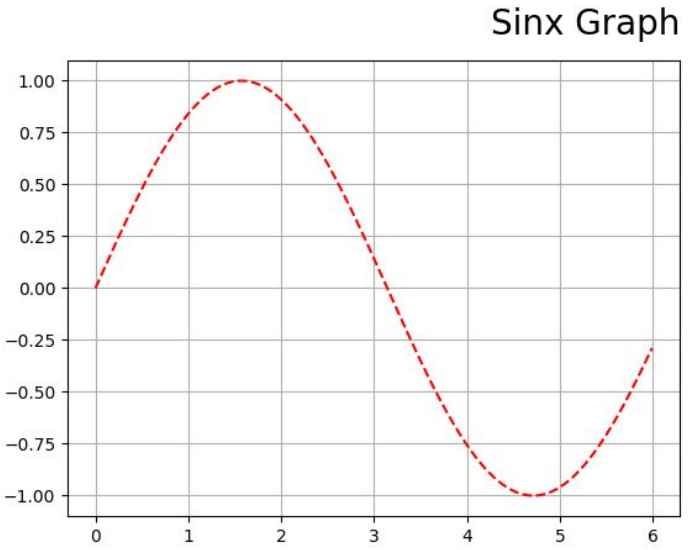
In [3]:
```
x = np.arange(0,6,0.01) # x축 범위 지정
y = np.sin(x) # 사인 함수 y 지정
lines = plt.plot(x, y,'--',c='r')
plt.grid(True) # 그리드 활성화
```



Title 설정

- **Figure**가 전달하고 싶은 내용을 **Title** 로 지정
- **plt.title(text)** 함수를 사용
- **loc :** 타이틀의 위치 설정 [center(default), right, left]
- **pad :** 타이틀과 Figure 사이 조정
- **fontsize** : 타이틀의 크기 설정

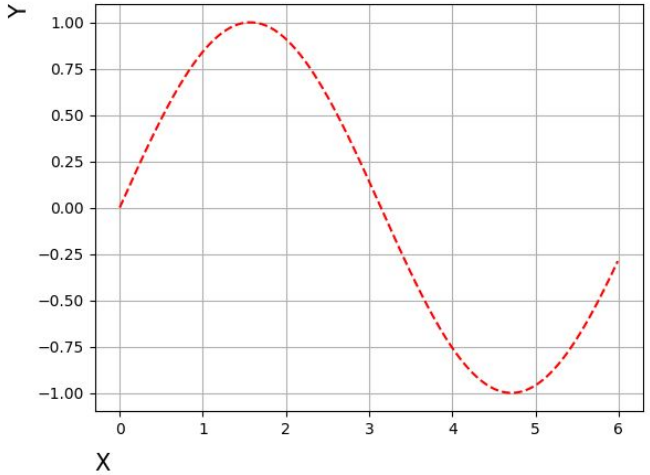
In [3]:
```
x = np.arange(0,6,0.01) # X축 범위 지정
y = np.sin(x) # 사인 함수 Y 지정
lines = plt.plot(x, y,'--',c='r')
plt.grid(True) # 그리드 활성화
plt.title('Sinx Graph', loc='right',pad=15) # Title 설정
```



X,Y 축 Label 설정

- **x축과 y축의 이름도 Title과 같이 추가**
- **plt.xlabel() 과 plt.ylabel()** 함수 활용
- 설정 가능한 인자들은 Title과 유사
  - **loc** : 라벨의 위치 지정
  - **labelpad** : figure와의 거리 설정
  - **fontdict** : font의 여러 설정값을 포함한 dict 추가

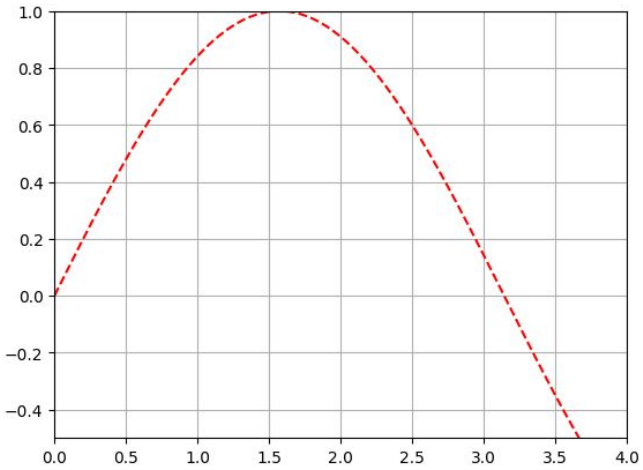
In [3]:
```
x = np.arange(0,6,0.01) # x축 범위 지정
y = np.sin(x) # 사인 함수 y 지정
lines = plt.plot(x, y,'--',c='r')
plt.grid(True) # 그리드 활성화
plt.xlabel('X',loc='left',labelpad=10, fontdict={'size':15}) #
x Label 지정
plt.ylabel('Y',loc='top',fontdict={'size':15}) # y축 Label 지정
```



X,Y 축 범위 지정

- x축과 y축의 범위는 자동으로 설정되지만
- **xlim**() 과 **ylim**() 함수를 통해 **임의로 지정이 가능**
- **axis**() 함수를 통해 x,y 축의 범위 정보를 얻을 수 있음
- **각 xlim, ylim 함수에 원하는 함수 범위를 지정**

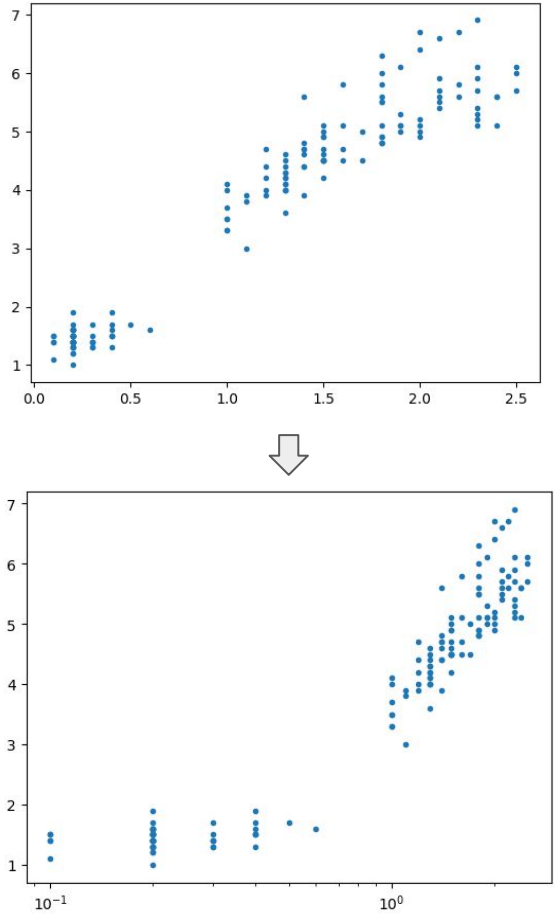
In [3]:
```
x = np.arange(0,6,0.01) # x축 범위 지정
y = np.sin(x) # 사인 함수 y 지정
lines = plt.plot(x, y,'--',c='r')
plt.grid(True) # 그리드 활성화
plt.xlim([0,4]) # 0~4 x축 제한
plt.ylim([-0.5,1]) # 0~4 y축 제한
```



X,Y 축 스케일 지정

- **y축이나 x축의 범위가 너무 커 표현이 어려운 경우**
- Linear Scale에서 Log Scale 등으로 변형 가능
- **plt.xscale() 과 plt.yscale()** 함수 사용
- **input : ‘linear’(default), ‘log’, ‘symlog’, ‘logit’**

In [3]:
```
x = iris.petal_width
y = iris.petal_length
plt.scatter(x,y,marker='.') # marker = '.'
plt.xscale('log') # 로그스케일로 변환
```



X,Y 축 스케일 지정

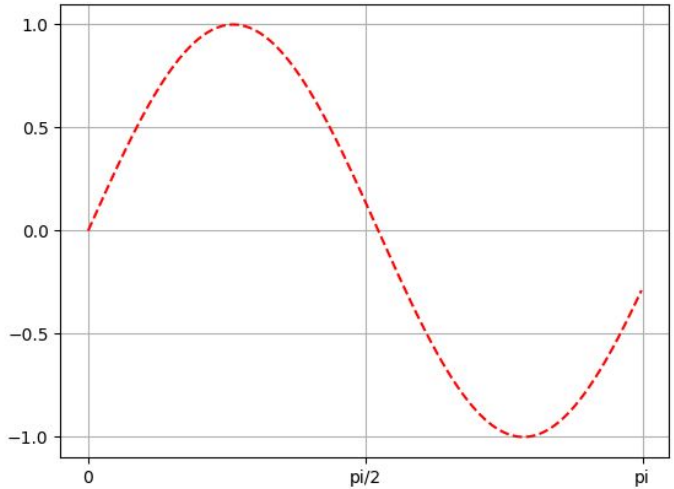
- **y축이나 x축의 범위가 너무 커 표현이 어려운 경우**
- Linear Scale에서 Log Scale 등으로 변형 가능
- **plt.xscale() 과 plt.yscale()** 함수 사용
- **input : ‘linear’(default), ‘log’, ‘symlog’, ‘logit’**

In [3]:
```
x = iris.petal_width
y = iris.petal_length
plt.scatter(x,y,marker='.') # marker = '.'
plt.xscale('log') # 로그스케일로 변환
```

축 눈금 조정하기

- Figure에서 눈금은 틱(Tick) 단위로 표시됨
- Tick 의 범위나 빈도, 등을 xticks(), yticks() 로 설정
- xticks(), yticks()에 원하는 tick 위치를 지정 : array
- 그 외에도 labels 인자에 각 tick마다 표현할 label 설정 : array

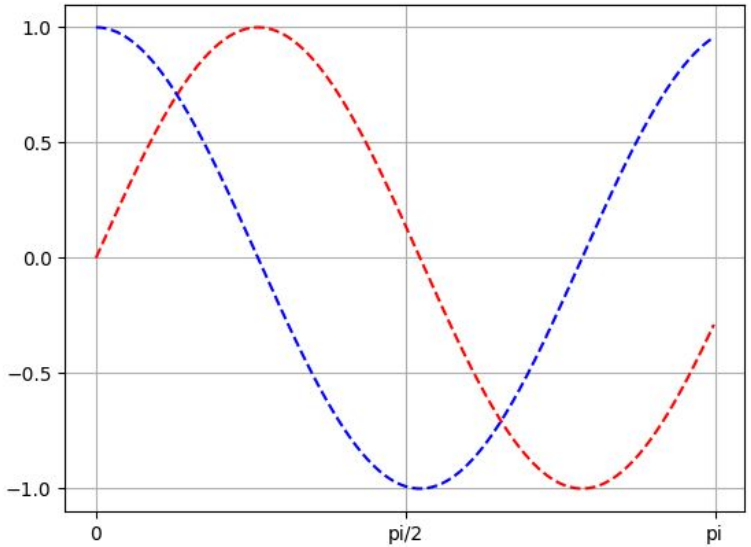
In [3]:
```
x = np.arange(0,6,0.01) # x축 범위 지정
y = np.sin(x) # 사인 함수 y 지정
lines = plt.plot(x, y,'--',c='r')
plt.grid(True) # 그리드 활성화
plt.xticks([0,3,6],labels = ['0','pi/2','pi'])
# x축 tick 설정
plt.yticks([-1,-0.5,0,0.5,1]) # y축 tick 설정
```



One Figure & Multiple Plot

- **plot() 함수를 여러번 호출하여 하나의 Figure에 여러개의 Plot 추가**
- 동일한 형태의 plot이 아니여도 가능
- 하지만 모든 plot의 x 축은 같은 길이를 가져야 함

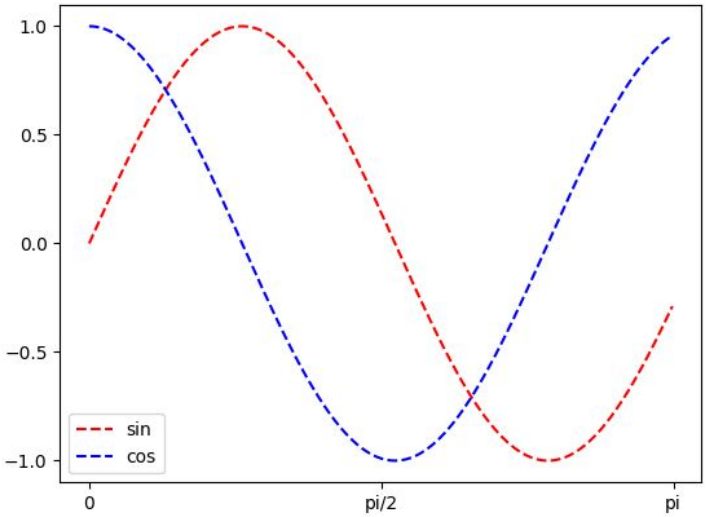
In [3]:
```
x = np.arange(0,6,0.01) # X축 범위 지정
y = np.sin(x) # 사인 함수 Y 지정
y2 = np.cos(x) # 사인 함수 Y 지정
plt.plot(x, y,'--',c='r')
plt.plot(x, y2,'--',c='b')
plt.grid(True) # 그리드 활성화
plt.xticks([0,3,6],labels = ['0','pi/2','pi'])
# x축 tick 설정
plt.yticks([-1,-0.5,0,0.5,1]) # y축 tick 설정
```



범례(Legend) 설정

- 여러개의 Plot이 있는 경우 구분을 위해 주로 사용
- Plot 함수 호출 시에, label을 사전에 지정
- plt.legend() 함수를 사용해 figure에 legend를 표시

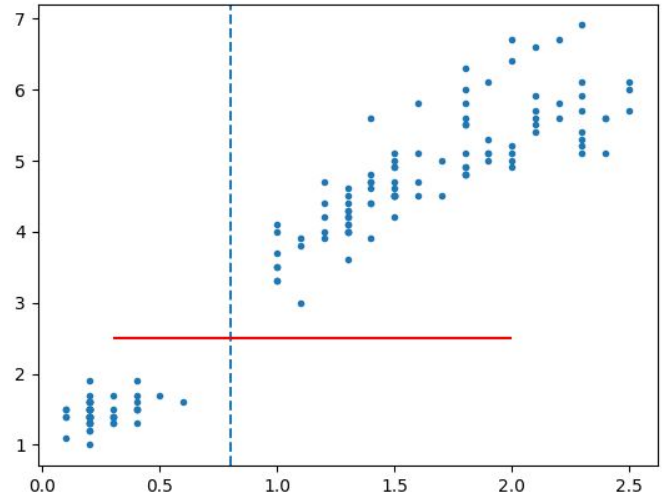
In [3]:
```
x = np.arange(0,6,0.01) # X축 범위 지정
y = np.sin(x) # 사인 함수 Y 지정
y2 = np.cos(x) # 사인 함수 Y 지정
plt.plot(x, y,'--',c='r',label='sin') # Label 설정
plt.plot(x, y2,'--',c='b',label='cos') # Label 설정
plt.legend() # 범례 표시

plt.xticks([0,3,6],labels = ['0','pi/2','pi'])
# x축 tick 설정
plt.yticks([-1,-0.5,0,0.5,1]) # y축 tick 설정
```



수평/수직선 추가하기

- 특정 위치에 수직 or 수평선을 표시하고 싶은 경우
- 점선 : axhline(), axvline(), hlines(), vlines()
- 각 함수들은 인자로 [x, ymin,ymax] 또는 [y,xmin,ymax] 세 가지를 받음
- color, linestyle 등 그 외 인자들은 line plot과 유사

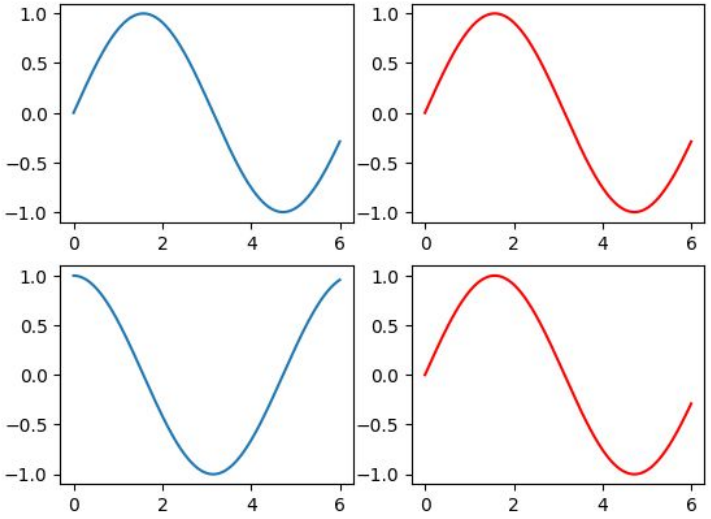
In [3]:
```
x = iris.petal_width
y = iris.petal_length
plt.scatter(x,y,marker='.') # marker = '.'
plt.axvline(0.8, linestyle='--') # 수직 점선
plt.hlines(2.5,0.3,2.0, color='red') # 수평선
```



Subplots 사용하기

- **plt.subplots()** 을 사용하여 각 **plot**들을 객체 단위로 관리 가능
- 특히, 여러 **figure**를 한번에 표현할 경우 **nrows, ncols**을 인자로 추가
- 하나의 figure에 **nrows * ncols** 개의 **plot**을 추가할 수 있다
- **figure**와 **nrows*ncols** 개의 **plot** 위치를 반환
- 해당하는 **plot** 위치에 plot 함수를 적용

In [3]:
```
fig, ax = plt.subplots(2,2) #총 2*2=4개의 subplot 생성
x = np.arange(0,6,0.01) # X축 범위 지정
y = np.sin(x) # 사인 함수 Y 지정
y2 = np.cos(x) # 사인 함수 Y 지정
ax[0][0].plot(x,y) # 좌측상단
ax[0][1].plot(x,y,c='r') #우측상단

ax[1][0].plot(x,y2) #좌측하단
ax[1][1].plot(x,y,c='r') #우측하단
plt.show()
```



실제 연습!

- 아래 그림과 유사한 모양의 **Figure**를 만들어봅시다
- 값은 정확히 일치하지 않아도 됨
- 색상, 그래프 스타일, **subplot**, 축 등을 설정해봅시다