서울시립대학교 토목공학과

파이썬의 기본 문법부터 딥러닝 맛보기까지

박상수 박사과정

2020년 8월 26일

강의 목차

- **Numpy 사용하기**
  - 과학 계산을 위한 Numpy 기초와 사용법
- **Matplotlib 차트/플롯 그리기**
  - 다양한 차트와 그래프를 쉽게 사용하는 방법
- **Pandas: 데이터 분석을 위한 기초**
  - 다양한 형태의 데이터를 다루는 방법

Numpy 배열: 배열의 생성

과학 계산과 다차원 배열 계산을 위한 라이브러리

Numpy 배열 생성

- 배열의 차원 (rank) 라 하며, 각 차원의 크기를 튜플로 표현 (shape)
- Numpy 배열을 생성하기 위해서는 리스트 또는. numpy 함수를 사용
  - 첫번째 numpy 배열: list1 (4개의 요소를 갖는 리스트)+ array 함수: rank (1), shape (4,)
  - 두번째 numpy 배열: 2*3 배열, shape (2,3)

```
import numpy as np

// 첫번째 생성 방법 (List + Array () 함수)
list1 = [1, 2, 3, 4]
a = np.array(list1)
print(a.shape) # (4, )

// 두번째 생성 방법 (Array () 함수)
b = np.array([[1,2,3],[4,5,6]])
print(b.shape) # (2, 3)
print(b[0,0]) # 1
```

Numpy 배열: 배열 초기화

Numpy 배열 초기화

- numpy에서 제공하는 함수를 사용, numpy 배열의 값을 특정 값으로 초기화
  - zeros()/ones(): 해당 배열을 모두 0/1로 초기화
  - full(): 배열에 사용자가 지정한 값으로 초기화
  - eye(): 대각선으로는 1이고 나머지는 0인 2차원 배열 생성
  - 숫자를 생성하는 range(n) 함수와 배열을 다차원으로 변형하는 reshape()

```
import numpy as np
a = np.zeros((2,2))
print(a)
# 출력:
# [[ 0. 0.]
# [ 0. 0.]]

a = np.ones((2,3))
print(a)
# 출력:
# [[ 1. 1. 1.]
# [ 1. 1. 1.]]

a = np.full((2,3), 5)
print(a)
# 출력:
# [[5 5 5]
# [5 5 5]]
```

```
a = np.eye(3)
print(a)
# 출력:
# [[ 1. 0. 0.]
# [ 0. 1. 0.]
# [ 0. 0. 1.]]

a = np.array(range(20)).reshape((4,5))
print(a)
# 출력:
# [[ 0 1 2 3 4]
# [ 5 6 7 8 9]
# [10 11 12 13 14]
# [15 16 17 18 19]]
```

Numpy 배열: 슬라이싱

슬라이스 (Slice)

- Numpy 배열을 슬라이싱 하기 위해서는 각 차원 별로 슬라이싱 범위를 지정

```
import numpy as np
lst = [
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]
]
arr = np.array(lst)
# 슬라이스
a = arr[0:2, 0:2]
print(a)

# 출력:
# [[1 2]
# [4 5]]
```

```
a = arr[1:, 1:]
print(a)
# 출력:
# [[5 6]
# [8 9]]
```

Numpy 배열: 인덱싱

정수 인덱싱 (Integer indexing)

- 각 차원 별로 선택되어지는 배열 요소의 인덱스들을 일렬로 나열하여 부분 집합을 구하는 방식
  - 슬라이싱은 각 배열 차원 별 최소-최대의 범위를 정하여 부분 집합을 구하는 방법
  - `a[[row1, row2], [col1, col2]]`는 `a[row1, col1], a[row2, col2]` 라는 두개의 배열 요소의 집합을 의미

```
import numpy as np
lst = [
[1, 2, 3, 4],
[5, 6, 7, 8],
[9, 10, 11, 12]
]
a = np.array(lst)
# 정수 인덱싱
s = a[[0, 2], [1, 3]]

print(s)
# 출력
# [2 12]
```

Numpy 연산: 사칙연산

- **Numpy를 사용하면 배열 간 연산을 쉽게 가능**
  - 행렬의 곱은 연산: +, -, *, / 등의 사칙연산과 add(), subtract(), multiply(), divide() 등의 함수 사용 가능
    - a+b를 하면 a[0]+b[0], a[1]+b[1], …과 같은 방식으로 계산 수행

```
import numpy as np
a = np.array([1,2,3])
b = np.array([4,5,6])

# 각 요소 더하기
c = a + b
# c = np.add(a, b)
print(c) # [5 7 9]

# 각 요소 빼기
c = a - b
# c = np.subtract(a, b)
print(c) # [-3 -3 -3]

# 각 요소 곱하기
# c = a * b
c = np.multiply(a, b)
print(c) # [4 10 18]

# 각 요소 나누기
# c = a / b
c = np.divide(a, b)
print(c) # [0.25 0.4 0.5]
```

Numpy 연산: 벡터와 행렬의 연산

Numpy를 사용하면 배열 간 연산을 쉽게 가능

- Vector와 Matrix의 Product를 구하기 위해서는 dot() 함수를 사용
- 각 배열의 요소들을 더하는 sum() 함수, 각 배열 요소들을 곱하는 prod() 함수
  - Axis 옵션: axis의 값이 1이면 행간의 연산, 0이면 열 간의 연산

```
import numpy as np
lst1 = [[1,2],[3,4]]
lst2 = [[5,6],[7,8]]

a = np.array(lst1)
b = np.array(lst2)
c = np.dot(a, b)
print(c)
# 출력:
# [[19 22]
# [43 50]]
```

```
import numpy as np
a = np.array([[1,2],[3,4]])
s = np.sum(a)
print(s) # 10

# axis=0 이면, 열끼리 더함
# axis=1 이면, 행끼리 더함
s = np.sum(a, axis=0)
print(s) # [4 6]
s = np.sum(a, axis=1)
print(s) # [3 7]
s = np.prod(a)
print(s) # 24
```

Numpy 실습문제

- **아래의 행렬을 Numpy로 만들고 Numpy API를 사용하여 계산합시오**
  - np.array() 함수를 활용하여 numpy 배열 생성
  - np.dot() 함수를 사용하여 계산하도록 프로그램을 만드세요

1	3	7
1	0	0
2	9	10

*

1	2	3
6	5	4
7	8	9

= ?

1	3	7
1	0	0
2	9	10

+

1	2	3
6	5	4
7	8	9

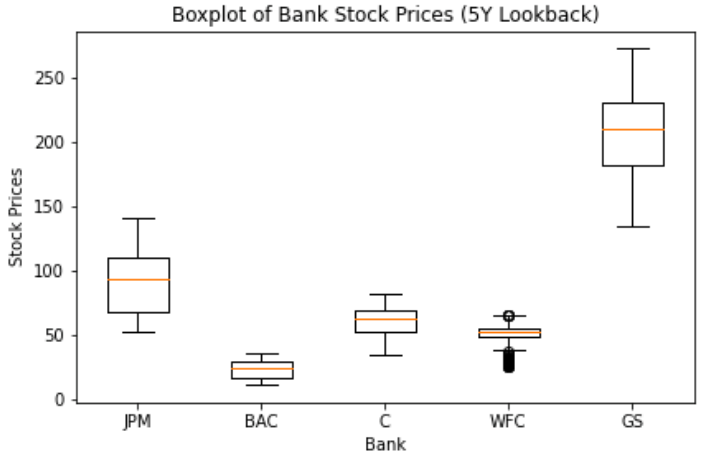
= ?

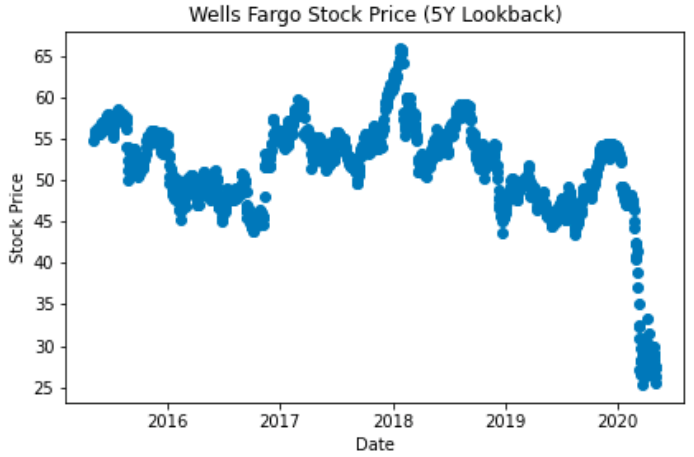
강의 목차

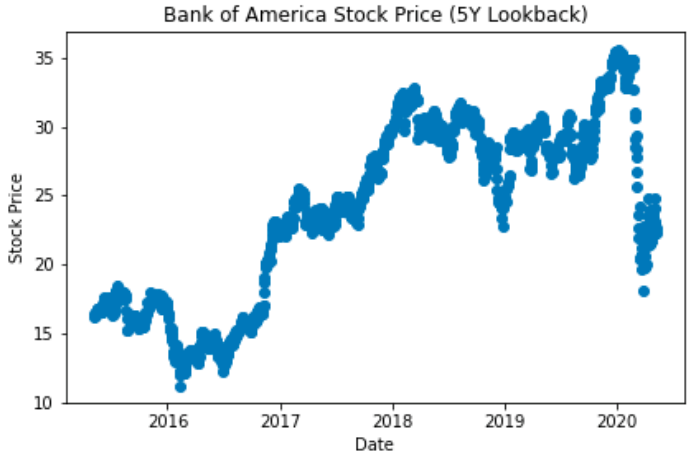
- **Numpy 사용하기**
  - 과학 계산을 위한 Numpy 기초와 사용법
- **Matplotlib 차트/플롯 그리기**
  - 다양한 차트와 그래프를 쉽게 사용하는 방법
- **Pandas: 데이터 분석을 위한 기초**
  - 다양한 형태의 데이터를 다루는 방법

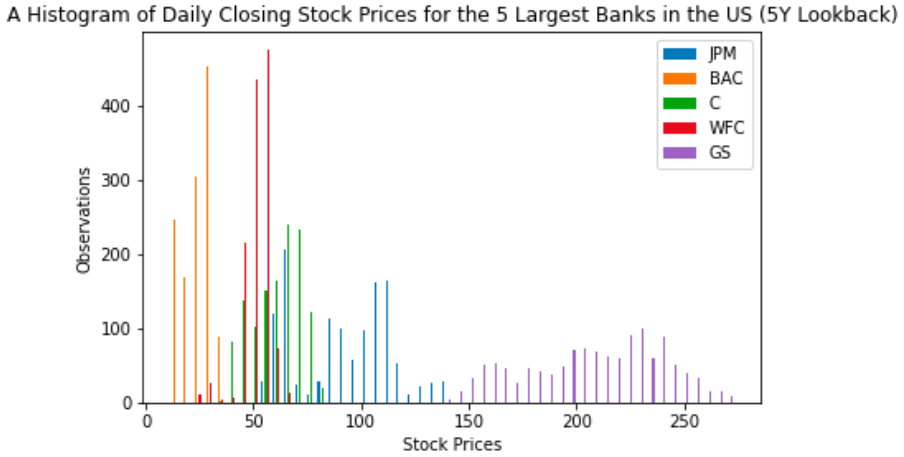
Matplotlib 개요

- **데이터를 차트나 플롯 (Plpt)으로 그려주는 데이터 시각화 라이브러리 패키지**
  - 라인 플롯, 바 차트, 파이 차트, 히스토그램 등 다양한 형태의 시각화 기능 가능







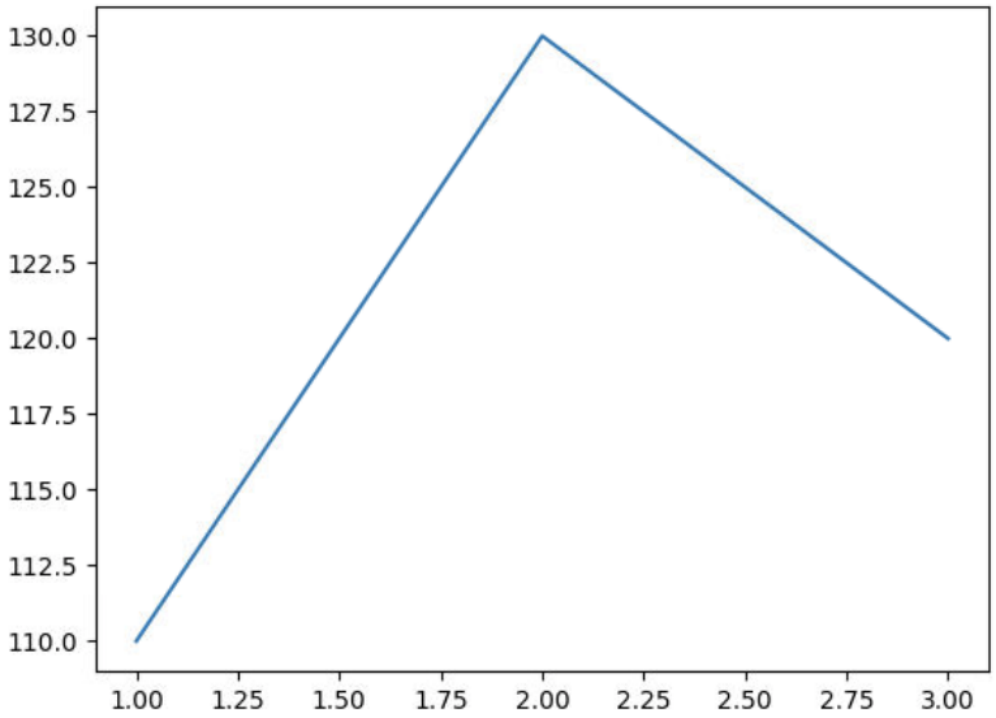


Matplotlib 사용법 #1

- **Matplotlib를 사용하기 위해서는 패키지의 import 필요**
  - Matplotlib.pyplot를 import
  - Plt.plot()은 라인 플롯을 그리는 함수
    - X축 (1,2,3) 값과 Y축 (110, 130, 120) 값을 사용하여 라인 플롯을 그리는 함수
  - Plt.show()를 호출하여 실제 그린 그림을 출력

```
from matplotlib import pyplot as plt

plt.plot([1,2,3], [110,130,120])
plt.show()
```

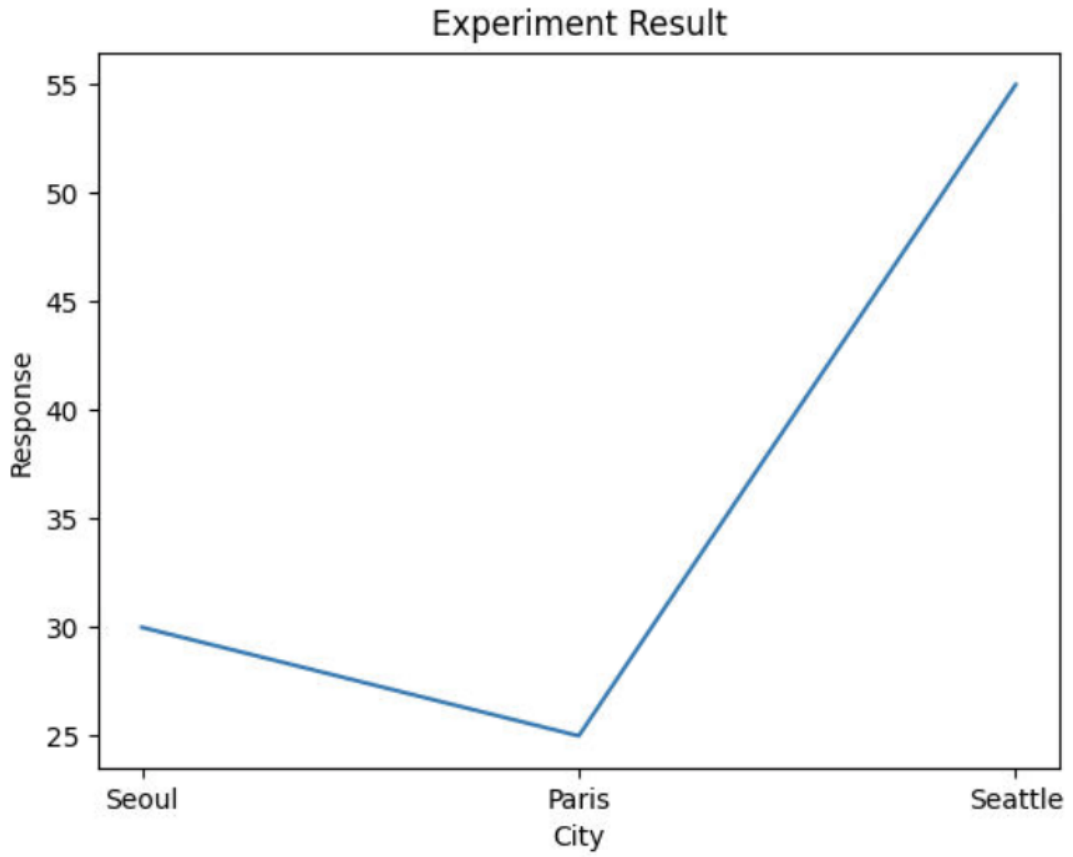


Matplotlib 사용법 #2

제목과 축 레이블

- 플롯에 X,Y축 레이블이나 제목을 붙이기 위해서는
  - `plt.xlabel/ylabel` (축이름), `plt.title` (제목)

```
from matplotlib import pyplot as plt
plt.plot(["Seoul","Paris","Seattle"], [30,25,55])
plt.xlabel('City')
plt.ylabel('Response')
plt.title('Experiment Result')
plt.show()
```

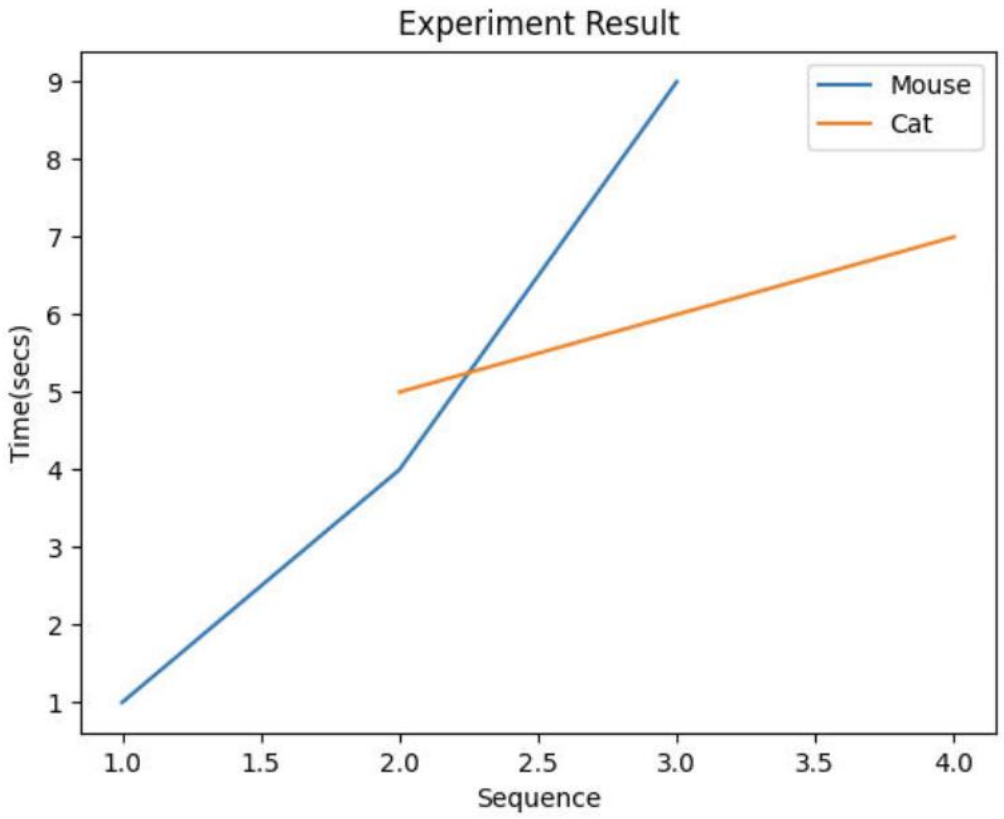


Matplotlib 사용법 #3

범례 추가

- 플롯에 여러 개의 라인들을 추가하기 위해서는
  - plt.plot()을 plt.show() 이전에 여러 번 사용
- 각 라인에 대한 범례를 추가하기 위해서는
  - plt.legend([라인1범례, 라인2범례]) 함수를 사용

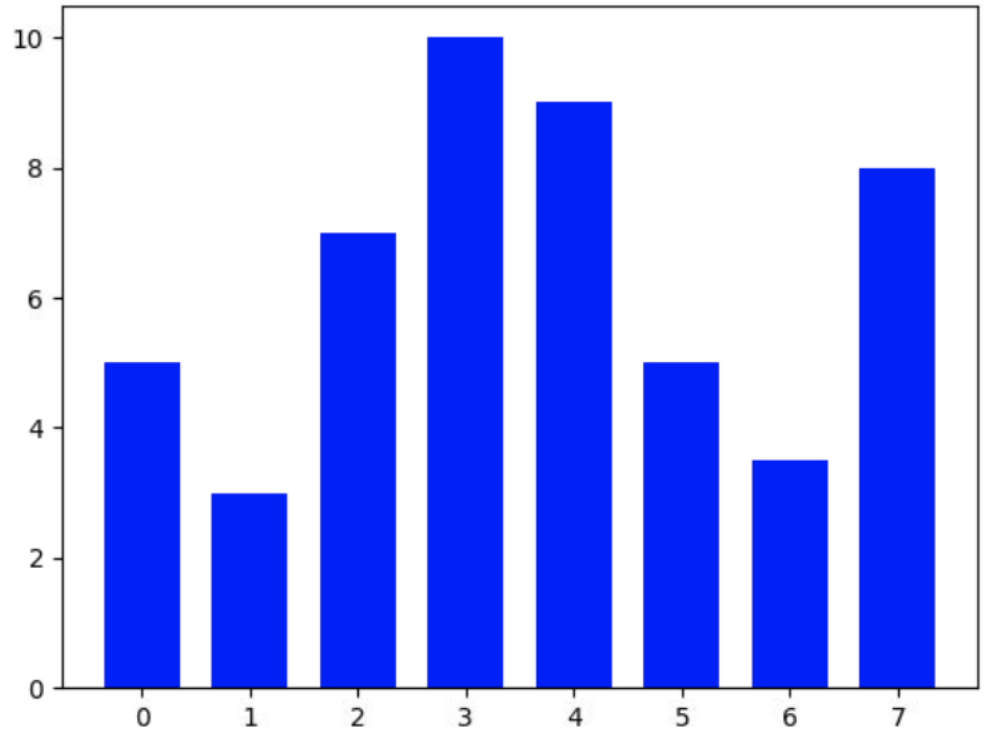
```
from matplotlib import pyplot as plt
plt.plot([1,2,3], [1,4,9])
plt.plot([2,3,4],[5,6,7])
plt.xlabel('Sequence')
plt.ylabel('Time(secs)')
plt.title('Experiment Result')
plt.legend(['Mouse', 'Cat'])
plt.show()
```



다양한 차트 및 플롯

- **Matplotlib에서는 다양한 차트 및 플롯 지원**
  - 각 차트/플롯마다 다른 함수를 사용
    - Bar 차트 (plt.bar() 함수), Pie 차트 (plt.pie() 함수)
    - 히스토그램 (plt.hist() 함수)

```
from matplotlib import pyplot as plt
y = [5, 3, 7, 10, 9, 5, 3.5, 8]
x = range(len(y))
plt.bar(x, y, width=0.7, color="blue")
plt.show()
```



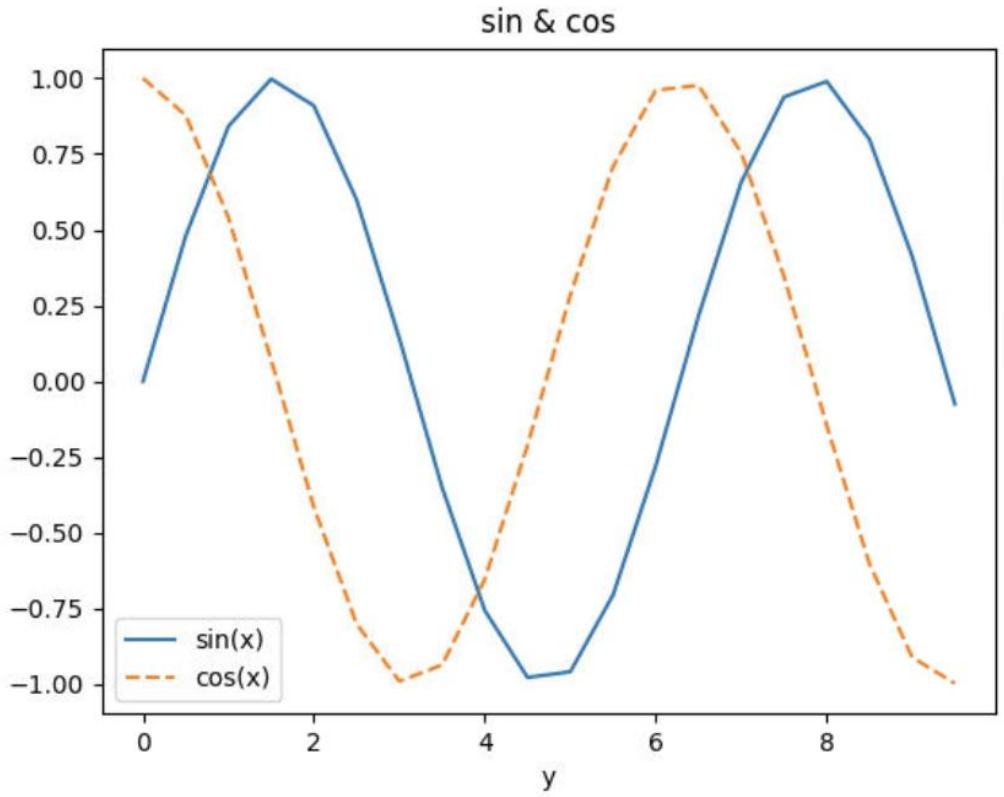
Matplotlib 연습문제

하나의 차트에 여러 개의 그래프를 그리는 예제

- x축과 y축의 `label`과 `title`이 잘 출력되도록 수정하세요

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

x = np.arange(0, 10, 0.5) # x축 (0~10, 0.5씩 증가)
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x, y1, label="___")
plt.plot(x, y2, label="___", linestyle="--")
plt.xlabel("_")
plt.xlabel("_")
plt.title("___")
plt.legend() # 범례
plt.show()
```



강의 목차

- **Numpy 사용하기**
  - 과학 계산을 위한 Numpy 기초와 사용법
- **Matplotlib 차트/플롯 그리기**
  - 다양한 차트와 그래프를 쉽게 사용하는 방법
- **Pandas: 데이터 분석을 위한 기초**
  - 다양한 형태의 데이터를 다루는 방법

Pandas 개요

데이터 분석을 위하 널리 사용되는 라이브러리 패키지

- `import pandas as pd`
- Pandas에서는 세 가지의 자료구조 지원
  - 1차원 (Series), 2차원 (DataFrame), 3차원 (Panel)

```
1  import pandas as pd
2
3  d0 = pd.read_csv('./datafile/E0.csv')
4  display(d0)
5  print(len(d0))
```

	Div	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	...	BbAv<2.5	BbAH
0	E0	16/08/14	Arsenal	Crystal Palace	2.0	1.0	H	1.0	1.0	D	...	2.10	24.0
1	E0	16/08/14	Leicester	Everton	2.0	2.0	D	1.0	2.0	A	...	1.80	22.0
2	E0	16/08/14	Man United	Swansea	1.0	2.0	A	0.0	1.0	A	...	2.13	25.0
3	E0	16/08/14	QPR	Hull	0.0	1.0	A	0.0	0.0	D	...	1.58	24.0
4	E0	16/08/14	Stoke	Aston Villa	0.0	1.0	A	0.0	0.0	D	...	1.60	23.0
5	E0	16/08/14	West Brom	Sunderland	2.0	2.0	D	1.0	1.0	D	...	1.70	22.0

Pandas: 1차원 데이터 (Series)

가장 간단한 1차원 자료구조

- 배열, 리스트와 같은 일련의 Sequence 데이터를 처리하는데 사용
- 별도의 인덱스 레이블을 지정하지 않으면 자동적으로 0부터 시작되는 정수형 인덱스 사용

```
import pandas as pd

data = [1, 3, 5, 7, 9]

// Series 정의하기
s = pd.Series(data)
print(s)
```

```
0    1
1    3
2    5
3    7
4    9
dtype: int64
```

Pandas: 2차원 데이터 (Data Frame)

2차원 자료구조

- 행과 열이 있는 테이블 데이터 (Tabular Data)를 처리하는 데 사용
- 열 (column)의 dictionary의 key로, 행 (row)을 value로 사용
- Pd.DataFrame()을 사용하여 pandas의 Data Frame 자료구조로 변환

```
import pandas as pd

data = {
    'year': [2016, 2017, 2018],
    'GDP rate': [2.8, 3.1, 3.0],
    'GDP': ['1.637M', '1.73M', '1.83M']
}

df = pd.DataFrame(data)
print(df)
```

```
   year  GDP rate      GDP
0  2016        2.8  1.637M
1  2017        3.1   1.73M
2  2018        3.0   1.83M
```

Pandas: 데이터 액세스 #1

- **Series, Data Frame등의 자료구조를 만든 후, 데이터 사용**
  - Pandas 자료구조에 대해 인덱싱 혹은 속성 (Attribute)를 사용
    - 예) df['year'] 혹은 df.year 사용
    - df[df['year]>2016]을 사용하면 어떤 결과 ?
    - df.head(), df.tail()

```
import pandas as pd

data = {
    'year': [2016, 2017, 2018],
    'GDP rate': [2.8, 3.1, 3.0],
    'GDP': ['1.637M', '1.73M', '1.83M']
}

df = pd.DataFrame(data)
print(df['year'])
print(df.year)
```

```
0    2016
1    2017
2    2018
Name: year, dtype: int64
```

감사합니다