



Numpy와 Pandas

최석재

lingua@naver.com

Numpy 소개

Introducing the Numerical Python



- Numpy는 숫자 데이터를 효과적으로 처리해주는 패키지이다
- 파이썬의 list와 유사하지만 규모가 커질수록 저장 및 처리에 효율적이다
- 대부분의 데이터분석 패키지가 Numpy를 이용하고 있다
- Numpy 패키지는 아나콘다를 설치했다면 기본적으로 포함되어 있다
- 패키지는 모듈이 여러 개 결합된 것이다 (`_init_.py` 등 필수 구성요소 있음)
- 패키지의 사용 방법은 모듈의 경우와 같다

Numpy 불러오기

- 넘파이의 별명은 보통 np를 사용한다
- 기본 유형은 배열이다. 리스트와 유사하나 더 효율적이다

```
import numpy as np
```

```
x = [1, 2, 3, 4, 5]  
print("파이썬 리스트: ", x)
```

```
x2 = np.array([1, 2, 3, 4, 5])  
print("넘파이 배열: ", x2)
```

```
파이썬 리스트: [1, 2, 3, 4, 5]  
넘파이 배열: [1 2 3 4 5]
```

배열 인덱싱

- 특정 위치의 원소를 가져오는 인덱싱을 수행할 수 있다

```
import numpy as np
```

```
x1 = np.array([1, 2, 3, 4, 5])  
print("x1: ", x1)
```

```
print("x1[0]: ", x1[0])  
print("x1[3]: ", x1[3])
```

```
x1: [1 2 3 4 5]  
x1[0]: 1  
x1[3]: 4
```

배열 슬라이싱

- 전체 중 일부의 복수 데이터를 가져오는 슬라이싱도 가능하다

```
import numpy as np
```

```
x = np.arange(10)  
x2 = x[:5]  
x3 = x[5:]  
x4 = x[3:5]
```

```
print(x)  
print(x2)  
print(x3)  
print(x4)
```

*# 10개의 수를 만든다 (0부터 9까지, 10개)
처음 5개의 수를 선택한다 (0부터 4까지)
5번째 이후의 수를 선택한다
3번부터 4까지의 수를 선택한다*

```
[0 1 2 3 4 5 6 7 8 9]  
[0 1 2 3 4]  
[5 6 7 8 9]  
[3 4]
```

배열 산술연산

- 배열은 산술연산을 다음과 같은 방식으로 수행한다

```
import numpy as np
```

```
x = np.arange(5)
```

```
print(x)
```

```
print(np.add(x, 2))  
print(np.subtract(x, 2))  
print(np.multiply(x, 2))  
print(np.divide(x, 2))
```

```
print("=== 간략한 방법 ===")
```

```
print(x+2)  
print(x-2)  
print(x*2)  
print(x/2)
```

```
[2 3 4 5 6]  
[-2 -1  0  1  2]  
[0 2 4 6 8]  
[0.  0.5 1.  1.5 2. ]
```

계산 함수들

- 파이썬 기본함수와 이름과 사용법이 유사하지만 훨씬 빠른 함수들이 있다

```
import numpy as np
```

```
n = np.random.random(100)  
print(n)
```

```
print("기본 함수")  
print(sum(n))  
print(min(n))  
print(max(n))
```

```
print("\n넘파이 함수")  
print(np.sum(n))      # 합  
print(np.min(n))      # 최소값  
print(np.max(n))      # 최대값
```

그 외 유용한 계산 함수

```
print("\n그 외 유용한 계산 함수")
```

```
print(np.mean(n))      # 평균  
print(np.median(n))    # 중앙값  
print(np.std(n))        # 표준편차  
print(np.var(n))        # 분산
```

```
print(np.any(n))        # 참이 있는지 검사  
print(np.all(n))        # 모두가 참인지 검사 (0이 없는지)
```


연습문제 1-1

- presidents_heights.csv 파일을 이용하여 미국 대통령의 평균 신장을 구하시오

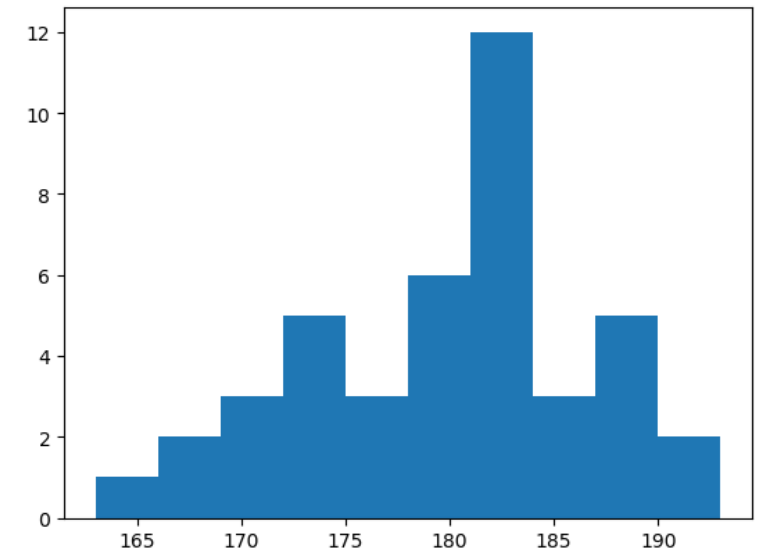
	order	name	height
0	1	George Washington	189
1	2	John Adams	170
2	3	Thomas Jefferson	189
3	4	James Madison	163
4	5	James Monroe	183
5	6	John Quincy Adams	171
6	7	Andrew Jackson	185
7	8	Martin Van Buren	168
8	9	William Henry Harrison	173
9	10	John Tyler	183
10	11	James K. Polk	173
11	12	Zachary Taylor	173
12	13	Millard Fillmore	175
13	14	Franklin Pierce	178
14	15	James Buchanan	183
15	16	Abraham Lincoln	193
16	17

연습문제 1-2

- 앞의 데이터를 이용하여 미국 대통령의 신장 분포를 그래프로 표현하시오

※ 답안^^

```
import matplotlib.pyplot as plt  
plt.hist(np.array(data['height']))  
plt.show()
```



Pandas 소개 *Introducing the Pandas*



- Pandas는 엑셀 시트와 유사한 자료구조를 이용한다
 - 형태가 엑셀 시트와 유사하기 때문에 엑셀로 저장된 데이터를 쉽게 처리할 수 있다
 - 행과 열에 관련된 유용한 함수를 제공하고 있기 때문에 간단한 조작으로 행과 열에 관한 계산을 수행할 수 있다
-
- Series와 DataFrame 두 개의 자료구조가 있다
 - 한 개의 열로 된 것은 Series로 담을 수 있으나,
 - 두 개 이상의 열로 된 것은 DataFrame로 담는다

Pandas 불러오기

- 판다스의 별명은 보통 pd를 사용한다
- Series는 파이썬의 리스트, 넘파이의 배열과 유사하다

```
import numpy as np
import pandas as pd
```

```
x = [1, 2, 3, 4, 5]
print("파이썬 리스트: ", x)
```

```
x2 = np.array([1, 2, 3, 4, 5])
print("넘파이 배열: ", x2)
```

```
x3 = pd.Series([1, 2, 3, 4, 5])
print("판다스 시리즈:")
print(x3)
```

파이썬 리스트: [1, 2, 3, 4, 5]

넘파이 배열: [1 2 3 4 5]

판다스 시리즈:

index	values
0	1
1	2
2	3
3	4
4	5

dtype: int64

Series 사용하기

- Series도 인덱싱과 슬라이싱을 지원한다

```
import pandas as pd
```

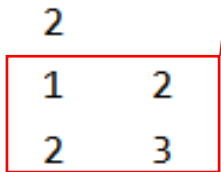
```
x3 = pd.Series([01, 12, 23, 34, 45])
```

```
print(x3[1])
```

```
print(x3[1:3])
```

인덱싱

슬라이싱



2	
1	2
2	3

dtype: int64

값만 출력하는 것도 가능하다

```
print(x3.values)
print(x3.values[1])
print(x3.values[1:3])
```

[1 2 3 4 5]

2

[2 3]

DataFrame 사용하기

- 열이 두 개 이상 있는 경우에는 DataFrame을 사용한다
- 사전형으로 만든 뒤, 데이터프레임으로 전환하여 만들 수 있다

```
import pandas as pd
```

```
data = {'도시': ['서울', '부산', '대구'], '인구': [9705, 3400, 2450]}  
data = pd.DataFrame(data)  
print(data)
```

	도시	인구
0	서울	9705
1	부산	3400
2	대구	2450

csv에서 불러들이기

- 그러나 DataFrame을 사용하는 상황은 보통 csv 파일을 읽을 때이다
- csv는 엑셀 파일과 유사하나 별도의 기능이 없는 단순한 형태이다

```
import pandas as pd
```

```
path = "./나의경로/"
```

```
data = pd.read_csv(path+'도시인구.csv', encoding="CP949") # CP949 또는 UTF8  
print(data)
```

	도시	인구
0	서울	9705
1	부산	3400
2	대구	2450
3	인천	2939
4	광주	1493

※ 만약 컬럼의 제목이 없다면, header=1을 사용한다

```
pd.read_csv(path+'도시인구.csv', encoding="CP949", header=1)
```

DataFrame 사용하기

- 미국 대통령 데이터를 불러들인다

```
import pandas as pd
```

```
path = "../나의경로/"
```

```
data = pd.read_csv(path+'presidents_heights.csv', encoding="UTF8")
```

```
print(data.head())
```

앞에서 5개 행을 보여준다

```
print(data.tail())
```

뒤에서 5개 행을 보여준다

```
print("=== 속성 ===")
```

```
print(data.columns)
```

컬럼의 이름

```
print(data.values)
```

각 행의 값

=== 속성 ===

Index(['order', 'name', 'height'], dtype='object')

[1 'George Washington' 189]

[2 'John Adams' 170]

[3 'Thomas Jefferson' 189]

[4 'James Madison' 163]

컬럼 선택

- 컬럼의 이름으로 선택하면 명확하다

```
import pandas as pd
```

```
path = "./나의경로/"
```

```
data = pd.read_csv(path+'presidents_heights.csv', encoding="UTF8")
```

```
print(data.columns)           # 컬럼의 이름 확인
```

```
print(data['name'])
```

```
print(data['height'])
```

```
Index(['order', 'name', 'height'], dtype='object')
```

0	George Washington
1	John Adams
2	Thomas Jefferson
3	James Madison
4	James Monroe
5	John Quincy Adams
6	Andrew Jackson
7	Martin Van Buren

```
Name: name, dtype: object
```

0	189
1	170
2	189
3	163
4	183
5	171
6	185

주의사항

- read_csv() 함수로 파일을 불러들인 경우에, 컬럼을 하나만 선택하면 그 결과는 Series이다
- DataFrame으로 받으려면 to_frame() 함수를 이용하거나 [['name']]와 같이 괄호를 두 번 사용한다

- `print(type(data['name']))` `<class 'pandas.core.series.Series'>`
- `print(type(data['name'].to_frame()))` `<class 'pandas.core.frame.DataFrame'>`

```
0    George Washington
1         John Adams
2    Thomas Jefferson
3       James Madison
4       James Monroe
Name: name, dtype: object
```

Series

```
      name
0  George Washington
1    John Adams
2  Thomas Jefferson
3    James Madison
4    James Monroe
```

DataFrame

복수 컬럼 선택

- 복수의 컬럼을 선택할 때는 내용을 리스트에 담는다

```
import pandas as pd
```

```
path = "./나의경로/"
```

```
data = pd.read_csv(path+'presidents_heights.csv', encoding="UTF8")
```

```
print(data.columns)
```

컬럼의 이름 확인

```
print(data[['order', 'name', 'height']])
```

선택할 열을 리스트로 담는다

```
Index(['order', 'name', 'height'], dtype='object')
```

	order	name	height
0	1	George Washington	189
1	2	John Adams	170
2	3	Thomas Jefferson	189
3	4	James Madison	163
4	5	James Monroe	183
5	6	John Quincy Adams	171

- 선택할 컬럼을 리스트로 먼저 만든 뒤, 전달하는 방법도 있다

```
cols = ['order', 'name', 'height']  
print(data[cols])
```

Series의 행 선택

- data[0], data.loc[0], data.iloc[0]
- 행 선택을 위한 위의 세 가지 방법이 어떻게 차이가 나는지 본다

```
import pandas as pd
```

```
data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5]) # 행의 이름에 해당하는 인덱스를 사용한다
print(data)
```

```
print("*"*10)
```

```
#print(data[0])
```

인덱스를 참조한다. 0 이라는 인덱스가 없으므로 에러가 발생한다

```
print(data[1])
```

인덱싱 때는 명시적인 인덱스를 사용한다

```
print(data[1:3])
```

But, 슬라이싱 때는 암묵적인 인덱스를 사용한다! & 1번째~2번째

1	a
3	b
5	c

dtype: object

a
3 b
5 c

dtype: object

.loc과 .iloc의 행 선택

0	1	a
1	3	b
2	5	c

dtype: object

암묵적 인덱스 명시적 인덱스

- .loc은 명시적인 인덱싱과 슬라이싱을 사용하므로 해당 인덱스가 없을 경우에 에러가 발생할 수 있다
- 그러나 .iloc은 암묵적인 인덱싱과 슬라이싱을 사용(행의 순서를 사용)하므로 에러가 발생할 가능성이 적다

```
print("=== loc ===")
print(data.loc[1])
print(data.loc[1:3])

print("\n=== iloc ===")
print(data.iloc[1])
print(data.iloc[1:3])
```

loc 속성은 명시적인 인덱싱과

명시적인 슬라이싱을 사용한다

iloc 속성은 암묵적인 인덱싱과

암묵적인 슬라이싱을 사용한다 & 1번째~2번째

=== loc ===	
1	a
3	b

dtype: object

=== iloc ===	
1	b
3	b
5	c

dtype: object

DataFrame의 행 선택

- DataFrame에서의 행 선택을 해본다

```
import pandas as pd
```

```
path = "../나의경로/"
```

```
data = pd.read_csv(path+'presidents_heights.csv', encoding="UTF8")
```

```
print(data.head())
```

```
print("=== 행 선택 ===")
```

```
print(data.loc[2])
```

```
print(data.loc[1:3])
```

```
print(data.iloc[2])
```

```
print(data.iloc[1:3])
```

	order	name	height
0	1	George Washington	189
1	2	John Adams	170
2	3	Thomas Jefferson	189
3	4	James Madison	163
4	5	James Monroe	183

```
order          3
name    Thomas Jefferson
height          189
Name: 2, dtype: object
```

```
   order  name    height
1      2  John Adams    170
2      3 Thomas Jefferson  189
3      4  James Madison    163
```

```
order          3
name    Thomas Jefferson
height          189
Name: 2, dtype: object
```

```
   order  name    height
1      2  John Adams    170
2      3 Thomas Jefferson  189
```

DataFrame의 열 선택

- DataFrame의 열을 선택하는 방법은 다음과 같다

- `data1 = data[['name', 'height']]`
- `data2 = data[data.columns[1:2]]`
- `data3 = data.loc[:, 'name':'height']`
- `data4 = data.iloc[:, 1:2]`

order		name	height
0	1	George Washington	189
1	2	John Adams	170
2	3	Thomas Jefferson	189
3	4	James Madison	163
4	5	James Monroe	183

※ 한 개 열만 선택하는 경우에는
`data.colname` 도 가능하다

```
print(data1.head())
```

	name	height
0	George Washington	189
1	John Adams	170
2	Thomas Jefferson	189
3	James Madison	163
4	James Monroe	183

```
print(data2.head())
```

	name
0	George Washington
1	John Adams
2	Thomas Jefferson
3	James Madison
4	James Monroe

```
print(data3.head())
```

	name	height
0	George Washington	189
1	John Adams	170
2	Thomas Jefferson	189
3	James Madison	163
4	James Monroe	183

```
print(data4.head())
```

	name
0	George Washington
1	John Adams
2	Thomas Jefferson
3	James Madison
4	James Monroe

DataFrame의 행과 열 선택

- DataFrame에서 행과 열을 동시에 선택하는 방법을 알아본다
- 기본적으로 콜론(:)과 쉼표(,)를 이용한다
- .loc 속성으로 컬럼을 선택할 때는 컬럼의 이름을 입력해야 한다

```
import pandas as pd
```

```
path = "./나의경로/"
```

```
data = pd.read_csv(path+'presidents_heights.csv', encoding="UTF8")
```

```
print(data.iloc[:, 1])
```

1번째 컬럼 선택

```
print("=== 1, 2 컬럼 선택 ===")
```

```
print(data.iloc[:, [1, 2]])
```

1번째와 2번째 컬럼 선택

DataFrame의 행과 열 선택

- 컬론의 앞쪽은 행, 뒤쪽은 열이 들어간다

```
import pandas as pd
```

```
path = "./나의경로/"
```

```
data = pd.read_csv(path+'presidents_heights.csv', encoding="UTF8")
```

```
print(data.iloc[0:3, :])          # 0~3 번째 행까지, 모든 컬럼 선택
```

```
print("=== 1번 컬럼 선택 ===")
```

```
print(data.iloc[0:3, 1])          # 0~3 번째 행까지, 1번 컬럼 선택
```

```
print("=== 1~2번 컬럼 선택 ===")
```

```
print(data.iloc[0:3, 1:3])        # 0~3 번째 행까지, 1~2번 컬럼 선택
```

조건적 선택

- 다음과 같이 조건을 두어 선택할 수 있다
- height가 185 이상인 행만 선택한다

```
import pandas as pd
```

```
path = "./나의경로/"
```

```
data = pd.read_csv(path+'presidents_heights.csv', encoding="UTF8")
```

```
print(data[data.height > 185])    # height 컬럼의 내용이 185를 넘는 것만
```

	order	name	height
0	1	George Washington	189
2	3	Thomas Jefferson	189
15	16	Abraham Lincoln	193
29	32	Franklin D. Roosevelt	188
33	36	Lyndon B. Johnson	193
38	41	George H. W. Bush	188
39	42	Bill Clinton	188

문자열 조건, 복합 조건

```
import pandas as pd
```

```
path = "../나의경로/"
```

```
data = pd.read_csv(path+'sales_result.csv', encoding="CP949")
```

```
# sales가 2000을 넘는 우수 실적 사원을 출력하세요  
print(data[data.sales>2000])
```

```
# 여자 사원을 출력하세요  
print(data[data.gender=="F"])
```

```
# sales가 2000 이상인 우수 여자 사원을 출력하세요  
print(data[(data.sales>2000) & (data.gender=="F")])
```

```
# sales가 2000 이상이거나 여자 사원을 출력하세요  
print(data[(data.sales>2000) | (data.gender=="F")])
```

	id	name	gender	sales
3	4	김선영	F	2200
6	7	최필선	M	2200

	id	name	gender	sales
0	1	김원경	F	1000
2	3	조해선	F	1500
3	4	김선영	F	2200
4	5	이화영	F	1700

	id	name	gender	sales
3	4	김선영	F	2200

	id	name	gender	sales
0	1	김원경	F	1000
2	3	조해선	F	1500
3	4	김선영	F	2200
4	5	이화영	F	1700
6	7	최필선	M	2200