

# 03. Numpy 와 Pandas

## 학습 목표

- Numpy 모듈을 활용해 직접 코딩 할 수 있다.
- Pandas 모듈을 활용해 직접 코딩 할 수 있다.
- Numpy,Pandas 모듈을 활용하여 데이터를 분석 할 수 있다.

## Numpy 모듈의 활용

### - Numpy 의 정의

- . 대규모, 다차원 배열을 쉽게 처리 할 수 있도록 도와주는 파이썬의 외부 모듈
- . 기본적으로 array 라는 자료형을 사용함
- . 행렬의 개념과 비슷함

### - 설치 방법

- . Anaconda 설치 시 함께 설치 됨
- . !pip install numpy 로도 설치 가능함

### - 호출방법

- . Import numpy as **np** 로 호출함
  - > 편의성을 위해 **np** 약칭 사용

```
import numpy as np
```

> numpy 모듈 사용 선언

```
a = [1,2,3,4,5]
```

```
b = np.array(a)
```

>파이썬 리스트 를 numpy 를 사용하여 배열화

```
print(b)
```

>numpy 배열 내용 출력

```
print(type(b))
```

>numpy 배열의 데이터 형식 확인

```
[1 2 3 4 5]
```

```
<class 'numpy.ndarray'>
```

### - numpy 모듈 의 특징

- . 파이썬의 리스트 자료형 과 아주 유사함
  - > 인덱싱 및 슬라이싱(자료추출)

```
import numpy as np
```

```
a = [1,2,3,4,5]
```

```
b = np.array(a)
```

```
print(type(a))
```

```
print(type(b))
```

```
<class 'list'>
```

```
<class 'numpy.ndarray'>
```

>리스트 와 numpy.array 데이터 형식은것을 확인

```
print(a[0])
```

```
print(b[0])
```

```
1
```

```
1
```

>index 를 통한 확인 가능

```
print(a[0:3])
```

```
print(b[0:3])
```

```
[1, 2, 3]
```

```
[1 2 3]
```

>범위 지정을 통한 데이터 추출 가능

## - 행렬 형태의 행렬 연산 지원

- . 리스트 : 연결, 반복 연산 만 지원
- . Numpy : 실제 행렬과 같이 행렬의 \*,+,- 등의 연산 을 제공
  - > 수학 계산에 용이함

```
import numpy as np
```

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

```
print(a+a)  
print(b+b)
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]  
[ 2  4  6  8 10]
```

> list 의연결

> array 의 합

> 리스트의 결과는 단순히 나열

> Numpy array 의 결과 는 두 데이터 index 값을 합산

```
import numpy as np
```

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

```
print(a * 3)  
print(b * 3)
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5] > * 연산도 같은 결과를 나타내는것을 확인 할 수 있다.  
[ 3  6  9 12 15]
```

\* '/' 연산 및 '-' 연산은 list 는 할 수 없으나 Numpy array 는 지원한다.

## - 다차원 행렬 및 행렬 연산 지원 (2차원 배열)

- . Shape : 현재 행렬의 크기 를 구할 수 있음

```
import numpy as np
```

```
a = [[1,1,1],[2,2,2],[3,3,3]]  
b = np.array(a)
```

```
print(a)  
print(b)
```

```
[[1, 1, 1], [2, 2, 2], [3, 3, 3]]  
[[1 1 1]  
 [2 2 2]  
 [3 3 3]]
```

```
print(b.shape)  
print(b * 3)
```

```
(3, 3)  
[[3 3 3]  
 [6 6 6]  
 [9 9 9]]
```

> b.shape 으로 배열의 형태를 확인 가능

> 3,3 배열

> \* 3 을 한 결과

## - 크기가 다른 두 행렬의 연산 지원 가능

```
import numpy as np
```

```
a1 = [[1,1,1],[2,2,2],[3,3,3]]  
a2 = [1,2,3]  
b1 = np.array(a1)  
b2 = np.array(a2)
```

> 2차원 요소 리스트 생성  
> 1차원 요소 리스트 생성  
> 2차원 배열 생성  
> 1차원 배열 생성

```
print(b1)  
print(b2)
```

```
[[1 1 1]  
 [2 2 2]  
 [3 3 3]]  
[1 2 3]
```

> 3, 3 배열 확인

> 1차원 배열 확인

```
print(b1 + b2)
```

```
[[2 3 4]  
 [3 4 5]  
 [4 5 6]]
```

> b1 의 각 행 값을 b2 와 합한 결과 산출

## - 리스트 자료형 보다 다양한 방식의 인덱싱 지원

. 리스트 : 요소 를 기준으로 인덱싱

. 행렬 : 위치를 기준으로 인덱싱

> [:, 2] : 2열에 있는 모든값

> [1, :2] : 1행(index 가 1행 = 두번째 행) 의 2번째까지의 값

```
import numpy as np
```

```
a = [[1,1,1],[2,2,2],[3,3,3]]  
b = np.array(a)
```

```
print(b, '\n\n')  
print(b[:,2], '\n\n')  
print(b[1,:2])
```

```
[[1 1 1]  
 [2 2 2]  
 [3 3 3]]
```

```
[1 2 3]
```

```
[2 2]
```

## - 제공하는 함수 종류

. 수학과 관련된 함수 : sqrt, log, max 등

. 행렬을 쉽게 정의할 수 있는 함수 : zeros, ones, arange 등

> 임의의 수를 추출 하는 random 함수

> 제곱근, 로그, 최대값 등을 산출하는 함수

```
#random 함수  
print(np.random.randint(1,10))  
print(np.random.normal())
```

```
6  
0.5203557227714081
```

```
print(np.sqrt(4))  
print(np.log(10))  
print(np.max([1,2,3]))
```

```
2.0  
2.302585092994046  
3
```

> 모든 요소 를 0 으로 초기화 하여 행렬을 생성 zeros

```
print(np.zeros((3,3)))
```

```
[[0. 0. 0.]  
 [0. 0. 0.]  
 [0. 0. 0.]]
```

## - Pandas 모듈의 개념

- . 행과 열로 이루어진 데이터 를 쉽게 다룰 수 있도록 도와주는 파이썬의 데이터 분석 전용 외부 모듈
  - > 특히 **대 용량**의 데이터를 처리하는데 편리함
- . 설치 방법
  - > Anaconda 설치 시 함께 설치
  - > !pip install pandas 로 설치 가능
- . 호출 방법
  - > import pandas as **pd** 로 호출 하여 사용
  - > **pd** 는 일반적으로 사용하는 Pandas 의 약칭

## - Series 자료형

- . index 와 값을 가지고 있음
- . **별도로 인덱스, 값을 출력 할 수 있음** (사용자가 index 를 임의로 지정 할 수 있음)
- . 정의할 때 인덱스를 따로 정해 줄 수 있음
  - > **S**eries 의 첫 문자 S 는 대문자

```
import pandas as pd
```

```
a = pd.Series([1,2,3,4])
print(a)
```

```
0    1
1    2
2    3
3    4
dtype: int64
```

> Series 자료형 생성

> index 가 자동으로 0, 1, 2, 3 으로 부여된 Series 가 생성 된 결과를 확인 할 수 있다.

```
print(type(a))
```

```
<class 'pandas.core.series.Series'>
```

> a 의 자료형은 a 인것을 확인 할 수 있다.

- . Series 의 index 를 임의로 부여

```
import pandas as pd
```

```
a = pd.Series([1,2,3,4] , index = ['a','b','c','d'])
print(a)
print(a.index)
```

```
a    1
b    2
c    3
d    4
dtype: int64
Index(['a', 'b', 'c', 'd'], dtype='object')
```

> index 를 'a','b','c','d' 로 정하여 부여함

> 각 index 와 값이 매칭되어 표현된다.

- . 파이썬의 딕셔너리 자료를 Series 로 변환 가능

> 딕셔너리 의 키 가 Series 의 인덱스 가 됨

```
import pandas as pd
```

```
dic = ({'a':1, 'b':2, 'c':3})
a = pd.Series(dic)
print(a)
print(a.index)
```

```
a    1
b    2
c    3
dtype: int64
Index(['a', 'b', 'c'], dtype='object')
```

. numpy 의 Array 자료형도 Series 로 변환 가능

```
import pandas as pd
import numpy as np

a = np.array([1,2,3,4])
b = pd.Series(a)
print(b)
```

```
0    1
1    2
2    3
3    4
dtype: int32
```

> numpy Array 자료형을 Series 로 할당  
\* index 는 자동으로 0 부터 할당됨

## - DataFrame 자료형

. 행과 열로 이루어진 자료형

. Series 와 마찬가지로 파이썬의 딕셔너리 자료형 또는 Numpy 의 array 로 변환 할 수 있음

. Series : 인덱스, 값으로만 구성, 1차원 배열 형태의 구조

. DataFrame : 행과 열로 구성, 2차원 테이블 형태의 자료구조

```
import pandas as pd

a = pd.Series({'a':[1,1,1], 'b':[2,2,2], 'c':[3,3,3]})
b = pd.DataFrame({'a':[1,1,1], 'b':[2,2,2], 'c':[3,3,3]})

print(type(a))
print(type(b))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.frame.DataFrame'>
```

> Series 와 DataFrame 의 생성 ( 유사함 )

```
print(a)
```

```
a    [1, 1, 1]
b    [2, 2, 2]
c    [3, 3, 3]
dtype: object
```

> Series : index 의 요소 로 배열을 정의함

```
print(b)
```

```
   a  b  c
0  1  2  3
1  1  2  3
2  1  2  3
```

> DataFrame : 열의 요소를 지정 하고

행에는 index 를 부여 하여 3,3 배열의 열 index 를 지정함

> 행 : index

> 열 : Columns

. 특정 컬럼의 데이터 를 손쉽게 변경 할 수 있음.

```
b['b'] = [5,6,7]
```

> b 컬럼의 데이터를 5,6,7 로 변경

```
print(b)
```

```
   a  b  c
0  1  5  3
1  1  6  3
2  1  7  3
```

> 결과

## - DataFrame 의 함수

.index 와 Columns 를 변경 할 수 있다.

```
In [17]: b.index = ['A', 'B', 'C']
b.columns = ['xx', 'yy', 'zz']
print(b)
```

	xx	yy	zz
A	1	5	3
B	1	6	3
C	1	7	3

> **.index** : 행의 index Key 변경  
> **.columns** : 열의 index Key 변경

**.iloc** : 행 인덱스로 값을 가져올 수 있음

	xx	yy	zz
A	1	5	3
B	1	6	3
C	1	7	3

```
: print(b['zz'])
```

```
A    3
B    3
C    3
Name: zz, dtype: int64
```

> 'zz' 컬럼의 데이터를 행 index 와 함께 모두 표현

```
: print(b.iloc[0])
```

```
xx    1
yy    5
zz    3
Name: A, dtype: int64
```

> iloc() : 0 행의 데이터를 컬럼 index 와 함께 표현한다.  
\* b.iloc[0] 에서 0 은 index 의 고유 주소 를 표기해야한다.  
b.iloc['A'] 는 사용할 수 없다.

**.loc** : 행 이름으로 값을 가져 올 수 있음

```
print(b.loc['A'])
```

```
xx    1
yy    5
zz    3
Name: A, dtype: int64
```

> 행 index 의 별칭으로 데이터 를 가져올 수 있다

**.describe()**

> DataFrame 에서 계산 가능한 값들에 대한 결과를 간략하게 보여줌

```
print(b.describe())
```

	xx	yy	zz
count	3.0	3.0	3.0
mean	1.0	6.0	3.0
std	0.0	1.0	0.0
min	1.0	5.0	3.0
25%	1.0	5.5	3.0
50%	1.0	6.0	3.0
75%	1.0	6.5	3.0
max	1.0	7.0	3.0

**.sum()**

> 합계 를 보여줌

```
print(b.sum())
```

```
xx    3
yy   18
zz    9
dtype: int64
```

> **axis** 옵션으로 행, 열 기준을 변경 할 수 있음

```
print(b.sum(axis=1))
```

```
A    9
B   10
C   11
dtype: int64
```

> axis = 1 로 행들의 합을 모두 나타낼 수 있다.  
\* **axis = 1** 은 고정 ( 행을 나타내는 문법)

> 기본적으로 컬럼 의 모든 값을 나타낸다.

- . 그 외의 다양한 함수들
  - > min , max : 최소 최대값
  - > argmin, argmax : 최소 최대값의 인덱스
  - > mean : 평균
  - > median : 중간값
  - > std, var : 표준편차, 분산
  - > unique : 특정 행 또는 열 에서 중복값을 제외한 유니크한 값을 반환.

\* 특정 행 또는 열의 unique 한 값을 반환하는 함수

```
print(b.iloc[0].unique())
```

[3]

```
print(b['zz'].unique())
```

[3]

## 영화 장르 별 빈도수 분석

### - 데이터 불러오기

- . Movies.csv 파일 위치 지정 및 현재 파이썬 폴더 로 옮기기
- . Pandas 모듈 의 read\_csv 함수 를 통해 DataFrame 으로 불러오기

```
import pandas as pd

data = pd.read_csv('movies.csv')
data
```

movieId		title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action Animation Comedy Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation Comedy Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

9742 rows × 3 columns

### - 장르 분리하기

- . 반복문을 활용하여 genres 컬럼의 장르 값들을 모두 분리하여 리스트에 저장

```
genre = []
for i in data['genres']:
    genre.extend(i.split('|'))
print(len(genre))
print(genre)
```

- > data['genre'] : genre 컬럼
- > genre 컬럼의 데이터를 '|' 로 분리한 리스트를 누적한다.
- > 누적된 총 장르의 개수 및 데이터 를 표현한다.

```
165
['Adventure', 'Animation', 'Children',
 'Romance', 'Comedy', 'Action', 'Crime',
 'iller', 'Comedy', 'Horror', 'Adventure'
 'Romance', 'Comedy', 'Comedy', 'Action'
 or', 'Mystery', 'Thriller', 'Action', '
 a', 'Romance', 'Crime', 'Drama', 'Drama
 'Comedy', 'Romance', 'Drama', 'War', 'A
 r', 'Drama', 'Romance', 'Mystery', 'Thr
 riller', 'Drama', 'Comedy', 'Crime', 'C
 ion', 'Comedy', 'Horror', 'Thriller', '
 r', 'Documentary', 'Crime', 'Drama', 'F
 a', 'Children', 'Comedy', 'Comedy', 'Ac
 a', 'Romance', 'Action', 'Adventure', '

```



### - 중복된 장르 제거하기

. Padnas 의 **unique** 함수 를 활용하여 중복을 제거한 장르를 저장

```
unique_genre = pd.unique(genre)
print(len(unique_genre))
print(unique_genre)

16
['Adventure' 'Animation' 'Children' 'Comedy' 'Fantasy' 'Romance' 'Drama'
 'Action' 'Crime' 'Thriller' 'Horror' 'Mystery' 'Sci-Fi' 'War' 'Musical'
 'Documentary']
```

>unique 함수로 중복을 제거한 데이터 를  
unique\_genre 리스트 로 생성

### - 빈도수 분석을 위한 DataFrame 생성하기

. numpy 모듈의 zeros 함수를 활용 하여 장르 개수만큼 비어있는 List 생성

```
import numpy as np

zero_data = np.zeros(len(unique_genre))
print(zero_data)
```

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] > 16 개의 빈 값이 있는 list 가 생성 되었다

. 장르 별 count 를 구분 할 수 있는 DataFrame 를 생성

> zero\_data 를 데이터로 컬럼의 이름은 count 로 설정 한 index 는 unique\_genre 로 하는 DataFrame 생성

```
resultDF = pd.DataFrame(zero_data, index = unique_genre, columns = ['count'])
print(result)
```

	count
Adventure	0.0
Animation	0.0
Children	0.0
Comedy	0.0
Fantasy	0.0
Romance	0.0
Drama	0.0
Action	0.0
Crime	0.0
Thriller	0.0
Horror	0.0
Mystery	0.0
Sci-Fi	0.0
War	0.0
Musical	0.0
Documentary	0.0
IMAX	0.0
Western	0.0
Film-Noir	0.0
(no genres listed)	0.0

> index = unique\_genre  
unique\_gere 의 리스트 를 행 index 로 설정함  
> columns = ['count']  
데이터 가 표현되는 컬럼 index 는 'count' 로 설정함

### - 반복문을 사용하여 빈도수 구하기

. 기존 전체 장르가 저장된 리스트를 하나씩 체크하여 해당 데이터의 값을 +1 해준다.

. genre 리스트 ( 모든 장르가 나열된 리스트 에서 반복적으로 하나씩 추출 하여 DataFrame 에 비교 후 등록)

```
for genrename in genre :
    resultDF.loc[genrename] += 1
print(result)
```

	count
Adventure	1263.0
Animation	611.0
Children	664.0
Comedy	3756.0
Fantasy	779.0
Romance	1596.0
Drama	4361.0
Action	1828.0
Crime	1199.0
Thriller	1894.0
Horror	978.0
Mystery	573.0
Sci-Fi	980.0
War	382.0
Musical	334.0
Documentary	440.0
IMAX	158.0
Western	167.0
Film-Noir	87.0
(no genres listed)	34.0