Python 실습 IV

- Pandas & Numpy

2021-1
산업컴퓨팅개론

목차

패키지 설치

패키지 설치

```
$ pip install pandas # numpy 도 함께 설치됨
```

- 터미널에 왼쪽과 같이 입력

```
(pandas_tutorial) C:₩Users₩antea>pip install pandas
Collecting pandas
  Downloading pandas-1.2.4-cp37-cp37m-win_amd64.whl (9.1 MB)
     |████████████████████████████████| 9.1 MB 6.8 MB/s
Collecting pytz>=2017.3
  Downloading pytz-2021.1-py2.py3-none-any.whl (510 kB)
     |████████████████████████████████| 510 kB 6.4 MB/s
Collecting numpy>=1.16.5
  Downloading numpy-1.20.2-cp37-cp37m-win_amd64.whl (13.6 MB)
     |████████████████████████████████| 13.6 MB 3.3 MB/s
Collecting python-dateutil>=2.7.3
  Downloading python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
     |████████████████████████████████| 227 kB 3.3 MB/s
Collecting six>=1.5
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, pytz, python-dateutil, numpy, pandas
Successfully installed numpy-1.20.2 pandas-1.2.4 python-dateutil-2.8.1 pytz-2021.1 six-1.16.0
```

Pandas 기초

Pandas

- 데이터 셋을 쉽게 다룰 수 있는 파이썬 라이브러리
- Data analysis, cleaning, exploring, manipulating 을 위한 함수 제공
- List, dictionary, numpy array, csv, excel 등 다양한 형태의 정형 데이터 처리 가능
- Series (1차원 데이터), DataFrame (2차원 데이터)
- 소스코드는 다음을 참고 https://github.com/pandas-dev/pandas

Pandas

- 실습에 필요한 모듈 import

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

데이터 구성, 불러오기 (Series)

- 1 차원 데이터 처리
- tuple, list, numpy array (1d) 등 1차원 배열을 인풋으로 함.
- Index는 데이터의 각 행을 나타내는 번호. 기본으로 0,1,.. 이지만 index 속성을 통해 지정 가능

```
# series 구성
s1 = pd.Series( tuple, list or numpy array(1d), index=[0,1,2,..] )
```

```
# Series
tuple_ = (1,2,3,4)
list_ = [1,2,3,4]
array_1d = np.array([1,2,3,4])

s1 = pd.Series(tuple_)
s2 = pd.Series(list_)
s3 = pd.Series(array_1d)
```

```
s1

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

```
s2

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

```
s3

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

데이터 구성, 불러오기 (DataFrame)

- 1, 2 차원 데이터 처리
- 1개 이상의 Series 를 모아 놓은 것.
- tuple, list, numpy array (1d) + list of list, dictionary, numpy array (2d)
- Columns, index 등의 속성으로 컬럼명, 인덱스 지정 가능

```
# dataframe 구성
df1 = pd.DataFrame( 1 or 2d array ,
                    columns=['col1', 'col2', ..],
                    index=[0, 1, 2, ..])
```

- read_csv (read_excel): csv나 excel 파일 읽기

```
# dataframe 구성 (file 읽기)
df1 = pd.read_csv( file.csv, index_col=0 )
```

데이터 구성, 불러오기 (DataFrame)

- examples
  - Dictionary: key 를 컬럼으로, value 를 리스트 형태의 값으로 지정

```
# DataFrame
# dictionary
mydataset = {
    'student':['mike', 'james', 'dean'],
    'score':[88, 92, 78]
}
```

```
df1 = pd.DataFrame(mydataset)
df1
```

	student	score
0	mike	88
1	james	92
2	dean	78

  - Numpy array: array 의 행과 열을 그대로 유지.

```
# np.array
array = np.array([['mike', 'james', 'dean'], [88, 92, 78]])
array.T
```

```
array([['mike', '88'],
       ['james', '92'],
       ['dean', '78']], dtype='<U11')
```

```
df2 = pd.DataFrame(array.T,
                   columns=['student', 'score'],
                   index=[1,2,3]) # default index = [0,1,2]
df2
```

	student	score
1	mike	88
2	james	92
3	dean	78

데이터 구성, 불러오기 (DataFrame)

- examples
  - read_csv

```
file_path = 'titanic.csv'
df1 = pd.read_csv(file_path)
```

```
In [18]: df1.head() # head(n): 첫 n 행 출력 # tail(n): 마지막 n 행 출력
```

	Passenger Class	Name	Sex	Age	No of Siblings or Spouses on Board	No of Parents or Children on Board	Ticket Number	Passenger Fare	Cabin	Port of Embarkation	Life Boat	Survived
0	First	Allen, Miss. Elisabeth Walton	Female	29.0	0.0	0.0	24160	211.3	B5	Southampton	2	Yes
1	First	Allison, Master. Hudson Trevor	Male	0.9	1.0	2.0	113781	151.6	C22 C26	Southampton	11	Yes
2	First	Allison, Miss. Helen Loraine	Female	2.0	1.0	2.0	113781	151.6	C22 C26	Southampton	NaN	No
3	First	Allison, Mr. Hudson Joshua Creighton	Male	30.0	1.0	2.0	113781	151.6	C22 C26	Southampton	NaN	No
4	First	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	Female	25.0	1.0	2.0	113781	151.6	C22 C26	Southampton	NaN	No

데이터 확인

- shape: dataframe 의 행, 열 수를 출력

```
df1.shape
```

```
(1309, 12)
```

- info(): 각 열의 데이터 타입과 전체 데이터프레임의 메모리 사용량을 확인 가능

```
df1.info() # 각 열의 데이터 타입과 전체 데이터프레임의 메모리 사용량을 확인 가능
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Passenger Class                 1309 non-null   object
 1   Name                            1309 non-null   object
 2   Sex                             1309 non-null   object
 3   Age                             1046 non-null   float64
 4   No of Siblings or Spouses on Board  1309 non-null   float64
 5   No of Parents or Children on Board  1309 non-null   float64
 6   Ticket Number                   1309 non-null   object
 7   Passenger Fare                  1308 non-null   float64
 8   Cabin                           295 non-null    object
 9   Port of Embarkation             1307 non-null   object
 10  Life Boat                       486 non-null    object
 11  Survived                        1309 non-null   object
dtypes: float64(4), object(8)
memory usage: 122.8+ KB
```

- columns: 컬럼 확인

```
df1.columns
```

```
Index(['Passenger Class', 'Name', 'Sex', 'Age',
       'No of Siblings or Spouses on Board',
       'No of Parents or Children on Board', 'Ticket Number', 'Passenger Fare',
       'Cabin', 'Port of Embarkation', 'Life Boat', 'Survived'],
      dtype='object')
```

- describe(): 데이터 통계치 확인

```
df.describe()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000

- median(), mean(), std(), max(), min(): 컬럼별 데이터 통계치 출력

```
df1['Age'].mean()
```

```
29.88116634799235
```

- unique(): 컬럼별 고유값 출력

```
df1['Passenger Class'].unique()
```

```
array(['First', 'Second', 'Third'], dtype=object)
```

데이터 추출

- 특정 행 추출

```
# 특정 행 추출
df.iloc[36] # df.iloc[36, :]
df.loc[36] # df.loc[36, :]
```

```
df1.iloc[36]
```

Passenger Class	First
Name	Bowerman, Miss. Elsie Edith
Sex	Female
Age	22.0
No of Siblings or Spouses on Board	0.0
No of Parents or Children on Board	1.0
Ticket Number	113505
Passenger Fare	55.0
Cabin	E33
Port of Embarkation	Southampton
Life Boat	6
Survived	Yes

Name: 36, dtype: object

- 특정 열 추출

```
# 특정 열 추출
df['col_name']
```

```
df1['Age']
```

```
0       29.0
1        0.9
2        2.0
3       30.0
4       25.0
       ...
1304    14.5
1305     NaN
1306    26.5
1307    27.0
1308    29.0
Name: Age, Length: 1309, dtype: float64
```

```
df1[['Age', 'Name']]
```

	Age	Name
0	29.0	Allen, Miss. Elisabeth Walton
1	0.9	Allison, Master. Hudson Trevor
2	2.0	Allison, Miss. Helen Loraine
3	30.0	Allison, Mr. Hudson Joshua Creighton
4	25.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
...	...	...

- 특정 행 + 특정 열 추출

```
# 특정 행 + 특정 열 추출
df.iloc[36, 2]
df.loc[36, 'col_name']
```

```
df1.iloc[0:2, 3:6]
```

	Age	No of Siblings or Spouses on Board	No of Parents or Children on Board
0	29.0	0.0	0.0
1	0.9	1.0	2.0

필터링

- 해당 조건에 만족하는 데이터만 추출

```
# filtering
df[ 조건 ]
```

- reset_index(): index를 0,1,... 로 재설정. (drop=True로 하면 기존 인덱스는 삭제됨.)

```
# reset_index
df1_filtered = df1[(df1.Sex == 'Male') & (df1.Age >= 30)].reset_index(drop=True)
```

```
df1_filtered.tail(3)
```

	Passenger Class	Name	Sex	Age	No of Siblings or Spouses on Board	No of Parents or Children on Board	Ticket Number	Passenger Fare	Cabin	Port of Embarkation	Life Boat	Survi
304	Third	Widegren, Mr. Carl/Charles Peter	Male	51.0	0.0	0.0	347064	7.8	NaN	Southampton	NaN	No
305	Third	Wittevrongel, Mr. Camille	Male	36.0	0.0	0.0	345771	9.5	NaN	Southampton	NaN	No
306	Third	Youseff, Mr. Gerious	Male	45.5	0.0	0.0	2628	7.2	NaN	Cherbourg	NaN	No

- ==, >, &, | 사용하여 필터링

```
# 성별이 남자이며, 나이가 30세 이상인 사람만 필터링
df1[(df1['Sex'] == 'Male') & (df1['Age'] >= 30)]
df1[(df1.Sex == 'Male') & (df1.Age >= 30)]  # 컬럼에 whitespace 가 없다면 . 으로 접근가능
```

- isin(): 특정값을 포함하는 데이터만 필터링

```
df1[df1['Port of Embarkation'].isin(['Southampton', 'Cherbourg'])]
```

- 문자열 메소드를 통한 필터링

```
# 이름에 M이 들어가고 A로 시작하는 인원만 필터링
df1[df1.Name.str.contains('M') & df1.Name.str.startswith('A')]
```

- columns.difference(): 특정 컬럼을 제외하도록 필터링

```
df1[df1.columns.difference(['Life Boat', 'Ticket Number'])]
```

결측치 처리

- 결측치: 누락된 값. 데이터에서 빈 칸으로 존재하는 값
- isna(): 결측치이면 True, 아니면 False를 반환

```
df1.isna()
```

	Passenger Class	Name	Sex	Age	No of Siblings or Spouses on Board	No of Parents or Children on Board	Ticket Number	Passenger Fare	Cabin	Port of Embarkation	Life Boat	Survived
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	True	False
4	False	False	False	False	False	False	False	False	False	False	True	False

```
df1.isna().sum()
```

```
Passenger Class                  0
Name                             0
Sex                              0
Age                            263
No of Siblings or Spouses on Board  0
No of Parents or Children on Board  0
Ticket Number                    0
Passenger Fare                   1
Cabin                         1014
Port of Embarkation              2
Life Boat                      823
Survived                         0
dtype: int64
```

- dropna(): 결측치 제거

```
df.dropna(axis=0)
# axis = 0 or 'index' : 결측치 포함 행 제거 (default)
# axis = 1 or 'columns' : 결측치 포함 행 제거
```

```
df1.shape
```

```
(1309, 12)
```

```
df1.dropna().shape
```

```
(180, 12)
```

```
df1.dropna(1).shape
```

```
(1309, 7)
```

결측치 처리

- fillna(): 결측치 채움

```
df.fillna(0)
# 지정한 값 (0) 으로 채움

df.fillna(method='bfill')# 'backfill', 'ffill', 'pad', default: None
# pad, ffill : 이전 값으로 채움
# backfill, bfill: 다음 값으로 채움
# None: 지정한 값으로 채울 때
```

```
df_example3 = pd.DataFrame({'a':[1,2,3, np.nan ,4], 'b':[5,6,7,8, np.nan]})
df_example3
```

	a	b
0	1.0	5.0
1	2.0	6.0
2	3.0	7.0
3	NaN	8.0
4	4.0	NaN

```
df_example3.fillna(method='bfill')
```

	a	b
0	1.0	5.0
1	2.0	6.0
2	3.0	7.0
3	4.0	8.0
4	4.0	NaN

```
df_example3.fillna(method='ffill')
```

	a	b
0	1.0	5.0
1	2.0	6.0
2	3.0	7.0
3	3.0	8.0
4	4.0	8.0

자료형 변경

- astype: 특정 series 의 데이터 타입을 변경

```
df_example = pd.DataFrame({'a':[1, 2, 3],
                           'b':['4', '5', '6'],
                           'c':[7.0, 8.0, 9.0]})
df_example.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   a       3 non-null      int64
 1   b       3 non-null      object
 2   c       3 non-null      float64
dtypes: float64(1), int64(1), object(1)
memory usage: 200.0+ bytes
```

```
df_example['a'] = df_example['a'].astype('float')
df_example['b'] = df_example['b'].astype('int')
df_example['c'] = df_example['c'].astype('str')
df_example.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   a       3 non-null      float64
 1   b       3 non-null      int32
 2   c       3 non-null      object
dtypes: float64(1), int32(1), object(1)
memory usage: 188.0+ bytes
```

자료형 변경 - 더미변수화

- get_dummies
- Object 타입을 숫자로 변경할 경우 사용
- 'male', 'female' 등 값의 우열이 없는 변수인 경우 1,2 등의 숫자로 바꾸면 정확한 분석이 이뤄질 수 없음.
- One-hot encoding 이란 해당 카테고리에 속하면 1, 그 이외의 카테고리는 0으로 바꾸는 처리방법

- 기존 dataframe

```
df_example2
```

	var1	var2	var3
0	male	1	0.2
1	female	2	0.6
2	male	3	0.2

- Series 를 인풋으로 받으면 각 값을 컬럼으로 하는 one hot encoding 진행

```
pd.get_dummies(df_example2['var1'])
```

	female	male
0	0	1
1	1	0
2	0	1

- DataFrame 을 인풋으로 받으면 prefix를 컬럼명, suffix를 값으로 처리

```
pd.get_dummies(df_example2)
```

	var2	var3	var1_female	var1_male
0	1	0.2	0	1
1	2	0.6	1	0
2	3	0.2	0	1

데이터 병합

- concat(): 두 데이터프레임을 위, 아래로 혹은 양 옆으로 병합

```
# concat
df_concat = pd.concat([df1, df2], axis=0) # axis=0: 위, 아래 (default), # axis=1: 양 옆
```

```
# concat
df_concat1 = pd.DataFrame({'a':[1,2,3], 'b':[10, 20, 30], 'c':[100, 200, 300]})
df_concat2 = pd.DataFrame({'a':[4,5,6], 'b':[40, 50, 60], 'c':[400, 500, 600]})
df_concat = pd.concat([df_concat1, df_concat2])
df_concat
```

	a	b	c
0	1	10	100
1	2	20	200
2	3	30	300
0	4	40	400
1	5	50	500
2	6	60	600

```
df_concat = pd.concat([df_concat1, df_concat2], axis=1)
df_concat
```

	a	b	c	a	b	c
0	1	10	100	4	40	400
1	2	20	200	5	50	500
2	3	30	300	6	60	600

- axis=0 일 경우 컬럼을 기준으로, 1일 경우 index를 기준으로 병합

```
# concat
df_concat1 = pd.DataFrame({'a':[1,2,3], 'b':[10, 20, 30], 'c':[100, 200, 300]})
df_concat2 = pd.DataFrame({'b':[4,5,6], 'c':[40, 50, 60], 'd':[400, 500, 600]})
df_concat = pd.concat([df_concat1, df_concat2])
df_concat
```

	a	b	c	d
0	1.0	10	100	NaN
1	2.0	20	200	NaN
2	3.0	30	300	NaN
0	NaN	4	40	400.0
1	NaN	5	50	500.0
2	NaN	6	60	600.0

```
df_concat1 = pd.DataFrame({'a':[1,2,3], 'b':[10, 20, 30], 'c':[100, 200, 300]})
df_concat2 = pd.DataFrame({'a':[4,5,6], 'b':[40, 50, 60], 'c':[400, 500, 600]}, index=[1,2,3])
df_concat = pd.concat([df_concat1, df_concat2], axis=1)
df_concat
```

	a	b	c	a	b	c
0	1.0	10.0	100.0	NaN	NaN	NaN
1	2.0	20.0	200.0	4.0	40.0	400.0
2	3.0	30.0	300.0	5.0	50.0	500.0
3	NaN	NaN	NaN	6.0	60.0	600.0

데이터 병합

- merge() : 두 데이터를 키 값을 기준으로 양 옆으로 병합. (Inner, outer, left, right)

```
# merge
df_merge = pd.merge(df1, df2, on='key', how='inner') # 'inner', 'outer', 'left', 'right'

# inner: df1, df2의 키 값의 교집합 병합
# outer: df1, df2의 키 값의 합집합 병합
# left: df1의 키 값을 기준으로 병합
# right: df2의 키 값을 기준으로 병합
```

```
df_left = pd.DataFrame(
    {
        "key": ["K0", "K1", "K3", "K5"],
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
    }
)


df_right = pd.DataFrame(
    {
        "key": ["K0", "K1", "K2", "K4"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    }
)


df_result_inner = pd.merge(df_left, df_right, on="key", how="inner")
df_result_outer = pd.merge(df_left, df_right, on="key", how="outer")
df_result_left = pd.merge(df_left, df_right, on="key", how="left")
df_result_right = pd.merge(df_left, df_right, on="key", how="right")
```

df_result_inner

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1

df_result_outer

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K3	A2	B2	NaN	NaN
3	K5	A3	B3	NaN	NaN
4	K2	NaN	NaN	C2	D2
5	K4	NaN	NaN	C3	D3

df_result_left

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K3	A2	B2	NaN	NaN
3	K5	A3	B3	NaN	NaN

df_result_right

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	NaN	NaN	C2	D2
3	K4	NaN	NaN	C3	D3

데이터 내보내기

- to_csv

```
# 파일 내보내기
df.to_csv('내보낼 파일명.csv', index=False)
```

Numpy

- 다차원 배열을 쉽게 처리하고 효율적으로 사용할 수 있도록 지원하는 파이썬 라이브러리
- Numerical Python
- 메모리 내 연속적으로 존재하기 때문에 list 보다 빠른 연산 수행

Array

- 배열 생성

```
# 1D
np.array([1,2,3,4,5])

# 2D
np.array([[1,2,3], [4,5,6]])

# 3D
np.array([[[1,2,3,4], [5,6,7,8]], [[9,10,11,12], [13,14,15,16]]])
```

- Shape, type, ndim 확인

```
arr = np.array([[[1,2,3,4], [5,6,7,8]], [[9,10,11,12], [13,14,15,16]]])

print('shape: ', arr.shape)
print('data type: ', arr.dtype)
print('number of dimensions: ', arr.ndim)
```

```
shape:  (2, 2, 4)
data type:  int32
number of dimensions:  3
```

Array type

- dtype

	dtype
정수	Int8, int16, int32, int64
실수	Float16, float32, float64
복소수	Complex64, complex128
그 외	Bool, object, string, …

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8], dtype='f8') # 약어 + bytes
print(arr.dtype)
print(arr)

float64
[1. 2. 3. 4. 5. 6. 7. 8.]
```

- `i` - integer
- `b` - boolean
- `u` - unsigned integer
- `f` - float
- `c` - complex float
- `m` - timedelta
- `M` - datetime
- `O` - object
- `S` - string
- `U` - unicode string
- `V` - fixed chunk of memory for other type (void)

- astype

```
print(arr.astype(int).dtype)

int32
```

```
print(arr.astype('i4').dtype)

int32
```

Array indexing & slicing

- List 를 인덱싱, 슬라이스 하는 방식과 유사
- indexing

```
arr
```

```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8]],

       [[ 9, 10, 11, 12],
        [13, 14, 15, 16]]])
```

```
# 첫 번째 배열의 두 번째 배열에 엑세스
arr[0, 1]
```

```
array([5, 6, 7, 8])
```

```
# 마지막 배열의 마지막 배열에 엑세스
arr[-1, -1]
```

```
array([13, 14, 15, 16])
```

- slice

```
arr[:,:,1:3]
```

```
array([[[ 2,  3],
        [ 6,  7]],

       [[10, 11],
        [14, 15]]])
```

Array reshape

- reshape

```
# reshape
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1)
print(arr) # 2D
print()
print(newarr) # 1D flatten
```

```
[[1 2 3]
 [4 5 6]]

[1 2 3 4 5 6]
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4,3)
print(arr) # 1D
print()
print(newarr) # 2D
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12]

[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Array Join & split

- Concatenate: dimension 추가 없이 두 array 병합

```
arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr_axis0 = np.concatenate((arr1, arr2), axis=0) # axis=0 default
print(arr_axis0)
print()
arr_axis1 = np.concatenate((arr1, arr2), axis=1)
print(arr_axis1)
```

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

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

- Stack: dimension 을 추가하여 두 array 병합

```
arr_stack = np.stack((arr1, arr2))

print(arr_stack)
```

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

 [[5 6]
  [7 8]]]
```

- Array_split: array 를 지정한 개수로 분할하여 리스트로 반환. 다차원 행렬을 분할하는 경우는 axis 설정 가능

```
## array split
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

Array 연산

- 기본적인 연산 +, *, -, / (broadcasting 방법): dimension 은 달라도 되지만 shape 이 맞아야 함.

```
# array 연산
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print('+: ', arr1 + arr2)
print('*: ', arr1 * arr2)
```

```
+:  [5 7 9]
*:  [ 4 10 18]
```

```
# diffent dimension (broadcasting)
arr1 = np.array([[1, 2, 3], [1, 2, 3]])
arr2 = np.array([4, 5, 6])

print('+: ', arr1 + arr2)
print('*: ', arr1 * arr2)
```

```
+:  [[5 7 9]
 [5 7 9]]
*:  [[ 4 10 18]
 [ 4 10 18]]
```

```
arr1 = np.array([1, 2])
arr2 = np.array([4, 5, 6])
print('+: ', arr1 + arr2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-144-6d82ae96ed09> in <module>
      1 arr1 = np.array([1, 2])
      2 arr2 = np.array([4, 5, 6])
----> 3 print('+: ', arr1 + arr2)
      4 print('*: ', arr1 * arr2)

ValueError: operands could not be broadcast together with shapes (2,) (3,)
```

- dot (matmul): 행렬 곱 연산

```
# np.dot
arr1 = np.array([[1, 2, 3], [1, 2, 3]])  # shape: (2,3)
arr2 = np.array([[4, 5, 6]]).T  # shape (3,1), T (transpose)
print(np.dot(arr1, arr2))
```

```
[[32]
 [32]]
```

과제

과제

- titanic.csv 전처리하기

1. 결측치 제거

- titanic.csv 를 읽고 각 컬럼 별로 결측치 수가 30% 이상이면 해당 컬럼을 제거하고 30% 이하이면 행을 제거하는 함수를 작성하시오.

- 함수의 return 에는 전처리된 데이터 프레임이 나와야 함
- return 할 데이터프레임은 인덱스가 0부터 재설정되어야 함.

2. 자료형 변경

- 위에서 전처리된 데이터프레임을 읽고 Age 컬럼의 타입을 integer, Sex, Passenger Class, Port of Embarkation 의 경우는 더미변수로 바꾸는 함수를 작성하시오. (데이터 병합 이용)

- 함수의 return 에는 전처리된 데이터 프레임이 나와야 함.
- return 할 데이터프레임은 인덱스가 0부터 재설정되어야 함.