

# 파이썬 실습1

---

- 중복 수를 제거하고, 입력된 순서대로 출력하는 프로그램 작성
  - [1,1,3,3,5,2,2,8]이 입력값이면, 출력은 [1,3,5,2,8]로 입력된 순서는 유지하고 중복값을 제거함
  - set 함수는 중복값은 제거하되, 정렬된 상태의 결과를 반환

# 파이썬 실습1 - itertools

- iterable 자료형에 대해 많은 부가기능을 제공해주는 파이썬의 대표적인 모듈
  - itertools, collections 모듈
- 다음 두 문자열의 카티션 곱(Cartesian product)을 구하는 문제를 생각
  - 즉, AB 와 xy 가 있다면,  $AB * xy = (Ax, Ay, Bx, By)$  의 결과임
  - 문자열 str1 = ABC, str2 = xyz 일 때, 카티션 곱을 구하는 프로그램을 작성

```
str1 = "ABC"; str2="xzy"

result = []
for i in str1:
    for j in str2:
        result.append(i+j)

print(result)
```

# 파이썬 실습1 - itertools

---

- 2차원 리스트를 1차원 리스트로 변경
  - `[[1,2,3],[5],[2,4,6]] → [1,2,3,5,2,4,6]`

# 파이썬 실습1 - itertools

## ■ itertools 의 product 메서드 적용

```
import itertools

str1 = "ABC"; str2="xzy"
result = list(itertools.product(str1,str2))
print(result)
print(list(map(".join, itertools.product(str1,str2))))
```

- 1차원 리스트의 값으로 이루어진 모든 순열을 사전순으로 정렬하여 리스트로 출력하는 프로그램 작성 → itertools 의 permutation 사용
  - $[1,2,3] \rightarrow [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]$ 
    - itertools 의 permutation은 입력값의 자료형이 str 임
  - 입력되는 값이 숫자형태의 값이면, 문자열로 변환한 후 permutation 메서드에 적용
  - 결과의 마지막 단계에서 원래의 숫자형태로 자료 변환을 해주어야 함

# 파이썬 실습1 - itertools

---

```
def permut(num):
    temp = map("".join, itertools.permutations(map(str,num)))
    result = list(map(list,temp))
    for i, value in enumerate(result):
        result[i] = list(map(int,value))
    return result

if __name__ == "__main__":
    num = [1,2,3]
    print(permut(num))
```

# 파이썬 실습2

---

- 다음 문자열 중에서 가장 많이 등장하는 알파벳을 사전 순으로 출력하는 프로그램 작성
  - 문자열은 모두 소문자임
  - 문자열의 길이는 1부터 100이하
  - 예:
    - "abba" → 'ab'
    - "cccaaa" → "ac"
    - "pythony" → 'y'

# 파이썬 실습3

---

- 리스트에 입력된 값 중에서 중복된 값을 찾아 출력하는 프로그램 작성
  - [10,20,50,70,30,20,70] → [20,70]
  - ['Jane','Python','Ann','Ruby','Python'] → ['Python']

# 파이썬 데이터처리 및 분석

## numpy/pandas 기본

---

2020.08. 19. 수요일

최희련

**En-CORE**

**Data Science Edu.**



# Google 의 colab 사용 시 데이터 읽기

---

- colab에서 필요한 패키지 또는 모듈 설치는 !pip install 패키지이름
- 내 컴퓨터의 파일 읽기
  - google.colab 의 drive 사용(<https://colab.research.google.com/>)

```
from google.colab import drive  
drive.mount('/gdrive', force_remount=True)
```

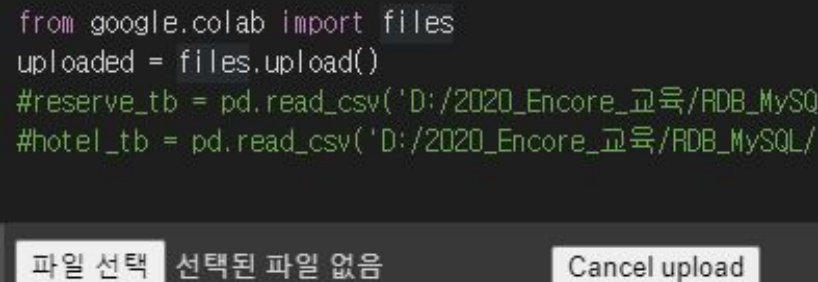
- 위의 코드를 작성하고 실행하면, 나타나는 URL 을 클릭 후 인증코드를 enter your authorization code 에 입력
- 구글 드라이브 설정메뉴의 일반사항에서 업로드변환 속성을 확인
  - google 문서편집기 형식으로 변환되지 않도록 함
- 업로드 후에 최종 경로는 /gdrive/My Drive/파일이름 으로 코드에 입력
- 특정 파일을 계속 사용할 경우에 추천

# Google 의 colab 사용 시 데이터 읽기

## ○ Google.colab 에서 files 사용

```
from google.colab import files
uploaded = files.upload()
```

- 위의 코드를 작성하고 실행하면, 하단에 파일선택이 나타남
- 파일선택을 클릭하면, 파일찾기 탐색기가 실행되고, 원하는 파일을 선택 후 <열기> 버튼을 클릭



```
from google.colab import files
uploaded = files.upload()
#reserve_tb = pd.read_csv('D:/2020_Encore_교육/RDB_MySQL/
#hotel_tb = pd.read_csv('D:/2020_Encore_교육/RDB_MySQL/
```

파일 선택    선택된 파일 없음    Cancel upload

- 이 방식은 colab 클라우드 상에 파일을 업로드 시키는 것으로 클라우드에 업로드된 파일은 일정시간이 지나면 자동 삭제

```
import os
for fn in uploaded.keys():
    print('User uploaded file "{}" '.format(fn))
20 df = pd.read_csv(io.StringIO(uploaded['reserve.csv'].decode('utf-8')))
```

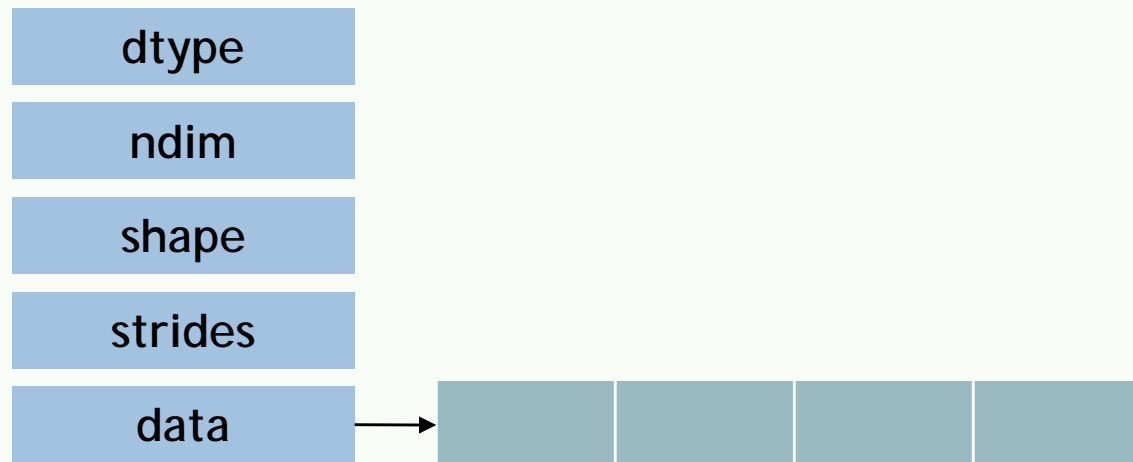
# 데이터분석 모듈 - Numpy

---

## ■ 수학, 과학 연산을 위한 패키지

- 다차원 배열 데이터를 효과적으로 처리할 수 있음
- [www.numpy.org](http://www.numpy.org) 에서 자세한 정보 확인

## ■ 다차원 배열 구성



- `strides` 는 걸음걸이의 의미로, 한 원소에서 또는 한 차원에서 다음으로 넘어가는 bytes 를 의미함 → `dtype` 으로 자료형의 사이즈를 확인 후 사용

# 데이터분석 모듈 - Numpy

## ■ 배열타입의 데이터 생성

```
import numpy as np

arr1 = np.array([1,2,3,4,5])# array 함수 내에 dtype 의 속성으로 자료값 변경
arr1_1 = np.array([10,20,30,40,50], dtype='float32')
arr1_2 = arr1_1.astype('int32')

arr2 = np.array([[1,2,3,4,5],
                 [10,5,3,8,6],
                 [9,6,11,4,2]],)

arr3 = np.arange(0,10,2)
arr4 = np.arange(12).reshape(3,4)
arr5 = np.linspace(1,10,5)
#linspace(start, end, num), start와 end 사이의 등간격으로 생성, num 생략 시 1임
arr6 = np.linspace(0,np.pi, 5)

print(arr1.dtype, arr1.ndim, arr1.strides, arr1.shape)
```

# 데이터분석 모듈 - Numpy

---

## ■ 배열 데이터 생성 시 random 사용

- `np.random.rand(n,m)`: 0~1 사이의 균일분포 정규분포를  $n*m$  만큼 발생
  - `np.random.rand(6)`, `np.random.rand(3,4)`
- `np.random.randint(start, end)`: start와 end 사이의 정수 발생
- `np.random.randn(n,m)`: 평균 0, 표준편차 1인 정규분포를  $n*m$  만큼 발생
  - `np.random.randn(8)`, `np.random.randn(2,3)`
- `np.random.shuffle`
- `np.random.choice(집합형데이터, n, replace=True, p=None)`
  - $n$  은 선택 개수, `replace`는 중복선택 여부,  $p$  은 데이터 선택의 확률
- `np.unique`: 중복이 제거된 유일 값
- `np.random.seed(seed값)`: 랜덤 씨드 설정

# 데이터분석 모듈 - Numpy

---

## ■ 데이터 선택

- 인덱스와 슬라이싱 이용
- 배열명[인덱스번호]
- 배열명[[인덱스번호1, 인덱스번호2,...,인덱스번호n]] : 여러 개의 원소 선택 시
- 배열명[행번호, 열번호] 또는 배열명[행번호][열번호]
- 배열명[[행번호1,행번호2,...,행번호n],[열번호1,열번호2,...,열번호n]]
- 배열명[start:end:step]

# 데이터분석 모듈 - Numpy

---

```
import numpy as np

arr1 = np.arange(1,10)
print(arr1)
print(arr1[2], arr1[[2,4,6]])
arr2 = arr1.reshape(3,3).copy()
arr2[1,1] = 10
print(arr2)
print(arr2[2][1], arr2[[0,1,2],[1,1,1]])
print(arr2[1:])#행 전체의 슬라이싱
print(arr2[1:2, 0:3])
print(arr2.tolist())
```

# 데이터분석 모듈 - Numpy

---

## ■ numpy 메소드

- min, max, sum, argmin, argmax, where

```
arr1 = np.array([10,30,50,20,80, 40, 60])  
Index : [0] [1] [2] [3] [4] [5] [6]
```

- where() :

- 배열에서 조건에 맞는 값의 색인 위치 확인
- 조건에 맞는 값의 인덱싱
- 조건에 맞는 값을 특정 다른 값으로 변환 - for loop & if 조건문 사용보다 속도가 빠름



# 데이터분석 모듈 - Numpy

---

```
import numpy as np
arr1 = np.array([10,30,50,20,80,40,60])
print(arr1.min(), np.min(arr1))
print(arr1.max(), np.max(arr1))
print(arr1.argmin(), np.argmin(arr1),arr1.argmax(),np.argmax(arr1))
result = np.where(arr1>50)
print(result, type(result))
print(result[0])
print(result[0][0])

result = arr1[np.where(arr1>50)]
print(result)

print(np.where(arr1>50, 15, arr1))#arr1에 결과가 반영되지는 않음
result = np.where(arr1>50, 15, arr1)#조건에 맞는 값을 15로 교체
print(result, arr1)
```

# 데이터분석 모듈 - pandas

## ■ pandas 자료형 - Series, DataFrame

- Series: pandas.Series() 사용

index :	0	1	2	3	4	...	n-1
values :	Value 1	Value 2	Value 3	Value 4	Value 5	...	Value n

딕셔너리의 key → index, value → values

```
dic1 = {'a': 20, 'b': 40, 'c': 30, 'd': 60, 'e': 70}
sr = pandas.Series(dic1)

print(sr.index, sr.values)
```

Key 가 없는 리스트 또는 튜플은 자동으로 index 생성

```
lit1 = ['2020-08-19', 'analysis', 9, True, ]
sr = pandas.Series(lit1)

print(sr.index, sr.values)
```

# 데이터분석 모듈 - pandas

```
import pandas as pd

lit1 = ['2020-08-19', 'analysis', 9 , True]
sr = pd.Series(lit1)
print(sr.index, sr.values)
sr = pd.Series(lit1, index=['a','b','c','d'])
#sr.index = ['aa','bb','cc','dd']
select1 = sr[[0,1]]#select1은 Series, sr[['a','b']]
select2 = sr[0]#select2 는 2020-08-19의 자료형인 str
print(type(select1), type(select2))
print(select1)
print(select1.values)
print(select2)
```

## ■ 다음 코드의 결과는?

```
print(sr[1: 3])
print(sr['b':'d'])
```

# 데이터분석 모듈 - pandas

- DataFrame: `pandas.DataFrame()` 사용의 2차원 자료

column

	column 1	column 2	column 3	...	column n
	value				
	value				
	value				
	value				
	value				
	value				

row index

행, row, record

Series 0      Series 1      Series 2      ....      Series n

- `pandas.DataFrame(데이터, index=, columns=, dtype=,)`

# 데이터분석 모듈 - pandas

```
import pandas as pd
from IPython.display import display
dic1 = {'seoul':[100,300,250], 'pusan':[120,220,180], 'incheon':[90,250,170]}
df1 = pd.DataFrame(dic1)
#display(df1)
df1.index = ['apple','pear','mandarin']
#display(df1)
df2 = pd.DataFrame(df1, copy=True, dtype=np.float)
df2.rename(index={'apple':'water_melon'}, inplace=True)
df1['seoul']['apple'] = 110
display(df1, df2)

df2.drop('water_melon', inplace=True) # axis=0 0 | default
display(df2)
```

- df2의 'seoul'의 column 을 삭제하려면?

# 데이터분석 모듈 - pandas

## ■ 행, 열 선택

구분	loc	iloc
탐색 대상	인덱스이름(index label) df.loc['apple'], df.loc[['apple', 'mandari,']]	정수형 위치 인덱스 df.iloc[0] df.iloc[[0, 2]]
범위 지정	가능(범위 끝 포함) df.loc['apple':'pear']	가능(범위 끝 제외) df.iloc[0:1]
행, 열 선택 시	[행이름, 열 이름] df.loc['apple', 'seoul'] df.loc['apple', ['seoul', 'Incheon']] df.loc['apple':'pear', 'seoul']	또는 [행번호, 열번호] df.iloc[0, 0] df.iloc[0, [0, 2]] df.iloc[0:2, 0]

## ■ 열 선택

- 데이터프레임객체[열이름], 데이터프레임객체.열이름

# 데이터분석 모듈 - pandas

```
print(df1['seoul'])
print(df1.seoul)
print()
print(df1[['seoul']])
#seoul의 apple 값 확인
print(df1.seoul['apple'])
print(df1['seoul']['apple'])
print(df1['seoul'][0])
print(df1.loc['apple','seoul'])
print(df1.iloc[0,0])
```

```
#seoul의 apple, pear, mandarin 확인
print(df1.seoul[:3])#df1['seoul'][:3]
print(df1.seoul[[0,1,2]])
print(df1.seoul['apple':'mandarin'])
```

```
#seoul, pusan의 mandarin 확인
print(df1.loc['mandarin',['seoul','pusan']])
```

#pear의 seoul, pusan, incheon 값?

#apple, pear, mandarin의 pusan, incheon 값?

# 데이터분석 모듈 - pandas

## ■ 행 추가

- 데이터프레임 객체.

loc['새로운 행이름'] =  
데이터값

## ■ 열 추가

- 데이터프레임 객체['새로운  
열이름'] = 데이터값

- 데이터프레임 객체.  
새로운 열이름 = 데이터값  
은 오류 발생

```
import pandas as pd
from IPython.display import display
dic1 = [[100,300,250], [120,220,180],[90,250,170]]
df1 = pd.DataFrame(dic1, \
                    columns=['seoul','puan','incheon'],\
                    index=['apple','pear','mandarin'])
display(df1)
df1['gwangju'] = [90, 310, 150]
display(df1)
df1.loc['grape'] = [280, 250, 180, 200]
df1.loc[2] = np.nan
display(df1)

#값 변경
df1['seoul']['apple'] = 110
df1['puan']['apple','pear'] = 130, 240
display(df1)

df1.loc['mandarin', 'incheon'] = 150
df1.T #column과 index 위치 바꾸기
```



# 데이터분석 모듈 - pandas

---

## ■ 인덱스 활용

○ set\_index() 메소드 사용: DataFrame 객체.set\_index(['열이름'] 또는 '열이름')

- 데이터프레임의 특정 열을 행 인덱스로 설정
- 단, 원본 데이터프레임을 바꾸지 않고 새로운 데이터프레임 객체를 반환함

```
new_df1 = df1.set_index(['seoul'])  
display(new_df1)  
new_df1.loc[110.0]
```

# 데이터분석 모듈 - pandas

---

## ○ 행 인덱스 재배열: reindex()

```
import pandas as pd
from IPython.display import display

date_index = pd.date_range('8/15/2020', periods=5, freq='D')
df1 = pd.DataFrame([100, np.nan, 80, 230, 100], columns=['prices'], index=date_index)
display(df1)

date_index2 = pd.date_range('8/13/2020', periods=8, freq='D')
df2 = df1.reindex(date_index2)#df1.reindex(index=date_index2),(columns=date_index2)
display(df2)

df2 = df1.reindex(date_index2, method='bfill')#(date_index2, fill_value=0)
display(df2)
df2.fillna(0)
```

# 데이터분석 모듈 - pandas

- 행 인덱스 초기화: `reset_index()`
  - 행 인덱스를 정수형 위치 인덱스로 초기화함
  - 이때 기존 행 인덱스는 열(컬럼)로 이동함
  - 새로운 데이터프레임 객체를 반환함

```
import pandas as pd
from IPython.display import display

df = pd.DataFrame({'month': [1, 3, 5, 7, 9, 11],
                   'year': ['2015', '2016', '2017', '2018', '2019', '2020'],
                   'sale': [300, 400, 200, 450, 500, 350]})
new_df = df.set_index('month')
display(new_df)
new_df1 = new_df.reset_index()
display(new_df1)

#행인덱스를 기준으로 데이터프레임 정렬
sort_df1 = new_df1.sort_index(ascending=False)
display(sort_df1)

#특정 열의 데이터값을 기준으로 데이터프레임 정렬
sort_value_df1 = new_df1.sort_values(by='sale', ascending=True)
display(sort_value_df1)
```

# 데이터분석 모듈 - pandas

## ■ 파일 입출력

- pandas.pydata.org 사이트 참조

파일 형식	읽어오기 함수	저장 함수
CSV	read_csv	to_csv
JSON	read_json	to_json
HTML	read_html	to_html
Local clipboard	read_clipboard	to_clipboard
MS Excel	read_excel	to_excel
HDF5 Format	read_hdf	to_hdf
SQL	read_sql	to_sql

- read\_csv(path, sep, header, index\_col, names, skiprows)
  - header - 열이름으로 사용될 행 번호(기본값은 0)
  - index\_col - 행인덱스로 사용할 열의 번호 또는 열 이름
  - names - 열 이름으로 사용할 문자열의 리스트

# 데이터분석 모듈 - pandas

## ■ 데이터 확인 작업

- head(), tail(), info(), describe(), min(), max(), mean(), median(), std(), corr() 등
- 판다스 내장 그래프 활용: plot() 메소드 적용, kind 속성 활용

kind 옵션	설명	kind 옵션	설명
'line'	선그래프	'kde'	커널밀도 그래프
'bar'	수직 막대	'area'	면적 그래프
'barh'	수평 막대	'pie'	파이 그래프
'his'	히스토그램	'scatter'	산점도 그래프
'box'	박스플롯	'hexbin'	고밀도 산점도

# 데이터분석 모듈 - pandas

---

- 통계청에서 남북한 발전 전력량을 다운(.xlsx)
- Colab을 사용한다면,
  - Google Drive에 My Drive 에 남북한발전전력량.xlsx 파일을 업로드 후
  - mount 을 시킨 다음(즉 colab과 google drive 를 연동 시킨 후) 파일 읽기 코드 작성(다음 코드는 google drive에 data 폴더를 생성 시킨 코드 임)

```
import pandas as pd
from google.colab import drive
drive.mount('/content/gdrive/', force_remount=True)

df = pd.read_excel('/content/gdrive/My Drive/data/남북한발전전력량.xlsx')
```

# 데이터분석 모듈 - pandas

## ■ Jupyter notebook 사용 시 xlsx 파일 읽기에서 실패 한 경우는

- pip, 또는 conda install xlrd 를 설치

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

```
df = pd.read_excel('D:/2020_Encore_교육/데이터분석및시각화/남북한발전전력량.xlsx')
```

```
df_ns = df.iloc[[0,5], 3:]
```

```
df_ns.index = ['South','North']
```

```
#df_ns.info()#
```

```
df_ns.columns = df_ns.columns.map(int)
```

```
arr = np.array(df_ns.loc[['South','North']]) #numpy의 where 등을 적용할 필요가 있을 때
```

```
df_ns.plot()
```

```
#행, 열 전치하여 다시 그리기
```

```
tdf_ns = df_ns.T
```

```
print(tdf_ns.head())
```

```
tdf_ns.plot()
```

```
#막대그래프
```

```
tdf_ns.plot(kind='bar')
```

```
#histogram
```

```
tdf_ns.plot(kind='hist')
```

```
#scatter
```

```
tdf_ns.plot(x='South',y='North', kind='scatter')
```

# 파이썬 데이터처리 및 분석

pandas 응용 - 함수매핑, 열 재구성, 피벗

---

2020.08. 19. 수요일

최희련

**En-CORE**

**Data Science Edu.**



# 데이터분석 모듈 - pandas

---

## ■ 함수 매핑

- 시리즈 또는 데이터프레임의 개별 원소를 특정 함수에 일대일 대응시키는 과정을 뜻함
- 사용자가 직접 만든 함수를 적용할 수 있으므로, 판다스 기본 함수로 처리하기 어려운 복잡한 연산을 적용할 수 있음

## ■ 각 객체에 함수 매핑

- apply 메소드 또는 pipe 메소드를 사용
- 시리즈 객체.apply(매핑 함수)
- 데이터프레임 객체. apply(매핑 함수)

# 데이터분석 모듈 - pandas

## ■ seaborn 패키지의 titanic 데이터 이용

```
import seaborn as sns
import pandas as pd
titanic = sns.load_dataset('titanic')
titanic.info(); titanic.head()
df = titanic.loc[:, ['age', 'fare']]
df['ten'] = 10
```

```
def add_10(n):
    return n+10
```

```
def add_two_(n,m):
    return n+m
```

```
def add_null(n):
    return n.isnull()
```

```
def min_max(n):
    return n.max() - n.min()
```

```
#Series 원소에 함수 적용
sr1 = df['age'].apply(add_10)
sr2 = df['age'].apply(add_two_, m=10)
sr3 = df['age'].apply(lambda x: add_10(x))
print(sr1, sr2, sr3)
```

```
#DataFrame 원소에 함수 적용, applymap 사용
df_map = df.apply(add_10)
df_map.head()
```

```
result = df.apply(add_null, axis=0) #axis의 default는 0
result.head()
```

```
min_max_result = df.apply(min_max)
min_max_result.head()
```

# 데이터분석 모듈 - pandas

## ■ 열 재구성

- 열 순서 변경: 데이터프레임 객체[재구성한 열 이름의 리스트]

```
df = titanic.loc[0:4, 'survived':'age']
df.head()
col_sort = sorted(df.columns.to_list())
df_sort = df[col_sort]
print(df_sort)
col_user = ['pclass', 'age', 'survived', 'sex']
df_user = df[col_user]
print(df_user)
```

- 열 분리

- 하나의 열이 여러 가지 정보를 담고 있는 경우
- 예를 들어 날짜의 컬럼 내의 연도/월/일의 정보를 분리하고 싶은 경우
- 기존의 열의 내용을 분리하여 새로운 컬럼으로 구성
- 명목데이터인 경우는 get\_dummies 메소드를 이용하여 원-핫 인코딩 함

```
pd.get_dummies(df['sex'])
```

# 데이터분석 모듈 - pandas

## ■ colab 사용 시

```
import pandas as pd
from google.colab import drive
drive.mount('/content/gdrive/', force_remount=True)

df = pd.read_excel('/content/gdrive/My Drive/data/주가데이터.xlsx')

df.info()
df.head()
```

### ○ 방법 1

```
df['연월일'] = df['연월일'].astype('str')
dates = df['연월일'].str.split('-')
dates.head()
df['년'] = dates.str.get(0)
df['월'] = dates.str.get(1)
df['일'] = dates.str.get(2)
```

# 데이터분석 모듈 - pandas

## ○ 방법 2

```
df['년'] = df['연월일'].astype(str).str[:4]  
df['월'] = df['연월일'].astype(str).str[5:7]  
df['일'] = df['연월일'].astype(str).str[8:]
```

## ○ 방법 3

```
df['연월일'] = pd.to_datetime(df['연월일'])  
df['년'] = df['연월일'].dt.year  
df['월'] = df['연월일'].dt.month  
df['일'] = df['연월일'].dt.day
```

```
df.drop(['연월일'], axis=1, inplace=True)
```

# 데이터분석 모듈 - pandas

---

## ■ 필터링

- 시리즈 또는 데이터프레임의 데이터 중에서 특정 조건식을 만족하는 원소만 따로 추출하는 개념
- boolean indexing, `isin()` 메소드의 방법 사용

## ■ boolean indexing

- 각 원소에 대한 참/거짓을 판별하여 불린값으로 구성된 객체를 반환

## ■ `isin()`

- 데이터프레임의 열에 `isin()` 메소드를 적용 → 특정 값을 가진 행들을 따로 추출할 수 있음, 조건에 해당되는 참값의 행들만 선택
- 데이터프레임의 열 객체.`isin(추출 값의 리스트)`

# 데이터분석 모듈 - pandas

---

```
import seaborn as sns
import pandas as pd
from IPython.display import display

titanic = sns.load_dataset('titanic')

#나이가 10대(10~19세)인 승객만 따로 선택하는 코드
mask1 = (titanic.age >= 10) & (titanic.age < 20)
print(mask1)
df_teenage = titanic.loc[mask1, :]
display(df_teenage)
```


```
#나이가 10대 미만이고, 성별이 여성인 승객만 따로 선택하는 코드
mask2 = (titanic.age < 10) & (titanic.sex == 'female')
df_female_under_10 = titanic.loc[mask2, :]#titanic.loc[mask2, ['survived','pclass','embark_town']]
display(df_female_under_10)
```

# 데이터분석 모듈 - pandas

```
import seaborn as sns
import pandas as pd
from IPython.display import display

titanic = sns.load_dataset('titanic')
pd.set_option('display.max_columns', 10) # 출력할 열의 개수 한도
# 형제 또는 배우자의 수가 3, 4, 5 명인 승객만 따로 선택하는 코드
mask3 = titanic['sibsp'] == 3
mask4 = titanic['sibsp'] == 4
mask5 = titanic['sibsp'] == 5

df_sibsp_boolean = titanic[mask3 | mask4 | mask5]
display(df_sibsp_boolean)
```



```
isin_mask = titanic['sibsp'].isin([3, 4, 5])
df_isin = titanic[isin_mask]
display(df_isin)
```



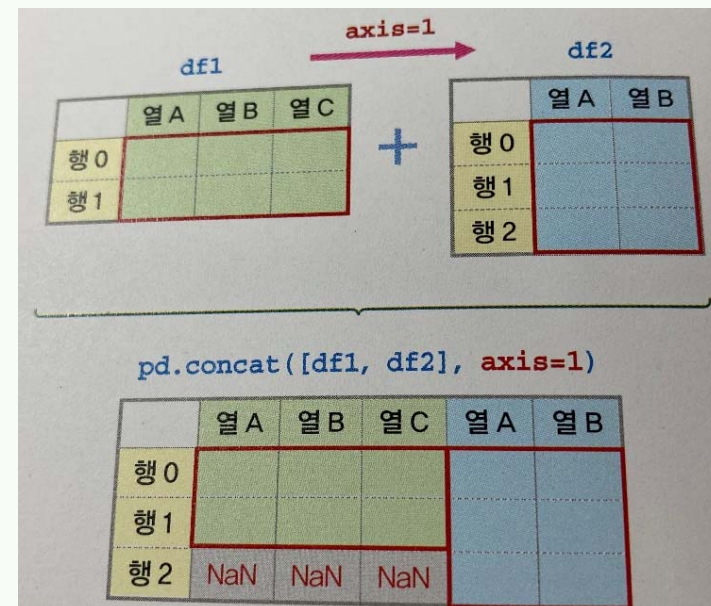
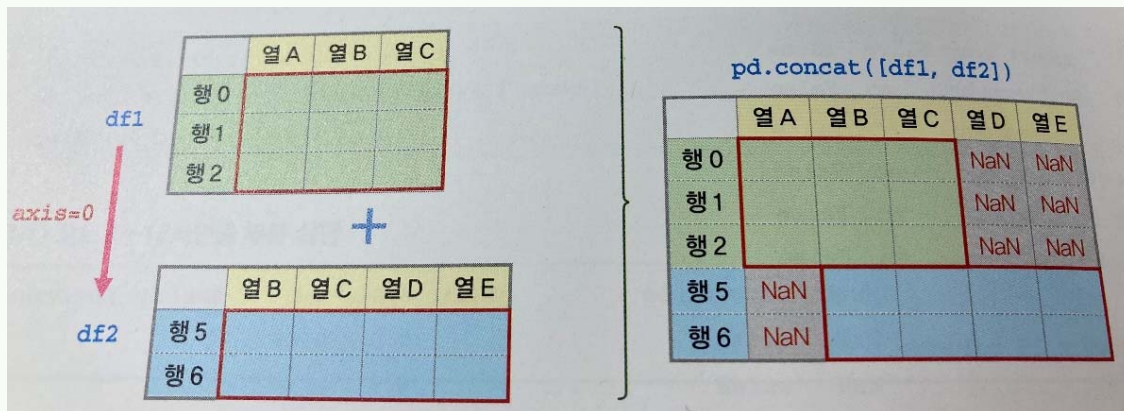
# 데이터분석 모듈 - pandas

## ■ 데이터프레임 합치기

- `concat()`, `merge()`, `join()` 등의 메소드 사용

## ■ `concat()`

- 서로 다른 데이터프레임들의 구성 형태, 속성이 균일한 경우
- 기존 데이터프레임의 형태를 유지하면서 이어 붙이는 개념
- `pandas.concat(데이터프레임의 리스트)`

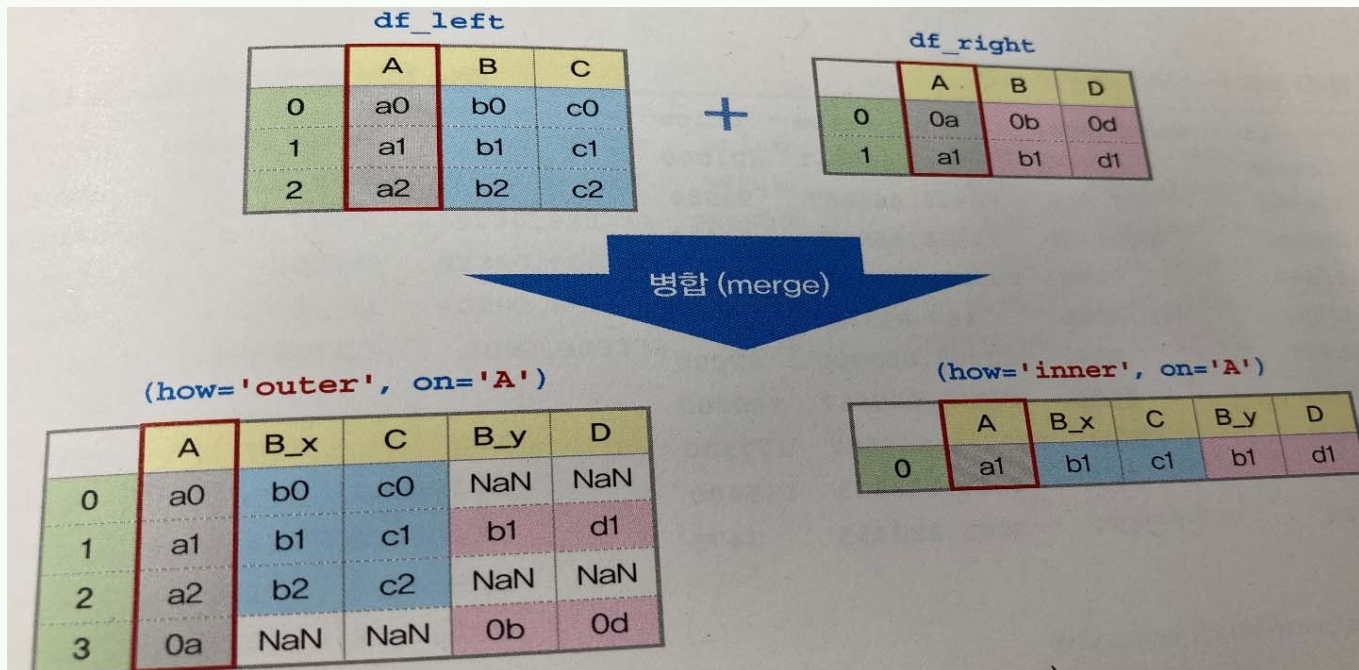


# 데이터분석 모듈 - pandas

## ■ merge()

- SQL의 join 과 비슷한 방식으로 어떤 기준에 의해 두 데이터프레임을 병합하는 개념
- 기준이 되는 열이나 인덱스를 키(key)라고 함, 이 키값은 병합하려는 두 데이터프레임에 모두 존재해야함
- pandas.merge(df\_left, df\_right, how=, on=)
  - how 는 병합 방식으로 'outer', '**inner**', 'left', 'right', 등이 있음
    - inner 는 기준이 되는 열의 데이터가 양쪽 데이터프레임에 공통으로 존재하는 교집합일 경우만 추출해서 병합
    - outer 는 기준되는 열의 데이터가 어느 한쪽에만 있어도 추출해서 병합, 기준이 되는 열의 데이터를 포함하는 모든 데이터가 병합, 어느 한 쪽이라도 데이터가 없는 열의 값은 NaN으로 채워짐
    - left는 df\_left에 속하는 데이터값을 기준으로 병합
    - right 는 df\_right에 속하는 데이터값을 기준으로 병합
  - on 은 key 값에 해당되는 열을 작성, default 는 **None**으로 두 데이터프레임의 공통으로 속하는 모든 열을 기준으로 병합한다는 의미임

# 데이터분석 모듈 - pandas



## ■ join()

- `merge()` 함수를 기반으로 만들어짐
- `merge`와의 차이점은 두 데이터프레임의 행 인덱스를 기준으로 결합함
- `join`의 속성 중 `on = key` 를 활용하면 열 기준으로 결합하는 것이 가능함
- `DataFrame1.join(DataFrame2, how='left')`

그림참조: 파이썬머신러닝 파다스데이터분석, 정보문화사

# 데이터분석 모듈 - pandas

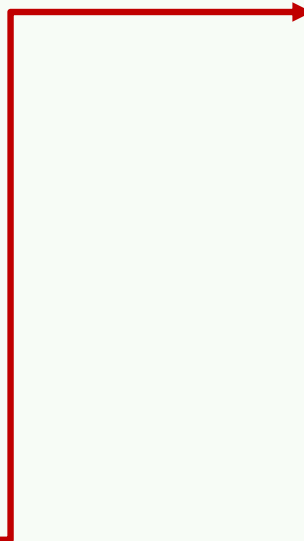
## ■ concat 예제

```
df1 = pd.DataFrame({'a': ['a0', 'a1', 'a2', 'a3'],  
                    'b': ['b0', 'b1', 'b2', 'b3'],  
                    'c': ['c0', 'c1', 'c2', 'c3']},  
                    index=[0, 1, 2, 3])
```


```
df2 = pd.DataFrame({'a': ['a2', 'a3', 'a4', 'a5'],  
                    'b': ['b2', 'b3', 'b4', 'b5'],  
                    'c': ['c2', 'c3', 'c4', 'c5'],  
                    'd': ['d2', 'd3', 'd4', 'd5']},  
                    index=[2, 3, 4, 5])
```

```
concat_df = pd.concat([df1, df2], axis = 0)
```

```
concat_df = pd.concat([df1, df2], axis = 0, ignore_index=True)
```



	a	b	c	d
0	a0	b0	c0	NaN
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	a3	b3	c3	NaN
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5



	a	b	c	d
0	a0	b0	c0	NaN
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	a3	b3	c3	NaN
4	a2	b2	c2	d2
5	a3	b3	c3	d3
6	a4	b4	c4	d4
7	a5	b5	c5	d5

# 데이터분석 모듈 - pandas

```
concat_df1 = pd.concat([df1,df2], axis = 1)  
display(concat_df1)
```

	a	b	c	a	b	c	d
0	a0	b0	c0	NaN	NaN	NaN	NaN
1	a1	b1	c1	NaN	NaN	NaN	NaN
2	a2	b2	c2	a2	b2	c2	d2
3	a3	b3	c3	a3	b3	c3	d3
4	NaN	NaN	NaN	a4	b4	c4	d4
5	NaN	NaN	NaN	a5	b5	c5	d5

```
concat_df1 = pd.concat([df1,df2], axis = 1, join=inner)  
display(concat_df1)
```

	a	b	c	a	b	c	d
2	a2	b2	c2	a2	b2	c2	d2
3	a3	b3	c3	a3	b3	c3	d3

```
concat_df = pd.concat([df1,df2], axis = 0, keys=['x','y'])
```

		a	b	c	d
x	0	a0	b0	c0	NaN
	1	a1	b1	c1	NaN
	2	a2	b2	c2	NaN
	3	a3	b3	c3	NaN
y	2	a2	b2	c2	d2
	3	a3	b3	c3	d3
	4	a4	b4	c4	d4
	5	a5	b5	c5	d5

# 데이터분석 모듈 - pandas

```
concat_df = pd.concat([df1,df2], axis = 0, verify_integrity=True)
```

```
-----TypeError Traceback (most recent call last)
<ipython-input-106-5041ef721b58> in <module>()
12 index=[2, 3, 4, 5])
13 --->
14 concat_df = pd.concat([df1,df2], axis = 0, verify_integrity=True)
TypeError: concat() got an unexpected keyword argument 'verify_integrity'
```

○ `verify_integrity` 는 합치는 열 또는 행의 인덱스가 동일여부를 파악,

- True 이면 동일여부에서 오류발생
- 이 속성을 이용하여 중복을 확인해 볼 수 있음

■ 함수 또는 메소드 등에 대한 도움말은 쥬피터 노트북에서

```
pandas.concat ? 또는 pd.concat?
```



# 데이터분석 모듈 - pandas

## merge 예제

```
import pandas as pd
from IPython.display import display
# 주식 데이터를 가져와서 데이터프레임 만들기
df1 = pd.read_excel('./stock price.xlsx')
df2 = pd.read_excel('./stock valuation.xlsx')
display(df1)
display(df2)
#eps:주당순이익, bps:주당순자산가치(기업순자산/발행주식수),
#per:주가수익비율(현재주가/eps),pbr:주가순자산비율(현재주가/pbs)
merge_inner = pd.merge(df1,df2)
display(merge_inner)
```



	id	stock_name	value	price	name	eps	bps	per	pbr
0	130960	CJ E&M	58540.666667	98900	CJ E&M	6301.333333	54068	15.695091	1.829178
1	139480	이마트	239230.833333	254500	이마트	18268.166667	295780	13.931338	0.860437
2	145990	삼양사	82750.000000	82000	삼양사	5741.000000	108090	14.283226	0.758627
3	185750	종근당	40293.666667	100500	종근당	3990.333333	40684	25.185866	2.470259
4	204210	모두투어리츠	3093.333333	3475	모두투어리츠	85.166667	5335	40.802348	0.651359

# 데이터분석 모듈 - pandas

```
# 데이터프레임 합치기 - 합집합
```

```
merge_outer = pd.merge(df1, df2, how='outer', on='id')  
display(merge_outer)  
print('\n')
```

```
# 데이터프레임 합치기 - 왼쪽 데이터프레임 기준, 키 값 분리
```

```
merge_left = pd.merge(df1, df2, how='left', left_on='stock_name', right_on='name')  
display(merge_left)  
print('\n')
```

```
# 데이터프레임 합치기 - 오른쪽 데이터프레임 기준, 키 값 분리
```

```
merge_right = pd.merge(df1, df2, how='right', left_on='stock_name', right_on='name')  
display(merge_right)  
print('\n')
```

## ○ left\_on, right\_on:

- 값이 일치해야할 왼쪽과 오른쪽 데이터프레임의 열을 지정
- 지정된 두 열의 값이 일치하면 how 의 기준으로 두 데이터프레임을 연결함



# 데이터분석 모듈 - pandas

```
#불린 인덱싱과 결합하여 원하는 데이터 찾기
price = df1[df1['price'] < 50000]
display(price.head())
print('\n')

value = pd.merge(price, df2)
display(value)
```

	id	stock_name	value	price
2	138250	엔에스쇼핑	14558.666667	13200
4	142280	녹십자엠에스	468.833333	10200
9	204210	모두투어리츠	3093.333333	3475

	id	stock_name	value	price	name	eps	bps	per	pbr
0	204210	모두투어리츠	3093.333333	3475	모두투어리츠	85.166667	5335	40.802348	0.651359

# 데이터분석 모듈 - pandas

## ■ join 예제

```
# 주식 데이터를 가져와서 데이터프레임 만들기
#행 인덱스를 기준으로 결합하므로 index_col의 속성을 통해 공통 인덱스를 정해 줌
df1 = pd.read_excel('./stock price.xlsx', index_col='id')
df2 = pd.read_excel('./stock valuation.xlsx', index_col='id')

# 데이터프레임 결합(join)
df3 = df1.join(df2)
print(df3)
print('\n')

# 데이터프레임 결합(join) - 교집합
df4 = df1.join(df2, how='inner')
print(df4)
```

# 데이터분석 모듈 - pandas

---

## ■ 그룹 연산

- 복잡한 데이터를 정해진 기준에 따라 여러 그룹으로 나누어 분석하는 방식
- 데이터를 집계, 변환 및 필터링하는데 효율적
- 3단계의 과정으로 이루어짐
  - 그룹 객체 만들기(분할단계): `groupby()`
  - 그룹 연산 메소드 (적용 단계): 데이터 집계에 필요한 메소드 적용
    - `mean, max, min, sum, count, size, var, std, describe`
    - `agg`
  - 그룹객체와 연산 메소드를 하나로 결합

# 데이터분석 모듈 - pandas

## ■ 예제

```
import pandas as pd
import seaborn as sns
from IPython.display import display
# titanic 데이터셋에서 age, sex 등 5개 열을 선택하여 데이터프레임 만들기
titanic = sns.load_dataset('titanic')
df = titanic.loc[:, ['age', 'sex', 'class', 'fare', 'survived']]

print('승객 수:', len(df))
display(df.head())
print('\n')

# class 열을 기준으로 분할
grouped = df.groupby(['class'])
print(grouped)
print('\n')

# 그룹 객체를 iteration으로 출력: head() 메소드로 첫 5행만을 출력
for key, group in grouped:
    print('* key :', key)
    print('* number :', len(group))
    display(group.head())
    print('\n')
```

# 데이터분석 모듈 - pandas

---

```
# 연산 메소드 적용
average = grouped.mean()
min_ = grouped.min()
display(average)
display(min_)
print('\n')

# 개별 그룹 선택하기
group3 = grouped.get_group('Third')
display(group3.head())
print('\n')
```

# 데이터분석 모듈 - pandas

## ■ 두 개의 기준으로 그룹

```
# class 열, sex 열을 기준으로 분할
grouped_two = df.groupby(['class', 'sex'])
```

```
# grouped_two 그룹 객체를 iteration으로 출력
for key, group in grouped_two:
    print('* key :', key)
    print('* number :', len(group))
    display(group.head())
    print('\n')
```

```
# grouped_two 그룹 객체에 연산 메소드 적용
average_two = grouped_two.mean()
display(average_two)
print('\n')
print(type(average_two))
```

```
# grouped_two 그룹 객체에서 개별 그룹 선택하기
group3f = grouped_two.get_group(('Third', 'female'))
display(group3f.head())
```

```
#각 열에 대한 집계값
average_age = grouped_two.age.mean()
display(average_age)
```

		age	fare	survived
class	sex			
First	female	34.611765	106.125798	0.968085
	male	41.281386	67.226127	0.368852
Second	female	28.722973	21.970121	0.921053
	male	30.740707	19.741782	0.157407
Third	female	21.750000	16.118810	0.500000
	male	26.507589	12.661633	0.135447

# 데이터분석 모듈 - pandas

---

```
grouped = df.groupby(['class'])
```

```
# 각 그룹에 대한 모든 열의 표준편차를 집계하여 데이터프레임으로 반환
```

```
std_all = grouped.std()
```

```
print(std_all)
```

```
print('\n')
```

```
print(type(std_all))
```

```
print('\n')
```

```
# 각 그룹에 대한 fare 열의 표준편차를 집계하여 시리즈로 반환
```

```
std_fare = grouped.fare.std()
```

```
print(std_fare)
```

```
print('\n')
```

```
print(type(std_fare))
```

```
print('\n')
```

# 데이터분석 모듈 - pandas

---

```
# 그룹 객체에 agg() 메소드 적용 - 사용자 정의 함수를 인수로 전달
def min_max(x): # 최대값 - 최소값
    return x.max() - x.min()

# 각 그룹의 최대값과 최소값의 차이를 계산하여 그룹별로 집계
agg_minmax = grouped.agg(min_max)
print(agg_minmax.head())
print('\n')

# 여러 함수를 각 열에 동일하게 적용하여 집계
agg_all = grouped.agg(['min', 'max'])
print(agg_all.head())
print('\n')

# 각 열마다 다른 함수를 적용하여 집계
agg_sep = grouped.agg({'fare':['min', 'max'], 'age':'mean'})
print(agg_sep.head())
```



# 데이터분석 모듈 - pandas

## ■ 그룹 객체에 filter 함수 적용

```
# class 열을 기준으로 분할
grouped = df.groupby(['class'])

# 데이터 개수가 200개 이상인 그룹만을 필터링하여 데이터프레임으로 반환
grouped_filter = grouped.filter(lambda x: len(x) >= 200)
display(grouped_filter.head())
print('\n')
print(type(grouped_filter))

# age 열의 평균이 30보다 작은 그룹만을 필터링하여 데이터프레임으로 반환
age_filter = grouped.filter(lambda x: x.age.mean() < 30)
display(age_filter.tail())
print('\n')
print(type(age_filter))

# 30이하의 평균 및 30이하의 group을 성별로 다시 재그룹
print(age_filter.mean())
age_filter_sex = age_filter.groupby(['sex'])
display(age_filter_sex.mean())
```

```
<class 'pandas.core.frame.DataFrame'>
age      26.692708
fare     15.580055
survived   0.305185
dtype: float64
```

	age	fare	survived
sex			
female	24.681818	18.140172	0.645455
male	27.698153	14.342196	0.140659

# 데이터분석 모듈 - pandas

---

## ■ 피벗(pivot)

### ○ 엑셀의 피벗 테이블과 비슷한 기능

- 대화형 테이블의 일종으로, 데이터의 나열 형태에 따라서 집계나 카운트 등의 계산을 하는 기능

### ○ 4가지 요소로 구성

- 행 인덱스, 열 인덱스, 데이터 값, 데이터 집계 함수

### ○ 4가지 요소에 적용할 데이터프레임의 열을 각각 지정하여 함수의 인자로 전달하는 형식

### ○ pivot\_table 메소드 사용

- `pandas.pivot_table(데이터프레임, index=[열이름], values=[열이름], aggfunc = 집계함수)`

\*\* 예를 들어 타이타닉에서 객실 클래스와 생존자와 관련된 데이터셋을 출력하고 객실 클래스로 그룹을 만들고, 그 안에 입력될 값은 평균이며, 데이터의 정렬은 생존자로 내림정렬하여 분석

# 데이터분석 모듈 - pandas

## ■ 피벗 예

```
import pandas as pd
import seaborn as sns
from IPython.display import display
```

```
# titanic 데이터셋에서 age, sex 등 5개 열을 선택하여 데이터프레임 만들기
titanic = sns.load_dataset('titanic')
df = titanic.loc[:, ['age', 'sex', 'class', 'fare', 'survived']]
display(df.head())
print('\n')
```

```
# 행, 열, 값, 집계에 사용할 열을 1개씩 지정 - 평균 집계
pdf1 = pd.pivot_table(df,          # 피벗할 데이터프레임
                      index='class', # 행 위치에 들어갈 열, 'sex' 를 대입
                      values='survived', # 데이터로 사용할 열
                      aggfunc='mean') # 데이터 집계 함수
```

```
display(pdf1.head())
print('\n')
```

	age	sex	class	fare	survived
0	22.0	male	Third	7.2500	0
1	38.0	female	First	71.2833	1
2	26.0	female	Third	7.9250	1
3	35.0	female	First	53.1000	1
4	35.0	male	Third	8.0500	0

survived	
class	
First	0.629630
Second	0.472826
Third	0.242363

survived	
sex	
female	0.742038
male	0.188908

# 데이터분석 모듈 - pandas

```
pdf1 = pd.pivot_table(df,          # 피벗할 데이터프레임
                      index='class', # 행 위치에 들어갈 열
                      columns='sex',  # 열 위치에 들어갈 열
                      values='age',   # 데이터로 사용할 열
                      aggfunc='mean') # 데이터 집계 함수
```

```
display(pdf1.head())
print('\n')
```

# 값에 적용하는 집계 함수를 2개 이상 지정 가능 - 생존율, 생존자 수 집계

```
pdf2 = pd.pivot_table(df,          # 피벗할 데이터프레임
                      index='class', # 행 위치에 들어갈 열
                      columns='sex',  # 열 위치에 들어갈 열
                      values='survived', # 데이터로 사용할 열
                      aggfunc=['mean', 'sum']) # 데이터 집계 함수
```

```
display(pdf2.head())
print('\n')
```

sex	female	male
class		
First	34.611765	41.281386
Second	28.722973	30.740707
Third	21.750000	26.507589

sex	mean		sum	
	female	male	female	male
class				
First	0.968085	0.368852	91	45
Second	0.921053	0.157407	70	17
Third	0.500000	0.135447	72	47

# 데이터분석 모듈 - pandas

## ■ 행, 열, 값에 사용할 열을 2개 이상 지정 예

```
# 행, 열, 값에 사용할 열을 2개 이상 지정 가능 - 평균 나이, 최대 요금 집계
pdf3 = pd.pivot_table(df,                                # 피벗할 데이터프레임
                      index=['class', 'sex'],            # 행 위치에 들어갈 열
                      columns='survived',                # 열 위치에 들어갈 열
                      values=['age', 'fare'],            # 데이터로 사용할 열
                      aggfunc=['mean', 'max'])           # 데이터 집계 함수
```

```
# IPython Console 디스플레이 옵션 설정
```

```
pd.set_option('display.max_columns', 10)    # 출력할 열의 개수 한도
display(pdf3.head())
print('\n')
```

```
# 행, 열 구조 살펴보기
```

```
print(pdf3.index)
print(pdf3.columns)
print('\n')
```

		mean				max				Level 0
		age		fare		age		fare		Level 1
survived	class	0	1	0	1	0	1	0	1	Level 2
		sex								
First	female	25.666667	34.939024	110.604167	105.978159	50.0	63.0	151.55	512.3292	
	male	44.581967	36.248000	62.894910	74.637320	71.0	80.0	263.00	512.3292	
Second	female	36.000000	28.080882	18.250000	22.288989	57.0	55.0	26.00	65.0000	
	male	33.369048	16.022000	19.488965	21.095100	70.0	62.0	73.50	39.0000	
Third	female	23.818182	19.329787	19.773093	12.464526	48.0	63.0	69.55	31.3875	

# 데이터분석 모듈 - pandas

## ■ 피벗테이블 객체.xs

- 피벗테이블의 행을 선택하기 위해 사용
- xs(행인덱서), axis 는 자동으로 0 이 지정
- xs(열인덱서, axis=1)

```
mean age 0 25.666667
      1 34.939024
      fare 0 110.604167
           1 105.978159
max age 0 50.000000
      1 63.000000
      fare 0 151.550000
           1 512.329200
Name: (First, female), dtype: float64
```

```
# xs 인덱서 사용 - 행 선택(default: axis=0)
print(pdf3.xs('First'))          # 행 인덱스가 First인 행을 선택
print('\n')
print(pdf3.xs(('First', 'female'))) # 행 인덱스가 ('First', 'female')인 행을 선택
print('\n')
print(pdf3.xs('male', level='sex')) # 행 인덱스의 sex 레벨이 male인 행을 선택
print('\n')
print(pdf3.xs(('Second', 'male'), level=[0, 'sex'])) # Second, male인 행을 선택
print('\n')
```

# 데이터분석 모듈 - pandas

## ■ xs의 속성의 level

- pivot table의 깊이를 의미함

```
print(pdf3.xs(('Second', 'male'))) # Second, male인 행을 선택
```

```
mean age 0 33.369048
      1 16.022000
      fare 0 19.488965
          1 21.095100
max age 0 70.000000
      1 62.000000
      fare 0 73.500000
          1 39.000000
Name: (Second, male), dtype: float64
```

```
print(pdf3.xs(('Second', 'male'), level=['class', 'sex'])) # Second, male인 행을 선택
```

```
              mean                max
              age                fare age                fare
survived      0      1      0      1      0      1      0      1
class sex
Second male 33.369048 16.022 19.488965 21.0951 70.0 62.0 73.5 39.0
```

# 데이터분석 모듈 - pandas

---

```
#xs 인덱서 사용 - 열 선택(axis=1 설정)
print(pdf3.xs('mean', axis=1))      # 열 인덱스가 mean인 데이터를 선택
print('\n')
print(pdf3.xs(('mean', 'age'), axis=1)) # 열 인덱스가 ('mean', 'age')인 데이터 선택
print('\n')
print(pdf3.xs(1, level='survived', axis=1)) # survived 레벨이 1인 데이터 선택
print('\n')
print(pdf3.xs(('max', 'fare', 0),
              level=[0,1,2], axis=1)) # max, fare, survived=0인 데이터 선택
```



# 데이터분석 모듈 - pandas

- 다음 데이터를 보고 분석에 필요한 pivot 테이블을 작성

```
import datetime
import numpy as np
import pandas as pd
from IPython.display import display
np.random.seed(777)

df = pd.DataFrame({'A': ['one', 'one', 'two', 'three'] * 6,
                   'B': ['A', 'B', 'C'] * 8,
                   'C': ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 4,
                   'D': np.random.randn(24),
                   'E': np.random.randn(24),
                   'F': [datetime.datetime(2013, i, 1) for i in range(1, 13)]
                      + [datetime.datetime(2013, i, 15) for i in range(1, 13)]})

display(df)
```

- one 클래스이면서 등급이 A이고 'foo'에 해당되는 D의 평균 및 합산은?

# 데이터 분석 실습

- 타이타닉 데이터에서 승선한 지역의 이름을 중복없이 출력하고, 가장 빈도가 높은 지역은?

```
embark = titanic['embark_town'].unique()
embark_count = titanic['embark_town'].value_counts()
display(embark_count)
print(embark, embark_count.index[0])
```

- 만약 데이터에 결측치가 있는 경우, 수행해야하는 시도는?
  - 데이터 프레임에 컬럼별 결측치가 얼마나 있는지 확인
  - 결측치 발생 시 다른값으로 치환 또는 수가 적어서 전체 분석에 영향이 없다면, 삭제 시도
  - 다른값으로 치환 시에는 최빈값, 평균값 또는 바로앞, 바로뒤의 값등으로 대체
  - 또는 smote와 k-NN 같은 방법을 사용해서 근사한 추정값으로 값을 대체
  - 이러한 테스트는 잘 알려진 titanic 등과 같은 데이터를 활용해 볼 수 있음

# 데이터 분석 실습

---

- Titanic에서 age, fare 컬럼에서 결측치를 확인하는 코드 작성

- Titanic에서 각 열의 누락데이터가 몇 건이 있는지 확인

# 데이터 분석 실습

---

- 타이타닉의 나이데이터의 결측치를 나이데이터의 중앙값으로 교체
- 위의 df 에 저장된 타이타닉의 컬럼명을 abcd 순서대로 출력