2. Pandas

2.1.Pandas 기본 개념

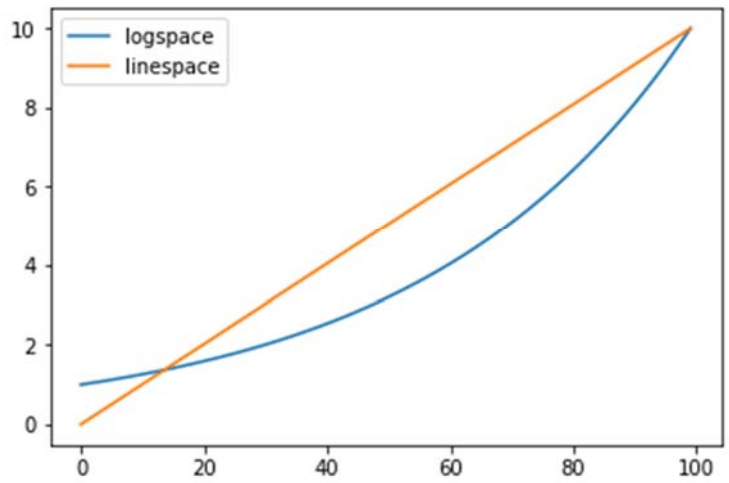
2.1.1. Pandas 구조

2.1.1.1. Pandas

- Numpy 기반으로 개발되어 고성능 데이터 분석 가능
- R언어에서 제공하는 DataFrame 자료형 제공
- 명시적으로 축의 이름에 따라 데이터 정렬 가능한 자료구조
- 통합된 Time Series 분석 기능
- 누락된 데이터를 유연하게 처리할 수 있는 기능
- 구조화된데이터의처리를지원하는Python 라이브러리

```
1  import numpy as np
2  import pandas as pd
3
4  logx = np.logspace(0,1,100)
5  linex = np.linspace(0,10,100)
6
7  df = pd.DataFrame() # pandas DataFrame 생성
8  df['logspace'] = logx
9  df['linespace'] = linex
10 df.head() #  top 5
11 df.plot()
```

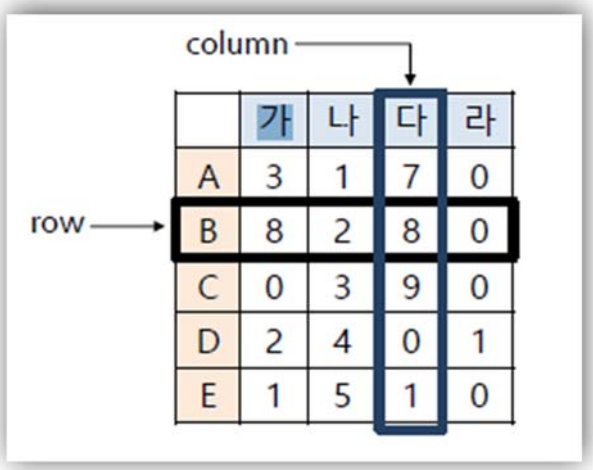
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ec32ea8fd0>
```



https://www.slideshare.net/TaeYoungLee1/1-115587182?from_action=*save*

2.1.1.2. DataFrame

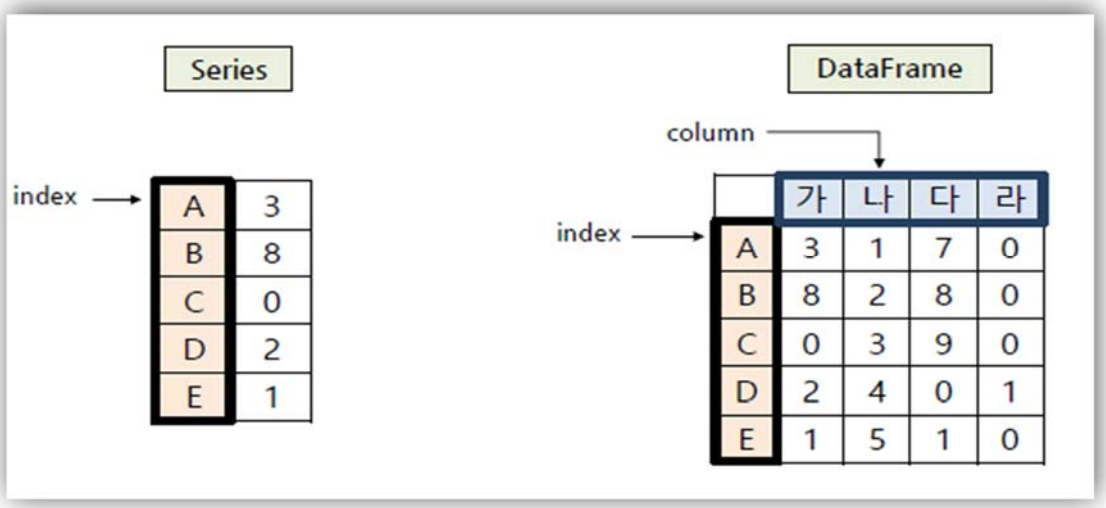
- 레이블(Labeled)된 행(Column)과 열(Row)을 가진 2차원 데이터구조
- 칼럼 마다 데이터 형식이 다를 수 있음
- 각 행(Column)과 열(Row)들을 산술연산이 가능한 구조
- DataFrame 크기는 유동적으로 변경 가능
- DataFrame끼리 여러 가지 조건을 사용한 결합처리 가능



https://www.slideshare.net/TaeYoungLee1/1-115587182?from_action=save

2.1.1.3. Series와 DataFrame

- Series : 인덱스 라벨이 붙은 1차원 리스트 데이터 구조
- DataFrame : Series가 모인 2차원 테이블 데이터 구조
- DataFrame이 핵심이고, Series는 개념으로 알아두자!



https://www.slideshare.net/TaeYoungLee1/1-115587182?from_action=*save*

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	1

Series

DataFrame 중 하나의 Column에 해당하는 데이터의 모음 Object

DataFrame

Data Table 전체를 포함하는 Object

2.1.2. Pandas 데이터 로딩

2.1.2.1. CSV

read_csv

➔ 대부분의 경우 CSV를 읽어 드려 사용경우가 대부분

```
%%writefile sample1.csv # 텍스트 파일 쓰기
c1, c2, c3
1, 1.11, one
2, 2.22, two
3, 3.33, three
```

```
Writing sample1.csv
```

```
pd.read_csv('sample1.csv') # csv 파일 읽어오기
```

	c1	c2	c3
0	1	1.11	one
1	2	2.22	two
2	3	3.33	three

```
1
2 data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/h
3 # data_url = './housing.data' #Data URL
4 df_data = pd.read_csv(data_url, sep='\s+', header = None) #csv 타입 데
```

```
1 df_data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

2.1.2.2. Web

pandas_datareader 패키지의 `DataReader` 을 사용하면 인터넷 사이트의 자료를 바로 pandas 로딩

```
1
```

```
1 import pandas_datareader as pdr
2 import datetime
3
4 dt_start = datetime.datetime(2015, 1, 1)
5 dt_end = "2016, 6, 30"
6 gdp = pdr.get_data_fred('GDP', dt_start, dt_end)
7 gdp.head()
```

	GDP
DATE	
2015-01-01	17970.422
2015-04-01	18221.299
2015-07-01	18331.093
2015-10-01	18354.372
2016-01-01	18409.130

2.1.2.3. 파이썬 자료구조 로딩

```
ds = [ {'고객':'A', '금액':1000, '가맹점':'0001', '카드':'9440'},
       {'고객':'B', '금액':2000, '가맹점':'0002', '카드':'4805'},
       {'고객':'C', '금액':5000, '가맹점':'0003', '카드':'4602'} ]
df = pd.DataFrame(ds)
df
```

	가맹점	고객	금액	카드
0	0001	A	1000	9440
1	0002	B	2000	4805
2	0003	C	5000	4602

list

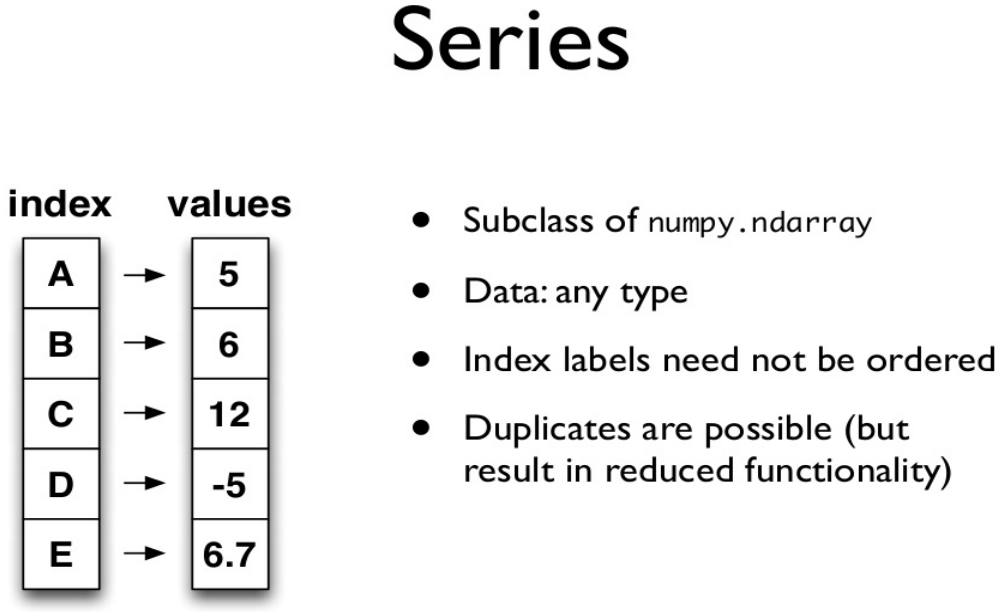
```
ds = [['A', 1000, '0001', '9440'],
      ['B', 2000, '0002', '4805'],
      ['C', 5000, '0003', '4602']]
lb = ['고객', '금액', '가맹점', '카드']
df = pd.DataFrame(ds, columns=lb)
df
```

	고객	금액	가맹점	카드
0	A	1000	0001	9440
1	B	2000	0002	4805
2	C	5000	0003	4602

```
ds = { '고객'   :['A','B','C'],
       '금액'   :[1000, 2000, 5000],
       '가맹점' :['0001', '0002', '0003'],
       '카드'   :['9440', '4805', '4602'] }
df = pd.DataFrame.from_dict(ds)
df
```

2.2.pandas 객체

2.2.1. Series 객체



https://www.slideshare.net/wesm/pandas-powerful-data-analysis-tools-for-python

```
1 data = pd.Series([0.25, 0.5, 0.75, 1])
2 data
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

```
1 data.values # numpy 배열
```

```
array([0.25, 0.5 , 0.75, 1.  ])
```

```
1 data.index # index 객체
```

```
RangeIndex(start=0, stop=4, step=1)
```

인덱스 설정

```
data = pd.Series([1,2,3] , index=['one','two','three'])
data
```

```
one      1
two      2
three    3
dtype: int64
```

```
data = pd.DataFrame([1,2,3], index=['one','two','three'],columns=['no'])
data
```

	no
one	1
two	2
three	3

딕셔너리 와 Series

```
score_dict = { 'c' : 80 , 'c++': 90 , 'java': 100 }
score = pd.Series(score_dict)
score
```

```
c       80
c++     90
java     A
dtype: object
```

```
score['c':'java'] # 슬라이싱
```

```
c       80
c++     90
java     A
dtype: object
```

시리즈의 연산에서 index매칭이 되지 않으면 NaN

```
subject = ['c' , 'c++', 'python', 'matlab']
score2 = Series(score, index = subject)
score2
```

```
c         80.0
c++       90.0
python     NaN
matlab     NaN
dtype: float64
```

```
score + score2  # java ??
```

```
c         160.0
c++       180.0
java        NaN
matlab      NaN
python      NaN
dtype: float64
```

아래 예제는 Series Data로 …

```
data['one'] #data index  접근
```

```
1
```

```
data["two"] = 22;
```

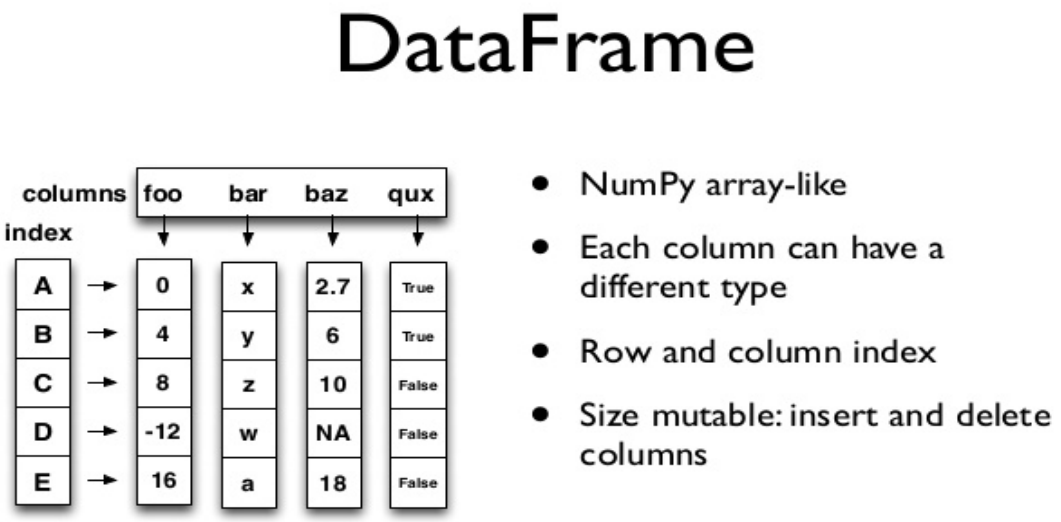
```
data.name = "Eng Dic"
data.index.name = '1!2!3!'
data
```

```
1!2!3!
one       1
two      22
three     3
Name: Eng Dic, dtype: int64
```

2.2.2. DataFrame 객체

Series를 모아서 만든Data Table → 2차원 형태의 테이블 구조

- 엑셀 sheet
- DBMS table , R data.frame



-

https://www.slideshare.net/wesm/pandas-powerful-data-analysis-tools-for-python

```
1 import pandas as pd
2 from pandas import Series , DataFrame
3
4 a = pd.DataFrame([
5     [1,2,3],
6     [4,5,6],
7     [7,8,9]])
8 a
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Dict 객체를 이용한 DataFrame 생성

```
data = {
    'subject':['c' , 'c++' , 'java' , 'python'],
    'score'  :[ 88 , 50 , 30 , 100]
}
df = pd.DataFrame(data)
df
```

	subject	score
0	c	88
1	c++	50
2	java	30
3	python	100

```
df = DataFrame(data , columns=['score', 'subject'])
df  # column 변경
```

	score	subject
0	88	c
1	50	c++
2	30	java
3	100	python

```
df = DataFrame(data , columns=['score', 'subject' , 'count'])
df  # column 추가
```

	score	subject	count
0	88	c	NaN
1	50	c++	NaN
2	30	java	NaN
3	100	python	NaN

```
df = DataFrame(data ,index = [1,2,3,4])
df  # index 변경
```

	subject	score
1	c	88
2	c++	50
3	java	30
4	python	100

```
#Example from - https://chrisalbon.com/python/pandas_map_values_to_valu
data = pd.read_csv('data/titanic.csv', sep = '\t')
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

2.2.2.1. DataFrame Columns

컬럼명으로 데이터 가져오기

```
DataFrame( data ,columns=['sex', 'city']).head() # 데이터 불러오기
```

	sex	city
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN

만약 컬럼이 존재하지 않을 경우 새로운 컬럼을 생성하고 NaN으로 초기화

```
df = DataFrame( data ,columns=['sex', 'city', 'salary']).head() # 컬럼
```

	sex	city	salary
0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	NaN	NaN	NaN

컬럼명을 통해 각각의 Series가져오기

```
df.last_name  # 컬럼명으로 Series 가져오기
```

```
0      Miller
1    Jacobson
2         Ali
3      Milner
4       Cooze
Name: last_name, dtype: object
```

```
df['last_name'] #컬럼명으로 Series 가져오기
```

```
0      Miller
1    Jacobson
2         Ali
3      Milner
4       Cooze
Name: last_name, dtype: object
```

Column명 변경

DataFrame의 rename

pd.DataFrame.columns = ['칼럼1', '칼럼2', … ,]

pd.DataFrame.rename({ 'OLD칼럼명' : 'NEW칼럼명', …, }, axis=1)

pd.DataFrame.rename(columns={ 'OLD칼럼명' : 'NEW칼럼명', …, })

```
df.rename( columns={'salary':'sal'} )
```

	sex	city	sal
0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	NaN	NaN	NaN

T

```
data = {
    'subject':['c' , 'c++' , 'java' , 'python'],
    'score'  :[ 88 , 50 , 30 , 100]
}
df = pd.DataFrame(data)
df
```

	subject	score
0	c	88
1	c++	50
2	java	30
3	python	100

```
df.T
```

	0	1	2	3
score	88	50	30	100
subject	c	c++	java	python

2.2.3. Index 객체

2.2.3.1. Index

- 불변의 배열 , 정렬된 집합
- Series와 DataFrame객체가 데이터를 참조하고 수정하게 해주는 역할
- Pandas index객체는 표형식의 데이터에서 각행과 열에 대한 헤더(이름)
- 메타데이터(축의 이름)을 저장하는 객체
- Series DataFrame객체 생성시 사용되는 배열이나 순차적인 이름은 내부적으로 indexing

```
idx = pd.Index([1,2,3,4,5])
idx
```

```
Int64Index([1, 2, 3, 4, 5], dtype='int64')
```

```
idx[0]
```

```
1
```

```
idx[::2] # 파이썬 표준 인덱싱 표기법
```

```
Int64Index([1, 3, 5], dtype='int64')
```

```
print(  idx.size, idx.shape, idx.ndim, idx.dtype) # numpy ??
```

```
5 (5,) 1 int64
```

```
idx[1] = 123
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-f518e69fe2bb> in <module>
----> 1 idx[1] = 123

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in __setitem__(self, key,
 value)
   2063
   2064     def __setitem__(self, key, value):
-> 2065         raise TypeError("Index does not support mutable operations")
   2066
   2067     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

인덱스는 변경 불가능한 배열 !!

2.2.3.2. Reindex

```
df = pd.DataFrame( np.arange(9).reshape(3,3),
                   index = [1,3,5] , columns=['x', 'y', 'z'])
df
```

	x	y	z
1	0	1	2
3	3	4	5
5	6	7	8

```

```

```
df2 = df.reindex([1,2,3,4] ,  method = 'ffill') # 4 추가 NaN
df2
```

	x	y	z
1	0	1	2
2	0	1	2
3	3	4	5
4	3	4	5

row 보간 가능

```
col = ['x', 'y' ,'v' ,'z']
df2.reindex(columns=col , method = 'ffill')    # colums reindex
```

	x	y	v	z
1	0	1	NaN	2
2	0	1	NaN	2
3	3	4	NaN	5
4	3	4	NaN	5

데이터 프레임의 보간은 row만 가능

、

2.3. 데이터 선택하기

2.3.1. Series Selection

Series.의 인덱스는 딕셔너리 타입과 유사하게 동작
딕셔너리 인터페이를 통해 Series 인덱스 조작

컬럼 이름으로 선택하기

```
1  from pandas import Series, DataFrame
2  #Example from - https://chrisalbon.com/python/pandas_map_values_to_valu
3  df = pd.read_csv('data/titanic.csv', sep = '\t')
4  df['Age'].head(3) # 1개의 컬럼 선택시 Series 객체 반환
```

```
0    22.0
1    38.0
2    26.0
Name: Age, dtype: float64
```

```
1  df[[ 'Name', 'Sex','Age' ]].head(3)
```

	Name	Sex	Age
0	Braund, Mr. Owen Harris	male	22.0
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	Heikkinen, Miss. Laina	female	26.0

1개이상의 컬럼이 선택될경우 DataFrame객체 반환

Index로 선택하기 : column명이 없으면 row 전체

```
df[:3] # row num
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	

```
df['Name'][:3] # Name 컬럼만 적용
```

```
0                              Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
2                               Heikkinen, Miss. Laina
Name: Name, dtype: object
```

Series 선택하기

```
import numpy as np
import pandas as pd
data = pd.Series([0.25, 0.5, 0.75, 1] , index = ['a', 'b' ,'c','d'])
data
```

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

```
data['b'] # key (index)값을 통해 접근
```

```
0.5
```

```
'a' in data  # 키/인덱스 값을 조사
```

```
True
```

```
data.keys()
```

```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
list(data.items()) #  딕셔너리와 유사한 인터페이스와 동작
```

```
[('a', 0.25), ('b', 0.5), ('c', 0.75), ('d', 1.0)]
```

주의 !!

```
Age_Series = df['Age']
Age_Series[1:10:2].head(3)
```

```
1    38.0
3    35.0
5     NaN
Name: Age, dtype: float64
```

```
Age_Series[[1,10,2]].head(3)
```

```
1     38.0
10     4.0
2     26.0
Name: Age, dtype: float64
```

Series 인덱서 (loc , iloc)

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

```
1    a
2    b
3    c
dtype: object
```

```
data[1]  #  명시적인 인데서 사용
```

```
'a'
```

```
data[1:3] # ?? 머가 나오나 ??
```

```
2    b
3    c
dtype: object
```

```
data.loc[1]  # 1에 해당하는 값이 나온다
```

```
'a'
```

```
data.iloc[1] #  2번째 값이 나온다.
```

```
'b'
```

```
data.loc[1:3]
```

```
1    a
2    b
3    c
dtype: object
```

```
data.iloc[1:3]
```

```
2    b
3    c
dtype: object
```

Series : 1차원 배열

```
data['a':'c'] # 명시적 인덱스로 슬라이싱
```

```
a    0.25
b    0.50
c    0.75
dtype: float64
```

```
data[0:2] # 암시적 정수값으로 슬라이싱
```

```
a    0.25
b    0.50
dtype: float64
```

```
data[ (data>0.3) & (data < 0.8) ] # 마스킹
```

```
b    0.50
c    0.75
dtype: float64
```

```
data[[ 'a' , 'd' ]] # 팬시 인덱싱
```

```
a    0.25
d    1.00
dtype: float64
```

```
data [ data <= 0.5 ] # boolean index
```

```
a    0.25
b    0.50
dtype: float64
```

Name 컬럼을 index로 지정하고 df에서 삭제

```
import pandas as pd
from pandas import Series, DataFrame
#Example from - https://chrisalbon.com/python/pandas_map_values_to_valu
df = pd.read_csv('data/titanic.csv', sep = '\t')
```

```
df.index = df['Name']  # 인덱스 변경
del df['Name']         #열 삭제
df.head()
```

Name	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
Braund, Mr. Owen Harris	1	0	3	male	22.0	1	0	A/5 21171	7.2500	Nal
Cumings,										

Loc : 인덱스 값 iloc: 순서

```
df[['Sex', 'Age']][:2]  # 컬럼 + 인덱스
```

Name	Sex	Age
Braund, Mr. Owen Harris	male	22.0
Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.0

```
df.loc['Braund, Mr. Owen Harris', 'Age']
```

22.0

```
df.iloc[:4,:3] # column no + index no
```

Name	PassengerId	Survived	Pclass
Braund, Mr. Owen Harris	1	0	3
Cumings, Mrs. John Bradley (Florence Briggs Thayer)	2	1	1
Heikkinen, Miss. Laina	3	1	3
Futrelle, Mrs. Jacques Heath (Lily May Peel)	4	1	1

2.3.2. Data drop

```
1 df.drop(1).head(2) # index no
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarke
0	1	0	3	male	22.0	1	0	A/5 21171	7.250	NaN	
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.925	NaN	

```
1 df.drop([1,3,5,7]).head(3) # 여러개
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarke
0	1	0	3	male	22.0	1	0	A/5 21171	7.250	NaN	
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.925	NaN	
4	5	0	3	male	35.0	0	0	373450	8.050	NaN	

```
1 df.drop('Fare', axis=1) # axis 기준으로 삭제
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Cabin	Embarked
0	1	0	3	male	22.0	1	0	A/5 21171	NaN	S
1	2	1	1	female	38.0	1	0	PC 17599	C85	C

2.4.데이터프레임 연산

2.4.1. Series 객체 연산

Index가 다른 series 객체 연산

Index가 매칭되는 값만 연산. 매칭되지 않으면 NaN

```
import pandas as pd
import numpy as np
from pandas import Series, DataFrame
# 시리즈  연산
s1 = Series([1,2,3,4], index = ['a', 'b','c','d'])
s2 = Series([4,3,2,1], index = ['a', 'b','e','f'])
s1 + s2  # index 같은 것끼지 덧셈
```

```
a    5.0
b    5.0
c    NaN
d    NaN
e    NaN
f    NaN
dtype: float64
```

DataFrame 연산

- 같은 자리의 값 덧셈

```
df1 = DataFrame(np.arange(9).reshape(3,3) ,  columns=['서울','대전','대구'] ,index = ['a', 'b','c'])
df2 = DataFrame(np.arange(9).reshape(3,3) ,  columns=['서울','대전','대구'] ,index = ['a', 'b','c'])

print(df1)
print(df2)
print(df1 + df2)

```

```
   서울  대전  대구
a   0   1   2
b   3   4   5
c   6   7   8
   서울  대전  대구
a   0   1   2
b   3   4   5
c   6   7   8
   서울  대전  대구
a   0   2   4
b   6   8  10
c  12  14  16
```

2.4.2. DataFrame 연산

두개에 dataframe에 매칭되는 컬럼이 없는 경우
DataFrame객체는 column과 index를 모두 고려하여 처리

```
df1 = DataFrame(np.arange(9).reshape(3,3) ,  columns=['서울','대전','대구'] ,index = ['a', 'b','c'])
df2 = DataFrame(np.arange(9).reshape(3,3) ,  columns=['서울','대전','부산'] ,index = ['a', 'b','d'])
print(df1)
print(df2)
print(df1 + df2)
```

```
   서울  대전  대구
a   0   1   2
b   3   4   5
c   6   7   8
   서울  대전  부산
a   0   1   2
b   3   4   5
d   6   7   8
   대구   대전  부산   서울
a NaN  2.0 NaN  0.0
b NaN  8.0 NaN  6.0
c NaN  NaN NaN  NaN
d NaN  NaN NaN  NaN
```

컬럼이 다를경우 매칭되지 않는 컬럼은 NaN

```
df1 = DataFrame(np.arange(9).reshape(3,3) ,  columns=['a','b','c'])
df2 = DataFrame(np.arange(12).reshape(3,4) ,  columns=['a','b','c','d'])
print(df1)
print(df2)
print(df1 + df2)
```

```
   a  b  c
0  0  1  2
1  3  4  5
2  6  7  8
   a  b   c   d
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
    a   b   c   d
0   0   2   4 NaN
1   7   9  11 NaN
2  14  16  18 NaN
```

Add 메소드를 사용하면 NaN값으로 0으로 변환하여 처리

```
df1.add(df2 , fill_value = 0)
```

	a	b	c	d
0	0	2	4	3.0
1	7	9	11	7.0
2	14	16	18	11.0

2.4.3. DataFrame + Series 연산

DataFrame과 Series간의 연산

Numpy 배열 연산 결과 확인

```
# numpy 브로드캐스팅 연산 복습
arr = np.arange(12).reshape(3,4)
arr - arr[0]
```

```
array([[0, 0, 0, 0],
       [4, 4, 4, 4],
       [8, 8, 8, 8]])
```

```
[[ 0,  1,  2,  3],
 [ 4,  5,  6,  7],    -   [0,1,2,3]
 [ 8,  9, 10, 11]]
          Broadcasting
[[0, 0, 0, 0],      [0,1,2,3]
 [4, 4, 4, 4],   -  [0,1,2,3]
 [8, 8, 8, 8]]      [0,1,2,3]
```

컬럼값을 기준으로 브로드캐스팅

```
#DataFrame
df = DataFrame(np.arange(12).reshape(3,4) ,  columns=['a','b','d','e'] )
df
```

	a	b	d	e
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

```
s1 = df.loc[0] # 1행 시리즈 변환
s1
```

```
a    0
b    1
d    2
e    3
Name: 0, dtype: int32
```

```
df - s1
```

	a	b	d	e
0	0	0	0	0
1	4	4	4	4
2	8	8	8	8

Series와 DataFrame연산은 컬럼을 기준으로 브로드캐스팅 연산

```
#DataFrame
df = DataFrame(np.arange(12).reshape(3,4) ,  columns=['a','b','d','e'] )
df
```

	a	b	d	e
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

```
s1 = df['a'] # 1행 시리즈 변환
s1
```

```
0    0
1    4
2    8
Name: a, dtype: int32
```

```
df - s1 #  컬럼을 기준으로 연산 !
```

	a	b	d	e	0	1	2
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df.sub(s1, axis=0)
```

	a	b	d	e
0	0	1	2	3
1	0	1	2	3
2	0	1	2	3

0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

0	0	0	0
4	4	4	4
8	8	8	8

[

2.5. Pandas U Function

2.5.1. DataFrame map, apply

Lambda 복습

- 한 줄로 함수를 표현하는 익명 함수 기법
- Lisp 언어에서 시작된 기법으로 오늘날 현대언어에 많이 사용

lambda argument : expression

```
1  f = lambda x: x * x
2  f(2)  #  4
3
4  values = [ i for i  in  map(  lambda x: x ** x, range(5)) ]
5  values
```

```
[1, 1, 4, 27, 256]
```

map 함수 사용

map(function, sequence)

```
1  values = [1,2,3,4,5]
2  list(map(lambda x: x* 2 , values)) # 파이썬3 list변환 필수
```

```
[2, 4, 6, 8, 10]
```

```
1  mask = [1,0,1,0,1]
2  list(map( lambda x,y: x * y , values, mask )) # 2이상일경우 리스트2개
```

```
[1, 0, 3, 0, 5]
```

Series 데이터의 map함수 사용

```
s = Series( np.arange(10) )
s.head(5)
```

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

```
s.map(lambda x : x **2).head(5)
```

```
0     0
1     1
2     4
3     9
4    16
dtype: int64
```

map함수를 이용한 Series 데이터 전처리

```
dic = {1:'a', 2:'b', 3:'c'} # dict 타입을 통해 데이터 교체
s.map(dic).head(5)
```

```
0    NaN
1      a
2      b
3      c
4    NaN
dtype: object
```

```
__s = Series(np.arange(10,20))
s.map(__s).head(5)
```

```
0    10
1    11
2    12
3    13
4    14
dtype: int32
```

https://www.kaggle.com/ljanjughazyan/wages wages.csv

```
df = pd.read_csv('data/wages.csv')
df.head()
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	male	white	16	49
1	96396.988643	66.23	female	white	16	62
2	48710.666947	63.77	female	white	16	33
3	80478.096153	63.22	female	other	16	95
4	82089.345498	63.08	female	white	17	43

describe

```
df.describe() # 요약 정보 표시
```

	earn	height	ed	age
count	1379.000000	1379.000000	1379.000000	1379.000000
mean	32446.292622	66.592640	13.354605	45.328499
std	31257.070006	3.818108	2.438741	15.789715
min	-98.580489	57.340000	3.000000	22.000000
25%	10538.790721	63.720000	12.000000	33.000000
50%	26877.870178	66.050000	13.000000	42.000000
75%	44506.215336	69.315000	15.000000	55.000000
max	317949.127955	77.210000	18.000000	95.000000

Sex column 확인

```
df.sex.unique() # 중복값 제거 확인
```

```
array(['male', 'female'], dtype=object)
```

Sex column 데이터를 기반으로 새로운 컬럼 생성

- Map + dict 사용

```
df['sex_code'] = df.sex.map( {'male': 0 , 'female':1 } )
df.head(5)
```

	earn	height	sex	race	ed	age	sex_code
0	79571.299011	73.89	male	white	16	49	0
1	96396.988643	66.23	female	white	16	62	1
2	48710.666947	63.77	female	white	16	33	1
3	80478.096153	63.22	female	other	16	95	1
4	82089.345498	63.08	female	white	17	43	1

Replace

- 변환 기능 담당 , 데이터 변환시 많이 사용

변환된 값 Series형태로 변환 → DataFrame의 값은 변화 없음

```
df.sex.replace( {'male': 0 , 'female':1 }  ).head()
```

```
0    0
1    1
2    1
3    1
4    1
Name: sex, dtype: int64
```

Replace + inplace = DataFrame값 변경

```
df.sex.replace( ['male', 'female'], [0,1],inplace = True ) # df 변경
df.head(5)
```

	earn	height	sex	race	ed	age	sex_code
0	79571.299011	73.89	0	white	16	49	0
1	96396.988643	66.23	1	white	16	62	1
2	48710.666947	63.77	1	white	16	33	1
3	80478.096153	63.22	1	other	16	95	1
4	82089.345498	63.08	1	white	17	43	1

Apply

- map과 달리 series전체에 함수를 적용

```
import pandas as pd
import numpy as np
from pandas import Series, DataFrame

df = DataFrame(np.random.randn(3,4) ,columns=list("abcd"))
df
```

	a	b	c	d
0	-1.650884	0.177436	0.810001	-0.690701
1	0.828963	-1.040844	0.394984	0.730493
2	-0.285950	-0.520928	-1.092211	-0.353299

```
df.apply( lambda x : x.max() ) # 열단위로 최대값  가져오기
```

```
a    0.828963
b    0.177436
c    0.810001
d    0.730493
dtype: float64
```

```
df.apply( lambda x : x.max() , axis = 1) # 행단위 최대값
```

```
0    0.810001
1    0.828963
2   -0.285950
dtype: float64
```

Applymap (

- series 단위가 아닌 element 단위로 함수를 적용함 (DataFrame)

```
format = lambda x : '%.1f' % x
print( df.applymap(format))
```

```
     a    b    c    d
0  0.8  1.4 -0.6 -1.8
1 -1.3 -0.2 -1.4  0.9
2 -0.9 -1.5 -0.8  0.0
```

2.5.2. function etc

describe

```
1 df.describe() # 요약 정보 표시
```

	earn	height	ed	age
count	1379.000000	1379.000000	1379.000000	1379.000000
mean	32446.292622	66.592640	13.354605	45.328499
std	31257.070006	3.818108	2.438741	15.789715
min	-98.580489	57.340000	3.000000	22.000000
25%	10538.790721	63.720000	12.000000	33.000000
50%	26877.870178	66.050000	13.000000	42.000000
75%	44506.215336	69.315000	15.000000	55.000000
max	317949.127955	77.210000	18.000000	95.000000

index labelling

```
1  # value와 key값 구하기
2  values =  np.array(list(enumerate(df['race'].unique())))[:,0].tolist()
3  keys   =  np.array(list(enumerate(df['race'].unique())))[:,1].tolist()
4  #keys = dict( enumerate(df['race'].unique())).keys()
5
6
7  values , keys
8  df['race'].replace(to_replace =  keys ,
9                     value = values,
10                    inplace = True)
11 df.head(5)
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	male	0	16	49
1	96396.988643	66.23	female	0	16	62
2	48710.666947	63.77	female	0	16	33
3	80478.096153	63.22	female	1	16	95
4	82089.345498	63.08	female	0	17	43

2.6.정렬과 순위

행의 색인이나 열색인 순으로 정렬 → 기준 필요

➔ 기본 오름차순 정렬 수행

```
f = DataFrame(np.random.randn(3,4) ,columns=list("cbda") , index=[3,1,2])
f
```

	c	b	d	a
3	0.510595	0.320232	-0.085458	-1.206954
1	0.853791	-0.794742	-1.314465	-1.267743
2	-0.292660	-0.698842	-1.294217	0.729347

```
df['a'].sort_index() # index 순으로 정렬 : Series
```

```
1   -0.192682
2   -1.243170
3   -0.092231
Name: a, dtype: float64
```

```
df.sort_index()     # index 순으로 정렬 : DataFrame
```

	c	b	d	a
1	0.853791	-0.794742	-1.314465	-1.267743
2	-0.292660	-0.698842	-1.294217	0.729347
3	0.510595	0.320232	-0.085458	-1.206954

sort columns

```
df.sort_index(axis = 1)   # column 기준으로 정렬
```

	a	b	c	d
3	-1.206954	0.320232	0.510595	-0.085458
1	-1.267743	-0.794742	0.853791	-1.314465
2	0.729347	-0.698842	-0.292660	-1.294217

내림차순 ascending = False

desc

```
df.sort_index(ascending = False)
```

	c	b	d	a
3	0.510595	0.320232	-0.085458	-1.206954
2	-0.292660	-0.698842	-1.294217	0.729347
1	0.853791	-0.794742	-1.314465	-1.267743

```
df.sort_index(axis = 1, ascending = False)
```

	d	c	b	a
3	-0.085458	0.510595	0.320232	-1.206954
1	-1.314465	0.853791	-0.794742	-1.267743
2	-1.294217	-0.292660	-0.698842	0.729347

Sort_values

```
df.sort_values('b')
```

	c	b	d	a
1	0.853791	-0.794742	-1.314465	-1.267743
2	-0.292660	-0.698842	-1.294217	0.729347
3	0.510595	0.320232	-0.085458	-1.206954

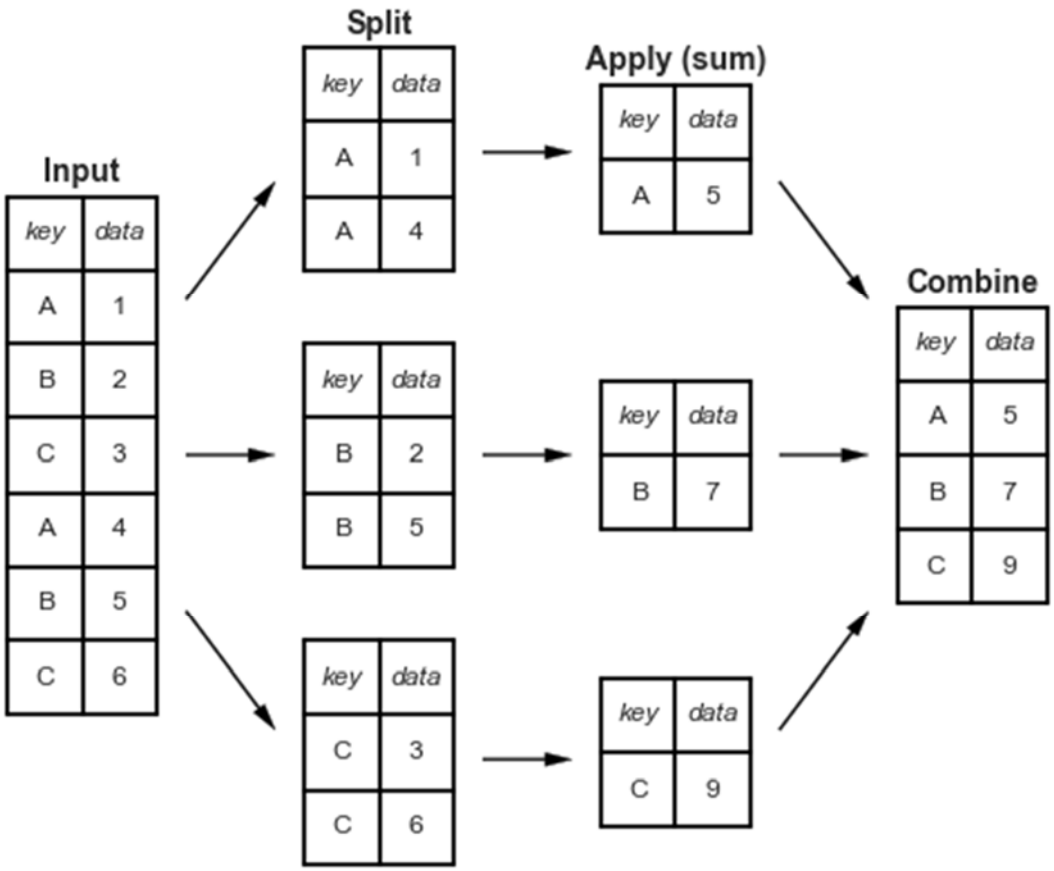
```
df.sort_values(['a','b'])
```

	c	b	d	a
1	0.853791	-0.794742	-1.314465	-1.267743
3	0.510595	0.320232	-0.085458	-1.206954
2	-0.292660	-0.698842	-1.294217	0.729347

2.7. Pandas Groupby

SQL groupby 명령어와 같음

- split → apply → combine 과정을 거쳐 연산함



	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
2	Devils	2	2014	863
3	Devils	3	2015	673
4	Kings	3	2014	741
5	kings	4	2015	812
6	Kings	1	2016	756
7	Kings	1	2017	788
8	Riders	2	2016	694
9	Royals	4	2014	701
10	Royals	1	2015	804
11	Riders	2	2017	690

Groupy(기준컬럼)[적용컬럼]. 집계함수()

```
1 df.groupby("Year")["Points"].sum()
```

```
Year
2014    3181
2015    3078
2016    1450
2017    1478
Name: Points, dtype: int64
```

```
1 df.groupby("Team")["Rank"].mean()
```

```
Team
Devils    2.500000
Kings     1.666667
Riders    1.750000
Royals    2.500000
kings     4.000000
Name: Rank, dtype: float64
```

Hierarchical index

- Groupby 명령의 결과물도 결국은 dataframe
- 두 개의 column으로 groupby를 할 경우, index가 두개 생성

qqqqqqqqqqqqqqqqqqqqqqqqqqqqq

```
1 h_index = df.groupby(["Team", "Year"])["Points"].sum()
2 h_index
```

```
Team    Year
Devils  2014    863
        2015    673
Kings   2014    741
        2016    756
        2017    788
Riders  2014    876
        2015    789
        2016    694
        2017    690
Royals  2014    701
        2015    804
kings   2015    812
Name: Points, dtype: int64
```

Hierarchical index – unstack()

- Group으로 묶여진 데이터를 matrix 형태로 전환해줌

```
1 h_index.unstack()
```

Year	2014	2015	2016	2017
Team				
Devils	863.0	673.0	NaN	NaN
Kings	741.0	NaN	756.0	788.0
Riders	876.0	789.0	694.0	690.0
Royals	701.0	804.0	NaN	NaN
kings	NaN	812.0	NaN	NaN

Hierarchical index – swaplevel

- Index level을 변경할 수 있음

```
1 h_index.swaplevel() # 레벨 변경
```

```
Year  Team
2014  Devils    863
2015  Devils    673
2014  Kings     741
2016  Kings     756
2017  Kings     788
2014  Riders    876
2015  Riders    789
2016  Riders    694
2017  Riders    690
2014  Royals    701
2015  Royals    804
      kings     812
Name: Points, dtype: int64
```

2.7.1. Groupby – aggregation

Groupby – gropued

- Groupby에 의해 Split된 상태를 추출 가능함

```
Teams = df.groupby('Team')  # Teams 그룹 생성
```

```
for name in Teams:
    print(name)
```

```
('Devils',    Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1    Team  Rank  Year  Points
2           2             2               2  Devils     2  2014     863
3           3             3               3  Devils     3  2015     673)
('Kings',    Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1   Team  Rank  Year  Points
4           4             4               4  Kings      3  2014     741
6           6             6               6  Kings      1  2016     756
7           7             7               7  Kings      1  2017     788)
('Riders',    Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1    Team  Rank  Year  Points
0           0             0               0  Riders     1  2014     876
1           1             1               1  Riders     2  2015     789
8           8             8               8  Riders     2  2016     694
11         11            11              11  Riders     2  2017     690)
('Royals',    Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1    Team  Rank  Year  Points
9           9             9               9  Royals     4  2014     701
10         10            10              10  Royals     1  2015     804)
('kings',    Unnamed: 0  Unnamed: 0.1  Unnamed: 0.1.1   Team  Rank  Year  Points
5           5             5               5  kings      4  2015     812)
```

```
Teams.get_group('Devils')
```

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	Team	Rank	Year	Points
2	2	2	2	Devils	2	2014	863
3	3	3	3	Devils	3	2015	673

```
Teams.agg(sum)
```

Team	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	Rank	Year	Points
Devils	5	5	5	5	4029	1536
Kings	17	17	17	5	6047	2285
Riders	20	20	20	7	8062	3049
Royals	19	19	19	5	4029	1505
kings	5	5	5	4	2015	812

Groupby – filter

- 특정 조건으로 데이터를 검색할 때 사용

```
df.groupby('Team').filter(lambda x: len(x) >= 3).head(3)
```

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
4	Kings	3	2014	741

```
df.groupby('Team').filter(lambda x: x["Points"].max() > 800).head(3)
```

	Team	Rank	Year	Points
0	Riders	1	2014	876

2.7.2. Pivot Table

```
df_phone = pd.read_csv("data/phone_data.csv")
df_phone.pivot_table(["duration"],
                     index=[df_phone.month,df_phone.item],  # 인덱스 컬럼
                     columns=df_phone.network, aggfunc="sum", fill_value=0)
```

		duration								
	network	Meteor	Tesco	Three	Vodafone	data	landline	special	voicemail	world
month	item									
2014-11	call	1521	4045	12458	4316	0.000	2906	0	301	0
	data	0	0	0	0	998.441	0	0	0	0
	sms	10	3	25	55	0.000	0	1	0	0
2014-12	call	2010	1819	6316	1302	0.000	1424	0	690	0
	data	0	0	0	0	1032.870	0	0	0	0
	sms	12	1	13	18	0.000	0	0	0	4
2015-01	call	2207	2904	6445	3626	0.000	1603	0	285	0
	data	0	0	0	0	1067.299	0	0	0	0
	sms	10	3	33	40	0.000	0	0	0	0
2015-02	call	1188	4087	6279	1864	0.000	730	0	268	0
	data	0	0	0	0	1067.299	0	0	0	0
	sms	1	2	11	23	0.000	0	2	0	0
2015-03	call	274	973	4966	3513	0.000	11770	0	231	0
	data	0	0	0	0	998.441	0	0	0	0
	sms	0	4	5	13	0.000	0	0	0	3

Crosstab

- 특허 두 칼럼에 교차 빈도, 비율, 덧셈 등을 구할 때 사용
- Pivot table의 특수한 형태

```
df_movie = pd.read_csv("data/movie_rating.csv")
df_movie.head()
```

	critic	title	rating
0	Jack Matthews	Lady in the Water	3.0
1	Jack Matthews	Snakes on a Plane	4.0
2	Jack Matthews	You Me and Dupree	3.5
3	Jack Matthews	Superman Returns	5.0
4	Jack Matthews	The Night Listener	3.0

```
df_movie.pivot_table(["rating"], index=df_movie.critic, columns=df_movie.title,
                     aggfunc="sum", fill_value=0)
```

	rating					
title	Just My Luck	Lady in the Water	Snakes on a Plane	Superman Returns	The Night Listener	You Me and Dupree
critic						
Claudia Puig	3.0	0.0	3.5	4.0	4.5	2.5
Gene Seymour	1.5	3.0	3.5	5.0	3.0	3.5
Jack Matthews	0.0	3.0	4.0	5.0	3.0	3.5
Lisa Rose	3.0	2.5	3.5	3.5	3.0	2.5
Mick LaSalle	2.0	3.0	4.0	3.0	3.0	2.0
Toby	0.0	0.0	4.5	4.0	0.0	1.0

3가지 방법으로 가능

```
df_movie.pivot_table(["rating"], index=df_movie.critic, columns=df_movie.title,
                     aggfunc="sum", fill_value=0)
```

	rating					
title	Just My Luck	Lady in the Water	Snakes on a Plane	Superman Returns	The Night Listener	You Me and Dupree
critic						
Claudia Puig	3.0	0.0	3.5	4.0	4.5	2.5
Gene Seymour	1.5	3.0	3.5	5.0	3.0	3.5
Jack Matthews	0.0	3.0	4.0	5.0	3.0	3.5
Lisa Rose	3.0	2.5	3.5	3.5	3.0	2.5
Mick LaSalle	2.0	3.0	4.0	3.0	3.0	2.0
Toby	0.0	0.0	4.5	4.0	0.0	1.0

crosstab

```
pd.crosstab(index=df_movie.critic,columns=df_movie.title,values=df_movie.rating,
            aggfunc="first").fillna(0)
```

title	Just My Luck	Lady in the Water	Snakes on a Plane	Superman Returns	The Night Listener	You Me and Dupree
critic						
Claudia Puig	3.0	0.0	3.5	4.0	4.5	2.5
Gene Seymour	1.5	3.0	3.5	5.0	3.0	3.5
Jack Matthews	0.0	3.0	4.0	5.0	3.0	3.5
Lisa Rose	3.0	2.5	3.5	3.5	3.0	2.5
Mick LaSalle	2.0	3.0	4.0	3.0	3.0	2.0
Toby	0.0	0.0	4.5	4.0	0.0	1.0

```
df_movie.groupby(["critic","title"]).agg({"rating":"first"}).unstack().fillna(0)
```

	rating					
title	Just My Luck	Lady in the Water	Snakes on a Plane	Superman Returns	The Night Listener	You Me and Dupree
critic						
Claudia Puig	3.0	0.0	3.5	4.0	4.5	2.5
Gene Seymour	1.5	3.0	3.5	5.0	3.0	3.5
Jack Matthews	0.0	3.0	4.0	5.0	3.0	3.5
Lisa Rose	3.0	2.5	3.5	3.5	3.0	2.5

Merge

- SQL에서 많이 사용하는 Merge와 같은 기능
- 두 개의 데이터를 하나로 합침

	subject_id	test_score
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14
6	8	15
7	9	1
8	10	61
9	11	16

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

```
pd.merge(df_a, df_b, on='subject_id')
```

	subject_id	test_score	first_name	last_name
0	4	61	Billy	Bonder
1	5	16	Brian	Black
2	7	14	Bryce	Brice
3	8	15	Betty	Btisan

```
pd.merge(df_a, df_b, on='subject_id', how='left')
```

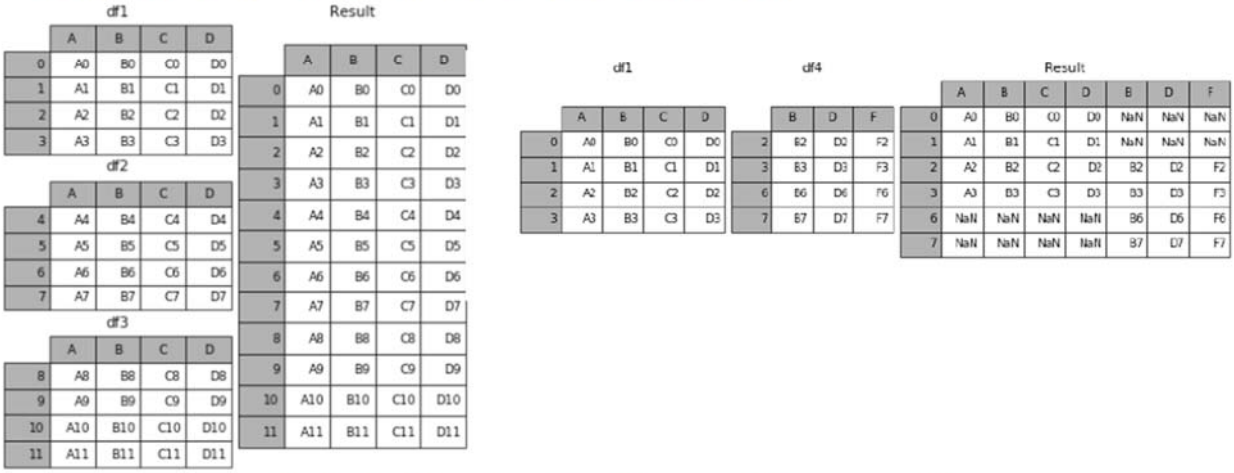
	subject_id	test_score	first_name	last_name
0	1	51	NaN	NaN
1	2	15	NaN	NaN
2	3	15	NaN	NaN
3	4	61	Billy	Bonder
4	5	16	Brian	Black
5	7	14	Bryce	Brice
6	8	15	Betty	Btisan
7	9	1	NaN	NaN
8	10	61	NaN	NaN
9	11	16	NaN	NaN

```
pd.merge(df_a, df_b, right_index=True, left_index=True)
```

	subject_id_x	test_score	subject_id_y	first_name	last_name
0	1	51	4	Billy	Bonder
1	2	15	5	Brian	Black
2	3	15	6	Bran	Balwner

Concat

- 같은 형태의 데이터를 붙이는 연산작업



https://pandas.pydata.org/pandas-docs/stable/merging.html

```
raw_data = {
        'subject_id': ['1', '2', '3', '4', '5'],
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
df_a = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'last_name'])
df_a
```

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches

```
raw_data = {
        'subject_id': ['4', '5', '6', '7', '8'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}
df_b = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name', 'last_name'])
df_b
```

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

```
df_new = pd.concat([df_a, df_b])
df_new.reset_index()
```

	index	subject_id	first_name	last_name
0	0	1	Alex	Anderson
1	1	2	Amy	Ackerman
2	2	3	Allen	Ali
3	3	4	Alice	Aoni
4	4	5	Ayoung	Atiches
5	0	4	Billy	Bonder
6	1	5	Brian	Black
7	2	6	Bran	Balwner
8	3	7	Bryce	Brice
9	4	8	Betty	Btisan

```
df_a.append(df_b)
```

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

```
df_new = pd.concat([df_a, df_b], axis=1)
df_new.reset_index()
```

	index	subject_id	first_name	last_name	subject_id	first_name	last_name
0	0	1	Alex	Anderson	4	Billy	Bonder
1	1	2	Amy	Ackerman	5	Brian	Black
2	2	3	Allen	Ali	6	Bran	Balwner
3	3	4	Alice	Aoni	7	Bryce	Brice
4	4	5	Ayoung	Atiches	8	Betty	Btisan