Pandas Tutorial

https://github.com/dlcjfgmlnasa/Tongmyong_College_Tensorflow_Tutorial

Created by **Choelhui lee**

Pandas (판다스)

- **Pandas** 는 numpy를 기반으로 구현된 파이썬 라이브러리
- **Pandas** 는 행과 열로 이루어진 데이터 객체를 만들어 다룰 수 있게 되며 보다 안정적으로 대용량의 데이터들을 처리하는데 매우 편리한 도구

Pandas Install

```
pip install pandas
```

Pandas import

```
import pandas as pd
```

02. Pandas 자료구조

- `Pandas` 에서는 기본적으로 정의되는 자료구조인 `Series` 와 `Data Frame` 을 사용
- `이 자료구조들은 빅 데이터 분석에 있어서 높은 수준의 성능을 보여줌`

01. Series 정의

```
obj = pd.Series([4, 7, -5, 3])
obj

>> 0 4
>> 1 7
>> 2 -5
>> 3 3
>> dtype: int64
```

02. Series의 값만 확인

```
obj.values

>> array([ 4,  7, -5,  3])
```

03. Series의 인덱스만 확인

```
obj.index

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

04. Series의 자료형 확인

```
obj.dtypes

>> dtype('int64')
```

05. 인덱스를 바꿀 수 있음

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
obj2

>> d 4
>> b 7
>> a -5
>> c 3
>> dtype: int64
```

06. python의 dictionary 자료형을 Series data로 만들 수 있음

```
sdata = {'Kim': 35000, 'Beomwoo': 67000,
         'Joan': 12000, 'Choi': 4000}
obj3 = pd.Series(sdata)
obj3

>> Kim 35000
>> Beomwoo 67000
>> Joan 12000
>> Choi 4000
>> dtype: int64
```

```
obj3.name = 'Salary'
obj3.index.name = "Names"
obj3

>> Names Kim 35000
>> Beomwoo 67000
>> Joan 12000
>> Choi 4000
>> Name: Salary, dtype: int64
```

07. index 변경

```
obj3.index = ['A', 'B', 'C', 'D']
obj3

>> A 35000
>> B 67000
>> C 12000
>> D 4000
>> Name: Salary, dtype: int64
```

03. DataFrame

01. DataFrame 정의

- 이전에 `DataFrame` 에 들어갈 데이터를 정의해주어야 하는데,
  이는 python의 dictionary 또는 numpy의 array로 정의할 수 있다

```
data = {
 'name': ['Beomwoo', 'Beomwoo', 'Beomwoo', 'Kim', 'Park'],
 'year': [2013, 2014, 2015, 2016, 2015],
 'points': [1.5, 1.7, 3.6, 2.4, 2.9]
}
df = pd.DataFrame(data)
```

	name	year	points
0	Beomwoo	2013	1.5
1	Beomwoo	2014	1.7
2	Beomwoo	2015	3.6
3	Kim	2016	2.4
4	Park	2015	2.9

02. 행 방향의 index

```
df.index

>> RangeIndex(start=0, stop=5, step=1)
```

03. 열 방향의 index

```
df.columns

>> Index(['name', 'year', 'points'], dtype='object')
```

04. 값 얻기

```
df.values

>> array([['Beomwoo', 2013, 1.5],
>>        ['Beomwoo', 2014, 1.7],
>>        ['Beomwoo', 2015, 3.6],
>>        ['Kim', 2016, 2.4],
>>        ['Park', 2015, 2.9]],
>>        dtype=object)
```

05. 각 인덱스에 대한 이름 설정

```
df.index.name = 'Num'
df.columns.name = 'Info'
```

Info	name	year	points
Num			
0	Beomwoo	2013	1.5
1	Beomwoo	2014	1.7
2	Beomwoo	2015	3.6
3	Kim	2016	2.4
4	Park	2015	2.9

06. DataFrame을 만들면서 columns와 index를 설정

```
df2 = pd.DataFrame(
          data,
          columns=['year', 'name', 'points', 'penalty'],
          index=['one', 'two', 'three', 'four', 'five'])
df2
```

	year	name	points	penalty
one	2013	Beomwoo	1.5	NaN
two	2014	Beomwoo	1.7	NaN
three	2015	Beomwoo	3.6	NaN
four	2016	Kim	2.4	NaN
five	2015	Park	2.9	NaN

- `DataFrame`을 정의하면서, data로 들어가는 python dictionary와 `columns의 순서가 달라도 알아서 맞춰서 정의된다.`
- 하지만 data에 포함되어 있지 않은 값은 `NaN(Not a Number)`으로 나타나게 되는데, 이는 `null과 같은 개념`
- `NaN`값은 추후에 어떠한 방법으로도 처리가 되지 않는 데이터
- 따라서 `올바른 데이터 처리를 위해 추가적으로 값을 넣어줘야 한다.`

07. describe() 함수는 DataFrame의 계산 가능한 값들에 대한 다양한 계산 값을 보여줌

```
df2.describe()
```

	year	points
count	5.000000	5.000000
mean	2014.600000	2.420000
std	1.140175	0.864292
min	2013.000000	1.500000
25%	2014.000000	1.700000
50%	2015.000000	2.400000
75%	2015.000000	2.900000
max	2016.000000	3.600000

04. DataFrame Indexing

```
data = {
  "names": ["Kilho", "Kilho", "Kilho", "Charles", "Charles"],
  "year": [2014, 2015, 2016, 2015, 2016],
  "points": [1.5, 1.7, 3.6, 2.4, 2.9]}

df = pd.DataFrame(
   data,
   columns=["year", "names", "points", "penalty"],
   index=["one", "two", "three", "four", "five"])
```

	year	names	points	penalty
one	2014	Kilho	1.5	NaN
two	2015	Kilho	1.7	NaN
three	2016	Kilho	3.6	NaN
four	2015	Charles	2.4	NaN
five	2016	Charles	2.9	NaN

01. DataFrame에서 열을 선택하고 조작

```
df['year']

>> one 2014
>> two 2015
>> three 2016
>> four 2015
>> five 2016
>> Name: year, dtype: int64
```

```
df.year

>> one 2014
>> two 2015
>> three 2016
>> four 2015
>> five 2016
>> Name: year, dtype: int64
```

```
df[['year','points']]
```

	year	points
one	2014	1.5
two	2015	1.7
three	2016	3.6
four	2015	2.4
five	2016	2.9

02. 특정 열에 대해 위와 같이 선택하고, 우리가 원하는 값을 대입

```
df['penalty'] = 0.5
```

	year	names	points	penalty
one	2014	Kilho	1.5	0.5
two	2015	Kilho	1.7	0.5
three	2016	Kilho	3.6	0.5
four	2015	Charles	2.4	0.5
five	2016	Charles	2.9	0.5

```
# python의 List나 numpy의 array
df['penalty'] = [0.1, 0.2, 0.3, 0.4, 0.5]
```

	year	names	points	penalty
one	2014	Kilho	1.5	0.1
two	2015	Kilho	1.7	0.2
three	2016	Kilho	3.6	0.3
four	2015	Charles	2.4	0.4
five	2016	Charles	2.9	0.5

```
df['zeros'] = np.arange(5)
```

	year	names	points	penalty	zeros
one	2014	Kilho	1.5	0.1	0
two	2015	Kilho	1.7	0.2	1
three	2016	Kilho	3.6	0.3	2
four	2015	Charles	2.4	0.4	3
five	2016	Charles	2.9	0.5	4

```
# Series를 추가할 수도 있다.
val = pd.Series([-1.2, -1.5, -1.7],
                index=['two','four','five'])

df['debt'] = val
```

	year	names	points	penalty	zeros	debt
one	2014	Kilho	1.5	0.1	0	NaN
two	2015	Kilho	1.7	0.2	1	-1.2
three	2016	Kilho	3.6	0.3	2	NaN
four	2015	Charles	2.4	0.4	3	-1.5
five	2016	Charles	2.9	0.5	4	-1.7

- **`하지만 Series로 넣을 때는 val와 같이 넣으려는 data의 index에 맞춰서 데이터가 들어간다.`**
- 이점이 python list나 numpy array로 데이터를 넣을때와 가장 큰 차이점

```
df['net_points'] = df['points'] - df['penalty']
df['high_points'] = df['net_points'] > 2.0
```

	year	names	points	penalty	zeros	debt	net_points	high_points
one	2014	Kilho	1.5	0.1	0	NaN	1.4	False
two	2015	Kilho	1.7	0.2	1	-1.2	1.5	False
three	2016	Kilho	3.6	0.3	2	NaN	3.3	True
four	2015	Charles	2.4	0.4	3	-1.5	2.0	False
five	2016	Charles	2.9	0.5	4	-1.7	2.4	True

03. 열 삭제하기

```
del df['high_points']

del df['net_points']
del df['zeros']
```

	year	names	points	penalty	debt
one	2014	Kilho	1.5	0.1	NaN
two	2015	Kilho	1.7	0.2	-1.2
three	2016	Kilho	3.6	0.3	NaN
four	2015	Charles	2.4	0.4	-1.5
five	2016	Charles	2.9	0.5	-1.7

```
df.columns

>> Index(['year', 'names', 'points', 'penalty', 'debt'],
>>       dtype='object')
```

```
df.index.name = 'Order'
df.columns.name = 'Info
```

Info	year	names	points	penalty	debt
Order					
one	2014	Kilho	1.5	0.1	NaN
two	2015	Kilho	1.7	0.2	-1.2
three	2016	Kilho	3.6	0.3	NaN
four	2015	Charles	2.4	0.4	-1.5
five	2016	Charles	2.9	0.5	-1.7

04. DataFrame에서 행을 선택하고 조작

- pandas에서는 DataFrame에서 행을 인덱싱하는 방법이 무수히 많다.
- 물론 위에서 소개했던 열을 선택하는 방법도 수많은 방법중에 하나에 불과

```
# 0번째 부터 2(3-1) 번째까지 가져온다.
# 뒤에 써준 숫자번째의 행은 뺀다.

df[0:3]
```

Info	year	names	points	penalty	debt
Order					
one	2014	Kilho	1.5	0.1	NaN
two	2015	Kilho	1.7	0.2	-1.2
three	2016	Kilho	3.6	0.3	NaN
four	2015	Charles	2.4	0.4	-1.5
five	2016	Charles	2.9	0.5	-1.7

```
# two라는 행부터 four라는 행까지 가져온다.
# 뒤에 써준 이름의 행을 빼지 않는다.

df['two':'four']          # 하지만 비추천!
```

Info	year	names	points	penalty	debt
Order					
one	2014	Kilho	1.5	0.1	NaN
two	2015	Kilho	1.7	0.2	-1.2
three	2016	Kilho	3.6	0.3	NaN

```
# 아래 방법을 권장한다.
# .loc 또는 .iloc 함수를 사용하는 방법.

df.loc['two'] # 반환 형태는 Series

>> Info
>> year 2015
>> names Kilho
>> points 1.7
>> penalty 0.2
>> debt -1.2
>> Name: two, dtype: obje
```

```
df.loc['two':'four']
```

Info	year	names	points	penalty	debt
Order					
two	2015	Kilho	1.7	0.2	-1.2
three	2016	Kilho	3.6	0.3	NaN
four	2015	Charles	2.4	0.4	-1.5

```
df.loc['two':'four', 'points']

>> Order
>> two 1.7
>> three 3.6
>> four 2.4
>> Name: points, dtype: float64
```

```
df.loc[:,'year'] # == df['year']

>> Order
>> one 2014
>> two 2015
>> three 2016
>> four 2015
>> five 2016 Name: year, dtype: int64
```

```
df.loc[:,['year','names']]
```

Info	year	names
Order		
one	2014	Kilho
two	2015	Kilho
three	2016	Kilho
four	2015	Charles
five	2016	Charles

```
df.loc['three':'five','year':'penalty']
```

Info	year	names	points	penalty
Order				
three	2016	Kilho	3.6	0.3
four	2015	Charles	2.4	0.4
five	2016	Charles	2.9	0.5

```
# 새로운 행 삽입하기
df.loc['six',:] = [2013,'Jun',4.0,0.1,2.1]
```

Info	year	names	points	penalty	debt
Order					
one	2014.0	Kilho	1.5	0.1	NaN
two	2015.0	Kilho	1.7	0.2	-1.2
three	2016.0	Kilho	3.6	0.3	NaN
four	2015.0	Charles	2.4	0.4	-1.5
five	2016.0	Charles	2.9	0.5	-1.7
six	2013.0	Jun	4.0	0.1	2.1

```
# .iloc 사용:: index 번호를 사용한다.
df.iloc[3] # 3번째 행을 가져온다.

>> Info
>> year 2015
>> names Charles
>> points 2.4
>> penalty 0.4
>> debt -1.5
>> Name: four, dtype: object
```

```
df.iloc[3:5, 0:2]
```

Info	year	names
Order		
four	2015.0	Charles
five	2016.0	Charles

```
df.iloc[[0,1,3], [1,2]]
```

Info	names	points
Order		
one	Kilho	1.5
two	Kilho	1.7
four	Charles	2.4

```
df.iloc[:,1:4]
```

Info	names	points	penalty
Order			
one	Kilho	1.5	0.1
two	Kilho	1.7	0.2
three	Kilho	3.6	0.3
four	Charles	2.4	0.4
five	Charles	2.9	0.5
six	Jun	4.0	0.1

```
df.iloc[1,1]

>> 'Kilho'
```

05. DataFrame에서의 boolean Indexing

Info	year	names	points	penalty	debt
Order					
one	2014.0	Kilho	1.5	0.1	NaN
two	2015.0	Kilho	1.7	0.2	-1.2
three	2016.0	Kilho	3.6	0.3	NaN
four	2015.0	Charles	2.4	0.4	-1.5
five	2016.0	Charles	2.9	0.5	-1.7
six	2013.0	Jun	4.0	0.1	2.1

```
# year가 2014보다 큰 boolean data
df['year'] > 2014

>> Order
>> one False
>> two True
>> three True
>> four True
>> five True
>> six False
>> Name: year, dtype: bool
```

```
# year가 2014보다 큰 모든 행의 값

df.loc[df['year']>2014,:]
```

Info	year	names	points	penalty	debt
Order					
two	2015.0	Kilho	1.7	0.2	-1.2
three	2016.0	Kilho	3.6	0.3	NaN
four	2015.0	Charles	2.4	0.4	-1.5
five	2016.0	Charles	2.9	0.5	-1.7

```
df.loc[df['names'] == 'Kilho',['names','points']]
```

Info	names	points
Order		
one	Kilho	1.5
two	Kilho	1.7
three	Kilho	3.6

```
# numpy에서와 같이 논리연산을 응용할 수 있다.

df.loc[(df['points']>2)&(df['points']<3),:]
```

Info	year	names	points	penalty	debt
Order					
four	2015.0	Charles	2.4	0.4	-1.5
five	2016.0	Charles	2.9	0.5	-1.7

```
# 새로운 값을 대입할 수도 있다.

df.loc[df['points'] > 3, 'penalty'] = 0
```

Info	year	names	points	penalty	debt
Order					
one	2014.0	Kilho	1.5	0.1	NaN
two	2015.0	Kilho	1.7	0.2	-1.2
three	2016.0	Kilho	3.6	0.0	NaN
four	2015.0	Charles	2.4	0.4	-1.5
five	2016.0	Charles	2.9	0.5	-1.7
six	2013.0	Jun	4.0	0.0	2.1

06. Data

```
# DataFrame을 만들때 index, column을 설정하지 않으면
# 기본값으로 0부터 시작하는 정수형 숫자로 입력된다.

df = pd.DataFrame(np.random.randn(6,4))
```

	0	1	2	3
0	0.682000	-0.570393	-1.602829	-1.316469
1	-1.176203	0.171527	0.387018	1.027615
2	-0.263178	-0.212049	1.006856	0.096111
3	2.679378	0.347145	0.548144	0.826258
4	-0.249877	0.018721	-0.393715	0.302361
5	0.851420	-0.440360	-0.345895	1.055936

```
df.columns = ['A', 'B', 'C', 'D']
df.index = pd.date_range('20160701', periods=6)

# pandas에서 제공하는 date range함수는 datetime 자료형으로 구성된,
# 날짜 시각등을 알 수 있는 자료형을 만드는 함수

df.index

>> DatetimeIndex(['2016-07-01', '2016-07-02', '2016-07-03',
>>                '2016-07-04', '2016-07-05', '2016-07-06'],
>>               dtype='datetime64[ns]', freq='D')
```

	A	B	C	D
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469
2016-07-02	-1.176203	0.171527	0.387018	1.027615
2016-07-03	-0.263178	-0.212049	1.006856	0.096111
2016-07-04	2.679378	0.347145	0.548144	0.826258
2016-07-05	-0.249877	0.018721	-0.393715	0.302361
2016-07-06	0.851420	-0.440360	-0.345895	1.055936

```
# np.nan은 NaN값을 의미한다.

df['F'] = [1.0, np.nan, 3.5, 6.1, np.nan, 7.0]
```

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-02	-1.176203	0.171527	0.387018	1.027615	NaN
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	NaN
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

07. NaN 없애기

```
# 행의 값중 하나라도 nan인 경우 그 행을 없앤다.

df.dropna(how='any')
```

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

```
# 행의 값의 모든 값이 nan인 경우 그 행을 없앤다.

df.dropna(how='all')
```

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-02	-1.176203	0.171527	0.387018	1.027615	NaN
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	NaN
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

주의 : drop함수는 특정 행 또는 열을 drop하고난 DataFrame을 반환

- 즉, 반환을 받지 않으면 기존의 DataFrame은 그대로이다.
- 아니면, inplace=True라는 인자를 추가하여, 반환을 받지 않고서도 기존의 DataFrame이 변경되도록 한다

```
# nan값에 값 넣기

df.fillna(value=0.5)
```

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-02	-1.176203	0.171527	0.387018	1.027615	0.5
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	0.5
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

```
# nan값인지 확인하기

df.isnull()
```

	A	B	C	D	F
2016-07-01	False	False	False	False	False
2016-07-02	False	False	False	False	True
2016-07-03	False	False	False	False	False
2016-07-04	False	False	False	False	False
2016-07-05	False	False	False	False	True
2016-07-06	False	False	False	False	False

```
# F열에서 nan값을 포함하는 행만 추출하기

df.loc[df.isnull()['F'],:]
```

	A	B	C	D	F
2016-07-02	-1.176203	0.171527	0.387018	1.027615	NaN
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	NaN

```
pd.to_datetime('20160701')

>> Timestamp('2016-07-01 00:00:00')
```

```
# 특정 행 drop하기

df.drop(pd.to_datetime('20160701'))
```

	A	B	C	D	F
2016-07-02	-1.176203	0.171527	0.387018	1.027615	NaN
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	NaN
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

```
# 2개 이상도 가능

df.drop([pd.to_datetime('20160702'),
         pd.to_datetime('20160704')])
```

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	NaN
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

```
# 특정 열 삭제하기

df.drop('F', axis = 1)
```

	A	B	C	D
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469
2016-07-02	-1.176203	0.171527	0.387018	1.027615
2016-07-03	-0.263178	-0.212049	1.006856	0.096111
2016-07-04	2.679378	0.347145	0.548144	0.826258
2016-07-05	-0.249877	0.018721	-0.393715	0.302361
2016-07-06	0.851420	-0.440360	-0.345895	1.055936

```
# 2개 이상의 열도 가능

df.drop(['B','D'], axis = 1)
```

	A	C	F
2016-07-01	0.682000	-1.602829	1.0
2016-07-02	-1.176203	0.387018	NaN
2016-07-03	-0.263178	1.006856	3.5
2016-07-04	2.679378	0.548144	6.1
2016-07-05	-0.249877	-0.393715	NaN
2016-07-06	0.851420	-0.345895	7.0

08. Data 분석용 함수들

```
data = [[1.4, np.nan],
        [7.1, -4.5],
        [np.nan, np.nan],
        [0.75, -1.3]]

df = pd.DataFrame(data,
                  columns=["one", "two"],
                  index=["a", "b", "c", "d"])
```

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

```
# 행방향으로의 합(즉, 각 열의 합)

df.sum(axis=0)

>> one 9.25
>> two -5.80
>> dtype: float64
```

```
# 행방향으로의 합(즉, 각 열의 합)

df.sum(axis=1)

>> a 1.40
>> b 2.60
>> c 0.00
>> d -0.55
>> dtype: float64
```

- 이때, **위에서 볼 수 있듯이 NaN값은 배제하고 계산**
- `NaN 값을 배제하지 않고 계산하려면 아래와 같이 skipna에 대해 false를 지정`

```
df.sum(axis=1, skipna=False)

>> a NaN
>> b 2.60
>> c NaN
>> d -0.55
>> dtype: float64
```

```
# 특정 행 또는 특정 열에서만 계산하기

df['one'].sum()

>> 9.25
```

```
df.loc['b'].sum()

>> 2.5999999999999996
```

pandas에서 DataFrame에 적용되는 함수

- **count** : 전체 성분의 (NaN이 아닌) 값의 갯수를 계산
- **min**, **max** : 전체 성분의 최솟, 최댓값을 계산
- **argmin**, **argmax** : 전체 성분의 최솟값, 최댓값이 위치한 (정수)인덱스를 반환
- **idxmin**, **idxmax** : 전체 인덱스 중 최솟값, 최댓값을 반환
- **quantile** : 전체 성분의 특정 사분위수에 해당하는 값을 반환 (0~1 사이)

- **sum** : 전체 성분의 합을 계산
- **mean** : 전체 성분의 평균을 계산
- **median** : 전체 성분의 중간값을 반환
- **mad** : 전체 성분의 평균값으로부터의 절대 편차(absolute deviation)의 평균을 계산
- **std, var** : 전체 성분의 표준편차, 분산을 계산
- **cumsum** : 맨 첫 번째 성분부터 각 성분까지의 누적합을 계산 (0에서부터 계속 더해짐)
- **cumprod** : 맨 첫번째 성분부터 각 성분까지의 누적곱을 계산 (1에서부터 계속 곱해짐)

```
df2 = pd.DataFrame(np.random.randn(6, 4),
                   columns=["A", "B", "C", "D"],
                   index=pd.date_range("20160701", periods=6))
```

	A	B	C	D
2016-07-01	1.024359	0.213159	0.277114	-0.304599
2016-07-02	-1.693934	0.619885	-1.620676	0.067814
2016-07-03	-0.216422	0.659557	0.342833	-0.141637
2016-07-04	-0.873492	0.669932	-0.515944	0.280424
2016-07-05	-0.923094	-1.174652	2.678657	-0.663322
2016-07-06	-1.580576	0.539906	-1.900090	-0.428551

```
# A열과 B열의 상관계수 구하기
df2['A'].corr(df2['B'])

>> -0.06715327766901227
```

```
# B열과 C열의 공분산 구하기
df2['B'].cov(df2['C'])

>> -1.0099019967454226
```

09. 정렬함수 및 기타함수

```
dates = df2.index
random_dates = np.random.permutation(dates)
df2 = df2.reindex(
    index=random_dates,
    columns=["D", "B", "C", "A"])
```

	D	B	C	A
2016-07-02	0.067814	0.619885	-1.620676	-1.693934
2016-07-06	-0.428551	0.539906	-1.900090	-1.580576
2016-07-03	-0.141637	0.659557	0.342833	-0.216422
2016-07-01	-0.304599	0.213159	0.277114	1.024359
2016-07-05	-0.663322	-1.174652	2.678657	-0.923094
2016-07-04	0.280424	0.669932	-0.515944	-0.873492

```
# index와 column의 순서가 섞여있다.
# 이때 index가 오름차순이 되도록 정렬해보자

df2.sort_index(axis=0)
```

	D	B	C	A
2016-07-01	-0.304599	0.213159	0.277114	1.024359
2016-07-02	0.067814	0.619885	-1.620676	-1.693934
2016-07-03	-0.141637	0.659557	0.342833	-0.216422
2016-07-04	0.280424	0.669932	-0.515944	-0.873492
2016-07-05	-0.663322	-1.174652	2.678657	-0.923094
2016-07-06	-0.428551	0.539906	-1.900090	-1.580576

```
# column을 기준으로?

df2.sort_index(axis=1)
```

	A	B	C	D
2016-07-02	-1.693934	0.619885	-1.620676	0.067814
2016-07-06	-1.580576	0.539906	-1.900090	-0.428551
2016-07-03	-0.216422	0.659557	0.342833	-0.141637
2016-07-01	1.024359	0.213159	0.277114	-0.304599
2016-07-05	-0.923094	-1.174652	2.678657	-0.663322
2016-07-04	-0.873492	0.669932	-0.515944	0.280424

```
# 내림차순으로는?

df2.sort_index(axis=1, ascending=False)
```

	D	C	B	A
2016-07-02	0.067814	-1.620676	0.619885	-1.693934
2016-07-06	-0.428551	-1.900090	0.539906	-1.580576
2016-07-03	-0.141637	0.342833	0.659557	-0.216422
2016-07-01	-0.304599	0.277114	0.213159	1.024359
2016-07-05	-0.663322	2.678657	-1.174652	-0.923094
2016-07-04	0.280424	-0.515944	0.669932	-0.873492

```
# 값 기준 정렬하기
# D열의 값이 오름차순이 되도록 정렬하기

df2.sort_values(by='D')
```

	D	B	C	A
2016-07-05	-0.663322	-1.174652	2.678657	-0.923094
2016-07-06	-0.428551	0.539906	-1.900090	-1.580576
2016-07-01	-0.304599	0.213159	0.277114	1.024359
2016-07-03	-0.141637	0.659557	0.342833	-0.216422
2016-07-02	0.067814	0.619885	-1.620676	-1.693934
2016-07-04	0.280424	0.669932	-0.515944	-0.873492

```
# B열의 값이 내림차순이 되도록 정렬하기

df2.sort_values(by='B', ascending=False)
```

	D	B	C	A
2016-07-04	0.280424	0.669932	-0.515944	-0.873492
2016-07-03	-0.141637	0.659557	0.342833	-0.216422
2016-07-02	0.067814	0.619885	-1.620676	-1.693934
2016-07-06	-0.428551	0.539906	-1.900090	-1.580576
2016-07-01	-0.304599	0.213159	0.277114	1.024359
2016-07-05	-0.663322	-1.174652	2.678657	-0.923094

```
df2["E"] = np.random.randint(0, 6, size=6)
df2["F"] = ["alpha", "beta", "gamma", "gamma", "alpha", "gamma"
df2
```

	D	B	C	A	E	F
2016-07-02	0.067814	0.619885	-1.620676	-1.693934	1	alpha
2016-07-06	-0.428551	0.539906	-1.900090	-1.580576	1	beta
2016-07-03	-0.141637	0.659557	0.342833	-0.216422	3	gamma
2016-07-01	-0.304599	0.213159	0.277114	1.024359	4	gamma
2016-07-05	-0.663322	-1.174652	2.678657	-0.923094	5	alpha
2016-07-04	0.280424	0.669932	-0.515944	-0.873492	0	gamma

```
# E열과 F열을 동시에 고려하여, 오름차순으로 하려면?

df2.sort_values(by=['E','F'])
```

	D	B	C	A	E	F
2016-07-04	0.280424	0.669932	-0.515944	-0.873492	0	gamma
2016-07-02	0.067814	0.619885	-1.620676	-1.693934	1	alpha
2016-07-06	-0.428551	0.539906	-1.900090	-1.580576	1	beta
2016-07-03	-0.141637	0.659557	0.342833	-0.216422	3	gamma
2016-07-01	-0.304599	0.213159	0.277114	1.024359	4	gamma
2016-07-05	-0.663322	-1.174652	2.678657	-0.923094	5	alpha

```
# 지정한 행 또는 열에서 중복값을 제외한 유니크한 값만 얻기

df2['F'].unique()

>> array(['alpha', 'beta', 'gamma'], dtype=object)
```

```
# 지정한 행 또는 열에서 값에 따른 개수 얻기

df2['F'].value_counts()

>> gamma 3
>> alpha 2
>> beta 1
>> Name: F, dtype: int64
```

```
# 지정한 행 또는 열에서 입력한 값이 있는지 확인하기

df2['F'].isin(['alpha','beta'])

# 아래와 같이 응용할 수 있다.

>> 2016-07-02 True
>> 2016-07-06 True
>> 2016-07-03 False
>> 2016-07-01 False
>> 2016-07-05 True
>> 2016-07-04 False
>> Name: F, dtype: bool
```

```
# F열의 값이 alpha나 beta인 모든 행 구하기

df2.loc[df2['F'].isin(['alpha','beta']),:]
```

	D	B	C	A	E	F
2016-07-02	0.067814	0.619885	-1.620676	-1.693934	1	alpha
2016-07-06	-0.428551	0.539906	-1.900090	-1.580576	1	beta
2016-07-05	-0.663322	-1.174652	2.678657	-0.923094	5	alpha

사용자가 직접 만든 함수를 적용하기

```
df3 = pd.DataFrame(
            np.random.randn(4, 3),
            columns=["b", "d", "e"],
            index=["Seoul", "Incheon", "Busan", "Daegu"])
```

	b	d	e
Seoul	0.806401	-0.101789	-0.201956
Incheon	0.666876	-2.144428	-0.013760
Busan	1.811228	0.225317	1.445495
Daegu	-0.608445	0.128199	-0.325935

```
func = lambda x: x.max() - x.min()
```

```
df3.apply(func, axis=0)

>> b 2.419673
>> d 2.369745
>> e 1.771431
>> dtype: float64
```