# Pandas

# Content

- Pandas 개요
- Pandas 대표적인 특징
- Pandas 설치 및 임포트
- Series 생성하기
- values, index, dtyle 속성 알아보기
- Series의 인덱스 지정하기
- Series의 인덱스 다시 지정하기
- 딕셔너리를 Series로 정의하기
- Series 이름과 index 이름의 대표 이름 재정의
- 퀴즈 - 시리즈 생성하기

# Content

# Content

# Content

# Pandas 개요

- 데이터 처리와 분석을 위한 파이썬 라이브러리
- R의 data.frame을 본떠서 설계한 DataFrame이라는 데이터 구조를 기반으로 제작
- 각 열의 데이터 타입이 달라도 되며 엑셀의 스프레드시트와 비슷한 테이블 형태 제공. 파이썬계의 엑셀
- http://pandas.pydata.org

  https://pandas.pydata.org/pandas-docs/stable/10min.html

# Pandas 대표적인 특징

- 자동적으로 혹은 명시적으로 축의 이름에 따라 데이터를 정렬할 수 있는 자료구조 통합된 시계열 기능
- 시계열 데이터와 비시계열 데이터를 함께 다룰 수 있는 통합 자료구조
- 누락된 데이터를 유연하게 처리할 수 있는 기능
- SQL과 같은 일반 데이터베이스처럼 데이터를 합치고 관계연산을 수행하는 기능
- 산술연산과 한 축의 모든 값을 더하는 등의 데이터 축약연산

# Pandas 자료 구조

- **Pandas의 2개의 자료구조**
  - 시리즈(Series)

    : 리스트와 딕셔너리 두가지의 장점을 섞어놓은 듯한 자료구조
  - 데이타프레임(DataFrame)

    : Row와 Column으로 이뤄진 2차원 형태의 자료구조. 시리즈(Series)의 결합체

# Pandas 자료 구조

| | 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

# Pandas 설치 및 임포트

- Pandas 는 외부 모듈
- 터미널 창을 이용하여 아래와 같이 설치한다.

```
pip install pandas
```

- Pandas 임포트 : 별칭 pd 이용

```
import pandas as pd
```

- Pandas 버전 확인

```
pandas.__version__
```

```
pd.__version__
```

# Series

Series : 인덱스와 값이 자동으로 생성되는 데이터 단위
: 1차원 배열

- 정수형 리스트 → 시리즈로 생성

시리즈이름 = pd.Series([리스트])

```
import numpy as np
import pandas as pd
obj = pd.Series([4, 7, -5, 3])
obj
```

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

# Series

Series : 인덱스와 값이 자동으로 생성되는 데이터 단위
: 1차원 배열

- 딕셔너리 → 시리즈로 생성

시리즈이름 = pd.Series(딕셔너리)

```
import numpy as np
import pandas as pd
# 딕셔너리로 시리즈 생성
myDict = {'a':'apple','b':'banana','c':'cat'}
mySeries = pd.Series(myDict)
mySeries
```

```
a     apple
b    banana
c       cat
dtype: object
```

# Series

Series : 인덱스와 값이 자동으로 생성되는 데이터 단위

- 정수형 리스트 → 시리즈로 생성

```
import numpy as np
import pandas as pd
obj = pd.Series([4, 7, -5, 3])
obj.values
obj.index
obj.dtype
```

```
array([ 4,  7, -5,  3], dtype=int64)

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

dtype('int64')
```

시리즈이름 = pd.Series([리스트])

시리즈이름 .values 값

시리즈이름 .index 인덱스

시리즈이름 .dtype 데이터 타입

# Series

Series : 인덱스와 값이 자동으로 생성되는 데이터 단위

- 정수형 리스트 → 인덱스 지정 시리즈

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

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

시리즈이름 = pd.Series([리스트], index=[인덱스트리스트])

# Series

## Series 의 인덱스 지정하기

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

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

```
obj.index=['d', 'b', 'a', 'c']
obj
```

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

시리즈이름 . index=[인덱스트리스트]

# Series

Series : 인덱스와 값이 자동으로 생성되는 데이터 단위

- 딕셔너리 → 시리즈 생성 (키 값이 index로 지정됨)

```
sdata = {"Charles": 35000, "Julia":71000,
"Hayoung":16000, "Sangjae":5000}
obj3 = pd.Series(sdata)
obj3
```

시리즈이름 = pd.Series(딕셔너리)

```
Charles     35000
Hayoung     16000
Julia       71000
Sangjae     5000
dtype: int64
```

# Series

## Series 이름과 index 이름의 대표 이름 재정의

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

시리즈이름.name = 이름
시리즈이름.index.name = 인덱스이름

```
Names
Charles      35000
Hayoung    16000
Kilho         71000
Sangjae     5000
Name: Salary, dtype: int64
```

# Series

## 퀴즈

```
과목
국어    90
수학    80
영어   100
과학    55
역사    70
Name: 중간고사 성적표, dtype: int64
```

# Series

## 퀴즈

```
grade = pd.Series([90,80,100,55,70])
grade.index = ['국어','수학','영어','과학','역사']
grade.name = '중간고사 성적표'
grade.index.name = '과목'
grade
```

# Dataframe

## Dataframe : 행과 열의 자료 구조

딕셔너리 - > 데이타프레임

딕셔너리 정의
데이타프레임이름 = pd.DataFrame(딕셔너리이름)

```
data = {"name":['Elise', 'Julia', 'Jhon', 'Charles', 'Charles'],
        "year":[2014, 2015, 2016, 2017, 2018],
        "points":[1.5, 1.7, 3.6, 2.5, 2.9]}
df = pd.DataFrame(data)
df
```

| | name | points | year |
|---|---|---|---|
| 0 | Elise | 1.5 | 2014 |
| 1 | Julia | 1.7 | 2015 |
| 2 | Jhon | 3.6 | 2016 |
| 3 | Charles | 2.5 | 2017 |
| 4 | Charles | 2.9 | 2018 |

# DataFrame 생성하기 - Nested Dictionary 이용

이중 딕셔너리 - > 데이타프레임

딕셔너리 정의 { key 1: {key2-1:value1. key2-2:value2 … }}
데이타프레임이름 = pd.DataFrame(딕셔너리이름)

```
pop =  {  'Nevada':{2001:2.4, 2002:2.9},
          'Ohio':{2000:1.5, 2001:1.7, 2002:3.6} }

df = pd.DataFrame(pop)
df
```

| | Nevada | Ohio |
|---|---|---|
| **2000** | NaN | 1.5 |
| **2001** | 2.4 | 1.7 |
| **2002** | 2.9 | 3.6 |

# dataframe

## Dataframe : 행과 열의 자료 구조

데이타프레임의 행 요소 조회 – 데이타프레임이름.index
데이타프레임의 열 요소 조회 – 데이타프레임이름.columns
데이타프레임의 요소 값 조회 – 데이타프레임이름.values

```
df.index
df.columns
df.values
```

```
RangeIndex(start=0, stop=5, step=1)
Index(['name', 'points', 'year'], dtype='object')
array([['Elise', 1.5, 2014],
       ['Julia', 1.7, 2015],
       ['Jhon', 3.6, 2016],
       ['Charles', 2.5, 2017],
       ['Charles', 2.9, 2018]], dtype=object)
```

| | name | points | year |
|---|---|---|---|
| 0 | Elise | 1.5 | 2014 |
| 1 | Julia | 1.7 | 2015 |
| 2 | Jhon | 3.6 | 2016 |
| 3 | Charles | 2.5 | 2017 |
| 4 | Charles | 2.9 | 2018 |

# dataframe

## Dataframe : 행과 열의 자료 구조

데이타프레임의 행이름 정의 – 데이타프레임이름.index.name
데이타프레임의 열이름 정의– 데이타프레임이름.columns.name

```
df.index.name = "Num“
df.columns.name = "Info“
df
```

| Info<br>Num | name | points | year |
|---|---|---|---|
| 0 | Elise | 1.5 | 2014 |
| 1 | Julia | 1.7 | 2015 |
| 2 | Jhon | 3.6 | 2016 |
| 3 | Charles | 2.5 | 2017 |
| 4 | Charles | 2.9 | 2018 |

# dataframe

## DataFrame 생성하기 - 인덱스명, 컬럼명

데이타프레임이름 = pd.DataFrame(딕셔너리, columns=[컬럼리스트],
index=[인덱스리스트])

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

| | year | name | points |
|---|---|---|---|
| one | 2014 | Elise | 1.5 |
| two | 2015 | Julia | 1.7 |
| three | 2016 | Jhon | 3.6 |
| four | 2017 | Charles | 2.5 |
| five | 2018 | Charles | 2.9 |

# dataframe

## Dataframe 의 재정의 – 새로운 컬럼 추가

데이타프레임이름2 = pd.DataFrame(데이타프레임이름1, columns=[컬럼리스트], index=[인덱스리스트])

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

| | year | name | points | penalty |
|---|---|---|---|---|
| one | 2014 | Elise | 1.5 | NaN |
| two | 2015 | Julia | 1.7 | NaN |
| three | 2016 | Jhon | 3.6 | NaN |
| four | 2017 | Charles | 2.5 | NaN |
| five | 2018 | Charles | 2.9 | NaN |

NaN
- Not a Number
- 값이 정의되지 않음

# dataframe

## Dataframe 정보 알아보기

데이타프레임이름.describe()
총합, 평균, 표준편차,최소값을 자동으로 출력

df2.describe()

| | year | points |
|---|---|---|
| count | 5.000000 | 5.00000 |
| mean | 2016.000000 | 2.44000 |
| std | 1.581139 | 0.86487 |
| min | 2014.000000 | 1.50000 |
| 25% | 2015.000000 | 1.70000 |
| 50% | 2016.000000 | 2.50000 |
| 75% | 2017.000000 | 2.90000 |
| max | 2018.000000 | 3.60000 |

# Dataframe Index

## Dataframe 열 필터링

df[컬럼명]
df.컬럼명
컬럼 추출시 시리즈(Series) 표시

```
data = {"name":["Haidi", "Haidi", "Haidi", "Charles", "Charles"],
      "year":[2014, 2015, 2016, 2015, 2017],
      "points":[1.5, 1.7, 3.6, 2.5, 2.9]}
df = pd.DataFrame(data, columns=["year", "name", "points",
"penalty"],
                  index=["one", "two", "three", "four", "five"])
df
```

# Dataframe Index

## Dataframe 열 필터링

df[컬럼명]
df.컬럼명
컬럼 추출시 시리즈(Series) 표시

```
df["year"]
df.year
```

| | year | name | points | penalty |
|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | NaN |
| two | 2015 | Haidi | 1.7 | NaN |
| three | 2016 | Haidi | 3.6 | NaN |
| four | 2015 | Charles | 2.5 | NaN |
| five | 2017 | Charles | 2.9 | NaN |

```
one      2014
two      2015
three    2016
four     2015
five     2017
Name: year, dtype: int64
```

# Dataframe Index

## Dataframe 열 필터링 - 다중

df[[컬럼명1, 컬럼명2...]]

df[["year", "points"]]

| | year | name | points | penalty |
|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | NaN |
| two | 2015 | Haidi | 1.7 | NaN |
| three | 2016 | Haidi | 3.6 | NaN |
| four | 2015 | Charles | 2.5 | NaN |
| five | 2017 | Charles | 2.9 | NaN |

| | year | points |
|---|---|---|
| one | 2014 | 1.5 |
| two | 2015 | 1.7 |
| three | 2016 | 3.6 |
| four | 2015 | 2.5 |
| five | 2017 | 2.9 |

# Dataframe Index

## Dataframe 열에 초기값 입력하기

df[컬럼명] = 값 : 같은값으로 초기화
df[컬럼명] = [값1, 값2 ... ] : 리스트형태로 나열

```
df["penalty"] = 0.5
df
```

| | year | name | points | penalty |
|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | NaN |
| two | 2015 | Haidi | 1.7 | NaN |
| three | 2016 | Haidi | 3.6 | NaN |
| four | 2015 | Charles | 2.5 | NaN |
| five | 2017 | Charles | 2.9 | NaN |

| | year | name | 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 | 2.5 | 0.5 |
| five | 2016 | Charles | 2.9 | 0.5 |

# Dataframe Index

## Dataframe 열에 초기값 입력하기

df[컬럼명] = 값 : 같은값으로 초기화
df[컬럼명] = [값1, 값2 ... ] : 리스트형태로 나열

```
df["penalty"] = [0.1, 0.2, 0.3,0.4, 0.5]
df
```

| | year | name | points | penalty |
|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | NaN |
| two | 2015 | Haidi | 1.7 | NaN |
| three | 2016 | Haidi | 3.6 | NaN |
| four | 2015 | Charles | 2.5 | NaN |
| five | 2017 | Charles | 2.9 | NaN |

| | year | name | points | penalty |
|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | 0.1 |
| two | 2015 | Haidi | 1.7 | 0.2 |
| three | 2016 | Haidi | 3.6 | 0.3 |
| four | 2015 | Charles | 2.5 | 0.4 |
| five | 2017 | Charles | 2.9 | 0.5 |

# Dataframe Index

## Dataframe 열 추가와 증가값 지정

df[컬럼명] = np.arrange(최종숫자)

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

| | year | name | points | penalty | zeros |
|---|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | 0.1 | 0 |
| two | 2015 | Haidi | 1.7 | 0.2 | 1 |
| three | 2016 | Haidi | 3.6 | 0.3 | 2 |
| four | 2015 | Charles | 2.5 | 0.4 | 3 |
| five | 2017 | Charles | 2.9 | 0.5 | 4 |

# Series 이용하여 컬럼 추가하기

## Dataframe 열 추가시 특정 행만 값 지정하기

시리즈(Series) 생성시 행의 index 이름값 사용
생성된 시리즈를 컬럼 추가시 사용

```
data = {"name":['Elise', 'Julia', 'Jhon', 'Charles', 'Charles'],
      "year":[2014, 2015, 2016, 2017, 2018],
      "points":[1.5, 1.7, 3.6, 2.5, 2.9],
      "penalty":[1,2,3,4,5]}
df = pd.DataFrame(data, columns=["year","name","points","penalty"],
              index=["one", "two", "three", "four", "five"])
df
```

| | year | name | points | penalty |
|---|---|---|---|---|
| **one** | 2014 | Elise | 1.5 | 1 |
| **two** | 2015 | Julia | 1.7 | 2 |
| **three** | 2016 | Jhon | 3.6 | 3 |
| **four** | 2017 | Charles | 2.5 | 4 |
| **five** | 2018 | Charles | 2.9 | 5 |

# Series 이용하여 컬럼 추가하기

## Dataframe 열 추가시 특정 행만 값 지정하기

시리즈(Series) 생성시 행의 index 이름값 사용
생성된 시리즈를 컬럼 추가시 사용

```
# 시리즈 생성하기
s = pd.Series([-1.2, -1.5, -1.7], index=["two", "four", "five"])
# Series를 데이타프레임의 새컬럼에 추가하기
df["debt"] = s
df
```

| | year | name | points | penalty | debt |
|---|---|---|---|---|---|
| **one** | 2014 | Elise | 1.5 | 1 | NaN |
| **two** | 2015 | Julia | 1.7 | 2 | -1.2 |
| **three** | 2016 | Jhon | 3.6 | 3 | NaN |
| **four** | 2017 | Charles | 2.5 | 4 | -1.5 |
| **five** | 2018 | Charles | 2.9 | 5 | -1.7 |

# Dataframe Index

## Dataframe 기존 열의 값 이용하여 새로운 열 추가하기

df[컬럼명] = df[컬럼1] - df[컬럼2]
df[컬럼명] = df[컬럼]을 이용한 비교연산자

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

| | year | name | points | penalty | zeros | debt |
|---|---|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | 0.1 | 0 | NaN |
| two | 2015 | Haidi | 1.7 | 0.2 | 1 | -1.2 |
| three | 2016 | Haidi | 3.6 | 0.3 | 2 | NaN |
| four | 2015 | Charles | 2.5 | 0.4 | 3 | -1.5 |
| five | 2017 | Charles | 2.9 | 0.5 | 4 | -1.7 |

| | year | name | points | penalty | zeros | debt | net_points | high_points |
|---|---|---|---|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | 0.1 | 0 | NaN | 1.4 | False |
| two | 2015 | Haidi | 1.7 | 0.2 | 1 | -1.2 | 1.5 | False |
| three | 2016 | Haidi | 3.6 | 0.3 | 2 | NaN | 3.3 | True |
| four | 2015 | Charles | 2.5 | 0.4 | 3 | -1.5 | 2.1 | True |
| five | 2017 | Charles | 2.9 | 0.5 | 4 | -1.7 | 2.4 | True |

# Dataframe Index

## Dataframe : 열 삭제

### del df[컬럼명]

```
del df["high_points"]
del df["debt"]
del df[“zeros"]
df
```

| | year | name | points | penalty | zeros | debt | net_points | high_points |
|---|---|---|---|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | 0.1 | 0 | NaN | 1.4 | False |
| two | 2015 | Haidi | 1.7 | 0.2 | 1 | -1.2 | 1.5 | False |
| three | 2016 | Haidi | 3.6 | 0.3 | 2 | NaN | 3.3 | True |
| four | 2015 | Charles | 2.5 | 0.4 | 3 | -1.5 | 2.1 | True |
| five | 2017 | Charles | 2.9 | 0.5 | 4 | -1.7 | 2.4 | True |

| | year | name | points | penalty | net_points |
|---|---|---|---|---|---|
| one | 2014 | Haidi | 1.5 | 0.1 | 1.4 |
| two | 2015 | Haidi | 1.7 | 0.2 | 1.5 |
| three | 2016 | Haidi | 3.6 | 0.3 | 3.3 |
| four | 2015 | Charles | 2.5 | 0.4 | 2.1 |
| five | 2017 | Charles | 2.9 | 0.5 | 2.4 |

# Dataframe Index

## Dataframe : 인덱스, 컬럼, 값 확인하기

데이타프레임이름.columns
데이타프레임이름.index
데이타프레임이름.values

df.columns
df.index
df.values

```
Index(['year', 'name', 'points', 'penalty', 'net_points'], dtype='object')
Index(['one', 'two', 'three', 'four', 'five'], dtype='object')
array([[2014, 'Haidi', 1.5, 0.1, 1.4],
       [2015, 'Haidi', 1.7, 0.2, 1.5],
       [2016, 'Haidi', 3.6, 0.3, 3.3000000000000003],
       [2015, 'Charles', 2.5, 0.4, 2.1],
       [2017, 'Charles', 2.9, 0.5, 2.4]], dtype=object)
```

# Dataframe Index

## Dataframe : 열과 행 이름 정의

데이타프레임이름.index.name = 행전체이름
데이타프레임이름.columns.name = 컬럼전체이름

```
df.index.name = "Order"
df.columns.name = "Info“
df
```

| Info<br>Order | year | name | points | penalty |
|---|---|---|---|---|
| one | 2014 | Elise | 1.5 | 1 |
| two | 2015 | Julia | 1.7 | 2 |
| three | 2016 | Jhon | 3.6 | 3 |
| four | 2017 | Charles | 2.5 | 4 |
| five | 2018 | Charles | 2.9 | 5 |

# Dataframe Index

## Dataframe : 행 추출

데이타프레임이름[start:end]
인덱스는 행 이름이나 숫자로 가능

df[0:3]

| Info | year | name | points | penalty | net_points |
|---|---|---|---|---|---|
| Order | | | | | |
| one | 2014 | Haidi | 1.5 | 0.1 | 1.4 |
| two | 2015 | Haidi | 1.7 | 0.2 | 1.5 |
| three | 2016 | Haidi | 3.6 | 0.3 | 3.3 |

# Dataframe Index

## Dataframe : 행 추출

데이타프레임이름[start:end]
인덱스는 행 이름이나 숫자로 가능

```
df["two":"four"]
```

| Info Order | year | name | points | penalty | net_points |
|---|---|---|---|---|---|
| two | 2015 | Haidi | 1.7 | 0.2 | 1.5 |
| three | 2016 | Haidi | 3.6 | 0.3 | 3.3 |
| four | 2015 | Charles | 2.5 | 0.4 | 2.1 |

# Dataframe Index

## Dataframe : 행 추출. loc 이용. Index Location

데이타프레임이름.loc[start:end]
인덱스는 행 이름으로 가능

df.loc["two"]

```
Info
year          2015
name          Haidi
points         1.7
penalty         0.2
net_points      1.5
Name: two, dtype: object
```

df.loc["two":"four"]

| Info Order | year | name | points | penalty | net_points |
|---|---|---|---|---|---|
| two | 2015 | Haidi | 1.7 | 0.2 | 1.5 |
| three | 2016 | Haidi | 3.6 | 0.3 | 3.3 |
| four | 2015 | Charles | 2.5 | 0.4 | 2.1 |

# Dataframe Index

dataframe : 특정 행의 특정 열 추출. loc 이용

df.loc[행인덱스start:행인덱스end, 컬럼명]
데이타프레임에서 start와 end-1까지의 행 추출시 컬럼 값만 추출

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

```
Order
two      1.7
three    3.6
four     2.5
Name: points, dtype: float64
```

# Dataframe Index

## dataframe : loc 이용. 컬럼 추출

데이타프레임이름.loc[ : , [컬럼명1, 컬럼명2] ]
데이타프레임이름.loc[행인덱스1:행인덱스2, [컬럼명1, 컬럼명2] ]

```
df.loc[:, "year"]
```

```
Order
one      2014
two      2015
three    2016
four     2015
five     2017
Name: year, dtype: int64
```

# Dataframe Index

## Dataframe : Loc 이용. 컬럼 추출

데이타프레임이름.loc[ : , [컬럼명1, 컬럼명2] ]
데이타프레임이름.loc[행인덱스1:행인덱스2, [컬럼명1, 컬럼명2] ]

```
df.loc[:, ["year","name"]]
```

| Info<br>Order | year | name |
|---|---|---|
| one | 2014 | Haidi |
| two | 2015 | Haidi |
| three | 2016 | Haidi |
| four | 2015 | Charles |
| five | 2017 | Charles |

# Dataframe Index

## Dataframe : Loc 이용. 컬럼 추출

데이타프레임이름.loc[ : , [컬럼명1, 컬럼명2] ]
데이타프레임이름.loc[행인덱스1:행인덱스2, [컬럼명1, 컬럼명2] ]

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

| Info Order | year | name | points | penalty |
|---|---|---|---|---|
| three | 2016 | Haidi | 3.6 | 0.3 |
| four | 2015 | Charles | 2.5 | 0.4 |
| five | 2017 | Charles | 2.9 | 0.5 |

# Dataframe Index

## Dataframe : loc 이용. 행 추가

데이타프레임이름.loc [ 행인덱스명, : ] = [값1, 값2 ...]

```
df.loc["six",:] = [2013, "hayoung", 4.0, 6]
df
```

| Info Order | year | name | points | penalty |
|---|---|---|---|---|
| **one** | 2014.0 | Elise | 1.5 | 1.0 |
| **two** | 2015.0 | Julia | 1.7 | 2.0 |
| **three** | 2016.0 | Jhon | 3.6 | 3.0 |
| **four** | 2017.0 | Charles | 2.5 | 4.0 |
| **five** | 2018.0 | Charles | 2.9 | 5.0 |
| **six** | 2013.0 | hayoung | 4.0 | 6.0 |

# Dataframe Index

## Dataframe : loc 이용. 열 추가

데이타프레임이름.loc [ :, 열인덱스명 ] = [값1, 값2 ...]

```
df.loc[:,'bonus']=[10,20,30,40,50,60]
df
```

| Info Order | year | name | points | penalty | bonus |
|---|---|---|---|---|---|
| one | 2014.0 | Elise | 1.5 | 1.0 | 10 |
| two | 2015.0 | Julia | 1.7 | 2.0 | 20 |
| three | 2016.0 | Jhon | 3.6 | 3.0 | 30 |
| four | 2017.0 | Charles | 2.5 | 4.0 | 40 |
| five | 2018.0 | Charles | 2.9 | 5.0 | 50 |
| six | 2013.0 | hayoung | 4.0 | 6.0 | 60 |

# Dataframe Index

## Dataframe : iloc 이용. 행추출. Index Position

데이타프레임이름.iloc[행인덱스] : 시리즈형태로 세로로 추출

df.iloc[3]

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

```
Info
year          2015
name       Charles
points         2.5
penalty        0.4
net_points     2.1
bonus           40
Name: four, dtype: object
```

# Dataframe Index

## Dataframe : iloc 이용. 행, 컬럼 범위 추출

데이타프레임이름. iloc[ row1:row2, column1:column2]

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

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

| Info Order | year | name |
|---|---|---|
| four | 2015.0 | Charles |
| five | 2017.0 | Charles |

# Dataframe Index

## Dataframe : iloc 이용. 행, 컬럼 인덱스 번호로 추출

데이타프레임이름.iloc[ [인덱스 숫자], [컬럼인덱스 숫자 ]]

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

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

| Info Order | name | points |
|---|---|---|
| one | Haidi | 1.5 |
| two | Haidi | 1.7 |
| four | Charles | 2.5 |

# Dataframe Index

## Dataframe : iloc 이용.  컬럼 인덱스 번호로  추출

데이타프레임이름.iloc[ :, [컬럼인덱스 숫자 ]]

df.iloc[:, 1:4]

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

| Info Order | name | points | penalty |
|---|---|---|---|
| one | Haidi | 1.5 | 0.1 |
| two | Haidi | 1.7 | 0.2 |
| three | Haidi | 3.6 | 0.3 |
| four | Charles | 2.5 | 0.4 |
| five | Charles | 2.9 | 0.5 |
| six | Hayoung | 4.0 | 0.1 |

# Dataframe Index

Dataframe : iloc 이용. **특정 행과열 인덱스 이용해서 한개만 추출하기**

데이타프레임이름.iloc[행인덱스, 열인덱스]

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

'Haidi'

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

# Dataframe Boolean Index

- 데이타프레임 인덱스시 사용됨]
- 마스크(Mask)라고도 함
- 조건에 맞으면 결과값이 True/False 형태의 Boolean으로 표시
- df.loc [ df[ 컬럼인덱스] 비교연산자 ]
- df.loc [ df[ 행인덱스] 비교연산자 ]
- 다중 조건시 논리 연산자 사용

```
data = { "year":[2014, 2015, 2016, 2015, 2017,2013]
       ,"name":['Haidi', 'Haidi', 'Haidi', 'Charles', 'Charles',  'Hayoung']
       , "points":[1.5, 1.7, 3.6, 2.5, 2.9, 4.0]
       , "penalty":[0.1,0.2, 0.3,0.4,0.5,0.1 ]
       , "net_points":[1.4,1.5,3.3,2.1,2.4,2.1]
       , "bonus":[10,20,30,40,50,60]}
df = pd.DataFrame(data,
         columns=["year","name","points","penalty","net_points","bonus"],
              index=["one", "two", "three", "four", "five","six"])
```

# Dataframe Boolean Index

```
df
```

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

# Dataframe Boolean Index

- 데이타프레임 인덱스시 사용됨]
- 마스크(Mask)라고도 함
- 조건에 맞으면 결과값이 True/False 형태의 Boolean으로 표시
- df.loc [ df[ 컬럼인덱스] 비교연산자 ]
- df.loc [ df[ 행인덱스] 비교연산자 ]
- 다중 조건시 논리 연산자 사용

```
df["year"] > 2014
```

| Info<br>Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

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

# Dataframe Boolean Index

- 데이타프레임 인덱스시 사용됨]
- 마스크(Mask)라고도 함
- 조건에 맞으면 결과값이 True/False 형태의 Boolean으로 표시
- df.loc [ df[ 컬럼인덱스] 비교연산자 ]
- df.loc [ df[ 행인덱스] 비교연산자 ]
- 다중 조건시 논리 연산자 사용

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

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |

# Dataframe Boolean Index

- 데이타프레임 인덱스시 사용됨]
- 마스크(Mask)라고도 함
- 조건에 맞으면 결과값이 True/False 형태의 Boolean으로 표시
- df.loc [ df[ 컬럼인덱스] 비교연산자 ]
- df.loc [ df[ 행인덱스] 비교연산자 ]
- 다중 조건시 논리 연산자 사용

```
df.loc[df["name"]== "Charles", ["name", "points"]]
```

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

| Info Order | name | points |
|---|---|---|
| four | Charles | 2.5 |
| five | Charles | 2.9 |

# Dataframe Boolean Index

- 데이타프레임 인덱스시 사용됨]
- 마스크(Mask)라고도 함
- 조건에 맞으면 결과값이 True/False 형태의 Boolean으로 표시
- df.loc [ df[ 컬럼인덱스] 비교연산자 ]
- df.loc [ df[ 행인덱스] 비교연산자 ]
- 다중 조건시 논리 연산자 사용

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

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |

# Dataframe Boolean Index

- Boolean Indexing 처리 후에 새로운 값 입력하기

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

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.3 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.1 | 2.1 | 60 |

| Info Order | year | name | points | penalty | net_points | bonus |
|---|---|---|---|---|---|---|
| one | 2014.0 | Haidi | 1.5 | 0.1 | 1.4 | 10 |
| two | 2015.0 | Haidi | 1.7 | 0.2 | 1.5 | 20 |
| three | 2016.0 | Haidi | 3.6 | 0.0 | 3.3 | 30 |
| four | 2015.0 | Charles | 2.5 | 0.4 | 2.1 | 40 |
| five | 2017.0 | Charles | 2.9 | 0.5 | 2.4 | 50 |
| six | 2013.0 | Hayoung | 4.0 | 0.0 | 2.1 | 60 |

# Dataframe - 랜덤 숫자로 구성

- 넘파이의 랜덤 함수 이용

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

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | -0.493212 | -0.703842 | -1.789079 | -1.865409 |
| 1 | 0.529652 | -0.900887 | 0.637774 | -1.647861 |
| 2 | -1.059571 | -0.667135 | 2.036180 | -1.799026 |
| 3 | 2.056895 | -0.953720 | 0.167671 | -0.137977 |
| 4 | 0.744166 | 0.273535 | -0.176022 | -0.068019 |
| 5 | -0.785232 | 0.452872 | 0.105017 | 0.378988 |

# Dataframe - 컬럼 이름

- 컬럼 이름 지정

```
df.columns = ["A","B","C","D"]
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **0** | -0.493212 | -0.703842 | -1.789079 | -1.865409 |
| **1** | 0.529652 | -0.900887 | 0.637774 | -1.647861 |
| **2** | -1.059571 | -0.667135 | 2.036180 | -1.799026 |
| 3 | 2.056895 | -0.953720 | 0.167671 | -0.137977 |
| 4 | 0.744166 | 0.273535 | -0.176022 | -0.068019 |
| 5 | -0.785232 | 0.452872 | 0.105017 | 0.378988 |

# 랜덤 날짜 데이터

- 날짜 데이터 지정하기

pd.date_range() 함수는 날짜 생성 - 초기날짜, 단계
pd.date_range( 초기날짜, periods=숫자)

```
df_date = pd.date_range("20181201", periods=10)
df_date
```

```
DatetimeIndex(['2018-12-01', '2018-12-02', '2018-12-03', '2018-12-04',
               '2018-12-05', '2018-12-06', '2018-12-07', '2018-12-08',
               '2018-12-09', '2018-12-10'],
              dtype='datetime64[ns]', freq='D')
```

# 랜덤 날짜 데이터를 인덱스로 이용하기

- 행 인덱스에 날짜 데이터 지정하기

pd.date_range() 함수는 날짜 생성 - 초기날짜, 단계
데이타프레임.index = pd.date_range( 초기날짜, periods=숫자)

```
df = pd.DataFrame(np.random.randn(5,4),
columns=["A","B","C","D"])
df
```

| | A | B | C | D |
|---|---|---|---|---|
| 0 | -0.603380 | 1.329248 | 0.668325 | -1.826107 |
| 1 | 0.532922 | -0.055208 | -0.372515 | -0.720241 |
| 2 | 0.067975 | -0.304375 | -0.199948 | -0.662369 |
| 3 | -0.967939 | 0.165162 | 0.238184 | -0.907745 |
| 4 | -1.867466 | 1.140963 | -0.601170 | -1.129813 |

# 랜덤 날짜 데이터를 인덱스로 이용하기

- 행 인덱스에 날짜 데이터 지정하기

```
df.index = pd.date_range("20181201", periods=5)
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **2018-12-01** | -0.603380 | 1.329248 | 0.668325 | -1.826107 |
| **2018-12-02** | 0.532922 | -0.055208 | -0.372515 | -0.720241 |
| **2018-12-03** | 0.067975 | -0.304375 | -0.199948 | -0.662369 |
| **2018-12-04** | -0.967939 | 0.165162 | 0.238184 | -0.907745 |
| **2018-12-05** | -1.867466 | 1.140963 | -0.601170 | -1.129813 |

# np.nan 값 입력하기

- np.nan 값을 새로운 F 열에 특정 셀에 추가하기

```
# F컬럼 추가하기
df["F"] = [1.0, np.nan, 3.5, 6.1, np.nan]
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **2018-12-01** | -0.603380 | 1.329248 | 0.668325 | -1.826107 |
| **2018-12-02** | 0.532922 | -0.055208 | -0.372515 | -0.720241 |
| **2018-12-03** | 0.067975 | -0.304375 | -0.199948 | -0.662369 |
| **2018-12-04** | -0.967939 | 0.165162 | 0.238184 | -0.907745 |
| **2018-12-05** | -1.867466 | 1.140963 | -0.601170 | -1.129813 |

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2018-12-01** | -0.603380 | 1.329248 | 0.668325 | -1.826107 | 1.0 |
| **2018-12-02** | 0.532922 | -0.055208 | -0.372515 | -0.720241 | NaN |
| **2018-12-03** | 0.067975 | -0.304375 | -0.199948 | -0.662369 | 3.5 |
| **2018-12-04** | -0.967939 | 0.165162 | 0.238184 | -0.907745 | 6.1 |
| **2018-12-05** | -1.867466 | 1.140963 | -0.601170 | -1.129813 | NaN |

# NaN 값이 들어있는 행 삭제하기

- 데이타프레임이름.dropna(how="any")
- NaN 값이 하나라도 들어가 있는 행 삭제

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

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2018-12-01** | -0.603380 | 1.329248 | 0.668325 | -1.826107 | 1.0 |
| **2018-12-02** | 0.532922 | -0.055208 | -0.372515 | -0.720241 | NaN |
| **2018-12-03** | 0.067975 | -0.304375 | -0.199948 | -0.662369 | 3.5 |
| **2018-12-04** | -0.967939 | 0.165162 | 0.238184 | -0.907745 | 6.1 |
| **2018-12-05** | -1.867466 | 1.140963 | -0.601170 | -1.129813 | NaN |

| | A | B | C | D | F |
|---|---|---|---|---|---|
| **2018-12-01** | -0.603380 | 1.329248 | 0.668325 | -1.826107 | 1.0 |
| **2018-12-03** | 0.067975 | -0.304375 | -0.199948 | -0.662369 | 3.5 |
| **2018-12-04** | -0.967939 | 0.165162 | 0.238184 | -0.907745 | 6.1 |

# NaN 값이 모두 들어있는 행 삭제하기

- 데이타프레임이름.dropna(how="all")

```
data = {"a":[2,np.NAN,3],
        "b":[np.NAN,np.NAN,np.NAN],
        "c":[1,np.NAN,np.NAN]}
df = pd.DataFrame(data)
df
```

| | a | b | c |
|---|---|---|---|
| **0** | 2.0 | NaN | 1.0 |
| **1** | NaN | NaN | NaN |
| **2** | 3.0 | NaN | NaN |

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

| | a | b | c |
|---|---|---|---|
| **0** | 2.0 | NaN | 1.0 |
| **2** | 3.0 | NaN | NaN |

# NaN값에 특정 값 입력하기

- 데이타프레임이름. fillna(value=값)

```
data = {"a":[2,np.NAN,3],
        "b":[np.NAN,np.NAN,np.NAN],
        "c":[1,np.NAN,np.NAN]}
df = pd.DataFrame(data)
df
```

| | a | b | c |
|---|---|---|---|
| **0** | 2.0 | NaN | 1.0 |
| **1** | NaN | NaN | NaN |
| **2** | 3.0 | NaN | NaN |

```
df=df.fillna(value=3.0)
df
```

| | a | b | c |
|---|---|---|---|
| **0** | 2.0 | 3.0 | 1.0 |
| **1** | 3.0 | 3.0 | 3.0 |
| **2** | 3.0 | 3.0 | 3.0 |

# NaN값에 Boolean 마스크 실행하기

- 데이타프레임이름. isnull()

```
data = {"a":[2,np.NAN,3],
        "b":[np.NAN,np.NAN,np.NAN],
        "c":[1,np.NAN,np.NAN]}
df = pd.DataFrame(data)
df
```

| | a | b | c |
|---|---|---|---|
| 0 | 2.0 | NaN | 1.0 |
| 1 | NaN | NaN | NaN |
| 2 | 3.0 | NaN | NaN |

```
df.isnull()
```

| | a | b | c |
|---|---|---|---|
| 0 | False | True | False |
| 1 | True | True | True |
| 2 | False | True | True |

# NaN 값이 들어있는 셀의 행 추출하기

- 데이타프레임이름. loc[df.isnull()["컬럼명"], :]

```
df = pd.DataFrame(np.random.randn(5,3), columns=["A","B","C"])
df
```

| | A | B | C |
|---|---|---|---|
| 0 | 0.734335 | 1.427806 | -0.009151 |
| 1 | -0.433304 | 0.306290 | -0.629809 |
| 2 | -1.512650 | 2.025063 | 0.458015 |
| 3 | 0.911953 | -0.285603 | -1.302019 |
| 4 | 1.443901 | 0.858044 | 0.564190 |

```
df.loc[0,"A"] = np.NaN
df.loc[3,"C"] = np.NaN
```

| | A | B | C |
|---|---|---|---|
| 0 | NaN | 1.427806 | -0.009151 |
| 1 | -0.433304 | 0.306290 | -0.629809 |
| 2 | -1.512650 | 2.025063 | 0.458015 |
| 3 | 0.911953 | -0.285603 | NaN |
| 4 | 1.443901 | 0.858044 | 0.564190 |

# NaN 값이 들어있는 셀의 행 추출하기

- 데이타프레임이름. loc[df.isnull()["컬럼명"], :]

```
df = pd.DataFrame(np.random.randn(5,3), columns=["A","B","C"])
df
```

| | A | B | C |
|---|---|---|---|
| 0 | 0.734335 | 1.427806 | -0.009151 |
| 1 | -0.433304 | 0.306290 | -0.629809 |
| 2 | -1.512650 | 2.025063 | 0.458015 |
| 3 | 0.911953 | -0.285603 | -1.302019 |
| 4 | 1.443901 | 0.858044 | 0.564190 |

```
df.loc[0,"A"] = np.NaN
df.loc[3,"C"] = np.NaN
df
```

| | A | B | C |
|---|---|---|---|
| 0 | NaN | 1.427806 | -0.009151 |
| 1 | -0.433304 | 0.306290 | -0.629809 |
| 2 | -1.512650 | 2.025063 | 0.458015 |
| 3 | 0.911953 | -0.285603 | NaN |
| 4 | 1.443901 | 0.858044 | 0.564190 |

# NaN 값이 들어있는 셀의 행 추출하기

- 데이타프레임이름. loc[df.isnull()["컬럼명"], :]

```
df.isnull()["A"]
```

```
0     True
1    False
2    False
3    False
4    False
Name: A, dtype: bool
```

```
# "A" 컬럼에 NaN이 있는 행 추출
df.loc[df.isnull()["A"],:]
```

| | A | B | C |
|---|---|---|---|
| **0** | NaN | 1.427806 | -0.009151 |

# 조건에 맞는 행 삭제하기

- 데이타프레임이름. drop[ 조건식 ]
- 문자열로 입력한 날짜를 날짜타입으로 변경
  pd.to_datetime('날짜데이타')
- 데이타프레임이름.drop(pd.to_datetime('날짜데이타'))

```
df = pd.DataFrame(np.random.randn(6,5),
                  index=pd.date_range("20180701", periods=6),
                  columns=["A","B","C","D","E"])
df
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **2018-07-01** | -0.025756 | 0.936718 | -1.149670 | -0.413544 | 1.408729 |
| **2018-07-02** | 1.846310 | -2.171667 | -0.824836 | 0.600107 | 0.698927 |
| **2018-07-03** | -0.449437 | -1.333603 | 0.614591 | 0.420033 | -1.602260 |
| **2018-07-04** | -0.794409 | -0.663277 | -0.982014 | -1.328624 | 1.064660 |
| **2018-07-05** | -0.286679 | 2.690161 | -1.249347 | 0.138258 | 0.805201 |
| **2018-07-06** | -0.397141 | -0.530680 | -0.269624 | -0.804408 | 0.607596 |

# 조건에 맞는 행 삭제하기

```
df.drop(pd.to_datetime("20180701"))
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **2018-07-02** | 1.846310 | -2.171667 | -0.824836 | 0.600107 | 0.698927 |
| **2018-07-03** | -0.449437 | -1.333603 | 0.614591 | 0.420033 | -1.602260 |
| **2018-07-04** | -0.794409 | -0.663277 | -0.982014 | -1.328624 | 1.064660 |
| **2018-07-05** | -0.286679 | 2.690161 | -1.249347 | 0.138258 | 0.805201 |
| **2018-07-06** | -0.397141 | -0.530680 | -0.269624 | -0.804408 | 0.607596 |

# 조건에 맞는 행 삭제하기

```
df.drop([pd.to_datetime("20180702"),pd.to_datetime("20180704")])
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **2018-07-01** | -0.025756 | 0.936718 | -1.149670 | -0.413544 | 1.408729 |
| **2018-07-03** | -0.449437 | -1.333603 | 0.614591 | 0.420033 | -1.602260 |
| **2018-07-05** | -0.286679 | 2.690161 | -1.249347 | 0.138258 | 0.805201 |
| **2018-07-06** | -0.397141 | -0.530680 | -0.269624 | -0.804408 | 0.607596 |

# 조건에 맞는 열 삭제하기

- 데이터프레임이름.drop("컬럼명", axis=1)

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

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **2018-07-01** | -0.025756 | 0.936718 | -1.149670 | -0.413544 | 1.408729 |
| **2018-07-02** | 1.846310 | -2.171667 | -0.824836 | 0.600107 | 0.698927 |
| **2018-07-03** | -0.449437 | -1.333603 | 0.614591 | 0.420033 | -1.602260 |
| **2018-07-04** | -0.794409 | -0.663277 | -0.982014 | -1.328624 | 1.064660 |
| **2018-07-05** | -0.286679 | 2.690161 | -1.249347 | 0.138258 | 0.805201 |
| **2018-07-06** | -0.397141 | -0.530680 | -0.269624 | -0.804408 | 0.607596 |

| | A | B | C | D |
|---|---|---|---|---|
| **2018-07-01** | -0.025756 | 0.936718 | -1.149670 | -0.413544 |
| **2018-07-02** | 1.846310 | -2.171667 | -0.824836 | 0.600107 |
| **2018-07-03** | -0.449437 | -1.333603 | 0.614591 | 0.420033 |
| **2018-07-04** | -0.794409 | -0.663277 | -0.982014 | -1.328624 |
| **2018-07-05** | -0.286679 | 2.690161 | -1.249347 | 0.138258 |
| **2018-07-06** | -0.397141 | -0.530680 | -0.269624 | -0.804408 |

# 조건에 맞는 열 삭제하기

- 데이터프레임이름.drop(["컬럼명1“,”컬럼명2”], axis=1)

```
df.drop(["A","B"],axis=1)
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **2018-07-01** | -0.025756 | 0.936718 | -1.149670 | -0.413544 | 1.408729 |
| **2018-07-02** | 1.846310 | -2.171667 | -0.824836 | 0.600107 | 0.698927 |
| **2018-07-03** | -0.449437 | -1.333603 | 0.614591 | 0.420033 | -1.602260 |
| **2018-07-04** | -0.794409 | -0.663277 | -0.982014 | -1.328624 | 1.064660 |
| **2018-07-05** | -0.286679 | 2.690161 | -1.249347 | 0.138258 | 0.805201 |
| **2018-07-06** | -0.397141 | -0.530680 | -0.269624 | -0.804408 | 0.607596 |

| | C | D | E |
|---|---|---|---|
| **2018-07-01** | -1.149670 | -0.413544 | 1.408729 |
| **2018-07-02** | -0.824836 | 0.600107 | 0.698927 |
| **2018-07-03** | 0.614591 | 0.420033 | -1.602260 |
| **2018-07-04** | -0.982014 | -1.328624 | 1.064660 |
| **2018-07-05** | -1.249347 | 0.138258 | 0.805201 |
| **2018-07-06** | -0.269624 | -0.804408 | 0.607596 |

# 행또는 열 별로 합 구하기

- 데이터 프레임 생성

```
# 행 값 입력하기
data = [[1.4, np.nan],
        [7.1, -4.5],
        [np.nan, np.nan],
        [0.75, -1.3]]
# 컬럼명과 행 인덱스 명 지정
df = pd.DataFrame(data, columns=["one","two"], index=["a", "b", "c", "d"])
df
```

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

# 행또는 열 별로 합 구하기

- df.sum(axis=0)
- df.sum(axis=1)
- df[열 또는 행의 인덱스].sum(axis=0)
- df.loc[행의 인덱스].sum()

```
# 행 방향 합
df.sum(axis=0)
```

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

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

# 행또는 열 별로 합 구하기

- 행또는 열 별로 합 구하기
- df.sum(axis=0)
- df.sum(axis=1)
- df[열 또는 행의 인덱스].sum(axis=0)
- df.loc[행의 인덱스].sum()

```
# 열 방향 합
df.sum(axis=1)
```

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

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

# 행또는 열 별로 합 구하기

- 행또는 열 별로 합 구하기
- df.sum(axis=0)
- df.sum(axis=1)
- df[열 인덱스].sum(axis=0)
- df.loc[행의 인덱스].sum()

```
# 특정 열 방향 합
df["one"].sum()
```

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

9.25

# 행또는 열 별로 합 구하기

- 행또는 열 별로 합 구하기
- df.sum(axis=0)
- df.sum(axis=1)
- df[열 또는 행의 인덱스].sum(axis=0)
- df.loc[행의 인덱스].sum()

```
# df.loc[].sum() 함수 이용하기
df.loc["b"].sum()
```

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

2.5999999999999996

# 함수 적용시 NaN 은 건너뛰기

- #열 방향으로 함수 적용시 NaN 은 건너뛰기
- 데이타프레임이름.sum(axis=1, skipna=False)

```
#열 방향으로 함수 적용시 NaN 은 건너뛰기
df.sum(axis=1, skipna=False)
```

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

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

# mean(), min() 이용하기

- 데이타프레임.mean()
- 데이타프레임.max()
- NaN 값을 최소값이나 평균값으로 대체하기
- 데이타프레임이름.fillna(value=최소값또는 평균값)

```
# 열의 평균 구하기
df.mean(axis=0)
```

```
one    3.083333
two   -2.900000
dtype: float64
```

평균값

```
two_min = df.min(axis=0)["two"]
two_min
```

```
-4.5
```

최소값

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

# NaN 값을 최소값이나 평균값으로 대체하기

- 데이타프레임이름.fillna(value=최소값또는 평균값)

```
#  값 입력하기
data = [[1.4, np.nan],
        [7.1, -4.5],
        [np.nan, np.nan],
        [0.75, -1.3]]
# 컬럼명과 행 인덱스 명 지정
df = pd.DataFrame(data, columns=["one","two"],
index=["a", "b", "c", "d"])
df
```

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

```
# 열의 평균값으로 NaN 값 채우기
df.fillna(value=df.mean(axis=0)
```

| | one | two |
|---|---|---|
| a | 1.400000 | -2.9 |
| b | 7.100000 | -4.5 |
| c | 3.083333 | -2.9 |
| d | 0.750000 | -1.3 |

# NaN 값을 최소값이나 평균값으로 대체하기

- 데이타프레임이름.fillna(value=최소값또는 평균값)

```
# 각 열의 최소값으로 NaN값 채우기
df.fillna(value=df.min(axis=0))
```

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

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

# 상관계수 함수 corr(), 공분산 cov() 함수 이용하기

- 상관계수 관계
  데이타프레임이름[열1].corr(데이타프레임이름[열2])
- 공분산
  데이타프레임이름[열1].cov(데이타프레임이름[열2])

```
df = pd.DataFrame(np.random.randn(6,4),
                  columns=["A", "B", "C", "D"],
                  index=pd.date_range("20180701", periods=6))
df
```

| | A | B | C | D |
|---|---|---|---|---|
| 2018-07-01 | 0.193433 | 1.427745 | -0.001331 | 0.873322 |
| 2018-07-02 | -2.110063 | -1.727995 | 0.809566 | 0.497778 |
| 2018-07-03 | -1.874124 | 0.446541 | 1.220408 | 1.481004 |
| 2018-07-04 | -0.933158 | 0.027666 | 0.138944 | -0.117399 |
| 2018-07-05 | 0.456592 | 0.000012 | -1.398606 | 0.528272 |
| 2018-07-06 | 0.226165 | -1.317479 | 1.149914 | -1.498222 |

A,B열 상관계수 관계

```
df["A"].corr(df["B"])
```

0.3029166909584385 4

# 상관계수 함수 corr(), 공분산 cov() 함수 이용하기

```
df2 = pd.DataFrame(np.random.randn(6,4),
                   columns=["A", "B", "C", "D"],
                   index=pd.date_range("20180701", periods=6))
df2
```

| | A | B | C | D |
|---|---|---|---|---|
| 2018-07-01 | 0.193433 | 1.427745 | -0.001331 | 0.873322 |
| 2018-07-02 | -2.110063 | -1.727995 | 0.809566 | 0.497778 |
| 2018-07-03 | -1.874124 | 0.446541 | 1.220408 | 1.481004 |
| 2018-07-04 | -0.933158 | 0.027666 | 0.138944 | -0.117399 |
| 2018-07-05 | 0.456592 | 0.000012 | -1.398606 | 0.528272 |
| 2018-07-06 | 0.226165 | -1.317479 | 1.149914 | -1.498222 |

B,C열 공분산

```
df2["B"].cov(df2["C"])
```

-0.4467984743473950 6

# 상관계수 함수 corr(), 공분산 cov() 함수 이용하기

- 모든 열 사이의 상관 계수와 공분산
  데이타프레임이름.corr()
  데이타프레임이름.cov()

```
df2 = pd.DataFrame(np.random.randn(6,4),
                   columns=["A", "B", "C", "D"],
                   index=pd.date_range("20180701", periods=6))
df2
```

| | A | B | C | D |
|---|---|---|---|---|
| 2018-07-01 | 0.193433 | 1.427745 | -0.001331 | 0.873322 |
| 2018-07-02 | -2.110063 | -1.727995 | 0.809566 | 0.497778 |
| 2018-07-03 | -1.874124 | 0.446541 | 1.220408 | 1.481004 |
| 2018-07-04 | -0.933158 | 0.027666 | 0.138944 | -0.117399 |
| 2018-07-05 | 0.456592 | 0.000012 | -1.398606 | 0.528272 |
| 2018-07-06 | 0.226165 | -1.317479 | 1.149914 | -1.498222 |

```
df2.corr()
```

| | A | B | C | D |
|---|---|---|---|---|
| A | 1.000000 | -0.740872 | -0.216146 | 0.541248 |
| B | -0.740872 | 1.000000 | -0.362687 | -0.776273 |
| C | -0.216146 | -0.362687 | 1.000000 | 0.642601 |
| D | 0.541248 | -0.776273 | 0.642601 | 1.000000 |

# 상관계수 함수 corr(), 공분산 cov() 함수 이용하기

- 모든 열 사이의 상관 계수와 공분산
  데이타프레임이름.corr()
  데이타프레임이름.cov()

```
df2 = pd.DataFrame(np.random.randn(6,4),
                   columns=["A", "B", "C", "D"],
                   index=pd.date_range("20180701", periods=6))
df2
```

| | A | B | C | D |
|---|---|---|---|---|
| 2018-07-01 | 0.193433 | 1.427745 | -0.001331 | 0.873322 |
| 2018-07-02 | -2.110063 | -1.727995 | 0.809566 | 0.497778 |
| 2018-07-03 | -1.874124 | 0.446541 | 1.220408 | 1.481004 |
| 2018-07-04 | -0.933158 | 0.027666 | 0.138944 | -0.117399 |
| 2018-07-05 | 0.456592 | 0.000012 | -1.398606 | 0.528272 |
| 2018-07-06 | 0.226165 | -1.317479 | 1.149914 | -1.498222 |

```
df2.cov()
```

| | A | B | C | D |
|---|---|---|---|---|
| A | 1.282142 | 0.398303 | -0.642289 | -0.491296 |
| B | 0.398303 | 1.348480 | -0.400262 | 0.671004 |
| C | -0.642289 | -0.400262 | 0.965762 | -0.166625 |
| D | -0.491296 | 0.671004 | -0.166625 | 1.044462 |

# index와 columns 순서 무작위로 섞기

- index와 columns 순서 무작위로 섞어서 DataFrame 생성
- 컬럼/ 인덱스 섞기 :
  np.random.permutation(데이타프레임.index)
  np.random.permutation(데이타프레임.columns)
- 컬럼과 인덱스 데이터 다시 지정하기
  데이타프레임.reindex(index=인덱스데이타, columns=컬럼데이타)

```
# 날짜 인덱스와 알파벳 컬럼의 데이터 프레임 생성하기
df = pd.DataFrame(np.random.randn(6,4),
                  columns=["A", "B", "C", "D"],
                  index=pd.date_range("20190701", periods=6))
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **2019-07-01** | -0.653359 | 0.878376 | 1.004609 | 1.070637 |
| **2019-07-02** | -0.662802 | 0.773630 | 0.769807 | 0.822322 |
| **2019-07-03** | -0.275014 | -0.273363 | -1.565990 | 0.024308 |
| **2019-07-04** | -1.251825 | 1.051170 | 0.879373 | 0.991912 |
| **2019-07-05** | -0.352138 | 1.014545 | 0.554070 | -3.647786 |
| **2019-07-06** | 0.518615 | -0.093021 | 1.606075 | -0.268321 |

# index와 columns 순서 무작위로 섞기

```
# 날짜 인덱스 섞기 : np.random.permutation(데이타프레임.index)
random_date = np.random.permutation(df.index)
random_date
```

```
array(['2019-07-03T00:00:00.000000000', '2019-07-01T00:00:00.000000000',
       '2019-07-04T00:00:00.000000000', '2019-07-06T00:00:00.000000000',
       '2019-07-05T00:00:00.000000000', '2019-07-02T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

```
# 컬럼 인덱스 섞기 : np.random.permutation(데이타프레임.columns)
random_columns = np.random.permutation(df.columns)
random_columns
```

```
array(['C', 'A', 'B', 'D'], dtype=object)
```

# index와 columns 순서 무작위로 섞기

```
# 날짜 인덱스 삽입과 컬럼명 다시 지정하기
#데이타프레임.reindex(index=인덱스데이타, columns=컬럼데이타)

df2 = df.reindex(index=random_date, columns=random_columns)
df2
```

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |

# 행별 열별로 정렬하기

- 데이타프레임이름.sort_index(axis=0)
- 데이타프레임이름 .sort_index(axis=1)
- 데이타프레임이름.sort_index(axis=0/1, ascending=False)

df2

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |

# 행별 열별로 정렬하기

```
# 행기준으로 정렬하기
df2.sort_index(axis=0)
```

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |

```
# 행기준으로 정렬하기 - 내림차순
df2.sort_index(axis=0, ascending=False))
```

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |

# 행별 열별로 정렬하기

```
# 열기준으로 정렬하기
df2.sort_index(axis=1)
```

| | A | B | C | D |
|---|---|---|---|---|
| **2019-07-03** | -0.275014 | -0.273363 | -1.565990 | 0.024308 |
| **2019-07-01** | -0.653359 | 0.878376 | 1.004609 | 1.070637 |
| **2019-07-04** | -1.251825 | 1.051170 | 0.879373 | 0.991912 |
| **2019-07-06** | 0.518615 | -0.093021 | 1.606075 | -0.268321 |
| **2019-07-05** | -0.352138 | 1.014545 | 0.554070 | -3.647786 |
| **2019-07-02** | -0.662802 | 0.773630 | 0.769807 | 0.822322 |

```
# 열기준으로 정렬하기 - 내림차순
df2.sort_index(axis=1, ascending=False)
```

| | D | C | B | A |
|---|---|---|---|---|
| **2019-07-03** | 0.024308 | -1.565990 | -0.273363 | -0.275014 |
| **2019-07-01** | 1.070637 | 1.004609 | 0.878376 | -0.653359 |
| **2019-07-04** | 0.991912 | 0.879373 | 1.051170 | -1.251825 |
| **2019-07-06** | -0.268321 | 1.606075 | -0.093021 | 0.518615 |
| **2019-07-05** | -3.647786 | 0.554070 | 1.014545 | -0.352138 |
| **2019-07-02** | 0.822322 | 0.769807 | 0.773630 | -0.662802 |

# 값 기준으로 정렬하기

- 데이타프레임이름. sort_values(by="컬럼명")

df2

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |

df2.sort_values(by="C")

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |

# 값 기준으로 정렬하기

- 데이타프레임이름. sort_values(by="컬럼명")

df2

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |

df2.sort_values(by="A")

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |

# 데이타프레임에 열 추가 후 값 기준으로 정렬하기

- 데이타프레임이름[컬럼명] = 컬럼데이타리스트

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

| | C | A | B | D |
|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 |

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |

# 데이타프레임에 열 추가 후 값 기준으로 정렬하기

- 값 기준으로 정렬하기
- 데이타베이스이름. sort_values(by=[컬럼명1,컬럼명2])

df2

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |

df2.sort_values(by='E')

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |

# 데이타프레임에 열 추가 후 값 기준으로 정렬하기

- 값 기준으로 정렬하기
- 데이타베이스이름. sort_values(by=[컬럼명1,컬럼명2])

df2.sort_values(by='F')

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | |

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

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |

# 중복 제거하기

- 데이타베이스이름[열이름].unique()

df2

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |

df2[“E"].unique()

array([5, 1, 2], dtype=int64)

df2["F"].unique()

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

# 특정값의 카운트

- 데이터프레임이름[열이름].value_counts()

df2

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |

df2[“E"].value_counts()

```
5    3
1    2
2    1
Name: E, dtype: int64
```

df2["F"].value_counts()

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

# isin() 함수 이용하기

- Boolean. 특정 데이터값 유무 확인하기
- 데이타프레임이름.isin([데이터값1, 데이터값2])
- 데이타프레임이름[컬럼명].isin([데이터값1, 데이터값2])
- True, False로 출력

df2

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |

df2.isin(['beta',5])

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | False | False | False | False | True | False |
| **2019-07-01** | False | False | False | False | False | True |
| **2019-07-04** | False | False | False | False | False | False |
| **2019-07-06** | False | False | False | False | True | False |
| **2019-07-05** | False | False | False | False | True | False |
| **2019-07-02** | False | False | False | False | False | False |

# isin() 함수 이용하기

df2

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |

df2['F'].isin(['gamma'])

```
2019-07-03    False
2019-07-01    False
2019-07-04     True
2019-07-06     True
2019-07-05    False
2019-07-02     True
Name: F, dtype: bool
```

# 특정 값을 가진 행 출력 : isin() 함수 이용

- 데이타프레임이름.loc[데이타프레임이름[열이름].isin([값1, 값2]),:]

df2

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-04** | 0.879373 | -1.251825 | 1.051170 | 0.991912 | 2 | gamma |
| **2019-07-06** | 1.606075 | 0.518615 | -0.093021 | -0.268321 | 5 | gamma |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |
| **2019-07-02** | 0.769807 | -0.662802 | 0.773630 | 0.822322 | 1 | gamma |

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

| | C | A | B | D | E | F |
|---|---|---|---|---|---|---|
| **2019-07-03** | -1.565990 | -0.275014 | -0.273363 | 0.024308 | 5 | alpha |
| **2019-07-01** | 1.004609 | -0.653359 | 0.878376 | 1.070637 | 1 | beta |
| **2019-07-05** | 0.554070 | -0.352138 | 1.014545 | -3.647786 | 5 | alpha |

# 람다 함수 이용하기

- 람다함수 정의 - 사용자정의함수
  - 함수명 = lambda 변수:수식 : 함수 정의
  - 데이타프레임명.apply(함수명, axis=0/1)

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

| | b | d | e |
|---|---|---|---|
| **Seoul** | -0.783617 | -1.425961 | -0.114423 |
| **Inchenon** | -0.350954 | -0.805269 | -1.400578 |
| **Budan** | -0.988832 | -0.683867 | -2.030023 |
| **Daegu** | 0.165356 | -0.100447 | 0.226286 |

# 람다 함수 이용하기

```
# 최대값에서 최소값 차이를 구하는 함수 정의
func = lambda x: x.max() - x.min()
df3.apply(func, axis=0)
```

최대값과 최소값을 뺀 결과 리턴

```
b    1.154189
d    1.325513
e    2.256309
dtype: float64
```

```
# 평균과의 차를 구하는 람다함수 정의
func_avr = lambda x: x.mean()-x
df3.apply(func_avr, axis=0)
```

| | b | d | e |
|---|---|---|---|
| **Seoul** | 0.294105 | 0.672075 | -0.715261 |
| **Inchenon** | -0.138558 | 0.051383 | 0.570893 |
| **Budan** | 0.499321 | -0.070019 | 1.200339 |
| **Daegu** | -0.654868 | -0.653439 | -1.055971 |

# Dataframe - Lending Club 데이터셋 분석

- 대출관련 csv 파일 읽어오기
- 파일 로딩 : pd.read_csv("파일경로", sep="구분자")
- shape 로 크기 확인 : 데이타프레임이름.shape
- 컬럼명 확인 : 데이타프레임이름.columns
- 5개 샘플 확인
- 필요한 컬럼만 확인
  - 데이타프레임이름.head()
  - 데이타프레임이름.tail()

```
df = pd.read_csv("data/loan.csv", sep=",")
df.shape
df.columns
df.head()
df.tail()
```

# Dataframe - Lending Club 데이터셋 분석

- 필요한 컬럼만 추출하기
- 필요한 컬럼만 추출하여 df2 변수에 저장
- 샘플 5개 추출 head()와 tail() 이용

```
df2 = df[["loan_amnt","loan_status", "grade", "int_rate","term"]]
df2.head()
```

| | loan_amnt | loan_status | grade | int_rate | term |
|---|---|---|---|---|---|
| 0 | 5000.0 | Fully Paid | B | 10.65 | 36 months |
| 1 | 2500.0 | Charged Off | C | 15.27 | 60 months |
| 2 | 2400.0 | Fully Paid | C | 15.96 | 36 months |
| 3 | 10000.0 | Fully Paid | C | 13.49 | 36 months |
| 4 | 3000.0 | Current | B | 12.69 | 60 months |

# Dataframe - Lending Club 데이터셋 분석

- 필요한 컬럼만 추출하기
- 필요한 컬럼만 추출하여 df2 변수에 저장
- 샘플 5개 추출 head()와 tail() 이용

```
df2 = df[["loan_amnt","loan_status", "grade", "int_rate","term"]]
df2.tail()
```

| | loan_amnt | loan_status | grade | int_rate | term |
|---|---|---|---|---|---|
| 887374 | 10000.0 | Current | B | 11.99 | 36 months |
| 887375 | 24000.0 | Current | B | 11.99 | 36 months |
| 887376 | 13000.0 | Current | D | 15.99 | 60 months |
| 887377 | 12000.0 | Current | E | 19.99 | 60 months |
| 887378 | 20000.0 | Current | B | 11.99 | 36 months |

# Dataframe - Lending Club 데이터셋 분석

- **중복 값 없애고 정보 표시하기**
- 데이타프레임이름[열].unique()

```
df2["loan_status"].unique()
```

```
array(['Fully Paid', 'Charged Off', 'Current', 'Default',
       'Late (31-120 days)', 'In Grace Period', 'Late (16-30 days)',
       'Does not meet the credit policy. Status:Fully Paid',
       'Does not meet the credit policy. Status:Charged Off', 'Issued'],
      dtype=object)
```

# Dataframe - Lending Club 데이터셋 분석

- **중복 값 없애고 정보 표시하기**
- 데이타프레임이름[열].unique()

```
df2["grade"].unique()
```

```
array(['B', 'C', 'A', 'E', 'F', 'D', 'G'], dtype=object)
```

```
df2["term"].unique()
```

```
array([' 36 months', ' 60 months'], dtype=object)
```

# Dataframe - Lending Club 데이터셋 분석

- **결측값 다루기**
- NaN 값이 들어있는 행 삭제하기
- dropna(how="any")

```
df2.shape
```

(887379, 5)

```
# 행에 대해서 결측값 제거
df2 = df2.dropna(how="any")
```

```
# 값이 같다면 결측값이 없다
df2.shape
```

(887379, 5)

# Dataframe - Lending Club 데이터셋 분석

- 대출 등급 확인하기
- 딕셔너리 생성
- 대출 상품의 총합계 표시하기
- 시리즈 데이터로 생성

```
term_to_loan_amt_dict  = {}  # 기간에 따른 대출 합계
uniq_terms = df2["term"].unique()
```

```
for term in uniq_terms:
    loan_amnt_sum = df2.loc[df2["term"] == term, "loan_amnt"].sum()
    term_to_loan_amt_dict[term] = loan_amnt_sum
```

```
term_to_loan_amt_dict
```

{' 36 months': 7752507375.0, ' 60 months': 5341004575.0}

# Dataframe - Lending Club 데이터셋 분석

- 대출 등급 확인하기
- 딕셔너리 생성
- 대출 상품의 총합계 표시하기
- 시리즈 데이터로 생성

```
term_to_loan_amt = pd.Series(term_to_loan_amt_dict)
term_to_loan_amt
```

```
36 months    7.752507e+09
 60 months    5.341005e+09
dtype: float64
```

# Dataframe - Lending Club 데이터셋 분석

- 우량과 불량 데이터로 구분하기

```
df2.head()
```

| | loan_amnt | loan_status | grade | int_rate | term |
|---|---|---|---|---|---|
| 0 | 5000.0 | Fully Paid | B | 10.65 | 36 months |
| 1 | 2500.0 | Charged Off | C | 15.27 | 60 months |
| 2 | 2400.0 | Fully Paid | C | 15.96 | 36 months |
| 3 | 10000.0 | Fully Paid | C | 13.49 | 36 months |
| 4 | 3000.0 | Current | B | 12.69 | 60 months |

```
df2["loan_status"].unique()
```

```
array(['Fully Paid', 'Charged Off', 'Current', 'Default',
       'Late (31-120 days)', 'In Grace Period', 'Late (16-30 days)',
       'Does not meet the credit policy. Status:Fully Paid',
       'Does not meet the credit policy. Status:Charged Off', 'Issued'],
      dtype=object)
```

# Dataframe - Lending Club 데이터셋 분석

- 우량과 불량 데이터로 구분하기

```
total_status_category = df2["loan_status"].unique()
bad_status_category = total_status_category[[1, 3, 4, 5, 6, 8]]
bad_status_category  # 불량상태
```

```
array(['Charged Off', 'Default', 'Late (31-120 days)', 'In Grace Period',
       'Late (16-30 days)',
       'Does not meet the credit policy. Status:Charged Off'],
      dtype=object)
```

```
df2["bad_loan_status"] = df2["loan_status"].isin(bad_status_category)
df2.head()
```

| | loan_amnt | loan_status | grade | int_rate | term | bad_loan_status |
|---|---|---|---|---|---|---|
| 0 | 5000.0 | Fully Paid | B | 10.65 | 36 months | False |
| 1 | 2500.0 | Charged Off | C | 15.27 | 60 months | True |
| 2 | 2400.0 | Fully Paid | C | 15.96 | 36 months | False |
| 3 | 10000.0 | Fully Paid | C | 13.49 | 36 months | False |
| 4 | 3000.0 | Current | B | 12.69 | 60 months | False |

# Dataframe - Lending Club 데이터셋 분석

- bad_loan_status 값에 따라 grade 분포 확인하기

```
bad_loan_status_to_grades = df2.loc[df2["bad_loan_status"] == True,
"grade"].value_counts()
bad_loan_status_to_grades
```

```
C    19054
D    15859
B    13456
E     9745
F     4383
A     3663
G     1269
Name: grade, dtype: int64
```

# Dataframe - Lending Club 데이터셋 분석

- bad_loan_status 값에 따라 grade 분포 확인하기

```
bad_loan_status_to_grades.sort_index()
```

```
A     3663
B    13456
C    19054
D    15859
E     9745
F     4383
G     1269
Name: grade, dtype: int64
```

# Dataframe - Lending Club 데이터셋 분석

- 상관계수 확인하기

```
df2.head()
```

| | loan_amnt | loan_status | grade | int_rate | term | bad_loan_status |
|---|---|---|---|---|---|---|
| 0 | 5000.0 | Fully Paid | B | 10.65 | 36 months | False |
| 1 | 2500.0 | Charged Off | C | 15.27 | 60 months | True |
| 2 | 2400.0 | Fully Paid | C | 15.96 | 36 months | False |
| 3 | 10000.0 | Fully Paid | C | 13.49 | 36 months | False |
| 4 | 3000.0 | Current | B | 12.69 | 60 months | False |

```
df2["loan_amnt"].corr(df2["int_rate"])
```

0.1450230992988395 5

# Dataframe - Lending Club 데이터셋 분석

- csv 파일 쓰기

```
bad_loan_status_to_grades
```

```
C    19054
D    15859
B    13456
E     9745
F     4383
A     3663
G     1269
Name: grade, dtype: int64
```

```
bad_loan_status_to_grades.to_csv("bad_loan_status.csv", sep=",")
```