
Lecture 4. Pandas 기초

기초 데이터 분석

Pandas 와 Numpy

- **Numpy와 유사하게**
 - Pandas는 대용량 데이터를 다루기 편하다
 - 여러 차원($1*N \sim N*M$)의 데이터를 다룰 수 있다.
- **하지만,**
 - Numpy는 주로 숫자 정보를 다루는 용도로 사용됨
 - Pandas는 다양한 타입의 데이터를 처리하기에 더 용이함
 - 각 특성들(Column)의 이름을 만들거나, 형태를 쉽게 변형할 수 있다.

Pandas 란?

- Pandas는 Numpy와 같이 Python에서 주로 사용되는 데이터분석 라이브러리
- Tabular 데이터를 다루기에 용이함
- 데이터는 행과 열로 정리되어 하나의 객체 단위로 사용이 가능함
 - 시리즈 (Series)
 - 데이터 프레임 (DataFrame)
- 효율적이고 빠르게 대용량 데이터를 활용할 수 있음

Import Pandas

- Pandas 설치

- Numpy와 유사하게 Anaconda를 설치 한 뒤
- Python Pip을 사용하여 설치

```
In [1,2]: 1 !pip install pandas      # pandas 설치  
          2 import pandas as pd  
          3 import numpy as np
```

- Pandas 라이브러리 불러오기

- import pandas
- 보통 Pandas 이름을 축약해서, “import pandas as pd” 로 사용

Series

Series

- 가장 기본적인 Pandas의 데이터 구조
- Numpy의 Array와 유사한 형태
- 1차원 배열로 데이터를 저장한다
- 각 데이터의 행은 Index Number를 가지고 있다.

Series 1

Mango	
0	4
1	5
2	6
3	3
4	1

Series 2

Apple	
0	5
1	4
2	3
3	0
4	2

Series

- Series 생성

- Series를 생성하기 위해서는 데이터가 필요 : (예시) [10,20,30]
- `pd.Series(data=None, Index=None)` 를 사용하여 생성
- 예시 1) `scores = pd.Series([10,20,30])`
- 예시 2) `costs = pd.Series(range(0,1000,10))`
- 예시 3) `names = pd.Series(["Tom", "John", "Jenny"])`

Series

- Series 생성

- Series를 생성하기 위해서는 데이터가 필요 : (예시) [10,20,30]
- `pd.Series(data=None, Index=None)` 를 사용하여 생성

```
In [3]: se1 = pd.Series() # 비어있는 시리즈
        se2 = pd.Series([1,2,3]) # 숫자 1,2,3 이 들어있는 시리즈
        se3 = pd.Series([[1,2,3],['a','b','c']]) # 각 [1,2,3] 과 ['a','b','c'] 가 요소인 시리즈
```

Series([], dtype: float64)	<pre>0 1 1 2 2 3 dtype: int64</pre>	<pre>0 [1, 2, 3] 1 [a, b, c] dtype: object</pre>
se1	se2	se3

Series

- 시리즈 내부에 들어가는 Type
 - 하나의 타입만 넣는 경우, type은 하나로 결정됨 : se1, se2
 - 여러개의 타입을 섞는 경우, type = 'object' : se3

```
In [3]: se1 = pd.Series() # 비어있는 시리즈
        se2 = pd.Series([1,2,3]) # 숫자 1,2,3 이 들어있는 시리즈
        se3 = pd.Series([[1,2,3],['a','b','c']]) # 각 [1,2,3] 과 ['a','b','c'] 가 요소인 시리즈
```

```
Series([], dtype: float64)
```

se1

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

se2

```
0    [1, 2, 3]
1    [a, b, c]
dtype: object
```

se3

Series

- Series 객체 활용
 - Index 지정하기
 - `pd.Series()`의 두번째 인자로 `index` 값을 넣어 행 이름을 설정
 - ex) `grades = pd.Series(data = [50,70,90], index=['Tom', 'John', 'Jenny'])`

```
In [4]: grades = pd.Series(data = [50,70,90], index=['Tom', 'John', 'Jenny'])  
grades
```

```
Tom      50  
John     70  
Jenny    90  
dtype: int64
```

Series

- Series 객체 활용

- 통계 값 구하기

- Numpy와 동일하게 통계값을 리턴하는 함수 지원
- describe(), mean(), std() 등
- 갯수, 평균, 쿼터별 값, median, 자료형 등의 정보를 돌려줌

```
In [5]: print(grades.mean()) # 평균 값
        print(grades.std()) # 표준 편차
        print(grades.count()) # 개수
        print(grades.describe()) # 전반적인 통계 정보
```

count	3.0
mean	70.0
std	20.0
min	50.0
25%	60.0
50%	70.0
75%	80.0
max	90.0
dtype:	float64

Series

- Series의 Indexing
 - 조건문(boolean)과 함께 사용하여 indexing
 - 원하는 조건을 만족하는 element만 가져온다
 - 조건문 여러개를 연결하는 것도 가능

```
In [5]: print(grades[grades>60]) # 60보다 큰 점수만 필터링
```

```
John    70  
Jenny   90  
dtype: int64
```

```
print(grades[(grades>60) & (grades<80)]) # 60보다 크고, 80보다 작은 점수만 필터링
```

```
John    70  
dtype: int64
```

DataFrame

DataFrame

- 데이터 프레임은 2차원 배열
- 각 열(Column)에 각 Series들이 모여 하나의 객체 형성
- 각 행(Row)는 각 column에 해당하는 값들을 가짐
- 예) 2번 index Row는 각각 6,3,5의 과일을 가짐

DataFrame

	Mango	Apple	Banana
0	4	5	2
1	5	4	3
2	6	3	5
3	3	0	2
4	1	2	7

DataFrame

DataFrame의 생성

- Series 생성 방법과 유사하게 `pd.DataFrame(data)`를 사용

```
In [6]: df = pd.DataFrame([10,20,30]) # 데이터 프레임 생성
df
```

	0
0	10
1	20
2	30

- 2차원 배열로 저장되기에 **Dictionary** 형태로 입력 가능
 - 각 dictionary의 value들은 모두 같은 길이를 가져야 함.

```
In [7]: df = pd.DataFrame({
          'A':[1,2,3], 'B': [10,20,30], 'C':[100,200,300]}) # Dict 형태의 데이터 입력
df
```

	A	B	C
0	1	10	100
1	2	20	200
2	3	30	300

DataFrame

DataFrame의 속성들

- `index` : Series와 동일하게 행이나 열의 위치를 나타냄
- `columns` : 열의 이름
- `shape` : DataFrame의 모양
- `dtypes` : column 들의 데이터 타입

```
In [6]: print(df.index) # index에 대한 정보
        print(df.columns) # 모든 column의 이름
        print(df.shape) # 데이터 프레임의 모양
        print(df.dtypes) # 데이터들의 타입
```

```
Index : RangeIndex(start=0, stop=3, step=1)
Columns : Index(['A', 'B', 'C'], dtype='object')
Shape : (3, 3)

Types : A      int64
       B      int64
       C      int64
dtype: object
```

DataFrame

DataFrame 불러오기

- Pandas는 외부 데이터를 데이터 프레임화해서 불러올 수 있음
- `read_파일형식(파일명)`의 함수를 사용
- csv 파일의 경우, `read_csv(파일명)`

```
In [6]: stock_df = pd.read_csv('stock-data.csv') # 예시 데이터인 주식 데이터 불러오기
stock_df.dtypes
```

```
Date      object
Close     int64
Start     int64
High      int64
Low       int64
Volume    int64
dtype: object
```

DataFrame

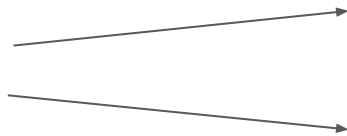
DataFrame의 내장 함수들

- `head(N)` : 데이터 프레임의 앞부분 N개를 미리 보여준다
- `tail(N)` : 데이터 프레임의 끝부분 N개를 미리 보여준다
- `describe()` : 데이터 프레임의 여러 통계 정보를 제공
- `info()` : 데이터 프레임의 컬럼별 여러 정보를 제공

In [6]:

```
print(stock_df.head())
```

```
print(stock_df.tail())
```



	Date	Close	Start	High	Low	Volume
0	2018-07-02	10100	10850	10900	10000	137977
1	2018-06-29	10700	10550	10900	9990	170253
2	2018-06-28	10400	10900	10950	10150	155769
3	2018-06-27	10900	10800	11050	10500	133548
4	2018-06-26	10800	10900	11000	10700	63039

	Date	Close	Start	High	Low	Volume
15	2018-06-08	11950	11950	12200	11800	59258
16	2018-06-07	11950	12200	12300	11900	49088
17	2018-06-05	12150	11800	12250	11800	42485
18	2018-06-04	11900	11900	12200	11700	25171
19	2018-06-01	11900	11800	12100	11750	32062

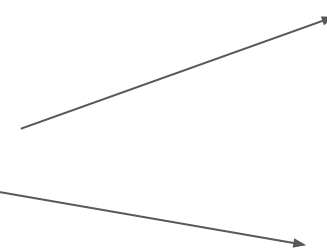
DataFrame

DataFrame의 내장 함수들

- `head(N)` : 데이터 프레임의 앞부분 N개를 미리 보여준다
- `tail(N)` : 데이터 프레임의 끝부분 N개를 미리 보여준다
- `describe()` : 데이터 프레임의 여러 통계 정보를 제공
- `info()` : 데이터 프레임의 컬럼별 여러 정보를 제공

In [6]:

```
print(stock_df.describe())
print(stock_df.info())
```



	Close	Start	High	Low	Volume
count	20.000000	20.000000	20.000000	20.000000	20.000000
mean	11662.500000	11755.000000	12015.000000	11374.500000	158014.150000
std	927.060294	865.250192	907.729962	884.369981	134473.512003
min	10100.000000	10550.000000	10900.000000	9990.000000	25171.000000
25%	11087.500000	11125.000000	11350.000000	10737.500000	58323.250000
50%	11725.000000	11800.000000	12025.000000	11500.000000	134176.500000
75%	11962.500000	12050.000000	12262.500000	11912.500000	185836.000000
max	13450.000000	13600.000000	13700.000000	13150.000000	558148.000000

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 6 columns):
#   Column  Non-Null Count  Dtype
---  ---
0   Date    20 non-null      object
1   Close   20 non-null      int64
2   Start   20 non-null      int64
3   High    20 non-null      int64
4   Low     20 non-null      int64
5   Volume  20 non-null      int64
dtypes: int64(5), object(1)
memory usage: 1.1+ KB
None
```

DataFrame

DataFrame 기본 인덱싱

- 특정 컬럼만 필터링하기
- 데이터 프레임에 원하는 column 이름 리스트를 사용해 필터링
- stock_df 의 [Date, Close, Start, High, Low, Volume] 중 앞 3개의 컬럼만 선택

```
In [6]: stock_df[['Date', 'Close', 'Start']].head() # 전체 컬럼 중, 'Date', 'Close', 'Start' 만 선택
```

	Date	Close	Start
0	2018-07-02	10100	10850
1	2018-06-29	10700	10550
2	2018-06-28	10400	10900
3	2018-06-27	10900	10800
4	2018-06-26	10800	10900

DataFrame

DataFrame 내보내기

- Pandas 데이터프레임은 외부로, 원하는 포맷으로 저장이 가능
- to_파일형식(파일 이름) : csv 파일의 경우, to_csv(파일명)
- csv, json, xlsx 등 다양한 서식으로 내보내기 가능
- stock_df 의 앞 세개 컬럼만 csv로 내보내자!

```
In [6]: new_stock = stock_df[['Date', 'Close', 'Start']] # 전체 컬럼 중, 'Date', 'Close', 'Start' 만 저장
        new_stock.to_csv('stock-data-new.csv', index=False) # 새로운 데이터프레임 csv로 추출
```

DataFrame

DataFrame 합치기

- 여러 데이터 프레임을 합쳐야 한다면?

In [6]:

```
df1 = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [10, 20, 30],
    'C': [100, 200, 300]})

df2 = pd.DataFrame({
    'A': [1, 2, 3, 4, 5, 6],
    'D': ['a', 'b', 'c', 'd', 'e', 'f']})
```

1 df1			
	A	B	C
0	1	10	100
1	2	20	200
2	3	30	300

1 df2		
	A	D
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e
5	6	f

DataFrame

DataFrame 합치기 - concat

- 여러 데이터 프레임을 이어붙여야 한다면?
- `pd.concat()` 함수를 통해 여러 데이터 프레임을 이어붙일 수 있다!
- 공유하지 않는 column은 NaN으로 설정됨
- Arguments
 - `ignore_index = False` : 인덱스를 재설정 or not
 - `axis = 0` : 세로(0), 가로(1) 방향 중 이어 붙일 곳 선택
 - `join = 'outer'` : 행, 열이 맞지 않을 경우 skip or NaN

In [6]: `pd.concat([df1, df2])` # df1과 df2 이어붙이기

	A	B	C	D
0	1	10.0	100.0	NaN
1	2	20.0	200.0	NaN
2	3	30.0	300.0	NaN
0	1	NaN	NaN	a
1	2	NaN	NaN	b
2	3	NaN	NaN	c
3	4	NaN	NaN	d
4	5	NaN	NaN	e
5	6	NaN	NaN	f

DataFrame

DataFrame 합치기 - merge

- 특정 **key**를 기준으로 여러 데이터 프레임을 합쳐야 한다면?
- `pd.merge()` 함수를 사용해 합칠 수 있다!
- **Arguments**
 - `on` : 기준 **Key**가 될 **Column**을 설정
 - `how = 'inner'` : 공통되지 않은 것들을 제거('inner'), 그대로 두고 **NaN**으로 채우기('outer'), 'left', 'right'

In [6]: `pd.merge(df1, df2, on = 'A', how = 'inner') # A 컬럼을 기준으로 df1과 df2 이어붙이기, how = inner`

	A	B	C	D
0	1	10	100	a
1	2	20	200	b
2	3	30	300	c

DataFrame

DataFrame 합치기 - merge

- 특정 key를 기준으로 여러 데이터 프레임을 합쳐야 한다면?
- `pd.merge()` 함수를 사용해 합칠 수 있다!
- Arguments
 - `on` : 기준 Key가 될 Column을 설정
 - `how` = 'inner' : 공통되지 않은 것들을 제거('inner'), 그대로 두고 NaN으로 채우기('outer'), 'left', 'right'

```
In [6]: # A 컬럼을 기준으로 df1과 df2 이어붙이기, how = outer
pd.merge(df1, df2, on = 'A', how = 'outer')
```

	A	B	C	D
0	1	10.0	100.0	a
1	2	20.0	200.0	b
2	3	30.0	300.0	c
3	4	NaN	NaN	d
4	5	NaN	NaN	e
5	6	NaN	NaN	f

DataFrame

DataFrame 합치기 - join

- Index를 기준으로 합쳐야 할때!
- join() 함수를 사용 : 단, 붙이고자 하는 데이터 프레임에 사용
 - ex. df.join(df2)
- Arguments
 - how : 교집합('inner'), 합집합('outer')