

07 정렬

[Chapter 05] 데이터 전처리

</> 1차원 객체 정렬

```
In [1]: dd = [2, 4, 3, 1]
...: dd.sort()
```

```
In [2]: dd
Out[2]: [1, 2, 3, 4]
```

```
In [3]: dd.sort(reverse = True)
...: dd
Out[3]: [4, 3, 2, 1]
```

</> 데이터 프레임 정렬

● 데이터 준비

```
1 df = pd.read_csv("bike.csv")
2 df_sub = df.iloc[:6, 9:12]
3 df_sub
```

</> 데이터 프레임 정렬

```
1 df_sub.sort_values(["casual"], ascending = [False])
2 df_sub.sort_values(["casual"], ascending = False)
3 df_sub.sort_values(["casual"], ascending = [True])
4 df_sub.sort_values(["casual"], ascending = True)
5 df_sub.sort_values(["casual"])
```

```
In [2]: df_sub.sort_values(["casual", "registered"], ascending = [False, False])
```

```
Out[2]:
```

	casual	registered	count
1	8	32	40
2	5	27	32
0	3	13	16
3	3	10	13
4	0	1	1
5	0	1	1

</> 데이터 프레임 정렬

```
1 df_sub = df_sub.sort_values(["casual", "registered"],
2                               ascending = [False, True])
3 df_sub = df_sub.reset_index(drop = True)
```

In [10]: df_sub

Out[10]:

	casual	registered	count
0	8	32	40
1	5	27	32
2	3	10	13
3	3	13	16
4	0	1	1
5	0	1	1

08 데이터 병합

[Chapter 05] 데이터 전처리

</> 색인(index)

함수	설명
reset_index()	색인을 초기화
reindex()	색인을 새로 입력한 리스트로 치환 단, 매칭이 되지 않는 색인의 경우 별도 처리 필요

```
1 import pandas as pd
2 bike = pd.read_csv("bike.csv")
3 dd = bike.sample(n = 3)
4 dd = dd.reset_index()
5 dd.reindex([0, 1, 2, 3])
```

08 데이터 병합

[Chapter 05] 데이터 전처리



단순 병합

라이브러리 및 데이터 준비

```
1 import numpy as np
2 import pandas as pd
3 df = pd.read_csv("bike.csv")
4 df_1 = df.head()
5 df_2 = df.tail()
```

row 기준 병합

```
In [2]: df_bind = pd.concat([df_1, df_2])
      ...: df_bind.shape
Out[2]: (10, 12)
```

```
1 pd.concat([df_1, df_2], axis = 1)
2 pd.concat([df_1, df_2.reset_index(drop = True)], axis = 1)
```


09 그룹 연산

[Chapter 05] 데이터 전처리

GroupBy

● 라이브러리 및 데이터 준비

```
1 import pandas as pd
2 bike = pd.read_csv("bike.csv")
```

● 모듈 활용

```
1 bike.groupby("season")["casual"].mean()
2 bike.groupby("season")["casual", "registered"].mean()
3
4 bike.groupby(["season", "holiday"])["casual"].mean()
```

09 그룹 연산

[Chapter 05] 데이터 전처리

</> 그룹별 연산과 변형

▶ Series 객체를 데이터프레임으로 변환하면 csv로 저장하기 용이하다.

```
In [2]: dd = bike.groupby("season")["casual"].mean()
....: dd = dd.to_frame()
....: type(dd)
Out[2]: pandas.core.frame.DataFrame
```

```
In [3]: dd
Out[3]:
```

	casual
season	
1	15.489576
2	47.446762
3	52.220271
4	28.580834

Melt

df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150



df3.melt(id_vars=['first', 'last'])

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

10 피보팅

[Chapter 05] 데이터 전처리

</> 피벗테이블

```
1 import pandas as pd
2 df = pd.read_csv("diamonds.csv")
3 df_sub = df.loc[:5, "x":]
4 df_sub["obs"] = df_sub.index
```

In [2]: df_sub

Out[2]:

	x	y	z	obs
0	3.95	3.98	2.43	0
1	3.89	3.84	2.31	1
2	4.05	4.07	2.31	2
3	4.20	4.23	2.63	3
4	4.34	4.35	2.75	4
5	3.94	3.96	2.48	5

In [3]: df_sub.melt(id_vars = "obs").head()

Out[3]:

	obs	variable	value
0	0	x	3.95
1	1	x	3.89
2	2	x	4.05
3	3	x	4.20
4	4	x	4.34

10 피보팅

[Chapter 05] 데이터 전처리

</> 교차일람표

● 라이브러리 및 데이터 준비

```
1 import pandas as pd
2 df = pd.read_csv("diamonds.csv")
```

● crosstab()

```
In [8]: pd.crosstab(df.cut, df.color)
```

```
Out[8]:
```

color cut	D	E	F	G	H	I	J
Fair	163	224	312	314	303	175	119
Good	662	933	909	871	702	522	307
Ideal	2834	3903	3826	4884	3115	2093	896
Premium	1603	2337	2331	2924	2360	1428	808
Very Good	1513	2400	2164	2299	1824	1204	678

11 데이터 변환

[Chapter 05] 데이터 전처리

</> 정규화(Normalization)

```
1 price = df["price"]  
2 df["price"] = (price - min(price)) / (max(price) - min(price))
```

```
In [7]: df.price.max()  
Out[7]: 1.0
```

```
In [8]: df.price.min()  
Out[8]: 0.0
```

</> 표준화(Normalization)

```
1 depth = df["depth"]  
2 df["depth"] = (depth - depth.mean()) / depth.std()
```

```
In [3]: df.depth.max()  
Out[3]: 12.041280468809875
```

```
In [4]: df.depth.min()  
Out[4]: -13.087481421530995
```

12 사용자 정의 함수

[Chapter 05] 데이터 전처리



사용자 정의 함수

- ▶ 특정 기능을 하는 코드 문치를 사용자 정의 함수로 만든다.
- ▶ 별도의 스크립트 파일에 저장하여 필요할 때 불러올 수 있다.
- ▶ 일회성 함수(lambda)를 활용하여 고급 연산을 수행할 수 있다.

User Defined Function →



스크립트 파일 실행

- ▶ 각종 환경설정 관련 코드를 별도로 관리하고자 하는 경우 다음과 같이 코드를 작성한다.

```
1 runfile("./hello.py")
```


12 사용자 정의 함수

[Chapter 05] 데이터 전처리

</> 사용자 정의 함수 예제

```
1 def udf_01(a):  
2     print(a)
```

```
In [4]: udf_01()  
Traceback (most recent call last):
```

```
File "<ipython-input-4-5913de205168>", line 1, in <module>  
    udf_01()
```

```
TypeError: udf_01() missing 1 required positional argument: 'a'
```

```
In [5]: udf_01("Hello?")  
Hello?
```


12 사용자 정의 함수

[Chapter 05] 데이터 전처리



lambda 함수

- ▶ 사용자 정의 함수 대신 사용가능한 일회성 함수
- ▶ 별도의 함수명을 지정하지 않고 사용
- ▶ 간단한 로직을 구현할 때 사용되며, 주로 pandas DataFrame의 .agg(), .apply()에 활용된다.



lambda 함수 작성

```
1 import pandas as pd
2 df = pd.read_csv("bike.csv")
3 df.groupby("season")["count"].agg(lambda x: round(x.mean()))
```

13 시간 데이터 다루기

[Chapter 05] 데이터 전처리

</> 기초 함수

● 라이브러리 및 데이터 준비

```
1 from datetime import datetime as dt
2 tm_now = dt.now()
```

● 시간 요소 추출

```
1 tm_now.year
2 tm_now.y
3 tm_now.month
4 tm_now.day
5 tm_now.hour
6 tm_now.minute
7 tm_now.second
8 tm_now.weekday()
```

<https://pandas.pydata.org/pandas-docs/stable/reference/series.html#time-series-related>

13 시간 데이터 다루기

[Chapter 05] 데이터 전처리

</> 기초 함수

● 시간차 계산

```
1 delta = dt(2019, 6, 1) - dt(2019, 1, 1, 1, 2, 3)
2 delta
3 delta.days
4 delta.seconds
```

● strptime()

```
1 dt_parsed = dt.strptime("@#2019!!12@@28", "@#%Y!!%m@@%d")
2 dt_parsed
3 dt_parsed.year
```

● origin

```
1 pd.to_datetime([1, 2, 3], unit = 'D',
2                 origin = pd.Timestamp('1960-01-01'))
```

13 시간 데이터 다루기

[Chapter 05] 데이터 전처리

</> 시간 변수 생성

```
1 from datetime import datetime as dt
2 import pandas as pd
3 df = pd.read_csv("bike.csv")
4 df["datetime"] = pd.to_datetime(df.datetime)
5 df["wday"] = df.datetime.dt.dayofweek
6 df["month"] = df.datetime.dt.month
7 df["hour"] = df.datetime.dt.hour
8 df.head()
9
```

IPython console

Console 1/A

Out[2]:

	datetime	season	holiday	...	wday	month	hour
0	2011-01-01 00:00:00	1	0	...	5	1	0
1	2011-01-01 01:00:00	1	0	...	5	1	1
2	2011-01-01 02:00:00	1	0	...	5	1	2
3	2011-01-01 03:00:00	1	0	...	5	1	3
4	2011-01-01 04:00:00	1	0	...	5	1	4

13 시간 데이터 다루기

[Chapter 05] 데이터 전처리

</> 필터링

라이브러리 및 데이터 준비

```
1 import pandas as pd
2 df = pd.read_csv("bike.csv")
```

텍스트 활용 슬라이싱

```
In [2]: df_sub = df.loc[(df["datetime"] >= "2012-01-01 00:00:00"),:]
...: df_sub.head()
```

Out[2]:

	datetime	season	holiday	...	casual	registered	count
5422	2012-01-01 00:00:00	1	0	...	5	43	48
5423	2012-01-01 01:00:00	1	0	...	15	78	93
5424	2012-01-01 02:00:00	1	0	...	16	59	75
5425	2012-01-01 03:00:00	1	0	...	11	41	52
5426	2012-01-01 04:00:00	1	0	...	0	8	8

[5 rows x 12 columns]