



CHAPTER

07

데이터 준비 Data Preparation

컴퓨터소프트웨어공학과
김대영

- 손실 데이터 처리
- 데이터 변환
- 문자열 처리

- **누락된 데이터 처리하기**(Filtering out missing data)
 - Pandas 라이브러리의 목표는 손실된 데이터를 쉽게 처리하는 것
 - Pandas는 손실 데이터를 표현하기 위해 NaN (Not a Number) 사용
 - 손실 데이터를 분리하기 위해 dropna 메소드 사용

```
data = pd.Series([1, NA, 3.5, NA, 7])
obj = data.dropna()
print(obj)
>      0      1.0
      2      3.5
      4      7.0
      dtype: float64
```

• 누락된 데이터 처리하기

- dropna() 메소드는 기본적으로 누락된 값(NA)이 포함된 모든 행을 삭제

```
data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],  
                    [NA, NA, NA], [NA, 6.5, 3.]])  
print(data)  
>  
      0      1      2  
0  1.0  6.5  3.0  
1  1.0  NaN  NaN  
2  NaN  NaN  NaN  
3  NaN  6.5  3.0  
cleaned = data.dropna()  
print(cleaned)  
>  
      0      1      2  
0  1.0  6.5  3.0
```

• 누락된 데이터 처리하기

- how='all' 옵션을 사용하면 로우의 모든 값이 NA인 것만 제외

```
obj = data.dropna(how='all')
```

```
print(obj)
```

```
>
      0      1      2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
3  NaN  6.5  3.0
```

- 같은 방법으로 컬럼 데이터를 제외하기 위해 axis=1 을 사용

```
data[3] = NA
```

```
print(data)
```

```
>
      0      1      2      3
0  1.0  6.5  3.0  NaN
1  1.0  NaN  NaN  NaN
2  NaN  NaN  NaN  NaN
3  NaN  6.5  3.0  NaN
```

```
obj = data.dropna(axis=1, how='all')
```

```
print(obj)
```

```
>
      0      1      2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0
```

• 누락된 데이터 처리하기

- 특정 수의 관측값을 포함하는 로우를 유지하기 위해 thresh 인수 사용

```
df = pd.DataFrame(np.random.randn(7,3))
df.iloc[:4,1] = NA
df.iloc[:2,2] = NA
print(df)
```

```
>      0      1      2
0  2.131860   NaN   NaN
1  2.733144   NaN   NaN
2  0.947790   NaN -0.831583
3 -0.566871   NaN -1.066718
4 -0.864277  1.040048  0.348852
5 -2.805533 -0.233832  0.758819
6 -1.673680 -0.197935  0.866938
```

```
obj = df.dropna()
print(obj)
```

```
>      0      1      2
4 -0.864277  1.040048  0.348852
5 -2.805533 -0.233832  0.758819
6 -1.673680 -0.197935  0.866938
```

```
obj = df.dropna(thresh=2)
print(obj)
```

```
>      0      1      2
2  0.947790   NaN -0.831583
3 -0.566871   NaN -1.066718
4 -0.864277  1.040048  0.348852
5 -2.805533 -0.233832  0.758819
6 -1.673680 -0.197935  0.866938
```

• 누락된 데이터 처리하기

- 누락된 데이터를 필터링하는 대신 여러 가지 방법으로 채울 수 있음
- fillna() 메소드를 사용하여 주어진 값으로 누락된 데이터를 대체함

```
df = pd.DataFrame(np.random.randn(6,3))
df.iloc[2:,1] = NA
df.iloc[4:,2] = NA
print(df)
```

>	0	1	2
0	0.694692	0.304833	0.518312
1	1.050243	0.208789	-1.198255
2	-0.428801	NaN	0.096578
3	0.343573	NaN	-1.019016
4	1.524255	NaN	NaN
5	-1.441387	NaN	NaN

```
obj = df.fillna(0)
print(obj)
```

>	0	1	2
0	0.694692	0.304833	0.518312
1	1.050243	0.208789	-1.198255
2	-0.428801	0.000000	0.096578
3	0.343573	0.000000	-1.019016
4	1.524255	0.000000	0.000000
5	-1.441387	0.000000	0.000000

• 누락된 데이터 처리하기

- 사전으로 fillna() 메소드를 호출하면 각 컬럼에 다른 채우기 값을 사용할 수 있음
- fillna() 메소드는 새 객체를 반환하지만 기존 객체 내부에서 수정 가능

```
obj = df.fillna({1:0.5, 2:0})
print(obj)
```

>	0	1	2
0	0.694692	0.304833	0.518312
1	1.050243	0.208789	-1.198255
2	-0.428801	0.500000	0.096578
3	0.343573	0.500000	-1.019016
4	1.524255	0.500000	0.000000
5	-1.441387	0.500000	0.000000

```
df.fillna(0, inplace=True)
print(df)
```

>	0	1	2
0	0.694692	0.304833	0.518312
1	1.050243	0.208789	-1.198255
2	-0.428801	0.000000	0.096578
3	0.343573	0.000000	-1.019016
4	1.524255	0.000000	0.000000
5	-1.441387	0.000000	0.000000

• 누락된 데이터 처리하기

- 재인덱싱에 사용할 수 있는 동일한 보간 방법을 fillna() 메소드와 함께 사용 가능

```
df = pd.DataFrame(np.random.randn(6,3))
df.iloc[2:,1] = NA
df.iloc[4:,2] = NA
print(df)
```

	0	1	2
0	0.229150	0.095137	1.055252
1	-0.493115	-1.823829	0.987571
2	-1.142701	NaN	0.522744
3	0.731767	NaN	-1.246407
4	-0.653457	NaN	NaN
5	-1.137879	NaN	NaN

```
obj = df.fillna(method='ffill')
print(obj)
```

	0	1	2
0	0.229150	0.095137	1.055252
1	-0.493115	-1.823829	0.987571
2	-1.142701	-1.823829	0.522744
3	0.731767	-1.823829	-1.246407
4	-0.653457	-1.823829	-1.246407
5	-1.137879	-1.823829	-1.246407

```
obj = df.fillna(method='ffill', limit=2)
print(obj)
```

• 누락된 데이터 처리하기

- fillna() 메소드 인자

value	누락된 데이터를 채우기 위한 값
method	누락된 데이터를 채우기 위한 방식, 'ffill'을 기본적으로 사용
axis	데이터에 대한 축
inplace	새로운 객체에 대한 복사본 생성 없이 기존 호출한 객체 수정
limit	누락된 데이터를 채우기 위한 최대 값의 수를 지정

• 중복 제거하기(Removing Duplicates)

- DataFrame 객체의 duplicated() 메소드를 사용하면, 중복 데이터에 대하여 Boolean Series를 리턴함

```
data = pd.DataFrame({'k1':['one','two']  
* 3 + ['two'], 'k2':[1,1,2,3,3,4,4]})
```

```
print(data)
```

```
>  k1  k2  
0  one   1  
1  two   1  
2  one   2  
3  two   3  
4  one   3  
5  two   4  
6  two   4
```

```
obj = data.duplicated()
```

```
print(obj)
```

```
>          0      False  
          1      False  
          2      False  
          3      False  
          4      False  
          5      False  
          6       True  
dtype: bool
```

• 중복 제거하기(Removing Duplicates)

- `drop_duplicates()` 메소드는 중복된 배열이 `false` 인 DataFrame 객체를 리턴함
- 중복 제거와 관련한 메소드는 기본적으로 모든 컬럼에 적용되지만, 컬럼의 부분 집합에 대한 중복을 검출할 수 있음

```
obj = data.drop_duplicates()
print(obj)
>  k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
```

```
obj = data.drop_duplicates(['k1'])
print(obj)
>  k1  k2
0  one   1
1  two   1
```

• 중복 제거하기(Removing Duplicates)

- duplicated()와 drop_duplicates()는 기본적으로 처음 발견된 데이터를 유지하지만, keep='last' 옵션을 적용하면 마지막 발견된 데이터를 유지하게 됨

```
obj = data.drop_duplicates(['k1','k2'])  
print(obj)
```

>	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	5

```
obj = data.drop_duplicates(['k1','k2'],  
                           keep='last')
```

```
print(obj)
```

>	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
6	two	4	6

• 데이터 수정하기(Replacing Values)

- fillna()메소드로 누락된 데이터를 채우지 않고 다른 데이터로 수정
- map() 메소드는 객체에 있는 값의 하위 집합을 수정하는 데 사용할 수 있으며, replace() 메소드는 더 간단하고 유연한 방법을 제공
- Pandas에서 오류 데이터를 NA값으로 변경하기 위해 replace()사용

```
data = pd.Series([1., -999., 2., -999., -1000., 3.])
obj = data.replace(-999, np.nan)
print(obj)
>      0      1.0
     1      NaN
     2      2.0
     3      NaN
     4 -1000.0
     5      3.0
     dtype: float64
```

• 데이터 수정하기(Replacing Values)

- 한 번에 여러 데이터를 변경하기 위해서, 변경 데이터의 리스트와 변경하고자 하는 데이터를 전달함

```
obj = data.replace([-999, -1000], np.nan)
print(obj)
>      0      1.0
      1      NaN
      2      2.0
      3      NaN
      4      NaN
      5      3.0
      dtype: float64
```

• 데이터 수정하기(Replacing Values)

- 각 데이터에 대해 다른 변경 데이터를 사용하려면 변경하고자 하는 데이터의 리스트를 전달함

```
obj = data.replace([-999, -1000], [np.nan, 0])
print(obj)
>      0      1.0
      1      NaN
      2      2.0
      3      NaN
      4      0.0
      5      3.0
      dtype: float64
```

• 데이터 수정하기(Replacing Values)

- 데이터를 수정하기 위해 `replace()`에 전달되는 인수는 사전이 사용될 수도 있음

```
obj = data.replace({'-999':np.nan, -1000:0})
print(obj)
> 0      1.0
   1      NaN
   2      2.0
   3      NaN
   4      0.0
   5      3.0
dtype: float64
```

• 축 이름 변경하기(Renaming Axis Indexes)

- 축 레이블은 다른 레이블 객체로 생성하기 위해 함수 또는 일부 형식의 매핑에 의해 변환할 수 있음
- 새로운 자료구조를 생성하지 않고 축을 수정할 수 있음
- Series 객체처럼 축 인덱스에는 map() 메소드 사용

```
data = pd.DataFrame(np.arange(12).reshape((3,4)),  
                    index=['Ohio', 'Colorado', 'New York'],  
                    columns=['one', 'two', 'three', 'four'])  
transform = lambda x: x[:4].upper()  
obj = data.index.map(transform)  
print(obj)  
> Index(['OHIO', 'COLO', 'NEW '], dtype='object')
```

• 축 이름 변경하기(Renaming Axis Indexes)

- DataFrame 객체는 제자리에서 인덱스 수정이 가능

```
data.index = data.index.map(transform)
print(data)
>      one  two  three  four
OHIO    0    1     2     3
COLO    4    5     6     7
NEW     8    9    10    11
```

- 원본 데이터를 수정하지 않고 데이터 셋의 변형된 버전을 생성하는 경우 사용할 수 있는 메소드는 rename()

```
obj = data.rename(index=str.title, columns=str.upper)
print(obj)
>      ONE  TWO  THREE  FOUR
Ohio    0    1     2     3
Colo    4    5     6     7
New     8    9    10    11
```

• 축 이름 변경하기(Renaming Axis Indexes)

- rename() 메소드는 축 레이블의 하위 집합에 대해 새 값을 제공하는 사전과 같은 객체와 함께 사용할 수 있음
- rename()을 사용하면 DataFrame을 수동으로 복사하고 인덱스 및 열 속성에 할당하는 번거로운 작업을 수행할 필요가 없음

```
obj = data.rename(index={'OHIO':'INDIANA'}, columns={'three':'peekaboo'})
print(obj)
```

>	one	two	peekaboo	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

- 축 이름 변경하기(Renaming Axis Indexes)

- 데이터 셋을 제자리에서 수정하기 위해 inplace=True를 전달

```
data.rename(index={'OHIO':'INDIANA'}, inplace=True)
```

```
print(data)
```

```
>          one  two  three  four
INDIANA    0    1     2     3
COLO       4    5     6     7
NEW        8    9    10    11
```

• 이산화 및 양자화(Discretization and Binning)

- 연속 데이터는 이산화되거나 분석을 위해 양자(bin)으로 분리
- 예제) 사람들을 ages 단위로 그룹을 구성하려고 한다.
 - 18~25, 26~35, 36~60, 마지막으로 61세 이상 그룹으로 구성
 - Pandas의 cut() 함수를 사용하여 구현

```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
cats = pd.cut(ages, bins)

print(cats)
> [(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35]
, (60, 100], (35, 60], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35, 60] <
(60, 100]]
```

• 이산화 및 양자화(Discretization and Binning)

- Pandas가 반환하는 객체는 특별한 categorical 객체로써, 그 출력은 pandas.cut()에 의해 계산된 양자(bin)을 설명함
- 내부적으로 코드 속성의 ages 데이터에 대한 카테고리 이름을 categories라는 배열에 포함하고 있음

```
print(cats.codes)
> [0 0 0 1 0 0 2 1 3 2 2 1]

print(cats.categories)
> IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]],
                closed='right',
                dtype='interval[int64]')
```

• 이산화 및 양자화(Discretization and Binning)

- `pd.value_counts(cats)`는 `pandas.cut()`의 결과에 대한 bin(양자) 카운트
- 괄호는 측면이 열려 있음을 의미하고 대괄호는 측면이 닫혀 있음을 의미(포함)함
 - `right=False`를 전달하여 닫히는 쪽을 변경

```
obj = pd.value_counts(cats)
print(obj)
>      (18, 25]      5
      (25, 35]      3
      (35, 60]      3
      (60, 100]     1
      dtype: int64
```

• 이산화 및 양자화(Discretization and Binning)

- 레이블 옵션에 목록 또는 배열을 전달하여 고유한 양자(bin) 이름을 전달할 수 있음

```
group_names= ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']  
obj = pd.cut(ages, bins, labels=group_names)  
  
print(obj)  
> ['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', ..., 'YoungAdult',  
  'Senior', 'MiddleAged', 'MiddleAged', 'YoungAdult']  
Length: 12  
Categories (4, object): ['Youth' < 'YoungAdult' < 'MiddleAged' < 'Senior']
```

• 이산화 및 양자화(Discretization and Binning)

- 명시적인 bin 경계 대신 잘라낼 정수 개수의 bin을 전달하면 데이터의 최소값과 최대값을 기반으로 동일한 길이의 bin을 계산

```
data = np.random.rand(20)
obj = pd.cut(data, 4, precision=2)

print(obj)
> [(0.03, 0.25], (0.47, 0.69], (0.69, 0.9], (0.69, 0.9], (0.47, 0.69]
, ..., (0.03, 0.25], (0.47, 0.69], (0.47, 0.69], (0.47, 0.69], (0.25,
0.47]]
Length: 20
Categories (4, interval[float64]): [(0.03, 0.25] < (0.25, 0.47] < (0.
47, 0.69] < (0.69, 0.9]]
```

• 이산화 및 양자화(Discretization and Binning)

- 밀접하게 관련된 함수인 `qcut()`은 샘플 수를 기반으로 데이터를 나눔
 - 데이터 분포에 따라 `cut()`을 사용하면 일반적으로 각 bin이 동일한 수의 데이터 포인트를 가지지 않음
 - `qcut()`은 대신 샘플 분위수를 사용하므로 정의에 따라 동일한 크기의 bin으로 나눔

```
data = np.random.randn(1000)
cats = pd.qcut(data, 4)
print(cats)
> (-0.715, 0.0189], (-2.87, -0.715], (-0.715, 0.0189], (-2.87, -0.715], (-2.87, -0.715],
..., (0.652, 2.825], (0.0189, 0.652], (0.0189, 0.652], (0.652, 2.825], (0.652, 2.825]]
Length: 1000
Categories (4, interval[float64]): [(-2.87, -0.715] < (-0.715, 0.0189] < (0.0189, 0.652] < (0.652, 2.825]]
```

```
obj = pd.value_counts(cats)
print(obj)
> (-3.199, -0.68]      250
   (-0.68, -0.00757]   250
   (-0.00757, 0.697]   250
   (0.697, 2.834]      250
dtype: int64
```