

파이썬 프로그래밍 기초

8기 교육부 김지나

Table of contents

01

판다스 입문

02

데이터프레임 다루기

03

데이터프레임 합치기

04

결측치 처리하기

01

판다스 입문

01 판다스 입문

-series

"인덱스와 짝지어진 데이터로 이루어진 1차원 배열"

| The Series | | |
|------------|--------------|----------|
| | Animals | ← Name |
| 0 | Dog | |
| 1 | Bear | |
| 2 | Tiger | |
| 3 | Moose | ← Values |
| 4 | Giraffe | |
| 5 | Hippopotamus | |
| 6 | Mouse | |

01 판다스 입문

-series

"딕셔너리의 key = 시리즈의 index, 딕셔너리의 값 = 시리즈의 데이터 값"

#Dictionary 형식 데이터를 1차원 자료로 환원

```
population1={'서울':9700000, '부산':4500000, '인천':4000000, '광주':2000000, '대구':2500000}
```

```
popu=pd.Series(population1)
```

```
print(popu)      #dictionary 형식 데이터에서 key가 Series의 index로 변환
```

```
서울    9700000
부산    4500000
인천    4000000
광주    2000000
대구    2500000
dtype: int64
```

-value_counts()

-unique() nunique()

-methods chaining

01 판다스 입문

-series

"value_counts()"

```
popu.value_counts()
```

```
2500000    1
4500000    1
2000000    1
4000000    1
9700000    1
dtype: int64
```

-각 범주별 데이터 값 개수 count

-빈도가 높은 데이터 값부터 출력

-역순 정렬 가능(reversed=True)

01 판다스 입문

-series

"unique(), nunique()"

```
popu.unique()
```

```
array([9700000, 4500000, 4000000, 2000000, 2500000], dtype=int64)
```

```
popu.nunique()
```

```
5
```

-시리즈 내 unique한 값 출력

-시리즈 내 unique한 값의 개수 출력

01 판다스 입문

-series

"Method chaining"

df2 데이터에서 각 class 별 생존율을 구하시오.

```
df2.groupby('class')['survived'].agg(lambda x: x.sum()/len(x))
```

```
class
First    0.562044
Second   0.416667
Third    0.236467
Name: survived, dtype: float64
```

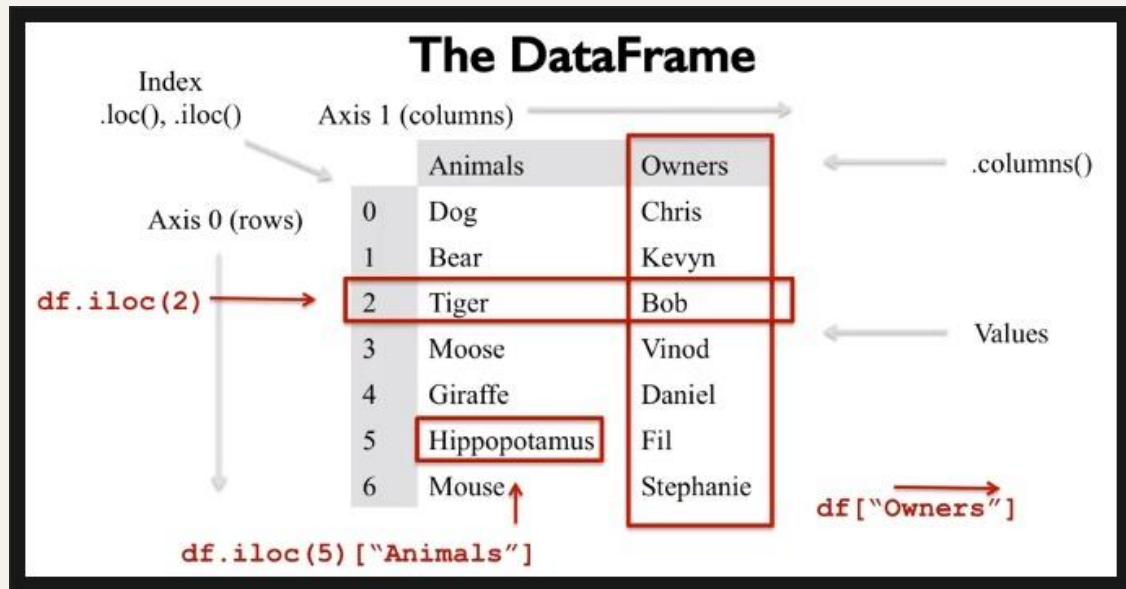
-method의 반환 값을 통해 또 다른 함수 호출

-간략하게 표현 가능, 길어지면 가독성 감소

01 판다스 입문

-Dataframe

"행과 열로 이루어진 2차원 형태의 데이터 배열"



01 판다스 입문

-Dataframe

"시리즈를 데이터프레임으로 변환할 때 열 이름 부여해 변환"

```
#앞의 예제에서 population에 열이름에 'pop'를 붙임.  
pd.DataFrame(popu, columns=['pop'])
```

```
# 1차원 array--> 2차원 행렬자료  
#열 벡터의 열 이름을 부여할 수 있음.
```

| | pop |
|----|---------|
| 서울 | 9700000 |
| 부산 | 4500000 |
| 인천 | 4000000 |
| 광주 | 2000000 |
| 대구 | 2500000 |

- index column values
- shape
- dtypes
- len
- head(), tail()
- describe()
- info()

01 판다스 입문

-Dataframe

"이후 진행을 위한 데이터셋 불러오기"

```
import seaborn as sns # seaborn을 불러오고 sns로 축약함.  
iris=sns.load_dataset('iris')  
iris
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|-----|--------------|-------------|--------------|-------------|-----------|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| ... | ... | ... | ... | ... | ... |
| 145 | 6.7 | 3.0 | 5.2 | 2.3 | virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | virginica |

150 rows × 5 columns

01 판다스 입문

-Dataframe

"데이터프레임 method 활용"

-DataFrame.method()

-DataFrame['변수'].method()

-Dataframe[['변수','변수',...]].method()

-> 다수의 열을 지정할 때 [[]] 사용!

01 판다스 입문

-Dataframe

"index, column, values"

iris.index

```
RangeIndex(start=0, stop=150, step=1)
```

iris.columns

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',  
      'species'],  
      dtype='object')
```

iris.values

```
array([[5.1, 3.5, 1.4, 0.2, 'setosa'],  
       [4.9, 3.0, 1.4, 0.2, 'setosa'],  
       [4.7, 3.2, 1.3, 0.2, 'setosa'],  
       [4.6, 3.1, 1.5, 0.2, 'setosa'],  
       [5.0, 3.6, 1.4, 0.2, 'setosa'],  
       [5.4, 3.9, 1.7, 0.4, 'setosa'],  
       [4.6, 3.4, 1.4, 0.3, 'setosa'],  
       [5.0, 3.4, 1.5, 0.2, 'setosa'],  
       [4.4, 2.9, 1.4, 0.2, 'setosa'],  
       [4.9, 3.1, 1.5, 0.1, 'setosa'],  
       [5.4, 3.7, 1.5, 0.2, 'setosa'],  
       [4.8, 3.4, 1.6, 0.2, 'setosa'],  
       [4.8, 3.0, 1.4, 0.1, 'setosa'],  
       [4.3, 3.0, 1.1, 0.1, 'setosa'],  
       [5.8, 4.0, 1.2, 0.2, 'setosa'],
```

-열의 개수가 많을 때 확인하기 유용

01 판다스 입문

-Dataframe

"shape"

```
iris.shape
```

```
(150, 5)
```

```
iris.shape[0]
```

```
150
```

ex) `iris.shape[1]`

↳

-데이터프레임 병합 시 확인 필요

-indexing으로 행의 개수, 열의 개수만 확인 가능

01 판다스 입문

-Dataframe

"dtypes"

```
iris.dtypes
```

```
sepal_length    float64  
sepal_width     float64  
petal_length    float64  
petal_width     float64  
species         object  
dtype: object
```

-데이터 타입 살펴보기

-열별 데이터 타입 확인 가능

01 판다스 입문

-Dataframe

"len"

```
len(iris)
```

```
150
```

```
iris.size
```

```
750
```

-행의 길이 확인

-size : 행의 개수 x 열의 개수 = 값의 개수

$$150 \times 5 = 750$$

01 판다스 입문

-Dataframe

"head(), tail()"

```
iris.head()
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

```
iris.tail()
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|-----|--------------|-------------|--------------|-------------|-----------|
| 145 | 6.7 | 3.0 | 5.2 | 2.3 | virginica |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 | virginica |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 | virginica |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | virginica |

-head -> 상위 데이터, tail -> 하위 데이터

-default = 5

01 판다스 입문

-Dataframe

"describe()"

```
iris.describe()
```

| | sepal_length | sepal_width | petal_length | petal_width |
|-------|--------------|-------------|--------------|-------------|
| count | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| mean | 5.843333 | 3.057333 | 3.758000 | 1.199333 |
| std | 0.828066 | 0.435866 | 1.765298 | 0.762238 |
| min | 4.300000 | 2.000000 | 1.000000 | 0.100000 |
| 25% | 5.100000 | 2.800000 | 1.600000 | 0.300000 |
| 50% | 5.800000 | 3.000000 | 4.350000 | 1.300000 |
| 75% | 6.400000 | 3.300000 | 5.100000 | 1.800000 |
| max | 7.900000 | 4.400000 | 6.900000 | 2.500000 |

-실수 타입 열 데이터에 대한 통계량

-결측치는 제외하고 계산한 값

01 판다스 입문

-Dataframe

"info()"

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 150 entries, 0 to 149
```

```
Data columns (total 5 columns):
```

| # | Column | Non-Null Count | Dtype |
|---|--------------|----------------|---------|
| 0 | sepal_length | 150 non-null | float64 |
| 1 | sepal_width | 150 non-null | float64 |
| 2 | petal_length | 150 non-null | float64 |
| 3 | petal_width | 150 non-null | float64 |
| 4 | species | 150 non-null | object |

```
dtypes: float64(4), object(1)
```

```
memory usage: 6.0+ KB
```

-데이터프레임의 전체적인 구조 확인

-열별 결측치 개수, 데이터타입 메모리 사용량 등

01 판다스 입문

-Dataframe

"Summerizing 데이터프레임"

```
iris.min()
```

```
sepal_length    4.3  
sepal_width      2  
petal_length     1  
petal_width     0.1  
species         setosa  
dtype: object
```

-시리즈로 출력

```
iris.quantile(0.25)
```

```
sepal_length    5.1  
sepal_width     2.8  
petal_length    1.6  
petal_width     0.3  
Name: 0.25, dtype: float64
```

-원하는 quantile 값 지정하기

02

데이터프레임
다루기

02 데이터프레임 다루기

-Indexing & Slicing

"loc, iloc" 차이 알아보기☆
~~~~~

```
## data Slicing1: loc
print(iris.loc[0:1,:]) #0행~1행까지의 자료를 Slicing : [a:b]는 a에서 b자리수 까지를 의미
```

|   | sepal_length | sepal_width | petal_length | petal_width | species |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 1 | 4.9          | 3.0         | 1.4          | 0.2         | setosa  |

```
## data Slicing1: iloc
print(iris.iloc[0:1,:]) #0행 자료를 Slicing : [a:b]는 a에서 b-1 자리수 까지를 의미
```

```
# loc vs iloc
```

|   | sepal_length | sepal_width | petal_length | petal_width | species |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |

-loc -> 라벨을 통한 인덱싱  
Ex) iris.loc[:, "sepal\_width"]

-iloc -> 인덱스를 통한 인덱싱  
Ex) iris.iloc[:, 1]

## 02 데이터프레임 다루기

### -Indexing & Slicing

"loc, iloc 예시"

```
iris.loc[:, 'sepal_width']
```

```
iris.iloc[:, 1]
```

```
0    3.5  
1    3.0  
2    3.2  
3    3.1  
4    3.6
```

```
...
```

```
145   3.0  
146   2.5  
147   3.0  
148   3.4  
149   3.0
```

```
Name: sepal_width, Length: 150, dtype: float64
```

```
iris.loc[0:2, ['sepal_width', 'sepal_length']]
```

|   | sepal_width | sepal_length |
|---|-------------|--------------|
| 0 | 3.5         | 5.1          |
| 1 | 3.0         | 4.9          |
| 2 | 3.2         | 4.7          |

여러 개의 행/열을 연속적으로 추출  
→ iloc가 더 유용

열의 개수/종류가 많을 때 필요한 열만 추출  
→ loc가 더 유용

## 02 데이터프레임 다루기

### -Indexing & Slicing

"Boolean indexing"

```
iris['sepal_width']>4
```

```
0    False
1    False
2    False
3    False
4    False
...
145   False
146   False
147   False
148   False
149   False
Name: sepal_width, Length: 150, dtype: bool
```

```
iris[iris['sepal_width']>4]
```

|    | sepal_length | sepal_width | petal_length | petal_width | species |
|----|--------------|-------------|--------------|-------------|---------|
| 15 | 5.7          | 4.4         | 1.5          | 0.4         | setosa  |
| 32 | 5.2          | 4.1         | 1.5          | 0.1         | setosa  |
| 33 | 5.5          | 4.2         | 1.4          | 0.2         | setosa  |

-[] 연산자 내에 불린 조건을 입력  
-> 조건에 해당하는 행 출력

-비교 연산자(==, >, >=, != 등)

-in 연산자 (in, ==, not in, !=)

-논리 연산자(and, or, not)



## 02 데이터프레임 다루기

### -Indexing & Slicing

"Boolean indexing(isin)"

```
iris[iris['sepal_length'].isin([5.0, 5.1])].head()
```

|    | sepal_length | sepal_width | petal_length | petal_width | species |
|----|--------------|-------------|--------------|-------------|---------|
| 0  | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 4  | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| 7  | 5.0          | 3.4         | 1.5          | 0.2         | setosa  |
| 17 | 5.1          | 3.5         | 1.4          | 0.3         | setosa  |
| 19 | 5.1          | 3.8         | 1.5          | 0.3         | setosa  |

-특정 값을 지닌 행들을 추출

## 02 데이터프레임 다루기

### -Indexing & Slicing

"Boolean indexing 예시"

```
iris[iris['species']=='setosa'][['sepal_width', 'sepal_length']].head()
```

|   | sepal_width | sepal_length |
|---|-------------|--------------|
| 0 | 3.5         | 5.1          |
| 1 | 3.0         | 4.9          |
| 2 | 3.2         | 4.7          |
| 3 | 3.1         | 4.6          |
| 4 | 3.6         | 5.0          |

```
iris[(iris['sepal_width']>3) & (iris['species']=='setosa')].head()
```

|   | sepal_length | sepal_width | petal_length | petal_width | species |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 2 | 4.7          | 3.2         | 1.3          | 0.2         | setosa  |
| 3 | 4.6          | 3.1         | 1.5          | 0.2         | setosa  |
| 4 | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| 5 | 5.4          | 3.9         | 1.7          | 0.4         | setosa  |

## 02 데이터프레임 다루기

### -Indexing & Slicing

"Query 함수"

```
iris[iris['sepal_length'].isin([5.0, 5.1])].head()
```

|    | sepal_length | sepal_width | petal_length | petal_width | species |
|----|--------------|-------------|--------------|-------------|---------|
| 0  | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 4  | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| 7  | 5.0          | 3.4         | 1.5          | 0.2         | setosa  |
| 17 | 5.1          | 3.5         | 1.4          | 0.3         | setosa  |
| 19 | 5.1          | 3.8         | 1.5          | 0.3         | setosa  |

-DataFrame.query('조건문')

-Boolean indexing에서 사용했던 연산자  
사용가능

-외부 변수명 또는 함수명 사용 시  
앞에 @를 붙여 사용

## 02 데이터프레임 다루기

### -Indexing & Slicing

"Query 함수 예시"

```
iris.query('sepal_length>5').head(3)
```

|    | sepal_length | sepal_width | petal_length | petal_width | species |
|----|--------------|-------------|--------------|-------------|---------|
| 0  | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 5  | 5.4          | 3.9         | 1.7          | 0.4         | setosa  |
| 10 | 5.4          | 3.7         | 1.5          | 0.2         | setosa  |

```
length = 5
```

```
iris.query('sepal_length==@length').head(3)
```

|    | sepal_length | sepal_width | petal_length | petal_width | species |
|----|--------------|-------------|--------------|-------------|---------|
| 4  | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| 7  | 5.0          | 3.4         | 1.5          | 0.2         | setosa  |
| 25 | 5.0          | 3.0         | 1.6          | 0.2         | setosa  |

```
iris.query('species == "setosa" and sepal_length in [5.0, 5.1]').head(3)
```

|   | sepal_length | sepal_width | petal_length | petal_width | species |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 4 | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| 7 | 5.0          | 3.4         | 1.5          | 0.2         | setosa  |

```
str_expr = "sepal_width == 3.5"
```

```
iris.query(str_expr).head(3)
```

|    | sepal_length | sepal_width | petal_length | petal_width | species |
|----|--------------|-------------|--------------|-------------|---------|
| 0  | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 17 | 5.1          | 3.5         | 1.4          | 0.3         | setosa  |
| 27 | 5.2          | 3.5         | 1.5          | 0.2         | setosa  |

## 02 데이터프레임 다루기

### -sorting

"sort\_values(by='기준 열')"

```
iris.sort_values(by='sepal_width')
```

|     | sepal_length | sepal_width | petal_length | petal_width | species    |
|-----|--------------|-------------|--------------|-------------|------------|
| 60  | 5.0          | 2.0         | 3.5          | 1.0         | versicolor |
| 62  | 6.0          | 2.2         | 4.0          | 1.0         | versicolor |
| 119 | 6.0          | 2.2         | 5.0          | 1.5         | virginica  |
| 68  | 6.2          | 2.2         | 4.5          | 1.5         | versicolor |
| 41  | 4.5          | 2.3         | 1.3          | 0.3         | setosa     |
| ... | ...          | ...         | ...          | ...         | ...        |
| 16  | 5.4          | 3.9         | 1.3          | 0.4         | setosa     |
| 14  | 5.8          | 4.0         | 1.2          | 0.2         | setosa     |
| 32  | 5.2          | 4.1         | 1.5          | 0.1         | setosa     |
| 33  | 5.5          | 4.2         | 1.4          | 0.2         | setosa     |
| 15  | 5.7          | 4.4         | 1.5          | 0.4         | setosa     |

150 rows × 5 columns

-기준 열을 기준으로 데이터프레임 정렬

-여러 개의 기준열을 사용할 경우  
앞서 있는 열 기준으로 먼저 정렬

-default = ascending

## 02 데이터프레임 다루기

### -function mapping

"apply와 lambda를 통한 매핑 "

#### 문제 [2-3] (10점)

iris의 species 열 값을 setosa는 1, 그 외의 값들은 0으로 변환하시오. (apply, lambda 사용)

```
iris['species']=iris['species'].apply(lambda x : 1 if x == 'setosa' else 0)
```

|     | sepal_length | sepal_width | petal_length | petal_width | species |
|-----|--------------|-------------|--------------|-------------|---------|
| 0   | 5.1          | 3.5         | 1.4          | 0.2         | 1       |
| 1   | 4.9          | 3.0         | 1.4          | 0.2         | 1       |
| 2   | 4.7          | 3.2         | 1.3          | 0.2         | 1       |
| 3   | 4.6          | 3.1         | 1.5          | 0.2         | 1       |
| 4   | 5.0          | 3.6         | 1.4          | 0.2         | 1       |
| ... | ...          | ...         | ...          | ...         | ...     |

-값들을 원하는 방식으로 변경

-function, lambda 모두 사용 가능

## 02 데이터프레임 다루기

### -group aggregation

"groupby, agg를 통한 데이터 집계 "

#### 문제 [3-5] (20점)

df2 데이터에서 각 class 별 생존율을 구하시오.

```
df2.groupby('class')['survived'].agg(lambda x: x.sum()/len(x))
```

```
class
First    0.562044
Second   0.416667
Third    0.236467
Name: survived, dtype: float64
```

-groupby -> key별 (class) 데이터 집계

-agg(내가 사용할 함수)

-집계된 데이터를 사용자 지정함수를 통해 분석

## 02 데이터프레임 다루기

### -group aggregation

"groupby를 통한 데이터 집계 "

↗ 데이터 프레임으로 결과 출력 가능

```
iris.groupby('species')['sepal_width'].sum().reset_index()
```

|   | species    | sepal_width |
|---|------------|-------------|
| 0 | setosa     | 171.4       |
| 1 | versicolor | 138.5       |
| 2 | virginica  | 148.7       |

-집계 결과를 reset\_index()를 통해  
새 데이터프레임으로 만들기



03

# 데이터프레임 합치기

## 03 데이터프레임 합치기

### -데이터프레임 생성

```
Q1=pd.DataFrame([[1,2,3],[4,5,6]],columns=list('abc'))
Q2=pd.DataFrame([[11,22,33],[44,55,66]],columns=list('bcd'))
print(Q1)
print('-----')
print(Q2)
```

|   | a | b | c |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |

|   | b  | c  | d  |
|---|----|----|----|
| 0 | 11 | 22 | 33 |
| 1 | 44 | 55 | 66 |

-concat(연결)

-merge(병합)

-join(결합)

## 03 데이터프레임 합치기

### -concat(연결)

```
print(pd.concat([Q1,Q2]))           #밀어서 붙이기
print('-----')
print(pd.concat([Q1,Q2], ignore_index=True))  #순서있는 인덱스를 원할 경우
print('-----')
print(pd.concat([Q1,Q2], join='inner'))      #공통으로 가지고 있는 열 'b' 와 'c' 로
```

|   | a   | b  | c  | d    |
|---|-----|----|----|------|
| 0 | 1.0 | 2  | 3  | NaN  |
| 1 | 4.0 | 5  | 6  | NaN  |
| 0 | NaN | 11 | 22 | 33.0 |
| 1 | NaN | 44 | 55 | 66.0 |

|   | a   | b  | c  | d    |
|---|-----|----|----|------|
| 0 | 1.0 | 2  | 3  | NaN  |
| 1 | 4.0 | 5  | 6  | NaN  |
| 2 | NaN | 11 | 22 | 33.0 |
| 3 | NaN | 44 | 55 | 66.0 |

|   | b  | c  |
|---|----|----|
| 0 | 2  | 3  |
| 1 | 5  | 6  |
| 0 | 11 | 22 |
| 1 | 44 | 55 |

-데이터프레임을 위아래로 연결  
-> axis=1을 사용해 옆으로 붙이기

-행 인덱스는 본래 형태 유지  
-> ignore\_index=True를 사용해 변경

-join='inner' 열 인덱스 교집합 기준

## 03 데이터프레임 합치기

### -merge(병합)

*#공통된 열이 있을때의 옆으로 붙이기.*

```
S1=pd.DataFrame([[1, 'aa', 3], [4, 'bb', 5]], columns=list('abc')) #새로운 data의 생성
S2=pd.DataFrame([[11, 'bb'], [22, 'aa']], columns=list('db'))
print(S1)
print(S2)
print('-----')
S3=pd.merge(S1, S2, on='b') #'b'열을 기준으로 옆으로 붙이기.
print(S3)
```

|   | a | b  | c |
|---|---|----|---|
| 0 | 1 | aa | 3 |
| 1 | 4 | bb | 5 |

|   | d  | b  |
|---|----|----|
| 0 | 11 | bb |
| 1 | 22 | aa |

-----

|   | a | b  | c | d  |
|---|---|----|---|----|
| 0 | 1 | aa | 3 | 22 |
| 1 | 4 | bb | 5 | 11 |

-default : on=None  
-> 공통된 모든 열을 기준으로

-공통된 열이 있을 때 on='열' 을 기준으로 병합

## 03 데이터프레임 합치기

### -merge(병합)

#공통된 열이 있을때의 옆으로 붙이기.: inner vs outer vs left

```
S3=pd.merge(S1,S2,how='inner') #공통 열 'b' 를 기준으로 교집합
```

```
print(S3)
```

```
print('-----')
```

```
S4=pd.merge(S1,S2,how='outer') #공통 열 'b' 를 기준으로 합집합
```

```
print(S4)
```

```
print('-----')
```

```
S5=pd.merge(S1,S2,how='left') #V1을 기준으로 'b' 열에 있는 ID가 있는 V2의 행만 옆 붙이기
```

```
print(S5)
```

|   | a | b  | c | d  |
|---|---|----|---|----|
| 0 | 1 | aa | 3 | 22 |

|   | a   | b  | c   | d    |
|---|-----|----|-----|------|
| 0 | 1.0 | aa | 3.0 | 22.0 |
| 1 | 4.0 | bb | 5.0 | NaN  |
| 2 | NaN | cc | NaN | 11.0 |

|   | a | b  | c | d    |
|---|---|----|---|------|
| 0 | 1 | aa | 3 | 22.0 |
| 1 | 4 | bb | 5 | NaN  |

-how='inner'

-> 기준이 되는 열의 데이터가 공통으로 존재하는  
교집합인 경우에만 추출

-how='outer'

-> 기준이 되는 열의 데이터가 한쪽에만 속하더라도  
포함

-how='left'

-> 왼쪽 데이터프레임 키열에 속하는 데이터 값을  
기준으로 병합

## 03 데이터프레임 합치기

### -join(연결)

```
df1 = pd.DataFrame({'a': ['a0', 'a1', 'a2'],  
                    'b': ['b0', 'b1', 'b2'],  
                    'c': ['c0', 'c1', 'c2']},  
                    index=[0,1,2])  
df2 = pd.DataFrame({'e': ['e1', 'e2', 'e3'],  
                    'f': ['f1', 'f2', 'f3'],  
                    'g': ['g1', 'g2', 'g3']},  
                    index=[1,2,3])  
df3 = df1.join(df2)  
df3
```

|   | a   | b   | c   | e   | f   | g   |
|---|-----|-----|-----|-----|-----|-----|
| 0 | a0  | b0  | c0  | NaN | NaN | NaN |
| 1 | a1  | b1  | c1  | e1  | f1  | g1  |
| 2 | a2  | b2  | c2  | e2  | f2  | g2  |
| 3 | NaN | NaN | NaN | e3  | f3  | g3  |

-merge와 비슷

-행 인덱스를 기준으로 결합한다는 점

04

결측치  
처리하기

## 04 결측치 처리하기

### -결측치 확인

```
titanic.head()
```

|   | survived | pclass | sex    | age  | sibsp | parch | fare    | embarked | class | who   | adult_male | deck | embark_town | alive       | alone |      |
|---|----------|--------|--------|------|-------|-------|---------|----------|-------|-------|------------|------|-------------|-------------|-------|------|
| 0 | 0        | 3      | male   | 22.0 | 1     | 0     | 7.2500  | S        | Third | man   | True       | NaN  | Southampton | no          | False |      |
| 1 | 1        | 1      | female | 38.0 | 1     | 0     | 71.2833 | C        | First | woman | False      | C    | Cherbourg   | yes         | False |      |
| 2 | 1        | 3      | female | 26.0 | 0     | 0     | 7.9250  | S        | Third | woman | False      | ✔    | NaN         | Southampton | yes   | True |
| 3 | 1        | 1      | female | 35.0 | 1     | 0     | 53.1000 | S        | First | woman | False      | C    | Southampton | yes         | False |      |
| 4 | 0        | 3      | male   | 35.0 | 0     | 0     | 8.0500  | S        | Third | man   | True       | ✔    | NaN         | Southampton | no    | True |



## 04 결측치 처리하기

### -결측치 확인

"isnull().sum() "

df에서 열 별 결측치 여부를 확인하고,

```
df.isnull().sum()
```

```
pclass      0  
age         137  
sibsp       0  
fare        0  
class       0  
dtype: int64
```

-isnull() <-> notnull()

-info()에서도 확인 가능

## 04 결측치 처리하기

### -결측치 처리

"dropna()"

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

```
(471, 5)
```

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

```
(608, 5)
```

-결측치가 있는 행을 제거

-how=""를 통해 옵션 주기

-reset\_index()를 통한 인덱스 정렬 필요

## 04 결측치 처리하기

### -결측치 처리

"fillna()"

```
df.fillna(df.mean(), inplace=True)  
df
```

|     | pclass | age       | sibsp | fare    | class  |
|-----|--------|-----------|-------|---------|--------|
| 0   | 3      | 26.000000 | 0     | 7.9250  | Third  |
| 1   | 3      | 35.000000 | 0     | 8.0500  | Third  |
| 2   | 3      | 31.397558 | 0     | 8.4583  | Third  |
| 3   | 1      | 54.000000 | 0     | 51.8625 | First  |
| 4   | 3      | 27.000000 | 0     | 11.1333 | Third  |
| ... | ...    | ...       | ...   | ...     | ...    |
| 603 | 3      | 39.000000 | 0     | 29.1250 | Third  |
| 604 | 2      | 27.000000 | 0     | 13.0000 | Second |
| 605 | 1      | 19.000000 | 0     | 30.0000 | First  |
| 606 | 1      | 26.000000 | 0     | 30.0000 | First  |
| 607 | 3      | 32.000000 | 0     | 7.7500  | Third  |

608 rows × 5 columns

-결측치를 다른 값으로 대체

-0, 최빈값, 평균값, 중앙값 등으로 대체

## 04 결측치 처리하기

### -결측치 처리

"fillna() 예시"

```
df.fillna(method='bfill')
```

|     | pclass | age  | sibsp | fare    | class  |
|-----|--------|------|-------|---------|--------|
| 0   | 3      | 26.0 | 0     | 7.9250  | Third  |
| 1   | 3      | 35.0 | 0     | 8.0500  | Third  |
| 2   | 3      | 54.0 | 0     | 8.4583  | Third  |
| 3   | 1      | 54.0 | 0     | 51.8625 | First  |
| 4   | 3      | 27.0 | 0     | 11.1333 | Third  |
| ... | ...    | ...  | ...   | ...     | ...    |
| 603 | 3      | 39.0 | 0     | 29.1250 | Third  |
| 604 | 2      | 27.0 | 0     | 13.0000 | Second |
| 605 | 1      | 19.0 | 0     | 30.0000 | First  |
| 606 | 1      | 26.0 | 0     | 30.0000 | First  |
| 607 | 3      | 32.0 | 0     | 7.7500  | Third  |

608 rows × 5 columns

-이후 값으로 대체

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

|     | pclass | age  | sibsp | fare    | class  |
|-----|--------|------|-------|---------|--------|
| 0   | 3      | 26.0 | 0     | 7.9250  | Third  |
| 1   | 3      | 35.0 | 0     | 8.0500  | Third  |
| 2   | 3      | 35.0 | 0     | 8.4583  | Third  |
| 3   | 1      | 54.0 | 0     | 51.8625 | First  |
| 4   | 3      | 27.0 | 0     | 11.1333 | Third  |
| ... | ...    | ...  | ...   | ...     | ...    |
| 603 | 3      | 39.0 | 0     | 29.1250 | Third  |
| 604 | 2      | 27.0 | 0     | 13.0000 | Second |
| 605 | 1      | 19.0 | 0     | 30.0000 | First  |
| 606 | 1      | 26.0 | 0     | 30.0000 | First  |
| 607 | 3      | 32.0 | 0     | 7.7500  | Third  |

608 rows × 5 columns

-이전 값으로 대체


```
df.fillna({'age': 0, 'fare': 1})
```

|     | pclass | age  | sibsp | fare    | class  |
|-----|--------|------|-------|---------|--------|
| 0   | 3      | 26.0 | 0     | 7.9250  | Third  |
| 1   | 3      | 35.0 | 0     | 8.0500  | Third  |
| 2   | 3      | 0.0  | 0     | 8.4583  | Third  |
| 3   | 1      | 54.0 | 0     | 51.8625 | First  |
| 4   | 3      | 27.0 | 0     | 11.1333 | Third  |
| ... | ...    | ...  | ...   | ...     | ...    |
| 603 | 3      | 39.0 | 0     | 29.1250 | Third  |
| 604 | 2      | 27.0 | 0     | 13.0000 | Second |
| 605 | 1      | 19.0 | 0     | 30.0000 | First  |
| 606 | 1      | 26.0 | 0     | 30.0000 | First  |
| 607 | 3      | 32.0 | 0     | 7.7500  | Third  |

608 rows × 5 columns

-열 별 다른 값으로 대체

결측값  
대체  
확인하기



Q & A

감사합니다!