

### 03. 데이터 정제 - 결측값(NaN)처리

```
df.dropna(how='all')      # 모든게 결측값인거 dropna시킴
df.dropna(how='any')     # 어느거 하나라도 결측값인거 dropna시킴
df.fillna(value=5.0)     # 결측값 대체 # 값이 하나일땐 value를 안써도됨
df.isnull()              # isnull() : 결측값입니까? True or False <-> notnull()
df.loc[df.isnull()['E'], :] # 'E'열에서 결측값을 가지고 있는 로우만 추출
```

### 04. 계층적 인덱싱 (다중인덱싱)

```
df.dropna(how='all')      # 모든게 결측값인거 dropna시킴 cf. how='any'
```

```
# set_index() : 리스트 형식으로, 계층순서대로 컬럼명을 주면 컬럼에 따라 인덱스가 바뀜
df3 = df2.set_index(['c','d'])      # c,d열이 인덱스로 가고, 데이터는 사라짐
df2.set_index(['c','d'], drop=False) # c,d열이 인덱스로 가지만, 데이터는 남아있음
df3.reset_index()                   # 인덱스 취소
```

```
df4.index.name = 'city'      # 인덱스, 컬럼에 제목넣기
df4.columns.name = 'number'  # names-> 인덱스 여러개일 때
```

| number | one | two | three |
|--------|-----|-----|-------|
| city   |     |     |       |
| Seoul  | 0   | 1   | 2     |
| Busan  | 3   | 4   | 5     |

```
df4.stack()      # 시리즈형태로 바뀜 <-> unstack()
```

| city  | number |   |
|-------|--------|---|
| Seoul | one    | 0 |
|       | two    | 1 |
|       | three  | 2 |
| Busan | one    | 3 |
|       | two    | 4 |
|       | three  | 5 |

```
pd.merge(df1, df2, on='key', how='outer') # df1, df2 합침 (how='outer' 모든 키값 보여줌)
pd.merge(df1, df2, on='key', how='left')  # 왼쪽에 있는 행은 고정. / df1의 key값 // (오른쪽(df2)은 없으면 제거됨..)
```

|   | key | data1 |
|---|-----|-------|
| 0 | b   | 0     |
| 1 | b   | 1     |
| 2 | a   | 2     |
| 3 | c   | 3     |
| 4 | a   | 4     |
| 5 | a   | 5     |
| 6 | b   | 6     |

|   | key | data2 |
|---|-----|-------|
| 0 | a   | 0     |
| 1 | b   | 1     |
| 2 | d   | 2     |

|   | key | data1 | data2 |
|---|-----|-------|-------|
| 0 | b   | 0.0   | 1.0   |
| 1 | b   | 1.0   | 1.0   |
| 2 | b   | 6.0   | 1.0   |
| 3 | a   | 2.0   | 0.0   |
| 4 | a   | 4.0   | 0.0   |
| 5 | a   | 5.0   | 0.0   |
| 6 | c   | 3.0   | NaN   |
| 7 | d   | NaN   | 2.0   |

|   | key | data1 | data2 |
|---|-----|-------|-------|
| 0 | b   | 0     | 1.0   |
| 1 | b   | 1     | 1.0   |
| 2 | a   | 2     | 0.0   |
| 3 | c   | 3     | NaN   |
| 4 | a   | 4     | 0.0   |
| 5 | a   | 5     | 0.0   |
| 6 | b   | 6     | 1.0   |

```
# concat() : 리스트 형식으로 들어온 객체를 단순히 연결하는 함수 # 이어붙이려는 개체는 리스트형태여야함
```

```
pd.concat([s1, s2, s3])      # 단순 연결, 행으로 추가(axis=0 생략)
pd.concat([s1, s2, s3], axis=1) # 단순 연결, 열로 추가
```

|              | s1 |              | s2 |              | s3 |
|--------------|----|--------------|----|--------------|----|
| a            | 0  | c            | 2  | f            | 5  |
| b            | 1  | d            | 3  | g            | 6  |
| dtype: int64 |    | dtype: int64 |    | dtype: int64 |    |

|   |   |
|---|---|
| a | 0 |
| b | 1 |
| c | 2 |
| d | 3 |
| e | 4 |
| f | 5 |
| g | 6 |

|   | 0   | 1   | 2   |
|---|-----|-----|-----|
| a | 0.0 | NaN | NaN |
| b | 1.0 | NaN | NaN |
| c | NaN | 2.0 | NaN |
| d | NaN | 3.0 | NaN |
| e | NaN | 4.0 | NaN |
| f | NaN | NaN | 5.0 |
| g | NaN | NaN | 6.0 |

## 05. 집계함수(GroupBy)

# 집계함수의 절차 : 분리(split)-적용(apply)-결합(combine)

```
df = pd.DataFrame({'key1':list('aabbba'),
                   'key2':['one','two','one','two','one'],
                   'data1':np.random.randn(5),
                   'data2':np.random.randn(5)})
```

|   | key1 | key2 | data1     | data2    |
|---|------|------|-----------|----------|
| 0 | a    | one  | -0.301045 | 0.900435 |
| 1 | a    | two  | -1.413850 | 0.340896 |
| 2 | b    | one  | 1.305095  | 1.966725 |
| 3 | b    | two  | -1.369326 | 0.096171 |
| 4 | a    | one  | 0.500293  | 0.290949 |

```
df['data1'].groupby(df['key1']).mean()
df.groupby(['key1'])['data1'].mean()
```

# key1열의 같은 행들에 대해 data1의 평균

```
key1
a    0.990007
b   -1.062996
```

```
df.groupby(['key1', 'key2']).mean()
```

# 여러개는 묶어서 (vector)

```
for name, group in df.groupby('key1'):
    print(name); print(group)
```

# key1열로 분류 - 이름, 그룹 출력

```
a
  key1 key2  data1  data2
0    a  one  1.857145 -1.283973
1    a  two  1.169426 -0.701425
4    a  one -0.056551 -1.823926

b
  key1 key2  data1  data2
2    b  one -1.543283 -1.465086
3    b  two -0.582708  0.214430
```

# key1,key2열로 분류 - 키값, 그룹 출력

```
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print(k1, k2); print(group)
```

```
a one
  key1 key2  data1  data2
0    a  one -0.893977 -1.296604
4    a  one  1.454044  0.788926

a two
  key1 key2  data1  data2
1    a  two  1.324021  2.421498
```

```
b one
  key1 key2  data1  data2
2    b  one -0.706422  0.101698

b two
  key1 key2  data1  data2
3    b  two  1.757961 -1.305872
```

```
# 개수
df.groupby(['key1', 'key2']).count()
```

|   | key1 | key2 | data1 | data2 |
|---|------|------|-------|-------|
| a | one  | two  | 2     | 2     |
|   |      | one  | 1     | 1     |
| b | one  | two  | 1     | 1     |
|   |      | one  | 1     | 1     |

## 06. 데이터 전처리

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

불러오기

```
titanic = sns.load_dataset('titanic')
```

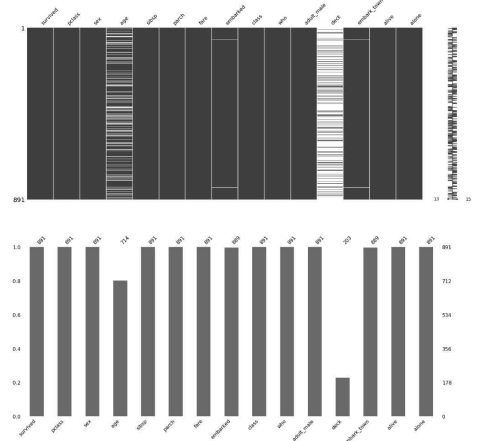
자료 정보확인

```
titanic.info()
```

missingno

```
# 아나콘다 프롬프트 관리자 - pip install missingno
import missingno as msno
msno.matrix(titanic)
plt.show()

msno.bar(titanic)
plt.show()
```

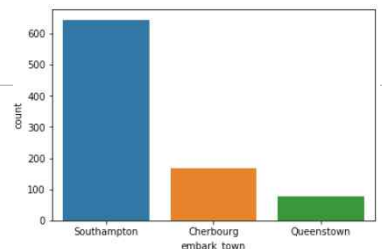


열 삭제

```
# 데이터가 절반 이상이 없는 열을 삭제
titanic = titanic.dropna(thresh=int(len(titanic)*0.5), axis=1)
```

결측값 메꾸기

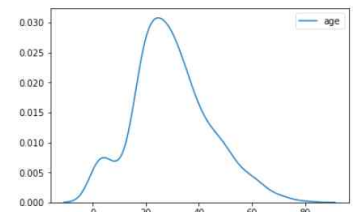
```
# 범주형 데이터는 최빈값으로 대체함 most_frequent
sns.countplot(titanic.embark_town)
plt.show()
```



```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='most_frequent') # 최빈값으로 결측값 채우기
titanic['embark_town'] = imputer.fit_transform(titanic[['embark_town']])
titanic['embarked'] = imputer.fit_transform(titanic[['embarked']]) # 2차원 배열처럼 써줘야함..
```

```
sns.kdeplot(titanic.age) # 밀도함수 -> 중앙값 확인
plt.show()
```

```
imputer_age = SimpleImputer(strategy='median') # 중앙값으로 채우기
titanic['age'] = imputer.fit_transform(titanic[['age']])
```



결측 데이터 수 확인

```
titanic.isnull().sum() # 결측데이터 수
```

피쳐(컬럼, 열) 제거

```
# 불필요한 피쳐를 제거하는 작업
df_titanic = titanic.drop(['class', 'alive', 'who', 'embarked', 'embark_town'], axis=1)
df_titanic.head()
```

## 데이터 확인

```
df_titanic.head()
```

|   | survived | pclass | sex    | age  | sibsp | parch | fare    | adult_male | alone |
|---|----------|--------|--------|------|-------|-------|---------|------------|-------|
| 0 | 0        | 3      | male   | 22.0 | 1     | 0     | 7.2500  | True       | False |
| 1 | 1        | 1      | female | 38.0 | 1     | 0     | 71.2833 | False      | False |
| 2 | 1        | 3      | female | 26.0 | 0     | 0     | 7.9250  | False      | True  |
| 3 | 1        | 1      | female | 35.0 | 1     | 0     | 53.1000 | False      | False |
| 4 | 0        | 3      | male   | 35.0 | 0     | 0     | 8.0500  | True       | True  |

## 데이터 시각화

# 등급별 고객정보, 각 개체별 건수

```
df_titanic['pclass'].value_counts()
```

(1-2-3 : 객실등급)

|   |     |
|---|-----|
| 3 | 491 |
| 1 | 216 |
| 2 | 184 |

```
df_titanic['survived'].value_counts().plot.bar()
```

(0 : 사망, 1 : 생존)

```
df_titanic['pclass'].value_counts().plot.bar()
```

```
ax = sns.countplot(x='pclass', data=df_titanic)
```

```
ax = sns.countplot(x='pclass', hue='survived', data=df_titanic)
```

```
ax = sns.countplot(x='sex', hue='survived', data=df_titanic)
```

```
ax = sns.countplot(x='alone', hue='survived', data=df_titanic)
```

