

# 01 Pandas란?

## Pandas

데이터 조작 및 분석을 위한 Python 프로그래밍 언어 용으로 작성된 소프트웨어 라이브러리



# 01 Pandas란?

## pandas import

- Colab 환경에서는 따로 설치 할 필요 없음
- Anaconda 환경에서는 내부에 존재

### pandas import

```
# Mac의 경우  
pip3 install pandas  
  
import pandas as pd
```

### Series 생성

- Series : 1차원 데이터를 저장하는 pandas의 기본 자료구조
- 하나의 Series 객체의 값은 동일한 데이터 타입의 값을 가짐

#### Series 생성

```
import numpy as np  
  
s = pd.Series([1, 3, 5, np.nan, 6, 8])  
print(s)
```

#### Result

```
0    1.0  
1    3.0  
2    5.0  
3    NaN  
4    6.0  
5    8.0  
dtype: float64
```

### Series 생성

0~9까지 랜덤한 5개의 데이터로 Series 생성

#### Series 생성

```
data = pd.Series(np.random.randint(10, size=5))  
print(data)
```

#### Result

```
0    0  
1    4  
2    0  
3    3  
4    7  
dtype: int32
```

### index 설정

#### index 설정

```
data = pd.Series(np.random.randint(10, size=5),  
                 index=["A", "B", "C", "D", "E"])  
print(data)
```

#### Result

```
A    0  
B    9  
C    1  
D    6  
E    4  
dtype: int64
```

### Series 내 value값 확인

#### value 확인 -1

```
data = pd.Series(np.random.randint(10,  
size=5), index=["A", "B", "C", "D", "E"])  
print(data)  
print(data.index)  
print(data.values)  
print(data.A)
```

#### Result

```
A 9  
B 2  
C 3  
D 5  
E 4  
dtype: int64  
Index(['A', 'B', 'C', 'D', 'E'], dtype='object')  
[9 2 3 5 4]  
9
```

### Series 내 value값 확인 (indexing)

value 확인 -2

```
data = pd.Series(np.random.randint(10, size=5)  
, index=["A", "B", "C", "D", "E"])  
print(data["C"])  
print(data[["B", "C", "E"]])  
print(data[1::2])  
print(data[::-1])
```

Result

```
7  
B      4  
C      7  
E      9  
dtype: int64  
B      4  
D      9  
dtype: int64  
E      9  
D      9  
C      7  
B      4  
A      8  
dtype: int64
```

### Series 이름과 인덱스 이름 설정

#### 이름 설정

```
data = pd.Series(np.random.randint(10, size=5)  
, index=["A", "B", "C", "D", "E"])  
data.name = "random_number"  
data.index.name = "index_number"  
print(data)
```

#### Result

index\_number

A 9

B 9

C 5

D 8

E 9

Name: random\_number,  
dtype: int64



### Series Broadcasting

#### Series Broadcasting

```
data = pd.Series(np.random.randint(10, size=5)  
, index=["A", "B", "C", "D", "E"])  
print(data * 10)
```

#### Result

A	60
B	80
C	80
D	40
E	80

dtype: int64

### ◆ 조건에 맞는 데이터 추출

#### 조건에 맞는 데이터 추출

```
data = pd.Series(np.random.randint(10, size=5),  
index=["A", "B", "C", "D", "E"])  
print(data)  
print('-'*20)  
print(data[data > 5])
```

#### Result

```
A 2  
B 9  
C 0  
D 6  
E 1  
dtype: int64
```

```
-----  
B 9  
D 6  
dtype: int64
```

### Series와 dictionary

- dictionary와 동일하게 series 순회 가능
- dictionary의 key, value가 series의 index, value에 대응하여 변환 가능

#### series와 dictionary

```
dic = {"D":3, "F":7, "E":5}
data = pd.Series(dic)
print(data)
for idx, val in data.items():
    print(idx, val)
```

#### Result

```
D      3
F      7
E      5
dtype: int64
D 3
F 7
E 5
```

### Series의 연산

- 연산 시 동일한 index에 대하여 연산됨
- index가 없는 경우에는 NaN 값이 됨

#### series의 연산

```
data = pd.Series([1,2,3,4,5],  
index=["A","B","C","D","E"])  
data2 = pd.Series([10,20,30],  
index=["A","C","E"])  
result = data + data2  
print(result)
```

#### Result

```
A    11.0  
B     NaN  
C    23.0  
D     NaN  
E    35.0  
dtype: float64
```

### Series 결측치 제거

결측치 : 데이터에서 측정되지 않거나 누락된 값

#### Series 결측치 제거

```
data = pd.Series([1,2,3,4,5],
index=["A", "B", "C", "D", "E"])
data2 = pd.Series([10,20,30],
index=["A", "C", "E"])
result = data + data2
print(data.notnull())
print(data[data.notnull()])
# data.dropna()도 같은 기능
```

#### Result

```
A      True
B      True
C      True
D      True
E      True
dtype: bool
A      1
B      2
C      3
D      4
E      5
dtype: int64
```

## 03 Dataframe

### ◈ Dataframe 생성

- Dataframe : Series의 집합으로 이루어진 데이터 타입
- row(index), series, column 으로 이루어짐

#### Dataframe 생성

```
df = pd.DataFrame(columns=["Email", "Name"])
df["Name"] = ["doori", "minsu"]
df["Email"] = ["dr@gmail.com", "ms@gmail.com"]
print(df)
print(type(df["Name"]))
```

#### Result

	Email	Name
0	dr@gmail.com	doori
1	ms@gmail.com	minsu

<class 'pandas.core.series.Series'>

### Dictionary로부터 dataframe 생성

#### Dictionary로부터 dataframe 생성

```
name = ["doori", "minsu"]
email = ["dr@gmail.com", "ms@gmail.com"]
ids = [1, 2]
dic = {"Name":name, "Email":email, "id": ids}
df = pd.DataFrame(dic)
print(df)
```

#### Result

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2

### ◈ Dataframe의 요소 가져오기

#### Dataframe의 요소 가져오기

```
print(df.index)
print(df.columns)
print(df.values)
```

df

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2

#### Result

```
RangeIndex(start=0, stop=2, step=1)
Index(['Name', 'Email', 'id'], dtype='object')
[['doori' 'dr@gmail.com' 1]
 ['minsu' 'ms@gmail.com' 2]]
```



## 03 Dataframe

### ◆ 인덱스 수정

#### 인덱스 수정

```
index_list = [0, 1]
df = pd.DataFrame(dic, index=index_list)
print(df)
```

#### Result

	Name	Email	id
0	doori	<u>dr@gmail.com</u>	1
1	minsu	<u>ms@gmail.com</u>	2

#### df

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2

## 03 Dataframe

### row(행) 선택 및 추가

#### row 선택 및 추가

```
print(df.loc[1])  
print()  
df.loc[2] = {"Email": "data@gmail.com",  
            "Name": "data", "id": 3}  
print(df)
```

df

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2

#### Result

```
Name      minsu  
Email      ms@gmail.com  
id          2  
Name: 1, dtype: object
```

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2
2	data	data@gmail.com	3

## 03 Dataframe

### row(행) 선택 - indexing과 slicing 활용

#### row slicing

```
df.loc[3] = {"Email": "data2@gmail.com", "Name": "data2", "id": 4}
print(df.loc[1:3])
```

#### Result

	Name	Email	id
1	minsu	ms@gmail.com	2
2	data	data@gmail.com	3
3	data2	data2@gmail.com	4

#### df

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2
2	data	data@gmail.com	3
3	data2	data2@gmail.com	4

## 03 Dataframe

### row(행) 선택 - indexing과 slicing 활용

row slicing 및 column 선택

```
print(df.loc[1:3, ["Email", "id"]])
```

Result

	Email	id
1	ms@gmail.com	2
2	data@gmail.com	3
3	data2@gmail.com	4

df

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2
2	data	data@gmail.com	3
3	data2	data2@gmail.com	4

## 03 Dataframe

### row(행) 선택 - indexing과 slicing 활용

row indexing 및 column 선택

```
print(df.loc[[1,3], ["Email", "Name"]])
```

Result

	Email	Name
1	ms@gmail.com	minsu
3	data2@gmail.com	data2

df

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2
2	data	data@gmail.com	3
3	data2	data2@gmail.com	4

## 03 Dataframe

### column(열) 추가

["column name"]을 이용하여 추가 가능

#### 빈 column 추가

```
df["Address"] = ""  
print(df)
```

#### Result

	Name	Email	id	Address
0	doori	dr@gmail.com	1	
1	minsu	ms@gmail.com	2	
2	data	data@gmail.com	3	
3	data2	data2@gmail.com	4	

df

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2
2	data	data@gmail.com	3
3	data2	data2@gmail.com	4

## 03 Dataframe

### column(열) 추가

리스트를 이용하여 데이터 삽입

row 수와 리스트의 갯수가 맞지 않으면 에러 발생

#### list를 이용한 column 추가

```
df["Address"] = ["Seoul", "Busan", "Jeju", "Deagu"]  
print(df)
```

#### Result

	Name	Email	id	Address
0	doori	dr@gmail.com	1	Seoul
1	minsu	ms@gmail.com	2	Busan
2	data	data@gmail.com	3	Jeju
3	data2	data2@gmail.com	4	Deagu

df

	Name	Email	id
0	doori	dr@gmail.com	1
1	minsu	ms@gmail.com	2
2	data	data@gmail.com	3
3	data2	data2@gmail.com	4

## 03 Dataframe

### append

append를 이용하여 서로 다른 dataframe을 합칠 수 있음

#### append

```
name1 = ["Adam", "Alvin", "Andrew"]
age1 = [23, 32, 33]
name2 = ["Anthony", "Arnold", "Jin"]
age2 = [25, 34, 31]
dic1 = {"Age": age1, "Name": name1}
dic2 = {"Age": age2, "Name": name2}
df1 = pd.DataFrame(dic1)
print(df1)
print('-'*20)
df2 = pd.DataFrame(dic2)
print(df2)
print('-'*20)
df3 = df1.append(df2)
print(df3)
```

#### Result

	Age	Name
0	23	Adam
1	32	Alvin
2	33	Andrew

	Age	Name
0	25	Anthony
1	34	Arnold
2	31	Jin

	Age	Name
0	23	Adam
1	32	Alvin
2	33	Andrew
0	25	Anthony
1	34	Arnold
2	31	Jin



## 03 Dataframe



### append

append시 인덱스를 새롭게 정의: ignore\_index

ignore\_index

```
df3 = df1.append(df2, ignore_index=True)
print(df3)
```

Result

	Age	Name
0	23	Adam
1	32	Alvin
2	33	Andrew
3	25	Anthony
4	34	Arnold
5	31	Jin

## 03 Dataframe



### concatenate

concat을 사용하여 축을 지정해 합칠 수 있음

#### concat1

```
df3 = pd.concat([df1, df2]).reset_index(drop=True)
print(df3)
```

#### Result

	Age	Name
0	23	Adam
1	32	Alvin
2	33	Andrew
3	25	Anthony
4	34	Arnold
5	31	Jin

#### concat2

```
df4 = pd.concat([df1, df2], axis=1)
print(df4)
```

#### Result

	Age	Name	Age	Name
0	23	Adam	25	Anthony
1	32	Alvin	34	Arnold
2	33	Andrew	31	Jin

## 03 Dataframe

### Group by

특정 column의 중복되는 row들을 합쳐 새로운 데이터 프레임을 만드는 방식

concat1

```
names = ['apple', 'pear', 'apple', 'banana', 'pear']
quantity = [3, 5, 1, 2, 2]
g_df = pd.DataFrame({'Name': names, 'Quantity': quantity})
print(g_df)
```

Result

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

### groupby-size

설정된 column으로 공통된 row 데이터가 합쳐지고 갯수 출력

#### groupby-size

```
print(g_df.groupby("Name").size())
```

#### Result

```
Name
apple      2
banana     1
pear       2
dtype: int64
```

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### groupby-size

설정된 column으로 공통된 row 데이터가 합쳐지고 갯수 출력

#### groupby-size

```
result_df = g_df.groupby("Name").size().reset_index(name="counts")  
print(result_df)
```

#### Result

	Name	counts
0	apple	2
1	banana	1
2	pear	2

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### groupby-sum

설정된 column으로 공통된 row 데이터가 합쳐지고 합 출력

#### groupby-sum

```
result_df = g_df.groupby("Name").sum()  
print(result_df)
```

#### Result

Quantity	
Name	
apple	4
banana	2
pear	7

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### sort

설정된 column으로 오름차순/내림차순으로 정렬하는 함수

sort

```
result_df = g_df.sort_values(by=["Quantity"], ascending=False)
result_df.reset_index(drop=True, inplace=True)
print(result_df)
```

Result

	Name	Quantity
0	pear	5
1	apple	3
2	banana	2
3	pear	2
4	apple	1

g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### min, max

name으로 그룹핑된 데이터에서 최소값/최대값 quantity 출력

#### min, max

```
print(g_df.groupby("Name").agg("min").reset_index())  
print('-'*20)  
print(g_df.groupby("Name").agg("max").reset_index())
```

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

#### Result

	Name	Quantity
0	apple	1
1	banana	2
2	pear	2

---

	Name	Quantity
0	apple	3
1	banana	2
2	pear	5



## 03 Dataframe

### mean, sum

name으로 그룹핑된 데이터에서 평균값, 합에 대한 quantity 출력

#### mean, sum

```
print(g_df.groupby("Name").agg("mean").reset_index())  
print('-'*20)  
print(g_df.groupby("Name").agg("sum").reset_index())
```

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

#### Result

	Name	Quantity
0	apple	2.0
1	banana	2.0
2	pear	3.5

-----

	Name	Quantity
0	apple	4
1	banana	2
2	pear	7

## 03 Dataframe

### ◆ agg로 여러 개 column 생성

name으로 그룹핑된 데이터에서 설정한 column들을 만들어 출력

mean, sum

```
df_agg = g_df.groupby("Name").agg(["min", "max", "mean"]).reset_index()
print(df_agg)
```

Result

	Name	Quantity		
		min	max	mean
0	apple	1	3	2.0
1	banana	2	2	2.0
2	pear	2	5	3.5

g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### select

dataframe을 출력하는 방법들

#### head, tail

```
print(g_df.head(2))  
print('-'*20)  
print(g_df.tail(3))
```

#### Result

	Name	Quantity
0	apple	3
1	pear	5
-----		
	Name	Quantity
2	apple	1
3	banana	2
4	pear	2

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### select

offset index를 사용하여 선택적 출력 가능

offset

```
print(g_df[1:3])  
print('-'*20)  
print(g_df.loc[2:])
```

Result

	Name	Quantity
1	pear	5
2	apple	1

---

	Name	Quantity
2	apple	1
3	banana	2
4	pear	2

g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe



### select

describe로 통계치 요약 가능

describe

```
print(g_df.describe())
```

Result

	Quantity
count	5.000000
mean	2.600000
std	1.516575
min	1.000000
25%	2.000000
50%	2.000000
75%	3.000000
max	5.000000

g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### ◆ 기초 통계 (sum, mean, var, std)

#### 기초 통계

```
print(g_df.Quantity.sum())  
print(g_df.Quantity.mean())  
print(g_df.Quantity.var())  
print(g_df.Quantity.std())
```

#### Result

```
13  
2.6  
2.3  
1.51657508881031
```

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### 기초 통계 (min, max, argmin, argmax)

#### 기초 통계

```
print(g_df.Quantity.min())  
print(g_df.Quantity.max())  
print(g_df.Quantity.argmin())  
print(g_df.Quantity.argmax())
```

#### Result

```
1  
5  
2  
1
```

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

## 03 Dataframe

### 정렬 (sort)

#### 정렬

```
print(g_df.sort_values('Quantity'))  
print('-'*20)  
print(g_df.sort_values('Name', ascending=False))
```

#### g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

#### Result

	Name	Quantity
2	apple	1
3	banana	2
4	pear	2
0	apple	3
1	pear	5

---

	Name	Quantity
1	pear	5
4	pear	2
3	banana	2
0	apple	3
2	apple	1



## 03 Dataframe

### 순위 (rank)

순위

```
print(g_df.Quantity.rank())
print('-'*30)
g_df['Rank'] = g_df.Quantity.rank()
print(g_df)
```

g\_df

	Name	Quantity
0	apple	3
1	pear	5
2	apple	1
3	banana	2
4	pear	2

Result

```
0    4.0
1    5.0
2    1.0
3    2.5
4    2.5
Name: Quantity, dtype: float64
-----
   Name  Quantity  Rank
0  apple         3   4.0
1  pear         5   5.0
2  apple         1   1.0
3 banana         2   2.5
4  pear         2   2.5
```

## 03 Dataframe

### 빈도수 (counts)

#### 빈도수

```
print(g_df['Quantity'].value_counts())  
print('-'*20)  
print(g_df.count())
```

#### g\_df

	Name	Quantity	Rank
0	apple	3	4.0
1	pear	5	5.0
2	apple	1	1.0
3	banana	2	2.5
4	pear	2	2.5

#### Result

```
2    2  
5    1  
3    1  
1    1  
Name: Quantity, dtype: int64  
-----  
Name          5  
Quantity      5  
Rank          5  
dtype: int64
```

## 03 Dataframe

### ◇ 원소 체크

#### 원소 체크

```
if 'apple' in list(g_df['Name']):  
    print("True")
```

#### Result

True

#### g\_df

	Name	Quantity	Rank
0	apple	3	4.0
1	pear	5	5.0
2	apple	1	1.0
3	banana	2	2.5
4	pear	2	2.5

## 03 Dataframe

### ◈ T (전치, Transpose)

행과 열을 바꾸어 줌

T

```
print(g_df.T)
```

Result

	0	1	2	3	4
Name	apple	pear	apple	banana	pear
Quantity	3	5	1	2	2
Rank	4.0	5.0	1.0	2.5	2.5

g\_df

	Name	Quantity	Rank
0	apple	3	4.0
1	pear	5	5.0
2	apple	1	1.0
3	banana	2	2.5
4	pear	2	2.5

## 03 Dataframe

### filtering

Name colum에 'e'가 있는 데이터 필터링

#### filtering

```
print(g_df[g_df["Name"].str.contains("e")])
```

#### Result

	Name	Quantity	Rank
0	apple	3	4.0
1	pear	5	5.0
2	apple	1	1.0
4	pear	2	2.5

#### g\_df

	Name	Quantity	Rank
0	apple	3	4.0
1	pear	5	5.0
2	apple	1	1.0
3	banana	2	2.5
4	pear	2	2.5

## 03 Dataframe

### csv/excel 파일 읽기, 쓰기

No module named 'xlsxwriter' 오류 방지를 위한 xlsxwriter 모듈 설치

```
!pip install xlsxwriter
```

#### csv/excel 파일 읽기, 쓰기

```
g_df.to_csv('./foo.csv', index=False, sep='\t')  
df = pd.read_csv('./foo.csv')
```

```
g_df.to_excel('./foo.xlsx', sheet_name='Sheet1', engine='xlsxwriter')  
df = pd.read_excel('./foo.xlsx', 'Sheet1')
```

파일



..

sample\_data



foo.csv



foo.xlsx

### Pandas Pivot

데이터 프레임의 컬럼 데이터에서 Index, columns, values를 선택해서 데이터 프레임을 만드는 방법

Pivot

```
df.pivot(index, cols, values)
```

### Pandas Pivot

데이터 프레임의 컬럼 데이터에서 Index, columns, values를 선택해서 데이터 프레임을 만드는 방법

Pivot

```
df.pivot(index, cols, values)
```



## 예제 : 타이타닉 데이터

기계 학습을 사용해서 타이타닉 난파선에서 살아남은 승객을 예측하는 모델 만들기  
 승객 데이터 (성별, 나이, 신분 등) 를 이용하여 어떤 종류의 사람이 생존할 가능성이 높은지 예측



	A	B	C	D	E	F	G	H	I	J	K	L
1	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
3	2	1	1	Cummings, Mrs. John Bradley (Flk)	female	38	1	0	PC 17599	71.2833	C85	C
4	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2.	7.925		S
5	4	1	1	Futrelle, Mrs. Jacques Heath (Lil)	female	35	1	0	113803	53.1	C123	S
6	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
7	6	0	3	Moran, Mr. James	male			0	330877	8.4583		Q
8	7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
9	8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S
10	9	1	3	Johnson, Mrs. Oscar W (Elisabef)	female	27	0	2	347742	11.1333		S
11	10	1	2	Nasser, Mrs. Nicholas (Adele Arf)	female	14	1	0	237736	30.0708		C
12	11	1	3	Sandstrom, Miss. Marguerite R	female		4	1	PP 9549	16.7	G6	S
13	12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
14	13	0	3	Saunders, Mr. William Henry	male	20	0	0	A/5. 2151	8.05		S
15	14	0	3	Andersson, Mr. Anders Johan	male	39	1	5	347082	31.275		S
16	15	0	3	Vestrom, Miss. Hulda Amanda	female	14	0	0	350406	7.8542		S
17	16	1	2	Hewlett, Mrs. (Mary D Kingcom)	female	55	0	0	248706	16		S
18	17	0	3	Rice, Master. Eugene	male		2	4	382652	29.125		Q
19	18	1	2	Williams, Mr. Charles Eugene	male			0	244272	12		S



### 예제 : 타이타닉 데이터

Survived : 0 = 사망, 1 = 생존

Pclass : 1 = 1등석, 2 = 2등석, 3 = 3등석

Sex : male = 남성, female = 여성

Age : 나이

SibSp : 타이타닉 호에 동승한 자매 / 배우자의 수

Parch : 타이타닉 호에 동승한 부모 / 자식의 수

Ticket : 티켓 번호

Fare : 승객 요금

Cabin : 방 호수

Embarked : 탑승지, C = 세르부르, Q = 퀸즈타운, S = 사우샘프턴

### 타이타닉 데이터 읽기

#### 데이터 read

```
import pandas as pd
titanic = pd.read_csv("./train.csv")
titanic.head()
```

#### Result

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

### ◆ 성별, 좌석등급에 따른 데이터의 수

groupby로 성별과 좌석등급의 중복 데이터를 합쳐주고 size 함수로 데이터의 갯수 컬럼(Counts)를 추가

#### Code

```
titanic_df1 = titanic.groupby(["Sex",  
"Pclass"]).size().reset_index(name="Counts")  
titanic_df1
```

#### Result

	Sex	Pclass	Counts
0	female	1	94
1	female	2	76
2	female	3	144
3	male	1	122
4	male	2	108
5	male	3	347

### ◈ 성별, 좌석등급에 따른 데이터의 수

pivot을 이용하여 index, column, value 데이터를 설정

#### Code

```
titanic_df2 = titanic_df1.pivot("Sex",  
                                "Pclass", "Counts")  
titanic_df2
```

#### Result

Pclass	1	2	3
female	94	76	144
male	122	108	347

### ◈ 성별에 따른 생존자 수

groupby로 성별과 생존의 중복 데이터를 합쳐주고 size 함수로 데이터의 갯수 컬럼(Counts)를 추가

#### Code

```
df2 = titanic.groupby(["Sex",  
"Survived"]).size().reset_index(name="Counts")  
df2
```

#### Result

	Sex	Survived	Counts
0	female	0	81
1	female	1	233
2	male	0	468
3	male	1	109

### ◈ 성별에 따른 생존자 수

pivot을 이용하여 index, column, value 데이터를 설정

#### Code

```
df2.pivot("Sex", "Survived", "Counts")
```

#### Result

Survived	0	1
Sex		
female	81	233
male	468	109

### ❖ 객실 등급에 따른 생존자 수

groupby로 객실등급과 생존의 중복 데이터를 합쳐주고 size 함수로 데이터의 갯수 컬럼(Counts)를 추가

#### Code

```
df3 = titanic.groupby(["Pclass",  
"Survived"]).size().reset_index(name="Counts")  
df3
```

#### Result

	Pclass	Survived	Counts
0	1	0	80
1	1	1	136
2	2	0	97
3	2	1	87
4	3	0	372
5	3	1	119



### ❖ 객실 등급에 따른 생존자 수

pivot을 이용하여 index, column, value 데이터를 설정

#### Code

```
result = df3.pivot("Pclass", "Survived",  
"Counts")  
result
```

#### Result

Survived	0	1
Pclass		
1	80	136
2	97	87
3	372	119

### Pivot table을 이용하기

데이터의 수를 나타내기 위해서 Counts 컬럼을 추가

#### Code

```
titanic["Counts"] = 1  
titanic.tail()
```

#### Result

	PassengerId	Survived	Pclass		Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Counts
886	887	0	2		Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S	1
887	888	1	1		Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S	1
888	889	0	3		Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S	1
889	890	1	1		Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C	1
890	891	0	3		Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q	1

### ◈ 성별, 좌석등급에 따른 데이터의 수

#### Code

```
import numpy as np
titanic.pivot_table("Counts", ["Sex"],
["Pclass"], aggfunc=np.sum)
```

#### Result

Pclass	1	2	3
Sex			
female	94	76	144
male	122	108	347

### ◈ 성별에 따른 생존자 수

#### Code

```
titanic.pivot_table("Counts", ["Sex"],  
["Survived"], aggfunc=np.sum)
```

#### Result

Survived	0	1
Sex		
female	81	233
male	468	109

### ❖ 객실등급에 따른 생존자 수

#### Code

```
titanic.pivot_table("Counts", ["Pclass"],  
["Survived"], aggfunc=np.sum)
```

#### Result

Survived	0	1
Pclass		
1	80	136
2	97	87
3	372	119

## 04 Pivot

### ❖ 여러 개의 index 데이터로 pivot

#### Code

```
result = titanic.pivot_table("Counts", ["Sex",  
"Pclass"], ["Survived"], aggfunc=np.sum)  
result
```

#### Result

Sex	Pclass	Survived	
		0	1
female	1	3	91
	2	6	70
	3	72	72
male	1	77	45
	2	91	17
	3	300	47

### ◈ 성별에 따른 생존자 Total 컬럼, 로우 추가 및 삭제

#### Code

```
df = titanic.pivot_table("Counts",  
["Survived"], ["Sex"], aggfunc=np.sum)  
df
```

total 데이터 추가

#### Code

```
df["total"] = df["female"] + df["male"]  
df
```

#### Result

	Sex	female	male
Survived			
0		81	468
1		233	109

#### Result

	Sex	female	male	total
Survived				
0		81	468	549
1		233	109	342

### ◆ 성별에 따른 생존자 Total 컬럼, 로우 추가 및 삭제

#### Code

```
df.loc["total"] = df.loc[0] + df.loc[1]  
df
```

#### Result

Sex	female	male	total
Survived			
0	81	468	549
1	233	109	342
total	314	577	891



### ◈ 성별에 따른 생존자 Total 컬럼, 로우 추가 및 삭제

데이터 삭제 방법 - row

#### Code

```
df.drop("total", inplace=True)  
df
```

데이터 삭제 방법 - column

#### Code

```
df.drop("total", axis=1, inplace=True)  
df
```

#### Result

Sex	female	male	total
Survived			
0	81	468	549
1	233	109	342

#### Result

Sex	female	male
Survived		
0	81	468
1	233	109

### ◈ Pivot table에서 NaN에 대한 처리

#### Code

```
df = titanic.pivot_table("Counts", ["Survived"], ["Parch", "Pclass"],  
aggfunc=np.sum)  
df
```

#### Result

Parch	0			1			2			3		4		5	6
Pclass	1	2	3	1	2	3	1	2	3	2	3	1	3	3	3
Survived															
0	64.0	86.0	295.0	10.0	8.0	35.0	5.0	3.0	32.0	NaN	2.0	1.0	3.0	4.0	1.0
1	99.0	48.0	86.0	21.0	24.0	20.0	16.0	13.0	11.0	2.0	1.0	NaN	NaN	1.0	NaN

### ◈ Pivot table에서 NaN에 대한 처리

#### Code

```
df = titanic.pivot_table("Counts", ["Survived"],  
["Parch", "Pclass"], aggfunc=np.sum, dropna=False, fill_value=0)  
df
```

#### Result

Parch	0			1			2			3			4			5			6		
Pclass	1	2	3	1	2	3	1	2	3	1	...	3	1	2	3	1	2	3	1	2	3
Survived																					
0	64	86	295	10	8	35	5	3	32	0	...	2	1	0	3	0	0	4	0	0	1
1	99	48	86	21	24	20	16	13	11	0	...	1	0	0	0	0	0	1	0	0	0

2 rows x 21 columns