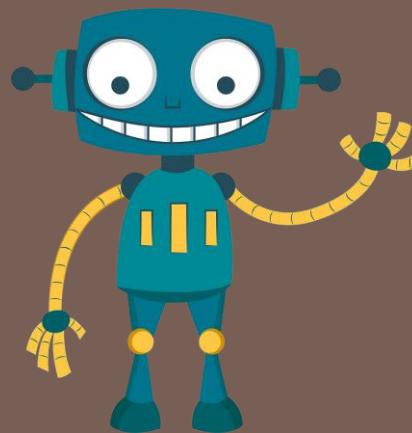
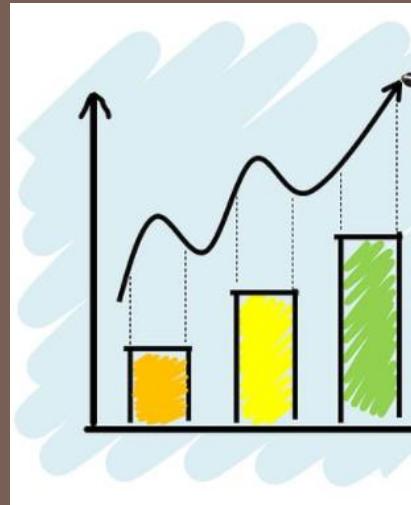


# 파이썬 인스프레스



15장 데이터 과학

# 학습 목표

- 데이터 과학의 개념을 살펴본다.
- 데이터 과학의 응용 분야를 살펴본다.
- 판다스의 각 기능을 간단히 살펴본다.
- 실제 CSV 파일을 읽어서 분석해본다.



# 이번 장에서 만드는 프로그램

- 타이타닉 승객 파일에서 여러 가지 정보를 추출해본다. 예를 들어서 승객 중에서 최고령자가 누구였을까?



Second



Third



Crew

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
1	0	3	Braund, Mr. Owen Harris	male	22	1	0
2	1	1	Cumings, Mrs. John	female	38	1	0
3	1	3	Heikkinen, Miles	female	26	0	0
4	1	1	Futrelle, Mrs. Jeanne	female	35	1	0
5	0	3	Allen, Mr. William Henry	male	35	0	0
6	0	3	Moran, Mr. James	male		0	0

# 데이터 과학이란?

- 현대 사회는 인구 통계, 교통 정보, 날씨 데이터, 회사의 매출 데이터, 웹 검색 기록, 메신저 대화 기록, 페이스북 댓글, [블로그](#) 사진, CCTV 정보 등과 같은 각종 데이터로 가득 차 있다. 이들 데이터에서 의미 있는 정보를 추출할 수 있을까?

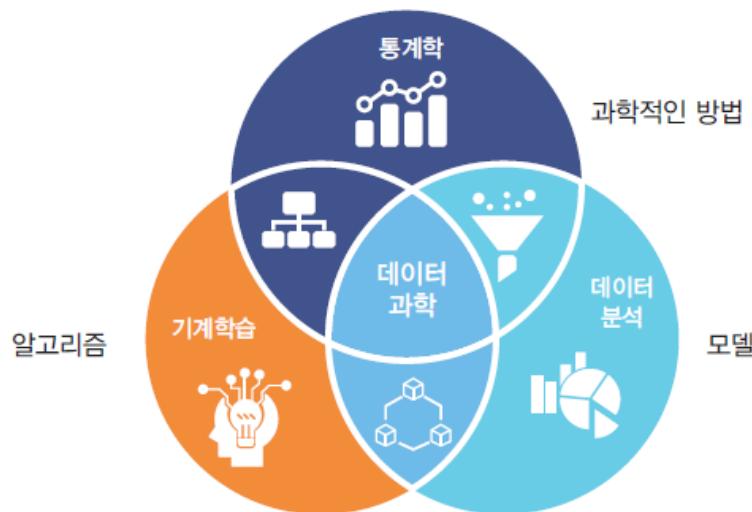


# 데이터 과학이란?

- ”카드 결제 데이터나 택배 송장 데이터를 이용하여 장사가 잘 되는 지역을 찾을 수 있을까?”
- ”산악기상 데이터나 등산로 등의 정보를 이용하여 산림재해를 예측할 수 있을까?”
- ”CCTV가 적게 설치된 곳에서 실제로 범죄가 많이 일어날까?”
- ”20-30대가 많이 사는 지역의 커피 샵이 더 많은 매출을 올리고 있을까?”
- ”지하철 승하차가 가장 많이 발생하는 역은 어떤 역일까“

# 데이터 과학과 파이썬

- 데이터 과학은 여러 학문 분야에 걸친 접근 방식(통계학, 컴퓨터 과학, 기계 학습 등의 많은 분야에서 추출한 기법과 이론)을 필요로 한다.
- 파이썬은 수많은 라이브러리를 가지고 있으며 데이터 과학의 요구를 쉽게 처리할 수 있는 기능을 내장하고 있는 동시에, 범용 프로그래밍 언어이기 때문에 최근에 데이터 과학 언어로 각광을 받고 있다



# 데이터 과학 적요의 예

- 서울시에서도 심야 버스 노선을 설계할 때, 데이터를 이용하였다.
  - 구체적으로 택시 승하차 정보와 이동 통신 사업자 KT의 통화량 데이터를 사용하였다

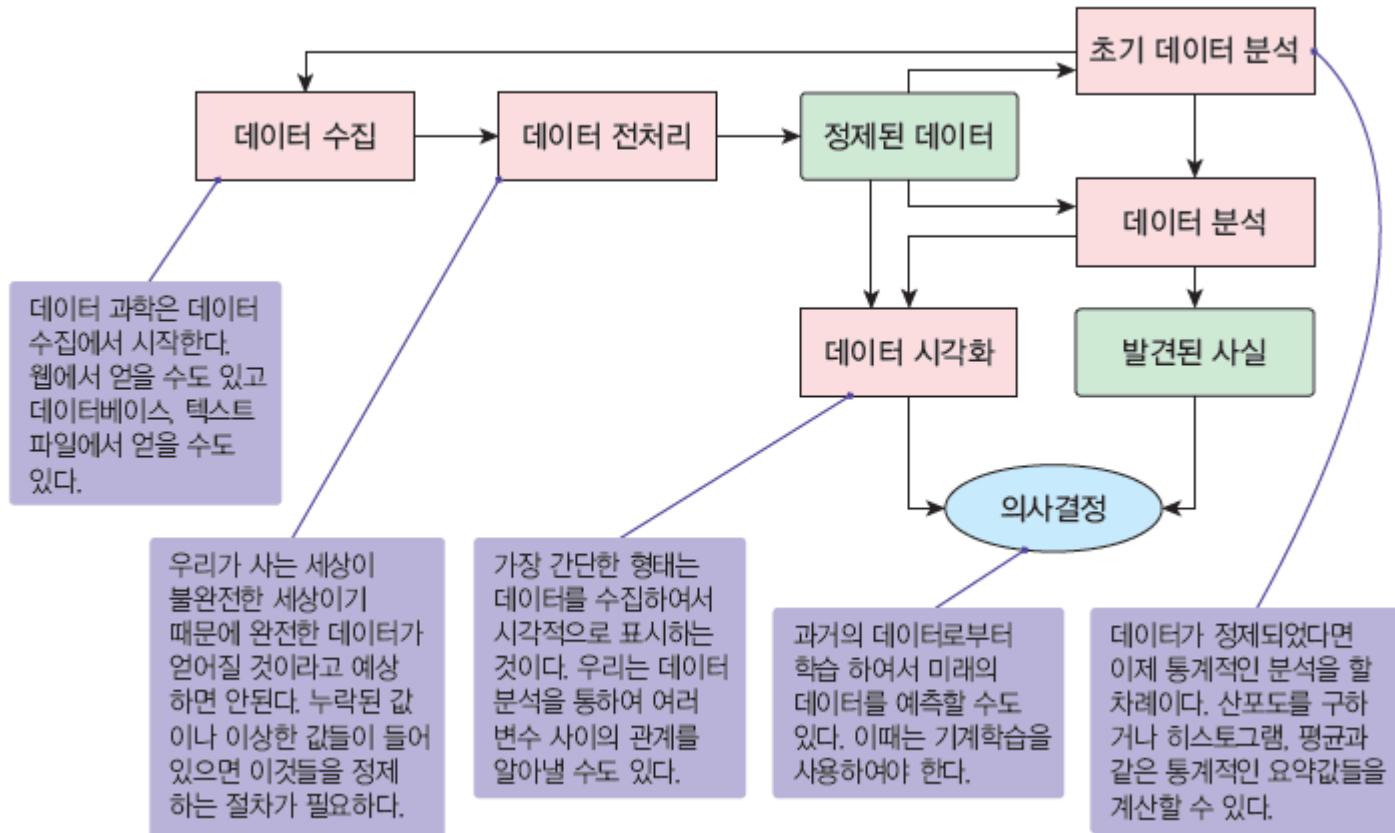


# 데이터 과학의 유통

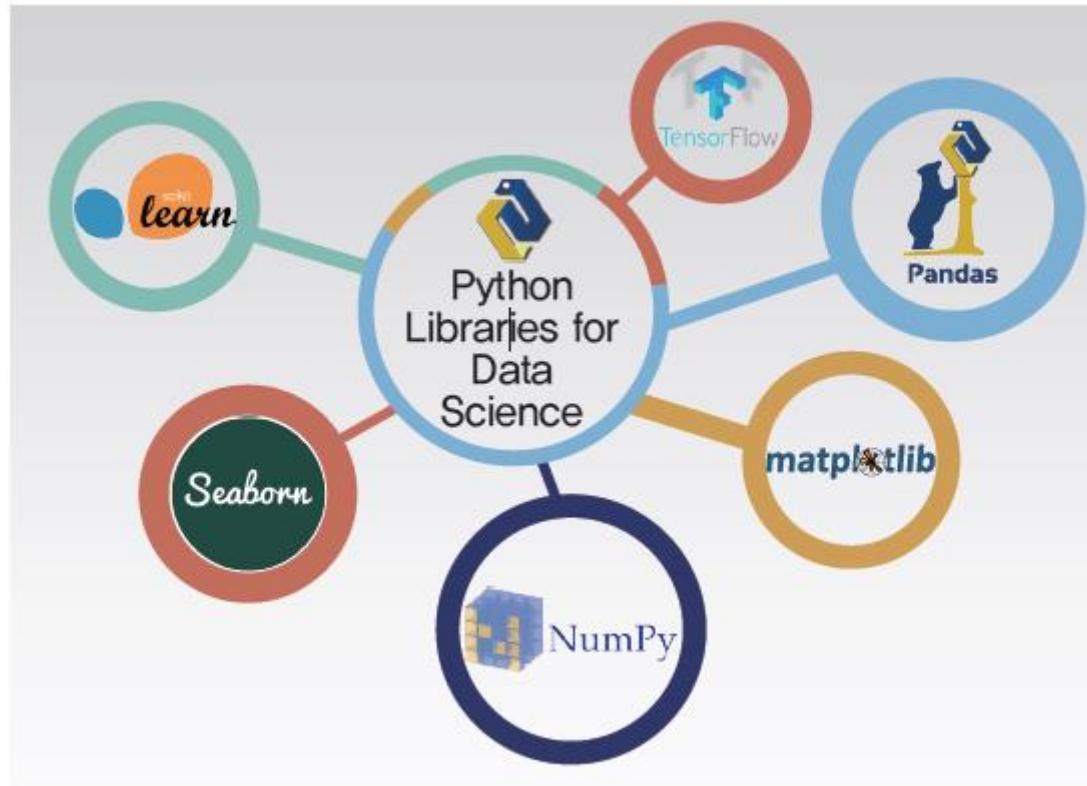


데이터 사이언스 활용

# 데이터 처리 절차



# 데이터 과학을 위한 파이썬 라이브러리



# 판다스와 파이썬

- 판다스(Pandas)는 강력한 데이터 구조를 사용하여 고성능 데이터 조작 및 데이터 분석에 사용되는 오픈 소스 파이썬 라이브러리이다.



# 판다스로 할 수 있는 작업

- 판다스로 CSV 파일이나 TSV 파일, 엑셀 파일 등을 열 수 있다.
- mean()로 모든 열의 평균을 계산할 수 있다.
- corr()로 데이터 프레임의 열 사이의 상관 관계를 계산할 수 있다.
- 조건을 사용하여 데이터를 필터링할 수 있다.
- sort\_values()로 데이터를 정렬할 수 있다.
- groupby()를 이용하여 기준에 따라 몇 개의 그룹으로 데이터를 분할 할 수 있다.
- 데이터의 누락 값을 확인할 수 있다.

# 판다스의 데이터 구조

- 시리즈(Series):

11	73	53	27	52	65	74	98	13	72
----	----	----	----	----	----	----	----	----	----

- 데이터 프레임(DataFrame):

	이름	나이	성별	평점
0	김철수	19	Male	3.45
1	김영희	22	Female	4.1
2	김명수	20	Male	3.9
3	최자영	26	Female	4.5

행(row)

열(column)

# index 와 columns 객체

- 데이터 프레임에서는 행이나 열에 붙인 레이블을 중요시한다.
- index 객체는 행들의 레이블(label)이고 columns 객체는 열들의 레이블이 저장된 객체이다.

The diagram illustrates a DataFrame structure. A red box highlights the first column, which contains numerical indices (0, 1, 2, 3). A blue box labeled 'index' points to this column. Another red box highlights the last column, which contains categorical labels ('이름', '나이', '성별', '평점'). A blue box labeled 'columns' points to this column. The rows represent individual data points: row 0 has values '김철수', 19, 'Male', 3.45; row 1 has values '김영희', 22, 'Female', 4.1; row 2 has values '김명수', 20, 'Male', 3.9; and row 3 has values '최자영', 26, 'Female', 4.5.

	이름	나이	성별	평점
0	김철수	19	Male	3.45
1	김영희	22	Female	4.1
2	김명수	20	Male	3.9
3	최자영	26	Female	4.5

# 타이타닉 데이터셋

- 우리는 판다스 튜토리얼 웹페이지에서 타이타닉 탑승자에 대한 데이터셋 titanic.csv를 다운로드 받을 수 있다

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
1	0	3	Braund, Mr. Owen	male	22	1	0
2	1	1	Cumings, Mrs. Frances	female	38	1	0
3	1	3	Heikkinen, Miss. Ida	female	26	0	0
4	1	1	Futrelle, Mrs. Jean	female	35	1	0
5	0	3	Allen, Mr. William	male	35	0	0
6	0	3	Moran, Mr. James	male		0	0

# 타이타닉 데이터셋

- **PassengerId**: 승객의 ID이다.
- **Survived**: 생존 여부
- **Pclass**: 탑승 등급을 나타낸다. 클래스 1, 클래스 2, 클래스 3의 3가지 클래스가 있다.
- **Name**: 승객의 이름.
- **Sex**: 승객의 성별.
- **Age**: 승객의 나이.
- **SibSp**: 승객에게 형제 자매와 배우자가 있음을 나타낸다.
- **Parch**: 승객이 혼자인지 또는 가족이 있는지 여부.
- **Ticket**: 승객의 티켓 번호.
- **Fare**: 운임.
- **Cabin** : 승객의 선실.
- **Embarked**: 탑승한 지역.

# 타이타닉 CSV 파일을 읽으려면?

```
>>> import pandas as pd  
>>> titanic = pd.read_csv("d:\\titanic.csv")  
>>> titanic  


|     | PassengerId | Survived | Pclass | ... | Fare    | Cabin | Embarked |
|-----|-------------|----------|--------|-----|---------|-------|----------|
| 0   | 1           | 0        | 3      | ... | 7.2500  | NaN   | S        |
| 1   | 2           | 1        | 1      | ... | 71.2833 | C85   | C        |
| 2   | 3           | 1        | 3      | ... | 7.9250  | NaN   | S        |
| 3   | 4           | 1        | 1      | ... | 53.1000 | C123  | S        |
| 4   | 5           | 0        | 3      | ... | 8.0500  | NaN   | S        |
| ..  | ...         | ...      | ...    | ... | ...     | ...   | ...      |
| 886 | 887         | 0        | 2      | ... | 13.0000 | NaN   | S        |
| 887 | 888         | 1        | 1      | ... | 30.0000 | B42   | S        |
| 888 | 889         | 0        | 3      | ... | 23.4500 | NaN   | S        |
| 889 | 890         | 1        | 1      | ... | 30.0000 | C148  | C        |
| 890 | 891         | 0        | 3      | ... | 7.7500  | NaN   | Q        |



[891 rows x 12 columns]


```

# 타이타닉 승객들의 나이를 추출하려면?

```
>> titanic["Age"]
```

```
0    22.0  
1    38.0  
2    26.0  
3    35.0  
4    35.0  
...  
Name: Age, Length: 891, dtype: float64
```

# 타이타닉 탑승객 중에서 최고령자를 알고 싶다면?

```
>>> titanic["Age"].max()
```

```
80.0
```

# 타이타닉 승객 데이터에 대한 기본 통계를 알고 싶다면?

- describe() 메소드는 숫자 데이터에 대한 간략한 개요를 제공한다. 문자열 데이터는 처리하지 않는다.

```
>>> titanic.describe()
```

	PassengerId	Survived	Pclass	...	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	...	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	...	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	...	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	...	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	...	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	...	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	...	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	...	8.000000	6.000000	512.329200

[8 rows x 7 columns]

# 데이터 시리즈 생성하기

- 시리즈는 이름이 붙여진 1차원적인 배열이나 마찬가지이다. 가장 기본적인 방법은 파이썬의 리스트에서 생성하는 것이다.

```
>>> data = ['Kim', 'Park', 'Lee', 'Choi']
>>> ser = pd.Series(data)
>>> ser
0    Kim
1   Park
2    Lee
3   Choi
```

# 데이터 프레임 생성하기

- 데이터 프레임은 행과 열에 이름이 붙여진 2차원 배열이다.

```
>>> data = {'Name':['Kim', 'Park', 'Lee', 'Choi'],
   'Age':[20, 23, 21, 26]}
>>> df = pd.DataFrame(data)
>>> df
   Name  Age
0  Kim   20
1  Park   23
2  Lee   21
3  Choi   26
```

# 데이터 프레임 생성하기

- 데이터 프레임에 index를 붙이려면 다음과 같이 index 매개 변수를 사용할 수 있다.

```
>>> df = pd.DataFrame(data, index=["학번 1", "학번 2", "학번 3", "학번 4"])
```

```
>>> df
```

Name	Age
학번 1	Kim 20
학번 2	Park 23
학번 3	Lee 21
학번 4	Choi 26

# CSV 파일을 읽어서 데이터 프레임 생성하기

- 판다스에서 데이터를 읽는 메소드는 항상 `read_xxx()`와 같은 형태이고, 반대로 데이터를 파일에 쓰는 메소드는 `to_xxx()`의 형태를 가진다.

```
titanic = pd.read_csv("d:\\titanic.csv")
```

```
>>> titanic.dtypes
```

Survived	int64
Pclass	int64
Name	object
Sex	object
Age	float64
Seconds/Spouses Aboard	int64
Parents/Children Aboard	int64
Fare	float64

# 인덱스 변경

- 파일에서 읽을 때 `index`를 변경할 수 있다. 예를 들어서 첫 번째 열을 `index` 객체로 사용할 수도 있다.

```
>>> titanic = pd.read_csv('d://titanic.csv', index_col=0)
>>> titanic
   Survived  Pclass ... Cabin Embarked
PassengerId      ...
1              0     3 ...   NaN      S
2              1     1 ... C85      C
3              1     3 ...   NaN      S
4              1     1 ... C123      S
5              0     3 ...   NaN      S
...
```

# 데이터 프레임의 몇 개 행을 보려면?

```
>>> titanic.head(8)
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S
3	4	1	1	...	53.1000	C123	S
4	5	0	3	...	8.0500	NaN	S
5	6	0	3	...	8.4583	NaN	Q
6	7	0	1	...	51.8625	E46	S
7	8	0	3	...	21.0750	NaN	S

[8 rows x 12 columns]

# 데이터 프레임을 엑셀 파일로 저장하려면?

```
>>> titanic.to_excel('titanic.xlsx', sheet_name='passengers', index=False)
```

# 이렇게 저장한 엑셀 파일을 다시 읽으려면 다음과 같이 한다.

```
>>> titanic = pd.read_excel('titanic.xlsx', sheet_name='passengers')
```

# 난수로 데이터 프레임 채우기

```
>>> df = pd.DataFrame(np.random.randint(0, 100, size=(5, 4)), columns=list('ABCD'))  
>>> df  
A   B   C   D  
0  59  71  53  19  
1  92  13  88  3  
2  69  89  9  77  
3  71  47  54  9  
4  44  45  90  68
```

# Lab: 데이터 프레임 만들어 보기

*countries.csv*

```
code,country,area,capital,population
KR,Korea,98480,Seoul,48422644
US,USA,9629091,Washington,310232863
JP,Japan,377835,Tokyo,127288000
CN,China,9596960,Beijing,1330044000
RU,Russia,17100000,Moscow,140702000
```

# Sol:

```
>>> import pandas as pd  
>>> countries = pd.read_csv('d://countries.csv')  
  
>>> countries  
code country area capital population  
0 KR Korea 98480 Seoul 48422644  
1 US USA 9629091 Washington 310232863  
2 JP Japan 377835 Tokyo 127288000  
3 CN China 9596960 Beijing 1330044000  
4 RU Russia 17100000 Moscow 140702000
```

# 타이타닉 데이터에서 승객의 나이만 추출하려면?

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
1	0	3	Braund, Mr. Owen	male	22	1	0
2	1	1	Cumings, Mrs. Home	female	38	1	0
3	1	3	Heikkinen, Miss. Laina	female	26	0	0
4	1	1	Futrelle, Mrs. Jacques	female	35	1	0
5	0	3	Allen, Mr. William	male	35	0	0
6	0	3	Moran, Mr. James	male		0	0



Age
22
38
26
35
35

```
>>> ages = titanic["Age"]
```

```
>>> ages.head()
```

```
0    22.0
```

```
1    38.0
```

```
2    26.0
```

```
3    35.0
```

```
4    35.0
```

```
Name: Age, dtype: float64
```

# 타이타닉 탑승객의 이름, 나이, 성별을 도시에 알고 싶으면?

```
>>> titanic[["Name", "Age", "Sex"]]
```

	Name	Age	Sex
0	Braund, Mr. Owen Harris	22.0	male
1	Cumings, Mrs. John Bradley (Florence Briggs Th... e)	38.0	female
2	Heikkinen, Miss. Laina	26.0	female
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	female
...			

# 20세 미만의 승객만 추리려면?(필터링)

```
>>> below_20 = titanic[titanic["Age"] < 20]
```

```
>>> below_20.head()
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
7	8	0	3	...	21.0750	NaN	S
9	10	1	2	...	30.0708	NaN	C
10	11	1	3	...	16.7000	G6	S
14	15	0	3	...	7.8542	NaN	S
16	17	0	3	...	29.1250	NaN	Q

[5 rows x 12 columns]

# 1등석이나 2등석에 탑승한 승객들을 출력하려면?

- 조건식과 유사하게 `isin()` 함수는 제공된 리스트에 있는 값들이 들어 있는 각 행에 대하여 `True`를 반환한다. `df["Pclass"].isin([1, 2])`은 `Pclass` 열이 1 또는 2인 행을 확인한다.

```
>>> titanic[titanic["Pclass"].isin([1, 2])]
```

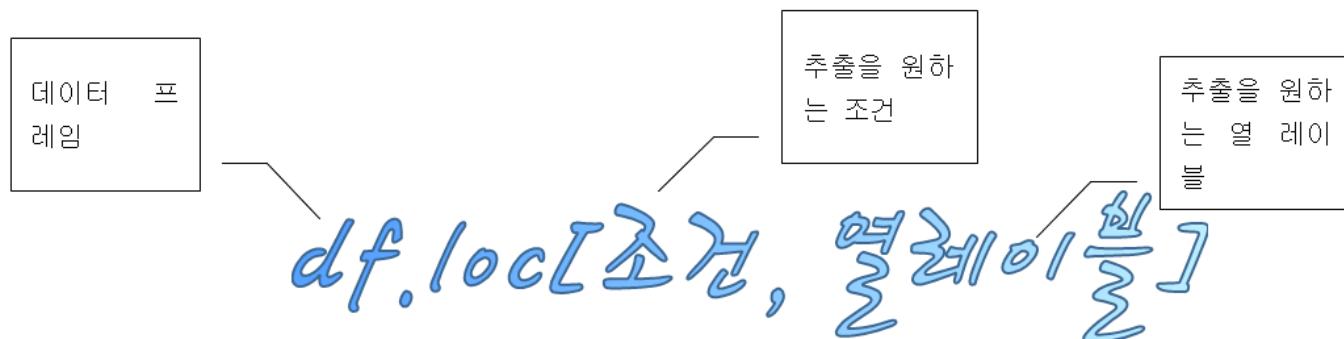
	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
1	2	1	1	...	71.2833	C85	C
3	4	1	1	...	53.1000	C123	S
6	7	0	1	...	51.8625	E46	S
9	10	1	2	...	30.0708	NaN	C

...

[400 rows x 12 columns]

# 20세 미만의 승객 이름에만 관심이 있다면?

```
>>> titanic.loc[titanic["Age"] < 20, "Name"]  
7      Palsson, Master. Gosta Leonard  
9      Nasser, Mrs. Nicholas (Adele Achem)  
10     Sandstrom, Miss. Marguerite Rut  
14     Vestrom, Miss. Hulda Amanda Adolfina  
16      Rice, Master. Eugene  
...  
Name: Name, Length: 164, dtype: object
```



# 20행에서 23행, 5열에서 7열에만 관심이 있다면?

```
>>> titanic.iloc[20:23, 5:7]
```

	Age	SibSp
20	35.0	0
21	34.0	0
22	15.0	0

데이터 프레임

행 범위

열 범위

*df.loc[20:23, 5:7]*

# 데이터를 정렬하는 방법

```
>>> titanic.sort_values(by="Age").head()
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
803	804	1	3	...	8.5167	NaN	C
755	756	1	2	...	14.5000	NaN	S
644	645	1	3	...	19.2583	NaN	C
469	470	1	3	...	19.2583	NaN	C
78	79	1	2	...	29.0000	NaN	S

# 데이터를 정렬하는 방법

```
>>> titanic.sort_values(by=['Pclass', 'Age'], ascending=False).head()
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
851	852	0	3	...	7.7750	NaN	S
116	117	0	3	...	7.7500	NaN	Q
280	281	0	3	...	7.7500	NaN	Q
483	484	1	3	...	9.5875	NaN	S
326	327	0	3	...	6.2375	NaN	S

# 여주가

```
>>> countries = pd.read_csv("d:\\countries.csv")  
>>> countries["density"] =  
countries["population"]/countries["area"]  
>>> countries
```

	code	country	area	capital	population	density
0	KR	Korea	98480	Seoul	48422644	491.700284
1	US	USA	9629091	Washington	310232863	32.218292
2	JP	Japan	377835	Tokyo	127288000	336.887795
3	CN	China	9596960	Beijing	1330044000	138.590137
4	RU	Russia	17100000	Moscow	140702000	8.228187

# 행주가

```
>>> df = pd.DataFrame({ "code": ["CA"] , "country": ["Canada"] ,  
        "area": [9984670] , "capital": ["Ottawa"] ,  
"population": [34300000] })  
  
>>> df2 = countries.append(df, ignore_index = True)  
>>> df2  
  
   code country      area      capital  population  
0    KR    Korea     98480       Seoul      48422644  
1    US      USA    9629091  Washington      310232863  
2    JP    Japan     377835      Tokyo      127288000  
3    CN    China    9596960      Beijing     1330044000  
4    RU  Russia   17100000      Moscow      140702000  
5    CA  Canada     9984670      Ottawa      34300000
```

# 행사제

```
>>> countries.drop(index=2, axis=0, inplace = True)

   code country        area      capital  population
0    KR    Korea     98480       Seoul    48422644
1    US      USA    9629091  Washington  310232863
3    CN    China    9596960      Beijing  1330044000
4    RU  Russia   17100000      Moscow  140702000
```

# 여자 예술

```
>>> countries.drop(["capital"], axis=1, inplace = True)

   code country        area  population
0    KR    Korea      98480     48422644
1    US      USA    9629091     310232863
2    JP    Japan     377835    127288000
3    CN    China    9596960   1330044000
4    RU  Russia   17100000    140702000
```

# 타이타닉 승객의 평균 연령은 얼마입니까?

```
>>> titanic["Age"].mean()
```

```
29.69911764705882
```

# 타이타닉 승객 연령과 탑승권 요금의 중간값은 얼마인가?

```
>>> titanic[["Age", "Fare"]].median()
```

```
Age      28.0000
```

```
Fare     14.4542
```

# 카테고리별로 그룹화된 통계

```
>>> titanic[["Sex", "Age"]].groupby("Sex").mean()
```

	Age
Sex	
female	27.915709
male	30.726645

우리의 관심은 각 성별의 평균 연령이므로 titanic[["Sex", "Age"]]에 의하여 이 두 열의 선택이 먼저 이루어진다. 다음으로, groupby() 메소드가 "Sex" 열에 적용되어 "Sex" 값에 따라서 그룹을 만든다. 이어서 각 성별의 평균 연령이 계산되어 반환된다.

# 성별 및 승객 등급 조합의 평균 탑승권 요금은 얼마인가?

```
>>> titanic.groupby(["Sex", "Pclass"])["Fare"].mean()
```

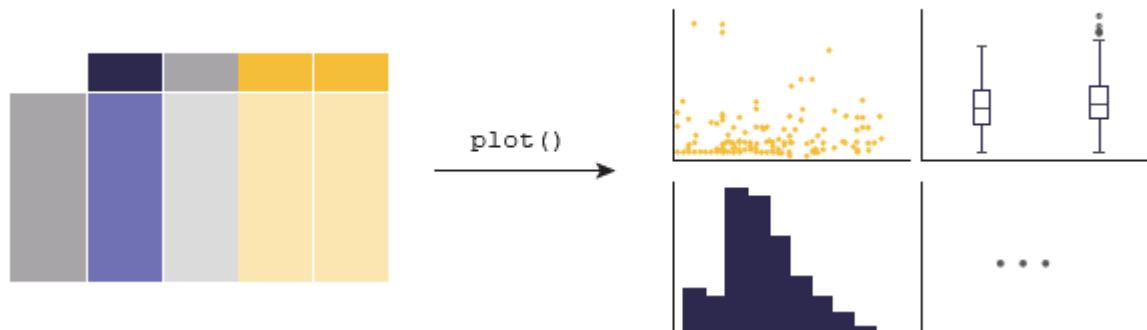
Sex	Pclass	
female	1	106.125798
	2	21.970121
	3	16.118810
male	1	67.226127
	2	19.741782
	3	12.661633

# 각 승객 등급의 수는 몇 명인가?

```
>>> titanic["Pclass"].value_counts()  
3    491  
1    216  
2    184
```

# 데이터로 차트 그리기

- `df.plot()`와 같이 호출하면 인덱스에 대하여 모든 열을 그린다.
- `df.plot(x='col1')`와 같이 호출하면 하나의 열만을 그린다.
- `df.plot(x='col1', y='col2')`와 같이 호출하면 특정 열에 대하여 다른 열을 그리게 된다.

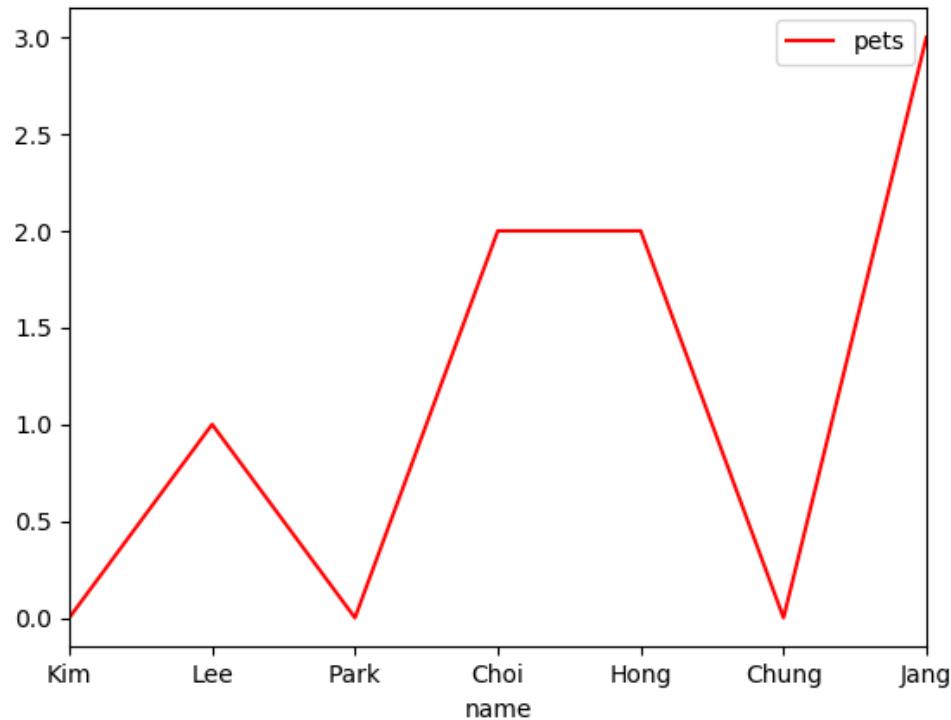


# 데이터 정의

```
import matplotlib.pyplot as plt  
import pandas as pd  
  
df = pd.DataFrame({  
    'name':['Kim','Lee','Park','Choi','Hong','Chung','Jang'],  
    'age':[22,26,78,17,46,32,21],  
    'city':['Seoul','Busan','Seoul','Busan','Seoul','Daejun','Daejun'],  
    'children':[2,3,0,1,3,4,3],  
    'pets':[0,1,0,2,2,0,3]  
})
```

# 그래프

```
df.plot(kind='line', x='name', y='pets', color='red')  
plt.show()
```



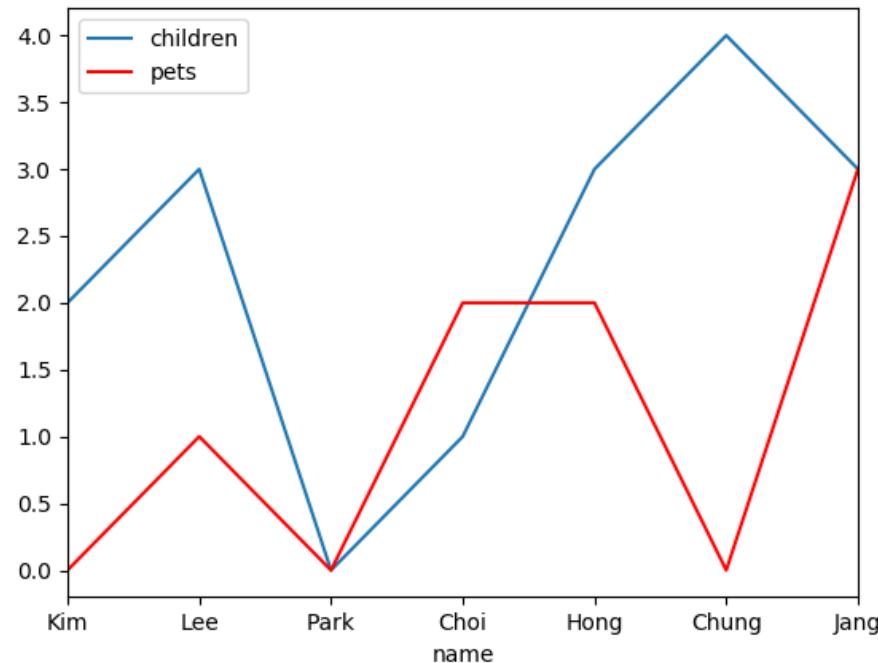
# 중첩 차트 그리기

```
ax = plt.gca()
```

```
df.plot(kind='line',x='name',y='children',ax=ax)
```

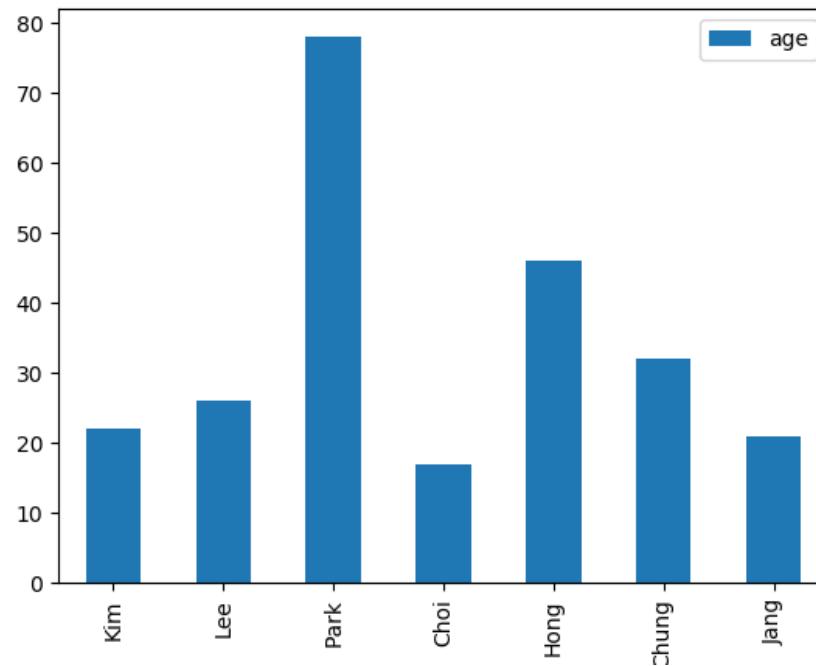
```
df.plot(kind='line',x='name',y='pets', color='red', ax=ax)
```

```
plt.show()
```



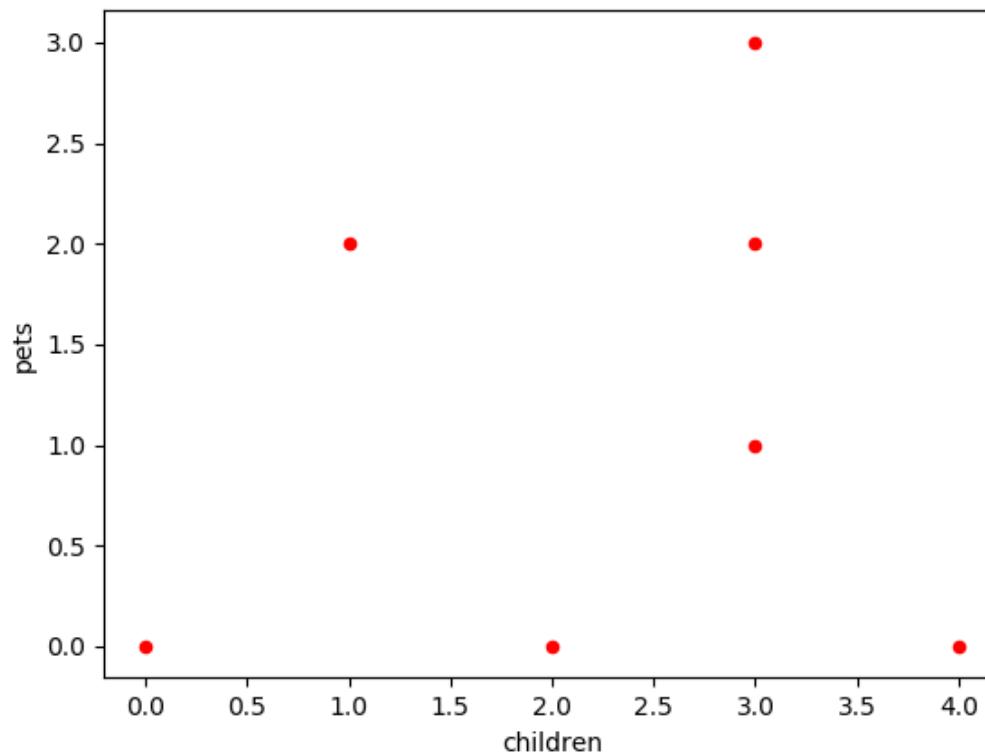
# 막대 그래프 그리기

```
df.plot(kind='bar',x='name',y='age')  
plt.show()
```



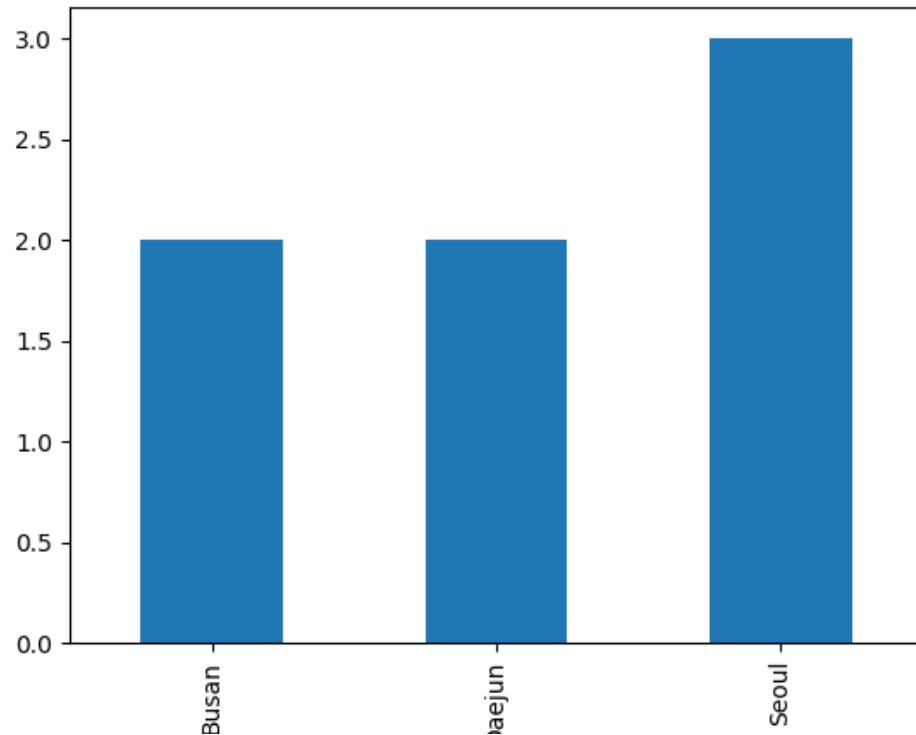
# 산포도 그리기

```
df.plot(kind='scatter',x='children',y='pets',color='red')  
plt.show()
```



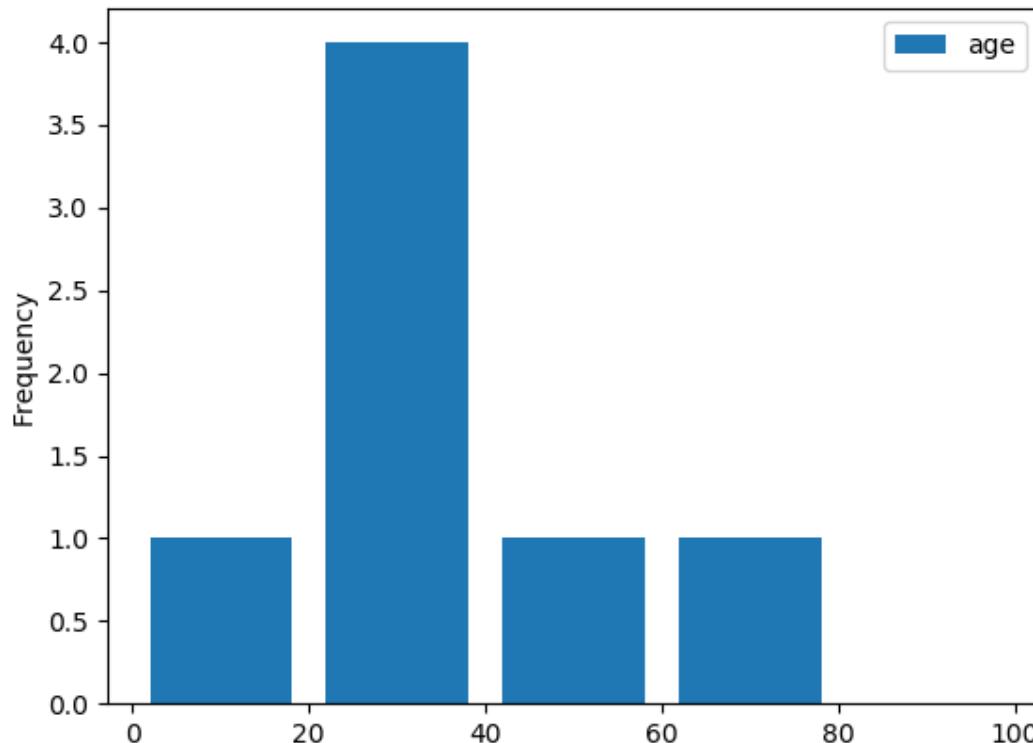
# 그룹핑하여 그리기

```
df.groupby('city')['name'].nunique().plot(kind='bar')  
plt.show()
```



# 히스토그램 그리기

```
df[['age']].plot(kind='hist', bins=[0, 20, 40, 60, 80, 100], rwidth=0.8)  
plt.show()
```



# 피벗 테이블

- 판다스 라이브러리는 값을 깔끔한 2차원 테이블로 요약한 `pivot_table()`이라는 함수를 제공한다.

The diagram illustrates the transformation of a wide DataFrame into a pivot table. On the left, a wide DataFrame shows student scores across three subjects (수학, 과학, 사회) for two students (홍길동, 최자영). On the right, a pivot table shows the same data in a long format, where rows are grouped by student and columns are grouped by subject.

학생	과목	성적	학생	수학	과학	사회
홍길동	수학	100	홍길동	100	95	90
홍길동	과학	95	최자영	90	95	100
홍길동	사회	90				
최자영	수학	90				
최자영	과학	96				
최자영	사회	100				

# 사용 데이터

```
>>> titanic.drop(['PassengerId','Ticket','Name'], inplace=True, axis=1)
```

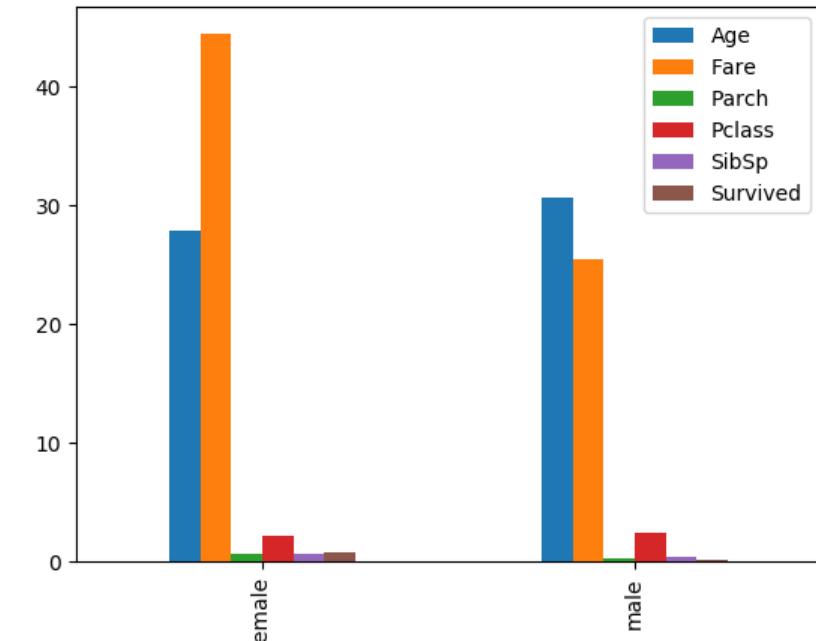
```
>>> titanic.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	NaN	S
1	1	1	female	38.0	1	0	71.2833	C85	C
2	1	3	female	26.0	0	0	7.9250	NaN	S
3	1	1	female	35.0	1	0	53.1000	C123	S
4	0	3	male	35.0	0	0	8.0500	NaN	S

# 피벗 테이블에서 인덱스를 사용하여 데이터를 그려화하자.

```
>>> table = pd.pivot_table(data=titanic, index=['Sex'])  
  
          Age      Fare     Parch   Pclass    SibSp  Survived  
Sex  
female  27.915709  44.479818  0.649682  2.159236  0.694268  0.742038  
male    30.726645  25.523893  0.235702  2.389948  0.429809  0.188908
```

```
>>> table.plot(kind="bar")
```



# 다중 인덱스로 표보하기

```
>>> table = pd.pivot_table(titanic, index=['Sex','Pclass'])  
          Age      Fare     Parch    SibSp  Survived  
Sex   Pclass  
female 1    34.611765  106.125798  0.457447  0.553191  0.968085  
      2    28.722973  21.970121  0.605263  0.486842  0.921053  
      3    21.750000  16.118810  0.798611  0.895833  0.500000  
male   1    41.281386  67.226127  0.278689  0.311475  0.368852  
      2    30.740707  19.741782  0.222222  0.342593  0.157407  
      3    26.507589  12.661633  0.224784  0.498559  0.135447
```

# 특징별로 다른 집계 합수 적용

```
>>> table = pd.pivot_table(titanic ,index=['Sex','Pclass'],  
aggfunc={'Age':np.mean,'Survived':np.sum})
```

		Age	Survived
Sex	Pclass		
female	1	34.611765	91
	2	28.722973	70
	3	21.750000	72
male	1	41.281386	45
	2	30.740707	17
	3	26.507589	47

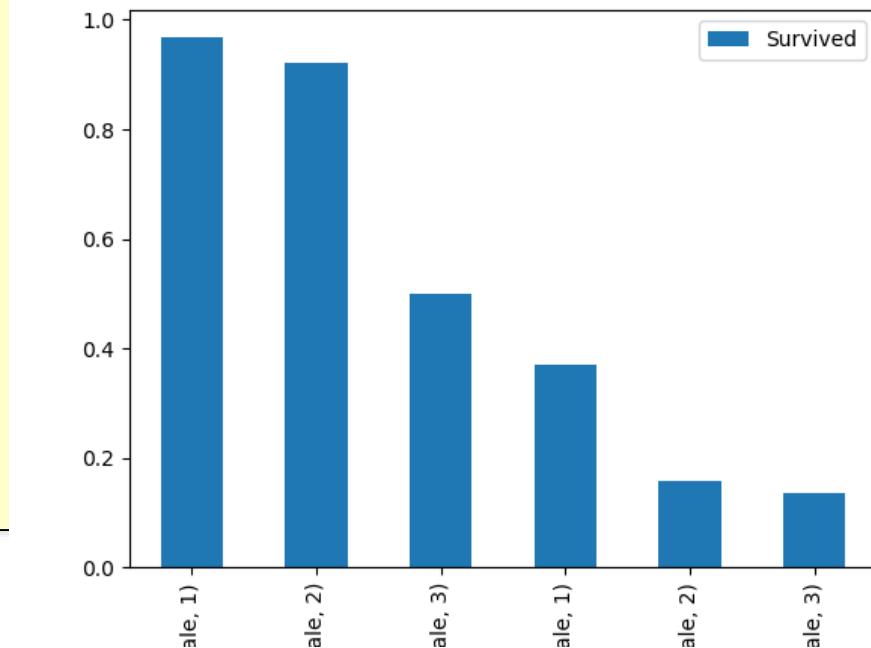
# value 매개 변수로 사용하여 특정한 데이터에 대한 짐 제

```
>>> table=pd.pivot_table(titanic, index=['Sex', 'Pclass'],  
values=['Survived'], aggfunc=np.mean)
```

Survived

Sex	Pclass	Survived
female	1	0.968085
	2	0.921053
	3	0.500000
male	1	0.368852
	2	0.157407
	3	0.135447

```
>>> table.plot(kind='bar')
```



# 데이터 간의 관계 찾기

```
>>> table = pd.pivot_table(titanic ,index=['Sex'],
   columns=['Pclass'],values=['Survived'],aggfunc=np.sum)
```

Survived

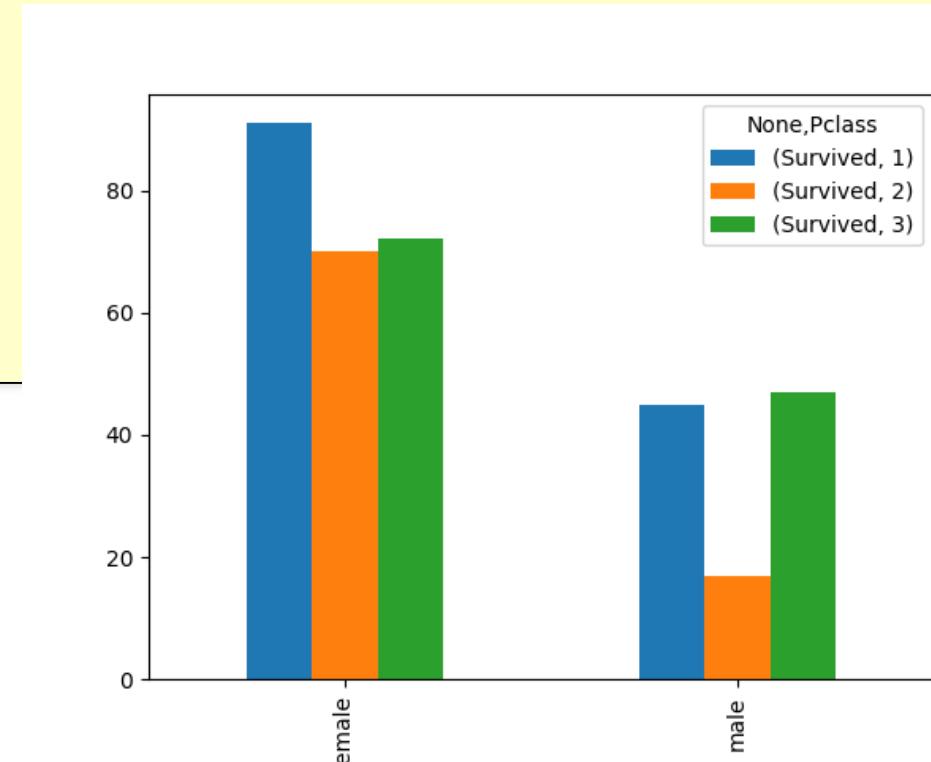
Pclass	1	2	3
--------	---	---	---

Sex

female	91	70	72
--------	----	----	----

male	45	17	47
------	----	----	----

```
>>> table.plot(kind='bar')
```



# 데이터 병합

- `merge()`을 사용하면 공통 열이나 인덱스를 사용하여 데이터를 결합한다.
- `join()`을 사용하면 키 열이나 인덱스를 사용하여 데이터를 결합한다.
- `concat()`을 사용하면 테이블의 행이나 열을 결합한다.

# merge()

- merge()를 사용하면 데이터베이스의 조인(join) 연산을 할 수 있다.

The diagram illustrates the merge operation using two input DataFrames and their resulting merged output.

**Input DataFrames:**

- Left DataFrame:** employee department  
0 Kim Accounting  
1 Lee Engineering  
2 Park HR  
3 Choi Engineering
- Right DataFrame:** employee age  
0 Kim 27  
1 Lee 34  
2 Park 26  
3 Choi 29

**Result:**

	employee	department	age
0	Kim	Accounting	27
1	Lee	Engineering	34
2	Park	HR	26
3	Choi	Engineering	29

# merge()

```
>>> df1 = pd.DataFrame({'employee': ['Kim', 'Lee', 'Park', 'Choi'],
   'department': ['Accounting', 'Engineering', 'HR', 'Engineering']})

>>> df2 = pd.DataFrame({'employee': ['Kim', 'Lee', 'Park', 'Choi'],
   'age': [27, 34, 26, 29]})

>>> df3 = pd.merge(df1, df2)
>>> df3
   employee  department  age
0      Kim  Accounting   27
1      Lee  Engineering  34
2     Park        HR    26
3     Choi  Engineering  29
```

# 결손값 삭제하기

- 실제 데이터셋들은 완벽하지 않다.
- 판다스에서는 결손값을 **NaN**으로 나타낸다. 판다스는 결손값을 탐지하고 수정하는 함수를 제공한다.



데이터 정제는 결손값을  
채우거나 적절하지 않은  
데이터 값을 수정하는  
것입니다.

# 결소값 삭제하기

countries1.csv

```
code,country,area,capital,population
KR,Korea,98480,Seoul,48422644
US,USA,9629091,Washington,310232863
JP,Japan,377835,Tokyo,127288000
CN,China,9596960,Beijing,1330044000
RU,Russia,17100000,Moscow,140702000
IN,India,,New Delhi,1368737513
```

면적 데이터가 누락 되어 있다!

# 파이썬 데이터 작업하기

```
>>> import pandas as pd  
>>> df = pd.read_csv('d:/countries1.csv', index_col=0)  
  
>>> df.dropna(how="all")  
country      area      capital  population  
  
KR  Korea    98480.0     Seoul   48422644  
US  USA    9629091.0  Washington  310232863  
JP  Japan   377835.0     Tokyo   127288000  
CN  China   9596960.0    Beijing  1330044000  
RU  Russia  17100000.0    Moscow   140702000
```

# 결소값 NaN 보정하기

```
>>> df_0 = df.fillna(0)
      country      area      capital  population
KR    Korea  98480.0     Seoul   48422644
US    USA  9629091.0  Washington  310232863
JP   Japan  377835.0     Tokyo  127288000
CN   China  9596960.0    Beijing 1330044000
RU  Russia 17100000.0     Moscow  140702000
IN   India      0.0  New Delhi 1368737513
```

# 결소값 보정하기

```
>>> df.fillna(df.mean()['area'])
```

	country	area	capital	population
KR	Korea	98480.0	Seoul	48422644
US	USA	9629091.0	Washington	310232863
JP	Japan	377835.0	Tokyo	127288000
CN	China	9596960.0	Beijing	1330044000
RU	Russia	17100000.0	Moscow	140702000
IN	India	7360473.2	New Delhi	1368737513

# Q & A

