Pandas 데이터 기초

2

Pandas 모듈 데이터는 series(1개 변수)와 dataFrame 2개 이상의 변수로 나뉜다. 외부 데이터를 pandas 모듈로 읽어 들이면 자동으로 dataFrame으로 된다. pandas 데이터프레임을 행, 열, 그리고 형식 기준으로 다룬다.

Series & DataFrame

개요

pandas 모듈에서 만들어지는 데이터는 Series 시리즈와 Dataframe 데이터프레임으로 나뉜다.

PANDAS 데이터는 행 인덱스, 영 인덱스와 행렬 데이터로 구성되어 있다. list 형식은 한 열(변수) 관측치만 존재한다면 Pandas Series는 행 인덱스와 관측값, DataFrame은 행, 열(변수) 인덱스와 행렬 관측값으로 구성되어 있다.

pandas Series

정규분포(평균=50, 표준편차=10)로부터 크기 20인 데이터를 생성하여 x에 저장 -> x의 형식은 list -> 이를

```
from scipy.stats import norm
x=norm.rvs(loc=50,scale=10,size=20)
```

```
type(x)
```

```
numpy.ndarray
```

```
x[0:3]
```

```
array([46.20941901, 49.09732209, 56.87725475])
```

- 리스트 형식을 pandas Series 형식으로 바꾸는 함수 - pd.Series(리스트명)
- pandas Series 첫 3개 행 관측치 출력 결과를 보면 행 인덱스가 0, 1, 2, ...
- ps_x[0:2] - 행 인덱스 0, 1=(2-1) 2개 관측값 출력하였음

```
import pandas as pd
ps_x=pd.Series(x)
```

```
type(ps_x)
```

```
pandas.core.series.Series
```

```
ps_x.head(3)
```

```
0    46.209419
1    49.097322
2    56.877255
dtype: float64
```

```
ps_x[0:2]
```

```
0    46.209419
1    49.097322
dtype: float64
```

- Series 형식에 가능한 통계량 구하기, 데이터 subset이 가능하다.

```
ps_x.mean() #평균
```

```
52.20257012876957
```

```
ps_x.quantile(0.9) #90% 백분위
```

```
62.107009801945644
```

```
ps_x[ps_x>63]
```

```
5    63.743034
14   70.246668
dtype: float64
```

데이터 프레임 구조

pandas 모듈을 이용하여 데이터를 만들면 dataFrame이다.

[예제]http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv

```
import pandas as pd
filename='http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv'
ds=pd.read_csv(filename, encoding='ms949')
```

```
type(ds)
```

```
pandas.core.frame.DataFrame
```

데이터 크기 & 일부 데이터 보기

데이터프레임 행과 열 개수

```
ds.shape
```

```
(60, 15)
```

- 행의 인덱스는 0부터 시작하므로 첫번째 개체는 0이라는 번호를 갖는다.

DataFrame.head(?) : 첫 번째 행부터 지정 ? 개수만큼 출력

```
ds.head(2)
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education
0	Akron, OH	921.87	27	71	59	36	11.4
1	Albany-Schenectady-Troy, NY	997.87	23	72	57	35	11.0

DataFrame.tail(?) : 마지막 번째 행부터 지정 ? 개수만큼 출력

```
ds.tail(2)
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education
58	York, PA	911.82	33	76	54	62	9.0
59	Youngstown-Warren, OH	954.44	28	72	58	38	10.7

데이터프레임 상세정보 DataFrame.info()

- 데이터 개수 n= 60 entry, 행 인덱싱 번호 0 to 59
- 열 변수 형식 출력 : 정수 int, 실수 float, 문자열 object
- 결측치 개수가 나타남 : income 변수에는 1개 결측치가 있음

```
ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 15 columns):
cit pop/house    60 non-null float64
M   income       59 non-null float64
Ja  HCPot        60 non-null int64
```

```
ds.income[19:22]
```

```
19    32000.0
20        NaN
21    29915.0
```

(21번째)

결측치 개수 출력

```
ds.isnull().sum()
```

```
city          0
Mortality     0
JanTemp       0
JulyTemp      0
pop/house     0
income        1
HCPot         0
```

결측치 행 데이터 삭제 : DataFrame.dropna()

- 행 데이터 중 어느 한 변수에도 결측치가 있는 경우 삭제함
- 향후 제거한 데이터프레임을 사용하려면 다른 이름으로 저장해야 한다.

```
ds.dropna().shape
```

```
(59, 15)
```

결측치 대체(값 대체) : DF.fillna(값, inplace=True)

```
ds.fillna(0,inplace=True)
ds.income
```

```
18    33858.0
19    32000.0
20        0.0
21    29915.0
22    29450.0
```

= inplace=True (바로 변경)

결측치 대체(평균대체) : DF.fillna(값, inplace=True)

```
ds.fillna(ds.mean(),inplace=True)
ds.income
```

```
19    32000.000000
20    33246.661017
21    29915.000000
22    29450.000000
```

DataFrame.clumns : 변수 이름 정보

```
ds.columns   #변수이름 출력 list(ds) 동일
```

```
Index(['city', 'Mortality', 'JanTemp', 'JulyTemp', 'RelHum', 'Rain',
       'Education', 'PopDensity', 'NonWhite', 'WC', 'pop/house', 'income',
       'HCPot', 'NOxPot', 'SO2Pot'],
      dtype='object')
```

열 변수 인덱스(이름) 재설정

전체 변수 이름 재설정 : DataFrame.columns=['새이름1', '새이름2', ...]

- 변수명을 차례로 재설정한다. 변수가 많은 경우 적절하지 않음

```
ds0=ds.iloc[:, 0:2]   #최초 2개 변수만 지정 (예제)
ds0.columns
```

```
Index(['city', 'Mortality'], dtype='object')
```

```
ds0.columns=['도시명','사망률지수'] #수정이름 차례로 부여하면 된다.
ds0.columns
```

```
Index(['도시명', '사망률지수'], dtype='object')
```

DataFrame.rename(columns={변경}, inplace=True)

- 원하는 변수이름만 수정한다. (강추)

```
ds.rename(columns={'city':'도시명'},inplace=True)
ds.columns
```

```
Index(['도시명', 'Mortality', 'JanTemp', 'JulyTemp', 'RelHum', 'Rain',
       'Education', 'PopDensity', 'NonWhite', 'WC', 'pop/house', 'income',
       'HCPot', 'NOxPot', 'SO2Pot'],
      dtype='object')
```

행 인덱스 재설정

행 인덱스는 0, 1, 2, ... 가 디폴트이다.

```
ds.head(2)
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education
0	Akron, OH	921.87	27	71	59	36	11.4
1	Albany-Schenectady-Troy, NY	997.87	23	72	57	35	11.0

주이름 - 행 인덱스 만들기

- str.split() ‘ ’을 n=1회 사용하여 expand=True 2개 문자열로 나누시오.
- [0] 첫 열 인덱스는 도시이름, [1]에는 주이름이 들어가 있음

```
ds['city'].str.split(' ', n=1, expand=True).head(3)
```

	0	1
0	Akron,	OH
1	Albany-Schenectady-Troy,	NY
2	Allentown,	Bethlehem, PA-NJ

- str.strip() - ‘,’ 콤마를 제외하였음
- str.rjust(30,”*) - 총 30 문자열 크기로 만들고 공백은 *로 채우고 우측 정렬 right adjust
- str.slice(28,31) - 30개 크기로 만들어진 문자열에서 28번째-사실은 29번째(0부터 시작하므로)에서 30번째 문자열을 2개를 가져온다. (주이름)

```
ds['도시명']=ds['city'].str.split(' ', n=1, expand=True)[0]
```

```
ds['주이름']=ds['city'].str.split(' ', n=1, expand=True)[1]
```

```
ds['도시명']=ds['도시명'].str.strip(',')
ds['주이름']=ds['주이름'].str.rjust(30,'*')
ds['주이름']=ds['주이름'].str.slice(28,31)
```

```
ds.head(3)
```

	city	Mortality	JanTemp
0	Akron, OH	921.87	27
1	Albany-Schenectady-Troy, NY	997.87	23
2	Allentown, Bethlehem, PA-NJ	962.35	29

(중략)

도시명	주이름
Akron	OH
Albany-chenectady-Troy	NY
Allentown	NJ

DataFrame.set_index('인덱스이름')

```
ds.set_index('주이름').head(3)
```

주이름	city	Mortality	JanTemp	JulyTemp	Rel
OH	Akron, OH	921.87	27	71	
NY	Albany-Schenectady-Troy, NY	997.87	23	72	
NJ	Allentown, Bethlehem, PA-NJ	962.35	29	74	

멀티인덱스(행) multi_index : DF.set_index(['이름1','이름2'])

```
ds.set_index(['주이름', '도시명']).head(3)
```

주이름	도시명	city	Mortality	JanTemp	JulyTemp	RelHum
OH	**Akron**	Akron, OH	921.87	27	71	59
NY	**Albany-Schenectady-Troy**	Albany-Schenectady-Troy, NY	997.87	23	72	57

- 첫 행 인덱스 사용하면 - "OH" ohio 주 도시 출력

```
ds.loc['OH']
```

도시명	city	Mortality	JanTemp	JulyTemp	RelHum	Rain
Akron	Akron, OH	921.87	27	71	59	36
Canton	Canton, OH	912.35	27	72	59	36
Cleveland	Cleveland,	985.95	28	71	60	35

- 첫 행, 두번째 인덱스 동시 사용

```
ds.loc['OH','Canton']
```

주이름	도시명	city	Mortality	JanTemp	JulyTemp	RelHum	Rain
OH	**Canton**	Canton, OH	912.35	27	72	59	36

멀티 인덱스(열)

[예제]http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv

```
import pandas as pd
url='http://203.247.53.31/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv'
ds=pd.read_csv(url)
```

```
ds.columns
```

```
Index(['city', 'Mortality', 'JanTemp', 'JulyTemp', 'RelHum', 'Rain',
       'Education', 'PopDensity', 'NonWhite', 'WC', 'pop/house', 'income',
       'HCPot', 'NOxPot', 'SO2Pot', '', '교육수준', '사망지수그룹'],
      dtype='object')
```

준비 : 새로운 변수 만들기 방법 1) df.loc[조건, '새변수이름']=변수값

- Education 교육수준 12(고졸 이상) 이상이면 고학력 도시, 12 미만이면 저학력 도시로 나눈 변수를 '교육수준' 이름으로 저장함
- 60개 도시가 2개 집단으로 분류되었음 - 이 형식은 집단이 3개 이상인 경우 사용하면 된다.

```
ds.loc[ds['Education']>=12,'교육수준']='고학력'
ds.loc[ds['Education']<12,'교육수준']='저학력'
```

```
ds['교육수준'].value_counts()
```

```
저학력    48
고학력    12
Name: 교육수준, dtype: int64
```

새 변수 방법 2) numpy 사용 : np.where(조건, 만족시, 만족하지 않을 시)

- ds.Mortality.mean() : 사망율(Mortality) 평균 계산
- 사망율 평균보다 높은 도시는 지수높음, 낮은 도시는 지수낮음

```
import numpy as np
ds['사망지수그룹']=np.where(ds['Mortality']>ds.Mortality.mean(),
                    '지수높음','지수낮음')
```

```
ds['사망지수그룹'].value_counts()
```

```
지수높음    31
지수낮음    29
Name: 사망지수그룹, dtype: int64
```

멀티 인덱스 만들기

DF.groupby(['그룹변수1','그룹변수2'])[['통계량변수1','변수2']].통계량()

- 통계량 = mean, var, std, kurtosis, skewness, quantile[0.9] 등 가능
- 만약 요약 통계량 변수 지정하지 않으면 데이터 내 숫자형 변수 데이터는 모두 구함

교육수준	사망지수그룹	NonWhite	PopDensity
고학력	지수낮음	6.363636	3656.909091
	지수높음	25.900000	5308.000000
저학력	지수낮음	6.761111	3442.833333
	지수높음	16.486667	4168.600000

열 인덱스, 행인덱스 두 번 동시 사용

```
ds_mean=ds.groupby(['교육수준','사망지수그룹'])[['NonWhite','PopDensity']].mean()
```

```
ds_mean.loc[['고학력'],['NonWhite']]
```

		NonWhite
교육수준	사망지수그룹	
고학력	지수낮음	6.363636
	지수높음	25.900000

```
ds_mean.iloc[:,[0,1]]
```

		NonWhite	PopDensity
교육수준	사망지수그룹		
고학력	지수낮음	6.363636	3656.909091
	지수높음	25.900000	5308.000000
저학력	지수낮음	6.761111	3442.833333
	지수높음	16.486667	4168.600000

개체 행(row)

예제 데이터 (데이터 크기 n=60)

http://wolfpack.hnu.ac.kr/Stat_Notes/example_data/US_crime.csv

```
import pandas as pd
url='http://wolfpack.hnu.ac.kr/Stat_Notes/example_data/US_crime.csv '
crime=pd.read_csv(url,encoding='ms949')
crime.columns
```

```
Index(['주이름', '도시이름', '인구수', '폭력사건', '살인', '강간', '강도', '폭행', '재산범죄', '주거침입',
       '절도', '차량절도'],
```

```
crime.shape
```

```
(9292, 12)
```

=> 변수(열 개수) 12개, 행 개수(데이터 크기 n=9,292)

행 인덱스(이름) df.index.names

Pandas 데이터를 만들면 행 인덱스는 공백이다.

```
crime.index.names
```

```
FrozenList([None])
```

```
crime.head(3)
```

	주이름	도시이름	인구수
0	ALABAMA	Abbeville	2645.0
1	ALABAMA	Adamsville	4481.0
2	ALABAMA	Addison	744.0

Pandas 행 데이터 불러오기

행 데이터 지정 df[시작행, 끝행+1]

- 첫 행 (0) 과 두번 째 행(2-1=1) 지정함

```
crime[0:2]
```

	주이름	도시이름	인구수	폭력사건	살인	강간
0	ALABAMA	Abbeville	2645.0	11.0	1	1.0
1	ALABAMA	Adamsville	4481.0	19.0	1	0.0

행 데이터 삭제하기 df.drop[0,2]

- 인덱스 0, 2 제외함

```
crime.drop([0,2]).head(2)
```

	주이름	도시이름	인구수	폭력사건	살인	강간
1	ALABAMA	Adamsville	4481.0	19.0	1	0.0
3	ALABAMA	Alabaster	31170.0	44.0	0	2.0

향후 삭제된 데이터를 사용하려면 저장해야 한다.

```
crime0=crime.drop([0,2])
crime0.shape
```

```
(9290, 12)
```

=> 행 2개가 제외되어 n=9,292-> 9,290

행 인덱싱 열 변수이름으로 대체하기

DataFrame.set_index('열변수이름')

- 0~59 숫자인 디폴트 인덱싱이 열 변수 city로 재 설정되었다.

```
crime.set_index('주이름').head(2)
```

주이름	도시이름	인구수	폭력사건	살인	강간	강도	폭행
ALABAMA	Abbeville	2645.0	11.0	1	1.0	2	7.0
ALABAMA	Adamsville	4481.0	19.0	1	0.0	7	11.0

DataFrame.set_index(['열변수1','열변수2'], inplace=True)

- 열변수를 행 인덱스로 대체한 후 동일 이름으로 저장하여 사용하려면 inplace=True를 사용하면 된다. 다른 이름으로 사용하려면 이름을 다르게 하여 저장하면 된다.

```
crime.set_index(['주이름','도시이름'],inplace=True)
crime.index.names
```

```
FrozenList(['주이름', '도시이름'])
```

행 멀티 인덱싱 사용하기

- 원데이터 행 인덱스는 0, 1, 2 숫자로 이름은 없었으나 (None)
- df.set_index(['이름1', '이름2']) - 행 인덱스를 이름1, 이름2 순으로 지정함
- crime00 데이터프레임의 행 인덱스는 '주이름', '도시이름'이다.

```
import pandas as pd
url='http://wolfpack.hnu.ac.kr/Stat_Notes/example_data/US_crime.csv '
crime=pd.read_csv(url,encoding='ms949')
crime00=crime.set_index(['주이름','도시이름'])
crime00.index.names
```

```
FrozenList(['주이름', '도시이름'])
```

첫 행 인덱스 지정

df.loc['관심범주'] - df.loc[['관심범주1', '범주2']]

```
crime00.loc['ALABAMA']
```

도시이름	인구수	폭력사건	살인	강간	강도	폭행
Abbeville	2645.0	11.0	1	1.0	2	7.0
Adamsville	4481.0	19.0	1	0.0	7	11.0
Addison	744.0	1.0	0	1.0	0	0.0

```
crime00.loc[['ALABAMA', 'OHIO']].head(2)
```

주이름	도시이름	인구수	폭력사건	살인	강간	강도	폭행	재산범죄
ALABAMA	**Abbeville**	2645.0	11.0	1	1.0	2	7.0	63.0
	Adamsville	4481.0	19.0	1	0.0	7	11.0	321.0

```
crime00.loc[['ALABAMA', 'OHIO']].tail(2)
```

두번째 행 지정 df.xs('범주', level='인덱스이름')

```
crime00.xs('Abbeville', level='도시이름')
```

주이름	인구수	폭력사건	살인	강간	강도	폭행	재산범죄
ALABAMA	2645.0	11.0	1	1.0	2	7.0	63.0
GEORGIA	2888.0	3.0	0	NaN	2	1.0	22.0

첫번째 행 인덱스에도 사용가능

```
crime00.xs('ALABAMA', level='주이름')
```

도시이름	인구수	폭력사건	살인	강간	강도	폭행	재산범죄
Abbeville	2645.0	11.0	1	1.0	2	7.0	63.0
Adamsville	4481.0	19.0	1	0.0	7	11.0	321.0
Addison	744.0	1.0	0	1.0	0	0.0	25.0

두번째 행 지정. df.query("행인덱스명=='범주' ")

```
crime00.query("도시이름=='Abbeville'")
```

주이름	도시이름	인구수	폭력사건	살인	강간	강도	폭행
ALABAMA	**Abbeville**	2645.0	11.0	1	1.0	2	7.0
GEORGIA	**Abbeville**	2888.0	3.0	0	NaN	2	1.0

```
crime00.query("주이름=='ALABAMA'")
```

주이름	도시이름	인구수	폭력사건	살인	강간	강도
ALABAMA	**Abbeville**	2645.0	11.0	1	1.0	2

멀티 인덱스 동시 사용하기 df.loc[('첫행범주', '2번째범주')]

```
crime00.loc[('ALABAMA', 'Abbeville')]
```

```
인구수     2645.0
폭력사건     11.0
살인          1.0
강간          1.0
강도          2.0
폭행          7.0
```

행 인덱스 이름 바꾸기 df.index.rename_axis()

```
crime00.rename_axis(index=['State_name', 'City_name'],inplace=True)
crime00.index.names
```

```
FrozenList(['State_name', 'City_name'])
```

변수(열 column)

변수(열 인덱스) 이름

[예제]http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv

```
import pandas as pd
filename='http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv'
ds=pd.read_csv(filename, encoding='ms949')
```

```
ds.columns   #변수이름 출력 list(ds) 동일
```

```
Index(['city', 'Mortality', 'JanTemp', 'JulyTemp', 'RelHum', 'Rain',
       'Education', 'PopDensity', 'NonWhite', 'WC', 'pop/house', 'income',
       'HCPot', 'NOxPot', 'SO2Pot'],
      dtype='object')
```

[열 변수 지정] 데이터이름['변수명'] I 데이터이름[['변수명1', '변수명2']]

```
ds['city']
```

```
0                Akron, OH
1    Albany-Schenectady-Troy, NY
2    Allentown, Bethlehem, PA-NJ
3                Atlanta, GA
```

```
ds[['city', 'Rain']]
```

	city	Rain
0	Akron, OH	36
1	Albany-Schenectady-Troy, NY	35
2	Allentown, Bethlehem, PA-NJ	44

열변수 행 인덱스 만들기

df.set_index(['열변수명1','열변수명2'], inplace=True)

외부 데이터를 읽어 들이면 행 인덱스는 0, 1, 2, ...가 된다.

```
ds.index.names
```

```
FrozenList([None])
```

```
ds.head(3)
```

	city	Mortality
0	Akron, OH	921.87
1	Albany-Schenectady-	997.87

열변수 이름 'city', 'Education'가 이제는 행 인덱스가 된다.

```
ds0=ds.set_index(['city','Education'])
ds0.index.names
```

```
FrozenList(['city', 'Education'])
```

새롭게 만들어진 데이터를 향후 동일 이름으로 사용하려면 inplace=True 사용

```
ds.set_index(['city','Education'],inplace=True)
```

변수 삭제

DataFrame.drop('변수명', axis=1)

```
ds.drop('city',axis=1)
```

	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education	PopDensity
0	921.87	27	71	59	36	11.4	3243
1	997.87	23	72	57	35	11.0	4281
2	962.35	29	74	54	44	9.8	4260
3	982.29	45	79	56	47	11.1	3125

DataFrame.drop(['변수명1', '변수명2'], axis=1)

```
ds.drop(['city','Rain'], axis=1)  # 2개 변수 지정
```

	Mortality	JanTemp	JulyTemp	RelHum	Education	PopDensity	N
0	921.87	27	71	59	11.4	3243	
1	997.87	23	72	57	11.0	4281	
2	962.35	29	74	54	9.8	4260	

새로운 변수 만들기

DataFrame['NEW변수명']= {DataFrame['기존변수명']} 계산식

- JanTemp, 1월 기온(화씨)를 섭씨 기온으로 바꾸어 데이터 마지막 열에 변수명 JTC로 저장된다.

```
ds['JTc']=(ds['JanTemp']-32)*9/5
```

‹Pot	S02Pot	JTc
15	59	-9.0
10	39	-16.2

```
ds.columns
```

```
Index(['city', 'Mortality', 'JanTemp', 'JulyTemp', 'RelHum', 'Rain',
       'Education', 'PopDensity', 'NonWhite', 'WC', 'pop/house', 'income',
       'HCPot', 'NOxPot', 'S02Pot', 'JTc'],
      dtype='object')
```

만약 원 변수를 사용할 계획이 없다면 New변수 이름을 기존 변수명과 동일하게 사용하면 원 값 대신 계산된 값이 저장된다.

```
ds['JanTemp']=(ds['JanTemp']-32)*9/5
```

```
ds.columns
```

```
Index(['city', 'Mortality', 'JanTemp', 'JulyTemp', 'RelHum', 'Rain',
       'Education', 'PopDensity', 'NonWhite', 'WC', 'pop/house', 'income',
       'HCPot', 'NOxPot', 'S02Pot'],
      dtype='object')
```

열 변수 조건 활용 행 데이터 subset

DataFrame[조건문], 조건문(예) DataFrame['변수명'] == 값

- 행 인덱스 번호를 보면 0이 없다. 첫번째 관측치가 조건을 만족하지 않아 삭제되었음

```
ds[ds['Mortality']>=950]  #Mortality 값 950
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education	PopDe
1	Albany-Schenectady-Troy, NY	997.87	23	72	57	35	11.0	
2	Allentown, Bethlehem, PA-NJ	962.35	29	74	54	44	9.8	
3	Atlanta, GA	982.29	45	79	56	47	11.1	

조건이 2개 이상인 경우에는 ()로 묶는다. | = or, & = and 의미

- 인덱스 번호가 1, 8, 51이다. 0, 2, ... 등은 조건을 만족하지 않아 삭제 되었음

```
ds[(ds['Mortality']>=950) & (ds['JanTemp']<=25)]  #Mortality 값 900 이상, JanTemp 250이하
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education	PopDensity
1	Albany-Schenectady-Troy, NY	997.87	23	72	57	35	11.0	4281
8	Buffalo, NY	1001.90	24	70	61	36	10.5	6582
51	Syracuse, NY	950.67	24	72	61	38	11.4	4923

변수 행(관측값) 대체

DataFrame.str.split()함수를 이용하여 city이름을 ' ' - 공백에 의해, n=1 한 번 사용, 열을 2개 만들었음

```
ds['city'].str.split(' ',n=1, expand=True)
```

	0	1
0	Akron,	OH
1	Albany-Schenectady-Troy,	NY

결과는 2개 열이고 두 번째 주이름의 인덱싱은 [1]이다.

```
ds['city'].str.split(' ',n=1, expand=True)[1]
```

```
0              OH
1              NY
2    Bethlehem, PA-NJ
3              GA
```

OH 문자열을 .replace()에 의해 'Ohio' 로 수정하였음

```
ds['city'].str.split(' ',n=1, expand=True)[1].replace('OH', 'Ohio')
```

```
0              Ohio
1              NY
2    Bethlehem, PA-NJ
3              GA
```

iloc, loc 활용

[예제]http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv

```
import pandas as pd
filename='http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv'
ds=pd.read_csv(filename, encoding='ms949')
```

```
ds.columns   #변수이름 출력 list(ds) 동일
```

```
Index(['city', 'Mortality', 'JanTemp', 'JulyTemp', 'RelHum', 'Rain',
       'Education', 'PopDensity', 'NonWhite', 'WC', 'pop/house', 'income',
       'HCPot', 'NOxPot', 'SO2Pot'],
      dtype='object')
```

참고 : DataFrame 행 인덱스는 0부터 시작하는 숫자, 이를 변수(columns) 이름으로 대체하고자 한다면,

```
ds.set_index('city', inplace=True)
ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60 entries, Akron, OH to Youngstown-Warren, OH
Data columns (total 14 columns):
Mortality     60 non-null float64
JanTemp       60 non-null int64
```

DataFrame.reset_index(inplace=True) : 현재 행 인덱스를 열 변수로 대체

DataFrame.iloc

행 번호, 열 번호로 위치를 포지션닝 하여 선택한다.

.iloc selections - position based selection

data.iloc[<row selection], <column selection>]

Integer list of rows: [0,1,2]	Integer list of columns: [0,1,2]
Slice of rows: [4:7]	Slice of columns: [4:7]
Single values: 1	Single column selections: 1

```
ds.iloc[0:3] # 첫 행(0)부터 2번째(=3-1) 행 포지션닝
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education
0	Akron, OH	921.87	27	71	59	36	11.4
1	Albany-Schenectady-Troy, NY	997.87	23	72	57	35	11.0
2	Allentown, Bethlehem, PA-NJ	962.35	29	74	54	44	9.8

```
ds.iloc[3] # 4번째(00이 첫 번째 포지션) 행 포지션닝
```

```
city          Atlanta, GA
Mortality          982.29
JanTemp                45
JulyTemp               79
RelHum                 56
Rain                   47
Education            11.1
PopDensity           3125
NonWhite             27.1
WC                   50.2
```

```
ds.iloc[-1] # 마지막 행 포지션닝
```

```
city          Youngstown-Warren, OH
Mortality                    954.44
JanTemp                          28
JulyTemp                         72
RelHum                           58
Rain                             38
Education                      10.7
```

In [79]:
```
ds.iloc[:,0]   #첫번째 열 포지션닝
```

Out[79]:
```
0                    Akron, OH
1     Albany-Schenectady-Troy, NY
2     Allentown, Bethlehem, PA-NJ
3                    Atlanta, GA
4                  Baltimore, MD
5                Birmingham, AL
6                    Boston, MA
7          Bridgeport-Milford, CT
8                    Buffalo, NY
9                    Canton, OH
10          Chattanooga, TN-GA
```

```
ds.iloc[:,0:4]   #첫번째 열에서 3(=4-1)번째 열 포지션닝
```

	city	Mortality	JanTemp	JulyTemp
0	Akron, OH	921.87	27	71
1	Albany-Schenectady-Troy, NY	997.87	23	72
2	Allentown, Bethlehem, PA-NJ	962.35	29	74
3	Atlanta, GA	982.29	45	79
4	Baltimore, MD	1071.29	35	77

```
ds.iloc[:,[0,3,5]]   #첫번째, 4번, 5번째 열 포지션닝
```

	city	JulyTemp	Rain
0	Akron, OH	71	36
1	Albany-Schenectady-Troy, NY	72	35
2	Allentown, Bethlehem, PA-NJ	74	44
3	Atlanta, GA	79	47

```
ds.iloc[0:3,[0,3,5]]   #첫번째, 4번, 5번째 열 포지션닝 + (1~3)개 행 선택
```

	city	JulyTemp	Rain
0	Akron, OH	71	36
1	Albany-Schenectady-Troy, NY	72	35
2	Allentown, Bethlehem, PA-NJ	74	44

```
ds.iloc[[0,2],[0,3,5]]   #첫번째, 4번, 5번째 열 포지션닝 + (1, 3)개 행 선택
```

	city	JulyTemp	Rain
0	Akron, OH	71	36
2	Allentown, Bethlehem, PA-NJ	74	44

DataFrame.loc

행 이름, 열 이름으로 위치를 포지션닝 하여 선택한다.

loc selections - position based selection

data.loc[<row selection], <column selection>]

Index/Label value: 'john' — *Named column: 'first_name'*
List of labels: ['john', 'sarah'] — *List of column names: ['first_name', 'age']*
Logical/Boolean index: data['age'] == 10 — *Slice of columns: 'first_name':'address'*

```
ds.loc[ds['JulyTemp']==70]   #July temp 값 = 70 행
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education
8	Buffalo, NY	1001.90	24	70	61	36	10.5
46	San Diego, CA	839.71	55	70	61	10	12.1
57	Worcester, MA	895.70	24	70	56	65	11.1

```
ds.loc[(ds['JulyTemp'] ==70) | (ds['JulyTemp'] ==71)] #July temp 값 = 70 or 71 행
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education	PopD
0	Akron, OH	921.87	27	71	59	36	11.4	
8	Buffalo, NY	1001.90	24	70	61	36	10.5	
13	Cleveland,	985.95	28	71	60	35	11.1	

```
ds.loc[(ds['JulyTemp'] >=70) & (ds['JulyTemp'] <=72)] #July temp 값 = 70이상, 72 이하 행
```

city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education	PopDensity	N
Akron, OH	921.87	27	71	59	36	11.4	3243	
Albany-Schenectady-Troy, NY	997.87	23	72	57	35	11.0	4281	
Buffalo, NY	1001.90	24	70	61	36	10.5	6582	
Canton, OH	912.35	27	72	59	36	10.7	4213	

행 조건에 맞는 관심 열(변수) 만 출력

```
ds.loc[ds['JulyTemp'] ==70, 'city'] #July temp 값 = 70 city 만 지정
```

```
8      Buffalo, NY
46    San Diego, CA
57    Worcester, MA
Name: city, dtype: object
```

```
ds.loc[ds['JulyTemp'] ==70, ['city','Rain']] #July temp 값 = 70 (city, Rain) 만 지정
```

	city	Rain
8	Buffalo, NY	36
46	San Diego, CA	10
57	Worcester, MA	65

변수명, 인덱스 숫자를 동시에 사용하는 경우 DataFrame.ix 사용하면 된다.

문자열 sring

특수문자 있는 문자열 분리
ColName.str.split(pat='??', n=?, expand=True)

- pat : 특수문자 지정, pat 생략 가능 \n, \t , - , (콤마 ,), 모든 분리 문자 가능
- ? : 분리 지정 문자 사용 회수
- expand = True : 분리된 문자를 분리하여 서로 다른 열에 저장한다.

'http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv' (예제 데이터)

```
import pandas as pd
filename='http://wolfpack.hnu.ac.kr/Stat_Notes/adv_stat/LinearModel/data/SMSA.csv'
ds=pd.read_csv(filename, encoding='ms949')
```

```
ds.head(2)
```

	city	Mortality	JanTemp	JulyTemp	RelHum	Rain	Education
0	Akron, OH	921.87	27	71	59	36	11.4
1	Albany-Schenectady-Troy, NY	997.87	23	72	57	35	11.0

열 city 문자열 내용이 분리 지정 문자 콤마(,)에 의해 n=1에 의해 한 번 분리되어 2개 열로 분리된다.

```
ds['city'].str.split(',', n = 1, expand = True)
```

	0	1
0	Akron	OH
1	Albany-Schenectady-Troy	NY
2	Allentown	Bethlehem, PA-NJ
3	Atlanta	GA

분리 데이터 저장 후 열 이름 지정

1개의 문자열 city가 2개의 열로 분리되어 이름은 0, 1로 저장된다.

```
new=ds['city'].str.split(',', n = 1, expand = True)
new[1]
```

```
0              OH
1              NY
2    Bethlehem, PA-NJ
3              GA
4              MD
5              AL
6              MA
```

열이름 지정 (형식) **DataFrame.columns=[‘대체이름1’,‘대체이름2’, ...]**

```
new=ds['city'].str.split(',', n = 1, expand = True)
new.columns=['도시','주']
new.head(3)
```

	도시	주
0	Akron	OH
1	Albany-Schenectady-Troy	NY
2	Allentown	Bethlehem, PA-NJ

변수이름 변경 (형식) DataFrame.rename(columns={변경}, inplace=True)

```
new.rename(columns={'주':'주이름'}, inplace=True)
new.head(3)
```

	도시	주이름
0	Akron	OH
1	Albany-Schenectady-Troy	NY
2	Allentown	Bethlehem, PA-NJ

공백없는 문자열 분리 DataFrame.str.strip() 이용

분리된 열 변수, ‘주이름’에는 OH가 보이나 실제는 공백이 있다.

```
new['주이름']=='OH'
```

```
0    False
1    False
2    False
3    False
4    False
```

- 공백을 str.strip()로 양쪽 모두 공백을 제거한다.

```
new['주이름'].str.strip() == 'OH'
```

```
0     True
1    False
2    False
3    False
4    False
```

```
new[new['주이름'].str.strip() == 'OH']
```

	도시	주이름
0	Akron	OH
9	Canton	OH
13	Cleveland	OH
14	Columbus	OH
16	Dayton-Springfield	OH
52	Toledo	OH
59	Youngstown-Warren	OH

도시 이름 왼쪽부터 위치 인덱싱 하여 필요한 부분 선택

DataFrame.str.slice(start=?, stop=??).astype(???)

- ? : 시작 위치 인덱싱 번호 (0이 시작)
- ?? : 끝나는 위치 (인덱싱 번호 -1)
- ??? : 불러오기 인덱싱, str, int

```
new['도시'].str.slice(start=0, stop=5).astype(str)
```

```
0    Akron
1    Alban
2    Allen
3    Atlan
```

```
new['도시'].str.slice(start=2, stop=5).astype(str)
```

```
0     ron
1    ban
2     len
3     lan
4     lti
```

날짜 date

예제 데이터

기상청 데이터 (2018년)
http://203.247.53.31/Stat_Notes/example_data/climate/2018climate.csv

A	B	C	D	E	F	G
지점	일시	평균기온(°C)	최저기온(°C)	최저기온 시각	최고기온(°C)	최고기온 시각
90	2018.1.1	1	-3.2	343	4.2	1443
90	2018.1.2	1.5	-2.1	2344	5.6	1358
90	2018.1.3	-1.6	-5.5	732	3.3	1432
90	2018.1.4	-1	-5.7	342	2.2	1449

- 일시는 문자열(string)으로 저장되었음. - (object)

```
import pandas as pd
url='http://203.247.53.31/Stat_Notes/example_data/%EA%B8%B0%
ds=pd.read_csv(url, encoding='ms949')
```

```
ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34675 entries, 0 to 34674
Data columns (total 61 columns):
지점                  34675 non-null int64
일시                  34675 non-null object
평균기온(°C)          34630 non-null float64
최저기온(°C)          34672 non-null float64
최저기온 시각(hhmi)    34672 non-null float64
```

미세먼지 데이터 (2018년 1분기)

http://203.247.53.31/Stat_Notes/example_data/mise/2018Q1.csv

A	B	C	D	E	F
지역	측정소코드	측정소명	측정일시	SO2	CO
서울 중구	111121	중구	2018010101	0.004	0.5
서울 중구	111121	중구	2018010102	0.004	0.4
서울 중구	111121	중구	2018010103	0.004	0.4

- 측정일시 - 숫자형(int)으로 저장되어 있음

```
ds2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768672 entries, 0 to 768671
Data columns (total 11 columns):
지역        768672 non-null object
측정소코드    768672 non-null int64
측정소명      768672 non-null object
측정일시      768672 non-null int64
SO2         720520 non-null float64
```

문자열 string 형식 날짜 형식으로 변환하기

pd.to_datetime(PD.Series)

```
import datetime
ds.일시= pd.to_datetime(ds.일시)
ds.일시.head(3)
```

```
0   2018-01-01
1   2018-01-02
2   2018-01-03
Name: 일시, dtype: datetime64[ns]
```

숫자 int 형식 날짜 형식으로 변환하기

df['숫자형변수'].astype(str)

숫자 형식은 문자형식으로 변환해야 한다. astype(str)으로 문자열 형식으로 변환하고 데이터 길이 날짜 형식에 8자리에 적합하도록 str.slice() 사용하여 첫 8자리만 지정한다. (문자열 데이터를 숫자로 바꾸는 경우 .astype(int))

```
ds2['측정일시']=ds2['측정일시'].astype(str)
ds2['측정일시']=ds2['측정일시'].str.slice(0,8)
ds2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768672 entries, 0 to 768671
Data columns (total 11 columns):
지역       768672 non-null object
측정소코드    768672 non-null int64
측정소명     768672 non-null object
측정일시     768672 non-null object
SO2      720520 non-null float64
```

pd.to_datetime()을 사용하여 문자열을 날짜 형식으로 변환한다.

```
ds2.측정일시= pd.to_datetime(ds2.측정일시)
ds2.측정일시.head(3)
```

```
0   2018-01-01
1   2018-01-01
2   2018-01-01
Name: 측정일시, dtype: datetime64[ns]
```

날짜 형식 활용 Series.dt.형식 (요일 구하기)

	ds2.측정일시.dt.year
0	2018
1	2018
2	2018

	ds2.측정일시.dt.month
0	1
1	1
2	1

	ds2.측정일시.dt.day
0	1
1	1
2	1

	ds2.측정일시.dt.quarter
0	1
1	1
2	1

	ds2.측정일시.dt.weekday
0	0
1	0
2	0

`Series.dt.weekday`

The day of the week with Monday=0, Sunday=6

시간 데이터 만들기 - datetime module

```
import datetime
```

```
datetime.datetime.now()
```

```
datetime.datetime(2019, 5, 9, 9, 21, 54, 215373)
```

- 시간 데이터 출력방법 : year, month, day, hour, minute, second

```
now=datetime.datetime.now()
```

```
print(now.year, now.hour, now.strftime("%A"))
```

```
2019 9 Thursday
```

strftime() - 시간오브젝트를 문자열 포맷으로 만든다. - 새변수 =DT.strftime('Di')

```
now.strftime("%A")
```

```
'Thursday'
```

```
weekday=now.strftime("%a")
month=now.strftime("%B")
print(weekday, month)
```

```
Thu May
```

datetime.date / datetime.datetime

```
x=datetime.date(1961,1,3)
x.strftime('%A')
```

```
'Tuesday'
```

생일 1961.1.3일은 목요일

```
y=datetime.datetime(1989,5,28,22,23)
print(y.strftime('%A'), y.strftime('%p'))
```

```
Sunday PM
```

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%