BITAMIN 4th Session

데이터 프레임의 다양한 응용 및 판다스 복습

발표자

이상연, 이수진

2조

김언지, 이상연, 이수진, 홍진수

CONTENTS

데이터 프레임의 다양한 응용 및 판다스 복습

시리즈 변환

- 딕셔너리, 리스트, 튜플 등을 판다스의 시리즈 타입으로 변환!
- 딕셔너리와 구조가 비슷해 딕셔너리를 시리즈로 변환하는 방법을 많이 사용

```
import pandas as pd
```

```
#series
키={'a':160,'b':180,'c':167,'d':179}
key=pd.Series(키)
key
```

```
a    160
b    180
c    167
d    179
dtype: int64
```

인덱스 배열 : Series객체.index
데이터 값 배열 : Series객체.values

```
#Series Index
key.index=['엄마','아빠','나','동생']
key
```

```
엄마    160
아빠    180
나     167
동생    179
dtype: int64
```

데이터프레임 변환

행 인덱스/ 열 이름 설정 :
pandas.DataFrame(2차원 배열,
index=행 인덱스 배열,
columns=열 이름 배열)

```
c = {'col_1':[1,11],'col_2':[2,12],'col_3':[3,13]}
cdf = pd.DataFrame(c, index = ['id_1','id_2'])
```

- 데이터프레임에서 ndarray, list, dictionary로 변환

```
#DataFrame을 ndarry, list, dictionary 로 변환

#DataFrame to ndarray  #자주사용
a2 = cdf.values #values를 사용해서 ndarray로 변환
print(type(a2))
print(a2.shape)
print(a2)
print('₩n')

#DataFrame to list
b2 = cdf.values.tolist()
print(type(b2))
print(b2)
print('₩n')

#DataFrame to dict

c2 = cdf.to_dict('list')
print(type(c2))
print(c2)
```

```
      col_1  col_2  col_3
id_1      1      2      3
id_2     11     12     13
```

```
<class 'numpy.ndarray'>
(2, 3)
[[ 1  2  3]
 [11 12 13]]
```

```
<class 'list'>
[[1, 2, 3], [11, 12, 13]]
```

```
<class 'dict'>
{'col_1': [1, 11], 'col_2': [2, 12], 'col_3': [3, 13]}
```

데이터프레임

```
df=pd.read_csv('./diamonds.csv')
df= df.iloc[0:5] #행선택
df=df[['carat','cut','color','clarity','depth','table','price']]
print(df)
print('₩n')
print(df.index)
print('₩n')
print(df.columns)
```

실습에 사용할 데이터 셋의 형태로 가공
→ 행/열 선택

```
   carat      cut color clarity  depth  table  price
0   0.23    Ideal     E     SI2   61.5   55.0    326
1   0.21  Premium     E     SI1   59.8   61.0    326
2   0.23     Good     E     VS1   56.9   65.0    327
3   0.29  Premium     I     VS2   62.4   58.0    334
4   0.31     Good     J     SI2   63.3   58.0    335


RangeIndex(start=0, stop=5, step=1)


Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price'],
dtype='object')
```

데이터프레임_열추가

열 추가 : DataFrame객체['추가하려는 열 이름'] = 데이터 값

```
#열 추가

df['num']=['dia1','dia2','dia3','dia4','dia5']
df
```

인덱싱으로 사용하기 위해
다이아번호를 만들어 줍니다.

	carat	cut	color	clarity	depth	table	price	num
0	0.23	Ideal	E	SI2	61.5	55.0	326	dia1
1	0.21	Premium	E	SI1	59.8	61.0	326	dia2
2	0.23	Good	E	VS1	56.9	65.0	327	dia3
3	0.29	Premium	I	VS2	62.4	58.0	334	dia4
4	0.31	Good	J	SI2	63.3	58.0	335	dia5

데이터프레임_인덱스

```
#특정열을 인덱스로 활용
ndf=df.set_index(['num'])
ndf
```

num	carat	cut	color	clarity	depth	table	price
dia1	0.23	Ideal	E	SI2	61.5	55.0	326
dia2	0.21	Premium	E	SI1	59.8	61.0	326
dia3	0.23	Good	E	VS1	56.9	65.0	327
dia4	0.29	Premium	I	VS2	62.4	58.0	334
dia5	0.31	Good	J	SI2	63.3	58.0	335

행 인덱스 변경 : DataFrame객체.index=새로운 행 인덱스 배열
DataFrame객체.rename(index={기존 인덱스: 새 인덱스, …})

열 이름 변경 : DataFrame객체.columns=새로운 열 이름 배열
DataFrame객체.rename(columns={기존 이름: 새이름,…})

특정열을 행 인덱스로 설정 : DataFrame객체.set_index([‘열이름’]또는 ' 열이름‘)

데이터프레임_행/열 삭제

행 삭제 : DataFrame객체.drop(행 인덱스 또는 배열, axis=0)
열 삭제 : DataFrame객체.drop(열 이름 또는 배열, axis=1)

Drop() 메소드는 기존 개체를 변경하지 않는다.
(Inplace=True)옵션을 추가하면 원본 객체를 변경

```
#행삭제
df2=ndf[:]
df2.drop('dia5',inplace=True)
print(df2)
```

num	carat	cut	color	clarity	depth	table	price
dia1	0.23	Ideal	E	SI2	61.5	55.0	326
dia2	0.21	Premium	E	SI1	59.8	61.0	326
dia3	0.23	Good	E	VS1	56.9	65.0	327
dia4	0.29	Premium	I	VS2	62.4	58.0	334

```
#열삭제
df2=ndf[:]
df2.drop('table', axis=1,inplace=True)
print(df2)
```

num	carat	cut	color	clarity	depth	price
dia1	0.23	Ideal	E	SI2	61.5	326
dia2	0.21	Premium	E	SI1	59.8	326
dia3	0.23	Good	E	VS1	56.9	327
dia4	0.29	Premium	I	VS2	62.4	334
dia5	0.31	Good	J	SI2	63.3	335

데이터프레임_행/열 선택

```
#행선택
label=ndf.loc[['dia1']]
position=ndf.iloc[0:1]
print(label)
print('₩n')
print(position)
```

```
      carat    cut color clarity  depth  table  price
num
dia1   0.23  Ideal     E     SI2   61.5   55.0    326


      carat    cut color clarity  depth  table  price
num
dia1   0.23  Ideal     E     SI2   61.5   55.0    326
```

```
#열선택 가격만 보기
price=ndf['price']
print(price)
```

```
num
dia1    326
dia2    326
dia3    327
dia4    334
dia5    335
Name: price, dtype: int64
```

구분	loc	Iloc
탐색 대상	인덱스 이름(index label)	정수형 위치 인덱스 (integer position)

열 1개 선택(시리즈 생성) : DataFrame객체[‘열 이름’] 또는 DataFrame객체.열 이름

열 n개 선택(데이터 프레임 생성) : DataFrame객체[[열1,열2, … , 열n]]

데이터프레임_행/열 선택

```
#고급 슬라이싱
jump=ndf.iloc[0:5:2]
rev=ndf.iloc[::-1]
print(jump)
print('₩n')
print(rev)
```

```
      carat    cut color clarity  depth  table  price
num
dia1   0.23  Ideal     E     SI2   61.5   55.0    326
dia3   0.23   Good     E     VS1   56.9   65.0    327
dia5   0.31   Good     J     SI2   63.3   58.0    335


      carat      cut color clarity  depth  table  price
num
dia5   0.31     Good     J     SI2   63.3   58.0    335
dia4   0.29  Premium     I     VS2   62.4   58.0    334
dia3   0.23     Good     E     VS1   56.9   65.0    327
dia2   0.21  Premium     E     SI1   59.8   61.0    326
dia1   0.23    Ideal     E     SI2   61.5   55.0    326
```

범위 슬라이싱
DataFrame객체.iloc[시작인덱스:끝인덱스:슬라이싱 간격}

역순 인덱싱
DataFrame객체.iloc[시작인덱스:끝인덱스: -1 }

데이터프레임_행/열 선택

인덱스 이름 : DataFrame객체.loc[행 인덱스,열 이름]

정수 위치 인덱스 : DataFrame객체.iloc[행 번호, 열 번호]

원소 값 변경 :DataFrame객체의 일부분 또는 원소를 선택 = 새로운 값

```
#원소 값 변경
ndf.loc['dia5','price']=400
ndf
```

num	carat	cut	color	clarity	depth	table	price
dia1	0.23	Ideal	E	SI2	61.5	55.0	326
dia2	0.21	Premium	E	SI1	59.8	61.0	326
dia3	0.23	Good	E	VS1	56.9	65.0	327
dia4	0.29	Premium	I	VS2	62.4	58.0	334
dia5	0.31	Good	J	SI2	63.3	58.0	400

```
ndf.iloc[0,6]=300
ndf
```

num	carat	cut	color	clarity	depth	table	price
dia1	0.23	Ideal	E	SI2	61.5	55.0	300
dia2	0.21	Premium	E	SI1	59.8	61.0	326
dia3	0.23	Good	E	VS1	56.9	65.0	327
dia4	0.29	Premium	I	VS2	62.4	58.0	334
dia5	0.31	Good	J	SI2	63.3	58.0	400

데이터프레임_행/열 추가

행 추가 : DataFrame.loc[‘새로운 행 이름‘] = 데이터 값(또는 배열)

열 추가 : DataFrame객체[‘추가하려는 열 이름’] = 데이터 값

```
#행추가
ndf.loc['dia6']=[0.30,'Ideal','E','VS1','63.0','60.0','401']
ndf
```

num	carat	cut	color	clarity	depth	table	price
dia1	0.23	Ideal	E	SI2	61.5	55	300
dia2	0.21	Premium	E	SI1	59.8	61	326
dia3	0.23	Good	E	VS1	56.9	65	327
dia4	0.29	Premium	I	VS2	62.4	58	334
dia5	0.31	Good	J	SI2	63.3	58	400
dia6	0.30	Ideal	E	VS1	63.0	60.0	401

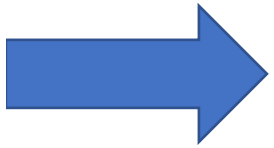
```
#기존 행 복사
ndf.loc['dia7']=ndf.loc['dia3']
ndf
```

num	carat	cut	color	clarity	depth	table	price
dia1	0.23	Ideal	E	SI2	61.5	55	300
dia2	0.21	Premium	E	SI1	59.8	61	326
dia3	0.23	Good	E	VS1	56.9	65	327
dia4	0.29	Premium	I	VS2	62.4	58	334
dia5	0.31	Good	J	SI2	63.3	58	400
dia6	0.30	Ideal	E	VS1	63.0	60.0	401
dia7	0.23	Good	E	VS1	56.9	65	327

데이터프레임_전치

행,열 바꾸기 : DataFrame객체.transpose() 또는 DataFrame객체.T

num	carat	cut	color	clarity	depth	table	price
dia1	0.23	Ideal	E	SI2	61.5	55	300
dia2	0.21	Premium	E	SI1	59.8	61	326
dia3	0.23	Good	E	VS1	56.9	65	327
dia4	0.29	Premium	I	VS2	62.4	58	334
dia5	0.31	Good	J	SI2	63.3	58	400
dia6	0.30	Ideal	E	VS1	63.0	60.0	401
dia7	0.23	Good	E	VS1	56.9	65	327



```
#열 위치 바꾸기
df=ndf.T
df
```

num	dia1	dia2	dia3	dia4	dia5	dia6	dia7
carat	0.23	0.21	0.23	0.29	0.31	0.3	0.23
cut	Ideal	Premium	Good	Premium	Good	Ideal	Good
color	E	E	E	I	J	E	E
clarity	SI2	SI1	VS1	VS2	SI2	VS1	VS1
depth	61.5	59.8	56.9	62.4	63.3	63.0	56.9
table	55	61	65	58	58	60.0	65
price	300	326	327	334	400	401	327

데이터 프레임 구조 확인

02. 데이터 살펴보기

데이터프레임 객체.head(): 첫 5줄 확인

데이터프레임 객체.tail(): 마지막 5줄 확인

diamonds.csv 파일 사용!!

```
dia = pd.read_csv('./diamonds.csv')
print(dia.head())
print('₩n')
print(dia.tail())
```

```
   carat      cut color clarity  depth  table  price     x     y     z
0   0.23    Ideal     E     SI2   61.5   55.0    326  3.95  3.98  2.43
1   0.21  Premium     E     SI1   59.8   61.0    326  3.89  3.84  2.31
2   0.23     Good     E     VS1   56.9   65.0    327  4.05  4.07  2.31
3   0.29  Premium     I     VS2   62.4   58.0    334  4.20  4.23  2.63
4   0.31     Good     J     SI2   63.3   58.0    335  4.34  4.35  2.75


       carat        cut color clarity  depth  table  price     x     y     z
53935   0.72      Ideal     D     SI1   60.8   57.0   2757  5.75  5.76  3.50
53936   0.72       Good     D     SI1   63.1   55.0   2757  5.69  5.75  3.61
53937   0.70  Very Good     D     SI1   62.8   60.0   2757  5.66  5.68  3.56
53938   0.86    Premium     H     SI2   61.0   58.0   2757  6.15  6.12  3.74
53939   0.75      Ideal     D     SI2   62.2   55.0   2757  5.83  5.87  3.64
```

데이터프레임 객체.shape(): 데이터프레임 크기 확인

데이터프레임 객체.info(): 데이터프레임 기본정보 확인

데이터프레임 객체.dtypes(): 각 열의 자료형 확인

```
print(dia.shape)
print('₩n')
print(dia.info())
```

```
(53940, 10)


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object
 2   color    53940 non-null  object
 3   clarity  53940 non-null  object
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
None
```

```
print(dia.dtypes)
```

```
carat      float64
cut         object
color       object
clarity     object
depth      float64
table      float64
price        int64
x          float64
y          float64
z          float64
dtype: object
```

데이터 요약 정보 / 개수 확인

데이터프레임 객체.describe(): 기술통계 요약 정보 확인

Include = 'all': 산술 데이터가 아닌 열에 대한 정보를 추가하는 옵션

```
print(dia.describe(include='all'))
```

	carat	cut	color	clarity	depth	table	₩
count	53940.000000	53940	53940	53940	53940.000000	53940.000000	
unique	NaN	5	7	8	NaN	NaN	
top	NaN	Ideal	G	SI1	NaN	NaN	
freq	NaN	21551	11292	13065	NaN	NaN	
mean	0.797940	NaN	NaN	NaN	61.749405	57.457184	
std	0.474011	NaN	NaN	NaN	1.432621	2.234491	
min	0.200000	NaN	NaN	NaN	43.000000	43.000000	
25%	0.400000	NaN	NaN	NaN	61.000000	56.000000	
50%	0.700000	NaN	NaN	NaN	61.800000	57.000000	
75%	1.040000	NaN	NaN	NaN	62.500000	59.000000	
max	5.010000	NaN	NaN	NaN	79.000000	95.000000	

	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000
unique	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN
mean	3932.799722	5.731157	5.734526	3.538734
std	3989.439738	1.121761	1.142135	0.705699
min	326.000000	0.000000	0.000000	0.000000
25%	950.000000	4.710000	4.720000	2.910000
50%	2401.000000	5.700000	5.710000	3.530000
75%	5324.250000	6.540000	6.540000	4.040000
max	18823.000000	10.740000	58.900000	31.800000

데이터프레임 객체.count(): 각 열의 데이터 개수 확인

데이터프레임 객체[열 이름].value_counts(): 고유값 확인

```
print(dia.count())
print('₩n')
print(type(dia.count()))
```

```
carat      53940
cut        53940
color      53940
clarity    53940
depth      53940
table      53940
price      53940
x          53940
y          53940
z          53940
dtype: int64


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

```
print(dia['cut'].value_counts())
```

```
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: cut, dtype: int64
```

통계함수 적용

<산술형 데이터만 출력>

데이터프레임.mean(): 평균

데이터프레임.std(): 표준편차

데이터프레임.median(): 중앙값

데이터프레임.corr(): 상관계수

```
print(dia.mean())
```

```
carat       0.797940
depth      61.749405
table      57.457184
price    3932.799722
x           5.731157
y           5.734526
z           3.538734
dtype: float64
```

```
print(dia.std())
```

```
carat       0.474011
depth       1.432621
table       2.234491
price    3989.439738
x           1.121761
y           1.142135
z           0.705699
dtype: float64
```

```
print(dia.median())
```

```
carat       0.70
depth      61.80
table      57.00
price    2401.00
x           5.70
y           5.71
z           3.53
dtype: float64
```

```
print(dia.corr())
```

```
          carat     depth     table     price         x         y         z
carat  1.000000  0.028224  0.181618  0.921591  0.975094  0.951722  0.953387
depth  0.028224  1.000000 -0.295779 -0.010647 -0.025289 -0.029341  0.094924
table  0.181618 -0.295779  1.000000  0.127134  0.195344  0.183760  0.150929
price  0.921591 -0.010647  0.127134  1.000000  0.884435  0.865421  0.861249
x      0.975094 -0.025289  0.195344  0.884435  1.000000  0.974701  0.970772
y      0.951722 -0.029341  0.183760  0.865421  0.974701  1.000000  0.952006
z      0.953387  0.094924  0.150929  0.861249  0.970772  0.952006  1.000000
```

<산술형/문자형 데이터 출력>

데이터프레임.max(): 최대값

데이터프레임.min(): 최소값

(문자열을 ASCII숫자로 변환해 크기를 비교)

```
print(dia[['carat','price']].max())
```

```
carat      5.01
price  18823.00
dtype: float64
```

```
print(dia[['carat','price']].min())
```

```
carat    0.2
price  326.0
dtype: float64
```

판다스 내장 그래프 도구 활용

hiring_rate.csv 파일 사용!!

```
hr = pd.read_csv('./hiring_rate.csv')
hr = hr.set_index(['Year'])
hr.index = hr.index.map(str)  #행 인덱스를 문자형으로 변경
print(hr.head())
print('₩n')
hr.plot()
```

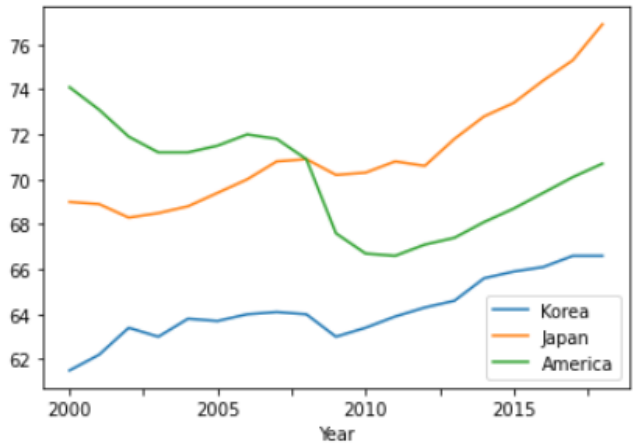
Year	Korea	Japan	America
2000	61.5	69.0	74.1
2001	62.2	68.9	73.1
2002	63.4	68.3	71.9
2003	63.0	68.5	71.2
2004	63.8	68.8	71.2

데이터프레임.plot(): 선형 그래프

데이터 프레임.plot(kind = 'bar'): 막대 그래프

데이터 프레임.plot(kind = 'hist'): 히스토그램

```
<matplotlib.axes._subplots.AxesSubplot at 0x1782e169b50>
```

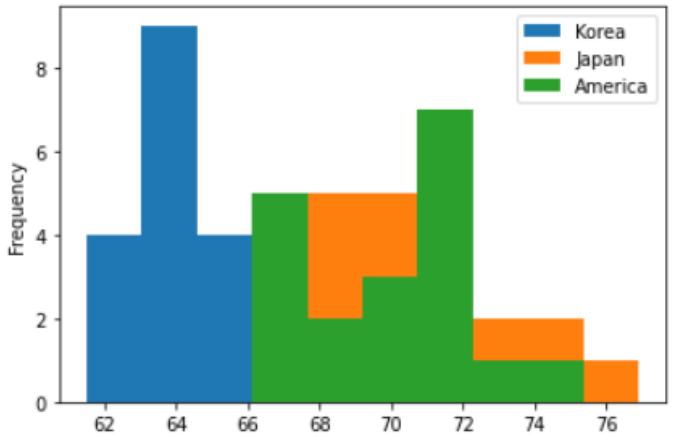



```
hr.plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1782e233af0>
```


```
hr.plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1782d72aeb0>
```

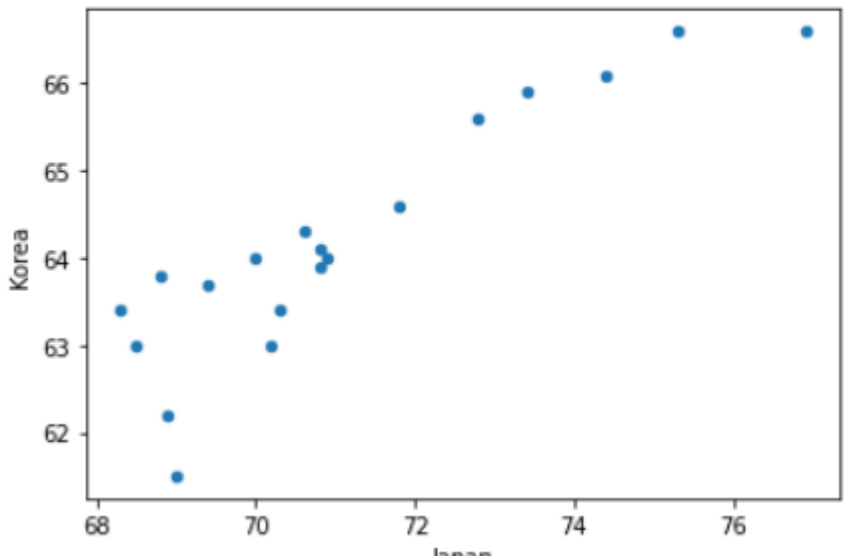


판다스 내장 그래프 도구 활용

데이터 프레임.plot(kind = 'scatter'): 산점도

```
hr.plot(x = 'Japan', y = 'Korea', kind = 'scatter')
```

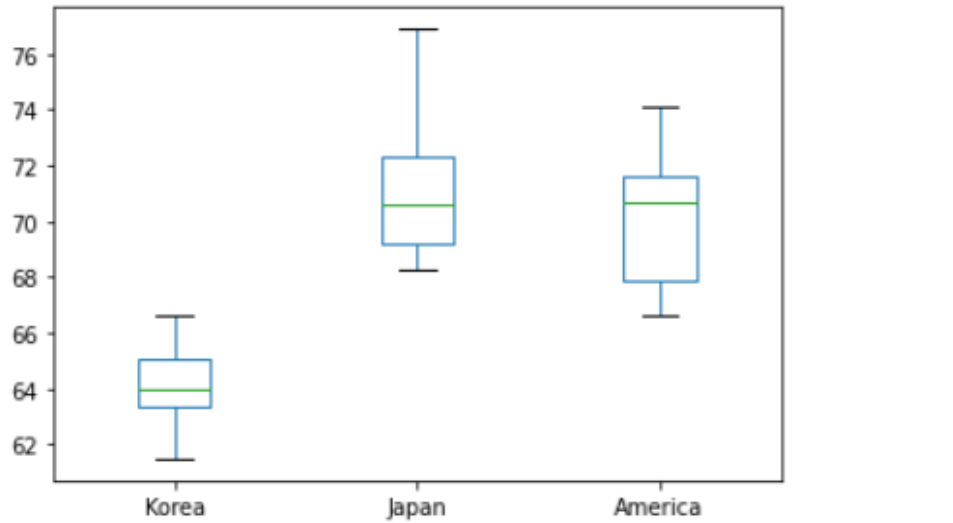
```
<matplotlib.axes._subplots.AxesSubplot at 0x1782e736c10>
```



데이터 프레임.plot(kind = 'box'): 박스플롯(상자그림)

```
hr[['Korea','Japan','America']].plot(kind ='box')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1782e2ccb20>
```



데이터 사전 처리

서울특별시 공공자전거 이용정보(월별)_201901_201906.csv 파일 사용!

```
import pandas as pd

bicycle = pd.read_excel("서울특별시 공공자전거 이용정보(월별)_201901_201906.xlsx")
bicycle
# 출처 : 서울시 열린데이터 광장
```

(실제 데이터이므로 행 수가 많아 불러오는데 시간이 좀 오래 걸립니다.)

	대여일자	대여소번호	대여소	대여구분코드	성별	연령대코드	이용건수	운동량	탄소량	이동거리(M)	이동시간(분)
0	2019-01-01	3	증랑센터	일일(회원)	M	AGE_003	12	288.87	2.25	9690	117
1	2019-01-01	3	증랑센터	일일(회원)	M	AGE_004	8	424.63	3.32	14310	151
2	2019-01-01	3	증랑센터	일일(회원)	M	AGE_005	27	4579.48	41.29	177910	542
3	2019-01-01	3	증랑센터	정기	M	AGE_003	17	3438.8	33.58	144730	593
4	2019-01-01	5	상암센터 정비실	일일(회원)	M	AGE_004	1	78.31	0.72	3090	33
...	...	...	...	...	...	...	...	...	...	...	...
284578	2019-06-01	9997	9997.강남센터	정기	M	AGE_002	2	95.7	0.85	3690	20
284579	2019-06-01	9997	9997.강남센터	정기	M	AGE_003	1	41.58	0.41	1750	14
284580	2019-06-01	9997	9997.강남센터	정기	M	AGE_005	1	35.64	0.28	1200	7

284583 rows × 11 columns

누락 데이터 확인

```
1 # 데이터 확인하기
2 # 자료형 및 결측값 확인
3 bicycle.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284583 entries, 0 to 284582
Data columns (total 11 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   대여일자      284583 non-null  datetime64[ns]
 1   대여소번호     284583 non-null  int64
 2   대여소       284583 non-null  object
 3   대여구분코드    284583 non-null  object
 4   성별        271732 non-null  object
 5   연령대코드     284583 non-null  object
 6   이용건수      284583 non-null  int64
 7   운동량       284583 non-null  object
 8   탄소량       284583 non-null  object
 9   이동거리(M)   284583 non-null  int64
 10  이동시간(분)   284583 non-null  int64
dtypes: datetime64[ns](1), int64(4), object(6)
memory usage: 23.9+ MB
```

누락 데이터 확인

284583행 중, 271732행만 non-null인 '성별'변수에 결측치가 있음을 확인

자료형 확인

'대여소번호' 변수가 정수형임은 맞지만 숫자의 상대적인 크기는 별 의미가 없음. 따라서 '대여소번호'열의 자료형은 **범주형으로 표현**하는 것이 적절

'대여구분코드' 변수는 문자형보다는 **범주형으로 표현**하는 것이 더 적절

연령대코드' 변수는 문자형보다는 **범주형으로 표현**하는 것이 더 적절

'운동량'변수과 '탄소량'변수는 **실수형일 것으로 추측**되지만 자료형이 문자형으로 되어 있음. 데이터에 오류가 있는지 확인이 필요

누락 데이터 처리

사전에 284583행 중, 271732행만 non-null인 '성별': 결측치가 있음을 확인

```
3 bicycle['성별'].value_counts(dropna = False)
```

```
        111295
M        86349
F        73977
NaN      12851
m           65
f           46
Name: 성별, dtype: int64
```

value_counts(dropna =False) 를 통해 결측치 개수 확인

결측치로 처리된 NaN 뿐만 아니라
공백 또한 데이터가 누락되었던 것으로 확인

따라서, NaN와 공백 모두 결측치로 처리!

```
import numpy as np
bicycle['성별'].replace(' ', np.nan, inplace = True)
```

```
(bicycle['성별'].isnull().sum(axis = 0) / len(bicycle))
```

```
성별'변수의 결측치 비율 :  0.4362382854914032
```

```
# 누락 데이터 ('성별') 제거
bicycle = bicycle.drop(columns = ['성별'], axis = 1)
# 확인
bicycle.info()
```

'성별' 변수의 43%가 결측치로 확인
결측치가 너무 많아 '성별'변수를 사용할 수 없을 것으로 판단!
열 삭제

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284583 entries, 0 to 284582
Data columns (total 10 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   대여일자      284583 non-null  datetime64[ns]
 1   대여소번호     284583 non-null  int64
 2   대여소       284583 non-null  object
 3   대여구분코드    284583 non-null  object
 4   연령대코드     284583 non-null  object
 5   이용건수      284583 non-null  int64
 6   운동량       284583 non-null  object
 7   탄소량       284583 non-null  object
 8   이동거리(M)   284583 non-null  int64
 9   이동시간(분)   284583 non-null  int64
dtypes: datetime64[ns](1), int64(4), object(5)
memory usage: 21.7+ MB
```

데이터 표준화 – 자료형 변환

```
print(bicycle['대여소번호'].unique())
bicycle['대여소번호'] = bicycle['대여소번호'].astype('category') # 총 1552개의 대여소
```

```
[   3    5  101 ...  462  972 9993]
```

```
print(bicycle['대여구분코드'].unique())
bicycle['대여구분코드'] = bicycle['대여구분코드'].astype('category')
```

```
['일일(회원)' '정기' '단체' '일일(비회원)']
```

```
print(bicycle['연령대코드'].unique())
bicycle['연령대코드'] = bicycle['연령대코드'].astype('category')
```

```
['AGE_003' 'AGE_004' 'AGE_005' 'AGE_002' 'AGE_008' 'AGE_001' 'AGE_006'
 'AGE_007']
```

'대여소번호' 변수가 정수형임은 맞지만 숫자의 상대적인 크기는 의미가 없음
범주형으로 표현하는 것이 적절

'대여구분코드' 변수는 문자형보다는 범주형으로 표현하는 것이 더 적절

연령대코드' 변수는 문자형보다는 범주형으로 표현하는 것이 더 적절

데이터 표준화 – 자료형 변환

'운동량'변수과 '탄소량'변수 실수형일 것으로 추측
하지만 자료형이 문자형으로 되어 있음

데이터에 오류가 있을 것으로 추측

```
print("운동량 : ")
print(bicycle['운동량'].value_counts())
print('₩n')
print("탄소량 : ")
print(bicycle['탄소량'].value_counts())
```

```
운동량 :
0          779
           276
39.64       63
45.3        57
54.05       52
           ...
10063.67     1
17061.54     1
1216.24      1
593.27       1
22812        1
Name: 운동량, Length: 185854, dtype: int64
```

'운동량'변수에서 공백 발견! 결측치로 간주

결측치 개수 파악하기

```
import numpy as np
# 공백을 NaN 처리하기

bicycle['운동량'].replace(' ', np.nan, inplace = True)
bicycle['탄소량'].replace(' ', np.nan, inplace = True)

# 확인
print("운동량'변수의 결측치 비율 : " , (bicycle['운동량'].isnull().sum(axis = 0) / len(bicycle)))
print("탄소량'변수의 결측치 비율 : " , (bicycle['탄소량'].isnull().sum(axis = 0) / len(bicycle)))
```

```
운동량'변수의 결측치 비율 :  0.0009698400818039025
탄소량'변수의 결측치 비율 :  0.0009698400818039025
```

운동량과 탄소량 결측치가 아주 적은 비율로 확인됨

데이터 표준화 – 자료형 변환

```
# 결측 행 제거
bicycle = bicycle.dropna(subset = ['운동량'], how = 'any', axis = 0)
bicycle = bicycle.dropna(subset = ['탄소량'], how = 'any', axis = 0)

# 확인
bicycle.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 284307 entries, 0 to 284582
Data columns (total 10 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   대여일자      284307 non-null  datetime64[ns]
 1   대여소번호     284307 non-null  int64
 2   대여소       284307 non-null  object
 3   대여구분코드    284307 non-null  object
 4   연령대코드     284307 non-null  object
 5   이용건수      284307 non-null  int64
 6   운동량       284307 non-null  float64
 7   탄소량       284307 non-null  float64
 8   이동거리(M)   284307 non-null  int64
 9   이동시간(분)   284307 non-null  int64
dtypes: datetime64[ns](1), float64(2), int64(4), object(3)
memory usage: 23.9+ MB
```

284583행 중, 276행만 결측
매우 적은 비율이므로 결측 행 제거
dropna 사용

결측행을 제거했으므로

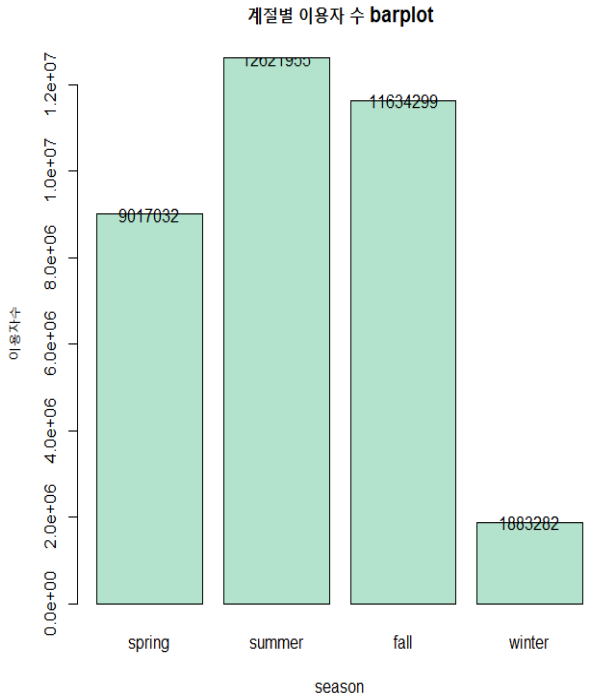
‘운동량‘ , ‘탄소량‘ 변수의 자료형이

실수형으로 변환 되었고

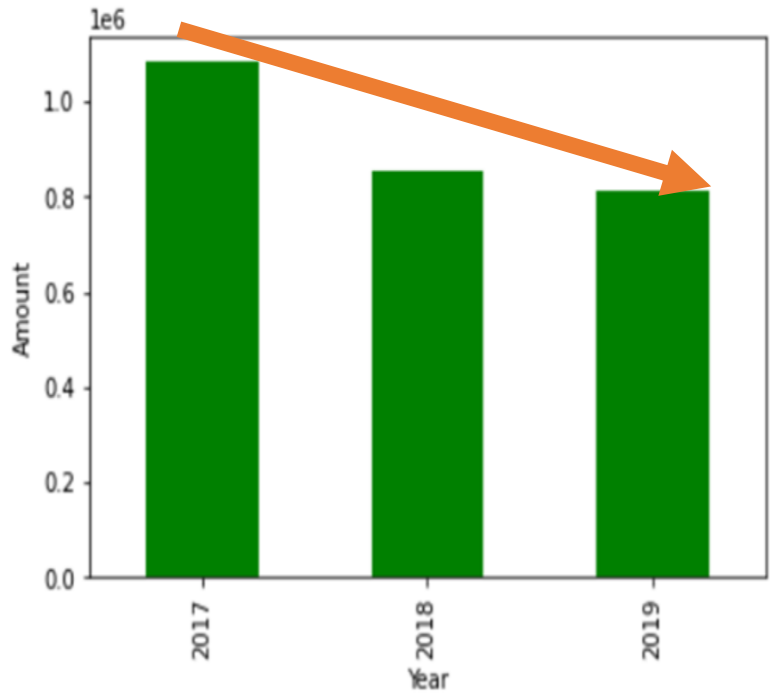
284583 – 276 = 284307행이 남았음을 확인

시계열 데이터란?

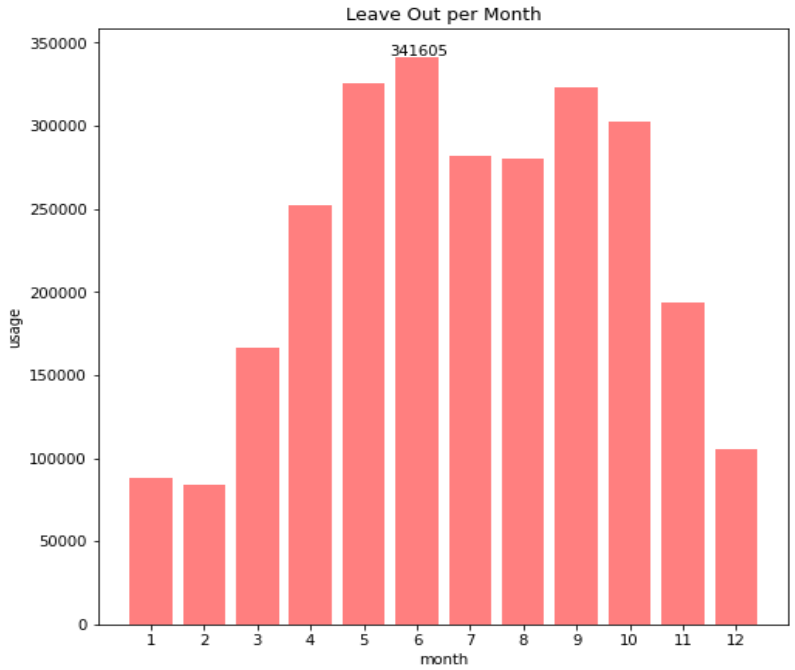
: 연도 별, 계절 별, 월 별, 일 별, 시, 분, 초 별로 시간의 흐름에 따라 관측된 자료



계절 별 시계열 데이터



연도 별 시계열 데이터



월 별 시계열 데이터

시계열 데이터 탐색적 자료 분석

날짜형 변수 '대여일자' 확인

```
1  bicycle['대여일자'].unique()
```

```
array(['2019-01-01T00:00:00.000000000', '2019-02-01T00:00:00.000000000',
       '2019-03-01T00:00:00.000000000', '2019-04-01T00:00:00.000000000',
       '2019-05-01T00:00:00.000000000', '2019-06-01T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

총 6개월의 대여 일자

- 월별 데이터 이므로 일은 모두 1일로만 통일 되어있음

```
3  bicycle_EDA = bicycle[:]
4  bicycle_EDA['대여일자'] = bicycle_EDA['대여일자'].dt.to_period(freq = 'M')
5
```

	대여일자
0	2019-01-01
1	2019-01-01
2	2019-01-01
3	2019-01-01
4	2019-01-01

	대여일자
0	2019-01
1	2019-01
2	2019-01
3	2019-01
4	2019-01

따라서 Timestamp를 Period로 변환하여 연-월-일 표기를 연-월로만 변경

시계열 데이터 탐색적 자료 분석

시계열 그래프 그리기 위해 '월' 변수 생성

: dt 속성을 이용하여 '대여일자'열의 연-월-일 정보를 월로 구분

```
bicycle_EDA['월'] = bicycle_EDA['대여일자'].dt.month
bicycle_EDA.head()
```

```
bicycle_EDA.info()
bicycle_EDA['월'] = bicycle_EDA['월'].astype('category')
```

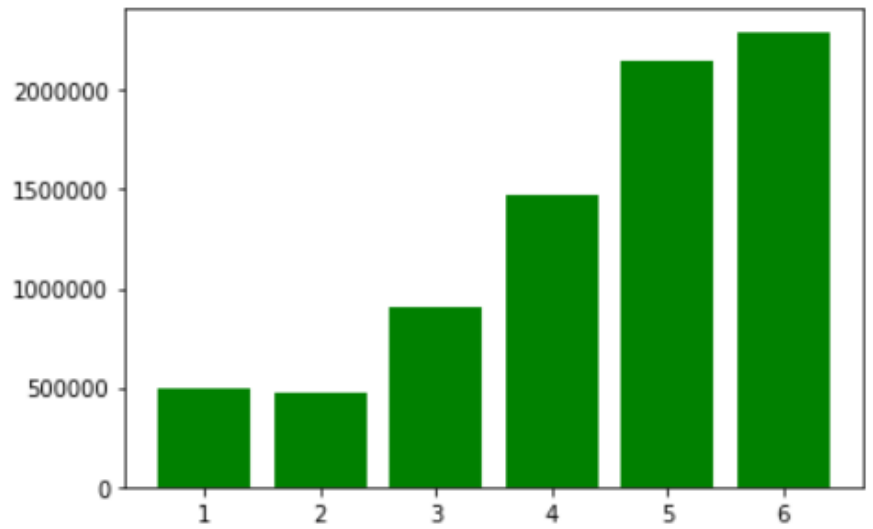
월별 이용자 수 그래프 그리기

```
# 월별 이용자 수 plot 그려보기
from matplotlib import pyplot as plt

# 월별 이용자 수 구하기
month_user = bicycle_EDA.groupby('월')['이용건수'].sum()
print(month_user)

plt.bar(month_user.index, month_user.values, color = 'green')
plt.show()
```

```
월
1      495535
2      471488
3      904744
4     1468794
5     2151012
6     2293801
Name: 이용건수, dtype: int64
```



범주형(카테고리) 데이터 처리

```
bicycle.info()
```

```
1   대여소번호     284307 non-null  category
2   대여소        284307 non-null  object
3   대여구분코드    284307 non-null  category
4   연령대코드     284307 non-null  category
```

범주형 변수

- 대여소번호 : 모델에 넣지 않을 것이므로 삭제
- 대여소 : 모델에 넣지 않을 것이므로 삭제
- 대여구분코드 : 더미 변수 생성
- 연령대코드 : 더미 변수 생성

```
bicycle.drop(['대여소번호', '대여소'], axis = 1, inplace = True)
bicycle.head()
```

	대여일자	대여구분코드	연령대코드	이용건수	운동량	탄소량	이동거리(M)	이동시간(분)
0	2019-01-01	일일(회원)	AGE_003	12	288.87	2.25	9690	117
1	2019-01-01	일일(회원)	AGE_004	8	424.63	3.32	14310	151
2	2019-01-01	일일(회원)	AGE_005	27	4579.48	41.29	177910	542
3	2019-01-01	정기	AGE_003	17	3438.80	33.58	144730	593
4	2019-01-01	일일(회원)	AGE_004	1	78.31	0.72	3090	33

범주형(카테고리) 데이터 처리

더미변수 생성

```
b1 = pd.get_dummies(bicycle['대여구분코드'])
b1
```

	단체	일일(비회원)	일일(회원)	정기
0	0	0	1	0
1	0	0	1	0
2	0	0	1	0
3	0	0	0	1
4	0	0	1	0
...	...	...	...	...
284578	0	0	0	1
284579	0	0	0	1
284580	0	0	0	1
284581	0	0	1	0
284582	0	0	0	1

284307 rows × 4 columns

```
b2 = pd.get_dummies(bicycle['연령대코드'])
b2
```

	AGE_001	AGE_002	AGE_003	AGE_004	AGE_005	AGE_006	AGE_007	AGE_008
0	0	0	1	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0	0
3	0	0	1	0	0	0	0	0
4	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...
284578	0	1	0	0	0	0	0	0
284579	0	0	1	0	0	0	0	0
284580	0	0	0	0	1	0	0	0
284581	0	0	0	1	0	0	0	0
284582	0	0	0	1	0	0	0	0

284307 rows × 8 columns

범주형(카테고리) 데이터 처리

데이터 프레임 합치기

```
bicycle = pd.concat([bicycle, b1, b2], axis = 1)
bicycle.drop(['대여구분코드', '연령대코드'], axis = 1, inplace = True)
bicycle.head()
```

	대여일자	이용건수	운동량	탄소량	이동거리(M)	이동시간(분)	단체	일일(비회원)	일일(회원)	정기	AGE_001	AGE_002	AGE_003	AGE_004	AGE_005	AGE_006	AGE_007	AGE_008
0	2019-01-01	12	288.87	2.25	9690	117	0	0	1	0	0	0	1	0	0	0	0	0
1	2019-01-01	8	424.63	3.32	14310	151	0	0	1	0	0	0	0	1	0	0	0	0
2	2019-01-01	27	4579.48	41.29	177910	542	0	0	1	0	0	0	0	0	1	0	0	0
3	2019-01-01	17	3438.80	33.58	144730	593	0	0	0	1	0	0	1	0	0	0	0	0
4	2019-01-01	1	78.31	0.72	3090	33	0	0	1	0	0	0	0	1	0	0	0	0

함수매핑

함수 매핑은 시리즈 또는 데이터프레임의 개별 원소를 특정 함수에 일대일 대응시키는 과정

사용하는 이유

- 사용자가 직접 만든 함수(lambda 포함)을 적용하는 것이 가능!
- 칼럼의 일괄적인 연산이 아닌 개별 레코드의 연산이 가능!

map은 시리즈
apply는 시리즈, 데이터프레임
applymap은 데이터프레임

	DataFrame	Series
apply	✅	✅
map		✅
applymap	✅	

함수매핑

시리즈 원소에 함수 매핑

시리즈 객체에 map() 혹은 apply() 메소드를 적용

매핑 함수에 시리즈의 모든 원소를 하나씩 입력하고 함수의 리턴값을 돌려받는다.

seaborn의 타이타닉 파일 사용!

```
import pandas as pd
import seaborn as sns

tdf = sns.load_dataset('titanic')
print(tdf[['age','fare']].head(3))

#시리즈 원소에 함수 매핑
#map()  시리즈의 값(value)하나하나를 lambda 인자로 넘김
sr1 = tdf['age'].map(lambda x: x+10) #apply써도 동일
print(sr1.head(3))
print('₩n')

def add_10(x):
    return x +10
sr2 = tdf['age'].apply(add_10) #map써도 동일
print(sr2.head(3))
print('₩n')
```

```
    age     fare
0  22.0   7.2500
1  38.0  71.2833
2  26.0   7.9250
```

+10

```
0    32.0
1    48.0
2    36.0
Name: age, dtype: float64


0    32.0
1    48.0
2    36.0
Name: age, dtype: float64
```

함수매핑

데이터프레임 각 행에 함수 매핑 (df1)

데이터프레임에 apply(axis=1) 메소드 적용
각 행을 매핑 함수의 인자로 전달한 후 행 인덱스가 매핑 결과로 반환되는 시리즈의 인덱스가 된다.

데이터프레임 각 열에 함수 매핑 (df2)

데이터프레임에 apply(axis=0) 메소드 적용
모든 열을 하나씩 분리하여 매핑 함수의 인자로 각 열이 전달된다.
(axis = 0이 디폴트)

데이터프레임 각 원소에 함수 매핑 (df3)

applymap() 을 이용해 모든 행렬 원소 하나하나에 동일한 함수 연산을 시행한다.

```
df1 = tdf[['age','fare']].apply(add_10,axis=1)
print(df1.head(3))
print('₩n')

df2 = tdf[['age','fare']].apply(add_10)
print(df2.head(3))
print('₩n')

df3 = tdf[['age','fare']].applymap(lambda x: x+10)
print(df3.head(3))
print('₩n')
```

```
    age     fare
0  32.0  17.2500
1  48.0  81.2833
2  36.0  17.9250
```

```
    age     fare
0  32.0  17.2500
1  48.0  81.2833
2  36.0  17.9250
```

```
    age     fare
0  32.0  17.2500
1  48.0  81.2833
2  36.0  17.9250
```

함수매핑

apply( ) 사용시 axis=0 과 axis=1의 차이점 유의하기

```
def maxf(series):
    return series.max()

df4 = tdf[['age','fare']].apply(maxf, axis=1) #각 행의 age와 fare 값을 비교
print(df4.head(6)) #각 행의 값들끼리 연산 후 시리즈 출력
print('₩n')


df5 = tdf[['age','fare']].apply(maxf, axis=0)#각 열 내에서의 값끼리 비교
print(df5.head(6)) #각 열내의 값끼리 연산 후 시리즈 출력
print('₩n')
print(tdf.age.max(), tdf.fare.max())
```

```
0    22.0000
1    71.2833
2    26.0000
3    53.1000
4    35.0000
5     8.4583
dtype: float64


age      80.0000
fare    512.3292
dtype: float64


80.0 512.3292
```

```
def min_max(x):
    return x.max() - x.min()
sr3 = tdf[['age','fare']].apply(min_max)
print(sr3) #각 열의 (최대값 - 최소값) 출력
```

```
age      79.5800
fare    512.3292
dtype: float64
```

함수매핑

값들을 분석 목적에 부합하도록 원하는 방식으로 변경

yes no를 0과 1로 표현

```
#응용
tdf['alive_n'] = tdf['alive'].apply(lambda x:1 if x=='yes' else 0)
tdf['child_adult'] = tdf['age'].apply(lambda x: 'child' if x<=15 else 'adult')
tdf[['age','child_adult']].head(10)

tdf['age_cat'] = tdf['age'].apply(lambda x: 'child' if x<= 15 else ('adult' if x < 65 else 'elderly'))
tdf[['age','age_cat']].iloc[110:120]

def get_category(age):
    x = ''
    if age <= 5:  x= 'baby'
    elif age <= 12: x='child'
    elif age <= 18: x='teenager'
    elif age <= 25: x='student'
    elif age <= 35: x='young adult'
    elif age <= 60: x='adult'
    else: x='elderly'

    return x

tdf['age_cat2'] = tdf['age'].apply(lambda x: get_category(x))
tdf[['age','age_cat','age_cat2']].iloc[110:120]
```

나이에 따라 새로운 기준으로 데이터 분류

열 재구성

하나의 열 내에 존재하는 여러가지 정보를 분리시키기

이전에 사용했던 공공자전거 데이터 파일 사용!

```
3  print(bicycle.head())
4  print(bicycle.dtypes)
5  bicycle['대여일자'] = bicycle['대여일자'].astype('str')
6  print(bicycle['대여일자'].dtype)
7  dates = bicycle['대여일자'].str.split('-')
8  print(dates.head())
9
0  #시리즈의 문자열 리스트 인덱싱  str.get(인덱스)
1  bicycle['연'] = dates.str.get(0)
2  bicycle['월'] = dates.str.get(1)
3  bicycle['일'] = dates.str.get(2)
4  print(bicycle.head())
```

```
0    [2019, 01, 01]
1    [2019, 01, 01]
2    [2019, 01, 01]
3    [2019, 01, 01]
4    [2019, 01, 01]
Name: 대여일자, dtype: object
        대여일자  대여소번호        대여소  대여구분코드 성별     연
0  2019-01-01       3      중랑센터  일일(회원)  M  AGE_003     12
1  2019-01-01       3      중랑센터  일일(회원)  M  AGE_004      8
2  2019-01-01       3      중랑센터  일일(회원)  M  AGE_005     27   4
3  2019-01-01       3      중랑센터       정기  M  AGE_003     17   34
4  2019-01-01       5  상암센터 정비실  일일(회원)  M  AGE_004      1

   이동거리(M)  이동시간(분)    연   월   일
0       9690       117  2019  01  01
1      14310       151  2019  01  01
2     177910       542  2019  01  01
3     144730       593  2019  01  01
4       3090        33  2019  01  01
```

여기선 날짜 데이터를 문자로 바꾼 후 만들어진 시리즈에 str.get( ) 을 이용해서 새로운 열을 데이터프레임에 추가

(시계열 데이터 탐색적 자료 분석 파트에서 datetime자료형 분리하는 법도 봤음!)

필터링 (불린 인덱싱)

데이터프레임의 각 열은 시리즈 객체입니다!

조건식을 적용하여 불린 시리즈를 만들고
데이터프레임에 대입해 조건을 만족하는
행들만 선택

다시 타이타닉 데이터입니다!

```
#승객 중 60세 이상인 데이터
tdf_60 = tdf[tdf['age']>=60]
print(tdf_60.head())
print(tdf_60[['age','alive']].head())

#loc이용
print(tdf.loc[tdf['age']>=60, ['age','alive']].head())
```

```
#조합
cond1=tdf['age']>60
cond2=tdf['pclass']==1
cond3=tdf['sex']=='female'
print(tdf[cond1&cond2&cond3])
```

Isin() 메소드 활용

DataFrame의 열 객체.isin(추출 값의 리스트)

```
#타이타닉 데이터에서 나이가 24, 25, 26인 사람만 추출하기

a1 = tdf['age']==24
a2 = tdf['age']==25
a3 = tdf['age']==26
print(tdf[a1|a2|a3].head())

print(tdf[tdf['age'].isin([24,25,26])])
```

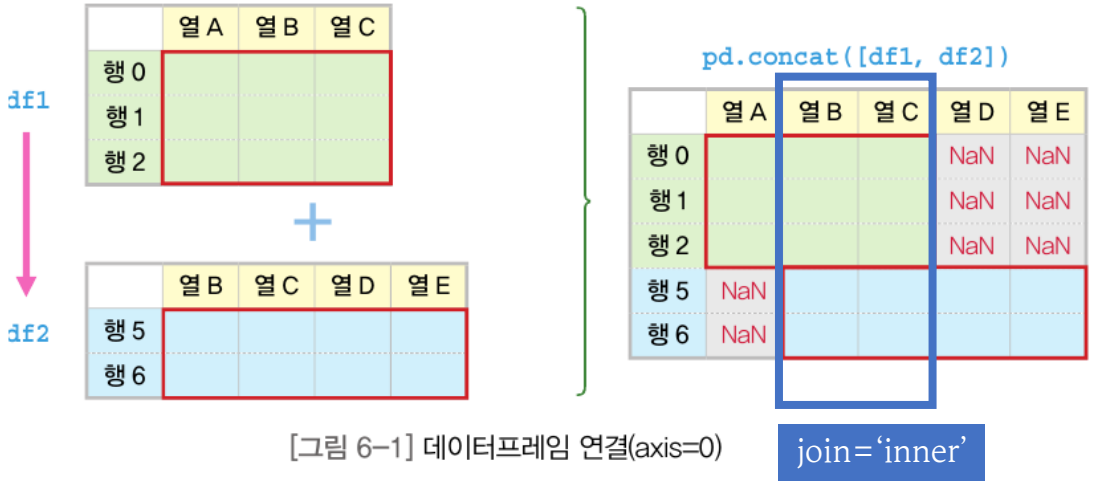
데이터프레임 합치기

연결 concat

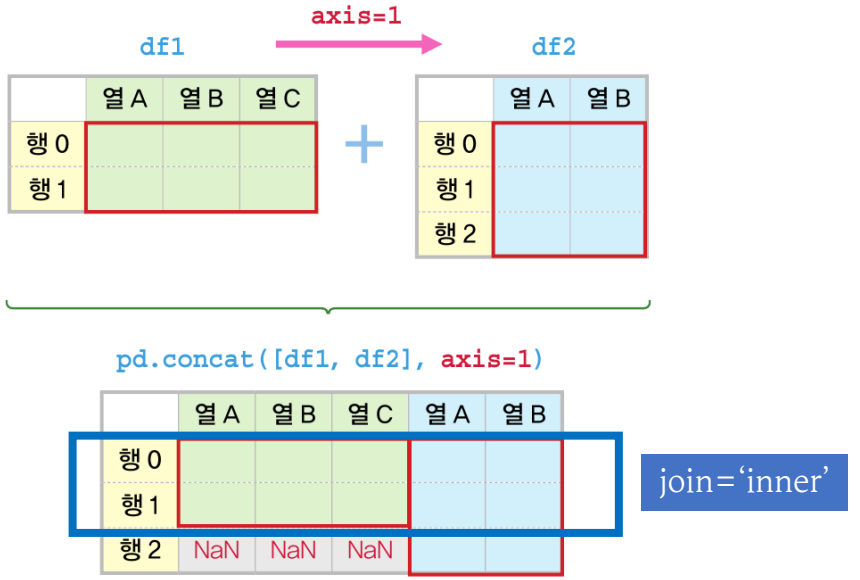
pd.concat([데이터프레임1, 데이터프레임2])

기본 설정
axis = 0 (위아래로 연결)
join = 'outer' (합집합)

추가 설정
axis = 1 (좌우로 연결)
join = 'inner' (교집합)



[그림 6-1] 데이터프레임 연결(axis=0)



[그림 6-2] 데이터프레임 연결 (axis=1)

데이터프레임 합치기

```
tdf1 = tdf.iloc[:5,:2]
tdf2 = tdf.iloc[3:7,1:3]
print(tdf1)
print(tdf2)
s1 = pd.concat([tdf1, tdf2])
print(s1)

s2 = pd.concat([tdf1, tdf2], ignore_index =True) #기존 행 인덱스 무시하고 index 새롭게 설정
print(s2)

s3 = pd.concat([tdf1, tdf2], join='inner')
print(s3)
```

```
   survived  pclass
0         0       3
1         1       1
2         1       3
3         1       1
4         0       3
   pclass     sex
3       1  female
4       3    male
5       3    male
6       1    male
```

s1

	survived	pclass	sex
0	0.0	3	NaN
1	1.0	1	NaN
2	1.0	3	NaN
3	1.0	1	NaN
4	0.0	3	NaN
3	NaN	1	female
4	NaN	3	male
5	NaN	3	male
6	NaN	1	male

s2

	survived	pclass	sex
0	0.0	3	NaN
1	1.0	1	NaN
2	1.0	3	NaN
3	1.0	1	NaN
4	0.0	3	NaN
5	NaN	1	female
6	NaN	3	male
7	NaN	3	male
8	NaN	1	male

s3

	pclass
0	3
1	1
2	3
3	1
4	3
3	1
4	3
5	3
6	1

데이터프레임 합치기

```
s4 = pd.concat([tdf1, tdf2],axis=1)
print(s4)

s5 = pd.concat([tdf1, tdf2],axis=1 ,join='inner')
print(s5)

sr_age = tdf.iloc[:7,3]
s6 = pd.concat([tdf2,sr_age],axis=1)
print(s6)
```

s4

	survived	pclass	pclass	sex
0	0.0	3.0	NaN	NaN
1	1.0	1.0	NaN	NaN
2	1.0	3.0	NaN	NaN
3	1.0	1.0	1.0	female
4	0.0	3.0	3.0	male
5	NaN	NaN	3.0	male
6	NaN	NaN	1.0	male

s5

	survived	pclass	pclass	sex
3	1	1	1	female
4	0	3	3	male

s6

	pclass	sex	age
0	NaN	NaN	22.0
1	NaN	NaN	38.0
2	NaN	NaN	26.0
3	1.0	female	35.0
4	3.0	male	35.0
5	3.0	male	NaN
6	1.0	male	54.0

데이터프레임 합치기

병합 merge

pd.merge(데이터프레임1, 데이터프레임2)

기본 설정

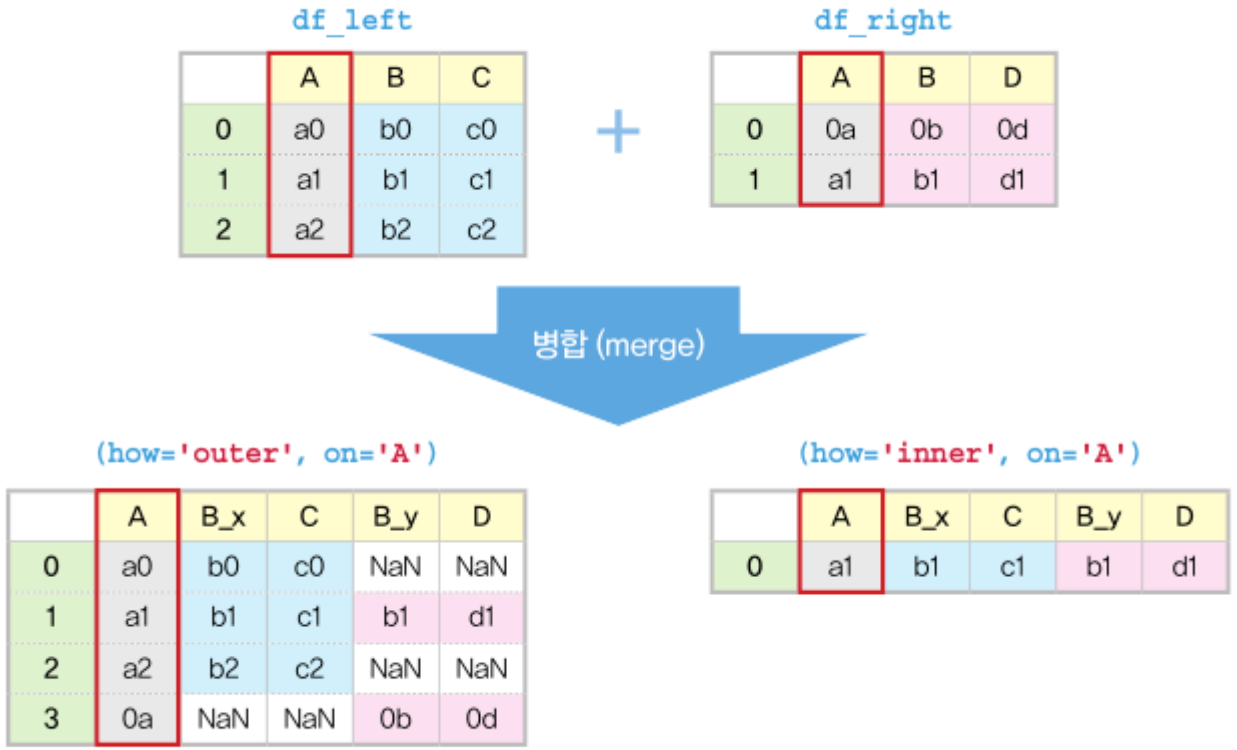
on = None (공통인 모든 열을 키로 인식)

how = 'inner' (기준 열의 데이터가 양쪽에 공통으로 존재할 때만 추출)

추가 설정

on = '특정 열' (특정 열을 기준으로 병합되며 공통되는 이름의 열 존재시 오른쪽 예시처럼 B_x B_y 이름으로 자동 변경)

how = 'outer' (기준이 되는 열의 데이터가 어느 한쪽에만 속하더라도 포함)



[그림 6-3] 데이터프레임 합치기(merge)

데이터프레임 합치기

목표! patient_df와 route_df를 patient_id열을 기준으로 데이터가 공통으로 존재하는 교집합일 경우들을 추출해 병합

잘못된 사례, 왜 잘못됐을까요?

```
mdf = pd.concat([patient_df, route_df], join='inner', axis=1)
mdf
```

2주차 복습과제의 코로나 데이터 이용

2개의 csv파일 이용
PatientInfo.csv – 확진자 정보
PatientRoute.csv – 확진자의 이동경로

	patient_id	global_num	sex	birth_year	age	country	province	city	disease	infection_case	...	deceased_date	state	patient_id
0	1000000001	2.0	male	1964.0	50s	Korea	Seoul	Gangseo-gu	NaN	overseas inflow	...	NaN	released	1000000002
1	1000000002	5.0	male	1987.0	30s	Korea	Seoul	Jungnang-gu	NaN	overseas inflow	...	NaN	released	1000000002
2	1000000003	6.0	male	1964.0	50s	Korea	Seoul	Jongno-gu	NaN	contact with patient	...	NaN	released	1000000002
3	1000000004	7.0	male	1991.0	20s	Korea	Seoul	Mapo-gu	NaN	overseas inflow	...	NaN	released	1000000002
4	1000000005	9.0	female	1992.0	20s	Korea	Seoul	Seongbuk-gu	NaN	contact with patient	...	NaN	released	1000000002
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
63	6001000554	NaN	female	1959.0	60s	Korea	Gyeongsangbuk-do	Gyeongsan-si	NaN	etc	...	NaN	isolated	1000000432
64	6001000555	NaN	female	1945.0	70s	Korea	Gyeongsangbuk-do	Gyeongsan-si	NaN	etc	...	NaN	isolated	1000000433

데이터프레임 합치기

잘못된 사례, 왜 잘못됐을까요?

```
mdf3 = pd.merge(patient_df, route_df)
mdf3
```

	patient_id	global_num	sex	birth_year	age	country	province	city	disease	infection_case	...	contac
0	1000000003	6.0	male	1964.0	50s	Korea	Seoul	Jongno-gu	NaN	contact with patient	...	
1	1000000003	6.0	male	1964.0	50s	Korea	Seoul	Jongno-gu	NaN	contact with patient	...	
2	1000000005	9.0	female	1992.0	20s	Korea	Seoul	Seongbuk-gu	NaN	contact with patient	...	
3	1000000006	10.0	female	1966.0	50s	Korea	Seoul	Jongno-gu	NaN	contact with patient	...	
4	1000000007	11.0	male	1995.0	20s	Korea	Seoul	Jongno-gu	NaN	contact with patient	...	
...	...	...	...	...	...	...	...	...	...	...	...	
123	1000000423	NaN	female	1966.0	50s	Korea	Seoul	Geumcheon-gu	NaN	etc	...	
124	1000000423	NaN	female	1966.0	50s	Korea	Seoul	Geumcheon-gu	NaN	etc	...	
125	1000000424	NaN	female	1971.0	40s	Korea	Seoul	Geumcheon-gu	NaN	etc	...	
126	1000000424	NaN	female	1971.0	40s	Korea	Seoul	Geumcheon-gu	NaN	etc	...	
127	1000000433	9627.0	male	1989.0	30s	Korea	Seoul	Seodaemun-gu	NaN	overseas inflow	...	

128 rows × 22 columns

on=None 으로 적용되기 때문에 공통된 모든 열을 키로 인식

즉, 여기선 확진자 id뿐만 아니라 확진자 본인의 주거 도,시와 이동경로의 도,시가 일치하는 값들을 출력한 것

데이터프레임 합치기

정답!

```
mdf2=pd.merge(patient_df, route_df,on='patient_id')
mdf2
```

확진자의 고유 정보

해당 확진자의 이동 경로

province_x	city_x	disease	infection_case	...	released_date	deceased_date	state	global_num_y	date	province_y	city_y
Seoul	Jungnang-gu	NaN	overseas inflow	...	2020-03-02	NaN	released	5.0	2020-01-26	Seoul	Gwangjin-gu
Seoul	Jungnang-gu	NaN	overseas inflow	...	2020-03-02	NaN	released	5.0	2020-01-27	Seoul	Gangbuk-gu
Seoul	Jungnang-gu	NaN	overseas inflow	...	2020-03-02	NaN	released	5.0	2020-01-28	Seoul	Gangbuk-gu
Seoul	Jungnang-gu	NaN	overseas inflow	...	2020-03-02	NaN	released	5.0	2020-01-29	Seoul	Seongbuk-gu
Seoul	Jungnang-gu	NaN	overseas inflow	...	2020-03-02	NaN	released	5.0	2020-01-30	Seoul	Seongbuk-gu
...	...	...	...	...	...	...	...	...	...	...	...
Seoul	Songpa-gu	NaN	overseas inflow	...	NaN	NaN	isolated	9599.0	2020-03-29	Seoul	Seongbuk-gu
Seoul	Seodaemun-gu	NaN	overseas inflow	...	NaN	NaN	isolated	9627.0	2020-03-27	Incheon	Jung-gu

데이터프레임 합치기

연결을 위한 또 하나의 메소드 join( )

join은 행 인덱스를 기준으로 결합

join()의 디폴트 how= ‘left’ 오른쪽 데이터는 왼쪽 데이터의 인덱스와 부합해야 출력

how=‘inner’ 이용 시 교집합을 출력

```
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'],'h':['h1','h2','h3']}, index=[1,2,3])
df3=df1.join(df2)
print(df3)
print('₩n')

df4 = df1.join(df2, how='inner')
print(df4)
```

```
    a   b   c    e    f    g    h
0  a0  b0  c0  NaN  NaN  NaN  NaN
1  a1  b1  c1   e1   f1   g1   h1
2  a2  b2  c2   e2   f2   g2   h2


    a   b   c   e   f   g   h
1  a1  b1  c1  e1  f1  g1  h1
2  a2  b2  c2  e2  f2  g2  h2
```

그룹연산

groupby([‘a열’])

2개 이상의 기준을 세울 경우: groupby([‘a열’ , ‘b열’ , ‘c열’])
이런 경우 인덱스에 일종의 계층이 만들어짐(멀티 인덱스)

데이터를 특정 기준(열)에 따라 여러 그룹으로 분할하여 처리

분할을 한 후 다양한 집계 기능 함수 mean() max() sum() count() var() 등을 사용할 수도 있고 필요에 따라 직접 만든 함수를 사용할 수도 있다.

멀티 인덱스란?
- 행이나 열에 여러 계층을 가지는 인덱스.

```python
import pandas as pd
import numpy as np
np.random.seed(0)
df_multi = pd.DataFrame(np.random.randint(100, size=(6, 4)),
                        columns=[["mid", "mid", "final", "final"],
                                 ["math", "literature", "math", "literature"]],
                        index=[[2019,2019,2019,2020,2020,2020],
                               ['id_'+str(i+1) for i in range(3)]*2])
df_multi.columns.names = ["exam", "subject"]
df_multi.index.names = ["year", "student_id"]
df_multi

```

	exam	mid		final	
	subject	math	literature	math	literature
year	student_id				
2019	id_1	44	47	64	67
	id_2	67	9	83	21
	id_3	36	87	70	88
2020	id_1	88	12	58	65
	id_2	39	87	46	88
	id_3	81	37	25	77

판다스 멀티인덱스 세부사항
https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#advanced-mi-slicers

그룹연산

3가지 종류의 class로 나뉘는 것을 확인

```
#타이타닉 데이터 사용
#groupby

sdf = tdf.loc[:,['age','sex','class','fare','survived']] #일부만 뜯어오기
sdf_g = sdf.groupby(['class'])
print(type(sdf_g))

#groupyby 뜯어보기
for key, group in sdf_g:
    print('key:', key)
    print('number:', len(group))
    print(group.head())
    print('₩n')
```

```
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
key: First
number: 216
     age     sex  class     fare  survived
1   38.0  female  First  71.2833         1
3   35.0  female  First  53.1000         1
6   54.0    male  First  51.8625         0
11  58.0  female  First  26.5500         1
23  28.0    male  First  35.5000         1


key: Second
number: 184
     age     sex   class     fare  survived
9   14.0  female  Second  30.0708         1
15  55.0  female  Second  16.0000         1
17   NaN    male  Second  13.0000         1
20  35.0    male  Second  26.0000         0
21  34.0    male  Second  13.0000         1


key: Third
number: 491
    age     sex  class     fare  survived
0  22.0    male  Third   7.2500         0
2  26.0  female  Third   7.9250         1
4  35.0    male  Third   8.0500         0
5   NaN    male  Third   8.4583         0
7   2.0    male  Third  21.0750         0
```

그룹연산

get_group( ) 을 이용해서 원하는 그룹의 데이터 추출

```
#1등석 승객 데이터 그룹만 따로 추출
print(sdf_g.get_group('First'))
print('₩n')
```

```
      age     sex  class     fare  survived
1    38.0  female  First  71.2833         1
3    35.0  female  First  53.1000         1
6    54.0    male  First  51.8625         0
11   58.0  female  First  26.5500         1
23   28.0    male  First  35.5000         1
..    ...     ...    ...      ...       ...
871  47.0  female  First  52.5542         1
872  33.0    male  First   5.0000         0
879  56.0  female  First  83.1583         1
887  19.0  female  First  30.0000         1
889  26.0    male  First  30.0000         1

[216 rows x 5 columns]
```

그룹별 평균

```
print(sdf_g.mean())
print('₩n')
```

```
              age       fare  survived
class
First   38.233441  84.154687  0.629630
Second  29.877630  20.662183  0.472826
Third   25.140620  13.675550  0.242363
```

그룹연산

멀티 인덱스 - 6개의 조합이 생성

```
#여러 열을 기준으로 그룹화
sdf_g2 = sdf.groupby(['class','sex'])
print(sdf_g2.mean())
print('₩n')

#조합 중 하나 추출
print('일등석 남성 데이터')
print(sdf_g2.get_group(('First','male')).head())
print('₩n')
```

```
                     age        fare  survived
class  sex
First  female  34.611765  106.125798  0.968085
       male    41.281386   67.226127  0.368852
Second female  28.722973   21.970121  0.921053
       male    30.740707   19.741782  0.157407
Third  female  21.750000   16.118810  0.500000
       male    26.507589   12.661633  0.135447


일등석 남성 데이터
     age   sex  class      fare  survived
6   54.0  male  First   51.8625         0
23  28.0  male  First   35.5000         1
27  19.0  male  First  263.0000         0
30  40.0  male  First   27.7208         0
34  28.0  male  First   82.1708         0
```

```
mul_df = sdf_g2.mean() #멀티인덱스를 가진 데이터프레임
print(mul_df)

#loc
print(mul_df.loc['First'])
print('₩n')
print(mul_df.loc[['First','Third']])
print('₩n')
print(mul_df.loc[('First','male')])
print('₩n')
print(mul_df.loc[('First','male'),'age'])
print('₩n')
```

```
              age        fare  survived
sex
female  34.611765  106.125798  0.968085
male    41.281386   67.226127  0.368852


                     age        fare  survived
class sex
First female  34.611765  106.125798  0.968085
      male    41.281386   67.226127  0.368852
Third female  21.750000   16.118810  0.500000
      male    26.507589   12.661633  0.135447


age         41.281386
fare        67.226127
survived     0.368852
Name: (First, male), dtype: float64


41.28138613861386
```

그룹연산

```
print(mul_df['fare'])
print(mul_df['fare']['First'])
print(mul_df['fare']['First']['male'])
```

```
class   sex
First   female    106.125798
        male       67.226127
Second  female     21.970121
        male       19.741782
Third   female     16.118810
        male       12.661633
Name: fare, dtype: float64
sex
female    106.125798
male       67.226127
Name: fare, dtype: float64
67.22612704918033
```

```
#iloc
print(mul_df.iloc[0])
print('₩n')
print(mul_df.iloc[[1,4]])
print('₩n')
print(mul_df.iloc[:,2])
print('₩n')
print(mul_df.iloc[5,0])
print('₩n')
print(mul_df.iloc[[0,2,4],[0,2]])
```

```
age          34.611765
fare        106.125798
survived      0.968085
Name: (First, female), dtype: float64


                     age       fare  survived
class sex
First male     41.281386  67.226127  0.368852
Third female   21.750000  16.118810  0.500000


class   sex
First   female    0.968085
        male      0.368852
Second  female    0.921053
        male      0.157407
Third   female    0.500000
        male      0.135447
Name: survived, dtype: float64


26.507588932806325


                      age  survived
class  sex
First  female  34.611765  0.968085
Second female  28.722973  0.921053
Third  female  21.750000  0.500000
```

멀티인덱스 인덱싱과 슬라이싱을 위해
xs() 메소드도 활용 가능 (피벗 테이블 파트에서 설명!)

그룹연산

```
#groupby에서 transform을 이용해 age의 z-score 값 계산
def zs(x):
    return (x-x.mean())/x.std()

print(sdf_g.age.transform(zs)) #반환은 시리즈 형태로
print('₩n')
print(sdf_g.age.apply(zs)) #apply 메소드도 가능
print('₩n')

#age열의 평균값이 30보다 작은 그룹 판별
age_f = sdf_g.apply(lambda x: x.age.mean() <30)
print(age_f)

for i in age_f.index:
    if age_f[i] == True:
        print(sdf_g.get_group(i).head())
        print('₩n')
```

```
0     -0.251342
1     -0.015770
2      0.068776
3     -0.218434
4      0.789041
         ...
886   -0.205529
887   -1.299306
888         NaN
889   -0.826424
890    0.548953
Name: age, Length: 891, dtype: float64

class
First     False
Second     True
Third      True
dtype: bool
     age     sex   class     fare  survived
9   14.0  female  Second  30.0708         1
15  55.0  female  Second  16.0000         1
17   NaN    male  Second  13.0000         1
20  35.0    male  Second  26.0000         0
21  34.0    male  Second  13.0000         1


    age     sex  class     fare  survived
0  22.0    male  Third   7.2500         0
2  26.0  female  Third   7.9250         1
4  35.0    male  Third   8.0500         0
5   NaN    male  Third   8.4583         0
7   2.0    male  Third  21.0750         0
```

피벗 테이블

- 피벗 테이블이란?
- 여러 데이터 중에서 자신이 원하는 데이터만을 가지고 원하는 행과 열에 데이터를 배치하여 새로운 보고서를 만드는 기능

	date	typecode	volume
0	20180901	A	10
1	20180901	B	100
2	20180901	C	1000
3	20180902	A	20
4	20180902	B	200
5	20180902	C	2000
6	20180903	A	30
7	20180903	B	300
8	20180903	C	3000

	date	A	B	C
0	20180901	10	100	1000
1	20180902	20	200	2000
2	20180903	30	300	3000

```
import pandas as pd
import seaborn as sns

titanic = sns.load_dataset('titanic')
titanic = titanic.loc[:, ['age', 'sex', 'class', 'fare', 'survived']]
print(titanic.head())
```

```
    age     sex  class     fare  survived
0  22.0    male  Third   7.2500         0
1  38.0  female  First  71.2833         1
2  26.0  female  Third   7.9250         1
3  35.0  female  First  53.1000         1
4  35.0    male  Third   8.0500         0
```

피벗테이블 생성

pd.pivot_table(데이터프레임
index = 행으로 사용할 열
columns = 열로 사용할 열
value = 데이터로 사용할 열
aggfun = 데이터 집계 함수)

```
pdf1 = pd.pivot_table(titanic,                  #피벗할 데이터프레임
                      index = 'class',          #행 위치에 들어갈 열
                      columns = 'sex',          #열 위치에 들어갈 열
                      values ='age',            #데이터로 사용할 열
                      aggfunc = 'mean')         #데이터 집계함수
print(pdf1.head())
```

```
sex         female       male
class
First    34.611765  41.281386
Second   28.722973  30.740707
Third    21.750000  26.507589
```

```
pdf2 = pd.pivot_table(titanic,
                      index = 'class',
                      columns = 'sex',
                      values ='survived',
                      aggfunc = ['mean', 'sum'])
print(pdf2.head())
```

```
            mean                sum
sex       female      male female male
class
First   0.968085  0.368852     91   45
Second  0.921053  0.157407     70   17
Third   0.500000  0.135447     72   47
```

복수 데이터 집계함수 사용 가능

aggfunc = [‘함수1’, ‘함수2’, …. , ‘함수N’]

피벗테이블 생성

```
pdf3 = pd.pivot_table(titanic,
                      index = ['class','sex'],
                      columns = 'survived',
                      values = ['age','fare'],
                      aggfunc = ['mean', 'max'])
print(pdf3.head())
print('₩n')

#행, 열 구조 살펴보기
print(pdf3.index)
print(pdf3.columns)
```

		mean				max			
		age		fare		age		fare	
survived		0	1	0	1	0	1	0	1
class	sex								
First	female	25.666667	34.939024	110.604167	105.978159	50.0	63.0	151.55	512.3292
	male	44.581967	36.248000	62.894910	74.637320	71.0	80.0	263.00	512.3292
Second	female	36.000000	28.080882	18.250000	22.288989	57.0	55.0	26.00	65.0000
	male	33.369048	16.022000	19.488965	21.095100	70.0	62.0	73.50	39.0000
Third	female	23.818182	19.329787	19.773093	12.464526	48.0	63.0	69.55	31.3875

- 복수 행, 열 선택 가능
- 복수 데이터 선택 가능
- 복수 데이터 집계 함수 선택 가능

행 구조 살펴보기 -> 데이터프레임 객체.index

열 구조 살펴보기 -> 데이터프레임 객체.columns

피벗 – xs 인덱서(행 선택)

xs 인덱서란?

- 데이터 프레임의 멀티 인덱싱 방법 중 하나
- 데이터프레임 객체.xs(aixs =0)이 디폴트 값
- xs 인덱서 사용시 특정 레벨 or 특정 축만 가능하고 동시 선택은 불가함

<행 인덱스의 level 1개 추출>

데이터프레임 객체.xs(행 인덱스)

```
# 축 값은 axis = 0으로 자동 설정됨
# 행 인덱스가 First인 행을 선택
print(pdf3.xs('First'))
```

```
              mean                                              max                 \
               age                 fare                         age          fare   
survived         0          1          0           1            0     1         0   
sex                                                                                 
female   25.666667  34.939024  110.604167  105.978159        50.0  63.0    151.55   
male     44.581967  36.248000   62.894910   74.637320        71.0  80.0    263.00   


survived         1  
sex                 
female    512.3292  
male      512.3292  
```

<행 인덱스의 level 2개 이상 추출>

데이터프레임 객체.xs((행1, 행2, … , 행N))

```
#행 인덱스가 ('First', 'female')인 행을 선택
print(pdf3.xs(('First','female')))
```

```
           survived
mean  age  0            25.666667
           1            34.939024
      fare 0           110.604167
           1           105.978159
max   age  0            50.000000
           1            63.000000
      fare 0           151.550000
           1           512.329200
Name: (First, female), dtype: float64
```

피벗 - xs 인덱서(행 선택)

<level 지정>

데이터프레임 객체.xs(행 인덱스, level = 레벨 인덱스)

```
#행 인덱스의 sex 레벨이 male인 행을 선택
print(pdf3.xs('male', level = 'sex'))
```

	mean				max			₩
	age		fare		age		fare	
survived	0	1	0	1	0	1	0	
class								
First	44.581967	36.248000	62.894910	74.637320	71.0	80.0	263.00	
Second	33.369048	16.022000	19.488965	21.095100	70.0	62.0	73.50	
Third	27.255814	22.274211	12.204469	15.579696	74.0	45.0	69.55	

survived	1
class	
First	512.3292
Second	39.0000
Third	56.4958

- 문자로 Level 지정

```
#행 인덱스의 레벨이 Second, male인 행을 선택
print(pdf3.xs(('Second', 'male'), level = ['class','sex']))
```

	mean				max			
	age		fare		age		fare	
survived	0	1	0	1	0	1	0	1
class sex								
Second male	33.369048	16.022	19.488965	21.0951	70.0	62.0	73.5	39.0

- 숫자로 Level 지정

```
#행 인덱스의 레벨이 Second, male인 행을 선택
print(pdf3.xs(('Second', 'male'), level = [0,1]))
```

	mean				max			
	age		fare		age		fare	
survived	0	1	0	1	0	1	0	1
class sex								
Second male	33.369048	16.022	19.488965	21.0951	70.0	62.0	73.5	39.0

문자형 level, 숫자형 level 모두 동일한 결과 return

피벗 - xs 인덱서(열 선택)

```
#열 선택시 axis = 1 설정 필요
#열 인덱스가 mean인 데이터를 선택
print(pdf3.xs('mean', axis = 1))
```

```
                    age                 fare
survived              0         1         0          1
class  sex
First  female  25.666667  34.939024  110.604167  105.978159
       male    44.581967  36.248000   62.894910   74.637320
Second female  36.000000  28.080882   18.250000   22.288989
       male    33.369048  16.022000   19.488965   21.095100
Third  female  23.818182  19.329787   19.773093   12.464526
       male    27.255814  22.274211   12.204469   15.579696
```

- 데이터프레임 객체.xs(열 인덱스, axis = 1)
- 행 선택과 달리 axis = 1 옵션 꼭 필요

```
#열 인덱스가 ('mean','age')인 데이터를 선택
print(pdf3.xs(('mean', 'age'), axis = 1))
```

```
survived              0         1
class  sex
First  female  25.666667  34.939024
       male    44.581967  36.248000
Second female  36.000000  28.080882
       male    33.369048  16.022000
Third  female  23.818182  19.329787
       male    27.255814  22.274211
```

- 2개 이상의 열 추출 가능
- 데이터프레임 객체.xs((열1,열2 , … ,열N), axis =1))

피벗 - xs 인덱서(열 선택)

<level 지정>

데이터프레임 객체.xs
(열 인덱스, level =레벨 인덱스, axis = 1)

- 문자로 level 지정

```
#survived 레벨이 1인 데이터 선택
print(pdf3.xs(1, level = 'survived', axis = 1))
```

		mean		max	
		age	fare	age	fare
class	sex				
First	female	34.939024	105.978159	63.0	512.3292
	male	36.248000	74.637320	80.0	512.3292
Second	female	28.080882	22.288989	55.0	65.0000
	male	16.022000	21.095100	62.0	39.0000
Third	female	19.329787	12.464526	63.0	31.3875
	male	22.274211	15.579696	45.0	56.4958

- 숫자로 level 지정

```
#survived 레벨이 1인 데이터 선택
print(pdf3.xs(1, level = 2, axis = 1))
```

		mean		max	
		age	fare	age	fare
class	sex				
First	female	34.939024	105.978159	63.0	512.3292
	male	36.248000	74.637320	80.0	512.3292
Second	female	28.080882	22.288989	55.0	65.0000
	male	16.022000	21.095100	62.0	39.0000
Third	female	19.329787	12.464526	63.0	31.3875
	male	22.274211	15.579696	45.0	56.4958

```
#max, fare, survived = 0인 데이터 선택
print(pdf3.xs(('max', 'fare', 0), level = [0,1,2], axis =1))
```

		max
		fare
survived		0
class	sex	
First	female	151.55
	male	263.00
Second	female	26.00
	male	73.50
Third	female	69.55
	male	69.55

Q & A

데이터 프레임의 다양한 응용 및 판다스 복습

- 끝 -