데이터 처리 프로그래밍

박소현

네 번째 주제:

데이터프레임과 시리즈 다루기-기초2

1. 외부데이터 가져오기

```
import pandas as pd
```

```
import seaborn as sns
```

```
df = sns.load_dataset("iris")
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- seaborn의 내장데이터 iris 가져오기
- sns.load_dataset( ) 이용

```
df.shape
```

```
(150, 5)
```

```
df.size
```

```
750
```

```
df.head(2)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa

전체 데이터 몇 행 몇 열짜리? 전체 데이터 개수?

```
df[['sepal_width','sepal_length','species']]
```

```
columns = ['sepal_width','sepal_length','species']
df[columns]
```

```
df['sepal234'] = df["sepal_length"] + df["sepal_width"]
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9
2	4.7	3.2	1.3	0.2	setosa	7.9
3	4.6	3.1	1.5	0.2	setosa	7.7
4	5.0	3.6	1.4	0.2	setosa	8.6

2. 정규 표현식을 활용한 열 검색

- 변수명.filter(regex = '정규표현식') 활용
- 특정 열을 가져옴(axis = 0인 경우 특정 행 추출)
  - regex = ' ' 옵션으로 정규표현식 이용 가능
- 정규표현식

df.filter(regex='.p') p앞에 문자 있는 것	₩d 숫자, [0-9]도 숫자 의미
df.filter(regex='length$') length로 끝나는 것	₩D 문자, [^0-9]도 문자 의미
df.filter(regex='^sepal') sepal로 시작하는 것	₩s 공백이나 탭
df.filter(regex='l₩d+') l(엘)뒤에 숫자 있는 것	₩S 공백 문자 이외의 문자
df.filter(regex='width?') width이 있는 것	₩w 알파벳
df.filter(regex='l₩d{3}') l뒤에 숫자 3개(이상) 있는 것	₩W 알파벳 이 외의 문자(한글..)
df.filter(regex='sepal' '|' 'width') sepal 이나 width 있는 것	₩ 이스케이프문자 (. * ? + 등의 앞에)

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름에 언더바 _ 있는 것

```
df.filter(regex='_')
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름이 p앞에 문자 있는 것

```
df.filter(regex='.p').head(2)
```

	sepal_length	sepal_width	species	sepal234
0	5.1	3.5	setosa	8.6
1	4.9	3.0	setosa	7.9

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름이 length로 끝나는 것

```
df.filter(regex='length$').head(2)
```

	sepal_length	petal_length
0	5.1	1.4
1	4.9	1.4

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름이 sepal로 시작하는 것

```
df.filter(regex='^sepal').head(2)
```

	sepal_length	sepal_width	sepal234
0	5.1	3.5	8.6
1	4.9	3.0	7.9

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름 l(엘)뒤에 숫자 있는 것

```
df.filter(regex='l\d+').head(2)
```

	sepal234
0	8.6
1	7.9

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름에 width 있는 것

```
df.filter(regex='width?').head(2)
```

	sepal_width	petal_width
0	3.5	0.2
1	3.0	0.2

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름이 l(엘)뒤에 숫자 3개(이상) 있는 것

```
df.filter(regex='l\d{3}').head(2)
```

	sepal234
0	8.6
1	7.9

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름에 sepal 이나 width 있는 것
  주의: and 연산은 안됨

```
df.filter(regex='sepal' '|' 'width').head(2)
```

	sepal_length	sepal_width	petal_width	sepal234
0	5.1	3.5	0.2	8.6
1	4.9	3.0	0.2	7.9

2. 정규 표현식을 활용한 열 검색 df.filter( )

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

```
df.filter(regex='pet' '|' 'length').head(2)
```

	sepal_length	petal_length	petal_width
0	5.1	1.4	0.2
1	4.9	1.4	0.2

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름이 l(엘) 뒤에 숫자 있는 것

```
df.filter(regex='l₩d')
```

```
df.filter(regex='l[0-9]').head(2)
```

	sepal234
0	8.6
1	7.9

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름이 l(엘) 뒤에 문자 있는 것

```
df.filter(regex='l[^0-9]').head(2)
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

- 열 이름이 문자로 끝나는 것

```
df.filter(regex='#D$').head(2)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa

2. 정규 표현식을 활용한 열 검색 df.filter( )

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

```
df.loc[2:5,'sepal_width':'petal_width']
```

	sepal_width	petal_length	petal_width
2	3.2	1.3	0.2
3	3.1	1.5	0.2
4	3.6	1.4	0.2
5	3.9	1.7	0.4

```
df.iloc[-3:,[1,2,4]]
```

	sepal_width	petal_length	species
147	3.0	5.2	virginica
148	3.4	5.4	virginica
149	3.0	5.1	virginica

2. 정규 표현식을 활용한 열 검색 df.filter( )

	sepal_length	sepal_width	petal_length	petal_width	species	sepal234
0	5.1	3.5	1.4	0.2	setosa	8.6
1	4.9	3.0	1.4	0.2	setosa	7.9

```
df.loc[df['sepal_length'] > 5.5, ['sepal_length','sepal_width']]
```

	sepal_length	sepal_width
14	5.8	4.0
15	5.7	4.4
18	5.7	3.8
50	7.0	3.2
51	6.4	3.2

3. Summarize Data

```
import pandas as pd
```

```
import seaborn as sns
```

```
df = sns.load_dataset("iris")
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

1) .value_counts( )

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa

각각의 값을 그룹화 해서 개수를 보여줌

```
df['species'].value_counts()
```

```
virginica     50
setosa        50
versicolor    50
Name: species, dtype: int64
```

결과를 데이터 프레임으로 따로 보기

```
pd.DataFrame(df['species'].value_counts())
```

	species
virginica	50
setosa	50
versicolor	50

2) len(변수명)

데이터프레임의 행이 몇 개

```
len(df)
```

```
150
```

데이터프레임의 행, 열 개수 알려주던 거?

```
df.shape[0]
```

```
150
```

```
df.shape[1]
```

```
5
```

```
len(df) == df.shape[0]
```

```
True
```

3) .nunique( ) 값의 종류가 몇 개

species 값 종류 몇 개?

	species
virginica	50
setosa	50
versicolor	50

```
df['species'].nunique()
```

```
3
```

```
df['species'].unique()
```

.unique( )는 값의 종류 뭐 있나?

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

4) .describe( )

함수(메서드) 이름	기능
count()	NaN값을 제외한 관측 데이터의 수 반환
min() max() sum()	관측 데이터의 최소값, 최대값, 합, 반환

median() quantile(q=0.5)	관측 데이터의 중간값, 분위수(기본값 0.5)값 반환
mean() std()	관측 데이터의 평균, 표준 편차 반환
var() skew() kurt()	관측 데이터의 분산, 왜도, 첨도 반환
describe()	- 수치 관측 데이터 : count, mean, std, min, 25%, 50%, 75%, max 값 반환 - 비수치 관측 데이터 : count, unique, top, freq 값 반환

4) .describe( )

```
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.82			
min	4.30			
25%	5.10			
50%	5.80			
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa

.describe(include = 'all') 수치, 비수치 모두 보여 줌

```
df.describe(include = 'all')
```

	sepal_length	sepal_width	petal_length	petal_width	species
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	virginica
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.057333	3.758000	1.199333	NaN
std	0.828066	0.435866	1.765298	0.762238	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

.describe(include = [np.object])
비수치 데이터의 요약통계

```
df.describe(include = [np.object])
```

	species
count	150
unique	3
top	virginica
freq	50

.describe(include = [np.number])
수치 데이터의 요약통계

```
df.describe(include = [np.number])
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

4. 결측값 처리

```
df
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

세번째 주제 수업 때 배운 내용 중에..

- 특정값 조회하는 isin( )
  df.A.isin([5])
- null값 조회하는 isnull( )
  pd.isnull(df4), df4.isnull( ),
  df['A'].isnull( ), ~df4.A.notnull( )
- null값 아닌 것 조회하는 notnull( )
  df4.A.notnull( )

1) .dropna( ) 결측값 제거

```
df.dropna?
```

df.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

axis : {0 or 'index', 1 or 'columns'}, default 0

* 0, or 'index' : Drop rows which contain missing values.
* 1, or 'columns' : Drop columns which contain missing value.

how = all : 전부 다 null값일때 drop적용 how = any : 하나라도 null값이 있으면 drop적용

4. 결측값 처리 1)df. dropna( )

```
df
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

```
df.dropna(axis = 1, how = 'all')
```

	A	B	D
0	NaN	2.0	0
1	3.0	4.0	1
2	NaN	NaN	5

```
df.dropna(axis = 0, how = 'all')
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

4. 결측값 처리 1)df. dropna( )

```
df
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

```
df.dropna(axis = 1, how = 'any')
```

	D
0	0
1	1
2	5

```
df.dropna(axis = 0, how = 'any')
```

A	B	C	D

2) .fillna( ) 결측값 채우기

누락 데이터(NaN, NA) 특정 값으로 대체

```
df변수이름.fillna(value=None, method=None, axis=None)
```

- value 대체할 값 (스칼라, dict, Series, DataFrame 등)
- method 누락 데이터 대체 방법 지정 ('backfill', 'bfill', 'pad', 'ffill')
- axis 대상 축 (0 또는 'index', 1 또는 'columns')

```
df
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

```
df.fillna(0)
```

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	0.0	0.0	0.0	5

```
df
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

```
values = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
values
```

```
{'A': 0, 'B': 1, 'C': 2, 'D': 3}
```

```
df.fillna(values)
```

	A	B	C	D
0	0.0	2.0	2.0	0
1	3.0	4.0	2.0	1
2	0.0	1.0	2.0	5

```
df
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

```
df['D'].mean()
```

```
2.0
```

```
df.fillna(df['D'].mean())
```

	A	B	C	D
0	2.0	2.0	2.0	0
1	3.0	4.0	2.0	1
2	2.0	2.0	2.0	5

```
df
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

```
df.fillna(method = 'ffill')
```

앞의 값을 결측값으로 대체

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	3.0	4.0	NaN	5

```
df.fillna(method = 'bfill')
```

뒤의 값을 결측값으로 대체

	A	B	C	D
0	3.0	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5

5. 새로운 열 만들기 – assign( )

```
df = pd.DataFrame({'A' : range(1, 11), 'B' : np.random.randn(10)})
df
```

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

- 세번째 주제 수업 때 배운 내용 중에..

df5['새 열이름'] = df5['pop'] – df5['GDP']
→ df5의 pop열 – GDP열 결과를 새 열로 추가

- A열에 log를 씌운 결과를 D열로 추가하려면

```
df['D'] = np.log(df.A)
df
```

	A	B	D
0	1	-0.862819	0.000000

원본에
바로 적용

```
df = pd.DataFrame({'A' : range(1, 11), 'B' : np.random.randn(10)})
df
```

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

- 변수명.assign( ) : (괄호안)의 결과를 새 열로 추가
  별 다른 옵션이 없는 메서드
- A열에 log를 씌운 결과를 C열로 추가하려면

```
df.assign(C = np.log(df.A))
```

	A	B	C
0	1	-0.862819	0.000000

```
df
```

	A	B
0	1	-0.862819

```
df = df.assign(C = np.log(df.A))
df
```

	A	B	C
0	1	-0.862819	0.000000

6. 값을 구간으로 나누기

- pd.cut( ) pd.qcut( )
- 1차원 연속형 숫자 값을 범주형 값으로 바꿀때 많이 활용

1) pd.cut( )

```
pd.cut?
```

pd.cut(x, bins, right=True, labels=None,

Use `cut` when you need to segment and sort data values into bins. This function is also useful for going from a continuous variable to a categorical variable. For example, `cut` could convert ages to groups of age ranges. Supports binning into an equal number of bins, or a pre-specified array of bins.

1) pd.cut( )

```
pd.cut( x, bins, right=True, labels=None,
```

pd.cut(1차원 데이터, 나눌 구간인 bins, right=True, labels = none)

- 1차원 데이터가 연속의미의 숫자일 때 이들을 특정 구간으로 끊어서 나누어 줌
- 나누고 싶은 구간은 bins 자리에서 지정하면 됨, 단일숫자 or 리스트
- 우측 경계숫자 빼려면 right = False 예) 4~8미만 ,우측 경계값 8 뺌
- 나뉜 애들 범주값 주려면 labels 옵션(구간 개수 맞춰서)

df

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

pd.cut(x, bins, right=True, labels=None,

```
pd.cut(df.A, 3)
```

```
0     (0.991, 4.0]
1     (0.991, 4.0]
2     (0.991, 4.0]
3     (0.991, 4.0]
4       (4.0, 7.0]
5       (4.0, 7.0]
6       (4.0, 7.0]
7     (7.0, 10.0]
8     (7.0, 10.0]
9     (7.0, 10.0]
Name: A, dtype: category
Categories (3, interval[float64]): [(0.991, 4.0] < (4.0, 7.0] < (7.0, 10.0]]
```

```
df['A'].dtypes
```

```
dtype('int64')
```

df의A열 값들을
3 구간(bins)으로
우측 경계값 포함(생략했으니)
별도의 범주값 없이(labels 생략)
cut 즉, 나누어라

df

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

pd.cut(x, bins, right=True, labels=None,

```
pd.cut(df.A, [0, 4, 7, 10])
```

```
0     (0, 4]
1     (0, 4]
2     (0, 4]
3     (0, 4]
4     (4, 7]
5     (4, 7]
6     (4, 7]
7    (7, 10]
8    (7, 10]
9    (7, 10]
Name: A, dtype: category
Categories (3, interval[int64]): [(0, 4] < (4, 7] < (7, 10]]
```

df의A열 값들을
0~4, 4~7, 7~10 구간(bins)으로
우측 경계값 포함(생략했으니)
별도의 범주값 없이(labels 생략)
cut 즉, 나누어라

pd.cut(x, bins, right=True, labels=None,

df

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

```
pd.cut(df.A, [0, 4, 7, 10], right = False)
```

```
0      [0.0, 4.0)
1      [0.0, 4.0)
2      [0.0, 4.0)
3      [4.0, 7.0)
4      [4.0, 7.0)
5      [4.0, 7.0)
6     [7.0, 10.0)
7     [7.0, 10.0)
8     [7.0, 10.0)
9             NaN
Name: A, dtype: category
Categories (3, interval[int64]): [[0, 4) < [4, 7) < [7, 10)]
```

df의A열 값들을
0~4, 4~7, 7~10 구간(bins)으로
우측 경계값 빼고
별도의 범주값 없이(labels 생략)
cut 즉, 나누어라

pd.cut(x, bins, right=True, labels=None,

df

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

```
pd.cut(df.A, [0, 4, 7, 10], labels = ['L', 'M', 'H'])
```

```
0    L
1    L
2    L
3    L
4    M
5    M
6    M
7    H
8    H
9    H
Name: A, dtype: category
Categories (3, object): [L < M < H]
```

df의A열 값들을
0~4, 4~7, 7~10 구간(bins)으로
우측 경계값 포함(생략했으니)
범주값 L, M, H로 주고(labels옵션)
cut 즉, 나누어라

pd.cut(x, bins, right=True, labels=None,

df

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

```
pd.cut(df.A, [0, 4, 7, 10], right = False,labels = ['L', 'M', 'H'])
```

```
0      L
1      L
2      L
3      M
4      M
5      M
6      H
7      H
8      H
9    NaN
Name: A, dtype: category
Categories (3, object): [L < M < H]
```

df의A열 값들을
0~4, 4~7, 7~10 구간(bins)으로
우측 경계값 빼고
범주값 L, M, H 주고(labels옵션)
cut 즉, 나누어라

```
df
```

	A	B
0	0	-0.316440
1	1	1.177644
2	3	0.306100
3	5	1.146508
4	7	1.028168
5	11	-1.255035
6	9	-0.564743
7	10	0.474414
8	1	-0.548062
9	2	0.431390

pd.cut(x, bins, right=True, labels=None,

```
pd.cut(df.A, 3)
```

```
0    (-0.011, 3.667]
1    (-0.011, 3.667]
2    (-0.011, 3.667]
3     (3.667, 7.333]
4     (3.667, 7.333]
5      (7.333, 11.0]
6      (7.333, 11.0]
7      (7.333, 11.0]
8    (-0.011, 3.667]
9    (-0.011, 3.667]
Name: A, dtype: category
Categories (3, interval[float64]): [(-0.011, 3.667] < (3.667, 7.333] < (7.333, 11.0]]
```

df의A열 값들을
알아서 3개 구간(bins)으로
우측 경계값 포함,
범주값 없이
나누어라

```
df
```

	A	B
0	0	-0.316440
1	1	1.177644
2	3	0.306100
3	5	1.146508
4	7	1.028168
5	11	-1.255035
6	9	-0.564743
7	10	0.474414
8	1	-0.548062
9	2	0.431390

pd.cut(x, bins, right=True, labels=None,

```
pd.cut(df.A, [0, 4, 7, 10])
```

```
0         NaN
1      (0.0, 4.0]
2      (0.0, 4.0]
3      (4.0, 7.0]
4      (4.0, 7.0]
5         NaN
6     (7.0, 10.0]
7     (7.0, 10.0]
8      (0.0, 4.0]
9      (0.0, 4.0]
[(0, 4] < (4, 7] < (7, 10]]
```

```
.A, [0, 4, 7, 10], right = False)
```

```
0      [0.0, 4.0)
1      [0.0, 4.0)
2      [0.0, 4.0)
3      [4.0, 7.0)
4     [7.0, 10.0)
5         NaN
6     [7.0, 10.0)
7         NaN
8      [0.0, 4.0)
9      [0.0, 4.0)
[[0, 4) < [4, 7) < [7, 10)]
```

2) pd.qcut( )

```
pd.qcut?
```

- pd.qcut(x, q, labels=None, retbins=False, precision=3, duplicates='raise')
- Quantile-based discretization function. Discretize variable into equal-sized buckets based on rank or based on sample quantiles. For example 1000 values for 10 quantiles would produce a Categorical object indicating quantile membership for each data point.
- 구간을 비슷하게 나누고 싶을때(사분위수 기반, 아주 똑같은 개수는 아니나 어느정도 비슷하게, 데이터 많을때)

- cut과 매우 유사, right옵션 없음
- 백분율 기반 개념이라 bins에 해당하는 q가 1을 넘으면 error남

6. 값을 구간으로 나누기– 2) pd.qcut( )

```
df
```

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

```
pd.qcut(df.A, 3)
```

```
0     (0.999, 4.0]
1     (0.999, 4.0]
2     (0.999, 4.0]
3     (0.999, 4.0]
4       (4.0, 7.0]
5       (4.0, 7.0]
6       (4.0, 7.0]
7      (7.0, 10.0]
8      (7.0, 10.0]
9      (7.0, 10.0]
Name: A, dtype: category
Categories (3, interval[float64]): [(0.999, 4.0] < (4.0, 7.0] < (7.0, 10.0]]
```

6. 값을 구간으로 나누기– 2) pd.qcut( )

```
df
```

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

```
pd.qcut(df.A, [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1 ])
```

```
0     (0.999, 1.9]
1       (1.9, 2.8]
2       (2.8, 3.7]
3       (3.7, 4.6]
4       (4.6, 5.5]
5       (5.5, 6.4]
6       (6.4, 7.3]
7       (7.3, 8.2]
8       (8.2, 9.1]
9     (9.1, 10.0]
Name: A, dtype: category
Categories (10, interval[float64]): [(0.999, 1.9] < (1.9, 2.8] <
(2.8, 3.7] < (3.7, 4.6] ... (6.4, 7.3] < (7.3, 8.2] < (8.2, 9.1] <
(9.1, 10.0]]
```

6. 값을 구간으로 나누기– 2) pd.qcut( )

```
df
```

	A	B
0	1	-0.862819
1	2	1.197189
2	3	-0.935074
3	4	-0.503223
4	5	-0.445516
5	6	-0.899583
6	7	-0.699034
7	8	-0.734427
8	9	0.407848
9	10	0.678296

```
pd.qcut(df.A, [0, 0.25, 0.5, 0.75, 1], labels = ['1st', '2nd', '3rd', '4th'])
```

```
0    1st
1    1st
2    1st
3    2nd
4    2nd
5    3rd
6    3rd
7    4th
8    4th
9    4th
Name: A, dtype: category
Categories (4, object): [1st < 2nd < 3rd < 4th]
```

labels 주려면 몇 개 입력해야 하나?

4개

6. 값을 구간으로 나누기– 2) pd.qcut( )

```
df
```

	A	B
0	0	-0.316440
1	1	1.177644
2	3	0.306100
3	5	1.146508
4	7	1.028168
5	11	-1.255035
6	9	-0.564743
7	10	0.474414
8	1	-0.548062
9	2	0.431390

```
pd.cut(df.A, 3)
```

```
0    (-0.011, 3.667]
1    (-0.011, 3.667]
2    (-0.011, 3.667]
3     (3.667, 7.333]
4     (3.667, 7.333]
5      (7.333, 11.0]
6      (7.333, 11.0]
7      (7.333, 11.0]
8    (-0.011, 3.667]
9    (-0.011, 3.667]
```

```
[(-0.011, 3.667] < (3.667, 7.333] < (7.333, 11.0]]
```

```
pd.qcut(df.A, 3)
```

```
0    (-0.001, 2.0]
1    (-0.001, 2.0]
2        (2.0, 7.0]
3        (2.0, 7.0]
4        (2.0, 7.0]
5       (7.0, 11.0]
6       (7.0, 11.0]
7       (7.0, 11.0]
8    (-0.001, 2.0]
9    (-0.001, 2.0]
```

```
[(-0.001, 2.0] < (2.0, 7.0] < (7.0, 11.0]]
```

문제) 다음 결과에서 구간별로 값이 몇 개씩인지 알려 달라 하려면?

```
aaa = pd.cut(df.A, [0, 4, 7, 10])
aaa
```

```
0    (0, 4]
1    (0, 4]
2    (0, 4]
3    (0, 4]
4    (4, 7]
5    (4, 7]
6    (4, 7]
7    (7, 10]
8    (7, 10]
9    (7, 10]
Name: A, dtype: category
```

앞에서 나왔던.. ppt22쪽에 있는!!

value_counts( )

3. Summarize Data 1)df['A'].value_counts( )

1) .value_counts( )

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa

각각의 값을 그룹화 해서 개수를 보여줌

```
df['species'].value_counts()
```

```
virginica     50
setosa        50
versicolor    50
Name: species, dtype: int64
```

결과를 데이터 프레임으로 따로 보기

```
pd.DataFrame(df['species'].value_counts())
```

	species
virginica	50
setosa	50
versicolor	50

문제) 다음 결과에서 구간별로 값이 몇 개씩인지 알려 달라 하려면?

```
aaa = pd.cut(df.A, [0, 4, 7, 10])
aaa
```

```
0     (0, 4]
1     (0, 4]
2     (0, 4]
3     (0, 4]
4     (4, 7]
5     (4, 7]
6     (4, 7]
7    (7, 10]
8    (7, 10]
9    (7, 10]
Name: A, dtype: category
```

```
aaa.value_counts()
```

```
(0, 4]     4
(7, 10]    3
(4, 7]     3
Name: A, dtype: int64
```

문제) 다음 결과에서 구간별로 값이 몇 개씩인지 알려 달라 하려면?

```
bbb = pd.cut(df.A, [0, 4, 7, 10],
             right = False,
             labels = ['L', 'M', 'H'])
bbb
```

```
0      L
1      L
2      L
3      M
4      M
5      M
6      H
7      H
8      H
9    NaN
Name: A, dtype: category
Categories (3, object): [L < M < H]
```

```
bbb.value_counts()
```

```
H    3
M    3
L    3
Name: A, dtype: int64
```

```
bbb.value_counts(ascending = True)
```

```
L    3
M    3
H    3
Name: A, dtype: int64
```

```
df
```

	A	B	C	D
0	1	-0.862819	0.000000	0.000000
1	2	1.197189	0.693147	0.693147
2	3	-0.935074	1.098612	1.098612
3	4	-0.503223	1.386294	1.386294
4	5	-0.445516	1.609438	1.609438
5	6	-0.899583	1.791759	1.791759
6	7	-0.699034	1.945910	1.945910
7	8	-0.734427	2.079442	2.079442
8	9	0.407848	2.197225	2.197225
9	10	0.678296	2.302585	2.302585

```
Parameters
----------
axis : {index (0), columns (1)}
    Axis for the function to be applied on.
```

```
df.max(axis = 1)
```

```
0     1.0
1     2.0
2     3.0
3     4.0
4     5.0
5     6.0
6     7.0
7     8.0
8     9.0
9    10.0
dtype: float64
```

```
df.max(axis = 0)
```

```
A    10.000000
B     1.197189
C     2.302585
D     2.302585
dtype: float64
```

df.max(axis = 1)
각 열들 중에서 최대값 인데..
세로 기준이 아니고
가로 비교한 결과..

8. 값을 제한하기

1) 임계치 만큼 값을 다듬어 주는 .clip( )

```
df.clip(lower=None, upper=None, axis=None, inplace=False,
```

```
df["A"]
```

0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

```
df['A'].clip(lower=-1,upper=5)
```

0	1
1	2
2	3
3	4
4	5
5	5
6	5
7	5
8	5
9	5

```
df['A'].clip(lower=2,upper=8)
```

0	2
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	8
9	8

```
>>> df
   col_0  col_1
0      9     -2
1     -3     -7
2      0      6
3     -1      8
4      5     -5
```

```
>>> t
0    2
1   -4
2   -1
3    6
4    3
```

t	t+4
2	6
-4	0
-1	3
6	10
3	7

axis = 0 이니까 index기준

0 : 2~6
1 : -4~0
2 : -1~3
3 : 6~10
4 : 3~7

```
>>> df.clip(t, t + 4, axis=0)
```

```
   col_0  col_1
0      6      2
1     -3     -4
2      0      3
3      6      8
4      5      3
```

8. 값을 제한하기– 1) .clip( )

```
df['B']
```

```
0   -0.862819
1    1.197189
2   -0.935074
3   -0.503223
4   -0.445516
5   -0.899583
6   -0.699034
7   -0.734427
8    0.407848
9    0.678296
Name: B, dtype: float64
```

```
df['B'].clip(lower=1,upper=3)
```

```
0    1.000000
1    1.197189
2    1.000000
3    1.000000
4    1.000000
5    1.000000
6    1.000000
7    1.000000
8    1.000000
9    1.000000
Name: B, dtype: float64
```

2) 절대값 구하는 .abs( )

별 다른 옵션이 없는 메서드

```
df['B']
```

```
0   -0.862819
1    1.197189
2   -0.935074
3   -0.503223
4   -0.445516
5   -0.899583
6   -0.699034
7   -0.734427
8    0.407848
9    0.678296
Name: B, dtype: float64
```

```
df['B'].abs()
```

```
0    0.862819
1    1.197189
2    0.935074
3    0.503223
4    0.445516
5    0.899583
6    0.699034
7    0.734427
8    0.407848
9    0.678296
Name: B, dtype: float64
```

```
import pandas as pd
import seaborn as sns
```

```
df = sns.load_dataset("mpg")
df.shape
```

```
(398, 9)
```

```
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

자동차와 관련된 데이터 인듯

중간고사 전에 익혔던 세 번째 주제 내용 중에..
index를 기준으로 행들을 정렬했던 매서드?

```
df5
```

	population	GDP	alpha-2	pop-GDP	aa2
Italy	59000000	1937894	IT	57062106	57062106
France	65000000	2583560	FR	62416440	62416440
Malta	434000	12011	MT	421989	421989
Maldives	434000	4520	MV	429480	429480
Brunei	434000	12128	BN	421872	421872
Iceland	337000	17036	IS	319964	319964
Nauru	11300	182	NR	11118	11118
Tuvalu	11300	38	TV	11262	11262
Anguilla	11300	311	AI	10989	10989

```
df5.sort_index(ascending = True)
```

	population	GDP	alpha-2	pop-GDP	aa2
Anguilla	11300	311	AI	10989	10989
Brunei	434000	12128	BN	421872	421872
France	65000000	2583560	FR	62416440	62416440
Iceland	337000	17036	IS	319964	319964
Italy	59000000	1937894	IT	57062106	57062106
Maldives	434000	4520	MV	429480	429480
Malta	434000	12011	MT	421989	421989
Nauru	11300	182	NR	11118	11118
Tuvalu	11300	38	TV	11262	11262

그러면,

sort_values('열이름')은 어떤 기능의 매서드 일까?

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320

```
df.sort_values('mpg')
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
28	9.0	8	304.0	193.0	4732	18.5	70	usa	hi 1200d
25	10.0	8	360.0	215.0	4615	14.0	70	usa	ford f250
26	10.0	8	307.0	200.0	4376	15.0	70	usa	chevy c20
103	11.0	8	400.0	150.0	4997	14.0	73	usa	chevrolet impala
124	11.0	8	350.0	180.0	3664	11.0	73	usa	oldsmobile omega

```
df.sort_values('cylinders').head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
111	18.0	3	70.0	90.0	2124	13.5	73	japan	maxda rx3
71	19.0	3	70.0	97.0	2330	13.5	72	japan	mazda rx2 coupe
334	23.7	3	70.0	100.0	2420	12.5	80	japan	mazda rx-7 gs
243	21.5	3	80.0	110.0	2720	13.5	77	japan	mazda rx-4
267	27.5	4	134.0	95.0	2560	14.2	78	japan	toyota corona

cylinders열의 값을 기준으로 오름차순 정렬, 앞부분 미리보기 5줄

```
df.sort_values('mpg', ascending = False).head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
322	46.6	4	86.0	65.0	2110	17.9	80	japan	mazda glc
329	44.6	4	91.0	67.0	1850	13.8	80	japan	honda civic 1500 gl
325	44.3	4	90.0	48.0	2085	21.7	80	europe	vw rabbit c (diesel)
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
326	43.4	4	90.0	48.0	2335	23.7	80	europe	vw dasher (diesel)

mpg값 기준으로 역순(내림차순) 정렬

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320

model_year를 year로 바꾸고 싶네.. 뭘 써야 할까?

힌트

df3

	1	2	3	4
a	4	5	6	6
b	7	8	9	9
c	10	11	12	12

```
df3.columns = ['a', 'b', 'c', 'd']
df3
```

	a	b	c	d
a	4	5	6	6
b	7	8	9	9
c	10	11	12	12

```
df3.index = [11, 22, 33]
df3
```

	a	b	c	d
11	4	5	6	6
22	7	8	9	9
33	10	11	12	12

또 가능한 거 뭐 있더라?

```
df33 = df3.rename(columns = {'a':'A', 'b':'B', 'c':'C', 'd':'D'})
df33
```

	A	B	C	D
11	4	5	6	6
22	7	8	9	9
33	10	11	12	12

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320

model_year를 year로 바꾸고 싶네.. 뭘 써야 할까?

```
df = df.rename(columns = {'model_year': 'year'})
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino