1. Pandas는
   1)Pandas는 label이 부여된 data를 쉽고 직관적으로 취급할 수 있도록 설계된 Python third-party package이다.
   2)Pandas의 2가지 주요 data 구조인 Series(1차원 data)와 DataFrame(2차원 data)은 금융, 통계, 사회과학 등 많은 분야의 data 처리에 적합하다.

2. Pandas의 특징
   1)Pandas의 주요 기능을 설명
      -쉬운 결손값(missing data)처리
      -Label 위치를 자동적/명시적으로 정리한 data 작성
      -Data 집약
      -고도의 label base의 slicing, 추출, 큰 dataset의 subset화
      -직감적인 dataset 결합
      -Dataset의 유연한 변환 및 변형
      -축의 계층적 label 붙임
      -여러가지 data 형식에 대응한 강력한 I/O
      -시계열 data 고유의 처리

   2)Pandas의 package 정보
      -Version : 0.25.1
      -공식 site : http://pandas.pydata.org
      -Repository : https://github.com/pandas-dev/pandas
      -PyPI : https://pypi.python.org/pypi/pandas

   3)설치 여부 확인
      -Jupyter 환경
         -!conda list | grep pandas
         -Windows에서는 grep명령어를 사용할 수 없음.
         -따라서 Windows에서는
            !conda list   #목록 중에서 찾아야 함.

      -PIP 환경
         -pip -V   #version 확인
         -pip install pandas   #관리자 권한으로 설치할 것
         -numpy도 같이 설치됨.
         -설치 후 pip list로 확인

   4)import pandas
      import pandas as pd

3. Series
   1)Series는 index라고 불리는 label을 가진 동일한 data형을 가지는 1차원 data이다.
      -Series는 DataFrame과 함께 pandas에서 제공하는 데이터 구조 중 하나이다.
      -DataFrame이 2차원 자료구조라면 Series는 1차원 자료구조이다.
      -DataFrame과 비슷하지만 1차원 자료구조이기 때문에 columns 속성이 없고 index 속성만 있다.

   2)다음의 특징이 있다.
      -Index(label)를 가지는 1차원 data
      -Index는 중복 가능
      -Label 또는 data의 위치를 지정한 추출가능.
      -Index에 대한 slice가 가능
      -산술 연산이 가능.
      -통계량을 산출하는 merit를 가지고 있음.

   3)Python 표준 list나 tuple 등에서 사용되는 index라는 언어와의 혼동을 피하기 위해 series의 index를 label이라고 한다.

   4)Series 작성하기
      -Series의 작성에는 pandas.series class를 사용한다.
      -pd.Series(data = [ ] , index = [ ])
      -제1인수에는 다음과 같은 1차원의 data를 넘겨준다.
         --List
         --Tuple
         --Dirctionary
         --numpy.ndarray
      -아래와 같이 keyword 인수 index에 label이 되는 값을 넘기는 것으로 data를 표시한다.

      -list로 Series 생성하기

         ser = pd.Series([10, 20, 30])
         ser   #index를 생략한 경우 : 0부터 차례대로 정수가 할당된다.
         ----
         0   10.0
         1   20.0
         2   0.3
         dtype: float64

         ser.index = ['a', 'b', 'c']
         ser
         ----
         a   10.0
         b   20.0
         c   0.3
         dtype: float64

```
ser1 = pd.Series([1,2,3], index=['a', 'b', 'c'])
ser1
-----------------------------------------------------
a    1
b    2
c    3
dtype: int64

ser2 = pd.Series( [1, 2, 3, 4], index = ['USA', 'Germany', 'France', 'Japan'] )
ser2
-----------------------------------------------------------
USA        1
Germany    2
France     3
Japan      4
dtype: int64

ser3 = pd.Series( [1, 2, 5, 4], index = ['USA', 'Germany', 'Italy', 'Japan'] )
ser3
-----------------------------------------------------------
USA        1
Germany    2
Italy      5
Japan      4
dtype: int64

fruits =  Series([2500,3800,1200,6000], index=['apple','banana','peer','cherry'])
fruits
-----------------------------------------------
apple             2500
banana            3800
peer                 1200
cherry            6000
dtype:   int64

fruits.values
-----------------------------------------------
array([2500, 3800, 1200, 6000], dtype=int64)

fruits.index
-----------------------------------------------------
Index(['apple', 'banana', 'peer', 'cherry'], dtype='object')
```

-dict로 Series 생성하기
--Series형태는 dict와 매우 유사 하여 key가 index로 바뀌었다고 착각할 수 있다.
--그러나 사실상 데이터 구조 자체가 매우 다르기 때문에 dict와는 사용하는 방식이 다르므로 주의해야 한다.

```
ser4 = pd.Series( {'a':100, 'b':200, 'c':300} )
ser4
-----------------------------------------------------
a    100
b    200
c    300
dtype: int64

fruits_dic   = {'apple':    2500,'banana':3800,'peer':1200,'cherry':6000}
fruits =  Series(fruits_dic)
type(fruits_dic)
-----------------------------------
dict

type(fruits)
-----------------------------------
pandas.core.series.Series
```

5)Label을 사용해서 data를 선택하기
-Series.loc를 사용해서 label에서 data를 선택한다.

```
ser = pd.Series([1,2,3], index=['a', 'b', 'c'])
ser.loc['b']
---------------
2
```

-loc를 사용하지 않는 서식
--loc를 사용하지 않는 다음과 같은 서식도 있다.

```
ser['b']
-----------
2
```

-Label의 범위 지정
--Label의 범위를 지정해서 slice를 할 수 있다.

```
ser.loc['b' : 'c']
----------------------------
b    2
c    3
dtype: int64
```

--Label에 따른 slice는 label의 시작 위치와 종료 위치를 포함한다.
--Python의 list나 tuple에 대한 slice와의 동작이 다름에 주의한다.

-복수의 요소 지정
--복수의 요소를 list로 지정할 수 있다.

```
ser.loc[['a', 'c']]
----------------------------
a    1
c    3
dtype: int64
```

6)위치를 지정해서 data 선택하기
-Series.iloc를 사용해서 data의 위치를 정수값으로 지정하고 data를 선택할 수 있다.

```
ser.iloc[1]
--------------------
2
```

-iloc의 slice는 Python 표준 list나 tuple에 대한 slice와 같이 동작한다.
-위치를 slice로 지정

```
ser.iloc[1:3]
----------------------------
b    2
c    3
dtype: int64
```

7)논리값을 사용해서 data 선택하기
-loc와 iloc에는 논리값의 list를 넘길 수 있다.

```
ser.loc[[True, False, True]]
----------------------------------
a    1
c    3
dtype: int64
```

-인수에 부여된 논리값 list는 Series의 index 위치에 대하여 True에 지정된 위치만 되돌아간다.
-Series에 대한 비교 연산을 통해 논리값을 되돌려준다.

```
ser != 2
----------------------------
a  True
b  False
c  True
dtype: bool
```

-이것을 이용해서 data를 추출할 수 있다.

```
ser.loc[ser != 2]
-------------------------
a    1
c    3
dtype: int64
```

8)Series data 삭제하기
-drop()은 데이터 구조의 row(행) 또는 column(열) 요소를 삭제.

```
fruits =  Series([2500, 3800,    1200,    6000],
index=['apple','banana','peer','cherry'])
fruits
-----------------------------------------
apple            2500
banana           3800
peer               1200
cherry           6000
dtype:   int64

new_fruits =  fruits.drop('banana')
new_fruits
-----------------------------------------
apple            2500
peer               1200
cherry           6000
```

```
dtype:  int64
```

9)Series data의 기본 연산

-다음 코드는 시리즈 데이터의 + 연산자로 더하는 예이다.

-연산하는 Series들의 index 중 하나라도 NaN(결측치)가 존재하면 연산 결과도 무조건 NaN으로 나온다.

```
fruits1  = Series([5,9,10,3],  index=['apple','banana','cherry','peer'])
fruits2  = Series([3,2,9,5,10],   index=['apple','orange','banana','cherry','mango'])
fruits1
-----------------------------------------------
apple              5
banana             9
cherry           10
peer                  3
dtype:  int64

fruits2
-----------------------------------------------
apple              3
orange             2
banana             9
cherry             5
mango              10
dtype:  int64
```

-Series의 연산은 index에 의해 연산된다.

-아래의 결과를 보면 'mango' index와 'orange' index는 fruits2 Series에만 있으므로 그 결과가 NaN이 되고

-'peer' index는 fruits1 Series만 있으므로 그 결과가 NaN이 된다.

```
fruits1 +   fruits2
-----------------------------------
apple              8.0
banana          18.0
cherry           15.0
mango                NaN
orange            NaN
peer                 NaN
dtype:  float64
```

10)Series data 정렬

-정렬은 sort_values()를 이용.

```
fruits =  Series([2500,3800,1200,6000], index=['apple','banana','peer','cherry'])
fruits.sort_values(ascending=False)
----------------------------------------
cherry           6000
banana           3800
apple            2500
peer                1200
dtype:  int64
```

11)Series를 DataFrame임으로

-to_frame() 함수를 이용.

```
fruits1  = Series([5,9,10,3],  index=['apple','banana','cherry','peer'])
fruits1.to_frame()
------------------------------
                0
apple      5
banana    9
cherry     10
peer       3
```

-열 단위의 DataFrame을 행단위로 바꾸려면 T 속성 또는 transpose() 함수를 이용.

-T 속성과 transpose() 함수는 행렬의 전치행렬을 반환한다.

```
fruits1.to_frame().T
--------------------------------
     apple banana  cherry   peer
0    5       9          10      3

fruits1.to_frame().transpose()
--------------------------------
     apple banana  cherry   peer
0    5       9          10      3
```

4. DataFrame

1)DataFrame은 행과 열에 label을 가진 2차원 data이다.

2)Data형은 열마다 다른 형을 가질 수 있다.

3)1차원 data인 Series의 집합으로 인식하는 것도 가능하다.
4)Series의 특징을 포함해서 DataFrame에는 다음과 같은 특징이 있다.
-행과 열에 label을 가진 2차원 data
-열마다 다른 형태를 가질 수 있음.
-Table형 data에 대해 불러오기 / data 쓰기가 가능
-DataFrame끼리 여러가지 조건을 사용한 결합 처리가 가능
-Cross 집계가 가능

5)Python 표준 list나 tuple 등에서 사용되는 index라는 언어와의 혼동을 피하기 위해서 DataFrame의 index를 label이라고 한다.

5. DataFrame 생성하기

1)DataFrame의 생성에는 pandas.DataFrame class를 사용한다.
2)제1인수에는 1차원 또는 2차원 data를 넘긴다.
3)Keyword 인수 index(행) 및 columns(열)에 label이 되는 값을 넘기는 것으로 data를 표시한다.
4)dict를 이용한 DataFrame 생성하기
-dict를 를 이용하면 dict의 key가 열 이름이 된다.

```
d = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data=d)
df
----------------------------------------
   col1  col2
0     1     3
1     2     4
```

-dict는 다음처럼 list에 저장되어 있어도 쉽게 DataFrame으로 만들 수 있다.

```
d = [{'col1': 1, 'col2': 3}, {'col1': 2, 'col2': 4}]
df = pd.DataFrame(data=d)
df
----------------------------------------
   col1  col2
0     1     3
1     2     4
```

-만일 list내의 dict의 요소의 수가 다를 경우에는 NaN으로 채워진다.

```
d = [{'col1': 1, 'col2': 3}, {'col1': 2, 'col2': 4}, {'col1': 2}]
df = pd.DataFrame(data=d)
df
----------------------------------------
   col1  col2
0     1     3.0
1     2     4.0
2     2     NaN

emp_list = [
    {'name':'John', 'age' : 25, 'job' : 'Manager'},
    {'name':'Smith', 'age': 30, 'job' : 'Salesman'}
]

df = pd.DataFrame(emp_list)
df
----------------------------------------
   age      job     name
0  25    Manager    John
1  30    Salesman   Smith
#list의 순서와 맞지 않음. key의 alphabet 순서와 동일

df = df[['name','age','job']]
df.head()
----------------------------------------
   name   age   job
0  John    25   Manager
1  Smith   30   Salesman
```

5)OrderDictionary 이용하기(key 순서 보장)

```
from collections import OrderedDict
emp_ordered_list = OrderedDict(
    [
        ('name', ['John', 'Smith']),
        ('age', [25, 30]),
        ('job', ['Manager', 'Salesman']),
    ]
)

df = pd.DataFrame.from_dict(emp_ordered_list)
df.head()
----------------------------------------------------
   name   age   job
0  John    25   Manager
```

```
1  Smith       30    Salesman
```

6)list를 이용해 DataFrame 만들기

```
df = pd.DataFrame(
        [[1, 10, 100], [2, 20, 200], [3, 30, 300]], index=['r1', 'r2', 'r3'],
        columns=['c1','c2','c3'])
df
```

```
    c1    c2    c3
r1  1     10    100
r2  2     20    200
r3  3     30    300
```

```
emp_list = [
      ['John', 25, 'Manager'],
      ['Smith', 30, 'Salesman']
   ]

column_names = ['name', 'age', 'job']
df = pd.DataFrame.from_records(emp_list, columns =column_names)
df.head()
```

```
    name    age     job
0   John      25    Manager
1   Smith     30    Salesman
```

```
emp_list = [
    ['name', ['John', 'Smith']],
    ['age', [25, 30]],
    ['job', ['Manager', 'Salesman']],
]

df = pd.DataFrame.from_items(emp_list)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: from_items is deprecated. Please use DataFrame.from_dict(dict(items), ...) instead. DataFrame.from_dict(OrderedDict(items)) may be used to preserve the key order.
"""Entry point for launching an IPython kernel.
```

```
df.head()
```

```
   name    age     job
0  John      25    Manager
1  Smith     30    Salesman
```

7)Series를 이용한 DataFrame 생성하기

```
list = [1,2,3]

ser1 = pd.core.series.Series(list)
ser2 = pd.core.series.Series(['one', 'two', 'three'])
pd.DataFrame(data=dict(num=ser1, word=ser2))
```

```
   num  word
0  1      one
1  2      two
2  3      three
```

8)read_csv()함수를 이용해서 DataFrame 만들기

-read_csv() 함수는 CSV 파일을 읽어 DataFrame으로 만든다.

-CSV 파일을 읽을 때 sep 매개변수로 구분자를 지정할 수 있다.

```
import   pandas  as pd
member_df    =  pd.read_csv("member_data.csv", sep=",")
member_df
```

```
   Name       Age      Email                        Address
0  홍길동     20    kildong@hong.com     서울시 강동구
1  홍길서     25    kilseo@hong.com          서울시 강서구
2  홍길남     26    south@hong.com           서울시 강남구
3  홍길북     27    book@hong.com        서울시 강북구
```

-csv파일에 헤더 정보가 없을 경우 header=None 인수를 포함하면 열 이름이 0, 1, 2, ...순으로 자동 지정된다.

9)sklearn.datasets module data를 DataFrame으로 변환하기

-Scikit-learn package에는 학습을 위한 많은 dataset이 제공된다.

-Scikit-learn에서 제공하는 dataset은 dict 형식으로 되어 있다.

```
import numpy as np
import pandas as pd
from sklearn import datasets
iris = datasets.load_iris()
iris
-----------------------------------------
{'data': array([[5.1, 3.5, 1.4, 0.2],
[4.9, 3. , 1.4, 0.2],
[4.7, 3.2, 1.3, 0.2],
[4.6, 3.1, 1.5, 0.2],
[5. , 3.6, 1.4, 0.2],
... 생략 ...
[6.7, 3. , 5.2, 2.3],
[6.3, 2.5, 5. , 1.9],
[6.5, 3. , 5.2, 2. ],
[6.2, 3.4, 5.4, 2.3],
[5.9, 3. , 5.1, 1.8]]),

'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
'DESCR': 'Iris Plants Database...
...
'feature_names':['sepal length (cm)',
'sepal width (cm)',
'petal length (cm)',
'petal width (cm)'],
'filename': 'c:\\pythonhome\\projectenv\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv'}


x = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.DataFrame(iris['target_names'][iris['target']], columns=["species"])
iris_df = pd.concat([x,y], axis=1)
iris_df.head()
-----------------------------------------------------------
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	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

6. 이름 지정하기

1)DataFrame이 열 또는 행의 이름을 지정하면 이름으로 데이터의 부분집합을 얻거나 정렬할 수 있다.

2)행의 이름은 index, 열의 이름은 columns 속성을 이용한다.

3)열 이름 지정하기

-DataFrame의 columns 속성을 이용하면 열의 이름들을 지정할 수 있다.

```
member_df.columns = ["이름", "나이", "이메일", "주소"]
member_df
-----------------------------------------------------------------
```

	이름	나이	이메일	주소
0	홍길동	20	kildong@hong.com	서울시 강동구
1	홍길서	25	kilseo@hong.com	서울시 강서구
2	홍길남	26	south@hong.com	서울시 강남구
3	홍길북	27	book@hong.com	서울시 강북구

```
member_df.columns
-----------------------------------------------------------
Index(['이름', '나이', '이메일', '주소'], dtype='object')
```

4)행 이름 지정하기

-행은 index 속성을 이용해 이름을 지정할 수 있다.

```
member_df.index = ["동", "서", "남", "북"]
member_df
-----------------------------------------------------------
```

	이름	나이	이메일	주소
동	홍길동	20	kildong@hong.com	서울시 강동구
서	홍길서	25	kilseo@hong.com	서울시 강서구
남	홍길남	26	south@hong.com	서울시 강남구
북	홍길북	27	book@hong.com	서울시 강북구

5)level 이름 지정하기

-DataFrame의 열 이름과 행 이름은 group을 지어 지정할 수 있다.

-이렇게 하면 한 개의 열은 두 개 이상의 열 이름 또는 행 이름을 가질 수 있는데 이때 level을 이용해 이름을 구분할 수 있다.

-level의 이름은 names 속성을 이용할 수 있다.

-열의 이름을 지정할 때에는 columns 속성을 이용하고, 열의 level을 지정하려면 columns.names 속성을 이용한다.

-행(인덱스)의 이름을 지정할 때에는 index 속성을 이용하고, 행의 레벨을 지정하려면 index.names 속성을 이용한다.

```
member_df    =  pd.read_csv("member_data.csv", comment='#')
member_df.columns    =  [["기본정보", "기본정보",   "추가정보",   "추가정보"],
                                                ["이름",   "나이",    "이메일", "주소"]]
member_df.columns.names =  ["정보구분",  "상세정보"]
member_df.index   =  [["좌우","좌우","상하","상하"],
                                        ["동", "서",  "남",  "북"]]
member_df.index.names =  ["위치구분",  "상세위치"]
member_df
```

정보구분		기본정보		추가정보	
	상세정보	이름	나이	이메일	주소
위치구분	상세위치				
좌우	동	홍길동	20	kildong@hong.com	서울시 강동구
	서	홍길서	25	kilseo@hong.com	서울시 강서구
상하	남	홍길남	26	south@hong.com	서울시 강남구
	북	홍길북	27	book@hong.com	서울시 강북구

7. 부분 데이터 조회

```
member_df    =  pd.read_csv("member_data.csv")
member_df
```

1)단일 열 조회

-참조 형식(.) 또는 배열 형식([ ])을 이용하면 열 하나를 조회할 수 있다.

-DataFrame의 열 이름을 참조 형식(. 연산자)을 이용해서 정보를 조회할 수 있다.

```
member_df.Name
```

```
0          홍길동
1          홍길서
2          홍길남
3          홍길북
Name:  Name,  dtype:  object
```

-열 정보를 조회할 때에는 배열 형식(["열이름"])으로도 가능하다.

```
member_df["Name"]
```

```
0          홍길동
1          홍길서
2          홍길남
3          홍길북
Name:  Name,  dtype:  object
```

2)loc를 이용한 이름으로 조회

-loc[ ]를 이용하면 행 또는 열 이름으로 부분 데이터셋을 조회할 수 있다.

-콜론(:)은 사이의 모든 행을 선택한다.

```
member_df.loc[0:2]
```

	Name	Age	Email	Address
0	홍길동	20	kildong@hong.com	서울시 강동구
1	홍길서	25	kilseo@hong.com	서울시 강서구
2	홍길남	26	south@hong.com	서울시 강남구

-loc에서의 숫자는 인덱스가 아니다.

-그러므로 0:2는 인덱싱 할 때의 from:to 개념을 적용한 0행부터 2행까지를 의미하는 것이 아니다.

-loc() 함수에서 0:2는 0, 1, 2 행을 의미한다.

-loc[ ]는 기본적으로 행의 이름을 이용해서 부분 데이터셋을 조회한다.

```
member_df.loc["Name":"Email"]  #  nothing
```

-행과 열의 이름을 모두 이용해서 부분 데이터셋을 조회할 수 있다.

```
member_df.loc[0:2,   "Name":"Email"]
```

	Name	Age	Email
0	홍길동	20	kildong@hong.com
1	홍길서	25	kilseo@hong.com
2	홍길남	26	south@hong.com

-행 또는 열의 이름으로 데이터를 조회려면 리스트 형식으로 지정한다.

```
member_df.loc[[0,2], ["Name","Email"]]
```

```
------------------------------------------------------------
      Name       Email
0     홍길동     kildong@hong.com
2     홍길남     south@hong.com
```

-모든 열을 지정하는 경우 요소에 [:]를 넘긴다.

```
member_df.loc[0:2, :]
-------------------------
      Name      Age      Email                        Address
0     홍길동    20    kildong@hong.com     서울시 강동구
1     홍길서    25    kilseo@hong.com          울시 강서구
2     홍길남    26    south@hong.com           서울시 강남구
```

-모든 행을 지정하는 경우도 같다.

```
member_df.loc[:, 'Email']
-------------------------
0    kildong@hong.com
1     kilseo@hong.com
2      south@hong.com
3       book@hong.com
Name: Email, dtype: object
```

3)iloc를 이용한 index로 조회
-iloc[from_index : to_index]는 index를 이용해서 부분 데이터셋을 조회한다.
-to_index는 포함되지 않는다.
-indexing 방법은 Python list의 indexing 방법을 사용할 수 있다.

```
member_df.iloc[1:3,   1:3]
-----------------------------
      Age      Email
1     25    kilseo@hong.com
2     26    south@hong.com

member_df.iloc[0:3,   0:3]
-----------------------------
      Name      Age      Email
0     홍길동    20    kildong@hong.com
1     홍길서    25    kilseo@hong.com
2     홍길남    26    south@hong.com
```

4)iloc[from_index : to_index : by] 형식을 사용할 수 있다.
-이것은 from_index 부터 to_index까지 매 by마다 데이터를 조회한다.

```
member_df.iloc[::-1]
---------------------------------------
      Name      Age      Email                        Address
3     홍길북    27    book@hong.com        서울시 강북구
2     홍길남    26    south@hong.com           서울시 강남구
1     홍길서    25    kilseo@hong.com          서울시 강서구
0     홍길동    20    kildong@hong.com     서울시 강동구

member_df.iloc[0::2,[1,3]]
-----------------------------------------------
     Age      Address
0    20    서울시 강동구
2    26    서울시 강남구
```

5)조건으로 조회하기
-조건으로 데이터 조회를 설명하기 전에 package를 import하고 예제로 사용할 데이터를 불러와 DataFrame으로 만든다.
-iris data는 sklearn package의 dataset에서 불러올 수도 있지만 DataFrame으로 사용하려면 statsmodels package의 dataset에서 불러오는 것이 더 쉽다.

```
import   numpy   as np
import   pandas  as pd
import   statsmodels.api  as sm  #pip install statsmodels
iris    =  sm.datasets.get_rdataset("iris",   package="datasets")
iris
-------------------------------------------
<class 'statsmodels.datasets.utils.Dataset'>
```

-get_rdataset() 함수로 불러온 데이터는 다음 속성을 가지고 있다.
--package : 데이터를 제공하는 R package 이름.
--title : 데이터의 이름.
--data : 데이터를 담고 있는 DataFrame.
--__doc__ : 데이터에 대한 설명 문자열.
--이 설명은 R package의 내용을 그대로 가져온 것이므로 예제 코드가 R로 되어 있어 Python에서 바로 사용할 수 없다.
-statsmodels 패키지의 get_rdataset() 함수로 불러온 데이터의 data 속성은 데이터를 담고 있는 DataFrame이다.

```
iris_df   =  iris.data
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

-다음 코드는 versicolor 종의 데이터만 조회한다.

```
iris_df.loc[iris_df['Species']=='versicolor'].head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor

-다음 코드는 versicolor 종의 Sepal.Length열과 Species 열 정보만 조회한다.

```
iris_df.loc[iris_df['Species']=='versicolor', ['Sepal.Length',  'Species']].head()
```

	Sepal.Length	Species
50	7.0	versicolor
51	6.4	versicolor
52	6.9	versicolor
53	5.5	versicolor
54	6.5	versicolor

-다음 코드는 versicolor 종들 중에서 Sepal.Length가 6.5보다 큰 데이터만 조회한다.

```
iris_df.loc[(iris_df['Species']=='versicolor')    &  (iris_df['Sepal.Length'].astype(float) >  6.5)].head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
50	7.0	3.2	4.7	1.4	versicolor
52	6.9	3.1	4.9	1.5	versicolor
58	6.6	2.9	4.6	1.3	versicolor
65	6.7	3.1	4.4	1.4	versicolor
75	6.6	3.0	4.4	1.4	versicolor

-좀 더 쉬운 다른 예제를 사용해 보자.

```
user_list = [
    {'Name':'John', 'Age':25, 'Gender' : 'male', 'Address':'Chicago'},
   {'Name':'Smith', 'Age':35, 'Gender' : 'male', 'Address':'Boston'},
   {'Name':'Jenny', 'Age':45, 'Gender' : 'female', 'Address':'Dallas'}
]

df = pd.DataFrame(user_list)
df = df[['Name', 'Age', 'Gender','Address']]
df.head()
```

	Name	Age	Gender	Address
0	John	25	male	Chicago
1	Smith	35	male	'Boston'
2	Jenny	45	female	Dallas

-age가 30보다 많은 사람의 정보

```
df[df.Age > 30]
```

	Name	Age	Gender	Address
1	Smith	35	male	Boston
2	Jenny	45	female	Dallas

```
df.query('Age > 30')
```

	Name	Age	Gender	Address
1	Smith	35	male	Boston
2	Jenny	45	female	Dallas

-age가 30보다 많고 이름이 'Smith'인 사람의 정보

```
df[(df.Age > 30) & (df.Name == 'Smith')]
# or df.loc[(df.Age > 30) & (df.Name == 'Smith')]
```

	Name	Age	Gender	Address
1	Smith	35	male	Boston

-Column name이 Name과 Gender인 column만 가져오기

```
df.filter(items=['Name', 'Gender'])
```

```
        Name    Gender
0       John    male
1       Smith   male
2       Jenny   female
```

-Column name이 'A'라는 글자가 있는 Column만 가져오기

```
df.filter(like = 'A')
---------------------------------
        Age     Address
0       25    Chicago
1       35    Boston
2       45    Dallas
```

-정규식을 이용하여 column name이 'e'로 끝나는 Column만 가져오기

```
df.filter(regex = 'e$')
--------------------------
        Name    Age
0       John    25
1       Smith   35
2       Jenny   45
```

-정규식을 이용하여 column name이 'A'로 시작하는 Column만 가져오기

```
df.filter(regex = '^A')
------------------------------
   Address  Age
0  Chicago   25
1   Boston   35
2   Dallas   45
```

8. 데이터 삭제하기

1)drop() 함수는 데이터 구조의 row(행) 또는 column(열) 요소를 삭제한다.

2)Syntax

```
DataFrame.drop(labels=None, axis=0, inplace=False)
```

-labels : 삭제할 index 또는 컬럼의 이름.

-axis : int 타입 또는 축의 이름.

--(0 또는 'index') 와 (1 또는 'columns') 중 하나

--1이면 열을 삭제.

-inplace : bool 타입.

--False(기본값)이면 삭제된 결과 DataFrame을 리턴하며, True 이면 현재 DataFrame에서 데이터를 삭제하고 None을 반환.

3)row index를 사용하여 삭제하기

```
user_list = [
    {'Age':25, 'Gender' : 'male', 'Address':'Chicago'},
    {'Age':35, 'Gender' : 'male', 'Address':'Boston'},
    {'Age':45, 'Gender' : 'female', 'Address':'Dallas'}
]

df = pd.DataFrame(user_list,
                  index = ['John', 'Smith', 'Jenny'],
                  columns = ['Age', 'Gender', 'Address'])

df.head()
-----------------------------------------
        Age     Gender     Address
John    25    male         Chicago
Smith   35    male         Boston
Jenny   45    female       Dallas

df.drop(['John', 'Jenny'])
-------------------------------------
        Age     Gender     Address
Smith     35    male         Boston

df.head()  #하지만 여전히 df는 예전값을 갖고 있다.

df = df.drop(['John', 'Jenny'])    #이렇게 하면 제거된 결과를 df가 갖게 된다.
df.head()
-------------------------------------
        Age     Gender     Address
Smith     35    male         Boston
```

4)inplace keyword 인수 이용하기

```
df = pd.DataFrame(user_list,
                  index = ['John', 'Smith', 'Jenny'],
                  columns = ['Age', 'Gender', 'Address'])
```

```
df.drop(['John', 'Jenny'], inplace=True)
df
-----------------------------------------------
           Age     Gender     Address
Smith      35   male          Boston
```

5)row가 숫자로 indexing되어 있을 때 삭제하기

```
user_list = [
    {'Name':'John', 'Age':25, 'Gender' : 'male', 'Address':'Chicago'},
    {'Name':'Smith', 'Age':35, 'Gender' : 'male', 'Address':'Boston'},
    {'Name':'Jenny', 'Age':45, 'Gender' : 'female', 'Address':'Dallas'}
]

df = pd.DataFrame(user_list)

df = df[['Name', 'Age', 'Gender','Address']]

df
-----------------------------------
      Name      Age     Gender     Address
0     John      25    male          Chicago
1     Smith     35    male          Boston
2     Jenny     45    female        Dallas

df = df.drop(df.index[[0, 2]])
df
-------------------------------------
   Name      Age     Gender     Address
1  Smith     35    male          Boston
```

6)Condition으로 삭제하기

```
df.head()
--------------------------------
        Name      Age     Gender     Address
0       John      25    male          Chicago
1       Smith     35    male          Boston
2       Jenny     45    female        Dallas
```

-Age가 30 이상인 사람의 정보만 저장함으로 나머지 정보는 삭제하는 방법

```
df = df[df.Age > 30]
df
-----------------------------
      Name      Age     Gender     Address
1     Smith     35    male          Boston
2     Jenny     45    female        Dallas
```

7)앞에서 사용했던 member_data.csv 파일 데이터를 이용해 보자.

```
member_df    = pd.read_csv("member_data.csv")
member_df.columns   = ["이름",  "나이",    "이메일", "주소"]
member_df.index   = ["동", "서",  "남",  "북"]
member_df
-----------------------------------------------------
      이름      나이      이메일                              주소
동    홍길동 20    kildong@hong.com       서울시 강동구
서    홍길서 25    kilseo@hong.com           서울시 강서구
남    홍길남 26    south@hong.com            서울시 강남구
북    홍길북 27    book@hong.com          서울시 강북구
```

8)단일 행 삭제하기

```
member_df   =  member_df.drop('북')  #axis=0(기본값)이면  행에서 찾아   삭제
member_df
-----------------------------------------------
      이름      나이      이메일                              주소
동    홍길동 20    kildong@hong.com    서울시 강동구
서    홍길서 25    kilseo@hong.com         서울시 강서구
남    홍길남 26    south@hong.com          서울시 강남구
```

-행의 이름을 지정하지 않았다면 기본값은 아마도 숫자일 것이다.
-이 경우 행 번호가 행의 이름이 된다.

```
member_df2  =  pd.read_csv("member_data.csv")
member_df2.drop(2,  axis=0)
```

9) 단일 열 삭제하기

-axis인자의 값이 1이면 열을 삭제한다.

```
member_df   =  member_df.drop('주소',axis=1)
member_df
```

	이름	나이	이메일
동	홍길동	20	kildong@hong.com
서	홍길서	25	kilseo@hong.com
남	홍길남	26	south@hong.com

```
user_list = [
    {'Name':'John', 'Age':25, 'Gender' : 'male', 'Address':'Chicago'},
    {'Name':'Smith', 'Age':35, 'Gender' : 'male', 'Address':'Boston'},
    {'Name':'Jenny', 'Age':45, 'Gender' : 'female', 'Address':'Dallas'}
]

df = pd.DataFrame(user_list)
df = df[['Name', 'Age', 'Gender','Address']]
df.head()
```

	Name	Age	Gender	Address
0	John	25	male	Chicago
1	Smith	35	male	Boston
2	Jenny	45	female	Dallas

```
df.drop('Age', axis= 1)
```

	Name	Gender	Address
0	John	male	Chicago
1	Smith	male	Boston
2	Jenny	female	Dallas

```
df.drop('Age', axis= 1,  inplace=True)
df
```

	Name	Gender	Address
0	John	male	Chicago
1	Smith	male	Boston
2	Jenny	female	Dallas

10) 복수 열 삭제하기

-여러 개 열을 삭제하려면 labels 인자를 이용한다.

-axis 인자가 1일 경우 열을 의미한다.

```
member_df.drop(labels=["이메일", "주소"],  axis=1)
```

-axis=1과 axls="columns"는 같은 의미이다.

위의 코드와 다음 코드 실행 결과는 같다.

```
member_df.drop(labels=["이메일", "주소"],  axis="columns")
```

	이름	나이
동	홍길동	20
서	홍길서	25
남	홍길남	26
북	홍길북	27

9. DataFrame 요소 추가

1)열 추가

-DataFrame에 없는 열을 지정해서 값을 할당하면 새로운 열이 만들어 진다.

-이 때 열은 가장 오른쪽에 만들어진다.

```
member_df = pd.read_csv("member_data.csv")
member_df["BirthYear"] = 2000
member_df
```

	Name	Age	Email	Address	BirthYear
0	홍길동	20	kildong@hong.com	서울시 강동구	2000
1	홍길서	25	kilseo@hong.com	서울시 강서구	2000
2	홍길남	26	south@hong.com	서울시 강남구	2000
3	홍길북	27	book@hong.com	서울시 강북구	2000

-열을 추가할 때 행별로 다른 값을 지정할 수 있다.

```
member_df   =  pd.read_csv("member_data.csv")
member_df["BirthYear"] = [2001, 2002, 2003, 2004]
member_df
```

	Name	Age	Email	Address	BirthYear
0	홍길동	20	kildong@hong.com	서울시 강동구	2001

```
1    홍길서    25    kilseo@hong.com        서울시 강서구    2002
2    홍길남    26    south@hong.com         서울시 강남구    2003
3    홍길북    27    book@hong.com       서울시 강북구    2004
```

-만일 누락되어야 하는 값이 있다면 None으로 지정한다.

```
member_df   =  pd.read_csv("member_data.csv")
member_df["BirthYear"] = [2001, 2002, 2003, None]
member_df
```

```
-----------------------------------------------------
     Name      Age      Email                    Address       BirthYear
0    홍길동    20    kildong@hong.com     서울시 강동구    2001.0
1    홍길서    25    kilseo@hong.com        서울시 강서구    2002.0
2    홍길남    26    south@hong.com         서울시 강남구    2003.0
3    홍길북    27    book@hong.com       서울시 강북구    NaN
```

-None 값을 포함하면 NaN은 실수 유형으로 간주되기 때문에 정수 자료형 값들은 모두 실수 자료형으로 바뀌어 저장된다.

2)Series를 이용한 열 추가

-Series를 이용하면 index의 이름(행번호)을 지정해서 값을 할당할 수 있다.

```
member_df    = pd.read_csv("member_data.csv")
member_df["BirthYear"]  =  pd.Series([2001,2002,   2004],  index=[0,1,3])
member_df
```

```
-----------------------------------------------------------
     Name      Age      Email                    Address       BirthYear
0    홍길동    20    kildong@hong.com     서울시 강동구    2001.0
1    홍길서    25    kilseo@hong.com        서울시 강서구    2002.0
2    홍길남    26    south@hong.com         서울시 강남구    NaN
3    홍길북    27    book@hong.com       서울시 강북구    2004.0
```

3)dict로 행 추가

-DataFrame의 행으로 새로운 데이터를 추가할 수 있다.

-추가할 데이터가 다음처럼 dict로 되어 있다면 쉽게 DataFrame에 행으로 추가할 수 있다.

```
member_df   =  pd.read_csv("member_data.csv")
new_member =  {"Name":  "한지민", "Age": 23, "Email": "jimin@naver.com", "Address":   "서울시   송파구"}
```

-추가할 데이터가 Series가 아니라면 ignore_index=True를 설정해야 한다.

```
new_df  =  member_df.append(new_member,   ignore_index=True)
new_df
```

```
-----------------------------------------------------
     Name      Age      Email                    Address
0    홍길동    20    kildong@hong.com     서울시 강동구
1    홍길서    25    kilseo@hong.com        서울시 강서구
2    홍길남    26    south@hong.com         서울시 강남구
3    홍길북    27    book@hong.com       서울시 강북구
4    한지민    23    jimin@naver.com        서울시 송파구
```

4)다른 예제로 연습해 보자.

```
user_list = [
       {'Name':'John', 'Age':15, 'Gender' : 'male', 'Address':'Chicago', 'Job' : 'Student'},
       {'Name':'Smith', 'Age':25, 'Gender' : 'male', 'Address':'Boston', 'Job' : 'Teachter'},
       {'Name':'Jenny', 'Age':17, 'Gender' : 'female', 'Address':'Dallas', 'Job' : 'Student'}
]

df = pd.DataFrame(user_list)
df = df[['Name', 'Age', 'Gender','Address', 'Job']]
df.head()
```

```
---------------------------------
     Name      Age      Gender     Address       Job
0    John     15    male        Chicago       Student
1    Smith      25    male        Boston         Teachter
2    Jenny      17    female      Dallas         Student
```

-column 새로 추가하기

```
df['Salary'] = 0

df.head()
```

```
-----------------------------------------------------
     Name      Age      Gender     Address    Job              Salary
0    John      15    male          Chicago    Student       0
1    Smith      25    male          Boston        Teacher       0
2    Jenny      17    female        Dallas        Student       0
```

-기존 column 값을 이용

-'Job'에 따라 'Salary' 여부 Column으로 수정

```
df['Salary'] = np.where(df['Job'] != 'Student', 'yes', 'no')
df.head()
```

```
-----------------------------------------------------------
     Name      Age      Gender     Address Job            Salary
```

```
   0    John      15   male        Chicago Student    no
   1    Smith     25   male        Boston    Teacher  yes
   2    Jenny     17   female      Dallas    Student  no
```

-'Total' column 추가

```
student_list = [
    {'Name':'John', 'Midterm':95, 'Final' : 85},
    {'Name':'Smith', 'Midterm':85, 'Final' : 80},
    {'Name':'Jenny', 'Midterm':30, 'Final' : 10},
]

df = pd.DataFrame(student_list, columns= ['Name', 'Midterm', 'Final'])

df.head()
-------------------------------------------------
   Name     Midterm Final
0  John     95      85
1  Smith    85      80
2  Jenny    30      10

df['Total'] = df['Midterm'] + df['Final']
df
-----------------------------------------
   Name     Midterm Final   Total
0  John     95      85    180
1  Smith    85      80    165
2  Jenny    30      10    40
```

-'Average' column 추가하기

```
df['Average'] = df['Total'] / 2
df.head()
-----------------------------------------
   Name     Midterm Final      Total   Average
0  John     95          85    180       90.0
1  Smith    85          80    165       82.5
2  Jenny    30          10    40      20.0
```

-'Grade' column 추가하기

```
grade_list = []
for row in df['Average']:
    if row <= 100 and row >= 90 :
        grade_list.append('A')
    elif row < 90 and row >= 80 :
        grade_list.append('B')
    elif row < 80 and row >= 70 :
        grade_list.append('C')
    elif row < 70 and row >= 60 :
        grade_list.append('D')
    else : grade_list.append('F')

df['Grade'] = grade_list
df
-----------------------------------------------
   Name     Midterm Final      Total   Average Grade
0  John     95      85      180       90.0       A
1  Smith    85      80      165       82.5       B
2  Jenny    30      10      40      20.0       F
```

10. 정렬

1)DataFrame을 정렬하려면 sort_index() 또는 sort_value() 함수를 이용한다.
2)정렬의 경우 에도 inplace 인자를 사용할 수 있다.
3)inplace=True인 경우 원본 데이터프레임이 변경된다.
4)Syntax

```
DataFrame.sort_index(axis=0, level=None,   ascending=True,
                                    inplace=False,    kind='quicksort',
                                    na_position='last',  sort_remaining=True,
                                    by=None)
DataFrame.sort_values(by, axis=0, ascending=True, inplace=False,
   kind='quicksort', na_position='last')
```

-label : 정렬할 level을 지정.
-axis : int 타입 또는 축의 이름. (0 또는 'index') 와 (1 또는 'columns') 중 하나
-ascending : True(기본값) 이면 오름차순, False 이면 내림차순으로 정렬.
-inplace : bool 타입이며, False(기본값)이면 삭제된 결과 DataFrame을 리턴하며, True 이면 현재 DataFrame에서 데이터를 삭제하고 None을 반환.
-kind : 정렬 알고리즘을 지정.
  --정렬 방법은 "quicksort", "mergesort", "heapsort" 중 하나.
  --기본값은 "quicksort"이며 이 옵션은 단일 열 또는 단일 index를 정렬할 때만 적용.
-na_position : NaN 값을 놓을 위치를 "first" 또는 "last"로 지정.

--기본값은 NaN을 마지막에 두는 "last".

-sort_remaining : 만일 True(기본값)이고 레벨(level)로 정렬하며 index가 멀티레벨일 경우 지정한 레벨로 정렬 할 후 다른 레벨을 이용해도 정렬.

-by : sort_values() 함수에서 by 인자는 정렬의 기준이 되는 열 또는 행의 이름을 지정.

```
member_df   = pd.read_csv("member_data.csv")
member_df.index  = ["동", "서", "남", "북"]
member_df
```

	Name	Age	Email	Address
동	홍길동	20	kildong@hong.com	서울시 강동구
서	홍길서	25	kilseo@hong.com	서울시 강서구
남	홍길남	26	south@hong.com	서울시 강남구
북	홍길북	27	book@hong.com	서울시 강북구

5)행 이름으로 정렬

-DataFrame을 index 기준으로 정렬하려면 sort_index()를 이용.

-sort_index() 함수는 DataFrame의 행 이름을 이용해서 정렬.

```
member_df.sort_index()
```

	Name	Age	Email	Address
남	홍길남	26	south@hong.com	서울시 강남구
동	홍길동	20	kildong@hong.com	서울시 강동구
북	홍길북	27	book@hong.com	서울시 강북구
서	홍길서	25	kilseo@hong.com	서울시 강서구

6)열 이름으로 열 순서 바꾸기

-DataFrame을 열 이름을 기준으로 정렬하려면 sort_index(axis=1)를 이용.

```
member_df.sort_index(axis=1)
```

	Address	Age	Email	Name
동	서울시 강동구	20	kildong@hong.com	홍길동
서	서울시 강서구	25	kilseo@hong.com	홍길서
남	서울시 강남구	26	south@hong.com	홍길남
북	서울시 강북구	27	book@hong.com	홍길북

7)값으로 정렬

-DataFrame의 값을 기준으로 정렬하려면 sort_values()를 이용.

```
member_df.sort_values(by=["Email"])
```

	Name	Age	Email	Address
북	홍길북	27	book@hong.com	서울시 강북구
동	홍길동	20	kildong@hong.com	서울시 강동구
서	홍길서	25	kilseo@hong.com	서울시 강서구
남	홍길남	26	south@hong.com	서울시 강남구

-여러 열을 이용해서 정렬 하고 싶을 때는 by 인자의 값을 여러 column 이름을 갖는 list 형식으로 지정하면 된다.

8)level로 정렬

-DataFrame의 열 이름 또는 행 이름에 level이 지정되어 있을 경우 level로 정렬할 수 있다.

```
member_df    = pd.read_csv("member_data.csv")
member_df.columns   = [["기본정보","기본정보",  "추가정보",  "추가정보"], ["이름",  "나이",  "이메일", "주소"]]
member_df.columns.names = ["정보구분", "상세정보"]
member_df.index   = [["좌우","좌우","상하","상하"],  ["동", "서", "남", "북"]]
member_df.index.names = ["위치구분", "상세위치"]
member_df
```

정보구분	기본정보		추가정보		
상세정보		이름	나이	이메일	주소
위치구분	상세위치				
좌우	동	홍길동	20	kildong@hong.com	서울시 강동구
	서	홍길서	25	kilseo@hong.com	서울시 강서구
상하	남	홍길남	26	south@hong.com	서울시 강남구
	북	홍길북	27	book@hong.com	서울시 강북구

```
member_df.sort_index(level=["위치구분"])
```

정보구분		기본정보	추가정보		
상세정보		이름	나이	이메일	주소
위치구분	상세위치				
상하	남	홍길남	26	south@hong.com	서울시 강남구
	북	홍길북	27	book@hong.com	서울시 강북구
좌우	동	홍길동	20	kildong@hong.com	서울시 강동구
	서	홍길서	25	kilseo@hong.com	서울시 강서구

```
member_df.sort_index(level=["상세위치"])
```

	정보구분	기본정보	추가정보		
	상세정보	이름	나이	이메일	주소
위치구분	상세위치				
상하	남	홍길남	26	south@hong.com	서울시 강남구
좌우	동	홍길동	20	kildong@hong.com	서울시 강동구
상하	북	홍길북	27	book@hong.com	서울시 강북구
좌우	서	홍길서	25	kilseo@hong.com	서울시 강서구

11. 데이터 그룹화 및 집계

1)Group by

-groupby() 함수는 데이터를 구분 할 수 있는 열(column)의 값들을 이용하여 데이터를 여러 기준에 의해 구분하여 그룹화 한 후 기초 통계 함수 등을 적용 할 수 있도록 한다.

-Syntax

```
DataFrame.groupby(by=None, axis=0, level=None,   as_index=True,
                                    sort=True, group_keys=True,  squeeze=False,
                                    observed=False, **kwargs)

import   statsmodels.api  as sm
iris    =  sm.datasets.get_rdataset("iris",   package="datasets")
iris_df   =  iris.data
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

2)단일 열로 그룹화

-groupby() 함수를 이용하여 그룹화 할 열을 지정.

-그룹이 지정되면 그 그룹에 기초통계 분석 함수를 사용하면 된다.

```
iris_grouped = iris_df.groupby(iris_df.Species)
iris_grouped
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001F477FF7808>

iris_grouped.mean()       #평균
```

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

3)다중 열로 그룹화

-그룹을 두 가지 이상으로 지정하고 싶을 때는 list를 이용해 그룹을 지정하면 된다.

-열 이름에 점(.)이 포함되었다면 df["열이름"] 형식으로 열을 지정해야 한다.

-다음 코드는 iris 데이터를 종(Species)별, 꽃받침 조각의 길이(Sepal.Length)별로 그룹화 하는 예이다.

```
iris_grouped2 =  iris_df.groupby([iris_df.Species,
iris_df["Sepal.Length"]])
iris_grouped2
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001F479369C08>

iris_grouped2.mean().head()
```

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	4.3	3.000000	1.100000	0.100
	4.4	3.033333	1.333333	0.200
	4.5	2.300000	1.300000	0.300
	4.6	3.325000	1.325000	0.225
	4.7	3.200000	1.450000	0.200

-그룹된 객체를 이용해 요약정보를 볼 수도 있다.

```
iris_grouped.describe()
```

-요약정보에서 생략된 부분(...)의 내용을 보려면 디스플레이 옵션을 설정한다.

```
pd.options.display.max_columns  =  999
```

4)그룹간 반복 처리

-그룹화 된 데이터에서 그룹의 타입과 그룹 객체를 반복문을 이용해 처리할 수 있다.

```
for    type, group    in  iris_grouped:
```

```
        print(type, '\n',  group.head())
        ----------------------------------------------
```

5)DataFrame group indexing

-DataFrame group에서 indexing을 위해서는 take() 함수를 이용.

-Syntax

```
DataFrameGroupBy.take(indices, axis, is_copy)
```

-indices : 가져올 index를 list 형식으로 지정.

-axis : 기본값 0이면 행 index를 이용해 가져오고, 1이면 열 index를 이용해 가져온다.

-is_copy : 기본값 True이면 객체의 복사본이 반환.

-다음 코드는 iris 데이터를 불러와 종(Species) 별로 grouping한 결과에서 부분집합을 가져온다.

```
import  statsmodels.api  as sm
iris   = sm.datasets.get_rdataset("iris",  package="datasets")
iris_df  = iris.data
iris_df_grouped  =  iris_df.groupby(iris_df.Species)
```

-다음 코드는 각 종별로 0, 1, 2행을 가져온다.

```
iris_df_grouped.take([0,1,2])
```

-연속적인 index는 range() 함수를 이용할 수 있다.

```
iris_df_grouped.take(range(0,3))
----------------------------------------------
```

		Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Species					
setosa	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
versicolor	50	7.0	3.2	4.7	1.4
	51	6.4	3.2	4.5	1.5
	52	6.9	3.1	4.9	1.5
virginica	100	6.3	3.3	6.0	2.5
	101	5.8	2.7	5.1	1.9
	102	7.1	3.0	5.9	2.1

6)좀 더 쉬운 예제로 Group By를 다뤄보자.

```
student_list = [
      {'Name': 'John', 'Major': "Computer Science", 'Gender': "male"},
      {'Name': 'Nate', 'Major': "Computer Science", 'Gender': "male"},
      {'Name': 'Abraham', 'Major': "Physics", 'Gender': "male"},
      {'Name': 'Brian', 'Major': "Psychology", 'Gender': "male"},
      {'Name': 'Janny', 'Major': "Economics", 'Gender': "female"},
      {'Name': 'Yuna', 'Major': "Economics", 'Gender': "female"},
      {'Name': 'Jeniffer', 'Major': "Computer Science", 'Gender': "female"},
      {'Name': 'Edward', 'Major': "Computer Science", 'Gender': "male"},
      {'Name': 'Zara', 'Major': "Psychology", 'Gender': "female"},
      {'Name': 'Wendy', 'Major': "Economics", 'Gender': "female"},
      {'Name': 'Sera', 'Major': "Psychology", 'Gender': "female"}
]

df = pd.DataFrame(student_list, columns = ['Name', 'Major', 'Gender'])
df
----------------------------------------------------------------
```

	Name	Major	Gender
0	John	Computer Science	male
1	Nate	Computer Science	male
2	Abraham	Physics	male
3	Brian	Psychology	male
4	Janny	Economics	female
5	Yuna	Economics	female
6	Jeniffer	Computer Science	female
7	Edward	Computer Science	male
8	Zara	Psychology	female
9	Wendy	Economics	female
10	Sera	Psychology	female

-전공별 Group By

```
groupby_major = df.groupby('Major')

groupby_major.groups
------------------------------------------------
{'Computer Science': Int64Index([0, 1, 6, 7], dtype='int64'),
 'Economics': Int64Index([4, 5, 9], dtype='int64'),
 'Physics': Int64Index([2], dtype='int64'),
 'Psychology': Int64Index([3, 8, 10], dtype='int64')}
```

-보기 좋게

```
for name, group in groupby_major:
    print(name + ": " + str(len(group)))
```

```
    print(group)
    print()
---------------------------------------------------------
Computer Science: 4
      Name    Major                Gender
0     John        Computer Science     male
1     Nate        Computer Science     male
6  Jeniffer    Computer Science   female
7     Edward      Computer Science     male

Economics: 3
      Name      Major          Gender
4  Janny      Economics    female
5     Yuna       Economics    female
9  Wendy   Economics    female

Physics: 1
      Name      Major            Gender
2  Abraham  Physics     male

Psychology: 3
      Name     Major            Gender
3     Brian       Psychology    male
8     Zara        Psychology    female
10    Sera        Psychology    female
```

-전공별 명수

```
df_major_cnt = pd.DataFrame({'Count':groupby_major.size()})
df_major_cnt
-------------------------------------------------
                          Count
Major
Computer Science     4
Economics            3
Physics              1
Psychology           3
```

-전공도 count와 같이

```
df_major_cnt = pd.DataFrame({'Count':groupby_major.size()}).reset_index()
df_major_cnt
-------------------------------------------------
      Major                Count
0     Computer Science     4
1     Economics            3
2     Physics              1
3     Psychology           3
```

-성별로 group by

```
groupby_gender = df.groupby('Gender')

for name, group in groupby_gender:
    print(name + ": " + str(len(group)))
    print(group)
    print()
-----------------------------------------------
female: 6
      Name      Major                  Gender
4     Janny     Economics          female
5     Yuna      Economics          female
6     Jeniffer    Computer Science     female
8     Zara        Psychology           female
9     Wendy     Economics          female
10    Sera        Psychology           female

male: 5
      Name      Major                  Gender
0     John        Computer Science     male
1     Nate        Computer Science     male
2  Abraham  Physics              male
3     Brian       Psychology           male
7     Edward      Computer Science     male
```

12. DataFrame에 함수 적용하기

-DataFrame의 데이터에서 합계나 평균 등 일반적인 통계는 DataFrame의 함수들을 사용하면 되지만, 판다스에서 제공하지 않는 기능을 커스텀 함수(custom function)로 구현해서 DataFrame에 적용하려면 apply(), applymap(), map() 등의 함수를 사용한다.

-적용할 함수의 이름은 다음과 같다.

method	통용대상	반환값
map	Series(값별)	Series
apply	DataFrame(열 또는 행별)	Series

applymap DataFrame(값별) DataFrame

1)apply()

-DataFrame에 사용자 정의 함수를 적용하기 위해서 apply() 함수를 사용한다.

-Syntax

```
DataFrame.apply(func, axis=0, raw=False,
                                result_type=None, args=(), **kwds)
```

-func : 각 열 또는 행에 적용할 함수.

-axis : 함수가 적용될 축.

--기본값(0 또는 'index')이면 각 열 별로 함수가 적용되며, 1 또는 'columns'이면 각 행 별로 함수가 적용.

-raw

--False(기본값)일 경우 각 행이나 열을 Series로 함수에 전달.

--True이면 전달 된 함수는 대신 ndarray 객체를 받는다.

-result_type : 'expand', 'reduce', 'broadcast', None 중 하나를 사용.

--기본값은 None.

--이것은 axis=1(columns)인 경우에만 작동.

--'expand' : 목록과 같은 결과가 열로 바뀐다.

--'reduce' : 목록과 같은 결과를 확장하지 않고 가능한 경우 Series를 반환한다.

---이것은 '확장'의 반대.

--'broadcast' : 결과가 DataFrame의 원래 모양으로 브로드 캐스팅되고 원본 인덱스와 열은 유지된다.

--None : 적용 함수의 반환 값에 따라 다르다.

---목록 같은 결과는 일련의 결과로 반환된다.

---그러나 apply 함수가 시리즈를 리턴하면 이들은 열로 확장된다.

-args : 배열/시리즈 외에도 func에 전달할 위치 인수를 tuple형식으로 지정.

-**kwds : func에 전달할 추가 키워드 인수를 지정.

```
import statsmodels.api as sm
iris = sm.datasets.get_rdataset("iris", package="datasets")
iris_df = iris.data
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

-iris 데이터에서 종(Species) 정보를 제외한 나머지 열 정보를 조회한다.

-이것은 apply() 함수를 사용하기 전/후를 비교하기 위해서이다.

```
iris_df.iloc[:,:-1].head()
```

	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
4	5.0	3.6	1.4	0.2

-다음 코드는 iris 데이터에 np.round 함수를 적용해서 데이터를 반올림 합니다.

```
import numpy as np
iris_df.iloc[:,:-1].apply(np.round).head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0	5.0	4.0	1.0	0.0
1	5.0	3.0	1.0	0.0
2	5.0	3.0	1.0	0.0
3	5.0	3.0	2.0	0.0
4	5.0	4.0	1.0	0.0

-apply() 함수에 사용하는 함수가 요소별로 동작하는 함수라면 위의 예에서처럼 각 요소에 함수를 적용한다.

-그러나 만일 함수가 요소별로 동작하지 않는 함수라면 axis 매개변수의 값에 따라 적용된 결과는 달라질 수 있다.

-다음 코드는 열 별 합계를 출력한다.

```
iris_df.iloc[:,:-1].apply(np.sum) # 열 별 합계 출력
```

```
Sepal.Length    876.5
Sepal.Width     458.6
Petal.Length    563.7
Petal.Width     179.9
dtype: float64
```

-위의 코드는 다음 lambda을 사용한 코드와 실행결과가 같다.

```
iris_df.iloc[:,:-1].apply(lambda x : np.sum(x))
```

```
Sepal.Length    876.5
Sepal.Width     458.6
Petal.Length    563.7
Petal.Width     179.9
dtype: float64
```

-다음 코드는 행 별 합계를 출력한다.

```
iris_df.iloc[:,:-1].apply(np.sum,  axis=1) # 행 별 합계  출력
-----------------------------------------------------
0                10.2
1                9.5
2                9.4
3                9.4
4                10.2
... 생략  ...
145              17.2
146              15.7
147              16.7
148              17.3
149              15.8
Length: 150, dtype:  float64
```

-다음 코드는 iris 데이터의 열 별 평균을 계산하고 각 데이터와 평균과의 차이를 apply() 함수를 이용해 계산한다.

```
iris_df2 = iris_df.iloc[:,:-1]
iris_avg = iris_df2.apply(np.average)
iris_avg
-------------------------------------------
Sepal.Length          5.843333
Sepal.Width           3.057333
Petal.Length          3.758000
Petal.Width              1.199333
dtype:  float64

iris_df2.apply(lambda x  :  x-iris_avg, axis=1).head()
-------------------------------------------------------------
0    -0.743333    0.442667    -2.358    -0.999333
1    -0.943333    -0.057333 -2.358    -0.999333
2    -1.143333    0.142667    -2.458    -0.999333
3    -1.243333    0.042667    -2.258    -0.999333
4    -0.843333    0.542667    -2.358    -0.999333
```

2)다른 예제로 apply()를 복습해보자.

```
student_list = [
    {'Name':'John', 'Midterm':95, 'Final' : 85},
    {'Name':'Smith', 'Midterm':85, 'Final' : 80},
    {'Name':'Jenny', 'Midterm':30, 'Final' : 10},
]

df = pd.DataFrame(student_list, columns= ['Name', 'Midterm', 'Final'])

df.head()
-------------------------------------------------
   Name      Midterm Final
0  John      95      85
1  Smith     85      80
2  Jenny     30      10

df['Total'] = df['Midterm'] + df['Final']
df
-----------------------------------------
   Name      Midterm Final    Total
0  John      95      85    180
1  Smith     85      80    165
2  Jenny     30      10    40
```

-'Average' column 추가하기

```
df['Average'] = df['Total'] / 2
df.head()
-----------------------------------------
   Name      Midterm Final      Total    Average
0  John      95          85    180        90.0
1  Smith     85          80    165        82.5
2  Jenny     30          10    40       20.0
```

-'Grade' column 추가하기

```
grade_list = []
for row in df['Average']:
    if row <= 100 and row >= 90 :
        grade_list.append('A')
    elif row < 90 and row >= 80 :
        grade_list.append('B')
    elif row < 80 and row >= 70 :
        grade_list.append('C')
    elif row < 70 and row >= 60 :
```

```
            grade_list.append('D')
        else : grade_list.append('F')

df['Grade'] = grade_list
df
-----------------------------------------
   Name    Midterm Final    Total    Average Grade
0  John    95      85       180      90.0    A
1  Smith   85      80       165      82.5    B
2  Jenny   30      10       40       20.0    F
```

-'Result' column 추가하기

```
def pass_or_fail(row):
    if row != 'F':
        return 'Pass'
    else :
        return 'Fail'

df['Result'] = df.Grade.apply(pass_or_fail)
df
-----------------------------------------------
   Name    Midterm Final    Total    Average Grade    Result
0  John    95      85       180      90.0    A        Pass
1  Smith   85      80       165      82.5    B        Pass
2  Jenny   30      10       40       20.0    F        Fail
```

-Column 추가하면서 각각의 값 조작하기

```
date_list = [
    { 'yyyy-mm-dd' : '2019-01-05'},
    { 'yyyy-mm-dd' : '2019-01-10'}
]

df = pd.DataFrame(date_list, columns = ['yyyy-mm-dd'])
df
---------------------------------------
   yyyy-mm-dd
0  2019-01-05
1  2019-01-10

def extract_year(row):
   return row.split('-')[0]

df['Year'] = df['yyyy-mm-dd'].apply(extract_year)
df
-------------------------------------
      yyyy-mm-dd    Year
0     2019-01-05    2019
1     2019-01-10    2019
```

-passing keyword parameter to apply function

```
date_list = [{'Jumin': '2000-06-27'},
       {'Jumin': '2002-09-24'},
       {'Jumin': '2005-12-20'}]
df = pd.DataFrame(date_list, columns = ['Jumin'])
df
-----------------------------------------------------
   Jumin
0  2000-06-27
1  2002-09-24
2  2005-12-20

def extract_year(row):
   return row.split('-')[0]

df['Born_Year'] = df['Jumin'].apply(extract_year)
df
-----------------------------------------------
   Jumin          Born_Year
0  2000-06-27     2000
1  2002-09-24     2002
2  2005-12-20     2005

def calc_age(year, current_year):
   return current_year - int(year)

df['Age'] = df['Born_Year'].apply(calc_age, current_year=2019)
df
-------------------------------------------------
   Jumin          Born_Year    Age
0  2000-06-27     2000         19
```

```
   1  2002-09-24      2002          17
   2  2005-12-20      2005          14

   def get_introduce(age, prefix, suffix):
      return prefix + str(age) + suffix

   df['introduce'] = df['age'].apply(get_introduce, prefix="I am ", suffix=" years old.")
   df
   ------------------------------------------------------------------
      Jumin           Born_Year    Age      Introduce
   0  2000-06-27      2000         19    I am 19 years old.
   1  2002-09-24      2002         17    I am 17 years old.
   2  2005-12-20      2005         14    I am 14 years old.


3)applymap()
   -applymap() 함수는 DataFrame의 함수이지만 apply() 함수처럼 각 행(row, axis=1) 또는 각 열(column, axis=0)별로 작동하는 함수가 아니다.
   -applymap() 함수는 각 요소(element)별로 작동한다.
   -Vector에 scala를 연산하면, 벡터의 요소 하나하나에 해당 연산을 해주는 것처럼 엘리먼트 와이즈(Element wise) 방식으로 적용하는 DataFrame의 각 요소마다 사용자 정의 함수를 수행한다.
   -이때 사용자 정의 함수는 반드시 단일 값(Single value)을 반환해야 한다.
   -Syntax
      DataFrame.applymap(func)

   -앞에서 apply() 함수의 예제에서 사용했던 np.sum() 함수를 applymap() 함수에 적용해 보자.
   -그러면 applymap() 함수가 요소별로 동작하는 것을 확인할 수 있다.

      iris_df.iloc[:,:-1].applymap(np.sum).head()
      ------------------------------------------------------------
         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
      4     5.0             3.6            1.4            0.2

   -이 결과는 각 요소에 대한 합계이므로 원래의 데이터와 같은 값이다.
   -다음 코드는 lambda식을 이용해서 각 요소의 값을 제곱한다.

      iris_df.iloc[:,:-1].applymap(lambda  x  :  x**2).head()
      ------------------------------------------------------
         Sepal.Length  Sepal.Width   Petal.Length   Petal.Width
      0     26.01           12.25          1.96           0.04
      1     24.01           9.00         1.96          0.04
      2     22.09           10.24          1.69           0.04
      3     21.16           9.61         2.25          0.04
      4     25.00           12.96          1.96           0.04

   -한번에 DataFrame에 있는 모든 요소들을 수정하고자 할 때 사용하자.

      data_list = [
         {'x': 5.5, 'y': -5.6},
         {'x': -5.2, 'y': 5.5},
         {'x': -1.6, 'y': -4.5}
      ]
      df = pd.DataFrame(data_list)
      df
      ----------------------------------
         x       y
      0  5.5     -5.6
      1  -5.2    5.5
      2  -1.6    -4.5

      import numpy as np

      df = df.applymap(np.around)  #반올림함수 사용
      df
      ---------------------------------------
         x       y
      0  6.0     -6.0
      1  -5.0    6.0
      2  -2.0    -4.0


4)map()
   -map() 함수는 Series 데이터의 요소 각각에 대해 함수 또는 dict 또는 다른 Series를 적용한다.
   -map() 함수는 DataFrame에는 사용할 수 없다.
   -다음 구문에서 보는 것처럼 Series 타입에서만 사용할 수 있다.
   -Syntax
      Series.map(arg, na_action=None)
   -arg : 시리즈의 값 하나 하나에 적용할 함수, dict 또는 Series.
   -na_action : NA 값이 매핑 함수의 영향을 받는지 여부를 제어.
   -사용 가능한 값은 None 또는 'ignore'이다.
      --'ignore'이면 NA일 경우 함수에 적용하지 않고 NA를 반환한다.
```

--None(기본값)이면 NA는 그대로 함수 또는 딕셔너리에 NA로 전달된다.

-map() 함수는 Series의 값 하나하나에 접근하면서 해당 함수를 수행한다.

```
import pandas as pd
x = pd.Series(['Hello', 'Python', 'World'], index=[1,2,3])
x
-----------------------------------------------------------------
1          Hello
2         Python
3          World
dtype: object
```

-사용자 정의 함수 사용

--사용자 정의 함수를 이 함수는 입력값과 입력값의 길이를 반환한다.

```
def my_func(data):
    return (data, len(str(data)))
```

--Series에 함수를 적용하면 함수 인자에 시리즈의 요소 하나가 전달된다.

```
x.map(my_func)
---------------------------------
1          (Hello, 5)
2         (Python,6)
3          (World, 5)
dtype: object
```

-dict 사용

--Series에 dict를 적용하면 dict의 key별로 Series의 값에 적용된다.

```
z = {"Hello": 'A', "Python": 'B', "World": 'C'}
x.map(z)
1         A
2         B
3         C
dtype: object
```

-Series 사용

-Series에 Series를 적용하면 원본 Series의 값에 적용할 Series의 index별로 적용된다.

```
y = pd.Series(['foo', 'bar', 'baz'], index=['Hello', 'Python', 'World'])
x.map(y)
-----------------------------------------------------------
1         foo
2         bar
3         baz
dtype: object
```

5)다른 예제로 map()을 연습해 보자.

```
date_list = [{'Date': '2000-06-27'},
             {'Date': '2002-09-24'},
             {'Date': '2005-12-20'}]
df = pd.DataFrame(date_list, columns = ['Date'])
df
-------------------------------------------
   Date
0  2000-06-27
1  2002-09-24
2  2005-12-20

def extract_year(date):
    return date.split('-')[0]

type(df['Date'])
-------------------------------------
pandas.core.series.Series

df['Year'] = df['Date'].map(extract_year)
df
-------------------------------------
   Date          Year
0  2000-06-27    2000
1  2002-09-24    2002
2  2005-12-20    2005
```

-map() 응용하기

```
data_list = [
    {'Name':'John', 'Age':15, 'Gender':'male', 'Job':'Student'},
    {'Name':'Smith', 'Age':25, 'Gender':'male', 'Job':'Teacher'},
    {'Name':'Jenny', 'Age':27, 'Gender':'female', 'Job':'Developer'},
]
```

```
df = pd.DataFrame(data_list, columns = ['Name', 'Age', 'Gender', 'Job'])
df
-------------------------------------------------------------
   Name      Age     Gender  Job
0  John      15    male      Student
1  Smith     25    male      Teacher
2  Jenny     27    female    Developer

df.Job = df.Job.map({'Student':1, 'Teacher':2, 'Developer':3})
df
-------------------------------------------
   Name      Age     Gender  Job
0  John      15    male      1
1  Smith     25    male      2
2  Jenny     27    female    3
```

6) na_action

-Series가 NaN 값을 포함할 경우 na_action 인자의 값에 따라 결과는 달라진다.

```
s  =  pd.Series([1,  2,  3,  None])
s
---------------------------------------
0          1.0
1          2.0
2          3.0
3          NaN
dtype:  float64
```

-기본값(na_action=None)일 경우 NA는 그대로 함수의 인자로 전달된다.

```
s.map(lambda   x: (x,   x**2))
-----------------------------------
0          (1.0, 1.0)
1          (2.0, 4.0)
2          (3.0, 9.0)
3          (nan, nan)
dtype:  object
```

-na_action='ignore'일 경우 적용한 결과가 NA가 된다.

```
s.map(lambda   x: (x,   x**2),   na_action='ignore')
-----------------------------------------------
0          (1.0, 1.0)
1          (2.0, 4.0)
2          (3.0, 9.0)
3          NaN
dtype:  obj
```

13. 데이터 전처리

1)fillna()

-fillna() 함수는 주어진 방법으로 NA 또는 NaN값을 채운다.

-Syntax

```
DataFrame.fillna(value=None,  method=None,   axis=None,
                                inplace=False,    limit=None,
                                   downcast=None, **kwargs)
```

-value : scalar, dict, Series, 또는 DataFrame을 지정.

--결측치를 채우는데 사용할 값.

--이 값은 list로 지정할 수 없다.

-method : {'backfill', 'bfill', 'pad', 'ffill', None}

--기본값 None

--재 색인화된 채우기에 사용될 방법을 지정한다.

--'ffill' : 이전의 유효한 관측값을 이용해 채운다.

--bfill : 이후의 유효한 관측값을 사용해서 채운다.

-axis : {0 또는 'index', 1 또는 'columns'}, 축을 지정한다.

-inplace : boolean, 기본값 False, 만일 True 이면 현재 객체를 수정한다.

-limit : int, 기본값 None

--method가 지정되고 있는 경우, 이것은 순방향/역방향으로 채울 수 있는 최대 연속 NaN 개수이다.

--method가 지정되지 않은 경우, 이것은 NaN이 채워지는 축 전체의 최대 항목 수이다.

--없으면 0보다 커야한다.

-downcast : dict, 기본값 None

--항목들을 downcast='infer'일 경우 적당한 동등한 타입으로 형변환(다운캐스트) 된다.

--예를 들면 변수가 float64일 경우 자동으로 int64로 변경된다.

--원래의 값들이 정수이더라도 NaN 값은 실수로 간주되기 때문에 모든 데이터들의 타입이 실수형으로 바뀐다.

--그럴 경우 유용하게 사용될 수 있다.

```
df =  pd.DataFrame([[np.nan,  2,  np.nan,  0], [3,   4,  np.nan,  1],
                                [np.nan,    np.nan,  np.nan,  5],
                                [np.nan,    3,  np.nan,  4]],
                                   columns=list('ABCD'))
df
```

```
------------------------------------------------
          A         B         C       D
0         NaN    2.0     NaN    0
1         3.0       4.0     NaN    1
2         NaN    NaN  NaN    5
3         NaN    3.0     NaN    4
```

-다음구문은 모든 NaN을 0으로 채운다.

```
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
3       0.0     3.0     0.0     4
```

-널(null)이 아닌 이전 또는 이후의 값을 이용해 채울 수 있다.
-ffill 은 이전의 널이 아닌 값을 이용해서 채운다.

```
df.fillna(method='ffill')
-------------------------------------
        A       B       C       D
0       NaN  2.0     NaN  0
1       3.0     4.0     NaN  1
2       3.0     4.0     NaN  5
3       3.0     3.0     NaN  4
```

-모든 열 'A', 'B', 'C', 'D'의 NaN 값을 각각 0, 1, 2, 3 값으로 채운다.

```
values  = {'A': 0, 'B':   1, 'C':   2, 'D':   3}
df.fillna(value=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
3       0.0     3.0     2.0     4
```

-limit는 채울 NaN의 개수를 지정한다.
-limit=1로 하면 다음은 처음의 NaN 값 하나만 채운다.

```
df.fillna(value=values, limit=1)
-------------------------------------------
        A       B       C       D
0       0.0     2.0     2.0     0
1       3.0     4.0     NaN  1
2       NaN  1.0     NaN  5
3       NaN  3.0     NaN  4
```

2)replace()
-replace() 함수는 to_replace 속성에 주어진 값을 value 값으로 바꾼다.
-DataFrame의 값은 다른 값으로 동적으로 대체e된다.
-이것은 .loc 또는 .iloc를 사용하여 업데이트하는 것과는 다르다.
-일부 값으로 업데이트 할 위치를 지정해야 한다.
-Syntax

```
DataFrame.replace(to_replace=None,   value=None,
                                  inplace=False,    limit=None,
                                     regex=False,  method='pad')
```

-to_replace : str, regex, list, dict, Series, int, float, 또는 None, 대체 될 값을 찾는 방법을 숫자(numeric), 문자(str) 또는 정규식(regex)으로 지정한다.
--숫자, 문자 또는 정규식의 리스트로 지정하면 to_replace와 value가 모두 목록인 경우
a. 길이가 같아야 한다.
b. regex=True이면 두 list에 있는 모든 문자열은 정규 표현식으로 해석된다.
--dict를 사용하여 다른 기존 값에 대해 다른 대체 값을 지정할 수 있다.
--예를 들어 {'a':'b', 'y':'z'}는 'a'값을 'b'로, 'y'를 'z'로 바꾼다.
--이 방법으로 dict를 사용하려면 value 매개변수는 None 이어야 한다.
--DataFrame의 경우 dict는 다른 값이 다른 열에서 대체되어야 한다고 지정할 수 있다.
--예를 들어 {'a':1, 'b':'z'}는 'a'열의 값 1과 'b'열의 'z'값을 찾고 value에 지정된 값으로 대체한다.
--이 경우 value 매개변수는 None이 아니어야 한다.
--{'a':{'b':np.nan}}과 같은 중첩 dict의 경우 'a'열에서 'b'값을 찾아 NaN으로 바꾼다.
--이 방법으로 중첩된 dict를 사용하려면 value 매개변수가 None 이어야 한다.
--정규식도 중첩 할 수 있다.
--그러나 열 이름(중첩된 dict의 최상위 dict key)은 정규식이 될 수 없다.
--즉, regex 인수는 문자열, 컴파일된 일반 표현식 또는 list, dict, ndarray 또는 Series와 같은 요소여야 한다.
--value도 None이면 중첩된 dict이나 Series여야 한다.
-inplace : boolean, 기본값 False, 만일 True 이면 현재 객체를 수정.
-limit : int, 기본값 None, method가 지정되고 있는 경우, 이것은 순방향/역방향으로 채울 수 있는 최대 연속 NaN 개수다.
-regex : bool 또는 to_replace와 같은 타입.
--기본값 False : to_replace 또는 value를 정규식으로 해석할지 여부
--이 값이 True이면 to_replace는 문자열이어야 한다.
--또는 정규 표현식 또는 list, dict 또는 정규 표현식의 배열이 될 수 있다.
--이 경우 to_replace는 None이어야 한다.

-method : {‘pad’, ‘ffill’, ‘bfill’, None}
--to_replace가 scala, list 또는 tuple이고 값이 None 인 경우 대체 할 때 사용할 방법이다.

```
s = pd.Series([0, 1, 2, 3, 4])
s.replace(0, 5)
-----------------------------------
0          5
1          1
2          2
3          3
4          4
dtype: int64
```

-다음 구문은 DataFrame 객체에서 0값을 5로 바꾼다.

```
df = pd.DataFrame({'A':[0,  1, 2, 3, 4],
                                'B':  [5,  6, 7, 8, 9],
                                   'C':  ['a',  'b',  'c',  'd',  'e']})
df.replace(0, 5)
-------------------------------------
     A    B    C
0    5    5    a
1    1    6    b
2    2    7    c
3    3    8    d
4    4    9    e
```

-다음 구문은 to_replace를 list로 지정한 예이다.
-DataFrame의 모든 0, 1, 2, 3을 4로 바꾼다.

```
df.replace([0, 1,2, 3],  4)
-----------------------------------
     A    B    C
0    4    5    a
1    4    6    b
2    4    7    c
3    4    8    d
4    4    9    e
```

-다음 구문은 to_replace와 value를 list로 지정한 예이다.
-0, 1, 2, 3을 각각 4, 3, 2,1로 바꾼다.

```
df.replace([0, 1, 2, 3],  [4,  3, 2, 1])
-------------------------------------------------
     A    B    C
0    4    5    a
1    3    6    b
2    2    7    c
3    1    8    d
4    4    9    e
```

-다음 구문은 1과 2 값을 이후의 1 또는 2가 아닌 값으로 채운다.

```
df.replace([1, 2],  method='bfill')
-----------------------------------------
0          0
1          3
2          3
3          3
4          4
dtype: int64
```

-to_replace를 딕셔너리 형식으로 지정한 예이다.
-0 값은 10으로, 1값은 100으로 바꾼다.

```
df.replace({0:  10,  1: 100})
-------------------------------------
        A       B       C
0       10  5       a
1       100     6       b
2       2       7       c
3       3       8       d
4       4       9       e
```

-다음 구문은 A열의 0값과 B열의 5값을 100으로 바꾼다.

```
df.replace({'A':  0, 'B':  5},  100)
-----------------------------------------
     A          B       C
0    100        100     a
1    1          6       b
2    2          7       c
3    3          8       d
4    4          9       e
```

-다음 구문은 A열의 0은 100으로 바꾸고 4는 400으로 바꾼다.

```
df.replace({'A': {0: 100, 4: 400}})
----------------------------------------
     A    B    C
0    100  5    a
1    1    6    b
2    2    7    c
3    3    8    d
4    400  9    e
```

-다음 구문은 to_replace에 정규표현식을 사용한 예이다.

```
df = pd.DataFrame({'A': ['bat', 'foo', 'bait'],
                   'B': ['abc', 'bar', 'xyz']})
```

-다음 구문은 ba로 시작하고 마지막 문자가 임의의 문자인 문자열을 'new'로 바꾼다.

```
df.replace(to_replace=r'^ba.$', value='new', regex=True)
-------------------------------------------------
     A    B
0    new  abc
1    foo  new
2    bait xyz
```

-다음 구문은 A열에서 ba로 시작하고 마지막 문자가 임의의 문자인 문자열을 'new'로 바꾼다.

```
df.replace({'A': r'^ba.$'}, {'A': 'new'}, regex=True)
-------------------------------------------------
     A    B
0    new  abc
1    foo  bar
2    bait xyz
```

-다음 구문은 regex 속성에 정규표현식을 지정한 예이다.

```
df.replace(regex=r'^ba.$', value='new')
-------------------------------------------------
     A    B
0    new  abc
1    foo  new
2    bait xyz
```

-regex 속성에 dict 형식으로 지정하면 정규표현식과 바꿀 값을 같이 지정할 수 있다.

```
df.replace(regex={r'^ba.$': 'new', 'foo': 'xyz'})
-------------------------------------------------
     A    B
0    new  abc
1    xyz  new
2    bait xyz
```

-regex 속성이 list일 경우 value 속성이 지정되어야 한다.

```
df.replace(regex=[r'^ba.$', 'foo'], value='new')
-------------------------------------------
     A    B
0    new  abc
1    new  new
2    bait xyz
```

-여러 개의 논리(bool)값 또는 날짜시간(datetime64) 객체를 바꿀 때 to_replace 매개변수의 데이터 유형이 바꿀 값의 데이터 유형과 일치해야 한다.
-다음 구문을 오류를 발생한다.

```
df = pd.DataFrame({'A': [True, False, True],
                   'B': [False, True, False]})
df.replace({'astring': 'new value', True: False})     # raises
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-2313e57ab11c> in <module>
----> 1 df.replace({'a string': 'new value', True: False})
... 생략 ...
TypeError: Cannot compare types 'ndarray(dtype=bool)' and 'str'
```

-to_replace()의 매개변수 특성을 이해하려면 s.replace({'a': None})와 s.replace('a', None)의 동작을 비교/이해해야 한다.
-만일 다음과 같은 데이터가 있을 경우

```
s = pd.Series([10, 'a', 'a', 'b', 'a'])
```

-s.replace({'a':None})는 s.replace(to_replace={'a': None}, value=None, method=None)와 같다.
-즉 모든 a의 값을 None으로 바꾼다.

```
s.replace({'a': None})
```

```
-----------------------------------------
0                10
1                None
2                None
3                b
4                None
dtype:  object
```

-value=None 이고 to_replace가 scala, list, tuple일 경우 replace() 함수는 method 매개변수에 기본값('pad')을 이용한다.
-그래서 s.replace('a',None)은 s.replace(to_replace='a', value=None, method='pad')와 같다.

```
s.replace('a',  None)
---------------------------------------
0          10
1          10
2          10
3             b
4             b
dtype:  object
```

3)where()
-where() 함수는 하나 이상의 조건에 대한 DataFrame을 확인하고 그에 따라 결과를 반환하는 데 사용된다.
-기본적으로 조건을 만족하지 않는 행은 NaN 값으로 채워진다.
-Syntax

```
DataFrame.where(cond,  other=nan,inplace=False,
                                axis=None,    level=None,   errors='raise',
                                   try_cast=False,  raise_on_error=None)
```

-cond : boolean NDFrame, array-like, 또는 호출가능객체(callable).
--만일 cond가 True이면 원래 값을 유지.
--만일 cond가 False일 경우 other에서 해당 값으로 대체.
--cond가 callable이면 NDFrame에서 계산되고 논리 NDFrame 또는 배열을 반환해야 한다.
--callable 객체는 입력된 NDFrame을 변경하면 안된다.
-other : scalar, NDFrame, 또는 callable.
--cond가 False인 항목은 other의 해당 값으로 대체된다.
--other가 callable 이면 NDFrame에서 계산되고 스칼라 또는 NDFrame을 반환해야 한다.
--callable 객체는 입력 된 NDFrame을 변경하면 안된다.
-inplace : boolean, 기본값 False.
--만일 True 이면 현재 객체를 수정.
-axis : {0 또는 'index', 1 또는 'columns'}, 축을 지정한다.
-level : int, 기본값 None, 정렬 수준을 지정한다.
-errors : str, {'raise', 'ignore'}, 기본값 raise.
--raise : 예외를 발생시킨다.
--ignore : 예외를 억제한다. 오류 시 원본 개체를 반환한다.
-try_cast : boolean, 기본값 False.
--가능한 경우 결과를 입력 유형으로 다시 형 변환 한다.
-raise_on_error : boolean, 기본값 True.
--유효한 데이터타입이 아니면 예외를 발생시킨다.
-where 메소드는 if-then 관용구의 응용 프로그램이다.
-호출하는 DataFrame의 각 요소에 대해 cond가 True이면 요소가 사용된다.
-그렇지 않으면 DataFrame other의 해당 요소가 사용된다.
-DataFrame.where()는 numpy.where()와 다르다.
-df1.where(m, df2)는 np.where(m, df1,df2)와 동일하다.

```
s  =  pd.Series(range(5))
```

-다음 구문은 s 객체에서 0보다 큰 값을 반환하고 그렇지 않은 경우 NaN을 반환한다.

```
s.where(s  >  0)
-----------------------
0          NaN
1          1.0
2          2.0
3          3.0
4          4.0
dtype:  float64
```

-반면, mask() 함수는 해당 조건을 만족하는 데이터에 대해 NaN을 반환한다.

```
s.mask(s   >  0)
-------------------------
0          0.0
1          NaN
2          NaN
3          NaN
4          NaN
dtype:  float64
```

-다음 코드는 s 객체에서 1보다 큰 값을 반환하고 그렇지 않으면 10을 반환한다.

```
s.where(s  >  1, 10)
----------------------------
0          10
1          10
```

```
2          2
3          3
4          4
dtype:  int64
```

-다음 코드는 0부터 10까지(10 포함 안됨) 데이터를 이용해 2열 짜리 DataFrame을 만들고 DataFrame의 값이 3으로 나눈 나머지가 0인 경우 그 값을 반환하며 그렇지 않으면 해당 값을 음수로 반환한다.

```
df  = pd.DataFrame(np.arange(10).reshape(-1, 2),    columns=['A','B'])
m = df % 3   ==   0
df.where(m,  -df)
```

```
----------------------------
     A     B
0    0     -1
1    -2 3
2    -4 -5
3    6     -7
4    –8 9
```

-다음 코드는 DataFrame.where()와 numpy.where()를 비교한 것이다.

```
df.where(m,  -df) == np.where(m,  df,-df)
```

```
-------------------------------------------
     A              B
0    True       True
1    True       True
2    True       True
3    True       True
4    True       True
```

-다음 코드는 where()와 mask() 함수를 비교한 것이다.
-mask() 함수의 조건에 not(~) 연산자가 붙어 있는 것을 확인할 것.

```
df.where(m,  -df)   ==   df.mask(~m, -df)
```

```
-------------------------------------------
     A              B
0    True       True
1    True       True
2    True       True
3    True       True
4    True       True
```

4)dropna

-dropna() 함수는 결측치(누락된 값)를 포함한 행 또는 열을 제거해 준다.

-Syntax

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

-axis : {0 or 'index', 1 or 'columns'}, 기본값 0.

--결측치를 포함하는 행을 제거할 것인지 아니면 열을 제거할 것인지를 지정한다.

--0, 또는 'index' : 누락된 값이 있는 행을 삭제한다.

--1, 또는 'columns' : 누락된 값이 포함 된 열을 삭제한다.

-how : {'any', 'all'}, 기본값 'any'.

--'any' : NA 값이 있는 경우 해당 행 또는 열을 삭제한다.

--'all' : 모든 값이 NA이면 행 또는 열을 삭제한다.

-thresh : int, 선택사항.

--NaN이 아닌 항목의 최소 개수를 지정한다.

--예를 들어 thresh=2 이면 NaN을 포함하더라도 NaN이 최소 2개 이상이면 삭제하지 않는다.

-subset : array-like, 선택사항.

--부분집합을 뽑을 다른 축의 이름이다.

--예: 행을 삭제하면 포함 할 열의 목록이 된다.

-inplace : boolean, 기본값 False.

--만일 True 이면 현재 객체를 수정한다.

```
df =  pd.DataFrame({"name":  ['Alfred',   'Batman',   'Catwoman'], "toy":[np.nan,   'Batmobile',   'Bullwhip'],
                    "born":  [pd.NaT,   pd.Timestamp("1940-04-25"), pd.NaT]})
df
```

```
-------------------------------------------
          name                toy                 born
0         Alfred               NaN                 NaT
1         Batman           Batmobile          1940-04-25
2         Catwoman        Bullwhip            NaT
```

-다음 구문은 최소 하나 이상의 요소가 누락된 값이 있는 행을 제거한다.

```
df.dropna()
```

```
-------------------------
     name        toy              born
1    Batman      Batmobile        1940-04-25
```

-다음 구문은 누락된 값을 포함한 열을 제거해 준다.

```
df.dropna(axis='columns')
```

```
---------------------------------
```

```
          name
0         Alfred
1         Batman
2         Catwoman
```

-다음 구문은 행의 모든 요소가 누락된 값일 경우 행을 제거해 준다.

```
df.dropna(how='all')
-----------------------------------
        name          toy          born
0       Alfred          NaN          NaT
1       Batman      Batmobile  1940-04-25
2       Catwoman   Bullwhip    NaT
```

-누락된 값이 아닌 요소가 최소 2개 이상인 행은 삭제하지 않는다.

```
df.dropna(thresh=2)
----------------------------------------
        name          toy              born
1       Batman      Batmobile     1940-04-25
2       Catwoman   Bullwhip       NaT
```

-누락된 값을 찾을 열을 지정한다.
-다음 구문은 DataFrame에서 toy열은 누락된 값의 유/무를 확인하지 않는다.

```
df.dropna(subset=['name',  'born'])
-----------------------------------------------
        name          toy              born
1       Batman      Batmobile     1940-04-25
```

-누락된 값을 제거하고 현재 DataFrame이 변경된다.

```
df.dropna(inplace=True)
df
-------------------------------
     name          toy              born
1    Batman      Batmobile     1940-04-25
```

5)astype
-Pandas의 객체를 주어진 dtype 속성으로 형변환 한다.
-Syntax
DataFrame.astype(dtype, copy=True, errors='raise', **kwargs)
-dtype : datatype 또는 dict.
--numpy.dtype 또는 Python의 datatype을 사용하여 전체 pandas 객체를 같은 유형으로 형변환 할 수 있다.
--{col: dtype, ...}처럼 dict 형식을 사용했을 경우 col은 열의 이름이고 dtype은 하나 이상의 열을 특정유형으로 형변환하는 numpy.dtype 또는 Python의 datatype 이다.
-copy : bool, 기본값 True.
--copy=True 일 때 복사본을 반환한다.
-errors : str, {‘raise’, ‘ignore’}, 기본값 raise.
--errors=’raise’이면 예외를 발생시킨다.
--errors=‘ignore’ 이면 예외를 억제한다.
--오류 시 원본 개체를 반환한다.
-kwargs : 생성자에 전달할 keyword 인수이다.

```
ser  =  pd.Series([1,  2],  dtype='int32')
ser
-------------------------
0         1
1         2
dtype:  int32

ser.astype('int64')
-----------------------------
0         1
1         2
dtype:  int64
```

-copy=False이면 반환받은 객체를 변경했을 경우 원본 데이터도 같이 변하므로 주의해야 한다.

```
s1 =  pd.Series([1,2])
s2 =  s1.astype('int64',  copy=False)
s2[0] =  10
s1   #  note  that  s1[0] has  changed too
-----------------------------------
0         10
1           2
dtype:  int64
```

6)중복된 값 제거하기

```
data_list = [
  {'Name':'John', 'Gender':'male', 'Job':'Student'},
```

```
    {'Name':'Smith','Gender':'male', 'Job':'Teacher'},
    {'Name':'Jenny','Gender':'female', 'Job':'Developer'},
    {'Name':'Smith','Gender':'male', 'Job':'Teacher'}
]
df = pd.DataFrame(data_list, columns = ['Name', 'Gender', 'Job'])

df.head()
-----------------------------------------------------
     Name   Gender     Job
0    John   male         Student
1    Smith  male         Teacher
2    Jenny  female       Developer
3    Smith  male         Teacher          #중복된 값
```

-중복된 값 확인하기

```
df.duplicated()
-----------------------------------
0    False
1    False
2    False
3    True              #여기가 중복된 값이 있다는 뜻
dtype: bool
```

-중복된 값 제거

```
df.drop_duplicates()
-------------------------------------
   Name   Gender     Job
0  John   male         Student
1  Smith  male         Teacher
2  Jenny  female       Developer
```

7)기타 Null 처리하기

```
student_list = [
    {'Name': 'John', 'Major': 'Computer Science', 'Gender': 'male', 'Age': 40},
      {'Name': 'Nate', 'Major': None, 'Gender': "male", 'Age':35},
            {'Name': 'Abraham', 'Major': 'Physics', 'Gender': 'male', 'Age':37},
            {'Name': 'Brian', 'Major': 'Psychology', 'Gender': 'male', 'Age':None},
            {'Name': 'Janny', 'Major': None, 'Gender': 'female', 'Age':10},
            {'Name': 'Yuna', 'Major': None, 'Gender': 'female', 'Age':12},
            {'Name': 'Jeniffer', 'Major': 'Computer Science', 'Gender': 'female', 'Age':45},
            {'Name': 'Edward', 'Major': 'Computer Science', 'Gender': 'male', 'Age':None},
            {'Name': 'Zara', 'Major': 'Psychology', 'Gender': 'female', 'Age':25},
            {'Name': 'Wendy', 'Major': 'Economics', 'Gender': 'female', 'Age':37},
            {'Name': 'Sera', 'Major': None, 'Gender': 'female', 'Age':None}
    ]
df = pd.DataFrame(student_list, columns = ['Name', 'Major', 'Gender', 'Age'])
df
-----------------------------------------------------------
     Name       Major                Gender     Age
0    John       Computer Science     male         40.0
1    Nate       None                 male         35.0
2    Abraham    Physics              male         37.0
3    Brian      Psychology           male         NaN
4    Janny      None                 female       10.0
5    Yuna       None                 female       12.0
6    Jeniffer   Computer Science     female       45.0
7    Edward     Computer Science     male         NaN
8    Zara       Psychology           female       25.0
9    Wendy      Economics            female       37.0
10   Sera       None                 female       NaN

df.shape
-----------------------------------
(11,4)

df.info()
-------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11 entries, 0 to 10
Data columns (total 4 columns):
Name      11 non-null object
Major     7 non-null object
Gender    11 non-null object
Age       8 non-null float64
dtypes: float64(1), object(3)
memory usage: 432.0+ bytes
```

-Null 확인하기

```
df.isna()
```

```
    ------------------------------------
        Name    Major Gender  Age
    0   False False False     False
    1   False True    False     False
    2   False False False     False
    3   False False False     True
    4   False True    False     False
    5   False True    False     False
    6   False False False     False
    7   False False False     True
    8   False False False     False
    9   False False False     False
    10 False True    False     True

    df.isnull()
    ------------------------------------
        Name    Major Gender  Age
    0   False False False     False
    1   False True    False     False
    2   False False False     False
    3   False False False     True
    4   False True    False     False
    5   False True    False     False
    6   False False False     False
    7   False False False     True
    8   False False False     False
    9   False False False     False
    10 False True    False     True
```

-None값을 다른 값으로 변경하기

```
    df.Age = df.Age.fillna(0)     #숫자 NaN을 0으로
    df.Major = df.Major.fillna('Unknown')   #글자 None을 Unknown으로
    df
    ---------------------------------------------
        Name    Major               Gender    Age
    0   John    Computer Science    male        40.0
    1   Nate    Unknown               male        35.0
    2   Abraham Physics             male        37.0
    3   Brian   Psychology          male        0.0
    4   Janny   Unknown               female      10.0
    5   Yuna    Unknown             female      12.0
    6   Jeniffer Computer Science   female      45.0
    7   Edward  Computer Science    male        0.0
    8   Zara    Psychology          female      25.0
    9   Wendy   Economics           female      37.0
    10 Sera     Unknown             female      0.0
```

8)Unique와 갯수 알아보기

-Unique 즉, 중복제거된 값 알아보기

```
    df.Major.unique()
    ---------------------------------------
    array(['Computer Science', 'Unknown', 'Physics', 'Psychology', 'Economics'], dtype=object)

    df.Gender.unique()
    ---------------------------------------
    array(['male', 'female'], dtype=object)

    df.Major.value_counts()
    ---------------------------------------
    Unknown             4
    Computer Science    3
    Psychology          2
    Physics             1
    Economics           1
    Name: Major, dtype: int64
```

14. DataFrame 합치기

```
    list1 = [
        {'Name': 'John', 'Job': 'Teacher'},
        {'Name': 'Nate', 'Job': 'Student'},
        {'Name': 'Fred', 'Job': 'Developer'}
    ]

    list2 = [
        {'Name': 'Ed', 'Job': 'Dentist'},
        {'Name': 'Jack', 'Job': 'Farmer'},
        {'Name': 'Ted', 'Job': 'Designer'}
    ]
```

```
df1 = pd.DataFrame(list1, columns = ['Name', 'Job'])
df2 = pd.DataFrame(list2, columns = ['Name', 'Job'])
```

-concat()로 합치기

```
result = pd.concat([df1, df2])
result
-----------------------------------
     Name    Job
0    John    Teacher
1    Nate    Student
2    Fred    Developer
0    Ed    Dentist                #0, 1, 2가 반복
1    Jack    Farmer
2    Ted    Designer

result = pd.concat([df1, df2], ignore_index = True)
result
-----------------------------------
     Name    Job
0    John    Teacher
1    Nate    Student
2    Fred    Developer
3    Ed    Dentist
4    Jack    Farmer
5    Ted    Designer
```

-append()로 합치기

```
result = df1.append(df2)
result
----------------------------------------
     Name    Job
0    John    Teacher
1    Nate    Student
2    Fred    Developer
0    Ed    Dentist                  #0, 1, 2가 반복
1    Jack    Farmer
2    Ted    Designer

result = pd.append([df1, df2], ignore_index = True)
result
-----------------------------------
     Name    Job
0    John    Teacher
1    Nate    Student
2    Fred    Developer
3    Ed    Dentist
4    Jack    Farmer
5    Ted    Designer
```

-Column으로 합치기

--두개의 DataFrame의 column이 서로 일치하지 않음.

```
list1 = [
    {'Name': 'John', 'Job': 'Teacher'},
    {'Name': 'Nate', 'Job': 'Student'},
    {'Name': 'Jack', 'Job': 'Developer'}
]

list2 = [
    {'Age': 25, 'Country': 'U.S'},
    {'Age': 30, 'Country': 'U.K'},
    {'Age': 45, 'Country': 'Korea'}
]

df1 = pd.DataFrame(list1, columns = ['Name', 'Job'])
df2 = pd.DataFrame(list2, columns = ['Age', 'Country'])

result = pd.concat([df1, df2], axis=1, ignore_index=True)
result
-------------------------------------------------------------------
     0        1              2     3
0    John    Teacher    25 U.S
1    Nate    Student    30 U.K
2    Jack    Developer  45 Korea
```

15. 기초 통계 분석

1)Pandas는 데이터를 보다 좀 더 편하게 다룰 수 있게 하는 데이터 구조 측면에서의 장점을 가진 패키지이다.

2)Pandas에서 제공하는 통계분석은 기본적인 기술통계 및 데이터 요약이다.

3)고급 통계 기법을 사용하고 싶다면 Scikit-learn 이나 다른 통계 패키지를 이용하여 수행 할 수 있다.

4)기술통계함수 목록

-count :NA를 제외한 개수
-min : 최솟값
-max : 최댓값
-sum : 합
-cumprod : 누적합
-mean : 평균
-median : 중앙값
-quantile : 분위수
-corr : 상관관계
-var : 표본분산
-std : 표본 정규분산

-다음 코드는 기술 통계량을 확인해 보기 위해 데이터를 불러오자.
-Statsmodels 패키지를 이용해 iris 데이터를 불러온다.

```
import   statsmodels.api  as sm
iris   =  sm.datasets.get_rdataset("iris",   package="datasets")
iris_df   =  iris.data
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
```

5)최솟값, 최댓값, 평균, 중위수
-Syntax
DataFrame.min(axis=None, skipna=None, level=None, numeric_only=None, **kwargs)
-axis : 0이면 index, 1이면 columns를 의미다.
-skipna : True(기본값)이면 NA 또는 null 값을 계산에서 제외한다.
-level : 다중 index일 경우 level을 지정한다.
-numeric_only : True일 경우 float, int, boolean 유형의 열들만 포함시킨다.
--기본값 None은 모든 열에 대해 연산을 시도한다.
--이 인자는 시리즈(Series)는 지원하지 않는다.

```
iris_df.min()
```

```
-------------------------
Sepal.Length                4.3
Sepal.Width                 2
Petal.Length                1
Petal.Width                   0.1
Species                       setosa
dtype:   object
```

```
iris_df.max()
```

```
-------------------------
Sepal.Length                7.9
Sepal.Width                 4.4
Petal.Length                6.9
Petal.Width                   2.5
Species                       virginica
dtype:   object
```

```
iris_df.mean()
```

```
---------------------------
Sepal.Length          5.843333
Sepal.Width           3.057333
Petal.Length          3.758000
Petal.Width              1.199333
dtype:   float64
```

```
iris_df.median()
```

```
-------------------------
Sepal.Length          5.80
Sepal.Width           3.00
Petal.Length          4.35
Petal.Width              1.30
dtype:   float64
```

6)자세한 내용는 pandas documentation API Reference를 참조한다.
-Series Computations / Descriptive Stats : http://pandas.pydata.org/pandas-docs/stable/api.html#computations-descriptive-stats
-DataFrame Computations / Descriptive Stats : http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-stats

7)요약 통계량
-describe() 함수는 요약 통계량을 출력한다.
-요약 통계량은 데이터의 개수, 평균, 표준편차, 최솟값, 25%, 50%, 75%, 그리고 최댓값 정보이다.
-Syntax
DataFrame.describe(percentiles=None, include=None, exclude=None)

-percentiles : 출력에 포함될 백분위 수를 0~1사이의 값으로 지정한다.
--기본 값은 [.25, .5, .75].
--이것은 25%, 50%, 75% 위치 데이터를 출력한다.
-include : 출력에 포함될 데이터의 유형을 지정한다.
--None(기본값) 이면 모든 숫자타입 열들을 출력에 포함시킨다.
--"all"이면 모든 열을 포함한다.
--정수형이면 "int64", 논리형이면 "bool", 실수형이면 "float64" 등으로 지정한다.
-exclude : 출력에서 제외할 데이터의 유형을 지정한다.
--None(기본값) 이면 아무것도 제외시키지 않는다.
-숫자 데이터의 경우 결과의 인덱스에는 count, mean, std, min, max 및 하위 백분위 수, 상위 백분위 수 및 상위 백분율이 포함된다.
-기본적으로 하위 백분위 수는 25이고 상위 백분위 수는 75이다.
-50 백분위 수는 중앙값과 같다.
-객체 데이터(예: 문자열 또는 timestamp)의 경우 결과 색인에 count, unique, top 그리고 freq가 포함된다.
-top가 가장 일반적인 값이다.
-freq는 가장 일반적인 값의 빈도수이다.
-timestamp는 첫 번째 요소와 마지막 요소도 포함한다.
-가장 높은 count를 갖는 값이 여러 개 일 경우, count와 top 결과는 가장 높은 count를 갖는 값 중에서 임의로 선택된다.
-DataFrame을 통해 제공되는 혼합 데이터 유형의 경우 기본값은 숫자 열의 분석만 반환한다.
-DataFrame이 숫자 열이 없는 개체 및 범주 데이터로만 구성된 경우 기본값은 개체열과 범주 형 열 모두의 분석을 반환하는 것이다.
-include='all' 매개변수가 제공되면 결과에는 각 유형의 속성이 결합된다.

8)기본 요약 통계량
-다음 코드는 iris 데이터의 요약 통계량을 출력한다.
-iris 데이터의 요약 통계량에는 종(Species) 정보는 출력되지 않는다.
-기본적으로 숫자 데이터의 요약 통계량이 출력된다.

```
iris_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        0.828066     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
```

-다음은 종(Species) 정보의 요약 통계량을 출력한다.
-count(전체 데이터의 수), unique(데의 종류), top(가장 많은 요소), freq(가장 많은 요소의 수)를 출력한다.

```
iris_df.Species.describe()
----------------------------------------
count              150
unique               3
top          virginica
freq                50
Name: Species, dtype: object
```

9)include와 exclude
-다음 데이터의 요약 통계량을 출력하면 a열과 b열의 요약 통계량만 출력된다.
-b열은 논리값을 가지므로 기본요약 통계량 출력에서 제외된다.

```
df = pd.DataFrame({'a': [1, 2] * 3, 'b': [True, False] * 3, 'c': [2.0, 4.0] * 3})
df.describe()
----------------------------------------
              a         c
count  6.000000  6.000000
mean   1.500000  3.000000
std    0.547723  1.095445
min    1.000000  2.000000
25%    1.000000  2.000000
50%    1.500000  3.000000
75%    2.000000  4.000000
max    2.000000  4.000000
```

-include 및 exclude 매개변수를 사용하여 DataFrame에서 출력용으로 분석되는 열을 제한할 수 있다.
-Series를 분석 할 때 매개변수는 무시된다.
-다음은 정수유형 열에 대해서만 요약통계량을 출력한다.
-int64 유형을 include 시키거나 나머지 유형들을 exclude 시키면 된다.
-앞의 a,b,c열을 갖는 데이터에서 다음 두 구문은 같은 결과를 출력할 것이다.

```
df.describe(include=["int64"])
df.describe(exclude=["bool", "float64"])
----------------------------------------------------
              a
count  6.000000
mean   1.500000
std    0.547723
min    1.000000
25%    1.000000
50%    1.500000
```

```
       75%        2.000000
       max        2.000000
```

-모든 요소에 대해 요약통계량을 출력하려면 include='all'을 이용한다.

```
       df.describe(include='all')
       ---------------------------------------
                    a                  b          c
       count     6.000000        6          6.000000
       unique    NaN             2          NaN
       top          NaN             True    NaN
       freq      NaN             3          NaN
       mean      1.500000        NaN     3.000000
       std          0.547723        NaN     1.095445
       min          1.000000        NaN     2.000000
       25%       1.000000        NaN     2.000000
       50%       1.500000        NaN     3.000000
       75%       2.000000        NaN     4.000000
       max       2.000000        NaN     4.000000
```

-다음처럼 include와 exclude에 같은 유형을 사용하면 오류가 발생한다.

```
       df.describe(include=["int64"],  exclude=["int64",   "float64"])
       ---------------------------------------------------------------------------
       ValueError                                Traceback (most   recent   call   last)
       <ipython-input-64-52780e7f55bd>  in <module>()
       ...
```

10)분산,표준편차

-var()는 분산(variance)을, std()는 표준편차(standard deviation)를 계산한다.

-Syntax

```
   DataFrame.var(axis=None,  skipna=None, level=None,   ddof=1,
                              numeric_only=None,   **kwargs)
   DataFrame.std(axis=None,  skipna=None, level=None,   ddof=1,
                              numeric_only=None,   **kwargs)
```

-ddof : 델타 자유도(Delta Degree of Freedom)를 지정한다.

--기본값은 1이다.

--계산에 사용되는 제수는 N-ddof이다.

--여기서 N은 요소의 수를 나타낸다.

```
   iris_df.var()
   -------------------------------
   Sepal.Length          0.685694
   Sepal.Width           0.189979
   Petal.Length          3.116278
   Petal.Width              0.581006
   dtype:   float64

   iris_df.std()
   -------------------------------
   Sepal.Length          0.828066
   Sepal.Width              0.435866
   Petal.Length          1.765298
   Petal.Width              0.762238
   dtype:   float64
```

11)공분산(covariance), 상관계수(correlation)

-cov()는 각 열들의 공분산 쌍을 계산한다.

-corr()는 각 열들의 상관계수 쌍을 계산한다.

-Syntax

```
   DataFrame.cov(min_periods=None)
   DataFrame.corr(method='pearson', min_periods=1)
```

-min_periods : 유효한 결과를 얻기 위해 열 쌍당 필요한 최소 관측 수를 지정한다.

-method : 상관계수를 계산할 방법을 지정한다.

--"pearson", "kendall", "spearman" 중 하나를 지정할 수 있다.

```
   iris_df.cov()
   ----------------------------
                        Sepal.Length     Sepal.Width   Petal.Length   Petal.Width
   Sepal.Length      0.685694          -0.042434    1.274315       0.516271
   Sepal.Width       -0.042434      0.189979       -0.329656    -0.121639
   Petal.Length      1.274315          -0.329656    3.116278       1.295609
   Petal.Width          0.516271          -0.121639    1.295609       0.581006

   iris_df.corr()
   -------------------------------
                        Sepal.Length  Sepal.Width   Petal.Length   Petal.Width
   Sepal.Length    1.000000       -0.117570    0.871754       0.817941
   Sepal.Width     -0.117570    1.000000       -0.428440    -0.366126
   Petal.Length    0.871754       -0.428440    1.000000       0.962865
   Petal.Width        0.817941       -0.366126    0.962865       1.000000
```

12)통계 산출하기 복습

-Series나 DataFrame에는 일반적인 수학적, 통계학적 계산을 실행하는 method를 사용할 수 있다.

```
df = pd.read_csv('pandas_data/sungjuk_utf8.csv', header = None,
      names = ['학번', '이름', '국어', '영어', '수학', '전산'])
df
```

```
   학번     이름   국어  영어  수학  전산
0  1101   한송이  78  87  83  78
1  1102   정다워  88  83  57  98
...
...
```

```
df.loc[:, '국어':'전산'].mean()
```

```
국어    84.916667
영어    80.416667
수학    75.333333
전산    79.750000
dtype: float64
```

-Series에서도 같은 method를 사용할 수 있다.

```
df['국어'].sum()
```

```
1019
```

-percent 표시는 백위수 값(전체를 100으로 작은 쪽부터세어서 몇 번째가 되는지 나타내는 수치이다. 50 백분위수가 중앙값이다)이다.

```
df.describe().round(1)    #round() 반올림함수
```

```
        국어    영어    수학    전산
count  12.0   12.0   12.0   12.0
mean   84.9   80.4   75.3   79.8
std    10.7   15.3   15.2   11.6
min    68.0   56.0   53.0   55.0
25%    77.5   67.0   61.5   76.8
50%    87.5   85.0   80.5   78.0
75%    91.2   91.0   87.2   88.0
max    98.0   99.0   93.0   98.0
```

-백분위수 값을 변경할 경우에는 keyword 인수 percentiles의 list 요소에 1이상의 소수 값을 지정한다.

```
df.describe(percentiles = [0.1, 0.9]).round(1)
```

```
        국어    영어    수학    전산
count  12.0   12.0   12.0   12.0
mean   84.9   80.4   75.3   79.8
std    10.7   15.3   15.2   11.6
min    68.0   56.0   53.0   55.0
10%    68.8   58.0   53.4   66.7
50%    87.5   85.0   80.5   78.0
90%    98.0   97.0   92.5   88.9
max    98.0   99.0   93.0   98.0
```

-위의 예에서는 2개의 값을 지정하고 있지만 3개 이상 지정하는 것도 가능하다.
-DataFrame에 대해서 통계적인 연산을 하는 method를 실행한 경우에는 수치형의 열이 대상이 된다.
-비수치열에 대해 describe()를 사용하는 경우에는 다음과 같은 기본 통계량이 산출된다.
  --count : 결손값을 제외한 data 수
  --unique : unique한 data 수
  --top : data의 수가 가장 많은 값
  --freq : top의 data 수

```
df[['학번', '이름']].describe()
```

```
          학번    이름
count     12    12
unique    12    12
top     1102  한산섬
freq       1     1
```

-논리값으로 data 추출하기

```
df.loc[df['국어'] > 90].head()
```

```
     학번     이름   국어  영어  수학  전산
5    1106   튼튼이  98  97  93  88
7    1108   더크게  98  67  93  78
10  1111   한산섬  98  89  73  78
```

```
df.query('학번 == 1104')
```

```
     학번     이름   국어  영어  수학  전산
3    1104   고아라  83  57  88  73
```

-where method로 data 추출하기

```
df.where(df['영어'] < 70)
```

	학번	이름	국어	영어	수학	전산
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN
2	1103.0	그리운	76.0	56.0	87.0	78.0
3	1104.0	고아라	83.0	57.0	88.0	73.0
4	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN
6	1107.0	한아름	68.0	67.0	83.0	89.0
7	1108.0	더크게	98.0	67.0	93.0	78.0
8	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN	NaN	NaN

-값 변경하기

```
df.head(3)
```

	학번	이름	국어	영어	수학	전산
0	1101	한송이	78	87	83	78
1	1102	정다워	88	83	57	98
2	1103	그리운	76	56	87	78

```
df.loc[1, '전산'] = np.nan

df.loc[1, '전산']
```

```
nan
```

```
df.head(3)
```

	학번	이름	국어	영어	수학	전산
0	1101	한송이	78	87	83	78.0
1	1102	정다워	88	83	57	NaN
2	1103	그리운	76	56	87	78.0

-복수의 값 변경

```
df.loc[df['학번'] > 1110, '수학'] = np.nan

df.tail(2)
```

	학번	이름	국어	영어	수학	전산
10	1111	한산섬	98	89	NaN	78.0
11	1112	하나로	89	97	NaN	88.0

-결손값 제외하기

```
df.loc[df['수학'].isnull()]
```

	학번	이름	국어	영어	수학	전산
10	1111	한산섬	98	89	NaN	78.0
11	1112	하나로	89	97	NaN	88.0

-결손값이 포함되어 있는 data 제외

```
df.dropna().loc[8:]   #8번째 이후 data 중 NaN값이 있는 data제외
```

	학번	이름	국어	영어	수학	전산
8	1109	더높이	88	99	53.0	88.0
9	1110	아리랑	68	79	63.0	66.0

-dropna()는 비파괴적 조작이다.
-따라서 df에는 이전 data가 남아있다.
-DataFrame의 내용을 파괴적으로 다시 쓰는 경우

```
df.dropna(inplace = True)
```

-Data 형
--Series나 DataFrame은 작성된 시점에 data형이 자동으로 설정된다.
--수치 data는 NumPy의 data형이 저장되고, 문자열 등의 data는 object 형으로 취급된다.
--Series의 data 형을 확인하는 경우
--Series의 data 형을 확인할 때에는 dtype을 참조한다.

```
df['국어'].dtype
```

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

-DataFrame의 data 형을 확인하는 경우
-DataFrame의 data 형을 확인할 때는 dtypes를 참조한다.

```
df.dtypes
---------------------------------
학번      int64
이름      object
국어      int64
영어      int64
수학      float64
전산      float64
dtype: object
```

-형을 변환하는 경우에는 astype() 을 사용한다.
-인수에는 type형 또는 NumPy의 data 형을 지정한다.

```
df['학번'].astype(np.str)
-------------------------------------------
0    1101
1    1102
2    1103
3    1104
...
...
10   1111
11   1112
Name: 학번, dtype: object
```

-복수열의 형을 변경하는 경우
-인수에 사전을 지정한다.

```
df.astype({'영어':np.float64, '수학':np.str})

df.dtypes
-----------------------------------------------------
학번      int64
이름      object
국어      int64
영어      int64
수학      float64            #변경되지 않음. 비파괴적이어서...
전산      float64
dtype: object
```

-DataFrame을 다시 쓰는 경우

```
df['수학'] = df['수학'].astype(np.str)
df.dtypes
-----------------------------------------------------
학번      int64
이름      object
국어      int64
영어      int64
수학      object          #변경됐음.
전산      float64
dtype: object
```

-Sort 하기

```
del df
df = pd.read_csv('sungjuk_utf8.csv', header = None,
       names = ['학번', '이름', '국어', '영어', '수학', '전산'])

df.sort_values('국어', ascending=False)
-------------------------------------------------------------
      학번          이름      국어   영어   수학   전산
5     1106    튼튼이 98 97 93 88
7     1108    더크게 98 67 93 78
10 1111    한산섬 98 89 73 78
...
...
```

-sort_values() 역시 비파괴적 조작이다.
-덮어쓰려면 inplace에 True를 할당한다.

16. 다양한 data 불러오기
  1)Pandas는 다음과 같이 다양한 형식의 data를 불러올 수 있다.
    -CSV
    -Excel
    -Database
    -JSON
    -MessagePack
    -HTML
    -Google BigQuery
    -Clipboard

-Pickle
-공공데이터포털 : https://www.data.go.kr/
-기타(http://pandas.pydata.org/pandas-docs/stable/io.html)

17. CSV file 불러오기
1)pandas.read_csv() 함수를 사용한다.
2)첫번째 인수에 file 경로를 넘겨주면 DataFrame 형 object를 넘겨준다.
3)File 경로 또는 URL 형식으로 지정할 수 있다.

```
import pandas as pd

df = pd.read_csv('friend_list.csv')
df.head()
```

```
-----------------------------------------------------
      name    age      job
0     John       20    student
1     Jenny      30    developer
2     Nate       30    teacher
3     Julia      40    dentist
4     Brian    45    manager
```

4)DataFrame.head()는 앞에서 5행분의 DataFrame을 넘겨준다.
5)인수에 정수값을 넘기는 방식으로 행수를 지정할 수 있다.

```
df = pd.read_csv('friend_list.csv')
df.head(2)
```

```
---------------------------------
   nameage      job
0  John     20    student
1  Jenny    30    developer
```

6)뒤에서부터 읽을 행의 갯수를 지정할 수도 있다.

```
df = pd.read_csv('friend_list.csv')
df.tail(2)
```

```
----------------------------------------
      name    age      job
4     Brian    45    manager
5     Chris    25    intern
```

7)각 열은 Series이다.

```
type(df.job)
```

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

8)지정한 열을 DataFrame의 index로 하기
-아래 예제처럼 keyword 인수 index_col에 수치 또는 열 이름을 지정하여 지정한 열을 DataFrame의 index로 한다.

```
# index로 지정할 열을 번호로 지정
df = pd.read_csv('friend_list.csv', index_col = 0)
df.head()
```

```
-----------------------------------------------------------------
            age      job
name
John        20    student
Jenny       30    developer
Nate        30    teacher
Julia       40    dentist
Brian    45    manager
```

```
#index로 지정할 열을 열 이름으로 지정
df = pd.read_csv('friend_list.csv', index_col = 'job')
df.head()
```

```
-------------------------------------------------------
                name    age
job
student         John        20
developer       Jenny       30
teacher     Nate        30
dentist         Julia       40
manager         Brian    45
```

9)지정한 열을 지정한 형으로 불러오기
-Keyword 인수 dtype에 열 이름(key)과 형(값)을 사전형으로 지정하여 지정한 열을 지정한 형으로 불러올 수 있다.

```
#형 지정
```

```
df = pd.read_csv('friend_list.csv', dtype={'age':float})
df.head()
-------------------------------------------
        name    age     job
0       John      20.0    student
1       Jenny     30.0    developer
2       Nate      30.0    teacher
3       Julia     40.0    dentist
4       Brian   45.0    manager
```

10)형식이 유사한 txt file 읽어오기
-CSV file처럼 txt file도 각 열의 구분을 ','로 할 경우

```
df = pd.read_csv('friend_list.txt')
df.head()
-------------------------------------------
        name    age     job
0       John      20    student
1       Jenny     30    developer
2       Nate      30    teacher
3       Julia     40    dentist
4       Brian   45    manager
```

11)만일 file의 column들이 쉼표로 구분되어 있지 않은 경우
-delimiter parameter에 구분자를 지정해서 column을 나눠야 한다.

```
df = pd.read_csv('friend_list_tab.txt')
df.head()
-------------------------------------------
        nameage   job
0  John\t20\tstudent
1  Jenny\t30\tdeveloper
2  Nate\t30\tteacher
3  Julia\t40\tdentist
4  Brian\t45\tmanager     #구분이 어려움.

df = pd.read_csv('friend_list_tab.txt', delimiter = '\t')
df.head()
-------------------------------------------
        name    age     job
0       John      20    student
1       Jenny     30    developer
2       Nate      30    teacher
3       Julia     40    dentist
4       Brian   45    manager
```

12)file에 data header가 없는 csv file을 사용할 때
-만일 data header가 없으면, header = None으로 지정해야 첫번째 data가 data header로 들어가는 것을 막을 수 있다.

```
df = pd.read_csv('friend_list_no_head.csv')
df.head()
-----------------------------------------------------------------
        John    20 student          #첫 번째 행이 header가 돼버림.
0       Jenny   30 developer
1       Nate    30 teacher
2       Julia   40 dentist
3       Brian 45 manager
4       Chris 25 intern

df = pd.read_csv('friend_list_no_head.csv', header = None)
df
-------------------------------------------------------
     0       1     2
0    John    20 student
1    Jenny   30 developer
2    Nate    30 teacher
3    Julia   40 dentist
4    Brian 45 manager
5    Chris 25 intern
```

-만일 header가 없는 data를 호출했을 경우, DataFrame 생성 후, column header를 지정할 수 있다.

```
df.columns = ['Name', 'Age', 'Job']
df.index = ['1101', '1102', '1103', '1104', '1105', '1106']
df
-------------------------------------------
           Name      Age     Job
1101    John      20    student
1102    Jenny     30    developer
1103    Nate      30    teacher
1104    Julia     40    dentist
```

```
1105    Brian    45    manager
1106    Chris    25    intern
```

-file을 열 때 동시에 header에 column을 지정해야 할 경우

```
df = pd.read_csv('friend_list_no_head.csv', header = None, names=['Name', 'Age', 'Job'])
df.head()
---------------------------------------
위의 결과와 동일
```

13)외부 CSV file 읽기
-https://github.com/vincentarelbundock/Rdatasets/tree/master/csv/datasets

14)기타 option
-read_csv()에는 다수의 option이 준비되어 있다.
-상세한 문서의 내용은 문서를 참조한다.
-http://panda.pydata.org/pandas-docs/stable/io.html#io-read-csv-table

15)DataFrame csv file로 저장하기

```
import pandas as pd

user_list = [
    {'Name':'John', 'Age':25, 'Gender' : 'male', 'Address':'Chicago'},
    {'Name':'Smith', 'Age':35, 'Gender' : 'male', 'Address':None},
    {'Name':'Jenny', 'Age':45, 'Gender' : 'female', 'Address':'Dallas'}
]

df = pd.DataFrame(user_list)

df = df[['Name', 'Age', 'Gender','Address']]

df.head()
---------------------------------------
Name        Age    Gender    Address
0  John     25    male      Chicago
1  Smith    35    male      None
2  Jenny    45    female    Dallas

df.to_csv('user_list.csv')
```

-기본적으로 to_csv()는 index = True, header = True가 설정되어 있다.
-만일 index = False로 설정할 경우

```
df.to_csv('user_list.csv', index=False)
```

-각 행의 index 즉 0, 1, 2가 사라진 csv file이 생성된다.
-또한 header = False로 설정하면 각 열의 header가 없어진다.

-Smith의 거주지가 None이기 때문에 빈칸으로 값이 들어간다.
-만일 csv로 file 저장시 빈칸대신 '-'를 넣어서 뭔가 있다는 것을 설정하려면,

```
df.to_csv('user_list.csv', na_rep = '-')
```

-이렇게 설정하면 해당 칸에는 '-'가 들어가게 된다.

18. Excel file 불러오기
1)pandas.read_excel() 함수를 사용한다.
2)사전준비를 위해 xlrd module을 설치여부를 확인한다.

```
!conda list | grep xlrd    #Windows에서는 사용 불가

$ conda install -y xrld==1.2.0   #설치안되어 있으면 설치

# Excel file 불러오기
df = pd.read_excel('재무실적.xlsx')
df.head()
```

3)불러오는 sheet 지정하기
-기본 설정으로는 첫 번째 sheet를 불러온다.
-Sheet 이름을 지정해서 불러올 때는 keyword 인수 sheet_name에 sheet이름을 지정한다.

```
df = pd.read_excel('재무실적.xlsx', sheet_name = 'data2')
df.head()
```

19. SQL을 사용해서 불러오기
1)pandas.read_sql() 함수를 사용한다.

2)첫번째 인수에 query를 실행하는 SQL문, 두번째 인수에 SQLAlchemy(http://docs.sqlalchemy.org/en/latest/dialects/index) 또는 DBAPI2(PEP 249, Python Database API Specification v2.0, https://www.python.org/dev/peps/pep-0249)의 접속 instance를 넘겨준다.

3)MariaDB with Python

-설치여부 확인하기

```
!conda list | grep mysql-connector-python
#Windows에서는 grep 명령어 사용 불가
```

-mysql-connector-python 설치하기

--In Anaconda Prompt,

```
$ conda install -y mysql-connector-python

import mysql.connector as mariadb

mariadb_connection = mariadb.connect(user='root', password='javamariadb', host='localhost', database='world')
cursor = mariadb_connection.cursor()

cursor.execute("SELECT ID, Name, CountryCode, District, Population FROM city WHERE CountryCode='KOR'")

for ID,Name,CountryCode,District,Population in cursor:
    print('ID = %d, Name = %s, CountryCode = %s, District = %s, Popluation = %d' % (ID, Name,
    CountryCode,District,Population))
-------------------------------------------------------
ID = 2331, Name = Seoul, CountryCode = KOR, District = Seoul, Popluation = 9981619
ID = 2332, Name = Pusan, CountryCode = KOR, District = Pusan, Popluation = 3804522
ID = 2333, Name = Inchon, CountryCode = KOR, District = Inchon, Popluation = 2559424
ID = 2334, Name = Taegu, CountryCode = KOR, District = Taegu, Popluation = 2548568
ID = 2335, Name = Taejon, CountryCode = KOR, District = Taejon, Popluation = 1425835
...
...
mylist = []
for ID,Name,CountryCode,District,Population in cursor:
    list = []
    list.append(ID);  list.append(Name);  list.append(CountryCode)
    list.append(District);    list.append(Population)
    mylist.append(list)

df = pd.DataFrame(data = mylist, columns = ['ID', 'Name', 'CountryCode', 'District','Population'])
print(df)
```

4)Oracle with Python

-Oracle cx_oracle 7

-https://www.oracle.com/database/technologies/appdev/python.html

-Install on Windows

```
$ python -m pip install cx_Oracle --upgrade
```

-In Anaconda Prompt

```
$ conda install cx_oracle
```

-Connection

```
import cx_Oracle

--conn = cx_Oracle.connect('hr', 'hr', 'localhost:1521/XE')

--conn1 = cx_Oracle.connect('scott/tiger@localhost:1521/XE')

print(conn1)
------------------------------------------------------------
<cx_Oracle.Connection to hr@localhost:1521/XE>

--dsn_tns = cx_Oracle.makedsn('localhost', 1521, 'XE')
print(dsn_tns)
------------------------------------------------------------
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNECT_DATA=(SID=XE)))

conn2 = cx_Oracle.connect('scott', 'tiger', dsn_tns)
print(conn2)
-------------------------------------------------
<cx_Oracle.Connection to
hr@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNECT_DATA=(SID=XE)))>

conn.version
---------------------------
'11.2.0.2.0'
```

-Cursor Objects

```
conn = cx_Oracle.connect('hr', 'hr', 'localhost:1521/XE')
cursor = conn.cursor()
sql = """SELECT employee_id, first_name, salary, to_char(hire_date, 'yyyy-mm-dd'), department_name, city
          from employees e inner join departments d on e.department_id = d.department_id
```

```
            inner join locations l on d.location_id = l.location_id"""
cursor.execute(sql)

for employee_id, first_name, salary, hire_date, department_name, city in cursor :
    print(employee_id, first_name, salary, hire_date, department_name, city)

cursor.close()
```