왜 파이선인가

파이선 소개

파이선은 1980년대 말 Guido van Rossum이 만든 오픈 소스 범용 프로그래밍 언어이다. 자바나 C 언어에 비해 배우기 쉽고, 기능도 충분하여 여러 분야에서 사용되고 있으며 최근 대학에서는 기본 언어로 파이선을 주로 채택하고 있다.

데이터 사이언스 분야나 고속 데이터 처리 분야에서 특히 파이선이 장점을 갖고 있는데, 예전에는 이러한 기능을 주로 C나 자바로 구현했었다. 파이선의 가장 큰 장점은 C나 자바에 비해 쉽다는 것이다. 그리고 이미 개발되어 있는 라이브러리를 불러서 사용함으로써 파이선의 느린 처리속도를 충분히 개선할 수 있다. 파이선 자체는 속도가 느리지만 이미 라이브러리화 되어 있는 모듈을 호출하면 속도 문제를 해결할 수 있다.

여기서 파이선이 속도가 느리다고 하는 이유는 파이선의 동작이 기본적으로 대화형 속성(interpreted nature) 모드로 동작하기 때문인데, 대화형 모드란 소스 코드를 컴파일하는 과정이 없이 코드의 동작을 바로 라인 단위로 실행하는 것을 말한다. 이러한 대화형 속성은 프로그램 개발은 쉽고, 직관적이며 빠르게 할 수 있다는 장점이 있다.

즉, 파이선은 문법도 쉽고, 편리한 개발 방법을 제공며, 동시에 다른 사람들이 만든 라이브러리가 풍부하여 실행시에는 빠른 처리속도를 얻는 장점을 가지고 있다.

파이선의 대표적인 라이브러리는 NumPy인데 이 라이브러리는 어레이를 편리하게 다룰 수 있는 기능을 제공한다. 어레이는 테이블 형태의 데이터 구조로 가장 일반적을 사용되는 데이터 포맷이다. ScitPy 라이브러리를 과학적인 계산을 지원하는 라이브러리이고, matplotlib는 그래프를 편리하게 그리는데 사용하는 라이브러리이다. panda는 데이터분석과 통계처리 라이브러리이다. scikit-learn은 기계학습 라이브러리이다.

파이선 개발을 위한 유용한 인터페이스로 Jupyter/IPython이 있는데, 이는 프로그램 코딩, 라인별 동작확인, 문서관리 등을 편리하게 제공하며 전세계 파이선 개발자가 애용하는 개발 환경도구이다. 이 책에서도 Jupyter를 사용하여 설명하겠다.

파이선 자체와 위에 소개한 라이브러리들을 모두 통칭하여 SciPy 스택 또는 PyData 플랫폼이라고도 부른다.

데이터 사이언스에 사용되는 다른 도구

과학이나 공학 분야에서 수치적 분석에 널리 사용되는 MATLAB이 있고, 통계분석으로 유명한 SPSS가 있다. 이들은 모두 유료 소프트웨어이다. 통계처리 언어로 파이선과 함께 널리 사용되는 R이 있는데 이는 무료 소프트웨어이다.

IPython

IPython(“interactice" Python)은 Fernando Perez가 2001년에 개발한 도구로, 개발자가 프로그램을 문장 단위로 실행해볼 수 있는 도구이다. 2011년에 IPyton에 노트북(Notebook)이라는 기능이 추가되었는데, 노트북은 프로그램 소스 코드 뿐 아니라, 프로그램에 대한 설명 도큐먼트, 결과 그래프, 수식 등을 모두 하나의 파일에 저장하고 관리하는 편리한 기능을 제공한다. 예전에도 프로그램에 주석문을 컴파달 수 있지만 텍스트형태로 제한된 도큐먼트만 가능했었다. 노트북을 사용하면 다양

한 서식과, 그리프, 그리고 스크롤바 같은 사용자 인터페이스도 한번에 다룰 수 있다. 쥬피터 노트북 사용자가 점점 많아지고 지원하는 언어도 파이선 뿐 아니라 R, Ruby, Julia 등으로 늘어나면서 노트북을 Jupyter라는 이름으로 사용하게 되었다 (2014년). IPython은 Jupyter 노트북의 핵심 기능, 즉, 컴파일, 변수관리 등을 담당하며 이를 kernel이라고 부른다. 쥬피터를 온라인을 실습한 수 있는 사이트가 있다. 쥬피터 노트북의 장점은 프로그램 개발과 실행 모든 동작을 웹기반으로 수행할 수 있다는 것이다.

웹기반으로 연습할 수도 있는데 아래 사이트를 참조:

try.jupyter.org

wakari.io

쥬피터 실행

터미널 창을 열고 작업할 디렉토리로 이동해서 아래 명령을 입력하면 쥬피터를 실행할 수 있다. 아래에서 현재 디렉토리가 홈디렉토리(~) 아래에 있는 code/data_sci 인 것을 나타내며 '$" 기호는 프롬프트이다. 즉, 입력할 명령문은 jupyter notebook인 것을 나타낸다.

~/code/data_sci $ jupyter notebook

쥬피터는 웹기반으로 동작하며 실습에서 사용하는 컴퓨터 자체가 로컬 서버역할을 하고 웹 브라우저처럼 유피터 대시보드가 나타난다. 아래에 쥬피터 대시보드를 나타냈다.

현재 디렉토리에 아무 파일도 없다는 것을 나타낸다. 대시보드에 3개의 탭이 있는데 각각의 기능은 다음과 같다. Files는 현재 디렉토리에 있는 모든 파일을 보여준다. Running 탭은 현재 실행중인 프로그램이 무엇인지를 알려주고 Clusters 탭은 여러개의 프로그램을 병렬로 실행시킬 때 사용한다.

쥬피터에서는 프로그램 파일을 모두 노트북(notebook)이라고 부르며 확장자는 .ipynb이다. 노트북에는 프로그램 코드 뿐 아니라, 설명문서, 그래프 등도 포함할 수 있다. 쥬피터는 일종의 파일관리자, 프로그램 실행관리자라고 볼 수 있으며, 실제로 파이선 코드를 실행하는 것은 커널(kernel) 프로세스가 담당한다. 쥬퍼터는 커널의 도움을 받아 프로그램을 실행하고 그 결과를 사용자에게 대화형으로 제공하는 인터페이스라고 할 수 있다.

새로운 노트북을 만들려면 맨 우측 New 버튼을 클릭하고 Python3 노트북을 선택하면 된다. 아래와 같은 새로운 노트북 창이 나타난다.

새로 만들어진 노트북의 이름은 "Untitled"부분을 클릭하여 바꿀 수 있다. 위에서 In [ ] 부분을 셀이라고 한는데 현재 이 셀의 타입이 Code로 되어 있다(툴발 중간에 Code라는 단어가 보임). 즉, 현재 In [ ] 부분은 코드셀(code cell)이라고 부르며 파이선 코드를 입력할 수 있는 상태를 말한다.

셀 타입이 코드셀이며 파이선 코드를 입력할 수 있고, 셀 타입을 문서 작성용으로 바꿀 수 있다. 아래와 같이 Code 우측의 드롭다운 메뉴를 누르면 Markdown을 선택할 수 있는 메뉴가 나타난다.

여기서 Markdown을 선택하면 아래와 같이 In [ ] 없는 입력창이 나타난다. 이를 마크다운 셀이라고 하는데 이 셀은 문서를 편집하여 도큐먼트를 만드는데 사용된다. 즉 쥬피터 노트북은 코드 영역과 문서 영역을 쉽게 구분하여 작성할 수 있게 해준다.

위의 마크다운 셀을 클릭하면 문서 내용을 입력할 수 있는데, 예를 들어 아래와같은 내용을 입력하였다고 하자.

위의 내용처럼, '#'은 제목의 수준을 타나내는데 '#'의 갯수에 따라 제목이 수준이 정해지며 '#'의 갯수가 적을수록 상위 레벨의 제목이 된다. 문서 내용을 입력한 후에 Shift + 엔터키를 입력하면 마크다운이 적용된 포맷으로 보인다(아래 그림 참조). 마크다운 셀을 다시 편집하려면 해당 마크다운 셀 내부를 두 번 클릭하면 된다.

즉, 마크다운 셀을 편집하려면 셀을 두번 클릭하면 되고, 마크다운이 적용된 (rendered) 표현을 보려면 Shift + 엔터 키를 입력하면 된다. 위의 마크다운 셀 실행 결과를 보면 맨 아래에 In [ ] 이 있는 셀 즉, 코드셀이 자동으로 나타나며 이곳에 파이선 코드를 입력할 수 있다.

아래에 코드셀에 간단한 파이선 코드를 입력하고 이를 실행 (Shift + 엔터)한 결과를 보였다. 코드셀을 편집하려면 해당 셀 내부를 한번만 클릭하면 되고 다시 해당셀을 실행하려면 Shift + 엔터로 실행할 수 있다. 셀을 실행시키지 않고 편집 모드에서 빠져나오기만 하려면 Esc 키를 누르거나 화면의 셀 외부 다른 곳을 클릭하면 된다 (이 모드를 명령 모드라고 부른다). 명령 모드에서 a나 b를 입력하면 빈 셀을 현재 셀의 위나 아래에 하나 추가한다. 편집 모드의 셀은 초록색이고 명령 모드는 파란색으로 보인다.

마크다운 셀이나 코드셀을 실행할 때 Shift + 엔터 가 아니라 Ctrl + 엔터 키를 사용하면 셀을 실행한 후 커서의 위치가 다음 셀로 이동하지 않고 현재 셀에 남아있게 된다. 셀 편집 모드에서 셀을 두개로 분할하려면 분할하고 싶은 위치에서 Shift + Ctrl + '-' 키를 입력하면 된다.

파이선 맛보기

코드셀(code cell)에 파이선 코드를 입력하고 Shift + 엔터키를 입력하면 코드가 실행된다

파이선 문법은 다음 사이트에 자세히 소개되어 있다 https://docs.python.org/3.4/contents.html

기초 문법

lists, strings, tuples, dictionaries, for-loop, while-loop, if-else

숫자, 문장 출력

숫자, 문자열 등을 출력하려면 pritn() 함수를 아래와 같이 사용하면 된다.

In [4]:
```
print("처음 실행하는 파이선...")
```

```
처음 실행하는 파이선...
```

실습예제1

본인의 이름을 출력해 보아라.

In [ ]:
```

```

노트북에서 코드셀에는 일렬 번호가 자동으로 배정된다 (위에서는 [2] 임). 이 번호 숫자는 크게 신경쓰지 않아도 된다. 번호는 나중에 모두 없애거나 새로운 번호로 새로 배정할 수도 있다.

셀을 삭제하거나 복사하려면 명령 모드에서 x, c, v 를 사용하면 된다. x는 셀 삭제, c는 복사, v는 붙여넣기를 한다. 두개의 셀을 합치려면 Shict + m을 입력하고, 하나의 셀을 둘로 나누려면 Shift + Ctrl + '-'를 입력한다. 새로운 셀 삽입은 a(위에 삽입), b(아래 삽입)을 한다 작업 중인 셀의 위치를 이동하려면 명령 모드에서 j, k 키를 사용하면 선택하는 셀의 위치를 위 아래로 이동한다 (또는 상하 표시 화살표를 이용해도 된다)

코드셀을 실행한 결과 화면을 잠시 가리려면 명령 모드에서 'o' 를 입력하면 된다

변수 사용

변수는 아래와 같이 숫자, 문자열 등을 임의로 배정할 수 있다.

In [5]:
```
age1 = 29
age2 = 33
name = '홍길동'
name2 = "홍길동2"

print(name, '나이는', age1)
print(name2, '나이는', age2)
```

```
홍길동 나이는 29
홍길동2 나이는 33
```

In [6]:
```
address = """홍길동의
집주소는
여러줄로 되어있습니다"""
print(address)
```

```
홍길동의
집주소는
여러줄로 되어있습니다
```

In [7]:
```
print(name[0])
# 문자열에서 특정 위치의 글자를 출력할 수 있다
```

```
홍
```

실습예제2

myName 변수를 만들어 본인의 이름을 배정하고, myAge 변수를 만들어 본인의 나이를 배정한 뒤 출력해 보아라.

In [ ]:
```

```

실습예제3

myName변수에서 본인의 성만 출력해 보아라.

In [ ]:
```

```

특수문자 출력

", ', \, 등의 특수 문자를 화면에 출력하려면 앞에 '\' 문자를 입력하면 된다. 새로운 라인을 출력하려면 \n 을 입력하면 된다.

print 문 안에서 문자열 전체를 그대로 출력하려면 'r'을 문자열 앞에 붙이면 된다(raw의 의미임).

In [8]:
```
print("Hello \"world\"")
print("A list:\n* item 1\n* item 2")
print("C:\\path\\on\\windows")
print(r"C:\path\on\windows")
```

```
Hello "world"
A list:
* item 1
```

* item 2
C:\path\on\windows
C:\path\on\windows

실습예제4

출력의 결과가 다음과 같이 나오도록 실습해 보아라.(코드는 한 줄로 작성, 새로운 라인포함 된것임.)

파이썬 "짱" 좋아요 너무좋아요!

In [ ]:

리스트

여러 항목의 집합을 말하며 항목의 타입은 서로 달라도 되나, 대부분 같은 타입을 사용한다. 리스트를 만들려면 아래와 같이 []를 사용하고 각 항목을 ','로 구분하면 된다. 리스트의 내용은 변경될 수 있다(mutable)

In [9]:
```
items = [1, 2, 3, 4, 'kim']
items
```

Out[9]: [1, 2, 3, 4, 'kim']

In [10]:
```
# 리스트의 길이는 내장함수인 `len()`을 사용하여 알 수 있다.
len(items)
```

Out[10]: 5

In [11]:
```
items[4]=5
# 5번째 항목을 문자열에서 숫자로 바꾸었다
items
```

Out[11]: [1, 2, 3, 4, 5]

내장 함수란 미리 만들어 둔 함수인데 아래 사이트에서 함수들을 확인할 수 있다.
https://docs.python.org/3.4/library/functions.html.

아래는 리스트의 합계를 구하는 내장함수 sum()의 사용예이다

리스트 내의 측정 항목은 아래와 같이 인덱스로 찾을 수 있다. 인덴스는 0부터 시작한다. 뒤에서부터 항목을 찾을 수도 있는데 이때는 -3과 같이 마이너스 기호를 붙이면 된다.

In [12]:
```
items[0]
```

Out[12]: 1

In [13]:
```
items[-1]
```

Out[13]: 5

In [14]:
```
# 리스트의 일부 구간만 얻으려면(slice) ':'를 사용한다
items[1:4]
```

Out[14]: [2, 3, 4]

실습예제5

1,2,3,4,5 순서대로 myList 변수에 리스트를 만들고 마지막에 6을 추가하는 내장함수를 사용해 보고 뒤에서 세번째까지만 출력해 보아라.

HINT. 리스트 a의 마지막에 x를 추가하는 함수는 다음과 같다. a.append(x)

In [ ]:

```
```

튜플

리스트와 달리 튜플**(Tuples)** 을 사용하면 고정된 값들로 구성된 리스트를 만들 수 있다. 즉, 값을 나중에 바꿀 수 없다. 이렇게 나중에 값을 바꿀 수 없는 것을 immutable하다고 부른다.

In [15]:

```
# 튜플은 ( )를 사용하여 만들거나 , 로 구분만 해도 된다
my_tuple = (1, 2, 3)
your_tuple = 4,5,6,7
print(my_tuple)
print(your_tuple)
```

```
(1, 2, 3)
(4, 5, 6, 7)
```

In [16]:

```
# 특정 항목을 찾을 때는 리스트와 같이 [ ]를 사용한다
print(my_tuple[0])
```

```
1
```

In [17]:

```
my_tuple[2] = 7
# 튜플 값을 변경하는 것은 불가능하다
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-b38a5dfc18b0> in <module>()
----> 1 my_tuple[2] = 7
      2 # 튜플 값을 변경하는 것은 안된다

TypeError: 'tuple' object does not support item assignment
```

실습예제6

괄호를 이용하여 myTuple변수에 1,3,5,7,9를 선언하고, 튜플의 값을 임의로 변경해 보아라.

In [ ]:

```
```

딕셔너리**(Dictionaries)**

딕셔너리를 사용하면 키-값의 조합을 항목으로 갖는 리스트를 만들 수 있다. 딕셔너리에서는 항목을 찾을 때 키를 사용하며, 딕셔너너리에서 순서는 중요시하지 않는다. 딕셔너리는 { }로 만든다.

In [18]:

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
print('a:', my_dict['a'])
print(my_dict.keys())
```

```
a: 1
dict_keys(['a', 'b', 'c'])
```

실습예제7

딕셔너리에도 내장함수가 존재한다. 딕셔너리 a의 Key들을 모아놓은 리스트를 돌려주는 함수는 a.keys()이다. 딕셔너리 myDic변수에 'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4, 'e': 5를 선언하고 딕셔너리 myDic의 key들을 모아놓은 리스트를 출력해 보아라.

In [ ]:
```
```

for 루프

여러 항목에 대해 동작을 반복할 때 for 루프를 실행할 수 있는데, 주의할 것은 for 문장의 끝에는 ':'가 있어야 하며 for 문이 적용되는 블록(block) 영역은 반드시 들여쓰기 (indentation)을 해야 한다. 들여쓰기는 한 프로그램 내에서 모두 같은 크기로 지정해야 하는데 일반적으로 스페이스 4개 또는 탭을 사용한다.

In [19]:
```
for item in items:
    print(item)
```

```
1
2
3
4
5
```

In [20]:
```
for i in (1,2,3,4,5):
    print (i*2)
```

```
2
4
6
8
10
```

In [21]:
```
for i in range(1,6):
    print (i*3)

# range 구간에서 마지막 값은 포함되지 않는다
```

```
3
6
9
12
15
```

In [22]:
```
# 아래와 같이 for 문장을 간결하게 쓰는 방법도 있다. 이를 list comprehension이라고 부른다.

squares = [i * i for i in items]
squares
```

Out[22]: [1, 4, 9, 16, 25]

In [23]:
```
# 시리즈를 사용하여 컬럼 값을 입력할 수 있다 (없는 인덱스는 NaN이 된다)
val = Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
```

```
frame2['debt'] = val
frame2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-8a101e12b11a> in <module>()
      1 # 시리즈를 사용하여 컬럼 값을 입력할 수 있다 (없는 인덱스는 NaN이 된다)
----> 2 val = Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
      3 frame2['debt'] = val
      4 frame2

NameError: name 'Series' is not defined
```

실습예제8

for문과 range를 사용하여 구구단 2단을 출력해 보아라.

In [ ]:

실습예제9

se_dic변수에 1:1, 2:4, 3:9로 딕셔너리를 선언해주고, key값과 value값을 출력해 보아라.

In [ ]:

if 문

In [24]:
```
for item in items:
    if item % 2 == 0:
        print(item)

# 위에서 조건문을 확인할 때에는 '=='를 사용하며, '=' 는 변수에 값을 지정할 때 사용한다. if문도 뒤에 `:`가 와야 하며,
# 이 조건이 만족될 때 실행되는 코드 블록은 들여쓰기를 해야 한다
```

```
2
4
```

실습예제10

myGrade변수에 100을 선언하고
if문을 사용하여 myGrade값이 100인 경우이면 "만점입니다!"를 출력해 보시오.

In [ ]:

실습예제11

myMoney변수에 3000을 선언하고 if문을 사용하여 3000원이상이면 "택시를 타라"를 출력해보아라.
(비교연산자 사용)

In [ ]:

라이브러리

In [25]:
```
# 라이브러리를 불러 쓰려면 import를 호출해야 한다
# 아래는 수학 라이브러리인 math를 부르는 명령이며 이 라이브러리 이름을= 편리하게 'm'으로 부른다는 뜻이다

import math as m
m.factorial(4)
```

Out[25]: 24

파이선이 제공하는 대표적인 라이브러리는 다음과 같다.

NumPy (Numerical Python) 다차원 어레이를 쓸 수 있게 한다. 선형대수 처리, 푸리에 변형(스펙트럼 분석) 등을 제공한다

SciPy (Scientific Python). SciPy는 NumPy 가 설치되어야 실행된다. 공학적, 과학적 계산 기능을 제공한다.

Matplotlib 히스토그램, 히트맵 등 다양한 그림을 그리는데 사용된다.

Pandas 데이터를 구조화해서 사용할 때 편리하다. 데이터의 구조를 파악하고 결측치 처리등을 할 때 편리하다.

Scikit Learn 기계하가습용 라이브러리로 NumPy, SciPy, matplotlib을 사용한다. 기계학습의 대표적인 기능은 분류, 리그레션, 클러스터링, 차원축소 등의 기능을 제공한다.

Statsmodels 통계모델링에 사용한다. 데이터를 탐색하고, 통계적모델을 만들고 테스트하는데 사용된다.

Seaborn matplotlib 위에서 실행되면 좀 더 세련된 그래픽, 시각화 기능을 제공한다.

함수

자주 사용되는 기능을 함수로 정의해 두고 나중에 불러쓰는 것이 프로그래밍의 기초이다. 아래는 입력받은 숫자(아래에서 number)가 짝수인지를 찾아내는 함수 정의이다. 함수는 def 문으로 정의하며 정의문은':'으로 끝나야 한다. 아래에서 """로 시작하고 끝나는 부분은 코멘트 영역이다.

함수가 실행되고 나면 'return' 부분이 출력으로 얻게 되는 부분이다. 위의 함수에서는 리턴문이 '=='를 포함하고 있으므로 이는 Boolean 값을 리턴한다. 즉 True나 False 중에 한 값을 리턴한다.

위에서 정의된 함수를 상용하는 예는 아래와 같다.

In [26]:
```
def is_even(number):
  """Return whether an integer is even or not."""
  return number % 2 == 0

is_even(3)

is_even(4)
```

Out[26]: True

실습예제**11**

더하기를 수행하는 함수를 만들어 보아라. 함수의 이름은 add이다. 인자는 a와 b를 주어라.

In [ ]:

```
```

함수 호출 방법

함수를 호출 할 때 인자 값으로 디폴트로 정해진 값을 사용하는 것이 가능하다. 아래의 예에서 두번째 인자 값을 지정하지 않으면 디폴트로 2가 사용된다. 또는 인자의 이름을 명시해서 호출하는 것도 가능하다.

In [27]:

```
def remainder(number, divisor=2):
    return number % divisor

remainder(5)

remainder(5, 3)

remainder(5, divisor=3)
```

Out[27]: 2

함수는 여러개의 변수를 인자로 받을 수 있는데, 위치로 값을 받는 인자를 위치(positional) 인자라고 하고 키값을 지정하는 인자를 키타입(keyword) 인자라고 한다.

In [28]:

```
def f(*args, **kwargs):
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)

f(1, 2, c=3, d=4)
# 위에서 `args`는 위치 인자들을 포함하는 튜플을 가리키고, `kwargs`는 키타입 인자를 포함하는 딕셔너리를 가리킨다.

'hello'.upper()
# 여기서 `upper()`함수는 `str` 객체가 지원하는 메소드이고 'hello'가 str 타입의 객체이므로 upper() 함수를 호출할 수 있는 것이다.
# 파이선 객체 뒤에 '.'을 입력하고 탭을 누르면 어떤 함수들이 지원되는지를 보여준다
```

```
Positional arguments: (1, 2)
Keyword arguments: {'c': 3, 'd': 4}
```

Out[28]: 'HELLO'

실습예제12

본인의 이름을 출력하는 함수를 만들어 보아라. 함수의 이름은 say_myName으로 하고, 인자는 2개로, 두번째오는 인자 값으로 디폴트로 정해준다. def say_myName(name, y = 'genius') 형식으로! 본인의 이름과 y를 출력하는 함수를 완성해 보아라

In [ ]:

```
```

In [1]:
```
# 숫자 다루기
# 튜플, 리스트, 배열
```

In [2]:
```
import numpy as np

# 튜플 만들기
x = (1,2,3,4,5)
print(x)
type(x)
```

```
(1, 2, 3, 4, 5)
```

Out[2]: tuple

In [3]:
```
x[4]
```

Out[3]: 5

In [4]:
```
x[2] =100 # 튜플은 값의 변경이 불가능하다.
x
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call la
st)
<ipython-input-4-80553d451659> in <module>()
----> 1 x[2] =100 # 튜플은 값의 변경이 불가능하다.
      2 x

TypeError: 'tuple' object does not support item assignment
```

In [5]:
```
# 리스트 만들기
y = [1,2,3,4,5]
print(y)
type(y)
```

```
[1, 2, 3, 4, 5]
```

Out[5]: list

In [6]:
```
y[2]
```

Out[6]: 3

In [7]:
```
y[2] = 100 # 리스트는 값으 변경이 가능하다.
y[2]
```

Out[7]: 100

In [8]:
```
# 리스트에는 임의의 타입 데이터를 담을 수 있다
z = [1,'A', 3.14]
type(z)
```

Out[8]: list

In [9]:
```
# 리스트로부터 어레이(배열)를 만드는 예
# np의 array 함수를 사용한다
```

```
data = [1,2,3,4,5,6] # 1차원 배열임
arr = np.array(data) # python list를 numpy ndarray로 바꾸는 함수를 사용
print(arr)
type(arr)
```

```
[1 2 3 4 5 6]
```

Out[9]:
```
numpy.ndarray
```

In [10]:
```
# 2차원 리스트로부터 2차원 배열을 만드는 예
data = [[1,2,3],[4,5,6],[7,8,9]]
print(data)
type(data)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Out[10]:
```
list
```

In [11]:
```
arr = np.array(data) # list를 ndarray로
print(arr)
type(arr)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Out[11]:
```
numpy.ndarray
```

In [12]:
```
# 배열의 각 항목의 타입을 알아보려면 dtype 속성값을 보면 된다
arr.dtype
```

Out[12]:
```
dtype('int32')
```

In [13]:
```
# 배열의 구조를 보려면 shape 속성을 보면 된다
arr.shape
```

Out[13]:
```
(3, 3)
```

In [14]:
```
# 초기값이 0인 배열을 만드는 법
np.zeros((2,3))
```

Out[14]:
```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

In [15]:
```
# 레인지(범위)를 만드는 법
x = range(5)
print(x)
x[2] # 레인지는 배열처럼 값을 불러올 수 있다.
```

```
range(0, 5)
```

Out[15]:
```
2
```

In [16]:
```
# 레인지 형태로 배열을 만드는 법
y = np.arange(5)
y
```

Out[16]:
```
array([0, 1, 2, 3, 4])
```

In [17]:
```
y[2]
```

Out[17]:
```
2
```

In [18]:
```
# 데이터의 타입을 변경하려면 astype을 사용한다
# 정수로 구성된 배열을 부동소수 타입을 변환하는 예
z = y.astype(np.float64)
z
```

Out[18]:
```
array([ 0.,  1.,  2.,  3.,  4.])
```

In [19]:
```
# 벡터화: 배열에 상수를 곱하거나 더할 수 있다
arr*2
```

Out[19]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

In [20]:
```
arr + 1000
```

Out[20]:
```
array([[1001, 1002, 1003],
       [1004, 1005, 1006],
       [1007, 1008, 1009]])
```

In [21]:
```
# 배열을 곱하면 각 항목별로 곱셈을 한다
arr*arr
```

Out[21]:
```
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])
```

실습예제 **1**

arr 배열에 3을 곱한 뒤 음수로 바꾸어보시오.

In [ ]:
```
```

In [22]:
```
# 리스트
x = [1,2,3,4,5]
x
```

Out[22]:
```
[1, 2, 3, 4, 5]
```

In [23]:
```
# 배열
y = np.array(x)
y
```

Out[23]:
```
array([1, 2, 3, 4, 5])
```

In [24]:
```
x[2] = 100
y[2] = 100
x
```

Out[24]:
```
[1, 2, 100, 4, 5]
```

In [25]:
```
y
```

Out[25]:
```
array([  1,   2, 100,   4,   5])
```

In [26]:
```
x_slice = x[3:5]
x_slice
```

Out[26]:
```
[4, 5]
```

In [27]:
```
y_slice = y[3:5]
y_slice
```

Out[27]:
```
array([4, 5])
```

In [28]:
```
x_slice[1] = 200
y_slice[1] = 200
```

In [29]:
```
x_slice
```

Out[29]:
```
[4, 200]
```

In [30]:
```
y_slice
```

Out[30]:
```
array([  4, 200])
```

In [31]:
```
# 리스트의 일부인 슬라이스 리스트에서 값을 바꾸어도 원본 리스트의 값은 달라지지 않는다
x
```

Out[31]:
```
[1, 2, 100, 4, 5]
```

In [32]:
```
# 그러나 배열에서는 슬라이스 배열의 값을 바꾸면 원본 배열의 값도 달라진다
# list는 새로운 주소값을 생성하여 참조하지만, 배열은 동일 주소값을 참조하기 때문이다.
# 리스트와 배열의 다른 점 중 하나
y
```

Out[32]:
```
array([  1,   2, 100,   4, 200])
```

실습예제 **2**

1부터 10까지의 정수를 원소로 가진 배열을 만들고, 4번째부터 6번째 까지의 값을 slice하여 출력해 보시오.

In [ ]:
```

```

In [33]:
```
arr
```

Out[33]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [34]:
```
arr[1,2]
```

Out[34]:
```
6
```

In [35]:
```
arr[1][2]
```

Out[35]:
```
6
```

In [36]:
```
# 2차원 배열 중에 한 차원의 배열만 얻으려면 인자를 한 차원 낮게 주면 된다
# 아래는 2차원 배열 중 첫번째 행만 얻는다
arr[0]
```

Out[36]:
```
array([1, 2, 3])
```

In [37]:
```
# 행은 1~2의 범위를 , 열은 1~3의 범위를 얻는 예이다
```

```
arr[:2,:3]
```

Out[37]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [38]:
```
# 전체는 ':'로 표시할 수 있다
arr[:2,:]
```

Out[38]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [39]:
```
# 배열의 불리언 색인
name = np.array(['kim', 'lee', 'park', 'kim'])

# 0~11 값을 얻고 이를 4x3 배열로 재구성한다
data = np.arange(12).reshape(4,3)
data
```

Out[39]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [40]:
```
name == 'kim'
# 첫번째, 네번째 항목만 True를 얻는다
```

Out[40]:
```
array([ True, False, False,  True], dtype=bool)
```

In [41]:
```
# 배열 data에서 첫번째, 네번째 행만 얻는 방법
data[name == 'kim']
```

Out[41]:
```
array([[ 0,  1,  2],
       [ 9, 10, 11]])
```

In [42]:
```
# 배열 data에서 'kim' 또는 'lee'의 위치의 행만 얻는 방법
data[(name == 'kim') | (name == 'lee')]
```

Out[42]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 9, 10, 11]])
```

In [43]:
```
# 5 보다 큰 값을 모두 5로 교체하는 방법
data[data > 5] = 5
data
```

Out[43]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [5, 5, 5],
       [5, 5, 5]])
```

실습예제 **3**

0부터 15까지의 정수를 가진 4*4 배열을 만들고, 7 이하의 정수를 모두 음수로 바꾸어보시오.

In [ ]:
```
```

In [44]:
```
# 이름이 'kim'에 해당하는 내용을 모두 100으로 바꾸는 예
data[name =='kim'] = 100
data
```

Out[44]:
```
array([[100, 100, 100],
       [  3,   4,   5],
       [  5,   5,   5],
       [100, 100, 100]])
```

In [45]:
```
data = np.arange(12).reshape(4,3)
data
```

Out[45]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [46]:
```
# 팬시 색인 - 어레이에서 특정 행을 바로 액세스할 수 있다
# 아래는 두번째와 세번째 행을 얻는 경우이다
data[[1,2]]
```

Out[46]:
```
array([[3, 4, 5],
       [6, 7, 8]])
```

In [47]:
```
data[[0]]
```

Out[47]:
```
array([[0, 1, 2]])
```

In [48]:
```
# 행의 순서를 임의의 순으로 바꿀 수 있다
# 팬시 색인으로 배열을 바꾸면 본사본이 생기며 원래의 내용을 바지 않는다
data[[3,2,0,1]]
```

Out[48]:
```
array([[ 9, 10, 11],
       [ 6,  7,  8],
       [ 0,  1,  2],
       [ 3,  4,  5]])
```

In [49]:
```
data
# data 내용은 그대로 남아 있다
```

Out[49]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [50]:
```
# 행과 열을 모두 바꾸려면 아래와 같이 한다
data[[3,2,1,0]][:,[2,1,0]]
```

Out[50]:
```
array([[11, 10,  9],
       [ 8,  7,  6],
       [ 5,  4,  3],
       [ 2,  1,  0]])
```

In [51]:
```
# 트랜스포즌 행과 열의 위치를 바꾸는 것이다
data.T
```

Out[51]:
```
array([[ 0,  3,  6,  9],
       [ 1,  4,  7, 10],
       [ 2,  5,  8, 11]])
```

In [52]:
```
# 배열의 곱셈은 np.dot()을 사용한다
np.dot(data,data.T)
```

Out[52]:
```
array([[  5,  14,  23,  32],
```

```
       [ 14,  50,  86, 122],
       [ 23,  86, 149, 212],
       [ 32, 122, 212, 302]])
```

In [53]:
```
# 조건에 따라 값을 다르게 선택하는 예
# np.where 를 사용한다
# 아래는 5보다 크면 6으로 바꾸고 5보다 작으면 모두 3으로 바꾼다
np.where(data>5, 6, 3)
```

Out[53]:
```
array([[3, 3, 3],
       [3, 3, 3],
       [6, 6, 6],
       [6, 6, 6]])
```

In [54]:
```
# 5보다 크면 그대로 두고 작으면 5로 바꾸는 예
np.where(data>5, data, 5)
```

Out[54]:
```
array([[ 5,  5,  5],
       [ 5,  5,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [55]:
```
# 전체의 평균을 구한다
data.mean()
```

Out[55]:
```
5.5
```

In [56]:
```
np.mean(data)
```

Out[56]:
```
5.5
```

In [57]:
```
data.sum()
```

Out[57]:
```
66
```

In [58]:
```
# 각 행에 대해서 평균을 구한다 axis=1이면 행을 지칭한다
data.mean(axis=1)
```

Out[58]:
```
array([  1.,   4.,   7.,  10.])
```

In [59]:
```
# axis를 생략해도 된다
data.mean(1)
```

Out[59]:
```
array([  1.,   4.,   7.,  10.])
```

In [60]:
```
# 각 열에 대해서 평균을 구한다 axis=0이면 열을 지칭한다
data.mean(0)
```

Out[60]:
```
array([ 4.5,  5.5,  6.5])
```

In [61]:
```
# 불리언 연산, True는 1로 False는 0으로 숫자로 치환하여 계산가능하다
1 + True
```

Out[61]:
```
2
```

In [62]:
```
# 배열에서 값이 특정 수보다 큰 원소의 갯수를 반환한다.
x = np.array([1,2,3,4,5,6])
(x>4).sum()
```

Out[62]:
```
2
```

In [63]:
```
(data>1).sum()
```

Out[63]:
```
10
```

In [64]:
```
(x>4)
```

Out[64]:
```
array([False, False, False, False,  True,  True], dtype=bool)
```

In [65]:
```
# True가 하나 이상 있는가?
(x==True).any()
```

Out[65]:
```
True
```

In [66]:
```
# 모두 True인가?
(x==True).all()
```

Out[66]:
```
False
```

실습예제 **4**

-5 부터 4 까지의 정수를 원소로 가진 배열을 만들고, True인 원소의 갯수를 출력해보시오.

In [ ]:
```

```

In [67]:
```
# 소팅
# 리스트
y = [2,4,1,5,8,6,7,3,9]
y
```

Out[67]:
```
[2, 4, 1, 5, 8, 6, 7, 3, 9]
```

In [68]:
```
sorted(y)
```

Out[68]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [69]:
```
# y의 내용은 바뀌지 않았다
y
```

Out[69]:
```
[2, 4, 1, 5, 8, 6, 7, 3, 9]
```

In [70]:
```
y.sort()
y
# y의 내용이 바뀌었다
```

Out[70]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [71]:
```
# 배열
z = np.array([2,4,1,5,8,6,7,3,9])
type(z)
```

Out[71]:
```
numpy.ndarray
```

In [72]:
```
z
```

Out[72]:
```
array([2, 4, 1, 5, 8, 6, 7, 3, 9])
```

In [73]:
```
sorted(z)
```

Out[73]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [74]:
```
z
```

Out[74]:
```
array([2, 4, 1, 5, 8, 6, 7, 3, 9])
```

In [75]:
```
z.sort()
```

In [76]:
```
t = np.array([[8,4,11,5],[2,10,7,3],[9,12,1,6]])
t
```

Out[76]:
```
array([[ 8,  4, 11,  5],
       [ 2, 10,  7,  3],
       [ 9, 12,  1,  6]])
```

In [77]:
```
# 열을 기준으로 소팅
t.sort(0)
```

In [78]:
```
t
```

Out[78]:
```
array([[ 2,  4,  1,  3],
       [ 8, 10,  7,  5],
       [ 9, 12, 11,  6]])
```

In [79]:
```
# 행에서 소팅
t.sort(1)
```

In [80]:
```
t
```

Out[80]:
```
array([[ 1,  2,  3,  4],
       [ 5,  7,  8, 10],
       [ 6,  9, 11, 12]])
```

In [81]:
```
# 집합 set
# 리스트나 배열에서 중복된 항목을 제외하고 유일하게 있는 항목만 얻는다
# 리스트
name = ['kim', 'lee', 'park', 'kim', 'lee']
```

In [82]:
```
set(name)
```

Out[82]:
```
{'kim', 'lee', 'park'}
```

In [83]:
```
name
# name은 바뀌지 않았다
```

Out[83]:
```
['kim', 'lee', 'park', 'kim', 'lee']
```

In [84]:
```
type(name)
```

Out[84]:
```
list
```

In [85]:
```
name_2 = np.unique(name)
# 배열을 리턴한다
```

In [86]:
```
name
```

Out[86]:
```
['kim', 'lee', 'park', 'kim', 'lee']
```

In [87]:
```
name_2
```

Out[87]:
```
array(['kim', 'lee', 'park'],
      dtype='<U4')
```

In [88]:
```
long_name = ['kim', 'kang', 'lee', 'park', 'kim', 'lee', 'min']
```

In [89]:
```
np.in1d(long_name, ['kim'])
# 'kim'이 들어 있는 위치를 찾는다
```

Out[89]:
```
array([ True, False, False, False,  True, False, False], dtype=bool)
```

In [90]:
```
np.in1d(long_name, name)
# 'kim', 'lee', 'park' 중 하나가 들어 있는 위치를 찾는다
```

Out[90]:
```
array([ True, False,  True,  True,  True,  True, False], dtype=bool)
```

In [91]:
```
b = np.in1d(long_name, name)
b
```

Out[91]:
```
array([ True, False,  True,  True,  True,  True, False], dtype=bool)
```

In [92]:
```
b.sum()
```

Out[92]:
```
5
```

In [93]:
```
np.in1d(long_name, name).sum()
```

Out[93]:
```
5
```

In [94]:
```
p=np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [95]:
```
# 순서를 임의로 바꾸는 방법
q = np.random.permutation(p)
q
```

Out[95]:
```
array([4, 2, 9, 5, 7, 3, 1, 8, 6])
```

In [96]:
```
q.sort()
```

In [97]:
```
q
```

Out[97]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [1]:
```
# 데이터구조 pandas
# 데이터 분석에서 가장 많이 사용하는 데이터 구조들을 소개한다
# 데이터프레임을 배운다
```

In [2]:
```
# 시리즈와 데이터프레임 두가지가 기본이다
# 시리즈는 1차원 배열과 같은 형태를 갖는데 다른점은 각 항목마다 인덱스를 지정할 수 있다
# 자동으로 인덱스가 만들어진다

import numpy as np
from pandas import Series, DataFrame
import pandas as pd

s = Series([34,54,78])
s
```

Out[2]:
```
0    34
1    54
2    78
dtype: int64
```

In [3]:
```
#시리즈 데이터셋을 구성한다.
s = Series(['kim', 'lee', 'park'])
s
```

Out[3]:
```
0     kim
1     lee
2    park
dtype: object
```

In [4]:
```
s.values
```

Out[4]:
```
array(['kim', 'lee', 'park'], dtype=object)
```

In [5]:
```
s.index
```

Out[5]:
```
RangeIndex(start=0, stop=3, step=1)
```

In [6]:
```
# 인덱스를 임의로 지정할 수 있다
s = Series(['kim', 'lee', 'park'], index=['a', 'b', 'r'])
s
```

Out[6]:
```
a     kim
b     lee
r    park
dtype: object
```

실습예제 **1**

'BMW', 'Nissan', 'Honda', 'Audi'를 값으로 갖고 'one', 'two', 'three', 'four'를 인덱스로 갖는 시리즈 데이터 셋을 선언하시오

In [ ]:
```
```

In [7]:
```
# 인덱스로 해당 값을 얻을 수 있다
s['r']
```

Out[7]: 'park'

In [8]:
```
#특정값이 데이터에 속해있는지 확인할수있다
'r' in s
```

Out[8]: True

In [9]:
```
'park' in s
```

Out[9]: False

In [10]:
```
# 파이썬 기본 타입인 사전으로부터 시리즈를 만들 수 있다
dic = {'서울':800, '부산':150, '대구': 100}
dic
```

Out[10]: {'대구': 100, '부산': 150, '서울': 800}

In [11]:
```
s = Series(dic)
s
```

Out[11]:
```
대구    100
부산    150
서울    800
dtype: int64
```

In [12]:
```
city = ['서울', '부산', '대구', '대전']
```

In [13]:
```
s2 = Series(s, index=city)
s2
# 인덱스 '대전'에 해당하는 값이 없으므로 NaN으로 표시된다
```

Out[13]:
```
서울    800.0
부산    150.0
대구    100.0
대전      NaN
dtype: float64
```

In [14]:
```
# NaN값이면 True를 반환하고 값이 있을 경우 False를 반환한다
pd.isnull(s2)
```

Out[14]:
```
서울    False
부산    False
대구    False
대전     True
dtype: bool
```

In [15]:
```
# NaN값이면 False를 반환하고 값이 있을 경우 True를 반환한다
pd.notnull(s2)
```

Out[15]:
```
서울     True
부산     True
대구     True
대전    False
dtype: bool
```

In [16]:
```
# 아래도 같은 동작을 한다 (인스턴스 메서드라고 부른다)
s2.isnull()
```

Out[16]:
```
서울    False
부산    False
```

```
대구    False
대전     True
dtype: bool
```

In [17]:
```
s2
```

Out[17]:
```
서울    800.0
부산    150.0
대구    100.0
대전      NaN
dtype: float64
```

In [18]:
```
# 변수 선언 없이 바로 사전으로 시리즈 데이터 셋으로 변환할수 있다
s3 = Series({'서울':800, '광주':300, '대구': 100, '부산':50})
```

In [19]:
```
s3
```

Out[19]:
```
광주    300
대구    100
부산     50
서울    800
dtype: int64
```

In [20]:
```
# 두 시리즈에 연산을 하면 공통의 인덱스가 있는 부부만 처리된다
s2 + s3
```

Out[20]:
```
광주      NaN
대구    200.0
대전      NaN
부산    200.0
서울   1600.0
dtype: float64
```

In [21]:
```
s2 - s3
```

Out[21]:
```
광주      NaN
대구      0.0
대전      NaN
부산    100.0
서울      0.0
dtype: float64
```

실습예제 **2**

'BMW':15000, 'Nissan':8000, 'Honda':8000, 'Audi':10000를 사전(dictionary)으로 선언하고 해당 사전을 이용하여 시리즈 데이터셋을 선언하시오

In [ ]:
```

```

In [22]:
```
# 인데스나 시리즈 자체에 이름(name)을 줄 수 있다

s2.name = '물가지수'
s2.index.name = '도시명'
s2
```

Out[22]:
```
도시명
서울    800.0
부산    150.0
```

```
대구    100.0
대전     NaN
Name: 물가지수, dtype: float64
```

In [23]:
```
# 데이터프레임은 테이블 구조의 데이터를 다루며 각 컬럼마다 임의의 데이터 형식을 담을 수 있다
# 2차원 구조의 사전형식이라고 볼 수 있다
# 시리즈의 성격을 가지고 있다

# 사전으로부터 데이터프레임을 만드는 법

data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame(data)
frame
```

Out[23]:

	pop	state	year
0	1.5	Ohio	2000
1	1.7	Ohio	2001
2	3.6	Ohio	2002
3	2.4	Nevada	2001
4	2.9	Nevada	2002

In [24]:
```
# 컬럼의 순서를 변경할 수 있다
DataFrame(data, columns=['year', 'state', 'pop'])
```

Out[24]:

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9

In [25]:
```
# 없는 컬럼 값을 주면 NaN으로 처리된다 (아래에서 debt)
# 아래에서 인덱스 값을 새롭게 정의할 수 있다
frame2 = DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
                   index=['one', 'two', 'three', 'four', 'five'])
frame2
```

Out[25]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN

실습예제 **3**

'name': ['Aaron Brooks', 'Brendan Haywood', 'Cory Joseph', 'Darius Miller'], 'height': [180, 210, 187.5, 200], 'weight': [72.45, 120.6, 86.85, 105.75]

인 사전을 이용하여 tFrame이란 이름을 가진 데이터프레임을 만드시오

In [26]:
```
sdata = {'name': ['Aaron Brooks', 'Brendan Haywood', 'Cory Joseph', 'Darius Miller'],
        'height': [180, 210, 187.5, 200],
        'weight': [72.45, 120.6, 86.85, 105.75]}
tFrame = DataFrame(sdata)
tFrame
```

Out[26]:

	height	name	weight
0	180.0	Aaron Brooks	72.45
1	210.0	Brendan Haywood	120.60
2	187.5	Cory Joseph	86.85
3	200.0	Darius Miller	105.75

실습예제 **4**

인덱스를 'one', 'two', 'three', 'four'로 재정의 하시오

In [ ]:
```

```

실습예제 **5**

컬럼의 순서를 name, height, weight순으로 변경하시오

In [ ]:
```

```

In [27]:
```
frame2.columns
```

Out[27]:
```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

In [28]:
```
# 컬럼(열)을 얻는 방법
frame2['state']
```

Out[28]:
```
one       Ohio
two       Ohio
three     Ohio
four    Nevada
five    Nevada
Name: state, dtype: object
```

In [29]:
```
# 같은 결과를 얻는다 (속성 값으로 액세스)
frame2.state
```

Out[29]:
```
one       Ohio
two       Ohio
three     Ohio
four    Nevada
five    Nevada
Name: state, dtype: object
```

In [30]:
```
# 로(행)을 얻는 법
frame2.ix['three']
```

Out[30]:
```
year    2002
state   Ohio
pop      3.6
debt     NaN
Name: three, dtype: object
```

실습예제 6

tFrame의 name값을 출력하시오

In [ ]:
```

```

실습예제 7

tFrame의 컬럼(열)을 출력하시오

In [ ]:
```

```

In [31]:
```
# 컬럼 값을 채워널 수 있다
# 벡터화 방법
frame2['debt'] = 16.5
frame2
```

Out[31]:

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5

In [32]:
```
# 갯수가 일치해야 한다
# numpy.arange(start, stop, step)는 start부터 stop까지 step간격으로 된 배열을 반환한다
# start는 default값으로 0을 가지고 step는 default값으로 1을 가진다
frame2['debt'] = np.arange(5)
frame2
```

Out[32]:

	year	state	pop	debt
one	2000	Ohio	1.5	0
two	2001	Ohio	1.7	1
three	2002	Ohio	3.6	2
four	2001	Nevada	2.4	3
five	2002	Nevada	2.9	4

In [33]:
```
# 시리즈를 사용하여 컬럼 값을 입력할 수 있다 (없는 인덱스는 NaN이 된다)
val = Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
```

```
frame2['debt'] = val
frame2
```

Out[33]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7

In [34]:
```
#frame2에 eastern 칼럼을 만들고 frame2.state가 Ohio면 True값을 넣고 아닐경우 False를 넣는다
frame2['eastern'] = frame2.state == 'Ohio'
frame2
```

Out[34]:

	year	state	pop	debt	eastern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True
four	2001	Nevada	2.4	-1.5	False
five	2002	Nevada	2.9	-1.7	False

실습예제 **8**

tFrame에 key 칼럼을 만들고 height가 200이상이면 True값을 넣고 200미만일 경우 False값을 넣으시오

In [ ]:
```

```

In [35]:
```
# 컬럼을 간단히 삭제하는 명령으로 del이 있다
# 참고로 복사본을 만들지 않고 원본을 바로 수정하므로 주의해야 한다
del frame2['eastern']
frame2.columns
```

Out[35]:
```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

실습예제 **9**

del명령어를 사용하여 tFrame에서 key 칼럼을 삭제하고 칼럼을 출력하시오

In [ ]:
```

```

In [36]:
```
# 중첩된 사전으로부터 바로 데이터프레임을 만들 수 있다
# 바깥의 인덱스 명이 열이 된다
pop = {'Nevada': {2001: 2.4, 2002: 2.9},
       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
frame3 = DataFrame(pop)
frame3
```

Out[36]:

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

In [37]:
```
frame3.T
# 행과 열을 간단히 바꿀 수 있다 (transpose)
```

Out[37]:

	2000	2001	2002
Nevada	NaN	2.4	2.9
Ohio	1.5	1.7	3.6

In [38]:
```
# 인덱스를 명시적으로 지정할 수 있다
# 값이 없으면 NaN이 된다
DataFrame(pop, index=[2001, 2002, 2003])
```

Out[38]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2003	NaN	NaN

In [39]:
```
# 인덕스와 열의 이를을 지정할 수 있다
frame3.index.name = 'year'; frame3.columns.name = 'state'
frame3
```

Out[39]:

state	Nevada	Ohio
year		
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

In [40]:
```
frame3.values
```

Out[40]:
```
array([[ nan,  1.5],
       [ 2.4,  1.7],
       [ 2.9,  3.6]])
```

In [41]:
```
frame2.values
```

Out[41]:
```
array([[2000, 'Ohio', 1.5, nan],
       [2001, 'Ohio', 1.7, -1.2],
       [2002, 'Ohio', 3.6, nan],
       [2001, 'Nevada', 2.4, -1.5],
       [2002, 'Nevada', 2.9, -1.7]], dtype=object)
```

In [42]:
```
# 색인(index) 객체
obj = Series(range(3), index=['a', 'b', 'c'])
index = obj.index
index
```

Out[42]: Index(['a', 'b', 'c'], dtype='object')

In [43]:
```
index[1:]
```

Out[43]: Index(['b', 'c'], dtype='object')

In [44]:
```
index[1] = 'd'
# 색인은 변경할 수 없다
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-8b0a29b62131> in <module>()
----> 1 index[1] = 'd'
      2 # 색인은 변경할 수 없다

C:\ProgramData\Anaconda3\lib\site-packages\pandas\indexes\base.py in __setitem__(self, key, value)
   1402
   1403     def __setitem__(self, key, value):
-> 1404         raise TypeError("Index does not support mutable operations")
   1405
   1406     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

In [45]:
```
# 시리즈와 데이터프레이의 주요 기능
# reindex: 새로 정하는 색인의 순서대로 시리즈를 재배열한다
obj = Series([10,20,30,40], index=['d', 'b', 'a', 'c'])
obj
```

Out[45]:
```
d    10
b    20
a    30
c    40
dtype: int64
```

In [46]:
```
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
obj2
```

Out[46]:
```
a    30.0
b    20.0
c    40.0
d    10.0
e     NaN
dtype: float64
```

In [47]:
```
# fill_value값을 주면 NaN값을 해당 값으로 대체한다
obj.reindex(['a', 'b', 'c', 'd', 'e'], fill_value=0)
```

Out[47]:
```
a    30
b    20
c    40
d    10
e     0
dtype: int64
```

In [48]:
```
obj3 = Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
obj3.reindex(range(6), method='ffill')
# 앞의 값으로 채운다 forward fill (pad)
```

Out[48]:
```
0     blue
```

```
1     blue
2   purple
3   purple
4   yellow
5   yellow
dtype: object
```

In [49]:
```
# 뒤의 값으로 채우려면 bfill(backward fill)을 사용한다
obj3.reindex(range(6), method='bfill')
```

Out[49]:
```
0     blue
1   purple
2   purple
3   yellow
4   yellow
5      NaN
dtype: object
```

In [50]:
```
# reshape함수를 이용하여 배열의 형태를 재설정 할 수있다
# reshape(A, B)는 배열을 AxB로 바꿔준다
frame = DataFrame(np.arange(9).reshape((3, 3)), index=['a', 'c', 'd'],
                  columns=['Ohio', 'Texas', 'California'])
frame
```

Out[50]:

	Ohio	Texas	California
a	0	1	2
c	3	4	5
d	6	7	8

In [51]:
```
# 기본적으로는 행을 중심으로 재배열을 수행한다
frame2 = frame.reindex(['a', 'b', 'c', 'd'])
frame2
```

Out[51]:

	Ohio	Texas	California
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

In [52]:
```
# 열의 이름을 명시적으로 지정하여 재배열할 수도 있다
states = ['Texas', 'Utah', 'California']
frame.reindex(columns=states)
```

Out[52]:

	Texas	Utah	California
a	1	NaN	2
c	4	NaN	5
d	7	NaN	8

In [53]:
```
frame.ix[['a', 'b', 'c', 'd'], states]
```

Out[53]:

	Texas	Utah	California

a	1.0	NaN	2.0
b	NaN	NaN	NaN
c	4.0	NaN	5.0
d	7.0	NaN	8.0

In [54]:
```
# drop으로 행 삭제하기
obj = Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
new_obj = obj.drop('c')
new_obj
```

Out[54]:
```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

In [55]:
```
obj.drop(['d', 'c'])
```

Out[55]:
```
a    0.0
b    1.0
e    4.0
dtype: float64
```

In [56]:
```
data = DataFrame(np.arange(16).reshape((4, 4)),
                 index=['Ohio', 'Colorado', 'Utah', 'New York'],
                 columns=['one', 'two', 'three', 'four'])
```

In [57]:
```
data.drop(['Colorado', 'Ohio'])
```

Out[57]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

In [58]:
```
data.drop('two', axis=1)
```

Out[58]:

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

In [59]:
```
data.drop(['two', 'four'], axis=1)
```

Out[59]:

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

실습예제 **10**

drop 함수를 이용하여 tFrame의 첫번째 행을 삭제하시오

In [ ]:

실습예제 **11**

drop함수를 이용하여 name 칼럼을 삭제하고 원본데이터에 적용하시오

In [ ]:

실습예제 **12**

del함수와 drop함수의 차이점을 서술하시오.

In [60]:
```
obj = Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
obj['b']
```

Out[60]: 1.0

In [61]:
```
obj[2:4]
```

Out[61]:
```
c    2.0
d    3.0
dtype: float64
```

In [62]:
```
obj[[1,3]]
```

Out[62]:
```
b    1.0
d    3.0
dtype: float64
```

In [63]:
```
obj[['b','a','c']]
```

Out[63]:
```
b    1.0
a    0.0
c    2.0
dtype: float64
```

In [64]:
```
obj[obj<2]
```

Out[64]:
```
a    0.0
b    1.0
dtype: float64
```

In [65]:
```
obj['b':'c']
# 양 끝점을 포함시킨다
```

Out[65]:
```
b    1.0
c    2.0
dtype: float64
```

In [66]:
```
data = DataFrame(np.arange(16).reshape((4, 4)),
```

```
            index=['Ohio', 'Colorado', 'Utah', 'New York'],
            columns=['one', 'two', 'three', 'four'])
data
```

Out[66]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

실습예제 **13**

인덱스가 'Brendan Haywood', 'Cory Joseph', 'Darius Miller'이고 컬럼(열)이 'one', 'two', 'three', 'four', 'five'인 3x5 데이터프레임을 만드시오(이름은 sample으로 하고, 값은 0부터 차례대로 채우시오)

In [ ]:

In [67]:
```
data['two']
# 'two'라는 컬럼을 가진 값들을 가져온다
```

Out[67]:
```
Ohio        1
Colorado    5
Utah        9
New York    13
Name: two, dtype: int32
```

In [68]:
```
data[['three', 'one']]
# 'three'와 'one'이라는 컬럼을 가진 값들을 가져온다
```

Out[68]:

	three	one
Ohio	2	0
Colorado	6	4
Utah	10	8
New York	14	12

In [69]:
```
data[:2]
# 2까지 로(행)을 가져온다
```

Out[69]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

In [70]:
```
data[data['three'] > 5]
# 칼럼 'three'의 값이 5 초과인 값들을 가져온다
```

Out[70]:

	one	two	three	four
Colorado	4	5	6	7

Utah	8	9	10	11
New York	12	13	14	15

실습예제 **14**

sample 데이터에서 컬럼 four의 값이 5 이상인 값들을 가져오시오

In [ ]:

실습예제 **15**

sample 데이터에서 2번째부터 3번째까지 로(행)을 가져오시오

In [ ]:

실습예제 **16**

sample 데이터에서 칼럼 'one'과 'five'를 가져오시오

In [ ]:

In [71]:
```
data.ix['Colorado', ['two', 'three']]
# Colorado 로(행)의 'two'와 'three'값을 Series 구조로 가져온다
```

Out[71]:
```
two     5
three   6
Name: Colorado, dtype: int32
```

In [72]:
```
data.ix[['Colorado', 'Utah'], [3, 0, 1]]
# Colorado와 'Utah'행의 3, 0, 1번째 컬럼(열) 순서로 가져온다
```

Out[72]:

	four	one	two
Colorado	7	4	5
Utah	11	8	9

In [73]:
```
data.ix[2]
# 2번째 로(행)의 값을 가져온다
```

Out[73]:
```
one     8
two     9
three   10
four    11
Name: Utah, dtype: int32
```

In [74]:
```
data.ix[:'Utah', 'two']
# 0부터 Utah까지 행의 컬럼 'two' 값을 가져온다
```

Out[74]:
```
Ohio      1
Colorado  5
Utah      9
Name: two, dtype: int32
```

In [75]:
```
data.ix[data.three > 5, :3]
# 컬럼 three의 값이 5 초과인 값의 컬럼 3번째 까지 값을 가져온다
```

Out[75]:

	one	two	three
Colorado	4	5	6
Utah	8	9	10
New York	12	13	14

실습예제 **17**

sample 데이터의 five 칼럼이 9인 값의 컬럼 3번째 까지 값을 가져오시오

In [ ]:
```
```

In [76]:
```
df1 = DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),
                index=['Ohio', 'Texas', 'Colorado'])
df2 = DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),
                index=['Utah', 'Ohio', 'Texas', 'Oregon'])
df1
```

Out[76]:

	b	c	d
Ohio	0.0	1.0	2.0
Texas	3.0	4.0	5.0
Colorado	6.0	7.0	8.0

In [77]:
```
df2
```

Out[77]:

	b	d	e
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0

In [78]:
```
df1 + df2
```

Out[78]:

	b	c	d	e
Colorado	NaN	NaN	NaN	NaN
Ohio	3.0	NaN	6.0	NaN
Oregon	NaN	NaN	NaN	NaN
Texas	9.0	NaN	12.0	NaN
Utah	NaN	NaN	NaN	NaN

In [79]:
```
df1 = DataFrame(np.arange(12.).reshape((3, 4)), columns=list('abcd'))
df2 = DataFrame(np.arange(20.).reshape((4, 5)), columns=list('abcde'))
df1
```

Out[79]:

	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

In [80]:
```
df2
```

Out[80]:

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	6.0	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

In [81]:
```
df1 + df2
```

Out[81]:

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	NaN
1	9.0	11.0	13.0	15.0	NaN
2	18.0	20.0	22.0	24.0	NaN
3	NaN	NaN	NaN	NaN	NaN

In [82]:
```
df1.add(df2, fill_value=0)
```

Out[82]:

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	11.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

In [83]:
```
# 함수 적용
frame = DataFrame(np.random.randn(4, 3), columns=list('bde'),
                  index=['Utah', 'Ohio', 'Texas', 'Oregon'])
frame
```

Out[83]:

	b	d	e
Utah	-0.447613	-0.433355	0.612768
Ohio	-1.252928	-2.373916	0.165255
Texas	-0.377568	-0.275341	1.730919
Oregon	1.284537	-1.490655	-0.933486

In [84]:
```
np.abs(frame)
```

Out[84]:

	b	d	e

Utah	0.447613	0.433355	0.612768
Ohio	1.252928	2.373916	0.165255
Texas	0.377568	0.275341	1.730919
Oregon	1.284537	1.490655	0.933486

In [85]:
```
f = lambda x: x.max() - x.min()
```

In [86]:
```
frame.apply(f)
```

Out[86]:
```
b    2.537464
d    2.098575
e    2.664405
dtype: float64
```

In [87]:
```
frame.apply(f,axis=1)
```

Out[87]:
```
Utah      1.060381
Ohio      2.539172
Texas     2.108487
Oregon    2.775192
dtype: float64
```

In [88]:
```
# 리턴 값이 스칼라가 아니라 시리즈로 될 수도 있다
def f(x):
    return Series([x.min(), x.max()], index=['min', 'max'])
frame.apply(f)
```

Out[88]:

	b	d	e
min	-1.252928	-2.373916	-0.933486
max	1.284537	-0.275341	1.730919

In [89]:
```
# 포맷을 바꾸는 예
format = lambda x: '%.4f' % x # 소수 4번째 자리까지 표시한다
frame.applymap(format)
# DataFrame구조에 applymap 함수를 이용하여 값에 함수를 적용한다
```

Out[89]:

	b	d	e
Utah	-0.4476	-0.4334	0.6128
Ohio	-1.2529	-2.3739	0.1653
Texas	-0.3776	-0.2753	1.7309
Oregon	1.2845	-1.4907	-0.9335

In [90]:
```
frame['e'].map(format)
# Series구조에 applymap 함수를 이용하여 값에 함수를 적용한다
```

Out[90]:
```
Utah      0.6128
Ohio      0.1653
Texas     1.7309
Oregon    -0.9335
Name: e, dtype: object
```

실습예제 **18**

lambda를 이용하여 sample데이터의 평균을 구하시오

In [ ]:

DataFrame 고급기술

In [1]:
```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame # Pandas에는 Series 와 DataFrame이라는 두 종류의 모듈이 있습니다
```

In [2]:
```
# 정렬
# 인덱스를 정렬하는 방법
obj = Series(range(4), index=['d', 'a', 'b', 'c'])
# Series의 기본 구조는 index와 그 index에 맵핑에 되는 values 값이 있습니다.
# Series(values 값 , index = index 값) // Series(values 값) 은 index가 기본적으로 0부터 시작됩니다.
# ex) 차례로 values 값이 2,4,6,8 이며 인덱스 값은 1,3,5,7이다.
#    exam = Series([2,4,6,8], index=[1,3,5,7])
obj
```

Out[2]:
```
d    0
a    1
b    2
c    3
dtype: int32
```

In [3]:
```
obj.sort_index() ## obj의 index를 osrt한 결과
```

Out[3]:
```
a    1
b    2
c    3
d    0
dtype: int32
```

In [4]:
```
obj ## 위에서 sort를 해줬으나 obj의 결과는 변하지 않는다.
```

Out[4]:
```
d    0
a    1
b    2
c    3
dtype: int32
```

실습 예제 **1**

series1, series2라는 Series 기본구조를 만든다. series1은 index는 ㄱ,ㄷ,ㄹ,ㄴ 순으로 하며, value 값은 index에 자음이 들어가는 단어로 자유롭게 정하시오. series2 에는 series1을 index 순으로 정렬된 Series를 넣으시고 series1, series2를 각각 출력하시오.

In [ ]:
```
```

In [5]:
```
frame = DataFrame(np.arange(8).reshape((2, 4)), index=['three', 'one'],
                  columns=['d', 'a', 'b', 'c'])
## frame변수에 DataFrame이라는 구조를 넣고 있습니다.
## DataFrame(value값, index값(행), columns값(열))
frame
```

Out[5]:

	d	a	b	c
three	0	1	2	3
one	4	5	6	7

In [6]:
```
frame.sort_index()
```

Out[6]:

	d	a	b	c
one	4	5	6	7
three	0	1	2	3

In [7]:
```
frame.sort_index(axis=1)
# 열을 기준으로 정렬
```

Out[7]:

	a	b	c	d
three	1	2	3	0
one	5	6	7	4

In [8]:
```
frame.sort_index(axis=1, ascending=False)
# 내림차순으로 정렬
```

Out[8]:

	d	c	b	a
three	0	3	2	1
one	4	7	6	5

실습예제2

위 frame.sort_index(axis=? , ascending = ?) 에서 axis = (0 or 1), ascending = (False, True)를 자유롭게 넣어 보고 결과를 살펴본 뒤 frame2 = DataFrame(np.arange(9).reshape((3, 3)), index=['2', '3', '1'], columns=['3','1','2'])를 행과 열을 1,2,3 순으로 바꾼 뒤 출력해보시오.

In [ ]:
```

```

In [9]:
```
frame = DataFrame({'b': [4, 7, 3, 2], 'a': [4,9,2,5], 'c': [5,3,7,9]})
frame
```

Out[9]:

	a	b	c
0	4	4	5
1	9	7	3
2	2	3	7
3	5	2	9

In [10]:
```
frame.sort_values(by='b')
```

Out[10]:

	a	b	c
3	5	2	9

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

In [11]:
```
frame.sort_values(by='a')
```

Out[11]:

	a	b	c
2	2	3	7
0	4	4	5
3	5	2	9
1	9	7	3

실습예제3

행 'c'를 기준으로 정렬한 Dataframe을 변수 frame2에 넣어 출력하시오.

In [ ]:
```
```

In [12]:
```
# 순위 매기기 rank
obj = Series([100, 23, 55, 44, 22, 99, 33])
obj.rank()
# 작은 값에 높은 순위를 준다
```

Out[12]:
```
0    7.0
1    2.0
2    5.0
3    4.0
4    1.0
5    6.0
6    3.0
dtype: float64
```

In [13]:
```
# 내림차순으로 순위를 매긴다
obj.rank(ascending=False)
```

Out[13]:
```
0    1.0
1    6.0
2    3.0
3    4.0
4    7.0
5    2.0
6    5.0
dtype: float64
```

In [14]:
```
obj = Series([100, 23, 100, 44, 22, 99, 33])
obj.rank()
# 동점이 있으면 평균값을 준다
```

Out[14]:
```
0    6.5
1    2.0
2    6.5
3    4.0
4    1.0
```

```
5    5.0
6    3.0
dtype: float64
```

In [15]:
```
obj.rank(method='first')
# 동일한 값이 존재 할 경우 먼저 나타나는 것에게 높은 순위를 줄 수 있다
```

Out[15]:
```
0    6.0
1    2.0
2    7.0
3    4.0
4    1.0
5    5.0
6    3.0
dtype: float64
```

In [16]:
```
frame = DataFrame({'b': [4,7,3,2], 'a': [4,9,2,5], 'c': [5,3,7,9]})
frame
```

Out[16]:

	a	b	c
0	4	4	5
1	9	7	3
2	2	3	7
3	5	2	9

실습예제 **4**

frame에 rank를 적용해보시오.

In [ ]:
```
```

In [17]:
```
frame.rank(axis=1) #행 기준으로 rank를 수행
```

Out[17]:

	a	b	c
0	1.5	1.5	3.0
1	3.0	2.0	1.0
2	1.0	2.0	3.0
3	2.0	1.0	3.0

In [18]:
```
# 합을 구하면 기본적으로 열을 기준으로 구한다
frame.sum()
```

Out[18]:
```
a    20
b    16
c    24
dtype: int64
```

In [19]:
```
# 행을 기준으로 구한다
frame.sum(axis=1)
```

Out[19]:
```
0    13
1    19
```

```
2    12
3    16
dtype: int64
```

In [20]:
```
# NaN이 있으면
frame = DataFrame({'b': [4, 7, 3, 2], 'a': [4,9,2,5], 'c': [5,3,7,np.nan
]})
frame
```

Out[20]:

	a	b	c
0	4	4	5.0
1	9	7	3.0
2	2	3	7.0
3	5	2	NaN

In [21]:
```
frame.sum()
```

Out[21]:
```
a    20.0
b    16.0
c    15.0
dtype: float64
```

In [22]:
```
frame.sum(skipna=False)
# NaN이 있으면 이를 반영한다 (스킵하지 않는다)
# skipna 은 skip NaN을 뜻함.
```

Out[22]:
```
a    20.0
b    16.0
c     NaN
dtype: float64
```

실습예제 **5**

frame 합을 행을 기준으로 NaN이 있으면 이를 반영하지 않는 합을 구해보시오.

In [ ]:
```

```

In [23]:
```
frame = DataFrame({'b': [4, 7, 3, 2], 'a': [4,9,2,5], 'c': [5,3,7,np.nan
]})
frame
```

Out[23]:

	a	b	c
0	4	4	5.0
1	9	7	3.0
2	2	3	7.0
3	5	2	NaN

In [24]:
```
# 최대치가 있는 위치를 반환한다
frame.idxmax()
```

Out[24]:
```
a    1
b    1
```

```
c    2
dtype: int64
```

In [25]:
```
# 최소치가 있는 위치를 반환한다
frame.idxmin()
```

Out[25]:
```
a    2
b    3
c    1
dtype: int64
```

In [26]:
```
# 기본 통계값을 보여준다
frame.describe()
# count = 갯수, mean = 평균, std = 표준편차 ,min = 최솟값, max = 최댓값
```

Out[26]:

	a	b	c
count	4.00000	4.000000	3.0
mean	5.00000	4.000000	5.0
std	2.94392	2.160247	2.0
min	2.00000	2.000000	3.0
25%	3.50000	2.750000	4.0
50%	4.50000	3.500000	5.0
75%	6.00000	4.750000	6.0
max	9.00000	7.000000	7.0

In [27]:
```
# 항목 갯수 세기

# 유니크한 값 찾기 (set)
obj = Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
obj
```

Out[27]:
```
0    c
1    a
2    d
3    a
4    a
5    b
6    b
7    c
8    c
dtype: object
```

In [28]:
```
uniques = obj.unique() ## value 값의 종류를 확인 할 수 있다.
uniques
```

Out[28]:
```
array(['c', 'a', 'd', 'b'], dtype=object)
```

In [29]:
```
# 빈도수를 간단히 알 수 있다
# 빈도수가 높은 순으로 정렬된다.
obj.value_counts()
```

Out[29]:
```
a    3
c    3
b    2
```

```
d    1
dtype: int64
```

In [30]:
```
# 빈도수와 관련없이 나타나는 순서대로 보려면
pd.value_counts(obj.values, sort=False)
```

Out[30]:
```
c    3
d    1
b    2
a    3
dtype: int64
```

In [31]:
```
# 아래는 같은 결과를 얻는다
obj.value_counts(sort=False)
```

Out[31]:
```
c    3
d    1
b    2
a    3
dtype: int64
```

In [32]:
```
# 특정한 내용이 들어있는지 알려면 isin()을 사용한다
mask = obj.isin(['b', 'c'])
mask
```

Out[32]:
```
0     True
1    False
2    False
3    False
4    False
5     True
6     True
7     True
8     True
dtype: bool
```

In [33]:
```
obj## obj 값 확인
```

Out[33]:
```
0    c
1    a
2    d
3    a
4    a
5    b
6    b
7    c
8    c
dtype: object
```

In [34]:
```
obj[mask] ## obj에서 mask 값이 true인 값만을 출력
```

Out[34]:
```
0    c
5    b
6    b
7    c
8    c
dtype: object
```

In [35]:
```
# 아래는 같은 결과를 얻는다
obj[obj.isin(['b', 'c'])] ## isin는 is in을 뜻한다.
```

Out[35]:
```
0    c
5    b
6    b
7    c
8    c
dtype: object
```

실습예제6

obj 에서 isin 문을 자유롭게 활용해보시오.

In [ ]:

In [36]:
```
# 데이터프레임에 대해서도 같은 작업을 할 수 있다
frame = DataFrame({'X':['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'],
                   'Y':['f', 'g', 'd', 'g', 'h', 'e', 'd', 'h', 'f'],
                    'Z':['a', 'e', 'd', 'g', 'd', 'e', 'q', 'b', 'c']})
```

In [37]:
```
frame
```

Out[37]:

	X	Y	Z
0	c	f	a
1	a	g	e
2	d	d	d
3	a	g	g
4	a	h	d
5	b	e	e
6	b	d	q
7	c	h	b
8	c	f	c

In [38]:
```
# 객수를 센다. 없는 값은 NaN으로 표시된다
result = frame.apply(pd.value_counts)
result
```

Out[38]:

	X	Y	Z
a	3.0	NaN	1.0
b	2.0	NaN	1.0
c	3.0	NaN	1.0
d	1.0	2.0	2.0
e	NaN	1.0	2.0
f	NaN	2.0	NaN
g	NaN	2.0	1.0
h	NaN	2.0	NaN

q	NaN	NaN	1.0

In [39]:
```
# 없는 값에 0을 대입한다
result = frame.apply(pd.value_counts).fillna(0) ## fillna 은 fii NaN을 뜻한다.
result
```

Out[39]:

	X	Y	Z
a	3.0	0.0	1.0
b	2.0	0.0	1.0
c	3.0	0.0	1.0
d	1.0	2.0	2.0
e	0.0	1.0	2.0
f	0.0	2.0	0.0
g	0.0	2.0	1.0
h	0.0	2.0	0.0
q	0.0	0.0	1.0

In [40]:
```
# 결측치 처리 , 결측치 란? python NaN으로 값이 없다는 뜻이다.
from numpy import nan as NA ## numpy 안에서 nan을 Na으로 대체한다는 뜻입니다.
data = Series([1, NA, 3.5, NA, 7])
data.dropna() # drop na은 drop NaN을 뜻하여 data에서 Na값들이 떨어져 나간 것을 볼 수 있습니다.
```

Out[40]:
```
0    1.0
2    3.5
4    7.0
dtype: float64
```

In [41]:
```
# 같은 결과
data[data.notnull()]
```

Out[41]:
```
0    1.0
2    3.5
4    7.0
dtype: float64
```

In [42]:
```
data = DataFrame([[NA, 6.5, 3.], [NA, NA, NA],
                  [NA, NA, NA], [NA, 6.5, 3.]])
data
```

Out[42]:

	0	1	2
0	NaN	6.5	3.0
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

In [43]:
```
# 한 항목이라고 NA가 있으면 해당 행을 삭제한다
cleaned = data.dropna()
cleaned
```

Out[43]:

	0	1	2

In [44]:
```
# 행의 모든 항목이 NA일때 해당 행을 삭제한다
data.dropna(how='all')
```

Out[44]:

	0	1	2
0	NaN	6.5	3.0
3	NaN	6.5	3.0

In [45]:
```
data
```

Out[45]:

	0	1	2
0	NaN	6.5	3.0
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

In [46]:
```
# 컬럼에 대한 삭제시는 axis=1을 사용한다
clean2 = data.dropna(axis=1)
clean2
```

Out[46]:

0
1
2
3

In [47]:
```
clean2 = data.dropna(axis=1, how='all')
clean2
```

Out[47]:

	1	2
0	6.5	3.0
1	NaN	NaN
2	NaN	NaN
3	6.5	3.0

In [48]:
```
df = DataFrame(np.random.randn(7 , 3))
df.ix[:4, 1] = NA #(0,1) , (1,1) , (2,1) , (3,1) , (4,1) 값들이 Na로 대체된 걸 확인 할 수 있다.
df.ix[:2, 2] = NA
df.ix[0,0] = NA
df
```

Out[48]:

	0	1	2
0	NaN	NaN	NaN
1	-0.551445	NaN	NaN

2	-0.827230	NaN	NaN
3	2.939233	NaN	-1.445335
4	-1.336096	NaN	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

실습예제**7**

위에서 np.random.randn(7,3)을 단독적으로 실행해보고 결과 값을 확인 해보시오. 활용 하여 np.random.randn 6 * 4 행렬을 만들고 DataFrame 형식으로 나타내보시오.

In [ ]:

실습예제**8**

위 예제에서 만들었던 DataFrame에 (0,0) (1,1) (2,2) (3,3) 위치에 있는 값들을 Na 값으로 대체해보시오.

In [ ]:

In [49]:
```
df # df 확인
```

Out[49]:

	0	**1**	**2**
0	NaN	NaN	NaN
1	-0.551445	NaN	NaN
2	-0.827230	NaN	NaN
3	2.939233	NaN	-1.445335
4	-1.336096	NaN	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

In [50]:
```
df.dropna(thresh=2) ## NaN 값이 2 이상인 열을 제거한다.
```

Out[50]:

	0	**1**	**2**
3	2.939233	NaN	-1.445335
4	-1.336096	NaN	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

In [51]:
```
df.fillna(0) ## fillna 은 fill nan을 뜻합니다. nan이 0으로 대체된 것을 볼 수 있습니다.
```

Out[51]:

	0	**1**	**2**
0	0.000000	0.000000	0.000000

1	-0.551445	0.000000	0.000000
2	-0.827230	0.000000	0.000000
3	2.939233	0.000000	-1.445335
4	-1.336096	0.000000	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

In [52]:
```
df ## 위와 같이 작업했으나 실제 df에는 영향을 주지 않습니다.
```

Out[52]:

	0	**1**	**2**
0	NaN	NaN	NaN
1	-0.551445	NaN	NaN
2	-0.827230	NaN	NaN
3	2.939233	NaN	-1.445335
4	-1.336096	NaN	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

In [53]:
```
# 컬럼별로 다른 값을 채울 수 있다. 사전을 사용한다
# 사전이란? 아래의 {1: 0.5, 2 : -1} 형태의 자료구조입니다.
df.fillna({1: 0.5, 2: -1})
```

Out[53]:

	0	**1**	**2**
0	NaN	0.500000	-1.000000
1	-0.551445	0.500000	-1.000000
2	-0.827230	0.500000	-1.000000
3	2.939233	0.500000	-1.445335
4	-1.336096	0.500000	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

In [54]:
```
df # fillna()로 내용은 바뀌지 않는다. 위도 마찬가지로 df에는 영향을 주지 않는 것을 볼 수 있습니다.
```

Out[54]:

	0	**1**	**2**
0	NaN	NaN	NaN
1	-0.551445	NaN	NaN
2	-0.827230	NaN	NaN
3	2.939233	NaN	-1.445335
4	-1.336096	NaN	-2.308019

5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

In [55]:
```
# 새로운 변수를 정의하면 바뀐 값을 얻는다
df2 = df.fillna({1: 0.5, 3: -1})
df2
```

Out[55]:

	0	**1**	**2**
0	NaN	0.500000	NaN
1	-0.551445	0.500000	NaN
2	-0.827230	0.500000	NaN
3	2.939233	0.500000	-1.445335
4	-1.336096	0.500000	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

In [56]:
```
# fillna함수의 inplace 파라미터를 true로 정의하면 내부 변경이 가능하다.
df.fillna({1: 0.5, 3: -1}, inplace=True)
```

Out[56]:

	0	**1**	**2**
0	NaN	0.500000	NaN
1	-0.551445	0.500000	NaN
2	-0.827230	0.500000	NaN
3	2.939233	0.500000	-1.445335
4	-1.336096	0.500000	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

In [57]:
```
# 바뀌어 있다
df
```

Out[57]:

	0	**1**	**2**
0	NaN	0.500000	NaN
1	-0.551445	0.500000	NaN
2	-0.827230	0.500000	NaN
3	2.939233	0.500000	-1.445335
4	-1.336096	0.500000	-2.308019
5	-0.696401	-1.338543	0.335230
6	-2.344175	-1.570471	1.619617

실습예제**9**

np.random.randn을 활용해 3*3 Dataframe 구조를 만든 뒤 값들을 확인하여 음수에 해당하는 값들을 0으로 대체해보시오.

In [ ]:

데이터프레임_더 나아가기

DataFrame의 data를 다루는 방법에 대하여 학습한다.

In [1]:
```
from pandas import Series, DataFrame
import pandas as pd
from numpy.random import randn
import numpy as np
import os
```

누락된 데이터 채우기

In [2]:
```
df=DataFrame(np.random.randn(7,3)) ## (7, 3)로 된 데이터 프레임을 생성하며 안의 값은 랜덤으로 생성한다.
df.ix[:4,1] = None   #데이터 프레임의 (5, 1)까지 NAN 값으로 채우며 입력된 열에서 해당 행까지 진행된다.
df.ix[:2,2] = None
df                   # 해당 변수의 값 출력(df에는 데이터프레임이 들어가있기때문에 데이터프레임을 출력한다.)
# 이를 반대로 행한다면 결측치를 특정 값으로 바꾸는것이 가능하다.
```

Out[2]:

	0	1	2
0	-0.905071	NaN	NaN
1	3.477435	NaN	NaN
2	-0.893022	NaN	NaN
3	0.856653	NaN	-0.600397
4	0.738119	NaN	-0.415846
5	0.030193	0.540700	0.984515
6	-0.215206	1.351922	0.098967

실습 예제 **1**번

(5,4)으로 구성되었고 안의 값은 랜덤인 데이터 프레임을 만드시오. 만든 후 4행까지의 1열과 2행까지의 3열을 모두 NaN으로 바꾸어 출력하시오

(데이터프레임의 이름은 prat이며 이후 예제에도 prat를 사용)

In [ ]:
```

```

In [3]:
```
df.dropna()   ## dropna는 행이나 열에서 NaN 값이 하나라도 있을 경우  생략하고 출력한다.
```

Out[3]:

	0	1	2
5	0.030193	0.540700	0.984515
6	-0.215206	1.351922	0.098967

실습 예제 **2**번

NaN값을 제외해서 출력하시오

In [ ]:

In [4]:
```
df.fillna(0)    # fillna는 NaN 값을 괄호 안에 있는 값으로 채워준다.
# 단 새로운 변수를 만들어 변경된 dataframe을 저장하거나, inplace 파라미터를 사용하지 않는다면 변경 값은 저장되지 않는다.
```

Out[4]:

	0	1	2
0	-0.905071	0.000000	0.000000
1	3.477435	0.000000	0.000000
2	-0.893022	0.000000	0.000000
3	0.856653	0.000000	-0.600397
4	0.738119	0.000000	-0.415846
5	0.030193	0.540700	0.984515
6	-0.215206	1.351922	0.098967

실습 예제 **3**번

NaN값을 3으로 대체하시오

In [ ]:

In [5]:
```
# 해당사항이 2개 이상인 경우만 처리 만약 1로 고칠경우 1~3까지의 행도 같이 나온다.
df.dropna(thresh=2)
```

Out[5]:

	0	1	2
3	0.856653	NaN	-0.600397
4	0.738119	NaN	-0.415846
5	0.030193	0.540700	0.984515
6	-0.215206	1.351922	0.098967

실습 예제 **4**번

행에서 NaN값을 제외한 수가 1개 이상이 나오도록 출력하시오

In [ ]:

In [6]:
```
df.fillna({1: 0.5, 3: -1})
# 콜론 앞에 쓰여진 열의 값을 콜론 뒤에 값으로 채워주는 역할이다 만약 해당 열이 없을 경우 아무일도 일어나지않는다
```

Out[6]:

	0	1	2

0	-0.905071	0.500000	NaN
1	3.477435	0.500000	NaN
2	-0.893022	0.500000	NaN
3	0.856653	0.500000	-0.600397
4	0.738119	0.500000	-0.415846
5	0.030193	0.540700	0.984515
6	-0.215206	1.351922	0.098967

실습 예제 **5**번

2열은 2로 4열은 5로 값을 대체해보시오

In [ ]:

In [7]:
```
# 기존의 객체를 변경하도록 하려면 inplace를 사용한다
df.fillna(2, inplace=True)
df
```

Out[7]:

	0	**1**	**2**
0	-0.905071	2.000000	2.000000
1	3.477435	2.000000	2.000000
2	-0.893022	2.000000	2.000000
3	0.856653	2.000000	-0.600397
4	0.738119	2.000000	-0.415846
5	0.030193	0.540700	0.984515
6	-0.215206	1.351922	0.098967

실습 예제 **6**번

NaN값을 모두 10으로 변경하시오

In [ ]:

In [8]:
```
data = Series([1., None, 2, None, 7]) # 시리즈를 만든 것이며 괄호안의 수로 순서대로 값을 채웠다
data.fillna(data.mean())    # 결측치를 평균값으로 채운다
```

Out[8]:
```
0    1.000000
1    3.333333
2    2.000000
3    3.333333
4    7.000000
dtype: float64
```

실습 예제 **7**번

prat2 라는 이름을 가지고 1, NaN, 10, 4, Nan값을 가진 시리지를 만든 후 Nan값을 평균값으로 채워 보아라

(이후 예제는 prat2를 활용한다)

In [ ]:

In [9]:
```
## 10개의 값을 가진 시리즈를 만든 후

data = Series(np.random.randn(10),
              index=[['a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'd', 'd'],
                     [1, 2, 3, 1, 2, 3, 1, 2, 2, 3]])
data
```

Out[9]:
```
a  1   -1.040215
   2   -1.018088
   3   -0.348830
b  1   -1.482556
   2   -0.192941
   3   -0.885719
c  1    0.369514
   2   -1.068391
d  2    0.664071
   3   -2.009124
dtype: float64
```

실습 예제 **8**번

랜덤으로 10개의 값을 채운 시리즈에 상위 색인은 a(2),c(3),d(2) b(3)로 구성하며 하위 색인은 1~10까지의 수로 한다

In [ ]:

In [10]:
```
data.index # levels은 Series의 중복없이 색인 값을 모아놓은 것이며 labels은 그것을 숫자로 표현한 것이다.
```

Out[10]:
```
MultiIndex(levels=[['a', 'b', 'c', 'd'], [1, 2, 3]],
           labels=[[0, 0, 0, 1, 1, 1, 2, 2, 3, 3], [0, 1, 2, 0, 1, 2, 0,
 1, 1, 2]])
```

In [11]:
```
data['b'] # 색인이 b인 값들을 출력
```

Out[11]:
```
1   -1.482556
2   -0.192941
3   -0.885719
dtype: float64
```

In [12]:
```
data['b':'c'] ## 색인이 b부터 c까지에 값들을 출력
```

Out[12]:
```
b  1   -1.482556
   2   -0.192941
   3   -0.885719
c  1    0.369514
   2   -1.068391
dtype: float64
```

In [13]:
```
data.ix[['b', 'd']]  ## 색인이 b와 d인 것만 출력
```

Out[13]:
```
b  1   -1.482556
   2   -0.192941
   3   -0.885719
d  2    0.664071
   3   -2.009124
dtype: float64
```

실습 예제 **9**번

색인이 a와 d인 값들만 출력하시오

In [ ]:
```
```

In [14]:
```
# 하위계층의 객체를 선택
data[:, 2]
```

Out[14]:
```
a   -1.018088
b   -0.192941
c   -1.068391
d    0.664071
dtype: float64
```

In [15]:
```
# 데이터의 배열을 새롭게 구성할 수 있다
data.unstack() #상위 계층은 행으로 하위 계층은 열로 바뀌었다
```

Out[15]:

	1	2	3
a	-1.040215	-1.018088	-0.348830
b	-1.482556	-0.192941	-0.885719
c	0.369514	-1.068391	NaN
d	NaN	0.664071	-2.009124

실습 예제 **10**번

prat2라는 시리즈를 데이터프레임 형태로 만들어보시오

In [ ]:
```
```

In [16]:
```
data.unstack().stack() ## 데이터의 배열을 새롭게 구성한 후에 stack을 통해 다시 원래대로 만들었다.
```

Out[16]:
```
a  1   -1.040215
   2   -1.018088
   3   -0.348830
b  1   -1.482556
   2   -0.192941
   3   -0.885719
c  1    0.369514
   2   -1.068391
d  2    0.664071
   3   -2.009124
dtype: float64
```

In [17]:
```
# 컬럼도 계층적으로 만들 수 있다
```

```
frame = DataFrame(np.arange(12).reshape((4, 3)),  # (4,3)로 구성된 데이터 프레임 생성
                  index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2],], # 상위 색인은 a와 b로 구성
                  columns=[['Ohio', 'Ohio', 'Colorado'],  # index와 columns을 만들때 해당 몇행 몇열인지를 보고 만들어야한다
                           ['Green', 'Red', 'Green'], ])
frame
```

Out[17]:

		Ohio		Colorado
		Green	Red	Green
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

실습 예제 **11**번

prat3이름을 가지고 (5,4)로 구성된 데이터 프레임을 생성하시오 상위 색인은 a(2), b(2), c(1)로 구성되며 하위 색인은 1,2, 1, 2, 1로 구성한다.

columns은 'busan', 'naju', 'naju', 'seoul'와 'korea', 'korea', 'usa', 'usa'로 구성된다. (이후 예제는 prat3를 사용)

In [ ]:

In [18]:
```
# 색인에 이름을 지정할 수 있다
frame.index.names = ['key1', 'key2']
frame.columns.names = ['state', 'color']
frame
```

Out[18]:

	state	Ohio		Colorado
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

실습 예제 **12**번

columns에 이름을 붙이되 나라에는 country, 지역에는 region의 이름을 붙여 보아라.

In [ ]:

In [19]:
```
frame['Ohio'] #Ohio columns만 보이기
```

Out[19]:

	color	Green	Red
key1	key2		
a	1	0	1
	2	3	4
b	1	6	7
	2	9	10

데이터프레임 순서 변경 및 정렬

In [20]:
```
frame.swaplevel('key1', 'key2') # 색인 자리 바꾸기
```

Out[20]:

	state	Ohio		Colorado
	color	Green	Red	Green
key2	key1			
1	a	0	1	2
2	a	3	4	5
1	b	6	7	8
2	b	9	10	11

In [21]:
```
frame.swaplevel(0, 1).sortlevel(0)
# 괄호안의 첫 숫자의 색인과 두 번째 숫자의 색인을 바꾸고 정렬하였다 첫번째 색인인 key2에 따라 정렬했다
```

Out[21]:

	state	Ohio		Colorado
	color	Green	Red	Green
key2	key1			
1	a	0	1	2
	b	6	7	8
2	a	3	4	5
	b	9	10	11

(기준)별 요약 통계

In [22]:
```
frame.sum(level='key2')  # 색인이 key2에 있는 열들의 값을 합친다
```

Out[22]:

state	Ohio		Colorado
color	Green	Red	Green
key2			
1	6	8	10
2	12	14	16

In [23]:
```
frame.sum(level='color', axis=1) # color라는 색인목록만 불러오는데 Green은 두 개 이기때문에 값을 더한다
```

Out[23]:

	color	Green	Red
key1	key2		
a	1	2	1
	2	8	4
b	1	14	7
	2	20	10

실습 예제 **13**번

columns이 region인 행들의 값을 출력하시오

In [ ]:
```

```

데이터프레임 열 조작

In [24]:
```
frame = DataFrame({'a': range(7), 'b': range(7, 0, -1),
                   'c': ['one', 'one', 'one', 'two', 'two', 'two', 'two'
],
                   'd': [0, 1, 2, 0, 1, 2, 3]})
# a는 0부터 7전 까지의 숫자로 값을 채우고
# b는 7부터 1까지의 숫자를 채우는데 내림차순으로 채운다.
# c와 d는 각각 안에 있는 괄호안에 있는 숫자로 채우며
# a b c d 중 하나라도 7개의 값을 가지지 못하면 에러가 뜬다
frame
```

Out[24]:

	a	b	c	d
0	0	7	one	0
1	1	6	one	1
2	2	5	one	2
3	3	4	two	0
4	4	3	two	1
5	5	2	two	2
6	6	1	two	3

실습 예제 **14**번

prat4 라는 데이터프레임을 만들되 columns이 a인 것은 0부터 7까지 나오게하고, b는 5부터 -2까지 나오게 한다. c는 naver(3), goolge(5)로 하며 d는 0~2, 0~4까지로 만든다. (이후 예제는 prat4를 사용)

In [ ]:
```

```

In [25]:
```
# 컬럼을 새로운 색인의 중심으로 재설정할 수 있다
# 아래는 컬럼 'c' 'd'를 가지고 재설정하는 예이다
frame2 = frame.set_index(['c', 'd'])
frame2
```

Out[25]:

		a	b
c	**d**		
one	**0**	0	7
	1	1	6
	2	2	5
two	**0**	3	4
	1	4	3
	2	5	2
	3	6	1

In [26]:
```
# 인덱스에 사용된 값을 그대로 남겨둘 수도 있다
frame.set_index(['c', 'd'], drop=False) # drop은 중심으로 설정한 컬럼을 제거할지의 여부를 정하는 명령어이다
```

Out[26]:

		a	b	c	d
c	**d**				
one	**0**	0	7	one	0
	1	1	6	one	1
	2	2	5	one	2
two	**0**	3	4	two	0
	1	4	3	two	1
	2	5	2	two	2
	3	6	1	two	3

실습 예제 **15**번

columns b와 c를 새로운 색인의 중심으로 설정해 보아라. 인덱스에 사용된 값은 그대로 남겨두어라

In [ ]:
```

```

Data loading, storage, and file formats

In [1]:
```
from __future__ import division
from numpy.random import randn
import numpy as np
import os
import sys
np.random.seed(12345)
from pandas import Series, DataFrame
import pandas as pd
np.set_printoptions(precision=4)
```

텍스트 형식 데이터 읽기 및 쓰기

In [2]:
```
df = pd.read_csv('data/ex1.csv') # csv파일 불러오기
df
```

Out[2]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [3]:
```
pd.read_table('data/ex1.csv', sep=',')
# read_table에서 디폴트 구분자는 탭을 사용한다
```

Out[3]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [4]:
```
# 자동으로 헤더를 만든다
pd.read_csv('data/ex2.csv', header=None)
```

Out[4]:

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [5]:
```
# 헤더 이름을 지정한다
pd.read_csv('data/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

Out[5]:

	a	b	c	d	message
0	1	2	3	4	hello

1	5	6	7	8	world
2	9	10	11	12	foo

실습 예제 **1**번

ex2.csv파일을 읽어와 헤더 이름을 x, y, z ,b, naju로 하여라

In [ ]:

```
```

In [6]:

```
# 특정 컬럼을 인덱스로 사용할 수도 있다
names = ['a', 'b', 'c', 'd', 'message']
pd.read_csv('data/ex2.csv', names=names, index_col='message')
```

Out[6]:

	a	**b**	**c**	**d**
message				
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

실습 예제 **2**번

ex2.csv파일을 읽어와 헤더 이름으로할 list를 만들고 안에 값은 (one, two, three, four, d)로 하고 columns 중 d를 인덱스로 한다

In [ ]:

```
```

In [7]:

```
# 색인 컬럼을 계층으로 줄 수도 있다
parsed = pd.read_csv('data/csv_mindex.csv', index_col=['key1', 'key2'])
parsed
```

Out[7]:

		value1	**value2**
key1	**key2**		
one	**a**	1	2
	b	3	4
	c	5	6
	d	7	8
two	**a**	9	10
	b	11	12
	c	13	14
	d	15	16

실습 예제 **3**번

csv_mindex.csv파일을 읽어와 컬럼중에 value1와 value2를 색인으로 한다. 이름은 prat2로 한다

In [ ]:

In [8]:
```
# 한개 또는 여러개의 스페이스를 구분자로 지정할 수 있다
result = pd.read_table('data/ex3.txt', sep='\s+')
result
# 첫번째 컬럼를 색인으로 자동으로 지정했다 (첫번째 행의 수가 하나 적으므로)
```

Out[8]:

	A	B	C
aaa	-0.264438	-1.026059	-0.619500
bbb	0.927272	0.302904	-0.032399
ccc	-0.264273	-0.386314	-0.217601
ddd	-0.871858	-0.348382	1.100491

In [9]:
```
# 0, 2, 3 번째 행을 삭제하라
pd.read_csv('data/ex4.csv', skiprows=[0, 2, 3])
```

Out[9]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

실습 예제 **4**번

ex4.csv 파일을 불러와 0,2, 3, 4행을 삭제해보아라

In [ ]:

In [10]:
```
result = pd.read_csv('data/ex5.csv')
result
```

Out[10]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

In [11]:
```
pd.isnull(result) ## 괄호 안의 데이터프레임에서 NaN값은 True를 출력하고 아니면 False를 출력한다
```

Out[11]:

	something	a	b	c	d	message
0	False	False	False	False	False	True
1	False	False	False	True	False	False
2	False	False	False	False	False	False

In [12]:
```
# 컬럼별로 결측치(NA 문자들)을 따로 지정할 수 있다
sentinels = {'message': ['foo', 'NA'], 'something': ['two']} ## message 컬럼
```

```
에 있는 foo를 NaN로 바꾸며 someghing에 two도 NaN으로 바꾼다
pd.read_csv('data/ex5.csv', na_values=sentinels)
```

Out[12]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	NaN	5	6	NaN	8	world
2	three	9	10	11.0	12	NaN

실습 예제 **5**번

message 컬럼에 있는 world와 a열에 있는 5를 NaN으로 만들어 보아라

In [ ]:

```
```

텍스트파일의 일부분 읽기

In [13]:

```
result = pd.read_csv('data/ex6.csv')
result.head(10)
```

Out[13]:

	one	two	three	four	key
0	0.467976	-0.038649	-0.295344	-1.824726	L
1	-0.358893	1.404453	0.704965	-0.200638	B
2	-0.501840	0.659254	-0.421691	-0.057688	G
3	0.204886	1.074134	1.388361	-0.982404	R
4	0.354628	-0.133116	0.283763	-0.837063	Q
5	1.817480	0.742273	0.419395	-2.251035	Q
6	-0.776764	0.935518	-0.332872	-1.875641	U
7	-0.913135	1.530624	-0.572657	0.477252	K
8	0.358480	-0.497572	-0.367016	0.507702	S
9	-1.740877	-1.160417	-1.637830	2.172201	G

In [14]:

```
pd.read_csv('data/ex6.csv', nrows=5) #5개의 색인만 가져오기
```

Out[14]:

	one	two	three	four	key
0	0.467976	-0.038649	-0.295344	-1.824726	L
1	-0.358893	1.404453	0.704965	-0.200638	B
2	-0.501840	0.659254	-0.421691	-0.057688	G
3	0.204886	1.074134	1.388361	-0.982404	R
4	0.354628	-0.133116	0.283763	-0.837063	Q

In [15]:

```
# 파일을 여러 조각으로 나눈 후에 각 조각으로부터 데이터를 읽을 수 있다
chunker = pd.read_csv('data/ex6.csv', chunksize=1000)
chunker
```

Out[15]:
```
<pandas.io.parsers.TextFileReader at 0x1361b27aa20>
```

In [16]:
```
chunker = pd.read_csv('data/ex6.csv', chunksize=1000)

tot = Series([])
for piece in chunker:
    tot = tot.add(piece['key'].value_counts(), fill_value=0)

#tot = tot.sort_values(ascending=False)
```

In [17]:
```
tot[:10]
```

Out[17]:
```
0    151.0
1    146.0
2    152.0
3    162.0
4    171.0
5    157.0
6    166.0
7    164.0
8    162.0
9    150.0
dtype: float64
```

텍스트 형식으로 데이터 쓰기

In [18]:
```
data = pd.read_csv('data/ex5.csv')
data
```

Out[18]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

In [19]:
```
data.to_csv('data/out.csv') # 데이터 파일을 out.csv파일로 출력해 보아라
```

실습 예제 **6**번

data 파일을 prat3.csv파일로 출력해 보아라

In [ ]:
```

```

In [20]:
```
data.to_csv(sys.stdout, sep='|')  #내보내지않고 여기에 출력하기하기 위해 sys_stdout을 사용했고 구분 기호로 |를 사용했다
```

```
|something|a|b|c|d|message
0|one|1|2|3.0|4|
1|two|5|6||8|world
2|three|9|10|11.0|12|foo
```

In [21]:
```
data.to_csv(sys.stdout, na_rep='NULL') # 빈값에 NULL값을 넣었다
```

```
,something,a,b,c,d,message
```

```
0,one,1,2,3.0,4,NULL
1,two,5,6,NULL,8,world
2,three,9,10,11.0,12,foo
```

In [22]:
```
data.to_csv(sys.stdout, index=False, header=False) # 컬럼과 색인을 모두 표기하지 않고 출력하기
```

```
one,1,2,3.0,4,
two,5,6,,8,world
three,9,10,11.0,12,foo
```

In [23]:
```
data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c']) #색인은 출력하지않고 컬럼에 a b c를 넣고 출력한다
```

```
a,b,c
1,2,3.0
5,6,
9,10,11.0
```

In [24]:
```
dates = pd.date_range('1/1/2000', periods=7) # 2000/1/1부터 6일 뒤에 날짜까지 리스트를 만든다
ts = Series(np.arange(7)*100, index=dates) # 이것을 시리즈로 만들고 색인은 dates에 있는 날짜이며 값은 0~6까지의 값에 곱하기 100을 한것이다
ts.to_csv('data/tseries.csv')
ts
```

Out[24]:
```
2000-01-01      0
2000-01-02    100
2000-01-03    200
2000-01-04    300
2000-01-05    400
2000-01-06    500
2000-01-07    600
Freq: D, dtype: int32
```

In [25]:
```
Series.from_csv('data/tseries.csv')  # 다시 csv파일을 시리즈로 불러오기
```

Out[25]:
```
2000-01-01      0
2000-01-02    100
2000-01-03    200
2000-01-04    300
2000-01-05    400
2000-01-06    500
2000-01-07    600
dtype: int64
```

실습 예제 **7**번

data2라는 변수의 이름을 가졌고 2017/08/11 부터 6일뒤까지의 날짜를 가진 리스트를 만들어보시오 또한 prat3의 이름을 가졌고 리스트의 값을 색인으로 하는 시리즈를 만들 되 시리즈안의 값들은 0부터 순서대로 나타나게 하시오

In [ ]:
```

```

특정 형식의 수동작업

In [26]:
```
import csv
f = open('data/ex7.csv') #f라는 변수에 ex7.csv 파일이 불러온다
print(f)  #이것을 출력하면 read와 다르게 작업할 수 있는 형식(r, w)이나 언어지원형식이 나온다
reader = csv.reader(f)
reader
```

```
<_io.TextIOWrapper name='data/ex7.csv' mode='r' encoding='cp949'>
```

Out[26]:
```
<_csv.reader at 0x1361b6e3660>
```

In [27]:
```
for line in reader: #reader안에 있는 값을 line에 넣어 그것을 출력한다
    print(line)
```

```
['a', 'b', 'c']
['1', '2', '3']
['1', '2', '3', '4']
```

In [28]:
```
lines = list(csv.reader(open('data/ex7.csv'))) #csv파일을 읽어와 list로 만든다
print(lines)
header, values = lines[0], lines[1:] # header에는 lines의 첫번째 리스트만, values에는 두번째부터 끝까지 리스트를 넣는다.
data_dict = {h: v for h, v in zip(header, zip(*values))}
data_dict
```

```
[['a', 'b', 'c'], ['1', '2', '3'], ['1', '2', '3', '4']]
```

Out[28]:
```
{'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

실습 예제 **8**번

csv파일을 읽어와 list로 만들어 prat4라는 이름으로 저장하고 prheader, prvalues에 각각 두번째리스트까지, 마지막 리스트까지 값을 넣으시오 그 후 두 리스트 모두 출력해보시오

In [ ]:
```
```

In [29]:
```
# JSON 형식
# 키: 밸류 형식을 가지며 키는 반드시 문자열이어야 한다
# JSON 데이터를 파이선에서 읽으려면 json.loads를 사용하면 된다

obj = """
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 25, "pet": "Zuko"},
              {"name": "Katie", "age": 33, "pet": "Cisco"}]
}
"""
```

In [30]:
```
import json
result = json.loads(obj)
result
type(result)
```

Out[30]:
```
dict
```

In [31]:
```
# 파이선 객체를 JSON 형식으로 바꾸려면 json.dumps를 사용하면 된다

asjson = json.dumps(result)
```

```
asjson
```

Out[31]:
```
'{"name": "Wes", "places_lived": ["United States", "Spain", "Germany"],
"pet": null, "siblings": [{"name": "Scott", "age": 25, "pet": "Zuko"}, {
"name": "Katie", "age": 33, "pet": "Cisco"}]}'
```

In [32]:
```
# 사전형식 데이터를 데이터프레임으로 바꾸고 일부 값만 추출할 수 있다
siblings = DataFrame(result['siblings'], columns=['name', 'age'])
siblings
```

Out[32]:

	name	age
0	Scott	25
1	Katie	33

Database와의 상호작용

In [33]:
```
import sqlite3
                    #query라는 데이터베이스 테이블을 만든다.
query = """
CREATE TABLE test
(a VARCHAR(20), b VARCHAR(20),
 c REAL,        d INTEGER
);"""

con = sqlite3.connect(':memory:') ## memory라는 db에 접속
con.execute(query) ## 실행할 query문을 괄호안에 넣는다
con.commit() #변경내용을 db에 반영
```

In [34]:
```
data = [('Atlanta', 'Georgia', 1.25, 6),
        ('Tallahassee', 'Florida', 2.6, 3),
        ('Sacramento', 'California', 1.7, 5)]
stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"

con.executemany(stmt, data)  # executemany는 execute와 다르게 복수개의 데이터를 넣게해
준다
                                    # stmt라는 변수를 사용하여 data를 db에 넣고 변경내용을 db
에 반영한다
con.commit()
```

In [35]:
```
cursor = con.execute('select * from test')
rows = cursor.fetchall() #조회된 결과에서 모든 데이터를 리스트 형태로 반환한다

rows
```

Out[35]:
```
[('Atlanta', 'Georgia', 1.25, 6),
 ('Tallahassee', 'Florida', 2.6, 3),
 ('Sacramento', 'California', 1.7, 5)]
```

In [36]:
```
cursor.description #cursor에 필드확인하기
```

Out[36]:
```
(('a', None, None, None, None, None, None),
 ('b', None, None, None, None, None, None),
 ('c', None, None, None, None, None, None),
 ('d', None, None, None, None, None, None))
```

In [37]:
```
DataFrame(rows, columns=list(zip(*cursor.description))[0]) # db에있는 값을
```

```
데이터프레임 형식으로 출력
```

Out[37]:

	a	b	c	d
0	Atlanta	Georgia	1.25	6
1	Tallahassee	Florida	2.60	3
2	Sacramento	California	1.70	5

In [38]:
```
import pandas.io.sql as sql        #pandas 모듈에서 sql명령어를 사용하여 보다 쉽게 데이터베이스 테이블 출력
sql.read_sql('select * from test', con)
```

Out[38]:

	a	b	c	d
0	Atlanta	Georgia	1.25	6
1	Tallahassee	Florida	2.60	3
2	Sacramento	California	1.70	5

실습 예제 **9**번

위에서 만든 con에 ('korea', 'naju', 2017, 8) 라는 데이터를 execute를 활용하여 넣고 리스트형식으로 출력해보시오.

In [ ]:
```

```

Data Wrangling: Clean, Transform, Merge, Reshape

In [1]:
```
from __future__ import division
from numpy.random import randn
import numpy as np
import os
import matplotlib.pyplot as plt
np.random.seed(12345)
plt.rc('figure', figsize=(10, 6))
from pandas import Series, DataFrame
import pandas
import pandas as pd
np.set_printoptions(precision=4, threshold=500)
pd.options.display.max_rows = 100
```

In [2]:
```
%matplotlib inline
```

데이터셋 결합 및 병합

DB-스타일 데이터프레임 병합

In [3]:
```
df1 = DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                 'data1': range(7)})
df2 = DataFrame({'key': ['a', 'b', 'd'],
                 'data2': range(3)})
df1
```

Out[3]:

	data1	key
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

In [4]:
```
df2
```

Out[4]:

	data2	key
0	0	a
1	1	b
2	2	d

In [5]:
```
pd.merge(df1, df2)
```

```
# 디폴트로 겹치는 컬럼이름을 기준으로 한다
```

Out[5]:

	data1	key	data2
0	0	b	1
1	1	b	1
2	6	b	1
3	2	a	0
4	4	a	0
5	5	a	0

In [6]:
```
pd.merge(df1, df2, on='key')
# 명시적으로 지정하는 것이 안전하다
```

Out[6]:

	data1	key	data2
0	0	b	1
1	1	b	1
2	6	b	1
3	2	a	0
4	4	a	0
5	5	a	0

In [7]:
```
df3 = DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                 'data1': range(7)})
df4 = DataFrame({'rkey': ['a', 'b', 'd'],
                 'data2': range(3)})
pd.merge(df3, df4, left_on='lkey', right_on='rkey') #merge함수는 중복된 값을 기준으로 해서 병합한다
```

Out[7]:

	data1	lkey	data2	rkey
0	0	b	1	b
1	1	b	1	b
2	6	b	1	b
3	2	a	0	a
4	4	a	0	a
5	5	a	0	a

실습 예제 **1**

컬럼명이 나이, 이름 두 가지로 구성된 데이터프레임 df_ex1, df_ex2를 생성하고, 나이를 기준으로 merge하여 출력하여라.(단, 한 프레임 당 행의 수는 6개 이상)

In [ ]:
```

```

In [8]:
```
data = Series([1., -999., 2., -999., -1000., 3.])
```

```
data
```

Out[8]:
```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

In [9]:
```
#치환하는 함수 replace
data.replace(-999, np.nan)
```

Out[9]:
```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

In [10]:
```
data.replace([-999, -1000], np.nan)
```

Out[10]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

In [11]:
```
data.replace([-999, -1000], [np.nan, 0])
```

Out[11]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

In [12]:
```
data.replace({-999: np.nan, -1000: 0})
```

Out[12]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

실습예제 **2**

0부터 10까지의 Series 자료구조를 만든 후 5 이상의 수를 Nan으로 replace하여라.

In [ ]:
```

```

이상치 필터링 및 검출

In [13]:
```
np.random.seed(12)
data = DataFrame(np.random.randn(1000, 4))
data.describe()
```

Out[13]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.019826	-0.020608	-0.036681	-0.032079
std	0.994230	0.998738	0.984190	0.994405
min	-3.147417	-4.011049	-3.015915	-3.710679
25%	-0.664809	-0.719489	-0.684463	-0.710911
50%	0.003597	-0.028044	-0.044665	-0.035832
75%	0.736373	0.682833	0.620109	0.612202
max	3.166557	2.978985	3.529275	3.344649

In [14]:
```
col = data[3]
col[np.abs(col) > 3] #절대값을 구하는 함수 abs()
```

Out[14]:
```
137   -3.710679
149   -3.155014
213    3.041318
445    3.344649
Name: 3, dtype: float64
```

In [15]:
```
data[(np.abs(data) > 3).any(1)] #데이터프레임의 행의 1개 이상의 원소가 절대값을 씌웠을 때 3보다 큰 값이 포함된 행이 있으면 그 행을 출력.
```

Out[15]:

	0	1	2	3
12	-3.147417	0.535136	0.232490	0.867612
27	3.041686	-0.626081	1.505901	-0.587336
60	0.224547	-1.163467	-3.015915	0.593969
124	3.166557	1.383956	-0.077316	-0.911826
137	-1.812846	0.916503	-0.888640	-3.710679
149	1.214205	-0.862325	-0.553625	-3.155014
213	-0.347810	1.281499	-0.217167	3.041318
263	-1.524350	-0.539390	3.087539	-0.370562
445	-0.284077	0.282750	0.096077	3.344649
591	1.303257	-1.362288	-3.015906	-0.747110
761	-0.173639	0.700492	3.529275	-0.229179
774	2.011489	-4.011049	-1.925463	-0.693331

In [16]:
```
data[np.abs(data) > 3] = np.sign(data) * 3 #절대값을 취한 것이 3보다 큰 값을 가진 데이터프레임 data에 sign함수를 취함
data.describe()
```

Out[16]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.019765	-0.019597	-0.037266	-0.031599
std	0.993130	0.995200	0.982040	0.990305
min	-3.000000	-3.000000	-3.000000	-3.000000
25%	-0.664809	-0.719489	-0.684463	-0.710911
50%	0.003597	-0.028044	-0.044665	-0.035832
75%	0.736373	0.682833	0.620109	0.612202
max	3.000000	2.978985	3.000000	3.000000

실습예제 **3**

위의 방법을 응용해 100행 5열의 난수로 구성된 데이터프레임을 생성하고 2번 열의 데이터가 음수인 것만 출력하여라.

In [ ]:

```
```

순열 및 랜덤 샘플링

In [17]:

```
df = DataFrame(np.arange(5 * 4).reshape(5, 4))
df
```

Out[17]:

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

In [18]:

```
len(df)
```

Out[18]:

```
5
```

In [19]:

```
# 행의 위치를 랜덤하게 바꾸려고 한다
sampler = np.random.permutation(5)
sampler
```

Out[19]:

```
array([1, 2, 4, 0, 3])
```

In [20]:

```
# 행을 선택할 때 take를 사용한다
df.take(sampler)
```

Out[20]:

	0	1	2	3
1	4	5	6	7
2	8	9	10	11

4	16	17	18	19
0	0	1	2	3
3	12	13	14	15

In [21]:
```
# 임의의 3개의 행만 추출하는 방법
df.take(np.random.permutation(len(df))[:3])
```

Out[21]:

	0	**1**	**2**	**3**
0	0	1	2	3
4	16	17	18	19
1	4	5	6	7

In [22]:
```
# 15개의 정수 난수를 만들고 이 위치에 해당하는 데이터를 샘플링하는 방법
# bag에서 임의의 갯수를 추출하는 방법
bag = np.array([5, 7, -1, 6, 4])
sampler = np.random.randint(0, len(bag), size=15)
```

In [23]:
```
sampler
```

Out[23]:
```
array([0, 0, 2, 3, 1, 2, 3, 4, 0, 0, 1, 4, 3, 1, 4])
```

In [24]:
```
draws = bag.take(sampler)
draws
```

Out[24]:
```
array([ 5,  5, -1,  6,  7, -1,  6,  4,  5,  5,  7,  4,  6,  7,  4])
```

지표**(key)** 계산 및 쓰레기**(**더미**)**값

In [25]:
```
# 더미 변수를 만드는 방법
df = DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                'data1': range(6)})
df
```

Out[25]:

	data1	**key**
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	b

In [26]:
```
pd.get_dummies(df['key']) # 더미값을 얻어오는 함수 get_dummies
```

Out[26]:

	a	**b**	**c**
0	0	1	0
1	0	1	0

2	1	0	0
3	0	0	1
4	1	0	0
5	0	1	0

In [27]:
```
# 두개의 데이터프레임을 합치려면 join을 사용한다

dummies = pd.get_dummies(df['key'], prefix='key')
df_with_dummy = dummies.join(df['data1'])
df_with_dummy
```

Out[27]:

	key_a	key_b	key_c	data1
0	0	1	0	0
1	0	1	0	1
2	1	0	0	2
3	0	0	1	3
4	1	0	0	4
5	0	1	0	5

실습예제 **4**

생성된 위의 df_with_dummy 데이터프레임의 행의 위치를 랜덤하게 바꿔 출력하여라.

In [ ]:
```

```

In [28]:
```
#개봉년도, 영화제목, 장르의 정보를 가진 20세기 영화 데이터
# 세개의 컬럼만 읽는다
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('data/movies.dat', sep='::', header=None, names=m
names)
movies[:10]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\__main__.py:4: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine do
es not support regex separators (separators > 1 char and different from
'\s+' are interpreted as regex); you can avoid this warning by specifyin
g engine='python'.
```

Out[28]:

	movie_id	title	genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance

7	8	Tom and Huck (1995)	Adventure|Children's
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller

In [29]:
```
genre_iter = (set(x.split('|')) for x in movies.genres)
print(genre_iter)
# 장르에서 유일한 값만 찾는다 set.union 사용
genres = sorted(set.union(*genre_iter))
print(genres)
```

```
<generator object <genexpr> at 0x0000021B76B85DB0>
['Action', 'Adventure', 'Animation', "Children's", 'Comedy', 'Crime', 'D
ocumentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Myst
ery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']
```

In [30]:
```
dummies = DataFrame(np.zeros((len(movies), len(genres))), columns=genres
)
# 0으로 초기화 한 데이터프레임
```

In [31]:
```
# 해당 장르가 있는 부분만 1로 표시한다
for i, gen in enumerate(movies.genres):
    dummies.ix[i, gen.split('|')] = 1
dummies[:10]
```

Out[31]:

	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantas
0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
5	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
7	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
8	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [32]:
```
movies_windic = movies.join(dummies.add_prefix('Genre_'))
#더미값으로 해당하는 장르에 0과 1로 표현
movies_windic.ix[0:10]
```

Out[32]:

	movie_id	title	genres	Genre_Action	Genre_Adventure	Ge
0	1	Toy Story (1995)	Animation|Children's|Comedy	0.0	0.0	1.0
1	2	Jumanji (1995)	Adventure|Children's|Fantasy	0.0	1.0	0.0

2	3	Grumpier Old Men (1995)	Comedy|Romance	0.0	0.0	0.0
3	4	Waiting to Exhale (1995)	Comedy|Drama	0.0	0.0	0.0
4	5	Father of the Bride Part II (1995)	Comedy	0.0	0.0	0.0
5	6	Heat (1995)	Action|Crime|Thriller	1.0	0.0	0.0
6	7	Sabrina (1995)	Comedy|Romance	0.0	0.0	0.0
7	8	Tom and Huck (1995)	Adventure|Children's	0.0	1.0	0.0
8	9	Sudden Death (1995)	Action	1.0	0.0	0.0
9	10	GoldenEye (1995)	Action|Adventure|Thriller	1.0	1.0	0.0
10	11	American President, The (1995)	Comedy|Drama|Romance	0.0	0.0	0.0

11 rows × 21 columns

In [33]:
```
np.random.seed(12345)
```

In [34]:
```
values = np.random.rand(10)
values
```

Out[34]:
```
array([ 0.9296,  0.3164,  0.1839,  0.2046,  0.5677,  0.5955,  0.9645,
        0.6532,  0.7489,  0.6536])
```

In [35]:
```
# 데이터가 어느 범주에 속하는지 간단히 찾을 수 있다
bins = [0, 0.2, 0.4, 0.6, 0.8, 1]
pd.get_dummies(pd.cut(values, bins))
```

Out[35]:

	(0, 0.2]	(0.2, 0.4]	(0.4, 0.6]	(0.6, 0.8]	(0.8, 1]
0	0	0	0	0	1
1	0	1	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0

5	0	0	1	0	0
6	0	0	0	0	1
7	0	0	0	1	0
8	0	0	0	1	0
9	0	0	0	1	0

실습예제 **5**

0~100 사이의 난수 10개를 생성하고 데이터가 10 단위의 범주 중 어느 범주에 속하는지를 구하여라

In [ ]:

```
```

문자열 조작

문자열 메소드

In [36]:

```
# 파이선은 문자 열을 다루는데 매우 편리하다
# 문자열 구분하기

val = 'a,b,  guido'
val.split(',')
```

Out[36]:

```
['a', 'b', '  guido']
```

In [37]:

```
# 공백 부분을 없애려면 strip을 사용한다
pieces = [x.strip() for x in val.split(',')]
pieces
```

Out[37]:

```
['a', 'b', 'guido']
```

In [38]:

```
first, second, third = pieces
print(first)
first + '::' + second + '::' + third
```

```
a
```

Out[38]:

```
'a::b::guido'
```

In [39]:

```
# 더 효과적인 방법
'::'.join(pieces)
```

Out[39]:

```
'a::b::guido'
```

In [40]:

```
# 단어가 포함되어 있는지를 알려준다
'guido' in val
```

Out[40]:

```
True
```

In [41]:

```
# 앞에서부터의 위치를 찾아준다
val.index(',')
```

Out[41]:

```
1
```

In [42]:
```
val.index('b')
```

Out[42]:
```
2
```

In [43]:
```
val.index('guido')
```

Out[43]:
```
6
```

In [44]:
```
val.index('NO')
# 없는 단어는 에러가 난다
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call la
st)
<ipython-input-44-87e649c803ba> in <module>()
----> 1 val.index('NO')
      2 # 없는 단어는 에러가 난다

ValueError: substring not found
```

In [45]:
```
val.find('NO')
# 단어가 없으면 오류가 아니라 '-1' 를 리턴한다
```

Out[45]:
```
-1
```

In [46]:
```
# 발생 횟수를 알려준다
val.count(',')
```

Out[46]:
```
2
```

In [47]:
```
val.replace(',', '::')
```

Out[47]:
```
'a::b::  guido'
```

In [48]:
```
val.replace(',', '')
```

Out[48]:
```
'ab  guido'
```

실습예제 **6**

위의 val문자열의 , 개수와 문자열의 길이를 합한 값을 출력하시오.

In [ ]:
```

```

실습예제 **7**

위의 val문자열의 ' '(공백)을 찾고 몇 번째 index에 위치해 있는지 나타내시오.

In [ ]:
```

```

정규식

In [49]:
```
# 정규표현식, 텍스트에서 문자열을 찾는 도구
```

```
# regex
# 패턴 매칭, 치환, 분리 기능 등을 제공한다
# 문자열을 분리하는 예로 하나 이상의 스페이스를 의미하는 '\s+'를 사용한다
import re
text = "foo    bar\t baz  \tqux"
re.split('\s+', text)
```

Out[49]:
```
['foo', 'bar', 'baz', 'qux']
```

In [50]:
```
# 정규표현식을 컴파일하고 이 객체를 이용하는 방법도 있다
# 반복적으로 사용될 때 편리하고 속도도 빠르다
regex = re.compile('\s+')
regex.split(text)
```

Out[50]:
```
['foo', 'bar', 'baz', 'qux']
```

In [51]:
```
regex.findall(text)
```

Out[51]:
```
['    ', '\t ', '  \t']
```

In [52]:
```
text = """Dave dave@google.com
Steve steve@gmail.com
Rob rob@gmail.com
Ryan ryan@yahoo.com
"""
pattern = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}' #해당 부분에 맞게 정규식 변수 pattern을 지정

# re.IGNORECASE는 대소문자 구분을 없애준다.
regex = re.compile(pattern, flags=re.IGNORECASE)
```

In [53]:
```
# 문자열에서 이 표현식과 일치하는 모든 부분을 찾는다
regex.findall(text)
```

Out[53]:
```
['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

실습예제 **8**

위의 text에서 이메일이 아닌 단어 단위를 찾아 출력하여라.

In [ ]:
```
```

In [54]:
```
# sesrch는 만족하는 첫번째 항목만 찾아준다
m = regex.search(text)
print(m)
```

```
<_sre.SRE_Match object; span=(5, 20), match='dave@google.com'>
```

In [55]:
```
text[m.start():m.end()] #첫번째 항목의 시작지점부터 끝까지를 보여준다
```

Out[55]:
```
'dave@google.com'
```

In [56]:
```
print(regex.match(text))
```

```
None
```

In [57]:
```
# 해당하는 패턴을 주어진 문자열로 치환한다
```

```
print(regex.sub('REDACTED', text))
```

```
Dave REDACTED
Steve REDACTED
Rob REDACTED
Ryan REDACTED
```

실습예제 **9**

예제 8에서 찾은 단어들을 'python'으로 치환하여라.

In [ ]:
```

```

In [58]:
```
# 패턴을 나누려면, 나눌 각 패턴을 ()로 묶는다
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
regex = re.compile(pattern, flags=re.IGNORECASE)
```

In [59]:
```
m = regex.match('wesm@bright.net')
print(m)
m.groups() #m을 정규식 패턴에 맞게 나누어 그룹화하는 함수
```

```
<_sre.SRE_Match object; span=(0, 15), match='wesm@bright.net'>
```

Out[59]:
```
('wesm', 'bright', 'net')
```

In [60]:
```
regex.findall(text)
```

Out[60]:
```
[('dave', 'google', 'com'),
 ('steve', 'gmail', 'com'),
 ('rob', 'gmail', 'com'),
 ('ryan', 'yahoo', 'com')]
```

In [61]:
```
print(regex.sub(r'Username: \1, Domain: \2, Suffix: \3', text)) #sub함수를 이용해 정규식의 각 부분에 제목(subject)을 명명할 수 있다.
```

```
Dave Username: dave, Domain: google, Suffix: com
Steve Username: steve, Domain: gmail, Suffix: com
Rob Username: rob, Domain: gmail, Suffix: com
Ryan Username: ryan, Domain: yahoo, Suffix: com
```

In [62]:
```
# 매치 그룹에 이름을 줄 수 있다
regex = re.compile(r"""
    (?P<username>[A-Z0-9._%+-]+)
    @
    (?P<domain>[A-Z0-9.-]+)
    \.
    (?P<suffix>[A-Z]{2,4})""", flags=re.IGNORECASE|re.VERBOSE)
```

In [63]:
```
m = regex.match('wesm@bright.net')
m.groupdict() #사전형태로 그룹화한다.
```

Out[63]:
```
{'domain': 'bright', 'suffix': 'net', 'username': 'wesm'}
```

pandas의 벡터화된 문자열 함수

```

```

In [64]:
```
data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
        'Rob': 'rob@gmail.com', 'Wes': np.nan}
data = Series(data)
```

In [65]:
```
data.isnull() #값이 비어있는지 확인하는 isnull()함수
```

Out[65]:
```
Dave     False
Rob      False
Steve    False
Wes       True
dtype: bool
```

실습예제**10**

data의 isnull()함수를 호출한 값에 True가 없도록 하는 data2를 만들고 다시 isnull()함수를 호출하여라.

In [ ]:
```
```

In [66]:
```
data.str.contains('gmail') #문자열에 'gmail'을 포함하고있는지 판별한다.
```

Out[66]:
```
Dave     False
Rob       True
Steve     True
Wes        NaN
dtype: object
```

In [67]:
```
pattern
```

Out[67]:
```
'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\\.([A-Z]{2,4})'
```

In [68]:
```
data.str.findall(pattern, flags=re.IGNORECASE)
```

Out[68]:
```
Dave     [(dave, google, com)]
Rob        [(rob, gmail, com)]
Steve    [(steve, gmail, com)]
Wes                        NaN
dtype: object
```

In [69]:
```
matches = data.str.match(pattern, flags=re.IGNORECASE)
matches
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: Futu
reWarning: In future versions of pandas, match will change to always ret
urn a bool indexer.
  if __name__ == '__main__':
```

Out[69]:
```
Dave     (dave, google, com)
Rob        (rob, gmail, com)
Steve    (steve, gmail, com)
Wes                      NaN
dtype: object
```

In [70]:
```
matches.str.get(1) #get(index)로 해당 위치의 내용을 불러옴
```

Out[70]:
```
Dave     google
Rob       gmail
Steve     gmail
Wes         NaN
```

```
dtype: object
```

In [71]:
```
matches.str[0] #str[index]로도 내용을 불러올 수 있음
```

Out[71]:
```
Dave      dave
Rob        rob
Steve    steve
Wes        NaN
dtype: object
```

In [72]:
```
data.str[:5]
```

Out[72]:
```
Dave     dave@
Rob      rob@g
Steve    steve
Wes        NaN
dtype: object
```

Example: 미국 농무부 음식 데이터

#데이터 정보 { "id": 21441, "description": "KENTUCKY FRIED CHICKEN, Fried Chicken, EXTRA CRISPY, Wing, meat and skin with breading", "tags": ["KFC"], "manufacturer": "Kentucky Fried Chicken", "group": "Fast Foods", "portions": [{ "amount": 1, "unit": "wing, with skin", "grams": 68.0 }, ...], "nutrients": [{ "value": 20.8, "units": "g", "description": "Protein", "group": "Composition" }, ...] }

In [73]:
```
import json
db = json.load(open('data/foods-2011-10-03.json'))
len(db)
```

Out[73]:
```
6636
```

In [74]:
```
# 각 항목에는 각 음식에 대한 정보를 담는다
db[0].keys()
```

Out[74]:
```
dict_keys(['id', 'description', 'tags', 'manufacturer', 'group', 'portio
ns', 'nutrients'])
```

In [75]:
```
db[0]['nutrients'][0]
```

Out[75]:
```
{'description': 'Protein',
 'group': 'Composition',
 'units': 'g',
 'value': 25.18}
```

실습예제**11**

위의 첫 번째 항목의 영양소를 표시한 것 처럼, 식품의 양(portions)을 출력하여라.

In [ ]:
```

```

In [76]:
```
nutrients = DataFrame(db[0]['nutrients'])
nutrients[:7]
```

Out[76]:

	description	group	units	value
0	Protein	Composition	g	25.18
1	Total lipid (fat)	Composition	g	29.20

2	Carbohydrate, by difference	Composition	g	3.06
3	Ash	Other	g	3.28
4	Energy	Energy	kcal	376.00
5	Water	Composition	g	39.28
6	Energy	Energy	kJ	1573.00

In [77]:
```
info_keys = ['description', 'group', 'id', 'manufacturer']
info = DataFrame(db, columns=info_keys)
```

In [78]:
```
info[:5]
```

Out[78]:

	description	group	id	manufacturer
0	Cheese, caraway	Dairy and Egg Products	1008	
1	Cheese, cheddar	Dairy and Egg Products	1009	
2	Cheese, edam	Dairy and Egg Products	1018	
3	Cheese, feta	Dairy and Egg Products	1019	
4	Cheese, mozzarella, part skim milk	Dairy and Egg Products	1028	

In [79]:
```
info.head(10)
```

Out[79]:

	description	group	id	manufacturer
0	Cheese, caraway	Dairy and Egg Products	1008	
1	Cheese, cheddar	Dairy and Egg Products	1009	
2	Cheese, edam	Dairy and Egg Products	1018	
3	Cheese, feta	Dairy and Egg Products	1019	
4	Cheese, mozzarella, part skim milk	Dairy and Egg Products	1028	
5	Cheese, mozzarella, part skim milk, low moisture	Dairy and Egg Products	1029	
6	Cheese, romano	Dairy and Egg Products	1038	
7	Cheese, roquefort	Dairy and Egg Products	1039	
8	Cheese spread, pasteurized process, american, ...	Dairy and Egg Products	1048	
9	Cream, fluid, half and half	Dairy and Egg Products	1049	

In [80]:
```
# 음식 그룹의 분포를 찾는다
pd.value_counts(info.group)[:10]
```

Out[80]:
```
Vegetables and Vegetable Products    812
Beef Products                        618
Baked Products                       496
Breakfast Cereals                    403
Legumes and Legume Products          365
Fast Foods                           365
Lamb, Veal, and Game Products        345
Sweets                               341
Fruits and Fruit Juices              328
Pork Products                        328
Name: group, dtype: int64
```

실습예제 **12**

영양소 자료구조의 그룹의 분포 출력하여라.

In [ ]:
```
```

In [81]:
```
'''
영양소 정보를 분석
'''
nutrients = []

for rec in db: #data cleaning을 위해 영양소 리스트의 id를 같은 음식별로 묶는 함수 rec
    fnuts = DataFrame(rec['nutrients'])
    fnuts['id'] = rec['id']
    nutrients.append(fnuts)

nutrients = pd.concat(nutrients, ignore_index=True) #concat함수를 이용해 기존의 nutrients에 id column을 새로 생성하여 붙임.
```

In [82]:
```
nutrients.head(10)
```

Out[82]:

	description	group	units	value	id
0	Protein	Composition	g	25.18	1008
1	Total lipid (fat)	Composition	g	29.20	1008
2	Carbohydrate, by difference	Composition	g	3.06	1008
3	Ash	Other	g	3.28	1008
4	Energy	Energy	kcal	376.00	1008
5	Water	Composition	g	39.28	1008
6	Energy	Energy	kJ	1573.00	1008
7	Fiber, total dietary	Composition	g	0.00	1008
8	Calcium, Ca	Elements	mg	673.00	1008
9	Iron, Fe	Elements	mg	0.64	1008

In [83]:
```
nutrients.duplicated().sum() #중복된 값 더함
```

Out[83]:
```
14179
```

In [84]:
```
nutrients = nutrients.drop_duplicates() #중복된 값을 제거
```

In [85]:
```
col_mapping = {'description' : 'food',
               'group'       : 'fgroup'}
info = info.rename(columns=col_mapping, copy=False) #col_mapping에서 새로 정한 이름으로 info를 rename
info.head(10)
```

Out[85]:

	food	fgroup	id	manufacturer
0	Cheese, caraway	Dairy and Egg Products	1008	
1	Cheese, cheddar	Dairy and Egg Products	1009	
2	Cheese, edam	Dairy and Egg Products	1018	
3	Cheese, feta	Dairy and Egg Products	1019	
4	Cheese, mozzarella, part skim milk	Dairy and Egg Products	1028	
5	Cheese, mozzarella, part skim milk, low moisture	Dairy and Egg Products	1029	
6	Cheese, romano	Dairy and Egg Products	1038	
7	Cheese, roquefort	Dairy and Egg Products	1039	
8	Cheese spread, pasteurized process, american, ...	Dairy and Egg Products	1048	
9	Cream, fluid, half and half	Dairy and Egg Products	1049	

실습예제13

위의 col_mapping의 'id'컬럼을 'food_id'로 변경하여라.

In [ ]:
```
```

In [86]:
```
col_mapping = {'description' : 'nutrient',
               'group' : 'nutgroup'}
nutrients = nutrients.rename(columns=col_mapping, copy=False)
nutrients.head(10)
```

Out[86]:

	nutrient	nutgroup	units	value	id
0	Protein	Composition	g	25.18	1008
1	Total lipid (fat)	Composition	g	29.20	1008
2	Carbohydrate, by difference	Composition	g	3.06	1008

3	Ash	Other	g	3.28	1008
4	Energy	Energy	kcal	376.00	1008
5	Water	Composition	g	39.28	1008
6	Energy	Energy	kJ	1573.00	1008
7	Fiber, total dietary	Composition	g	0.00	1008
8	Calcium, Ca	Elements	mg	673.00	1008
9	Iron, Fe	Elements	mg	0.64	1008

In [87]:
```
ndata = pd.merge(nutrients, info, on='id', how='outer') #id를 키로 해서 외부 조인
```

In [88]:
```
ndata.head(10)
```

Out[88]:

	nutrient	nutgroup	units	value	id	food	fgroup	manufacturer
0	Protein	Composition	g	25.18	1008	Cheese, caraway	Dairy and Egg Products	
1	Total lipid (fat)	Composition	g	29.20	1008	Cheese, caraway	Dairy and Egg Products	
2	Carbohydrate, by difference	Composition	g	3.06	1008	Cheese, caraway	Dairy and Egg Products	
3	Ash	Other	g	3.28	1008	Cheese, caraway	Dairy and Egg Products	
4	Energy	Energy	kcal	376.00	1008	Cheese, caraway	Dairy and Egg Products	
5	Water	Composition	g	39.28	1008	Cheese, caraway	Dairy and Egg Products	
6	Energy	Energy	kJ	1573.00	1008	Cheese, caraway	Dairy and Egg Products	
7	Fiber, total dietary	Composition	g	0.00	1008	Cheese, caraway	Dairy and Egg Products	
8	Calcium, Ca	Elements	mg	673.00	1008	Cheese, caraway	Dairy and Egg Products	
9	Iron, Fe	Elements	mg	0.64	1008	Cheese, caraway	Dairy and Egg Products	

실습예제 **14**

nutrients와 info를 innerjoin한 ndata2를 생성하여라.

In [ ]:
```
```

In [89]:
```
ndata.ix[30000] #x번째 행을 나타내는 ix[x]
```

Out[89]:
```
nutrient                                          Glycine
nutgroup                                        Amino Acids
units                                                 g
value                                              0.04
id                                                 6158
food            Soup, tomato bisque, canned, condensed
fgroup                        Soups, Sauces, and Gravies
manufacturer
Name: 30000, dtype: object
```

In [90]:
```
by_nutrient = ndata.groupby(['nutgroup', 'nutrient']) #groupby()함수를 사용하여 nutgroup에 따른 nutrient로 group화한다.

get_maximum = lambda x: x.xs(x.value.idxmax()) #람다함수는 함수를 지정(직접 만드는 방식),
                                                #이 함수는 value가 가장 높은 수의 인덱스를 찾아줌 -> idxmax()함수
get_minimum = lambda x: x.xs(x.value.idxmin()) # -> 반대로 가장 낮은 수의 인덱스를 찾아줌 -> idxmin() 함수

max_foods = by_nutrient.apply(get_maximum)[['value', 'food']] #apply (lambda 함수) 문법을 이용해 만든 람다 함수를 실행

max_foods.food = max_foods.food.str[:50]
```

In [91]:
```
max_foods.ix['Amino Acids']['food']
```

Out[91]:
```
nutrient
Alanine                             Gelatins, dry powder, unsweetened
Arginine                                 Seeds, sesame flour, low-fat
Aspartic acid                                  Soy protein isolate
Cystine               Seeds, cottonseed flour, low fat (glandless)
Glutamic acid                                  Soy protein isolate
Glycine                             Gelatins, dry powder, unsweetened
Histidine                 Whale, beluga, meat, dried (Alaska Native)
Hydroxyproline    KENTUCKY FRIED CHICKEN, Fried Chicken, ORIGINA...
Isoleucine        Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Leucine           Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Lysine            Seal, bearded (Oogruk), meat, dried (Alaska Na...
Methionine                    Fish, cod, Atlantic, dried and salted
Phenylalanine     Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Proline                             Gelatins, dry powder, unsweetened
Serine            Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Threonine         Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Tryptophan         Sea lion, Steller, meat with fat (Alaska Native)
Tyrosine          Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Valine            Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Name: food, dtype: object
```

In [ ]:

Plotting and Visualization

In [1]:
```
## 모듈이란 함수나 변수 또는 클래스들을 모아 놓은 파일입니다. 밑 코드들은 모듈들을 불러오는 과정입니다.
## 파이썬으로 프로그래밍을 할 때 굉장히 많은 모듈을 사용하기 때문에 밑 코드들은 많이 보시게 될 것 입니다.
## import ex1 => ex1.py 모듈 파일을 불러오는 과정으로 ex1 안에 존재하는 함수를 사용할 수 있게 됩니다.
## from ex1 import sum => ex1.py 모듈파일에서 sum이라는 함수만 사용하는 방식입니다.
from numpy.random import randn ## numpy.random이라는 모듈 안의 randn 함수를 불러오는 코드.
import numpy as np ## numpy 모듈을 불러오며, numpy를 np로 대체
import matplotlib.pyplot as plt ## matplotlib.pyplot ## matplotlib.pylot 모듈을 불러오며 plt로 대체
from pandas import *
```

In [2]:
```
##numpy.random 서브패키지는 numpy의 랜덤넘버 생성 관련 함수를 모아 놓은 것으로 다음과 같은 함수를 제공합니다.
## seed : pseudo random 상태 설정
## shuffle : 조합
## choice : 순열
## random_integers : uniform integer
## rand : uniform
## randn : Gaussina nomal
## 컴퓨터 프로그램에서 무작위와 관련된 알고리즘은 사실 무작위가 아니라 시작 숫자를 정해주면 그 다음에는 정해진 알고리즘에 의해 마치 난수 처럼
## 보이는 수열을 생성 하는 것입니다. 이와 같이 시작 숫자를 seed라 지칭합니다.
## seed는 0보다 크거나 같은 정수의 값을 가질 수 있습니다.간단히 seed의 역할을 밑에서 보여드리겠습니다.
```

In [3]:
```
np.random.seed(0) ## 시드를 0으로 설정해준다.
a = np.random.rand(5) ## rand명령은 0과 1 사이의 난수를 발생시키는 명령어로 인수로 받은 숫자 횟수 만큼 난수를 발생시킵니다.
b = np.random.rand(10) ## 각각 난수를 a,b 변수에 저장합니다.
print(a)## a,b 출력과정
print(b)
```

```
[ 0.5488135   0.71518937  0.60276338  0.54488318  0.4236548 ]
[ 0.64589411  0.43758721  0.891773    0.96366276  0.38344152  0.79172504
  0.52889492  0.56804456  0.92559664  0.07103606]
```

In [4]:
```
np.random.seed(0) ## 시디를 0으로 재설정해줍니다.
a = np.random.rand(5)
b = np.random.rand(10)
print(a)
print(b) ## 다시 난수를 발생시키는데 위와 동일한 값이 나오는 것을 확인 할 수 있습니다. seed 영향이죠.
```

```
[ 0.5488135   0.71518937  0.60276338  0.54488318  0.4236548 ]
[ 0.64589411  0.43758721  0.891773    0.96366276  0.38344152  0.79172504
  0.52889492  0.56804456  0.92559664  0.07103606]
```

In [5]:
```
##np.random.seed(12345) ## seed를 12345로 설정
```

```
##plt.rc('figure', figsize=(10, 6))
```

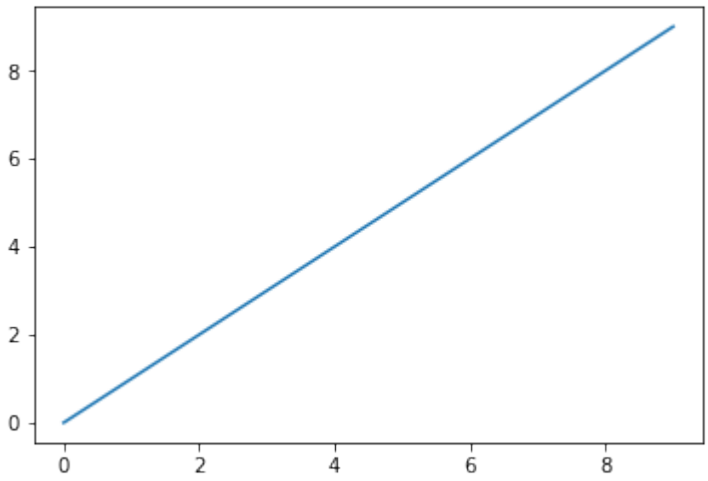
In [6]:
```
import pandas as pd ## pands 모듈을 불러오며 pd로 대체
np.set_printoptions(precision=4)
```

In [7]:
```
%matplotlib inline
```

간단한 **matplotlib API**

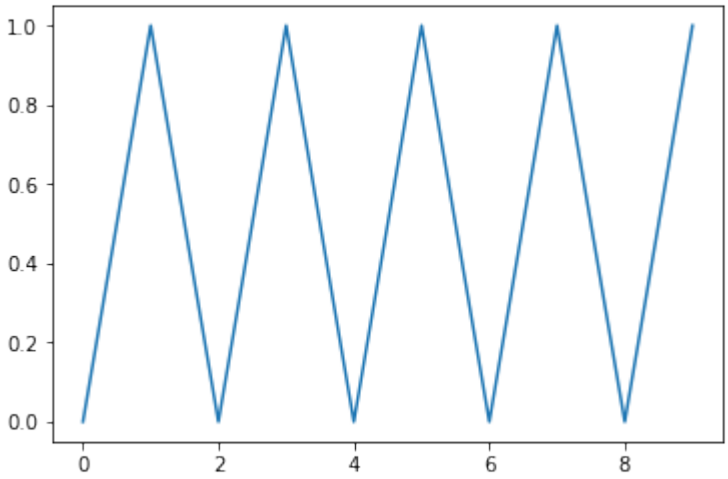
In [8]:
```
plt.plot(np.arange(10)) ## matplotlib.plot(a) 형태입니다. 하나 값만 정해질 경우 a인수는 y 값에 해당되며 x값은 자동적으로 y 갯수 만큼의 정수로
                        ## 정해집니다.
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x24f8132b470>]
```



In [9]:
```
plt.plot(np.arange(10),[0,1,0,1,0,1,0,1,0,1]) ## plot(a,b) a와 b값 모두 정해준 형태입니다. a는 x값이 되며 b는 y값이 된다.
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x24f80e3f6a0>]
```



실습예제 **1**

y=x인 그래프를 그리시오(범위 0~100)

In [ ]:
```

```

수치 및 다중그림

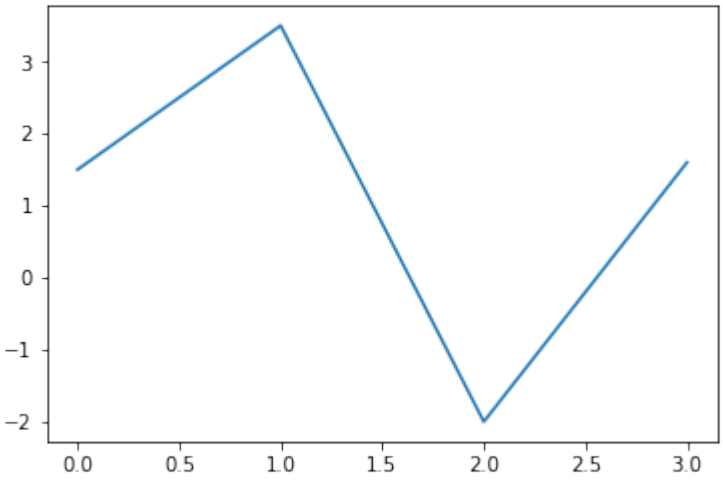
In [10]:
```
# 그래프는 Figure 객체내에 존재한다. 아래는 새로운 Figure 객체를 만든다
#fig = plt.figure()
```

In [11]:
```
# 서브플롯을 만들어 그림을 추가한다
#ax1 = fig.add_subplot(2, 2, 1)
```

In [12]:
```
#ax2 = fig.add_subplot(2, 2, 2)
#ax3 = fig.add_subplot(2, 2, 3)
```

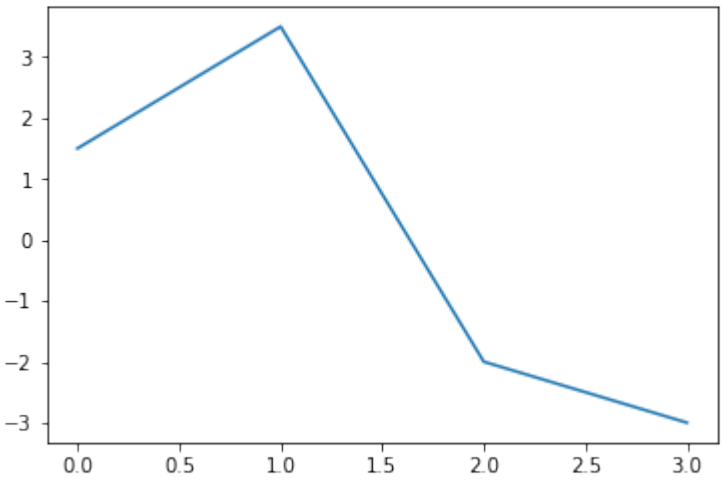
In [13]:
```
plt.plot([1.5,3.5,-2,1.6]) ## plt.plot(a) 형식
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x24f81155cc0>]
```



In [14]:
```
plt.plot([1.5,3.5,-2,-3]) ## plt.plot(a) 형식
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0x24f813af5f8>]
```



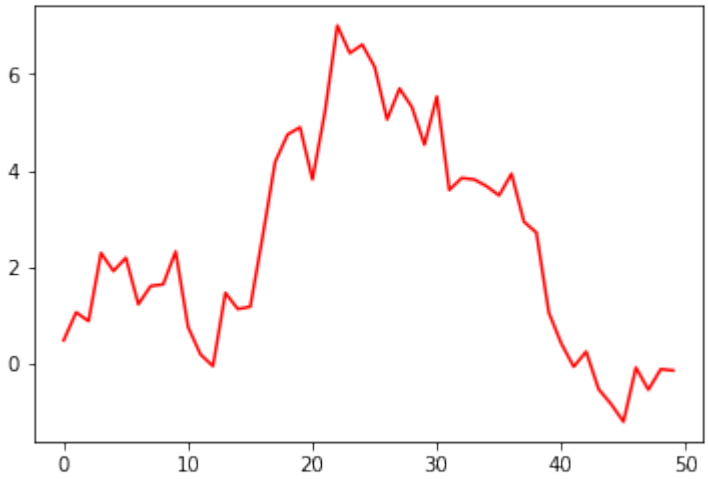
실습예제2

총 7번 꺾이는 꺾은선 그래프를 그리시오.

In [ ]:
```

```

In [15]:
```
plt.plot(randn(50).cumsum(), color = 'r') ##randn().cumsum()은 cumsum()함수는 누적합을 구하는 함수이다.
                                          ##plt.plot(a,color='?') 은 y값을 a로 받고, color를 배정하는 형식이다. 'r' = red , 'b' = blue 등이있다.
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x24f81448518>]
```

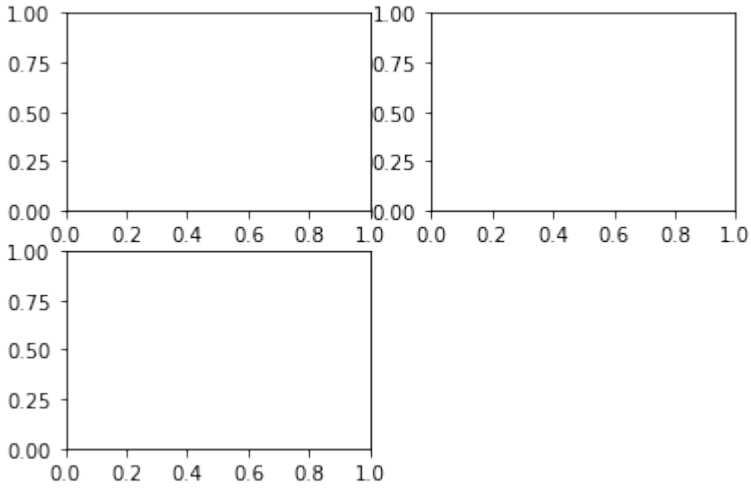


실습예제3

위의 그래프 그리는 방법을 응용하여 x축의 범위를 100으로 하는 그래프를 그리시오. 또 선의 색을 blue로 설정하시오.

In [ ]:
```
```

In [16]:
```
fig = plt.figure() # 그래프는 Figure 객체내에 존재한다. 아래는 새로운 Figure 객체를 만든다
                   # 아래 출력 된 것이 Figure라 보면 됩니다.
ax1 = fig.add_subplot(2, 2, 1) ##fig.add_subplot(a,b,c) 는 Figure 객체내 서브plot를 넣는 것으로 a*b 행렬 에서 c는 위치를 나태낸다.
ax2 = fig.add_subplot(2, 2, 2) ## 2*2 행렬에서 2번째에 위치하는 plot
ax3 = fig.add_subplot(2, 2, 3)
```



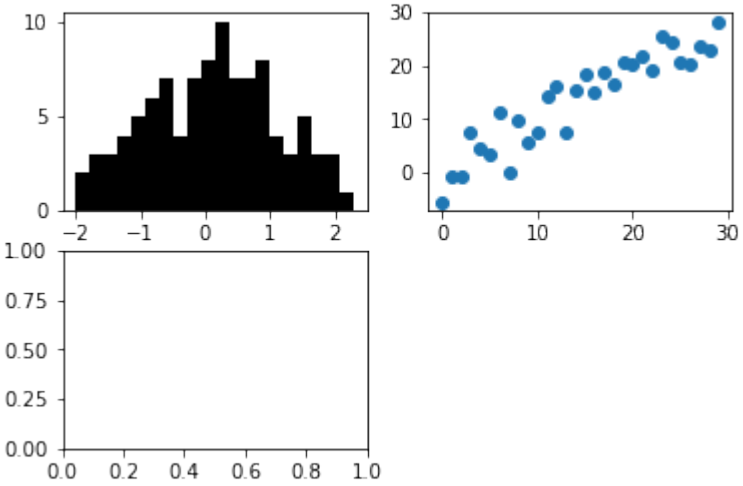
In [17]:
```
ax1.hist(randn(100), bins=20, color='k') ## matplotlib.pyplot에는 여러가지 형태의 plot(그래프) 형식이 제공됩니다.
                                         ## hist : 히스토그램 . bar : 막대그래프 scatter : 산점도 pie: 원그래프 등등.
```

```
                                                                    ## ax1에 히스토그램 형식
이며 bins 는 폭 설정 , color는 색깔 설정입니다.
ax2.scatter(np.arange(30), np.arange(30) + 3 * randn(30)) ## scatter 산점도
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x24f81640668>
```

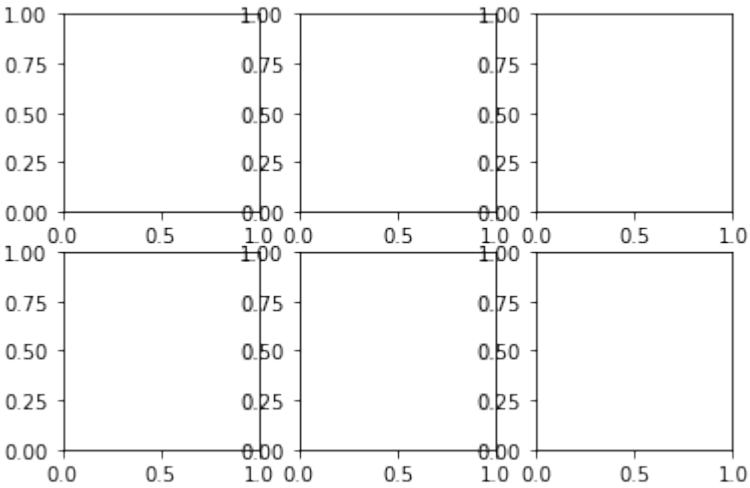
In [18]:
```
fig
```

Out[18]:



In [19]:
```
axes = plt.subplots(2, 3) ## 2*3을 설정해주고 아래의 결과를 볼 수 있다.
axes
```

Out[19]:
```
(<matplotlib.figure.Figure at 0x24f81617da0>,
 array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000024F813D
1CC0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F8194
7128>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F8199
EE48>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x0000024F81A1
0D30>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F81A6
57F0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F81AC
EAC8>]], dtype=object))
```



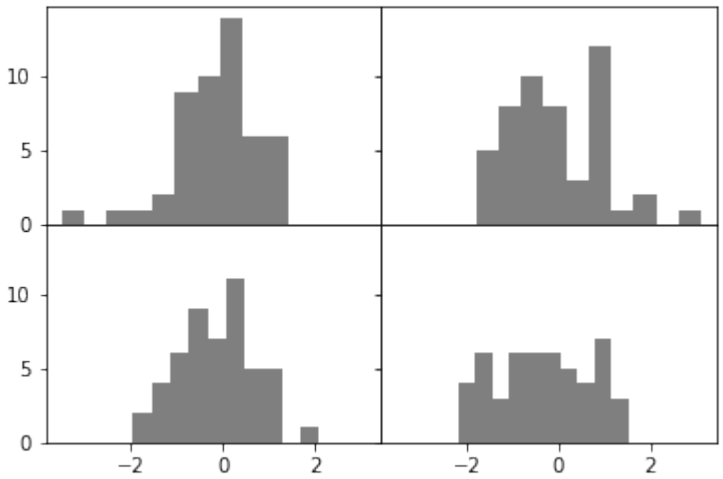
subplot 주변의 공간 고정

In [20]:
```
plt.subplots_adjust(left=None, bottom=None, right=None, top=None,
```

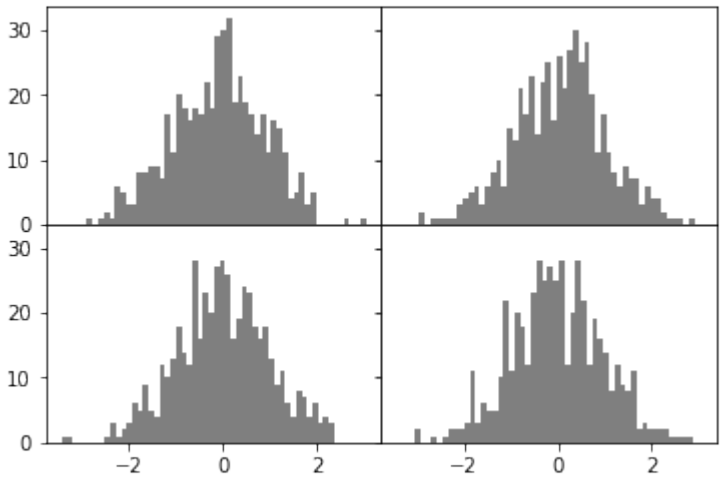
```
                    wspace=None, hspace=None) # subplot 간의 간격을 조정하는 함수 _adjust()
```

```
<matplotlib.figure.Figure at 0x24f81ae4a20>
```

In [21]:
```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True) #각 subplot이 sharex=true x축 공유 sharey=y축을 공유한다.
for i in range(2):
    for j in range(2): # for 구문은 for i in range() 범위 만큼 i를 반복 한다는 구문이다.
        axes[i, j].hist(randn(50), bins=10, color='k', alpha=0.5) ## 이중 for 구문으로 [0,0] , [0,1] , [1,0] , [1,1] 순으로 반복.
plt.subplots_adjust(wspace=0, hspace=0) # 인수 값들을 변경해서 어떤 인수들이 어떤 작용을 하는지 볼 수 있다!
```



In [22]:
```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=0, hspace=0)
```



실습예제 **4**

위의 코드를 참고하여 9개의 랜덤한 히스토그램을 그리시오.

In [ ]:
```

```

색, 점 모양, 선 스타일

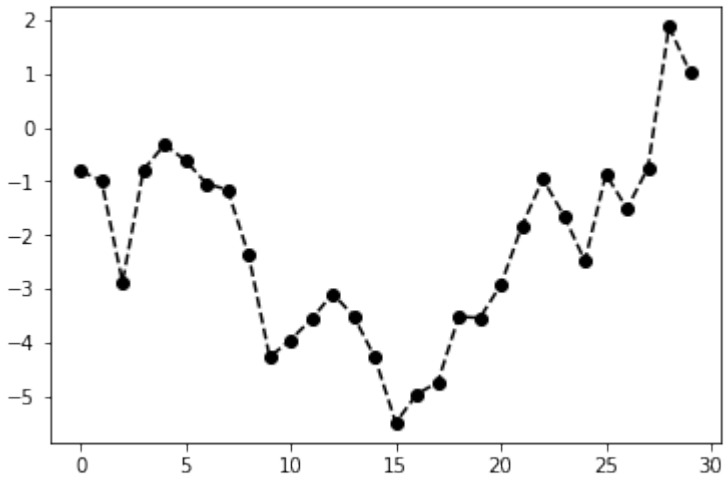
In [23]:
```
plt.figure()
```

Out[23]:
```
<matplotlib.figure.Figure at 0x24f8146beb8>
```

```
<matplotlib.figure.Figure at 0x24f8146beb8>
```

In [24]:
```
plt.plot(randn(30).cumsum(), 'ko--') #cumsum()함수는 행렬의 누적 합을 구함 ex) 행렬이 1 2 3
                                     #                                  4 5 6 일 경우
                                     #cumsum() 후의 값은 1 3 6 10 15 21 이 됨.
                                     #'ko--'는 k색깔의 선의 형태를 o--를 뜻하며 'ko'로 인수를 바꾸어 결과의 변화를 확인해보자
```

Out[24]:
```
[<matplotlib.lines.Line2D at 0x24f833b3278>]
```



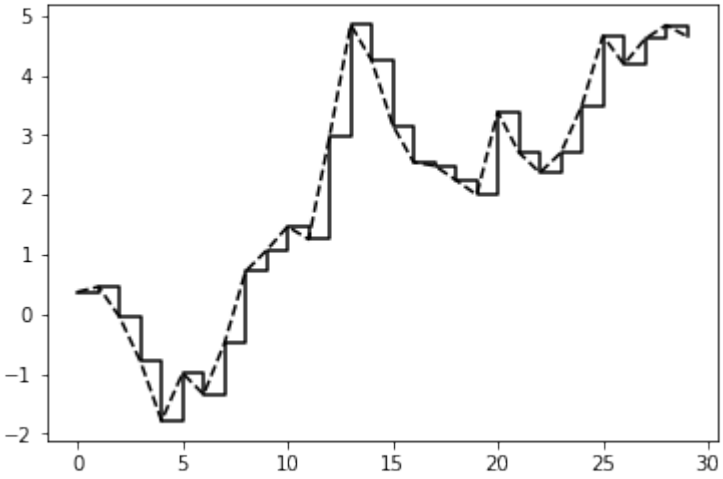
실습예제5

꺾이는 부분에 사각형으로 점을 찍고 점선이며 파란 색인 그래프를 그리시오.

In [ ]:
```
```

In [25]:
```
data = randn(30).cumsum()
plt.plot(data, 'k--', label='Default')
plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post') # drawstyle='steps-post' 계단식 표현 뜻한다
```

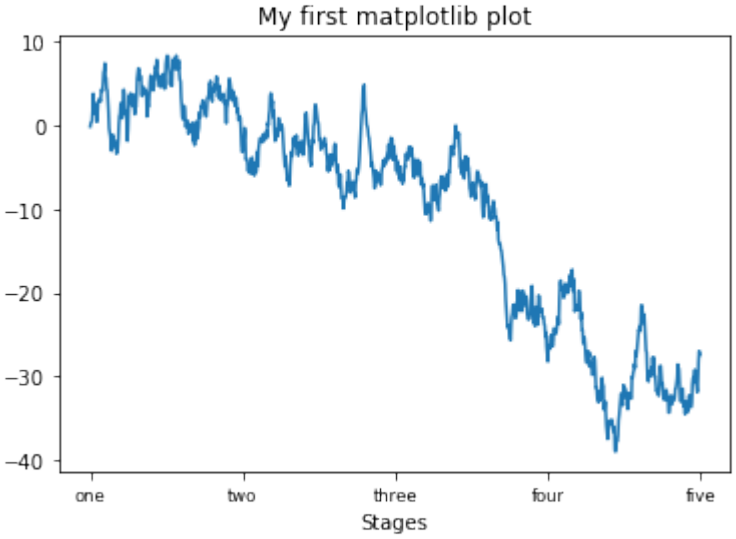
Out[25]:
```
[<matplotlib.lines.Line2D at 0x24f833de080>]
```



그래프 세부내용(범례, 축 이름 등)

In [26]:
```
fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
ax.plot(randn(1000).cumsum())

ticks = ax.set_xticks([0, 250, 500, 750, 1000])
labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
                            fontsize='small')
ax.set_title('My first matplotlib plot') ## title 설정
ax.set_xlabel('Stages') ## x축 label 설정
```

Out[26]:
```
<matplotlib.text.Text at 0x24f8334acf8>
```



실습예제6

위의 예제 그래프를 그대로 그리되 그래프의 제목, x축 이름을 한글로 바꾸고 y축의 이름을 추가하시오.

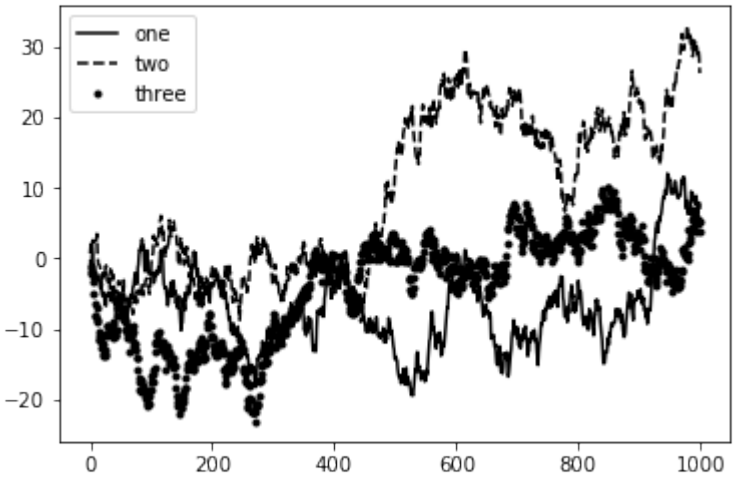
In [ ]:
```

```

범주 추가

In [27]:
```
fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
```

```
ax.plot(randn(1000).cumsum(), 'k', label='one')
ax.plot(randn(1000).cumsum(), 'k--', label='two')
ax.plot(randn(1000).cumsum(), 'k.', label='three')

ax.legend(loc='best') #최적의 장소에 범례를 위치 (loc = location)
```

Out[27]: <matplotlib.legend.Legend at 0x24f833bb748>



실습예제7

위의 그래프의 범례를 오른쪽 아래 위치로 이동시키십시오.(힌트 : location 위치에 아무 글씨나 집어넣으면 힌트가 나옵니다.)

In [ ]:

```
```

subplot 주석달기 및 그리는 방법

In [28]:

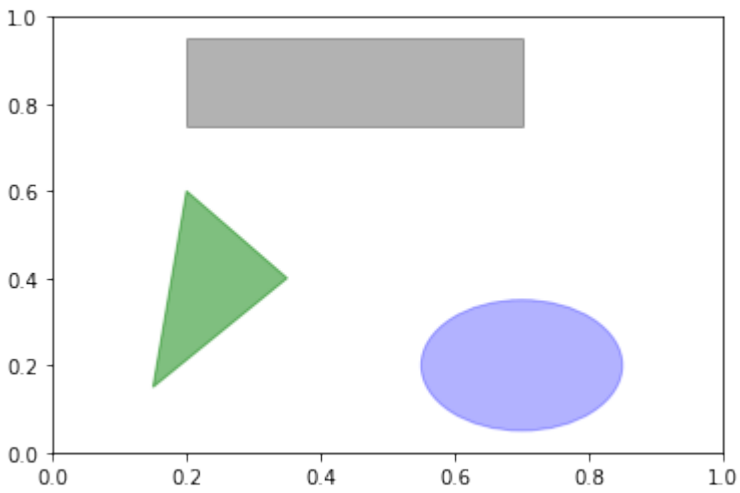
```
from datetime import datetime

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

data = pd.read_csv('data/spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']

spx.plot(ax=ax, style='k-')

crisis_data = [
    (datetime(2007, 10, 11), 'Peak of bull market'),
    (datetime(2008, 3, 12), 'Bear Stearns Fails'),
    (datetime(2008, 9, 15), 'Lehman Bankruptcy')
]

for date, label in crisis_data:
    ax.annotate(label, xy=(date, spx.asof(date) + 50), #asof함수를 사용, x,y 좌표를 해당 날짜의 가장 근접한 위치로 함, +는 위치설정
                xytext=(date, spx.asof(date) + 200),
                arrowprops=dict(facecolor='black'), #화살표
                horizontalalignment='left', verticalalignment='top')#주석달기

ax.set_xlim(['1/1/2007', '1/1/2011'])
```

```
ax.set_ylim([600, 1800])

ax.set_title('Important dates in 2008-2009 financial crisis')
```

Out[28]: <matplotlib.text.Text at 0x24f82cddf98>

In [29]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

rect = plt.Rectangle((0.2, 0.75), 0.5, 0.2, color='k', alpha=0.3) ##(0.2,0.75) 사각형의 시작점 0.5 가로길이 0.2 세로길이를 뜻한다.
                                                                  ## alpha 색깔의 명도를 나타내준다 인수를 바꿔서 결과 변화를 확인해보자!
circ = plt.Circle((0.7, 0.2), 0.15, color='b', alpha=0.3) ## (0,7,0,2) 원의 중심 을 뜻한다 / 0.15 원의 반지름을 뜻한다.
pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]], #꼭지점의 좌표
                   color='g', alpha=0.5)

ax.add_patch(rect) ## ax subplot에 도형을 추가한다.
ax.add_patch(circ)
ax.add_patch(pgon)
```

Out[29]: <matplotlib.patches.Polygon at 0x24f834dccc0>



실습예제8

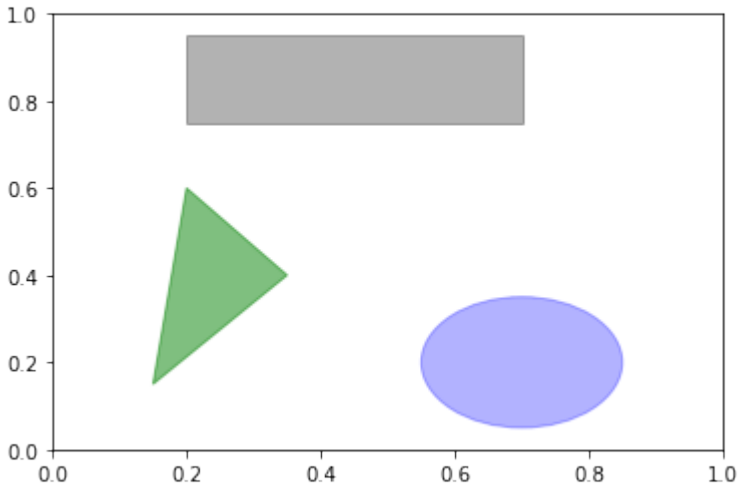
위의 그래프를 응용해 오각형 도형을 그리시오.

In [ ]:

그림(그래프)를 파일로 저장하기

In [30]:
```
fig
```

Out[30]:



In [31]:
```
fig.savefig('figpath.svg')
```

In [32]:
```
fig.savefig('figpath.png', dpi=400, bbox_inches='tight') #save figure (.savefig())
```

In [33]:
```
from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()
```

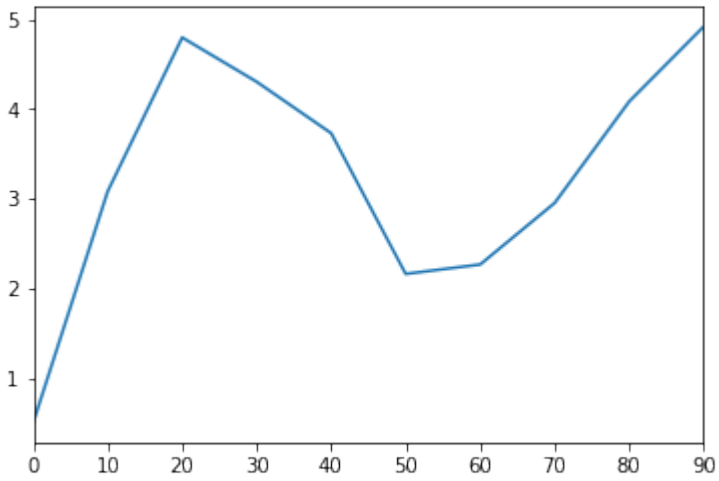
```
<matplotlib.figure.Figure at 0x24f8347f6d8>
```

pandas에서의 제도 방법

선 그래프

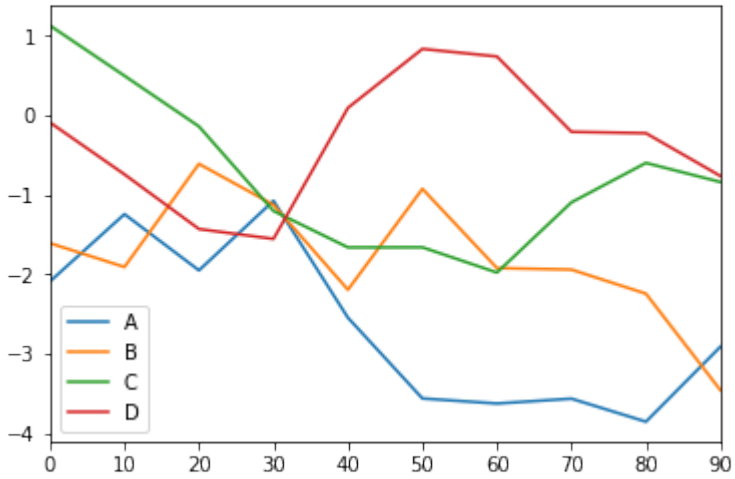
In [34]:
```
s = Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
s.plot()
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24f83499be0>
```



In [35]:
```
df = DataFrame(np.random.randn(10, 4).cumsum(0),
               columns=['A', 'B', 'C', 'D'],
               index=np.arange(0, 100, 10))
df.plot()
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x24f83546c50>



실습예제9

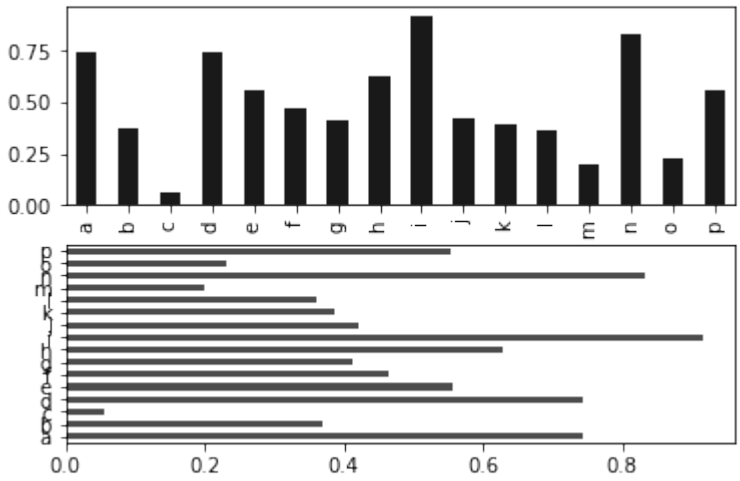
위 그래프의 라인을 7개로 늘리십시오.

In [ ]:
```
```

막대 그래프

In [36]:
```
fig, axes = plt.subplots(2, 1)
data = Series(np.random.rand(16), index=list('abcdefghijklmnop')) #x축의 이름을 list로 문자열형태로 지정
data.plot(kind='bar', ax=axes[0], color='k', alpha=0.9)
data.plot(kind='barh', ax=axes[1], color='k', alpha=0.7)
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x24f8368d780>



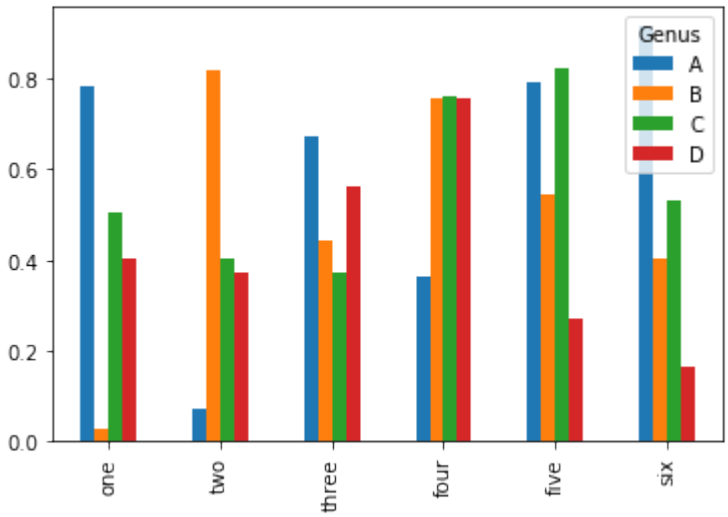
실습예제**10**

상단의 1번 그래프의 x축과 2번그래프의 y축을 0부터 9까지의 10개의 숫자로 바꾸고 그래프의 색을 노란색으로 바꿔 그리십시오.

In [ ]:

In [37]:
```
df = DataFrame(np.random.rand(6, 4),
               index=['one', 'two', 'three', 'four', 'five', 'six'],
               columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
df
df.plot(kind='bar')
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x24f837b0240>



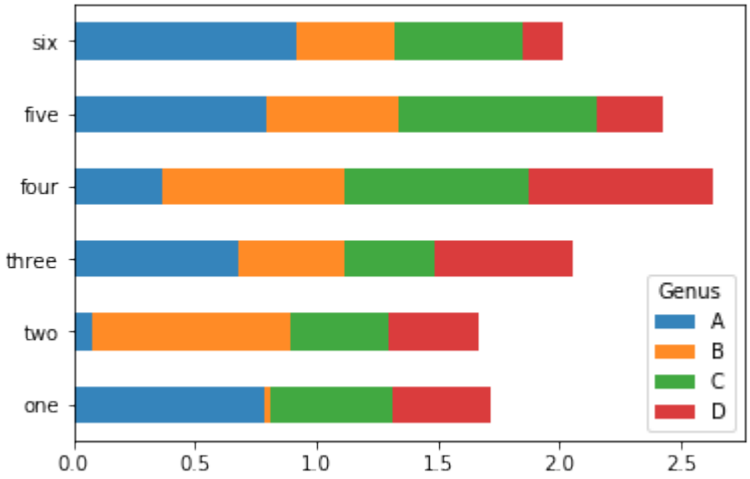
In [38]:
```
plt.figure()
```

Out[38]: <matplotlib.figure.Figure at 0x24f8360eb70>

<matplotlib.figure.Figure at 0x24f8360eb70>

In [39]:
```
df.plot(kind='barh', stacked=True, alpha=0.9)
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x24f832cf198>



In [40]:
```
tips = pd.read_csv('data/tips.csv')
party_counts = pd.crosstab(tips['day'], tips['size'])
print(party_counts)
# 2~5명의 파티 데이터만 추출
ptt=party_counts.ix[:, 2:5]
print(ptt)
```

```
size  1   2   3   4  5  6
day
Fri   1  16   1   1  0  0
Sat   2  53  18  13  1  0
Sun   0  39  15  18  3  1
Thur  1  48   4   5  1  3
size   2   3   4  5
day
Fri   16   1   1  0
Sat   53  18  13  1
Sun   39  15  18  3
Thur  48   4   5  1
```

In [41]:
```
tips.head(10)
```

Out[41]:

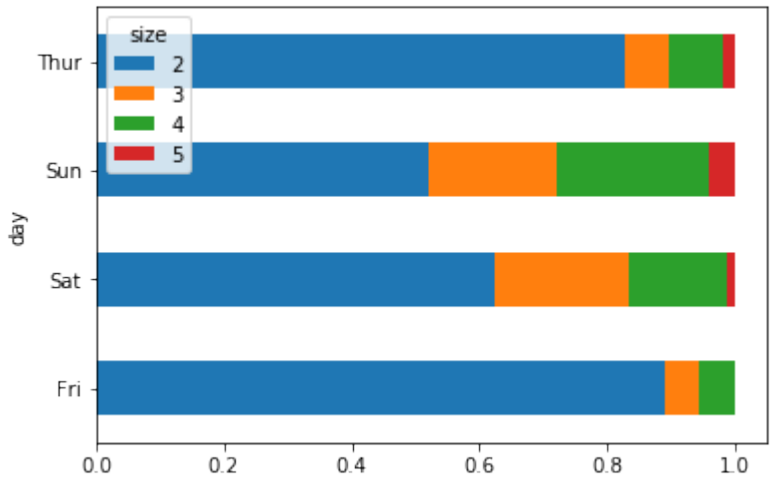
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2

In [42]:
```
party_pcts = ptt.div(ptt.sum(1).astype(float), axis=0)
print(party_pcts)
```

```
party_pcts.plot(kind='barh', stacked=True)
```

```
size          2         3         4         5
day
Fri   0.888889  0.055556  0.055556  0.000000
Sat   0.623529  0.211765  0.152941  0.011765
Sun   0.520000  0.200000  0.240000  0.040000
Thur  0.827586  0.068966  0.086207  0.017241
```

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x24f830ae400>



실습예제**11**

위 그래프를 가로축 중심의 그래프로 그리시오.

In [ ]:

히스토그램 및 밀도 그래프

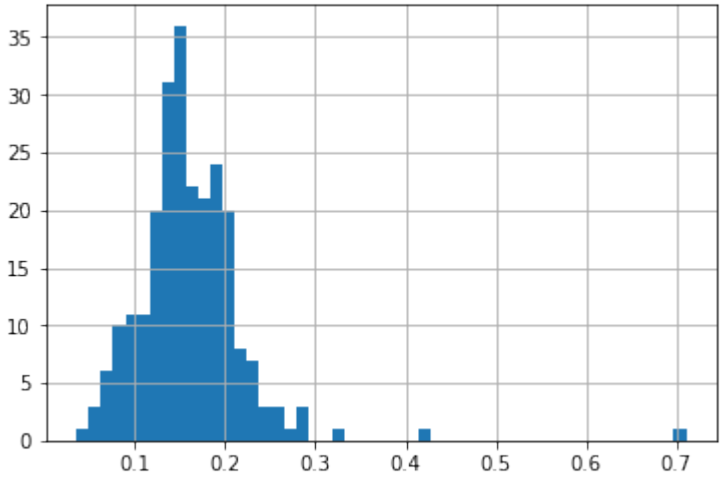
In [43]:
```
plt.figure()
```

Out[43]: <matplotlib.figure.Figure at 0x24f83494128>

<matplotlib.figure.Figure at 0x24f83494128>

In [44]:
```
tips['tip_pct'] = tips['tip'] / tips['total_bill']
tips['tip_pct'].hist(bins=50) # bins는 막대의 범위(굵기)이다.
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x24f832036a0>



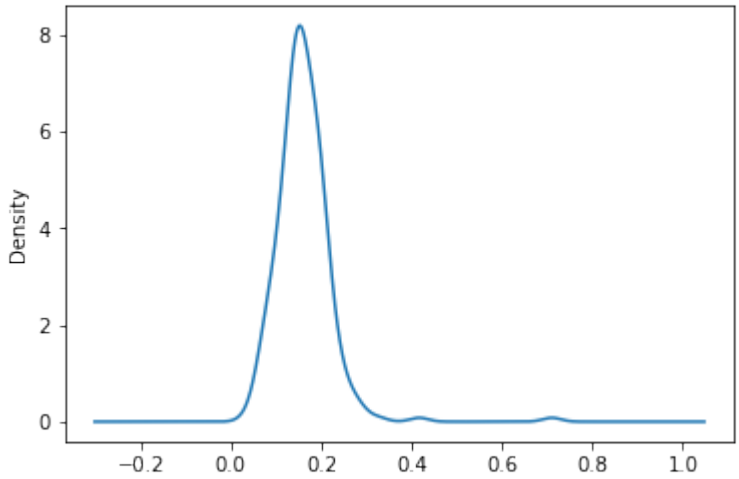
In [45]:
```
plt.figure()
```

Out[45]:
```
<matplotlib.figure.Figure at 0x24f83577b38>
```

```
<matplotlib.figure.Figure at 0x24f83577b38>
```

In [46]:
```
# 커널 밀도 추정 그래프 Kernel Sensity Estimate
tips['tip_pct'].plot(kind='kde')
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24f83139b38>
```



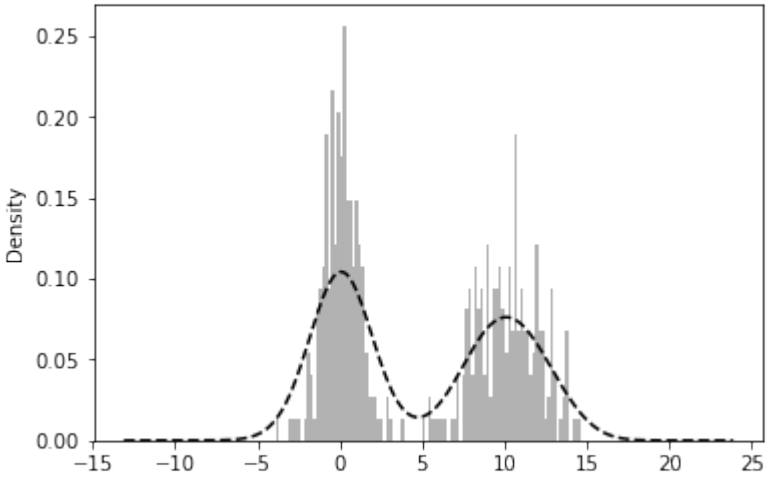
In [47]:
```
plt.figure()
```

Out[47]:
```
<matplotlib.figure.Figure at 0x24f830bd2b0>
```

```
<matplotlib.figure.Figure at 0x24f830bd2b0>
```

In [48]:
```
comp1 = np.random.normal(0, 1, size=200)  # N(0, 1)
comp2 = np.random.normal(10, 2, size=200)  # N(10, 4)
values = Series(np.concatenate([comp1, comp2])) #concat(1,2) -1번째 문자열에 두번째 문자열을 합치는 함수와 동일
values.hist(bins=100, alpha=0.3, color='k', normed=True)
values.plot(kind='kde', style='k--')
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24f84051e80>
```



실습예제**12**

위의 그래프의 막대 부분의 색을 진하게 하고 점선으로 된 커널 밀도 추정 그래프를 실선으로 표시하시오.

In [ ]:

```
```

산점도 그래프

In [49]:

```
macro = pd.read_csv('data/macrodata.csv')
data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
trans_data = np.log(data).diff().dropna() #dropna는 nan 즉 0의 값을 뺀다는 의미. diff()는 미분하는 함수, log()는 log함수를 취하는 함수.
trans_data[-5:]
```

Out[49]:

	cpi	m1	tbilrate	unemp
198	-0.007904	0.045361	-0.396881	0.105361
199	-0.021979	0.066753	-2.277267	0.139762
200	0.002340	0.010286	0.606136	0.160343
201	0.008419	0.037461	-0.200671	0.127339
202	0.008894	0.012202	-0.405465	0.042560

In [50]:

```
plt.figure()
```

Out[50]:

```
<matplotlib.figure.Figure at 0x24f8334a588>
```

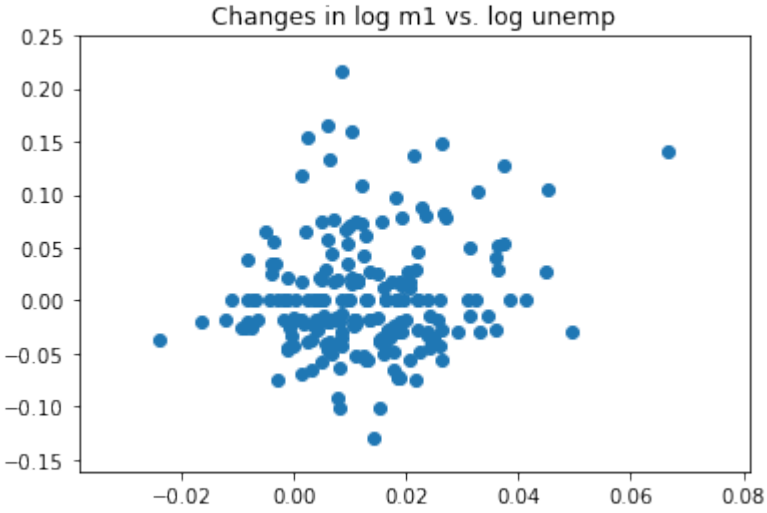
```
<matplotlib.figure.Figure at 0x24f8334a588>
```

In [51]:

```
plt.scatter(trans_data['m1'], trans_data['unemp'])
plt.title('Changes in log %s vs. log %s' % ('m1', 'unemp'))
```

Out[51]:

```
<matplotlib.text.Text at 0x24f84150c18>
```



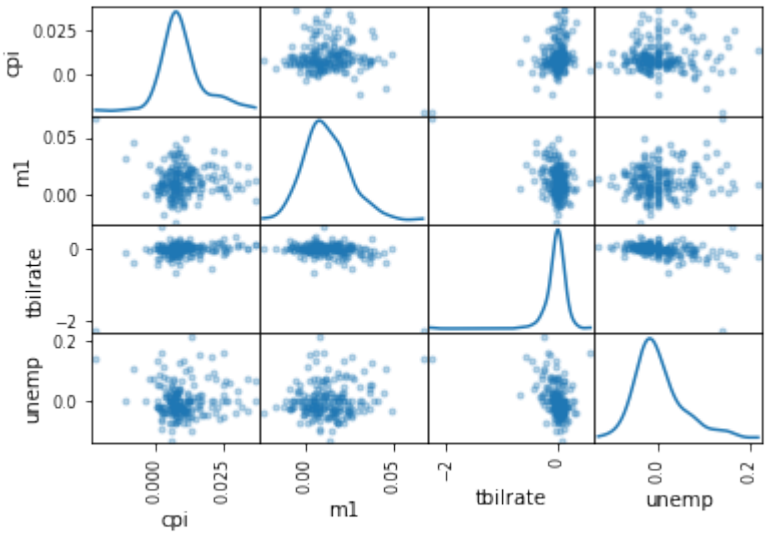
실습예제**13**

위의 Scatter plot의 x축을 trans_data의 cpi, y축을 tbilrate로 한 scatter plot을 그리고 제목을 그에 맞게 바꾸시오.

In [ ]:

In [52]:
```
pd.scatter_matrix(trans_data, diagonal='kde', alpha=0.3) #모든 x축과 y축을 사용해 행렬의 형태로 나타내는 scatter_matrix
```

Out[52]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000024F84217518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F842C87F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F84370C18>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F8444A358>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000024F844B4780>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F844B47B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F845780F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F8610AE10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000024F86179358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F861C8C50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F8623F198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F8625D080>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000024F862FAF98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F8636B2E8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F863BDBE0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000024F86425EF0>]], dtype=object)
```



실습예제**14**

위의 산점도 매트릭스의 산점도의 점 색을 붉은 색으로 하고 모양을 네모로 바꾸시오(marker 이용)

In [ ]:

클러스터링(K-Means)

In [1]:
```
# 모듈 선언
import pandas as pd
import numpy as np
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial import distance
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import seaborn as sns
```

In [2]:
```
from sklearn.metrics import jaccard_similarity_score
# 자카드 유사도 - (두 집합의 교집합 수) / (두 집합의 합집합 수)
y_pred = [0, 1, 2, 7]
y_true = [0, 1, 2, 7]
print(jaccard_similarity_score(y_true, y_pred))
# print(jaccard_similarity_score(y_true, y_pred, normalize=False))
```

```
1.0
```

Example 1 - 계층적 클러스터링와 KMeans 클러스터링

계층적 클러스터링

In [3]:
```
# scipy 덴드로그램
data1 = pd.read_csv('data/sample2d.csv') # 주어진 데이터를 read
data1 # 출력
```

Out[3]:

	x	y	name
0	4.0	3.5	A
1	3.5	4.0	B
2	4.0	4.0	C
3	5.0	5.0	D
4	5.0	5.5	E
5	5.5	5.0	F
6	4.0	6.0	G
7	6.5	4.5	H

In [4]:
```
# data1의 원소들중 무작위 n개 추출 => data1.sample(n)
ksample = data1.sample(8) # 8중 8개를 추출하여, 원소의 순서만 바꾼 결과가 됨
print(ksample)
```

```
     x    y name
2  4.0  4.0    C
3  5.0  5.0    D
0  4.0  3.5    A
```

```
5  5.5  5.0    F
6  4.0  6.0    G
4  5.0  5.5    E
7  6.5  4.5    H
1  3.5  4.0    B
```

In [9]:
```
ksample = data1.drop('name', axis=1) # 'name' 순으로 정렬
print(ksample)
```

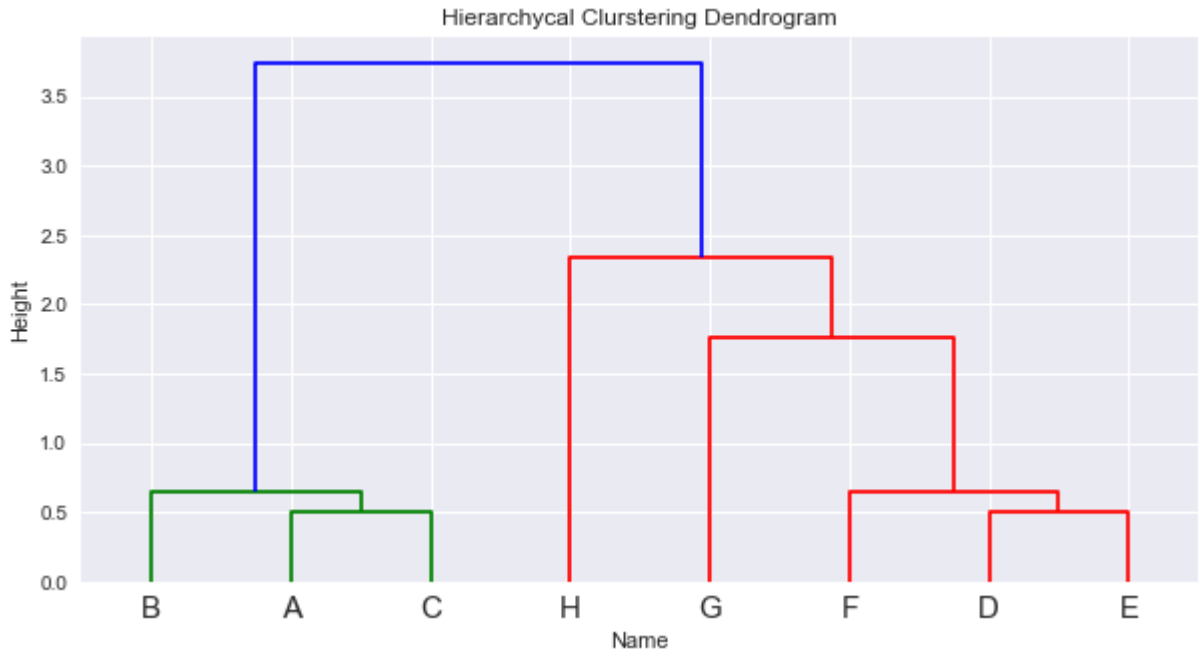
```
     x    y
0  4.0  3.5
1  3.5  4.0
2  4.0  4.0
3  5.0  5.0
4  5.0  5.5
5  5.5  5.0
6  4.0  6.0
7  6.5  4.5
```

In [13]:
```
# scipy Linkage
Z = linkage(ksample, metric='euclidean', method='ward') # 유클리드 거리를 이용해 Linkage Matrix를 생성
Z
```

Out[13]:
```
array([[  0.        ,   2.        ,   0.5       ,   2.        ],
       [  3.        ,   4.        ,   0.5       ,   2.        ],
       [  1.        ,   8.        ,   0.64549722,   3.        ],
       [  5.        ,   9.        ,   0.64549722,   3.        ],
       [  6.        ,  11.        ,   1.75594229,   4.        ],
       [  7.        ,  12.        ,   2.33452351,   5.        ],
       [ 10.        ,  13.        ,   3.74277081,   8.        ]])
```

In [29]:
```
# 상단에서 구한 Linkage Matrix를 출력
plt.clf()
plt.figure(figsize=(10, 5))
plt.title('Hierarchycal Clurstering Dendrogram')
plt.xlabel('Name')
plt.ylabel('Height')
dendrogram(Z, leaf_font_size=15, labels=list(data1['name']))
plt.show()
```

```
<matplotlib.figure.Figure at 0x2d235ec1550>
```



KMeans 클러스터링

In [30]:
```
ksample.values # k sample에 속한 원소의 값을 출력
# 즉 pandas dataframe엔 array가 있다.
```

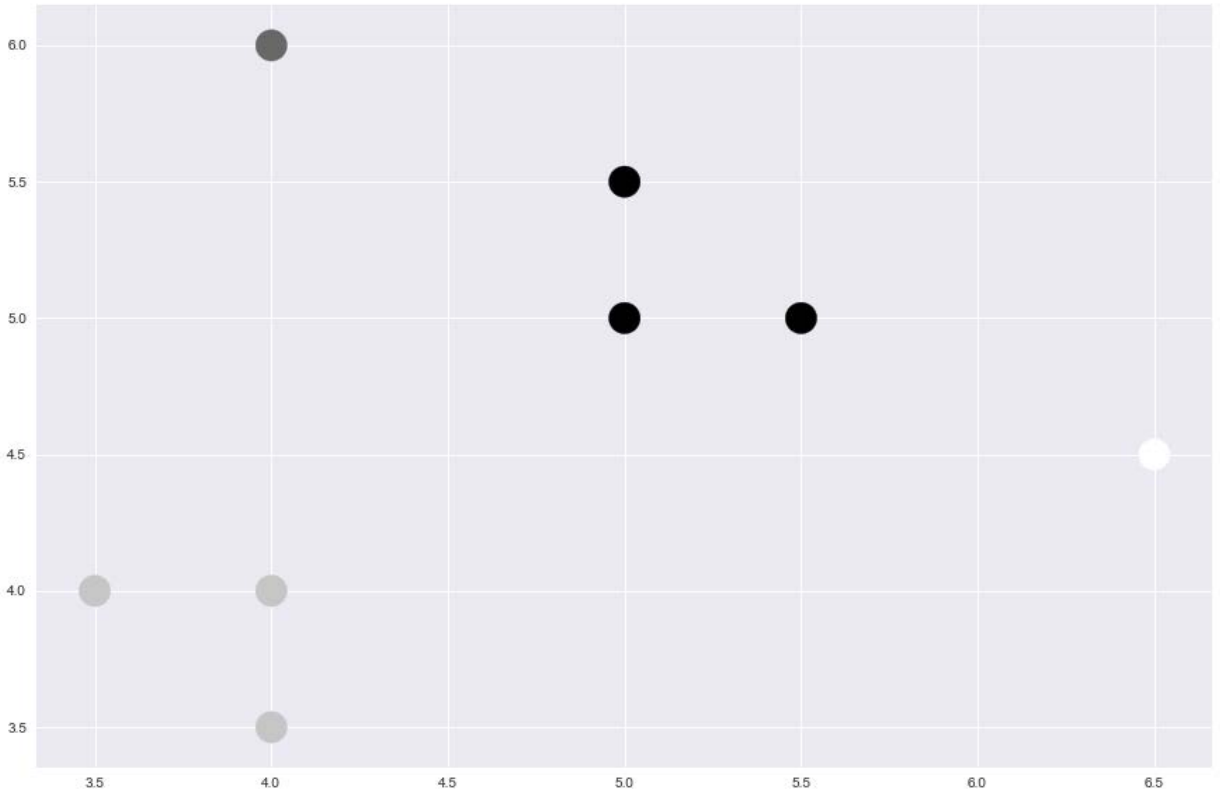
Out[30]:
```
array([[ 4. ,  3.5],
       [ 3.5,  4. ],
       [ 4. ,  4. ],
       [ 5. ,  5. ],
       [ 5. ,  5.5],
       [ 5.5,  5. ],
       [ 4. ,  6. ],
       [ 6.5,  4.5]])
```

In [31]:
```
X = ksample.values # k sample에 속한 원소의 값의 type을 출력
type(X) # array의 type은 numpy의 n dimension array
```

Out[31]:
```
numpy.ndarray
```

In [51]:
```
# sklearn모듈의 KMeans함수를 사용해 clustering을
k = KMeans(n_clusters= 4).fit(ksample)

plt.clf()
plt.figure(figsize=(15, 10))
plt.scatter(ksample['x'], ksample['y'], c=k.labels_[:],s=500)
# 결과를 출력 => ksample의 좌표를 찍고, color를 클러스터값으로 칠한다
plt.show()
```

```
<matplotlib.figure.Figure at 0x2d238c81240>
```



실습예제 **1**

sklearn 모듈의 KMeans 알고리즘을 사용하여, KSample을 3그룹으로 clustering 한 뒤 출력해 보시오.

In [ ]:

Example 2 - 운동선수 클러스터링

운동 선수들의 키와 몸무게의 정보를 이용하여, 같은 운동을 하는 선수들의 그룹을 예상하여 만들어 보는 예제이다.

In [67]:
```
heightweight = pd.read_csv('data/heightweight.csv') # 높이와 무게의 data를 read
```

In [68]:
```
heightweight.info() # 읽어온 data의 information을 출력
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 2 columns):
Height    36 non-null int64
Weight    36 non-null int64
dtypes: int64(2)
memory usage: 656.0 bytes
```

In [69]:
```
heightweight.describe() # heightweight의 수, 편차, 평균, 최소값, 최대값, 등을 출력하는 함수
```

Out[69]:

	Height	Weight
count	36.000000	36.000000
mean	171.361111	67.361111

std	15.133978	14.799909
min	143.000000	40.000000
25%	160.000000	58.500000
50%	170.000000	67.500000
75%	178.000000	78.000000
max	203.000000	92.000000

In [70]:
```
# z 정규화
zscore = lambda x : (x - x.mean())/ x.std() # 정규화 식을 lamda를 이용해 함수 zscore 생성
heightweight_z = heightweight.apply(zscore) # 정규화 함수를 이용해 새로운 dataframe 생성
heightweight_z
```

Out[70]:

	Height	Weight
0	-0.089937	0.245872
1	-1.411467	-1.443327
2	0.636904	-0.227104
3	-1.874002	-1.646031
4	-0.089937	0.583712
5	-1.015008	-0.497375
6	1.363745	0.853984
7	-0.750702	-1.510895
8	-1.741849	-1.848735
9	-0.948932	-1.240623
10	-0.222090	-0.024400
11	1.165516	0.921552
12	1.429822	0.583712
13	1.826280	0.381008
14	2.090586	0.718848
15	0.108292	-0.902783
16	-1.081085	-1.375759
17	-0.750702	-1.037919
18	-0.222090	-0.091968
19	-0.552473	-0.227104
20	-0.222090	-0.227104
21	0.108292	-0.159536
22	0.372598	0.043168
23	0.240445	0.178304

24	0.306521	-0.024400
25	-0.089937	-0.362239
26	-1.081085	-1.713599
27	0.769057	0.516144
28	1.495898	0.718848
29	1.760204	0.989120
30	-0.552473	0.718848
31	-0.288167	1.056688
32	-0.089937	1.462096
33	0.108292	1.664800
34	-0.750702	1.259392
35	0.042216	1.664800

In [71]:
```
cluster1 = KMeans(n_clusters= 3).fit(heightweight_z) #3 그룹으로 클러스터링
cluster1
```

Out[71]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [72]:
```
#데이터에 clurster colum 추가
heightweight['cluster'] = cluster1.labels_[:] # example1과 다르게 dataframe 내부에 정보 삽입
heightweight
```

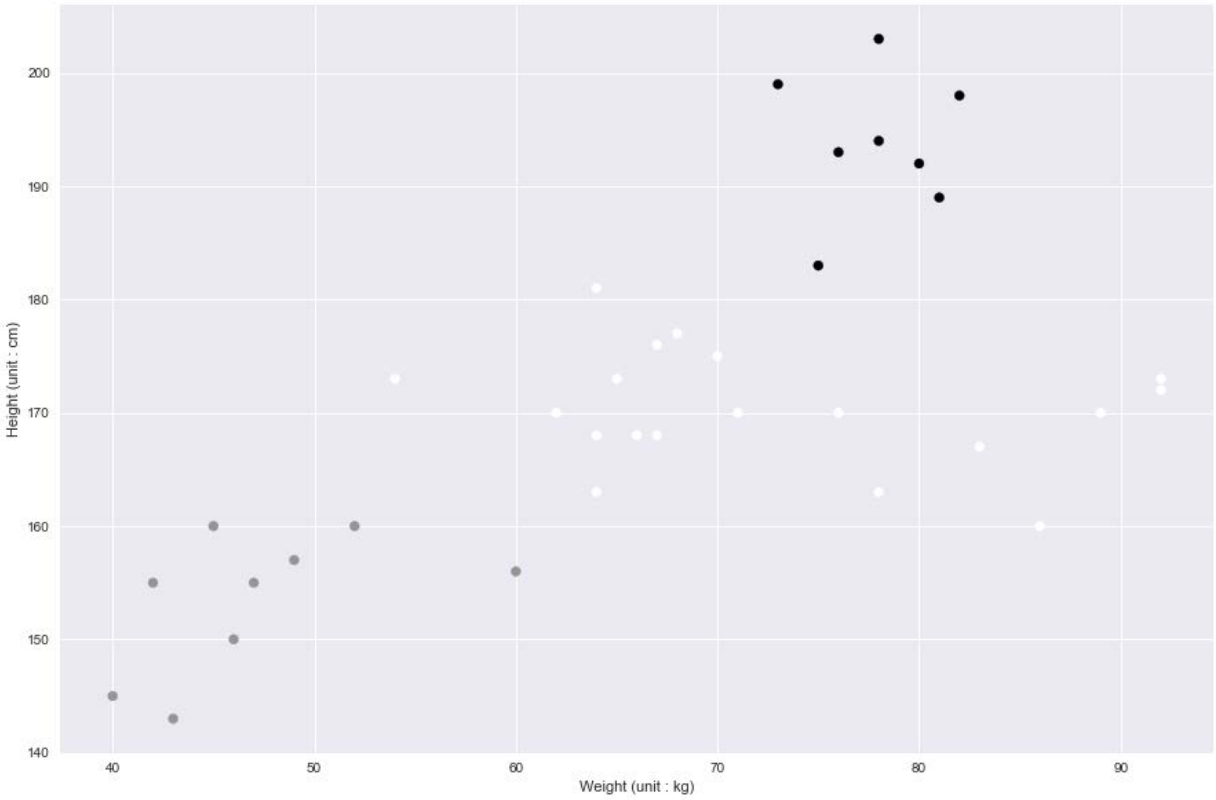
Out[72]:

	Height	Weight	cluster
0	170	71	0
1	150	46	1
2	181	64	0
3	143	43	1
4	170	76	0
5	156	60	1
6	192	80	2
7	160	45	1
8	145	40	1
9	157	49	1
10	168	67	0
11	189	81	2
12	193	76	2
13	199	73	2
14	203	78	2

15	173	54	0
16	155	47	1
17	160	52	1
18	168	66	0
19	163	64	0
20	168	64	0
21	173	65	0
22	177	68	0
23	175	70	0
24	176	67	0
25	170	62	0
26	155	42	1
27	183	75	2
28	194	78	2
29	198	82	2
30	163	78	0
31	167	83	0
32	170	89	0
33	173	92	0
34	160	86	0
35	172	92	0

In [73]:
```
plt.clf()
plt.figure(figsize=(15, 10))
plt.ylabel('Height (unit : cm)')
plt.xlabel('Weight (unit : kg)')
plt.scatter(heightweight['Weight'], heightweight['Height'], c=heightweig
ht['cluster'])
# dataframe 내부에 clurster정보가 존재하기 때문에 color에 cluster1.labels_[:]를 사용하지 않음
plt.show()
```

```
<matplotlib.figure.Figure at 0x2d235ea1ba8>
```



실습예제 **2**

sklearn 모듈의 kmeans 함수를 이용해 heightweight를 5그룹으로 clustering 한 뒤 출력 해 보시오. (단, cluster정보는 dataframe 내부에 존재해야 한다.)

In [ ]:

```
```

Example 3 - 자동차 클러스터링

자동차들의 정보들을 이용하여 클러스터링하는 예제

- MPG(1갤런으로 갈 수 있는 마일)
- GPM(100마일을 가는데 필요한 갤런)
- WT(차의 무게 1000파운드 단위)
- DIS(배기량)
- NC(실린더 수)
- HP(마력 수)
- ACC(가속도)
- ET(엔진 유형)

In [74]:

```
fuel = pd.read_csv('data/FuelEfficiency.csv') # data read
fuel.head(20)
```

Out[74]:

	MPG	GPM	WT	DIS	NC	HP	ACC	ET
0	16.9	5.917	4.360	350	8	155	14.9	1
1	15.5	6.452	4.054	351	8	142	14.3	1
2	19.2	5.208	3.605	267	8	125	15.0	1
3	18.5	5.405	3.940	360	8	150	13.0	1

4	30.0	3.333	2.155	98	4	68	16.5	0
5	27.5	3.636	2.560	134	4	95	14.2	0
6	27.2	3.676	2.300	119	4	97	14.7	0
7	30.9	3.236	2.230	105	4	75	14.5	0
8	20.3	4.926	2.830	131	5	103	15.9	0
9	17.0	5.882	3.140	163	6	125	13.6	0
10	21.6	4.630	2.795	121	4	115	15.7	0
11	16.2	6.173	3.410	163	6	133	15.8	0
12	20.6	4.854	3.380	231	6	105	15.8	0
13	20.8	4.808	3.070	200	6	85	16.7	0
14	18.6	5.376	3.620	225	6	110	18.7	0
15	18.1	5.525	3.410	258	6	120	15.1	0
16	17.0	5.882	3.840	305	8	130	15.4	1
17	17.6	5.682	3.725	302	8	129	13.4	1
18	16.5	6.061	3.955	351	8	138	13.2	1
19	18.2	5.495	3.830	318	8	135	15.2	1

In [75]:
```
fuel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38 entries, 0 to 37
Data columns (total 8 columns):
MPG    38 non-null float64
GPM    38 non-null float64
WT     38 non-null float64
DIS    38 non-null int64
NC     38 non-null int64
HP     38 non-null int64
ACC    38 non-null float64
ET     38 non-null int64
dtypes: float64(4), int64(4)
memory usage: 2.5 KB
```

In [76]:
```
fuel.describe()
```

Out[76]:

	MPG	GPM	WT	DIS	NC	HP	ACC	ET
count	38.000000	38.000000	38.000000	38.000000	38.000000	38.000000	38.000000	38.
mean	24.760526	4.330605	2.862895	177.289474	5.394737	101.736842	14.857895	0.2
std	6.547314	1.156002	0.706870	88.876747	1.603029	26.444929	1.561294	0.4
min	15.500000	2.681000	1.915000	85.000000	4.000000	65.000000	11.300000	0.0
25%	18.525000	3.292500	2.207500	105.000000	4.000000	78.500000	14.025000	0.0
50%	24.250000	4.159500	2.685000	148.500000	4.500000	100.000000	14.800000	0.0
75%	30.375000	5.397750	3.410000	229.500000	6.000000	123.750000	15.775000	1.0

max	37.300000	6.452000	4.360000	360.000000	8.000000	155.000000	19.200000	1.0

In [77]:
```
# Number of Cylinders
fuel['NC'].value_counts()
```

Out[77]:
```
4    19
6    10
8     8
5     1
Name: NC, dtype: int64
```

In [78]:
```
fuel_z = fuel.apply(zscore)
```

In [79]:
```
fuel_clusters = KMeans(n_clusters=5).fit(fuel_z)
fuel_clusters.get_params()
```

Out[79]:
```
{'algorithm': 'auto',
 'copy_x': True,
 'init': 'k-means++',
 'max_iter': 300,
 'n_clusters': 5,
 'n_init': 10,
 'n_jobs': 1,
 'precompute_distances': 'auto',
 'random_state': None,
 'tol': 0.0001,
 'verbose': 0}
```

In [80]:
```
#데이터에 clurster colum 추가
fuel_z['clurster'] = fuel_clusters.labels_[:]
fuel_z
```

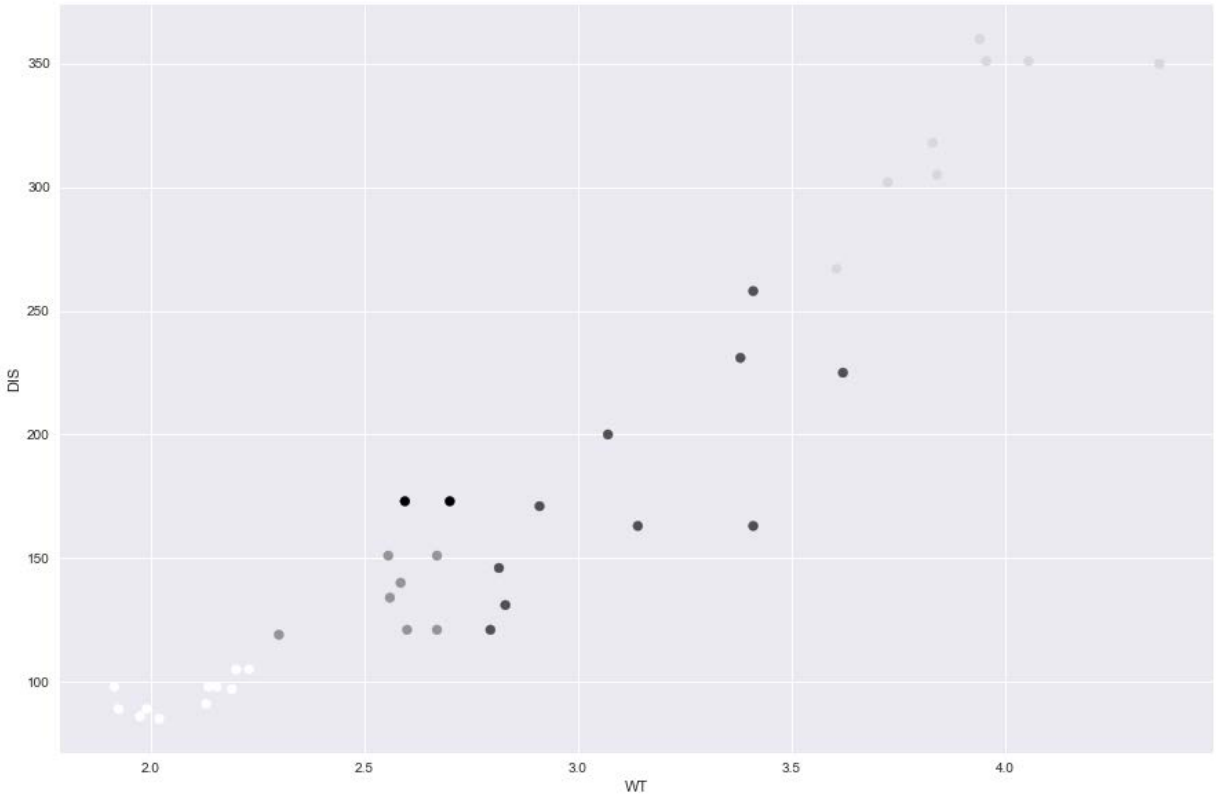
Out[80]:

	MPG	GPM	WT	DIS	NC	HP	ACC	ET	c
0	-1.200573	1.372312	2.117935	1.943259	1.625213	2.014116	0.026968	1.545947	1
1	-1.414401	1.835114	1.685040	1.954510	1.625213	1.522528	-0.357328	1.545947	1
2	-0.849284	0.758991	1.049846	1.009381	1.625213	0.879683	0.091018	1.545947	1
3	-0.956198	0.929406	1.523766	2.055774	1.625213	1.825044	-1.189971	1.545947	1
4	0.800248	-0.862979	-1.001449	-0.892128	-0.870064	-1.275740	1.051759	-0.629830	0
5	0.418412	-0.600869	-0.428501	-0.487073	-0.870064	-0.254750	-0.421378	-0.629830	2
6	0.372592	-0.566267	-0.796320	-0.655846	-0.870064	-0.179121	-0.101131	-0.629830	2
7	0.937709	-0.946889	-0.895348	-0.813368	-0.870064	-1.011039	-0.229230	-0.629830	0
8	-0.681276	0.515047	-0.046536	-0.520828	-0.246244	0.047766	0.667463	-0.629830	3
9	-1.185299	1.342035	0.392017	-0.160779	0.377575	0.879683	-0.805675	-0.629830	3
10	-0.482721	0.258992	-0.096050	-0.633343	-0.870064	0.501539	0.539364	-0.629830	3
11	-1.307487	1.593765	0.773982	-0.160779	0.377575	1.182199	0.603413	-0.629830	3
12	-0.635455	0.452763	0.731542	0.604326	0.377575	0.123394	0.603413	-0.629830	3
13	-0.604909	0.412971	0.292989	0.255528	0.377575	-0.632894	1.179858	-0.629830	3
14	-0.940924	0.904320	1.071067	0.536817	0.377575	0.312467	2.460847	-0.629830	3

15	-1.017291	1.033212	0.773982	0.908117	0.377575	0.690611	0.155067	-0.629830	3
16	-1.185299	1.342035	1.382298	1.436940	1.625213	1.068755	0.347215	1.545947	1
17	-1.093659	1.169025	1.219609	1.403185	1.625213	1.030941	-0.933773	1.545947	1
18	-1.261666	1.496879	1.544987	1.954510	1.625213	1.371271	-1.061872	1.545947	1
19	-1.002018	1.007261	1.368151	1.583210	1.625213	1.257827	0.219117	1.545947	1
20	0.265677	-0.481492	-0.393134	-0.419564	-0.870064	-0.519451	-0.293279	-0.629830	2
21	-0.436901	0.203628	0.066639	-0.070766	0.377575	0.274652	1.115809	1.545947	3
22	1.426459	-1.209000	-1.256093	-1.027147	-0.870064	-1.389183	0.219117	-0.629830	0
23	1.579193	-1.281664	-1.340974	-0.892128	-0.870064	-0.821966	-0.293279	-0.629830	0
24	0.403138	-0.588758	-0.272886	-0.633343	-0.870064	-0.821966	0.091018	-0.629830	2
25	1.029349	-0.999657	-1.234872	-0.993392	-0.870064	-1.162296	0.026968	-0.629830	0
26	0.723881	-0.813671	-1.029743	-0.892128	-0.870064	-1.275740	1.115809	-0.629830	0
27	0.555873	-0.700350	-0.272886	-0.295797	-0.870064	-0.443822	0.731512	-0.629830	2
28	0.616967	-0.742737	-0.378987	-0.048263	0.377575	0.501539	-2.278812	1.545947	4
29	0.311498	-0.518689	-0.230445	-0.048263	0.377575	0.501539	-1.254021	1.545947	4
30	1.334818	-1.164017	-0.434160	-0.295797	-0.870064	-0.443822	-1.061872	-0.629830	2
31	1.441732	-1.216785	-0.937788	-0.813368	-0.870064	-1.200111	-1.061872	-0.629830	0
32	1.075170	-1.025609	-1.192432	-1.038398	-0.870064	-1.389183	2.781094	-0.629830	0
33	1.915209	-1.426992	-1.036816	-0.970889	-0.870064	-1.237925	-0.101131	-0.629830	0
34	0.876615	-0.909692	-0.951935	-0.903380	-0.870064	-0.897595	-0.485427	-0.629830	0
35	-0.421627	0.185462	-0.067756	-0.352055	0.377575	-0.179121	-0.229230	-0.629830	3
36	-0.497995	0.277158	-0.371914	-0.633343	-0.870064	0.312467	-1.318070	-0.629830	2
37	1.090443	-1.034259	-1.326827	-0.993392	-0.870064	-1.162296	-0.549477	-0.629830	0

In [81]:
```
plt.clf()
plt.figure(figsize=(15, 10))
plt.xlabel('WT')
plt.ylabel('DIS')
plt.scatter(fuel['WT'], fuel['DIS'], c=fuel_z['clurster']) #클러스터링 된 자료를 무게와 배기량의 관계로 표현
plt.show()
```

```
<matplotlib.figure.Figure at 0x2d2384799b0>
```



실습예제 **3**

상단의 클러스터링 예제를 실린더 수, 마력 수 의 관계로 그래프를 출력하시오

In [ ]:

K-Nearest Neighbor(k-최근접이웃)

사용데이터 - 대형 매장에 방문하는 중국인과 일본인의 정보 (건물 입구부터 매장 상담원이 있는 곳까지 걸어 오는데 소요된 시간, 진열된 상품을 둘러본 횟수, 성별, 지불금액, 출신 국가)

위의 정보를 학습하여(성별 제외), 출신 국가가 블라인드된 정보만 보고 출신 국가를 예측하려 한다.

In [1]:
```
# 모듈 선언
import pandas as pd
from matplotlib import pyplot as plt
```

In [2]:
```
# data read
mydata = pd.read_csv('data/mydata.csv')
del mydata['sex'] # load된 dataframe에서 성별을 제거
```

In [3]:
```
mydata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 5 columns):
id          500 non-null int64
walk        500 non-null int64
view        500 non-null int64
payment     500 non-null int64
country     500 non-null object
dtypes: int64(4), object(1)
memory usage: 19.6+ KB
```

In [4]:
```
mydata['country'].value_counts() # 국가별 매장 방문 인원
```

Out[4]:
```
C    300
J    200
Name: country, dtype: int64
```

In [5]:
```
# 주어진 수치값의 최대치와 최소치의 차이값으로 정규화
nomalize = lambda x : (x - min(x)) / (max(x) - min(x))
```

In [6]:
```
# 더미 코딩
mydata['country'] = (mydata['country']=='C') # china일 경우 true, japan일 경우 false
```

In [13]:
```
mydata.ix[:, 1:4].head(10) # dataframe의 walk, view, payment만 추출하여 탐색
```

Out[13]:

	walk	view	payment
0	17	5	176
1	12	5	194
2	32	12	384
3	25	9	329
4	23	8	290
5	24	8	246

6	22	6	235
7	32	11	353
8	30	10	324
9	13	2	93

In [12]:
```
mydata_n = mydata.ix[:, 1:4].apply(nomalize) # dataframe의 walk, view, payment를 이용하여 정규화
mydata_n.head(10)
```

Out[12]:

	walk	view	payment
0	0.240741	0.157895	0.268025
1	0.148148	0.157895	0.296238
2	0.518519	0.526316	0.594044
3	0.388889	0.368421	0.507837
4	0.351852	0.315789	0.446708
5	0.370370	0.315789	0.377743
6	0.333333	0.210526	0.360502
7	0.518519	0.473684	0.545455
8	0.481481	0.421053	0.500000
9	0.166667	0.000000	0.137931

In [57]:
```
feature_names = ['walk', 'view', 'payment'] # feature_name에 학습에 사용할 column name을 저장
X = mydata_n[feature_names] # feature_name에 해당하는 column만 정규화된 mydata_n에서 추출하여 X(Large x)에 저장
```

In [58]:
```
y = mydata['country'] # y(small y)에 mydata의 country를 저장
```

In [59]:
```
# 훈련 데이터와 테스트 데이터 분리
from sklearn.model_selection import train_test_split

#X와 y의 training data와 test data의 비율을 3:1로 지정(0.75, 0.25)
mydata_train, mydata_test, mydata_train_labels, mydata_test_labels = tra
in_test_split(X, y,

                test_size=0.25, random_state=33)
```

In [60]:
```
# KNN Algorism 실행
from sklearn.neighbors import KNeighborsClassifier as knn # knn모듈을 import
X = mydata_train # train (훈련 데이터중 column) => X
y = mydata_train_labels # train labels (훈련 데이터중 출신 국가) => y
mydata_test_pred = knn(n_neighbors=17) # neighbor의 수를 17로 정함
mydata_test_pred = mydata_test_pred.fit(X, y) # data를 학습
```

In [61]:
```
# 분류 검증
from sklearn import metrics
y_pred = mydata_test_pred.predict(mydata_test) # test용으로 정해놓았던 data를 학
```

```
습된 데이터를 이용해 출신 국가를 예측
print(metrics.classification_report(mydata_test_labels, y_pred)) # test용으로 빼 놓았던 data(출신국가 정답)과 예측값을 비교
```

```
             precision    recall  f1-score   support

      False       0.93      0.96      0.95        56
       True       0.97      0.94      0.96        69

avg / total       0.95      0.95      0.95       125
```

실습예제

sklearn 모듈의 knn 함수를 이용하여 반지름을 30으로 설정하고, train data와 test data를 9:1로 정하여 예측해보시오.

In [ ]:

의사결정트리

knn 예제와 동일한 예측을 의사결정트리 알고리즘으로 풀이 한다.

In [12]:
```
import pandas as pd
from sklearn import tree
from matplotlib import pyplot as plt
```

In [13]:
```
data = pd.read_csv('data/mydata.csv')
data.head(100)
```

Out[13]:

	id	walk	view	sex	payment	country
0	1	17	5	M	176	C
1	2	12	5	M	194	C
2	3	32	12	F	384	C
3	4	25	9	F	329	C
4	5	23	8	F	290	C
5	6	24	8	M	246	C
6	7	22	6	F	235	C
7	8	32	11	F	353	C
8	9	30	10	F	324	C
9	10	13	2	M	93	C
10	11	27	9	M	306	C
11	12	30	10	F	330	C
12	13	30	9	F	314	C
13	14	8	3	M	42	C
14	15	23	7	F	267	C
15	16	39	13	F	440	C
16	17	31	11	F	369	C
17	18	21	6	F	247	C
18	19	13	3	M	71	C
19	20	28	9	F	287	C
20	21	24	6	F	222	C
21	22	32	10	F	359	C
22	23	27	8	F	269	C
23	24	26	8	F	266	C
24	25	26	7	M	224	C

25	26	26	7	F	255	C
26	27	16	4	M	80	C
27	28	35	11	F	414	C
28	29	15	4	M	120	C
29	30	25	7	M	243	C
...	...	...	...	...	...	...
70	71	31	11	F	368	C
71	72	27	8	F	281	C
72	73	21	6	F	196	C
73	74	22	7	F	225	C
74	75	21	7	M	230	C
75	76	26	8	F	256	C
76	77	26	8	F	287	C
77	78	31	9	F	300	C
78	79	27	8	F	299	C
79	80	22	6	F	233	C
80	81	18	6	M	235	C
81	82	29	8	F	294	C
82	83	25	7	M	254	C
83	84	26	7	F	252	C
84	85	33	11	F	339	C
85	86	28	12	F	364	C
86	87	18	5	M	141	C
87	88	30	12	F	362	C
88	89	19	10	M	334	C
89	90	27	7	F	249	C
90	91	27	7	F	262	C
91	92	23	6	M	212	C
92	93	35	11	F	387	C
93	94	20	5	M	159	C
94	95	30	9	F	296	C
95	96	25	7	M	259	C
96	97	18	5	M	164	C
97	98	34	11	F	371	C
98	99	10	2	M	97	C
99	100	21	8	F	270	C

100 rows × 6 columns

In [14]:
```
# 더미 코딩
data['country'] = (data['country']=='C')
data.head(5)
# 이곳 까지 KNN과 동일
```

Out[14]:

	id	walk	view	sex	payment	country
0	1	17	5	M	176	True
1	2	12	5	M	194	True
2	3	32	12	F	384	True
3	4	25	9	F	329	True
4	5	23	8	F	290	True

In [19]:
```
feature_names = ['walk', 'view'] # walk와 view 두개의 column name을 feature_name에 저장
X = data[feature_names] # data 정제 후 X에 저장
X.head(5) # 임의 출력
```

Out[19]:

	walk	view
0	17	5
1	12	5
2	32	12
3	25	9
4	23	8

In [20]:
```
y = data['country'] # country 관련 data만 y에 저장
```

In [21]:
```
# 학습용 데이터와 테스트용 데이터 작성
from sklearn.model_selection import train_test_split
# training data와 test data의 비율을 3:1로 지정
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=33)
```

In [28]:
```
clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=2, min_
samples_leaf=3) # 의사결정트리 생성(트리depth2)
clf = clf.fit(X_train, y_train) # 의사결정트리에 training data 학습
clf
```

Out[28]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth
=2,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_split=1e-07, min_samples_leaf=3,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort=False, random_state=None, splitter='best')
```

In [24]:
```
with open("./mytree.dot", 'w') as  f:
    dot_data=tree.export_graphviz(clf, out_file=f, feature_names=feature
_names) # 작성된 의사결정 트리를 파일로 작성
```

In [26]:
```
# 작성된 파일을 그림파일로 만들기
# 트리 출력은 마크다운 모드에서 ![Alt text](./mytree.png) 입력 후 실행

# !dot -Tpng mytree.dot -o mytree.png
# 상단의 명령어는 리눅스에서 동작하기 때문에 의사결정 트리의 png파일은 미리 제작하여 첨부
```

'dot'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는
배치 파일이 아닙니다.

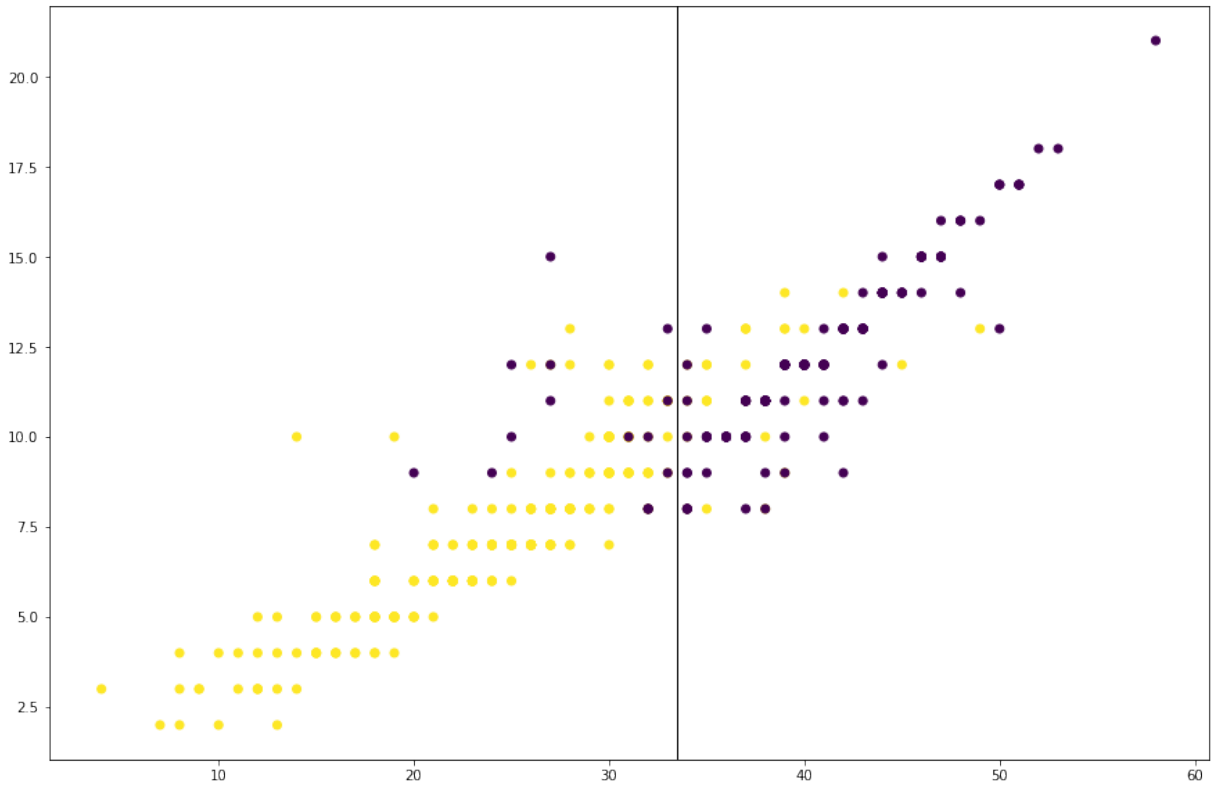
In [25]:
```
from sklearn import metrics
y_pred = clf.predict(X_test) # 학습된 의사결정트리를 이용해 출신국가 예측
print(metrics.classification_report(y_test, y_pred)) # 예측 결과 출력
```

```
             precision    recall  f1-score   support

      False       0.86      0.96      0.91        56
       True       0.97      0.87      0.92        69

avg / total       0.92      0.91      0.91       125
```

첨부한 의사결정트리

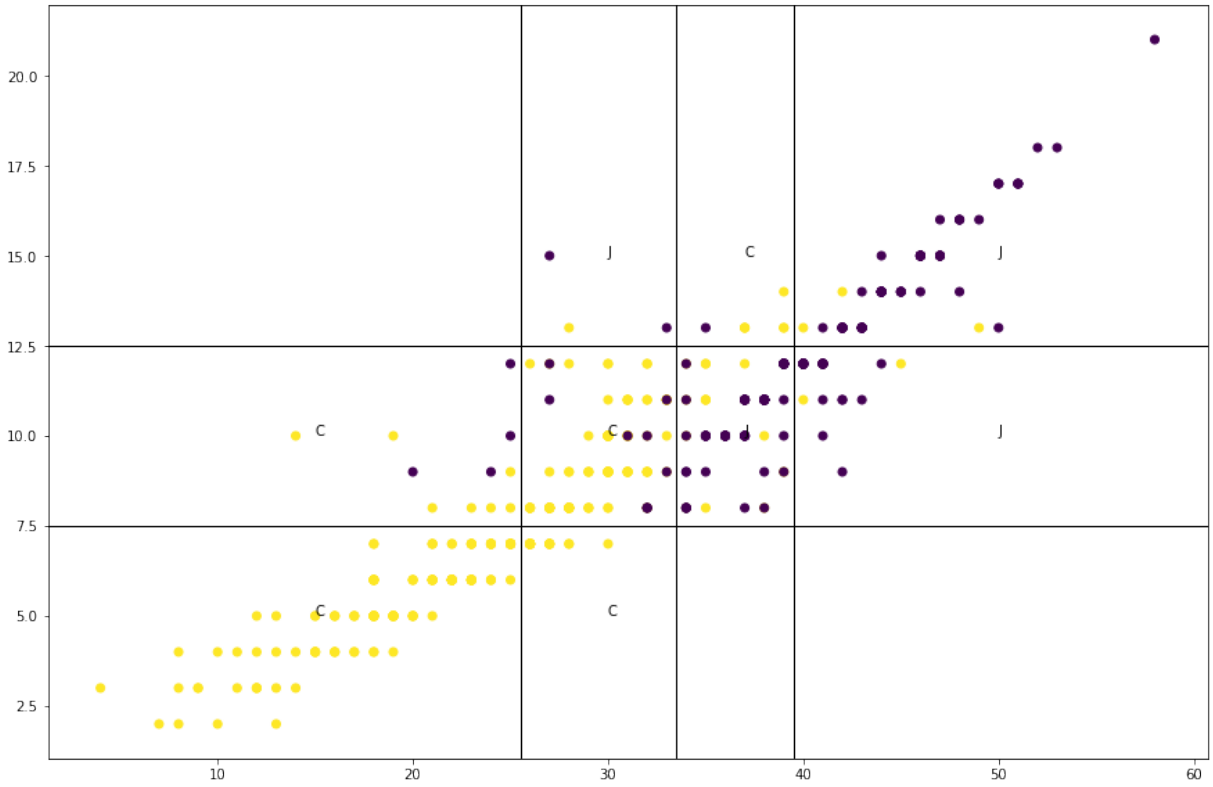
In [69]:
```
# 의사결정트리 1단계 정보를 이용한 시각화 예시
plt.clf()
plt.figure(figsize=(15, 10))
plt.xlabel='walk'
plt.ylabel='view'
plt.scatter(x=data['walk'], y=data['view'], c=data['country'])
plt.axvline(x = 33.5, color='k', linewidth=1)
plt.show()
```

```
<matplotlib.figure.Figure at 0x7fd82387ba90>
```



In [90]:
```
# 의사결정트리를 이용한 시각화 예시
plt.clf()
plt.figure(figsize=(15, 10))
plt.xlabel='walk'
plt.ylabel='view'
plt.scatter(data['walk'], data['view'], c=data['country'])
plt.axvline(x = 33.5, color='k', linewidth=1)
plt.axvline(x = 39.5, color='k', linewidth=1)
plt.axvline(x = 25.5, color='k', linewidth=1)
plt.axhline(y = 7.5, color='k', linewidth=1)
plt.axhline(y = 12.5, color='k', linewidth=1)

plt.text(x=15, y=5, s='C')
plt.text(x=15, y=10, s='C')
plt.text(x=30, y=10, s='C')
plt.text(x=30, y=5, s='C')
plt.text(x=30, y=15, s='J')
plt.text(x=37, y=10, s='J')
plt.text(x=37, y=15, s='C')
plt.text(x=50, y=10, s='J')
plt.text(x=50, y=15, s='J')

plt.show()
```

```
<matplotlib.figure.Figure at 0x7fd871ef9748>
```



실습예제

상단의 고객데이터를 이용해 의사결정트리의 깊이를 3으로 정하고, train과 test의 데이터를 9:1로 설정하여 예측하여 report 표를 출력하시오.

In [ ]:
```
```

나이브 베이어스

해당 메일이 스팸메일인지 아닌지 예측하는 문제이다.

data에는 스팸의 여부와 메일 내용이 저장되어있다.

In [28]:
```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
```

In [29]:
```
import codecs

# utf-8 디코딩 문제로 인해 해당 코드로 파일을 읽어왔다.
with codecs.open('data/sms_spam.csv', "r",encoding='utf-8', errors='ignore') as fdata:
    sms_raw = pd.read_csv(fdata)
```

In [30]:
```
sms_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5559 entries, 0 to 5558
Data columns (total 2 columns):
type    5559 non-null object
text    5559 non-null object
dtypes: object(2)
memory usage: 86.9+ KB
```

In [31]:
```
sms_raw['type'].value_counts()
```

Out[31]:
```
ham     4812
spam     747
Name: type, dtype: int64
```

In [32]:
```
sms_raw.head(10) # 임의 출력
```

Out[32]:

	type	text
0	ham	Hope you are having a good week. Just checking in
1	ham	K..give back my thanks.
2	ham	Am also doing in cbe only. But have to pay.
3	spam	complimentary 4 STAR Ibiza Holiday or £10,000 ...
4	spam	okmail: Dear Dave this is your final notice to...
5	ham	Aiya we discuss later lar... Pick u up at 4 is...
6	ham	Are you this much buzy
7	ham	Please ask mummy to call father
8	spam	Marvel Mobile Play the official Ultimate Spide...
9	ham	fyi I'm at usf now, swing by the room whenever

In [33]:
```
sms_corpus = sms_raw['text'] #dataframe의 'text'의 정보를 따로 저장
```

In [34]:
```
# text의 전처리
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=5) # 5개 이하로 나온 단어 무시
X = vectorizer.fit_transform(sms_corpus) # 문서-단어 행렬 작성 후 X에 저장
# 이는 해당 단어의 총 출연 횟수를 text에 등장한 단어 순서에 맞게 저장한 CounterVector 이다.
X.shape
```

Out[34]:
```
(5559, 1800)
```

In [35]:
```
print(X[:3]) # countervector 임의 출력
type(X)
```

```
  (0, 773)      1
  (0, 326)      1
  (0, 819)      1
  (0, 1693)     1
  (0, 656)      1
  (0, 698)      1
  (0, 162)      1
  (0, 1791)     1
  (0, 731)      1
  (1, 1514)     1
  (1, 1032)     1
  (1, 198)      1
  (1, 643)      1
  (2, 1139)     1
  (2, 1557)     1
  (2, 695)      1
  (2, 272)      1
  (2, 1101)     1
  (2, 309)      1
  (2, 468)      1
  (2, 135)      1
  (2, 137)      1
  (2, 773)      1
```

Out[35]:
```
scipy.sparse.csr.csr_matrix
```

In [36]:
```
X = X.toarray() # 학습을 위해서는 X가 array나 ndarray의 형태로 존재해야 함
y = sms_raw['type'] # 스팸 여부를 저장
```

In [37]:
```
from sklearn.model_selection import train_test_split
# 훈련데이터 75%, 테스트 데이터 25%를 지정
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=33)
```

In [38]:
```
from sklearn.naive_bayes import GaussianNB as nb
cl = nb() # 가우시안 나이브 베이어스 모델 생성
cl.fit(X_train, y_train) # 학습

from sklearn import metrics
y_pred = cl.predict(X_test) # 학습된 모델을 이용하여 예측
print(metrics.classification_report(y_test, y_pred)) # 예측 결과 출력
```

```
              precision    recall  f1-score   support

         ham       0.98      0.85      0.91      1187
```

```
          spam       0.51      0.90      0.65       203

avg / total       0.91      0.86      0.87      1390
```

In [39]:
```
# 라플라스 추정
# 라플라스 추정을 위해 MultinomialNB를 사용
from sklearn.naive_bayes import MultinomialNB as nb
cl = nb() # 멀티노미얼 나이브 베이어스 모델 생성
cl.fit(X_train, y_train) # 학습
```

Out[39]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [40]:
```
from sklearn import metrics
y_pred = cl.predict(X_test) # 학습된 모델을 이용하여 예측
print(metrics.classification_report(y_test, y_pred)) # 예측 결과 출력
```

```
             precision    recall  f1-score   support

        ham       0.99      0.99      0.99      1187
       spam       0.96      0.95      0.95       203

avg / total       0.99      0.99      0.99      1390
```

실습 예제

상단의 예제에서 주어진 email data를 train과 test데이터의 비율은 8:2로 설정한 뒤, 가우시안 나이브 베이어스 학습 모델을 이용하여 예측 결과를 출력하시오.

In [ ]:
```
```

선형회귀분석

Linear regression example 1

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

In [2]:
```
student = pd.read_csv('data/student.csv') # 데이터 읽기
```

In [3]:
```
student.head(100)
```

Out[3]:

	language	english	math	science
0	55	91	33	51
1	60	85	52	43
2	45	89	21	18
3	75	30	95	98
4	60	88	40	31
5	60	83	51	47
6	63	63	62	65
7	54	49	58	60
8	53	93	31	42
9	28	37	89	97
10	62	88	40	37
11	73	28	89	98
12	56	48	58	60
13	64	61	62	61
14	62	95	34	47
15	54	97	28	34
16	60	88	43	34
17	44	51	75	78
18	76	28	100	100
19	75	87	36	24

In [4]:
```
student.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 4 columns):
```

```
language    20 non-null int64
english     20 non-null int64
math        20 non-null int64
science     20 non-null int64
dtypes: int64(4)
memory usage: 720.0 bytes
```

In [5]:
```
student.describe() # 데이터 기초통계량 확인
```

Out[5]:

	language	english	math	science
count	20.000000	20.00000	20.000000	20.000000
mean	58.950000	68.95000	54.850000	56.250000
std	11.591626	25.07982	23.862049	25.894269
min	28.000000	28.00000	21.000000	18.000000
25%	54.000000	48.75000	35.500000	36.250000
50%	60.000000	84.00000	51.500000	49.000000
75%	63.250000	88.25000	65.250000	68.250000
max	76.000000	97.00000	100.000000	100.000000

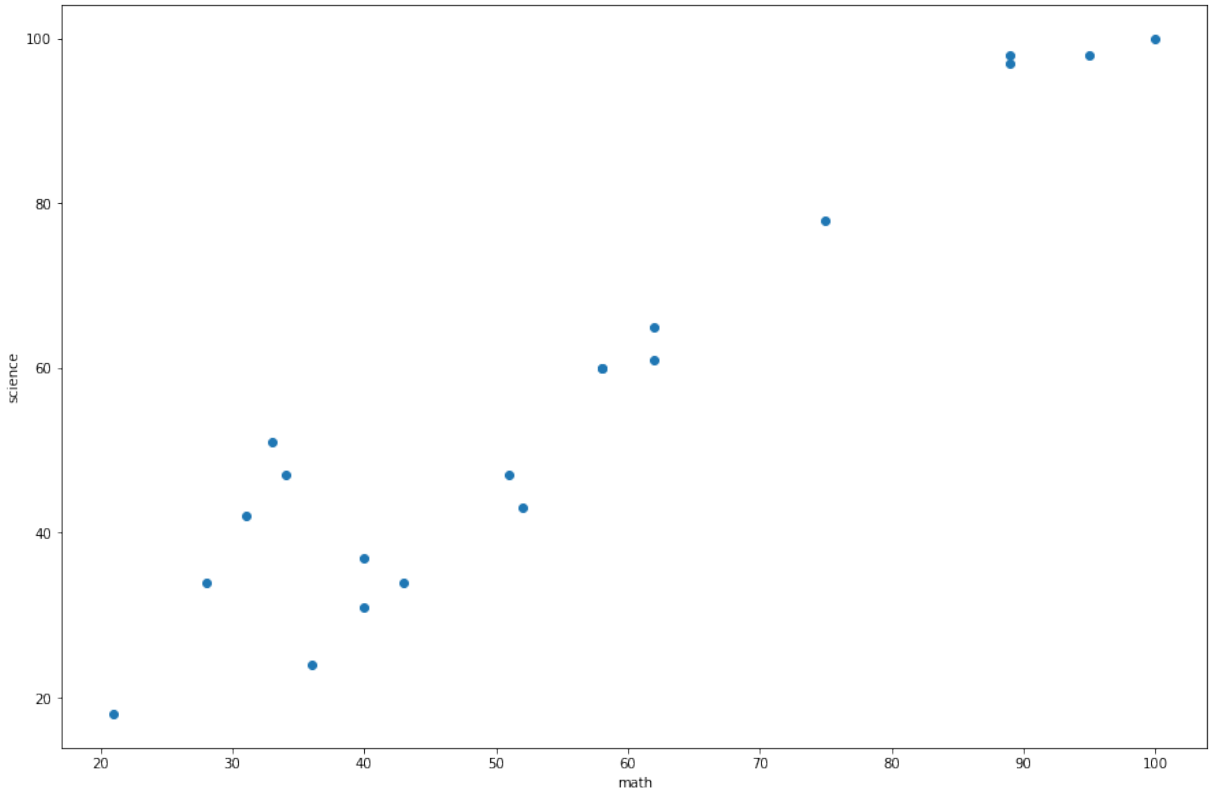
In [6]:
```
# 피어슨 상관계수
pd.concat([student, student], axis=1).corr(method='pearson')
```

Out[6]:

	language	english	math	science	language	english	math	scie
language	1.000000	-0.125833	0.183212	0.090698	1.000000	-0.125833	0.183212	0.09(
english	-0.125833	1.000000	-0.945780	-0.919824	-0.125833	1.000000	-0.945780	-0.91
math	0.183212	-0.945780	1.000000	0.951689	0.183212	-0.945780	1.000000	0.95
science	0.090698	-0.919824	0.951689	1.000000	0.090698	-0.919824	0.951689	1.00(
language	1.000000	-0.125833	0.183212	0.090698	1.000000	-0.125833	0.183212	0.09(
english	-0.125833	1.000000	-0.945780	-0.919824	-0.125833	1.000000	-0.945780	-0.91
math	0.183212	-0.945780	1.000000	0.951689	0.183212	-0.945780	1.000000	0.95
science	0.090698	-0.919824	0.951689	1.000000	0.090698	-0.919824	0.951689	1.00(

In [7]:
```
plt.clf()
plt.figure(figsize=(15, 10))
plt.xlabel('math')
plt.ylabel('science')
plt.scatter(student['math'], student['science'])
plt.show()
# 수학과 과학과의 상관계수
np.corrcoef(student['math'], student['science'])[0, 1]
```

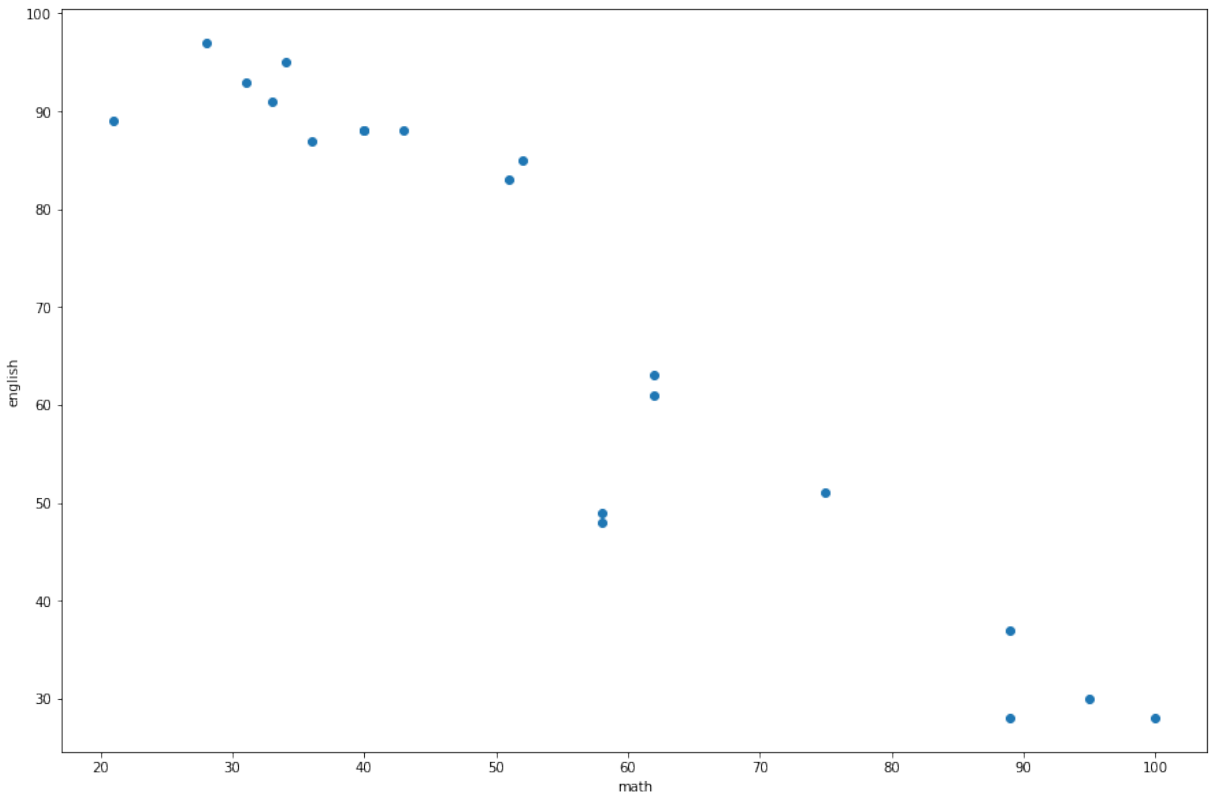
```
<matplotlib.figure.Figure at 0x22397c6de48>
```



Out[7]: 0.95168911053673177

In [8]:
```
plt.clf()
plt.figure(figsize=(15, 10))
plt.xlabel('math')
plt.ylabel('english')
plt.scatter(student['math'], student['english'])
plt.show()
# 수학과 영어과의 상관계수
np.corrcoef(student['math'], student['english'])[0, 1]
```

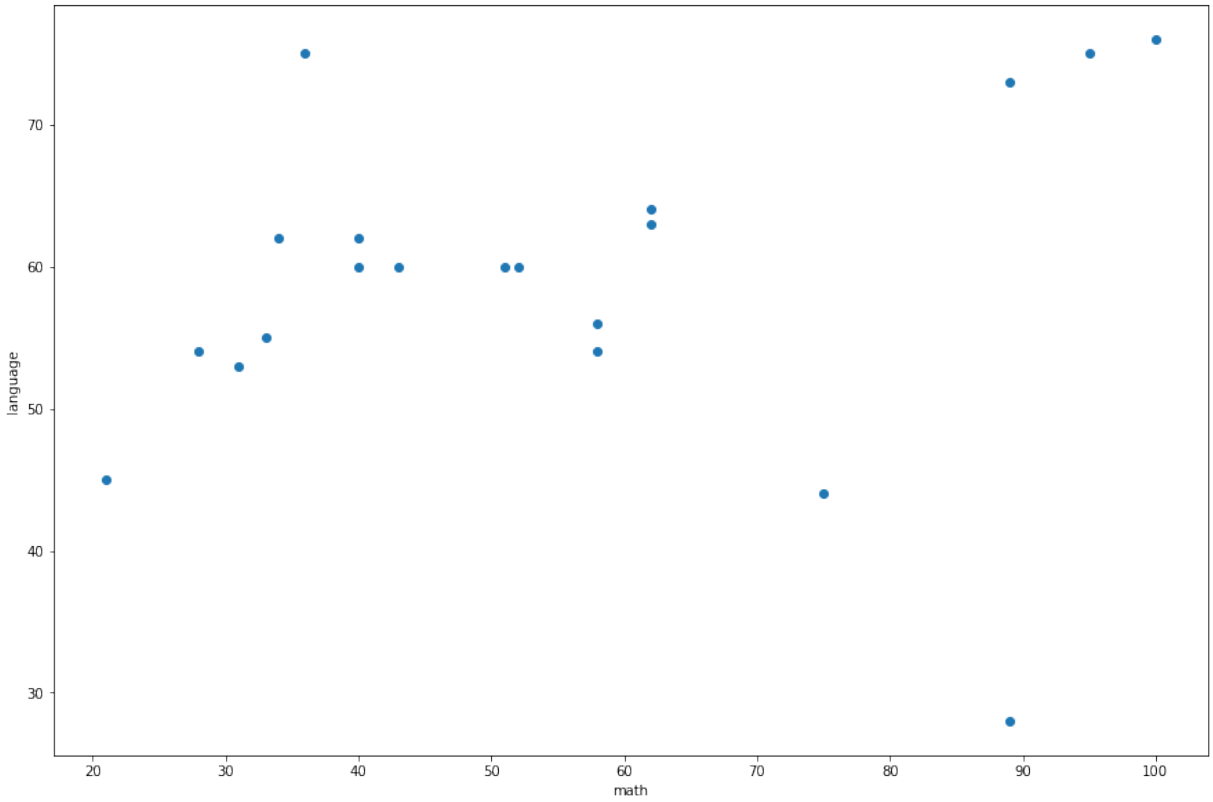
```
<matplotlib.figure.Figure at 0x22397c6d9b0>
```



Out[8]: -0.94578048089858313

In [9]:
```
plt.clf()
plt.figure(figsize=(15, 10))
plt.xlabel('math')
plt.ylabel('language')
plt.scatter(student['math'], student['language'])
plt.show()
#수학과 언어와의 상관계수
np.corrcoef(student['math'], student['language'])[0, 1]
```

```
<matplotlib.figure.Figure at 0x22397dbaa20>
```



Out[9]: 0.18321159342393156

In [10]:
```
#스피어만 상관계수
pd.concat([student, student], axis=1).corr(method='spearman')
```

Out[10]:

	language	english	math	science	language	english	math	scie
language	1.000000	-0.316428	0.362091	0.242150	1.000000	-0.316428	0.362091	0.242
english	-0.316428	1.000000	-0.941154	-0.802339	-0.316428	1.000000	-0.941154	-0.80
math	0.362091	-0.941154	1.000000	0.878205	0.362091	-0.941154	1.000000	0.878
science	0.242150	-0.802339	0.878205	1.000000	0.242150	-0.802339	0.878205	1.000
language	1.000000	-0.316428	0.362091	0.242150	1.000000	-0.316428	0.362091	0.242
english	-0.316428	1.000000	-0.941154	-0.802339	-0.316428	1.000000	-0.941154	-0.80
math	0.362091	-0.941154	1.000000	0.878205	0.362091	-0.941154	1.000000	0.878
science	0.242150	-0.802339	0.878205	1.000000	0.242150	-0.802339	0.878205	1.000

Linear regression example 1

- 선형회귀분석을 통해 과학(science) 성적을 이용하여 수학(math)성적 예측

In [11]:
```
# 회귀분석
from sklearn import linear_model
```

In [12]:
```
X = student['science']
y = student['math']

# 모듈에 맞게 data reshape
X = X.values.reshape(len(X), 1)
y = y.values.reshape(len(X), 1)
```

In [13]:
```
# 훈련데이어, 테스트 데이터 분할
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=33)
```

In [14]:
```
y_test
```

Out[14]:
```
array([[ 75],
       [ 31],
       [ 28],
       [100],
       [ 51]], dtype=int64)
```

In [15]:
```
# 선형 회귀분석 모델 생성
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

Out[15]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
lse)
```

In [16]:
```
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % np.mean((regr.predict(X_test) - y_test) ** 2))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score (X_test, y_test))
```

```
Coefficients:
 [[ 0.81443861]]
Mean squared error: 68.13
Variance score: 0.91
```

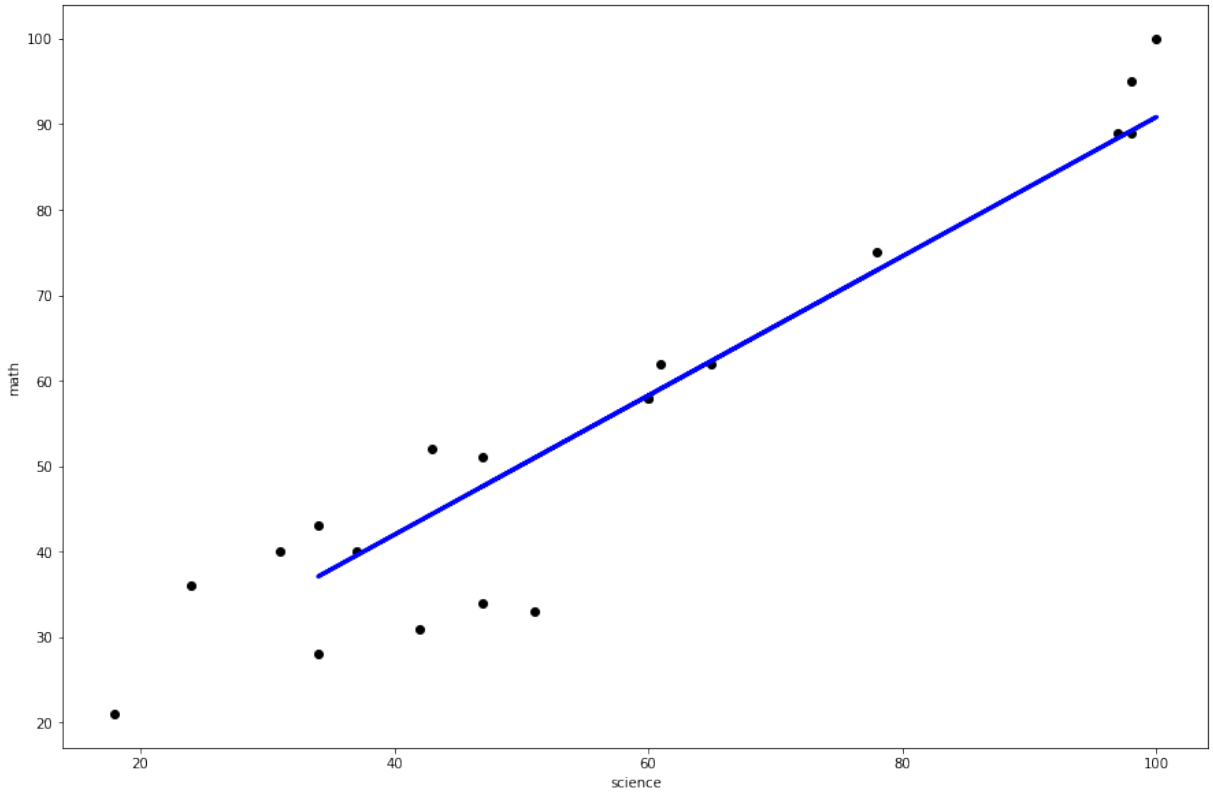
In [17]:
```
# 학생이 받은 과학성적 값(예: 50점) 입력
preds = regr.predict(X=50)
print(preds)
```

```
[[ 50.1154362]]
```

In [18]:
```
plt.clf()
plt.figure(figsize=(15, 10))
plt.xlabel('science')
plt.ylabel('math')
```

```
plt.scatter(X, y,  color='black')
plt.plot(X_test, regr.predict(X_test), color='blue', linewidth=3)
plt.show()
```

```
<matplotlib.figure.Figure at 0x22399024be0>
```



실습예제1

테스트 데이터 비율을 0.2로 수정하고 학생이 받은 과학성적이 80일때 수학성적을 예측하시오.

In [19]:
```
# 회귀분석
from sklearn import linear_model
```

In [20]:
```
X = student['science']
y = student['math']

# 모듈에 맞게 data reshape
X = X.values.reshape(len(X), 1)
y = y.values.reshape(len(X), 1)
```

In [21]:
```
# 훈련데이어, 테스트 데이터 분할
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 random_state=33)
```

In [22]:
```
# 선형 회귀분석 모델 생성
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

Out[22]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
lse)
```

In [23]:
```
# 학생이 받은 과학성적 값(예: 50점) 입력
```

```
preds = regr.predict(X=50)
print(preds)
```

```
[[ 50.33486731]]
```

Linear regression example 2

- 다중 선형회귀분석을 통해 과학(science), 언어(language), 영어(english)가 성적을 이용하여 수학(math)성적 예측

In [24]:
```
# 다중 회귀 분석
X = student[['science', 'language' ,'english']]
y = student['math']
```

In [25]:
```
# 훈련데이터, 테스트 데이터
from sklearn.model_selection import train_test_split
X.values.tolist()
y = y.values.reshape(len(X), 1) # 배열 형태 변경

# 훈련데이터, 테스트 데이터 분리(테스트 데이터 비율(0.25))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=33)
```

In [26]:
```
# 선형 회귀 분석 모델 생성
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

Out[26]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
lse)
```

In [27]:
```
# 학생이 받은 성적 값(예: 과학: 89점, 언어: 50점, 영어: 40점) 입력
table = np.array([100, 100, 50])
# 수학 성적 예측
preds = regr.predict(table.reshape(1,-1))
print(preds)
```

```
[[ 89.05862169]]
```

Linear regression example 3

- 선형회귀분석을 통해 매장을 둘러보는 횟수(view)을 이용해 매출액(payment)를 예측

In [28]:
```
# 매장 매출 예측 예
mydata2 = pd.read_csv('data/mydata2.csv')
mydata2.head(5)
```

Out[28]:

	id	walk	view	sex	payment	country
0	1	15	5	M	176	C
1	2	15	5	M	194	C
2	3	27	12	F	384	C
3	4	24	9	F	329	C
4	5	21	8	F	290	C

In [29]:
```
X = mydata2['view'] #설명변수
y = mydata2['payment'] #목적변수
```

In [30]:
```
from sklearn.model_selection import train_test_split
X = X.values.reshape(len(X), 1)
y = y.values.reshape(len(y), 1)
#훈련데이터, 테스트 데이터 분리(테스트 데이터 비율(0.25))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=33)
```

In [31]:
```
#선형 회귀분석 모델 생성
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

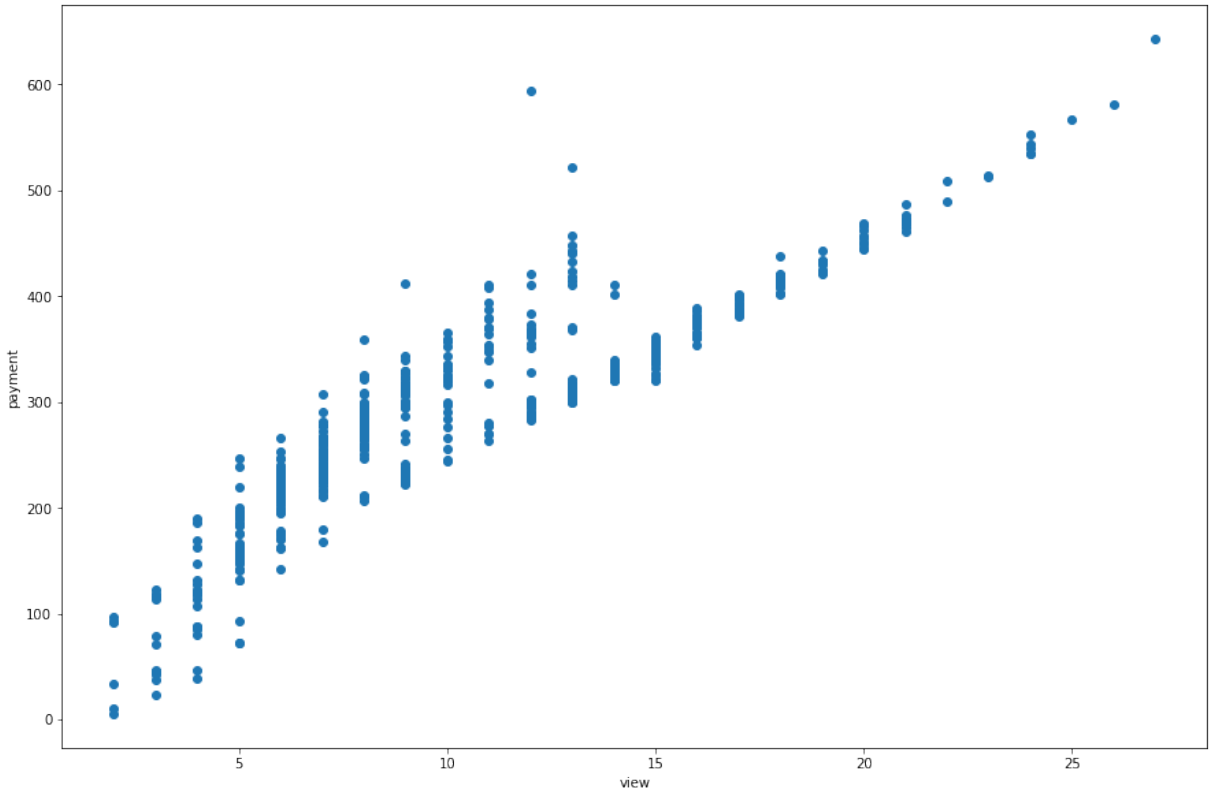
Out[31]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
lse)
```

In [32]:
```
#view가 30일때 payment 예측
preds = regr.predict(30)
print(preds)
```

```
[[ 654.37586279]]
```

In [33]:
```
plt.clf()
plt.figure(figsize=(15, 10))
plt.xlabel('view')
plt.ylabel('payment')
plt.scatter(mydata2['view'], mydata2['payment'])
plt.show()
```

```
<matplotlib.figure.Figure at 0x22399199390>
```



Linear regression example 4

- 선형 회귀분석을 통해 걷는 시간(walk)을 이용한 매출액(payment) 예측

In [34]:
```
# walk를 이용한 payment 예측
X = mydata2['walk']
y = mydata2['payment']
```

In [35]:
```
from sklearn.model_selection import train_test_split
X = X.values.reshape(len(X), 1)
y = y.values.reshape(len(y), 1)
# 훈련데이터, 테스트 데이터 분리(테스트 데이터 비율(0.25))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=33)
```

In [36]:
```
# 선형 회귀분석 모델 생성
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

Out[36]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
lse)
```

In [37]:
```
# walk가 40일때 payment 예측
preds = regr.predict(40)
print(preds)
```

```
[[ 380.87943924]]
```

In [ ]:
```

```

실습예제2

매장매출데이터(mydata2.csv)를 살펴보고 다중 선형회귀분석을 통해 걷는시간(walk), 매장을 둘러보는 횟수(view)를 이용하여 걷는시간이 30, 매장을 둘러보는 횟수가 15일때 매출액(payment)을 예측하시오

In [38]:
```
# 매장 매출 예측 예
df = pd.read_csv('data/mydata2.csv')
df.head(5)
```

Out[38]:

	id	walk	view	sex	payment	country
0	1	15	5	M	176	C
1	2	15	5	M	194	C
2	3	27	12	F	384	C
3	4	24	9	F	329	C
4	5	21	8	F	290	C

다중 회귀 분석

In [39]:
```
# 다중 회귀 분석
X = df[['walk', 'view']]
y = df['payment']
```

In [40]:
```
# 훈련데이터, 테스트 데이터
from sklearn.model_selection import train_test_split
X.values.tolist()
y = y.values.reshape(len(X), 1) # 배열 형태 변경

# 훈련데이터, 테스트 데이터 분리(테스트 데이터 비율(0.25))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=33)
```

In [41]:
```
# 선형 회귀 분석 모델 생성
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

Out[41]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
lse)
```

In [42]:
```
# 걷는시간 30, 매장을 둘러보는 횟수 15
table = np.array([30, 15])
# 매출액 예측
preds = regr.predict(table.reshape(1,-1))
print(preds)
```

```
[[ 411.98292433]]
```

Logistic regression example 1

- 타이타닉 데이터로 로지스틱 회귀분석 연습.
- 데이터를 통해 성별, 나이, 객실 등급이 승객의 생존에 어떤 영향을 끼쳤는지 분석.
- 참고.http://jy93630.blog.me/220442325857

In [1]:
```
import pandas as pd
import statsmodels.api as sm
import pylab as pl
import numpy as np
```

In [2]:
```
df = pd.read_csv("data/train2.csv") # 데이터 읽기
df.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [3]:
```
cols_to_keep = ['Survived', 'Age', 'Fare'] # 분류할 수 없는 컬럼들
```

In [4]:
```
# 분류할 수 있는 컬럼들은 더미 컬럼를 만든다.
dummy_Pclass = pd.get_dummies(df['Pclass'], prefix='Pclass')
dummy_Sex = pd.get_dummies(df['Sex'], prefix='Sex')
print(dummy_Pclass.head(5))
print(dummy_Sex.head(5))
```

```
   Pclass_1  Pclass_2  Pclass_3
0         0         0         1
```

```
1          1          0          0
2          0          0          1
3          1          0          0
4          0          0          1
   Sex_female  Sex_male
0           0         1
1           1         0
2           1         0
3           1         0
4           0         1
```

In [5]:
```
# 더미를 데이터에 이어 붙인다.
data = df[cols_to_keep].join(dummy_Pclass.ix[:,'Pclass_2':]) # Pclass_2 부터 이어 붙임. 이래야 분석에 편리함
data = data.join(dummy_Sex.ix[:,'Sex_male':]) # Sex_male만 이어 붙임
print(dummy_Sex.head(5))
```

```
   Sex_female  Sex_male
0           0         1
1           1         0
2           1         0
3           1         0
4           0         1
```

In [6]:
```
data['intercept'] = 1.0
```

In [7]:
```
# 지금까지의 데이터 확인
data.head()
```

Out[7]:

	Survived	Age	Fare	Pclass_2	Pclass_3	Sex_male	intercept
0	0	22.0	7.2500	0	1	1	1.0
1	1	38.0	71.2833	0	0	0	1.0
2	1	26.0	7.9250	0	1	0	1.0
3	1	35.0	53.1000	0	0	0	1.0
4	0	35.0	8.0500	0	1	1	1.0

In [8]:
```
# logistic regression
train_cols = data.columns[1:] # train_cols는 설명 변수
logit = sm.Logit(data['Survived'], data[train_cols]) # Survived는 목적 변수
```

In [9]:
```
# fit the model
result = logit.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.451818
         Iterations 6
```

In [10]:
```
result.summary()
```

Out[10]:

Logit Regression Results

Dep. Variable:	Survived	**No. Observations:**	891
Model:	Logit	**Df Residuals:**	885

Method:	MLE	**Df Model:**	5
Date:	Fri, 28 Jul 2017	**Pseudo R-squ.:**	0.3215
Time:	17:23:45	**Log-Likelihood:**	-402.57
converged:	True	**LL-Null:**	-593.33
		LLR p-value:	2.853e-80

	coef	std err	z	P>|z|	[95.0% Conf. Int.]
Age	-0.0330	0.007	-4.457	0.000	-0.048 -0.019
Fare	0.0007	0.002	0.340	0.734	-0.003 0.005
Pclass_2	-1.0809	0.286	-3.778	0.000	-1.642 -0.520
Pclass_3	-2.2794	0.280	-8.142	0.000	-2.828 -1.731
Sex_male	-2.6049	0.188	-13.881	0.000	-2.973 -2.237
intercept	3.4772	0.418	8.318	0.000	2.658 4.297

In [11]:
```
# odds ratios only
np.exp(result.params) # 오즈 비(Odds Ratio) 출력
```

Out[11]:
```
Age           0.967515
Fare          1.000714
Pclass_2      0.339281
Pclass_3      0.102351
Sex_male      0.073911
intercept    32.367967
dtype: float64
```

In [12]:
```
#최종결과, predict가 생존확률임.
data["predict"] = result.predict(data[train_cols])
data.head()
```

Out[12]:

	Survived	Age	Fare	Pclass_2	Pclass_3	Sex_male	intercept	predict
0	0	22.0	7.2500	0	1	1	1.0	0.106363
1	1	38.0	71.2833	0	0	0	1.0	0.906625
2	1	26.0	7.9250	0	1	0	1.0	0.585365
3	1	35.0	53.1000	0	0	0	1.0	0.913663
4	0	35.0	8.0500	0	1	1	1.0	0.071945

Logistic regression example 2

- gpa, gre, 모교 우선순위(prestige)가 대학원 입학 여부에 어떻게 영향을 끼치는가?

In [13]:
```
import pandas as pd
import statsmodels.api as sm
import pylab as pl
import numpy as np
```

In [14]:
```
df = pd.read_csv("https://stats.idre.ucla.edu/stat/data/binary.csv") # 데이터 읽기
print(df.head())
```

```
   admit  gre   gpa  rank
0      0  380  3.61     3
1      1  660  3.67     3
2      1  800  4.00     1
3      1  640  3.19     4
4      0  520  2.93     4
```

In [15]:
```
df.columns = ["admit", "gre", "gpa", "prestige"] # df의 column 이름 바꾸기
print(df.columns)
```

```
Index(['admit', 'gre', 'gpa', 'prestige'], dtype='object')
```

In [16]:
```
df.describe() # 빈도수, 평균, 분산, 최솟값, 최댓값, 1/4분위수, 중위값, 1/4분위수를 나타냄
```

Out[16]:

	admit	gre	gpa	prestige
count	400.000000	400.000000	400.000000	400.00000
mean	0.317500	587.700000	3.389900	2.48500
std	0.466087	115.516536	0.380567	0.94446
min	0.000000	220.000000	2.260000	1.00000
25%	0.000000	520.000000	3.130000	2.00000
50%	0.000000	580.000000	3.395000	2.00000
75%	1.000000	660.000000	3.670000	3.00000
max	1.000000	800.000000	4.000000	4.00000

In [17]:
```
df.std() # 분산 출력
```

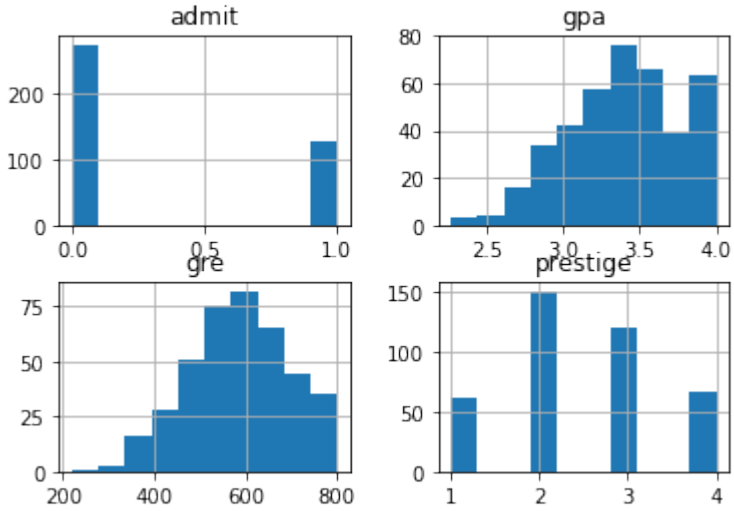
Out[17]:
```
admit         0.466087
gre         115.516536
gpa           0.380567
prestige      0.944460
dtype: float64
```

In [18]:
```
# 크로스탭 출력
pd.crosstab(df['admit'], df['prestige'], rownames=['admit'])
```

Out[18]:

prestige	1	2	3	4
admit				
0	28	97	93	55
1	33	54	28	12

In [19]:
```
df.hist()
pl.show()
# pl.show()를 해야 화면에 띄워준다! 결과는 아래와 같다.
# 모든 컬럼에 대해 히스토그램을 그림
```



In [20]:
```
# 더미코딩
dummy_ranks = pd.get_dummies(df['prestige'], prefix='prestige')
dummy_ranks.head()
```

Out[20]:

	prestige_1	prestige_2	prestige_3	prestige_4
0	0	0	1	0
1	0	0	1	0
2	1	0	0	0
3	0	0	0	1
4	0	0	0	1

In [21]:
```
#create a clean data frame for the regression
cols_to_keep = ['admit', 'gre', 'gpa']
data = df[cols_to_keep].join(dummy_ranks.ix[:, 'prestige_2':])
data.head()
```

Out[21]:

	admit	gre	gpa	prestige_2	prestige_3	prestige_4
0	0	380	3.61	0	1	0
1	1	660	3.67	0	1	0
2	1	800	4.00	0	0	0
3	1	640	3.19	0	0	1
4	0	520	2.93	0	0	1

In [22]:
```
data['intercept'] = 1.0
```

In [23]:
```
train_cols = data.columns[1:]
logit = sm.Logit(data['admit'], data[train_cols])
result = logit.fit()
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.573147
         Iterations 6
```

Out[23]:

Logit Regression Results

Dep. Variable:	admit	**No. Observations:**	400
Model:	Logit	**Df Residuals:**	394
Method:	MLE	**Df Model:**	5
Date:	Fri, 28 Jul 2017	**Pseudo R-squ.:**	0.08292
Time:	17:23:48	**Log-Likelihood:**	-229.26
converged:	True	**LL-Null:**	-249.99
		LLR p-value:	7.578e-08

	coef	std err	z	P>|z|	[95.0% Conf. Int.]
gre	0.0023	0.001	2.070	0.038	0.000 0.004
gpa	0.8040	0.332	2.423	0.015	0.154 1.454
prestige_2	-0.6754	0.316	-2.134	0.033	-1.296 -0.055
prestige_3	-1.3402	0.345	-3.881	0.000	-2.017 -0.663
prestige_4	-1.5515	0.418	-3.713	0.000	-2.370 -0.733
intercept	-3.9900	1.140	-3.500	0.000	-6.224 -1.756

결과

coef에 주목한다. gre:0.0023 gpa :0.840, prestige_2 : -0.6754 등등... coef(편회귀계수)의 값이 양수이면 그 컬럼의 값이 커질수록 목적변수가 TRUE일 확률 즉, admit=1일 확률이 높아진다. 반대로 coef의 값이 음수이면 그 컬럼의 값이 커질수록 목적변수가 FALSE일 확률 즉, admin=0일 확률이 높아진다.

GRE나 GPA가 커질수록 대학원에 입학할 확률은 커지고 prestige_2, prestige_3이 커질수록 대학원에 입학할 확률은 작아진다. 이러한 경향은 pretige가 낮아질수록 심해진다.

실습예제

로지스틱 회귀분석을 통해 오즈(odds)값을 확인하고 흑인과 백인이라는 인종정보가 사형선고를 받는데 영향을 주었는지 확인하시오

<데이터 설명>
* Agg : 범죄의 심각도 지수(1~6의 값이며 클수록 심각한 범죄)
* Vrace : 피의자의 인종(백인이면 1, 흑인이면 0)
* DEATH : 사형선고 여부

In [24]:
```
import pandas as pd
import statsmodels.api as sm
import pylab as pl
import numpy as np
import statsmodels.formula.api as smf
```

In [25]:
```
df = pd.read_csv('data/DeathPenalty.csv') # 데이터 읽기
df.head(5)
```

Out[25]:

	Agg	VRace	Death
0	1	1	1
1	1	1	1
2	1	1	0
3	1	1	0
4	1	1	0

In [26]:
```
df.columns[0:1]
```

Out[26]:
```
Index(['Agg'], dtype='object')
```

In [27]:
```
# logistic regression
train_cols = df.columns[1:2] # train_cols는 설명 변수
logit = sm.Logit(df['Death'], df[train_cols]) # Death 목적 변수
```

In [28]:
```
# fit the model
result = logit.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.675869
         Iterations 4
```

In [29]:
```
print(result.summary())
```

```
                           Logit Regression Results

==============================================================================
======
Dep. Variable:                   Death   No. Observations:
   362
Model:                           Logit   Df Residuals:
   361
Method:                            MLE   Df Model:
     0
Date:                Fri, 28 Jul 2017   Pseudo R-squ.:                  -
0.5202
Time:                        17:23:48   Log-Likelihood:                 -
244.66
converged:                       True   LL-Null:                        -
160.94
                                        LLR p-value:
   nan
==============================================================================
======
                 coef    std err          z      P>|z|      [95.0% Conf.
 Int.]
------------------------------------------------------------------------------
------
VRace         -0.6360      0.184     -3.450      0.001        -0.997
-0.275
==============================================================================
======
```

In [30]:
```
# odds ratios only
np.exp(result.params) # 오즈 비(Odds Ratio) 출력
```

Out[30]:
```
VRace    0.529412
dtype: float64
```

In [31]:
```
result.params
```

Out[31]:
```
VRace   -0.635989
dtype: float64
```

In [32]:
```
from patsy import dmatrices
import pandas as pd
from sklearn.linear_model import LogisticRegression
import statsmodels.discrete.discrete_model as sm

# sklearn output
model = LogisticRegression(fit_intercept = False, C = 1e9)
y, X = dmatrices('Death ~ Agg+VRace', df, return_type = 'dataframe')
mdl = model.fit(X, y)
print(np.exp(model.coef_))
print(model.coef_)

# sm
logit = sm.Logit(df['Death'], df[train_cols])
a = logit.fit(disp=0)
print(a.summary())
print(np.exp(a.params))
```

```
[[  1.26096112e-03   4.66292675e+00   6.11412935e+00]]
[[-6.67588106  1.53964331  1.81060238]]
                           Logit Regression Results

==============================================================================
======
Dep. Variable:                   Death   No. Observations:
   362
Model:                           Logit   Df Residuals:
   361
Method:                            MLE   Df Model:
     0
Date:                Fri, 28 Jul 2017   Pseudo R-squ.:                  -
0.5202
Time:                        17:23:48   Log-Likelihood:                -
244.66
converged:                       True   LL-Null:                       -
160.94
                                        LLR p-value:
   nan
==============================================================================
======
                 coef    std err          z      P>|z|      [95.0% Conf.
 Int.]
------------------------------------------------------------------------------
------
VRace         -0.6360      0.184     -3.450      0.001        -0.997
-0.275
==============================================================================
======
VRace    0.529412
dtype: float64
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:5
26: DataConversionWarning: A column-vector y was passed when a 1d array
```

```
was expected. Please change the shape of y to (n_samples, ), for example
 using ravel().
  y = column_or_1d(y, warn=True)
```