CSV 파일 읽고 쓰기

csv는 Comma Seperated Values의 약자로, 말 그대로 콤마(,)로 구분된 데이터들을 말합니다. 위키피디아 (https://en.wikipedia.org/wiki/Comma-separated_values)에서 예제를 확인할 수 있습니다.

In [695]:

```
# csv 파일로 바꿔봅시다.
import csv

# https://docs.python.org/2/library/csv.html#csv.reader
# delimiter, qoutechar, qouting 옵션이 뭔지 직접 해봅시다.
with open('sample.csv', 'w') as csvfile:
    # default delimiter는 , quotechar는 "입니다.
    writer = csv.writer(csvfile, delimiter=',',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)

    writer.writerow(['검사외전', '범죄', '126분'])
    # 아래와 같이 제목에 콤마(,)가 들어가면 delimiter인 콤마(,)와 헷갈리게 됩니다.
    # 그래서 quotechar(|)로 제목을 감싸주게 됩니다.
    # 직접 확인해보죠.
    writer.writerow(['쿵푸팬더3 (Kung Fu Panda 3, 2016)', '애니메이션', '95분'])
```

해보기

이번에는 delimiter를 세미콜론(';')으로 하고, quotechar는 '/'로 해봅시다. 여러줄을 한 번에 쓸때는 `writer.writerows` 하면 됩니다.

In [724]:

```
data = [['검;사;외;전', '범죄', '126분'], \
        ['쿵;푸;팬;더;3 (Kung Fu Panda 3, 2016)', \
         '애;니;메;이;션', '95분']]

def write_csv(data, file_name):
    # 해보기...
    return file_name
```

In [713]:

```
# write_csv(data,'movies_sample.csv')
```

Out[713]:

```
'movies_sample.csv'
```

In [719]:

```
def read_csv(file_name, delimiter, quotechar):
    data = []
    with open(file_name, 'r') as csvfile:
        reader = csv.reader(csvfile, delimiter=delimiter, quotechar=quotechar)
        for row in reader:
            data.append(row)
    return data
```

In [720]:

```
data = read_csv('movies_sample.csv', ';', '/')
```

In [723]:

```
for item in data:
    print item[0], item[1]
```

```
검;사;외;전 범죄
쿵;푸;팬;더;3 (Kung Fu Panda 3, 2016) 애;니;메;이;션
```

지난주에 배운 movie_total을 csv로 저장해봅시다.

In [722]:

```
from bs4 import BeautifulSoup as bs
import requests

# http://www.imdb.com/movies-coming-soon/2015-01
# 2015-01 ~ 2015-12까지 순회하며 크롤링을 할 것이므로, url 파라미터를 받는 함수를 만듭니다.
def movie_crawler(url):
    res = requests.get(url)

    # 예상치 못하게 404나 500을 받았는데, html을 파싱하려고 하면
    # 에러가 발생하므로, 200(성공)일때만 코드가 실행되도록 합니다.
    if res.status_code == 200:
        soup = bs(res.text)

        # 이렇게 css selector를 가지고 받아올 수도 있고
        movies = soup.select('#main > div > div.list.detail > div')
        # 이렇게 속성을 가지고 받아올 수도 있습니다.
        movies = soup.findAll('div', {'itemtype': 'http://schema.org/Movie'})

        # title, image, running_time, score, genre, directors, actors 순으로
        # 넣겠습니다.
        table = []
        for movie in movies:
            row = []

            title = movie.findAll('h4', {'itemprop': 'name'})
            score = movie.select('div.rating_txt > div > strong')
            genres = movie.findAll('span', {'itemprop': 'genre'})

            # strip()은 앞뒤 공백을 지워줍니다.
            # title가 빈 리스트([])인데, title[0]를 하면 index out of range
            # 에러가 납니다.
            # 에러가 나는 것을 방지해주기 위해서 뒤에 len(title)이 0보다 클때만 title[0]를
            # 하게 하고 아니면 "" 빈 스트링을 row에 append 합니다.
            row.append(title[0].text.strip() if len(title)>0 else "")
            row.append(score[0].text.strip() if len(score)>0 else "")

            # genre는 여러개 경우가 있기 때문에 list로 받아와서 join으로 연결시켜줍니다.
            row.append("/".join([genre.text for genre in genres]))

            table.append(row)

        return table

jan_2015 = movie_crawler('http://www.imdb.com/movies-coming-soon/2015-01')
print jan_2015[0]
```

```
[u'The Woman in Black 2: Angel of Death (2014)', 'http://ia.media
-imdb.com/images/M/MV5BMTgxMjUyNTAxNF5BMl5BanBnXkFtZTgwNTk4MDUyMz
E@._V1_UY209_CR0,0,140,209_AL_.jpg', u'98 min', u'42', u'Drama/Ho
rror/Thriller', u'Tom Harper', u'Helen McCrory/Jeremy Irvine/Phoe
be Fox/Leanne Best']
```

In [33]:

```
# 그러면 이제 2015-01부터 2015-12까지 영화정보를 긁어봅시다.
target_url = 'http://www.imdb.com/movies-coming-soon/{0}'
movie_total = []
for i in range(1,13):
    # string.zfill(2)을 사용해보세요. zero padding이 생깁니다.
    date = "2015-" + str(i).zfill(2)
    print target_url.format(date) + " crawling.."
    movie_total += movie_crawler(target_url.format(date))
```

```
http://www.imdb.com/movies-coming-soon/2015-01 (http://www.imdb.c
om/movies-coming-soon/2015-01) crawling..
http://www.imdb.com/movies-coming-soon/2015-02 (http://www.imdb.c
om/movies-coming-soon/2015-02) crawling..
http://www.imdb.com/movies-coming-soon/2015-03 (http://www.imdb.c
om/movies-coming-soon/2015-03) crawling..
http://www.imdb.com/movies-coming-soon/2015-04 (http://www.imdb.c
om/movies-coming-soon/2015-04) crawling..
http://www.imdb.com/movies-coming-soon/2015-05 (http://www.imdb.c
om/movies-coming-soon/2015-05) crawling..
http://www.imdb.com/movies-coming-soon/2015-06 (http://www.imdb.c
om/movies-coming-soon/2015-06) crawling..
http://www.imdb.com/movies-coming-soon/2015-07 (http://www.imdb.c
om/movies-coming-soon/2015-07) crawling..
http://www.imdb.com/movies-coming-soon/2015-08 (http://www.imdb.c
om/movies-coming-soon/2015-08) crawling..
http://www.imdb.com/movies-coming-soon/2015-09 (http://www.imdb.c
om/movies-coming-soon/2015-09) crawling..
http://www.imdb.com/movies-coming-soon/2015-10 (http://www.imdb.c
om/movies-coming-soon/2015-10) crawling..
http://www.imdb.com/movies-coming-soon/2015-11 (http://www.imdb.c
om/movies-coming-soon/2015-11) crawling..
http://www.imdb.com/movies-coming-soon/2015-12 (http://www.imdb.c
om/movies-coming-soon/2015-12) crawling..
```

In [34]:

```
with open('movie.csv', 'w') as csvfile:
    writer = csv.writer(csvfile, delimiter=',',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)

    # utf-8로 인코딩해줍니다.
    writer.writerows([[item.encode('utf-8') for item in row] for row in movie_
```

In [725]:

```
# 자, 이제 파이썬 데이터 분석툴 pandas 소개합니다.
from pandas import DataFrame
```

In [727]:

```
movie_df = DataFrame(movie_total, columns=['title', 'image', 'running time', '
```

In [729]:

```
len(movie_df)
```

Out[729]:

389

In [39]:

```
# csv format으로 저장하기 (encoding utf-8)
movie_df.to_csv('./movie_from_df.csv', encoding='utf-8', index=False, header=F
```

iPython notebook 설치하기

iPython 설치 가이드 (./ipython_notebook.pdf)

iPython notebook은 현재 보고 있는 이 웹 인터페이스를 지원하고, numpy와 pandas는 파이썬 데이터 분석에 가장 많이 활용되는 패키지입니다. 앞으로 데이터를 다루고 분석하는데 Pandas를 주로 사용하겠지만, Pandas는 numpy 자료구조에 dependency가 있습니다. numpy는 수치 연산에 강력한 ndarray 자료구조와 각종 메소드들을 지원합니다.

Pandas.DataFrame > Pandas.Series > numpy.ndarray

1. Pandas.DataFrame은 2차원 Data type. '테이블'을 생각하시면 편합니다.
2. Pandas.Series는 index가 있는 1차원 Data type. 테이블에서 '하나의 column' 혹은 '하나의 row'를 생각하시면 편합니다.
3. numpy.ndarray는 파이썬 기본 자료구조인 list보다 훨씬 파워풀한 1차원 Array.

In [730]:

```
# 일단은 Numpy, Series, DataFrame 관계부터 배워봅시다.
import pandas
import numpy
```

In [249]:

```
# Pandas DataFrame 입니다.
pd_df = pandas.DataFrame([[1,2,3],[3,4,5]])
pd_df
```

Out[249]:

	0	1	2
0	1	2	3
1	3	4	5

In [250]:

```
# Pandas Series 입니다.
pd_sr = pandas.Series([1,2,3])
pd_sr
```

Out[250]:

```
0    1
1    2
2    3
dtype: int64
```

In [268]:

```
# Numpy array 입니다.
# 중요: numpy.array 메소드로 numpy.ndarray 객체를 생성합니다.
np_ar = numpy.array([1,2,3])
np_ar
```

Out[268]:

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

In [269]:

```
# Series의 value 값이 Numpy array와 같네요.
pd_sr.values
```

Out[269]:

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

In [271]:

```
# DataFrame 첫번째 row는 Series와 같네요.
pd_df.ix[0]
```

Out[271]:

```
0    1
1    2
2    3
Name: 0, dtype: int64
```

In [277]:

```
# 그 value 값들이 Numpy array와 같네요.
pd_df.ix[0].values
```

Out[277]:

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

Pandas

In [833]:

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

In [834]:

```
# DataFrame은 Pandas에 구현되어 있는 기본 자료구조로,
# 2차원(row, column) 데이터를 표현할 수 있습니다.

# key가 column명, value 하나 하나가 row에 들어갑니다.
data = {
    'name': ['똘기', '떵이', '호치', '새촘'],
    'chinese zodiac': ['Rat', 'Pig', 'Tiger', 'Rabbit'],
    'year': ['1984','1985','1986','1987']
}

df = DataFrame(data)
df
```

Out[834]:

	chinese zodiac	name	year
0	Rat	똘기	1984
1	Pig	떵이	1985
2	Tiger	호치	1986
3	Rabbit	새촘	1987

In [845]:

```
# column 순서 바꾸기
df = DataFrame(data, columns=['name', 'year', 'chinese zodiac'])
df
```

Out[845]:

	name	year	chinese zodiac
0	똘기	1984	Rat
1	떵이	1985	Pig
2	호치	1986	Tiger
3	새촘	1987	Rabbit

In [846]:

```
# column 추가하기
df['한국 띠'] = ['쥐띠','돼지띠','호랑이띠','토끼띠']
```

In [847]:

```
df
```

Out[847]:

	name	year	chinese zodiac	한국 띠
0	똘기	1984	Rat	쥐띠
1	떵이	1985	Pig	돼지띠
2	호치	1986	Tiger	호랑이띠
3	새촘	1987	Rabbit	토끼띠

In [848]:

```
# row 추가하기
df.loc[len(df)] = ['드라곤', 1988, 'Dragon', '용띠']
```

In [849]:

```
df
```

Out[849]:

	name	year	chinese zodiac	한국 띠
0	똘기	1984	Rat	쥐띠
1	떵이	1985	Pig	돼지띠
2	호치	1986	Tiger	호랑이띠
3	새촘	1987	Rabbit	토끼띠
4	드라곤	1988	Dragon	용띠

In [851]:

```
# index(row 명)도 따로 정해줄 수 있습니다.
df = DataFrame(data, \
               columns=['name', 'year', 'chinese zodiac'], \
               index=[1,2,3,4])
df
```

Out[851]:

	name	year	chinese zodiac
1	똘기	1984	Rat
2	떵이	1985	Pig
3	호치	1986	Tiger
4	새촘	1987	Rabbit

In [852]:

```
# column명을 바꾸기 위해서는
df.columns = ['이름', '년도', '띠']
df
```

Out[852]:

	이름	년도	띠
1	똘기	1984	Rat
2	떵이	1985	Pig
3	호치	1986	Tiger
4	새촘	1987	Rabbit

In [204]:

```
# column을 선택하려면 아래와 같이 합니다.
df['이름']
```

Out[204]:

```
12-1    똘기
2       떵이
3       호치
4       새촘
Name: 이름, dtype: object
```

In [208]:

```
# Series type 입니다. Series의 정체는 뭘까요?
type(df['이름'])
```

Out[208]:

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

In [206]:

```
# row를 선택하려면 ix 메소드를 통해서 고릅니다.
# 주의: df.ix[0]은 0이 index로 지정되어있지 않기 때문에 에러가 납니다!
df.ix[1]
```

Out[206]:

```
이름         떵이
출생년도    1985
띠          Pig
Name: 2, dtype: object
```

In [209]:

```
# 이 녀석도 Series type 입니다.
type(df.ix[1])
```

Out[209]:

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

In [177]:

```
# column을 확인합니다.
df.columns
```

Out[177]:

```
Index([u'이름', u'년도', u'띠'], dtype='object')
```

In [183]:

```
# index를 확인합니다. data type이 integer네요.
df.index
```

Out[183]:

```
Int64Index([1, 2, 3, 4], dtype='int64')
```

In [185]:

```
df.index.dtype
```

Out[185]:

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

In [186]:

```
type(df.index.dtype)
```

Out[186]:

```
numpy.dtype
```

In [187]:

```
# 컬럼명 지정해서 바꾸려면 rename을 합니다.
df.rename(columns = {'년도': '출생년도'}, index = {1:'12-1'})
```

Out[187]:

	이름	출생년도	띠
12-1	똘기	1984	Rat
2	떵이	1985	Pig
3	호치	1986	Tiger
4	새촘	1987	Rabbit

In [188]:

```
# 실제로는 바뀌지 않았습니다. 바꿔주려면 inplace=True 옵션을 줍니다.
df
```

Out[188]:

	이름	년도	띠
1	똘기	1984	Rat
2	떵이	1985	Pig
3	호치	1986	Tiger
4	새촘	1987	Rabbit

In [190]:

```
# index도 아직 integer 타입이죠?
df.index
```

Out[190]:

```
Int64Index([1, 2, 3, 4], dtype='int64')
```

In [210]:

```
# 컬럼명 지정해서 바꾸려면 rename을 합니다.
df.rename(columns = {'년도': '출생년도'}, index = {1:'12-1'}, inplace=True)
```

In [211]:

```
df
```

Out[211]:

	이름	출생년도	띠
12-1	똘기	1984	Rat
2	떵이	1985	Pig
3	호치	1986	Tiger
4	새촘	1987	Rabbit

In [213]:

```
# type이 int64에서 object로 바뀌었네요
df.index
```

Out[213]:

```
Index([u'12-1', 2, 3, 4], dtype='object')
```

In [214]:

```
df.index.dtype
```

Out[214]:

```
dtype('O')
```

In [215]:

```
# row를 선택하려면 ix 메소드를 통해서 고릅니다.
df.ix['12-1']
```

Out[215]:

```
이름          똘기
출생년도     1984
띠           Rat
Name: 12-1, dtype: object
```

In [731]:

```
# index의 type이 object로 바뀌니 index 0에 접근이 가능하네요!
df.ix[0]
```

Out[731]:

```
filtercol    0
Name: 0, dtype: int64
```

해보기

In [755]:

```
data = [
    ['영화 제목1', '상영 시간1', '영화 감독1'],
    ['영화 제목2', '상영 시간2', '영화 감독2'],
    ['영화 제목3', '상영 시간3', '영화 감독3']
]

# 1. DataFrame에 넣고 column에 [제목, 상영 시간, 감독]이라고 지정해보세요.
# 2. index도 0,1,2가 아니라 1,2,3으로 지정해보세요.

# df = DataFrame(...)
# df
```

해보기

In [756]:

```
data = {
    '제목': ['영화 제목1', '영화 제목2', '영화 제목3'],
    '상영 시간': ['상영 시간1', '상영 시간2', '상영 시간3'],
    '감독': ['영화 감독1', '영화 감독2', '영화 감독3']
}

# 1. DataFrame으로 만들고, 컬럼을 제목->상영 시간->감독 순으로 바꿔보세요.
# 2. 컬럼명을 제목->title, 감독->director, 상영 시간->running time으로 바꿔보세요.
# 3. index를 '1관', '2관', '3관'으로 바꿔보세요.
# 4. 1관을 선택해서 출력해보세요.
```

Pandas로 데이터 읽어오기

In [278]:

```
# http://wiki.stat.ucla.edu/socr/index.php/SOCR_Data_MLB_HeightsWeights
# 위 사이트에 들어가서 표를 긁어서 복사(ctrl+c, 맥에서는 cmd+c) 합니다.
baseball_players = pandas.read_clipboard()
```

In [280]:

```
# DataFrame의 head() 메소드를 활용해 잘 붙었는지 확인해볼까요?
baseball_players.head()
```

Out[280]:

	Name	Team	Position	Height(inches)	Weight(pounds)	Age
0	Adam_Donachie	BAL	Catcher	74	180	22.99
1	Paul_Bako	BAL	Catcher	74	215	34.69
2	Ramon_Hernandez	BAL	Catcher	72	210	30.78
3	Kevin_Millar	BAL	First_Baseman	72	210	35.43
4	Chris_Gomez	BAL	First_Baseman	73	188	35.71

In [281]:

```
# 긁어온 데이터를 csv로 저장해봅시다.
baseball_players.to_csv('./baseball_player.csv', index=False)
```

In [604]:

```
# 저장한 .csv 데이터를 다시 읽어볼까요? 현재 폴더에 가서 잘 있는지 확인해봅시다.
bp_data = pandas.read_csv('./baseball_player.csv')
```

In [605]:

```
bp_data.head()
```

Out[605]:

	Name	Team	Position	Height(inches)	Weight(pounds)	Age
0	Adam_Donachie	BAL	Catcher	74	180	22.99
1	Paul_Bako	BAL	Catcher	74	215	34.69
2	Ramon_Hernandez	BAL	Catcher	72	210	30.78
3	Kevin_Millar	BAL	First_Baseman	72	210	35.43
4	Chris_Gomez	BAL	First_Baseman	73	188	35.71

In [606]:

```
bp_data.columns
```

Out[606]:

```
Index([u'Name', u'Team', u'Position', u'Height(inches)', u'Weight
(pounds)',
       u'Age'],
      dtype='object')
```

In [607]:

```
# Series
heights = bp_data['Height(inches)']
heights.head()
```

Out[607]:

```
0    74
1    74
2    72
3    72
4    73
Name: Height(inches), dtype: int64
```

In [608]:

```
# numpy ndarray 자료구조로 value들을 리턴합니다.
heights.values
```

Out[608]:

```
array([74, 74, 72, ..., 75, 75, 73])
```

In [609]:

```
# 이렇게 index도 받아올 수 있습니다.
heights.index
```

Out[609]:

```
Int64Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,
9,
            ...
            1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032,
1033],
           dtype='int64', length=1034)
```

Numpy 부터 보고갑시다

일단 numpy에서는 아까 잠시 살펴본 ndarray 자료구조를 잘 익히면 됩니다.

In [761]:

```
# numpy로 array를 만들어봅시다.
np.array([1,2,3,4])
```

Out[761]:

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

In [762]:

```
type(np.array([1,2,3,4]))
```

Out[762]:

```
numpy.ndarray
```

In [764]:

```
# list와 비슷해보이죠?
[1,2,3,4]
```

Out[764]:

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

In [769]:

```
# list와 numpy ndarray가 뚜렷하게 차이나는 점.

# 1. ndarray는 하나의 type만 넣을 수 있음.(type coercion)
print [1,2,'3']
print np.array([1,2,'3'])
print np.array([1,2,3]).dtype
print np.array([1,2,'3']).dtype
# (numpy에서는 single data type을 사용하기에 속도가 굉장히 빠르다고 합니다.)

# 2. 기본 연산(+,*,-,+)이 다른 의미를 가짐
print [1,2,3]+[4,5,6]
print [1,2,3]*3
print np.array([1,2,3])+np.array([4,5,6])
print np.array([1,2,3])*3
```

```
[1, 2, '3']
['1' '2' '3']
int64
|S21
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
[5 7 9]
[3 6 9]
```

In [770]:

```
np1 = np.array([1,2,3])
np2 = np.array([2,2,2])
```

In [773]:

```
np1*np2
```

Out[773]:

```
array([2, 4, 6])
```

In [775]:

```
np1+np2
```

Out[775]:

```
array([3, 4, 5])
```

In [781]:

```
np1**2
```

Out[781]:

```
array([1, 4, 9])
```

In [783]:

```
np2/2
```

Out[783]:

```
array([1, 1, 1])
```

In [784]:

```
# np_heights에 농구선수들의 키를 ndarray로 가져옵시다.
np_heights = heights.values
```

In [785]:

```
np_heights[:5]
```

Out[785]:

```
array([74, 74, 72, 72, 73])
```

In [803]:

```
# 근데 농구선수 키가 왜 이렇게 작은걸까요? 인치를 센티미터로 바꿔봅시다.
inch_to_cm = 2.54
pound_to_kg = 0.453592

cm_heights = np_heights*inch_to_cm
cm_heights
```

Out[803]:

```
array([ 187.96,  187.96,  182.88, ...,  190.5 ,  190.5 ,  185.4
2])
```

In [806]:

```
# 전체 농구선수 중에..
len(cm_heights)
```

Out[806]:

```
1034
```

In [807]:

```
# 180이상이 몇명이나 있는지 찾아봅시다.
len(cm_heights[cm_heights>180])
```

Out[807]:

```
954
```

In [789]:

```
def square_array(arr):
    return arr**2

square_array(np2)
```

Out[789]:

```
array([4, 4, 4])
```

해보기

야구선수들의 bmi(body mass index)를 구해봅시다.

$$bmi = weight(kg)/height(m^2)$$

In [795]:

```
# ndarray자료구조인 weight, height를 받아 bmi를 계산해서 리턴합니다.
# kg, m^2 단위 조심하세요!
def cal_bmi(weight, height):
    bmi = 0
    # ...
    return bmi
```

In [798]:

```
# 일단 bp_data로 부터 weight를 가져오고, numpy ndarray로 만듭니다.
# pounds->kg으로 바꿔주세요.
# kg_weights = ...
```

In [792]:

```
# m_heights = ...
```

In [808]:

```
bmi = cal_bmi(kg_weights, m_heights)
```

해보기

1. bmi가 30이상인 비만 야구선수들을 찾아봅시다.
2. 20 미만인 저체중 선수들을 찾아봅시다.

In [853]:

```
# ...
```

다시 Pandas

In [857]:

```
# apply는 이런식으로 사용합니다.
def square(x):
    return x*x

Series([1,2,3]).apply(square)
```

Out[857]:

```
0    1
1    4
2    9
dtype: int64
```

In [858]:

```
def get_odd_even(x):
    if x%2==0:
        return 'Even'
    else:
        return 'Odd'

Series([1,2,3,4,5,6,7,8]).apply(get_odd_even)
```

Out[858]:

```
0     Odd
1    Even
2     Odd
3    Even
4     Odd
5    Even
6     Odd
7    Even
dtype: object
```

In [859]:

```
# bmi 컬럼을 추가하고,
bp_data['bmi'] = bmi
```

In [619]:

```
# 30초과는 fat,
# 25초과는 over weight,
# 20초과는 normal,
# 그 미만은 low weight로 표기해봅시다.
def set_status(bmi):
    if bmi > 30:
        return 'fat'
    elif bmi > 25:
        return 'over weight'
    elif bmi > 20:
        return 'normal'
    else:
        return 'low weight'

bp_data['bmi'].apply(set_status).head()
```

Out[619]:

```
0         normal
1    over weight
2    over weight
3    over weight
4         normal
Name: bmi, dtype: object
```

In [620]:

```
bp_data['status'] = bp_data['bmi'].apply(set_status)
```

In [621]:

```
bp_data.head()
```

Out[621]:

	Name	Team	Position	Height(inches)	Weight(pounds)	Age
0	Adam_Donachie	BAL	Catcher	74	180	22.99
1	Paul_Bako	BAL	Catcher	74	215	34.69
2	Ramon_Hernandez	BAL	Catcher	72	210	30.78
3	Kevin_Millar	BAL	First_Baseman	72	210	35.43
4	Chris_Gomez	BAL	First_Baseman	73	188	35.71

해보기

age_status라는 컬럼을 만들고, Age를 기준으로 35 이상은 agegroup_1, 30 이상은 agegroup_2, 25이상은 agegroup_3, 그 이하는 agegroup_4로 저장해봅시다.

In [862]:

```
def set_age_status(age):
    #...
    pass

# bp_data['age_status'].apply(set_age_status)
```

In [622]:

```
# Height(cm) 컬럼을 추가해봅시다.
def inch_to_cm(inch):
    return inch*2.54

bp_data['Height(cm)'] = bp_data['Height(inches)'].apply(inch_to_cm)
```

해보기

In [570]:

```
# Weight(kg) 컬럼을 추가해봅시다.
def pound_to_kg(pound):
    # ...
    pass

bp_data['Weight(kg)'] = 0 # ...
```

In [623]:

```
bp_data.head()
```

Out[623]:

	Name	Team	Position	Height(inches)	Weight(pounds)	Age
0	Adam_Donachie	BAL	Catcher	74	180	22.99
1	Paul_Bako	BAL	Catcher	74	215	34.69
2	Ramon_Hernandez	BAL	Catcher	72	210	30.78
3	Kevin_Millar	BAL	First_Baseman	72	210	35.43
4	Chris_Gomez	BAL	First_Baseman	73	188	35.71

In [625]:

```
# Position 별로 얼마나 있는지 살펴봅시다.
bp_data['Position'].value_counts()
```

Out[625]:

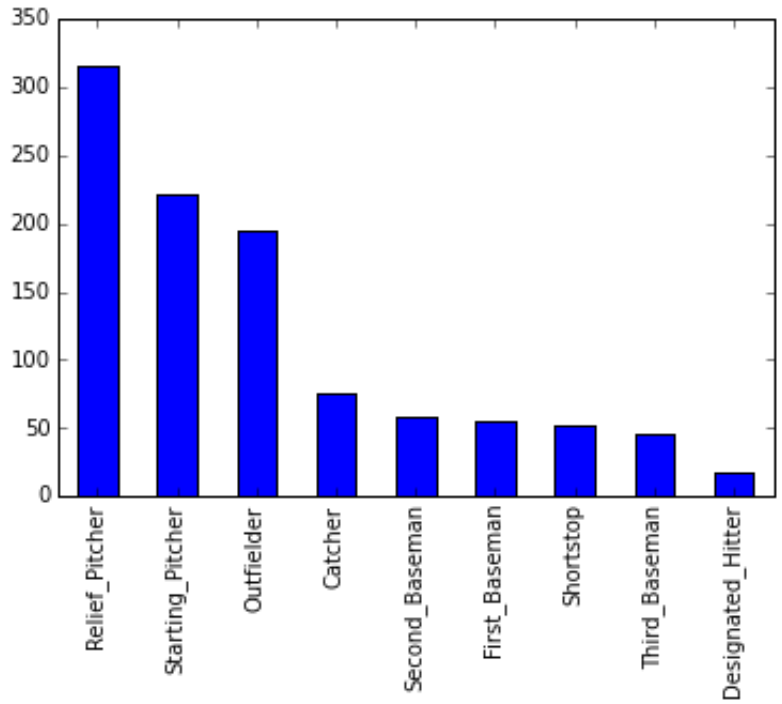
```
Relief_Pitcher       315
Starting_Pitcher     221
Outfielder           194
Catcher               76
Second_Baseman        58
First_Baseman         55
Shortstop             52
Third_Baseman         45
Designated_Hitter     18
Name: Position, dtype: int64
```

In [626]:

```
# Numeric data to plot.
bp_data['Position'].value_counts().plot(kind='bar')
```

Out[626]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x105696cd0>
```



In [627]:

```
bp_data['Age'].value_counts().head()
```

Out[627]:

```
24.94    7
27.12    6
24.63    5
31.28    5
29.86    4
Name: Age, dtype: int64
```

In [628]:

```
# 만약 값들이 continous 할 경우, 범위를 지정해서 하고 싶다면?
filter_values = [10,20,23,25,28,30,35,bp_data['Age'].max()]
filtered = pd.cut(bp_data['Age'], bins=filter_values)

# print filtered
counts = pd.value_counts(filtered)

# filtered.cat.categories
counts.reindex(filtered.cat.categories)
```

Out[628]:

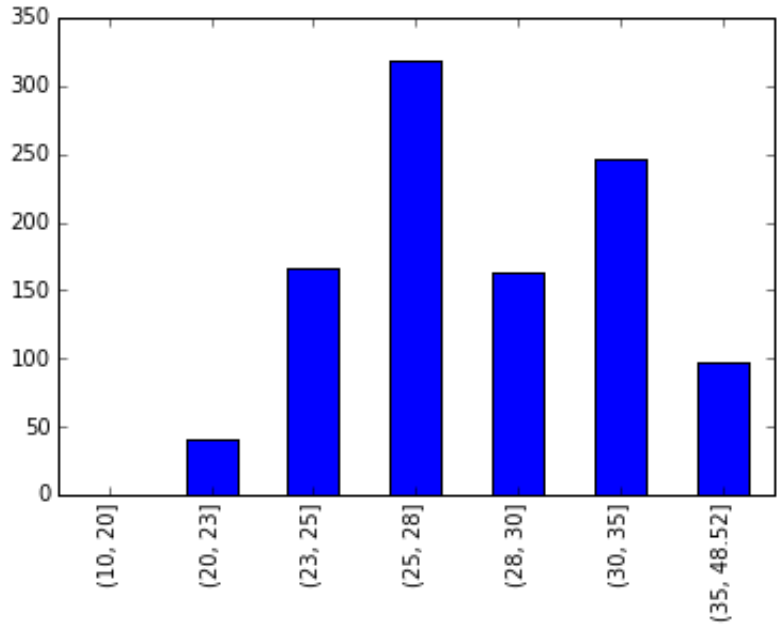
```
(10, 20]         0
(20, 23]        41
(23, 25]       167
(25, 28]       319
(28, 30]       163
(30, 35]       246
(35, 48.52]     98
dtype: int64
```

In [629]:

```
# 그래프로 보기
counts.reindex(filtered.cat.categories).plot(kind='bar')
```

Out[629]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10603c210>
```



In [866]:

```
bp_data[bp_data['status']=='normal']['Name'].head()
```

Out[866]:

```
0       Adam_Donachie
4         Chris_Gomez
10    Jeff_Fiorentino
11      Freddie_Bynum
12      Nick_Markakis
Name: Name, dtype: object
```

In [630]:

```
# normal한 체격의 30세 이하, 포수를 뽑아서 러브콜을 보내봅시다!
bp_data[(bp_data['status']=='normal') & \
        (bp_data['Age'] < 30) & (bp_data['Position'] == 'Catcher')]
```

Out[630]:

	Name	Team	Position	Height(inches)	Weight(pounds)	Age	bmi
0	Adam_Donachie	BAL	Catcher	74	180	22.99	23.1
69	Jeff_Mathis	ANA	Catcher	72	180	23.92	24.4
105	George_Kottaras	BOS	Catcher	72	180	23.79	24.4
139	Victor_Martinez	CLE	Catcher	74	190	28.19	24.3
316	Shawn_Riggans	TB	Catcher	74	190	26.60	24.3
385	Chris_Stewart	TEX	Catcher	76	205	25.03	24.9
795	Carlos_Ruiz	PHI	Catcher	72	170	28.10	23.0
898	Jesus_Flores	WAS	Catcher	73	180	22.34	23.7

In [631]:

```
# bmi 25 미만인 Starting_Pitcher나 Relief_Pitcher 상관없이 모든 Pitcher들을 데려와봅시다
bp_data[(bp_data['Position'].str.contains('Pitcher'))&(bp_data['bmi']<25)].hea
```

Out[631]:

	Name	Team	Position	Height(inches)	Weight(pounds)	Age
17	Erik_Bedard	BAL	Starting_Pitcher	73	189	27.99
18	Hayden_Penn	BAL	Starting_Pitcher	75	185	22.38
21	Steve_Trachsel	BAL	Starting_Pitcher	76	205	36.33
23	Kris_Benson	BAL	Starting_Pitcher	76	195	32.31
24	Scott_Williamson	BAL	Relief_Pitcher	72	180	31.03

In [691]:

```
# 종합 정보 보기. 25%, 50%, 75%는 사분위수(Quartiles)입니다.
bp_data.describe()
```

Out[691]:

	Height(inches)	Weight(pounds)	Age	bmi	Height(cm)
count	1034.000000	1033.000000	1034.000000	1033.000000	1034.000000
mean	73.697292	201.689255	28.736712	26.093497	187.191122
std	2.305818	20.991491	4.320310	2.300437	5.856779
min	67.000000	150.000000	20.900000	19.498447	170.180000
25%	72.000000	187.000000	25.440000	24.407276	182.880000
50%	74.000000	200.000000	27.925000	26.087904	187.960000
75%	75.000000	215.000000	31.232500	27.604061	190.500000
max	83.000000	290.000000	48.520000	35.261949	210.820000

In [692]:

```
# 테이블 정보입니다.
bp_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1034 entries, 0 to 1033
Data columns (total 9 columns):
Name               1034 non-null object
Team               1034 non-null object
Position           1034 non-null object
Height(inches)     1034 non-null int64
Weight(pounds)     1033 non-null float64
Age                1034 non-null float64
bmi                1033 non-null float64
status             1034 non-null object
Height(cm)         1034 non-null float64
dtypes: float64(4), int64(1), object(4)
memory usage: 80.8+ KB
```

In [633]:

```
bp_data['bmi'].isnull().sum()
```

Out[633]:

1

In [634]:

```
# 1034, 1033? 빈 row 찾기.
bp_data.isnull().sum()
```

Out[634]:

```
Name               0
Team               0
Position           0
Height(inches)     0
Weight(pounds)     1
Age                0
bmi                1
status             0
Height(cm)         0
dtype: int64
```

In [635]:

```
# null인 선수 찾기
bp_data[bp_data['bmi'].isnull()]
```

Out[635]:

	Name	Team	Position	Height(inches)	Weight(pounds)	Age
640	Kirk_Saarloos	CIN	Starting_Pitcher	72	NaN	27.77

In [644]:

```
# 바로 적용하기
# bp_data.drop(640, inplace=True)
bp_data_not_null = bp_data.drop(640)
```

In [662]:

```
# 640번 row가 drop 되었습니다.
bp_data_not_null.describe()
```

Out[662]:

	Height(inches)	Weight(pounds)	Age	bmi	Height(cm)
count	1033.000000	1033.000000	1033.000000	1033.000000	1033.000000
mean	73.698935	201.689255	28.737648	26.093497	187.195295
std	2.306330	20.991491	4.322298	2.300437	5.858077
min	67.000000	150.000000	20.900000	19.498447	170.180000
25%	72.000000	187.000000	25.440000	24.407276	182.880000
50%	74.000000	200.000000	27.930000	26.087904	187.960000
75%	75.000000	215.000000	31.240000	27.604061	190.500000
max	83.000000	290.000000	48.520000	35.261949	210.820000

In [649]:

```
bp_data_not_null[635:645]
```

Out[649]:

	Name	Team	Position	Height(inches)	Weight(pounds)	Ag
635	Ryan_Freel	CIN	Outfielder	70	180	30
636	Kyle_Lohse	CIN	Starting_Pitcher	74	201	28
637	Bronson_Arroyo	CIN	Starting_Pitcher	77	190	30
638	Eric_Milton	CIN	Starting_Pitcher	75	208	31
639	Aaron_Harang	CIN	Starting_Pitcher	79	240	28
641	Elizardo_Ramirez	CIN	Starting_Pitcher	72	180	24
642	Todd_Coffey	CIN	Relief_Pitcher	77	230	26
643	Brian_Shackelford	CIN	Relief_Pitcher	73	195	30
644	Bill_Bray	CIN	Relief_Pitcher	75	215	23
645	Bobby_Livingston	CIN	Relief_Pitcher	75	190	24

In [667]:

```
bp_data_not_null.reset_index(inplace=True)
```

In [669]:

```
# 다시 index를 겁니다.
bp_data_not_null[635:645]
```

Out[669]:

	level_0	index	Name	Team	Position	Height(inches)	Weig
635	635	635	Ryan_Freel	CIN	Outfielder	70	180
636	636	636	Kyle_Lohse	CIN	Starting_Pitcher	74	201
637	637	637	Bronson_Arroyo	CIN	Starting_Pitcher	77	190
638	638	638	Eric_Milton	CIN	Starting_Pitcher	75	208
639	639	639	Aaron_Harang	CIN	Starting_Pitcher	79	240
640	640	641	Elizardo_Ramirez	CIN	Starting_Pitcher	72	180
641	641	642	Todd_Coffey	CIN	Relief_Pitcher	77	230
642	642	643	Brian_Shackelford	CIN	Relief_Pitcher	73	195
643	643	644	Bill_Bray	CIN	Relief_Pitcher	75	215
644	644	645	Bobby_Livingston	CIN	Relief_Pitcher	75	190

In [683]:

```
# csv format으로 저장하기 (encoding utf-8)
movie_df = pd.read_csv('./movie_from_df.csv')
movie_df.head()
```

Out[683]:

	title	image	running time	score	genre
0	The Woman in Black 2: Angel of Death (2014)	http://ia.media-imdb.com/images/M/MV5BMTgxMjUy...	98 min	42	Drama/Horro
1	A Most Violent Year (2014)	http://ia.media-imdb.com/images/M/MV5BMjE4OTY4...	125 min	79	Action/Crime
2	Leviafan (2014)	http://ia.media-imdb.com/images/M/MV5BMjAwMTY3...	140 min	92	Drama
3	[REC] 4: Apocalipsis (2014)	http://ia.media-imdb.com/images/M/MV5BOTU3OTU2...	95 min	53	Action/Horro
4	The Search for General Tso (2014)	http://ia.media-imdb.com/images/M/MV5BODc5MzA4...	71 min	72	Documentar

In [685]:

```
movie_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 389 entries, 0 to 388
Data columns (total 7 columns):
title           389 non-null object
image           389 non-null object
running time    389 non-null object
score           373 non-null float64
genre           389 non-null object
directors       389 non-null object
actors          388 non-null object
dtypes: float64(1), object(6)
memory usage: 24.3+ KB
```

In [686]:

```
movie_df.describe()
```

Out[686]:

	score
count	373.000000
mean	58.104558
std	17.687013
min	11.000000
25%	46.000000
50%	58.000000
75%	72.000000
max	95.000000

오늘의 마지막 해보기

크롤링으로 긁어온 데이터들을 가지고, 아래의 일들을 해봅시다.

1. score를 0~50, 50~70, 70~90, 90~100 range로 나누고, comment 컬럼에 bad, so so, good, fantastic을 넣어봅시다.
2. 그래프로 그려봅시다.
3. drama 장르인 영화들을 뽑아봅시다.
4. Comedy이면서 러닝타임이 120분 이상인 것들을 골라봅시다.
5. 러닝 타임이 120분 이상인 fantastic한 Comedy를 골라봅시다.

참고

- Numpy data type에 대하여 (http://docs.scipy.org/doc/numpy-1.10.1/user/basics.types.html)
- Pandas DataFrame에 대하여 (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html)
- DataFrame에서 String은 왜 dtype이 object인가요 (http://stackoverflow.com/questions/21018654/strings-in-a-dataframe-but-dtype-is-object)
- Python and 와 &는 무엇이 다른가요 (http://stackoverflow.com/questions/22646463/difference-between-and-boolean-vs-bitwise-in-python-why-difference-i)
- 백분위수, 사분위수란? (https://docs.tibco.com/pub/spotfire_web_player/6.0.0-november-2013/ko-KR/WebHelp/GUID-10AD38BA-A888-4CEB-B716-44E2E02C8B03.html)
- DataFrame row 추가하는 법 (http://stackoverflow.com/questions/19365513/how-to-add-an-extra-row-to-a-pandas-dataframe)
- Series apply 함수 (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.apply.html)