Pandas

pandas is a Python library for data analysis. It offers a number of data exploration, cleaning and transformation operations that are critical in working with data in Python.

pandas build upon *numpy* and *scipy* providing easy-to-use data structures and data manipulation functions with integrated indexing.

The main data structures *pandas* provides are *Series* and *DataFrames*. After a brief introduction to these two data structures and data ingestion, the key features of *pandas* this notebook covers are:

- Generating descriptive statistics on data
- Data cleaning using built in pandas functions
- Frequent data operations for subsetting, filtering, insertion, deletion and aggregation of data
- Merging multiple datasets using dataframes
- Working with timestamps and time-series data

Additional Recommended Resources:

- *pandas* Documentation: http://pandas.pydata.org/pandas-docs/stable/ (http://pandas.pydata.org/pandas-docs/stable/)
- *Python for Data Analysis* by Wes McKinney
- *Python Data Science Handbook* by Jake VanderPlas

Let's get started with our first *pandas* notebook!

Import Libraries

In [1]:

```
import pandas as pd
```

Introduction to pandas Data Structures

pandas has two main data structures it uses, namely, *Series* and *DataFrames*.

pandas Series

pandas Series **one-dimensional** labeled array.

In [2]:

```
ser = pd.Series([100, 'food', 300, 'bar', 500], ['tom', 'bob', 'nancy', 'dan','eric
```

In [3]:

```
#ser = pd.Series(data = ser[100, 'food', 300, 'bar', 500], index = ['tom', 'bob', '
```

In [4]:

```
ser
```

Out[4]:

```
tom        100
bob       food
nancy      300
dan        bar
eric       500
dtype: object
```

In [5]:

```
ser.index
```

Out[5]:

```
Index(['tom', 'bob', 'nancy', 'dan', 'eric'], dtype='object')
```

In [6]:

```
ser['nancy']
```

Out[6]:

```
300
```

In [7]:

```
ser.loc['nancy'] #location
```

Out[7]:

```
300
```

In [8]:

```
ser.loc[['nancy','bob']]
```

Out[8]:

```
nancy      300
bob       food
dtype: object
```

In [9]:

```
ser[[4,3,1]]
```

Out[9]:

```
eric     500
dan      bar
bob     food
dtype: object
```

In [10]:

```
ser.iloc[2] #index location
```

Out[10]:

```
300
```

In [11]:

```
'bob' in ser
```

Out[11]:

```
True
```

In [12]:

```
ser
```

Out[12]:

```
tom       100
bob      food
nancy     300
dan       bar
eric      500
dtype: object
```

In [13]:

```
ser*2
```

Out[13]:

```
tom           200
bob      foodfood
nancy         600
dan        barbar
eric         1000
dtype: object
```

In [14]:

```
ser[['tom', 'nancy','eric']]**2
```

Out[14]:

```
tom       10000
nancy     90000
eric     250000
dtype: object
```

pandas DataFrame

pandas DataFrame is a **2-dimensional** labeled data structure.

Create DataFrame from dictionary of Python Series

In [15]:

```
d = {'one' : pd.Series([100.,200.,300.], index = ['apple', 'ball', 'clock']),
     'two' : pd.Series([111.,222.,333.,4444.], index = ['apple','ball', 'cerill', 'c
```

In [16]:

```
df = pd.DataFrame(d)
df
```

Out[16]:

	one	two
apple	100.0	111.0
ball	200.0	222.0
cerill	NaN	333.0
clock	300.0	NaN
dancy	NaN	4444.0

In [17]:

```
print(df) # print出來效果跟直接打df不一樣, 但結果是一樣
```

```
          one     two
apple   100.0   111.0
ball    200.0   222.0
cerill    NaN   333.0
clock   300.0     NaN
dancy     NaN  4444.0
```

In [18]:

```
df.index
```

Out[18]:

```
Index(['apple', 'ball', 'cerill', 'clock', 'dancy'], dtype='object')
```

In [19]:

```
df.columns
```

Out[19]:

```
Index(['one', 'two'], dtype='object')
```

In [20]:

```
pd.DataFrame(d, index = ['dancy','ball', 'apple'])
```

Out[20]:

```
         one     two
dancy    NaN  4444.0
 ball  200.0   222.0
apple  100.0   111.0
```

In [21]:

```
#如果選擇的data不在column labe的話,它還會列印出沒有的那一column, 但數值都是NaN
pd.DataFrame(d, index = ['dancy','ball','apple'], columns = ['two', 'five'])
```

Out[21]:

```
          two  five
dancy  4444.0   NaN
 ball   222.0   NaN
apple   111.0   NaN
```

Create DataFrame from list of Python dictionaries

In [22]:

```
data = [{'Alex' : 1, 'Joe' : 2}, {'Ema' : 5, 'Dora' : 10, 'Alice' : 20}] #建造一個字典
```

In [23]:

```
pd.DataFrame(data)
```

Out[23]:

```
   Alex  Alice  Dora  Ema  Joe
0   1.0    NaN   NaN  NaN  2.0
1   NaN   20.0  10.0  5.0  NaN
```

In [24]:

```
pd.DataFrame(data, index = ['orange','red']) #將 0, 1改成 orange, red
```

Out[24]:

```
        Alex  Alice  Dora  Ema  Joe
orange   1.0    NaN   NaN  NaN  2.0
   red   NaN   20.0  10.0  5.0  NaN
```

In [25]:

```
pd.DataFrame(data, columns = ['Joe','Dora','Alice']) #We will have 'only' the colum
```

Out[25]:

	Joe	Dora	Alice
0	2.0	NaN	NaN
1	NaN	10.0	20.0

Basic DataFrame operations

In [26]:

```
df
```

Out[26]:

	one	two
apple	100.0	111.0
ball	200.0	222.0
cerill	NaN	333.0
clock	300.0	NaN
dancy	NaN	4444.0

In [27]:

```
df['one']
```

Out[27]:

```
apple     100.0
ball      200.0
cerill      NaN
clock     300.0
dancy       NaN
Name: one, dtype: float64
```

In [28]:

```
df['three'] = df['one'] * df['two'] #create column ['three']
df
```

Out[28]:

	one	two	three
apple	100.0	111.0	11100.0
ball	200.0	222.0	44400.0
cerill	NaN	333.0	NaN
clock	300.0	NaN	NaN
dancy	NaN	4444.0	NaN

In [29]:

```
df['flag'] = df['one'] > 250
df
```

Out[29]:

	one	two	three	flag
apple	100.0	111.0	11100.0	False
ball	200.0	222.0	44400.0	False
cerill	NaN	333.0	NaN	False
clock	300.0	NaN	NaN	True
dancy	NaN	4444.0	NaN	False

In [30]:

```
three = df.pop('three') #df.pop() 將選擇的column指定到其他地方(variable)
```

In [31]:

```
three
```

Out[31]:

```
apple     11100.0
ball      44400.0
cerill        NaN
clock         NaN
dancy         NaN
Name: three, dtype: float64
```

In [32]:

```
df
```

Out[32]:

	one	two	flag
apple	100.0	111.0	False
ball	200.0	222.0	False
cerill	NaN	333.0	False
clock	300.0	NaN	True
dancy	NaN	4444.0	False

In [33]:

```
#刪除column
del df['two']
```

In [34]:

```
df
```

Out[34]:

	one	flag
apple	100.0	False
ball	200.0	False
cerill	NaN	False
clock	300.0	True
dancy	NaN	False

In [35]:

```
#新增column
df.insert(2, 'copy_of_one', df['one'])#第二column加入 copy_of_one
df
```

Out[35]:

	one	flag	copy_of_one
apple	100.0	False	100.0
ball	200.0	False	200.0
cerill	NaN	False	NaN
clock	300.0	True	300.0
dancy	NaN	False	NaN

In [36]:

```
df['one_upper_half'] = df['one'][:2] #新增column ['one_upper_half], 僅取one的數值第0,
```

In [37]:

```
df
```

Out[37]:

	one	flag	copy_of_one	one_upper_half
apple	100.0	False	100.0	100.0
ball	200.0	False	200.0	200.0
cerill	NaN	False	NaN	NaN
clock	300.0	True	300.0	NaN
dancy	NaN	False	NaN	NaN

Case Study: Movie Data Analysis

This notebook uses a dataset from the MovieLens website. We will describe the dataset further as we explore with it using *pandas*.

Download the Dataset

Please note that **you will need to download the dataset**. Although the video for this notebook says that the data is in your folder, the folder turned out to be too large to fit on the edX platform due to size constraints.

Here are the links to the data source and location:

- **Data Source:** MovieLens web site (filename: ml-20m.zip)
- **Location:** https://grouplens.org/datasets/movielens/ (https://grouplens.org/datasets/movielens/)

Once the download completes, please make sure the data files are in a directory called *movielens* in your *Week-3-pandas* folder.

Let us look at the files in this dataset using the UNIX command ls.

In [38]:

```
# Note: Adjust the name of the folder to match your local directory
# Let's view the contents of that MovieLens directory.
!ls ./movielens
```

```
Fun().png          genome-scores.csv movies.csv
Icon?              genome-tags.csv   ratings.csv
README.txt         links.csv         tags.csv
```

In [39]:

```
# the cat command and give movies.csv as the input to it.
!cat ./movielens/movies.csv
```

```
movieId,title,genres

1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy

2,Jumanji (1995),Adventure|Children|Fantasy

3,Grumpier Old Men (1995),Comedy|Romance

4,Waiting to Exhale (1995),Comedy|Drama|Romance

5,Father of the Bride Part II (1995),Comedy

6,Heat (1995),Action|Crime|Thriller

7,Sabrina (1995),Comedy|Romance

8,Tom and Huck (1995),Adventure|Children

9,Sudden Death (1995),Action
```

In [40]:

```
!cat ./movielens/movies.csv | wc -l #查看多少資料
```

```
   27279
```

In [41]:

```
#!head -N 查看前N筆資料
!head -5 ./movielens/movies.csv #這邊只顯示前五名的電影，因對第一row是名稱
```

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
```

In [42]:

```
!head -5 ./movielens/tags.csv
```

```
userId,movieId,tag,timestamp
18,4141,Mark Waters,1240597180
65,208,dark hero,1368150078
65,353,dark hero,1368150079
65,521,noir thriller,1368149983
```

In [43]:

```
!head -5 ./movielens/ratings.csv
```

```
userId,movieId,rating,timestamp
1,2,3.5,1112486027
1,29,3.5,1112484676
1,32,3.5,1112484819
1,47,3.5,1112484727
```

Use Pandas to Read the Dataset

In this notebook, we will be using three CSV files:

- **ratings.csv :** *userId*,*movieId*,*rating*, *timestamp*
- **tags.csv :** *userId*,*movieId*, *tag*, *timestamp*
- **movies.csv :** *movieId*, *title*, *genres*

Using the *read_csv* function in pandas, we will ingest these three files.

In [44]:

```
movies = pd.read_csv('./movielens/movies.csv', sep = ",") # we give the data file a
print(type(movies))
movies.head(15) #如果不設定，預設值為前五項
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[44]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
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
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller
10	11	American President, The (1995)	Comedy|Drama|Romance
11	12	Dracula: Dead and Loving It (1995)	Comedy|Horror
12	13	Balto (1995)	Adventure|Animation|Children
13	14	Nixon (1995)	Drama
14	15	Cutthroat Island (1995)	Action|Adventure|Romance

In [45]:

```
# Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of Ja
tags = pd.read_csv('./movielens/tags.csv', sep = ',')
tags.head()
```

Out[45]:

	userId	movieId	tag	timestamp
0	18	4141	Mark Waters	1240597180
1	65	208	dark hero	1368150078
2	65	353	dark hero	1368150079
3	65	521	noir thriller	1368149983
4	65	592	dark hero	1368150078

In [46]:

```
ratings = pd.read_csv('./movielens/ratings.csv', sep = ',', parse_dates = ['timestam
ratings.head()
```

Out[46]:

	userId	movieId	rating	timestamp
0	1	2	3.5	1112486027
1	1	29	3.5	1112484676
2	1	32	3.5	1112484819
3	1	47	3.5	1112484727
4	1	50	3.5	1112484580

In [47]:

```
# For current analysis, we will remove timestamp (we will come back to it!)

del ratings['timestamp']
del tags['timestamp']
```

Data Structures

Series

In [48]:

```
#Extract 0th row: notice that it is infact a Series
row_0 = tags.iloc[0]
type(row_0)
```

Out[48]:

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

In [49]:

```
print(row_0)
```

```
userId                18
movieId             4141
tag         Mark Waters
Name: 0, dtype: object
```

In [50]:

```
row_0.index
```

Out[50]:

```
Index(['userId', 'movieId', 'tag'], dtype='object')
```

In [51]:

```
row_0['userId']
```

Out[51]:

```
18
```

In [52]:

```
'rating' in row_0
```

Out[52]:

```
False
```

In [53]:

```
row_0.name
```

Out[53]:

```
0
```

In [54]:

```
row_0 = row_0.rename('first_row')
```

In [55]:

```
row_0
```

Out[55]:

```
userId                18
movieId             4141
tag         Mark Waters
Name: first_row, dtype: object
```

DataFrames

In [56]:

```
tags.head()
```

Out[56]:

	userId	movieId	tag
0	18	4141	Mark Waters
1	65	208	dark hero
2	65	353	dark hero
3	65	521	noir thriller
4	65	592	dark hero

In [57]:

```
tags.index
```

Out[57]:

```
RangeIndex(start=0, stop=465564, step=1)
```

In [58]:

```
tags.columns
```

Out[58]:

```
Index(['userId', 'movieId', 'tag'], dtype='object')
```

In [59]:

```
# Extract row 0, 11, 2000 from DataFrame

tags.iloc[[0,11,2000]] #取出 row = 0, 11, 2000的值
```

Out[59]:

	userId	movieId	tag
0	18	4141	Mark Waters
11	65	1783	noir thriller
2000	910	68554	conspiracy theory

Descriptive Statistics

Let's look how the ratings are distributed!

In [60]:

```
ratings['rating'].describe()
```

Out[60]:

```
count    2.000026e+07
mean     3.525529e+00
std      1.051989e+00
min      5.000000e-01
25%      3.000000e+00
50%      3.500000e+00
75%      4.000000e+00
max      5.000000e+00
Name: rating, dtype: float64
```

Count : it means standard deviation in all of those

A count of the defined or useful values in this column shows that there are more than two million ratings recorded with a mean of 3.53

50% is 3.5, which meands more than half of rating are 3.5 or less

75% is 4.0, which means 75% of the rating are below 4.0

In [61]:

```
ratings.describe()
```

Out[61]:

	userId	movieId	rating
count	2.000026e+07	2.000026e+07	2.000026e+07
mean	6.904587e+04	9.041567e+03	3.525529e+00
std	4.003863e+04	1.978948e+04	1.051989e+00
min	1.000000e+00	1.000000e+00	5.000000e-01
25%	3.439500e+04	9.020000e+02	3.000000e+00
50%	6.914100e+04	2.167000e+03	3.500000e+00
75%	1.036370e+05	4.770000e+03	4.000000e+00
max	1.384930e+05	1.312620e+05	5.000000e+00

In [62]:

```
ratings['rating'].mean()
```

Out[62]:

```
3.5255285642993797
```

In [63]:

```
ratings.mean()
```

Out[63]:

```
userId     69045.872583
movieId     9041.567330
rating         3.525529
dtype: float64
```

In [64]:

```
ratings['rating'].min()
```

Out[64]:

```
0.5
```

In [65]:

```
ratings['rating'].max()
```

Out[65]:

```
5.0
```

In [66]:

```
ratings['rating'].std() #standard deviation function
```

Out[66]:

```
1.051988919275684
```

In [67]:

```
ratings['rating'].mode() #In this case, 4.0 is the most frequent rating, 大部分評分是
```

Out[67]:

```
0    4.0
dtype: float64
```

In [68]:

```
ratings.corr() #as we know these variable can't be related
```

Out[68]:

	userId	movieId	rating
userId	1.000000	-0.000850	0.001175
movieId	-0.000850	1.000000	0.002606
rating	0.001175	0.002606	1.000000

In [69]:

```
filter_1 = ratings['rating'] > 5 #篩選出rating 大於5
filter_1.any() #只要任何一個大於5就顯示True
```

Out[69]:

```
False
```

結果得知, 沒有任何一個rating 是大於5

In [70]:

```
filter_1 = ratings['rating'] > 5
print(filter_1) #印出來每個row結果
filter_1.any()
```

```
0           False
1           False
2           False
3           False
4           False
5           False
6           False
7           False
8           False
9           False
10          False
11          False
12          False
13          False
14          False
15          False
16          False
17          False
18          False
19          False
20          False
21          False
22          False
23          False
24          False
25          False
26          False
27          False
28          False
29          False
            ...
20000233    False
20000234    False
20000235    False
20000236    False
20000237    False
20000238    False
20000239    False
20000240    False
20000241    False
20000242    False
20000243    False
20000244    False
20000245    False
20000246    False
20000247    False
20000248    False
20000249    False
20000250    False
20000251    False
20000252    False
20000253    False
20000254    False
20000255    False
20000256    False
```

```
20000257    False
20000258    False
20000259    False
20000260    False
20000261    False
20000262    False
Name: rating, Length: 20000263, dtype: bool
```

Out[70]:

```
False
```

In [71]:

```
filter_2 = ratings['rating'] > 0
print(filter_2)
filter_2.all()
```

```
0           True
1           True
2           True
3           True
4           True
5           True
6           True
7           True
8           True
9           True
10          True
11          True
12          True
13          True
14          True
15          True
16          True
17          True
18          True
19          True
20          True
21          True
22          True
23          True
24          True
25          True
26          True
27          True
28          True
29          True
            ...
20000233    True
20000234    True
20000235    True
20000236    True
20000237    True
20000238    True
20000239    True
20000240    True
20000241    True
20000242    True
20000243    True
20000244    True
20000245    True
20000246    True
20000247    True
20000248    True
20000249    True
20000250    True
20000251    True
20000252    True
20000253    True
20000254    True
20000255    True
20000256    True
```

```
20000257    True
20000258    True
20000259    True
20000260    True
20000261    True
20000262    True
Name: rating, Length: 20000263, dtype: bool
```

Out[71]:

```
True
```

Data Cleaning: Handling Missing Data

In [72]:

```
movies.shape #(rows, columns)
```

Out[72]:

```
(27278, 3)
```

In [73]:

```
# is any row NULL, 有的話顯示True

movies.isnull().any()
```

Out[73]:

```
movieId    False
title      False
genres     False
dtype: bool
```

In [74]:

```
ratings.shape
```

Out[74]:

```
(20000263, 3)
```

In [75]:

```
ratings.isnull().any()
```

Out[75]:

```
userId     False
movieId    False
rating     False
dtype: bool
```

In [76]:

```
tags.shape
```

Out[76]:

```
(465564, 3)
```

In [77]:

```
tags.isnull().any()
```

Out[77]:

```
userId     False
movieId    False
tag         True
dtype: bool
```

We have some tags which are NULL.

In [78]:

```
tags.isnull().sum() #查看miss多少
```

Out[78]:

```
userId      0
movieId     0
tag        16
dtype: int64
```

In [79]:

```
tags = tags. dropna() #axis = 0, filter out those NaN data
```

In [80]:

```
tags.isnull().any()
```

Out[80]:

```
userId     False
movieId    False
tag        False
dtype: bool
```

In [81]:

```
tags.shape
```

Out[81]:

```
(465548, 3)
```

Thats nice ! No NULL values ! Notice the number of lines have reduced.

Data Visualization

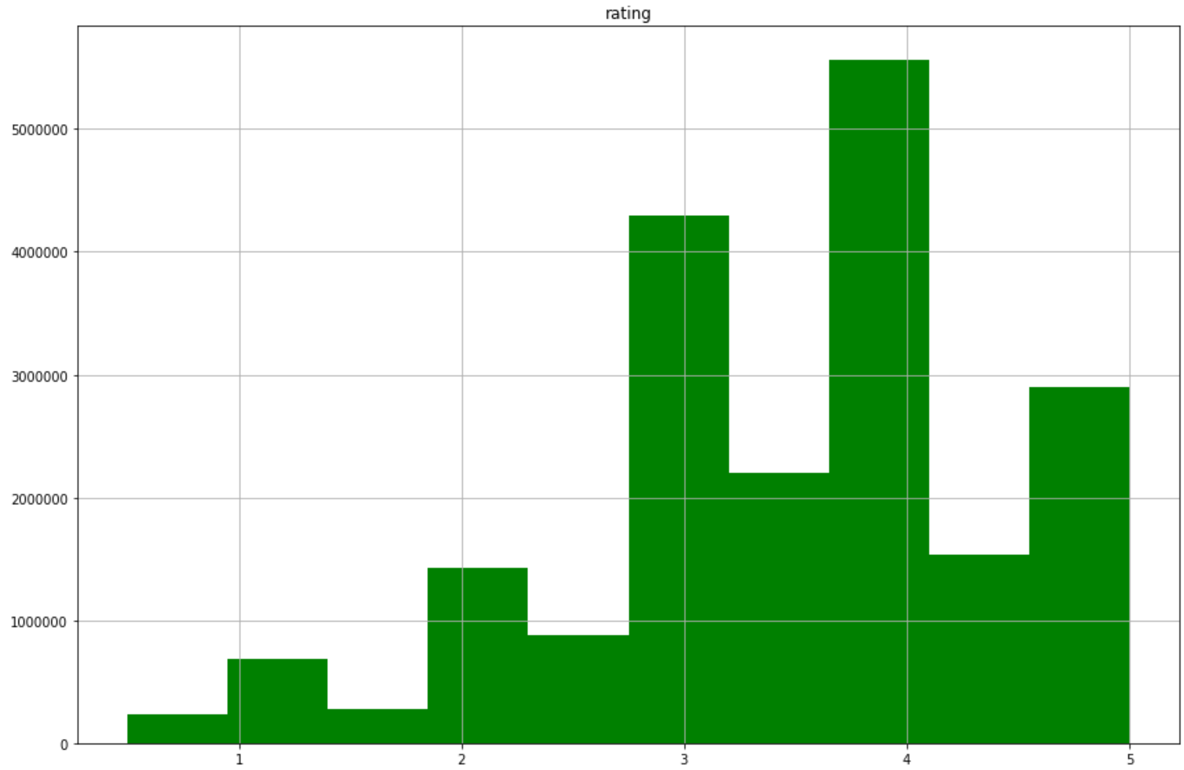
In [82]:

```
import matplotlib as plt
%matplotlib inline

ratings.hist(column = 'rating', figsize = (15,10), color = 'green')
```

Out[82]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x118691208
>]],
      dtype=object)
```

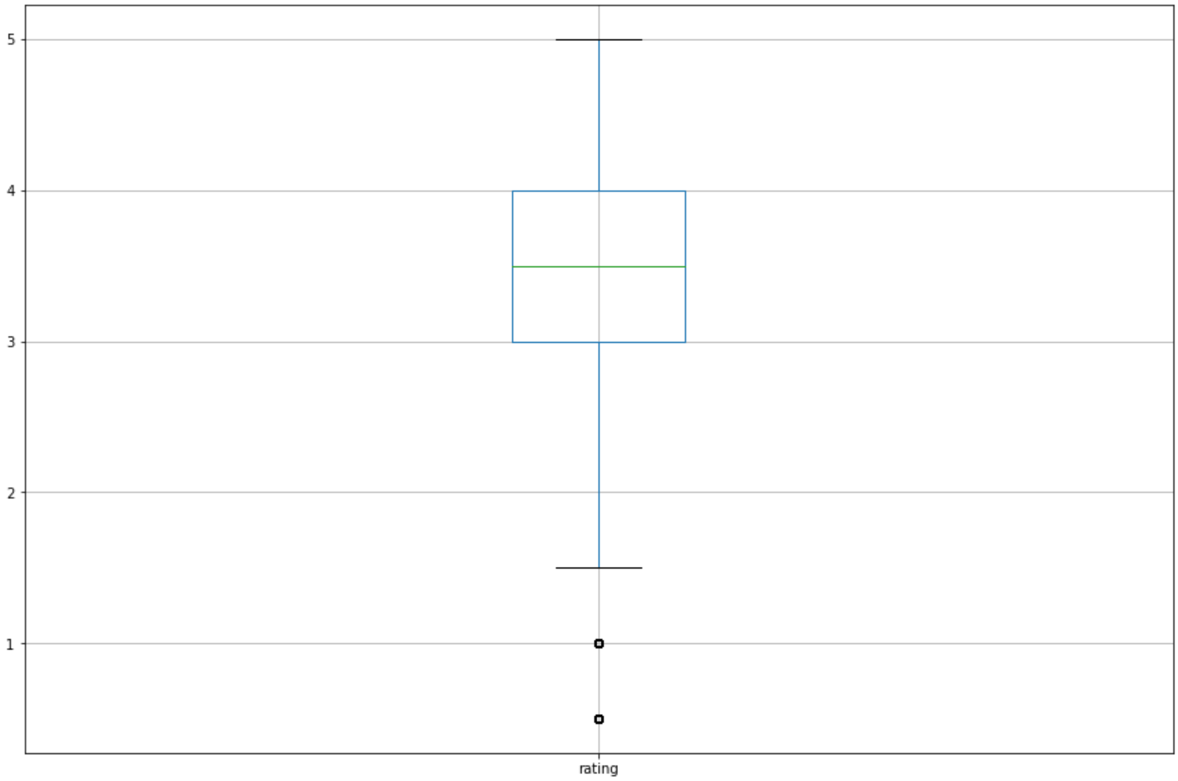


In [83]:

```
ratings.boxplot(column = 'rating', figsize = (15,10))
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1183186a0>
```

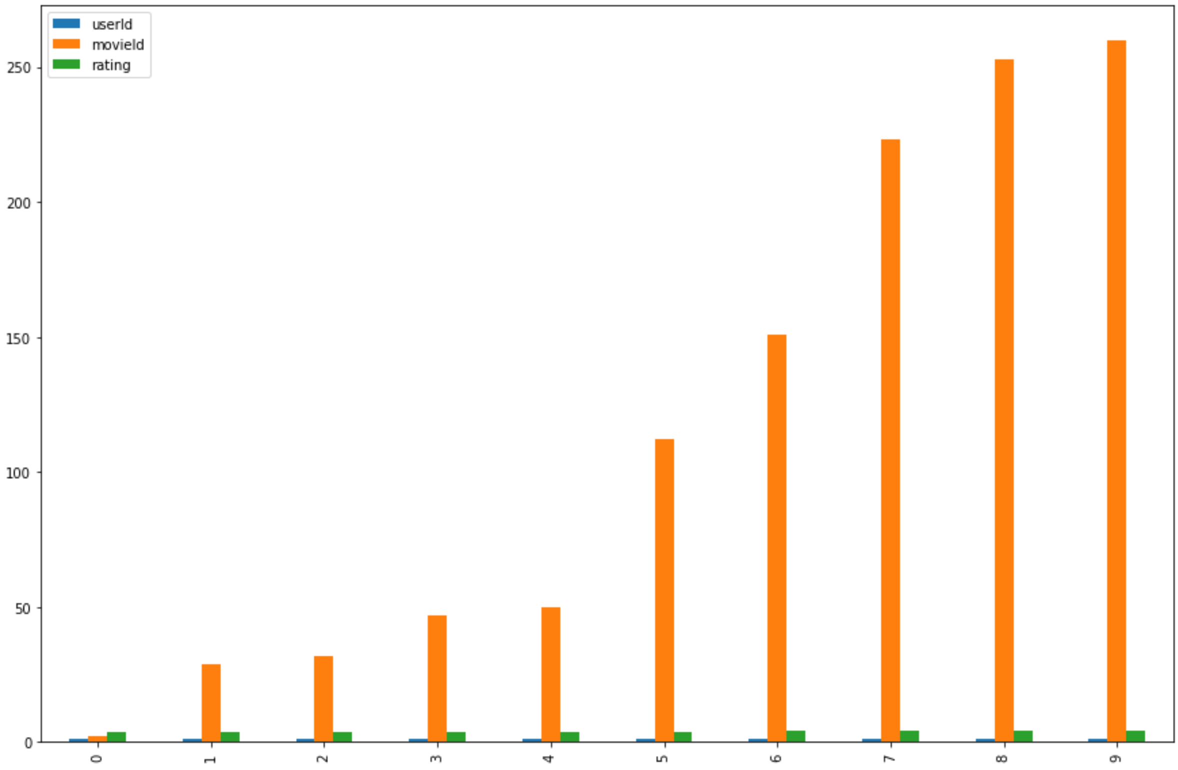


In [84]:

```
ratings[:10].plot(kind = 'bar', figsize = (15,10))
```

Out[84]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x118592860>
```



Slicing Out Columns

In [85]:

```
tags['tag'].head() # 選擇tag, 列出前五項
```

Out[85]:

```
0      Mark Waters
1        dark hero
2        dark hero
3    noir thriller
4        dark hero
Name: tag, dtype: object
```

In [86]:

```
movies[['title','genres']].head()
```

Out[86]:

	title	genres
0	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	Jumanji (1995)	Adventure|Children|Fantasy
2	Grumpier Old Men (1995)	Comedy|Romance
3	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	Father of the Bride Part II (1995)	Comedy

In [87]:

```
ratings[1000:1010] #列出1000 - 1009數值
```

Out[87]:

	userId	movieId	rating
1000	11	527	4.5
1001	11	531	4.5
1002	11	541	4.5
1003	11	546	5.0
1004	11	551	5.0
1005	11	586	4.0
1006	11	587	4.5
1007	11	588	5.0
1008	11	589	4.5
1009	11	592	4.5

In [88]:

```
ratings[:10] #列出前10
```

Out[88]:

	userId	movieId	rating
0	1	2	3.5
1	1	29	3.5
2	1	32	3.5
3	1	47	3.5
4	1	50	3.5
5	1	112	3.5
6	1	151	4.0
7	1	223	4.0
8	1	253	4.0
9	1	260	4.0

In [89]:

```
ratings[-10:] #列出倒數10名
```

Out[89]:

	userId	movieId	rating
20000253	138493	60816	4.5
20000254	138493	61160	4.0
20000255	138493	65682	4.5
20000256	138493	66762	4.5
20000257	138493	68319	4.5
20000258	138493	68954	4.5
20000259	138493	69526	4.5
20000260	138493	69644	3.0
20000261	138493	70286	5.0
20000262	138493	71619	2.5

In [90]:

```
tag_counts = tags['tag'].value_counts() #累計總數，由多到小排列
tag_counts
```

Out[90]:

```
sci-fi                                          3384
based on a book                                 3281
atmospheric                                     2917
comedy                                          2779
action                                          2657
surreal                                         2427
BD-R                                            2334
twist ending                                    2323
funny                                           2072
dystopia                                        1991
stylized                                        1941
quirky                                          1906
dark comedy                                     1899
classic                                         1769
psychology                                      1754
fantasy                                         1703
time travel                                     1549
romance                                         1534
visually appealing                              1509
disturbing                                      1487
aliens                                          1428
thought-provoking                               1422
social commentary                               1417
Nudity (Topless)                                1400
violence                                        1336
drugs                                           1312
Criterion                                       1286
true story                                      1276
nudity (topless)                                1245
adventure                                       1243
                                                ... 
PG-13:intense sci-fi action and violence           1
david vs goliath                                   1
budget                                             1
Jim Hanon                                          1
admissions officer                                 1
Jillian Schlesinger                                1
pickpocket                                         1
Annette Badland                                    1
child pornography                                  1
no spoken words                                    1
Scenes With Writer And Adult Pi                    1
setting:North Africa                               1
Macintosh                                          1
lewd larceny                                       1
not good at all                                    1
NOT DISNEY                                         1
Irene Papas                                        1
Pena de muerte                                     1
Roy Dupuis                                         1
reconstruction of lost film                        1
hindu                                              1
whisky                                             1
suburus                                            1
```

```
baby boom                                      1
snappy dialouge                                1
x-from Bro                                     1
Rose Bosch                                     1
established characters are ignored             1
good kids movie                                1
what you do not what you say                   1
Name: tag, Length: 38643, dtype: int64
```

In [91]:

```
tag_counts = tags['tag'].value_counts() #累計總數，由多到小排列
tag_counts[:10] #列前10
```

Out[91]:

```
sci-fi             3384
based on a book    3281
atmospheric        2917
comedy             2779
action             2657
surreal            2427
BD-R               2334
twist ending       2323
funny              2072
dystopia           1991
Name: tag, dtype: int64
```

In [92]:

```
tag_counts = tags['tag'].value_counts() #累計總數，由多到小排列
tag_counts[-10:] #列出倒數10名
```

Out[92]:

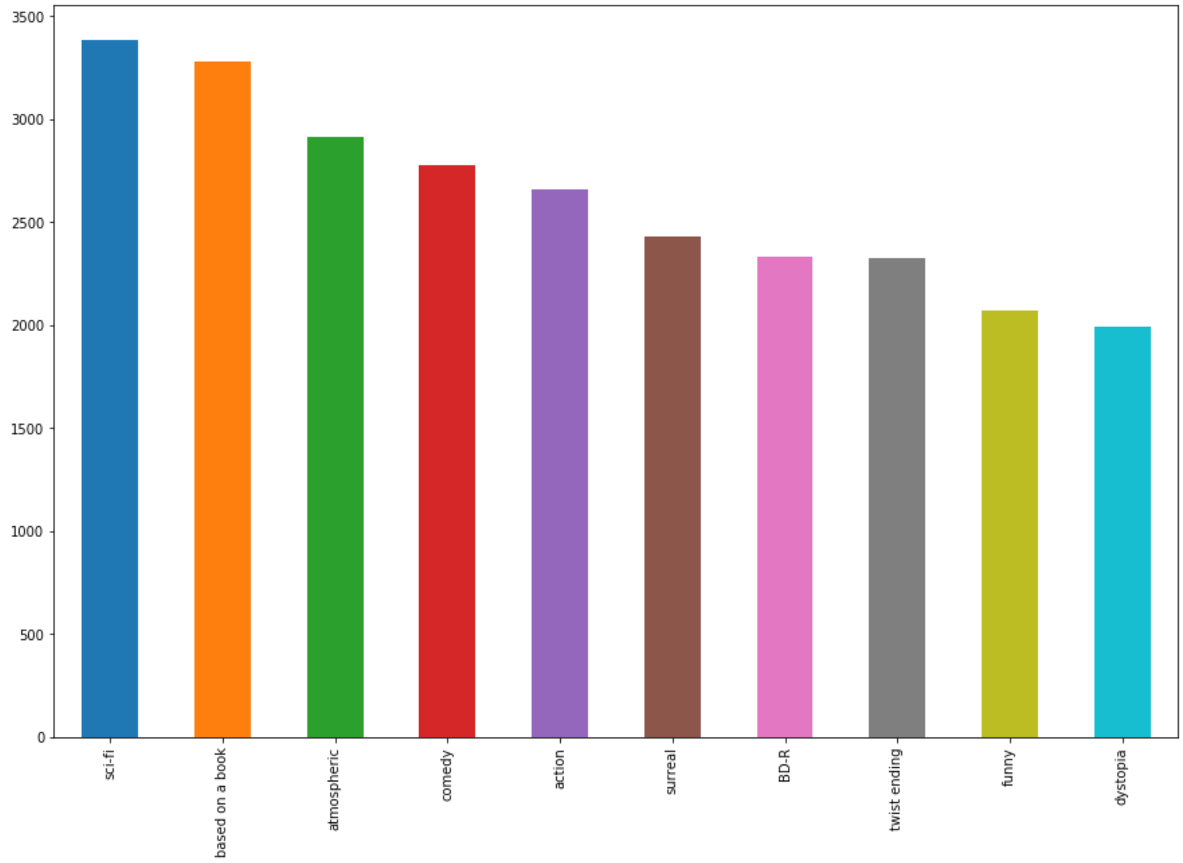
```
hindu                                 1
whisky                                1
suburus                               1
baby boom                             1
snappy dialouge                       1
x-from Bro                            1
Rose Bosch                            1
established characters are ignored    1
good kids movie                       1
what you do not what you say          1
Name: tag, dtype: int64
```

In [93]:

```
tag_counts[:10].plot(kind = 'bar', figsize = (15,10))
```

Out[93]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1162ddd30>
```

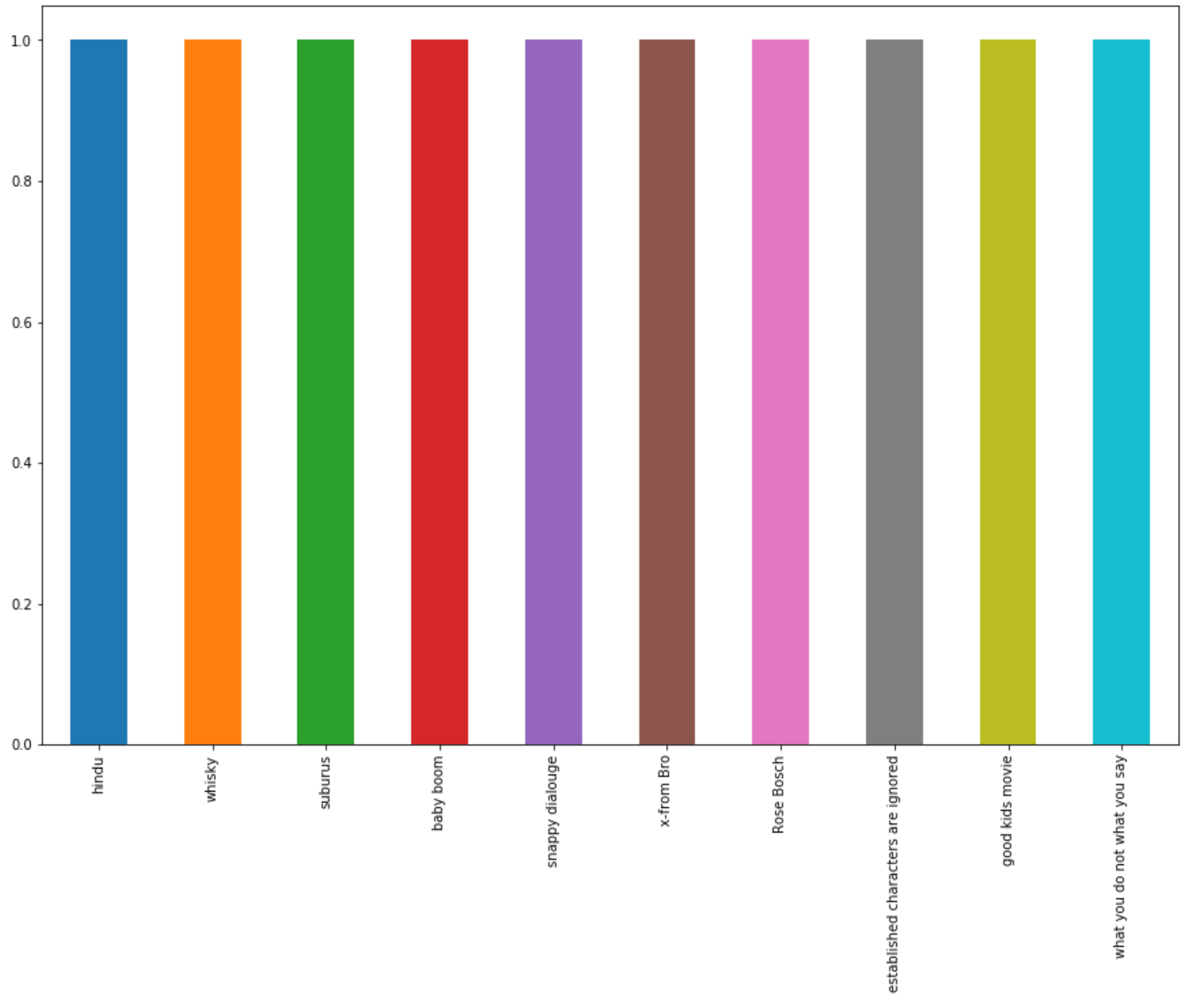


In [94]:

```
tag_counts[-10:].plot(kind = 'bar', figsize = (15,10))
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1183300b8>
```



Filters for Selecting Rows

In [95]:

```
is_hightly_rated = ratings['rating'] >= 4.0
ratings[is_hightly_rated][30:50]
```

Out[95]:

	userId	movieId	rating
68	1	2021	4.0
69	1	2100	4.0
70	1	2118	4.0
71	1	2138	4.0
72	1	2140	4.0
73	1	2143	4.0
74	1	2173	4.0
75	1	2174	4.0
76	1	2193	4.0
79	1	2288	4.0
80	1	2291	4.0
81	1	2542	4.0
82	1	2628	4.0
90	1	2762	4.0
92	1	2872	4.0
94	1	2944	4.0
96	1	2959	4.0
97	1	2968	4.0
101	1	3081	4.0
102	1	3153	4.0

In [96]:

```
is_animation = movies['genres'].str.contains('Animation') #篩選有動畫片的電影，記得用st
movies[is_animation][5:15]
```

Out[96]:

	movieId	title	genres
310	313	Swan Princess, The (1994)	Animation|Children
360	364	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical|IMAX
388	392	Secret Adventures of Tom Thumb, The (1993)	Adventure|Animation
547	551	Nightmare Before Christmas, The (1993)	Animation|Children|Fantasy|Musical
553	558	Pagemaster, The (1994)	Action|Adventure|Animation|Children|Fantasy
582	588	Aladdin (1992)	Adventure|Animation|Children|Comedy|Musical
588	594	Snow White and the Seven Dwarfs (1937)	Animation|Children|Drama|Fantasy|Musical
589	595	Beauty and the Beast (1991)	Animation|Children|Fantasy|Musical|Romance|IMAX
590	596	Pinocchio (1940)	Animation|Children|Fantasy|Musical
604	610	Heavy Metal (1981)	Action|Adventure|Animation|Horror|Sci-Fi

In [97]:

```
movies[is_animation].head(15)
```

Out[97]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
12	13	Balto (1995)	Adventure|Animation|Children
47	48	Pocahontas (1995)	Animation|Children|Drama|Musical|Romance
236	239	Goofy Movie, A (1995)	Animation|Children|Comedy|Romance
241	244	Gumby: The Movie (1995)	Animation|Children
310	313	Swan Princess, The (1994)	Animation|Children
360	364	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical|IMAX
388	392	Secret Adventures of Tom Thumb, The (1993)	Adventure|Animation
547	551	Nightmare Before Christmas, The (1993)	Animation|Children|Fantasy|Musical
553	558	Pagemaster, The (1994)	Action|Adventure|Animation|Children|Fantasy
582	588	Aladdin (1992)	Adventure|Animation|Children|Comedy|Musical
588	594	Snow White and the Seven Dwarfs (1937)	Animation|Children|Drama|Fantasy|Musical
589	595	Beauty and the Beast (1991)	Animation|Children|Fantasy|Musical|Romance|IMAX
590	596	Pinocchio (1940)	Animation|Children|Fantasy|Musical
604	610	Heavy Metal (1981)	Action|Adventure|Animation|Horror|Sci-Fi

Group By and Aggregate

In [98]:

```
ratings_count = ratings[['movieId','rating']].groupby('rating').count() #group over
ratings_count
```

Out[98]:

	movieId
rating	
0.5	239125
1.0	680732
1.5	279252
2.0	1430997
2.5	883398
3.0	4291193
3.5	2200156
4.0	5561926
4.5	1534824
5.0	2898660

In [99]:

```
average_rating = ratings[['movieId','rating']].groupby('movieId').mean()
average_rating.head() #看到每個電影的平均評價分數,列出前五項
```

Out[99]:

	rating
movieId	
1	3.921240
2	3.211977
3	3.151040
4	2.861393
5	3.064592

In [100]:

```
movie_count = ratings[['movieId','rating']].groupby('movieId').count()
movie_count.head() #可以看到每個電影的rating總數
```

Out[100]:

	rating
movieId	
1	49695
2	22243
3	12735
4	2756
5	12161

In [101]:

```
movie_count = ratings[['movieId','rating']].groupby('movieId').count()
movie_count.tail() #列出最後五項電影的rating
```

Out[101]:

	rating
movieId	
131254	1
131256	1
131258	1
131260	1
131262	1

Merge Dataframes

In [102]:

```
ratings.head()
```

Out[102]:

	userId	movieId	rating
0	1	2	3.5
1	1	29	3.5
2	1	32	3.5
3	1	47	3.5
4	1	50	3.5

In [103]:

```
tags.head()
```

Out[103]:

	userId	movieId	tag
0	18	4141	Mark Waters
1	65	208	dark hero
2	65	353	dark hero
3	65	521	noir thriller
4	65	592	dark hero

In [104]:

```
movies.head()
```

Out[104]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [105]:

```
t1 = movies.merge(tags, on = 'movieId', how = 'inner') # We are merging movie with
t1.head()
```

Out[105]:

	movieId	title	genres	userId	tag
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1644	Watched
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	computer animation
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	Disney animated feature
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	Pixar animation
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	TÃ©a Leoni does not star in this movie

In [106]:

```
t2 = movies.merge(ratings, on = 'movieId' , how = 'inner')
t2.head()
```

Out[106]:

	movieId	title	genres	userId	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3	4.0
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	6	5.0
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	8	4.0
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	10	4.0
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	11	4.5

We are merged both the rating data and the movies data all into one frame
More examples: http://pandas.pydata.org/pandas-docs/stable/merging.html (http://pandas.pydata.org/pandas-docs/stable/merging.html)

In [107]:

```
t3 = pd.concat([movies, ratings], join = 'inner', axis = 1) #把ratings合併到movies. m
t3.head()
```

Out[107]:

	movieId	title	genres	userId	movieId	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1	2	3.5
1	2	Jumanji (1995)	Adventure|Children|Fantasy	1	29	3.5
2	3	Grumpier Old Men (1995)	Comedy|Romance	1	32	3.5
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	1	47	3.5
4	5	Father of the Bride Part II (1995)	Comedy	1	50	3.5

In [108]:

```
t4 = pd.concat([movies, tags], join = 'inner', axis = 1)
t4.head()
```

Out[108]:

	movieId	title	genres	userId	movieId	tag
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	18	4141	Mark Waters
1	2	Jumanji (1995)	Adventure|Children|Fantasy	65	208	dark hero
2	3	Grumpier Old Men (1995)	Comedy|Romance	65	353	dark hero
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	65	521	noir thriller
4	5	Father of the Bride Part II (1995)	Comedy	65	592	dark hero

Combine aggreagation, merging, and filters to get useful analytics

In [109]:

```
avg_ratings = ratings.groupby('movieId', as_index = False).mean() #as_index = False,
del avg_ratings['userId']
avg_ratings.head()
```

Out[109]:

	movieId	rating
0	1	3.921240
1	2	3.211977
2	3	3.151040
3	4	2.861393
4	5	3.064592

In [110]:

```
avg_ratings = ratings.groupby('movieId', as_index = True).mean()
del avg_ratings['userId']
avg_ratings.head()
```

Out[110]:

	rating
movieId	
1	3.921240
2	3.211977
3	3.151040
4	2.861393
5	3.064592

In [111]:

```
box_office = movies.merge(avg_ratings, on = 'movieId', how = 'inner')
box_office.tail()
```

Out[111]:

	movieId	title	genres	rating
26739	131254	Kein Bund für's Leben (2007)	Comedy	4.0
26740	131256	Feuer, Eis & Dosenbier (2002)	Comedy	4.0
26741	131258	The Pirates (2014)	Adventure	2.5
26742	131260	Rentun Ruusu (2001)	(no genres listed)	3.0
26743	131262	Innocence (2014)	Adventure|Fantasy|Horror	4.0

In [112]:

```
# Filter
is_hightly_rated = box_office['rating'] >= 4.0
box_office[is_hightly_rated][-5:] # 最後五項
```

Out[112]:

	movieId	title	genres	rating
26737	131250	No More School (2000)	Comedy	4.0
26738	131252	Forklift Driver Klaus: The First Day on the Jo...	Comedy|Horror	4.0
26739	131254	Kein Bund für's Leben (2007)	Comedy	4.0
26740	131256	Feuer, Eis & Dosenbier (2002)	Comedy	4.0
26743	131262	Innocence (2014)	Adventure|Fantasy|Horror	4.0

In [113]:

```
is_comedy = box_office['genres'].str.contains('Comedy') #篩選Comedy movie

box_office[is_comedy][:5] #列出前五項
```

Out[113]:

	movieId	title	genres	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921240
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.151040
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	2.861393
4	5	Father of the Bride Part II (1995)	Comedy	3.064592
6	7	Sabrina (1995)	Comedy|Romance	3.366484

In [114]:

```
box_office[is_comedy & is_hightly_rated][-5:]
```

Out[114]:

	movieId	title	genres	rating
26736	131248	Brother Bear 2 (2006)	Adventure|Animation|Children|Comedy|Fantasy	4.0
26737	131250	No More School (2000)	Comedy	4.0
26738	131252	Forklift Driver Klaus: The First Day on the Jo...	Comedy|Horror	4.0
26739	131254	Kein Bund für's Leben (2007)	Comedy	4.0
26740	131256	Feuer, Eis & Dosenbier (2002)	Comedy	4.0

In [115]:

```
box_office[is_comedy & is_hightly_rated][:5] #列出前五的高分comedy movie
```

Out[115]:

	movieId	title	genres	rating
81	82	Antonia's Line (Antonia) (1995)	Comedy|Drama	4.004925
229	232	Eat Drink Man Woman (Yin shi nan nu) (1994)	Comedy|Drama|Romance	4.035610
293	296	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller	4.174231
352	356	Forrest Gump (1994)	Comedy|Drama|Romance|War	4.029000
602	608	Fargo (1996)	Comedy|Crime|Drama|Thriller	4.112359

In [116]:

```
is_crime = box_office['genres'].str.contains('Crime')
is_low_rated = box_office['rating'] <= 1
box_office[is_crime & is_low_rated][-5:]
```

Out[116]:

	movieId	title	genres	rating
25310	121436	The Squeeze (1987)	Action|Comedy|Crime|Romance|Thriller	1.0
25420	122799	The New Centurions (1972)	Action|Crime|Drama	1.0
25996	127068	The Castle of Fu Manchu (1969)	Action|Crime|Horror	1.0
25999	127074	Fuck Up (2012)	Comedy|Crime|Drama	1.0
26487	129873	Gardenia (1979)	Crime	0.5

Vectorized String Operations

In [117]:

```
movies.head()
```

Out[117]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Split 'genres' into multiple columns

In [118]:

```
#把genres的類別拆開
movie_genres = movies['genres'].str.split('|', expand = True) #expand = True, 這邊會
movie_genres[:10]
```

Out[118]:

	0	1	2	3	4	5	6	7	8	9
0	Adventure	Animation	Children	Comedy	Fantasy	None	None	None	None	None
1	Adventure	Children	Fantasy	None	None	None	None	None	None	None
2	Comedy	Romance	None	None	None	None	None	None	None	None
3	Comedy	Drama	Romance	None	None	None	None	None	None	None
4	Comedy	None	None	None	None	None	None	None	None	None
5	Action	Crime	Thriller	None	None	None	None	None	None	None
6	Comedy	Romance	None	None	None	None	None	None	None	None
7	Adventure	Children	None	None	None	None	None	None	None	None
8	Action	None	None	None	None	None	None	None	None	None
9	Action	Adventure	Thriller	None	None	None	None	None	None	None

Add a new column for comedy genre flag

In [119]:

```
movie_genres['isComedy'] = movies['genres'].str.contains('Comedy')
movie_genres[:10]
```

Out[119]:

	0	1	2	3	4	5	6	7	8	9	isComedy
0	Adventure	Animation	Children	Comedy	Fantasy	None	None	None	None	None	True
1	Adventure	Children	Fantasy	None	None	None	None	None	None	None	False
2	Comedy	Romance	None	None	None	None	None	None	None	None	True
3	Comedy	Drama	Romance	None	None	None	None	None	None	None	True
4	Comedy	None	None	None	None	None	None	None	None	None	True
5	Action	Crime	Thriller	None	None	None	None	None	None	None	False
6	Comedy	Romance	None	None	None	None	None	None	None	None	True
7	Adventure	Children	None	None	None	None	None	None	None	None	False
8	Action	None	None	None	None	None	None	None	None	None	False
9	Action	Adventure	Thriller	None	None	None	None	None	None	None	False

Extract year from title e.g. (1995)

In [120]:

```
movies['year'] = movies['title'].str.extract('.*\((.*)\).*', expand = True)
#.*\((.*)\).* means that extract any value and within the parentheses
#新增一column叫做year在最後面，把title中的年份拿出來
movies.tail()
```

Out[120]:

	movieId	title	genres	year
27273	131254	Kein Bund für's Leben (2007)	Comedy	2007
27274	131256	Feuer, Eis & Dosenbier (2002)	Comedy	2002
27275	131258	The Pirates (2014)	Adventure	2014
27276	131260	Rentun Ruusu (2001)	(no genres listed)	2001
27277	131262	Innocence (2014)	Adventure|Fantasy|Horror	2014

More here: http://pandas.pydata.org/pandas-docs/stable/text.html#text-string-methods (http://pandas.pydata.org/pandas-docs/stable/text.html#text-string-methods)

Parsing Timestamps

Timestamps are common in sensor data or other time series datasets. Let us revisit the **tags.csv** dataset and read the timestamps!

In [121]:

```
tags = pd.read_csv('./movielens/tags.csv' , sep = ',')
tags.dtypes
```

Out[121]:

```
userId        int64
movieId       int64
tag          object
timestamp     int64
dtype: object
```

Unix time / POSIX time / epoch time records time in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970

In [122]:

```
tags.head()
```

Out[122]:

	userId	movieId	tag	timestamp
0	18	4141	Mark Waters	1240597180
1	65	208	dark hero	1368150078
2	65	353	dark hero	1368150079
3	65	521	noir thriller	1368149983
4	65	592	dark hero	1368150078

In [123]:

```
tags['parsed_time'] = pd.to_datetime(tags['timestamp'], unit = 's')
```

Data Type datetime64[ns] maps to either M8[ns] depending on the hardware

In [124]:

```
tags['parsed_time'].dtype
```

Out[124]:

```
dtype('<M8[ns]')
```

In [125]:

```
tags.head() # parsed_time會依照 YYYY-MM-DD HH:MM:SS
```

Out[125]:

	userId	movieId	tag	timestamp	parsed_time
0	18	4141	Mark Waters	1240597180	2009-04-24 18:19:40
1	65	208	dark hero	1368150078	2013-05-10 01:41:18
2	65	353	dark hero	1368150079	2013-05-10 01:41:19
3	65	521	noir thriller	1368149983	2013-05-10 01:39:43
4	65	592	dark hero	1368150078	2013-05-10 01:41:18

Selecting rows based on timestamps

In [126]:

```
greater_than_t = tags['parsed_time'] >= '2015-02-01' # Create a filter

selected_rows = tags[greater_than_t]

tags.shape, selected_rows.shape
```

Out[126]:

```
((465564, 5), (12130, 5))
```

結論告訴我們共有12130筆資料, 是在2015-02-01後加入的

Sorting the table using the timestamps

In [127]:

```
tags.sort_values(by='parsed_time', ascending=True)[-10:] #用parsed_time來做篩選， 看後
```

Out[127]:

	userId	movieId	tag	timestamp	parsed_time
290528	88044	106782	Economically Illiterate Writers	1427753739	2015-03-30 22:15:39
290530	88044	106782	inaccurate	1427753806	2015-03-30 22:16:46
290531	88044	106782	Jonah Hill	1427753849	2015-03-30 22:17:29
290526	88044	106782	Amoral	1427753913	2015-03-30 22:18:33
290527	88044	106782	crime	1427753921	2015-03-30 22:18:41
290535	88044	106782	profanity	1427754096	2015-03-30 22:21:36
288375	87797	215	Vienna	1427755801	2015-03-30 22:50:01
158763	46072	3409	premonition	1427760726	2015-03-31 00:12:06
158780	46072	6058	premonition	1427760764	2015-03-31 00:12:44
339178	102853	115149	russian mafia	1427771352	2015-03-31 03:09:12

Average Movie Ratings over Time

Are Movie ratings related to the year of launch?

In [128]:

```
average_rating = ratings[['movieId', 'rating']].groupby('movieId', as_index = False
average_rating.tail()
```

Out[128]:

	movieId	rating
26739	131254	4.0
26740	131256	4.0
26741	131258	2.5
26742	131260	3.0
26743	131262	4.0

In [129]:

```
joined = movies.merge(average_rating, on = 'movieId', how = 'inner')
joined.head()
```

Out[129]:

	movieId	title	genres	year	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	3.921240
1	2	Jumanji (1995)	Adventure|Children|Fantasy	1995	3.211977
2	3	Grumpier Old Men (1995)	Comedy|Romance	1995	3.151040
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	1995	2.861393
4	5	Father of the Bride Part II (1995)	Comedy	1995	3.064592

In [130]:

```
joined.corr() #movieId 跟 rating的correlation
```

Out[130]:

	movieId	rating
movieId	1.000000	-0.090369
rating	-0.090369	1.000000

In [131]:

```
yearly_average = joined[['year', 'rating']].groupby('year', as_index=False).mean()
yearly_average[:10]
```

Out[131]:

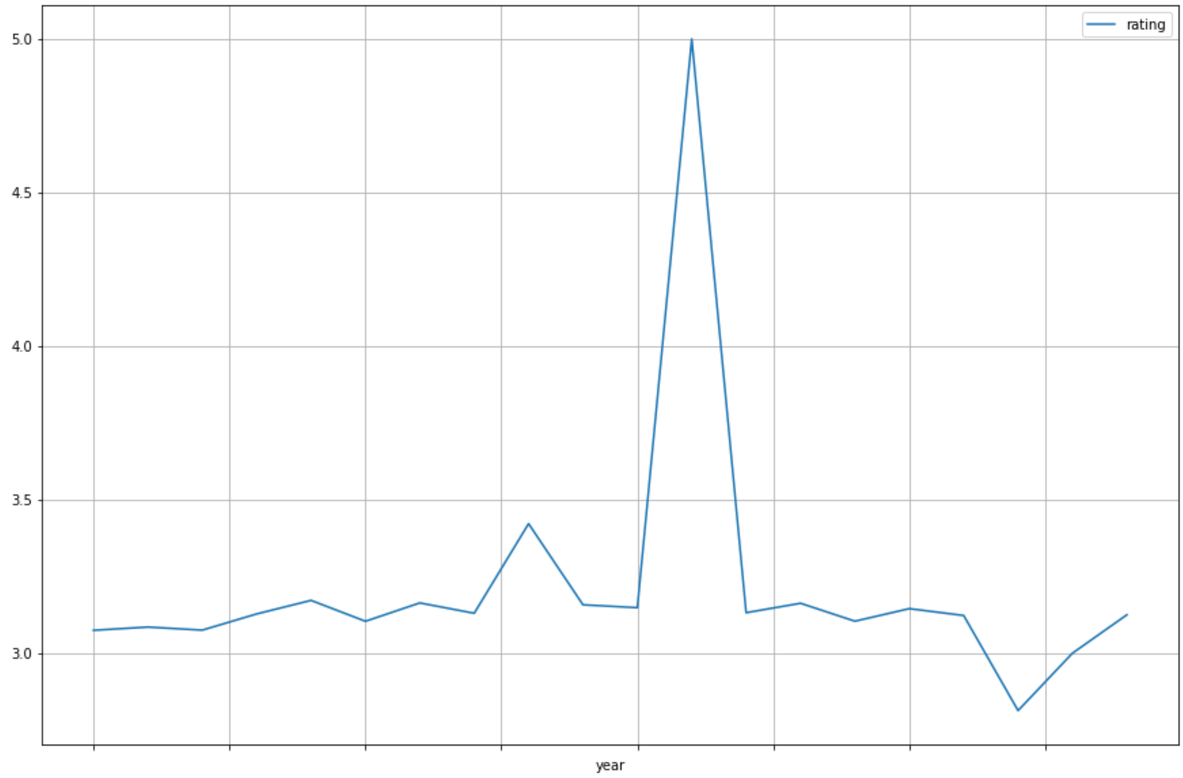
	year	rating
0	1891	3.000000
1	1893	3.375000
2	1894	3.071429
3	1895	3.125000
4	1896	3.183036
5	1898	3.850000
6	1899	3.625000
7	1900	3.166667
8	1901	5.000000
9	1902	3.738189

In [132]:

```
yearly_average[-20:].plot(x = 'year', y = 'rating', figsize=(15,10), grid = True)
```

Out[132]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11ccd9588>
```



Do some years look better for the boxoffice movies than others?

Does any data point seem like an outlier in some sense?

自我練習 merge用法

In [133]:

```
# Test 1
import pandas as pd
import numpy as np

left1 = pd.DataFrame({'key' : ['K0', 'K1', 'K2', 'K3'],
                      'A' : ['A0', 'A1', 'A2', 'A3'],
                      'B' : ['B0', 'B1', 'B2', 'B3']})
right1 = pd.DataFrame({'key' : ['K0', 'K1', 'K2', 'K3'],
                      'C' : ['C0', 'C1', 'C2', 'C3'],
                      'D' : ['D0', 'D1', 'D2', 'D3']})
```

In [134]:

```
left1
```

Out[134]:

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

In [135]:

```
right1
```

Out[135]:

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

In [136]:

```
#merge 合併, right1加入到left1
res1 = pd.merge(left1, right1, on = 'key')
res1
```

Out[136]:

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

In [137]:

```
# Test 2
left2 = pd.DataFrame({'key1' : ['K0','K1','K2','K3'],
                      'key2' : ['K0','K1','K2','K3'],
                      'A' : ['A0','A1','A2','A3'],
                      'B' : ['B0','B1','B2','B3']})
right2 = pd.DataFrame({'key1' : ['K0','K1','K2','K3'],
                      'key2' : ['K0','K1','K2','K4'],
                      'C' : ['C0','C1','C2','C3'],
                      'D' : ['D0','D1','D2','D3']})
```

In [138]:

```
left2
```

Out[138]:

	key1	key2	A	B
0	K0	K0	A0	B0
1	K1	K1	A1	B1
2	K2	K2	A2	B2
3	K3	K3	A3	B3

In [139]:

```
right2
```

Out[139]:

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K1	C1	D1
2	K2	K2	C2	D2
3	K3	K4	C3	D3

In [140]:

```
# 合并两列，默认方法是how=inner，只合并相同的部分，how的取值可以为['left', 'right', 'outer'
res2 = pd.merge(left2, right2, on = ['key1','key2'], how = 'inner') # how = inner 保
res2
```

Out[140]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K1	A1	B1	C1	D1
2	K2	K2	A2	B2	C2	D2

In [141]:

```
res2 = pd.merge(left2, right2, on = ['key1','key2'], how = 'outer') # how = outer 全
res2
```

Out[141]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K1	A1	B1	C1	D1
2	K2	K2	A2	B2	C2	D2
3	K3	K3	A3	B3	NaN	NaN
4	K3	K4	NaN	NaN	C3	D3

In [142]:

```
res2 = pd.merge(left2, right2, on = ['key1','key2'], how = 'right') # how = right 保
res2
```

Out[142]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K1	A1	B1	C1	D1
2	K2	K2	A2	B2	C2	D2
3	K3	K4	NaN	NaN	C3	D3

In [143]:

```
# Test 3
# 通过indicator表明merge的方式
res3 = pd.merge(left2, right2, on = ['key1', 'key2'], how = 'outer', indicator = Tru
res3
```

Out[143]:

	key1	key2	A	B	C	D	_merge
0	K0	K0	A0	B0	C0	D0	both
1	K1	K1	A1	B1	C1	D1	both
2	K2	K2	A2	B2	C2	D2	both
3	K3	K3	A3	B3	NaN	NaN	left_only
4	K3	K4	NaN	NaN	C3	D3	right_only

left_only : 只有左邊的row有值
right_only : 只有右邊的row有值

In [144]:

```
#修改indicator的名字
res4 = pd.merge(left2, right2, on = ['key1','key2'], how = 'outer', indicator = 'ind
res4
```

Out[144]:

	key1	key2	A	B	C	D	indicator
0	K0	K0	A0	B0	C0	D0	both
1	K1	K1	A1	B1	C1	D1	both
2	K2	K2	A2	B2	C2	D2	both
3	K3	K3	A3	B3	NaN	NaN	left_only
4	K3	K4	NaN	NaN	C3	D3	right_only

In [145]:

```
# Test 4
# 定义数据
left = pd.DataFrame({ 'A': ['A0', 'A1', 'A2', 'A3'],
                      'B': ['B0', 'B1', 'B2', 'B3']},
                      index = ['K0', 'K1', 'K2', 'K3'])
right = pd.DataFrame({'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']},
                      index = ['K0', 'K1', 'K2', 'K3'])
```

In [146]:

```
left
```

Out[146]:

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2
K3	A3	B3

In [147]:

```
right
```

Out[147]:

	C	D
K0	C0	D0
K1	C1	D1
K2	C2	D2
K3	C3	D3

In [148]:

```
#merge數據
res = pd.merge(left,right, left_index = True, right_index = True, how = 'outer')
#left_index & right_index一定要寫True, 不然無法合併數據，因為沒有common column
res
```

Out[148]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C1	D1
K2	A2	B2	C2	D2
K3	A3	B3	C3	D3

In [149]:

```
# test 5
# 定義數據
left = pd.DataFrame({'A' : ['A0', 'A1', 'A2', 'A3'],
                     'B' : ['B0', 'B1', 'B2', 'B3']})
right = pd.DataFrame({'A' : ['A0', 'A1', 'A2', 'A3'],
                      'B' : ['D0', 'D1', 'D2', 'D3']})
left
```

Out[149]:

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3

In [150]:

```
right
```

Out[150]:

	A	B
0	A0	D0
1	A1	D1
2	A2	D2
3	A3	D3

In [151]:

```
#區分兩個B
res = pd.merge(left, right, on = ['A'], how = 'inner',suffixes = ['_left', '_right'
res
```

Out[151]:

	A	B_left	B_right
0	A0	B0	D0
1	A1	B1	D1
2	A2	B2	D2
3	A3	B3	D3