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 [2]:
```
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 [4]:
```
ser = pd.Series([100, 'foo', 300, 'bar', 500], index= ['tom', 'bob', 'nancy',
'dan', 'eric'])
```

In [5]:
```
ser
```

Out[5]:
```
tom      100
bob      foo
nancy    300
dan      bar
eric     500
dtype: object
```

In [6]:
```
ser.index
```

Out[6]:
```
Index(['tom', 'bob', 'nancy', 'dan', 'eric'], dtype='object')
```

In [7]:
```
ser.loc[['nancy','bob']]
```

Out[7]:
```
nancy    300
bob      foo
dtype: object
```

In [8]:
```
ser[[4, 3, 1]]
```

Out[8]:
```
eric    500
dan     bar
bob     foo
dtype: object
```

In [9]:
```
ser.iloc[2]
```

Out[9]:
```
300
```

In [10]:
```
'bob' in ser
```

Out[10]:
```
True
```

In [11]:
```
ser
```

Out[11]:
```
tom      100
bob      foo
nancy    300
dan      bar
eric     500
dtype: object
```

In [12]:
```
ser * 2
```

Out[12]:
```
tom         200
bob      foofoo
nancy       600
dan      barbar
eric       1000
dtype: object
```

In [13]:
```
ser[['nancy', 'eric']] ** 2
```

Out[13]:
```
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 [5]:
```
d = {'one' : pd.Series([100., 200., 300.], index=['apple', 'ball', 'clock']),
     'two' : pd.Series([111., 222., 333., 4444.], index=['apple', 'ball', 'cer
ill', 'dancy'])}
```

In [17]:
```
df = pd.DataFrame(d)
df
```

Out[17]:

	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 [16]:
```
df.index
```

Out[16]:
```
Index(['apple', 'ball', 'cerill', 'clock', 'dancy'], dtype='object')
```

In [18]:
```
df.columns
```

Out[18]:
```
Index(['one', 'two'], dtype='object')
```

In [19]:
```
pd.DataFrame(d, index=['dancy', 'ball', 'apple'])
```

Out[19]:

	one	two
dancy	NaN	4444.0
ball	200.0	222.0
apple	100.0	111.0

In [20]:
```
pd.DataFrame(d, index=['dancy', 'ball', 'apple'], columns=['two', 'five'])
```

Out[20]:

	two	five
dancy	4444.0	NaN
ball	222.0	NaN
apple	111.0	NaN

Create DataFrame from list of Python dictionaries

In [21]:
```
data = [{'alex': 1, 'joe': 2}, {'ema': 5, 'dora': 10, 'alice': 20}]
```

In [22]:
```
pd.DataFrame(data)
```

Out[22]:

	alex	alice	dora	ema	joe
0	1.0	NaN	NaN	NaN	2.0
1	NaN	20.0	10.0	5.0	NaN

In [23]:
```
pd.DataFrame(data, index=['orange', 'red'])
```

Out[23]:

	alex	alice	dora	ema	joe
orange	1.0	NaN	NaN	NaN	2.0
red	NaN	20.0	10.0	5.0	NaN

In [24]:
```
pd.DataFrame(data, columns=['joe', 'dora','alice'])
```

Out[24]:

	joe	dora	alice
0	2.0	NaN	NaN
1	NaN	10.0	20.0

Basic DataFrame operations

In [25]:
```
df
```

Out[25]:

	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 [26]:
```
df['one']
```

Out[26]:
```
apple     100.0
ball      200.0
cerill      NaN
clock     300.0
dancy       NaN
Name: one, dtype: float64
```

In [27]:
```
df['three'] = df['one'] * df['two']
df
```

Out[27]:

	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 [28]:
```
df['flag'] = df['one'] > 250
df
```

Out[28]:

	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 [29]:
```
three = df.pop('three')
```

In [30]:
```
three
```

Out[30]:
```
apple     11100.0
ball      44400.0
cerill        NaN
clock         NaN
dancy         NaN
Name: three, dtype: float64
```

In [31]:
```
df
```

Out[31]:

	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 [32]:
```
del df['two']
```

In [33]:
```
df
```

Out[33]:

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

In [34]:
```
df.insert(2, 'copy_of_one', df['one'])
df
```

Out[34]:

	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 [35]:
```
df['one_upper_half'] = df['one'][:2]
df
```

Out[35]:

	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 [3]:
```
# Note: Adjust the name of the folder to match your local directory
pwd
!ls ./week-4-pandas/week-4-pandas/ml-20m
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-a34a1128ece6> in <module>()
      1 # Note: Adjust the name of the folder to match your local directory
----> 2 pwd
      3 get_ipython().system('ls ./week-4-pandas/week-4-pandas/ml-20m')

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

In [ ]:
```
!cat ./movielens/movies.csv | wc -l
```

In [ ]:
```
!head -5 ./movielens/ratings.csv
```

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 [4]:
```
movies = pd.read_csv('./ml-20m/movies.csv', sep=',')
print(type(movies))
movies.head(15)
```

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

Out[4]:

	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 [6]:
```
# Timestamps represent seconds since midnight Coordinated Universal Time (UTC)
 of January 1, 1970

tags = pd.read_csv('./ml-20m/tags.csv', sep=',')
tags.head()
```

Out[6]:

	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 [7]:
```
ratings = pd.read_csv('./ml-20m/ratings.csv', sep=',', parse_dates=['timestam
p'])
ratings.head()
```

Out[7]:

	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 [8]:
```
# For current analysis, we will remove timestamp (we will come back to it!)

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

Data Structures

Series

In [9]:
```
#Extract 0th row: notice that it is infact a Series

row_0 = tags.iloc[0]
type(row_0)
```

Out[9]: pandas.core.series.Series

In [10]:
```
print(row_0)
```

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

In [11]:
```
row_0.index
```

Out[11]: Index(['userId', 'movieId', 'tag'], dtype='object')

In [12]:
```
row_0['userId']
```

Out[12]: 18

In [13]:
```
'rating' in row_0
```

Out[13]: False

In [14]:
```
row_0.name
```

Out[14]: 0

In [15]:
```
row_0 = row_0.rename('first_row')
row_0.name
```

Out[15]: 'first_row'

DataFrames

In [16]:
```
tags.head()
```

Out[16]:

	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 [17]:
```
tags.index
```

Out[17]: RangeIndex(start=0, stop=465564, step=1)

In [18]:
```
tags.columns
```

Out[18]: Index(['userId', 'movieId', 'tag'], dtype='object')

In [19]:
```
# Extract row 0, 11, 2000 from DataFrame

tags.iloc[ [0,11,2000] ]
```

Out[19]:

	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 [20]:
```
ratings.head(5)
```

Out[20]:

	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 [21]:
```
ratings['movieId'].describe()
```

Out[21]:
```
count    2.000026e+07
mean     9.041567e+03
std      1.978948e+04
min      1.000000e+00
25%      9.020000e+02
50%      2.167000e+03
75%      4.770000e+03
max      1.312620e+05
Name: movieId, dtype: float64
```

In [22]:
```
ratings.describe()
```

Out[22]:

	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 [23]:
```
ratings['rating'].mean()
```

Out[23]:
```
3.5255285642993797
```

In [24]:
```
ratings.mean()
```

Out[24]:
```
userId     69045.872583
movieId     9041.567330
rating         3.525529
dtype: float64
```

In [25]:
```
ratings['rating'].min()
```

Out[25]:
```
0.5
```

In [26]:
```
ratings['rating'].max()
```

Out[26]:
```
5.0
```

In [27]:
```
ratings['rating'].std()
```

Out[27]:
```
1.051988919275684
```

In [28]:
```
ratings['rating'].mode()
```

Out[28]:
```
0    4.0
dtype: float64
```

In [29]:
```
ratings.corr()
```

Out[29]:

	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 [30]:
```
filter_1 = ratings['rating'] > 5
print(filter_1)
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[30]: False

In [31]:
```
filter_2 = ratings['rating'] > 0

filter_2.all()
```

Out[31]: True

Data Cleaning: Handling Missing Data

In [32]:
```
movies.shape
```

Out[32]: (27278, 3)

In [33]:
```
#is any row NULL ?

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

Out[33]:
```
movieId    False
title      False
genres     False
dtype: bool
```

Thats nice ! No NULL values !

In [34]:
```
ratings.shape
```

Out[34]: (20000263, 3)

In [35]:
```
#is any row NULL ?

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

Out[35]:
```
userId     False
movieId    False
rating     False
dtype: bool
```

Thats nice ! No NULL values !

In [36]:
```
tags.shape
```

Out[36]: (465564, 3)

In [37]:
```
#is any row NULL ?

type((tags.isnull().any()))
print (tags.isnull().any())
```

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

We have some tags which are NULL.

In [38]:
```
tags = tags.dropna()
```

In [39]:
```
#Check again: is any row NULL ?

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

Out[39]:
```
userId     False
movieId    False
tag        False
dtype: bool
```

In [40]:
```
tags.shape
```

Out[40]:
```
(465548, 3)
```

In [41]:
```
movies
```

Out[41]:

	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
15	16	Casino (1995)	Crime|Drama
16	17	Sense and Sensibility (1995)	Drama|Romance
17	18	Four Rooms (1995)	Comedy
18	19	Ace Ventura: When Nature Calls (1995)	Comedy
19	20	Money Train (1995)	Action|Comedy|Crime|Drama|Thriller
20	21	Get Shorty (1995)	Comedy|Crime|Thriller
21	22	Copycat (1995)	Crime|Drama|Horror|Mystery|Thriller
22	23	Assassins (1995)	Action|Crime|Thriller
23	24	Powder (1995)	Drama|Sci-Fi
24	25	Leaving Las Vegas (1995)	Drama|Romance
25	26	Othello (1995)	Drama
26	27	Now and Then (1995)	Children|Drama
27	28	Persuasion (1995)	Drama|Romance

	movieId	title	genres
28	29	City of Lost Children, The (Cité des enfants p...	Adventure|Drama|Fantasy|Mystery|Sci-Fi
29	30	Shanghai Triad (Yao a yao yao dao waipo qiao) ...	Crime|Drama
...	...	...	...
27248	131146	Werner - Volles Rooäää (1999)	Animation|Comedy
27249	131148	What A Man (2011)	Comedy|Romance
27250	131150	7 Dwarves: The Forest Is Not Enough (2006)	Comedy
27251	131152	The Fat Spy (1966)	Comedy
27252	131154	Die Bademeister – Weiber, saufen, Leben retten...	Comedy
27253	131156	Ants in the Pants 2 (2002)	Comedy
27254	131158	Manta, Manta (1991)	Comedy
27255	131160	Oscar and the Lady in Pink (2009)	Drama
27256	131162	Por un puñado de besos (2014)	Drama|Romance
27257	131164	Vietnam in HD (2011)	War
27258	131166	WWII IN HD (2009)	(no genres listed)
27259	131168	Phoenix (2014)	Drama
27260	131170	Parallels (2015)	Sci-Fi
27261	131172	Closed Curtain (2013)	(no genres listed)
27262	131174	Gentlemen (2014)	Drama|Romance|Thriller
27263	131176	A Second Chance (2014)	Drama
27264	131180	Dead Rising: Watchtower (2015)	Action|Horror|Thriller
27265	131231	Standby (2014)	Comedy|Romance
27266	131237	What Men Talk About (2010)	Comedy
27267	131239	Three Quarter Moon (2011)	Comedy|Drama
27268	131241	Ants in the Pants (2000)	Comedy|Romance
27269	131243	Werner - Gekotzt wird später (2003)	Animation|Comedy

	movieId	title	genres
27270	131248	Brother Bear 2 (2006)	Adventure|Animation|Children|Comedy|Fantasy
27271	131250	No More School (2000)	Comedy
27272	131252	Forklift Driver Klaus: The First Day on the Jo...	Comedy|Horror
27273	131254	Kein Bund für's Leben (2007)	Comedy
27274	131256	Feuer, Eis & Dosenbier (2002)	Comedy
27275	131258	The Pirates (2014)	Adventure
27276	131260	Rentun Ruusu (2001)	(no genres listed)
27277	131262	Innocence (2014)	Adventure|Fantasy|Horror

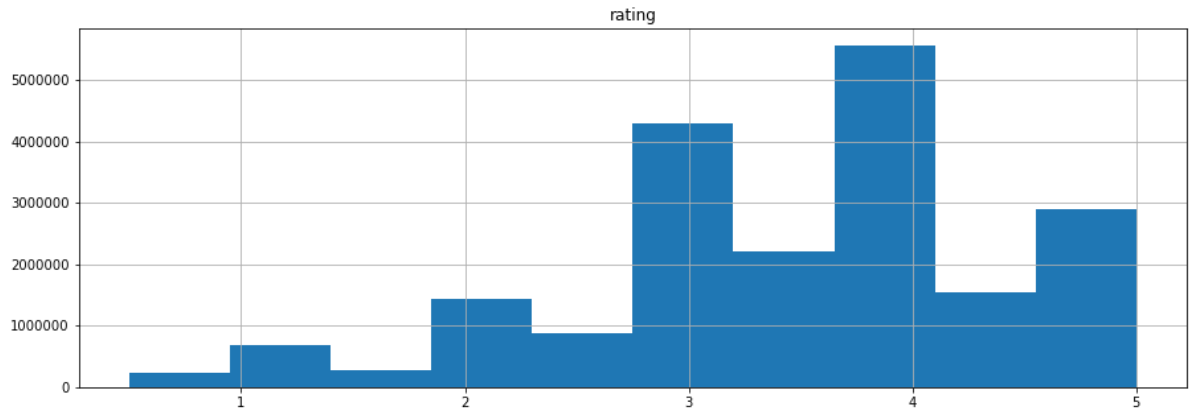
27278 rows × 3 columns

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

Data Visualization

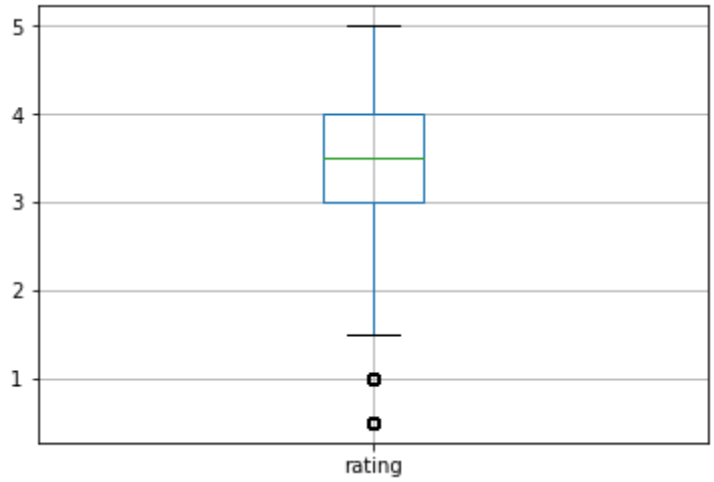
In [42]:
```
%matplotlib inline

ratings.hist(column='rating', figsize=(15,5))
```

Out[42]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001BF817D8208
>]], dtype=object)
```



In [43]:
```
ratings.boxplot(column='rating', figsize=(15,20))
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bfcd7814e0>
```



Slicing Out Columns

In [44]:
```
tags['tag'].head()
```

Out[44]:
```
0      Mark Waters
1        dark hero
2        dark hero
3    noir thriller
4        dark hero
Name: tag, dtype: object
```

In [45]:
```
movies[['title','genres']].head()
```

Out[45]:

	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 [46]:
```
ratings[ratings['userId']==11]
print("---")
ratings[-10:]
```

```
---
```

Out[46]:

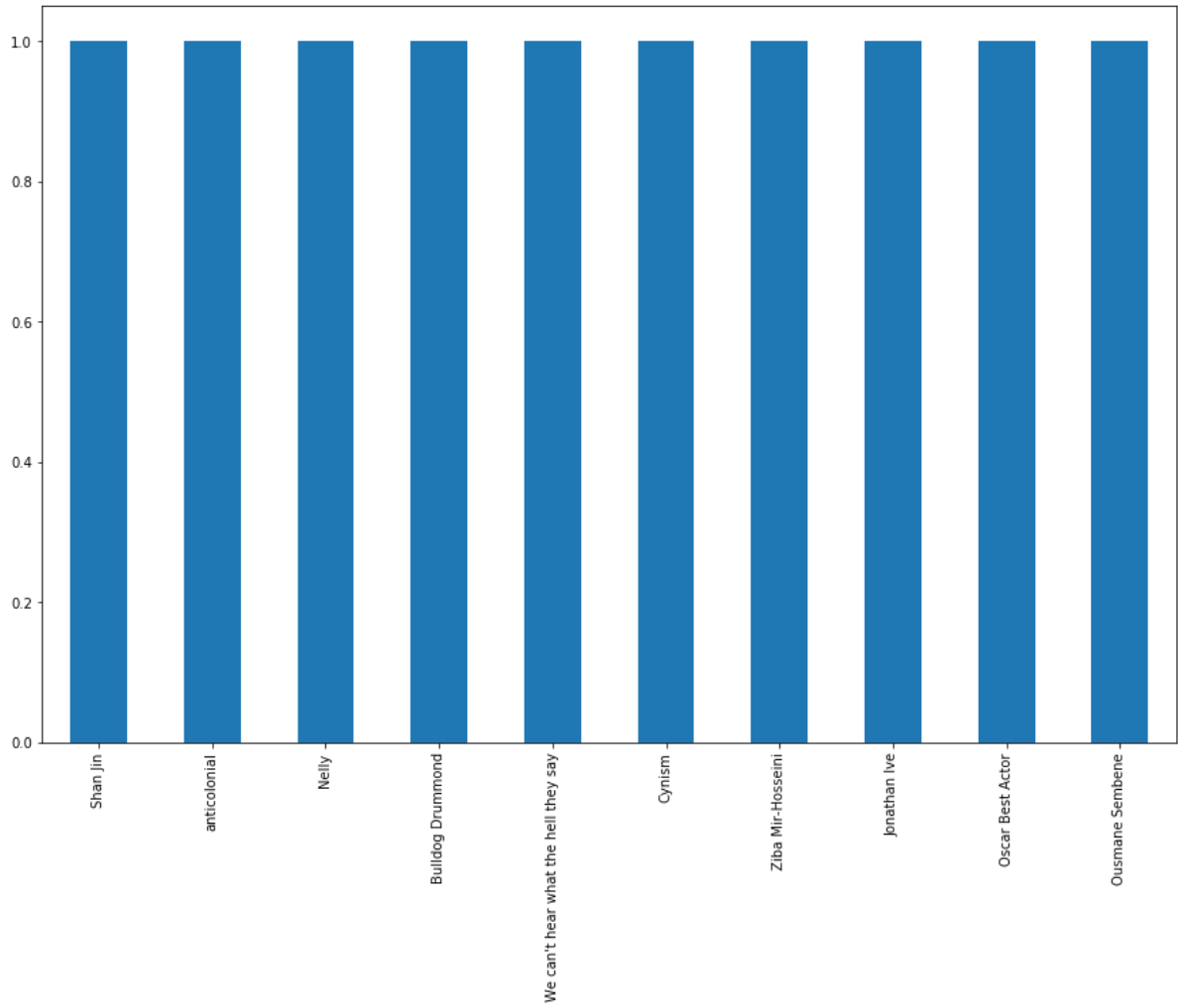
	userId	movieId	rating
20000253	138493	60816	4.5
20000254	138493	61160	4.0
20000255	138493	65682	4.5
20000256	138493	66762	4.5
20000257	138493	68319	4.5
20000258	138493	68954	4.5
20000259	138493	69526	4.5
20000260	138493	69644	3.0
20000261	138493	70286	5.0
20000262	138493	71619	2.5

In [47]:
```
tag_counts = tags['tag'].value_counts()
tag_counts[:10]
```

Out[47]:
```
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 [48]:
```
%matplotlib inline
tag_counts[-10:].plot(kind='bar', figsize=(15,10))
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf80e52cc0>
```



Filters for Selecting Rows

In [49]:
```
is_highly_rated = ratings['rating'] >= 4.0

ratings[is_highly_rated][-30:]
```

Out[49]:

	userId	movieId	rating
20000223	138493	47465	5.0
20000224	138493	48043	4.5
20000225	138493	48082	5.0
20000226	138493	48229	4.5
20000228	138493	48385	5.0
20000229	138493	48516	5.0
20000230	138493	48780	5.0
20000231	138493	49651	4.0
20000235	138493	51662	4.5
20000236	138493	51884	4.5
20000237	138493	52579	4.0
20000238	138493	52975	4.0
20000239	138493	53123	4.0
20000241	138493	53322	4.0
20000242	138493	53464	4.0
20000243	138493	53996	4.5
20000244	138493	55269	5.0
20000245	138493	55814	5.0
20000248	138493	58879	4.5
20000249	138493	59315	4.0
20000251	138493	59784	5.0
20000252	138493	60069	4.0
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
20000261	138493	70286	5.0

In [50]:
```
is_animation = movies['genres'].str.contains('Animation')

movies[is_animation][5:15]
```

Out[50]:

	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 [51]:
```
movies[is_animation].head(15)
```

Out[51]:

	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 [52]:
```
ratings_count = ratings[['movieId','rating']].groupby('rating').count()

ratings_count
```

Out[52]:

	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 [53]:
```
average_rating = ratings[['movieId','rating']].groupby('movieId').mean()
average_rating.head(10)
```

Out[53]:

	rating
movieId	
1	3.921240
2	3.211977
3	3.151040
4	2.861393
5	3.064592
6	3.834930
7	3.366484
8	3.142049
9	3.004924
10	3.430029

In [54]:
```
movie_count = ratings[['movieId','rating']].groupby('movieId').count()
movie_count.head()
```

Out[54]:

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

In [55]:
```
movie_count = ratings[['movieId','rating']].groupby('movieId').count()
movie_count.tail()
```

Out[55]:

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

Merge 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]:
```
movies.head()
```

Out[57]:

	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 [58]:
```
t = movies.merge(tags, on='movieId', how='inner')
t.head()
```

Out[58]:

	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

More examples: http://pandas.pydata.org/pandas-docs/stable/merging.html (http://pandas.pydata.org/pandas-docs/stable/merging.html)

Combine aggreagation, merging, and filters to get useful analytics

In [59]:
```
avg_ratings = ratings.groupby('movieId', as_index=False).mean()
del avg_ratings['userId']
avg_ratings.head()
```

Out[59]:

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

In [60]:
```
box_office = movies.merge(avg_ratings, on='movieId', how='inner')
box_office.tail()
```

Out[60]:

	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 [61]:
```
is_highly_rated = box_office['rating'] >= 4.0

box_office[is_highly_rated][-5:]
```

Out[61]:

	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 [62]:
```
is_comedy = box_office['genres'].str.contains('Comedy')

box_office[is_comedy][:5]
```

Out[62]:

	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 [63]:
```
box_office[is_comedy & is_highly_rated][-5:]
```

Out[63]:

	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

Vectorized String Operations

In [64]:
```
movies.head()
```

Out[64]:

	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 [65]:
```
movie_genres = movies['genres'].str.split('|', expand=True)
```

In [66]:
```
movie_genres[:10]
```

Out[66]:

	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 [67]:
```
movie_genres['isComedy'] = movies['genres'].str.contains('Comedy')
```

In [68]:
```
movie_genres[:10]
```

Out[68]:

	0	1	2	3	4	5	6	7	8	9	isCo
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 [69]:
```
movies['year'] = movies['title'].str.extract('.*\((.*)\).*', expand=True)
```

In [70]:
```
movies.tail()
```

Out[70]:

	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

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 [73]:
```
tags = pd.read_csv('./ml-20m/tags.csv', sep=',')
```

In [74]:
```
tags.dtypes
```

Out[74]:
```
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 [75]:
```
tags.head(5)
```

Out[75]:

	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 [76]:
```
tags['parsed_time'] = pd.to_datetime(tags['timestamp'], unit='s')
```

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

In [77]:
```
tags['parsed_time'].dtype
```

Out[77]:
```
dtype('<M8[ns]')
```

In [78]:
```
tags.head(2)
```

Out[78]:

	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

Selecting rows based on timestamps

In [79]:
```
greater_than_t = tags['parsed_time'] > '2015-02-01'

selected_rows = tags[greater_than_t]

tags.shape, selected_rows.shape
```

Out[79]: ((465564, 5), (12130, 5))

Sorting the table using the timestamps

In [80]:
```
tags.sort_values(by='parsed_time', ascending=True)[:10]
```

Out[80]:

	userId	movieId	tag	timestamp	parsed_time
333932	100371	2788	monty python	1135429210	2005-12-24 13:00:10
333927	100371	1732	coen brothers	1135429236	2005-12-24 13:00:36
333924	100371	1206	stanley kubrick	1135429248	2005-12-24 13:00:48
333923	100371	1193	jack nicholson	1135429371	2005-12-24 13:02:51
333939	100371	5004	peter sellers	1135429399	2005-12-24 13:03:19
333922	100371	47	morgan freeman	1135429412	2005-12-24 13:03:32
333921	100371	47	brad pitt	1135429412	2005-12-24 13:03:32
333936	100371	4011	brad pitt	1135429431	2005-12-24 13:03:51
333937	100371	4011	guy ritchie	1135429431	2005-12-24 13:03:51
333920	100371	32	bruce willis	1135429442	2005-12-24 13:04:02

Average Movie Ratings over Time

Are Movie ratings related to the year of launch?

In [81]:
```
average_rating = ratings[['movieId','rating']].groupby('movieId', as_index=False).mean()
average_rating.tail()
```

Out[81]:

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

In [82]:
```
joined = movies.merge(average_rating, on='movieId', how='inner')
joined.head()
joined.corr()
```

Out[82]:

	movieId	rating
movieId	1.000000	-0.090369
rating	-0.090369	1.000000

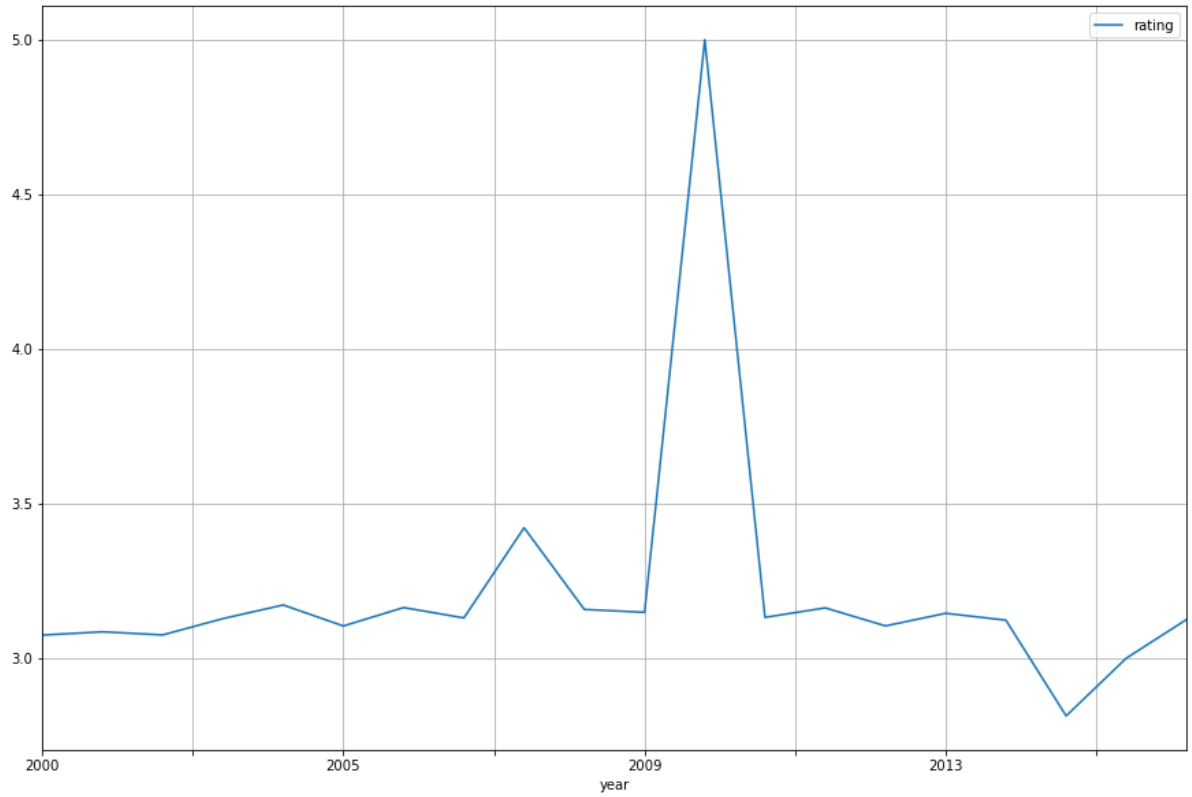
In [83]:
```
yearly_average = joined[['year','rating']].groupby('year', as_index=False).mean()
yearly_average[:10]
```

Out[83]:

	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 [84]:
```
yearly_average[-20:].plot(x='year', y='rating', figsize=(15,10), grid=True)
```

Out[84]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf83ddba20>
```



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

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

In [ ]:
```

```