A Short Intro to Pandas

A notebook created by Wes McKinney (@wesmckinn) and modified by Lynn Cherny (@arnicas) for the Python Data Science Afternoon, 12/2/12

In [6]:
```
from IPython.core.display import Image
Image(filename='screencaps/pandas_goals.png')
# From Python for Data Analysis (Wes McKinney), page 111:
```

Out[6]:

- Data structures with labeled axes supporting automatic or explicit data alignment. This prevents common errors resulting from misaligned data and working with differently-indexed data coming from different sources.
- Integrated time series functionality.
- The same data structures handle both time series data and non-time series data.
- Arithmetic operations and reductions (like summing across an axis) would pass on the metadata (axis labels).
- Flexible handling of missing data.
- Merge and other relational operations found in popular database databases (SQL-based, for example).

In [7]:
```
import numpy as np
import pandas as pd
import os
pd.set_printoptions(notebook_repr_html=True) # your choice- will format pret
```

In [8]:
```
# Defining a trivial data frame, by hand, from an example by @fonnesbeck

df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] * 6,
                   'B' : ['A', 'B', 'C'] * 8,
                   'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 4,
                   'D' : np.random.randn(24),
                   'E' : np.random.randn(24),
                   'F' : pd.DateRange(start='4/1/2012', periods=24)})

df
```

```
/Library/Frameworks/Python.framework/Versions/7.3/lib/python2.7/site-packages/pandas/core/daterange.py:21: FutureWarning: DateRange is deprecated, use DatetimeIndex instead
  FutureWarning)
```

Out[8]:

	A	B	C	D	E	F
0	one	A	foo	-0.298035	0.895074	2012-04-02 00:00:00
1	one	B	foo	0.227347	-0.465783	2012-04-03 00:00:00
2	two	C	foo	0.314661	1.285555	2012-04-04 00:00:00

3	three	A	bar	1.287470	-0.151542	2012-04-05 00:00:00
4	one	B	bar	0.779750	0.170240	2012-04-06 00:00:00
5	one	C	bar	-0.230199	2.008694	2012-04-09 00:00:00
6	two	A	foo	-0.866977	0.157307	2012-04-10 00:00:00
7	three	B	foo	-1.538491	2.029785	2012-04-11 00:00:00
8	one	C	foo	-1.844552	-0.512606	2012-04-12 00:00:00
9	one	A	bar	0.668057	0.434789	2012-04-13 00:00:00
10	two	B	bar	1.751206	1.700019	2012-04-16 00:00:00
11	three	C	bar	0.081031	-0.730185	2012-04-17 00:00:00
12	one	A	foo	1.240262	0.370668	2012-04-18 00:00:00
13	one	B	foo	0.037507	0.233086	2012-04-19 00:00:00
14	two	C	foo	1.347206	-0.626061	2012-04-20 00:00:00
15	three	A	bar	2.156120	0.572495	2012-04-23 00:00:00
16	one	B	bar	0.107094	-0.973417	2012-04-24 00:00:00
17	one	C	bar	0.108058	-0.003237	2012-04-25 00:00:00
18	two	A	foo	-0.277029	1.734459	2012-04-26 00:00:00
19	three	B	foo	-0.308738	-0.783287	2012-04-27 00:00:00
20	one	C	foo	0.511415	-1.202979	2012-04-30 00:00:00
21	one	A	bar	-2.074412	0.247890	2012-05-01 00:00:00
22	two	B	bar	1.328055	1.598737	2012-05-02 00:00:00
23	three	C	bar	-0.693149	-1.587220	2012-05-03 00:00:00

Let's read in some real data (movie lens data, a data set of ratings, from http://www.grouplens.org/node/73

In [9]:
```
base_dir = 'moviedata'

unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table(os.path.join(base_dir, 'users.dat'), sep='::',
                      header=None, names=unames)
```

In [10]:
```
users
```

Out[10]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6040 entries, 0 to 6039
Data columns:
user_id       6040  non-null values
gender        6040  non-null values
age           6040  non-null values
```

```
occupation    6040  non-null values
zip           6040  non-null values
dtypes: int64(3), object(2)
```

In [11]:
```
from IPython.core.display import Image
Image(filename='screencaps/read_csv.png')
# From Python for Data Analysis (Wes McKinney), chapter 6:
```

Out[11]:

Table 6-2. read_csv /read_table function arguments

Argument	Description
path	String indicating filesystem location, URL, or file-like object
sep or delimiter	Character sequence or regular expression to use to split fields in each row
header	Row number to use as column names. Defaults to 0 (first row), but should be None if there is no header row
index_col	Column numbers or names to use as the row index in the result. Can be a single name/number or a list of them for a hierarchical index
names	List of column names for result, combine with header=None
skiprows	Number of rows at beginning of file to ignore or list of row numbers (starting from 0) to skip
na_values	Sequence of values to replace with NA
comment	Character or characters to split comments off the end of lines
parse_dates	Attempt to parse data to datetime; False by default. If True, will attempt to parse all columns. Otherwise can specify a list of column numbers or name to parse. If element of list is tuple or list, will combine multiple columns together and parse to date (for example if date/time split across two columns)
keep_date_col	If joining columns to parse date, drop the joined columns. Default True
converters	Dict containing column number of name mapping to functions. For example {'foo': f} would apply the function f to all values in the 'foo' column
dayfirst	When parsing potentially ambiguous dates, treat as international format (e.g. 7/6/2012 -> June 7, 2012). Default False
date_parser	Function to use to parse dates
nrows	Number of rows to read from beginning of file
iterator	Return a TextParser object for reading file piecemeal
chunksize	For iteration, size of file chunks
skip_footer	Number of lines to ignore at end of file
verbose	Print various parser output information, like the number of missing values placed in non-numeric columns
encoding	Text encoding for unicode. For example 'utf-8' for UTF-8 encoded text
squeeze	If the parsed data only contains one column return a Series
thousands	Separator for thousands, e.g. ',' or '.'

In [12]:
```
users[:12]
```

Out[12]:

	user_id	gender	age	occupation	zip
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117

3	4	M	45	7	02460
4	5	M	25	20	55455
5	6	F	50	9	55117
6	7	M	35	1	06810
7	8	M	25	12	11413
8	9	M	25	17	61614
9	10	F	35	1	95370
10	11	F	25	1	04093
11	12	M	25	12	32793

In [13]:
```
users['age']#[:10]
# the [:10] is a slice - a short primer on slice syntax in python: http://sta
```

Out[13]:
```
0       1
1      56
2      25
3      45
4      25
5      50
6      35
7      25
8      25
9      35
10     25
11     25
12     45
13     35
14     25
...
6025    35
6026    18
6027    18
6028    25
6029    25
6030    18
6031    45
6032    50
6033    25
6034    25
6035    25
6036    45
6037    56
6038    45
6039    25
Name: age, Length: 6040
```

In [14]:
```
users.get_value(5,'age')  #5th item, column age
```

Out[14]: 50

In [15]:
```
users['age']<10
#users[users['age'] < 10]
```

Out[15]:
```
0        True
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
...
6025    False
6026    False
6027    False
6028    False
6029    False
6030    False
6031    False
6032    False
6033    False
6034    False
6035    False
6036    False
6037    False
6038    False
6039    False
Name: age, Length: 6040
```

In [90]:
```
users['age'].values
#list(users['age'].values) # to convert to a simple list again.
```

Out[90]:
```
array([ 1, 56, 25, ..., 56, 45, 25], dtype=int64)
```

Simple Stats

In [17]:
```
users.describe()
```

Out[17]:

	user_id	age	occupation
count	6040.000000	6040.000000	6040.000000
mean	3020.500000	30.639238	8.146854

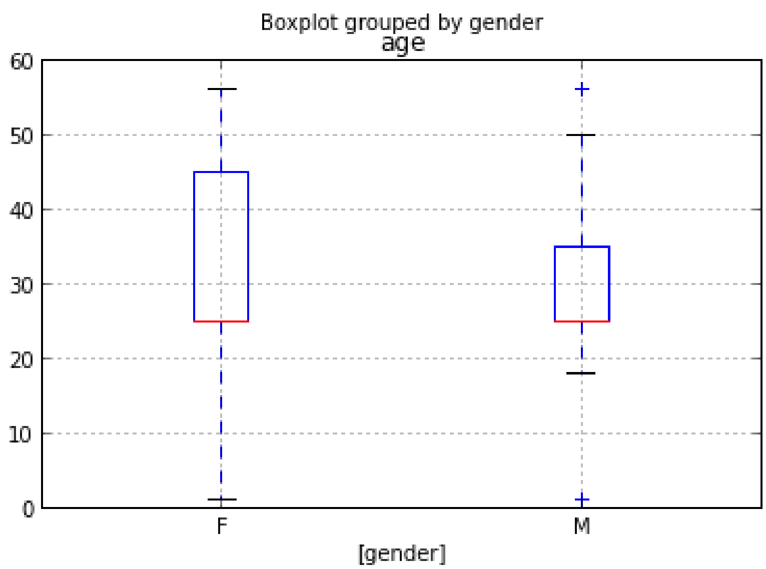
std	1743.742145	12.895962	6.329511
min	1.000000	1.000000	0.000000
25%	1510.750000	25.000000	3.000000
50%	3020.500000	25.000000	7.000000
75%	4530.250000	35.000000	14.000000
max	6040.000000	56.000000	20.000000

In [18]:
```
users[users['gender']=='F']['age'].mean() # try changing it to 'M' too?
```

Out[18]:
```
30.859566998244588
```

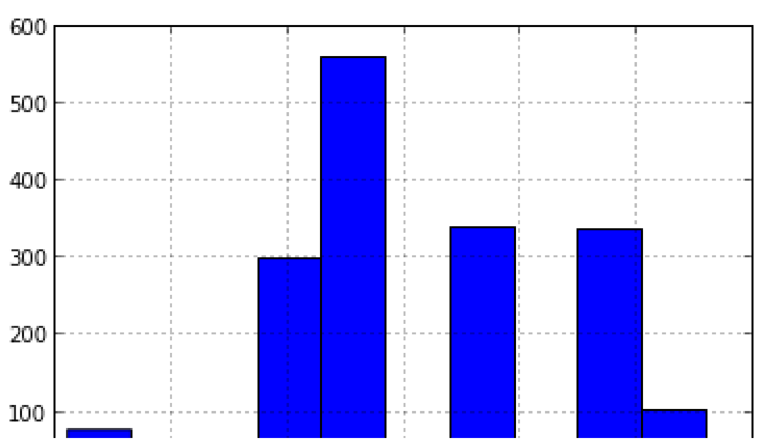
In [19]:
```
users.boxplot(column="age", by="gender")
```

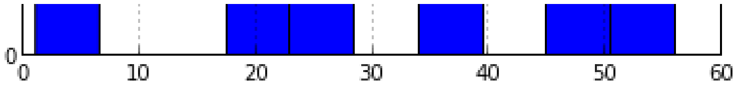
Out[19]:
```
<matplotlib.axes.AxesSubplot at 0x4b561f0>
```



In [20]:
```
users[users['gender']=='F']['age'].hist()
# do this with =='M' instead to see the subtle contrast.
# If you want, insert a new cell below this one and do it there.
```

Out[20]:
```
<matplotlib.axes.AxesSubplot at 0x4b70b90>
```





In [21]:

```
users.groupby('gender').count()
# users.groupby('gender')['age'].count()
```

Out[21]:

	user_id	gender	age	occupation	zip
gender					
F	1709	1709	1709	1709	1709
M	4331	4331	4331	4331	4331

In [22]:

```
from IPython.core.display import Image
Image(filename='screencaps/desc_stats.png')
# From Python for Data Analysis (Wes McKinney), page 139:
```

Out[22]:

Table 5-10. Descriptive and summary statistics

Method	Description
`count`	Number of non-NA values
`describe`	Compute set of summary statistics for Series or each DataFrame column
`min, max`	Compute minimum and maximum values
`argmin, argmax`	Compute index locations (integers) at which minimum or maximum value obtained, respectively
`idxmin, idxmax`	Compute index values at which minimum or maximum value obtained, respectively
`quantile`	Compute sample quantile ranging from 0 to 1
`sum`	Sum of values
`mean`	Mean of values
`median`	Arithmetic median (50% quantile) of values
`mad`	Mean absolute deviation from mean value
`var`	Sample variance of values
`std`	Sample standard deviation of values
`skew`	Sample skewness (3rd moment) of values
`kurt`	Sample kurtosis (4th moment) of values
`cumsum`	Cumulative sum of values
`cummin, cummax`	Cumulative minimum or maximum of values, respectively
`cumprod`	Cumulative product of values
`diff`	Compute 1st arithmetic difference (useful for time series)
`pct_change`	Compute percent changes

Merge Data Sets

In [23]:

```
from IPython.core.display import Image
```

```python
Image(filename='screencaps/merge.png')
# From Python for Data Analysis (Wes McKinney), page 139:
```

Out[23]:

- `pandas.merge` connects rows in DataFrames based on one or more keys. This will be familiar to users of SQL or other relational databases, as it implements database *join* operations.
- `pandas.concat` glues or stacks together objects along an axis.
- `combine_first` instance method enables splicing together overlapping data to fill in missing values in one object with values from another.

In [24]:

```python
rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table(os.path.join(base_dir, 'ratings.dat'), sep='::',
                        header=None, names=rnames)
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table(os.path.join(base_dir, 'movies.dat'), sep='::',
                       header=None, names=mnames)
```

In [25]:

```python
ratings
```

Out[25]:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns:
user_id      1000209  non-null values
movie_id     1000209  non-null values
rating       1000209  non-null values
timestamp    1000209  non-null values
dtypes: int64(4)
```

In [26]:

```python
ratings.head() # or .tail()
```

Out[26]:

	user_id	movie_id	rating	timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

In [27]:

```python
movies[:5]
```

Out[27]:

	movie_id	title	genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance

3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

In [28]:
```
ratings
```

Out[28]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns:
user_id      1000209  non-null values
movie_id     1000209  non-null values
rating       1000209  non-null values
timestamp    1000209  non-null values
dtypes: int64(4)
```

In [29]:
```
data = pd.merge(pd.merge(ratings, users), movies)
```

In [30]:
```
data
```

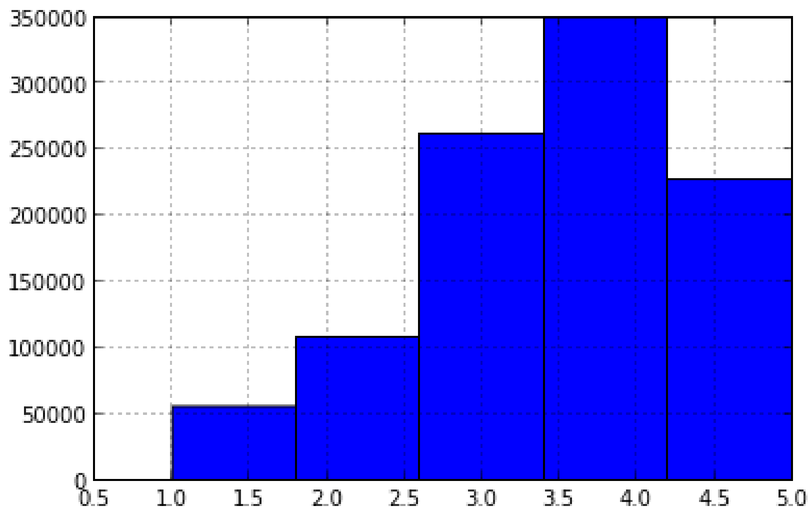
Out[30]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns:
user_id       1000209  non-null values
movie_id      1000209  non-null values
rating        1000209  non-null values
timestamp     1000209  non-null values
gender        1000209  non-null values
age           1000209  non-null values
occupation    1000209  non-null values
zip           1000209  non-null values
title         1000209  non-null values
genres        1000209  non-null values
dtypes: int64(6), object(4)
```

In [31]:
```
data.ix[0]
# change the 0 to another number to see another record... is this someone ra
```

Out[31]:
```
user_id                            1
movie_id                           1
rating                             5
timestamp                  978824268
gender                             F
age                                1
occupation                        10
zip                            48067
title               Toy Story (1995)
genres        Animation|Children's|Comedy
Name: 0
```

In [32]:
```
data['rating'].hist(bins=5) # the ratings are whole numbers, it seems
```

Out[32]: <matplotlib.axes.AxesSubplot at 0xe433bb0>



Grouping Data

In [33]:
```
mean_ratings = data.pivot_table('rating', rows='title',
                                cols='gender', aggfunc='mean')
```

In [34]:
```
mean_ratings[:5]
```

Out[34]:

gender	F	M
title		
$1,000,000 Duck (1971)	3.375000	2.761905
'Night Mother (1986)	3.388889	3.352941
'Til There Was You (1997)	2.675676	2.733333
'burbs, The (1989)	2.793478	2.962085
...And Justice for All (1979)	3.828571	3.689024

In [35]:
```
users.groupby('gender').count()
# users.groupby('gender')['age'].count()
```

Out[35]:

	user_id	gender	age	occupation	zip
gender					
F	1709	1709	1709	1709	1709
M	4331	4331	4331	4331	4331

In [36]:
```
from IPython.core.display import Image
```

```
Image(filename='screencaps/group_by.png')
# From Python for Data Analysis (Wes McKinney):
```

Out[36]:

Table 9-1. Optimized groupby methods

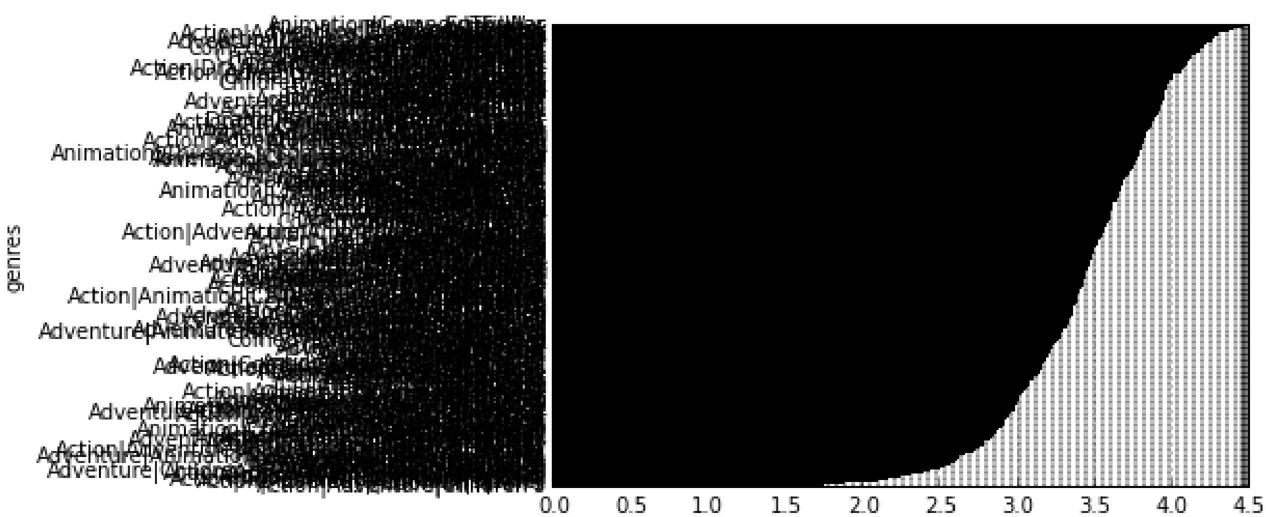
Function name	Description
count	Number of non-NA values in the group
sum	Sum of non-NA values
mean	Mean of non-NA values
median	Arithmetic median of non-NA values
std, var	Unbiased (n - 1 denominator) standard deviation and variance
min, max	Minimum and maximum of non-NA values
prod	Product of non-NA values
first, last	First and last non-NA values

In [37]:
```
ratings_by_genre = data.groupby('genres')
```

Try some graphs of different aspects of this, if you want...

In [38]:
```
ratings_by_genre.mean().sort_index(by='rating',ascending=True)['rating'].plot
# try changing the ascending argument to False; and then try just first 10..
# ratings_by_genre.mean().sort_index(by='rating',ascending=True)['age'].plot
#ratings_by_genre.mean().sort_index(by='rating',ascending=True)[:10]['rating
```

Out[38]: `<matplotlib.axes.AxesSubplot at 0x4c17c70>`



In [39]:
```
pd.crosstab(data.genres, data.gender, margins=True, colnames=['gender'])[:10
```

Out[39]:

gender	F	M	All
genres			
Action	1611	10700	12311
Action|Adventure	1978	8468	10446

Action|Adventure|Animation	64	281	345
Action|Adventure|Animation|Children's|Fantasy	41	94	135
Action|Adventure|Animation|Horror|Sci-Fi	71	547	618
Action|Adventure|Children's	4	40	44
Action|Adventure|Children's|Comedy	123	395	518
Action|Adventure|Children's|Fantasy	7	37	44
Action|Adventure|Children's|Sci-Fi	55	295	350
Action|Adventure|Comedy	433	1644	2077

In [40]:
```
pd.pivot_table(data, values=['rating'], rows=['genres','gender'], aggfunc=len
#pd.pivot_table(data, values=['rating'], rows=['genres','gender'], aggfunc=me
#pd.pivot_table(data, values=['rating'], rows=['genres','gender'], aggfunc=me
#pd.pivot_table(data, values=['rating'], rows=['genres','gender'], aggfunc=le
```

Out[40]:

		rating
genres	**gender**	
Action	**F**	1611
	M	10700
Action|Adventure	**F**	1978
	M	8468
Action|Adventure|Animation	**F**	64
	M	281
Action|Adventure|Animation|Children's|Fantasy	**F**	41
	M	94
Action|Adventure|Animation|Horror|Sci-Fi	**F**	71
	M	547

In [41]:
```
ratings_by_title = data.groupby('title').size()
ratings_by_title[:10]
```

Out[41]:
```
title
$1,000,000 Duck (1971)            37
'Night Mother (1986)              70
'Til There Was You (1997)         52
'burbs, The (1989)               303
...And Justice for All (1979)    199
1-900 (1994)                       2
10 Things I Hate About You (1999) 700
101 Dalmatians (1961)            565
101 Dalmatians (1996)            364
12 Angry Men (1957)              616
```

Some More Advanced Split-Apply-Combine Fu (Reminiscent of Plyr in R)

In [91]:
```
def top(df, n=5, column='rating'):
    return df.sort_index(by=column)[-n:]
```

In [92]:
```
top(ratings, n=6)
```

Out[92]:

	user_id	movie_id	rating	timestamp
760012	4516	2858	5	964856837
760011	4516	296	5	964856871
321543	1906	2176	5	974690515
760009	4516	924	5	964856468
760487	4518	3429	5	964843900
0	1	1193	5	978300760

In [93]:
```
data.groupby('gender').apply(top)
```

Out[93]:

		user_id	movie_id	rating	timestamp	gender	age	occupation	zi
gender									
F	**746956**	3601	2762	5	966637611	F	35	3	95
	746950	3590	2762	5	966652161	F	18	15	02
	746949	3589	2762	5	1019404113	F	45	0	80
	289418	4268	1177	5	965303869	F	45	20	04
	0	1	1	5	978824268	F	1	10	48
	316009	524	1212	5	976172449	M	18	0	91
	316008	509	1212	5	976297749	M	25	2	55

M	**316003**	444	1212	5	976241256	M	56	0	55
	773553	5504	2872	5	959735285	M	45	7	85 27
	299199	6022	1196	5	956756284	M	25	17	57

Subset the Data

In [42]:
```
active_titles = ratings_by_title.index[ratings_by_title >= 250]
```

In [43]:
```
active_titles[:15]
```

Out[43]:
```
Index(['burbs, The (1989), 10 Things I Hate About You (1999), 101
Dalmatians (1961), 101 Dalmatians (1996), 12 Angry Men (1957), 13th
Warrior, The (1999), 2 Days in the Valley (1996), 20,000 Leagues Under the
Sea (1954), 2001: A Space Odyssey (1968), 2010 (1984), 28 Days (2000), 39
Steps, The (1935), 54 (1998), 7th Voyage of Sinbad, The (1958), 8MM
(1999)], dtype=object)
```

In [44]:
```
# select only the ones with active titles
mean_ratings = mean_ratings.ix[active_titles]
```

In [45]:
```
mean_ratings
```

Out[45]:
```
<class 'pandas.core.frame.DataFrame'>
Index: 1216 entries, 'burbs, The (1989) to eXistenZ (1999)
Data columns:
F    1216  non-null values
M    1216  non-null values
dtypes: float64(2)
```

In [46]:
```
ratings_by_title[:12]
```

Out[46]:
```
title
$1,000,000 Duck (1971)             37
'Night Mother (1986)               70
'Til There Was You (1997)          52
'burbs, The (1989)                303
...And Justice for All (1979)     199
1-900 (1994)                        2
10 Things I Hate About You (1999)   700
```

```
101 Dalmatians (1961)              565
101 Dalmatians (1996)              364
12 Angry Men (1957)                616
13th Warrior, The (1999)           750
187 (1997)                          55
```

In [47]:
```
top_female_ratings = mean_ratings.sort_index(by='F', ascending=False)
```

In [48]:
```
top_female_ratings[:12]
```

Out[48]:

gender	F	M
title		
Close Shave, A (1995)	4.644444	4.473795
Wrong Trousers, The (1993)	4.588235	4.478261
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	4.572650	4.464589
Wallace & Gromit: The Best of Aardman Animation (1996)	4.563107	4.385075
Schindler's List (1993)	4.562602	4.491415
Shawshank Redemption, The (1994)	4.539075	4.560625
Grand Day Out, A (1992)	4.537879	4.293255
To Kill a Mockingbird (1962)	4.536667	4.372611
Creature Comforts (1990)	4.513889	4.272277
Usual Suspects, The (1995)	4.513317	4.518248
It Happened One Night (1934)	4.500000	4.163934
Rear Window (1954)	4.484536	4.472991

In [49]:
```
pd.set_printoptions(max_columns=20)
# needed because otherwise the male ratings won't print, due to long title
```

In [50]:
```
top_male_ratings = mean_ratings.sort_index(by='M', ascending=False)
```

In [51]:
```
top_male_ratings[:10] #.plot(kind='barh')
```

Out[51]:

gender	F	M
title		
Godfather, The (1972)	4.314700	4.583333
Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (1954)	4.481132	4.576628
Shawshank Redemption, The (1994)	4.539075	4.560625
Raiders of the Lost Ark (1981)	4.332168	4.520597

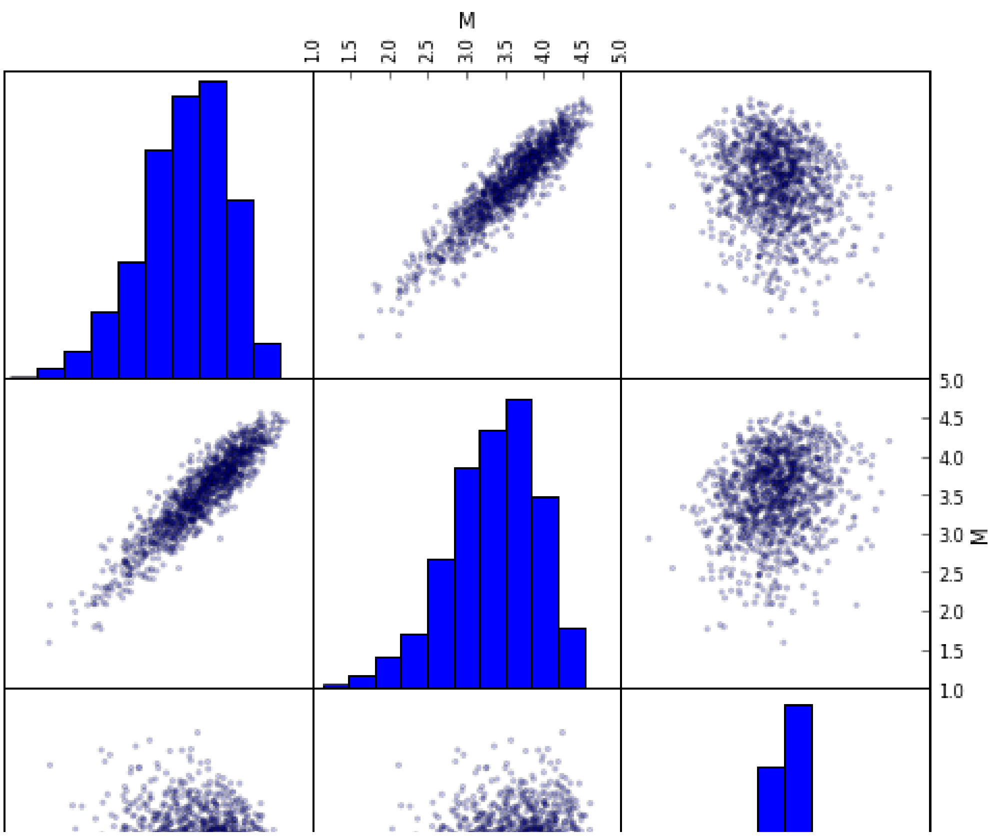
Usual Suspects, The (1995)	4.513317	4.518248
Star Wars: Episode IV - A New Hope (1977)	4.302937	4.495307
Schindler's List (1993)	4.562602	4.491415
Wrong Trousers, The (1993)	4.588235	4.478261
Close Shave, A (1995)	4.644444	4.473795
Rear Window (1954)	4.484536	4.472991

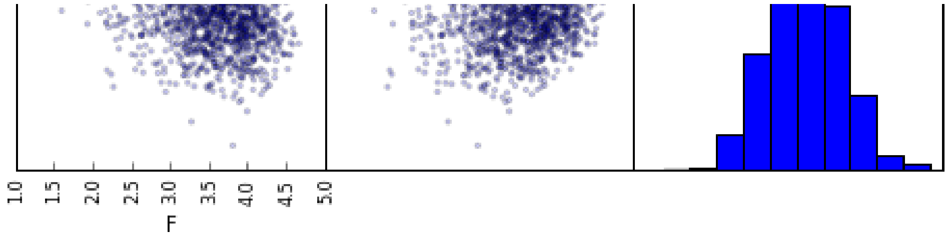
Measuring disagreement - Add Column

In [52]:
```
# add a column
mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
```

In [53]:
```
pd.scatter_matrix(mean_ratings,alpha=0.2, figsize=(8, 8)) # ,diagonal='kde'
```

Out[53]:
```
array([[Axes(0.125,0.641667;0.258333x0.258333),
        Axes(0.383333,0.641667;0.258333x0.258333),
        Axes(0.641667,0.641667;0.258333x0.258333)],
       [Axes(0.125,0.383333;0.258333x0.258333),
        Axes(0.383333,0.383333;0.258333x0.258333),
        Axes(0.641667,0.383333;0.258333x0.258333)],
       [Axes(0.125,0.125;0.258333x0.258333),
        Axes(0.383333,0.125;0.258333x0.258333),
        Axes(0.641667,0.125;0.258333x0.258333)]], dtype=object)
```





In [54]:
```
mean_ratings['F'].median()
#mean_ratings['M'].median()
```

Out[54]: 3.6045643873654982

In [55]:
```
mean_ratings.describe()
# or mean_ratings['M'].describe() etc
```

Out[55]:

gender	F	M	diff
count	1216.000000	1216.000000	1216.000000
mean	3.548584	3.541090	-0.007494
std	0.508329	0.513287	0.208956
min	1.574468	1.616949	-0.830782
25%	3.243709	3.208182	-0.141280
50%	3.604564	3.605161	-0.009800
75%	3.924150	3.913895	0.116268
max	4.644444	4.583333	0.726351

In [56]:
```
sorted_by_diff = mean_ratings.sort_index(by='diff')  # default is ascending=
```

In [57]:
```
sorted_by_diff[:15]
```

Out[57]:

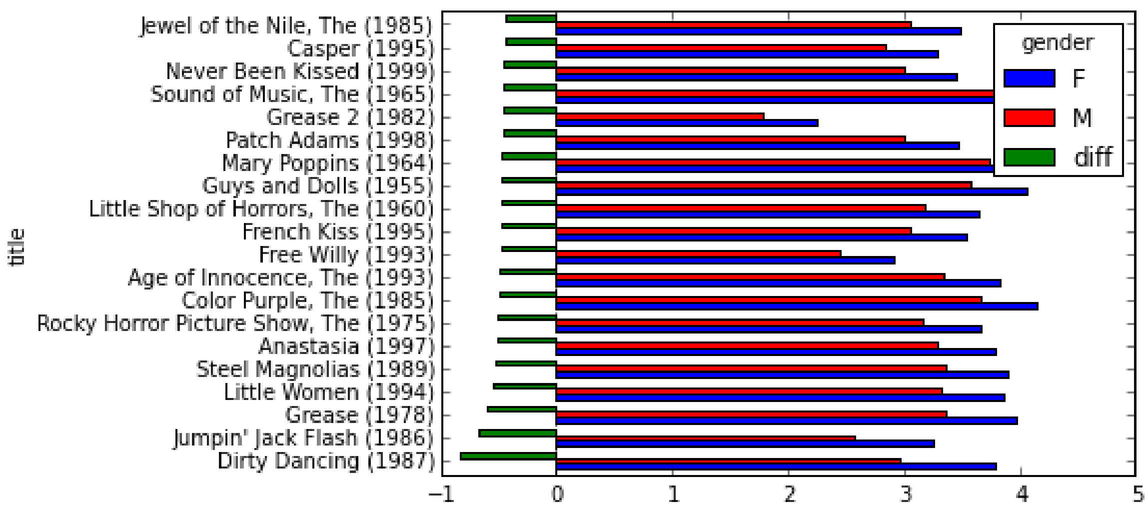
gender	F	M	diff
title			
Dirty Dancing (1987)	3.790378	2.959596	-0.830782
Jumpin' Jack Flash (1986)	3.254717	2.578358	-0.676359
Grease (1978)	3.975265	3.367041	-0.608224
Little Women (1994)	3.870588	3.321739	-0.548849
Steel Magnolias (1989)	3.901734	3.365957	-0.535777
Anastasia (1997)	3.800000	3.281609	-0.518391
Rocky Horror Picture Show, The (1975)	3.673016	3.160131	-0.512885
Color Purple, The (1985)	4.158192	3.659341	-0.498851
Age of Innocence, The (1993)	3.827068	3.339506	-0.487561

Age of Innocence, The (1993)	[illegible]	[illegible]	[illegible]
Free Willy (1993)	2.921348	2.438776	-0.482573
French Kiss (1995)	3.535714	3.056962	-0.478752
Little Shop of Horrors, The (1960)	3.650000	3.179688	-0.470312
Guys and Dolls (1955)	4.051724	3.583333	-0.468391
Mary Poppins (1964)	4.197740	3.730594	-0.467147
Patch Adams (1998)	3.473282	3.008746	-0.464536

A Graphical Exploration For Another Time, Probably

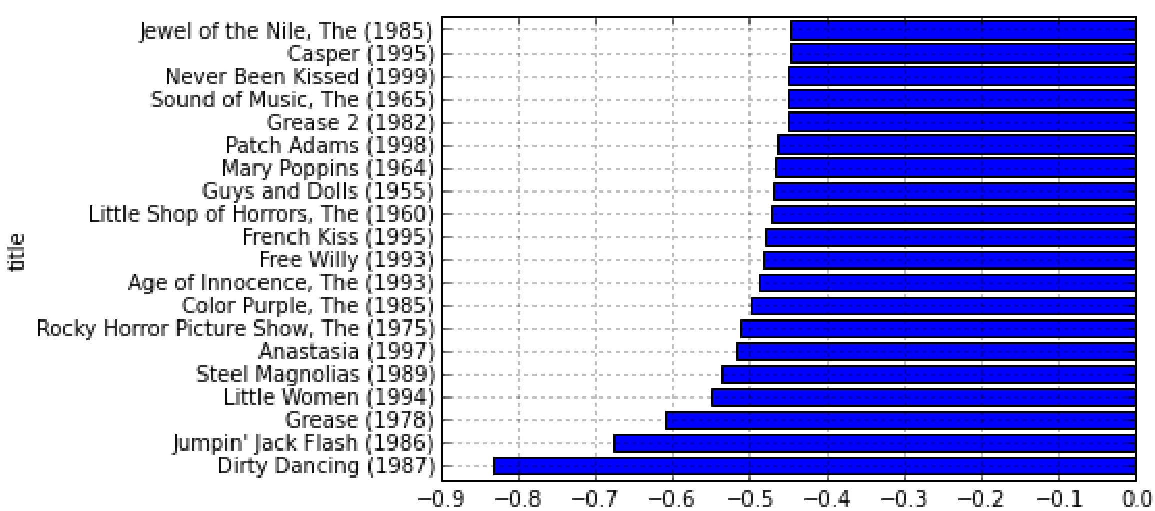
In [58]:
```
sorted_by_diff[:20].plot(kind='barh')
```

Out[58]: <matplotlib.axes.AxesSubplot at 0x5e39bf0>



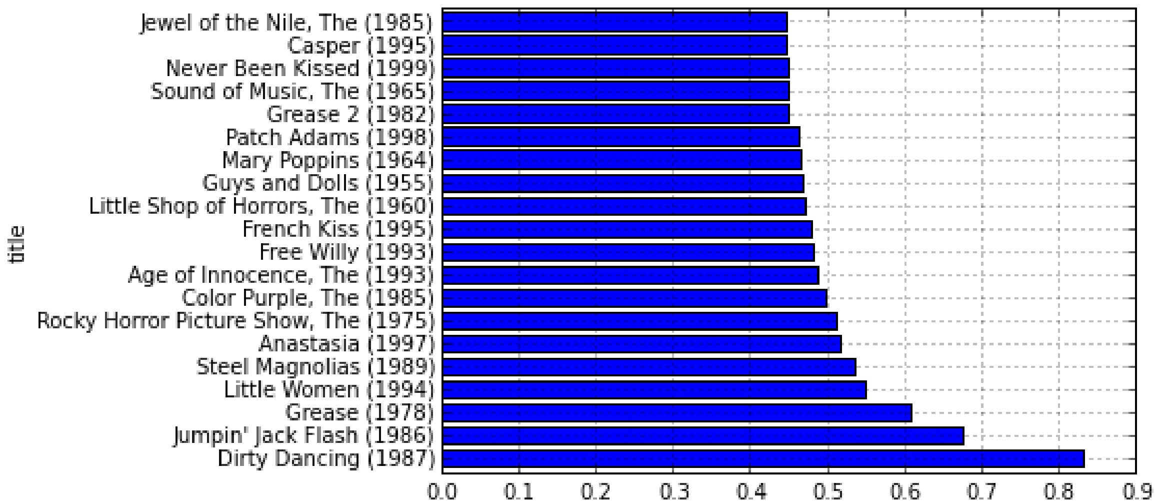
In [59]:
```
sorted_by_diff['diff'][:20].plot(kind='barh')
```

Out[59]: <matplotlib.axes.AxesSubplot at 0x5eeecf0>



In [60]:
```
# fix the orientation by using a numpy absolute value function
np.abs(sorted_by_diff['diff'][:20]).plot(kind='barh')
```

Out[60]: <matplotlib.axes.AxesSubplot at 0x655f3d0>



In [61]:
```
sorted_by_diff[::-1][:15]
# the ones that men liked and women didn't - [::-1] reverses a list in pytho
```

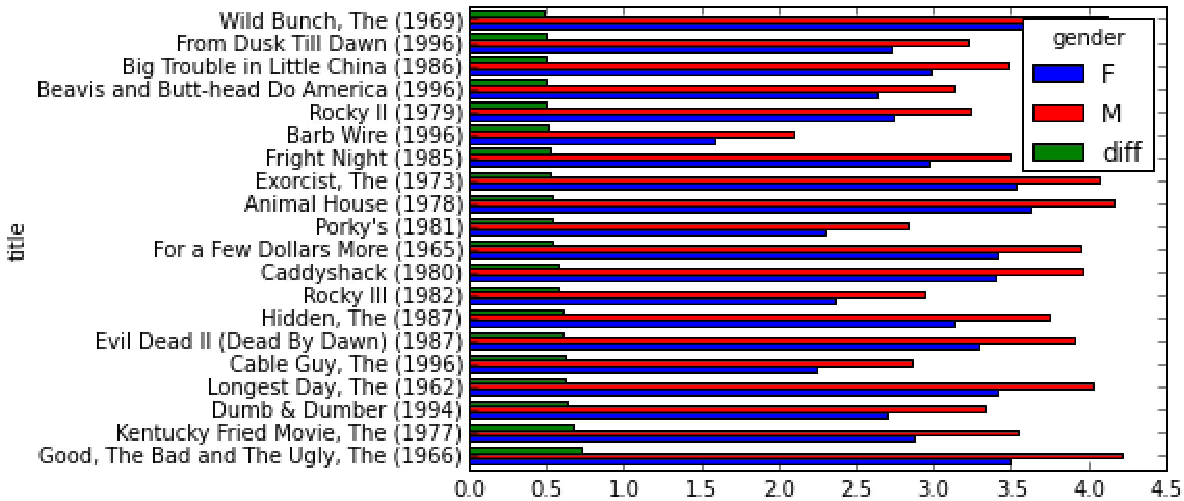
Out[61]:

gender	F	M	diff
title			
Good, The Bad and The Ugly, The (1966)	3.494949	4.221300	0.726351
Kentucky Fried Movie, The (1977)	2.878788	3.555147	0.676359
Dumb & Dumber (1994)	2.697987	3.336595	0.638608
Longest Day, The (1962)	3.411765	4.031447	0.619682
Cable Guy, The (1996)	2.250000	2.863787	0.613787
Evil Dead II (Dead By Dawn) (1987)	3.297297	3.909283	0.611985
Hidden, The (1987)	3.137931	3.745098	0.607167
Rocky III (1982)	2.361702	2.943503	0.581801
Caddyshack (1980)	3.396135	3.969737	0.573602
For a Few Dollars More (1965)	3.409091	3.953795	0.544704
Porky's (1981)	2.296875	2.836364	0.539489
Animal House (1978)	3.628906	4.167192	0.538286
Exorcist, The (1973)	3.537634	4.067239	0.529605
Fright Night (1985)	2.973684	3.500000	0.526316
Barb Wire (1996)	1.585366	2.100386	0.515020

In [62]:
```
sorted_by_diff[::-1][:20].plot(kind='barh')
```

```
# a dumb plot first... but shows some movies have low ratings overall - look
```

Out[62]: <matplotlib.axes.AxesSubplot at 0x66281b0>



In [63]:
```
# let's add a simple avg of the 2 ratings as a new column
mean_ratings['avg'] = (mean_ratings['M'] + mean_ratings['F']) / 2
```

In [64]:
```
mean_ratings[:5]  # you should see the new col, avg
```

Out[64]:

gender	F	M	diff	avg
title				
'burbs, The (1989)	2.793478	2.962085	0.168607	2.877782
10 Things I Hate About You (1999)	3.646552	3.311966	-0.334586	3.479259
101 Dalmatians (1961)	3.791444	3.500000	-0.291444	3.645722
101 Dalmatians (1996)	3.240000	2.911215	-0.328785	3.075607
12 Angry Men (1957)	4.184397	4.328421	0.144024	4.256409

In [65]:
```
sorted_by_diff = mean_ratings.sort_index(by='diff', ascending=False)  # mean.
# you could make a sorted_by_avg too.  try it?
```

In [66]:
```
sorted_by_diff[:10]
```

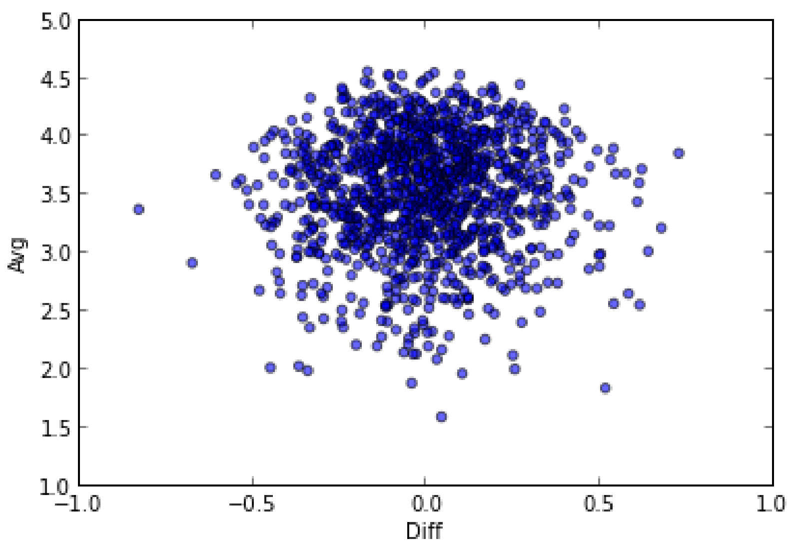
Out[66]:

gender	F	M	diff	avg
title				
Good, The Bad and The Ugly, The (1966)	3.494949	4.221300	0.726351	3.858125
Kentucky Fried Movie, The (1977)	2.878788	3.555147	0.676359	3.216967
Dumb & Dumber (1994)	2.697987	3.336595	0.638608	3.017291

Dumb & Dumber (1994)	[illegible]	[illegible]	[illegible]	[illegible]
Longest Day, The (1962)	3.411765	4.031447	0.619682	3.721606
Cable Guy, The (1996)	2.250000	2.863787	0.613787	2.556894
Evil Dead II (Dead By Dawn) (1987)	3.297297	3.909283	0.611985	3.603290
Hidden, The (1987)	3.137931	3.745098	0.607167	3.441515
Rocky III (1982)	2.361702	2.943503	0.581801	2.652602
Caddyshack (1980)	3.396135	3.969737	0.573602	3.682936
For a Few Dollars More (1965)	3.409091	3.953795	0.544704	3.681443

In [67]:
```
plt.scatter(sorted_by_diff['diff'], sorted_by_diff['avg'],alpha=.6)
xlabel("Diff")
ylabel("Avg")
```

Out[67]: <matplotlib.text.Text at 0x6a50e30>



In [95]:
```
# Remember how to look at a command's options in the notebook: (Click the ba
plt.scatter?
```

In [69]:
```
sorted_by_diff.describe()
```

Out[69]:

gender	F	M	diff	avg
count	1216.000000	1216.000000	1216.000000	1216.000000
mean	3.548584	3.541090	-0.007494	3.544837
std	0.508329	0.513287	0.208956	0.500015
min	1.574468	1.616949	-0.830782	1.595709
25%	3.243709	3.208182	-0.141280	3.236225
50%	3.604564	3.605161	-0.009800	3.610063
75%	3.924150	3.913895	0.116268	3.906921

max	4.644444	4.583333	0.726351	4.559119

In [70]:
```
sorted_by_diff[sorted_by_diff['diff']< -.83]
```

Out[70]:

gender	**F**	**M**	**diff**	**avg**
title				
Dirty Dancing (1987)	3.790378	2.959596	-0.830782	3.374987

In [71]:
```
sorted_by_diff
```

Out[71]:
```
<class 'pandas.core.frame.DataFrame'>
Index: 1216 entries, Good, The Bad and The Ugly, The (1966) to Dirty
Dancing (1987)
Data columns:
F        1216  non-null values
M        1216  non-null values
diff     1216  non-null values
avg      1216  non-null values
dtypes: float64(4)
```

In [72]:
```
# Since the data frame is indexed by the movie names, we can do this:
sorted_by_diff.index[0]
# since it was already sorted, we can look at the first item's name this way
# sorted_by_diff.index # try this too...
# sorted_by_diff.ix[0] # gives the whole record at that index.
```

Out[72]:
```
'Good, The Bad and The Ugly, The (1966)'
```

More Simple Stats: Movies With Biggest Opinion Spread

In [73]:
```
rating_std_by_title = data.groupby('title')['rating'].std()
```

In [74]:
```
rating_std_by_title = rating_std_by_title.ix[active_titles]
```

In [75]:
```
rating_std_by_title.order(ascending=False)[:10]
```

Out[75]:
```
title
Dumb & Dumber (1994)                        1.321333
Blair Witch Project, The (1999)             1.316368
Natural Born Killers (1994)                 1.307198
Tank Girl (1995)                            1.277695
Rocky Horror Picture Show, The (1975)       1.260177
Eyes Wide Shut (1999)                       1.259624
Evita (1996)                                1.253631
```

```
Billy Madison (1995)                       1.249970
Fear and Loathing in Las Vegas (1998)      1.246408
Bicentennial Man (1999)                    1.245533
Name: rating
```

String Operations - Ratings by decade

In [76]:

```
mean_ratings_noidx = mean_ratings.reset_index()
```

In [77]:

```
import re
year_re = r'.* \(([\d]+)\)'

year = mean_ratings_noidx.title.str.match(year_re).str[0]

movie_decade = pd.Series(year.astype(int).values // 10 * 10,
                         index=mean_ratings_noidx.title)
movie_decade
```

Out[77]:

```
title
'burbs, The (1989)                      1980
10 Things I Hate About You (1999)       1990
101 Dalmatians (1961)                   1960
101 Dalmatians (1996)                   1990
12 Angry Men (1957)                     1950
13th Warrior, The (1999)                1990
2 Days in the Valley (1996)             1990
20,000 Leagues Under the Sea (1954)     1950
2001: A Space Odyssey (1968)            1960
2010 (1984)                             1980
28 Days (2000)                          2000
39 Steps, The (1935)                    1930
54 (1998)                               1990
7th Voyage of Sinbad, The (1958)        1950
8MM (1999)                              1990
...
Working Girl (1988)                       1980
World Is Not Enough, The (1999)           1990
Wrong Trousers, The (1993)                1990
Wyatt Earp (1994)                         1990
X-Files: Fight the Future, The (1998)     1990
X-Men (2000)                              2000
Year of Living Dangerously (1982)         1980
Yellow Submarine (1968)                   1960
You've Got Mail (1998)                    1990
Young Frankenstein (1974)                 1970
Young Guns (1988)                         1980
Young Guns II (1990)                      1990
Young Sherlock Holmes (1985)              1980
Zero Effect (1998)                        1990
eXistenZ (1999)                           1990
Length: 1216
```

In [78]:
```
from IPython.core.display import Image
Image(filename='screencaps/vec_string.png')
# From Python for Data Analysis (Wes McKinney), chapter 7:
```

Out[78]:

Table 7-5. Vectorized string methods

Method	Description
`cat`	Concatenate strings element-wise with optional delimiter
`contains`	Return boolean array if each string contains pattern/regex
`count`	Count occurrences of pattern
`endswith, startswith`	Equivalent to `x.endswith(pattern)` or `x.startswith(pattern)` for each element.
`findall`	Compute list of all occurrences of pattern/regex for each string
`get`	Index into each element (retrieve i-th element)
`join`	Join strings in each element of the Series with passed separator
`len`	Compute length of each string
`lower, upper`	Convert cases; equivalent to `x.lower()` or `x.upper()` for each element.
`match`	Use `re.match` with the passed regular expression on each element, returning matched groups as list.
`pad`	Add whitespace to left, right, or both sides of strings
`center`	Equivalent to `pad(side='both')`
`repeat`	Duplicate values; for example `s.str.repeat(3)` equivalent to `x * 3` for each string.
`replace`	Replace occurrences of pattern/regex with some other string
`slice`	Slice each string in the Series.
`split`	Split strings on delimiter or regular expression
`strip, rstrip, lstrip`	Trim whitespace, including newlines; equivalent to `x.strip()` (and `rstrip`, `lstrip`, respectively) for each element.

In [79]:
```
data_sub = data[data.title.isin(active_titles)]  #isin is cool!
data_sub
```

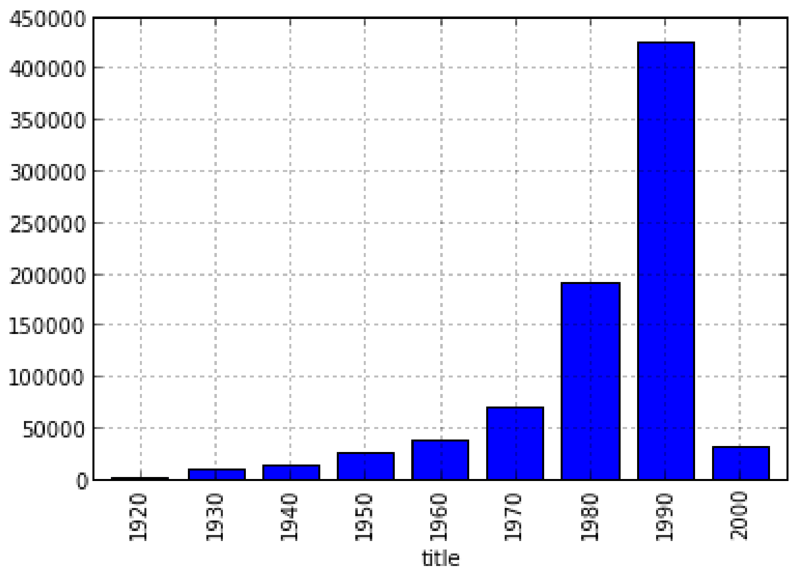
Out[79]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 808922 entries, 0 to 1000208
Data columns:
user_id       808922  non-null values
movie_id      808922  non-null values
rating        808922  non-null values
timestamp     808922  non-null values
gender        808922  non-null values
age           808922  non-null values
occupation    808922  non-null values
zip           808922  non-null values
title         808922  non-null values
genres        808922  non-null values
dtypes: int64(6), object(4)
```

In [80]:
```
by_decade = data_sub.groupby(data_sub.title.map(movie_decade))
by_decade.size()
```

Out[80]:
```
title
1920       663
1930      8889
1940     13843
1950     26596
1960     38816
1970     69768
1980    192012
1990    425623
2000     32712
```

In [81]:
```
by_decade.size().plot(kind='bar')
```

Out[81]: <matplotlib.axes.AxesSubplot at 0x6a7ba50>



In [82]:
```
decade = data_sub.title.map(movie_decade)
decade.name = 'decade'
by_decade_gender = data_sub.groupby([decade, 'gender'])
by_decade_gender.rating.mean()
```

Out[82]:
```
decade  gender
1920    F         4.047244
        M         4.145522
1930    F         4.178956
        M         4.065204
1940    F         4.179606
        M         4.207828
1950    F         4.096413
        M         4.041530
1960    F         3.993481
        M         3.999227
1970    F         3.872816
        M         3.928632
1980    F         3.717687
        M         3.687492
1990    F         3.613303
        M         3.556675
```

```
2000    F         3.519964
        M         3.482841
Name: rating
```

In [83]:
```
by_decade_gender.rating.mean().unstack()
# make them columns
```

Out[83]:

gender	F	M
decade		
1920	4.047244	4.145522
1930	4.178956	4.065204
1940	4.179606	4.207828
1950	4.096413	4.041530
1960	3.993481	3.999227
1970	3.872816	3.928632
1980	3.717687	3.687492
1990	3.613303	3.556675
2000	3.519964	3.482841

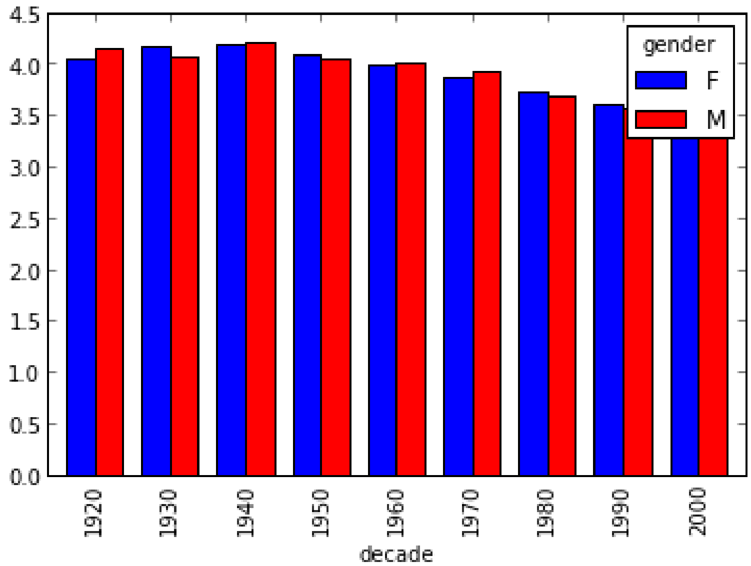
In [84]:
```
by_decade_gender.rating.mean().unstack
```

Out[84]:
```
<bound method Series.unstack of decade  gender
1920    F         4.047244
        M         4.145522
1930    F         4.178956
        M         4.065204
1940    F         4.179606
        M         4.207828
1950    F         4.096413
        M         4.041530
1960    F         3.993481
        M         3.999227
1970    F         3.872816
        M         3.928632
1980    F         3.717687
        M         3.687492
1990    F         3.613303
        M         3.556675
2000    F         3.519964
        M         3.482841
Name: rating>
```

In [85]:
```
mrat_decade = by_decade_gender.rating.mean()
#mrat_decade.plot(kind='bar')  # uncommment this line, and see what it looks
mrat_decade.unstack().plot(kind='bar')
```

Out[85]: <matplotlib.axes.AxesSubplot at 0x6a50850>

```
Out[85]: <matplotlib.axes.AxesSubplot at 0x6a50850>
```



Output Some Data Fast

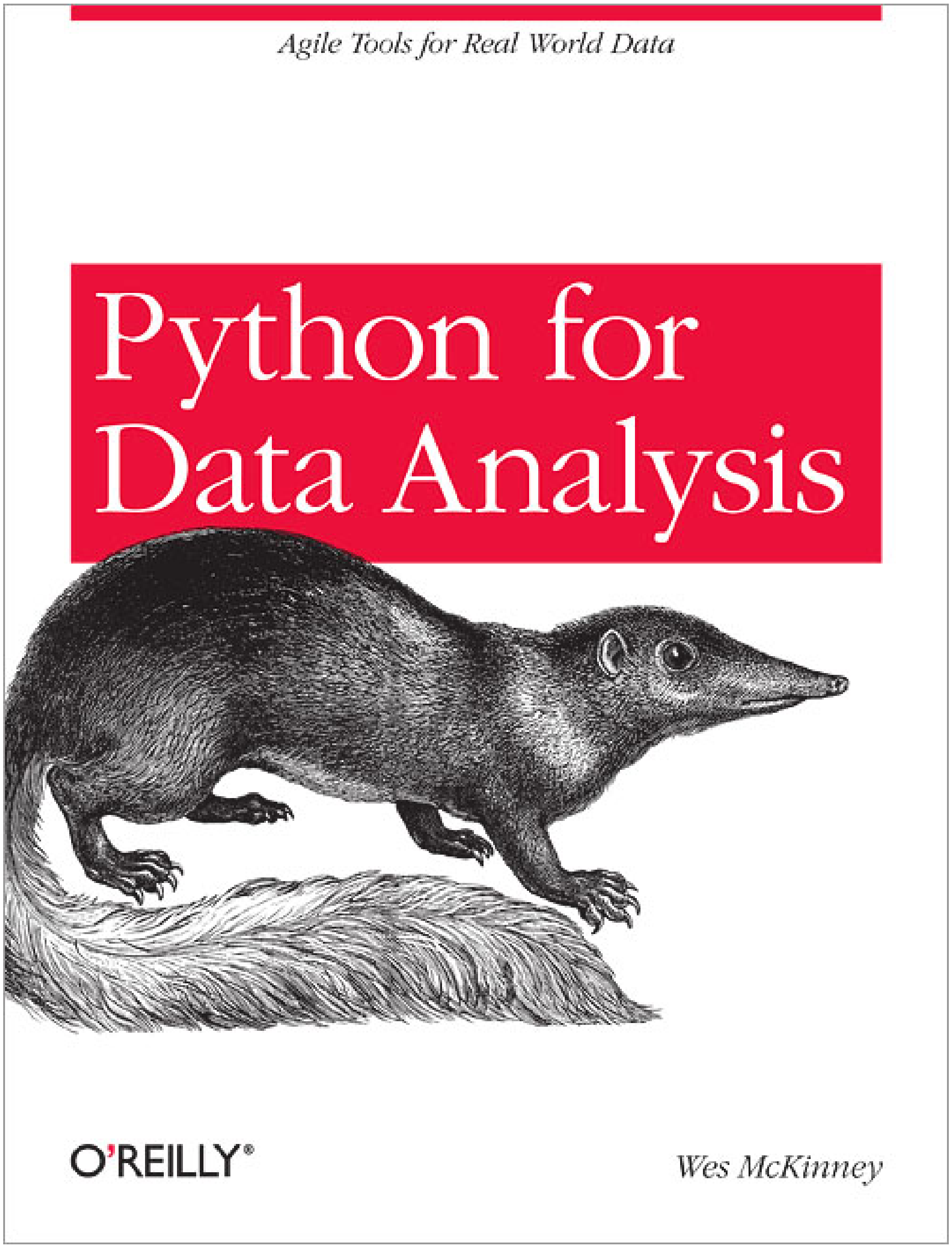
In [86]:
```
by_decade_gender.rating.mean().to_csv("my file2.csv")
```

In [87]:
```
ls
```

```
Fonnesbeck_Python_for_R.pdf     Intro_to_Pandas.pdf
IPython Notebook Intro.pdf      Python_for_R_Fonnesbeck.ipynb
IPythonNotebooks.ipynb          moviedata/
Intro_Pandas_Movies.ipynb       my file2.csv
Intro_Resources.key             screencaps/
```

A Few More links

- Dev site with many materials: https://github.com/pydata
- Home page: http://pandas.pydata.org/
- Get the book! Python for Data Analysis by Wes McKinney: http://shop.oreilly.com/product/0636920023784.do


Agile Tools for Real World Data
Python for Data Analysis
O'REILLY®
Wes McKinney