In [1]:

```
import requests
from pattern import web
from bs4 import BeautifulSoup
```

Scrapping Demo

Task

Find and print the movie title, list of genres, runtime, and score of all movies on this page (http://www.imdb.com/search/title?at=0&sort=num_votes,desc&start=1&title_type=feature&year=1940,2015)

Base URL and params

In [78]:

```
url = 'http://www.imdb.com/search/title'
params = dict(sort='num_votes,desc', start=5551, title_type='feature', year='1940,2015')

r = requests.get(url, params=params)
print r.url  # notice it constructs the full url for you
```

```
http://www.imdb.com/search/title?sort=num_votes%2Cdesc&start=5551&title_type=feature&y
ear=1940%2C2015
```

Using Pattern

Selection in pattern follows the rules of CSS:

In [83]:

```
dom = web.Element(r.text)

for movie in dom.by_tag('tr.detailed'):
    title = movie.by_tag('td.title')[0]
    imdbID = title.by_tag('a')[0].href.replace('/title/', '').strip('/')
    name = web.plaintext(title.by_tag('a')[0].content)
    year = title.by_tag('span.year_type')[0].content
    genres = title.by_tag('span.genre')[0].by_tag('a')
    genres = [g.content for g in genres]
    try:
        runtime = title.by_tag('span.runtime')[0].content
    except IndexError:
        runtime = "NA"
    rating = title.by_tag('span.value')[0].content
    votes = movie.by_tag('td.sort_col')[0].content
    print imdbID, name, year, genres, runtime, rating, votes
```

```
tt2771372 Veronica Mars (2014) [u'Comedy', u'Crime', u'Drama', u'Mystery'] 107 mins.
6.9 35,763
tt0074812 Logan's Run (1976) [u'Action', u'Adventure', u'Sci-Fi'] 119 mins. 6.8 35,689
tt0089670 European Vacation (1985) [u'Comedy'] 95 mins. 6.1 35,687
tt1527788 The Man from Nowhere (2010) [u'Action', u'Crime', u'Thriller'] 119 mins. 7.9
35,648
tt0096933 Black Rain (1989) [u'Action', u'Crime', u'Thriller'] 125 mins. 6.6 35,637
tt1214962 Seeking Justice (2011) [u'Action', u'Drama', u'Thriller'] 105 mins. 6.2 35,6
33
tt0055031 Judgment at Nuremberg (1961) [u'Drama', u'History', u'War'] 186 mins. 8.3 3
5,623
tt0104684 Hard Boiled (1992) [u'Action', u'Crime', u'Drama', u'Thriller'] 128 mins.
7.9 35,603
tt0155975 Psycho (1998) [u'Horror', u'Mystery', u'Thriller'] 105 mins. 4.6 35,593
tt0271668 National Security (2003) [u'Action', u'Comedy', u'Crime', u'Thriller'] 88 mi
ns. 5.5 35,594
```

```
tt0266465 A Man Apart (2003) [u'Action', u'Crime', u'Drama', u'Thriller'] 109 mins.
6.1 35,554
tt0331811 11:14 (2003) [u'Comedy', u'Crime', u'Drama', u'Mystery'] 86 mins. 7.2 35,543
tt1185834 Star Wars: The Clone Wars (2008) [u'Animation', u'Action', u'Adventure', u'F
antasy', u'Sci-Fi', u'Thriller'] 98 mins. 5.8 35,542
tt0113117 French Kiss (1995) [u'Comedy', u'Drama', u'Romance'] 111 mins. 6.5 35,523
tt0097202 The Killer (1989) [u'Action', u'Crime', u'Drama', u'Thriller'] 111 mins. 7.9
35,512
tt0444682 The Reaping (2007) [u'Horror', u'Thriller'] 99 mins. 5.7 35,506
tt0086425 Terms of Endearment (1983) [u'Comedy', u'Drama'] 132 mins. 7.4 35,492
tt0091129 The Golden Child (1986) [u'Action', u'Adventure', u'Comedy', u'Fantasy', u'M
ystery'] 94 mins. 5.8 35,441
tt0102536 Night on Earth (1991) [u'Comedy', u'Drama'] 129 mins. 7.8 35,442
tt0362165 Son of the Mask (2005) [u'Adventure', u'Comedy', u'Family', u'Fantasy'] 94 m
ins. 2.1 35,405
tt0089370 The Jewel of the Nile (1985) [u'Action', u'Adventure', u'Comedy', u'Romanc
e'] 106 mins. 6.0 35,398
tt1191111 Enter the Void (2009) [u'Drama', u'Fantasy'] 161 mins. 7.3 35,407
tt1175709 All Good Things (2010) [u'Crime', u'Drama', u'Mystery', u'Romance', u'Thrill
er'] 101 mins. 6.3 35,395
tt0283509 No Man's Land (2001) [u'Drama', u'War'] 98 mins. 8.0 35,382
tt0892255 Che: Part One (2008) [u'Biography', u'Drama', u'History', u'War'] 134 mins.
7.2 35,377
tt0118771 Breakdown (1997) [u'Action', u'Crime', u'Drama', u'Mystery', u'Thriller'] 93
mins. 6.9 35,373
tt0089907 The Return of the Living Dead (1985) [u'Comedy', u'Horror'] 91 mins. 7.3 3
5,354
tt0875034 Nine (2009) [u'Drama', u'Musical', u'Romance'] 118 mins. 5.9 35,323
tt0279113 The Good Girl (2002) [u'Comedy', u'Drama', u'Romance'] 93 mins. 6.5 35,323
tt0116277 The Fan (1996) [u'Action', u'Drama', u'Sport', u'Thriller'] 116 mins. 5.8 3
5,319
tt1308729 Bullet to the Head (2012) [u'Action', u'Crime', u'Thriller'] 92 mins. 5.7 3
5,306
tt0115956 Courage Under Fire (1996) [u'Drama', u'Mystery', u'Thriller', u'War'] 117 mi
ns. 6.6 35,311
tt0977855 Fair Game (2010) [u'Biography', u'Drama', u'Thriller'] 108 mins. 6.8 35,287
tt0107554 Menace II Society (1993) [u'Crime', u'Drama', u'Thriller'] 97 mins. 7.5 35,2
74
tt0160797 Rules of Engagement (2000) [u'Action', u'Drama', u'War'] 128 mins. 6.4 35,26
2
tt0120731 The Legend of 1900 (1998) [u'Drama', u'Music', u'Romance'] 165 mins. 8.1 3
5,223
tt0082158 Chariots of Fire (1981) [u'Drama', u'History', u'Sport'] 124 mins. 7.2 35,21
7
tt0430304 Littleman (2006) [u'Comedy', u'Crime', u'Fantasy'] 98 mins. 4.2 35,192
tt0929425 Gomorrah (2008) [u'Crime', u'Drama'] 137 mins. 7.0 35,170
tt1439572 Perfect Sense (2011) [u'Drama', u'Romance', u'Sci-Fi'] 92 mins. 7.1 35,141
tt2402157 The November Man (2014) [u'Action', u'Crime', u'Thriller'] 108 mins. 6.3 3
5,189
tt0416044 Mongol: The Rise of Genghis Khan (2007) [u'Adventure', u'Biography', u'Dram
a', u'History', u'War'] 126 mins. 7.3 35,115
tt0240468 Kung Pow: Enter the Fist (2002) [u'Action', u'Comedy'] 81 mins. 6.2 35,112
tt1205535 The Rebound (2009) [u'Comedy', u'Romance'] 95 mins. 6.4 35,113
tt2980648 The Hundred-Foot Journey (2014) [u'Comedy', u'Drama'] 122 mins. 7.3 35,170
tt0046911 Diabolique (1955) [u'Horror', u'Mystery', u'Thriller'] 116 mins. 8.2 35,093
tt0457433 Perfect Stranger (2007) [u'Crime', u'Mystery', u'Thriller'] 109 mins. 5.7 3
5,063
tt0857191 The Visitor (2007) [u'Drama'] 104 mins. 7.7 35,043
tt0107659 Loaded Weapon 1 (1993) [u'Action', u'Comedy', u'Crime', u'Thriller'] 84 min
s. 6.1 35,004
tt0109424 Chungking Express (1994) [u'Drama', u'Mystery', u'Romance'] 98 mins. 8.1 3
5,001
```

Scrape to a file

Let us now scrape the first 10,000 results into a tab delimited file:

In [84]:

```
with open ('../data/imdb.txt','a') as file:
    for start in range(1,10000,50):
        params = dict(sort='num_votes,desc', start=start, title_type='feature', year='1940,2015')
        r = requests.get(url, params=params)
        dom = web.Element(r.text)
        for movie in dom.by_tag('tr.detailed'):
            title = movie.by_tag('td.title')[0]
            imdbID = title.by_tag('a')[0].href.replace('/title/', '').strip('/')
            name = web.plaintext(title.by_tag('a')[0].content).encode('utf8')
            year = title.by_tag('span.year_type')[0].content
            genres = title.by_tag('span.genre')[0].by_tag('a')
            genres =  "|".join([g.content for g in genres])
            try:
                runtime = title.by_tag('span.runtime')[0].content
            except IndexError:
                runtime = "NA"
            rating = title.by_tag('span.value')[0].content
            votes = movie.by_tag('td.sort_col')[0].content
            file.write("{}\t{} {}\t{}\t{}\t{}\t{}\t{}\n".format(imdbID, name, year, year, rating,
runtime, genres, votes))
```

Using BeautifulSoup

Using `find` and `findAll`:

In [59]:

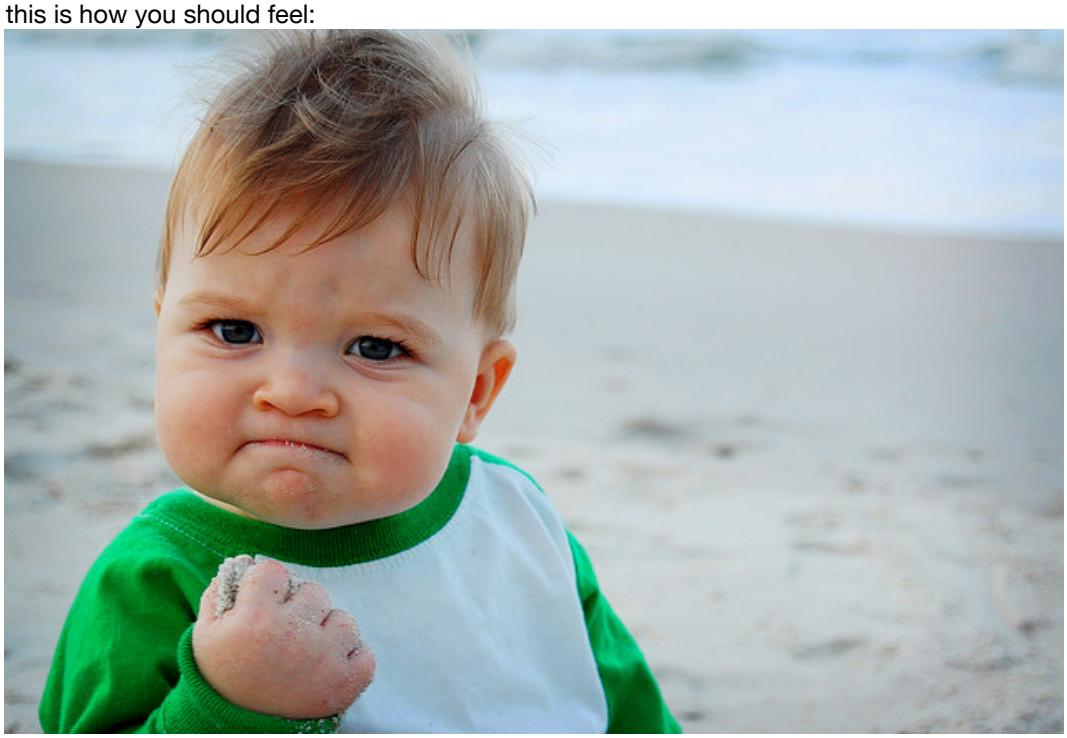
```
bs = BeautifulSoup(r.text)

for movie in bs.findAll('td', 'title'):
    title = movie.find('a').contents[0]
    genres = movie.find('span', 'genre').findAll('a')
    genres = [g.contents[0] for g in genres]
    runtime = movie.find('span', 'runtime').contents[0]
    rating = movie.find('span', 'value').contents[0]
    print title, genres, runtime, rating
```

The Shawshank Redemption [u'Crime', u'Drama'] 142 mins. 9.3
The Dark Knight [u'Action', u'Crime', u'Drama'] 152 mins. 9.0
Inception [u'Action', u'Mystery', u'Sci-Fi', u'Thriller'] 148 mins. 8.8
Fight Club [u'Drama'] 139 mins. 8.9
Pulp Fiction [u'Crime', u'Drama'] 154 mins. 8.9
The Lord of the Rings: The Fellowship of the Ring [u'Adventure', u'Fantasy'] 178 mins. 8.8
The Lord of the Rings: The Return of the King [u'Adventure', u'Fantasy'] 201 mins. 8.9
The Matrix [u'Action', u'Sci-Fi'] 136 mins. 8.7
Forrest Gump [u'Drama', u'Romance'] 142 mins. 8.8
The Godfather [u'Crime', u'Drama'] 175 mins. 9.2
The Dark Knight Rises [u'Action', u'Thriller'] 165 mins. 8.5
The Lord of the Rings: The Two Towers [u'Adventure', u'Fantasy'] 179 mins. 8.8
Se7en [u'Drama', u'Mystery', u'Thriller'] 127 mins. 8.7
Gladiator [u'Action', u'Drama'] 155 mins. 8.5
Batman Begins [u'Action', u'Adventure'] 140 mins. 8.3
The Avengers [u'Action', u'Adventure', u'Sci-Fi'] 143 mins. 8.2
Avatar [u'Action', u'Adventure', u'Fantasy', u'Sci-Fi'] 162 mins. 7.9
Django Unchained [u'Western'] 165 mins. 8.5
Star Wars [u'Action', u'Adventure', u'Fantasy', u'Sci-Fi'] 121 mins. 8.7
Saving Private Ryan [u'Action', u'Drama', u'War'] 169 mins. 8.6
The Departed [u'Crime', u'Drama', u'Thriller'] 151 mins. 8.5
The Silence of the Lambs [u'Drama', u'Thriller'] 118 mins. 8.6
Schindler's List [u'Biography', u'Drama', u'History'] 195 mins. 8.9
Memento [u'Mystery', u'Thriller'] 113 mins. 8.5
Inglourious Basterds [u'Adventure', u'Drama', u'War'] 153 mins. 8.3
American Beauty [u'Drama'] 122 mins. 8.4
The Prestige [u'Drama', u'Mystery', u'Thriller'] 130 mins. 8.5
Pirates of the Caribbean: The Curse of the Black Pearl [u'Adventure', u'Fantasy'] 143 mins. 8.1
Titanic [u'Drama', u'Romance'] 194 mins. 7.7
V for Vendetta [u'Action', u'Drama', u'Thriller'] 132 mins. 8.2
Star Wars: Episode V - The Empire Strikes Back [u'Action', u'Adventure', u'Fantasy', u'Sci-Fi'] 124 mins. 8.8
American History X [u'Crime', u'Drama'] 119 mins. 8.6
The Godfather: Part II [u'Crime', u'Drama'] 200 mins. 9.1
The Green Mile [u'Crime', u'Drama', u'Fantasy', u'Mystery'] 189 mins. 8.5
Shutter Island [u'Mystery', u'Thriller'] 138 mins. 8.1
The Usual Suspects [u'Crime', u'Drama', u'Thriller'] 106 mins. 8.7
Braveheart [u'Action', u'Biography', u'Drama', u'History', u'War'] 177 mins. 8.4
Terminator 2: Judgment Day [u'Action', u'Sci-Fi'] 137 mins. 8.5
Goodfellas [u'Biography', u'Crime', u'Drama'] 146 mins. 8.7
Kill Bill: Vol. 1 [u'Action'] 111 mins. 8.1
The Sixth Sense [u'Drama', u'Mystery', u'Thriller'] 107 mins. 8.2
Léon: The Professional [u'Crime', u'Drama', u'Thriller'] 110 mins. 8.6
The Hunger Games [u'Adventure', u'Sci-Fi'] 142 mins. 7.3
Back to the Future [u'Adventure', u'Comedy', u'Sci-Fi'] 116 mins. 8.5
WALL·E [u'Animation', u'Adventure', u'Sci-Fi'] 98 mins. 8.4
One Flew Over the Cuckoo's Nest [u'Drama'] 133 mins. 8.7
Sin City [u'Crime', u'Thriller'] 124 mins. 8.1
Iron Man [u'Action', u'Adventure', u'Sci-Fi'] 126 mins. 7.9
Interstellar [u'Adventure', u'Sci-Fi'] 169 mins. 8.7
The Wolf of Wall Street [u'Biography', u'Comedy', u'Crime', u'Drama'] 180 mins. 8.2

All scrapping follows the same pattern. Finding out which DOM tag we need and then looping over and stripping out everything else.

At the end this is how you should feel:



Wrangling Demo

To get started let's import the libraries we will need and minimize matplotlib's chart junk:

In [2]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

#tell pandas to display wide tables as pretty HTML tables
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 100)

def remove_border(axes=None, top=False, right=False, left=True, bottom=True):
    """
    Minimize chartjunk by stripping out unnecesasry plot borders and axis ticks

    The top/right/left/bottom keywords toggle whether the corresponding plot border is drawn
    """
    ax = axes or plt.gca()
    ax.spines['top'].set_visible(top)
    ax.spines['right'].set_visible(right)
    ax.spines['left'].set_visible(left)
    ax.spines['bottom'].set_visible(bottom)

    #turn off all ticks
    ax.yaxis.set_ticks_position('none')
    ax.xaxis.set_ticks_position('none')

    #now re-enable visibles
    if top:
        ax.xaxis.tick_top()
    if bottom:
        ax.xaxis.tick_bottom()
    if left:
        ax.yaxis.tick_left()
    if right:
        ax.yaxis.tick_right()
```

The basic workflow pattern is as follows:

1. **Build** a DataFrame from the data (ideally, put *all* data in this object)
2. **Clean** the DataFrame. It should have the following properties:
    - Each row describes a single object
    - Each column describes a property of that object
    - Columns are numeric whenever appropriate
    - Columns contain atomic properties that cannot be further decomposed
3. Explore **global properties**. Use histograms, scatter plots, and aggregation functions to summarize the data.
4. Explore **group properties**. Use `groupby` and small multiples to compare subsets of the data.

This process transforms your data into a format which is easier to work with, gives you a basic overview of the data's properties, and likely generates several questions for you to followup in subsequent analysis.

Here's a preview of the raw data we'll use -- it's a list of the 10,000 movies made since 1940 with the most IMDB user ratings.

In [3]:

```
!head ../data/imdb_top_10000.txt
```

```
tt0111161	The Shawshank Redemption (1994) (1994)	9.3	142 mins.	Crim
e|Drama 1,434,256
tt0468569	The Dark Knight (2008)	(2008)	9.0	152 mins.	Action|Crime|D
rama	1,404,761
tt1375666	Inception (2010)	(2010)	8.8	148 mins.	Action|Myster
y|Sci-Fi|Thriller	1,203,863
tt0137523	Fight Club (1999)	(1999)	8.9	139 mins.	Drama	1,11
8,852
tt0110912	Pulp Fiction (1994)	(1994)	8.9	154 mins.	Crime|Drama
1,113,162
tt0120737	The Lord of the Rings: The Fellowship of the Ring (2001)	(2001)
8.8	178 mins.	Adventure|Fantasy	1,052,988
tt0167260	The Lord of the Rings: The Return of the King (2003)	(2003)	8.9
201 mins.	Adventure|Fantasy	1,026,828
tt0133093	The Matrix (1999)	(1999)	8.7	136 mins.	Action|Sci-Fi
1,023,145
tt0109830	Forrest Gump (1994)	(1994)	8.8	142 mins.	Drama|Romance
1,018,579
tt0068646	The Godfather (1972)	(1972)	9.2	175 mins.	Crime|Drama
985,270
```

Notice the runtime is currently a string rather than an integer number. The genres column is not atomic. Finally the movie year is repeated in the year and title column and the votes are represented as formated integers.

1. Build a DataFrame

The textfile is tab-separated, and doesn't have any column headers. We set the appropriate keywords in `pd.read_csv` to handle this:

In [4]:

```
col_names = ['imdbID', 'title', 'year', 'score','runtime','genres','votes']
data = pd.read_csv('../data/imdb_top_10000.txt', delimiter='\t', names=col_names, index_col='imdb
ID',na_values=['NA'])
print "Number of rows: %i" % data.shape[0]
data.head()
```

```
Number of rows: 10000
```

Out[4]:

	title	year	score	runtime	genres	votes
imdbID						
tt0111161	The Shawshank Redemption (1994)	(1994)	9.3	142 mins.	Crime|Drama	1,434,256
tt0468569	The Dark Knight (2008)	(2008)	9.0	152 mins.	Action|Crime|Drama	1,404,761
tt1375666	Inception (2010)	(2010)	8.8	148 mins.	Action|Mystery|Sci-Fi|Thriller	1,203,863
tt0137523	Fight Club (1999)	(1999)	8.9	139 mins.	Drama	1,118,852
tt0110912	Pulp Fiction (1994)	(1994)	8.9	154 mins.	Crime|Drama	1,113,162

2. Clean the DataFrame

There are several problems with the DataFrame at this point:

1. The runtime column describes a number, but is stored as a string
2. The genres column is not atomic -- it aggregates several genres together. This makes it hard, for example, to extract which movies are Comedies.
3. The movie year is repeated in the title and year column

Fixing the runtime column

The following snipptet converts a string like '142 mins.' to the number 142:

In [5]:

```
runtime = '142 mins.'
number, unit = runtime.split(' ')
print int(number)
```

```
142
```

We can package this up into a list comprehension:

In [6]:

```
data['runtime'] = data['runtime'].dropna().map(lambda x: int(x.split(' ')[0]))
print "Number of rows: %i" % data.shape[0]
data.head()
```

```
Number of rows: 10000
```

Out[6]:

	title	year	score	runtime	genres	votes
imdbID						
tt0111161	The Shawshank Redemption (1994)	(1994)	9.3	142	Crime|Drama	1,434,256
tt0468569	The Dark Knight (2008)	(2008)	9.0	152	Action|Crime|Drama	1,404,761
tt1375666	Inception (2010)	(2010)	8.8	148	Action|Mystery|Sci-Fi|Thriller	1,203,863
tt0137523	Fight Club (1999)	(1999)	8.9	139	Drama	1,118,852
tt0110912	Pulp Fiction (1994)	(1994)	8.9	154	Crime|Drama	1,113,162

Splitting up the genres

We can use the concept of *indicator variables* to split the genres column into many columns. Each new column will correspond to a single genre, and each cell will be True or False.

In [7]:

```
#determine all the unique genres
genres = set()
for m in data.genres:
    genres.update(g for g in m.split('|'))

genres = sorted(genres)

#make a column for each genres
for genre in genres:
    data[genre] = [genre in m.split('|') for m in data.genres]

data.head()
```

Out[7]:

	title	year	score	runtime	genres	votes	Action	Adult	Adventure	Ani
imdbID										
tt0111161	The Shawshank Redemption	(1994)	9.3	142	Crime|Drama	1,434,256	False	False	False	Fals
tt0468569	The Dark Knight	(2008)	9.0	152	Action|Crime|Drama	1,404,761	True	False	False	Fals
tt1375666	Inception	(2010)	8.8	148	Action|Mystery|Sci-Fi|Thriller	1,203,863	True	False	False	Fals
tt0137523	Fight Club	(1999)	8.9	139	Drama	1,118,852	False	False	False	Fals
tt0110912	Pulp Fiction	(1994)	8.9	154	Crime|Drama	1,113,162	False	False	False	Fals

Removing year from the title

We can fix each element by stripping off the last 7 characters:

In [8]:

```
data['title'] = [t[0:-7] for t in data.title]
data.head()
```

Out[8]:

	title	year	score	runtime	genres	votes	Action	Adult	Adventure	Ani
imdbID										
tt0111161	The Shawshank Redemption	(1994)	9.3	142	Crime|Drama	1,434,256	False	False	False	Fals
tt0468569	The Dark Knight	(2008)	9.0	152	Action|Crime|Drama	1,404,761	True	False	False	Fals
tt1375666	Inception	(2010)	8.8	148	Action|Mystery|Sci-Fi|Thriller	1,203,863	True	False	False	Fals
tt0137523	Fight Club	(1999)	8.9	139	Drama	1,118,852	False	False	False	Fals
tt0110912	Pulp Fiction	(1994)	8.9	154	Crime|Drama	1,113,162	False	False	False	Fals

Removing brackets around year

We can fix each element by getting the number from within the brackets:

In [9]:

```
data['year'] = [int(y.replace('(', '').strip(')')) for y in data.year]
data.head()
```

Out[9]:

	title	year	score	runtime	genres	votes	Action	Adult	Adventure	Anin
imdbID										
tt0111161	The Shawshank Redemption	1994	9.3	142	Crime|Drama	1,434,256	False	False	False	Fals
tt0468569	The Dark Knight	2008	9.0	152	Action|Crime|Drama	1,404,761	True	False	False	Fals
tt1375666	Inception	2010	8.8	148	Action|Mystery|Sci-Fi|Thriller	1,203,863	True	False	False	Fals
tt0137523	Fight Club	1999	8.9	139	Drama	1,118,852	False	False	False	Fals
tt0110912	Pulp Fiction	1994	8.9	154	Crime|Drama	1,113,162	False	False	False	Fals

Fixing votes format

We can fix each vote similarly to above:

In [10]:

```
data['votes'] = [int(v.replace(',', '')) for v in data.votes]
data.head()
```

Out[10]:

	title	year	score	runtime	genres	votes	Action	Adult	Adventure	Anim
imdbID										
tt0111161	The Shawshank Redemption	1994	9.3	142	Crime|Drama	1434256	False	False	False	False
tt0468569	The Dark Knight	2008	9.0	152	Action|Crime|Drama	1404761	True	False	False	False
tt1375666	Inception	2010	8.8	148	Action|Mystery|Sci-Fi|Thriller	1203863	True	False	False	False
tt0137523	Fight Club	1999	8.9	139	Drama	1118852	False	False	False	False
tt0110912	Pulp Fiction	1994	8.9	154	Crime|Drama	1113162	False	False	False	False

3. Explore Global Properties

Next, we get a handle on some basic, global summaries of the DataFrame.

Call `describe` on relevant columns

In [11]:

```
data[['year','score','votes','runtime']].describe()
```

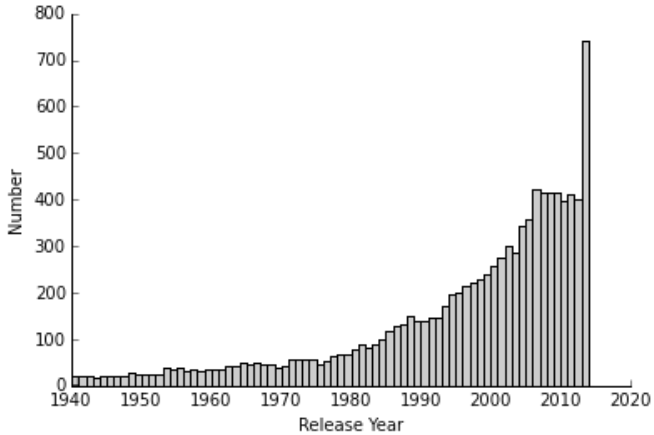
Out[11]:

	year	score	votes	runtime
count	10000.000000	10000.000000	10000.000000	9968.000000
mean	1995.985500	6.575040	43533.835700	107.854133
std	16.621068	1.086435	87569.574657	21.555782
min	1940.000000	1.600000	3401.000000	46.000000
25%	1989.000000	6.000000	5830.750000	94.000000
50%	2001.000000	6.700000	12812.000000	104.000000
75%	2008.000000	7.400000	40203.750000	117.000000
max	2015.000000	9.400000	1434256.000000	450.000000

Make some basic plots

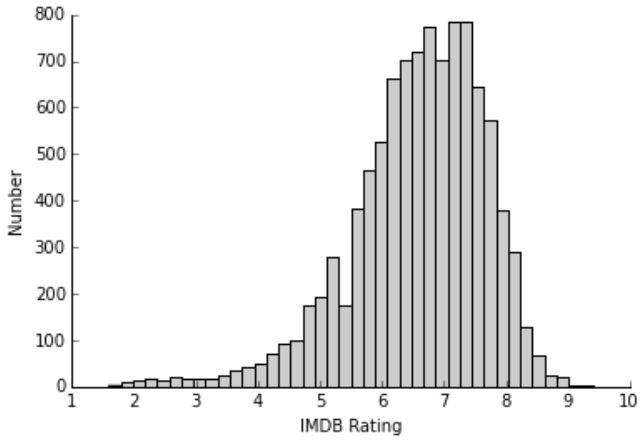
In [12]:

```
plt.hist(data.year,bins=np.arange(1940,2015),color='#cccccc')
plt.xlabel("Release Year")
plt.ylabel("Number")
remove_border()
```



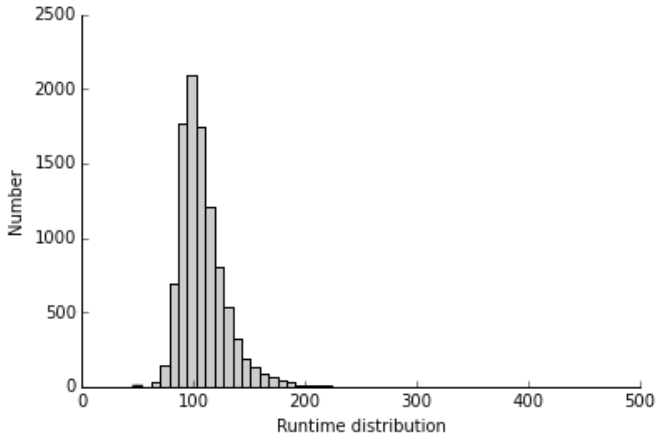
In [13]:

```
plt.hist(data.score, bins=40, color='#cccccc')
plt.xlabel("IMDB Rating")
plt.ylabel("Number")
remove_border()
```



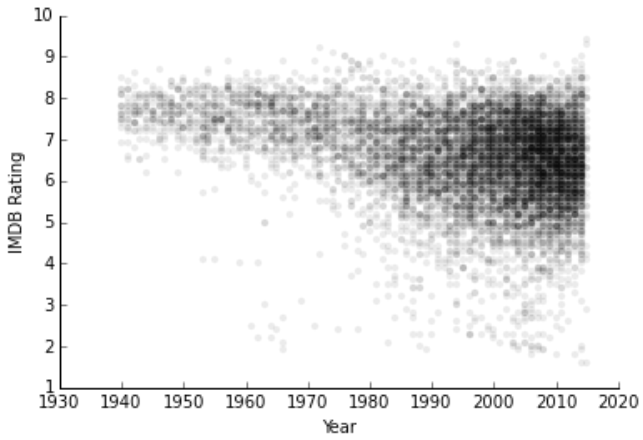
In [14]:

```
plt.hist(data.runtime.dropna(), bins=50, color='#cccccc')
plt.xlabel("Runtime distribution")
plt.ylabel("Number")
remove_border()
```



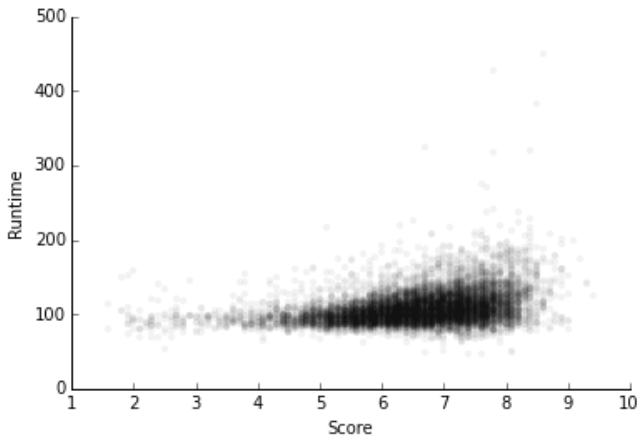
In [15]:

```
plt.scatter(data.year, data.score, lw=0, alpha=.08, color='k')
plt.xlabel("Year")
plt.ylabel("IMDB Rating")
remove_border()
```



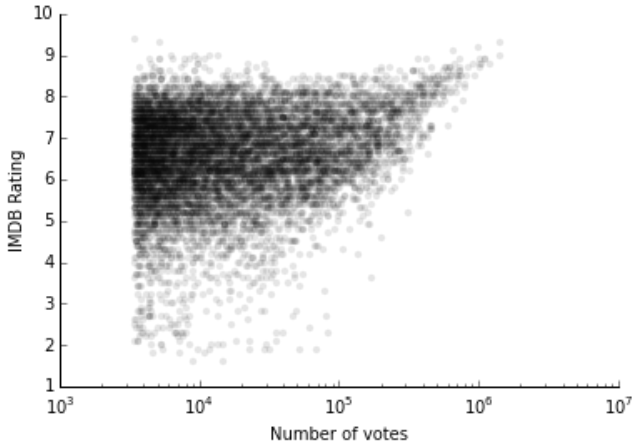
In [30]:

```
plt.scatter(data.score, data.runtime, lw=0, alpha=.05, color='k')
plt.xlabel("Score")
plt.ylabel("Runtime")
remove_border()
```



In [16]:

```
plt.scatter(data.votes, data.score, lw=0, alpha=.1, color='k')
plt.xlabel("Number of votes")
plt.ylabel("IMDB Rating")
plt.xscale('log')
remove_border()
```



Identify some outliers

In [192]:

```
data[(data.votes > 9e4) & (data.score < 5)][['title','year','score','votes','genres']]
```

Out[192]:

	title	year	score	votes	genres
imdbID					
tt1259571	The Twilight Saga: New Moon	2009	4.6	201124	Adventure|Drama|Fantasy|Romance
tt0118688	Batman & Robin	1997	3.6	169303	Action
tt1325004	The Twilight Saga: Eclipse	2010	4.9	165479	Adventure|Drama|Fantasy|Romance
tt1324999	The Twilight Saga: Breaking Dawn - Part 1	2011	4.9	163994	Adventure|Drama|Fantasy|Romance
tt2322441	Fifty Shades of Grey	2015	4.2	138293	Drama|Romance
tt0120891	Wild Wild West	1999	4.8	119127	Action|Western|Comedy|Sci-Fi
tt0938283	The Last Airbender	2010	4.3	106900	Action|Adventure|Family|Fantasy
tt0305357	Charlie's Angels: Full Throttle	2003	4.8	93240	Action|Adventure|Comedy|Crime

In [193]:

```
data[data.score == data.score.min()][['title','year','score','votes','genres']]
```

Out[193]:

	title	year	score	votes	genres
imdbID					
tt4458206	Kod Adi K.O.Z.	2015	1.6	14729	Crime|Mystery
tt4009460	Saving Christmas	2014	1.6	9002	Comedy|Family

In [194]:

```
data[data.score == data.score.max()][['title','year','score','votes','genres']]
```

Out[194]:

	title	year	score	votes	genres
imdbID					
tt4523112	Selam: Bahara Yolculuk	2015	9.4	3473	Biography|Drama|History

In [217]:

```
data[data.runtime == data.runtime.max()][['title','year','runtime','score','votes','genres']]
```

Out[217]:

	title	year	runtime	score	votes	genres
imdbID						
tt0111341	Satantango	1994	450	8.6	5300	Comedy|Drama

Run aggregation functions like `sum` over several rows or columns

What genres are the most frequent?

In [195]:

```
genres_count = pd.DataFrame({'Genre Count': data[genres].sum()})
genres_count.sort(columns='Genre Count',ascending=False)
```

Out[195]:

	Genre Count
Drama	5573
Comedy	3834
Thriller	2871
Romance	2306
Action	2005
Crime	1885
Adventure	1356
Horror	1223
Mystery	1014
Sci-Fi	959
Fantasy	868
Family	752
War	480
Biography	456
Animation	389
History	366
Music	330
Sport	300
Musical	261
Western	210
Film-Noir	78
Adult	1

How many genres does a movie have, on average?

In [196]:

```
genre_count = data[genres].sum(axis=1)
print "Average movie has %0.2f genres" % genre_count.mean()
genre_count.describe()
```

```
Average movie has 2.75 genres
```

Out[196]:

```
count    10000.000000
mean         2.751700
std          1.169779
min          1.000000
25%          2.000000
50%          3.000000
75%          3.000000
max         10.000000
dtype: float64
```

Explore Group Properties

Let's split up movies by half-decades:

In [233]:

```
hdecades = (data.year //5)*5

tyd = data[['title', 'year']]
tyd['hdecade'] = hdecades

tyd.head()
```

```
/usr/local/lib/python2.7/site-packages/IPython/kernel/__main__.py:4: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/indexing.html#indexing-view-versus-copy
```

Out[233]:

	title	year	hdecade
imdbID			
tt0111161	The Shawshank Redemption	1994	1990
tt0468569	The Dark Knight	2008	2005
tt1375666	Inception	2010	2010
tt0137523	Fight Club	1999	1995
tt0110912	Pulp Fiction	1994	1990

GroupBy (http://pandas.pydata.org/pandas-docs/dev/groupby.html) will gather movies into groups with equal decade values:

In [209]:

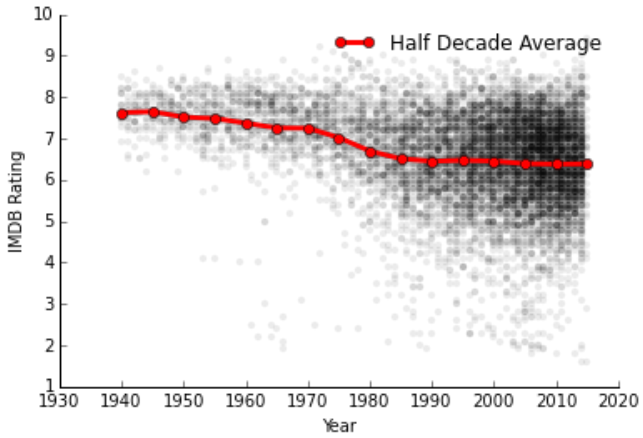
```
hdecade_mean = data.groupby(hdecades).score.mean()
hdecade_mean.name = 'Half Decade mean score'
print hdecade_mean
```

```
year
1940    7.623469
1945    7.647863
1950    7.514685
1955    7.484971
1960    7.370588
1965    7.254709
1970    7.249434
1975    7.018855
1980    6.694266
1985    6.508683
1990    6.450311
1995    6.472678
2000    6.454241
2005    6.390976
2010    6.378626
2015    6.385294
Name: Half Decade mean score, dtype: float64
```

We can plot the mean score over the scatter plot:

In [210]:

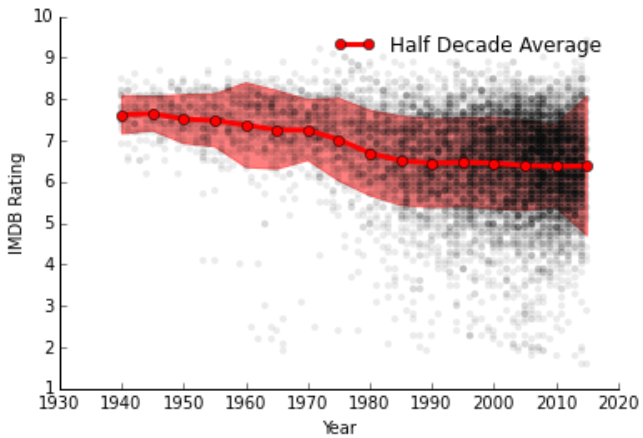
```
plt.plot(hdecade_mean.index, hdecade_mean.values, 'o-', color='r',lw=3, label='Half Decade Averag
e')
plt.scatter(data.year, data.score, lw=0, alpha=.08, color='k')
plt.xlabel("Year")
plt.ylabel("IMDB Rating")
plt.legend(frameon=False)
remove_border()
```



We can go one further, and compute the scatter in each year as well:

In [211]:

```
hdecade_std = data.groupby(decades).score.std()
plt.plot(hdecade_mean.index, hdecade_mean.values, 'o-', color='r',lw=3, label='Half Decade Averag
e')
plt.fill_between(hdecade_mean.index, (hdecade_mean + hdecade_std).values, (hdecade_mean - hdecad
e_std).values, color='r', alpha=0.5)
plt.scatter(data.year, data.score, lw=0, alpha=.08, color='k')
plt.xlabel("Year")
plt.ylabel("IMDB Rating")
plt.legend(frameon=False)
remove_border()
```



You can also iterate over a `GroupBy` object. Each iteration yields two variables: one of the distinct values of the group key, and the subset of the dataframe where the key equals that value. To find the most popular movie each year:

In [212]:

```
for year, subset in data.groupby('year'):
    print year, subset[subset.score == subset.score.max()].title.values
```

```
1940 ['The Great Dictator']
1941 ['Citizen Kane']
1942 ['Casablanca']
1943 ['The Ox-Bow Incident' 'The Life and Death of Colonel Blimp']
```

```
1944 ['Double Indemnity']
1945 ['Children of Paradise']
1946 ["It's a Wonderful Life"]
1947 ['Out of the Past']
1948 ['Bicycle Thieves' 'I Remember Mama']
1949 ['The Third Man']
1950 ['Sunset Blvd.']
1951 ['Ace in the Hole' 'Early Summer']
1952 ["Singin' in the Rain"]
1953 ['The Wages of Fear' 'Tokyo Story']
1954 ['Seven Samurai']
1955 ['Diabolique' 'Rififi' 'Pather Panchali']
1956 ['A Man Escaped']
1957 ['12 Angry Men']
1958 ['Vertigo']
1959 ['North by Northwest']
1960 ['Psycho']
1961 ['Yojimbo']
1962 ['Harakiri']
1963 ['High and Low']
1964 ['Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb']
1965 ["Operation 'Y' & Other Shurik's Adventures"]
1966 ['The Good, the Bad and the Ugly']
1967 ['Samurai Rebellion' 'Kidnapping, Caucasian Style']
1968 ['Once Upon a Time in the West']
1969 ['The Diamond Arm']
1970 ['Michael the Brave']
1971 ['Anand']
1972 ['The Godfather']
1973 ['My Dear Brother']
1974 ['The Godfather: Part II']
1975 ["One Flew Over the Cuckoo's Nest"]
1976 ['The Chaos Class Failed the Class' 'The Foster Brothers']
1977 ['The Chaos Class Is Waking Up' 'Saban, Son of Saban']
1978 ['The Chaos Class Is on Vacation' 'The Girl with the Red Scarf'
 'Happy Days']
1979 ['Gol Maal']
1980 ["Who's Singin' Over There?"]
1981 ['Raiders of the Lost Ark']
1982 ['The Marathon Family']
1983 ['Who Pays the Piper']
1984 ['Balkan Spy']
1985 ['The Broken Landlord']
1986 ['Aliens' 'Cat City']
1987 ['Nayakan']
1988 ['Cinema Paradiso' 'Grave of the Fireflies']
1989 ["Don't Let Them Shoot the Kite"]
1990 ['Goodfellas']
1991 ['The Silence of the Lambs']
1992 ['We Are Not Angels']
1993 ["Schindler's List"]
1994 ['The Shawshank Redemption']
1995 ['Se7en' 'The Usual Suspects']
1996 ['The Bandit' 'Pretty Village, Pretty Flame']
1997 ['Life Is Beautiful']
1998 ['Saving Private Ryan' 'American History X']
1999 ['Fight Club']
2000 ['Gladiator' 'Memento' 'Hera Pheri' "A Dog's Will"]
2001 ['The Lord of the Rings: The Fellowship of the Ring']
2002 ['The Lord of the Rings: The Two Towers']
2003 ['The Lord of the Rings: The Return of the King' 'Love is God']
2004 ['Black Friday' 'The Lizard']
2005 ['Athadu']
2006 ['The Departed' 'The Prestige' 'The Lives of Others']
2007 ['Like Stars on Earth']
2008 ['The Dark Knight']
2009 ['3 Idiots']
2010 ['Inception']
2011 ['The Intouchables']
2012 ['The Dark Knight Rises' 'Django Unchained']
2013 ['Drishyam']
2014 ['Chaar Sahibzaade']
2015 ['Selam: Bahara Yolculuk']
```

Small multiples plots

Let's split up the movies by genre, and look at how their release year/runtime/IMDB score vary. The distribution for all movies is shown as a grey background.

This isn't a standard groupby, so we can't use the `groupby` method here. A manual loop is needed.

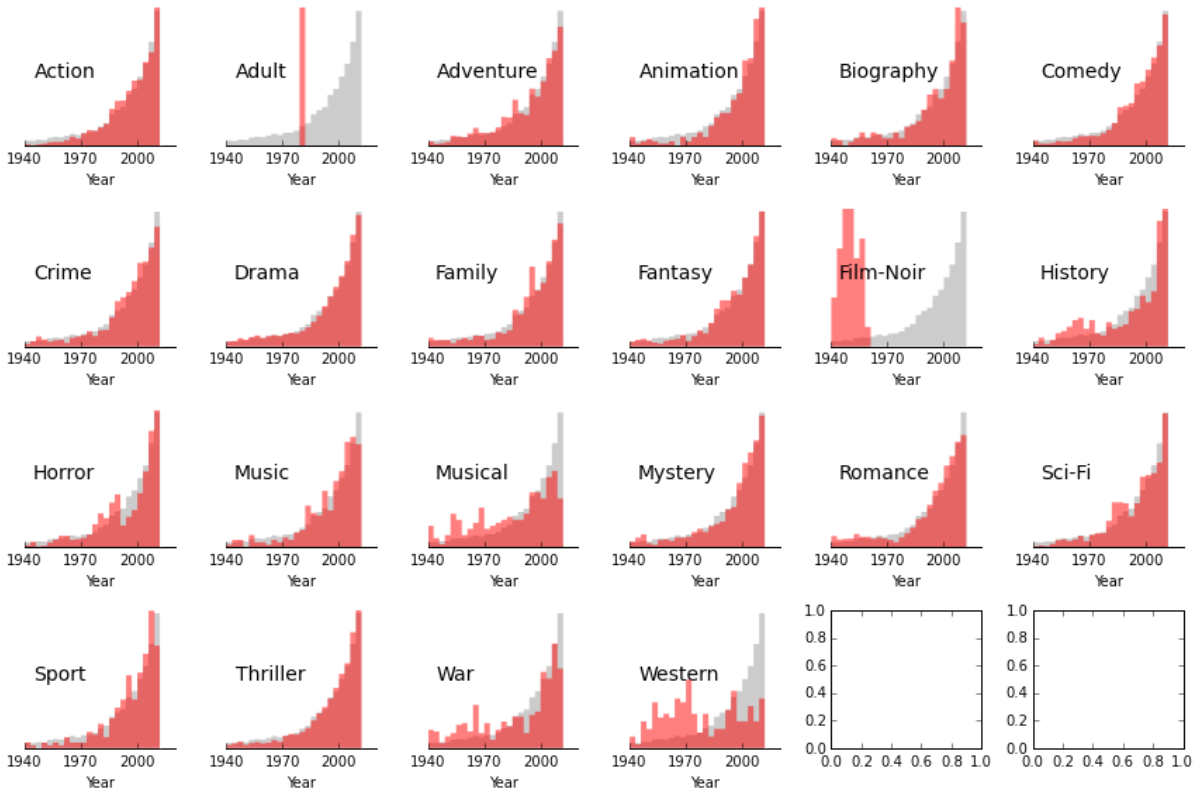
In [18]:

```
fig, axes = plt.subplots(nrows=4, ncols=6, figsize=(12,8), tight_layout=True)

bins = np.arange(1940, 2015, 3)

for ax, genre in zip(axes.ravel(), genres):
    ax.hist(data[data[genre] == True].year, bins=bins,
            histtype='stepfilled', normed=True, color='r', ec='none', alpha=0.5)

    ax.hist(data.year, bins=bins, histtype='stepfilled', normed=True, ec='none', zorder=0, colo
r='#cccccc')
    remove_border(ax,left=False)
    ax.set_xlabel('Year')
    ax.xaxis.set_ticks(np.arange(1940, 2015, 30))
    ax.annotate(genre, xy=(1945, 3e-2), fontsize=14)
    ax.set_yticks([])
```



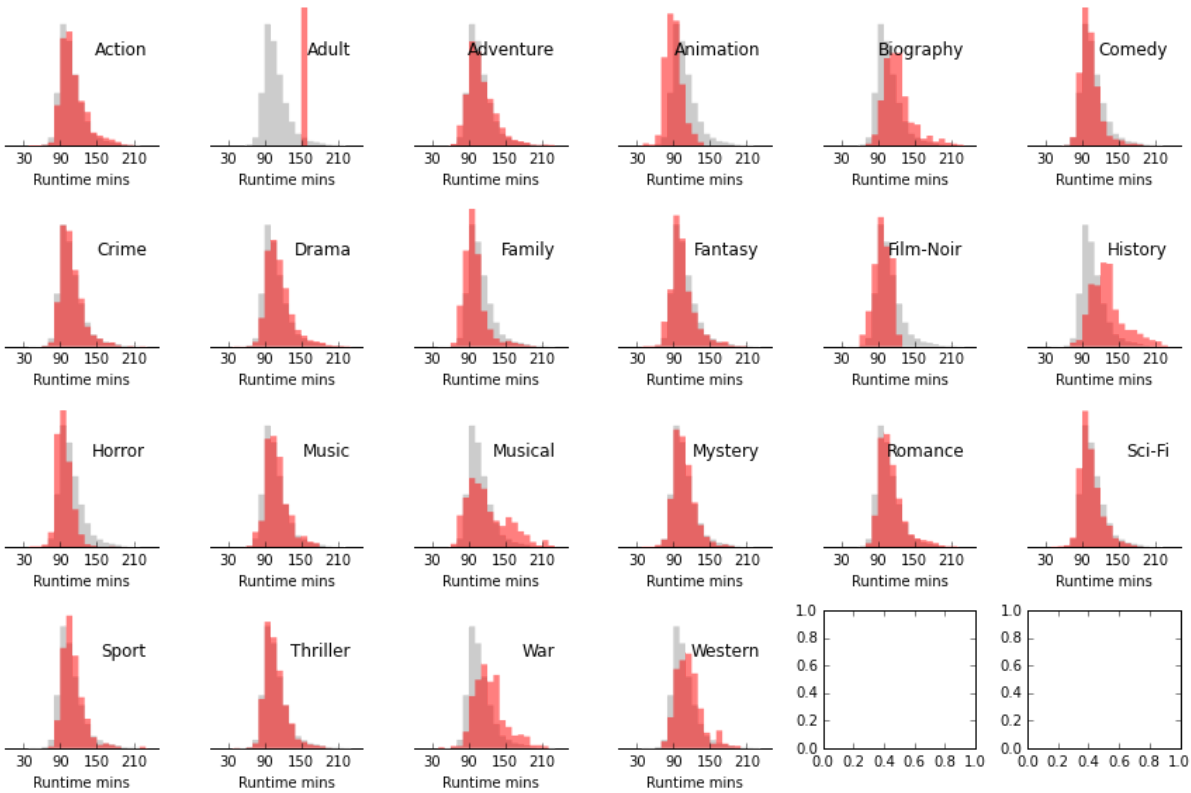
In [238]:

```
fig, axes = plt.subplots(nrows=4, ncols=6, figsize=(12,8), tight_layout=True)

bins = np.arange(30, 240, 10)

for ax, genre in zip(axes.ravel(), genres):
    ax.hist(data[data[genre] == True].runtime, bins=bins,
            histtype='stepfilled', normed=True, color='r', ec='none', alpha=0.5)

    ax.hist(data.runtime, bins=bins, histtype='stepfilled', normed=True, ec='none', zorder=0, col
or='#cccccc')
    remove_border(ax,left=False)
    ax.set_xlabel("Runtime mins")
    ax.xaxis.set_ticks(np.arange(30, 240, 60))
    ax.annotate(genre, xy=(230, .02), fontsize=12, ha='right')
    ax.set_yticks([])
```



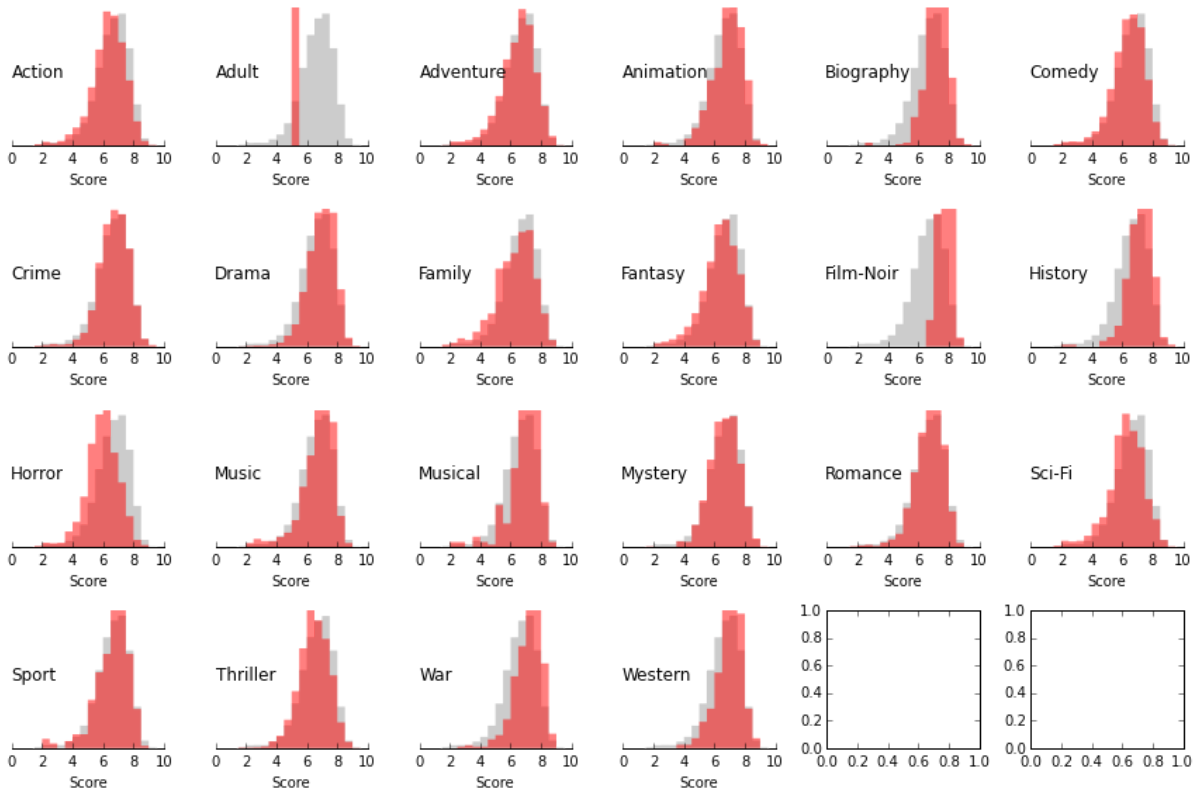
In [25]:

```
fig, axes = plt.subplots(nrows=4, ncols=6, figsize=(12,8), tight_layout=True)

bins = np.arange(0, 10, 0.5)

for ax, genre in zip(axes.ravel(), genres):
    ax.hist(data[data[genre] == True].score, bins=bins,
            histtype='stepfilled', normed=True, color='r', ec='none', alpha=0.5)

    ax.hist(data.score, bins=bins, histtype='stepfilled', normed=True, ec='none', zorder=0, colo
r='#cccccc')
    remove_border(ax,left=False)
    ax.set_xlabel("Score")
    ax.annotate(genre, xy=(0, .2), fontsize=12, ha='left')
    ax.set_ylim(0, .4)
    ax.set_yticks([])
```



css tweaks in this cell