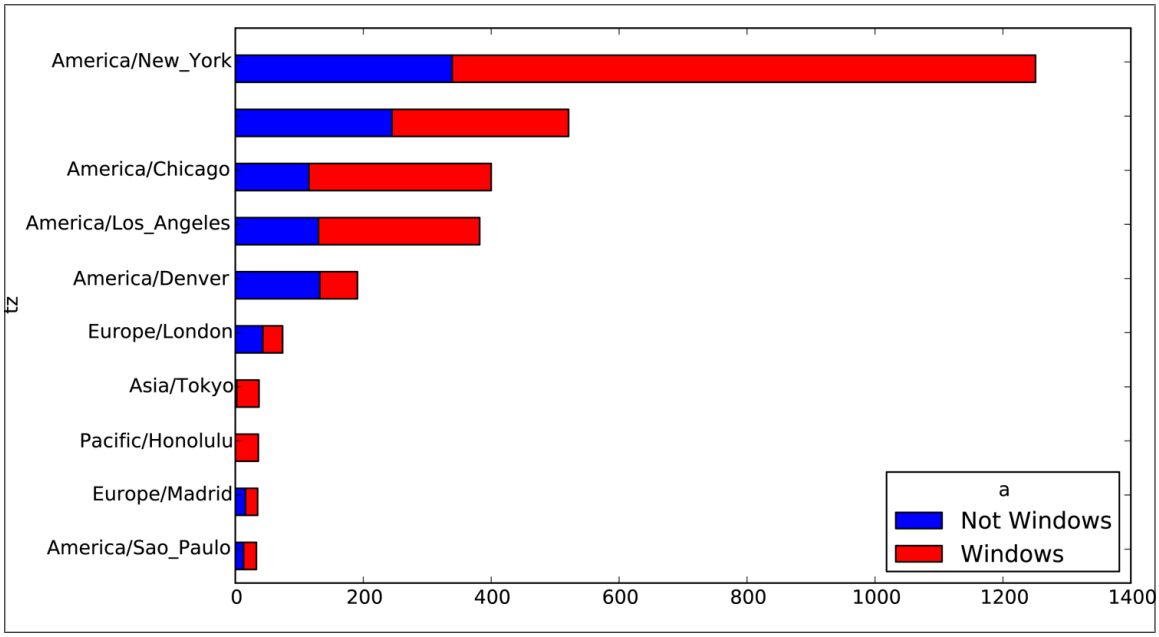


*Figure 2-2. Top time zones by Windows and non-Windows users*

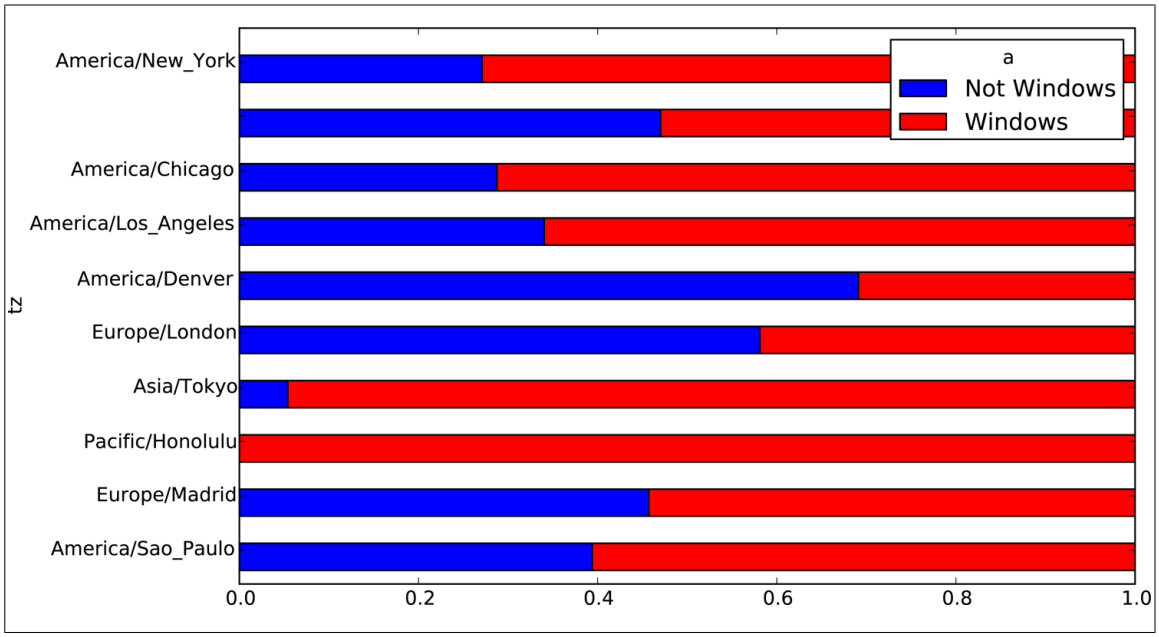


*Figure 2-3. Percentage Windows and non-Windows users in top-occurring time zones*

All of the methods employed here will be examined in great detail throughout the rest of the book.

# MovieLens 1M Data Set

GroupLens Research (*http://www.grouplens.org/node/73*) provides a number of collections of movie ratings data collected from users of MovieLens in the late 1990s and

early 2000s. The data provide movie ratings, movie metadata (genres and year), and demographic data about the users (age, zip code, gender, and occupation). Such data is often of interest in the development of recommendation systems based on machine learning algorithms. While I will not be exploring machine learning techniques in great detail in this book, I will show you how to slice and dice data sets like these into the exact form you need.

The MovieLens 1M data set contains 1 million ratings collected from 6000 users on 4000 movies. It's spread across 3 tables: ratings, user information, and movie information. After extracting the data from the zip file, each table can be loaded into a pandas DataFrame object using `pandas.read_table`:

```
import pandas as pd

unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('ml-1m/users.dat', sep='::', header=None,
                      names=unames)

rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('ml-1m/ratings.dat', sep='::', header=None,
                        names=rnames)

mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('ml-1m/movies.dat', sep='::', header=None,
                        names=mnames)
```

You can verify that everything succeeded by looking at the first few rows of each DataFrame with Python's slice syntax:

```
In [334]: users[:5]
Out[334]:
   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

In [335]: ratings[:5]
Out[335]:
   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 [336]: movies[:5]
Out[336]:
   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 [337]: ratings
Out[337]:
<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)
```

Note that ages and occupations are coded as integers indicating groups described in the data set's `README` file. Analyzing the data spread across three tables is not a simple task; for example, suppose you wanted to compute mean ratings for a particular movie by sex and age. As you will see, this is much easier to do with all of the data merged together into a single table. Using pandas's `merge` function, we first merge `ratings` with `users` then merging that result with the `movies` data. pandas infers which columns to use as the merge (or *join*) keys based on overlapping names:

```
In [338]: data = pd.merge(pd.merge(ratings, users), movies)

In [339]: data
Out[339]:
<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 [340]: data.ix[0]
Out[340]:
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 this form, aggregating the ratings grouped by one or more user or movie attributes is straightforward once you build some familiarity with pandas. To get mean movie ratings for each film grouped by gender, we can use the `pivot_table` method:

```
In [341]: mean_ratings = data.pivot_table('rating', index='title',
.....:                                     columns='gender', aggfunc='mean')

In [342]: mean_ratings[:5]
Out[342]:
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
```

This produced another DataFrame containing mean ratings with movie totals as row labels and gender as column labels. First, I'm going to filter down to movies that received at least 250 ratings (a completely arbitrary number); to do this, I group the data by title and use `size()` to get a Series of group sizes for each title:

```
In [343]: ratings_by_title = data.groupby('title').size()

In [344]: ratings_by_title[:10]
Out[344]:
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

In [345]: active_titles = ratings_by_title.index[ratings_by_title >= 250]

In [346]: active_titles
Out[346]:
Index(['burbs, The (1989), 10 Things I Hate About You (1999),
       101 Dalmatians (1961), ..., Young Sherlock Holmes (1985),
       Zero Effect (1998), eXistenZ (1999)], dtype=object)
```

The index of titles receiving at least 250 ratings can then be used to select rows from `mean_ratings` above:

```
In [347]: mean_ratings = mean_ratings.ix[active_titles]

In [348]: mean_ratings
Out[348]:
<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)
```

To see the top films among female viewers, we can sort by the `F` column in descending order:

```
In [350]: top_female_ratings = mean_ratings.sort_index(by='F', ascending=False)

In [351]: top_female_ratings[:10]
Out[351]:
gender                                                 F         M
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
```

## Measuring rating disagreement

Suppose you wanted to find the movies that are most divisive between male and female viewers. One way is to add a column to `mean_ratings` containing the difference in means, then sort by that:

```
In [352]: mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
```

Sorting by `'diff'` gives us the movies with the greatest rating difference and which were preferred by women:

```
In [353]: sorted_by_diff = mean_ratings.sort_index(by='diff')

In [354]: sorted_by_diff[:15]
Out[354]:
gender                                       F         M      diff
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
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
```

Reversing the order of the rows and again slicing off the top 15 rows, we get the movies preferred by men that women didn't rate as highly:

```
# Reverse order of rows, take first 15 rows
In [355]: sorted_by_diff[::-1][:15]
Out[355]:
gender                                     F         M      diff
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
```

Suppose instead you wanted the movies that elicited the most disagreement among viewers, independent of gender. Disagreement can be measured by the variance or standard deviation of the ratings:

```
# Standard deviation of rating grouped by title
In [356]: rating_std_by_title = data.groupby('title')['rating'].std()

# Filter down to active_titles
In [357]: rating_std_by_title = rating_std_by_title.ix[active_titles]

# Order Series by value in descending order
In [358]: rating_std_by_title.order(ascending=False)[:10]
Out[358]:
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
```

You may have noticed that movie genres are given as a pipe-separated (|) string. If you wanted to do some analysis by genre, more work would be required to transform the genre information into a more usable form. I will revisit this data later in the book to illustrate such a transformation.