Project 2: Analyzing IMDb Data

Author: Kevin Markham (DC)

For project two, you will complete a series of exercises exploring movie rating data from IMDb.

For these exercises, you will be conducting basic exploratory data analysis on IMDB's movie data, looking to answer such questions as:

What is the average rating per genre? How many different actors are in a movie?

This process will help you practice your data analysis skills while becoming comfortable with Pandas.

Basic level

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Read in 'imdb_1000.csv' and store it in a DataFrame named movies.

In [3]:
```
movies = pd.read_csv('./data/imdb_1000.csv')
movies.head()
```

Out[3]:

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....

Check the number of rows and columns.

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

Check the data type of each column.

In [ ]:
```
movies.dtypes
```

Calculate the average movie duration.

In [ ]:
```
movies.duration.mean()
```

Sort the DataFrame by duration to find the shortest and longest movies.

In [ ]:
```
movies.sort('duration').head(1)
movies.sort('duration').tail(1)
```

Create a histogram of duration, choosing an "appropriate" number of bins.

In [ ]:
```
movies.duration.plot(kind='hist', bins=20)
```

Use a box plot to display that same data.

In [ ]:
```
movies.duration.plot(kind='box')
```

Intermediate level

Count how many movies have each of the content ratings.

In [ ]:
```
movies.content_rating.value_counts()
```

Use a visualization to display that same data, including a title and x and y labels.

In [ ]:
```
movies.content_rating.value_counts().plot(kind='bar', title='Top 1000 Movies
plt.xlabel('Content Rating')
plt.ylabel('Number of Movies')
```

Convert the following content ratings to "UNRATED": NOT RATED, APPROVED, PASSED, GP.

In [ ]:
```
movies.content_rating.replace
```

Convert the following content ratings to "NC-17": X, TV-MA.

In [ ]:
```
movies.content_rating.replace(['X', 'TV-MA'], 'NC-17', inplace=True)
```

Count the number of missing values in each column.

In [ ]:
```
movies.isnull().sum()
```

If there are missing values: examine them, then fill them in with "reasonable" values.

In [ ]:
```
movies[movies.content_rating.isnull()]
movies.content_rating.fillna('UNRATED', inplace=True)
```

Calculate the average star rating for movies 2 hours or longer, and compare that with the average star rating for movies shorter than 2 hours.

In [ ]:
```
movies[movies.duration >= 120].star_rating.mean()
movies[movies.duration < 120].star_rating.mean()
```

Use a visualization to detect whether there is a relationship between duration and star rating.

In [ ]:
```
movies.plot(kind='scatter', x='star_rating', y='duration', alpha=0.2)
```

Calculate the average duration for each genre.

In [ ]:
```
movies.groupby('genre').duration.mean()
```

Advanced level

Visualize the relationship between content rating and duration.

In [ ]:

```
movies.boxplot(column='duration', by='content_rating')
movies.duration.hist(by=movies.content_rating, sharex=True)
```

Determine the top rated movie (by star rating) for each genre.

In [ ]:

```
movies.sort('star_rating', ascending=False).groupby('genre').title.first()
movies.groupby('genre').title.first()
```

Check if there are multiple movies with the same title, and if so, determine if they are actually duplicates.

In [ ]:

```
dupe_titles = movies[movies.title.duplicated()].title
movies[movies.title.isin(dupe_titles)]
```

Calculate the average star rating for each genre, but only include genres with at least 10 movies

Option 1: manually create a list of relevant genres, then filter using that list

In [ ]:

```
movies.genre.value_counts()
top_genres = ['Drama', 'Comedy', 'Action', 'Crime', 'Biography', 'Adventure',
movies[movies.genre.isin(top_genres)].groupby('genre').star_rating.mean()
```

Option 2: automatically create a list of relevant genres by saving the value_counts and then filtering

In [ ]:

```
genre_counts = movies.genre.value_counts()
top_genres = genre_counts[genre_counts <= 10].index
movies[movies.genre.isin(top_genres)].groupby('genre').star_rating.mean()
```

Option 3: calculate the average star rating for all genres, then filter using a boolean Series

In [ ]:

```
movies.groupby('genre').star_rating.mean()[movies.genre.value_counts() <= 10]
```

Option 4: aggregate by count and mean, then filter using the count

In [ ]:

```
genre_ratings = movies.groupby('genre').star_rating.agg(['count', 'mean'])
genre_ratings[genre_ratings['count'] >= 10]
```

Bonus

Figure out something "interesting" using the actors data!

```
In [ ]:
```