Final Project - Predicting Movie Genres!

Welcome to the final project of CS109b.

The overall theme of the final project is movie data with a focus on movie genre prediction, because it is an area where we are all more or less application domain experts. First, you will explore your data and the challenges of the problem by exploratory data analysis. Use visualizations to find features that correlate with movie genres. These can be extracted from the movie posters, or meta data, or other data you gather, for example plot summaries or even movie transcripts. You will then compare traditional statistical or machine learning methods like generalized additive models, random forest, Bayesian prediction methods, boosting, and SVM, to deep learning models for movie genre prediction.

For this project you will work in teams of 3-4 people and there are weekly milestones to guide you along the way. Even though the milestones are graded, they are mainly in place to make sure you stay in contact with your TF and make progress with the project. Throughout the project you also have room for creativity and to pursue your own ideas. While you need to hand in the milestones at the appropriate due date, there is nothing preventing you from working on a later milestone ahead of time. We suggest that you read through the whole project and all milestones in the beginning to be able to plan ahead. The project is pretty open-ended, so you can be creative and let your data science knowledge shine!

For each milestone you will submit a notebook, in raw (`.ipynb`) and PDF formats, containing the deliverables of that week and the extra work you did so far. The notebooks need to contain your code, comments, explanations, thoughts, and visualizations. The final deliverables are a two-minute screencast, a report in paper style for a general data science audience, and all your data and code that you developed throughout the project.

Below is a description of the data and the milestones with their due dates. All work is due by 11:59PM on the due date unless otherwise specified. We expect you to have the mandatory parts finished by the milestone due dates, and there will be no extensions. However, we strongly encourage you to plan ahead. For example, you need to think about the classification task early on to plan how you want to assemble your training data, and it is beneficial to start the deep learning work as early as possible. There is nothing hindering you to already train a model in the EDA phase to get a better feel for what challenges might lie ahead with the data. You should also see the milestone requirements as a basis for your own creativity, and we expect that most of you will go beyond the mandatory deliverables. For example, if you have a great idea about an interesting question that has to do with movie genre, but cannot be answered with the data from TMDb or IMDb, feel free to gather more data from somewhere else.

We provide a data interface in Python, because it is convenient for IMDb, and we will use Python for the deep learning part. Specifically we will use Keras, a deep learning library that provides a high level interface to Google's Tensorflow framework for deep learning. However, if you feel that you prefer to do some of the work, e.g., visualizations or data cleanup, in R then feel free to use it. You can also use Spark to preprocess your data, especially if you collect large amounts of it from other sources.

Important: Your grade for a milestone will depend on the required deliverables you submit at the due date for that milestone. But every milestone, especially the final project submission, can contain additional cool work you did that goes beyond the deliverables spelled out below.

Logistics

Please adhere to the following guidelines for all submissions:

- one submission per team
- notebooks should be submitted as PDF and as raw (`.ipynb`) version
- all notebooks should be executed so they contain relevant visualizations, and other results
- try to make it as easy as possible for the TFs to get all relevant information about your work
- do not submit big data sets, please provide a readme file with a link instead
- the final report should also be submitted as pdf

Movie Data:

The project is based on two different sources of movie data: IMDb (http://www.imdb.com/) and TMDb (https://www.themoviedb.org/). TMDb is great, because it provides the movie posters in addition to the metadata. This is crucial for the deep learning part, in which you will try to predict movie genres from posters. IMDb has more metadata available and will supplement the TMDb data you have.

TMDb provides an easy to use API (https://www.themoviedb.org/documentation/api) that allows you to download the data selectively. IMDb does not provide an API, but there is a Python interface available to access the metadata. We will use IMDbPY (http://imdbpy.sourceforge.net/), which is already installed on the AMI and virtual box images for your convenience.

Important: Please remember to limit your data rate when obtaining the data. Play nicely and do not just spam servers as fast as you can. This will prevent your IP from getting banned. The easiest way to do this is to use the sleep (http://stackoverflow.com/questions/510348/how-can-i-make-a-time-delay-in-python) function in Python.

Milestone 1: Getting to know your data, due Wednesday, April 5, 2017

In the beginning you should get acquainted with the data sources and do some EDA. Sign up for the TMDb API (https://www.themoviedb.org/documentation/api), and try to download the poster of your favorite movie from within your notebook. Compare the genre entries of IMDb and TMDb for this movie and see if they are the same. Think about and write down some questions that you would like to answer in the following weeks. Keep the storytelling aspect of your final report in mind and do some pen and paper sketches about the visualizations you would like to produce. Include photographs of those sketches in your notebook.

Most of the time a data scientist spends on a project is spent on cleaning the data. We are lucky that the data we have is already pretty clean. The Python interface to the IMDb ftp files does a lot of the additional work of cleaning as well. However, you will notice that the genre list for each movie from both databases can have different lengths. This needs to be changed in order to train a model to predict the movie genre. It is up to you to think about possible ways to address this problem and to implement one of them. There is no absolute right answer here. It depends on your interests and which questions you have in mind for the project.

Optionally, you could also scrape additional data sources, such as Wikipedia, to obtain plot summaries. That data may give you additional useful features for genre classification.

To guide your decision process, provide at least one visualization of how often genres are mentioned together in pairs. Your visualization should clearly show if a horror romance is more likely to occur in the data than a drama romance.

The notebook to submit for this milestone needs to at least include:

- API code to access the genre and movie poster path of your favorite movie
- Genre for this movie listed by TMDb and IMDb
- A list of the 10 most popular movies of 2016 from TMDb and their genre obtained via the API
- Comments on what challenges you see for predicting movie genre based on the data you have, and how to address them
- Code to generate the movie genre pairs and a suitable visualization of the result
- Additional visualization sketches and EDA with a focus on movie genres
- A list of questions you could answer with this and related data. Get creative here!

The EDA questions do not necessarily have to tie into the modeling part later on. Think freely about things that might be interesting, like which actors are very specific to a genre? Are action movies more prone to producing sequels than romances? However, as you keep the focus on movie genres, think also about correlations you might discover that can help build features from the metadata for prediction. Is the length of a movie title correlated with genre?

In [53]:
```
import time
import seaborn as sns
import numpy as np
import pandas as pd
from IPython.display import Image
import matplotlib.pyplot as plt
from collections import Counter
%matplotlib inline

## Installed by running this line in terminal: pip install IMDbPY
## Tutorial found here http://imdbpy.sourceforge.net/support.html
import imdb

### Downloaded this via this line: pip install tmdbsimple
## Tutorial found here https://pypi.python.org/pypi/tmdbsimple
import tmdbsimple as tmdb
```

In [30]:
```
## Pass in our tmdb Key
tmdb.API_KEY = '352e668a0df90032e0f1097459228131'

# Create the object that will be used to access the IMDb's database.
ia = imdb.IMDb()
```

#1: API code to access the genre and movie poster path of your favorite movie

- Favorite movie: 300
- Database: TMDB

In [31]:
```
# first need to get the tmdb id for 300
search = tmdb.Search()
response = search.movie(query="300")

for res in search.results:
    print res['title'], res['id'], res['release_date'],
res['popularity']

## in this case I just took the top result
fav_movie_tmdb = search.results[0]
```

```
300 1271 2006-12-09 3.745886
300: Rise of an Empire 53182 2014-03-05 3.863153
Decameron '300 436902 1972-08-11 1.01302
300 Souls 332423 2015-03-25 1.006227
United 300 204183 2007-04-30 1.000996
300 Miles 409110 2016-08-11 1.028247
300 Killers 54799 2011-01-18 1.002222
300 Pounds 68690 2007-01-01 1.000451
Last Stand of the 300 37122 2007-05-27 1.249439
The 300 Spartans 19972 1962-08-01 1.379025
300 Worte Deutsch 309038 2015-02-05 1.144064
300 Miles to Heaven 155325 1989-10-30 1.095081
300 Miles to Freedom 329318 2011-01-01 1.002061
4 Damas en 300 293965 2011-07-17 1.00101
Kamis Ke 300 406842  1.000911
Planet Deutschland - 300 Millionen Jahre 295831 2014-10-02 1.00244
Million Dollar Eel 285994 1971-05-06 1
Fangio: Una vita a 300 all'ora 435791 1980-01-01 1.000024
75 habitantes, 20 casas, 300 vacas 202434 2011-05-11 1.004443
Roberto Carlos a 300 Quilômetros Por Hora 273427 1971-06-01 1.001025
```

In [32]:
```
fav_movie_id = search.results[0]["id"]
print "Movie ID for '300': ", fav_movie_id
```

```
Movie ID for '300':  1271
```

In [33]:
```
# get the movie
movie = tmdb.Movies(1271)
info = movie.info()
print "TMDB Genres for 300: ", [genre["name"] for genre in
info["genres"]]
print "TMDB Poster path for 300: ", info["poster_path"]
```

```
TMDB Genres for 300:  [u'Action', u'Adventure', u'War']
TMDB Poster path for 300:  /bYR8O1H1ZlME7Dm9ysfTYZnRDpw.jpg
```

In [34]:
```
# Search for a movie (get a list of Movie objects).
s_result = ia.search_movie('300')
for item in s_result:
    print item['long imdb canonical title'], item.movieID
```

```
300 (2006) 0416449
"300" (2014) 4066210
"300 (2014) (TV Episode)  - Season 3 | Episode 5  - Honest Trailers" (2
012) 3614658
300: Rise of an Empire (2014) 1253863
Idiocracy (2006) 0387808
3000 Miles to Graceland (2001) 0233142
Mystery Science Theater 3000: The Movie (1996) 0117128
300 the Resurgence (in development) (????) 4576032
300 Spartans, The (1962) 0055719
"Mystery Science Theater 3000" (1988) 0094517
Last Stand of the 300 (2007) (TV) 0892737
Decameron '300 (1972) 0068460
"300 (2005) (TV Episode)  - Season 21 | Episode 27  - The Bill" (1984)
 0524939
Gruz 300 (1990) 0309643
300 Worte Deutsch (2013) 2288044
"300 (2014) (TV Episode)  - Season 1 | Episode 9  - Le Plectroscope" (2
011) (mini) 5785046
"300 (2007) (TV Episode)  - Season 4 | Episode 2  - STC Previews, The"
 (2003) 1084548
"300 (2012) (TV Episode)  - Season 3 | Episode 9  - Bath Bayakha" (201
1) (mini) 5152204
"300 (2009) (TV Episode)  - Season 1 | Episode 73  - Merci Qui?" (2008)
5274864
"300 (2007) (TV Episode)  - Season 1 | Episode 5  - History in Focus"
 (2006) 1142331
```

In [35]:
```
fav_movie_imdb = s_result[0]
ia.update(fav_movie_imdb)
print "IMDB Genres for 300: ",  fav_movie_imdb['genre']
```

```
IMDB Genres for 300:  [u'Action', u'Fantasy']
```

#2: Genre for this movie listed by TMDb and IMDb

In summary:

```
* TMDB says the genres are Action, Adventure, and War
* IMDB says the genres are Action, and Fantasy
```

This difference could lead to problems in the future when we are trying to compare similar movies but that come have different supposed "genres." One way that we might avoid this is by doing a merge between the genres or selecting genres that appear in both (in this case that would be 'action')

#3: A list of the 10 most popular movies of 2016 from TMDb and their genre obtained via the API

In [36]:
```
# abdapted from http://programtalk.com/python-examples/tmdbsimple.Discov
er/
discover = tmdb.Discover()
movies_2016 = discover.movie(year=2016)
```

In [37]:
```
for movie in discover.results[0:10]:
    _id = movie["id"]
    movie = tmdb.Movies(_id)
    response = movie.info()
    print "Movie title: {0}".format(response["title"])
    print "Movie popularity: {0}".format(response["popularity"])
    print "Movie genre: {0}".format([info["name"] for info in movie.genr
es])
    print "\n"
```

```
Movie title: Sing
Movie popularity: 75.005907
Movie genre: [u'Animation', u'Comedy', u'Drama', u'Family', u'Music']


Movie title: Fantastic Beasts and Where to Find Them
Movie popularity: 38.239379
Movie genre: [u'Adventure', u'Action', u'Fantasy']


Movie title: Split
Movie popularity: 33.606082
Movie genre: [u'Horror', u'Thriller']


Movie title: Finding Dory
Movie popularity: 30.769363
Movie genre: [u'Adventure', u'Animation', u'Comedy', u'Family']


Movie title: Deadpool
Movie popularity: 25.382598
Movie genre: [u'Action', u'Adventure', u'Comedy', u'Romance']


Movie title: Rogue One: A Star Wars Story
Movie popularity: 24.082126
Movie genre: [u'Action', u'Drama', u'Science Fiction', u'War']


Movie title: Doctor Strange
Movie popularity: 22.441433
Movie genre: [u'Action', u'Adventure', u'Fantasy', u'Science Fiction']


Movie title: Arrival
Movie popularity: 21.476241
Movie genre: [u'Drama', u'Science Fiction']


Movie title: Captain America: Civil War
Movie popularity: 20.467025
Movie genre: [u'Action', u'Science Fiction']


Movie title: John Wick
Movie popularity: 19.206953
Movie genre: [u'Action', u'Thriller']
```

#4: Comments on what challenges you see for predicting movie genre based on the data you have, and how to address them

1) There are differences in the genres listed by TMDb vs. IMDb

```
- We could use a union of the genres so that only those genres in both datab
ases are considered correct
This also leads to a bigger question of "What does it mean to correctly sele
ct a movie genre?" Does thie mean that we select the best genre from IMDb, T
MDb? Does this mean we might select multiple genres? These are all questions
 that we will need to face.
```

2) The data will take a long time to load from the APIs because we need to restrict the rate at which we pull large amounts of data - ethics

```
- We need to work ahead and use AWS. In particular, if we are using a large
 dataset, we might need to pull a significant amount of our data in advance.
```

3) We need to figure out what the "Correct" Genre means

```
- As we can see from the print out above, many movies have more than one gen
re. In this case we have that doctor strangelove is considered action, adven
ture, fantasy, AND science fiction. However, we probably want to create a mo
del that only predicts one of these genres and not all of them. One way that
 we can break this down and make this easier is by creating a subset of genr
es that we believe exist in the world. (e.g. taking the top 50 genres.) Then
 for each of our movies, we make a prediction that only exists within that r
ealm of genres. Another option is for us to try and predict all genres that
 might be attached to a movie, though this data might be too sparse to be pr
edictive.
```

#5: Code to generate the movie genre pairs and a suitable visualization of the result

This section gives a brief look at the relationship between genres. We will then extend upon it to choose only one genre per a movie.

In [38]:
```
genres_obj = tmdb.Genres()
genre_ids = {genre["name"]: genre["id"] for genre in genres_obj.list()
["genres"]}

print "Number of genres: {0}".format(len(genre_ids.keys()))
genre_ids
```

```
Number of genres: 19
```

Out[38]:
```
{u'Action': 28,
 u'Adventure': 12,
 u'Animation': 16,
 u'Comedy': 35,
 u'Crime': 80,
 u'Documentary': 99,
 u'Drama': 18,
 u'Family': 10751,
 u'Fantasy': 14,
 u'History': 36,
 u'Horror': 27,
 u'Music': 10402,
 u'Mystery': 9648,
 u'Romance': 10749,
 u'Science Fiction': 878,
 u'TV Movie': 10770,
 u'Thriller': 53,
 u'War': 10752,
 u'Western': 37}
```

In [39]:
```
reverse_genre_ids = {v: k for k, v in genre_ids.iteritems()}
reverse_genre_ids
```

Out[39]:
```
{12: u'Adventure',
 14: u'Fantasy',
 16: u'Animation',
 18: u'Drama',
 27: u'Horror',
 28: u'Action',
 35: u'Comedy',
 36: u'History',
 37: u'Western',
 53: u'Thriller',
 80: u'Crime',
 99: u'Documentary',
 878: u'Science Fiction',
 9648: u'Mystery',
 10402: u'Music',
 10749: u'Romance',
 10751: u'Family',
 10752: u'War',
 10770: u'TV Movie'}
```

In [40]:
```
discover = tmdb.Discover()
movie_dict = {genre: [] for genre in genre_ids.iterkeys()}

# count how many results we get for each genre
movie_cnts = {genre: 0 for genre in genre_ids.iterkeys()}

# need ids for each genre
for genre in genre_ids.keys():
    # scan 20 pages for movies
    for p in range(1, 20):
        # find all movies with a given id on a given page
        discover.movie(with_genres = genre_ids[genre], page = p)
        for page in discover.results:
            # we found another movie
            movie_cnts[genre] += 1

            # add the ids from this new movie to the list
            movie_dict[genre].extend(page["genre_ids"])

    # sleep so that we don't get kicked off the API
    time.sleep(10)
```

In [41]:
```
# make a copy of the movie dict so that we don't have
# to rerun the above cell if we mess up
genre_percentages = movie_dict.copy()

# for each genre in the dictionary
for genre in genre_percentages:

    # make a dictionary of frequencies instead of raw ids
    genre_percentages[genre] = Counter(genre_percentages[genre])

    # normalize each raw number into a percentage
    for genre_id in genre_percentages[genre]:
        genre_percentages[genre][genre_id] = genre_percentages[genre][ge
nre_id] * 1.0 / movie_cnts[genre]
```

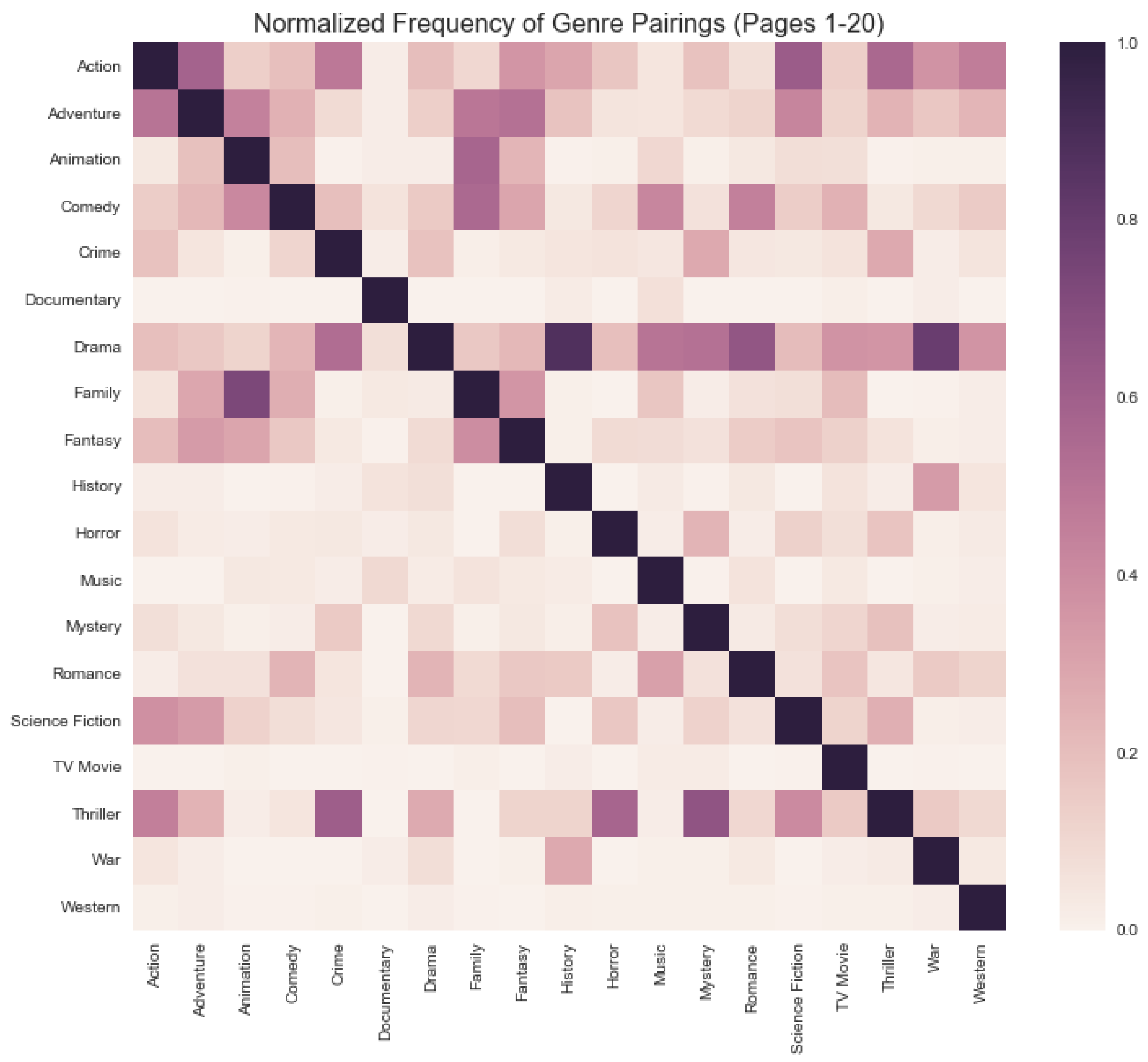
In [42]:
```
genre_percentages
```

```
Out[42]: {u'Action': Counter({12: 0.5026315789473684,
                    14: 0.20526315789473684,
                    16: 0.042105263157894736,
                    18: 0.19736842105263158,
                    27: 0.05789473684210526,
                    28: 1.0,
                    35: 0.14210526315789473,
                    36: 0.021052631578947368,
                    37: 0.010526315789473684,
                    53: 0.45,
                    80: 0.18421052631578946,
                    878: 0.37894736842105264,
                    9648: 0.07105263157894737,
                    10749: 0.015789473684210527,
                    10751: 0.05789473684210526,
                    10752: 0.04736842105263158}),
          u'Adventure': Counter({12: 1.0,
                    14: 0.3368421052631579,
                    16: 0.18947368421052632,
                    18: 0.1631578947368421,
                    27: 0.02368421052631579,
                    28: 0.5842105263157895,
                    35: 0.22631578947368422,
                    36: 0.021052631578947368,
                    37: 0.018421052631578946,
                    53: 0.24473684210526317,
                    80: 0.05,
                    878: 0.3368421052631579,
                    9648: 0.039473684210526314,
                    10402: 0.002631578947368421,
                    10749: 0.06578947368421052,
                    10751: 0.29210526315789476,
                    10752: 0.021052631578947368}),
          u'Animation': Counter({12: 0.44473684210526315,
                    14: 0.29473684210526313,
                    16: 1.0,
                    18: 0.11842105263157894,
                    27: 0.015789473684210527,
                    28: 0.1394736842105263,
                    35: 0.4105263157894737,
                    36: 0.002631578947368421,
                    37: 0.005263157894736842,
                    53: 0.021052631578947368,
                    80: 0.007894736842105263,
                    99: 0.005263157894736842,
                    878: 0.12368421052631579,
                    9648: 0.007894736842105263,
                    10402: 0.039473684210526314,
                    10749: 0.06578947368421052,
                    10751: 0.7289473684210527,
                    10752: 0.002631578947368421,
                    10770: 0.010526315789473684}),
          u'Comedy': Counter({12: 0.24736842105263157,
                    14: 0.16052631578947368,
                    16: 0.2,
                    18: 0.23421052631578948,
                    27: 0.031578947368421054,
                    28: 0.19473684210526315,
                    35: 1.0,
                    36: 0.005263157894736842,
                    37: 0.005263157894736842,
                    53: 0.05,
                    80: 0.11052631578947368,
                    878: 0.07894736842105263,
                    9648: 0.015789473684210527,
                    10402: 0.031578947368421054,
                    10749: 0.24210526315789474,
                    10751: 0.2631578947368421,
                    10752: 0.002631578947368421}),
           u'Crime': Counter({12: 0.0868421052631579,
                    14: 0.031578947368421054,
                    16: 0.005263157894736842,
                    18: 0.5368421052631579,
                    27: 0.03684210526315789,
                    28: 0.48157894736842105,
                    35: 0.19210526315789472,
                    36: 0.018421052631578946,
                    37: 0.010526315789473684,
                    53: 0.6078947368421053,
                    80: 1.0,
                    878: 0.04473684210526316,
                    9648: 0.15263157894736842,
                    10402: 0.015789473684210527,
                    10749: 0.04736842105263158,
                    10751: 0.007894736842105263}),
           u'Documentary': Counter({12: 0.015789473684210527,
                    14: 0.005263157894736842,
                    16: 0.015789473684210527,
                    18: 0.07105263157894737,
                    27: 0.015789473684210527,
                    28: 0.018421052631578946,
                    35: 0.05526315789473684,
                    36: 0.05526315789473684,
                    53: 0.002631578947368421,
                    80: 0.018421052631578946,
                    99: 1.0,
                    878: 0.010526315789473684,
                    9648: 0.002631578947368421,
                    10402: 0.1,
                    10751: 0.034210526315789476,
                    10752: 0.018421052631578946,
                    10769: 0.007894736842105263,
                    10770: 0.005263157894736842}),
           u'Drama': Counter({12: 0.13421052631578947,
                    14: 0.0868421052631579,
                    16: 0.015789473684210527,
                    18: 1.0,
                    27: 0.03684210526315789,
                    28: 0.2026315789473684,
                    35: 0.15526315789473685,
                    36: 0.07368421052631578,
                    37: 0.018421052631578946,
                    53: 0.2736842105263158,
                    80: 0.1868421052631579,
                    878: 0.10789473684210527,
                    9648: 0.09473684210526316,
                    10402: 0.02631578947368421,
                    10749: 0.24210526315789474,
                    10751: 0.02894736842105263,
                    10752: 0.07631578947368421}),
           u'Family': Counter({12: 0.49473684210526314,
                    14: 0.39210526315789473,
                    16: 0.5736842105263158,
                    18: 0.16052631578947368,
                    28: 0.10263157894736842,
                    35: 0.5552631578947368,
                    36: 0.002631578947368421,
                    37: 0.005263157894736842,
                    53: 0.002631578947368421,
                    80: 0.013157894736842105,
                    99: 0.002631578947368421,
                    878: 0.10263157894736842,
                    9648: 0.010526315789473684,
                    10402: 0.060526315789473685,
                    10749: 0.09210526315789473,
                    10751: 1.0,
                    10770: 0.013157894736842105}),
           u'Fantasy': Counter({12: 0.5184210526315789,
                    14: 1.0,
                    16: 0.23421052631578948,
                    18: 0.2236842105263158,
                    27: 0.07631578947368421,
                    28: 0.3605263157894737,
                    35: 0.29473684210526313,
                    36: 0.002631578947368421,
                    37: 0.002631578947368421,
                    53: 0.11842105263157894,
                    80: 0.031578947368421054,
                    99: 0.002631578947368421,
                    878: 0.2026315789473684,
                    9648: 0.03684210526315789,
                    10402: 0.034210526315789476,
                    10749: 0.16052631578947368,
                    10751: 0.3605263157894737,
                    10752: 0.007894736842105263,
                    10770: 0.002631578947368421}),
           u'History': Counter({12: 0.18157894736842106,
                    14: 0.007894736842105263,
                    16: 0.005263157894736842,
                    18: 0.8763157894736842,
                    27: 0.007894736842105263,
                    28: 0.29473684210526313,
                    35: 0.042105263157894736,
                    36: 1.0,
                    37: 0.013157894736842105,
                    53: 0.11842105263157894,
                    80: 0.04736842105263158,
                    99: 0.02368421052631579,
                    9648: 0.013157894736842105,
                    10402: 0.02368421052631579,
                    10749: 0.15263157894736842,
                    10751: 0.007894736842105263,
                    10752: 0.28157894736842104,
                    10769: 0.005263157894736842,
                    10770: 0.015789473684210527}),
           u'Horror': Counter({12: 0.05263157894736842,
                    14: 0.08947368421052632,
                    16: 0.007894736842105263,
                    18: 0.19736842105263158,
                    27: 1.0,
                    28: 0.16842105263157894,
                    35: 0.11315789473684211,
                    37: 0.007894736842105263,
                    53: 0.5710526315789474,
                    80: 0.05789473684210526,
                    878: 0.16578947368421051,
                    9648: 0.1868421052631579,
                    10402: 0.002631578947368421,
                    10749: 0.018421052631578946,
                    10752: 0.002631578947368421}),
           u'Music': Counter({12: 0.05,
                    14: 0.08421052631578947,
                    16: 0.10526315789473684,
                    18: 0.5026315789473684,
                    27: 0.021052631578947368,
                    28: 0.05,
                    35: 0.4236842105263158,
                    36: 0.02894736842105263,
                    37: 0.010526315789473684,
                    53: 0.021052631578947368,
                    80: 0.04473684210526316,
                    99: 0.06842105263157895,
                    878: 0.021052631578947368,
                    9648: 0.015789473684210527,
                    10402: 1.0,
                    10749: 0.3131578947368421,
                    10751: 0.16842105263157894,
                    10752: 0.007894736842105263,
                    10769: 0.002631578947368421,
                    10770: 0.02368421052631579}),
           u'Mystery': Counter({12: 0.09210526315789473,
                    14: 0.06315789473684211,
                    16: 0.007894736842105263,
                    18: 0.5157894736842106,
                    27: 0.2394736842105263,
                    28: 0.18421052631578946,
                    35: 0.06315789473684211,
                    36: 0.005263157894736842,
                    37: 0.010526315789473684,
                    53: 0.6631578947368421,
                    80: 0.28421052631578947,
                    878: 0.12368421052631579,
                    9648: 1.0,
                    10402: 0.005263157894736842,
                    10749: 0.06578947368421052,
                    10751: 0.018421052631578946,
                    10752: 0.007894736842105263,
                    10770: 0.02368421052631579}),
           u'Romance': Counter({12: 0.11578947368421053,
                    14: 0.14736842105263157,
                    16: 0.03684210526315789,
                    18: 0.6526315789473685,
                    27: 0.015789473684210527,
                    28: 0.07105263157894737,
                    35: 0.4473684210526316,
                    36: 0.039473684210526314,
                    37: 0.010526315789473684,
                    53: 0.10263157894736842,
                    80: 0.04473684210526316,
                    878: 0.06315789473684211,
                    9648: 0.02894736842105263,
                    10402: 0.05789473684210526,
                    10749: 1.0,
                    10751: 0.06315789473684211,
                    10752: 0.031578947368421054,
                    10770: 0.002631578947368421}),
           u'Science Fiction': Counter({12: 0.42894736842105263,
                    14: 0.1736842105263158,
                    16: 0.07894736842105263,
                    18: 0.21052631578947367,
                    27: 0.12631578947368421,
                    28: 0.618421052631579,
                    35: 0.14210526315789473,
                    37: 0.005263157894736842,
                    53: 0.4052631578947368,
                    80: 0.042105263157894736,
                    878: 1.0,
                    9648: 0.07894736842105263,
                    10749: 0.06315789473684211,
                    10751: 0.07368421052631578,
                    10752: 0.002631578947368421,
                    10770: 0.005263157894736842}),
           u'TV Movie': Counter({12: 0.12105263157894737,
                    14: 0.12631578947368421,
                    16: 0.07368421052631578,
                    18: 0.37105263157894736,
                    27: 0.07368421052631578,
                    28: 0.1394736842105263,
                    35: 0.24736842105263157,
                    36: 0.060526315789473685,
                    37: 0.007894736842105263,
                    53: 0.15263157894736842,
                    80: 0.05789473684210526,
                    99: 0.013157894736842105,
                    878: 0.12105263157894737,
                    9648: 0.11052631578947368,
                    10402: 0.031578947368421054,
                    10749: 0.1736842105263158,
                    10751: 0.21052631578947367,
                    10752: 0.015789473684210527,
                    10770: 1.0}),
           u'Thriller': Counter({12: 0.24473684210526317,
                    14: 0.060526315789473685,
                    16: 0.005263157894736842,
                    18: 0.3605263157894737,
                    27: 0.17894736842105263,
                    28: 0.5605263157894737,
                    35: 0.042105263157894736,
                    36: 0.021052631578947368,
                    37: 0.007894736842105263,
                    53: 1.0,
                    80: 0.28157894736842104,
                    878: 0.25526315789473686,
                    9648: 0.18947368421052632,
                    10749: 0.04473684210526316,
                    10752: 0.02894736842105263}),
           u'War': Counter({12: 0.16578947368421051,
                    14: 0.013157894736842105,
                    16: 0.010526315789473684,
                    18: 0.7947368421052632,
                    27: 0.013157894736842105,
                    28: 0.37105263157894736,
                    35: 0.09473684210526316,
                    36: 0.33421052631578946,
                    37: 0.015789473684210527,
                    53: 0.15526315789473685,
                    80: 0.021052631578947368,
                    99: 0.015789473684210527,
                    878: 0.013157894736842105,
                    9648: 0.018421052631578946,
                    10402: 0.007894736842105263,
                    10749: 0.15263157894736842,
                    10751: 0.005263157894736842,
                    10752: 1.0,
                    10769: 0.002631578947368421,
                    10770: 0.005263157894736842}),
           u'Western': Counter({12: 0.23684210526315788,
                    14: 0.018421052631578946,
                    16: 0.010526315789473684,
                    18: 0.36578947368421055,
                    27: 0.02894736842105263,
                    28: 0.4631578947368421,
                    35: 0.15,
                    36: 0.04736842105263158,
                    37: 1.0,
                    53: 0.09473684210526316,
                    80: 0.05263157894736842,
                    878: 0.018421052631578946,
                    9648: 0.02368421052631579,
                    10402: 0.018421052631578946,
                    10749: 0.11578947368421053,
                    10751: 0.015789473684210527,
                    10752: 0.034210526315789476,
                    10769: 0.007894736842105263,
                    10770: 0.002631578947368421})}
```

In [43]:
```
percent_df = pd.DataFrame([pd.Series(genre_percentages[genre]) for genre
 in genre_ids.keys()])
percent_df = percent_df.T
percent_df.rename(columns=dict(zip(range(0, 19), genre_ids.keys())), inp
lace = True)

# drop this rogue ID
percent_df.drop([10769], inplace = True)

percent_df.rename(index=dict(zip(percent_df.index, [reverse_genre_ids[ge
nre_id] for genre_id in percent_df.index])), inplace = True)

# replace all nans with 0, indicating a 0 percent mention rate for that
 pair
percent_df.fillna(0, inplace = True)

# reorder the axes so that they match up on the diagonals
percent_df = percent_df.reindex_axis(sorted(percent_df.columns), axis=1)
percent_df = percent_df.reindex_axis(sorted(percent_df.columns), axis=0)
```

```
In [44]: plt.figure(figsize=(12, 10))
         plt.title("Normalized Frequency of Genre Pairings (Pages 1-20)", fontsiz
         e=16)
         sns.heatmap(percent_df)
```

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x10bc04e50>
```



Insights

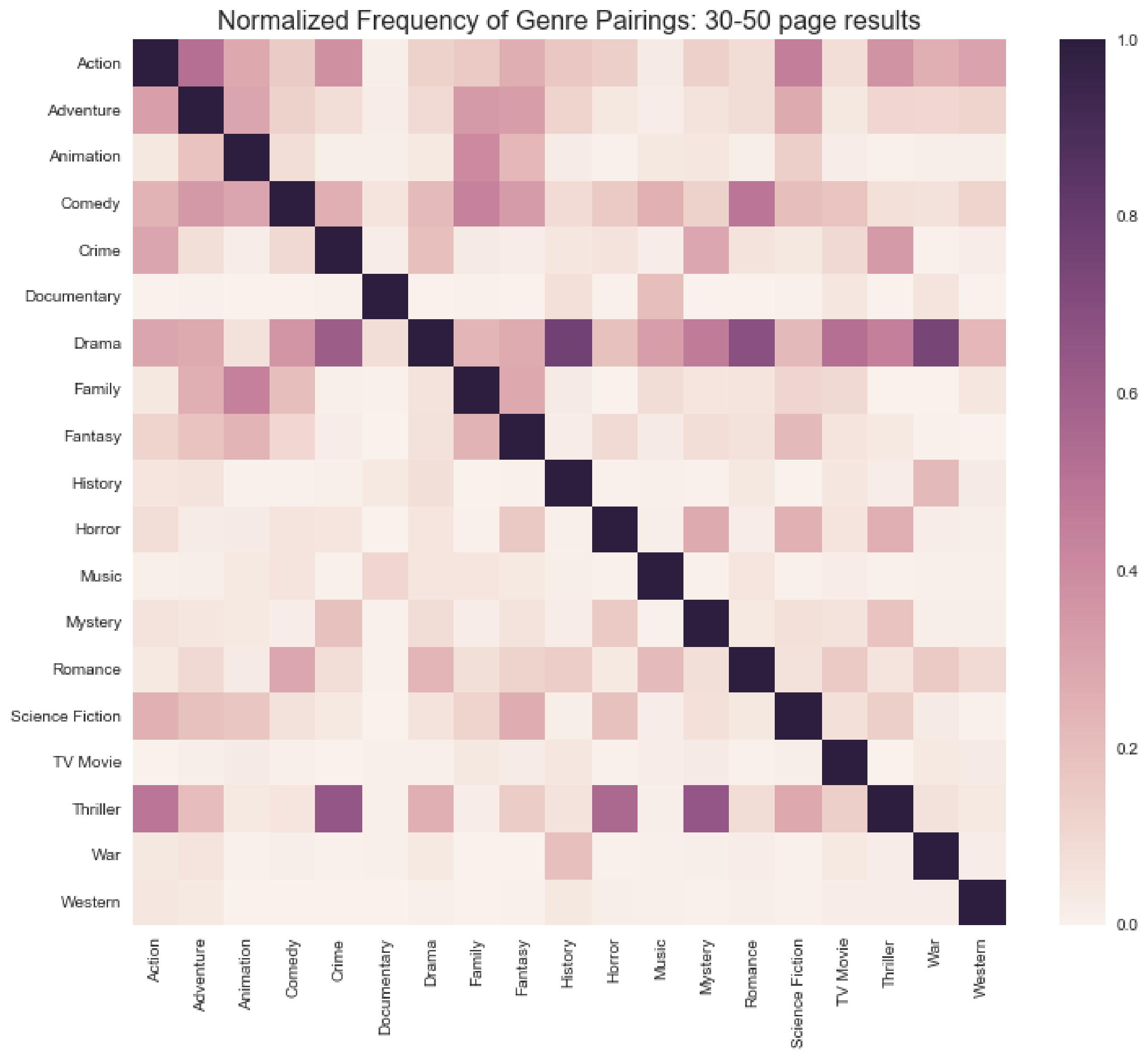
- Examining our heatmap for the first twenty page instances per genre, we see that the pairing are intuitive.
- Here, we see overlap between history and drama films along with adventure and action. In addition, we see high crossover between family and animation along with crime and thriller.
- Beyond seeing the tendancy for some genres to be listed together, we also see that some categories are vaguer than others. Here, for example, we see that Thriller and Drama serve as catch-alls whereas documentary has a relatively specific definition.

#6: Additional visualization sketches and EDA with a focus on movie genres

In [45]:
```
discover_ = tmdb.Discover()
movie_dict = {genre: [] for genre in genre_ids.iterkeys()}

# count how many results we get for each genre
movie_cnts = {genre: 0 for genre in genre_ids.iterkeys()}

# need ids for each genre
for genre in genre_ids.keys():
    # scan 20 pages for movies
    for p in range(30, 50):
        # find all movies with a given id on a given page
        discover.movie(with_genres = genre_ids[genre], page = p)
        for page in discover.results:
            # we found another movie
            movie_cnts[genre] += 1

            # add the ids from this new movie to the list
            movie_dict[genre].extend(page["genre_ids"])

    # sleep so that we don't get kicked off the API
    time.sleep(10)

# make a copy of the movie dict so that we don't have
# to rerun the above cell if we mess up
genre_percentages = movie_dict.copy()

# for each genre in the dictionary
for genre in genre_percentages:

    # make a dictionary of frequencies instead of raw ids
    genre_percentages[genre] = Counter(genre_percentages[genre])

    # normalize each raw number into a percentage
    for genre_id in genre_percentages[genre]:
        genre_percentages[genre][genre_id] = genre_percentages[genre][ge
nre_id] * 1.0 / movie_cnts[genre]

percent_df = pd.DataFrame([pd.Series(genre_percentages[genre]) for genre
 in genre_ids.keys()])
percent_df = percent_df.T
percent_df.rename(columns=dict(zip(range(0, 19), genre_ids.keys())), inp
lace = True)

# drop this rogue ID
percent_df.drop([10769], inplace = True)

percent_df.rename(index=dict(zip(percent_df.index, [reverse_genre_ids[ge
nre_id] for genre_id in percent_df.index])), inplace = True)

# replace all nans with 0, indicating a 0 percent mention rate for that
```

```
 pair
percent_df.fillna(0, inplace = True)

# reorder the axes so that they match up on the diagonals
percent_df = percent_df.reindex_axis(sorted(percent_df.columns), axis=1)
percent_df = percent_df.reindex_axis(sorted(percent_df.columns), axis=0)

plt.figure(figsize=(12, 10))
plt.title("Normalized Frequency of Genre Pairings: 30-50 page results",
fontsize=16)
sns.heatmap(percent_df)
```

```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x10c03ba50>
```



Checking Consistancy with Pages 30-50

Comparing the results with the initial 1-20 pages, we see that our findings vary but remain consistant within a reasonable degree. This is to be expect as we have utilized large numbers to get a sample of a greater amount of data.

In [46]:
```
### There are 19 general genres, so 19 choose 2 (171) pairs)
### I Found these ideas by searching for movies that I knew that were ro
mance/drama/horror, finding and then finding
### That genre id
DRAMA_ID = 18
HORROR_ID = 27
ROMANCE_ID = 10749

movie = tmdb.Movies(565)
response = movie.info()
movie.genres

## initialize discover
discover = tmdb.Discover()

## create a list of romance movies
romance_list = []

## iterate through 20 pages collecting movies
for p in range(1,20):
    discover.movie(with_genres = ROMANCE_ID, page = p)
    romance_list.append(discover.results)
```

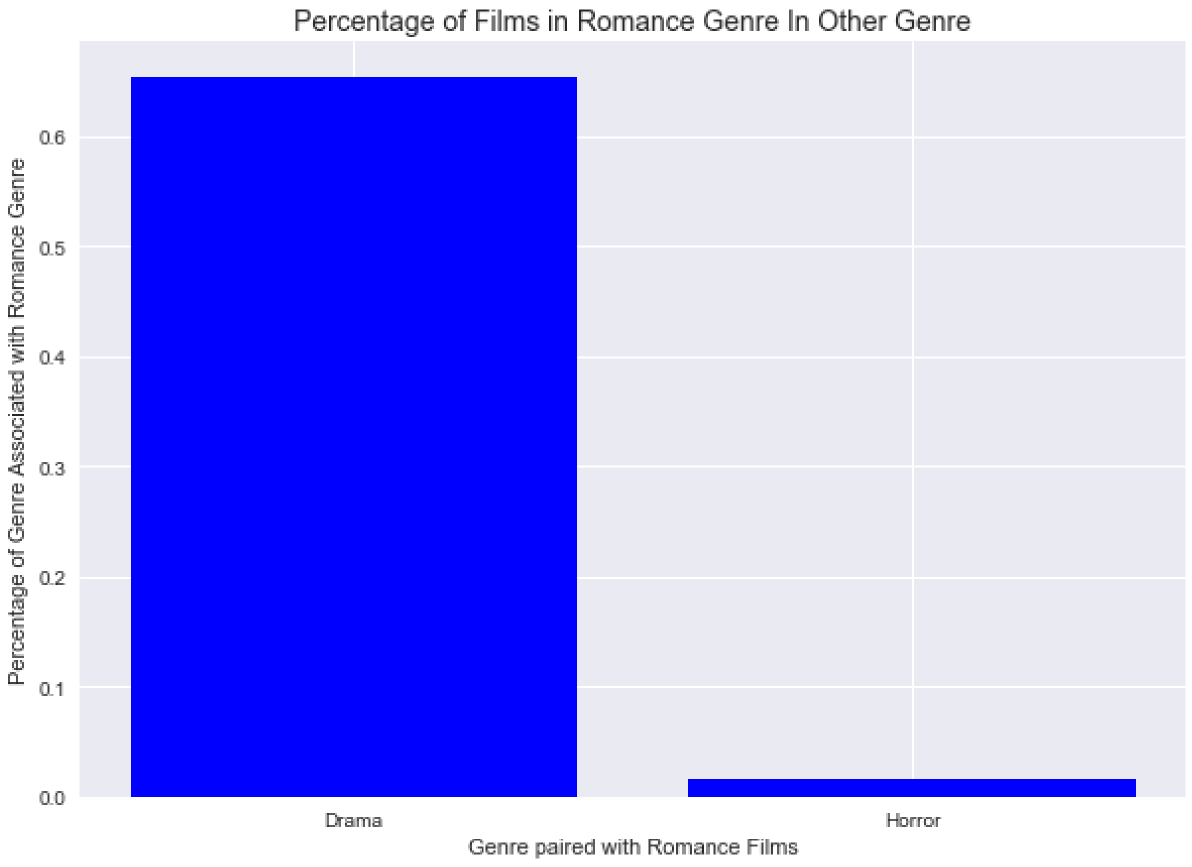
In [47]:
```
## iterate through the pages and then the movies
### this will give us back a list of lists of pairings
genre_pairings = []
for page in romance_list:
    for movie in page:
        genre_pairings.append(movie["genre_ids"])
```

In [48]:
```
### This will give
horror_count = 0
drama_count = 0
## iterate through the genre pairings for each movie
## add a count if horror appears or drama appears

for pairing in genre_pairings:
    if HORROR_ID in pairing:
        horror_count += 1
    if DRAMA_ID in pairing:
        drama_count += 1
```

In [49]:
```
num_movies = len(genre_pairings)
print "We test on this many movies classified as Romance:", num_movies

drama_romance_perc = drama_count / float(num_movies)
horror_romance_perc= horror_count / float(num_movies)
print "Percentage of Romance Movies Paired with Drama:", drama_romance_p
erc
print "Percentage of Romance Movies Paired with Horror:", horror_romance
_perc
```

```
We test on this many movies classified as Romance: 380
Percentage of Romance Movies Paired with Drama: 0.652631578947
Percentage of Romance Movies Paired with Horror: 0.0157894736842
```

```
In [50]: ## initialize X and Y for plotting
         Y = np.array([drama_romance_perc, horror_romance_perc])
         LABELS = np.array(["Drama", "Horror"])
         X = [1,2]

         plt.figure(figsize=(10, 7))
         plt.bar(X, Y,  color="blue")
         plt.xticks(X, LABELS)
         plt.title("Percentage of Films in Romance Genre In Other Genre", fontsiz
         e=14)
         plt.xlabel("Genre paired with Romance Films")
         plt.ylabel("Percentage of Genre Associated with Romance Genre")

Out[50]: <matplotlib.text.Text at 0x10cb36c10>
```



Explanation of Romance and Other Dramas

The takeaway from the above bar graph is that more romance fils are paired with drama than they are by horror. (BY A LOT.) However, let's explain how we received our results.

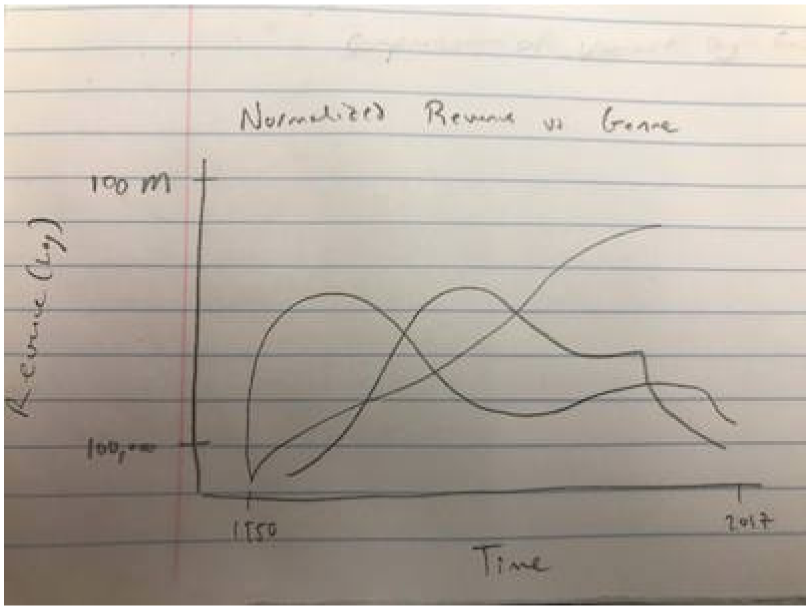
We decided to compare the percentage of romance films that were ALSO associated with other genres. To do this, we selected a random subset of our romance films. We used roughly 380 romance films randomly selected from all films with the genre "romance" tag. Of these 380 films, we then looked at the percentage of these films that also shared the genre tag "horror" or "drama" or both. We divided by the total number of films with the horror/drama tag by all romance films used. This gave us are above percentages.

Sketches and Future EDA

Here are sketches of other visualizations we plan to complete.

```
In [52]: Image(filename='sketches/sketch1.jpg')
```

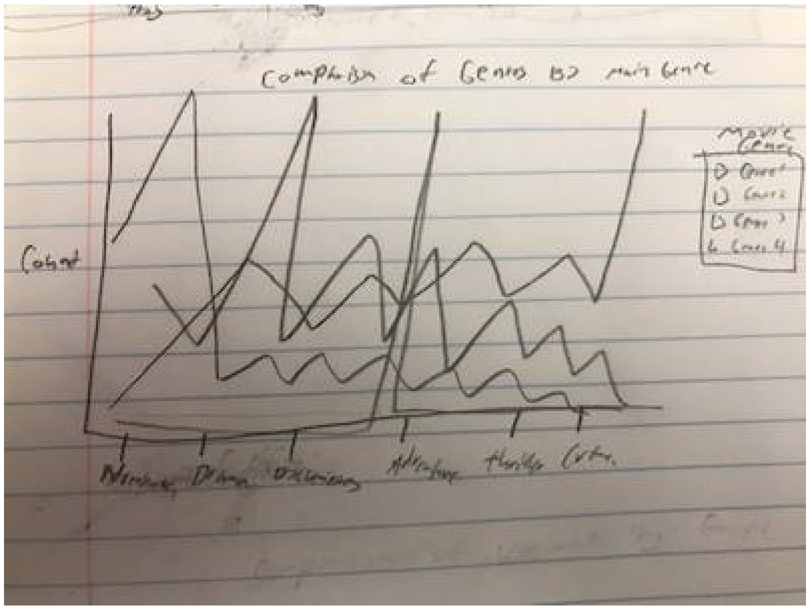
Out[52]:



This is a normalized revenue graph, displaying the amount of revenue that genres have made over time.

```
In [59]: Image(filename='sketches/sketch2.jpg')
```

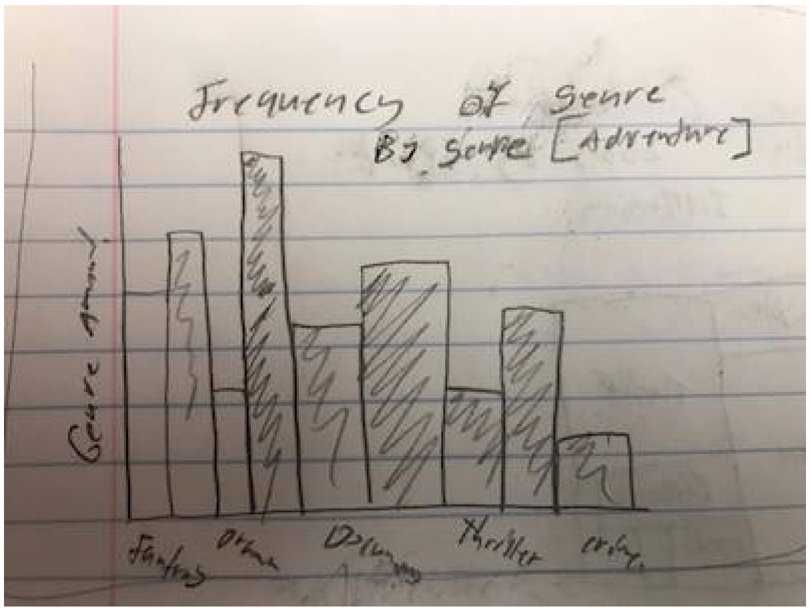
Out[59]:



This graph is a color-coded line chart. This shows the distribution of secondary genres by main genres in comparison to each other genre.

```
In [58]: Image(filename='sketches/sketch3.jpg')
```

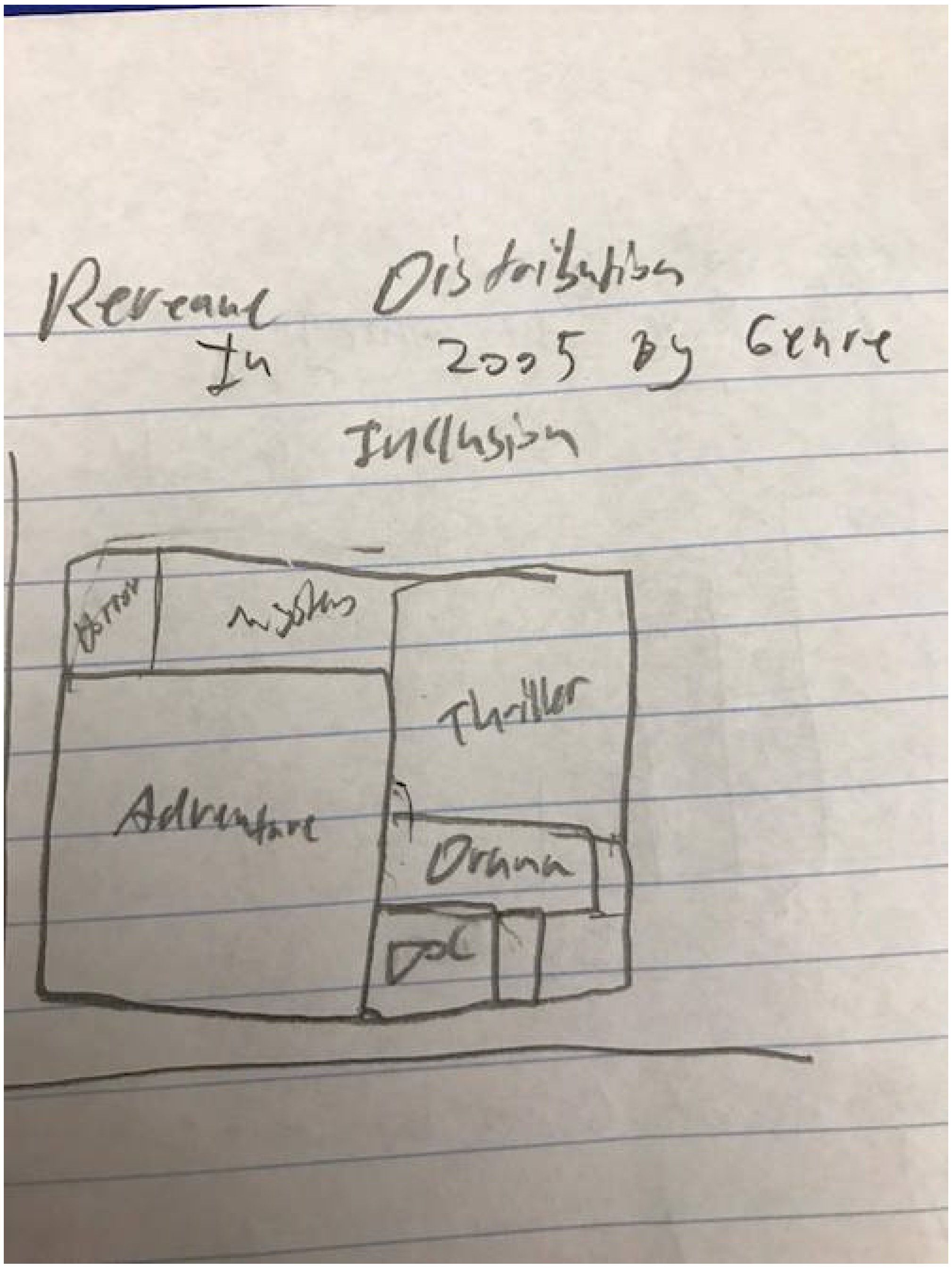
Out[58]:



This graph is a bar graph that shows the frequency of other genres within a defined genre. For example, here we have the genre of “Adventure” along with all the subcategories that also exist within it.

In [57]:
```
Image(filename='sketches/sketch4.jpg')
```

Out[57]:



Finally, we have a treemap to show the relative amount of revenue by genre for a subset year.

#7: A list of questions you could answer with this and related data. Get creative here!

- Which movies are the most fluid across genres?
- Which genres are most constrictive (i.e. does having genre X mean you're less likely to be paired with another genre?)
- What pairing of movies genres are most likely to go together?
- Do certain genre pairings result in higher amounts of revenue (e.g. "action romance" doing better than "action")?
- Are there statistically significant differences in movie length by genre? For example, are comedies consistently lower?
- Are there statistically significant differences in movie rating by movie length? Do people tend to like shorter movies more or longer movies more? How does this difference interact with genre?
- What is the highest revenue per minute by genre (we could answer this by dividing average revenue by average movie length)?
- Are adult films frequently labelled other genres? For example, are there such things as "adult fantasy" films and "adult comedy" films?
- How do these english genre distributions compare versus a similar analysis done on a different language's database? Are french movies, for instance, more likely to have different genre pairings than english movies?
- How does genre density differ as a function of time? Are some genres more popular now vs. long ago? What about pairing of genres?
- Does having an unoriginal movie title (e.g. "300") help revenue or hurt?
- How has (normalized) revenue changed over time as a function of genre? What about popularity?